Fault-Tolerant Computing in Wireless Ad Hoc Networks

Gregory Chockler, IBM Research

chockler@il.ibm.com

http://theory.csail.mit.edu/~grishac

http://www.research.ibm.com/people/c/chockler

Notes

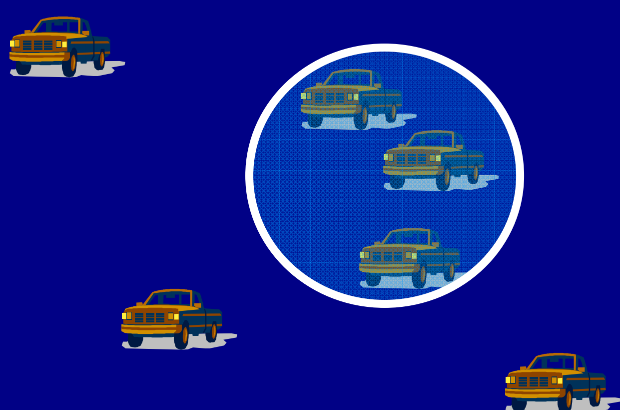
- Names in brackets, as in [Xyz00], refer to a document in the list of references
- There might be some slight differences with the slides on CD
  - The final version will be available at http://www.research.ibm.com/people/c/chockler

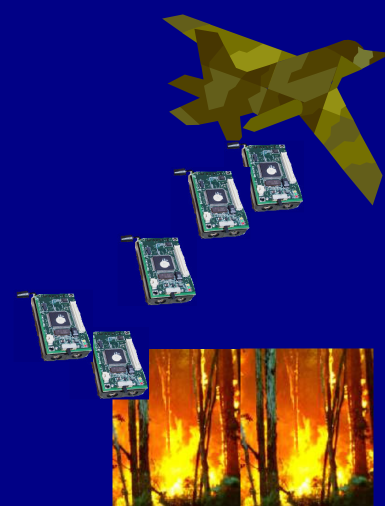
Disclaimer

- Fault tolerance in wireless networks is a new rapidly evolving research area
  - This tutorial is by no means exhaustive
  - Many interesting topics not covered in the tutorial due to lack of time
- The material selection reflects to a large extent my personal taste and experience
  - Most results are theoretical
  - Only a small portion was implemented

Wireless Ad Hoc Networks

- Radio-equipped devices
- Spontaneous connectivity
  - – No networking infrastructure

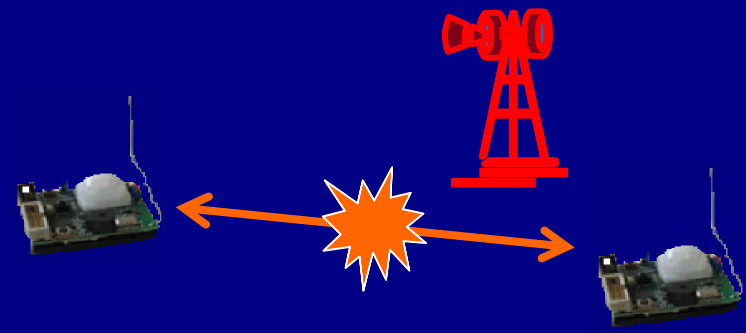


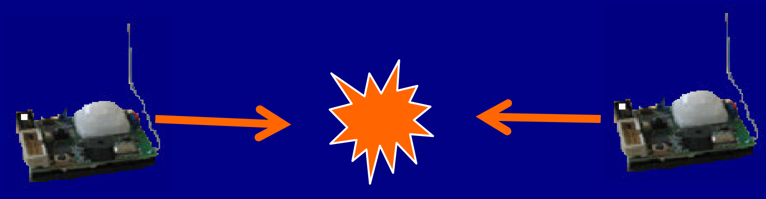


Failures in Wireless Networks

- Device failures
  - Limited battery life
  - Small size and fragility
  - Software bugs

- Message loss
  - Collision, interference, hidden terminals



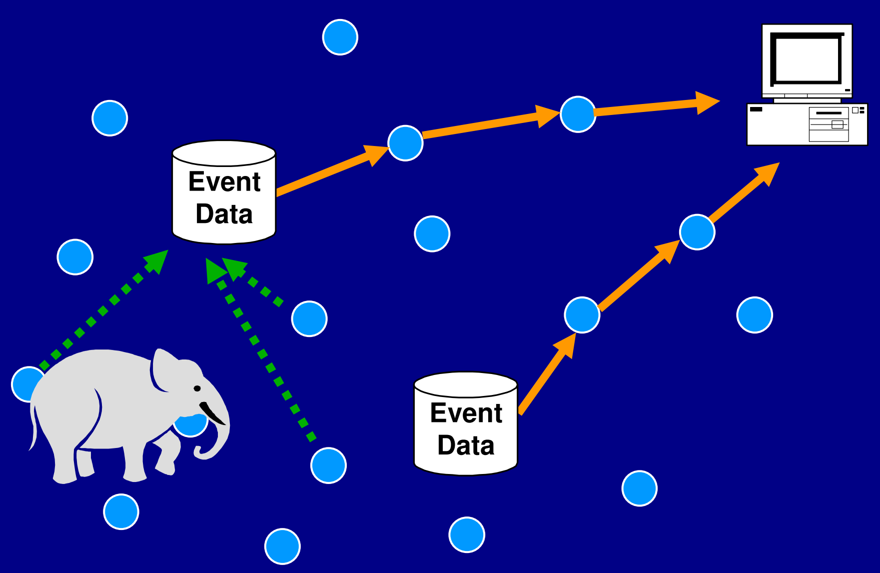


More Limitations

- No unique IDs
- Unknown topology
- Inaccurate knowledge of location
- Drifting clocks
- Mobility

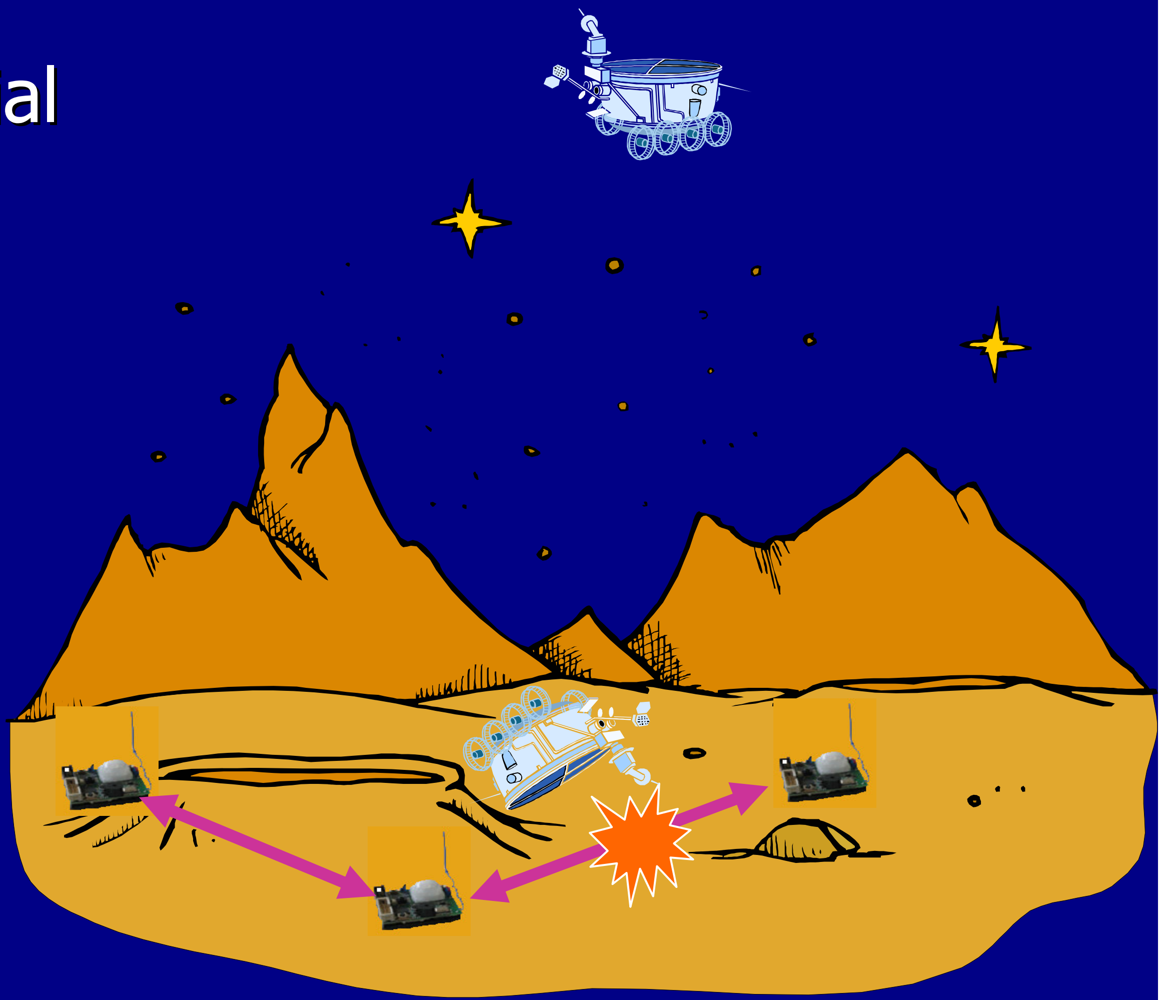
Robustness to Failures

- Many applications could live with best-effort guarantees
  - E.g., data collection, aggregation, querying, monitoring, etc…



Robustness to Failures

- Well-defined guarantees are crucial for mission critical tasks
  - Emergency response
  - Coordinated lander guidance
  - Rover navigation
  - Autonomic flight and traffic control
  - Coordinated UAVs



Supporting Robustness

- Develop a suite of services (*middleware*) to mask failures
  - Well-defined guarantees
  - Comprehensive
  - Powerful
  - Realistic
  - Simple to understand and use
  - Modular

Fault-Tolerance Middleware

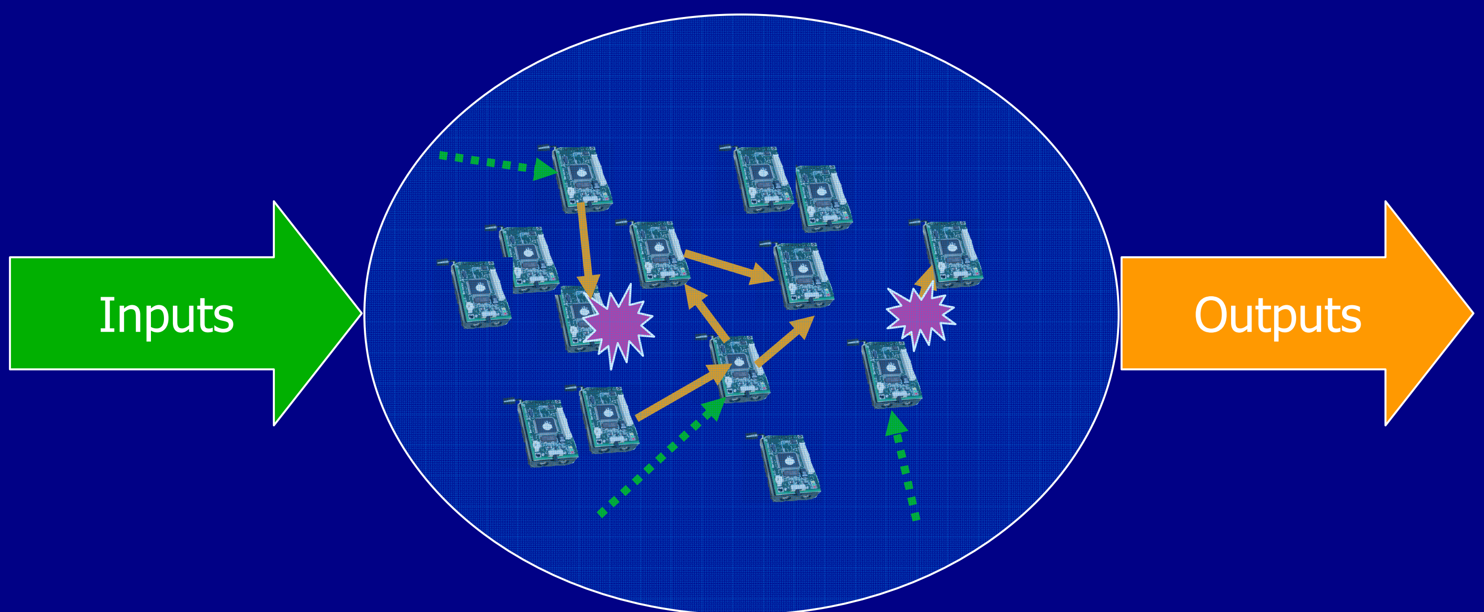
- Local infrastructure
  - Local agreement
  - State machines
  - Virtual nodes
- Global infrastructure
  - Round synchronization
  - Broadcast
  - Quorums
- Applications

Fault-Tolerance Middleware

- Local infrastructure
  - State machines and virtual nodes
  - Local agreement
- Global infrastructure
  - Round synchronization
  - Broadcast
  - Quorums

Local Infrastructure

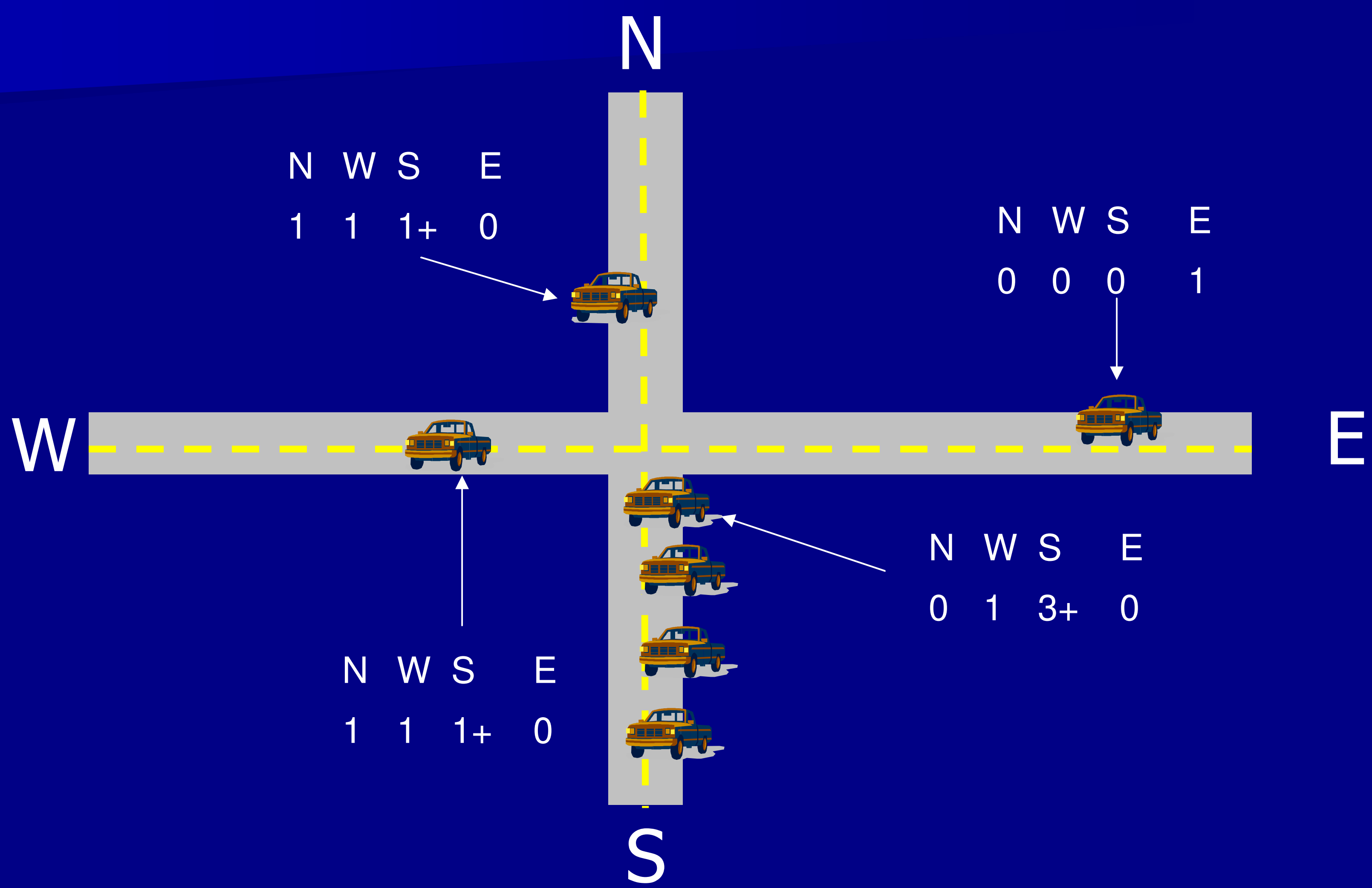
- Objective: create a single reliable entity from a collection of closely coupled, unreliable devices



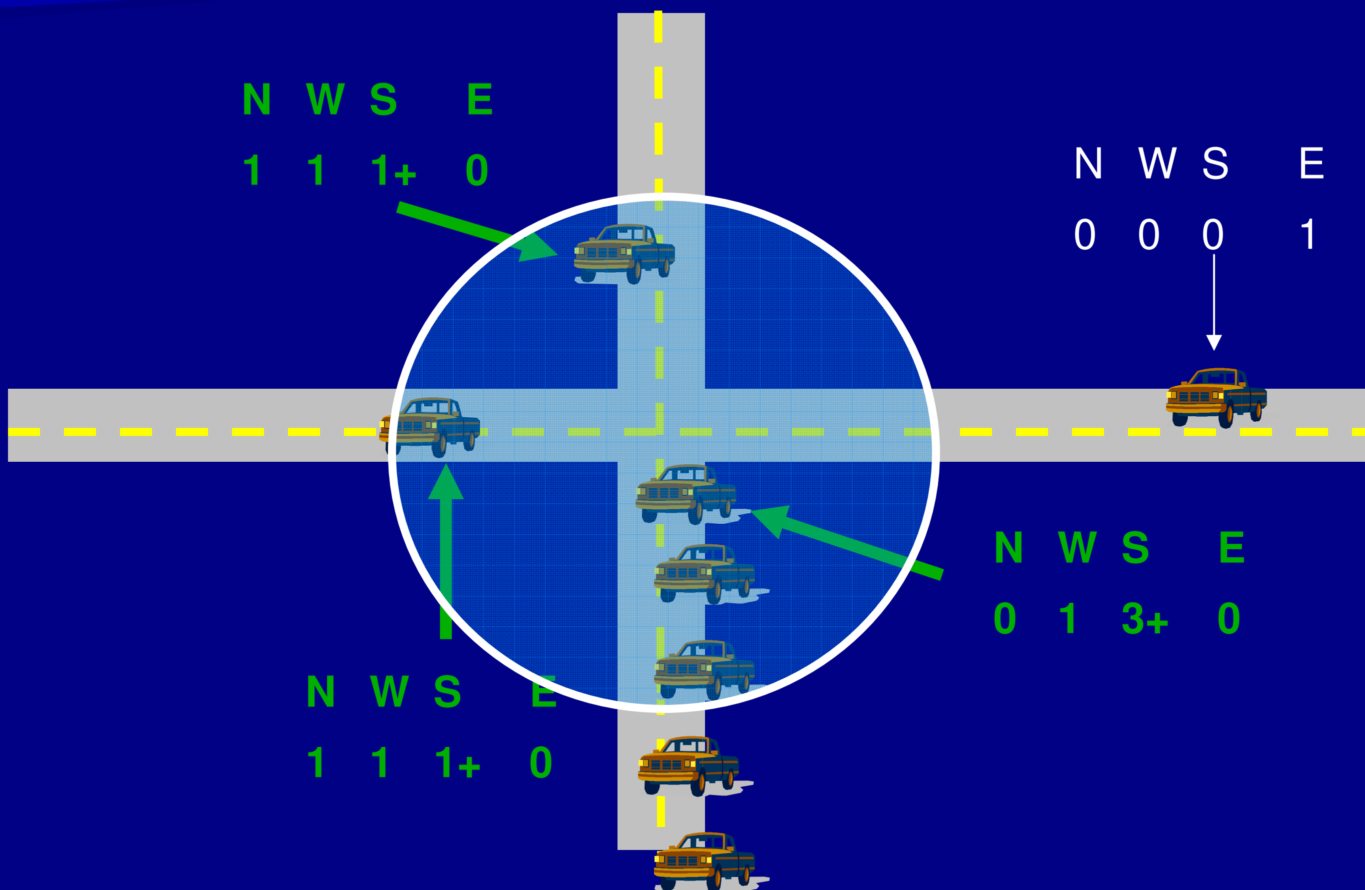
Local State Machine

- All the nodes within the communication range of one another emulate a persistent state machine
  - Inputs: environment stimuli
  - Outputs: consistent actions based on the state machine transition function

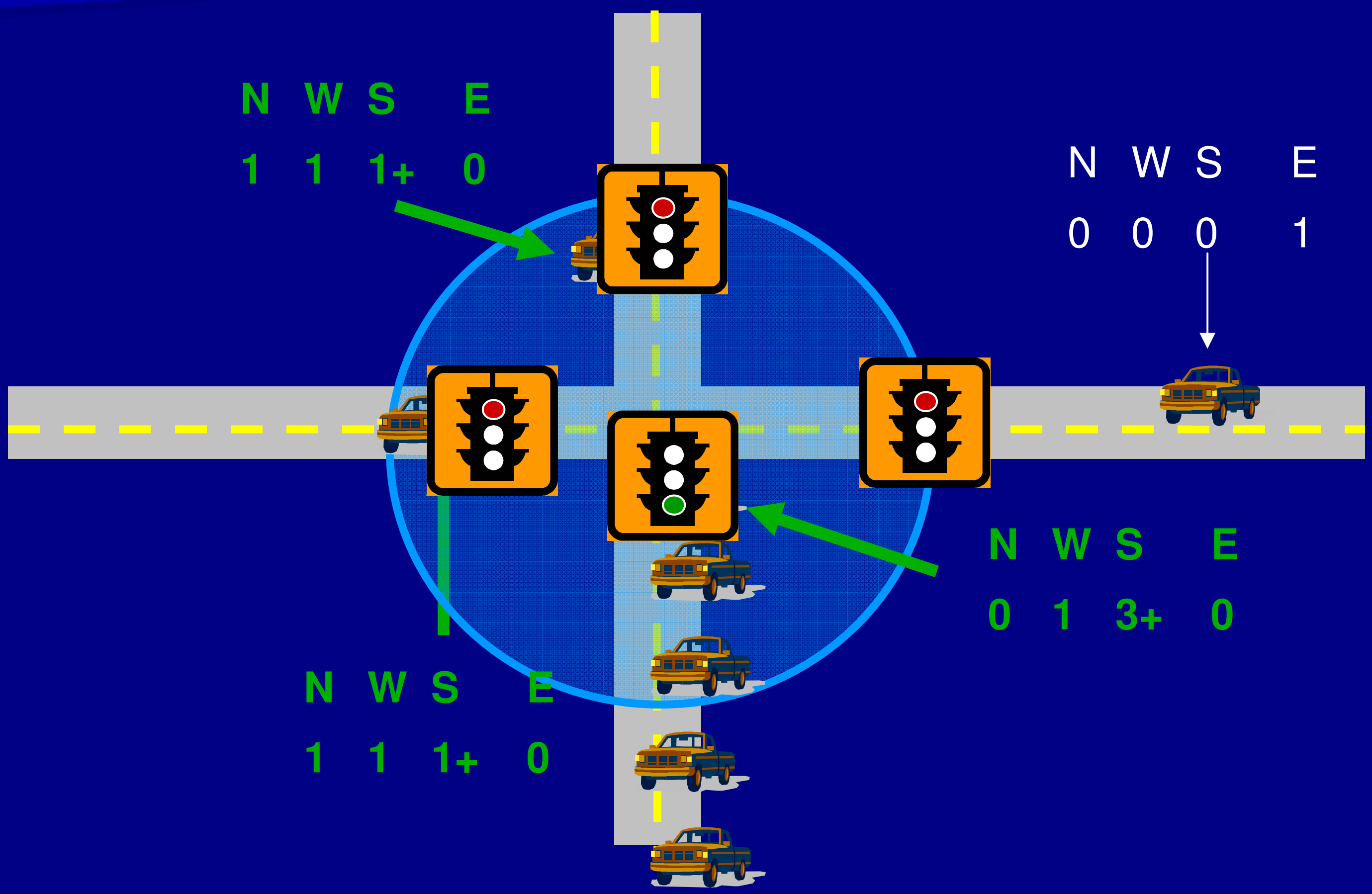
Example: Virtual Traffic Lights



Example: Virtual Traffic Lights



Example: Virtual Traffic Lights



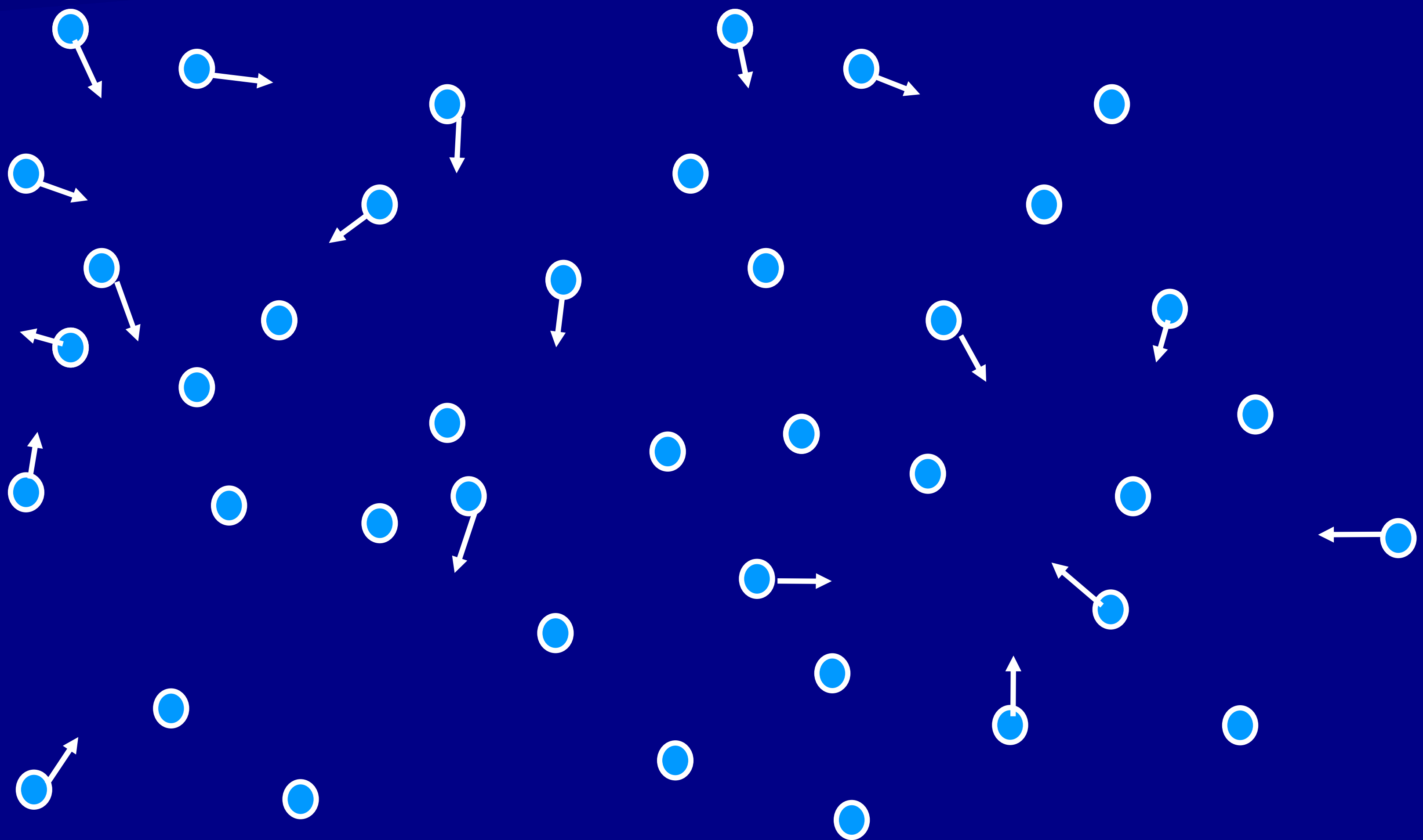
Fault-Tolerance Middleware

- Local infrastructure
  - State machines and virtual nodes
  - Local agreement
- Global infrastructure
  - Round synchronization
  - Broadcast
  - Quorums

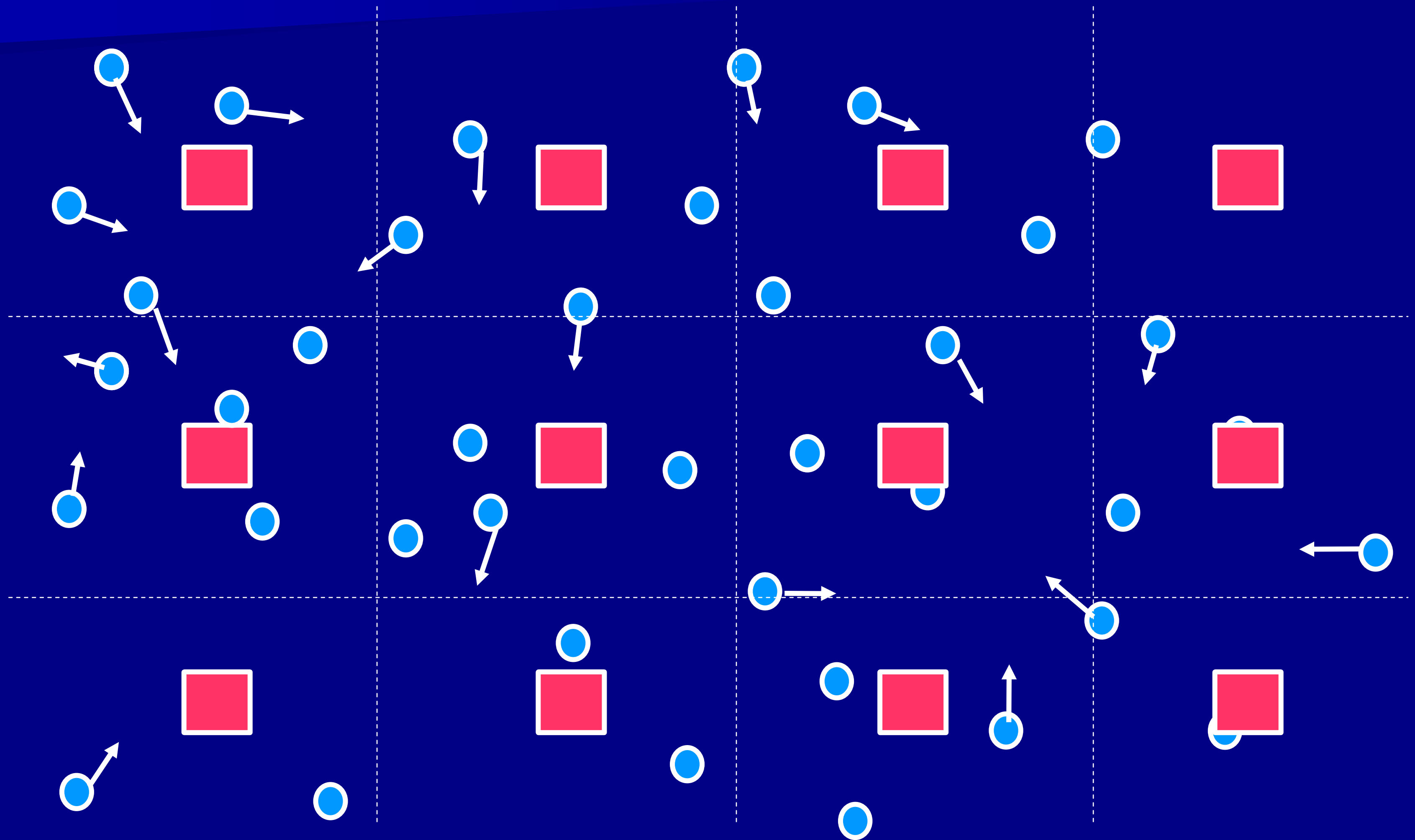
Virtual Nodes

- Emulate a persistent virtual node in each locality populated by physical nodes
- Input: message←recv()
- Output: send(vn,message)
- The applications are deployed at virtual nodes as though they are real nodes
  - Programmers do not need to care about "peculiarities" of wireless networks

Virtual Nodes



Virtual Nodes



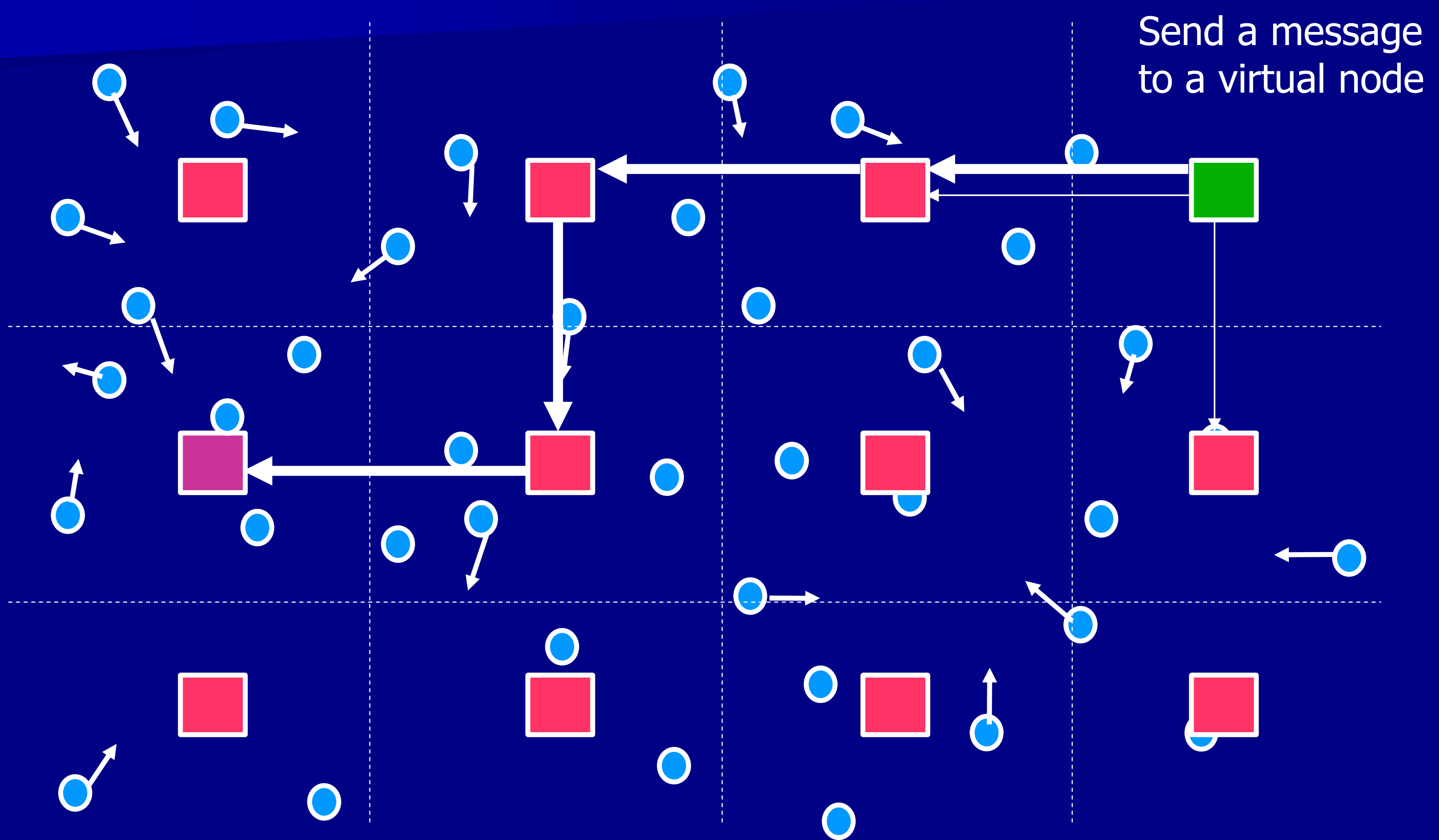
Applications

- Location management
- Routing
- Tracking
- Motion coordination
- Traffic management
- Traffic coordination
- *Many others*

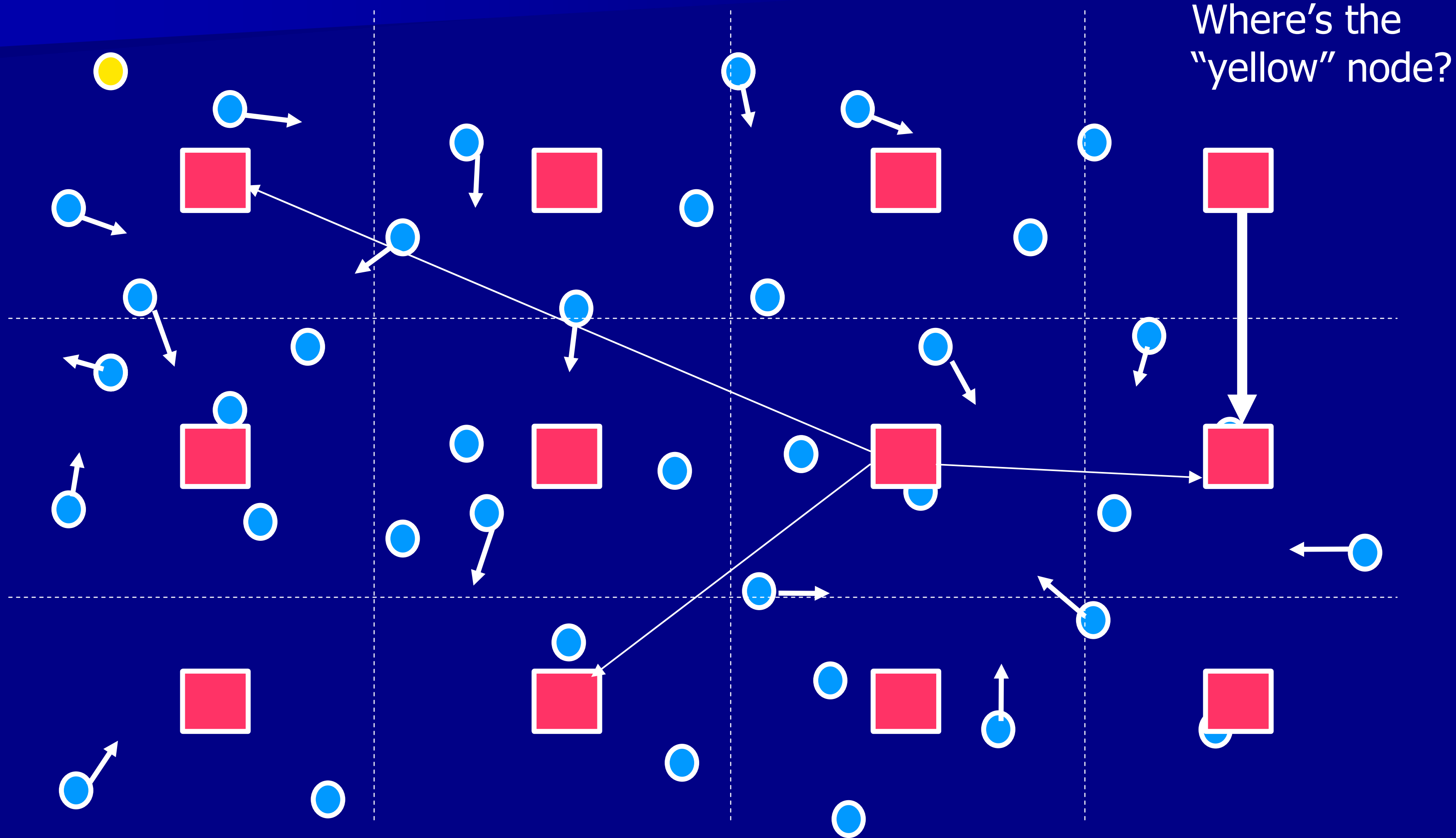
GeoCast Routing

- Location-based routing
  - Requires knowing precise location
- Use broadcast to disseminate messages to the neighbors
- The neighbor closest to the destination will forward the message in the same manner

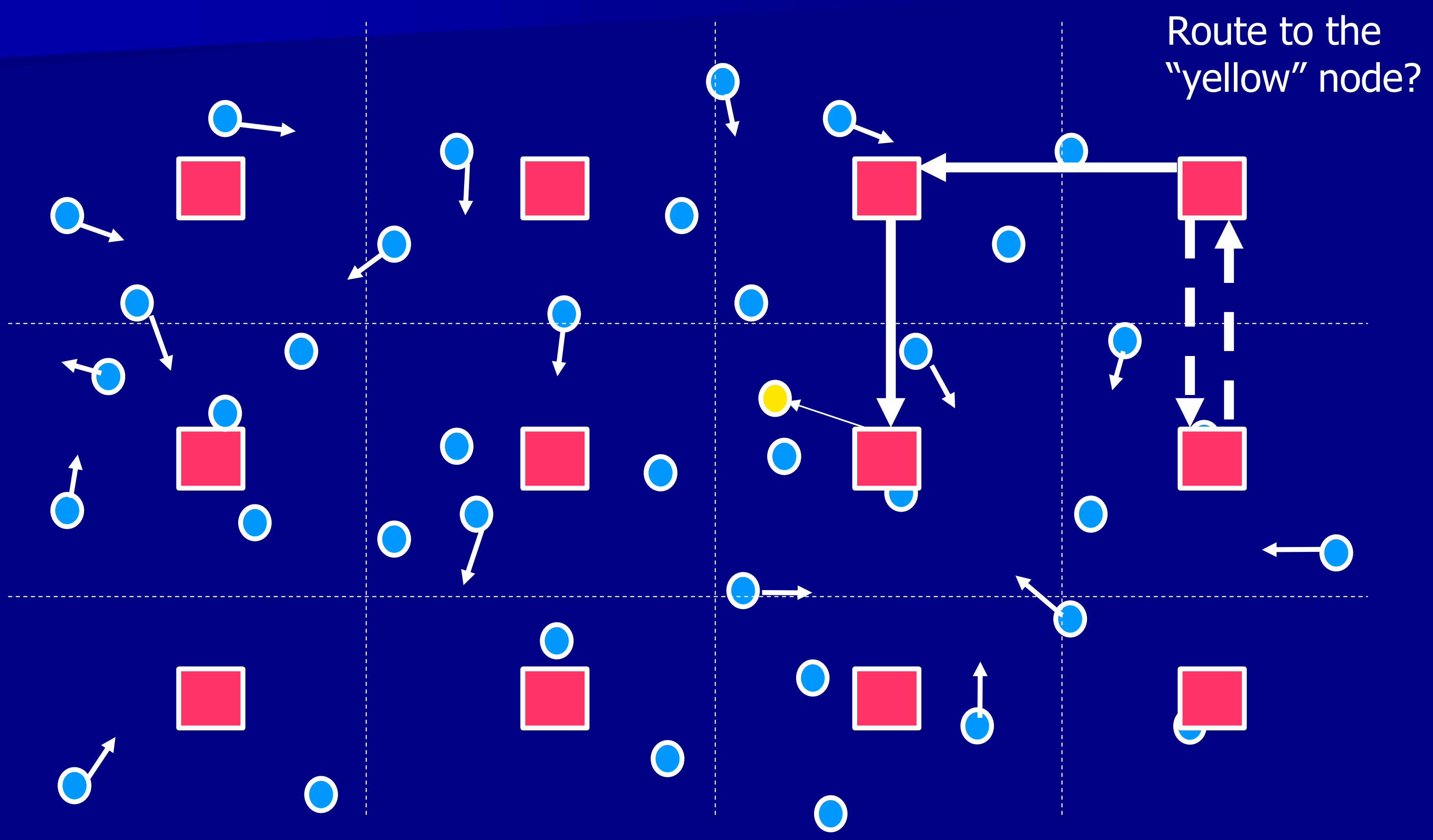
GeoCast Routing



Home Location



Point-to-Point Routing



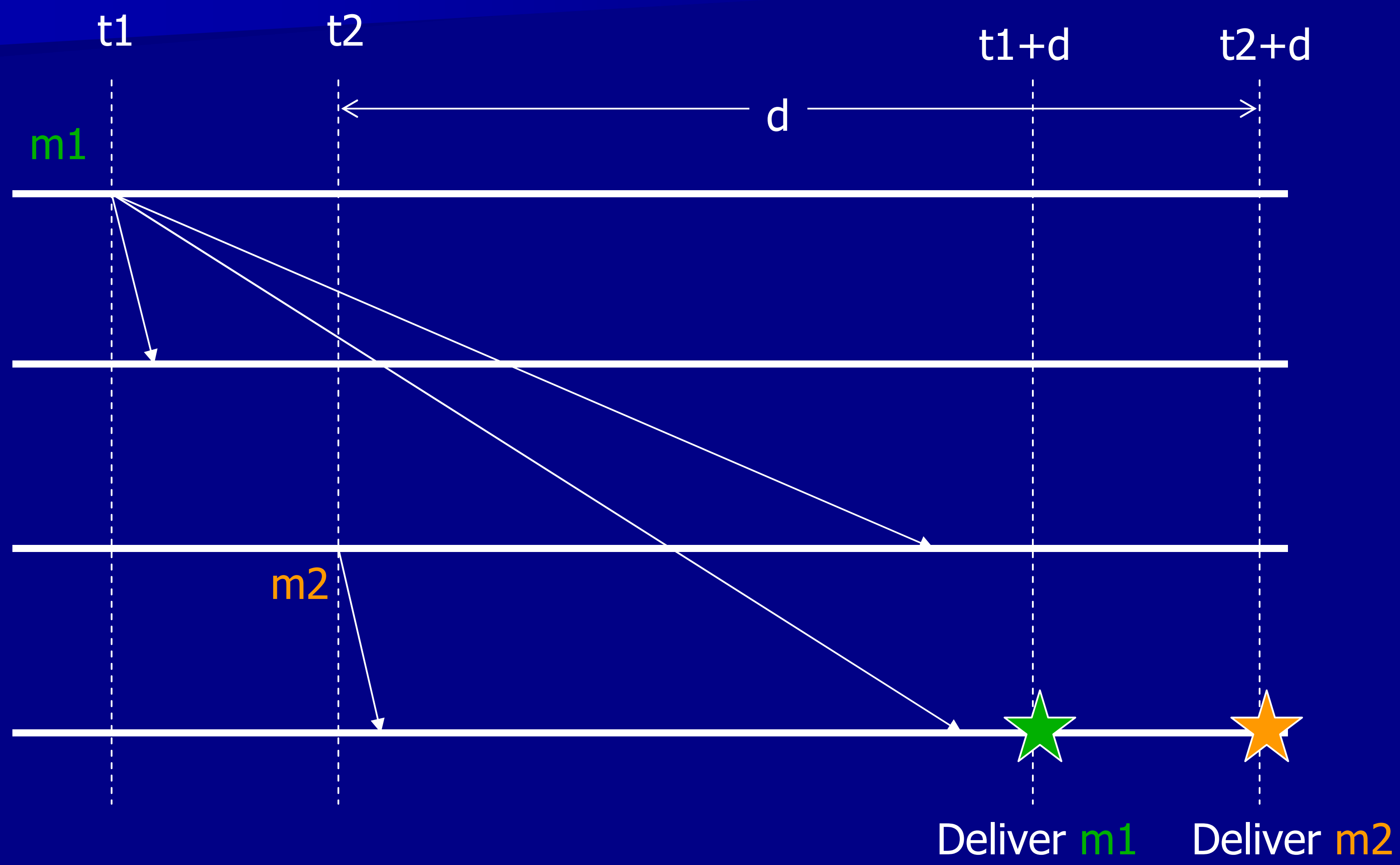
Implementing a Virtual Node

- State-machine replication [8]
- Replicate the virtual node state at the physical nodes within the region
- Broadcast each received message within the region using a total-order broadcast
- Total-order broadcast (TO-Broadacst): messages are delivered at the same order at all nodes

Implementing a Virtual Node [5,6,7]

- Tight clock synch within a region
- Location/time awareness (GPS)
- Known bound on message delay ***d***
- One node is a leader

Implementing TO-Broadcast [5,7,8]



Implementing TO-Broadcast [5,7,8]

- Affix message M a unique timestamp:
  - TS := clock()
- Locally broadcast (M,TS,Sender)
- For each received (m,ts,sender) such that clock()=ts+d, move (m,ts,sender) to out-buffer
- Deliver messages in the out-buffer in the timestamp order
  - Break ties using the sender id

Implementing a Virtual Node

- A physical node receives m:
  - TO-Broadcast(m) within the region
- Upon delivery of a TO message m:
  - Perform the transition triggered by m
  - If a new message m' should be sent:
    - If (leader?) then send m' to the destination VN using Geocast

Towards a More Realistic Model

- The VN implementation relies on
  - Reliable local broadcast
  - Known identifiers
  - Known number of nodes
- These assumptions are not always realistic in wireless networks
- How to relax these assumptions in a meaningful way?

The New Model

- Unknown number of nodes, no unique ids
- Messages can be lost due to **collision** and other anomalies
- Round-based computation: Each process in each round:
    1. Broadcasts a message
    2. Receives messages
    3. Performs computation

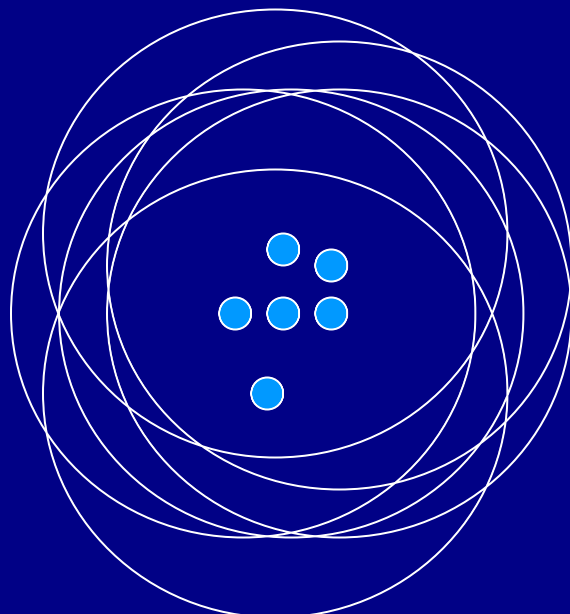
  - Messages broadcast in r are received in r

Round-Based Computation

- Computation proceeds in rounds
- In each round r, each process P:
  - Sends a message
  - Receives messages
  - Performs computation
- Might seem unrealistic, but can be easily emulated with
  - Bounded drift clocks and message delay
- We'll see an implementation later

Local Agreement

- Can we still implement a VN?
  - What is necessary for that?
- Single-hop environment
- We investigate this using a *local agreement* problem ➔



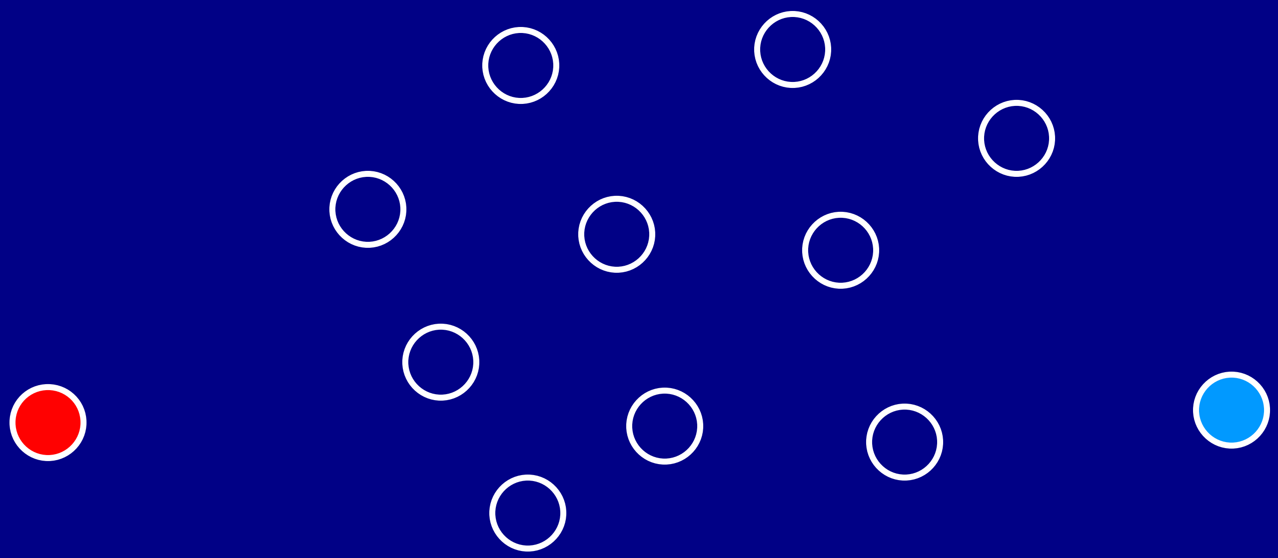
Fault-Tolerance Middleware

- Local infrastructure
  - State machines and virtual nodes
  - Local agreement
- Global infrastructure
  - Round synchronization
  - Broadcast
  - Quorums

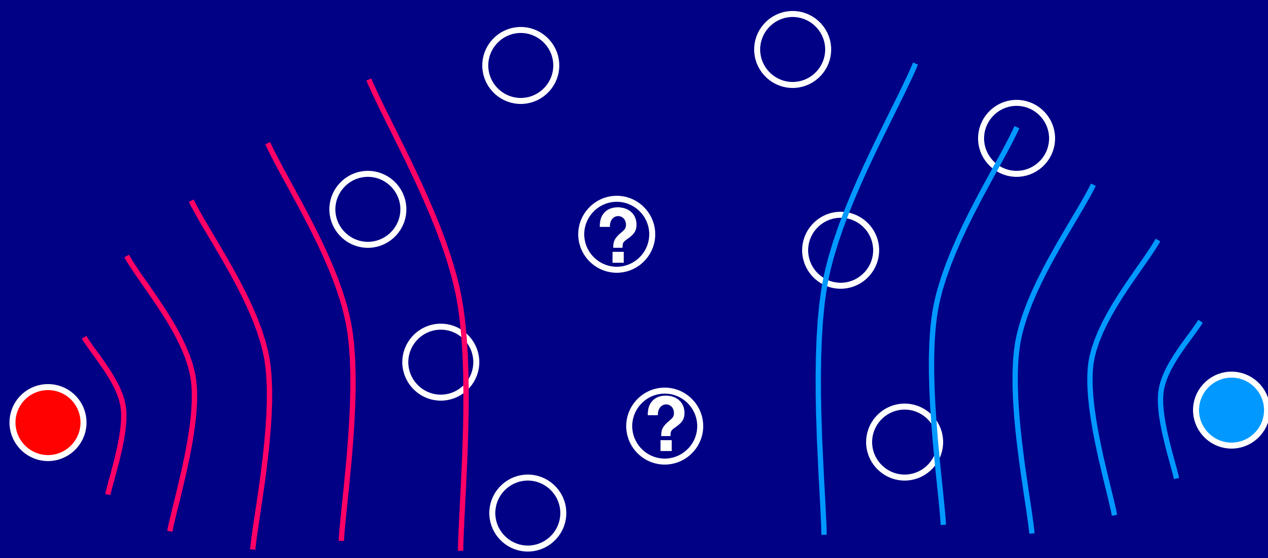
Local Agreement (Consensus) [2,3]

- Start with possibly different input values
- Agreement:
  - Different inputs → (eventually) the same output at each participating node
- Validity:
  - Each output is the input of some process

Characterizing Collision



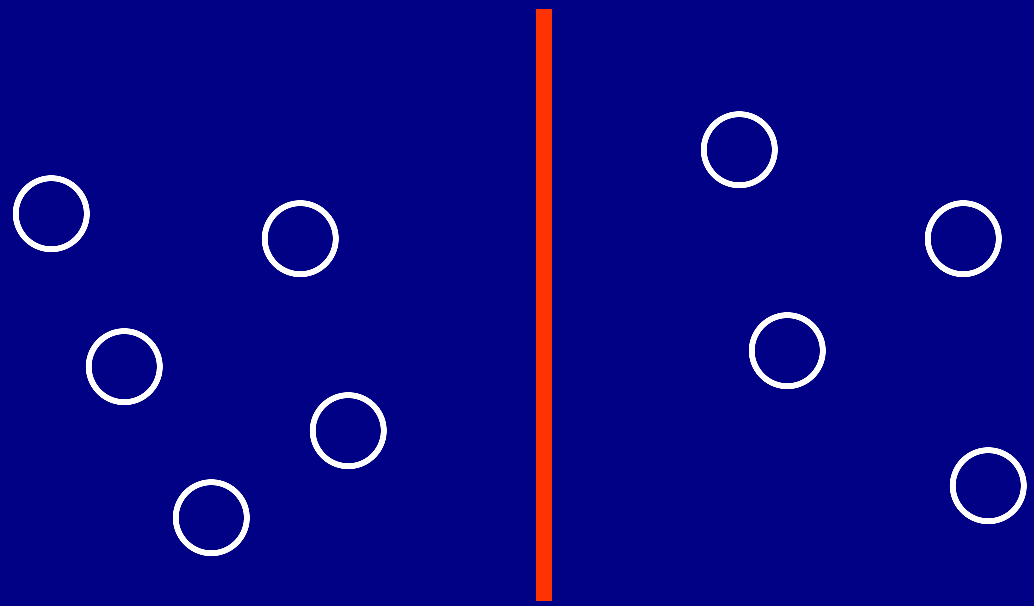
Characterizing Collision



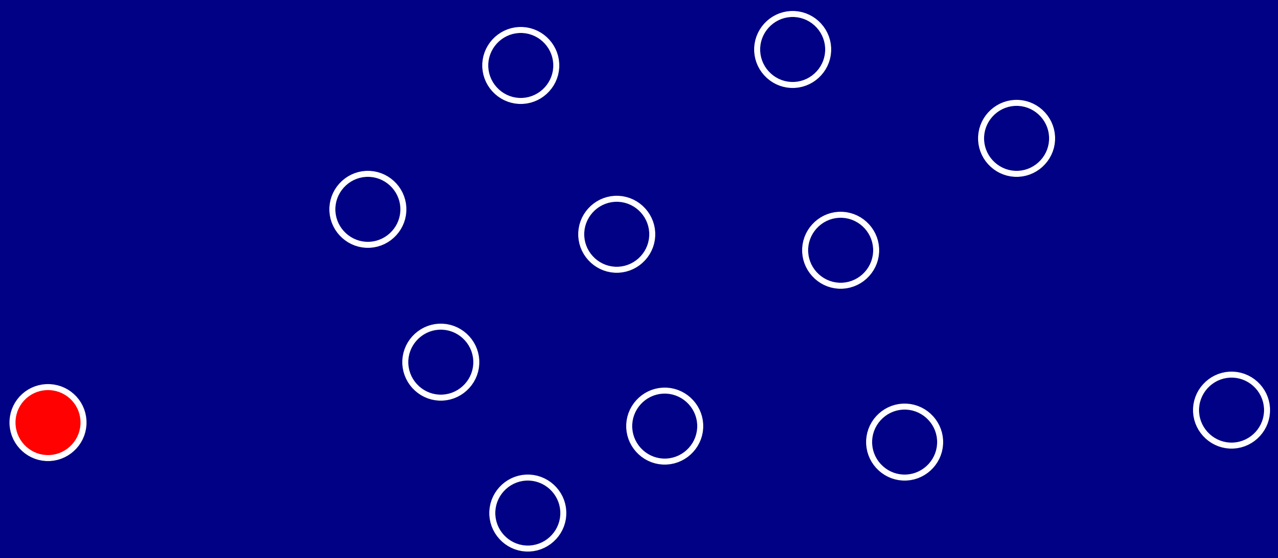
Non-Uniform Collisions: Any node can loose any message in any round

Unfortunately...

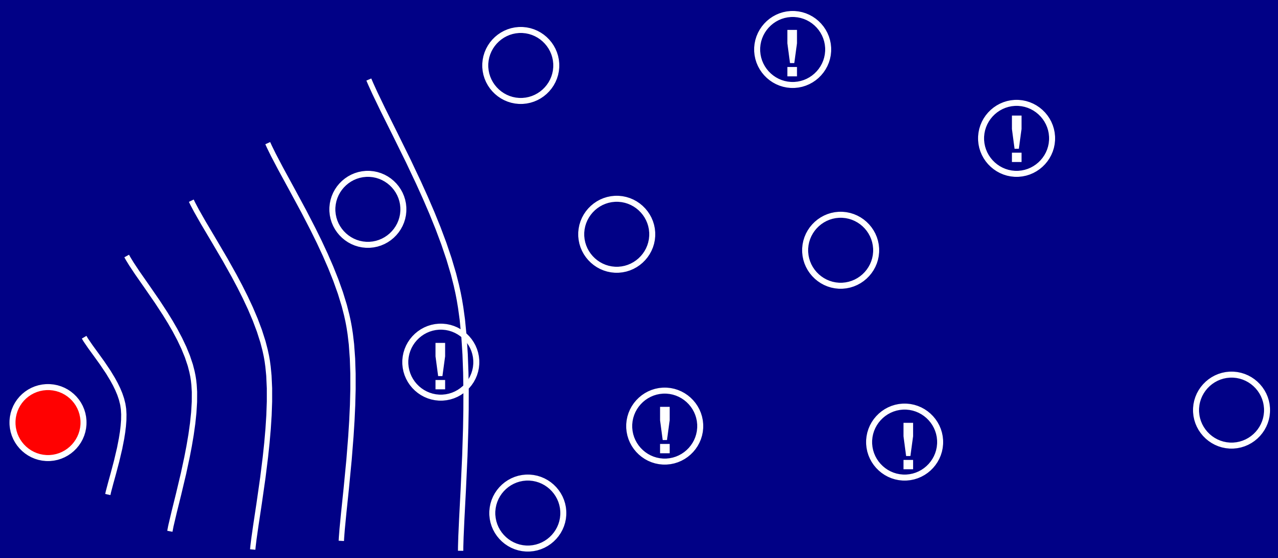
Agreement is **impossible** with non-uniform collisions.



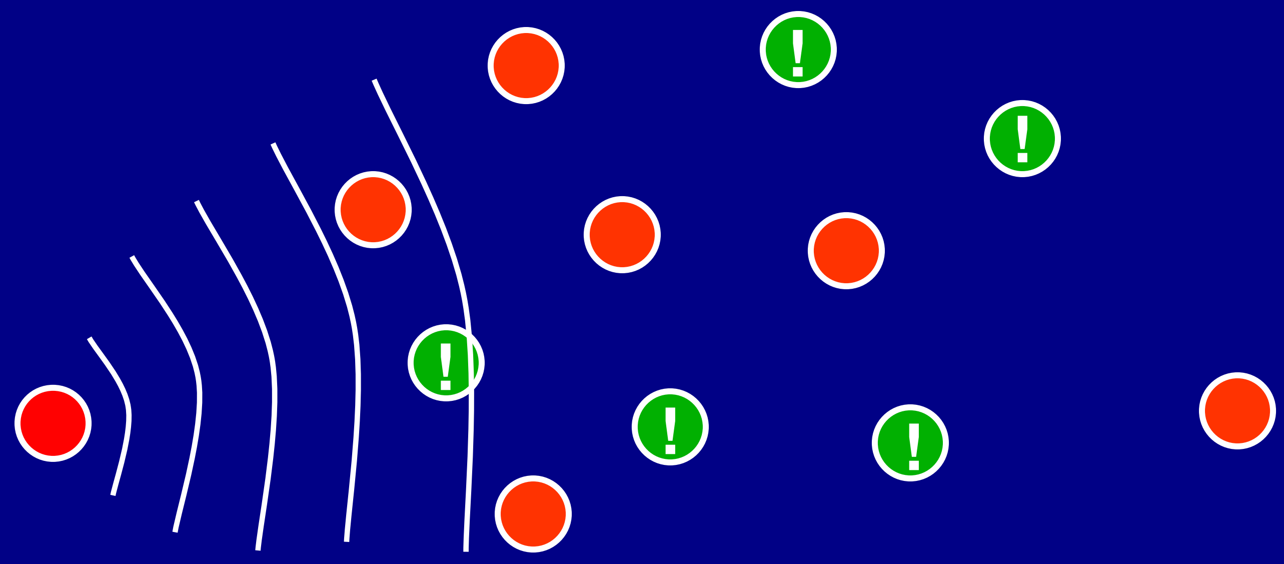
Solution: Collision Detection



Collision Detection



Collision Detection



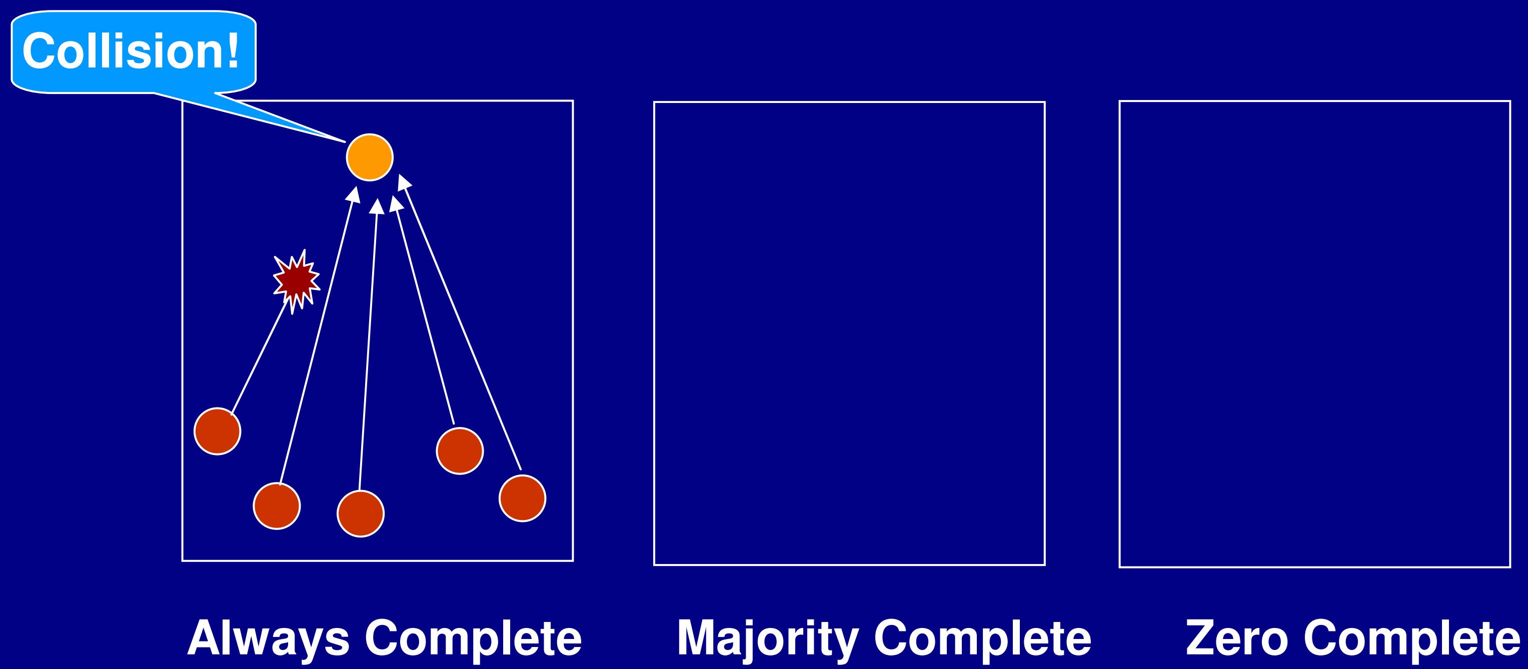
Receiver-based collision detection

Collision Detectors

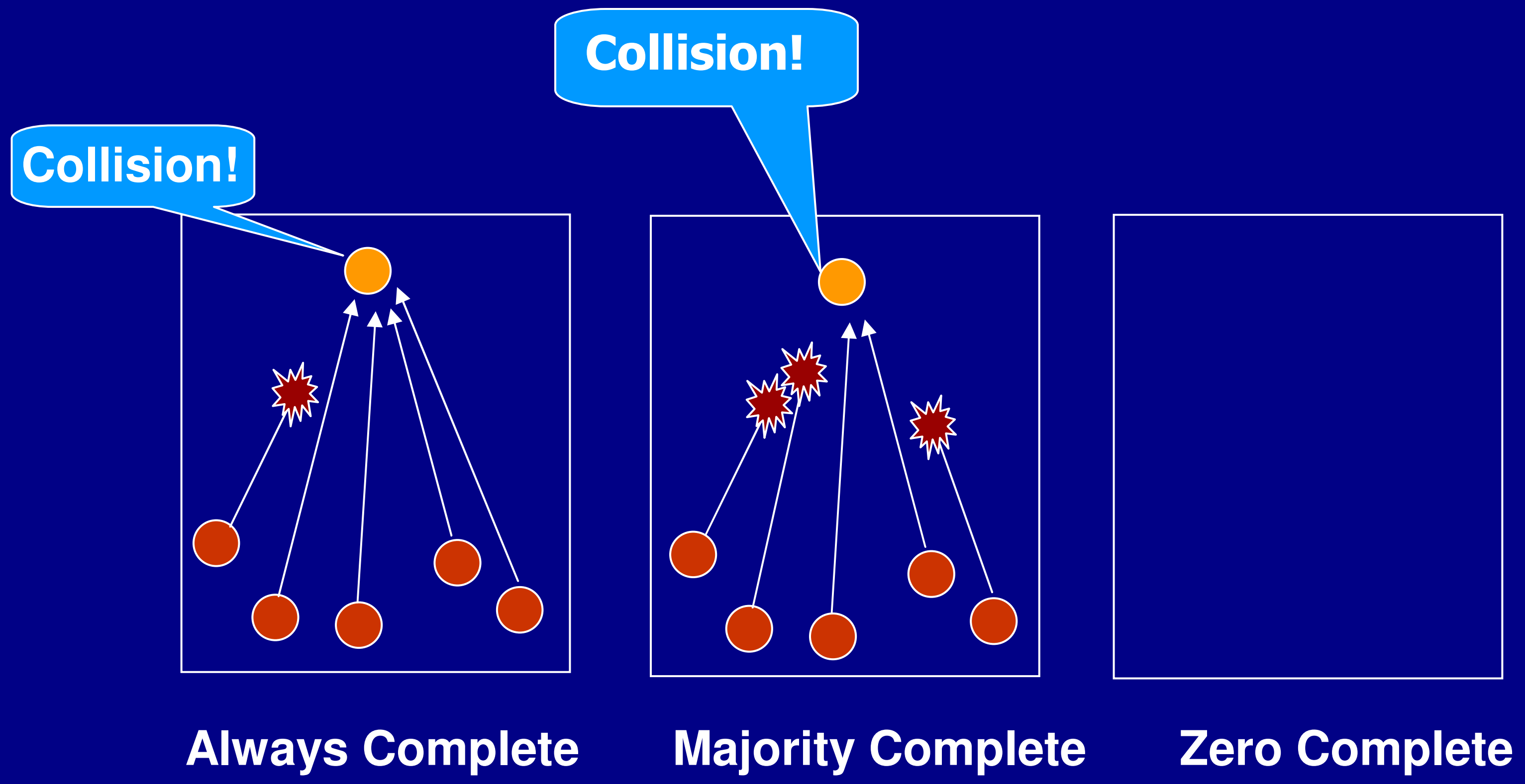
- Properties:
  - Completeness: If P loses a message, …
  - Accuracy: If P loses no messages, …

- Question: Find a CD which is both realistic and powerful enough to solve agreement efficiently

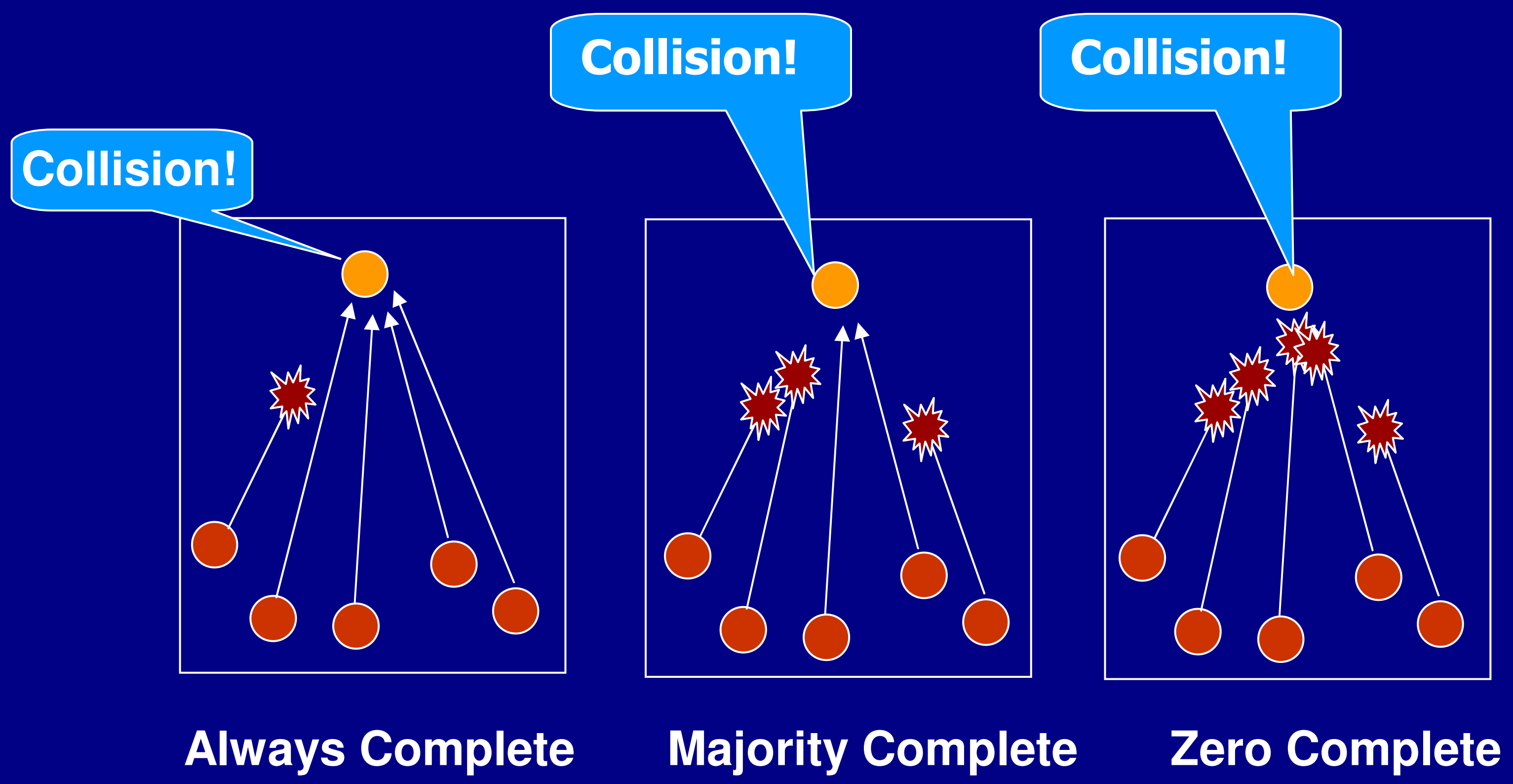
Completeness Degrees



Completeness Degrees



Completeness Degrees



Collision Detector Classes

	(Always) Accurate: *A*	Eventually Accurate: ◊*A*
(Always) Complete: *C*	*AC*	◊*AC*
Majority Complete: Maj-*C*	Maj-*AC*	Maj- ◊*AC*
0-Complete: 0-*C*	0-*AC*	0- ◊*AC*

Agreement is impossible with ◊C

Collision Detector Classes

	(Always)	Eventually ...te: ◊*A*
(Always) Comple... *C*		...*AC*
Majority Complete: *Maj-C*	*Maj-AC*	*Maj-* ◊*AC*
0-Complete: 0-*C*	0-*AC*	0- ◊*AC*

If ≥½ messages are lost, then report collision.

Agreement is impossible with ◊C

Collision Detector Classes

	(Always)	Eventually te: ◊*A*
(Always) Comple *C*		*AC*
Majority Complete: *Maj-C*		
0-Complete: *0-C*	0-*AC*	0- ◊*AC*

If ≥½ messages are lost, then report collision.

If all messages are lost, then report a collision.

Agreement is impossible with ◊C

Agreement with CD

	No Collision Freedom
$\mathcal{AC}$	$\Theta(\log \lvert V \rvert)$
maj-$\mathcal{AC}$	$\Theta(\log \lvert V \rvert)$
0-$\mathcal{AC}$	$\Theta(\log \lvert V \rvert)$
$\diamond\mathcal{AC}$	Impossible
maj-$\diamond\mathcal{AC}$	Impossible
0-$\diamond\mathcal{AC}$	Impossible

V is the value domain

Agreement with CD

(Always) Accurate

	No Collision Freedom
$\mathcal{AC}$	$\Theta(\log\|V\|)$
maj-$\mathcal{AC}$	$\Theta(\log\|V\|)$
0-$\mathcal{AC}$	$\Theta(\log\|V\|)$
$\diamond\mathcal{AC}$	Impossible
maj-$\diamond\mathcal{AC}$	Impossible
0-$\diamond\mathcal{AC}$	Impossible

V is the value domain

Agreement with CD

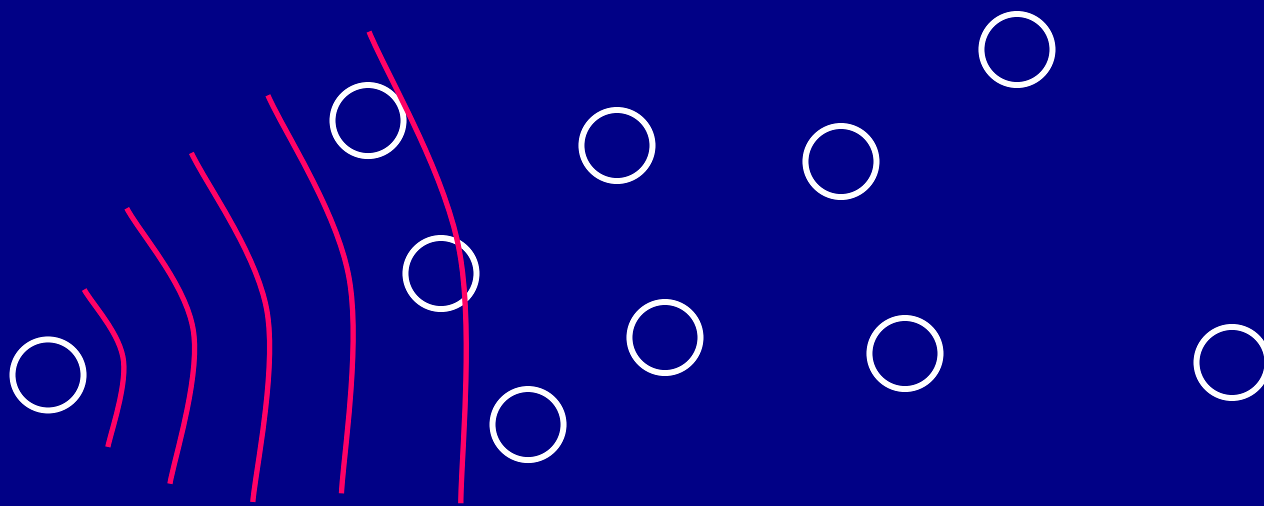
	No Collision Freedom
$\mathcal{AC}$	$\Theta(\log \lvert V \rvert)$
maj-$\mathcal{AC}$	$\Theta(\log \lvert V \rvert)$
0-$\mathcal{AC}$	$\Theta(\log \lvert V \rvert)$
$\Diamond\mathcal{AC}$	Impossible
maj-$\Diamond\mathcal{AC}$	Impossible
0-$\Diamond\mathcal{AC}$	Impossible

Eventually Accurate

V is the value domain

Eventual Collision Freedom

- Eventually, if only 1 node broadcasts…



Eventual Collision Freedom

- Eventually, if only 1 node broadcasts, then no collision occurs
- Use a contention manager
  - Outputs "active/passive" at each node
  - Implementation: randomized backoff, e.g.

Eventual Collision Freedom

- Eventually, if only 1 node broadcasts*, then no collision occurs
- Use a contention manager
  - Outputs "active/passive" at each node
  - Implementation: randomized backoff, e.g.

* If $\leq b$ nodes broadcast, then no collisions
- b is an unknown MAC layer constant
- b could be as low as 1

Agreement with CD

	No Collision Freedom
$\mathcal{AC}$	$\Theta(\log \lvert V \rvert)$
maj-$\mathcal{AC}$	$\Theta(\log \lvert V \rvert)$
0-$\mathcal{AC}$	$\Theta(\log \lvert V \rvert)$
$\Diamond\mathcal{AC}$	Impossible
maj-$\Diamond\mathcal{AC}$	Impossible
0-$\Diamond\mathcal{AC}$	Impossible

V is the value domain

Agreement with CD

	Eventual Collision Freedom	**No Collision Freedom**
$\mathcal{AC}$	$\Theta(1)$	$\Theta(\log\|V\|)$
maj-$\mathcal{AC}$	$\Theta(1)$	$\Theta(\log\|V\|)$
0-$\mathcal{AC}$	$\Theta(\log\|V\|)$	$\Theta(\log\|V\|)$
$\diamond\mathcal{AC}$	$\Theta(1)$	Impossible
maj-$\diamond\mathcal{AC}$	$\Theta(1)$	Impossible
0-$\diamond\mathcal{AC}$	$\Theta(\log\|V\|)$	Impossible

V is the value domain

Agreement with CD

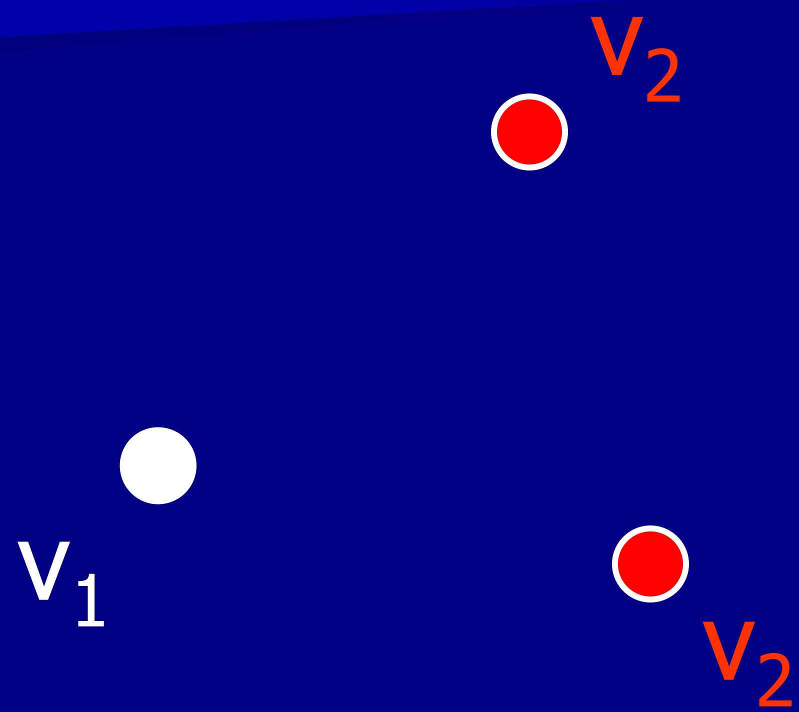
	Eventual Collision Freedom	**No Collision Freedom**
$\mathcal{AC}$	$\Theta(1)$	$\Theta(\log \|V\|)$
maj-$\mathcal{AC}$	$\Theta(1)$	$\Theta(\log \|V\|)$
0-$\mathcal{AC}$	$\Theta(\log \|V\|)$	$\Theta(\log \|V\|)$
$\diamond\mathcal{AC}$	$\Theta(1)$	Impossible
maj-$\diamond\mathcal{AC}$	$\Theta(1)$	Impossible
0-$\diamond\mathcal{AC}$	$\Theta(\log \|V\|)$	Impossible

V is the value domain

Agreement with ◊AC

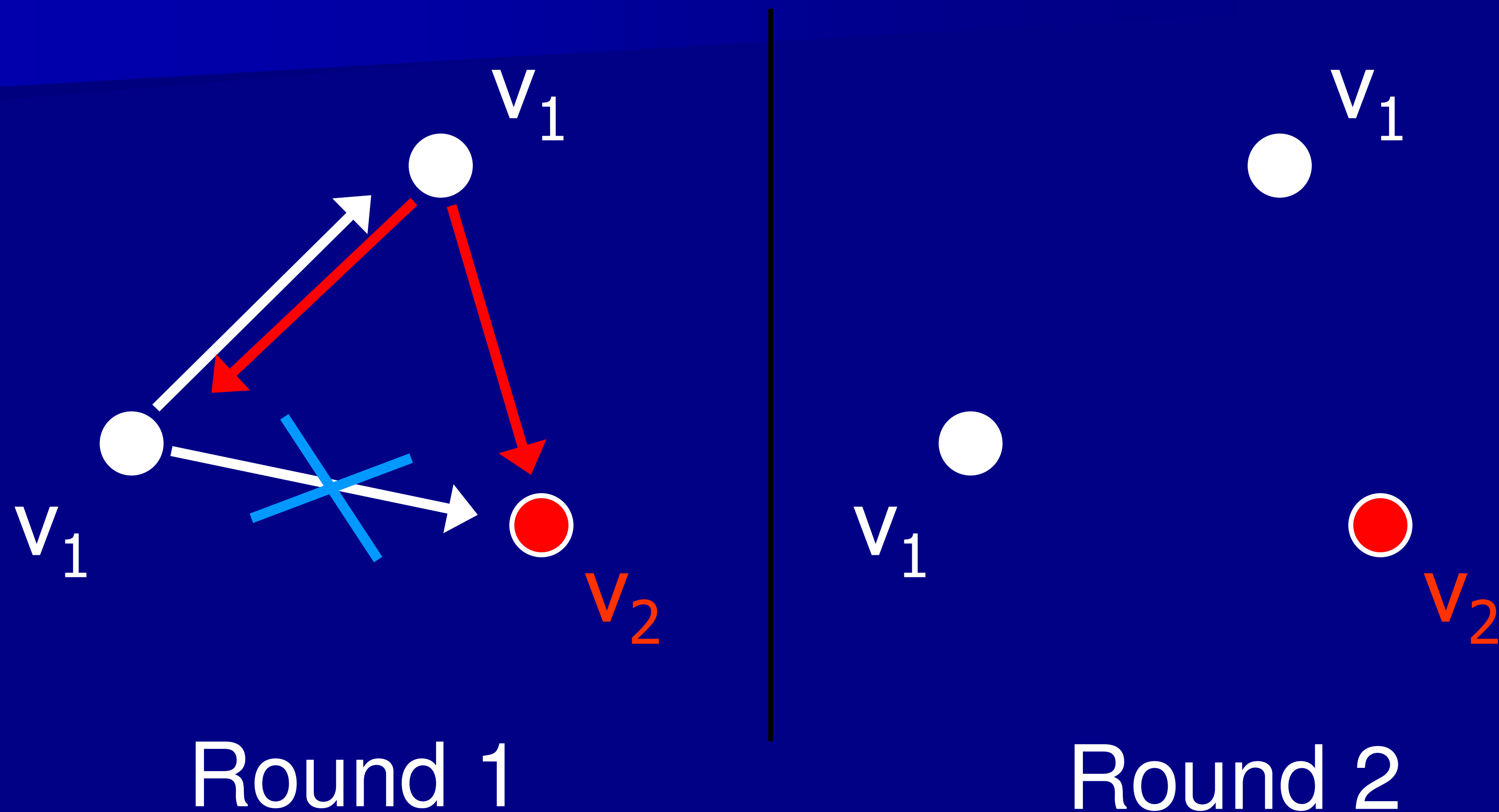
- Estimate := initial value
- Algorithm executes in super-rounds:
  - Round 1: Vote round
    - Active nodes vote on a value
    - If no collisions detected, then estimate := the smallest value heard
  - Round 2: Veto round
    - Anybody can veto if collision detected in Round 1
    - If nobody vetoes, then decide estimate and halt

Agreement with ◊*AC*

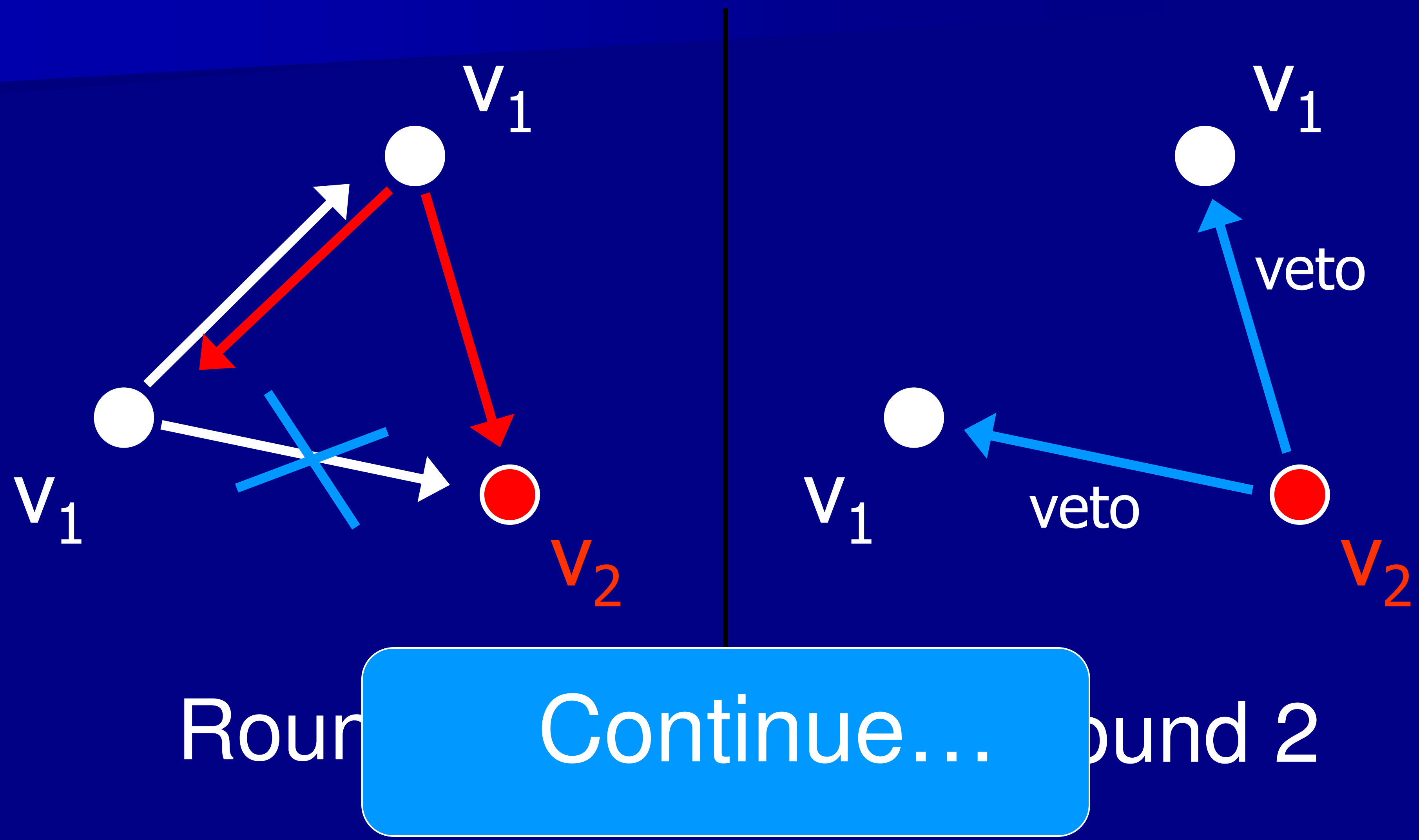


Round 1

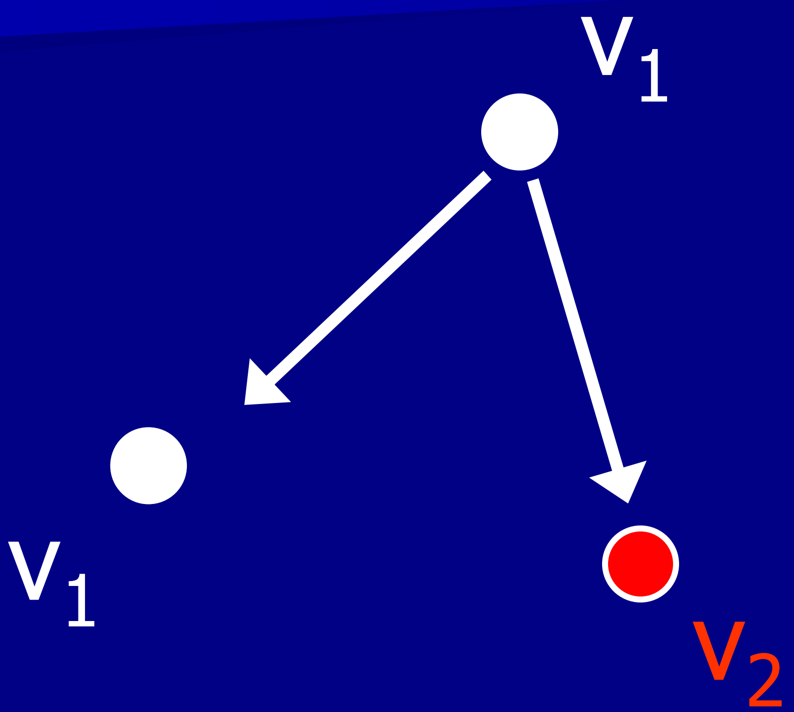
Agreement with ◊$\mathcal{AC}$



Agreement with ◊*AC*

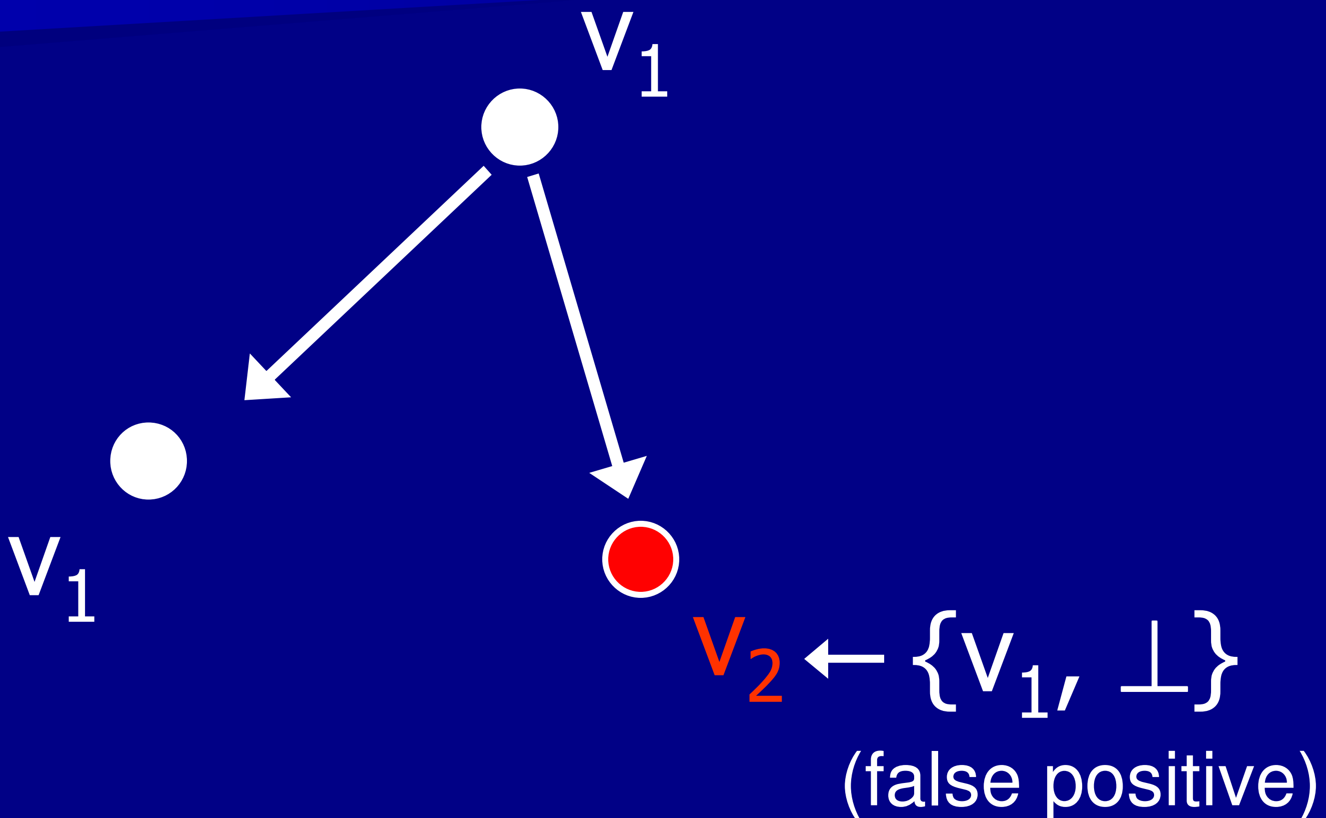


Agreement with ◊$\mathcal{AC}$



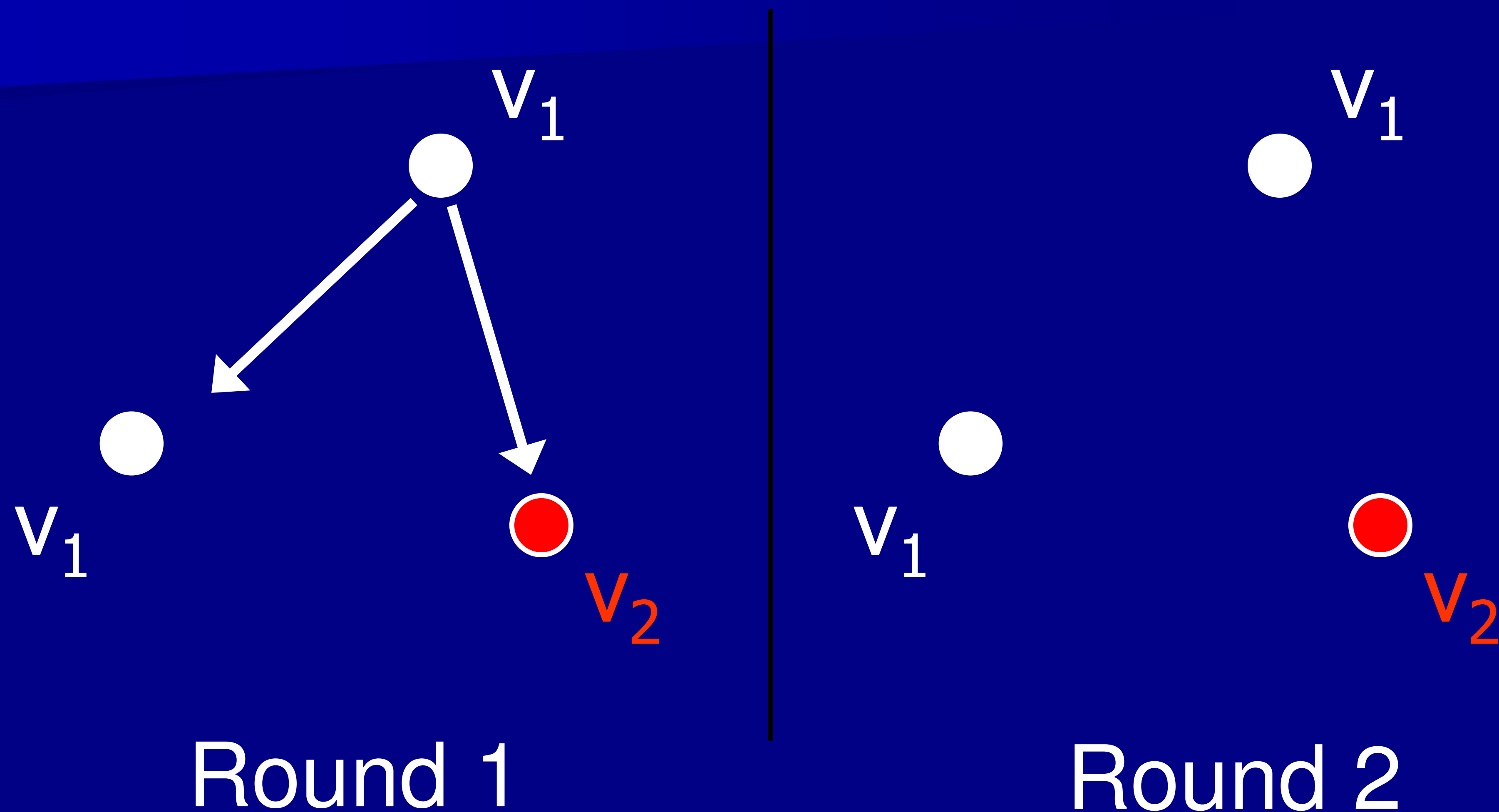
Round 1

Agreement with ◊AC

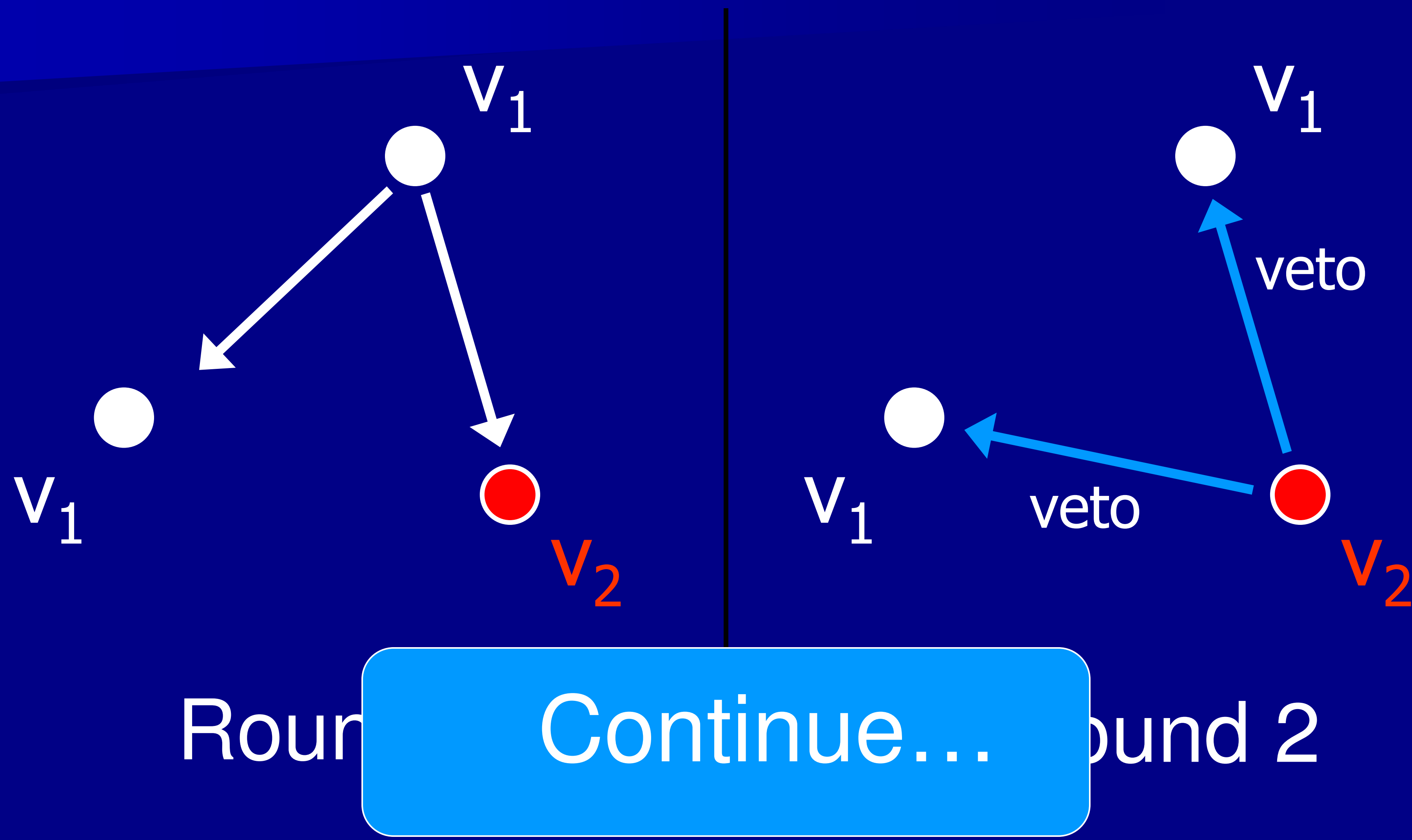


Round 1

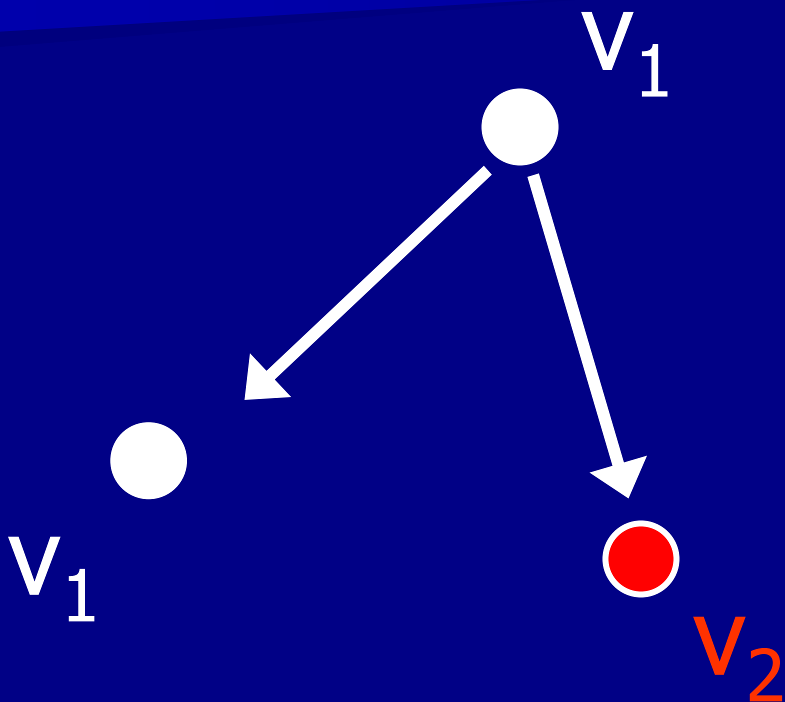
Agreement with ◊$\mathcal{AC}$



Agreement with ◊*AC*

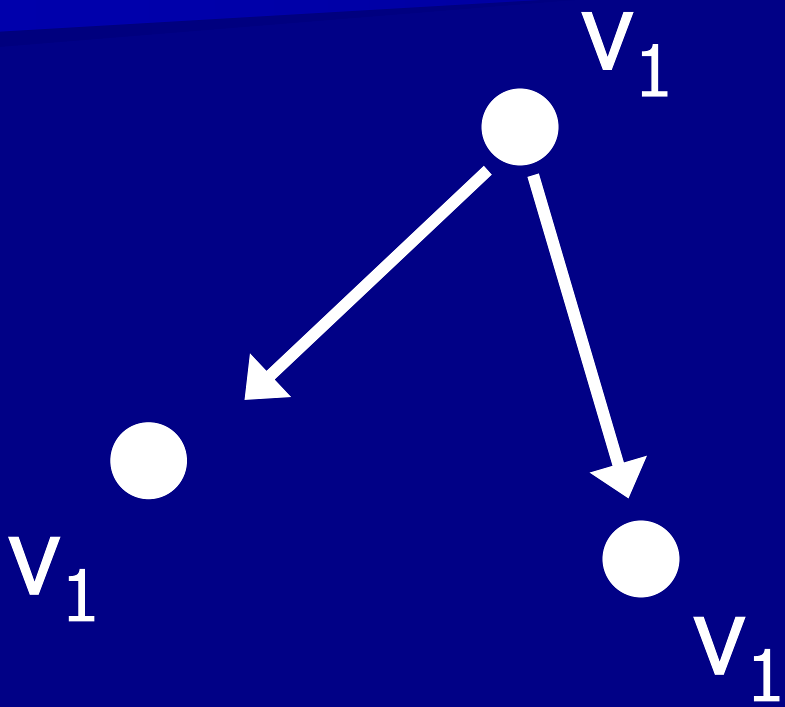


Agreement with ◊*AC*



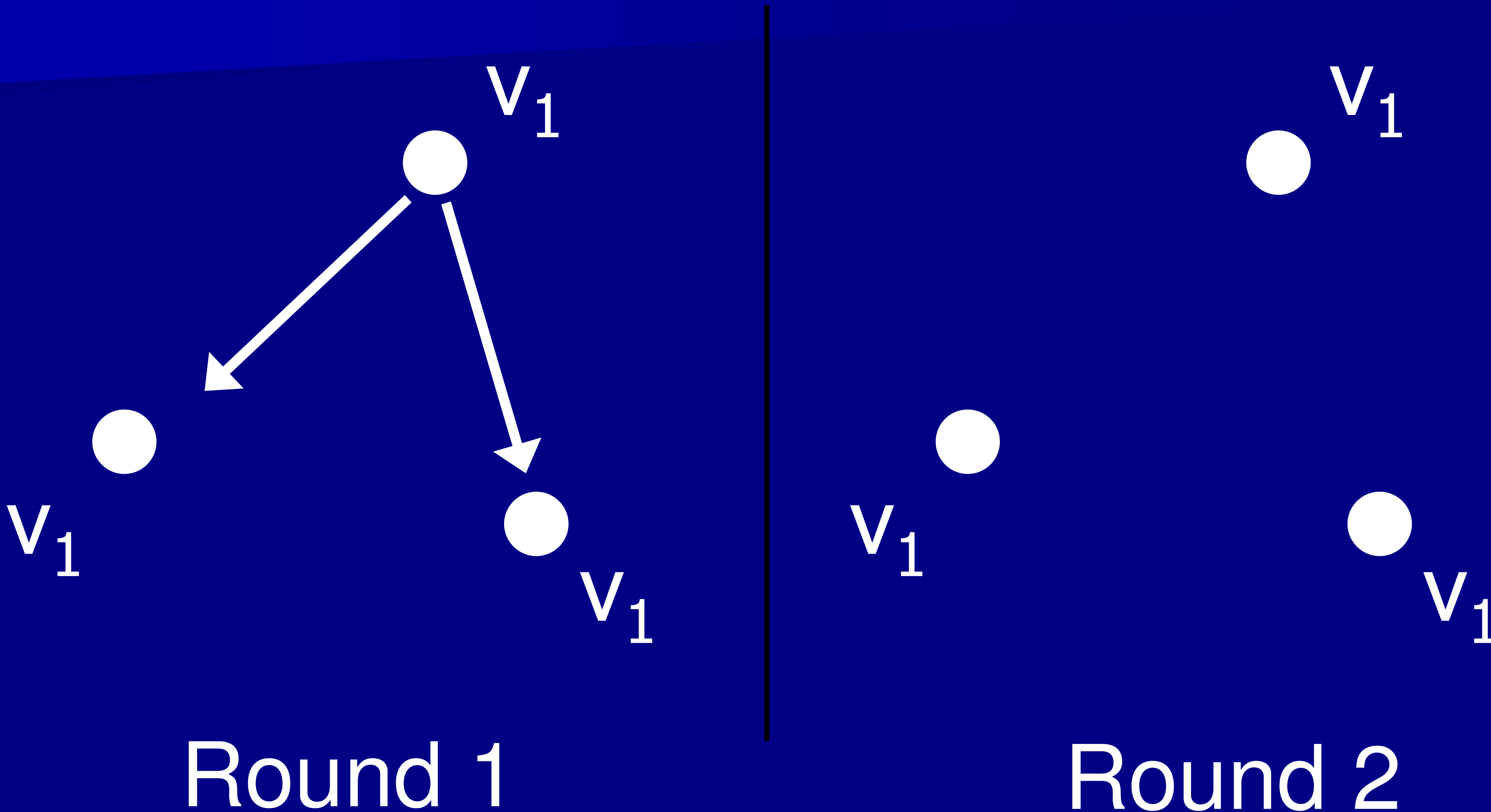
Round 1

Agreement with ◊*AC*

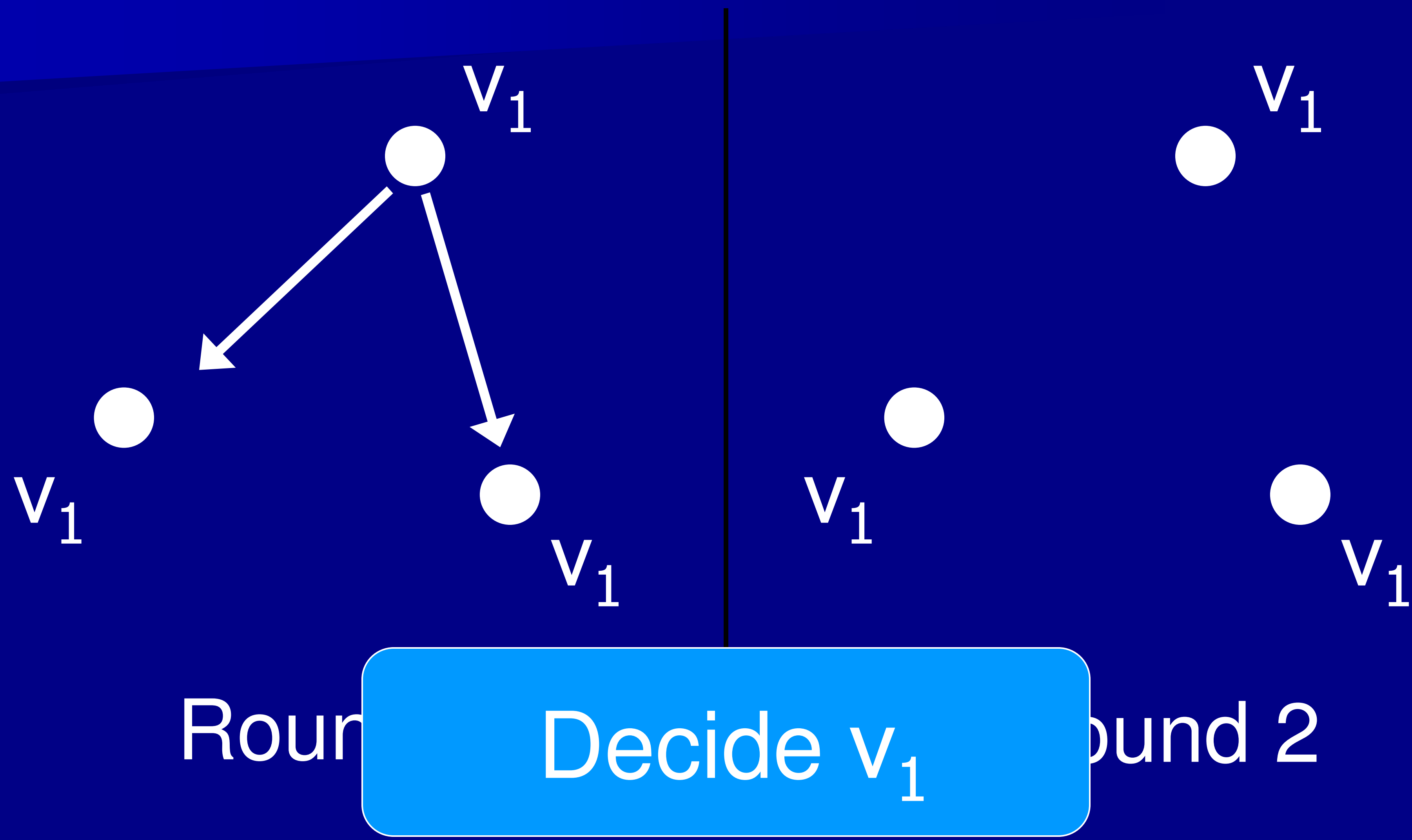


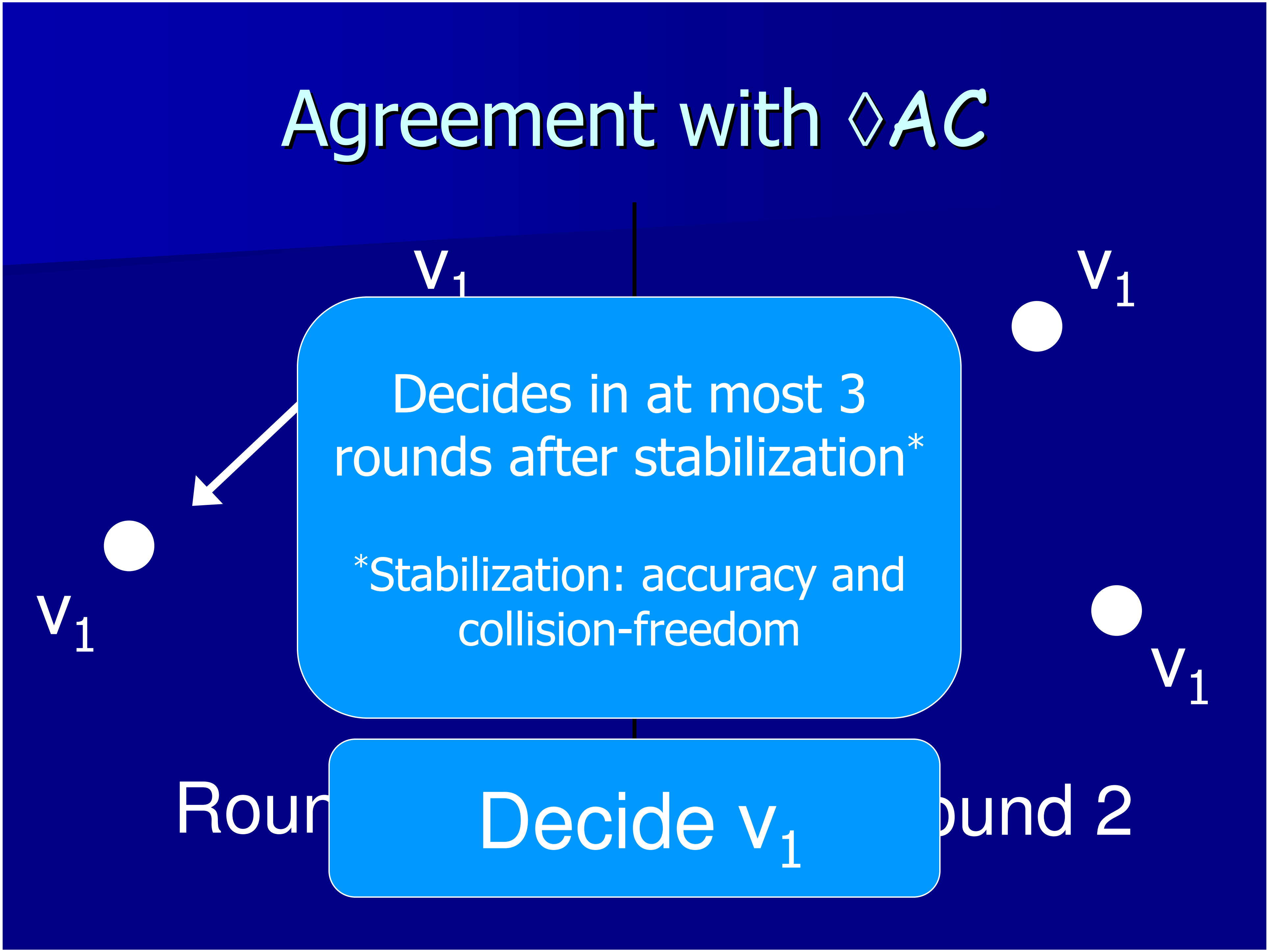
Round 1

Agreement with ◊*AC*



Agreement with ◊*AC*





Agreement with ◊AC
v1
v1
v1
v1
Decides in at most 3 rounds after stabilization*
*Stabilization: accuracy and collision-freedom
Roun
ound 2
Decide v1

Agreement with CD

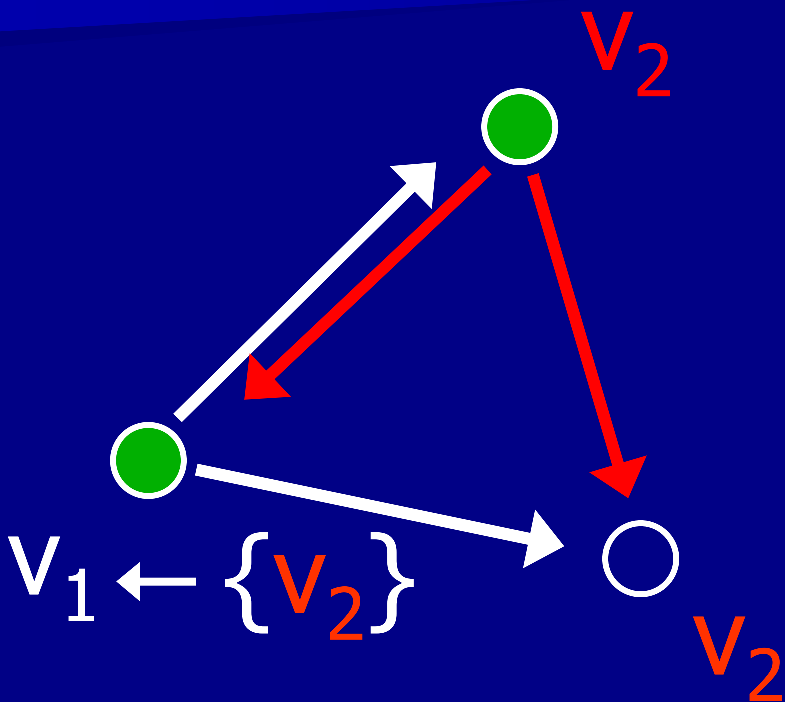
	Eventual Collision Freedom	**No Collision Freedom**
$\mathcal{AC}$	$\Theta(1)$	$\Theta(\log \|V\|)$
maj-$\mathcal{AC}$	$\Theta(1)$	$\Theta(\log \|V\|)$
0-$\mathcal{AC}$	$\Theta(\log \|V\|)$	$\Theta(\log \|V\|)$
$\diamond\mathcal{AC}$	$\Theta(1)$	Impossible
maj-$\diamond\mathcal{AC}$	$\Theta(1)$	Impossible
0-$\diamond\mathcal{AC}$	$\Theta(\log \|V\|)$	Impossible

V is the value domain

Agreement with *Maj-◊AC*

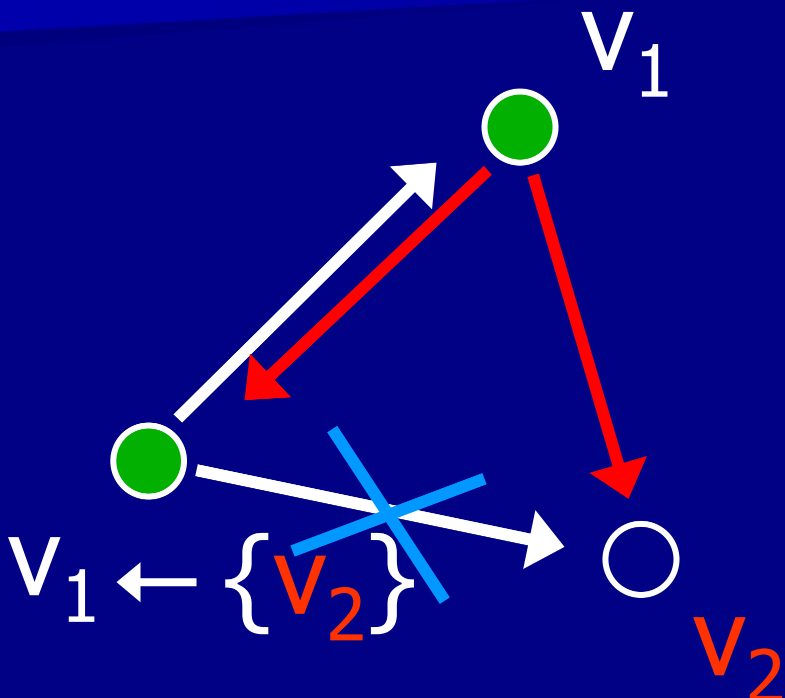
- Estimate := initial value
- Algorithm executes in super-rounds:
  - Round 1: Vote round
    - Active nodes vote on a value
    - If no collisions detected, then estimate := the smallest value heard
  - Round 2: Veto round
    - Veto if collision detected in Round 1 or **#different values received in Round 1 > 1**
    - If nobody vetoes, then decide estimate and halt

Agreement with *Maj-◊AC*



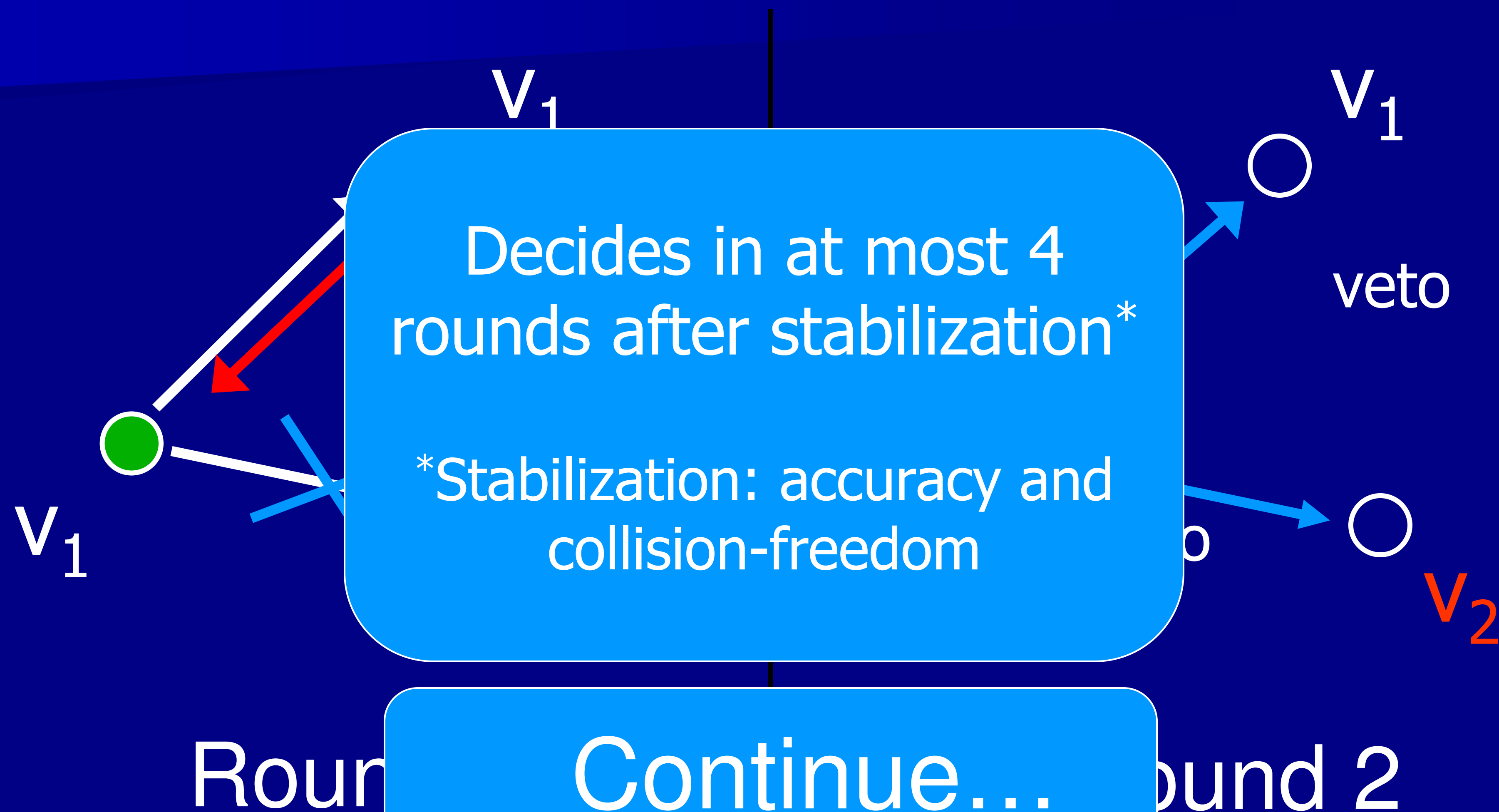
Round 1

Agreement with *Maj-◊AC*



Round 1

Agreement with *Maj-◊AC*

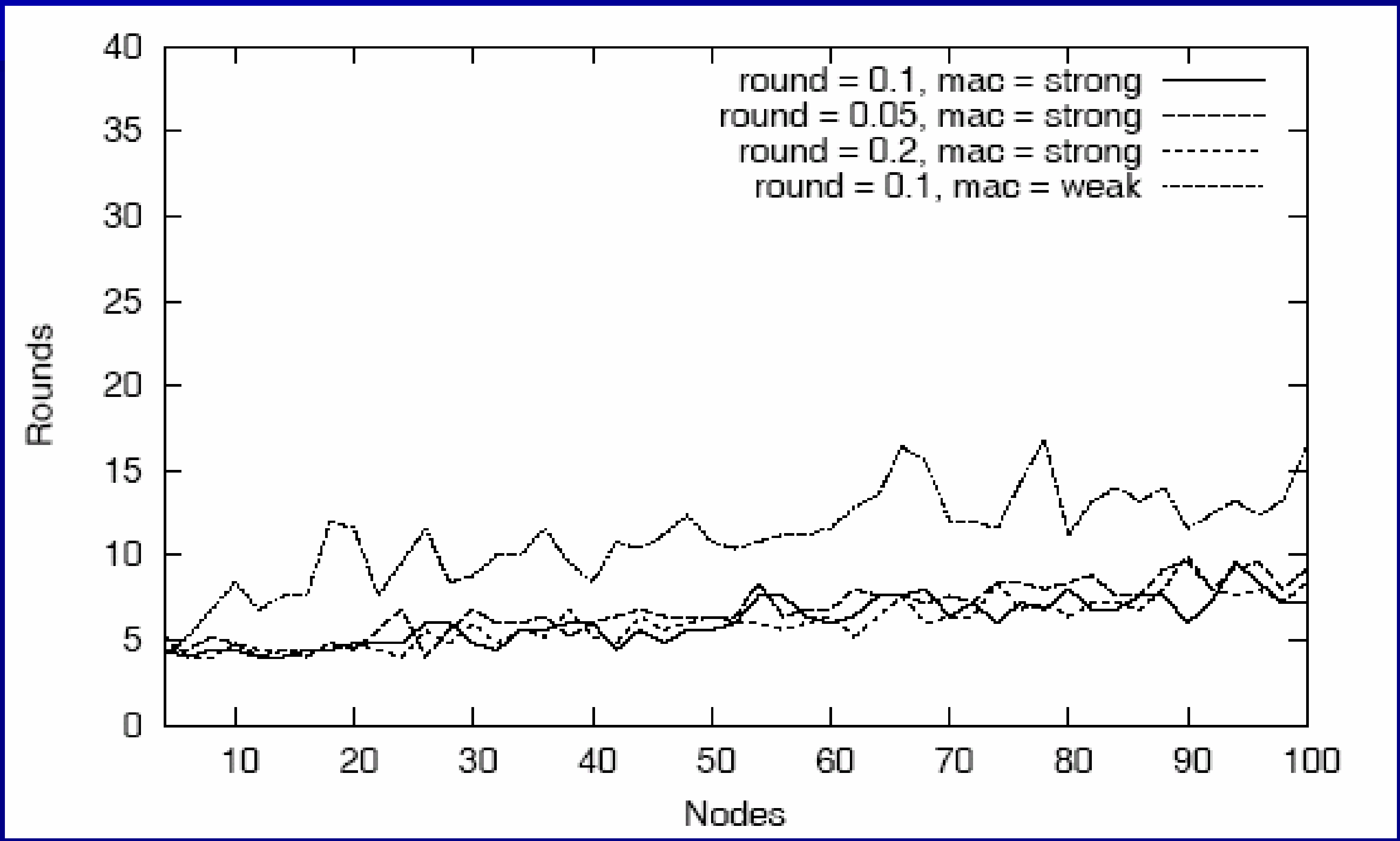


Decides in at most 4 rounds after stabilization*

*Stabilization: accuracy and collision-freedom

Continue…

Maj-◊AC Consensus: Simulations



NS-2, 802.11

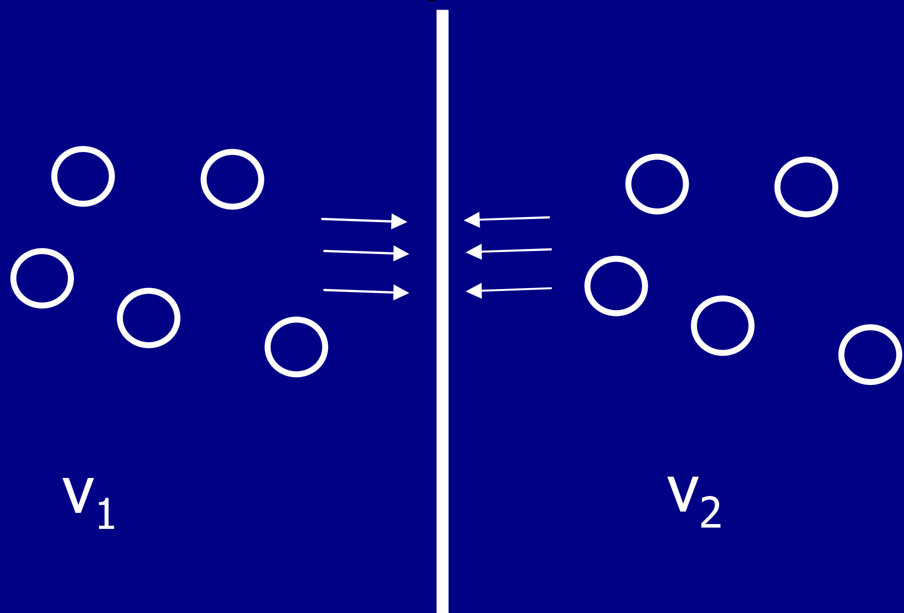
Consensus with CD

	Eventual Collision Freedom	No Collision Freedom
$\mathcal{AC}$	$\Theta(1)$	$\Theta(\log \lvert V \rvert)$
maj-$\mathcal{AC}$	$\Theta(1)$	$\Theta(\log \lvert V \rvert)$
0-$\mathcal{AC}$	$\Theta(\log \lvert V \rvert)$	$\Theta(\log \lvert V \rvert)$
$\Diamond\mathcal{AC}$	$\Theta(1)$	Impossible
maj-$\Diamond\mathcal{AC}$	$\Theta(1)$	Impossible
0-$\Diamond\mathcal{AC}$	$\Theta(\log \lvert V \rvert)$	Impossible

V is the value domain

Agreement with ½-AC

- ½-complete, accurate collision detector



2^r broadcast schedules for the first r rounds

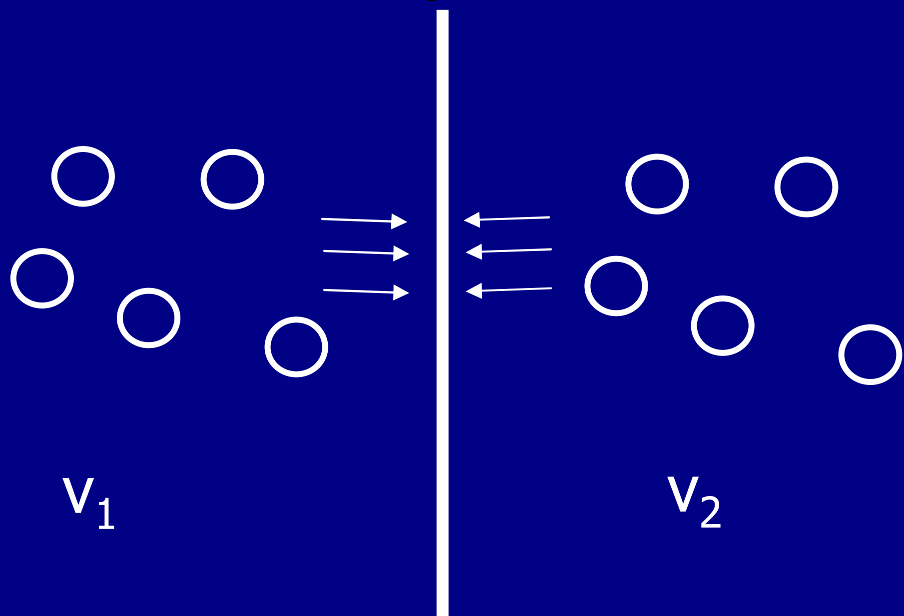
|V| possible values

For k < log(|V|), at most |V|/2 broadcast schedules to follow ➔

Exists two values resulting in the same broadcast schedule of length k

Agreement with ½-*AC*

- ½-complete, accurate collision detector



2^r broadcast schedules for the first r rounds

|V| possible values

For k < log(|V|), at most |V|/2 broadcast schedules to follow ➔

Exists two values resulting in the same broadcast schedule of length k

Agreement with 0-◊*AC*

- Everybody broadcasts its initial value
  - estimate := $\perp \in$ M ? initVal : min(M)
  - abort := 0

pre-
pare

- For every bit B of estimate:
  - If (B = 1 or abort) then broadcast Veto
  - If received something and B=0, abort := 1

pro-
pose

- If abort, then broadcast Veto
  - If nothing received, decide estimate, halt

de-
cide

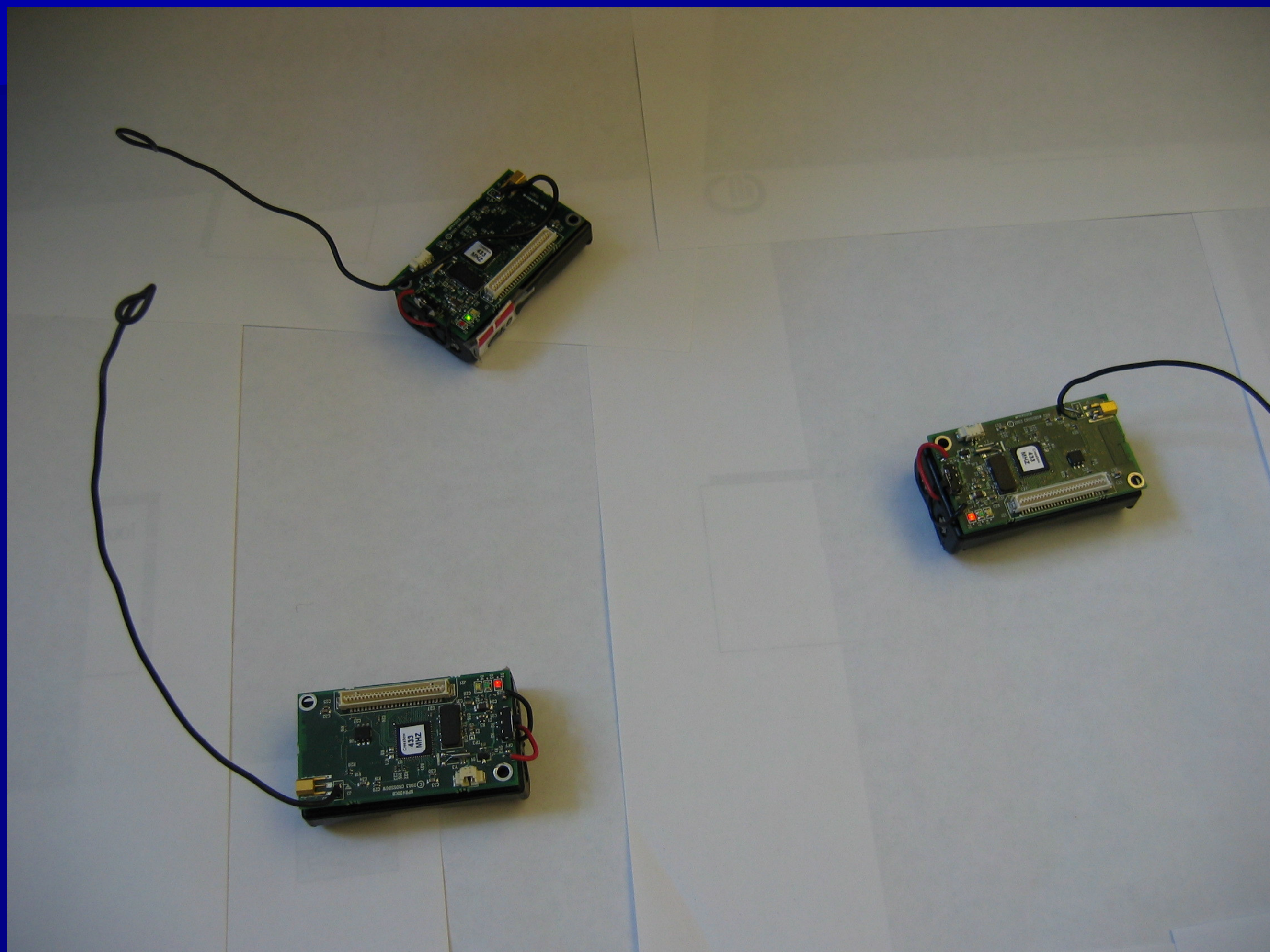
Implementing Collision Detection

- Carrier sensing
  - – CSMA: 802.11, 802.15.4, sensor wireless MAC
  - – Sense carrier in the idle mode
- Cyclic Redundancy Check (CRC)
- Preamble detection
  - – Normally, preamble is only detected in the synchronization state
  - – If detected in the receive state → collision

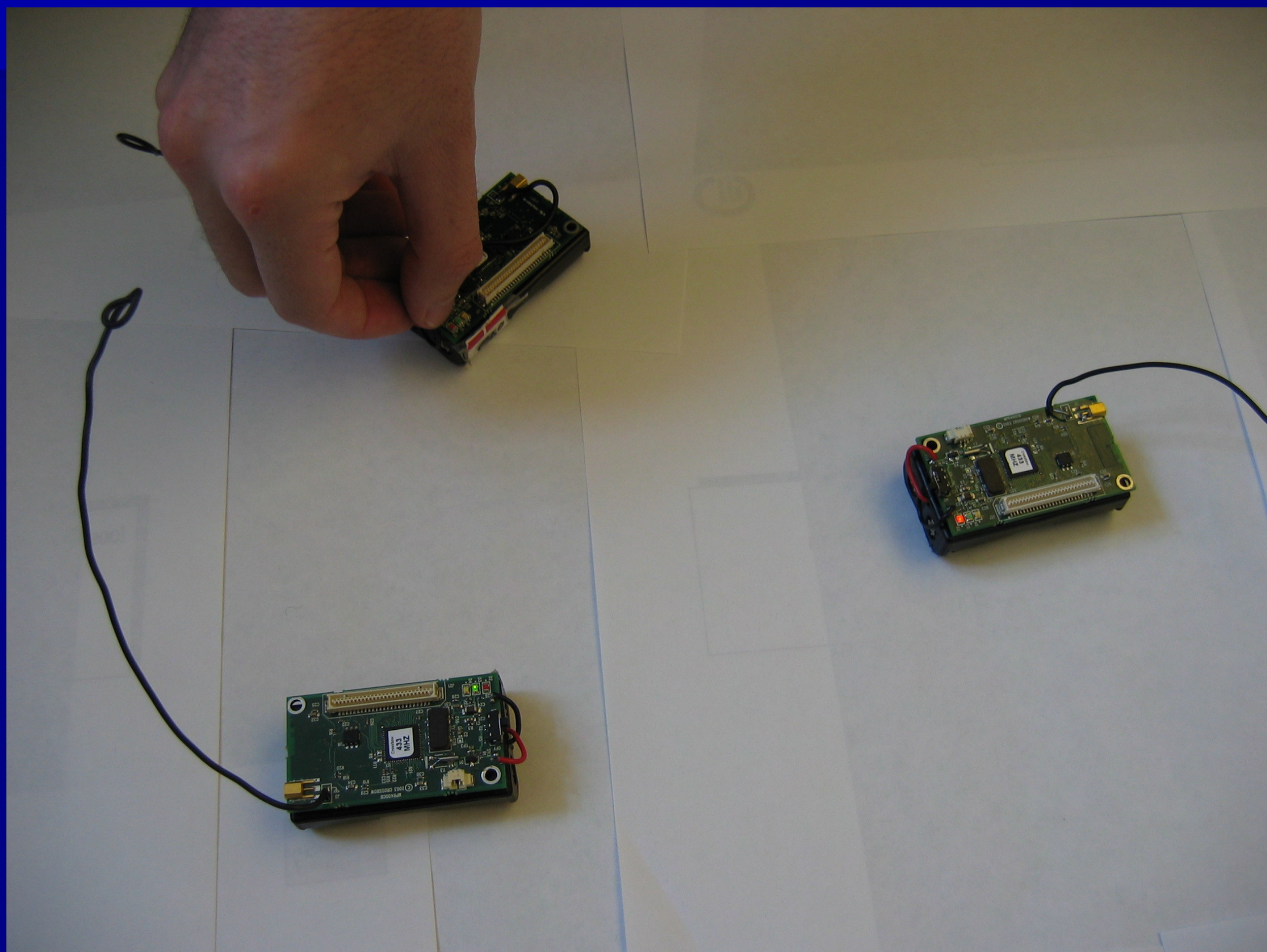
Local Agreement: Conclusions

- Local infrastructure for realistic collision models
  - Non-uniform collision
- Necessary building blocks:
  - Collision detector for consistency
  - Contention manager for progress
- The most realistic yet powerful collision detector is *Maj-◊AC*

Prototype Implementation



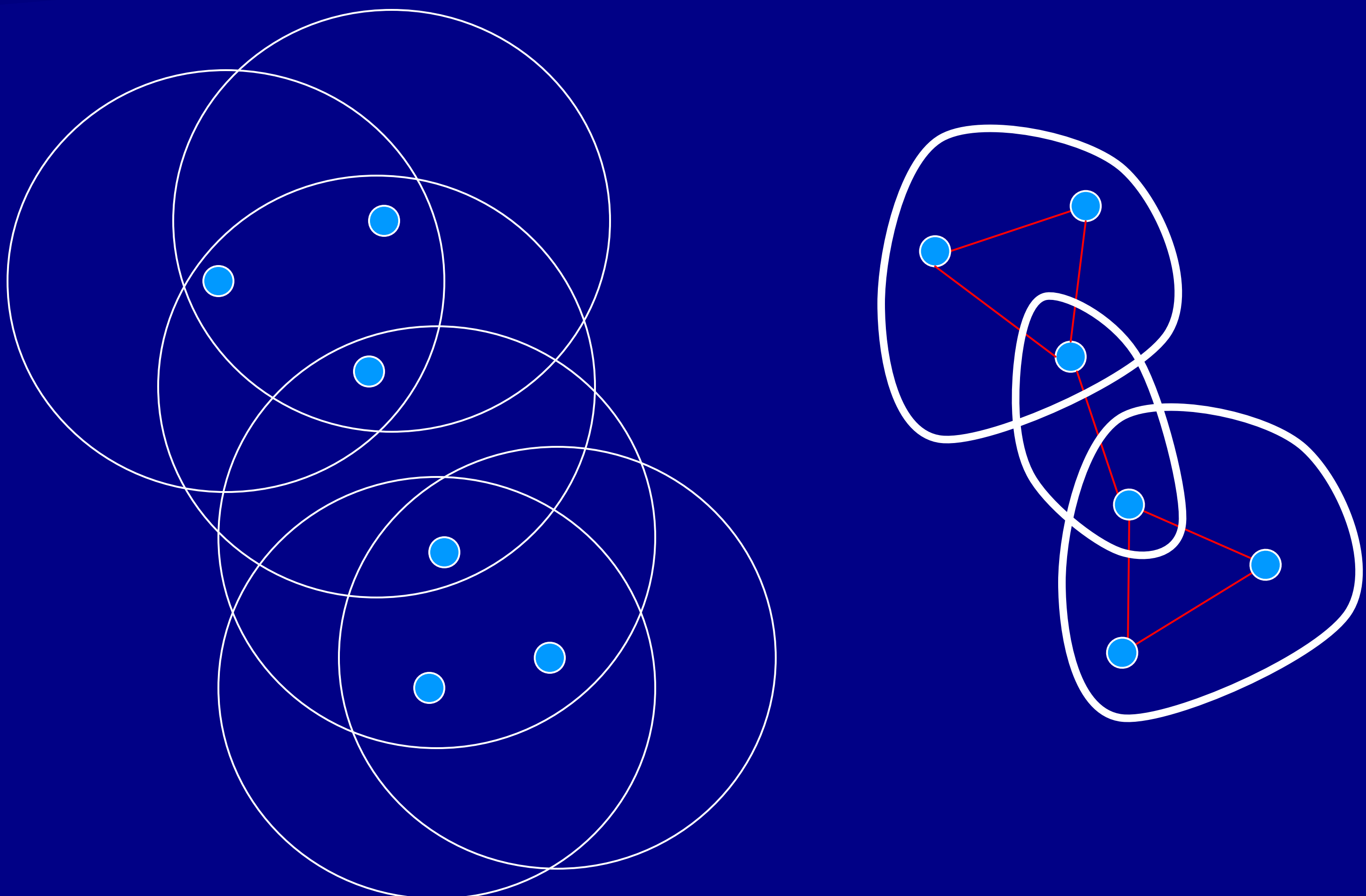
Prototype Implementation



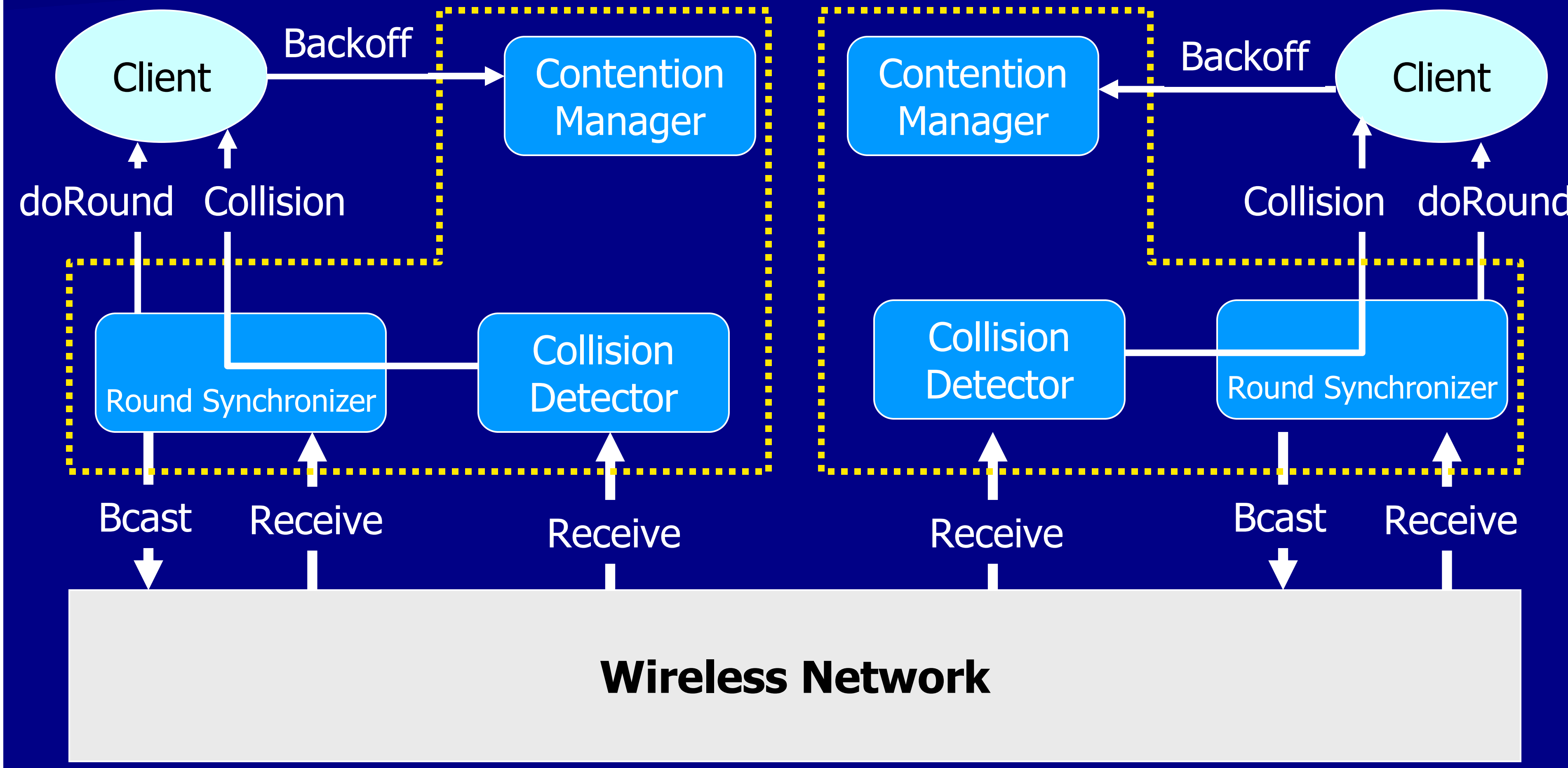
Fault-Tolerance Middleware

- Local infrastructure
  - State machines and virtual nodes
  - Local agreement
- Global infrastructure
  - Round synchronization
  - Broadcast
  - Quorums

Multi-Hop Wireless Networks



Middleware for Multi-Hop Networks [1]



Round Synchronizer

- Supports synchronous protocols
- Nodes synched with neighbors

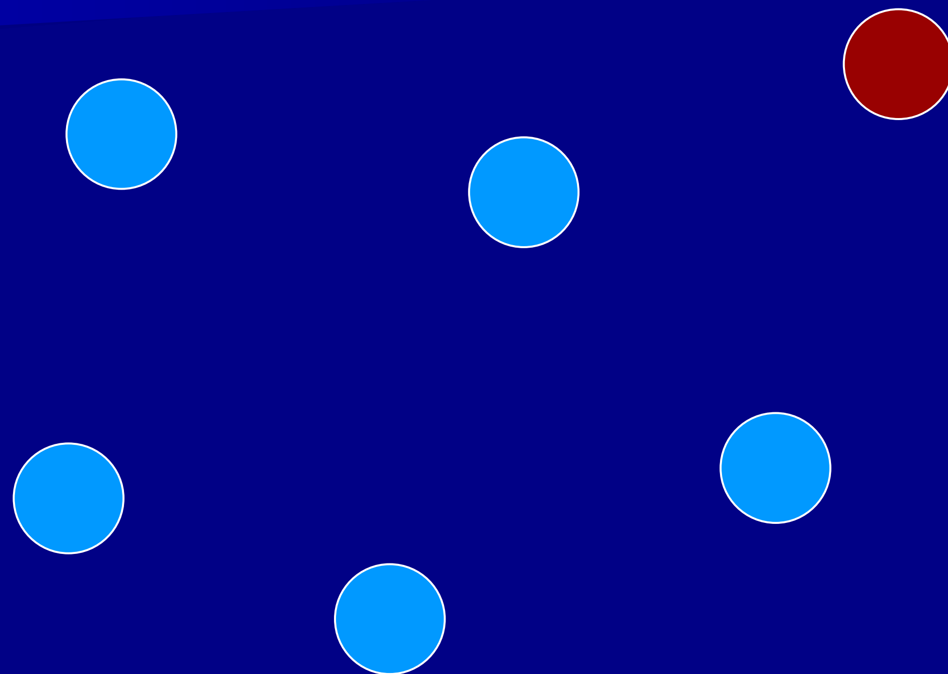
During each round *r*, a protocol running on process *p* is allowed to broadcast one or zero messages to *p*'s neighbors. The component returns to the protocol a set containing all round *r* messages sent by *p*'s neighbors and successfully received by *p*.

An Example: Reliable Broadcast

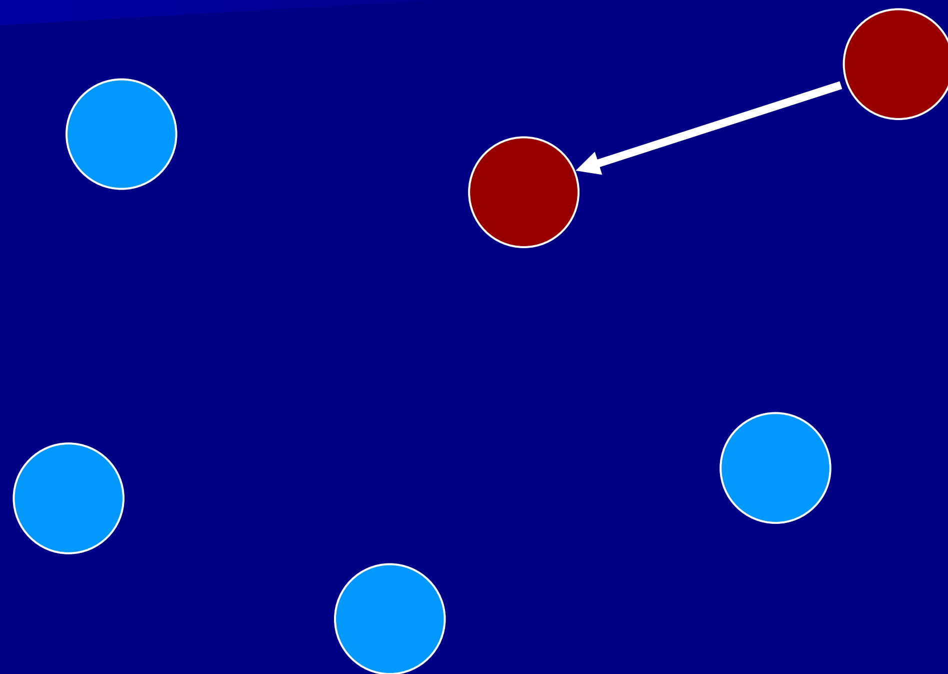
```
init(m);
    started ← false
    active ← false
    msg ← m

doRound(rnd, msgs[], collision)
    if rnd mod 2 = 0 then  // Receive a message in even rounds.
        if (not started) and (|msgs| > 0) then
            started ← true
            msg ← msgs[0]
        if collision then
            return veto  // Broadcast a veto if receive failed.
        else
            return ∅
    else if rnd mod 2 = 1 then  // Receive a veto in odd rounds.
        if started then
            if (not collision) and (|msgs| = 0) and active then
                halt()  // If no vetoes, then done.
            else if (collision) or (|msgs| > 0) then
                active ← Backoff(eTooMany)
            else if (not collision) and (|msgs| = 0) and (not active) then
                active ← Backoff(eTooFew)
            if active then
                return msg  // Broadcast message.
        else return ∅
```

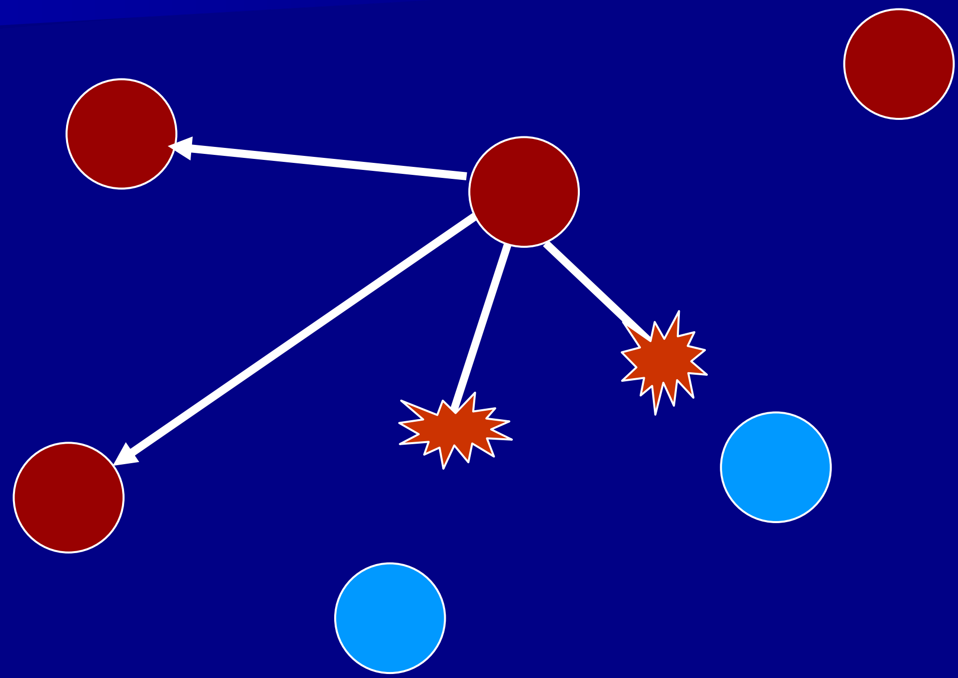
Reliable Broadcast



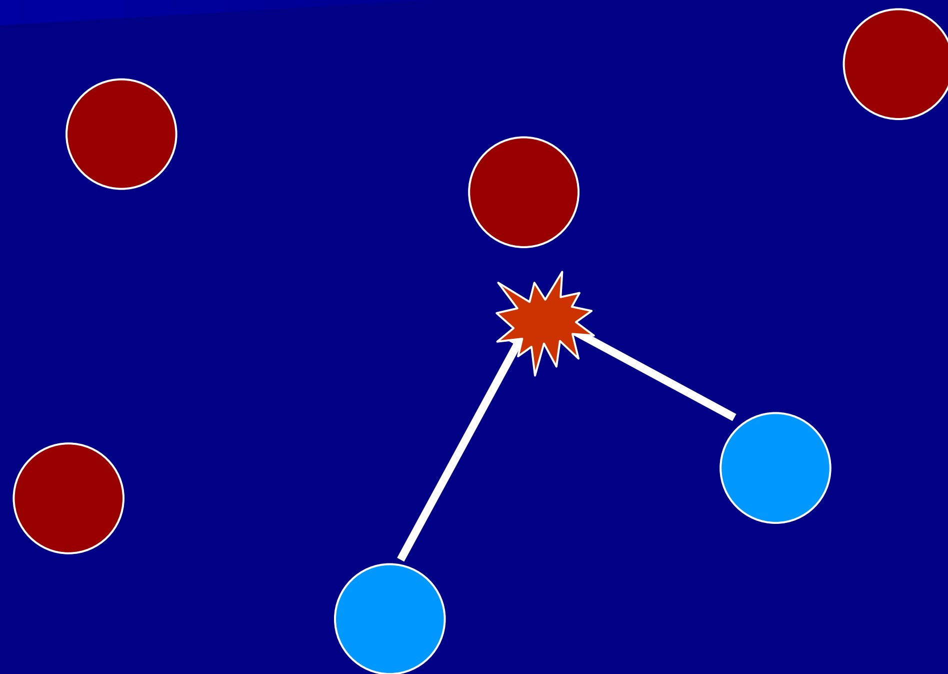
Reliable Broadcast



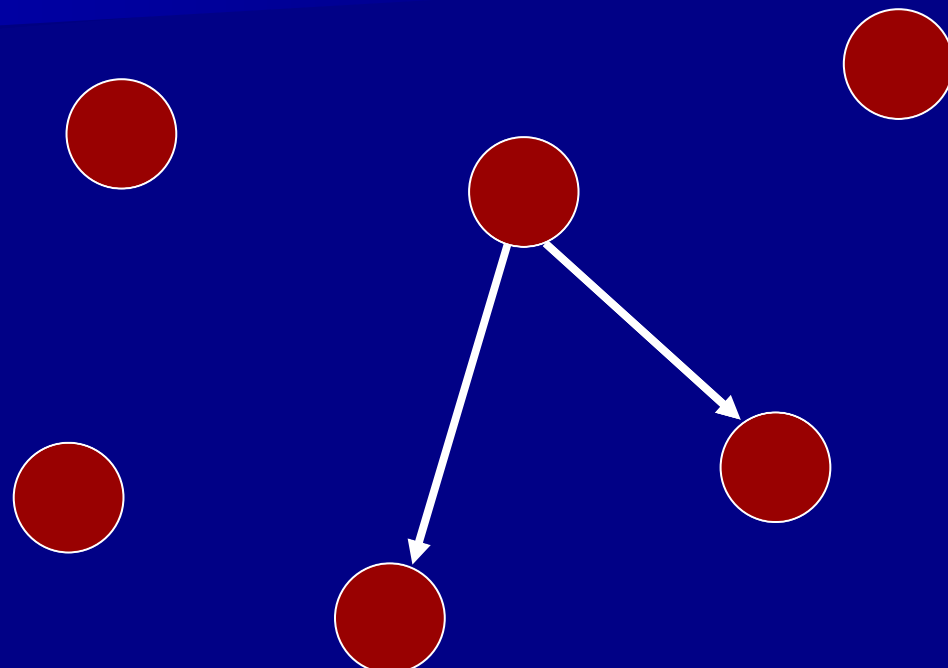
Reliable Broadcast



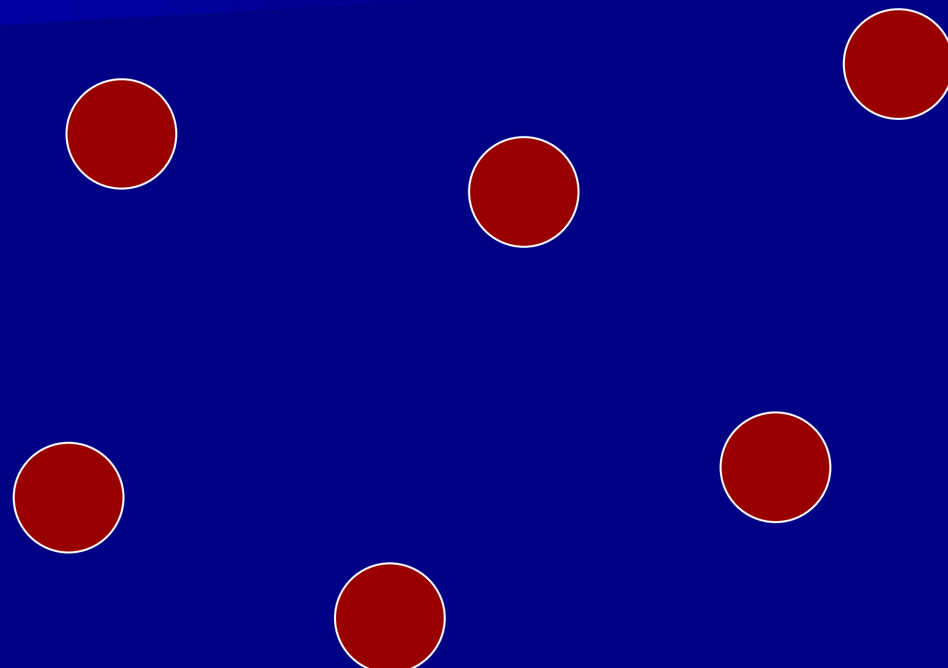
Reliable Broadcast



Reliable Broadcast



Reliable Broadcast



Implementing Round Synchronizer

- Use “start” message, or collision detection to synch with neighbors
- Use local timer to maintain local synch for bounded number of rounds
- Periodic resynchronizations required
  - Compensate for clock drift

Fault-Tolerance Middleware

- Local infrastructure
  - State machines and virtual nodes
  - Local agreement
- Global infrastructure
  - Round synchronization
  - Broadcast
  - Quorums

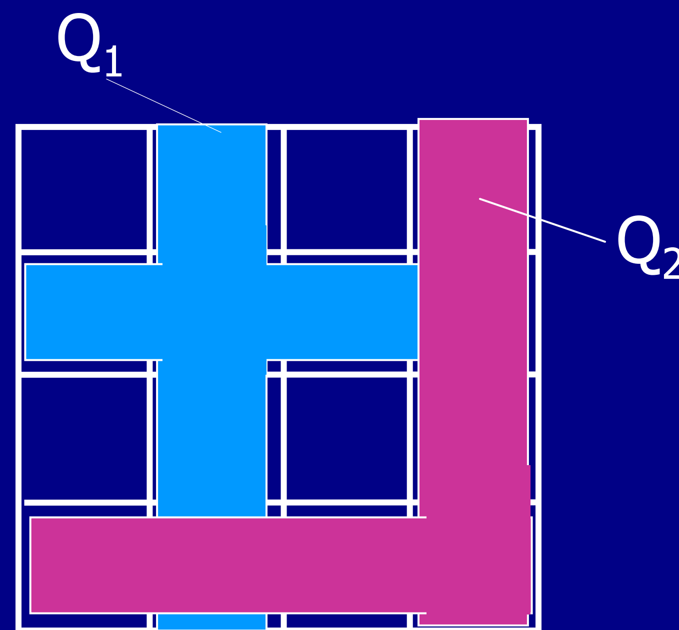
Quorum Systems

- Universe *U* of servers
- Quorum system: $\mathcal{Q} \subseteq 2^U : \forall Q_1, Q_2 \in \mathcal{Q} \; : \; Q_1 \cap Q_2 \neq \emptyset$
  - Intersection for coordination and information sharing among clients
- Advantages: Improved load and availability
- Applications: data replication, data dissemination, mutual exclusion, etc.

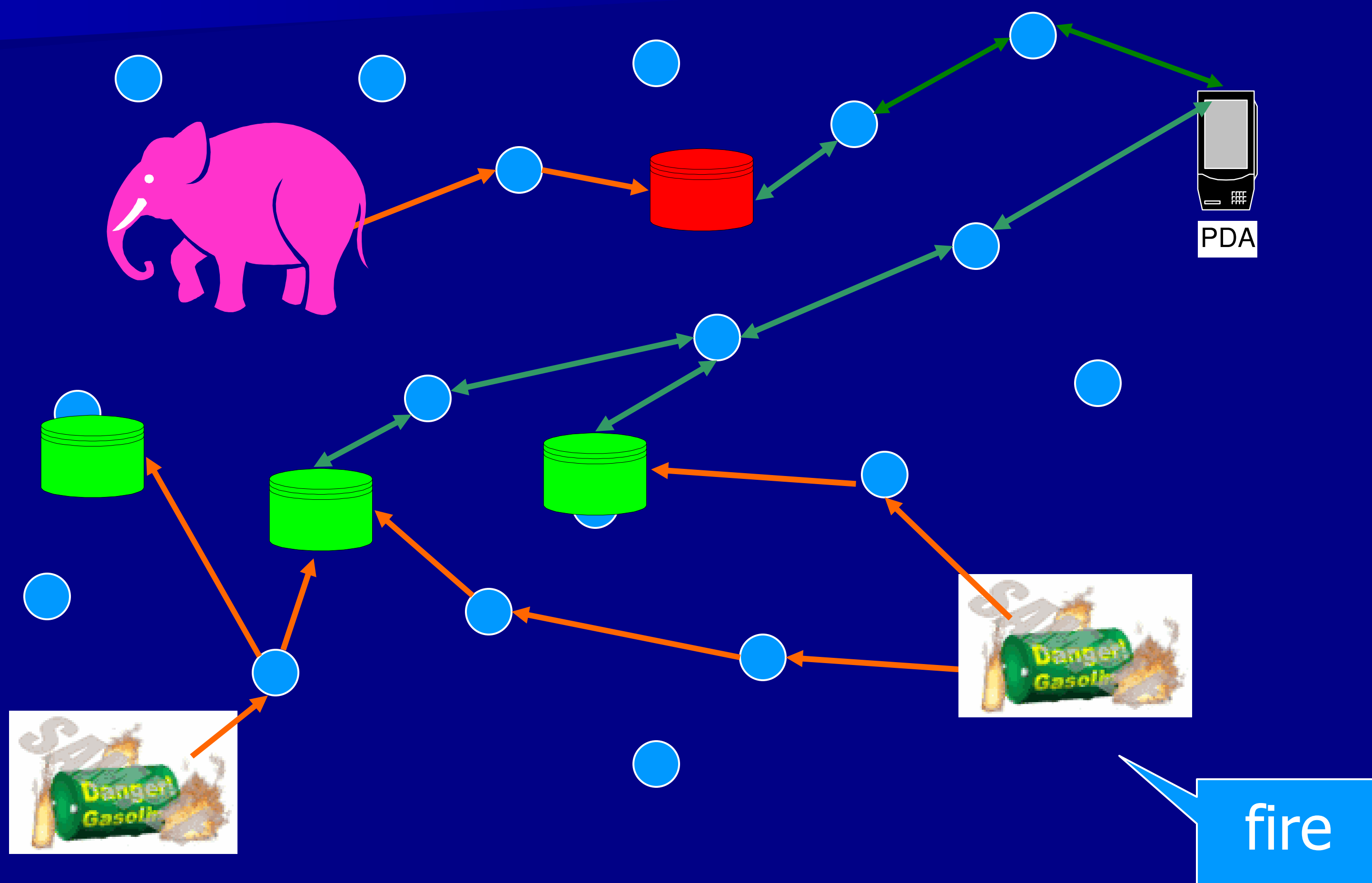
Quorum System Examples

- Threshold QS: a set of all sets containing a majority of servers in *U*
- Grid QS:

S_1	S_2	S_3	S_4
S_5	S_6	S_7	S_8
S_9	S_{10}	S_{11}	S_{12}
S_{13}	S_{14}	S_{15}	S_{16}



Data-Centric Event Storage

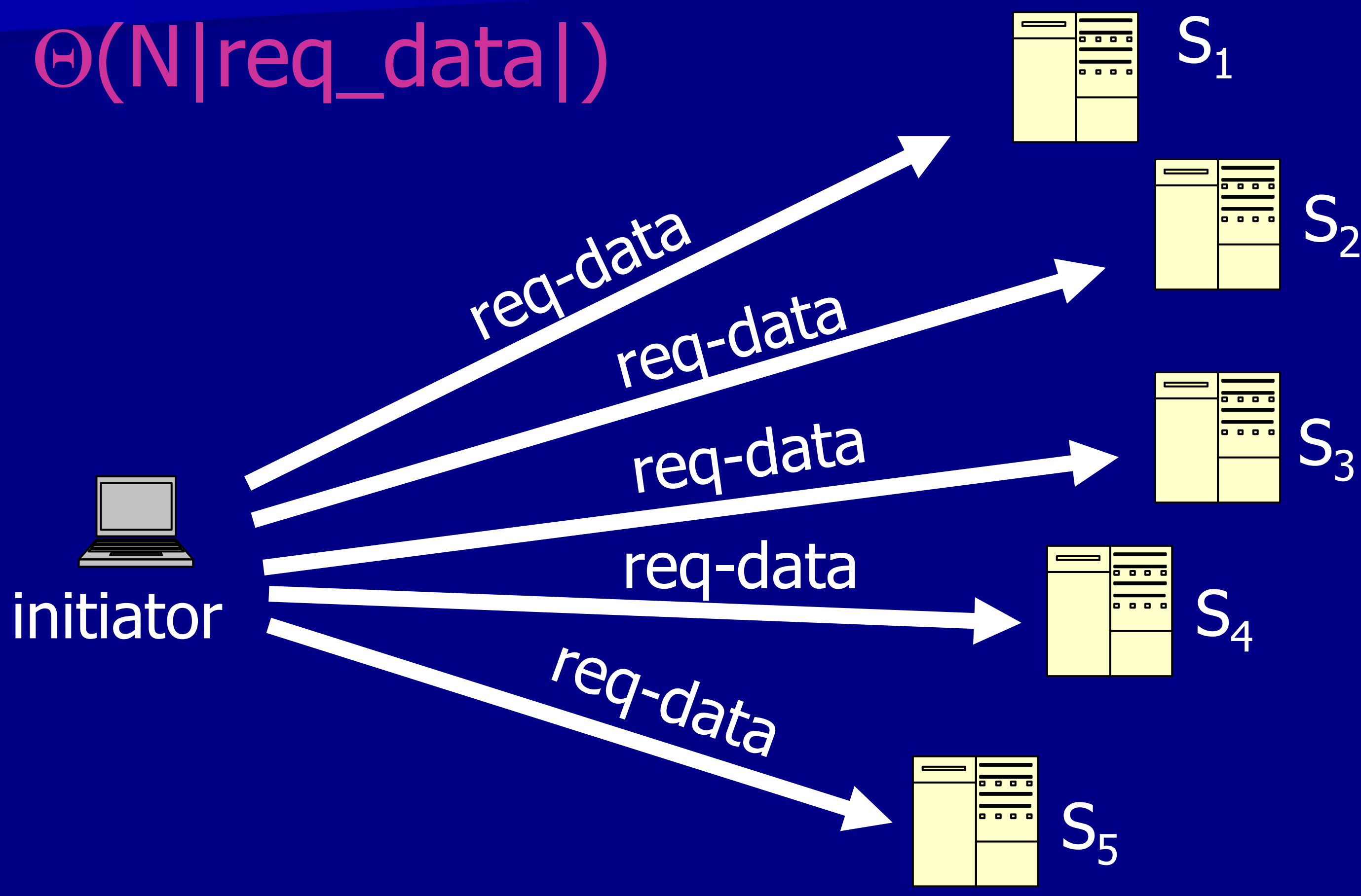


Accessing Quorums

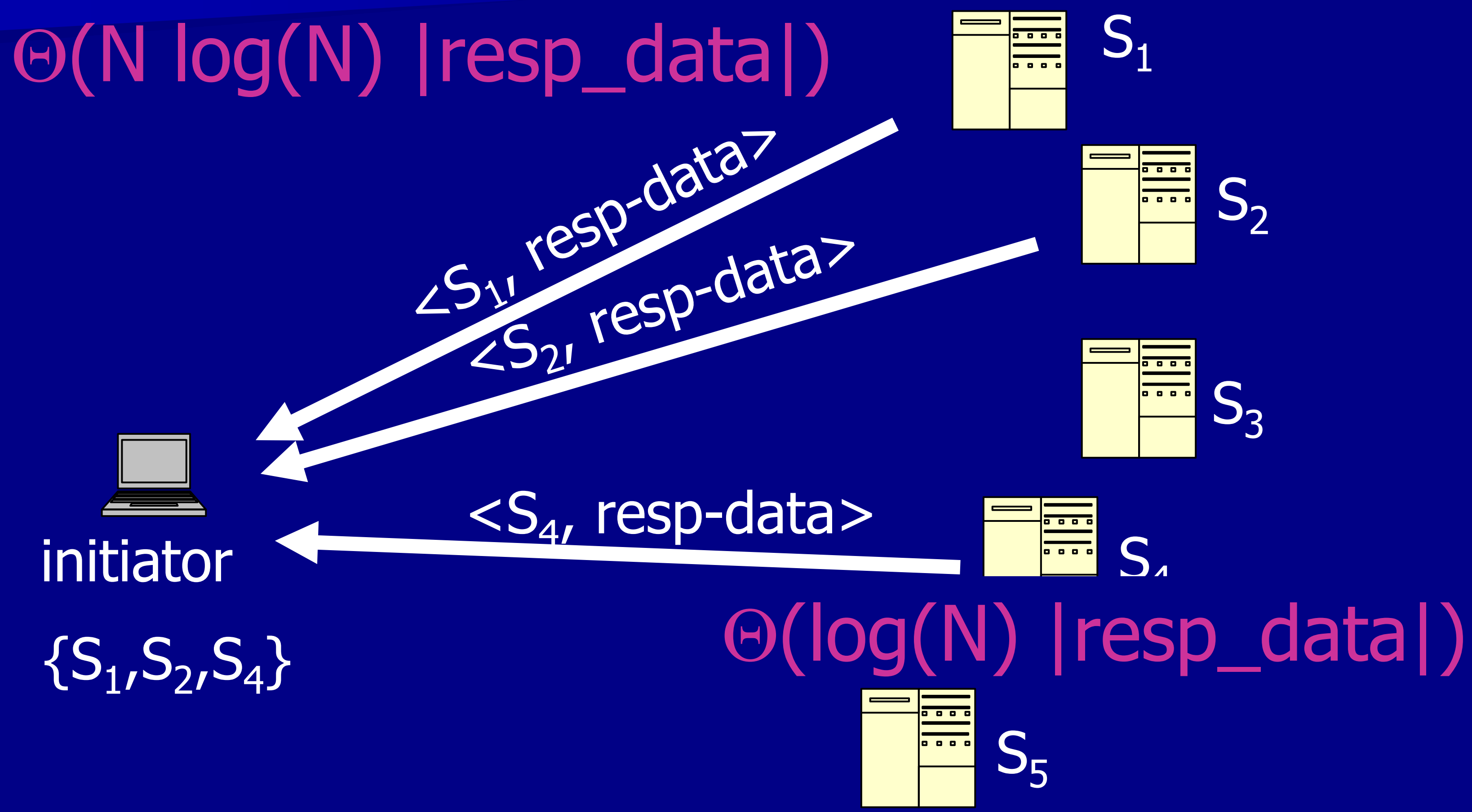
- Client (initiator) contacts servers until a full quorum of replies is collected
  - Variety of ways for doing that
- The initiator must be able to identify responding nodes
  - Majority: count responses
  - Grid: identify the square to fill

Accessing Quorums

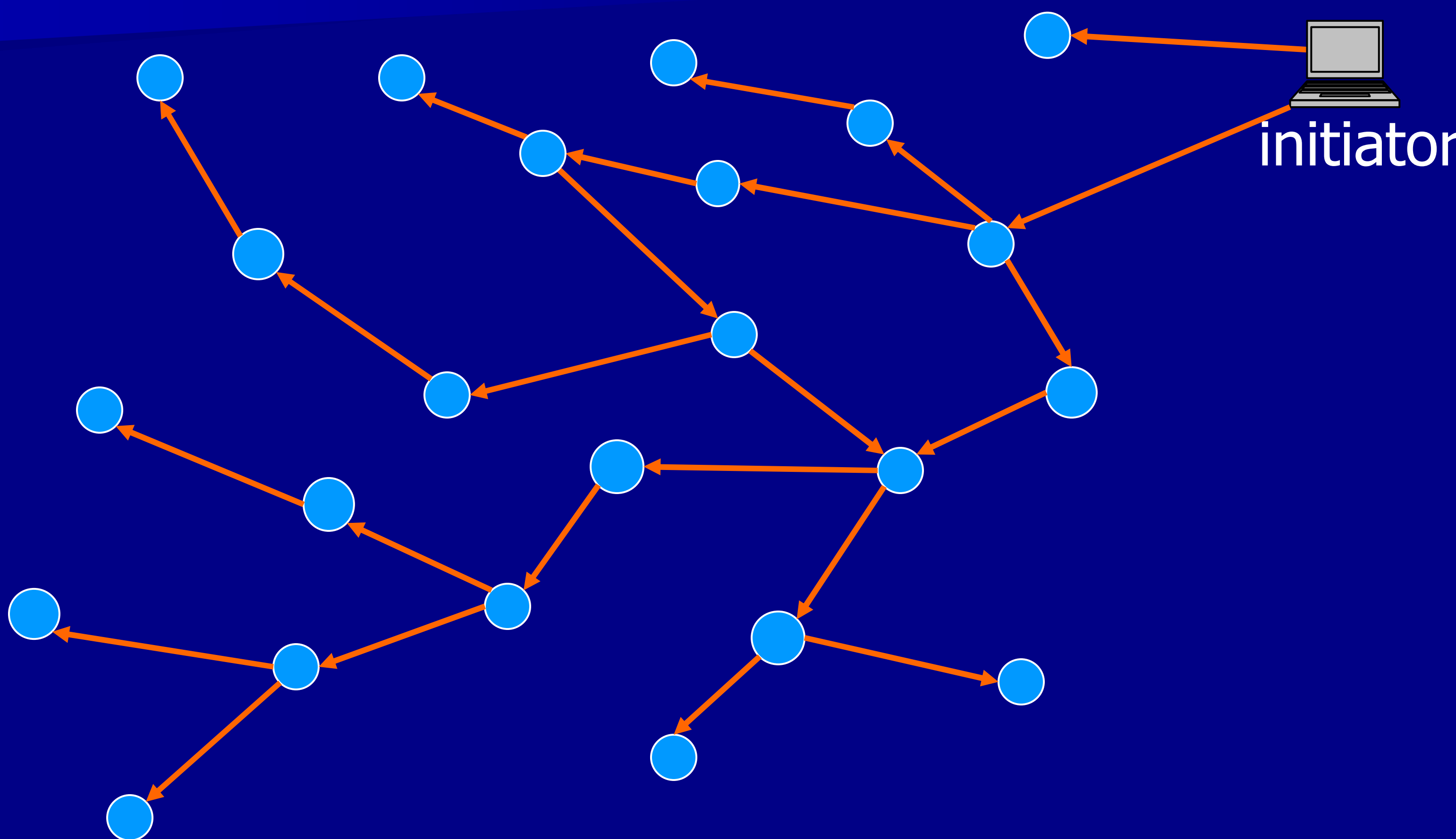
$\Theta(N|req\_data|)$



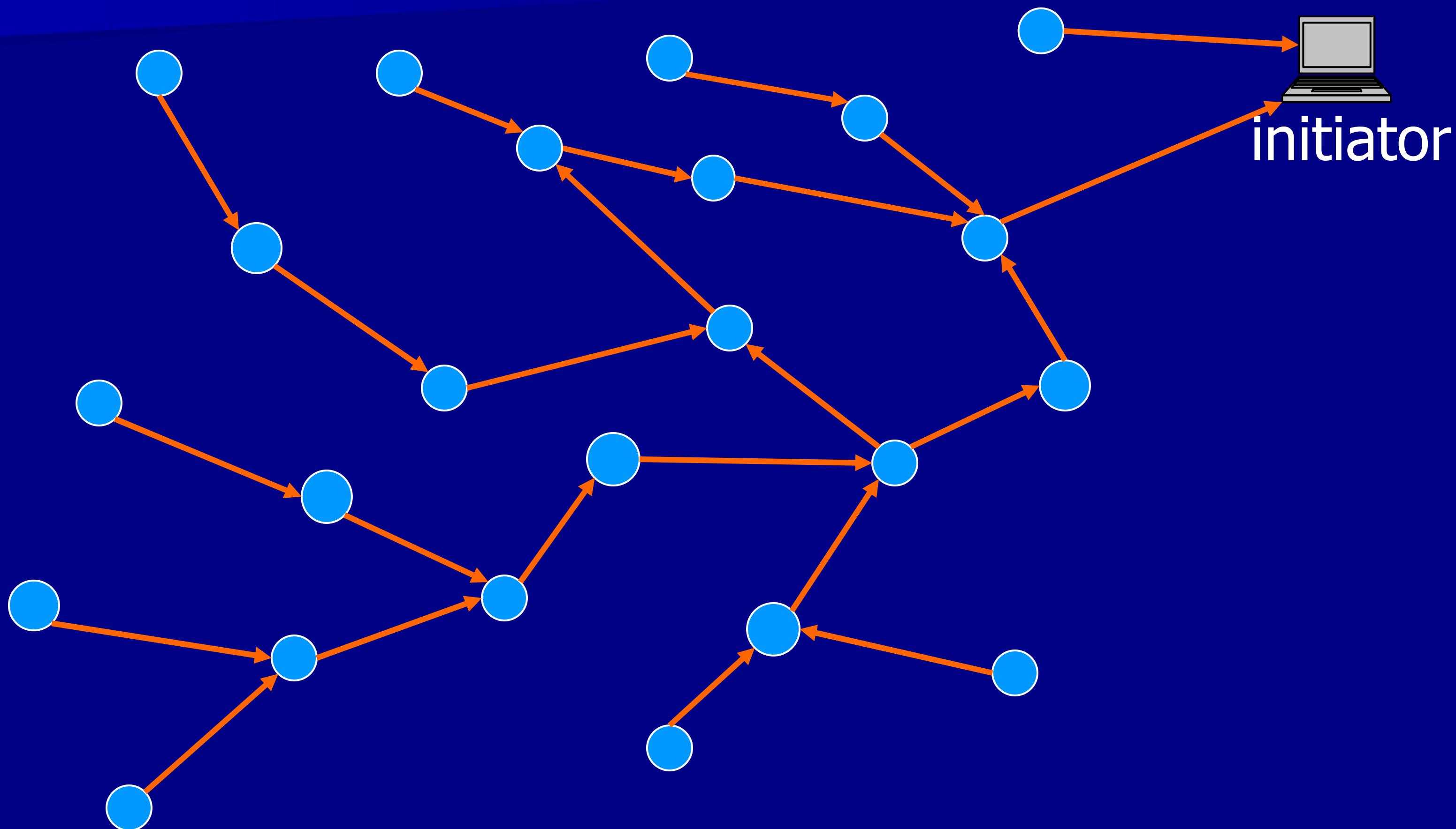
Accessing Quorums



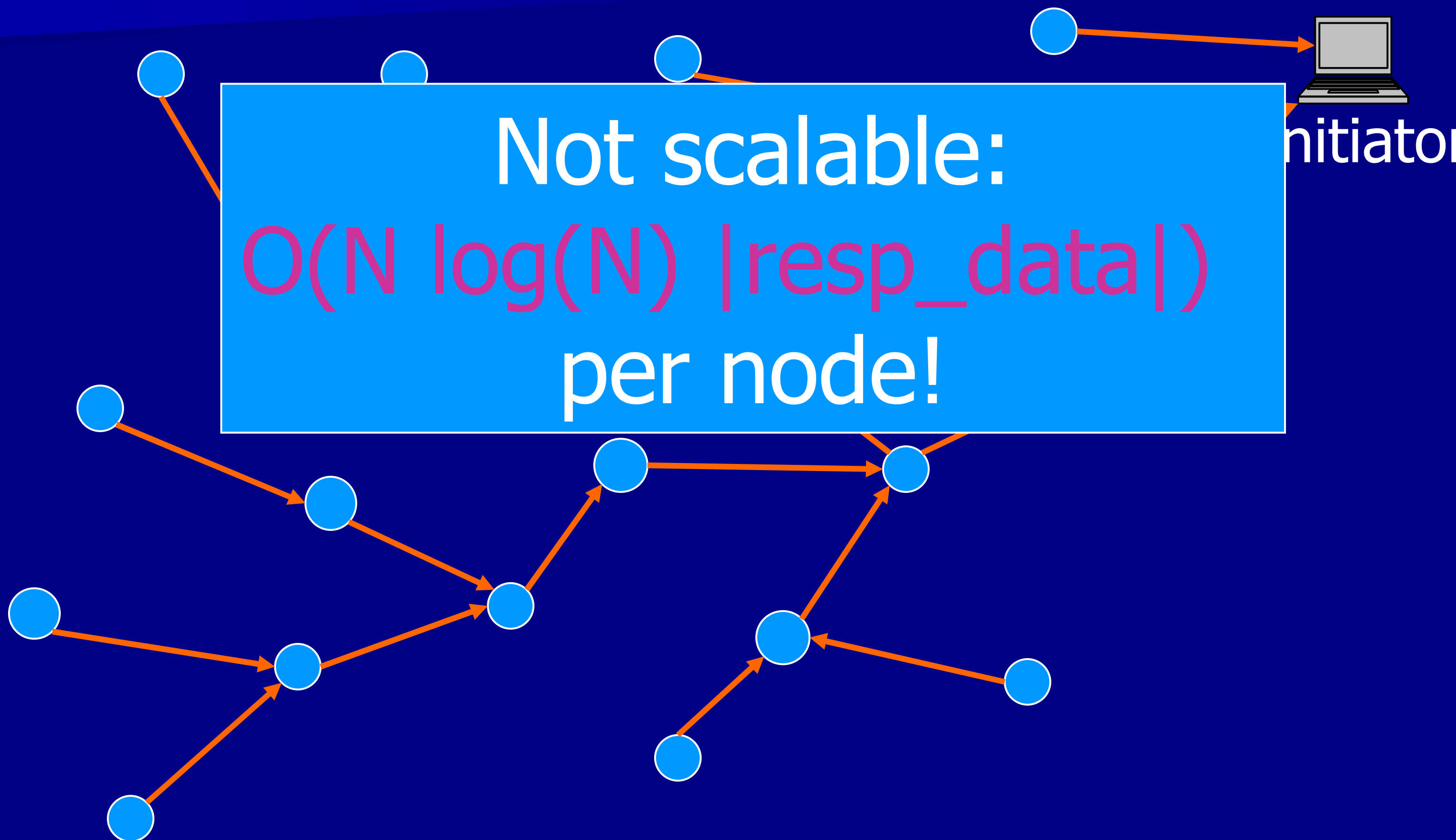
Accessing Quorums in Sensornets



Accessing Quorums in Sensornets



Accessing Quorums in Sensornets



Not scalable:

O(N log(N) |resp_data|)

per node!

Communication Complexity of QS

- #bits transmitted in one quorum access
  - Does not depend on the access pattern

	P-to-p (threshold)	Gossip (threshold)	Best that was known
Total	$\Theta(N\log(N))$	$O(N^2\log(N))$	$\Theta(\sqrt{N}\log(N))$
Per-node	$\Theta(\log(N))$	$O(N\log(N))$	Gossip: $\Theta(\sqrt{N}\log(N))$ p-to-p: $\Theta(\log(N))$

$|\text{req\_data}| = |\text{resp\_data}| = O(\log(n))$

Low Bandwidth Quorum Access [4]

- Idea: Use probabilistic sampling to achieve polylog communication complexity
  - Use gossip (flooding) for robustness

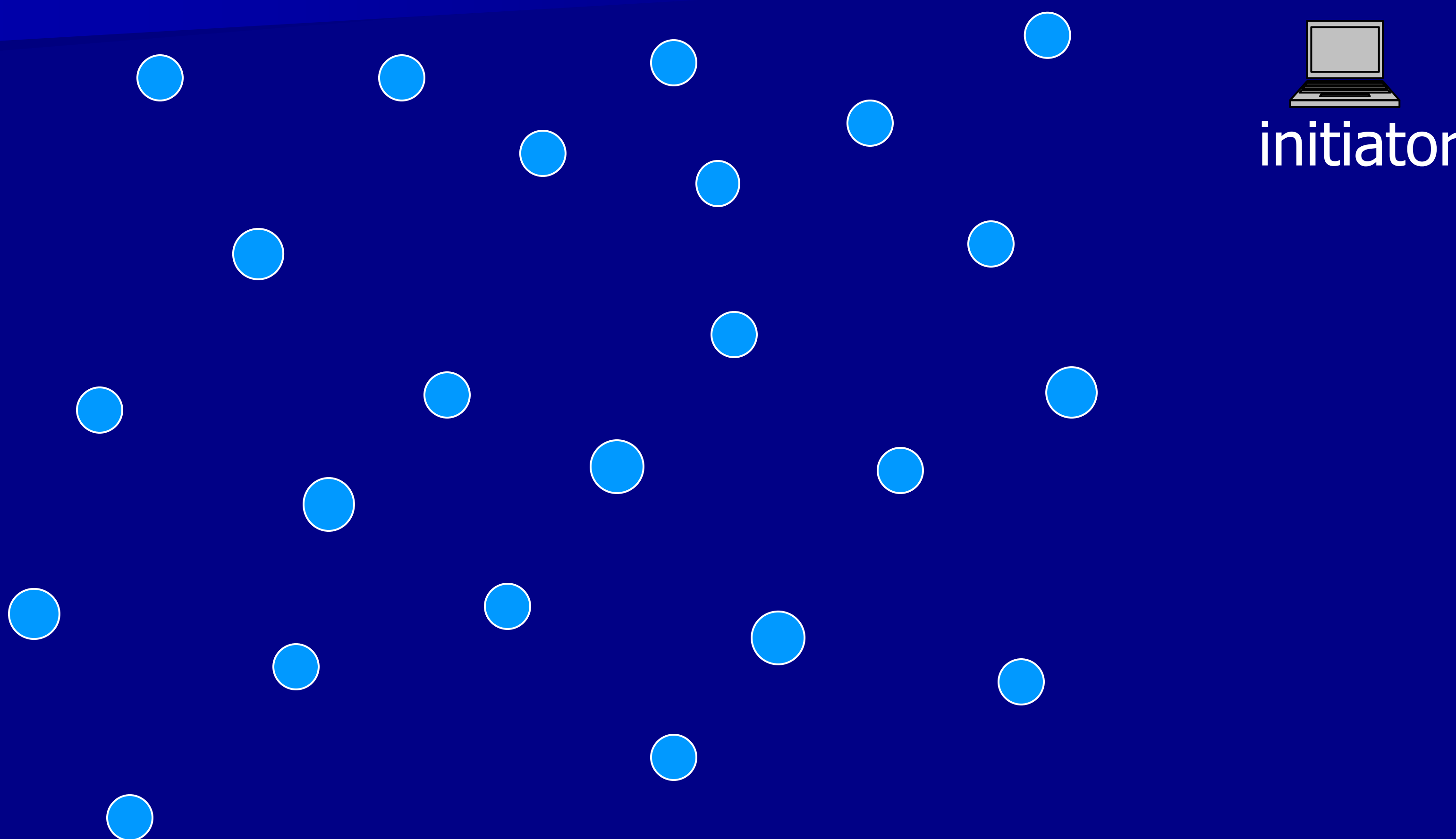
Sampling-Based Quorum Access [4]

- Initiator:

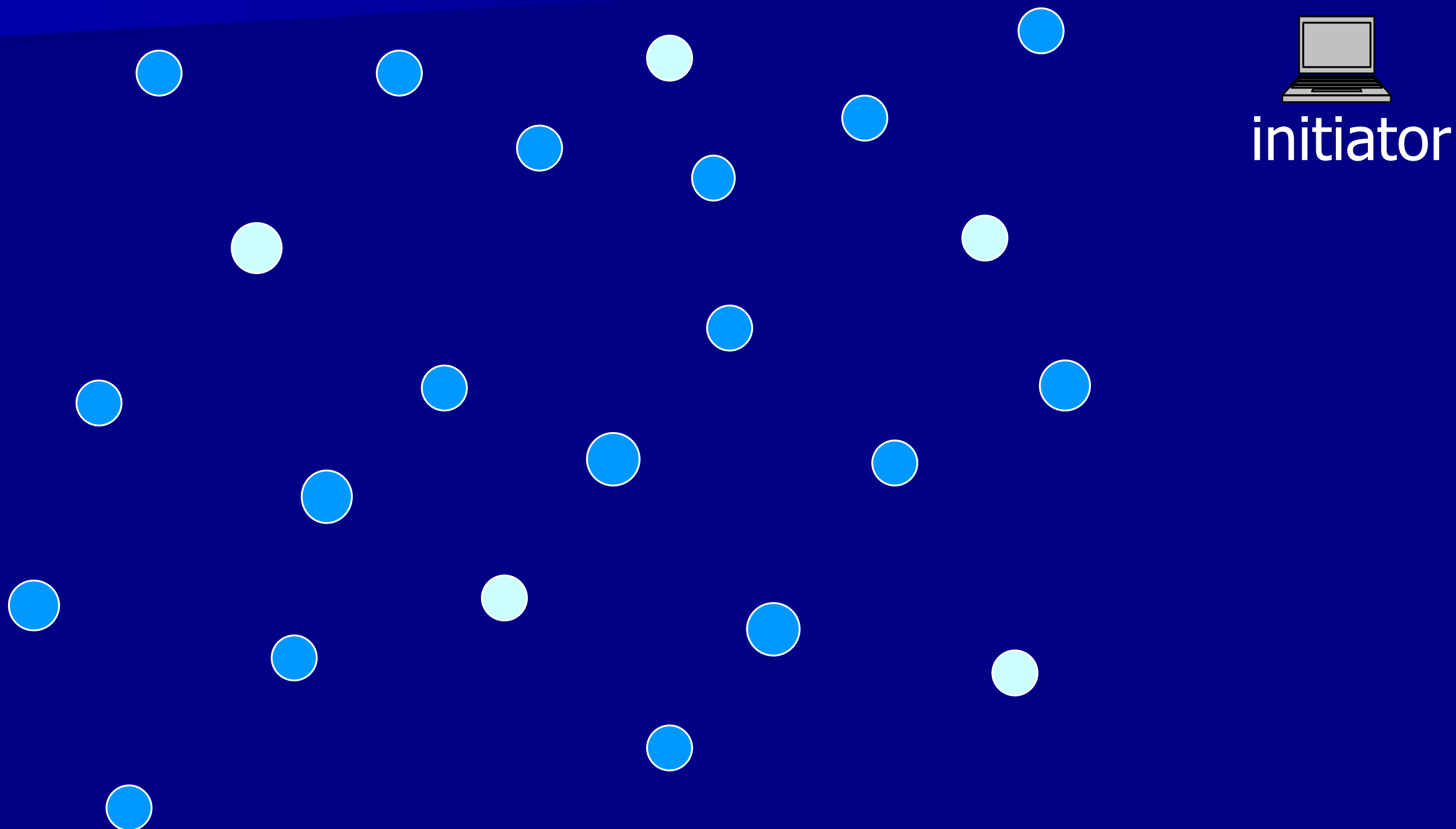
  (1) X:=Θ(c log(N))-sized sample of nodes chosen U.A.R.

  (2) Gossip <X,req_data>
- Everybody forwards <X,req_data>
- Node p:
  - If p in X, gossip back <X,resp_data>
- Everybody forwards <X,resp_data>

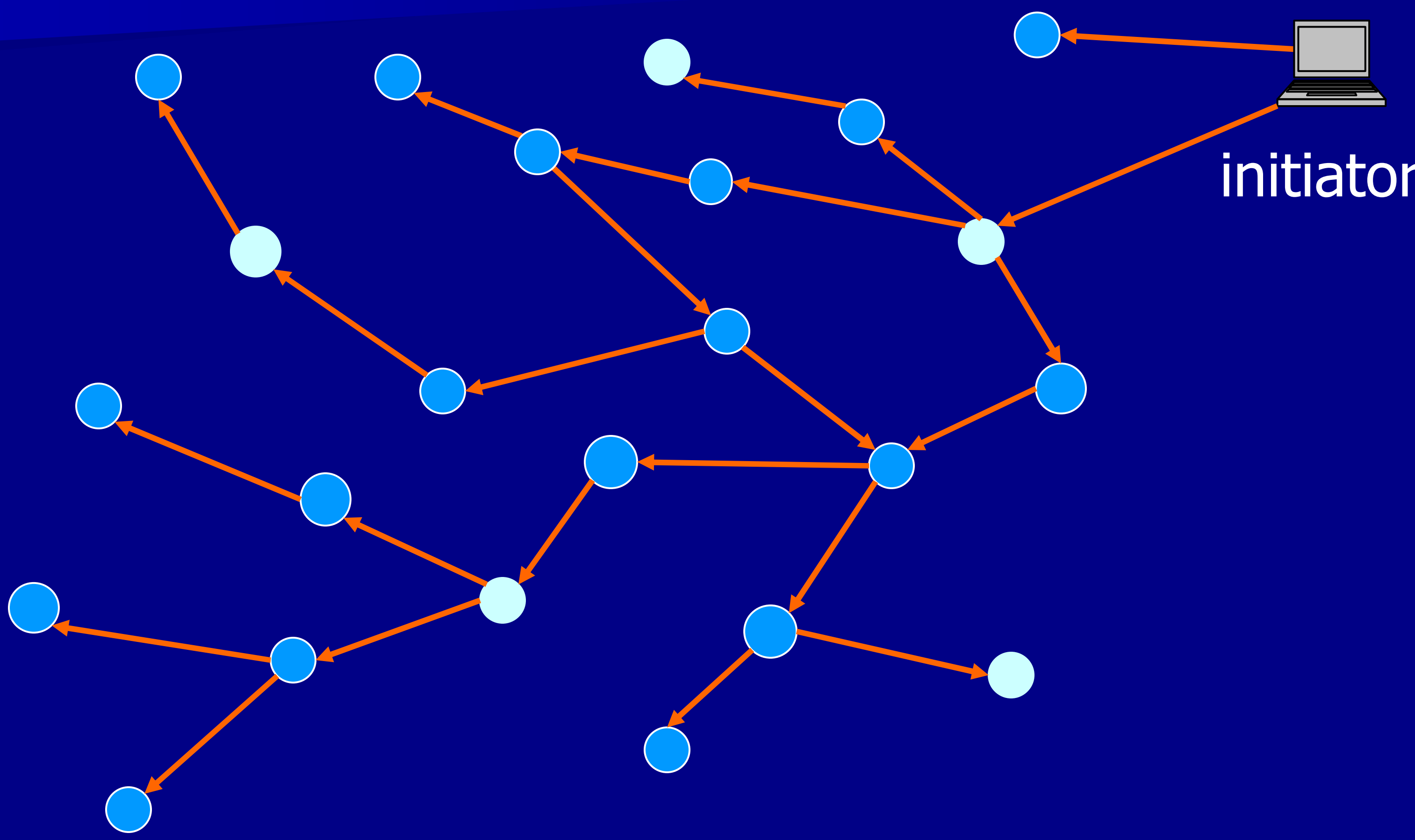
Accessing Quorums with Sampling



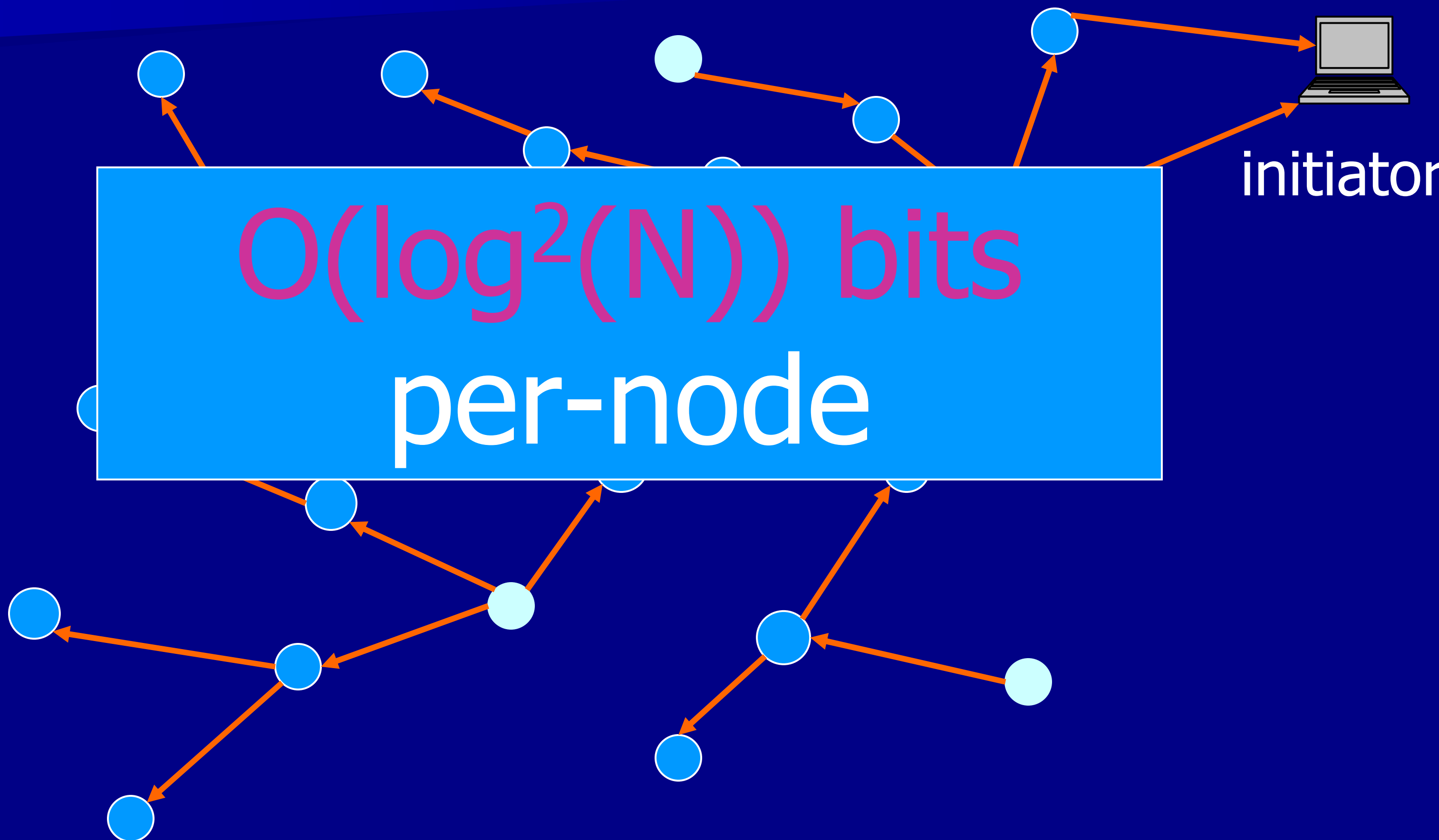
Accessing Quorums with Sampling



Accessing Quorums with Sampling



Accessing Quorums with Sampling



$O(\log^2(N))$ bits

per-node

Why this works?

- Lemma: If X is a sample of size $\Theta(c \log(N))$ chosen U.A.R, then % of nodes that receive the request is the same in both X and the entire population w.h.p.
- Proof: Follows from a Chernoff bound
  - See the paper for details

Updates vs. Queries

- Update: Ensure that enough nodes got the data though only a log-sized sample responds
  - The protocol described thus far
- Query: Ensure that the sample "hits" some updated nodes
  - Using samples of size $\Theta(c \log(N))$ guarantees intersection w.h.p.
  - Proof: Chernoff bound and union bound

Adding Fault Tolerance

- Assume a fraction p of nodes can crash or disconnect
- Modify the access protocol so that only a fraction r of nodes in X is required to respond
  - In the paper: $p<0.25$, $r=0.6$
  - p can be made asymptotically close to 0.5

The Initiator Protocol

$\mathsf{Update}(value)$

$sample \leftarrow \mathsf{Random}(S, r)$

$responses \leftarrow \emptyset$

while $|responses| < (1 - p - \tau)r$

  do $responses \leftarrow \mathsf{Gossip}(\mathsf{update}, sample, value)$

return

r=c log(N)

τ=(1-p)/5

$\mathsf{Query}()$

$sample \leftarrow \mathsf{Random}(S, r)$

$responses \leftarrow \emptyset$

while $|responses| < (1 - p - \tau)r$

  do $responses \leftarrow \mathsf{Gossip}(query, sample, \perp)$

return $responses$

r=c log(N)

τ=(1-p)/5

Quorum Systems Summary

- Low communication complexity is important for environments with scarce resources, such as sensor and ad hoc networks
- Probabilistic, sampling-based QS
  - polylog communication complexity
  - Available as long as ≥50% of nodes are alive and connected

Conclusions

- Middleware for fault-tolerant computing in realistic wireless ad hoc networks
- Low-level components
  - Collision detectors
  - Contention manager
  - Round synchronizer
  - Reliable broadcast
  - Quorums
- High-level components
  - Virtual nodes and state machines
  - Local agreement

Future Work

- Malicious failures
- Weakest collision detector for agreement
- Implementing collision detectors
- More efficient/resilient implementations
- Implementations in real networks
- Applications

References

[1] G. Chockler, M. Demirbas, S. Gilbert, and C. Newport. **A Middleware Framework for Robust Applications in Wireless Ad Hoc Networks.** *Proceeding of the 43rd Allerton Conference on Communication, Control, and Computing*, September, 2005

[2]G. Chockler, M. Demirbas, S. Gilbert, C. Newport, and T. Nolte. **Consensus and Collision Detectors in Wireless Ad Hoc Networks**. *24th Annual Symposium on the Principles of Distributed Computing (PODC)*, July, 2005

[3] G. Chockler, M. Demirbas, S. Gilbert, N. Lynch, C. Newport, and T. Nolte. **Reconciling the Theory and Practice of UnReliable Wireless Broadcast.** *International Workshop on Assurance in Distributed Systems and Networks (ADSN)*, June, 2005

[4] G. Chockler, S. Gilbert, and B. Patt-Shamir. **Communication-Efficient Probabilistic Quorum Systems.** *Proceedings of the International Workshop on Foundations and Algorithms for Wireless Networking (FAWN)*, March, 2006.

[5] S. Dolev, S. Gilbert, L. Lahiani, N. Lynch, and T. Nolte. **Timed Virtual Stationary Automata for Mobile Networks.** *9th International Conference on Principles of Distributed Systems (OPODIS)*, December, 2005

[6] S. Dolev, S. Gilbert, N. Lynch, A. Shvartsman, and J. Welch. **GeoQuorums: Implementing Atomic Memory in Mobile Ad Hoc Networks.** *Distributed Computing*, 125–155, November, 2005

[7] S. Dolev, S. Gilbert, N. Lynch, E. Schiller, A. Shvartsman, and J. Welch. **Virtual Mobile Nodes for Mobile Adhoc Networks.** *Proceeding of the 18th International Conference on Distributed Computing (DISC)*, October, 2004.

[8] L. Lamport. **Time, clocks, and the ordering of events in a distributed system**. Communications of the ACM: 21(7), 1978

URLs:

Virtual Nodes: http://theory.lcs.mit.edu/~sethg/biblio-projects.html#vi

Fault-tolerance middleware: http://theory.lcs.mit.edu/~sethg/biblio-projects.html#consensus

Thank You !