SINGLEHOP COORDINATION &COLLABORATION PRIMITIVES FOR WSANS

Murat Demirbas
SUNY Buffalo
www.cse.buffalo.edu/ubicomp

My current projects

Singlehop collaboration and coordination framework for wireless sensor actor networks (NSF Career 2008)

Tool-support for producing high-assurance reliable software for WSANs (NSF CSR 2009)

Efficient and resilient querying and tracking services for WSNs (Office of Naval Research, 2009)

Crowdsourced sensing and collaboration using Twitter (Google Research Award, 2010)

My current projects

Singlehop collaboration and coordination framework for wireless sensor actor networks (NSF Career 2008)

Tool-support for producing high-assurance reliable software for WSANs (NSF CSR 2009)

Efficient and resilient querying and tracking services for WSNs (Office of Naval Research, 2009)

Crowdsourced sensing and collaboration using Twitter (Google Research Award, 2010)

Outline

Transact: Singlehop coordination framework for WSANs

Pollcast: Singlehop collaborative feedback collection

Causataxis: Coordinated locomotion of mobile WSNs

Outline

Transact: Singlehop coordination framework for WSANs

Pollcast: Singlehop collaborative feedback collection

Causataxis: Coordinated locomotion of mobile WSNs

Wireless sensor-actor networks (WSANs)

- Process control systems
  - vibration control of assembly lines
  - valve control
- Multi-robot cooperative control
  - robotic highway safety/construction markers
  - search & rescue operations
- Resource/task allocation
  - tracking of targets via distributed smart cameras

WSANs programming challenges

- Consistency & coordination are important
  - in contrast to WSNs, where eventual consistency & loose synchrony is sufficient for most applications and services

- Effective management of concurrent execution is needed
  - for consistency reasons concurrency needs to be tamed to prevent unintentional nondeterministic executions
  - on the other hand, for real-time guarantees concurrency needs to be boosted to achieve timeliness

Transact: A transactional framework for WSANs

- Transact eliminates unintentional nondeterministic executions while retaining the concurrency of executions
  - Conflict serializability: any property proven for the single threaded coarse-grain executions of the system is a property of the concurrent fine-grain executions of the system

- Transact enables ease of programming for WSANs
  - Transact introduces a novel "consistent write-all" paradigm that enables a node to update the state of its neighbors in a consistent and simultaneous manner

Transact programs

- bool leader_election(){

  X=read("*.leader");

  if (X={}) then return write-all("*.leader="+ID);

  return FAILURE;}

- bool resource_allocation(){

  X=read("*.allocation");

  Y=select a subset of non-allocated members in X;

  return write-all("Y.allocation="+ID);}

Challenges & Opportunities

- In contrast to database systems, in distributed WSANs there is no central database repository or arbiter
  - the control and sensor variables, on which the transactions operate, are maintained distributedly over several nodes
- Broadcast communication opens novel ways for optimizing the implementation of read and write operations
  - A broadcast is received by the recipients **atomically**
    This enables us to order transactions, and synchronize nodes to build a structured transaction operation
  - Broadcast allows **snooping**
    This enables us to detect conflicts in a decentralized manner

Overview of Transact

- Optimistic concurrency control (OCC) idea
  - Read & write-tentatively, Validate, Commit

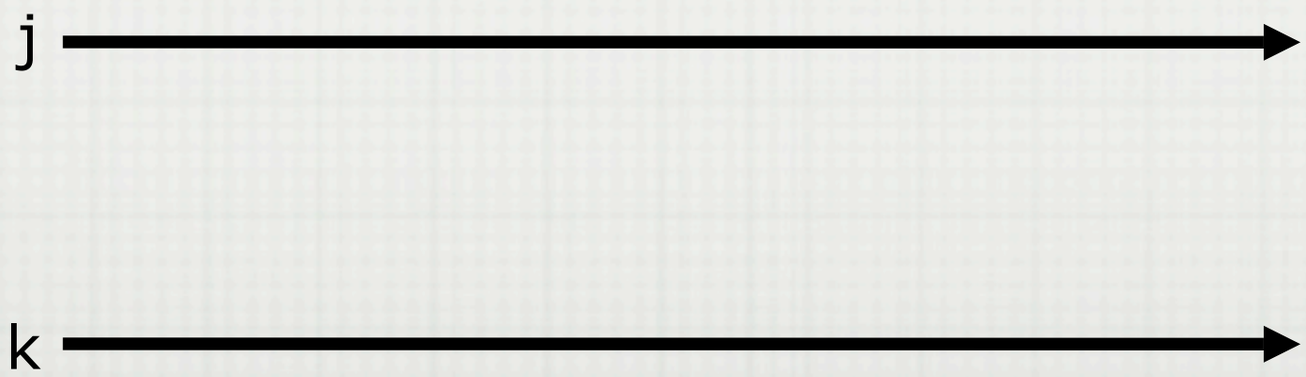
- In Transact, a transaction is structured as (read, write-all)
  - Read operation reads variables from some nodes in singlehop, and write-all operation writes to variables of a set of nodes in singlehop
  - Read operations are always compatible with each other: since reads do not change the state, it is allowable to swap the order of reads

Overview of Transact...

- A write-all operation may fail to complete when a conflict with another transaction is reported
  - Since the write-all operation is placed at the end of the transaction, if write-all fails no state is changed (no side effects !). An aborted transaction can be retried later
- If there are no conflicts reported, write-all succeeds by updating the state of the nodes in a consistent and observably-simultaneous manner

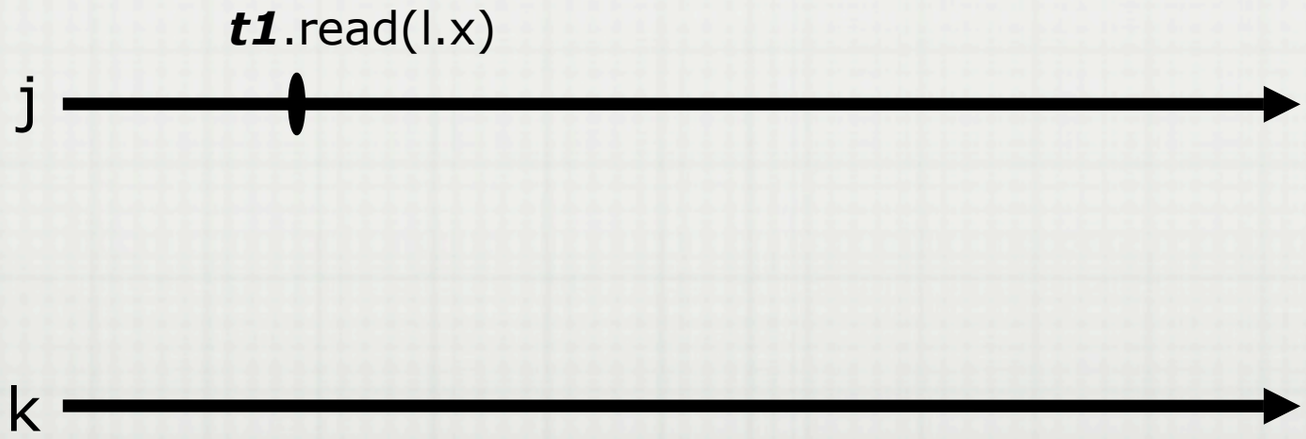
Conflicting transactions

- Any two transactions t1 and t2 are conflicting iff
  - a read-write incompatibility introduces a causality from t1 to t2
  - a write-write or read-write incompatibility introduces a causality from t2 to t1



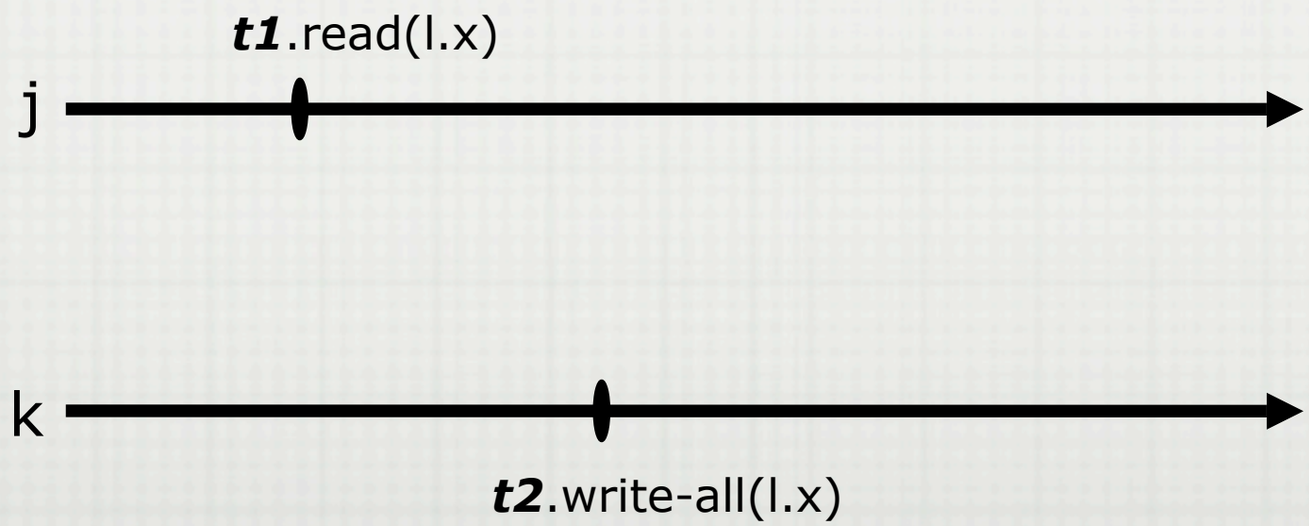
Conflicting transactions

- Any two transactions t1 and t2 are conflicting iff
  - a read-write incompatibility introduces a causality from t1 to t2
  - a write-write or read-write incompatibility introduces a causality from t2 to t1



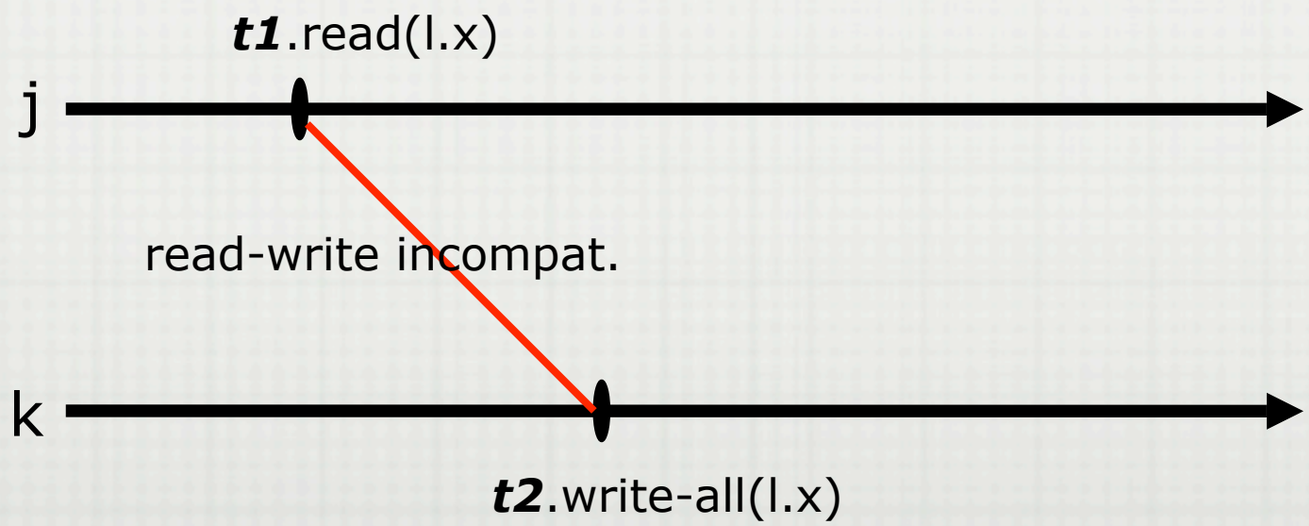
Conflicting transactions

- Any two transactions t1 and t2 are conflicting iff
  - a read-write incompatibility introduces a causality from t1 to t2
  - a write-write or read-write incompatibility introduces a causality from t2 to t1



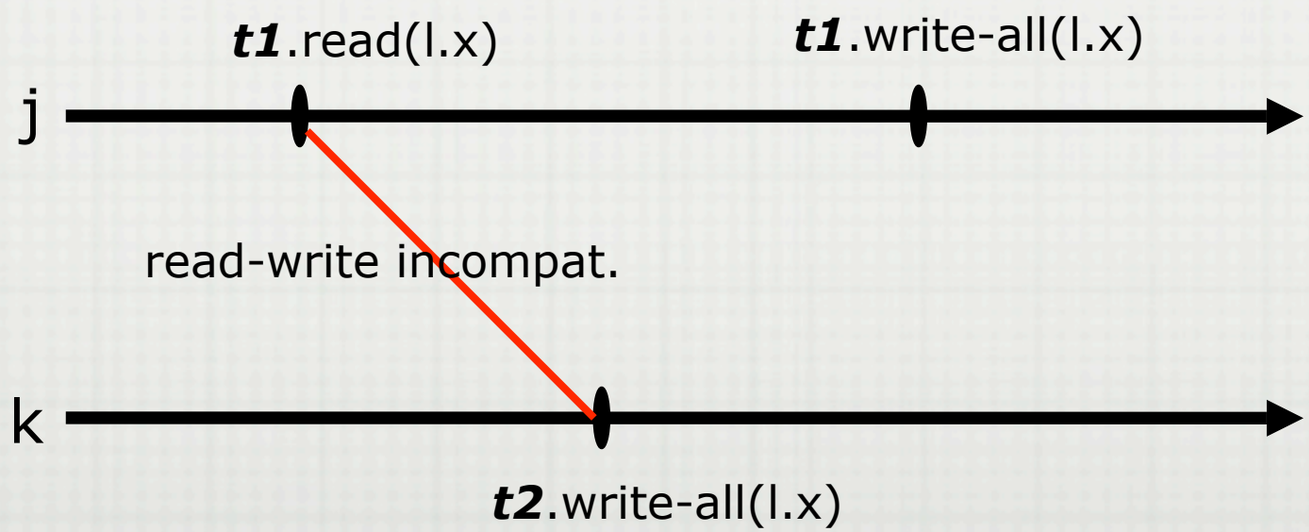
Conflicting transactions

- Any two transactions t1 and t2 are conflicting iff
  - a read-write incompatibility introduces a causality from t1 to t2
  - a write-write or read-write incompatibility introduces a causality from t2 to t1



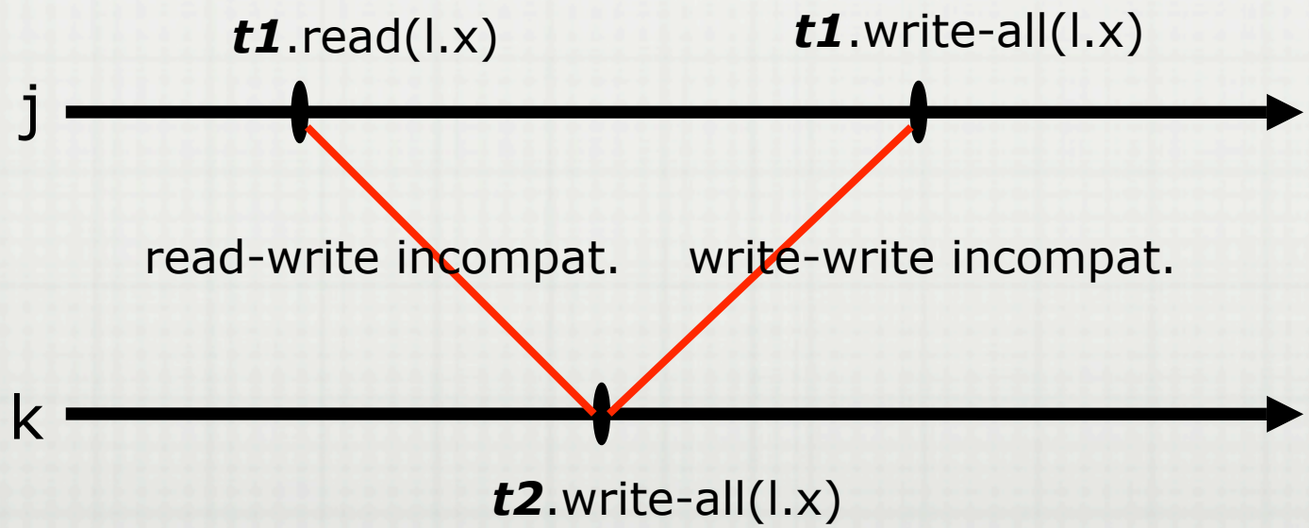
Conflicting transactions

- Any two transactions t1 and t2 are conflicting iff
  - a read-write incompatibility introduces a causality from t1 to t2
  - a write-write or read-write incompatibility introduces a causality from t2 to t1

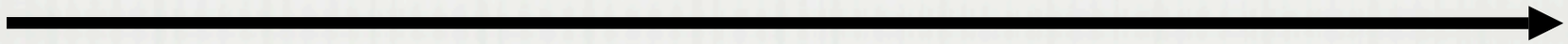


Conflicting transactions

- Any two transactions t1 and t2 are conflicting iff
  - a read-write incompatibility introduces a causality from t1 to t2
  - a write-write or read-write incompatibility introduces a causality from t2 to t1

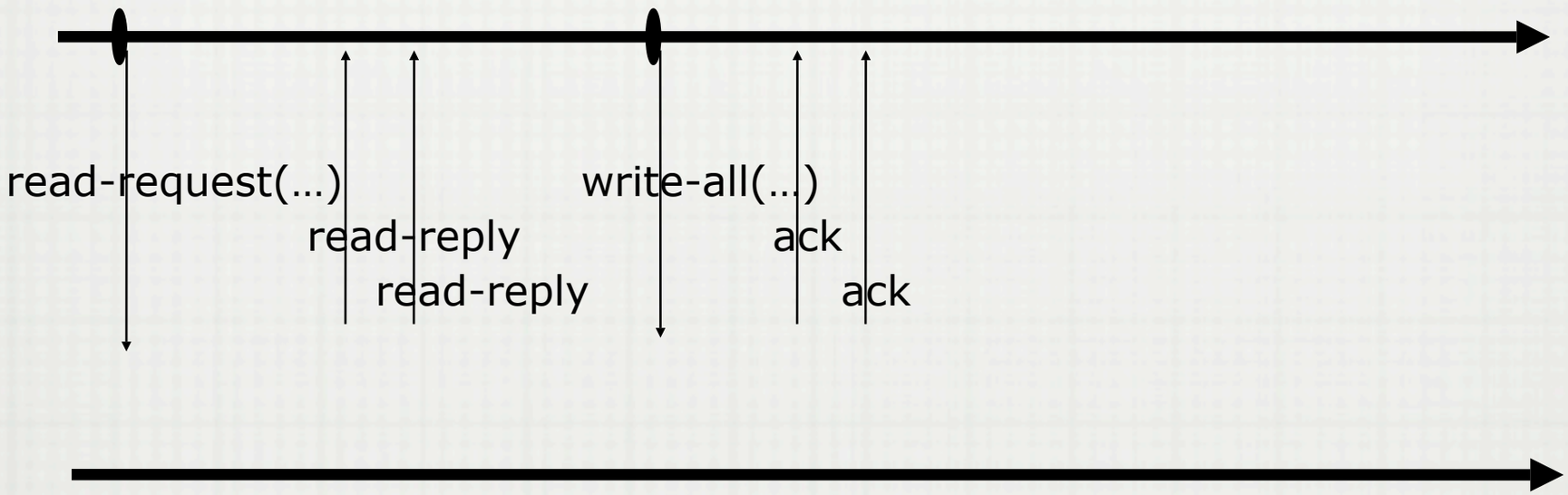


Timeline of a transaction

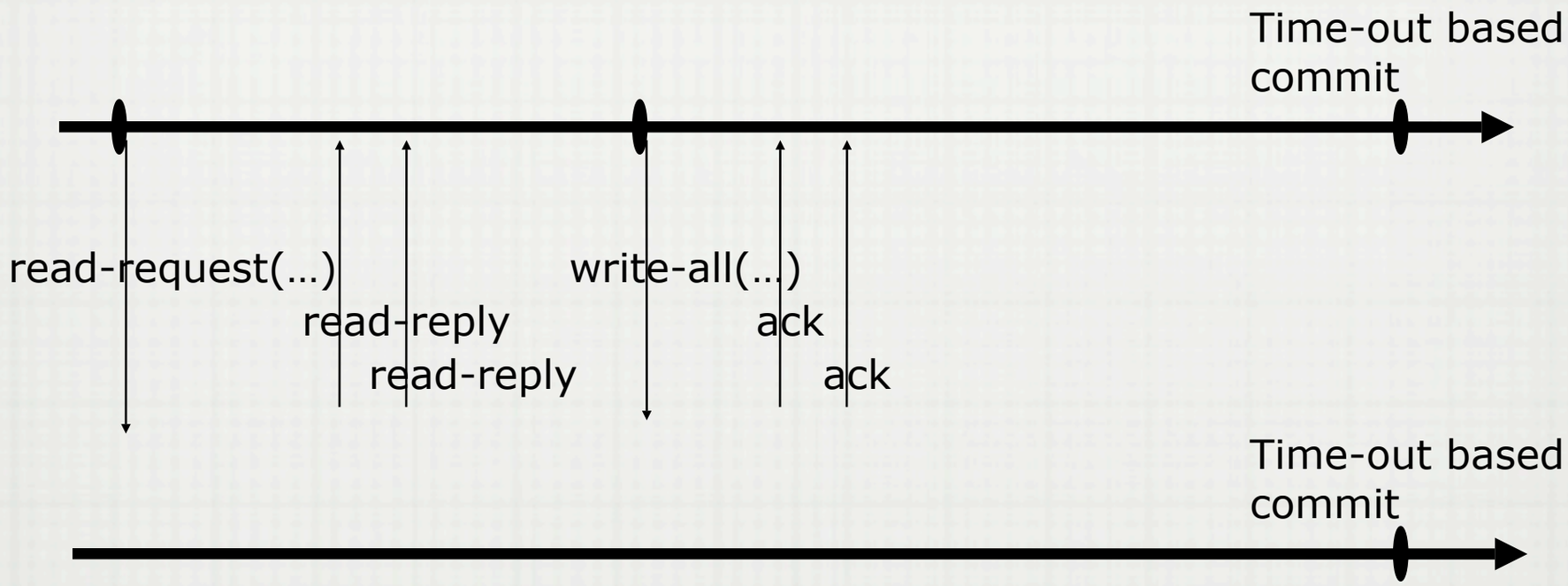


Timeline of a transaction

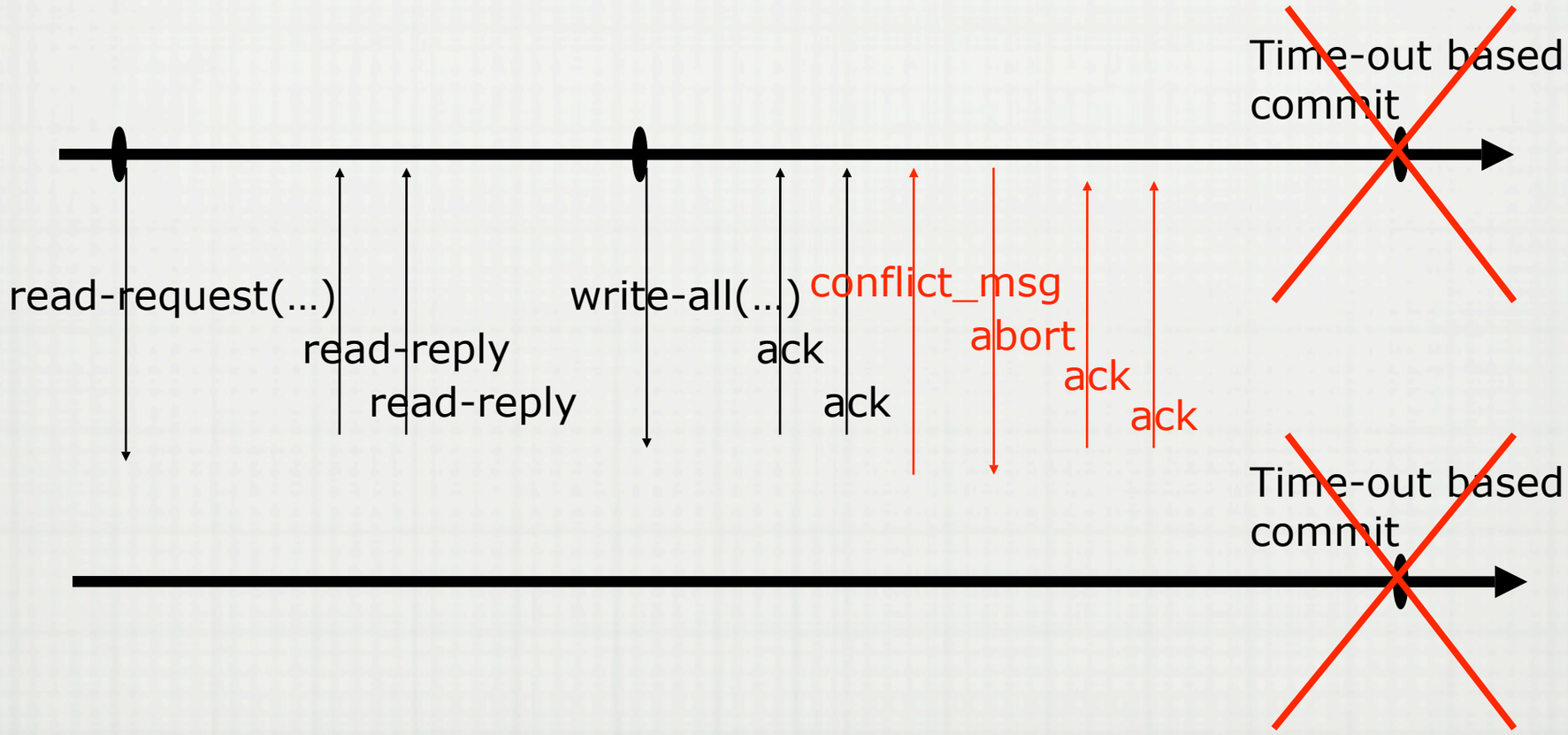
Timeline of a transaction



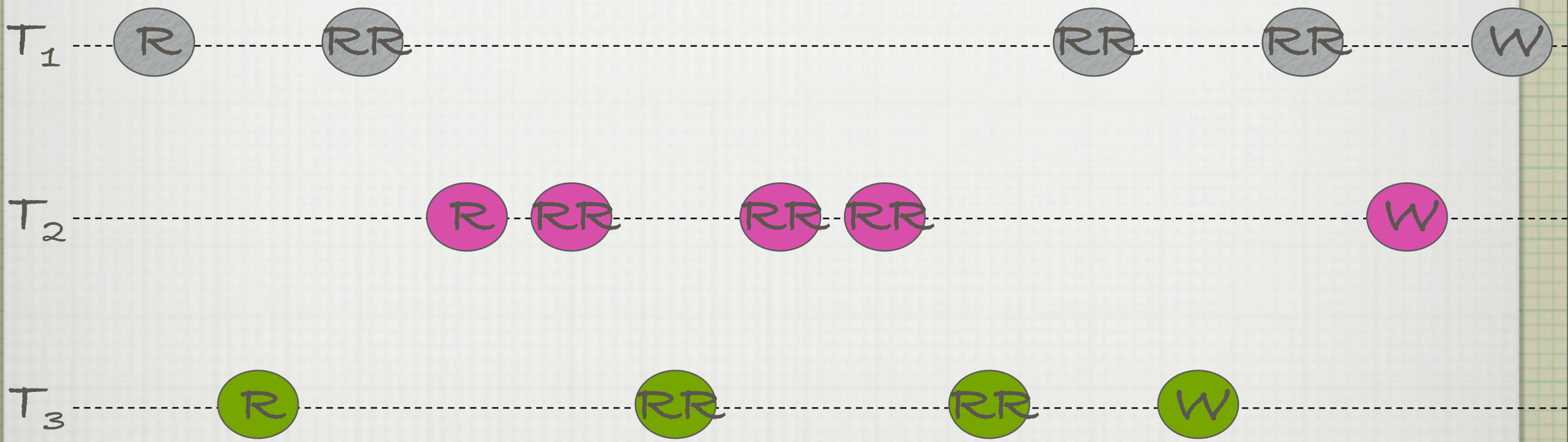
Timeline of a transaction



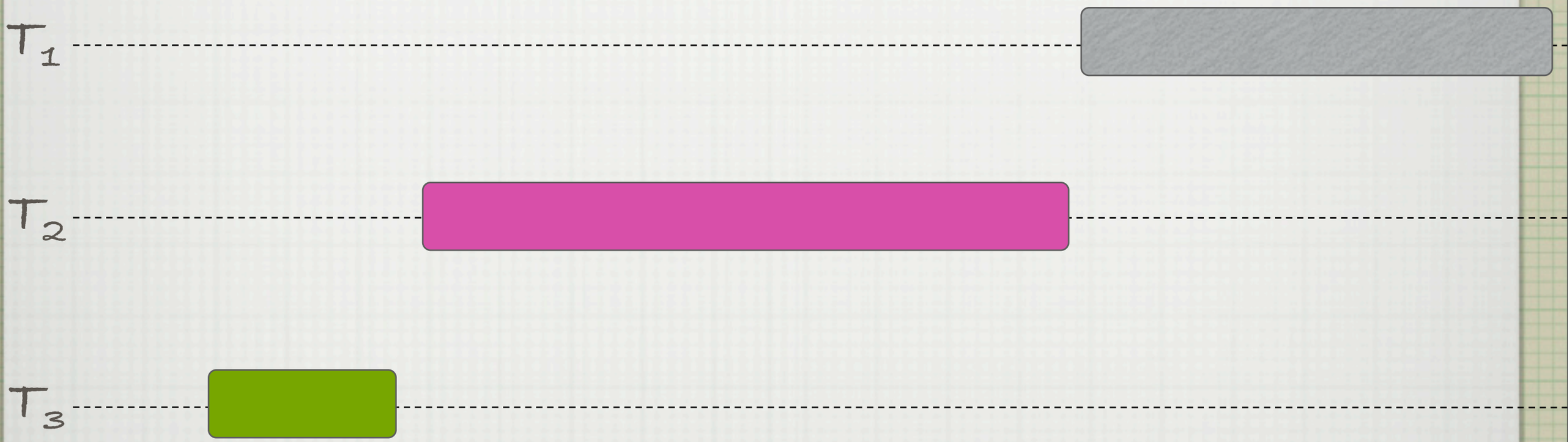
Timeline of a transaction



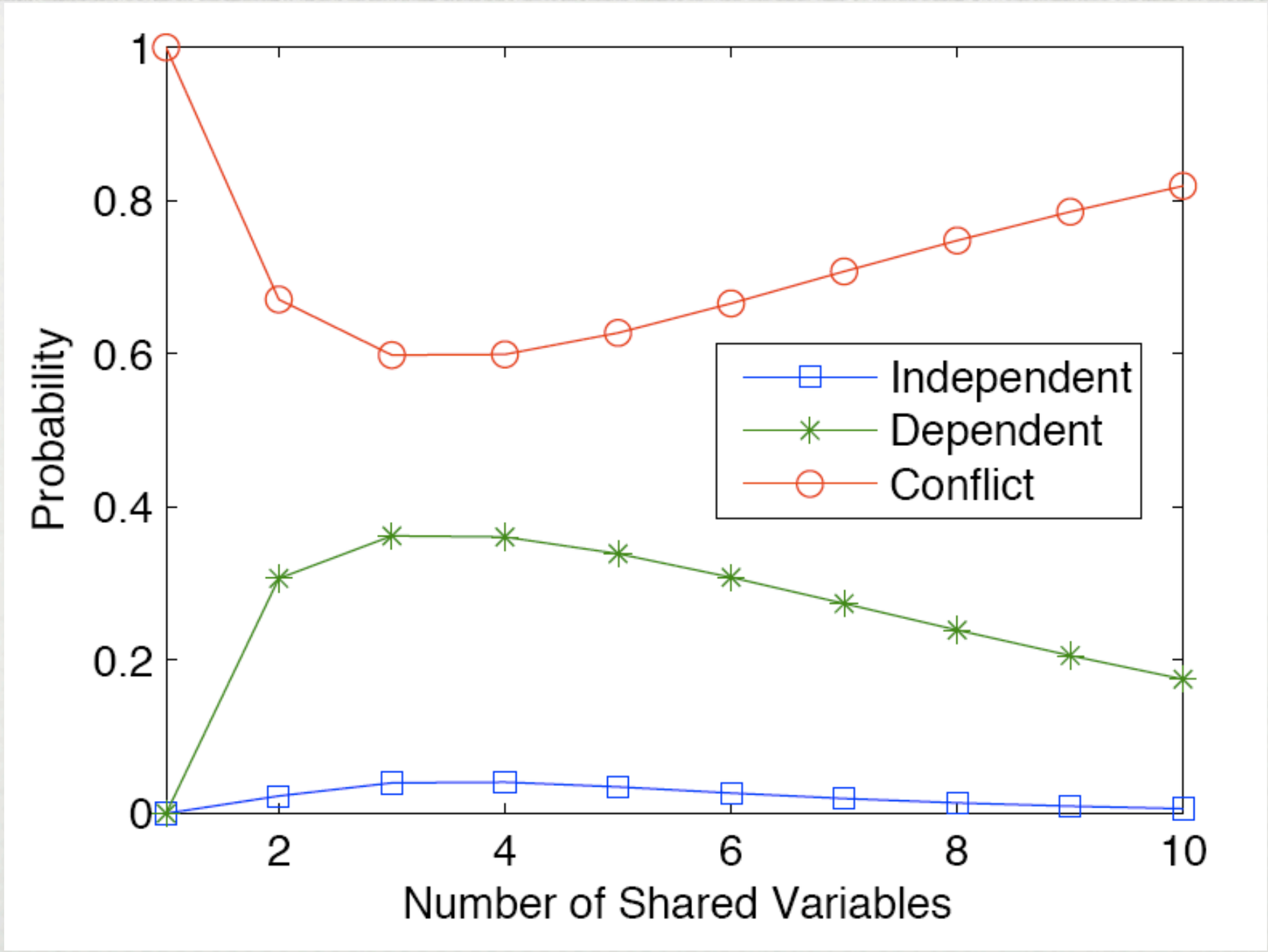
Concurrent



Serializable results

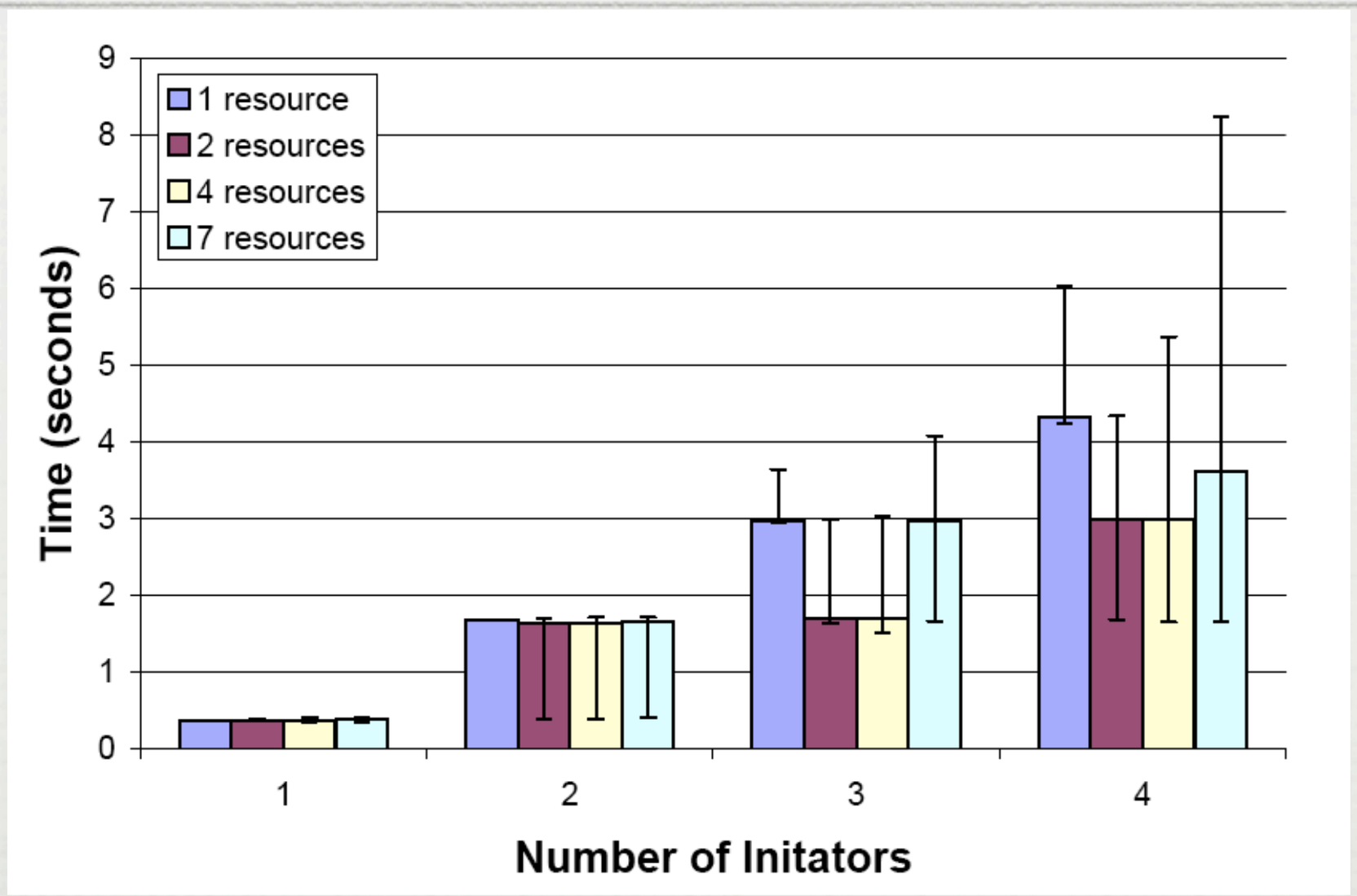


Analytical results on resource/task allocation problem



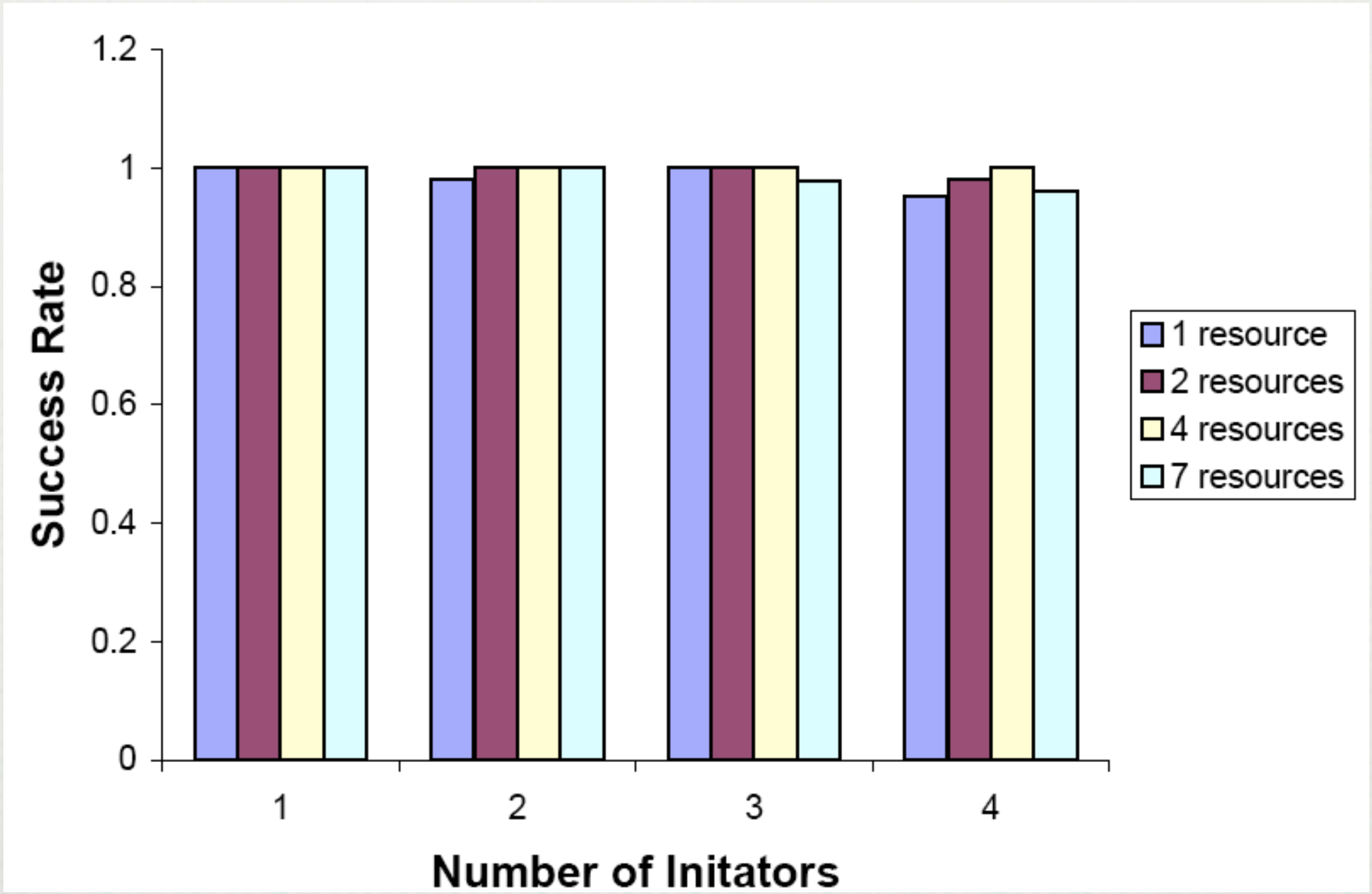
- analysis for two initiators

Implementation results



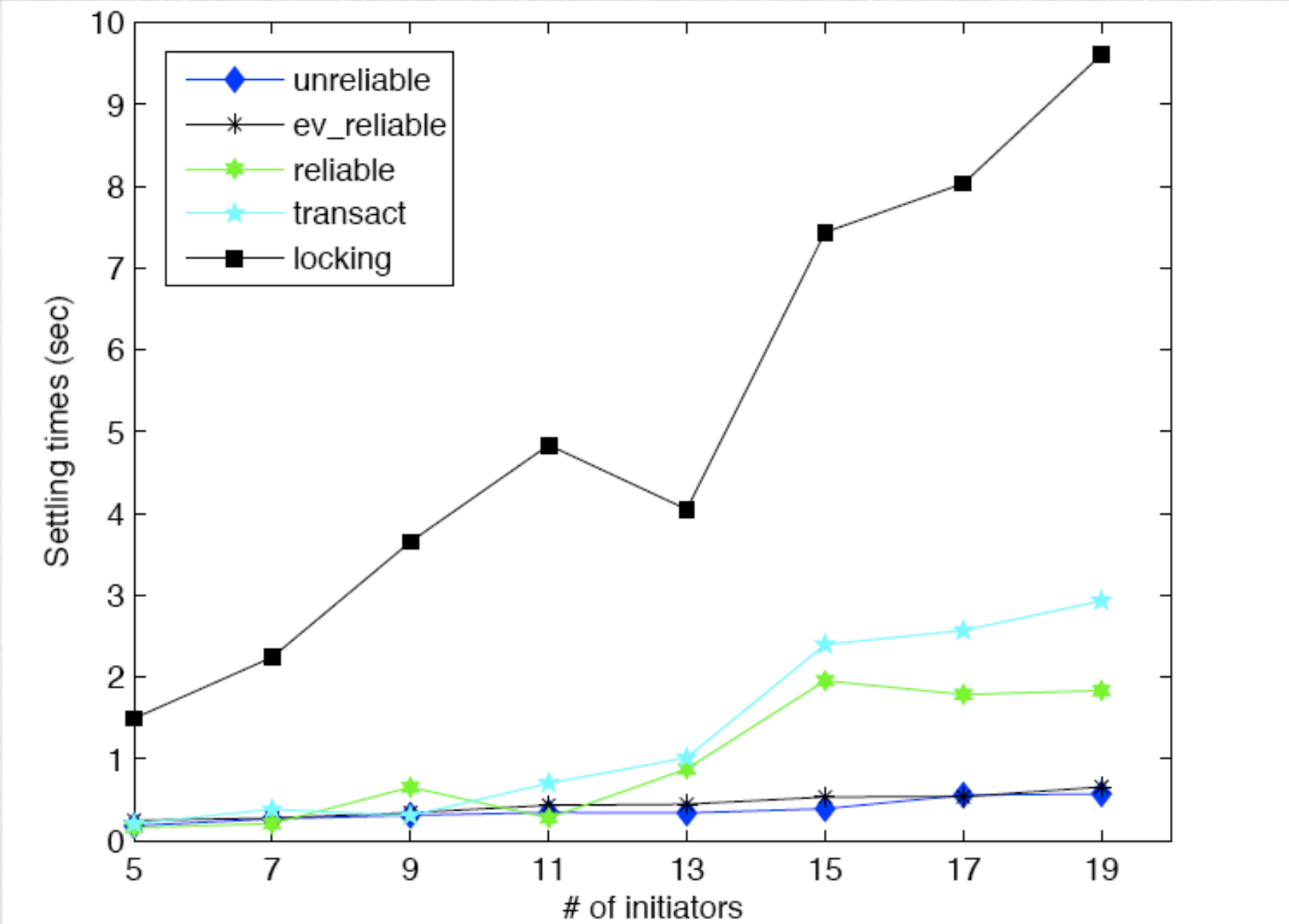
- 11 Tmotes, upto 4 initiators and 7 resources
- Settling time graph when stress-testing the application

Implementation results…



- Consistency results

Simulation results



- 10-by-10 network, Prowler simulation

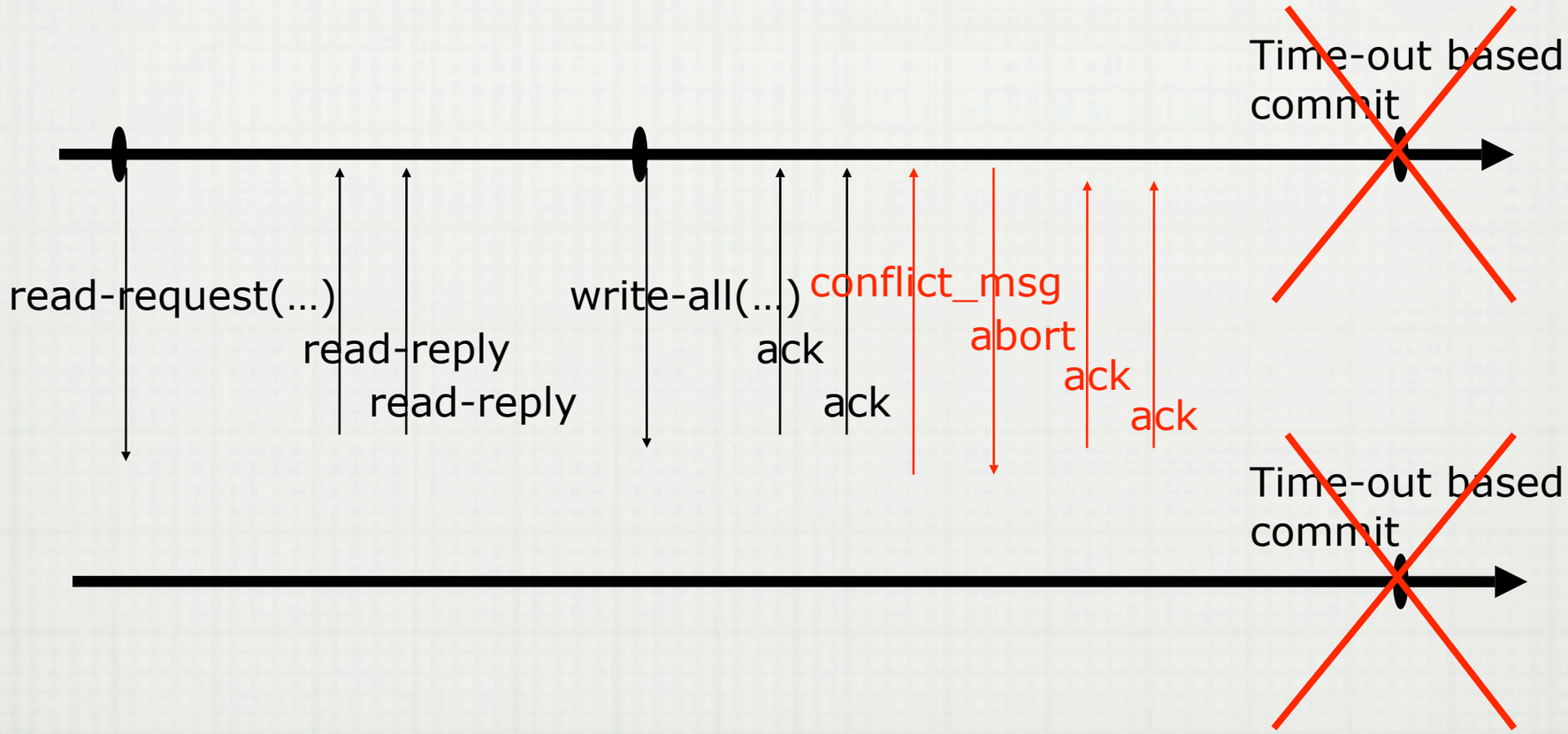
Weaknesses of 1st implementation

- Persistent message losses violates safety
  - Initiator may not get all nodes to abort before time-triggered commit
  - We cannot achieve progress due to impossibility of coordinated attack, when some messages start getting through we achieve progress
- Initiator failure after write-all message violates safety
  - This reduces to the above problematic scenario for message loss
  - We cannot achieve progress under fully-asynchronous model due to FLP; with timeout assumptions it is possible to complete transaction

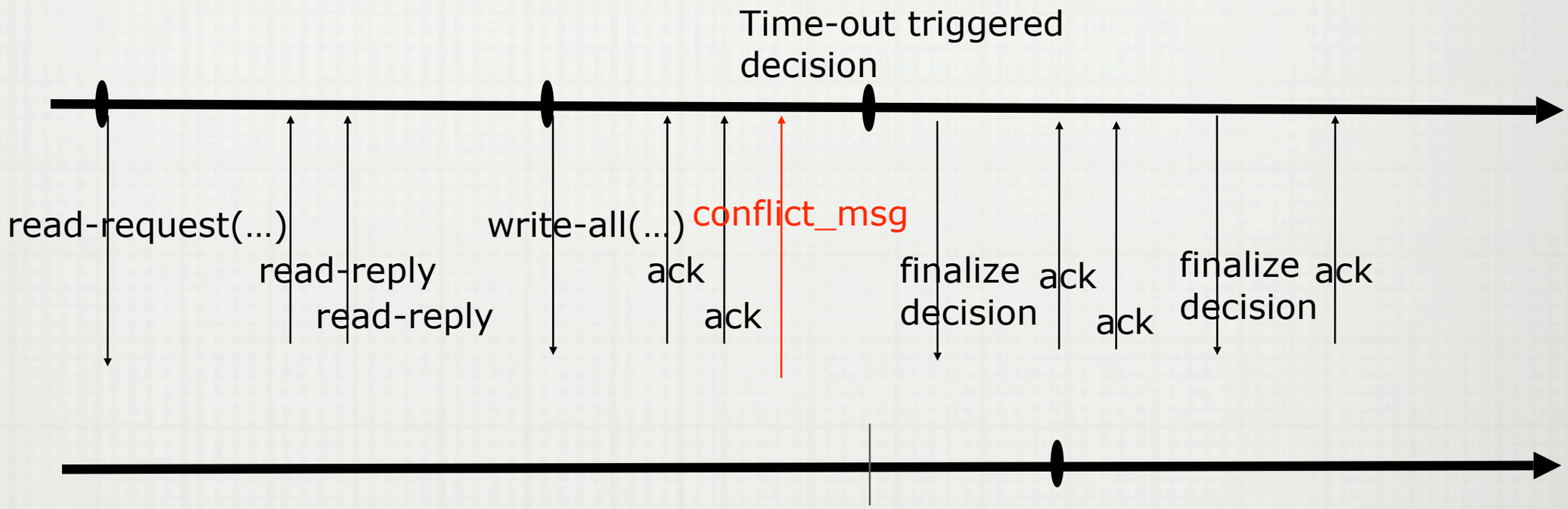
Fault-tolerant implementation

- We use 2-phase commit to combat faults
- Initiator aborts if
  - some read-replies are missing; no need to notify participants
  - some write-all acks are missing; notify participants by abort-msg until ack is received from all
- Commit is also explicit
  - Initiator repeats commits until ack is received from all

Timeline of 1st implementation



Timeline of 2nd implementation



Guaranteed conflict detection

- Conflict detection was best effort in the 1st implementation; what if 2+ transactions did not intersect at common node that could detect the cycle?

- Use a single moderator that is aware of all existing transactions; this moderator is included as dummy participant in all transactions

- Moderator death does not violate safety, a new moderator needs to be chosen via consensus for progress

  - Or we can have masking fault-tolerance by using Paxos for implementing the moderator via replicated state machines

Extensions: A lightweight alternative to Transact

- RAWS: Read-All, Write-Self
- Initiator can read from all neighbors during read, but can only write to itself
- Much faster but less expressive
- Consensus among n nodes requires at least n transactions in RAWS, whereas one Transact transaction is enough for this

Extensions: Building multihop programs

- Transact can be used for efficient realizations of high-level programming abstractions, Linda & virtual node(VN)

- In Linda, coordination among nodes is achieved through **in, out** operations using which tuples can be added to or retrieved from a tuplespace shared among nodes

  - Transact can maintain the reliability and consistency of the shared tuplespace to the face of concurrent execution

- VN provides stability & robustness in spite of mobility of nodes

  - Transact can implement VN abstraction under realistic WSN settings

Transact recap

- Transact is a transactional programming framework for WSANs
  - provides ease of programming and reasoning in WSANs without curbing the concurrency of execution
  - facilitates achieving consistency and coordination via the consistent write-all primitive
- Future work
  - Verification support: Transact already provides conflict serializability, the burden on the verifier is significantly reduced
  - Transact patterns: programmers can adapt commonly occurring patterns for faster development

Outline

Transact: Singlehop coordination framework for WSANs

Pollcast: Singlehop collaborative feedback collection

Causataxis: Coordinated locomotion of mobile WSNs

Lightweight singlehop collaboration

- The idea is to use receiver-side collision detection (RCD) for lightweight singlehop collaborative feedback

- Pollcast: Does **P** hold for the neighborhood?
  - Initiator performs binary probing instead of full-fledged reads
  - Nodes where **P** holds answer simultaneously
  - Initiator uses RCD to detect whether there are answers

Applications

- In-network processing
  - Data aggregation / Filtering
  - False positive elimination
- Local decision making
  - Barrier synchronization
  - Resource allocation

Challenges for achieving RCD

- ☐ Not directly supported by hardware (CC2420)
- ☐ Shadowing
- ☐ RSSI-based RCD requires extra processing, unreliable
- ☐ CRC based collision detection only works when preamble and packet frames are received

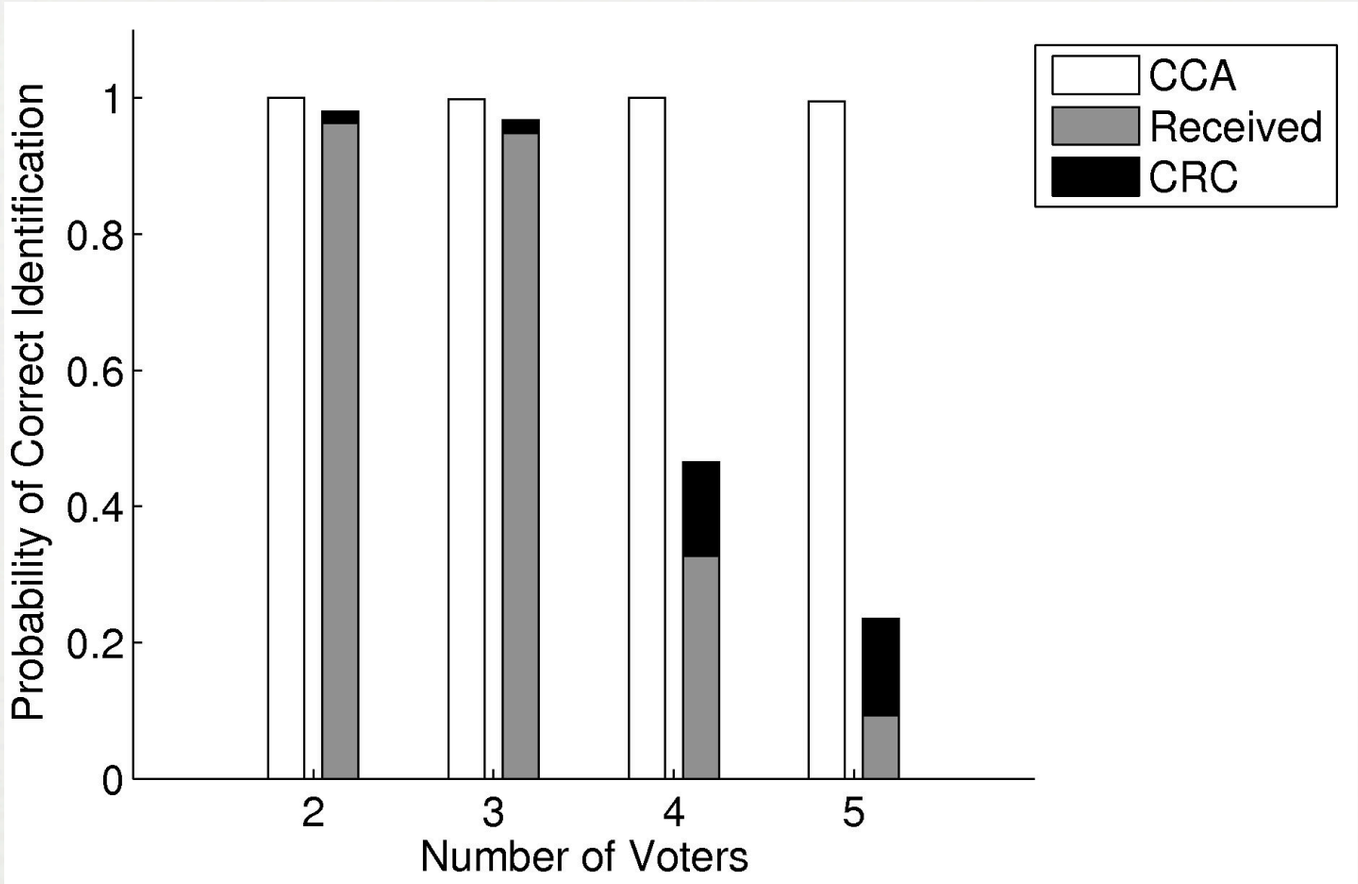
RCD implementations

- Clear Channel Assessment (CCA): Achieved by tricking the radio to execute CCA, and then not sending the message
  - Demirbas, Soysal, Hussein (Infocom 2008)
- Backcast: Achieved by exploiting the non-destructive collisions of simultaneous & identical Hardware-ACKs (HACKs) at the receiver-side
  - Dutta, Musaloiu-E, Stoica, Terzis (HotNets 2008)

Pollcast Implementation

- Poller:
  - Queries a predicate P in a set of voters using PollP message with CSMA
  - Then switches to RCD; Collision means P is true in at least one voter
- Voter:
  - If predicate P is true, immediately sends a VoteP message
  - VoteP message is sent without CSMA

RCD Experiments



- Singlehop environment
- All motes in 10m radius with clear line of sight
- 200 trials for each data point

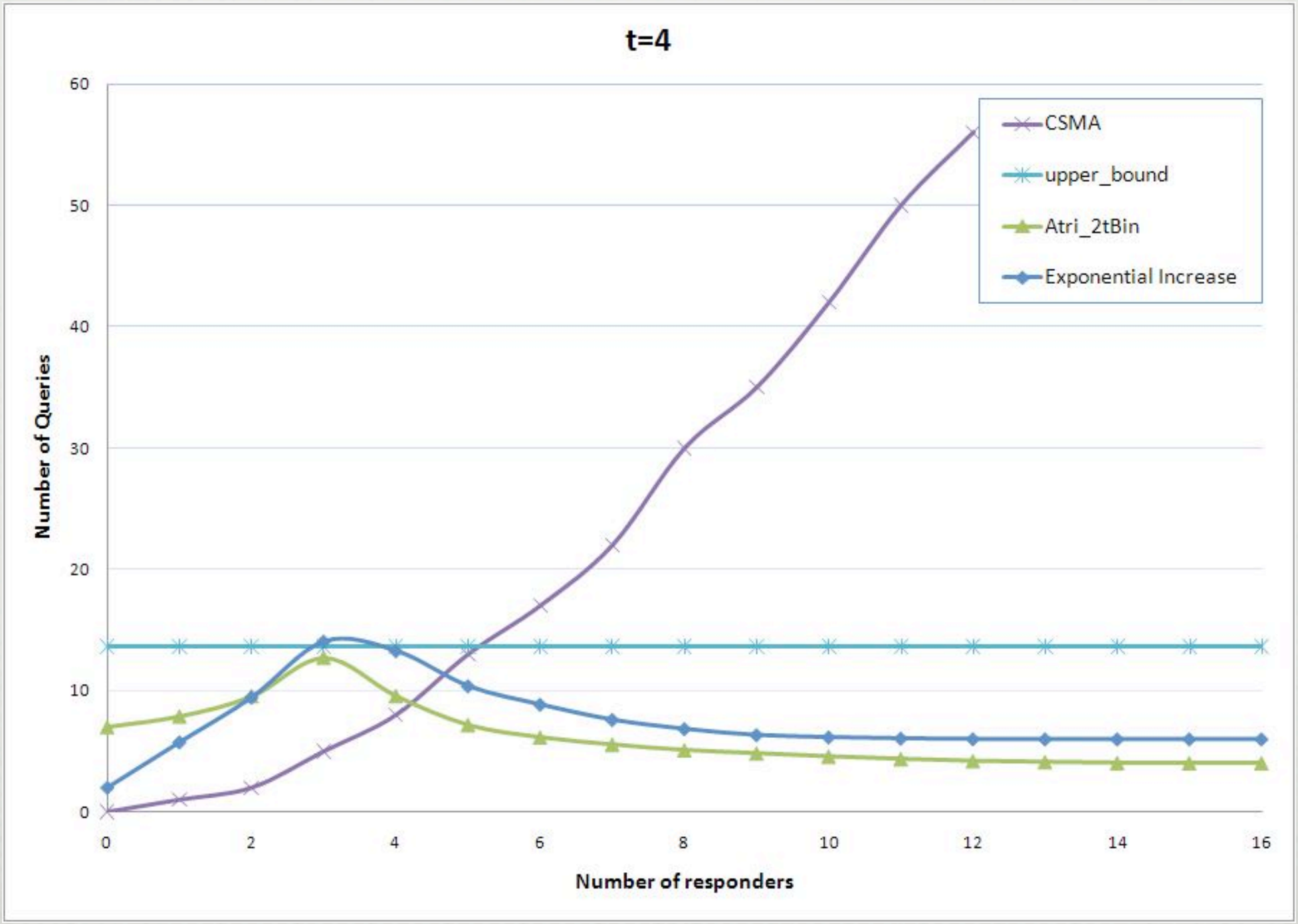
Extension: approximate counting

- ☐ Initiator uses pollcasts to approximate # of nodes P holds
- ☐ Each node has a 0.5 probability of voting at each round & they only vote once in their lifetime
- ☐ Expected number of voters is halved in each round
- ☐ Expected number of such rounds for **n** nodes is log**n**
- ☐ More precise randomized algorithms are possible for approximate counting (a la synopsis diffusion Sensys'04)

Extension: Threshold querying

- Initiator uses pollcasts to determine whether the # of nodes P holds is above a threshold t (tcast operation)
- At each round, initiator groups nodes to 2t bins & polls each bin
  Two possible cases at each round:
  - if t bins respond (1+ count detected by RCD) then threshold is reached
  - else initiator discards these t silent bins, and moves on to next round to regroup and query the nodes in the remaining bins (stops if this # < t)
- tcast completes in 2t*log(n/2t) time/work

tcast simulations



Pollcast recap

- ☐ We can't avoid collisions, we can as well use it
- ☐ Pollcast: O(1) time query on a group of motes
- ☐ Extensions
  - ☐ approximate counting
  - ☐ checking predicate P for at least-t motes

Outline

Transact: Singlehop coordination framework for WSANs

Pollcast: Singlehop collaborative feedback collection

Causataxis: Coordinated locomotion of mobile WSNs

Causataxis

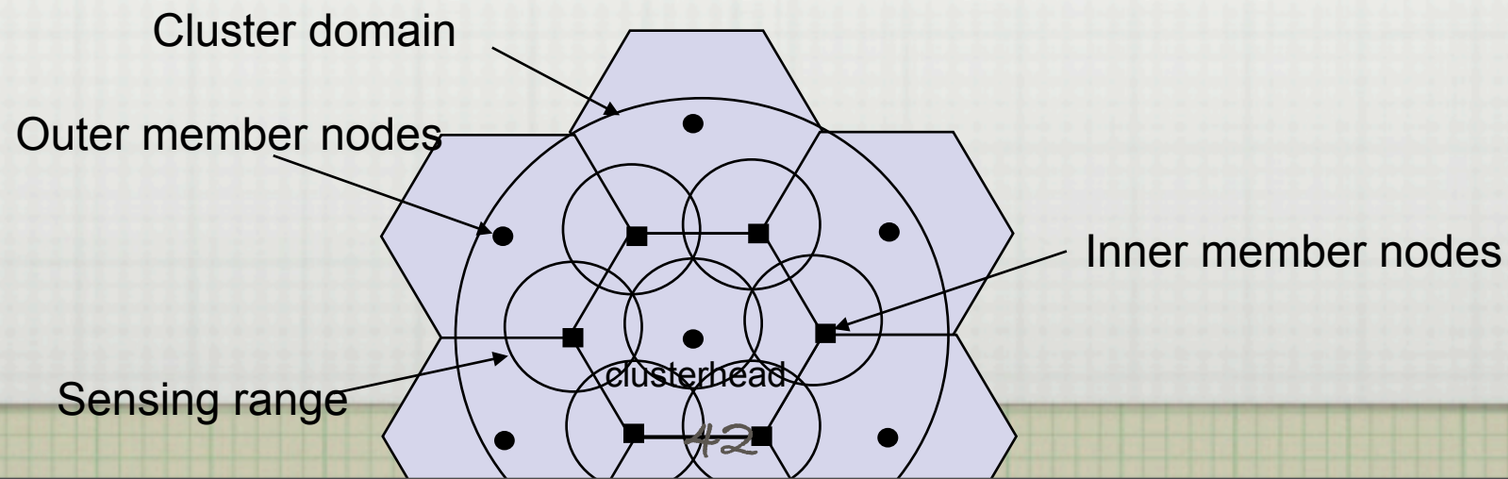
- Coordinated locomotion via grow and rot
- Grow: MSN expands toward the area of more interest
- Rot: Regions with low interest lose nodes
- Scalable control of the MSN via a backbone-tree infrastructure, maintained over clusterhead nodes
- Clusterheads ossify, cluster-member nodes are fluid

Backbone tree formation

- Clusterheads form a backbone network over the MSN and coordinate the relocation of mobile nodes
- Pipelined recruitment of mobile nodes from rotting regions towards growing regions

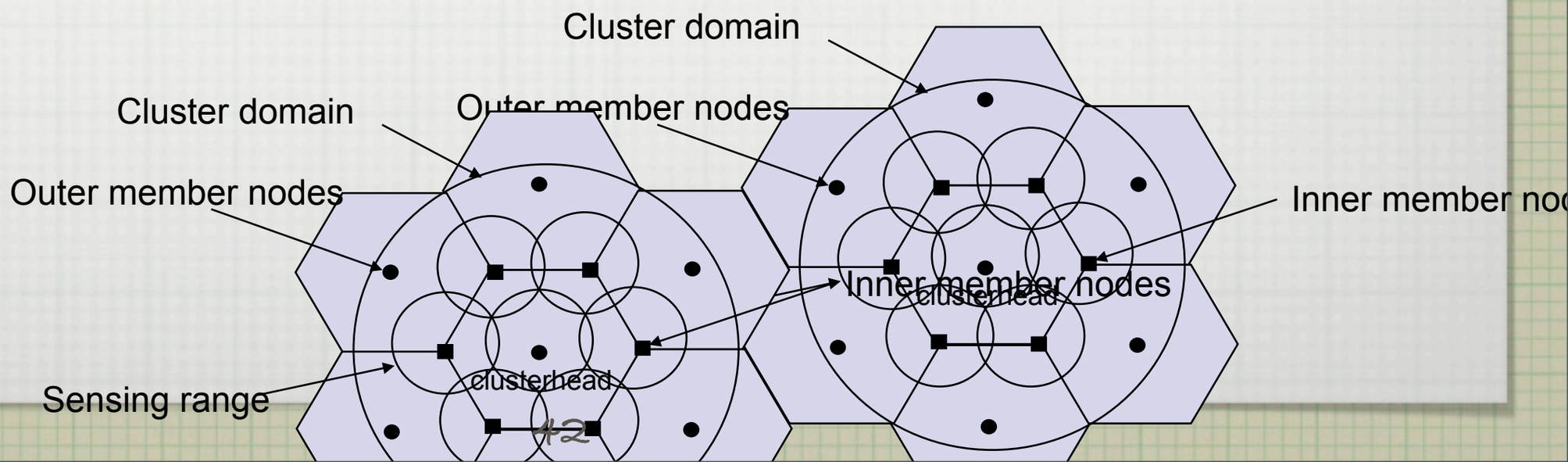
Backbone tree formation

- Clusterheads form a backbone network over the MSN and coordinate the relocation of mobile nodes
- Pipelined recruitment of mobile nodes from rotting regions towards growing regions



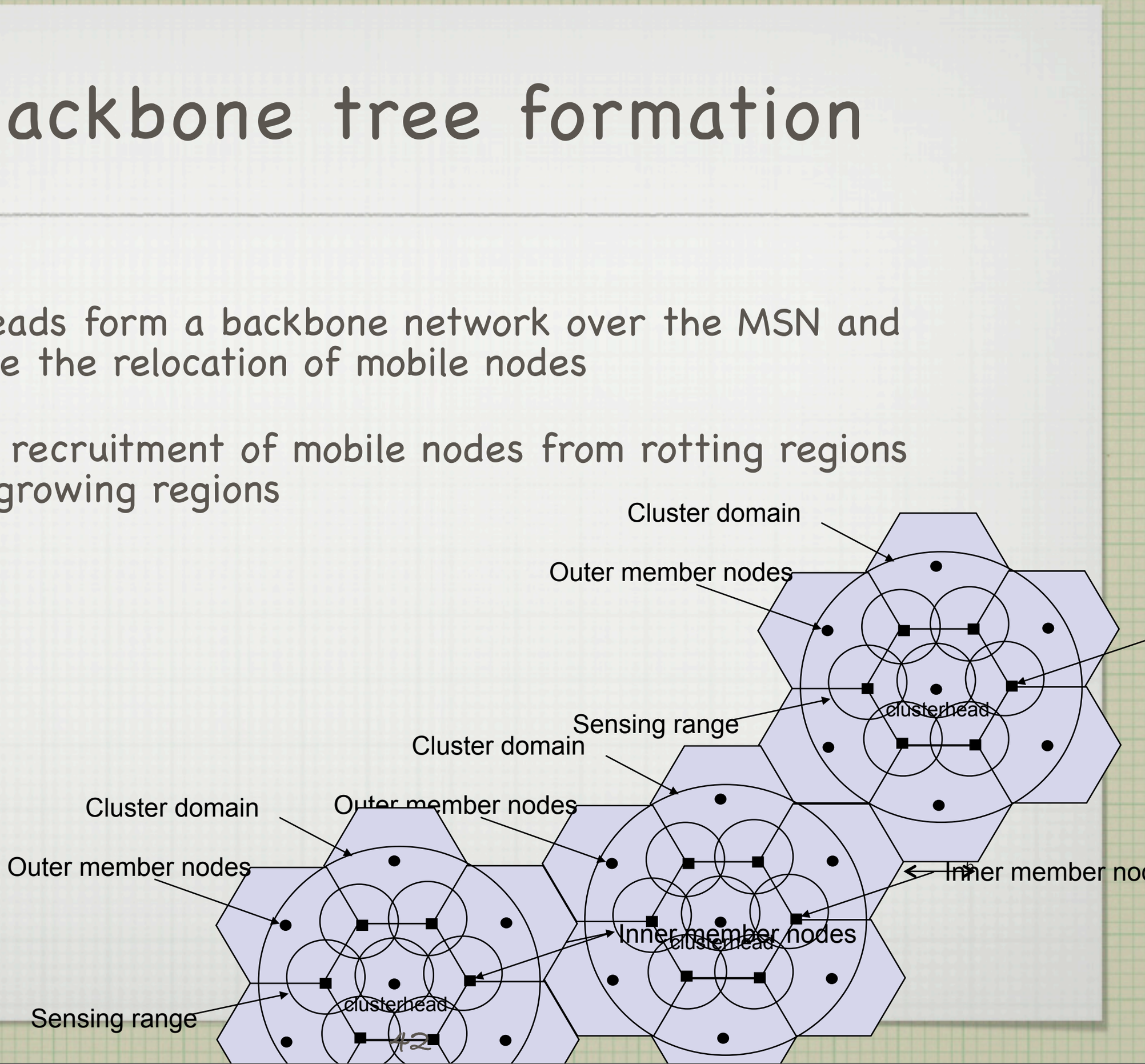
Backbone tree formation

- Clusterheads form a backbone network over the MSN and coordinate the relocation of mobile nodes
- Pipelined recruitment of mobile nodes from rotting regions towards growing regions



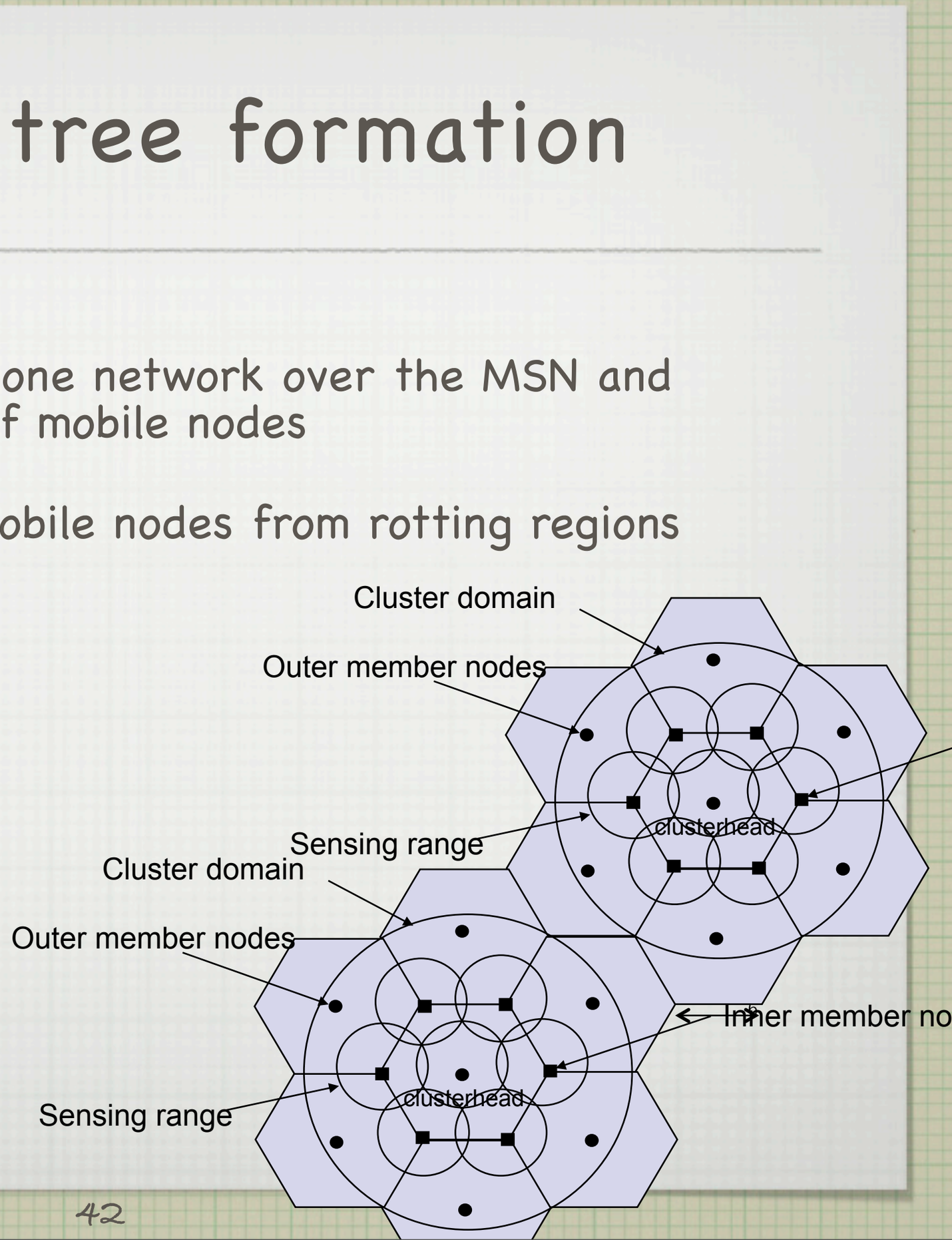
Backbone tree formation

- Clusterheads form a backbone network over the MSN and coordinate the relocation of mobile nodes
- Pipelined recruitment of mobile nodes from rotting regions towards growing regions



Backbone tree formation

- Clusterheads form a backbone network over the MSN and coordinate the relocation of mobile nodes
- Pipelined recruitment of mobile nodes from rotting regions towards growing regions



My current projects

Singlehop collaboration and coordination framework for wireless sensor actor networks (NSF Career 2008)

Tool-support for producing high-assurance reliable software for WSANs (NSF CSR 2009)

Efficient and resilient querying and tracking services for WSNs (Office of Naval Research, 2009)

Crowdsourced sensing and collaboration using Twitter (Google Research Award, 2010)

Refining shared memory model

- Shared memory and I/O automaton models are where distributed algorithms are written & verified traditionally
- We provide an automatic refinement from shared memory to WSNs (write-all with collisions model)
- Our goals are to preserve fault-tolerance/self-stabilization properties and to provide comparable performance to applications designed by hand
- We exploit Transact framework and model-revision to achieve these goals

CO-PI: SANDEEP KULKARNI, MICHIGAN STATE U.

My current projects

Singlehop collaboration and coordination framework for wireless sensor actor networks (NSF Career 2008)

Tool-support for producing high-assurance reliable software for WSANs (NSF CSR 2009)

Efficient and resilient querying and tracking services for WSNs (Office of Naval Research, 2009)

Crowdsourced sensing and collaboration using Twitter (Google Research Award, 2010)

Efficient & Reliable Querying and Tracking

STATUS QUO

Scalability challenge: centralized services do not scale for large multihop WSNs

Reliability challenge: message loss rate of 30-50% makes it hard to build consistent, reliable services in WSNs

NEW INSIGHTS

Geometric approaches lead to lightweight scalable querying & tracking

Self stabilization offers uniform mechanism to handle unanticipated faults for querying & tracking

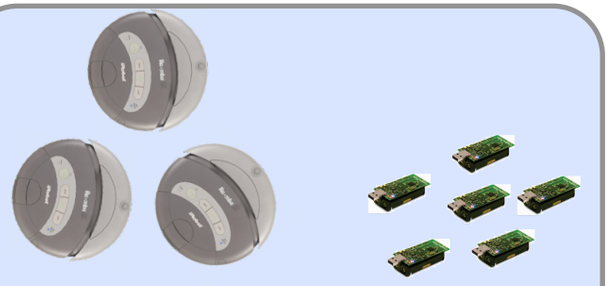
For static WSNs, investigate holistic solutions by integrating the effect of the pursuer to the problem

For passively mobile WSNs, investigate efficient solutions that learn mobility patterns of nodes

For actively mobile WSNs, investigate more controllable and predictive alternatives to swarms

For each case, investigate self stabilization & fault-local recovery against unanticipated faults

QUANTITATIVE IMPACT



Improve performance and dependability of intrusion detection, querying & tracking of targets in static and mobile WSNs

END-OF-PHASE GOAL

Goal: Achieve novel, lightweight, scalable, and reliable querying & tracking for static, passively/actively mobile WSNs

Geometric approaches & stabilization enable scalable services in WSNs

My current projects

Singlehop collaboration and coordination framework for wireless sensor actor networks (NSF Career 2008)

Tool-support for producing high-assurance reliable software for WSANs (NSF CSR 2009)

Efficient and resilient querying and tracking services for WSNs (Office of Naval Research, 2009)

Crowdsourced sensing and collaboration using Twitter (Google Research Award, 2010)

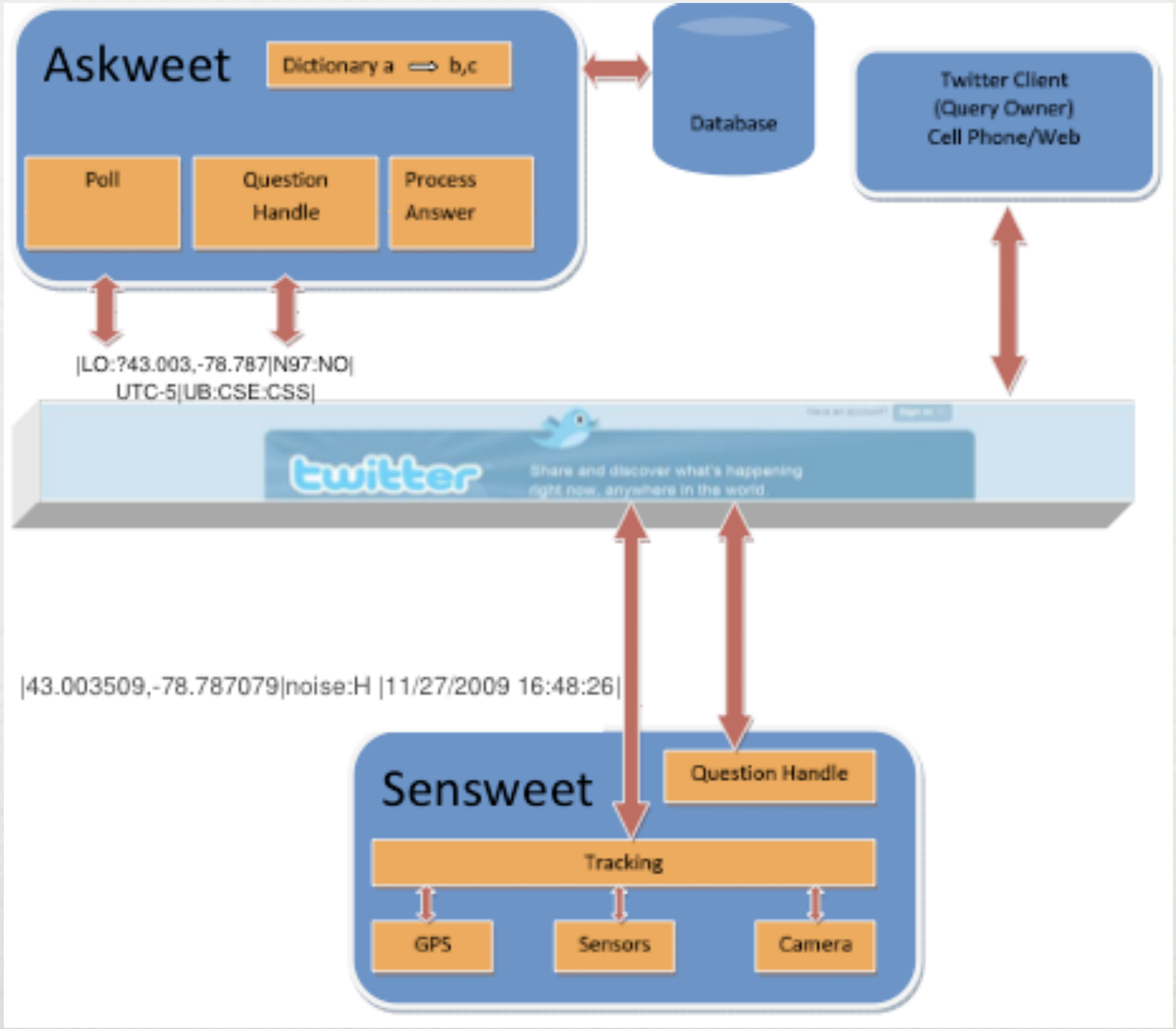
The missing piece

- Despite availability of sensors and smartphones, state-of-the-art falls short of the ubiquitous computing vision

- The missing piece is the infrastructure to task/utilize these devices for collaboration!

- We propose that Twitter can provide an open publish-subscribe infrastructure for sensors and smartphones, and enable crowdsourced sensing & collaboration

System architecture



RAINRADAR & NOISE MAPPING APPLICATIONS