Trade-Offs in Implementing Consistent Distributed Storage

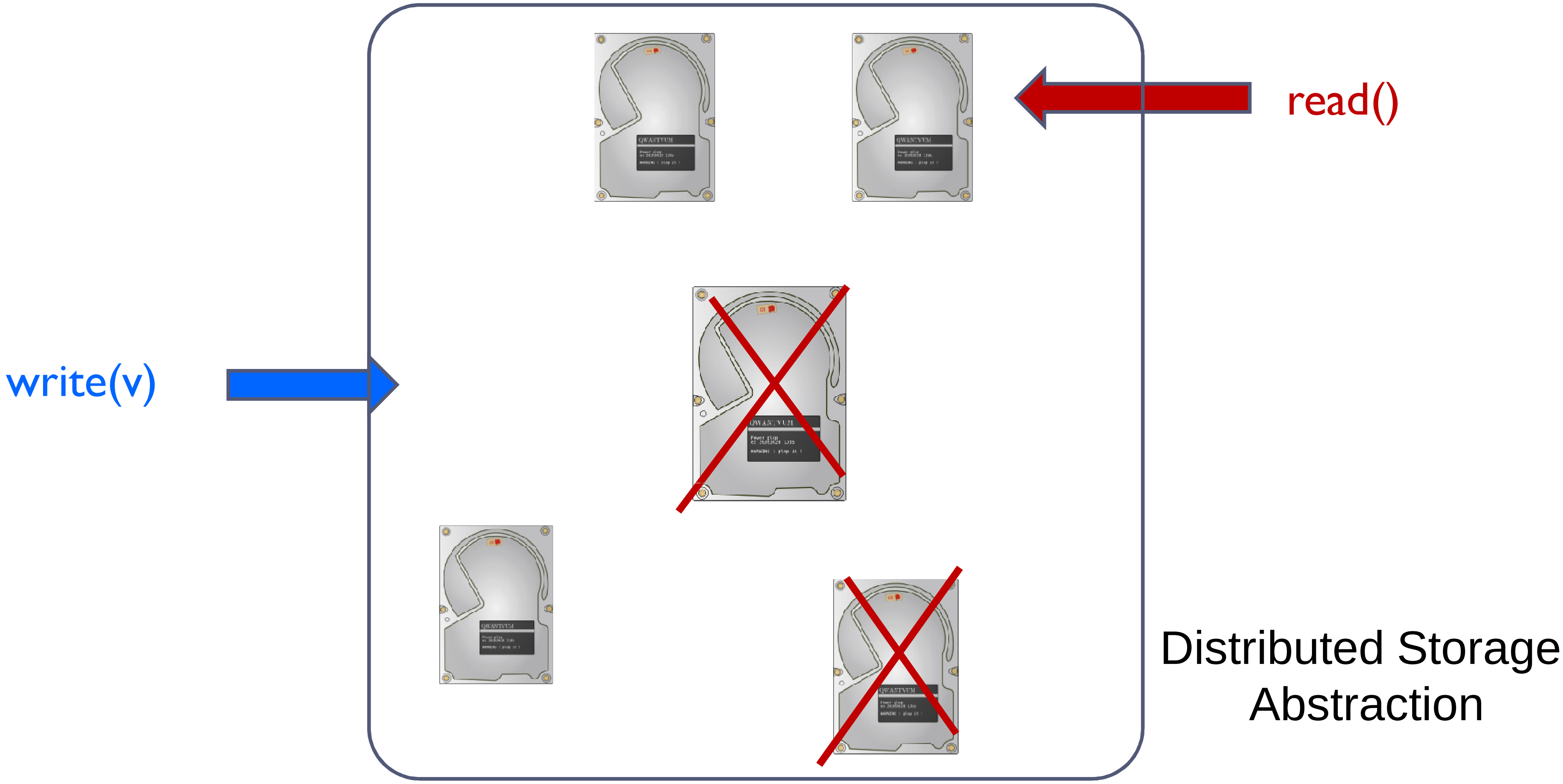
PhD Dissertation Defense

Nicolas Nicolaou

Major Advisor: Dr. Alexanter A. Shvartsman

Associate Advisors: Dr. Alexander Russell, Dr. Aggelos Kiayias, Dr. Chryssis Georgiou

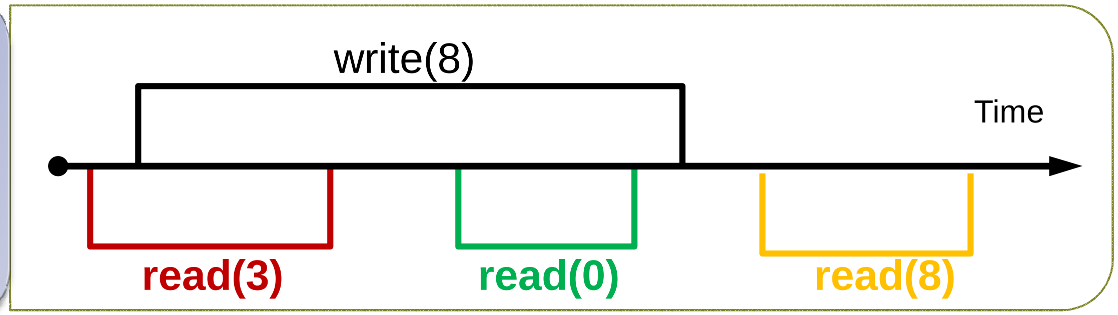
What is a Distributed Storage System?



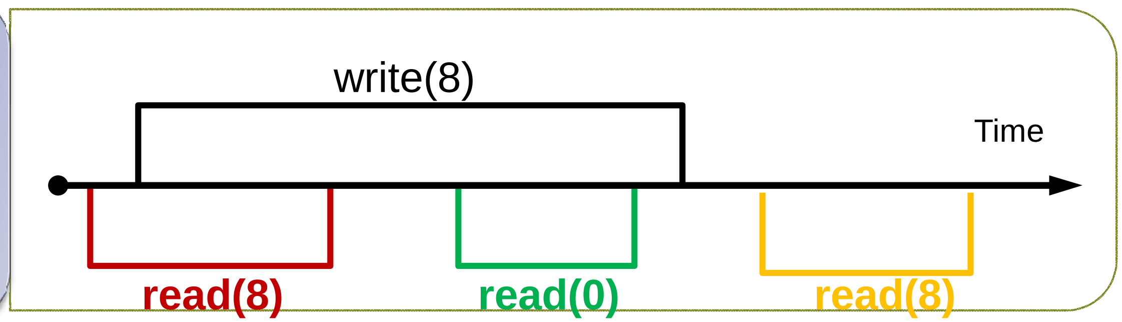
- Data Replication – Servers/Disks
  - Survivability and Availability
- Read/Write operations
- Consistency Semantics

Consistency Semantics [Lamport86]

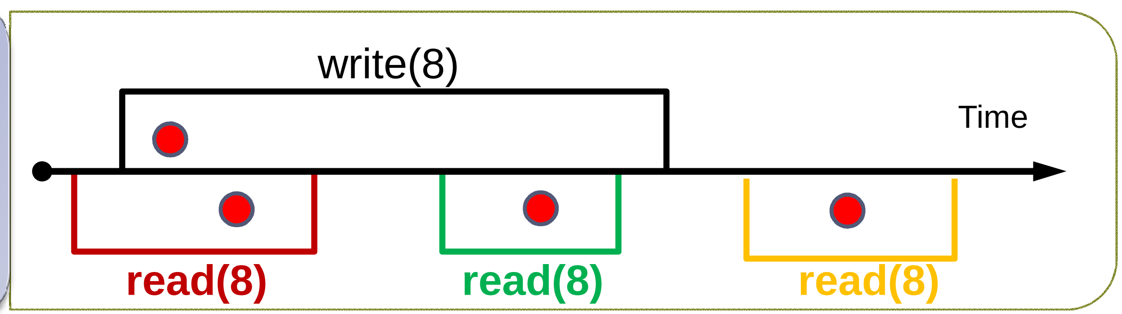
Safety



Regularity



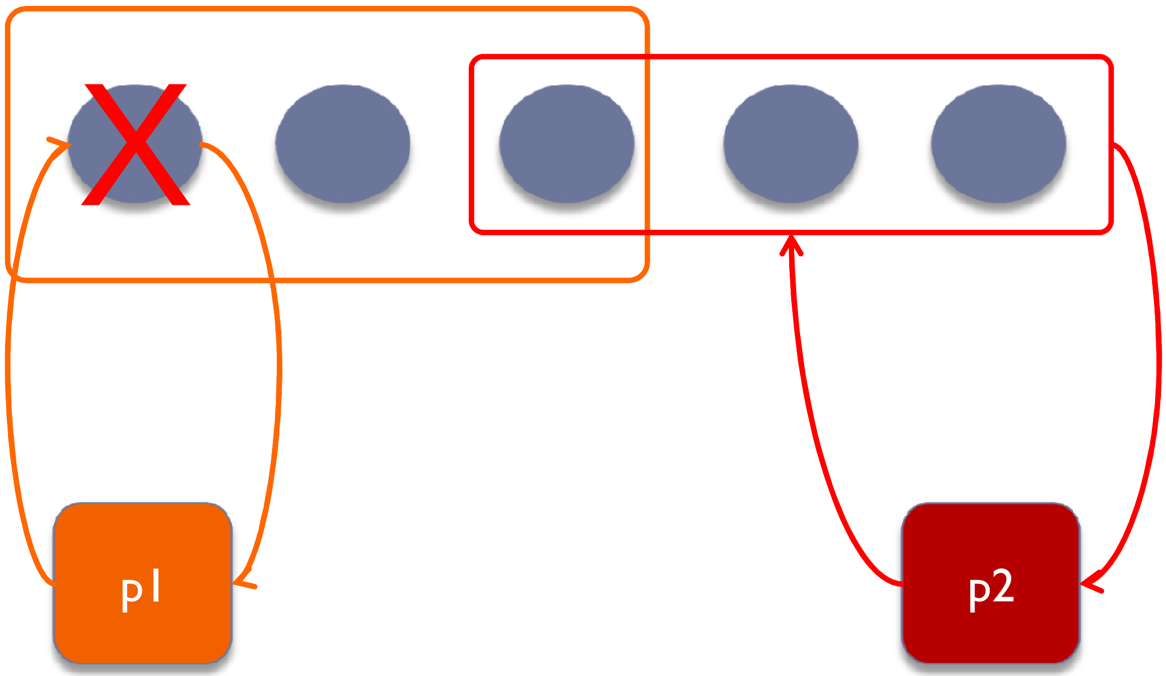
Atomicity



How to order read/write operations?

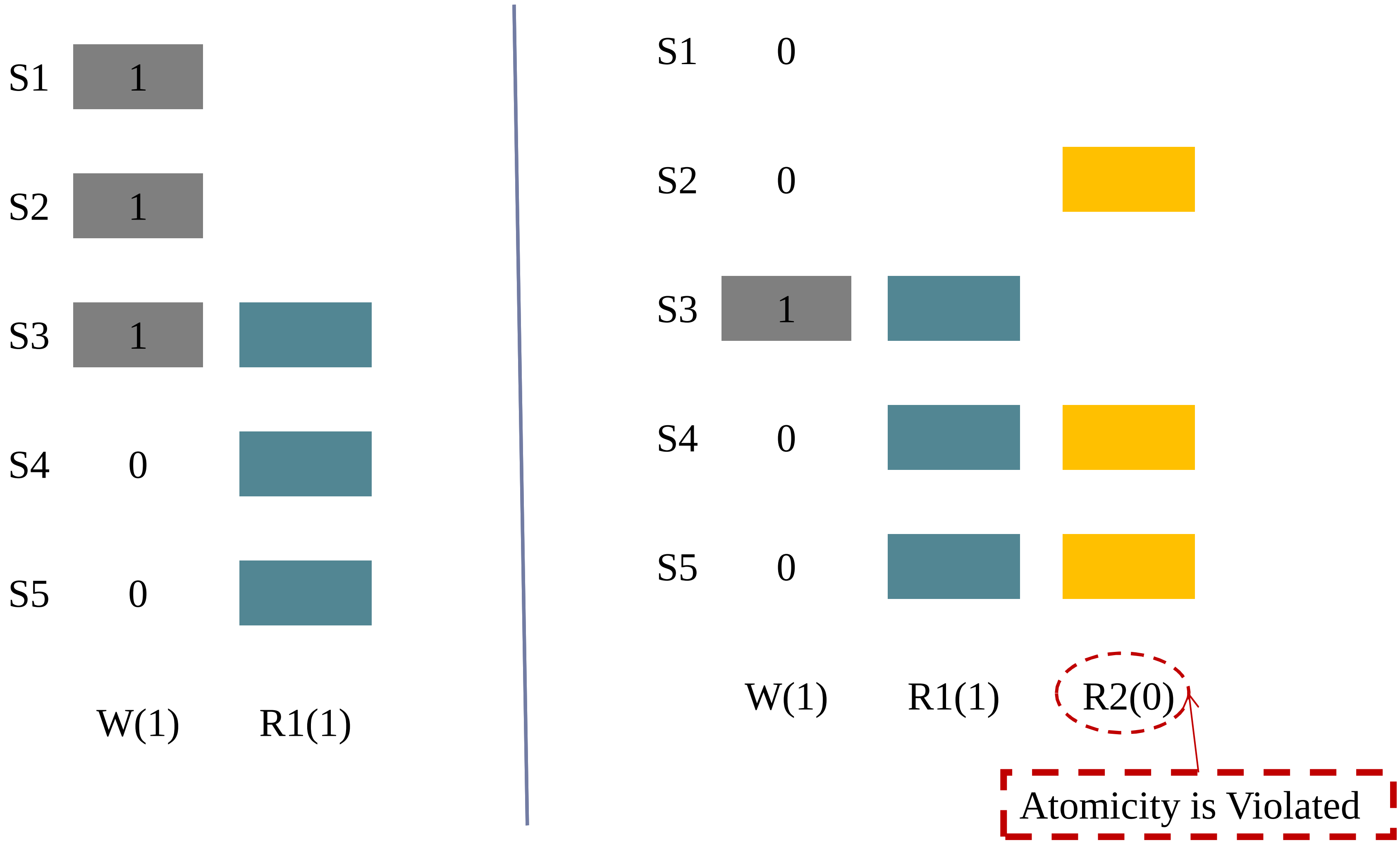
- Based on the value each operation writes/returns
  - Non-unique Values
- Using the "time" at which each operation is invoked
  - Clock Synchronization
- Associate a sequence number with each value written
  - SWMR: timestamps
  - MWMR: tags=<timestamp, wid>

Challenges – Communication Rounds



Multiple Round-Trips

- Consider the following example [Attiya et al. 96]:



Efficiency Measure-Operation Latency

Operation Latency is measured in
Communication Rounds (round-trips)

Prior Work: Traditional Implementations

SWMR

- e.g., [Attiya et al. 96]
- Single round writes
- Two round reads
  - Phase 1: Obtain latest value
  - Phase 2: Propagate latest value
    - Folklore belief: "Reads must Write"

MWMR

- e.g., [LS97, ES00, LS02]
- Two round writes
  - Phase 1: Discover latest value
  - Phase 2: Order new value after the latest and propagate
    - Belief: "Writes must Read"
- Two round reads

Prior Work: "Fast" Implementations

MWMR Crashes

- [Dolev et al. 03, Chockler et al. 09]
- Single round (***fast***) reads
  - Only if tag is *confirmed*
  - When the written value is propagated to a *full quorum*

SWMR Crashes

- [Dutta et al. 04]
- Single round (***fast***) reads and writes
  - All operations are fast
  - Bounded readers: $R<(S/f)-2$ where S servers & f failures
  - Impossible in MWMR model

SWMR Byzantine

- "Lucky" Operations [GV 06]
  - Synchronous (receive replies from all servers within some interval)
  - Contention free (not concurrent with a write)
- Refine Quorum Systems [GV 07]
  - One to Three round reads
  - Eventual Synchrony and use of timeouts

Goal of this Thesis….

What is the operation latency of atomic register implementations in an unconstrained, fail-prone, message-passing, asynchronous distributed system?

What are the trade-offs to achieve such performance?

Model - Definitions

- Asynchronous, Message-Passing model
  - Process sets: writers W, readers R, servers S (replica hosts)
  - Reliable Communication Channels (unless otherwise stated)
  - Well Formedness

- Environments:
  - SWMR: $|W|=1$, $|R|\geq 1$
  - MWMR: $|W|\geq 1$, $|R|\geq 1$

- Failures:
  - Crash Failures

- Correctness: Atomicity (safety), Termination (liveness)

Definition: Quorum Systems

- Quorum System $\mathbf{Q}$:

$$\mathbf{Q} = \{Q : Q \subseteq S\}\, s.t.\, \forall Q_i, Q_j \in \mathbf{Q} : Q_i \cap Q_j \neq \emptyset$$

- n-wise Quorum System $\mathbf{Q}$:

$$\mathbf{Q} = \{Q : Q \subseteq S\}\, where\, \forall A \subseteq \mathbf{Q} : |A| = n \text{ and } \bigcap_{Q \in A} Q \neq \emptyset$$

  - $2 \leq n \leq |\mathbf{Q}|$: intersection degree

- Faulty Quorum: Contains a faulty process
  - At least a single quorum contains non-faulty replicas

- Faulty Quorum System: Every quorum is faulty

Definition: Fastness

- A process p performs a communication round during an operation π if:
  - p sends a message m to a set of servers for π
  - Any server that receives m replies to p
  - Once p receives "enough" responses completes π or proceed to a next communication round

- Fast Operation
  - Completes after the end of its first round

- Fast Implementation
  - All operations are fast

- Communication scheme
  - Message delivery: Servers to Clients
  - No server to server or client to client communication

Can we trade fastness for scalability?

Question

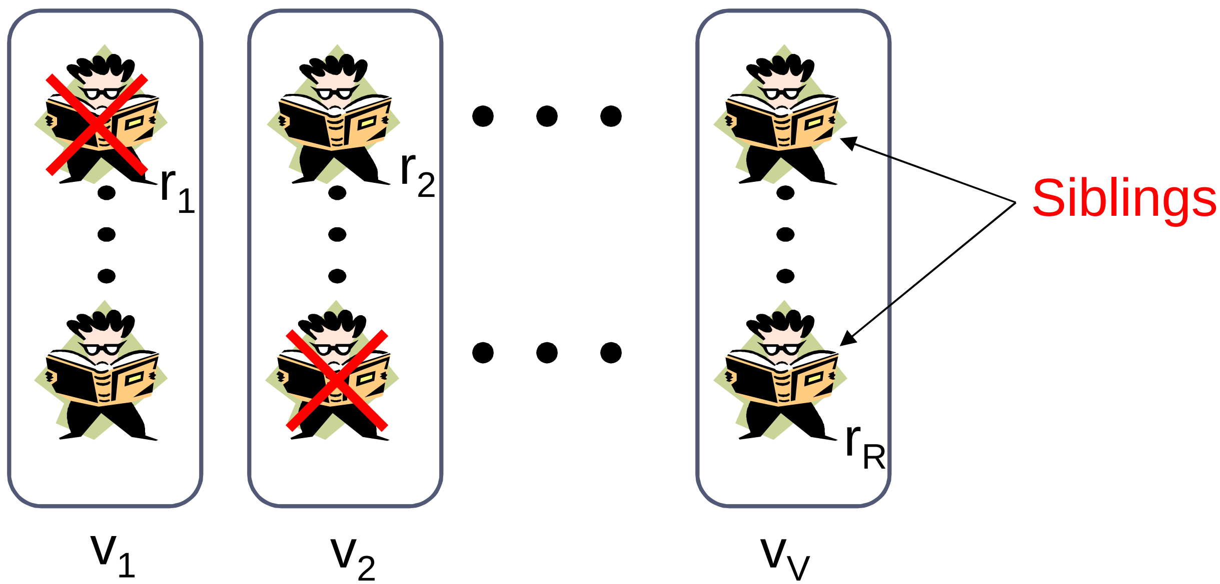
Can we allow fast operations in atomic register implementations with unbounded number of readers?

Definition: Semifast Implementations

- Writes are ***fast***
- Reads perform ***1 or 2 rounds***
- Only a ***single complete slow read per write operation***; any read that proceeds or succeeds the slow read and returns the same value is fast.
- There exists an execution with ***only fast operations***

Algorithm: SF

- Idea: Group readers into Virtual Nodes
  - Local vid assignment per process
  - *V*: set of virtual node identifiers
  - Challenge: achieve atomicity between siblings



Algorithm: SF

Write Protocol: one round

- Increment ts and send <ts,v> to S-f servers

Read Protocol: one or two rounds

- Collect S-f replies and find the maxTS
  - **Fast**: maxTS seen by "few" VN and is not confirmed => maxTS-1
  - **Fast**: maxTS seen by "enough" VN or ≥f+1 confirmed maxTS=> maxTS
  - **Slow**: maxTS seen by exact # of VN or <f+1 confirmed maxTS => maxTS

Server Protocol

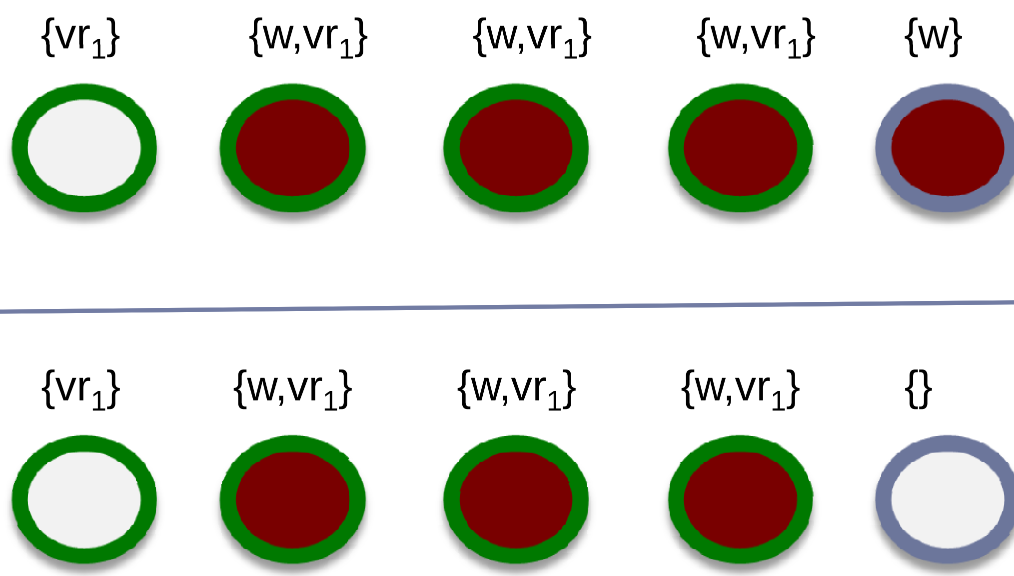
- Receive read/write request:
  - Update replica ts and value and record requester's vid
- Received inform request: mark ts as confirmed

Idea of the Predicate

- Assume |S|=5, f=1 and operations
  - write(v) => |S|-f servers
  - Complete read() from $<r_1,vr_1>$ => |S|-f servers
    - Witness v in |S|-2f servers => |seen| = 2
    - returns v to preserve atomicity (both executions)

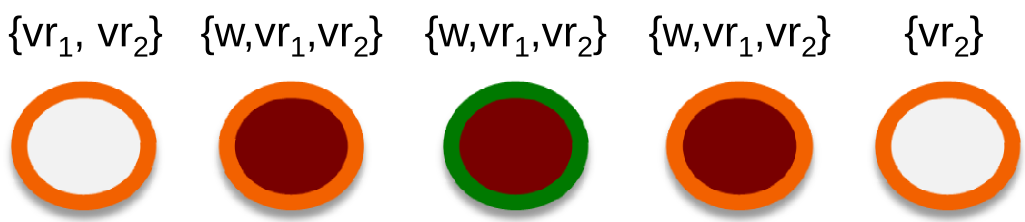
Execution (a):
write(v) Complete

Execution (b):
write(v) Incomplete

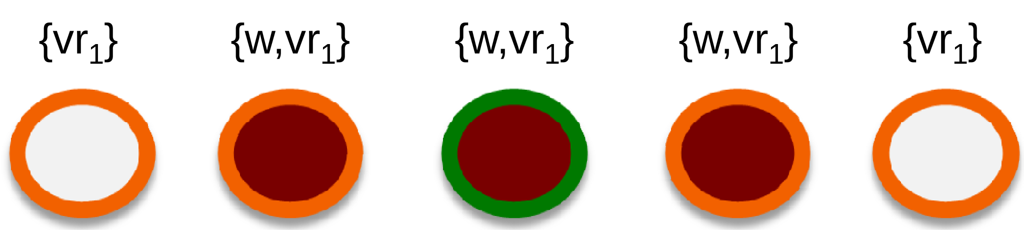


Idea of the Predicate (Cont.)

- Extend (b) by read() from $\langle r_2, vr_2 \rangle$ (not sibling with r_1):
  - Witness v in |S|-3f servers, |seen| = 3
  - Returns v to preserve atomicity



- Extend (b) by read() from $\langle r_2, vr_1 \rangle$ (sibling with r_1):
  - Witness v in |S|-3f servers, |seen| = 2
  - Has to return v to preserve atomicity => r_1 needs 2^{nd} round



Impossibility Results

Theorem: A semifast implementation is not possible if the number of virtual nodes is

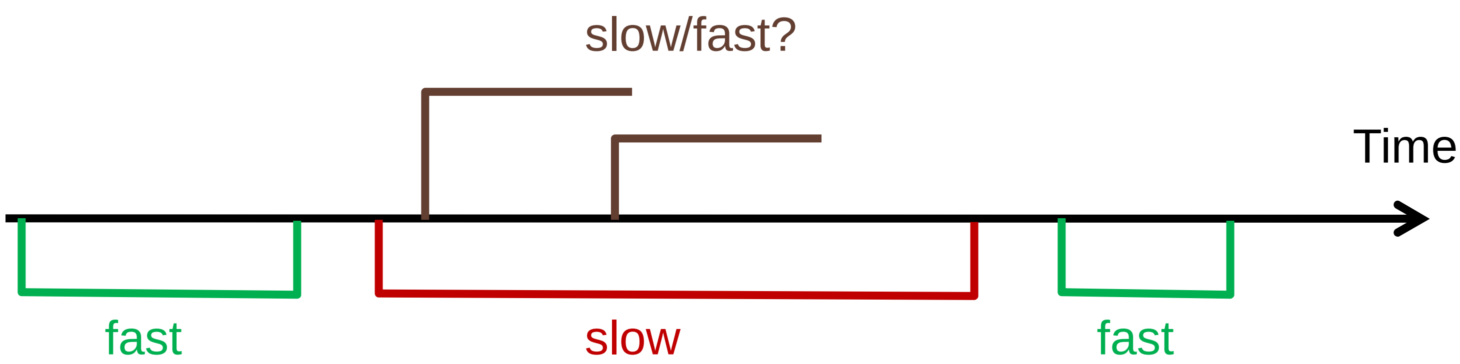
$$|V| \geq (|S|/f)-2$$

and a second round contacts fewer than $3f$ servers.

Theorem: It is not possible to devise a MWMR semifast implementation even with $|W|=2$, $|R|=2$ and $f=1$.

Multiple Slow Reads per Write

- By Definition: One complete slow read per write
  - No guarantees for reads concurrent with the slow read!!



Measuring the Number of Slow Reads

Probabilistic Bounds

- Low Contention: $O(log|R|)$
- High Contention: $O(|R|)$

Simulation

- Stochastic: 10% (worst case)
- Fix Interval: 60% (worst case)

Observation

- Fast Implementations [Dutta et al. 04]:
  - By their bound: $f<|S|/(|R|+2)$
    - So ***$f<|S|/4$*** if we want to support 2 readers
- Semifast Implemementations:
  - By our bound: $f<|S|/(|V|+2)$
    - So ***$f<|S|/3$*** since a single VN accommodates unbounded readers
- ABD Algorithm [ABD 96] (all slow reads):
  - Majorities: ***$f<|S|/2$***
- ***Is there a relation between server organization and fastness?***

Quorum-Based Implementations

Question

Can we devise atomic register implementations that allow fast operations using a general quorum system construction?

First the Bad News

Theorem: Fast and Semifast implementations are possible in an unconstrained quorum-based environment *iff* the underlying quorum system Q is a |Q|-wise quorum system

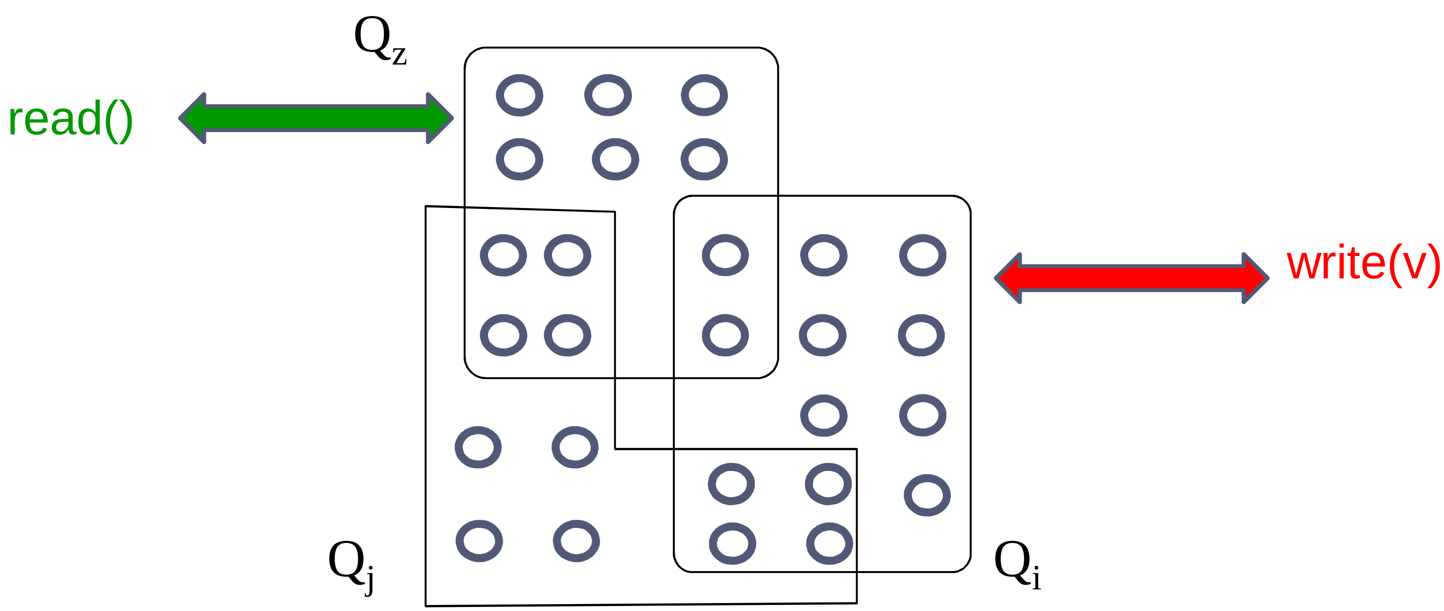
$$\bigcap_{Q \in \mathbb{Q}} Q \neq \emptyset$$

Remark: Fast and Semifast quorum-based implementations of atomic register are ***not fault-tolerant***.

- Single failure in the common intersection disables the quorum system.

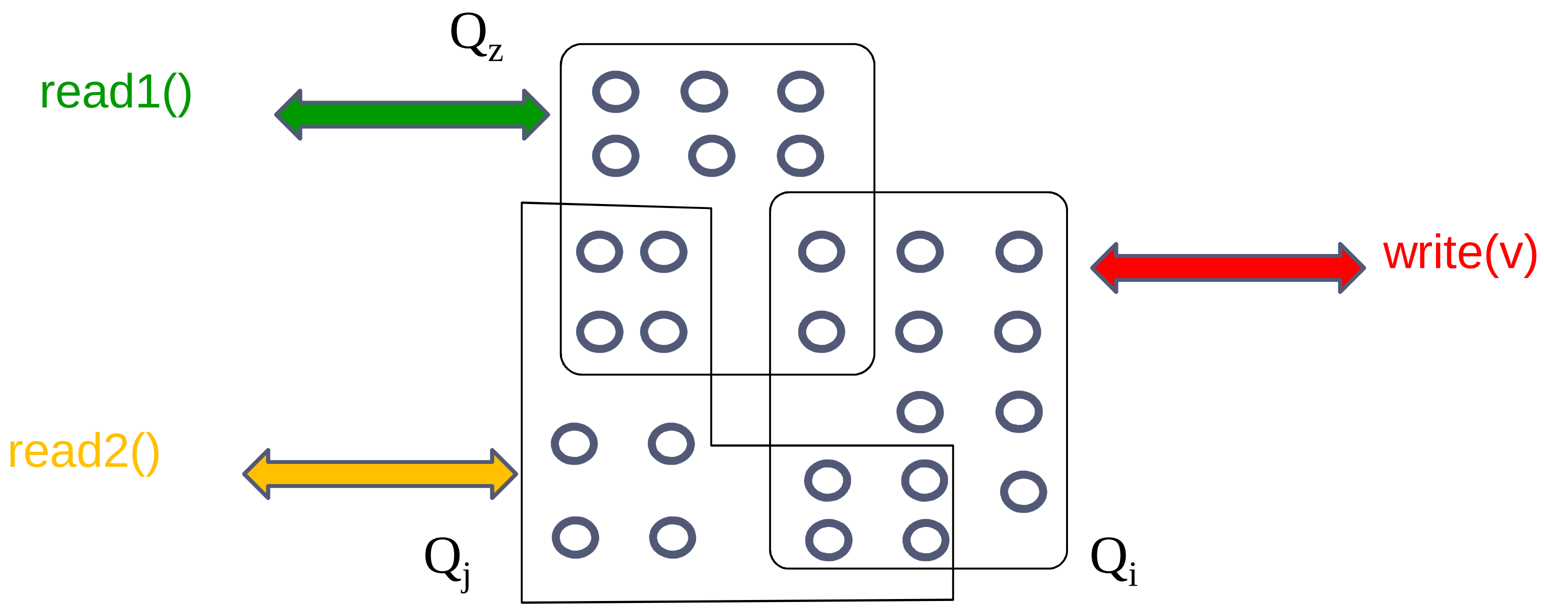
Non-Robust Fast Implementations: Proof Sketch

- Execution a:
  - Complete write(v) => Q_i
  - Complete read() => Q_z
    - read() returns v to preserve atomicity



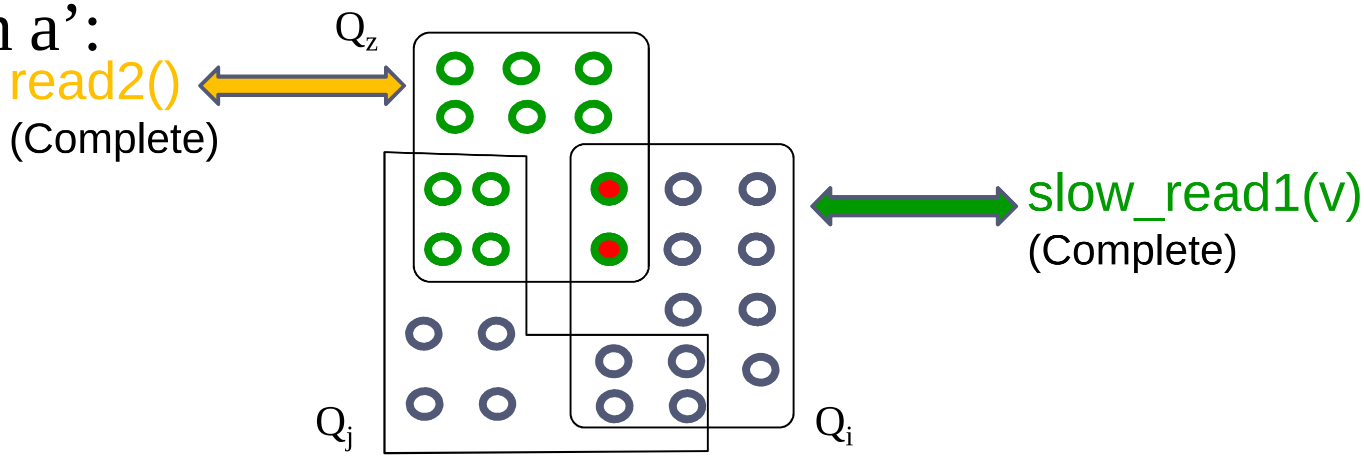
Not Robust Fast Implementations (Proof Sketch)

- Execution b:
  - Incomplete write(v) => $Q_i \cap Q_z$
  - Complete read1() => Q_z
    - read1() cannot distinguish between executions a and b, thus returns v
  - Complete read2() => Q_j
    - read2() returns an older value since does not observe v

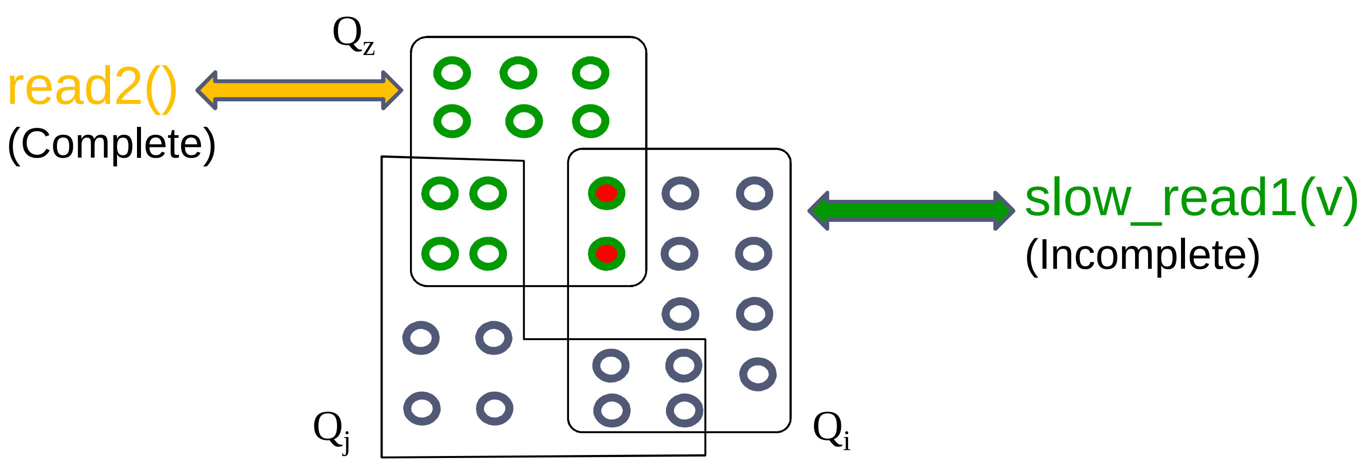


Not Robust SemiFast Implementations (Proof Sketch)

Execution a':



Execution b':



Now the Good News

- Introduce Weak Semifast Implementations
  - Trade speed for efficiency and fault-tolerance
    - Allow multiple "slow" reads per write operation but maintain the fast behavior when possible
  - To do so, we introduce **Quorum Views**

- Simulations of Quorum View Implementation
  - <13% of slow reads in realistic scenarios

Quorum Views

Idea:

- Try to determine the state of the write operation based on the distribution of the maxTS in the replied quorum.

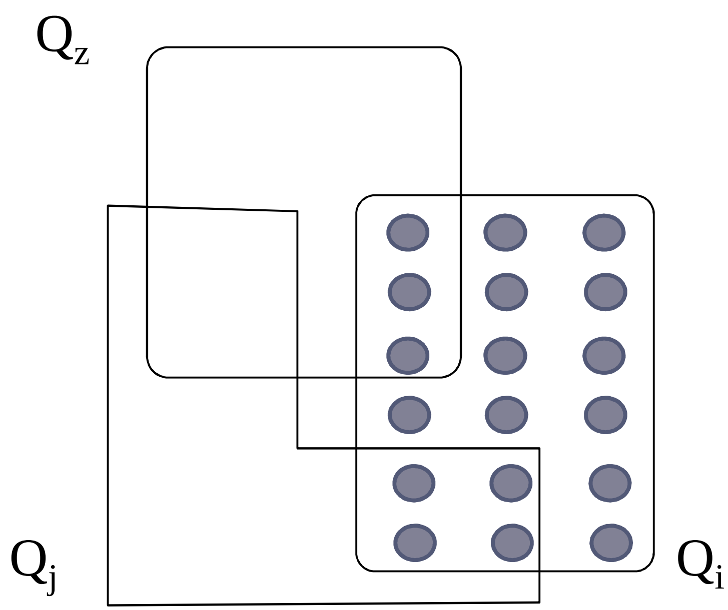
- Write State in the First Round of Read Operation

  Determinable => Read is Fast

  Undeterminable => Read is Slow

Determinable Write - Qview(1)

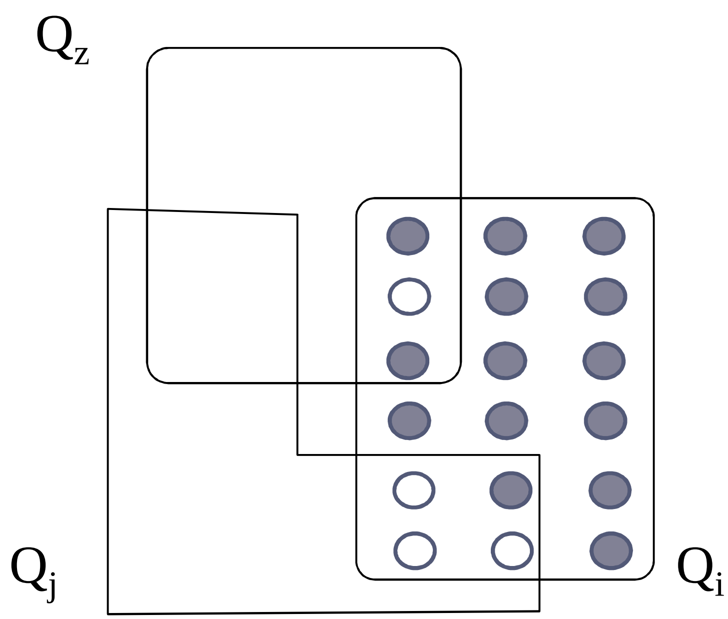
- All members of a quorum contain the maxTS



(Potentially) Write Completed

Determinable Write - Qview(2)

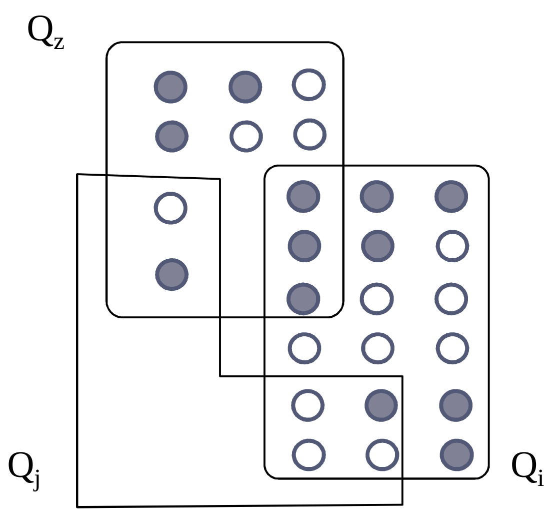
- Every intersection contains a member with ts<maxTS



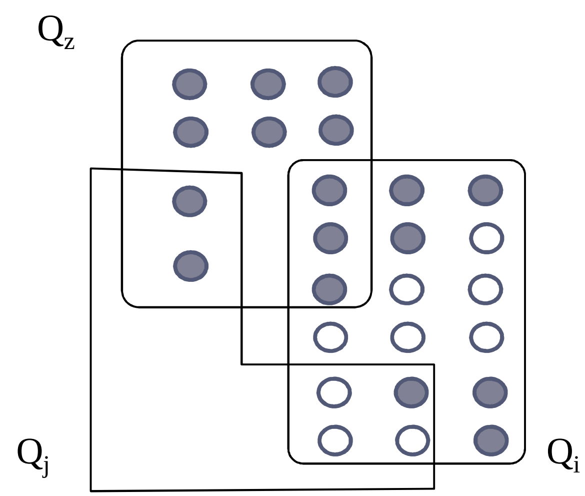
(Definitely) Write Incomplete

Undeterminable Write - Qview(3)

- There is intersection with all its members with ts=maxTS



qV(3) and Incomplete Write



qV(3) and Complete Write

Undeterminable => second Com. Round

Algorithm: SLIQ

Write Protocol: one round

- P1: Writer increments ts and propagates the <ts,v> to a quorum

Read Potocol: one or two rounds

- P1: send read requests and wait for replies from a quorum Q
  - $QView_Q(1)$ – **Fast** and return maxTS
  - $QView_Q(2)$ – **Fast** and return maxTS-1
  - $QView_Q(3)$ – **Slow** proceed to P2 and return maxTS
- P2: propagate <maxTS,v> to a quorum and return <maxTS,v>

Server Protocol: passive role

- Receive requests, update local timestamp and return <ts,v>

Thus Far…

SWMR Fast

- Single round (***fast***) writes and reads
  - Bounded readers: $R<(S/f)-2$ where *S* servers & *f* failures
  - Impossible in MWMR model

SWMR Semifast

- Fast writes
- Only a single complete 2-round (***slow***) read per write
  - Unbounded readers
  - Impossible in the MWMR model

SWMR Weak-Semifast

- General Quorum System
- Fast writes and Multiple slow reads per write
  - Allows concurrent fast reads with writes
  - Unknown if applicable in MWMR model

What about MWMR?

Question

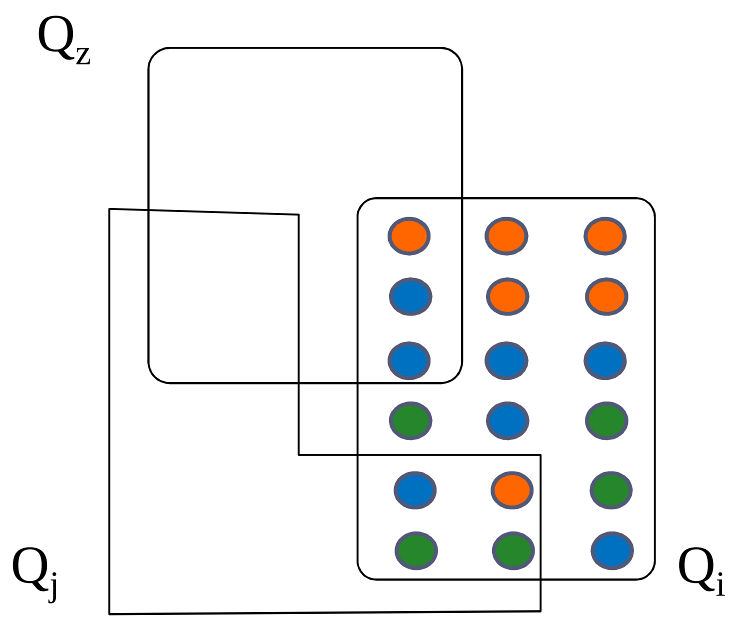
Can we use Quorum Views to devise MWMR atomic register implementations that allow executions that contain fast operations?

Quorum Views – CWFR Algorithm

- Idea:
  - Adopt techniques developed for the SWMR
- Quorum Views
  - Allow fast operations in unconstrained SWMR environments
- Generalize the Quorum Views for MWMR

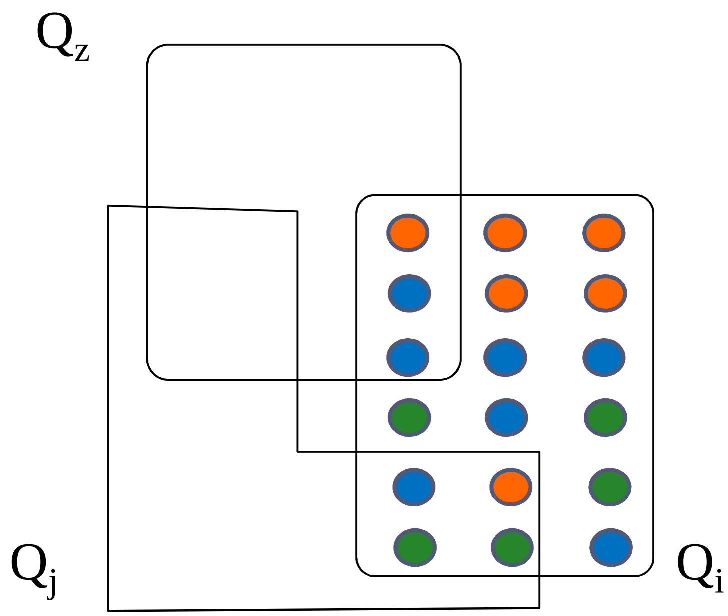
What happens in MWMR?

- MWMR environment
  - Concurrent writes
  - Multiple concurrent values
- For values <tag1,v1> , <tag2, v2>, <tag3,v3>
  - Let tag1 < tag2 < tag3



Idea: Uncover the Past

- Discover the latest potentially completed write
- For values <tag1,v1> , <tag2, v2>, <tag3,v3>:
  - <tag3,v3> not completed (servers possibly contained <tag2, v2>)
  - <tag2, v2> not completed (servers possibly contained <tag1,v1>)
  - <tag1,v1> potentially completed



Algorithm: CWFR

Traditional Write Protocol: two rounds

- P1: Query a single quorum for the latest tag
- P2: Increment the max tag, send <newtag, v> quorum

Read Protocol: one or two rounds

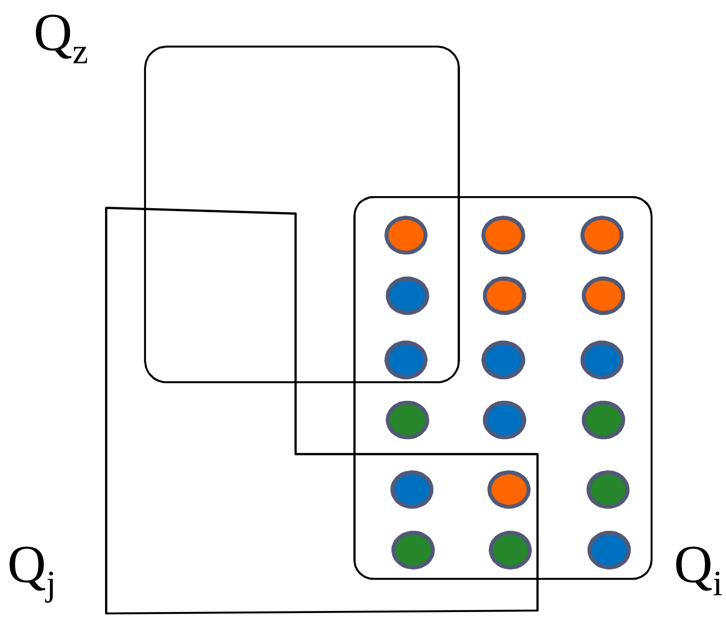
- Iterate to discover smallest completed write
- P1: receive replies from a quorum Q
  - $QView_Q(1)$ – **Fast**: return maxTS of current iteration
  - $QView_Q(2)$ – remove servers with maxTS and re-evaluate
  - $QView_Q(3)$ – **Slow**: propagate and return $maxTS_0$

Server Protocol: passive role

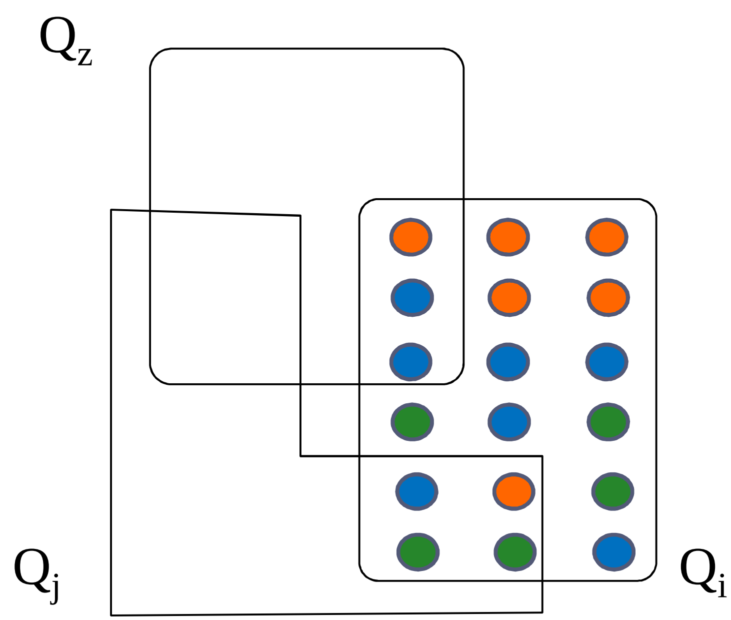
- Receive requests, update local timestamp and return <ts,v>

Read Iteration: Discard Incomplete Tags

- For values <tag1,v1> , <tag2, v2>, <tag3,v3>:
  - <tag3,v3> not completed: remove servers that contain <tag3,v3>
  - <tag2, v2> not completed: remove servers that contain <tag2, v2>
  - <tag1,v1> potentially completed in Q_i
    - Qview(1) : all remaining servers contain <tag1,v1>



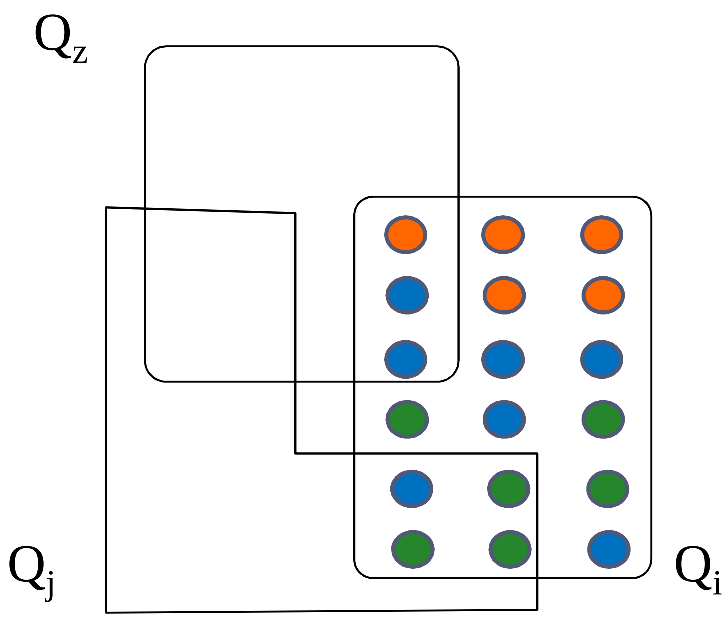
Server Removal



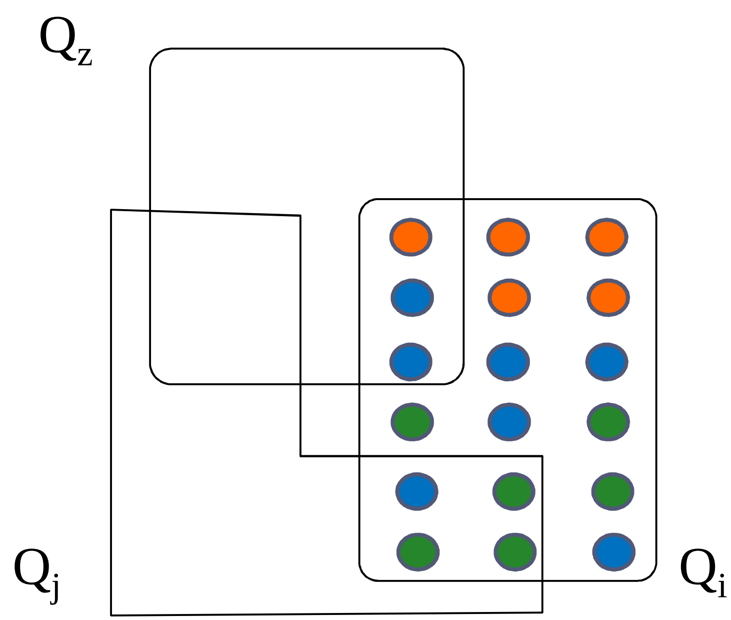
Past Prediction

Read Iteration: Discard Incomplete Tags

- For values <tag1,v1> , <tag2, v2>, <tag3,v3>:
  - <tag3,v3> not completed: remove servers that contain <tag3,v3>
  - <tag2, v2> potentially completed in Q_j
    - Qview(3) : an intersection of the remaining servers contains <tag2, v2>
    - P2: propagate <tag3,v3> to a complete quorum (help <tag3,v3> to complete)



Server Removal



Past Prediction

What about fast writes?

Question

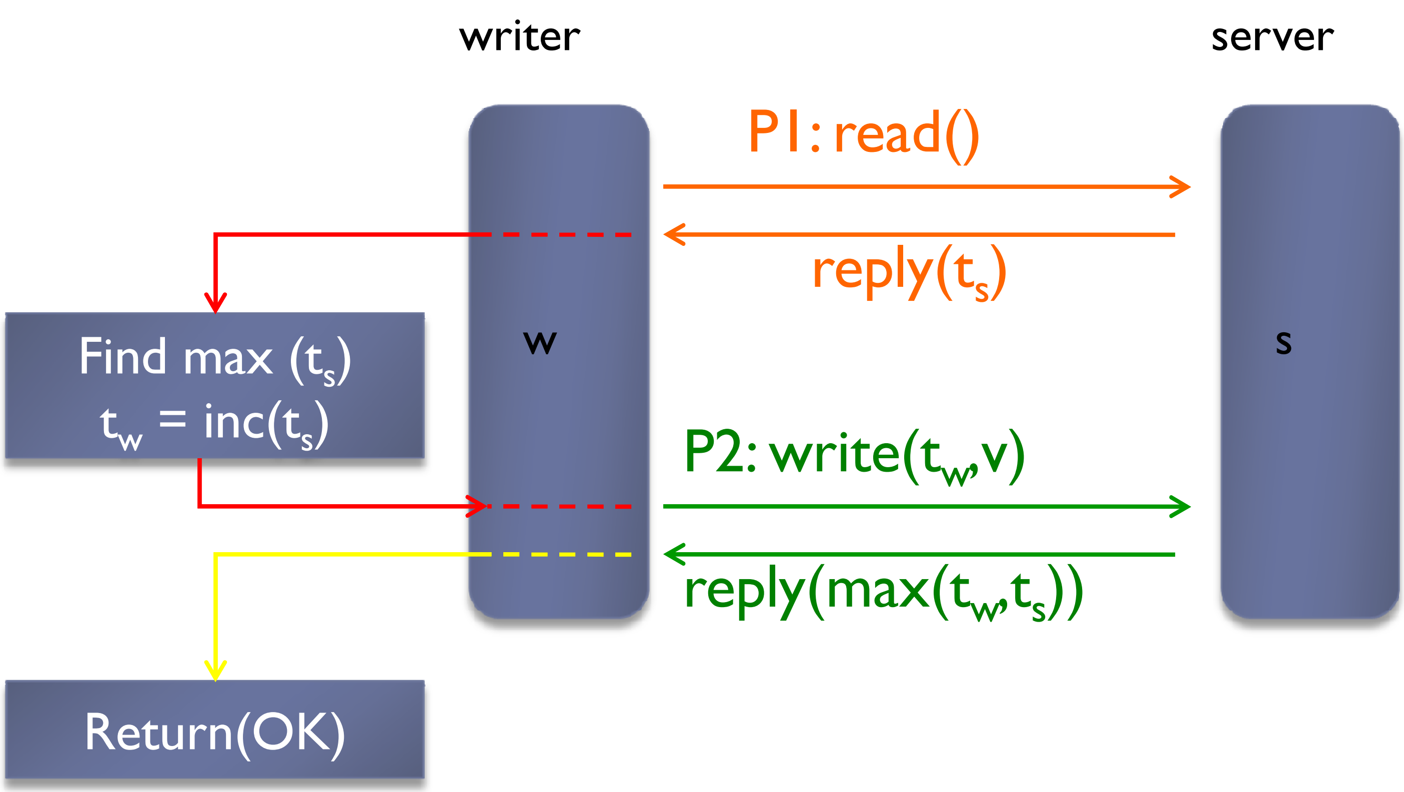
Can we devise MWMR atomic register implementations that allow executions that contain both fast read and write operations?

New Technique - SSO

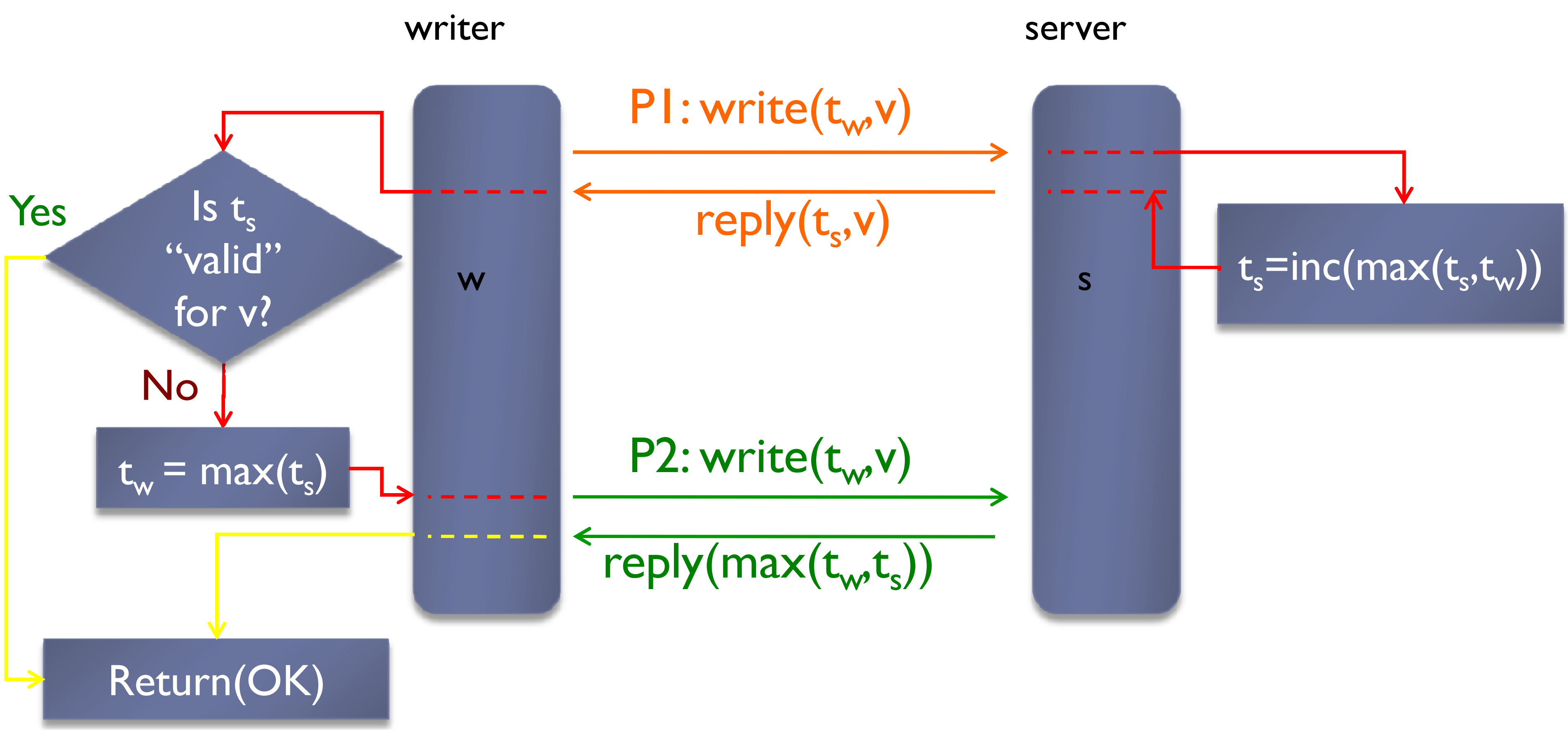
- SSO: Server Side Ordering
  - Tag is incremented by the servers and not by the writer.
    - Generated tags may be different across servers
    - Clients decide operation ordering based on server responses

- SFW Algorithm
  - Enables Fast Writes and Reads -- first such algorithm
  - Allows Unbounded Participation

Traditional Writer-Server Interaction



SFW Writer-Server Interaction



Algorithm: SFW

Write Protocol: one or two rounds

- P1: send v and gather candidate tags from a quorum
  - Exists tag t propagated in a bigger than (n/2-1)-wise intersection
    - YES – assign t to the written value and return => **FAST**
    - NO - propagate the unique largest tag to a quorum => **SLOW**

Read Protocol: one or two rounds

- P1: collect list of writes and their tags from a quorum
  - Exists max write tag t in a bigger than (n/2-2)-wise intersection
    - YES – return the value written by that write => **FAST**
    - NO - propagate the largest confirmed tag to a quorum => **SLOW**

Server Protocol

- Increment tag when receive write request and record the latest writes
- Upon read/write request send the recording set

Lower Bounds (Definitions)

- Consecutive operations:
  - Invoked by different processes
  - They are complete
  - They are not concurrent
- Quorum Shifting operation set Π:
  - Any π_1,π_2 in Π are consecutive
  - if π_1 contacts Q and π_2 contacts Q' then Q not equal to Q'

Lower bounds

Theorem: No execution of safe register implementation that use an N-wise quorum system, contains more than $N-1$ consecutive, quorum shifting, fast writes.

Theorem: It is impossible to get MWMR safe register implementations that exploit an N-wise quorum system, if

$$|W \cup R| > N-1$$

Remarks

<u>Remark 1</u>: SFW algorithm is near optimal since it allows less than $N/2$ consecutive, quorum shifting fast writes.

<u>Remark 2:</u> Our participation bound applies in previously presented fast implementations (i.e. [Dutta et al. 04]).

Contributions – Trade offs in SWMR

Traded Speed for Scalability

- Semifast Implementations - Algorithm SF
- Unbounded Number of Readers
- Single slow read per write
- Impossible if V<|S|/f – 2
- Impossible in the MWMR setting

Traded Speed for Fault-Tolerance

- (Semi)Fast Implementations Not Fault-Tolerant
  - Common Intersection among Quorums
- Weak Semifast Implementations – Algorithm SLIQ
- Quorum Views – general quorum systems

Contributions – Trade offs in MWMR

Traded Write Speed for Fast Reads

- Quorum Views in the MWMR environment
- Algorithm CWFR
  - Traditional two round writes
  - Some single round reads – even when reads are concurrent with writes
  - Utilizes any General Quorum System

Traded Quorum Generality for Fast Writes

- Algorithm SFW
  - Server Side Ordering
  - Allows both single round reads and writes in MWMR
  - Fastness depends on n-wise quorum intersections
- n-1 consecutive fast writes are possible in MWMR
  - SFW near optimal – Allows O(n/2) consecutive fast writes

What's Next

- Dynamism
  - Dynamic Systems
  - Partitionable Networks
- Byzantine Failures
  - Replica Hosts
  - Clients
- Partially Synchronous Environments

List of References

1. Chryssis Georgiou, Nicolas C. Nicolaou, Alexander Russell, and Alexander A. Shvartsman, **Towards Feasible Implementations of Low-Latency Multi-Writer Atomic Registers,** *Tech Report, University of Cyprus and under revision in NCA2011*
2. Chryssis Georgiou, Nicolas C. Nicolaou, and Alexander A. Shvartsman, **Fault-Tolerant SemiFast Implementations of Atomic Read/Write Registers,** *in Journal of Parallel and Distributed Computing (JPDC), 69(1): 62-79, Elsevier, 2009.*
3. Burkhard Englert, Chryssis Georgiou, Peter Musial, Nicolas Nicolaou, and Alexander A. Shvartsman: **On the Efficiency of Atomic Multi-Reader, Multi-Writer Distributed Memory,** *in Proceedings of the 13th International Conference on Principles of Distributed Systems (OPODIS 2009), Nimes, France, 2009.*
4. Chryssis Georgiou, Sotirios Kentros, Nicolas Nicolaou, and Alexander A. Shvartsman:**Analyzing the Number of Slow Reads for Semifast Atomic Read/Write Register Implementations,** *in the Proceedings of the 21st International Conference on Parallel and Distributed Computing and Systems (PDCS 2009), pages 229-236, Cambridge, MA, 2009.*
5. Chryssis Georgiou, Nicolas Nicolaou, and Alexander A. Shvartsman, **On the Robustness of (Semi)Fast Quorum-Based Implementations of Atomic Shared Memory,** *in Proceedings of the 22nd International Symposium on Distributed Computing (DISC 2008), pages 289-304, Arcachon, France, 2008.*
6. K.M. Konwar, P.M. Musial, N.C. Nicolaou, A.A. Shvartsman: **Implementing Atomic Data through Indirect Learning in Dynamic Networks.** *In Proceedings of 6th IEEE International Symposium on Network Computing and Applications (IEEE NCA 2007), pages 223-230, 2007.*
7. Chryssis Georgiou, Nicolas C. Nicolaou, Alexander A. Shvasrtsman, **Fault-Tolerant SemiFast Implementations of Atomic Read/Write Registers.** *In Proceedings of the 18th annual ACM symposium on Parallelism in Algorithms and Architectures (SPAA 2006), pages 281-290, 2006.*

Other Publications

Parallel Systems

1. Sotiris Kentros, Aggelos Kiayias, Nicolas Nicolaou and Alexander A. Shvartsman: ***At-Most-Once Semantics in Asynchronous Shared Memory***, in Proceedings of the 22nd International Symposium on Distributed Computing (DISC 2009), pages 258-273, Elche, Spain, 2009.

Sensor Networks

1. Peng Xie, Zhong Zhou, Nicolas Nicolaou, Andrew See, Jun-Hong Cui, and Zhijie Shi: ***Efficient Vector-Based Forwarding for Underwater Sensor Networks,*** in EURASIP Journal on Wireless Communications and Networking, vol. 2010, Article ID 195910, 13 pages, 2010
2. Nicolas C. Nicolaou, Andrew G. See, Peng Xie, Jun Hong Cui, Dario Maggiorini**:** ***Improving the Robustness of Location-Based Routing for Underwater Sensor Networks***, in Proceedings of IEEE OCEANS'07, Aberdeen, Scotland, pages 1-6, 2007.

E-Voting

1. T. Antonyan, N. Nicolaou, A. Shvartsman and T. Smith: ***Determining the Causes of AccuVote Optical Scan Voting Terminal Memory Card Failures,*** in Online Proceedings of Electronic Voting Technology Workshop/Workshop of Trustworthy Elections (EVT/WOTE'10), Washington DC, USA, 2010.
2. Tigran Antonyan, Seda Davtyan, Sotiris Kentros, Aggelos Kiayias, Laurent Michel, Nicolas Nicolaou, Alexander Russell and Alexander A. Shvartsman: ***State-Wide Elections, Optical Scan Voting and the Pursuit of Integrity***, in IEEE Trans. on Information Forensics and Security (Special Issue on Electronic Voting), Volume 4, No 4, pages 597-610, IEEE, December 2009
3. Tigran Antonyan, Seda Davtyan, Sotiris Kentros, Aggelos Kiayias, Laurent Michel, Nicolas Nicolaou, Alexander Russell and Alexander Shvartsman: ***Automating Voting Terminal Event Log Analysis***, in Online Proceedings of Electronic Voting Technology Workshop/Workshop of Trustworthy Elections (EVT/WOTE'09), Montreal, Canada, 2009.
4. Seda Davtyan, Sotiris Kentros, Aggelos Kiayias, Laurent Michel, Nicolas Nicolaou, Alexander Russell, Andrew See, Narasimha Shashidhar, Alexander A. Shvartsman: ***Taking Total Control of Voting Systems: Firmware Manipulations on an Optical Scan Voting Terminal***, in Proceedings of the 24th Annual ACM Symposium on Applied Computing (SAC '09), pages 2049-2053, March 9-12, Honolulu, Hawaii, 2009.
5. Seda Davtyan, Sotiris Kentros, Aggelos Kiayias, Laurent Michel, Nicolas Nicolaou, Alexander Russell, Andrew See, Narasimha Shashidhar, Alexander A. Shvartsman: ***Pre-Election Testing and Post-Election Audit of Optical Scan Voting Terminal Memory Cards***, in Proceedings of the 2008 Electronic Voting Technology Workshop (EVT '08), July 28–29, San Jose, 2008.