WHAT WE TALK ABOUT WHEN WE TALK ABOUT DISTRIBUTED SYSTEMS

ALVARO VIDELA

DISTRIBUTED SYSTEMS FOR THE IKEA FAMILY

ALVARO VIDELA

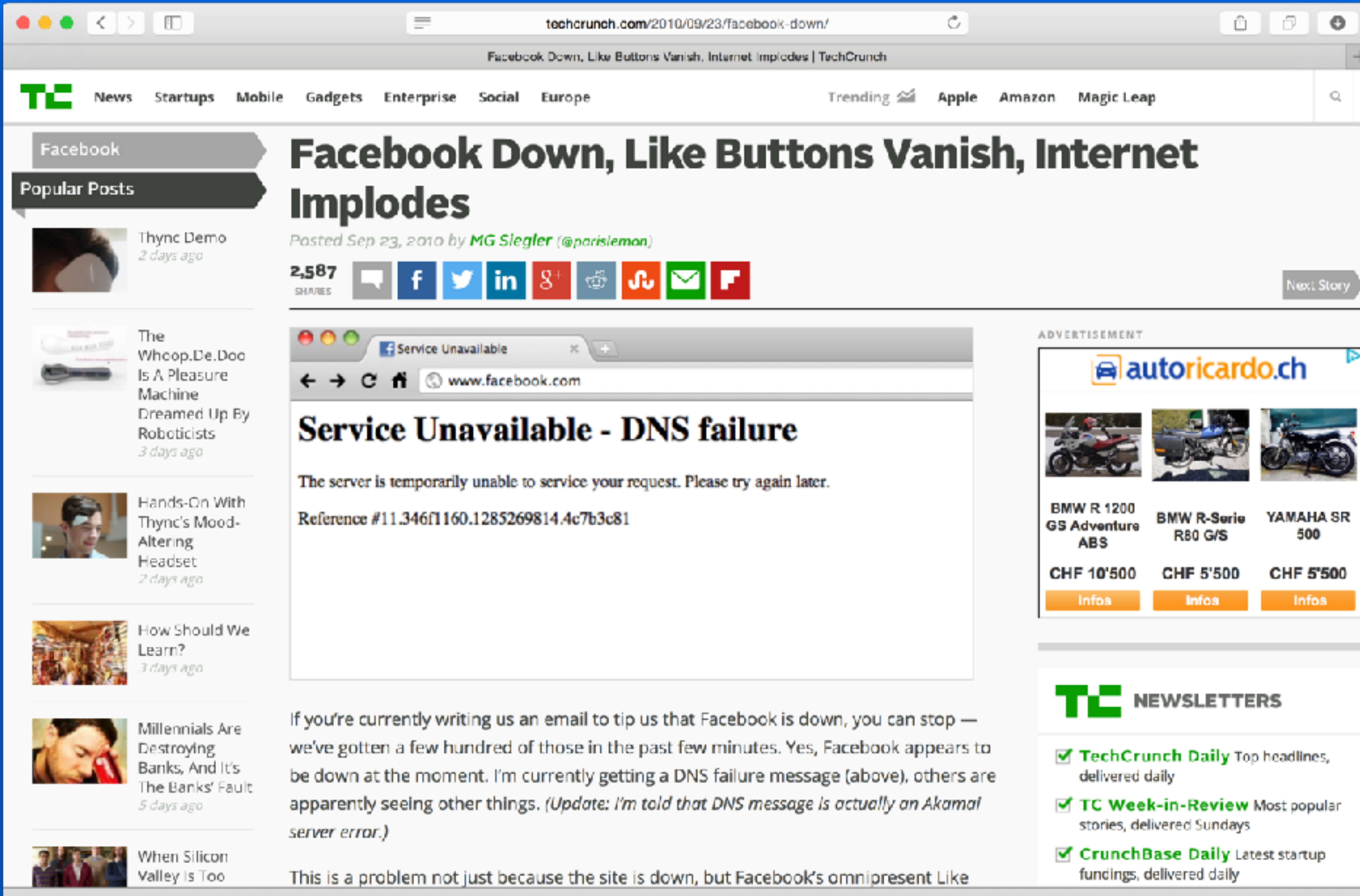
HTTP://BIT.LY/DIST-SYS101

@HINTJENS

DISTRIBUTED SYSTEMS

"A DISTRIBUTED SYSTEM IS ONE IN WHICH THE FAILURE OF A COMPUTER YOU DID NOT EVEN KNOW EXISTED CAN RENDER YOUR OWN COMPUTER UNUSABLE"

Leslie Lamport



techcrunch.com/2010/09/23/facebook-down/

Facebook Down, Like Buttons Vanish, Internet Implodes | TechCrunch

News Startups Mobile Gadgets Enterprise Social Europe Trending Apple Amazon Magic Leap

Facebook

Popular Posts

Thync Demo
2 days ago

The Whoop.De.Doo Is A Pleasure Machine Dreamed Up By Roboticists
3 days ago

Hands-On With Thync's Mood-Altering Headset
2 days ago

How Should We Learn?
3 days ago

Millennials Are Destroying Banks, And It's The Banks' Fault
5 days ago

When Silicon Valley Is Too

Facebook Down, Like Buttons Vanish, Internet Implodes

Posted Sep 23, 2010 by MG Siegler (@parislemon)

2,587 SHARES

Next Story

Service Unavailable

www.facebook.com

Service Unavailable - DNS failure

The server is temporarily unable to service your request. Please try again later.

Reference #11.346f1160.1285269814.4c7b3c81

If you're currently writing us an email to tip us that Facebook is down, you can stop — we've gotten a few hundred of those in the past few minutes. Yes, Facebook appears to be down at the moment. I'm currently getting a DNS failure message (above), others are apparently seeing other things. *(Update: I'm told that DNS message is actually an Akamai server error.)*

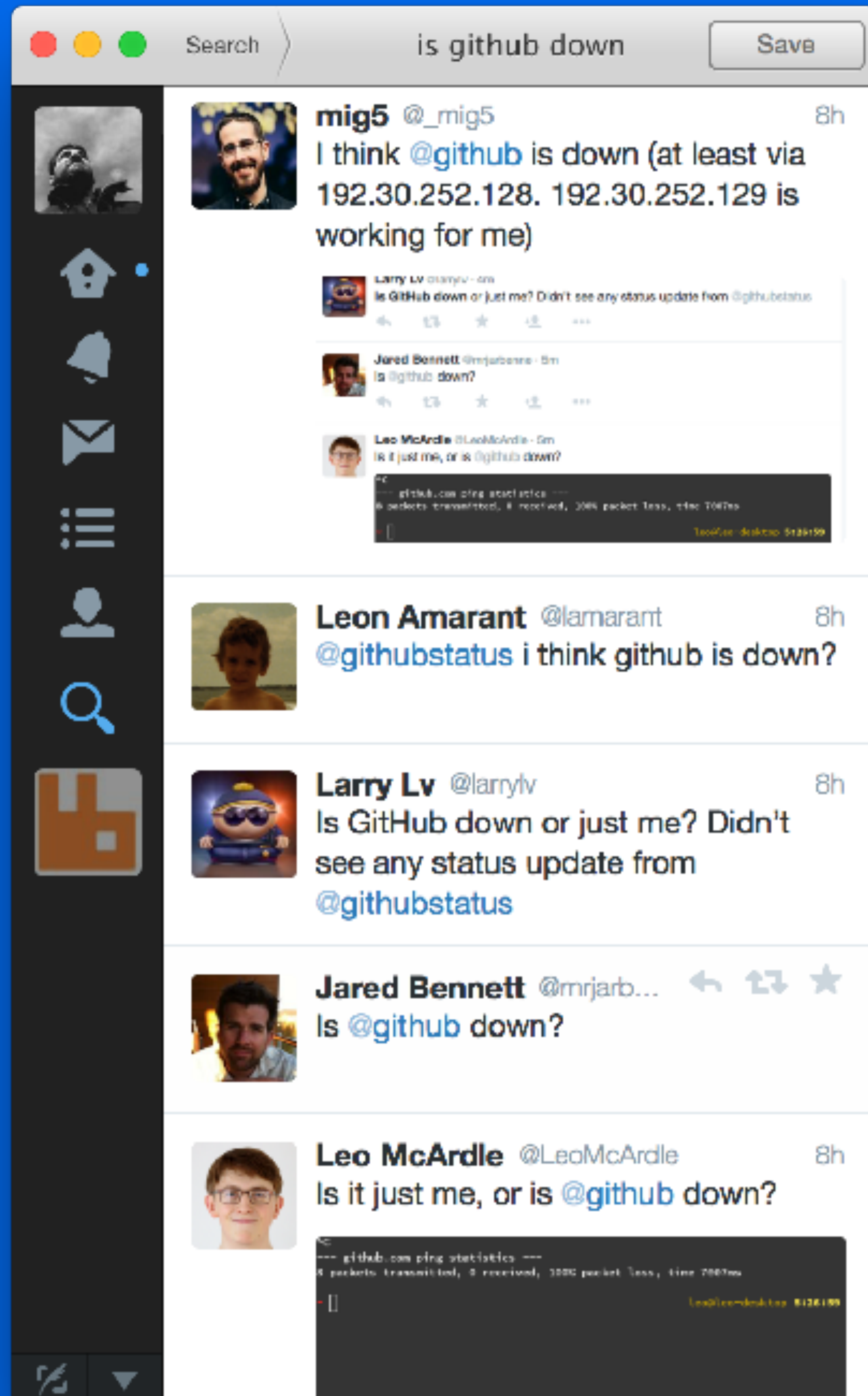
This is a problem not just because the site is down, but Facebook's omnipresent Like

NEWSLETTERS

TechCrunch Daily Top headlines, delivered daily

TC Week-in-Review Most popular stories, delivered Sundays

CrunchBase Daily Latest startup fundings, delivered daily



Search
is github down
Save
mig5 @_mig5 8h
I think @github is down (at least via 192.30.252.128. 192.30.252.129 is working for me)
Leon Amarant @lamarant 8h
@githubstatus i think github is down?
Larry Lv @larrylv 8h
Is GitHub down or just me? Didn't see any status update from @githubstatus
Jared Bennett @mrjarb...
Is @github down?
Leo McArdle @LeoMcArdle 8h
Is it just me, or is @github down?

Google: define jargon

jar·gon[1]

/ˈjärgən/

noun

special words or expressions that are used by a particular profession or group and are difficult for others to understand.

"legal jargon"

synonyms: specialized language, slang, cant, idiom, argot, patter; More

- a form of language regarded as barbarous, debased, or hybrid.

Translations, word origin, and more definitions

DISTRIBUTED SYSTEMS

DISTRIBUTED SYSTEMS

- Many entities trying to solve a problem (nodes, processes)

DISTRIBUTED SYSTEMS

- Many entities trying to solve a problem (nodes, processes)
- Partial Knowledge

DISTRIBUTED SYSTEMS

- Many entities trying to solve a problem (nodes, processes)
- Partial Knowledge
- Uncertainty

DEEP RABBIT HOLE

WHAT TO READ?

WHICH PAPERS?

Impossibility of Distributed Consensus with One Faulty Process[†]

Michael J Fischer
Yale University
New Haven, Connecticut

Nancy A Lynch
Massachusetts Institute of Technology*
Cambridge, Massachusetts

Michael S Paterson
University of Warwick
Coventry, England

Abstract

The consensus problem involves an asynchronous system of processes, some of which may be unreliable The problem is for the reliable processes to agree on a binary value We show that every protocol for this problem has the possibility of nontermination, even with only one faulty process By way of contrast, solutions are known for the synchronous case, the "Byzantine Generals" problem

A well-known form of the problem is the "transaction commit problem" which arises in distributed database systems [DS1, G, LS, La, Le, Li, R, RLS, S, SS] The problem is for all the data manager processes which have participated in the processing of a particular transaction to agree on whether to install the transaction's results in the database or to discard them The latter action might be necessary, for example, if some data managers were for any reason unable to carry out the required transaction processing Whatever decision is made, all data managers must make the same decision in order to preserve the consistency of the database

Impossibility of Distributed Consensus with One Faulty Process†

Michael J Fischer
Yale University
New Haven, Connecticut

Nancy A Lynch
Massachusetts Institute of Technology*
Cambridge, Massachusetts

Michael S Paterson
University of Warwick
Coventry, England

Abstract

The consensus problem involves an asynchronous system of processes, some of which may be unreliable The problem is for the reliable processes to agree on a binary value We show that every protocol for this problem has the possibility of nontermination, even with only one faulty process By way of contrast, solutions are known for the synchronous case, the "Byzantine Generals" problem

A well-known form of the problem is the "transaction commit problem" which arises in distributed database systems [DS1, G, LS, La, Le, Li, R, RLS, S, SS] The problem is for all the data manager processes which have participated in the processing of a particular transaction to agree on whether to install the transaction's results in the database or to discard them The latter action might be necessary, for example, if some data managers were for any reason unable to carry out the required transaction processing Whatever decision is made, all data managers must make the same decision in order to preserve the consistency of

The Part-Time Parliament

LESLIE LAMPORT
Digital Equipment Corporation

Recent archaeological discoveries on the island of Paxos reveal that the parliament functioned despite the peripatetic propensity of its part-time legislators. The legislators maintained consistent copies of the parliamentary record, despite their frequent forays from the chamber and the forgetfulness of their messengers. The Paxon parliament's protocol provides a new way of implementing the state machine approach to the design of distributed systems.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*network operating systems*; D.4.5 [**Operating Systems**]: Reliability—*fault-tolerance*; J.1 [**Computer Applications**]: Administrative Data Processing—*government*

General Terms: Design, Reliability

Additional Key Words and Phrases: State machines, three-phase commit, voting

Impossibility of Distributed Consensus with One Faulty Process†

Michael J Fischer
Yale University
New Haven, Connecticut

Nancy A Lynch
Massachusetts Institute of Technology*
Cambridge, Massachusetts

Michael S Paterson
University of Warwick
Coventry, England

Abstract

A well-known form of the problem is the "transaction commit problem" which arises in
ase systems [DS1, G, LS, La, Le,
] The problem is for all the data
s which have participated in the
particular transaction to agree on
ll the transaction's results in the
discard them The latter action
ary, for example, if some data
r any reason unable to carry out
nsaction processing Whatever
all data managers must make the
rder to preserve the consistency of

Operating Systems — R. Stockton Gaines, Editor

Time, Clocks, and the Ordering of Events in a Distributed System

Leslie Lamport
Massachusetts Computer Associates, Inc.

The concept of one event happening before another in a distributed system is examined, and is shown to define a partial ordering of the events. A distributed algorithm is given for synchronizing a system of logical clocks which can be used to totally order the events. The use of the total ordering is illustrated with a method for solving synchronization problems. The algorithm is then specialized for synchronizing physical clocks, and a bound is derived on how far out of synchrony the clocks can become.

Key Words and Phrases: distributed systems, computer networks, clock synchronization, multiprocess systems

CR Categories: 4.32, 5.29

art-Time Parliament

MPORT
ipment Corporation

eological discoveries on the island of Paxos reveal that the parliament functioned
peripatetic propensity of its part-time legislators. The legislators maintained
pies of the parliamentary record, despite their frequent forays from the chamber
etfulness of their messengers. The Paxon parliament's protocol provides a new
menting the state machine approach to the design of distributed systems.

nd Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distrib-
s—*network operating systems*; D.4.5 [**Operating Systems**]: Reliability—*fault-*
[**Computer Applications**]: Administrative Data Processing—*government*

ns: Design, Reliability

Additional Key Words and Phrases: State machines, three-phase commit, voting

Impossibility of Distributed Consensus wi

Michael J Fischer
Yale University
New Haven, Connecticut

Nancy A Lynch
Massachusetts Institute of Technolo
Cambridge, Massachusetts

Abstract

A well-k
"transaction
...ary, for example, if some data
...r any reason unable to carry out
...nsaction processing Whatever
...all data managers must make the
...rder to preserve the consistency of

A simple totally ordered broadcast protocol

Benjamin Reed
Yahoo! Research
Santa Clara, CA - USA
breed@yahoo-inc.com

Flavio P. Junqueira
Yahoo! Research
Barcelona, Catalunya - Spain
fpj@yahoo-inc.com

ABSTRACT

This is a short overview of a totally ordered broadcast protocol used by ZooKeeper, called Zab. It is conceptually easy to understand, is easy to implement, and gives high performance. In this paper we present the requirements ZooKeeper makes on Zab, we show how the protocol is used, and we give an overview of how the protocol works.

chines providing the service and always has a consistent view of the ZooKeeper state. The service tolerates up to f crash failures, and it requires at least $2f + 1$ servers.

Applications use ZooKeeper extensively and have tens to thousands of clients accessing it concurrently, so we require high throughput. We have designed ZooKeeper for workloads with ratios of read to write operations that are higher than 2:1; however, we have found that ZooKeeper's

Operating Systems

R. Stockton Gaines
Editor

Time, Clocks, and the Ordering of Events in a Distributed System

Leslie Lamport
Massachusetts Computer Associates, Inc.

The concept of one event happening before another in a distributed system is examined, and is shown to define a partial ordering of the events. A distributed algorithm is given for synchronizing a system of logical clocks which can be used to totally order the events. The use of the total ordering is illustrated with a method for solving synchronization problems. The algorithm is then specialized for synchronizing physical clocks, and a bound is derived on how far out of synchrony the clocks can become.

Key Words and Phrases: distributed systems, computer networks, clock synchronization, multiprocess systems

CR Categories: 4.32, 5.29

...art-Time Parliament

...MPORT
...ipment Corporation

...eological discoveries on the island of Paxos reveal that the parliament functioned ... peripatetic propensity of its part-time legislators. The legislators maintained ... pies of the parliamentary record, despite their frequent forays from the chamber ... etfulness of their messengers. The Paxon parliament's protocol provides a new ... menting the state machine approach to the design of distributed systems.

...nd Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distrib-... s—*network operating systems*; D.4.5 [**Operating Systems**]: Reliability—*fault-* ... [**Computer Applications**]: Administrative Data Processing—*government*

...ns: Design, Reliability

Additional Key Words and Phrases: State machines, three-phase commit, voting

Impossibility of Distributed Consensus wi

Michael J Fischer
Yale University
New Haven, Connecticut

Nancy A Lynch
Massachusetts Institute of Technolo
Cambridge, Massachusetts

Abstract

A well-k
"transaction

A simple totally ordered broadcast protocol

Benjamin Reed
Yahoo! Research
Santa Clara, CA - USA
breed@yahoo-inc.com

Flavio P. Junqueira
Yahoo! Research
Barcelona, Catalunya - Spain
fpj@yahoo-inc.com

ABSTRACT

...rvice and always has a consistent view
e. The service tolerates up to f crash
s at least $2f + 1$ servers.
ooKeeper extensively and have tens
s accessing it concurrently, so we re-
t. We have designed ZooKeeper for
of read to write operations that are
ver, we have found that ZooKeeper's

Operating
Systems

Time, Clo
Ordering
a Distribu

Leslie Lamport
Massachusetts Com

The concept of one
in a distributed system i
define a partial ordering
algorithm is given for sy
clocks which can be used to totally order the events.
The use of the total ordering is illustrated with a method for solving synchronization problems. The algorithm is then specialized for synchronizing physical clocks, and a bound is derived on how far out of synchrony the clocks can become.

Key Words and Phrases: distributed systems, computer networks, clock synchronization, multiprocess systems

CR Categories: 4.32, 5.29

In Search of an Understandable Consensus Algorithm

Diego Ongaro and John Ousterhout
Stanford University

(Draft of May 22, 2013; under submission)

Abstract

Raft is a consensus algorithm for managing a replicated log. It produces a result equivalent to Paxos, and it is as efficient as Paxos, but its structure is different from Paxos; this makes Raft more understandable than Paxos and also provides a better foundation for building practical systems. In order to enhance understandability, Raft separates the key elements of consensus, such as leader election and log replication, and it enforces a stronger degree of coherency to reduce the number of states that must be considered. Raft also includes a new mechanism for changing the cluster membership, which uses overlapping majorities to guarantee safety. Results from a user study demonstrate that Raft is easier for students to learn than Paxos.

was our most important criterion in evaluating design alternatives. We applied specific techniques to improve understandability, including decomposition (Raft separates leader election, log replication, and safety so that they can be understood relatively independently) and state space reduction (Raft reduces the degree of nondeterminism and the ways servers can be inconsistent with each other, in order to make it easier to reason about the system).

- **Strong leader:** Raft differs from other consensus algorithms in that it employs a strong form of leadership where only leaders (or would-be leaders) issue requests; other servers are completely passive. This makes Raft easier to understand and also simplifies the implementation.

...eological discoveries on the island of Paxos reveal that the parliament functioned
peripatetic propensity of its part-time legislators. The legislators maintained
pies of the parliamentary record, despite their frequent forays from the chamber
etfulness of their messengers. The Paxon parliament's protocol provides a new
menting the state machine approach to the design of distributed systems.

nd Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distrib-
s—*network operating systems*; D.4.5 [**Operating Systems**]: Reliability—*fault-*
[**Computer Applications**]: Administrative Data Processing—*government*

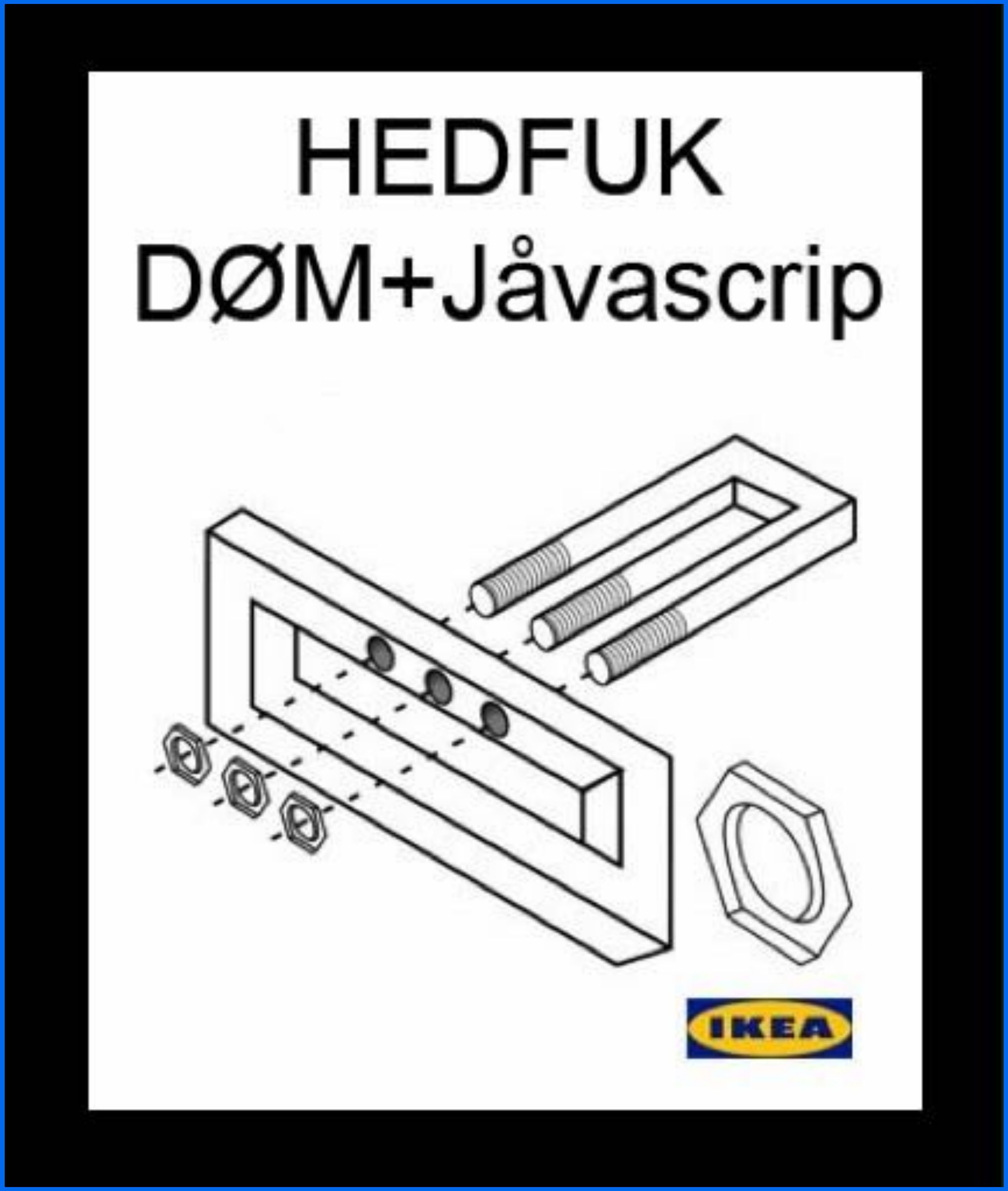
ns: Design, Reliability

Additional Key Words and Phrases: State machines, three-phase commit, voting

WHICH BOOKS?


ACTORS
GUL AGHA
FAULT-TOLERANT REAL-TIME SYSTEMS
Poledna
KAP
Programming Distributed Computing Systems
Varela
Herlihy
Shavit
THE ART of MULTIPROCESSOR PROGRAMMING
REVISED FIRST EDITION
MK
DISTRIBUTED COMPUTING through COMBINATORIAL TOPOLOGY
Herlihy
Kozlov
Rajsbaum
MK
RAYNAL
COMMUNICATION AND AGREEMENT ABSTRACTIONS FOR FAULT-TOLERANT ASYNCHRONOUS DISTRIBUTED SYSTEMS
MORGAN&CLAYPOOL
RAYNAL
FAULT-TOLERANT AGREEMENT IN SYNCHRONOUS MESSAGE-PASSING SYSTEMS
MORGAN&CLAYPOOL
VUKOLIC
QUORUM SYSTEMS
MORGAN & CLAYPOOL
REPLICATION TECHNIQUES IN DISTRIBUTED SYSTEMS
UB-980-387
1996
TCS
Birman
Guide to Reliable Distributed Systems
Charron-Bost • Pedone
Schiper (Eds.)
LNCS
5959
Replication
Cachin · Guerraoui
Rodrigues
Introduction to Reliable and
Secure Distributed Programming
2nd Ed.
Raynal
Distributed Algorithms for Message-Passing Systems
Raynal
Concurrent Programming: Algorithms,
Principles, and Foundations
Lynch
Distributed Algorithms
MORGAN
KAUFMANN

WHY?



http://tobielangel.com

THE PROBLEM

DIFFERENT MODELS

DIFFERENT MODELS

- Timing Model

DIFFERENT MODELS

- Timing Model
- Inter Process Communication Used (IPC method)

DIFFERENT MODELS

- Timing Model
- Inter Process Communication Used (IPC method)
- Failure Modes

TIMING MODEL

TIMING MODEL

- Synchronous Model

TIMING MODEL

- Synchronous Model
- Asynchronous Model

TIMING MODEL

- Synchronous Model
- Asynchronous Model
- Semi-synchronous Model

INTERPROCESS COMMUNICATION

INTERPROCESS COMMUNICATION

- Message Passing

INTERPROCESS COMMUNICATION

- Message Passing
- Shared Memory

FAILURE MODES

FAILURE MODES

- Crash-stop

FAILURE MODES

- Crash-stop
- Crash-recovery

FAILURE MODES

- Crash-stop
- Crash-recovery
- Omission Faults

FAILURE MODES

- Crash-stop
- Crash-recovery
- Omission Faults
- Arbitrary Failures Mode (Byzantine)

LIVENESS AND SAFETY

LIVENESS AND SAFETY PROPERTIES OF ALGORITHMS

DEFINING LIVENESS *

Bowen ALPERN and Fred B. SCHNEIDER

Department of Computer Science, Cornell University, 405 Upson Hall, Ithaca, NY 14853, U.S.A.

Communicated by David Gries
Received 5 November 1984
Revised 20 February 1985

A formal definition for liveness properties is proposed. It is argued that this definition captures the intuition that liveness properties stipulate that 'something good' eventually happens during execution. A topological characterization of safety and liveness is given. Every property is shown to be the intersection of a safety property and a liveness property.

SAFETY

Some "bad" thing does not happens during execution

SAFETY

"Communication links should not invent messages out of thin air"

LIVENESS

A “good” thing happens during execution

LIVENESS

"A destination process eventually delivers the message"

LET'S TAKE A LOOK AT FLP[1]

1 - Fischer, Lynch, Paterson

Impossibility of Distributed Consensus with One Faulty Process†

Michael J Fischer
Yale University
New Haven, Connecticut

Nancy A Lynch
Massachusetts Institute of Technology*
Cambridge, Massachusetts

Michael S Paterson
University of Warwick
Coventry, England

Abstract

The consensus problem involves an asynchronous system of processes, some of which may be unreliable The problem is for the reliable processes to agree on a binary value We show that every protocol for this problem has the possibility of nontermination, even with only one faulty process By way of contrast, solutions are known for the synchronous case, the "Byzantine Generals" problem

A well-known form of the problem is the "transaction commit problem" which arises in distributed database systems [DS1, G, LS, La, Le, Li, R, RLS, S, SS] The problem is for all the data manager processes which have participated in the processing of a particular transaction to agree on whether to install the transaction's results in the database or to discard them The latter action might be necessary, for example, if some data managers were for any reason unable to carry out the required transaction processing Whatever decision is made, all data managers must make the same decision in order to preserve the consistency of the database

IMPOSSIBILITY OF DISTRIBUTED CONSENSUS WITH ONE FAULTY PROCESS

In this paper, we show the surprising result that no completely asynchronous consensus protocol can tolerate even a single unannounced process death We do not consider Byzantine failures, and we assume that the message system is reliable — it delivers all messages correctly and exactly once

IMPOSSIBILITY OF DISTRIBUTED CONSENSUS WITH ONE FAULTY PROCESS

Nevertheless, even with these assumptions, the stopping of a single process at an inopportune time can cause any distributed commit protocol to fail to reach agreement Thus, this important problem has no robust solution without further assumptions about the computing environment or still greater restrictions on the kind of failures to be tolerated!

IMPOSSIBILITY OF DISTRIBUTED CONSENSUS WITH ONE FAULTY PROCESS

Crucial to our proof is that processing is completely asynchronous, that is, we make no assumptions about the relative speeds of processes nor about the delay time in delivering a message We also assume that processes do not have access to synchronized clocks, so algorithms based on timeouts, for example, cannot be used (In particular, the solutions in [DS1] are not applicable) Finally, we do not postulate the ability to detect the death of a process, so it is impossible for one process to tell whether another has died (stopped entirely) or is just running very slowly

IMPOSSIBILITY OF DISTRIBUTED CONSENSUS WITH ONE FAULTY PROCESS

Our system model is rather strong so as to make our impossibility proof as widely applicable as possible Processes are modelled as automata (with possibly infinitely many states) which communicate by means of messages In one atomic step, a process can attempt to receive a message, perform local computation based on whether or not a message was delivered to it and if so on which one, and send an arbitrary but finite set of messages to other processes In particular, an "atomic broadcast" capability is assumed, so a process can send the same message in one step to all other processes with the knowledge that if any nonfaulty process receives the message, then all the nonfaulty processes will

IMPOSSIBILITY OF DISTRIBUTED CONSENSUS WITH ONE FAULTY PROCESS

> Every message is eventually delivered as long as the destination process makes infinitely many attempts to receive, but messages can be delayed arbitrarily long and delivered out of order

WHAT’S CONSENSUS ANYWAY?

"THE CONSENSUS PROBLEM IS A PARADIGM OF AGREEMENT PROBLEMS"

https://dl.acm.org/citation.cfm?id=1052796.1052806

PROPERTIES OF CONSENSUS

PROPERTIES OF CONSENSUS

- **C-Termination**: Every correct process eventually decides on some value

PROPERTIES OF CONSENSUS

- **C-Termination**: Every correct process eventually decides on some value
- **C-Validity**: If a process decides ***v***, then ***v*** was proposed by some process

PROPERTIES OF CONSENSUS

- **C-Termination**: Every correct process eventually decides on some value
- **C-Validity**: If a process decides ***v***, then ***v*** was proposed by some process
- **C-Agreement**: No two correct processes decide differently

PROPERTIES OF UNIFORM CONSENSUS

- **C-Termination**: Every correct process eventually decides on some value
- **C-Validity**: If a process decides ***v***, then ***v*** was proposed by some process
- **C-Agreement**: No two correct processes decide differently
- **C-Uniform Agreement**: No two processes (correct or not) decide differently.

WE NEED CONSENSUS WHEN:

A SET OF PROCESSES HAVE TO AGREE TO TAKE A COMMON ACTION

WE NEED CONSENSUS WHEN:

A SET OF PROCESSES HAVE TO AGREE TO TAKE A COMMON ACTION

Atomic Broadcast

WE NEED CONSENSUS WHEN:

A SET OF PROCESSES HAVE TO AGREE TO TAKE A COMMON ACTION

Atomic Broadcast

Group Membership

ATOMIC BROADCAST

“CORRECT PROCESSES DELIVER THE SAME SET OF MESSAGES IN THE SAME ORDER”

FLP TELLS US THAT IF CONSENSUS CANNOT BE ACHIEVED, THEN ATOMIC BROADCAST OR GROUP MEMBERSHIP CANNOT BE ACHIEVED EITHER

SO, WE PACK OUR BAGS AND GO?

NOTHING TO SEE HERE?

STUMBLING OVER CONSENSUS RESEARCH:

MISUNDERSTANDING AND ISSUES

Marcos K. Aguilera

FAILURE DETECTORS

Unreliable Failure Detectors for Reliable Distributed Systems

TUSHAR DEEPAK CHANDRA

I.B.M. Thomas J. Watson Research Center, Hawthorne, New York

AND

SAM TOUEG

Cornell University, Ithaca, New York

We introduce the concept of unreliable failure detectors and study how they can be used to solve Consensus in asynchronous systems with crash failures. We characterise unreliable failure detectors in terms of two properties—completeness and accuracy. We show that Consensus can be solved even with unreliable failure detectors that make an infinite number of mistakes, and determine which ones can be used to solve Consensus despite any number of crashes, and which ones require a majority of correct processes. We prove that Consensus and Atomic Broadcast are reducible to each other in asynchronous systems with crash failures; thus, the above results also apply to Atomic Broadcast. A companion paper shows that one of the failure detectors introduced here is the weakest failure detector for solving Consensus [Chandra et al. 1992].

FAILURE DETECTORS

FAILURE DETECTORS

- External process

FAILURE DETECTORS

- External process
- Provides information about **suspected** processes

FAILURE DETECTORS

- External process
- Provides information about **suspected** processes
- Completeness property (crashed processes are detected)

FAILURE DETECTORS

- External process
- Provides information about **suspected** processes
- Completeness property (crashed processes are detected)
- Accuracy (correct process are never suspected)

“RUB SOME PERFECT FAILURE DETECTOR ON IT”

PERFECT FAILURE DETECTOR

Module 2.6: Interface and properties of the perfect failure detector

Module:

Name: PerfectFailureDetector, **instance** $\mathcal{P}$.

Events:

Indication: $\langle \mathcal{P}, Crash \mid p \rangle$: Detects that process p has crashed.

Properties:

PFD1: *Strong completeness:* Eventually, every process that crashes is permanently detected by every correct process.

PFD2: *Strong accuracy:* If a process p is detected by any process, then p has crashed.

http://www.amazon.com/Introduction-Reliable-Secure-Distributed-Programming/dp/3642152597

EVENTUALLY ACCURATE FAILURE DETECTOR

EVENTUALLY ACCURATE FAILURE DETECTOR

- **Strong Completeness**: Eventually, every process that crashes is permanently suspected by every correct process.

EVENTUALLY ACCURATE FAILURE DETECTOR

- **Strong Completeness**: Eventually, every process that crashes is permanently suspected by every correct process.
- **Eventual Weak Accuracy**: There is a time after which some correct process is never suspected by the correct processes.

EVENTUALLY ACCURATE FAILURE DETECTOR

- **Strong Completeness**: Eventually, every process that crashes is permanently suspected by every correct process.
- **Eventual Weak Accuracy**: There is a time after which some correct process is never suspected by the correct processes.

http://dl.acm.org/citation.cfm?id=1052806

Unreliable Failure Detectors for Reliable Distributed Systems

TUSHAR DEEPAK CHANDRA

I.B.M. Thomas J. Watson Research Center, Hawthorne, New York

AND

SAM TOUEG

Cornell University, Ithaca, New York

We introduce the concept of unreliable failure detectors and study how they can be used to solve Consensus in asynchronous systems with crash failures. We characterise unreliable failure detectors in terms of two properties—completeness and accuracy. We show that Consensus can be solved even with unreliable failure detectors that make an infinite number of mistakes, and determine which ones can be used to solve Consensus despite any number of crashes, and which ones require a majority of correct processes. We prove that Consensus and Atomic Broadcast are reducible to each other in asynchronous systems with crash failures; thus, the above results also apply to Atomic Broadcast. A companion paper shows that one of the failure detectors introduced here is the weakest failure detector for solving Consensus [Chandra et al. 1992].

QUORUMS

TL;DR:

INTERSECTING SETS

QUORUMS

"A QUORUM IN A SYSTEM WITH N CRASH-FAULT PROCESS ABSTRACTIONS [...] IS ANY MAJORITY OF PROCESSES, I.E., ANY SET OF MORE THAN N/2 PROCESSES"

QUORUMS

“IF F < N/2 PROCESSES FAIL BY CRASHING, THERE IS ALWAYS AT LEAST ONE QUORUM OF NONCRASHED PROCESSES IN SUCH SYSTEMS”

QUORUMS

A - B - C - D - E

QUORUMS

A - ~~B~~ - C - ~~D~~ - E

QUORUMS

A - B' - C - D' - E

QUORUMS

A – B' – ~~C~~ – D' – ~~E~~

QUORUMS

A - B' - C' - D' - E'

QUORUMS

<u>A</u> - B' - C' - D' - E'

CONSISTENCY

Linearizability: A Correctness Condition for Concurrent Objects

MAURICE P. HERLIHY and JEANNETTE M. WING
Carnegie Mellon University

A concurrent object is a data object shared by concurrent processes. Linearizability is a correctness condition for concurrent objects that exploits the semantics of abstract data types. It permits a high degree of concurrency, yet it permits programmers to specify and reason about concurrent objects using known techniques from the sequential domain. Linearizability provides the illusion that each operation applied by concurrent processes takes effect instantaneously at some point between its invocation and its response, implying that the meaning of a concurrent object's operations can be given by pre- and post-conditions. This paper defines linearizability, compares it to other correctness conditions, presents and demonstrates a method for proving the correctness of implementations, and shows how to reason about concurrent objects, given they are linearizable.

CONCURRENT FIFO QUEUE

CONSISTENCY CONDITIONS

CONSISTENCY CONDITIONS

- Atomic Consistency (Linearizabilty)

CONSISTENCY CONDITIONS

- Atomic Consistency (Linearizabilty)
- Sequential Consistency

CONSISTENCY CONDITIONS

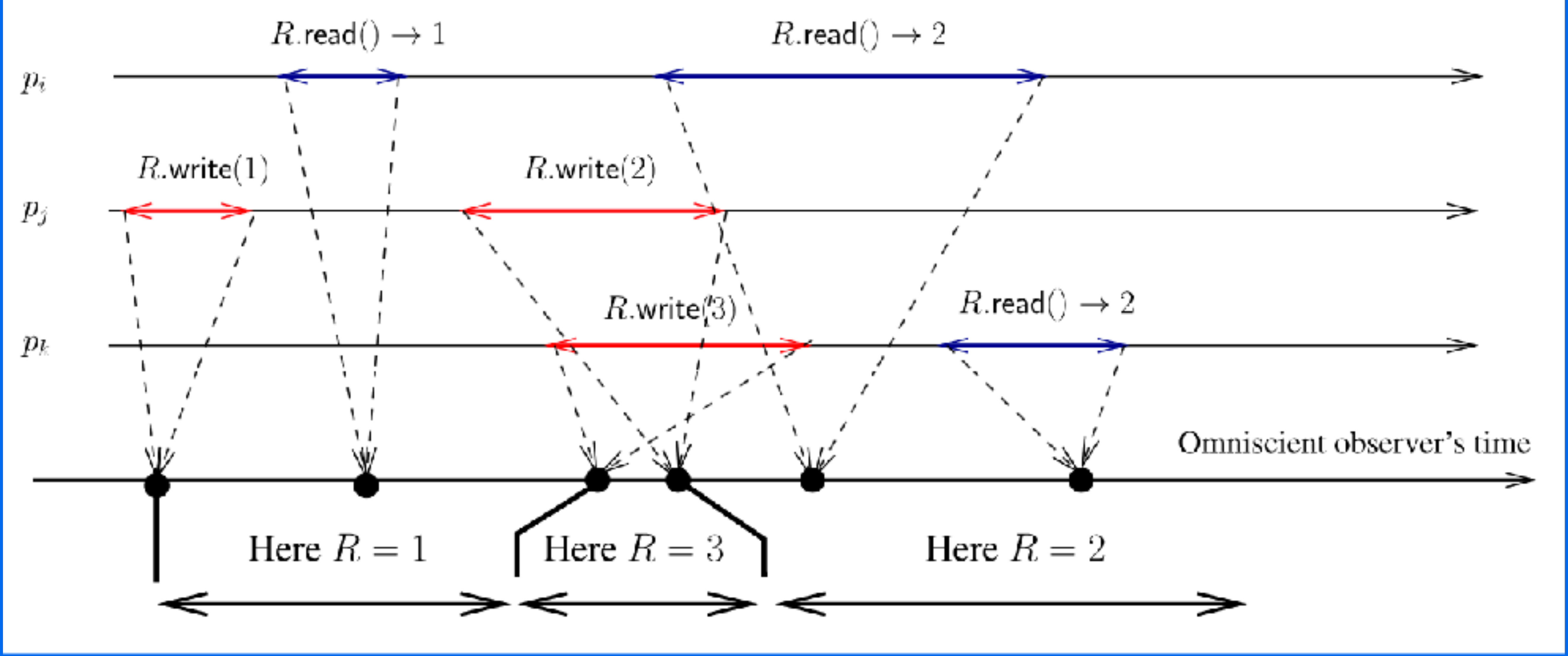
- Atomic Consistency (Linearizabilty)
- Sequential Consistency
- Causal Consistency

CONSISTENCY CONDITIONS

- Atomic Consistency (Linearizabilty)
- Sequential Consistency
- Causal Consistency

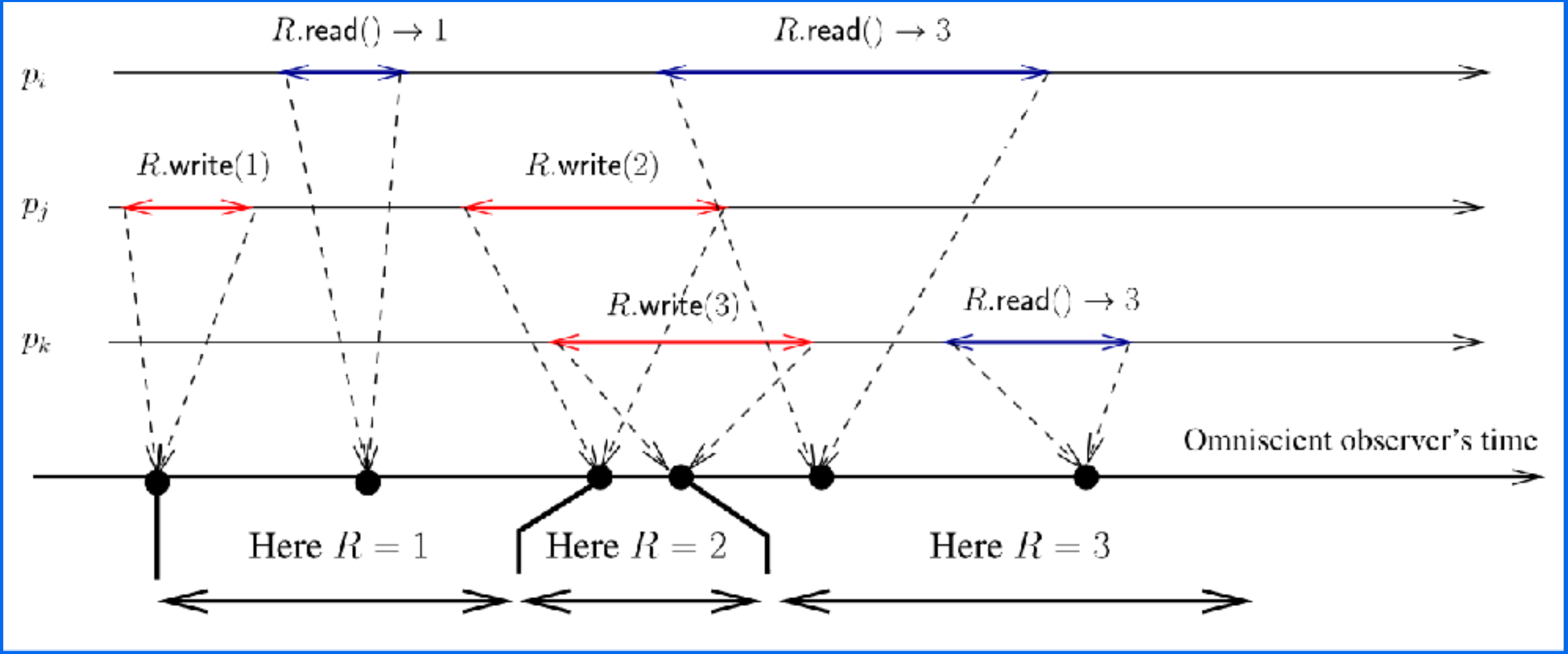
https://aphyr.com/posts/313-strong-consistency-models

LINEARIZABILTY



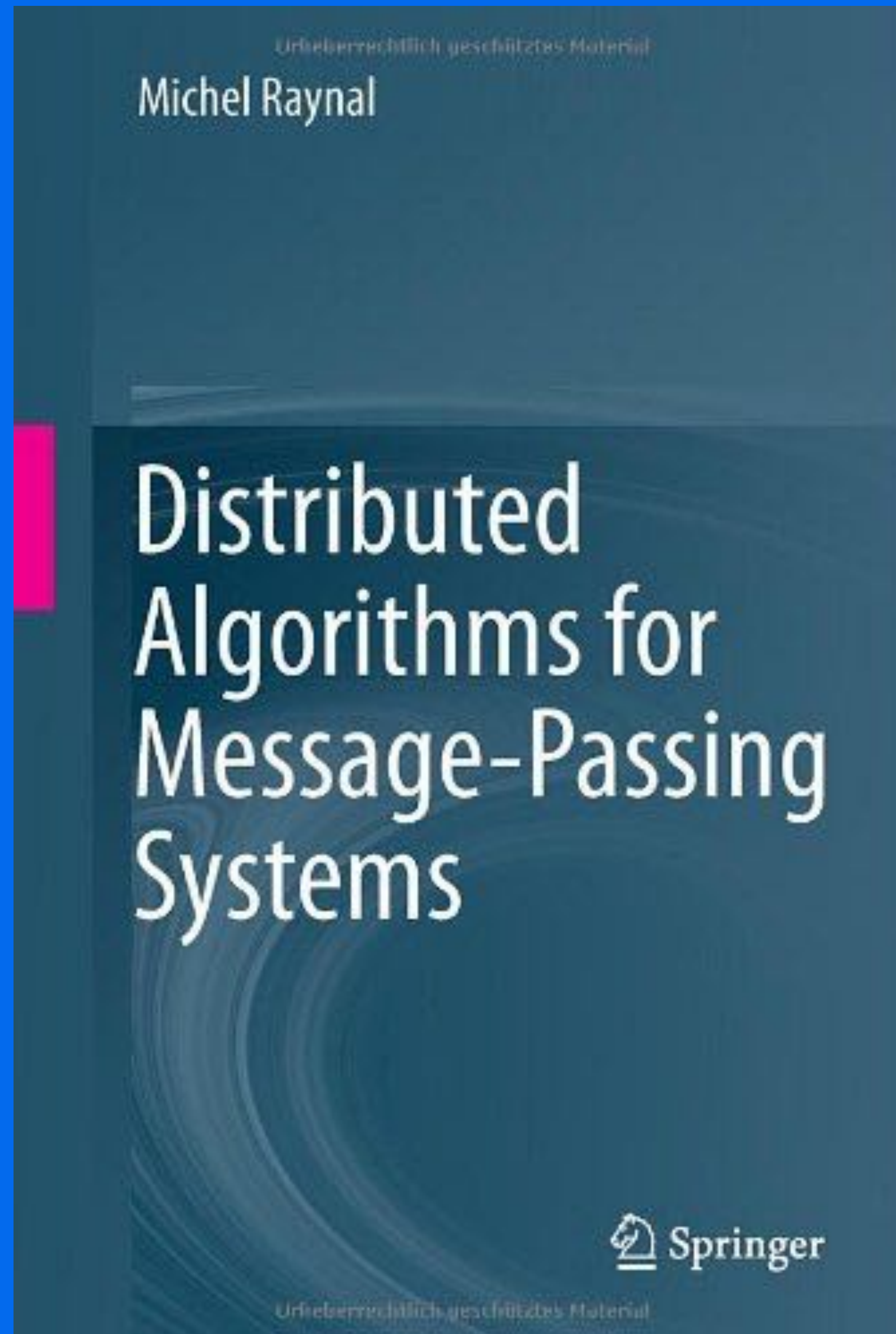
http://www.amazon.com/Distributed-Algorithms-Message-Passing-Systems-Michel/dp/3642381227/

LINEARIZABILTY

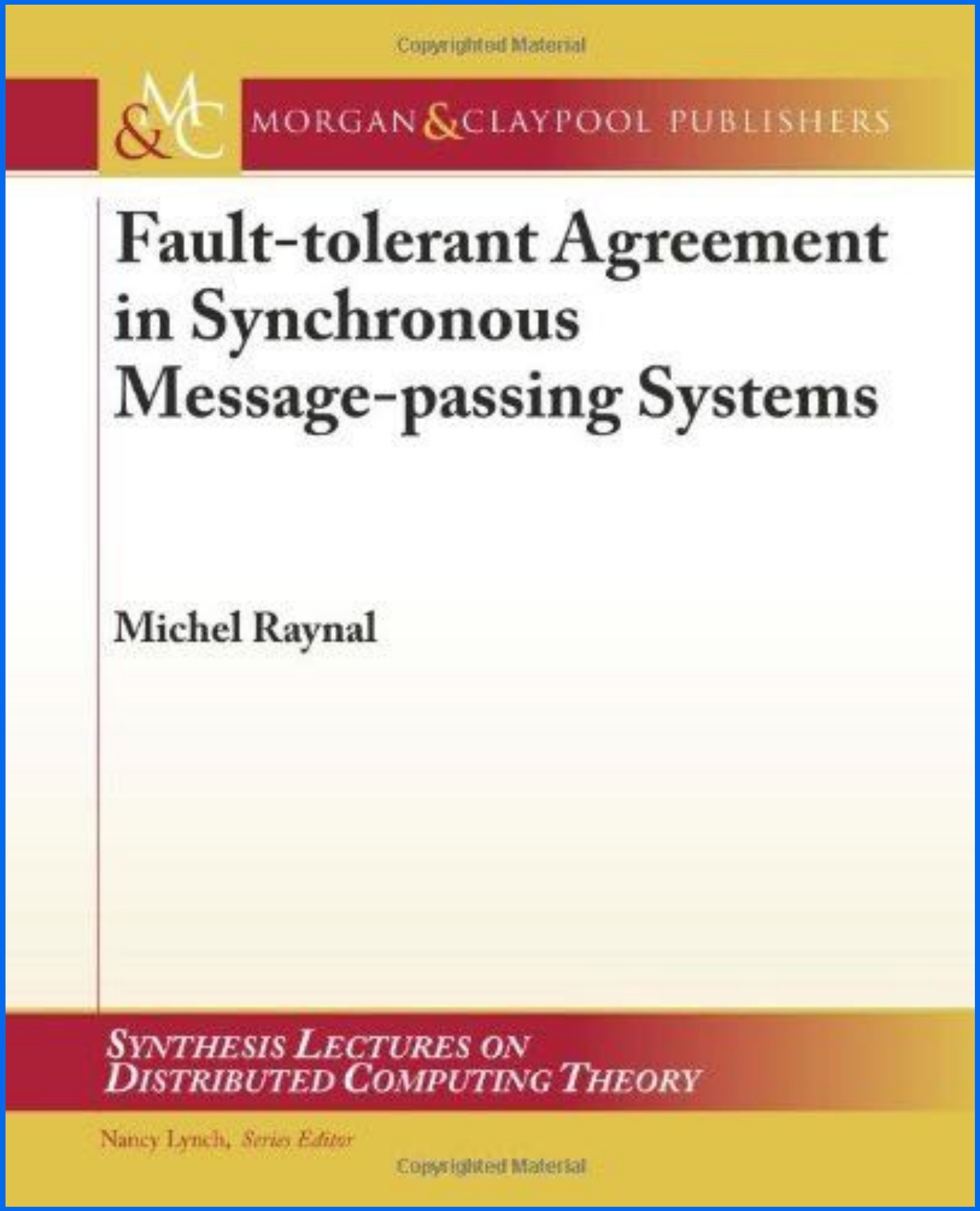


http://www.amazon.com/Distributed-Algorithms-Message-Passing-Systems-Michel/dp/3642381227/

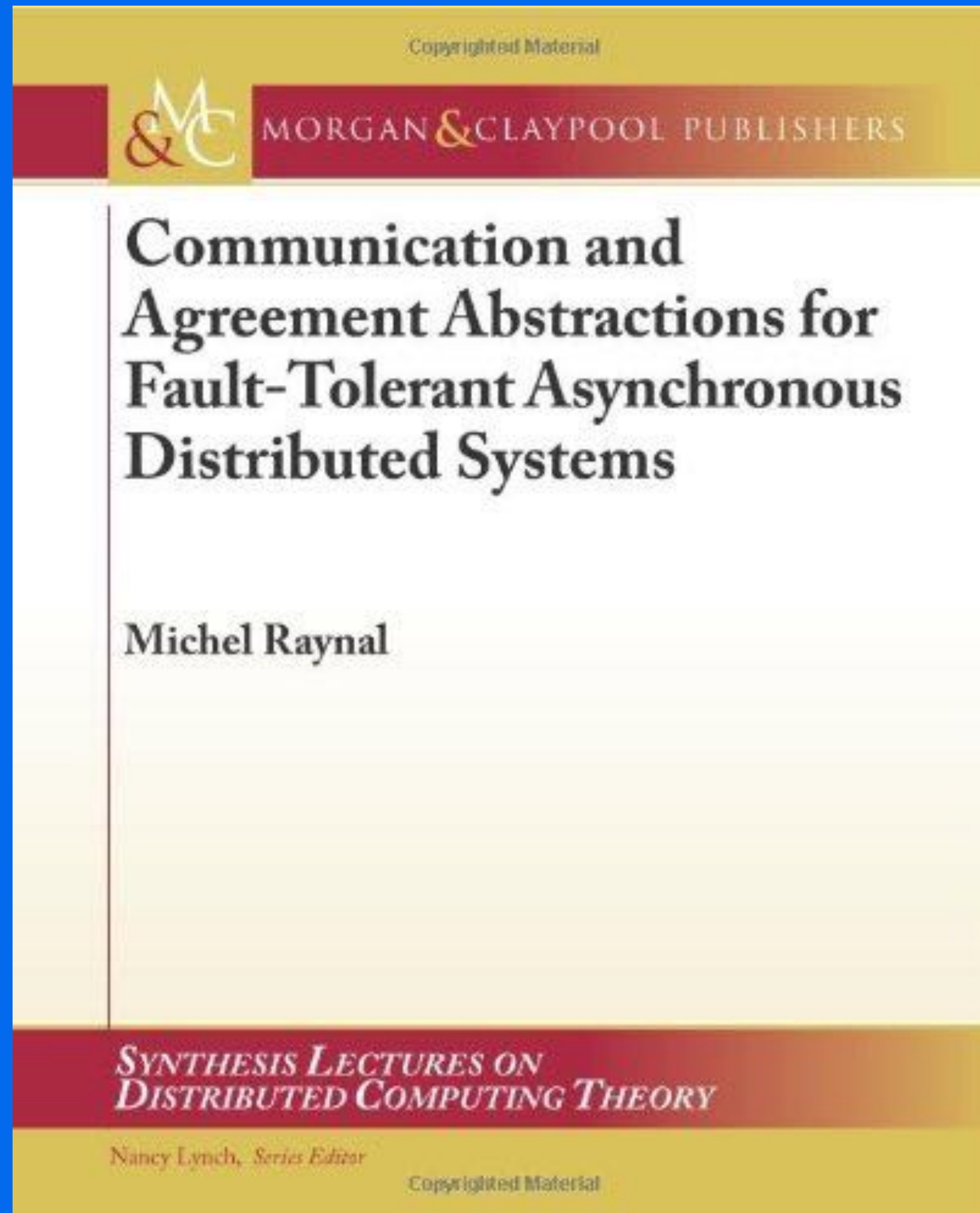
SOME BOOKS



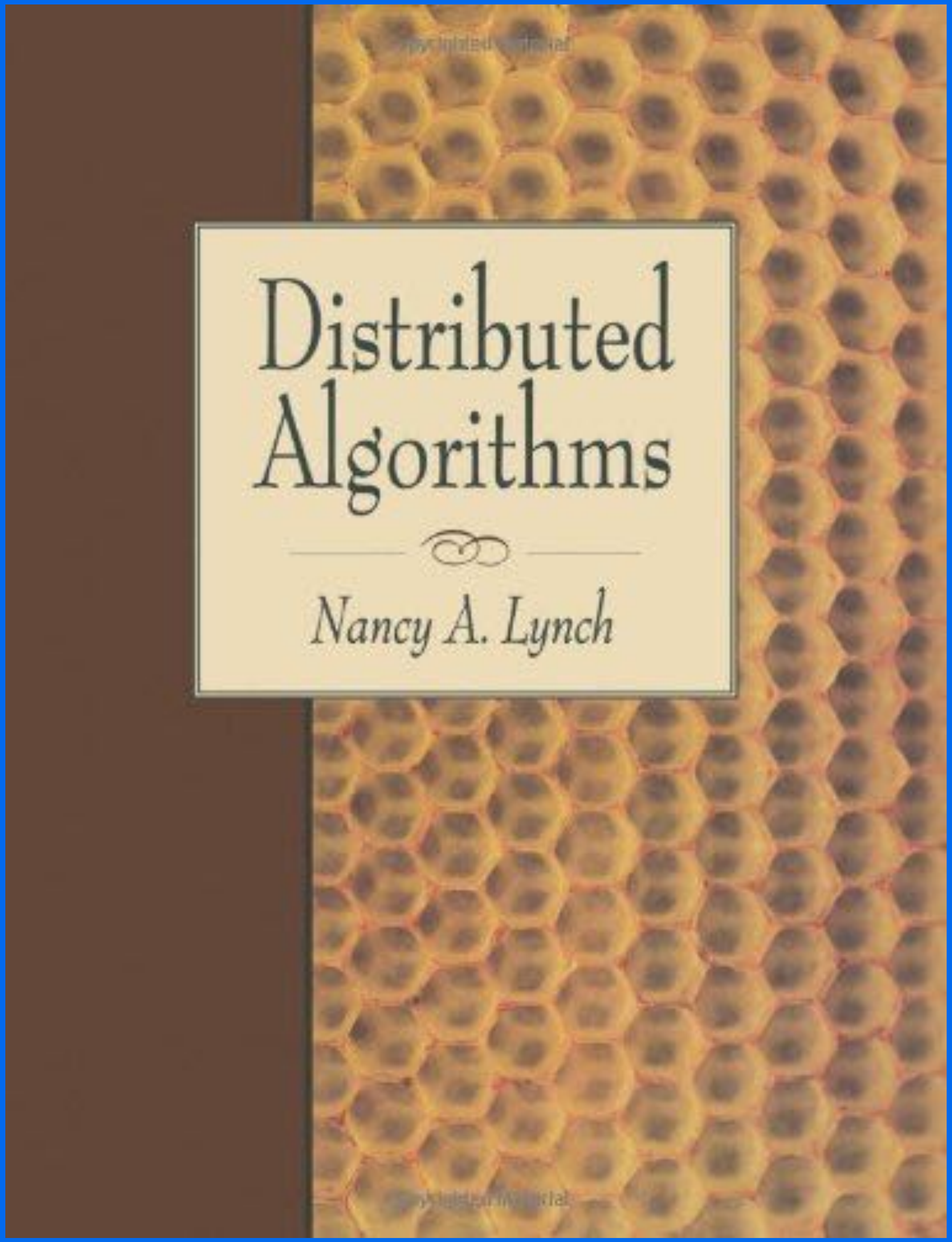
http://www.amazon.com/Distributed-Algorithms-Message-Passing-Systems-Michel/dp/3642381227/



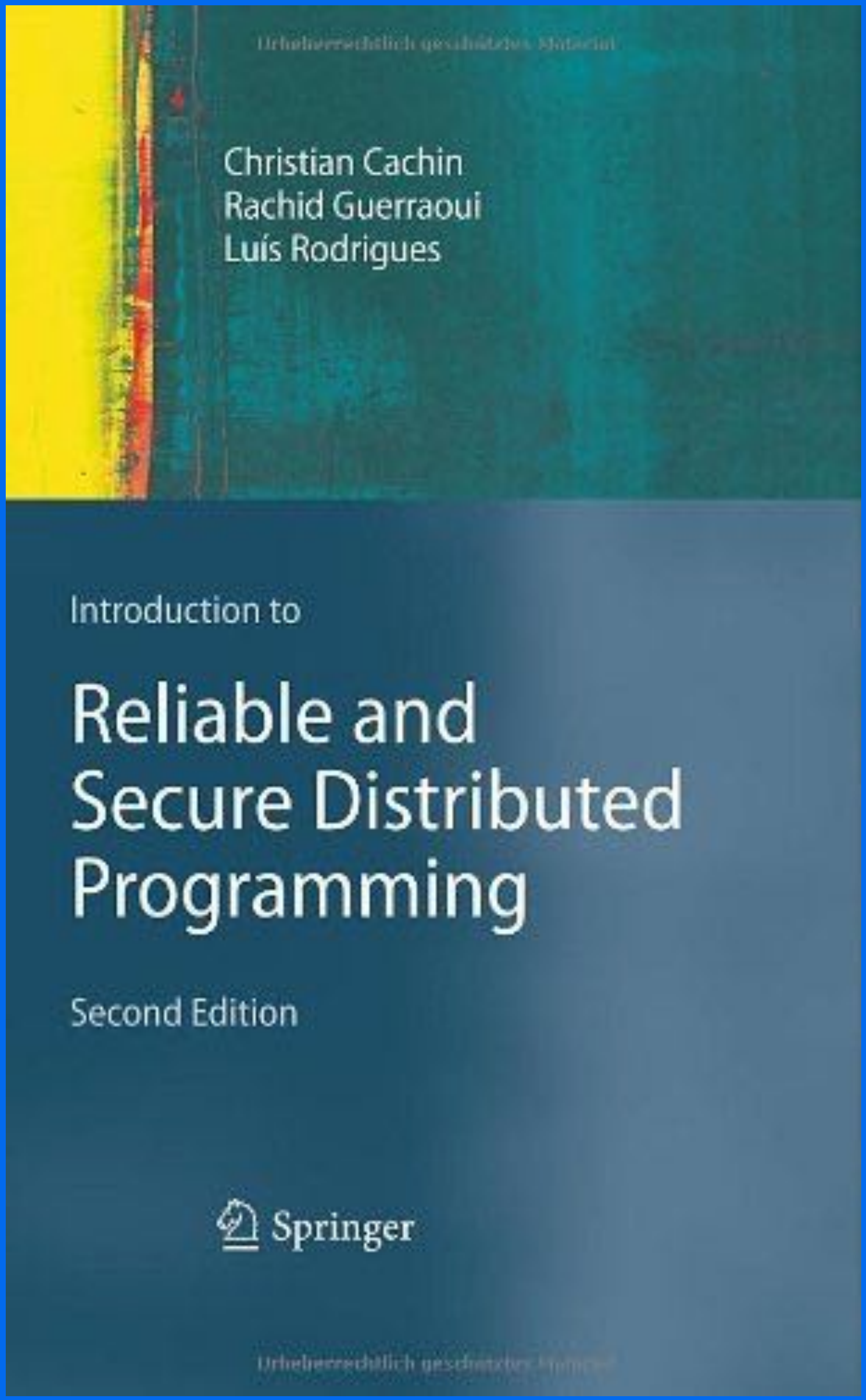
http://www.amazon.com/Fault-tolerant-Agreement-Synchronous-Message-passing-Distributed/dp/1608455254/



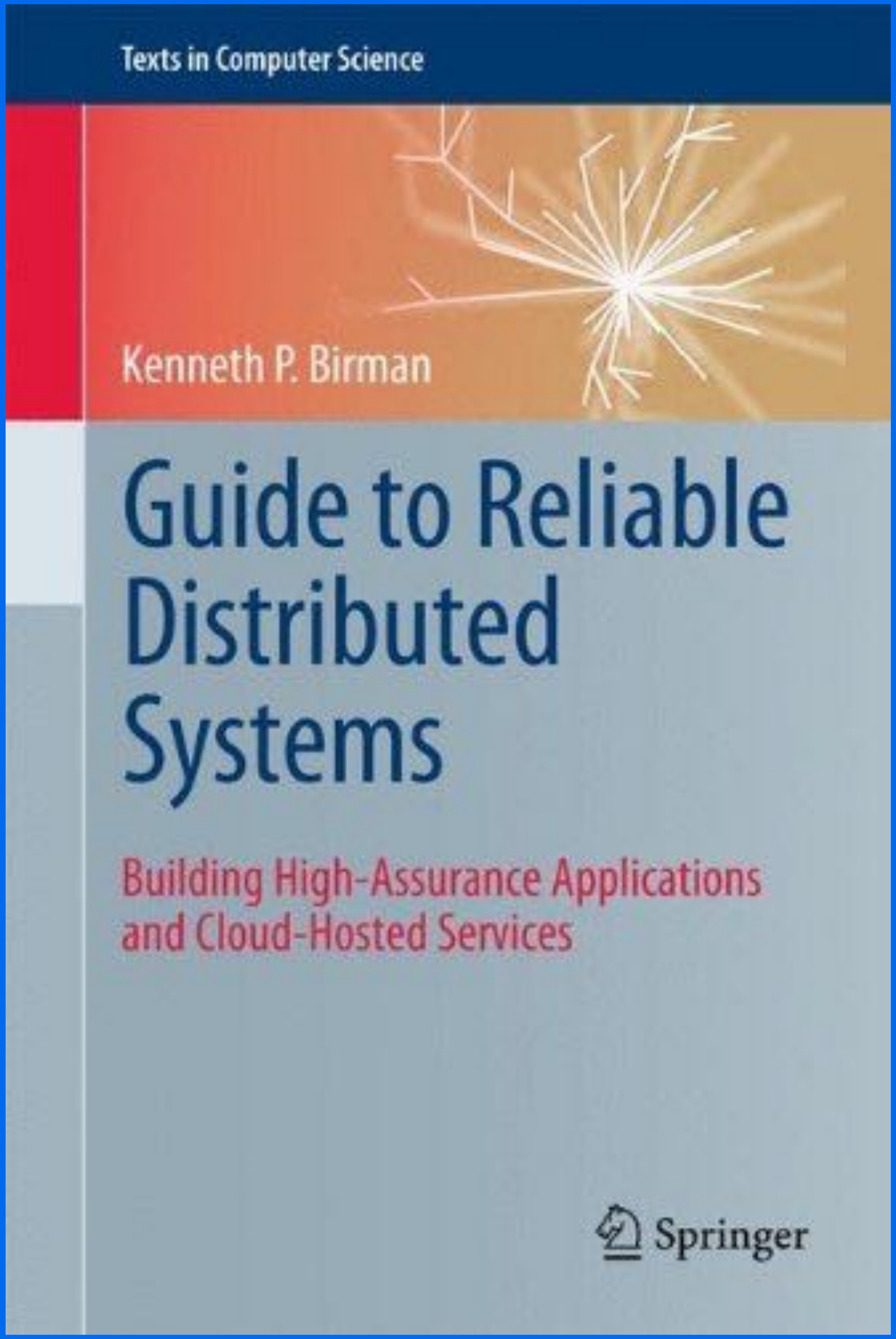
http://www.amazon.com/Communication-Abstractions-Fault-tolerant-Asynchronous-Distributed/dp/160845293X/



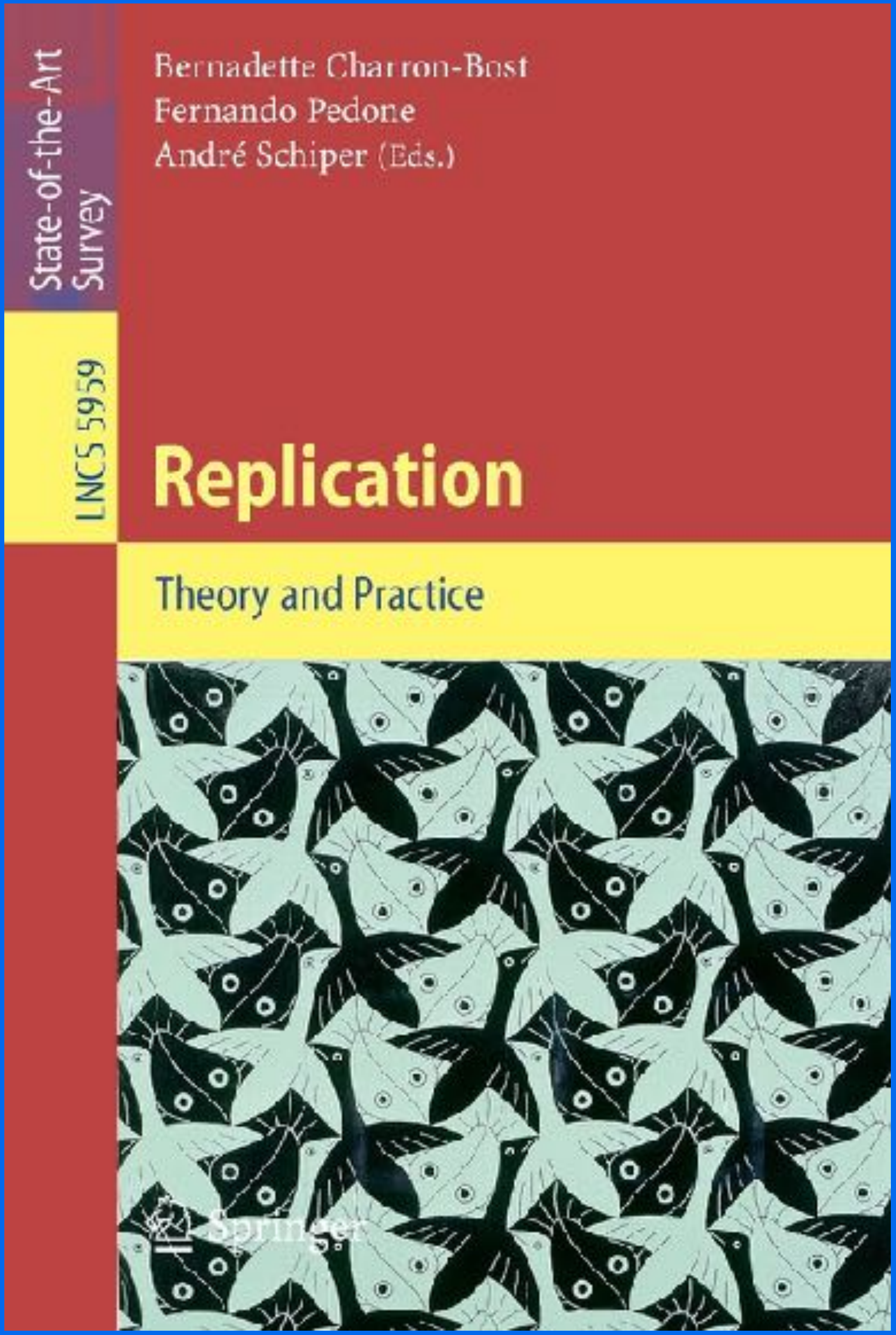
http://www.amazon.com/Distributed-Algorithms-Kaufmann-Management-Systems/dp/1558603484/



http://www.amazon.com/Introduction-Reliable-Secure-Distributed-Programming/dp/3642152597



http://www.amazon.com/Guide-Reliable-Distributed-Systems-High-Assurance/dp/1447124154/



http://www.amazon.com/Replication-Practice-Lecture-Computer-Theoretical/dp/3642112935/

FINDING NON PAYWALLED PAPERS

CONCLUSION

CONCLUSION

- Deep Rabbit Hole

CONCLUSION

- Deep Rabbit Hole
- Computing Science where Science is **Still a Thing™**

CONCLUSION

- Deep Rabbit Hole
- Computing Science where Science is **Still a Thing™**
- History of the Field Matters

CONCLUSION

- Deep Rabbit Hole
- Computing Science where Science is **Still a Thing™**
- History of the Field Matters
- Read, read, read

THANKS!

@old_sound