A Bororo told me...

(Thoughts on Informatics and distributed computing)

Michel RAYNAL

Academia Europaea

Institut Universitaire de France

IRISA, Université de Rennes, France

Polytechnic University (PolyU), Hong Kong

"In keeping with the spirit with which it has been imbued since it was founded, the Collège de France is responsible to give its attendees, not granted truths, but the idea of a free research."

"Ce que le Collège de France, depuis sa fondation, est chargé de donner à ses auditeurs, ce ne sont pas des vérités acquises, c'est l'idée d'une recherche libre."

Maurice Merleau-Ponty (1908-1961)

"Enseigner c'est réfléchir à voix haute devant les étudiants."

"Teaching is thinking aloud in front of students."

Henri Lebesgue (1875-1941)

What is our job?

Une réflexion
un tantinet provocatrice

A thought, somehow provocative

1955, Collection "Terre humaine" (Jean Malaurie)

Incipit : "Je hais les voyages et les explorateurs."
(Que voulait-il dire ?)

"I hate travels and explorers" (What did he want to say?)

A translation

A Bororo told me

“Je hais les ordinateurs et leurs applications.”

“I hate computers and their applications.”

In other words

- Informatics is eaten by its applications
  It is anti-Chronos (see research credits)
- 1936 : Alan Mathison Turing (1912-1954)
  - ⋆ Foundations of sequential computing
  - ⋆ We know what is computable, and what is not
  - ⋆ Theory preceded applications
- Change of paradigm
  - ⋆ Today: the scheme has reversed
  - ⋆ A lot of applications precede theory ..., but very few fertilize it
  - ⋆ Unfortunately a lot applications are to Informatics what sandwich loaf is to bread .... ("sliced and packaged")
  - ⋆ Informatics cannot be reduced to an application megalopolis

What is our job?

- Understand difficult thinks (in our domain)
- Make things as simple as possible
- Research is the "raison d'être" of universities
- Universe, universality, university, ..., same root!
- Year after year ...

this creates a revolution in ... industry!

"Teaching is not an accumulation of facts."

Lamport, Teaching concurrency, ACM SIGACT News, 40(1):58-62 (2009)

Distributed computing for dummies

"A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable" (L. Lamport)

The essence eof distributed computing

"In sequential systems, computability is understood through the Church-Turing Thesis: anything that can be computed, can be computed by a Turing Machine.

In distributed systems, where computations require coordination among multiple participants, computability questions have a different flavor. Here, too, there are many problems which are not computable, but these limits to computability reflect the difficulty of making decisions in the face of ambiguity, and have little to do with the inherent computational power of individual participants."

- Herlihy M., Rajsbaum S., and Raynal M., Power and limits of distributed computing shared memory models. *Theoretical Computer Science*, 509:3-24 (2013)

Intrinsic difficultly : master the uncertainty created by the non-determinism inherent to distributed executions

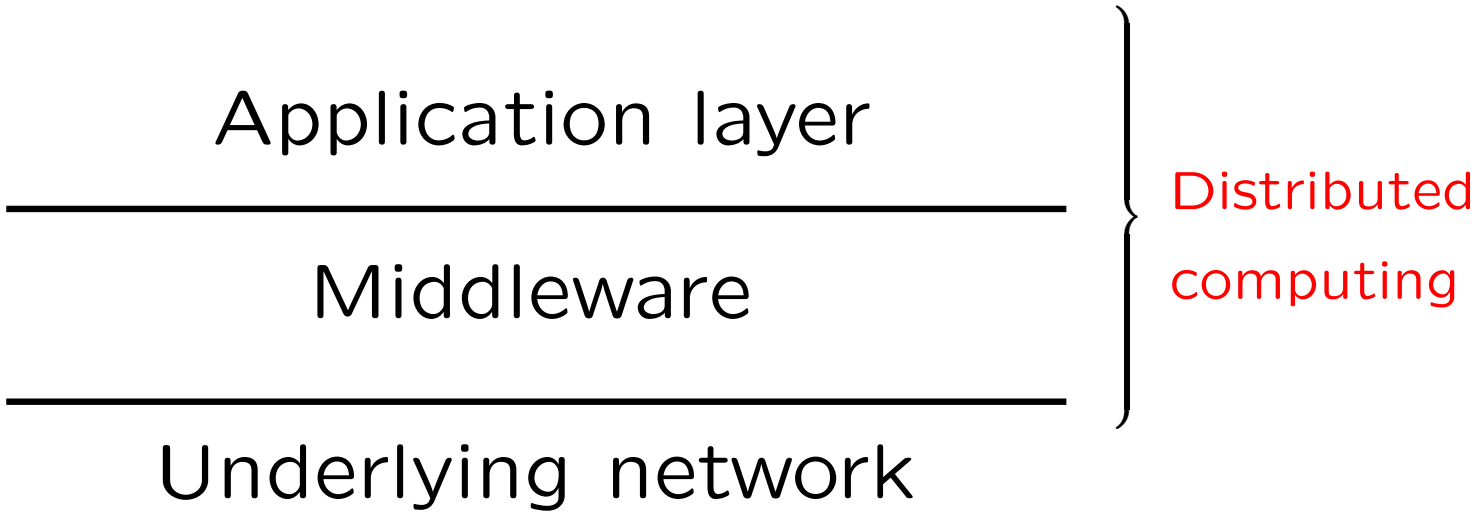
A few adversaries (environment)

- Multiplicity of geo-distributed control flows
- Synchrony/asynchrony
- Failures
- Mobility
- Anonymity
- Dynamicity
- Energy consumption (e.g., sensors)
- Et cetera

The (non-deterministic) environment in which runs a distributed algorithm constitutes one of its inputs

To summarize

Distributed computing is the added value on top of an imposed communication infrastructure



On the distributed computing side
A quick look at the past

A few important dates

Lors de sa Leçon inaugurale le 5 Janvier 1960 (Chaire d'Anthropologie sociale, 1959-1982) Claude Lévi-Strauss n'eut de cesse de rappeler le nom de quelques grands prédécesseurs dont disait-il

"nous nous offrons le luxe d'oublier leur contribution"
"we have the luxury to forget their contribution"

A few Distributed Computing related dates

The future?

"Prediction is very difficult, especially when it is about the future." (Niels Bohr, 1885-1962)

- No prediction, only an "LL(k)" view (with k small!)
- A gentle advice (?) to PhD students to select a good PhD topic:

  Consider a "good" topic in a "good" research team, and then work hard

  - ⋆ "good" $\neq$ trendy, catchword, popular, buzzword, ...
  - ⋆ You must choose so that you will not go the garbage when your ad hoc PdD topic will go to the garbage

The world is distributed

$\Rightarrow$

DC is fundamental!

A few DC problems

A few important topics (1)

Distributed recursivity

- Recursion parameter is
  - ⋆ not related to the input values
  - ⋆ related to nb of processes seen as participating
  - ⋆ Hence, it is failure-related

    (a hidden "input" of the execution!)

- Lamport L., Shostak E., and Pease M.C., The Byzantine general problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382-401 (1982)

- Gafni E. and Rajsbaum S., Recursion in distributed computing. *Proc. 12th Intl Symp. on Stabilization, Safety, and Security of Dist.Syst.*, Springer LNCS 6366, pp. 362-376 (2010)

- Raynal M., Concurrent programming: algorithms, principles and foundations. *Springer*, 515 pages (2012)

A few important topics (2)

Which relations between SM and MP?

- The d-solo model:
  - ⋆ 1-solo model = wait-free read/write model
  - ⋆ n-solo model = wait-free message-passing model

    - M. Herlihy, S. Rajsbaum, M. Raynal, and and J. Stainer, From wait-free to arbitrary concurrent solo executions in colorless distributed computing. *Theoretical Computer Science*, 683:1-21 (2017)

  - ⋆ On the computability side:

    $$\text{1-solo} \succ \cdots \; d\text{-solo} \succ (d+1)d\text{-solo} \succ \cdots \succ n\text{-solo}$$

  - ⋆ non-trivial task: (d, ϵ)-approximate agreement
- Objects weaker than read/write registers?

A few important topics (3)

Trading t-resilience for efficiency

- Example: Byzantine world (Byz reliable broadcast)
- t = max nb of Byzantine processes
- Main issue: ensure that if a correct process delivers a message m from another process p_j (be p_j correct or Byzantine), all correct processes delivers the very same message m

	fault resilience	communication steps message types	number of messages
Bracha's algorithm	$n > 3t$	3	$2n^2 - n - 1$
Imbs-Raynal's algorithm	$n > 5t$	2	$n^2 - 1$

- G. Bracha, Asynchronous Byzantine agreement protocols. *Information & Computation*, 75(2):130-143 (1987)

- D. Imbs and M. Raynal, Trading t-resilience for efficiency in asynchronous Byzantine reliable broadcast. *Parallel Processing Letters*, Vol. 26(4), 8 pages (2016)

A few important topics (4)

- Informatics is a science of abstractions

  Informatics is a science of abstractions, and a main difficulty consists in providing users with a "desired level of abstraction and generality: one that is broad enough to encompass interesting new situations, yet specific enough to address the crucial issues", M. Fischer and M. Merritt, Appraising two decades of distributed computing theory research. *Distributed Computing*, 16(2-3):239-247 (2003)

- Classical examples

  ⋆ Sequential programming languages

  ⋆ Synchronization side: STM

  ⋆ Distributed computing

    ∗ Communication abstractions
    ∗ Agreement objects
    ∗ Symmetry-breaking objects
    ∗ etc.

  in the presence of adversaries such as asynchrony, failures, mobility, etc.

On communication abstractions in DC

send/receive, basic broadcast: network machine language

Relation agreement objects $\leftrightarrow$ comm. abstractions

Concurrent object	Communication abstraction

Consensus	Total order broadcast
Causal memory	Causal order broadcast
k-set agreement	k-BO-broadcast (DISC'17)
Snapshot object	SCD-broadcast (ICDCN'18)

One coin: two faces!

A few important topics (5)

Many other fascinating topics: bulk examples

- Which is the weakest FD for k-set agreement in MP?
- Relation between concurrency and failures
- Anonymity: process anonymity vs memory anonymity
- Unify Process adversary and message adversaries

A few important topics (6)

Objects with no sequential specification

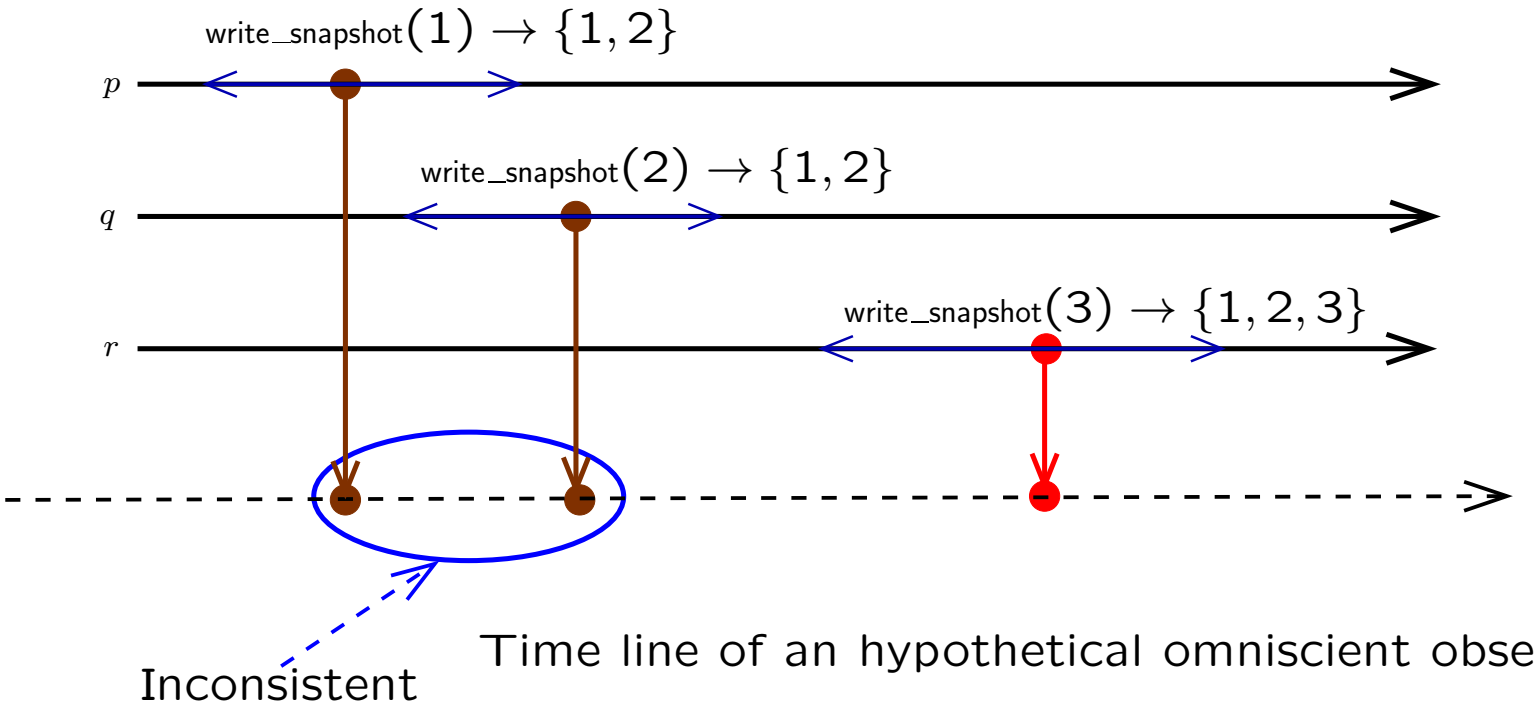
(Beyond distributed "classical" state machines)

- State machine = sequential behavior
- Cooperating distributed state machines
- Beyond distributed state machines
- Distributed objects defined by
  - ⋆ sequential spec.: many consistency conditions
  - ⋆ non-sequential specification: very few results

- A.Castañeda, S. Rajsbaum, and M. Raynal, Unifying concurrent objects and distributed tasks: Interval-linearizability. *Journal of the ACM*, 65(6), Article 45, 42 pages (2018)

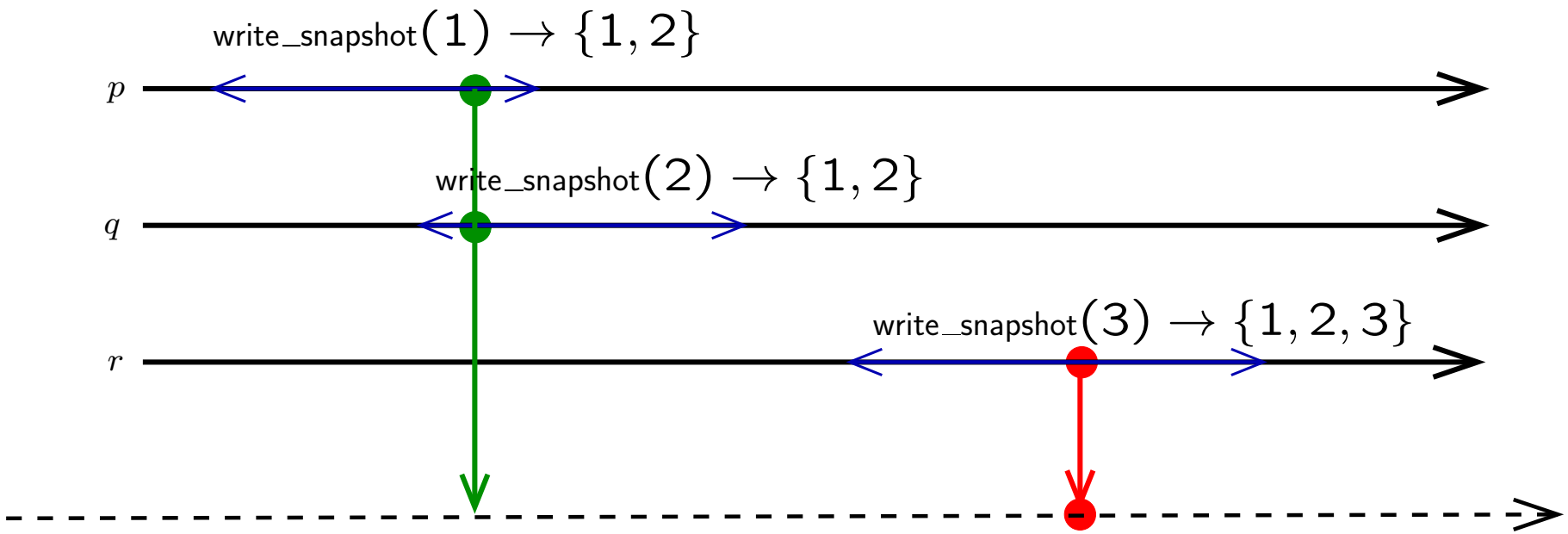
Not linearizable

A simple example



Not allowed to read in the future

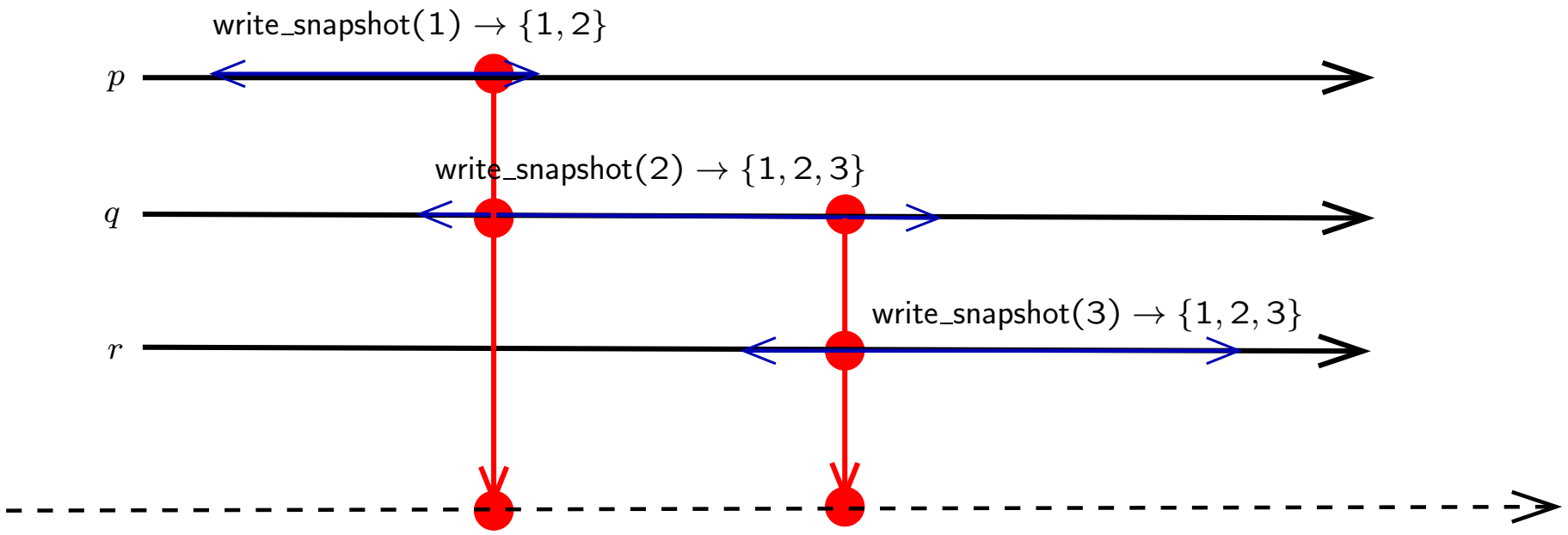
... but set-linearizable



Time line of an hypothetical omniscient observer

This behavior is not allowed by a sequential specification

Neither linearizable, nor set-linearizable



Time line of an hypothetical omniscient observer

This behavior is not allowed by a set-sequential specification

Parallel computing versus Distributed computing

A big confusion

Criterion: Which is the main issue to solve?

- Parallel computing: design choice
  - ⋆ Aim: Efficiency
  - ⋆ Mean: leverage data independence
- Distributed computing:

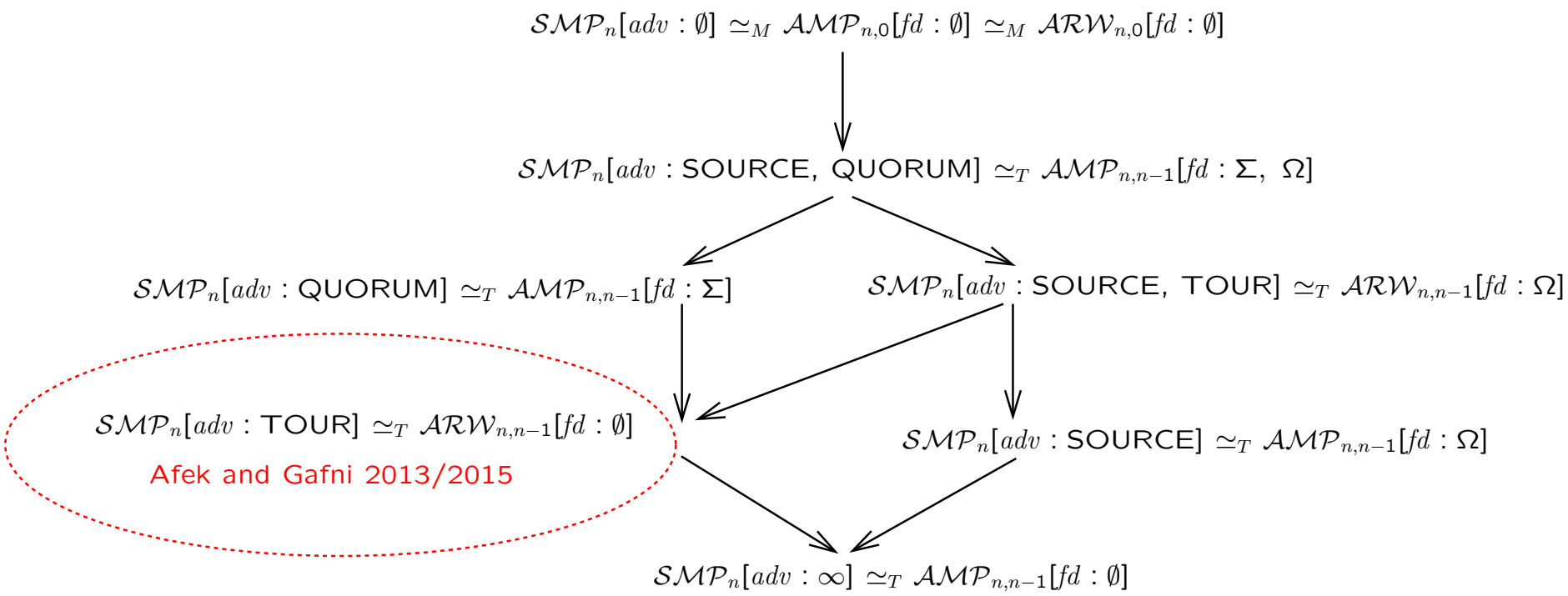
  inherent constraints imposed by the environment
  - ⋆ Aim: master uncertainty created by adversaries
  - ⋆ Issue: the behavior of the envirt is an input!

We have to better understand... (some duality?)

M. Raynal, A pleasant stroll through the land of distributed machines, computation, and universality. *Proc. 8th In'l Conference on Machines, Computations, and Universality (MCU'18)*, Springer LNCS 10881, pp. 34-50 (2018)

A map of distributed computing models

Message adversaries/failure detectors equivalences

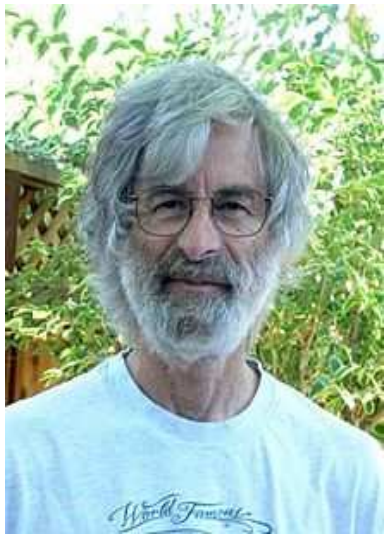


TOUR: For each pair of processes p_i and p_j, and in each synchronous round, is allowed to suppress either the message sent by p_i to p_j or the message sent by p_j to p_i, but not both

Synchronous system weakened by TOUR and the asynchronous crash-prone RW system have the same computability power for task solvability

From E.W. Dijkstra to L. Lamport (and all of us!)

We still have
to better
understand
the long path



from mutual exclusion to consensus
the keystones of distributed computing

From physical resources (mutex)

to data virtualization (digitalized distributed objects)

S. Rajsbaum and M. Raynal,
Mastering concurrent computing through sequential thinking: a half-century evolution, To appear in *Communications of the ACM* (2019)

Additional material

SCD-broadcast: definition

The SCD-Broadcast abstraction: definition (1)

SCD = Set-Constrained Delivery

- Two operations:
  - ⋆ scd_broadcast(m): broadcasts a message m
  - ⋆ scd_deliver(): returns a non-$\emptyset$ set of messages
- Five properties:
  - ⋆ Validity:
    If a process scd-delivers a set containing a message m, then m was scd-broadcast by some process
  - ⋆ Integrity:
    A msg is scd-delivered at most once by each process

The SCD-Broadcast abstraction: definition (2)

- MS-Ordering:
  A process p_i scd-delivers a message set ms_i containing a message m and later a message set ms'_i containing a message m'
  $\Rightarrow$
  no process scd-delivers first a message set ms'_j containing m' and later a message ms_j containing m

- Termination-1:
  If a non-faulty process scd-broadcasts a message m, it terminates its scd-broadcast invocation and scd-delivers a message set containing m

- Termination-2:
  If a process scd-delivers a message m, every non-faulty process scd-delivers a msg set containing m

A simple example

- Messages SCD-broadcast by processes:
  m_1, m_2, m_3, m_4, m_5, m_6, m_7 and m_8
- SCD-deliveries:
  - ⋆ at p_1: $\{m_1, m_2\}$, $\{m_3, m_4, m_5\}$, $\{m_6\}$, $\{m_7, m_8\}$
  - ⋆ at p_2: $\{m_1\}$, $\{m_3, m_2\}$, $\{m_6, m_4, m_5\}$, $\{m_7\}$, $\{m_8\}$
  - ⋆ at p_3: $\{m_3, m_1, m_2\}$, $\{m_6, m_4, m_5\}$, $\{m_7\}$, $\{m_8\}$

A simple example: incorrect SCD-deliveries

- Messages SCD-broadcast by processes: m_1, m_2, m_3, m_4, m_5, m_6, m_7 and m_8
- SCD-deliveries:
  - $\star$ at p_1: $\{m_1, m_2\}$, $\{m_3, m_4, m_5\}$, ...
  - $\star$ at p_2: $\{m_1, m_3\}$, $\{m_2\}$, ...

Main features

- Necessary and sufficient assumption: $t < \frac{n}{2}$
- Distributed software engineering:

  All "technical details" are hidden in this algorithm which is implemented and proved once for all!
- On tne efficiency side
  - ⋆ Assumption:
    - ∗ Let Δ = message max latency
    - ∗ Local computation: zero cost
  - ⋆ Cost:
    - ∗ Time: 2Δ ($2 \times$ network latency)
    - ∗ Messages: n^2

SCD-broadcast in action

Implement an atomic snapshot object

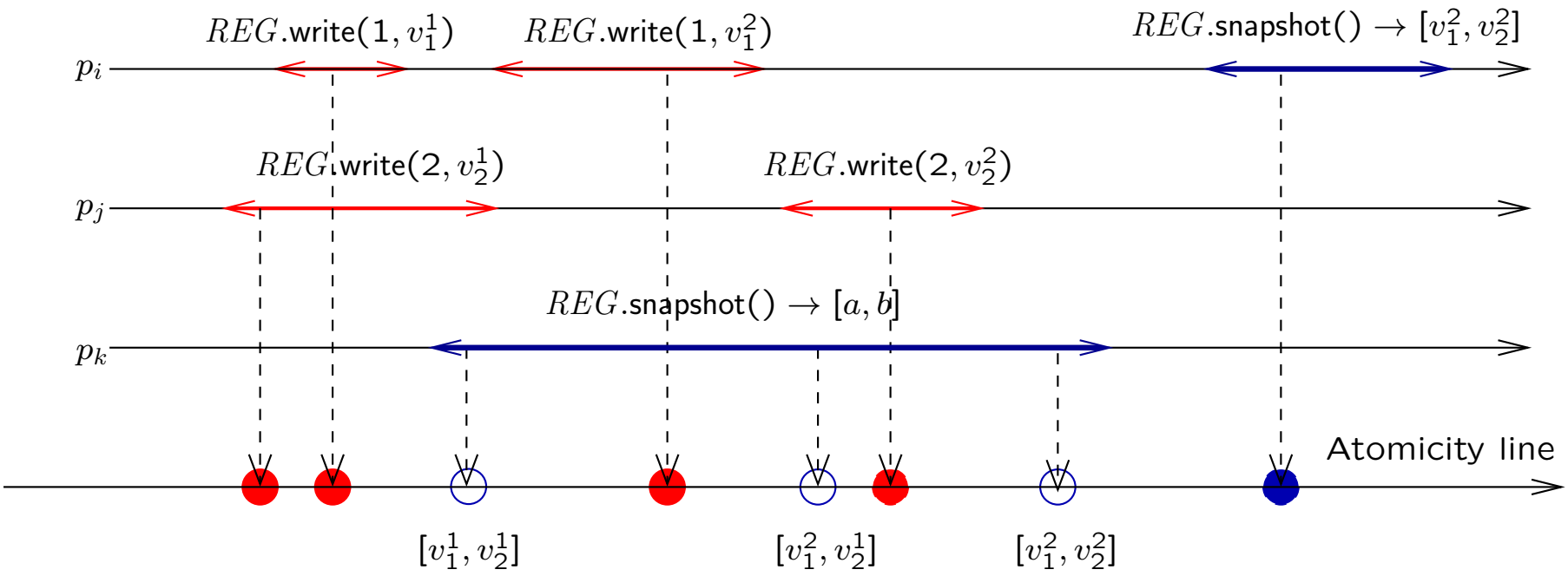
The atomic snapshot object was introduced in

- Afek Y., Attiya H., Dolev D., Gafni E., Merritt M., and Shavit N., Atomic snapshots of shared memory. *Journal of the ACM*, 40(4):873-890 (1993)

Snapshot object

- array $REG[1..m]$ of atomic read/write registers with two operations, write() and snapshot()
- MWMR snapshot
- write(r, v) assigns v to $REG[r]$
- snapshot() returns the value of the full array as if the operation had been executed instantaneously (atomicity consistency)

Example of an MWMR atomic snapshot object



$[v_1^1, v_2^2]$ cannot be output

Snapshot from SCD-broadcast

Local representation of the snapshot object REG

- $reg_i[1..m]$: current value of $REG[1..m]$, as known by p_i
- $done_i$: Boolean variable
- $tsa_i[1..m]$: array of timestamps associated with the values stored in $reg_i[1..m]$

  $\star$ $tsa_i[j].date$ and $tsa_i[j].proc$ (timestamp of $reg_i[j]$)

- Lexicographical total order $<_{ts}$:

  $\star$ $ts1 = \langle h1, i1 \rangle$ and $ts2 = \langle h2, i2 \rangle$

  $\star$ $ts1 <_{ts} ts2 \stackrel{def}{=} (h1 < h2) \vee ((h1 = h2) \wedge (i1 < i2))$

Algorithm: snapshot operation

operation snapshot() **by** p_i **is**
$done_i \leftarrow$ `false`;
scd_broadcast SYNC (i);
wait($done_i$); % end of synchronization
return($reg_i[1..m]$).

- SYNC (i) synchronization message
- allows p_i to obtain an atomic value of $REG[1..m]$

Algorithm: write operation

operation write(r, v) **by** p_i **is**

$done_i \leftarrow$ `false`;
scd_broadcast SYNC (i);
wait$(done_i)$; % end of synchronization 1
$done_i \leftarrow$ `false`;
scd_broadcast WRITE $(r, v, \langle tsa_i[r].date + 1, i\rangle)$;
wait$(done_i)$. % end of synchronization 2

Algorithm: message reception

when the message set
$\{\text{WRITE}(r_{j_1}, v_{j_1}, \langle date_{j_1}, j_1\rangle),\ \cdots,\ \text{WRITE}(r_{j_x}, v_{j_x}, \langle date_{j_x}, j_x\rangle),$
$\text{SYNC}(j_{x+1}),\ \cdots,\ \text{SYNC}(j_y)\ \}$ **is scd-delivered do**

for each r such that $\text{WRITE}(r, -, -) \in$ the message set **do**
let $\langle date, writer\rangle$ = greatest timestamp in $\text{WRITE}(r, -, -)$;
if $(tsa_i[r] <_{ts} \langle date, writer\rangle)$
then let v the value in $\text{WRITE}(r, -, \langle date, writer\rangle)$;
$reg_i[r] \leftarrow v$; $tsa_i[r] \leftarrow \langle date, writer\rangle$
end if
end for;
if $\exists\ \ell : j_\ell = i$ **then** $done_i \leftarrow$ `true` **end if**.

Observation: no quorum at this abstraction level!

Comparing cost

- Stacking apatach vs scd-broadcat
- $O(n^2)$ messages in both cases
- time:

	RW-stacking	SCD-broadcast
snapshot	$n^2\Delta$	2Δ
write	$n\Delta$	4Δ

From atomicity to sequential consistency

Suppress the messages SYNC!

These messages ensure compliance wrt real-time

operation snapshot() **by** p_i **is**
return($reg_i[1..m]$).

operation write(r, v) **by** p_i **is**
$done_i \leftarrow$ `false`;
scd_broadcast WRITE $(r, v, \langle tsa_i[r].date + 1, i\rangle)$;
wait($done_i$).

when the message set
$\{\text{WRITE}(r_{j_1}, v_{j_1}, \langle date_{j_1}, j_1\rangle),\ \ldots, \text{WRITE}(r_{j_x}, v_{j_x}\}$
is scd-delivered do ..... same as before

Conclusion

Ultimate: the quest of the holly Grail?

- Too many distributed computing models
- Is there a Great Unified Model??
- Our job is also looking for Simplicity
  - ⋆ "Everything should be made as simple as possible, but not simpler!" Albert Einstein (1879-1955)
  - ⋆ "Je vous écris une longue lettre parce que je n'ai pas eu le temps d'en écrire une plus courte." Blaise Pascal (1623-1662)
  - ⋆ "Simplicity does not precede complexity, but follows it." "Alan Perlis (1922-190), First Turing Award
- Simplicity is difficult, ...

but it must be a central part of our job!☺

- Alan Key's answer to Dan Pendery's question on the future "tendencies" to adapt the products of Xeros

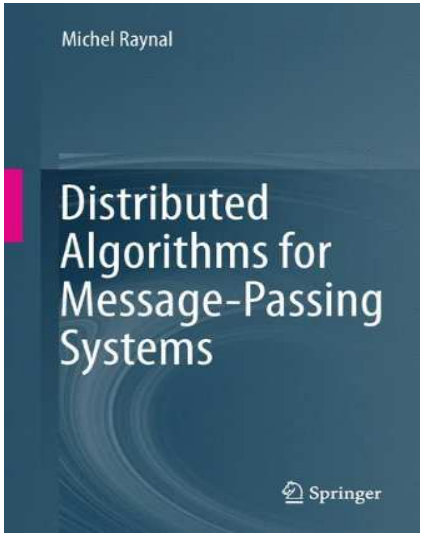
  The best way to predict the future is to invent it !

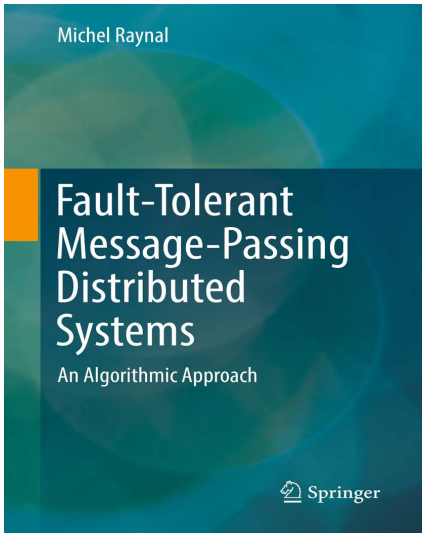
- In our quest of the lost universality ☺[1]

  The paradise is never lost because it consists of its own (and infinitely recursive) quest!

[1] Remind Rachid's Inaugural Lecture
"À la recherche de l'universalité perdue." (In search of the lost universality)

Personal contribution





- Concurrent Programming: Algorithms, Principles and Foundations. *Springer*, 515 pages (2012)
- Distributed Algorithms for Message-passing Systems. *Springer*, 510 pages (2013)
- Fault-Tolerant Message-passing Distributed Systems: An Algorithmic Approach. *Springer*, 492 pages (2018)