# Anonymous Read/Write Systems
# a Short Introduction

Michel RAYNAL

*Institut Universitaire de France

◇IRISA, Université de Rennes, France

†Polytechnic University, Hong Kong



AIM:

Understand Anonymity
and its implication on algorithms



## Summary

- Concurrent objects and their progress conditions
- Two types of anonymity
- Anonymity and any number of crash failures
- Examples: snapshot, consensus, mutex
- Conclusion



## Main bibliography


- Bouzid Z., Raynal M., and Sutra P.,
Anonymous obstruction-free $(n, k)$-set agreement with $(n-k+1)$ atomic read/write registers. *Distributed Computing*, 31:99-117 (2018)
- Guerraoui R. and Ruppert E.,
Anonymous and fault-tolerant shared-memory computations. *Distributed Computing*, 20:165-177 (2007)
- Rabin M.,
The choice coordination problem. *Acta Informatica*, 17(2):121-134 (1982)
- Taubenfeld G.,
Coordination without prior agreement. *Proc. 36th ACM Symposium on Principles of Distributed Computing (PODC'17)*, pp. 325-334 (2017)

# Concurrent objects
# and their progress conditions



## Once upon a time ...

- Sequential programming:
  - ⋆ C.A.R. Hoare: the notion of a Record class (1965)
  - ⋆ O.-J., Dahl, K. Nygaard: SIMULA 67 introduced
    - ∗ The notion of an object (encapsulation, prefix/heritage)
    - ∗ The notion of a co-routine (thread)
- Concurrent programming: E.W. Dijkstra (1965)
  - ⋆ Notion of a semaphore, notion of a process
- O.-J. Dahl, E.W.D. Dijkstra et C.A.R. Hoare, Structured Programming Academic Press, 1972 (ISBN 0-12-200550-3)



## Concurrent object

- Object: operations + specification
- Objects were introduced in prog. languages in 1967!
- concurrent object:
  object that can be accessed by several processes
- Specification
  - ⋆ Sequential (considered here)
  - ⋆ Not sequential (e.g., rendezvous, NBAC)
    the specification involves physical time or the behavior of the environment



## Not-sequential specification: NBAC example

- One-shot operation nbac_propose($v$), where $v \in \{\texttt{yes}, \texttt{no}\}$
- properties:
- NBAC-validity. nbac_propose() returns only `commit` or `abort`
  - ⋆ NBAC-justification. If a process returns `commit`, all processes voted `yes`
  - ⋆ NBAC-obligation. If all processes vote `yes` and no process crashes, abort cannot be decided
- NBAC-agreement. No two processes decide differently
- NBAC-termination. Every correct process decides

## Progress conditions in failure-free systems

- Deadlock-freedom (server-oriented)
- Starvation-freedom (client-oriented)



## Progress conditions in crash prone systems (1)

were introduced in the following articles

- Herlihy M.P., Luchangco V., and Moir M.,
  Obstruction-free synchronization: double-ended queues as an example.
  *23th Int'l IEEE Conf. on Distributed Computing Systems (ICDCS'03)*, IEEE Press, pp. 522-529 (2003)
- Herlihy M.P. and Wing J.M.,
  Linearizability: a correctness condition for concurrent objects.
  *ACM Transactions on Progr. Languages and Systems*,12(3):463-492, (1990)
- Herlihy M.,
  Wait-free synchronization.
  *ACM Transactions on Progr. Languages and Systems*, 13 (1):124-149 (1991)

Which invariant characterizes these three articles?



## Progress conditions in crash prone systems (2)

- Obstruction-freedom
  requires that, if a process $p$ invokes an operation on an object $O$, and all other processes that have pending operations on $O$ pause during a long enough period, then process $p$ terminates its operation
- Non-blocking
  requires that, if several processes have concurrent invocations on an object $O$, and one of them does not crash, then one of these invocations terminates
- Wait-freedom
  requires that, if a process invokes an operation on an object $O$ and does not crash, it terminates its operation (i.e., no other process can prevent it from terminating)



## Progress conditions in crash prone systems (3)

- The definition of
  - ⋆ obstruction-freedom and non-blocking
    depends on the concurrency pattern
  - ⋆ wait-freedom
    does not depend on the concurrency pattern
- Obstruction-freedom:
  meaningfull in a failure-free system!
- In all cases, the internal representation of the object (which is built) can be concurrently accessed/modified by several processes (mutex-freedom)

# What is anonymity?

Process anonymity vs Memory anomymity



## Computing model

- $n$ sequential asynchronous processes: $p_1$, .., $p_n$
- Cooperation: atomic read/write registers only



## Process anonymity

- Motivations: privacy, tiny devices (sensors), etc.
- Processes: no name, cannot be distinguished
- $i$: index of $p_i$, known only by an external observer
- Main question: what can be deterministically implemented in the process-anonymous crash-prone model?

Elements for an answer in:
Guerraoui R. and Ruppert E., Anonymous and fault-tolerant shared-memory computations. *Distributed Computing*, 20:165-177 (2007)



## Memory anonymity: definition

- Definition:
  There is no a priori agreement between processes on the names of shared memory locations
- Implicitly used in the early eighties:
  Rabin M., The choice coordination problem. *Acta Informatica*, 17(2):121-134 (1982)
- Defined and considered as a concept in:
  Taubenfeld G., Coordination without prior agreement. *Proc. 36th ACM Symposium on Principles of Distributed Computing (PODC'17)*, pp. 325-334 (2017)

(Epistemology: from mechanisms to concepts)

## Memory anonymity: example

Shared memory: 3 atomic read/write registers $SM[1..3]$

| names for the global observer | names for process $p_i$ | names for process $p_j$ |
|---|---|---|
| $SM[1]$ | $SM[2]$ | $SM[3]$ |
| $SM[2]$ | $SM[3]$ | $SM[1]$ |
| $SM[3]$ | $SM[1]$ | $SM[2]$ |

The permutations are defined by the adversary
No process knows these permutations!



## Spirit of the talk/article

- Algorithmic thinking in the presence of anonymity
  - ⋆ Process anonymity
  - ⋆ Memory anonymity
  - ⋆ Not both!
- Like "selected pieces" in literature, musics, etc.



## Selected piece 1

## Process-Anonymity and Any Number of Process Crashes

## Snapshot object



## Snapshot object (1)

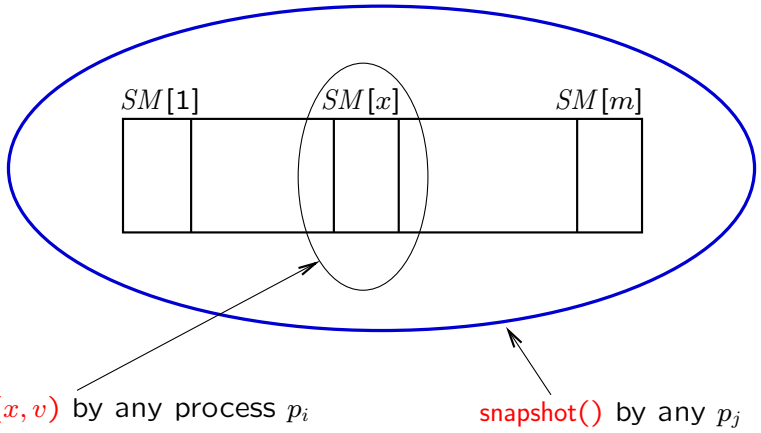
- Introduced independently in

  - 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)

  - Anderson J., Multi-writer composite registers. *Distributed Computing*, 7(4):175-195 (1994)

- Made up of $m$ components ( atomic read/write reg.)
- Higher abstraction level than read/write operations
- Two operations
  - ⋆ write$(x, v)$ where $1 \leq x \leq m$ ($x$ component nb)
  - ⋆ snapshot() is on all the components

## Snapshot object (2)






## Snapshot object (3)

- write$(x, v)$ and snapshot() are atomic (linearizable)
- This means:
  - ⋆ appear as if they have been executed sequentially
  - ⋆ this sequence
    - ∗ is such that, if an operation op1() terminated before an operation op2() started, op1() appears before op2() in this sequence
    - ∗ belongs to sequential specification of the snapshot object



## Internal representation of a snapshot object

A snapshot object is represented by

- In the shared memory
  - ⋆ an array $SM[1..m]$ of MWMR atomic read/write reg.
  - ⋆ each $SM[x]$ is a pair $SM[x] = \langle SM[x].ts, SM[x].value \rangle$ initialized to the pair $\langle -, \perp \rangle$
- At every process $p_i$: a counter $ts_i$ initialized to $-1$



## Principles and operation write()

Basic strategy:
Associate a sequence number with each written value

**operation** write$(x, v)$ **is**
$ts_i \leftarrow ts_i + 1$;
$SM[x] \leftarrow \langle ts_i, v \rangle$;
return().

**operation** snapshot() **is**
classical double asynchronous collect strategy
see next slide.

meaning of "**for each** $x \in \{1, \ldots, m\}$ **do** $\cdots$ **end for**":
there is no constraint on the order in which $x$ takes its successive values

## How and why it works (1)

From where the difficulty is coming?

- As processes do not have names, it is not possible to identify a written value with pair (proc. id, seq. number)
- Nevertheless counters can be used to tag values
- But ...
  the same pair $\langle ts, v\rangle$ can be produced by several proc.



## How and why it works (2)

When a process $p_i$ executes snapshot()

- a given pair $\langle ts, v\rangle$ can be written at most $(n-1)$ times (once by each other process)
- and this can occur for each of the $m$ atomic registers
- from which follows that if $p_i$ sees $m(n-1)+2$ collects returning the same array, at least two consecutive of them are not separated by an operation write()



## Operation snapshot()

Double asynchronous collect

**operation** snapshot() **is**
$count_i \leftarrow 1$;
**for each** $x \in \{1,\ldots,m\}$ **do** $sm1_i[x] \leftarrow SM[x]$ **end for**;
**repeat forever**
  **for each** $y \in \{1,\ldots,m\}$ **do** $sm2_i[y] \leftarrow SM[y]$ **end for**;
  **if** $(\forall\ x \in \{1,\cdots,m\} :\ sm1_i[x] = sm2_i[x])$
    **then** $count_i \leftarrow count_i + 1$;
      **if** $\boxed{count_i = m(n-1)+2}$
        **then** return$(sm1_i[1..m].value)$ **end if**
    **else** $count_i \leftarrow 1$
  **end if**;
  $sm1_i[1..m] \leftarrow sm2_i[1..m]$
**end repeat**.



## On the termination side

- write$(x, v)$: always terminates
- snapshot():
  - ⋆ is trivially obstruction-free
  - ⋆ Satisfies also non-blocking
- It is possible to implement a snapshot object satisfying the wait-freedom progress condition despite process-anonymity and any number of process crashes
- It follows that, when considering the implementation of a snapshot object, the "process anonymity" adversary does not create a *computability threshold* in the progress condition hierarchy

Selected piece 2

Process-Anonymity
Obstruction-freedom
and any Number of Process Crashes

Binary consensus object



## Consensus object

- One of the most fundamental objects of fault-tolerant distributed computing
- One-shot concurrent object that has a single operation, denoted propose()
- Defined by the following safety properties:
  - ⋆ Validity.
    If a value is decided by a process, it was proposed
  - ⋆ Agreement.
    No two processes decide different values.
- Binary vs multivalued consensus



## A fundamental *impossibility* result

- It is impossible to design a deterministic algorithm that implements a wait-free consensus object in asynchronous systems where even only one process may crash, processes have identities, and communicate by read/write registers or message-passing.
- Intuitively: asynchrony ⇒ no process can know if anotherprocess is slow ot crashed
- - Fischer M.J., Lynch N.A., and Paterson M.S., Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374-382 (1985)

  - Loui M.C., and Abu-Amara H.H., Memory requirements for agreement among unreliable asynchronous processes. *Parallel and Distributed Computing: Vol. 4 of Advances in Computing Research,* JAI Press, 4:163-183 (1987)



## What is possible

- Consensus experiences a *computability threshold* separating the obstruction-freedom and wait-freedom progress conditions in read/write asynchronous systems.
- In the following is presented a simple obstruction-free process-anonymous binary consensus algorithm.

## Obstruction-free process anonymous binary consensus

- This algorithm is from
  - Guerraoui R. and Ruppert E., Anonymous and fault-tolerant shared-memory computations. *Distributed Computing*, 20:165-177 (2007)
- It is a deramdomized version of a randomized algorithm
  - Chandra T.D., Polylog randomized wait-free consensus. *Proc. 15nd ACM Symposium on Principles of Distributed Computing(PODC'96)*, ACM Press, pp. 166-175 (1996)
- The algorithm rules a competition between two teams
  - ⋆ the team of the processes that champion 0, and
  - ⋆ the team of the processes that champion 1,

  In the competition a process can change its mind (going from a team to the other team) according to its current view of the computation



## Internal representation of a consensus object

A consensus object is represented by

- In the shared memory:
  - ⋆ a two-dimensional array $SM[0..1, 1..]$ of MWMR atomic read/write registers
  - ⋆ each $SM[x, y]$ is a flag initialized to the value `down`, and may later take the value `up`
  - ⋆ $SM[0, -]$ measures the progress of value 0
  - ⋆ $SM[1, -]$ measures the progress of value 1
- At every process $p_i$:
  - ⋆ a round counter $k_i$
  - ⋆ current local estimate of the decision $est_i \in \{0, 1\}$



## Algorithm

**operation** propose($v_i$) **is**
$est_i \leftarrow v$; $k_i \leftarrow 0$;
**repeat forever**
  $k_i \leftarrow k_i + 1$;
  **if** $\boxed{SM[1 - est_i, k_i] = \texttt{down}}$ % value $(1 - est_i)$ is "late"
    **then** $SM[est_i, k_i] \leftarrow \texttt{up}$; % $p_i$ marks its advance
      **if** $(k_i > 1) \wedge (SM[1 - est_i, k_i - 1] = \texttt{down})$
        % value $(1 - est_i)$ is 2 rounds "late"
        **then** return($est_i$) **end if**
    **else** $est_i \leftarrow (1 - est_i)$ % $p_i$ changes its mind
  **end if**
**end repeat**.



## Selected piece 3

## Memory-Anonymity and Failure-free system

## Mutex object

## Reminder

| names for the global observer | names for process $p_i$ | names for process $p_j$ |
|---|---|---|
| $SM[1]$ | $SM[2]$ | $SM[3]$ |
| $SM[2]$ | $SM[3]$ | $SM[1]$ |
| $SM[3]$ | $SM[1]$ | $SM[2]$ |

No process knows these permutations!



## Mutex object (Lock)

- The oldest (1965) and most important synchro object
- Two operations acquire() and release()
- Imposed "bracket structure" pattern:

  acquire(); critical section; release()*
- Safety property:
  no two processes simultaneously in their critical section
- Progress condition: Deadlock-freedom



## Notion of a symmetric algorithm

- Aim: obtain algorithms "as general as possible", i.e., which rely on "as weak as possible" assumptions
- On the process side:
  - ⋆ Processes have distinct ids, but the same code
  - ⋆ Process identities can only be compared (= or ≠)
  - ⋆ Process identities are not ordered
  - ⋆ No process crashes
- Consider (for the moment) $n = 2$ processes
- Let $m$ be an odd integer, greater than $n = 2$



## Internal representation of the mutex object

The mutex object is represented by

- An array $SM[1..m]$ intialized to $[0, \cdots, 0]$
  - ⋆ known as $SM_i[1..m]$ by $p_i$
  - ⋆ known as $SM_j[1..m]$ by $p_j$
- Ech process $p_i$ manages
  - ⋆ a local index $k_i$
  - ⋆ a local array $sm_i[1..m]$

## Algorithm: operation release()

**operation** release() **is**
**for each** $k_i \in \{1, \cdots, m\}$ **do** $SM_i[k_i] \leftarrow 0$ **end for**;
return().



## Algorithm: operation acquire()

**operation** acquire() **is**
**repeat**
**for** $k_i \in \{1, \cdots, m\}$ **do if** $(SM_i[k_i] = 0)$ **then** $SM_i[k_i] \leftarrow i$ **end**
**for** $k_i \in \{1, \cdots, m\}$ **do** $sm_i[k_i] \leftarrow SM_i[k_i]$ **end for**;
**if** $\boxed{|\{x \text{ such that } sm_i[x] = i\}| < \lceil \frac{m}{2} \rceil}$
**then for** $k_i \in \{1, \cdots, m\}$ **do**
**if** $(SM_i[k_i] = i)$ **then** $SM_i[k_i] \leftarrow 0$ **end if**
**end for**;
**repeat**
**for** $k_i \in \{1, \cdots, m\}$ **do** $sm_i[k_i] \leftarrow SM_i[k_i]$ **end for**
**until** $sm_i[1..m] = [0, \ldots, 0]$ **end repeat**
**end if**
**until** $\boxed{sm_i[1..m] = [i, \ldots, i]}$ **end repeat**;
return().



## On the computability side

- There is a memory-anonymous symmetric deadlock-free mutex algorithm for $n = 2$ processes, which uses $m \geq 2$ atomic registers, if and only if $m$ is odd
- For $n \geq 3$ processes, the existence of a deadlock-free memory-anonymous symmetric mutex algorithm, was an open problem
- Recent solution
  Optimal Memory-Anonymous Symmetric Deadlock-Free Mutual Exclusion
  Z. Aghazaded, D. Imbs, M Raynal, G. Taubenfeld, Ph. Woelfel
- Optimality (NS condition) wrt the size $m$ of the anonymous memory:

$$m \in M(n) = \{m : \forall\ \ell : 1 < \ell \leq n :\ \ \gcd(\ell, m) = 1\}$$



Selected piece 4

Memory-Anonymity
Obstruction-freedom
any number of process crashes

Multivalued consensus object

## Internal representation of the consensus object

The consensus object is represented by

- An array $SM[1..2n-1]$ of atomic read/write registers
- $SM[x]$ is pair $\langle SM[x].id, SM[x].val\rangle$, initialized to $\langle -, \perp\rangle$
  - ⋆ known as $SM_i[1..2n-1]$ by $p_i$
  - ⋆ known as $SM_j[1..2n-1]$ by $p_j$
- Ech process $p_i$ manages
  - ⋆ a local estimate of the decision value $est_i$
  - ⋆ a local index $k_i$
  - ⋆ a local rray $sm_i[1..2n-1]$



## Algorithm

**operation** propose$(v)$ **is**
$est_i \leftarrow v$;
**repeat**
**for each** $k_i \in \{1, \cdots, 2n-1\}$ **do** $sm_i[k_i] \leftarrow SM_i[k_i]$ **end for**;
**if** $\boxed{\exists v \neq \perp : |\{k \text{ such that } sm_i[k].val = v\}| \geq n}$
**then** $est_i \leftarrow v$
**end if**;
**if** ($\exists\ x \in \{1, \cdots, 2n-1\}$ such that $sm_i[x] \neq \langle i, est_i\rangle$)
**then** $SM_i[x] \leftarrow \langle i, est_i\rangle$
**end if**
**until** $\boxed{sm_i[1..2n-1] = [\langle i, est_i\rangle, \cdots, \langle i, est_i\rangle]}$ **end repeat**;
return$(est_i)$.



Selected piece 5

Process-Anonymity, Obstruction-freedom

Universal construction

Bouzid Z., Raynal M., and Sutra P.,
Anonymous obstruction-free $(n,k)$-set agreement with $(n-k+1)$ atomic read/write registers.
*Distributed Computing*, 31:99-117 (2018)



## $k$-Set agreement ($k$-SA)

- Safety properties:
  - ⋆ Validity: If a process decides a value, this value was proposed by a process
  - ⋆ Agreement: At most $k$ different values are decided
- Liveness property: obstruction-freedom

1-SA is consensus
$k$-SA is strictly stronger than $(k+1)$-set agreement

## A result

- $(n-k+1)$ atomic registers are sufficient to build an obstruction-free $k$-SA object in an $n$-process-anomymous system
- Optimal fo repeated $k$-set agreement (a single $k$-SA object)
- Case $k=1$: 1-SA is consensus:
  - ⋆ At least $(n-1)$ atomic read/write regsiters are necessary in non-anonulous systems 1-SA is consensus
    - Zhu L.,A tight space bound for consensus. *Proc. 48th STOC*, pp. 345-350 (2016)
  - ⋆ Conjecture: $n$ regsiters are necesary and sufficient



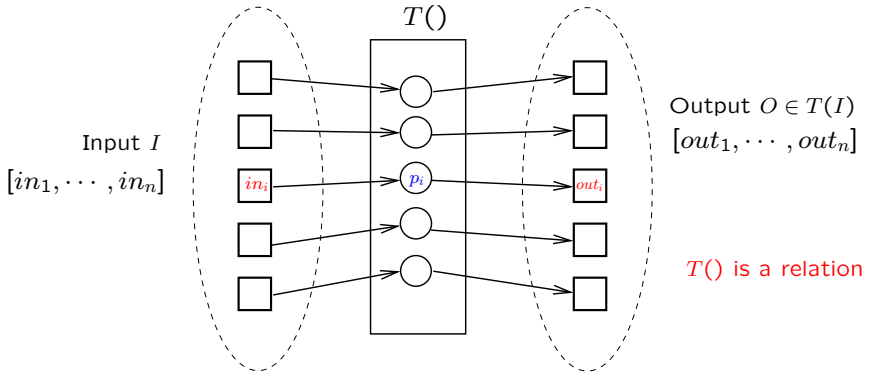
## Power of repeated anomymous OB-free $k$-SA (1)

On the object side

- Let $O$ be object that can be obstruction-free implemented by $n$ anonymous processes and any number of MWMR atomic read/write registers
- $O$ can be obstruction-free implemented by $n$ anonymous processes and $n$ MWMR atomic read/write registers



## Distributed task






## Power of repeated anomymous OB-free $k$-SA (2)

On the distributed task side $T = (\Delta, \mathcal{I}, \mathcal{O})$

- If a (colored/colorless) task is obstruction-free solvable by $n$ anonymous processes and any number of MWMR atomic read/write registers, then it is obstruction-free solvable by $n$ anonymous processes with no more than $n$ MWMR atomic read/write registers
- If a colorless task $T$ is obstruction-free solvable in a non-anonymous $n$-process system using any number of SWMR atomic registers, it is also obstruction-free solvable in an anonymous $n$-process system with $n$ MWMR atomic registers

# Conclusion



## A summary table

| Object | Anonymity | Failure | Prog. | Reference |
|---|---|---|---|---|
| snapshot | processes | crash | WF | Guerraoui-Rupert 07 |
| Binary cons. | processes | crash | OB | Chandra 96, Guerraoui-Rupert 07 |
| $k$-Set Agr. | processes | crash | OB | Bouzid-Raynal-Sutra 18 |
| Mutex ($n = 2$) | memory | no failure | DF | Taubenfeld 17 |
| Mutex ($n \geq 2$) | memory | no failure | DF | AIRTW 2018 |
| Consensus | memory | crash | OB | Taubenfeld 17 |



- A gentle introduction to
  - ⋆ process-anonymity
  - ⋆ memory-anonymity
- Computability issues
- Promising research domain
  we are far from knowing everything!
- Many open problems



## More important: He told me


"Algorithms are at the core of Informatics"

## And, maybe more important, SHE told me

"Synchronization and non-determinism are among their most fundamental concepts"