BLOCKCHAIN:
Buzzword/Engineering/Science?

Michel Raynal

Institut Universitaire de France

Academia Europaea

IRISA, Université de Rennes, France

Polytechnic University (PolyU), Hong Kong

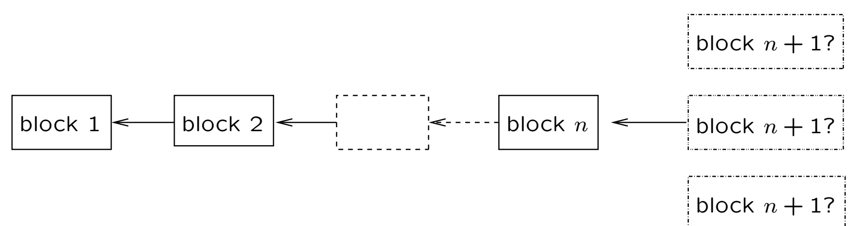
A global view

A few facts

- A few publications (for the moment?)
- More and more prototypes
- Claimed: possibly numerous applications

What is a blockchain?

- A sequential data structure (its “atoms” are called blocks)
- A ledger = sequence (list) of linked blocks
- Operations: append a new block, read the whole chain
- A block: sequence of records (e.g., transactions)



Basic distributed computing problems

- Coordination, synchronization, agreement
- Fault-tolerance
- According to the applications
  - ⋆ Security
  - ⋆ Anonymity (Anonymous vs pseudonymous)
  - ⋆ Dynamicity

More specifically

Seem to be central in blockchains

- Distributed agreement
  to build mutual trust without centralized control
- Cryptography: public key cryptosystems (RSA-like)
  - ⋆ to sign
  - ⋆ to hide
- Dynamicity

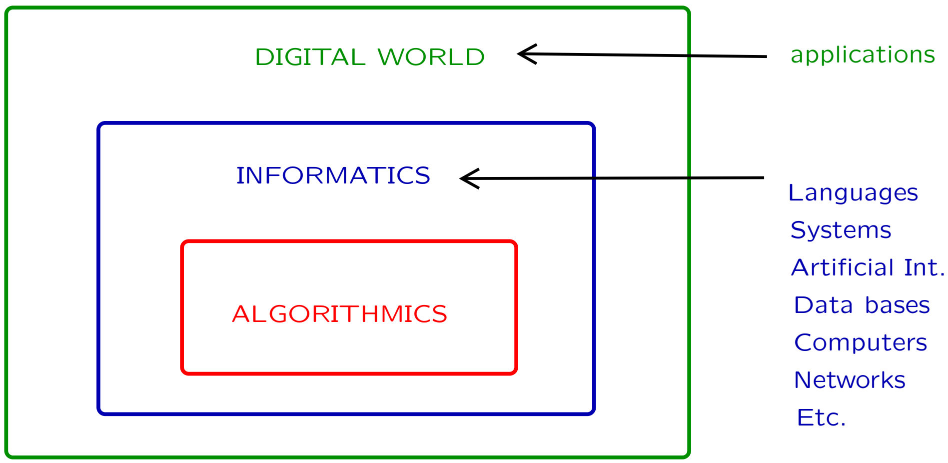
They are distinct issues

Historical perspective: from exchange to trust

- Internet: democratization of exchanges
  - ⋆ Communication (email)
  - ⋆ Centralized services (data centers)
  - ⋆ Publish/subscribe
  - ⋆ Web pages (queries/responses)
  - ⋆ Information (wikipedia)
  - ⋆ Etc.
- Blockchain: mutual agreement-based trust
  - ⋆ store, authenticate, verify, trust
  - ⋆ without centralized control/authority

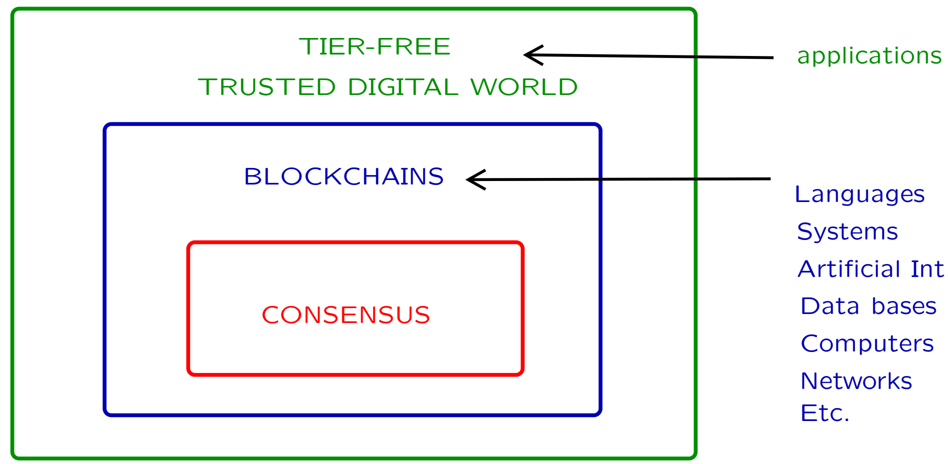
Looking to the basics

What is informatics?



"Imitation game": What is a blockchain?

A view of the position of blockchains



A past history

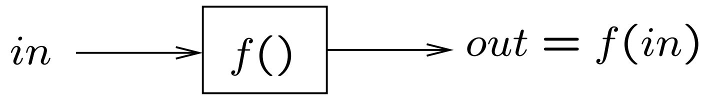
Centralized operating systems in a single slide

- Before Dijkstra: a set of tricks (prehistory era)
- After Dijkstra: history era starts in 1965
  - ⋆ Processes
  - ⋆ Synchronization,
  - ⋆ Weakest pre-condition
    (why to ask more when we could ask less?)
  - ⋆ Etc.
- : We have concepts, we can reason, prove, transmit, understand, explain, etc. (no more "hand-waving"-only)

Going at the scientific heart

Fault-tolerant distributed computing

Reminder: the basic unit of sequential computing



- The notion of a function
- Sequential algorithm
- The notion of computability (Turing machine)
- The notion of impossibility (e.g., halting problem)
- The fundamental hierarchy
  FSA $\subset$ Pushdown Automata $\subset$ Turing Machines
- Church-Turing's Thesis

The case of parallel computing

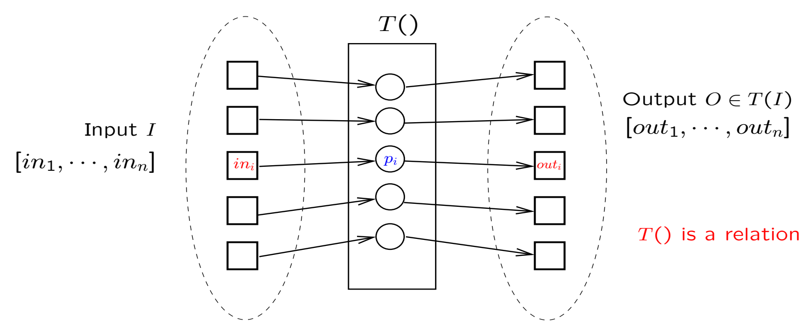
- We look inside the box implementing $f()$
  - ⋆ mono-processor
  - ⋆ multiprocessor : to be more efficient
- The problem could be solved by a sequential algorithm, but can be solved more efficiently with several computing entities
- Parallel computing is an "extension" of sequential computing looking for efficiency
- The aim is to benefit as much as possible from data independance
- This has a long story and introduced new techniques and concepts (e.g., task graphs, scheduling, etc.)

What is distributed computing?

DC arises when one has to solve a problem in terms of entities (processes, agents, sensors, peers, actors, nodes, processors, ...) such that each entity has only a partial knowledge of the many parameters involved in the problem that has to be solved

DC is about Mastering UNCERTAINTY

The basic unit of distributed computing



- The notion of a task: from an input vector to an output
- The inputs are DISTRIBUTED (this is imposed and not under the control of the algorithm designer)
- Distributed computing $\neq$ parallel computing
- Failures belong to the model (in nearly all cases)

The notion of a (distributed) task

- A task T is a triple $(\mathcal{I}, \mathcal{O}, \Delta)$
  - ⋆ $\mathcal{I}$: set of input vectors (of size n)
  - ⋆ $\mathcal{O}$: set of output vectors (of size n)
  - ⋆ Δ: relation from $\mathcal{I}$ into $\mathcal{O}$: $\forall I \in \mathcal{I}: \Delta(I) \subseteq \mathcal{O}$
- $I[i]$: private input of p_i
- $O[i]$: private output of p_i
- $\forall I \in \mathcal{I}$:
  $\Delta(I)$ defines the set of legal output vectors that can be decided from the input vector I

Distributed computing: birth certificates

"From prehistory to history"

- L. Lamport, Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558-565 (1978)
  - ⋆ Partial order on events
  - ⋆ Scalar clocks
  - ⋆ State machine replication
- 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)
  - ⋆ Impossibility result in asynch. crash-prone systems
  - ⋆ Notion of valence (captures non-determinism)

A few historical dates: victories & defeats

- Distributed state machine: 1978
- Byzantine processes (synchronous systems): 1980
- Impossibility of consensus: 1985

A famous quote ... and its formalization

- "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)
- 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)

Reminder: DC is about Mastering UNCERTAINTY!

To summarize

- Real-time: masters On-time computing
- Parallelism: provides Efficiency
- Distributed computing:

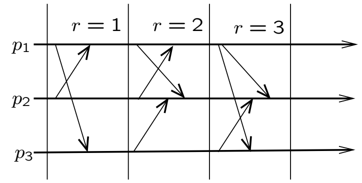
masters Uncertainty

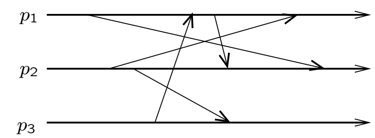
(We are -more or less- implicitly using a lot of heuristics!)

Fundamental issue:
cope with the non-determinism created by the environment (asynchrony, failures, mobility, dynamicity, etc.)

DC: Fundamental basic notions

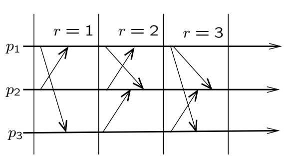
- process model: sequential computing entities
- distributed control: each process is client and server
- communication model: messages vs shared memory
- timing model: synchronous vs asynchronous
- failure model: process crash vs Byzantine behavior

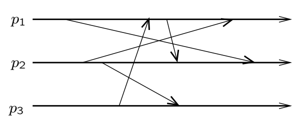




DC: timing models

- Synchrony: lock-step progress
  - ⋆ Processes execute a sequence of synchronized rounds
  - ⋆ A round: broadcast, receive, local computation
  - ⋆ Messages received in the round they were sent
- Asynchrony: no timing assumption
  - ⋆ No assumption on process speed (arbitrary)
  - ⋆ No assumption on message transfer delays (finite)



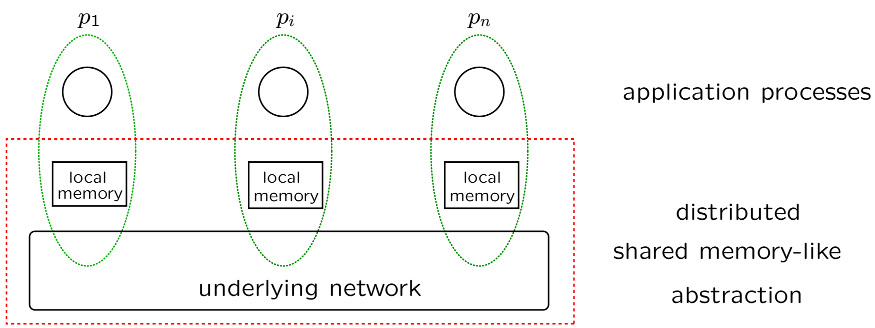


DC: failure models

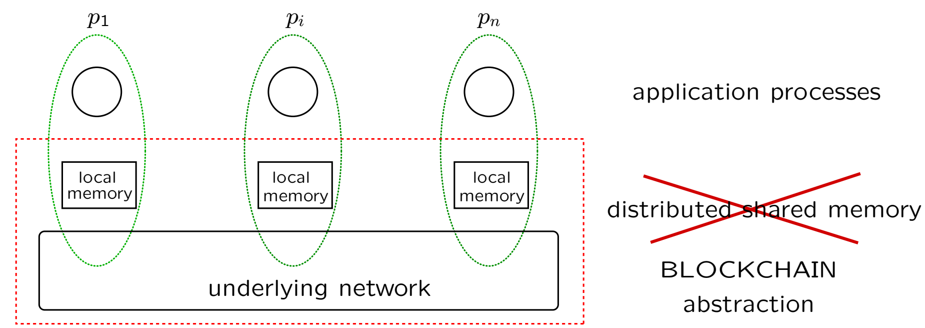
- No failure: irrealistic
- Crash failure
  - ⋆ premature unexpcted halt
  - ⋆ a process behaves correctly until it (possibly) crashes
- Byzantine behavior
  - ⋆ arbitrary behavior (malicious or not)
  - ⋆ incorrect behavior vs incorrect inputs
  - ⋆ Signatures can restrict bad behaviors

- Lamport L., Shostack R., and Pease M., The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3)-382-401 (1982)

A classical architecture



A "small" change...



Three fundamental problems

- Implement reliable broadcast
- Build a read/write memory (atomic RW register)
- Distributed agreement among participants (consensus)

Appear in one way or another in the implementation of a blockchain

What can be done? What cannot be done?

$\boxed{t}$ = maximal number of processes that can be faulty

Reliable broadcast: definition

- Validity:
  - ⋆ Crash model: if a process delivers a message from a process p, this message has been broadcast by p
  - ⋆ Byzantine model: if a correct process delivers a message m from a correct sender, then the sender broadcast m
- Integrity: no message is delivered more than once
- Termination:
  - ⋆ Crash model: if (a) a correct process broadcasts a message or (b) a process delivers a message m, then all correct processes deliver m
  - ⋆ Byzantine model: if a correct process broadcasts or delivers a message m, then all correct processes deliver m

Reliable broadcast in a crash model poss./imposs.

- Reliable network: $t < n$
- Network with fair channels:
  - ⋆ $t < n/2$
  - ⋆ Additional computability power is needed to have a quiescent algorithm

    The quiescence property is related to implmentations: an application message cannot give rise to an infinity of protocol messages

Fair channel: intuitively a channel can experience transient message losses

Reliable broadcast in a Byz model poss./imposs.

- Reliable network
- $t < n/3$
- Tradeoff:

	fault resilience	communication steps message types	number of messages
Bracha	$t < n/3$	3	$2n^2 - n - 1$
Imbs-Raynal	$t < n/5$	2	$n^2 - 1$

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

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

Implement an atomic register

- Asynchronous crash failure model: $t < n/2$

  - Attiya H., Bar-Noy A. and Dolev D., Sharing memory robustly in message passing systems. *Journal of the ACM*, 42(1):121-132 (1995)

- Asynchronous Byzantine failure model: $t < n/3$

  - Imbs D., Rajsbaum S., Raynal M., and Stainer J.,
  Read/Write shared memory abstraction on top of an asynchronous Byzantine message-passing system. *Journal of Parallel and Distributed Computing*, 93-94:1-9 (2016)

Consensus: definition

- Validity:
  - ⋆ Crash model: a decided value is a proposed value
  - ⋆ Byzantine model: il all correct processes propose the same value, this value is decided
- Agreement:
  - ⋆ Crash model: no two different processes decide different values
  - ⋆ Byzantine model: no two correct processes decide different values
- Termination:

  If a correct process invokes propose() it decides a value

A fundamental result of FT distributed computing

Fischer-Lynch-Paterson's Impossibility result (1985)

- There is no deterministic algorithm that wait-free implements a consensus object
  - ⋆ Whatever the number of processes $n \geq 2$
  - ⋆ Whatever the communication medium (read/write registers or message-passing)
  - ⋆ Even if a single process may crash
  - ⋆ Even if processes have to agree on a single bit!

- 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)

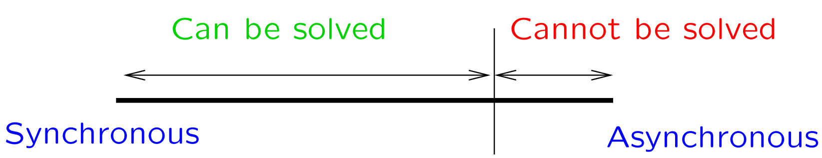
How to circumvent consensus impossibility

Three approaches

- Add an oracle (which provides additional computational power)
  - ⋆ Failure detectors
  - ⋆ Randomization
- Restrict the set of input vectors

The failure detector approach

- Given a problem: Find the weakest "assumptions" that has to added to an asynchronous system in order problems can be solved



Failure detectors

- Provide each process with a read-only local variable giving (possibly unreliable) information on failures
- Given a problem (object), give as few information as possible while allowing the object to be implemented
- According to the information on failures that is given, several "classes" of failure detectors can be defined

- Chandra T. and Toueg S., Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225-267 (1996)

- Raynal M., *Communication and agreement abstractions for fault-tolerant asynchronous distributed systems.* Morgan & Claypool Pub., 251 pages (2010)

The weakest failure detector to solve consensus

- Ω: provides each process p_i with a read-only local variable $leader_i$ such that, after an unknown but finite time, the variables $leader_i$ of the non-crashed processes contain forever the same process identity of a non-crashed process
- Ω: weakest FD that allows consensus to be solved

- Chandra T.D., Hadzilacos V. and Toueg S., The Weakest Failure Detector for Solving Consensus. *Journal of the ACM*, 43(4):685-722 (1996)

- Lamport L., The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133-169 (1998) (First version appeared as DEC Research Report #49, September 1989)

- Fernández A., Jiménez E., Raynal M., and Trédan G., A timing assumption and two t-resilient protocols for implementing an eventual leader service in asynchronous shared-memory systems. *Algorithmica*, 56(4):550-576 (2010)

The notion of an indulgent distributed algorithm

- A distributed algorithm is indulgent with respect to a failure detector FD it uses to solve a problem Pb if
  - ⋆ it always guarantees the safety property defining Pb (i.e., whatever the correct/incorrect behavior of FD),
  - ⋆ and satisfies the liveness property associated with Pb at least when FD behaves correctly
- Hence, when the implementation of FD does not satisfies its specification, the algorithm may not terminate, but if it terminates its results are correct
- All Ω-based algorithms are indulgent
- Notions of stable vs unstable periods

- Guerraoui R., Indulgent algorithms. *Proc. 19th ACM Symposium on Principles of Distributed Computing (PODC'00)*, ACM Press, pp. 289-298 (2000)

Randomization

- A classical way to break non-determinism
- Asynchronous round-based algorithms
- Requires to modify the termination property which becomes:

  The probability that a non-faulty process has decided by round r tends to 1, when the number of rounds tends to $+\infty$
- Notion of expected number of rounds to decide

- Ben-Or M., Another advantage of free choice: completely asynchronous agreement protocol. *Proc. 2d ACM Symposium on Principles of Distributed Computing (PODC'83)*, ACM Press, pp. 27-30, (1983)

- Mostéfaoui A., Moumen H., and Raynal M., Signature-free asynchronous binary Byzantine consensus with $t < n/3$, $O(n^2)$ messages, and $O(1)$ expected time. *Journal of ACM*, 62(4), Article 31, 21 pages (2015)

Beyond state machines and blockchains

1 out-of k

- Assume n sequential state machines (blockchains)
- Requirement:
  at least one of them must progress forever

Gafni E. and Guerraoui R., Generalizing universality.
Proc. 22nd Int'l Conference on Concurrency Theory (CONCUR'11), Springer LNCS 6901, pp. 17-27 (2011)

ℓ out-of k

- Assume n sequential state machines (blockchains)
- Requirement:
  at least ℓ of them most progress forever, $1 \leq \ell \leq k$

Raynal M., Stainer J., and Taubenfeld G., Distributed universality.
Algorithmica, 76(2):502-535 (2016)

ℓ out-of k (cont'd)

- Introduces (k, ℓ)-consensus objects (k, ℓ constant)
- Considering k objects (seq. state machines, blockchains) , it introduces a (k, ℓ)-universal construction
  - ⋆ in which ℓ ($1 \leq \ell \leq k$) objects progress forever
  - ⋆ in which the progress condition is wait-freedom
  - ⋆ that is contention-aware (only read/write registers are used in the absence of contention)
  - ⋆ that is generous wrt to the obstruction-freedom progress condition
- Shows that (k, ℓ)-consensus objects are necessary and sufficient for such a (k, ℓ)-universal construction

CONCLUSION

What has been learnt

- A visit to
  state machine replication in asynchronous systems

  What can/cannot be done in static asynch Byz systems

- A try to understand blockchain concept:

  is the novelty:

  Synchronous state machine replication
  in dynamic systems?

- To be done:

  Extract the problems, concepts and abstractions which are new wrt state machine duplication and solve them!

- Looking at the future: coordination of blockchains? (biology-like)

Two of my leimotivs

- When something works, we need to kow why it works, and ...
  when something does not work, we need to kow why it does not work

- Correctness may be theoretical ...
  but incorrectness has practical impact!

A few recent research papers on blockchain consensus

- Crain T., Gramoli V., Larrea M. and Raynal, M., (Leader/Randomization/Signature)-free Byzantine Consensus for Consortium Blockchains. https://arxiv.org/abs/1702.03068 (2017)
- Eyal I., Gencer A.E., Sirer E.G., and van Renesse R., Bitcoin-NG: a scalable blockchain protocol. *Proc. 13th Usenix Conference on Networked Systems Design and Implementation (NSDI'16)*, pp.45-59 (2016)
- Hearn M., Corda: a distributed ledger. Version 0.5 (2016)
- Kwong J., Tendermint: Consensus without mining. v.0.7 (2016)
- Luu L., Narayanan V., Zheng C., Baweja K., Gilbert S. and Saxena P., A secure sharding protocol for open blockchains. *ACM Conference on Computer and Communications Security (CCS'16)*, ACM Press, pp. 17-30 (2016)
- Mazieres D., The stellar consensus protocol: A federated model for internet-level consensus (2015)
- Natoli C. and Gramoli V., The balance attack against proof-of-work blockchains: The R3 testbed as an example. https://arxiv.org/abs/1612.09426 (2016)
- Natoli C. and Gramoli V., The blockchain anomaly. *Proc. 5th IEEE Int'l Symposium on Network Computing and Applications (NCA'16)*, IEEE ComputerPress, pp. 310-317 (2016)