Data in Cloud Computing

RDBMS

- Lots of work/research for the past 40 years
- Mostly centralized model → 1 machine { HD1, HD2, HD3 } \ / RAID / ZFS
- Different cost model than in the past
- Different paradigm than most programming languages
- Provide a lot of guarantees

données BD ≠ données prog.

ACID

- Andreas Reuter & Theo Härder in 1983.
- Atomicity
- Consistency
- Isolation
- Durability

Une transaction doit réussir ou échouer complètement

Le état valide → état valide

Plusieurs transactions concurrentes doivent amener la base dans le même état que si elles avaient été exécutées séquentiellement.

Le résultat d'une transaction doit être maintenu malgré les pannes

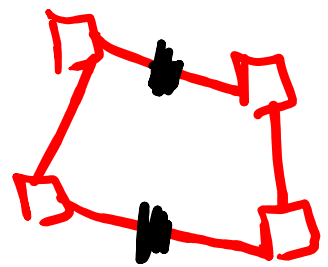
What now ?

- More data (like really more)
  - Not all well structured/organized
- Cheap hardware and not so cheap engineers
  - Many machines, 1 engineer
  - The network is everywhere
  - Machines or network **will** fail
- Is ACID possible in a distributed environment ?

CAP – 1

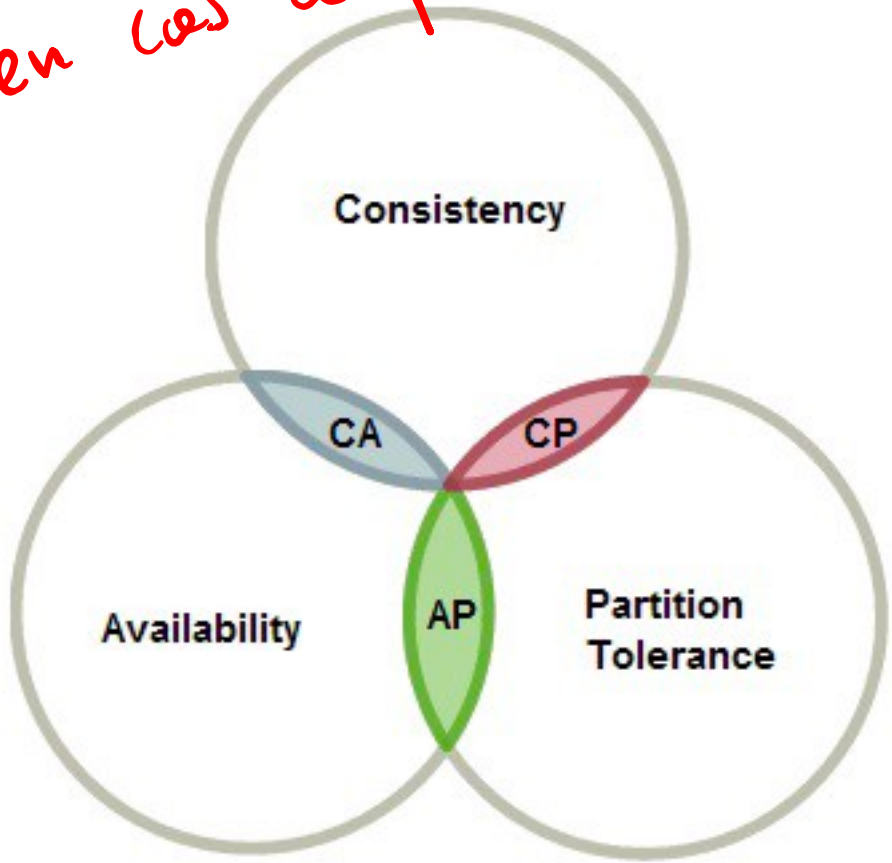
- Eric Brewer, PODC keynote (2000)
- 3 properties to build large scale distributed systems
- Consistency → op. élémentaires
- Availability → dispo en permanence
- Partition Tolerance

Le système doit continuer à fonctionner quand un ou plusieurs nœuds sont isolés des autres.



CAP - 2

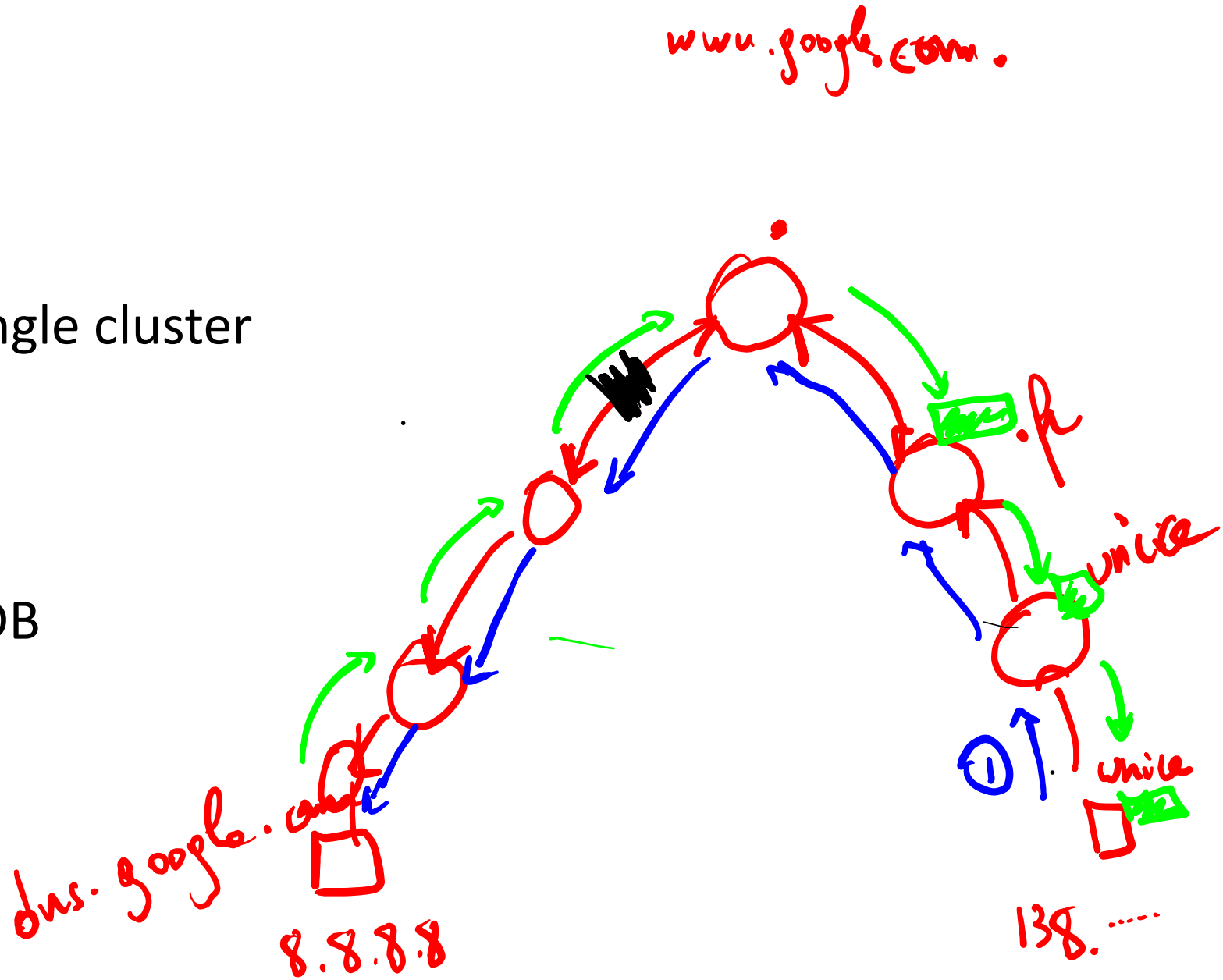
Si AP : on peut être cohérent sauf en cas de panne.



https://towardsdatascience.com/cap-theorem-and-distributed-database-management-systems-5c2be977950e

CAP in the wild

- A distributed DB on a single cluster
  - C-A
- Domain Name Server
  - A – P
- A multi-site distributed DB
  - C - P



BASE

- ACID is not possible in a distributed environment
- Move to less strict guarantees
- **B**asically **A**vailable
- **S**oft-State
- **E**ventual **C**onsistency

Le système répondra toujours avec des données qui pourront être inconsistantes, non dispo, ou en cours de modification.

L'état du système peut changer sans entrées extérieures

Le système deviendra éventuellement cohérent si il n'y a plus d'entrées extérieures

What is consistency

- A contract between a database and a programmer
  - Follow some rules and your data will be consistent
- Many different models
  - Strict, sequential, causal, eventual…
  - Ordered from strong to weak

Strict consistency

- A write to a variable is instantaneously seen by all processors

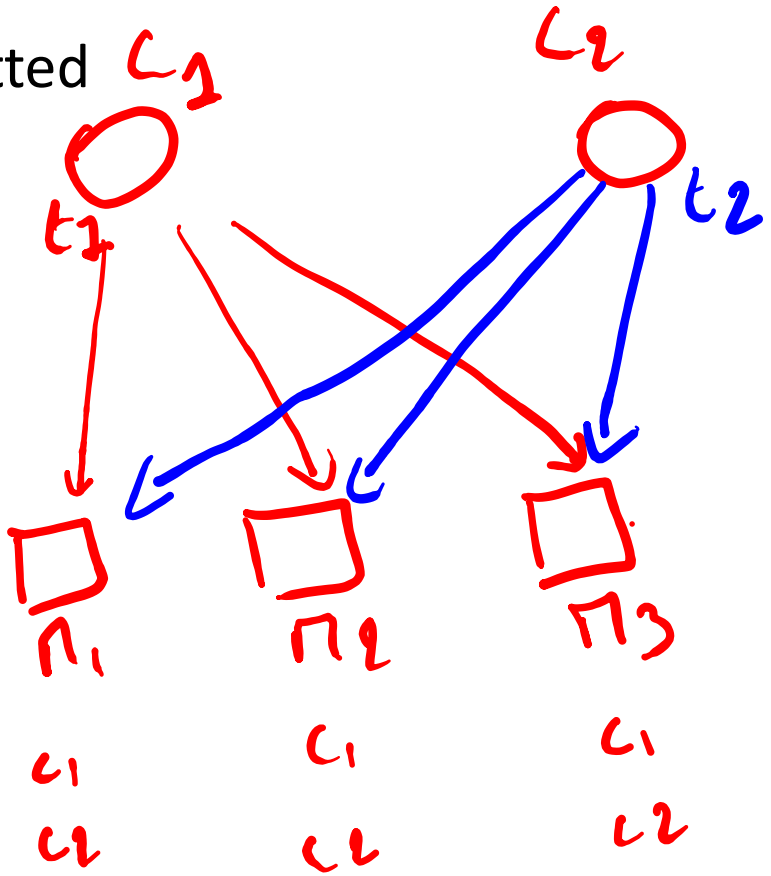
Sequence	Strict model		Non-strict model	
	P1	P2	P1	P2
1	$W(x)1$		$W(x)1$	
2		$R(x)1$		$R(x)0$
3				$R(x)1$

https://en.wikipedia.org/wiki/Consistency_model#Strict_consistency

Atomic Consistency

- Operations are executed in the same order on all machines
  - Uses a global clock
  - Same order as they were emitted
- Always deterministic

google Spanner



NTP

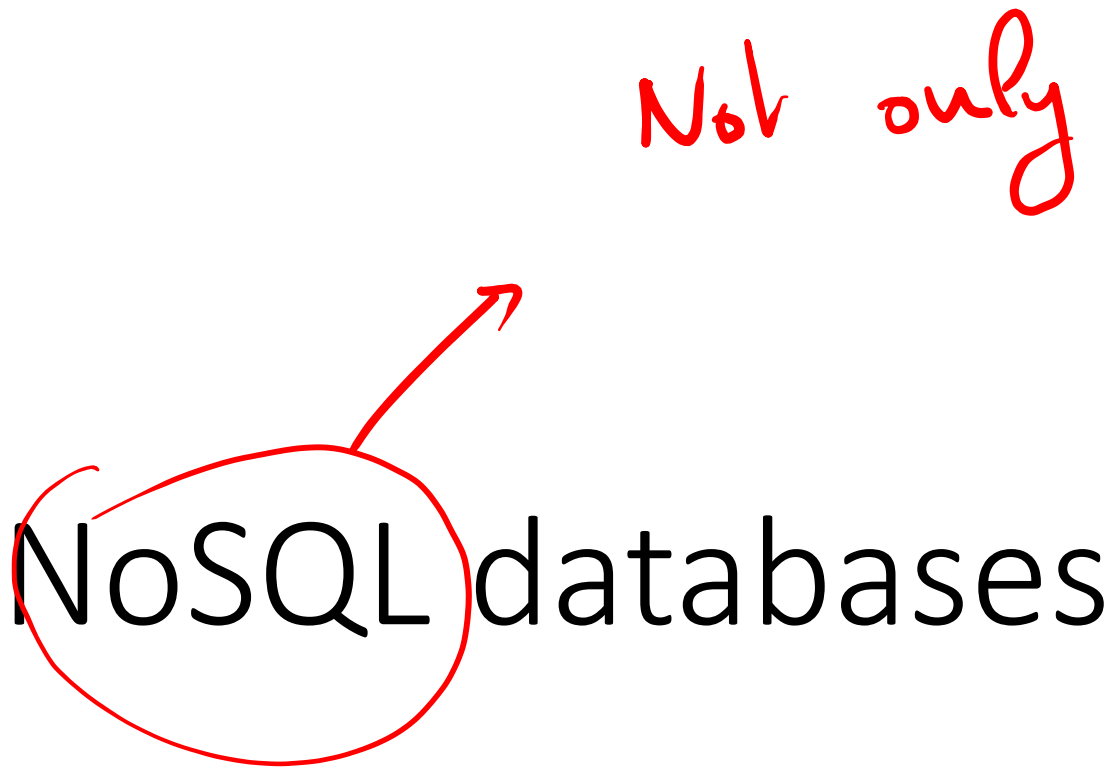
atomic clock

Sequential Consistency

- Weaker than strict consistency
- All write operations by multiple processors have to been seen in the same order
  - No specific order initially
  - Not necessarily consistent between various executions
- Sequential consistency + time => atomic consistency (e.g Google Spanner)

Eventual Consistency

- Weak consistency
  - Given enough time without update, all read access to a variable will return the latest value.


Not only
NoSQL databases

Principles

- Not Only SQL
- All follow the BASE principles
- Provides various properties under CAP
- Designed to scale horizontally
- Replication
  - Data is copied on multiples machines
- Various designs

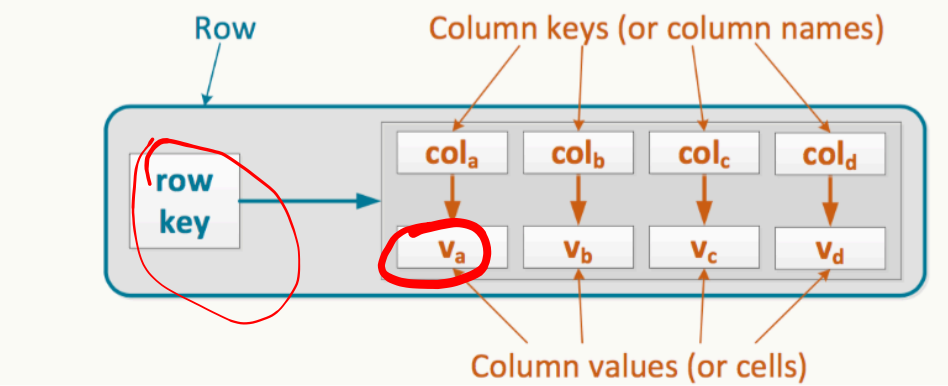
Key-Value

- Data are stored as unique key-value pairs
- Very simple API
  - Get, put, delete
  - Range queries often not supported
- Usually relies on consistent hashing
  - Spread keys among multiples machines
  - Copy pairs for redundancy
- Examples : DynamoDB, Redis, Riak

Pb de biais

Wide Column

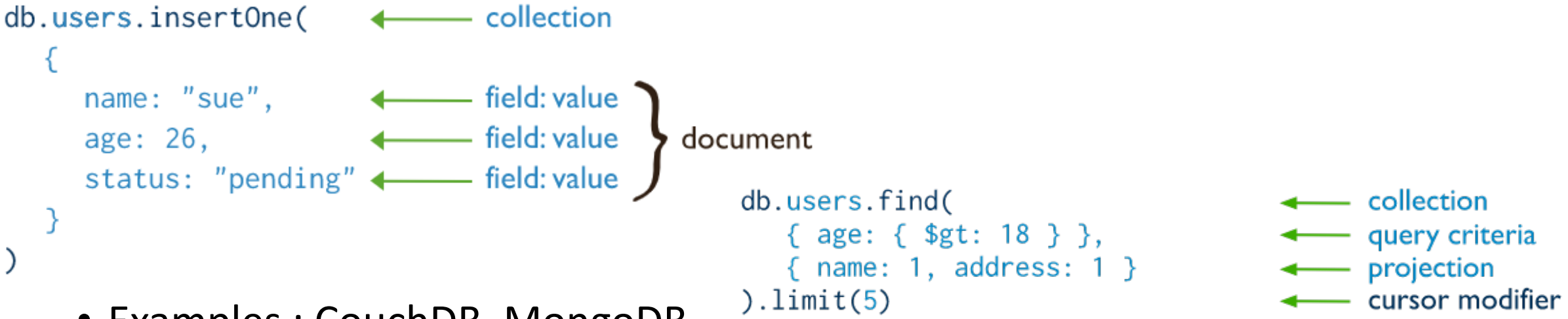
- Use row/columns to store data
  - Like RDBMS except columns have usually no fixed type
  - Number of columns can vary from row to row
- Can be seen as a 2D key-value store
- Examples : Apache Hbase, Cassandra



https://pandaforme.gitbooks.io/introduction-to-cassandra/understand_the_cassandra_data_model.html

Document

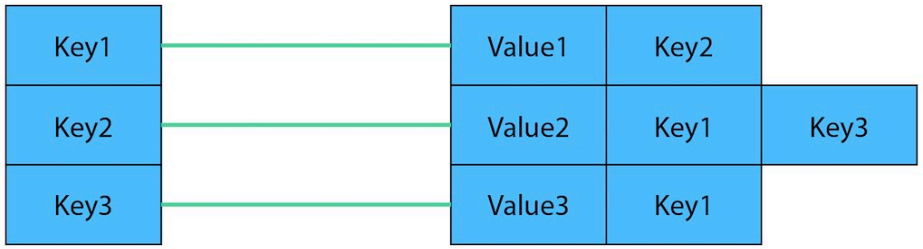
- Data are stored as documents (XML, JSON…)
  - Rich data structures
  - Support versioning
- An API allows complex queries

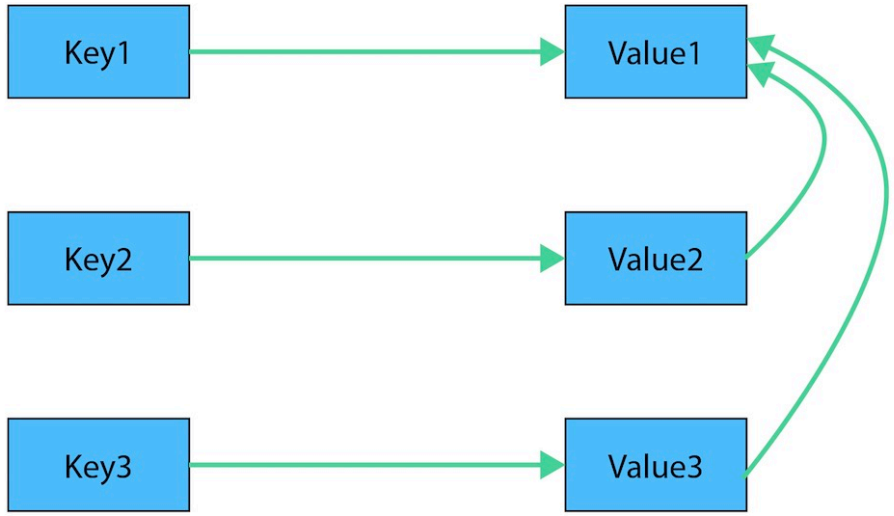


- Examples : CouchDB, MongoDB

Graph Oriented

- Consider data as graphs
  - Introduce relations more complex than key-value





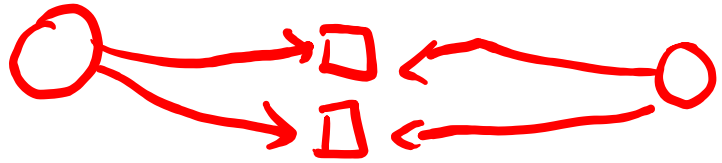
  - Examples : Neo4J, RedisGraph

https://neo4j.com/developer/graph-db-vs-nosql/

Replication and consistency

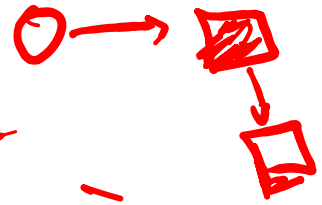
Definitions

- Copies of a data are called replicates
- A group of machines storing the same data is called a replication group
- Different ways to update replicates
  - Active replication
  - Passive synchronous replication
  - Passive asynchronous replication
  - Optimistic replication



Les requêtes d'un client sont envoyées à tous les replicats + seq. consistency

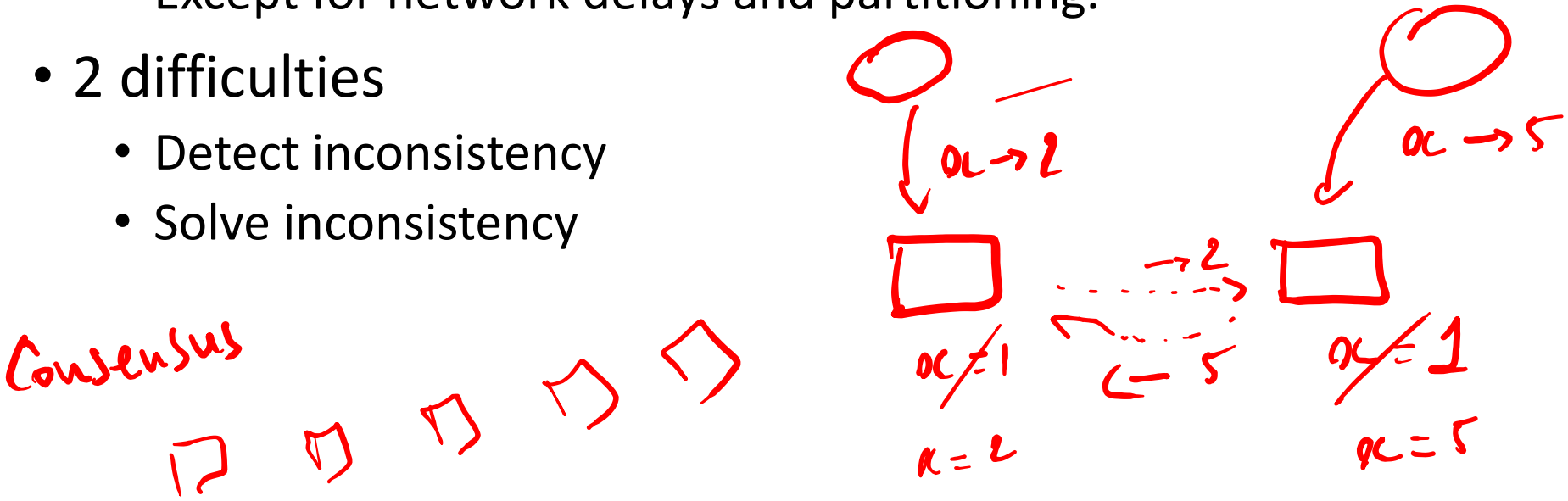
Un replicat est master et fait la mise à jour. Le client est bloqué.



N'importe quel replicat peut accepter une mise à jour.

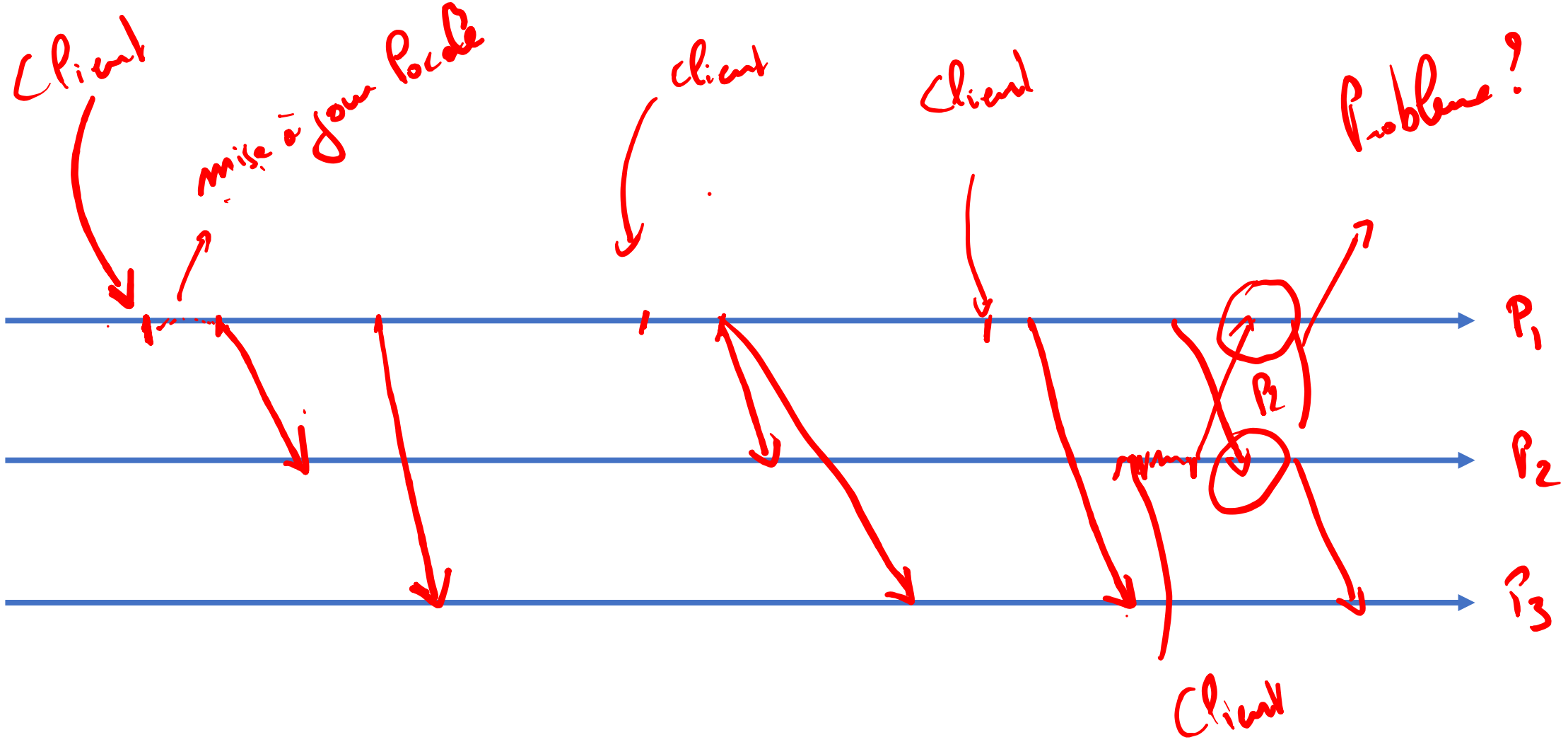
Ensuring consistency with optimistic replication

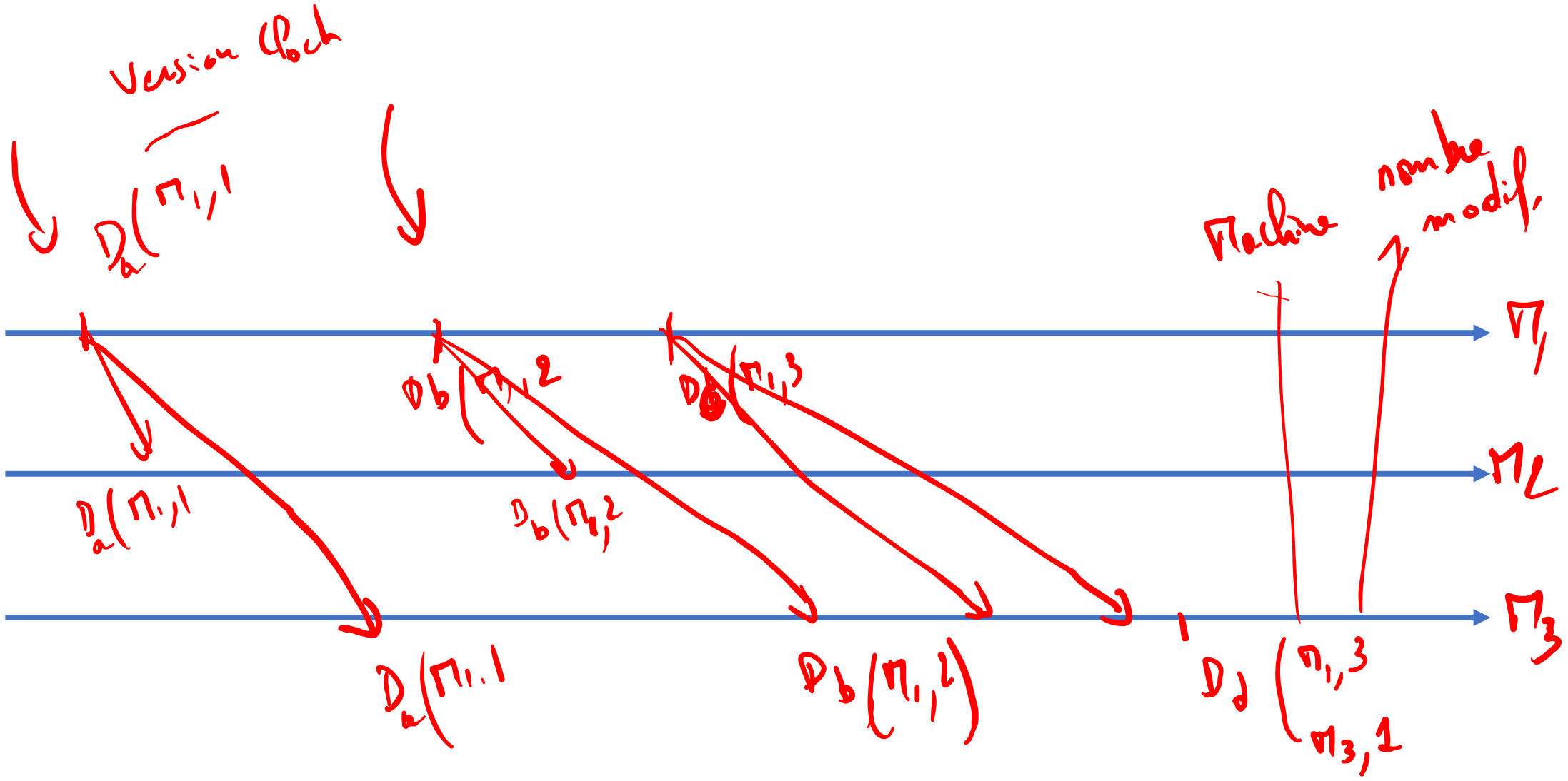
- Any machine can update data
- Should work properly most of the times
  - Except for network delays and partitioning.
- 2 difficulties
  - Detect inconsistency
  - Solve inconsistency



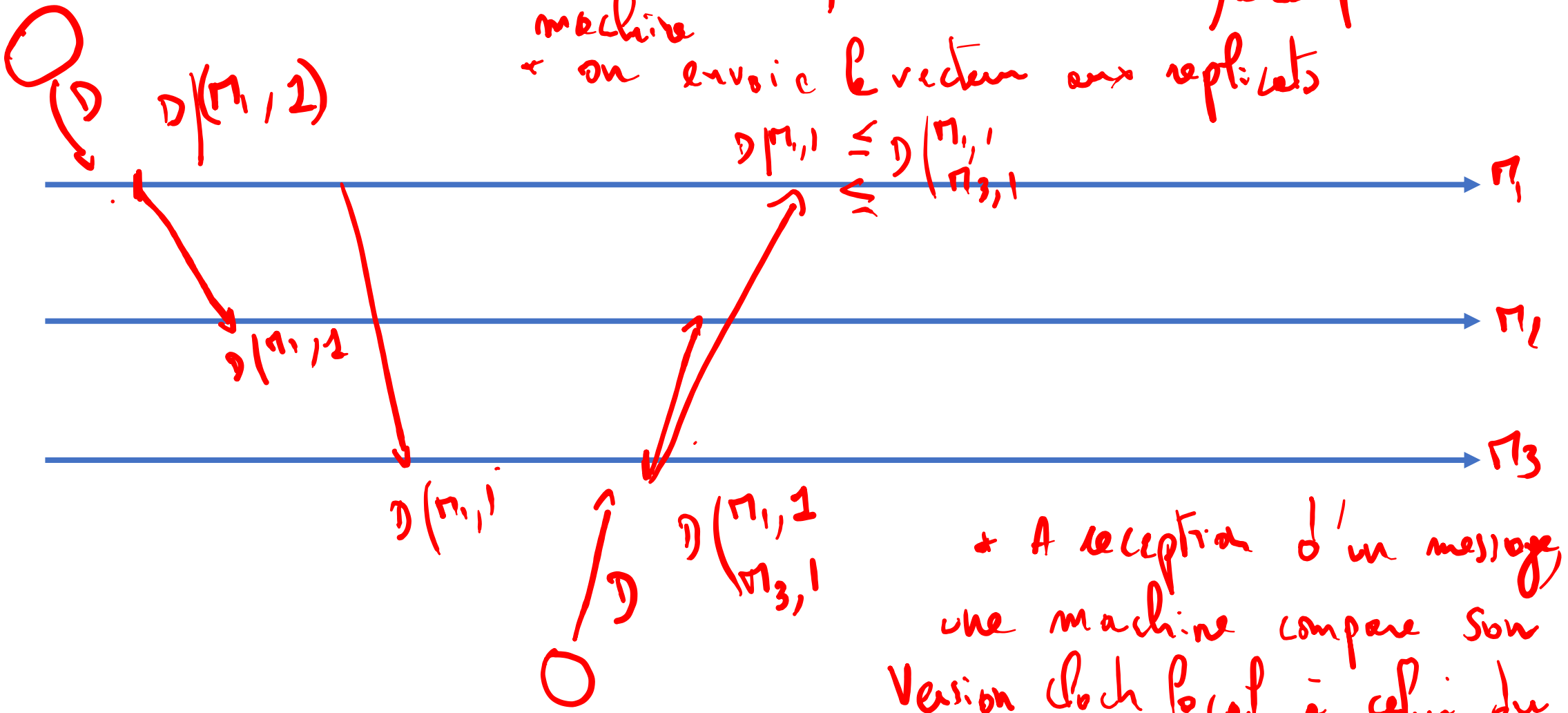
Version clocks (aka Version Vectors)

- Inspired by Vector Clocks
  - Inspired by Lamport's clocks.
- Associate an update vector to each data
  - Vector size is number of machines
  - Each update by client increase counter of corresponding machine
  - When update sent to replica, send the whole counter
- Update is valid is received vector > local vector
  - Only look at entries existing in both vectors

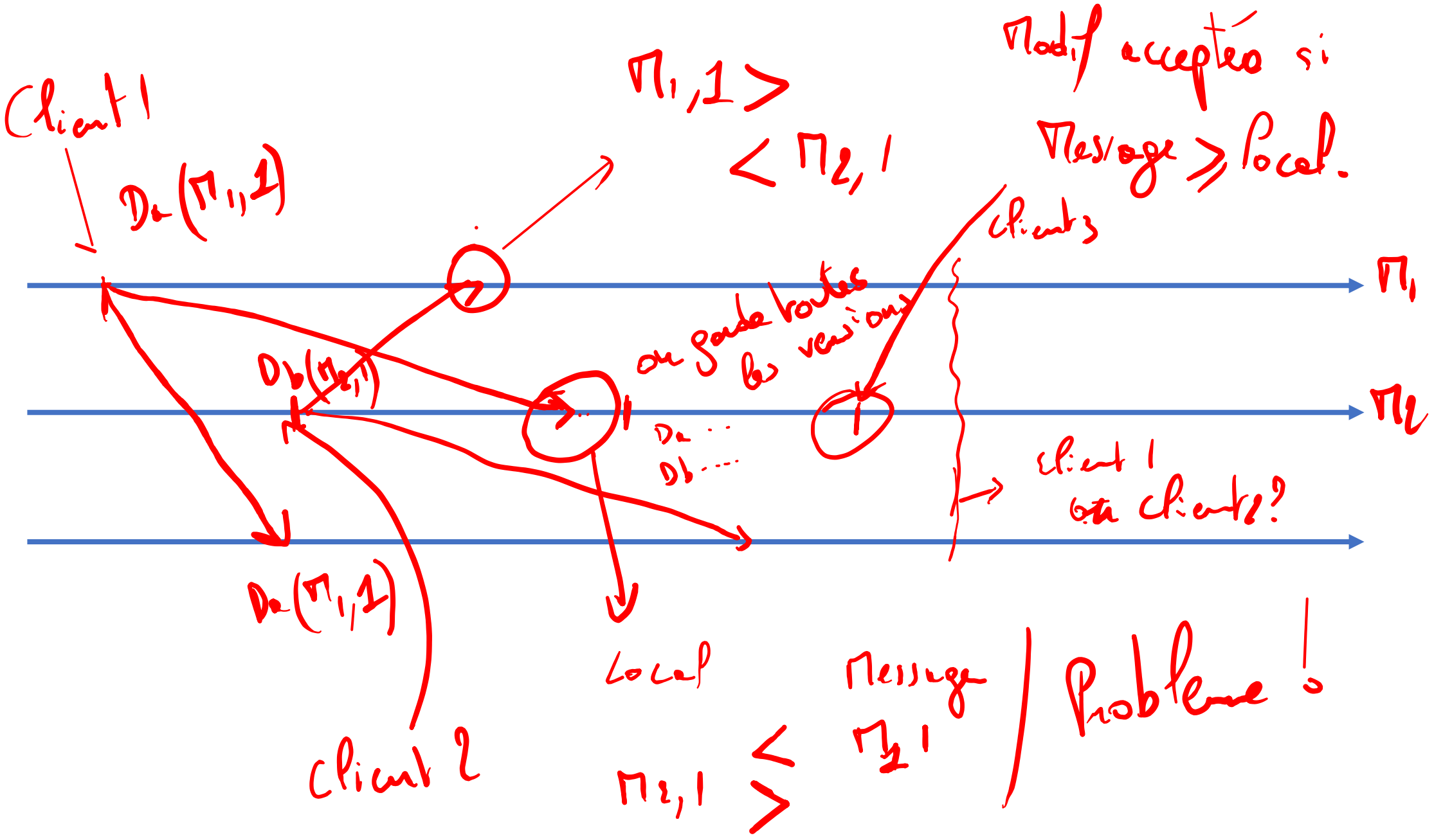

Client
mise à jour Poche
client
Client
Problème ?
P1
P2
P3
P2
Client


Version Bach
Machine
nombre modif.

* on associe à chaque donnée un vecteur de taille # replicat
* chaque entrée = nbre de modifs de la donnée faite par 1 machine
* on envoie le vecteur aux replicats



* A réception d'un message une machine compare son Vector Clock local à celui du message.


Client 1
$D_a(\pi_{1,2})$
$\pi_{1,1} >$
$< \pi_{2,1}$
Modif acceptée si
Message ≥ Local.
Clients
π_1
π_2
$D_b(\pi_{2,1})$
ou seule toutes les versions
$D_a \ldots$
$D_b \ldots$
client 1
tous clients?
$D_a(\pi_{1,2})$
Local
Message $\pi_{2,1}$
$\pi_{2,1} <$ / Problème !
client 2