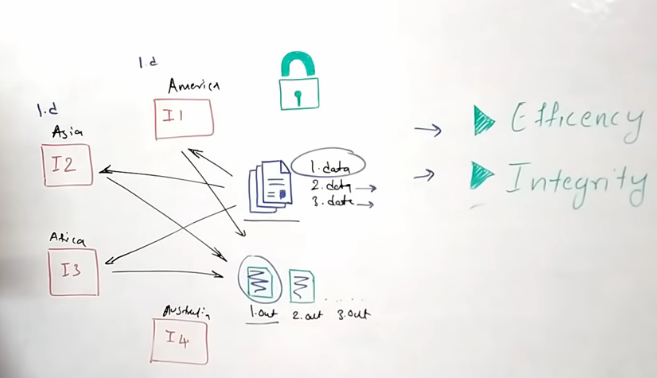
# Distributed Locks

In distributed system we need distributed locks when multiple instances started working on the same file.

## Why need distributed locks



When multiple machines do processing on file we must need.

* Efficiency
* Integrity : The state of data should not be corrupted

Ideally one machine should process one file, but what if the file is very huge and needs multiple machine to process. But during that state as both the instances are working on same time there could be huge change the data wrote to out file would be incorrect or corrupt.

The ideal solution from here to allow multiple instance processing on same file is to **lock** the file when one instance is processing.

Here come the **major problem.** This is not one machine where lock will reside but rather lock should be distributed across all the instances running across the world.

## Solutions

### Approach one with DB solving

What we can do is maintain a table in DB (or a cache system like Redis) and if one of the instance picked a file maintain a record tracking that instance. So when other instance trying to pick same or other file, first it checks if any of the instance have a running record on it and based on that decision is taken.

Here the problems come out is we are introducing a **single point failure.** Any time that DB node failed which could corrupt a whole day long processing computation time.

### Modified approach one

We can build the DB or Redis system distributed. Like a Master and Slave so when data is wrote to Master it will **sync** the data (**Asynchronously**) to all the Slave nodes.

Here if the DB nodes are present at various geographical region, then the sync will also take some time and at a particular state there could be **No single source of truth!!** When the Master data is not synced to other nodes.

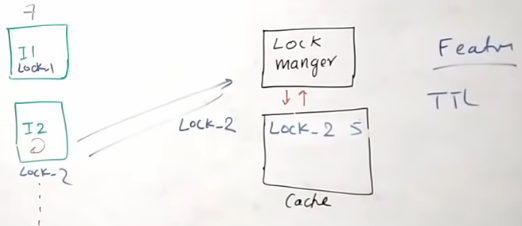
So we needs to design our system which answers both **Single point of failure** and **No Single source of truth.**

### Properties of lock

* Mutual Exclusive (MUTEX) – No two threads or processes should have the same file reading at same time.
* Dead Lock Free
* Fault Tolerant

### TTL making system fault tolerant

Here we need a **Lock manager** design to make the system dead-lock free and fault tolerant.



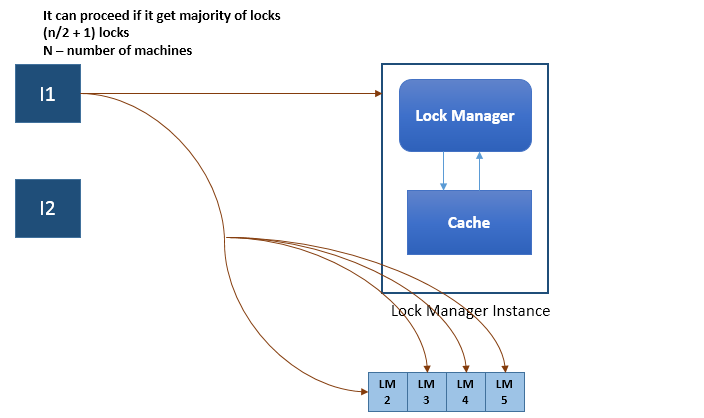
The communication will happen through Lock manager where it will create a lock with **unique lock id** and with **Time out.** Then propagate this information to underlying cache.

We need **Time out** to make the system dead-lock free and need **unique lock id** to ensure no other instance is releasing the lock created by some other instance.

### Solving single point of failure

Here we need to distribute the **LMI (**Lock Manager Instances). And one Instance now will try to get lock from **all the LMIs.** And once it get majority of locks (N/2 + 1) then only that instance can proceed to operate on that file.

Here **N** is the number of active instances of LMI. (N/2 + 1) is the majority of locks from all active LMIs.



## Libraries available

There are so many libraries available, from Redis we have **Red Lock**