# HW2 Wet - Blockchain with Adaptive Batching

## Introduction

Blockchain is a distributed ledger in which every block contains a list of transactions as well as the hash of its predecessor block.
Blockchain can be defined as either unpermissioned or permissioned. In the unpermissioned model, the parties are not pre-known and cannot be trusted under any circumstances. Such a model exists, for example, in Bitcoin. Contrary, in the permissioned model the parties are pre-known and might be trusted if the system's settings enable it, e.g., in a private, isolated system.
Permissioned Blockchains can be used to solve various problems (see here an example), enriching the system with scalable, robust and available ledger services.
A correct implementation of Blockchain must guarantee the following properties:

- ***Validity***: every block in the chain must satisfy a predefined VALID method.
- ***Agreement***: every prefix of the chain is agreed by all the nodes who hold that prefix.
- ***Termination***: If a valid block is repeatedly being proposed, eventually every correct node will add it to its chain.

A common design pattern for a Blockchain algorithm is as follows:

```
while(true)
    b <- new_block
    atomic_braodacast(b)

upon atomic_deliver a valid block b':
    add b' to chain
```

(Recall that *atomic_broadcast* delivers the messages in the same order at all nodes). It is easy to see that the above pattern preserves validity, agreement and termination of Blockchain.
Alas, as simple as it is, this pattern is not very efficient because the *atomic_broadcast* primitive encapsulates a consensus mechanism to ensure the total ordering. Hence, every block contains a batch of transactions (rather than a single transaction). The batch size imposes a trade-off between scalability and latency; the larger block is, the cost of atomic_broadcast is amortized among the batch's transactions. On the other hand, bigger blocks increase the latency because of (i) the network latency and (ii) the time that is needed to fill the block.
Therefore, setting the batch size requires careful handling.
One way to optimize the above trade-off is by dynamically adjust the batch size with respect to some parameters (e.g the server's load). This method is called an adaptive batching.

# Your mission

Following the increasing popularity of the Blockchain technology, the Fed (Central Bank of America) asked you to develop a Blockchain platform that has an adaptive batching feature.

# Detailed Description

Implement a permissioned benign Blockchain platform with an adaptive batching feature. Analyze its performance, strengths and weaknesses and write a report to the FED in which you present your work and explain why they should choose your solution and how it will benefit their finance tech.

## Environment Assumptions:

1. There are ***n*** pre-known nodes. New nodes are not allowed to join after the system was started.
2. There may exist up to ***f*** < n/2 faulty nodes. The failure pattern is benign, meaning that a node may crash (but not to behave maliciously). A crashed node stops sending messages and does not recover while the current running.
3. You can assume that initially all nodes are alive.
4. The system is asynchronous but you are allowed to use any type of failure detector (as long you implement it correctly)
5. Since this is a permissioned Blockchain, you can assume a small set of servers replicating the blockchain on behalf of the clients.
6. There is no need for signing nor encrypting messages. No need for a Byzantine consensus protocol.
7. Instead of creating a real hash in the blocks' header you can simply index them (according to their location in the chain)
8. For simplicity, you can assume a single dedicated contact server per client - this means that each client knows the address of a server in advance, so there is no need to maintain a naming mechanism.
9. You can define your VALID method, but it should be nontrivial (it has to return true for at least some nonempty set of different transactions).
10. You are allowed to implement any type of Blockchain algorithm (leader based, leader free, FD based and so on) as long it is following the above three properties.

## General Guidelines:

1. You should develop your system using Java (of any version you wish). If you want to use another programming language contact the teaching assistants (Yehonatan) for approval.
2. Use the techniques and principles you have learned in the lectures.

3. Use the frameworks you have learned in the tutorial:
    a. Communication between servers should be done by JAVA-RMI, gRPC or REST.
    b. Define your REST-API for client-server communication.
    c. Use ZooKeeper for implementing FDs, atomic broadcast, membership service and other distributed abstractions.
        i. Hint: For atomic broadcast, recall that Znode holds only a small amount of data. How can you leverage this fact to disseminate big messages?
4. Define your system inputs\outputs' formats as you see fit.
5. Assume the syntax is correct – no need for fancy parsers and checking for input errors.
6. No need for fancy UI (although it would be nice).
7. No need to save results after the system was shut down (no need for DB services).
8. You can test your system locally on your machine:
    a. Using different threads.
    b. VM's
    c. Dockers (Using dockers correctly will grant up to 10 bonus points)

    No need to work with AWS, Azure and similar platforms.
9. You have much freedom with designing, developing, choosing frameworks to work with and presenting your project.

## Grading Guidelines

1. You should submit your project in pairs. If you cannot find a partner contact the course staff for solutions.
2. You should create a detailed external documentation that describes how you solved the exercise and, in particular, explain and argue your design choices.
3. Each team will present their project to the course staff in a live demo (dates to be assigned later on).
4. The system performance and design, as well as creativity and innovation, will play a major role in the grading consideration.

## Submission

Due date 3/1/19

You should submit hw2.zip containing:

1. project.pdf which contains:
    a. Documentation, analysis and justifications of your implementation.
2. src.zip which contain:
    a. All your system's source code.

Have a pleasant journey