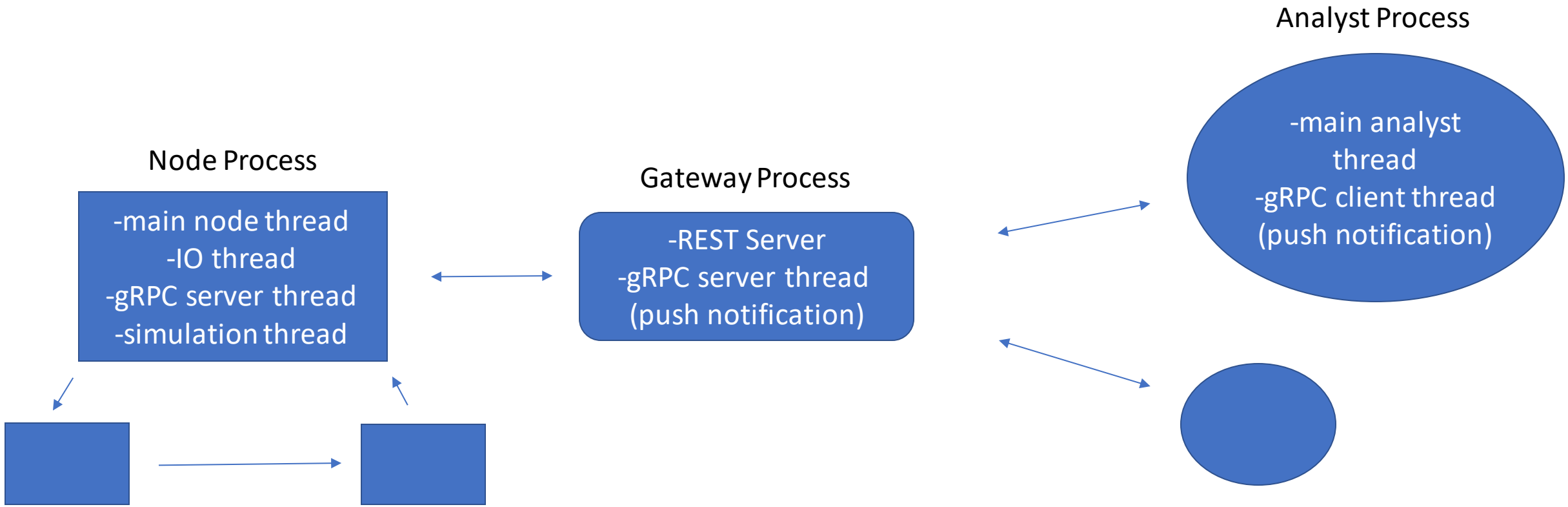
Project work for the exam Dystributed and Pervasive Systems

University of Milan, Master of Science in Computer Science, 2019/2020

Nicola Amadio

High Level Project Architecture



Gateway

- **REST API: CRUD operations on Gateway's main data structures:**

1. Nodes: list of nodes registered to the Gateway.
2. Stats: list of simulation samples registered to the Gateway.

- Consistency is mainteined by synchronizing CRUD operations.
- Blocking time is reduced by having two separate locks for the two data structures.

- **GRPC Server handling push notification:**
- RPC function 'notify' receives new statistics and broadcasting them to all the registered Analyst Clients.
- 'Publisher' singleton, calls 'notify' every time the REST server signals that a new statistic has arrived.
- 'notify' defines a bi-directional stream of NotificationMessage protos such that on one end of the stream Publisher adds new stats by calling onNext on 'notify', and on the other end the Client Analyst implements the onNext and elaborates the messages broadcasted by 'notify'.

Analyst

- Provides a **UI to the users** of the network, offering a menu with options for getting information about the network status and the statistics gathered since the system is on.
- Has a **thread** connected to the gRPC push notification server of the Gateway, implementing the onNext and **printing out notifications in the console**.

Node, aka p2p network

- **Threads:**

1. Main node thread: main function in NodeProcess class.
2. IO thread: used to insert 'q' and let the node exit the network.
3. Simulation thread: external package which automatically generates statistics.
4. Grpc Server: powering P2PNetServiceImpl RPC functions.

- **Helper classes:**

1. NodeWrapper: core functions related to the node activity; from **creating the token** (more on this later) to be passed to the network (in case it's the first node) to pulling the **statistics to put into the token**, from **broadcasting a message to the network when entering/exiting** it to signaling the gateway about its exit.
2. Mybuffer: implements sliding window functionality into a cache storing samples data.
3. NodesNetwork: stores a TreeMap with an ordered set of nodes representing the network architecture and handles CRUD operation on it providing consistency through synchronization of its methods.

- **Other:**

1. Token: lock object used to synchronize the passage of the token to the node's exit.
2. Exiting: additional object used to synchronize the node's exit when it's the only node left in the network.
3. P2pNetServiceImpl: implementation of node's gRPC methods: one to forward the token, and two to update its nodesNetwork TreeMap, i.e. its local copy of the network architecture.

How does this p2p network work?

- It's a token ring: a circle of nodes and a token; the token is held by only one node at a time and each node updates its content by adding the statistics that are being accumulated by its sampling process, then it forwards it to the next node.
- passToken RPC function: receives token from previous node, updates it, checks in its local copy of the architecture what is the next node, grabs its port and ip address and then forwards the token by calling passToken on the next node; it is built in a way such that, together with the rest of the architecture and its data structures, guarantees a high level of efficiency and resilience to edge cases.

Edge Cases

- **Node exits when it has the token:**

1. This is avoided by **synchronizing the shutDown()** method (which terminates the gRPC server and its RPC) **to the passToken function**, with the help of an empty class we called **'Token'**, to which both the close() method in GrpcServer class and the passToken synchronize on.
2. Additionally, the **Exiting class** provides and **additional gate** which allowes close() to shutdown the node **if it is the only node left in the network and has requested to exit**.

- **Simultaneous entrance/exit of different nodes to the network:**

1. The RPC functions that updates each node's architecture with new nodes entering/exiting it live in the multithreaded environment of a gRPC server, which means they could be called by different threads at the same time, which could cause consistency issues. This is avoided by **relying on the NodesNetwork class and the synchronization of the CRUD operations on it**.
2. PassToken is not 100% synchronized with the updates on its local copy of the architecture, which means it could end up sending an RPC to a node which is no longer connected, thus causing the token's loss. This is avoided by handling the result of a passToken to a next node, in the passToken of the present node, **looping** (checking in my local copy of the architecture which one is the next node, and calling a passToken on its gRPC server) **until I receive confirmation of my call's success**.

Thank you!