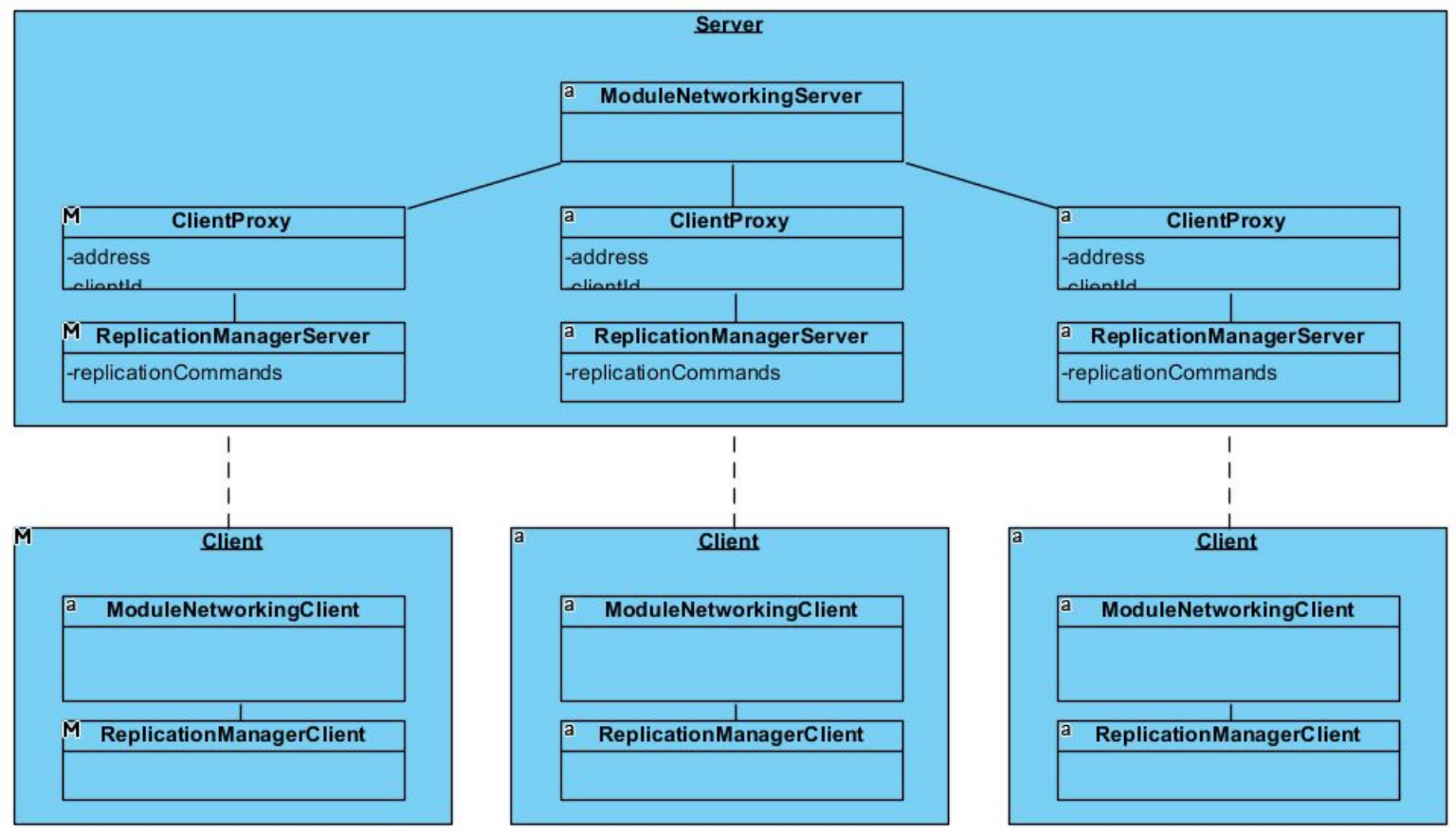
Multiplayer Game in C++

World state replication

Networks and Online Games

World state replication



World state replication: Server side

```
enum class ReplicationAction
{ None, Create, Update, Destroy };

struct ReplicationCommand
{
    ReplicationAction action;
    uint32 networkId;
};
```

```
class ReplicationManagerServer
{
public:

    void create(uint32 networkId);
    void update(uint32 networkId);
    void destroy(uint32 networkId);

    void write(OutputMemoryStream &packet);

    // More members...
};
```

World state replication: Server side

```
enum class ReplicationAction
{ None, Create, Update, Destroy };

struct ReplicationCommand
{
    ReplicationAction action;
    uint32 networkId;
};
```

```
class ReplicationManagerServer
{
public:

    void create(uint32 networkId);
    void update(uint32 networkId);
    void destroy(uint32 networkId);

    void write(OutputMemoryStream &packet);

    // More members...
};
```

networkId	0	1	2	3	4	5	6	7
action	**None**	**Update**	**None**	**Update**	**Destroy**	**None**	**Create**	**Create**

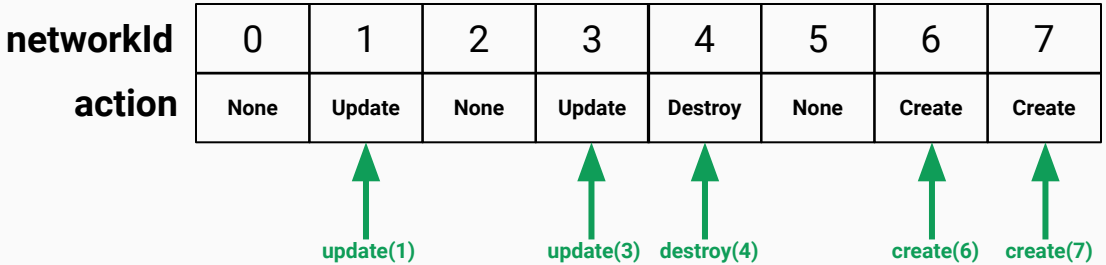
Should contain some kind of data structure that allows to map a *networkId* to a *ReplicationCommand*.

- std::vector
- std::unordered_map
- plain C array
- ...

World state replication: Server side

```
enum class ReplicationAction
{ None, Create, Update, Destroy };

struct ReplicationCommand
{
    ReplicationAction action;
    uint32 networkId;
};
```

```
class ReplicationManagerServer
{
public:

    void create(uint32 networkId);
    void update(uint32 networkId);
    void destroy(uint32 networkId);

    void write(OutputMemoryStream &packet);

    // More members...
};
```



Insert replication commands

- From ModuleNetworkingServer
  - To spawn objects
- From Gameplay systems
  - To update objects
  - To destroy objects

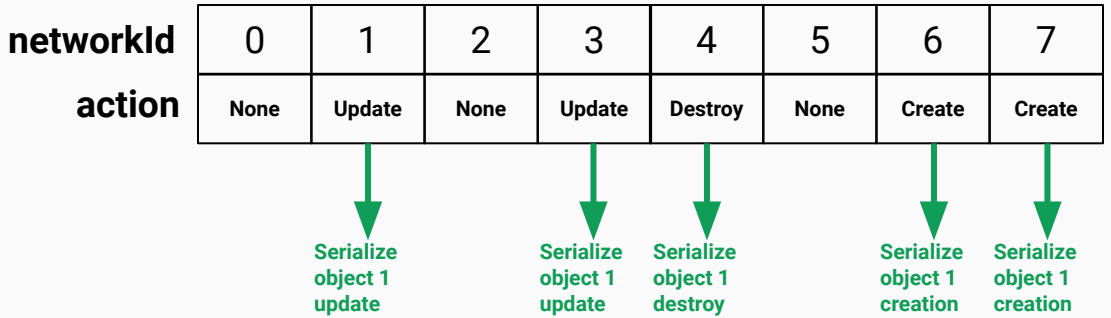
Mainly event notifications that can happen at any time:

- A new player joined
- A network game object changed its position
- A laser exceeded its lifetime and must disappear

World state replication: Server side

```
enum class ReplicationAction
{ None, Create, Update, Destroy };

struct ReplicationCommand
{
    ReplicationAction action;
    uint32 networkId;
};
```

```
class ReplicationManagerServer
{
public:

    void create(uint32 networkId);
    void update(uint32 networkId);
    void destroy(uint32 networkId);

    void write(OutputMemoryStream &packet);

    // More members...
};
```



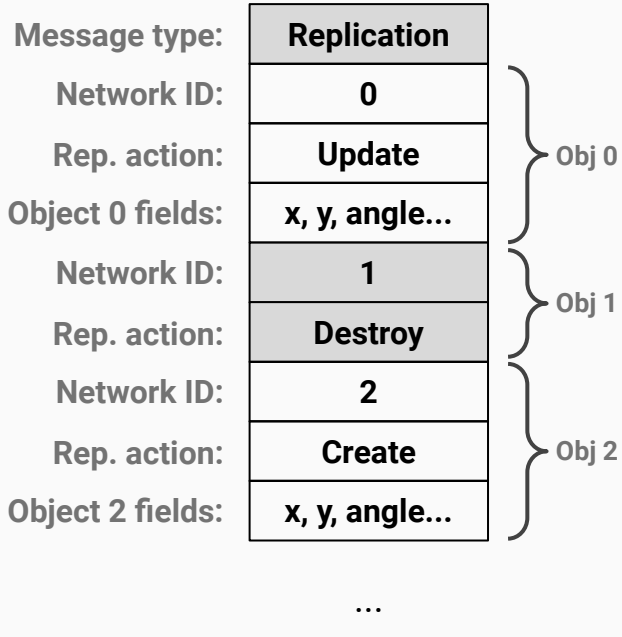
Write replication packet

- From ModuleNetworkingServer
  - In onUpdate()
  - At regular intervals
    - Could be done at each frame, but that consumes bandwidth...
  - Then send packet to client

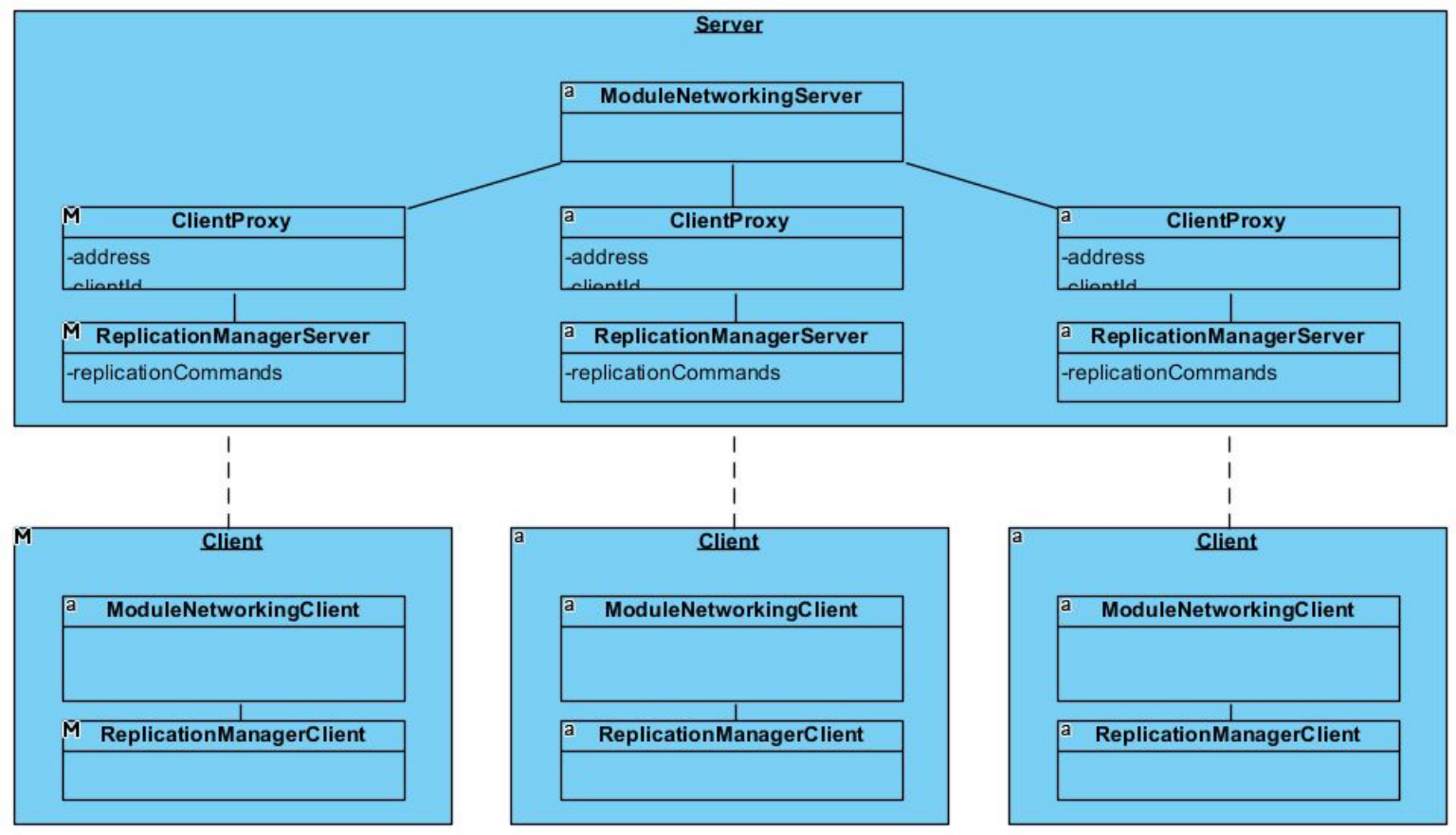
World state replication: Server side

ReplicationManagerServer::Write() - For all replication commands

- Write the *networkId*
- Write the *replicationAction*
- If *replicationAction* is Create
  - Get the object from linking context
  - Serialize its fields
- Else if *replicationAction* is Update
  - Get the object from linking context
  - Serialize its fields
- Else if *replicationAction* is Destroy
  - Nothing else to do
- Clear/remove the replication command
  - With this we are assuming reliability...



World state replication



World state replication: Client side

```
class ReplicationManagerClient
{
public:

    void read(const InputMemoryStream &packet);
};
```

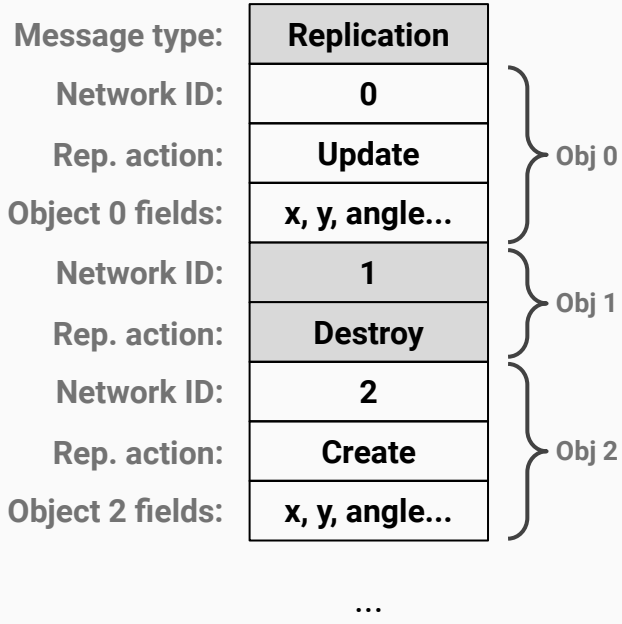
Read replication packet

- From ModuleNetworkingClient
  - In onPacketReceived()
- The method read() itself performs the replication commands in the client

World state replication: Client side

ReplicationManagerClient::Read() - While packet is not empty...

- Read the *networkId*
- Read the *replicationAction*
- If *replicationAction* is Create
  - Instantiate new object
  - Register it into the linking context
  - Deserialize its fields
- Else if *replicationAction* is Update
  - Get the object from the linking context
  - Deserialize its fields
- Else if *replicationAction* is Destroy
  - Get the object from the linking context
  - Unregister it from the linking context
  - Destroy it



TODOs

Implement the class ReplicationManagerClient

Implement the class ReplicationManagerServer

In ModuleNetworkingClient

- Declare a member of type ReplicationManagerClient and use it when receiving Replication packets

In ModuleNetworkingServer

- For each ClientProxy, declare a member of type ReplicationManagerServer, and use it where needed