Protocol Translator Server (PTS)

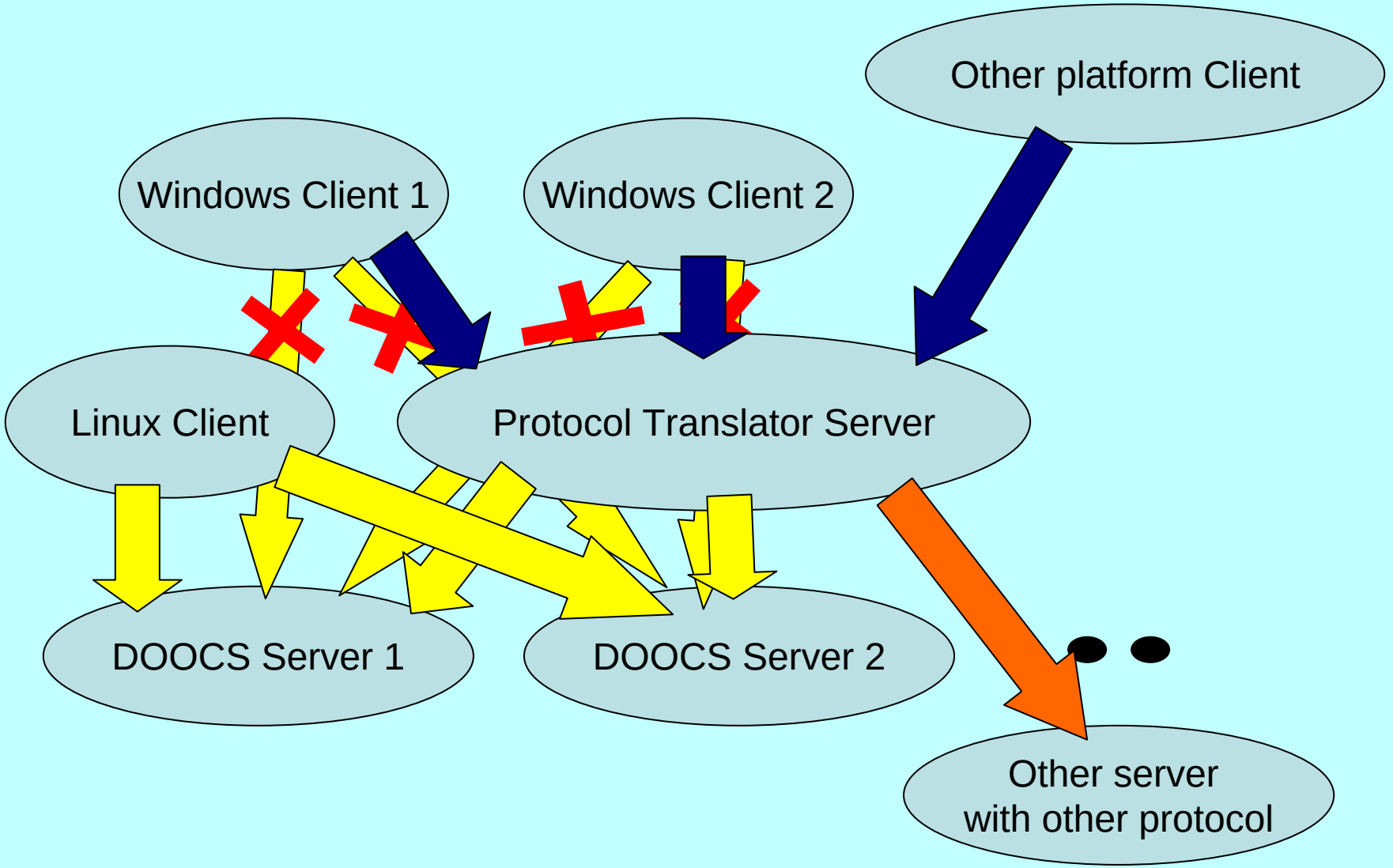
23 February 2012

Davit Kalantaryan

Content

1. Introduction (scheme of system)
2. Realization
3. C interface functions
4. C++ classes
5. Possible use cases
6. Why this server has been created
7. Examples
8. Outlook

Introduction (scheme of system)



Realization

Independent thread in server side for each client

Every client that connects to PTS, has possibility to ask the server to keep socket connected and create individual thread for serving only this client. This fact makes very fast client talking (communication) with server. Only initial connection will make a very small delay.

Possibility send string code instead of string

There is a container in server side that keeps all long strings those clients use during all of operation time of server. When server receives from client any string, then after completion the client required operation, server looks in container, and if this string doesn’t exist yet, then server adds this string to container. After this, it is possible to send to server only the index (in 2 bytes: short int) of this string and server will find the string from container by this index.
This again will save the time because of networking.

Example of using server container

Example is again connected to DOOCS protocol. If any application needs many times read or write data from/to same DOOCS address, then it is not necessary for client each time to send to PTS server same DOOCS address. Client can only once send address to PTS then receive a code for this DOOCS address. After client has this 2 byte code, client can send only the code to server and code is enough for server for knowing what address client wants to manipulate.
Due to networking this fact will save time.

Realization

Connection threads

To serve the clients in faster way the server (PTS) creates individual threads for each client.
Does this solutuion take too much resources???

Following are the threads in server
1.Main thread which is in idle mod.
2.Connection threads (10) which are in suspended state and will be activated only in case there is a connection from a new client.
3.CreateOrRemove thread which is creating/removing individual threads for clients.
4.Talk threads with number of clients (ex. 4 clients → 4 threads)

Function for connection threads

```
/*
 * 'c' - Create new gate ( Socket )
 * 'd' - Direct use
 * 'r' -  Ready
 */
void CDOOCS_Server::InfoReceiver( CSocketDOOCS* a_pSocket, char* a_pcBuffer, int a_nBufLen, void* a_pReserved )
{
        CTaskItem* pTaskItem = (CTaskItem*)a_pReserved;
        a_pSocket->SendData( "r", 1 );
        a_pSocket->RecvDataTm( a_pcBuffer, 1 );

        switch( a_pcBuffer[0] )
        {
        case 'c':
                m_MessageItem = a_pSocket ;
                m_cMessageForCreateOrRemove.SetValue( 'c' );
                break;
        case 'd':
                pTaskItem->SetSocket( a_pSocket );
                pTaskItem->ItemThreadFunc();
                pTaskItem->SetSocket( 0 );
                delete a_pSocket ;
                break;
        default:
                if( a_nBufLen == 0 ){ a_pSocket->SendData( FAILURE ); }
                delete a_pSocket ;
                break;
        }
```

Function for these threads is checking the connected socket, and if received ‘c’, telling to CreateOrRemove thread to create new gate for this new client. In the case of ‘d’ – directly doing job.

Function for CreateOrRemove thread

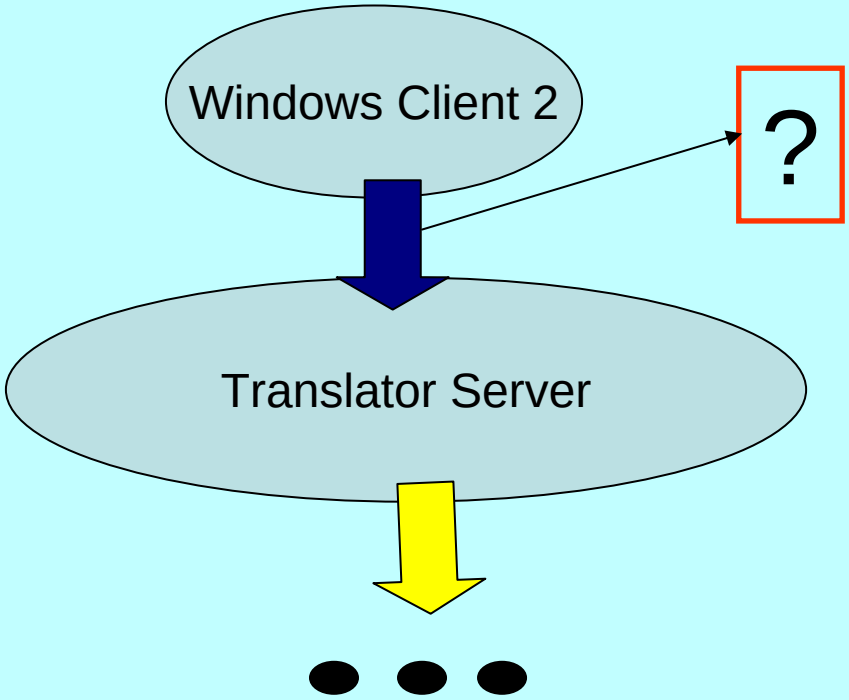
Thread CreateOrRemove doesn’t take measurable resource from CPU. This function receives message from Connection thread, creates a new gate for new client and runs independent thread (Talk thread) for this new user. This thread will also remove gates (or client threads) by receiving messages from Talk thread.

```
/*
 * 'c' - Create New Item ( or open new gate for client )
 * 'e' - Exit
 * 'i' - Idle state ( not do anything )
 * 'r' - Remove Gate
 */
void* CDOOCS_Server::CreateOrRemoveThread( void* a_pArg )
{
        /**** Declaration Place *******/
        HashTblDv* pTasksItems = (HashTblDv*)a_pArg;
        CTaskItem* pTaskItem;
        bool bIsWork( true );
        CSocketDOOCS* pSocket;
        size_t i, unSize;
        /*******************************/
        while( bIsWork )
        {
                switch( m_cMessageForCreateOrRemove.GetValue() )
                {
                case 'c': /* Create New Item */
                        pTaskItem = new CTaskItem( m_MessageItem );
                        pSocket = m_MessageItem.pSocket.GetValue();
                        pTasksItems->AddElement( pTaskItem, &pSocket, sizeof( void* ) );

                        pTaskItem->RunThread( ItemThreadFunc, pTaskItem );
                        m_cMessageForCreateOrRemove.SetValue( 'i' );
                        g_Log.Write( "Function CDOOCS_Server::CreateOrRemoveThread" );
                        g_Log.Write( "case 'c': - Create" );
                        break;
                        …
                }

                SLEEPM( 100 );
        }
        return 0;
}
```

Communication function (Talk thread)



Talk threads

This thread is always in suspended state. It will be activated only by corresponding client, then will do the job that client required, and will go to suspended state again.

```
/*
 *
 * 'a' - Address New
 * 'b' - Break (End)
 * 'c' - GetInt
 * 'd' - SetInt
 * 'e' - Error Get
 * 'f' - GetString
 * 'g' - GetFloat
 * 's' - SetFloat
 *
 * 'f' - Failed to receive
 *
 */
```

```
...

nRecieved = pSocket->RecvDataInf( m_pcBuffer, 1 );

switch( m_pcBuffer[0] )
{/* switch */

                case 'a':
                                pSocket->RecvValue( bIsSucceed, nIncBufLen );

...
```

C interface functions

For using described server static and dynamic libraries created for 32 bit and 64 bit windows systems. The names of libs are following:

libDOOCS_Funcs_FNLA32.dll
libDOOCS_Funcs_FNLA64.dll

libDOOCS_Funcs_FNLA32_St.lib
libDOOCS_Funcs_FNLA64_St.lib

The names of functions are following:

```
bool InitWinDOOCSLibNew( const char* Server_IP );

void CleanWinDOOCSLib();

void* CreateNewGate( int TimeOut );

void RemoveGate( void* pGate );

bool AddressIndex( void* pGate, const char* Address, short*const AddressIndex );

int GetDOOCSError( void* Gate );
```

C interface functions

```
float GetDOOCSFloatFast2( void* pGate, short AddressIndex );

float GetDOOCSFloatFast( void* Gate, const char* Address );

void SetDOOCSFloatFast2( void* Gate, short AddressIndex, float Value );

void SetDOOCSFloatFast( void* Gate, const char* Address, float Value );

int GetDOOCSIntFast2( void* Gate, short AddressIndex );

int GetDOOCSIntFast( void* Gate, const char* Address );

void SetDOOCSIntFast2( void* Gate, short AddressIndex, int Value );

void SetDOOCSIntFast( void* Gate, const char* Address, int Value );

char* GetDOOCSStringFast2( void* Gate, short AddressIndex, char* String, int StrLen );

char* GetDOOCSStringFast( void* Gate, const char* Address, char* String, int StrLen );

void SetDOOCSStringFast( void* Gate, short AddressIndex, char* String );

void SetDOOCSStringFast( void* Gate, const char* Address, char* String );
```

C++ classes

The names of the libraries are following:

libDOOCS_CLASSES32.dll
libDOOCS_CLASSES64.dll

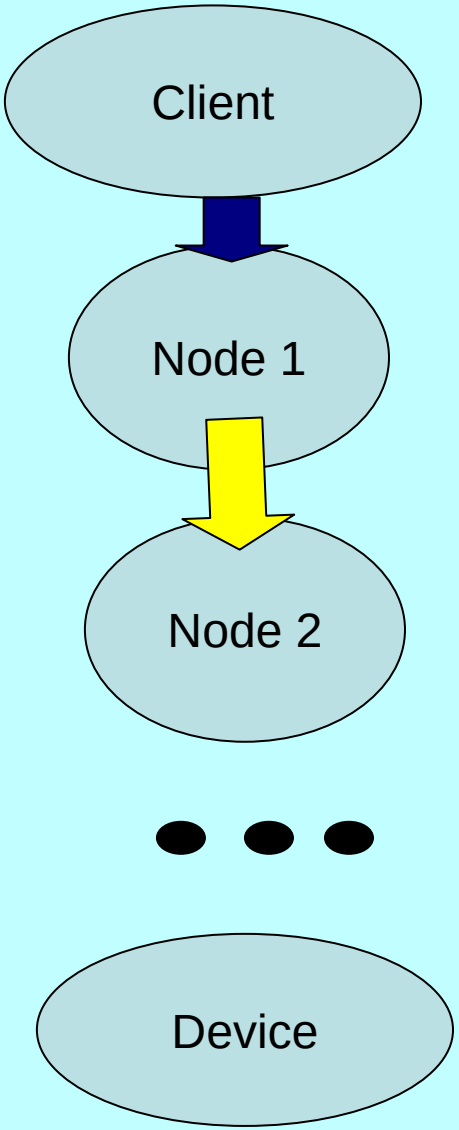
libDOOCS_CLASSES32_St.lib
libDOOCS_CLASSES64_St.lib

There are several classes in these libs. The class that allows to use this server is "class Not_Lin_DOOCS_Funcs". There is also a class which uses direct connection to DOOCS servers in linux and TS server in windows
"class CDOOCS_Funcs"

```
class CDOOCS_Funcs :
#ifdef LINUX
        public Lin_DOOCS_Funcs
#else
        public Not_Lin_DOOCS_Funcs
#endif
{

        ...
};
```

Main disadvantage of this



The more nodes between the client and the device → the more difficulties due to

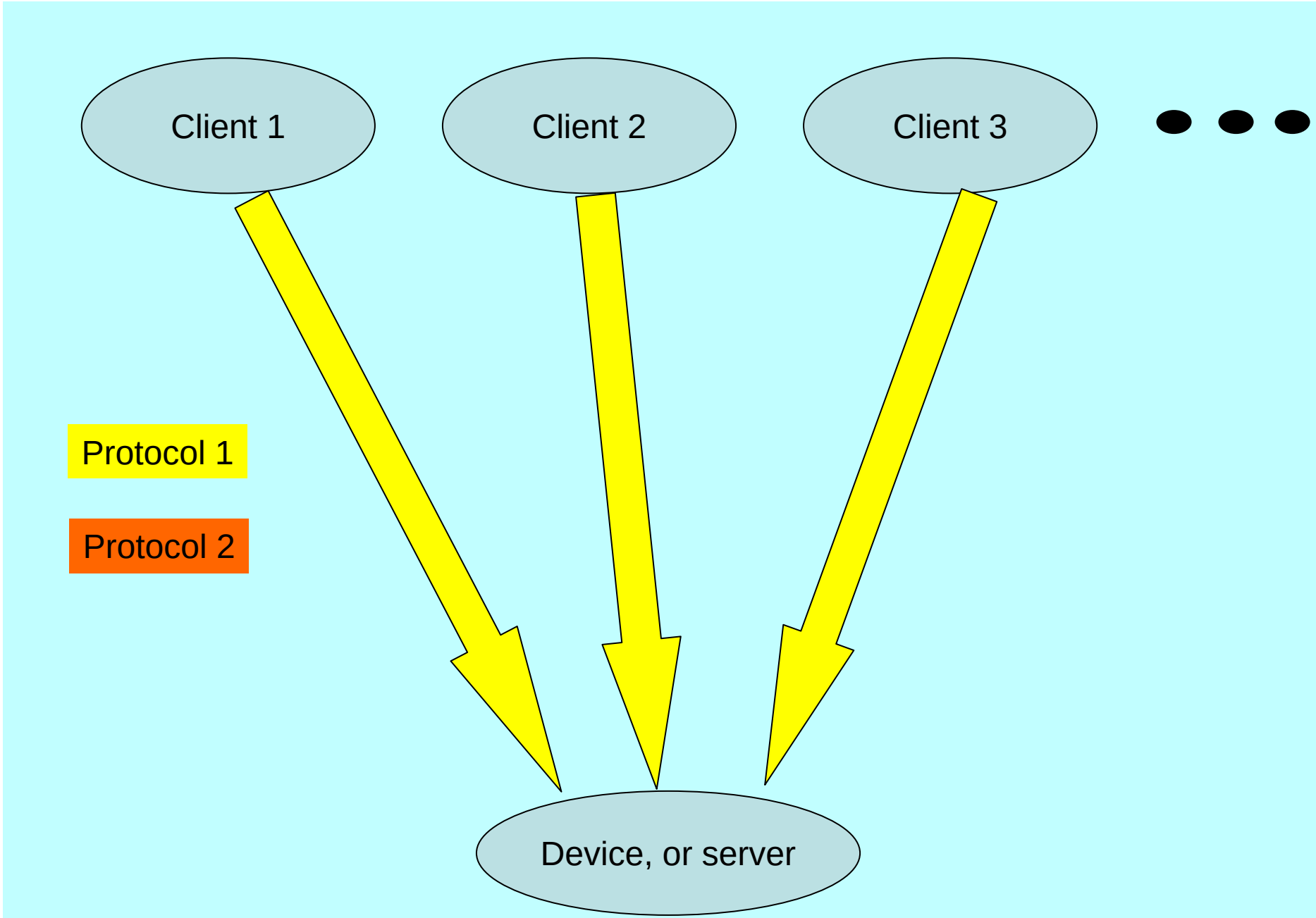
- Possible errors inside the nodes
- Networking between the nodes

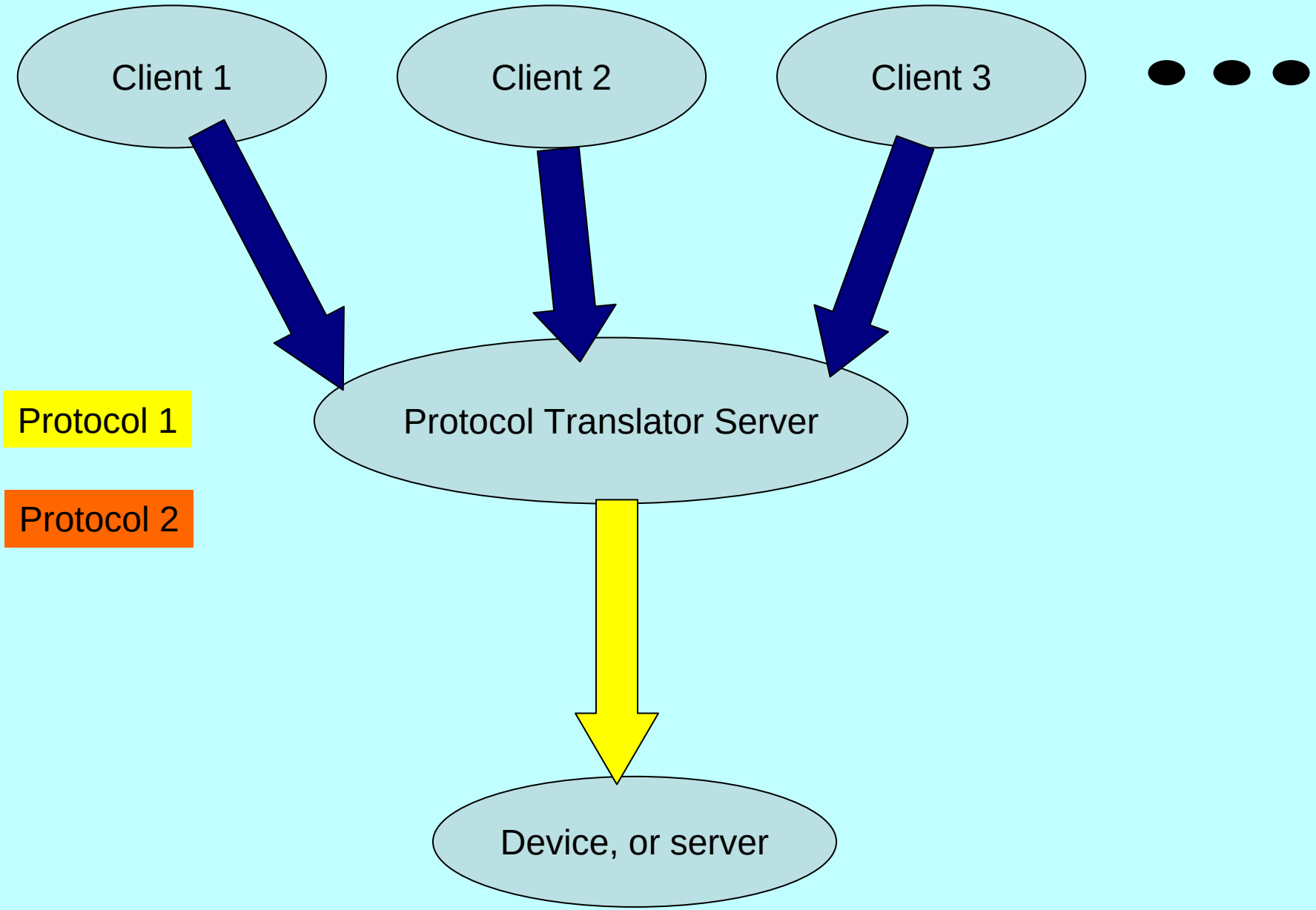
Possible use cases

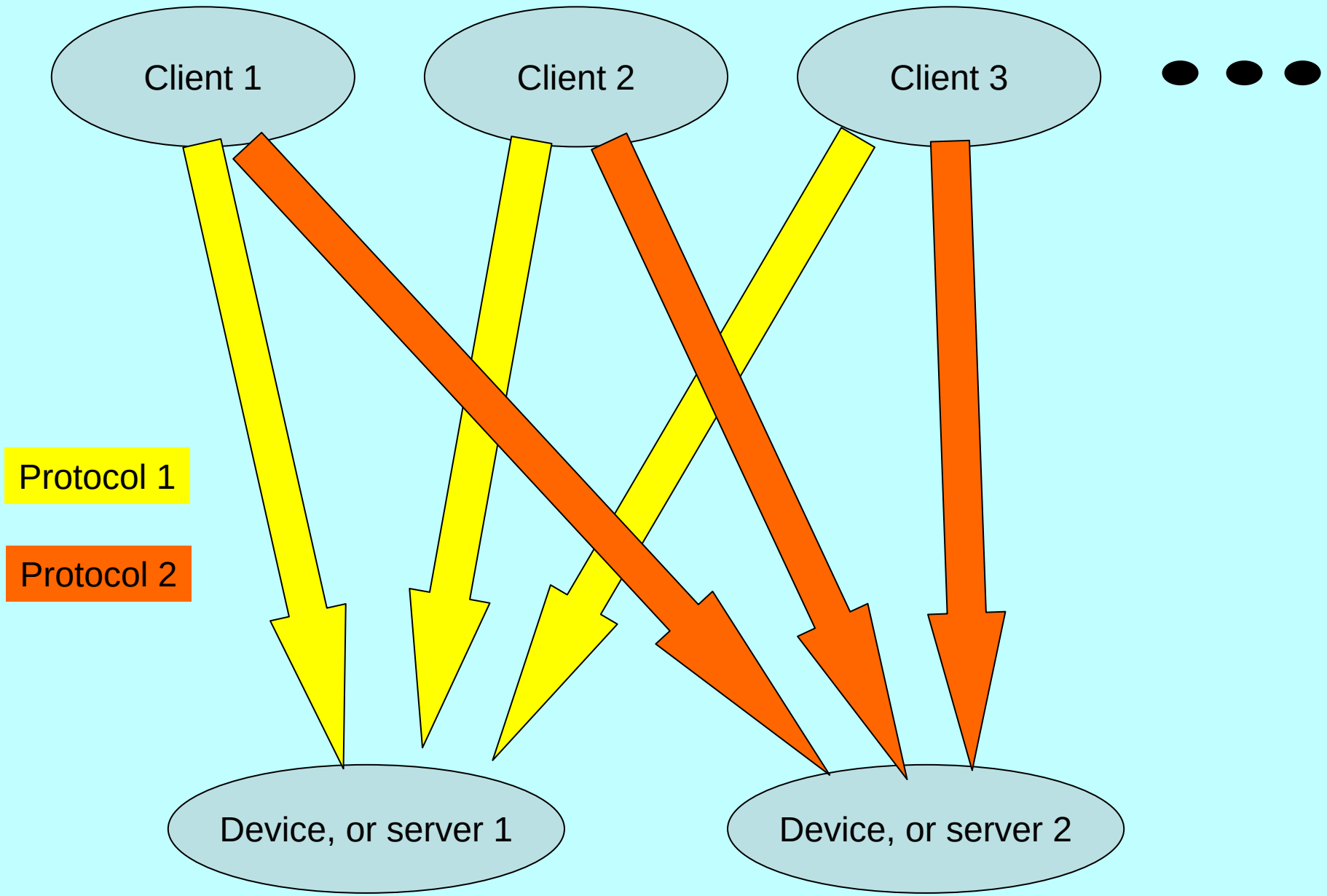
I think in some cases this server (or servers like this which translates unknown protocol for some operation systems) can be useful.

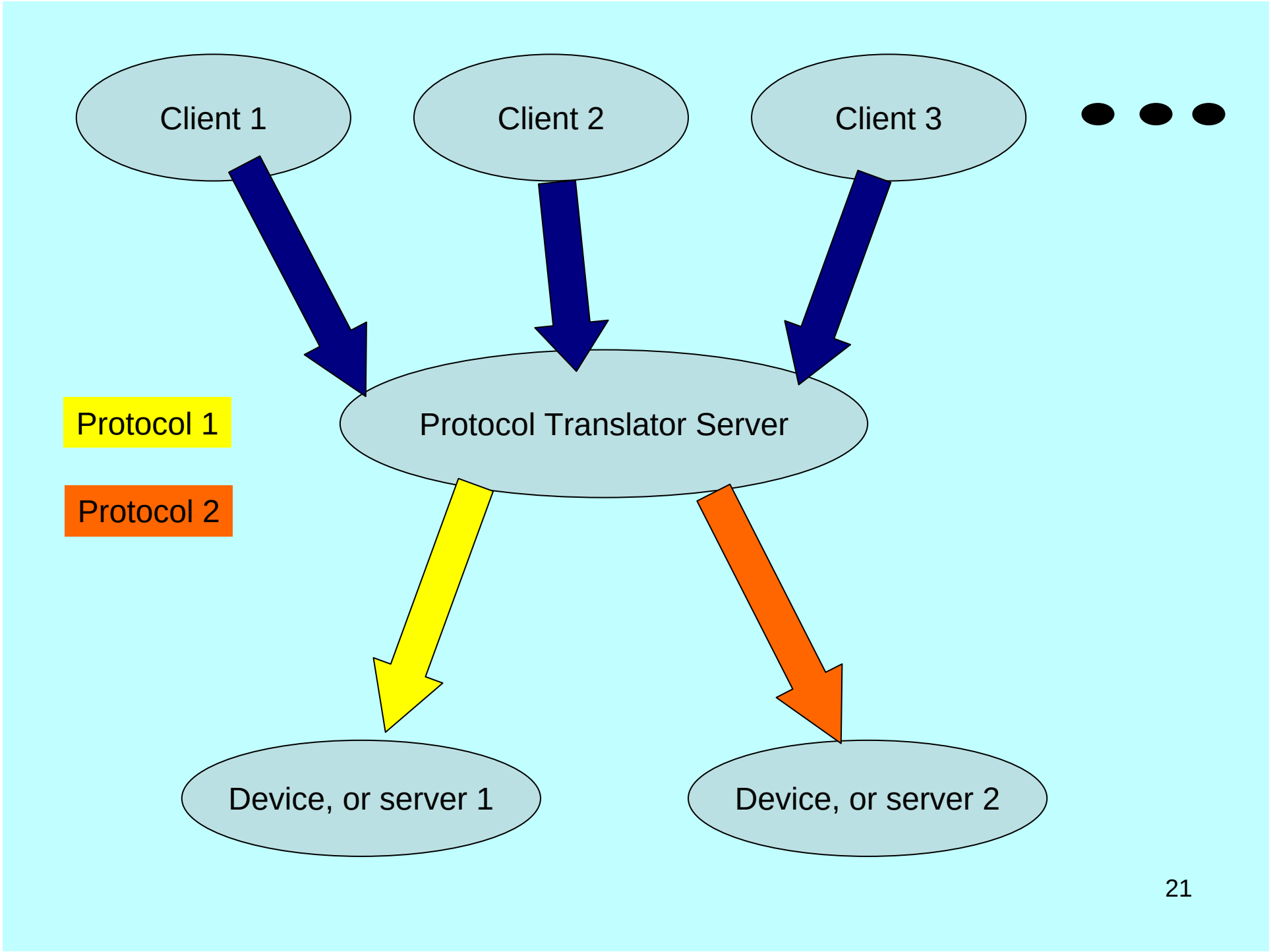
- Some physicists write their own codes in C or in C++. If they need to create program that works with DOOCS servers in windows environment, then PTS server can be helpful.

- PTS server can be temporal solution if there is unknown protocol for any operation system and very urgently some project must be created which uses this protocol.


Client 1
Client 2
Client 3
Protocol 1
Protocol 2
Device, or server


Client 1
Client 2
Client 3
Protocol 1
Protocol 2
Protocol Translator Server
Device, or server


Client 1
Client 2
Client 3
Protocol 1
Protocol 2
Device, or server 1
Device, or server 2


Client 1
Client 2
Client 3
Protocol 1
Protocol 2
Protocol Translator Server
Device, or server 1
Device, or server 2

Why this server has been created

There was a problem of creation of booster steering functions for MATLAB, that will read/write data from/to DOOCS servers. I got a suggestion to make a possibility to have all of these functions available also in windows. Some steering functions do many number of iterations that is why I decided to use C++ (for good performance). I created PTS server to temporary solve the problem of communication with DOOCS servers from windows.

In the future I'll change steering functions to avoid this additional node between windows client and devices. Possible solutions advised by colleagues are the following

1. Using TINE protocol from windows

2. Using JAVA DOOCS client lib for windows

Thank you for your attention !