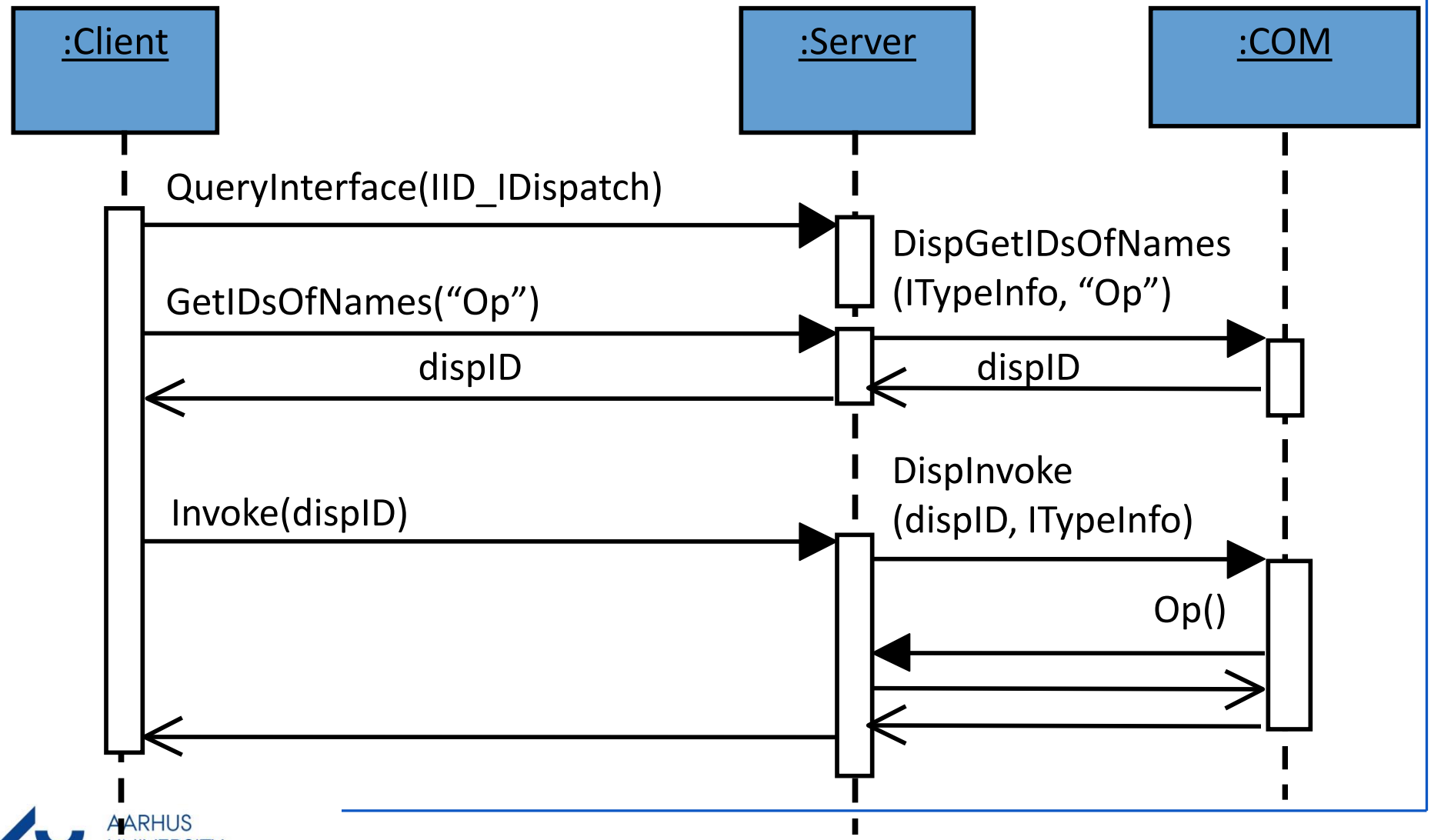
Dynamic Requests in COM

IDispatch

Dynamic Requests in COM

- COM is often used with interpreted scripting languages (e.g. VBScript)
- Interpreters of these languages need to make dynamic requests.
- Dynamic Requests in COM are defined in the **`IDispatch`** interface
- Any COM server that implements **`IDispatch`** can be requested dynamically

Dynamic Request in COM



COM's IDispatch Interface

```
interface IDispatch : IUnknown {
 ...
 HRESULT GetIDsOfNames([in] REFIID riid, //reserved
  [in,size_is(cNames)] LPOLESTR *rgszNames,//method+pars
  [in] UINT cNames, //No of Names
  [in] LCID lcid, //locale ID
  [out,size_is(cNames)] DISPID *rgdispid); //name tokens

 HRESULT Invoke([in] DISPID dispID, //token of op
  [in] REFIID riid, //reserved
  [in] LCID lcid, //locale ID
  [in] unsigned short wFlags, //method, put or get
  [in,out] DISPPARAMS *pDispParams, //logical pars
  [out] VARIANT *pVarResult, //logical result
  [out] EXCEPINFO *pExcepInfo, //IErrorInfo pars
  [out] UINT *puArgErr); //type errors
};
```

Dual Interfaces

- Are accessible both via stubs and via dynamic invocation
- Example: Interface *Player*:

```
[object,dual,uuid(75DA6450-DD0E-00d1-8B59-0089C73915CB]
interface DIPlayer: IDispatch {
  [id(1),propget] HRESULT Name([out] BSTR val);
  [id(2),propget] HRESULT Number([out] short val);
  [id(3)] HRESULT book([in] Date val)
};
```

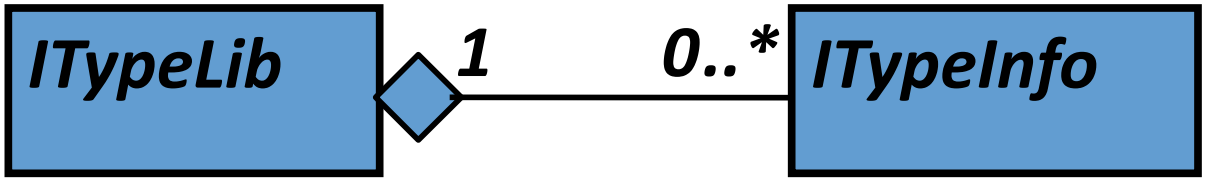
- Interfaces have to be defined as dual!

Transparency of Dynamic Invocation

- Client programs have to be written differently ⇒ Use of dynamic invocation interfaces is not transparent to client programmers
- Interfaces of server objects have to be designed as dual or dispinterface
  ⇒ Use of dynamic invocation not transparent in server design
  - But if you use ATL then it's only one click away

COM Type Library

- COM's provision of run-time type information
- Raw information generated by MIDL compiler
- Stored in tokenized form (.TLB files)
- Main interfaces:



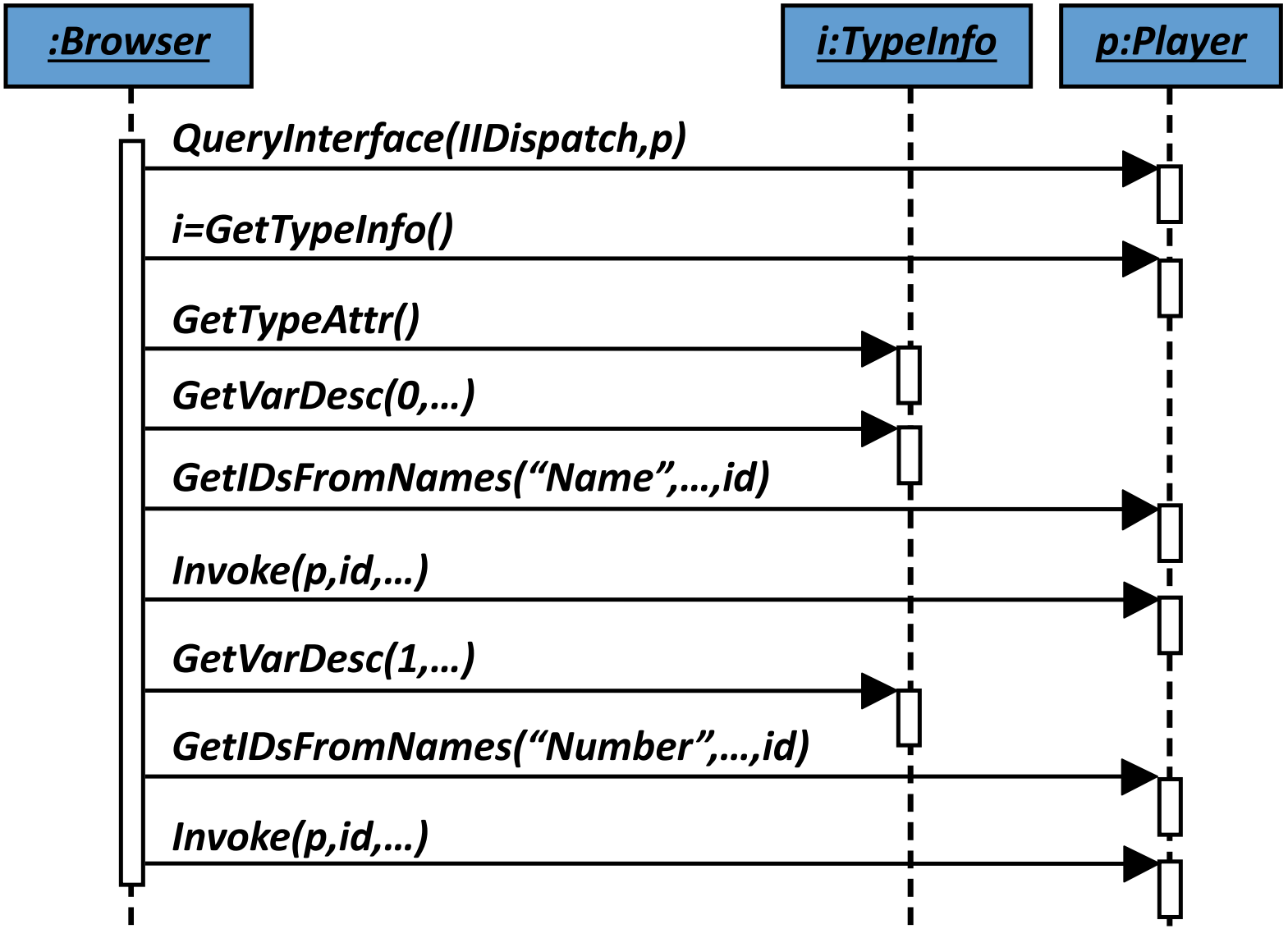
ITypeLib

- Provides operations to browse through all interfaces contained in the type library
  - GetTypeInfoCount (returns number of TypeInfo objects in the library)
  - GetTypeInfo (can be used to obtain type info at a particular index number)
- Locate ITypeInfo objects using the GUIDs

ITypeInfo

```
interface ITypeInfo : IUnknown {
 HRESULT GetFuncDesc( UINT index, FUNCDESC **ppFuncDesc);
 HRESULT GetIDsOfNames( OLECHAR **rgszNames,
                UINT cNames, DISPID *pMemId);
 HRESULT GetNames(DISPID memid, BSTR *rgBstrNames,
                UINT cMaxNames, UINT *pcNames);
 HRESULT GetTypeAttr(TYPEATTR **ppTypeAttr);
 HRESULT GetVarDesc(UINT index, VARDESC **ppVarDesc);
 HRESULT Invoke(VOID *pvInstance, DISPID memid, USHORT
                 wFlags, DISPPARAMS *pDispParams,
                 VARIANT *pVarResult,
                 EXCEPINFO *pExcepInfo, UINT *puArgErr);
...
};
```

Object Browser in COM



Static Invocation

- Advantages:
  - Requests are simple to define.
  - Availability of operations checked by programming language compiler.
  - Requests can be implemented fairly efficiently.
- Disadvantages:
  - Generic applications cannot be build.
  - Recompilation required after operation interface modification.
  - Not accessible from scripting languages

Dynamic Invocation

- Advantages:
  - accessible from scripting languages etc.
  - Components can be built without having the interfaces they use,
  - Higher degree of concurrency through deferred synchronous execution.
  - Components can react to changes of interfaces.
- Disadvantages:
  - Less efficient,
  - More complicated to use from C++
  - Type compatibility checked at run-time not compile-time