CORBA-I

Prof. David Bakken

Cpt. S 464/564 Lecture

Sept 18, 2000

Administrative Items

- Homework #1 is due now
- Handouts today
  - Lecture notes (didn't quite make my 8am promise for them being on web…)
  - Project #1 handout
- Conventions:
  - "PROG" == "Programmer's Guide", by VisiBroker, for C++ or Java
  - "REF" == "Reference Guide", by VisiBroker, for C++ or Java
- Both PROG and REF are available via the class web page or from the VisiBroker vendor, Inprise
- Suggested Reading for Project1
  - PROG Chap 4: Developing an Example Application
  - PROG Chap 5: Handling Exceptions
  - PROG Chap 10: Client-Side Basics (skip the "QoS" stuff)
  - PROG Chap 6: Server-Side Basics
  - PROG Chap 7: Using POAs
    - Just skim – lots of details you don't need to know now, maybe never

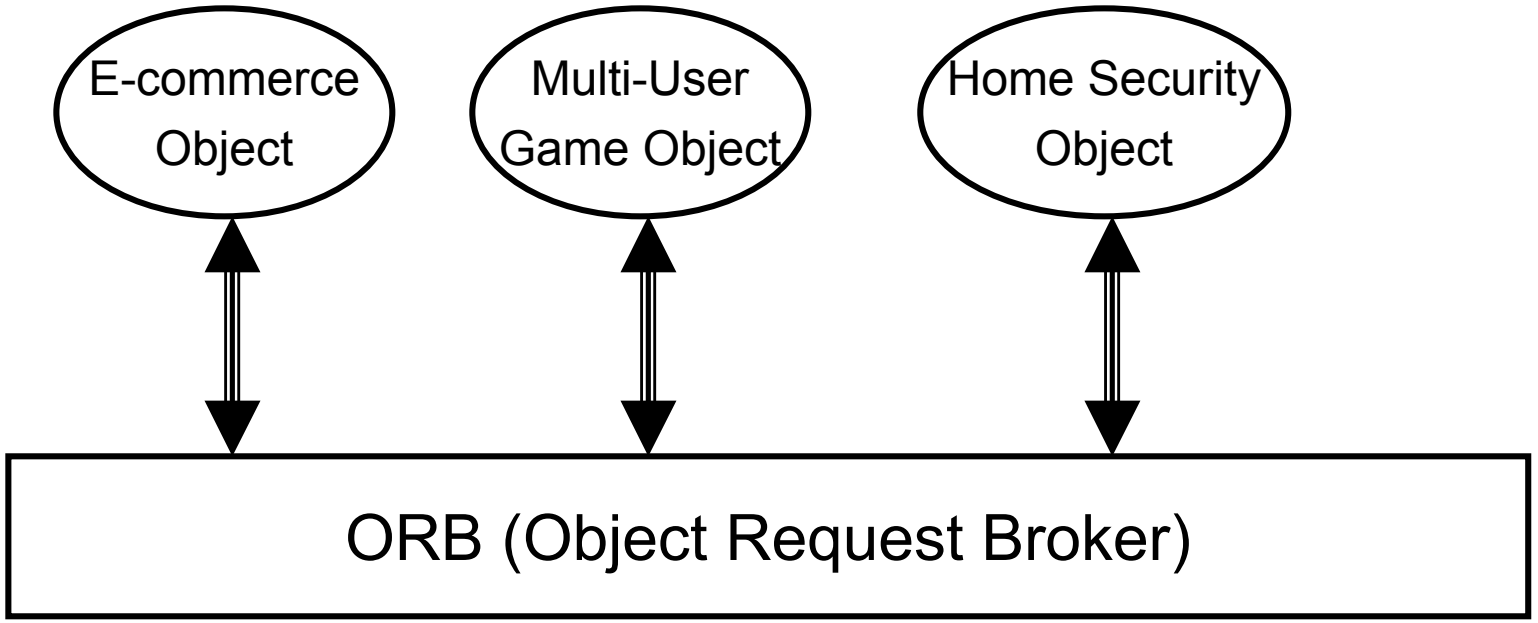
CORBA Features

- Transparencies
  - Programming language
  - CORBA vendor
  - Operating Systems
  - Location
  - Network HW/SW
  - Access
- Dynamic Binding
- Dynamic Typing
- Object Orientation
  - Encapsulation
  - Polymorphism
  - Inheritance
- Instantiation
- Extended Services
  - Naming/trader
  - Events/notification
  - Transactions
  - Security, domains
  - …

In an open specification with multivendor support

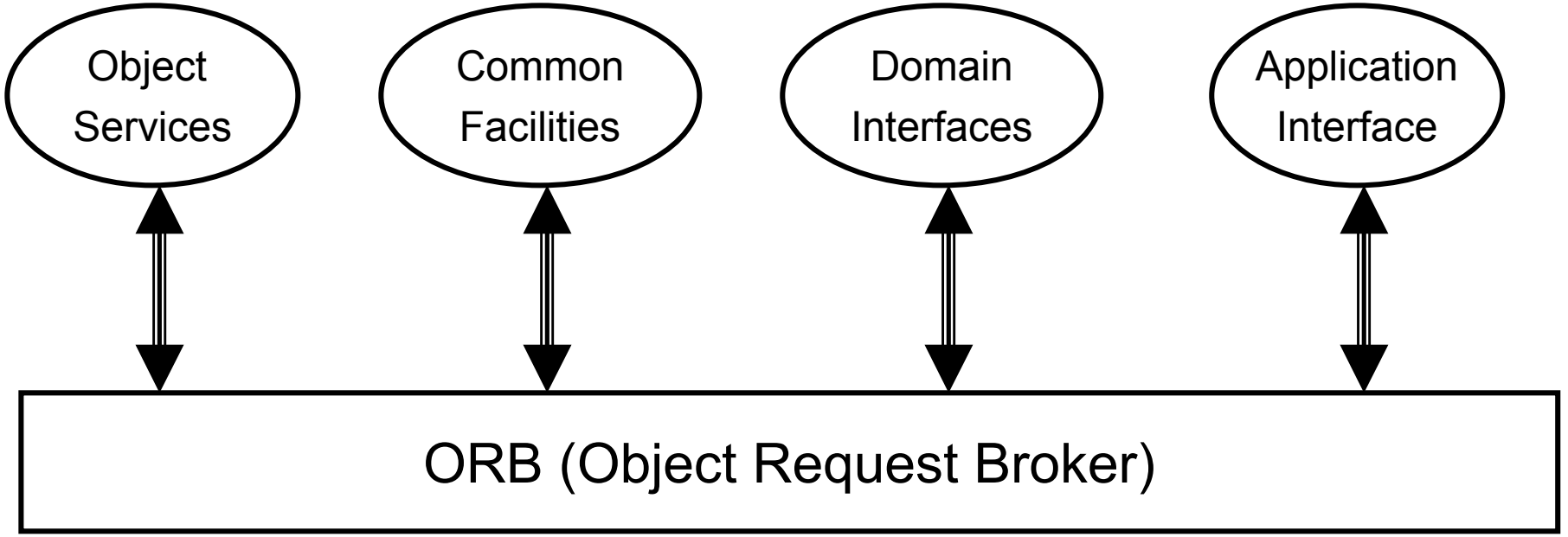
See Object Management Group (OMG) site, www.omg.org

CORBA: A "Software Bus"



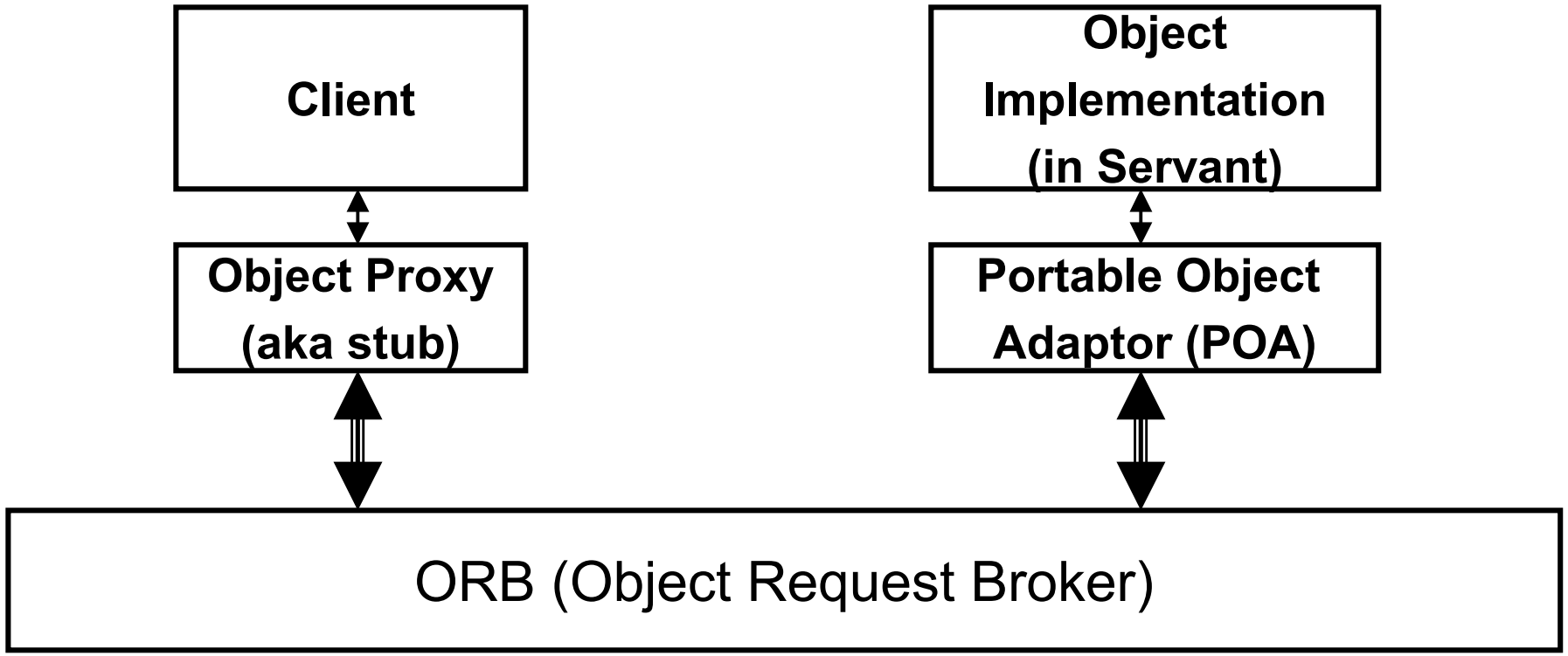
- Hardware bus: lets chips "plug and play" in a standard way
- Software bus: same idea but for software objects

Object Management Architecture (OMA)



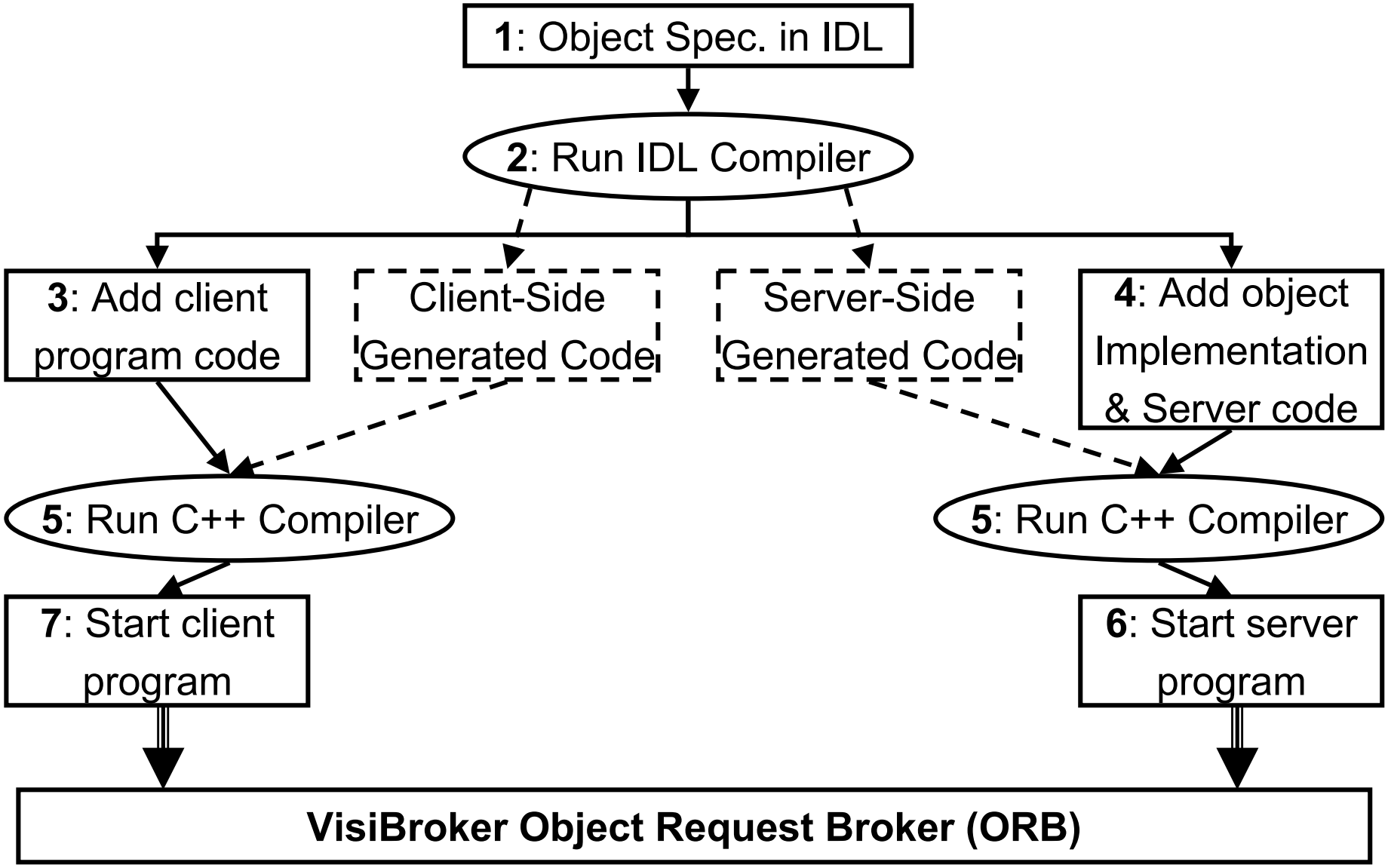
- Object Services: useable by all objects
  - Events, Trader, Security, Naming, Transactions, …
- Common Facilities: useable by all applications
  - Scripting, compound documents, ….
- Domain interfaces: industry-specific APIs
  - Finance, telecom, ….
- Application Interface:
  - What you provide….

ORB, Proxies, and POA



- Note: POA has a tree of objects, starting with root "/"

VisiBroker C++ App. Development Steps



Note: see Chapter 4, “Developing an example application with VisiBroker” in VBcpp PG

1: Object Spec. in IDL

```
// Bank.idl

module Bank {
        interface Account {
                float balance();
        };

        interface AccountManager {
                Account open(in string name);
        };
};
```

2: Run IDL Compiler

- Compiles Bank.idl
- Generates client-side code
  - Bank_c.hh: definitions for Account and AccountManager (stub) classes
  - Bank_c.cpp: internal stub routines used by client
- Generates server-side code
  - Bank_s.hh: definitions for AccountPOA and AccountManagerPOA servant classes
  - Bank_s.cpp: internal routines used by the server
  - Reminder: “server” is a process/program, “servant” is a running piece of code that provides functionality for an object reference

IDL to C++ mapping notes

- IDL isolates implementation from interface
  - Allows clients and servants to be in different languages
  - Allows for an interface to not be bound to a single implementation
    - Useful for many server-side optimizations
    - E.g., multiple servants can service requests to a single object reference
  - Allows for code to be inserted above ORB while still meeting stub API
    - E.g., QuO delegates, in our "Distributed Quality of Service" lectures
    - Caching
- Mappings of Basic IDL types (see REF chapter 3 for details)

IDL type	Java	C++	VisiBroker C++
short	**short**	**short**	CORBA::Short
long	**int**	---[1]	CORBA::Long
unsigned long	**int**	**unsigned long**	CORBA::ULong
float	**float**	**float**	CORBA::Float
double	**double**	**double**	CORBA::Double
boolean	**boolean**	**unsigned char**	CORBA::Boolean
long long	**long**	---[1]	CORBA::LongLong

1: Platform dependent: use the VisiBroker (standard CORBA) types for portability

3: Add client program code (Client.C)

```
#include "Bank_c.hh“  // IDL mappings to Account and AccountManager
int main(int argc, char* const* argv)
{
    try {
        // Initialize the ORB.
        CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

        // Get the manager Id
        PortableServer::ObjectId_var managerId =
                PortableServer::string_to_ObjectId("BankManager");

        // Locate an account manager.
        // Give the full POA name and the servant ID.
        Bank::AccountManager_var manager =
            Bank::AccountManager::_bind(“/bank_agent_poa", managerId);
```

3: Add client program code (cont.)

```
    // use argv[1] as the account name, or a default.
    const char* name = argc > 1 ? argv[1] : "Jack B. Quick";

    // Request the account manager to open a named account.
    Bank::Account_var account = manager->open(name);

    // Get the balance of the account.
    CORBA::Float balance = account->balance();

    // Print out the balance.
    cout << "The balance in " << name << "'s account is $"
         << balance << endl;
} // try block
```

3: Add client program code (cont.)

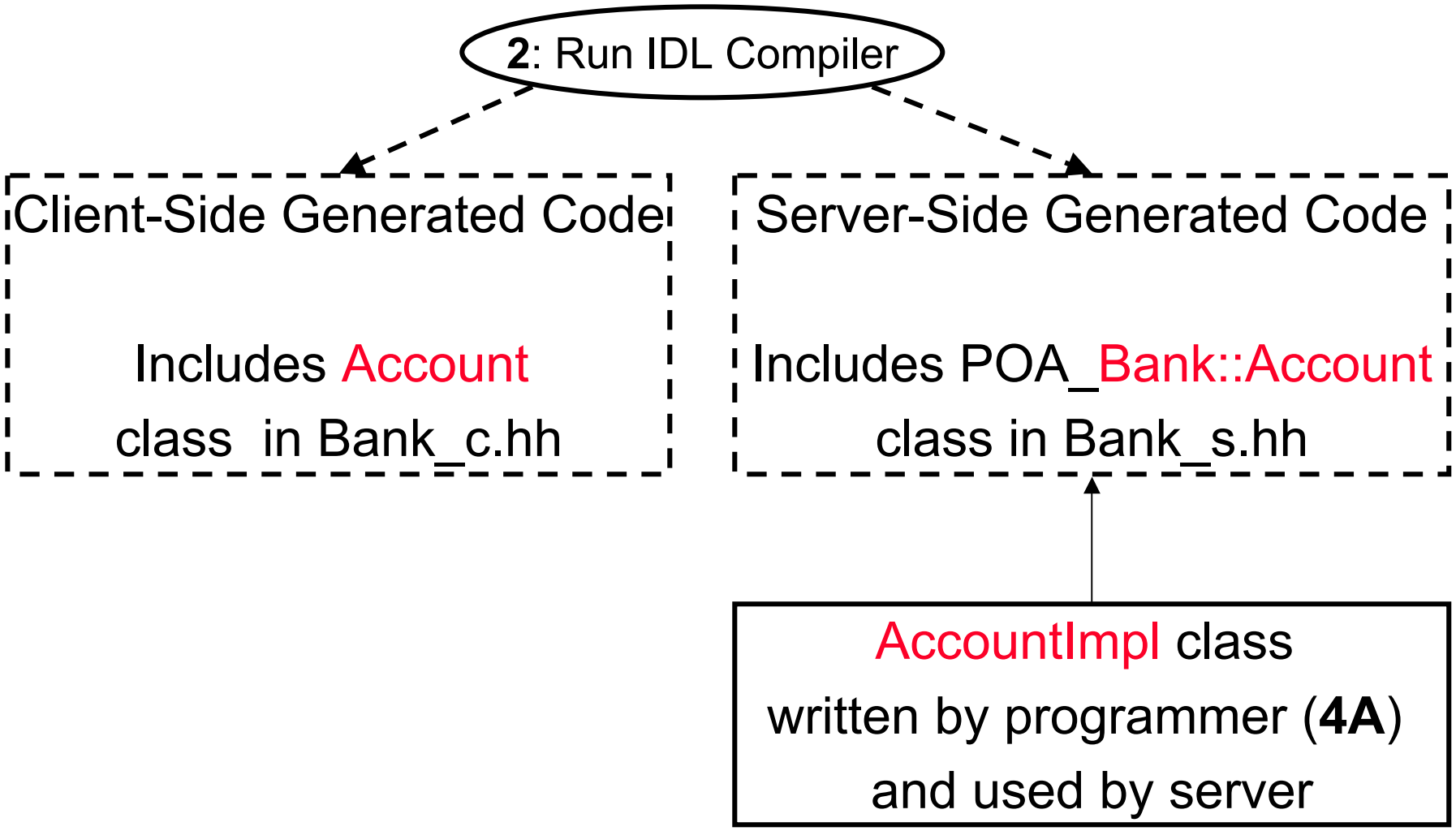
```
        // Print out the balance.
        cout << "The balance in " << name << "'s account is $"
                << balance << endl;
    }  // try block

    catch(const CORBA::Exception& e) {
        cerr << e << endl;
        return 1;
    }

  return 0;
}
```

Peek at the files Bank_c.hh and Bank_c.cpp to see what the client uses …

4A: Add Object Implementations (BankImpl.h)



4A: Add Object Implementations (cont.)

```
class AccountImpl : public virtual POA_Bank::Account,
                    public virtual PortableServer::RefCountServantBase
{
  public:
    AccountImpl(CORBA::Float balance) : _balance(balance) {
    }

    CORBA::Float balance() {
          return _balance;
     }

  private:
    CORBA::Float _balance;
};
```

4A: Add Object Implementations (cont.)

```
class AccountManagerImpl : public POA_Bank::AccountManager
{
  public:
    AccountManagerImpl() {
    }

    Bank::Account_ptr open(const char* name) {
            // Lookup the account in the account dictionary.
            PortableServer::ServantBase_var servant = _accounts.get(name);

             if (servant == PortableServer::ServantBase::_nil()) {
                    // Make up the account's balance, between 0 and 1000 dollars.
                    …
            }
```

4B: Implement the Server(Server.C)

- Initializes the ORB
- Creates a Portable Object Adaptor (POA) with the required policies
- Creates the account manager servant object
- Activates the servant object
- Activates the POA manager (and the POA)
- Waits for incoming requests

4B: Implement the Server (cont.)

```
include "BankImpl.h"

int main(int argc, char* const* argv)
{
  try {
    // Initialize the ORB.
    CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

    // get a reference to the root POA
    CORBA::Object_var obj = orb->resolve_initial_references("RootPOA");
    PortableServer::POA_var rootPOA = PortableServer::POA::_narrow(obj);

    CORBA::PolicyList policies;
    policies.length(1);
    policies[(CORBA::ULong)0] = rootPOA->create_lifespan_policy(
                                    PortableServer::PERSISTENT);
```

4B: Implement the Server (cont.)

```
// get the POA Manager
PortableServer::POAManager_var poa_manager = rootPOA->the_POAManager();

// Create myPOA with the right policies
PortableServer::POA_var myPOA = rootPOA->create_POA("bank_agent_poa",
                                                    poa_manager, policies);
// Create the servant
AccountManagerImpl managerServant;

// Decide on the ID for the servant
PortableServer::ObjectId_var managerId =
                    PortableServer::string_to_ObjectId("BankManager");
// Activate the servant with the ID on myPOA
myPOA->activate_object_with_id(managerId, &managerServant);

// Activate the POA Manager
poa_manager->activate();
```

4B: Implement the Server (cont.)

```
    CORBA::Object_var reference = myPOA->servant_to_reference(
                                                    &managerServant);
    cout << reference << " is ready" << endl;

    // Wait for incoming requests
    orb->run();
  }
  catch(const CORBA::Exception& e) {
    cerr << e << endl;
    return 1;
  }
  return 0;
}
```