Object Persistence with Relational Databases in a Client/Server Environment

Chuck Allison

Obligatory Bio

- **Reusable Component Developer**
  Information & Communications Systems
  LDS Church Headquarters
  Salt Lake City, UT
  72640.1507@compuserve.com
- Contributing Member, **C++ Standards**
  (designed bitset template class)
- Consulting Editor, **C/C++ Users Journal**

Workshop Goals

- Appreciate Challenges of Client/Server
- Understand Persistence Issues
- Examine our (evolving) solution: **PFX**
- Illustrate Object-oriented Design
- See C++ in action in Corporate Applications

Persistence Alternatives

- OODBMS
- Off-the-shelf Middleware
- Roll your own

Alternative: OODBMS

- Easy to use
- New Technology
  - not yet embraced by IS (not yet P.C.)
- RDBMS is *Entrenched*
  - mature (efficient, robust)
  - existing mission-critical databases
  - expensive to migrate

Alternative: OTS Middleware

- Not yet “mature”
  - lacking in functionality
  - few platforms supported
- Mapping to tables inflexible or inefficient

Alternative: In-house

- Can also be expensive
- “Guaranteed” to meet requirements
- *Can* be reused
- Lotsa fun!

Our Solution

PFX (Persistence Framework)

- Reusable C++ Framework
- Small (6 classes, 1 DLL)
- Efficient
- Supports concurrent users
- Manages distributed objects
- Limited transaction and ad hoc query support

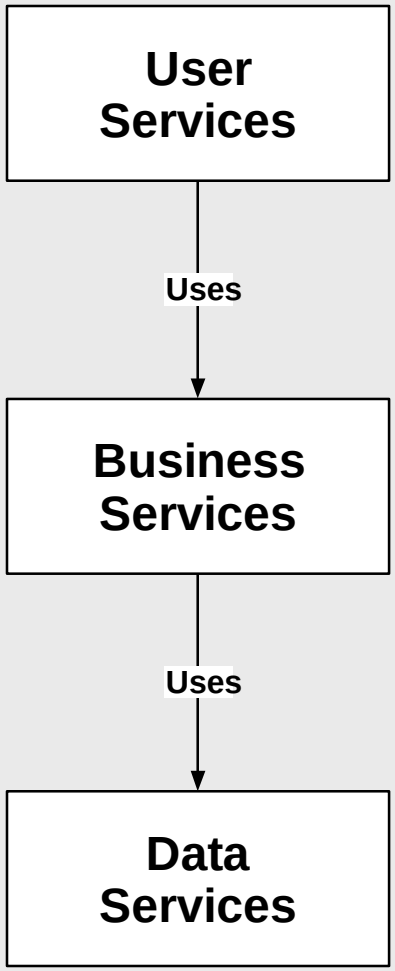
Client/Server
Driving Forces

- Downsizing
  - lower cost of PCs
- Upsizing
  - shared devices, workgroups
- Rightsizing
  - distributed services
  - network of heterogeneous services *is* the computer

Client/Server
Enabling Technologies

- Encapsulation
  - Separate *Interface* from *Implementation*
- Concurrent Processing
- Distributed Processing
  - separate producer and consumer
  - optimize throughput

3 Tiers of C/S Architecture



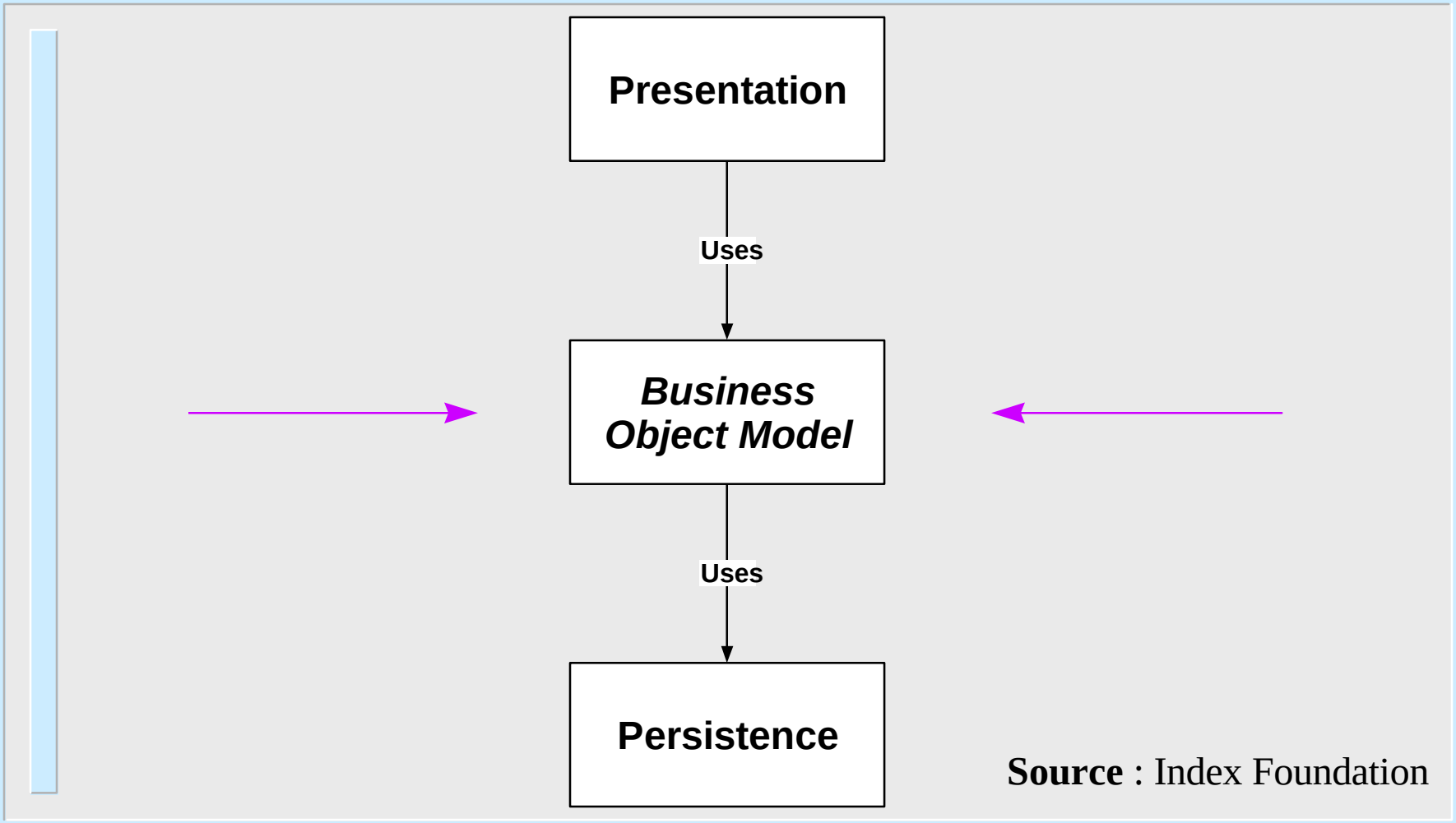
3-tier C/S Architecture
Benefits

- Division of Labor
  - development & production
  - right platform for the task
- Localized maintenance
- Reuse Opportunities
  - shared resources
  - pluggability
  - extensibility

The Role of Objects

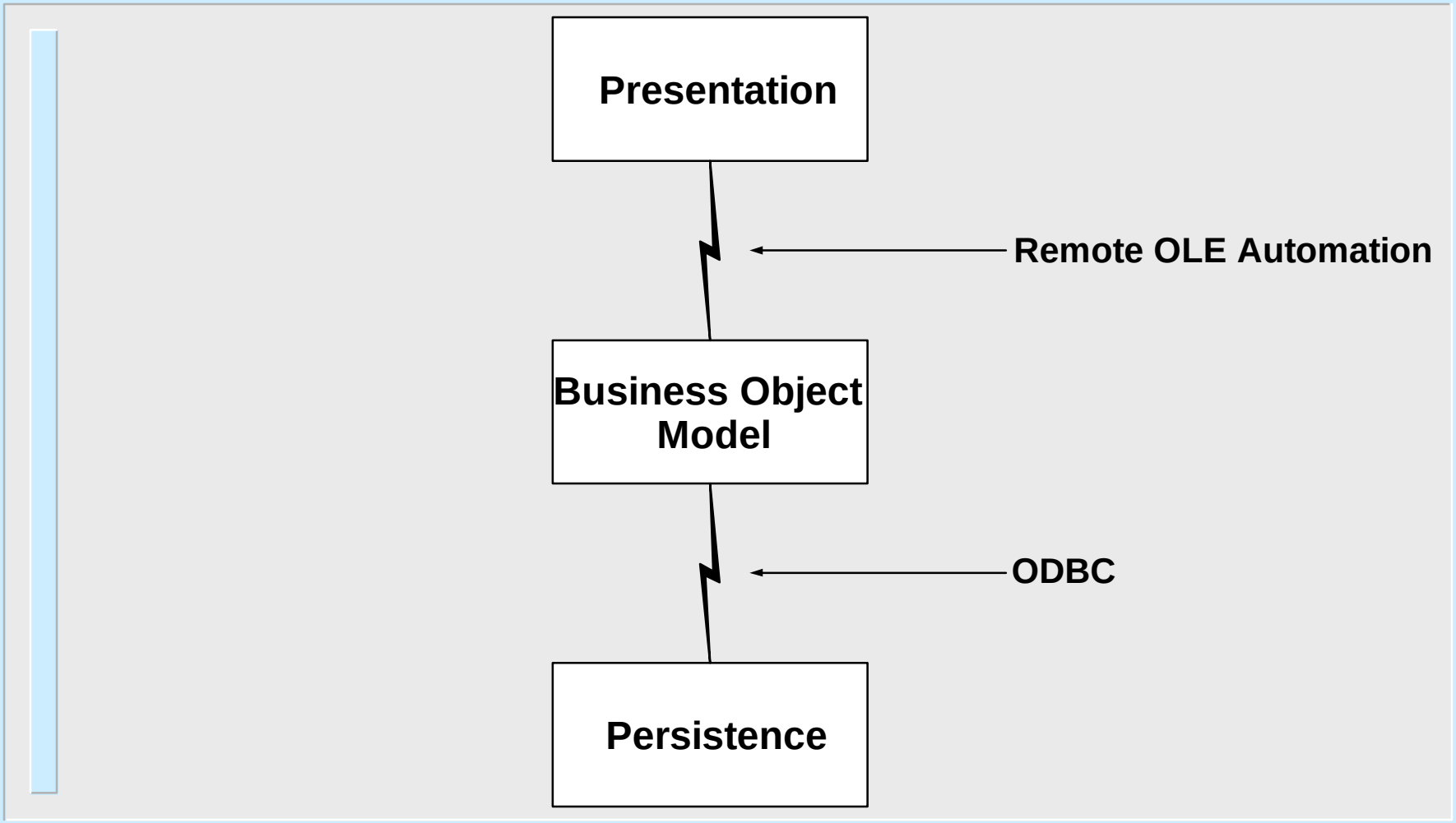
- Model business directly
- Business objects are the *focus*
- User Services merely *present* business objects
- Data Services merely provide *persistence*

Object-oriented 3-tier Architecture

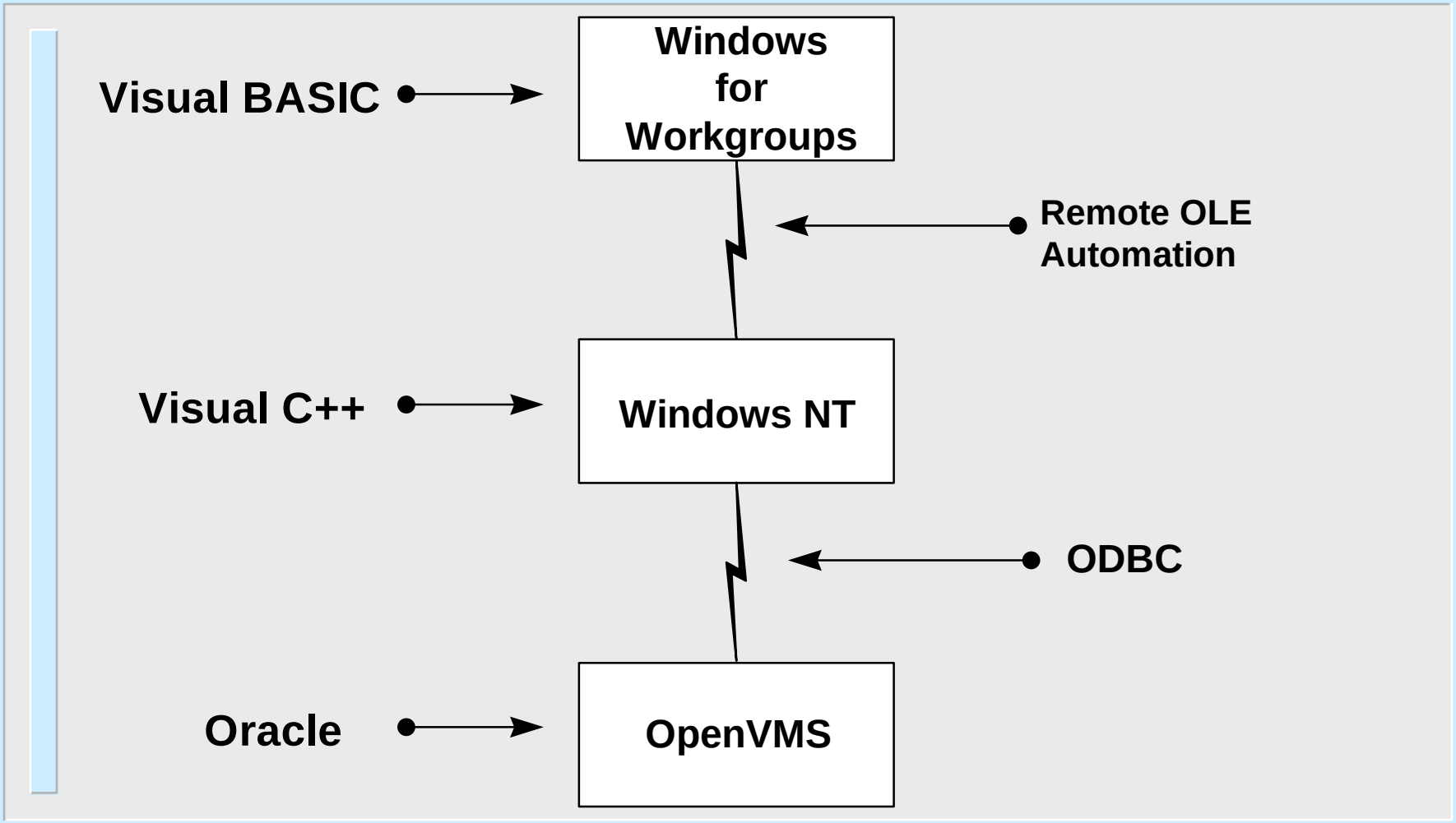


Source : Index Foundation

Connectivity



Platforms & Tools

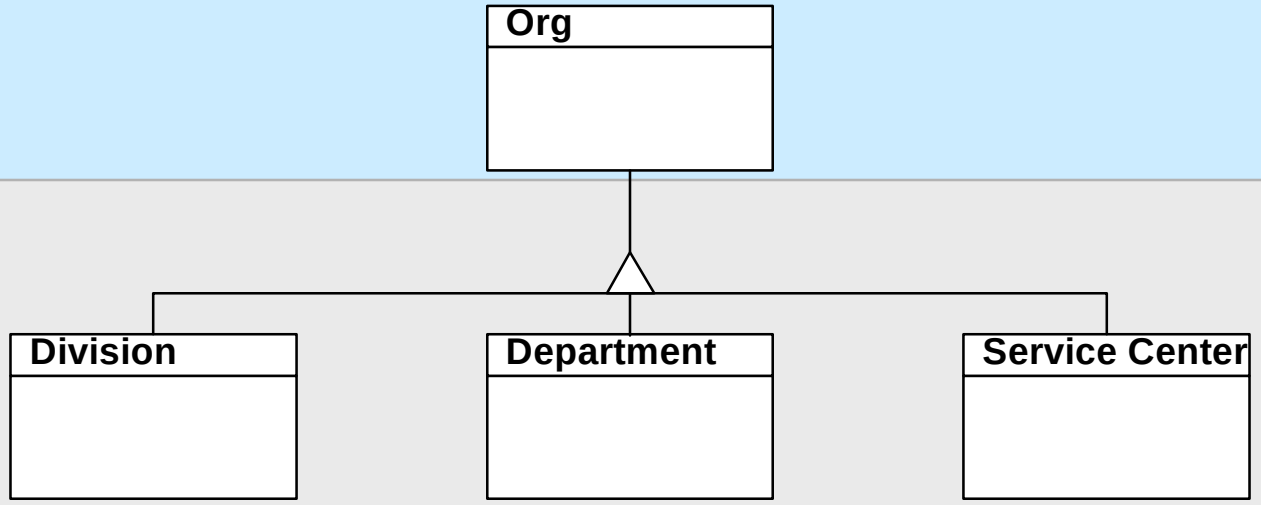


Mapping Objects to Tables

- Each class in its own table
  - what about embedded subobjects?
  - what about object referents?
- Inheritance alternatives:
  - 1) each class in its own table
  - 2) only leaf classes in own tables
  - 3) entire hierarchy in one table

Our Mapping Approach

- One *hierarchy* per table
  - typeID field
  - variable-length records
- Trivial subobjects “flattened”
  - objects w/o “identity”
  - stored *in situ* in DB row
- Business subobjects are pointed to:
  - object ID as foreign key in DB row
  - pointer in business object
  - queries performed as needed transparently



ObjectID	TypeID	Name	ManagerID	LocationID
1	2	Accounting	1017	23
2	3	Western Region	876	9

C++ Class Definitions

```
class Org
{
	long ObjectID;
	long TypeID;
	CString Name;
	long LocationID;
	long LeaderID;
public:
	Org* getParentOrg() const;
	Location* getLocation() const;
	Person* getLeader() const;
protected:
	Org(long id = 0);
};

class Division : public Org
{
public:
	Division(long id = 0);	// Will invoke Org ctor
	// etc...
```

ODBC

- “Open Database Connectivity”
- Connect to remote databases
  - relational
  - flat-file
- Supports SQL
- A *Standard*
  - requires drivers for your platform/DBMS
  - need more 32-bit support

ODBC via MFC

- **`CDatabase`** class
  - manages a *connection*
- **`CRecordset`** class
  - manages query/update operations
  - associated with a table or view
  - created by the VC++ *Class Wizard*

Class **EmpRecSet**

North Wind Traders DB (MSAccess)

```
class EmpRecSet : public CRecordset
{
public:
    EmpRecSet(CDatabase* pDatabase = NULL);

    long        m_Employee_ID;
    CString     m_Last_Name;
    CString     m_First_Name;
    CString     m_Home_Phone;
    long        m_Reports_To;
};
```

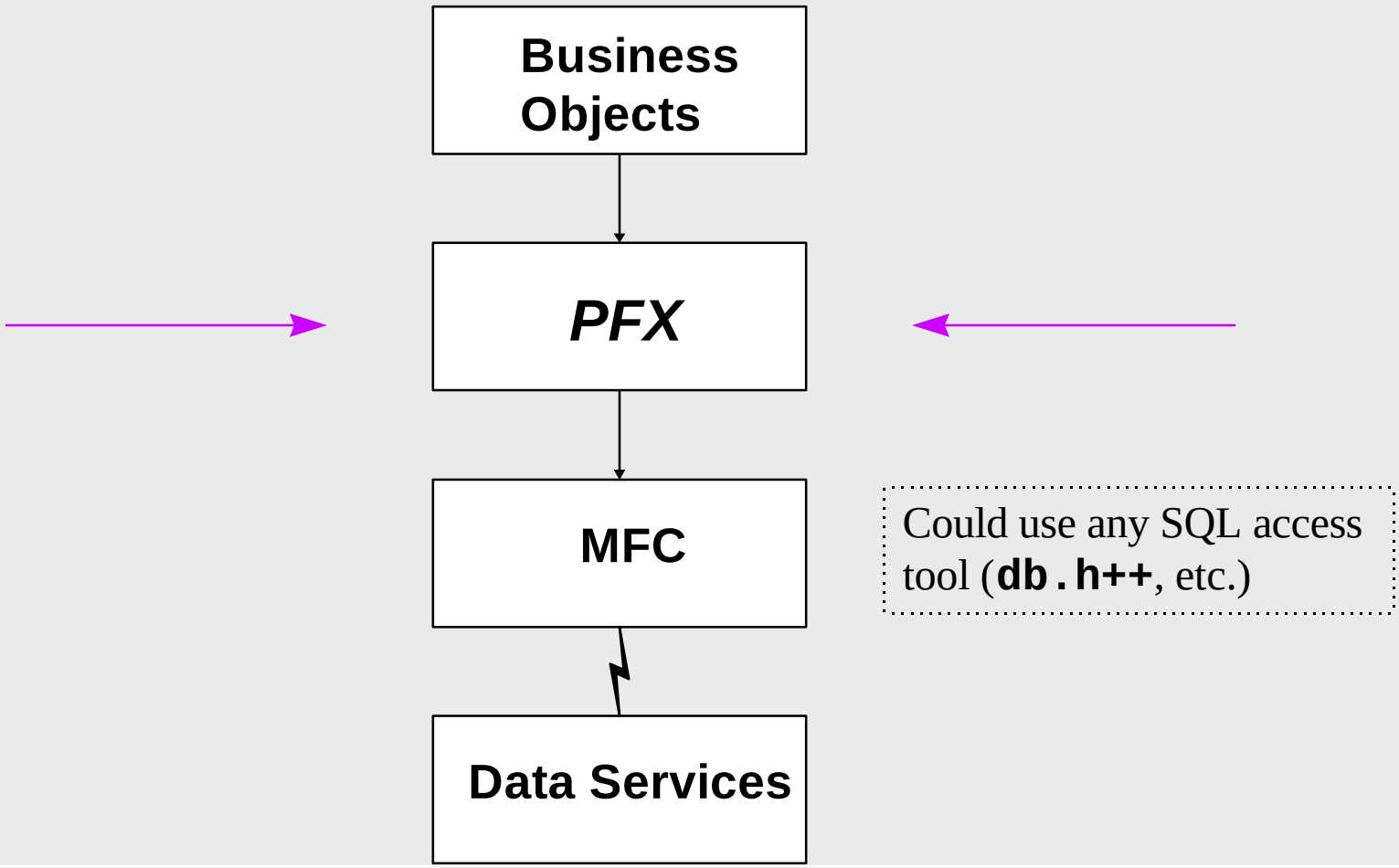
MFC Example

```
CDatabase db;
db.Open(0,FALSE,TRUE,"ODBC;DSN=North Wind");

EmpRecSet e(&db);
e.m_strFilter = "[Employee ID] > 5";
e.Open();
while (!e.IsEOF())
{
    outf << '{' << e.m_Employee_ID
         << ',' << e.m_Last_Name
         << ',' << e.m_First_Name << "}\n";
    e.MoveNext();
}
```

Modularity Concerns

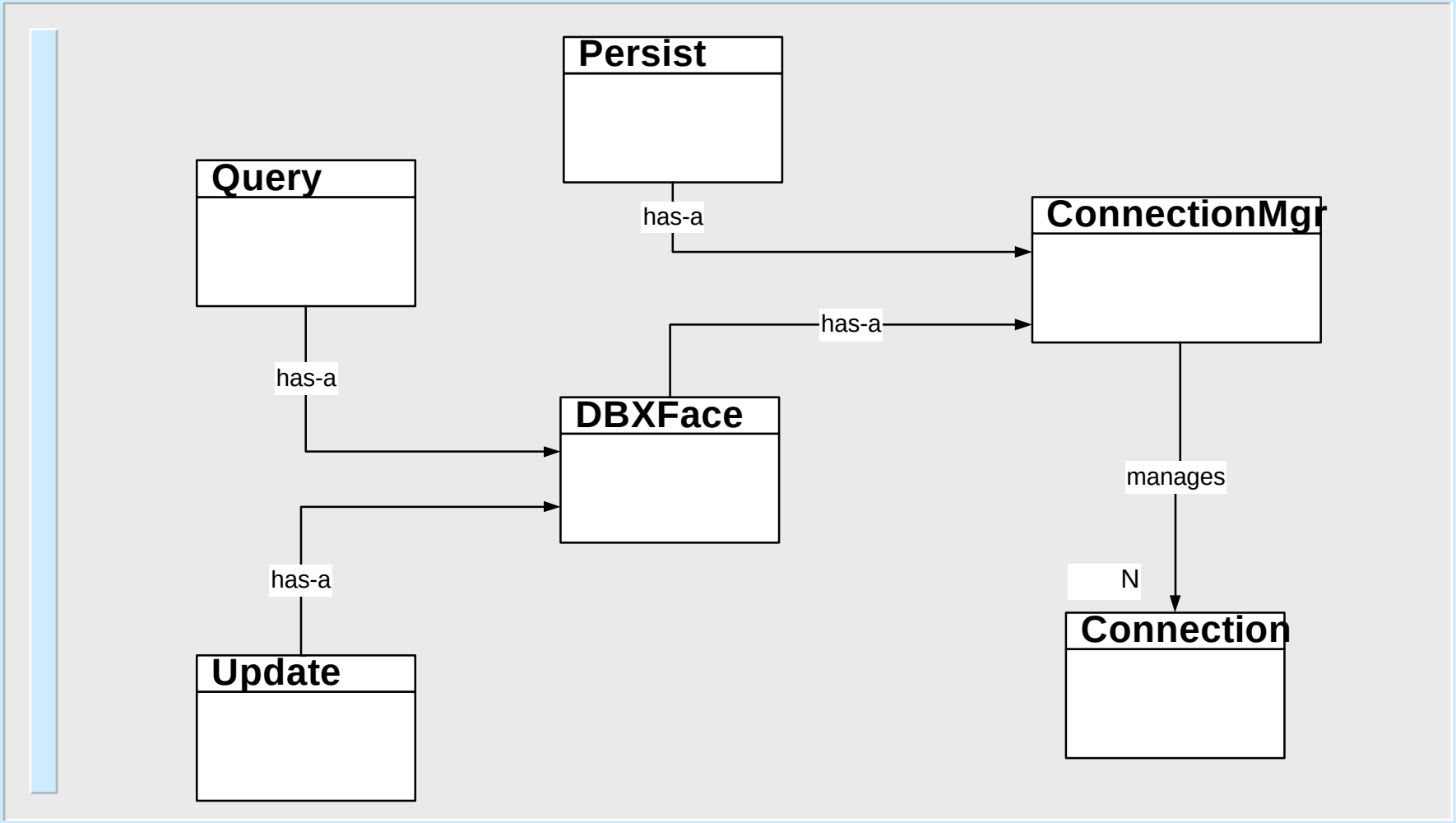
- Business Objects shouldn’t know about MFC
- We may change vendors, tools, etc.
- Want true pluggability
  - both software and hardware

PFX: A Persistence Framework



PFX

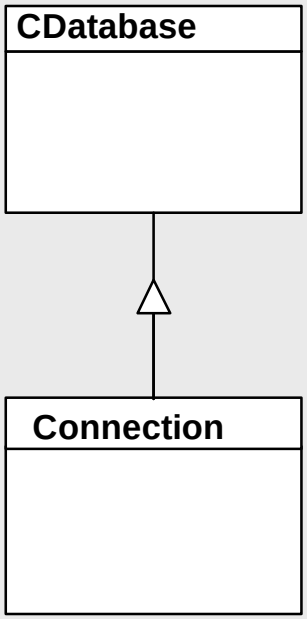
Class Diagram



Class **`Connection`**

(*concrete*)

- Establishes an ODBC connection
- Transactions via `CDatabase`
- Corrects `CDatabase::Open`



Class **`ConnectionMgr`**

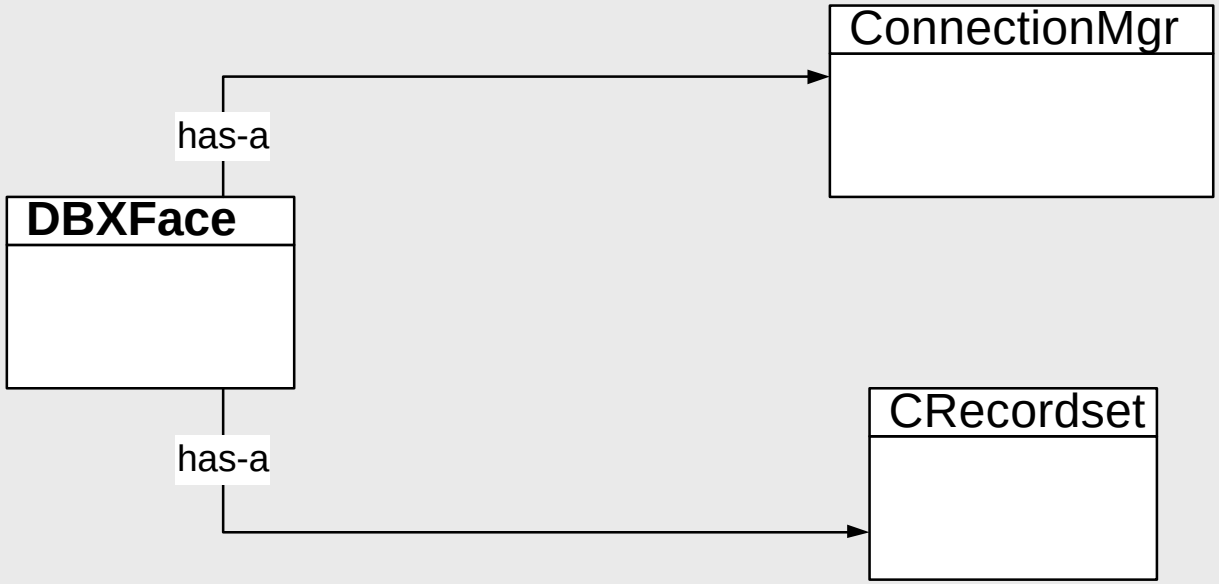
(concrete)

- The User's handle into the system
- Manages a list of Connections
- Keeps reference count of all client objects

Class **DBXFace**

(abstract)

- Manages all interaction with DB
- Constructed by user `Query` & `Update` derived objects



Class **DBXFace**

Public Interface

```
class DBXFace
{
public:
    virtual CString  BusObjName() const = 0;
    virtual CString  TableName() const = 0;
    virtual CString  ConnectionStr() const = 0;
    virtual CString  SequenceStr() const = 0;
    virtual double  GetObjID() const = 0;
    virtual ~DBXFace();
```

Class **DBXFace**

Protected & Private Declarations

```
protected:
    DBXFace(ConnectionMgr* pCM);
    CRecordset* m_pRecordset;
    ConnectionMgr* m_pConnectionMgr;

    static CString s_IniFile;

private:
    // Disallowed operations
    DBXFace(const DBXFace&);
    void operator=(const DBXFace&);
};
```

Sample INI file

```
[Employee]
Table=Employee
ConnectionStr=ODBC;DSN=North Wind
SequenceStr=S_EmployeeID
NumFields=5
0=[Employee ID]
1=[Last Name]
2=[First Name]
3=[Home Phone]
4=[Reports To]
```

*(1 for each object type; objects can be **distributed**)*

Class **`Query`**

(abstract)

- Uses a DBXFace to read databases
- Limited ad hoc query capability
- Navigates result set

Class **Query**

Public Interface

```
class Query
{
public:
    BOOL Execute(UINT = CRecordset::snapshot);

    // Navigation functions:
    BOOL First();
    BOOL Last();
    BOOL Next();
    BOOL Prev();
    BOOL Move(long);
    BOOL IsEmpty() const;
    BOOL IsEOF() const;
    BOOL IsBOF() const;
    long Index() const;
    long GetRecordCount() const;
```

Class **Query**

Public Interface (cont.)

```
// Query-building functions:
CString ColumnName(int) const;
void StartSelect();
void AddSelectCriteria(int, const CString&);
void AddSelect();
void EndSelect();
void ResetSelect();

void StartOrder();
void AddOrderCriteria(int, int);
void EndOrder();
void ResetOrder();
```

Class **Query**

Protected Interface

```
protected:
    Query(DBXFace*);
    DBXFace* m_ pDBXFace;
};
```

Class **`Update`**

(abstract)

- Uses objects derived from DBXFace to update database polymorphically
  - insert
  - update
  - remove

Class **Update**

```
class Update
{
public:
    BOOL  Write();
    void  Remove();

protected:
    DBXFace* m_pDBXFace;

    Update(DBXFace*);
    double  GetObjID() const;
    double  NextID();
};
```

```
double Update::NextID()
{
    DualSet d(m_pDBXFace->SequenceStr(),
              m_pDBXFace->GetConnection());
    d.Open();
    return d.m_ID;
}

BOOL Update::Write()
{
    CRecordset* pRS = m_pDBXFace->GetRecordset();
    return pRS-> Update();
}

void Update::Remove()
{
    CRecordset* pRS = m_pDBXFace->GetRecordset();
    pRS-> Delete();
}
```

Class **`Persist`**

(abstract)

- *mixin* for business object classes
- provides object ID
- cooperates with user `Query` and `Update` derived objects polymorphically

Class **Persist**

Public Interface

```
class Persist
{
public:
     virtual ~Persist();
     virtual BOOL  Write() = 0;
     virtual void  Remove() = 0;
     operator void*() const;

     // Accessors
     double GetObjID() const;
     ConnectionMgr* GetConnectionMgr() const;
```

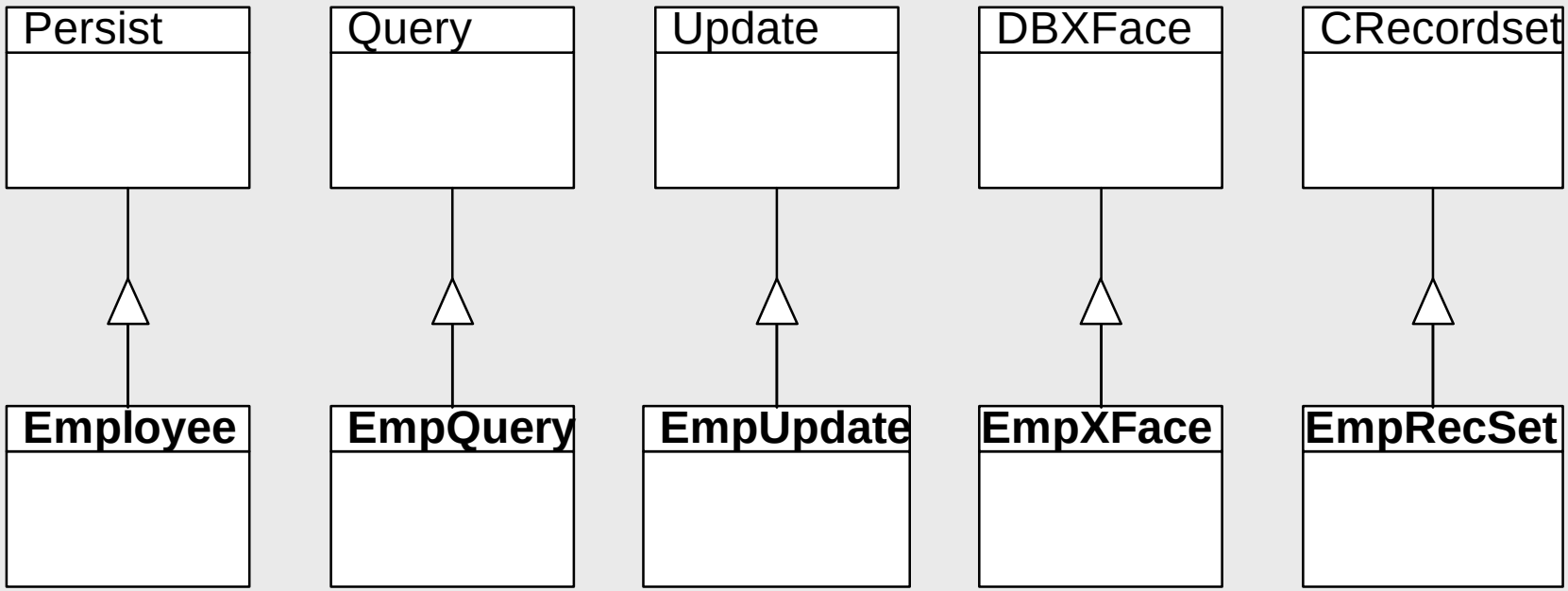
Class **Persist**

Protected Interface

```
protected:
      Persist(ConnectionMgr*, double id = 0.0);

      double m_ ObjID;
      ConnectionMgr* m_ pConnectionMgr;
      BOOL m_ValidFlag;
};
```

Using ***PFX***

- Derive from framework (Employee example):



Class **EmpXFace**

```
class EmpXFace : public  DBXFace
{
public:
    typedef EmpRecSet* RecPtr;          // For casting
    enum{OBJ_ID,LAST_NAME,FIRST_NAME,HOME_PHONE,REPORTS_TO};

    EmpXFace(ConnectionMgr* pCM);
    CString BusObjName() const;
    CString TableName() const;
    CString ConnectionStr() const;
    CString SequenceStr() const;
    double GetObjID() const;

private:
    static const CString s_BusObjName;
    static CString s_TableName;
    static CString s_ConnectionStr;
    static CString s_SequenceStr;
};
```

```
// empxfce.cpp:
const CString EmpXFace::  s_BusObjName = _T("Employee");
CString EmpXFace::s_TableName = _T("");
CString EmpXFace::s_ConnectionStr = _T("");
CString EmpXFace::s_SequenceStr = _T("");

EmpXFace::EmpXFace(ConnectionMgr* pCM)
        : DBXFace(pCM)
{
    Connection* pCon =
        m_pConnectionMgr->Connect(ConnectionStr());
    m_pRecordset = new EmpRecSet(pCon);
}

CString EmpXFace::BusObjName() const
{
    return s_BusObjName;
}
```

Class **EmpQuery**

```
class EmpQuery : public  Query
{
    typedef EmpXFace::RecPtr RecType;

public:
    EmpQuery(ConnectionMgr*);
    double GetObjID() const;
    CString GetLastName() const;
    CString GetFirstName() const;
    CString GetHomePhone() const;
    long GetReportsTo() const;
};
```

Class **EmpQuery**

continued...

- All functions inlined

```
inline EmpQuery::EmpQuery(ConnectionMgr* pCM)
                : Query(new EmpXFace(pCM))
{}

inline double EmpQuery::GetObjID() const
{
 return
 RecType(m_pDBXFace->GetRecordset())->m_Employee_ID;
}

// etc.
```

Using **EmpQuery**

```
ConnectionMgr* pCM = ConnectionMgr::Create();
EmpQuery q(pCM);

// Build query:
q.StartSelect();
q.AddSelectCriteria(EmpXFace::OBJ_ID,"field > 5");
q.EndSelect();
q.Execute();

// Navigate:
while (!q.IsEOF())
{
    // whatever, then advance:
    q.Next();
}
pCM->Detach();
```

Class **EmpUpdate**

```
class EmpUpdate : public  Update
{
    typedef EmpXFace::RecPtr RecType;
    friend class Employee;

public:
    EmpUpdate(ConnectionMgr*, double = 0.0);
    void Copy(const Employee&);
};
```

```
EmpUpdate::EmpUpdate(ConnectionMgr* pCM, double id)
          : Update(new  EmpXFace(pCM))
{
    // Initialize recordset with where clause
    RecType pRS = RecType(m_pDBXFace->GetRecordset());
    char buf[21];
    sprintf(buf,"%s = %f",
            m_pDBXFace->  SQLName(EmpXFace:: OBJ_ID), id);
    pRS->m_strFilter = buf;
    pRS->Open();

    // Put recordset in correct mode
    if (id != 0.0)
    {
        pRS-> Edit();    // edit existing record
    }
    else
    {
        pRS-> AddNew();  // Insert new record
        pRS->m_Employee_ID =   NextID();
    }
}
```

Class **Employee**

Public Interface

```
class Employee : public  Persist
{
public:
    Employee(ConnectionMgr* cp, double id = 0.0);
    Employee(const EmpQuery& q);
    BOOL  Write();
    void  Remove();

    // Get- and Set- functions:
    CString GetLastName() const;
    CString GetFirstName() const;
    CString GetHomePhone() const;
    long GetReportsTo() const;
    void SetLastName(const CString&);
    void SetFirstName(const CString&);
    void SetHomePhone(const CString&);
    void SetReportsTo(long);
```

Class **Employee**

Private Declarations

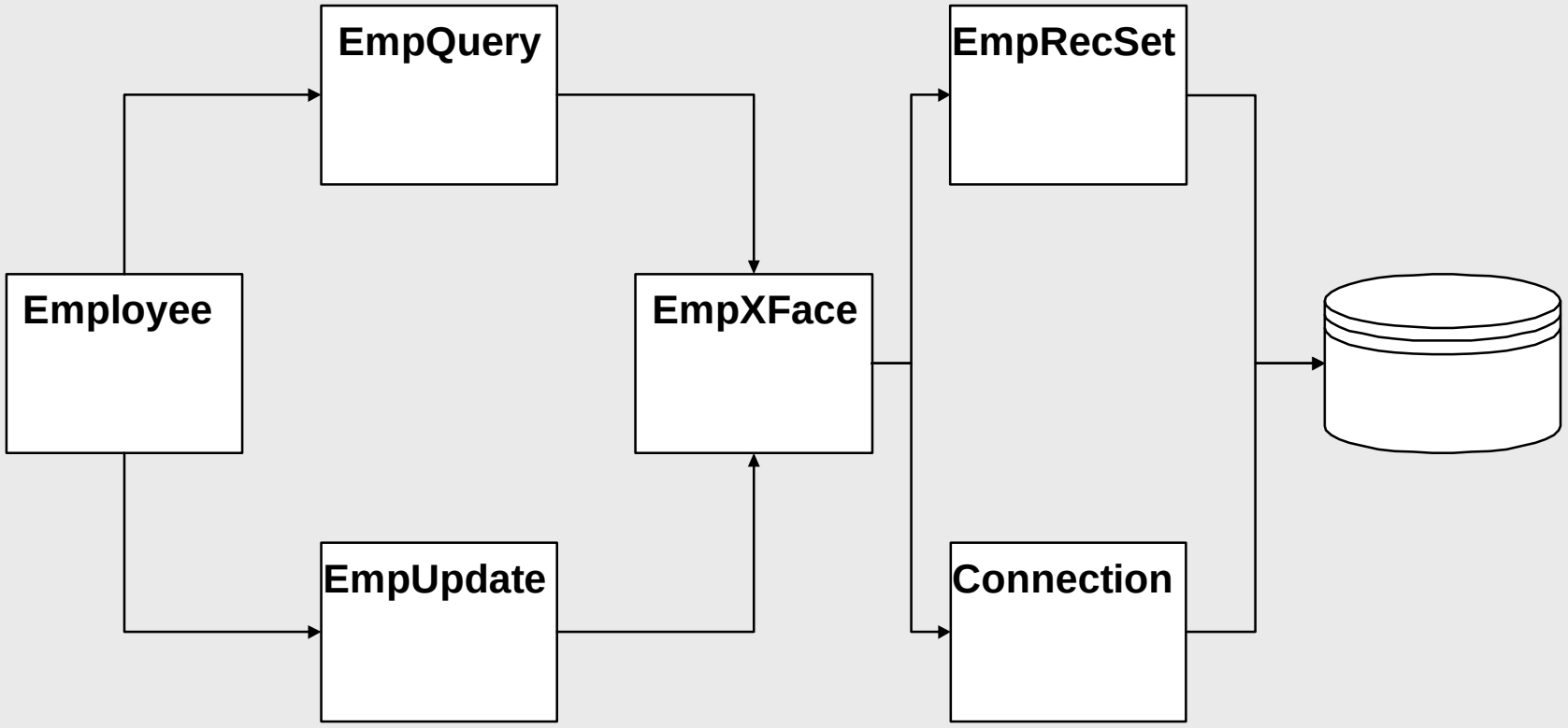
```
private:
    CString m_LastName;
    CString m_FirstName;
    CString m_HomePhone;
    long m_ReportsTo;

    // Disabled operations :
    Employee(const Employee&);
    void operator=(const Employee&);
};
```

Using an **Employee** Object

```
// Request a ConnectionMgr:
ConnectionMgr* pCM = ConnectionMgr::Create();

// Retrieve an employee (via ObjID):
Employee emp(pCM, 1);
cout << emp.GetLastName();    // etc.

// Create a new employee:
Employee newEmp(pCM);
newEmp.SetLastName("Doe");    // etc.
newEmp.Write();

pCM->Detach();
```

Employee Example

Object Interaction Graph



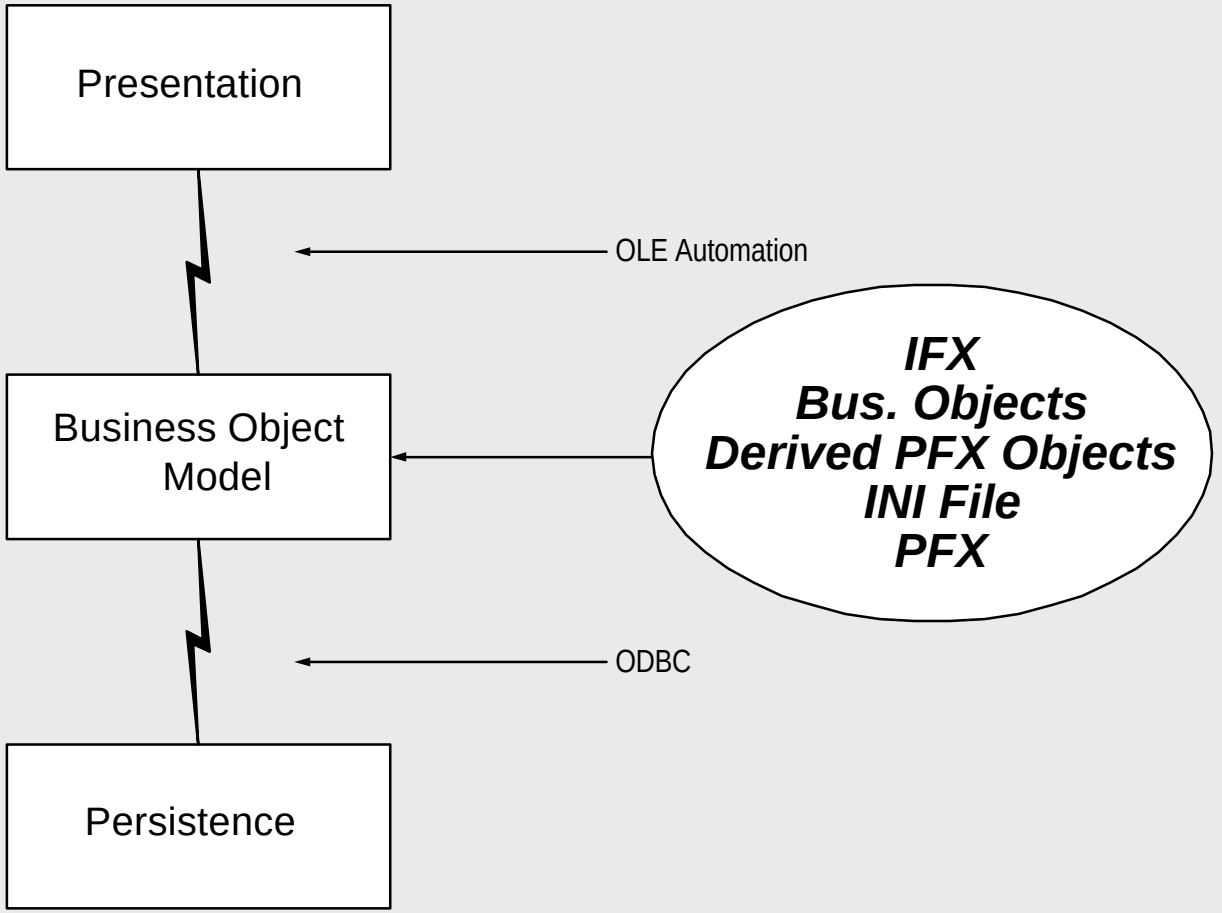
Using ***PFX***

Summary

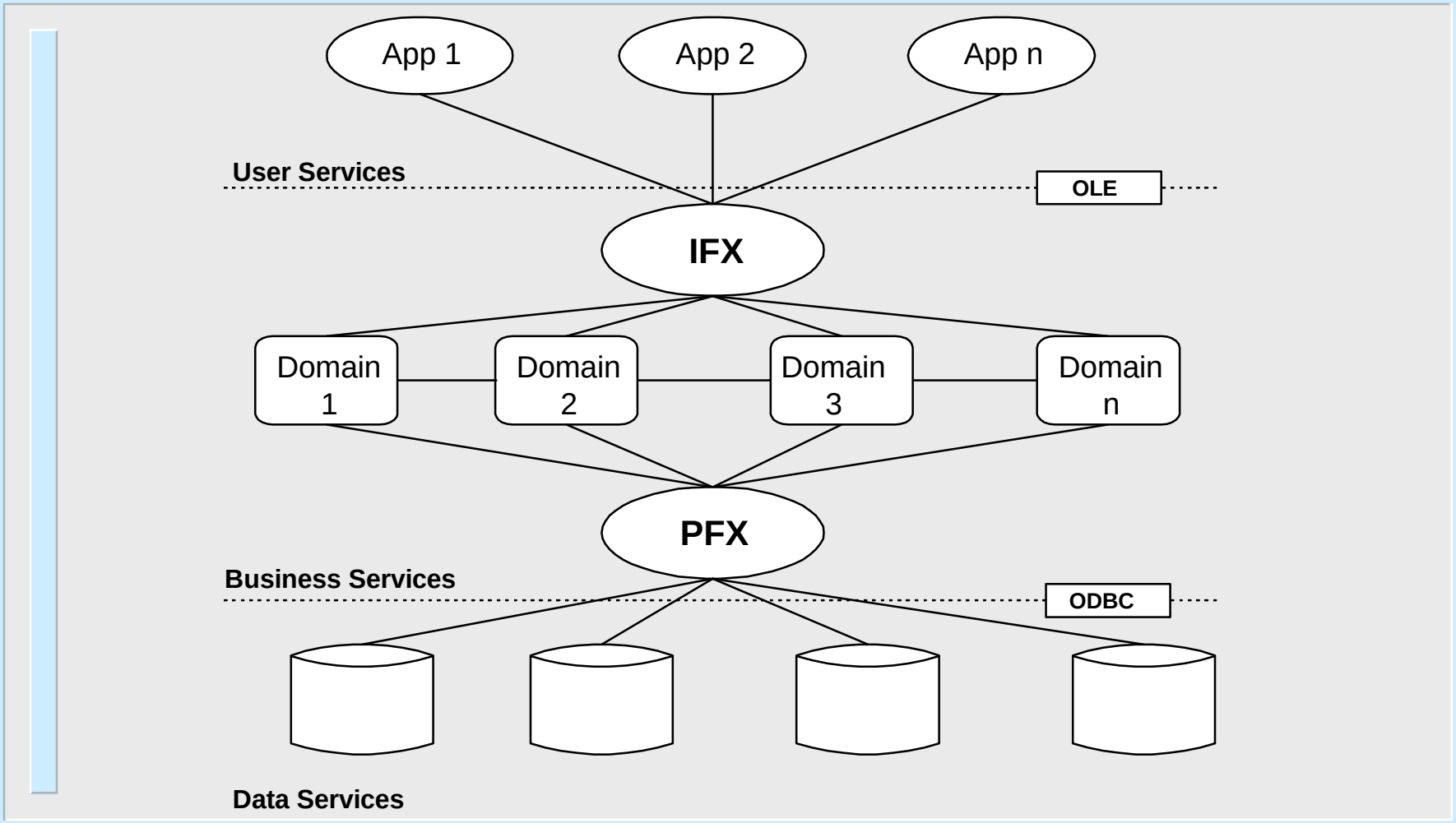
- Create new Recordset with *Class Wizard*
- Derive from `DBXFace`
  - business object name, field IDs
- Derive from `Query`
  - instantiate `DBXFace` object, accessors
- Derive from `Update`
  - open recordset in edit/add mode
- Derive from `Persist`
  - use query for reading, update for writing

Summary

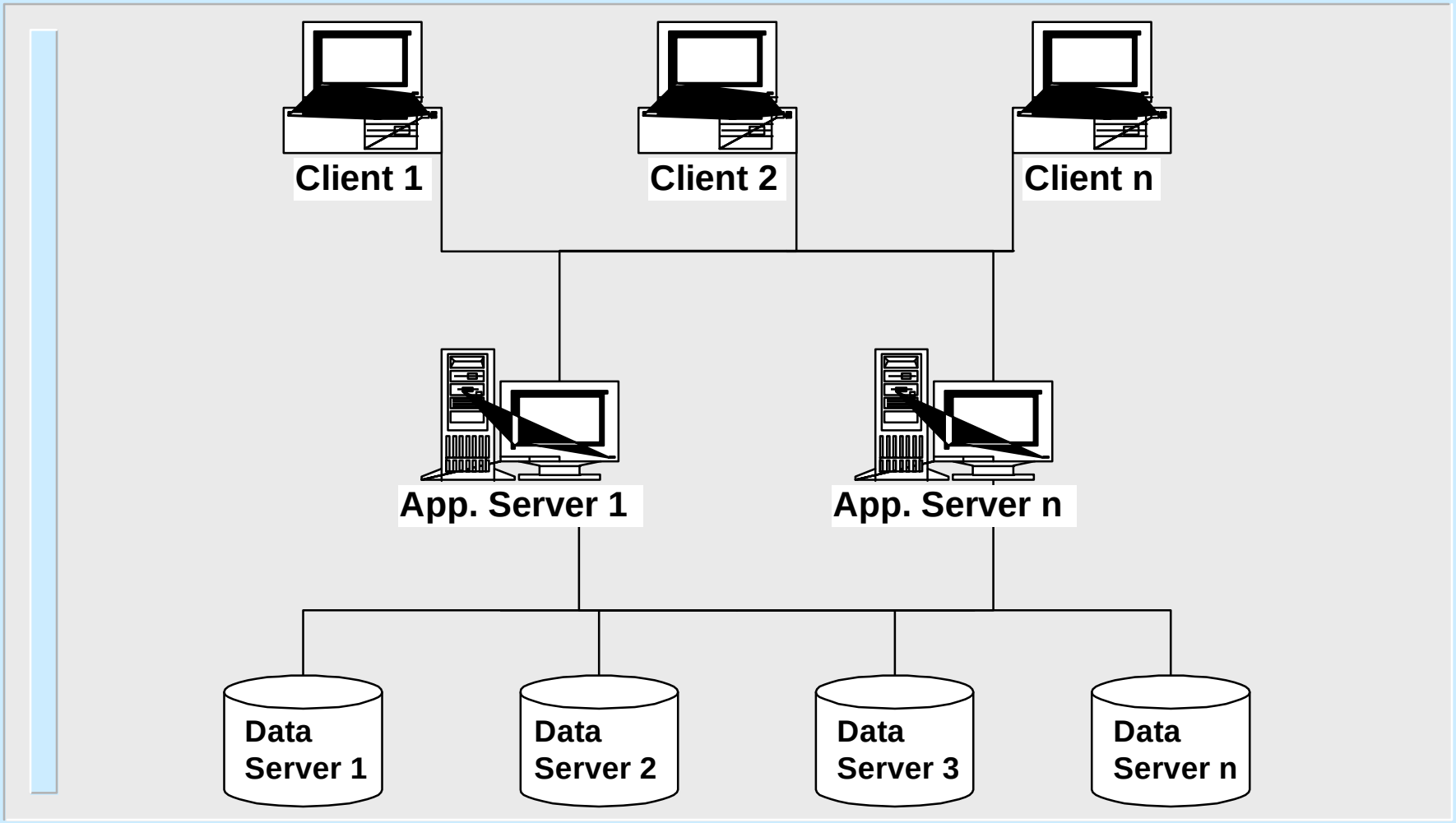
PFX Role in C/S



The “Big Picture”



Network Architecture



Learnings

- 3-tier makes sense
  - implementation can be physical 2-tier
- Focus on Business Objects
  - User Services just presents
  - Data Services just stores
- C++ is ***Cool***
  - fast, flexible, fun
- So is ***PFX!***