The System Object Model (SOM): A Technology for Language Independent Objects

Roger Sessions
Object Technology Products Group
IBM, Austin, Texas

Copyright

What will be covered

- SOM Overview - Goals and Architecture
- Examples of SOM classes
- Comparison to existing technology
- Summary
- References

SOM Overview

SOM is a technology for packaging object-oriented class libraries.

SOM Design Goals:

- Language Neutral
- State of the Art Object-Oriented Capability
- Support for Industrial Strength Class
- Libraries

Who Needs SOM?

You Need SOM If...

- Your business is creating Class Libraries OR
- Your applications make use of other people's Class Libraries (e.g. the WPS)

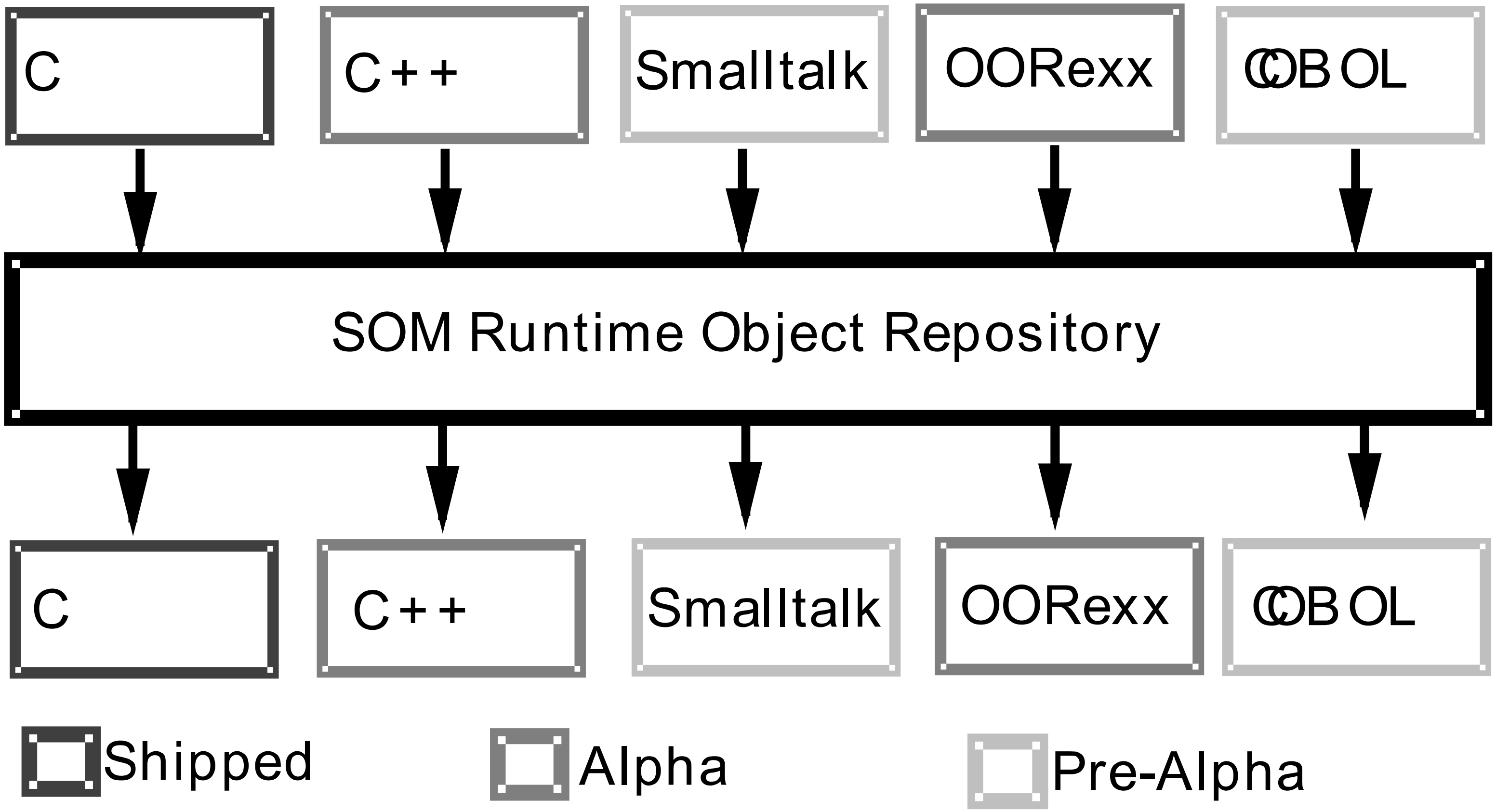
You Do Not Need SOM If...

- Your business is creating standalone applications AND
- You are not using SOM packaged libraries.

SOM and the Workplace Shell

- The Workplace Shell is the first commercially available Class Library Packaged with SOM
- SOM is the packaging technology
- The Workplace Shell is the Class Library
- SOM is the tool you will use to modify
- the Workplace Shell Class Library

SOM Architecture



C

C++

Smalltalk

OORexx

COBOL

SOM Runtime Object Repository

C

C++

Smalltalk

OORexx

COBOL

Shipped

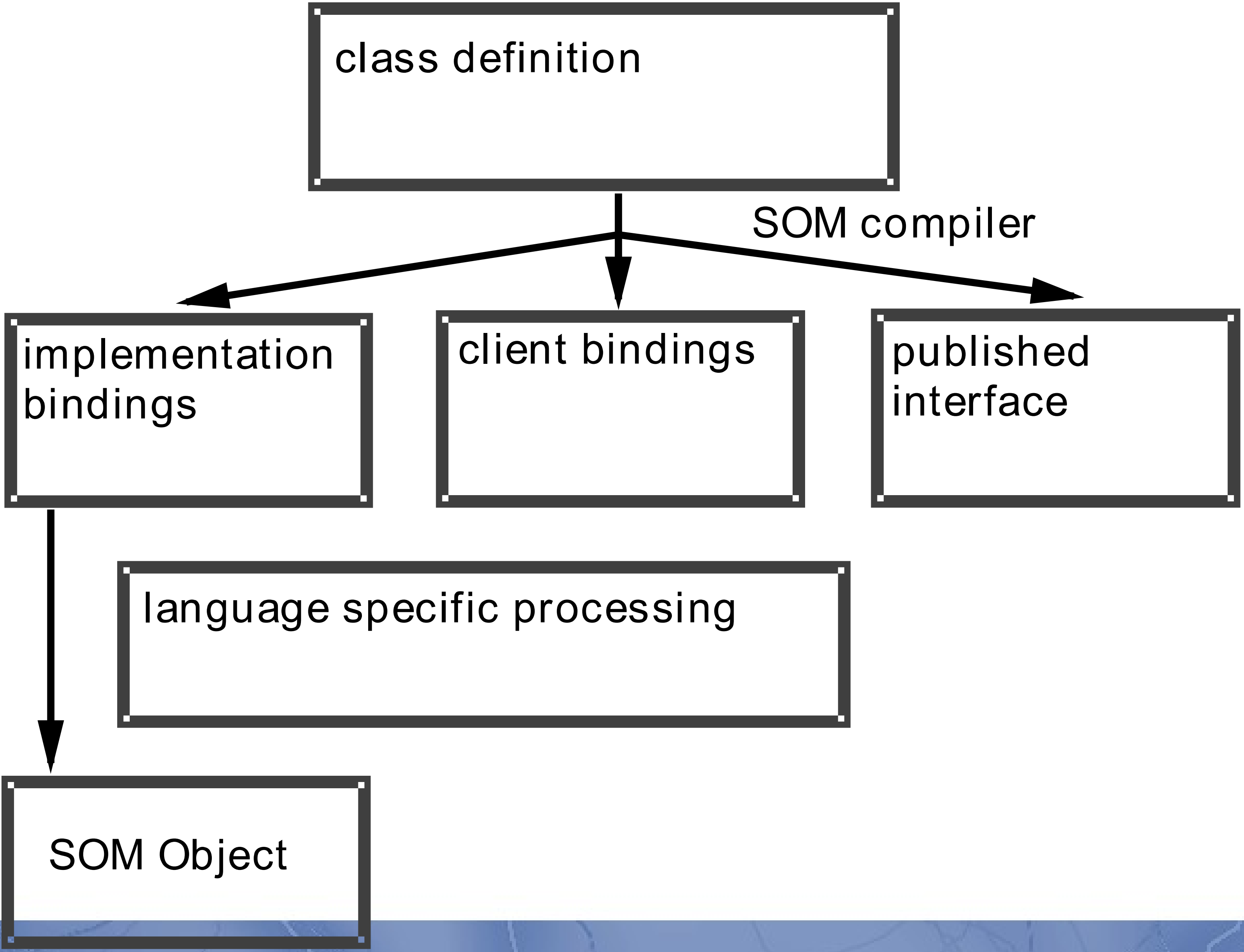
Alpha

Pre-Alpha

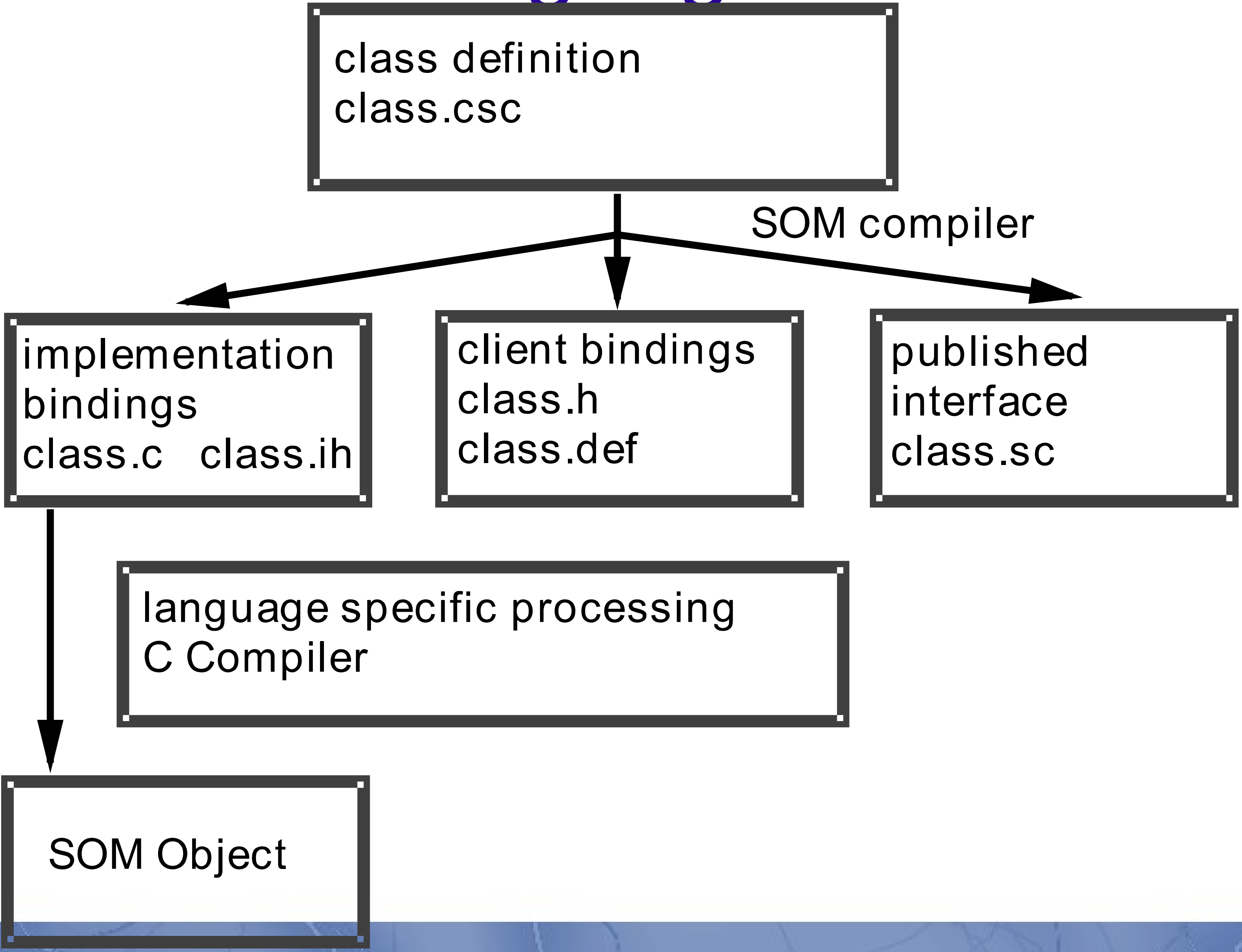
SOM enhances existing languages

SOM does not replace existing languages

The SOM Process



The C Language Process



class definition
class.csc

SOM compiler

implementation
bindings
class.c class.ih

client bindings
class.h
class.def

published
interface
class.sc

language specific processing
C Compiler

SOM Object

A Programming Example -The C Bindings

Definition of a Class

- A relationship between a data template and a set of behaviors

Example of a class

- The animal class

What is an animal?

- an animal has a name
- an animal eats some kind of food
- an animal can tell you about itself

Defining the class animal

```
#include <somobj.sc>
class: animal;
parent: SOMObject;
data:
    char name[100];
    char food[100];
```

Defining the class animal

```
methods:
    void setName(char *newName);
    char *getName();
    void setFood(char *newFood);
    char *getFood();
    void print();
```

SOM compiler creates a method template file...

```
#define animal_Class_Source
#include "animal.ih"
#include <string.h>
#include <stdio.h>

#undef SOM_CurrentClass
#define SOM_CurrentClass \
animalCClassData.parentMtab
```

SOM compiler creates a method template file...

```
static void setName(animal *somSelf,
    char *newName)
{
    animal Data
    *somThis=animalGetData(somSelf);
    animalMethodDebug("animal",
    "setName");
}
```

SOM compiler creates a method template file...

```
static char *getName(animal *somSelf)
{
 animalData
 *somThis=animalGetData(somSelf);
 animalMethodDebug("animal",
 "getName");
}
/* etc. */
```

...which you fill in:

```
static void setName(animal *somSelf,
 char *newName)
{
    animalData
    *somThis=animalGetData(somSelf);
    animalMethodDebug("animal",
    "setName");
    strcpy(_name, newName);
}
```

...which you fill in:

```
static char *getName(animal *somSelf)
{
    animalData
    *somThis=animalGetData(somSelf);
    animalMethodDebug("animal",
    "getName");
    return _name;
}
```

...which you fill in:

```
static void print (animal *somSelf)
{
    animalData
     *somThis=animalGetData(somSelf);
     animalMethodDebug("animal",
     "print");
     printf("My name is: %s\n",
         _getName(somSelf));
     printf("My favorite food is: %s\n",
         _getFood(somSelf));
}
```

The animal client

```
#include "animal.h"
int main()

{

    animal *pooh;
    animal *bugs;


    pooh=animalNew();
    bugs=animalNew();


    _setName (pooh, "Pooh Bear");
    _setName (bugs, "Bugs Bunny);
```

The animal client

```
    _setFood(pooh, "Honey");
    _setFood(bugs, "Carrots");


    _print(pooh);
    _print(bugs);


    return 0;
}
```

Program output:

```
             My name is: Pooh Bear
My favorite food is: Honey
             My name is: Bugs Bunny
My favorite food is: Carrots
```

Inheritance (Class Derivation)

We have an animal class. Objects of type animal can

- be assigned a name
- be assigned a food

Consider writing a dog class. Objects of type dog can

- be assigned a name
- be assigned a food
- make a noise

We can derive a new class, dog, from an existing class, animal

Implementation of dog class

dog.csc:

```
#include "animal.sc"
class: dog;
parent: animal;
methods:
    void bark();
```

Implementation of dog class

dog.c:

```
static void bark (dog *somSelf)
{
    dogMethodDebug("dog","bark");
    printf("Unknown dog noise\n");
}
```

Client Code

```
#include "animal.h"
#include "dog.h"

int main ()

{
    animal *pooh;
    dog *snoopie;

    pooh = animalNew ();
    snoopie = dogNew ();
```

Client Code

```
_setName(pooh, "Pooh Bear");
_setName(snoopie, "Snoopie");

_setFood(pooh, "Honey");
_setFood(snoopie, "Dog Food");
_print(pooh);

_print(snoopie);
_bark(snoopie);

return 0; }
```

Client Output

```
            My name is: Pooh Bear
My favorite food is: Honey
           My name is: Snoopie
My favorite food is: Dog Food
Unknown dog noise
```

Polymorphism

- animal defines getName(), setName(), getFood(), setFood(), print()
- Dog is derived from anima, and adds one new method, bark ()
- LittleDog and bigDog are both derived from dog, and overrride the bark() method

bdog.csc

```
#include "dog.sc"
class: bigDog, local;
parent: dog;
methods:
   override bark;
```

bdog.c

```
static void bark(bigDog *somSelf)
{
    bigDogMethodDebug("bigDog", "bark");
    printf("Woof Woof\n");
    printf("Woof Woof\n");
    printf("Woof Woof\n");
    printf("Woof Woof\n");
    printf("Woof Woof\n");
}
```

ldog.csc

```
#include "dog.sc"
class: littleDog, local;
parent: dog;
methods:
    override bark;
```

ldog.csc

```
#include "dog.sc"
class: littleDog, local;
parent: dog;
methods:
    override bark;
```

ldog.c

```
static void bark(littleDog *somSelf)
{
    littleDogMethodDebug("littleDog","bark");
    printf("woof woof\n");
    printf("woof woof\n");
}
```

Client Code

```
#include "dog.h"
#include "bdog.h"
#include "ldog.h"

int main()

{

    dog *snoopie;
    littleDog *toto;
    bigDog *lassie;
```

Client Code

```
snoopie = dogNew();
toto = littledogNew();
lassie = bigdogNew();


_setName(snoopie, "Snoopie");
_setName(toto, "Toto");
_setName(lassie, "Lassie");


_setFood(snoopie, "Dog Food");
_setFood(toto, "LittleDog Food");
_setFood(lassie, "BigDog Food");
```

Client Code

```
    printDog(snoopie);
    printDog(toto);
    printDog(lassie);

    return 0
}
```

Client Code

```
void printDog(dog *thisDog)
{
    _print(thisDog);
    _bark(thisDog);
}
```

Client Output

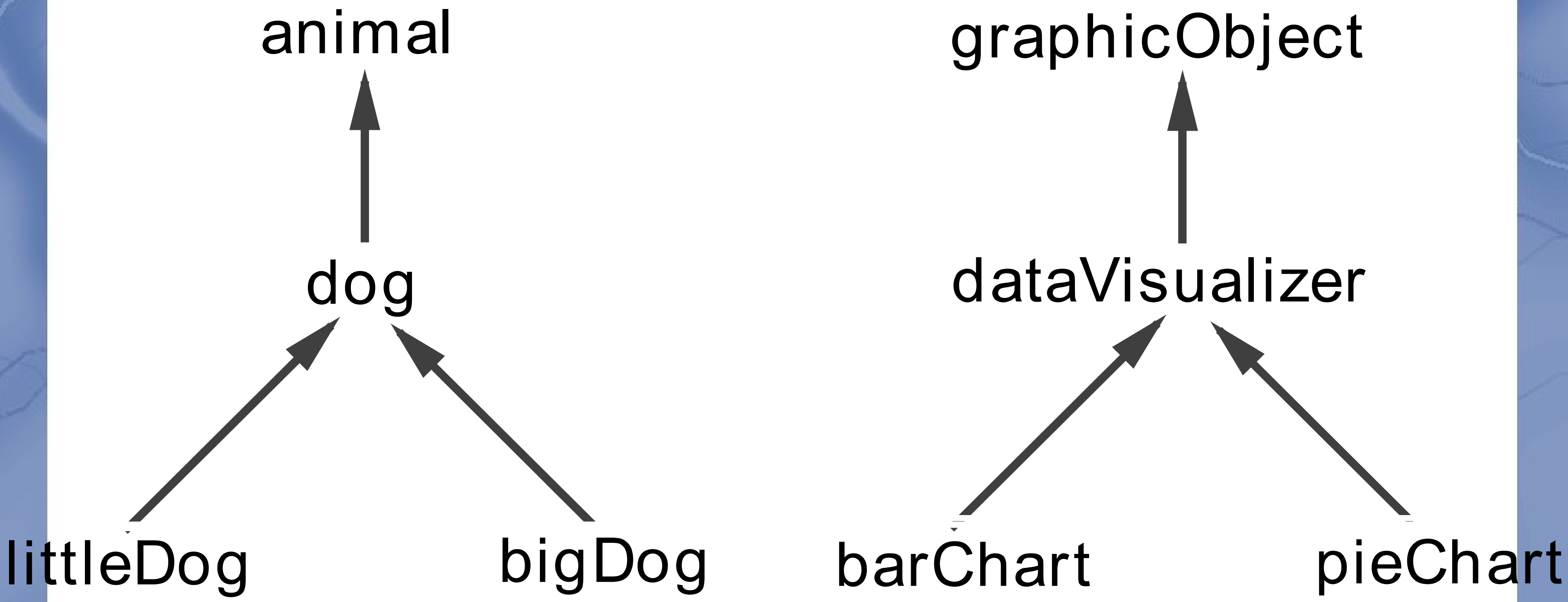
```
        My name is: Snoopie
My favorite food is: Dog Food
Unknown dog noise

        My name is: Toto
My favorite food is: LittleDog Food
woof woof
woof woof
```

Client Output

```
        My name is: Lassie
My favorite food is: BigDog Food
Woof Woof
Woof Woof
Woof Woof
Woof Woof
Woof Woof
```

Real Class Libraries



animal

dog

littleDog

bigDog

graphicObject

dataVisualizer

barChart

pieChart

SOM Design Goals

- Language Neutral
- State of the Art Object-Oriented Capability
- Support for Industrial Strength Class Libraries

SOM Design Goals

Language Neutral

Class libraries built with SOM can be defined and implemented in one language, and be used from another language.

The class client can...

- instantiate objects
- derive new classes
- override existing methods

Even from a non-object oriented language!!

SOM Design Goals

State of the Art Object-Oriented Capability

SOM Today:

- C Language Bindings
- Inheritance, Polymorphism, and Encapsulation
- Class Objects
- Programmer control of method dispatching

SOM Design Goals

State of the Art Object-Oriented Capability

SOM Development:

- Additional Language Bindings
- Multiple Inheritance
- True separation of class and type
- Extended OMG CORBA IDL
- Standardization with other vendors
- Platform Independence (AIX,...)

SOM Design Goals

State of the Art Object-Oriented Capability

Class Libraries in Development:

- Persistent Objects
- Distributed Objects (Simple and Replicated)
- User Interface Frameworks

SOM Design Goals

State of the Art Object-Oriented Capability

Existing technology does not support industrial grade class library products.

Characteristics of industrial grade class libraries:

- Ship without source code
- Upward binary compatibility
- Support shared libraries

Industrial grade class libraries

Ship without source code

A Shipped C++ Product:

```
class linkedList {
private:
    link *currentLink;
    link *headLink;
    void moveLink (link *target,link*object);
public:
    void setHead();
    void setTail(); ... }
```

Industrial grade class libraries

Ship without source code

A Shipped SOM Product:

```
class: linkedList;
methods:
    void setHead();
    void setTail();
    . . .
```

Industrial grade class libraries

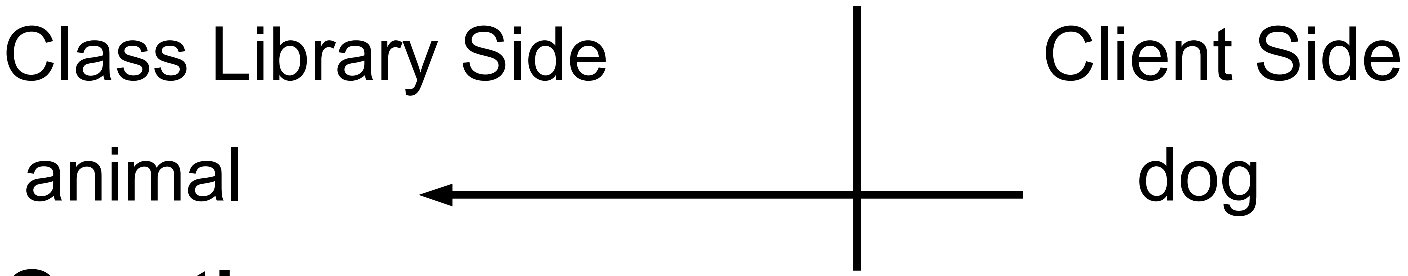
Ship without source code

A Shipped C++ Product:

```
void moveLink(link *target, link *object)
    . . .
void setHead()
    . . .
void setTail()
    . . .
```

Industrial grade class libraries

Upward binary compatibility:



Class Library Side

Client Side

animal

dog

Question:

If you change animal, does dog need recompiling?

Answer:

C++: Yes

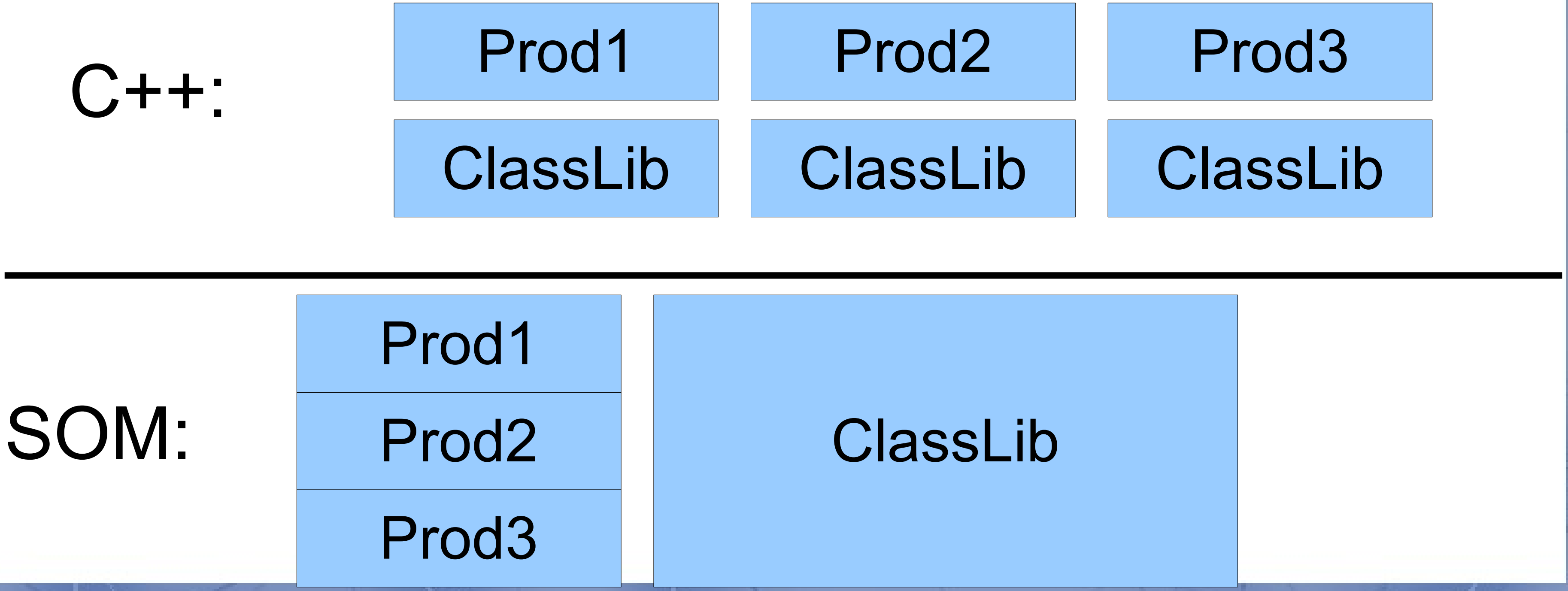
SOM: No

Industrial grade class libraries

Support shared libraries:

Prod1 from Vendor1 uses ClassLib
Prod2 from Vendor2 uses ClassLib
Prod3 from Vendor3 uses ClassLib

Memory Usage



C++:

Prod1 Prod2 Prod3

ClassLib ClassLib ClassLib

SOM:

Prod1
Prod2
Prod3

ClassLib

Important SOM Features

	Library Producer	Library Client
Language Neutral	☑	☑
State of the Art OO	☑	
Industrial Strength Libraries	☑	☑
Ship without source	☑	
Upward Binary Compatibility	☑	☑
Shared Libraries		☑

References

- Introduction to SOM
- Object-Oriented Programming in OS/2 2.0, by Roger Sessions and Nurcan Coskun. IBM Personal Systems Developer, Winter, 1992
- Class Objects in SOM, by Nurcan Coskun and Roger Sessions. IBM OS/2 Developer, Summer, 1992
- OS/2 2.0 Technical Library System Object Model Guide and Reference, IBM Doc S10G6309

References

Object-Oriented Programming and C++

- Class Construction in C and C++ - Object-Oriented Programming Fundamentals, by Roger Sessions. Prentice Hall, Englewood Cliffs, New Jersey, 1992. IBM Doc S242-0086