

Large-Scale C++: Advanced *Levelization* Techniques

John Lakos

Wednesday, May 13, 2015

This version is for C++Now'15, Aspen CO.

Copyright Notice

© 2014 Bloomberg L.P. Permission is granted to copy, distribute, and display this material, and to make derivative works and commercial use of it. The information in this material is provided "AS IS", without warranty of any kind. Neither Bloomberg nor any employee guarantees the correctness or completeness of such information. Bloomberg, its employees, and its affiliated entities and persons shall not be liable, directly or indirectly, in any way, for any inaccuracies, errors or omissions in such information. Nothing herein should be interpreted as stating the opinions, policies, recommendations, or positions of Bloomberg.

Abstract

Developing a large-scale software system in C++ requires more than just a sound understanding of the logical design issues covered in most books on C++ programming. To be successful, one also needs a grasp of physical design concepts that, while closely tied to the technical aspects of development, include a dimension with which even expert software developers may have little or no experience.

In this talk we begin by briefly reviewing the basics of physical design. We then present a variety of *levelization* and *insulation* techniques, and apply them in present-day, real-word examples to avoid cyclic, excessive, or otherwise inappropriate physical dependencies. Along the way, we comment on how to make the best use of what the C++ language has to offer.

What's The Problem?

What's The Problem?

Large-Scale C++ Software Design is Multi-Dimensional:

What's The Problem?

Large-Scale C++ Software Design is Multi-Dimensional:

- It involves many subtle *logical* and *physical* aspects.

What's The Problem?

Large-Scale C++ Software Design is Multi-Dimensional:

- It involves many subtle logical and physical aspects.
- It requires an ability to isolate and modularize **functionality** within discrete, fine-grained **physical components**.

What's The Problem?

Large-Scale C++ Software Design is Multi-Dimensional:

- It involves many subtle *logical* and *physical* aspects.
- It requires an ability to isolate and modularize logical functionality within discrete, fine-grained physical components.
- It requires the designer to delineate **logical behavior** precisely, while managing the **physical dependencies** on other subordinate components.

What's The Problem?

Large-Scale C++ Software Design is Multi-Dimensional:

- It involves many subtle *logical* and *physical* aspects.
- It requires an ability to isolate and modularize logical functionality within discrete, fine-grained physical components.
- It requires the designer to delineate logical behavior precisely, while managing the physical dependencies on other subordinate components.
- It requires attention to numerous **logical** and **physical** rules that govern sound software design.

Purpose of this Talk

Purpose of this Talk

1. Review the basics of component-based design:

Purpose of this Talk

1. Review the basics of component-based design:

- Component Properties and Logical Diagrams
- Implied Dependency and Level Numbers
- The Two Most Important Physical Design Rules
- Guidelines for Collocating Classes in a Component
- Our Three-Level Physical-Packaging Hierarchy
- Logical *Encapsulation* Versus Physical *Insulation*
- When to #include a Header File in a Header

Purpose of this Talk

1. Review the basics of component-based design:
 - Component Properties and Logical Diagrams
 - Implied Dependency and Level Numbers
 - The Two Most Important Physical Design Rules
 - Guidelines for Collocating Classes in a Component
 - Our Three-Level Physical-Packaging Hierarchy
 - Logical *Encapsulation* Versus Physical *Insulation*
 - When to `#include` a Header File in a Header
2. Introduce a variety of *levelization* and *insulation* techniques, and then apply them to present-day real-world examples to avoid cyclic, excessive, or otherwise inappropriate *physical dependencies*.

Outline

1. Review of Elementary Physical Design
Components, Modularity, Physical Dependencies
2. Survey of Advanced *Levelization* Techniques
Avoiding Cyclic, Excessive, or Inappropriate Dependencies
3. Total and Partial *Insulation* Techniques
Avoiding Unnecessary Compile-Time Coupling
4. Present-Day, Real-World Design Examples
Prioritizing Physical Design to Improve Software Quality

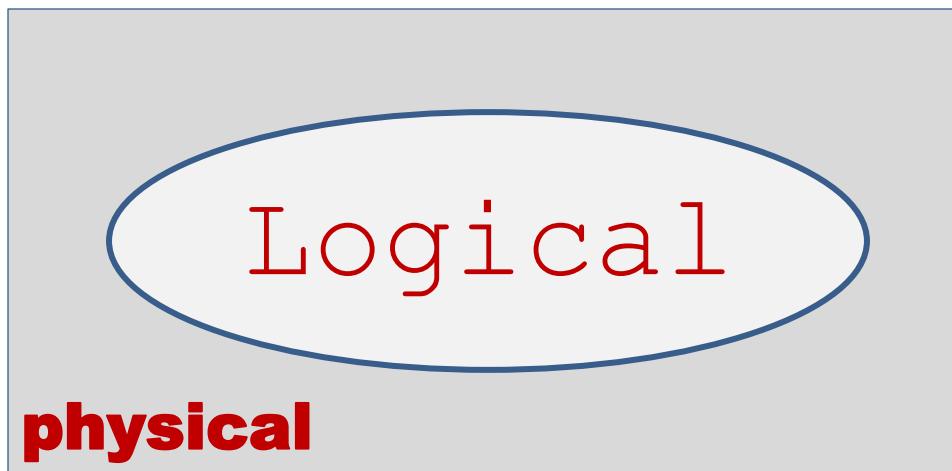
Outline

1. Review of Elementary Physical Design
Components, Modularity, Physical Dependencies
2. Survey of Advanced *Levelization* Techniques
Avoiding Cyclic, Excessive, or Inappropriate Dependencies
3. Total and Partial *Insulation* Techniques
Avoiding Unnecessary Compile-Time Coupling
4. Present-Day, Real-World Design Examples
Prioritizing Physical Design to Improve Software Quality

1. Review of Elementary Physical Design

Logical versus Physical Design

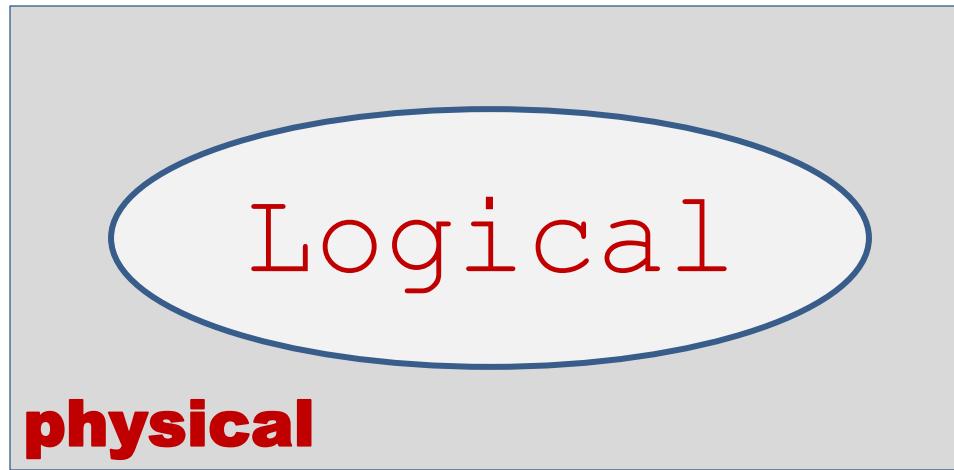
What distinguishes *Logical* from *Physical* Design?



1. Review of Elementary Physical Design

Logical versus Physical Design

What distinguishes *Logical* from *Physical* Design?

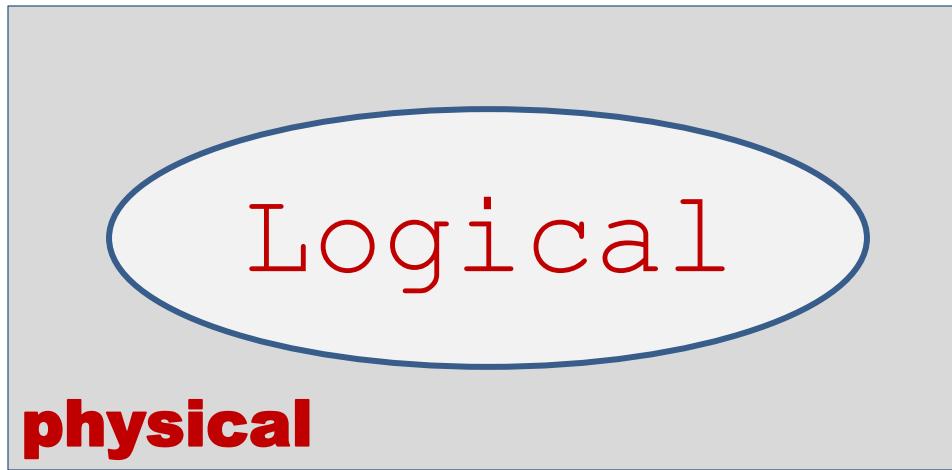


Logical: Classes and Functions

1. Review of Elementary Physical Design

Logical versus Physical Design

What distinguishes *Logical* from *Physical* Design?



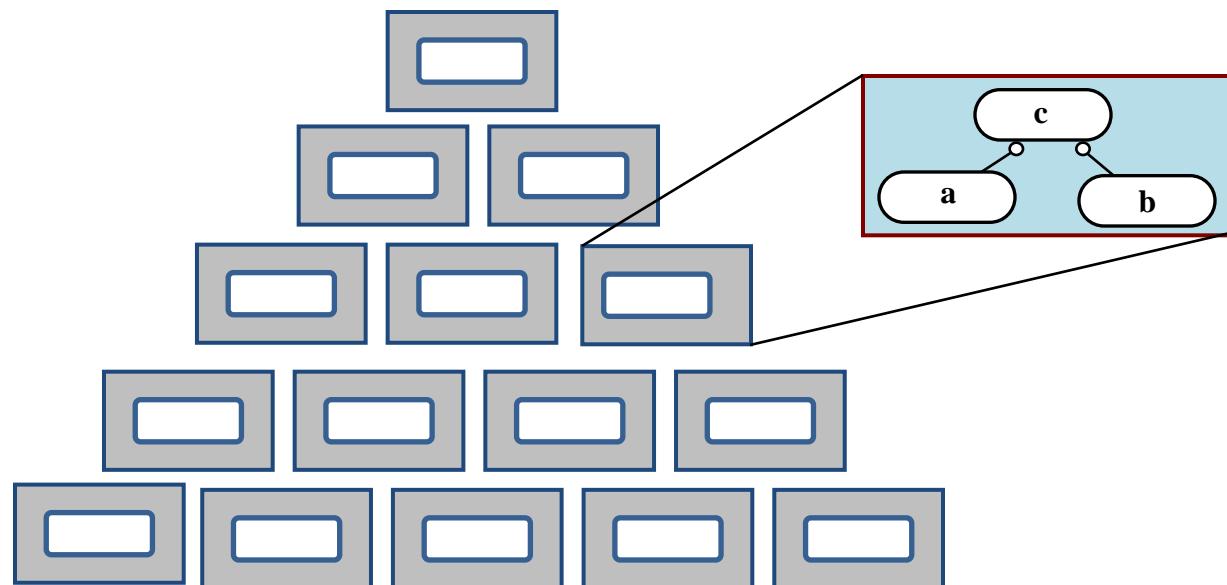
Logical: Classes and Functions

Physical: Files and Libraries

1. Review of Elementary Physical Design

Logical versus Physical Design

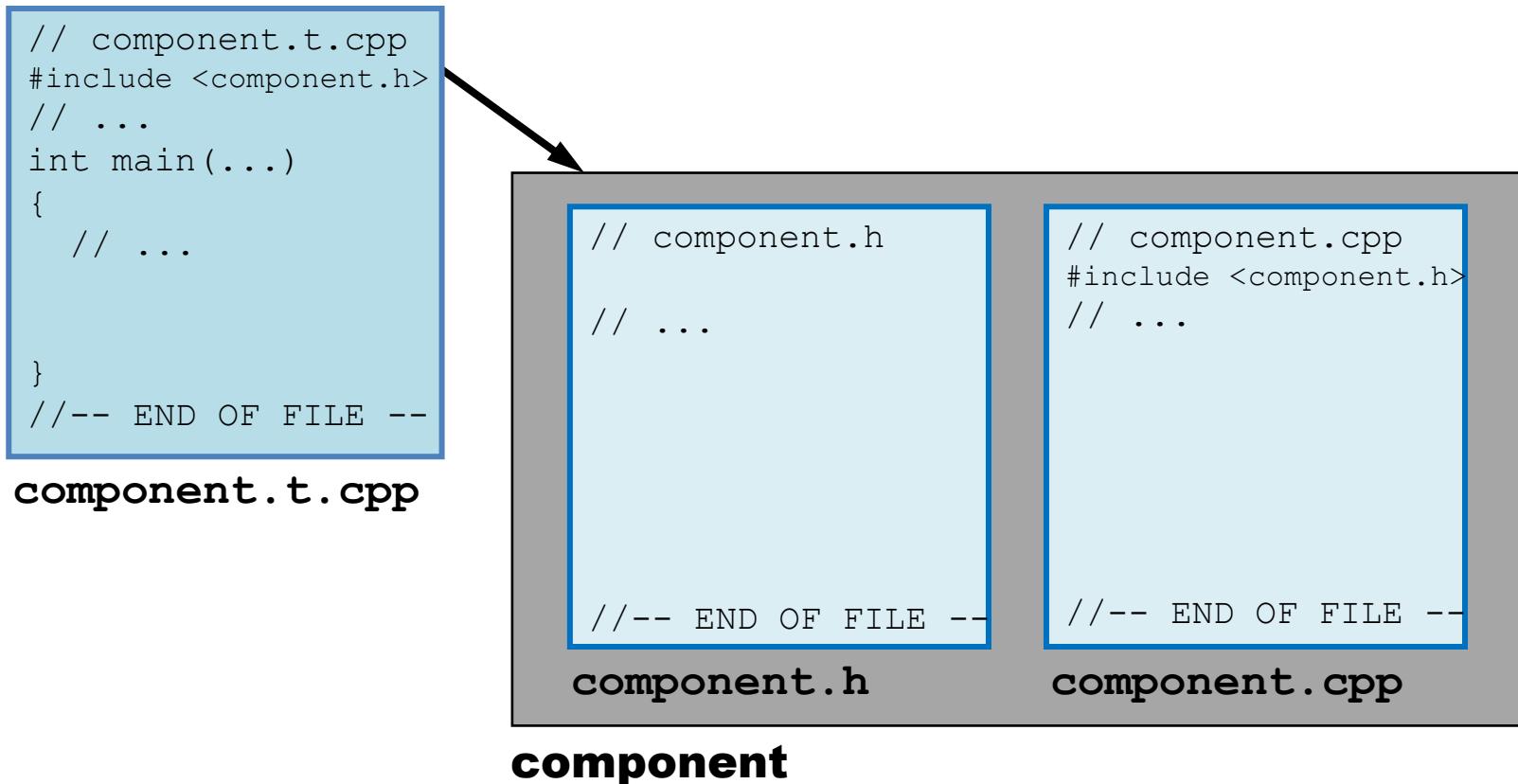
Logical content aggregated into a
Physical hierarchy of **components**



1. Review of Elementary Physical Design

Component: Uniform Physical Structure

A Component Is Physical



1. Review of Elementary Physical Design

Component: Uniform Physical Structure

Implementation

```
// component.t.cpp
#include <component.h>
// ...
int main(...)

{
    // ...
}

//-- END OF FILE --
```

component.t.cpp

```
// component.h
```

```
// ...
```

```
//-- END OF FILE --
```

component.h

```
// component.cpp
#include <component.h>
// ...
```

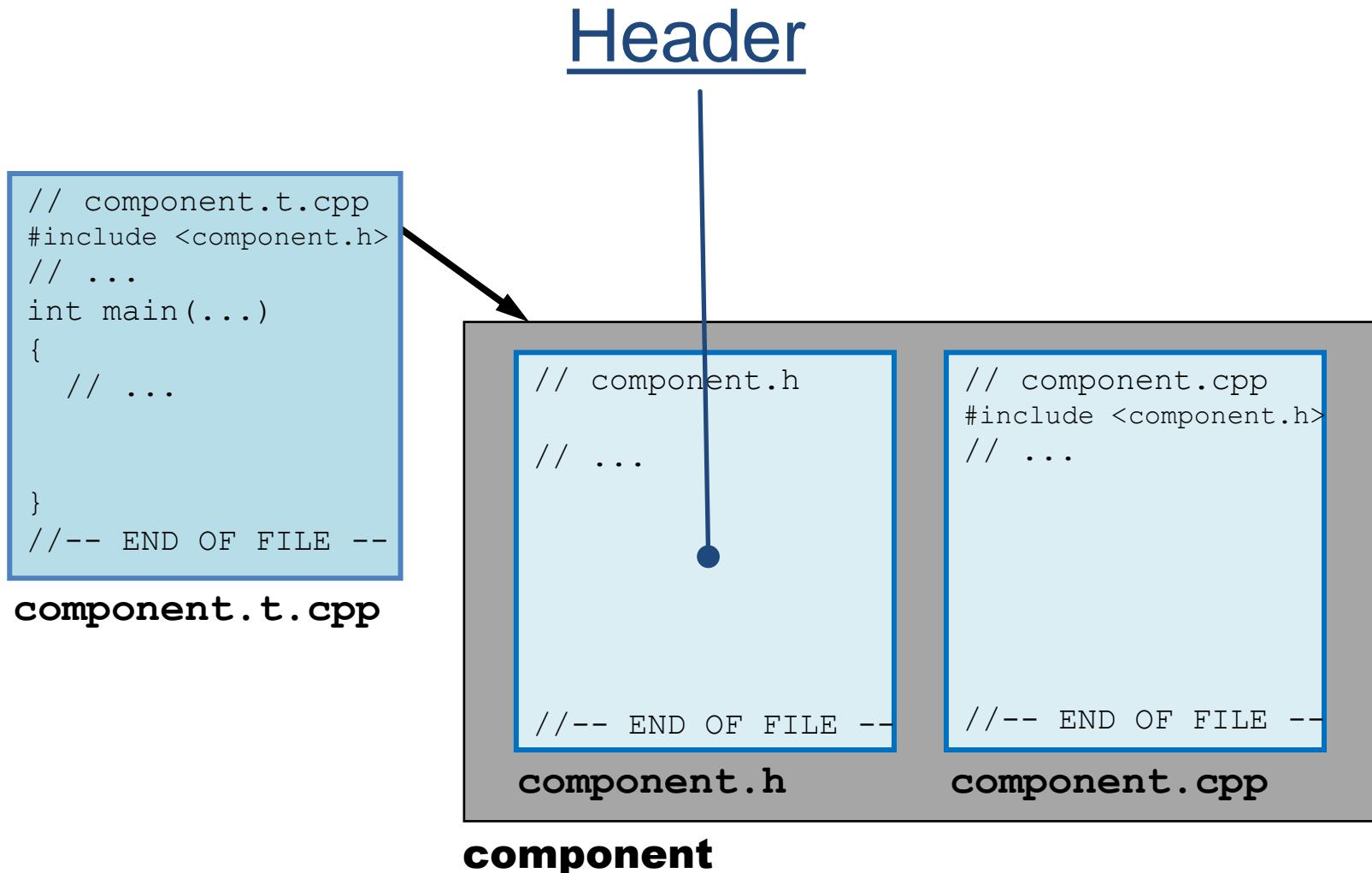
```
//-- END OF FILE --
```

component.cpp

component

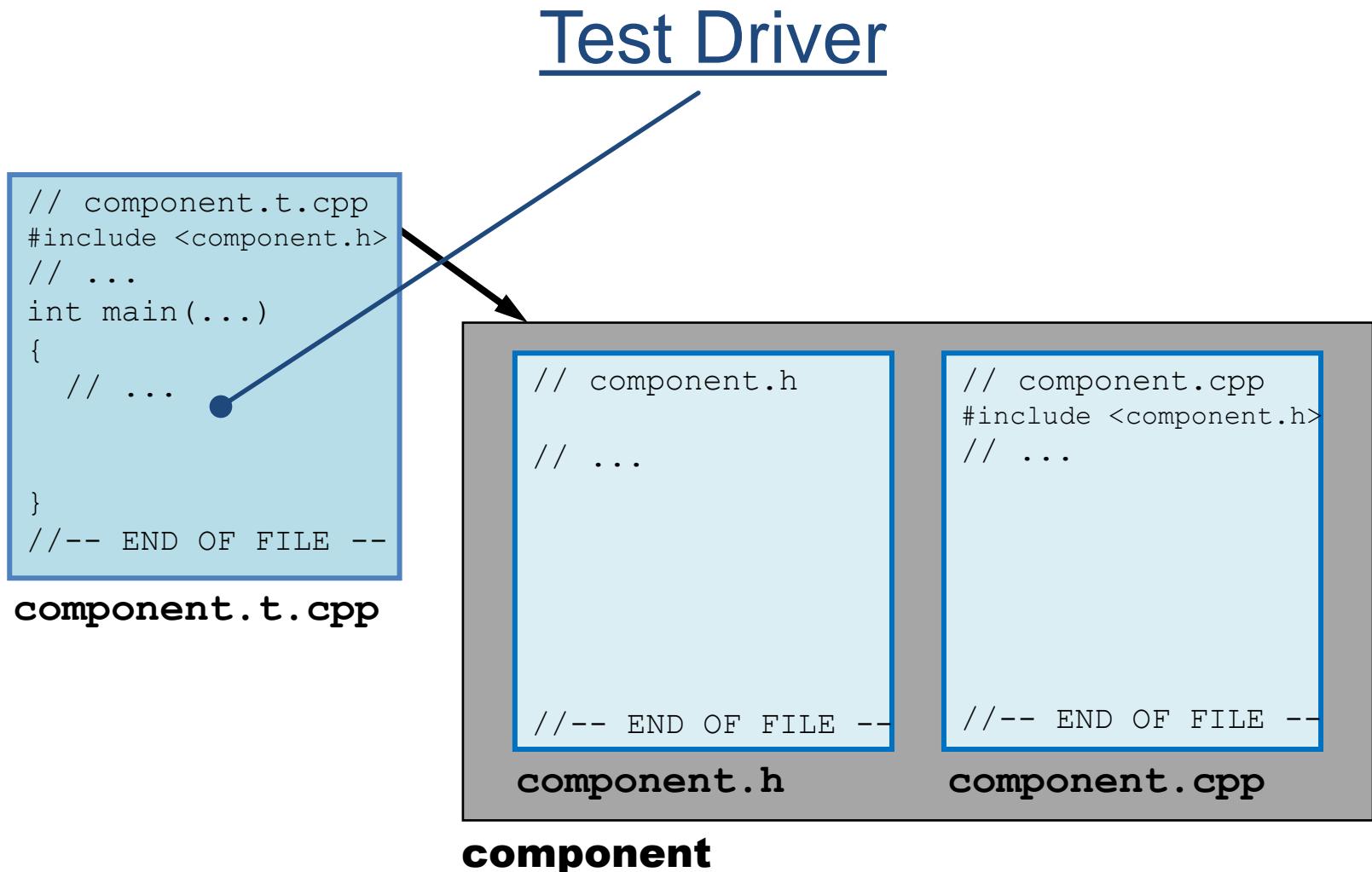
1. Review of Elementary Physical Design

Component: Uniform Physical Structure



1. Review of Elementary Physical Design

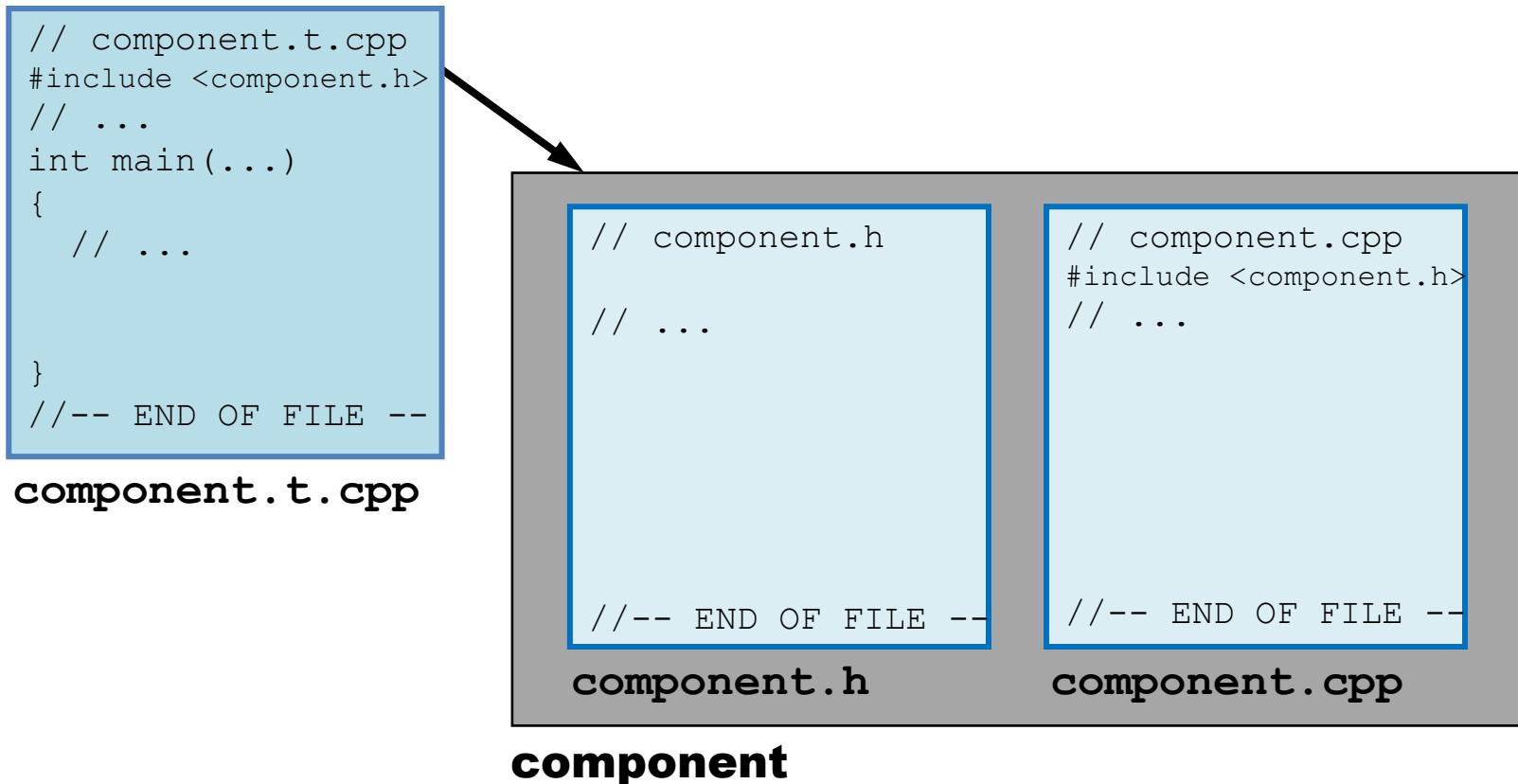
Component: Uniform Physical Structure



1. Review of Elementary Physical Design

Component: Uniform Physical Structure

The Fundamental Unit of Design



1. Review of Elementary Physical Design

Component: Not Just a .h/.cpp Pair



my::Widget

my_widget

1. Review of Elementary Physical Design

Component: Not Just a .h/.cpp Pair

There are four Properties...

1. Review of Elementary Physical Design

Component: Not Just a .h/.cpp Pair

1.  The **.cpp** file includes its **.h** file as the first substantive line of code.

**EVEN IF THE .CPP IS
OTHERWISE EMPTY!**

1. Review of Elementary Physical Design

Component: Not Just a .h/.cpp Pair

1.  The `.cpp` file includes its `.h` file as the first substantive line of code.
2.  All logical constructs having external *linkage* defined in a `.cpp` file are declared in the corresponding `.h` file.

1. Review of Elementary Physical Design

Component: Not Just a .h/.cpp Pair

1.  The **.cpp** file includes its **.h** file as the first substantive line of code.
2.  All logical constructs having external *linkage* defined in a **.cpp** file are declared in the corresponding **.h** file.
3.  All constructs having external or dual *bindage* declared in a **.h** file (if defined at all) are defined within the component.

1. Review of Elementary Physical Design

Component: Not Just a .h/.cpp Pair

1.  The **.cpp** file includes its **.h** file as the first substantive line of code.
2.  All logical constructs having external *linkage* defined in a **.cpp** file are declared in the corresponding **.h** file.
3.  All constructs having external or dual *bindage* declared in a **.h** file (if defined at all) are defined within the component.
4.  A component's functionality is accessed via a **#include** of its header, and never via a forward (**extern**) declaration.

1. Review of Elementary Physical Design

Logical Relationships

PointList

PointList_Link

Polygon

Point

Shape

1. Review of Elementary Physical Design

Logical Relationships

PointList

PointList_Link

Polygon

Point

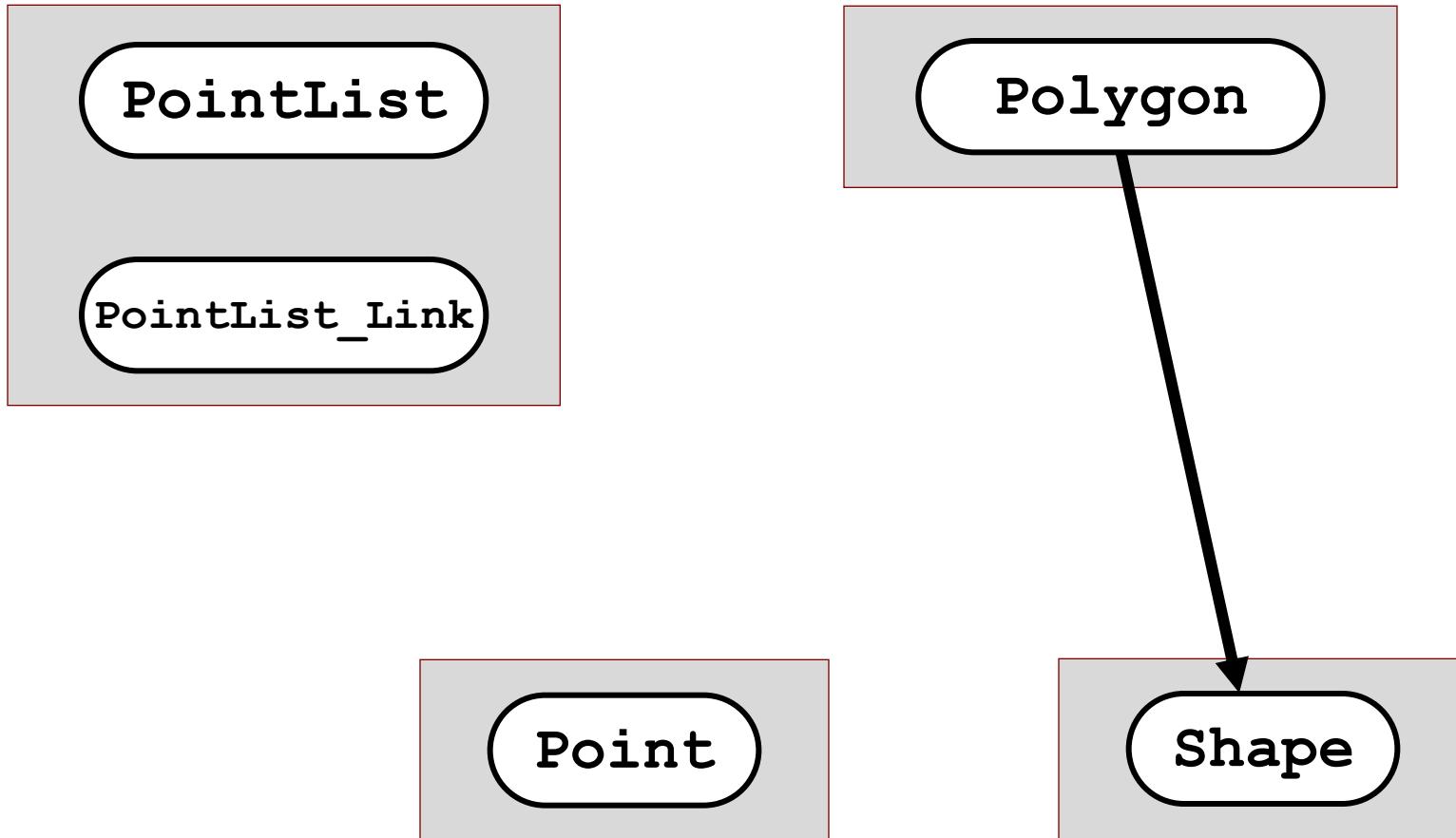
Shape



Is-A

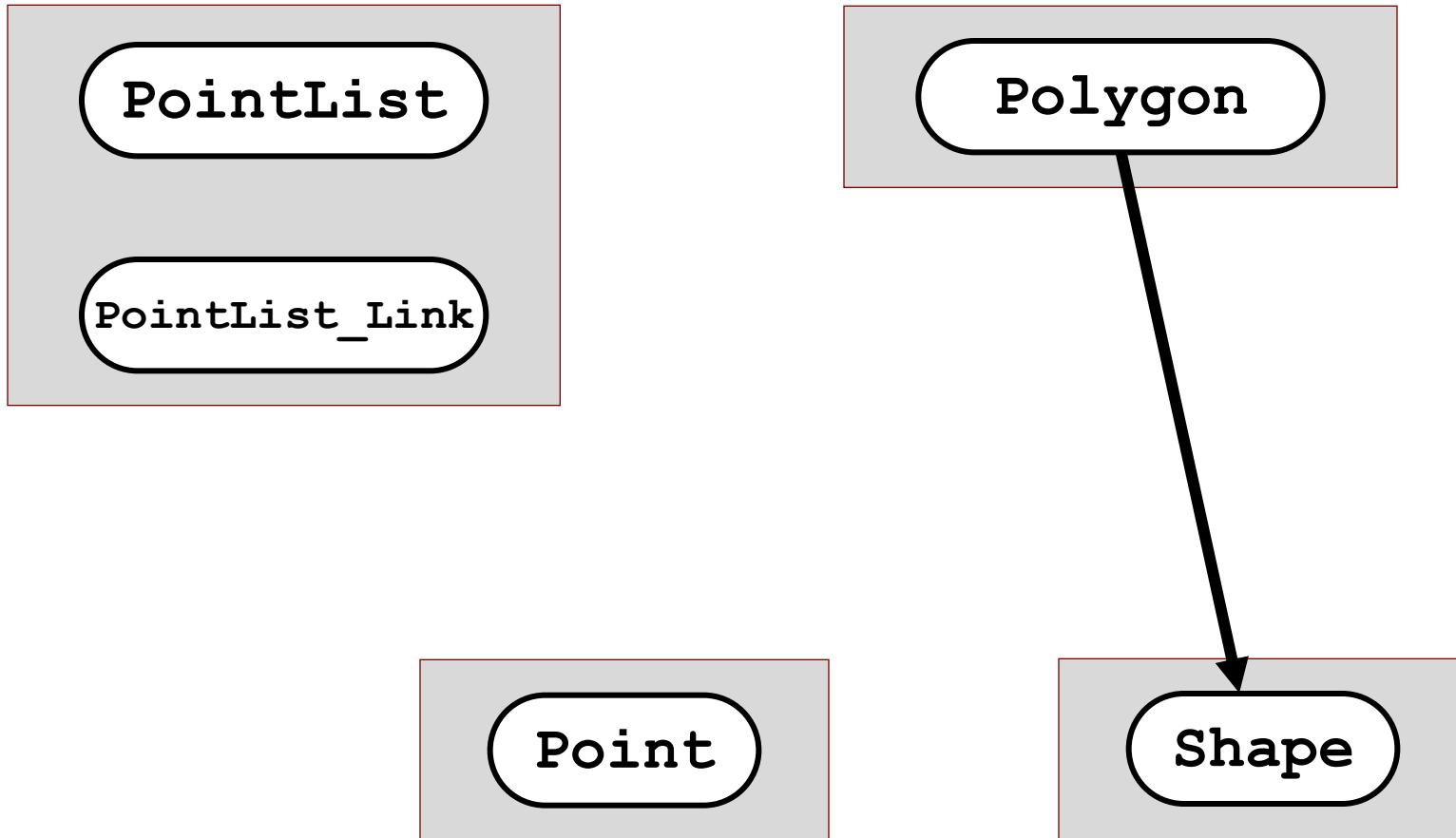
1. Review of Elementary Physical Design

Logical Relationships



1. Review of Elementary Physical Design

Logical Relationships



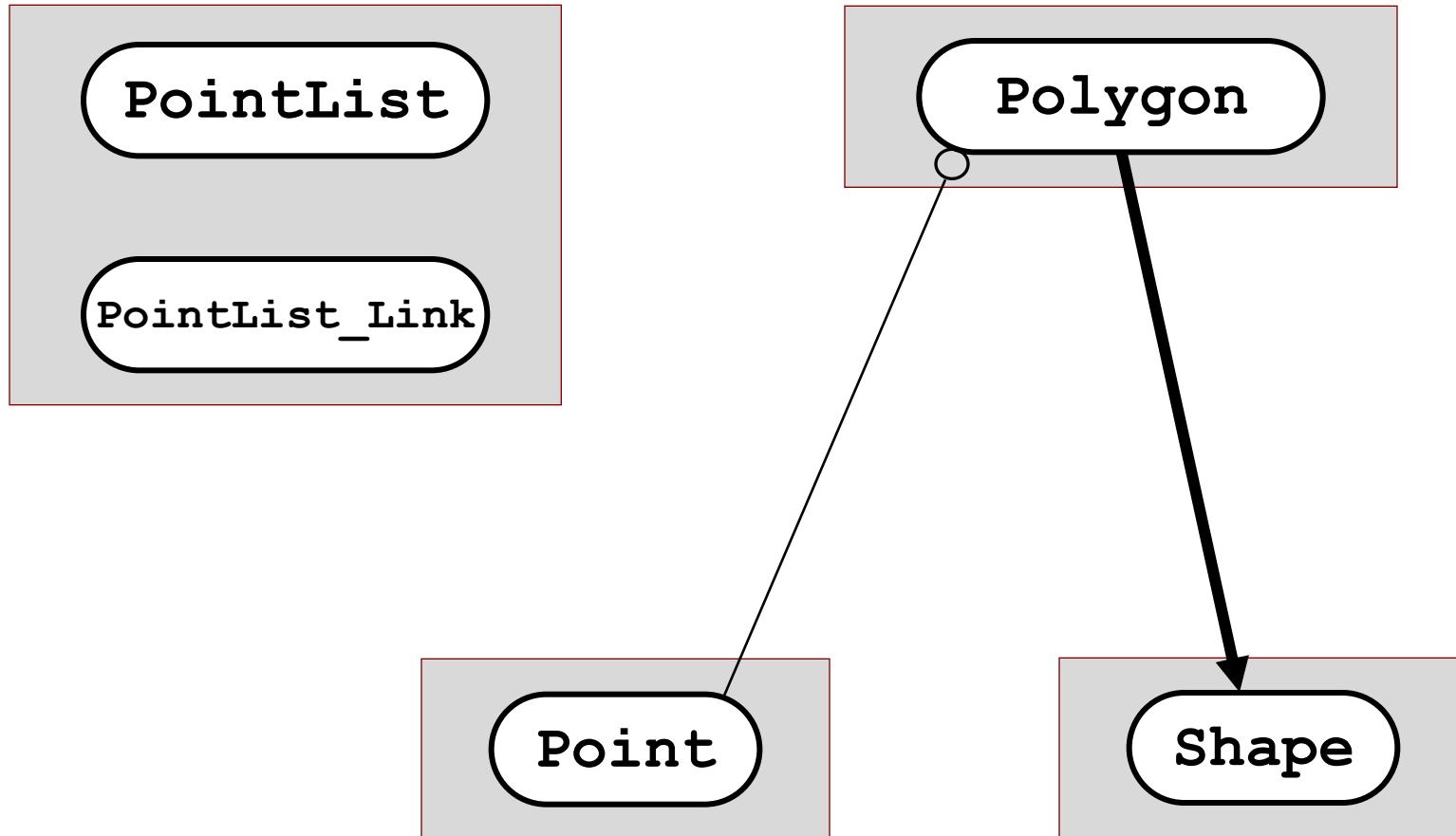
Uses-in-the-Interface



Is-A

1. Review of Elementary Physical Design

Logical Relationships



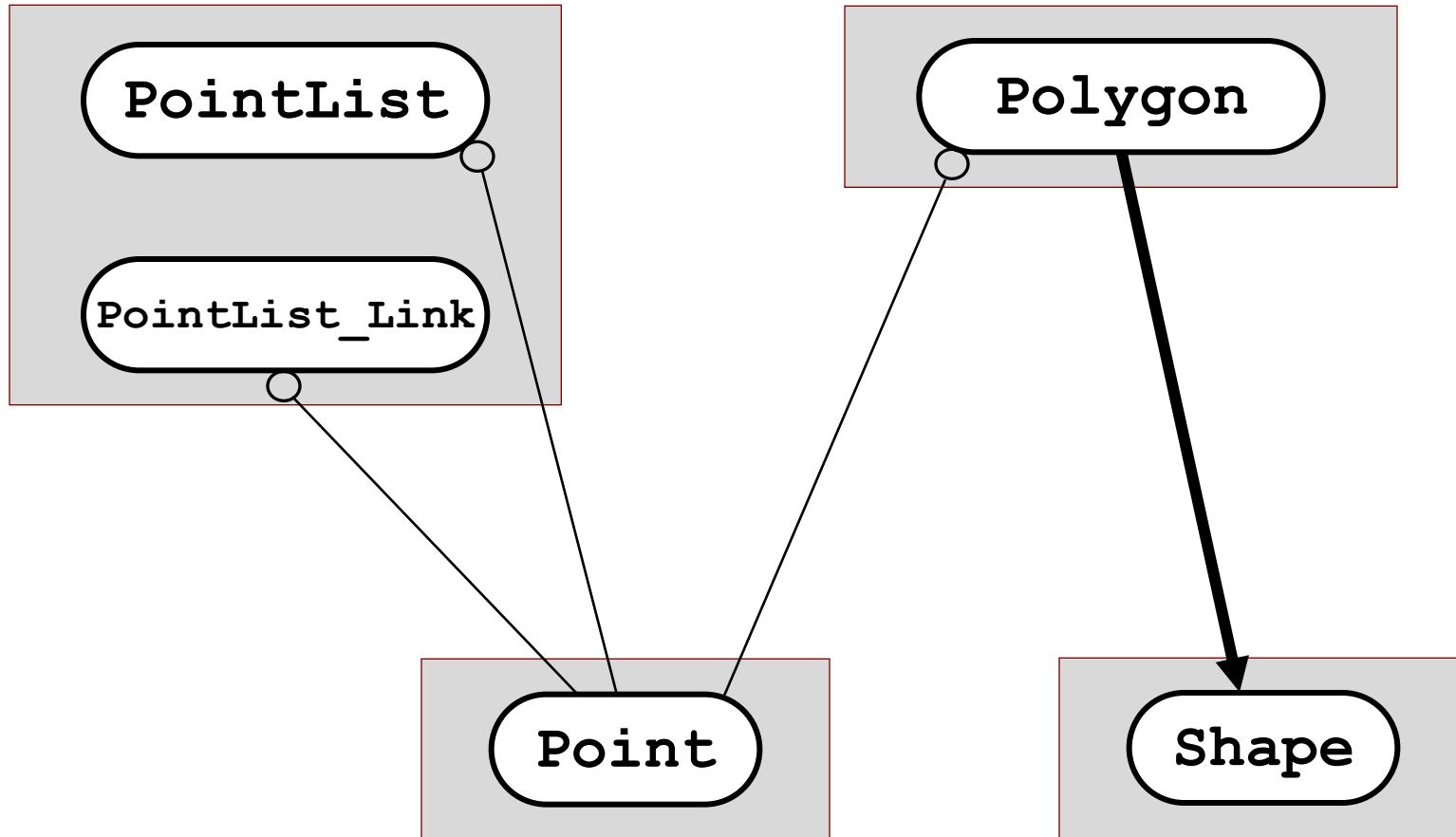
Uses-in-the-Interface



Is-A

1. Review of Elementary Physical Design

Logical Relationships



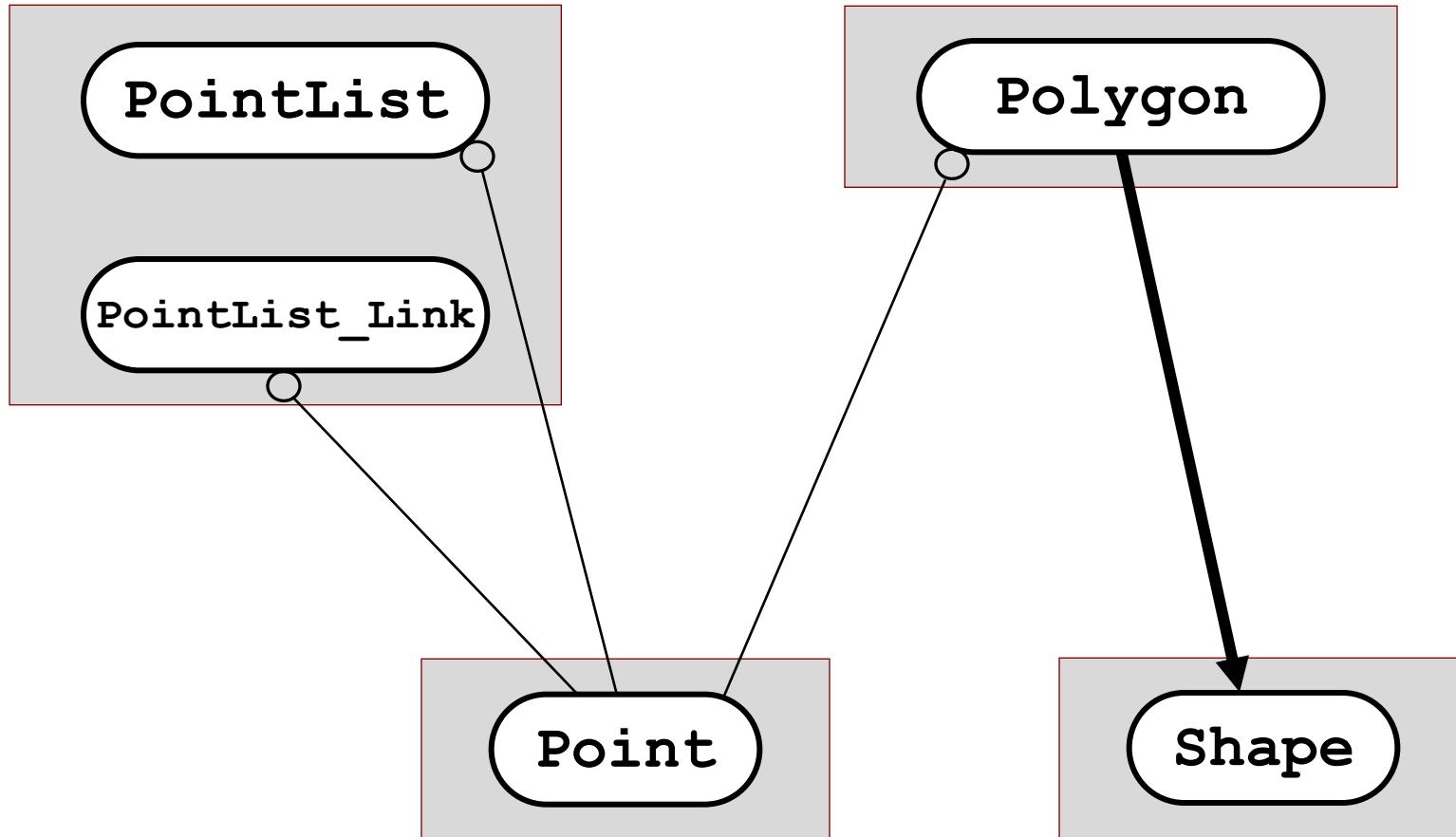
Uses-in-the-Interface



Is-A

1. Review of Elementary Physical Design

Logical Relationships



○ —

Uses-in-the-Interface

● —

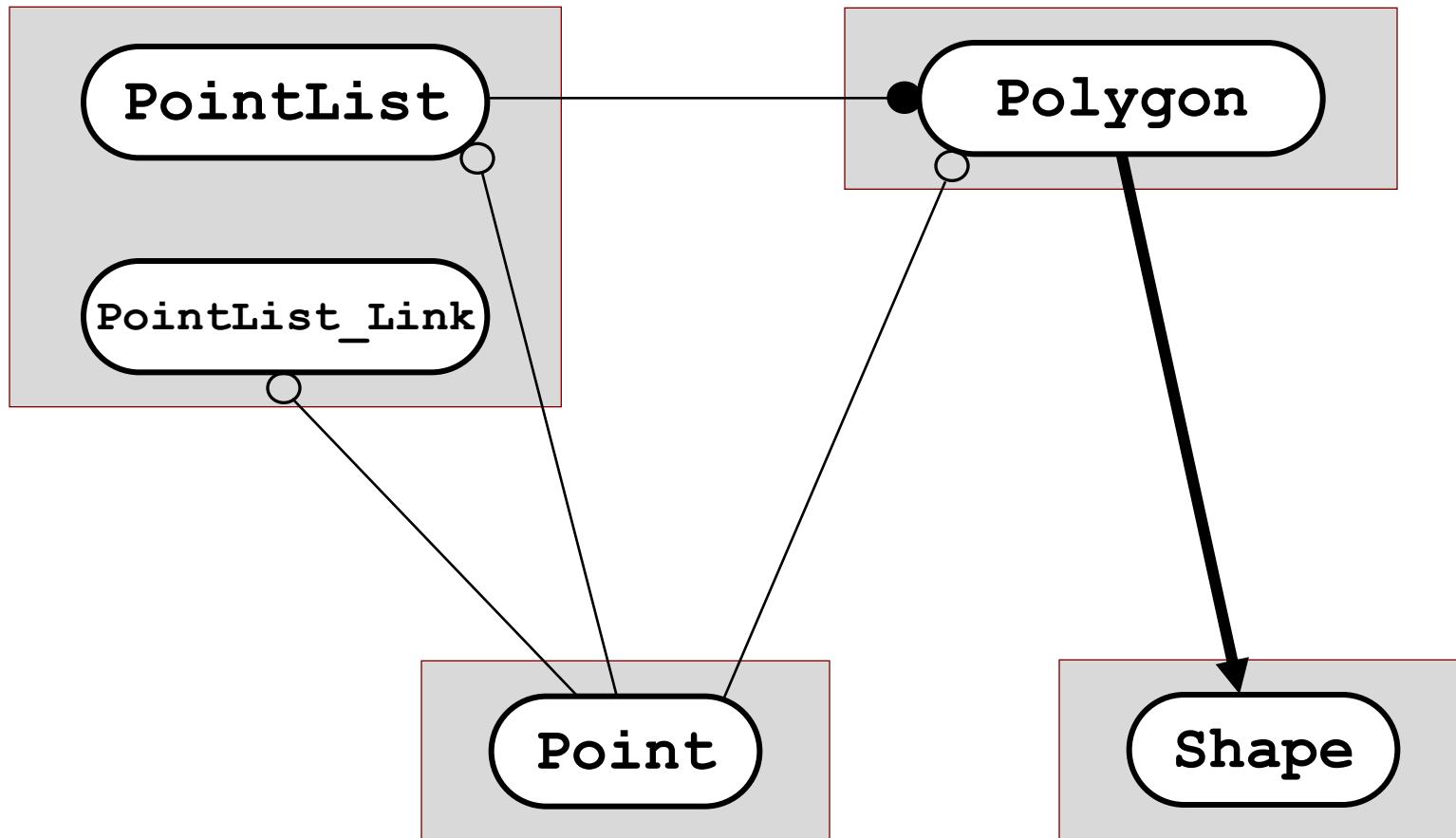
Uses-in-the-Implementation

→

Is-A

1. Review of Elementary Physical Design

Logical Relationships

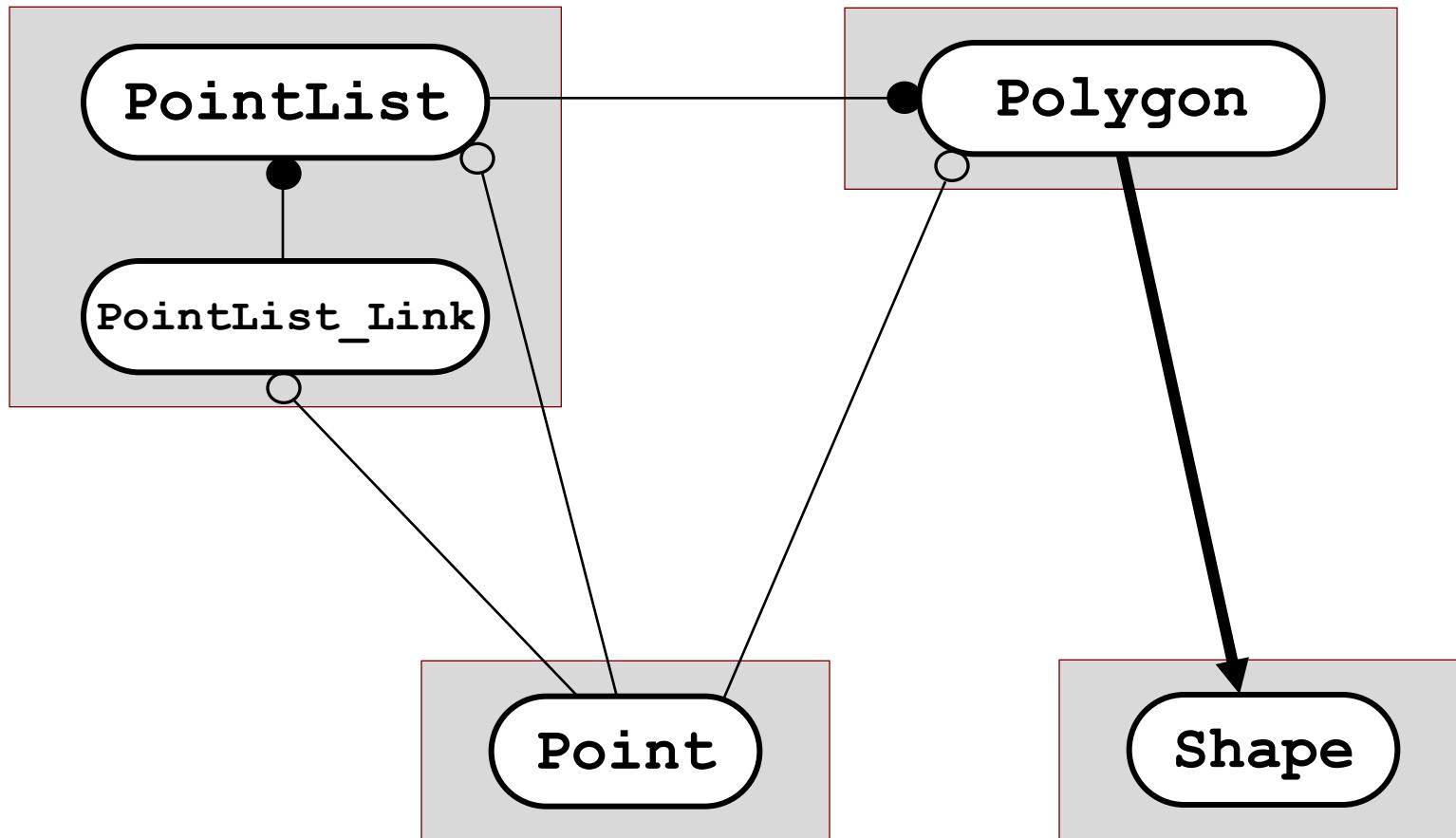


○ ————— Uses-in-the-Interface
● ————— Uses-in-the-Implementation

—————> Is-A

1. Review of Elementary Physical Design

Logical Relationships



Uses-in-the-Interface



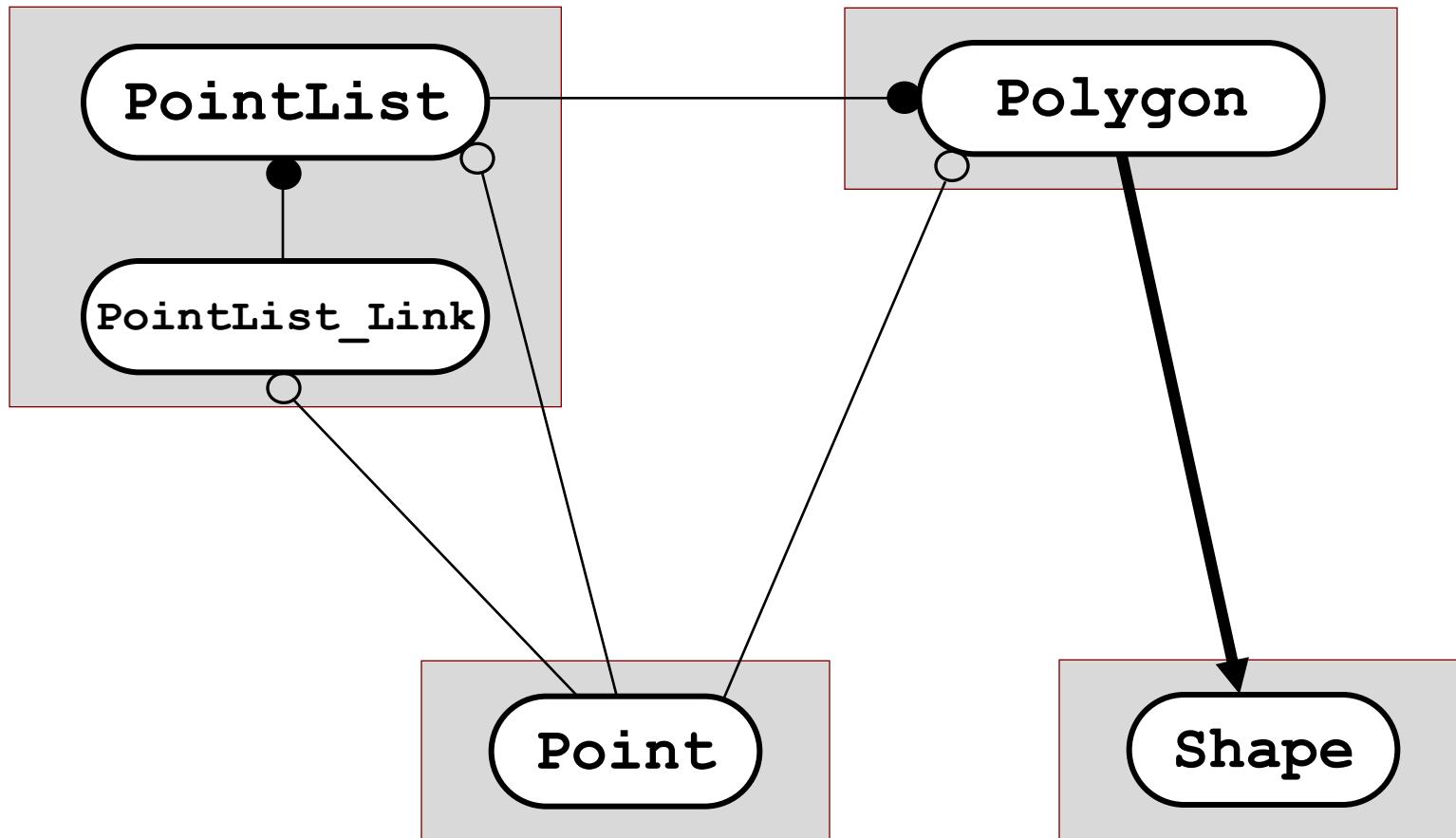
Uses-in-the-Implementation



Is-A

1. Review of Elementary Physical Design

Logical Relationships

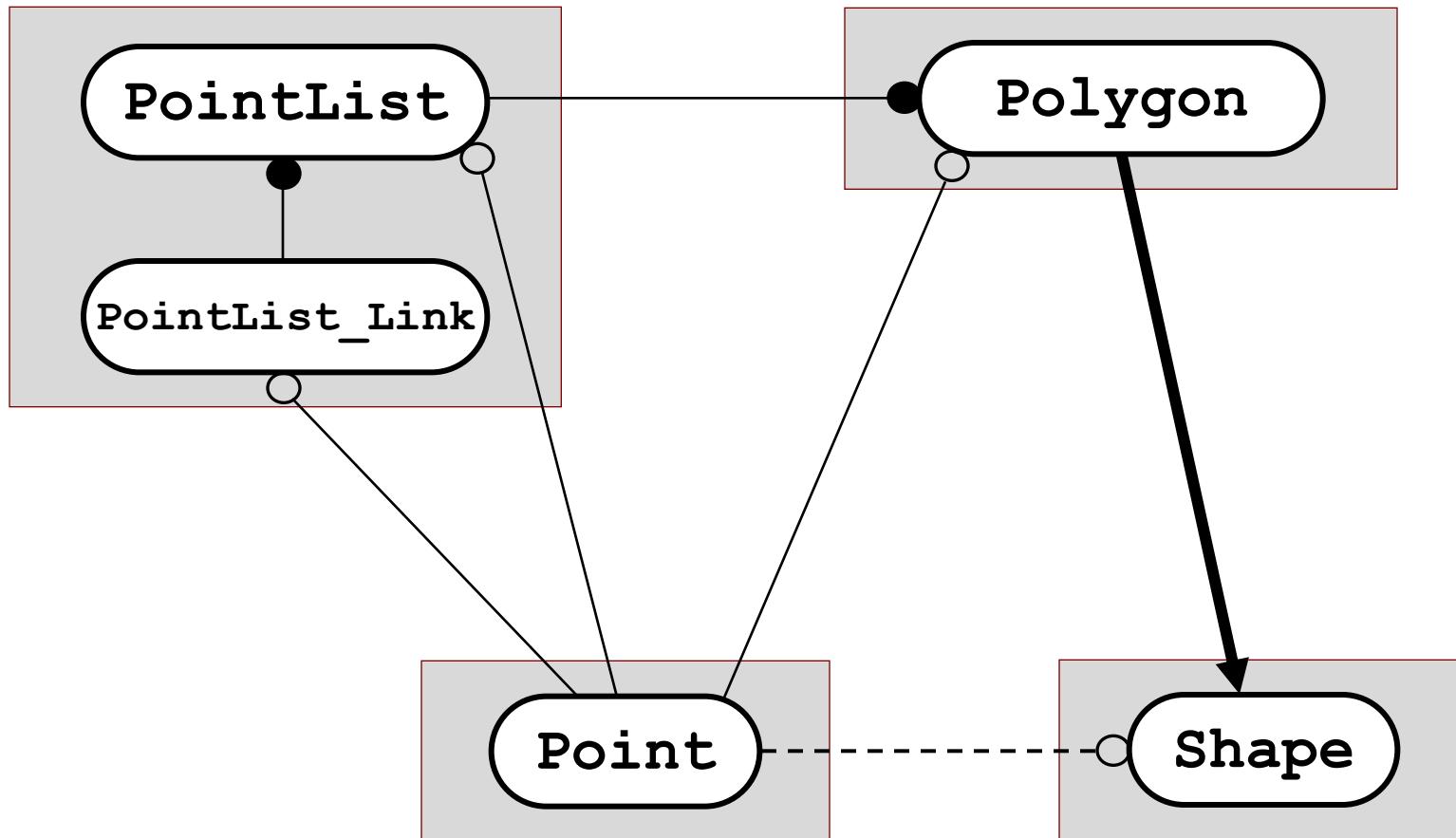


○— Uses-in-the-Interface
●— Uses-in-the-Implementation

○----- Uses in name only
→ Is-A

1. Review of Elementary Physical Design

Logical Relationships

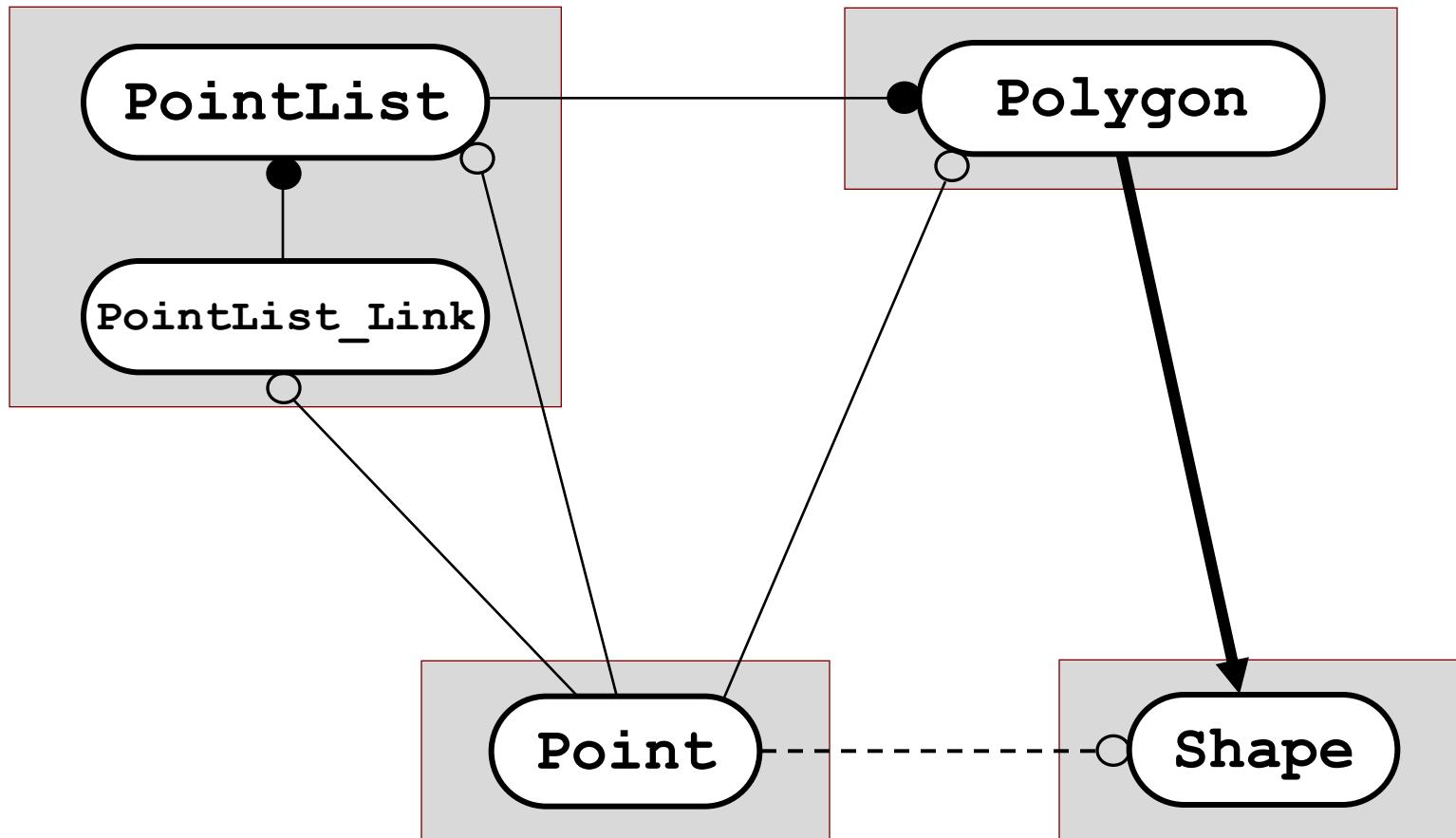


○— Uses-in-the-Interface
●— Uses-in-the-Implementation

○----- Uses in name only
→ Is-A

1. Review of Elementary Physical Design

Implied Dependency

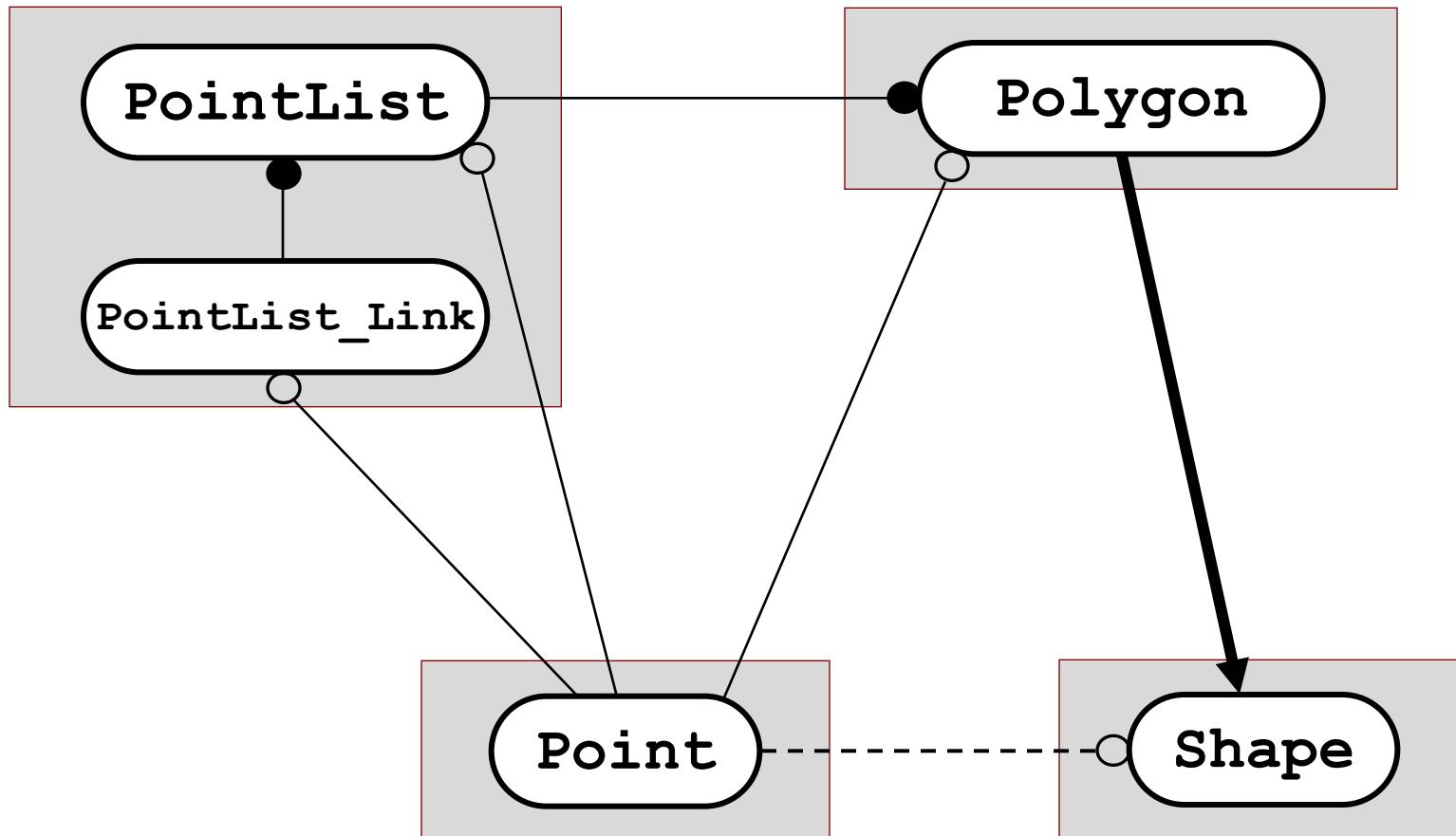


○— Uses-in-the-Interface
●— Uses-in-the-Implementation

○----- Uses in name only
→ Is-A

1. Review of Elementary Physical Design

Implied Dependency

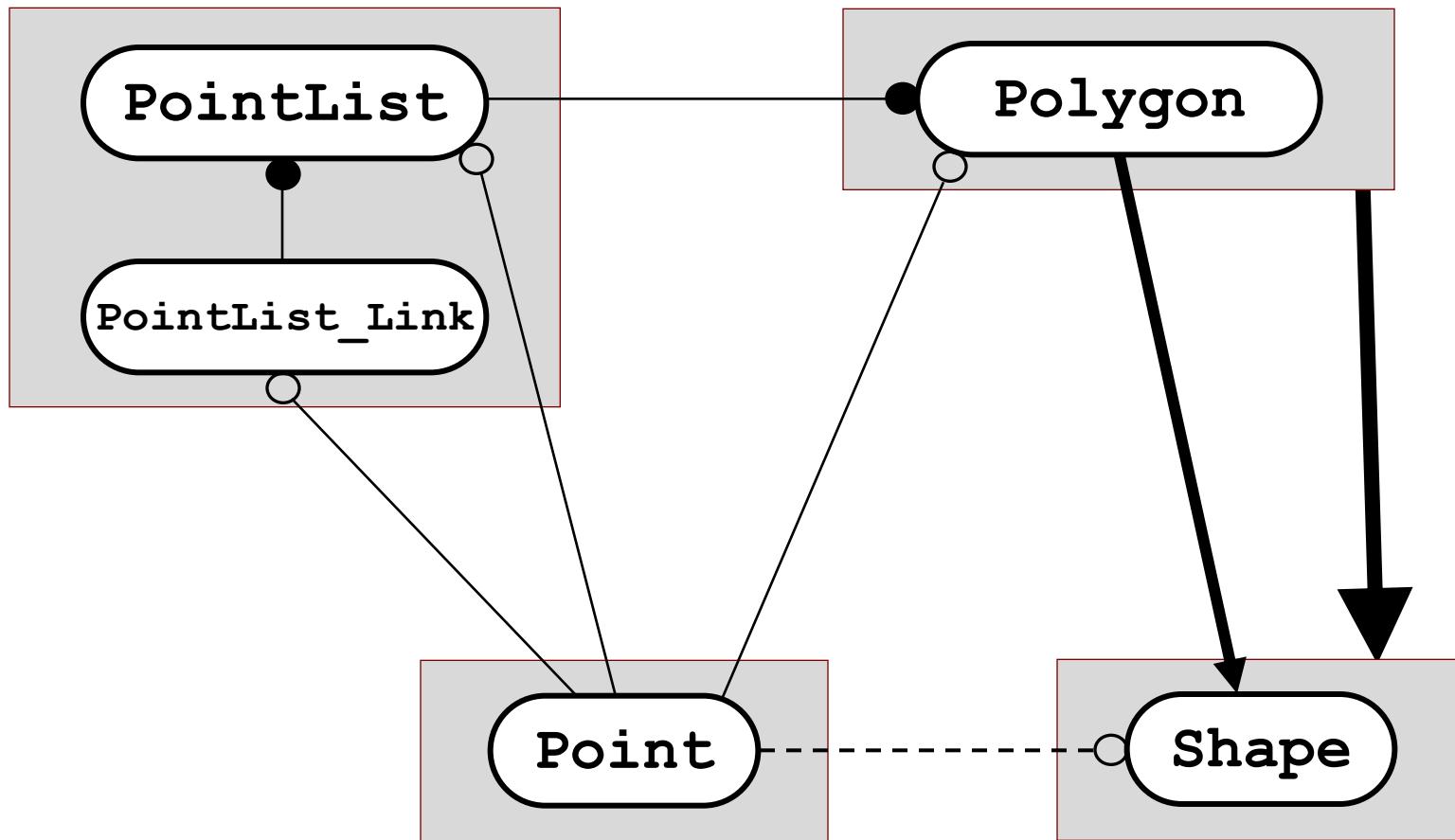


○— Uses-in-the-Interface
●— Uses-in-the-Implementation

→ Depends-On
○--- Uses in name only
→ Is-A

1. Review of Elementary Physical Design

Implied Dependency

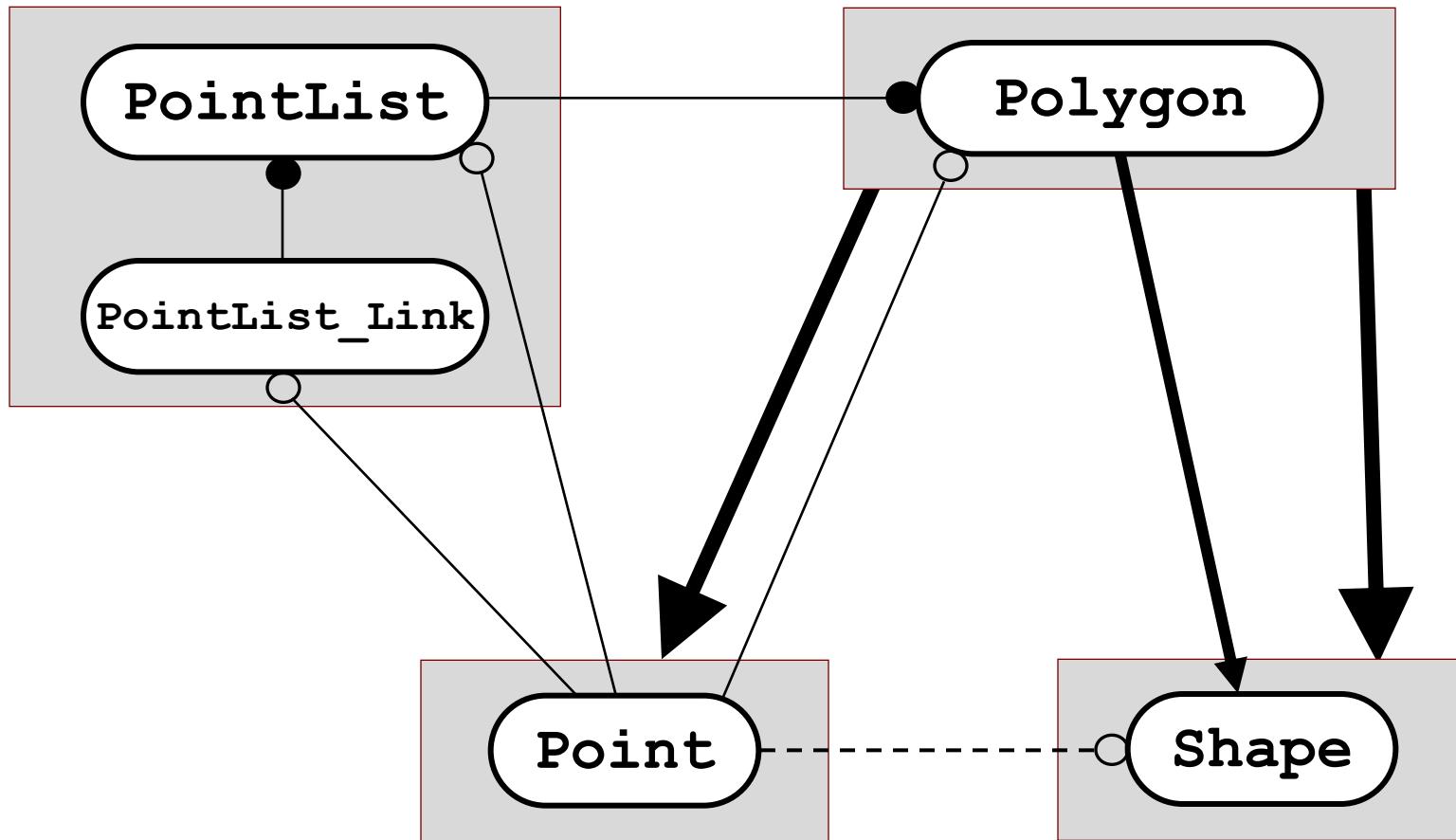


○—> Uses-in-the-Interface
●—> Uses-in-the-Implementation

→ Depends-On
○-----> Uses in name only
→ Is-A

1. Review of Elementary Physical Design

Implied Dependency

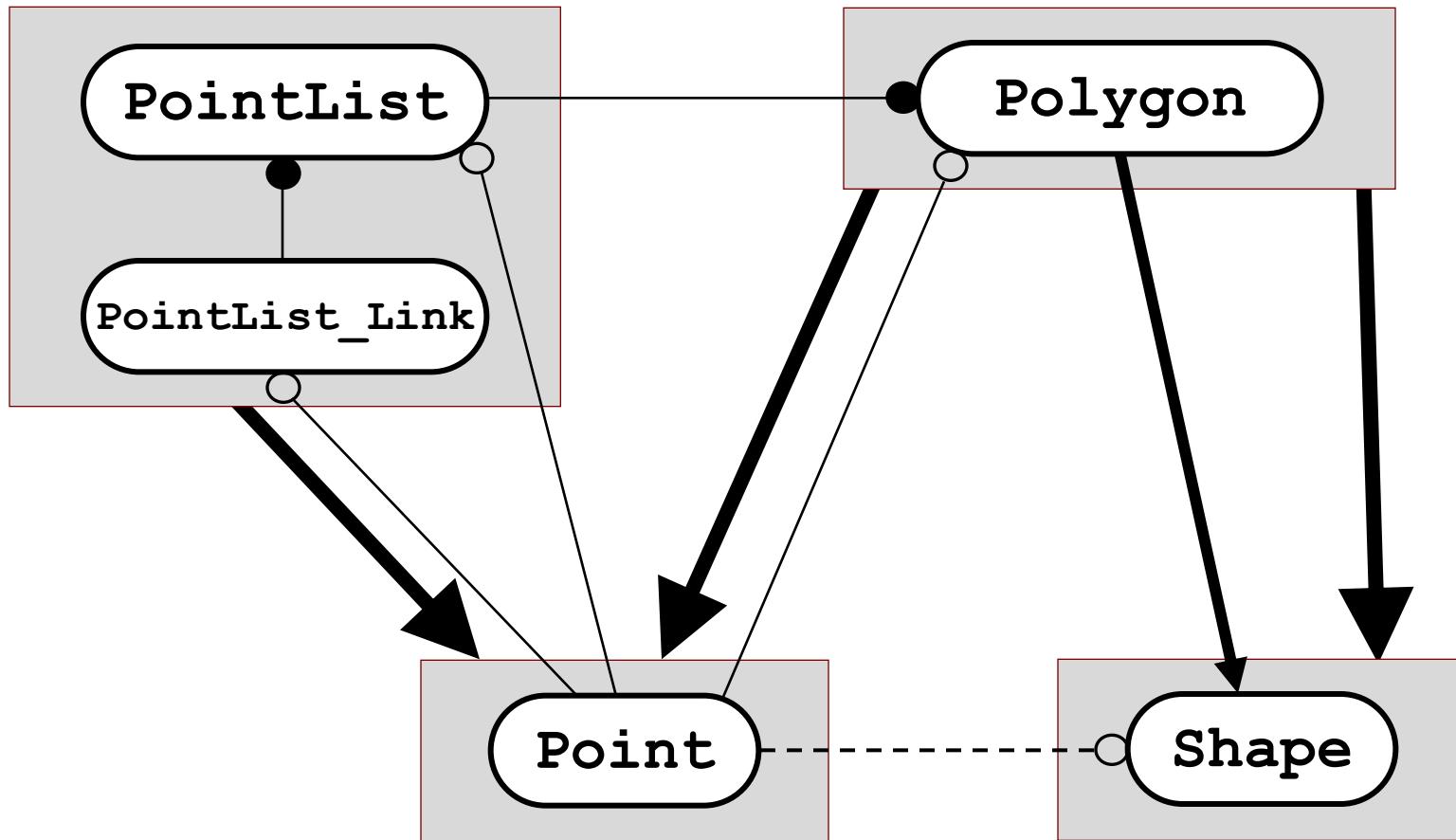


○ ——> Uses-in-the-Interface
● ——> Uses-in-the-Implementation

→ Depends-On
○ - - -> Uses in name only
→ Is-A

1. Review of Elementary Physical Design

Implied Dependency

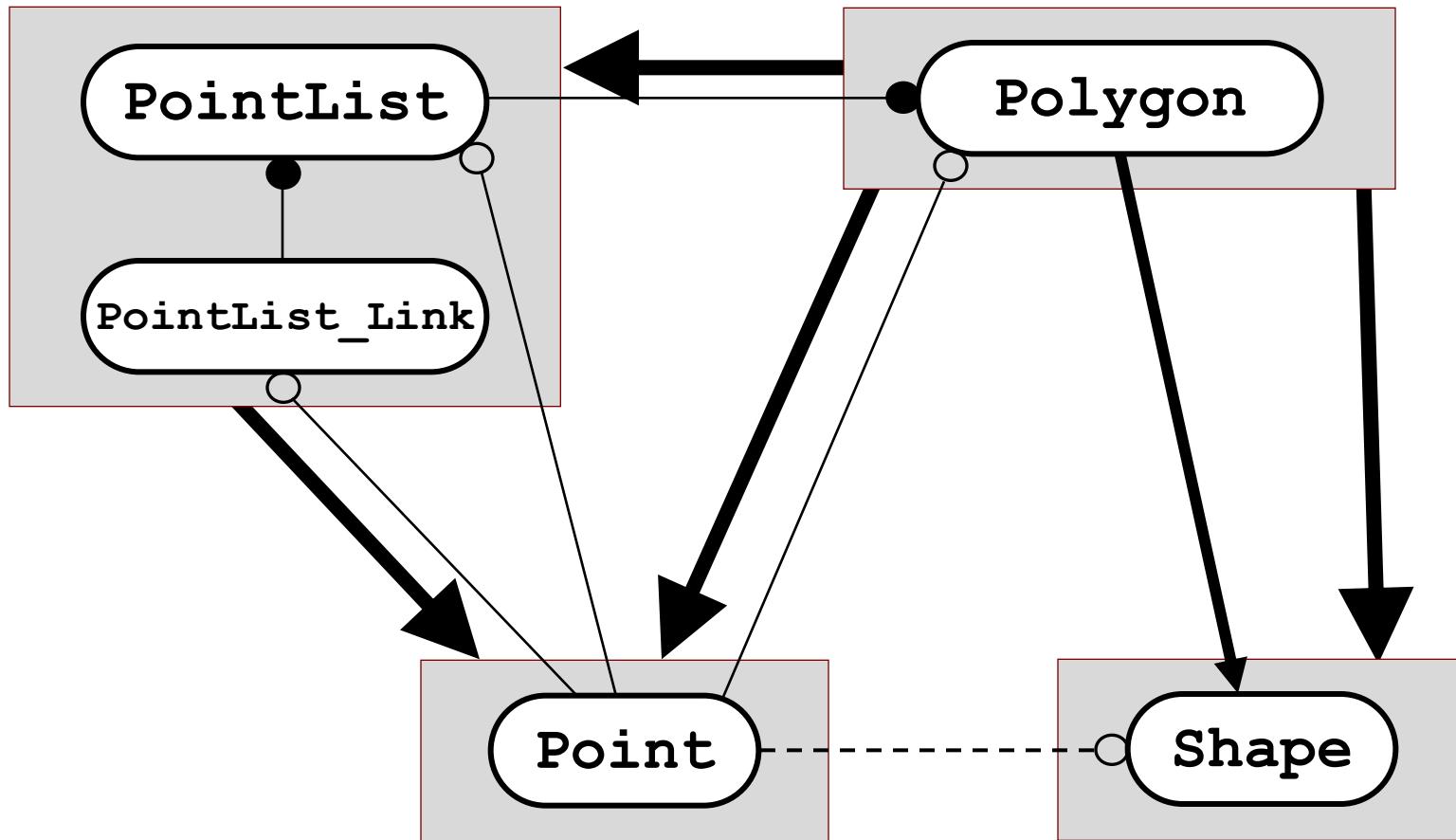


○—→ Uses-in-the-Interface
●—→ Uses-in-the-Implementation

→ Depends-On
○----- Uses in name only
→ Is-A

1. Review of Elementary Physical Design

Implied Dependency

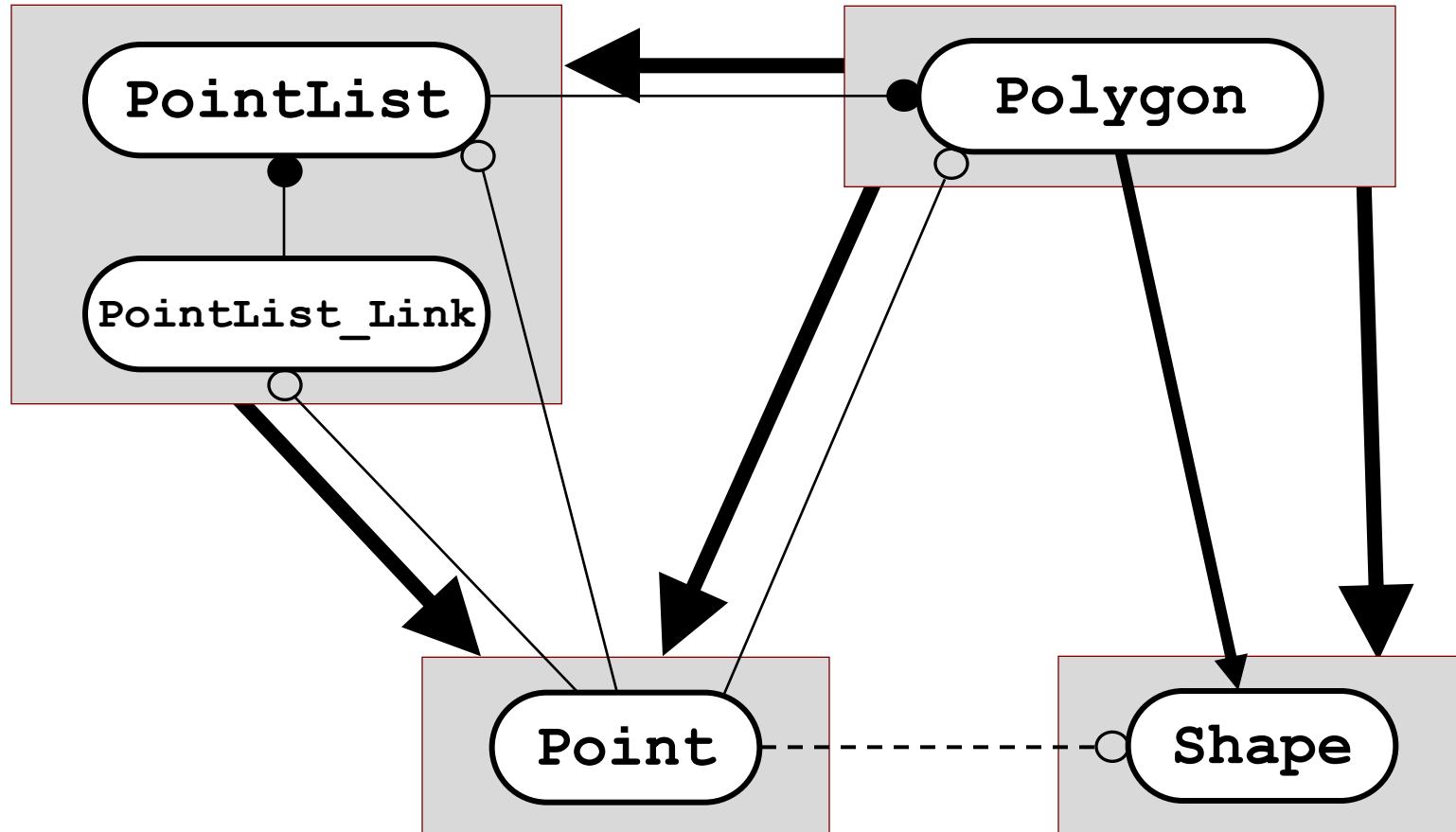


○—> Uses-in-the-Interface
●—> Uses-in-the-Implementation

→ Depends-On
○-----> Uses in name only
→ Is-A

1. Review of Elementary Physical Design

Level Numbers

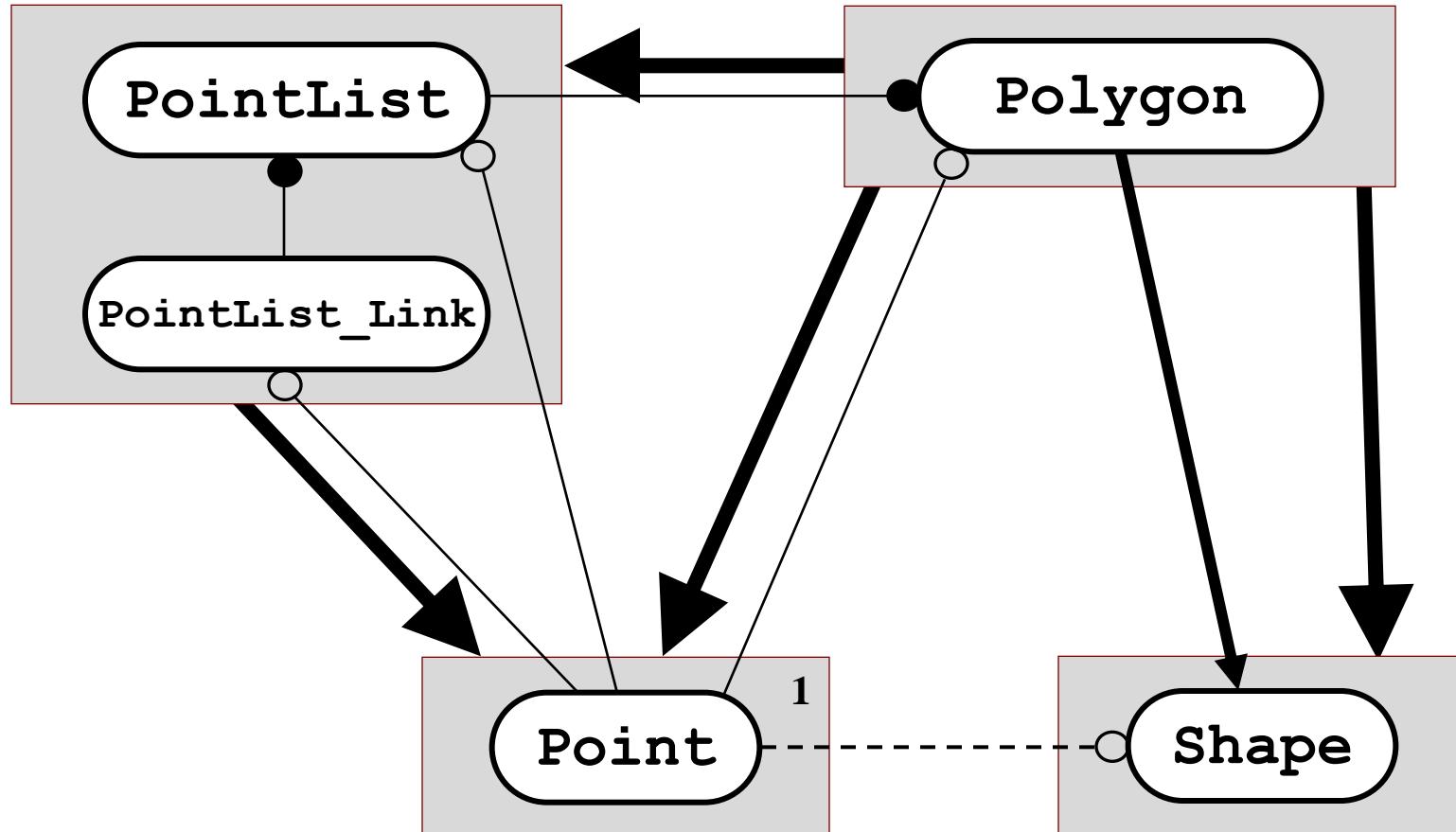


○—> Uses-in-the-Interface
●—> Uses-in-the-Implementation

→ Depends-On
○-----> Uses in name only
→ Is-A

1. Review of Elementary Physical Design

Level Numbers

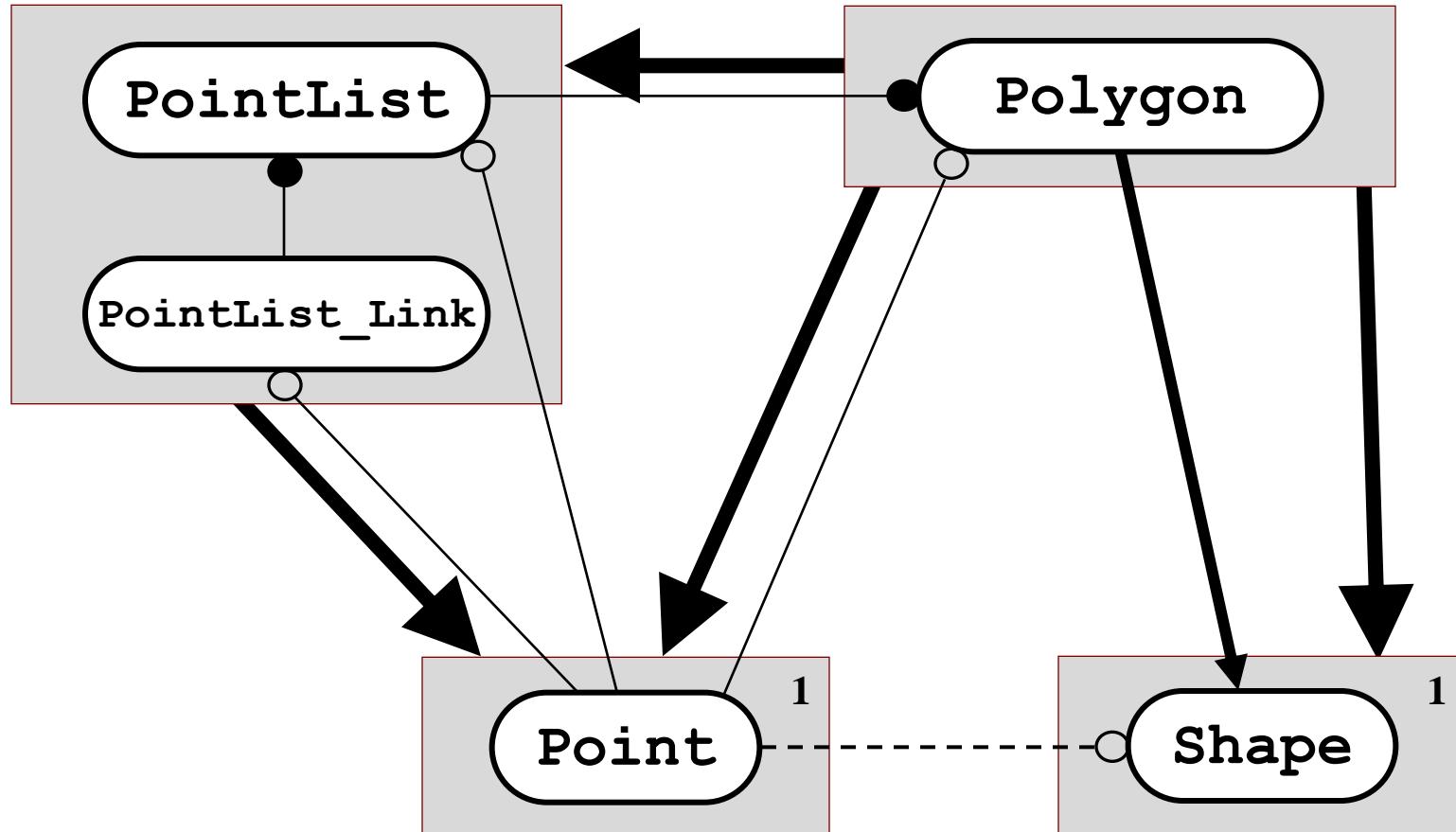


○— Uses-in-the-Interface
●— Uses-in-the-Implementation

→ Depends-On
○--- Uses in name only
→ Is-A

1. Review of Elementary Physical Design

Level Numbers

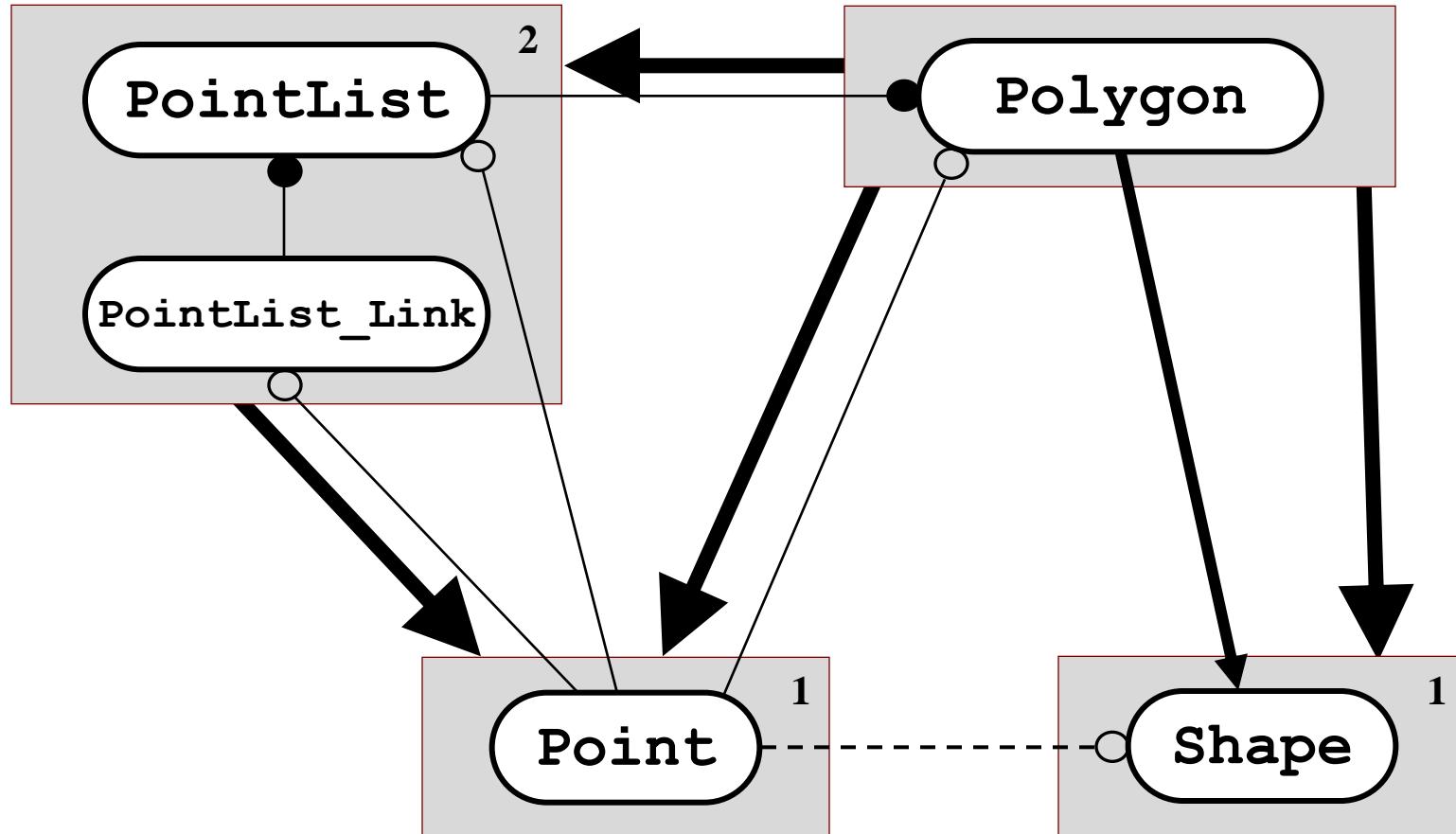


○— Uses-in-the-Interface
●— Uses-in-the-Implementation

→ Depends-On
○--- Uses in name only
→ Is-A

1. Review of Elementary Physical Design

Level Numbers

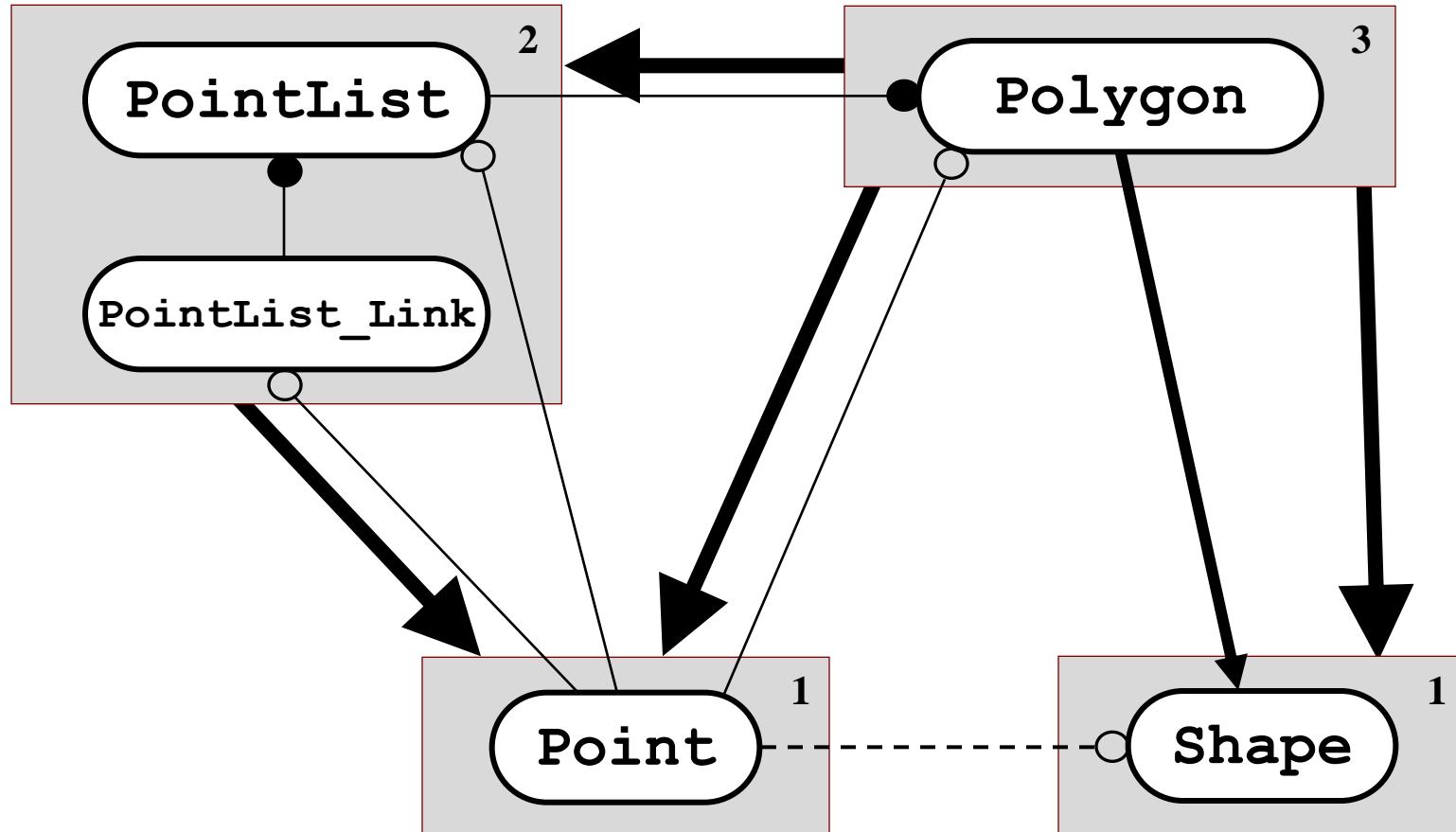


○—→ Uses-in-the-Interface
●—→ Uses-in-the-Implementation

→ Depends-On
○----- Uses in name only
→ Is-A

1. Review of Elementary Physical Design

Level Numbers



○ ————— Uses-in-the-Interface
● ————— Uses-in-the-Implementation

→ Dependence
○ - - - - - Use in name only
→ Is-A

1. Review of Elementary Physical Design

Essential Physical Design Rules

1. Review of Elementary Physical Design

Essential Physical Design Rules

There are two:

1. Review of Elementary Physical Design

Essential Physical Design Rules

There are two:

1. No *Cyclic* Physical Dependencies!

1. Review of Elementary Physical Design

Essential Physical Design Rules

There are two:

1. No *Cyclic* Physical
Dependencies!

2. No *Long-Distance*
Friendships!

1. Review of Elementary Physical Design

Criteria for Colocating “Public” Classes

1. Review of Elementary Physical Design

Criteria for Colocating “Public” Classes

There are four:

1. Review of Elementary Physical Design

Criteria for Colocating “Public” Classes

There are four:

1. Friendship.

1. Review of Elementary Physical Design

Criteria for Colocating “Public” Classes

There are four:

1. Friendship.

2. Cyclic Dependency.

1. Review of Elementary Physical Design

Criteria for Colocating “Public” Classes

There are four:

1. Friendship.

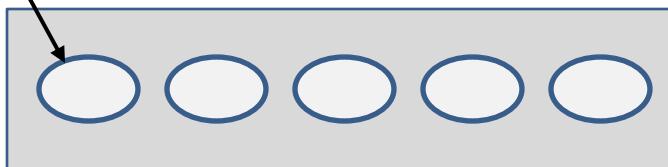
2. Cyclic Dependency.

3. Single Solution.

1. Review of Elementary Physical Design

Criteria for Colocating “Public” Classes

Not reusable
independently.

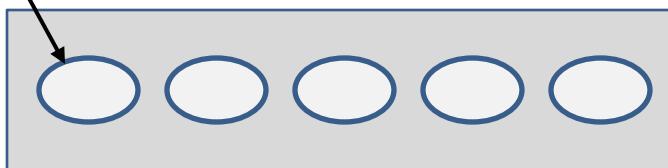


Single Solution

1. Review of Elementary Physical Design

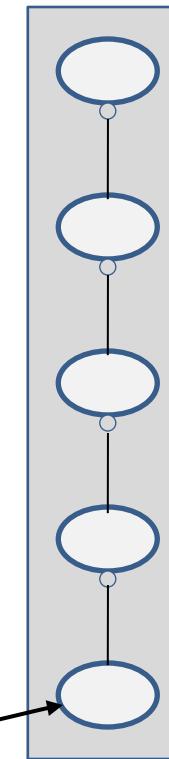
Criteria for Colocating “Public” Classes

Not reusable
independently.



Single Solution

Independently
reusable.

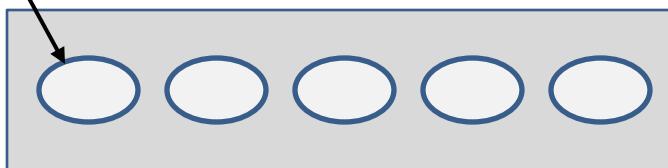


Hierarchy of Solutions

1. Review of Elementary Physical Design

Criteria for Colocating “Public” Classes

Not reusable
independently.



Single Solution

Independently
reusable.

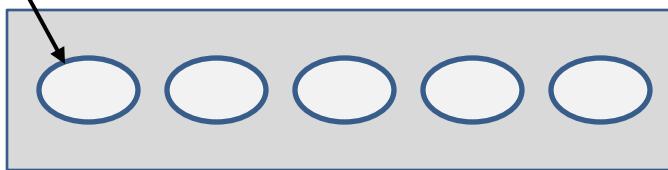


Hierarchy of Solutions

1. Review of Elementary Physical Design

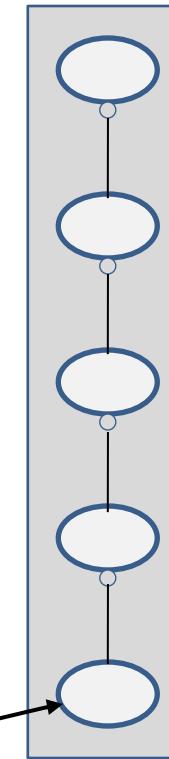
Criteria for Colocating “Public” Classes

Not reusable
independently.



Single Solution

Independently
reusable.

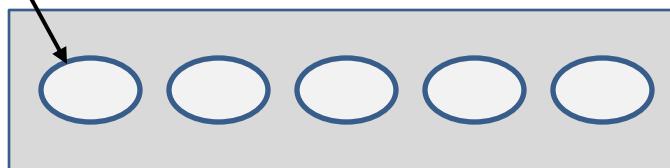


Hierarchy of Solutions

1. Review of Elementary Physical Design

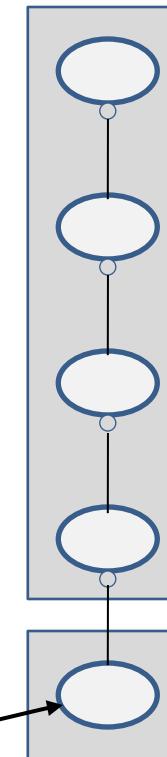
Criteria for Colocating “Public” Classes

Not reusable
independently.



Single Solution

Independently
reusable.

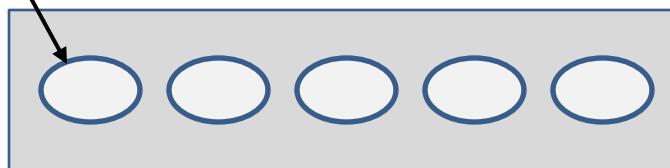


Hierarchy of Solutions

1. Review of Elementary Physical Design

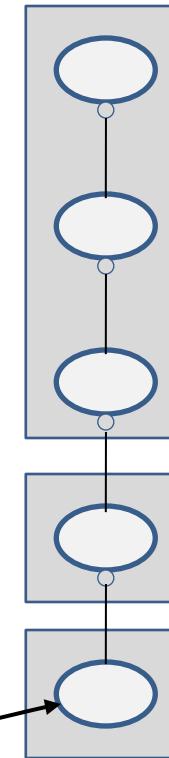
Criteria for Colocating “Public” Classes

Not reusable
independently.



Single Solution

Independently
reusable.

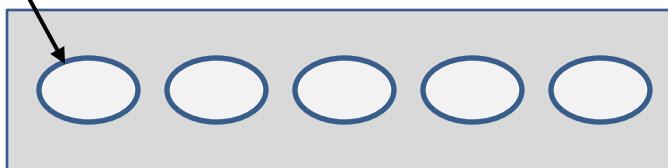


Hierarchy of Solutions

1. Review of Elementary Physical Design

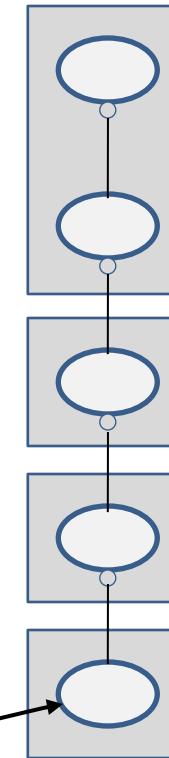
Criteria for Colocating “Public” Classes

Not reusable
independently.



Single Solution

Independently
reusable.

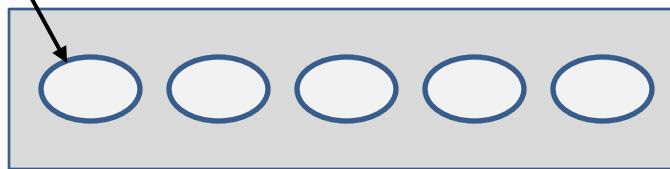


Hierarchy of Solutions

1. Review of Elementary Physical Design

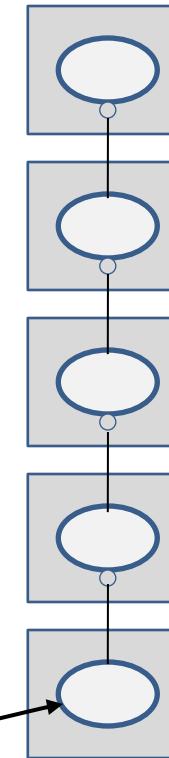
Criteria for Colocating “Public” Classes

Not reusable
independently.



Single Solution

Independently
reusable.

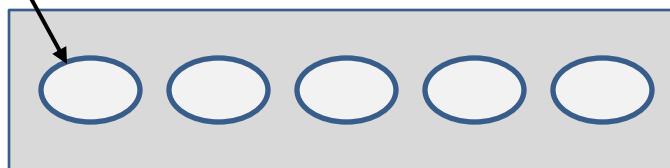


Hierarchy of Solutions

1. Review of Elementary Physical Design

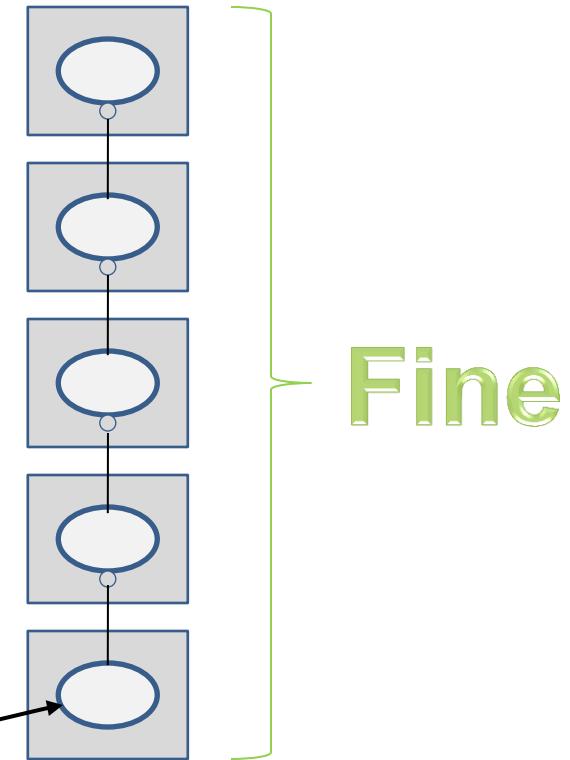
Criteria for Colocating “Public” Classes

Not reusable
independently.



Single Solution

Independently
reusable.



Hierarchy of Solutions

1. Review of Elementary Physical Design

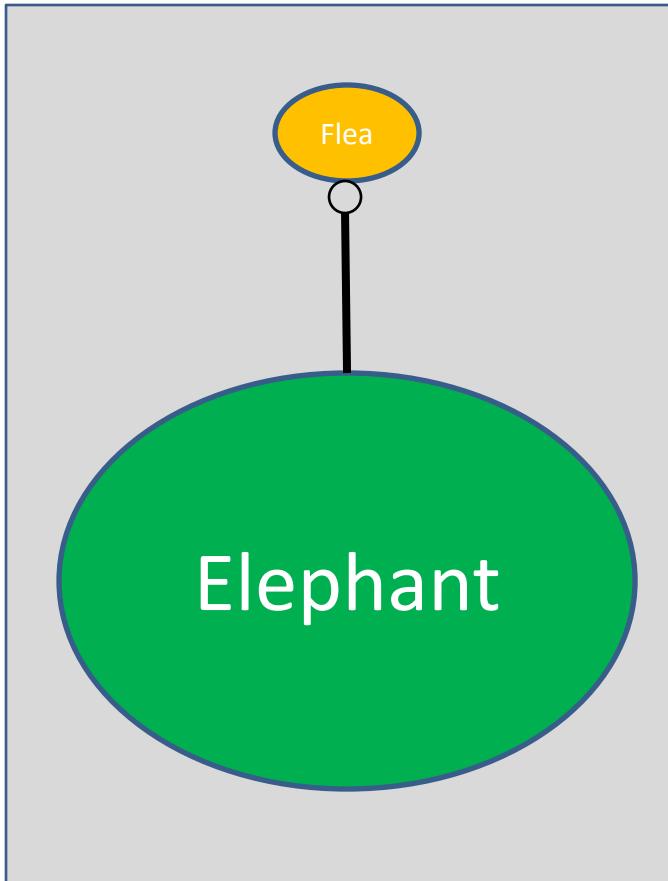
Criteria for Colocating “Public” Classes

There are four:

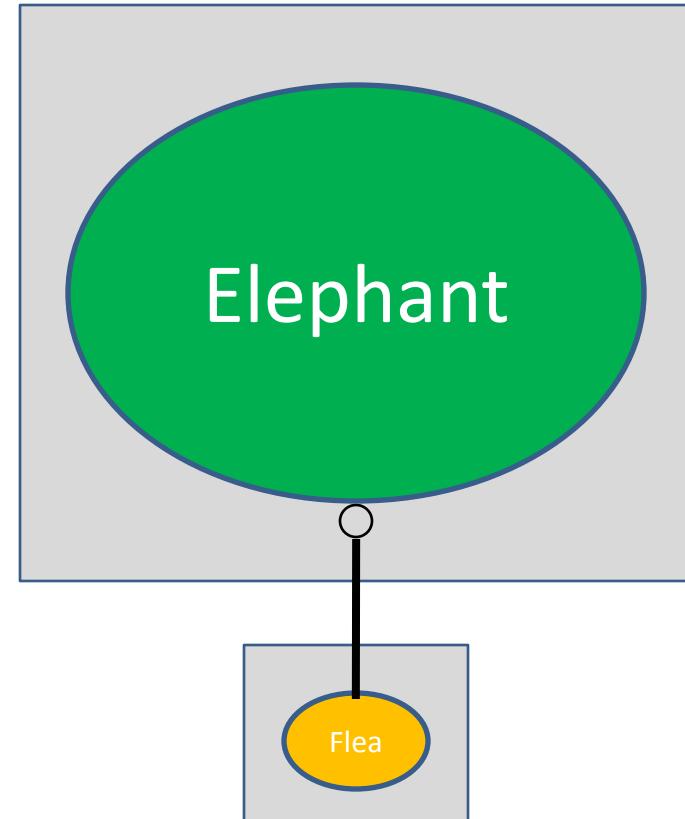
1. Friendship.
2. Cyclic Dependency.
3. Single Solution.
4. “Flea on an Elephant.”

1. Review of Elementary Physical Design

Criteria for Colocating “Public” Classes



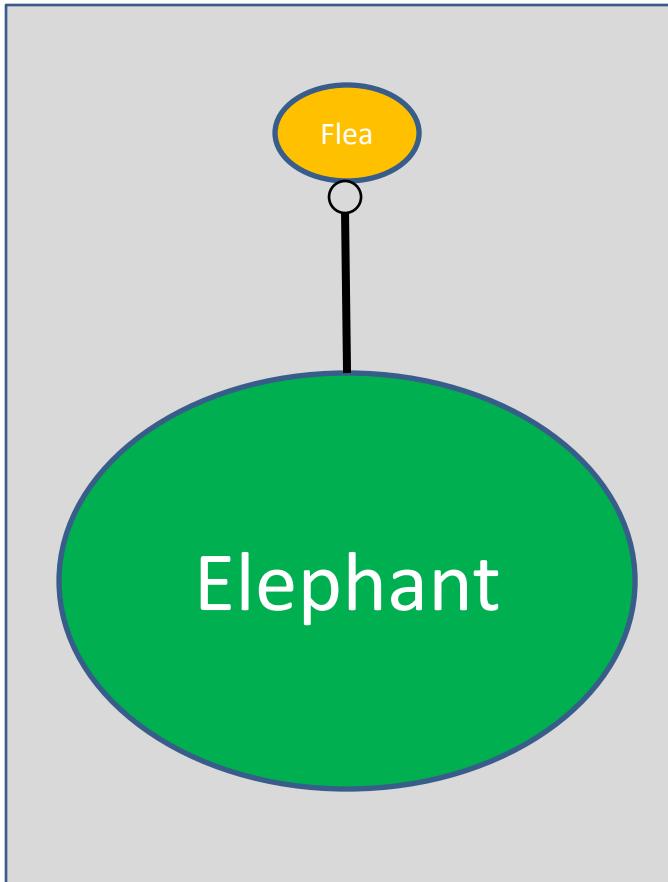
“Flea on an Elephant”



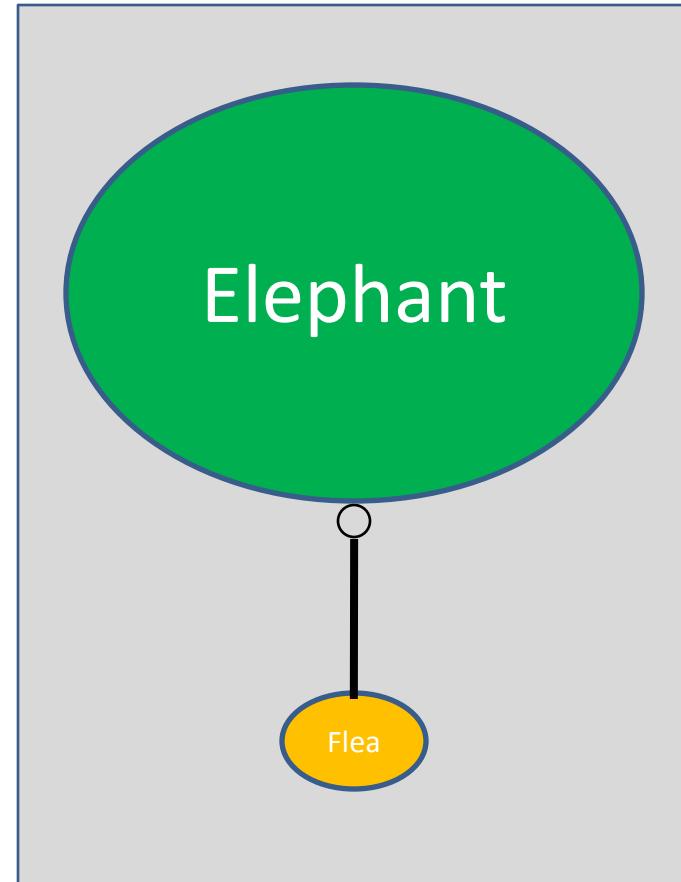
(Elephant on a Flea)

1. Review of Elementary Physical Design

Criteria for Colocating “Public” Classes



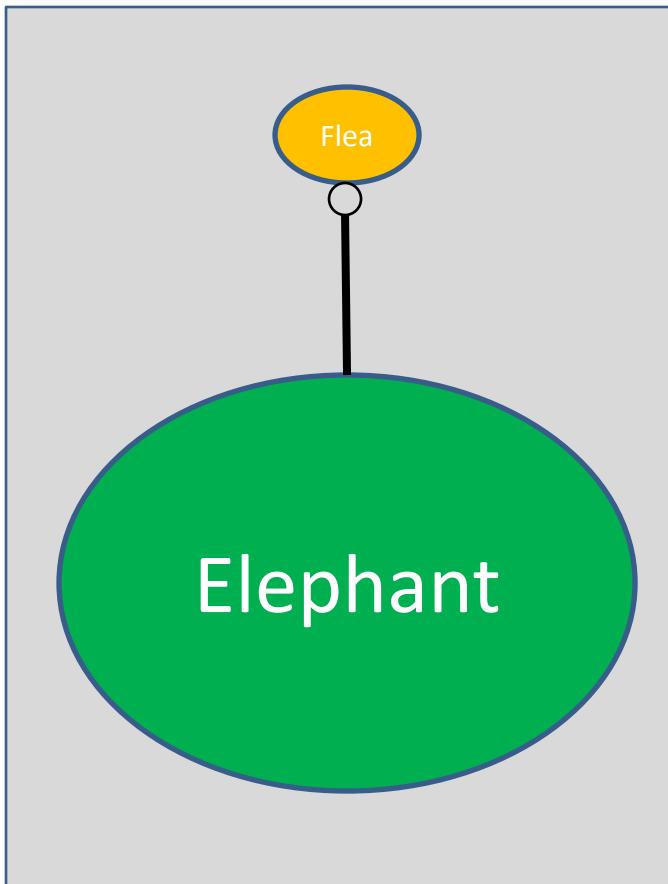
“Flea on an Elephant”



(Elephant on a Flea)

1. Review of Elementary Physical Design

Criteria for Colocating “Public” Classes



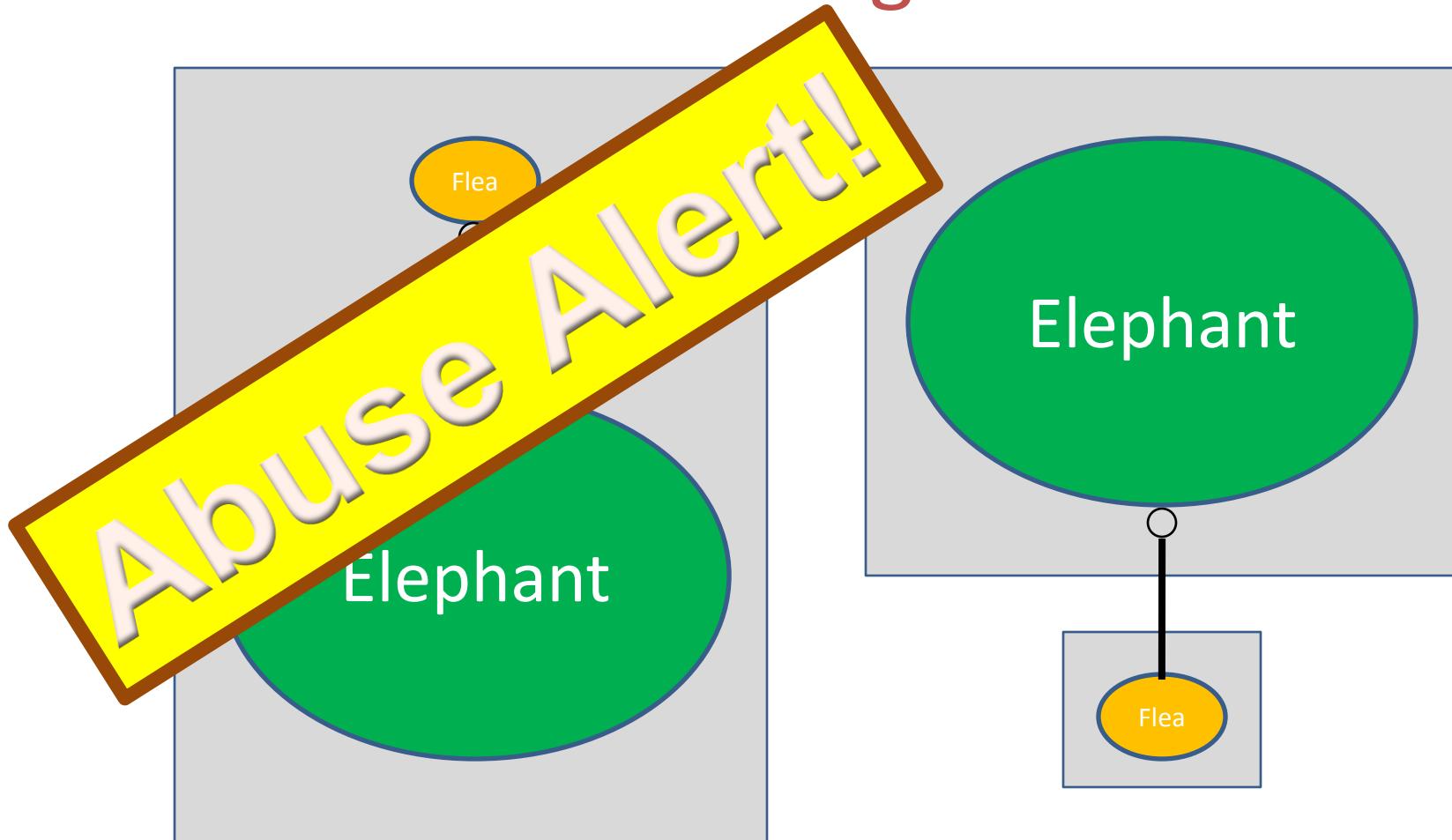
“Flea on an Elephant”



(Elephant on a Flea)

1. Review of Elementary Physical Design

Criteria for Colocating “Public” Classes



“Flea on an Elephant”

(Elephant on a Flea)

1. Review of Elementary Physical Design

Physical Dependency

Five levels of **physical dependency**:

Level 5:



Level 4:



Level 3:



Level 2:



Level 1:



1. Review of Elementary Physical Design

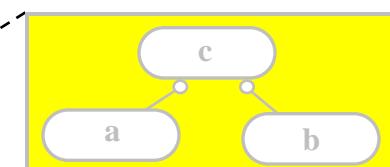
Physical Aggregation

Only one level of **physical aggregation**:

Level 5:



Level 4:



Level 3:



Level 2:



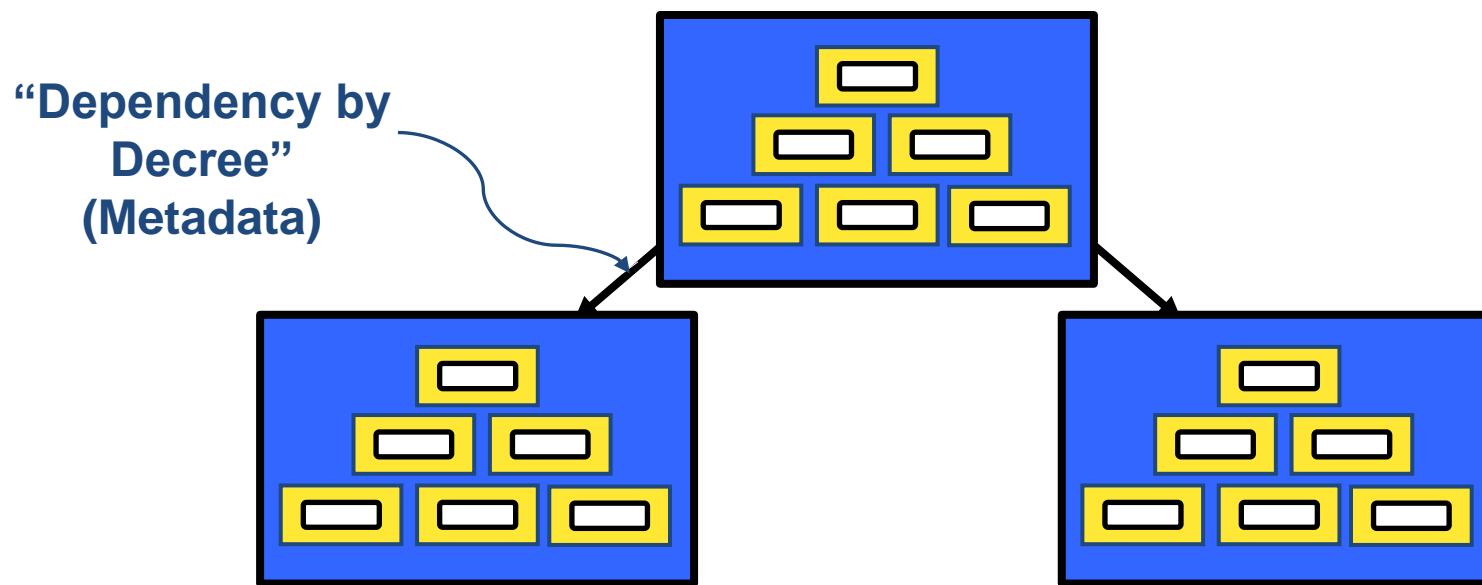
Level 1:



1. Review of Elementary Physical Design

The Package

Two levels of physical aggregation:

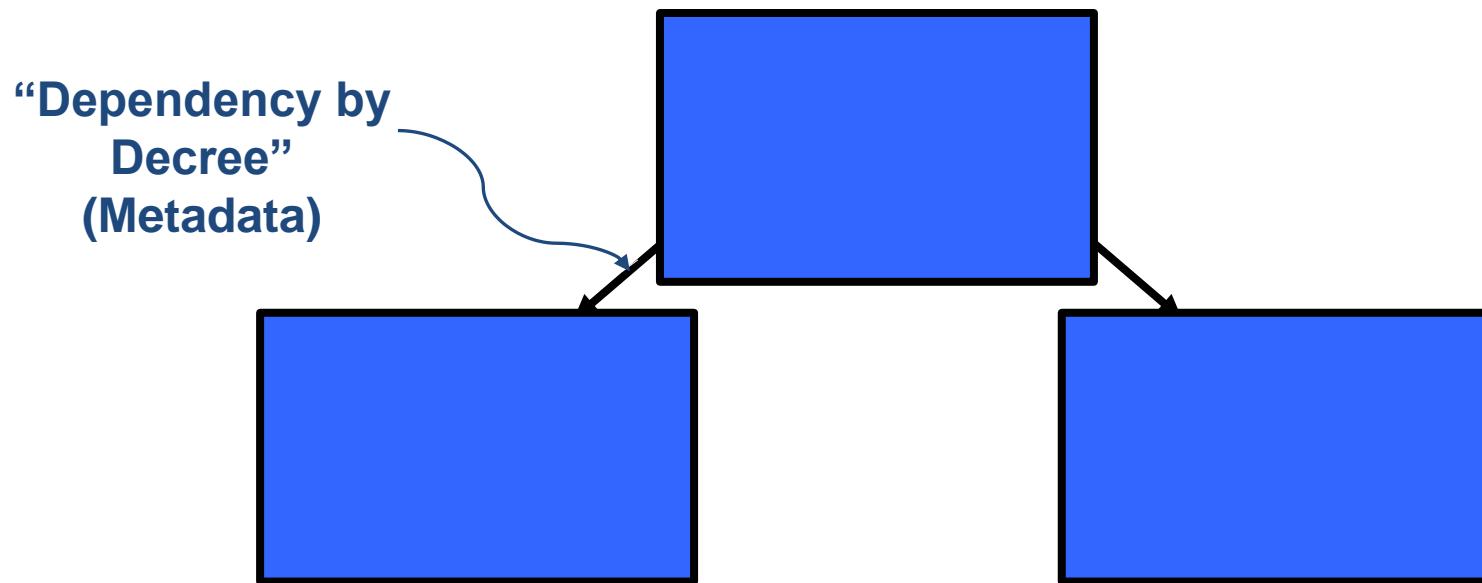


“A Hierarchy of Component Hierarchies”

1. Review of Elementary Physical Design

The Package

Two levels of physical aggregation:

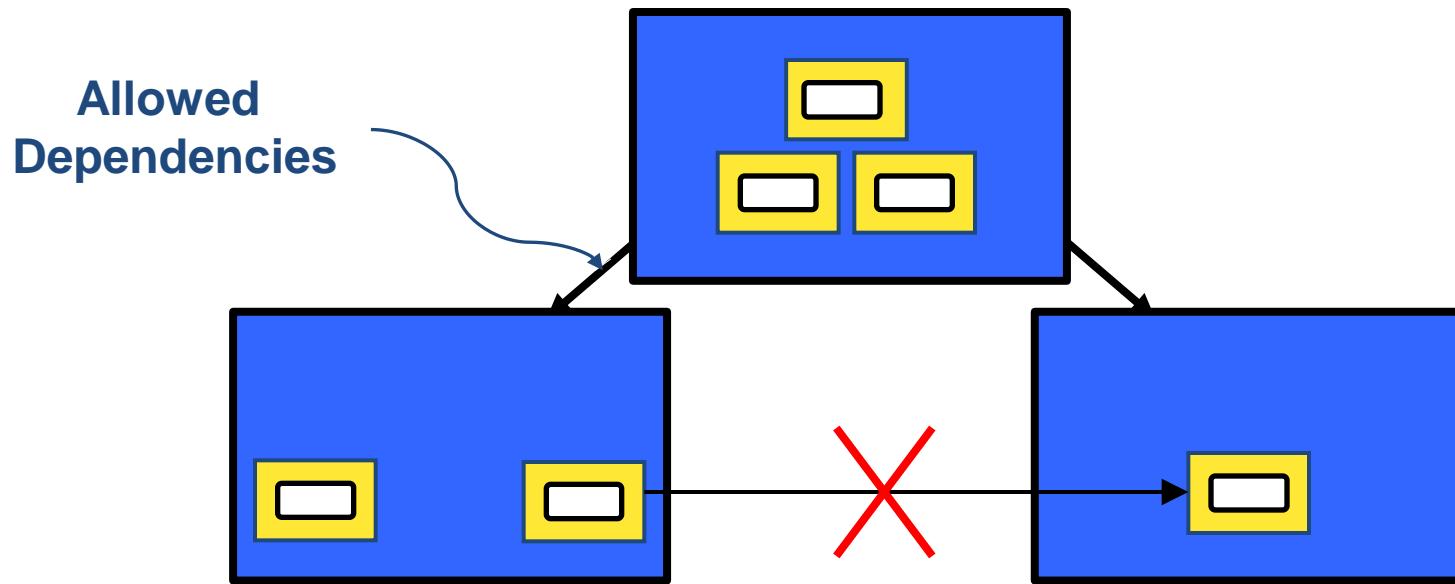


Metadata governs, even absent of any components!

1. Review of Elementary Physical Design

The Package

Two levels of physical aggregation:



Metadata governs allowed dependencies.

1. Review of Elementary Physical Design

Package Dependencies

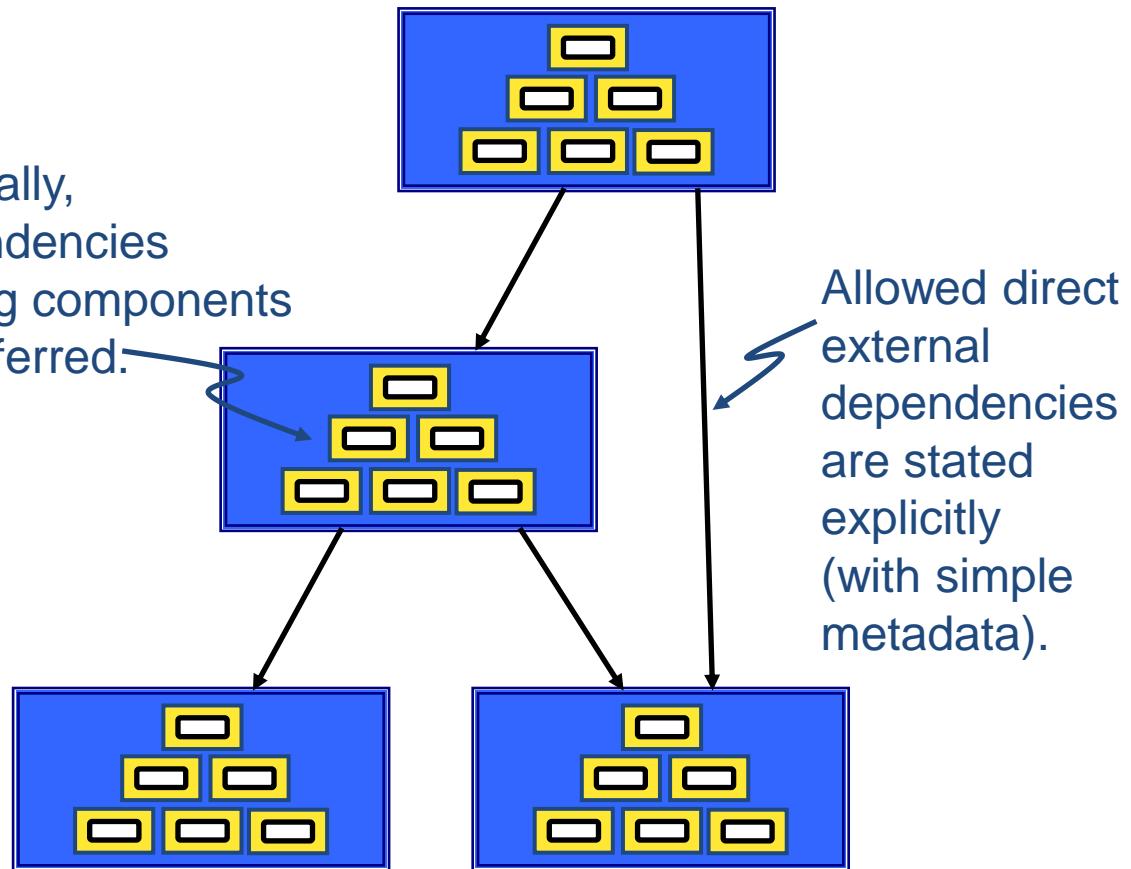
Aggregate dependencies:

Aggregate Level 3:

Internally,
dependencies
among components
are inferred.

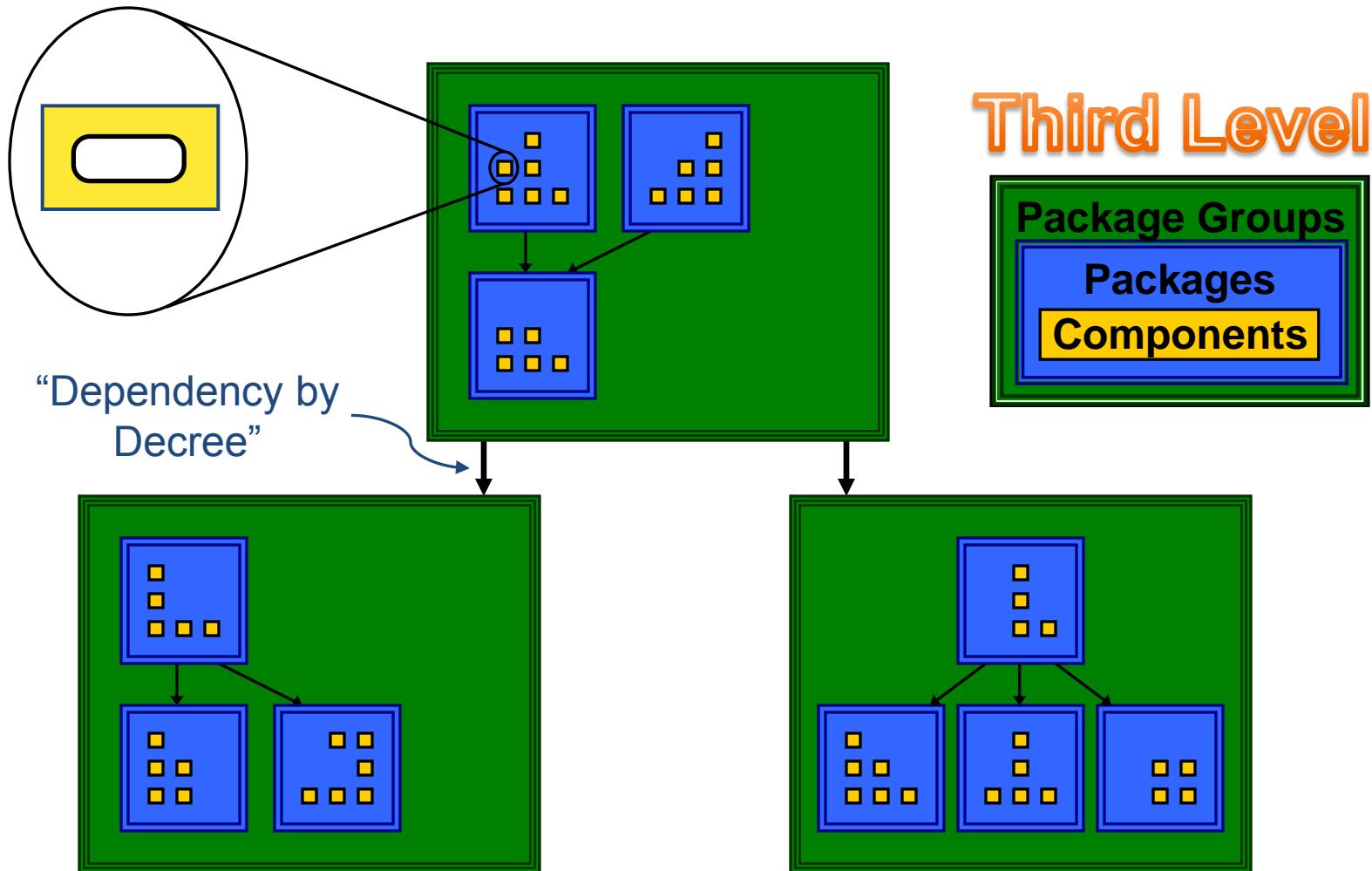
Aggregate Level 2:

Aggregate Level 1:



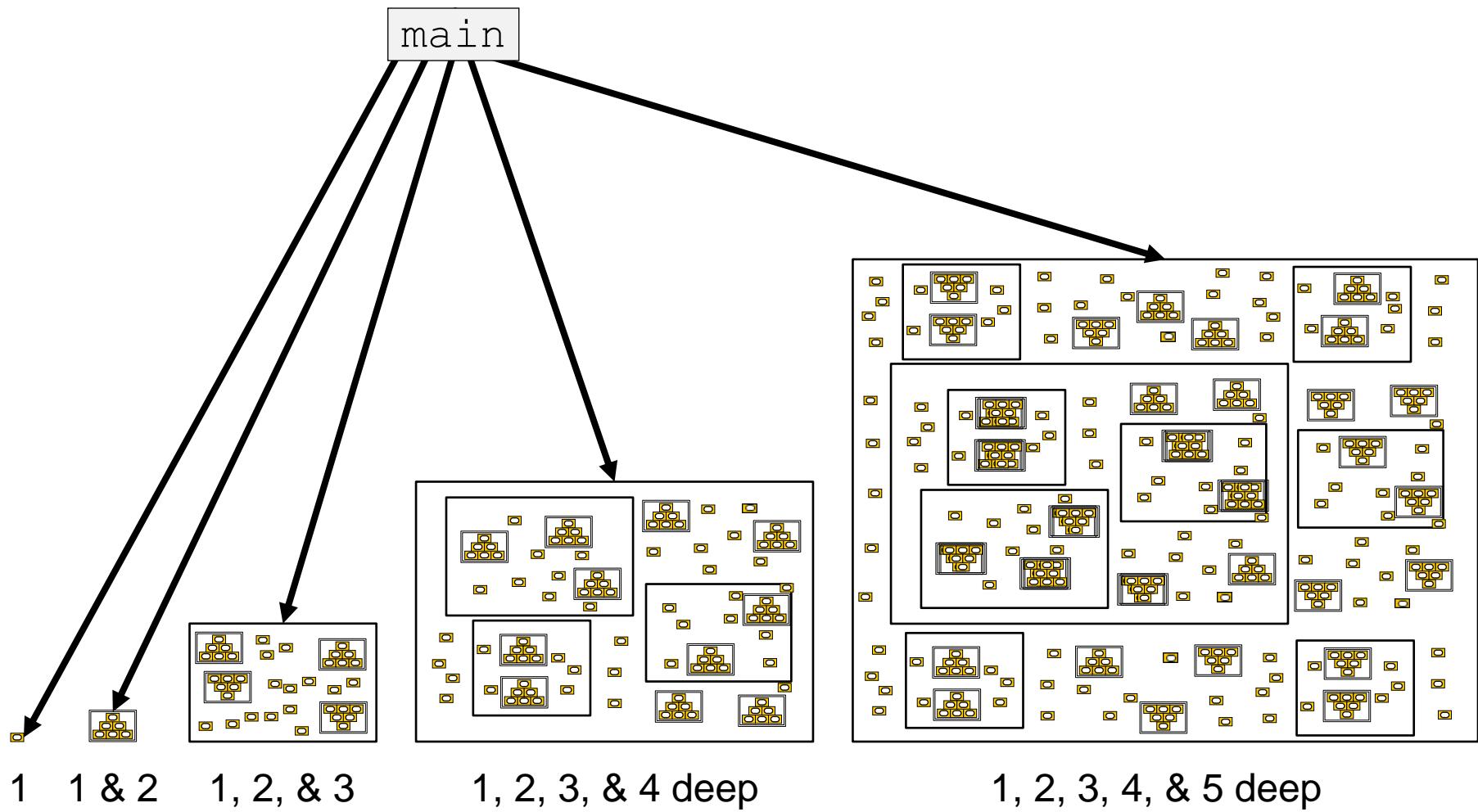
1. Review of Elementary Physical Design

The Package Group



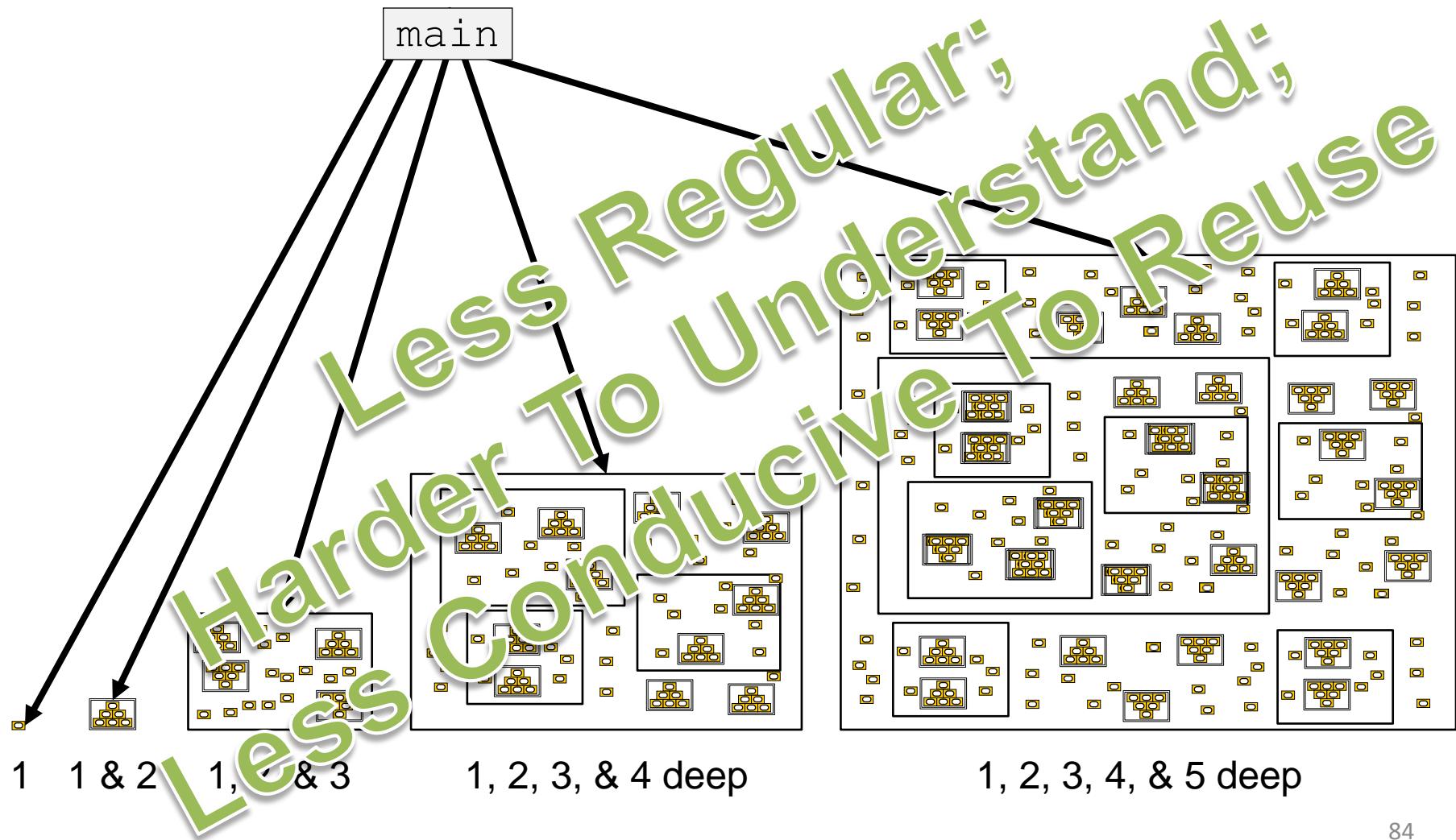
1. Review of Elementary Physical Design

Non-Uniform Physical-Aggregation Depth



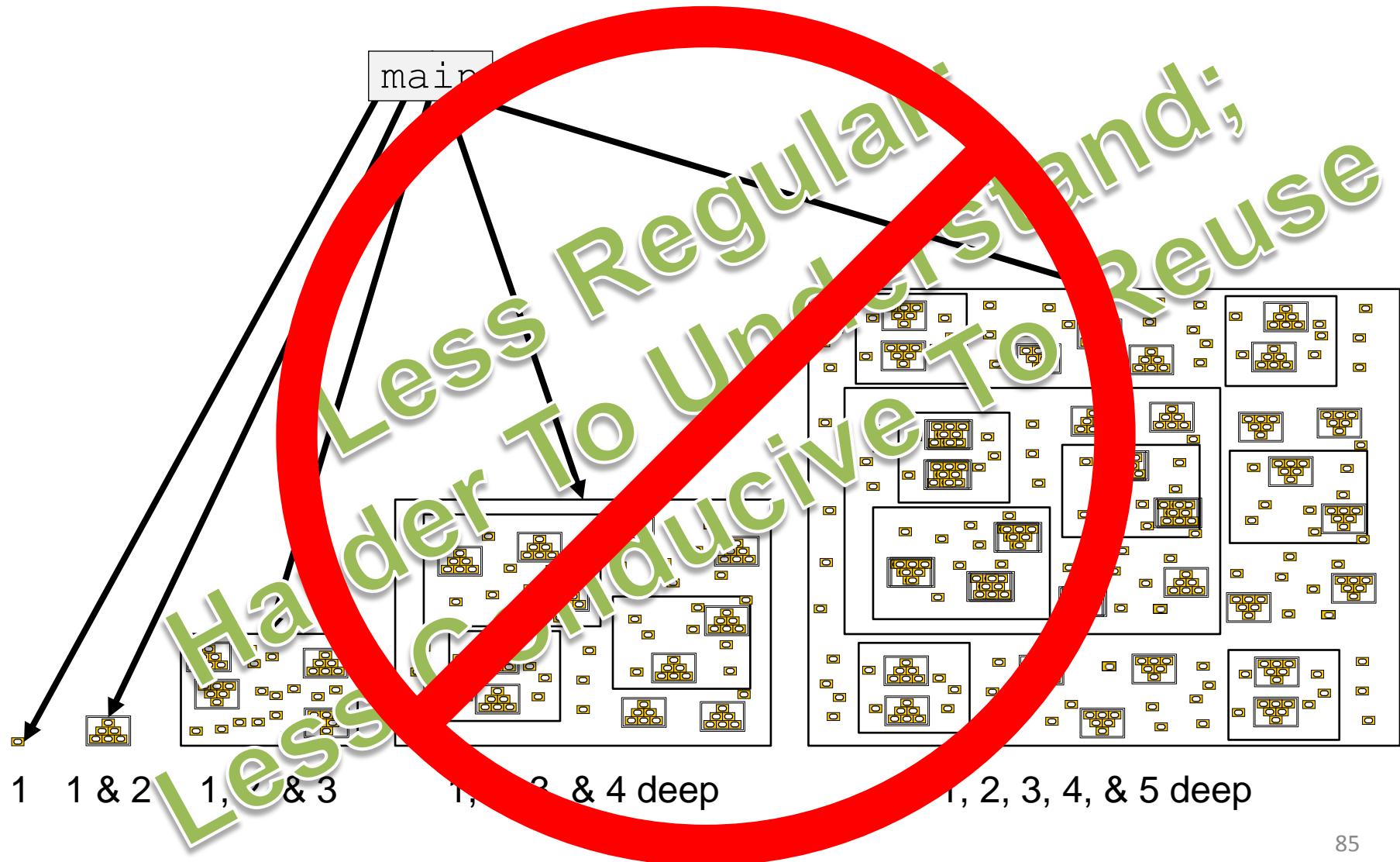
1. Review of Elementary Physical Design

Non-Uniform Physical-Aggregation Depth



1. Review of Elementary Physical Design

Non-Uniform Physical-Aggregation Depth



1. Review of Elementary Physical Design

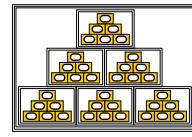
Uniform Depth of Physical Aggregation



Component



Package



Package Group

1. Review of Elementary Physical Design

Uniform Depth of Physical Aggregation

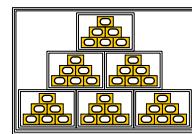
Exactly Three Levels
of Physical Aggregation



Component



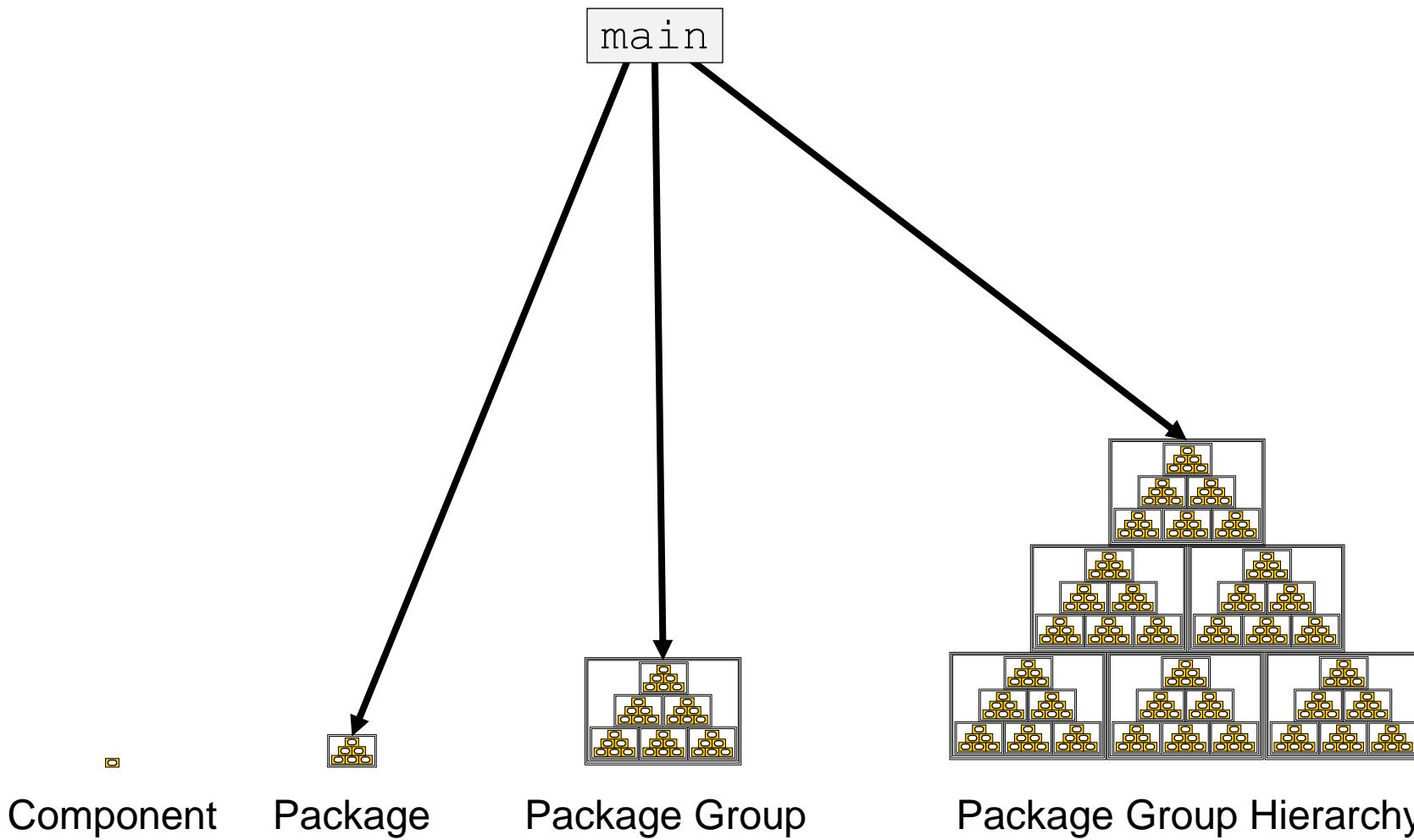
Package



Package Group

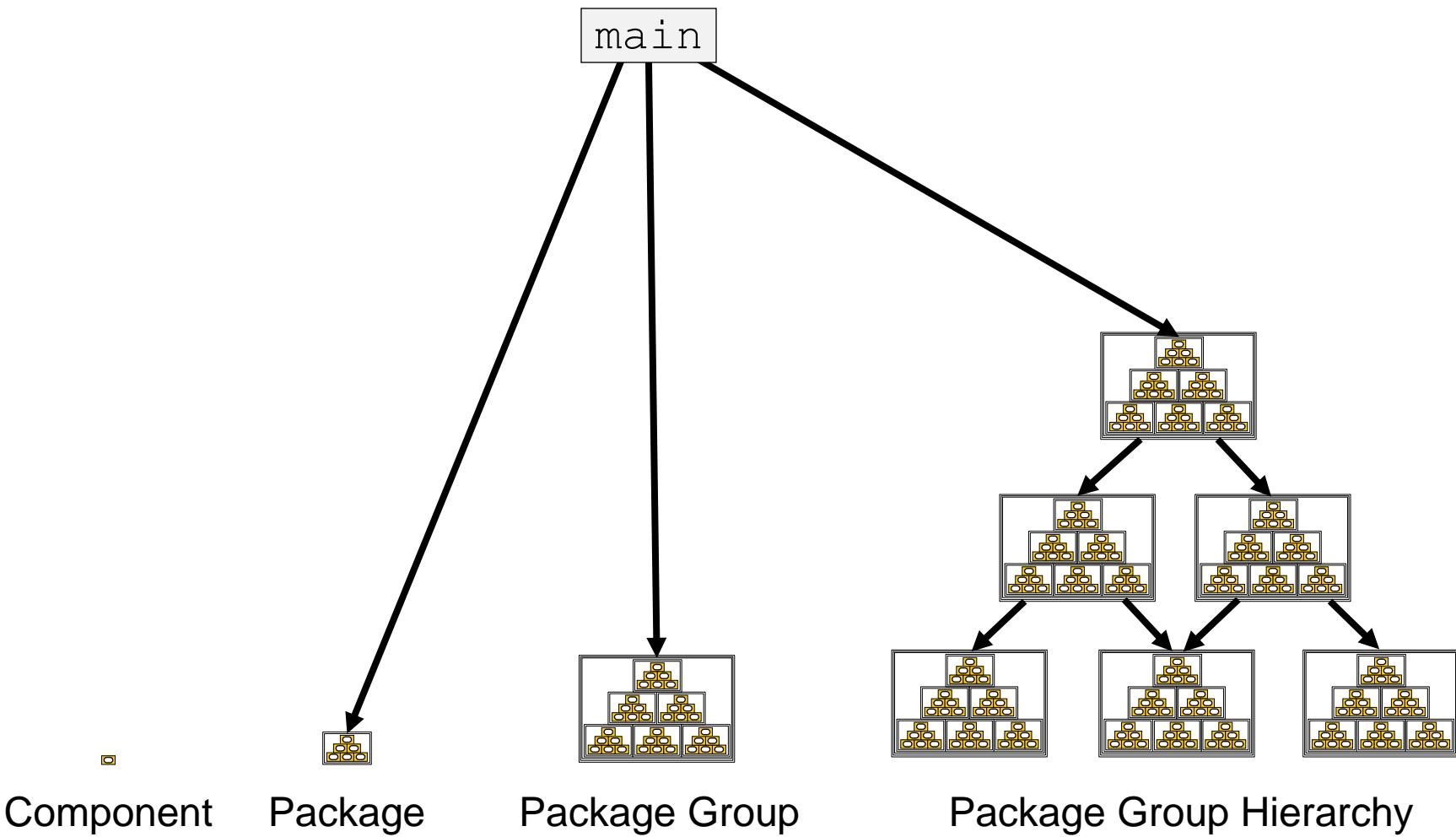
1. Review of Elementary Physical Design

Uniform Depth of Physical Aggregation



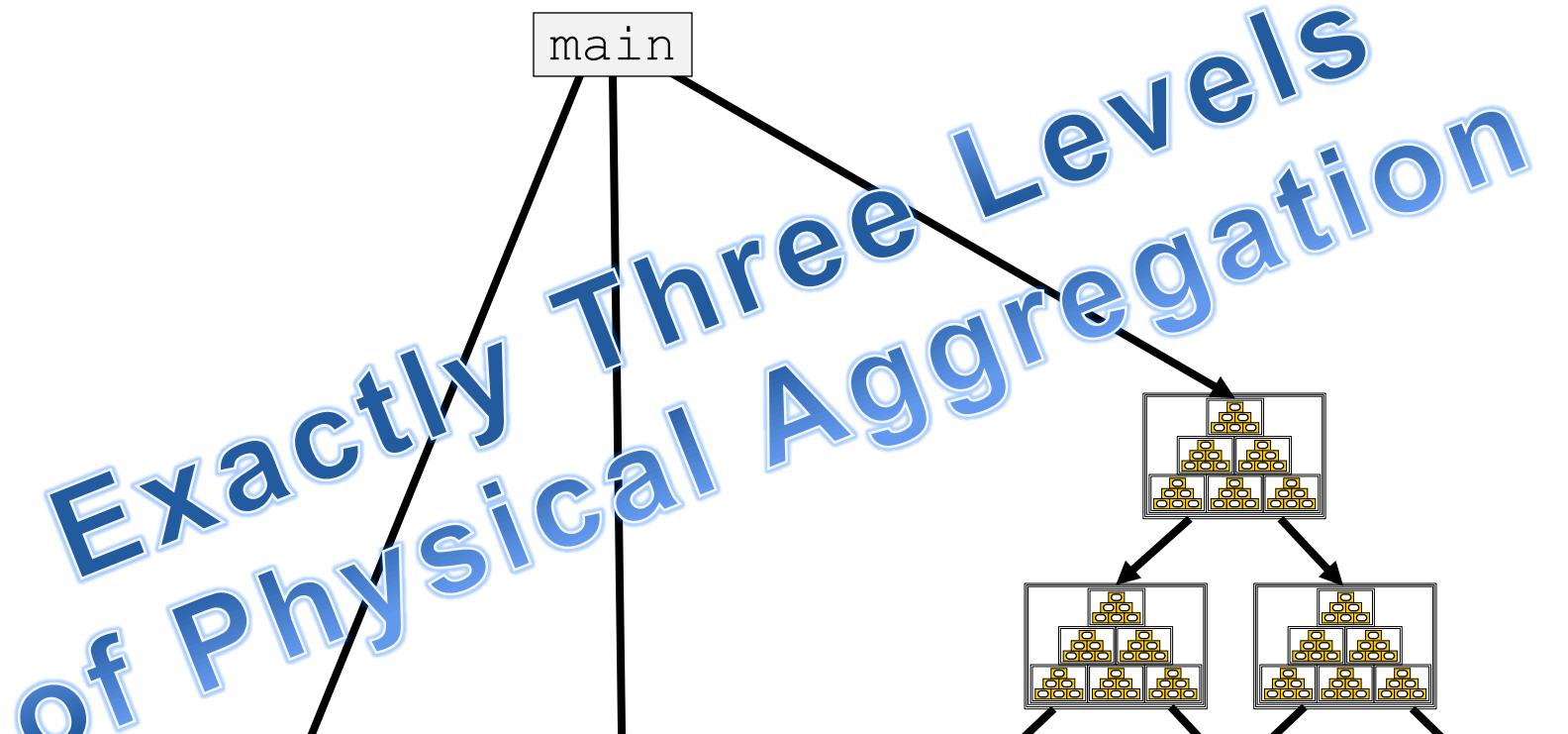
1. Review of Elementary Physical Design

Uniform Depth of Physical Aggregation



1. Review of Elementary Physical Design

Uniform Depth of Physical Aggregation



Component

Package

Package Group

Package Group Hierarchy

1. Review of Elementary Physical Design

End of Section

Questions?

1. Review of Elementary Physical Design

What Questions are we Answering?

- What distinguishes *Logical* from *Physical* Design?
- What are the fundamental properties of a `.h/.cpp` pair that make it a **component**?
- How do we infer physical relationships (*Depends-On*) from logical relationships (e.g., *Is-A* and *Uses*)?
- What are *level numbers*, and how do we determine them?
- How do we extract **component dependencies** efficiently?
- What essential **physical design rules** must be followed?
- What are the criteria for **colocating classes & functions**?
- How are **components** aggregated into larger **physical units**?
- How many levels of **physical aggregation** do we employ?

1. Review of Elementary Physical Design

What Questions are we Answering?

- What distinguishes *Logical* from *Physical* Design?
- What are the fundamental properties of a `.h/.cpp` pair that make it a **component**?
- How do we infer physical relationships (*Depends-On*) from logical relationships (e.g., *Is-A* and *Uses*)?
- What are *level numbers*, and how do we determine them?
- How do we extract **component dependencies** efficiently?
- What essential **physical design rules** must be followed?
- What are the criteria for **colocating classes & functions**?
- How are **components** aggregated into larger **physical units**?
- How many levels of **physical aggregation** do we employ?

Outline

1. Review of Elementary Physical Design
Components, Modularity, Physical Dependencies
2. Survey of Advanced *Levelization* Techniques
Avoiding Cyclic, Excessive, or Inappropriate Dependencies
3. Total and Partial *Insulation* Techniques
Avoiding Unnecessary Compile-Time Coupling
4. Present-Day, Real-World Design Examples
Prioritizing Physical Design to Improve Software Quality

Outline

1. Review of Elementary Physical Design
Components, Modularity, Physical Dependencies
2. Survey of Advanced *Levelization* Techniques
Avoiding Cyclic, Excessive, or Inappropriate Dependencies
3. Total and Partial *Insulation* Techniques
Avoiding Unnecessary Compile-Time Coupling
4. Present-Day, Real-World Design Examples
Prioritizing Physical Design to Improve Software Quality

2. Survey of Advanced *Levelization* Techniques

Levelization

Levelize (v.); Levelizable (a.); Levelization (n.)

2. Survey of Advanced *Levelization* Techniques

Levelization

Levelize (v.); Levelizable (a.); Levelization (n.)

Usage:

2. Survey of Advanced *Levelization* Techniques

Levelization

Levelize (v.); Levelizable (a.); Levelization (n.)

Usage:

2. Survey of Advanced *Levelization* Techniques

Levelization

Levelize (v.); Levelizable (a.); Levelization (n.)

Usage:

- We need to *levelize* that design – i.e., we need to make its physical dependency graph acyclic.

2. Survey of Advanced *Levelization* Techniques

Levelization

Levelize (v.); **Levelizable** (a.); Levelization (n.)

Usage:

- We need to *levelize* that design – i.e., we need to make its physical dependency graph acyclic.

2. Survey of Advanced *Levelization* Techniques

Levelization

Levelize (v.); **Levelizable** (a.); Levelization (n.)

Usage:

- We need to *levelize* that design – i.e., we need to make its physical dependency graph acyclic.
- Are you sure that design is **levelizable** – i.e., do we know how to make its physical dependencies acyclic?

2. Survey of Advanced *Levelization* Techniques

Levelization

Levelize (v.); Levelizable (a.); **Levelization (n.)**

Usage:

- We need to *levelize* that design – i.e., we need to make its physical dependency graph acyclic.
- Are you sure that design is *levelizable* – i.e., do we know how to make its physical dependencies acyclic?

2. Survey of Advanced *Levelization* Techniques

Levelization

Levelize (v.); Levelizable (a.); **Levelization (n.)**

Usage:

- We need to *levelize* that design – i.e., we need to make its physical dependency graph acyclic.
- Are you sure that design is *levelizable* – i.e., do we know how to make its physical dependencies acyclic?
- What *levelization* techniques would you use – i.e., what techniques would you use to *levelize* your design?

2. Survey of Advanced *Levelization* Techniques

Levelization

Levelize (v.); Levelizable (a.); Leveling (n.)

Usage:

- We need to levelize our design – i.e., does it have cyclic dependencies? (Shameless)
- Are dependencies acyclic? i.e., do we know how to levelize them?
- What techniques would you use – i.e., what techniques would you use to *levelize* your design?

Advertisement

Note that Lakos'96 described 9 different ways to untangle cyclic physical dependencies: ***Escalation, Demotion, Opaque Pointers, Dumb Data, Redundancy, Callbacks, Manager Class, Factoring, and Escalating Encapsulation.***¹⁰⁴

2. Survey of Advanced *Levelization* Techniques

Levelization

Let's learn these techniques now!

Note that Lakos'96 described 9 different ways to untangle cyclic physical dependencies: ***Escalation, Demotion, Opaque Pointers, Dumb Data, Redundancy, Callbacks, Manager Class, Factoring, and Escalating Encapsulation.***¹⁰⁵

2. Survey of Advanced *Levelization* Techniques

Levelization

Levelize (v.); Levelizable (a.); Leveling (n.)

Later, we'll apply them
to real-world examples.

- Given a set of techniques would you use – i.e., what techniques would you use to *levelize* your design?

Note that Lakos'96 described 9 different ways to untangle cyclic physical dependencies: ***Escalation, Demotion, Opaque Pointers, Dumb Data, Redundancy, Callbacks, Manager Class, Factoring, and Escalating Encapsulation.***¹⁰⁶

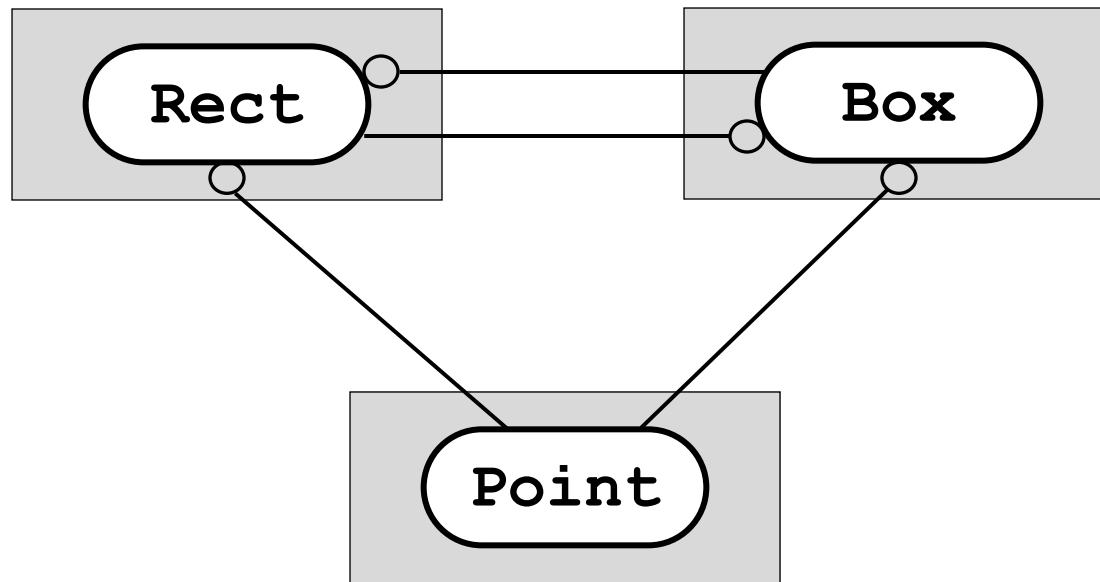
2. Survey of Advanced *Levelization* Techniques

Escalation

Escalation – Moving
mutually dependent
functionality higher in
the physical hierarchy.

2. Survey of Advanced *Levelization* Techniques

Escalation



2. Survey of Advanced *Levelization* Techniques

Escalation

```
// rect.h
#include <point.h>
#include <box.h>
class Rect {
    Point d_origin;
    int    d_width;
    int    d_length;
public:
    // ...
    Rect(const Box& b);
    // ...
};
```

```
// box.h
#include <point.h>
#include <rect.h>
class Box {
    Point d_lowerLeft;
    Point d_upperRight;
public:
    // ...
    Box(const Rect& r);
    // ...
};
```

2. Survey of Advanced *Levelization* Techniques

Escalation

```
// rect.h
#include <point.h>
#include <box.h>
class Rect {
    Point d_origin;
    int    d_width;
    int    d_length;
public:
    // ...
    Rect(const Box& b);
    // ...
};
```

```
// box.h
#include <point.h>
#include <rect.h>
class Box {
    Point d_lowerLeft;
    Point d_upperRight;
public:
    // ...
    Box(const Rect& r);
    // ...
};
```

2. Survey of Advanced *Levelization* Techniques

Escalation

```
// rect.h
#include <point.h>
#include <box.h>
class Rect {
    // ...
public:
    Rect();
    Rect(const Point& o,
          int w, int l);
    Rect(const Box& b);
    // ...
};
```

```
// box.h
#include <point.h>
#include <rect.h>
class Box {
    // ...
public:
    Box();
    Box(const Point& ll,
         const Point& ur);
    Box(const Rect& r);
    // ...
};
```

2. Survey of Advanced *Levelization* Techniques

Escalation

```
// rect.h
#include <point.h>
#include <box.h>
class Rect {
    // ...
public:
    Rect();
    Rect(const Point& o,
          int w, int l);
    Rect(const Box& b);
    // ...
};
```

```
// box.h
#include <point.h>
#include <rect.h>
class Box {
    // ...
public:
    Box();
    Box(const Point& ll,
         const Point& ur);
    Box(const Rect& r);
    // ...
};
```

2. Survey of Advanced *Levelization* Techniques

Escalation

The diagram illustrates the concept of implicit conversions between two classes, Rect and Box. A blue callout bubble labeled "Implicit Conversions" points from the Rect class definition to the Box class definition. A red oval encloses both code snippets.

```
// rect.h
#include <point.h>
#include <box.h>
class Rect {
    // ...
public:
    Rect();
    Rect(const Point& ll, const Point& ur);
    Rect(const Box& b);
    // ...
};

// box.h
#include <rect.h>
#include <point.h>
class Box {
    // ...
public:
    Box();
    Box(const Point& ll, const Point& ur);
    Box(const Rect& r);
    // ...
};
}
```

2. Survey of Advanced *Levelization* Techniques

Escalation

```
// rect.h
#include <point.h>
#include <box.h>
class Rect {
    // ...
public:
    Rect();
    Rect(const Point& o,
          int w, int l);
    Rect(const Box& b);
    // ...
};
```

```
// box.h
#include <point.h>
#include <rect.h>
class Box {
    // ...
public:
    Box();
    Box(const Point& ll,
         const Point& ur);
    Box(const Rect& r);
    // ...
};
```

2. Survey of Advanced *Levelization* Techniques

Escalation

```
// rect.h
#include <point.h>
#include <box.h>
class Rect {
    // ...
public:
    Rect();
    Rect(const Point& o,
          int w, int l);
    Rect(const Box& b);
    // ...
};
```

```
// box.h
#include <point.h>
#include <rect.h>
class Box {
    // ...
public:
    Box();
    Box(const Point& ll,
         const Point& ur);
    Box(const Rect& r);
    // ...
};
```

2. Survey of Advanced *Levelization* Techniques

Escalation

```
// rect.h
#include <point.h>
class Box;
class Rect {
    // ...
public:
    Rect();
    Rect(const Point& o,
          int w, int l);
    Rect(const Box& b);
    // ...
};
```

```
// box.h
#include <point.h>
class Rect;
class Box {
    // ...
public:
    Box();
    Box(const Point& ll,
         const Point& ur);
    Box(const Rect& r);
    // ...
};
```

2. Survey of Advanced *Levelization* Techniques

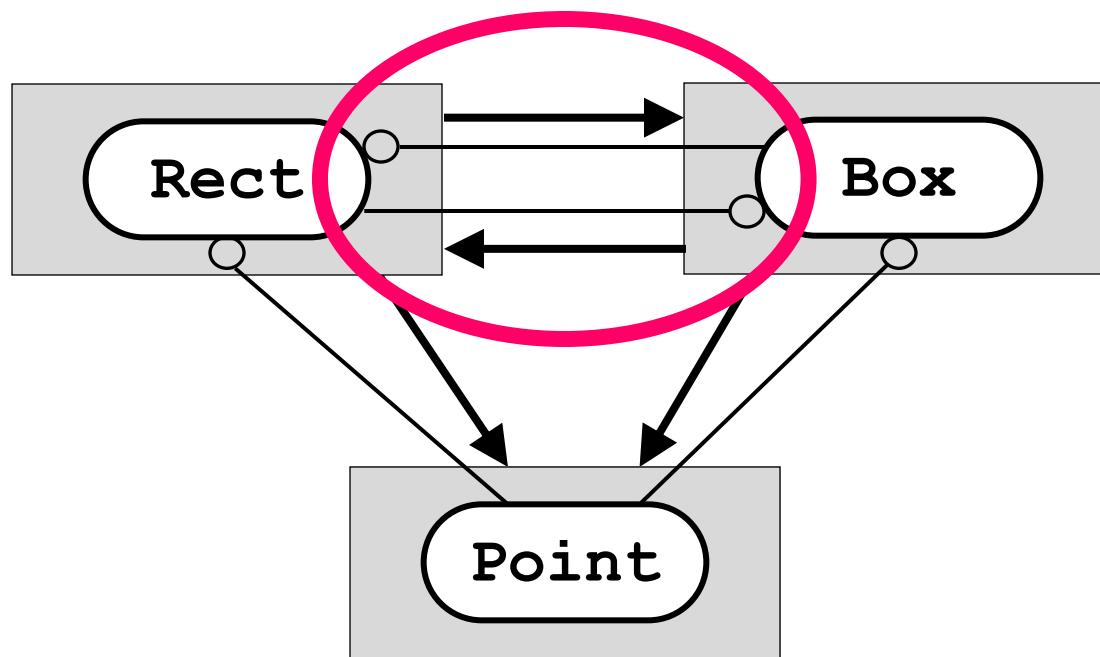
Escalation

```
// rect.h
#include <point.h>
class Box;
class Rect {
    // ...
public:
    Rect();
    Rect(const Point& o,
          int w, int l);
    Rect(const Box& b);
    // ...
};
```

```
// box.h
#include <point.h>
class Rect;
class Box {
    // ...
public:
    Box();
    Box(const Point& ll,
         const Point& ur);
    Box(const Rect& r);
    // ...
};
```

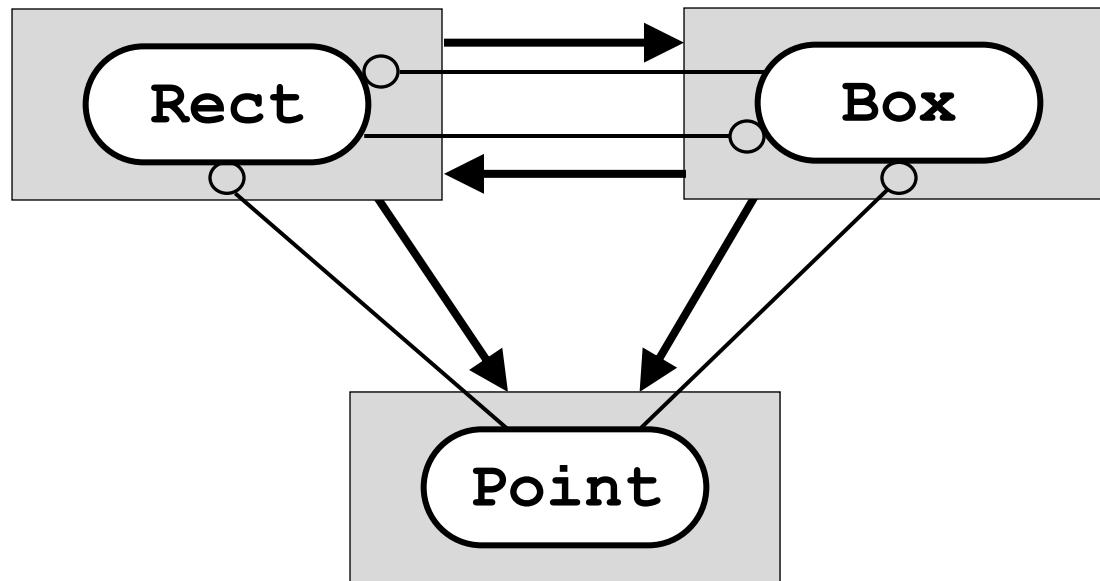
2. Survey of Advanced *Levelization* Techniques

Escalation



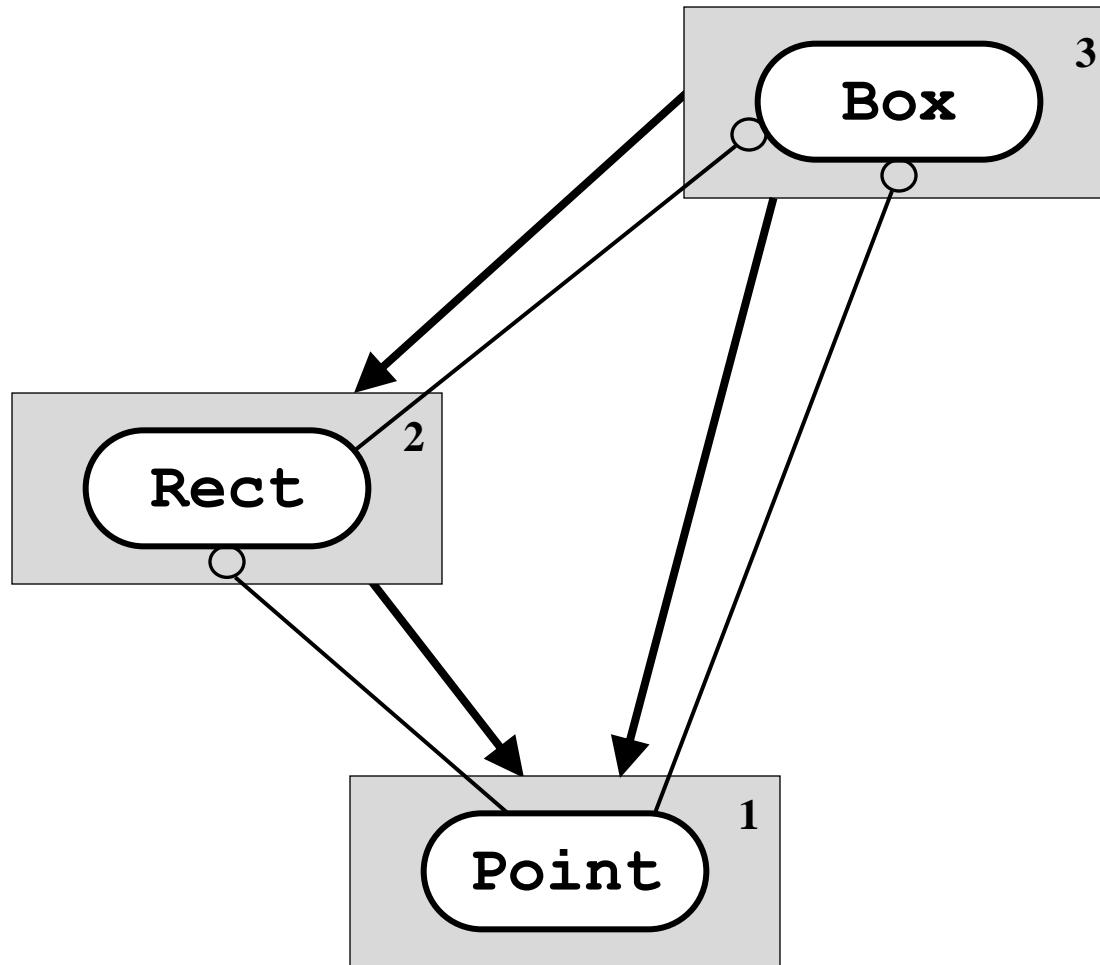
2. Survey of Advanced *Levelization* Techniques

Escalation



2. Survey of Advanced *Levelization* Techniques

Escalation



2. Survey of Advanced *Levelization* Techniques

Escalation

```
// rect.h
#include <point.h>
class Rect {
    // ...
public:
    Rect();
    Rect(const Point& o,
          int w, int l);
    // ...
};
```

```
// box.h
#include <point.h>
#include <rect.h>
class Box {
    // ...
public:
    Box();
    Box(const Point& ll,
         const Point& ur);
    Box(Rect& r);
    // ...
operator Rect() const;
    // ...
};
```

2. Survey of Advanced *Levelization* Techniques

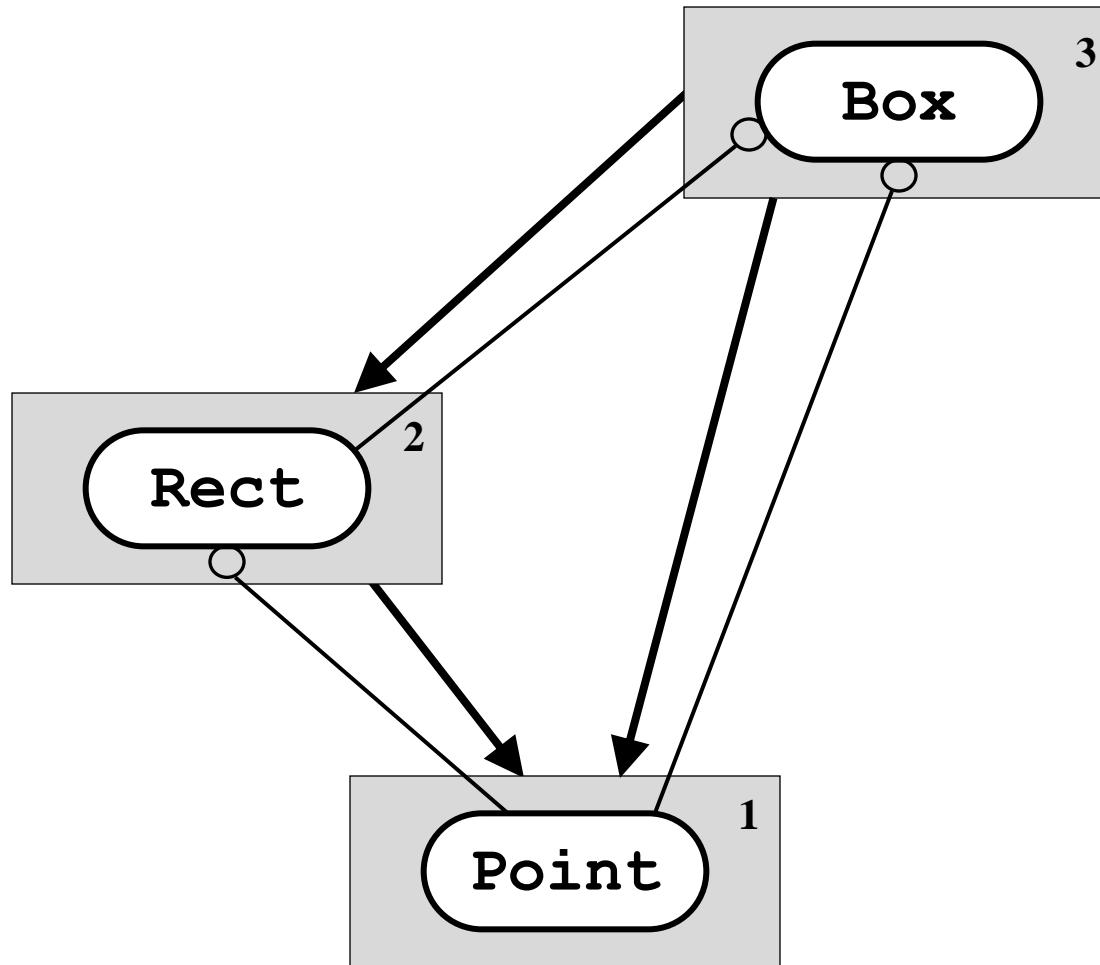
Escalation

```
// rect.h
#include <point.h>
class Rect {
    // ...
public:
    Rect();
    Rect(const Point& o,
          int w, int l);
    // ...
};
```

```
// box.h
#include <point.h>
//include <rect.h>
class Box {
    // ...
public:
    Box();
    Box(const Point& ll,
         const Point& ur);
    Box(Rect& r);
    // ...
    operator Rect() const;
    // ...
};
```

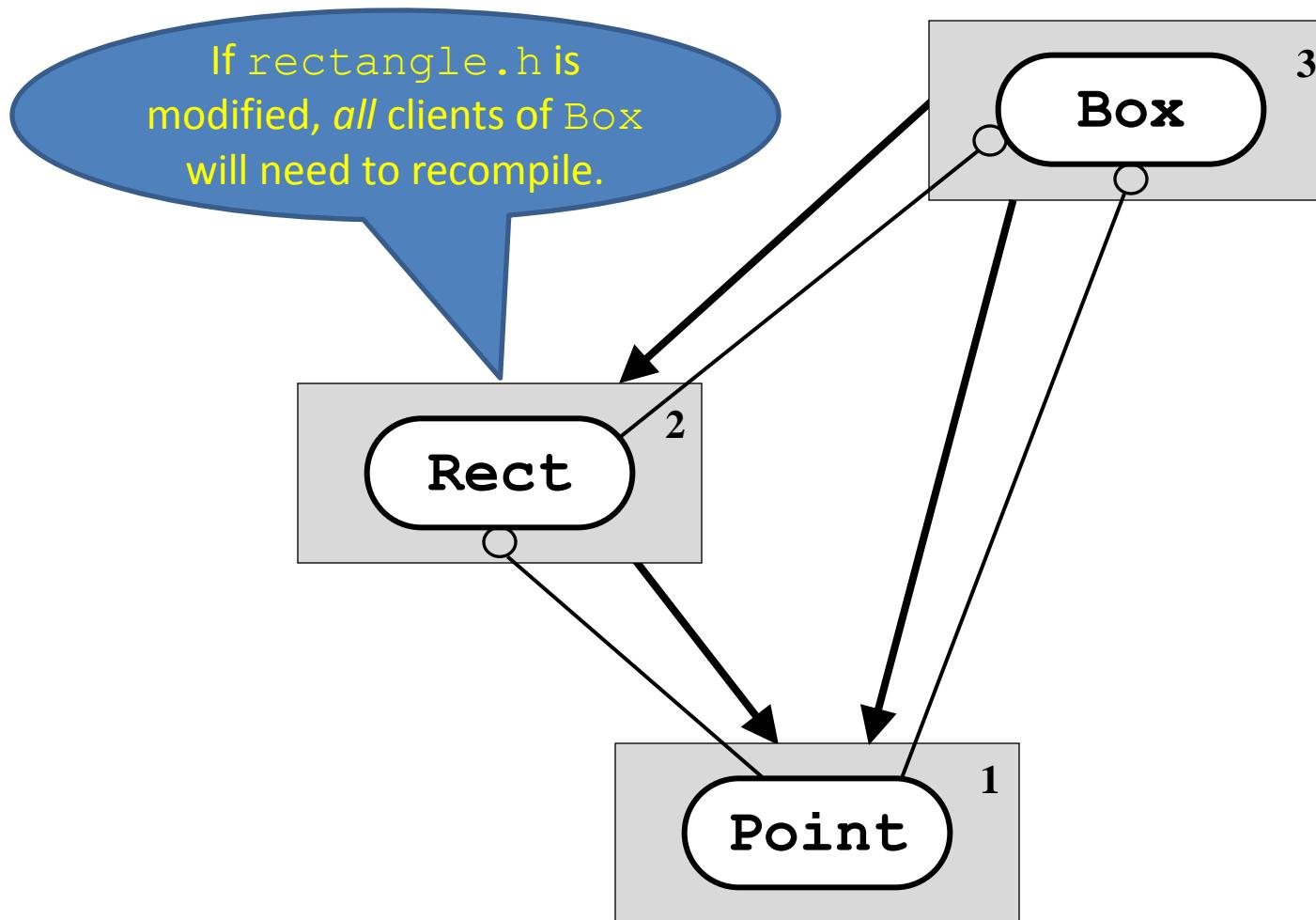
2. Survey of Advanced *Levelization* Techniques

Escalation



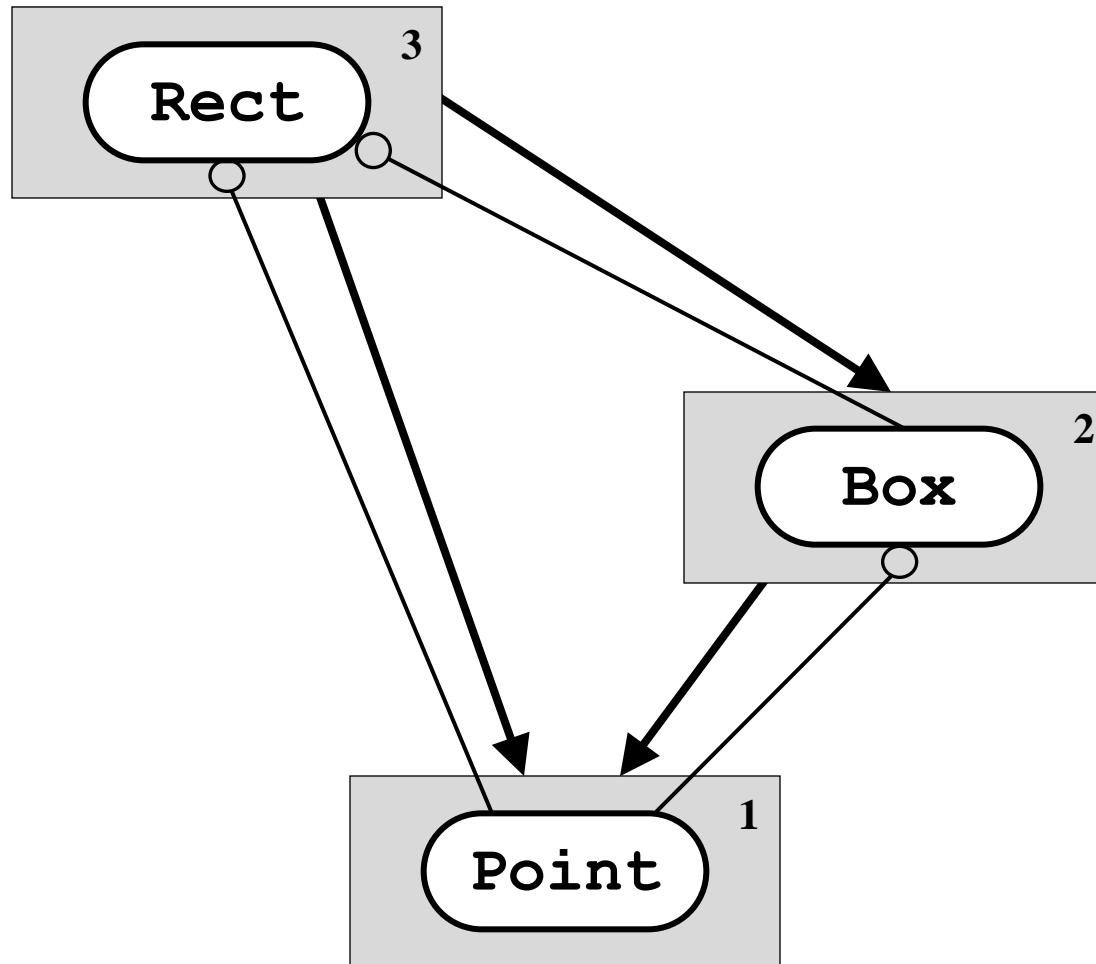
2. Survey of Advanced *Levelization* Techniques

Escalation



2. Survey of Advanced *Levelization* Techniques

Escalation



2. Survey of Advanced *Levelization* Techniques

Escalation

```
// rect.h
#include <point.h>
class Box;
class Rect {
    // ...
public:
    Rect();
    Rect(const Point& o,
          int w, int l);
    Rect(const Box& b);
    // ...
    operator Box() const;
    // ...
};
```

```
// box.h
#include <point.h>
class Box {
    // ...
public:
    Box();
    Box(const Point& ll,
         const Point& ur);
    // ...
};
```

2. Survey of Advanced *Levelization* Techniques

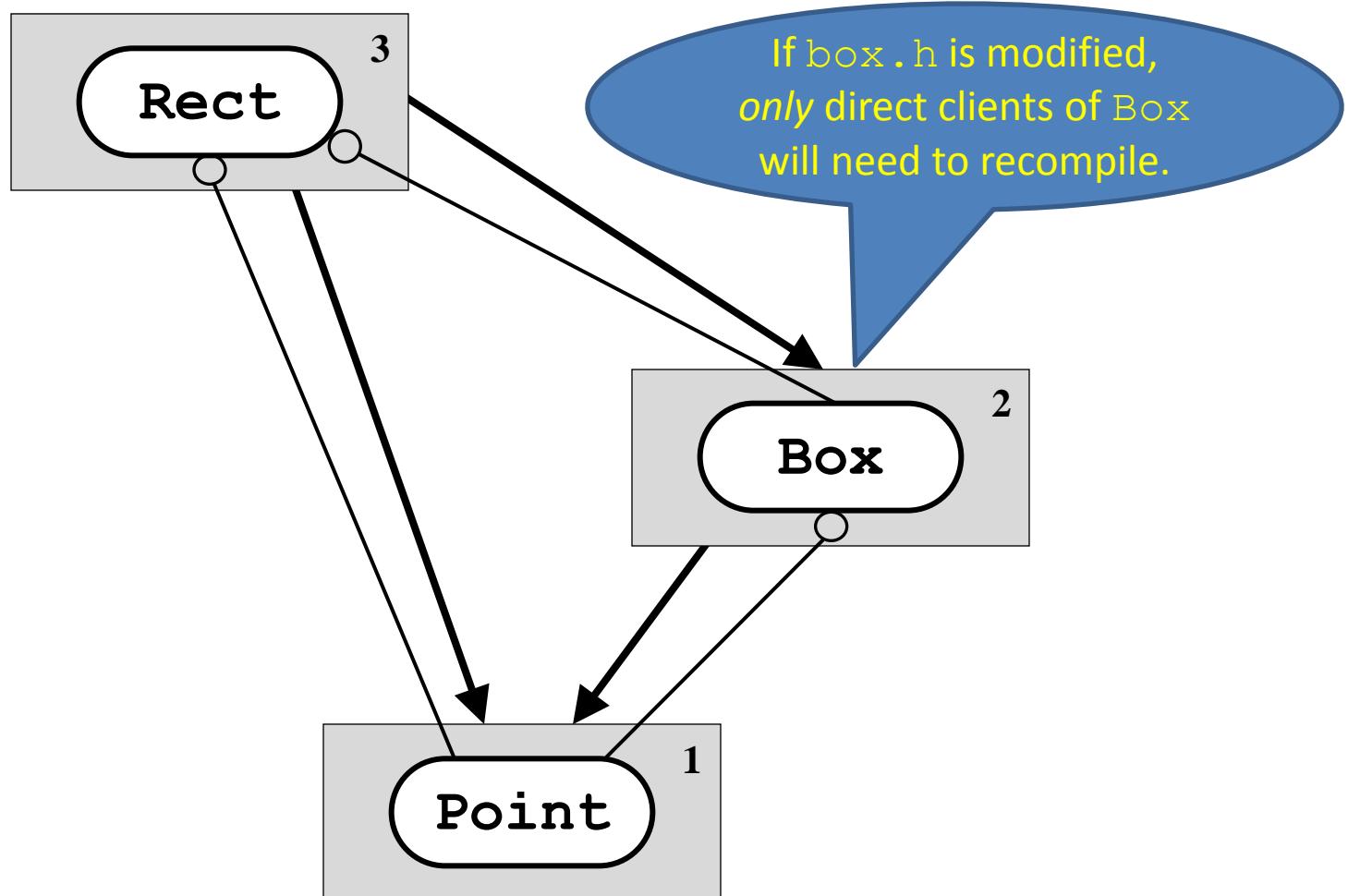
Escalation

```
// rect.h
#include <point.h>
class Box;
class Rect {
    // ...
public:
    Rect();
    Rect(const Point& o,
          int w, int l);
    Rect(const Box& b);
    // ...
    operator Box() const;
    // ...
};
```

```
// box.h
#include <point.h>
class Box {
    // ...
public:
    Box();
    Box(const Point& ll,
         const Point& ur);
    // ...
};
```

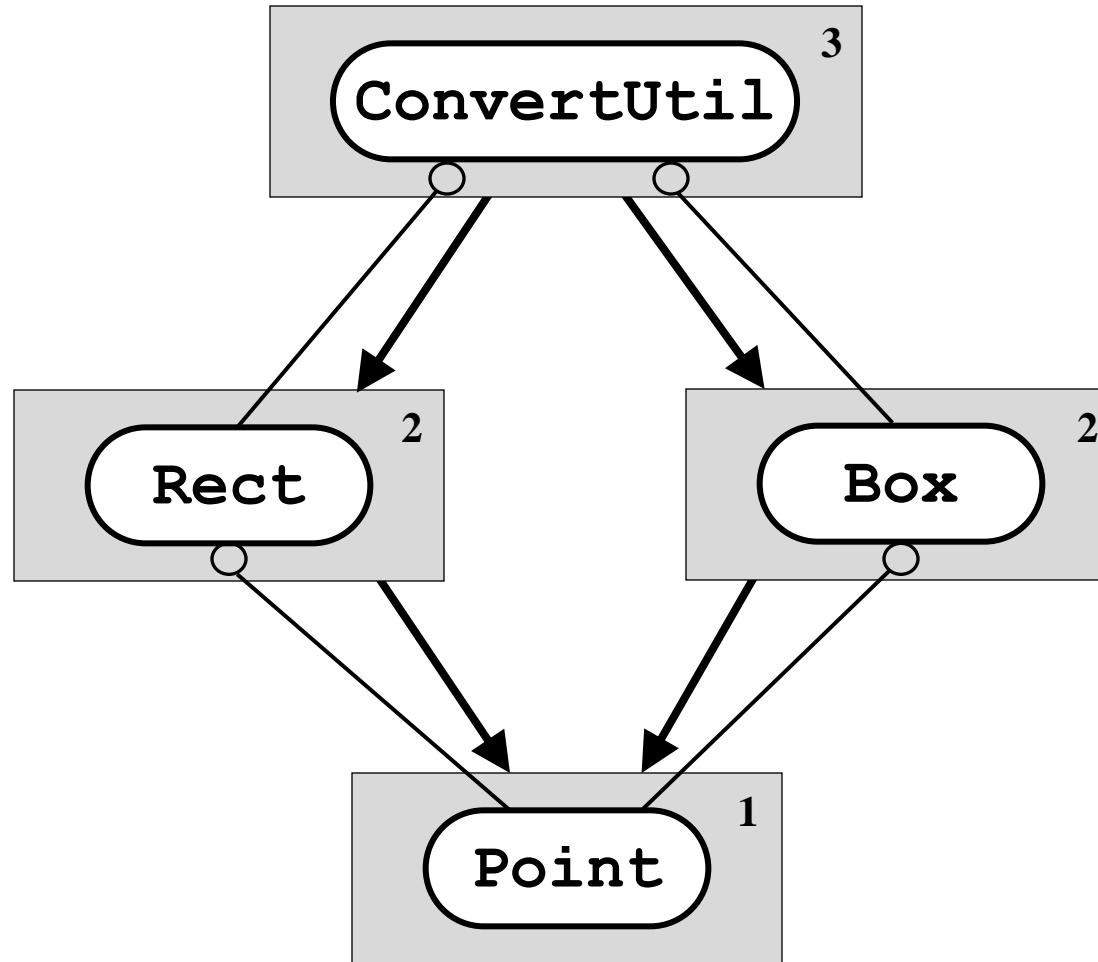
2. Survey of Advanced *Levelization* Techniques

Escalation



2. Survey of Advanced *Levelization* Techniques

Escalation



2. Survey of Advanced *Levelization* Techniques

Escalation

```
// convertutil.h
#include <rect.h>
#include <box.h>
#include <point.h>
struct ConvertUtil {
    static Box boxFromRect(const Rect& r);
    static Rect rectFromBox(const Box& b);
    // ...
};
```

```
// rect.h
#include <point.h>
class Rect {
    // ...
public:
    Rect();
    Rect(const Point& o,
          int w, int l);
    // ...
};
```

```
// box.h
#include <point.h>
class Box {
    // ...
public:
    Box();
    Box(const Point& ll,
         const Point& ur);
    // ...
};
```

2. Survey of Advanced *Levelization* Techniques

Escalation

```
// convertutil.h
#include <rect.h>
#include <box.h>
#include <point.h>
struct ConvertUtil {
    static Box boxFromRect(const Rect& r);
    static Rect rectFromBox(const Box& b);
    // ...
};
```

```
// rect.h
#include <point.h>
class Rect {
    // ...
public:
    Rect();
    Rect(const Point& o,
          int w, int l);
    // ...
};
```

```
// box.h
#include <point.h>
class Box {
    // ...
public:
    Box();
    Box(const Point& ll,
         const Point& ur);
    // ...
};
```

2. Survey of Advanced *Levelization* Techniques

Escalation

```
// convertutil.h
#include <rect.h>
#include <box.h>
#include <point.h>
struct ConvertUtil {
    static Box boxFromRect(const Rect& r);
    static Rect rectFromBox(const Box& b);
    // ...
};
```

in order to support
inline functions

```
// rect.h
#include <point.h>
class Rect {
    // ...
public:
    Rect();
    Rect(const Point& o,
          int w, int l);
    // ...
};
```

```
// box.h
#include <point.h>
class Box {
    // ...
public:
    Box();
    Box(const Point& ll,
         const Point& ur);
    // ...
};
```

2. Survey of Advanced *Levelization* Techniques

Escalation

```
// convertutil.h
#include <rect.h>
#include <box.h>
#include <point.h>
struct ConvertUtil {
    static Box boxFromRect(const Rect& r);
    static Rect rectFromBox(const Box& b);
    // ...
};
```

in order to support
inline functions

more on
this later

```
// rect.h
#include <point.h>
class Rect {
    // ...
public:
    Rect();
    Rect(const Point& o,
          int w, int l);
    // ...
};
```

```
// box.h
#include <point.h>
class Box {
    // ...
public:
    Box();
    Box(const Point& ll,
         const Point& ur);
    // ...
};
```

2. Survey of Advanced *Levelization* Techniques

Escalation

```
// convertutil.h
#include <rect.h>
#include <box.h>
#include <point.h>
struct Convertutil {
    static Box boxFromRect(const Rect& r);
    static Rect rectFromBox(const Box& b);
    // ...
};
```

```
// rect.h
#include <point.h>
class Rect {
    // ...
public:
    Rect();
    Rect(const Point& o,
          int w, int l);
    // ...
};
```

```
// box.h
#include <point.h>
class Box {
    // ...
public:
    Box();
    Box(const Point& ll,
         const Point& ur);
    // ...
};
```

2. Survey of Advanced *Levelization* Techniques

Escalation

```
// convertutil.h
#include <rect.h>
#include <box.h>
#include <point.h>
struct Convertutil {
    static Box boxFromRect(const Rect& r);
    static Rect rectFromBox(const Box& b);
    // ...
};
```

in order to avoid
transitive includes

```
// rect.h
#include <point.h>
class Rect {
    // ...
public:
    Rect();
    Rect(const Point& o,
          int w, int l);
    // ...
};
```

```
// box.h
#include <point.h>
class Box {
    // ...
public:
    Box();
    Box(const Point& ll,
         const Point& ur);
    // ...
};
```

2. Survey of Advanced *Levelization* Techniques

Escalation

```
// convertutil.h
#include <rect.h>
#include <box.h>
#include <point.h>
struct Convertutil {
    static Box boxFromRect
    static Rect rectFromBox
    // ...
};
```

in order to avoid
transitive includes

Presumes inline
implementations of
(elided) functions using
Point.

```
// rect.h
#include <point.h>
class Rect {
    // ...
public:
    Rect();
    Rect(const Point& o,
          int w, int l);
    // ...
};
```

```
// box.h
#include <point.h>
class Box {
    // ...
public:
    Box();
    Box(const Point& ll,
         const Point& ur);
    // ...
};
```

2. Survey of Advanced *Levelization* Techniques

Escalation

```
// convertutil.h
#include <rect.h>
#include <box.h>
#include <point.h>
struct Convertutil {
    static Box boxFromRect
    static Rect rectFromBox
    // ...
};
```

in order to avoid
transitive includes

Presumes inline
implementations of
(elided) functions using
Point.

```
// re
inline
#include <rect.h>
class Box {
public:
    Box(const Point& origin, int length, int width);
    const Point& origin() const;
    int length() const;
    int width() const;
private:
    Point origin;
    int length;
    int width;
};
```

```
Box::boxFromRect(const Rect& r)
{
    int length = r.ur.x() - r.ll.x();
    int width = r.ur.y() - r.ll.y();
    Point origin(r.ll.x() + length/2,
                 r.ll.y() + width/2);
    return Box(origin, length, width);
}
```

2. Survey of Advanced *Levelization* Techniques

Escalation

```
// convertutil.h
#include <rect.h>
#include <box.h>
#include <point.h>
struct Convertutil {
    static Box boxFromRect
    static Rect rectFromBox
    // ...
};
```

in order to avoid
transitive includes

Presumes inline
implementations of
(elided) functions using
Point.

```
// rect.h
#include <point.h>
class Rect {
    // ...
public:
    Rect();
    Rect(const Point& o,
          int w, int l);
    // ...
};
```

```
// box.h
#include <point.h>
class Box {
    // ...
public:
    Box();
    Box(const Point& ll,
         const Point& ur);
    // ...
};
```

2. Survey of Advanced *Levelization* Techniques

Escalation

```
// convertutil.h
#include <rect.h>
#include <box.h>

struct ConvertUtil {
    static Box boxFromRect
    static Rect rectFromBox
    // ...
};
```

in order to avoid
transitive includes

Presumes inline
implementations of
(elided) functions using
Point.

```
// rect.h
#include <point.h>
class Rect {
    // ...
public:
    Rect();
    Rect(const Point& o,
          int w, int l);
    // ...
};
```

```
// box.h
#include <point.h>
class Box {
    // ...
public:
    Box();
    Box(const Point& ll,
         const Point& ur);
    // ...
};
```

2. Survey of Advanced *Levelization* Techniques

Escalation

```
// convertutil.h
#include <rect.h>
#include <box.h>

struct Convertutil {
    static Box boxFromRect(const Rect& r);
    static Rect rectFromBox(const Box& b);
    // ...
};
```

```
// rect.h
#include <point.h>
class Rect {
    // ...
public:
    Rect();
    Rect(const Point& o,
          int w, int l);
    // ...
};
```

```
// box.h
#include <point.h>
class Box {
    // ...
public:
    Box();
    Box(const Point& ll,
         const Point& ur);
    // ...
};
```

2. Survey of Advanced *Levelization* Techniques

Escalation

```
// convertutil.h
#include <rect.h>
#include <box.h>

struct Convertutil {
    static Box boxFromRect(const Rect& r);
    static Rect rectFromBox(const Box& b);
    // ...
};
```

```
// rect.h
class Point;
class Rect {
    // ...
public:
    Rect();
    Rect(const Point& o,
          int w, int l);
    // ...
};
```

```
// box.h
class Point;
class Box {
    // ...
public:
    Box();
    Box(const Point& ll,
         const Point& ur);
    // ...
};
```

2. Survey of Advanced *Levelization* Techniques

Escalation

```
// convertutil.h
#include <rect.h>
#include <box.h>

class ConvertUtil {
public:
    Box FromRect(const Rect& r);
    Rect FromBox(const Box& b);
```

**Convertutil.h
No Longer Compiles!**

```
// convertutil.h
class Rect {
    // ...
public:
    Rect();
    Rect(const Point& o,
          int w, int l);
    // ...
};
```

```
Box();
Box(const Rect& r,
     const Point& ur);
// ...
};
```

2. Survey of Advanced *Levelization* Techniques

Escalation

```
// convertutil.h
#include <rect.h>
#include <box.h>
#include <point.h>
struct Convertutil {
    static Box boxFromRect
    static Rect rectFromBox
    // ...
};
```

in order to avoid
transitive includes

Presumes inline
implementations of
(elided) functions using
Point.

```
// rect.h
class Point;
class Rect {
    // ...
public:
    Rect();
    Rect(const Point& o,
          int w, int l);
    // ...
};
```

```
// box.h
class Point;
class Box {
    // ...
public:
    Box();
    Box(const Point& ll,
         const Point& ur);
    // ...
};
```

2. Survey of Advanced *Levelization* Techniques

Escalation

```
// convertutil.h
#include <rect.h>
#include <box.h>
#include <point.h>
struct ConvertUtil {
    static Box boxFromRect(const Rect& r);
    static Rect rectFromBox(const Box& b);
    // ...
};
```

in order to avoid
transitive includes

```
// rect.h
class Point;
class Rect {
    // ...
public:
    Rect();
    Rect(const Point& p);
    // ...
};
```

```
// box.h
class Point;
class Box {
```

convertutil.h
Again Compiles!

2. Survey of Advanced *Levelization* Techniques

Escalation

```
// convertutil.h
#include <rect.h>
#include <box.h>
#include <point.h> • •
struct ConvertUtil {
    static Box boxFromRect(const Rect& r);
    static Rect rectFromBox(const Box& b);
    // ...
};
```

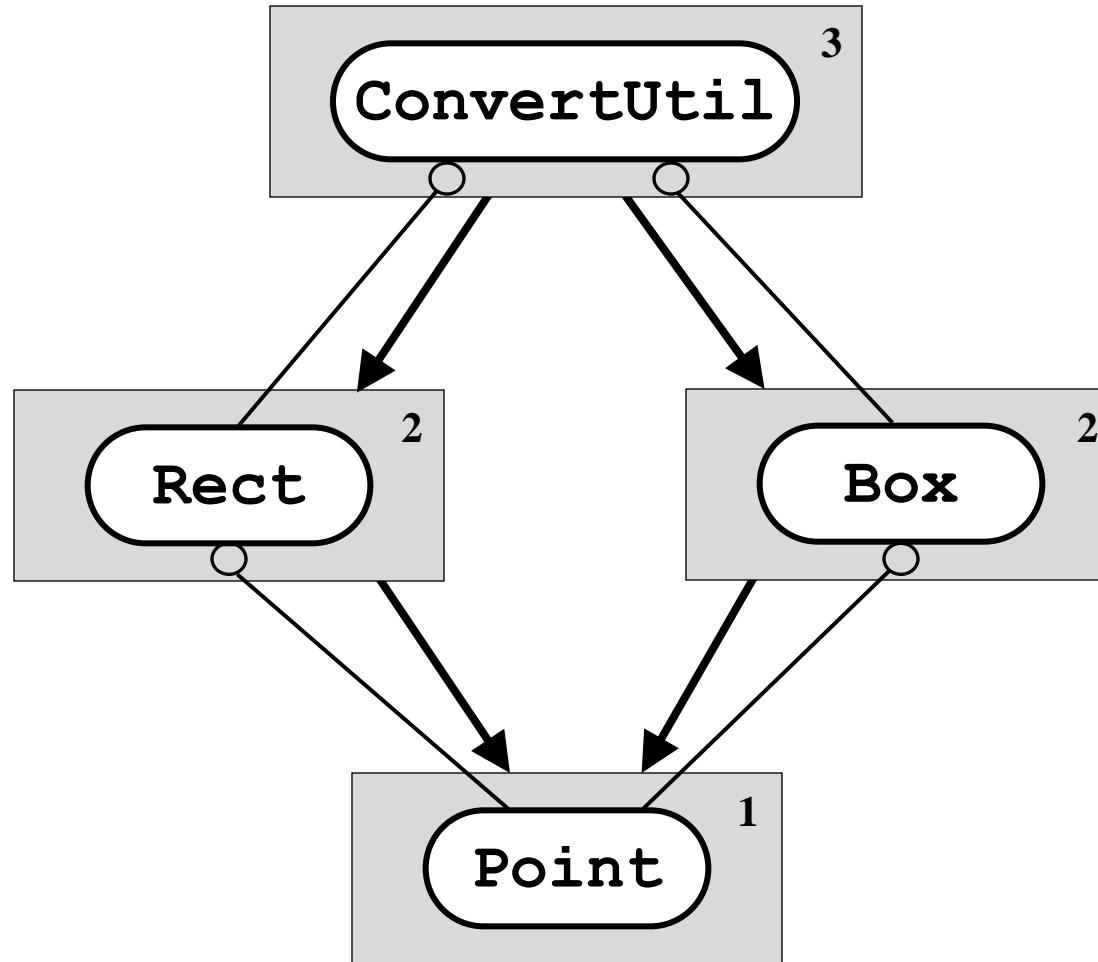
in order to avoid
transitive includes

```
// rect.h
class Point;
class Rect {
    // ...
public:
    Rect();
    Rect(const Point& o,
          int w, int l);
    // ...
};
```

```
// box.h
class Point;
class Box {
    // ...
public:
    Box();
    Box(const Point& ll,
         const Point& ur);
    // ...
};
```

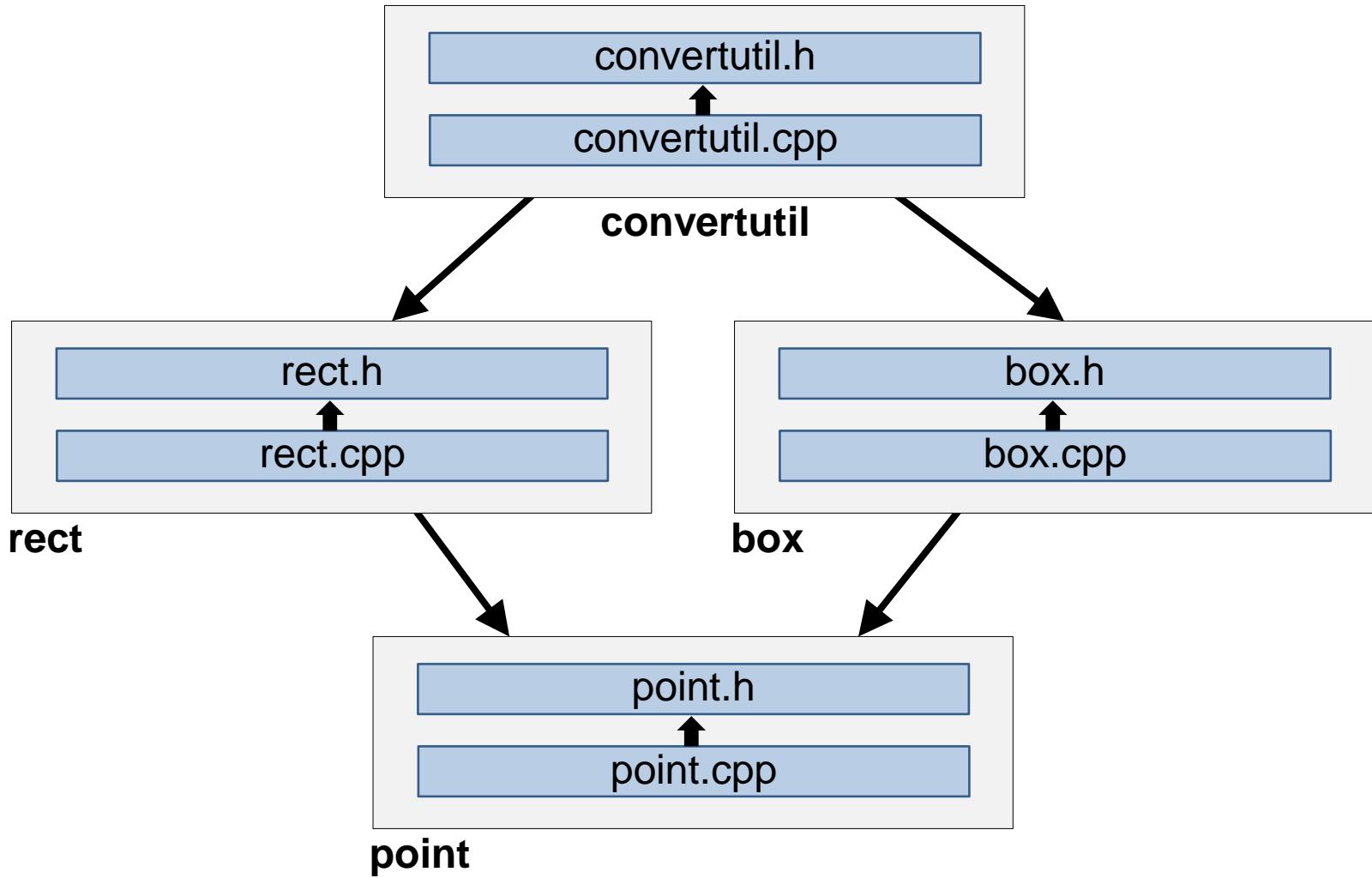
2. Survey of Advanced *Levelization* Techniques

Escalation



2. Survey of Advanced *Levelization* Techniques

Escalation



2. Survey of Advanced *Levelization* Techniques

Escalation

Discussion?

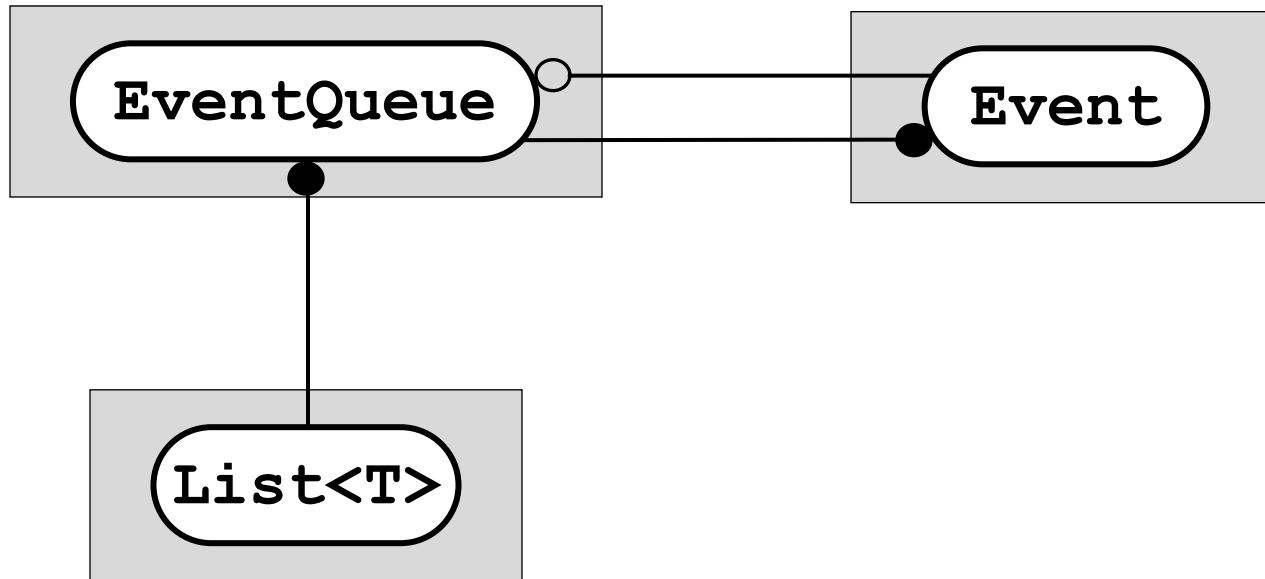
2. Survey of Advanced *Levelization* Techniques

Demotion

Demotion – Moving common functionality lower in the physical hierarchy.

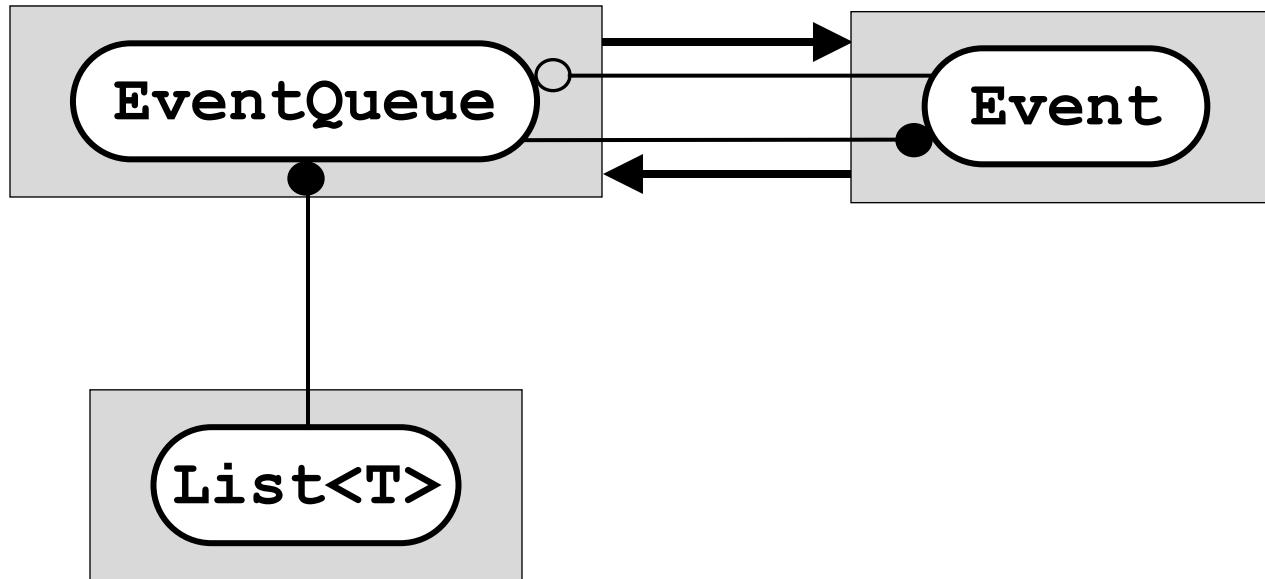
2. Survey of Advanced *Levelization* Techniques

Demotion



2. Survey of Advanced *Levelization* Techniques

Demotion



2. Survey of Advanced *Levelization* Techniques

Demotion

```
// eventqueue.h
#include <event.h>
#include <list.h>
class EventQueue {
    List<Event> d_events;

    // common event info.

    // other stuff

public:
    // ...
};
```

```
// event.h
class Event {
    EventQueue *d_common_p;

    // event-specific
    // information

public:
    // ...
    int commonData() const;
    // ...
};
```

2. Survey of Advanced *Levelization* Techniques

Demotion

```
// eventqueue.h
#include <event.h>
#include <list.h>
class EventQueue {
    List<Event> d_events;

    // common event info.

    // other stuff

public:
    // ...
};

}
```

```
// event.h
class Event {
    EventQueue *d_common_p;

    // event-specific
    // information

public:
    // ...
    int commonData() const;
    // ...
};

}
```

2. Survey of Advanced *Levelization* Techniques

Demotion

```
// eventqueue.h
#include <event.h>
#include <list.h>
class EventQueue {
    List<Event> d_events;

    // common event info.

    // other stuff

public:
    // ...
};

}
```

```
// event.h
class Event {
    EventQueue *d_common_p;

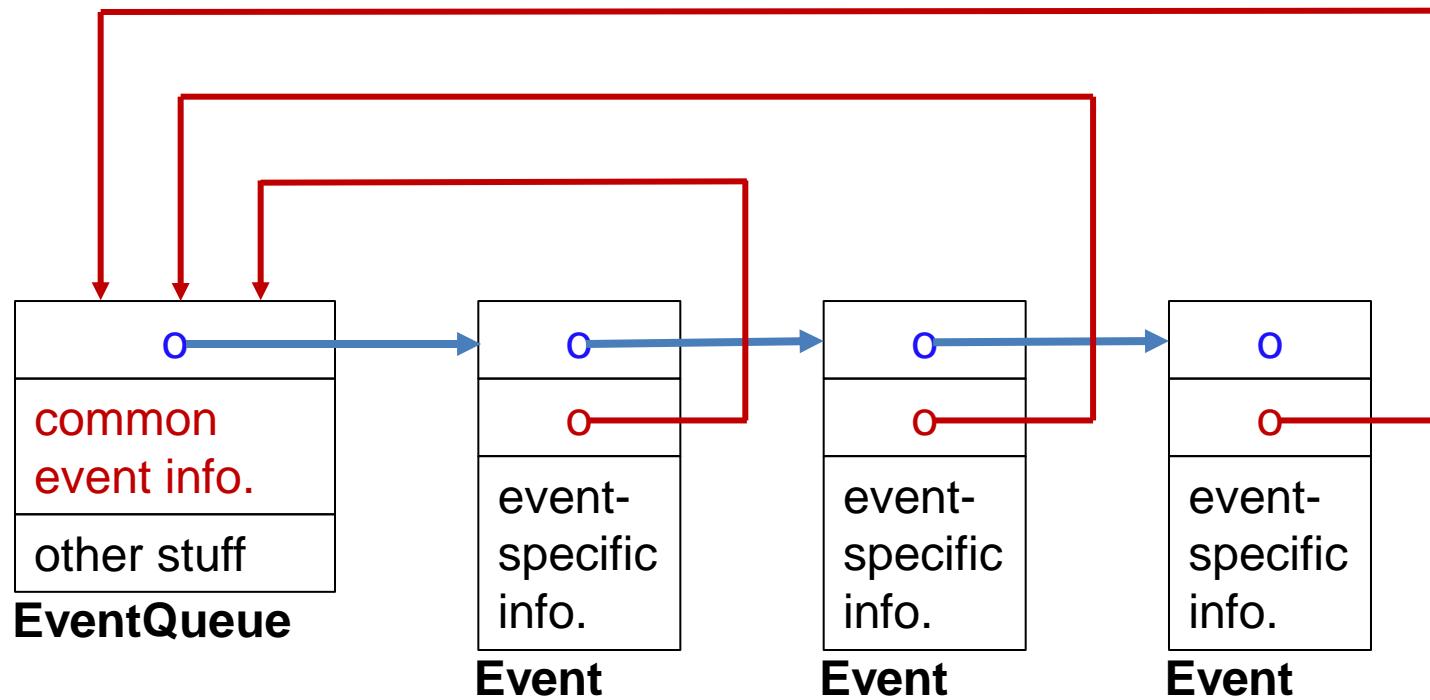
    // event-specific
    // information

public:
    // ...
    int commonData() const;
    // ...
};

}
```

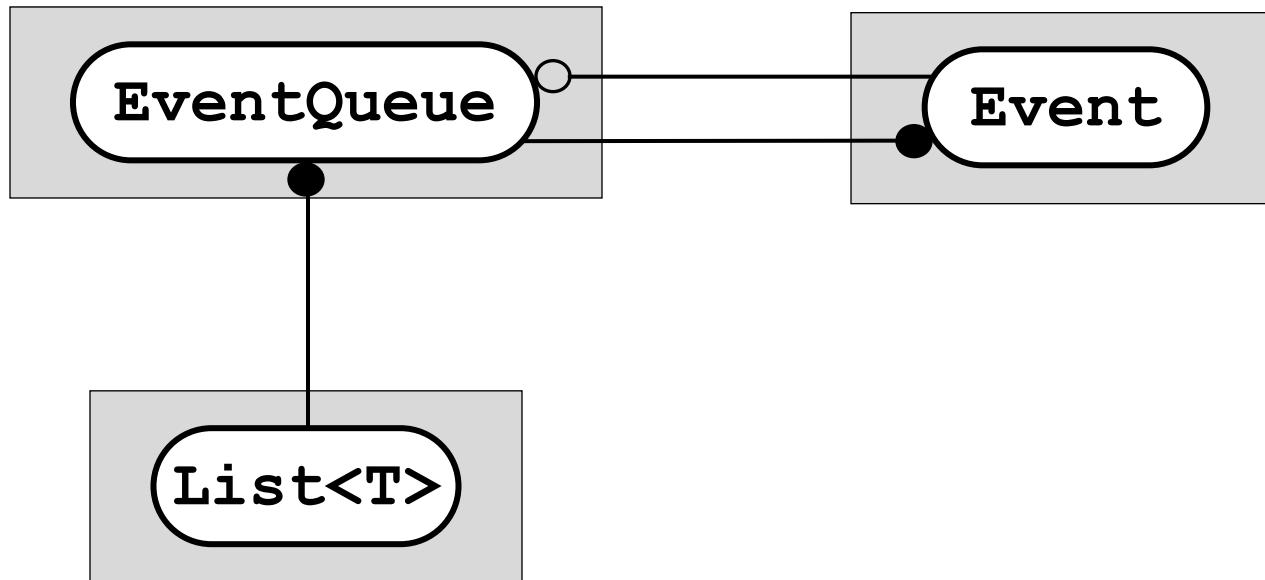
2. Survey of Advanced *Levelization* Techniques

Demotion



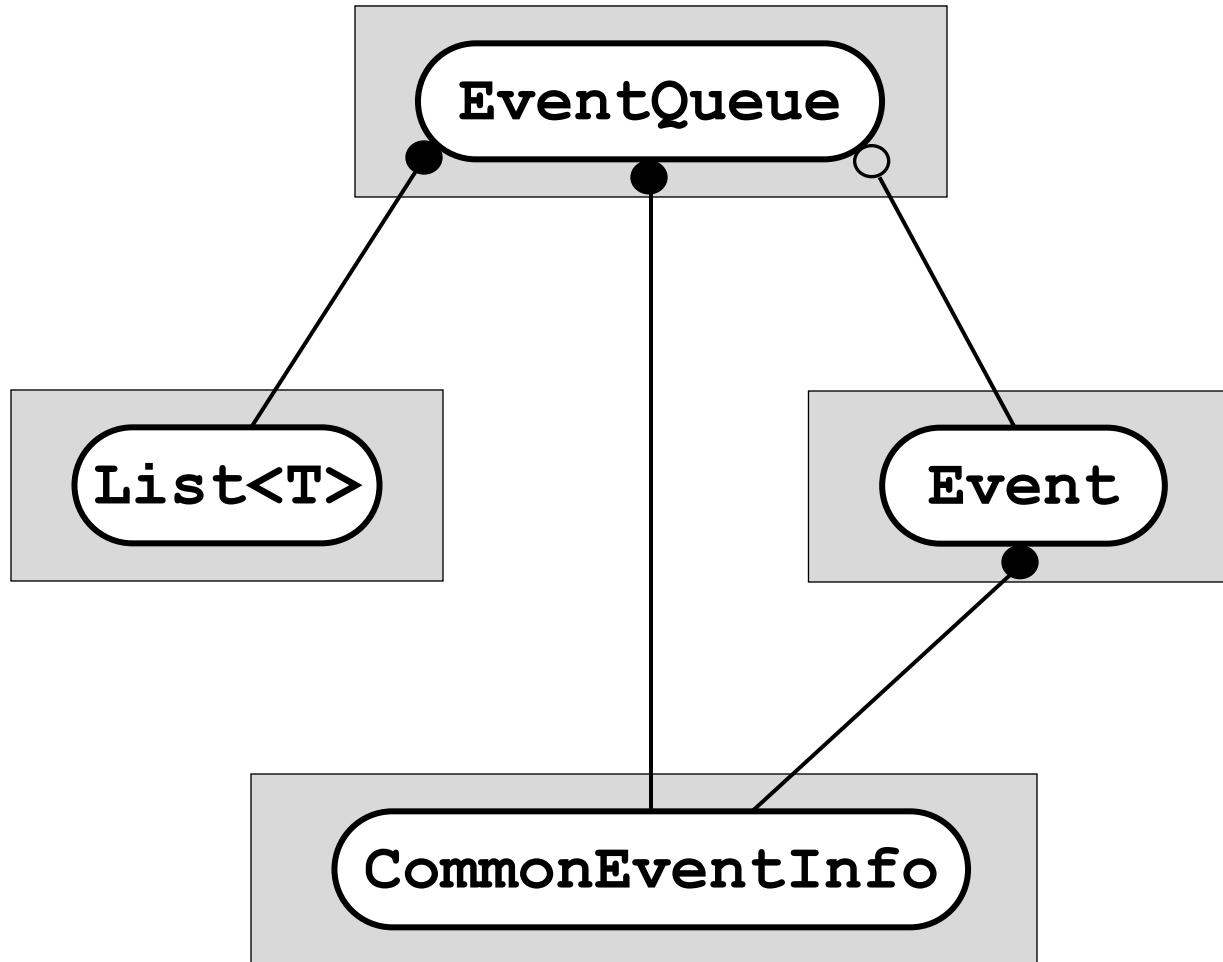
2. Survey of Advanced *Levelization* Techniques

Demotion



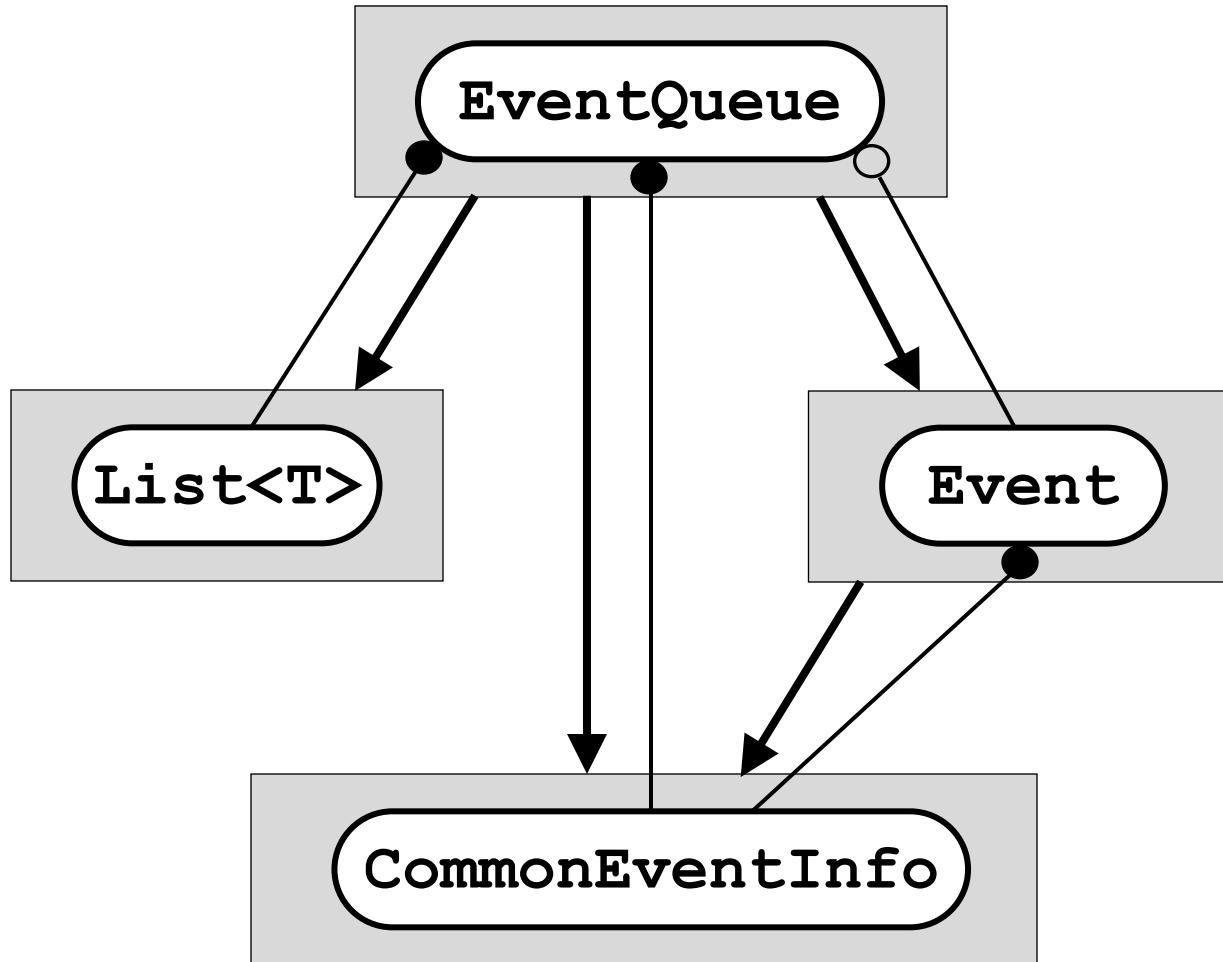
2. Survey of Advanced *Levelization* Techniques

Demotion



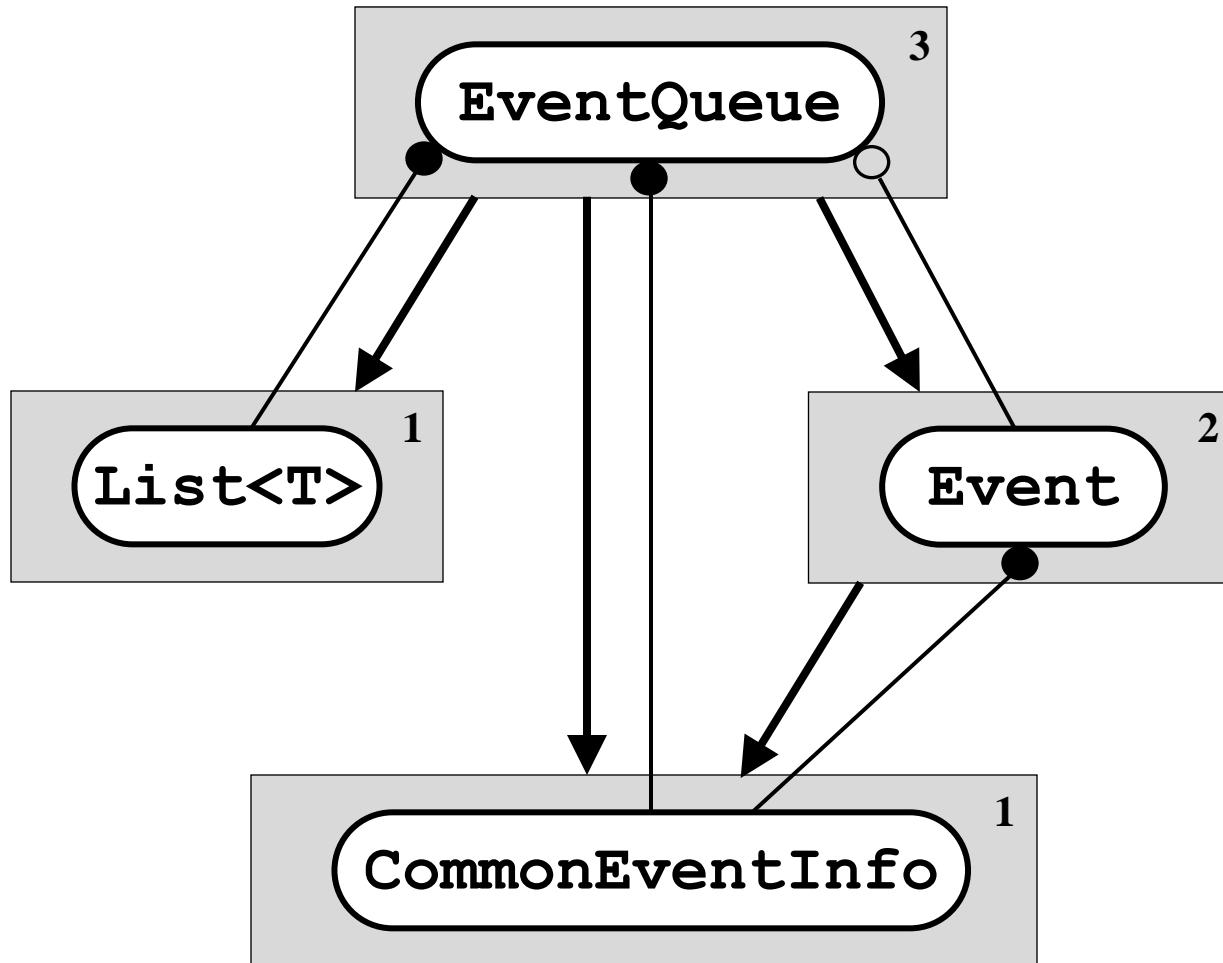
2. Survey of Advanced *Levelization* Techniques

Demotion



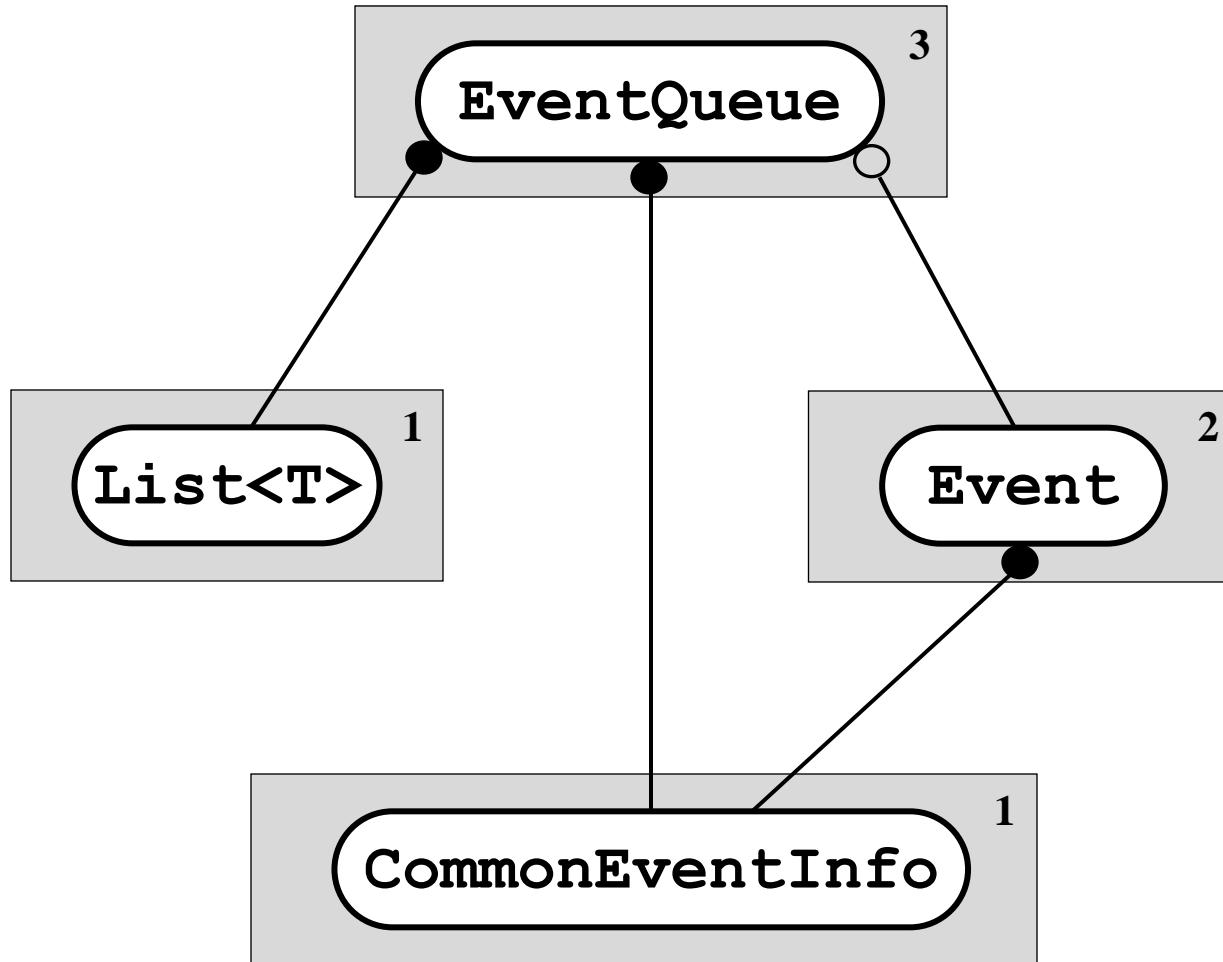
2. Survey of Advanced *Levelization* Techniques

Demotion



2. Survey of Advanced *Levelization* Techniques

Demotion



2. Survey of Advanced *Levelization* Techniques

Demotion

```
// eventqueue.h
#include <event.h>
#include <list.h>
class EventQueue {
    List<Event> d_events;

    // common event info.

    // other stuff

public:
    // ...
};

}
```

```
// event.h
class Event {
    EventQueue *d_common_p;

    // event-specific
    // information

public:
    // ...
    int commonData() const;
    // ...
};

}
```

2. Survey of Advanced *Levelization* Techniques

Demotion

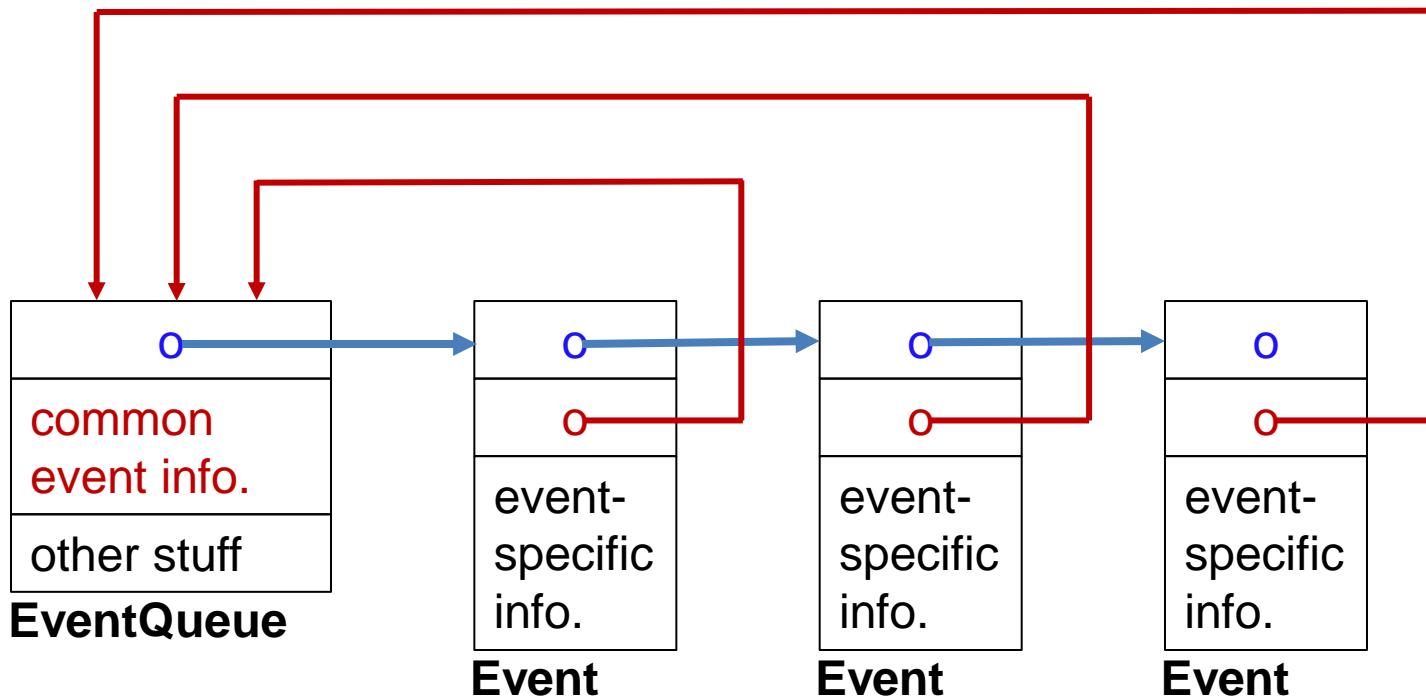
```
// eventqueue.h
#include <event.h>
#include <list.h>
#include <commoneventinfo.h>
class EventQueue {
    List<Event> d_events;
    CommonEventInfo d_info;
    // other stuff
public:
    // ...
};
```

```
// event.h
#include <commoneventinfo.h>
class Event {
    CommonEventInfo *d_common_p;
    // event-specific
    // information
public:
    // ...
    int commonData() const;
    // ...
};
```

```
// commoneventinfo.h
class CommonEventInfo {
    // common event info.
public:
    // ...
};
```

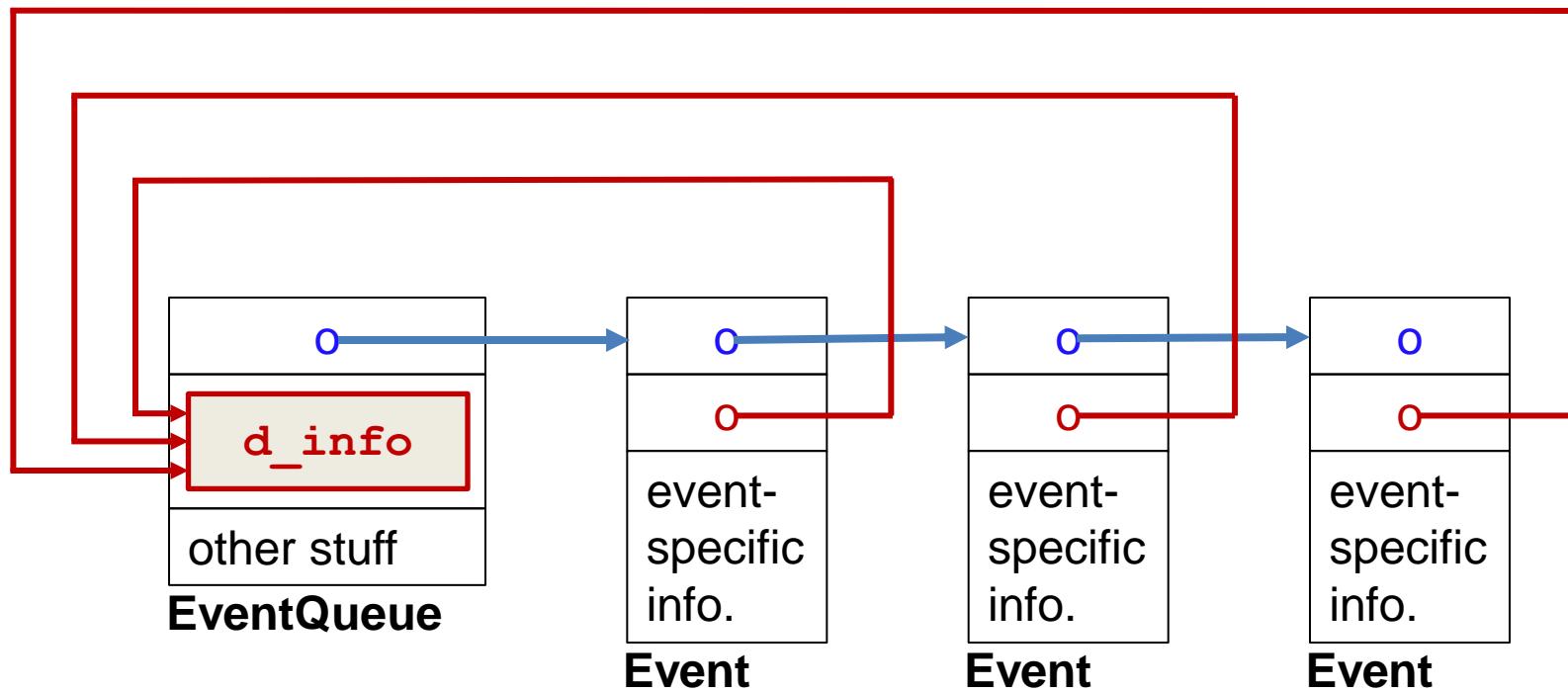
2. Survey of Advanced *Levelization* Techniques

Demotion



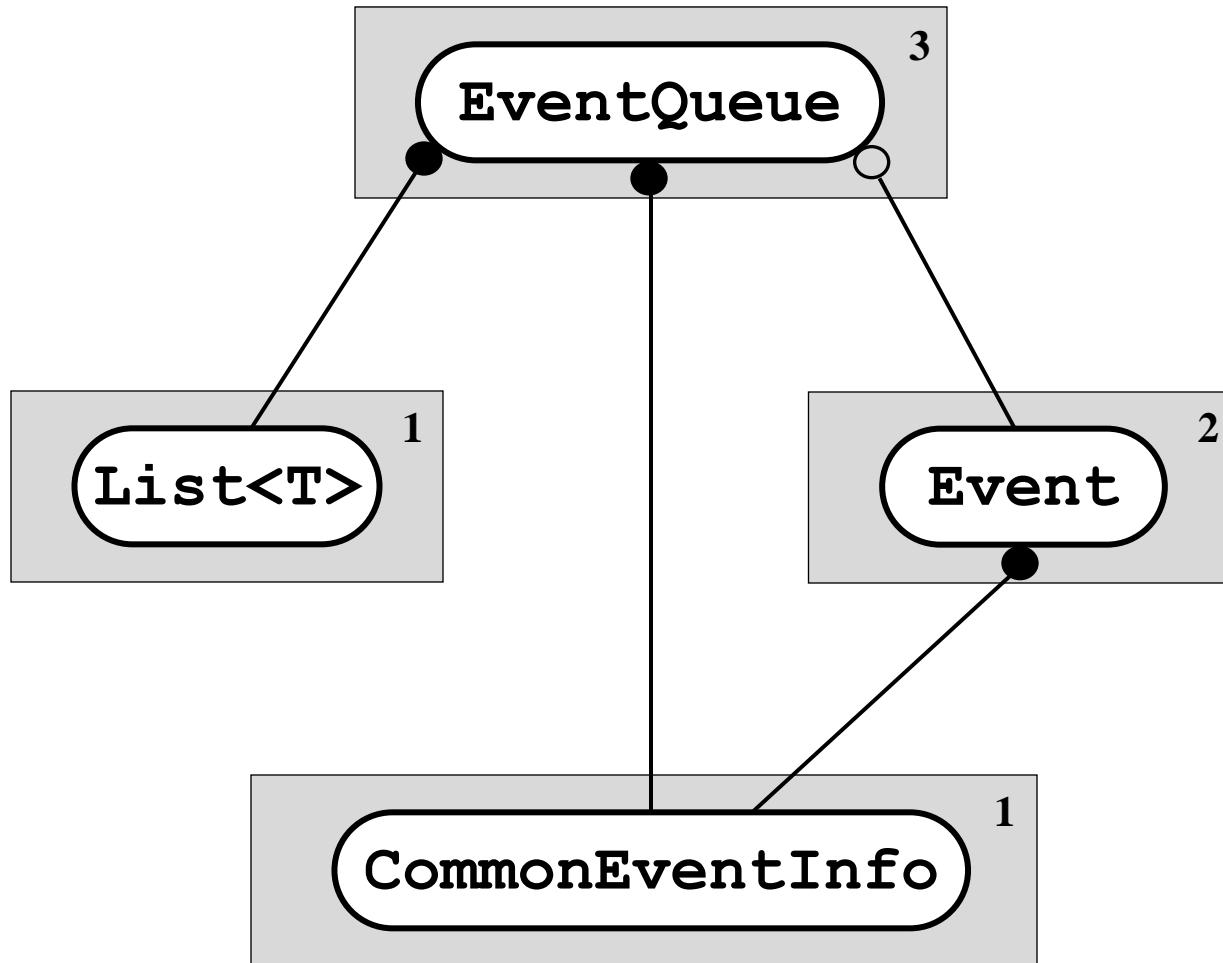
2. Survey of Advanced *Levelization* Techniques

Demotion



2. Survey of Advanced *Levelization* Techniques

Demotion



2. Survey of Advanced *Levelization* Techniques

Demotion

Discussion?

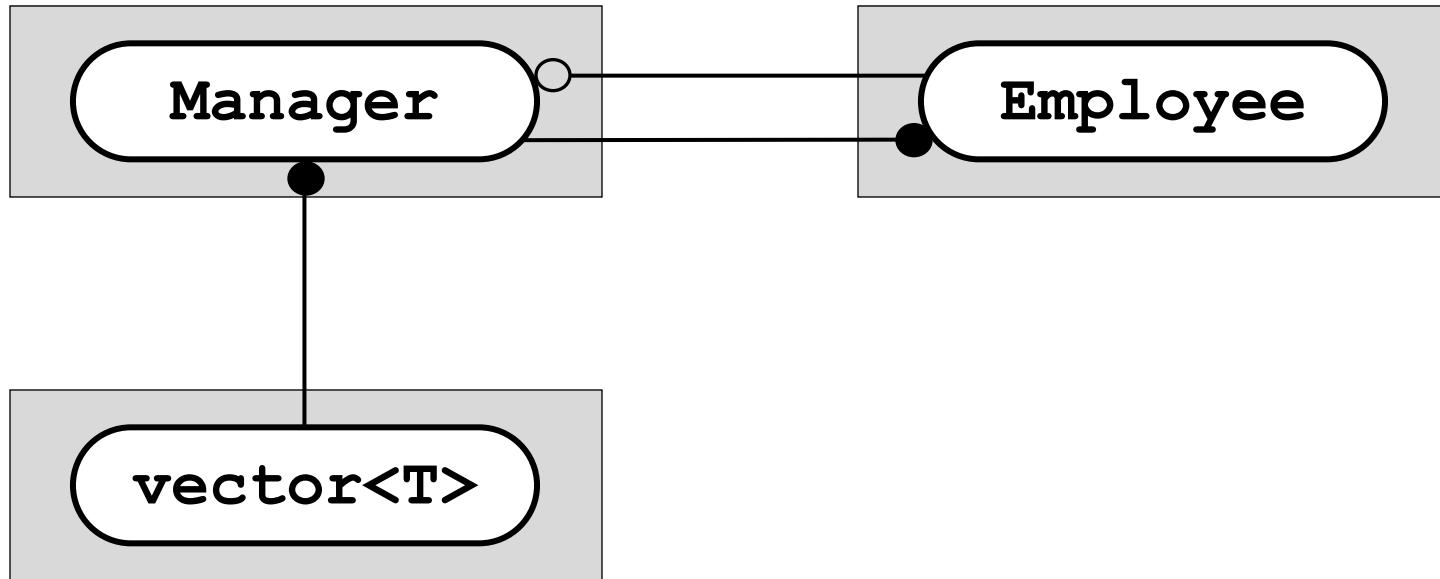
2. Survey of Advanced *Levelization* Techniques

Opaque Pointers

Opaque Pointers –
Having an object use
another *in name only*.

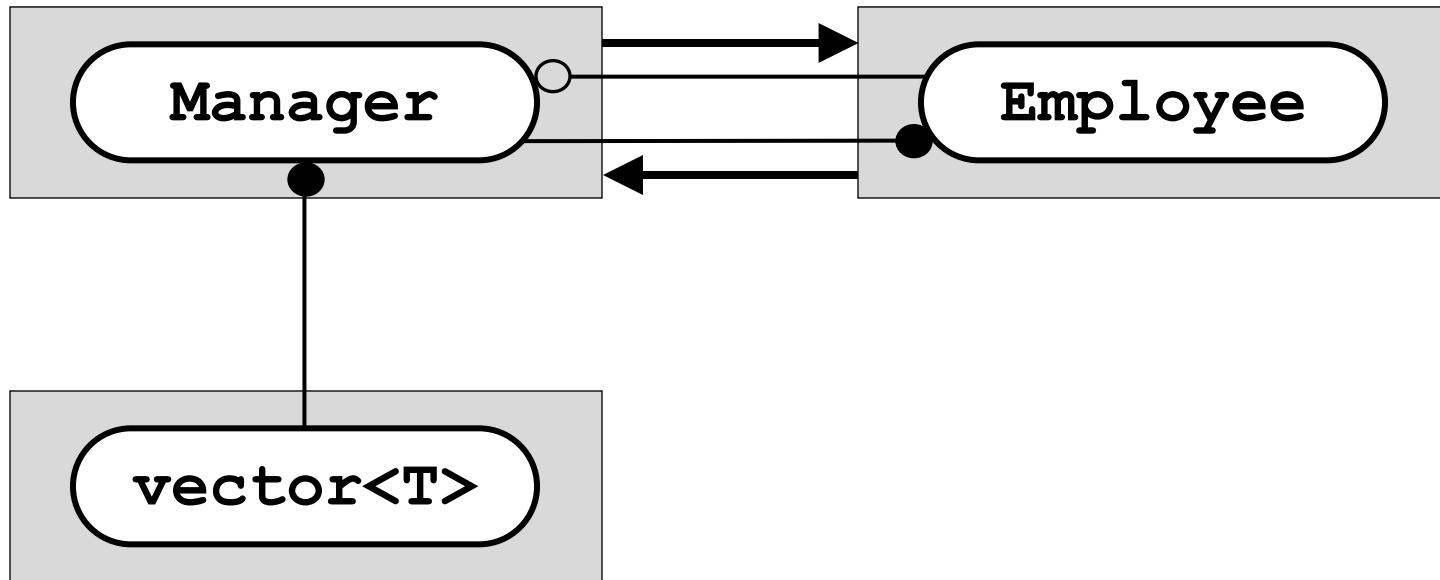
2. Survey of Advanced *Levelization* Techniques

Opaque Pointers



2. Survey of Advanced *Levelization* Techniques

Opaque Pointers



2. Survey of Advanced *Levelization* Techniques

Opaque Pointers

```
// manager.h
#include <employee.h>
#include <vector>
class Manager {
    vector<Employee> d_staff;
public:
    // ...
    int addEmployee(...);
    const Employee&
        employee(int id) const;
    // ...
};
```

```
// employee.h
#include <manager.h>
class Employee {
    Manager *d_boss_p;
public:
    Employee(Manager *boss, ...);
    // ...
    int numStaff() const;
};
```

2. Survey of Advanced *Levelization* Techniques

Opaque Pointers

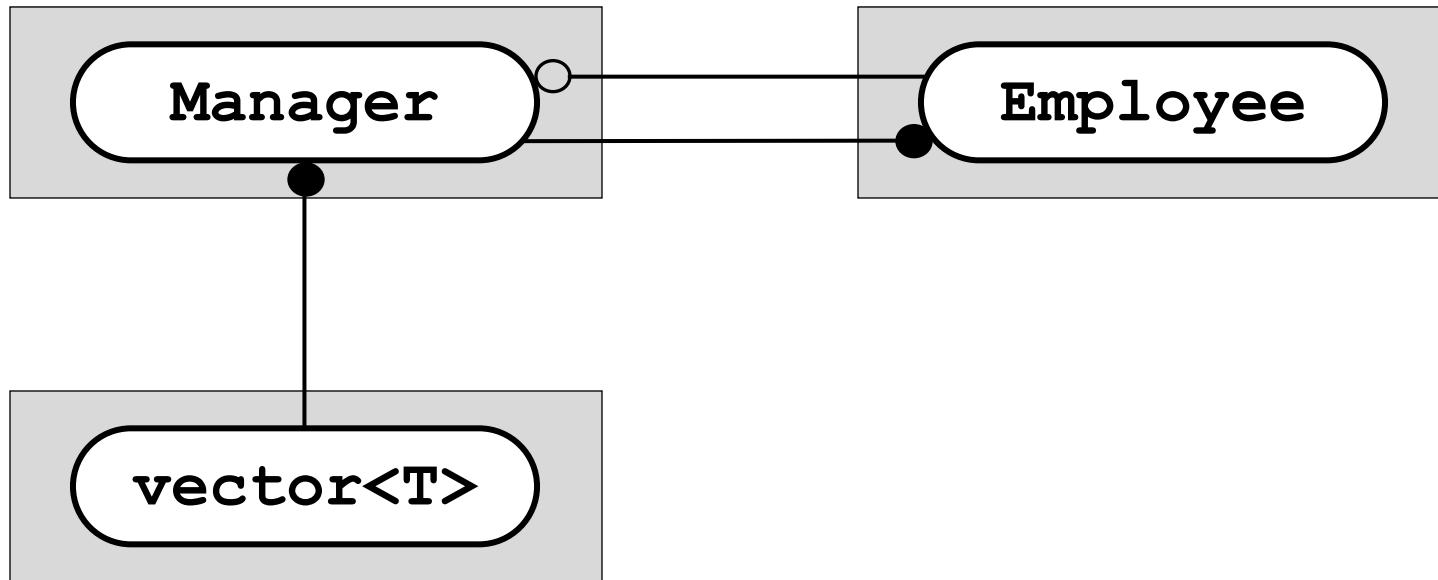
```
// manager.h
#include <employee.h>
#include <vector>
class Manager {
    vector<Employee> d_staff;
public:
    // ...
    int addEmployee(...);
    const Employee&
        employee(int id) const;
    // ...
};
```

```
// employee.h
#include <manager.h>
class Employee {
    Manager *d_boss_p;
public:
    Employee(Manager *boss, ...);
    // ...
    int numStaff() const;
};
```

```
#include <employee.h>
int askEmployeeNumStaff(const Employee& employee) {
    return employee.numStaff();
}
```

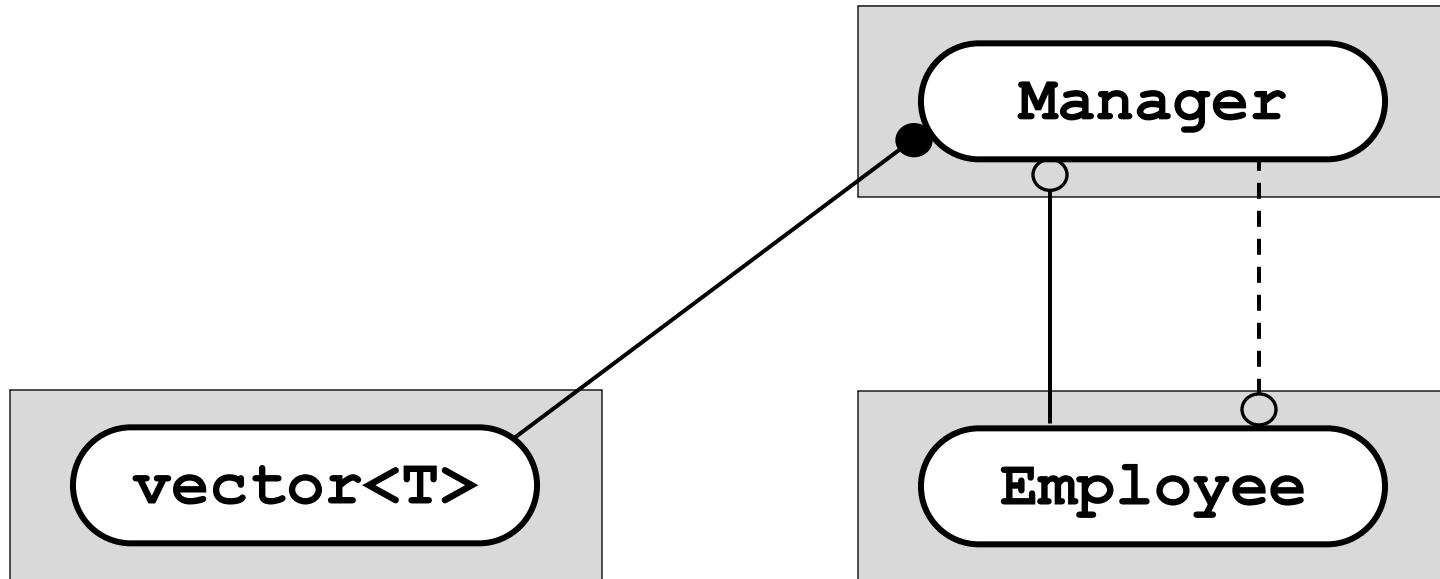
2. Survey of Advanced *Levelization* Techniques

Opaque Pointers



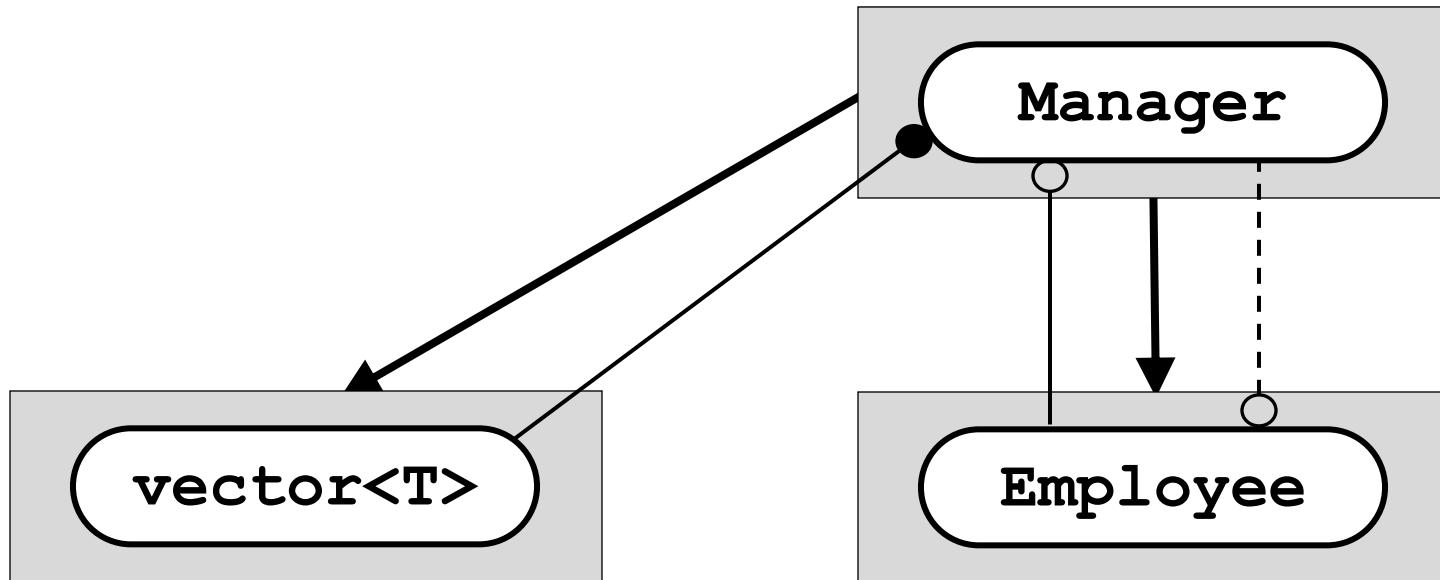
2. Survey of Advanced *Levelization* Techniques

Opaque Pointers



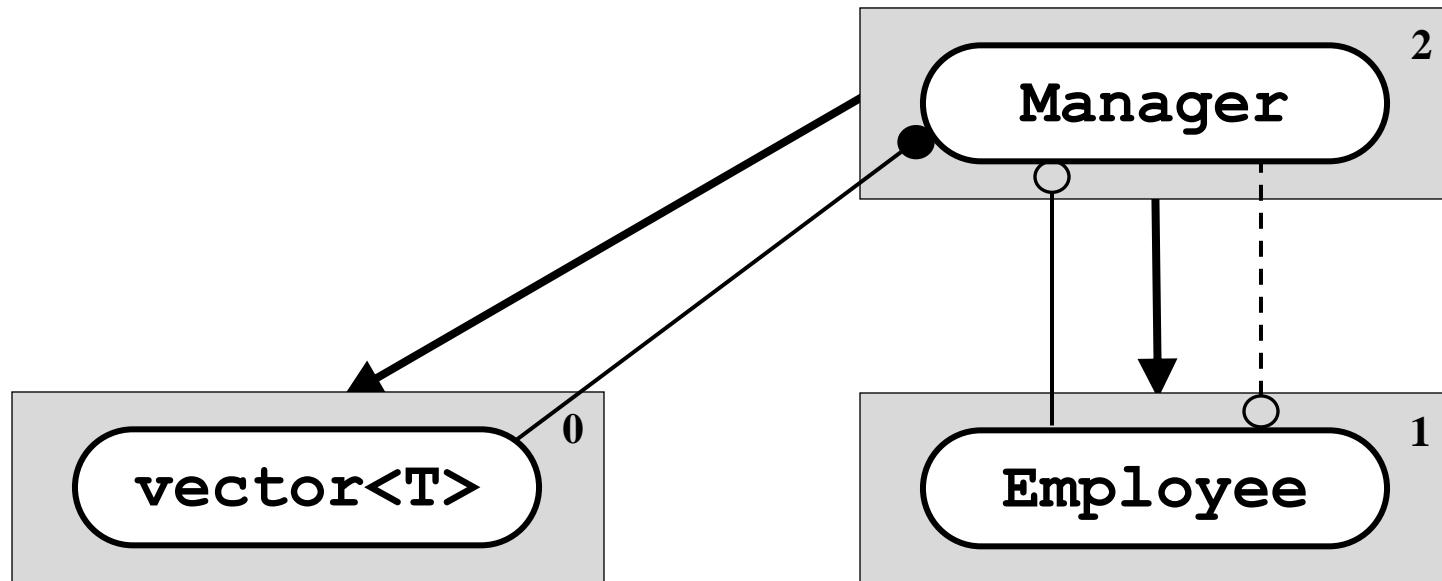
2. Survey of Advanced *Levelization* Techniques

Opaque Pointers



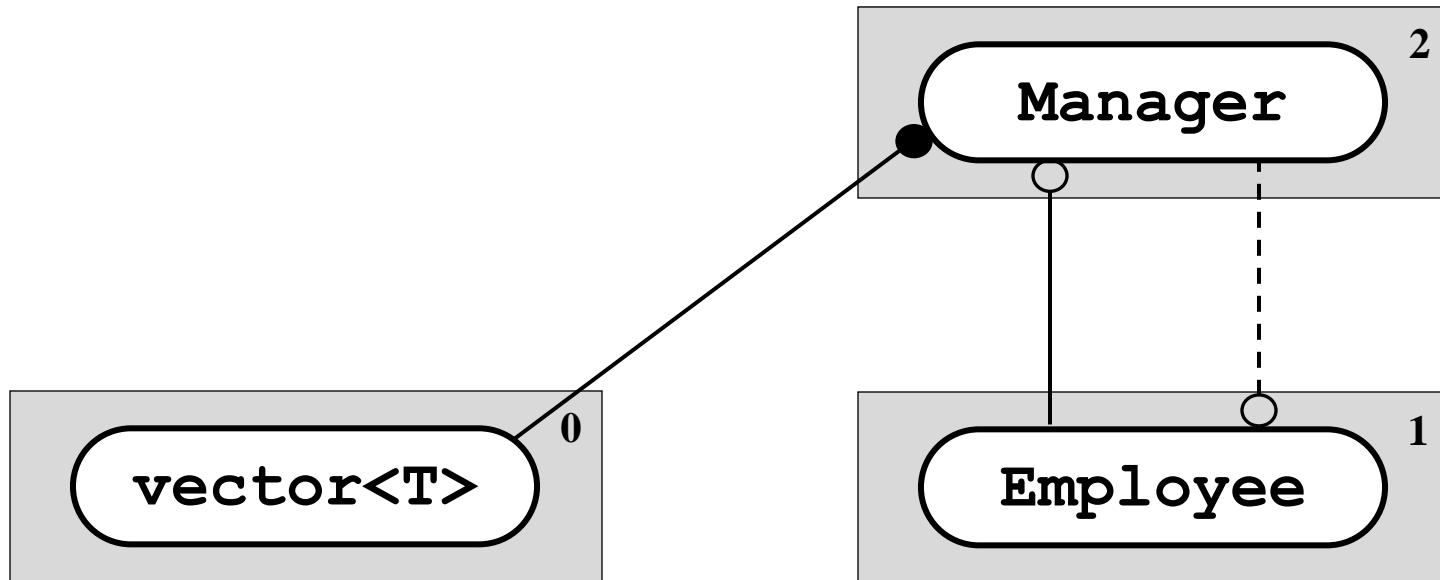
2. Survey of Advanced *Levelization* Techniques

Opaque Pointers



2. Survey of Advanced *Levelization* Techniques

Opaque Pointers



2. Survey of Advanced *Levelization* Techniques

Opaque Pointers

```
// manager.h
#include <employee.h>
#include <vector>
class Manager {
    vector<Employee> d_staff;
public:
    // ...
    int addEmployee(...);
    const Employee&
        employee(int id) const;
    // ...
};
```

```
// employee.h
#include <manager.h>
class Employee {
    Manager *d_boss_p;
public:
    Employee(Manager *boss, ...);
    // ...
    int numStaff() const;
};
```

2. Survey of Advanced *Levelization* Techniques

Opaque Pointers

```
// manager.h
#include <employee.h>
#include <vector>
class Manager {
    vector<Employee> d_staff;
public:
    // ...
    int addEmployee(...);
    const Employee&
        employee(int id) const;
    // ...
};
```

```
// employee.h
#include <manager.h>
class Employee {
    Manager *d_boss_p;
public:
    Employee(Manager *boss, ...);
    // ...
    int numStaff() const;
};
```

2. Survey of Advanced *Levelization* Techniques

Opaque Pointers

```
// manager.h
#include <employee.h>
#include <vector>
class Manager {
    vector<Employee> d_staff;
public:
    // ...
    int addEmployee(...);
    const Employee&
        employee(int id) const;
    // ...
};
```

```
// employee.h
class Manager;
class Employee {
    Manager *d_boss_p;
public:
    Employee(Manager *boss, ...);
    // ...
    int numStaff() const;
};
```

2. Survey of Advanced *Levelization* Techniques

Opaque Pointers

```
// manager.h
#include <employee.h>
#include <vector>
class Manager {
    vector<Employee> d_staff;
public:
    // ...
    int addEmployee(...);
    const Employee&
        employee(int id) const;
    // ...
};
```

```
// employee.h
class Manager;
class Employee {
    Manager *d_boss_p;
public:
    Employee(Manager *boss, ...);
    // ...
    int numStaff() const;
};
```

2. Survey of Advanced *Levelization* Techniques

Opaque Pointers

```
// manager.h
#include <employee.h>
#include <vector>
class Manager {
    vector<Employee> d_staff;
public:
    // ...
    int addEmployee(...);
    const Employee&
        employee(int id) const;
    // ...
};
```

```
// employee.h
class Manager;
class Employee {
    Manager *d_boss_p;
public:
    Employee(Manager *boss, ...);
    // ...
    int numStaff() const;
};
```

2. Survey of Advanced *Levelization* Techniques

Opaque Pointers

```
// manager.h
#include <employee.h>
#include <vector>
class Manager {
    vector<Employee> d_staff;
public:
    // ...
    int addEmployee(...);
    const Employee&
        employee(int id) const;
    // ...
};
```

```
// employee.h
class Manager;
class Employee {
    Manager *d_boss_p;
public:
    Employee(Manager *boss, ...);
    // ...
    const Manager
        *manager() const;
};
```

2. Survey of Advanced *Levelization* Techniques

Opaque Pointers

```
// manager.h
#include <employee.h>
#include <vector>
class Manager {
    vector<Employee> d_staff;
public:
    // ...
    int addEmployee(...);
    const Employee&
        employee(int id) const;
    // ...
};
```

```
// employee.h
class Manager;
class Employee {
    Manager *d_boss_p;
public:
    Employee(Manager *boss, ...);
    // ...
    const Manager
        *manager() const;
};
```

2. Survey of Advanced *Levelization* Techniques

Opaque Pointers

```
// manager.h
#include <employee.h>
#include <vector>
class Manager {
    vector<Employee> d_staff;
public:
    // ...
    int addEmployee(...);
    const Employee&
        employee(int id) const;
    // ...
};
```

```
// employee.h
class Manager;
class Employee {
    Manager *d_boss_p;
public:
    Employee(Manager *boss, ...);
    // ...
    const Manager
        *manager() const;
};
```

2. Survey of Advanced *Levelization* Techniques

Opaque Pointers

```
// manager.h
#include <employee.h>
#include <vector>
class Manager {
    vector<Employee> d_staff;
public:
    // ...
    int addEmployee(...);
    const Employee&
        employee(int id) const;
    int numStaff() const;
};
```

```
// employee.h
class Manager;
class Employee {
    Manager *d_boss_p;
public:
    Employee(Manager *boss, ...);
    // ...
    const Manager
        *manager() const;
};
```

2. Survey of Advanced *Levelization* Techniques

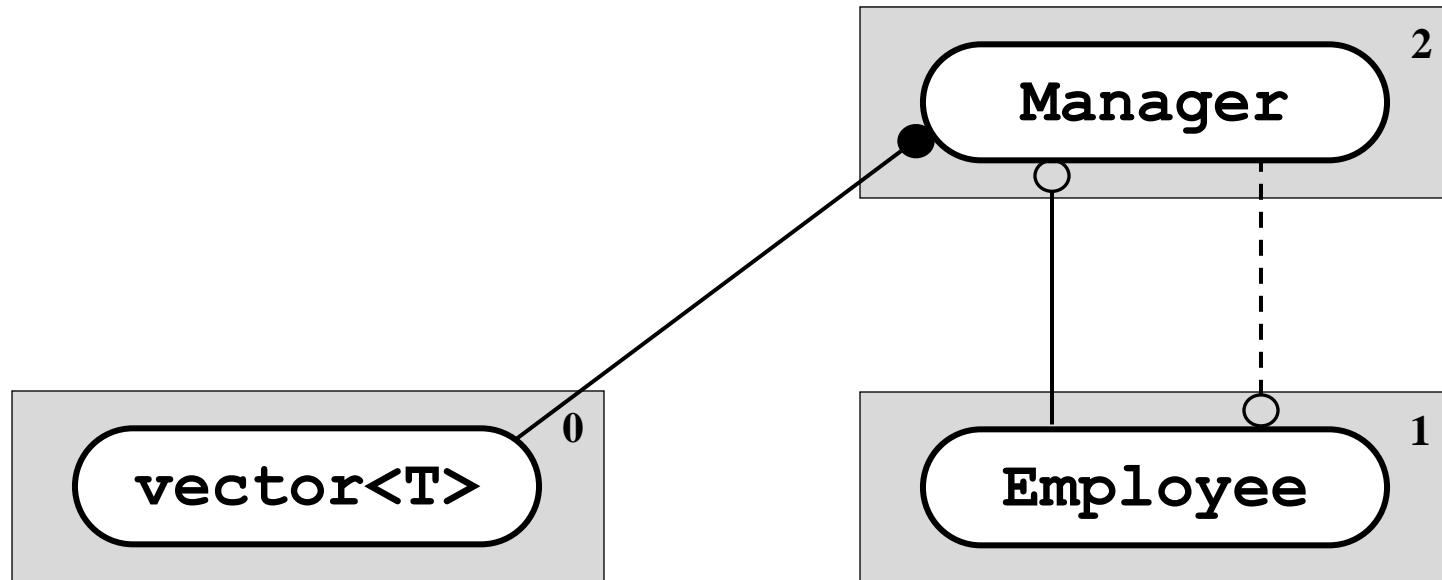
Opaque Pointers

```
// manager.h
#include <employee.h>
#include <vector>
class Manager {
    vector<Employee> d_staff;
public:
    // ...
    int addEmployee(...);
    const Employee&
        employee(int id) const;
    int numStaff() const;
};
```

```
// employee.h
class Manager;
class Employee {
    Manager *d_boss_p;
public:
    Employee(Manager *boss, ...);
    // ...
    const Manager
        *manager() const;
};
```

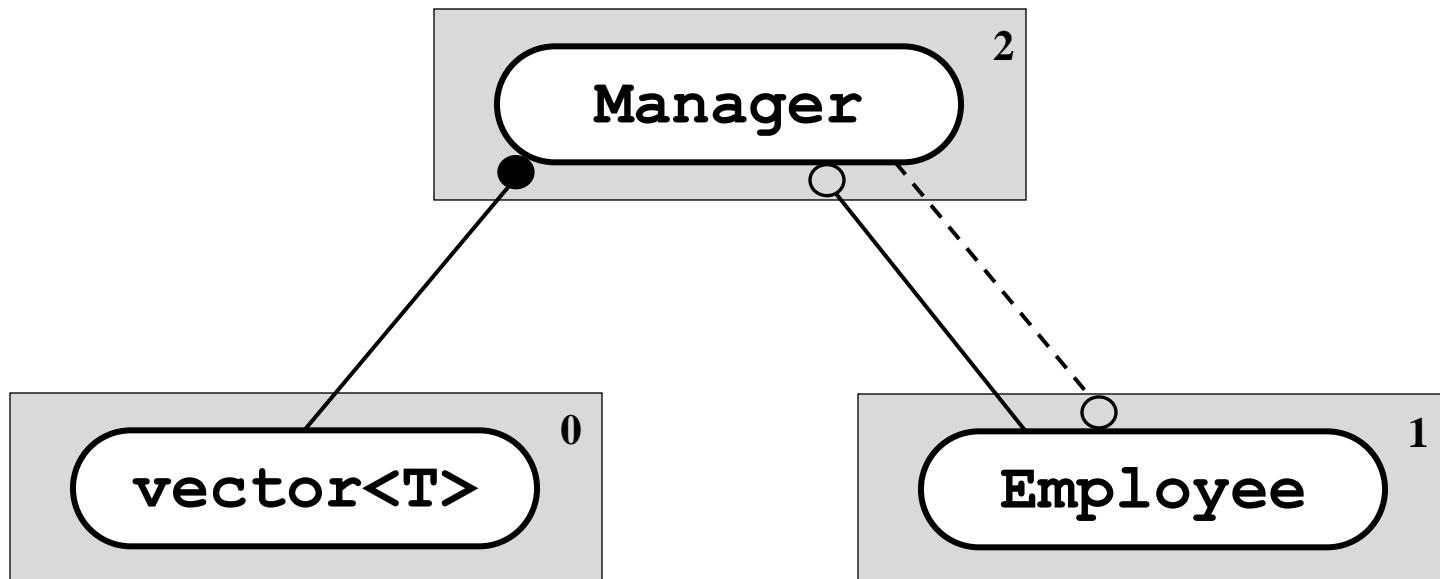
2. Survey of Advanced *Levelization* Techniques

Opaque Pointers



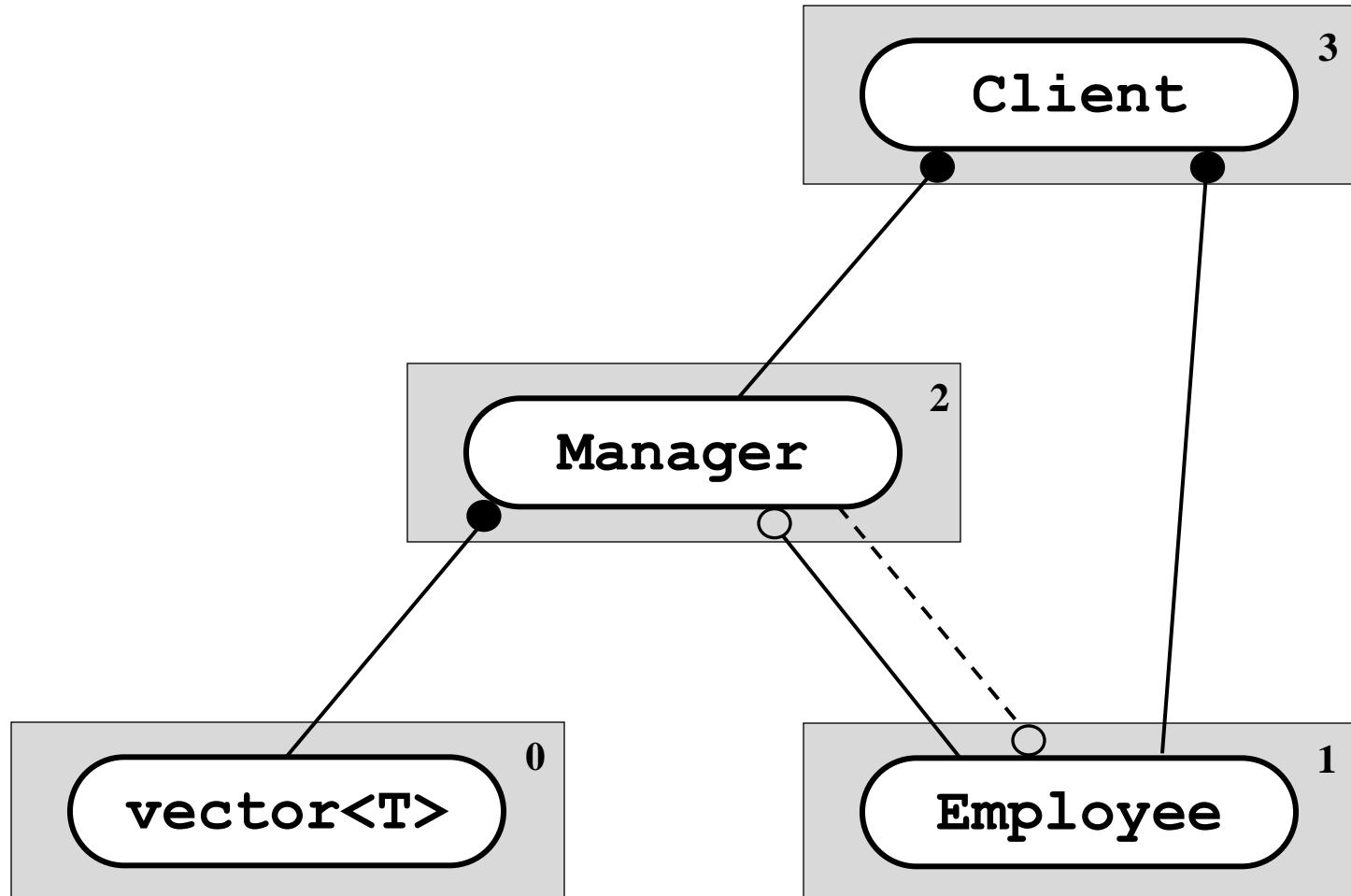
2. Survey of Advanced *Levelization* Techniques

Opaque Pointers



2. Survey of Advanced *Levelization* Techniques

Opaque Pointers



2. Survey of Advanced *Levelization* Techniques

Opaque Pointers

```
// manager.h
#include <employee.h>
#include <vector>
class Manager {
    vector<Employee> d_staff;
public:
    // ...
    int addEmployee(...);
    const Employee&
        employee(int id) const;
    int numStaff() const;
};
```

```
// employee.h
class Manager;
class Employee {
    Manager *d_boss_p;
public:
    Employee(Manager *boss, ...);
    // ...
    const Manager
        *manager() const;
};
```

2. Survey of Advanced *Levelization* Techniques

Opaque Pointers

```
// manager.h
#include <employee.h>
#include <vector>
class Manager {
    vector<Employee> d_staff;
public:
    // ...
    int addEmployee(...);
    const Employee&
        employee(int id) const;
    int numStaff() const;
};
```

```
// employee.h
class Manager;
class Employee {
    Manager *d_boss_p;
public:
    Employee(Manager *boss, ...);
    // ...
    const Manager
        *manager() const;
};
```

```
// client.cpp
#include <employee.h>
#include <manager.h>
int askEmployeeNumStaff(const Employee& employee) {
    return employee.manager()->numStaff();
}
```

2. Survey of Advanced *Levelization* Techniques

Opaque Pointers

```
// manager.h
#include <employee.h>
#include <vector>
class Manager {
    vector<Employee> d_staff;
public:
    // ...
    int addEmployee(...);
    const Employee&
        employee(int id) const;
    int numStaff() const;
};
```

```
// employee.h
class Manager;
class Employee {
    Manager *d_boss_p;
public:
    Employee(Manager *boss, ...);
    // ...
    const Manager
        *manager() const;
};
```

```
// client.cpp
#include <employee.h>
#include <manager.h>
int askEmployeeNumStaff(const Employee& employee) {
    return employee.manager()->numStaff();
}
```

2. Survey of Advanced *Levelization* Techniques

Opaque Pointers

```
// manager.h
#include <employee.h>
#include <vector>
class Manager {
    vector<Employee> d_staff;
public:
    // ...
    int addEmployee(...);
    const Employee&
        employee(int id) const;
    int numStaff() const;
};
```

```
// employee.h
class Manager;
class Employee {
    Manager *d_boss_p;
public:
    Employee(Manager *boss, ...);
    // ...
    const Manager
        *manager() const;
};
```

```
// client.cpp
#include <employee.h>
#include <manager.h>
int askEmployeeNumStaff(const Employee& employee){
    return employee.manager()->numStaff();
}
```

2. Survey of Advanced *Levelization* Techniques

Opaque Pointers

```
// manager.h
#include <employee.h>
#include <vector>
class Manager {
    vector<Employee> d_staff;
public:
    // ...
    int addEmployee(...);
    const Employee&
        employee(int id) const;
    int numStaff() const;
};
```

```
// employee.h
class Manager;
class Employee {
    Manager *d_boss_p;
public:
    Employee(Manager *boss, ...);
    // ...
    const Manager
        *manager() const;
};
```

```
// client.cpp
#include <employee.h>
#include <manager.h>
int askEmployeeNumStaff(const Employee& employee) {
    return employee.manager()->numStaff();
}
```

2. Survey of Advanced *Levelization* Techniques

Opaque Pointers

Discussion?

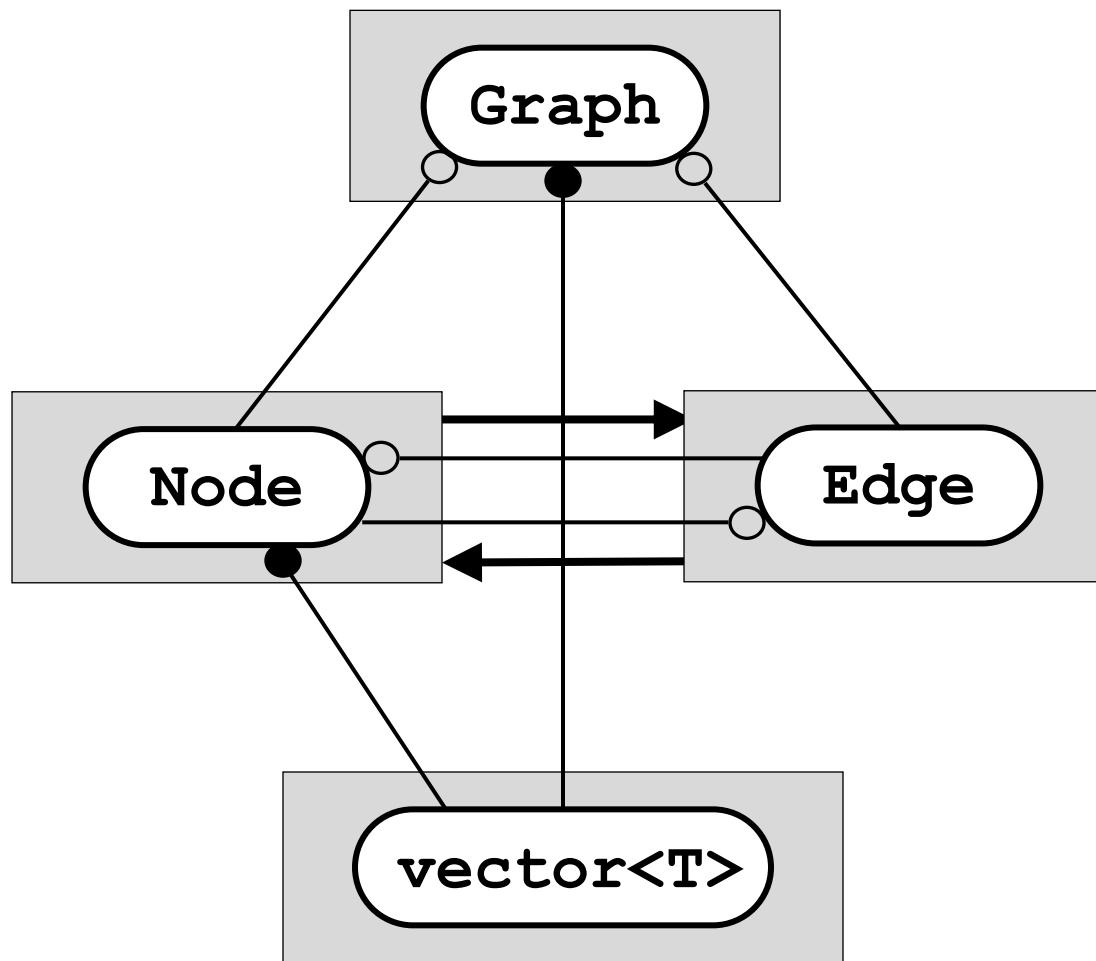
2. Survey of Advanced *Levelization* Techniques

Dumb Data

Dumb Data – Using data that indicates a dependency on a peer object, but only in the context of a separate, higher-level object.

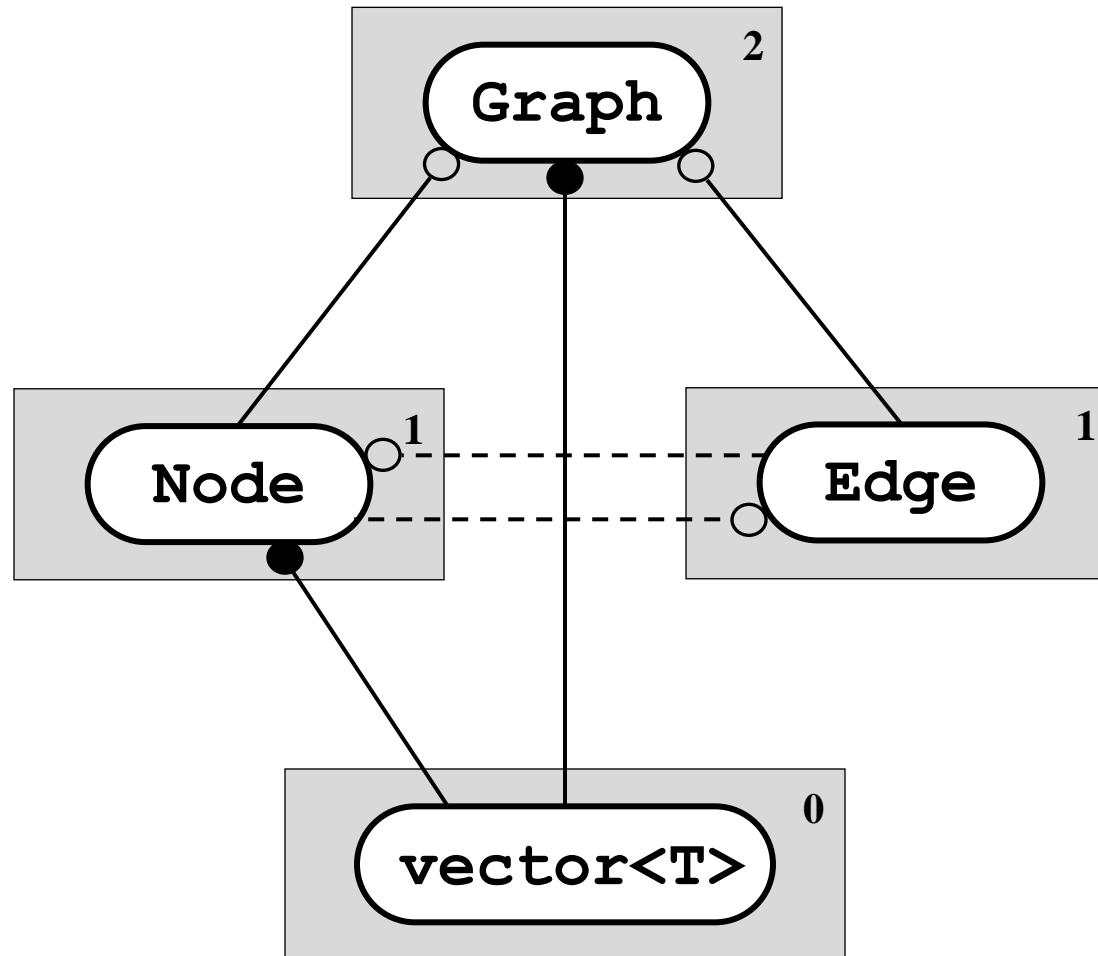
2. Survey of Advanced *Levelization* Techniques

Dumb Data



2. Survey of Advanced *Levelization* Techniques

Dumb Data



2. Survey of Advanced *Levelization* Techniques

Dumb Data

```
// node.h
#include <vector>
#include <edge.h>
class Node {
    vector<Edge *> d_edges;
    // ...
public:
    Node();
    // ...
    void addEdge(Edge *edge);
    // ...
    int numEdges() const;
    // ...
    const Edge *edge(int idx) const;
    // ...
    int value() const;
    // ...
};
```

```
// edge.h
#include <node.h>
class Edge {
    Node *d_head_p;
    Node *d_tail_p;
    // ...
public:
    Edge(Node *head, Node *tail);
    // ...
    void setWeight(double value);
    // ...
    const Node *head() const;
    const Node *tail() const;
    // ...
    double weight() const;
    // ...
};
```

2. Survey of Advanced *Levelization* Techniques

Dumb Data

```
// node.h
#include <vector>
#include <edge.h>;
class Node {
    vector<Edge *> d_edges;
    // ...
public:
    Node();
    // ...
    void addEdge(Edge *edge);
    // ...
    int numEdges() const;
    // ...
    const Edge *edge(int idx) const;
    // ...
    int value() const;
    // ...
};
```

```
// edge.h
#include <node.h>
class Edge {
    Node *d_head_p;
    Node *d_tail_p;
    // ...
public:
    Edge(Node *head, Node *tail);
    // ...
    void setWeight(double value);
    // ...
    const Node *head() const;
    const Node *tail() const;
    // ...
    double weight() const;
    // ...
};
```

2. Survey of Advanced *Levelization* Techniques

Dumb Data

```
// node.h
#include <vector>
class Edge;
class Node {
    vector<Edge *> d_edges;
    // ...
public:
    Node();
    // ...
    void addEdge(Edge *edge);
    // ...
    int numEdges() const;
    // ...
    const Edge *edge(int idx) const;
    // ...
    int value() const;
    // ...
};
```

```
// edge.h
class Node;
class Edge {
    Node *d_head_p;
    Node *d_tail_p;
    // ...
public:
    Edge(Node *head, Node *tail);
    // ...
    void setWeight(double value);
    // ...
    const Node *head() const;
    const Node *tail() const;
    // ...
    double weight() const;
    // ...
};
```

2. Survey of Advanced *Levelization* Techniques

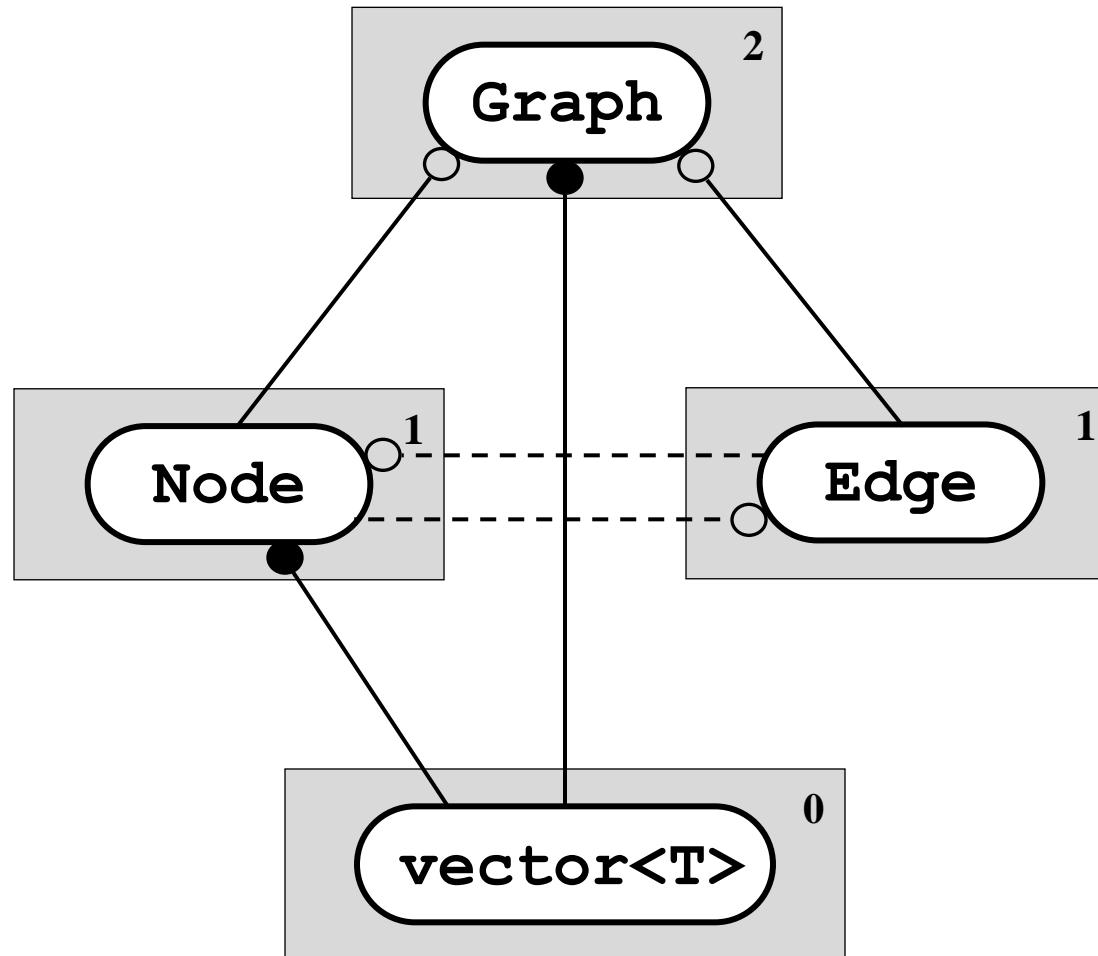
Dumb Data

```
// node.h
#include <vector>
class Edge;
class Node {
    vector<Edge *> d_edges;
    // ...
public:
    Node();
    // ...
    void addEdge(Edge *edge);
    // ...
    int numEdges() const;
    // ...
    const Edge *edge(int idx) const;
    // ...
    int value() const;
    // ...
};
```

```
// edge.h
class Node;
class Edge {
    Node *d_head_p;
    Node *d_tail_p;
    // ...
public:
    Edge(Node *head, Node *tail);
    // ...
    void setWeight(double value);
    // ...
    const Node *head() const;
    const Node *tail() const;
    // ...
    double weight() const;
    // ...
};
```

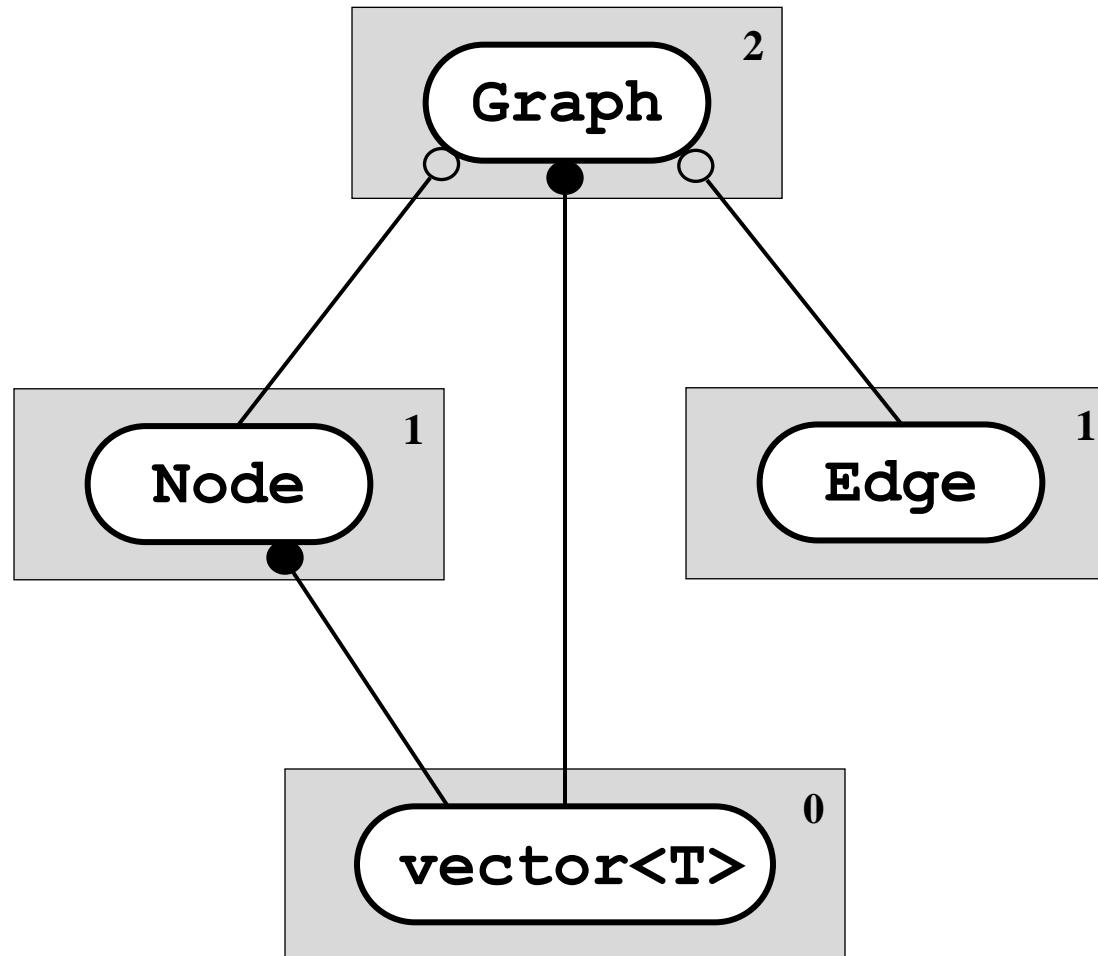
2. Survey of Advanced *Levelization* Techniques

Dumb Data



2. Survey of Advanced *Levelization* Techniques

Dumb Data



2. Survey of Advanced *Levelization* Techniques

Dumb Data

```
// node.h
#include <vector>
class Edge;
class Node {
    vector<Edge *> d_edges;
    // ...
public:
    Node();
    // ...
    void addEdge(Edge *edge);
    // ...
    int numEdges() const;
    // ...
    const Edge *edge(int idx) const;
    // ...
    int value() const;
    // ...
};
```

```
// edge.h
class Node;
class Edge {
    Node *d_head_p;
    Node *d_tail_p;
    // ...
public:
    Edge(Node *head, Node *tail);
    // ...
    void setWeight(double value);
    // ...
    const Node *head() const;
    const Node *tail() const;
    // ...
    double weight() const;
    // ...
};
```

2. Survey of Advanced *Levelization* Techniques

Dumb Data

```
// node.h
#include <vector>
class Edge;
class Node {
    vector<Edge *> d_edges;
    // ...
public:
    Node();
    // ...
    void addEdge(Edge *edge);
    // ...
    int numEdges() const;
    // ...
    const Edge *edge(int idx) const;
    // ...
    int value() const;
    // ...
};
```

```
// edge.h
class Node;
class Edge {
    Node *d_head_p;
    Node *d_tail_p;
    // ...
public:
    Edge(Node *head, Node *tail);
    // ...
    void setWeight(double value);
    // ...
    const Node *head() const;
    const Node *tail() const;
    // ...
    double weight() const;
    // ...
};
```

2. Survey of Advanced *Levelization* Techniques

Dumb Data

```
// node.h
#include <vector>

class Node {
    vector<int> d_edgeIds;
    // ...
public:
    Node();
    // ...
    void addEdge(int edgeId);
    // ...
    int numEdges() const;
    // ...
    int edgeId(int idx) const;
    // ...
    int value() const;
    // ...
};
```

```
// edge.h
class Edge {
    int d_headId;
    int d_tailId;
    // ...
public:
    Edge(int headId, int tailId);
    // ...
    void setWeight(double value);
    // ...
    int headId() const;
    int tailId() const;
    // ...
    double weight() const;
    // ...
};
```

2. Survey of Advanced *Levelization* Techniques

Dumb Data

```
// node.h
#include <vector>

class Node {
    vector<int> d_edgeIds;
    // ...
public:
    Node();
    // ...
    void addEdge(int edgeId);
    // ...
    int numEdges() const;
    // ...
    int edgeId(int idx) const;
    // ...
    int value() const;
    // ...
};
```

```
// edge.h

class Edge {
    int d_headId;
    int d_tailId;
    // ...
public:
    Edge(int headId, int tailId);
    // ...
    void setWeight(double value);
    // ...
    int headId() const;
    int tailId() const;
    // ...
    double weight() const;
    // ...
};
```

2. Survey of Advanced *Levelization* Techniques

Dumb Data

```
// graph.h
#include <node.h>
#include <edge.h>
#include <vector>
class Graph {
    vector<Node> d_nodes;
    vector<Edge> d_edges;
public:
    Graph();
    // ...
    const Node *addNode();
    const Edge *addEdge(const Node *tail, const Node *head);
    // ...
    int numNodes() const;
    int numEdges() const;
    const Node *node(int nodeId);
    const Edge *edge(int edgeId);
};
```

2. Survey of Advanced *Levelization* Techniques

Dumb Data

```
// graph.h
#include <node.h>
#include <edge.h>
#include <vector>
class Graph {
    vector<Node> d_nodes;
    vector<Edge> d_edges;
public:
    Graph();
    // ...
    const Node *addNode();
    const Edge *addEdge(const Node *tail, const Node *head);
    // ...
    int numNodes() const;
    int numEdges() const;
    const Node *node(int nodeId);
    const Edge *edge(int edgeId);
};
```

2. Survey of Advanced *Levelization* Techniques

Dumb Data

```
// graph.h
#include <node.h>
#include <edge.h>
#include <vector>
class Graph {
    vector<Node> d_nodes;
    vector<Edge> d_edges;
public:
    Graph();
    // ...
    const Node *addNode();
    const Edge *addEdge(const Node *tail, const Node *head);
    // ...
    int numNodes() const;
    int numEdges() const;
    const Node *node(int nodeId);
    const Edge *edge(int edgeId);
};
```

2. Survey of Advanced *Levelization* Techniques

Dumb Data

```
// client.h
#include <graph.h>
#include <node.h>
#include <edge.h>

double firstEdgeWeight(const Node *node)
{
    return node->edge(0)->weight();
}

int headValue(const Edge *edge)
{
    return edge->head()->value();
}

int firstAdjacentValue(const Node *node)
{
    return node->edge(0)->head()->value();
}
```

Opaque Pointers

2. Survey of Advanced *Levelization* Techniques

Dumb Data

```
// client.h
#include <graph.h>
#include <node.h>
#include <edge.h>
```

```
double firstEdgeWeight(const Node *node)
{
    return node->edge(0)->weight();
}
```

```
int headValue(const Edge *edge)
{
    return edge->head()->value();
}
```

```
int firstAdjacentValue(const Node *node)
{
    return node->edge(0)->head()->value();
}
```

Opaque Pointers

2. Survey of Advanced *Levelization* Techniques

Dumb Data

Opaque Pointers

```
double firstEdgeWeight(const Node *node)
{
    return node->edge(0)->weight();
}
```

2. Survey of Advanced *Levelization* Techniques

Dumb Data

Opaque Pointers

```
double firstEdgeWeight(const Node *node)
{
    return node->edge(0)->weight();
}
```

Dumb Data

```
double firstEdgeWeight(const Graph& g, const Node *node)
{
    return g.edge(node->edgeId(0))->weight();
}
```

2. Survey of Advanced *Levelization* Techniques

Dumb Data

```
// client.h
#include <graph.h>
#include <node.h>
#include <edge.h>

double firstEdgeWeight(const Graph& g, const Node *node)
{
    return g.edge(node->edgeId(0))->weight();
}

int headValue(const Graph& g, const Edge *edge)
{
    return g.node(edge->headId())->value();
}

int firstAdjacentValue(const Graph& g, const Node *node)
{
    return g.node(g.edge(node->edgeId(0))->headId())->value();
}
```

Dumb Data

2. Survey of Advanced *Levelization* Techniques

Dumb Data

```
// client.h
#include <graph.h>
#include <node.h>
#include <edge.h>

double firstEdgeWeight(const Graph& g, const Node *node)
{
    return g.edge(node->edgeId(0))->weight();
}

int headValue(const Graph& g, const Edge *edge)
{
    return g.node(edge->headId())->value();
}

int firstAdjacentValue(const Graph& g, const Node *node)
{
    return g.edge(node->edgeId(0))->headId()->value();
```

Dumb Data
Node and Edge
are Independently Externalizable!

2. Survey of Advanced *Levelization* Techniques

Dumb Data

Discussion?

2. Survey of Advanced *Levelization* Techniques

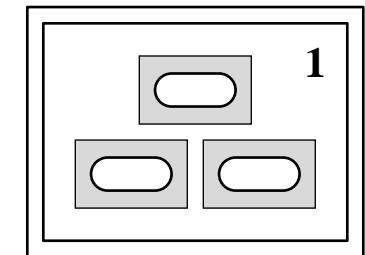
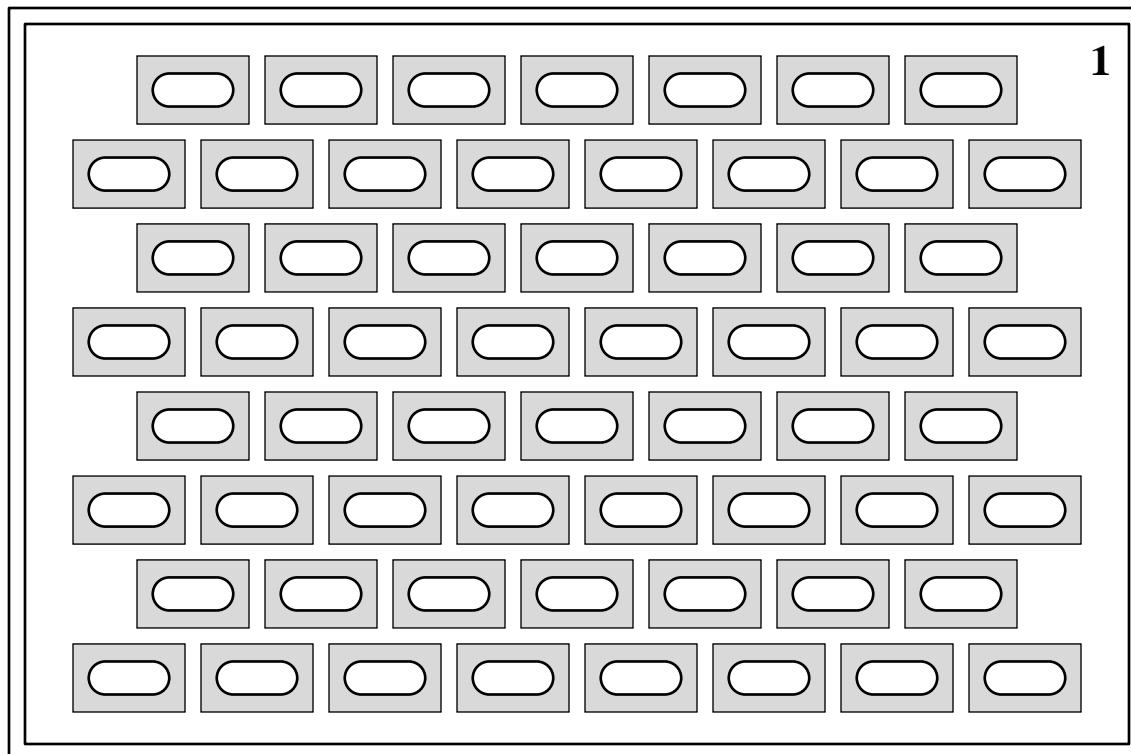
Redundancy

Redundancy –

Deliberately avoiding reuse by repeating a small amount of code or data to avoid coupling.

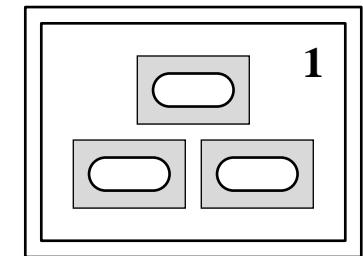
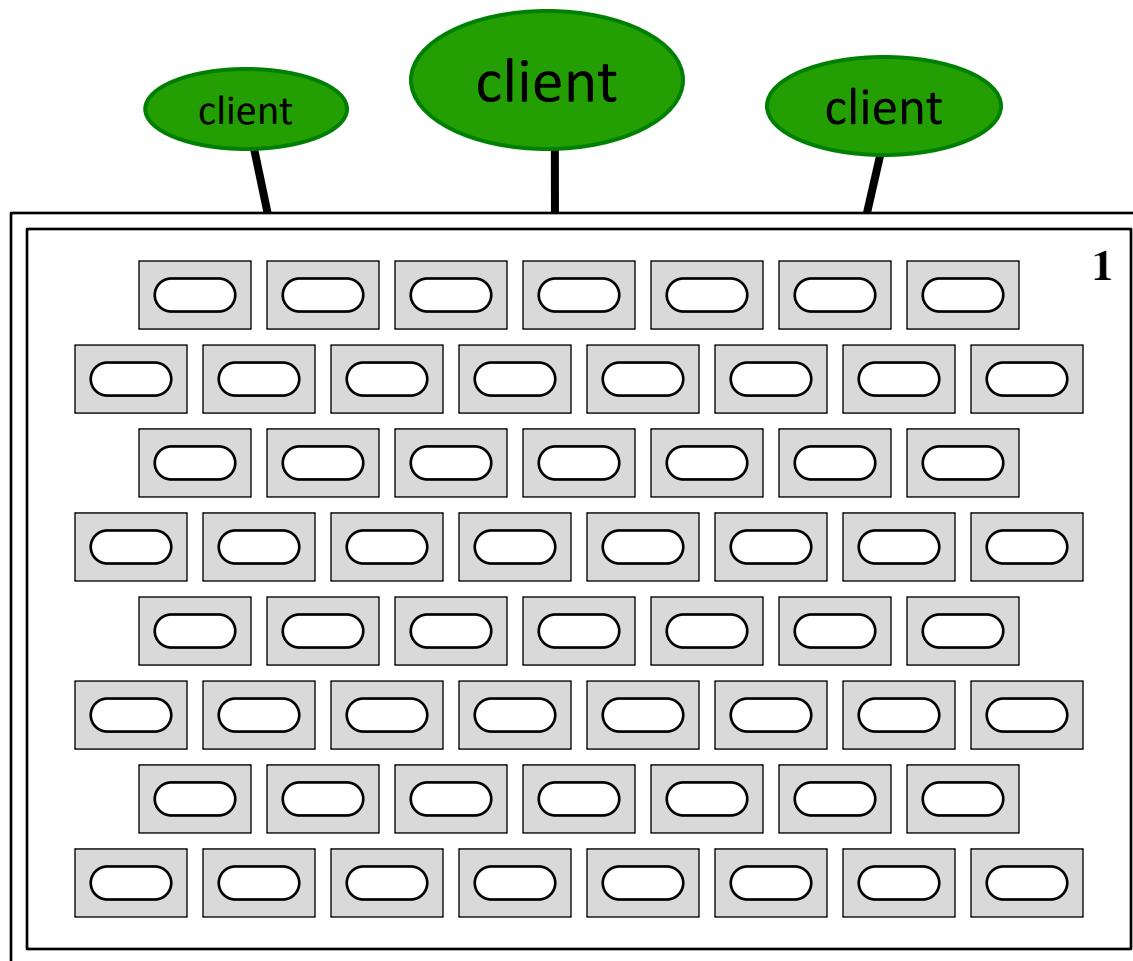
2. Survey of Advanced *Levelization* Techniques

Redundancy



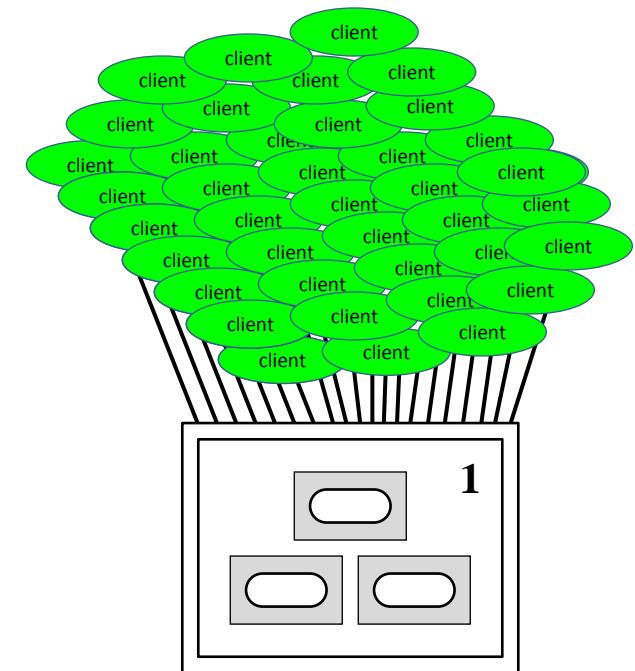
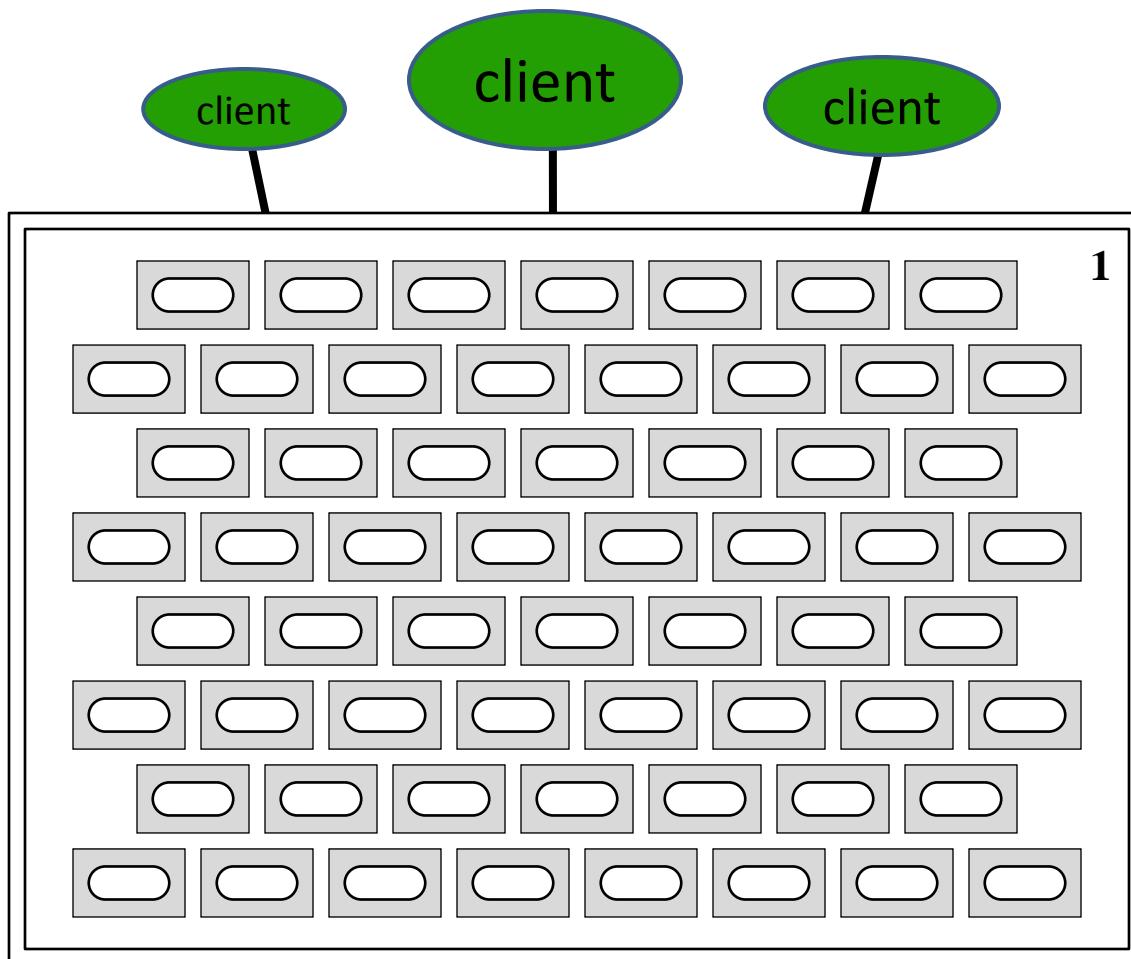
2. Survey of Advanced *Levelization* Techniques

Redundancy



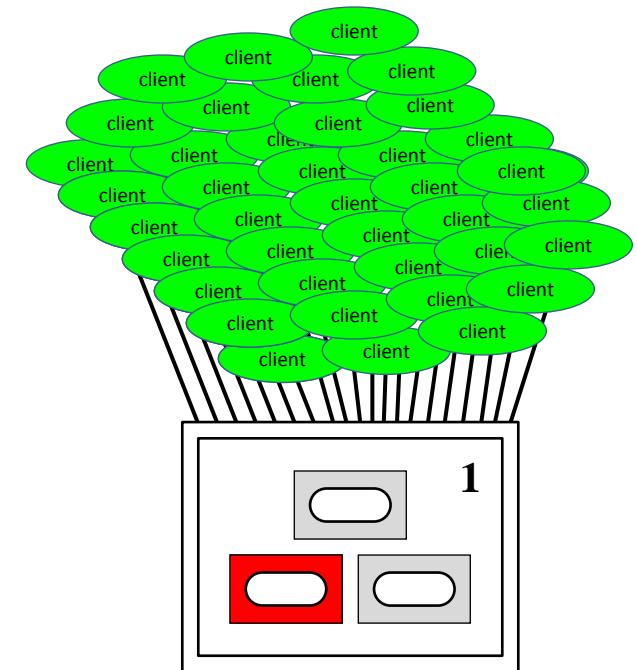
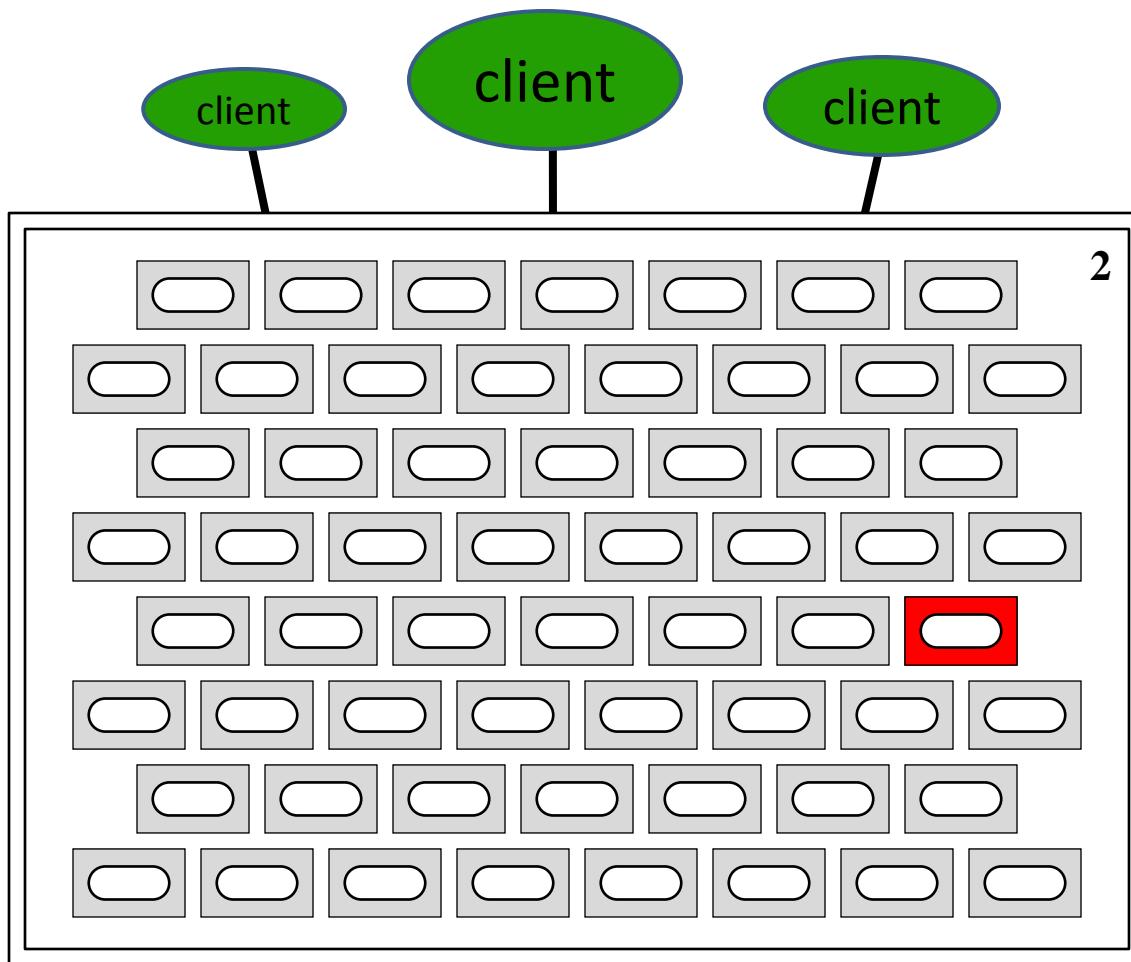
2. Survey of Advanced *Levelization* Techniques

Redundancy



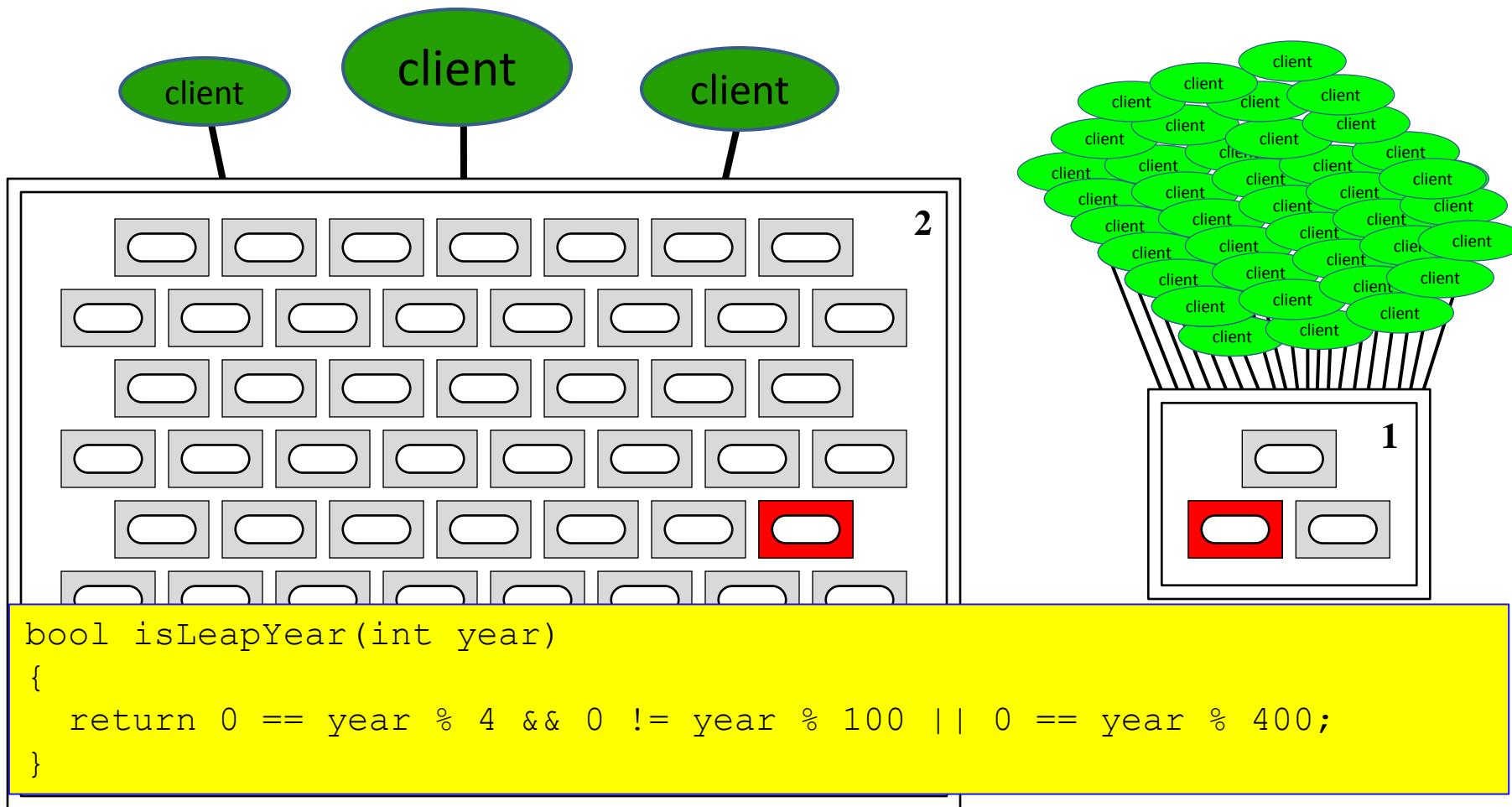
2. Survey of Advanced *Levelization* Techniques

Redundancy



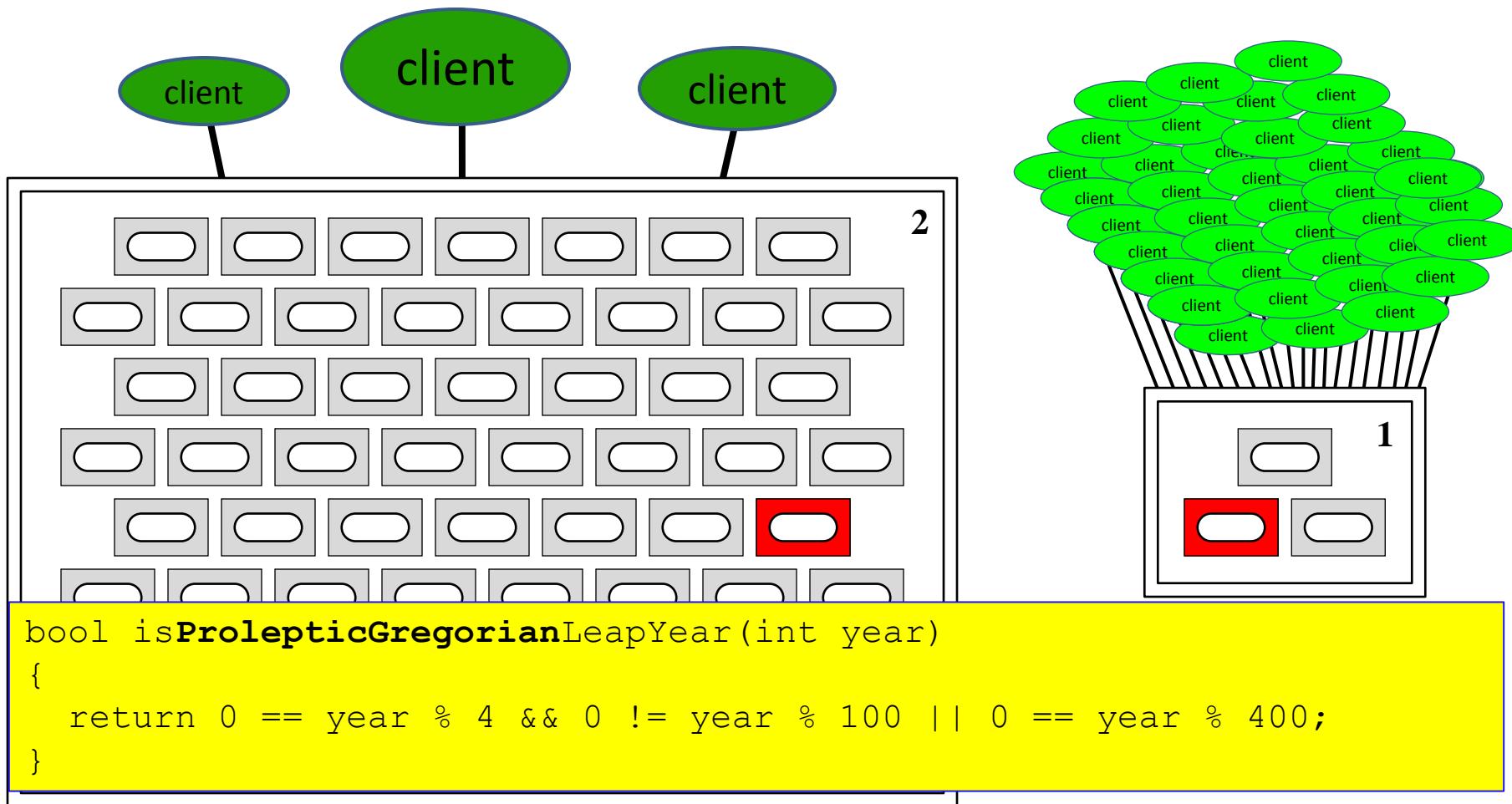
2. Survey of Advanced *Levelization* Techniques

Redundancy



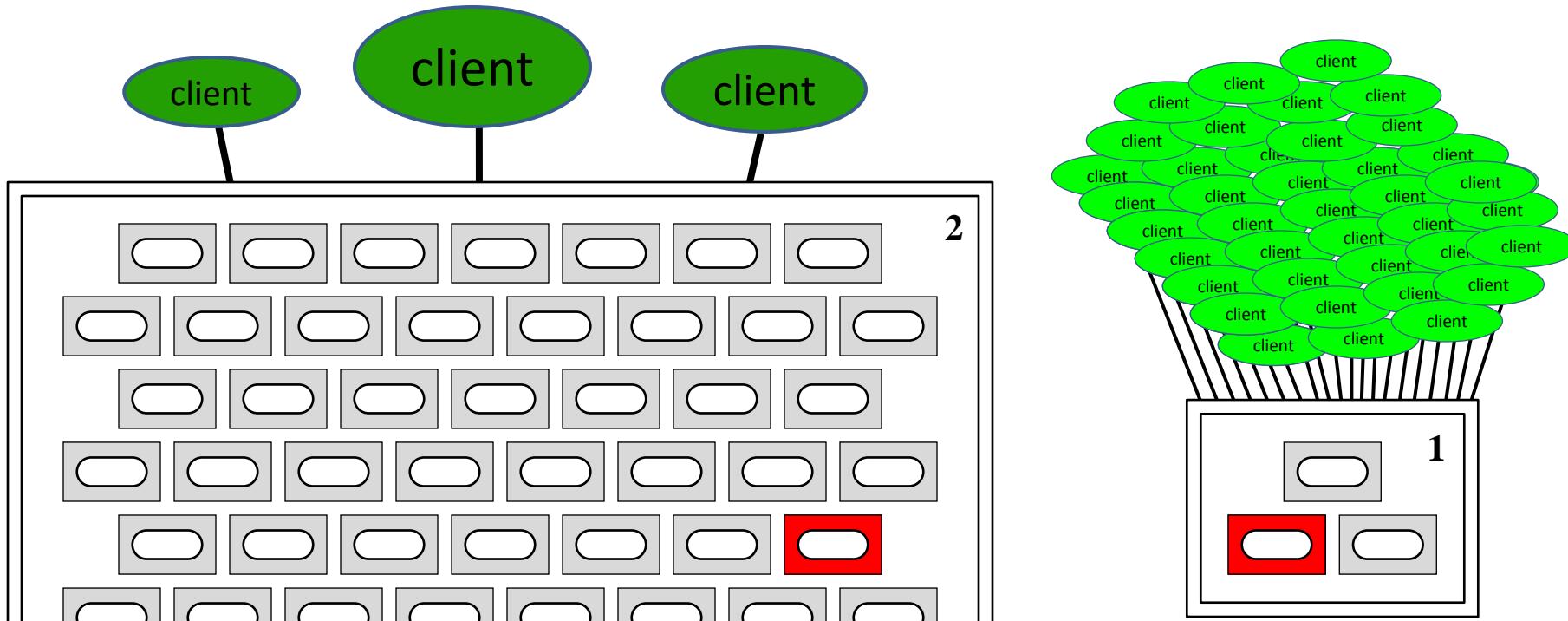
2. Survey of Advanced *Levelization* Techniques

Redundancy



2. Survey of Advanced *Levelization* Techniques

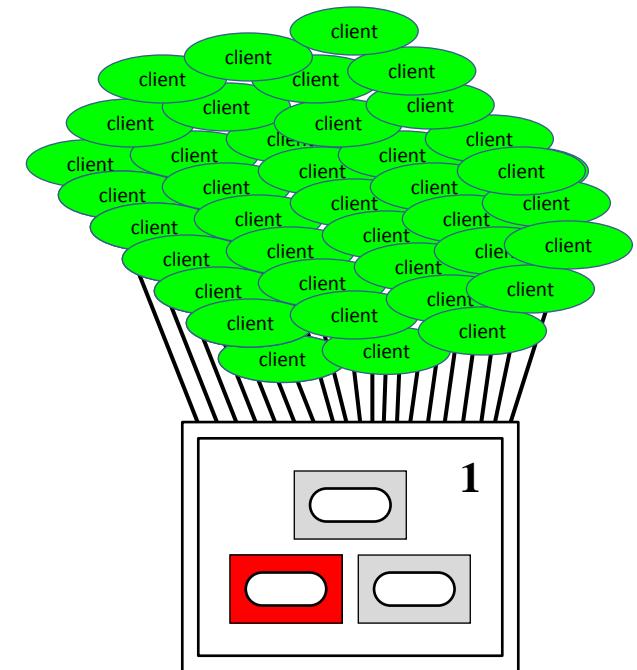
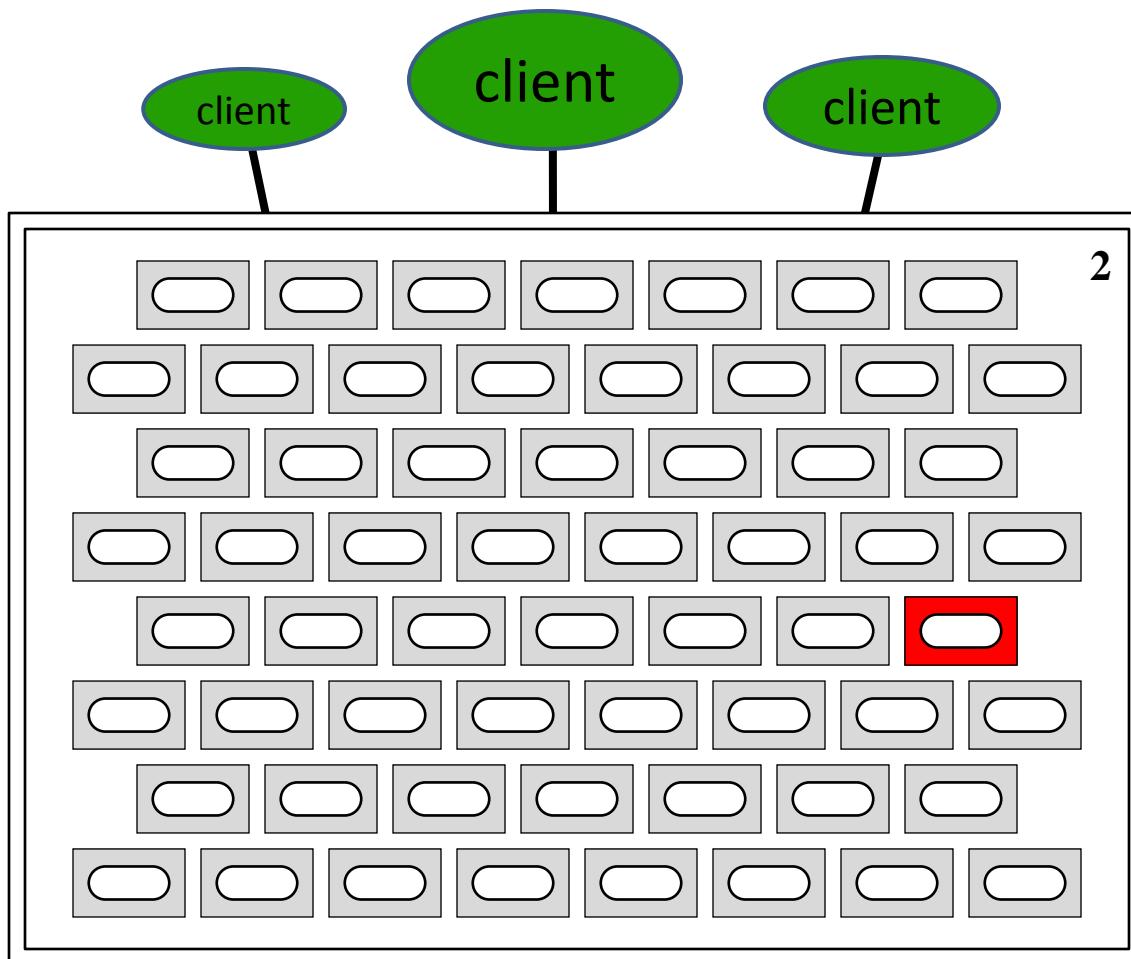
Redundancy



```
inline int min(int a, int b)
{
    return a < b ? a : b;
}
```

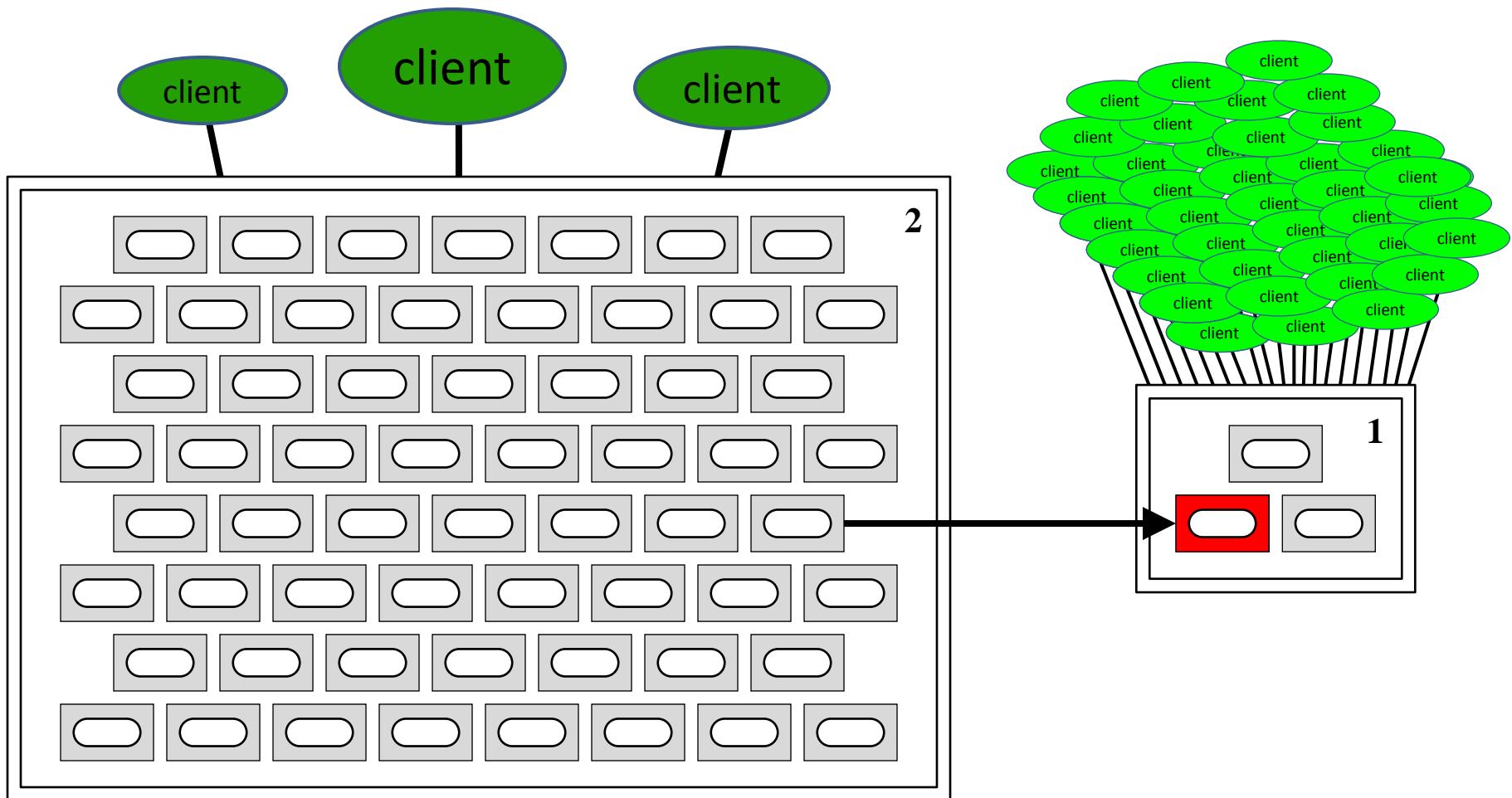
2. Survey of Advanced *Levelization* Techniques

Redundancy



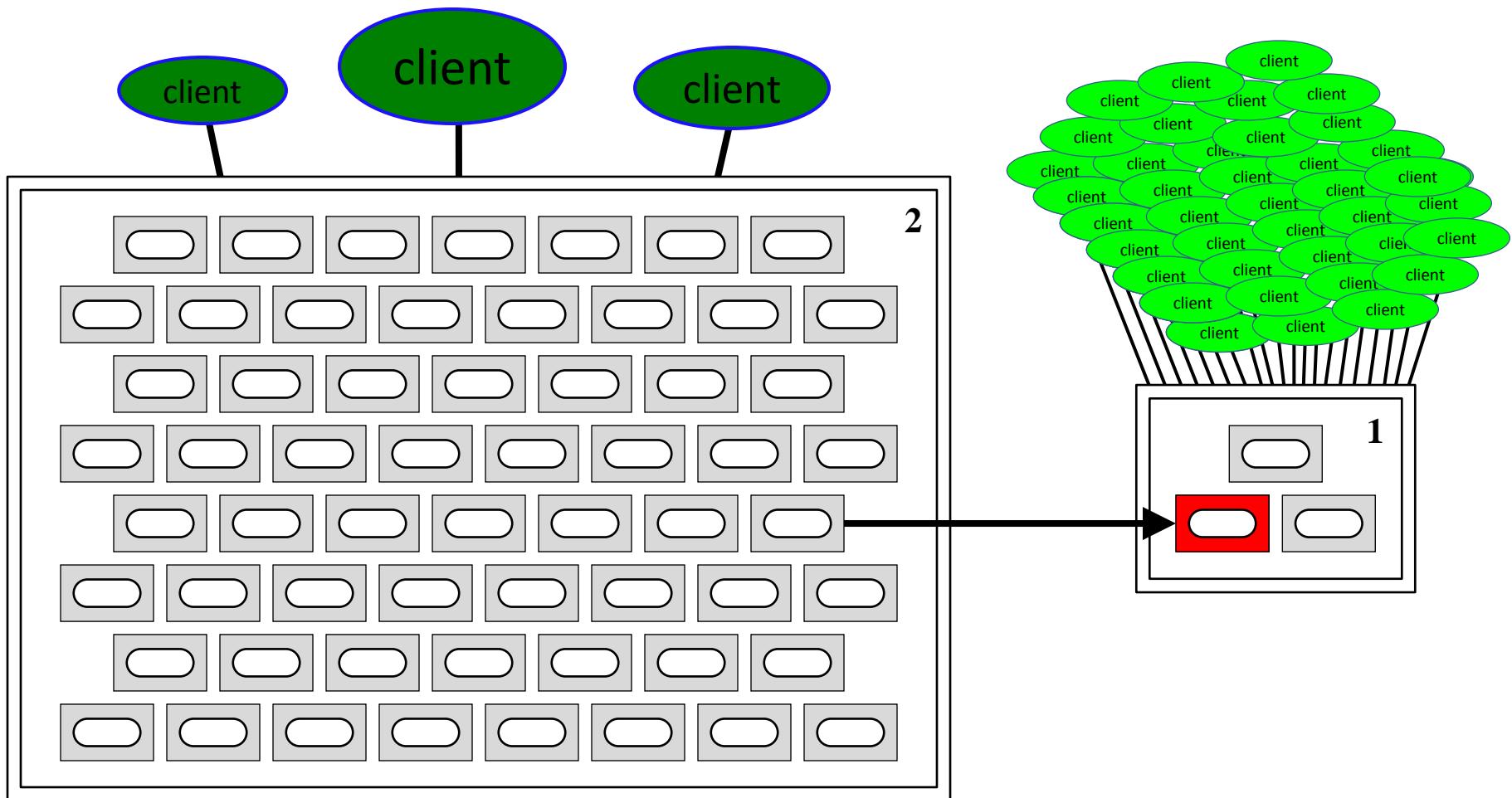
2. Survey of Advanced *Levelization* Techniques

Redundancy



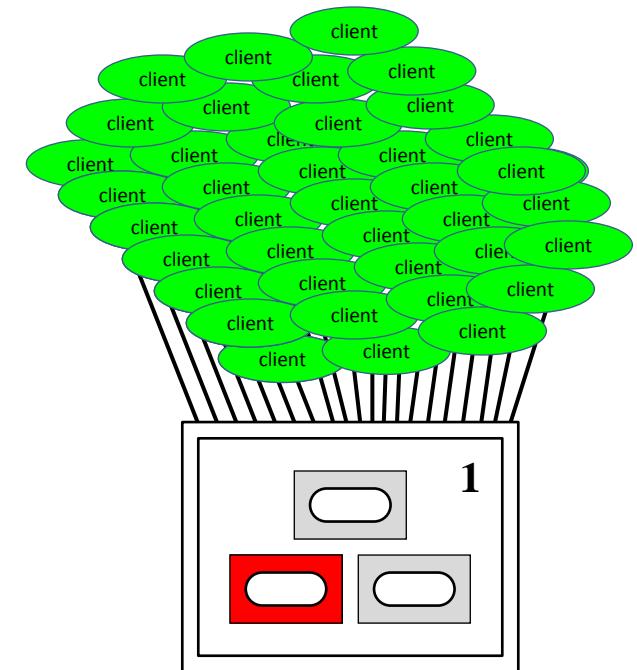
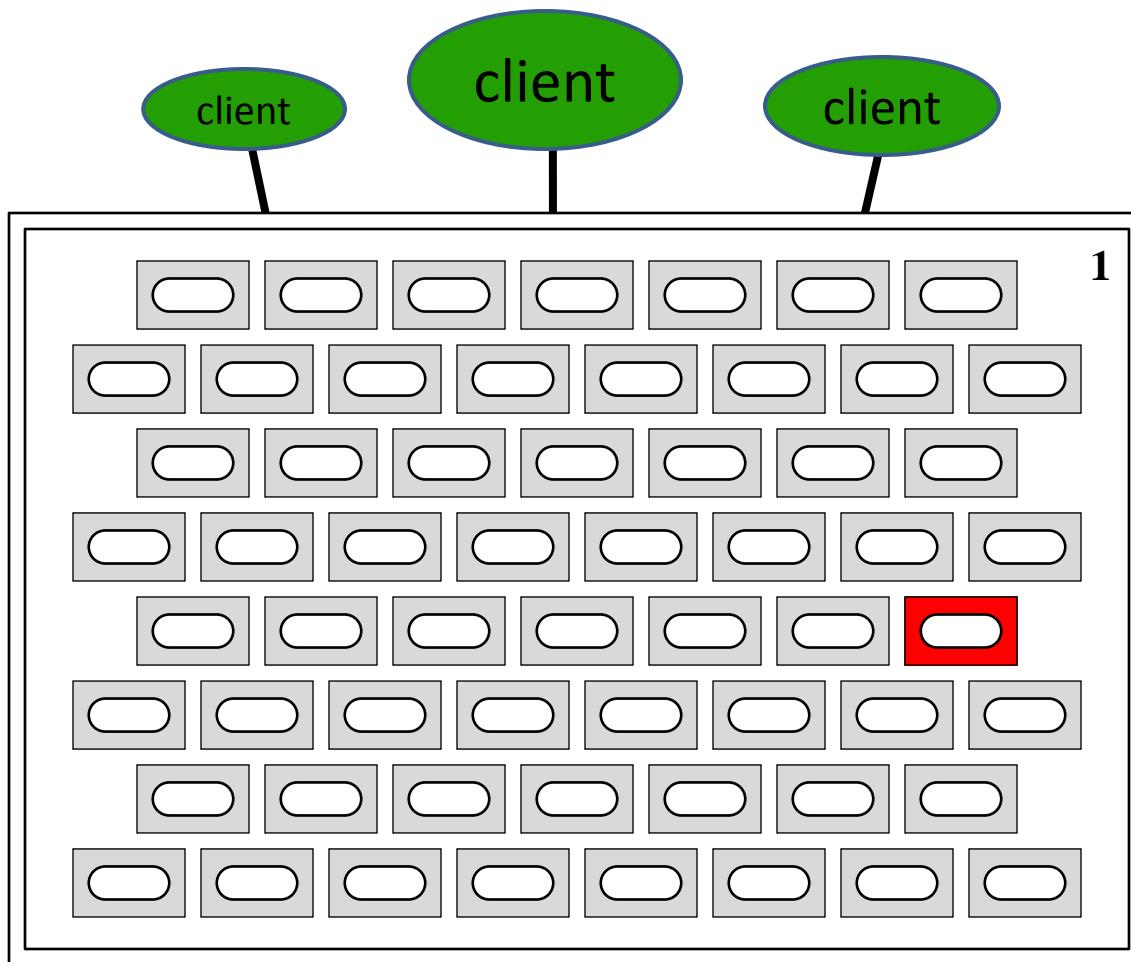
2. Survey of Advanced *Levelization* Techniques

Redundancy



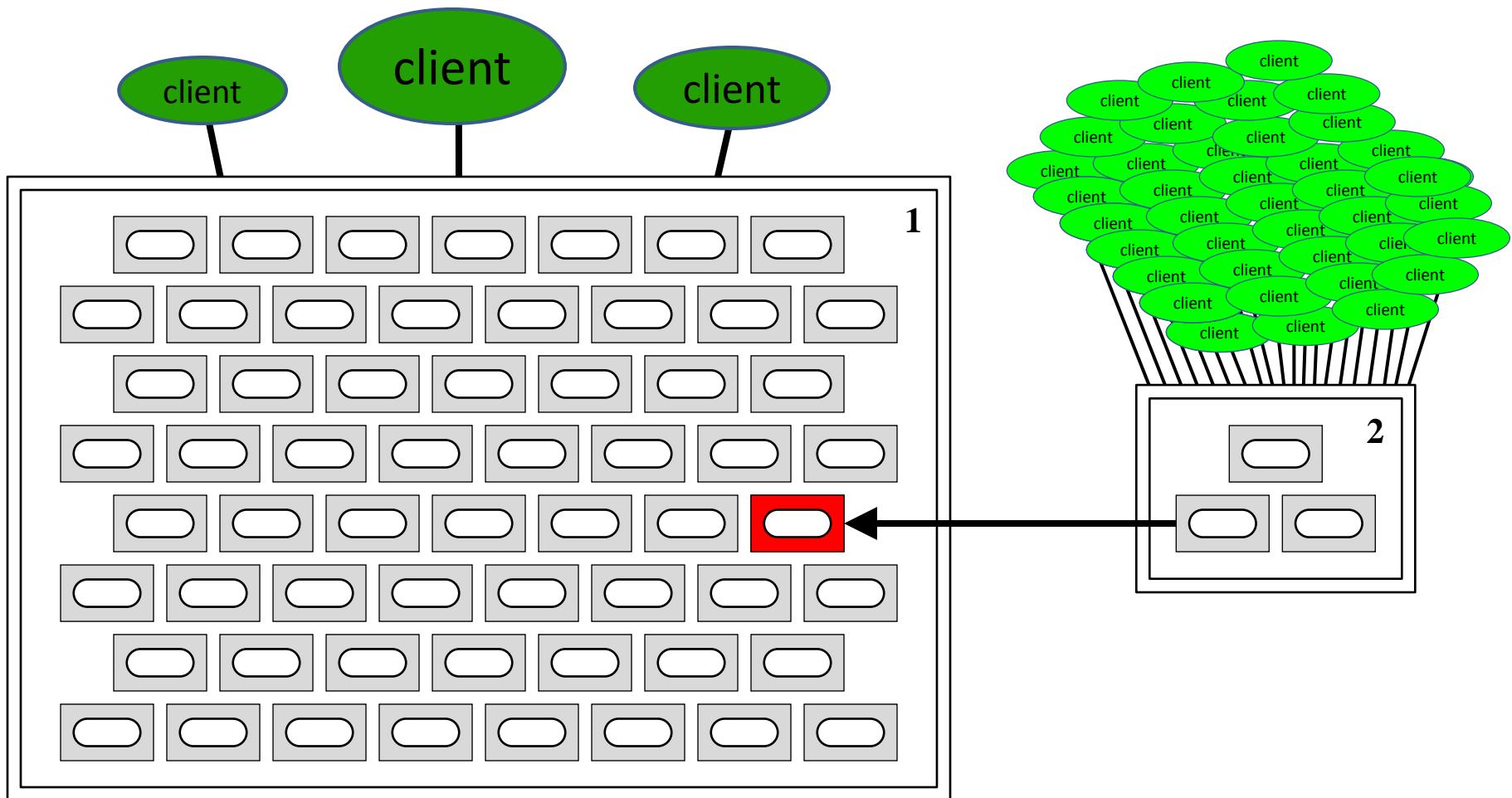
2. Survey of Advanced *Levelization* Techniques

Redundancy



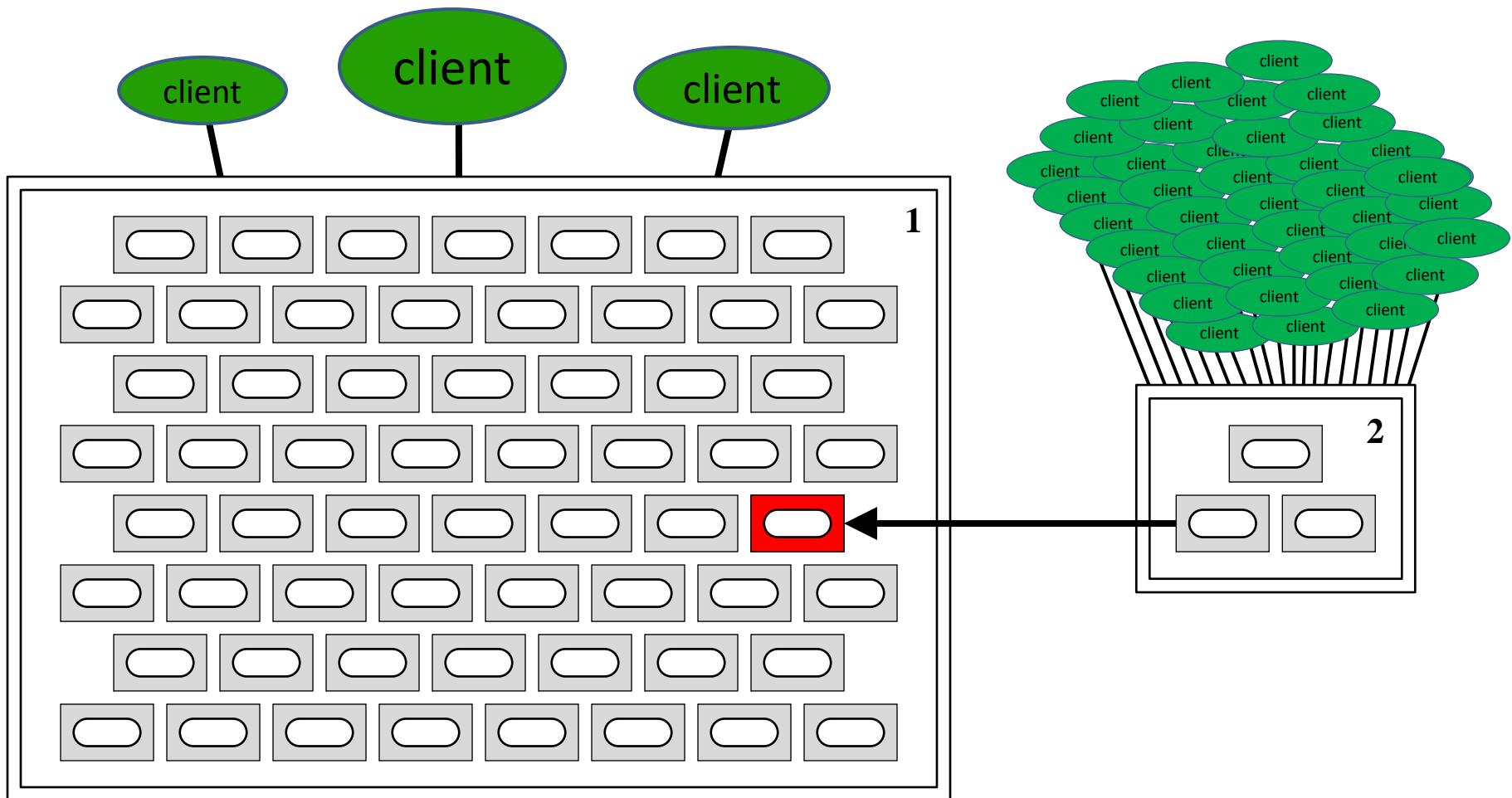
2. Survey of Advanced *Levelization* Techniques

Redundancy



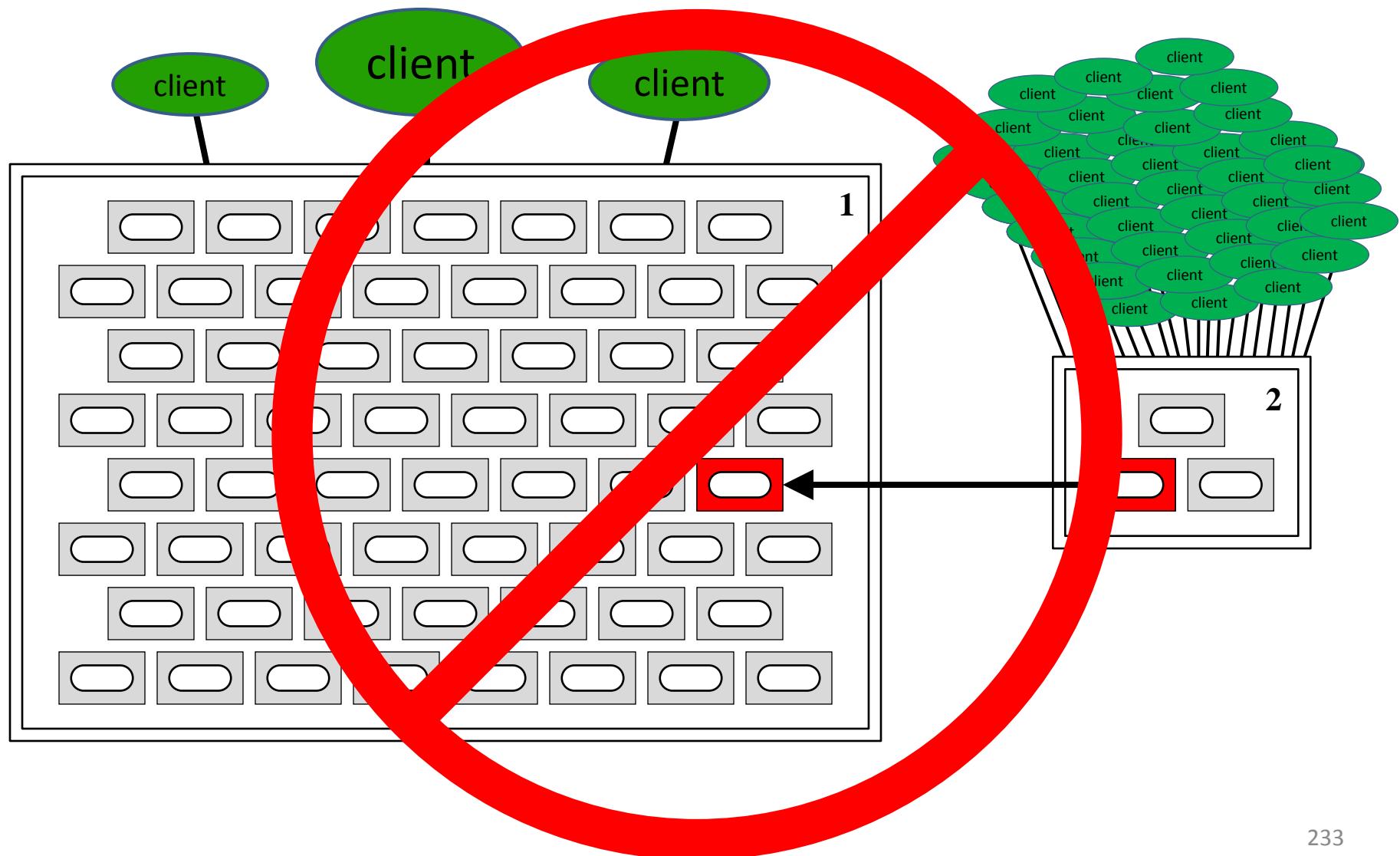
2. Survey of Advanced *Levelization* Techniques

Redundancy



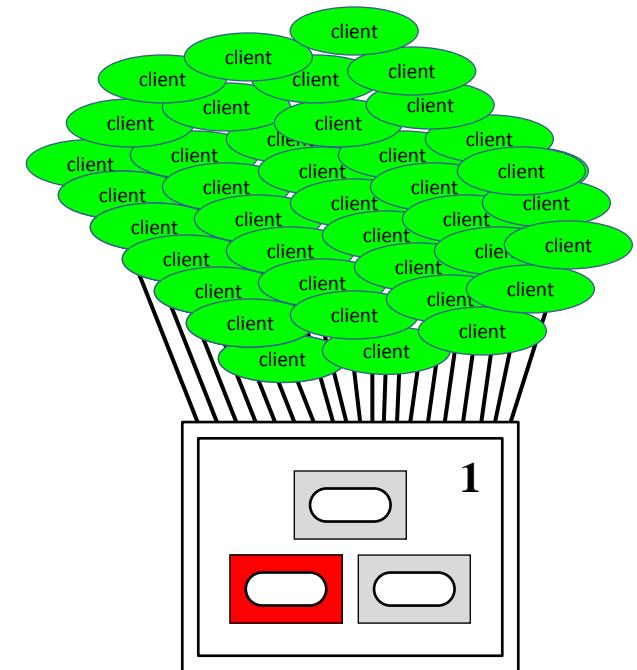
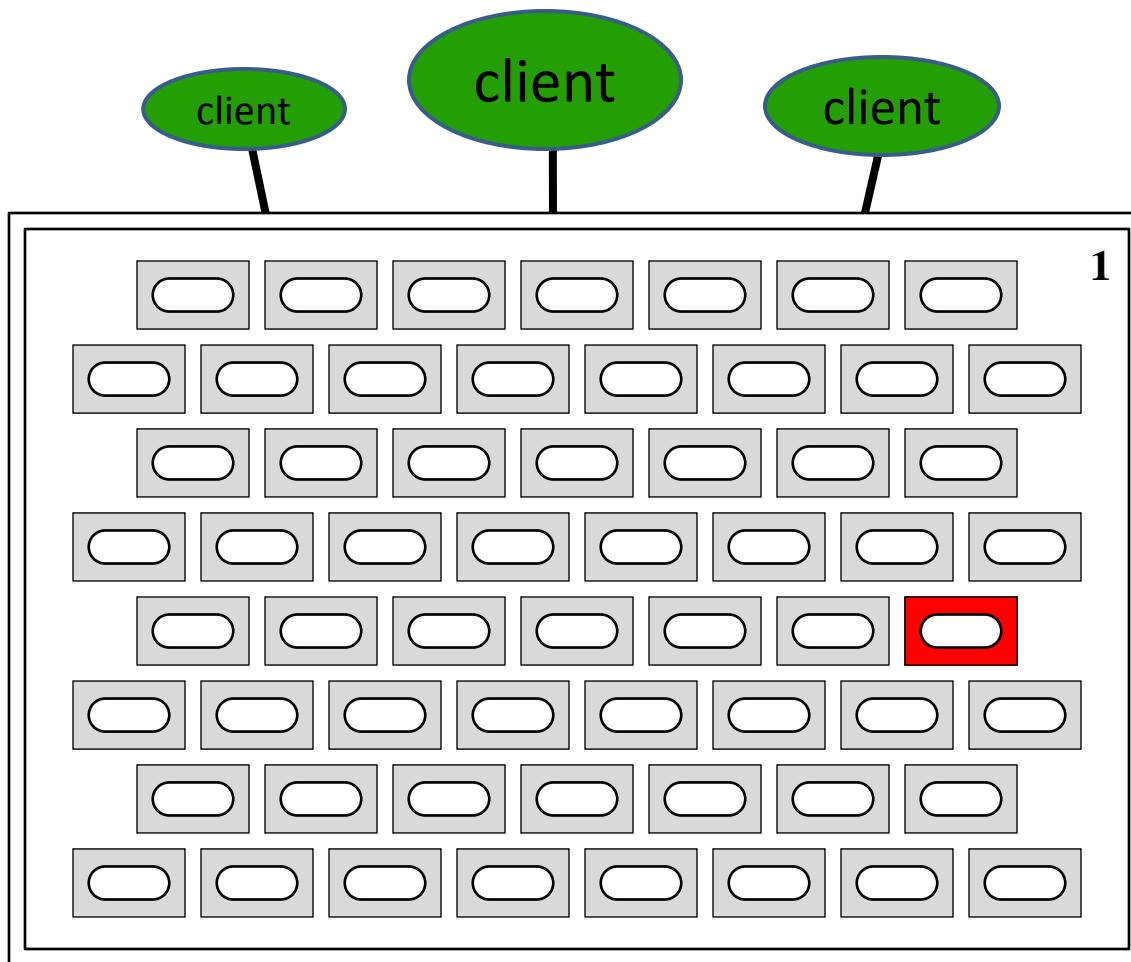
2. Survey of Advanced *Levelization* Techniques

Redundancy



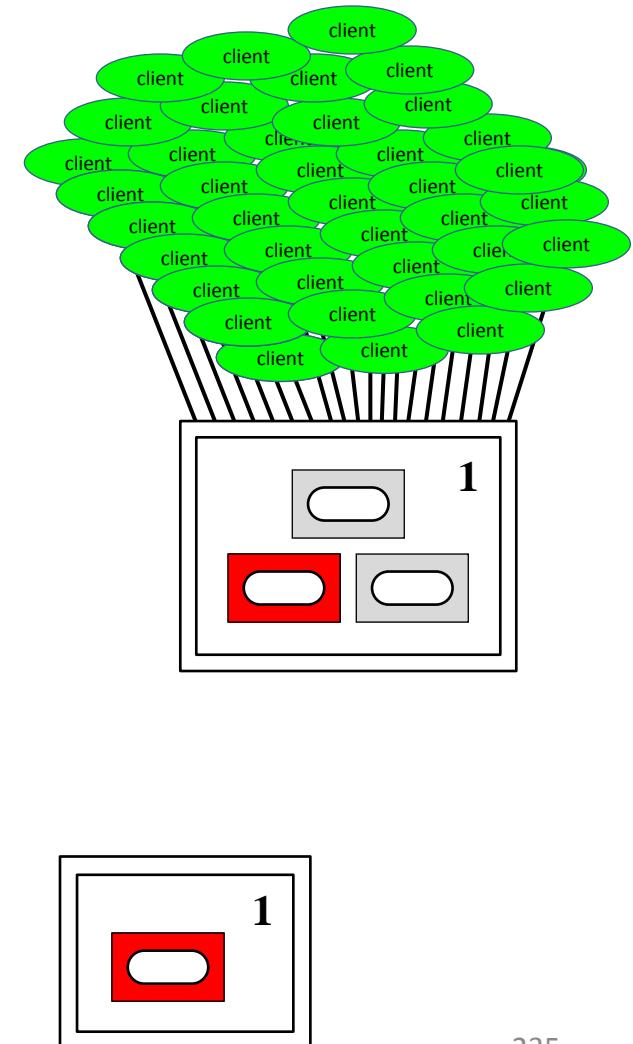
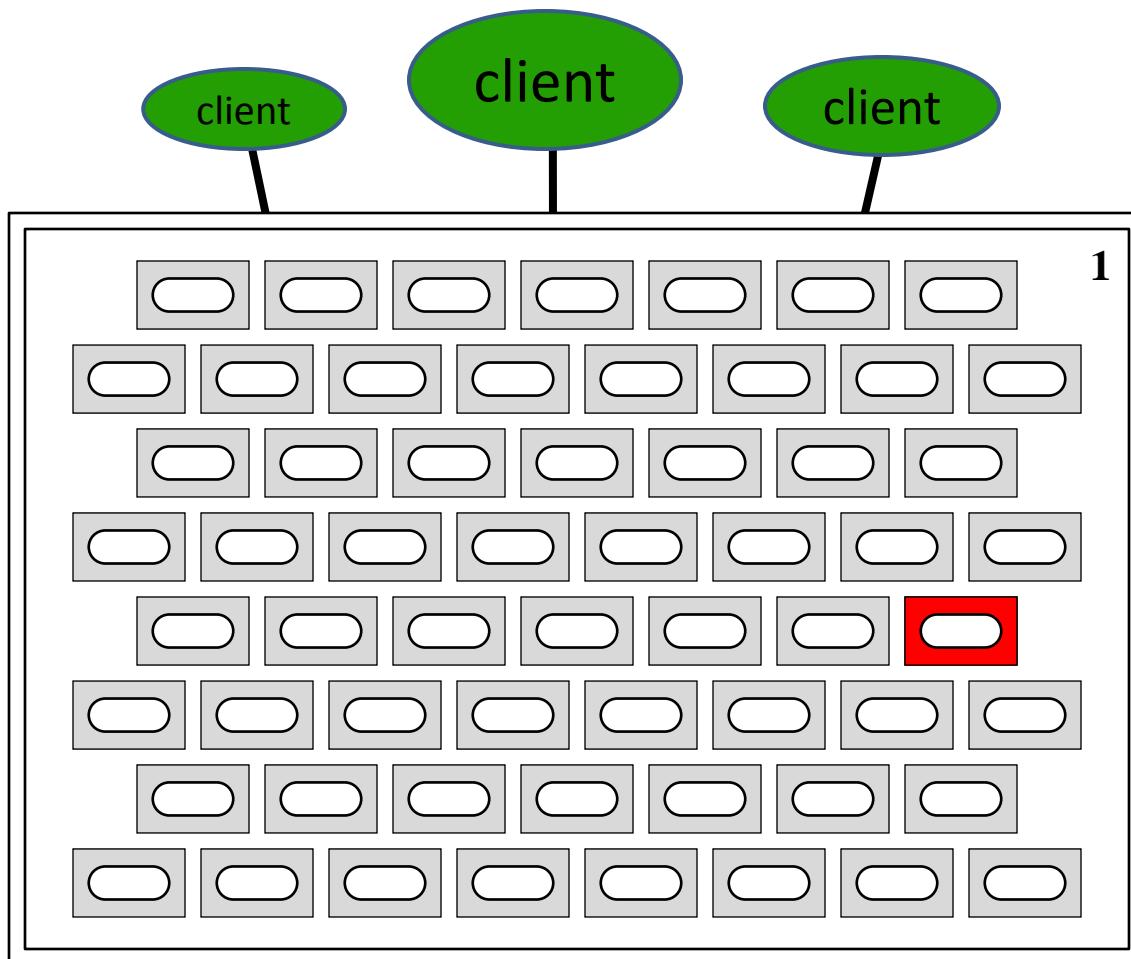
2. Survey of Advanced *Levelization* Techniques

Redundancy



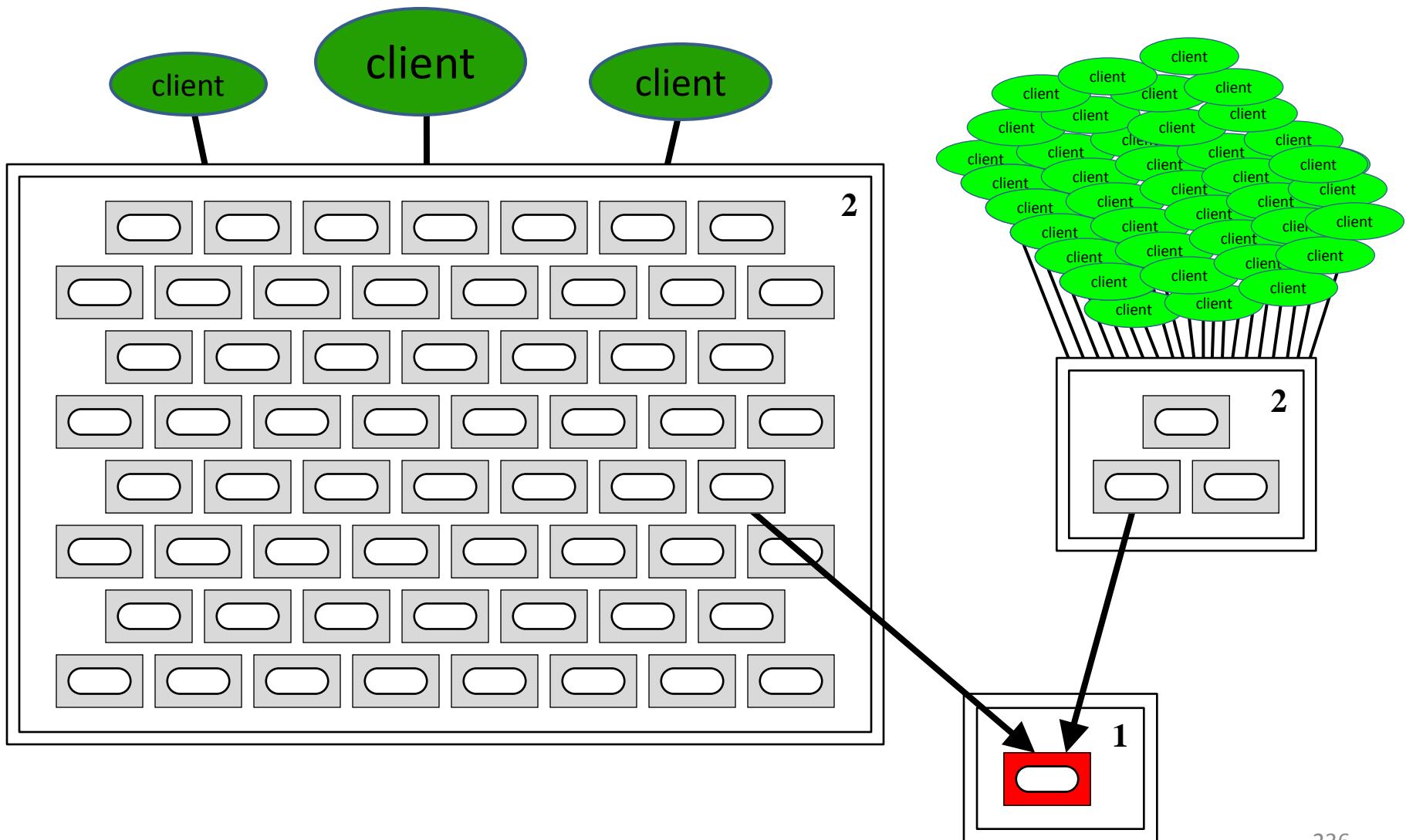
2. Survey of Advanced *Levelization* Techniques

Redundancy



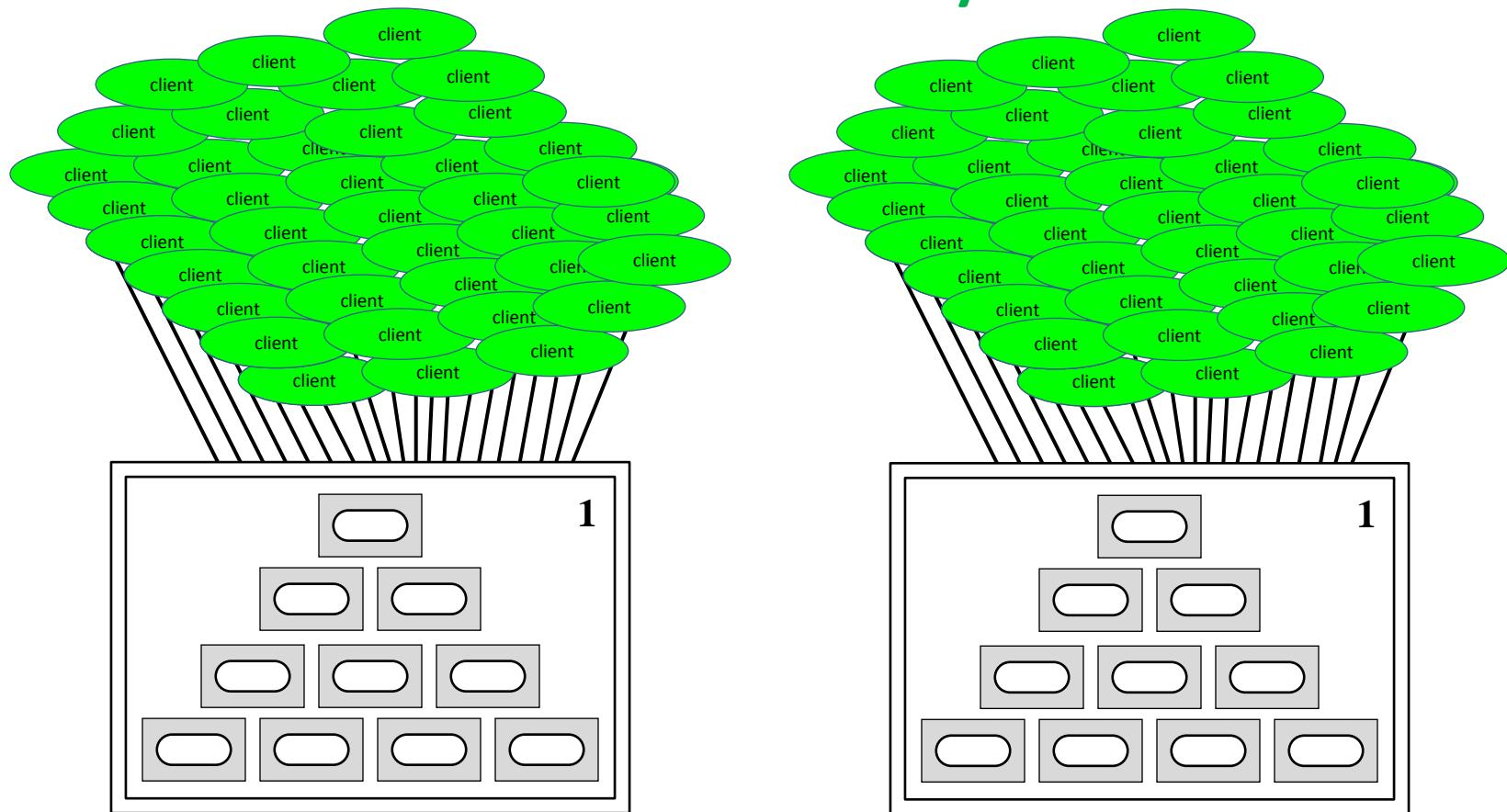
2. Survey of Advanced *Levelization* Techniques

Redundancy



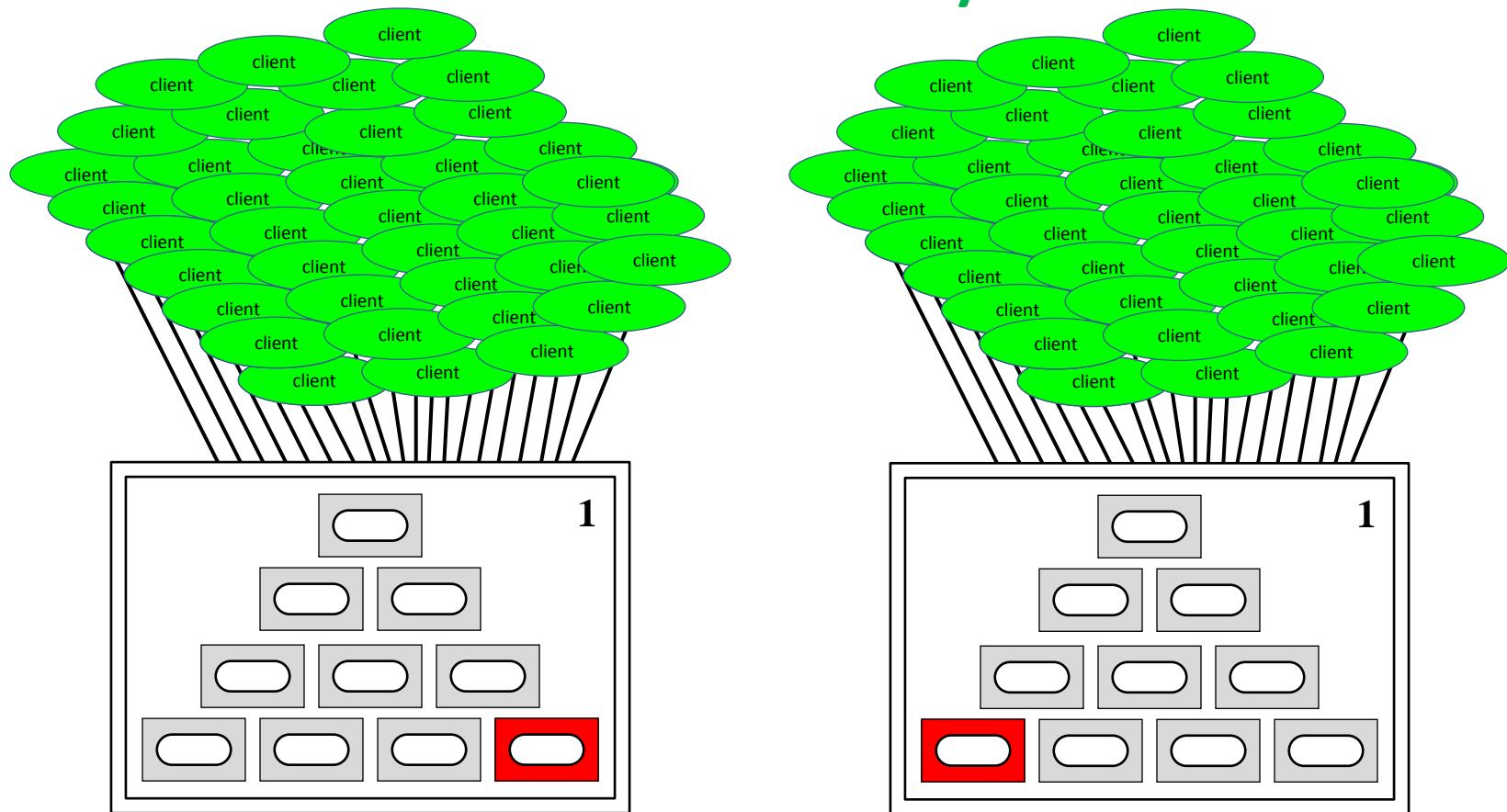
2. Survey of Advanced *Levelization* Techniques

Redundancy



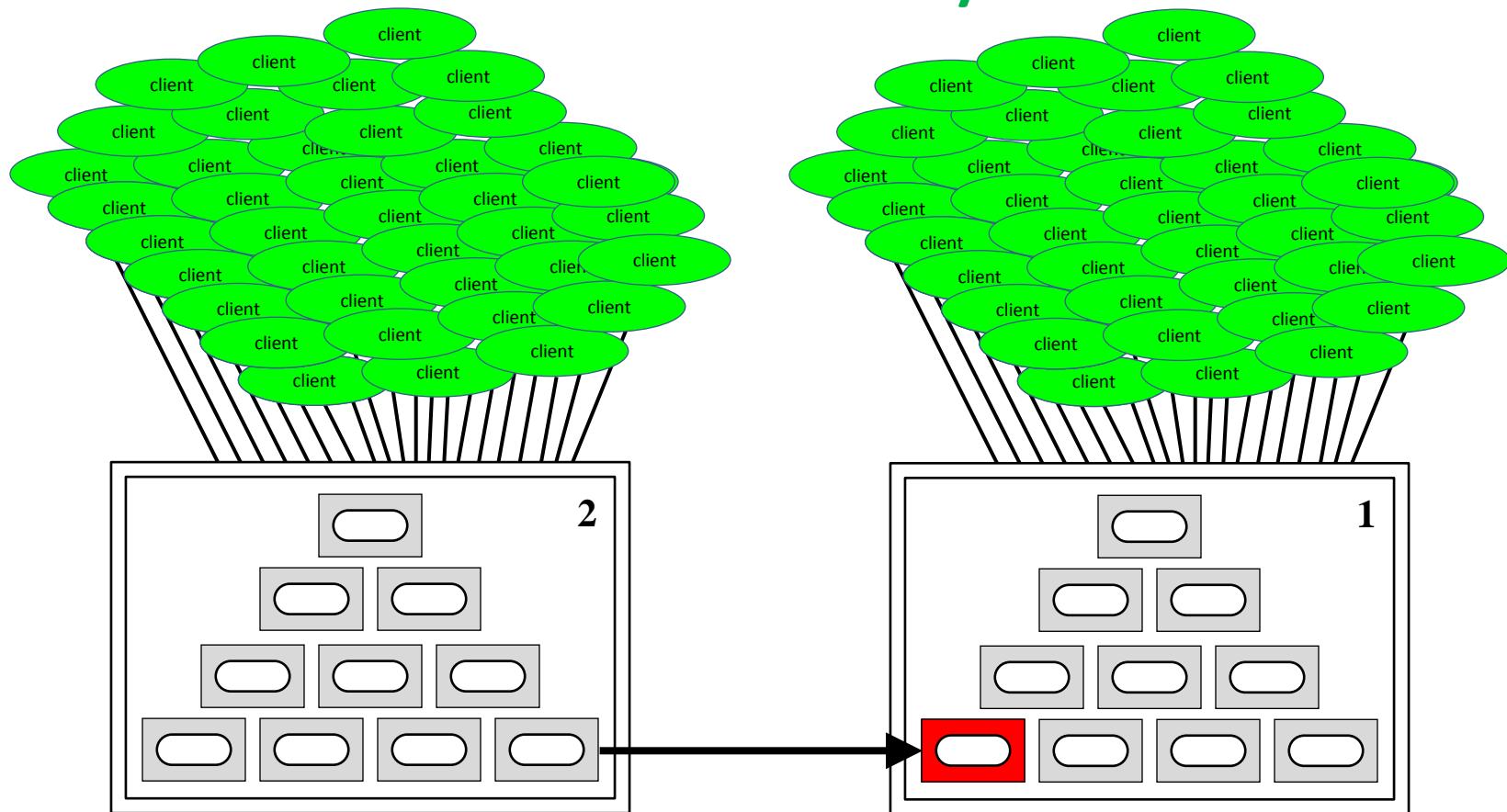
2. Survey of Advanced *Levelization* Techniques

Redundancy



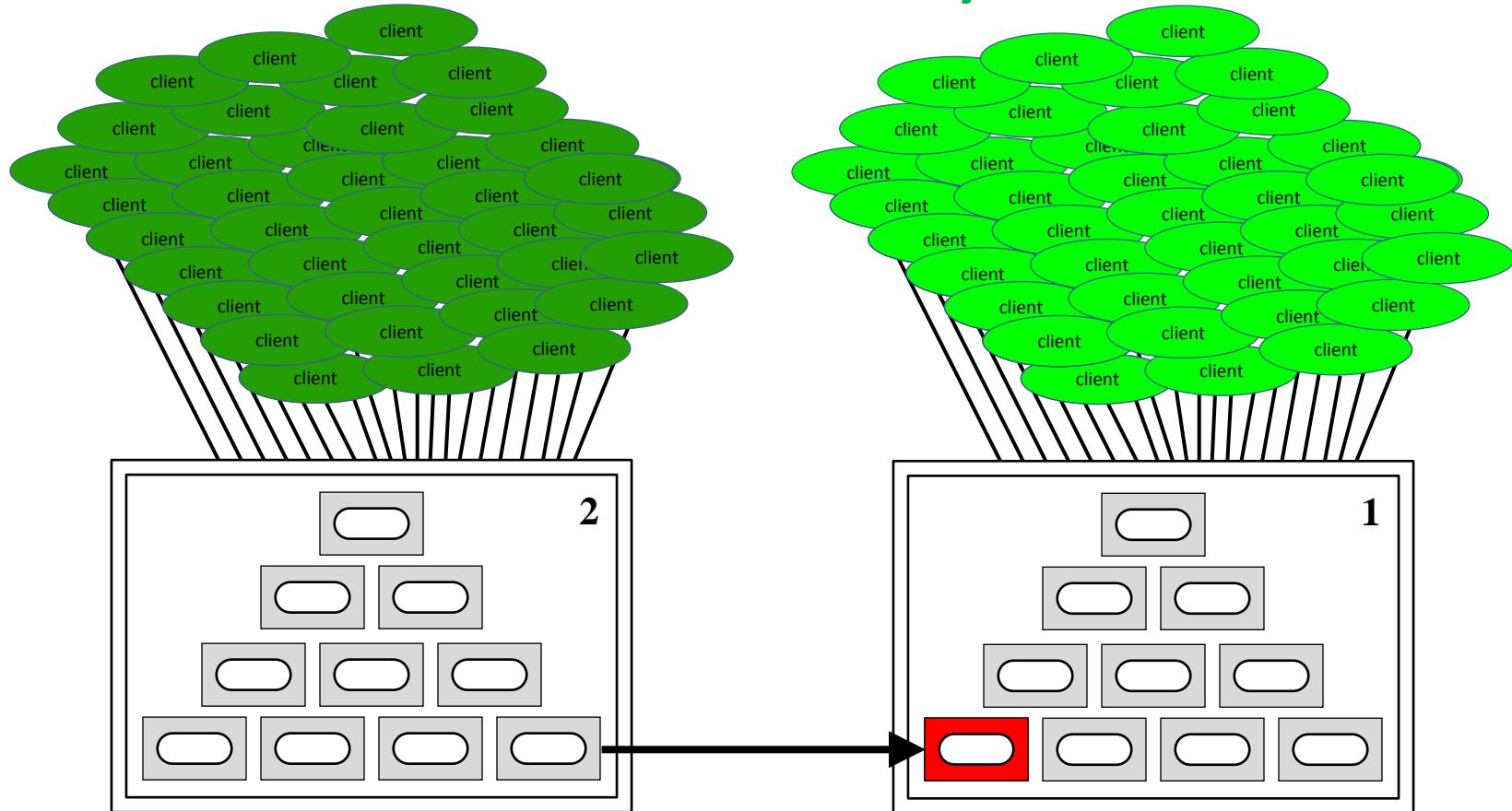
2. Survey of Advanced *Levelization* Techniques

Redundancy



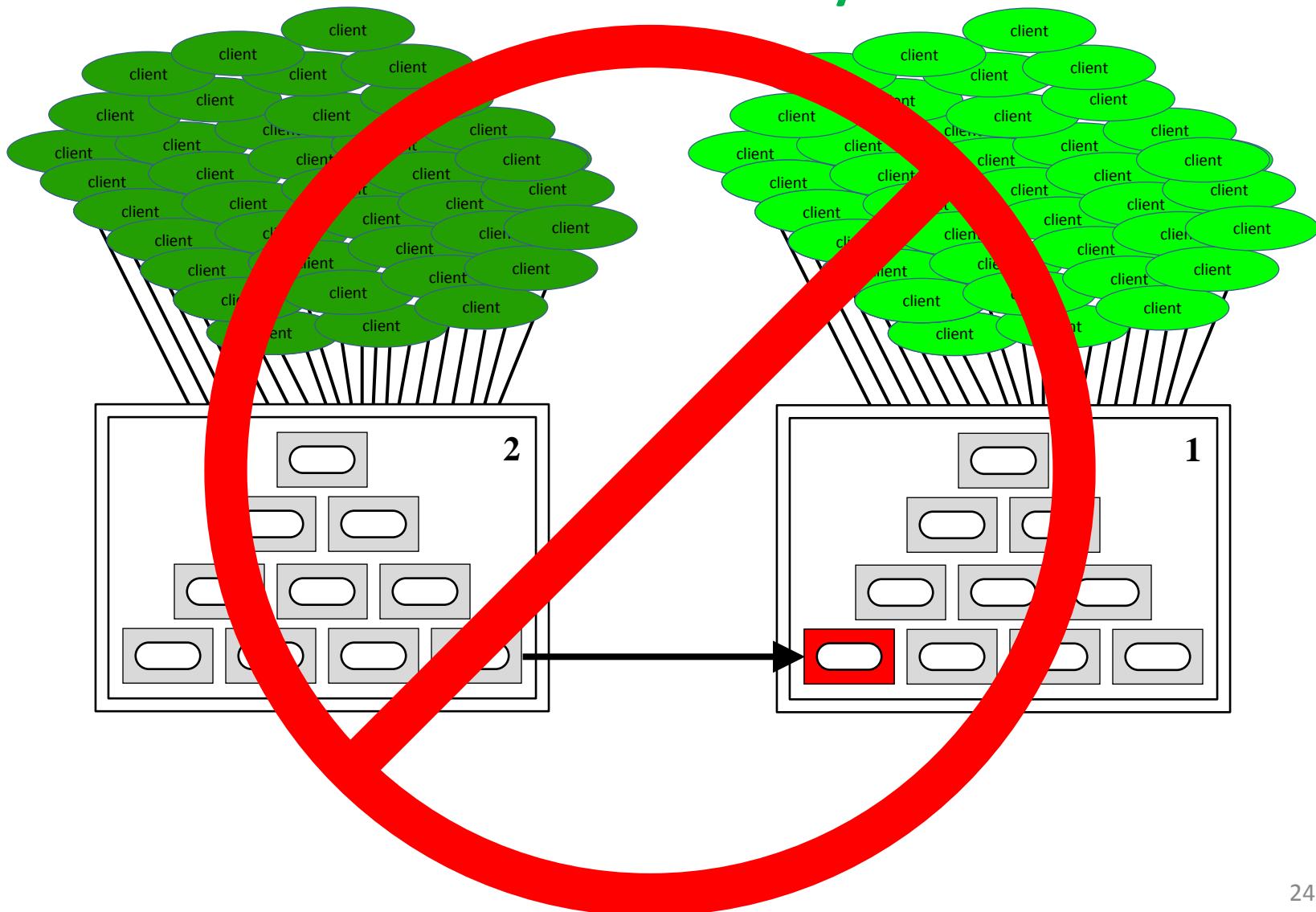
2. Survey of Advanced *Levelization* Techniques

Redundancy



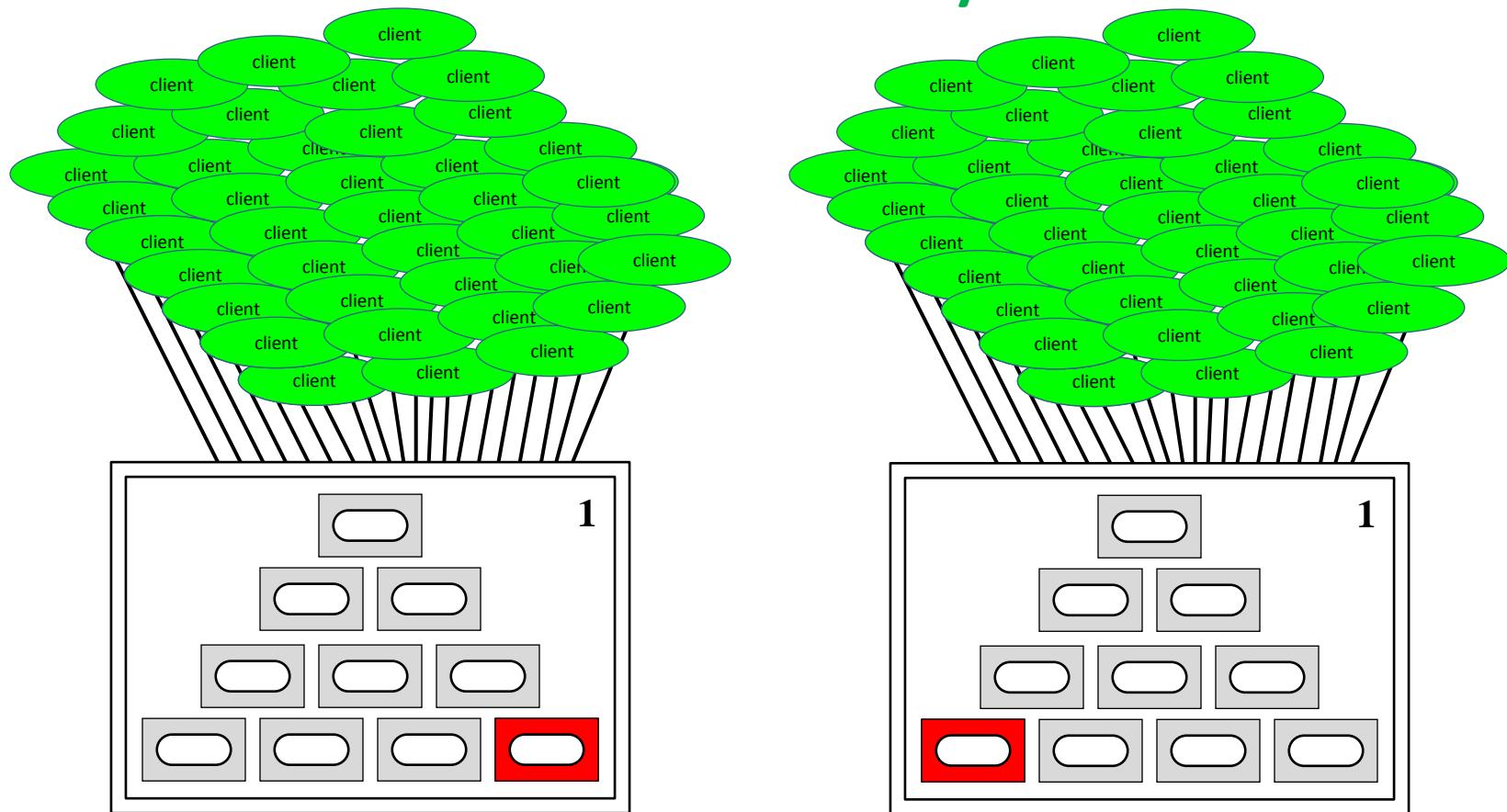
2. Survey of Advanced *Levelization* Techniques

Redundancy



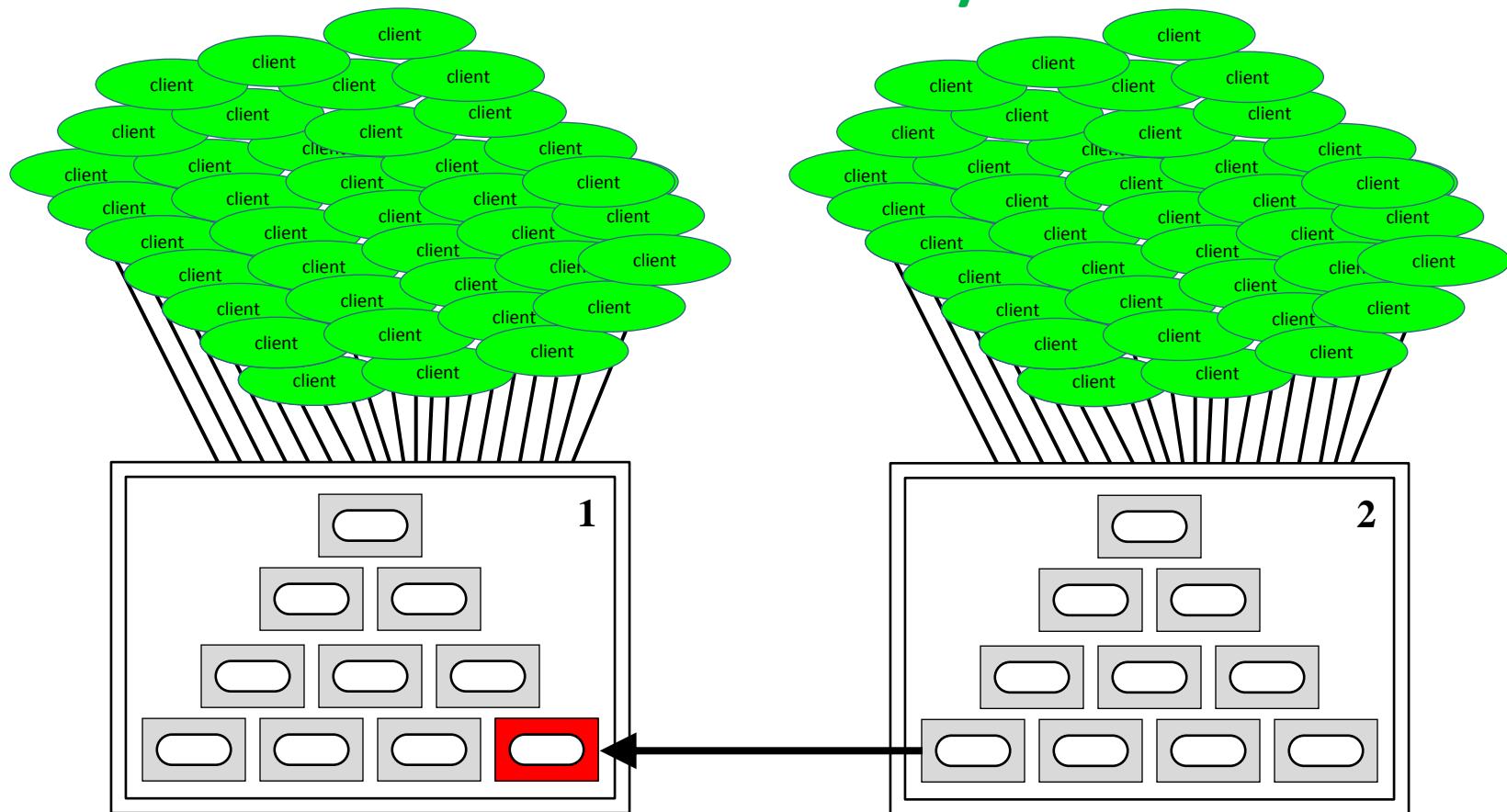
2. Survey of Advanced *Levelization* Techniques

Redundancy



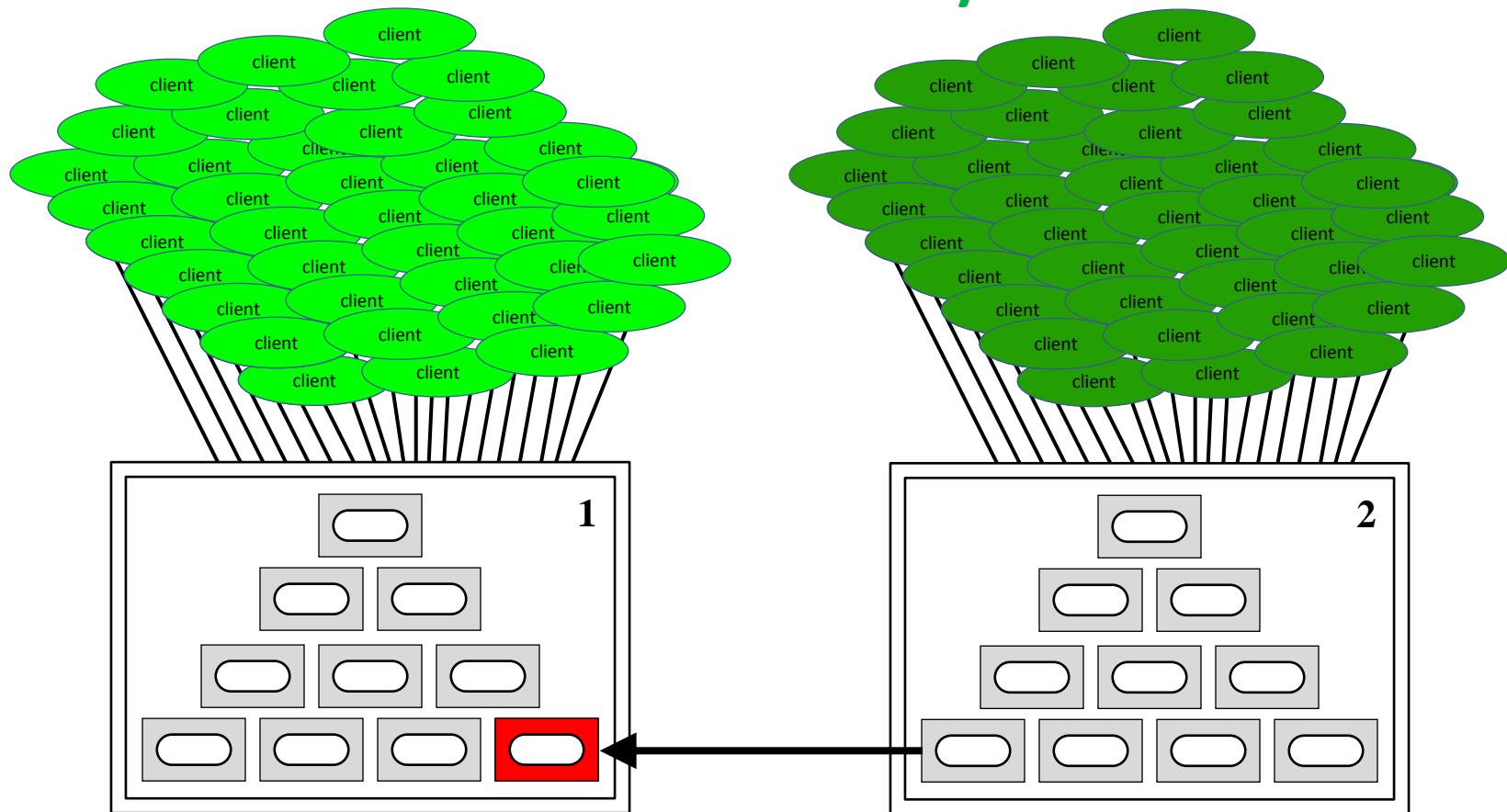
2. Survey of Advanced *Levelization* Techniques

Redundancy



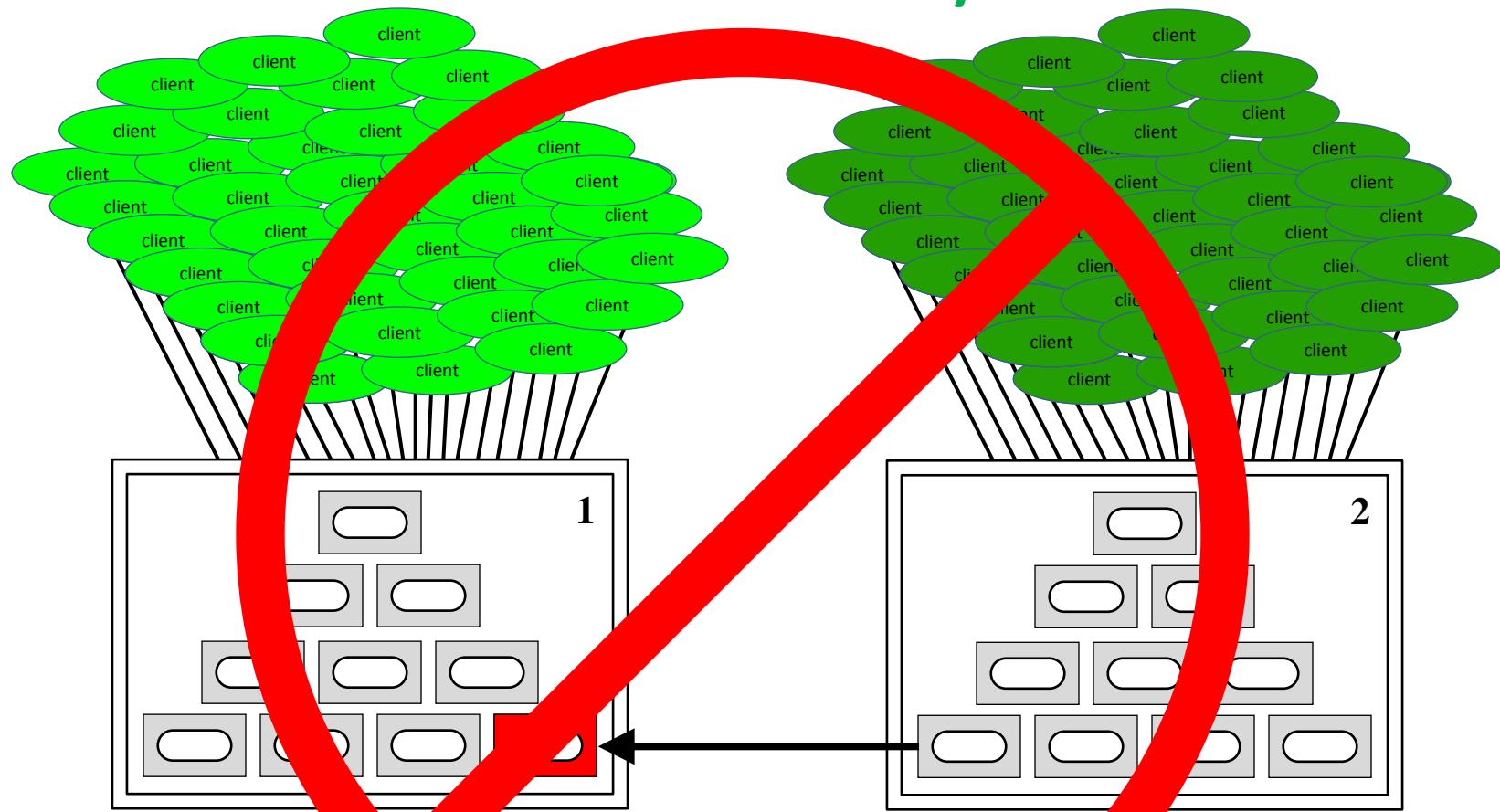
2. Survey of Advanced *Levelization* Techniques

Redundancy



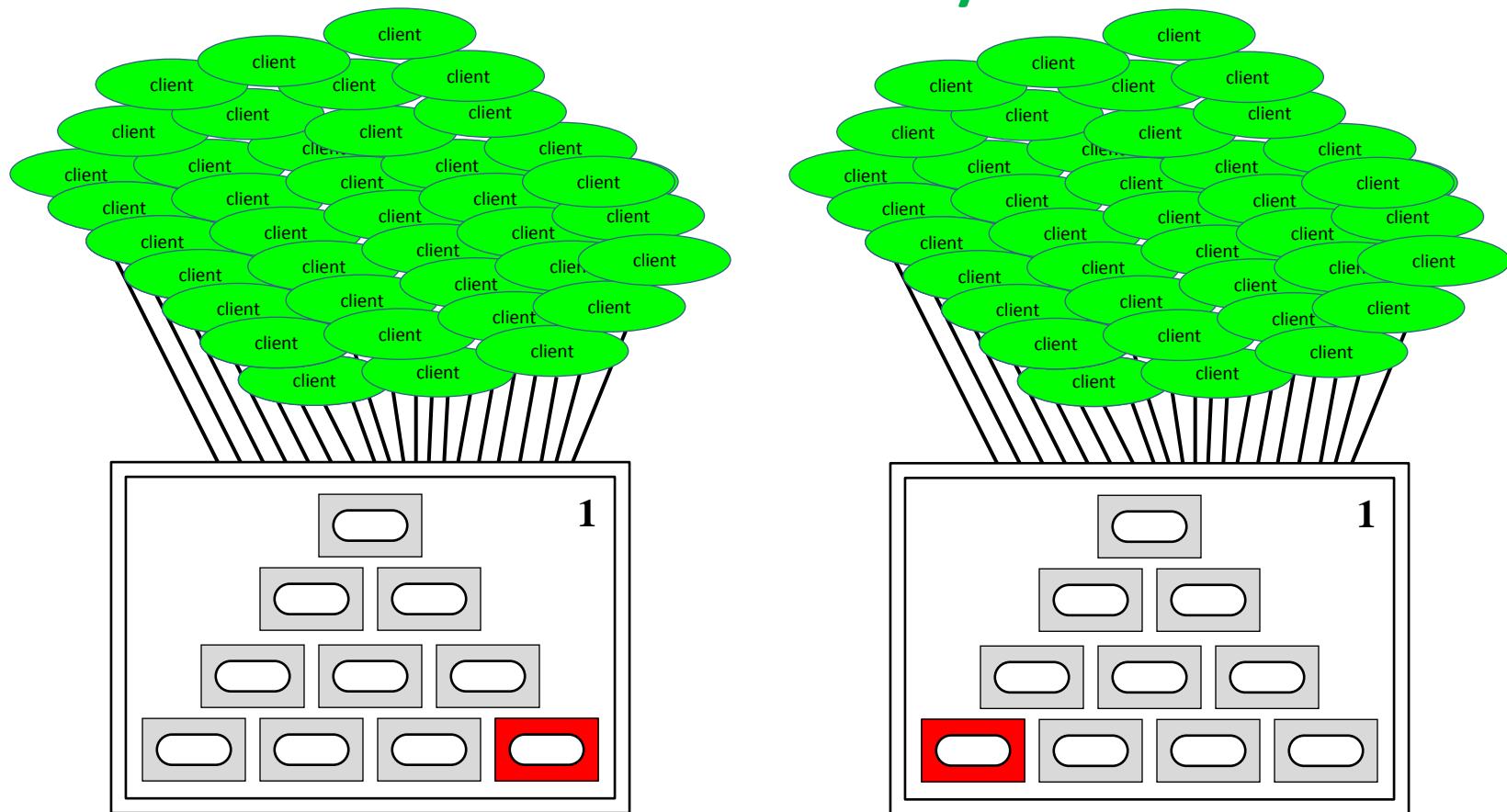
2. Survey of Advanced *Levelization* Techniques

Redundancy



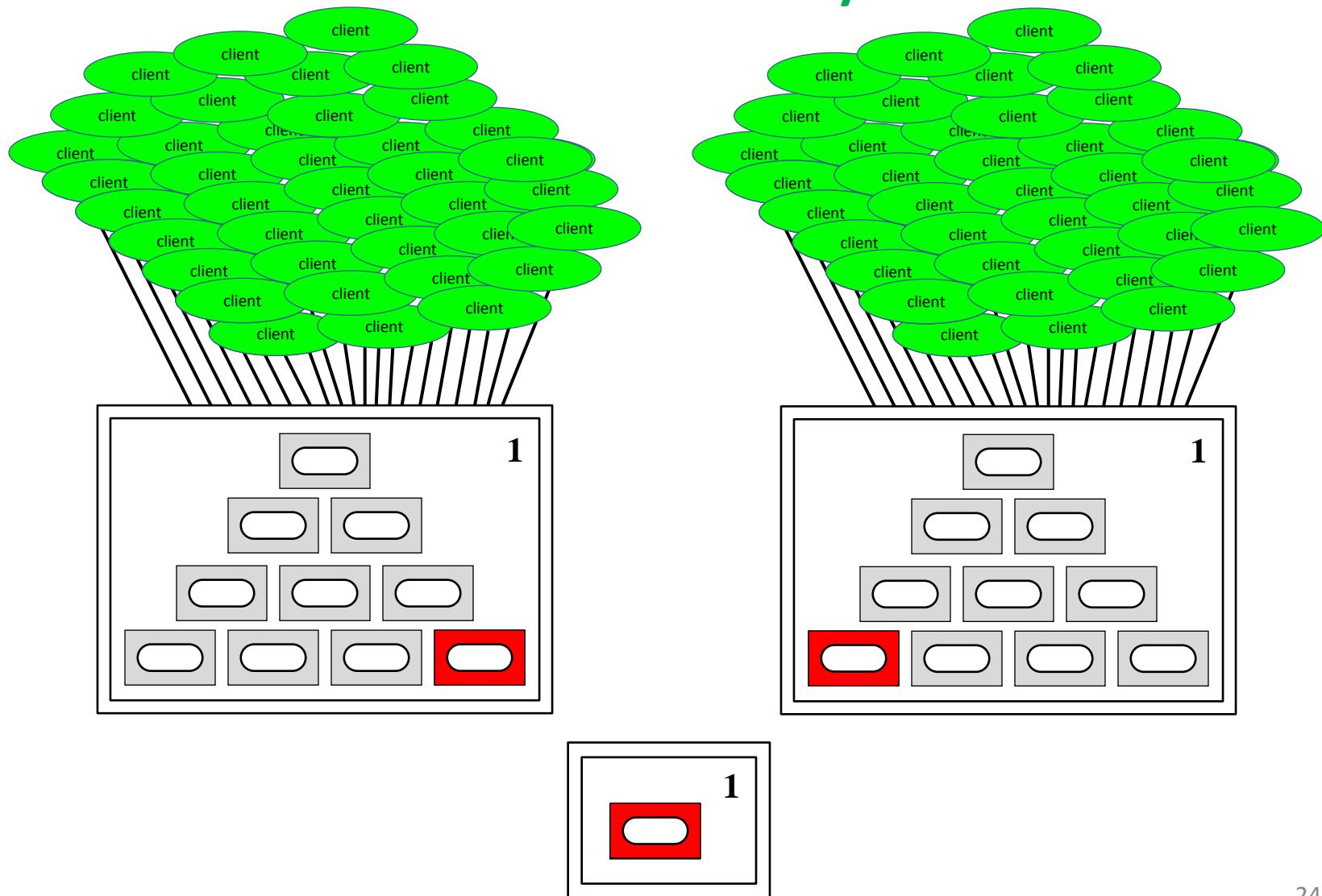
2. Survey of Advanced *Levelization* Techniques

Redundancy



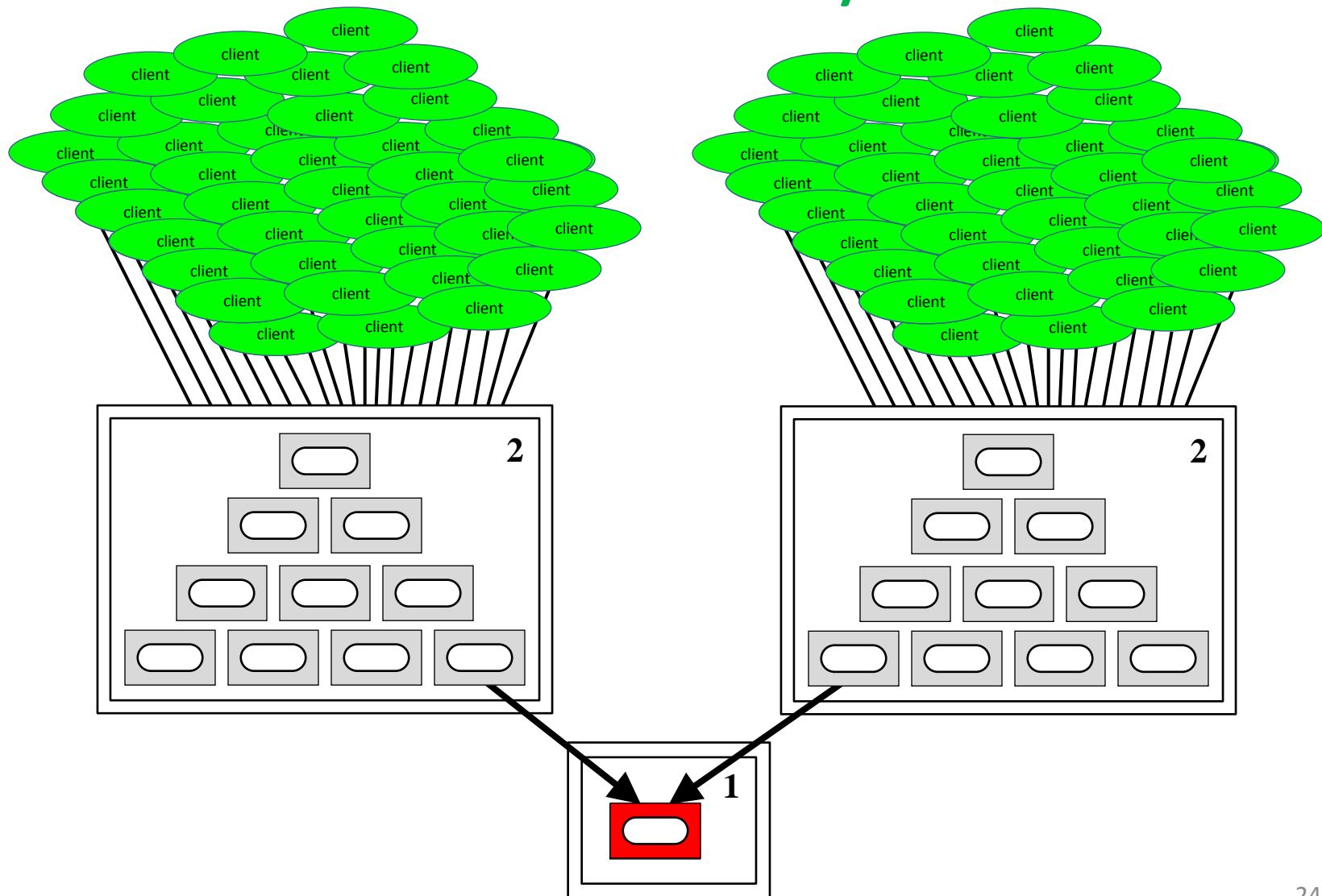
2. Survey of Advanced *Levelization* Techniques

Redundancy



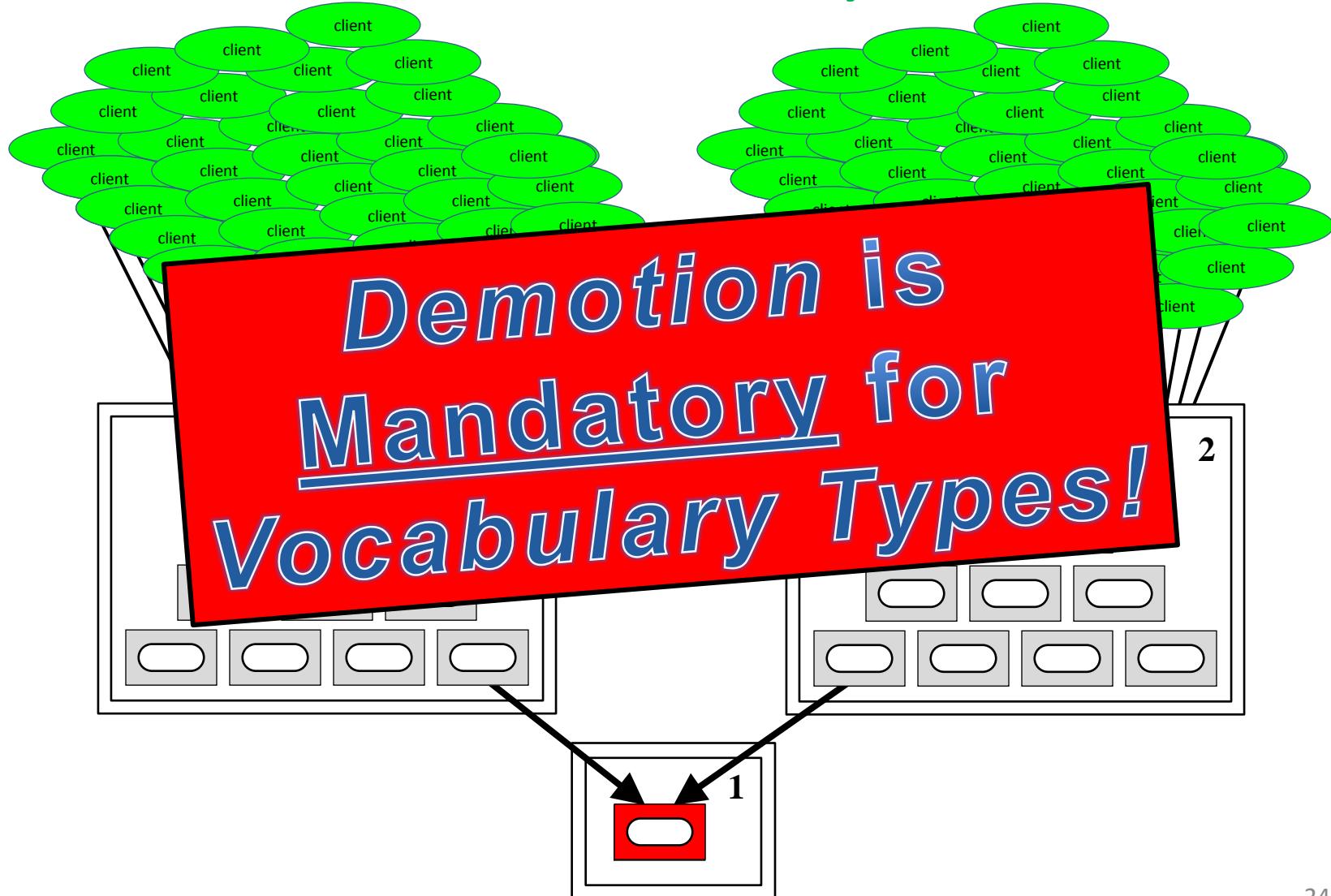
2. Survey of Advanced *Levelization* Techniques

Redundancy



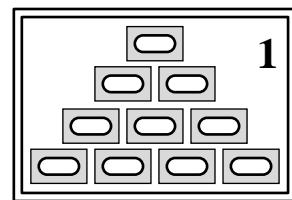
2. Survey of Advanced *Levelization* Techniques

Redundancy



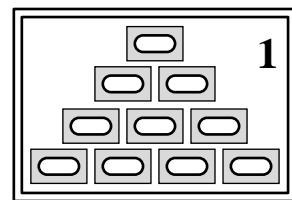
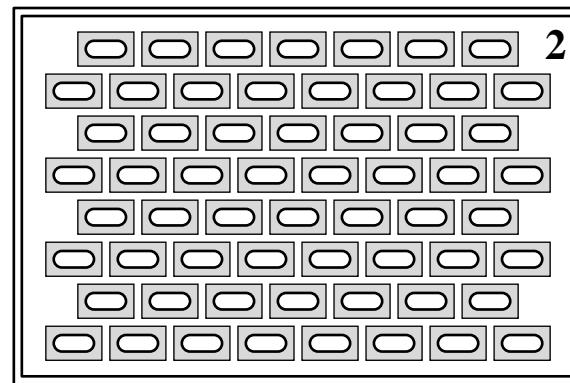
2. Survey of Advanced *Levelization* Techniques

Redundancy



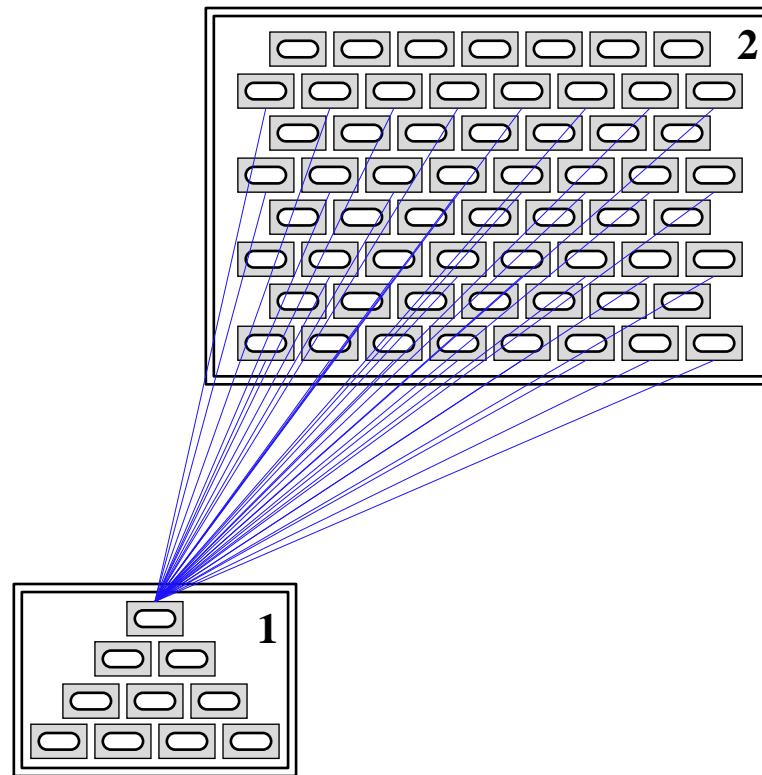
2. Survey of Advanced *Levelization* Techniques

Redundancy



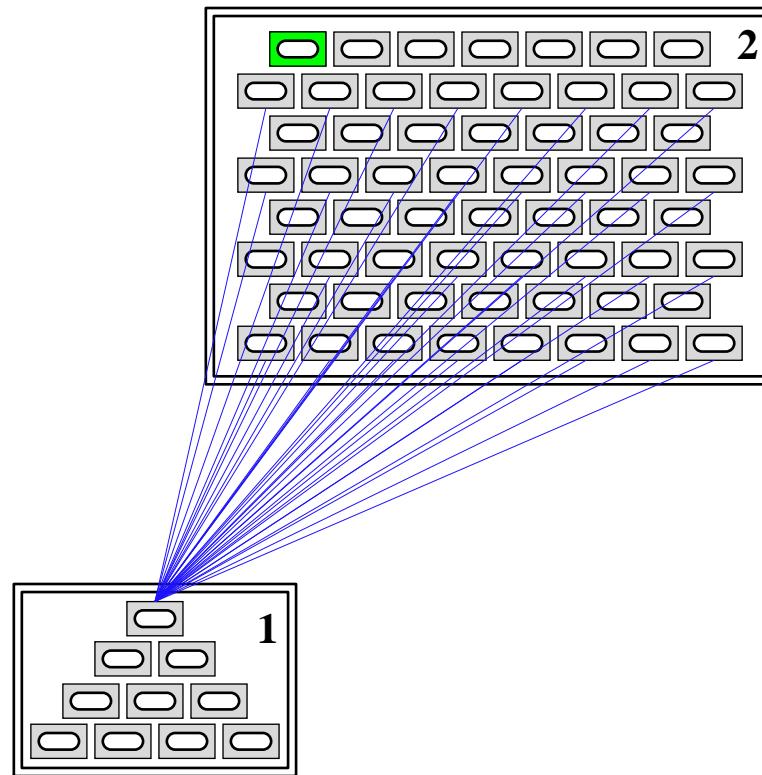
2. Survey of Advanced *Levelization* Techniques

Redundancy



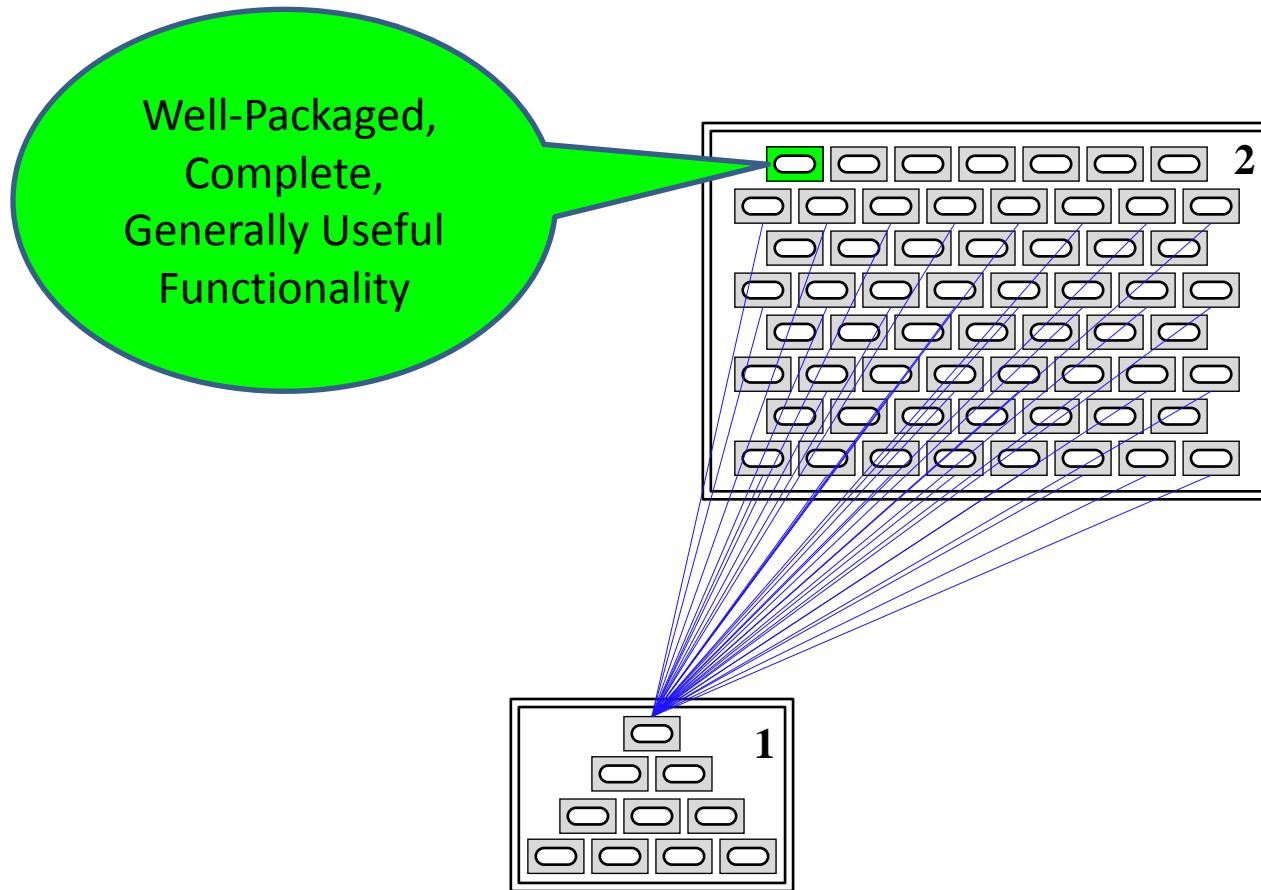
2. Survey of Advanced *Levelization* Techniques

Redundancy



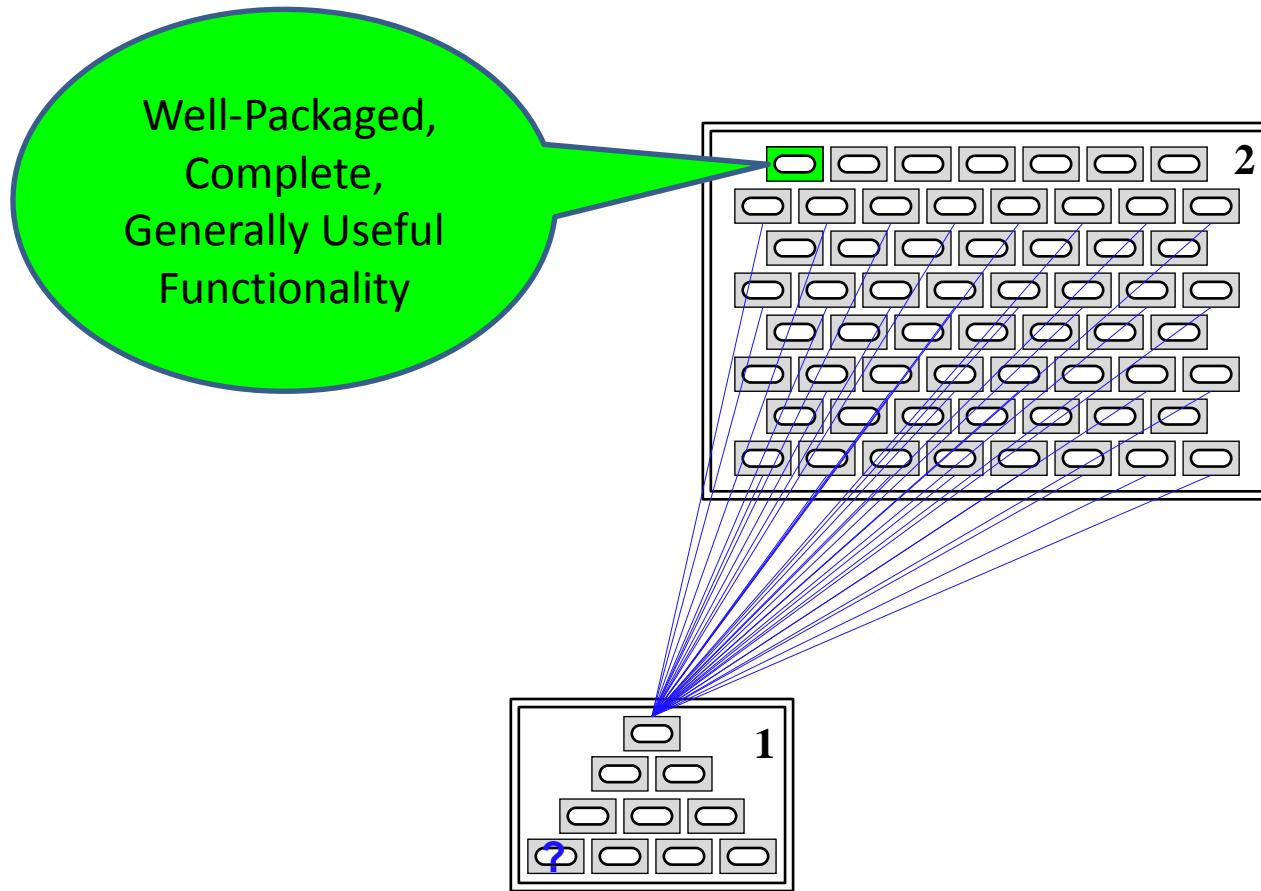
2. Survey of Advanced *Levelization* Techniques

Redundancy



2. Survey of Advanced *Levelization* Techniques

Redundancy

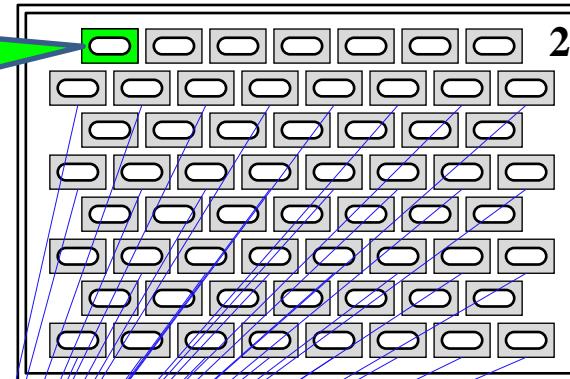


2. Survey of Advanced *Levelization* Techniques

Redundancy

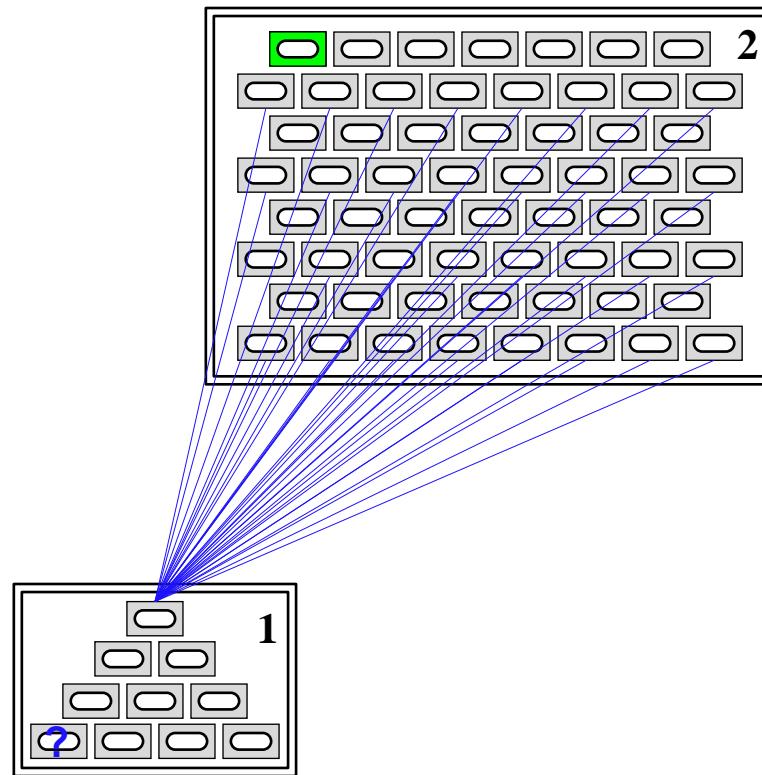
Well-Packaged,
Complete,
Generally Useful
Functionality

Requires Only
Tiny Subset of
Higher-Level
Functionality



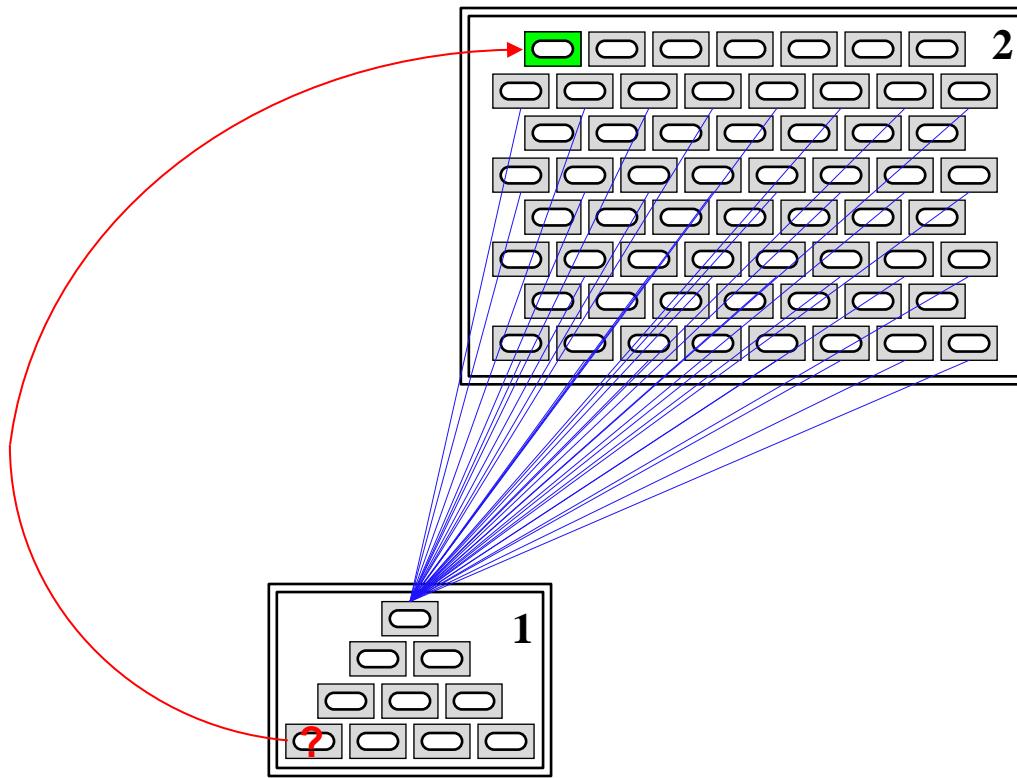
2. Survey of Advanced *Levelization* Techniques

Redundancy



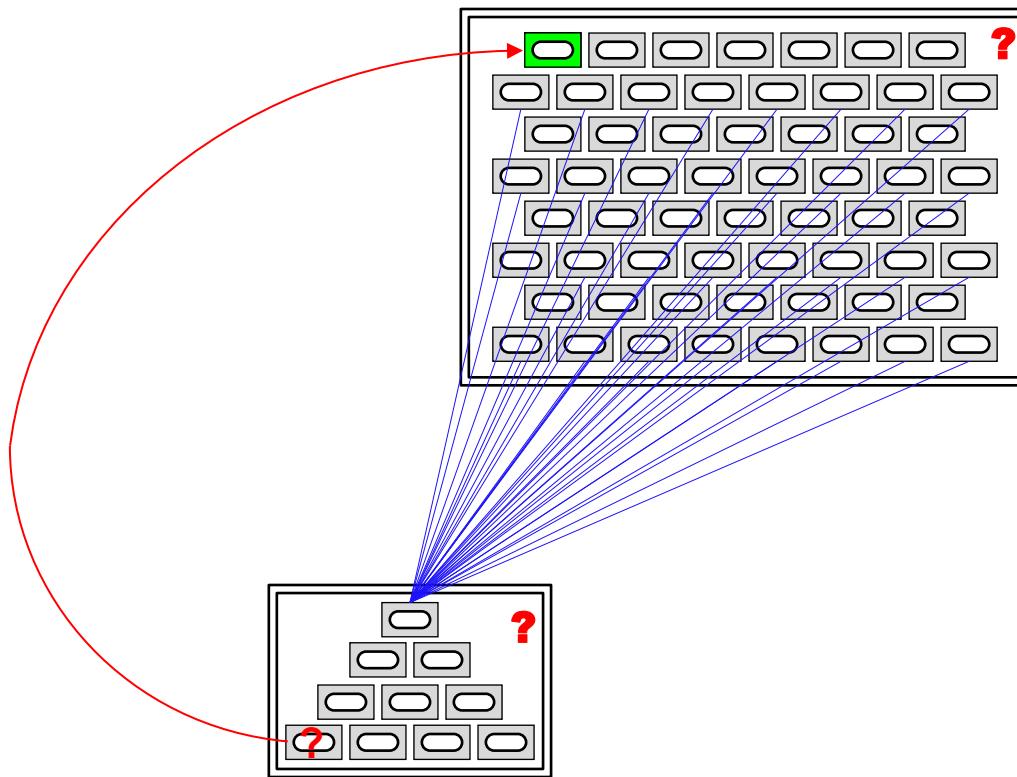
2. Survey of Advanced *Levelization* Techniques

Redundancy



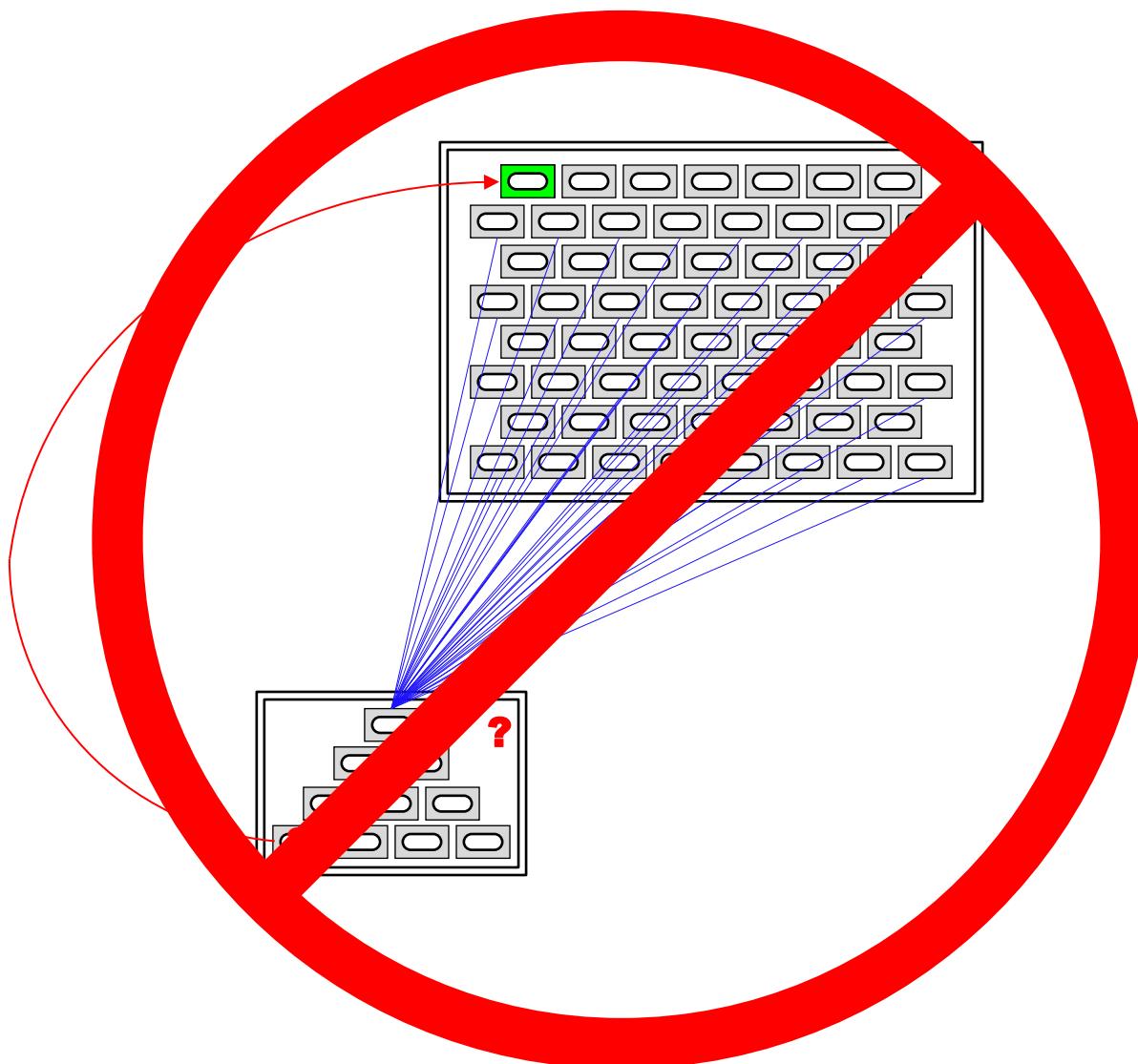
2. Survey of Advanced *Levelization* Techniques

Redundancy



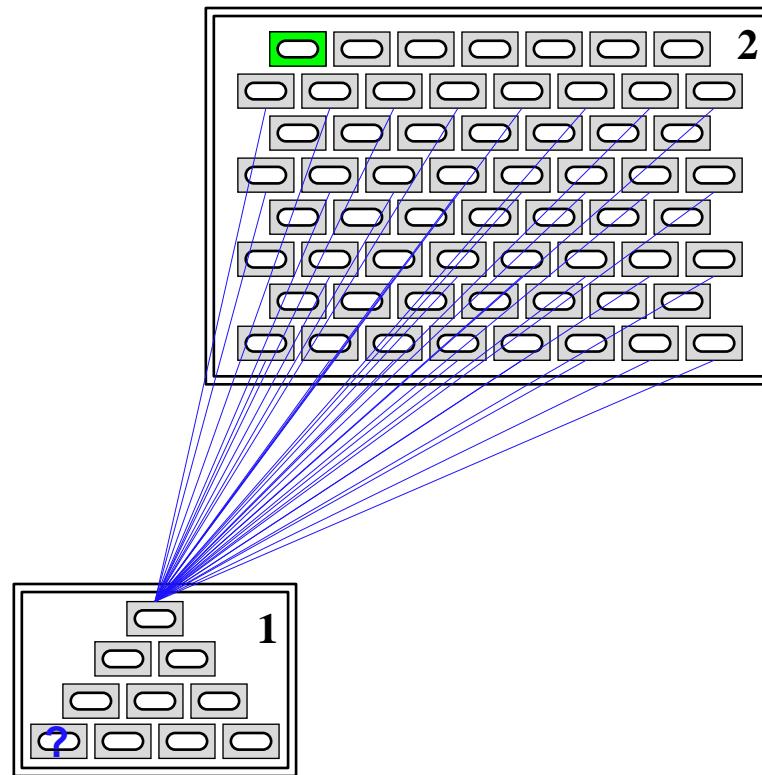
2. Survey of Advanced *Levelization* Techniques

Redundancy



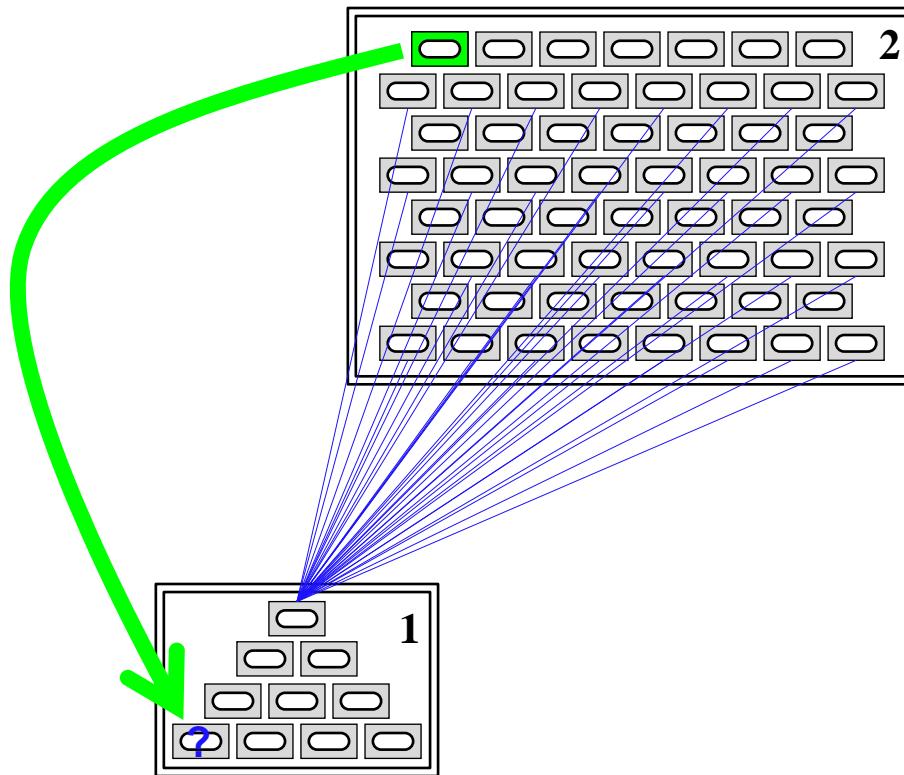
2. Survey of Advanced *Levelization* Techniques

Redundancy



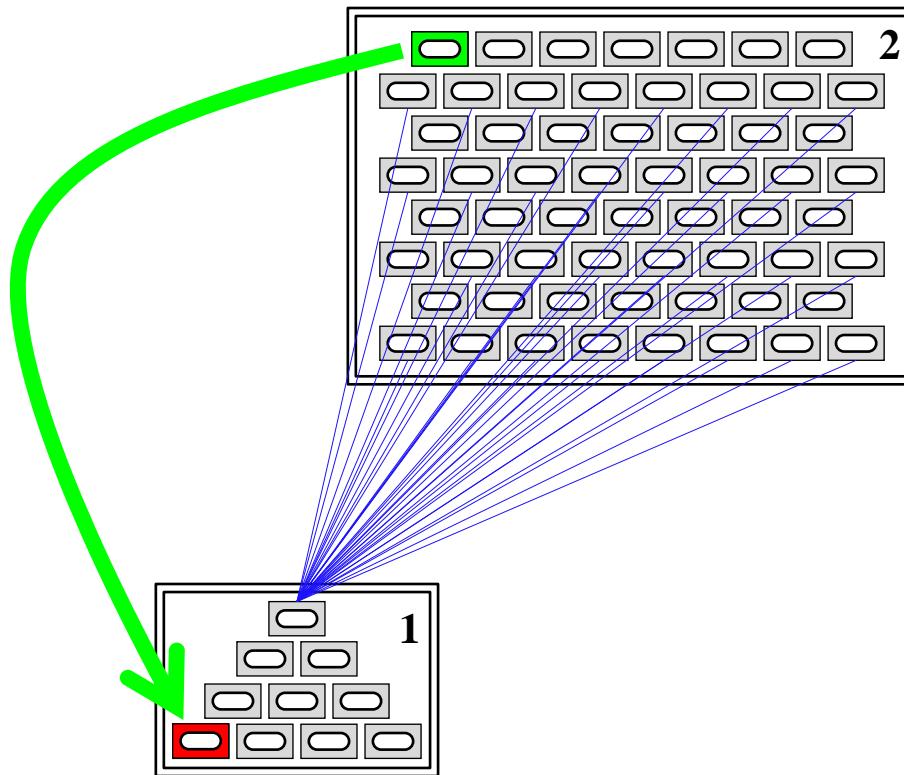
2. Survey of Advanced *Levelization* Techniques

Redundancy



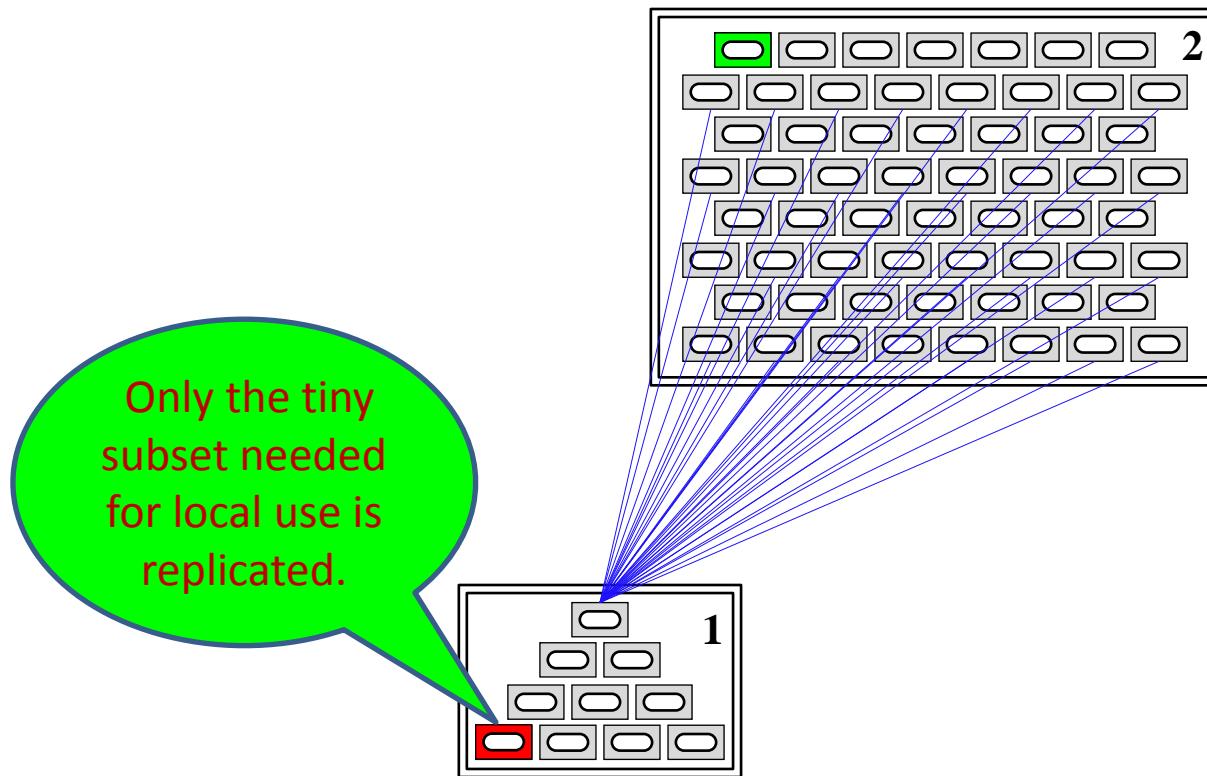
2. Survey of Advanced *Levelization* Techniques

Redundancy



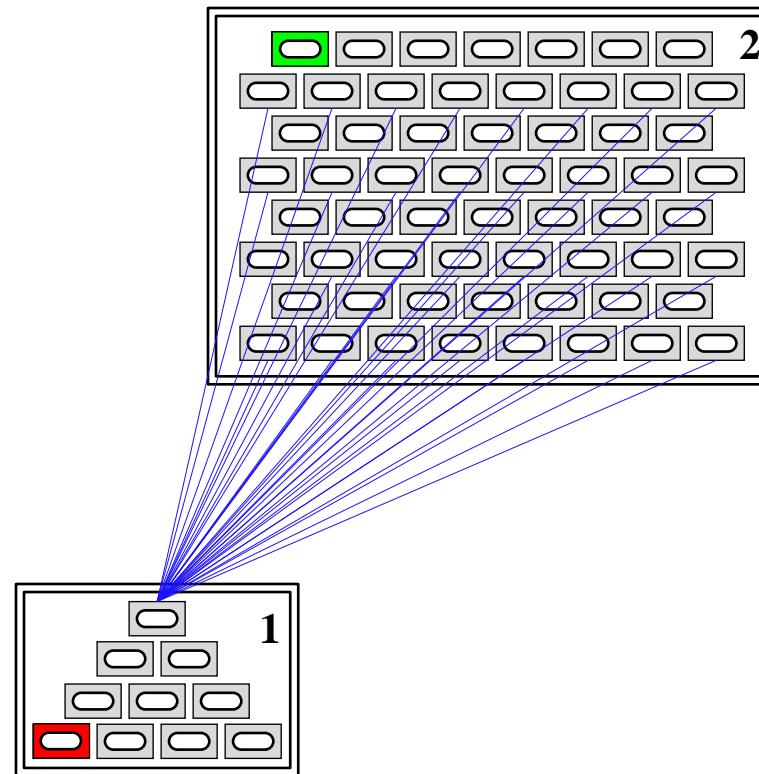
2. Survey of Advanced *Levelization* Techniques

Redundancy



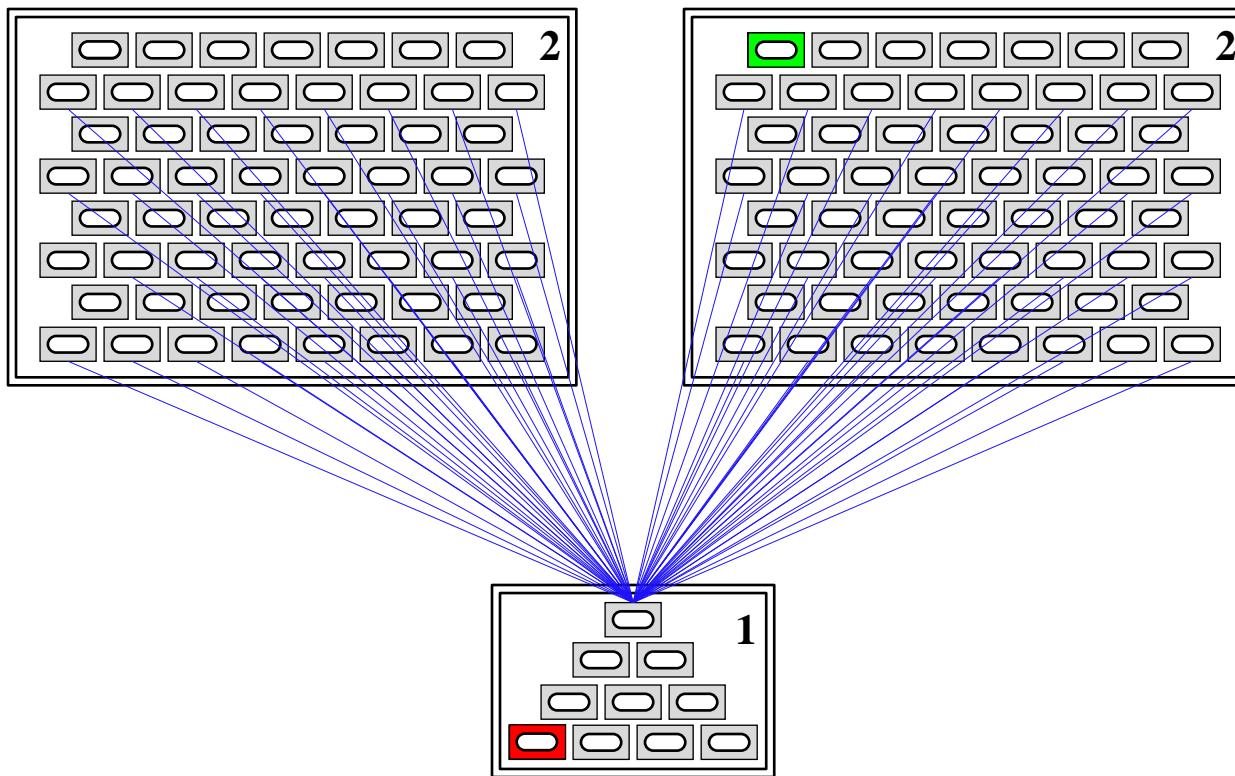
2. Survey of Advanced *Levelization* Techniques

Redundancy



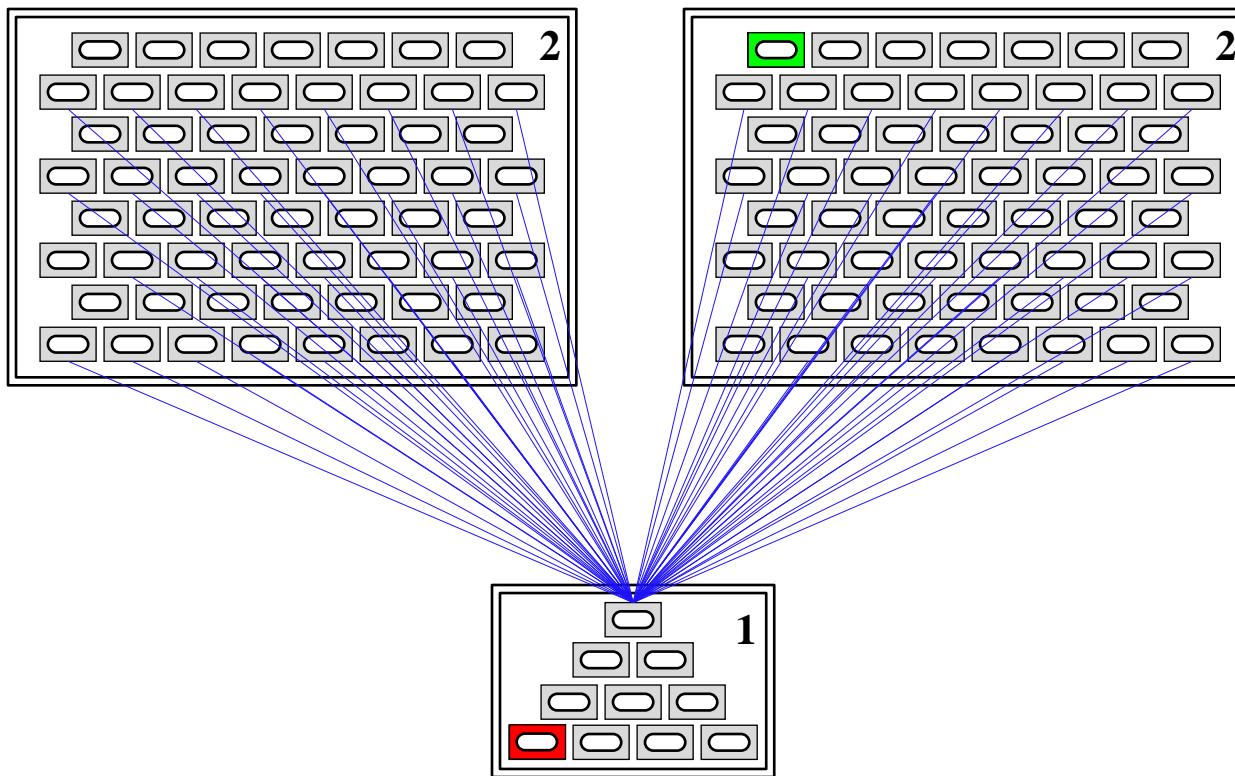
2. Survey of Advanced *Levelization* Techniques

Redundancy



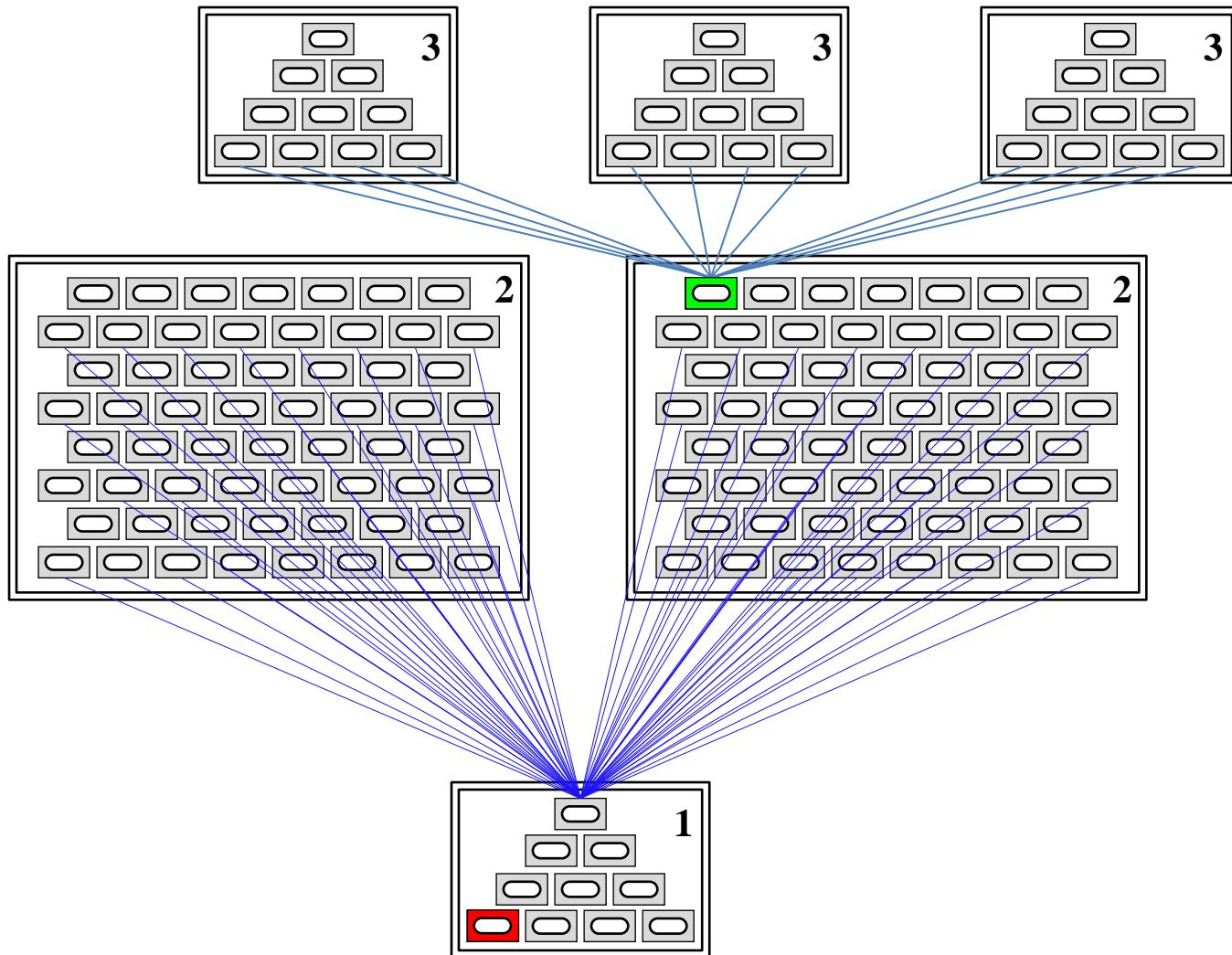
2. Survey of Advanced *Levelization* Techniques

Redundancy



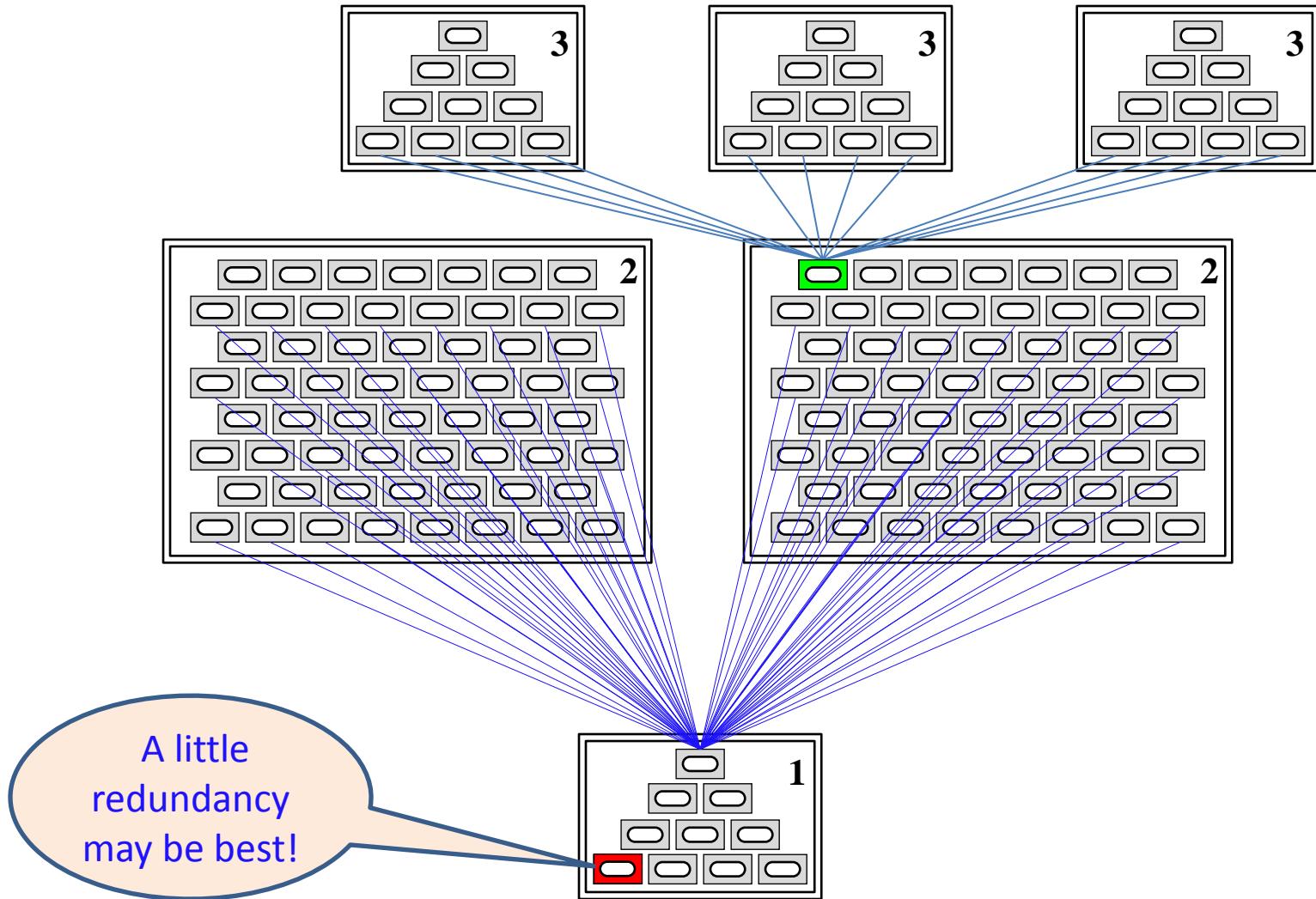
2. Survey of Advanced *Levelization* Techniques

Redundancy



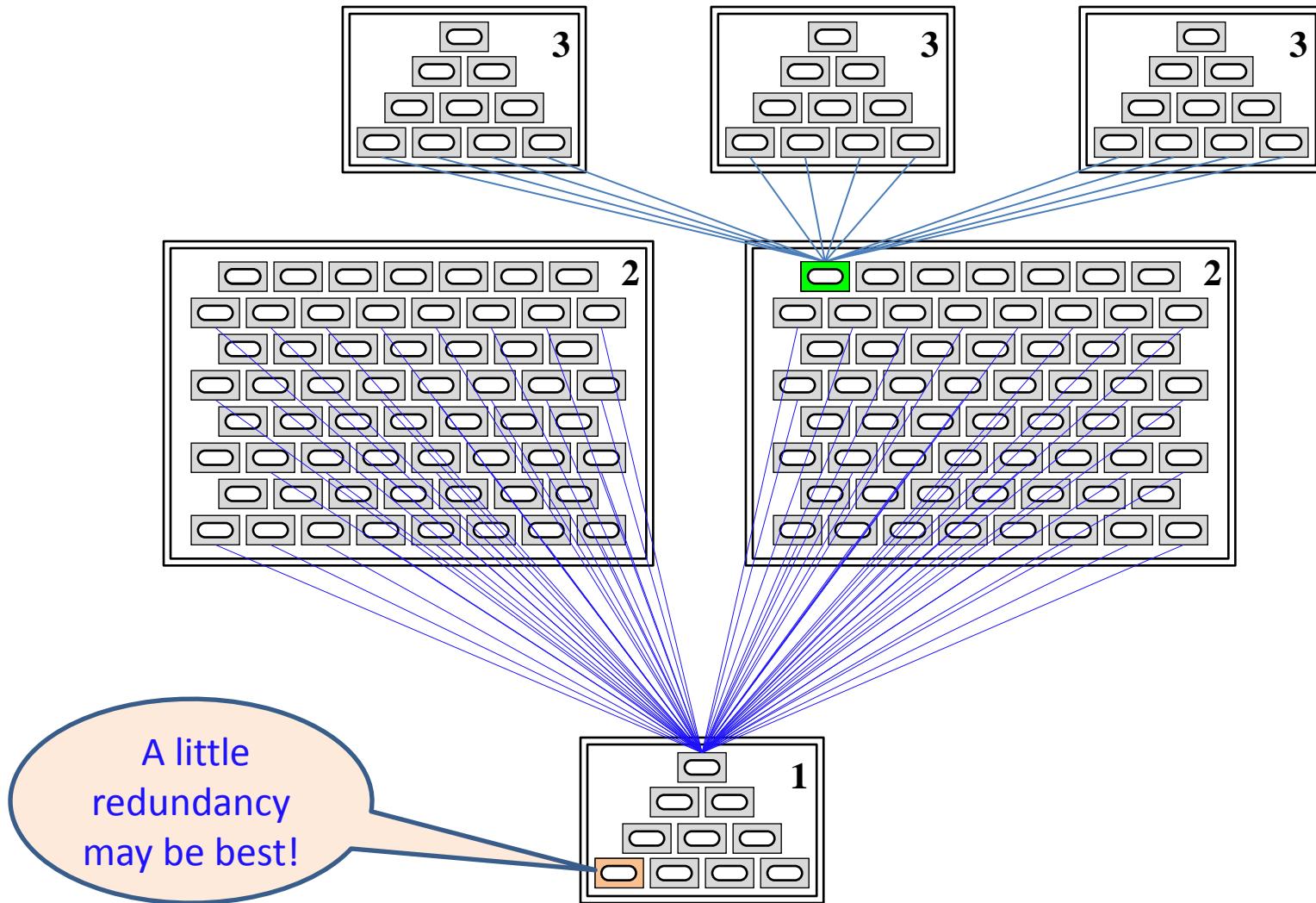
2. Survey of Advanced *Levelization* Techniques

Redundancy



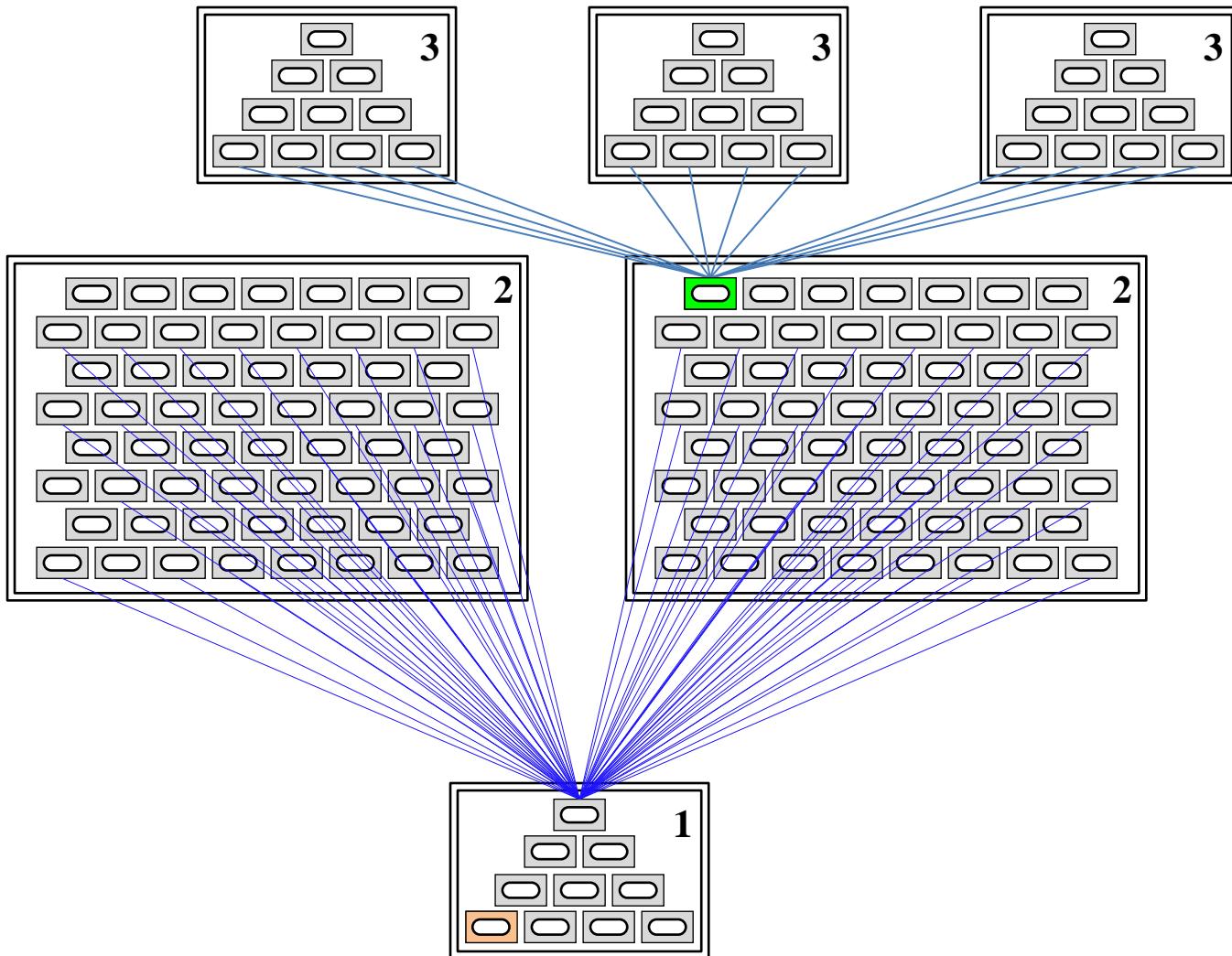
2. Survey of Advanced *Levelization* Techniques

Redundancy



2. Survey of Advanced *Levelization* Techniques

Redundancy



2. Survey of Advanced *Levelization* Techniques

Redundancy

Discussion?

2. Survey of Advanced *Levelization* Techniques

Callbacks

Callbacks – Client-supplied functions/data that enable lower-level subsystems to perform specific tasks in a more global context.

2. Survey of Advanced *Levelization* Techniques

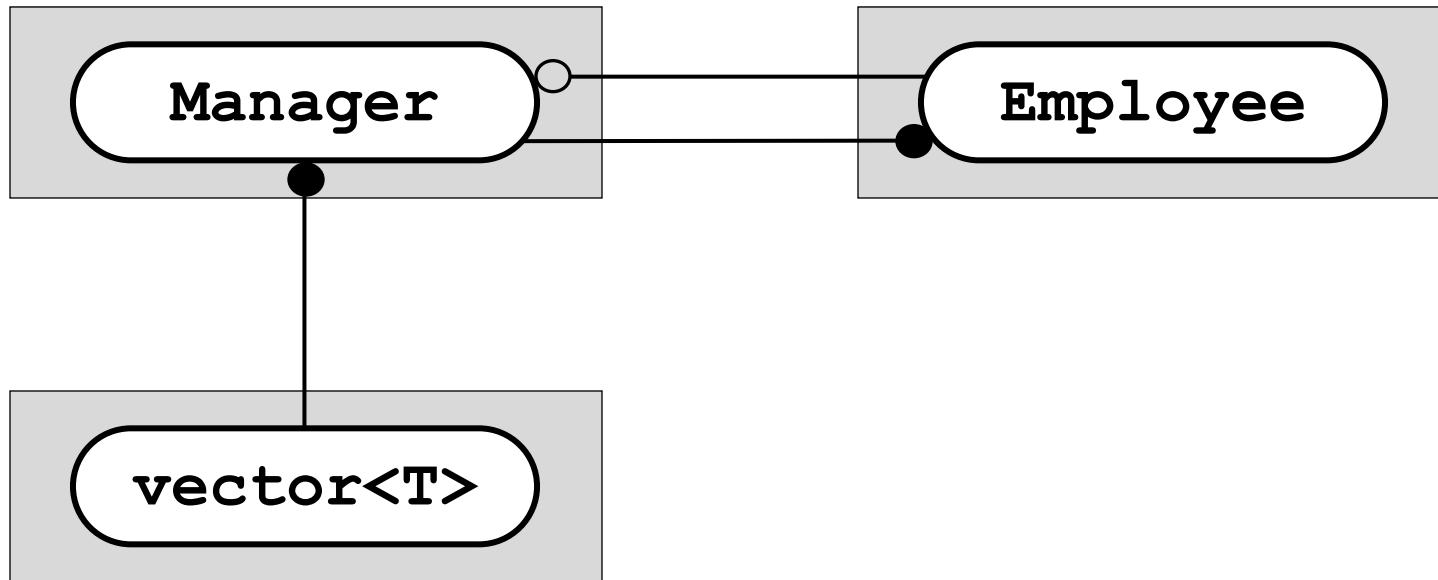
Callbacks

There are several flavors:

1. DATA (Effectively *Demotion*)
2. FUNCTION (Stateless/Stateful)
3. FUNCTOR (Function Object)
4. PROTOCOL (Abstract Interface)
5. CONCEPT (Structural Interface)

2. Survey of Advanced *Levelization* Techniques

Callbacks: DATA



2. Survey of Advanced *Levelization* Techniques

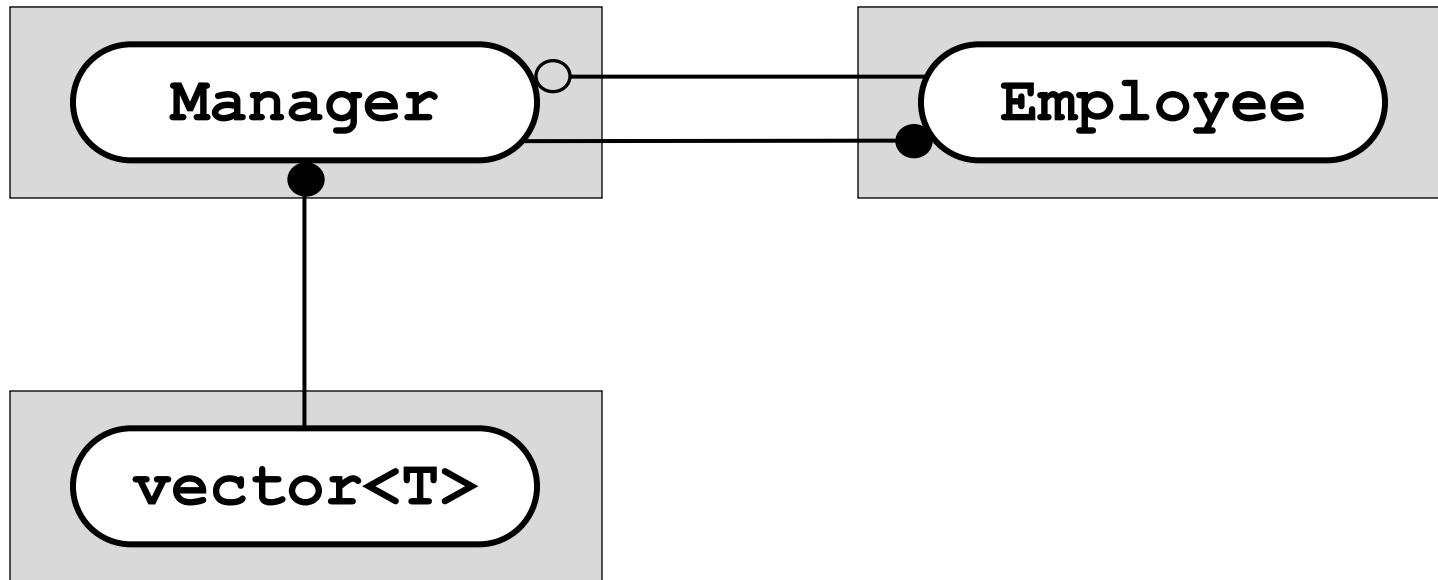
Callbacks: DATA

```
// manager.h
#include <employee.h>
#include <vector.h>
class Manager {
    vector<Employee> d_staff;
public:
    // ...
    int addEmployee(...);
    const Employee&
        employee(int id) const;
};
```

```
// employee.h
class Manager;
class Employee {
    Manager *d_boss_p;
public:
    Employee(Manager *boss, ...);
    // ...
    int numStaff() const;
};
```

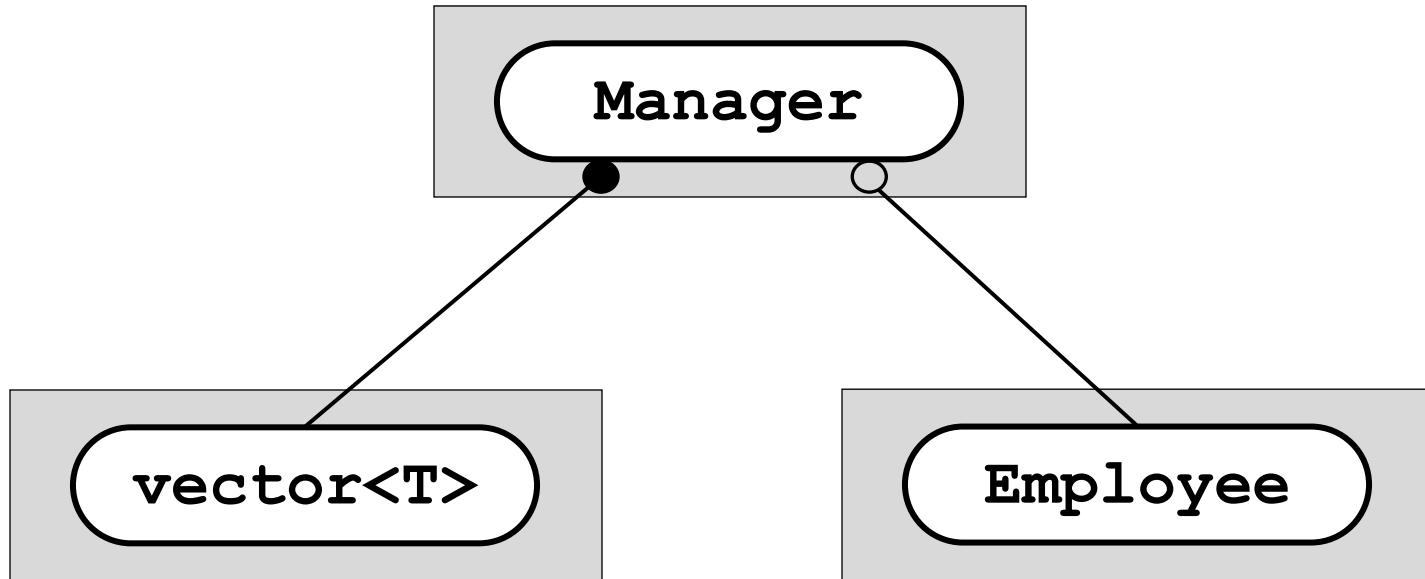
2. Survey of Advanced *Levelization* Techniques

Callbacks: DATA



2. Survey of Advanced *Levelization* Techniques

Callbacks: DATA



2. Survey of Advanced *Levelization* Techniques

Callbacks: DATA

```
// manager.h
#include <employee.h>
#include <vector.h>
class Manager {
    vector<Employee> d_staff;

public:
    // ...
    int addEmployee();
    const Employee&
        employee(int id) const;
};
```

```
// employee.h
class Manager;
class Employee {
    Manager *d_boss_p;
public:
    Employee(Manager *boss);
    // ...
    int numStaff() const;
};
```

2. Survey of Advanced *Levelization* Techniques

Callbacks: DATA

```
// manager.h
#include <employee.h>
#include <vector.h>
class Manager {
    vector<Employee> d_staff;

public:
    // ...
    int addEmployee();
    const Employee&
        employee(int id) const;
};
```

```
// employee.h
class Manager;
class Employee {
    Manager *d_boss_p;
public:
    Employee(Manager *boss);
    // ...
    int numStaff() const;
};
```

2. Survey of Advanced *Levelization* Techniques

Callbacks: DATA

```
// manager.h
#include <employee.h>
#include <vector.h>
class Manager {
    vector<Employee> d_staff;

public:
    // ...
    int addEmployee();
    const Employee&
        employee(int id) const;
};
```

```
// employee.h
class Employee {
    int *const d_numStaff_p;
public:
    Employee(int *nStaffAddr);
    // ...
    int numStaff() const;
};
```

2. Survey of Advanced *Levelization* Techniques

Callbacks: DATA

```
// manager.h
#include <employee.h>
#include <vector.h>
class Manager {
    vector<Employee> d_staff;

public:
    // ...
    int addEmployee();
    const Employee&
        employee(int id) const;
};
```

```
// employee.h

class Employee {
    int *const d_numStaff_p;
public:
    Employee(int *nStaffAddr);
    // ...
    int numStaff() const;
};

inline int
Employee::numStaff() const {
    return *d_numStaff_p;
}
```

2. Survey of Advanced *Levelization* Techniques

Callbacks: DATA

```
// manager.h
#include <employee.h>
#include <vector.h>
class Manager {
    vector<Employee> d_staff;
    int d_nStaff; // init to 0
public:
    // ...
    int addEmployee();
    const Employee&
        employee(int id) const;
};
```

```
// employee.h

class Employee {
    int *const d_numStaff_p;
public:
    Employee(int *nStaffAddr);
    // ...
    int numStaff() const;
}

inline int
Employee::numStaff() const {
    return *d_numStaff_p;
}
```

2. Survey of Advanced *Levelization* Techniques

Callbacks: DATA

```
// manager.h
#include <employee.h>
#include <vector.h>
class Manager {
    vector<Employee> d_staff;
    int d_nStaff; // init to 0
public:
    // ...
    int addEmployee();
    const Employee&
        employee(int id) const;
};

inline int
Manager::addEmployee() {
    d_staff.push_back(&d_nStaff);
    return d_nStaff++;
}
```

```
// employee.h

class Employee {
    int *const d_numStaff_p;
public:
    Employee(int *nStaffAddr);
    // ...
    int numStaff() const;
};

inline int
Employee::numStaff() const {
    return *d_numStaff_p;
}
```

2. Survey of Advanced *Levelization* Techniques

Callbacks: DATA

```
// manager.h
#include <employee.h>
#include <vector.h>
class Manager {
    vector<Employee> d_staff;
    int d_nStaff; // init to 0
public:
    // ...
    int addEmployee();
    const Employee&
        employee(int id) const;
};
inline int
Manager::addEmployee() {
    d_staff.push_back(*d_nStaff);
    return d_nStaff;
}
```

Effectively

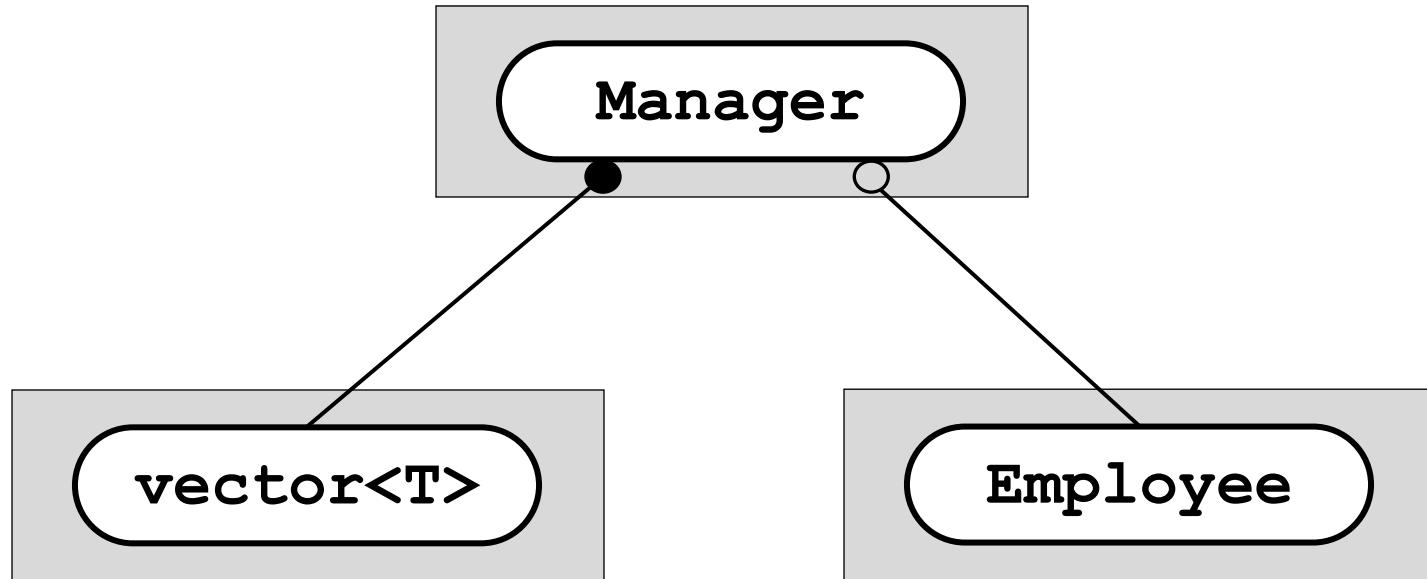
```
// employee.h
class Employee {
    int *const d_numStaff_p;
public:
    Employee(int *nStaffAddr);
    // ...
    int numStaff() const;
};

inline int
Employee::numStaff() const {
    return *d_numStaff_p;
}
```

Demotion!

2. Survey of Advanced *Levelization* Techniques

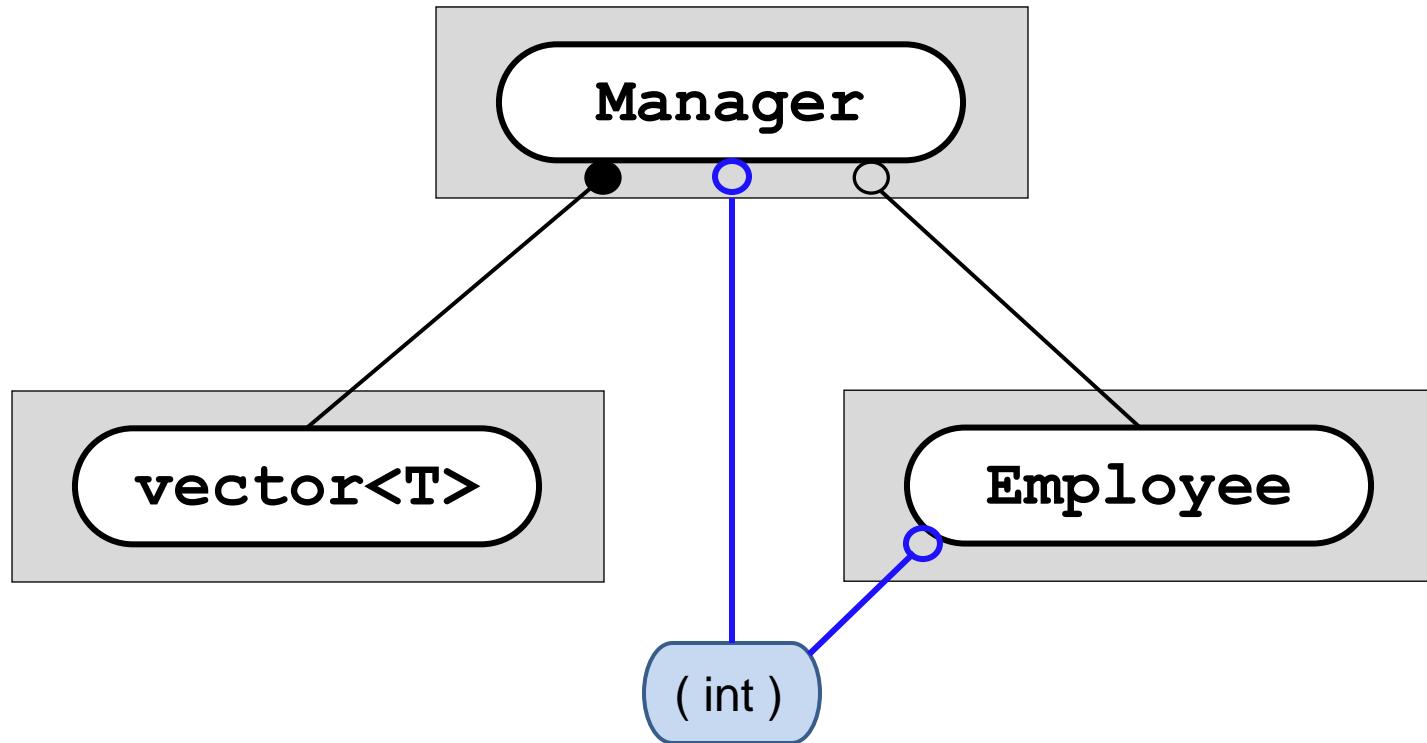
Callbacks: DATA



Effectively *Demotion!*

2. Survey of Advanced *Levelization* Techniques

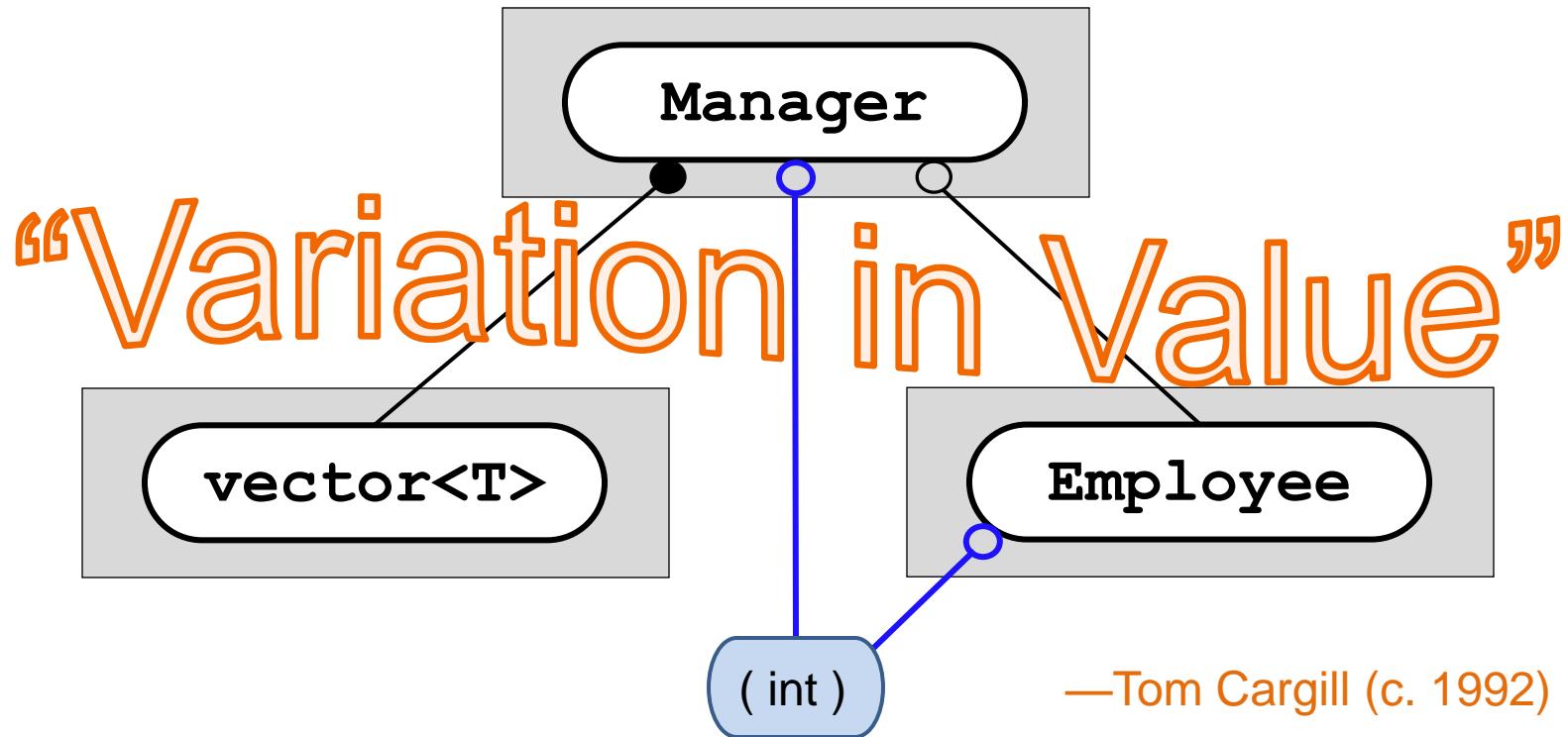
Callbacks: DATA



Effectively *Demotion!*

2. Survey of Advanced *Levelization* Techniques

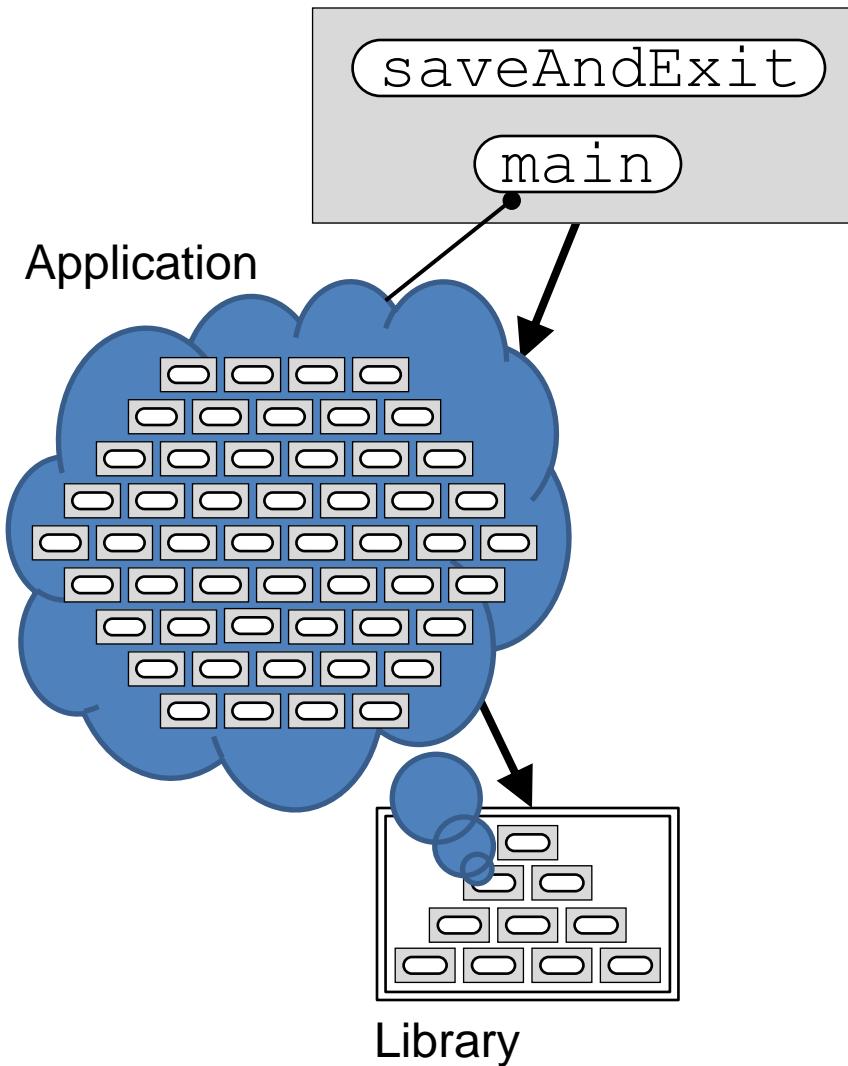
Callbacks: DATA



Effectively *Demotion!*

2. Survey of Advanced *Levelization* Techniques

Callbacks: Function



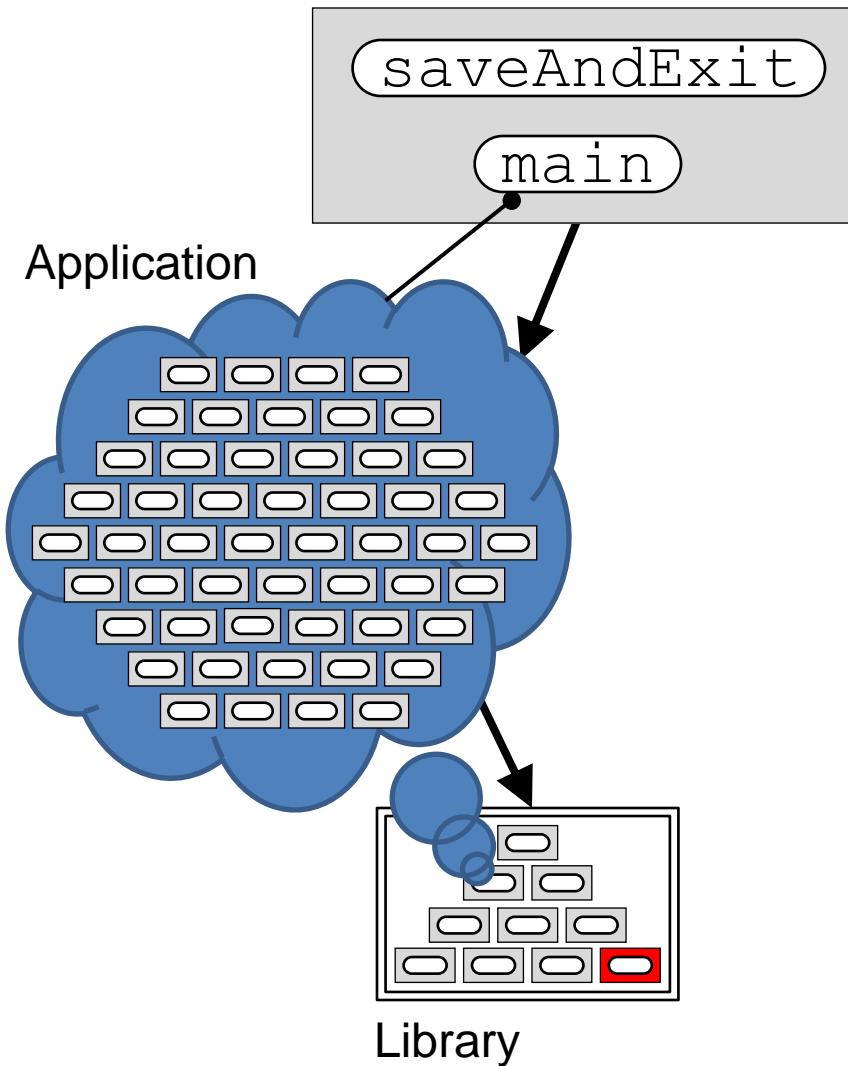
```
// main.cpp
// ...
static MyState globalState;
void saveAndExit()
{
    // First save work.
    // ...
    std::exit(1);
}

int main()
{
    // Do useful work.
    // ...

    return 0;
}
```

2. Survey of Advanced *Levelization* Techniques

Callbacks: Function



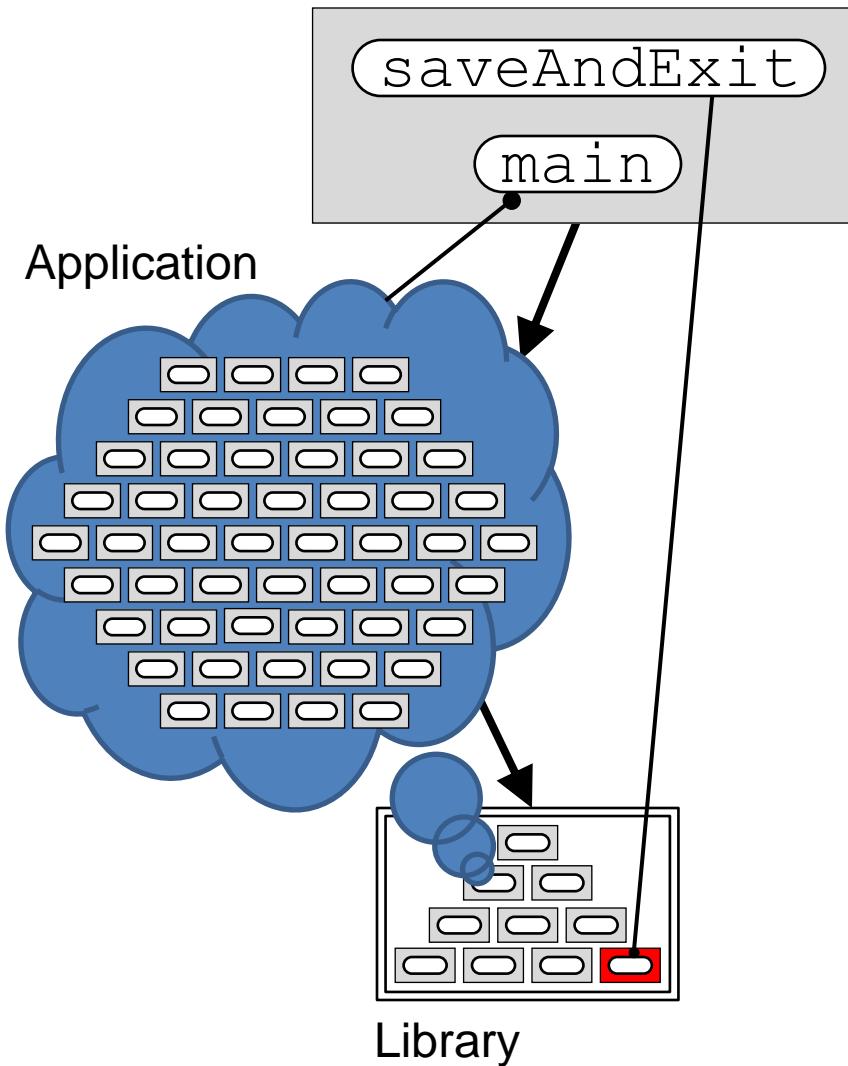
```
// main.cpp
// ...
static MyState globalState;
void saveAndExit()
{
    // First save work.
    // ...
    std::exit(1);
}

int main()
{
    // Do useful work.
    // ...

    return 0;
}
```

2. Survey of Advanced *Levelization* Techniques

Callbacks: Function



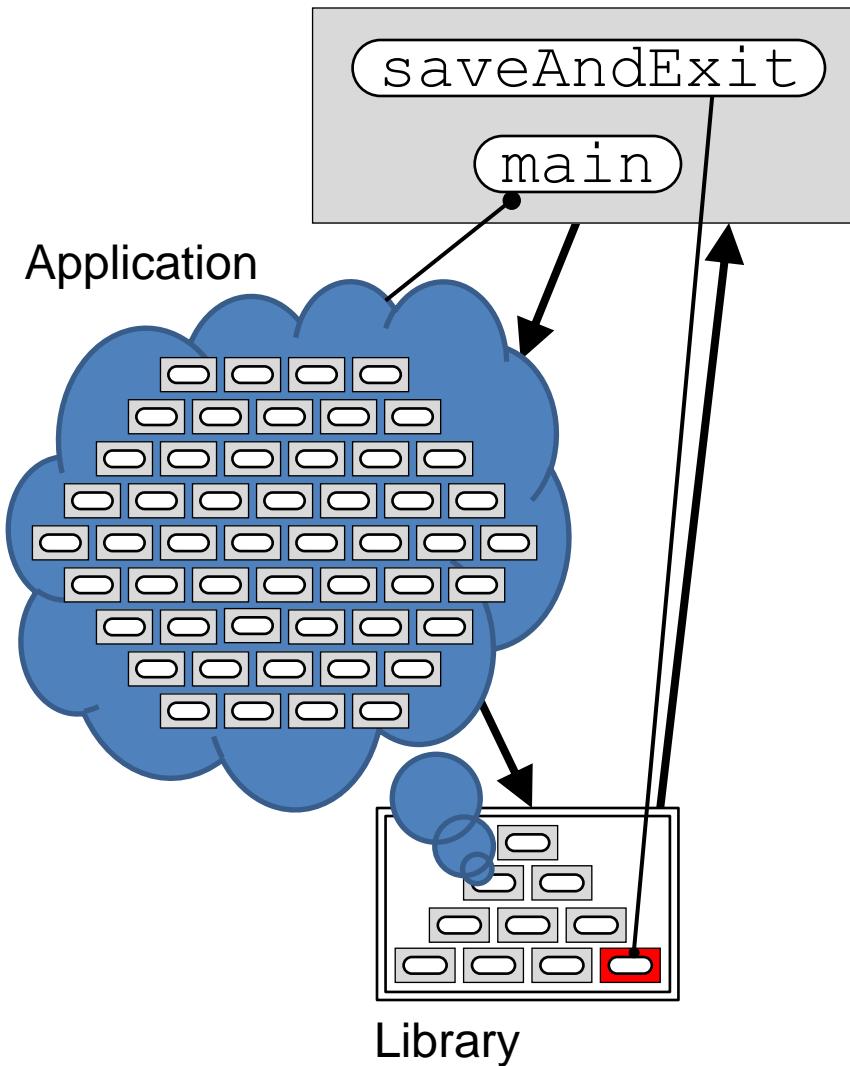
```
// main.cpp
// ...
static MyState globalState;
void saveAndExit()
{
    // First save work.
    // ...
    std::exit(1);
}

int main()
{
    // Do useful work.
    // ...

    return 0;
}
```

2. Survey of Advanced *Levelization* Techniques

Callbacks: Function



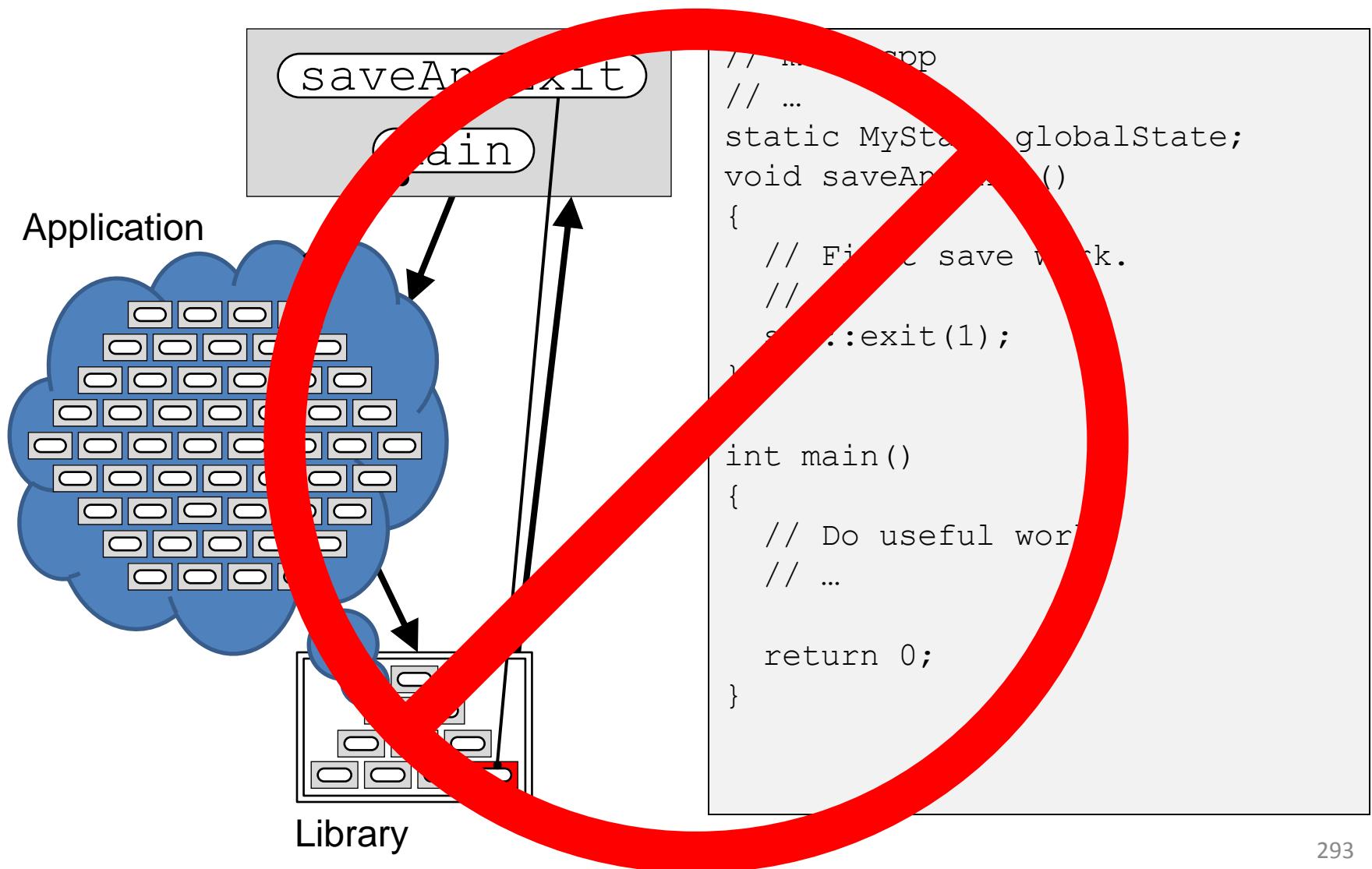
```
// main.cpp
// ...
static MyState globalState;
void saveAndExit()
{
    // First save work.
    // ...
    std::exit(1);
}

int main()
{
    // Do useful work.
    // ...

    return 0;
}
```

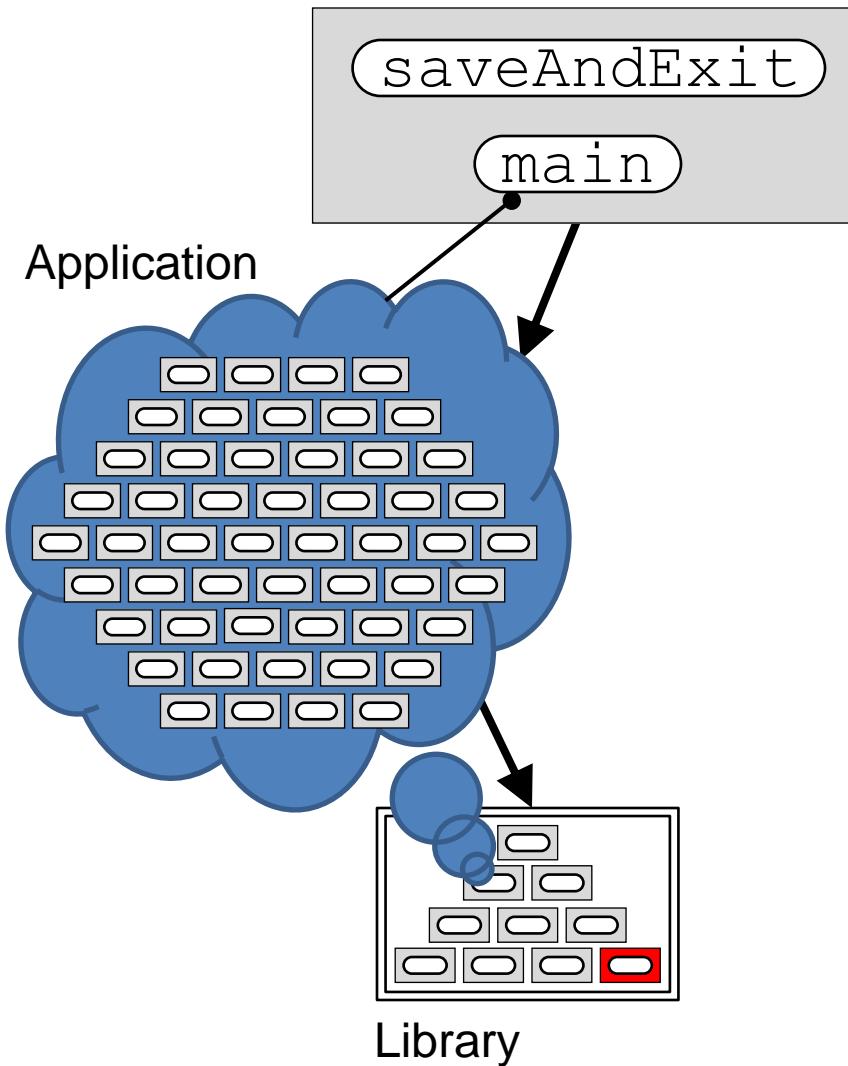
2. Survey of Advanced *Levelization* Techniques

Callbacks: Function



2. Survey of Advanced *Levelization* Techniques

Callbacks: Function



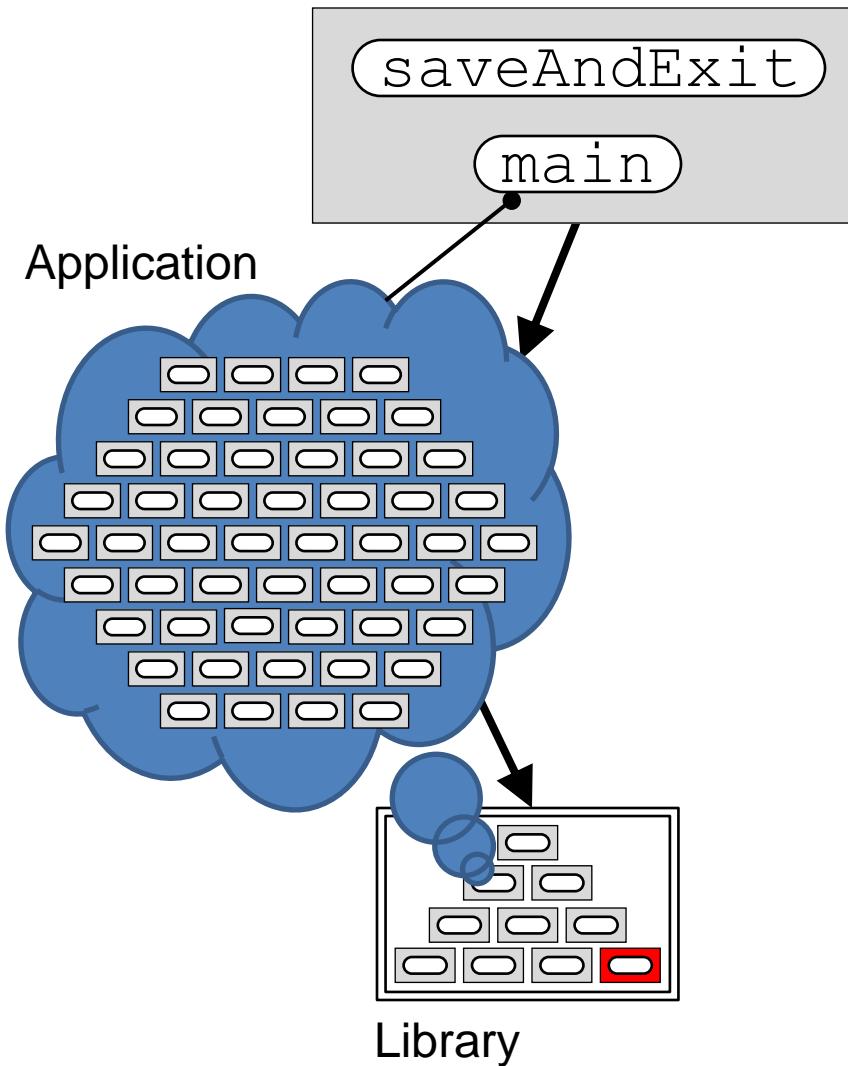
```
// main.cpp
// ...
static MyState globalState;
void saveAndExit()
{
    // First save work.
    // ...
    std::exit(1);
}

int main()
{
    // Do useful work.
    // ...

    return 0;
}
```

2. Survey of Advanced *Levelization* Techniques

Callbacks: Function



```
// main.cpp
// ...
static MyState globalState;
void saveAndExit()
{
    // First save work.
    // ...
    std::exit(1);
}

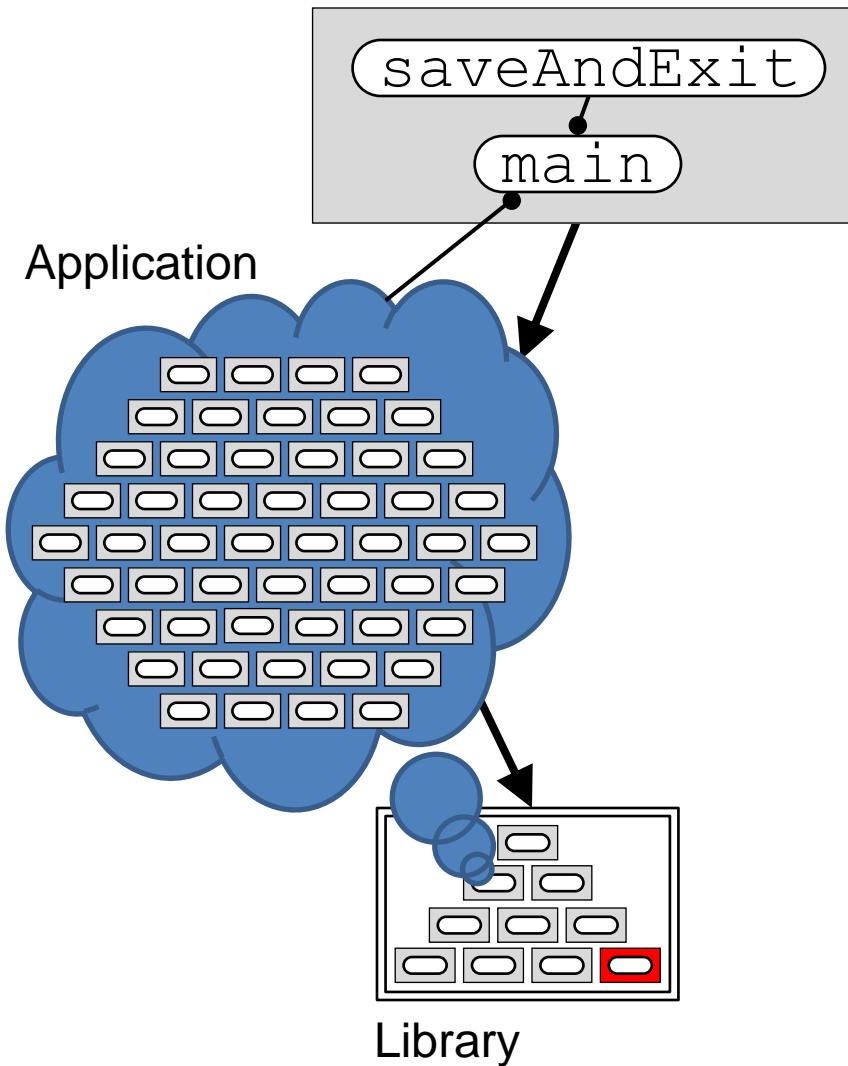
int main()
{
    set_new_handler(saveAndExit);

    // Now do useful work.
    // ...

    return 0;
}
```

2. Survey of Advanced *Levelization* Techniques

Callbacks: Function



```
// main.cpp
// ...
static MyState globalState;
void saveAndExit()
{
    // First save work.
    // ...
    std::exit(1);
}

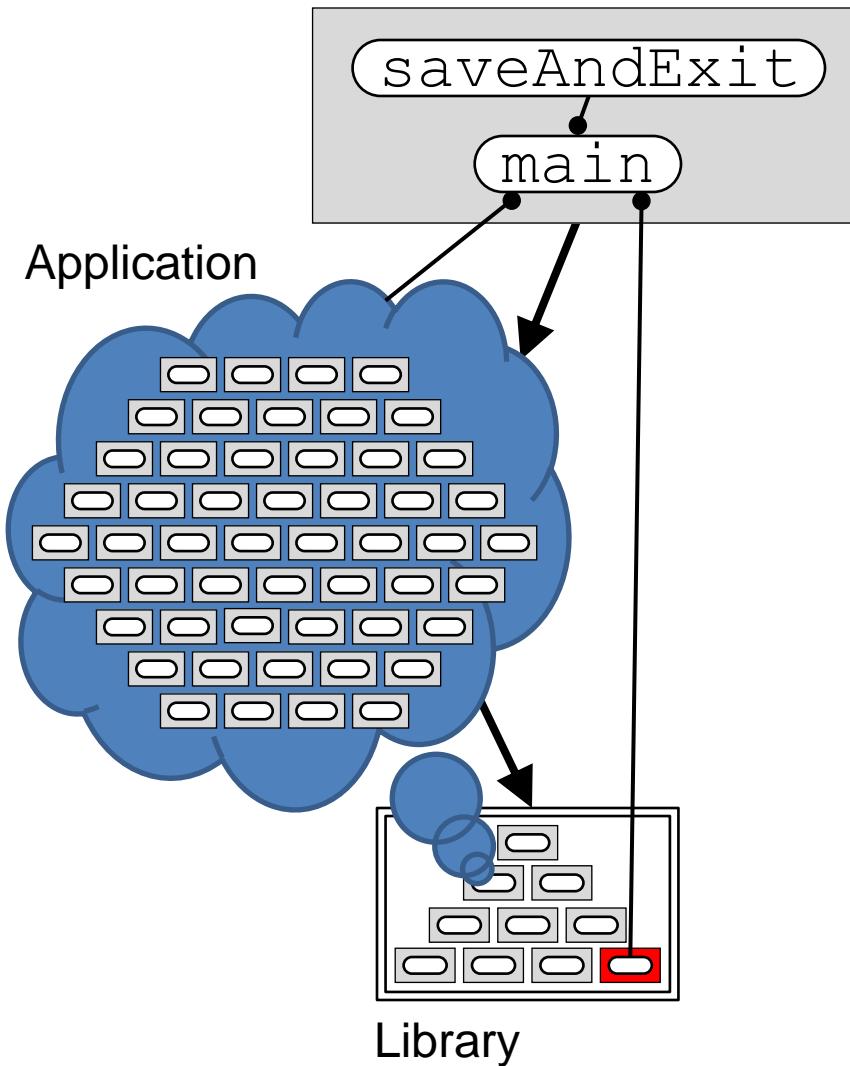
int main()
{
    set_new_handler(saveAndExit);

    // Now do useful work.
    // ...

    return 0;
}
```

2. Survey of Advanced *Levelization* Techniques

Callbacks: Function



```
// main.cpp
// ...
static MyState globalState;
void saveAndExit()
{
    // First save work.
    // ...
    std::exit(1);
}

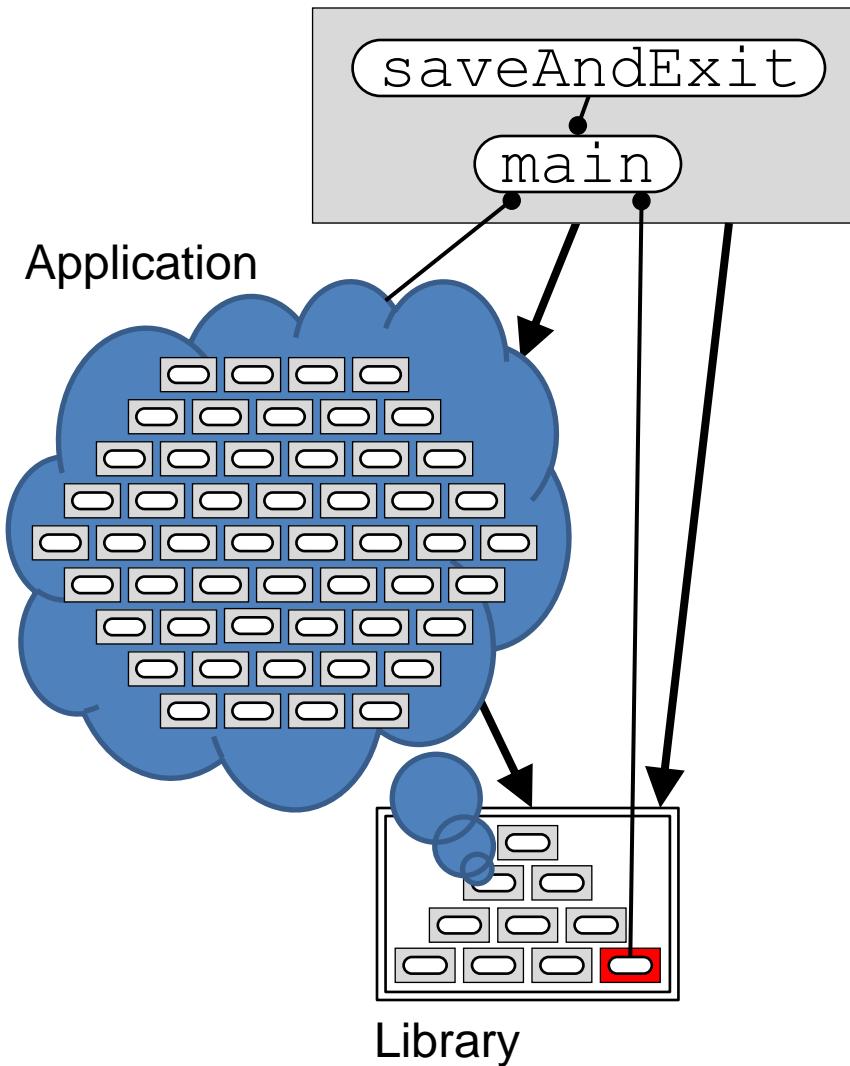
int main()
{
    set_new_handler(saveAndExit);

    // Now do useful work.
    // ...

    return 0;
}
```

2. Survey of Advanced *Levelization* Techniques

Callbacks: Function



```
// main.cpp
// ...
static MyState globalState;
void saveAndExit()
{
    // First save work.
    // ...
    std::exit(1);
}

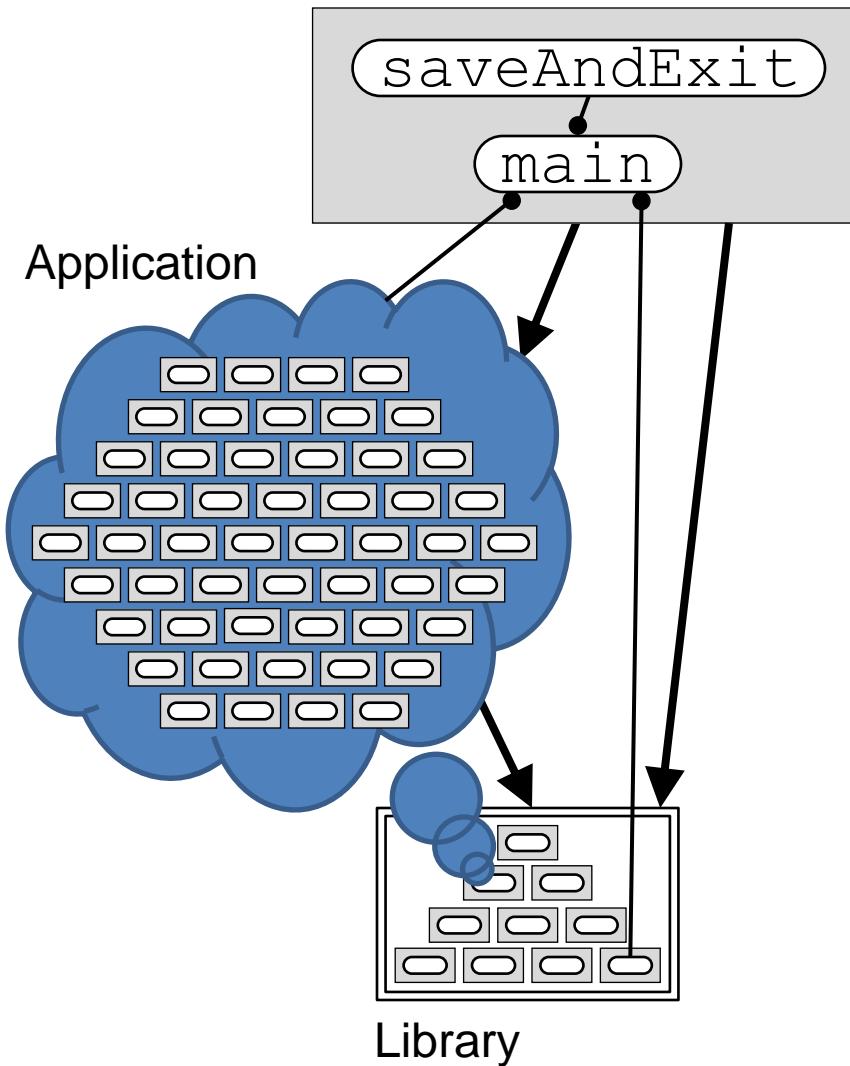
int main()
{
    set_new_handler(saveAndExit);

    // Now do useful work.
    // ...

    return 0;
}
```

2. Survey of Advanced *Levelization* Techniques

Callbacks: Function



```
// main.cpp
// ...
static MyState globalState;
void saveAndExit()
{
    // First save work.
    // ...
    std::exit(1);
}

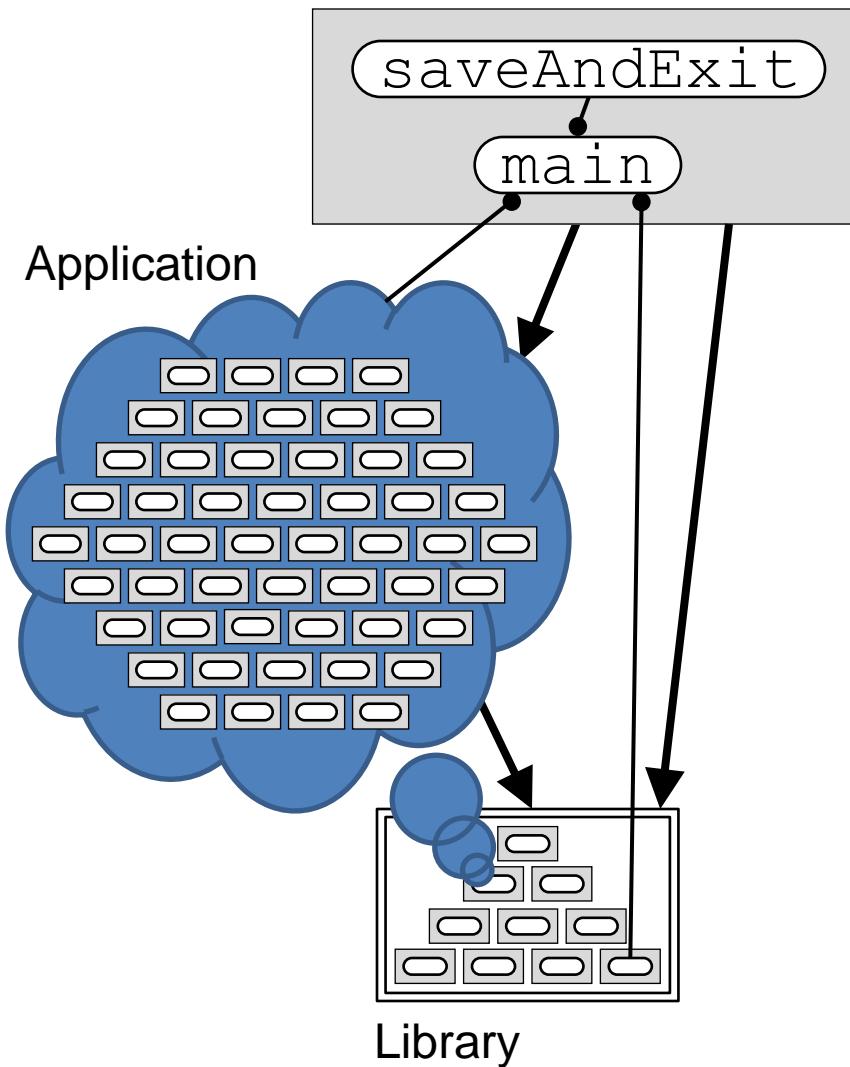
int main()
{
    set_new_handler(saveAndExit);

    // Now do useful work.
    // ...

    return 0;
}
```

2. Survey of Advanced *Levelization* Techniques

Callbacks: Function



```
// main.cpp
// ...
static MyState globalState;
void saveAndExit()
{
    // First save work.
    // ...
    std::exit(1);
}

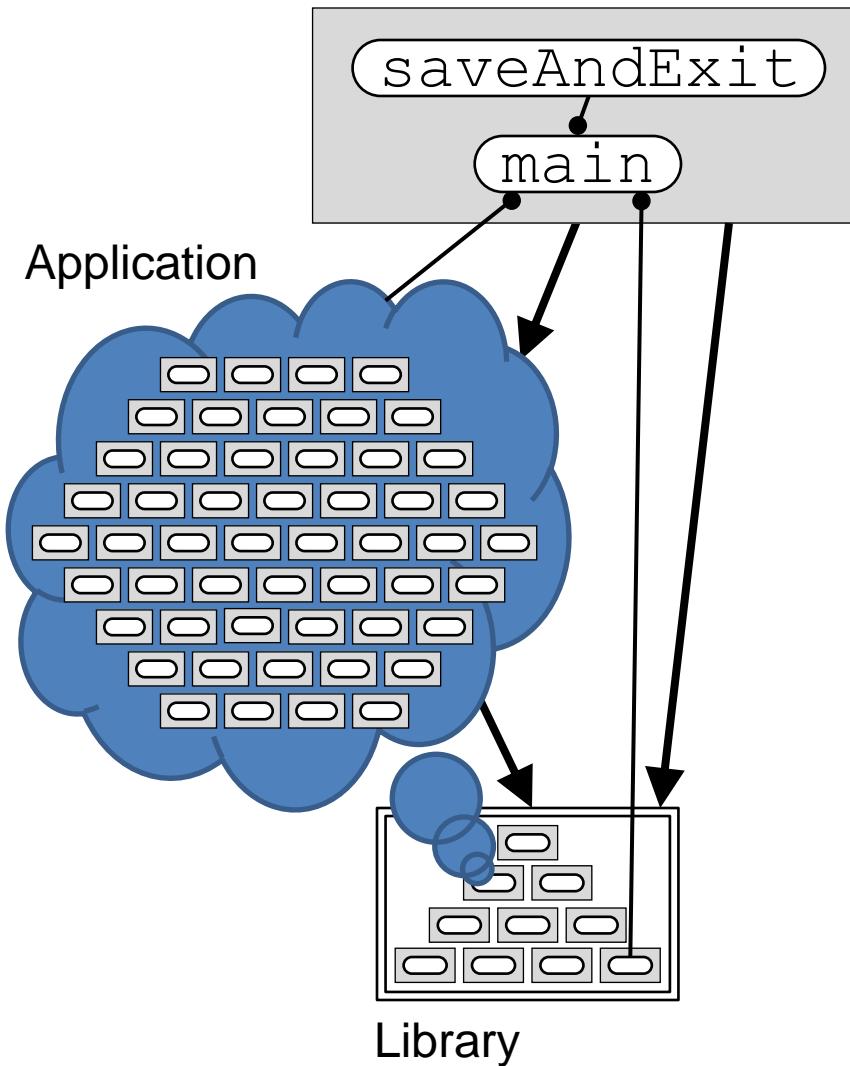
int main()
{
    set_new_handler(saveAndExit);

    // Now do useful work.
    // ...

    return 0;
}
```

2. Survey of Advanced *Levelization* Techniques

Callbacks: Function



```
// main.cpp
// ...
static MyState globalState;
static void saveAndExit()
{
    // First save work.
    // ...
    std::exit(1);
}

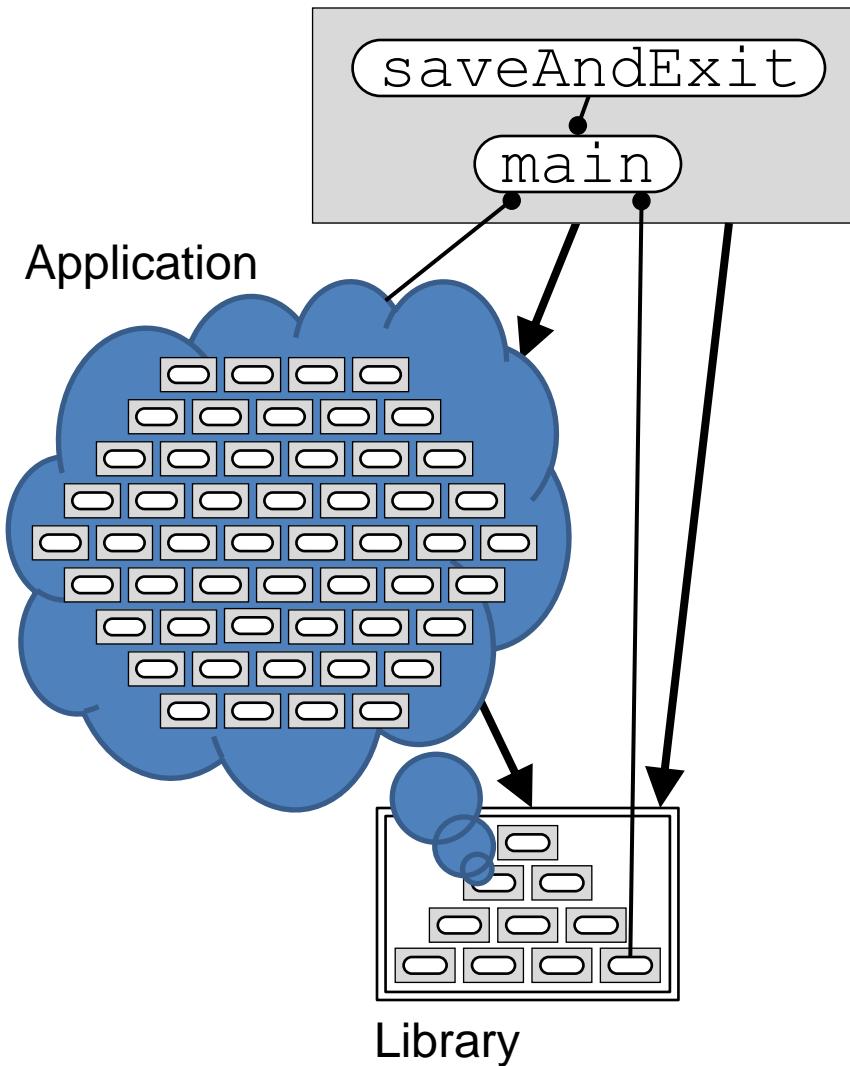
int main()
{
    set_new_handler(saveAndExit);

    // Now do useful work.
    // ...

    return 0;
}
```

2. Survey of Advanced *Levelization* Techniques

Callbacks: Function



```
// main.cpp
// ...
static MyState globalState;
static void saveAndExit()
{
    // First save work.
    // ...
    std::exit(1);
}

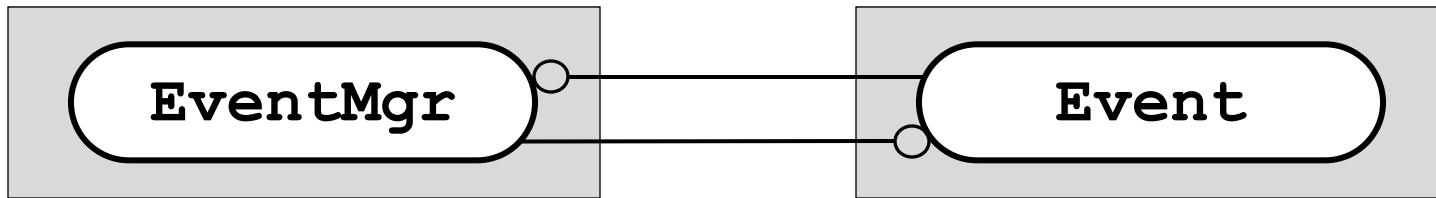
int main()
{
    set_new_handler(saveAndExit);

    // Now do useful work.
    // ...

    return 0;
}
```

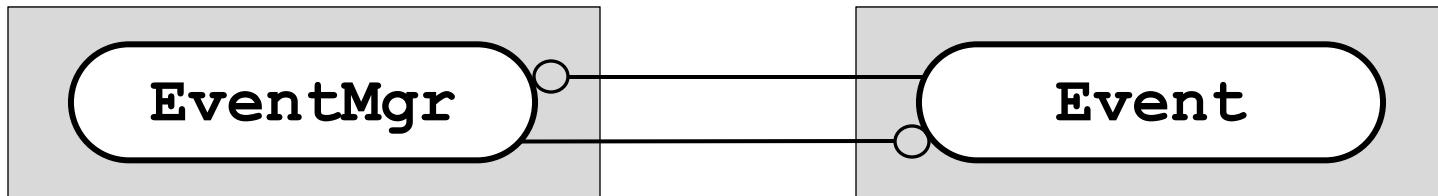
2. Survey of Advanced *Levelization* Techniques

Callbacks: Function



2. Survey of Advanced *Levelization* Techniques

Callbacks: Function



```
// eventmgr.h
#include <event.h>
class EventMgr {
    // ...
public:
    // ...
    void schedule(Event *event);
};

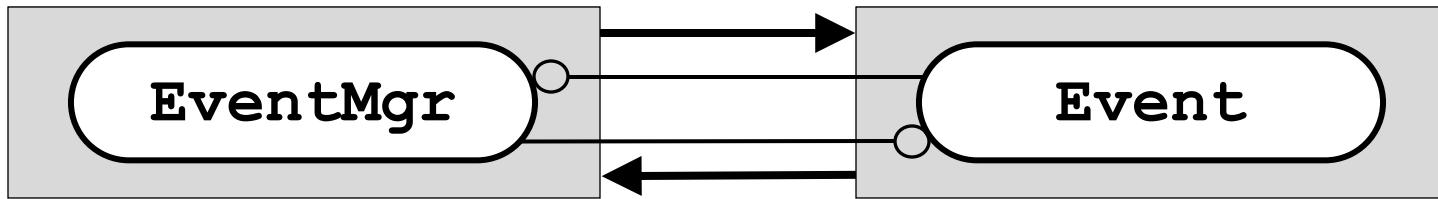
inline void EventMgr::invoke()
{
    d_head->invoke();
    // ...
}
```

```
// event.h
#include <eventmgr.h>
class Event {
    // ...
public:
    // ...
    void schedule(EventMgr *eMgr);
    void invoke();
};

inline
void schedule(EventMgr *eMgr)
{
    eMgr->schedule(this);
}
```

2. Survey of Advanced *Levelization* Techniques

Callbacks: Function



```
// eventmgr.h
#include <event.h>
class EventMgr {
    // ...
public:
    // ...
    void schedule(Event *event);
};

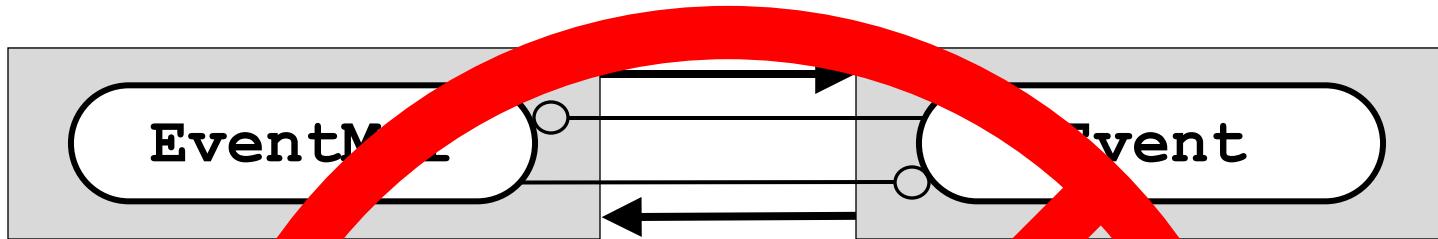
inline void EventMgr::invoke()
{
    d_head->invoke();
    // ...
}
```

```
// event.h
#include <eventmgr.h>
class Event {
    // ...
public:
    // ...
    void schedule(EventMgr *eMgr);
    void invoke();
};

inline
void schedule(EventMgr *eMgr)
{
    eMgr->schedule(this);
}
```

2. Survey of Advanced *Levelization* Techniques

Callbacks: Function



```
// eventmgr.h
#include <event.h>
class EventMgr {
    // ...
public:
    // ...
    void schedule(Event *event);
};

inline void EventMgr::invoke()
{
    d_head->invoke
    // ...
}
```

```
// event.h
#include <eventmgr.h>
class Event {
    // ...
public:
    // ...
    void schedule(EventMgr *eMgr);
    void invoke();
};

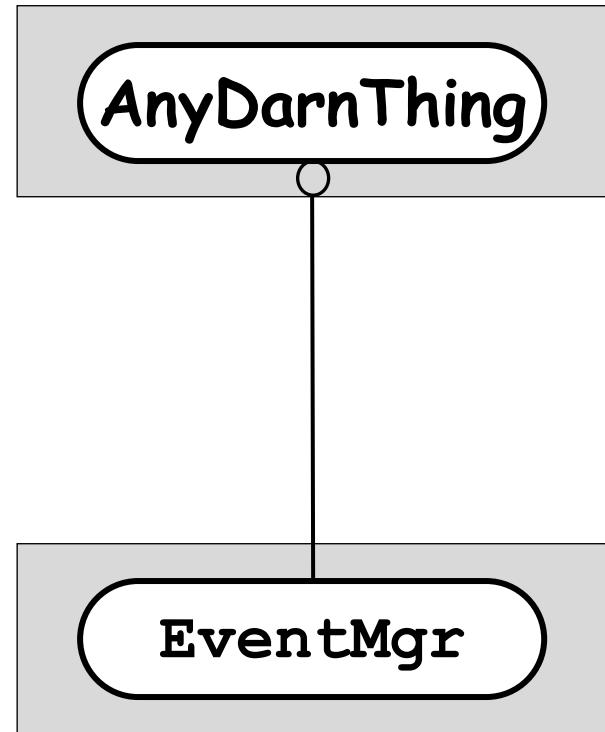
inline
void schedule(EventMgr *eMgr)
{
    eMgr->schedule(this);
}
```

2. Survey of Advanced *Levelization* Techniques

Callbacks: Function

```
// anydarnthing.h
#include <eventmgr.h>
class AnyDarnThing {
    static void invoke(void *me);
public:
    // ...
    void schedule(EventMgr *eMgr);
}
```

```
// eventmgr.h
// ...
class EventMgr {
    // ...
public:
    typedef void (*EventCb)(void *);
    // ...
    void schedule(EventCb cb
                  void     *data);
};
```



2. Survey of Advanced *Levelization* Techniques

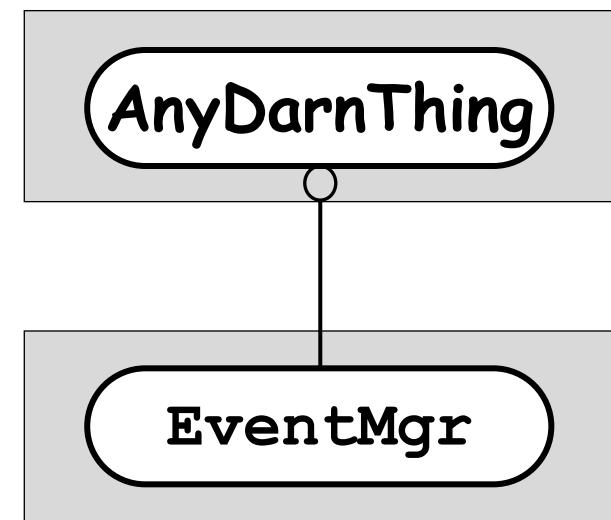
Callbacks: Function

```
// anydarnthing.h
#include <eventmgr.h>
class AnyDarnThing {
    static void invoke(void *me);
public:
    // ...
    void schedule(EventMgr *eMgr);
}
```

```
// anydarnthing.cpp
// ...

void AnyDarnThing::schedule(
    EventMgr *eMgr) {
    eMgr->schedule(invoker, this);
}
```

```
// eventmgr.h
// ...
class EventMgr {
    // ...
public:
    typedef void (*EventCb)(void *);
    // ...
    void schedule(EventCb cb
                  void     *data);
};
```



2. Survey of Advanced *Levelization* Techniques

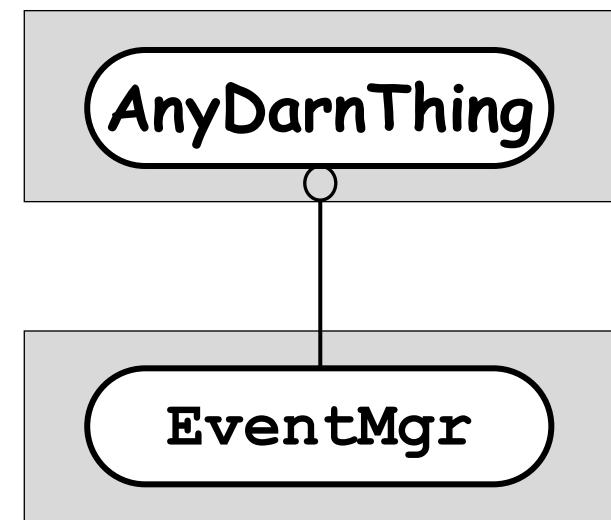
Callbacks: Function

```
// anydarnthing.h
#include <eventmgr.h>
class AnyDarnThing {
    static void invoke(void *me);
public:
    // ...
    void schedule(EventMgr *eMgr);
}
```

```
// anydarnthing.cpp
// ...

void AnyDarnThing::schedule(
    EventMgr *eMgr) {
    eMgr->schedule(invoker, this);
}
```

```
// eventmgr.h
// ...
class EventMgr {
    // ...
public:
    typedef void (*EventCb)(void *);
    // ...
    void schedule(EventCb cb
                  void     *data);
};
```



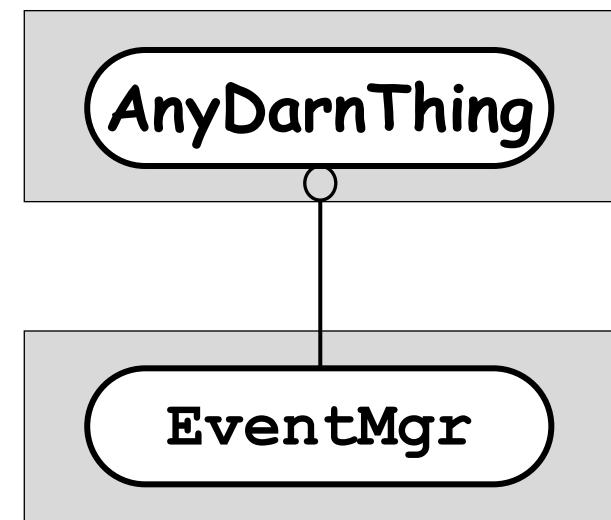
2. Survey of Advanced *Levelization* Techniques

Callbacks: Function

```
// anydarnthing.h
class EventMgr;
class AnyDarnThing {
    static void invoke(void *me);
public:
    // ...
    void schedule(EventMgr *eMgr);
}
```

```
// anydarnthing.cpp
#include <eventmgr.h>
// ...
void AnyDarnThing::schedule(
    EventMgr *eMgr) {
    eMgr->schedule(invoker, this);
}
```

```
// eventmgr.h
// ...
class EventMgr {
    // ...
public:
    typedef void (*EventCb)(void *);
    // ...
    void schedule(EventCb cb
                  void     *data);
};
```



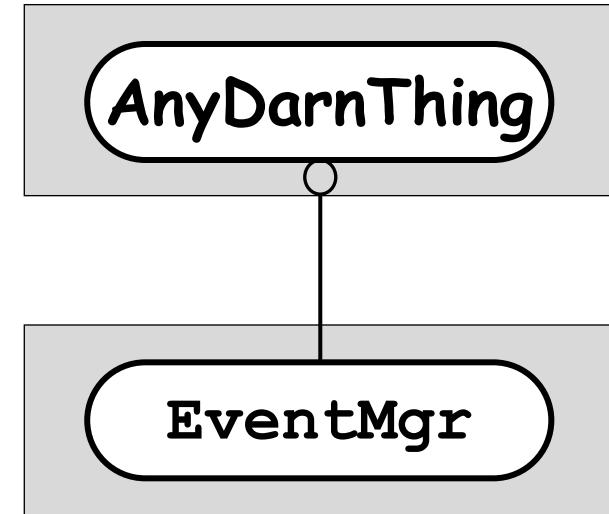
2. Survey of Advanced *Levelization* Techniques

Callbacks: Function

```
// anydarnthing.h
class EventMgr;
class AnyDarnThing {
    static void invoke(void *me);
public:
    // ...
    void schedule(EventMgr *eMgr);
}
```

```
// anydarnthing.cpp
#include <eventmgr.h>
// ...
void AnyDarnThing::schedule(
    EventMgr *eMgr) {
    eMgr->schedule(invoker, this);
}
```

```
// eventmgr.h
// ...
class EventMgr {
    // ...
public:
    typedef void (*EventCb)(void *);
    // ...
    void schedule(EventCb cb
                  void     *data);
};
```



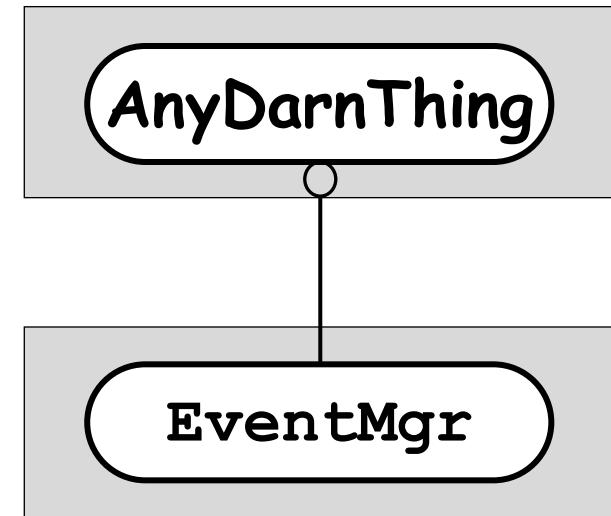
2. Survey of Advanced *Levelization* Techniques

Callbacks: Functor

```
// anydarnthing.h
class EventMgr;
class AnyDarnThing {
    static void invoke(void *me);
public:
    // ...
    void schedule(EventMgr *eMgr);
}
```

```
// anydarnthing.cpp
#include <eventmgr.h>
// ...
void AnyDarnThing::schedule(
    EventMgr *eMgr) {
    eMgr->schedule(invoker, this);
}
```

```
// eventmgr.h
// ...
class EventMgr {
    // ...
public:
    typedef void (*EventCb)(void *);
    // ...
    void schedule(EventCb cb
                  void     *data);
};
```



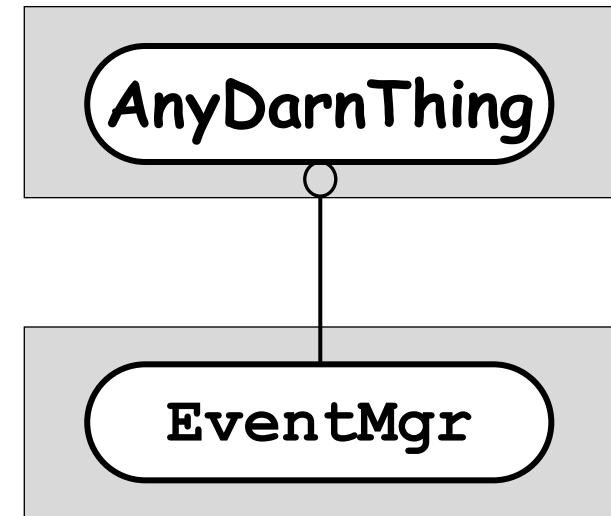
2. Survey of Advanced *Levelization* Techniques

Callbacks: Functor

```
// anydarnthing.h
class EventMgr;
class AnyDarnThing {
    static void invoke(void *me);
public:
    // ...
    void schedule(EventMgr *eMgr);
}
```

```
// anydarnthing.cpp
#include <eventmgr.h>
// ...
void AnyDarnThing::schedule(
    EventMgr *eMgr) {
    eMgr->schedule(invoker, this);
}
```

```
// eventmgr.h
// ...
class EventMgr {
    // ...
public:
    typedef void (*EventCb)(void *);
    // ...
    void schedule(EventCb cb
                  void     *data);
};
```



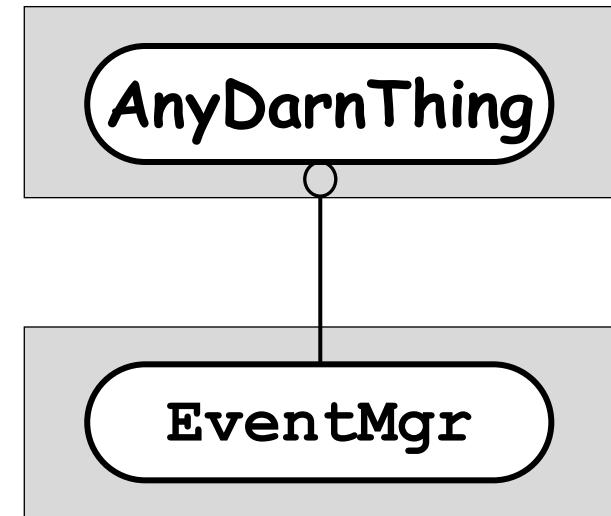
2. Survey of Advanced *Levelization* Techniques

Callbacks: Functor

```
// anydarnthing.h
class EventMgr;
class AnyDarnThing {
    void invoke();
public:
    // ...
    void schedule(EventMgr *eMgr);
}
```

```
// anydarnthing.cpp
#include <eventmgr.h>
// ...
void AnyDarnThing::schedule(
    EventMgr *eMgr) {
    eMgr->schedule(invoker, this);
}
```

```
// eventmgr.h
// ...
class EventMgr {
    // ...
public:
    typedef void (*EventCb)(void *);
    // ...
    void schedule(EventCb cb
                  void     *data);
};
```



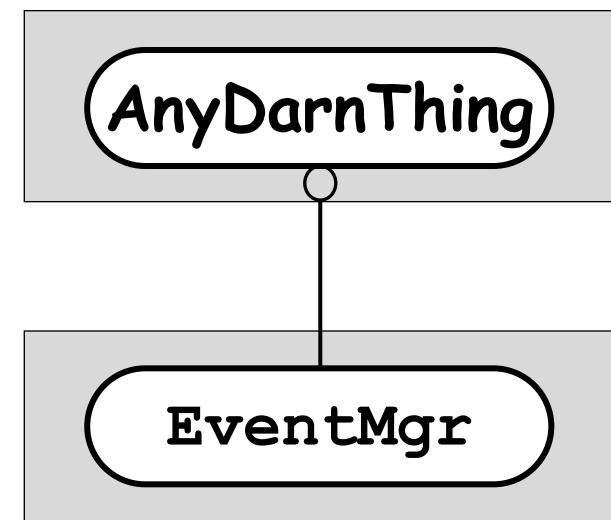
2. Survey of Advanced *Levelization* Techniques

Callbacks: Functor

```
// anydarnthing.h
class EventMgr;
class AnyDarnThing {
    void invoke();
public:
    // ...
    void schedule(EventMgr *eMgr);
}
```

```
// anydarnthing.cpp
#include <eventmgr.h>
// ...
void AnyDarnThing::schedule(
    EventMgr *eMgr) {
    eMgr->schedule(invoker, this);
}
```

```
// eventmgr.h
// ...
class EventMgr {
    // ...
public:
    typedef void (*EventCb)(void *);
    // ...
    void schedule(EventCb cb
                  void     *data);
};
```



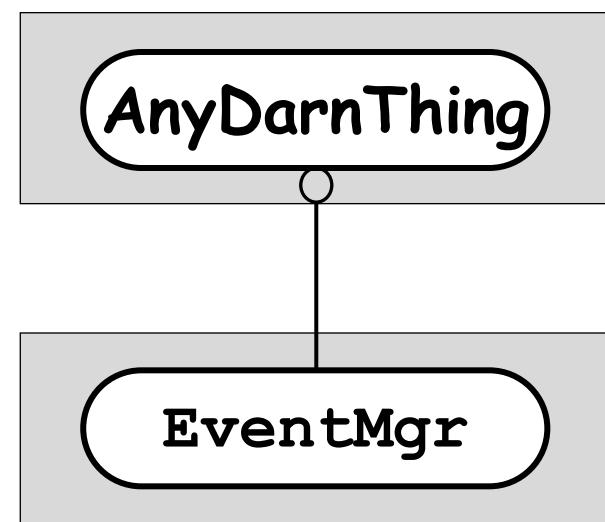
2. Survey of Advanced *Levelization* Techniques

Callbacks: Functor

```
// anydarnthing.h
class EventMgr;
class AnyDarnThing {
    void invoke();
public:
    // ...
    void schedule(EventMgr *eMgr);
}
```

```
// anydarnthing.cpp
#include <eventmgr.h>
// ...
void AnyDarnThing::schedule(
    EventMgr *eMgr) {
    eMgr->schedule(invoker, this);
}
```

```
// eventmgr.h
// ...
class EventMgr {
    // ...
public:
    typedef
        std::function<void()>& EventCb;
    // ...
    void schedule(EventCb cb);
};
```



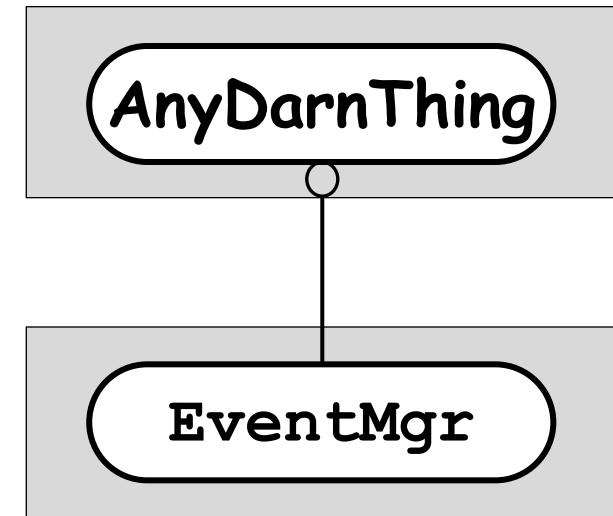
2. Survey of Advanced *Levelization* Techniques

Callbacks: Functor

```
// anydarnthing.h
class EventMgr;
class AnyDarnThing {
    void invoke();
public:
    // ...
    void schedule(EventMgr *eMgr);
}
```

```
// anydarnthing.cpp
#include <eventmgr.h>
// ...
void AnyDarnThing::schedule(
    EventMgr *eMgr) {
    eMgr->schedule(invoker, this);
}
```

```
// eventmgr.h
// ...
class EventMgr {
    // ...
public:
    typedef
        std::function<void()>& EventCb;
    // ...
    void schedule(EventCb cb);
};
```



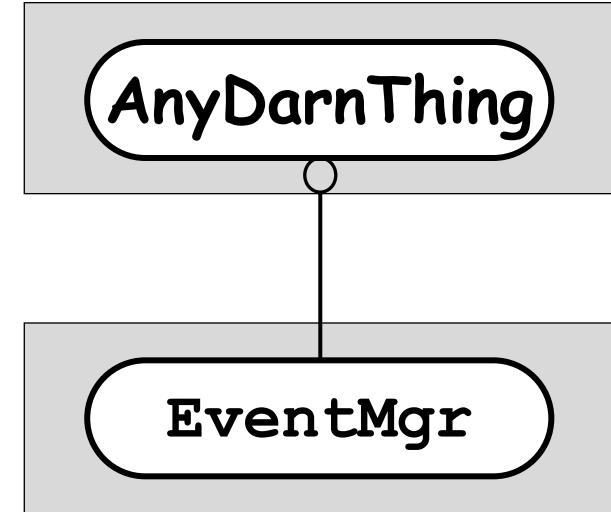
2. Survey of Advanced *Levelization* Techniques

Callbacks: Functor

```
// anydarnthing.h
class EventMgr;
class AnyDarnThing {
    void invoke();
public:
    // ...
    void schedule(EventMgr *eMgr);
}
```

```
// anydarnthing.cpp
#include <eventmgr.h>
// ...
void AnyDarnThing::schedule(
    EventMgr *eMgr) {
    eMgr->schedule(bind(
        &AnyDarnThing::invoke, this));
}
```

```
// eventmgr.h
// ...
class EventMgr {
    // ...
public:
    typedef
        std::function<void()>& EventCb;
    // ...
    void schedule(EventCb cb);
};
```



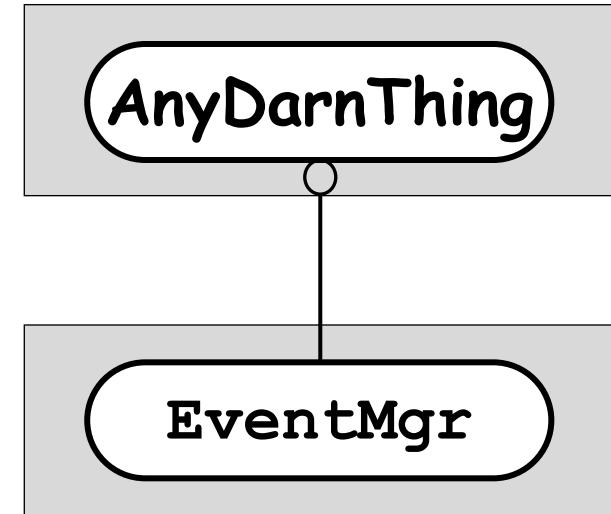
2. Survey of Advanced *Levelization* Techniques

Callbacks: Functor

```
// anydarnthing.h
class EventMgr;
class AnyDarnThing {
    void invoke();
public:
    // ...
    void schedule(EventMgr *eMgr);
}
```

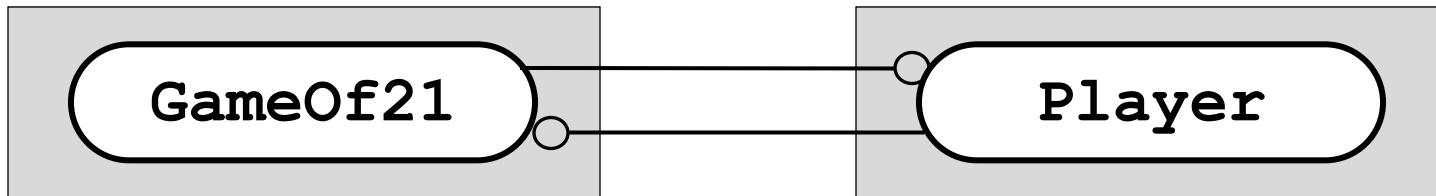
```
// anydarnthing.cpp
#include <eventmgr.h>
// ...
void AnyDarnThing::schedule(
    EventMgr *eMgr) {
    eMgr->schedule(
        [this]{ invoke(); } );
}
```

```
// eventmgr.h
// ...
class EventMgr {
    // ...
public:
    typedef
        std::function<void()>& EventCb;
    // ...
    void schedule(EventCb cb);
};
```



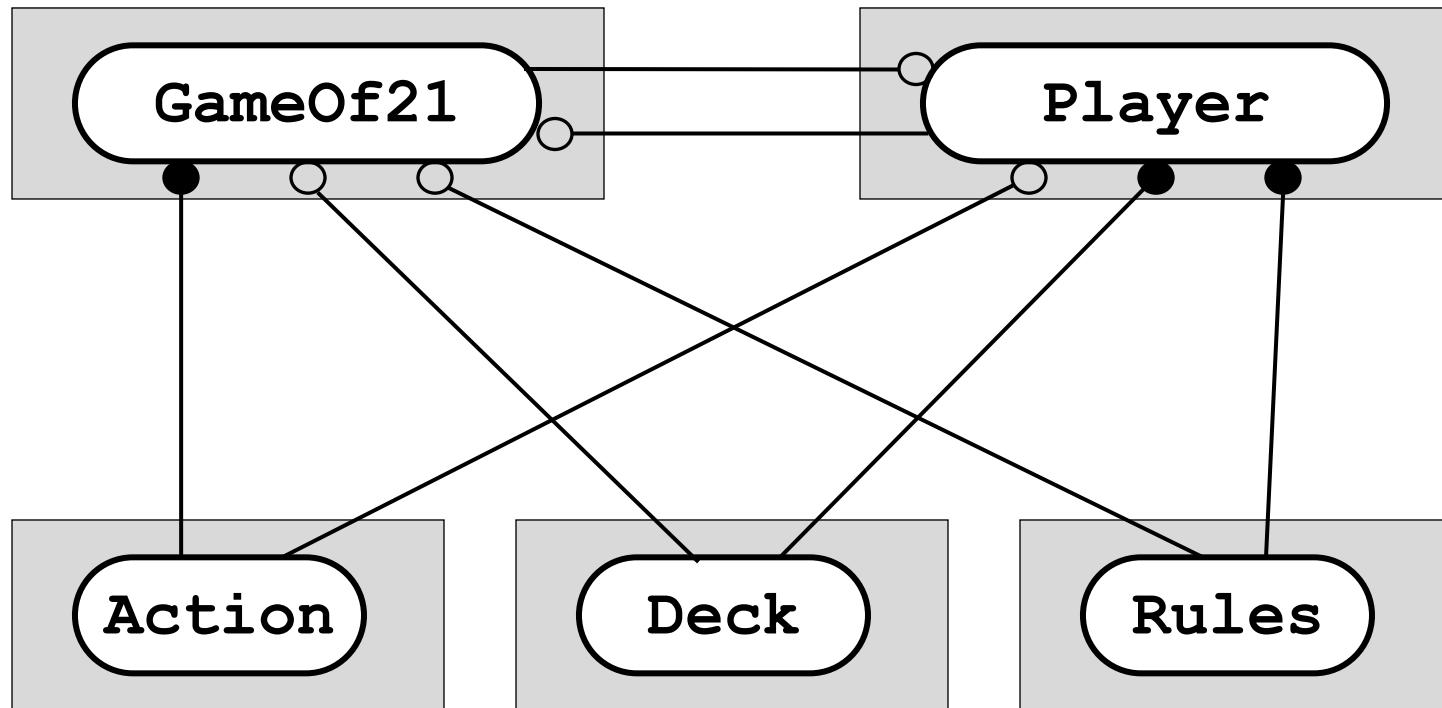
2. Survey of Advanced *Levelization* Techniques

Callbacks: Protocol



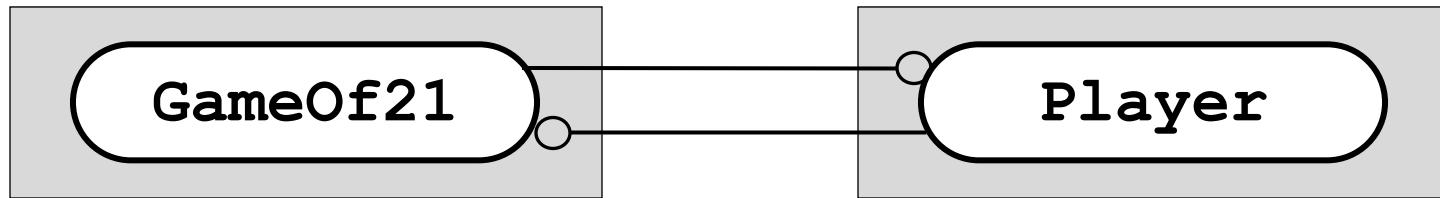
2. Survey of Advanced *Levelization* Techniques

Callbacks: Protocol



2. Survey of Advanced *Levelization* Techniques

Callbacks: Protocol

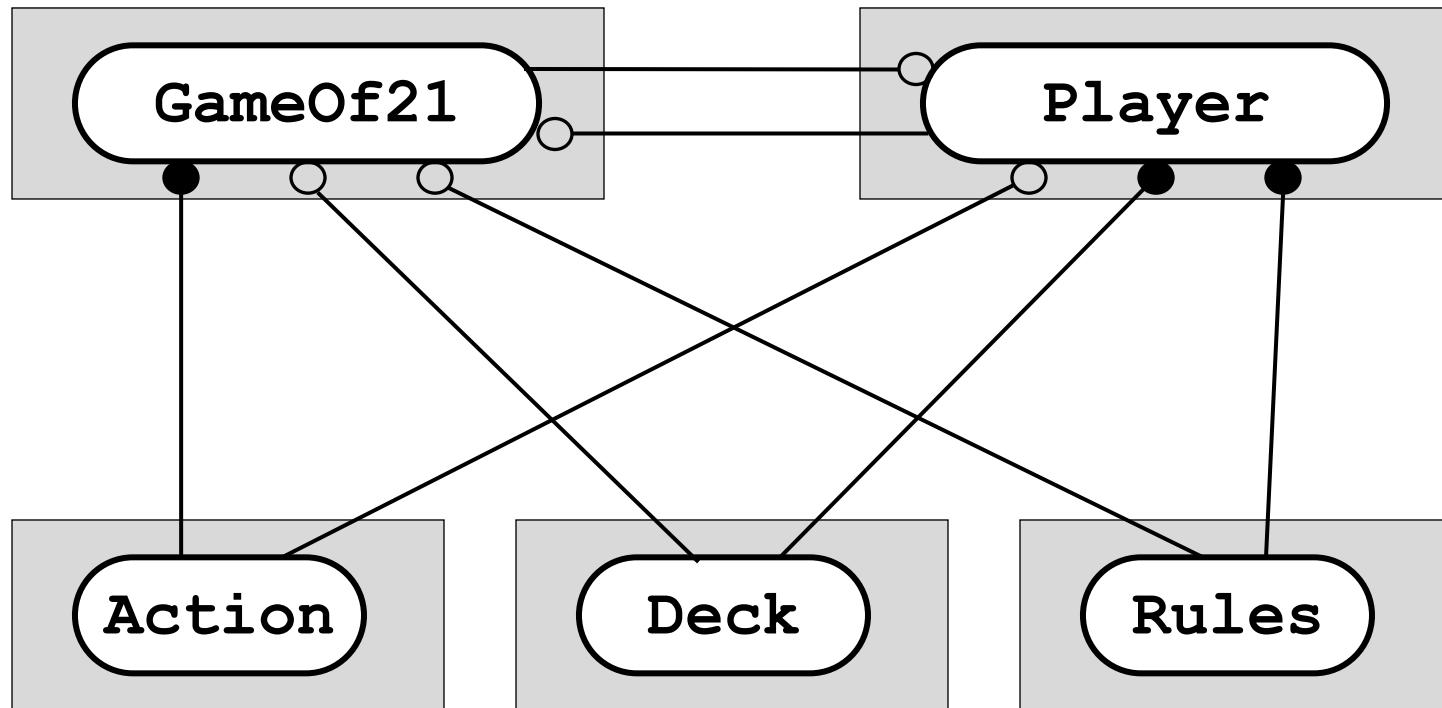


```
// gameof21.h
#include <player.h>
class GameOf21 {
// ...
public:
// ...
    addPlayer(const Player *p,
              int             cash);
void dealCards();
void shuffleCards();
// ...
const Deck& deck() const;
const Rules& rules() const;
};
```

```
// player.h
#include <gameof21.h>
class Player {
// ...
public:
    Player(const char *name);
// ...
    int bet(const GameOf21&
            game) const;
    Action choice(const GameOf21&
                  game) const;
    const char *name() const;
};
```

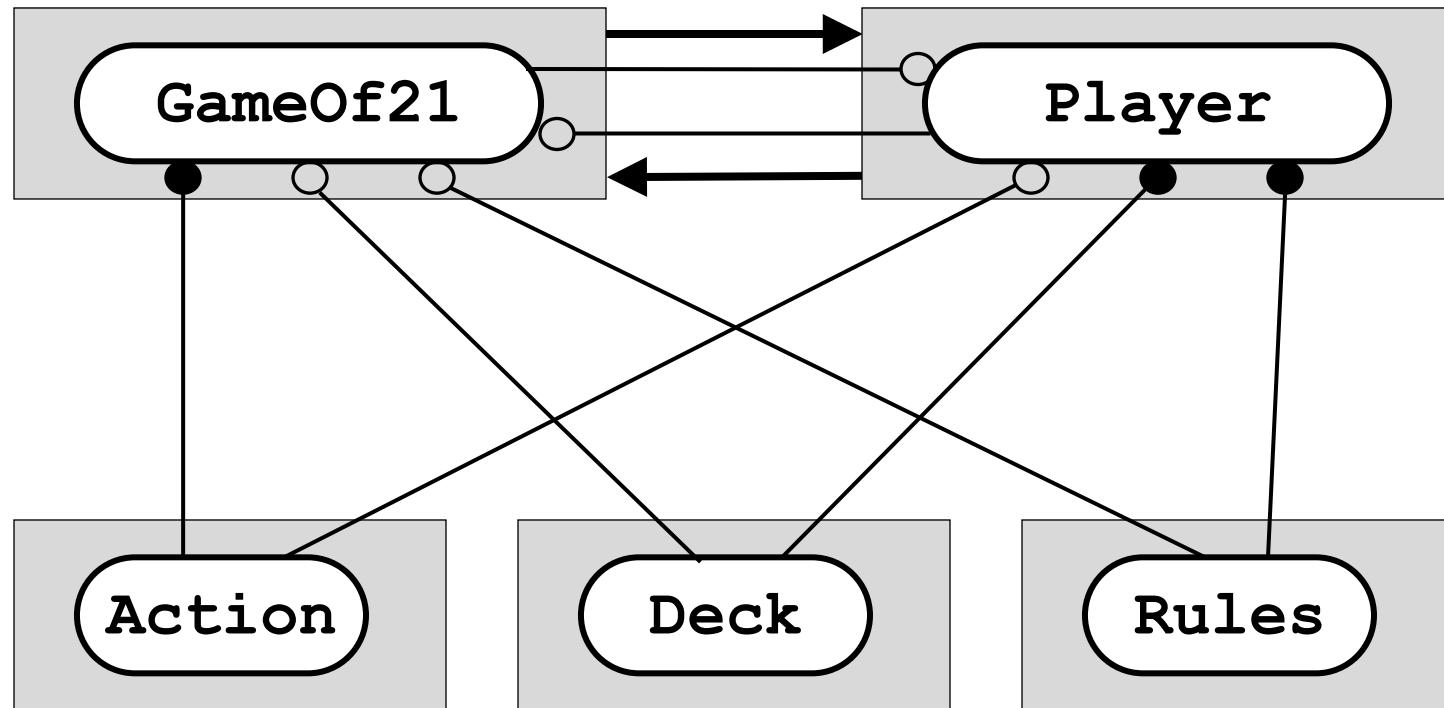
2. Survey of Advanced *Levelization* Techniques

Callbacks: Protocol



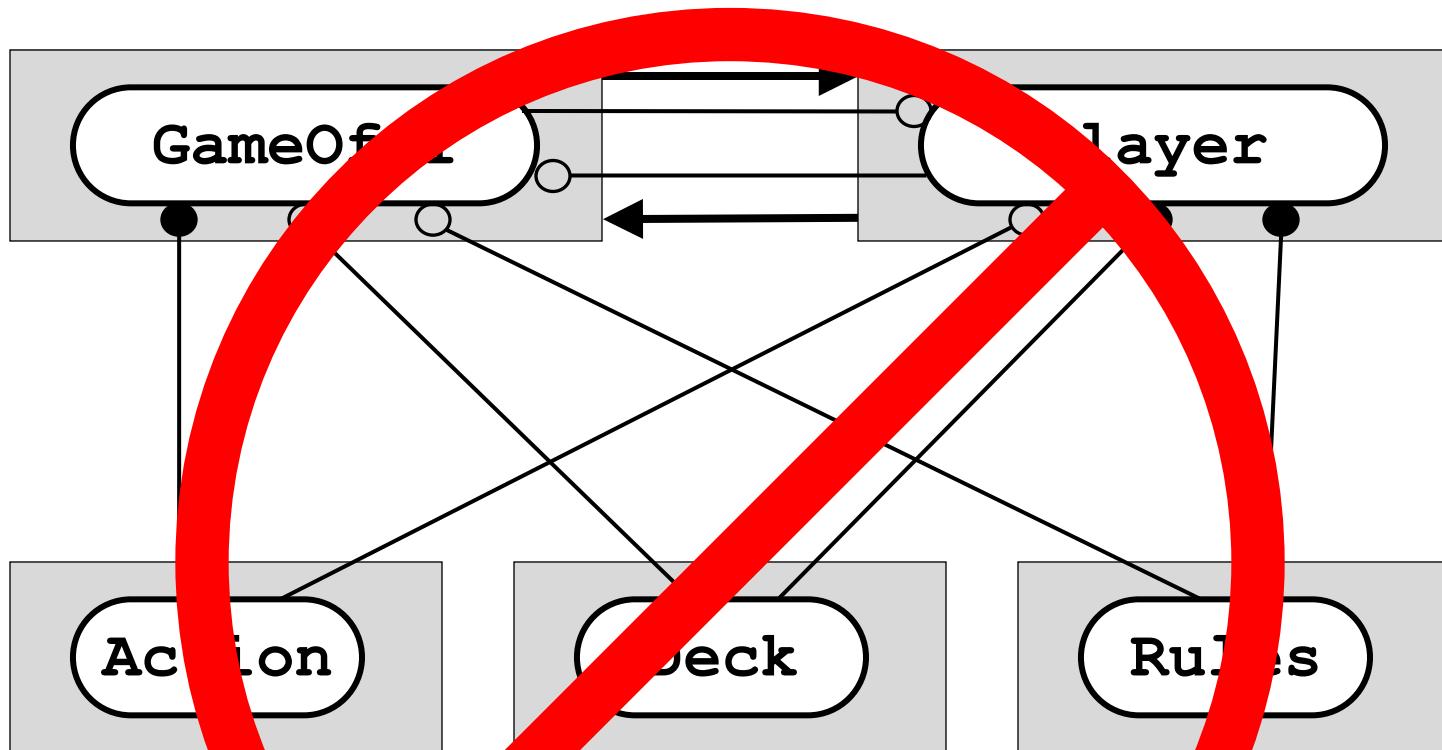
2. Survey of Advanced *Levelization* Techniques

Callbacks: Protocol



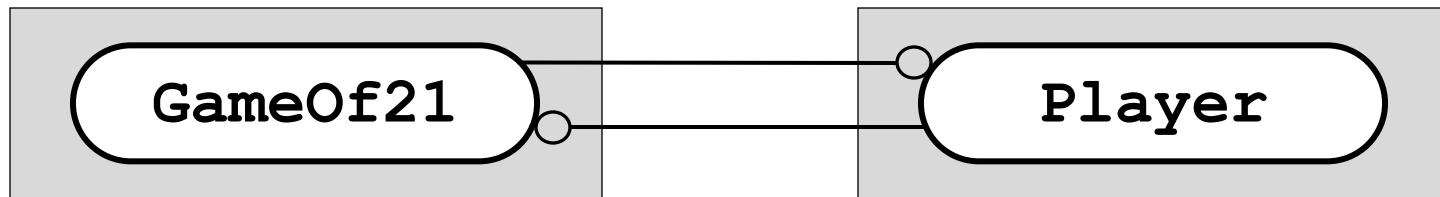
2. Survey of Advanced *Levelization* Techniques

Callbacks: Protocol



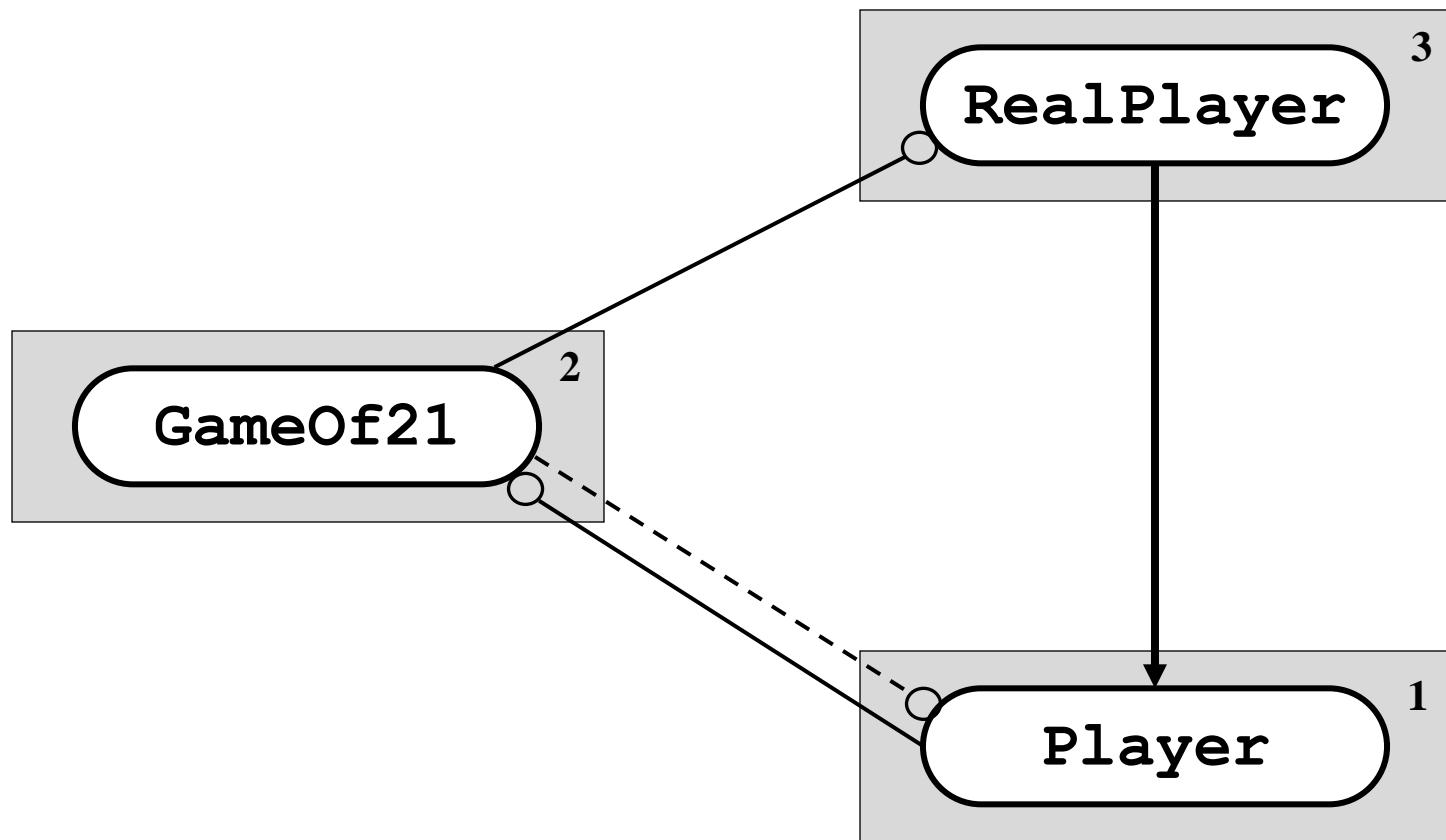
2. Survey of Advanced *Levelization* Techniques

Callbacks: Protocol



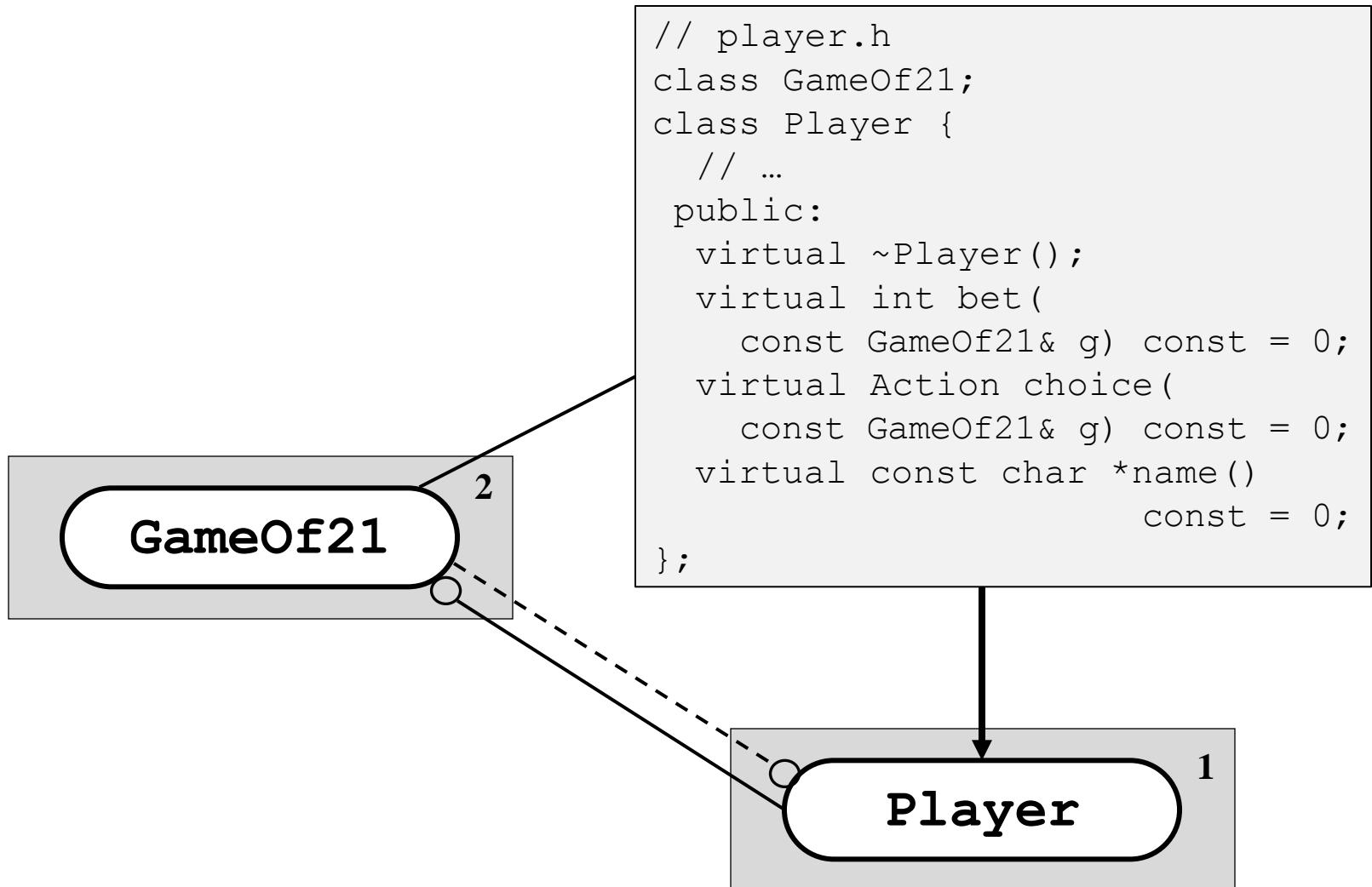
2. Survey of Advanced *Levelization* Techniques

Callbacks: Protocol



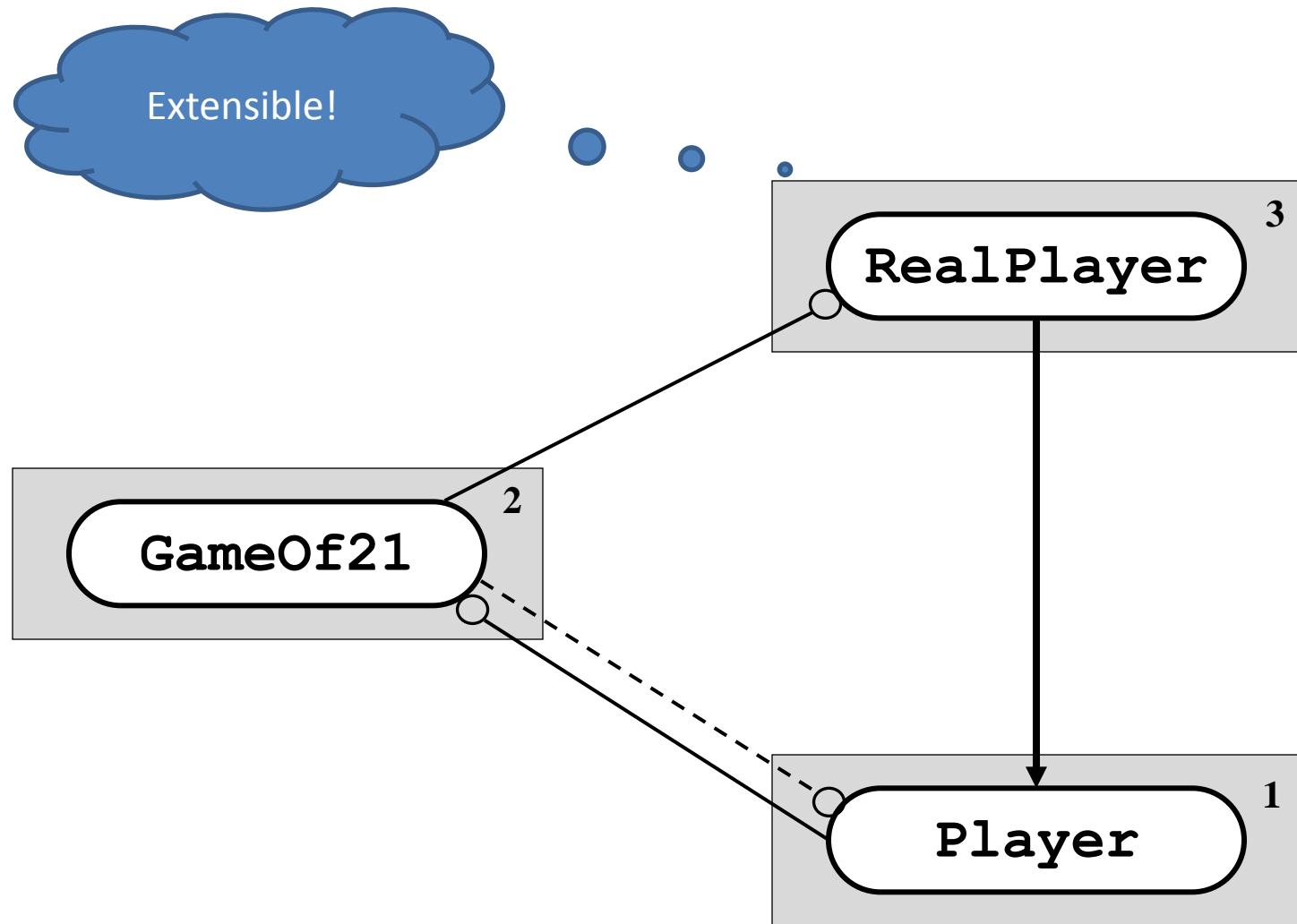
2. Survey of Advanced *Levelization* Techniques

Callbacks: Protocol



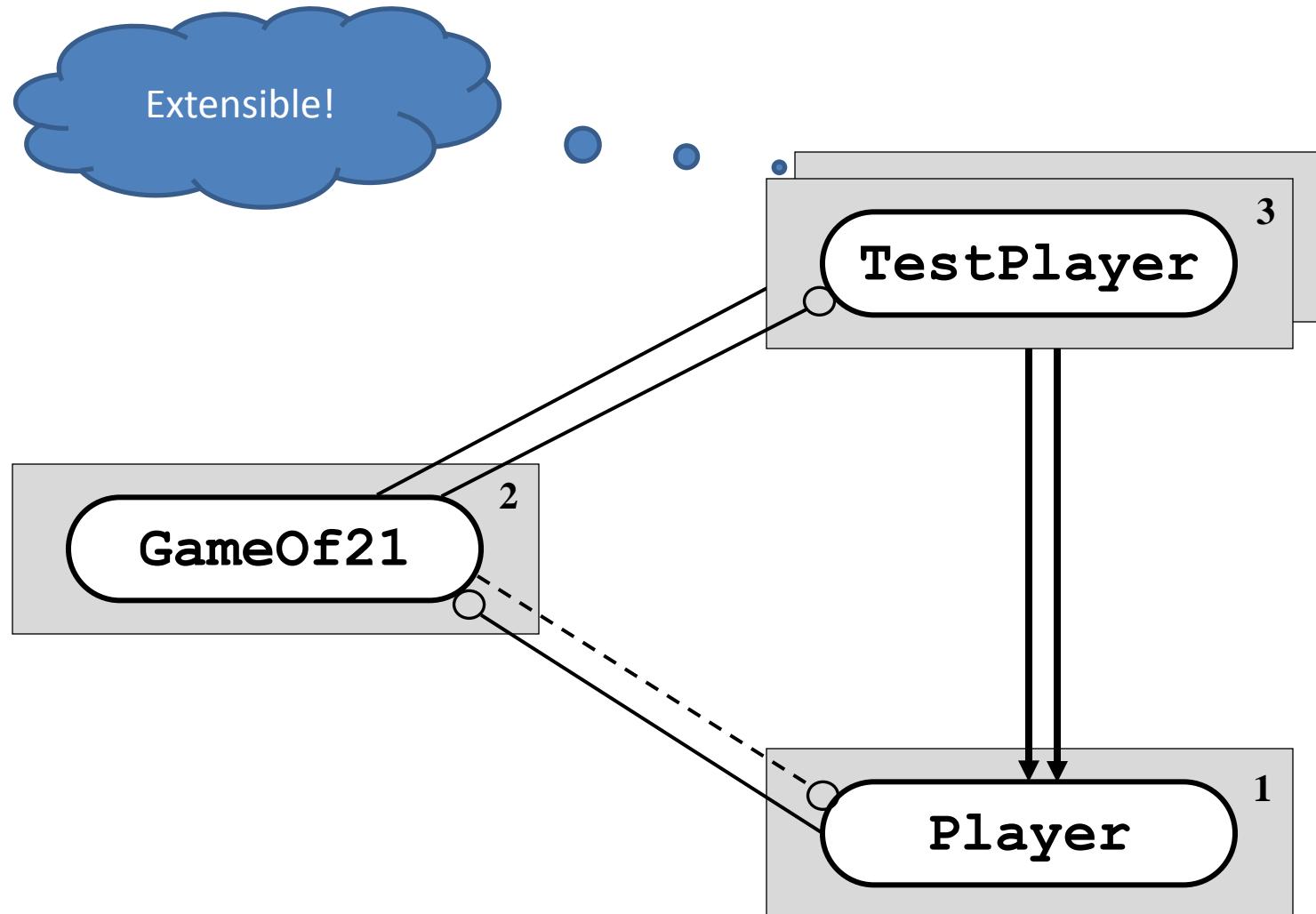
2. Survey of Advanced *Levelization* Techniques

Callbacks: Protocol



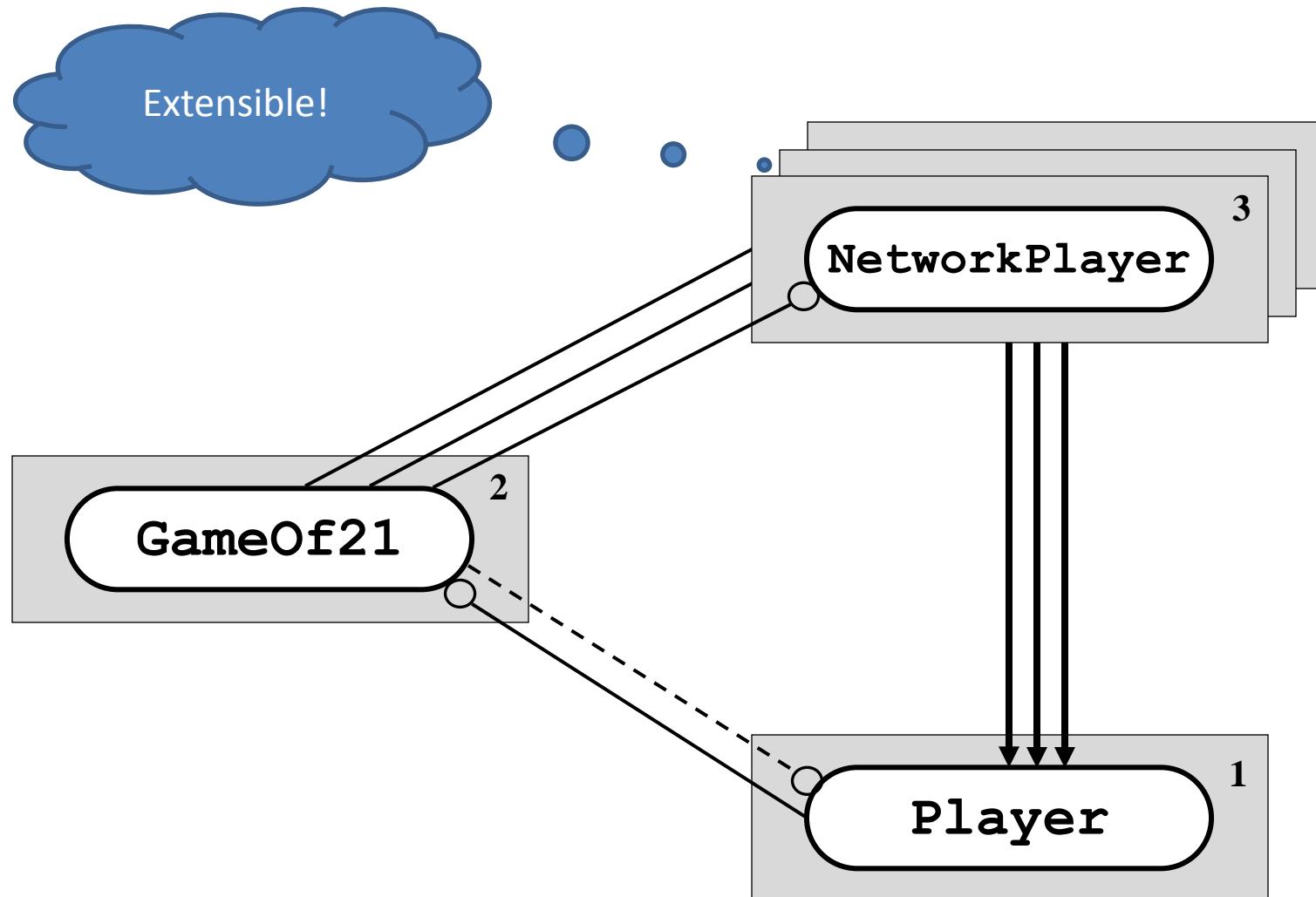
2. Survey of Advanced *Levelization* Techniques

Callbacks: Protocol



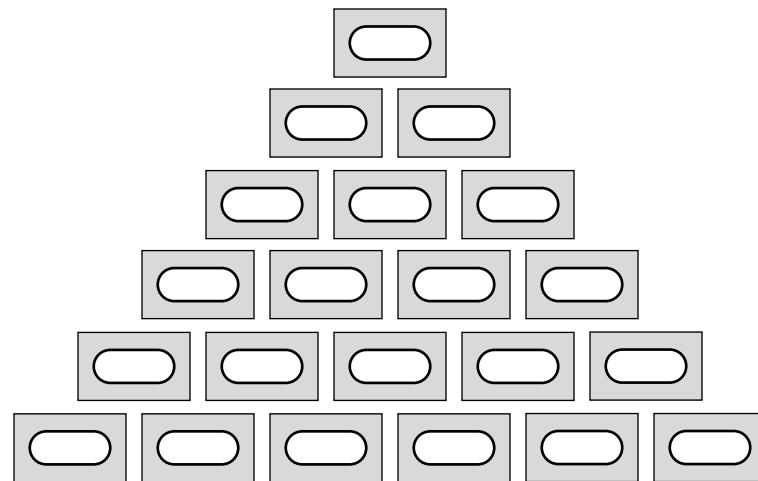
2. Survey of Advanced *Levelization* Techniques

Callbacks: Protocol



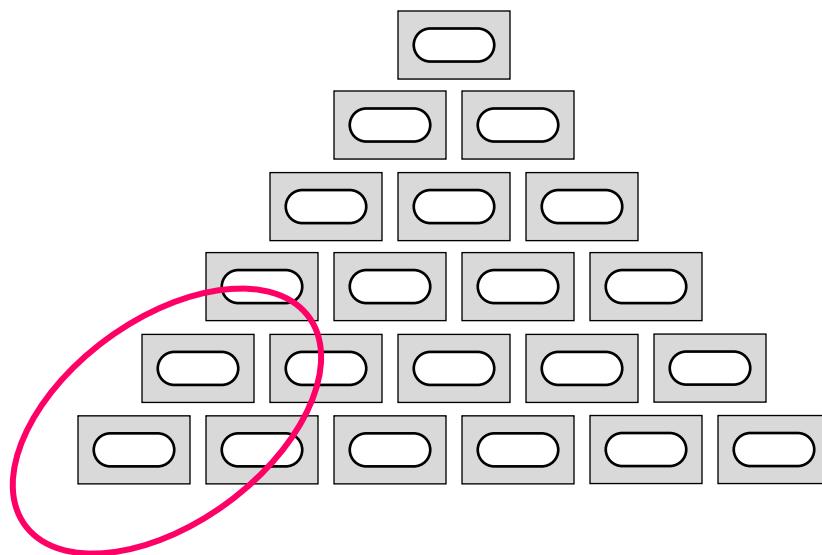
2. Survey of Advanced *Levelization* Techniques

Callbacks: Protocol



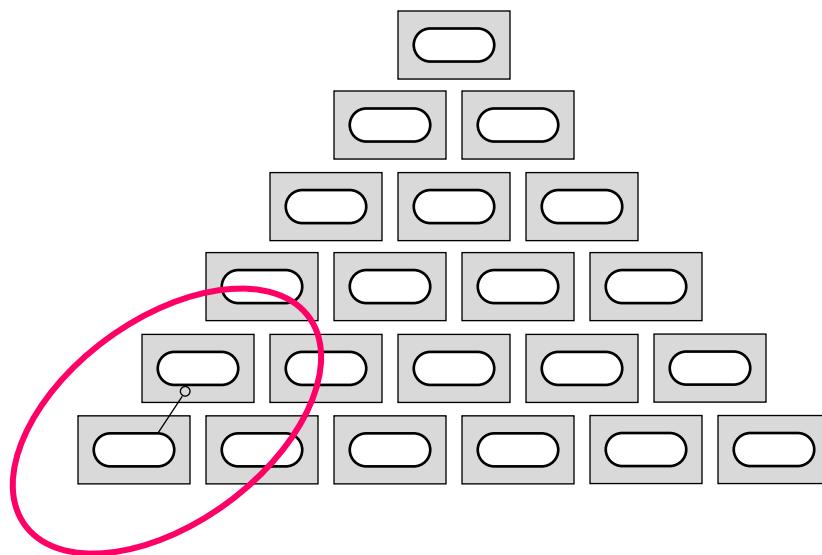
2. Survey of Advanced *Levelization* Techniques

Callbacks: Protocol



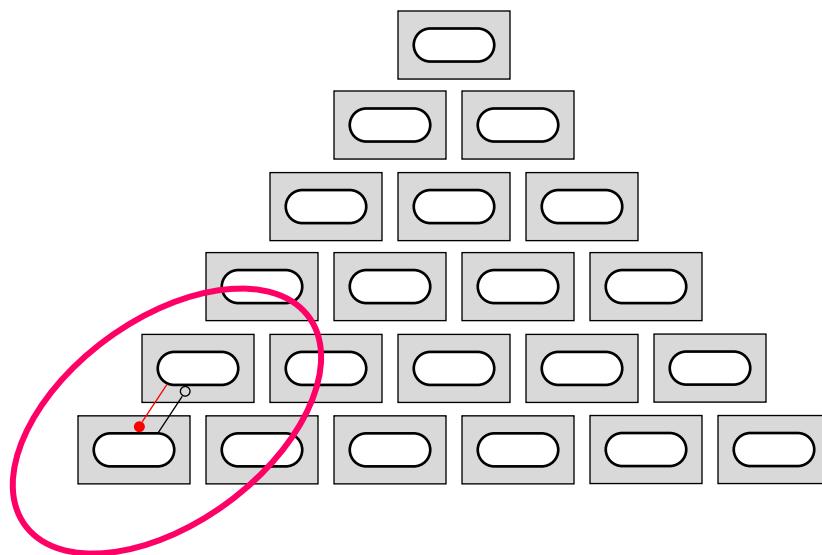
2. Survey of Advanced *Levelization* Techniques

Callbacks: Protocol



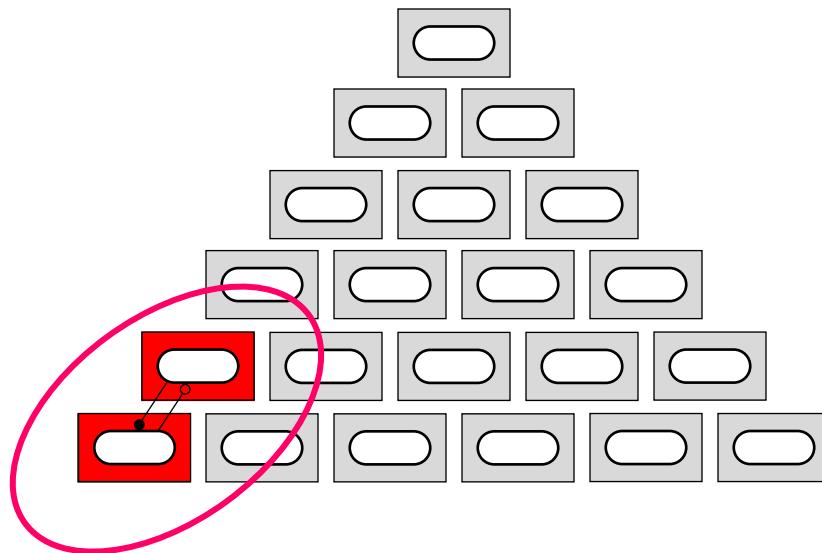
2. Survey of Advanced *Levelization* Techniques

Callbacks: Protocol



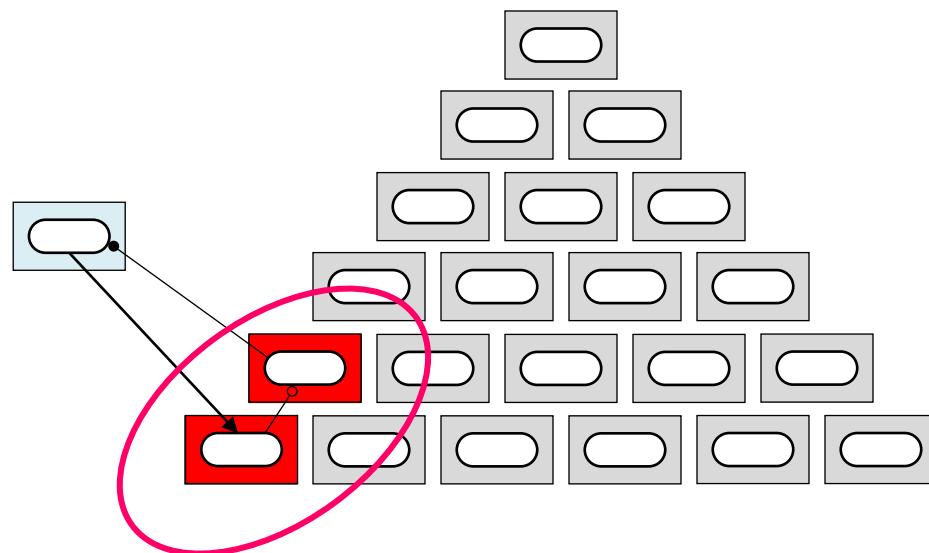
2. Survey of Advanced *Levelization* Techniques

Callbacks: Protocol



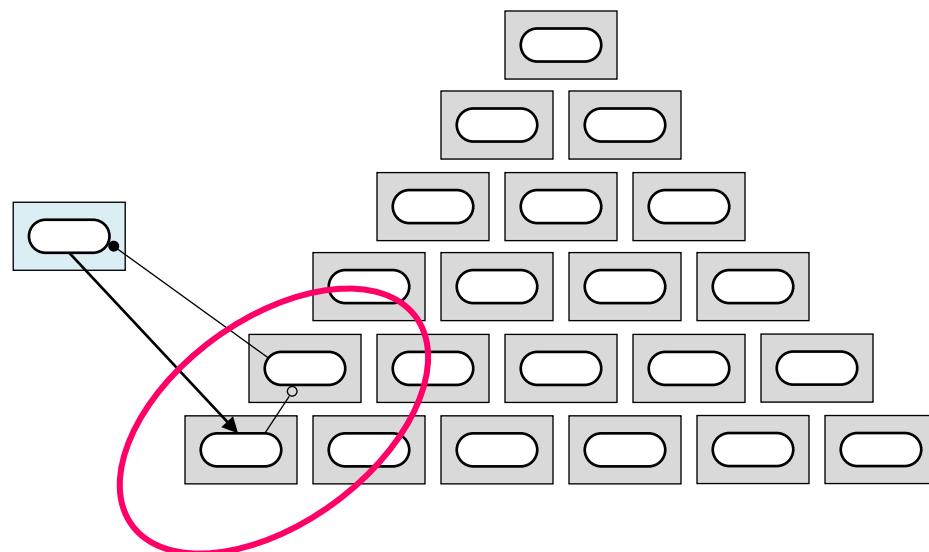
2. Survey of Advanced *Levelization* Techniques

Callbacks: Protocol



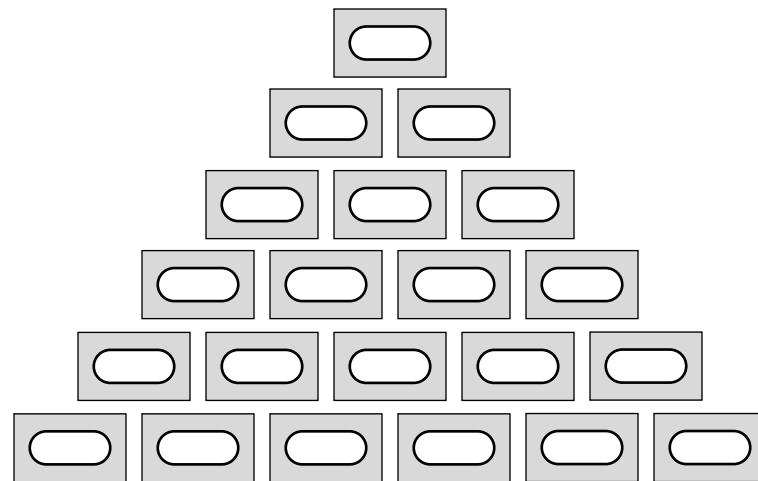
2. Survey of Advanced *Levelization* Techniques

Callbacks: Protocol



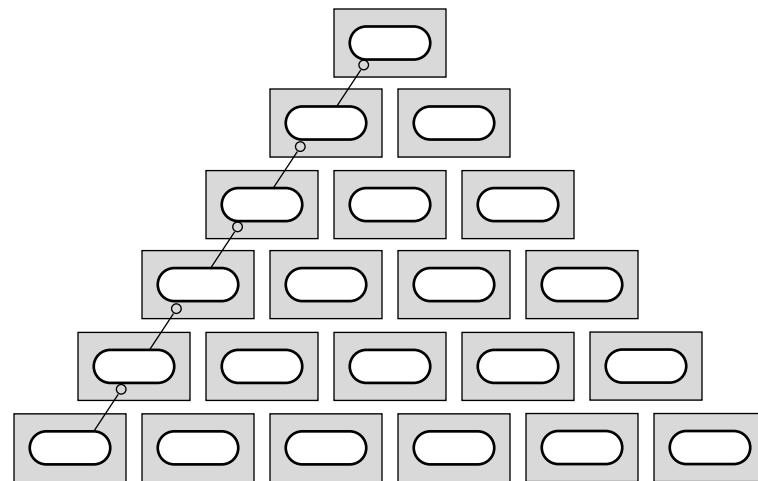
2. Survey of Advanced *Levelization* Techniques

Callbacks: Protocol



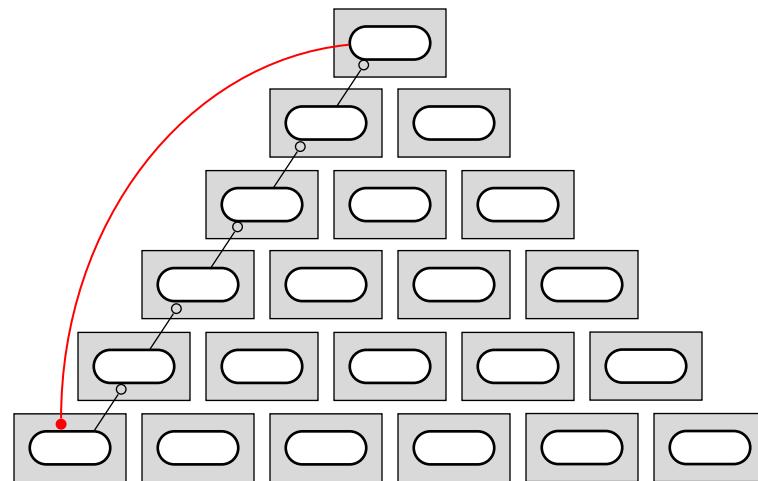
2. Survey of Advanced *Levelization* Techniques

Callbacks: Protocol



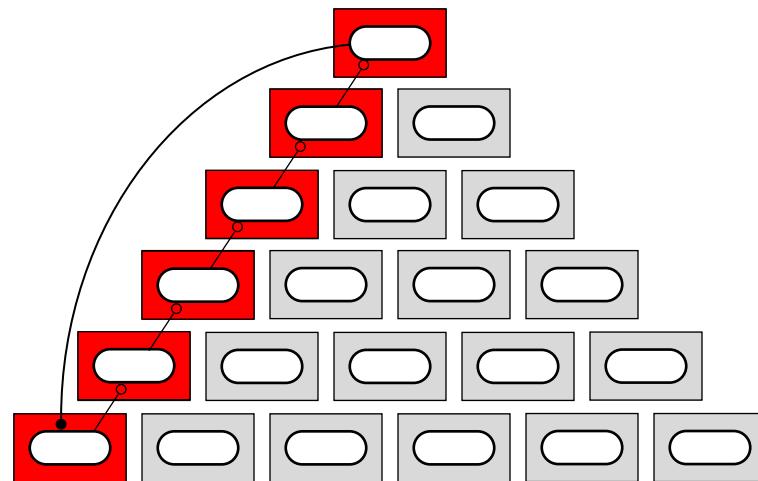
2. Survey of Advanced *Levelization* Techniques

Callbacks: Protocol



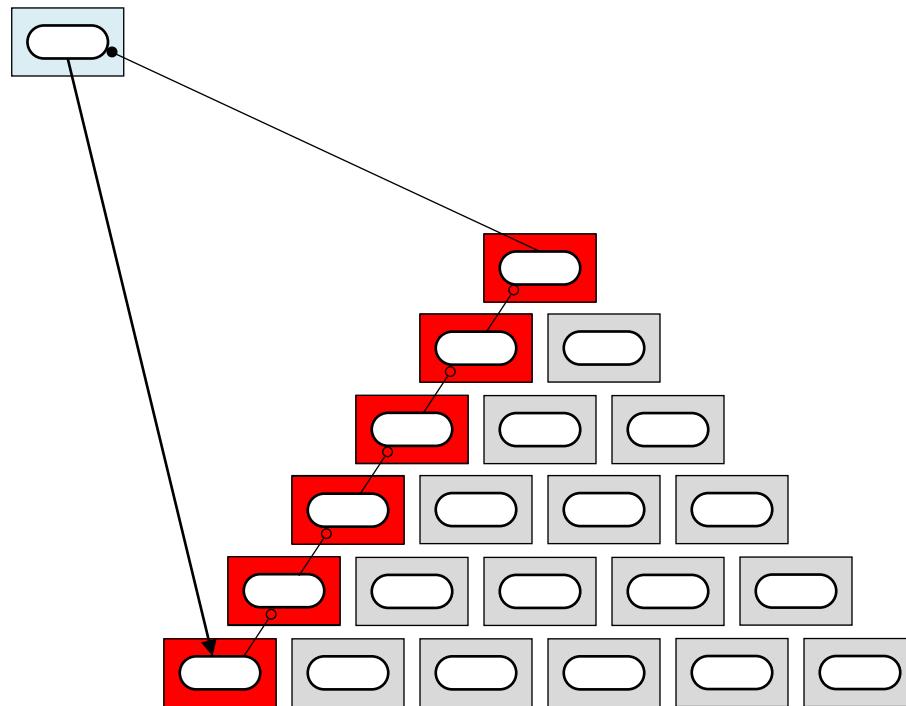
2. Survey of Advanced *Levelization* Techniques

Callbacks: Protocol



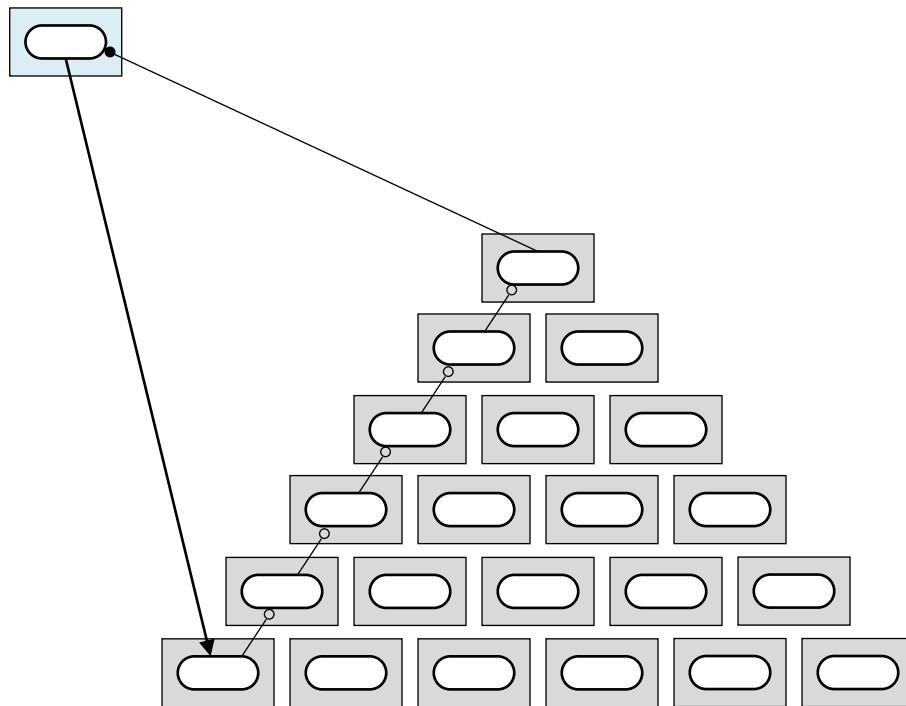
2. Survey of Advanced *Levelization* Techniques

Callbacks: Protocol



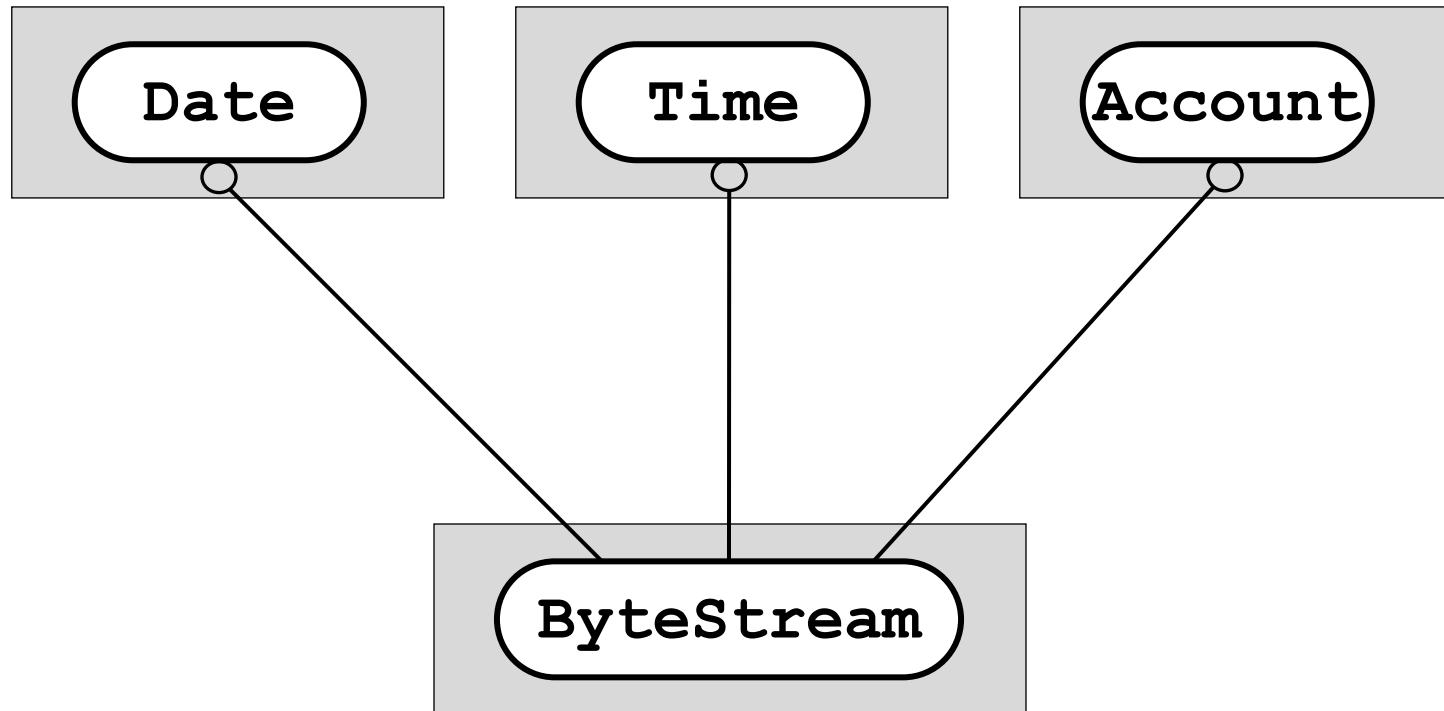
2. Survey of Advanced *Levelization* Techniques

Callbacks: Protocol



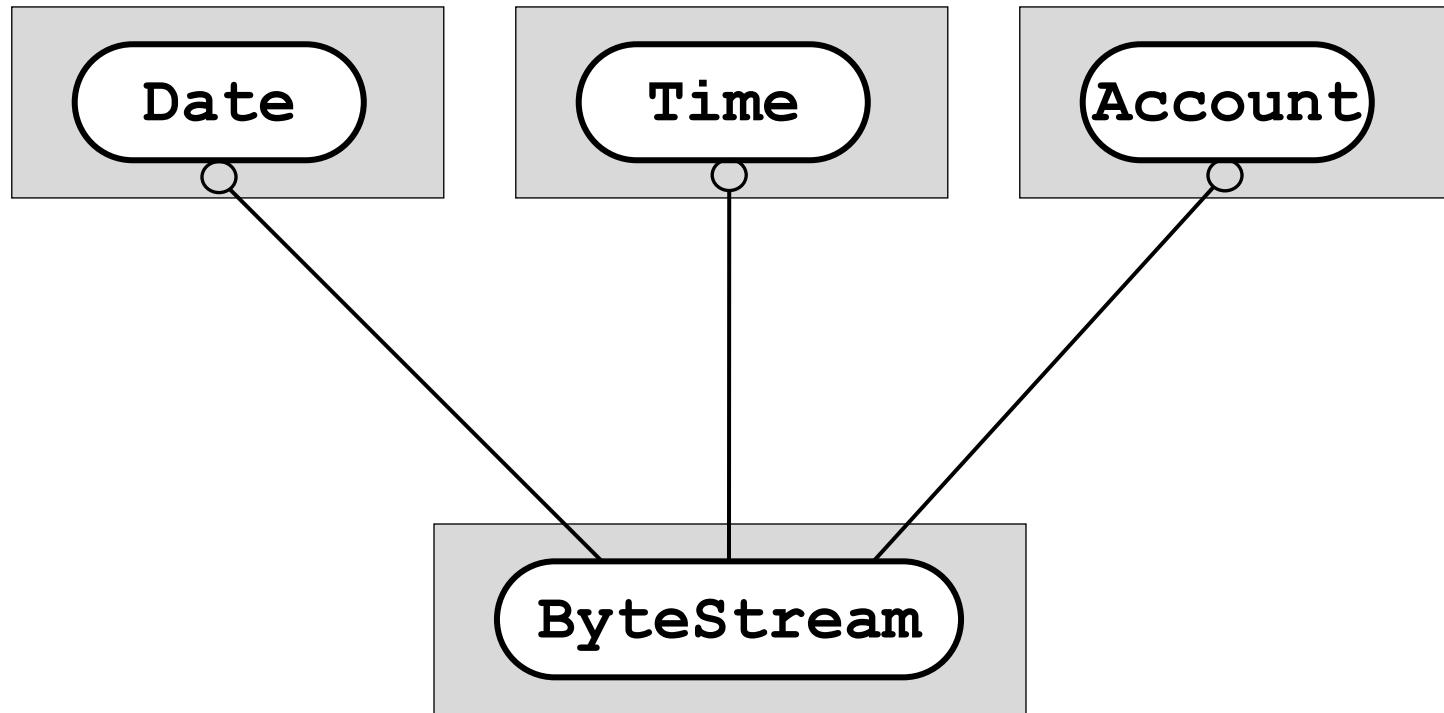
2. Survey of Advanced *Levelization* Techniques

Callbacks: Concept



2. Survey of Advanced *Levelization* Techniques

Callbacks: Concept



Externalization

2. Survey of Advanced *Levelization* Techniques

Callbacks: Concept



Externalization

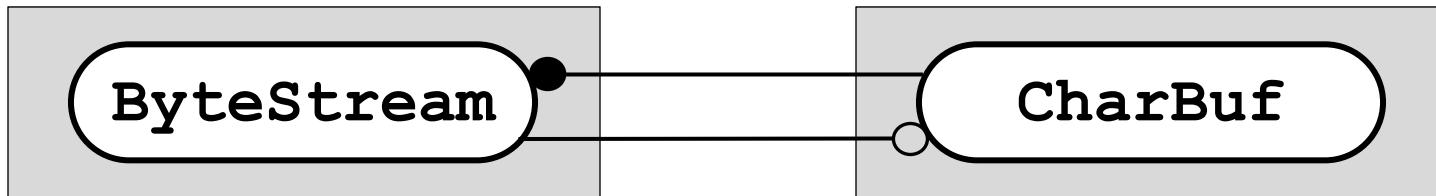
2. Survey of Advanced *Levelization* Techniques

Callbacks: Concept

The slide features a central decorative graphic. At the top, the word "Cool" is written in large, red, 3D-style letters. Below it, the word "Graphics" is also in large, red, 3D-style letters. To the left of "Graphics", there is a small, light gray rectangular callout box with a rounded bottom right corner containing the word "Date". To the right of "Graphics", there is another small, light gray rectangular callout box with a rounded bottom right corner containing the word "Time". To the right of "Graphics", there is a third small, light gray rectangular callout box with a rounded bottom right corner containing the word "Account". Below the "Cool Graphics" section, the word "Alert" is written in large, red, 3D-style letters. Below "Alert", the word "Externalization" is written in large, green, 3D-style letters.

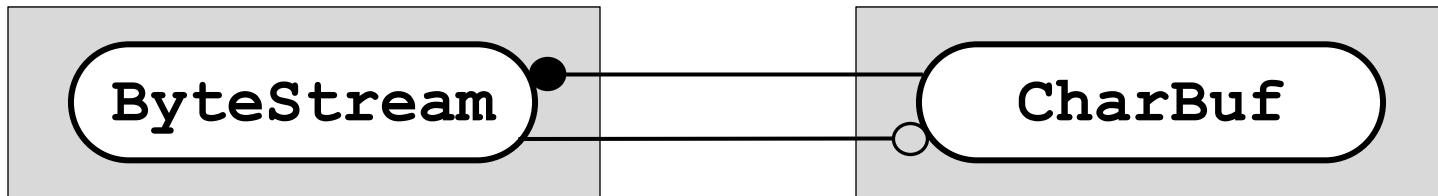
2. Survey of Advanced *Levelization* Techniques

Callbacks: Concept



2. Survey of Advanced *Levelization* Techniques

Callbacks: Concept

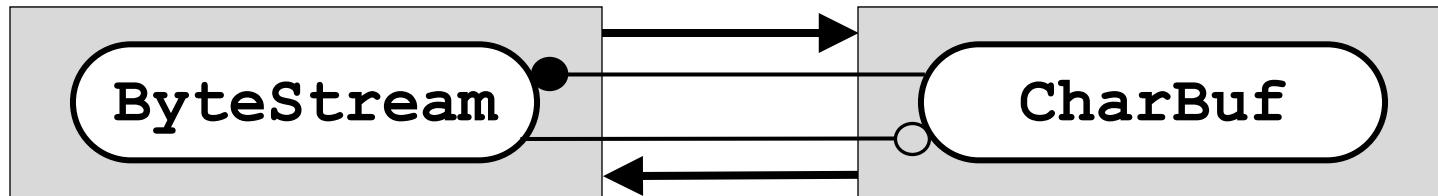


```
// bytestream.h
#include <charbuf.h>
class ByteStream {
    CharBuf d_streamData;
    // ...
public:
    // ...
    void putInt8(int value);
    void putInt16(int value);
    // ...
    void putFloat32(double value);
    void putFloat64(double value);
    // ...
};
```

```
// charbuf.h
#include <bytestream.h>
class CharBuf {
    char *d_buffer_p;
    // ...
public:
    // ...
    void appendChar(char c);
    size_t length() const;
    char operator[](size_t i) const;
    // ...
    void
    streamOut(ByteStream *s) const;
};
```

2. Survey of Advanced *Levelization* Techniques

Callbacks: Concept

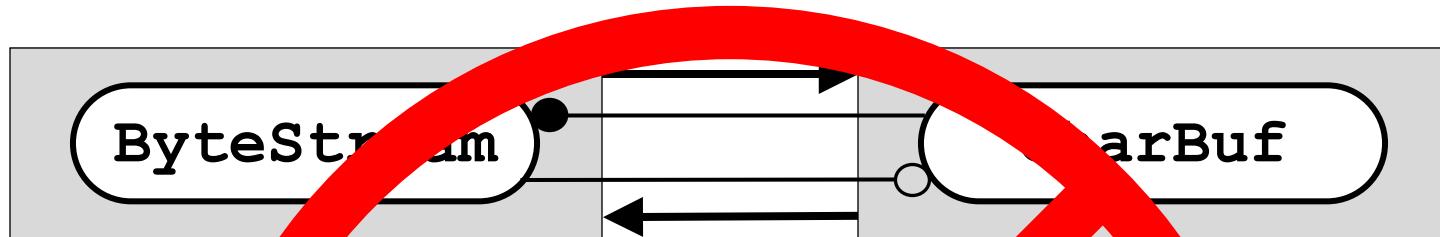


```
// bytestream.h
#include <charbuf.h>
class ByteStream {
    CharBuf d_streamData;
    // ...
public:
    // ...
    void putInt8(int value);
    void putInt16(int value);
    // ...
    void putFloat32(double value);
    void putFloat64(double value);
    // ...
};
```

```
// charbuf.h
#include <bytestream.h>
class CharBuf {
    char *d_buffer_p;
    // ...
public:
    // ...
    void appendChar(char c);
    size_t length() const;
    char operator[](size_t i) const;
    // ...
    void
    streamOut(ByteStream *s) const;
};
```

2. Survey of Advanced *Levelization* Techniques

Callbacks: Concept

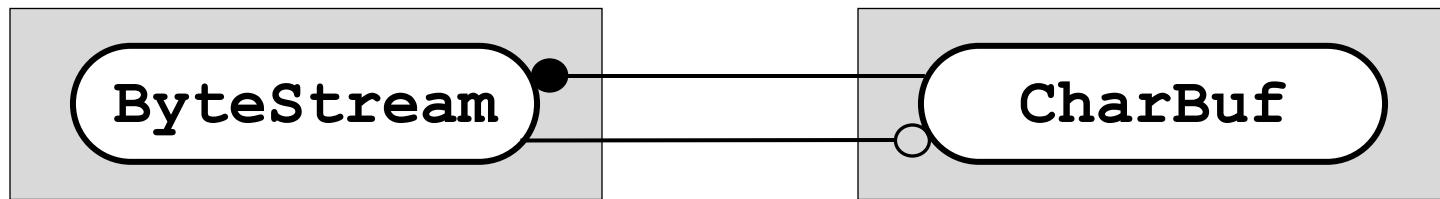


```
// bytestream.h
#include <charbuf.h>
class ByteStream {
    CharBuf d_streamData;
    // ...
public:
    // ...
    void putInt(int value);
    void putInt64(int value);
    // ...
    void putFloat(double value);
    void putFloat64(double value);
    // ...
};
```

```
// charbuf.h
#include <bytestream.h>
class CharBuf {
    char *d_buffer_p
    // ...
public:
    // ...
    void appendChar(char c);
    size_t length() const;
    char operator[](size_t i) const;
    // ...
    void
    streamOut(ByteStream *s) const;
};
```

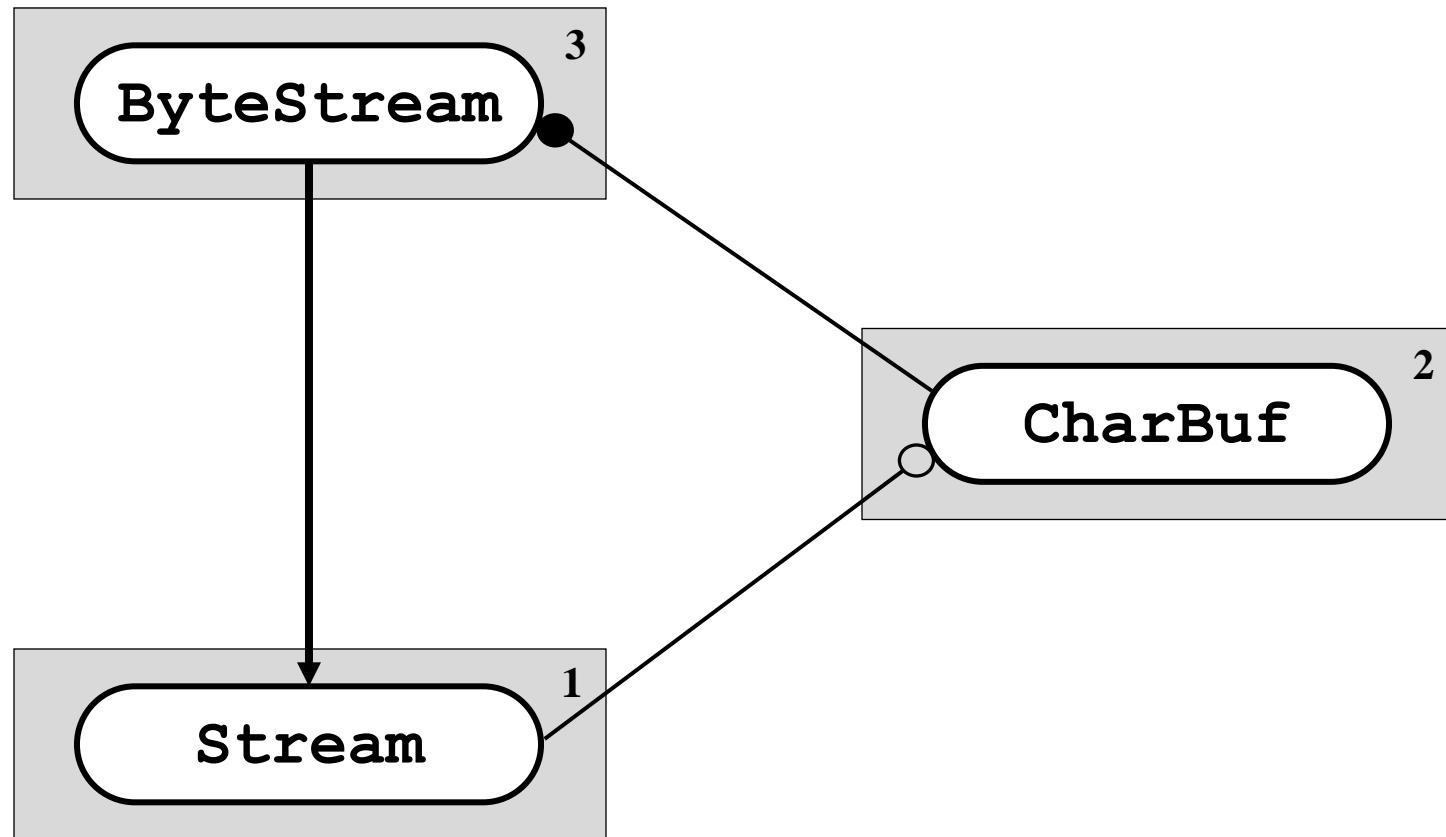
2. Survey of Advanced *Levelization* Techniques

Callbacks: Concept



2. Survey of Advanced *Levelization* Techniques

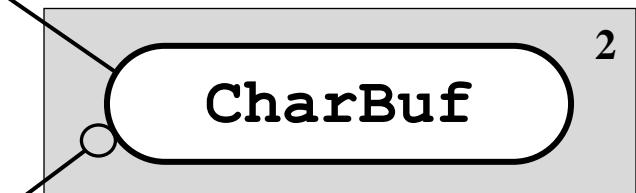
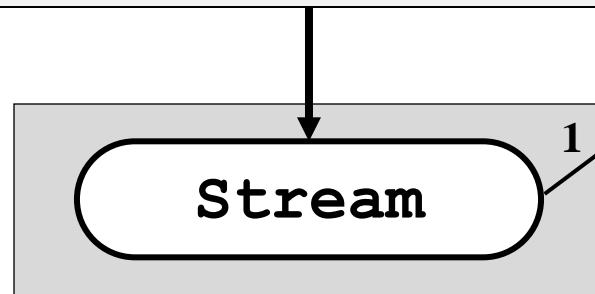
Callbacks: Concept



2. Survey of Advanced *Levelization* Techniques

Callbacks: Concept

```
// stream.h
class Stream {
public:
    virtual ~Stream() ;
    // ...
    virtual void
        putInt8(int value) = 0;
    virtual void
        putInt16(int value) = 0;
    // ...
};
```

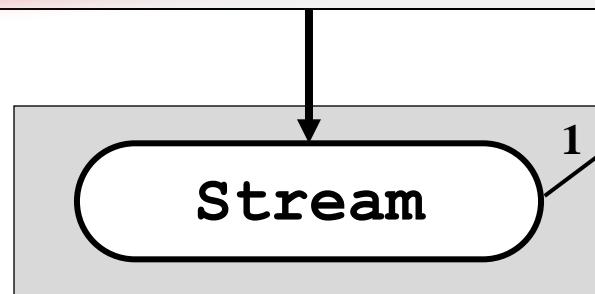


2. Survey of Advanced *Levelization* Techniques

Callbacks: Concept

```
// stream.h
class Stream {
public:
    virtual ~Stream();
    // ...
};
```

Too #@%& Slow!!



2. Survey of Advanced *Levelization* Techniques

Callbacks: Concept

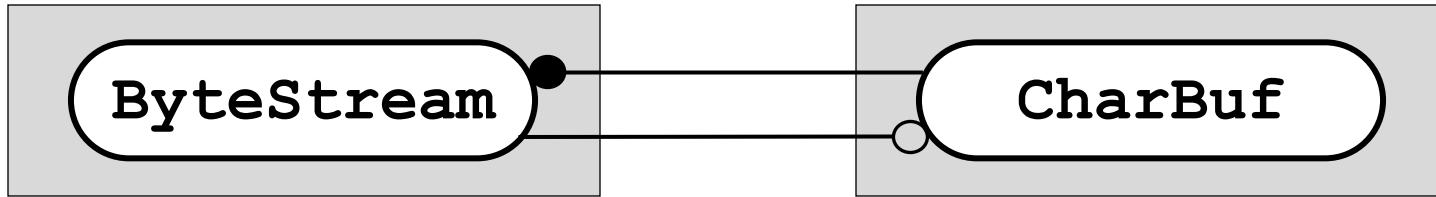
```
// stream.h
class Stream {
public:
    virtual ~Stream();
    // ...
};
```

Too #@%\$ Slow!!



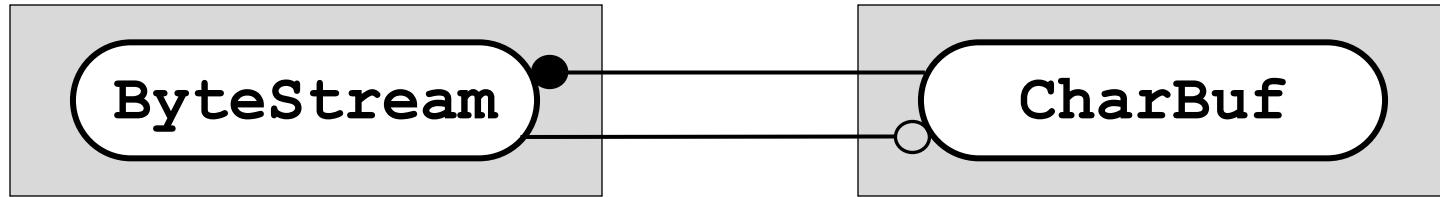
2. Survey of Advanced *Levelization* Techniques

Callbacks: Concept



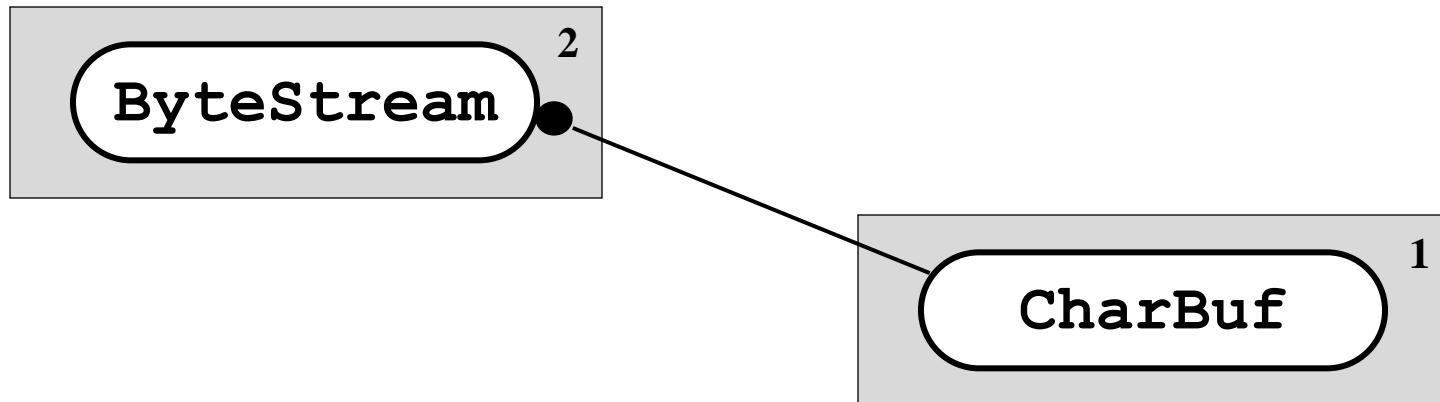
2. Survey of Advanced *Levelization* Techniques

Callbacks: Concept



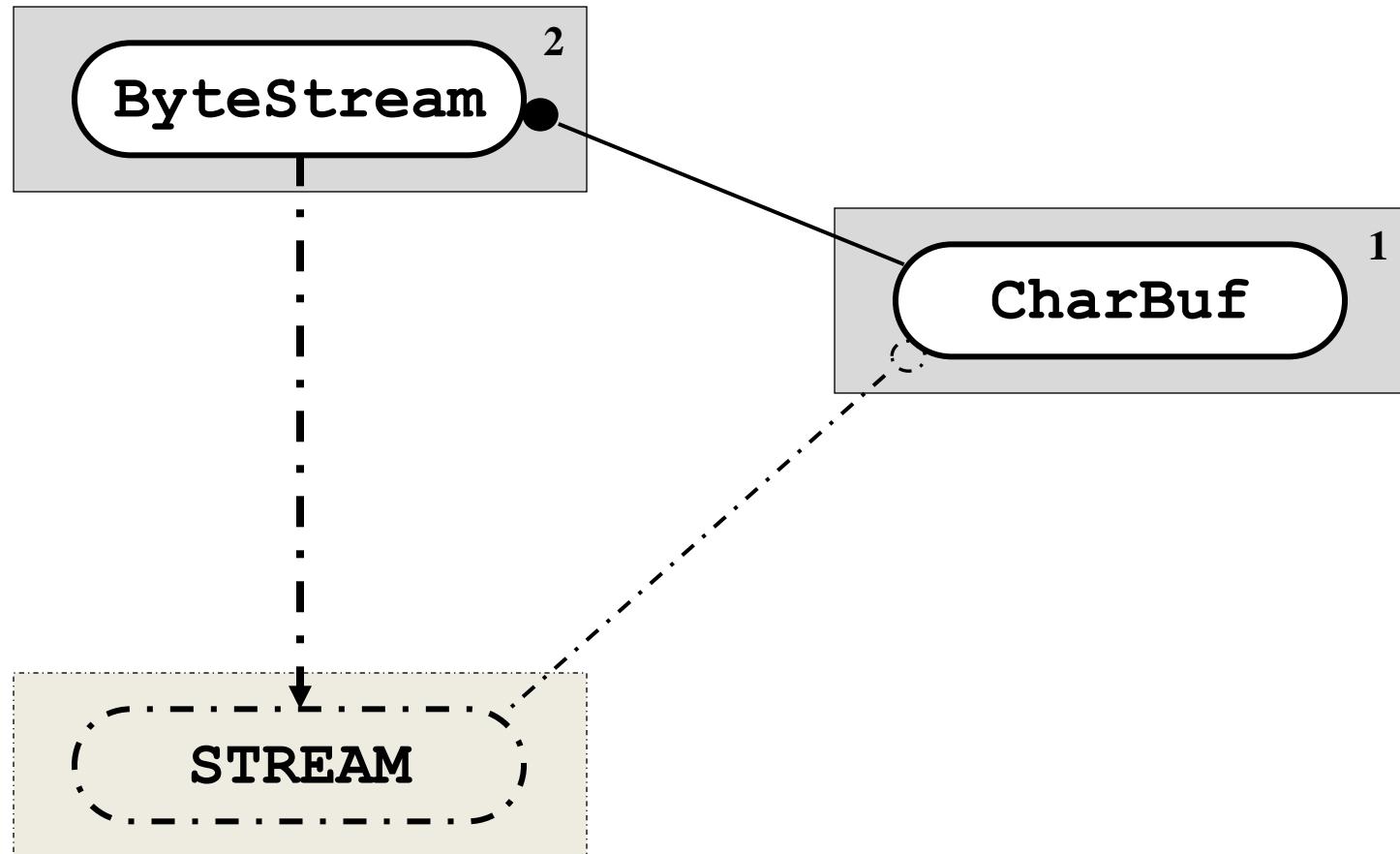
2. Survey of Advanced *Levelization* Techniques

Callbacks: Concept



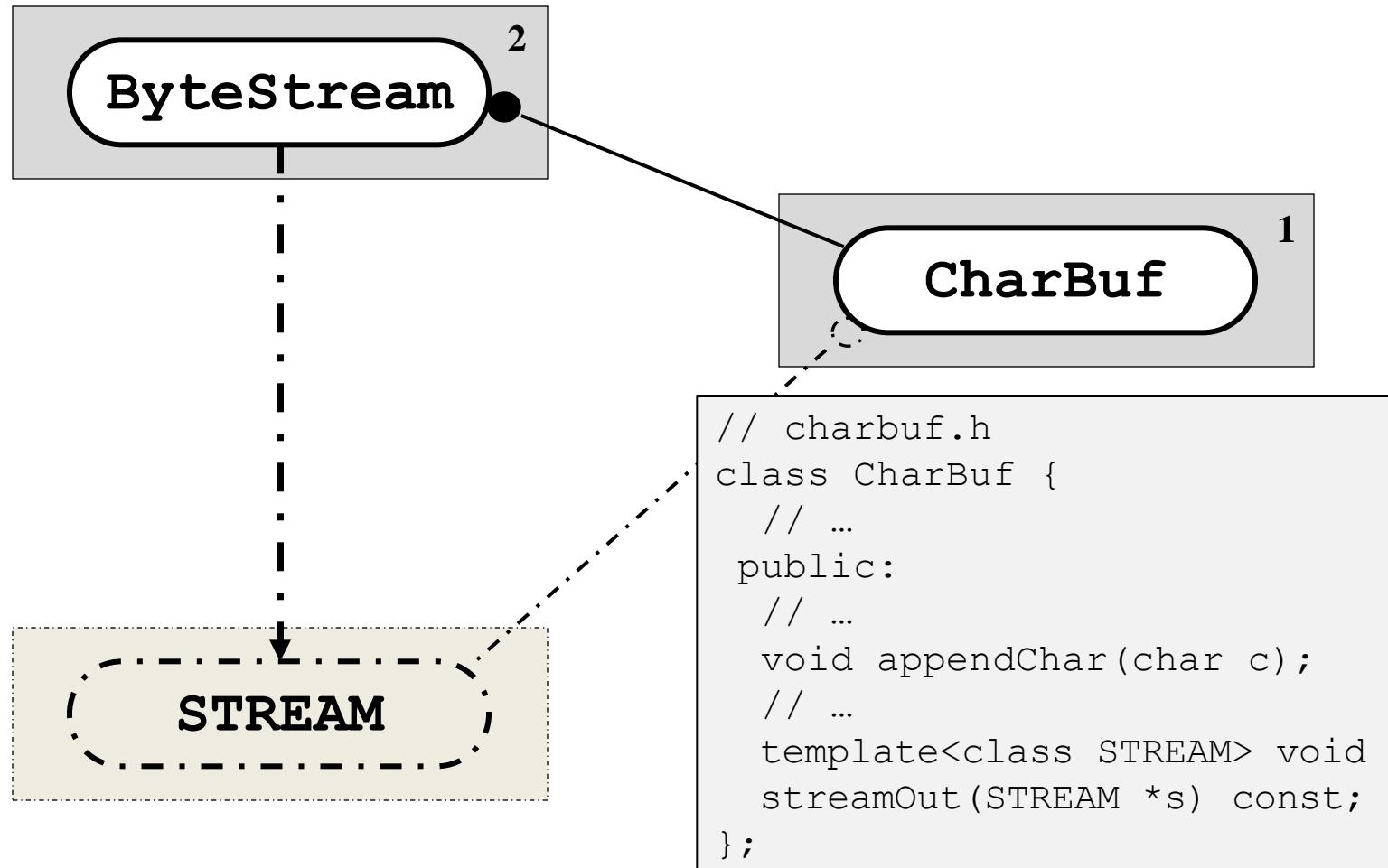
2. Survey of Advanced *Levelization* Techniques

Callbacks: Concept



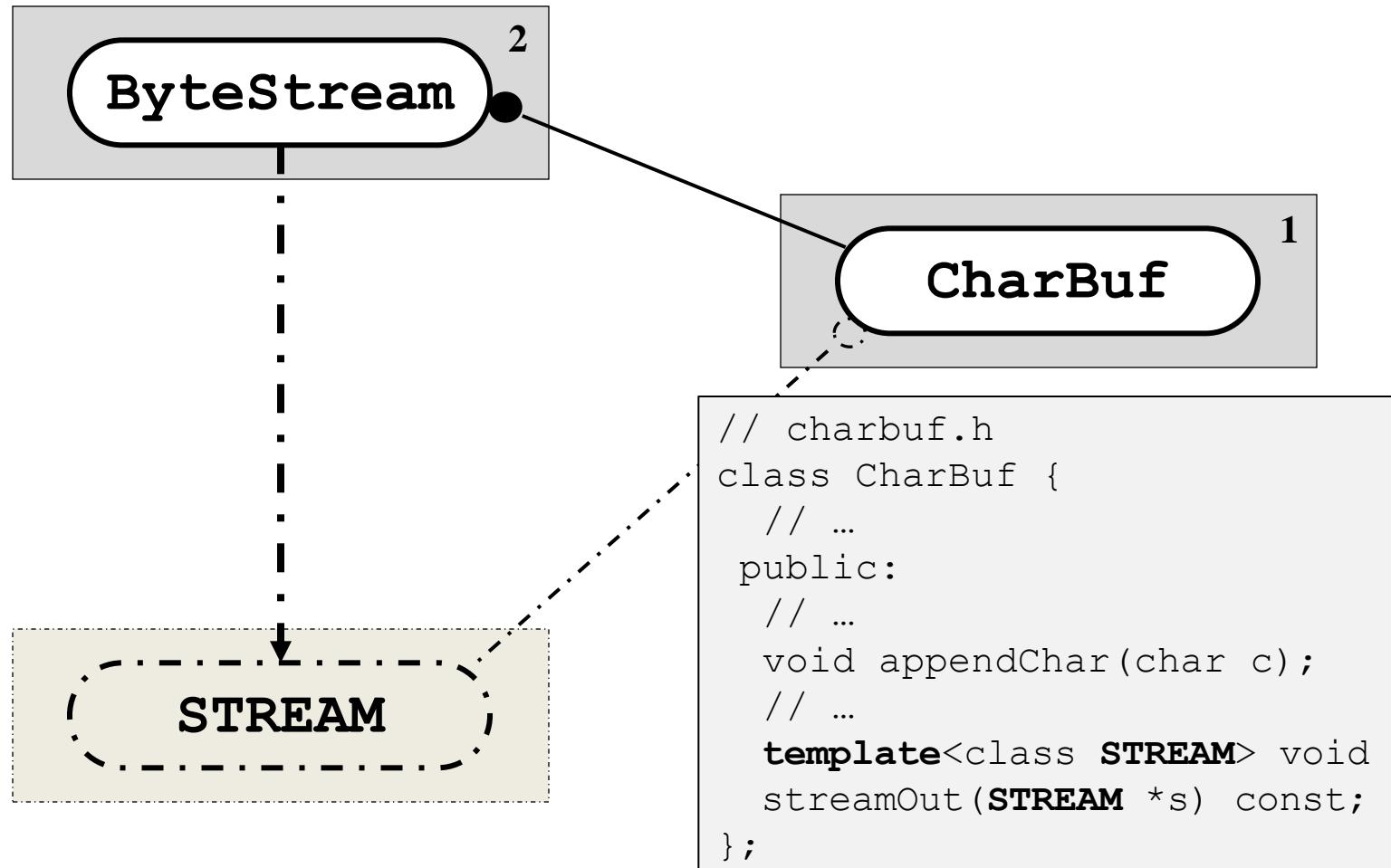
2. Survey of Advanced *Levelization* Techniques

Callbacks: Concept



2. Survey of Advanced *Levelization* Techniques

Callbacks: Concept



2. Survey of Advanced *Levelization* Techniques

Callbacks: Concept

ByteStream

2

CharBuf

1

```
// bytestream.h
#include <charbuf.h>
class ByteStream {
    CharBuf d_streamData;
    // ...
public:
    // ...
    void putInt8(int value);
    void putInt16(int value);
    // ...
    void putFloat32(double value);
    void putFloat64(double value);
    // ...
};
```

```
// charbuf.h
class CharBuf {
    // ...
public:
    // ...
    void appendChar(char c);
    // ...
    template<class STREAM> void
    streamOut(STREAM *s) const;
};
```

2. Survey of Advanced *Levelization* Techniques

Callbacks: Concept

ByteStream

2

CharBuf

1

```
// bytestream.h
#include <charbuf.h>
class ByteStream {
    CharBuf d_streamData;
    // ...
public:
    // ...
    void putInt8(int value);
    void putInt16(int value);
    // ...
    void putFloat32(double value);
    void putFloat64(double value);
};
```

```
// charbuf.h
class CharBuf {
    // ...
public:
    // ...
    void appendChar(char c);
    // ...
    template<class STREAM> void
    streamOut(STREAM *s) const;
};
```

charBuf is
Externalizable!

2. Survey of Advanced *Levelization* Techniques

Callbacks

Discussion?

2. Survey of Advanced *Levelization* Techniques

Manager Class

Manager Class –

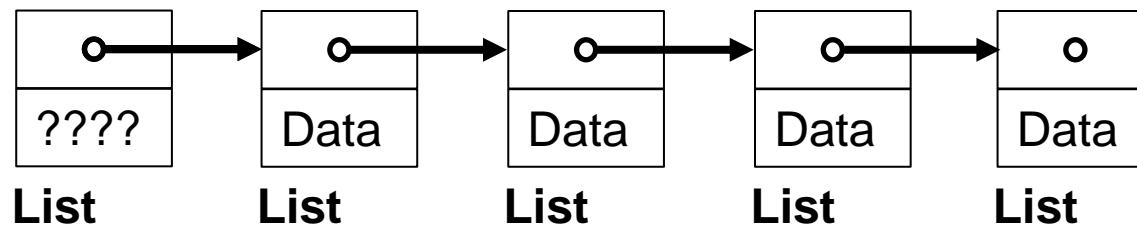
Establishing a class that owns and coordinates lower-level objects.

2. Survey of Advanced *Levelization* Techniques

Manager Class

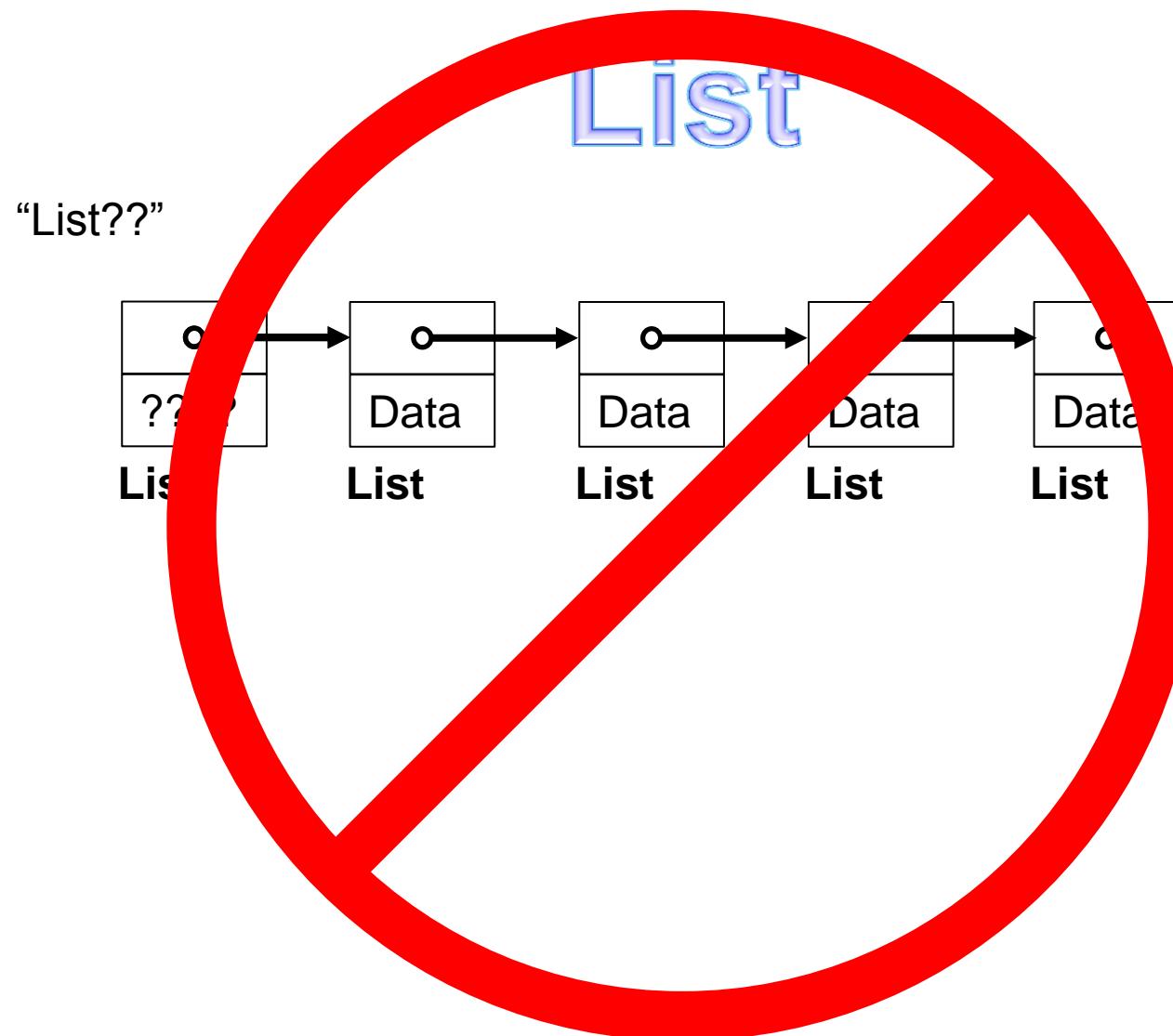
List

“List??”



2. Survey of Advanced *Levelization* Techniques

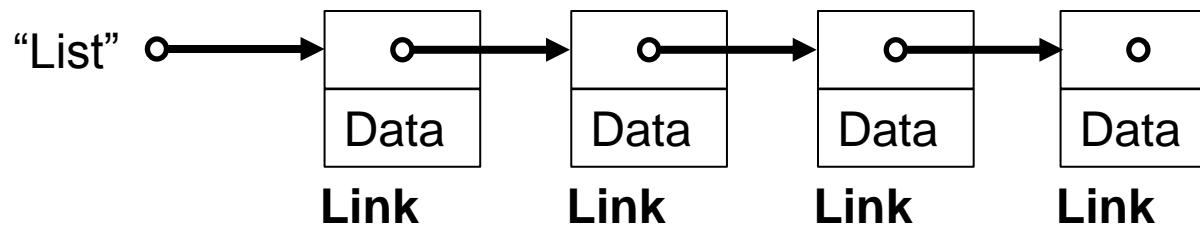
Manager Class



2. Survey of Advanced *Levelization* Techniques

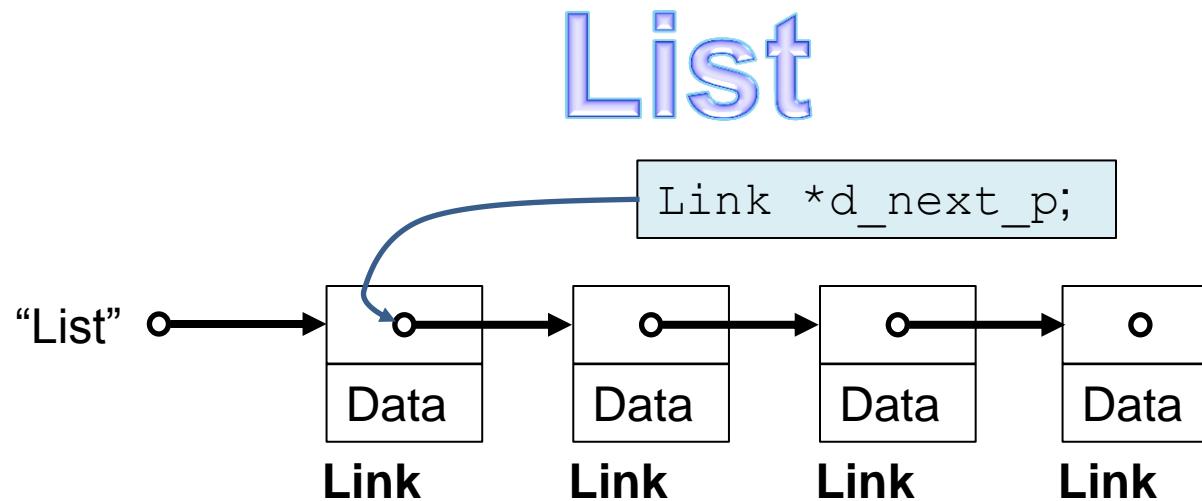
Manager Class

List



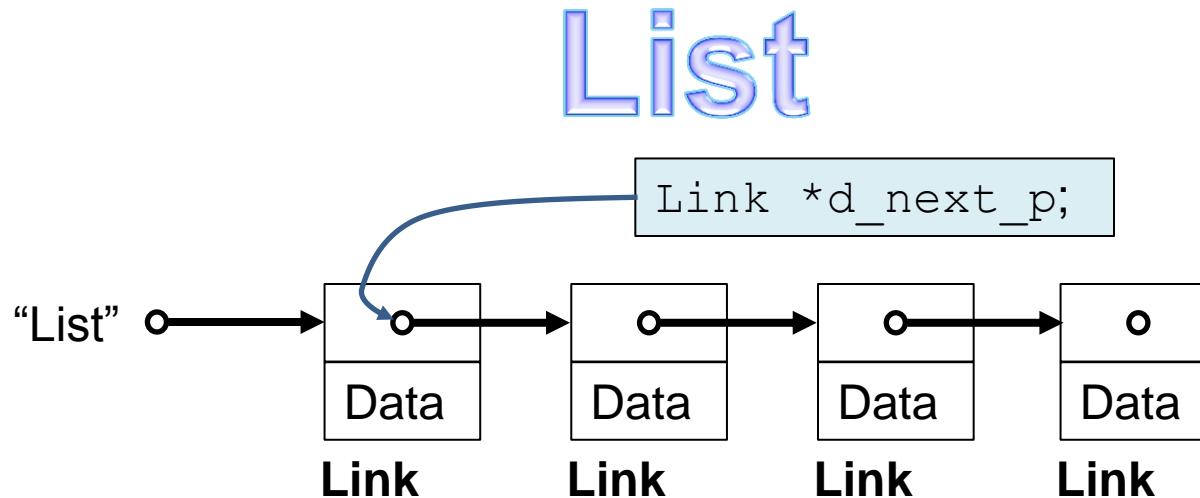
2. Survey of Advanced *Levelization* Techniques

Manager Class



2. Survey of Advanced *Levelization* Techniques

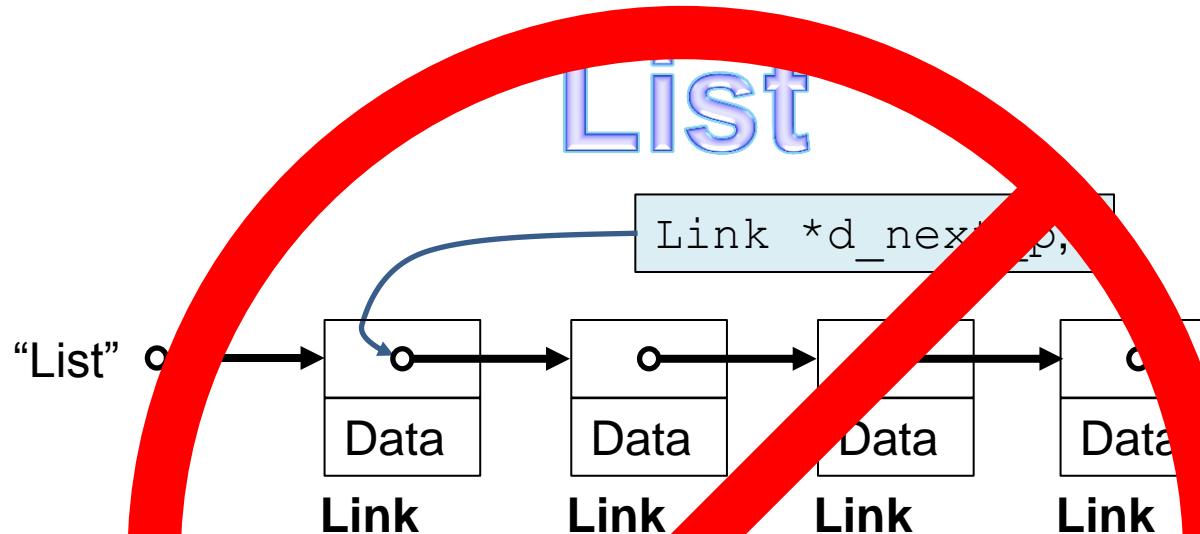
Manager Class



```
// link.cpp
// ...
Link::~Link()
{
    delete d_next_p;
}
```

2. Survey of Advanced *Levelization* Techniques

Manager Class

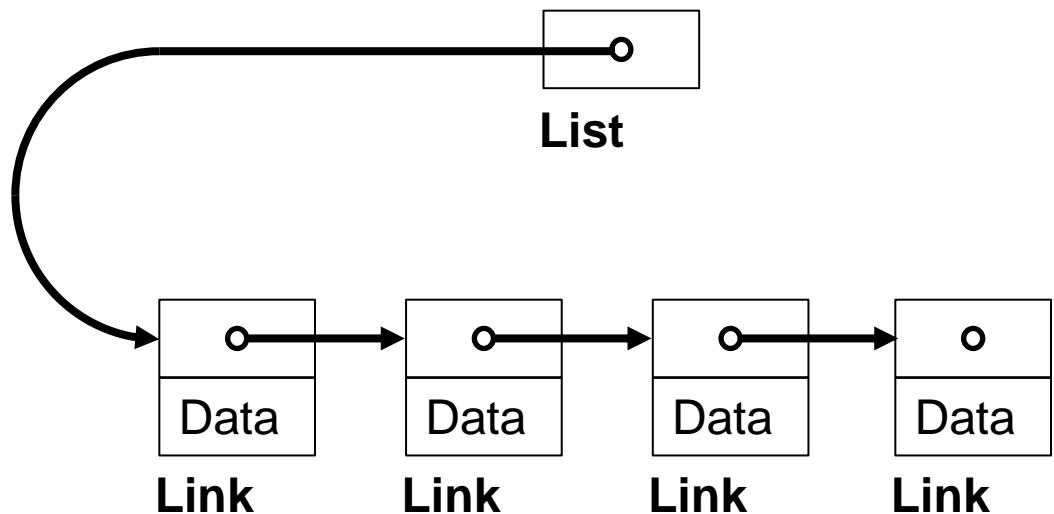


```
// link.cpp
// ...
Link::~Link()
{
    delete d_ex;
}
```

2. Survey of Advanced *Levelization* Techniques

Manager Class

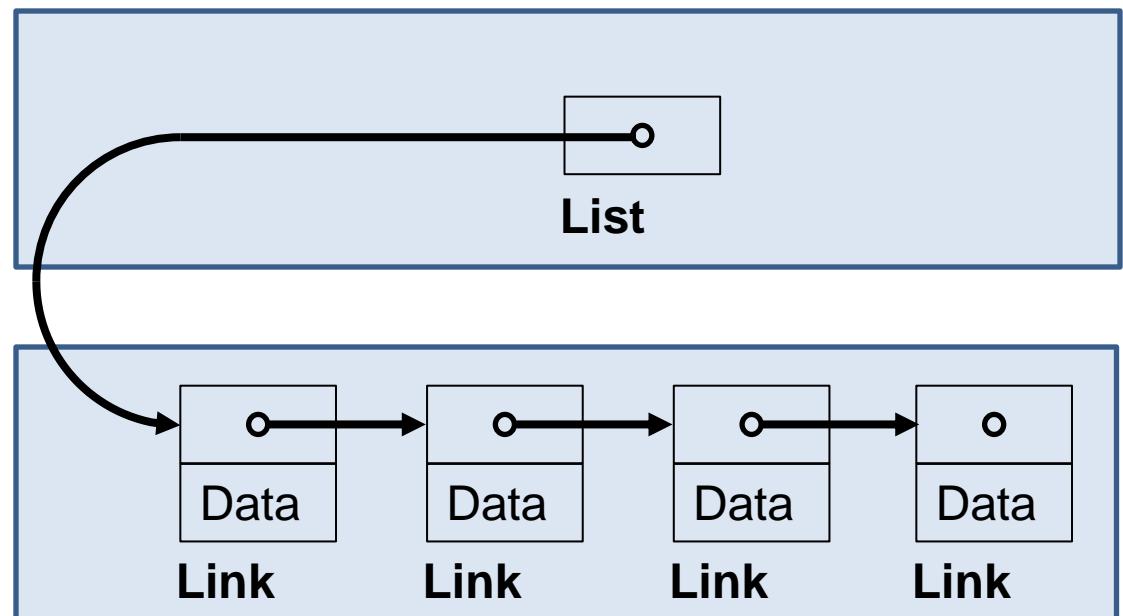
List



2. Survey of Advanced *Levelization* Techniques

Manager Class

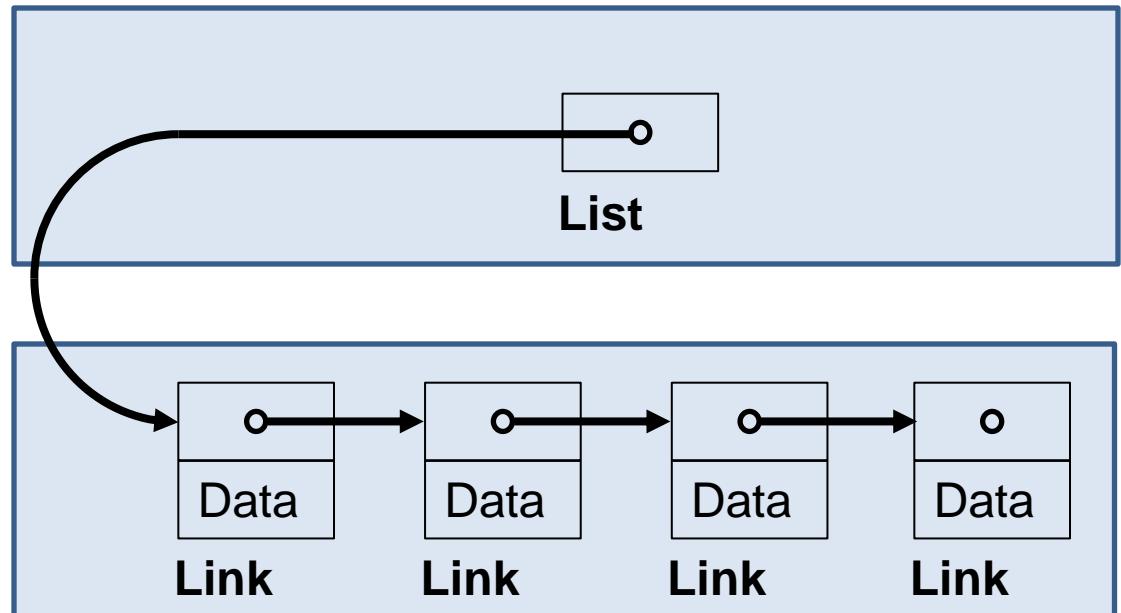
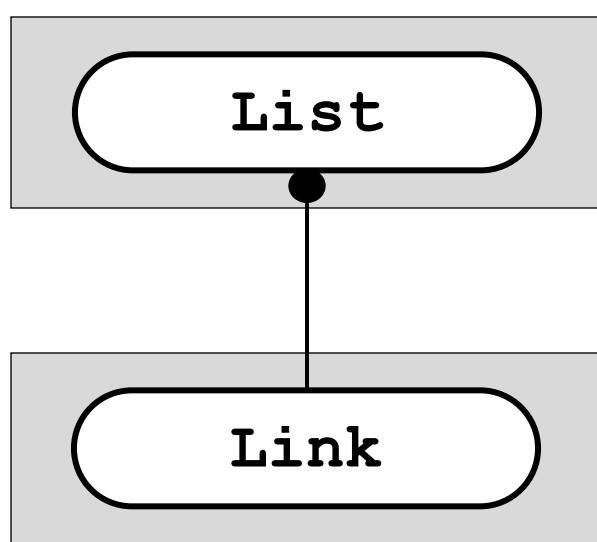
List



2. Survey of Advanced *Levelization* Techniques

Manager Class

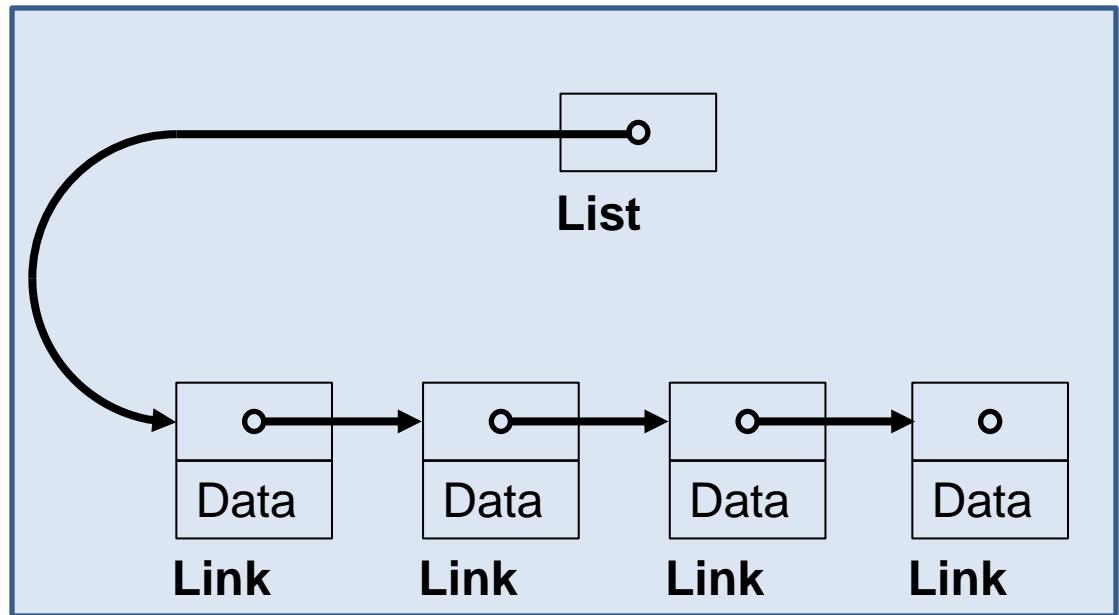
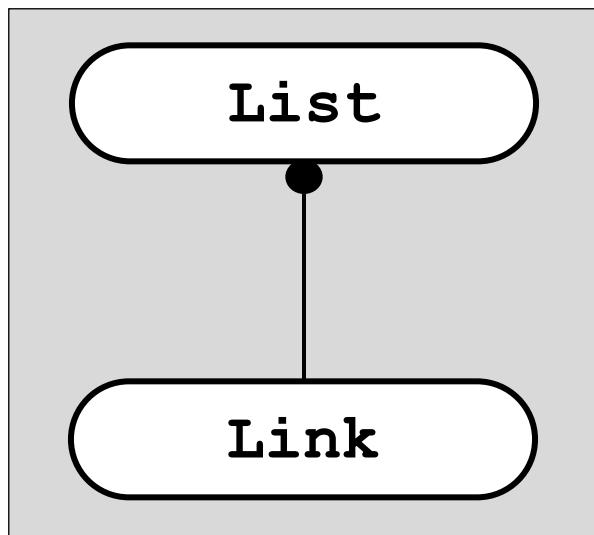
List



2. Survey of Advanced *Levelization* Techniques

Manager Class

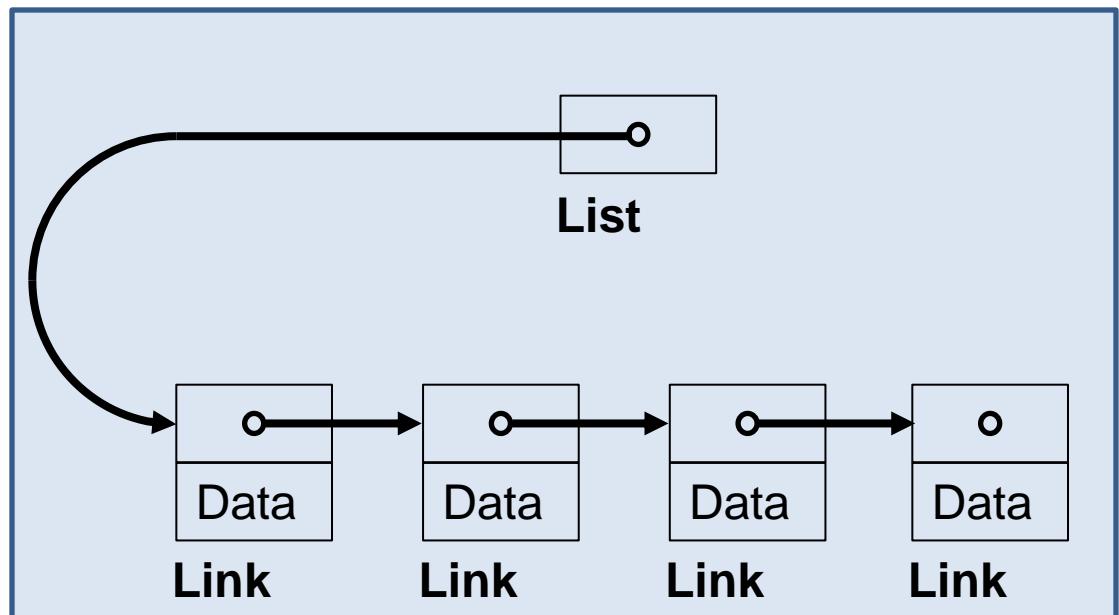
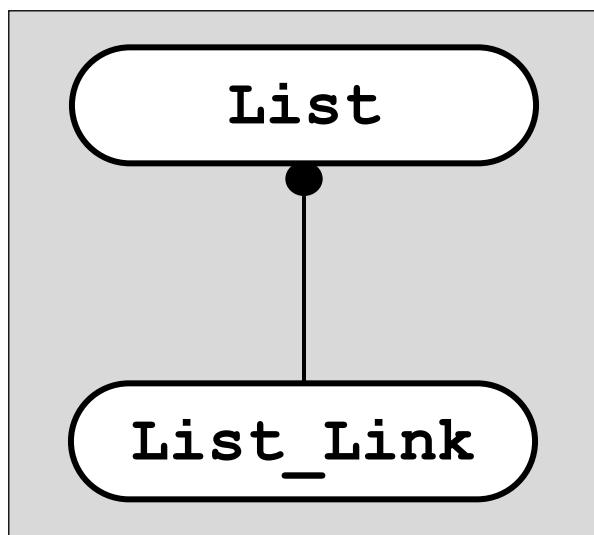
List



2. Survey of Advanced *Levelization* Techniques

Manager Class

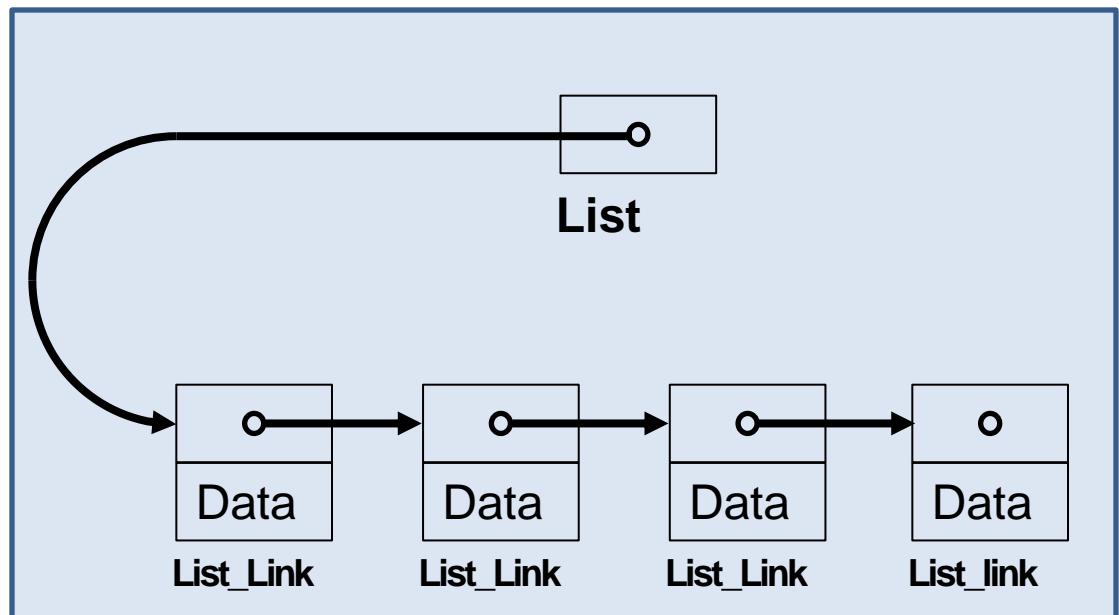
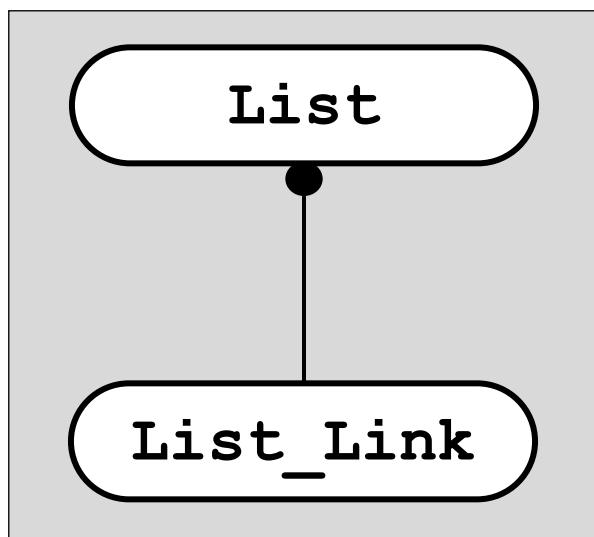
List



2. Survey of Advanced *Levelization* Techniques

Manager Class

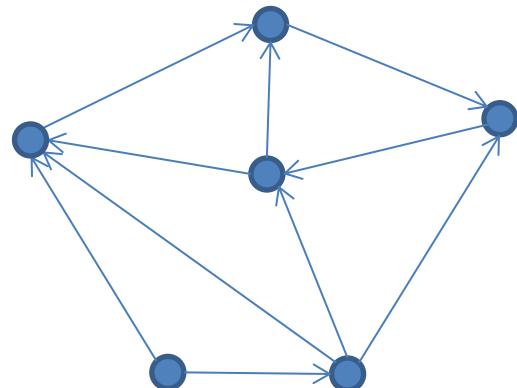
List



2. Survey of Advanced *Levelization* Techniques

Manager Class

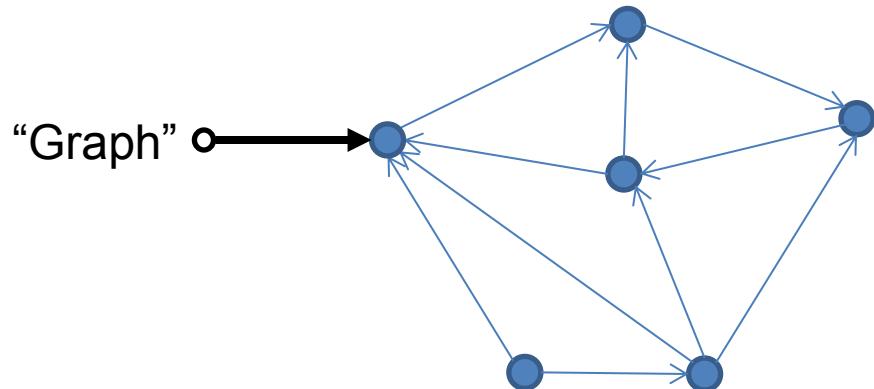
Graph



2. Survey of Advanced *Levelization* Techniques

Manager Class

Graph

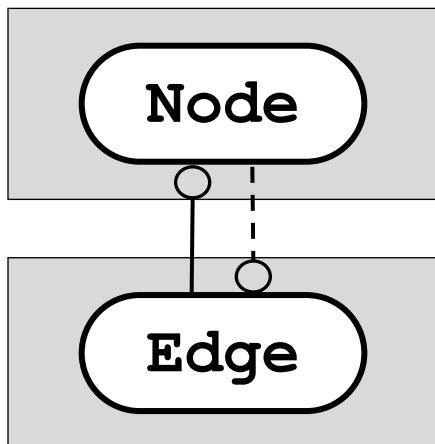
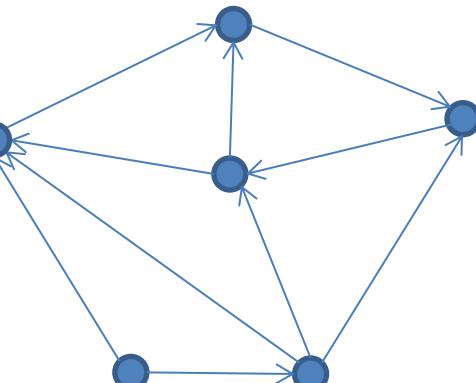


2. Survey of Advanced *Levelization* Techniques

Manager Class

Graph

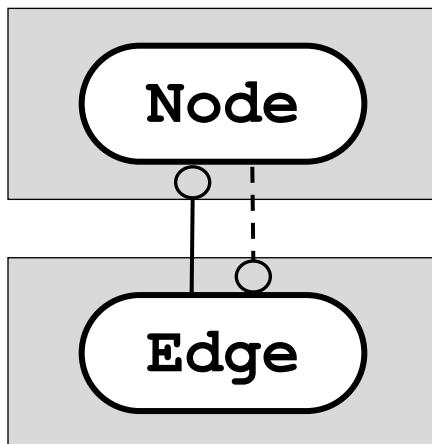
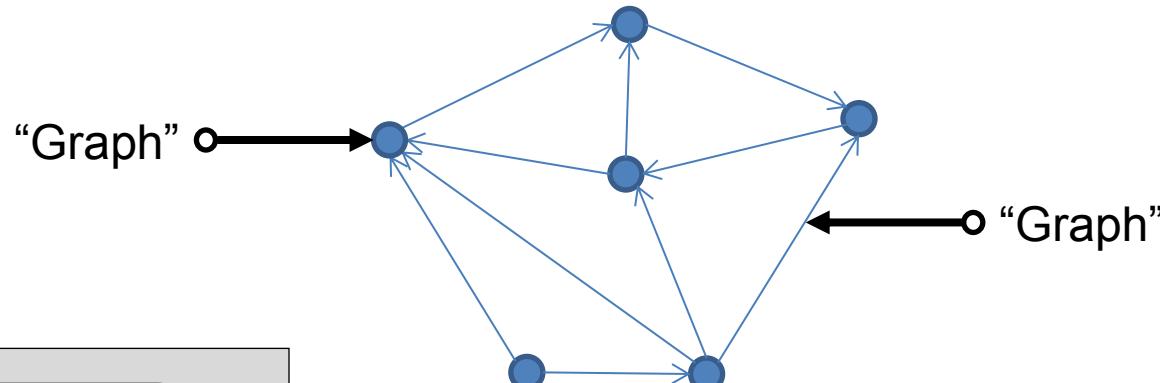
“Graph” o



2. Survey of Advanced *Levelization* Techniques

Manager Class

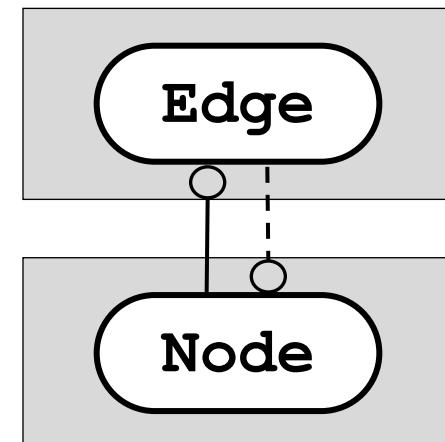
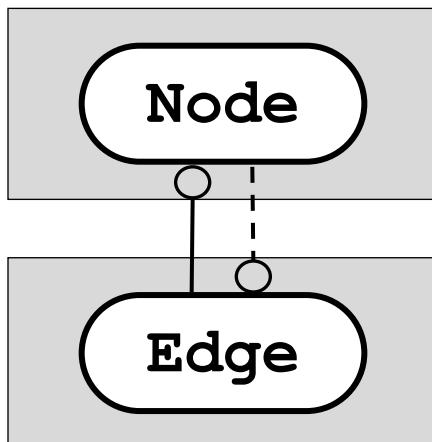
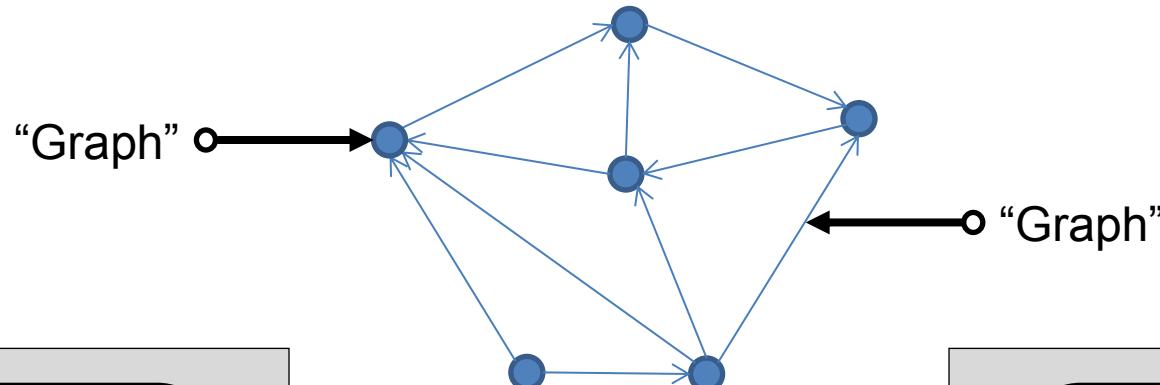
Graph



2. Survey of Advanced *Levelization* Techniques

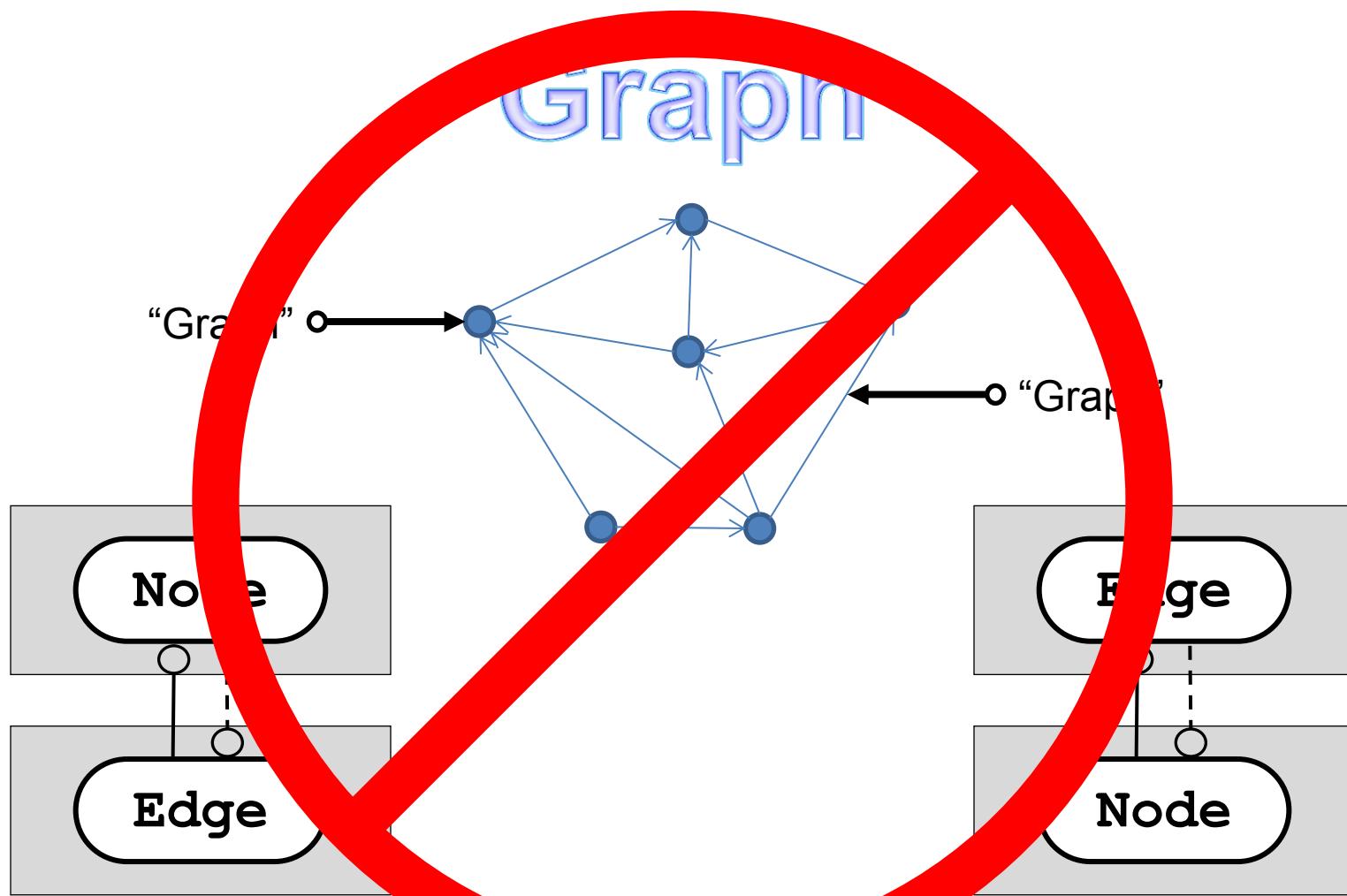
Manager Class

Graph



2. Survey of Advanced *Levelization* Techniques

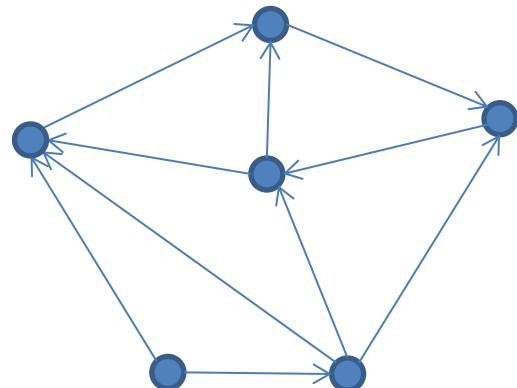
Manager Class



2. Survey of Advanced *Levelization* Techniques

Manager Class

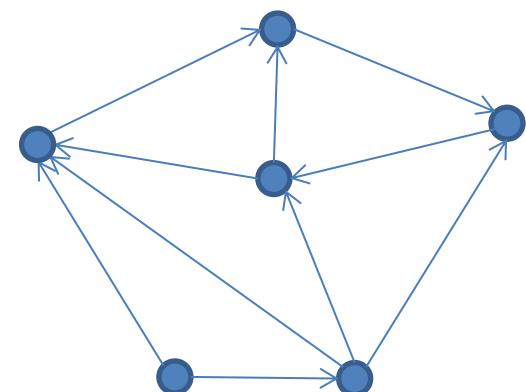
Graph



2. Survey of Advanced *Levelization* Techniques

Manager Class

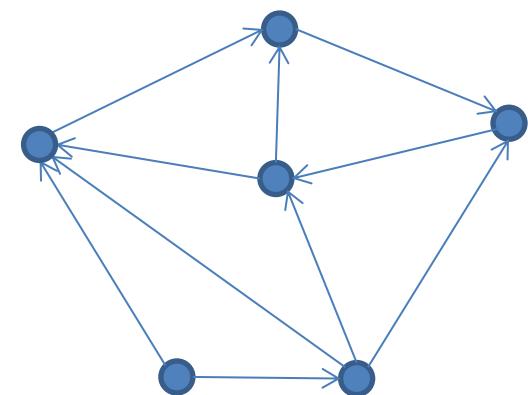
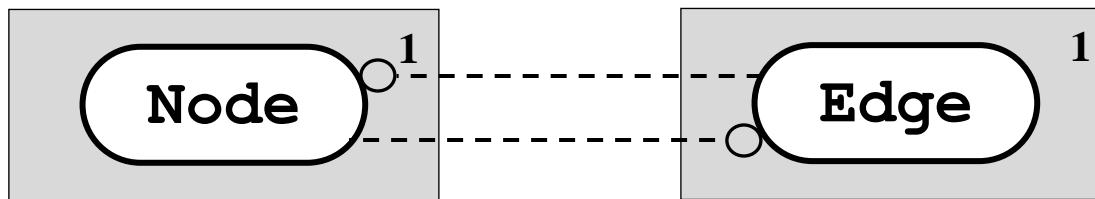
Graph



2. Survey of Advanced *Levelization* Techniques

Manager Class

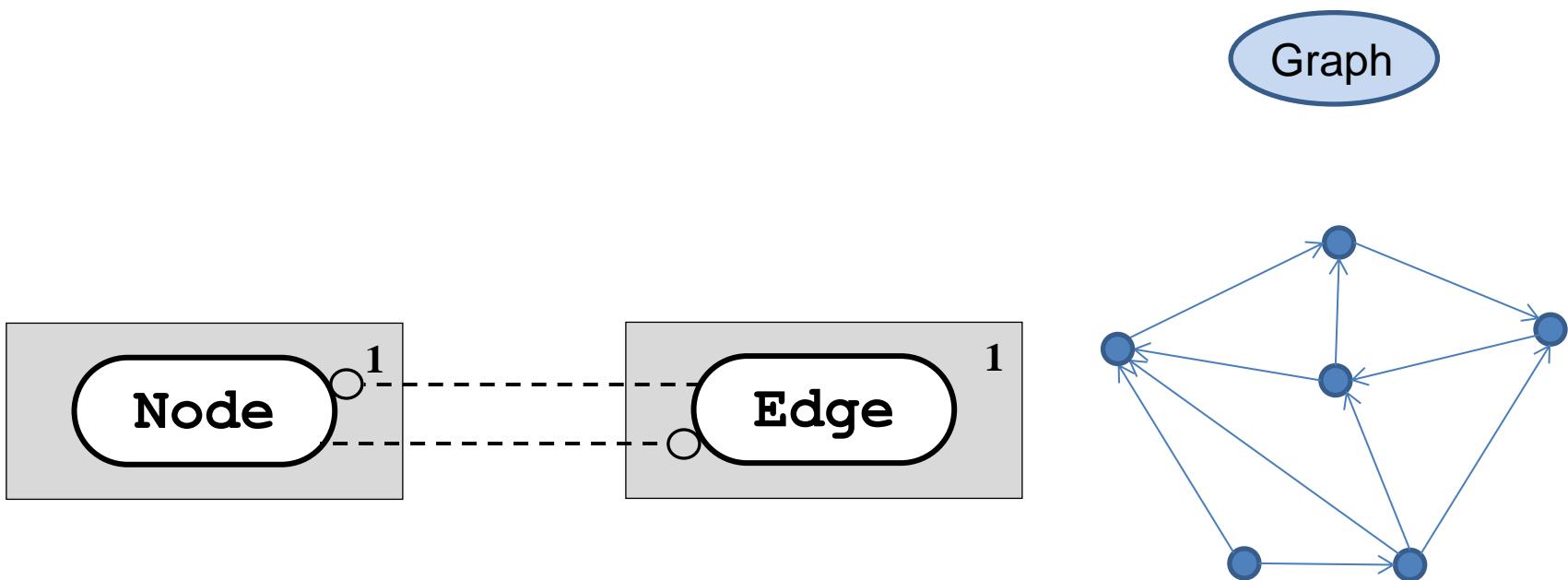
Graph



2. Survey of Advanced *Levelization* Techniques

Manager Class

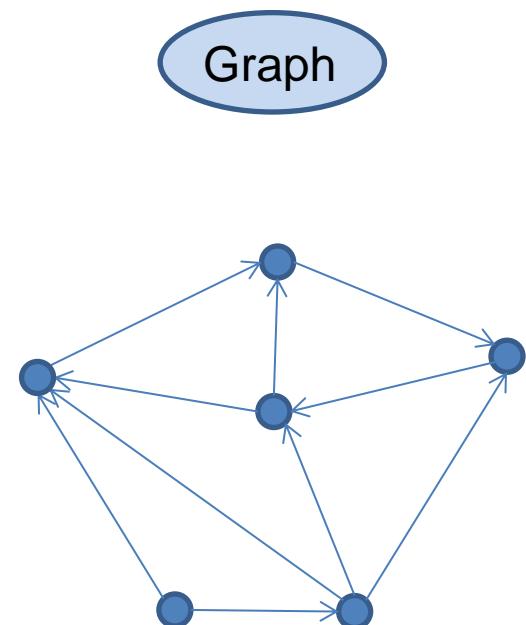
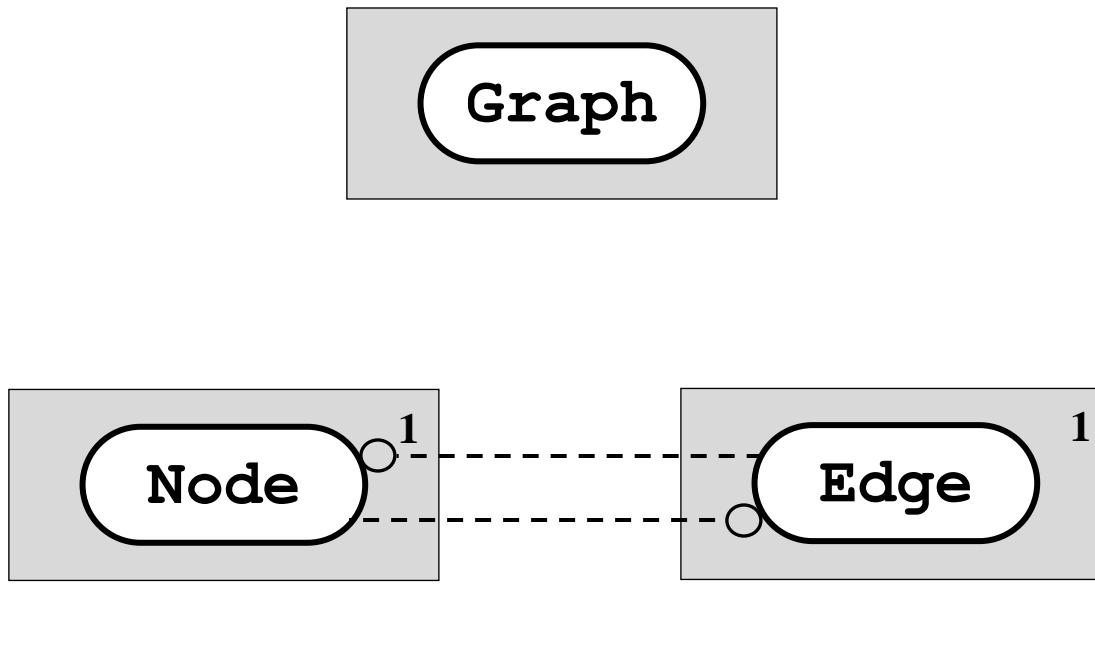
Graph



2. Survey of Advanced *Levelization* Techniques

Manager Class

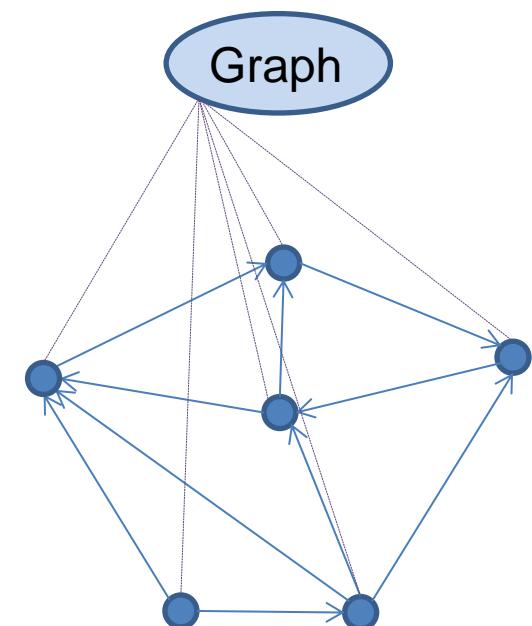
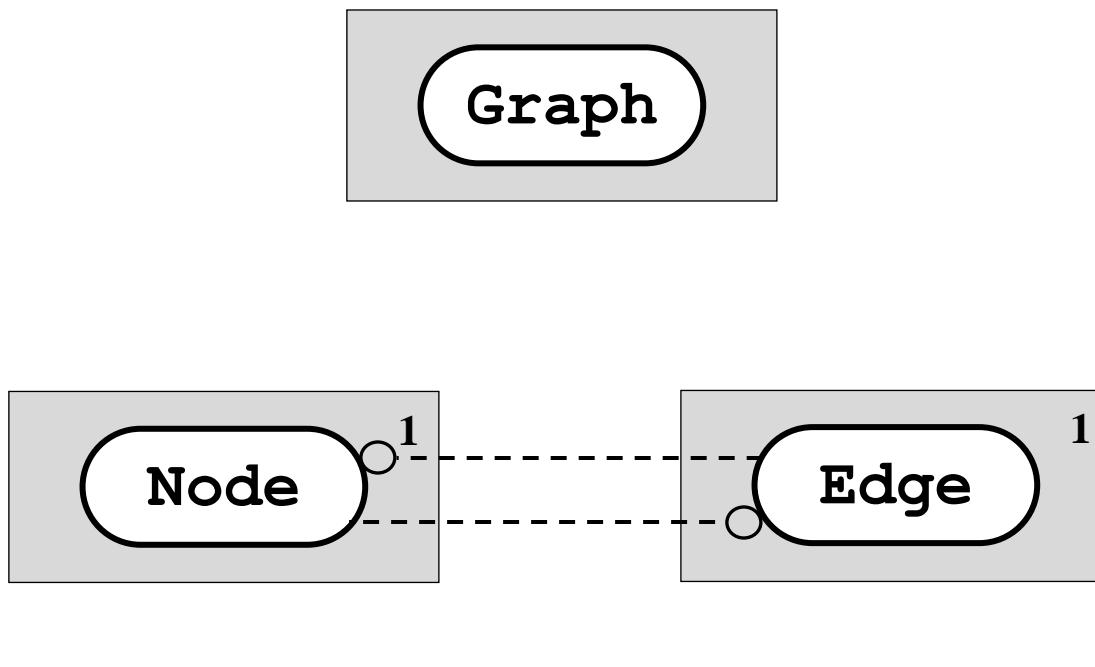
Graph



2. Survey of Advanced *Levelization* Techniques

Manager Class

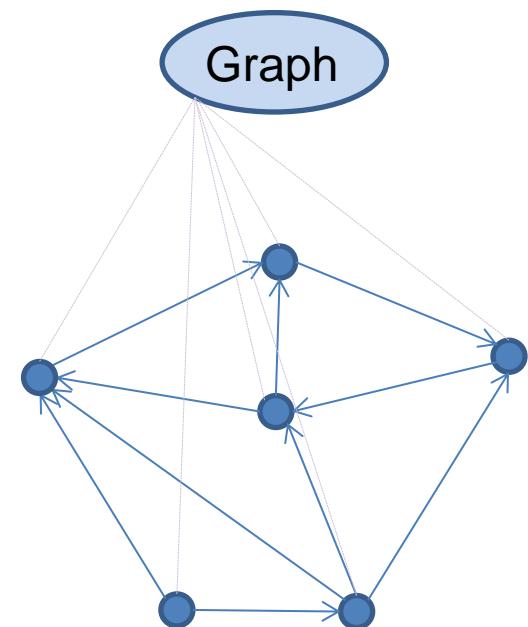
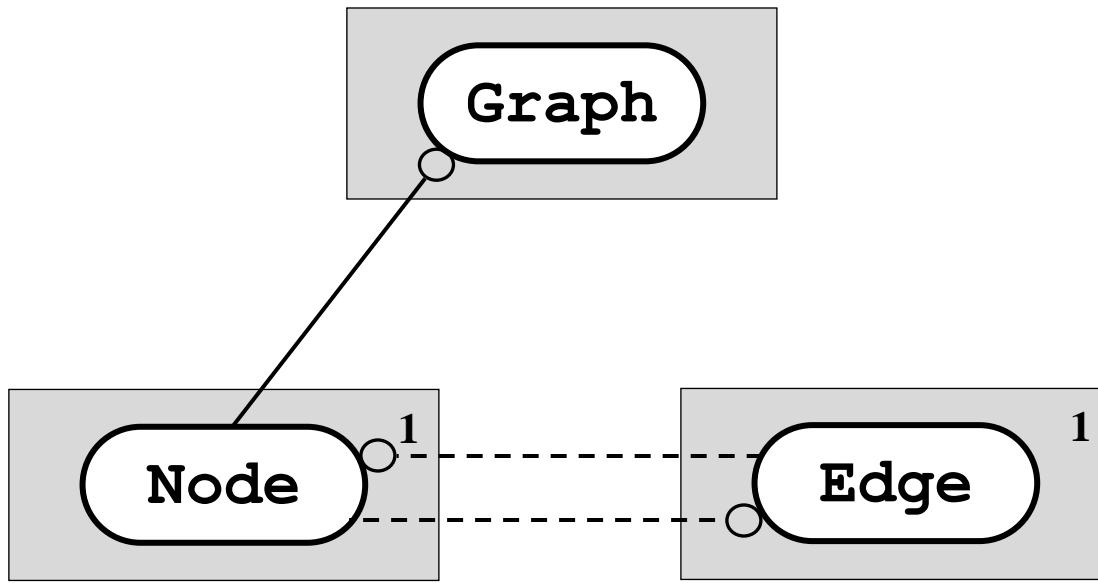
Graph



2. Survey of Advanced *Levelization* Techniques

Manager Class

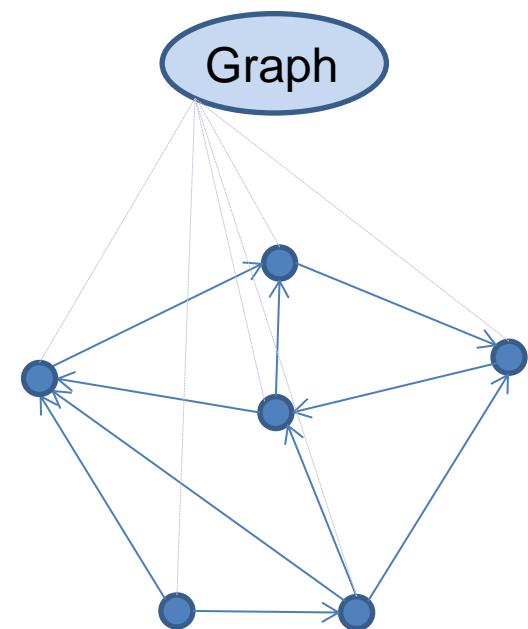
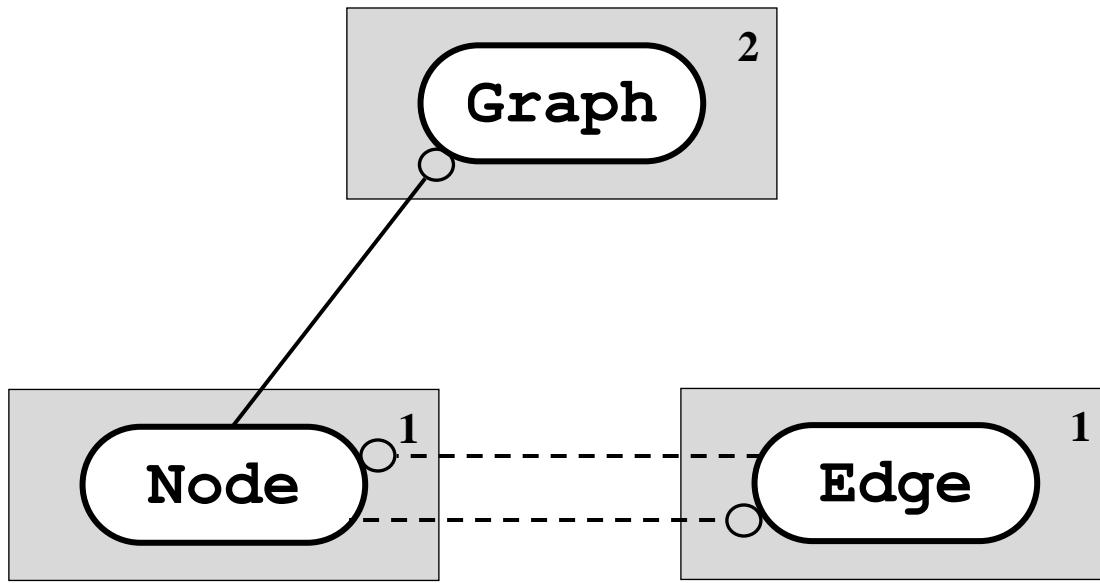
Graph



2. Survey of Advanced *Levelization* Techniques

Manager Class

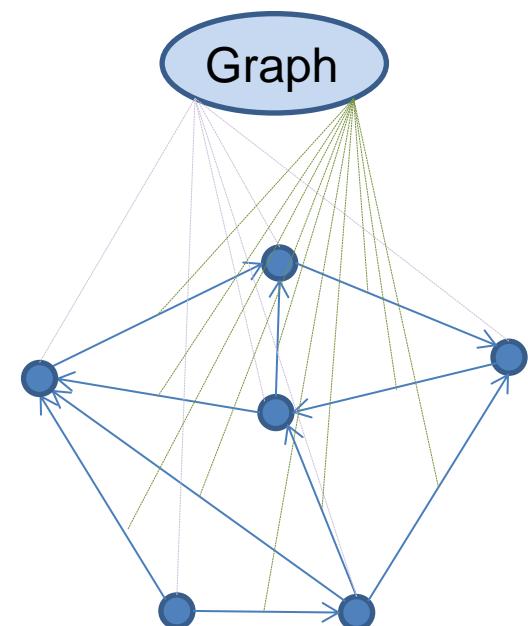
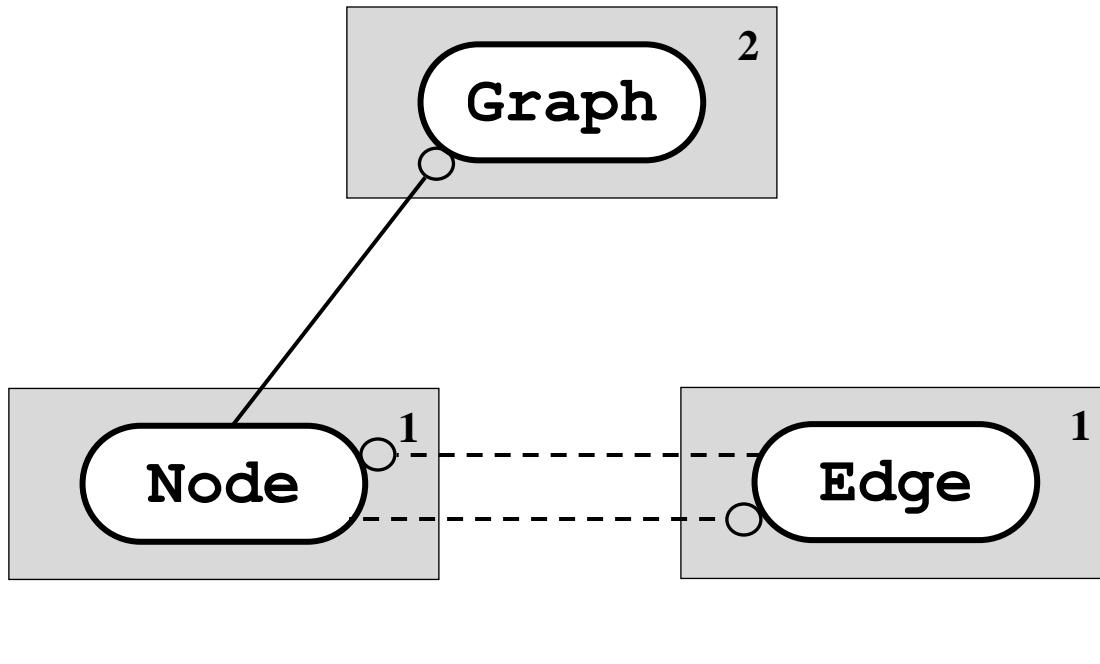
Graph



2. Survey of Advanced *Levelization* Techniques

Manager Class

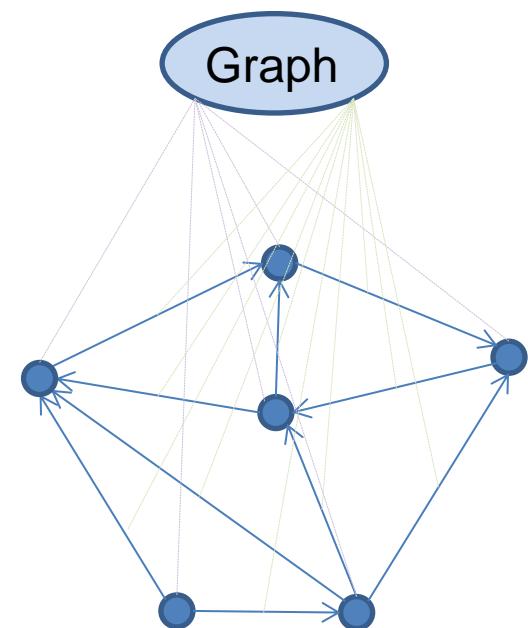
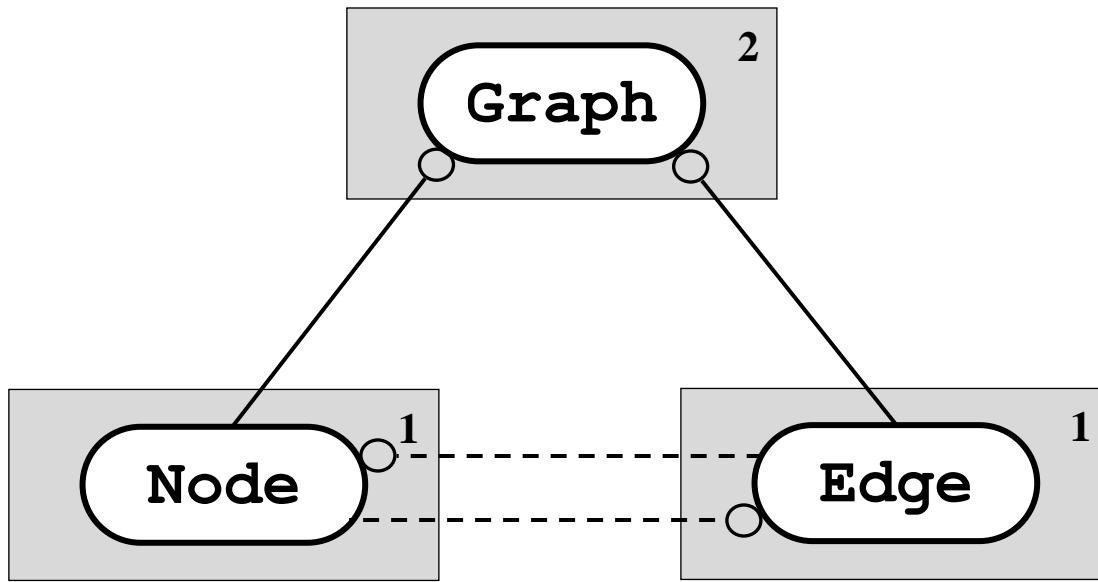
Graph



2. Survey of Advanced *Levelization* Techniques

Manager Class

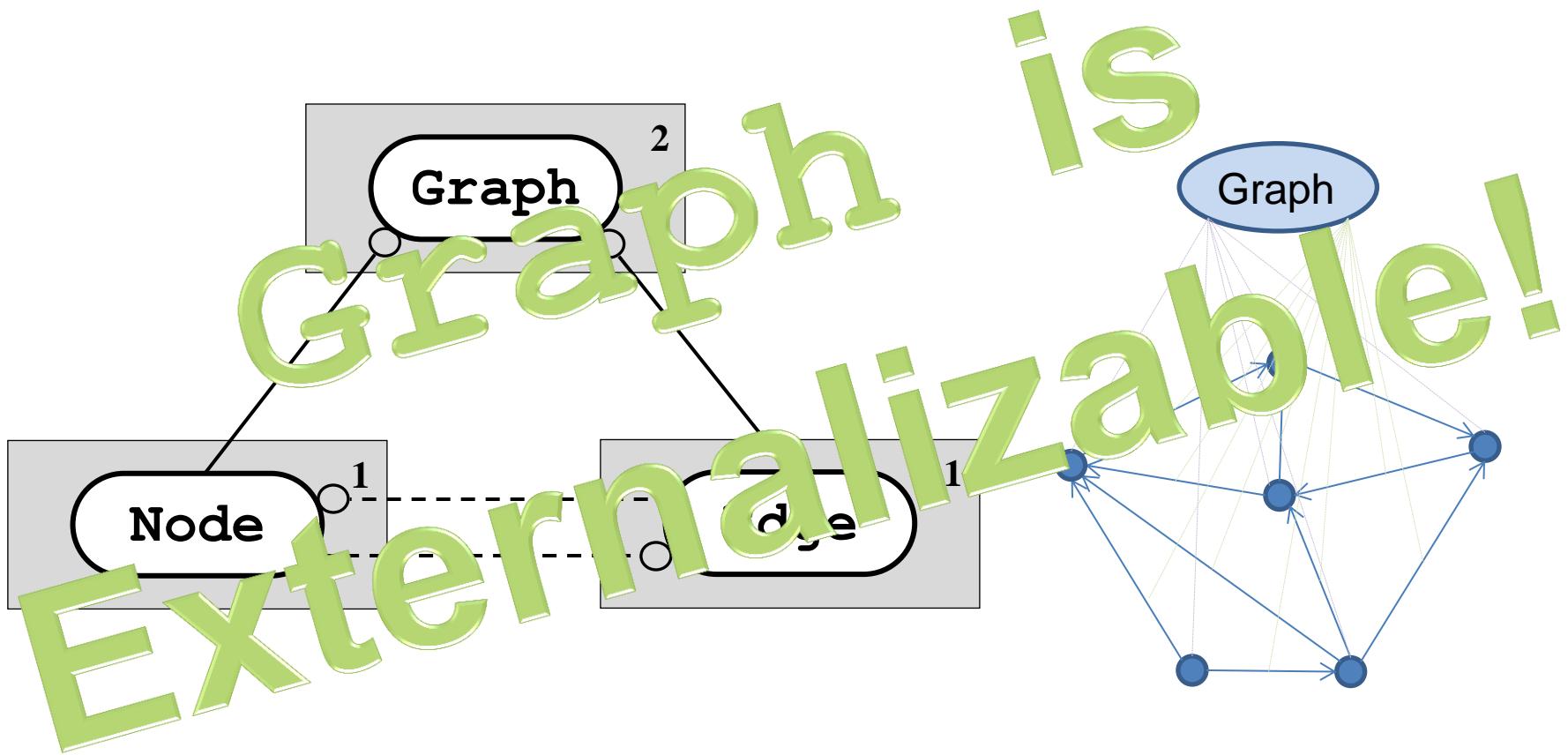
Graph



2. Survey of Advanced *Levelization* Techniques

Manager Class

Graph



2. Survey of Advanced *Levelization* Techniques
Manager Class

Discussion?

2. Survey of Advanced *Levelization* Techniques

Factoring

Factoring – Moving independently testable sub-behavior out of the implementation of a complex component involved in excessive physical coupling.

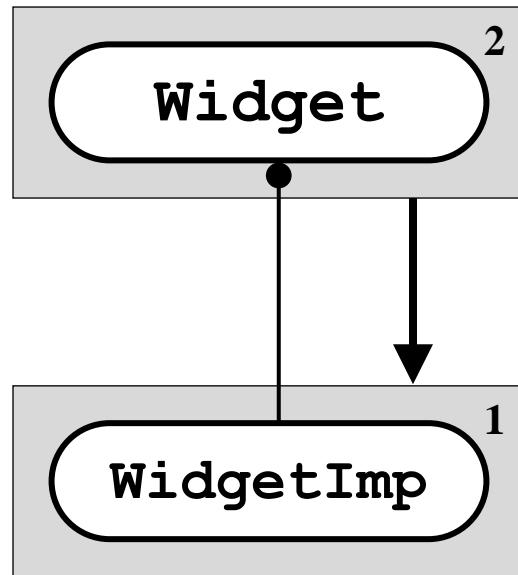
2. Survey of Advanced *Levelization* Techniques

Factoring



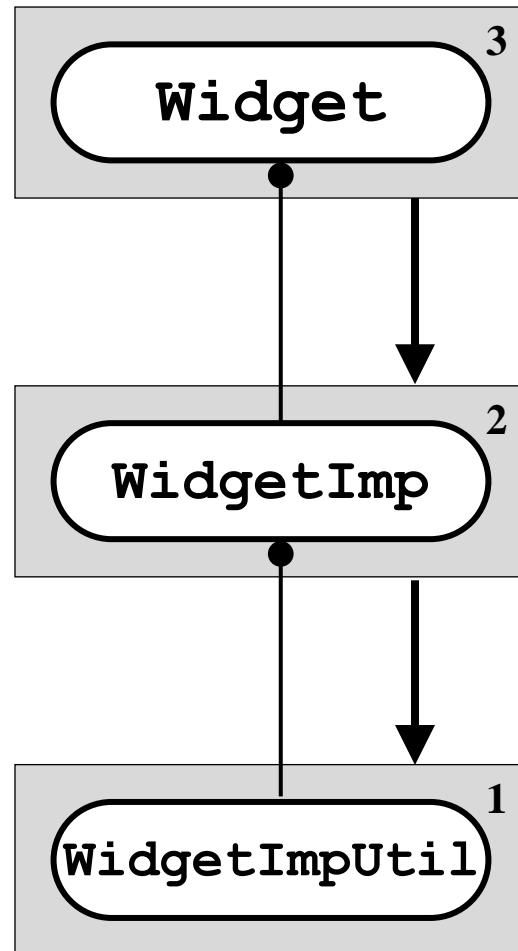
2. Survey of Advanced *Levelization* Techniques

Factoring



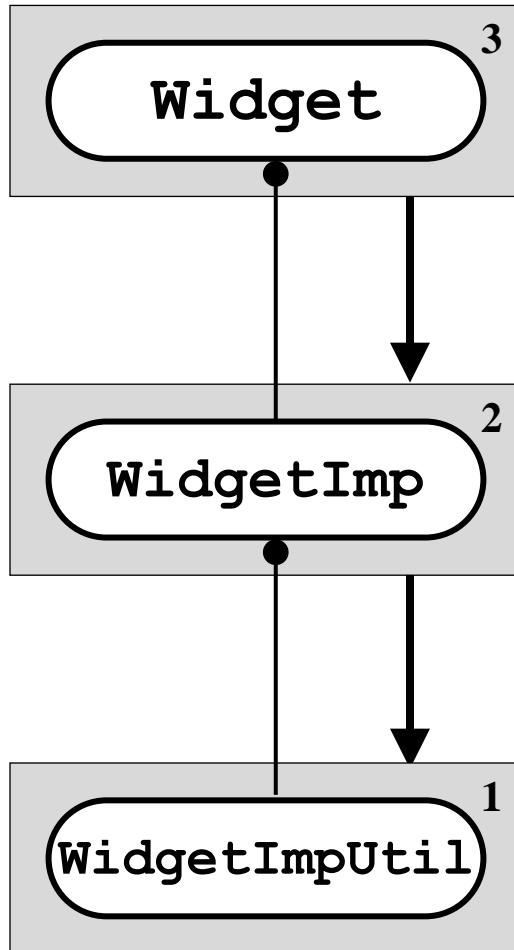
2. Survey of Advanced *Levelization* Techniques

Factoring



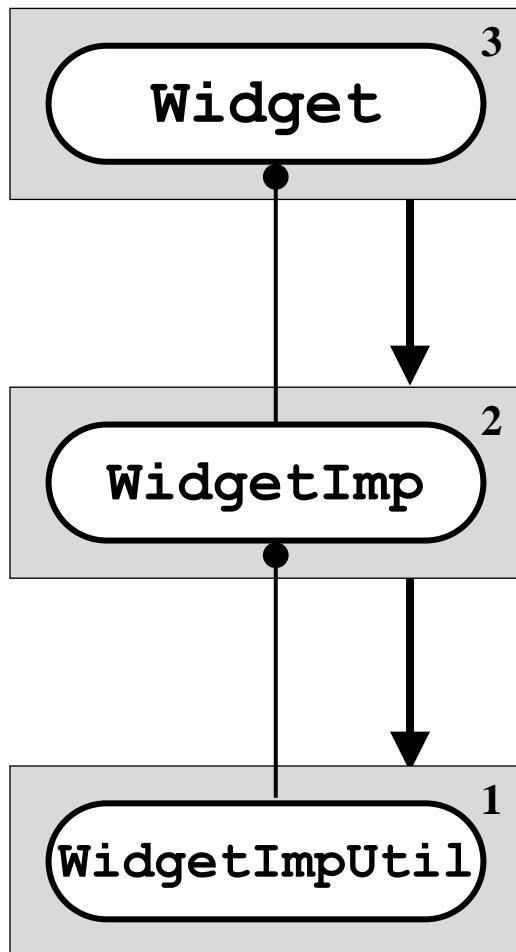
2. Survey of Advanced *Levelization* Techniques

Factoring



2. Survey of Advanced *Levelization* Techniques

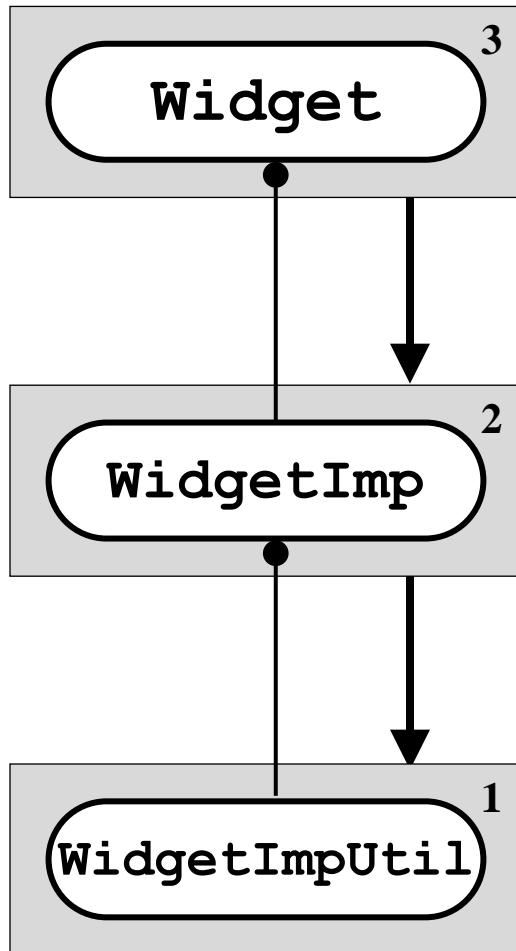
Factoring



Architecturally Significant
Client-Facing Interface!

2. Survey of Advanced *Levelization* Techniques

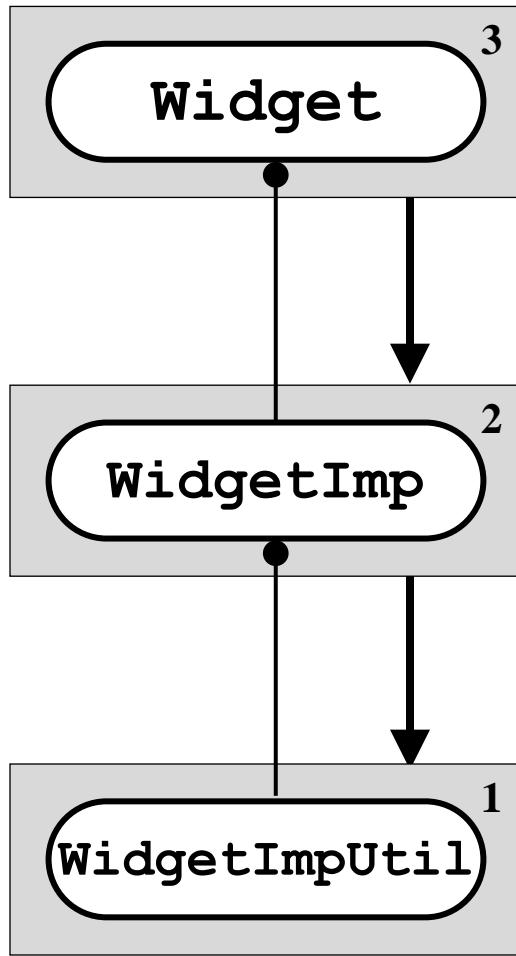
Factoring



- { Architecturally Significant Client-Facing Interface!
- { Private methods above become public ones in this class as sole data member.

2. Survey of Advanced *Levelization* Techniques

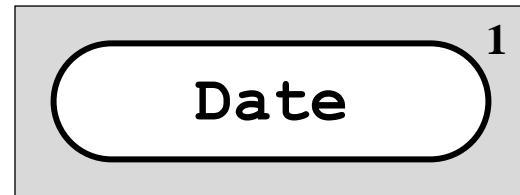
Factoring



- { Architecturally Significant Client-Facing Interface!
- { Private methods above become public ones in this class as sole data member.
- { File-scope static methods in .cpp file are made public in separate utility struct.
405

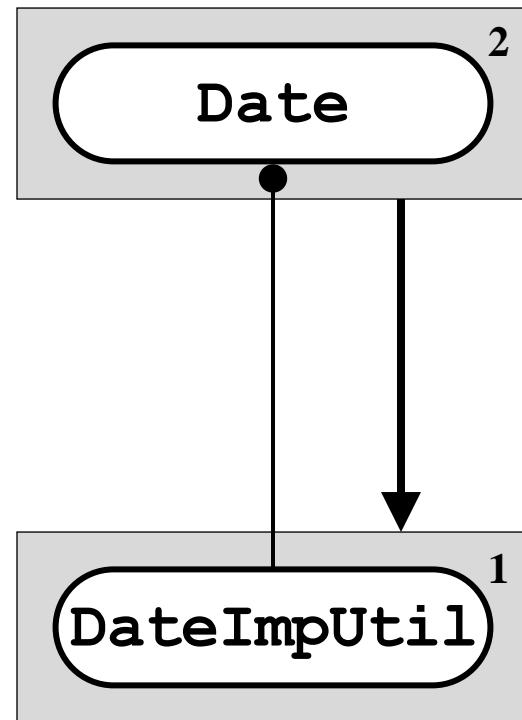
2. Survey of Advanced *Levelization* Techniques

Factoring



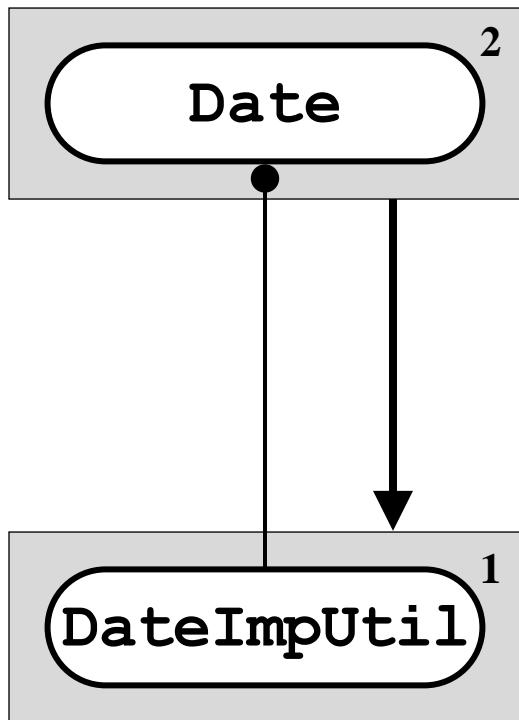
2. Survey of Advanced *Levelization* Techniques

Factoring



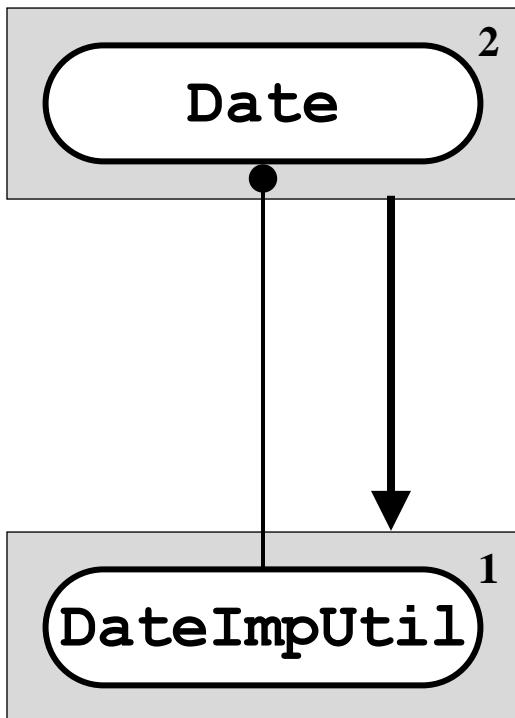
2. Survey of Advanced *Levelization* Techniques

Factoring



2. Survey of Advanced *Levelization* Techniques

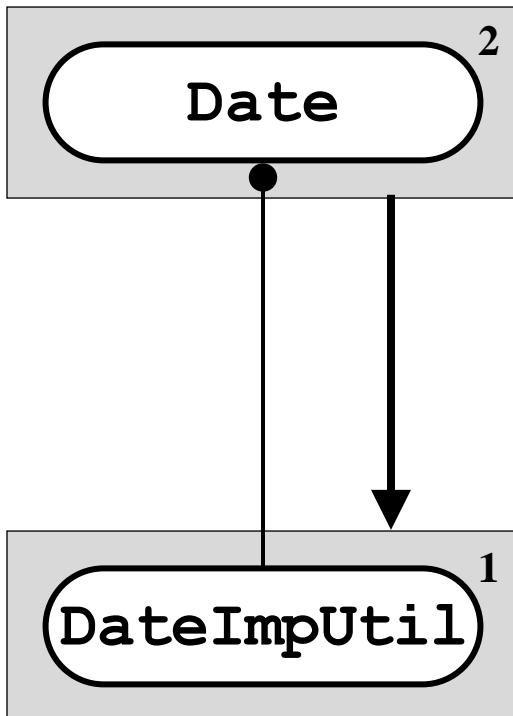
Factoring



```
// date.h
#include <dateimputil.h>
class Date {
    int d_serialDate;
public:
    Date(int y, int m, int d);
    // ...
    void setYmd(int y, int m, int d);
    int setYmdIfValid(int y, int m, int d);
    // ...
    bool isLeapyear() const;
};
```

2. Survey of Advanced *Levelization* Techniques

Factoring



```
// date.h
#include <dateimputil.h>
class Date {
    int d_serialDate;
public:
    Date(int y, int m, int d);
    // ...
    void setYmd(int y, int m, int d);
    int setYmdIfValid(int y, int m, int d);
    // ...
    bool isLeapyear() const;
};
```

```
// dateimputil.h
// ...
struct DateImpUtil {
    static int toSerialDate(int y, int m, int d);
    // ...
    static bool isLeapYear(int y);
};
```

2. Survey of Advanced *Levelization* Techniques

Factoring

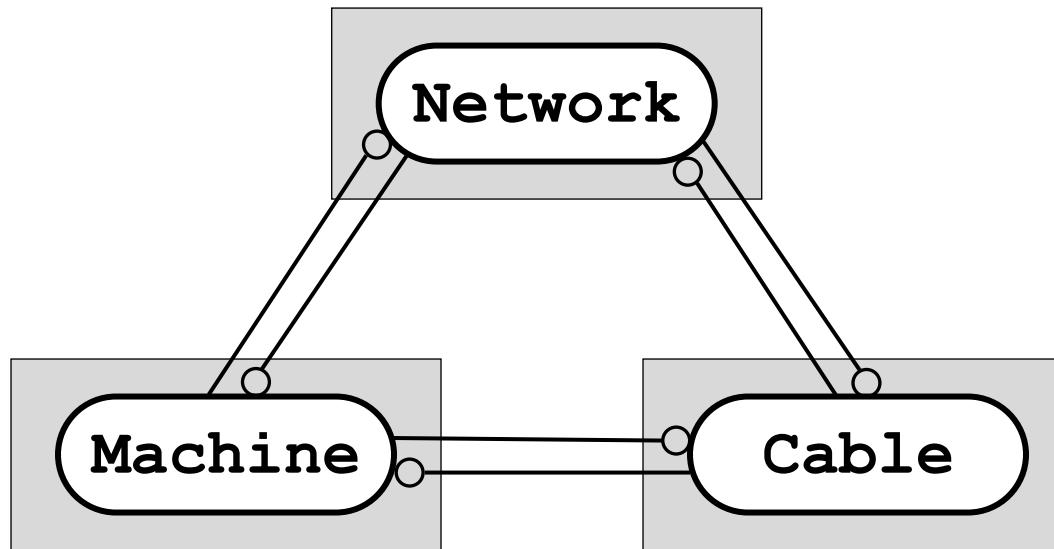
Network

Machine

Cable

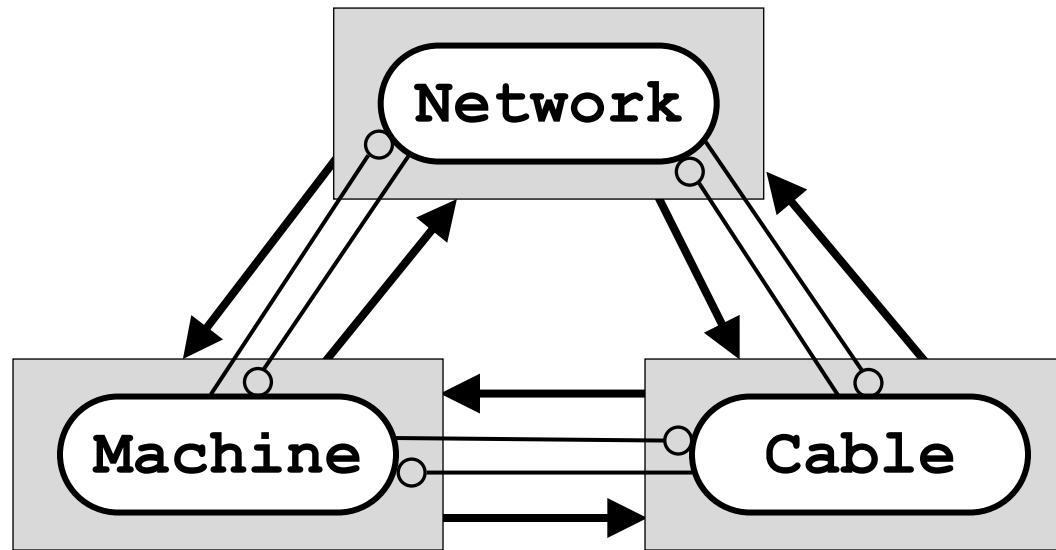
2. Survey of Advanced *Levelization* Techniques

Factoring



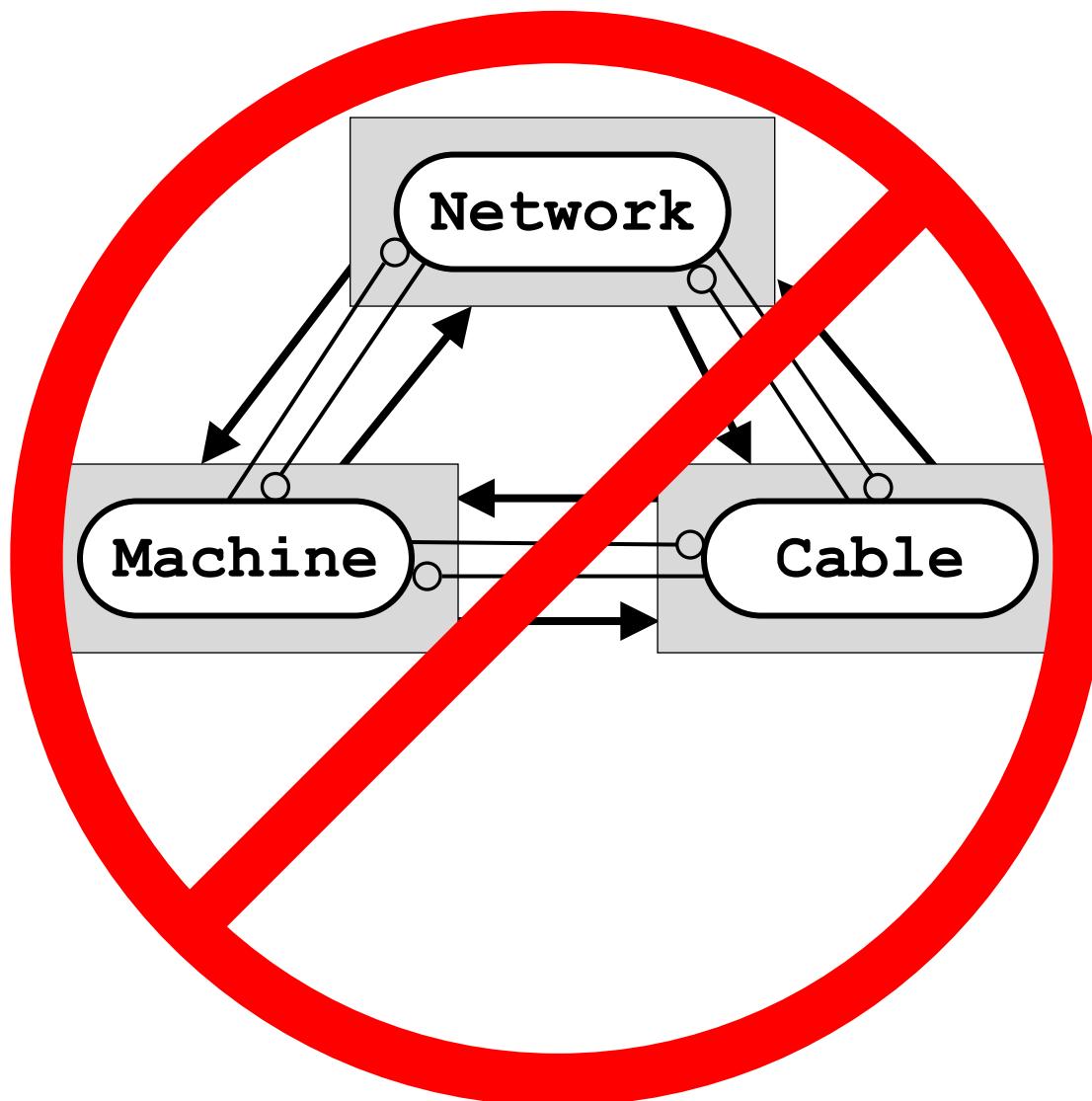
2. Survey of Advanced *Levelization* Techniques

Factoring



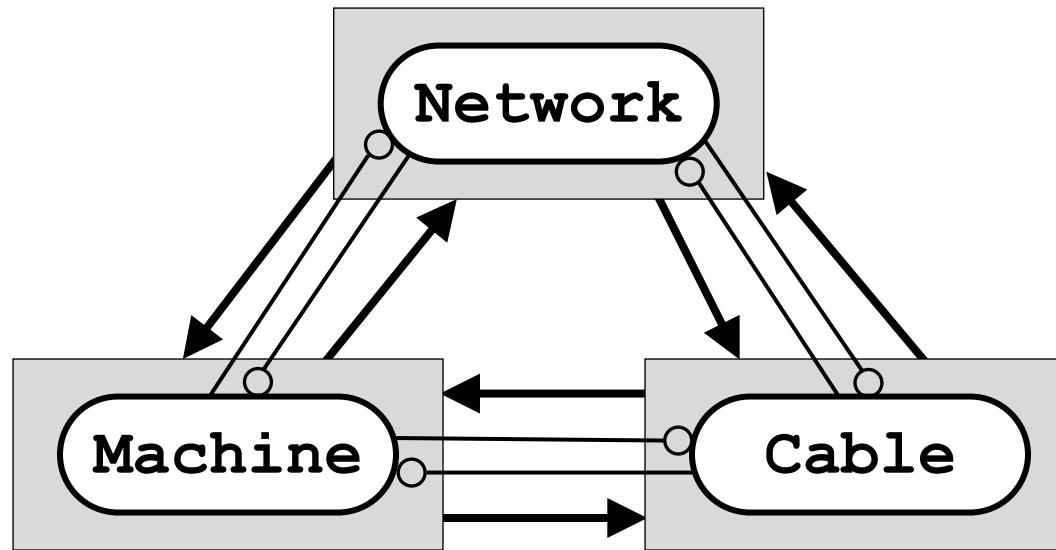
2. Survey of Advanced *Levelization* Techniques

Factoring



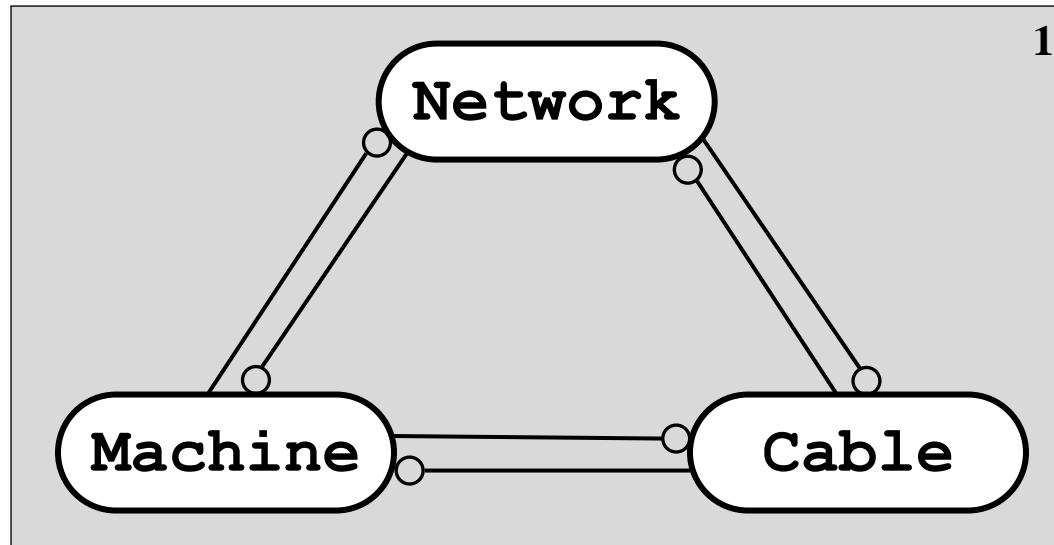
2. Survey of Advanced *Levelization* Techniques

Factoring



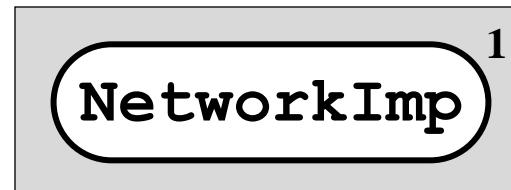
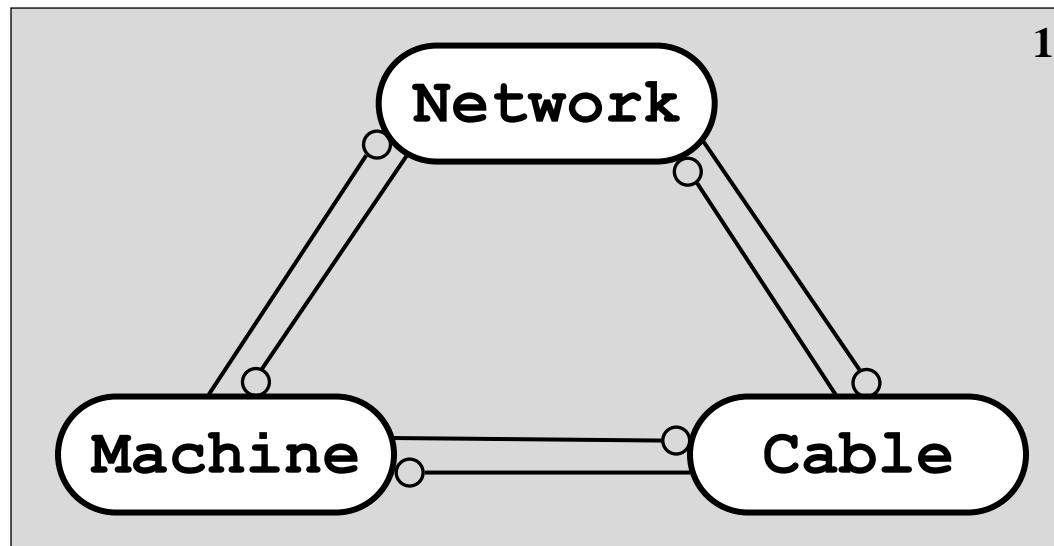
2. Survey of Advanced *Levelization* Techniques

Factoring



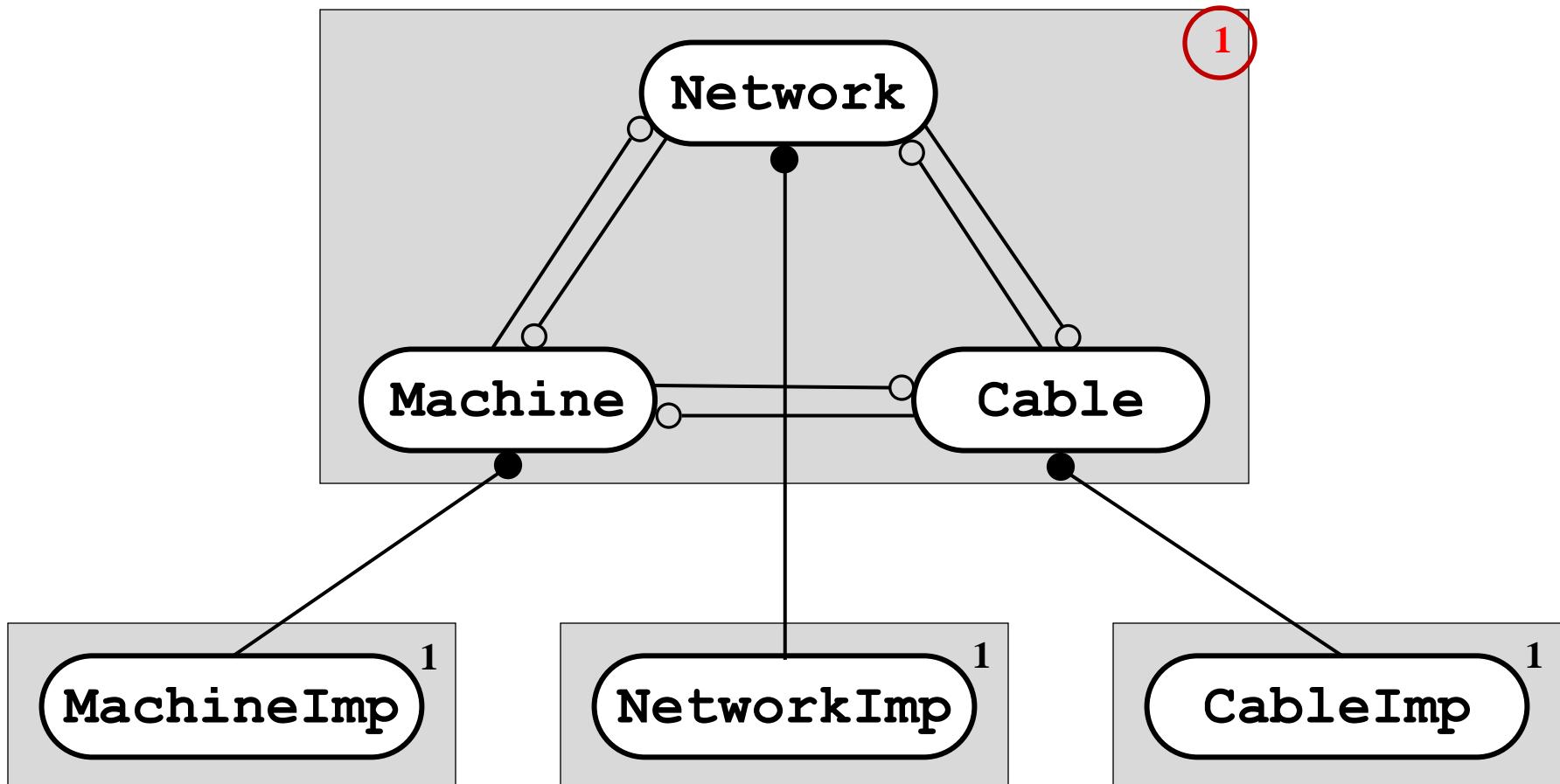
2. Survey of Advanced *Levelization* Techniques

Factoring



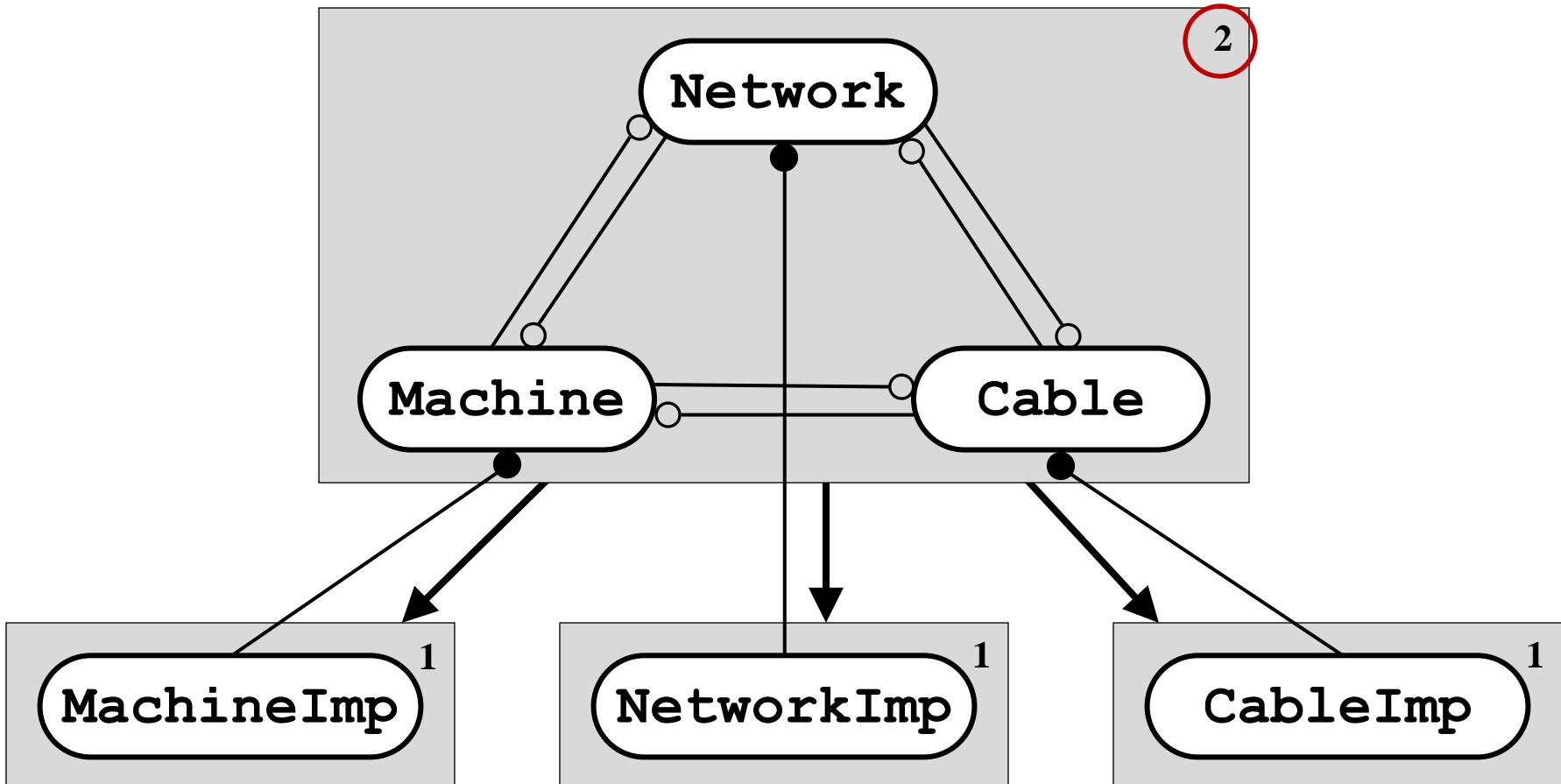
2. Survey of Advanced *Levelization* Techniques

Factoring



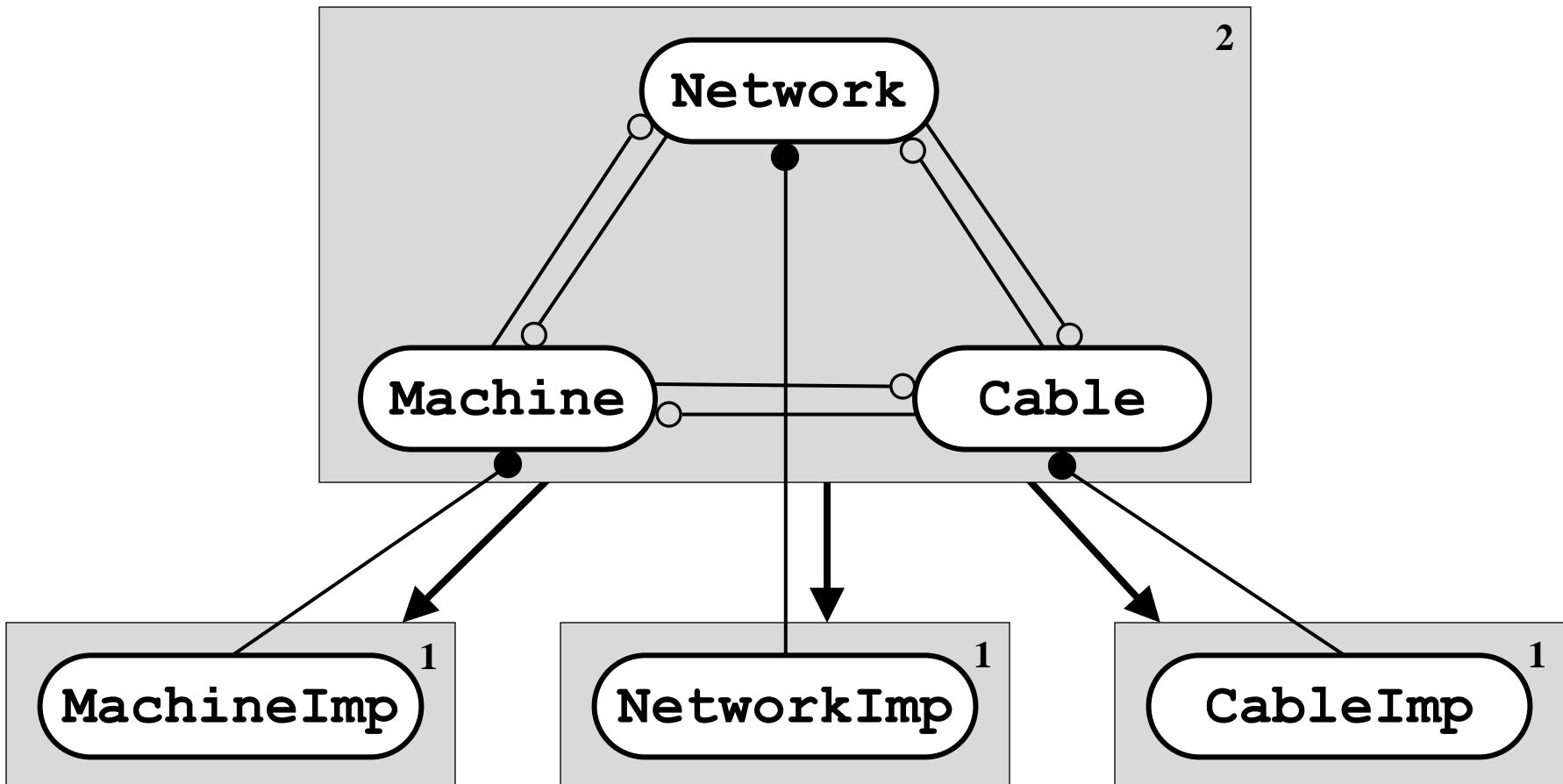
2. Survey of Advanced *Levelization* Techniques

Factoring



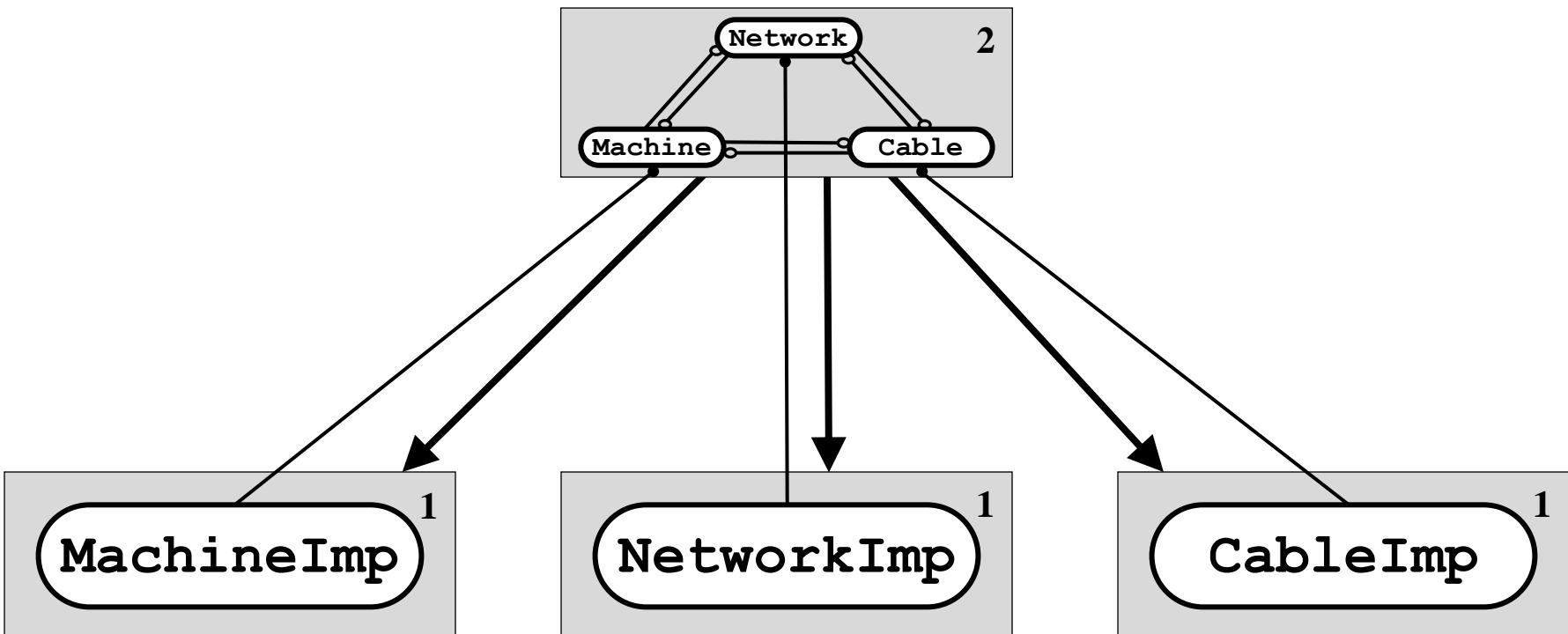
2. Survey of Advanced *Levelization* Techniques

Factoring



2. Survey of Advanced *Levelization* Techniques

Factoring



2. Survey of Advanced *Levelization* Techniques

Factoring

Discussion?

2. Survey of Advanced *Levelization* Techniques

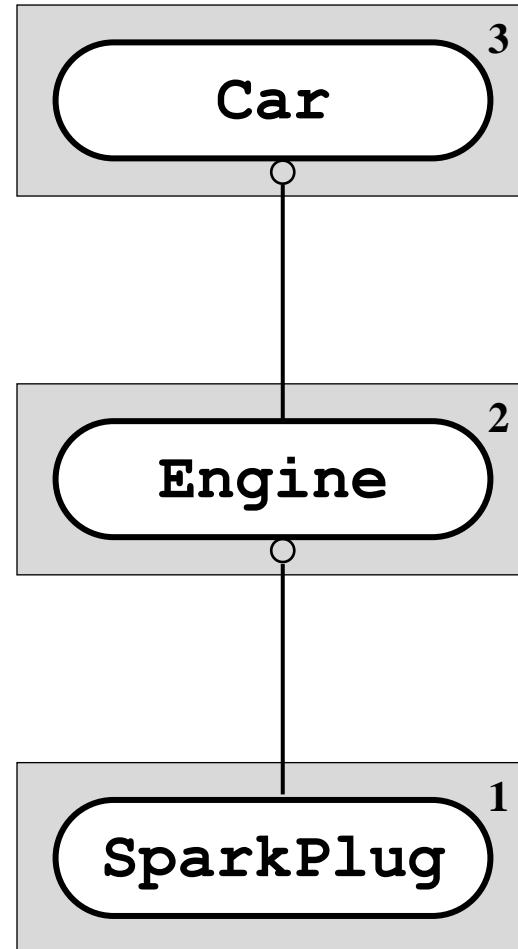
Escalating Encapsulation

Escalating Encapsulation

– Moving the point at which implementation details are hidden from clients to a higher level in the physical hierarchy.

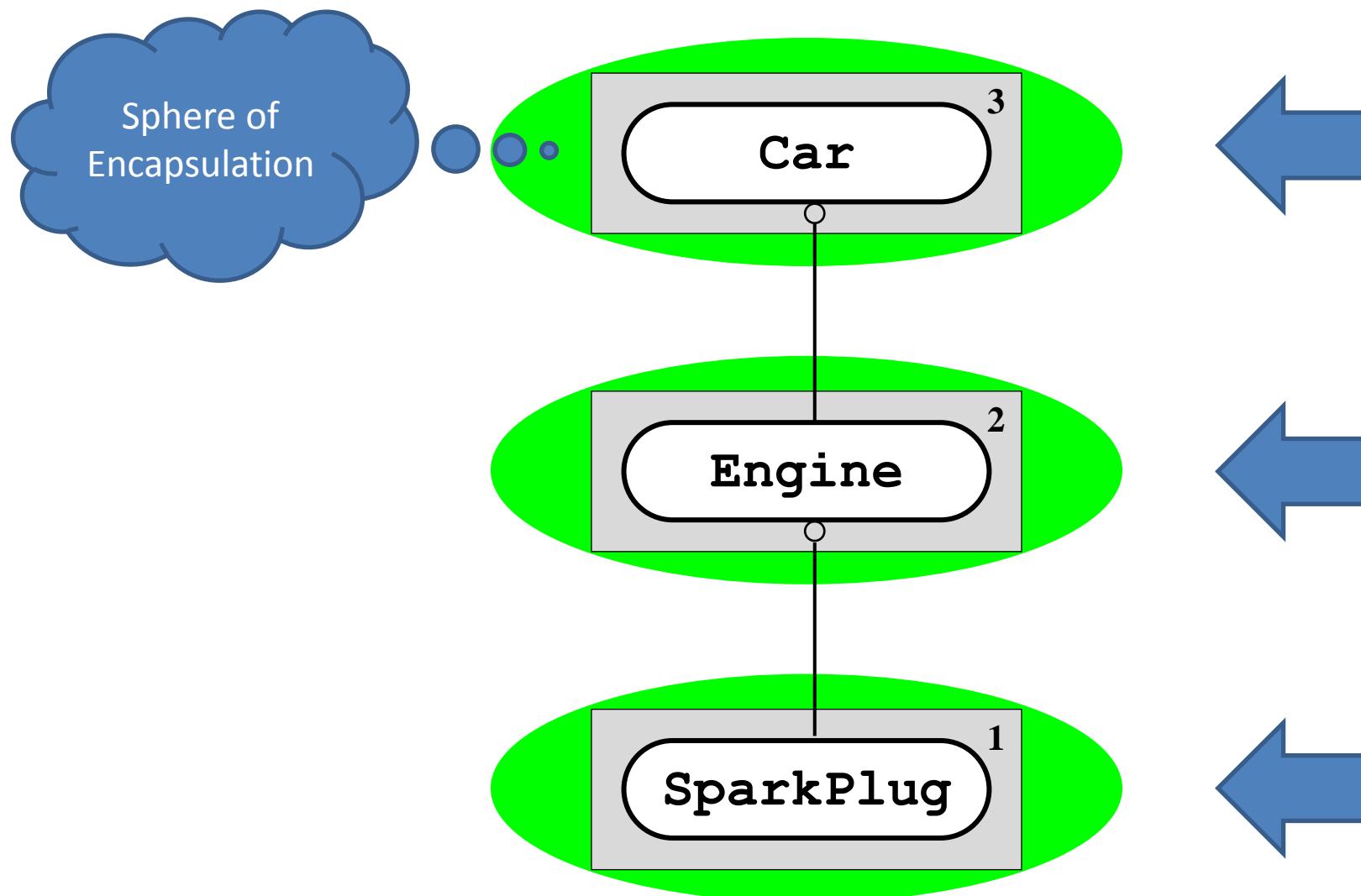
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation



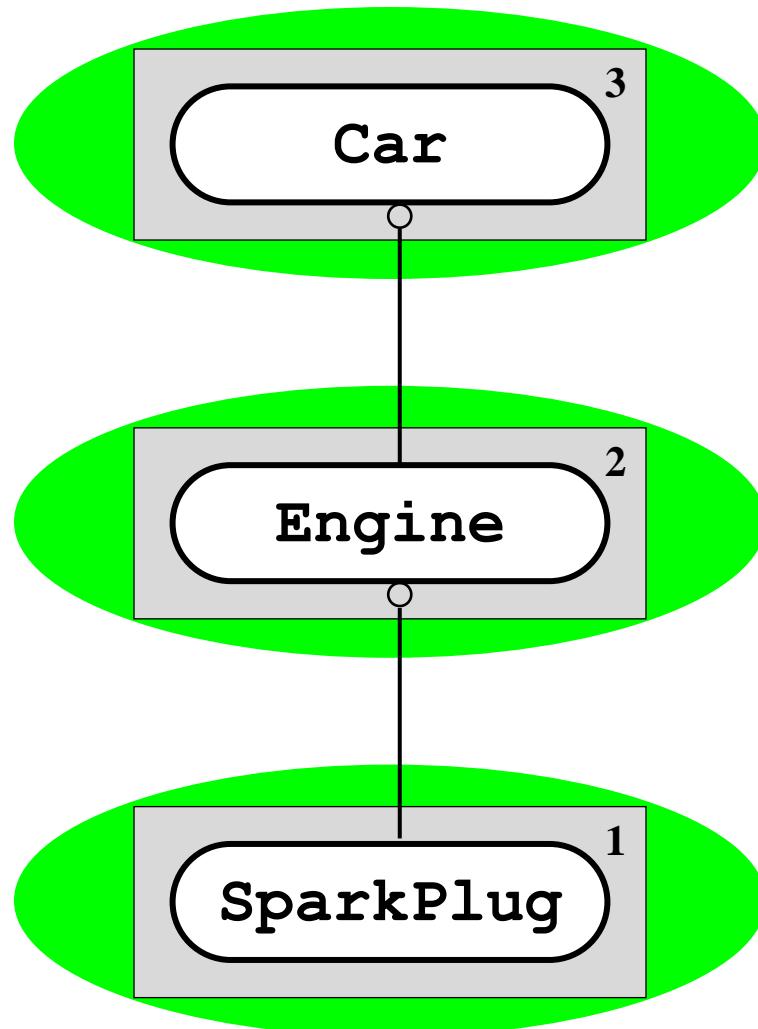
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation



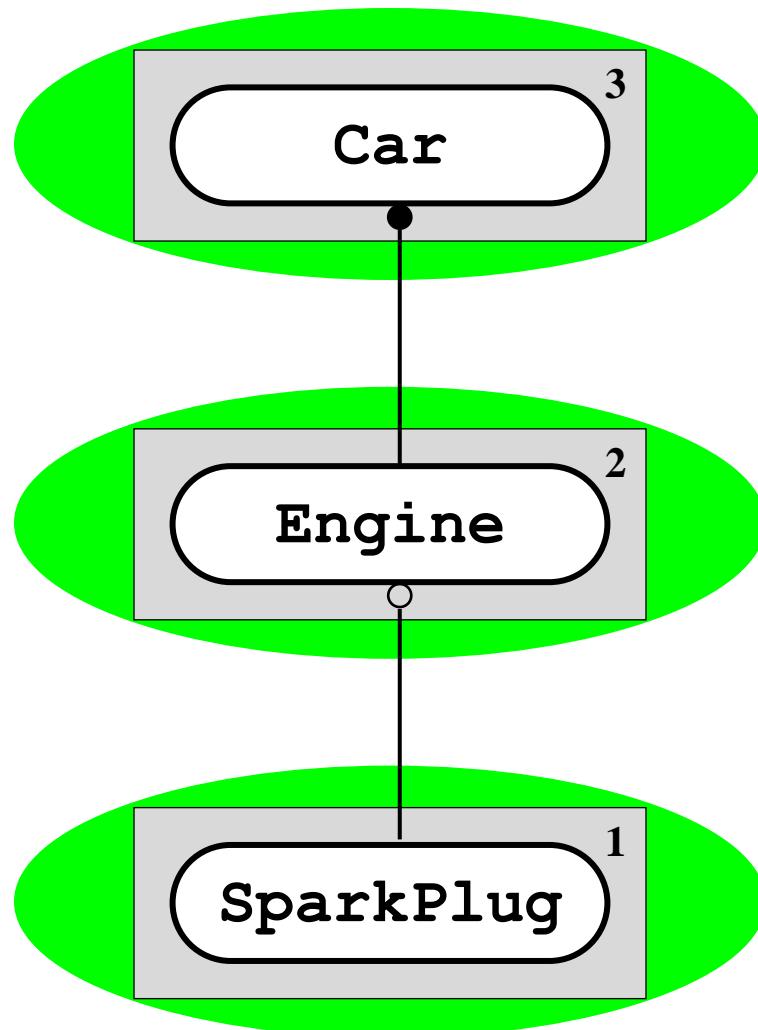
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation



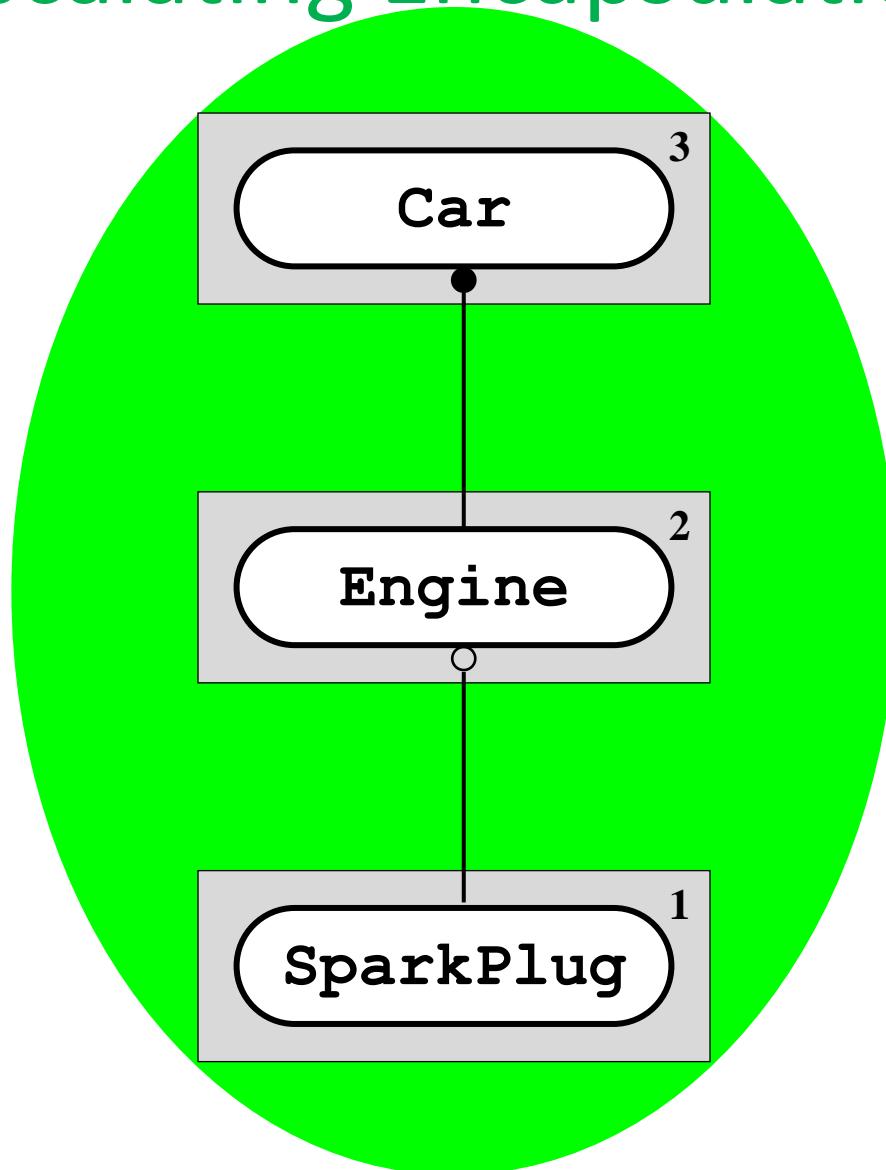
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation



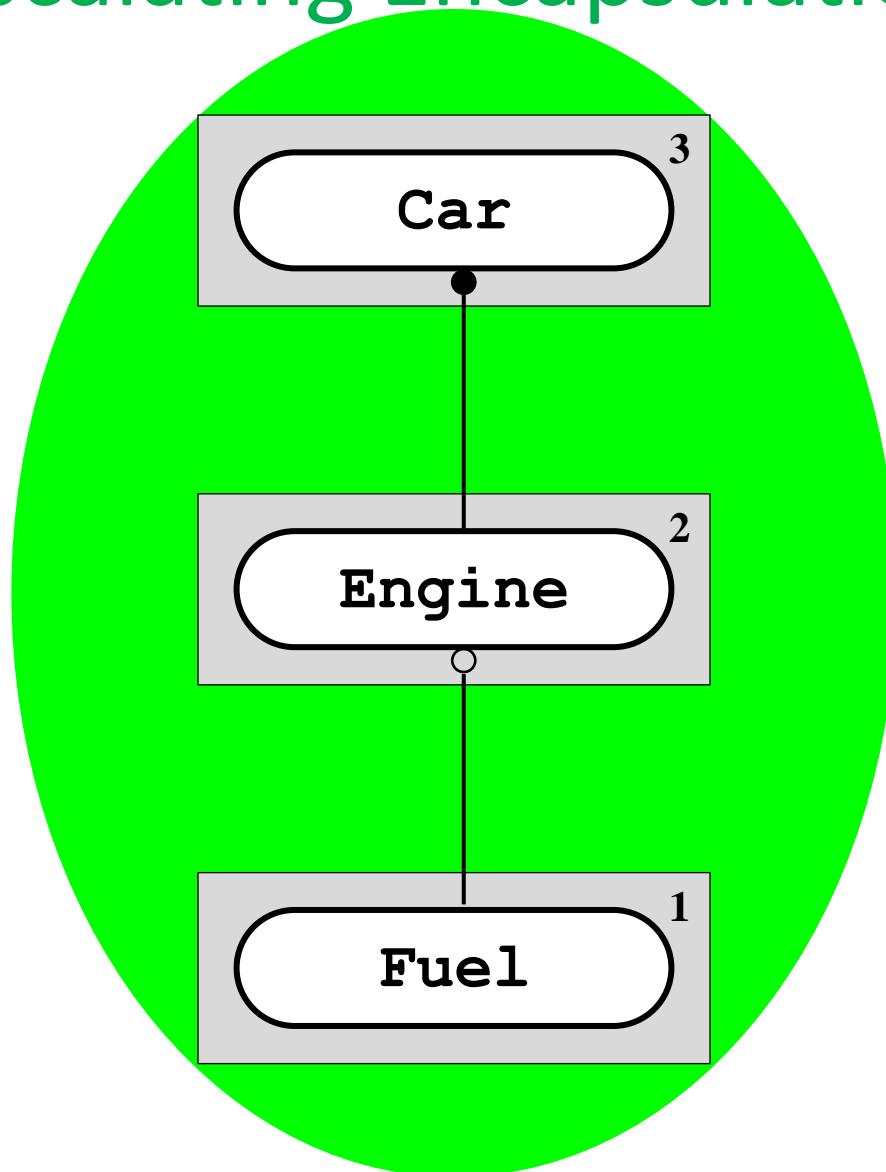
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation



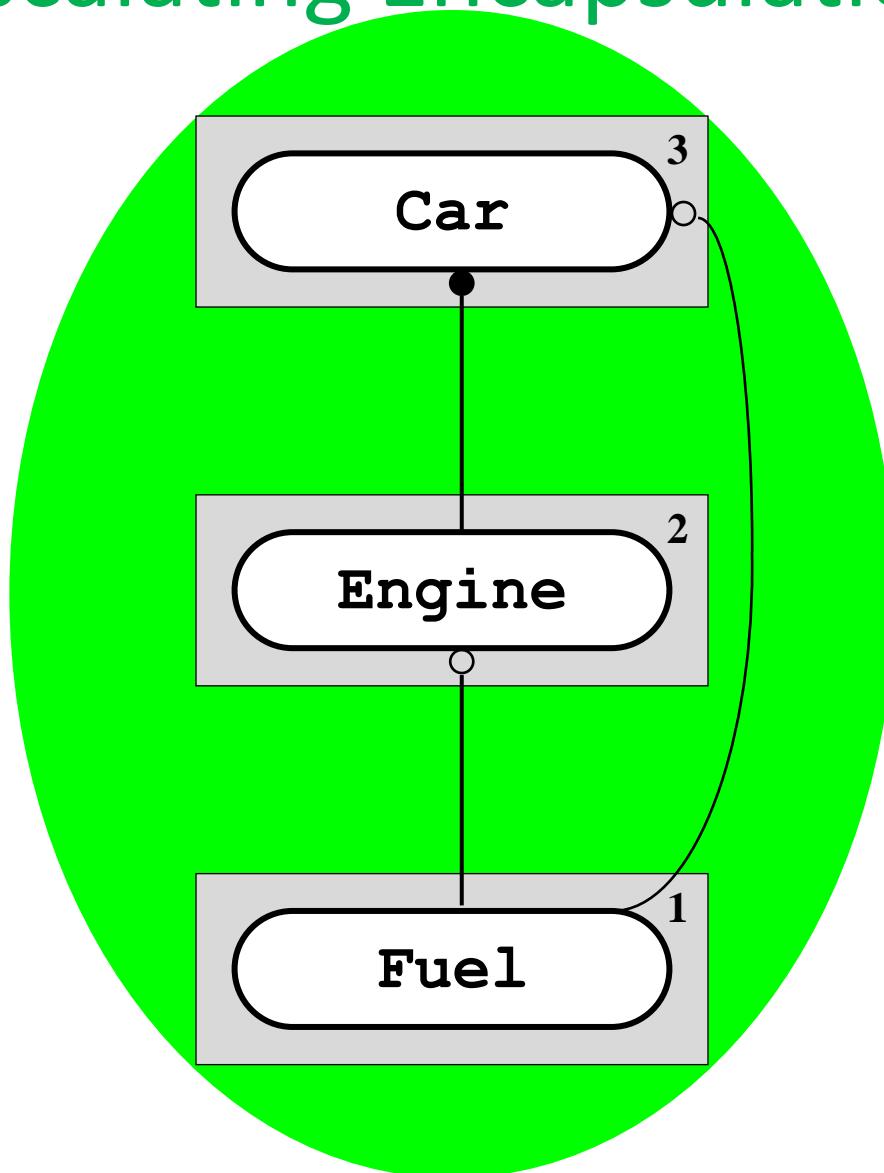
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation



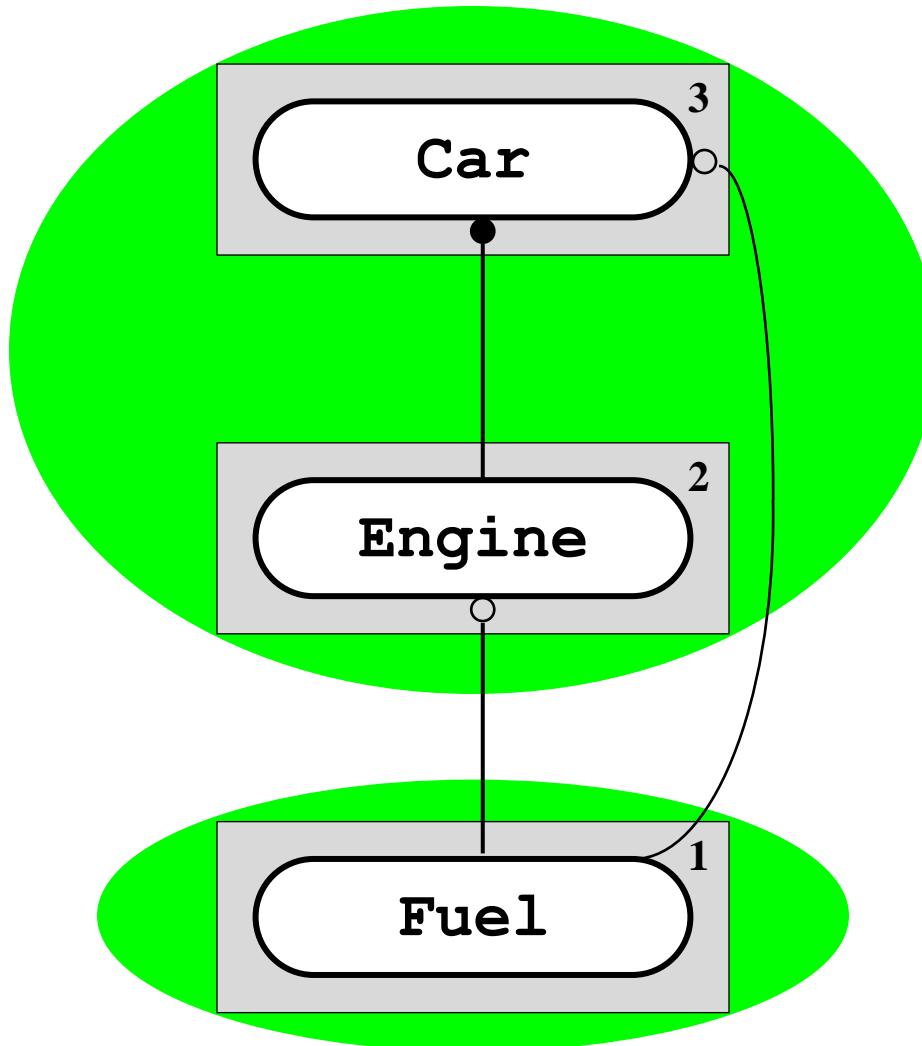
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation



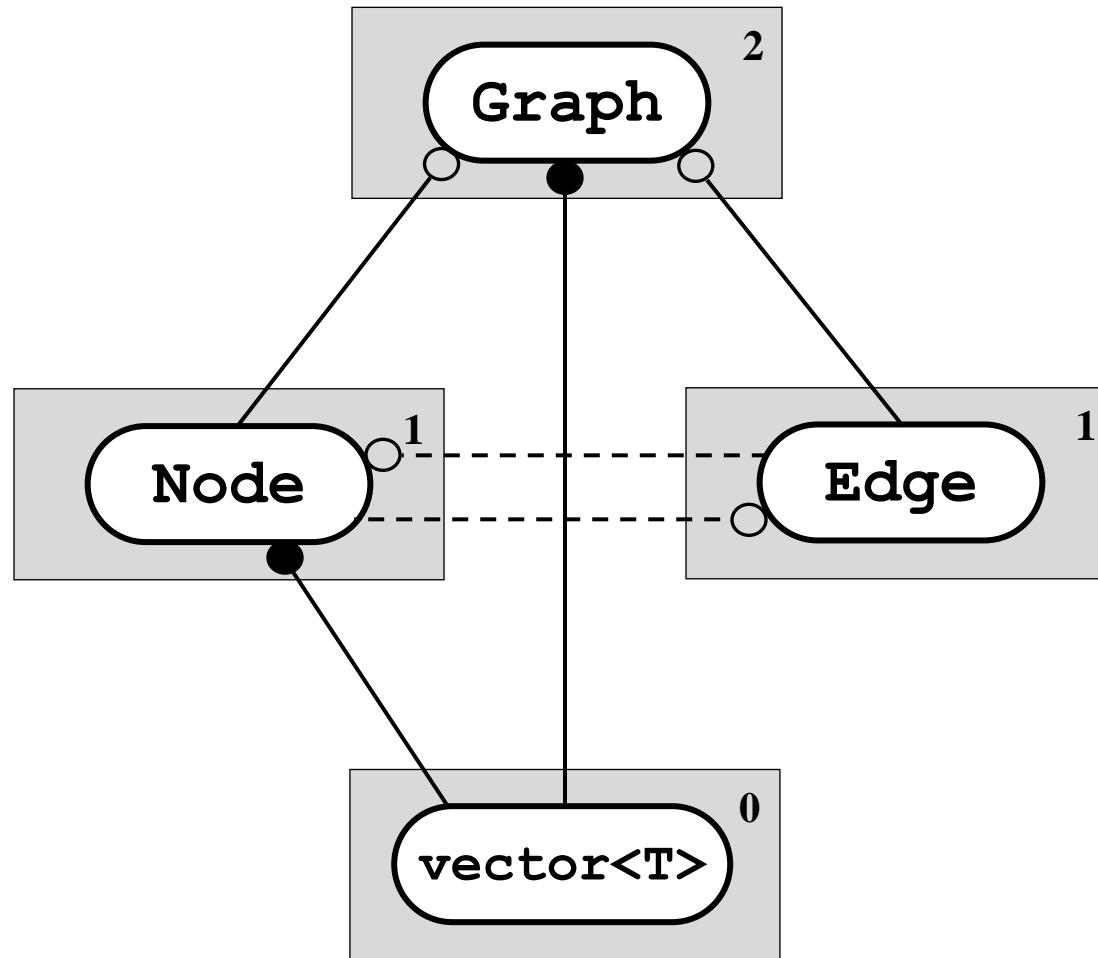
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation



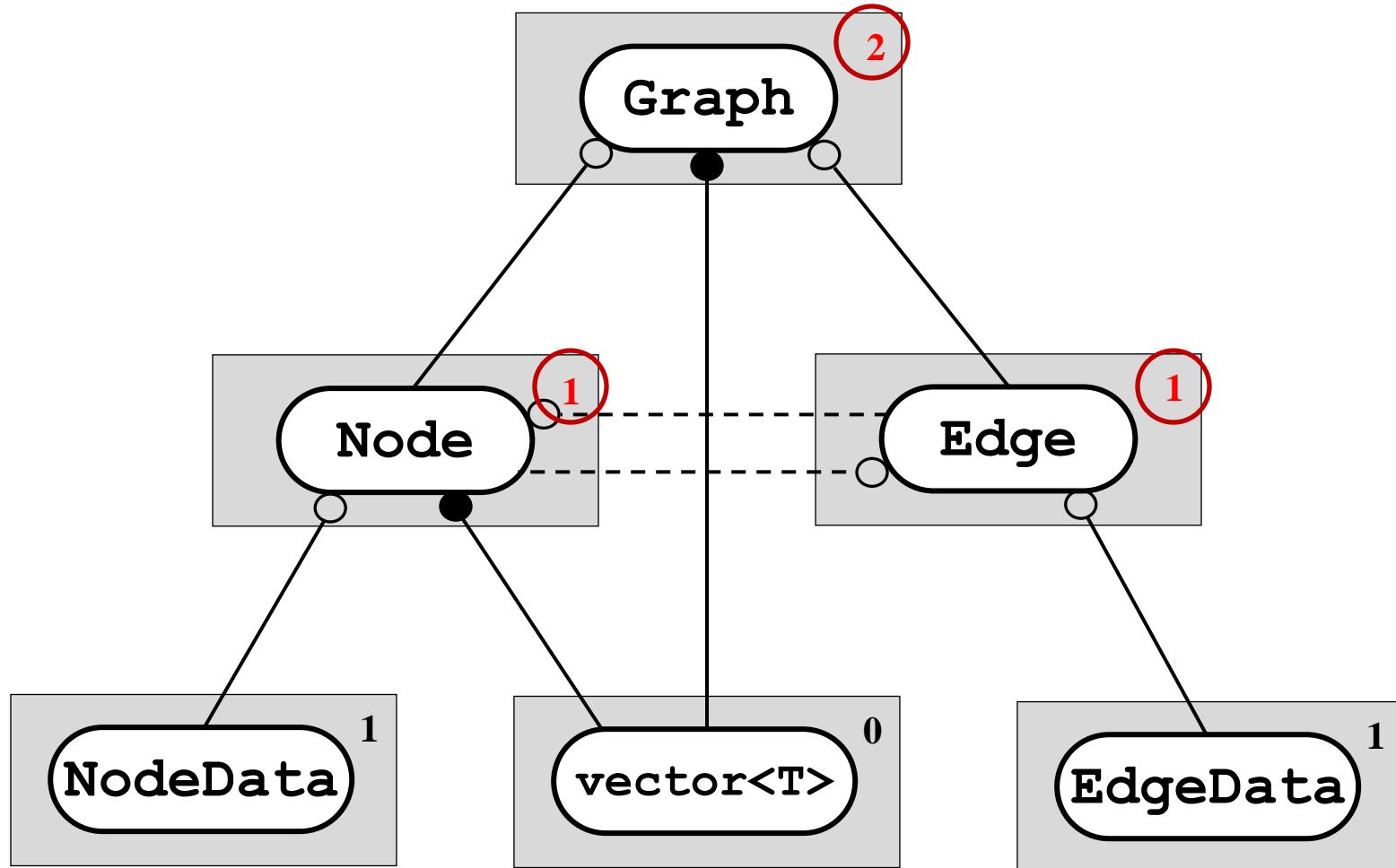
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation



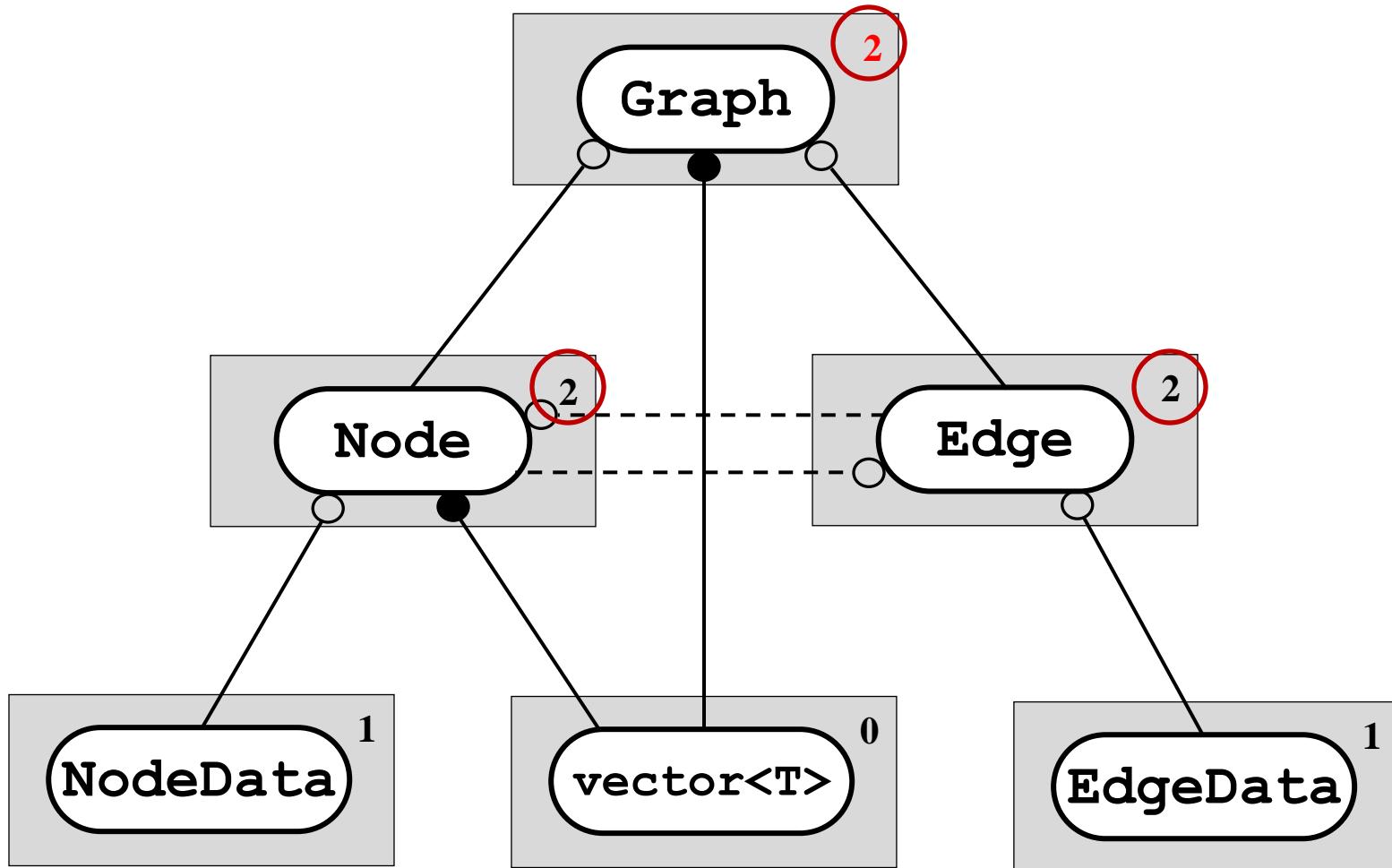
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation



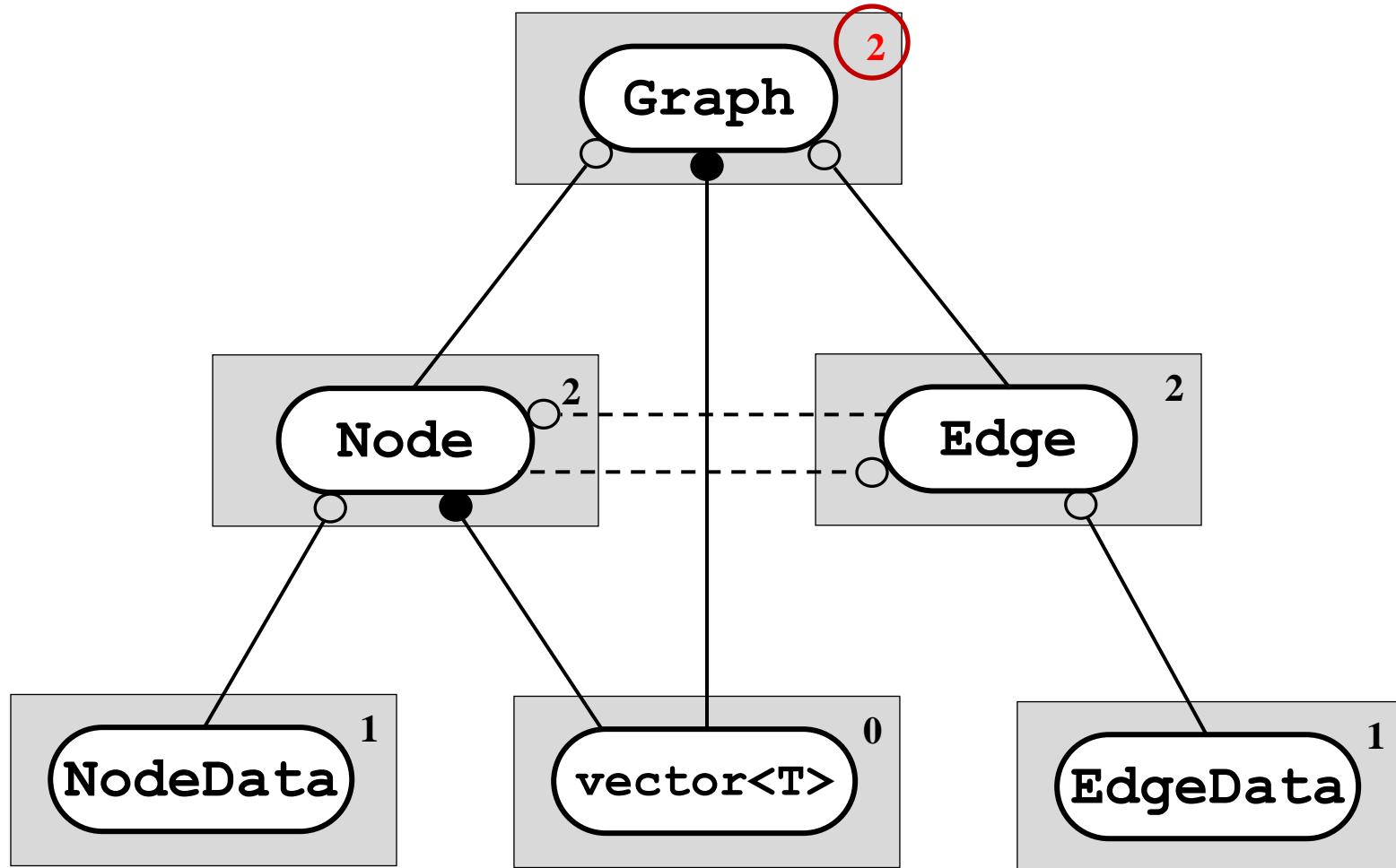
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation



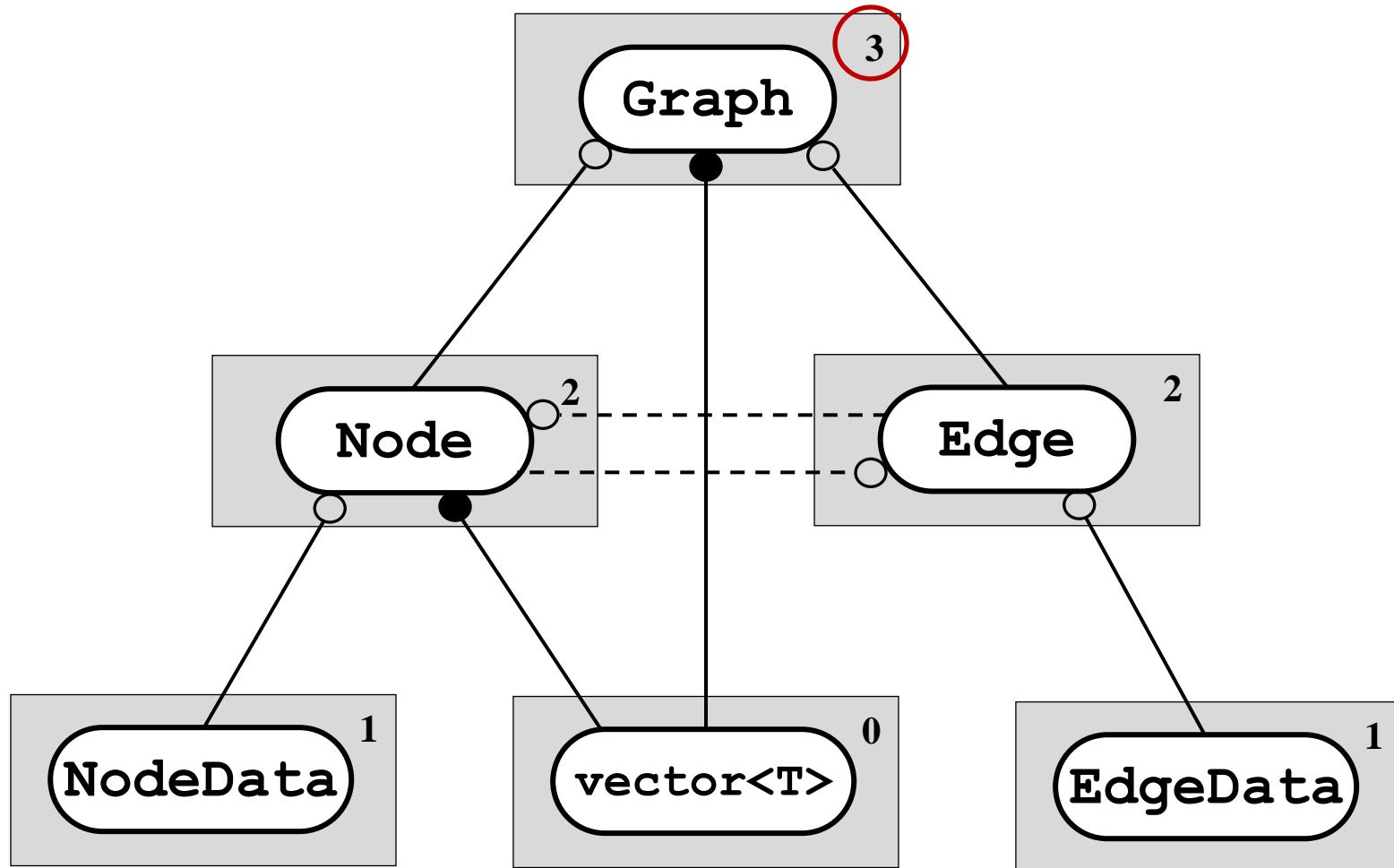
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation



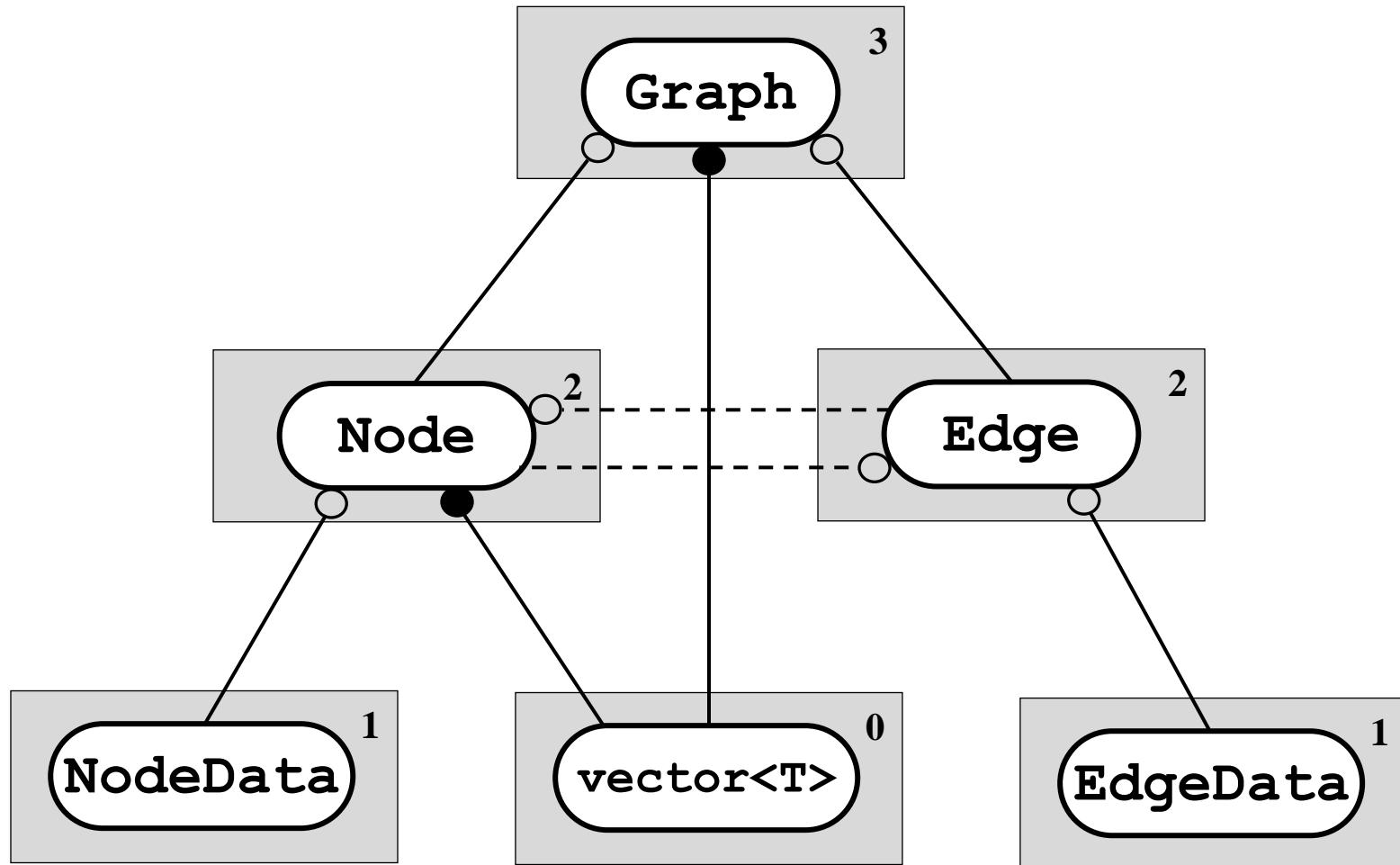
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

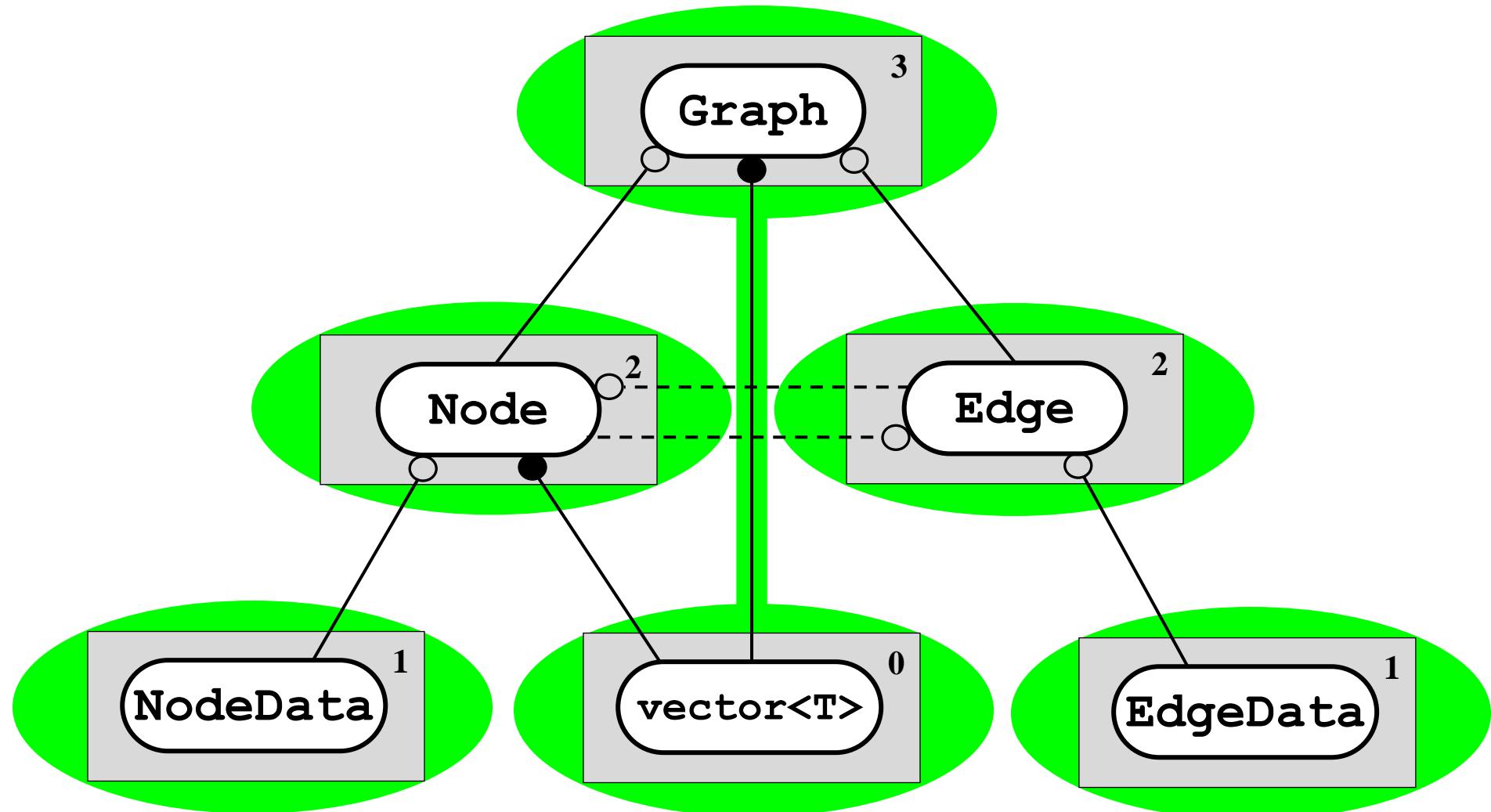


2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

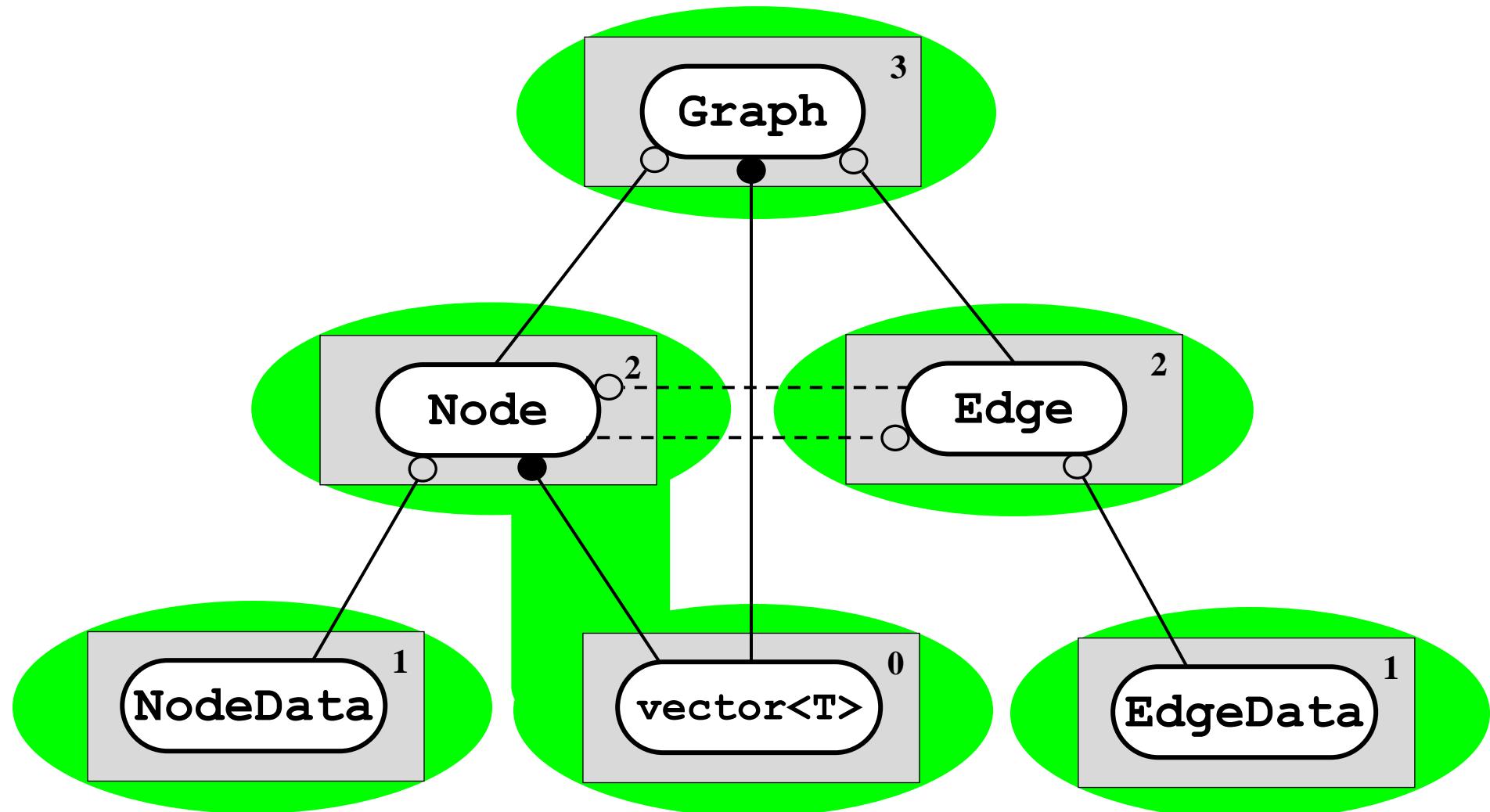


2. Survey of Advanced *Levelization* Techniques Escalating Encapsulation



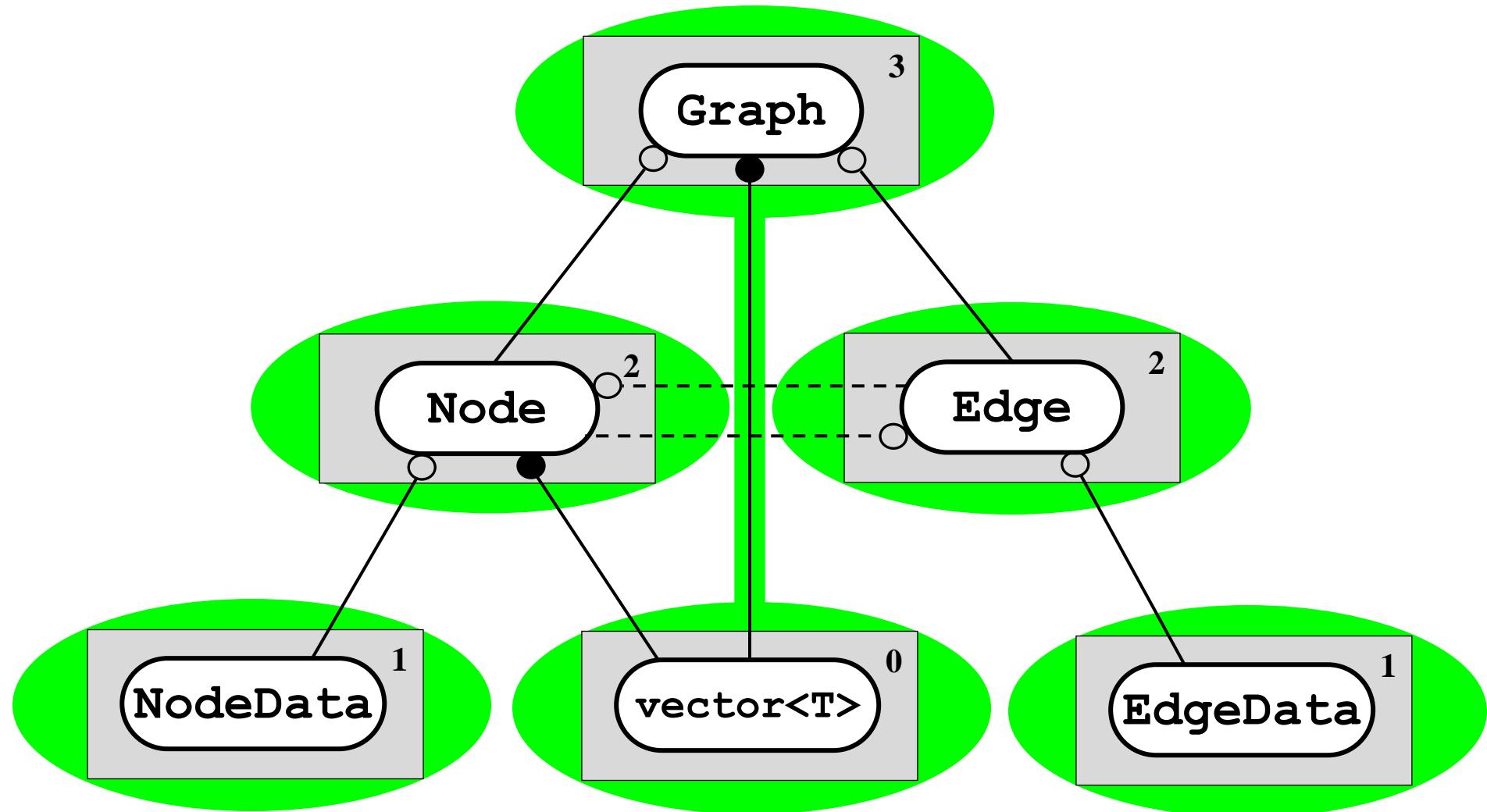
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation



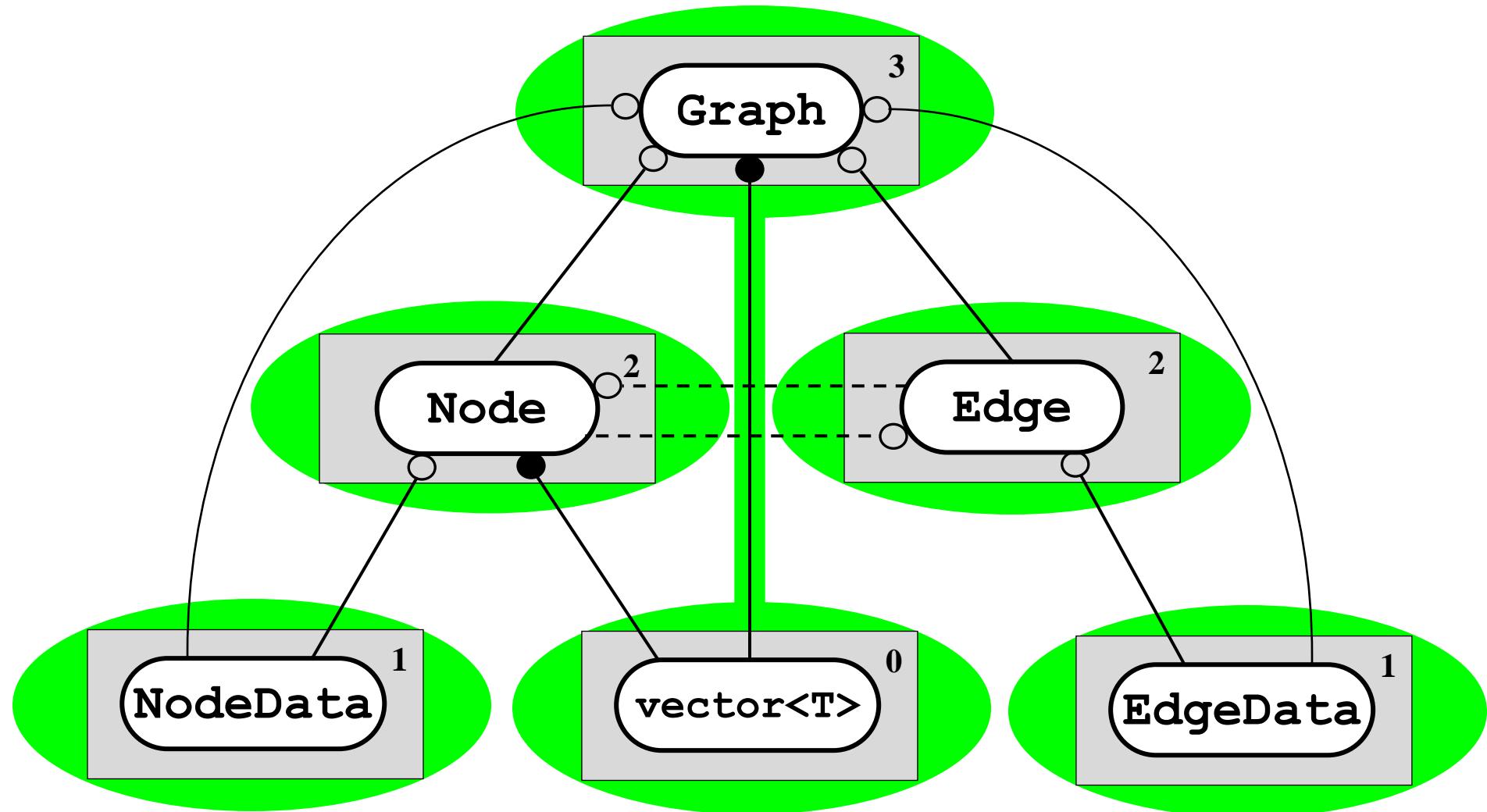
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation



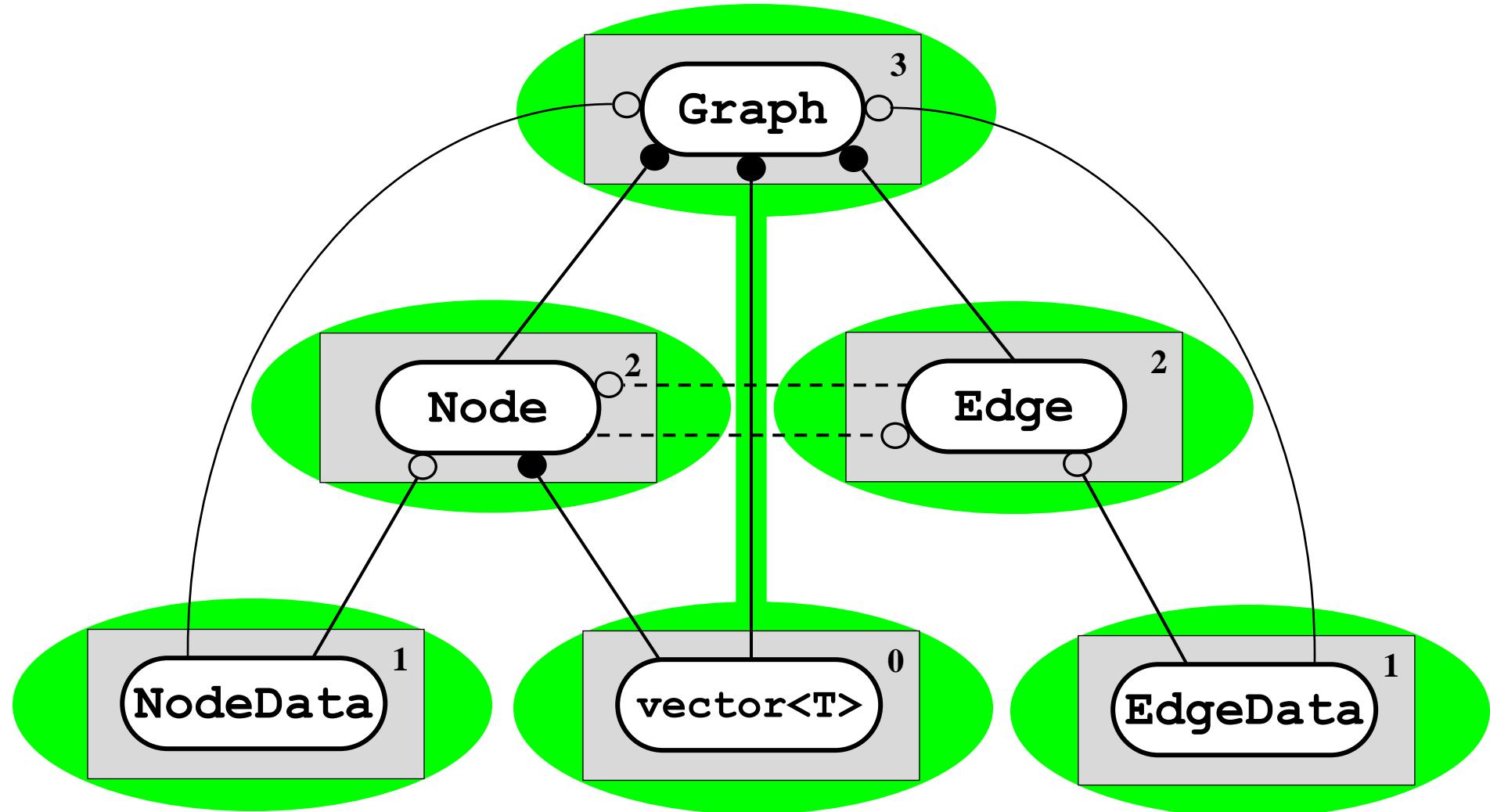
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation



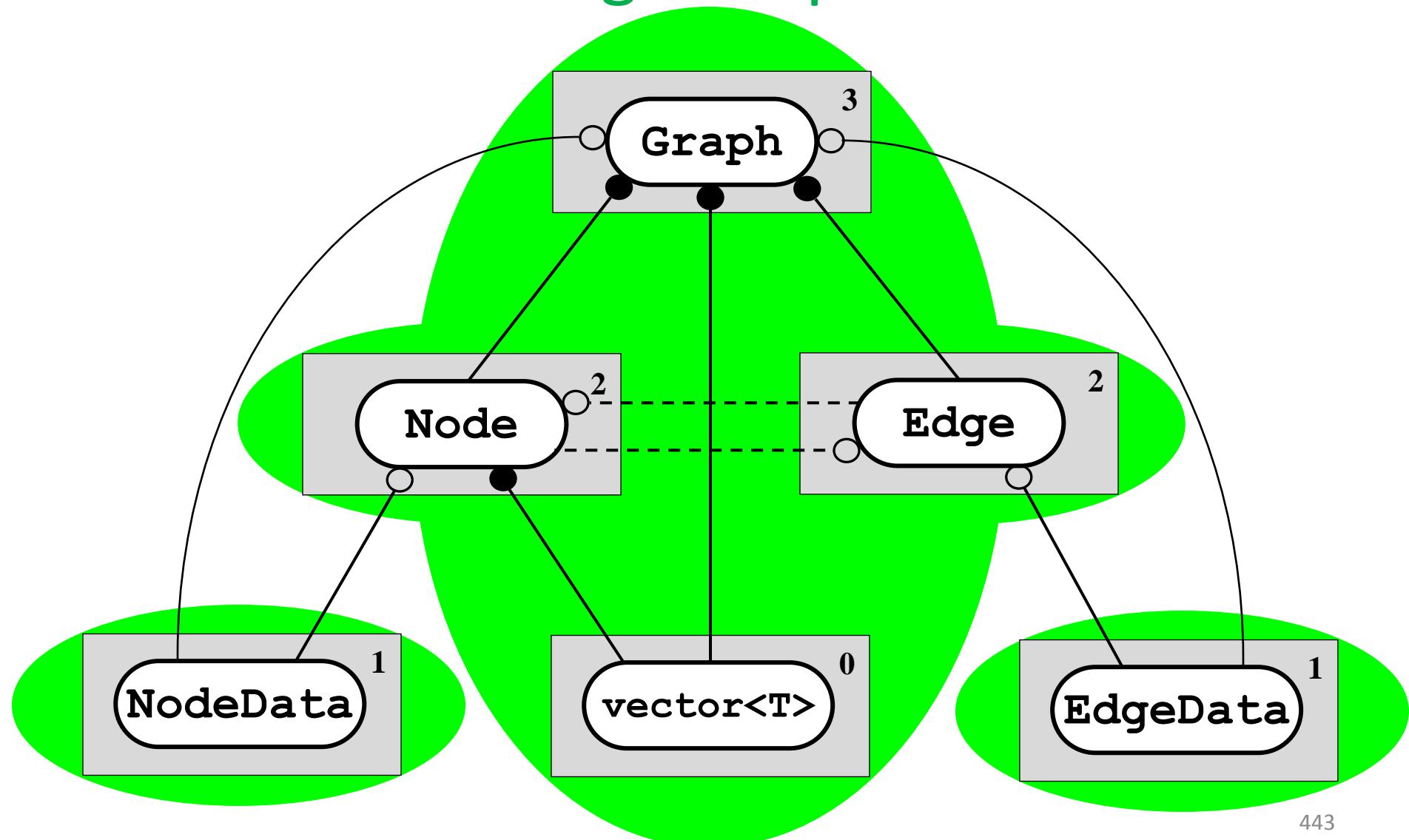
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation



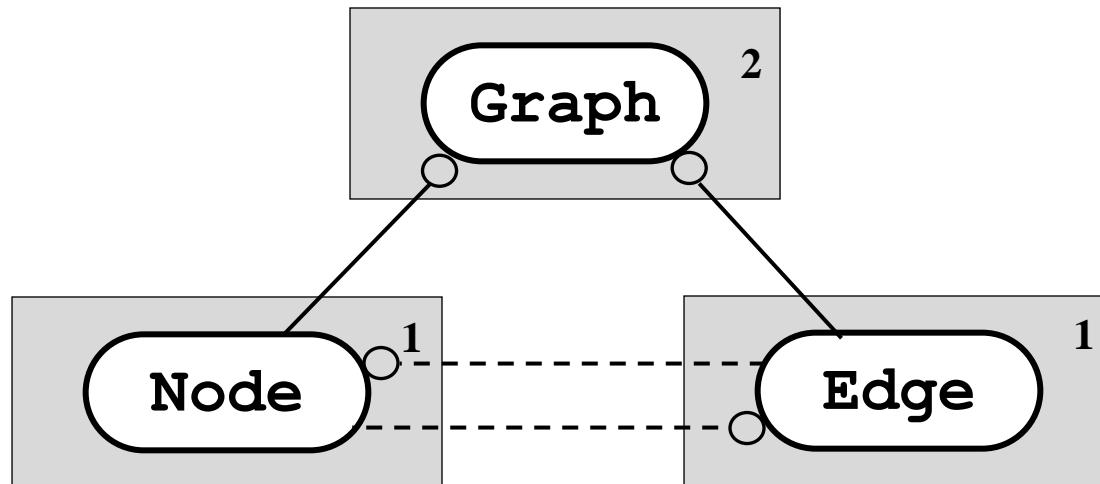
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation



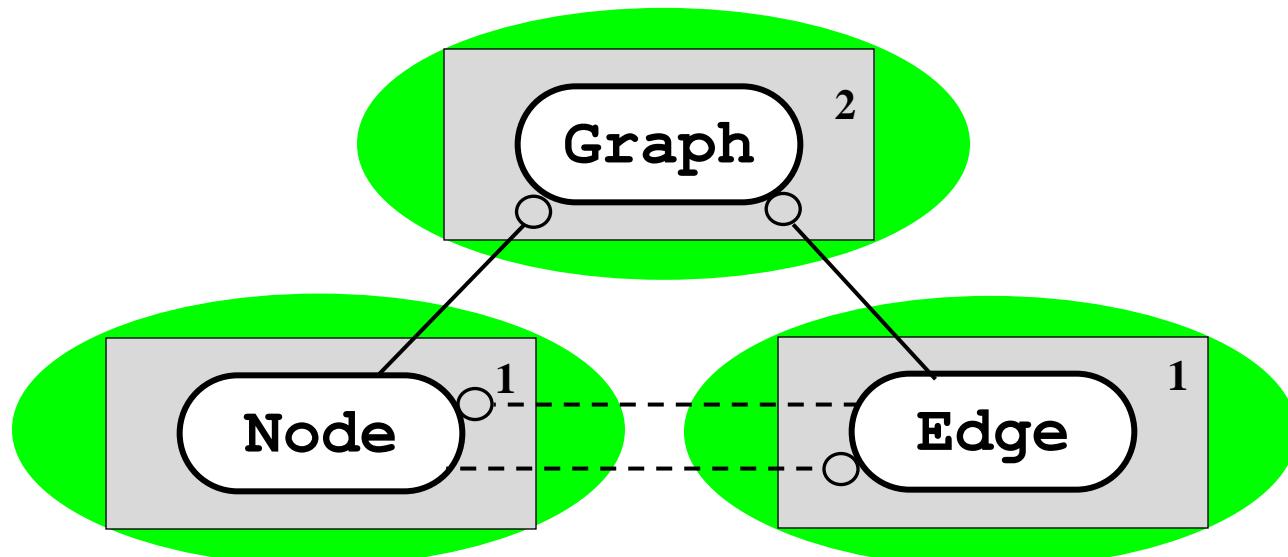
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation



2. Survey of Advanced *Levelization* Techniques

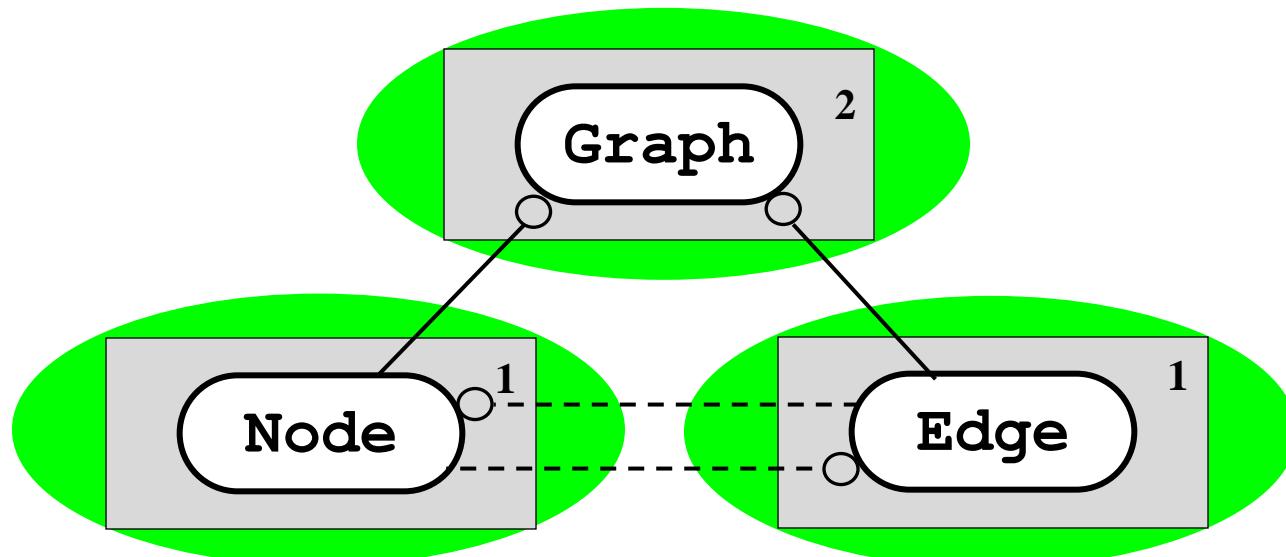
Escalating Encapsulation



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

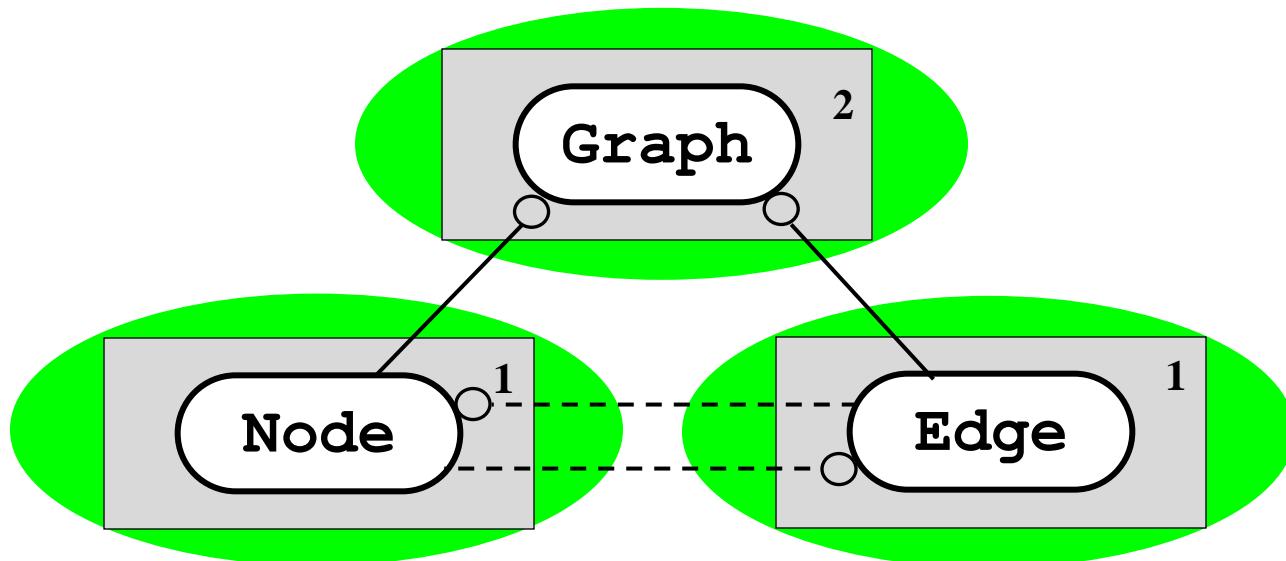
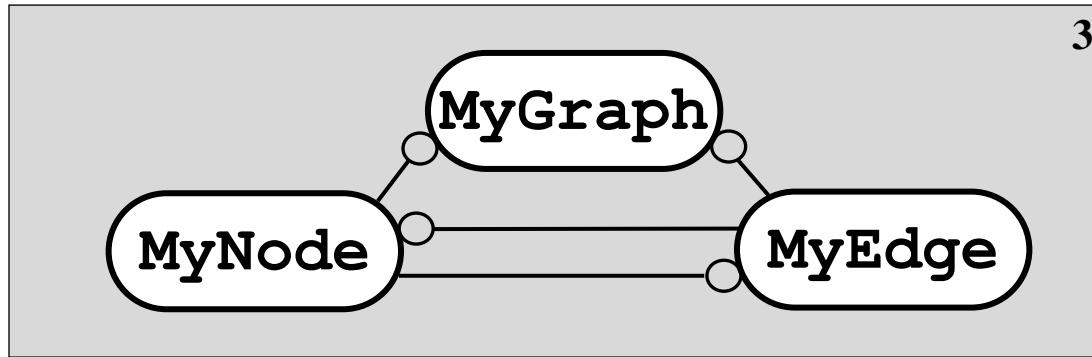
Single-Component Wrapper



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

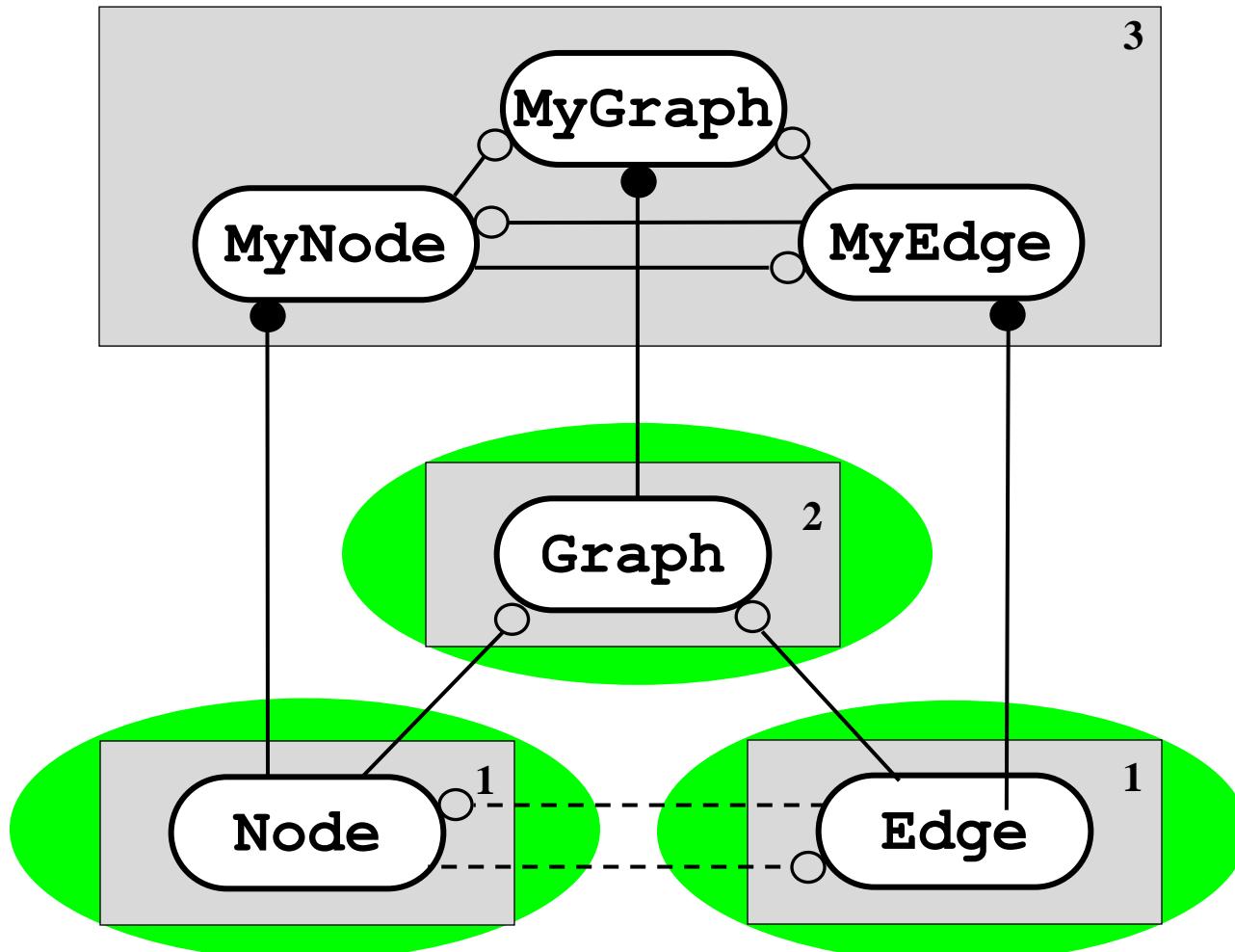
Single-Component Wrapper



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

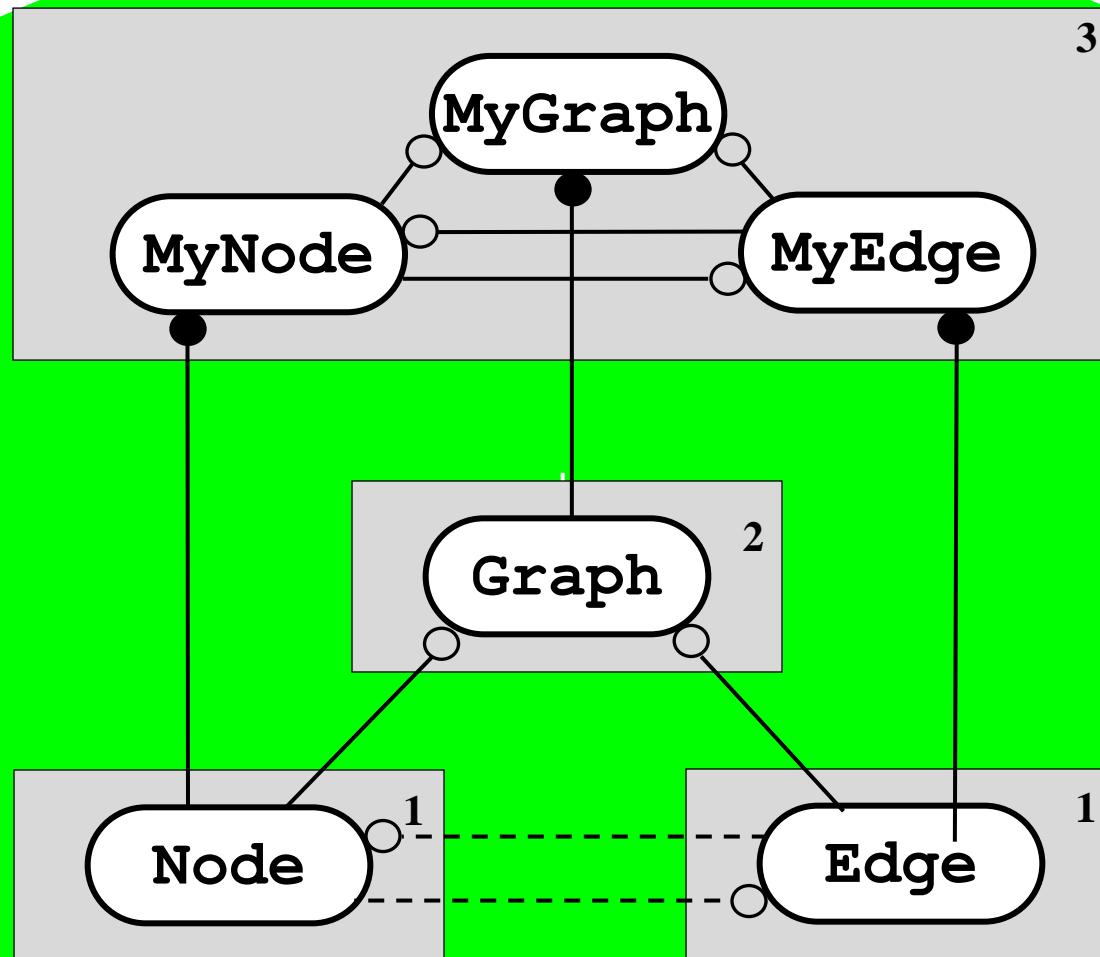
Single-Component Wrapper



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

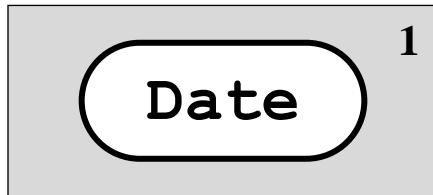
Single-Component Wrapper



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

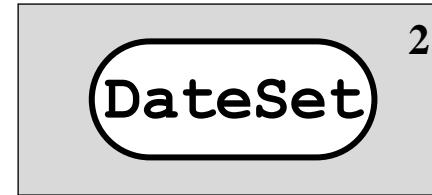
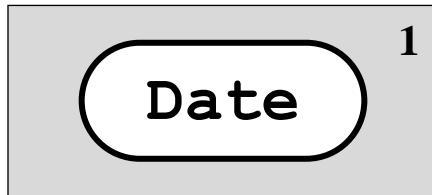
Multi-Component Wrapper



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

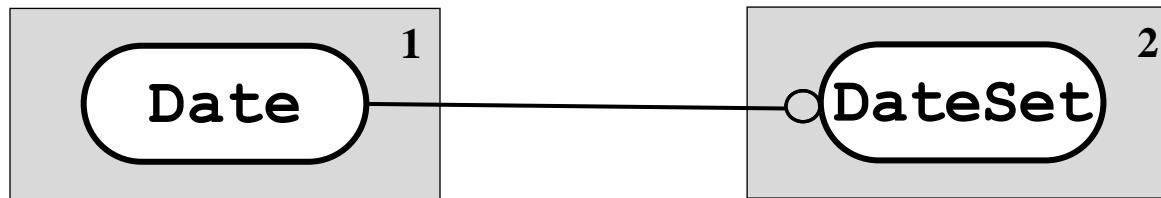
Multi-Component Wrapper



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

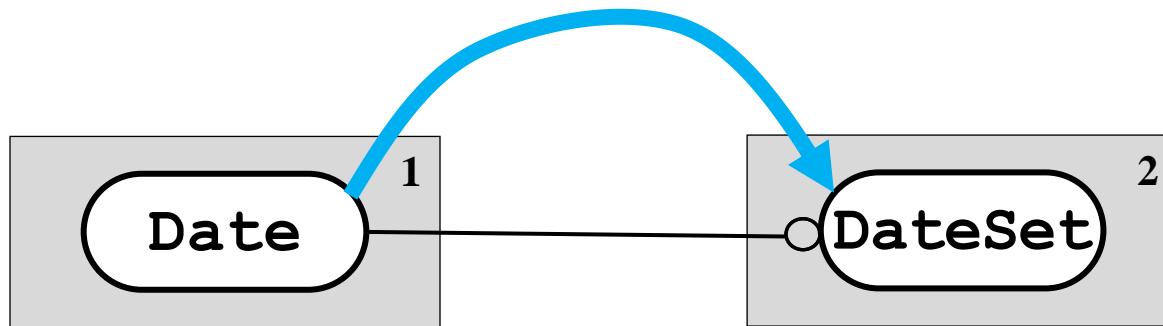
Multi-Component Wrapper



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

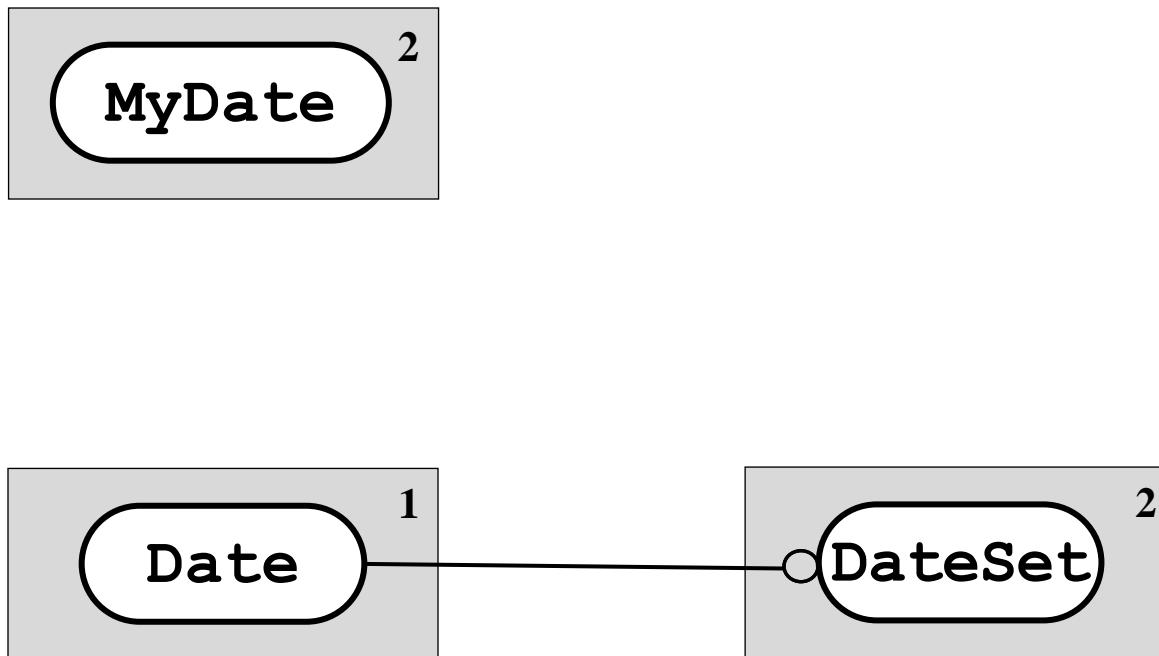
Multi-Component Wrapper



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

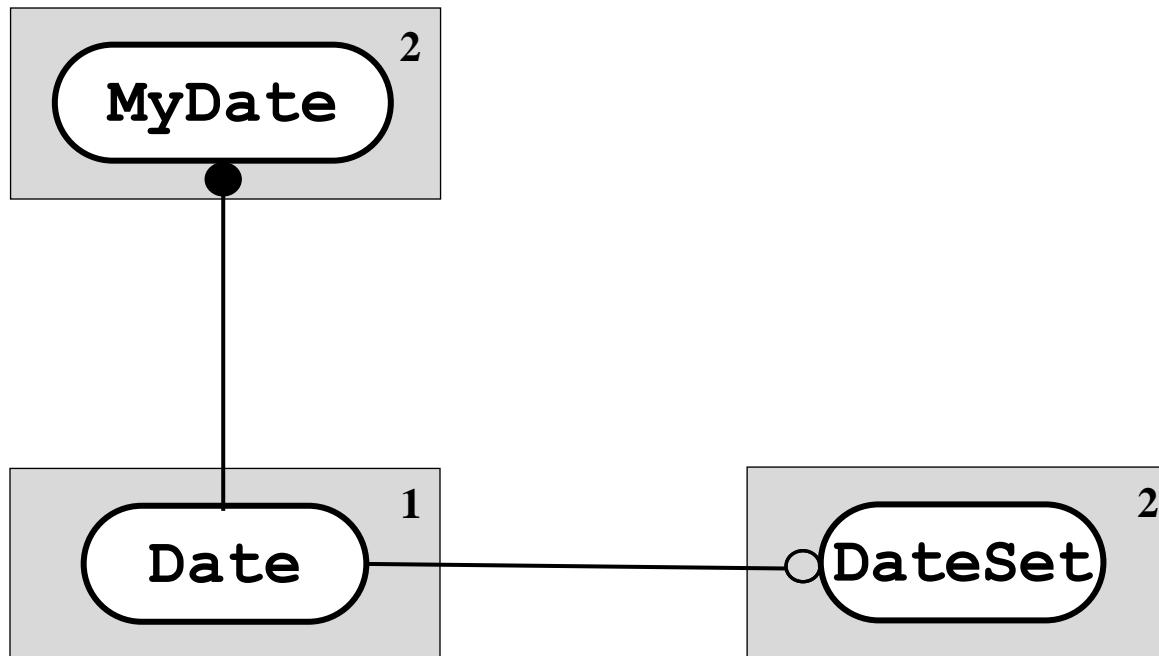
Multi-Component Wrapper



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

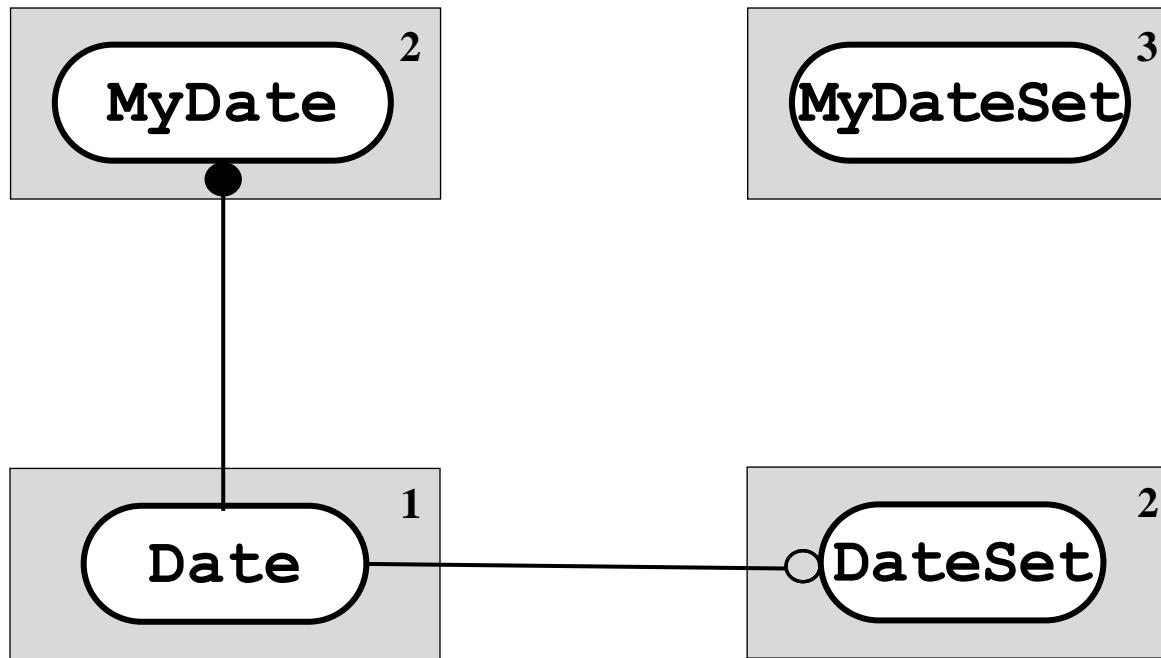
Multi-Component Wrapper



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

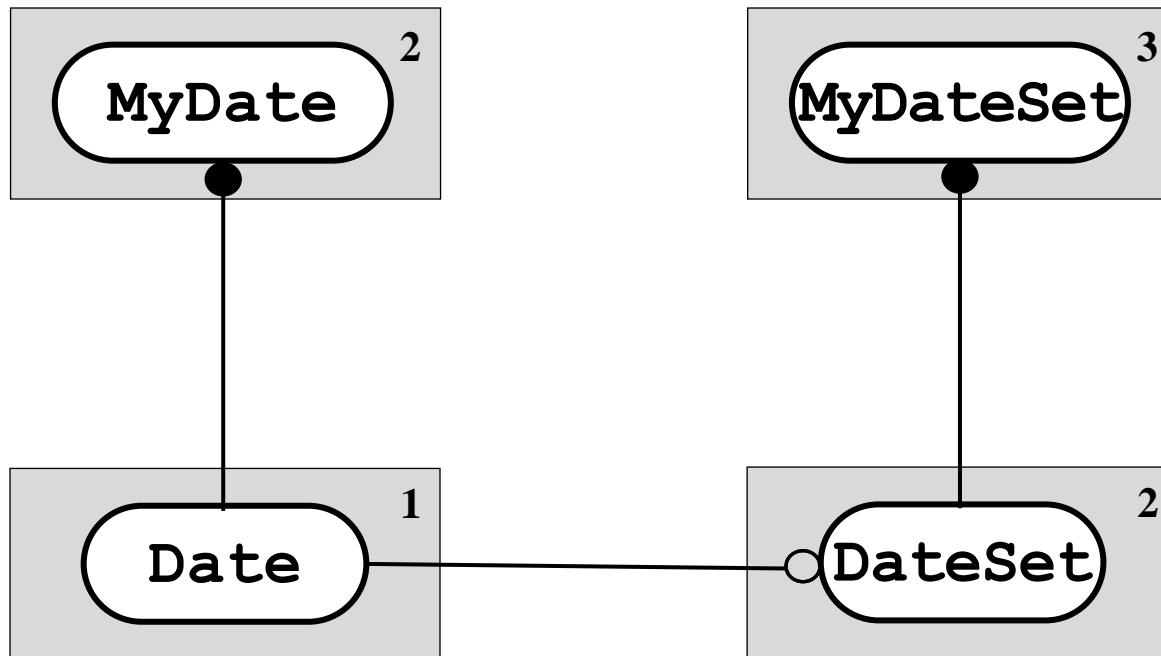
Multi-Component Wrapper



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

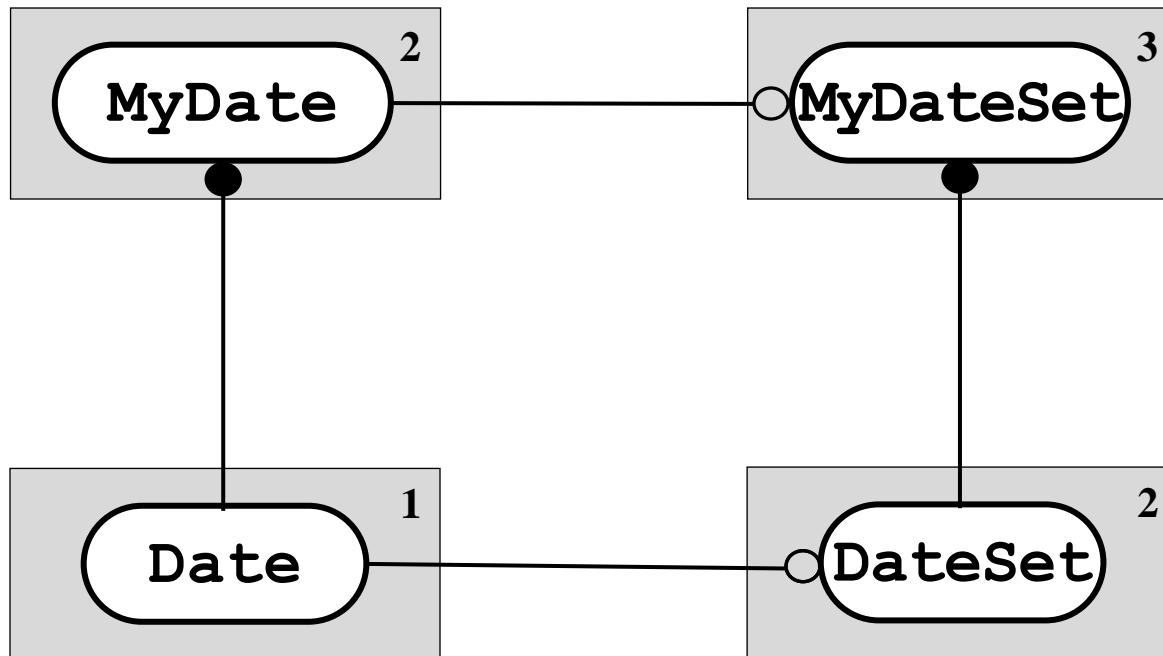
Multi-Component Wrapper



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

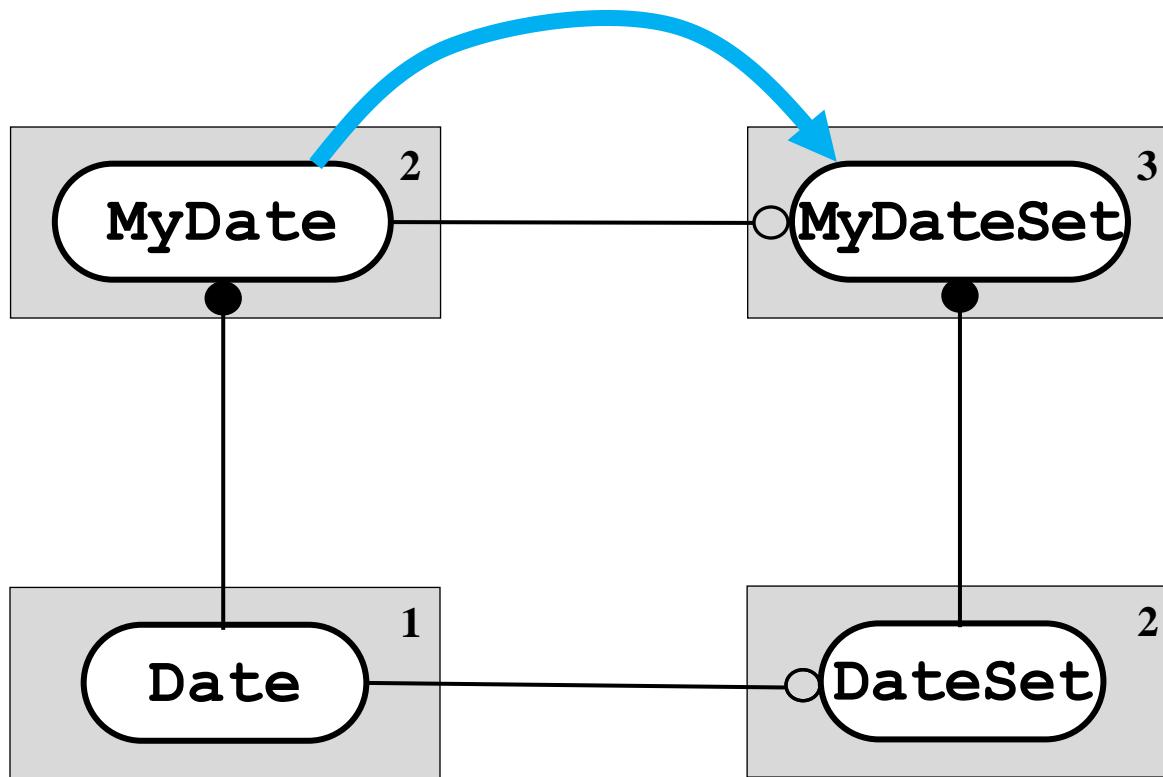
Multi-Component Wrapper



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

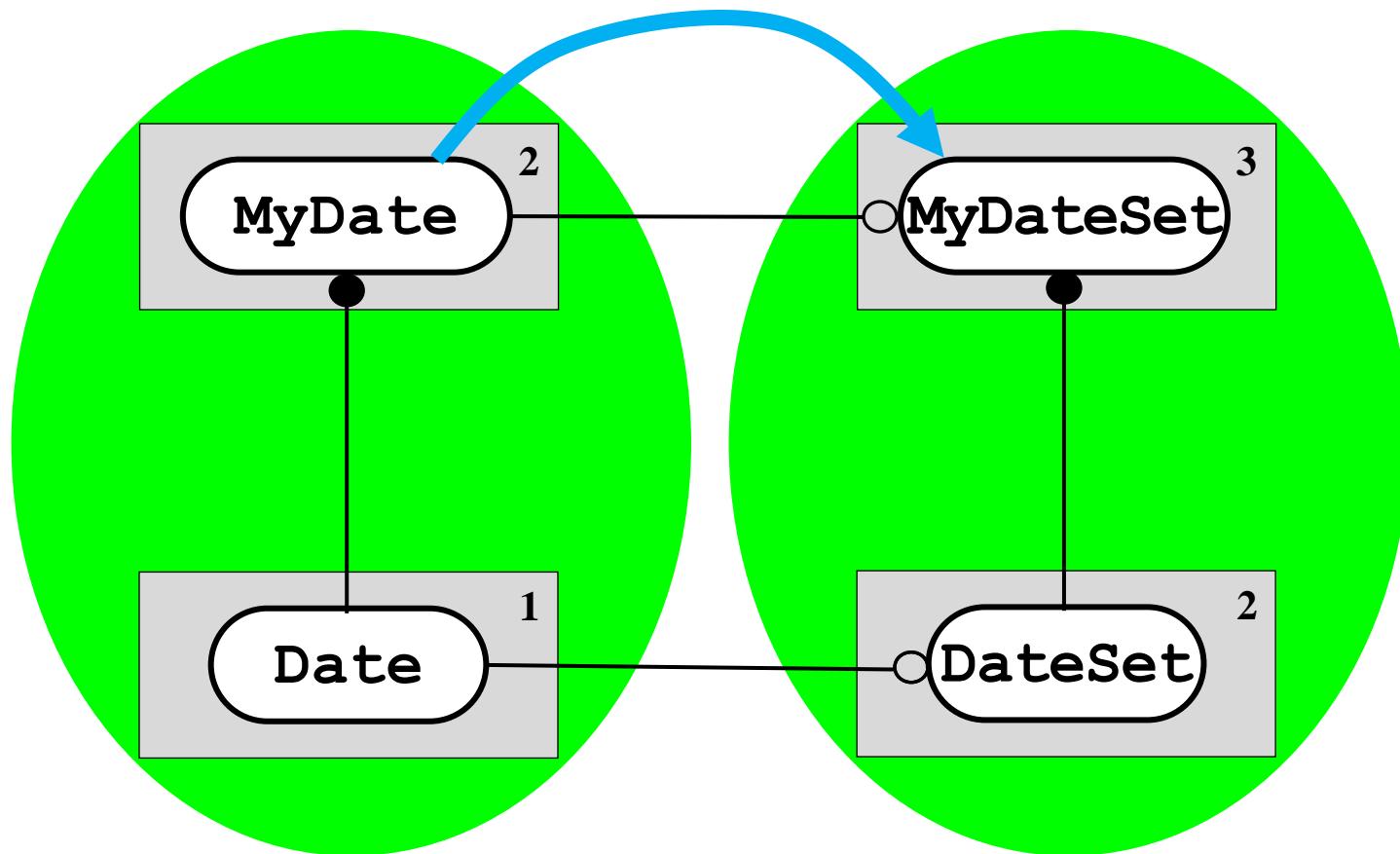
Multi-Component Wrapper



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

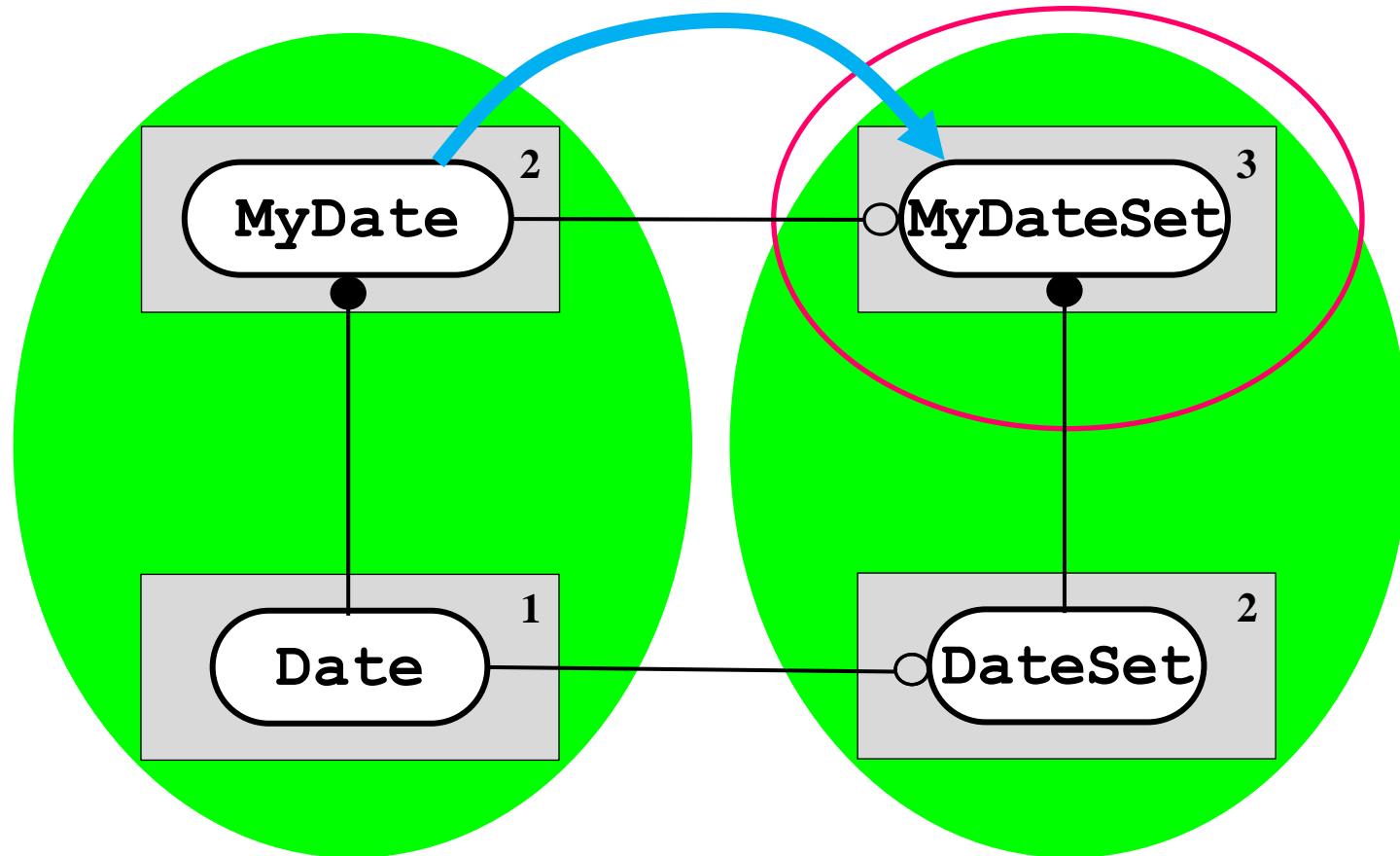
Multi-Component Wrapper



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

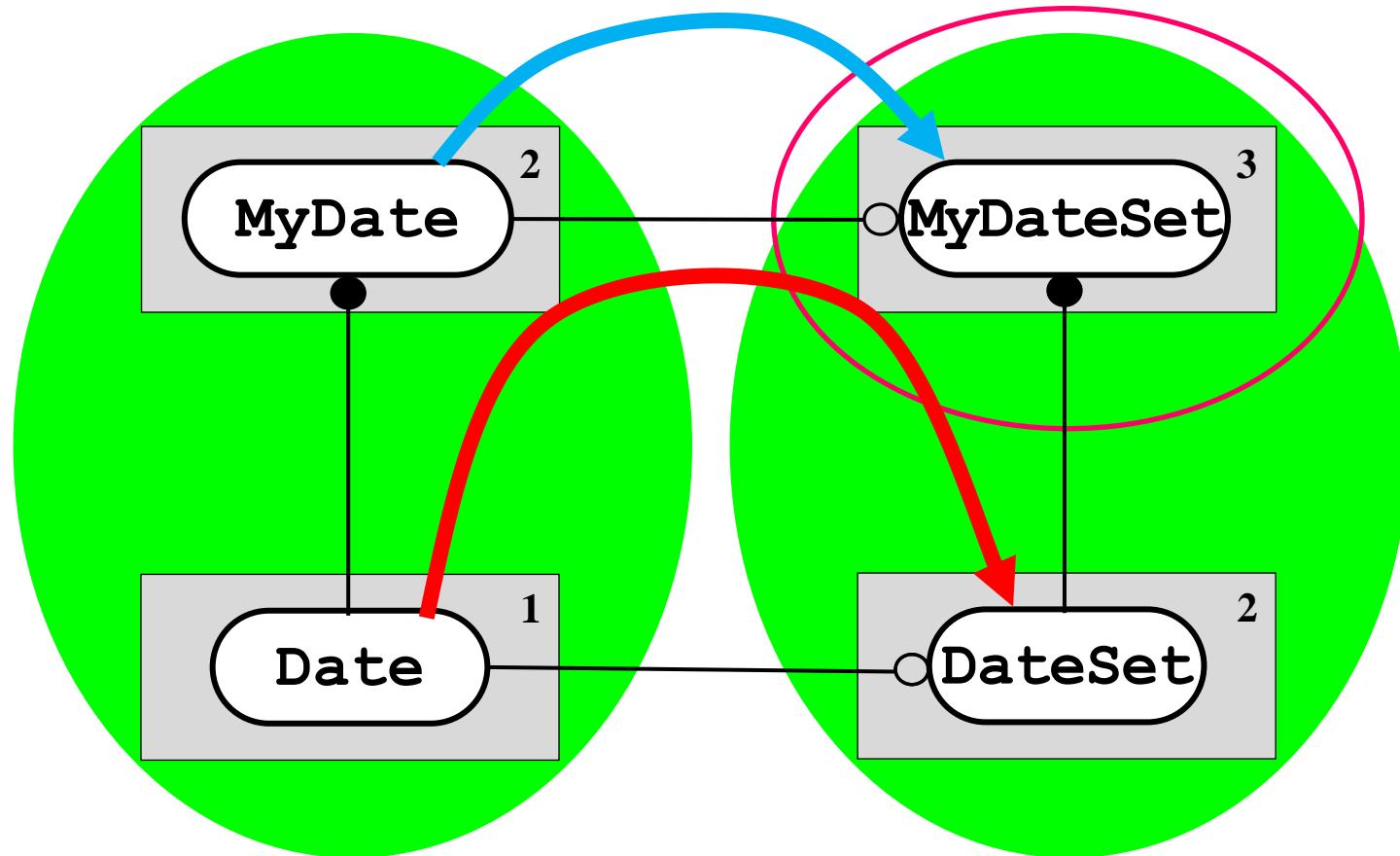
Multi-Component Wrapper



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

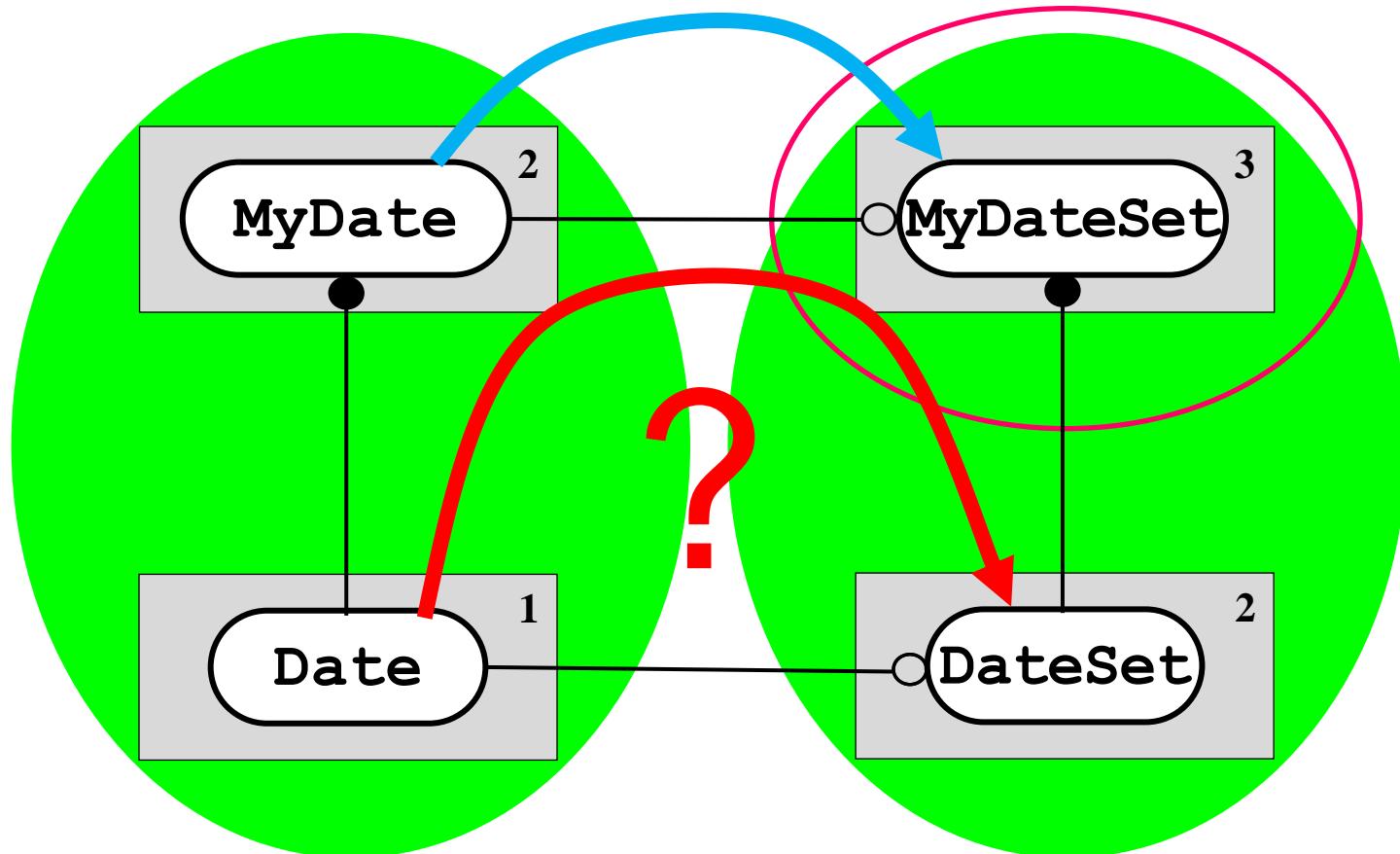
Multi-Component Wrapper



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

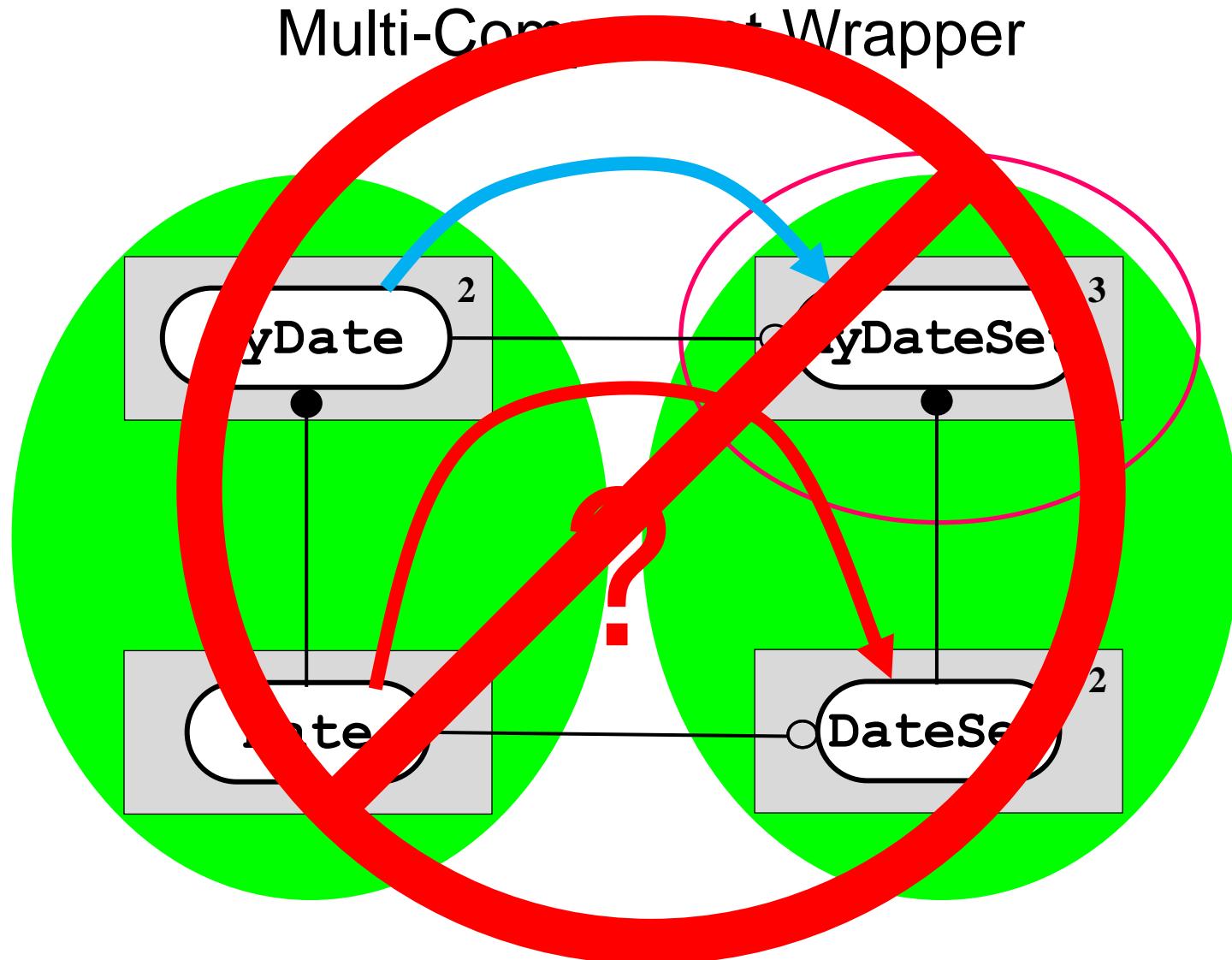
Multi-Component Wrapper



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

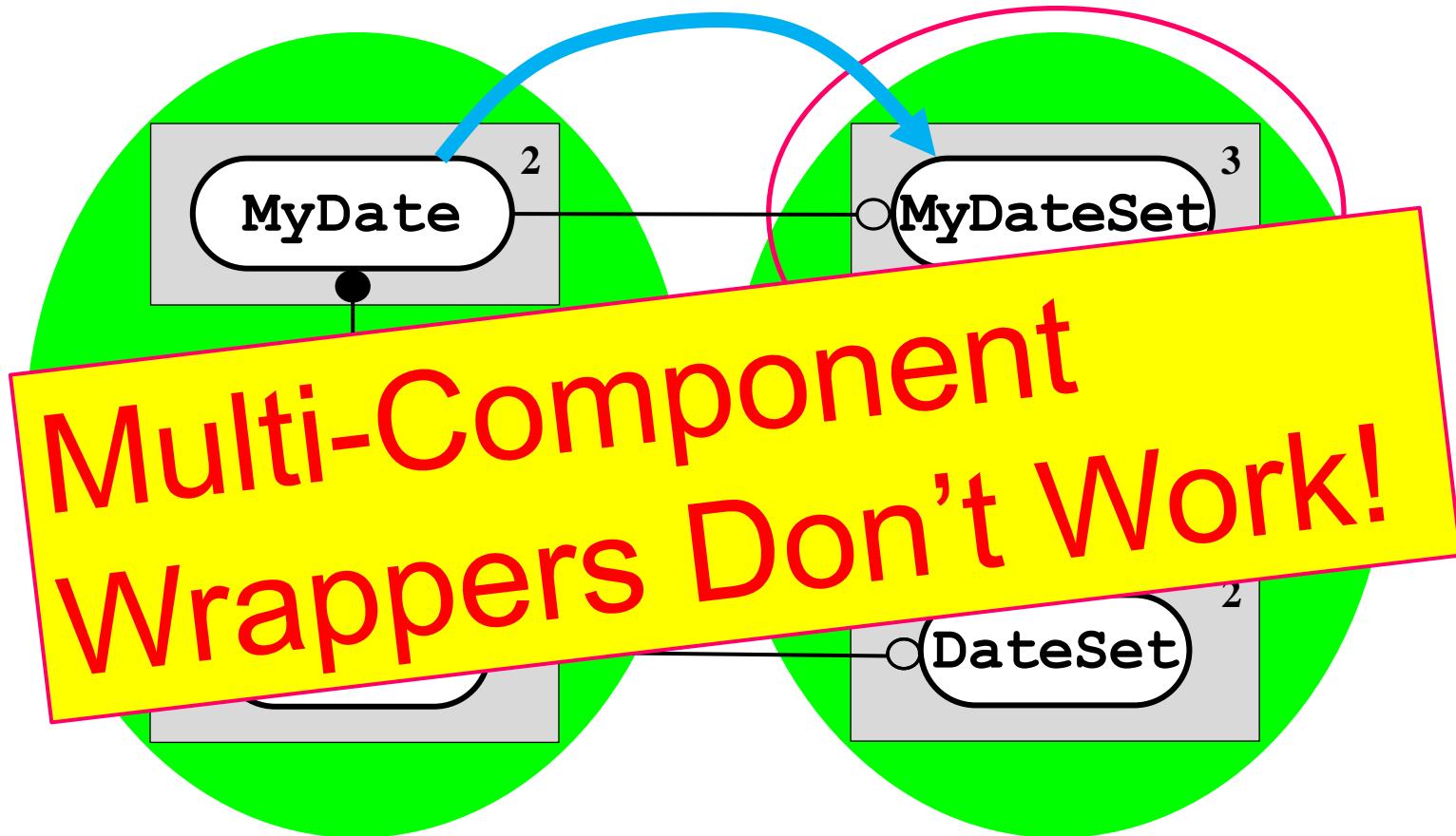
Multi-Component → Wrapper



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

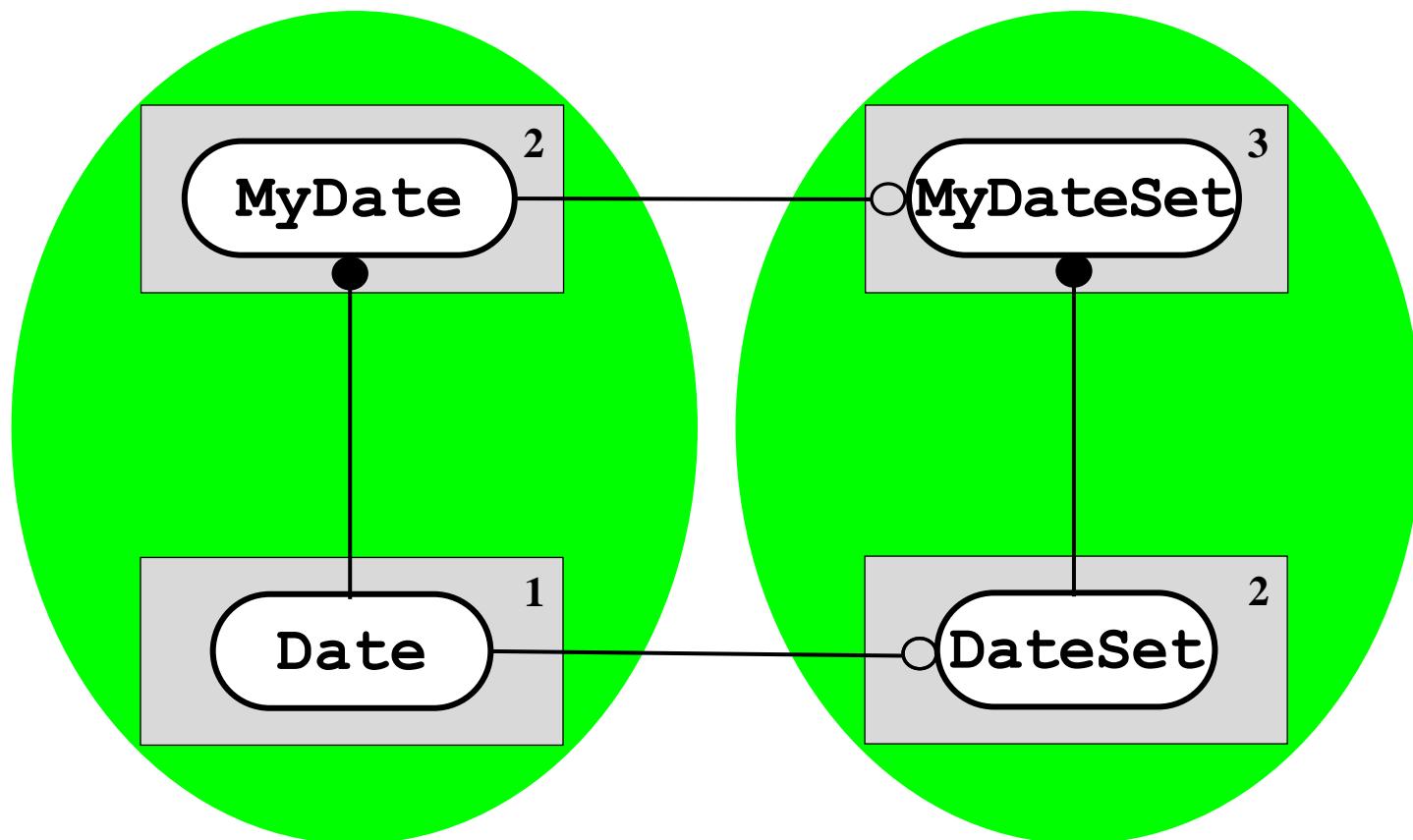
Multi-Component Wrapper



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

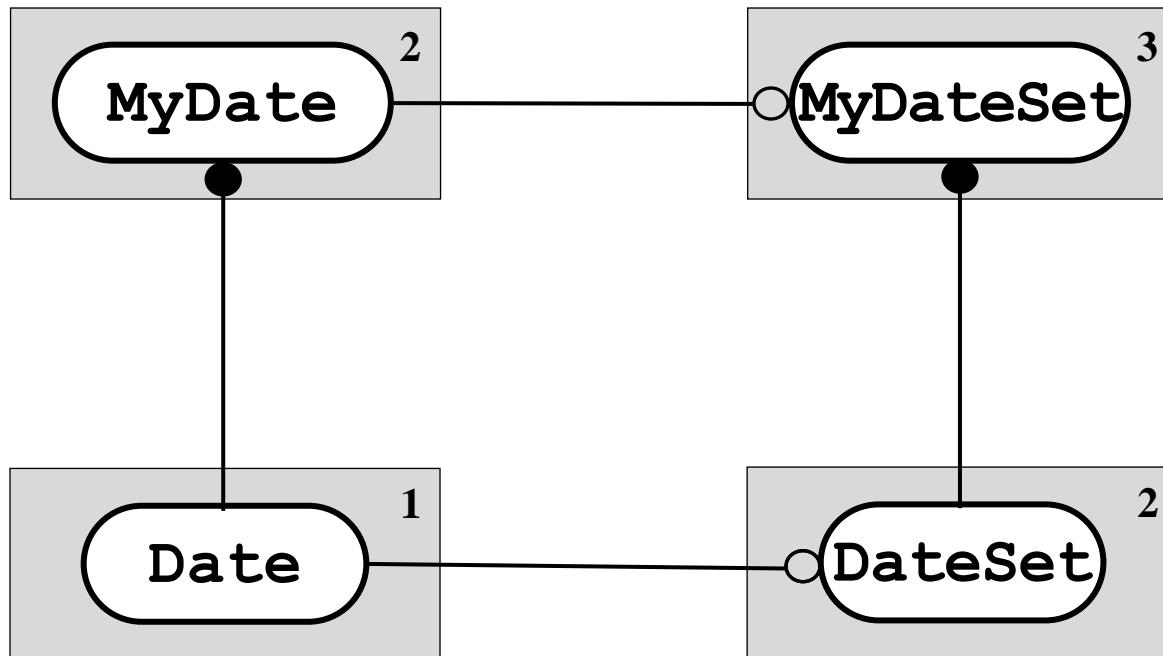
Multi-Component Wrapper



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

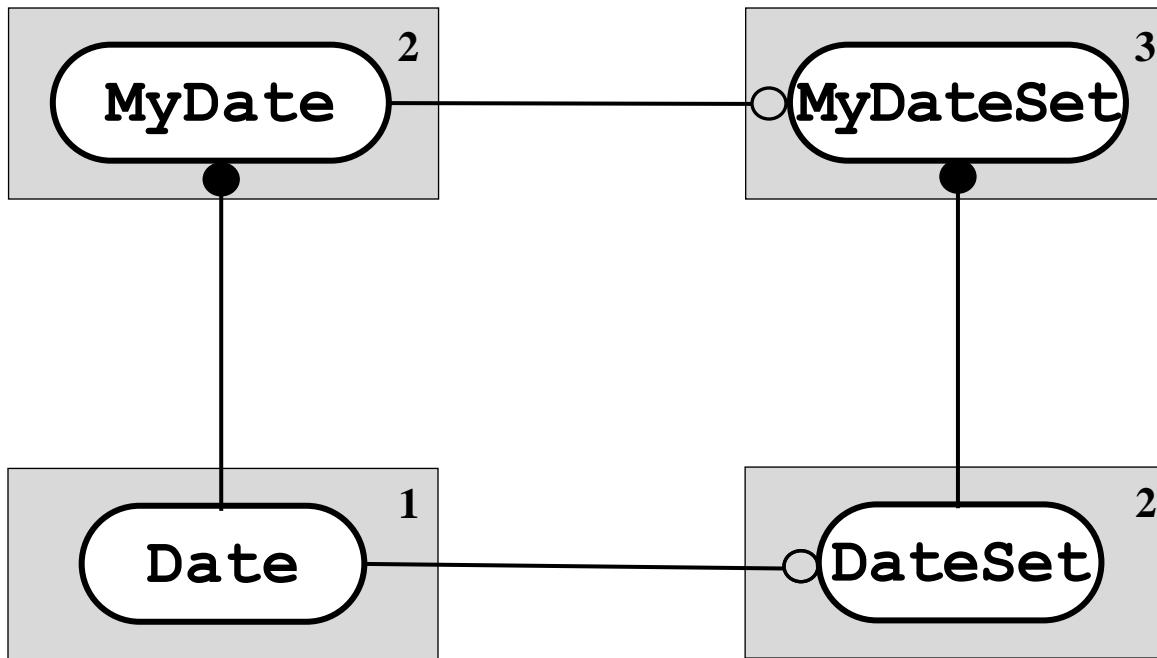
Multi-Component Wrapper



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

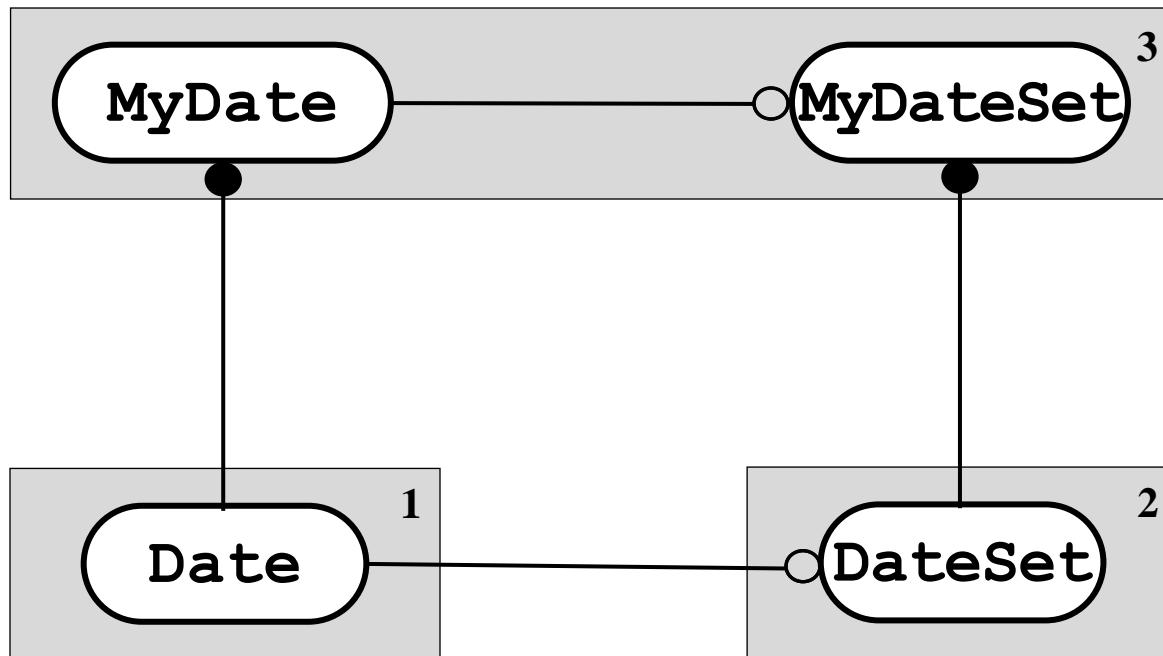
Single-Component Wrapper



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

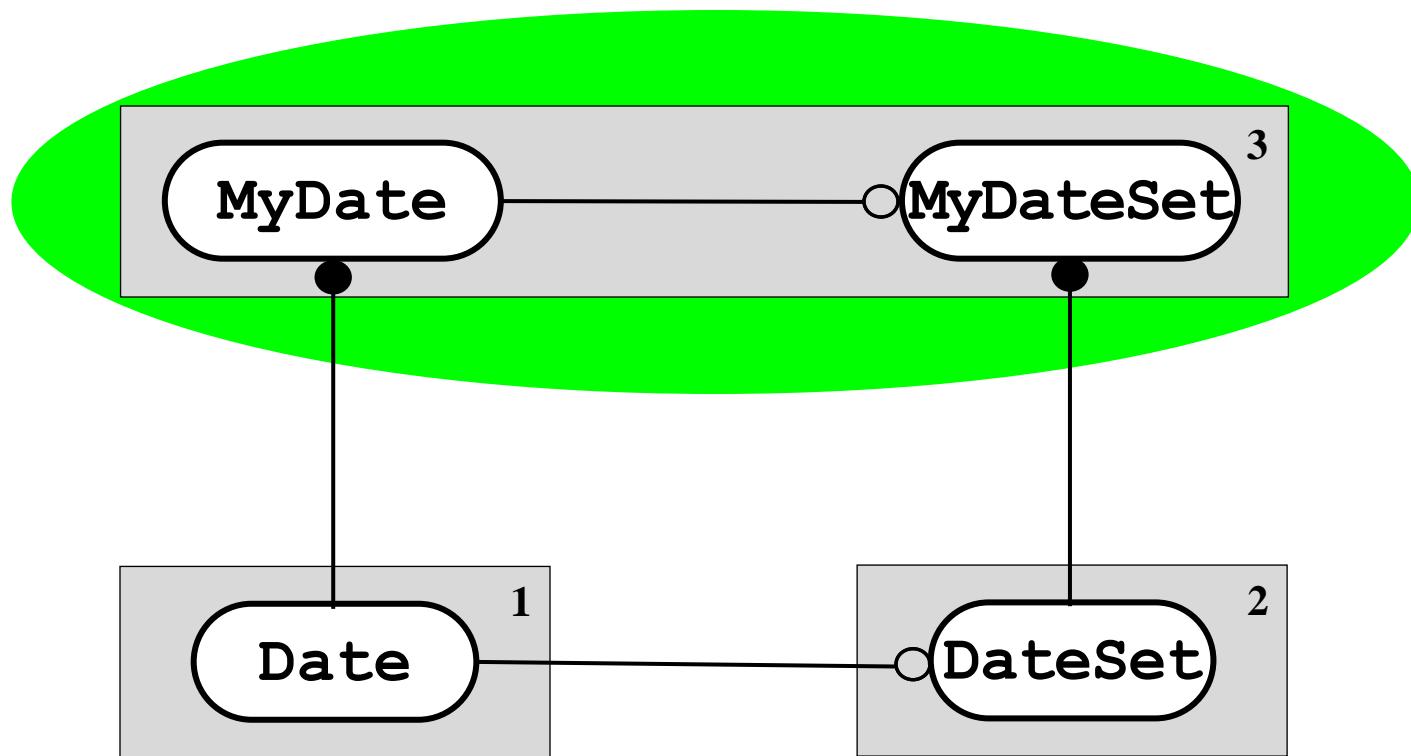
Single-Component Wrapper



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

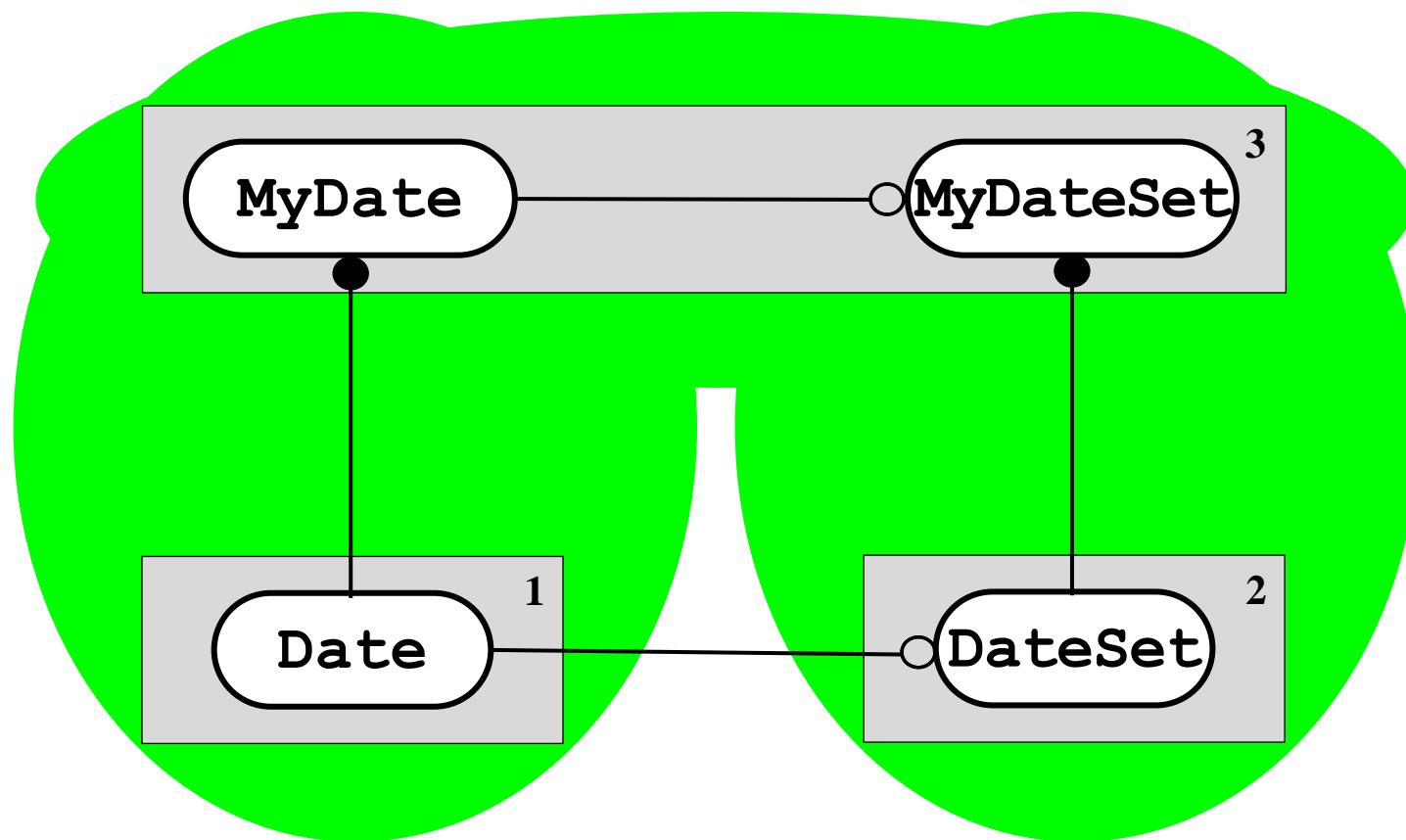
Single-Component Wrapper



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

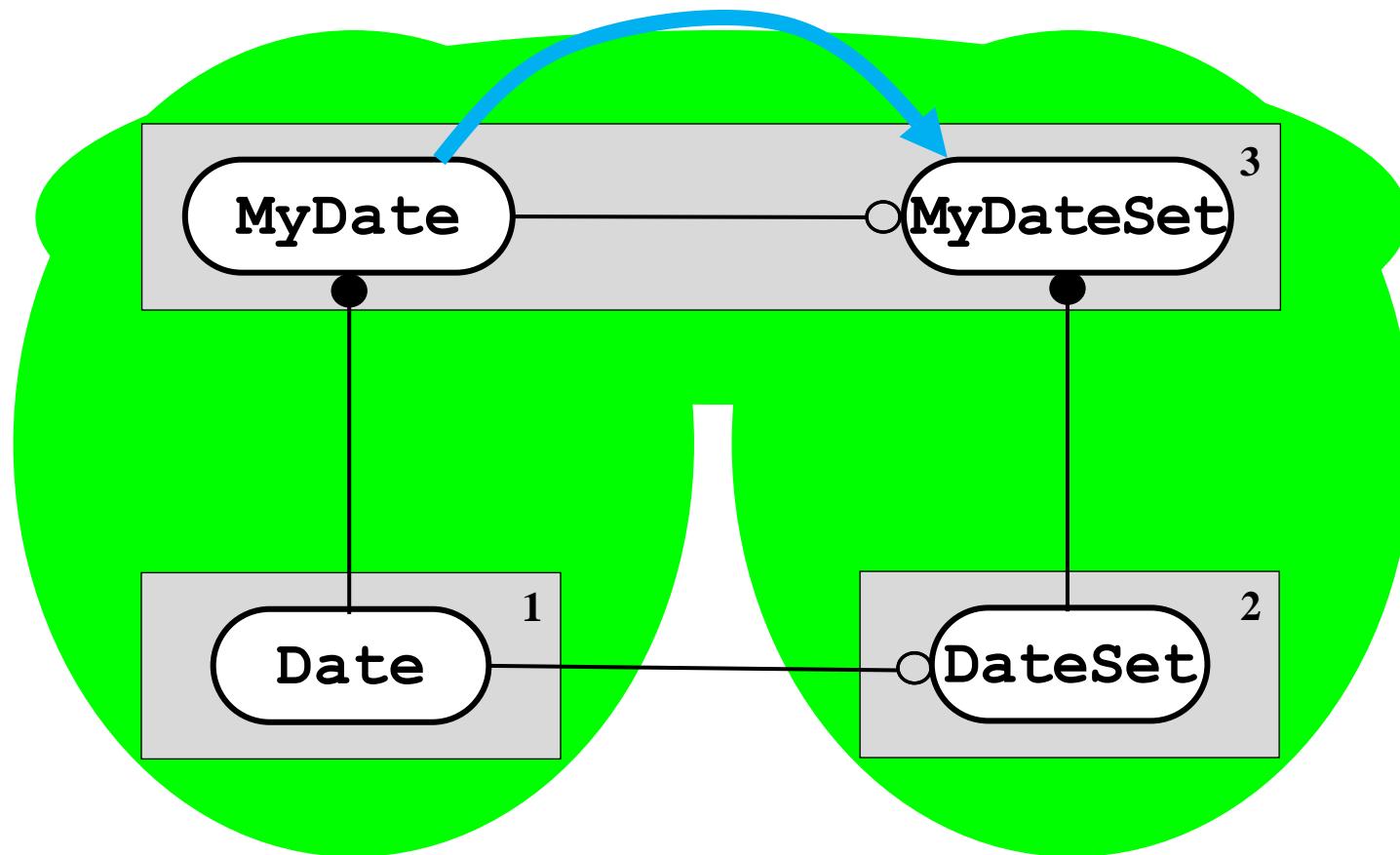
Single-Component Wrapper



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

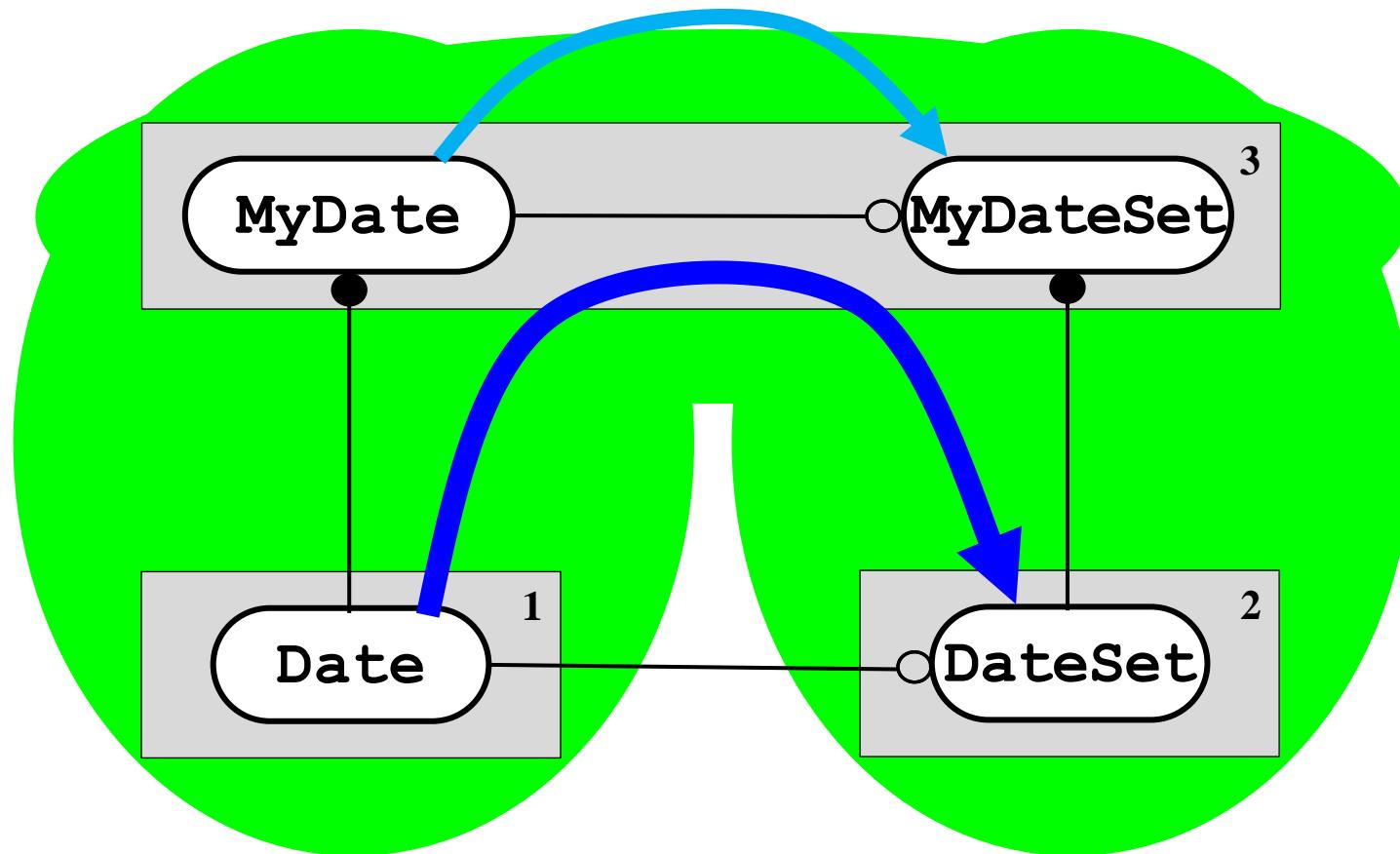
Single-Component Wrapper



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

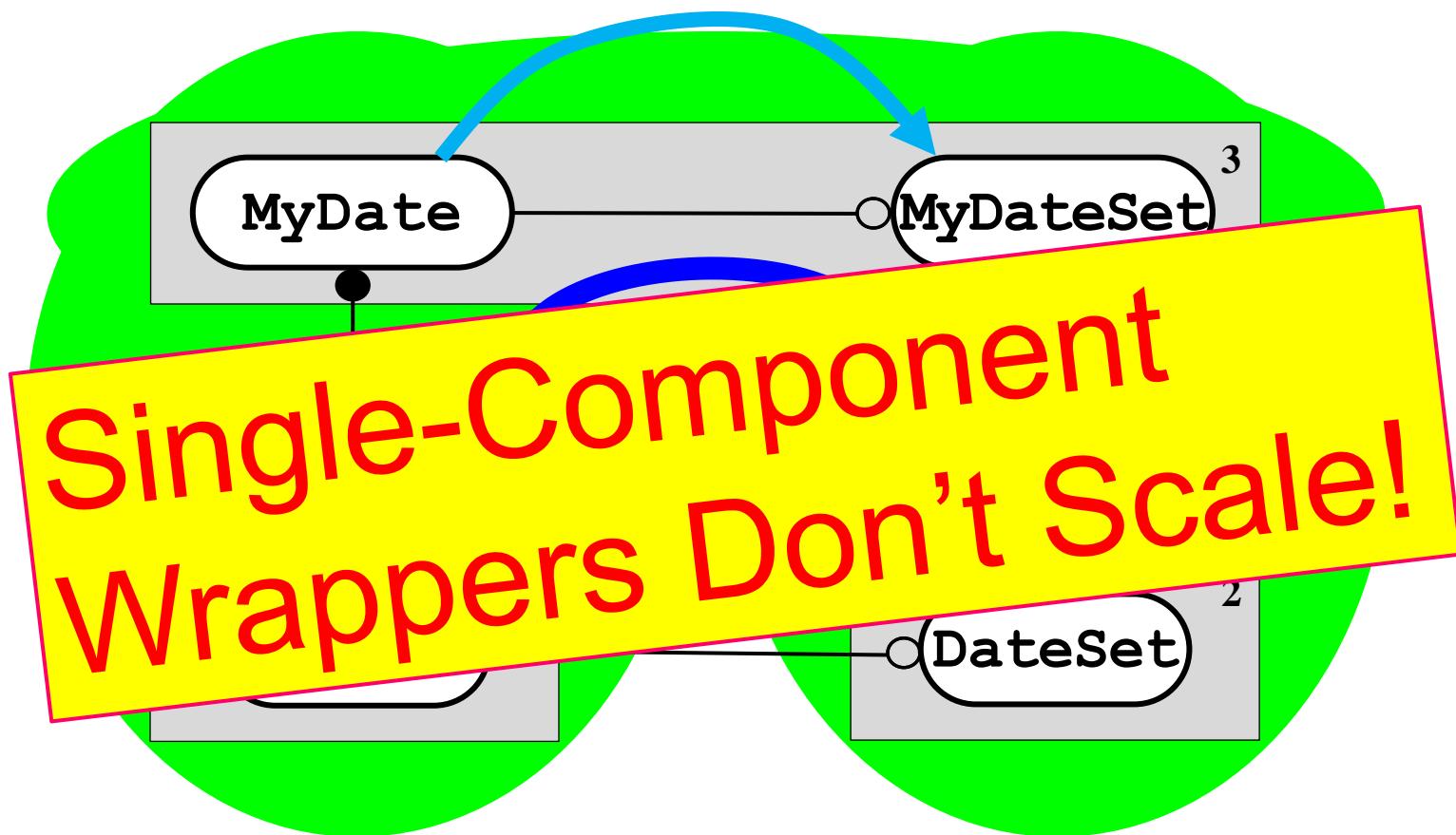
Single-Component Wrapper



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

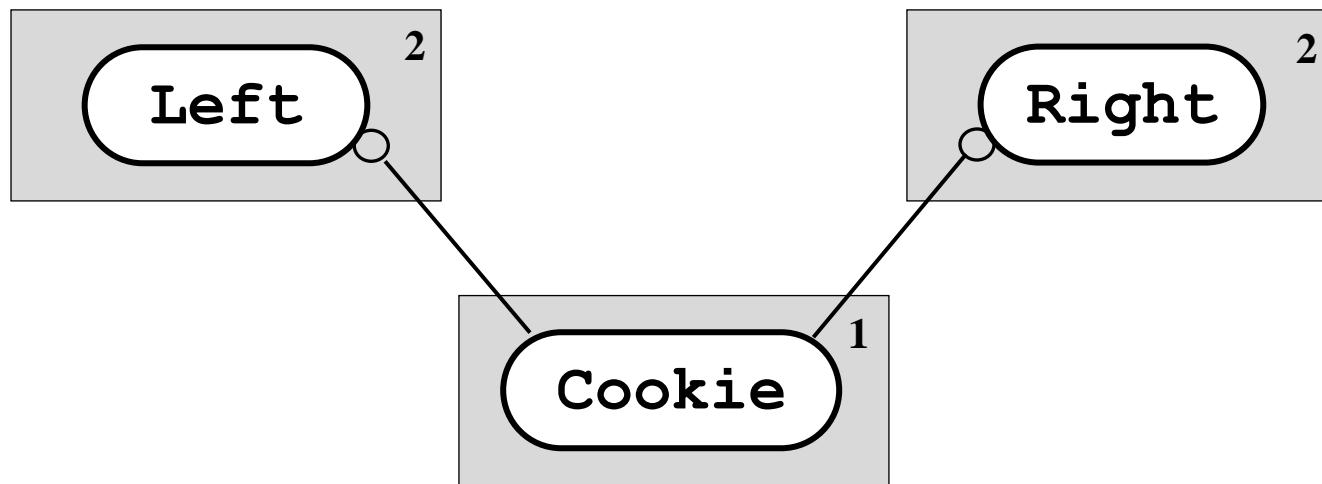
Single-Component Wrapper



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

Hiding Header Files

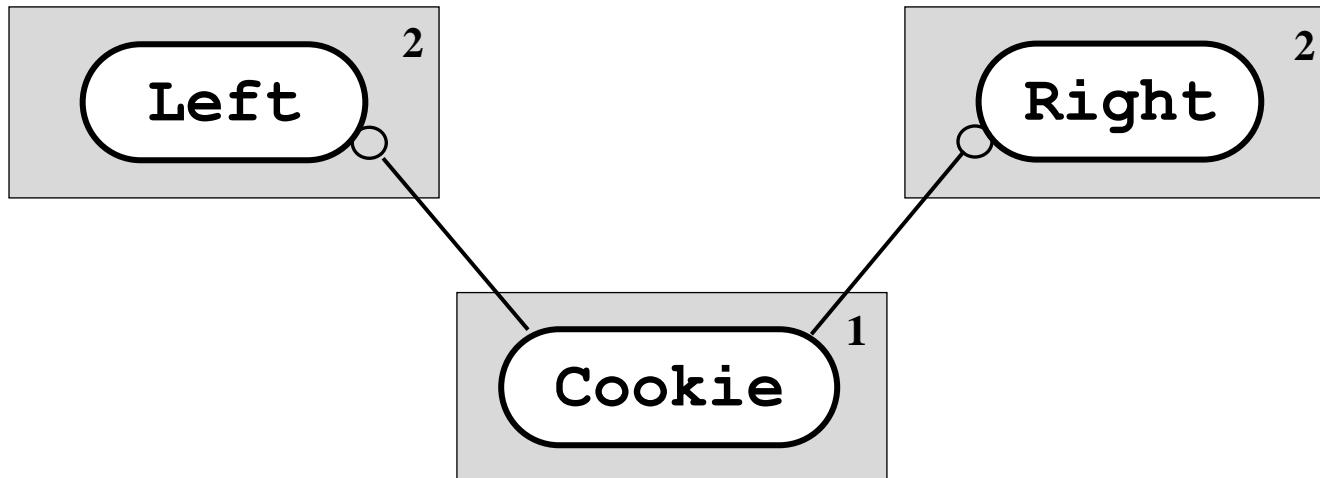


2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

Hiding Header Files

```
// left.h
#include <cookie.h>
class Left {
    // ...
    void setC(const Cookie& c);
    // ...
};
```



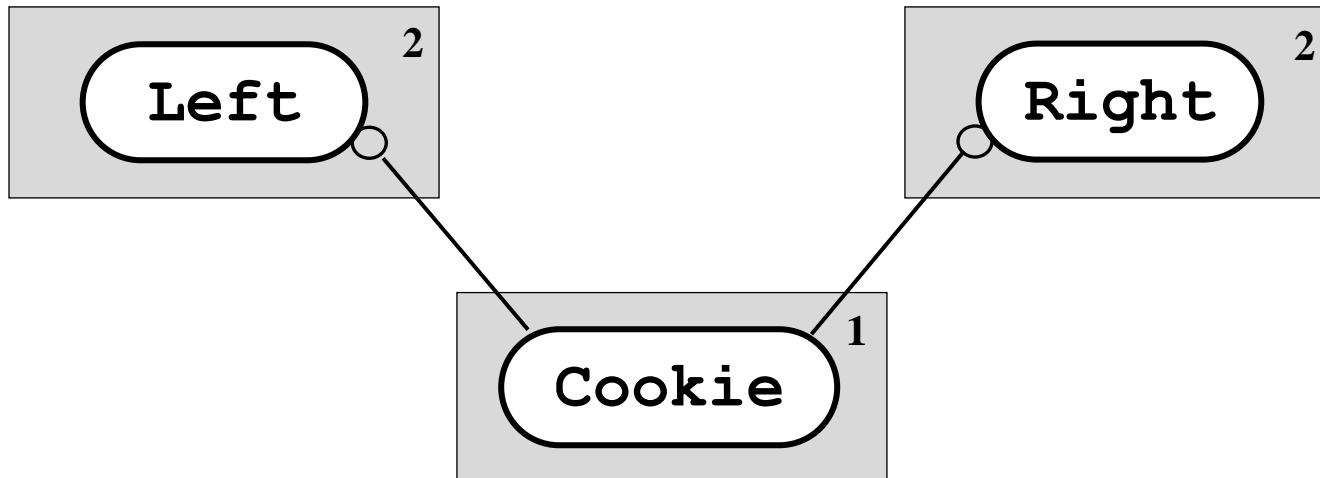
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

Hiding Header Files

```
// left.h
#include <cookie.h>
class Left {
    // ...
    void setC(const Cookie& c);
    // ...
};
```

```
// right.h
#include <cookie.h>
class Right {
    // ...
    const Cookie& getC() const;
    // ...
};
```



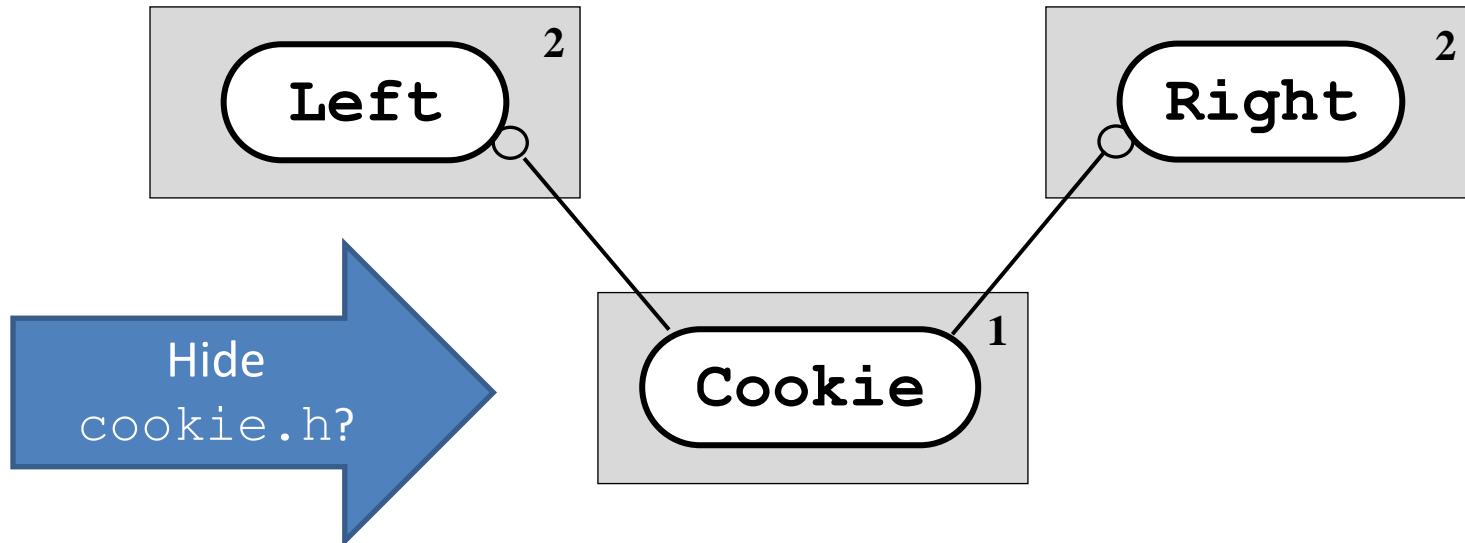
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

Hiding Header Files

```
// left.h
#include <cookie.h>
class Left {
    // ...
    void setC(const Cookie& c);
    // ...
};
```

```
// right.h
#include <cookie.h>
class Right {
    // ...
    const Cookie& getC() const;
    // ...
};
```



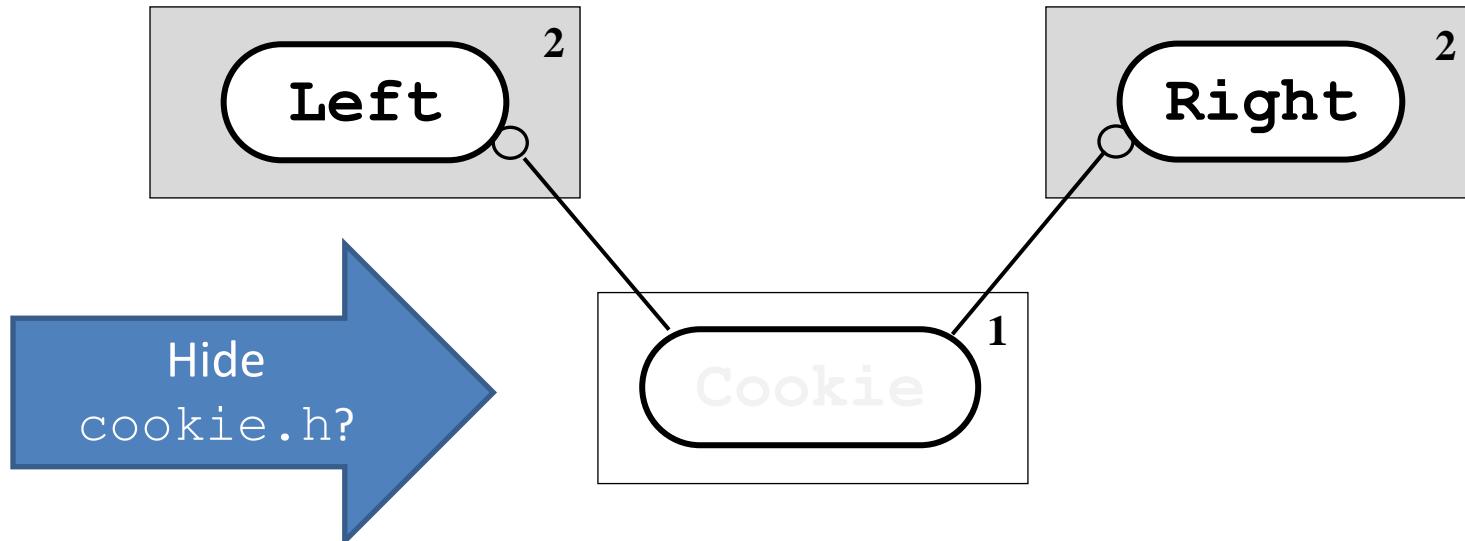
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

Hiding Header Files

```
// left.h
#include <cookie.h>
class Left {
    // ...
    void setC(const Cookie& c);
    // ...
};
```

```
// right.h
#include <cookie.h>
class Right {
    // ...
    const Cookie& getC() const;
    // ...
};
```



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

Hiding Header Files

Bad Idea:

Hide
cookie.h?



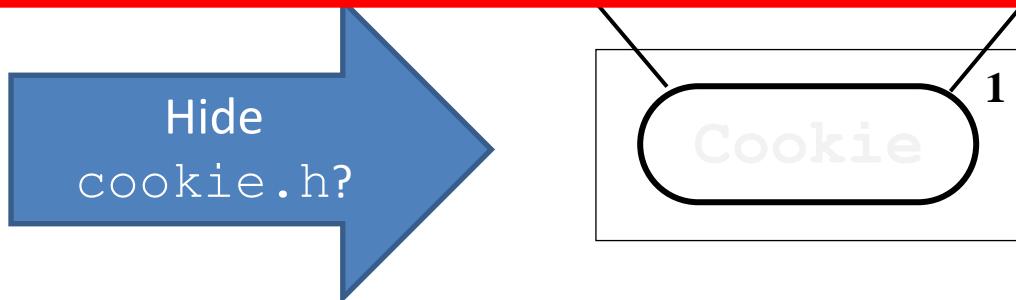
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

Hiding Header Files

Bad Idea:

(1) Convolves architecture with deployment.



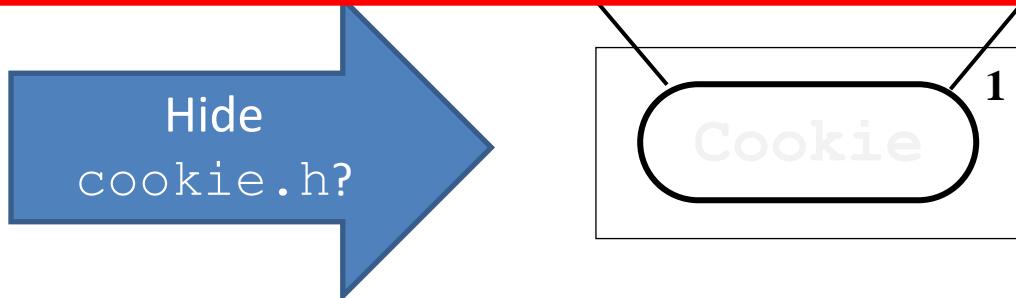
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

Hiding Header Files

Bad Idea:

- (1) Convolves architecture with deployment.
- (2) Inhibits side-by-side reuse of the “hidden” component.



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

Hiding Header Files

Bad Idea:

- (1) Convolves architecture with deployment.
- (2) Inhibits side-by-side reuse of the “hidden” component.

Hide
cookie.h?



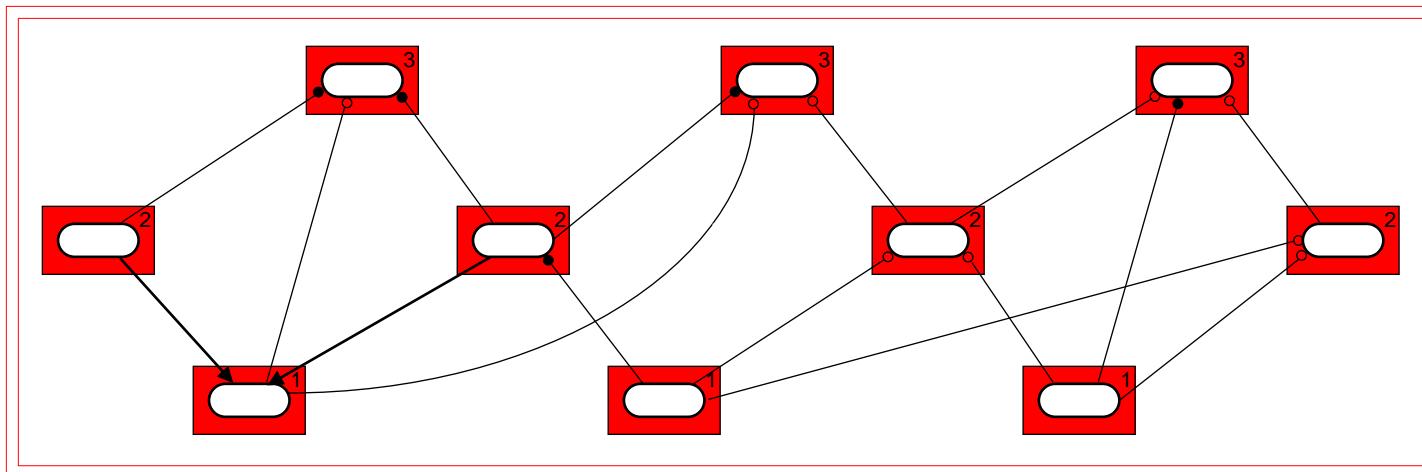
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation Wrapper Package

2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

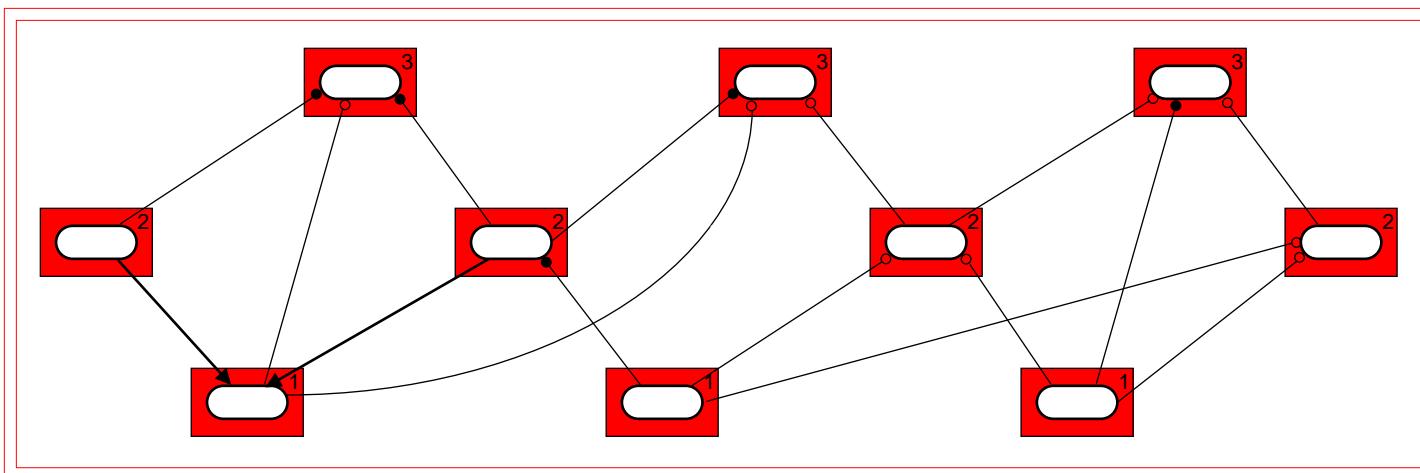
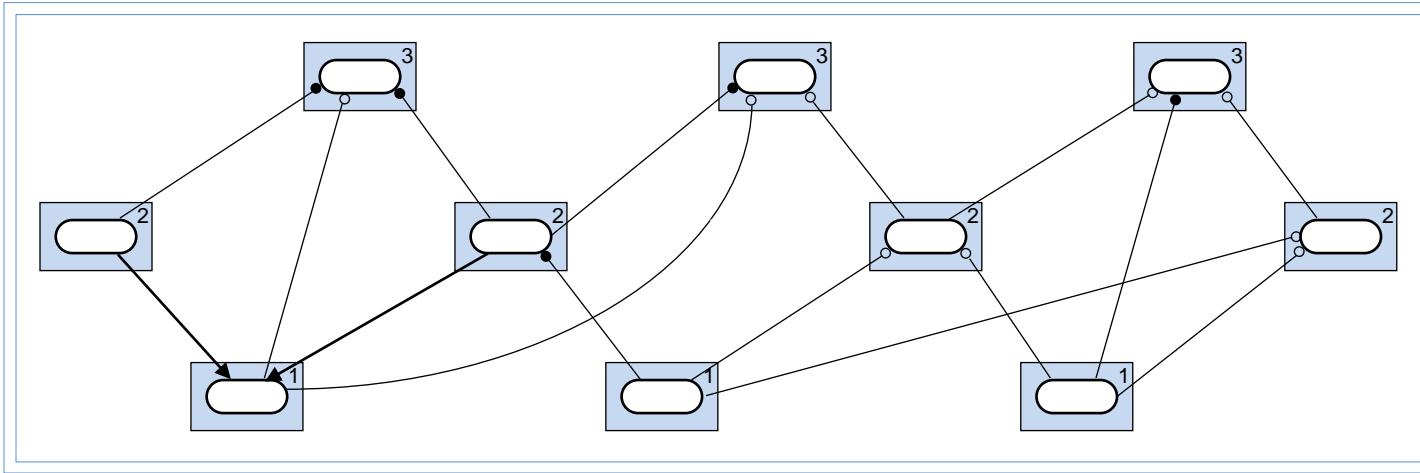
Wrapper Package



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

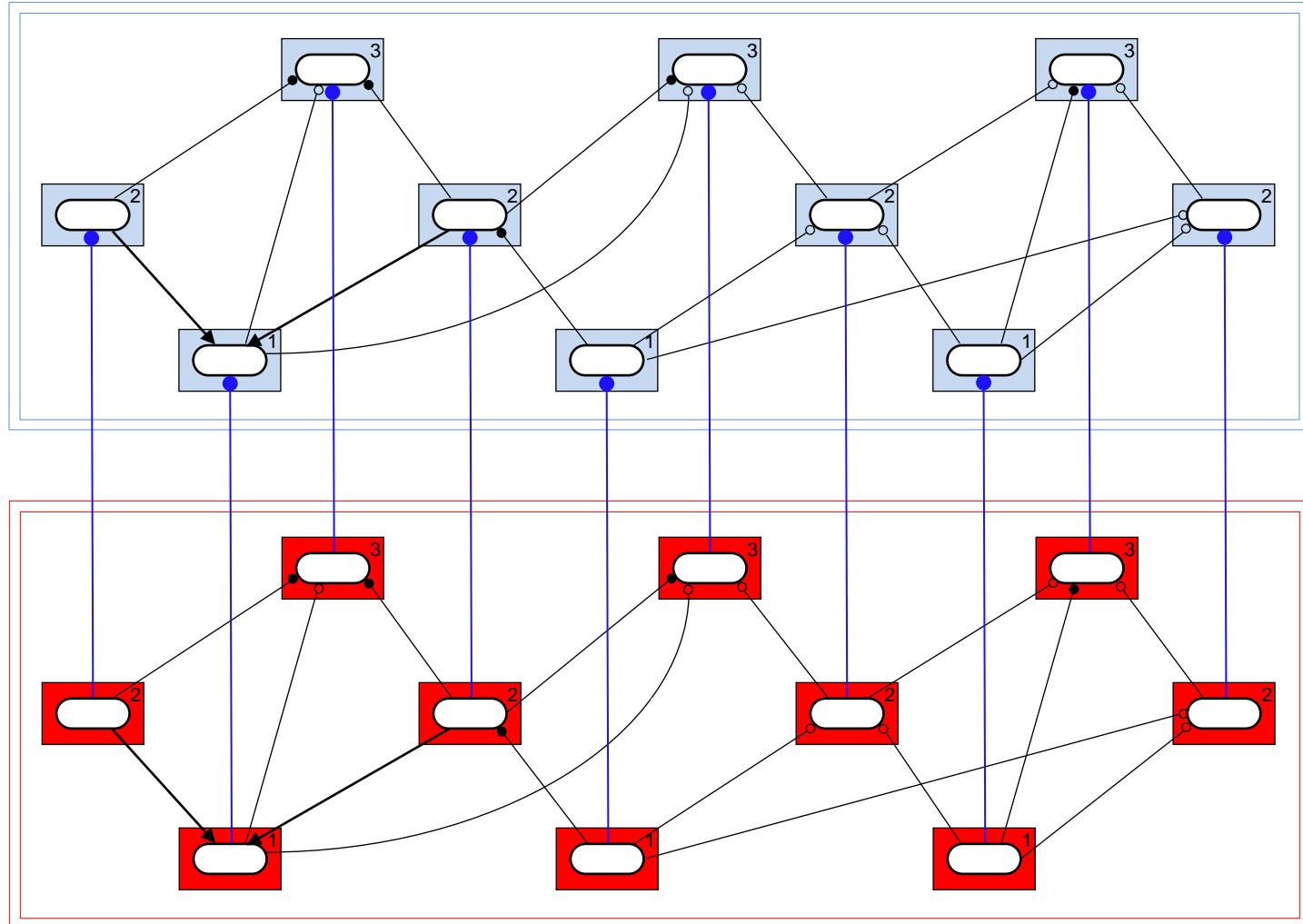
Wrapper Package



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

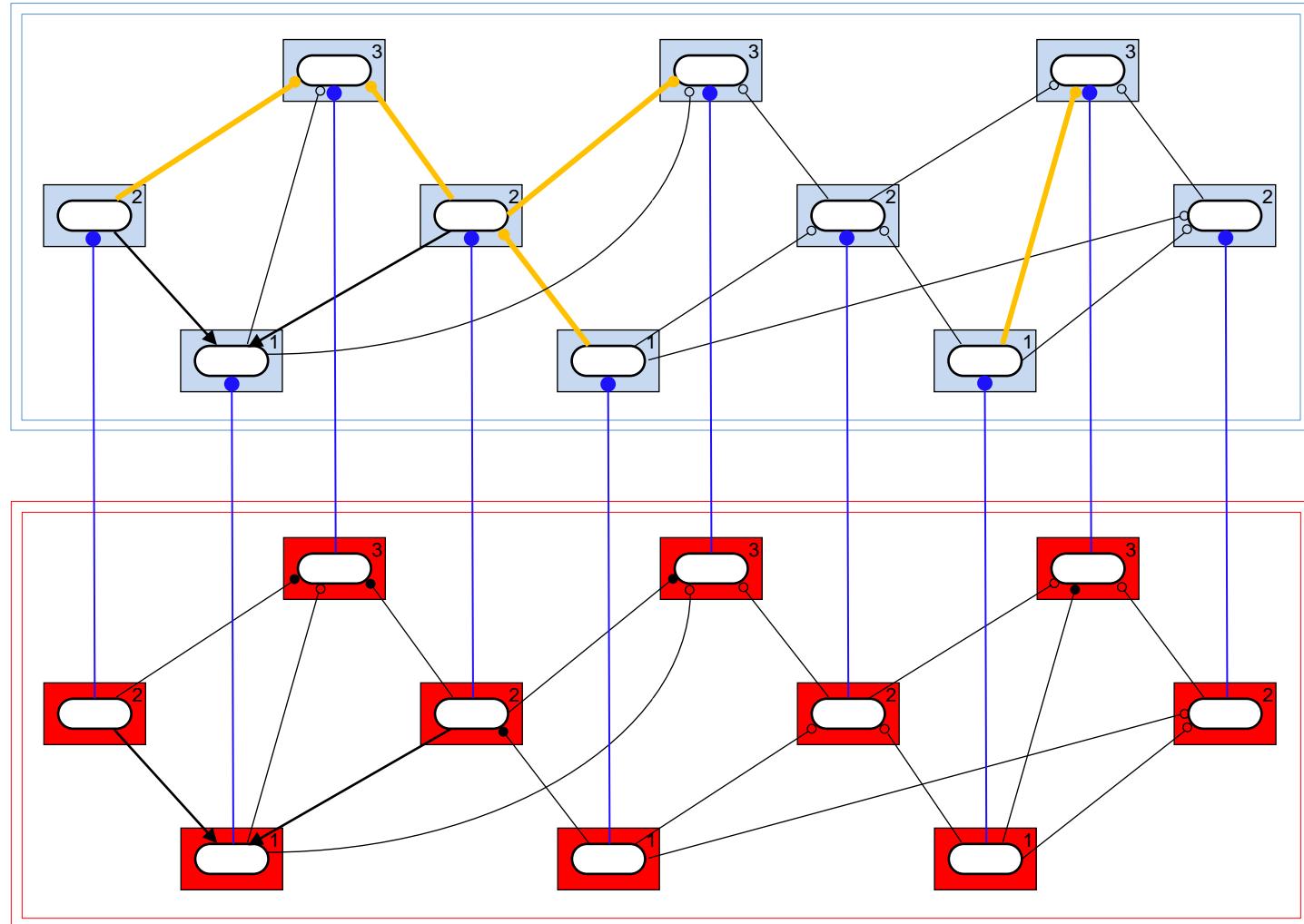
Wrapper Package



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

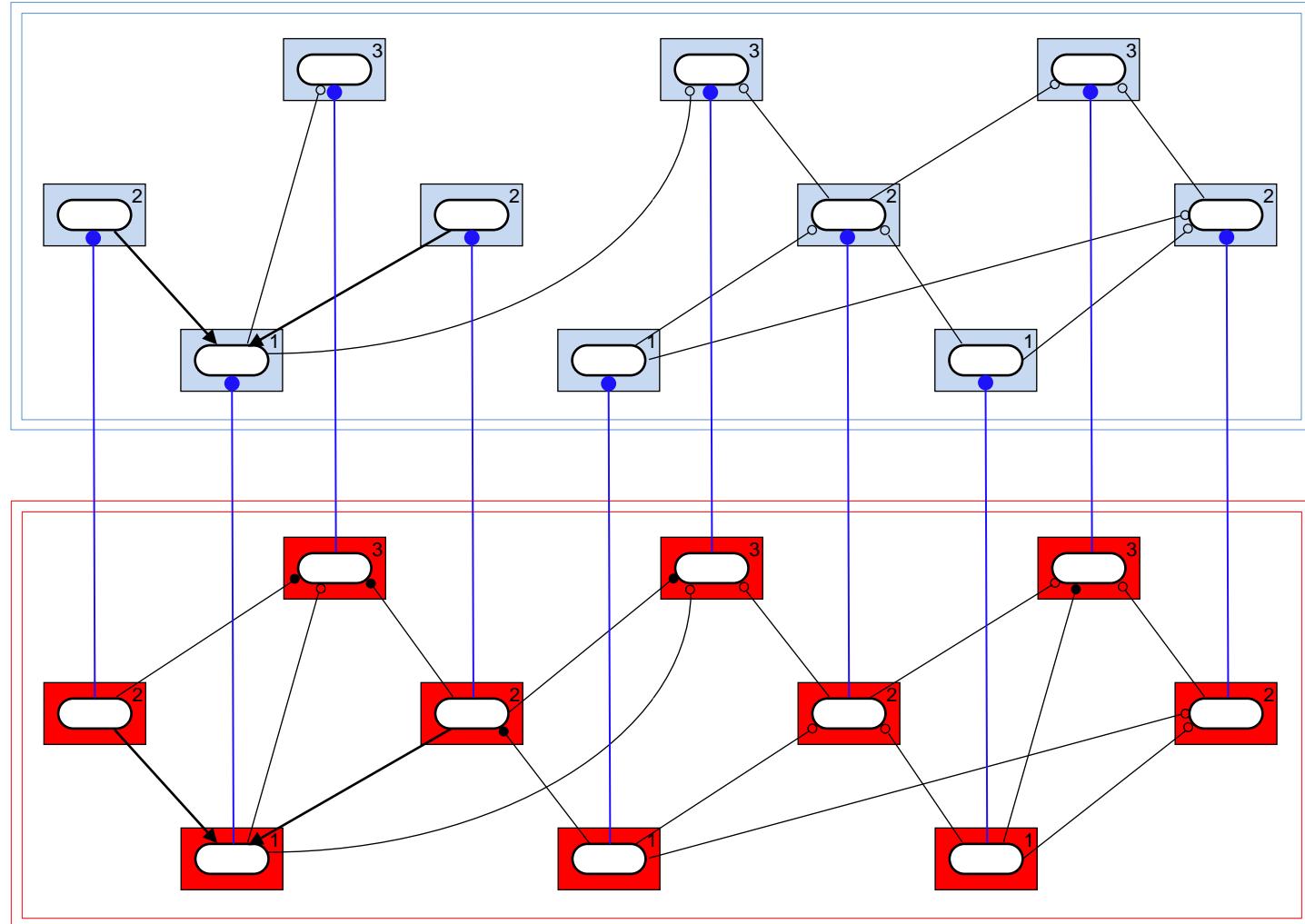
Wrapper Package



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

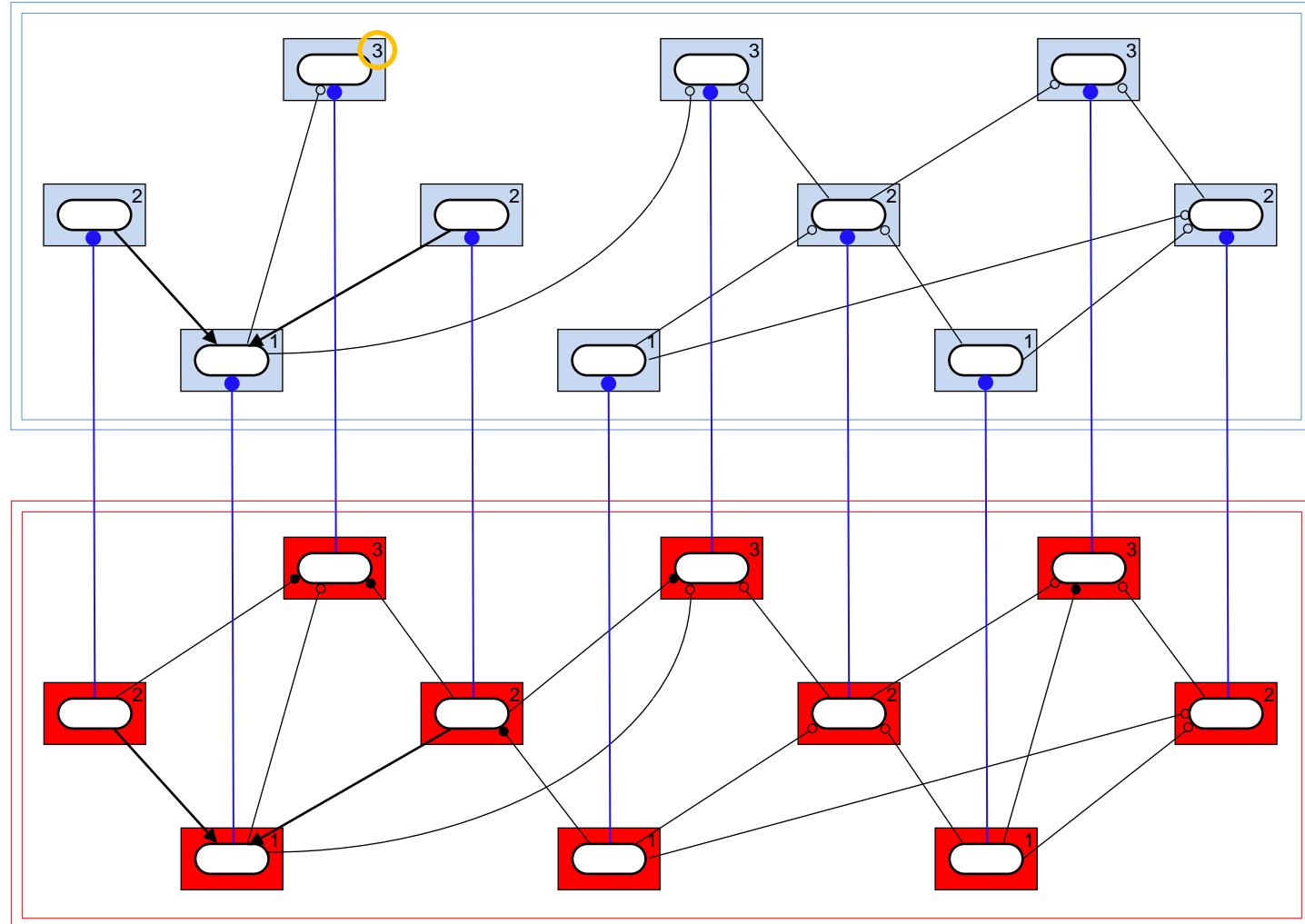
Wrapper Package



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

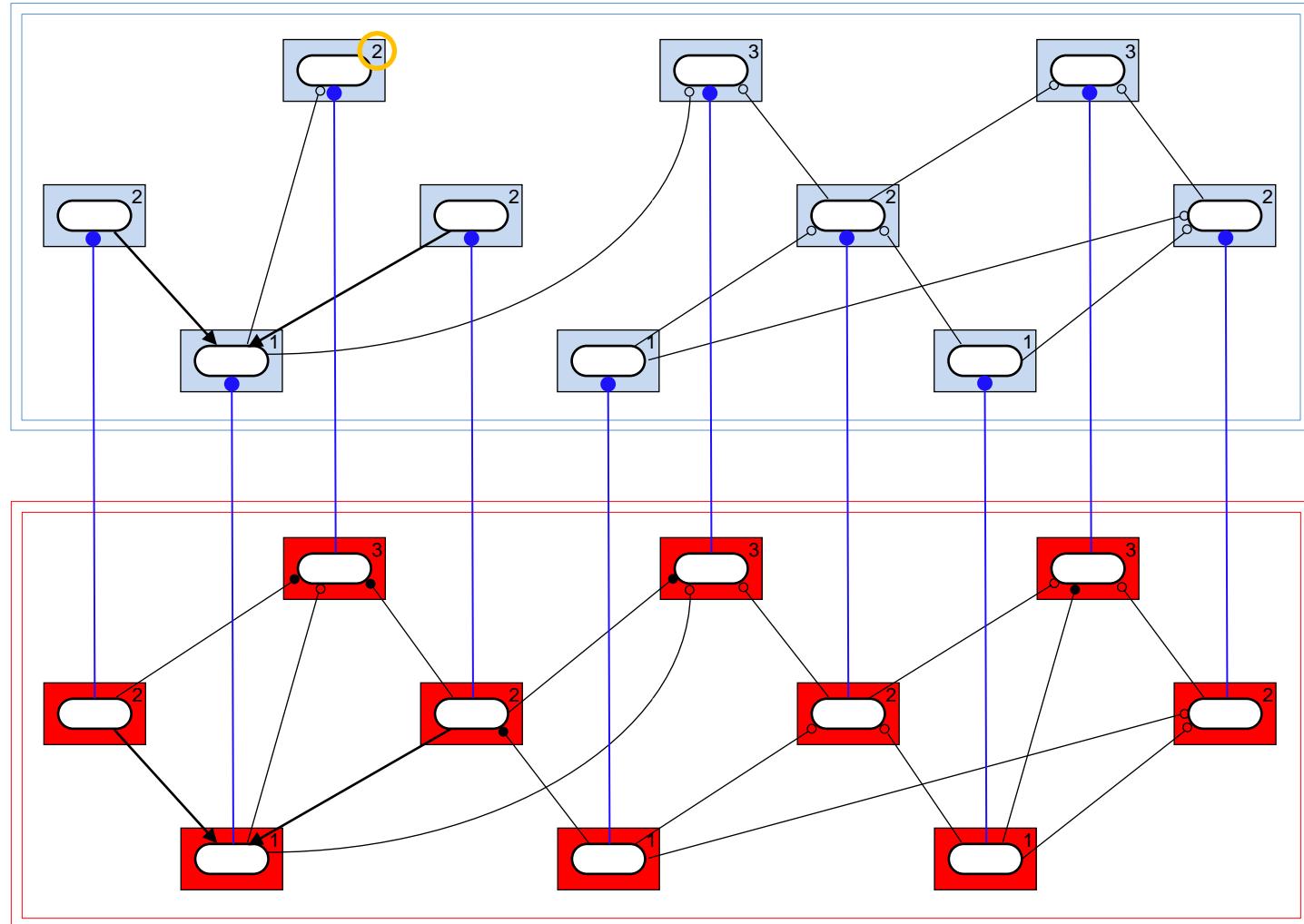
Wrapper Package



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

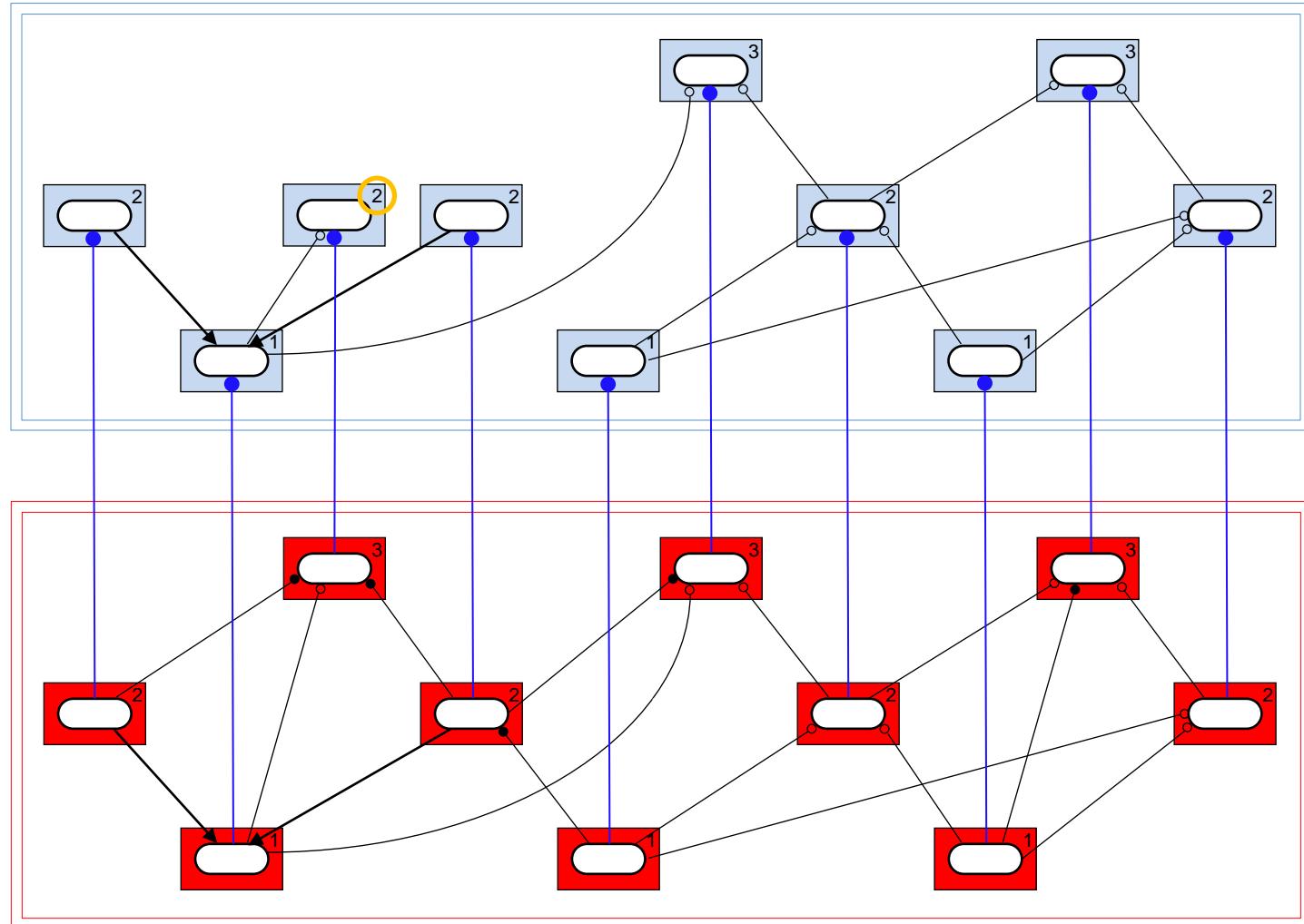
Wrapper Package



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

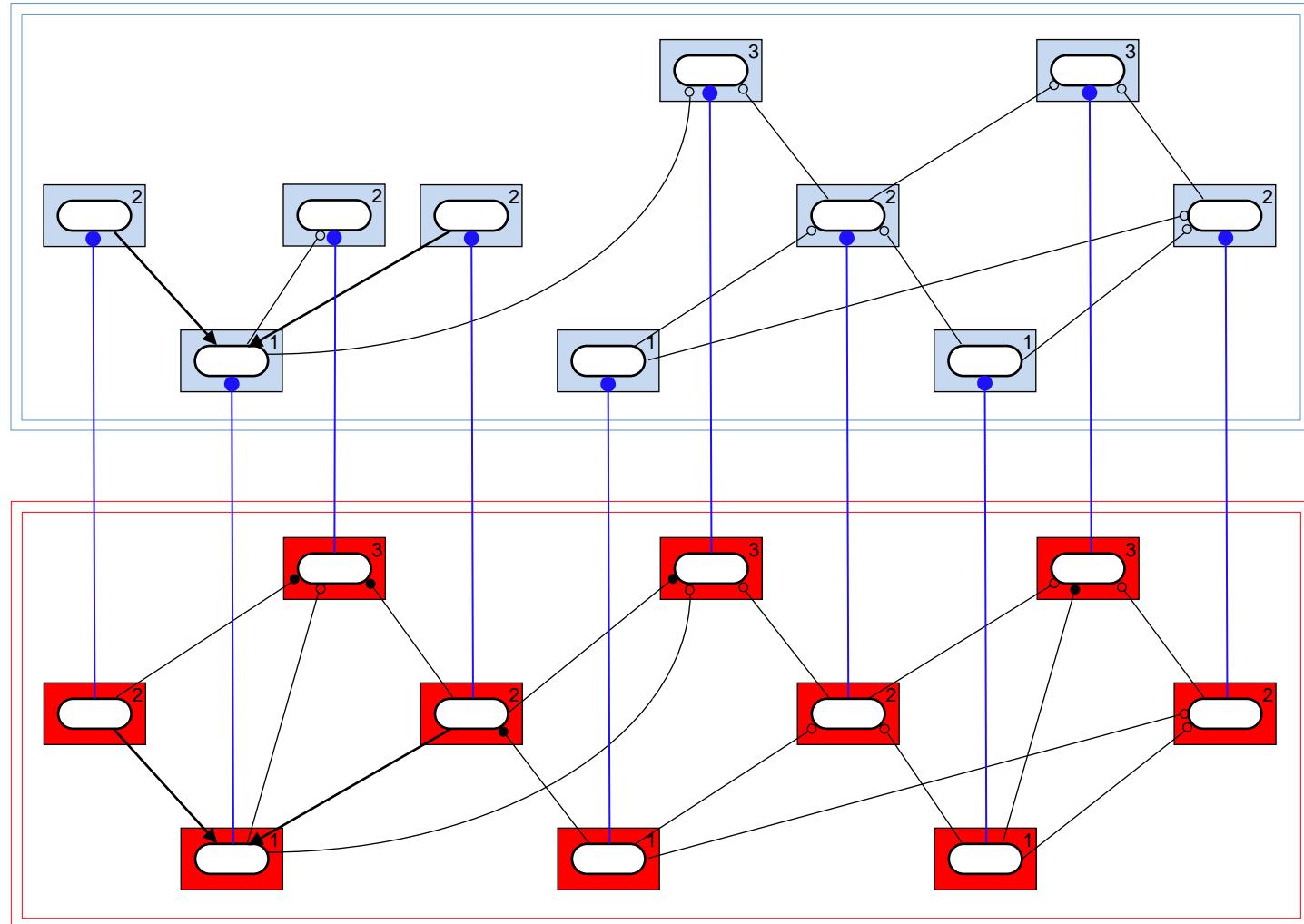
Wrapper Package



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

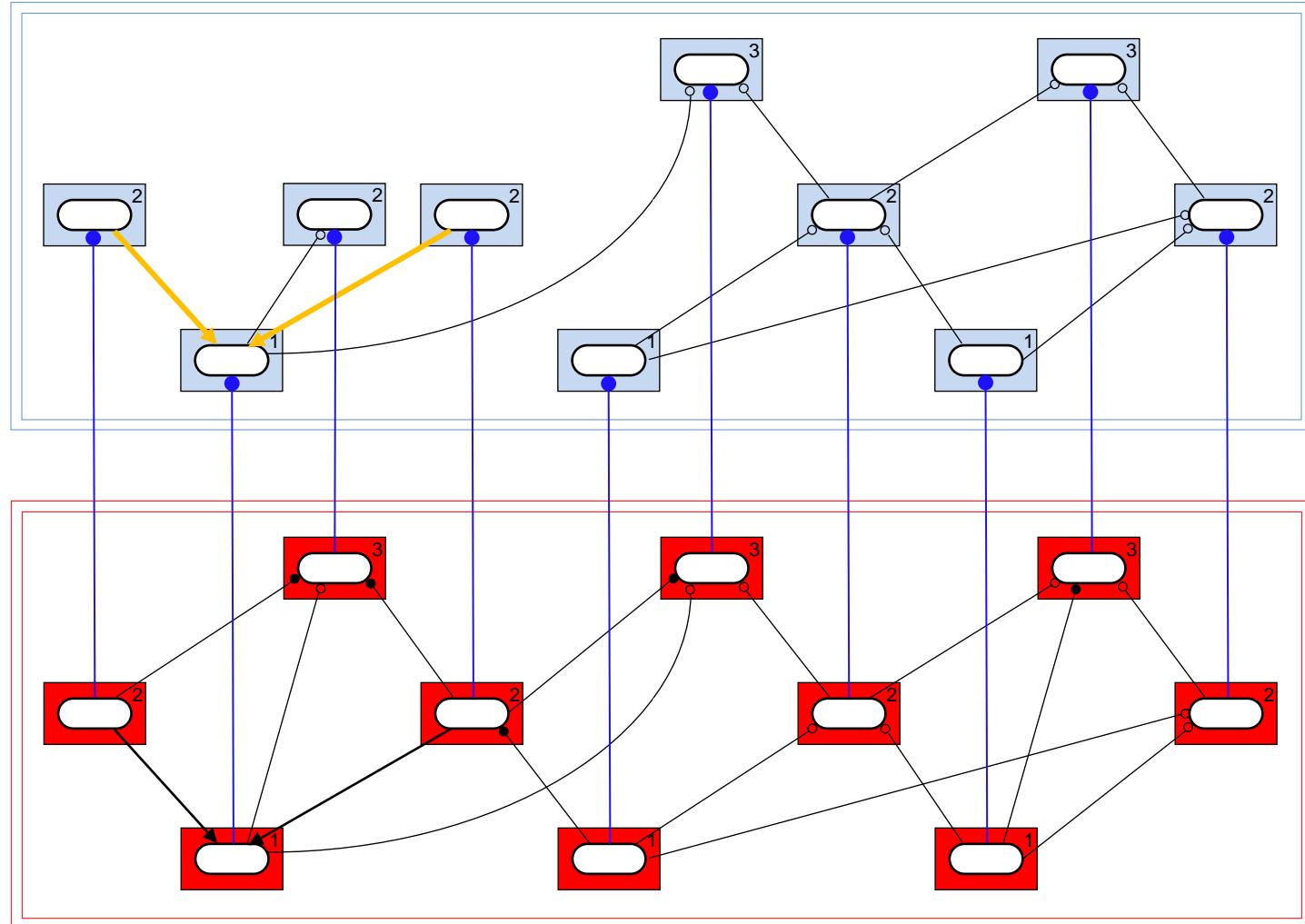
Wrapper Package



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

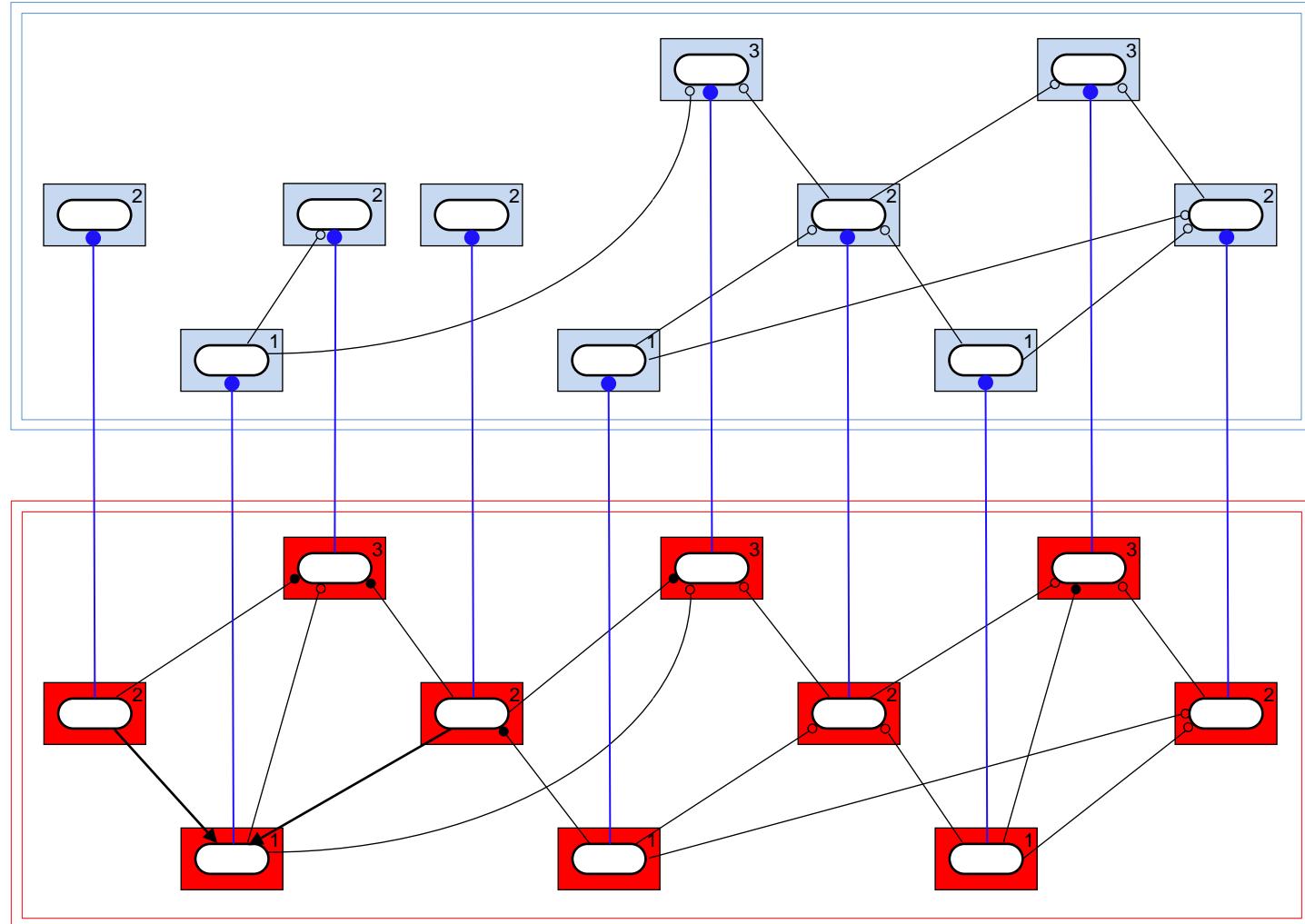
Wrapper Package



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

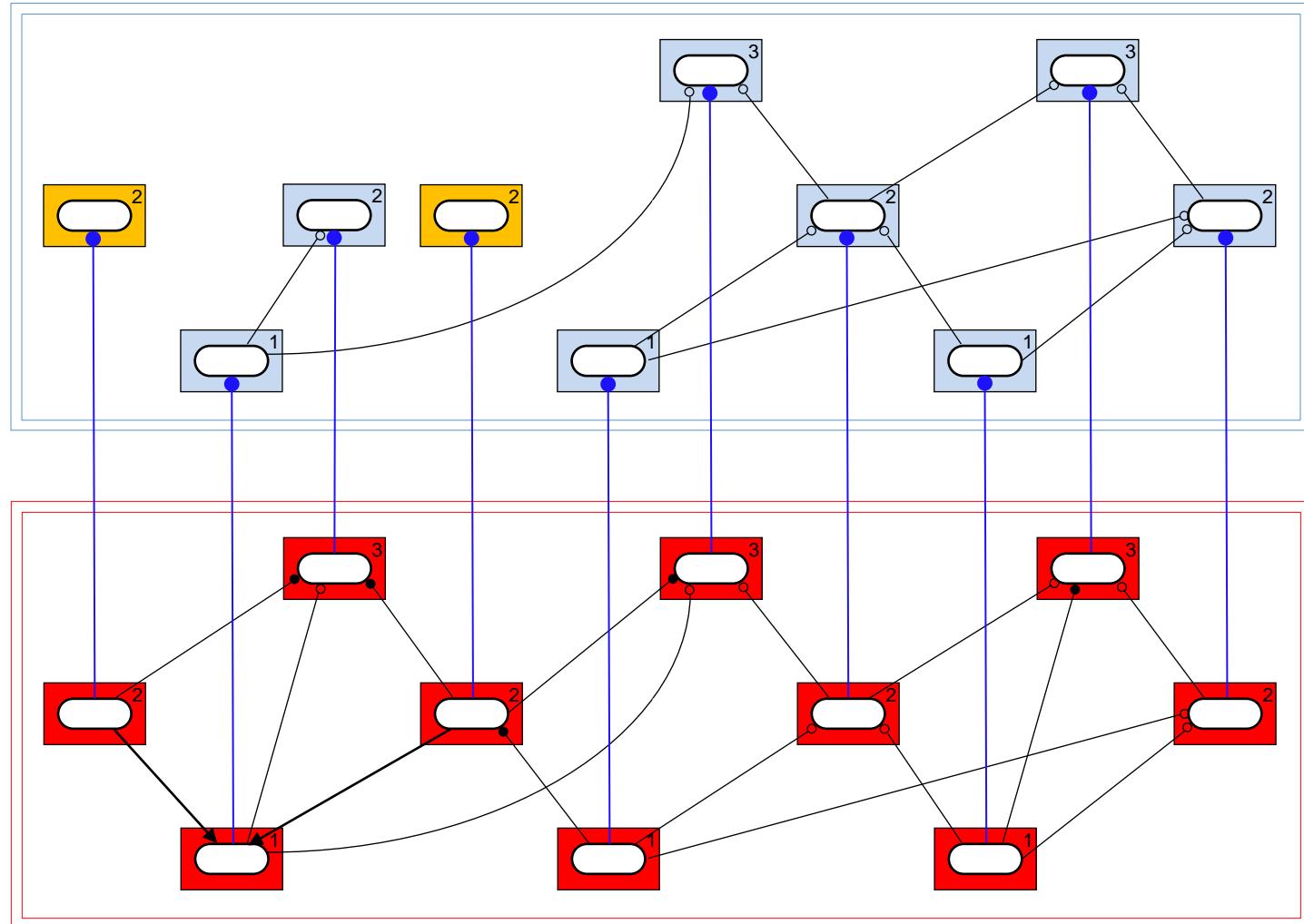
Wrapper Package



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

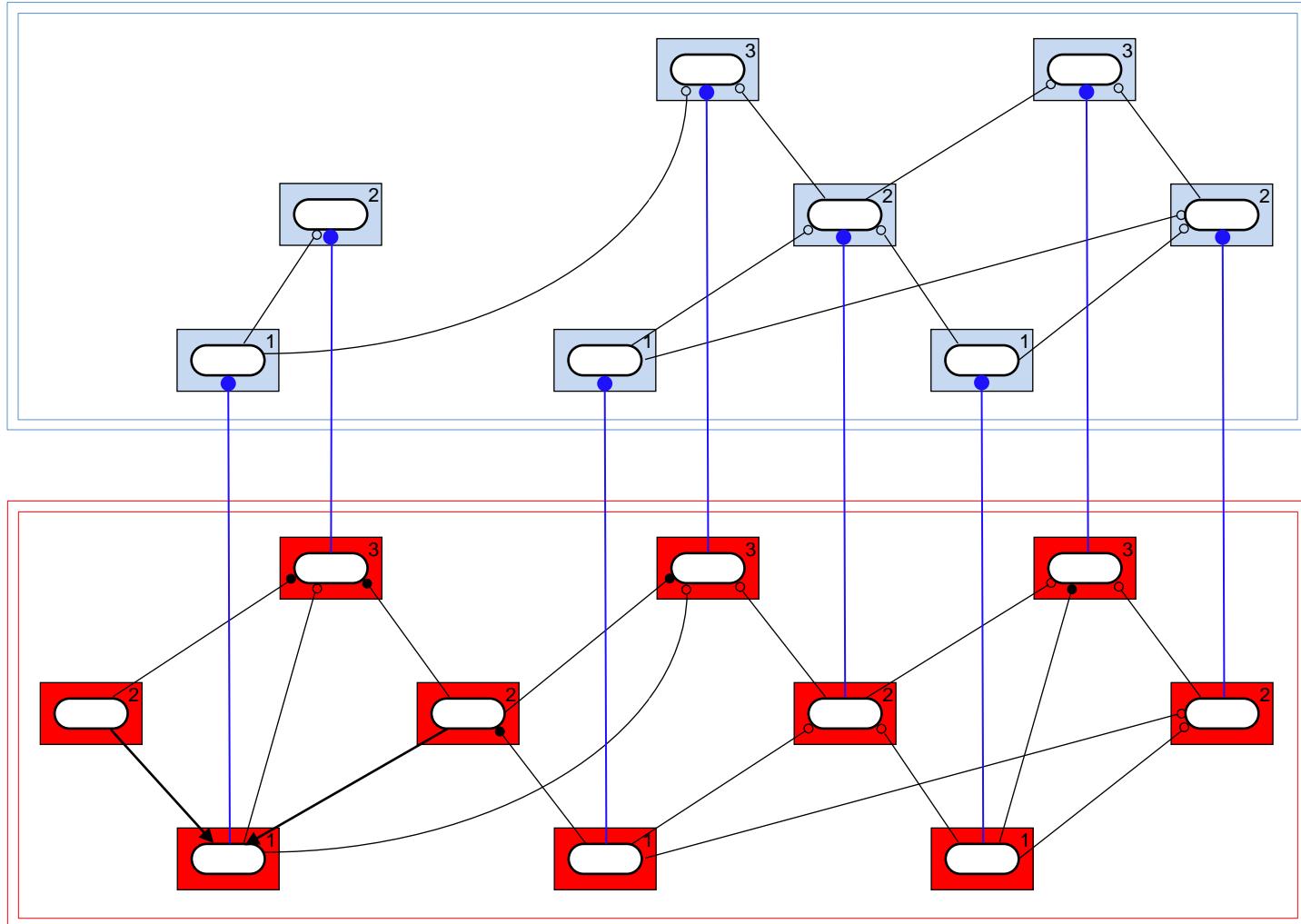
Wrapper Package



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

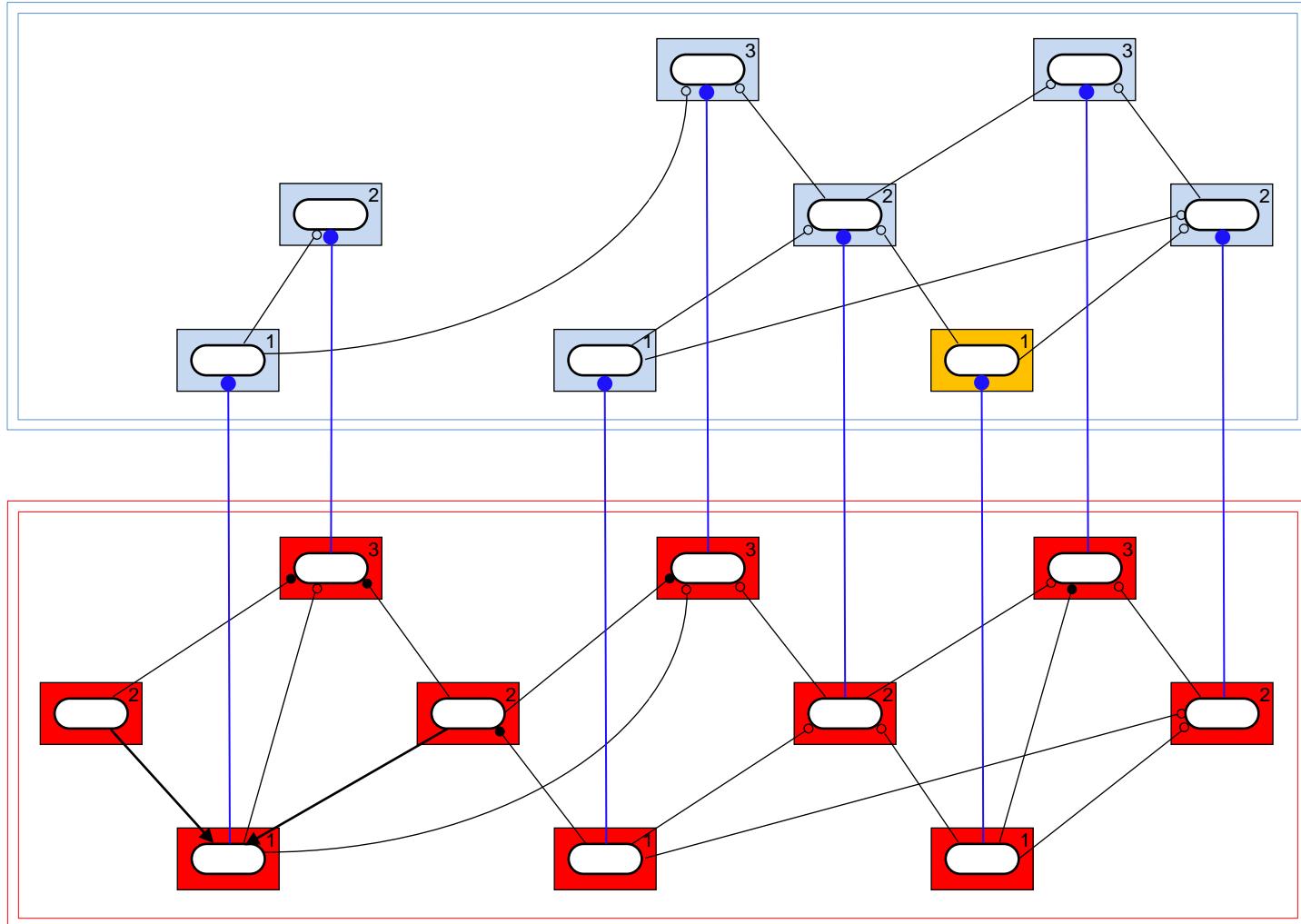
Wrapper Package



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

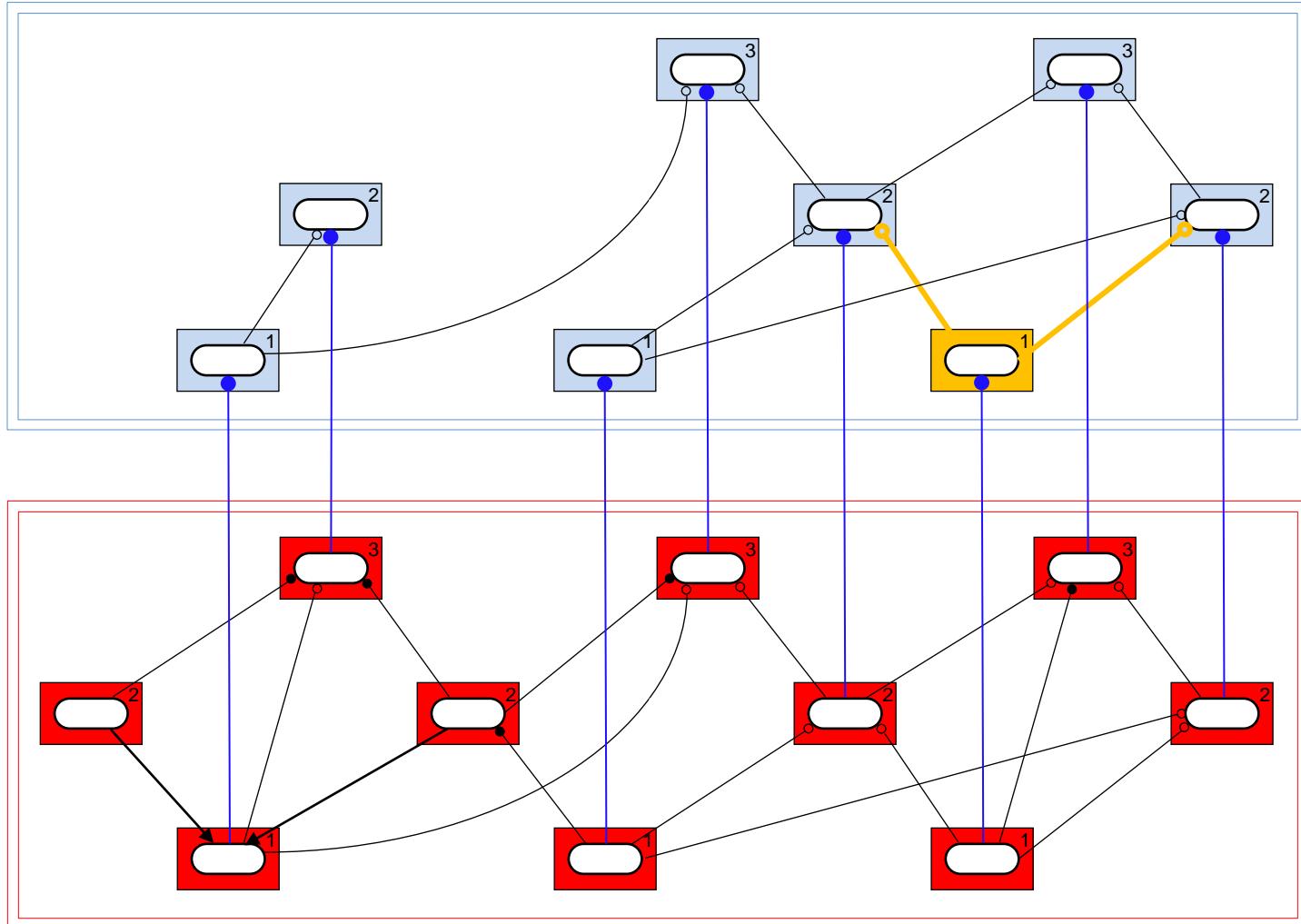
Wrapper Package



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

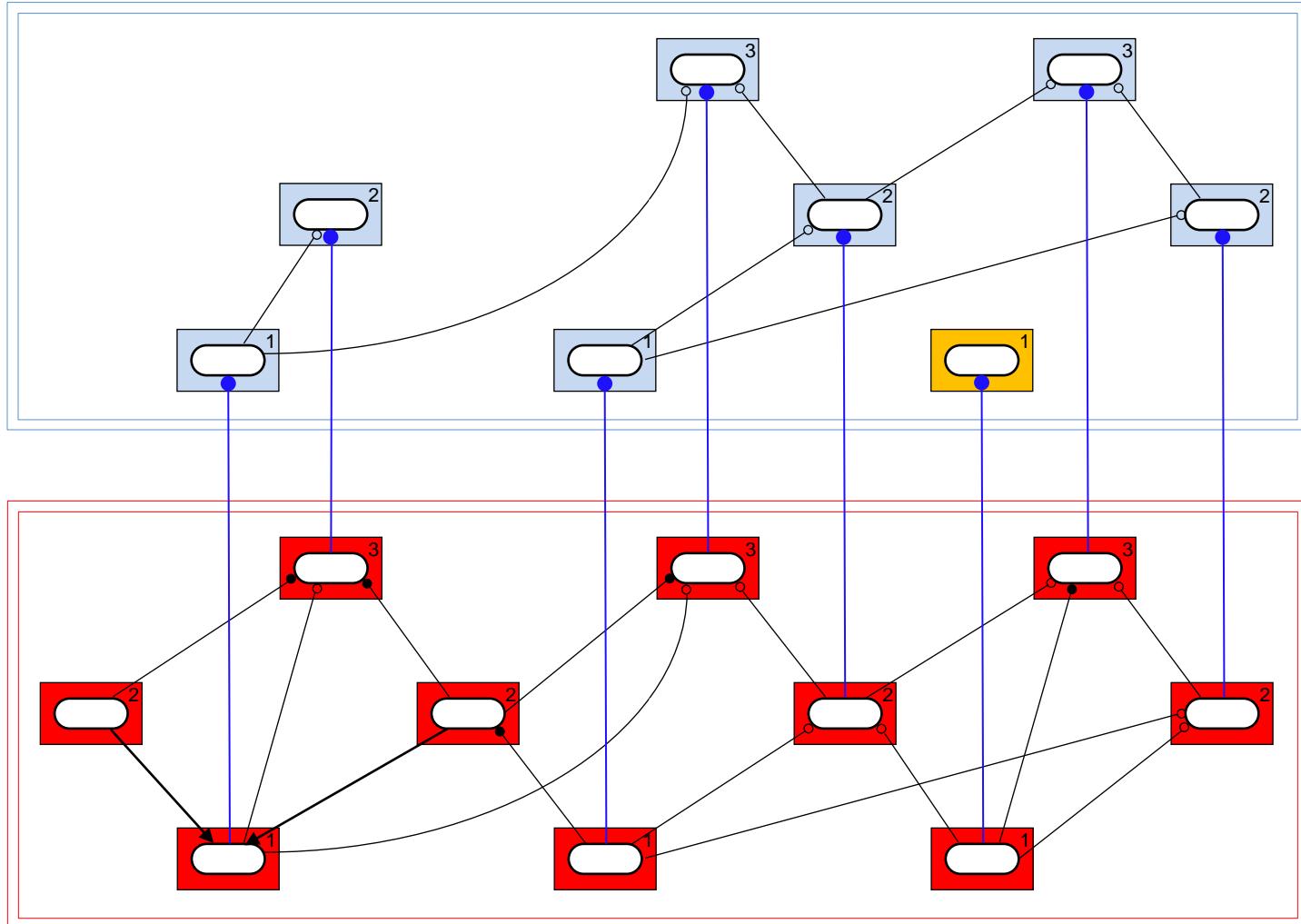
Wrapper Package



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

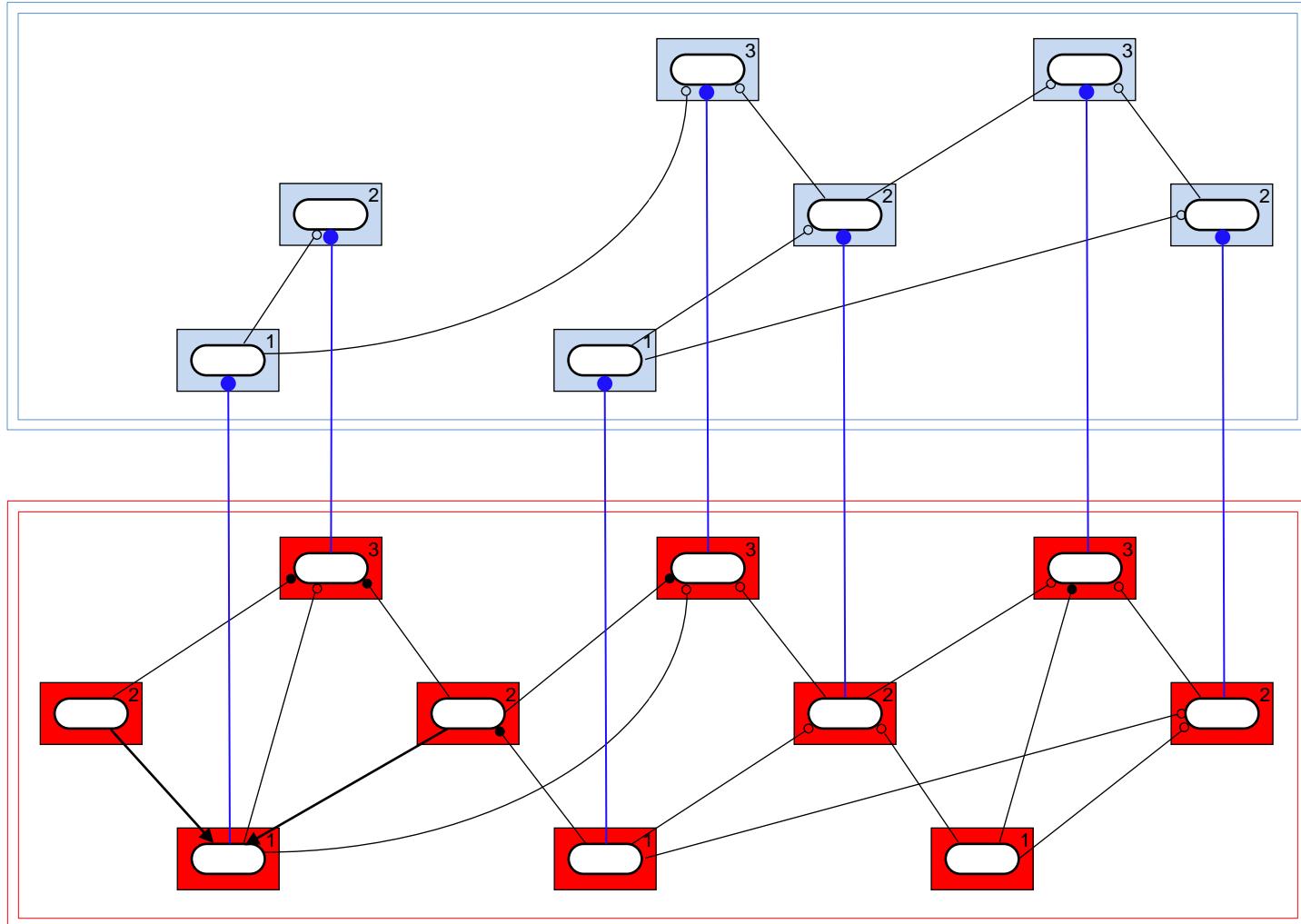
Wrapper Package



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

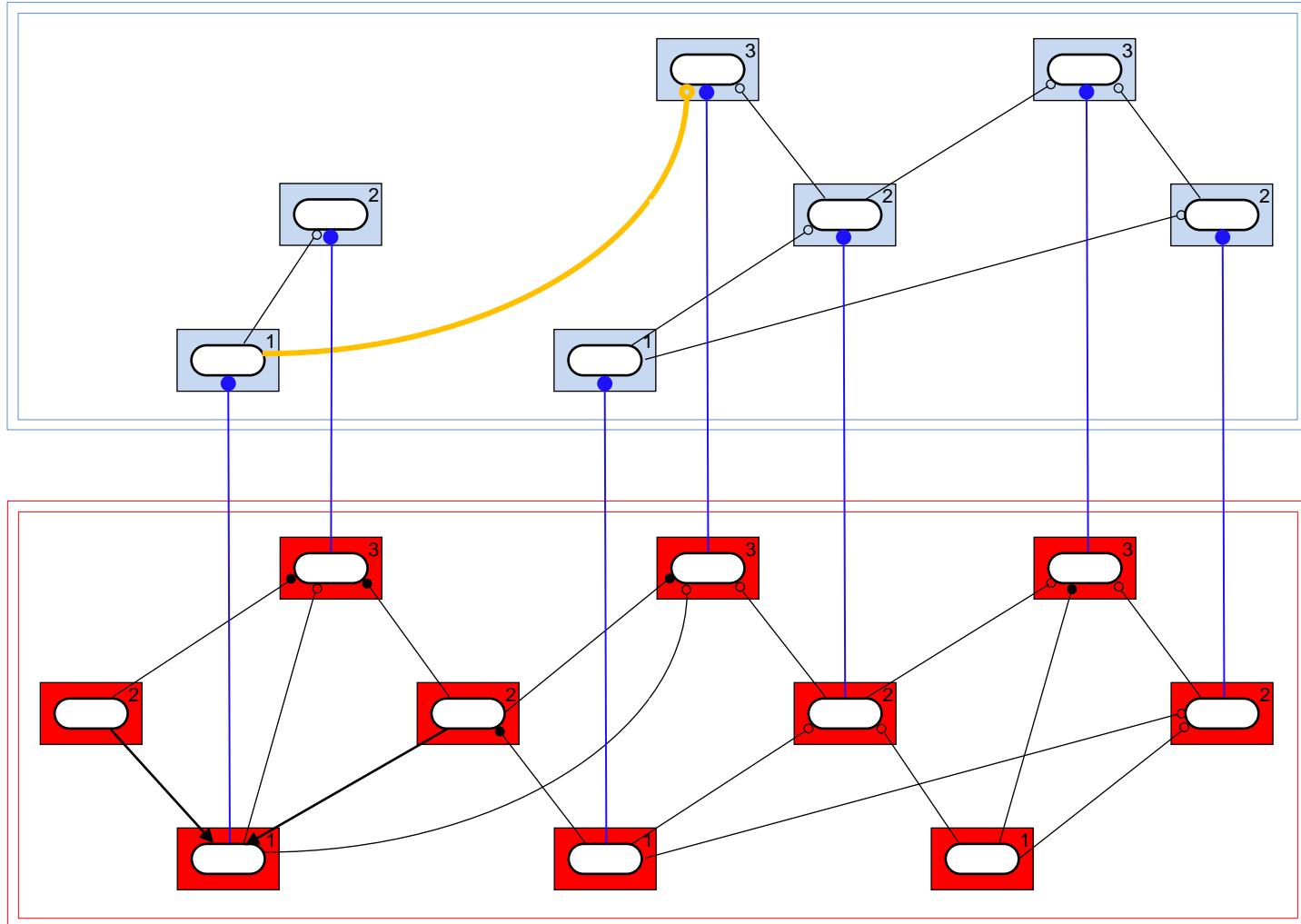
Wrapper Package



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

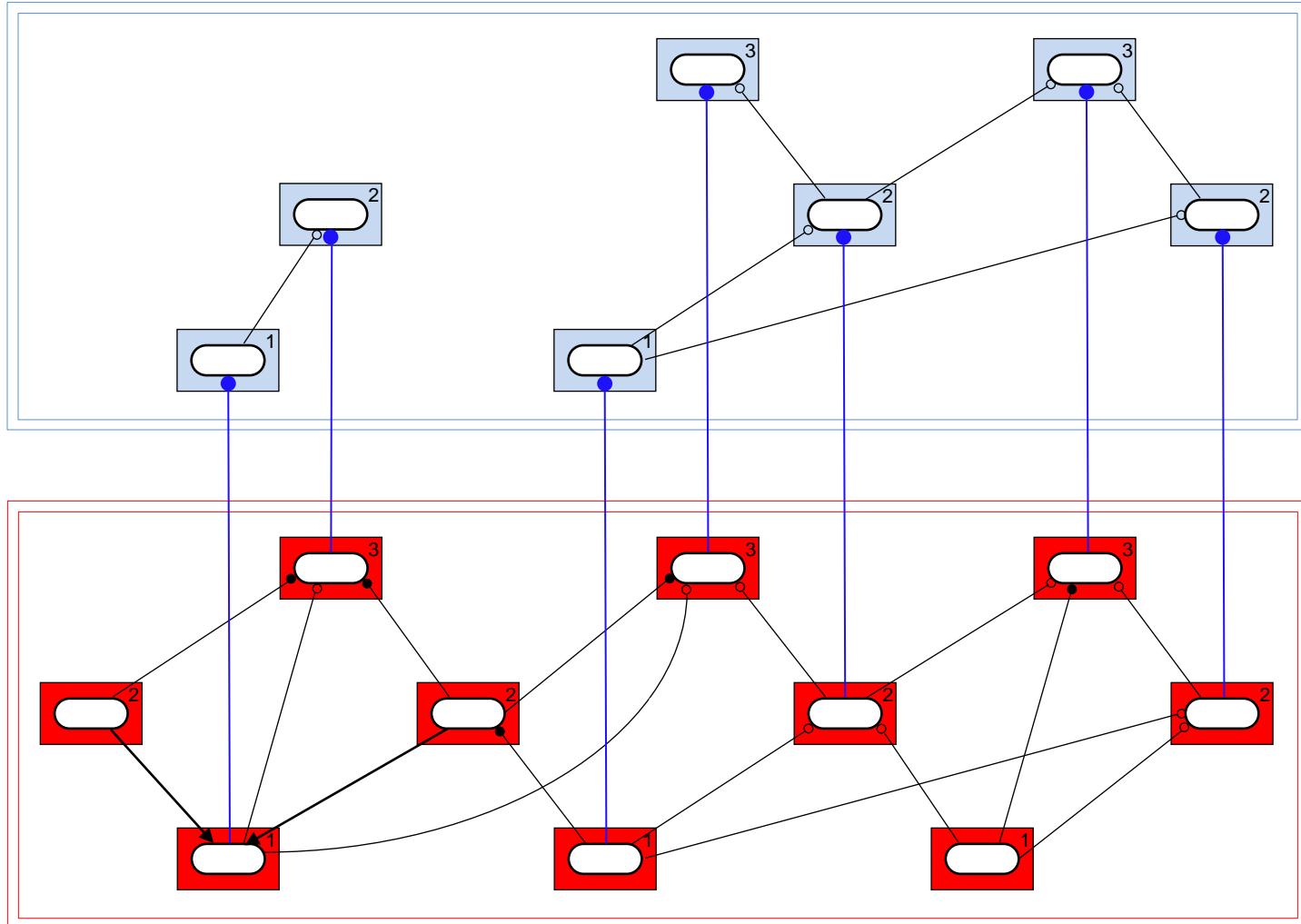
Wrapper Package



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

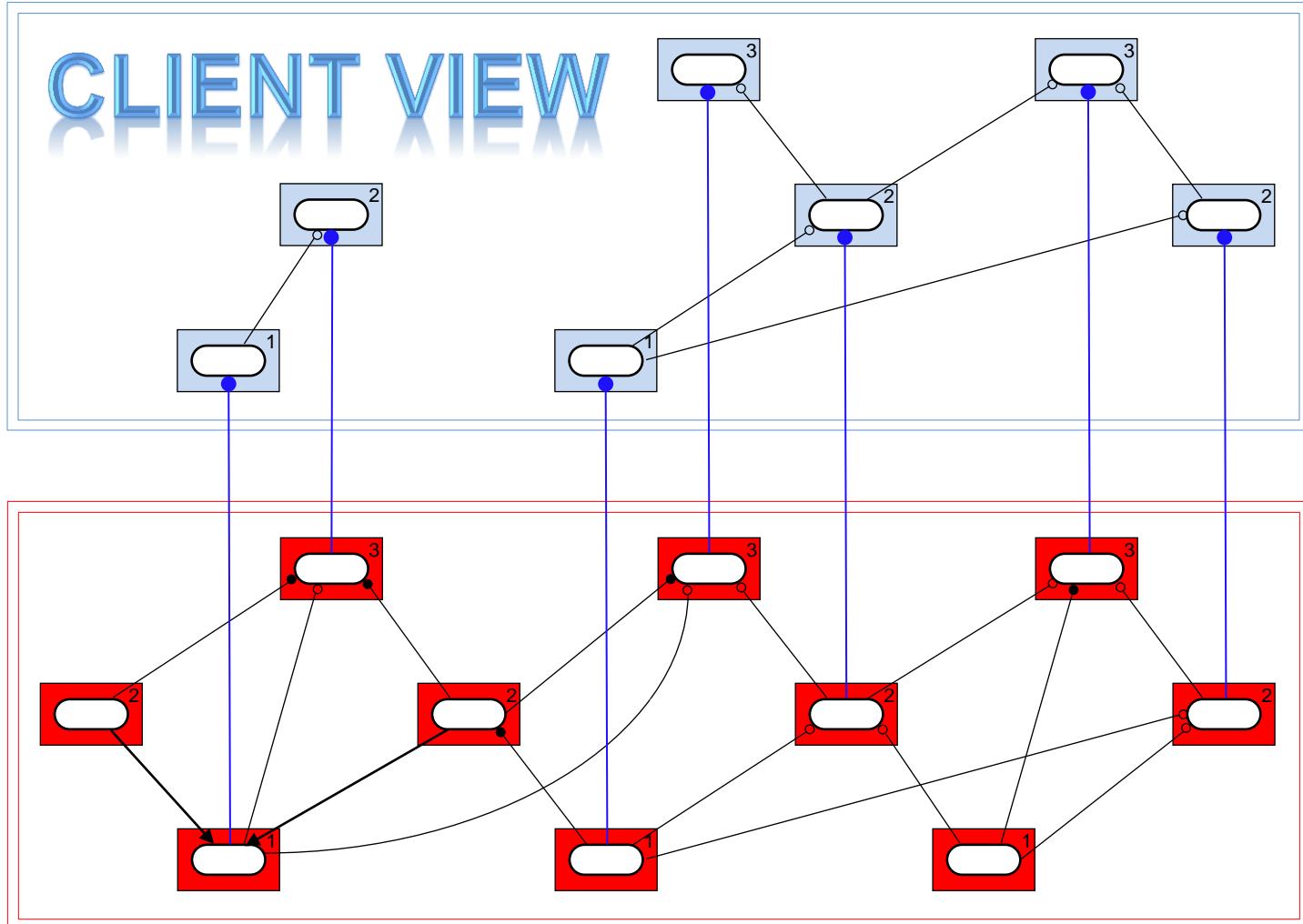
Wrapper Package



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

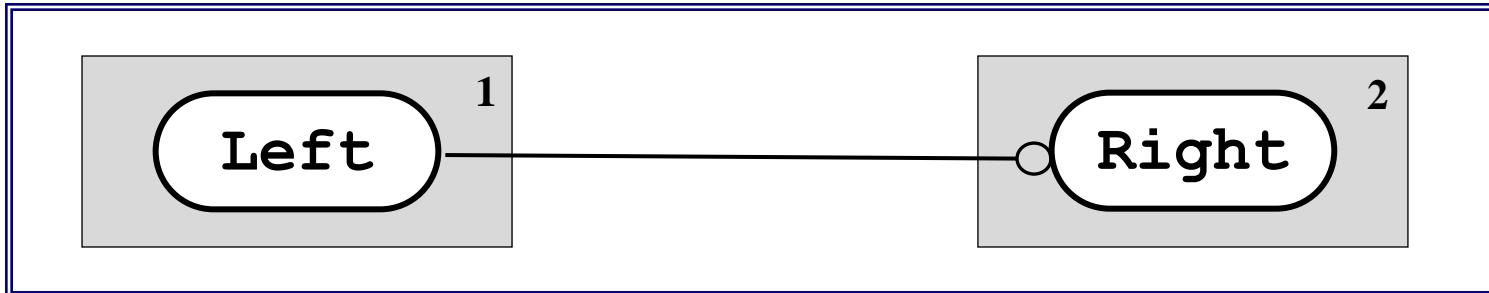
Wrapper Package



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

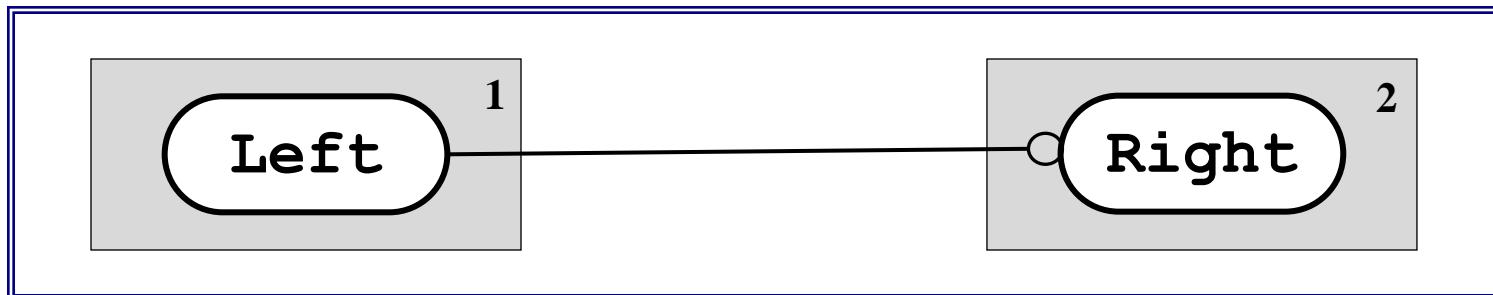
Wrapper Package



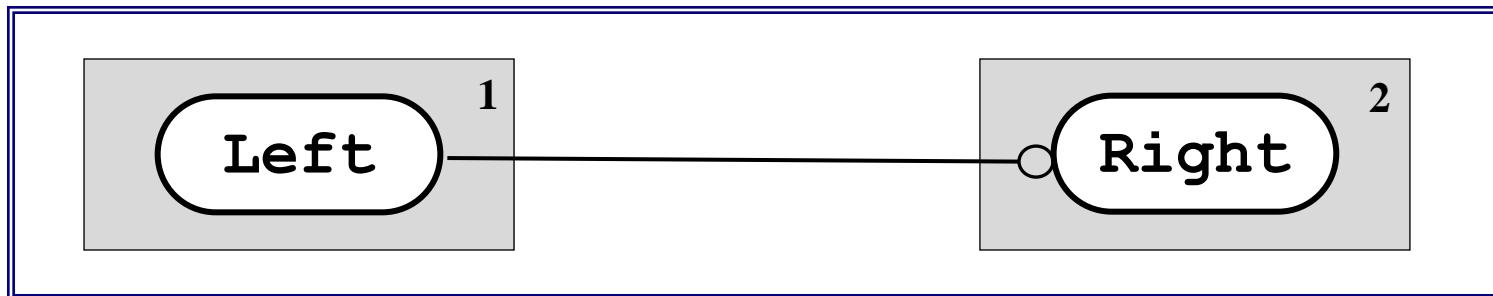
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

Wrapper Package



wrap

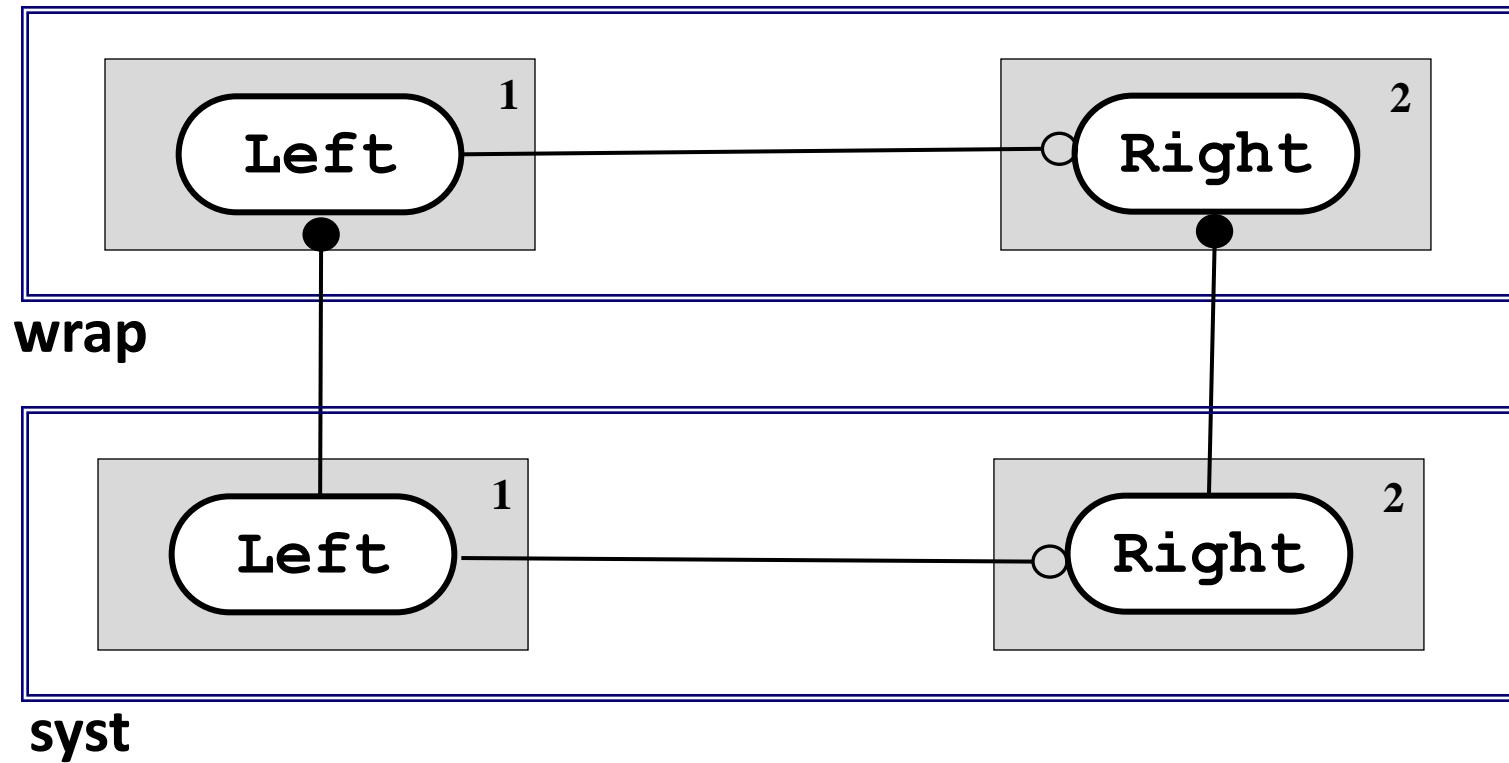


syst

2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

Wrapper Package



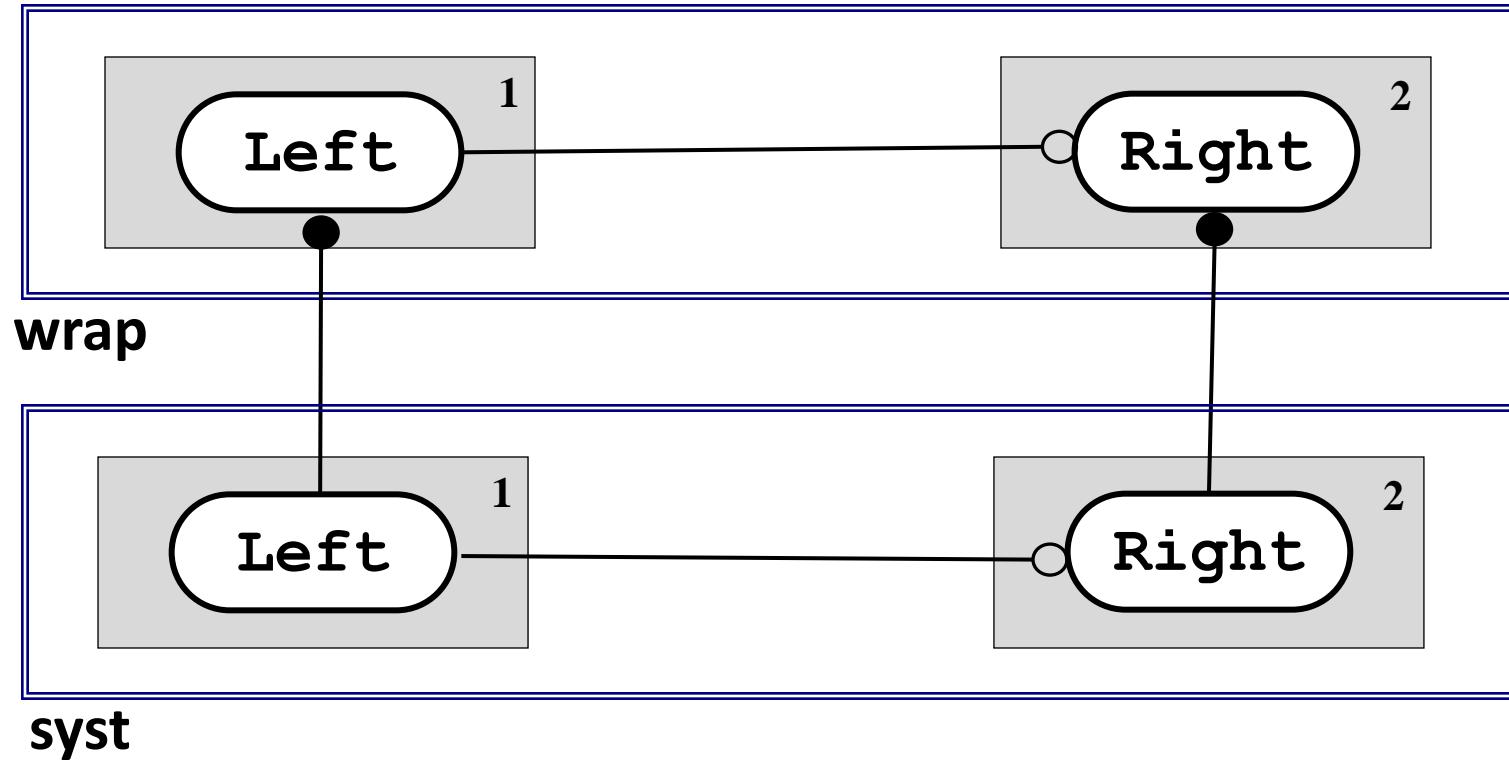
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

Wrapper Package

```
// wrap_left.h  
// ...  
class wrap_Left {  
    syst_Left d_imp;  
public: // ...
```

```
// wrap_right.h  
// ...  
class wrap_Right {  
    syst_Right d_imp;  
public: // ...
```



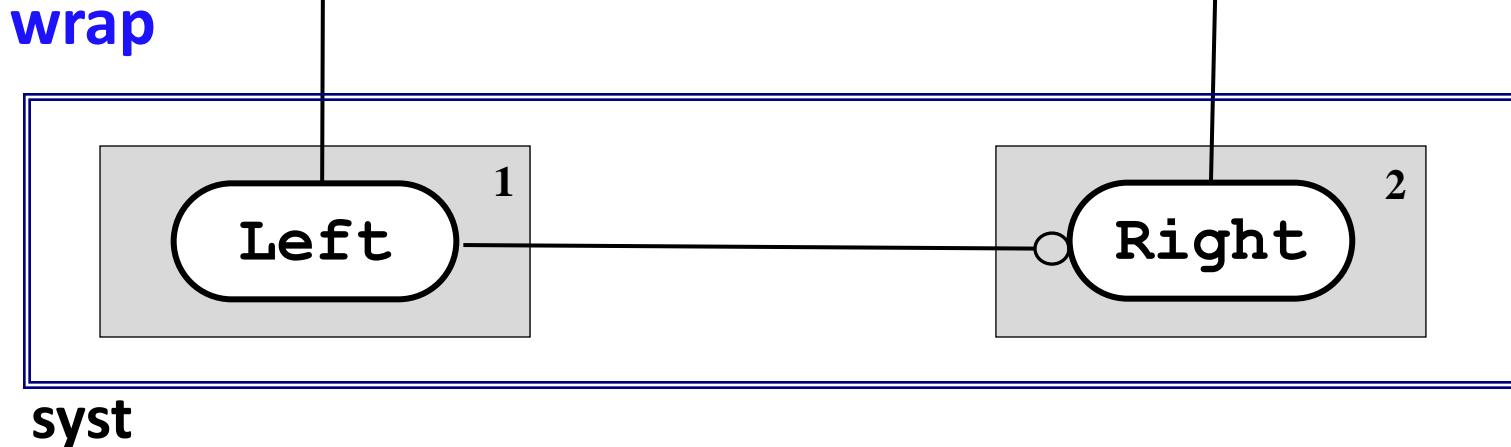
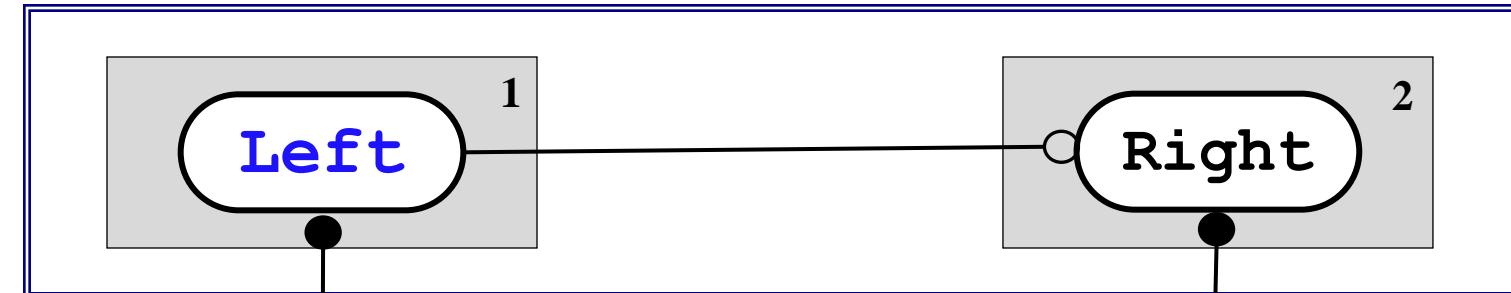
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

Wrapper Package

```
// wrap_left.h  
// ...  
class wrap_Left {  
    syst_Left d_imp;  
public: // ...
```

```
// wrap_right.h  
// ...  
class wrap_Right {  
    syst_Right d_imp;  
public: // ...
```



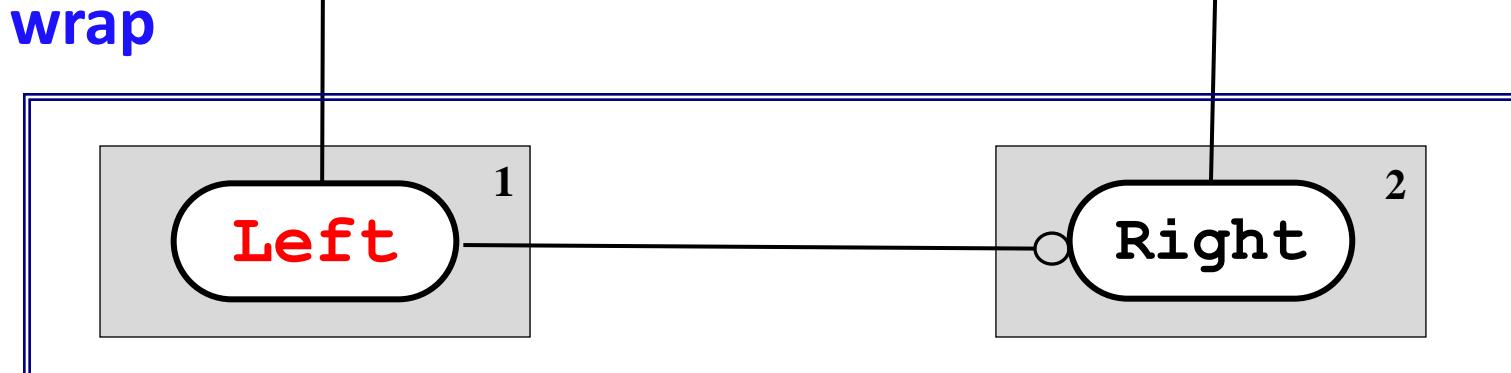
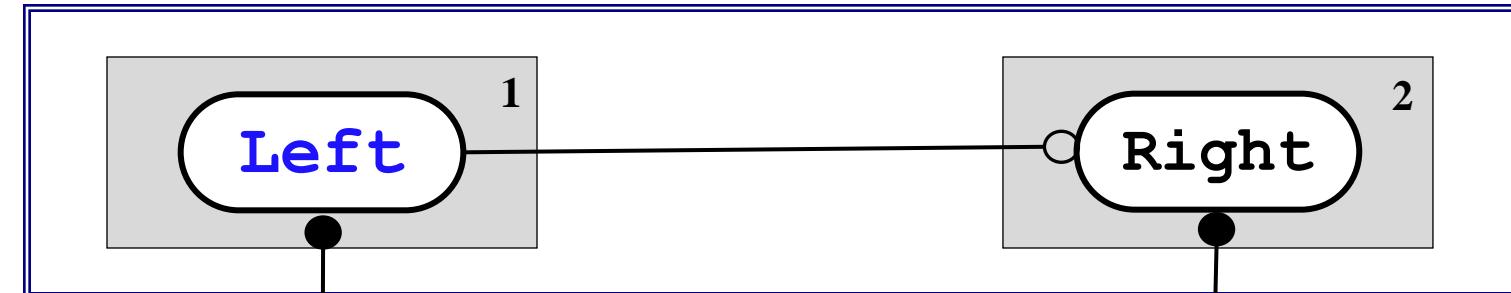
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

Wrapper Package

```
// wrap_left.h  
// ...  
class wrap_Left {  
    syst_Left d_imp;  
public: // ...
```

```
// wrap_right.h  
// ...  
class wrap_Right {  
    syst_Right d_imp;  
public: // ...
```



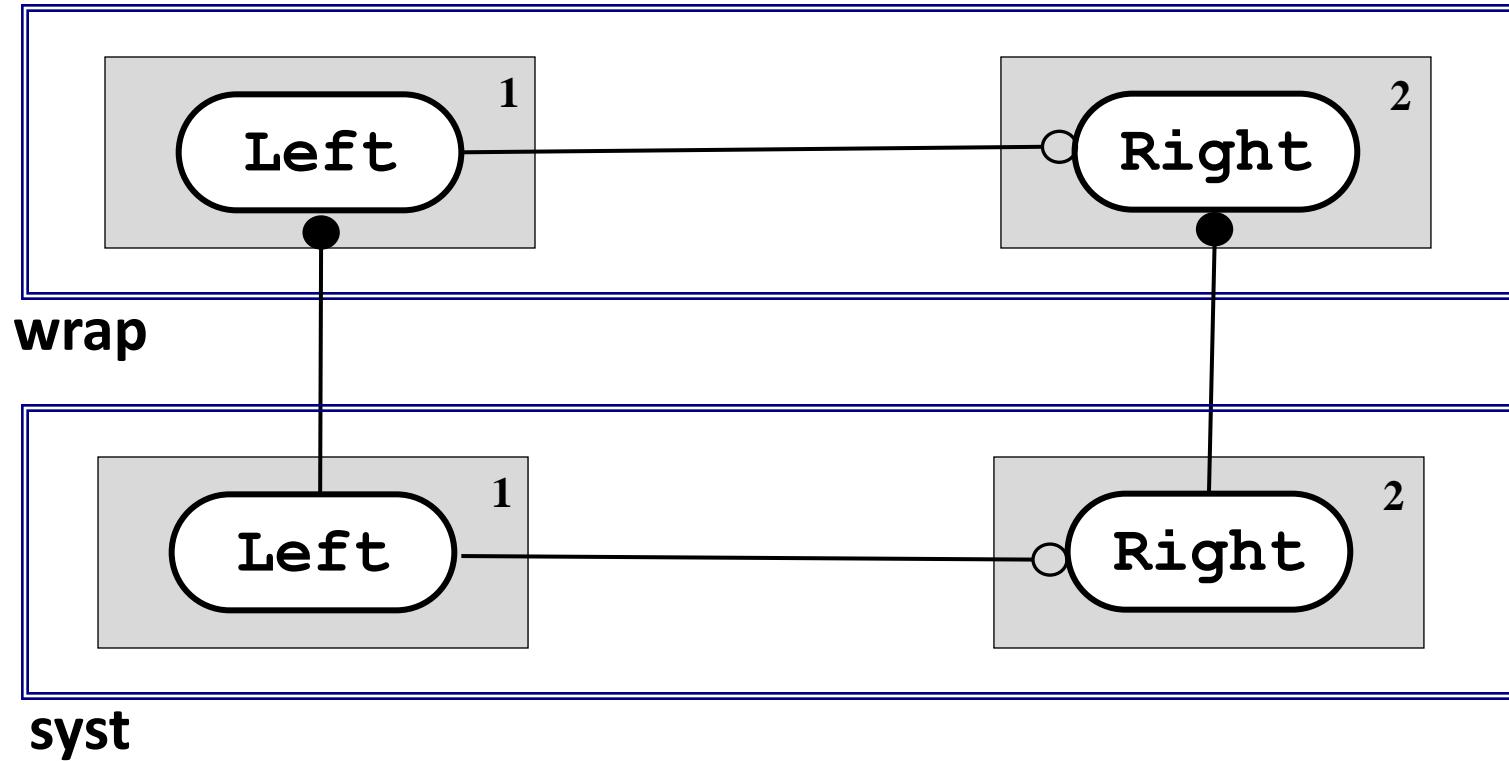
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

Wrapper Package

```
// wrap_left.h  
// ...  
class wrap_Left {  
    syst_Left d_imp;  
public: // ...
```

```
// wrap_right.h  
// ...  
class wrap_Right {  
    syst_Right d_imp;  
public: // ...
```



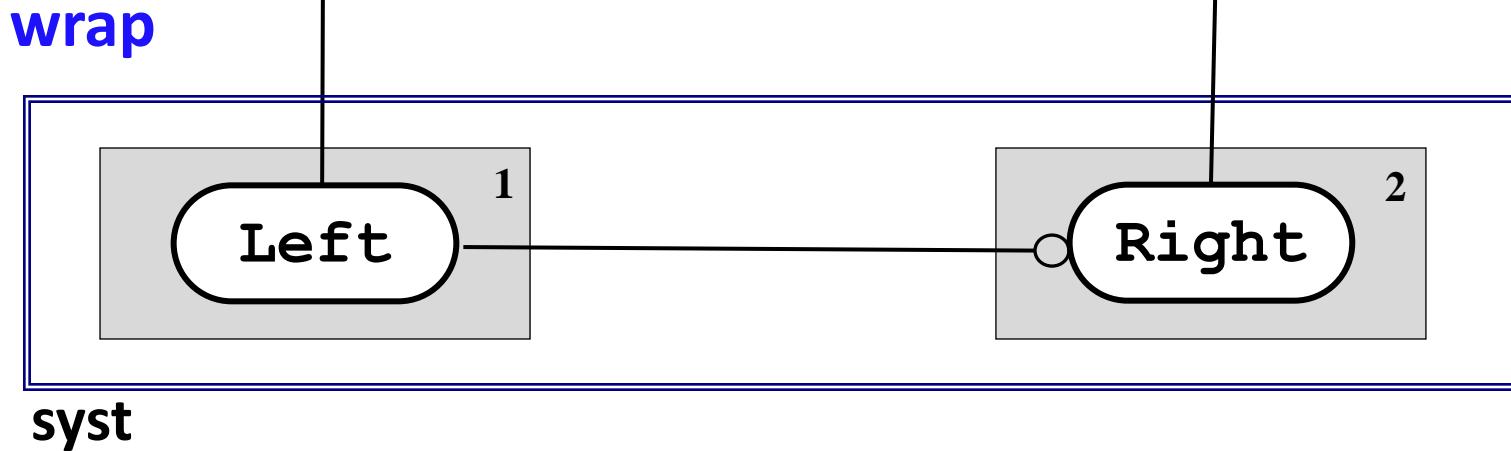
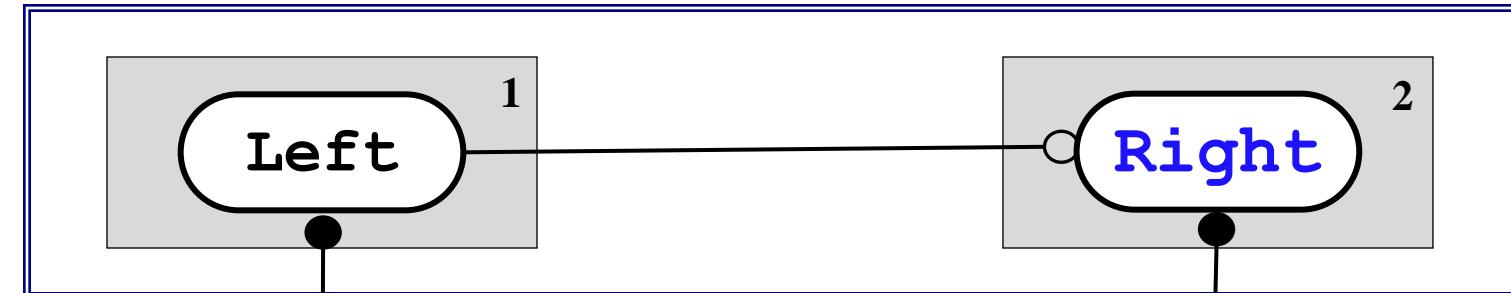
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

Wrapper Package

```
// wrap_left.h  
// ...  
class wrap_Left {  
    syst_Left d_imp;  
public: // ...
```

```
// wrap_right.h  
// ...  
class wrap_Right {  
    syst_Right d_imp;  
public: // ...
```



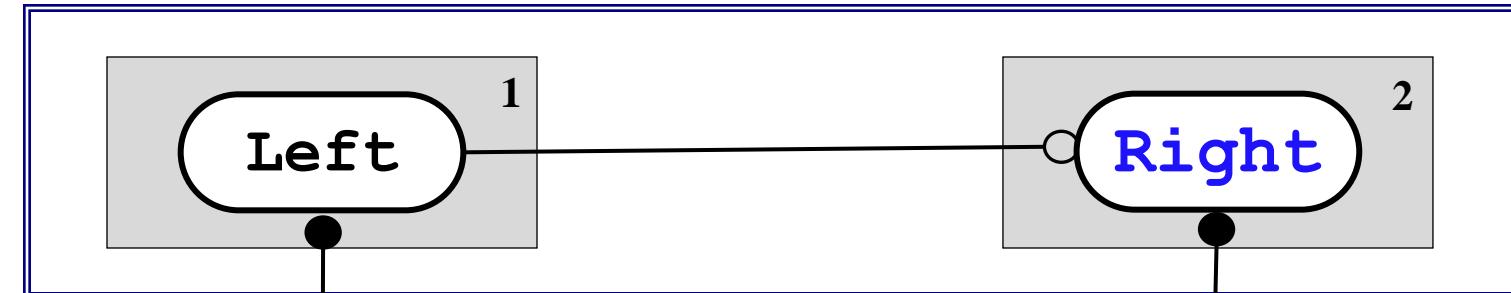
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

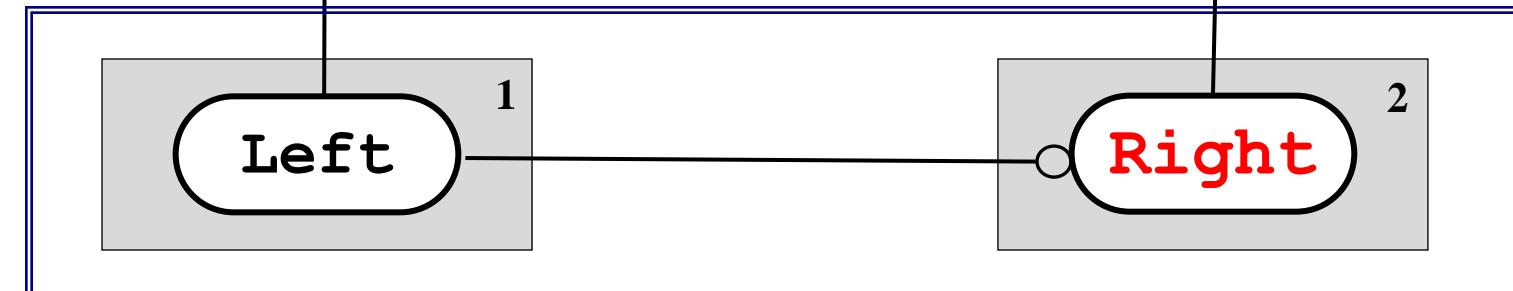
Wrapper Package

```
// wrap_left.h  
// ...  
class wrap_Left {  
    syst_Left d_imp;  
public: // ...
```

```
// wrap_right.h  
// ...  
class wrap_Right {  
    syst_Right d_imp;  
public: // ...
```



wrap



syst

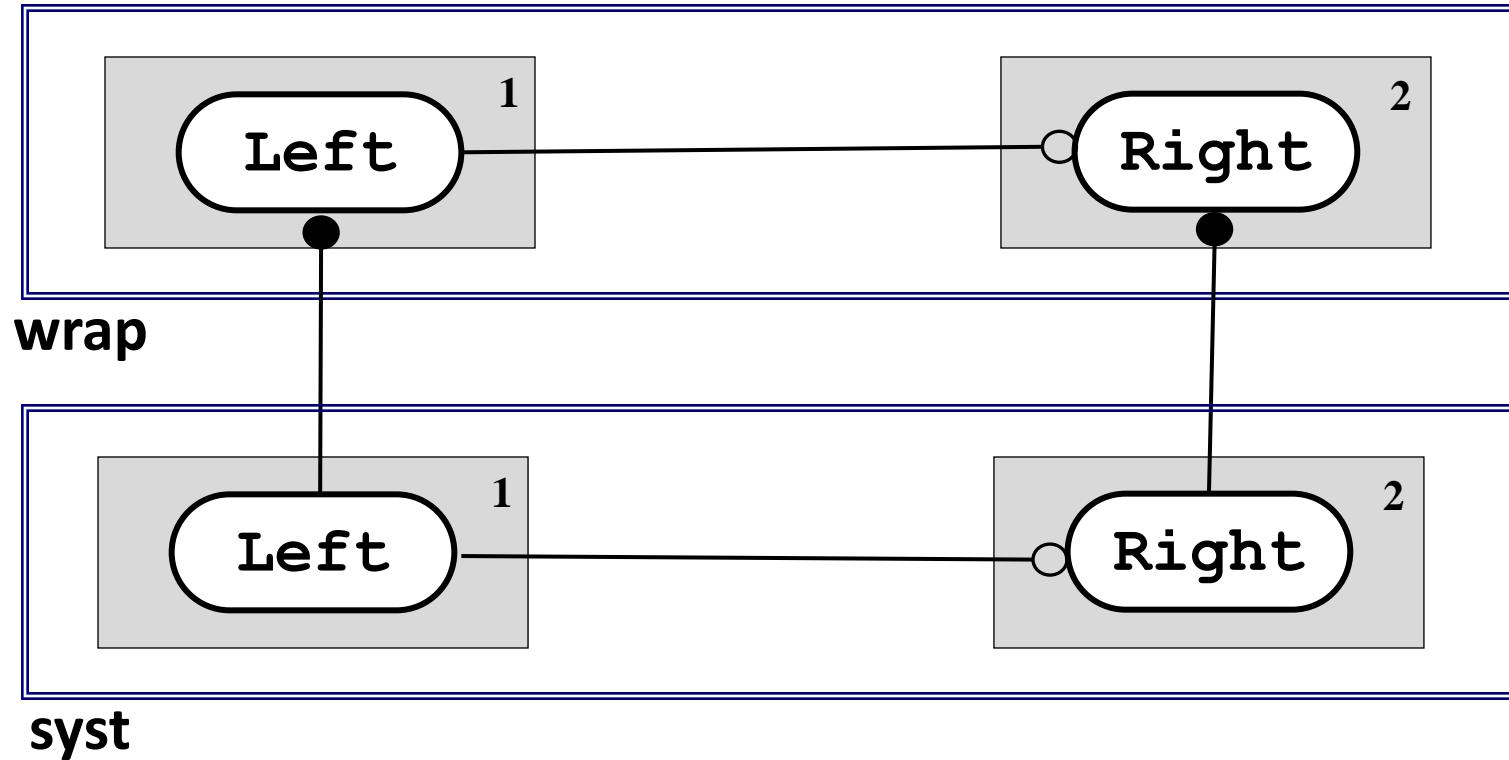
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

Wrapper Package

```
// wrap_left.h  
// ...  
class wrap_Left {  
    syst_Left d_imp;  
public: // ...
```

```
// wrap_right.h  
// ...  
class wrap_Right {  
    syst_Right d_imp;  
public: // ...
```



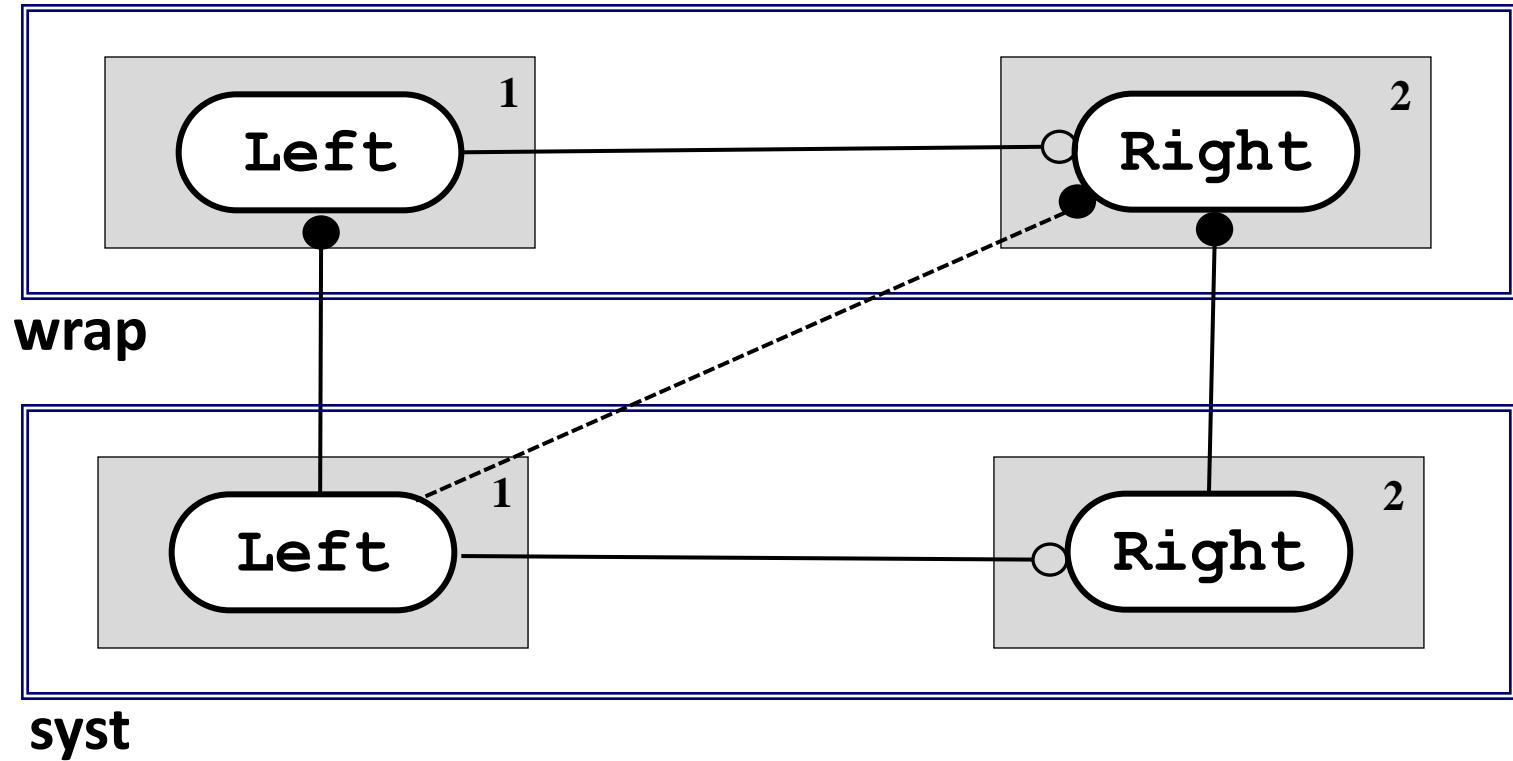
2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

Wrapper Package

```
// wrap_left.h  
// ...  
class wrap_Left {  
    syst_Left d_imp;  
public: // ...
```

```
// wrap_right.h  
// ...  
class wrap_Right {  
    syst_Right d_imp;  
public: // ...
```

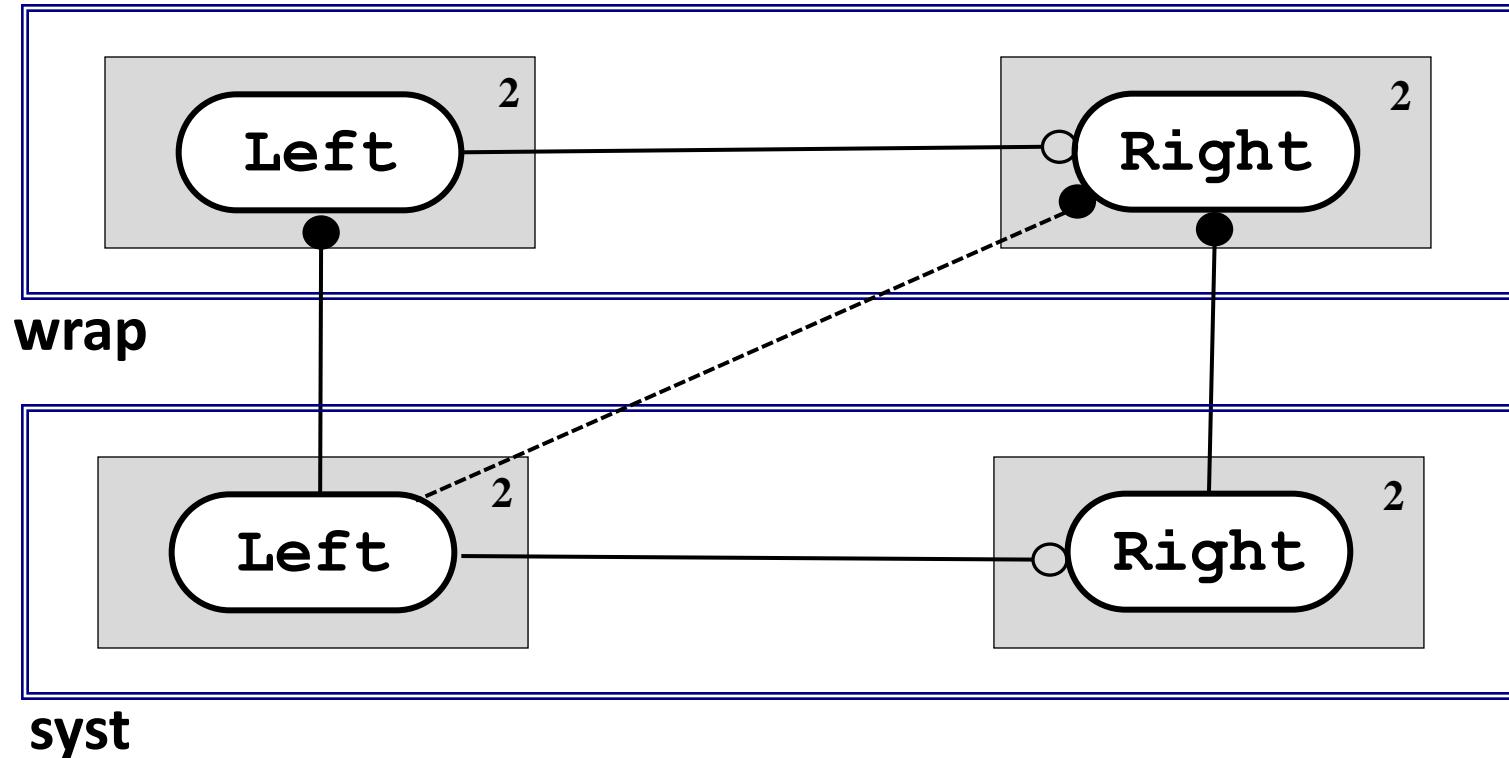


2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

Wrapper Package

```
// wrap_right.cpp
// ...
void wrap_Right::someFunction(const wrap_Left& v) {
    const syst_Left& vImp = *reinterpret_cast<syst_Left *>(&v);
    d_imp.someFunction(vImp);
```

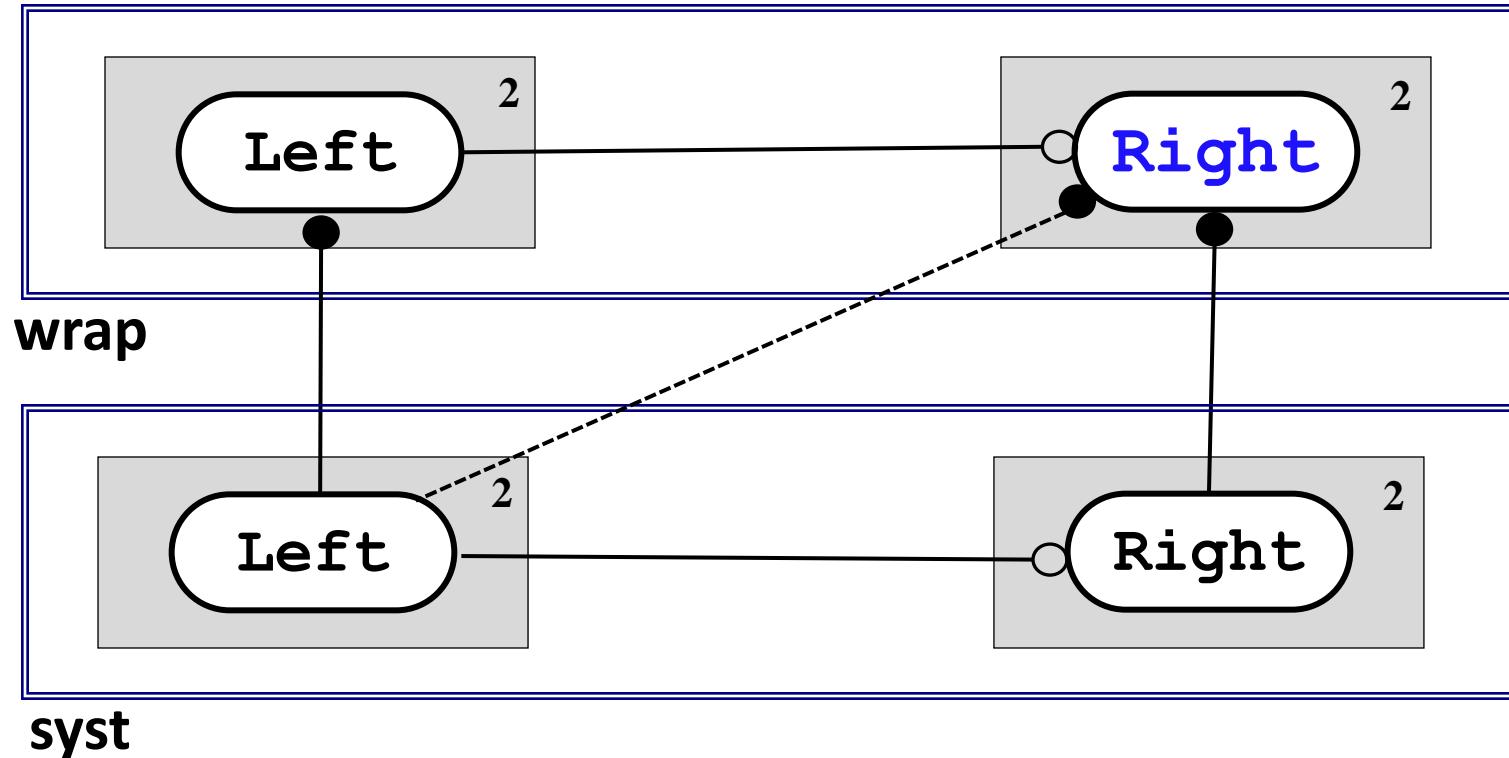


2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

Wrapper Package

```
// wrap_right.cpp
// ...
void wrap_Right::someFunction(const wrap_Left& v) {
    const syst_Left& vImp = *reinterpret_cast<syst_Left *>(&v);
    d_imp.someFunction(vImp);
```

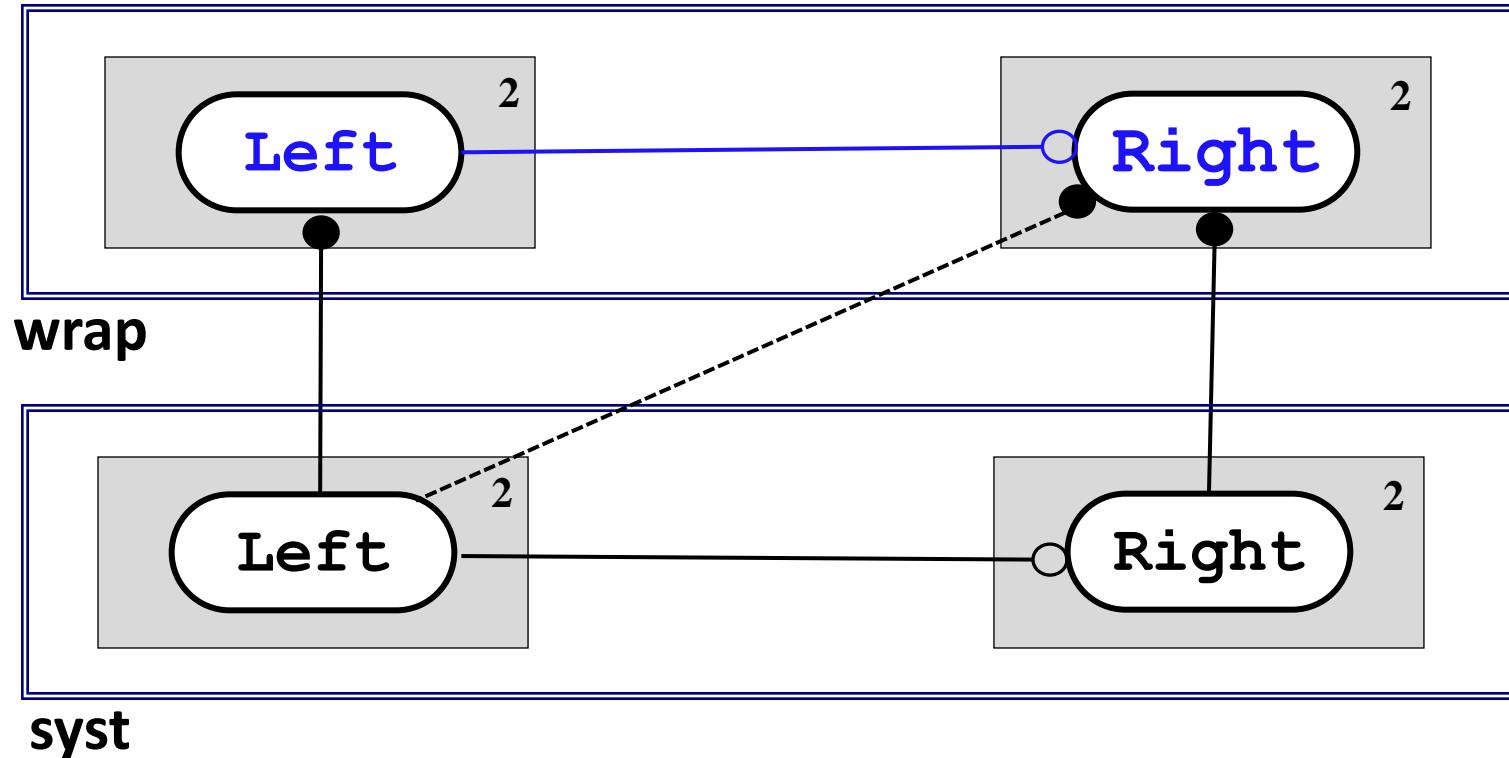


2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

Wrapper Package

```
// wrap_right.cpp
// ...
void wrap_Right::someFunction(const wrap_Left& v) {
    const syst_Left& vImp = *reinterpret_cast<syst_Left *>(&v);
    d_imp.someFunction(vImp);
```

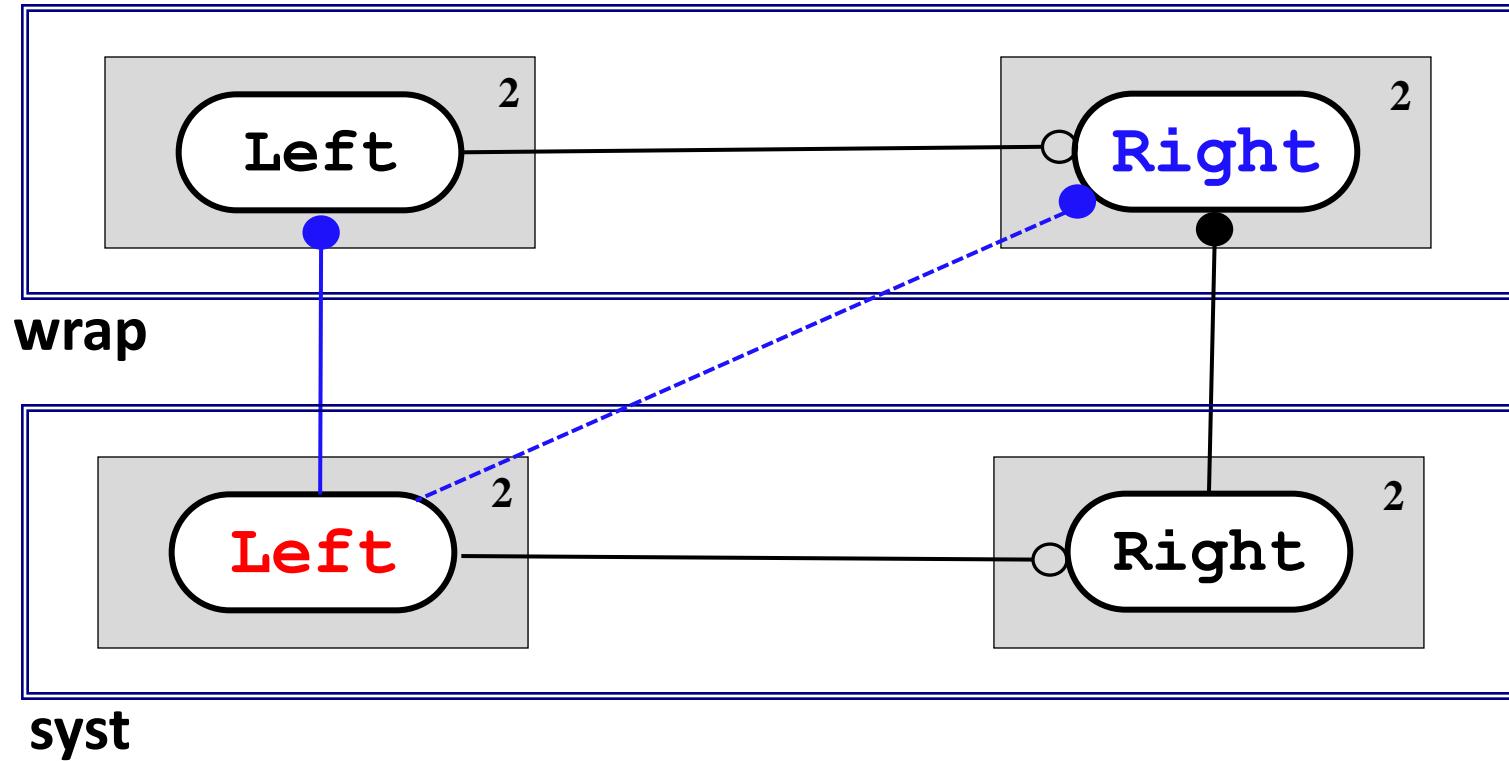


2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

Wrapper Package

```
// wrap_right.cpp
// ...
void wrap_Right::someFunction(const wrap_Left& v) {
    const syst_Left& vImp = *reinterpret_cast<syst_Left *>(&v);
    d_imp.someFunction(vImp);
```

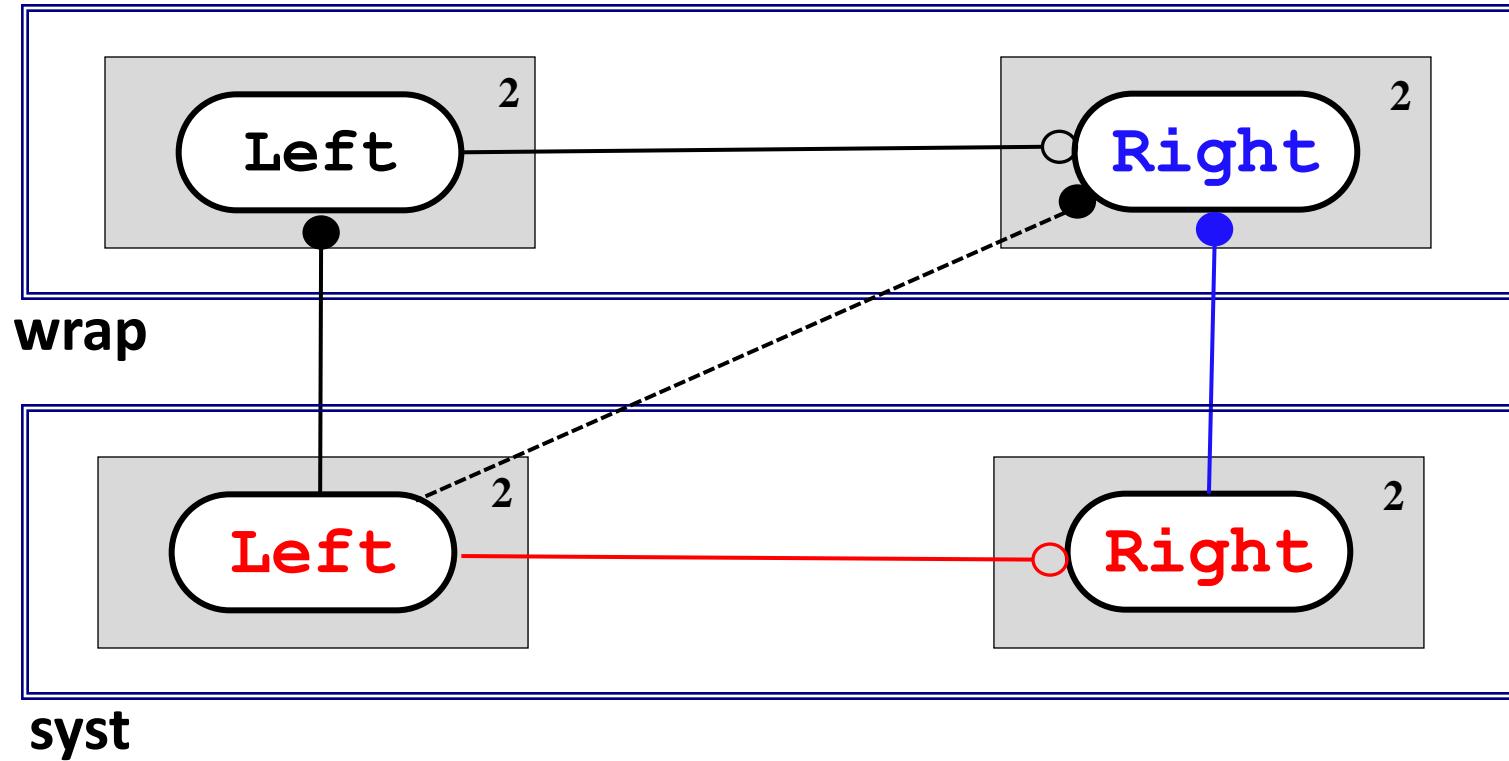


2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

Wrapper Package

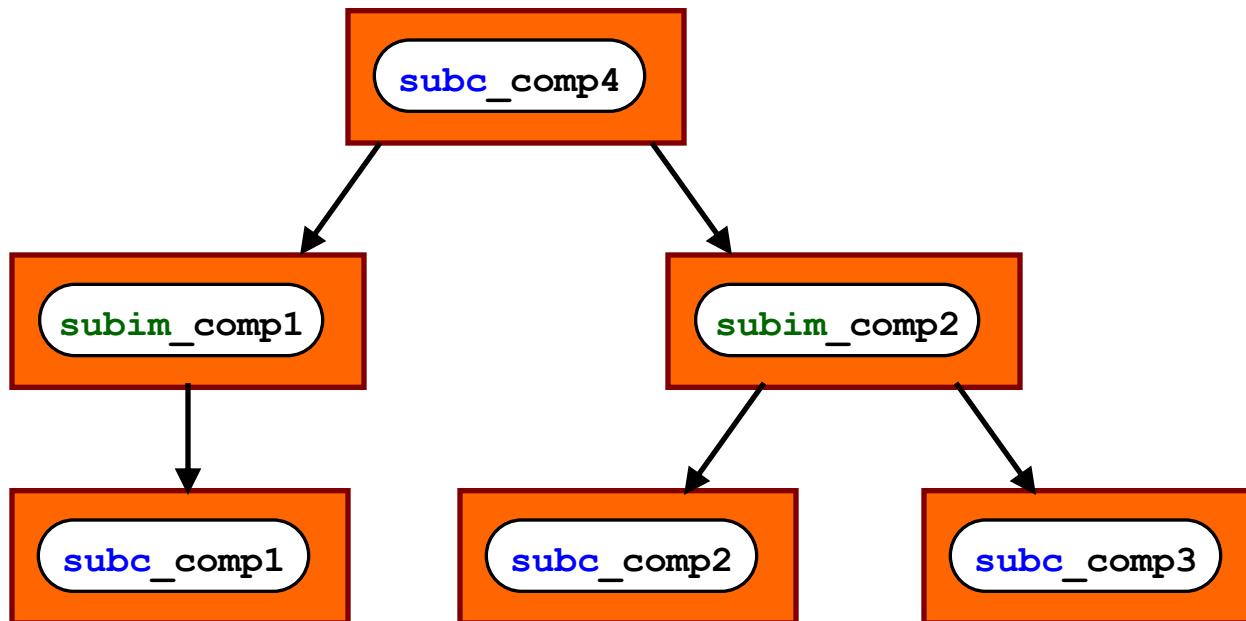
```
// wrap_right.cpp
// ...
void wrap_Right::someFunction(const wrap_Left& v) {
    const syst_Left& vImp = *reinterpret_cast<syst_Left *>(&v);
    d_imp.someFunction(vImp);
```



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

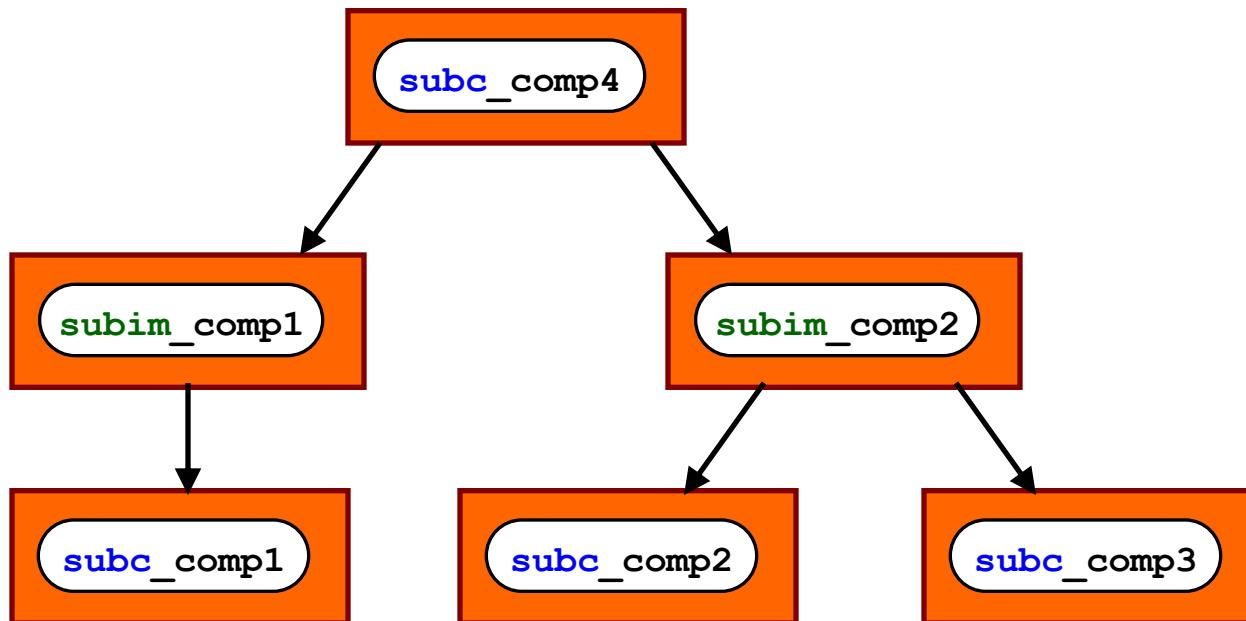
Package naming is more than just a convention:



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

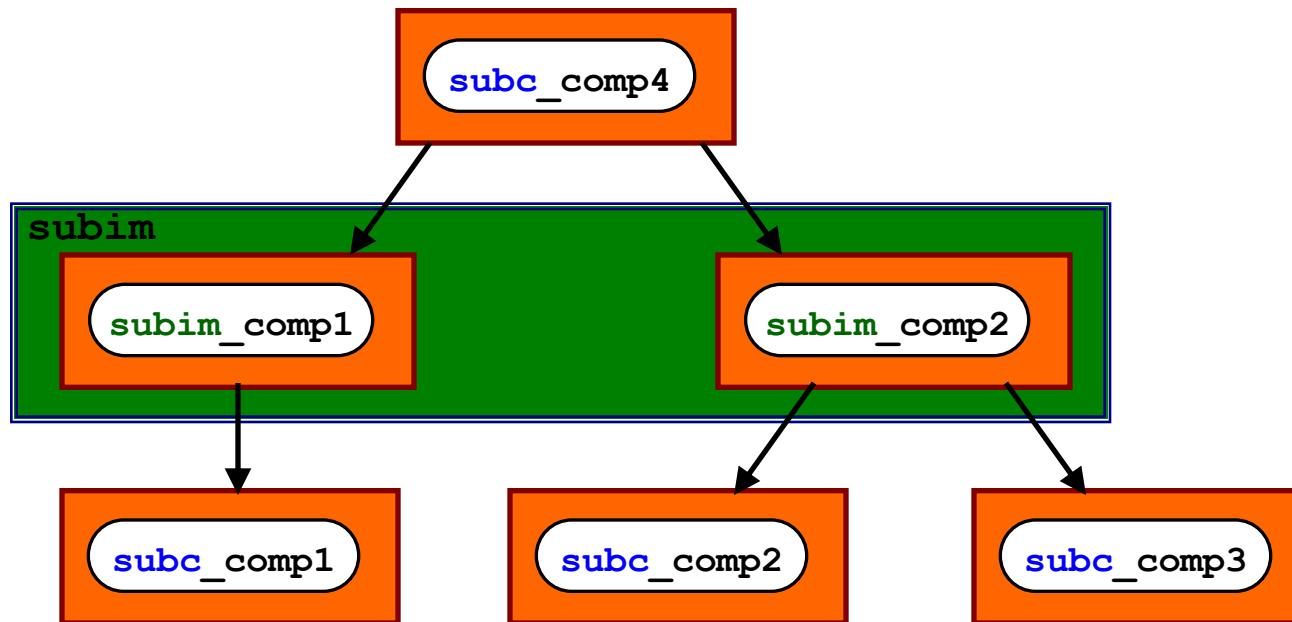
Package naming is more than just a convention:



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

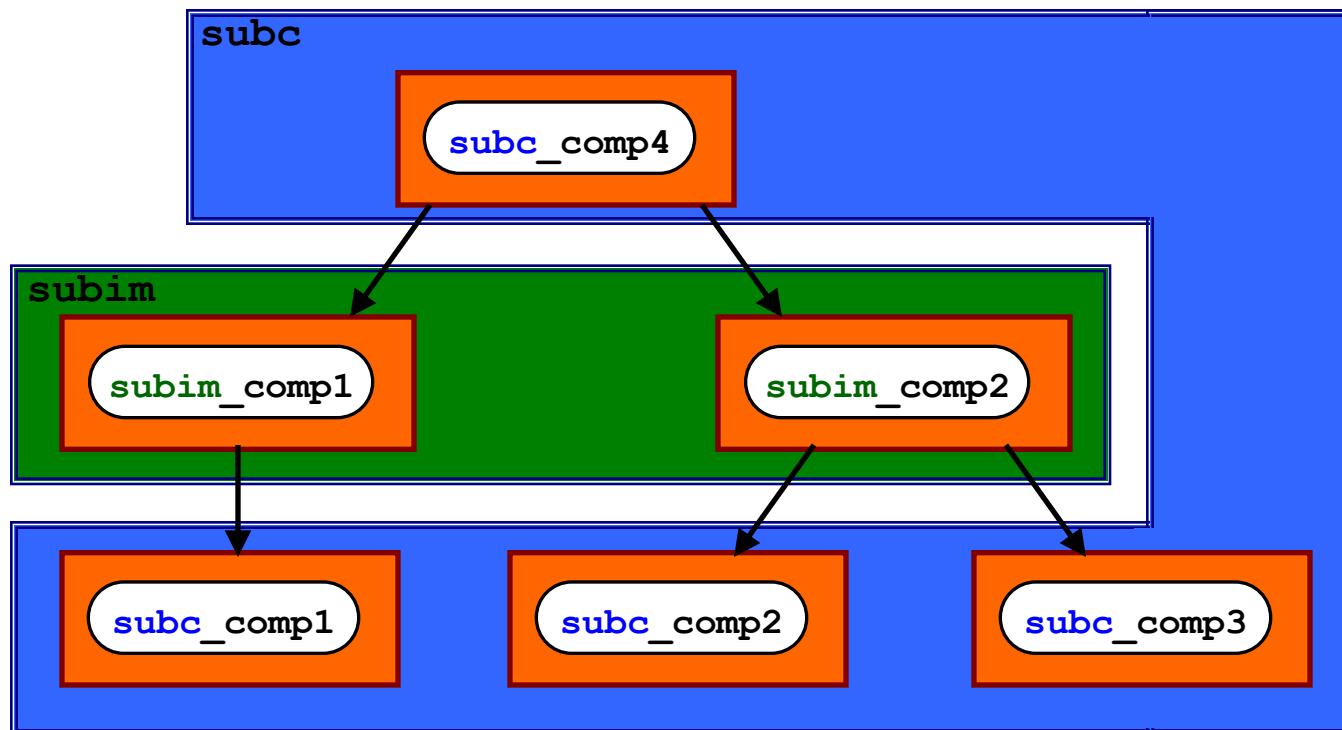
Package naming is more than just a convention:



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

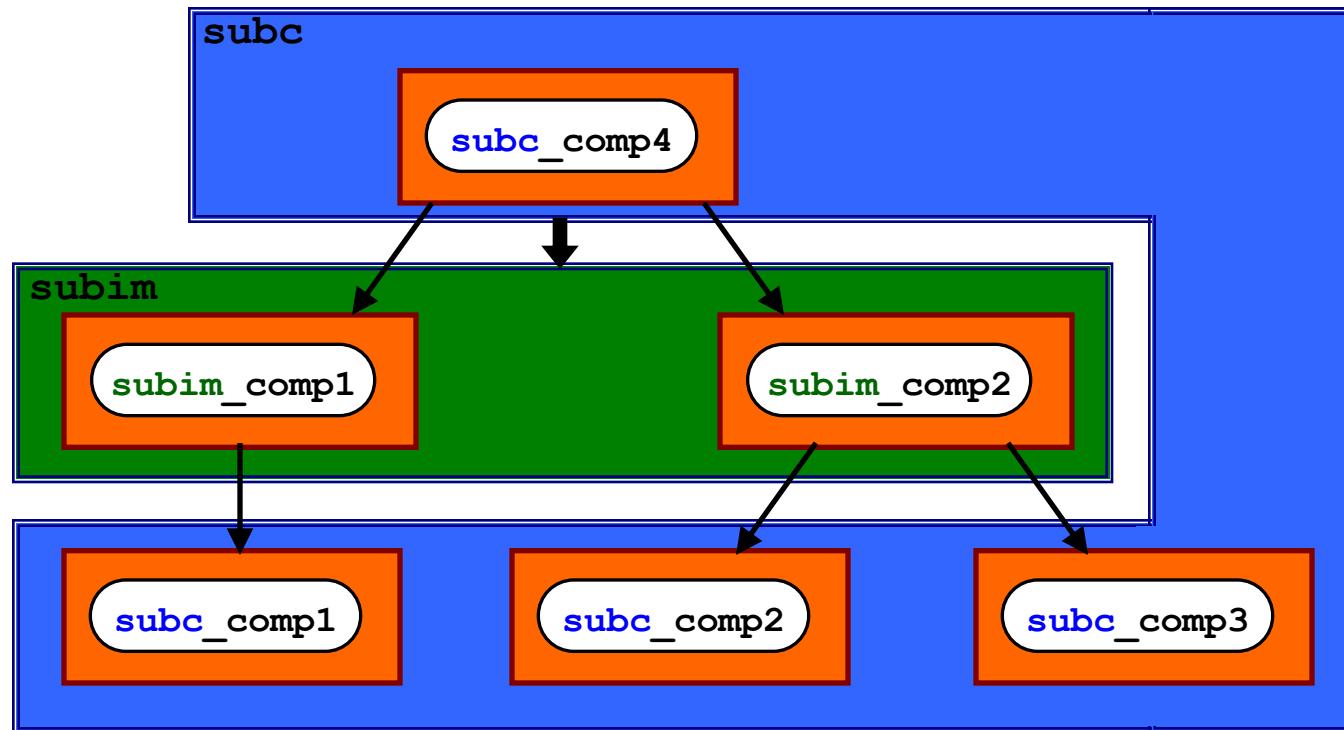
Package naming is more than just a convention:



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

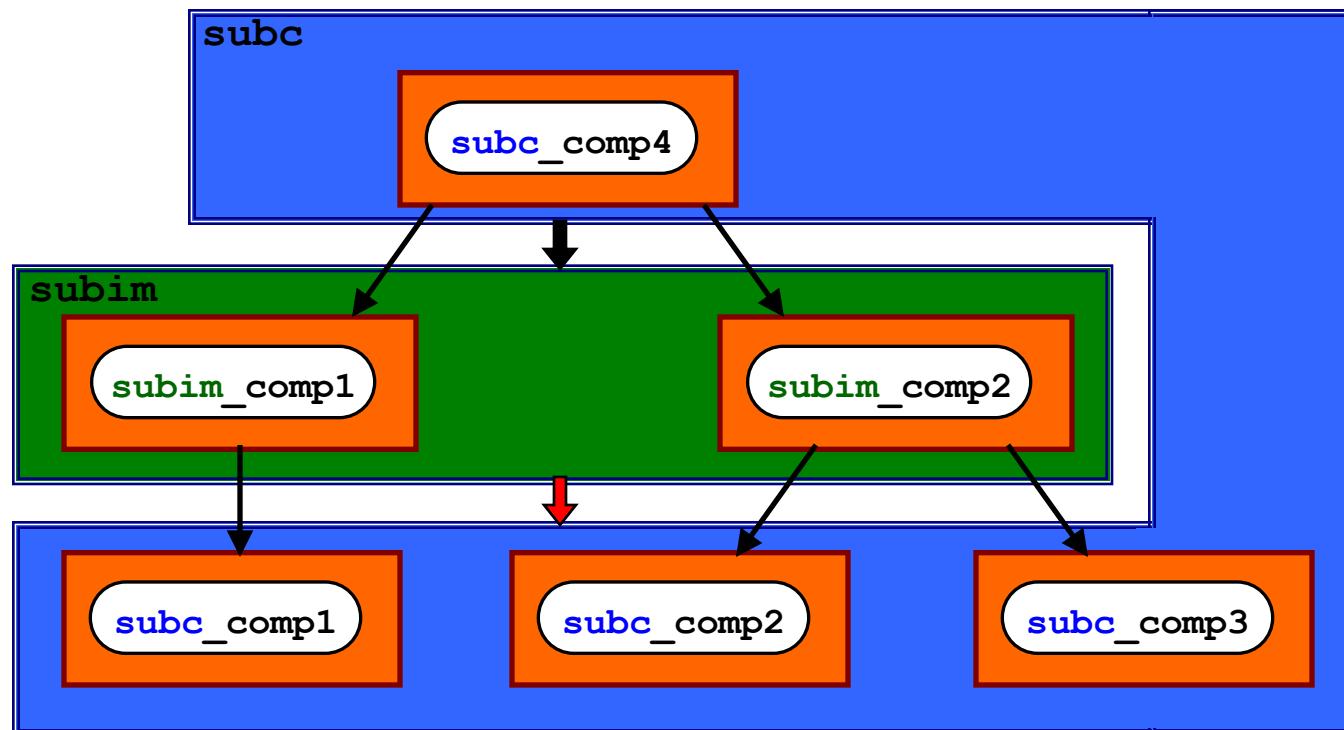
Package naming is more than just a convention:



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

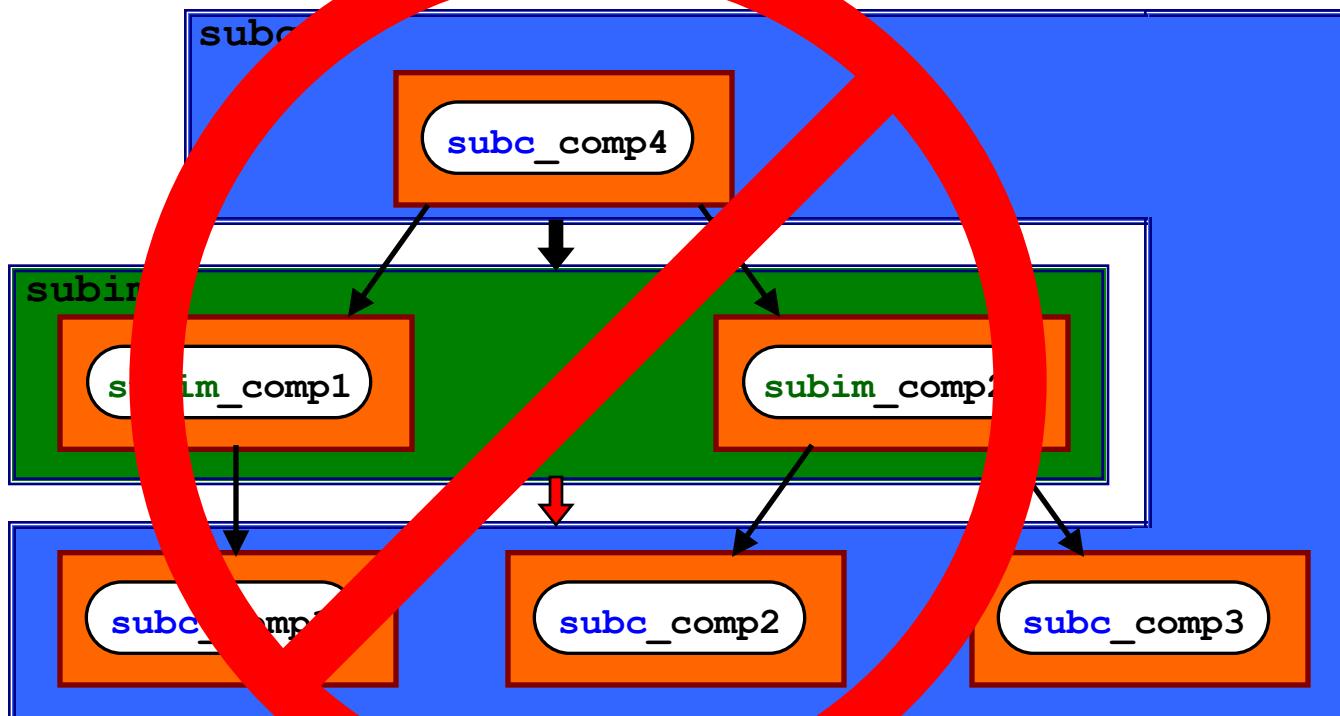
Package naming is more than just a convention:



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

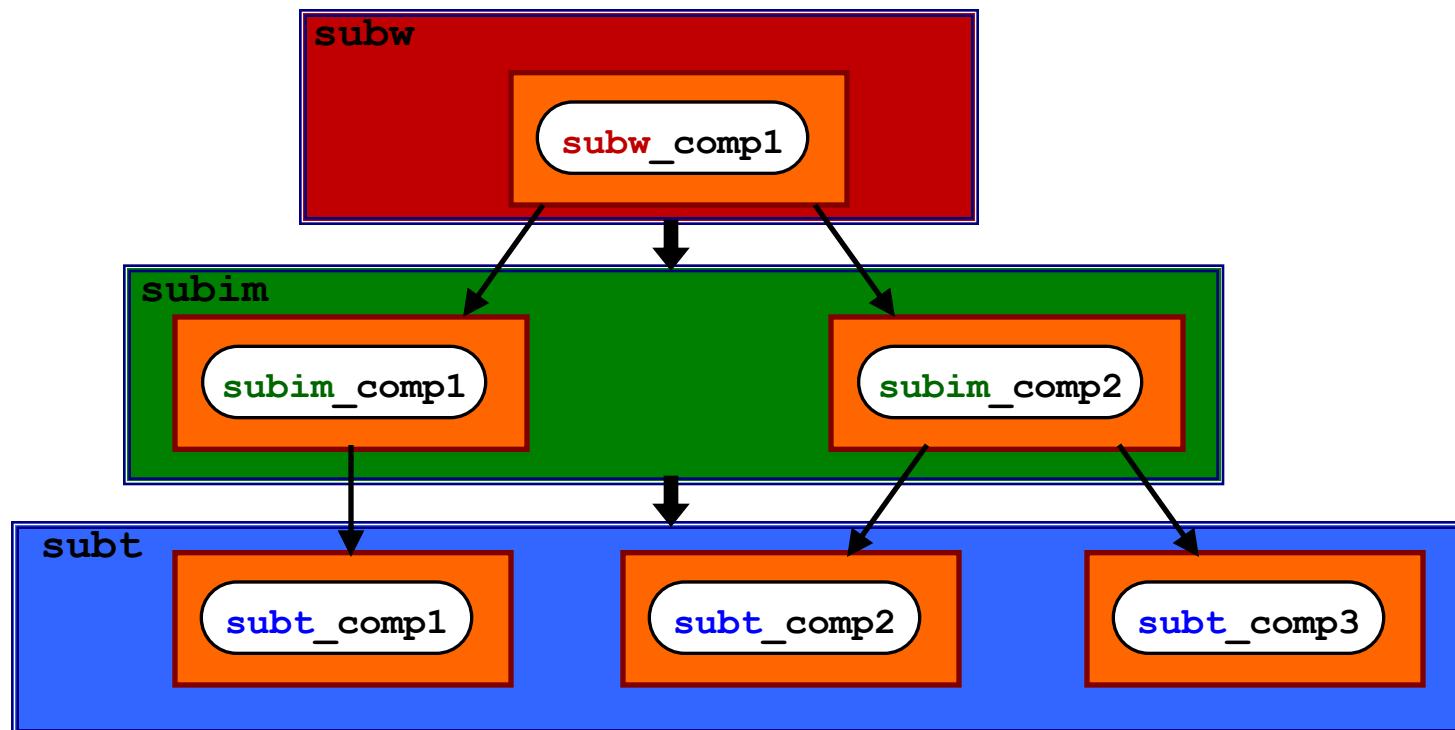
Package naming is more than just a convention:



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

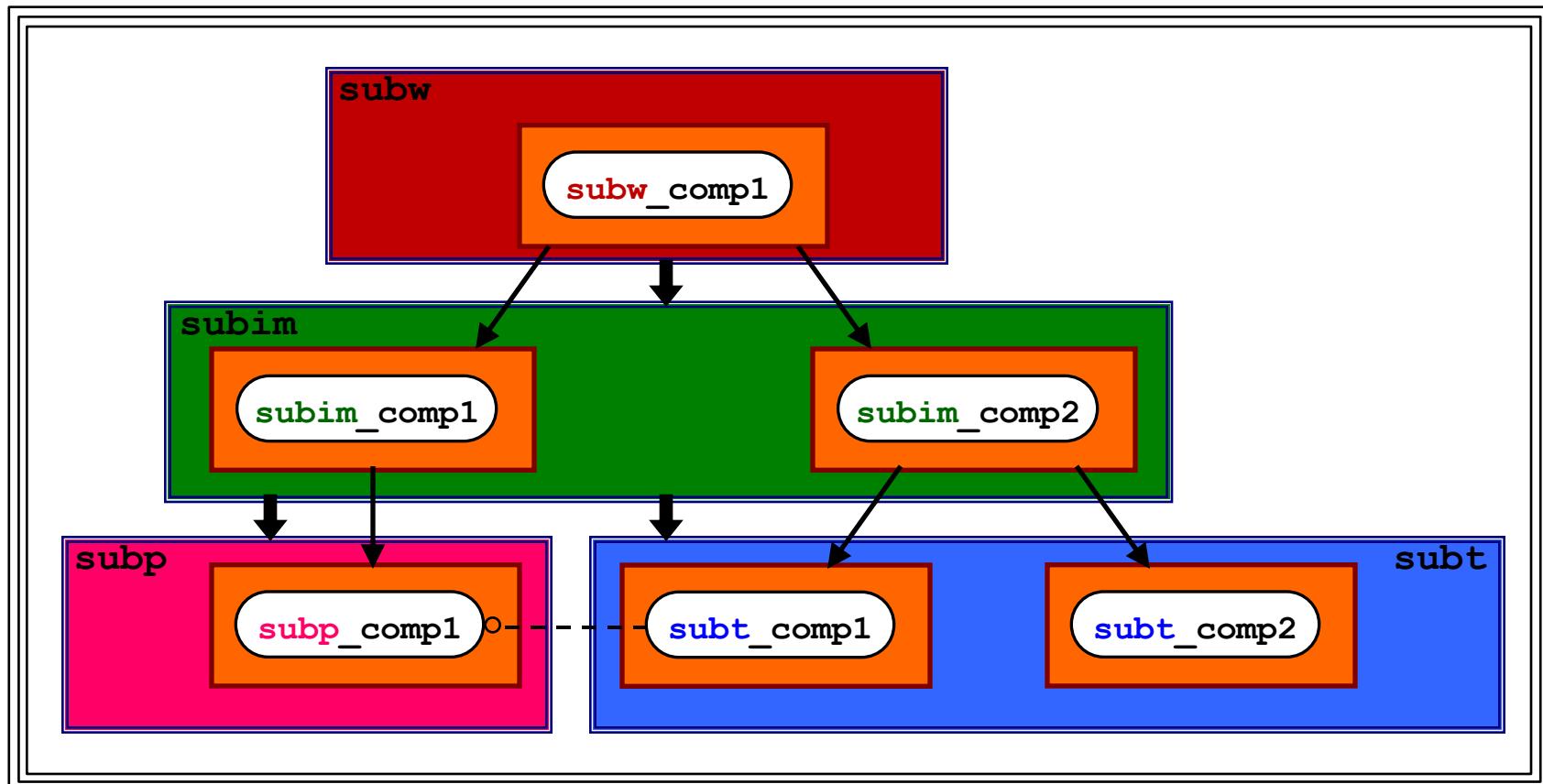
Package naming is more than just a convention:



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

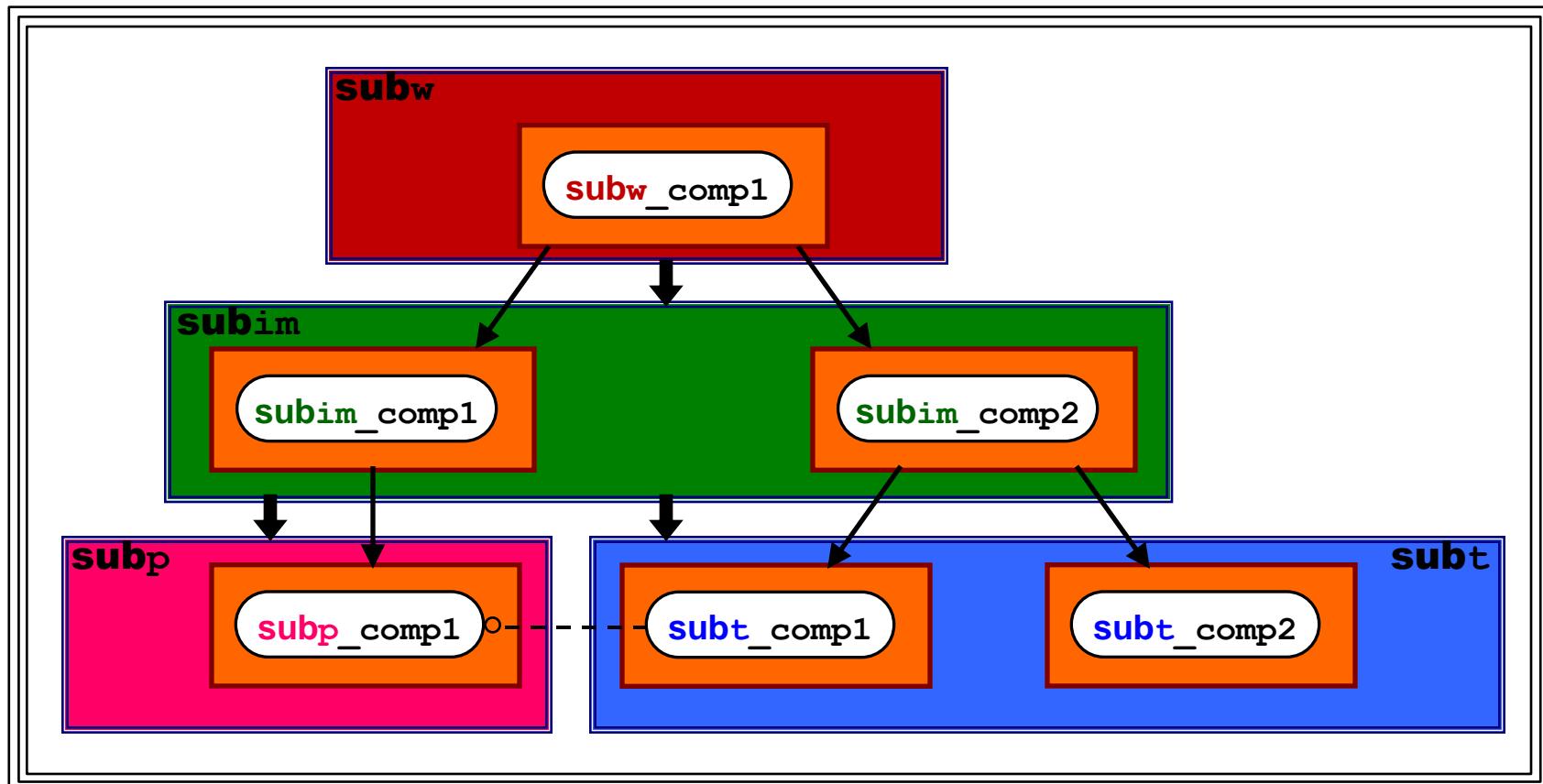
Package Group



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

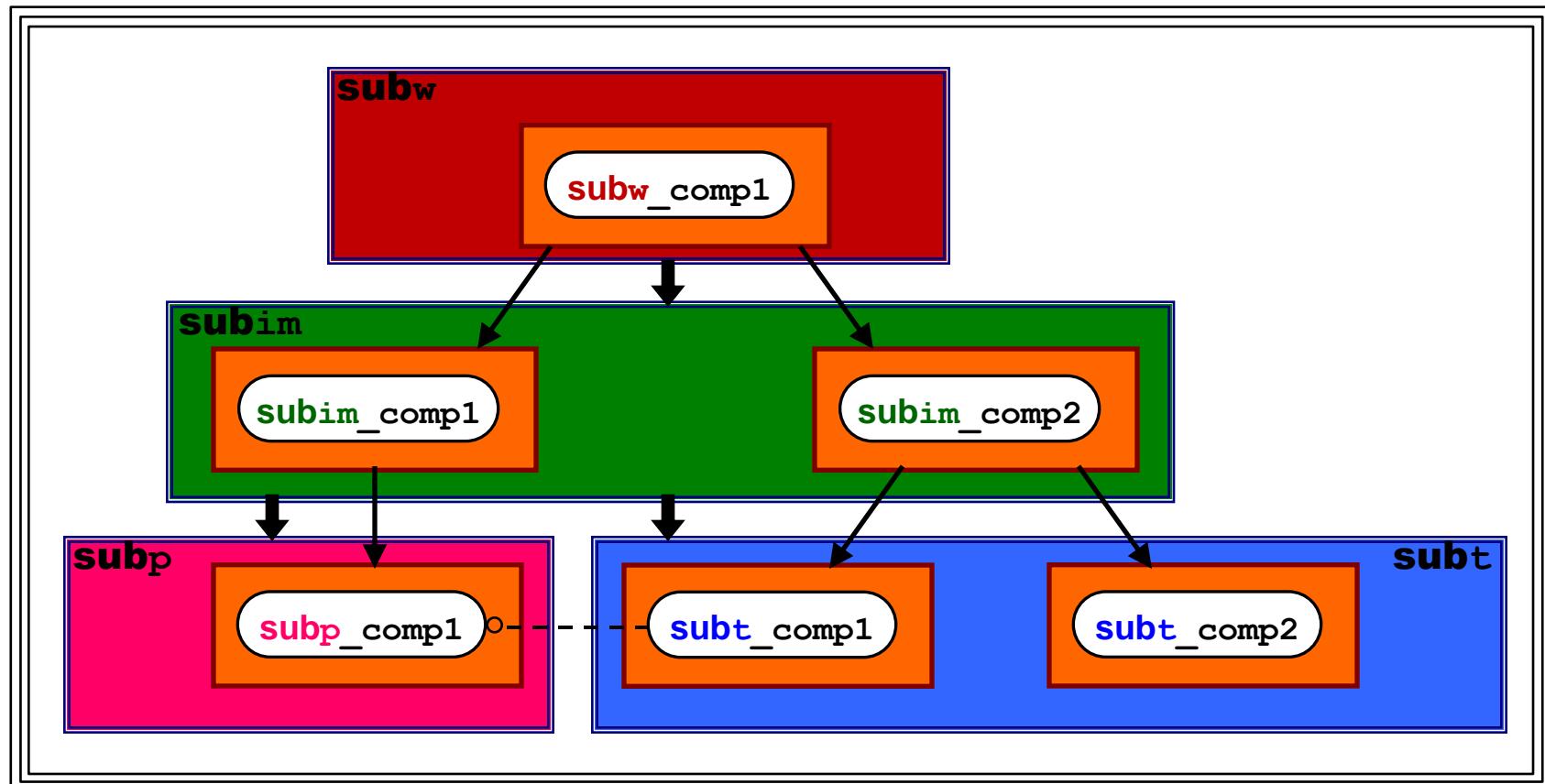
Package Group



2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

Package Group

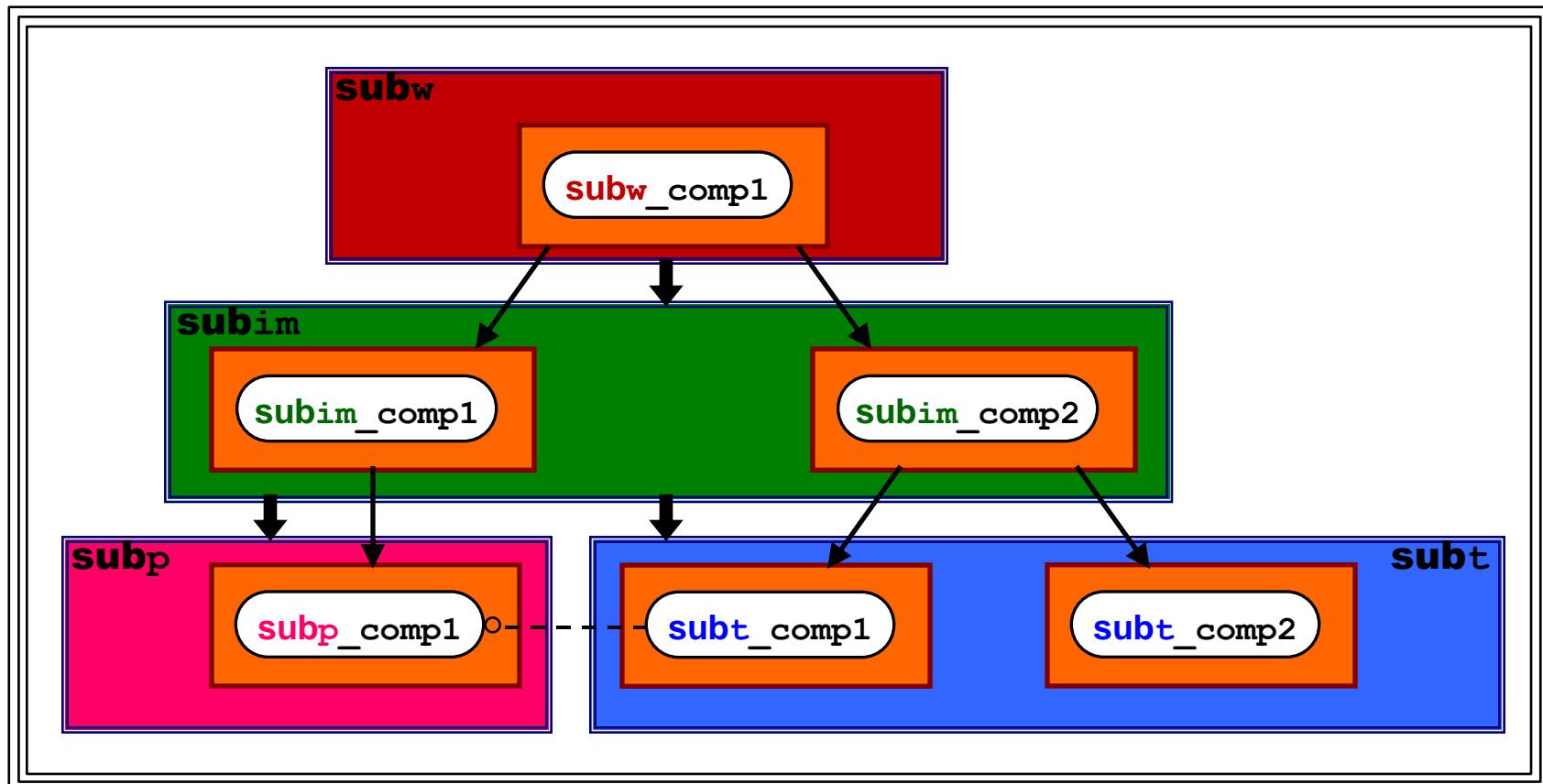


sub

2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

Package Group



sub ← Exactly Three Characters

2. Survey of Advanced *Levelization* Techniques

Escalating Encapsulation

Discussion?

2. Survey of Advanced *Levelization* Techniques

End of Section

Questions?

2. Survey of Advanced *Levelization* Techniques

Levelization Techniques (Summary)

Escalation – Moving mutually dependent functionality higher in the physical hierarchy.

Demotion – Moving common functionality lower in the physical hierarchy.

Opaque Pointers – Having an object use another *in name only*.

Dumb Data – Using data that indicates a dependency on a peer object, but only in the context of a separate, higher-level object.

Redundancy – Deliberately avoiding reuse by repeating a small amount of code or data to avoid coupling.

Callbacks – Client-supplied functions/data that enable lower-level subsystems to perform specific tasks in a more global context.

Manager Class – Establishing a class that owns and coordinates lower-level objects.

Factoring – Moving independently testable sub-behavior out of the implementation of a complex component involved in excessive physical coupling.

Escalating Encapsulation – Moving the point at which implementation details are hidden from clients to a higher level in the physical hierarchy.

Outline

1. Review of Elementary Physical Design
Components, Modularity, Physical Dependencies
2. Survey of Advanced *Levelization* Techniques
Avoiding Cyclic, Excessive, or Inappropriate Dependencies
3. Total and Partial *Insulation* Techniques
Avoiding Unnecessary Compile-Time Coupling
4. Present-Day, Real-World Design Examples
Prioritizing Physical Design to Improve Software Quality

Outline

1. Review of Elementary Physical Design
Components, Modularity, Physical Dependencies
2. Survey of Advanced *Levelization* Techniques
Avoiding Cyclic, Excessive, or Inappropriate Dependencies
3. Total and Partial *Insulation* Techniques
Avoiding Unnecessary Compile-Time Coupling
4. Present-Day, Real-World Design Examples
Prioritizing Physical Design to Improve Software Quality

3. Total and Partial *Insulation* Techniques

Insulation

3. Total and Partial *Insulation* Techniques

Insulation

Logical encapsulation versus *physical insulation*:

3. Total and Partial *Insulation* Techniques

Insulation

*Logical **encapsulation*** versus *physical **insulation***:

- An implementation detail of a component (type, data, or function) that can be altered, added, or removed *without* forcing clients to *rework their code* is said to be **encapsulated**.

3. Total and Partial *Insulation* Techniques

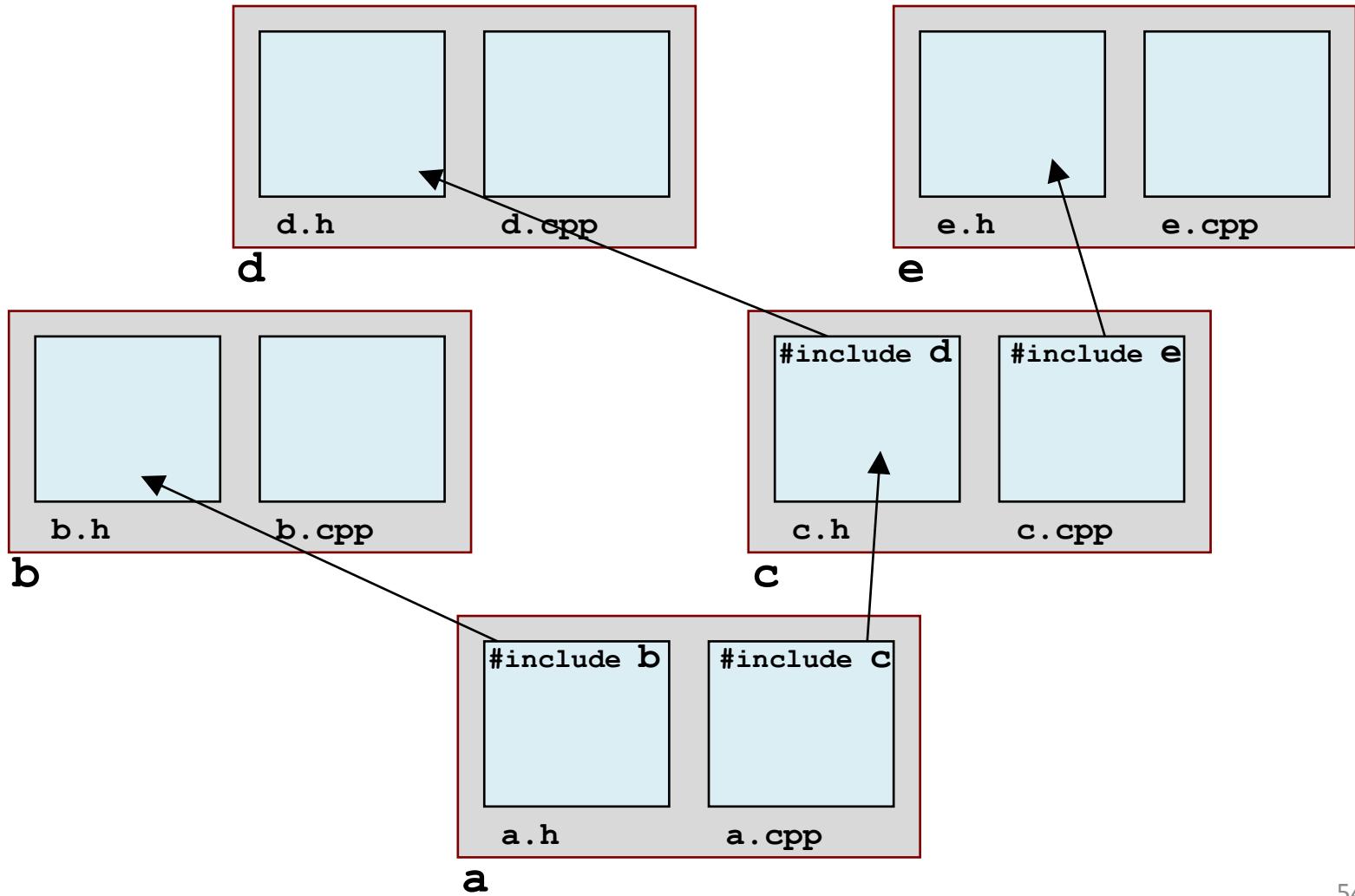
Insulation

*Logical **encapsulation** versus physical **insulation**:*

- An implementation detail of a component (type, data, or function) that can be altered, added, or removed *without* forcing clients to rework their code is said to be *encapsulated*.
- An implementation detail of a component (type, data, or function) that can be altered, added, or removed *without* forcing clients to recompile is said to be *insulated*.

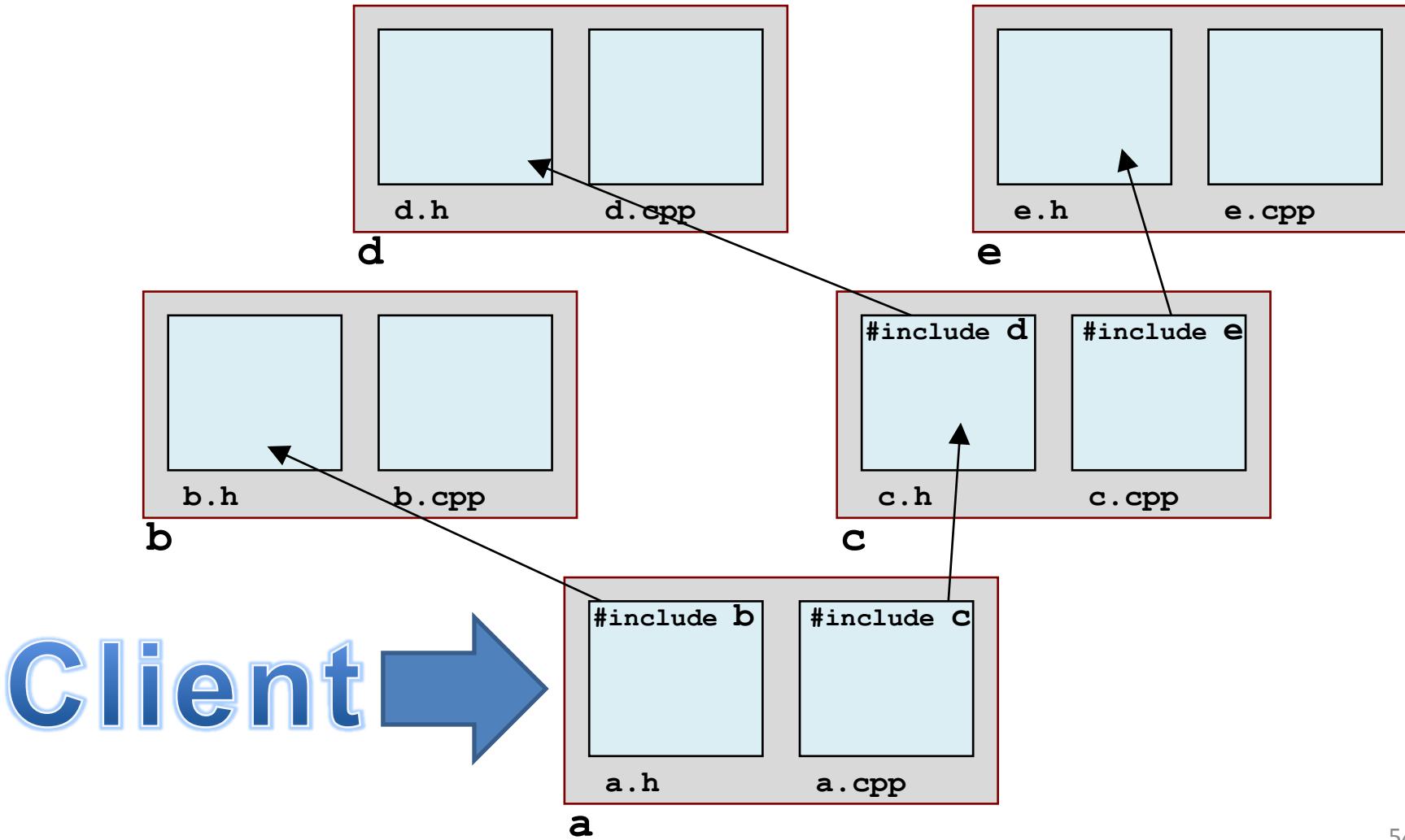
3. Total and Partial *Insulation* Techniques

Insulation



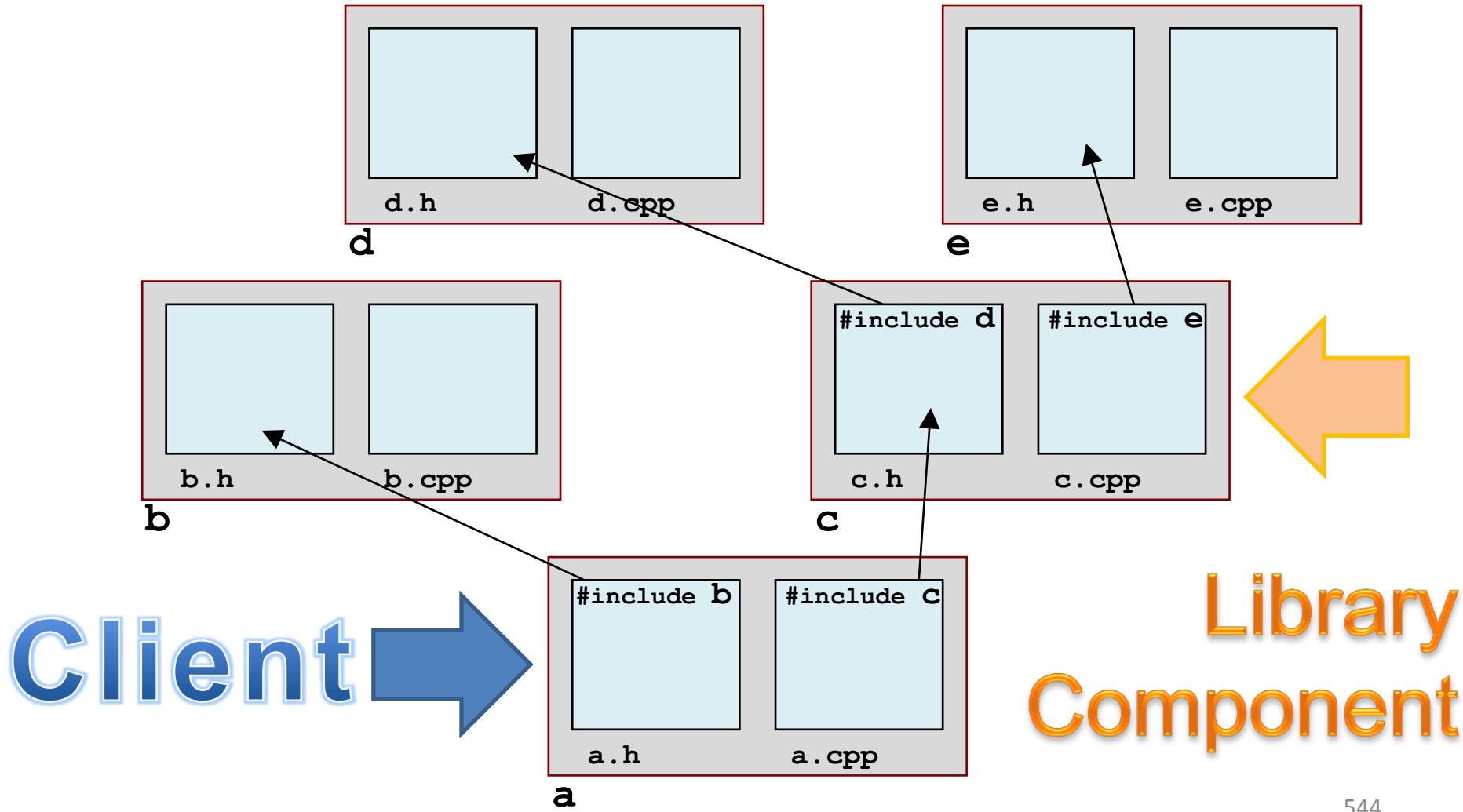
3. Total and Partial *Insulation* Techniques

Insulation



3. Total and Partial *Insulation* Techniques

Insulation



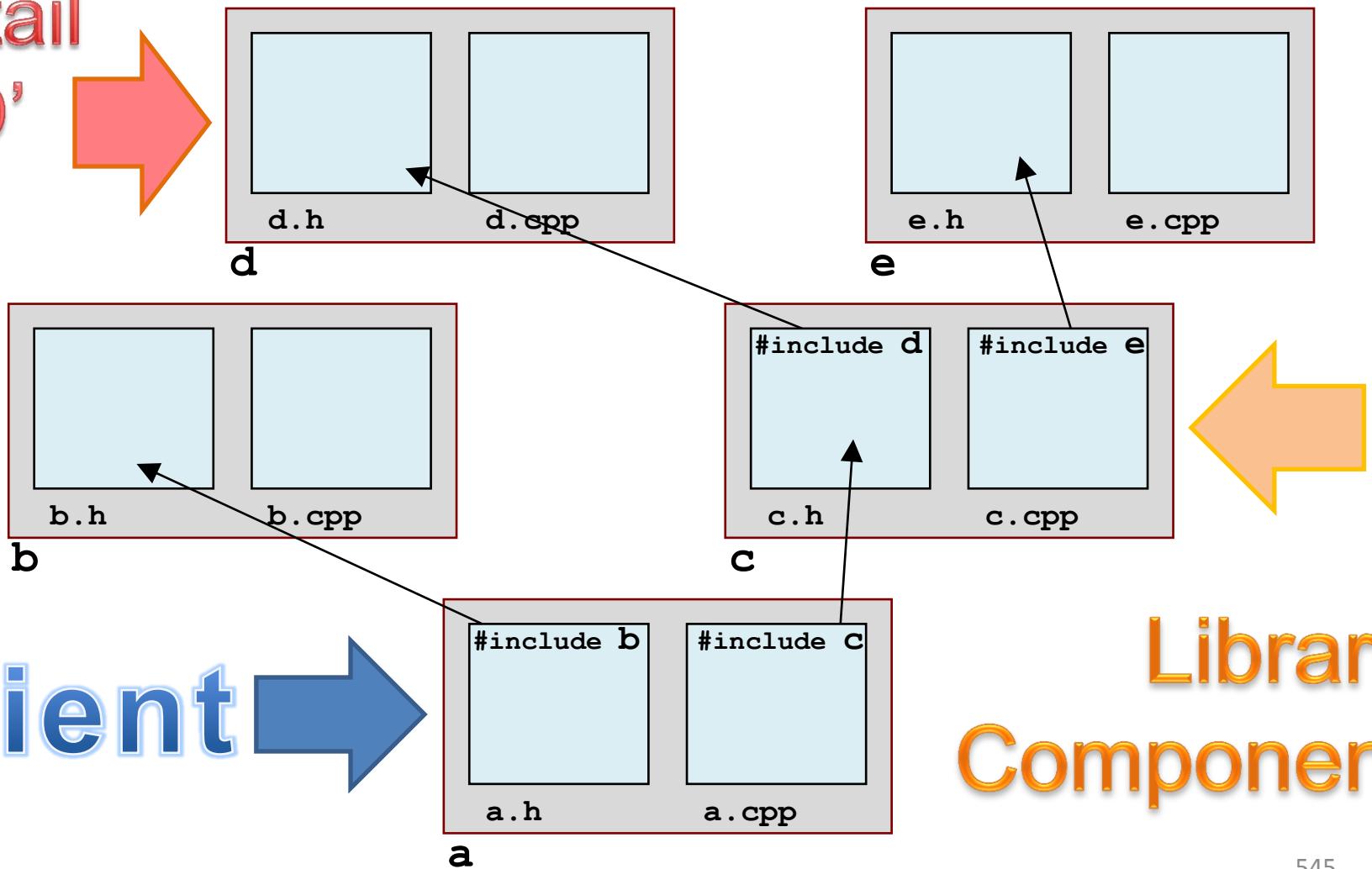
3. Total and Partial *Insulation* Techniques

Insulation

Imp

Detail

'D'

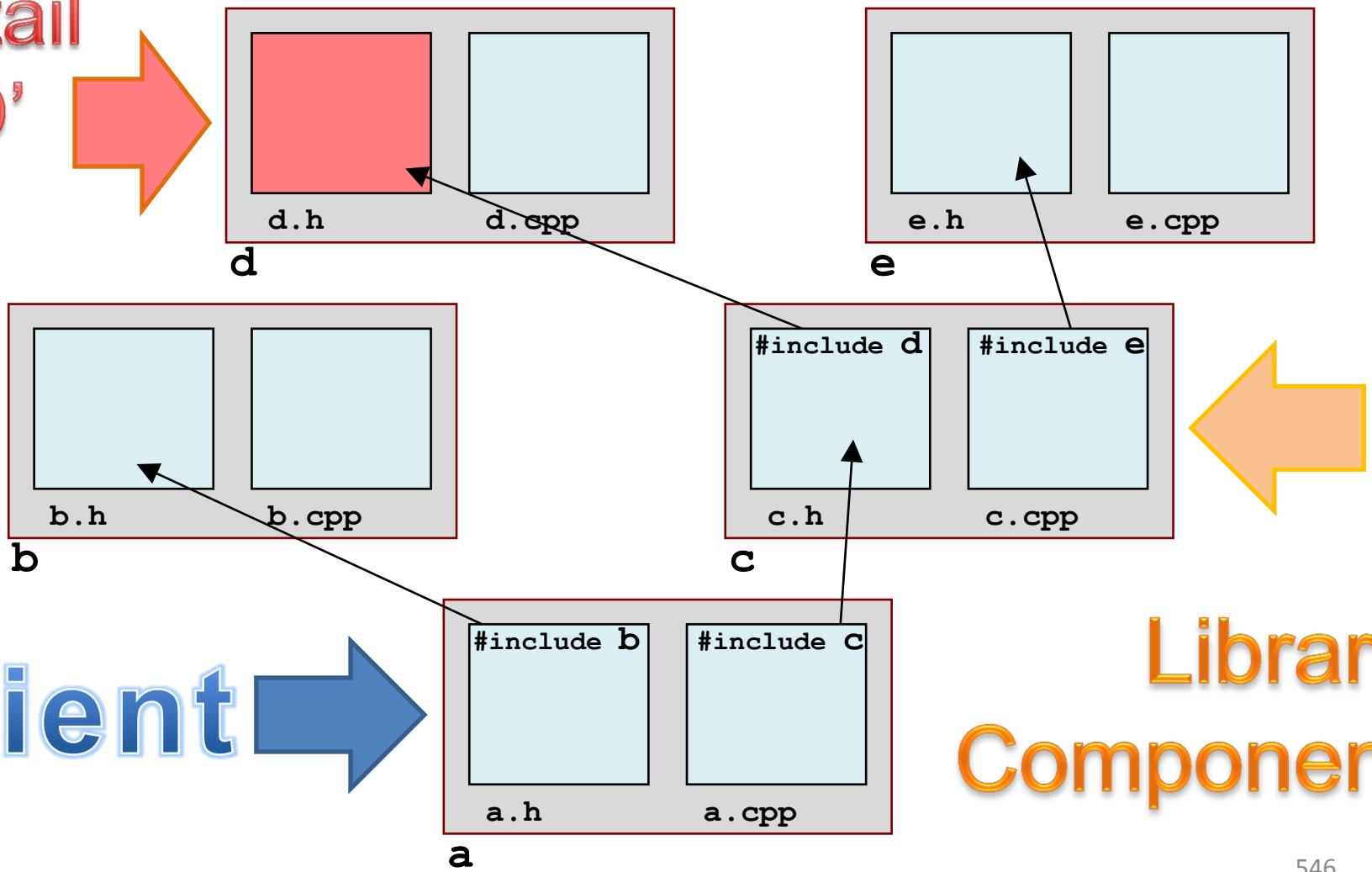


3. Total and Partial *Insulation* Techniques

Insulation

Imp
Detail

'D'

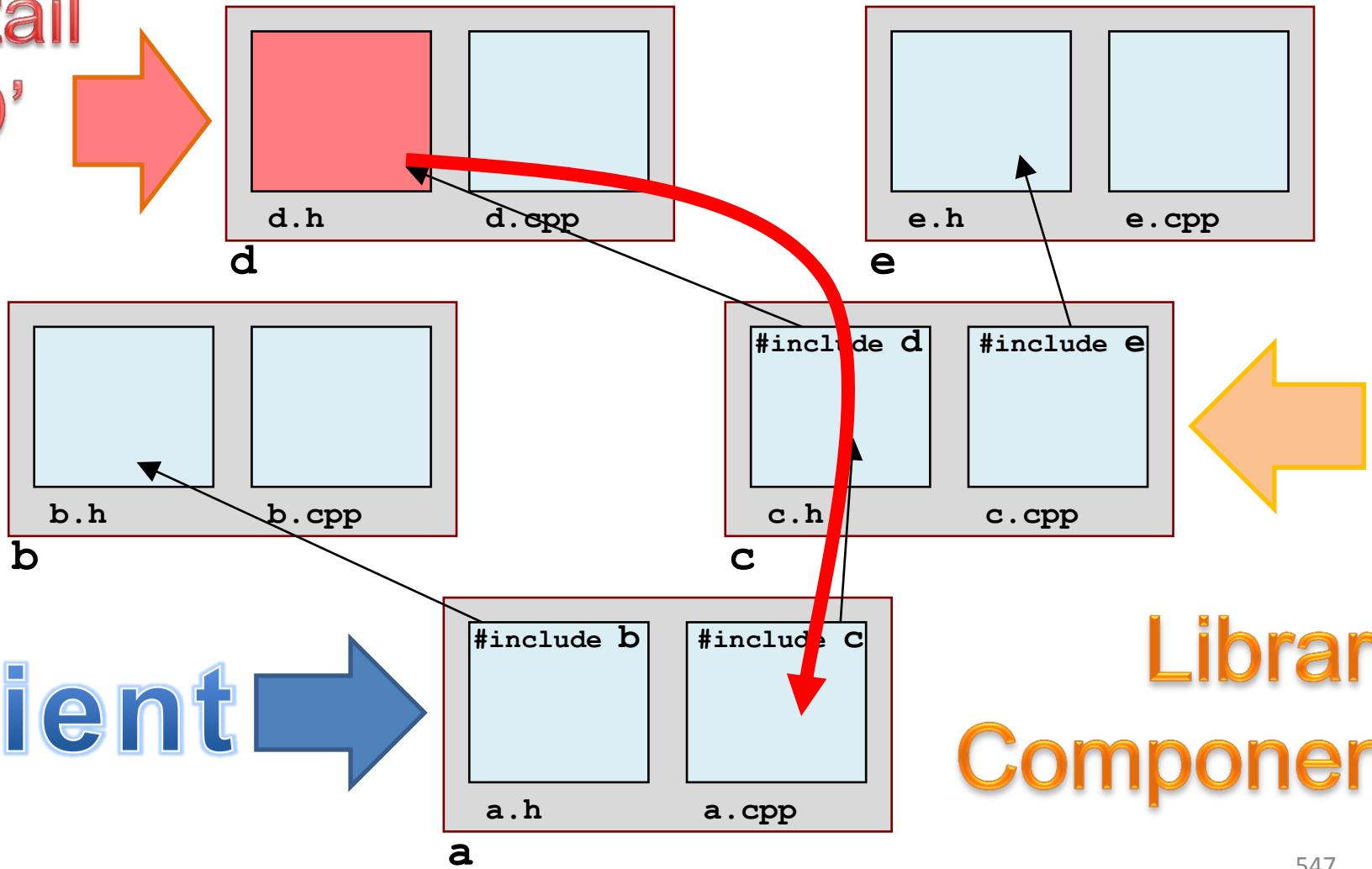


3. Total and Partial *Insulation* Techniques

Insulation

Imp
Detail

'D'

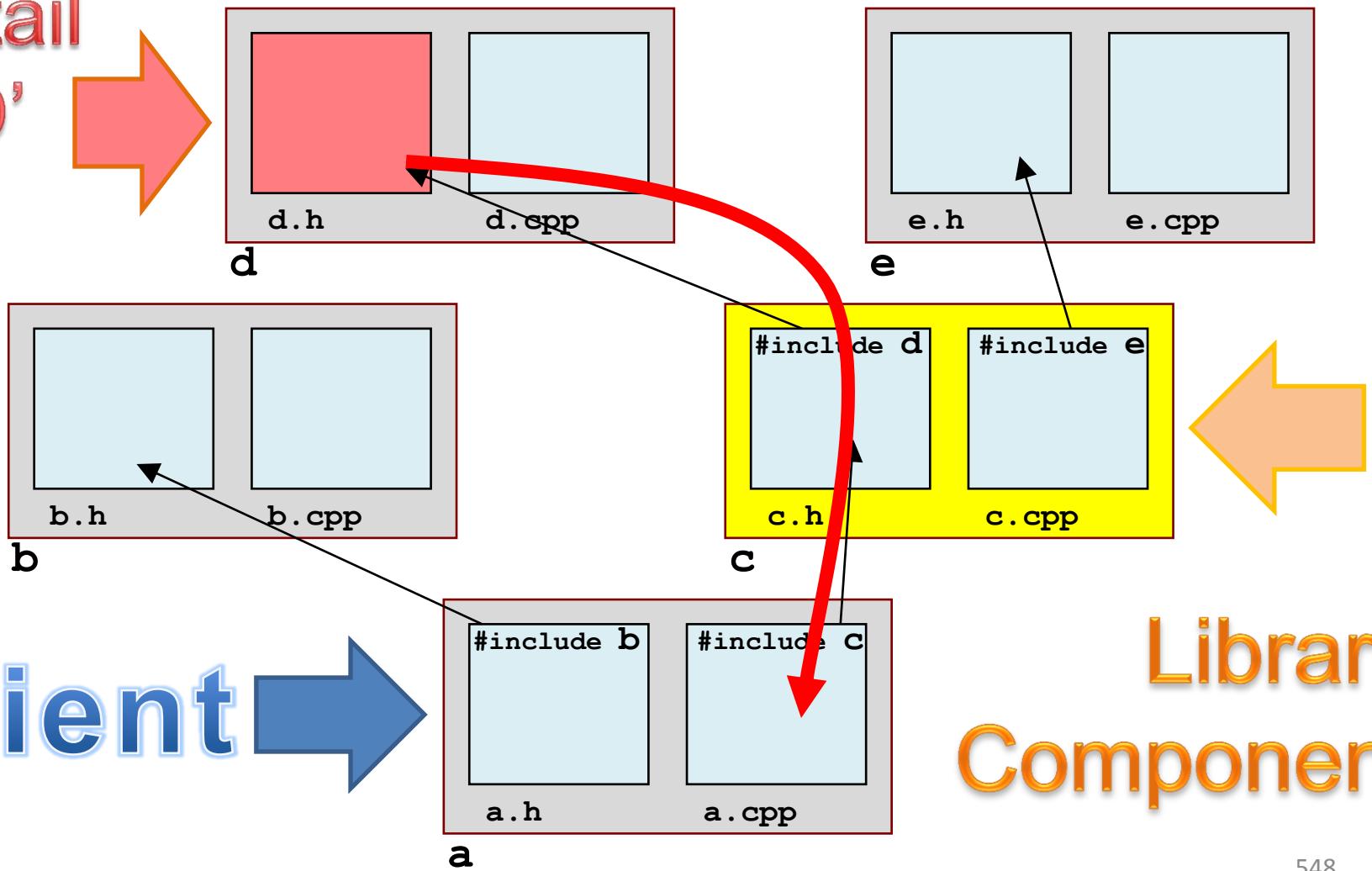


3. Total and Partial *Insulation* Techniques

Insulation

Imp
Detail

'D'

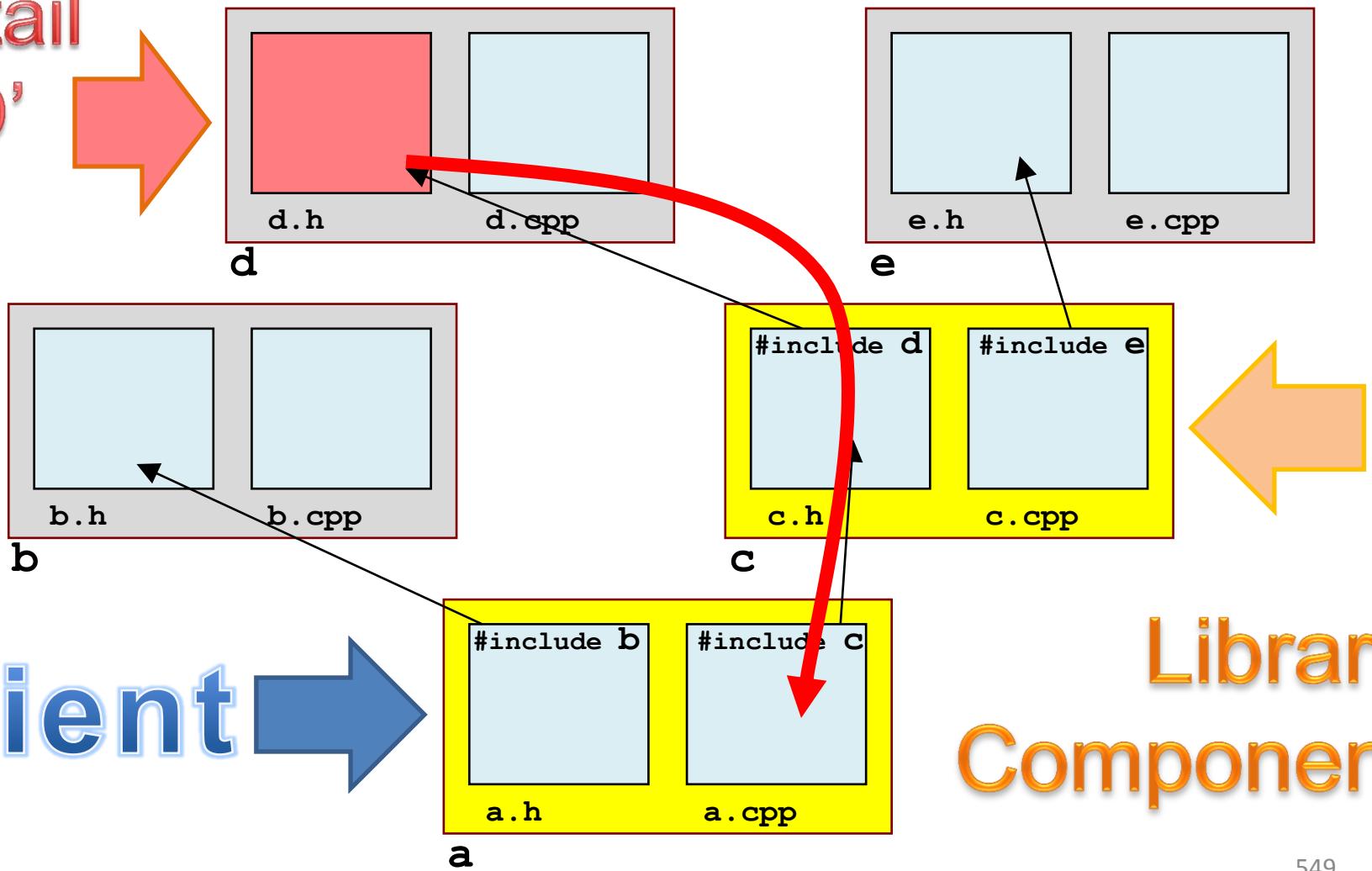


3. Total and Partial *Insulation* Techniques

Insulation

Imp
Detail

'D'



Library

Component

3. Total and Partial *Insulation* Techniques

Imp
Detail

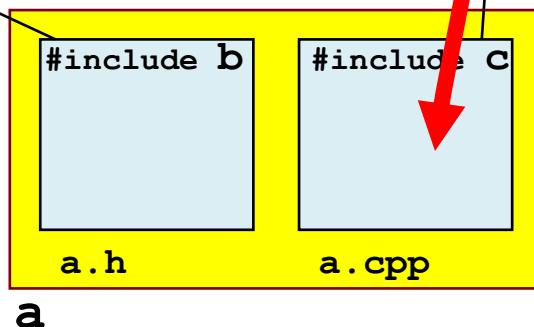
Insulation

'D' is
Encapsulated
by 'C'

b

b.cpp

Client →



a

c

c.h

#include d

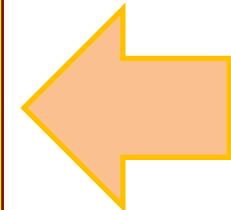
c.cpp

#include e

e

e.h

e.cpp



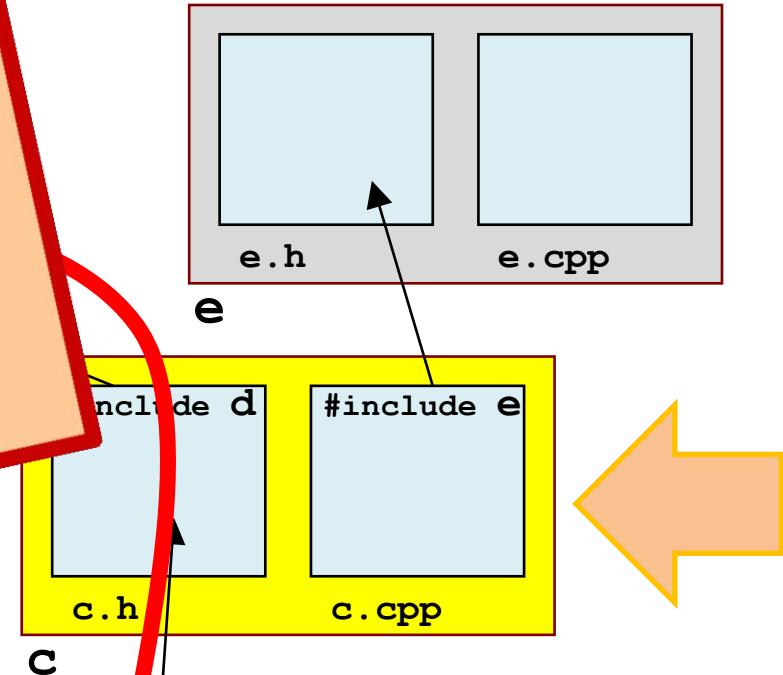
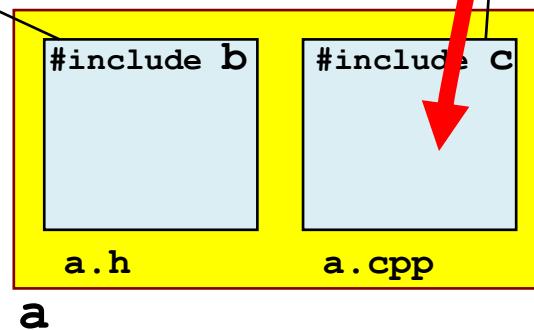
Library
Component

3. Total and Partial *Insulation* Techniques

Imp
Detail

(Use of) 'D' is
Encapsulated
by 'C'

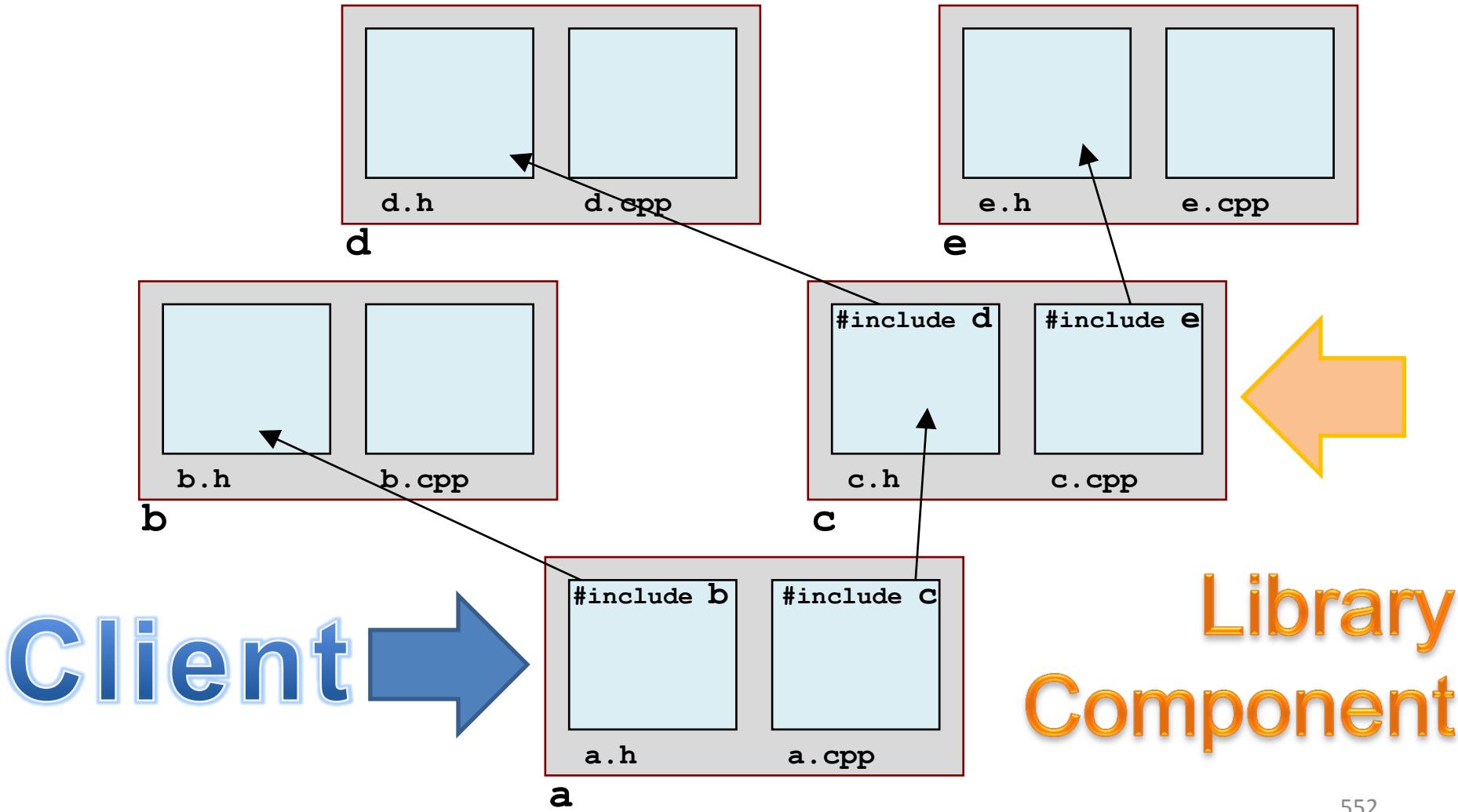
Client →



Library
Component

3. Total and Partial *Insulation* Techniques

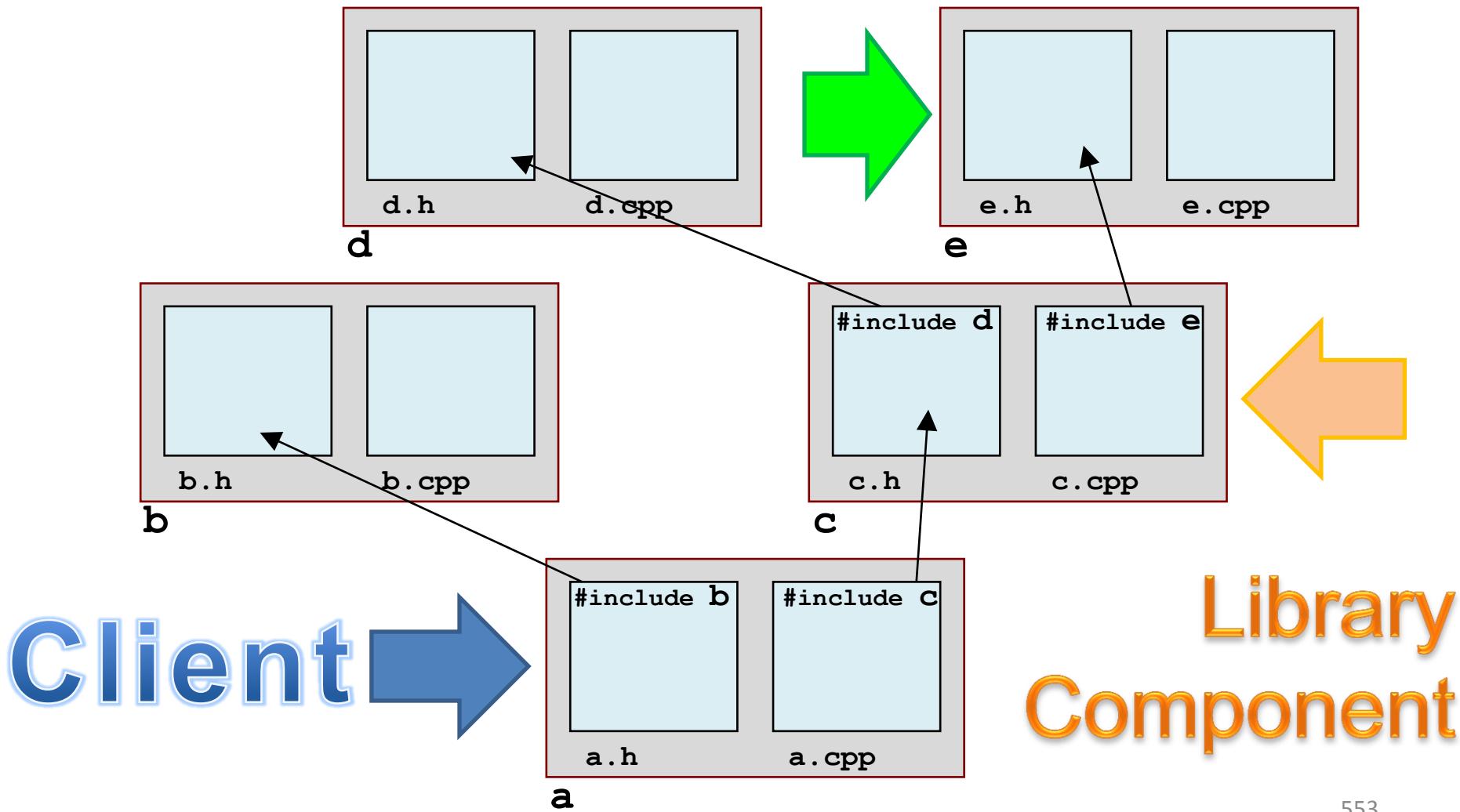
Insulation



3. Total and Partial *Insulation* Techniques

Insulation

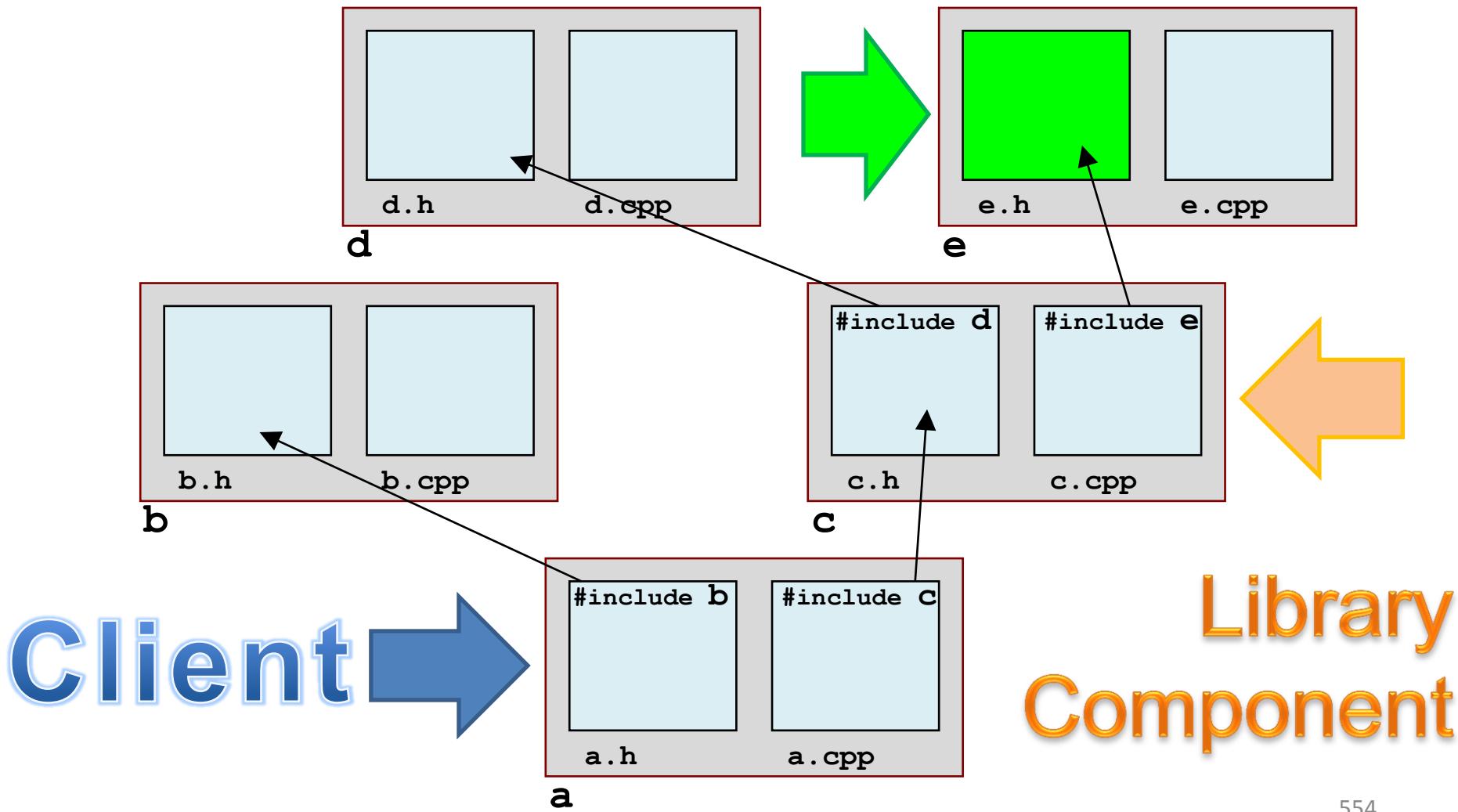
Imp
Detail 'E'



3. Total and Partial *Insulation* Techniques

Insulation

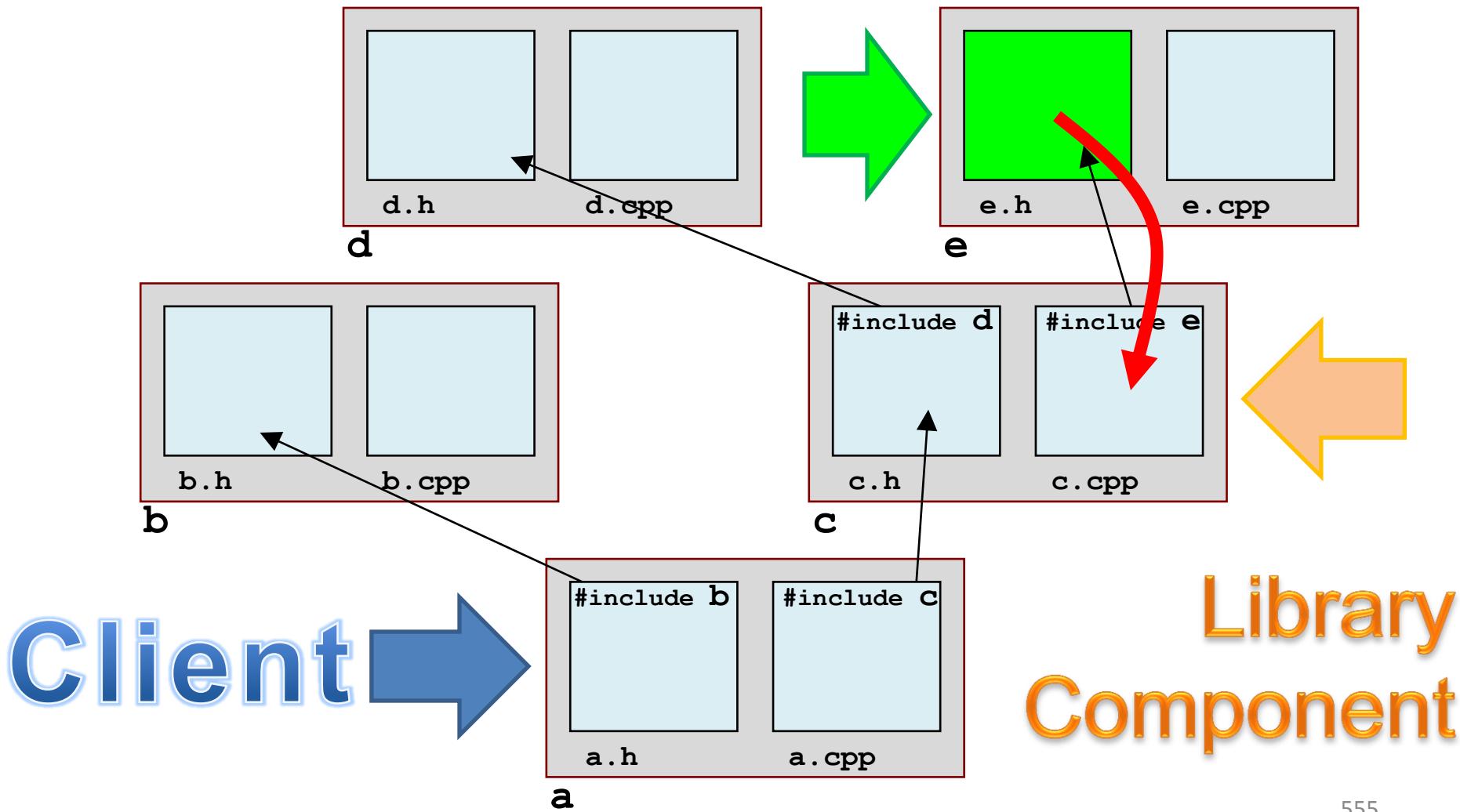
Imp
Detail 'E'



3. Total and Partial *Insulation* Techniques

Insulation

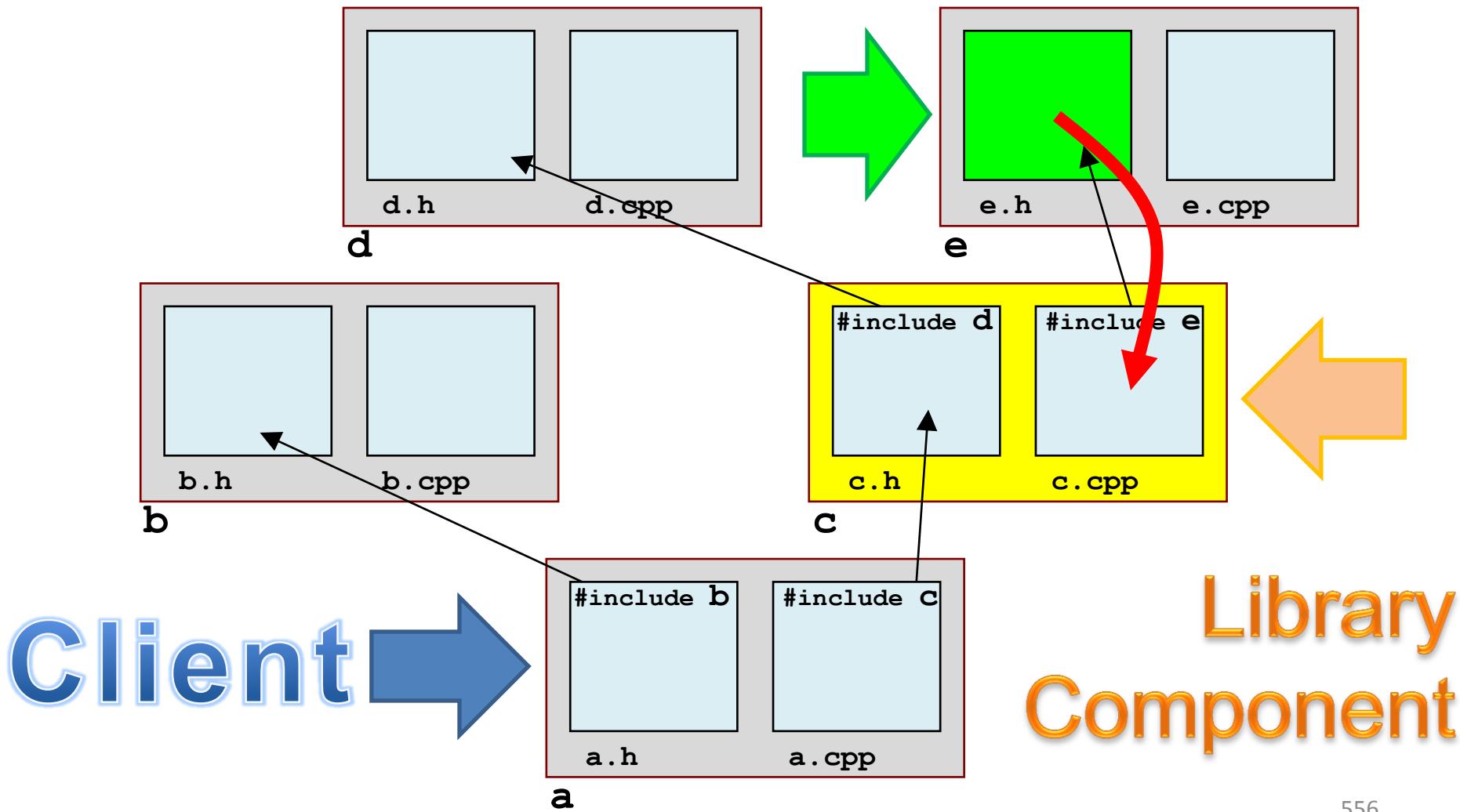
Imp
Detail 'E'



3. Total and Partial *Insulation* Techniques

Insulation

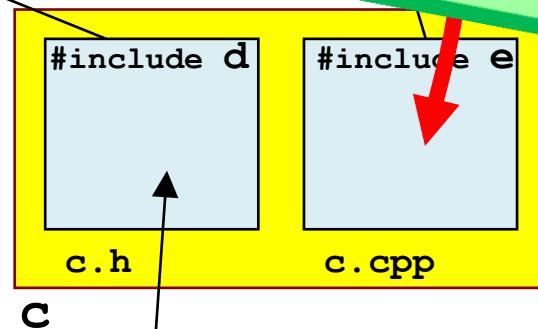
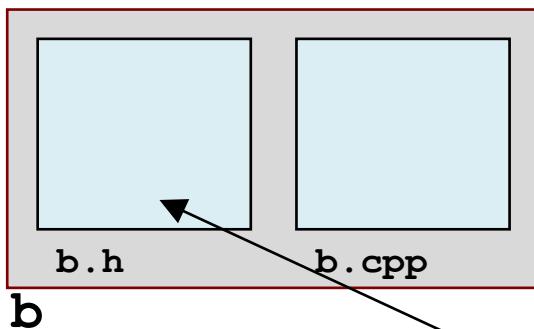
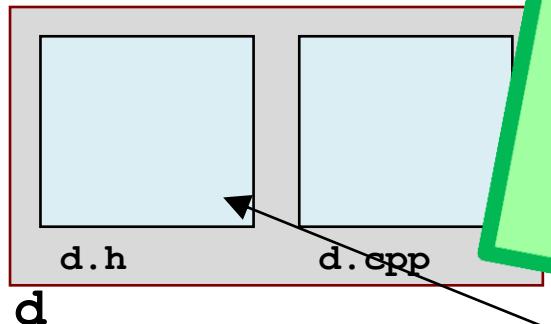
Imp
Detail 'E'



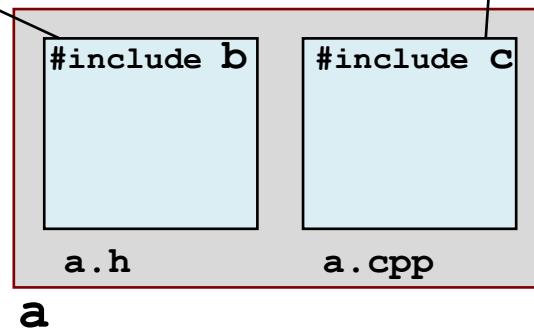
3. Total and Partial Insulation Techniques

Insulation

**'E' is
Insulated
by 'C'**

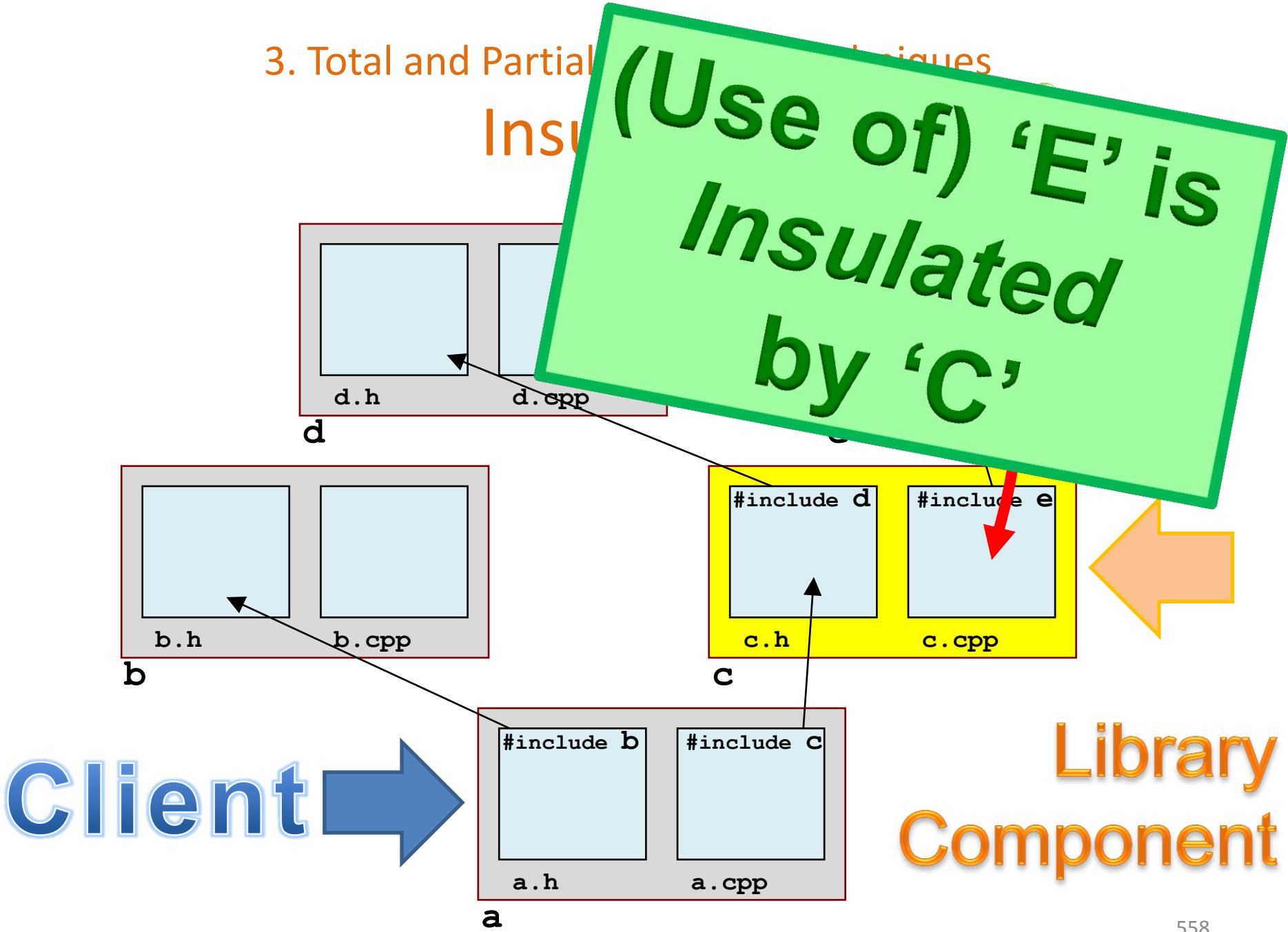


Client →



Library Component

3. Total and Partial Insulation



3. Total and Partial *Insulation* Techniques

Criteria for having a #include in a .h File

3. Total and Partial *Insulation* Techniques

Criteria for having a `#include` in a .h File

Recall that:

A header file must
be “self-sufficient”
w.r.t. compilation.

3. Total and Partial *Insulation* Techniques

Criteria for having a #include in a .h File

There are five:

3. Total and Partial *Insulation* Techniques

Criteria for having a #include in a .h File

There are five:

1. Is-A

3. Total and Partial *Insulation* Techniques

Criteria for having a #include in a .h File

There are five:

1. *Is-A*
2. *Has-A*

3. Total and Partial *Insulation* Techniques

Criteria for having a #include in a .h File

There are five:

1. Is-A

2. Has-A



But not Uses !

3. Total and Partial *Insulation* Techniques

Criteria for having a #include in a .h File

There are five:

1. Is-A

2. Has-A



But not Uses !

```
#include <point.h>
Point appendVertex(int index,
                    const Point& vertex);
```

3. Total and Partial *Insulation* Techniques

Criteria for having a #include in a .h File

There are five:

1. Is-A

2. Has-A



But not Uses !

#include <point.h>

Point appendVertex(int index,
const Point& vertex);



3. Total and Partial *Insulation* Techniques

Criteria for having a #include in a .h File

There are five:

1. Is-A

2. Has-A



But not Uses!

```
class Point; ←  
Point appendVertex(int index,  
                   const Point& vertex);
```

3. Total and Partial *Insulation* Techniques

Criteria for having a #include in a .h File

There are five:

1. Is-A

2. Has-A



But not Uses !

#include <point.h>

Point appendVertex(int index,
const Point& vertex);



3. Total and Partial *Insulation* Techniques

Criteria for having a #include in a .h File

There are five:

1. Is-A

2. Has-A



But not Uses !

```
class Point; ←  
Point appendVertex(int index,  
                   const Point& vertex);
```

3. Total and Partial *Insulation* Techniques

Criteria for having a #include in a .h File

There are five:

1. *Is-A*
2. *Has-A*

3. Total and Partial *Insulation* Techniques

Criteria for having a #include in a .h File

There are five:

1. *Is-A*
2. *Has-A*
3. `inline` (used in function body)

3. Total and Partial *Insulation* Techniques

Criteria for having a #include in a .h File

There are five:

1. *Is-A*
2. *Has-A*
3. inline (used in function body)
4. enum

3. Total and Partial *Insulation* Techniques

Criteria for having a #include in a .h File

There are five:

1. *Is-A*
2. *Has-A*
3. `inline` (used in function body)
4. `enum`
5. `typedef` (e.g., template specialization)

3. Total and Partial *Insulation* Techniques

Criteria for having a #include in a .h File

There are five:

1. *Is-A*
2. *Has-A*
3. `inline` (used in function body)
4. `enum`
5. `typedef` (e.g., template specialization)

Note: Covariant return types is another edge case.

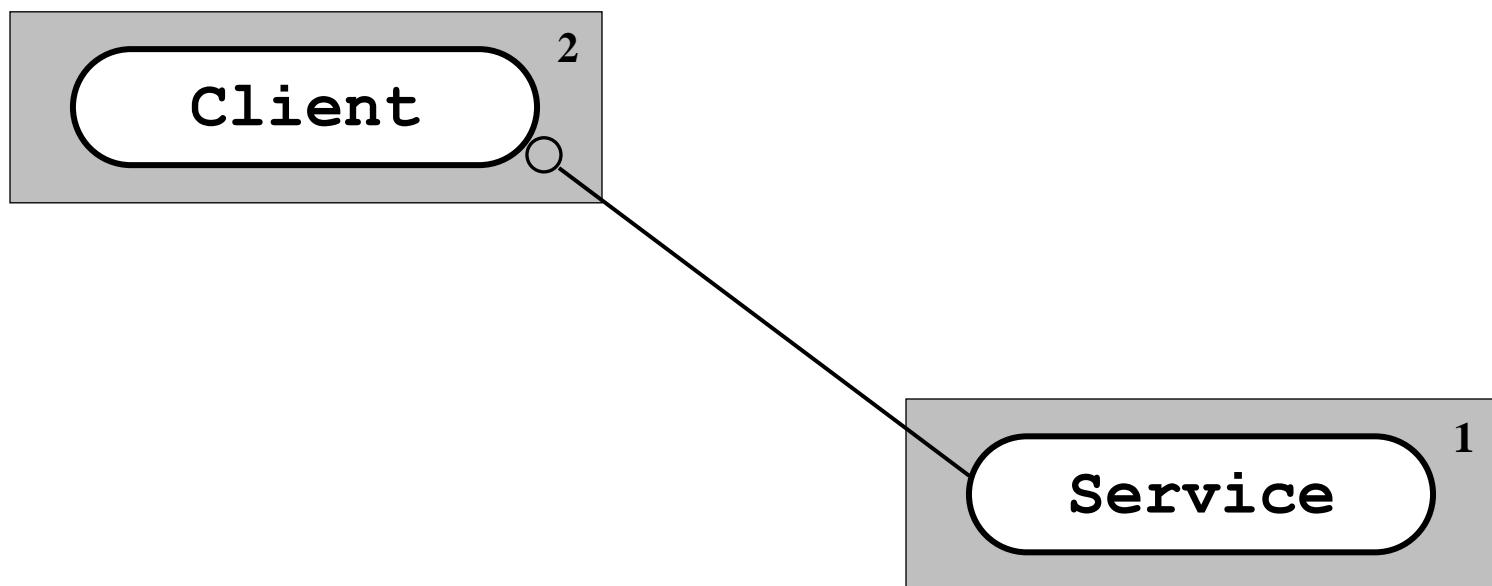
3. Total and Partial *Insulation* Techniques

Total Insulation Techniques

- I. Pure Abstract Interface
(Protocol Class)
- II. Fully Insulating Concrete Class
("Pimple")
- III. Procedural Interface

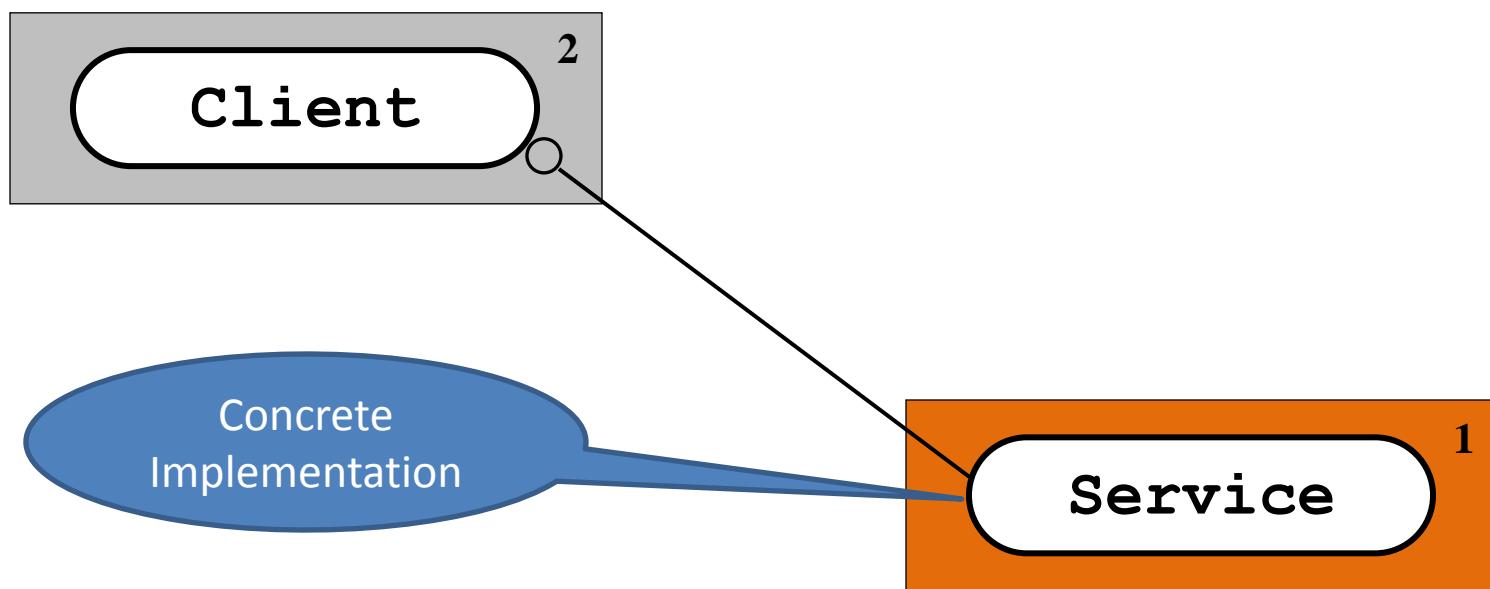
3. Total and Partial *Insulation* Techniques

Pure Abstract Interface (Protocol Class)



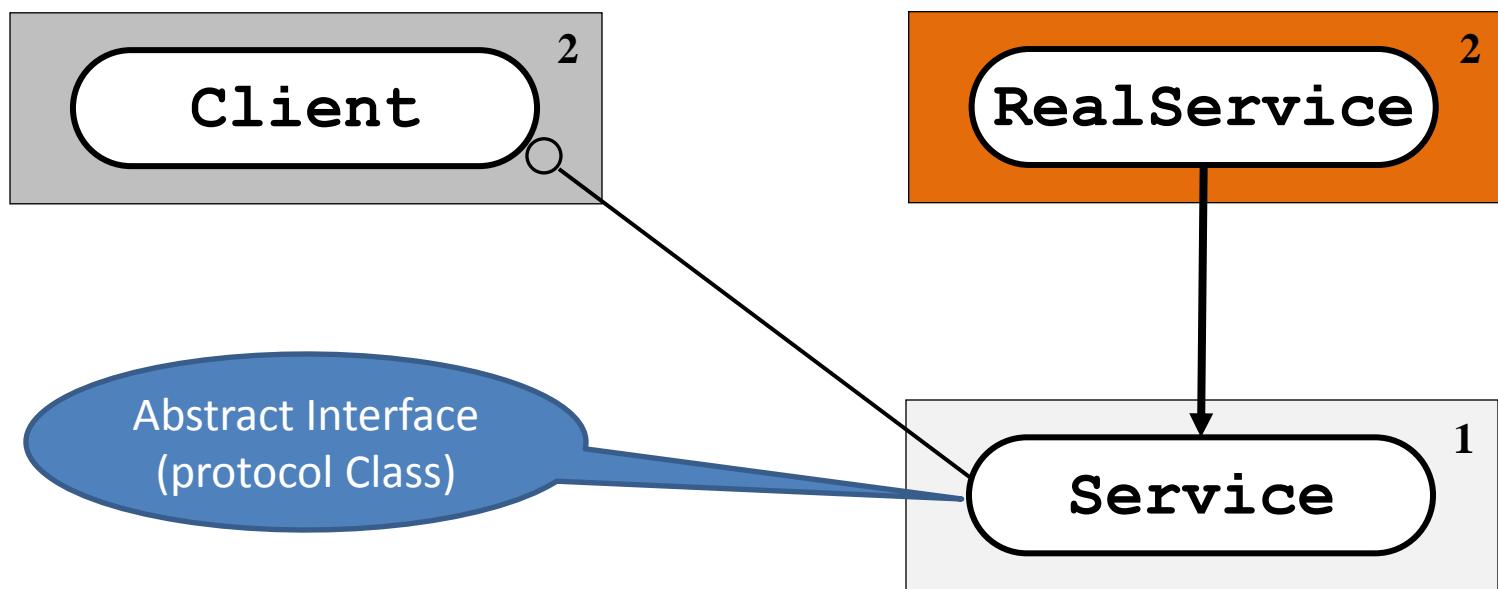
3. Total and Partial *Insulation* Techniques

Pure Abstract Interface (Protocol Class)



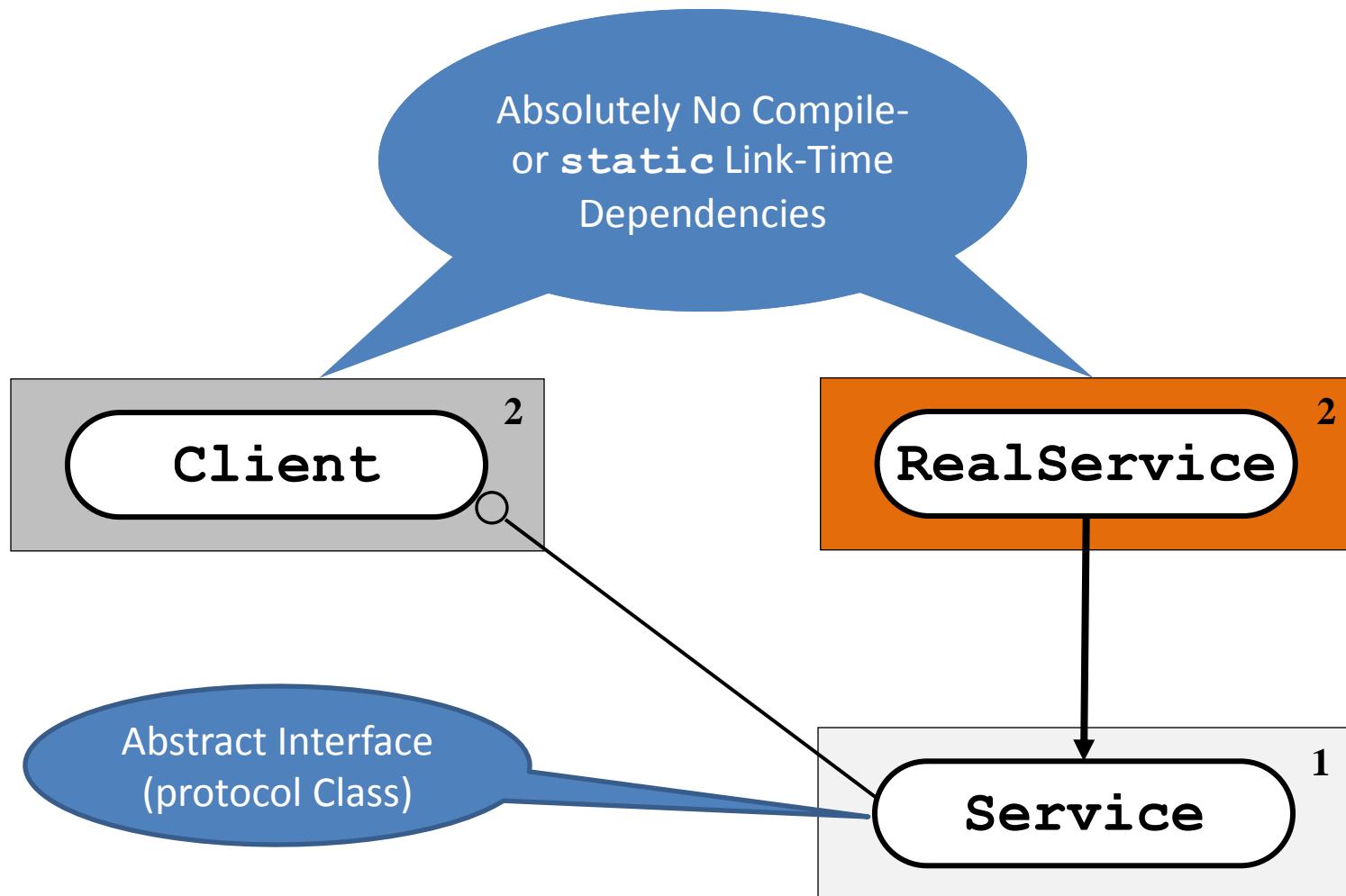
3. Total and Partial *Insulation* Techniques

Pure Abstract Interface (Protocol Class)



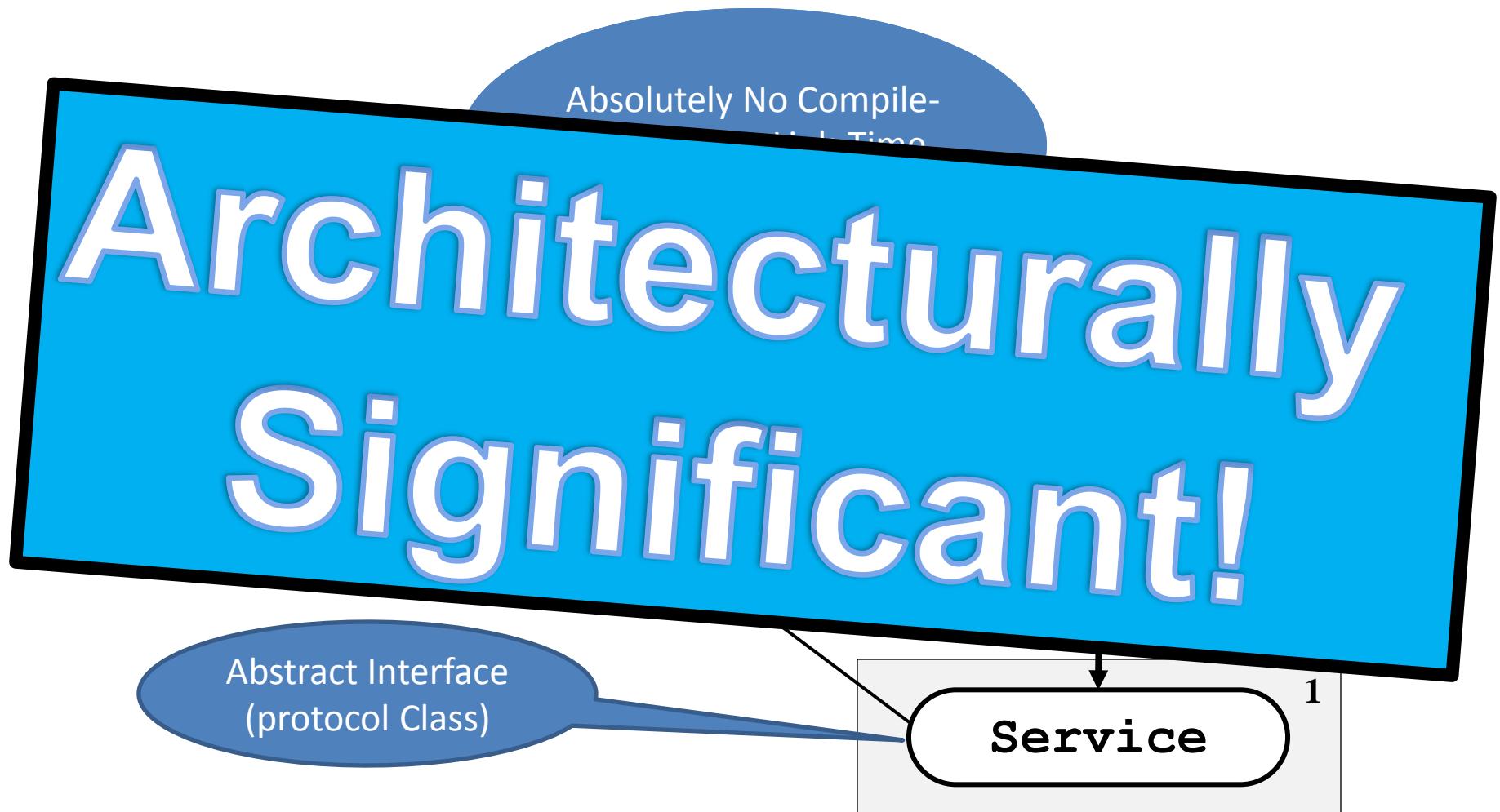
3. Total and Partial *Insulation* Techniques

Pure Abstract Interface (Protocol Class)



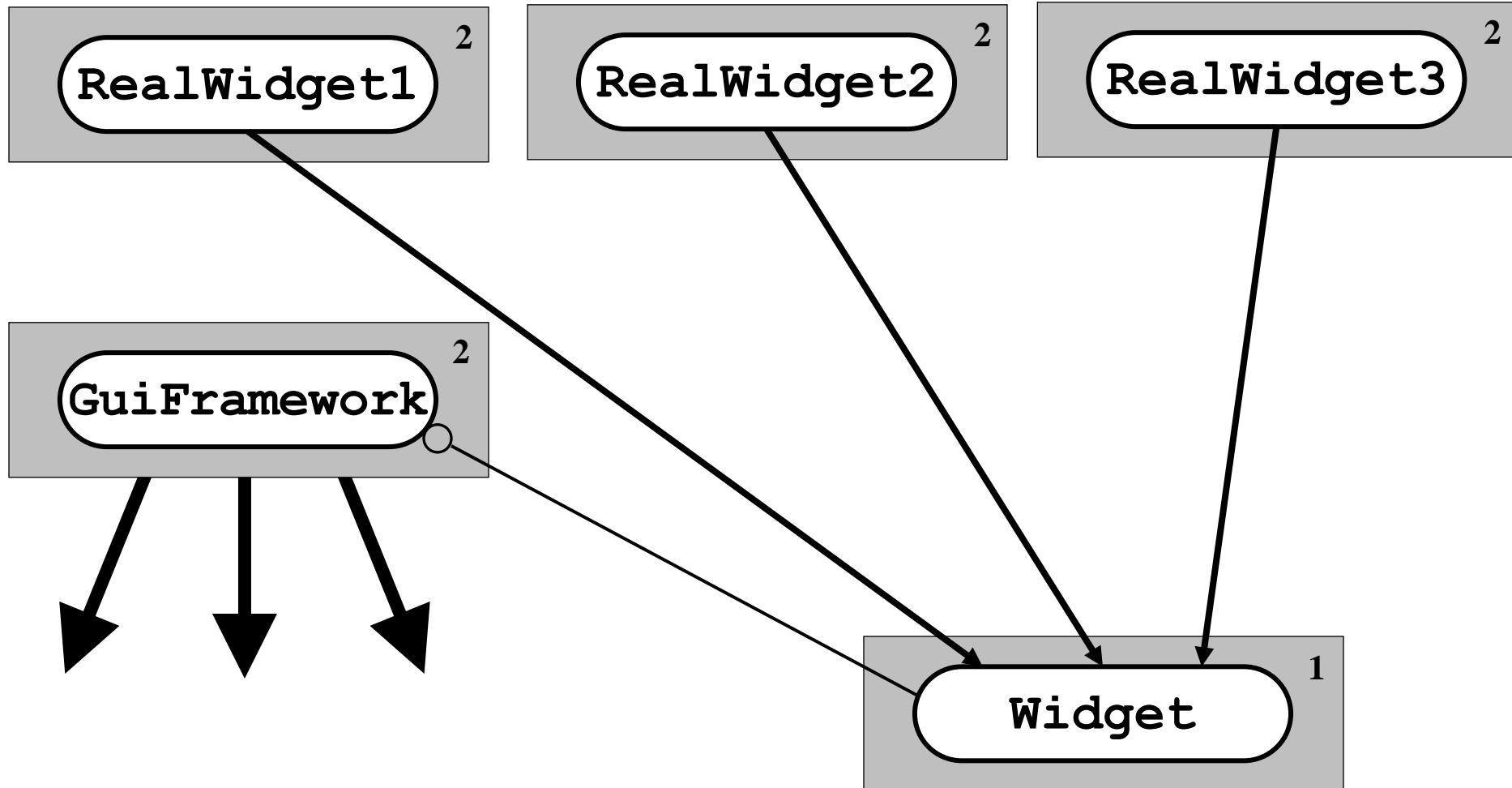
3. Total and Partial *Insulation* Techniques

Pure Abstract Interface (Protocol Class)



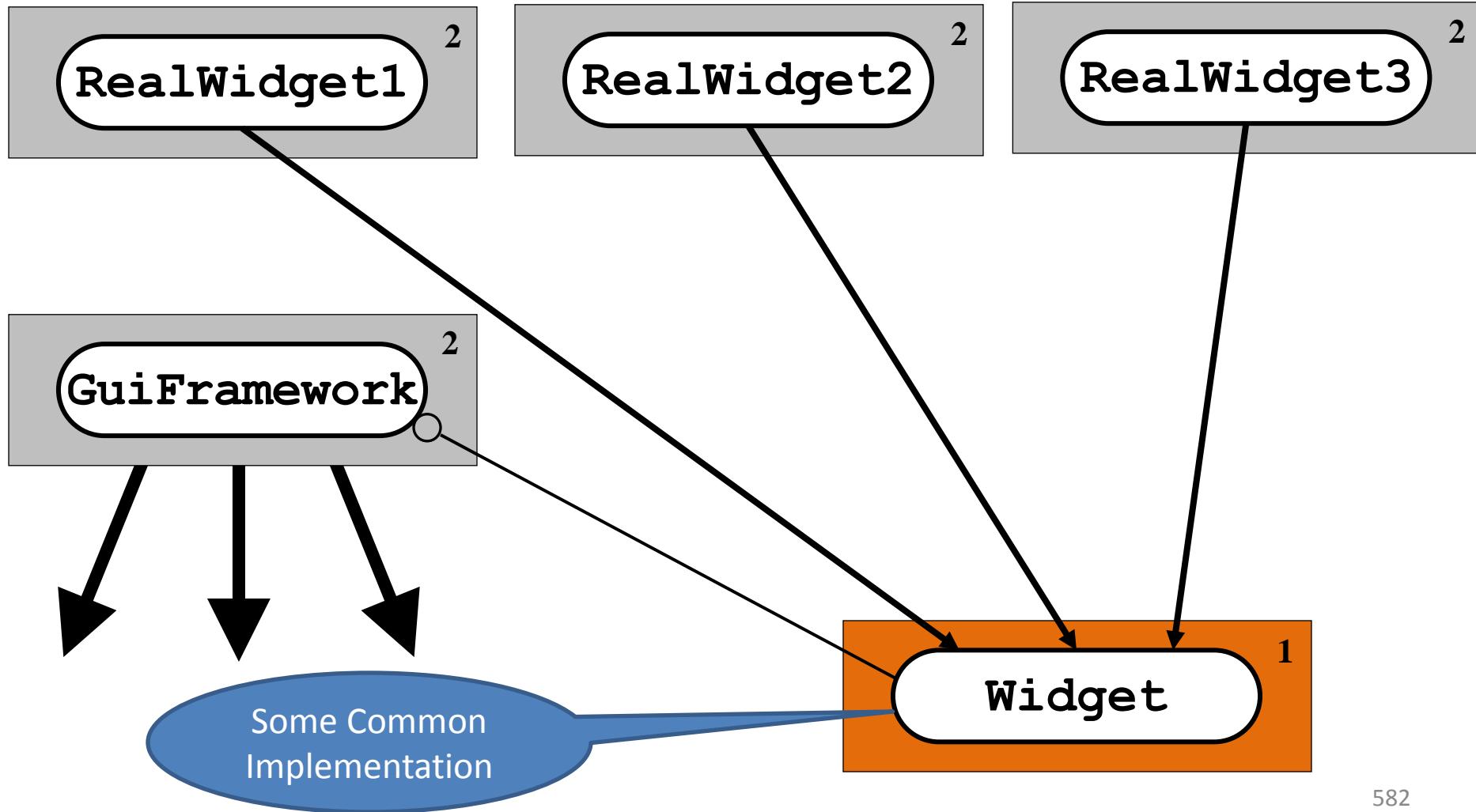
3. Total and Partial *Insulation* Techniques

Pure Abstract Interface (Protocol Class)



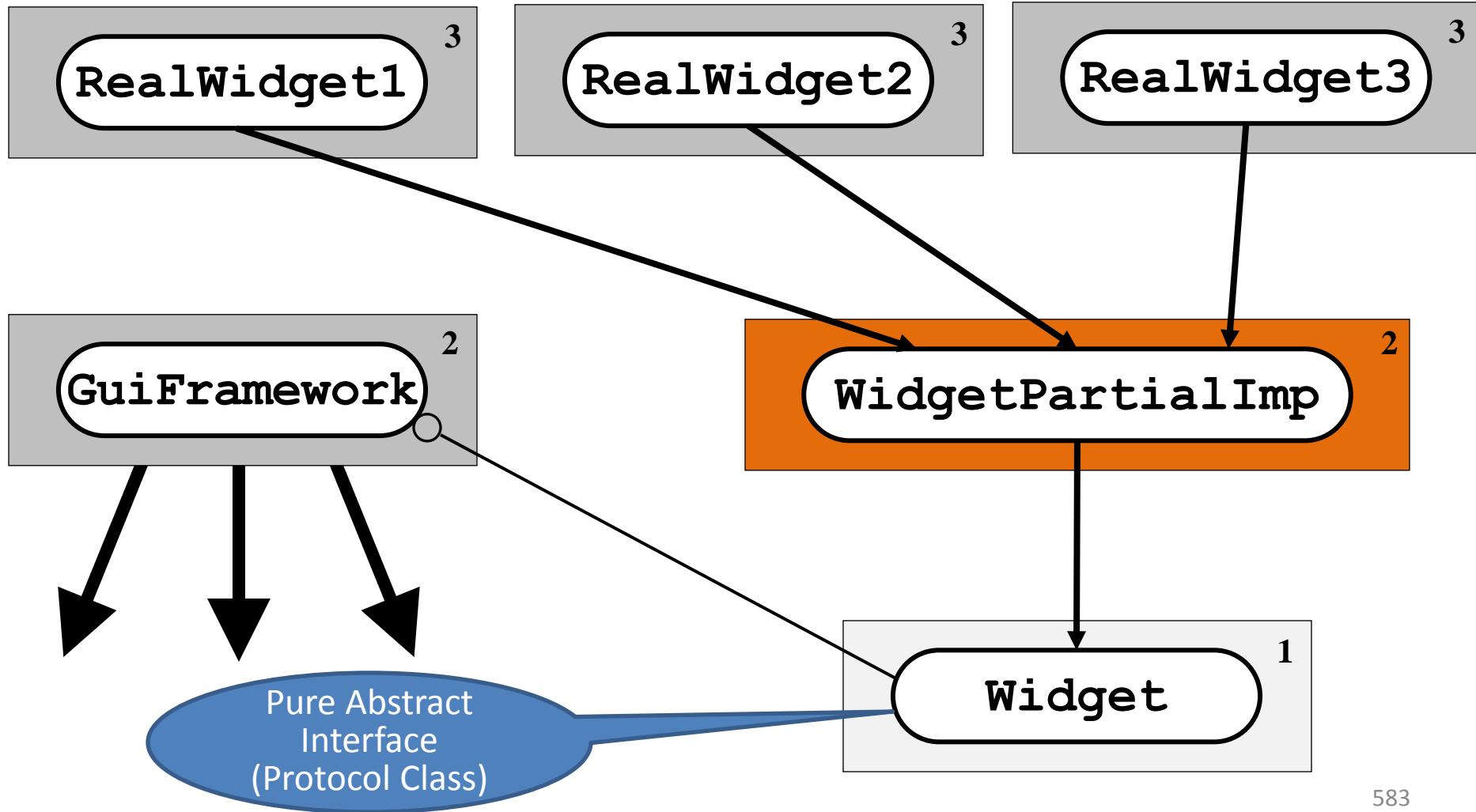
3. Total and Partial *Insulation* Techniques

Pure Abstract Interface (Protocol Class)



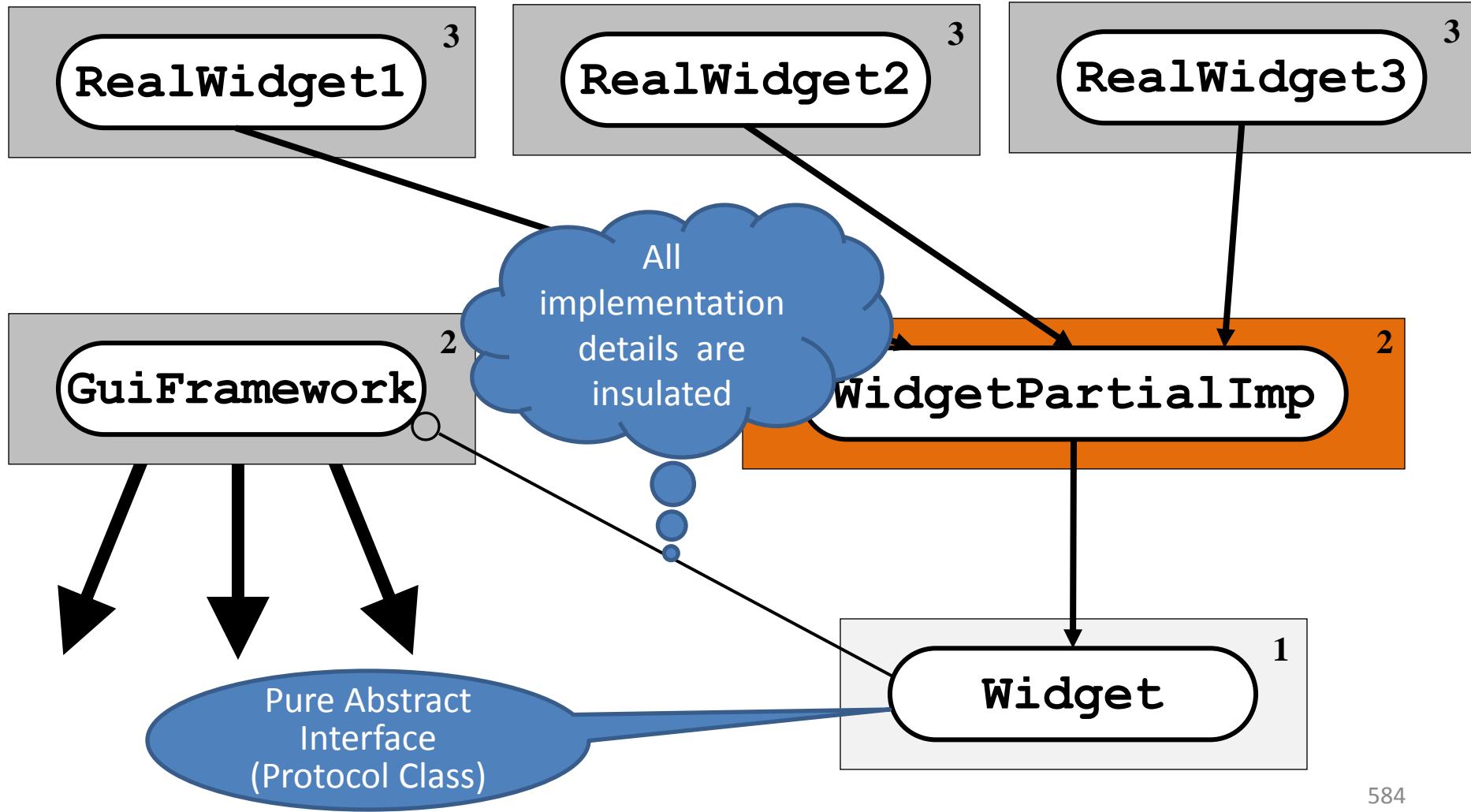
3. Total and Partial *Insulation* Techniques

Pure Abstract Interface (Protocol Class)



3. Total and Partial *Insulation* Techniques

Pure Abstract Interface (Protocol Class)



3. Total and Partial *Insulation* Techniques

Pure Abstract Interface (Protocol Class)

Discussion?

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

What do we mean by “Pimple”?

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

What do we mean by “Pimple”?

“Pointer to
implementation”

3. Total and Partial *Insulation* Techniques

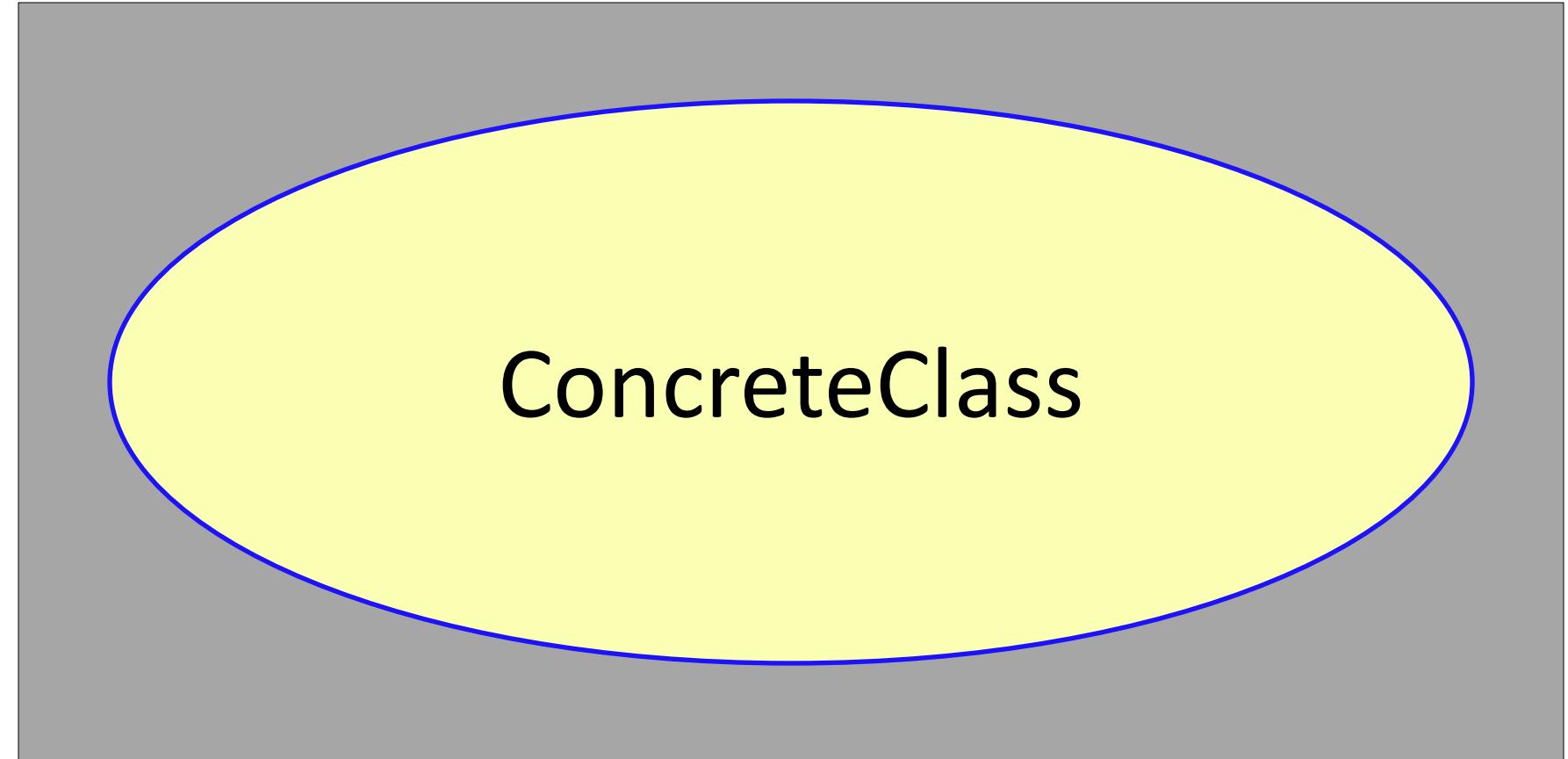
Fully Insulating Concrete Class (“Pimple”)

a.k.a. “Pimpl”

“Pointer to
implementation”

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

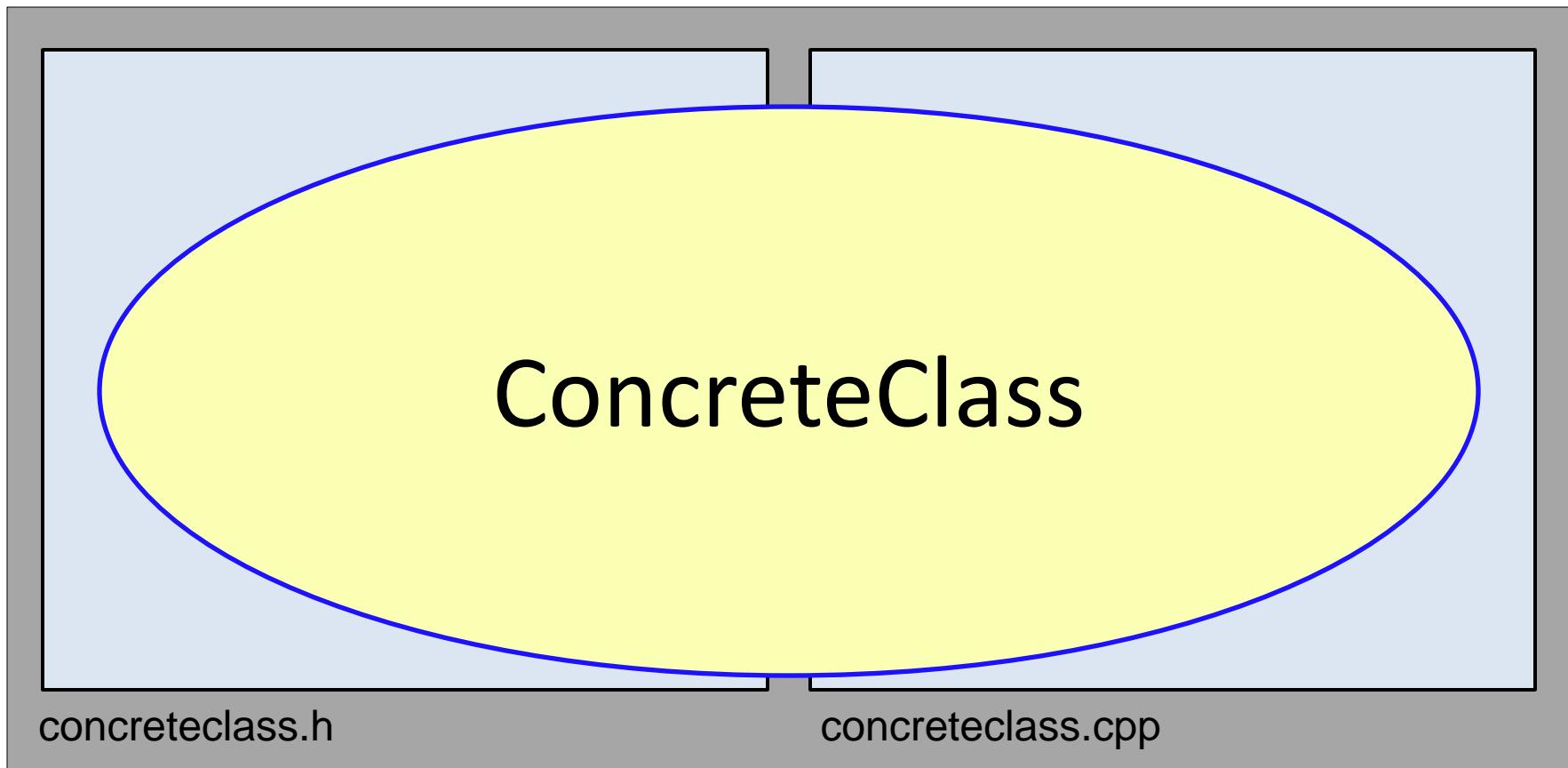


A diagram illustrating the "Pimple" technique for insulating a concrete class. It features a large yellow oval centered on a grey background. The oval is bounded by a thick blue line. Inside the oval, the text "ConcreteClass" is written in a large, black, sans-serif font.

ConcreteClass

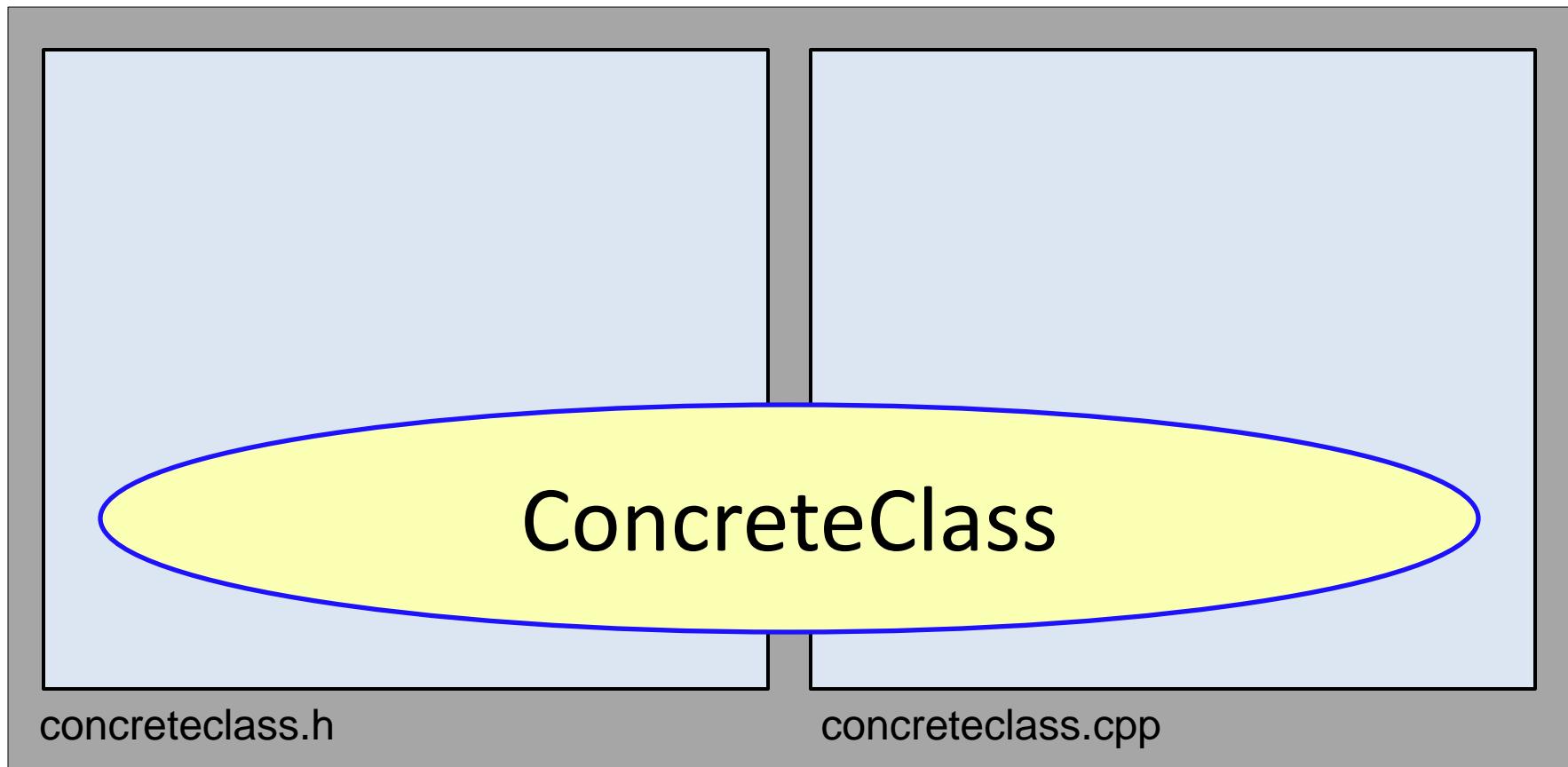
3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)



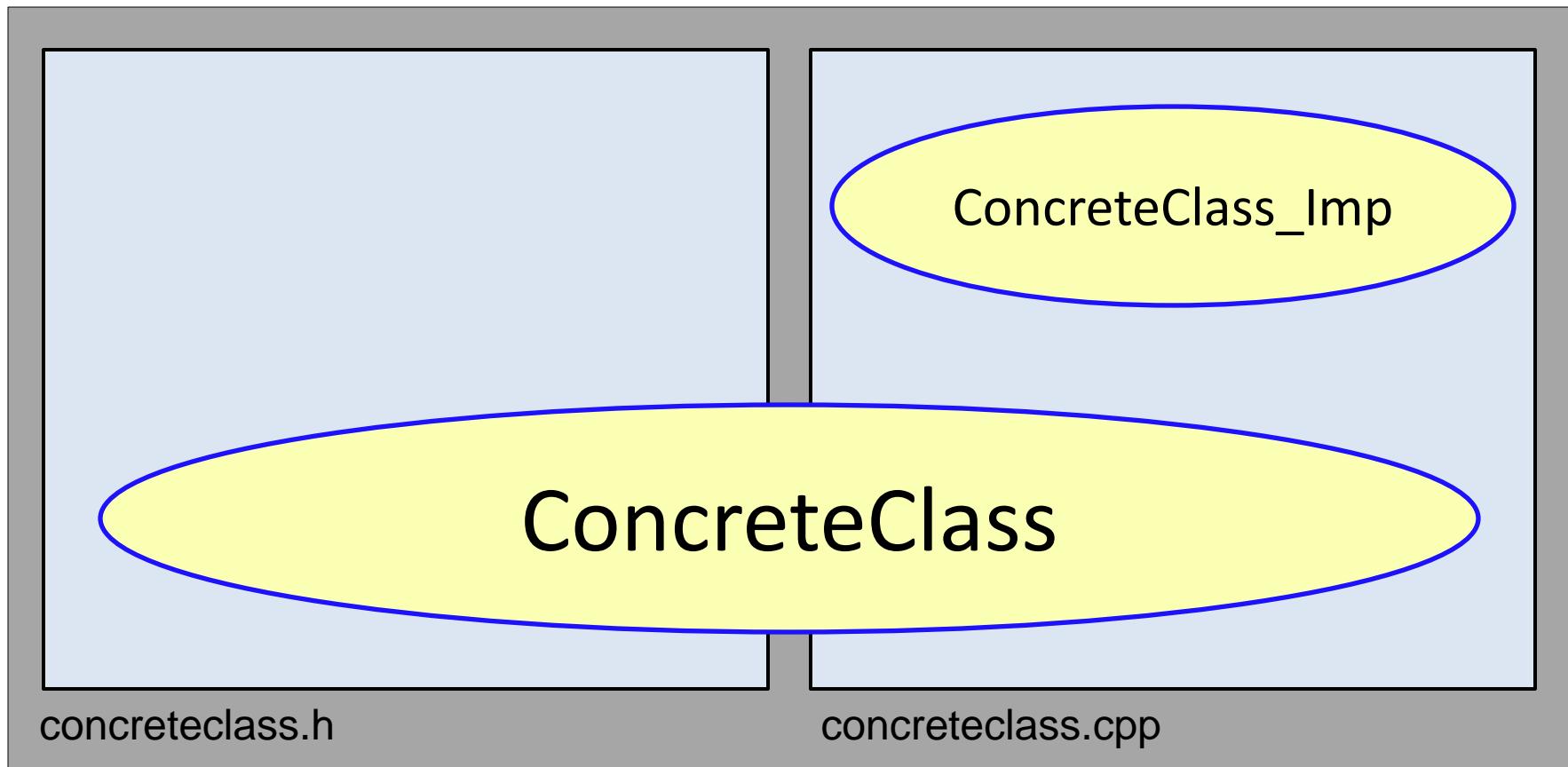
3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)



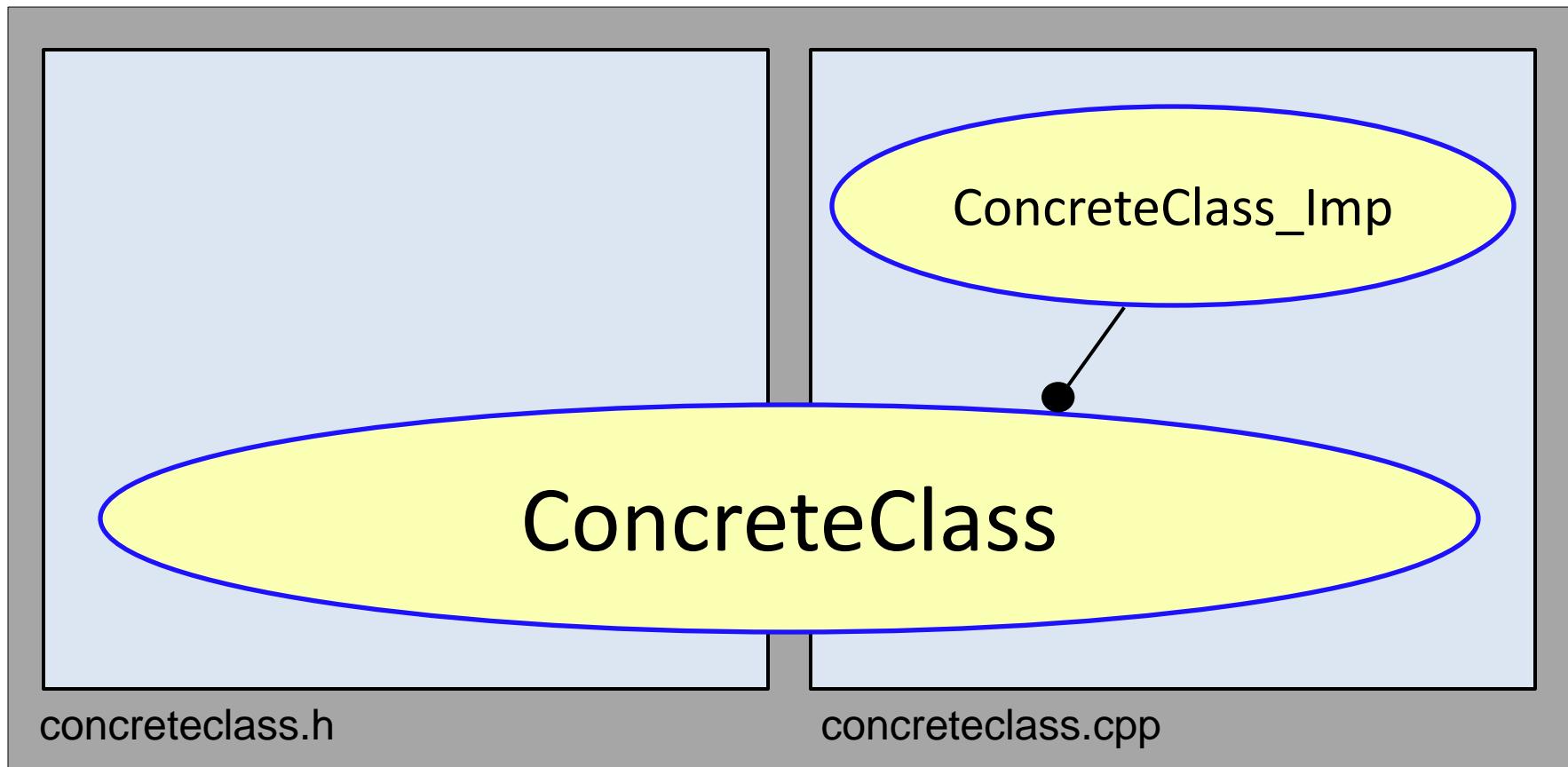
3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)



3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)



3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...

class MyClass {
    int d_data; // some data
    // ...
public:
    explicit MyClass(int data)
        : d_data(data) { }
    MyClass(const MyClass& orig)
        : d_data(orig.d_data) { }
    ~MyClass() { /* XYZZY */ }
    // ...
    int data() { return d_data; }
};

bool
operator==(const MyClass& a,
            const MyClass& b) {
    return a.d_data == b.d_data; }
// ...
```

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...

class MyClass {
    int d_data; // some data
    // ...
public:
    explicit MyClass(int data)
        : d_data(data) { }
    MyClass(const MyClass& orig)
        : d_data(orig.d_data) { }
    ~MyClass() { /* XYZZY */ }
    // ...
    int data() { return d_data; }
};

bool
operator==(const MyClass& a,
            const MyClass& b) {
    return a.d_data == b.d_data; }
// ...
```

} Member Data

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...

class MyClass {
    int d_data; // some data
    // ...
public:
    explicit MyClass(int data)
        : d_data(data) { }
    MyClass(const MyClass& orig)
        : d_data(orig.d_data) { }
    ~MyClass() { /* XYZZY */ }
    // ...
    int data() { return d_data; }
};

bool
operator==(const MyClass& a,
            const MyClass& b) {
    return a.d_data == b.d_data;
}
// ...
```

Creators

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...

class MyClass {
    int d_data; // some data
    // ...
public:
    explicit MyClass(int data)
        : d_data(data) { }
    MyClass(const MyClass& orig)
        : d_data(orig.d_data) { }
    ~MyClass() { /* XYZZY */ }
    // ...
    int data() { return d_data; }
};

bool
operator==(const MyClass& a,
            const MyClass& b) {
    return a.d_data == b.d_data; }
// ...
```

Creators

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...

class MyClass {
    int d_data; // some data
    // ...
public:
    explicit MyClass(int data)
        : d_data(data) { }
    MyClass(const MyClass& orig)
        : d_data(orig.d_data) { }
    ~MyClass() { /* XYZZY */ }
    // ...
    int data() { return d_data; }
};

bool
operator==(const MyClass& a,
            const MyClass& b) {
    return a.d_data == b.d_data;
}
// ...
```

Creators

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...

class MyClass {
    int d_data; // some data
    // ...
public:
    explicit MyClass(int data)
        : d_data(data) { }
    MyClass(const MyClass& orig)
        : d_data(orig.d_data) { }
    ~MyClass() { /* XYZZY */ }
    // ...
    int data() { return d_data; }
};

bool
operator==(const MyClass& a,
            const MyClass& b) {
    return a.d_data == b.d_data;
}
// ...
```

} Manipulators
& Accessors

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...

class MyClass {
    int d_data; // some data
    // ...
public:
    explicit MyClass(int data)
        : d_data(data) { }
    MyClass(const MyClass& orig)
        : d_data(orig.d_data) { }
    ~MyClass() { /* XY */ }
    // ...
    int data() { return d_data; }
};

bool
operator==(const MyClass& a,
            const MyClass& b) {
    return a.d_data == b.d_data;
}
// ...
```

Accessors should be
const-qualified.

} Manipulators
& Accessors

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...

class MyClass {
    int d_data; // some data
    // ...
public:
    explicit MyClass(int data)
        : d_data(data) { }
    MyClass(const MyClass& orig)
        : d_data(orig.d_data) { }
    ~MyClass() { /* XYZZY */ }
    // ...
    int data() { return d_data; }
};

bool
operator==(const MyClass& a,
            const MyClass& b) {
    return a.d_data == b.d_data;
}
// ...
```



Free Operators

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...
struct MyClass_Imp; // insulated
class MyClass {
    int d_data; // some data
    // ...
public:
    explicit MyClass(int data)
        : d_data(data) { }
    MyClass(const MyClass& orig)
        : d_data(orig.d_data) { }
    ~MyClass() { /* XYZZY */ }
    // ...
    int data() { return d_data; }
};

bool
operator==(const MyClass& a,
            const MyClass& b) {
    return a.d_data == b.d_data; }
// ...
```

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...
struct MyClass_Imp; // insulated
class MyClass {
    int d_data; // some data
    // ...
public:
    explicit MyClass(int data)
        : d_data(data) { }
    MyClass(const MyClass& orig)
        : d_data(orig.d_data) { }
    ~MyClass() { /* XYZZY */ }
    // ...
    int data() { return d_data; }
};

bool
operator==(const MyClass& a,
            const MyClass& b) {
    return a.d_data == b.d_data;
}
// ...
```

```
// myclass.cpp
#include <myclass.h>

struct MyClass_Imp {
    // ...
}
```

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...
struct MyClass_Imp; // insulated
class MyClass {
    int d_data; // some data
    // ...
public:
    explicit MyClass(int data)
        : d_data(data) { }
    MyClass(const MyClass& orig)
        : d_data(orig.d_data) { }
    ~MyClass() { /* XYZZY */ }
    // ...
}
```

```
// myclass.cpp
#include <myclass.h>

struct MyClass_Imp {
    // ...
}
```

Duplicate All
Member Data
and Creators!

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...
struct MyClass_Imp; // insulated
class MyClass {
    int d_data; // some data
    // ...
public:
    explicit MyClass(int data)
        : d_data(data) { }
    MyClass(const MyClass& orig)
        : d_data(orig.d_data) { }
    ~MyClass() { /* XYZZY */ }
    // ...
}
```

```
// myclass.cpp
#include <myclass.h>

struct MyClass_Imp {
    int d_data;
    // ...
    explicit MyClass_Imp(int data)
        : d_data(data) { }

    MyClass_Imp(const MyClass_Imp& orig)
        : d_data(orig.d_data) { }

    ~MyClass_Imp() { /* XYZZY */ }
};
```

Duplicate All
Member Data
and Creators!

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...
struct MyClass_Imp; // insulated
class MyClass {
    int d_data; // some data
    // ...
public:
    explicit MyClass(int data)
        : d_data(data) { }
    MyClass(const MyClass& orig)
        : d_data(orig.d_data) { }
    ~MyClass() { /* XYZZY */ }
    // ...
    int data() { return d_data; }
};

bool
operator==(const MyClass& a,
            const MyClass& b) {
    return a.d_data == b.d_data;
}
// ...
```

```
// myclass.cpp
#include <myclass.h>

struct MyClass_Imp {
    int d_data;
    // ...
    explicit MyClass_Imp(int data)
        : d_data(data) { }

    MyClass_Imp(const MyClass_Imp& orig)
        : d_data(orig.d_data) { }

    ~MyClass_Imp() { /* XYZZY */ }
};
```

Replace Data
with SINGLE
Pointer to Imp.

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...
struct MyClass_Imp; // insulated
class MyClass {
    MyClass_Imp *d_imp_p;
public:
    explicit MyClass(int d_data)
        : d_data(d_data) { }
    MyClass(const MyClass& orig)
        : d_data(orig.d_data) { }
    ~MyClass() { /* XYZZY */ }
    // ...
    int data() { return d_data; }
};
bool
operator==(const MyClass& a,
            const MyClass& b) {
    return a.d_data == b.d_data;
}
// ...
```

```
// myclass.cpp
#include <myclass.h>

struct MyClass_Imp {
    int d_data;
    // ...
    explicit MyClass_Imp(int data)
        : d_data(data) { }

    MyClass_Imp(const MyClass_Imp& orig)
        : d_data(orig.d_data) { }

    ~MyClass_Imp() { /* XYZZY */ }
};
```

Replace Data
with SINGLE
Pointer to Imp.

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...
struct MyClass_Imp; // insulated
class MyClass {
    MyClass_Imp *d_imp_p;

public:
    explicit MyClass(int data)
        : d_data(data) { }
    MyClass(const MyClass& orig)
        : d_data(orig.d_data) { }
    ~MyClass() { /* XYZZY */ }
    // ...
    int data() { return d_data; }
};

bool
operator==(const MyClass& a,
            const MyClass& b) {
    return a.d_data == b.d_data;
}
// ...
```

Eliminate
All inline
functions!

```
// myclass.h
#ifndef MYCLASS_H
#define MYCLASS_H
#include <iostream>
#include <memory>

class MyClass {
public:
    MyClass(const MyClass& orig)
        : d_data(orig.d_data) { }

    ~MyClass() { /* XYZZY */ }

private:
    MyClass_Imp* d_imp_p;
};

MyClass_Imp::MyClass_Imp(const MyClass_Imp& orig)
    : d_data(orig.d_data) { }

~MyClass_Imp() { /* XYZZY */ }

MyClass_Imp::operator=(const MyClass_Imp& orig)
{
    d_data = orig.d_data;
    return *this;
}

MyClass_Imp::MyClass_Imp()
    : d_data(0) { }

MyClass_Imp::operator int() const
{
    return d_data;
}
```

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...
struct MyClass_Imp; // insulated
class MyClass {
    MyClass_Imp *d_imp_p;

public:
    explicit MyClass(int data) ;

    MyClass(const MyClass& orig);

    ~MyClass() ;
    // ...
    int data() ;

};

bool
operator==(const MyClass& a,
            const MyClass& b) ;

// ...
```

Eliminate All inline functions!

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...
struct MyClass_Imp; // insulated
class MyClass {
    MyClass_Imp *d_imp_p;

public:
    explicit MyClass(int data);

    MyClass(const MyClass& orig);

    ~MyClass();
    // ...
    int data();
};

bool
operator==(const MyClass& a,
            const MyClass& b);

// ...
```

```
// myclass.cpp
#include <myclass.h>

struct MyClass_Imp {
    int d_data;
    // ...
    explicit MyClass_Imp(int data)
        : d_data(data) { }

    MyClass_Imp(const MyClass_Imp& orig)
        : d_data(orig.d_data) { }

    ~MyClass_Imp() { /* XYZZY */ }
};
```

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...
struct MyClass_Imp; // insulated
class MyClass {
    MyClass_Imp *d_imp_p;

public:
    explicit MyClass(int data);

    MyClass(const MyClass& orig);

    ~MyClass();
    // ...
    int data();
};

bool
operator==(const MyClass& a,
            const MyClass& b);

// ...
```

```
// myclass.cpp
#include <myclass.h>

struct MyClass_Imp {
    int d_data;
    // ...
    explicit MyClass_Imp(int data)
        : d_data(data) { }

    MyClass_Imp(const MyClass_Imp& orig)
        : d_data(orig.d_data) { }

    ~MyClass_Imp() { /* XYZZY */ }
};
```

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...
struct MyClass_Imp; // insulated
class MyClass {
    MyClass_Imp *d_imp_p;

public:
    explicit MyClass(int data);

    MyClass(const MyClass& orig);

    ~MyClass();
    // ...
    int data() const;
};

bool
operator==(const MyClass& a,
            const MyClass& b);

// ...
```

```
// myclass.cpp
#include <myclass.h>

struct MyClass_Imp {
    int d_data;
    // ...
    explicit MyClass_Imp(int data)
        : d_data(data) { }

    MyClass_Imp(const MyClass_Imp& orig)
        : d_data(orig.d_data) { }

    ~MyClass_Imp() { /* XYZZY */ }
};
```

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...
struct MyClass_Imp; // insulated
class MyClass {
    MyClass_Imp *d_imp_p;

public:
    explicit MyClass(int data);

    MyClass(const MyClass& orig);

    ~MyClass();
    // ...
    int data() const;
};

bool
operator==(const MyClass& a,
            const MyClass& b);

// ...
```

```
// myclass.cpp
#include <myclass.h>

struct MyClass_Imp {
    int d_data;
    // ...
    explicit MyClass_Imp(int data)
        : d_data(data) { }

    MyClass_Imp(const MyClass_Imp& orig)
        : d_data(orig.d_data) { }

    ~MyClass_Imp() { /* XYZZY */ }
};
```



Now
There's
Room!

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...
struct MyClass_Imp; // insulated
class MyClass {
    MyClass_Imp *d_imp_p;

public:
explicit MyClass(int data);

MyClass(const MyClass& orig);

~MyClass();
// ...
int data() const;
};

bool
operator==(const MyClass& a,
            const MyClass& b);

// ...
```

```
// myclass.cpp
#include <myclass.h>

struct MyClass_Imp {
    int d_data;
    // ...
explicit MyClass_Imp(int data)
: d_data(data) {}

MyClass_Imp(const MyClass_Imp& orig)
: d_data(orig.d_data) {}

~MyClass_Imp() { /* XYZZY */ }

MyClass::MyClass(int data)
: d_imp_p(new MyClass_Imp(data))
{}
```

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...
struct MyClass_Imp; // insulated
class MyClass {
    MyClass_Imp *d_imp_p;

public:
explicit MyClass(int data);

MyClass(const MyClass& orig);

~MyClass();
// ...
int data() const;
};

bool
operator==(const MyClass& a,
            const MyClass& b);

// ...
```

```
// myclass.cpp
#include <myclass.h>

struct MyClass_Imp {
    int d_data;
    // ...
explicit MyClass_Imp(int data)
: d_data(data) {}

MyClass_Imp(const MyClass_Imp& orig)
: d_data(orig.d_data) {}

~MyClass_Imp() { /* XYZZY */ }

MyClass::MyClass(int data)
: d_imp_p(new MyClass_Imp(data))
{}
```

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...
struct MyClass {
    int d_data;
    // ...
    MyClass(int data)
        : d_data(data) { }

    ~MyClass() { /* XYZZY */ }

    class MyClass(int data)
        : d_imp_p(new MyClass_Imp(data)) {
    }
};

// ...
```

(In Practice,
We Would
Now Also
Pass an
Optional
Allocator at
construction.)

```
// myclass.cpp
#include <myclass.h>

struct MyClass_Imp {
    int d_data;
    // ...
    explicit MyClass_Imp(int data)
        : d_data(data) { }

    MyClass_Imp(const MyClass_Imp& orig)
        : d_data(orig.d_data) { }

    ~MyClass_Imp() { /* XYZZY */ }

    class MyClass(int data)
        : d_imp_p(new MyClass_Imp(data)) {
    }
};
```

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...
struct MyClass_Imp; // insulated
class MyClass {
    MyClass_Imp *d_imp_p;

public:
    explicit MyClass(int data,
                      bslma::Allocator *a = 0);
    MyClass(const MyClass& orig,
            bslma::Allocator *a = 0);
    ~MyClass();
    // ...
    int data() const;
};

bool
operator==(const MyClass& a,
            const MyClass& b);

// ...
```

```
// myclass.cpp
#include <myclass.h>
#include <bslma_allocator.h>
struct MyClass_Imp {
    int d_data;
    // ...
    bslma::Allocator *d_allocator_p;

    MyClass_Imp(int data,
                bslma::Allocator *a)
        : d_data(data)
        , d_allocator_p(a) { }
    // ...
};

MyClass::MyClass(int data,
                 bslma::Allocator *a)
: d_imp_p(new(*a) MyClass_Imp(data, a))
{}
```

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...
struct MyClass {
    int d_data;
    // ...
    void f(int a, int b);
    // ...
};
```

(We Will
Ignore
Allocators
for Now.)

```
// myclass.cpp
#include <myclass.h>
#include <bslma_allocator.h>
struct MyClass_Imp {
    int d_data;
    // ...
    bslma::Allocator *d_allocator_p;
};

MyClass_Imp(int data,
            bslma::Allocator *a)
: d_data(data)
, d_allocator_p(a) { }
// ...

class MyClass(int data,
              bslma::Allocator *a)
: d_imp_p(new(*a) MyClass_Imp(data, a))
{ }
```

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...
struct MyClass_Imp; // insulated
class MyClass {
    MyClass_Imp *d_imp_p;

public:
explicit MyClass(int data);

MyClass(const MyClass& orig);

~MyClass();
// ...
int data() const;
};

bool
operator==(const MyClass& a,
            const MyClass& b);

// ...
```

```
// myclass.cpp
#include <myclass.h>

struct MyClass_Imp {
    int d_data;
    // ...
explicit MyClass_Imp(int data)
: d_data(data) {}

MyClass_Imp(const MyClass_Imp& orig)
: d_data(orig.d_data) {}

~MyClass_Imp() { /* XYZZY */ }

MyClass::MyClass(int data)
: d_imp_p(new MyClass_Imp(data))
{}
```

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...
struct MyClass_Imp; // insulated
class MyClass {
    MyClass_Imp *d_imp_p;

public:
    explicit MyClass(int data);

MyClass(const MyClass& orig);

    ~MyClass();
    // ...
    int data() const;
};

bool
operator==(const MyClass& a,
            const MyClass& b);

// ...
```

```
// myclass.cpp
#include <myclass.h>

struct MyClass_Imp {
    int d_data;
    // ...
    explicit MyClass_Imp(int data)
        : d_data(data) { }

    MyClass_Imp(const MyClass_Imp& orig)
        : d_data(orig.d_data) { }

    ~MyClass_Imp() { /* XYZZY */ }
};

MyClass::MyClass(const MyClass& orig)
: d_imp_p(
    newMyClass_Imp(*orig.d_imp_p))
{}
```

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...
struct MyClass_Imp; // insulated
class MyClass {
    MyClass_Imp *d_imp_p;

public:
    explicit MyClass(int data);

    MyClass(const MyClass& orig);

    ~MyClass();
    // ...
    int data() const;
};

bool
operator==(const MyClass& a,
            const MyClass& b);

// ...
```

```
// myclass.cpp
#include <myclass.h>

struct MyClass_Imp {
    int d_data;
    // ...
    explicit MyClass_Imp(int data)
        : d_data(data) { }

    MyClass_Imp(const MyClass_Imp& orig)
        : d_data(orig.d_data) { }

    ~MyClass_Imp() { /* XYZZY */ }
};

MyClass::~MyClass()
{
    delete d_imp_p;
}
```

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...
struct MyClass_Imp; // insulated
class MyClass {
    MyClass_Imp *d_imp_p;

public:
    explicit MyClass(int data);

    MyClass(const MyClass& orig);

    ~MyClass();
    // ...
    int data() const;
};

bool
operator==(const MyClass& a,
            const MyClass& b);

// ...
```

```
// myclass.cpp
#include <myclass.h>

struct MyClass_Imp {
    int d_data;
    // ...
    explicit MyClass_Imp(int data)
        : d_data(data) { }

    MyClass_Imp(const MyClass_Imp& orig)
        : d_data(orig.d_data) { }

    ~MyClass_Imp() { /* XYZZY */ }
};

int MyClass::data() const
{
    return d_imp_p->d_data;
}
```

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...
struct MyClass_Imp; // insulated
class MyClass {
    MyClass_Imp *d_imp_p;

public:
    explicit MyClass(int data);

    MyClass(const MyClass& orig);

    ~MyClass();
    // ...
    int data() const;
};

bool
operator==(const MyClass& a,
            const MyClass& b);

// ...
```

```
// myclass.cpp
#include <myclass.h>

struct MyClass_Imp {
    int d_data;
    // ...
    explicit MyClass_Imp(int data)
        : d_data(data) { }

    MyClass_Imp(const MyClass_Imp& orig)
        : d_data(orig.d_data) { }

    ~MyClass_Imp() { /* XYZZY */ }
};

int MyClass::data() const
{
    return d_imp_p->d_data;
}
```

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...
public:
    explicit MyClass_<int>(int data);

    MyClass(const MyClass& orig);

    ~MyClass();
    // ...
    int data() const;
};

bool
operator==(const MyClass& a,
            const MyClass& b);

// ...
```

Just access
the data
directly.

```
// myclass.cpp
#include <myclass.h>

struct MyClass_Imp {
    int d_data;
    // ...
    explicit MyClass_Imp(int data)
        : d_data(data) { }

    MyClass_Imp(const MyClass_Imp& orig)
        : d_data(orig.d_data) { }

    ~MyClass_Imp() { /* XYZZY */ }

    int MyClass::data() const
    {
        return d_imp_p->d_data;
    }
};
```

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...
struct MyClass_Imp; // insulated
class MyClass {
    MyClass_Imp *d_imp_p;

public:
    explicit MyClass(int data);

    MyClass(const MyClass& orig);

    ~MyClass();
    // ...
    int data() const;
};

bool
operator==(const MyClass& a,
            const MyClass& b);

// ...
```

```
// myclass.cpp
#include <myclass.h>

struct MyClass_Imp {
    int d_data;
    // ...
    explicit MyClass_Imp(int data)
        : d_data(data) { }

    MyClass_Imp(const MyClass_Imp& orig)
        : d_data(orig.d_data) { }

    ~MyClass_Imp() { /* XYZZY */ }

    bool
    MyClass::operator==(MyClass& a,
                        MyClass& b)
    {
        return a.d_imp_p->d_data
               == b.d_imp_p->d_data;
    }
}
```

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
// myclass.h
// ...
struct MyClass_Imp; // insulated
class MyClass {
    MyClass_Imp *d_imp_p;

public:
    explicit MyClass(int data);

    MyClass(const MyClass& orig);

    ~MyClass();
    // ...
    int data() const;
};

bool
operator==(const MyClass& a,
            const MyClass& b);

// ...
```

One Extra Indirection!

```
// myclass.cpp
#include "myclass.h"

MyClass::MyClass(int data)
    : d_data(data) { }

MyClass::MyClass(const MyClass& orig)
    : d_data(orig.d_data) { }

MyClass::~MyClass() /* XYZZY */ { }

bool
MyClass::operator==(MyClass& a,
                     MYClass& b)
{
    return a.d_imp_p->d_data
        == b.d_imp_p->d_data;
}
```

3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)



3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

```
/* myclass.h
```

```
// myc
```

One Extra
And Not
**Under No circumstances
Expose the Pointer
to the Implementation!**
I can't!

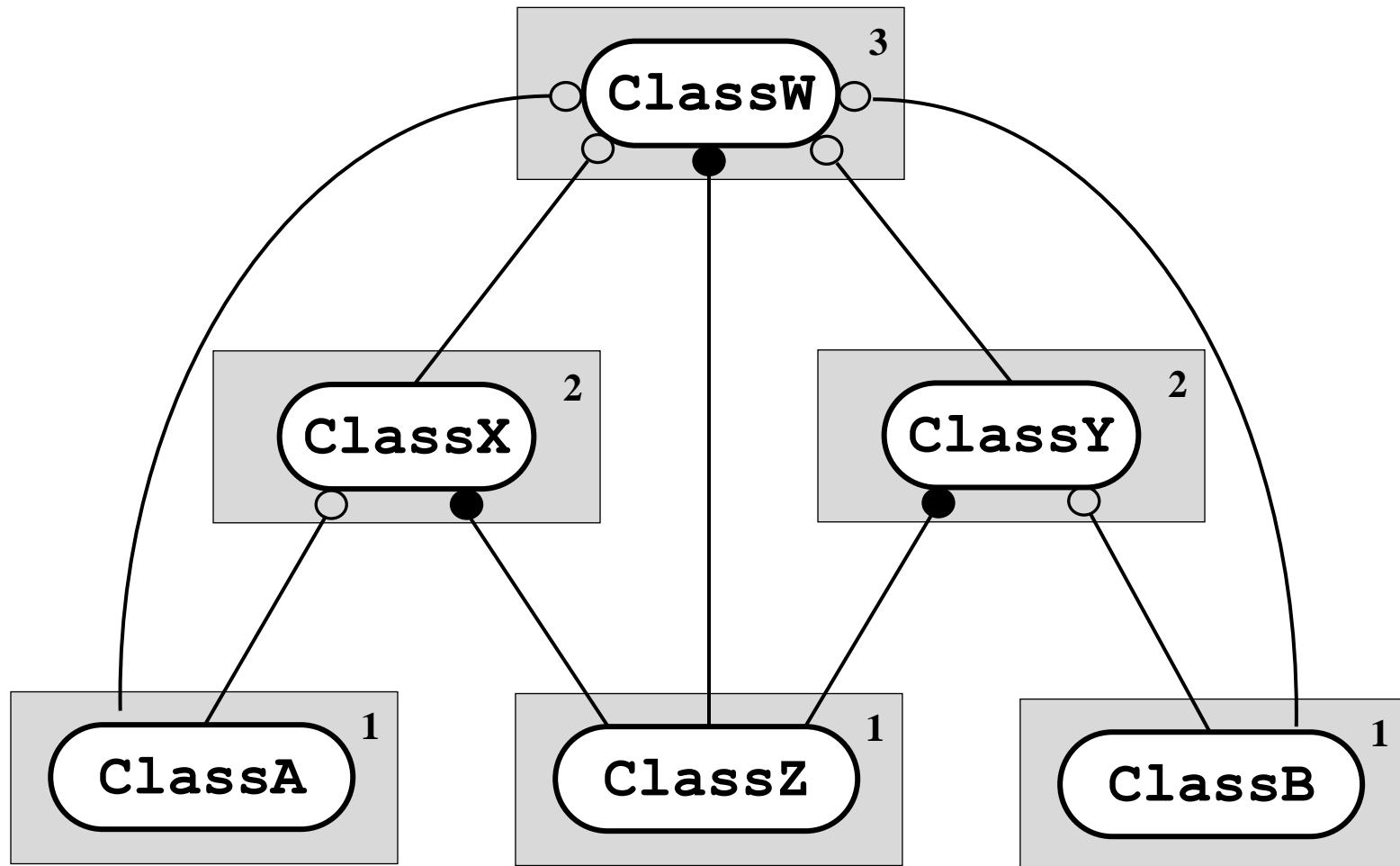
3. Total and Partial *Insulation* Techniques

Fully Insulating Concrete Class (“Pimple”)

Discussion?

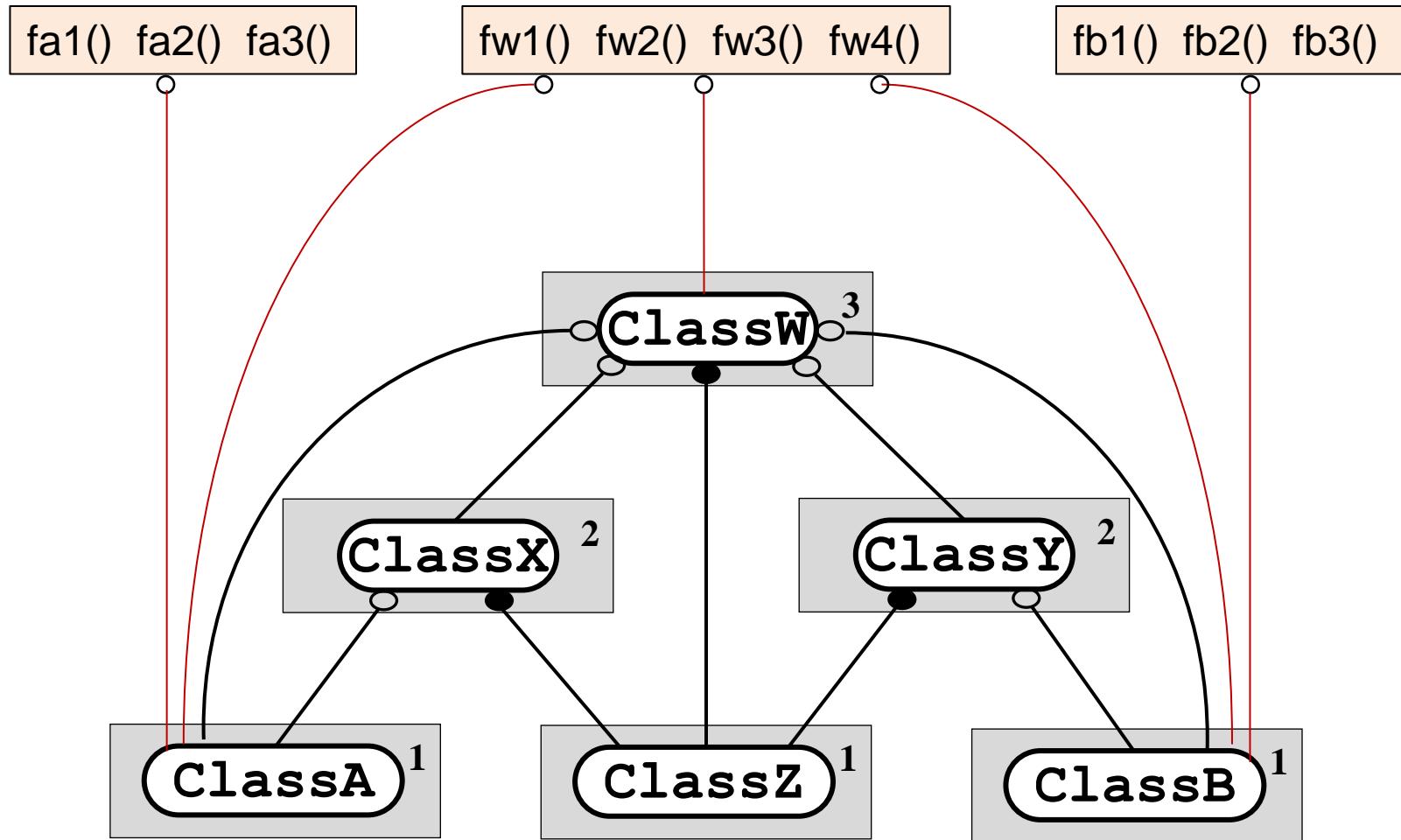
3. Total and Partial *Insulation* Techniques

Procedural Interface



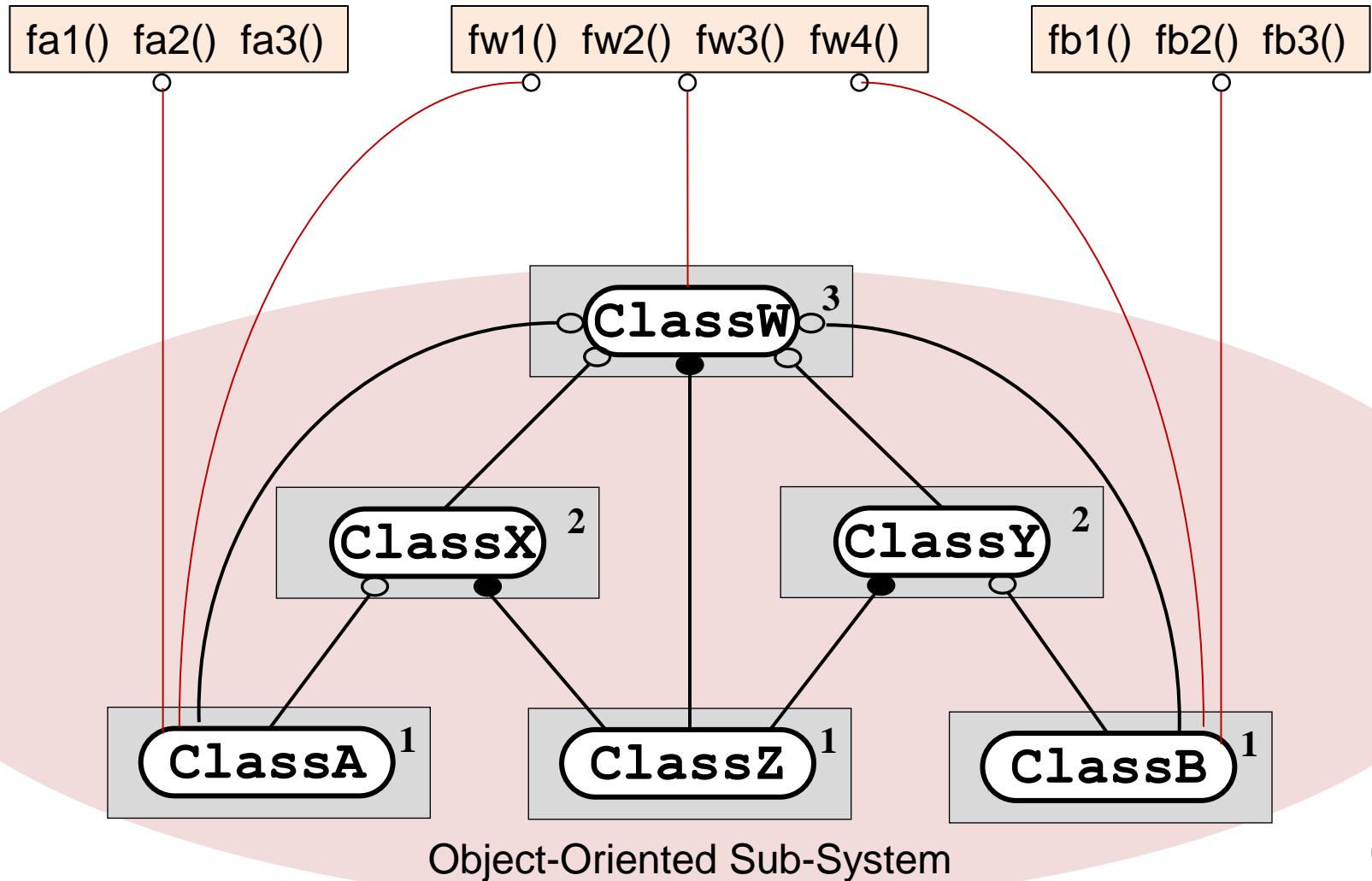
3. Total and Partial *Insulation* Techniques

Procedural Interface



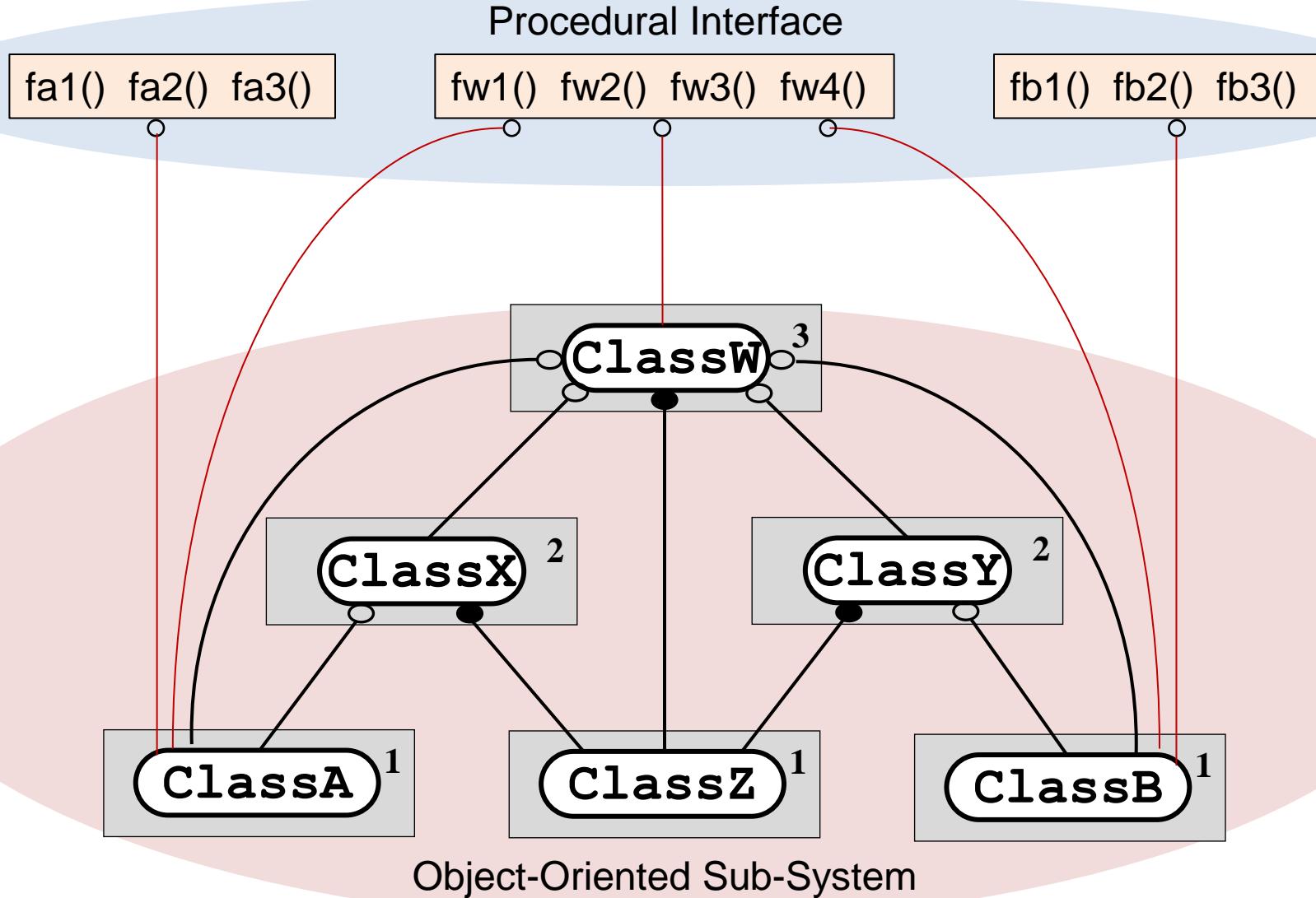
3. Total and Partial *Insulation* Techniques

Procedural Interface



3. Total and Partial *Insulation* Techniques

Procedural Interface



3. Total and Partial *Insulation* Techniques

Procedural Interface

Every procedural interface (PI) function is:

3. Total and Partial *Insulation* Techniques

Procedural Interface

Every procedural interface (PI) function is:

1. Entirely independent of every other PI function.

3. Total and Partial *Insulation* Techniques

Procedural Interface

Every procedural interface (PI) function is:

1. Entirely independent of every other PI function.
2. Physically separate from any C++ components.

3. Total and Partial *Insulation* Techniques

Procedural Interface

Every procedural interface (PI) function is:

1. Entirely independent of every other PI function.
2. Physically separate from any C++ components.
3. Totally devoid of additional implementation
(beyond that needed for insulation purposes).

3. Total and Partial *Insulation* Techniques

Procedural Interface

Every procedural interface (PI) function is:

1. Entirely independent of every other PI function.
2. Physically separate from any C++ components.
3. Totally devoid of additional implementation
(beyond that needed for insulation purposes).
4. Named in a natural, regular, and predictable way.

3. Total and Partial *Insulation* Techniques

Procedural Interface

Every procedural interface (PI) function is:

1. Entirely independent of every other PI function.
2. Physically separate from any C++ components.
3. Totally devoid of additional implementation
(beyond that needed for insulation purposes).
4. Named in a natural, regular, and predictable way.
5. Callable from both C and C++.

3. Total and Partial *Insulation* Techniques

Procedural Interface

Every procedural interface (PI) function is:

1. Entirely independent of every other PI function.
2. Physically separate from any C++ components.
3. Totally devoid of additional implementation
(beyond that needed for insulation purposes).
4. Named in a natural, regular, and predictable way.
5. Callable from both C and C++.
6. Exposing the actual underlying opaque C++ types.

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// bdeg_point.h
// ...
namespace bdeg {
class Point {
    int d_x;
    int d_y;
public:
    // CREATORS
    Point();
    Point(int x, int y);
    Point(const Point& other);
    ~Point()
    // MANIPULATORS
    Point& operator=(const Point& rhs);
    void setX(int x);
    void setY(int y);
    // ACCESSORS
    int x() const;
    int y() const;
};
// FREE OPERATORS
bool operator==(const Point& lhs,
                  const Point& rhs);
// ... (e.g., output operator)
```

```
// bdeg_point.h (continued)
inline
Point::Point()
: d_x(0), d_y(0) { }

inline
Point::Point(int x, int y)
: d_x(x), d_y(y) { }

inline
Point::Point(const Point& other)
: d_x(other.d_x)
, d_y(other.d_y) { }

inline
Point::~Point() { }

inline
Point& Point::operator=(const Point&
rhs) {
    d_x = rhs.d_x; d_y = rhs.d_y;
    return *this;
}
// ...
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// bdeg_point.h
// ...
namespace bdeg {
class Point {
    int d_x;
    int d_y;
public:
    // CREATORS
    Point();
    Point(int x, int y);
    Point(const Point& other);
    ~Point()
    // MANIPULATORS
    Point& operator=(const Point& rhs);
    void setX(int x);
    void setY(int y);
    // ACCESSORS
    int x() const;
    int y() const;
};
// FREE OPERATORS
bool operator==(const Point& lhs,
                  const Point& rhs);
// ... (e.g., output operator)
```

Note that Internal
Include Guards and
Enterprise-Wide
Namespace
are Elided.

```
// bdeg_point.h (continued)
inline
Point::Point()
: d_x(0), d_y(0) { }

inline
Point::Point(int x, int y)
: d_x(x), d_y(y) { }

inline
Point::Point(const Point& other)
: d_x(other.d_x)
, d_y(other.d_y) { }

inline
Point::~Point() { }

inline
Point& Point::operator=(const Point&
rhs) {
    d_x = rhs.d_x; d_y = rhs.d_y;
    return *this;
}
// ...
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// bdeg_point.h
// ...
namespace bdeg {
class Point {
    int d_x;
    int d_y;
public:
    // CREATORS
    Point();
    Point(int x, int y);
    Point(const Point& other);
    ~Point()
    // MANIPULATORS
    Point& operator=(const Point& rhs);
    void setX(int x);
    void setY(int y);
    // ACCESSORS
    int x() const;
    int y() const;
};
// FREE OPERATORS
bool operator==(const Point& lhs,
                  const Point& rhs);
// ... (e.g., output operator)
```

```
// bdeg_point.cpp
#include <bdeg_point.h>
#include <iostream>

namespace MyLongCompanyName {

namespace bdeg { // empty
} // package namespace

std::ostream&
bdeg::operator<<(std::ostream&
stream, const Point& point)
{
    return stream << '(' <<
        point.x() << ',' <<
        point.y() << ')' <<
        std::flush;
}

// ...
} // enterprise namespace
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// bdeg_point.h
// ...
namespace bdeg {
class Point {
    int d_x;
    int d_y;
public:
    // CREATORS
    Point();
    Point(int x, int y);
    Point(const Point& other);
    ~Point()
    // MANIPULATORS
    Point& operator=(const Point& rhs);
    void setX(int x);
    void setY(int y);
    // ACCESSORS
    int x() const;
    int y() const;
};
// FREE OPERATORS
bool operator==(const Point& lhs,
                  const Point& rhs);
// ... (e.g., output operator)
```

```
// bdeg_point.cpp <- Always Have .cpp
#include <bdeg_point.h>
#include <iostream>

namespace MyLongCompanyName {

namespace bdeg { // empty
} // package namespace

std::ostream&
bdeg::operator<<(std::ostream&
stream, const Point& point)
{
    return stream << '(' <<
        point.x() << ',' <<
        point.y() << ')' <<
        std::flush;
}

// ...
} // enterprise namespace
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// bdeg_point.h
// ...
namespace bdeg {
class Point {
    int d_x;
    int d_y;
public:
    // CREATORS
    Point();
    Point(int x, int y);
    Point(const Point& other);
    ~Point()
    // MANIPULATORS
    Point& operator=(const Point& rhs);
    void setX(int x);
    void setY(int y);
    // ACCESSORS
    int x() const;
    int y() const;
};
// FREE OPERATORS
bool operator==(const Point& lhs,
                  const Point& rhs);
// ... (e.g., output operator)
```

```
// bdeg_point.cpp
#include <bdeg_point.h> <- Always First
#include <iostream>

namespace MyLongCompanyName {

namespace bdeg { // empty
} // package namespace

std::ostream&
bdeg::operator<<(std::ostream&
stream, const Point& point)
{
    return stream << '(' <<
        point.x() << ',' <<
        point.y() << ')' <<
        std::flush;
}

// ...
} // enterprise namespace
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// bdeg_point.h
// ...
namespace bdeg {
class Point {
    int d_x;
    int d_y;
public:
    // CREATORS
    Point();
    Point(int x, int y);
    Point(const Point& other);
    ~Point()
    // MANIPULATORS
    Point& operator=(const Point& rhs);
    void setX(int x);
    void setY(int y);
    // ACCESSORS
    int x() const;
    int y() const;
};
// FREE OPERATORS
bool operator==(const Point& lhs,
                  const Point& rhs);
// ... (e.g., output operator)
```

```
// bdeg_point.cpp
#include <bdeg_point.h>
#include <iostream>
All our code resides in the Enterprise Namespace.
namespace MyLongCompanyName {

namespace bdeg { // empty
} // package namespace

std::ostream&
bdeg::operator<<(std::ostream&
stream, const Point& point)
{
    return stream << '(' <<
        point.x() << ',' <<
        point.y() << ')' <<
        std::flush;
}

// ...
} // enterprise namespace
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// bdeg_point.h
// ...
namespace bdeg {
class Point {
    int d_x;
    int d_y;
public:
    // CREATORS
    Point();
    Point(int x, int y);
    Point(const Point& other);
    ~Point()
    // MANIPULATORS
    Point& operator=(const Point& rhs);
    void setX(int x);
    void setY(int y);
    // ACCESSORS
    int x() const;
    int y() const;
};
// FREE OPERATORS
bool operator==(const Point& lhs,
                  const Point& rhs);
// ... (e.g., output operator)
```

```
// bdeg_point.cpp
#include <bdeg_point.h>
#include <iostream>

namespace MyLongCompanyName {
    Only members defined in package namespace!
    namespace bdeg { // empty
    } // package namespace

    std::ostream&
    bdeg::operator<<(std::ostream&
        stream, const Point& point)
    {
        return stream << '(' <<
            point.x() << ',' <<
            point.y() << ')' <<
            std::flush;
    }
    // ...
}
```

} // enterprise namespace

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// bdeg_point.h
// ...
namespace bdeg {
class Point {
    int d_x;
    int d_y;
public:
    // CREATORS
    Point();
    Point(int x, int y);
    Point(const Point& other);
    ~Point()
    // MANIPULATORS
    Point& operator=(const Point& rhs);
    void setX(int x);
    void setY(int y);
    // ACCESSORS
    int x() const;
    int y() const;
};
// FREE OPERATORS
bool operator==(const Point& lhs,
                  const Point& rhs);
// ... (e.g., output operator)
```

```
// bdeg_point.cpp
#include <bdeg_point.h>
#include <iostream>

namespace MyLongCompanyName {

namespace bdeg { // empty
} // package namespace
Free operators/functions are defined outside.
std::ostream&
bdeg::operator<<(std::ostream&
stream, const Point& point)
{
    return stream << '(' <<
        point.x() << ',' <<
        point.y() << ')' <<
        std::flush;
}
// ...
} // enterprise namespace
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h
#ifndef INCLUDED_Z_BDEG_POINT
#define INCLUDED_Z_BDEG_POINT
#ifdef __cplusplus
extern "C" {
    namespace MyLongCompanyName {
        namespace bdeg {
            class Point;
        } // package namespace
    } // enterprise namespace
    typedef
        MyLongCompanyName::bdeg::Point
    z_bdeg_Point;
} // else
    typedef struct z_bdeg_Point
        z_bdeg_Point;
#endif // internal include guard
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h
#ifndef INCLUDED_Z_BDEG_POINT
#define INCLUDED_Z_BDEG_POINT
#ifdef __cplusplus
extern "C" {
    namespace MyLongCompanyName {
        namespace bdeg {
            class Point;
        } // package namespace
    } // enterprise namespace
    typedef
        MyLongCompanyName::bdeg::Point
    z_bdeg_Point;
} // else
    typedef struct z_bdeg_Point
        z_bdeg_Point;
#endif // internal include guard
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h
#ifndef INCLUDED_Z_BDEG_POINT
#define INCLUDED_Z_BDEG_POINT
#ifdef __cplusplus
extern "C" {
    namespace MyLongCompanyName {
        namespace bdeg {
            class Point;
        } // package namespace
    } // enterprise namespace
    typedef
    MyLongCompanyName::bdeg::Point
    z_bdeg_Point;
} // enterprise namespace
#endif // internal include guard
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h
#ifndef INCLUDED_z_BDEG_POINT
#define INCLUDED_z_BDEG_POINT
#ifndef __cplusplus
extern "C" {
    namespace MyLongCompanyName {
        namespace bdeg {
            class Point;
        } // package namespace
    } // enterprise namespace
    typedef
    MyLongCompanyName::bdeg::Point
    z_bdeg_Point;
} // else
typedef struct z_bdeg_Point
    z_bdeg_Point;
#endif // internal include guard
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h
#ifndef INCLUDED_Z_BDEG_POINT
#define INCLUDED_Z_BDEG_POINT
#ifndef __cplusplus
extern "C" {
    namespace MyLongCompanyName {
        namespace bdeg {
            class Point;
        } // package namespace
    } // enterprise namespace
    typedef
    MyLongCompanyName::bdeg::Point
    z_bdeg_Point;
} // else
    typedef struct z_bdeg_Point
                    z_bdeg_Point;
#endif // internal include guard
```

The **Z_** Prefix
Provides One-To-One
Mapping Between
Procedural Wrappers
and Corresponding
C++ Components.

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h
#ifndef INCLUDED_Z_BDEG_POINT
#define INCLUDED_Z_BDEG_POINT
#ifdef __cplusplus
extern "C" {

namespace MyLongCompanyName {
    namespace bdeg {
        class Point;
    } // package namespace
} // enterprise namespace
typedef
MyLongCompanyName::bdeg::Point
z_bdeg_Point;
#else
typedef struct z_bdeg_Point
    z_bdeg_Point;
#endif // internal include guard
```

```
// z_bdeg_point.h (continued)
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate();
z_bdeg_Point *z_bdeg_Point_fCreateInit(
    int x, int y);
z_bdeg_Point *z_bdeg_Point_fCreateCopy(
    const z_bdeg_Point *other);
void z_bdeg_Point_fDestroy(
    z_bdeg_Point *object);

// MANIPULATORS
void z_bdeg_Point_fAssign(
    z_bdeg_Point *object,
    const z_bdeg_Point *other);
void z_bdeg_Point_fSetX(
    z_bdeg_Point *object, int x);
void z_bdeg_Point_fSetY(
    z_bdeg_Point *object, int y);

// ACCESSORS
int z_bdeg_Point_fx(
    const z_bdeg_Point *object);
int z_bdeg_Point_fy(
    const z_bdeg_Point *object);

// FREE OPERATIONS
int z_bdeg_Point_fIsEqual(
    const z_bdeg_Point *object,
    const z_bdeg_Point *other); // ...
#ifdef __cplusplus
} // extern "C" linkage
#endif
#endif // internal include guard
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h
#ifndef INCLUDED_Z_BDEG_POINT
#define INCLUDED_Z_BDEG_POINT
#ifdef __cplusplus
extern "C" {

namespace MyLongCompanyName {
    namespace bdeg {
        class Point;
    } // package namespace
} // enterprise namespace
typedef
MyLongCompanyName::bdeg::Point
z_bdeg_Point;
#else
typedef struct z_bdeg_Point
    z_bdeg_Point;
#endif // internal include guard
```

```
// z_bdeg_point.h (continued)
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate();
z_bdeg_Point *z_bdeg_Point_fCreateInit(
    int x, int y);
z_bdeg_Point *z_bdeg_Point_fCreateCopy(
    const z_bdeg_Point *other);
void z_bdeg_Point_fDestroy(
    z_bdeg_Point *object);

// MANIPULATORS
void z_bdeg_Point_fAssign(
    z_bdeg_Point *object,
    const z_bdeg_Point *other);
void z_bdeg_Point_fSetX(
    z_bdeg_Point *object, int x);
void z_bdeg_Point_fSetY(
    z_bdeg_Point *object, int y);

// ACCESSORS
int z_bdeg_Point_fx(
    const z_bdeg_Point *object);
int z_bdeg_Point_fy(
    const z_bdeg_Point *object);

// FREE OPERATIONS
int z_bdeg_Point_fIsEqual(
    const z_bdeg_Point *object,
    const z_bdeg_Point *other); // ...
#ifdef __cplusplus
} // extern "C" linkage
#endif
#endif // internal include guard
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h
#ifndef INCLUDED_Z_BDEG_POINT
#define INCLUDED_Z_BDEG_POINT
#ifdef __cplusplus
extern "C" {

    namespace MyLongCompanyName {
        namespace bdeg {
            class Point;
        } // package namespace
    } // enterprise namespace
typedef
MyLongCompanyName::bdeg::Point
z_bdeg_Point;

#else

    typedef struct z_bdeg_Point
        z_bdeg_Point;

#endif // internal include guard
```

```
// z_bdeg_point.h (continued)
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate();
z_bdeg_Point *z_bdeg_Point_fCreateInit(
                                int x, int y);
z_bdeg_Point *z_bdeg_Point_fCreateCopy(
                                const z_bdeg_Point *other);
void z_bdeg_Point_fDestroy(
                            z_bdeg_Point *object);

// MANIPULATORS
void z_bdeg_Point_fAssign(
                            z_bdeg_Point *object,
                            const z_bdeg_Point *other);
void z_bdeg_Point_fSetX(
                            z_bdeg_Point *object, int x);
void z_bdeg_Point_fSetY(
                            z_bdeg_Point *object, int y);

// ACCESSORS
int z_bdeg_Point_fx(
                            const z_bdeg_Point *object);
int z_bdeg_Point_fy(
                            const z_bdeg_Point *object);

// FREE OPERATIONS
int z_bdeg_Point_fIsEqual(
                            const z_bdeg_Point *object,
                            const z_bdeg_Point *other); // ...
#ifdef __cplusplus
} // extern "C" linkage
#endif
#endif // internal include guard
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h (continued)
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate();
z_bdeg_Point *z_bdeg_Point_fCreateInit(
    int x, int y);
z_bdeg_Point *z_bdeg_Point_fCreateCopy(
    const z_bdeg_Point *other);
void z_bdeg_Point_fDestroy(
    z_bdeg_Point *object);

// MANIPULATORS
void z_bdeg_Point_fAssign(
    z_bdeg_Point *object,
    const z_bdeg_Point *other);
void z_bdeg_Point_fSetX(
    z_bdeg_Point *object, int x);
void z_bdeg_Point_fSetY(
    z_bdeg_Point *object, int y);

// ACCESSORS
int z_bdeg_Point_fx(
    const z_bdeg_Point *object);
int z_bdeg_Point_fy(
    const z_bdeg_Point *object);

// FREE OPERATIONS
int z_bdeg_Point_fIsEqual(
    const z_bdeg_Point *object,
    const z_bdeg_Point *other); // ...
#endif __cplusplus
} // extern "C" linkage
#endif
#endif // internal include guard
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h (continued)
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate();
z_bdeg_Point *z_bdeg_Point_fCreateInit(
    int x, int y);
z_bdeg_Point *z_bdeg_Point_fCreateCopy(
    const z_bdeg_Point *other);
void z_bdeg_Point_fDestroy(
    z_bdeg_Point *object);

// MANIPULATORS
void z_bdeg_Point_fAssign(
    z_bdeg_Point *object,
    const z_bdeg_Point *other);
void z_bdeg_Point_fSetX(
    z_bdeg_Point *object, int x);
void z_bdeg_Point_fSetY(
    z_bdeg_Point *object, int y);

// ACCESSORS
int z_bdeg_Point_fx(
    const z_bdeg_Point *object);
int z_bdeg_Point_fy(
    const z_bdeg_Point *object);

// FREE OPERATIONS
int z_bdeg_Point_fisEqual(
    const z_bdeg_Point *object,
    const z_bdeg_Point *other); // ...

#ifndef __cplusplus
} // extern "C" linkage
#endif
#endif // internal include guard
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h (continued)
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate();
z_bdeg_Point *z_bdeg_Point_fCreateInit(
    int x, int y);
z_bdeg_Point *z_bdeg_Point_fCreateCopy(
    const z_bdeg_Point *other);
void z_bdeg_Point_fDestroy(
    z_bdeg_Point *object);

// MANIPULATORS
void z_bdeg_Point_fAssign(
    z_bdeg_Point *object,
    const z_bdeg_Point *other);
void z_bdeg_Point_fSetX(
    z_bdeg_Point *object, int x);
void z_bdeg_Point_fSetY(
    z_bdeg_Point *object, int y);

// ACCESSORS
int z_bdeg_Point_fx(
    const z_bdeg_Point *object);
int z_bdeg_Point_fy(
    const z_bdeg_Point *object);

// FREE OPERATIONS
int z_bdeg_Point_fIsEqual(
    const z_bdeg_Point *object,
    const z_bdeg_Point *other); // ...

#ifndef __cplusplus
} // extern "C" linkage
#endif
#endif // internal include guard
```

```
// bdeg_point.h
// ...
namespace bdeg {
class Point {
    int d_x;
    int d_y;
public:
    // CREATORS
    Point();
    Point(int x, int y);
    Point(const Point& other);
    ~Point();

    // MANIPULATORS
    Point& operator=(const Point& rhs);
    void setX(int x);
    void setY(int y);

    // ACCESSORS
    int x() const;
    int y() const;
};

// FREE OPERATORS
bool operator==(const Point& lhs,
                  const Point& rhs);
// ... (e.g., output operator)
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h (continued)
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate();
z_bdeg_Point *z_bdeg_Point_fCreateInit(
    int x, int y);
z_bdeg_Point *z_bdeg_Point_fCreateCopy(
    const z_bdeg_Point *other);
void z_bdeg_Point_fDestroy(
    z_bdeg_Point *object);

// MANIPULATORS
void z_bdeg_Point_fAssign(
    z_bdeg_Point *object,
    const z_bdeg_Point *other);
void z_bdeg_Point_fSetX(
    z_bdeg_Point *object, int x);
void z_bdeg_Point_fSetY(
    z_bdeg_Point *object, int y);

// ACCESSORS
int z_bdeg_Point_fx(
    const z_bdeg_Point *object);
int z_bdeg_Point_fy(
    const z_bdeg_Point *object);

// FREE OPERATIONS
int z_bdeg_Point_fIsEqual(
    const z_bdeg_Point *object,
    const z_bdeg_Point *other); // ...

#ifndef __cplusplus
} // extern "C" linkage
#endif
#endif // internal include guard
```

```
// bdeg_point.h
// ...
namespace bdeg {
class Point {
    int d_x;
    int d_y;
public:
    // CREATORS
    Point();
    Point(int x, int y);
    Point(const Point& other);
    ~Point();

    // MANIPULATORS
    Point& operator=(const Point& rhs);
    void setX(int x);
    void setY(int y);

    // ACCESSORS
    int x() const;
    int y() const;
};

// FREE OPERATORS
bool operator==(const Point& lhs,
                  const Point& rhs);
// ... (e.g., output operator)
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h (continued)
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate();
z_bdeg_Point *z_bdeg_Point_fCreateInit(
    int x, int y);
z_bdeg_Point *z_bdeg_Point_fCreateCopy(
    const z_bdeg_Point *other);
void z_bdeg_Point_fDestroy(
    z_bdeg_Point *object);

// MANIPULATORS
void z_bdeg_Point_fAssign(
    z_bdeg_Point *object,
    const z_bdeg_Point *other);
void z_bdeg_Point_fSetX(
    z_bdeg_Point *object, int x);
void z_bdeg_Point_fSetY(
    z_bdeg_Point *object, int y);

// ACCESSORS
int z_bdeg_Point_fx(
    const z_bdeg_Point *object);
int z_bdeg_Point_fy(
    const z_bdeg_Point *object);

// FREE OPERATIONS
int z_bdeg_Point_fIsEqual(
    const z_bdeg_Point *object,
    const z_bdeg_Point *other); // ...

#ifndef __cplusplus
} // extern "C" linkage
#endif
#endif // internal include guard
```

```
// bdeg_point.h
// ...
namespace bdeg {
class Point {
    int d_x;
    int d_y;
public:
    // CREATORS
    Point();
    Point(int x, int y);
    Point(const Point& other);
    ~Point();

    // MANIPULATORS
    Point& operator=(const Point& rhs);
    void setX(int x);
    void setY(int y);

    // ACCESSORS
    int x() const;
    int y() const;
};

// FREE OPERATORS
bool operator==(const Point& lhs,
                  const Point& rhs);
// ... (e.g., output operator)
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h (continued)
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate();
z_bdeg_Point *z_bdeg_Point_fCreateInit(
    int x, int y);
z_bdeg_Point *z_bdeg_Point_fCreateCopy(
    const z_bdeg_Point *other);
void z_bdeg_Point_fDestroy(
    z_bdeg_Point *object);

// MANIPULATORS
void z_bdeg_Point_fAssign(
    z_bdeg_Point *object,
    const z_bdeg_Point *other);
void z_bdeg_Point_fSetX(
    z_bdeg_Point *object, int x);
void z_bdeg_Point_fSetY(
    z_bdeg_Point *object, int y);

// ACCESSORS
int z_bdeg_Point_fx(
    const z_bdeg_Point *object);
int z_bdeg_Point_fy(
    const z_bdeg_Point *object);

// FREE OPERATIONS
int z_bdeg_Point_fIsEqual(
    const z_bdeg_Point *object,
    const z_bdeg_Point *other); // ...

#ifndef __cplusplus
} // extern "C" linkage
#endif
#endif // internal include guard
```

```
// bdeg_point.h
// ...
namespace bdeg {
class Point {
    int d_x;
    int d_y;
public:
    // CREATORS
    Point();
    Point(int x, int y);
Point(const Point& other);
    ~Point();

    // MANIPULATORS
    Point& operator=(const Point& rhs);
    void setX(int x);
    void setY(int y);

    // ACCESSORS
    int x() const;
    int y() const;
};

    // FREE OPERATORS
    bool operator==(const Point& lhs,
                      const Point& rhs);
    // ... (e.g., output operator)
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h (continued)
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate();
z_bdeg_Point *z_bdeg_Point_fCreateInit(
    int x, int y);
z_bdeg_Point *z_bdeg_Point_fCreateCopy(
    const z_bdeg_Point *other);
void z_bdeg_Point_fDestroy(
    z_bdeg_Point *object);

// MANIPULATORS
void z_bdeg_Point_fAssign(
    z_bdeg_Point *object,
    const z_bdeg_Point *other);
void z_bdeg_Point_fSetX(
    z_bdeg_Point *object, int x);
void z_bdeg_Point_fSetY(
    z_bdeg_Point *object, int y);

// ACCESSORS
int z_bdeg_Point_fx(
    const z_bdeg_Point *object);
int z_bdeg_Point_fy(
    const z_bdeg_Point *object);

// FREE OPERATIONS
int z_bdeg_Point_fIsEqual(
    const z_bdeg_Point *object,
    const z_bdeg_Point *other); // ...

#ifndef __cplusplus
} // extern "C" linkage
#endif
#endif // internal include guard
```

```
// bdeg_point.h
// ...
namespace bdeg {
class Point {
    int d_x;
    int d_y;
public:
    // CREATORS
    Point();
    Point(int x, int y);
    Point(const Point& other);
    ~Point();

    // MANIPULATORS
    Point& operator=(const Point& rhs);
    void setX(int x);
    void setY(int y);

    // ACCESSORS
    int x() const;
    int y() const;
};

    // FREE OPERATORS
    bool operator==(const Point& lhs,
                      const Point& rhs);
    // ... (e.g., output operator)
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h (continued)
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate();
z_bdeg_Point *z_bdeg_Point_fCreateInit(
    int x, int y);
z_bdeg_Point *z_bdeg_Point_fCreateCopy(
    const z_bdeg_Point *other);
void z_bdeg_Point_fDestroy(
    z_bdeg_Point *object);

// MANIPULATORS
void z_bdeg_Point_fAssign(
    z_bdeg_Point *object,
    const z_bdeg_Point *other);

void z_bdeg_Point_fSetX(
    z_bdeg_Point *object, int x);
void z_bdeg_Point_fSetY(
    z_bdeg_Point *object, int y);

// ACCESSORS
int z_bdeg_Point_fx(
    const z_bdeg_Point *object);
int z_bdeg_Point_fy(
    const z_bdeg_Point *object);

// FREE OPERATIONS
int z_bdeg_Point_fIsEqual(
    const z_bdeg_Point *object,
    const z_bdeg_Point *other); // ...

#ifndef __cplusplus
} // extern "C" linkage
#endif
#endif // internal include guard
```

```
// bdeg_point.h
// ...
namespace bdeg {
class Point {
    int d_x;
    int d_y;
public:
    // CREATORS
    Point();
    Point(int x, int y);
    Point(const Point& other);
    ~Point();

    // MANIPULATORS
    Point& operator=(const Point& rhs);
    void setX(int x);
    void setY(int y);

    // ACCESSORS
    int x() const;
    int y() const;
};

// FREE OPERATORS
bool operator==(const Point& lhs,
                  const Point& rhs);
// ... (e.g., output operator)
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h (continued)
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate();
z_bdeg_Point *z_bdeg_Point_fCreateInit(
    int x, int y);
z_bdeg_Point *z_bdeg_Point_fCreateCopy(
    const z_bdeg_Point *other);
void z_bdeg_Point_fDestroy(
    z_bdeg_Point *object);

// MANIPULATORS
void z_bdeg_Point_fAssign(
    z_bdeg_Point *object,
    const z_bdeg_Point *other);
void z_bdeg_Point_fSetX(
    z_bdeg_Point *object, int x);
void z_bdeg_Point_fSetY(
    z_bdeg_Point *object, int y);

// ACCESSORS
int z_bdeg_Point_fx(
    const z_bdeg_Point *object);
int z_bdeg_Point_fy(
    const z_bdeg_Point *object);

// FREE OPERATIONS
int z_bdeg_Point_fIsEqual(
    const z_bdeg_Point *object,
    const z_bdeg_Point *other); // ...

#ifndef __cplusplus
} // extern "C" linkage
#endif
#endif // internal include guard
```

```
// bdeg_point.h
// ...
namespace bdeg {
class Point {
    int d_x;
    int d_y;
public:
    // CREATORS
    Point();
    Point(int x, int y);
    Point(const Point& other);
    ~Point();

    // MANIPULATORS
    Point& operator=(const Point& rhs);
    void setX(int x);
    void setY(int y);

    // ACCESSORS
    int x() const;
    int y() const;
};

// FREE OPERATORS
bool operator==(const Point& lhs,
                  const Point& rhs);
// ... (e.g., output operator)
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h (continued)
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate();
z_bdeg_Point *z_bdeg_Point_fCreateInit(
    int x, int y);
z_bdeg_Point *z_bdeg_Point_fCreateCopy(
    const z_bdeg_Point *other);
void z_bdeg_Point_fDestroy(
    z_bdeg_Point *object);

// MANIPULATORS
void z_bdeg_Point_fAssign(
    z_bdeg_Point *object,
    const z_bdeg_Point *other);
void z_bdeg_Point_fSetX(
    z_bdeg_Point *object, int x);
void z_bdeg_Point_fSetY(
    z_bdeg_Point *object, int y);

// ACCESSORS
int z_bdeg_Point_fx(
    const z_bdeg_Point *object);
int z_bdeg_Point_fy(
    const z_bdeg_Point *object);

// FREE OPERATIONS
int z_bdeg_Point_fIsEqual(
    const z_bdeg_Point *object,
    const z_bdeg_Point *other); // ...

#ifndef __cplusplus
} // extern "C" linkage
#endif
#endif // internal include guard
```

```
// bdeg_point.h
// ...
namespace bdeg {
class Point {
    int d_x;
    int d_y;
public:
    // CREATORS
    Point();
    Point(int x, int y);
    Point(const Point& other);
    ~Point();

    // MANIPULATORS
    Point& operator=(const Point& rhs);
    void setX(int x);
void setY(int y);

    // ACCESSORS
    int x() const;
    int y() const;
};

// FREE OPERATORS
bool operator==(const Point& lhs,
                  const Point& rhs);
// ... (e.g., output operator)
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h (continued)
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate();
z_bdeg_Point *z_bdeg_Point_fCreateInit(
    int x, int y);
z_bdeg_Point *z_bdeg_Point_fCreateCopy(
    const z_bdeg_Point *other);
void z_bdeg_Point_fDestroy(
    z_bdeg_Point *object);

// MANIPULATORS
void z_bdeg_Point_fAssign(
    z_bdeg_Point *object,
    const z_bdeg_Point *other);
void z_bdeg_Point_fSetX(
    z_bdeg_Point *object, int x);
void z_bdeg_Point_fSetY(
    z_bdeg_Point *object, int y);

// ACCESSORS
int z_bdeg_Point_fx(
    const z_bdeg_Point *object);
int z_bdeg_Point_fy(
    const z_bdeg_Point *object);

// FREE OPERATIONS
int z_bdeg_Point_fIsEqual(
    const z_bdeg_Point *object,
    const z_bdeg_Point *other); // ...

#ifndef __cplusplus
} // extern "C" linkage
#endif
#endif // internal include guard
```

```
// bdeg_point.h
// ...
namespace bdeg {
class Point {
    int d_x;
    int d_y;
public:
    // CREATORS
    Point();
    Point(int x, int y);
    Point(const Point& other);
    ~Point();

    // MANIPULATORS
    Point& operator=(const Point& rhs);
    void setX(int x);
    void setY(int y);

    // ACCESSORS
    int x() const;
    int y() const;
};

// FREE OPERATORS
bool operator==(const Point& lhs,
                  const Point& rhs);
// ... (e.g., output operator)
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h (continued)
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate();
z_bdeg_Point *z_bdeg_Point_fCreateInit(
    int x, int y);
z_bdeg_Point *z_bdeg_Point_fCreateCopy(
    const z_bdeg_Point *other);
void z_bdeg_Point_fDestroy(
    z_bdeg_Point *object);

// MANIPULATORS
void z_bdeg_Point_fAssign(
    z_bdeg_Point *object,
    const z_bdeg_Point *other);
void z_bdeg_Point_fSetX(
    z_bdeg_Point *object, int x);
void z_bdeg_Point_fSetY(
    z_bdeg_Point *object, int y);

// ACCESSORS
int z_bdeg_Point_fx(
    const z_bdeg_Point *object);
int z_bdeg_Point_fy(
    const z_bdeg_Point *object);

// FREE OPERATIONS
int z_bdeg_Point_fIsEqual(
    const z_bdeg_Point *object,
    const z_bdeg_Point *other); // ...

#ifndef __cplusplus
} // extern "C" linkage
#endif
#endif // internal include guard
```

```
// bdeg_point.h
// ...
namespace bdeg {
class Point {
    int d_x;
    int d_y;
public:
    // CREATORS
    Point();
    Point(int x, int y);
    Point(const Point& other);
    ~Point();

    // MANIPULATORS
    Point& operator=(const Point& rhs);
    void setX(int x);
    void setY(int y);

    // ACCESSORS
    int x() const;
    int y() const;
};

// FREE OPERATORS
bool operator==(const Point& lhs,
                  const Point& rhs);
// ... (e.g., output operator)
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h (continued)
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate();
z_bdeg_Point *z_bdeg_Point_fCreateInit(
    int x, int y);
z_bdeg_Point *z_bdeg_Point_fCreateCopy(
    const z_bdeg_Point *other);
void z_bdeg_Point_fDestroy(
    z_bdeg_Point *object);

// MANIPULATORS
void z_bdeg_Point_fAssign(
    z_bdeg_Point *object,
    const z_bdeg_Point *other);
void z_bdeg_Point_fSetX(
    z_bdeg_Point *object, int x);
void z_bdeg_Point_fSetY(
    z_bdeg_Point *object, int y);

// ACCESSORS
int z_bdeg_Point_fx(
    const z_bdeg_Point *object);
int z_bdeg_Point_fy(
    const z_bdeg_Point *object);

// FREE OPERATIONS
int z_bdeg_Point_fIsEqual(
    const z_bdeg_Point *object,
    const z_bdeg_Point *other); // ...

#ifndef __cplusplus
} // extern "C" linkage
#endif
#endif // internal include guard
```

```
// bdeg_point.h
// ...
namespace bdeg {
class Point {
    int d_x;
    int d_y;
public:
    // CREATORS
    Point();
    Point(int x, int y);
    Point(const Point& other);
    ~Point();

    // MANIPULATORS
    Point& operator=(const Point& rhs);
    void setX(int x);
    void setY(int y);

    // ACCESSORS
    int x() const;
    int y() const;
};

// FREE OPERATORS
bool operator==(const Point& lhs,
                  const Point& rhs);
// ... (e.g., output operator)
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h (continued)
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate();
z_bdeg_Point *z_bdeg_Point_fCreateInit(
    int x, int y);
z_bdeg_Point *z_bdeg_Point_fCreateCopy(
    const z_bdeg_Point *other);
void z_bdeg_Point_fDestroy(
    z_bdeg_Point *object);

// MANIPULATORS
void z_bdeg_Point_fAssign(
    z_bdeg_Point *object,
    const z_bdeg_Point *other);
void z_bdeg_Point_fSetX(
    z_bdeg_Point *object, int x);
void z_bdeg_Point_fSetY(
    z_bdeg_Point *object, int y);

// ACCESSORS
int z_bdeg_Point_fx(
    const z_bdeg_Point *object);
int z_bdeg_Point_fy(
    const z_bdeg_Point *object);

// FREE OPERATORS
int z_bdeg_Point_fIsEqual(
    const z_bdeg_Point *object,
    const z_bdeg_Point *other); // ...

#ifndef __cplusplus
} // extern "C" linkage
#endif
#endif // internal include guard
```

```
// z_bdeg_point.cpp
#include <z_bdeg_point.h>
#include <bdeg_point.h>
#include <cassert>
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate() {
    return new bdeg::Point;
}
z_bdeg_Point *z_bdeg_Point_fCreateInit(int x, int y) {
    return new bdeg::Point(x, y);
}
z_bdeg_Point *z_bdeg_Point_fCreateCopy(const z_bdeg_Point *other) {
    assert(other);
    return new bdeg::Point(*other);
}
void z_bdeg_Point_fDestroy(z_bdeg_Point *object) {
    assert(object);
    delete object;
}

// MANIPULATORS
void z_bdeg_Point_fAssign(z_bdeg_Point *object, z_bdeg_Point *other) {
    assert(object);
    assert(other);
    *object = *other;
}
void z_bdeg_Point_fSetX(z_bdeg_Point *object, int x) {
    assert(object);
    object->setX(x);
}
void z_bdeg_Point_fSetY(z_bdeg_Point *object, int y) {
    assert(object);
    object->setY(y);
}

// ACCESSORS
int z_bdeg_Point_fx(const z_bdeg_Point *object) {
    assert(object);
    return object->x();
}
int z_bdeg_Point_fy(const z_bdeg_Point *object) {
    assert(object);
    return object->y();
}

// FREE OPERATORS
int z_bdeg_Point_fAreEqual(const z_bdeg_Point *object,
    const z_bdeg_Point *other) {
    assert(object);
    assert(other);
    return *object == *other;
} // ...
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h (continued)
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate();
z_bdeg_Point *z_bdeg_Point_fCreateInit(
    int x, int y);
z_bdeg_Point *z_bdeg_Point_fCreateCopy(
    const z_bdeg_Point *other);
void z_bdeg_Point_fDestroy(
    z_bdeg_Point *object);

// MANIPULATORS
void z_bdeg_Point_fAssign(
    z_bdeg_Point *object,
    const z_bdeg_Point *other);
void z_bdeg_Point_fSetX(
    z_bdeg_Point *object, int x);
void z_bdeg_Point_fSetY(
    z_bdeg_Point *object, int y);

// ACCESSORS
int z_bdeg_Point_fx(
    const z_bdeg_Point *object);
int z_bdeg_Point_fy(
    const z_bdeg_Point *object);

// FREE OPERATORS
int z_bdeg_Point_fIsEqual(
    const z_bdeg_Point *object,
    const z_bdeg_Point *other); // ...

#ifndef __cplusplus
} // extern "C" linkage
#endif
#endif // internal include guard
```

```
// z_bdeg_point.cpp
#include <z_bdeg_point.h>
#include <bdeg_point.h>
#include <cassert>
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate() {
    return new bdeg::Point;
}
z_bdeg_Point *z_bdeg_Point_fCreateInit(int x, int y) {
    return new bdeg::Point(x, y);
}
z_bdeg_Point *z_bdeg_Point_fCreateCopy(const z_bdeg_Point *other) {
    assert(other);
    return new bdeg::Point(*other);
}
void z_bdeg_Point_fDestroy(z_bdeg_Point *object) {
    assert(object);
    delete object;
}

// MANIPULATORS
void z_bdeg_Point_fAssign(z_bdeg_Point *object, z_bdeg_Point *other) {
    assert(object);
    assert(other);
    *object = *other;
}
void z_bdeg_Point_fSetX(z_bdeg_Point *object, int x) {
    assert(object);
    object->setX(x);
}
void z_bdeg_Point_fSetY(z_bdeg_Point *object, int y) {
    assert(object);
    object->setY(y);
}

// ACCESSORS
int z_bdeg_Point_fx(const z_bdeg_Point *object) {
    assert(object);
    return object->x();
}
int z_bdeg_Point_fy(const z_bdeg_Point *object) {
    assert(object);
    return object->y();
}

// FREE OPERATORS
int z_bdeg_Point_fAreEqual(const z_bdeg_Point *object,
    const z_bdeg_Point *other) {
    assert(object);
    assert(other);
    return *object == *other;
} // ...
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h (continued)
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate();
z_bdeg_Point *z_bdeg_Point_fCreateInit(
    int x, int y);
z_bdeg_Point *z_bdeg_Point_fCreateCopy(
    const z_bdeg_Point *other);
void z_bdeg_Point_fDestroy(
    z_bdeg_Point *object);

// MANIPULATORS
void z_bdeg_Point_fAssign(
    z_bdeg_Point *object,
    const z_bdeg_Point *other);
void z_bdeg_Point_fSetX(
    z_bdeg_Point *object, int x);
void z_bdeg_Point_fSetY(
    z_bdeg_Point *object, int y);

// ACCESSORS
int z_bdeg_Point_fx(
    const z_bdeg_Point *object);
int z_bdeg_Point_fy(
    const z_bdeg_Point *object);

// FREE OPERATORS
int z_bdeg_Point_fIsEqual(
    const z_bdeg_Point *object,
    const z_bdeg_Point *other); // ...

#ifndef __cplusplus
} // extern "C" linkage
#endif
#endif // internal include guard
```

```
// z_bdeg_point.cpp
#include <z_bdeg_point.h>
#include <bdeg_point.h>
#include <cassert>
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate() {
    return new bdeg::Point;
}
z_bdeg_Point *z_bdeg_Point_fCreateInit(int x, int y) {
    return new bdeg::Point(x, y);
}
z_bdeg_Point *z_bdeg_Point_fCreateCopy(const z_bdeg_Point *other) {
    assert(other);
    return new bdeg::Point(*other);
}
void z_bdeg_Point_fDestroy(z_bdeg_Point *object) {
    assert(object);
    delete object;
}

// MANIPULATORS
void z_bdeg_Point_fAssign(z_bdeg_Point *object, z_bdeg_Point *other) {
    assert(object);
    assert(other);
    *object = *other;
}
void z_bdeg_Point_fSetX(z_bdeg_Point *object, int x) {
    assert(object);
    object->setX(x);
}
void z_bdeg_Point_fSetY(z_bdeg_Point *object, int y) {
    assert(object);
    object->setY(y);
}

// ACCESSORS
int z_bdeg_Point_fx(const z_bdeg_Point *object) {
    assert(object);
    return object->x();
}
int z_bdeg_Point_fy(const z_bdeg_Point *object) {
    assert(object);
    return object->y();
}

// FREE OPERATORS
int z_bdeg_Point_fAreEqual(const z_bdeg_Point *object,
    const z_bdeg_Point *other) {
    assert(object);
    assert(other);
    return *object == *other;
} // ...
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h (continued)
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate();
z_bdeg_Point *z_bdeg_Point_fCreateInit(
    int x, int y);
z_bdeg_Point *z_bdeg_Point_fCreateCopy(
    const z_bdeg_Point *other);
void z_bdeg_Point_fDestroy(
    z_bdeg_Point *object);

// MANIPULATORS
void z_bdeg_Point_fAssign(
    z_bdeg_Point *object,
    const z_bdeg_Point *other);
void z_bdeg_Point_fSetX(
    z_bdeg_Point *object, int x);
void z_bdeg_Point_fSetY(
    z_bdeg_Point *object, int y);

// ACCESSORS
int z_bdeg_Point_fx(
    const z_bdeg_Point *object);
int z_bdeg_Point_fy(
    const z_bdeg_Point *object);

// FREE OPERATORS
int z_bdeg_Point_fIsEqual(
    const z_bdeg_Point *object,
    const z_bdeg_Point *other); // ...

#ifndef __cplusplus
} // extern "C" linkage
#endif
#endif // internal include guard
```

```
// z_bdeg_point.cpp
#include <z_bdeg_point.h>
#include <bdeg_point.h>
#include <cassert>
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate() {
    return new bdeg::Point;
}
z_bdeg_Point *z_bdeg_Point_fCreateInit(int x, int y) {
    return new bdeg::Point(x, y);
}
z_bdeg_Point *z_bdeg_Point_fCreateCopy(const z_bdeg_Point *other) {
    assert(other);
    return new bdeg::Point(*other);
}
void z_bdeg_Point_fDestroy(z_bdeg_Point *object) {
    assert(object);
    delete object;
}

// MANIPULATORS
void z_bdeg_Point_fAssign(z_bdeg_Point *object, z_bdeg_Point *other) {
    assert(object);
    assert(other);
    *object = *other;
}
void z_bdeg_Point_fSetX(z_bdeg_Point *object, int x) {
    assert(object);
    object->setX(x);
}
void z_bdeg_Point_fSetY(z_bdeg_Point *object, int y) {
    assert(object);
    object->setY(y);
}

// ACCESSORS
int z_bdeg_Point_fx(const z_bdeg_Point *object) {
    assert(object);
    return object->x();
}
int z_bdeg_Point_fy(const z_bdeg_Point *object) {
    assert(object);
    return object->y();
}

// FREE OPERATORS
int z_bdeg_Point_fAreEqual(const z_bdeg_Point *object,
    const z_bdeg_Point *other) {
    assert(object);
    assert(other);
    return *object == *other;
} // ...
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h (continued)
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate();
z_bdeg_Point *z_bdeg_Point_fCreateInit(
    int x, int y);
z_bdeg_Point *z_bdeg_Point_fCreateCopy(
    const z_bdeg_Point *other);
void z_bdeg_Point_fDestroy(
    z_bdeg_Point *object);

// MANIPULATORS
void z_bdeg_Point_fAssign(
    z_bdeg_Point *object,
    const z_bdeg_Point *other);
void z_bdeg_Point_fSetX(
    z_bdeg_Point *object, int x);
void z_bdeg_Point_fSetY(
    z_bdeg_Point *object, int y);

// ACCESSORS
int z_bdeg_Point_fx(
    const z_bdeg_Point *object);
int z_bdeg_Point_fy(
    const z_bdeg_Point *object);

// FREE OPERATORS
int z_bdeg_Point_fIsEqual(
    const z_bdeg_Point *object,
    const z_bdeg_Point *other); // ...

#ifndef __cplusplus
} // extern "C" linkage
#endif
#endif // internal include guard
```

```
// z_bdeg_point.cpp
#include <z_bdeg_point.h>
#include <bdeg_point.h>
#include <cassert>
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate() {
    return new bdeg::Point;
}
z_bdeg_Point *z_bdeg_Point_fCreateInit(int x, int y) {
    return new bdeg::Point(x, y);
}
z_bdeg_Point *z_bdeg_Point_fCreateCopy(const z_bdeg_Point *other) {
    assert(other);
    return new bdeg::Point(*other);
}
void z_bdeg_Point_fDestroy(z_bdeg_Point *object) {
    assert(object);
    delete object;
}

// MANIPULATORS
void z_bdeg_Point_fAssign(z_bdeg_Point *object, z_bdeg_Point *other) {
    assert(object);
    assert(other);
    *object = *other;
}
void z_bdeg_Point_fSetX(z_bdeg_Point *object, int x) {
    assert(object);
    object->setX(x);
}
void z_bdeg_Point_fSetY(z_bdeg_Point *object, int y) {
    assert(object);
    object->setY(y);
}

// ACCESSORS
int z_bdeg_Point_fx(const z_bdeg_Point *object) {
    assert(object);
    return object->x();
}
int z_bdeg_Point_fy(const z_bdeg_Point *object) {
    assert(object);
    return object->y();
}

// FREE OPERATORS
int z_bdeg_Point_fAreEqual(const z_bdeg_Point *object,
    const z_bdeg_Point *other) {
    assert(object);
    assert(other);
    return *object == *other;
} // ...
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h (continued)
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate();
z_bdeg_Point *z_bdeg_Point_fCreateInit(
    int x, int y);
z_bdeg_Point *z_bdeg_Point_fCreateCopy(
    const z_bdeg_Point *other);
void z_bdeg_Point_fDestroy(
    z_bdeg_Point *object);

// MANIPULATORS
void z_bdeg_Point_fAssign(
    z_bdeg_Point *object,
    const z_bdeg_Point *other);
void z_bdeg_Point_fSetX(
    z_bdeg_Point *object, int x);
void z_bdeg_Point_fSetY(
    z_bdeg_Point *object, int y);

// ACCESSORS
int z_bdeg_Point_fx(
    const z_bdeg_Point *object);
int z_bdeg_Point_fy(
    const z_bdeg_Point *object);

// FREE OPERATORS
int z_bdeg_Point_fIsEqual(
    const z_bdeg_Point *object,
    const z_bdeg_Point *other); // ...

#ifndef __cplusplus
} // extern "C" linkage
#endif
#endif // internal include guard
```

```
// z_bdeg_point.cpp
#include <z_bdeg_point.h>
#include <bdeg_point.h>
#include <cassert>
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate() {
    return new bdeg::Point;
}
z_bdeg_Point *z_bdeg_Point_fCreateInit(int x, int y) {
    return new bdeg::Point(x, y);
}
z_bdeg_Point *z_bdeg_Point_fCreateCopy(const z_bdeg_Point *other) {
    assert(other);
    return new bdeg::Point(*other);
}
void z_bdeg_Point_fDestroy(z_bdeg_Point *object) {
    assert(object);
    delete object;
}

// MANIPULATORS
void z_bdeg_Point_fAssign(z_bdeg_Point *object, z_bdeg_Point *other) {
    assert(object);
    assert(other);
    *object = *other;
}
void z_bdeg_Point_fSetX(z_bdeg_Point *object, int x) {
    assert(object);
    object->setX(x);
}
void z_bdeg_Point_fSetY(z_bdeg_Point *object, int y) {
    assert(object);
    object->setY(y);
}

// ACCESSORS
int z_bdeg_Point_fx(const z_bdeg_Point *object) {
    assert(object);
    return object->x();
}
int z_bdeg_Point_fy(const z_bdeg_Point *object) {
    assert(object);
    return object->y();
}

// FREE OPERATORS
int z_bdeg_Point_fAreEqual(const z_bdeg_Point *object,
                           const z_bdeg_Point *other) {
    assert(object);
    assert(other);
    return *object == *other;
} // ...
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h (continued)
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate();
z_bdeg_Point *z_bdeg_Point_fCreateInit(
    int x, int y);
z_bdeg_Point *z_bdeg_Point_fCreateCopy(
    const z_bdeg_Point *other);
void z_bdeg_Point_fDestroy(
    z_bdeg_Point *object);

// MANIPULATORS
void z_bdeg_Point_fAssign(
    z_bdeg_Point *object,
    const z_bdeg_Point *other);
void z_bdeg_Point_fSetX(
    z_bdeg_Point *object, int x);
void z_bdeg_Point_fSetY(
    z_bdeg_Point *object, int y);

// ACCESSORS
int z_bdeg_Point_fx(
    const z_bdeg_Point *object);
int z_bdeg_Point_fy(
    const z_bdeg_Point *object);

// FREE OPERATORS
int z_bdeg_Point_fIsEqual(
    const z_bdeg_Point *object,
    const z_bdeg_Point *other); // ...

#ifndef __cplusplus
} // extern "C" linkage
#endif
#endif // internal include guard
```

```
// z_bdeg_point.cpp
#include <z_bdeg_point.h>
#include <bdeg_point.h>
#include <cassert>           <- defensive programming

// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate() {
    return new bdeg::Point;
}
z_bdeg_Point *z_bdeg_Point_fCreateInit(int x, int y) {
    return new bdeg::Point(x, y);
}
z_bdeg_Point *z_bdeg_Point_fCreateCopy(const z_bdeg_Point *other) {
    assert(other);
    return new bdeg::Point(*other);
}

void z_bdeg_Point_fDestroy(z_bdeg_Point *object) {
    assert(object);
    delete object;
}

// MANIPULATORS
void z_bdeg_Point_fAssign(z_bdeg_Point *object, z_bdeg_Point *other) {
    assert(object);
    assert(other);
    *object = *other;
}
void z_bdeg_Point_fSetX(z_bdeg_Point *object, int x) {
    assert(object);
    object->setX(x);
}
void z_bdeg_Point_fSetY(z_bdeg_Point *object, int y) {
    assert(object);
    object->setY(y);
}

// ACCESSORS
int z_bdeg_Point_fx(const z_bdeg_Point *object) {
    assert(object);
    return object->x();
}
int z_bdeg_Point_fy(const z_bdeg_Point *object) {
    assert(object);
    return object->y();
}

// FREE OPERATORS
int z_bdeg_Point_fAreEqual(const z_bdeg_Point *object,
                           const z_bdeg_Point *other) {
    assert(object);
    assert(other);
    return *object == *other;
}
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h (continued)
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate();
z_bdeg_Point *z_bdeg_Point_fCreateInit(
    int x, int y);
z_bdeg_Point *z_bdeg_Point_fCreateCopy(
    const z_bdeg_Point *other);
void z_bdeg_Point_fDestroy(
    z_bdeg_Point *object);

// MANIPULATORS
void z_bdeg_Point_fAssign(
    z_bdeg_Point *object,
    const z_bdeg_Point *other);
void z_bdeg_Point_fSetX(
    z_bdeg_Point *object, int x);
void z_bdeg_Point_fSetY(
    z_bdeg_Point *object, int y);

// ACCESSORS
int z_bdeg_Point_fx(
    const z_bdeg_Point *object);
int z_bdeg_Point_fy(
    const z_bdeg_Point *object);

// FREE OPERATORS
int z_bdeg_Point_fIsEqual(
    const z_bdeg_Point *object,
    const z_bdeg_Point *other); // ...

#ifndef __cplusplus
} // extern "C" linkage
#endif
#endif // internal include guard
```

```
// z_bdeg_point.cpp
#include <z_bdeg_point.h>
#include <bdeg_point.h>
#include <cassert>           <- defensive programming

// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate() {
    return new bdeg::Point;
}
z_bdeg_Point *z_bdeg_Point_fCreateInit(int x, int y) {
    return new bdeg::Point(x, y);
}
z_bdeg_Point *z_bdeg_Point_fCreateCopy(const z_bdeg_Point *other) {
    assert(other);
    return new bdeg::Point(*other);
}
void z_bdeg_Point_fDestroy(z_bdeg_Point *object) {
    assert(object);
    delete object;
}

// MANIPULATORS
void z_bdeg_Point_fAssign(z_bdeg_Point *object, z_bdeg_Point *other) {
    assert(object);
    assert(other);
    *object = *other;
}
void z_bdeg_Point_fSetX(z_bdeg_Point *object, int x) {
    assert(object);
    object->setX(x);
}
void z_bdeg_Point_fSetY(z_bdeg_Point *object, int y) {
    assert(object);
    object->setY(y);
}

// ACCESSORS
int z_bdeg_Point_fx(const z_bdeg_Point *object) {
    assert(object);
    return object->x();
}
int z_bdeg_Point_fy(const z_bdeg_Point *object) {
    assert(object);
    return object->y();
}

// FREE OPERATORS
int z_bdeg_Point_fAreEqual(const z_bdeg_Point *object,
                           const z_bdeg_Point *other) {
    assert(object);
    assert(other);
    return *object == *other;
} // ...
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h (continued)
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate();
z_bdeg_Point *z_bdeg_Point_fCreateInit(
    int x, int y);
z_bdeg_Point *z_bdeg_Point_fCreateCopy(
    const z_bdeg_Point *other);
void z_bdeg_Point_fDestroy(
    z_bdeg_Point *object);

// MANIPULATORS
void z_bdeg_Point_fAssign(
    z_bdeg_Point *object,
    const z_bdeg_Point *other);
void z_bdeg_Point_fSetX(
    z_bdeg_Point *object, int x);
void z_bdeg_Point_fSetY(
    z_bdeg_Point *object, int y);

// ACCESSORS
int z_bdeg_Point_fx(
    const z_bdeg_Point *object);
int z_bdeg_Point_fy(
    const z_bdeg_Point *object);

// FREE OPERATORS
int z_bdeg_Point_fIsEqual(
    const z_bdeg_Point *object,
    const z_bdeg_Point *other); // ...

#ifndef __cplusplus
} // extern "C" linkage
#endif
#endif // internal include guard
```

```
// z_bdeg_point.cpp
#include <z_bdeg_point.h>
#include <bdeg_point.h>
#include <bsls_assert.h>           <- defensive programming

// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate() {
    return new bdeg::Point;
}
z_bdeg_Point *z_bdeg_Point_fCreateInit(int x, int y) {
    return new bdeg::Point(x, y);
}
z_bdeg_Point *z_bdeg_Point_fCreateCopy(const z_bdeg_Point *other) {
    assert(other);
    return new bdeg::Point(*other);
}
void z_bdeg_Point_fDestroy(z_bdeg_Point *object) {
    assert(object);
    delete object;
}

// MANIPULATORS
void z_bdeg_Point_fAssign(z_bdeg_Point *object, z_bdeg_Point *other) {
    assert(object);
    assert(other);
    *object = *other;
}
void z_bdeg_Point_fSetX(z_bdeg_Point *object, int x) {
    assert(object);
    object->setX(x);
}
void z_bdeg_Point_fSetY(z_bdeg_Point *object, int y) {
    assert(object);
    object->setY(y);
}

// ACCESSORS
int z_bdeg_Point_fx(const z_bdeg_Point *object) {
    assert(object);
    return object->x();
}
int z_bdeg_Point_fy(const z_bdeg_Point *object) {
    assert(object);
    return object->y();
}

// FREE OPERATORS
int z_bdeg_Point_fAreEqual(const z_bdeg_Point *object,
                           const z_bdeg_Point *other) {
    assert(object);
    assert(other);
    return *object == *other;
} // ...
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h (continued)
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate();
z_bdeg_Point *z_bdeg_Point_fCreateInit(
    int x, int y);
z_bdeg_Point *z_bdeg_Point_fCreateCopy(
    const z_bdeg_Point *other);
void z_bdeg_Point_fDestroy(
    z_bdeg_Point *object);

// MANIPULATORS
void z_bdeg_Point_fAssign(
    z_bdeg_Point *object,
    const z_bdeg_Point *other);
void z_bdeg_Point_fSetX(
    z_bdeg_Point *object, int x);
void z_bdeg_Point_fSetY(
    z_bdeg_Point *object, int y);

// ACCESSORS
int z_bdeg_Point_fx(
    const z_bdeg_Point *object);
int z_bdeg_Point_fy(
    const z_bdeg_Point *object);

// FREE OPERATORS
int z_bdeg_Point_fIsEqual(
    const z_bdeg_Point *object,
    const z_bdeg_Point *other); // ...

#ifndef __cplusplus
} // extern "C" linkage
#endif
#endif // internal include guard
```

```
// z_bdeg_point.cpp
#include <z_bdeg_point.h>
#include <bdeg_point.h>
#include <bsls_assert.h>           <- defensive programming

// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate() {
    return new bdeg::Point;
}
z_bdeg_Point *z_bdeg_Point_fCreateInit(int x, int y) {
    return new bdeg::Point(x, y);
}
z_bdeg_Point *z_bdeg_Point_fCreateCopy(const z_bdeg_Point *other) {
    BSLS_ASSERT(other);
    return new bdeg::Point(*other);
}

void z_bdeg_Point_fDestroy(z_bdeg_Point *object) {
    BSLS_ASSERT(object);
    delete object;
}

// MANIPULATORS
void z_bdeg_Point_fAssign(z_bdeg_Point *object, z_bdeg_Point *other) {
    BSLS_ASSERT(object);
    BSLS_ASSERT(other);
    *object = *other;
}
void z_bdeg_Point_fSetX(z_bdeg_Point *object, int x) {
    BSLS_ASSERT(object);
    object->setX(x);
}
void z_bdeg_Point_fSetY(z_bdeg_Point *object, int y) {
    BSLS_ASSERT(object);
    object->setY(y);
}

// ACCESSORS
int z_bdeg_Point_fx(const z_bdeg_Point *object) {
    BSLS_ASSERT(object);
    return object->x();
}
int z_bdeg_Point_fy(const z_bdeg_Point *object) {
    BSLS_ASSERT(object);
    return object->y();
}

// FREE OPERATORS
int z_bdeg_Point_fAreEqual(const z_bdeg_Point *object,
                           const z_bdeg_Point *other) {
    BSLS_ASSERT(object);
    BSLS_ASSERT(other);
    return *object == *other;
} // ...
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h (continued)
// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate();
z_bdeg_Point *z_bdeg_Point_fCreateInit(
    int x, int y);
z_bdeg_Point *z_bdeg_Point_fCreateCopy(
    const z_bdeg_Point *other);
void z_bdeg_Point_fDestroy(
    z_bdeg_Point *object);

// MANIPULATORS
void z_bdeg_Point_fAssign(
    z_bdeg_Point *dest,
    z_bdeg_Point *src);

void z_bdeg_Point_fSet(
    z_bdeg_Point *object,
    int x);
void z_bdeg_Point_fSet(
    z_bdeg_Point *object,
    int x,
    int y);

// ACCESSORS
int z_bdeg_Point_fX(
    const z_bdeg_Point *object);
int z_bdeg_Point_fY(
    const z_bdeg_Point *object);

// FREE OPERATORS
int z_bdeg_Point_fAreEqual(
    const z_bdeg_Point *object,
    const z_bdeg_Point *other);

#ifndef __cplusplus
} // extern "C" linkage
#endif
#endif // internal include guard
```

```
// z_bdeg_point.cpp
#include <z_bdeg_point.h>
#include <bdeg_point.h>
#include <bsls_assert.h>           <- defensive programming

// CREATORS
z_bdeg_Point *z_bdeg_Point_fCreate() {
    return new bdeg_Point();
}
z_bdeg_Point *z_bdeg_Point_fCreateInit(int x, int y) {
    return new bdeg_Point(x, y);
}
z_bdeg_Point *z_bdeg_Point_fCreateCopy(const z_bdeg_Point *other) {
    return new bdeg_Point(*other);
}

// MANIPULATORS
void z_bdeg_Point_fAssign(z_bdeg_Point *dest, z_bdeg_Point *src) {
    dest->x = src->x;
    dest->y = src->y;
}

void z_bdeg_Point_fSet(z_bdeg_Point *object, int x) {
    object->x = x;
}
void z_bdeg_Point_fSet(z_bdeg_Point *object, int x, int y) {
    object->x = x;
    object->y = y;
}

// ACCESSORS
int z_bdeg_Point_fX(const z_bdeg_Point *object) {
    BSLS_ASSERT(object != 0);
    return object->x;
}
int z_bdeg_Point_fY(const z_bdeg_Point *object) {
    BSLS_ASSERT(object != 0);
    return object->y();
}

// FREE OPERATORS
int z_bdeg_Point_fAreEqual(const z_bdeg_Point *object,
                           const z_bdeg_Point *other) {
    BSLS_ASSERT(object != 0);
    BSLS_ASSERT(other != 0);
    return *object == *other;
}
} // ...
```

We must ensure
exceptions
don't leak out into
C code!

3. Total and Partial *Insulation* Techniques

Procedural Interface

```
// z_bdeg_point.h (continued)
```

```
// z_bdeg_point.cpp
#include <z_bdeg_point.h>
#include <bdeg_point.h>
#include <bsls_assert.h>
```

<- defensive programming

Clearly
Architecturally
Significant!

```
} // extern "C" linkage
```

```
#endif
```

```
#endif // internal include guard
```

```
BSLS_ASSERT(other);
return *object == *other;
} //
```

3. Total and Partial *Insulation* Techniques

Procedural Interface

Discussion?

3. Total and Partial *Insulation* Techniques

Examples of Partial Insulation

1. `const` Array Access

2. Toy Stack

3. Adaptive Memory Pool
(Simplified)

3. Total and Partial *Insulation* Techniques

const Array Access

```
// datautil.h
// ...
class DataUtil {
    static const int *s_data_p;
    static const int s_length;

public:
    static int data(int i) {
        return s_data_p[i];
    }

    static int dataLength() {
        return s_length;
    }
};

// ...
```

3. Total and Partial *Insulation* Techniques

const Array Access

```
// datautil.h
// ...
class DataUtil {
    static const int *s_data_p;
    static const int s_length;

public:
    static int data(int i) {
        return s_data_p[i];
    }

    static int dataLength() {
        return s_length;
    }
};

// ...
```

3. Total and Partial *Insulation* Techniques

const Array Access

```
// datautil.h
// ...
class DataUtil {
    static const int *s_data_p;
    static const int s_length;

public:
    static int data(int i) {
        return s_data_p[i];
    }

    static int dataLength() {
        return s_length;
    }
};

// ...
```

3. Total and Partial *Insulation* Techniques

const Array Access

```
// datautil.h
// ...
class DataUtil {
    static const int *s_data_p;
    static const int s_length;

public:
    static int data(int i) {
        return s_data_p[i];
    }

    static int dataLength() {
        return s_length;
    }
};

// ...
```

```
// datautil.cpp
#include <datautil.h>
// ...

static const int DATA[] =
{ 2, 5, 11, 17, 23, 31,
  41, 47, 59, 67, 89, 101
};

const int *
DataUtil::s_data_p = DATA;

const int
DataUtil::s_length =
    sizeof DATA / sizeof *DATA;
```

3. Total and Partial *Insulation* Techniques

const Array Access

```
// datautil.h
// ...
class DataUtil {
    static const int *s_data_p;
    static const int s_length;

public:
    static int data(int i) {
        return s_data_p[i];
    }

    static int dataLength() {
        return s_length;
    }
};

// ...
```

```
// datautil.cpp
#include <datautil.h>
// ...

static const int DATA[] =
{ 2, 5, 11, 17, 23, 31,
  41, 47, 59, 67, 89, 101
};

const int *
DataUtil::s_data_p = DATA;

const int
DataUtil::s_length =
    sizeof DATA / sizeof *DATA;
```

3. Total and Partial *Insulation* Techniques

const Array Access

```
// datautil.h
// ...
class DataUtil {
    static const int *s_data_p;
    static const int s_length;

public:
    static int data(int i) {
        return s_data_p[i];
    }

    static int dataLength() {
        return s_length;
    }
};

// ...
```

```
// datautil.cpp
#include <datautil.h>
// ...

static const int DATA[] =
{ 2, 5, 11, 17, 23, 31,
  41, 47, 59, 67, 89, 101
};

const int *
DataUtil::s_data_p = DATA;

const int
DataUtil::s_length =
    sizeof DATA / sizeof *DATA;
```

3. Total and Partial *Insulation* Techniques

const Array Access

```
// datautil.h
// ...
class DataUtil {
    static const int *s_data_p;
static const int s_length;

public:
    static int data(int i) {
        return s_data_p[i];
    }

    static int dataLength() {
        return s_length;
    }
};

// ...
```

```
// datautil.cpp
#include <datautil.h>
// ...

static const int DATA[] =
{ 2, 5, 11, 17, 23, 31,
  41, 47, 59, 67, 89, 101
};

const int *
DataUtil::s_data_p = DATA;

const int
DataUtil::s_length =
    sizeof DATA / sizeof *DATA;
```

3. Total and Partial *Insulation* Techniques

const Array Access

```
// datautil.h
// ...
class DataUtil {
    static const int *s_data_p;
    static const int s_length;

public:
    static int data(int i) {
        return s_data_p[i];
    }

    static int dataLength() {
        return s_length;
    }
};

// ...
```

```
// datautil.cpp
#include <datautil.h>
// ...

static const int DATA[] =
{ 2, 5, 11, 17, 23, 31,
  41, 47, 59, 67, 89, 101
};

const int *
DataUtil::s_data_p = DATA;

const int
DataUtil::s_length =
    sizeof DATA / sizeof *DATA;
```

3. Total and Partial *Insulation* Techniques

const Array Access

```
// datautil.h
// ...
class DataUtil {
    static const int *s_data_p;
    static const int s_length;

public:
    static int data(int i) {
        return s_data_p[i];
    }

    static int dataLength() {
        return s_length;
    }
};

// ...
```

```
// datautil.cpp
#include <datautil.h>
// ...
static const int DATA[] =
{ 2, 5, 11, 17, 23, 31,
  41, 47, 59, 67, 89, 101
};

const int *
DataUtil::s_data_p = DATA;

const int
DataUtil::s_length =
    sizeof DATA / sizeof *DATA;
```

3. Total and Partial *Insulation* Techniques

const Array Access

```
// datautil.h
// ...
class DataUtil {
    static const int *s_data_p;
    static const int s_length;

public:
    static int data(int i) {
        return s_data_p[i];
    }

    static int dataLength() {
        return s_length;
    }
};

// ...
```

```
// datautil.cpp
#include <datautil.h>
// ...
static const int DATA[] =
{ 2, 5, 11, 17, 23, 31,
  41, 47, 59, 67, 89, 101
};

const int *
DataUtil::s_data_p = DATA;

const int
DataUtil::s_length =
    sizeof DATA / sizeof *DATA;
```

3. Total and Partial *Insulation* Techniques

const Array Access

Insulated
Change!

```
// datautil.h
// ...
class DataUtil {
    static const int *s_data_p;
    static const int s_length;

public:
    static int data(int i) {
        return s_data_p[i];
    }

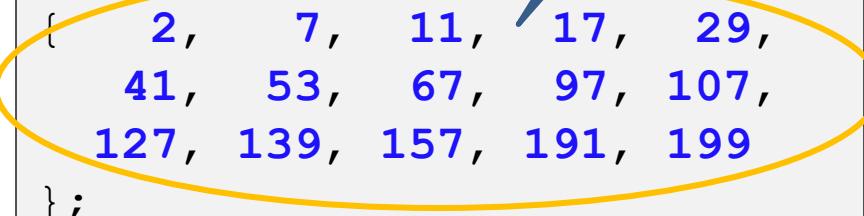
    static int dataLength() {
        return s_length;
    }
};

// ...
```

```
// datautil.cpp
#include <datautil.h>
// ...
static const int DATA[] =
{
    2,    7,    11,   17,   29,
    41,   53,   67,   97,   107,
    127,  139,  157,  191,  199
};

const int *
DataUtil::s_data_p = DATA;

const int
DataUtil::s_length =
    sizeof DATA / sizeof *DATA;
```



3. Total and Partial *Insulation* Techniques

const Array Access

```
// datautil.h
// ...
class DataUtil {
    static const int s_data_p;
    static const int s_length;

public:
    static int data(int i) {
        return s_data_p[i];
    }

    static int dataLength() {
        return s_length;
    }
};

// ...
```

Client
Does Not
Recompile!

```
// datautil.cpp
#include <datautil.h>
// ...
static const int DATA[] =
{
    2,    7,    11,   17,   29,
    41,   53,   67,   97,   107,
    127,  139,  157,  191,  199
};

const int *
DataUtil::s_data_p = DATA;

const int
DataUtil::s_length =
    sizeof DATA / sizeof *DATA;
```

Insulated
Change!

3. Total and Partial *Insulation* Techniques
const Array Access

Discussion?

3. Total and Partial *Insulation* Techniques

Toy Stack

```
// stack.h
// ...
#include <toy.h>
class Stack {
    Toy           *d_stack_p;
    std::size_t    d_capacity;
    std::size_t    d_length;
private:
    void resize();
public:
    Stack();
    // ...
    void push(const Toy& v);
    // ...
};
```

3. Total and Partial *Insulation* Techniques

Toy Stack

```
// stack.h
// ...
#include <toy.h>
class Stack {
    Toy           *d_stack_p;
    std::size_t    d_capacity;
    std::size_t    d_length;
private:
    void resize();
public:
Stack();
    // ...
    void push(const Toy& v);
    // ...
};

// ...
```

3. Total and Partial *Insulation* Techniques

Toy Stack

```
// stack.h
// ...
#include <toy.h>
class Stack {
    Toy           *d_stack_p;
    std::size_t    d_capacity;
    std::size_t    d_length;
private:
    void resize();
public:
    Stack();
    // ...
    void push(const Toy& v);
    // ...
};
```

```
// stack.cpp
#include <stack.h>
// ...
Stack::Stack()
: d_stack_p(0)
, d_capacity(0)
, d_length(0)
{ }
// ...
```

3. Total and Partial *Insulation* Techniques

Toy Stack

```
// stack.h
// ...
#include <toy.h>
class Stack {
    Toy *d_stack_p;
    std::size_t d_capacity;
    std::size_t d_length;
private:
    void resize();
public:
    Stack();
    // ...
    void push(const Toy& v);
    // ...
};
```

```
// stack.cpp
#include <stack.h>
// ...
Stack::Stack()
: d_stack_p(0)
, d_capacity(0)
, d_length(0)
{ }
```

3. Total and Partial *Insulation* Techniques

Toy Stack

```
// stack.h
// ...
#include <toy.h>
class Stack {
    Toy *d_stack_p;
    std::size_t d_capacity;
    std::size_t d_length;
private:
    void resize();
public:
    Stack();
    // ...
    void push(const Toy& v);
    // ...
};
```

```
// stack.cpp
#include <stack.h>
// ...
Stack::Stack()
: d_stack_p(0)
, d_capacity(0)
, d_length(0)
{ }
// ...
void
Stack::push(const Toy& v) {
    if (d_length == d_capacity)
        resize();
    new(d_stack_p+d_length) Toy(v);
    ++d_length;
};
```

3. Total and Partial *Insulation* Techniques

Toy Stack

```
// stack.h
// ...
#include <toy.h>
class Stack {
    Toy *d_stack_p;
    std::size_t d_capacity;
    std::size_t d_length;
private:
    void resize();
public:
    Stack();
    // ...
    void push(const Toy& v);
    // ...
};
```

```
// stack.cpp
#include <stack.h>
// ...
Stack::Stack()
: d_stack_p(0)
, d_capacity(0)
, d_length(0)
{ }
// ...
void
Stack::push(const Toy& v) {
    if (d_length == d_capacity)
        resize();
    new(d_stack_p+d_length) Toy(v);
    ++d_length;
}
```

inline

3. Total and Partial *Insulation* Techniques

Toy Stack

```
// stack.h
// ...
#include <toy.h>
class Stack {
    // ...
    void resize();
public:
    // ...
    void push(const Toy& v) {
        if (d_length == d_capacity)
            resize();
        new(d_stack_p+d_length) Toy(v);
        ++d_length;
    }
};
```

```
// ...
```

3. Total and Partial *Insulation* Techniques

Toy Stack

```
// stack.h
// ...
#include <toy.h>
class Stack {
    // ...
    void resize();
public:
    // ...
    void push(const Toy& v) {
        if (d_length == d_capacity)
            resize();
        new(d_stack_p+d_length) Toy(v);
        ++d_length;
    }
};
```

```
// ...
```

3. Total and Partial *Insulation* Techniques

Toy Stack

```
// stack.h
// ...
#include <toy.h>
class Stack {
    // ...
    void resize();
public:
    // ...
    void push(const Toy& v) {
        if (d_length == d_capacity)
            resize();
        new(d_stack_p+d_length) Toy(v);
        ++d_length;
    }
};
```

```
// ...
```

3. Total and Partial *Insulation* Techniques

Toy Stack

```
// stack.h
// ...
#include <toy.h>
class Stack {
    // ...
    void resize();
public:
    // ...
    void push(const Toy& v) {
        if (d_length == d_capacity)
            resize();
        new(d_stack_p+d_length) Toy(v);
        ++d_length;
    }
};
```

// ...



3. Total and Partial *Insulation* Techniques

Toy Stack

```
// stack.h
// ...
#include <toy.h>
class Stack {
    // ...
    void resize();
public:
    // ...
    void push(const Toy& v) {
        if (d_length == d_capacity)
            resize();
        new(d_stack_p+d_length) Toy(v);
        ++d_length;
    }
};
```

3. Total and Partial *Insulation* Techniques

Toy Stack

```
// stack.h
// ...
#include <toy.h>
class Stack {
    // ...
    void resize();
public:
    // ...
    void push(const Toy& v) {
        if (d_length == d_capacity)
            resize();
        new(d_stack_p+d_length) Toy(v);
        ++d_length;
    }
};
```

```
// ...
```

3. Total and Partial *Insulation* Techniques

Toy Stack

```
// stack.h
// ...
#include <toy.h>
class Stack {
    // ...
    void resize();
public:
    // ...
    void push(const Toy& v) {
        if (d_length == d_capacity)
            resize();
        new(d_stack_p+d_length) Toy(v);
        ++d_length;
    }
};
```

3. Total and Partial *Insulation* Techniques

Toy Stack

```
// stack.h
// ...
#include <toy.h>
class Stack {
    // ...
    void resize();
public:
    // ...
    void push(const Toy& v) {
        if (d_length == d_capacity)
            resize();
        new(d_stack_p+d_length) Toy(v);
        ++d_length;
    }
};
```

}

;

// ...

3. Total and Partial *Insulation* Techniques

Toy Stack

```
// stack.h
// ...
#include <toy.h>
class Stack {
    // ...
    void resize();
public:
    // ...
    void push(const Toy& v) {
        if (d_length == d_capacity)
            resize();
        new(d_stack_p+d_length) Toy(v);
        ++d_length;
    }
};
```

3. Total and Partial *Insulation* Techniques

Toy Stack

```
// stack.h
// ...
#include <toy.h>
class Stack {
    // ...
    void resize();
public:
    // ...
    void push(const Toy& v) {
        if (d_length == d_capacity)
            resize();
        new(d_stack_p+d_length) Toy(v);
        ++d_length;
    }
};
```



```
// ...
```

3. Total and Partial *Insulation* Techniques

Toy Stack

```
// stack.h
// ...
#include <toy.h>
class Stack {
    // ...
    void resize();
public:
    // ...
    void push(const Toy& v) {
        if (d_length == d_capacity)
            resize();
        new(d_stack_p+d_length) Toy(v);
        ++d_length;
    }
};
```

calling into .cpp file

```
// stack.cpp
#include <stack.h>

// ... (Include any extra header files that
// ... are needed.)

void Stack::resize() {
    std::size_t newCap =
        0 == d_capacity ? 1 : 2 * d_capacity;
    Toy *tmp = (Toy *)
        ::operator new(newCap * sizeof *tmp);
    NewDeleteProctor ndp(tmp);
    RangeProctor<Toy> rp(tmp);
    for (std::size_t i = 0;
         i < d_length; ++i, rp.advance())
        new(tmp+i) Toy(d_stack_p[i]);

    rp.release(); // Copy was successful.
    ndp.release();
    std::swap(d_stack_p, tmp); // +capacity
    d_capacity = newCap;
    for (std::size_t i = d_length; i;
         (tmp + --i)->~Toy());
    ::operator delete(tmp);
}
```

3. Total and Partial *Insulation* Techniques

Toy Stack

```
// stack.h
// ...
#include <toy.h>
class Stack {
    // ...
    void resize();
public:
    // ...
    void push(const Toy& v) {
        if (d_length == d_capacity)
            resize();
        new(d_stack_p+d_length) Toy(v);
        ++d_length;
    }
};
```

calling into .cpp file

```
// stack.cpp
#include <stack.h>

// ... (Include any extra header files that
// ... are needed.)

void Stack::resize() {
    std::size_t newCap =
        0 == d_capacity ? 1 : 2 * d_capacity;
    Toy *tmp = (Toy *)operator new(newCap * sizeof *tmp);
    NewDeleteProctor ndp(tmp);
    RangeProctor<Toy> rp(tmp);
    for (std::size_t i = 0;
         i < d_length; ++i, rp.advance())
        new(tmp+i) Toy(d_stack_p[i]);
    rp.release(); // Copy was successful.
    ndp.release();
    std::swap(d_stack_p, tmp); // +capacity
    d_capacity = newCap;
    for (std::size_t i = d_length; i;
         (tmp + --i)->~Toy());
    ::operator delete(tmp);
}
```

3. Total and Partial *Insulation* Techniques

Toy Stack

Insulated
Change!

```
// stack.h
// ...
#include <toy.h>
class Stack {
    // ...
    void resize();
public:
    // ...
    void push(const Toy& v) {
        if (d_length == d_capacity)
            resize();
        new(d_stack_p+d_length) Toy(v);
        ++d_length;
    }
};
```

calling into .cpp file

```
// stack.cpp
#include <stack.h>

// ... (Include any extra header files that
// ... are needed.)

void Stack::resize() {
    std::size_t newCap =
        0 == d_capacity ? 2 : 1.5*d_capacity;
    Toy *tmp = (Toy *)operator new(newCap * sizeof *tmp);
    NewDeleteProctor ndp(tmp);
    RangeProctor<Toy> rp(tmp);
    for (std::size_t i = 0;
         i < d_length; ++i, rp.advance())
        new(tmp+i) Toy(d_stack_p[i]);
    rp.release(); // Copy was successful.
    ndp.release();
    std::swap(d_stack_p, tmp); // +capacity
    d_capacity = newCap;
    for (std::size_t i = d_length; i;
         (tmp + --i)->~Toy());
    ::operator delete(tmp);
}
```

3. Total and Partial *Insulation* Techniques

Toy Stack

```
// stack.h
// ...
#include <toy.h>
class Stack {
    // ...
    void resize();
public:
    // ...
    void push(const Toy& v) {
        if (d_length == d_capacity)
            resize();
        new(d_stack_p+d_length) Toy(v);
        ++d_length;
    }
};
```

calling into .cpp file

```
// stack.cpp
#include <stack.h>

// ... (Include any extra header files that
// ... are needed.)

void Stack::resize() {
    std::size_t newCap =
        0 == d_capacity ? 2 : 1.5*d_capacity;
    Toy *tmp = (Toy *)
        ::operator new(newCap * sizeof *tmp);
    NewDeleteProctor ndp(tmp);
    RangeProctor<Toy> rp(tmp);
    for (std::size_t i = 0;
        i < d_length, ++i, rp.advance())
        new(tmp+i) Toy(d_stack_p[i]);
    rp.release(); // Copy was successful.
    ndp.release();
    std::swap(d_stack_p, tmp); // +capacity
    d_capacity = newCap;
    for (std::size_t i = d_length; i;
        (tmp + --i)->~Toy());
    ::operator delete(tmp);
}
```

3. Total and Partial *Insulation* Techniques

Toy Stack

```
// stack.h
// ...
#include <toy.h>
class Stack {
    // ...
    void resize();
public:
    // ...
    void push(const Toy& v) {
        if (d_length == d_capacity)
            resize();
        new(d_stack_p+d_length) Toy(v);
        ++d_length;
    }
};
```

calling into .cpp file

```
// stack.cpp
#include <stack.h>

// ... (Include any extra header files that
// ... are needed.)

void Stack::resize() {
    std::size_t newCap =
        0 == d_capacity ? 2 : 1.5*d_capacity;
    Toy *tmp = (Toy *)
        ::operator new(newCap * sizeof *tmp);
    NewDeleteProctor ndp(tmp);
    RangeProctor<Toy> rp(tmp);
    for (std::size_t i = 0;
        i < d_length, ++i, rp.advance())
        new(tmp+i) Toy(std::move(d_stack_p[i]));
    rp.release(); // Copy was successful.
    ndp.release();
    std::swap(d_stack_p, tmp); // +capacity
    d_capacity = newCap;
    for (std::size_t i = d_length;
        i < newCap, ++i)
        (tmp+i);
    ::operator delete(tmp);
}
```

Insulated
Optimization!

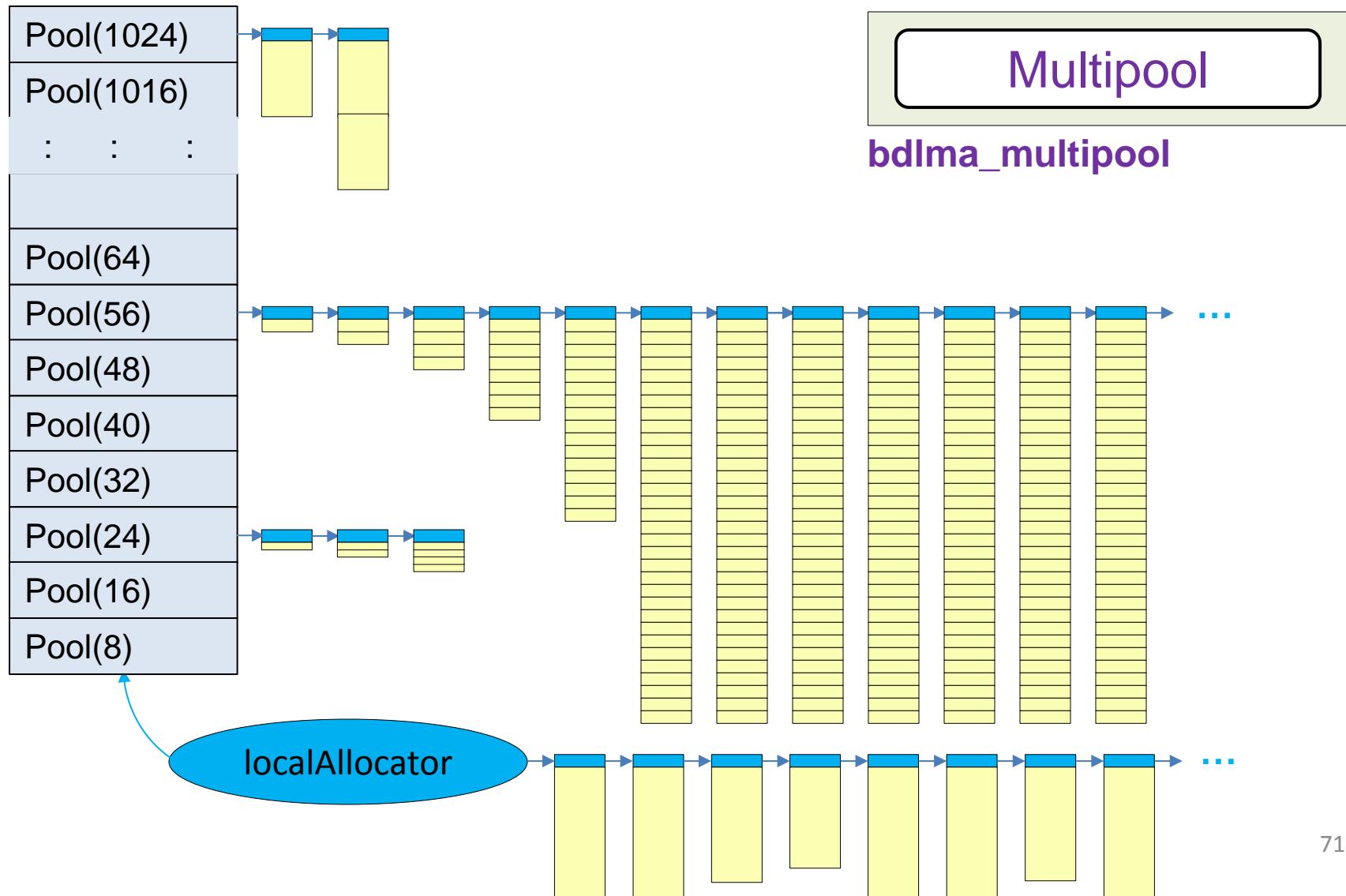
3. Total and Partial *Insulation* Techniques

Toy Stack

Discussion?

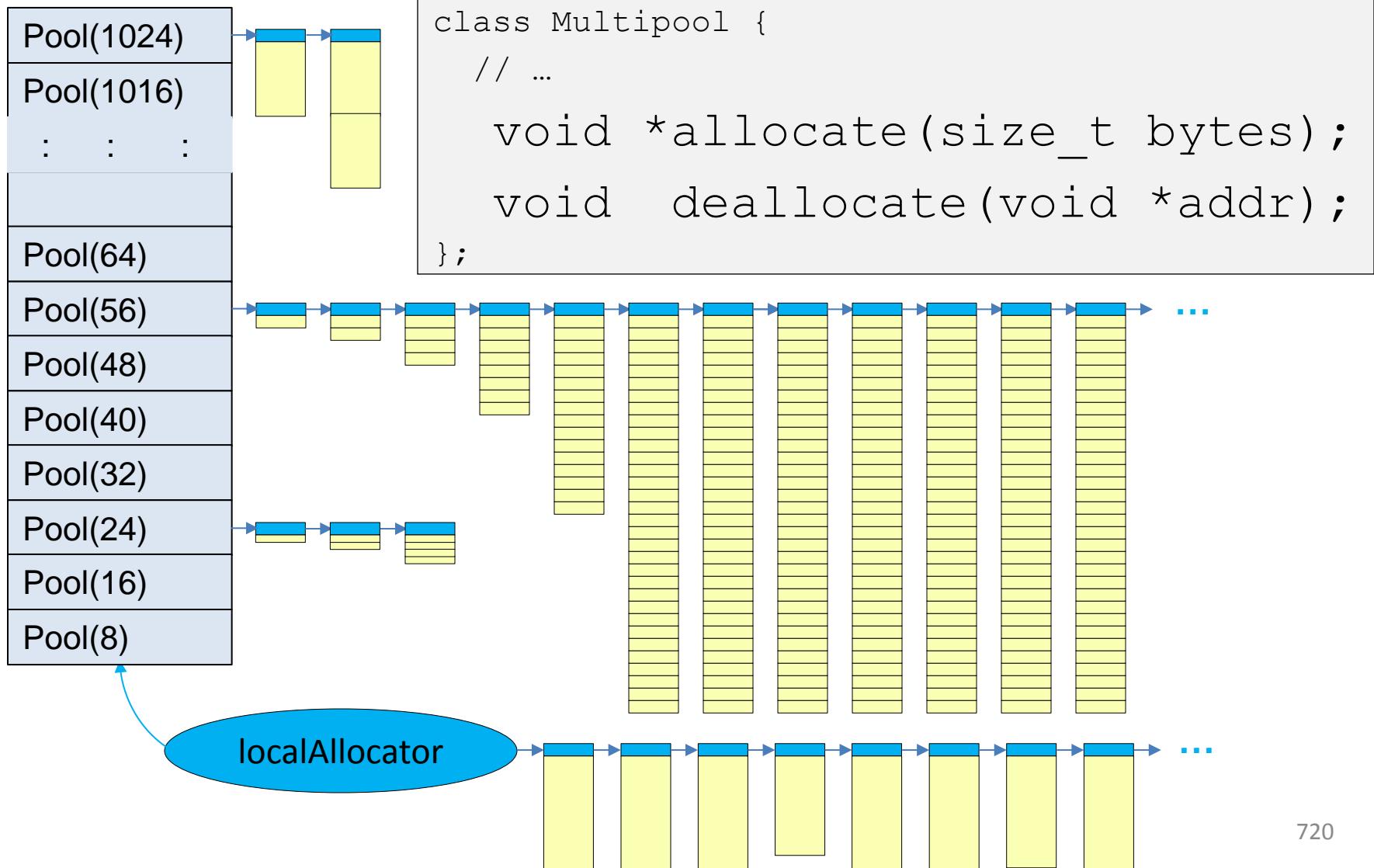
3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)



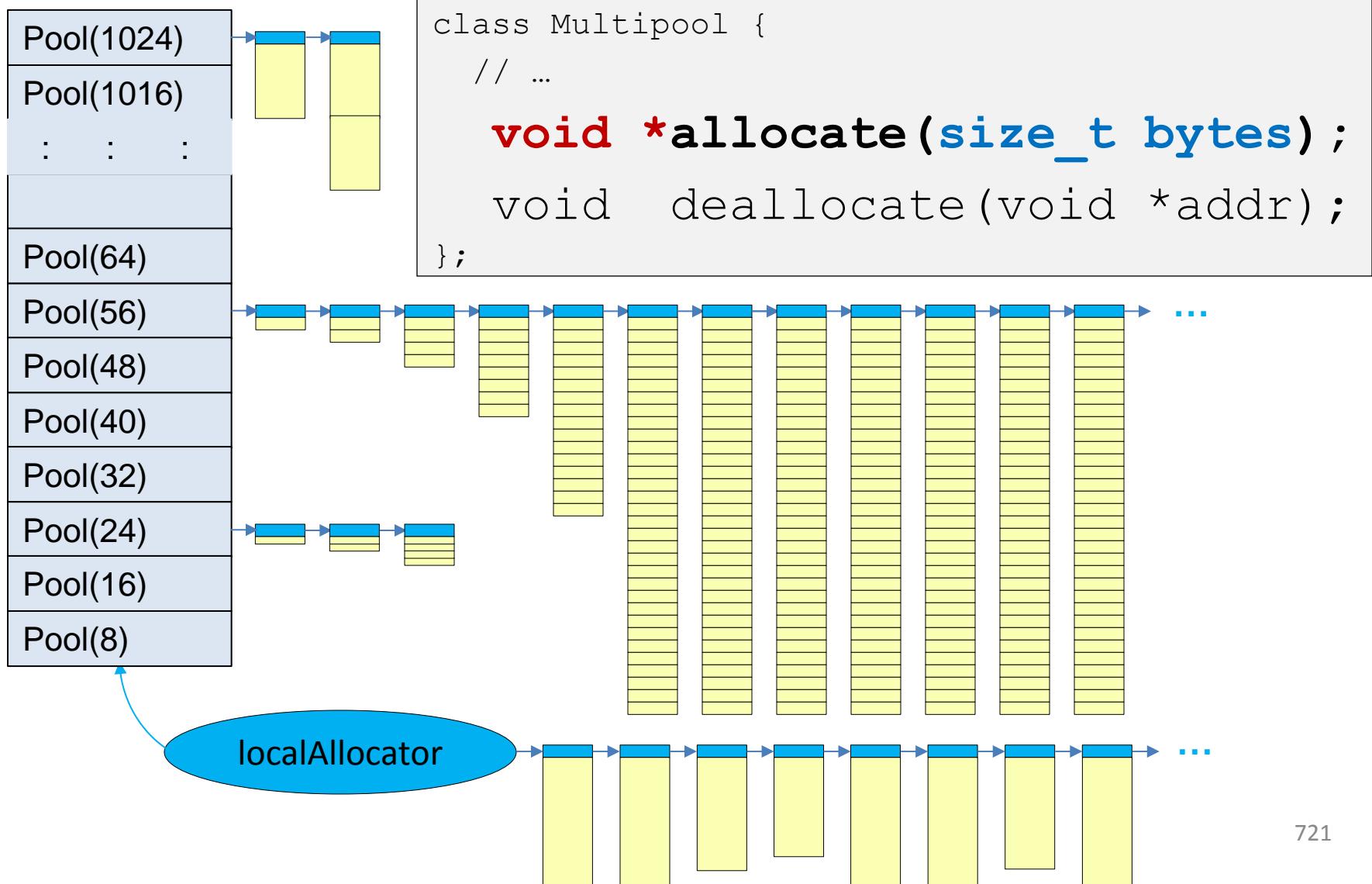
3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)



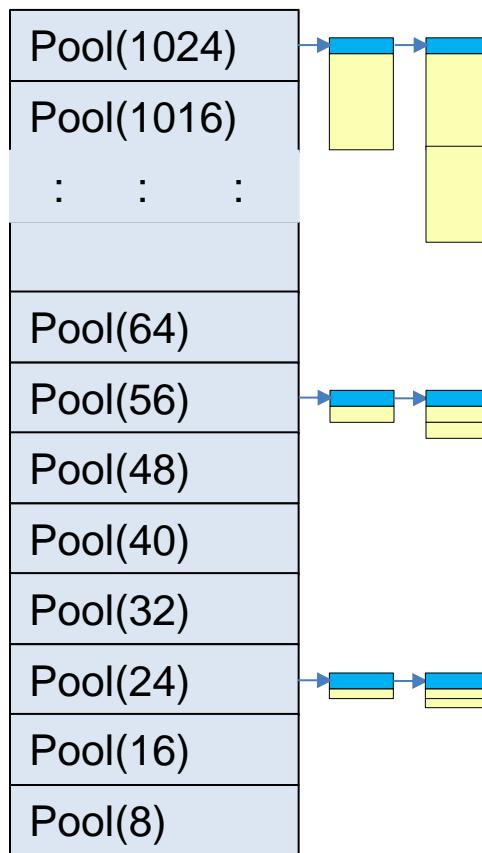
3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

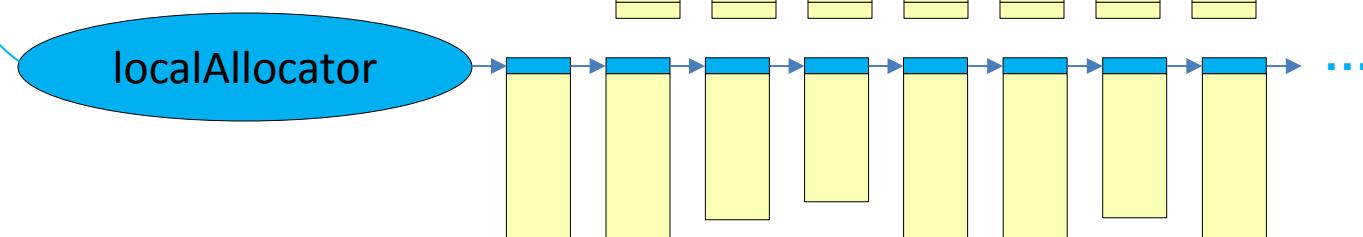


3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

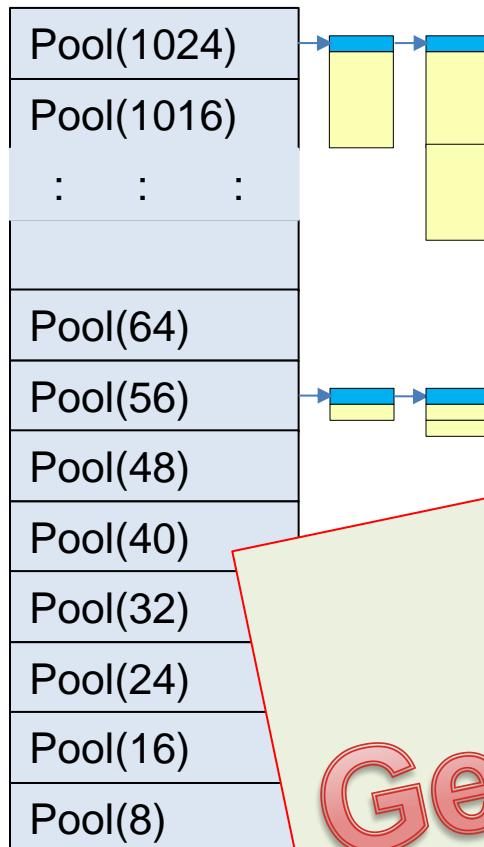


```
class Multipool {  
    // ...  
    void *allocate(size_t bytes);  
    void deallocate(void *addr);  
};
```



3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

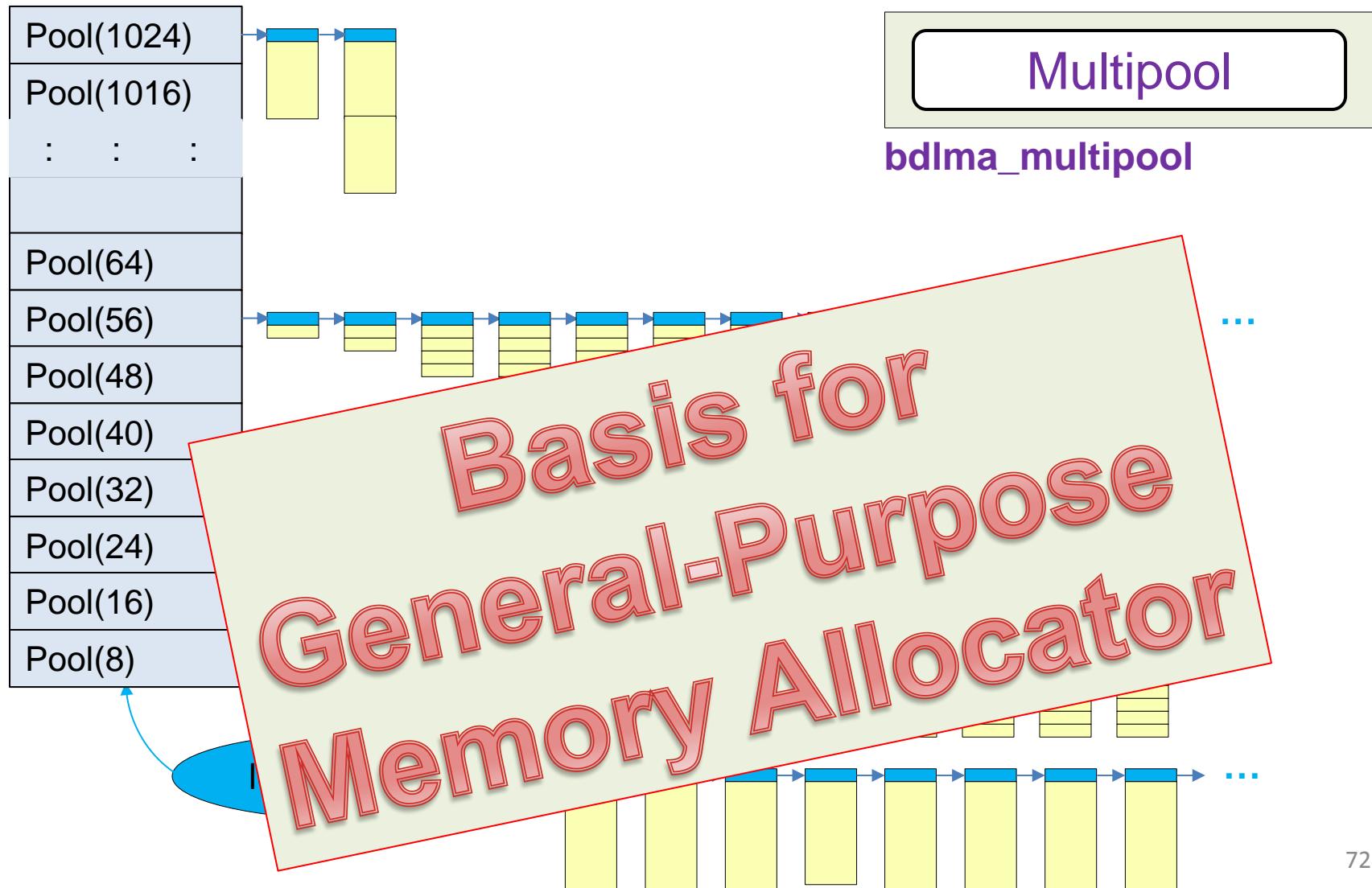


```
class Multipool {  
    // ...  
    void *allocate(size_t bytes);  
    void deallocate(void *addr);  
};
```

**Basis for
General-Purpose
Memory Allocator**

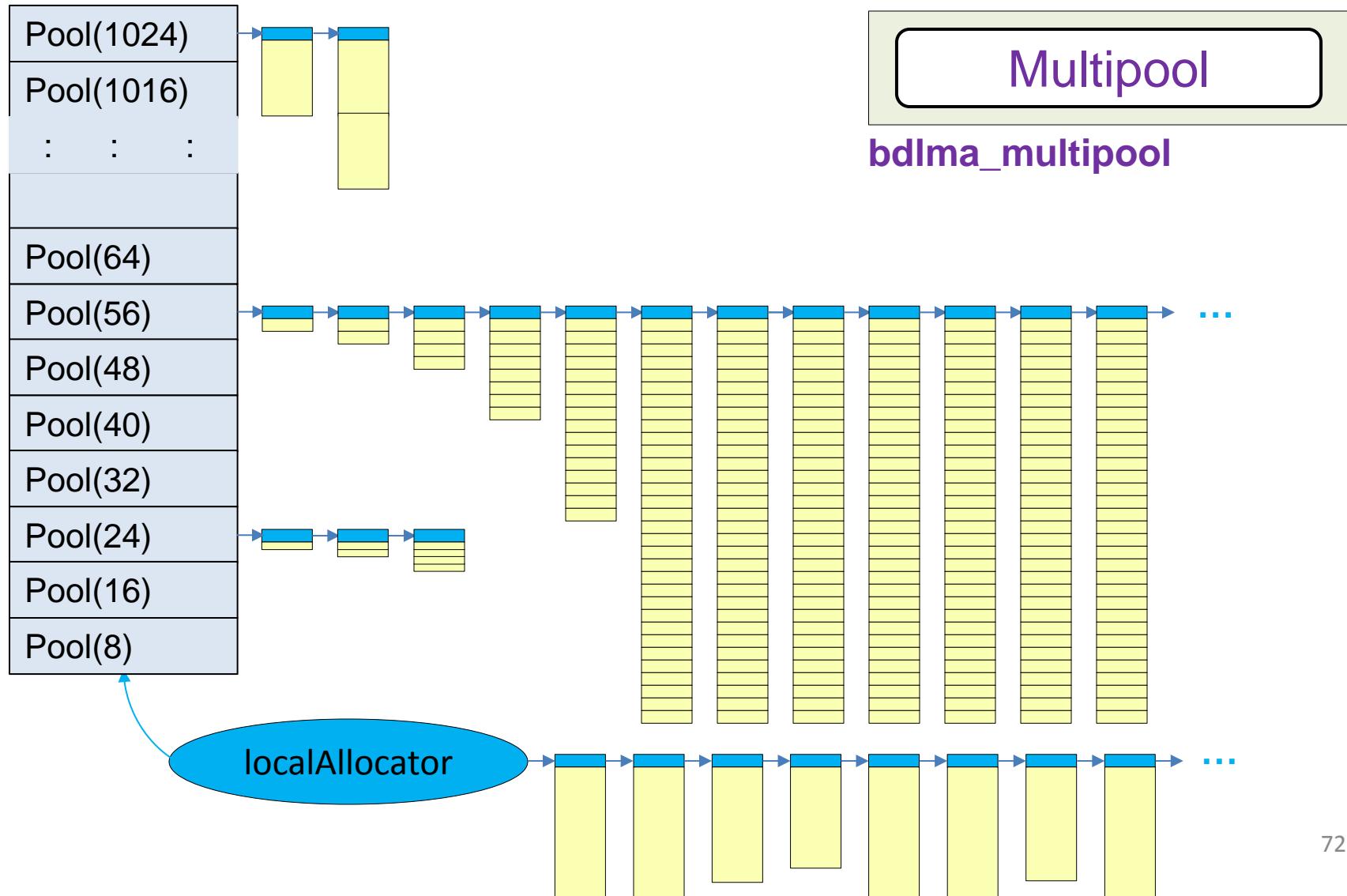
3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)



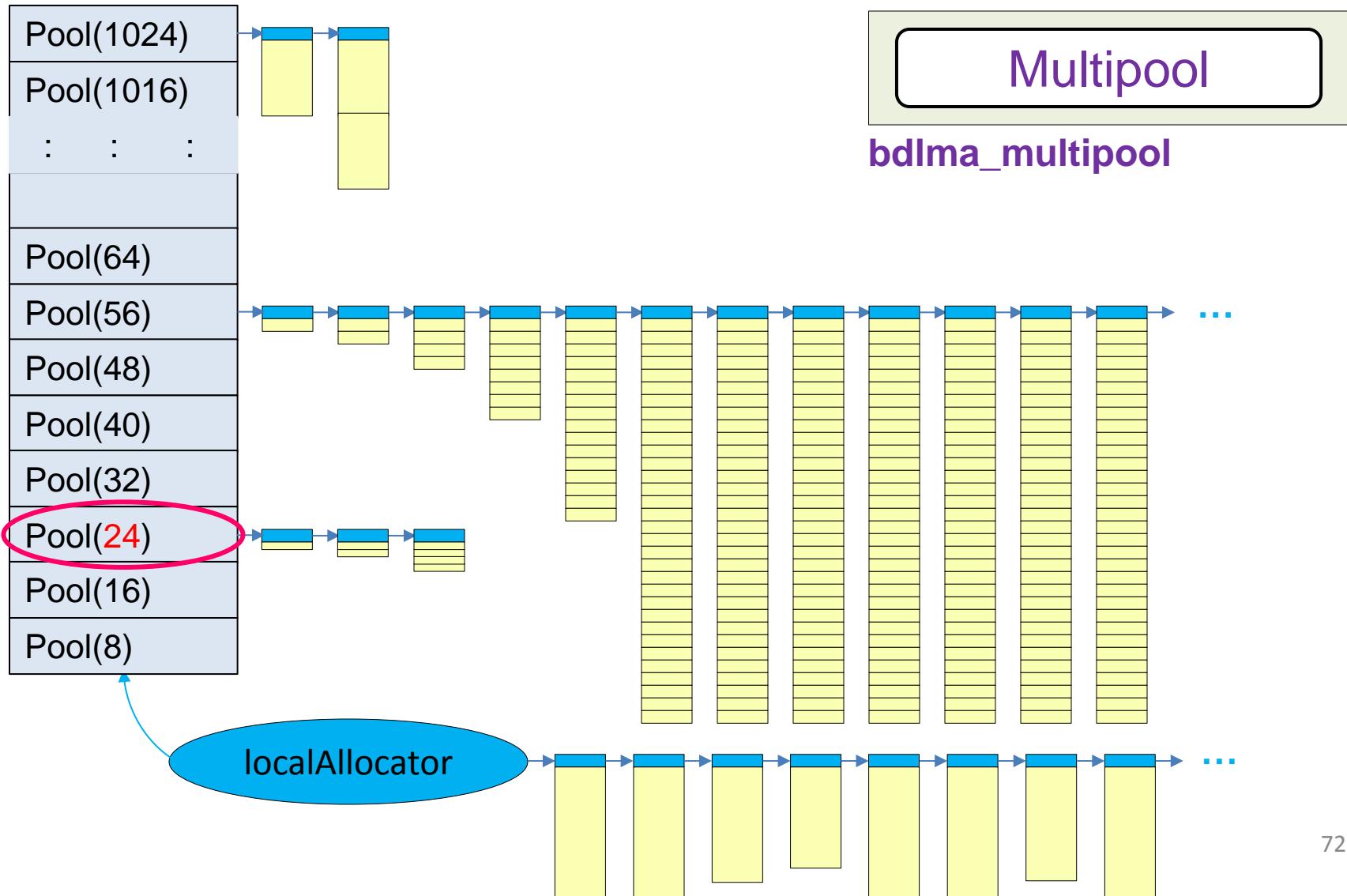
3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)



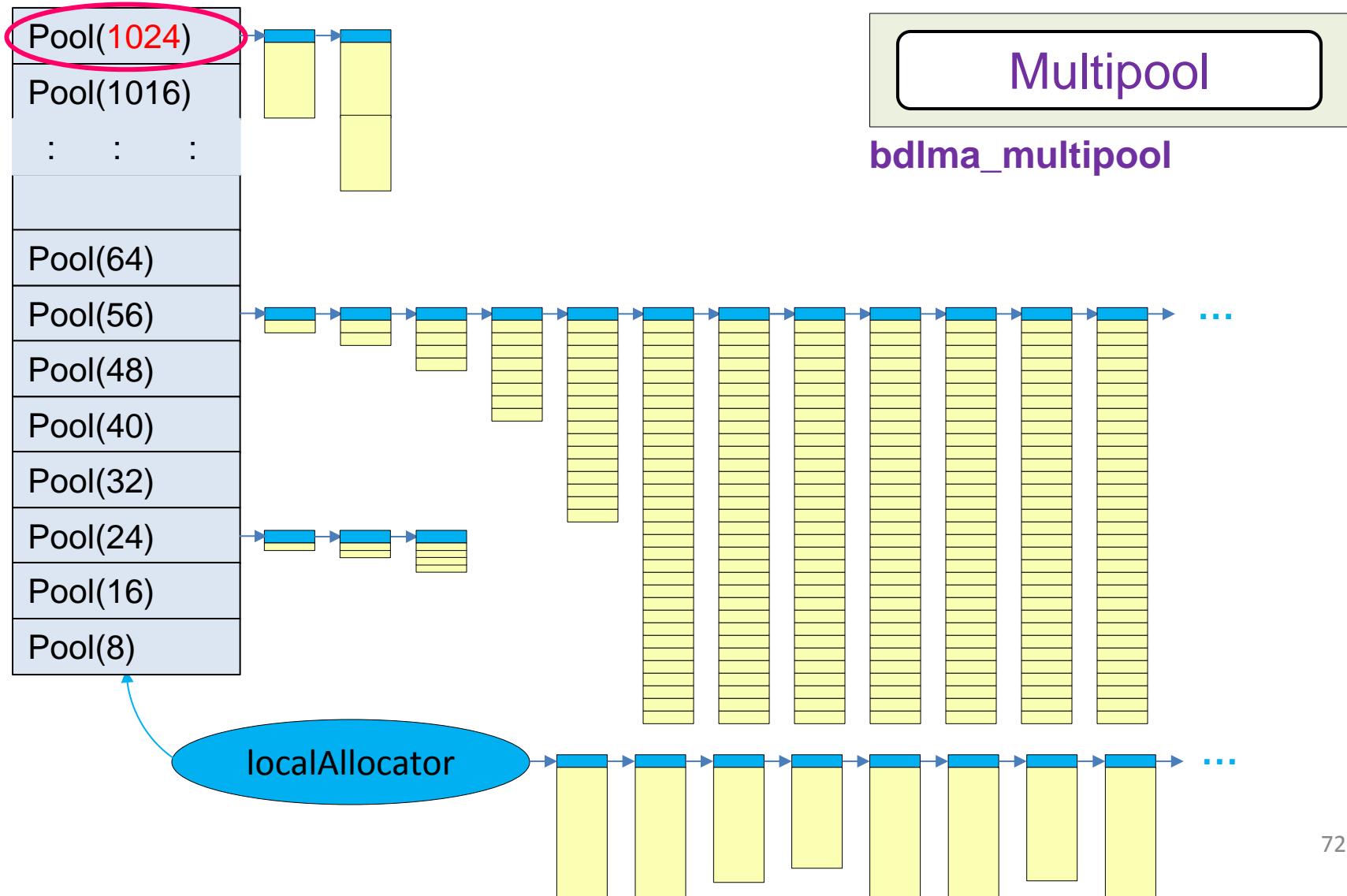
3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)



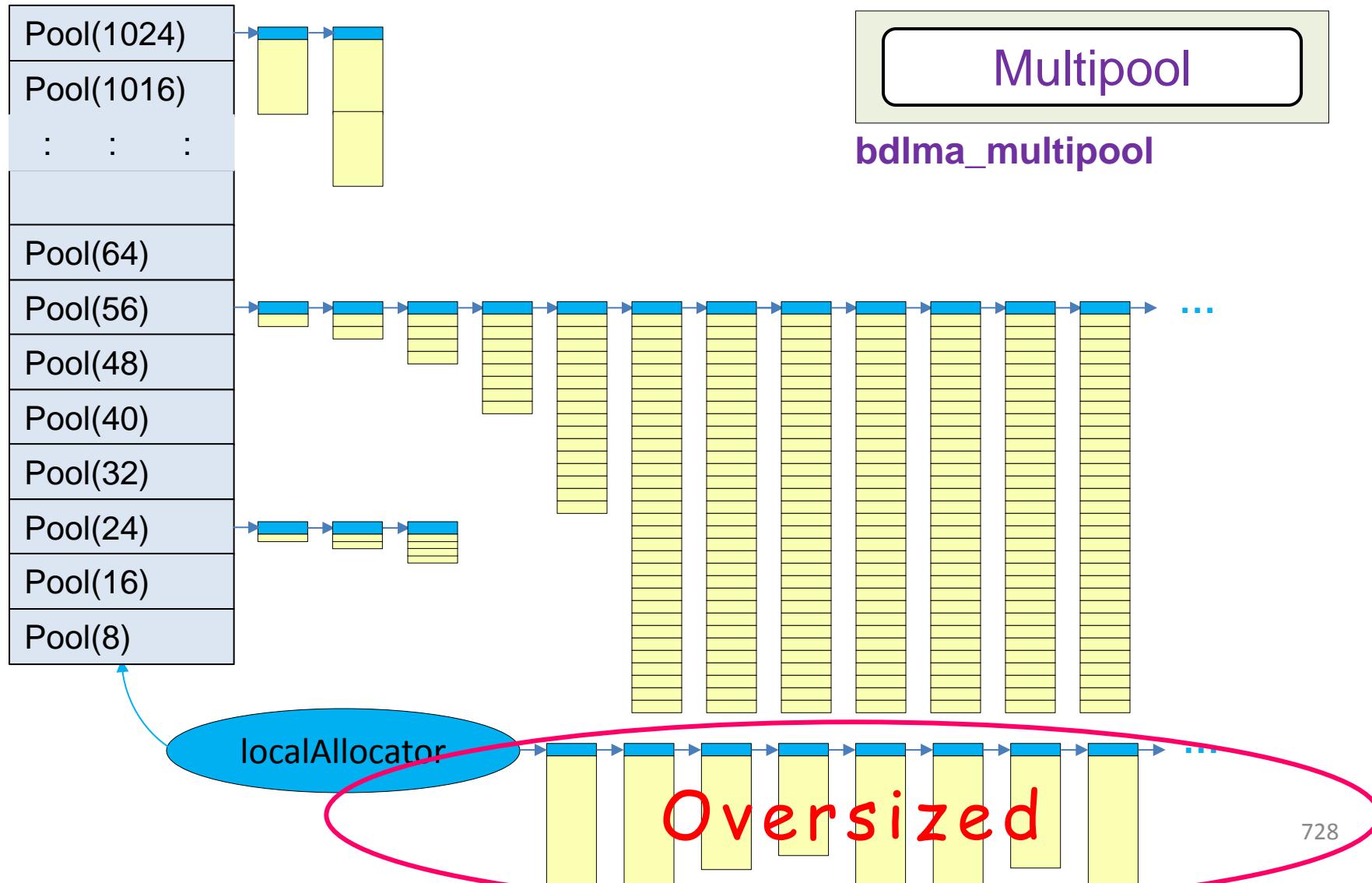
3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)



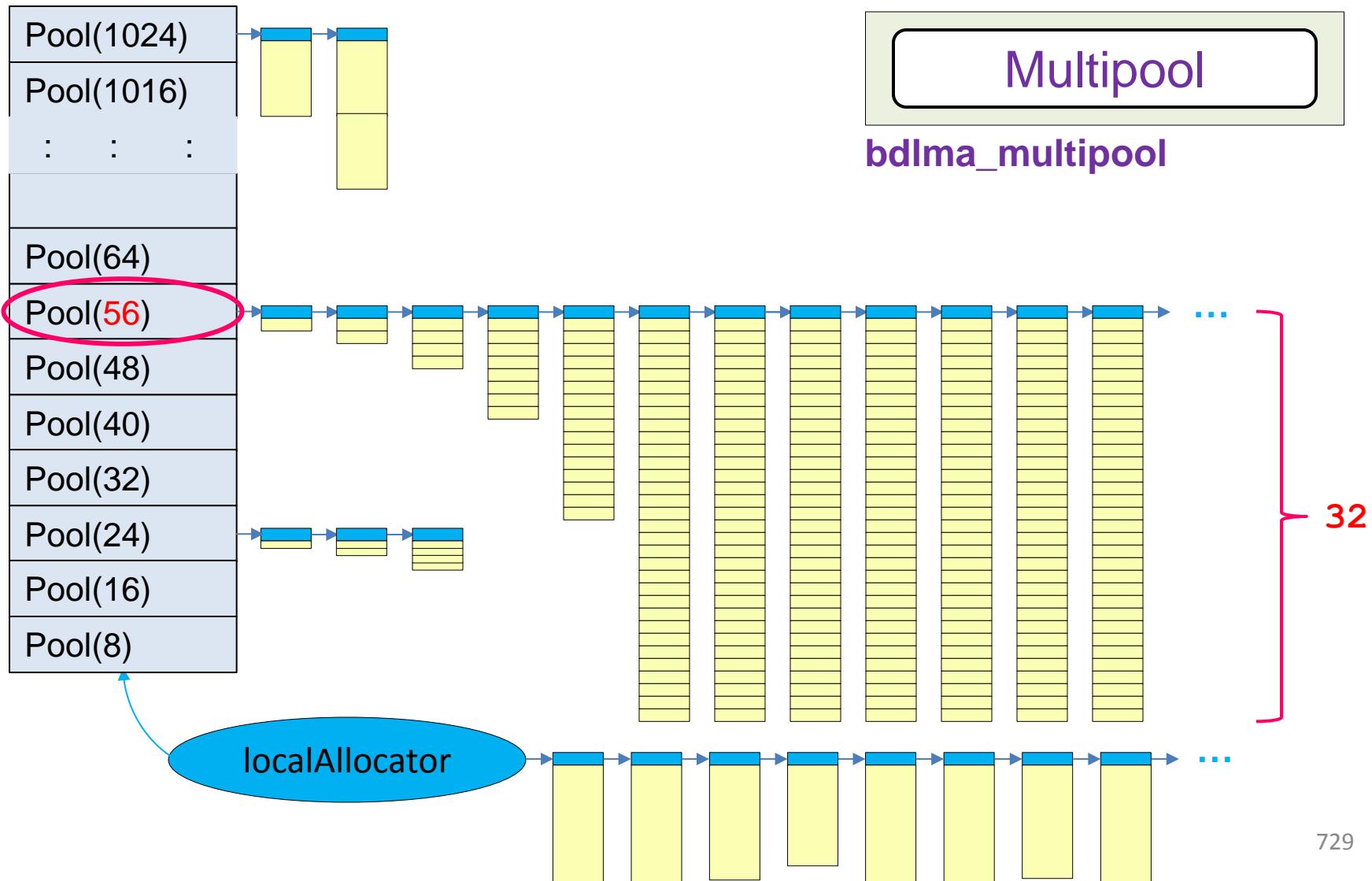
3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)



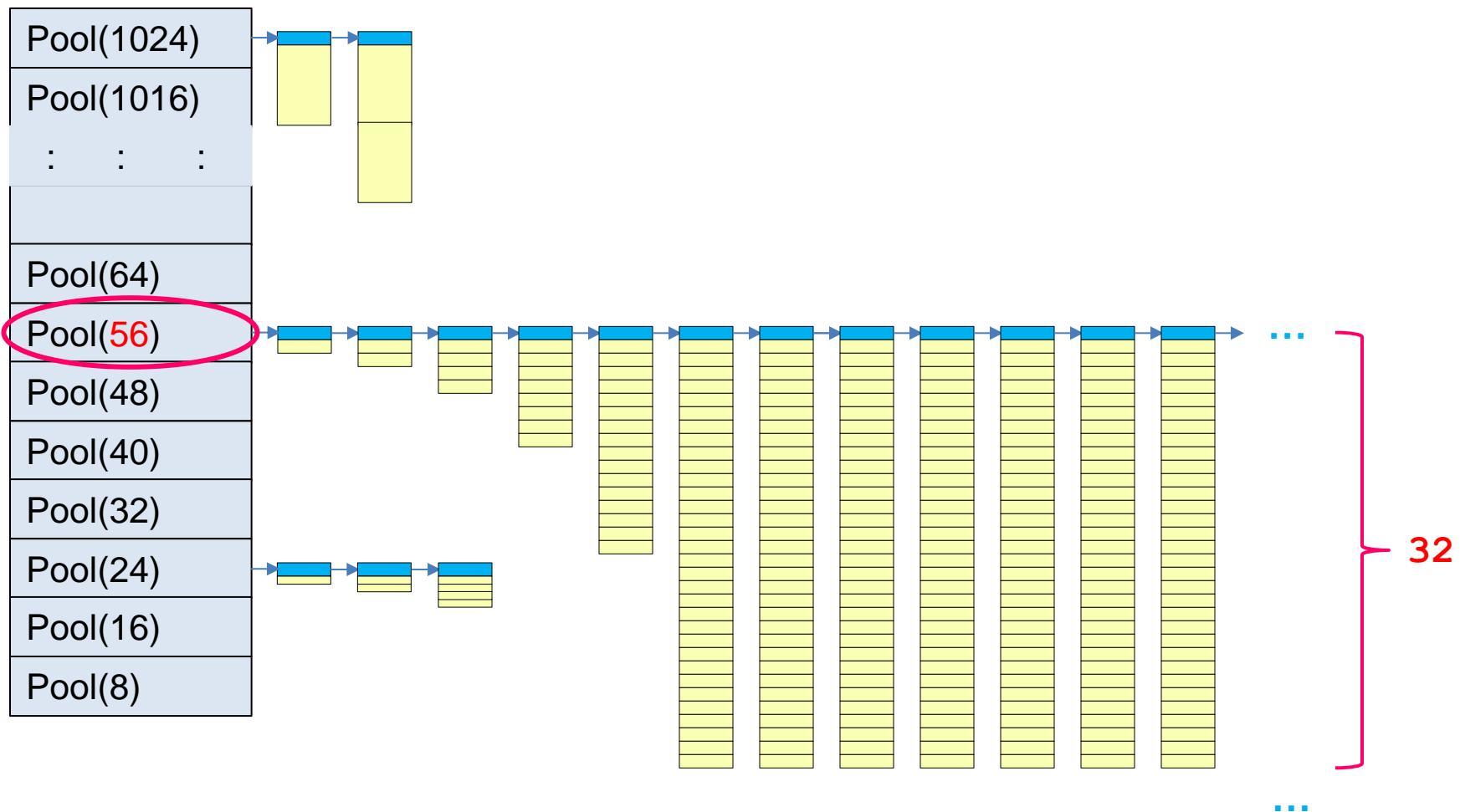
3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)



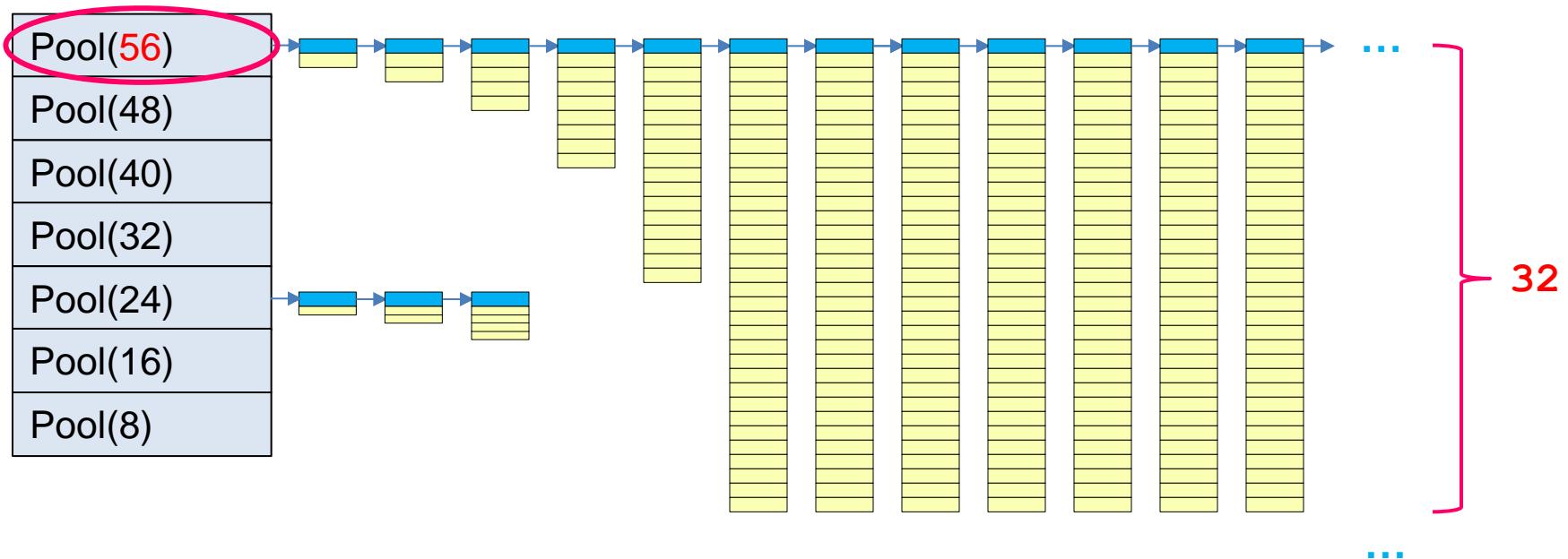
3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)



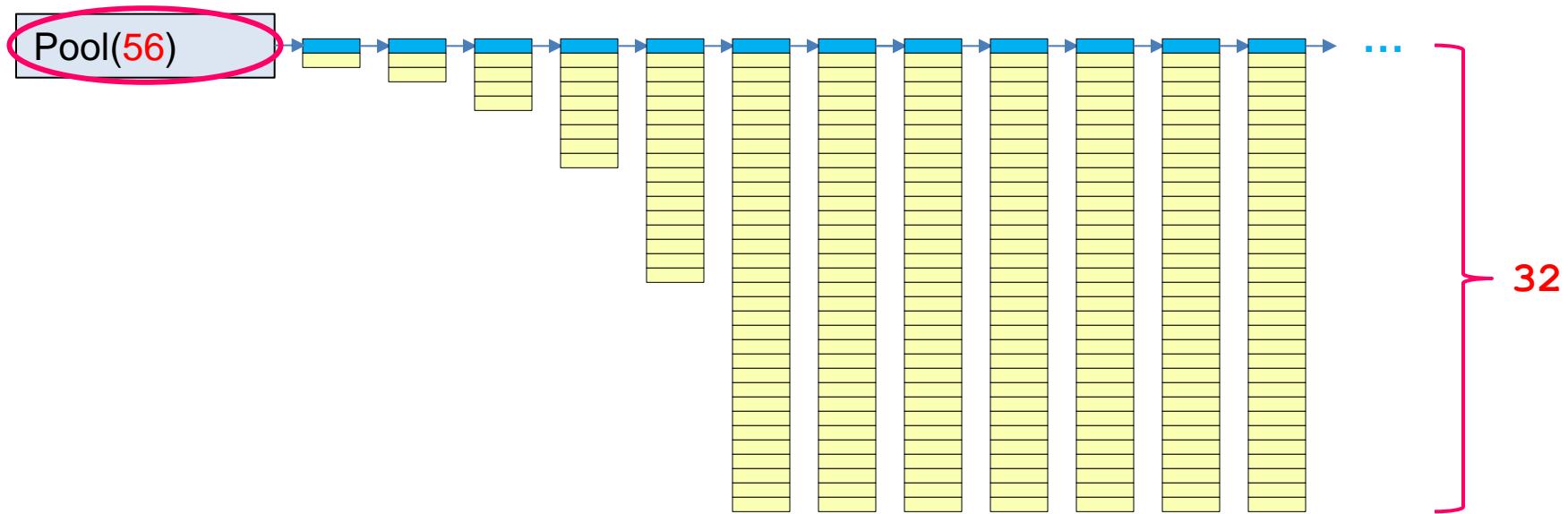
3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)



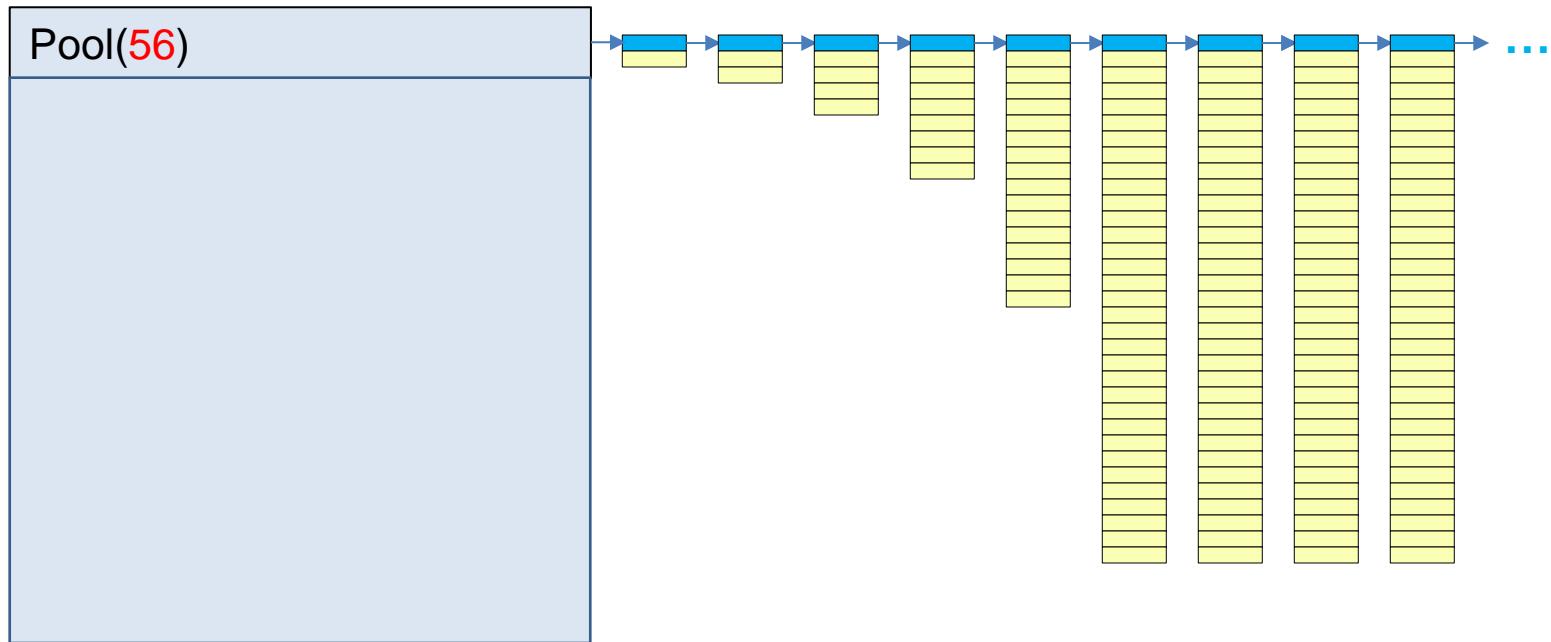
3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)



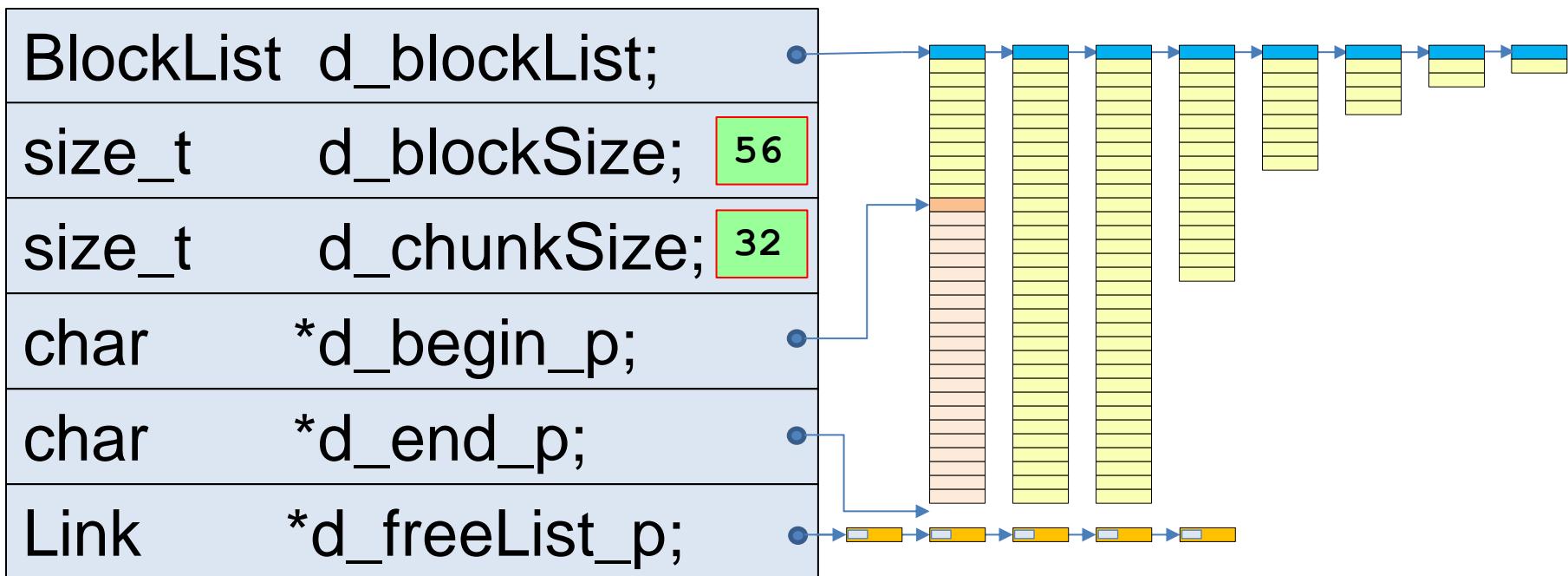
3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)



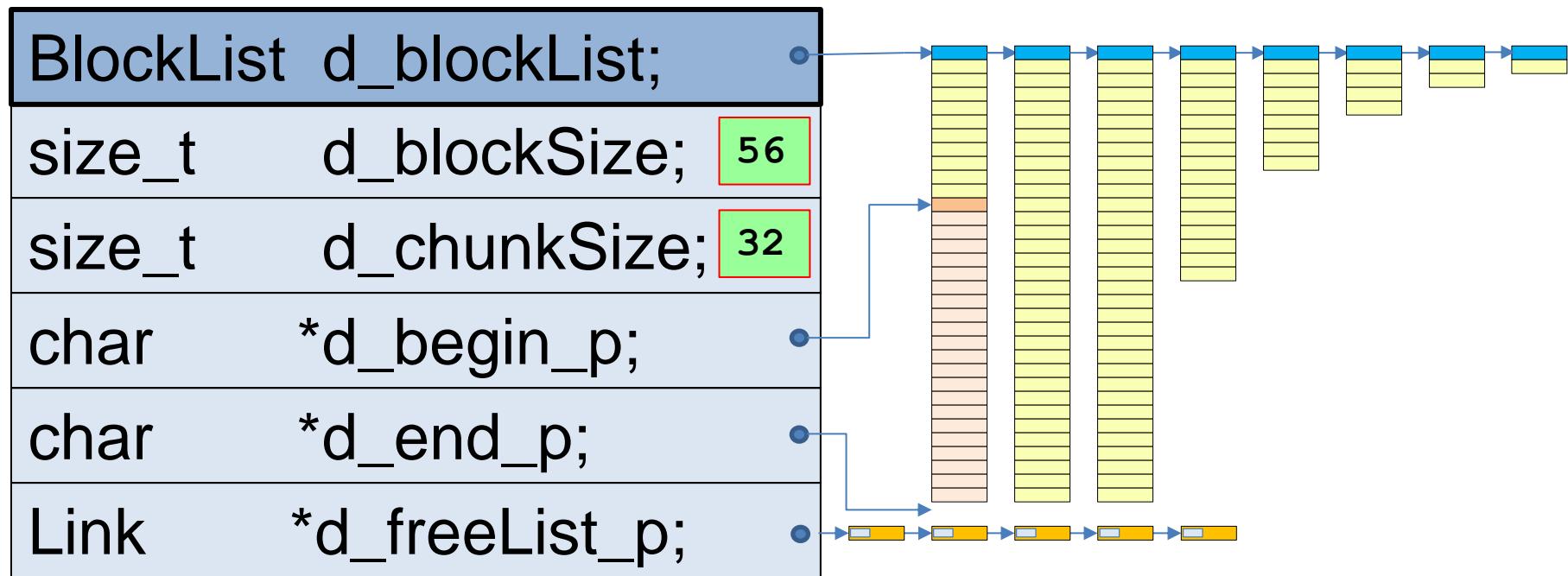
3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)



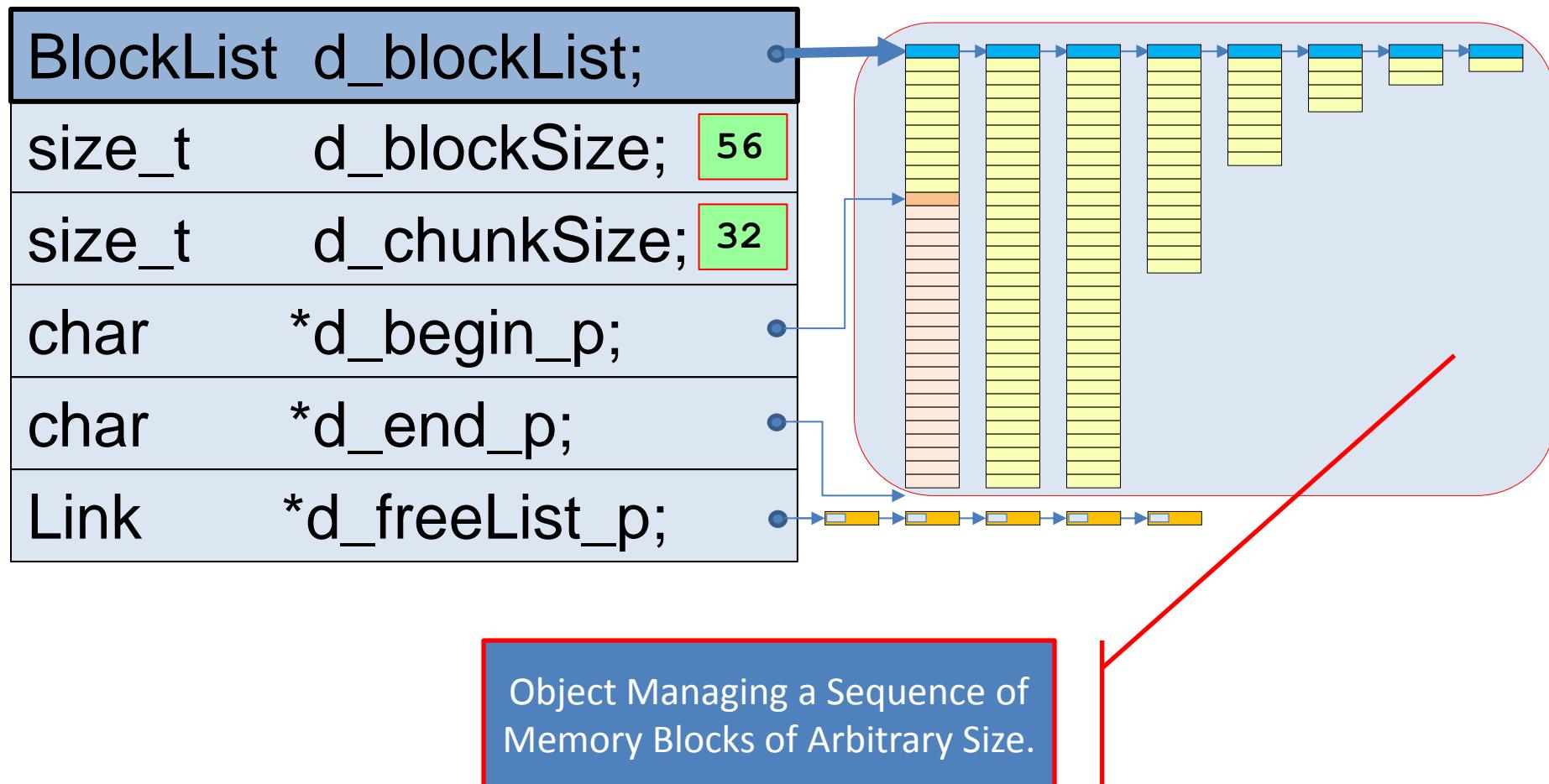
3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)



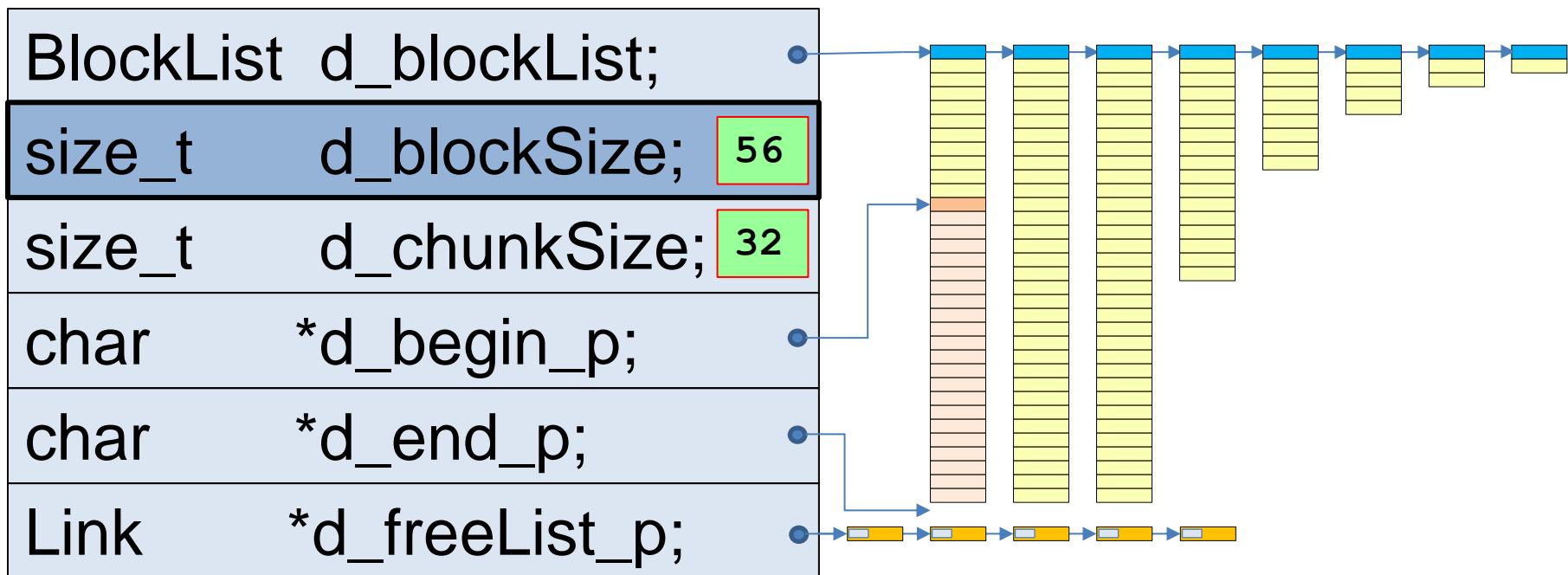
3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)



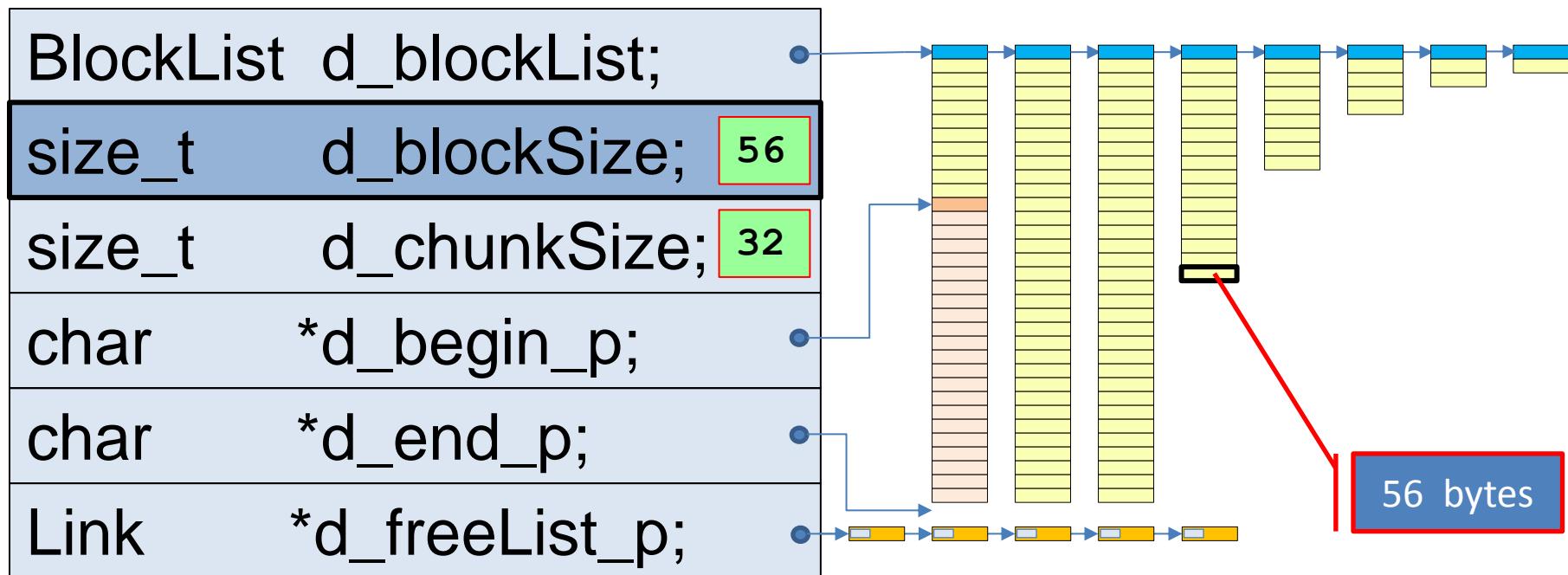
3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)



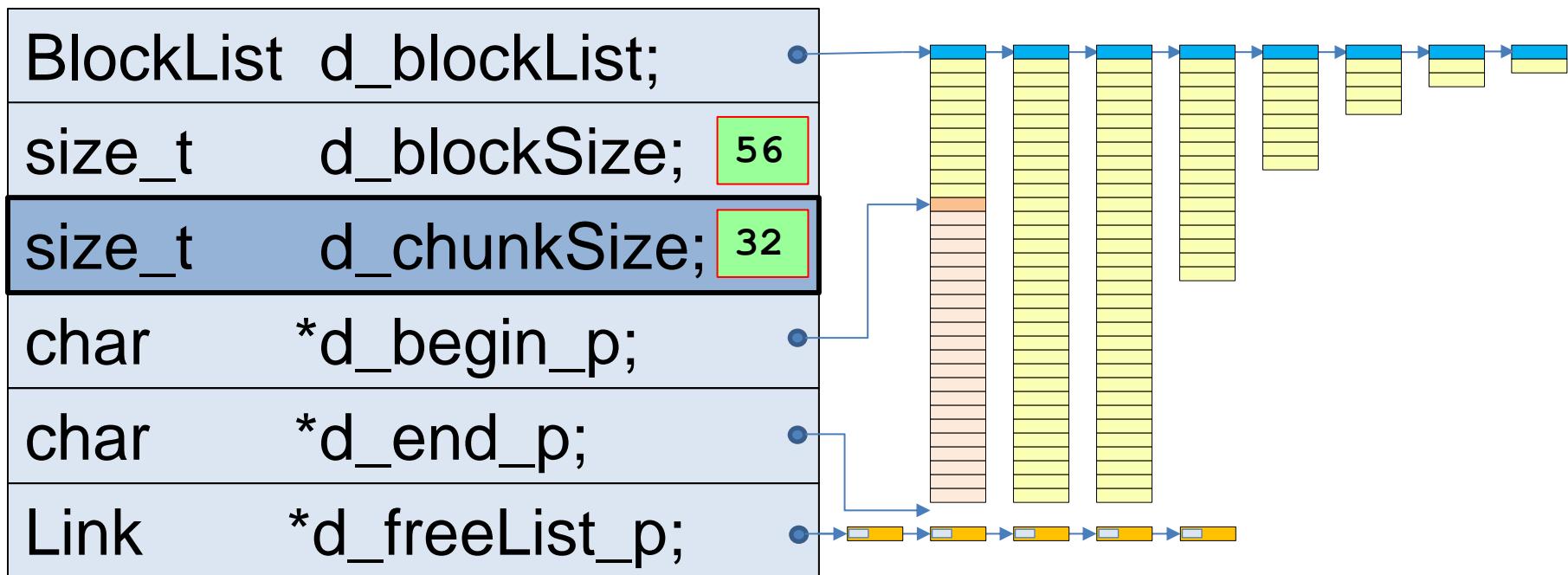
3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)



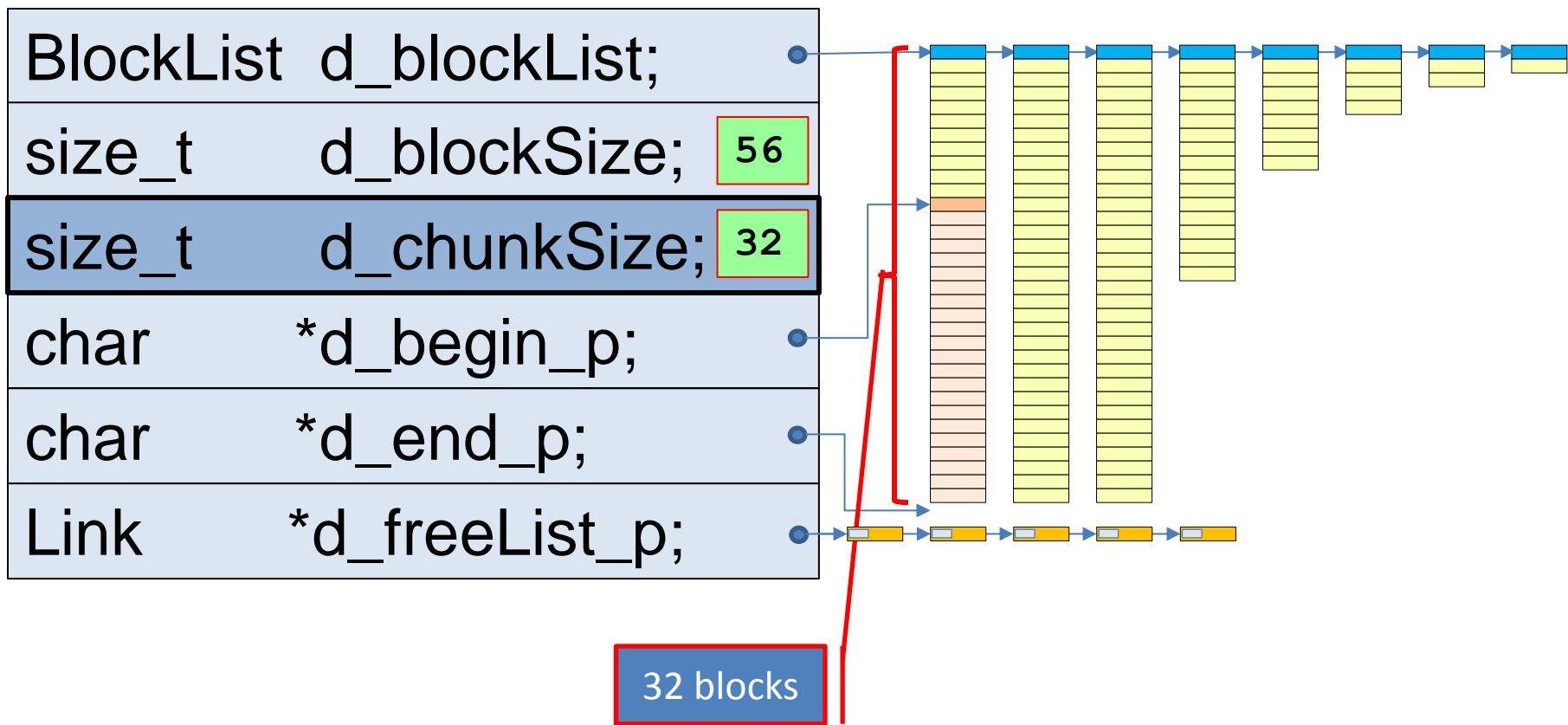
3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)



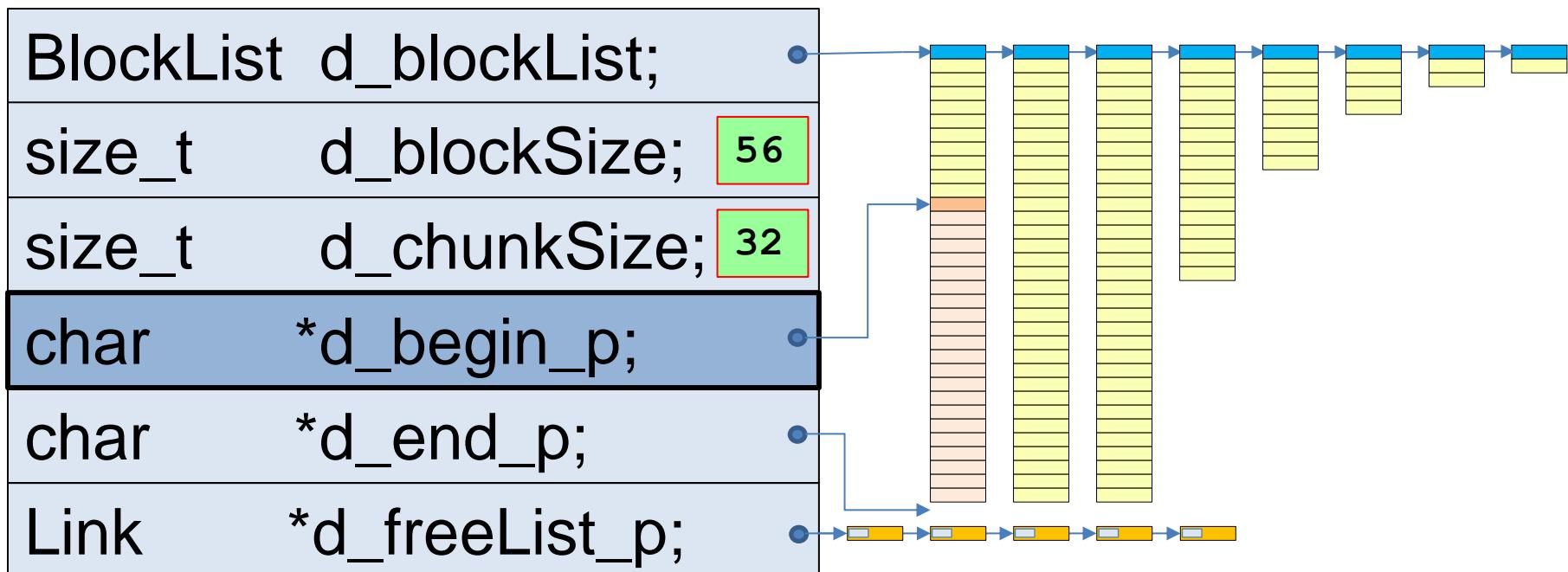
3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)



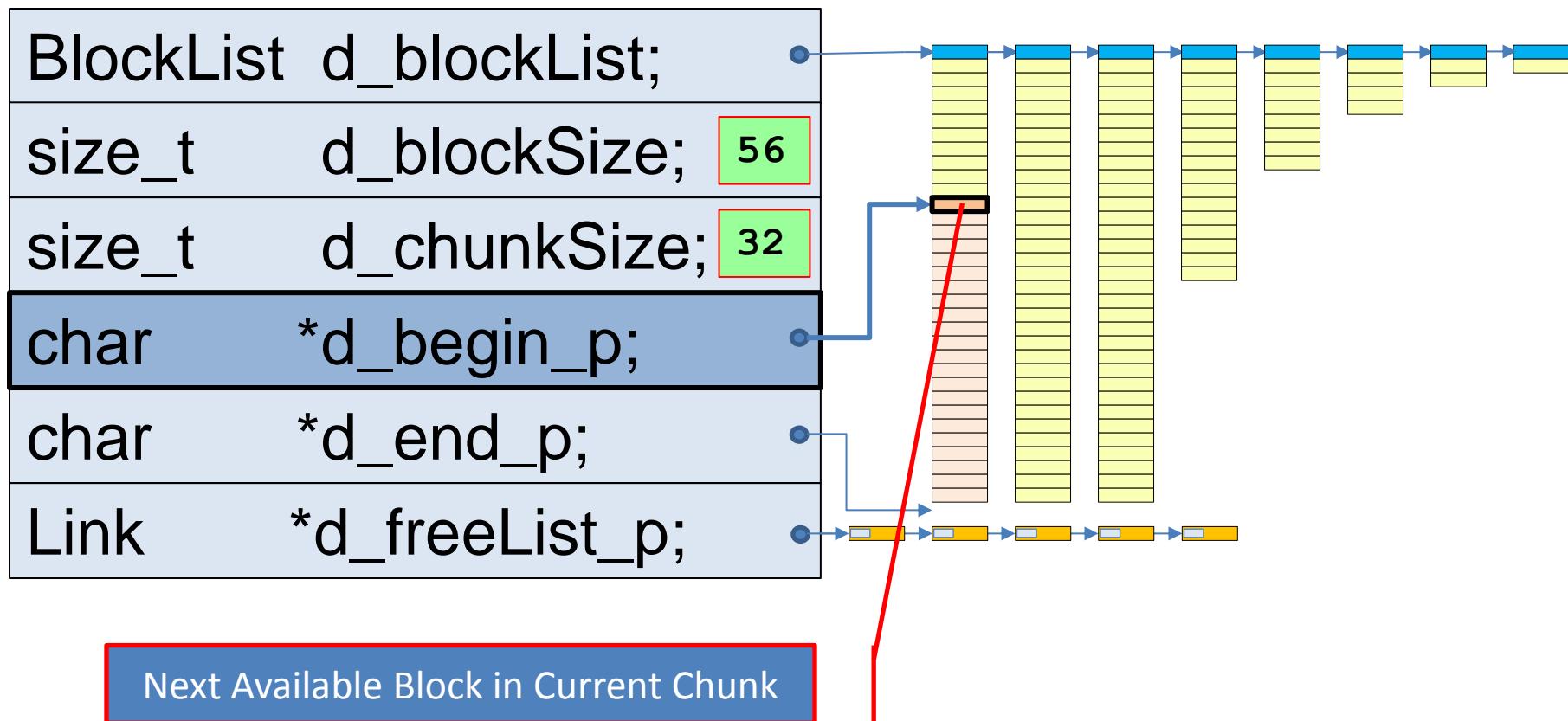
3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)



3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)



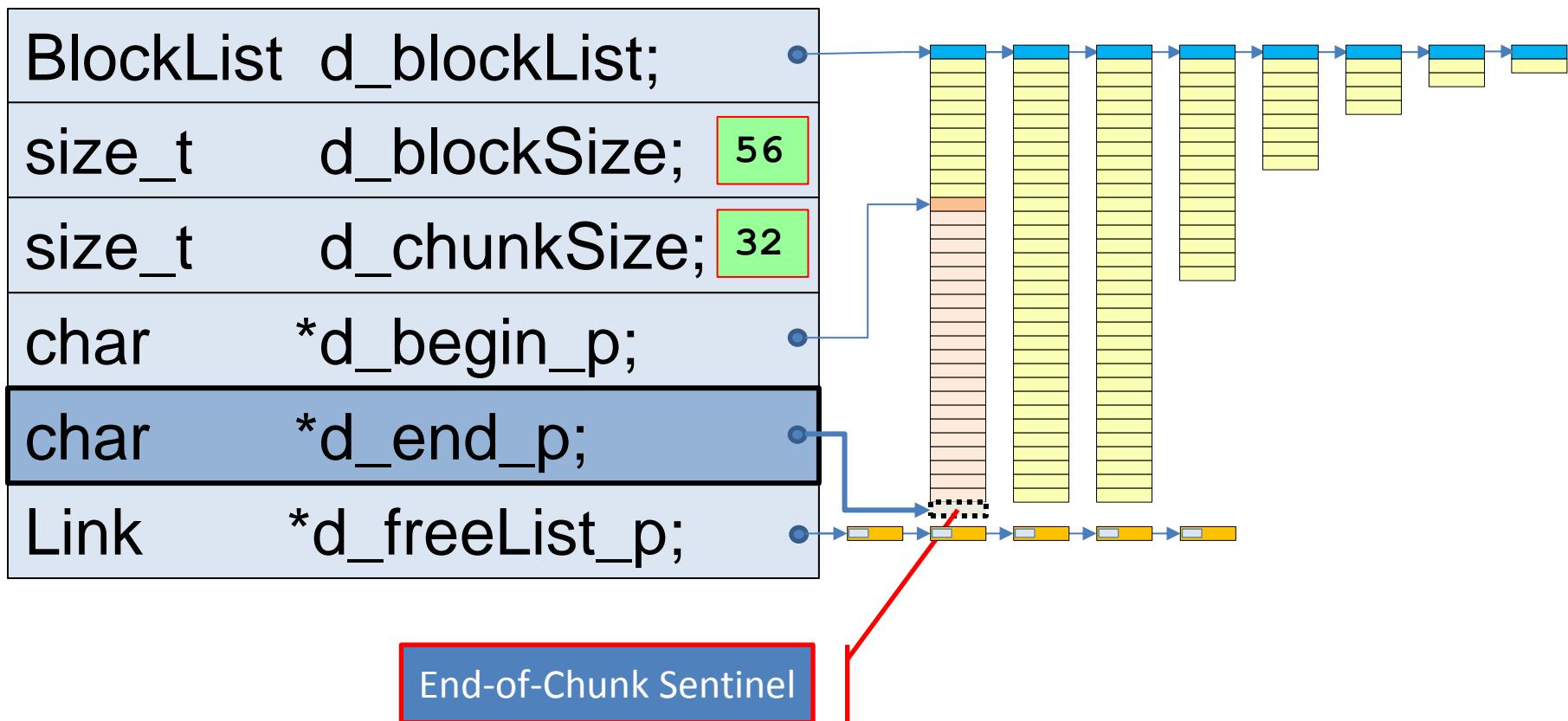
3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)



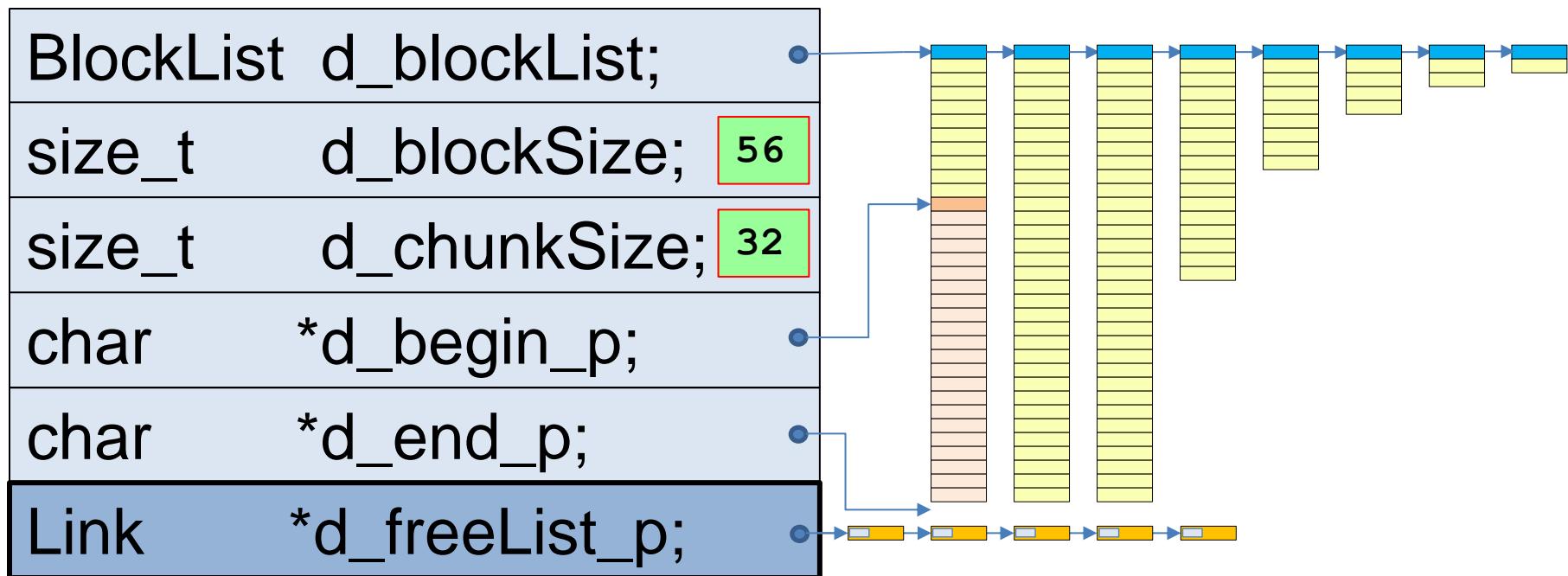
3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)



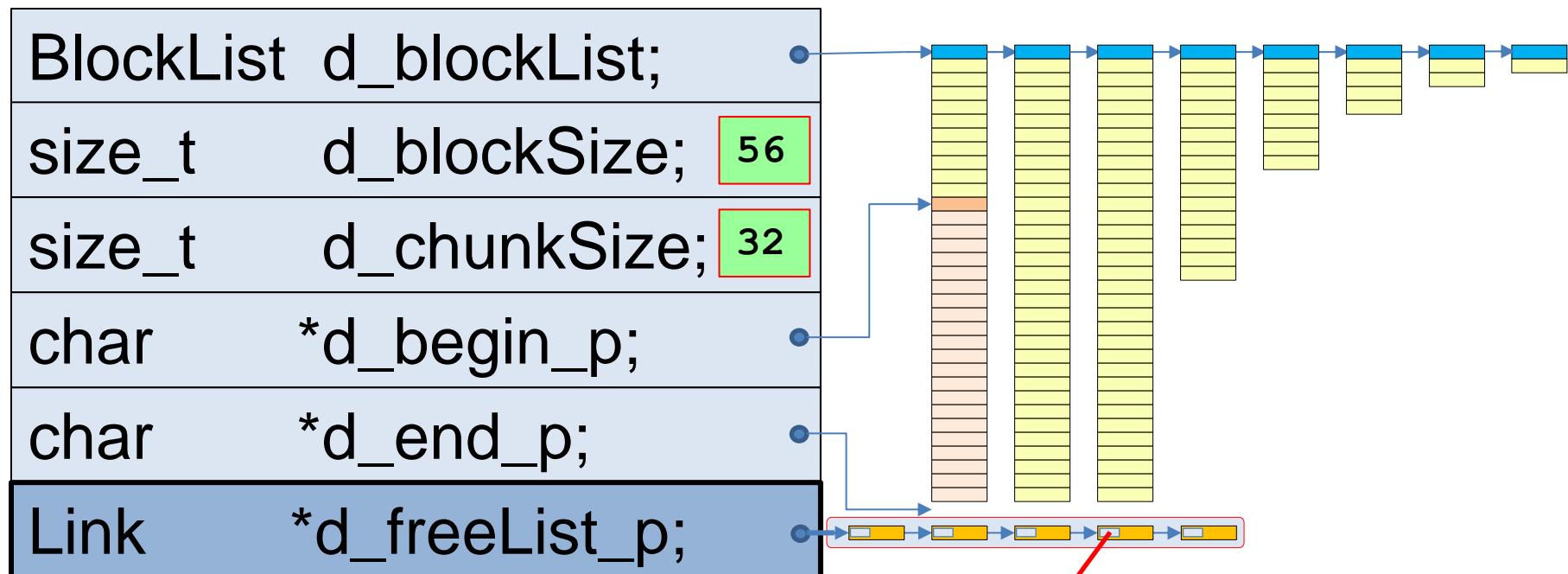
3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)



3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

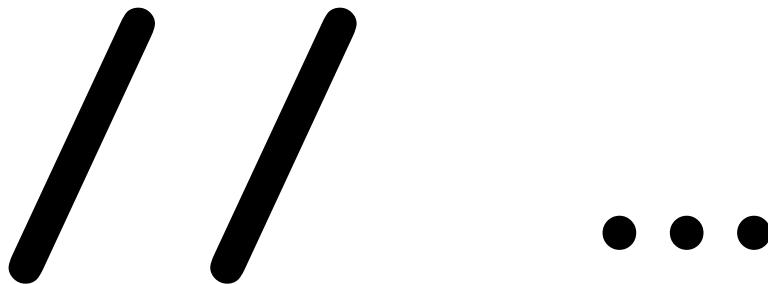


List of Individual Blocks Deallocated Since
the Most-Recent Chunk was Allocated

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
class Pool {
```

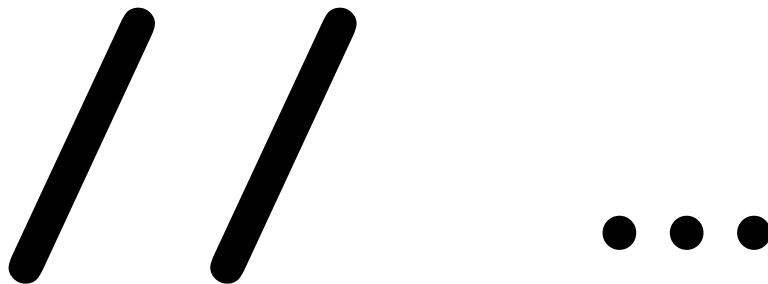


```
public:  
    explicit Pool(size_t blockSize);  
    void *allocate();  
    void deallocate(void *blockAddress);  
};
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
class Pool {
```



```
public:  
    explicit Pool(size_t blockSize);  
    void *allocate();  
    void deallocate(void *blockAddress);  
};
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
class Pool {  
    BlockList d_blockList;      ◦  
    size_t     d_blockSize;     56  
    size_t     d_chunkSize;     0  
    char       *d_begin_p;      ◦  
    char       *d_end_p;        ◦  
    Link       *d_freeList_p;    ◦  
  
public:  
    explicit Pool(size_t blockSize);  
    void *allocate();  
    void deallocate(void *blockAddress);  
};
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
class Pool {
```

```
    BlockList d_blockList;
```

```
    size_t      d_blockSize; 56
```

```
    size_t      d_chunkSize; 0
```

```
    char       *d_begin_p;
```

```
    char       *d_end_p;
```

```
    Link      *d_freeList_p;
```

```
public:
```

```
    explicit Pool(size_t blockSize);
```

```
    void *allocate();
```

```
    void deallocate(void *blockAddress);
```

```
} ;
```



Empty List



3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
class Pool {
```

```
    BlockList d_blockList;
```

```
    size_t      d_blockSize; 56
```

```
    size_t      d_chunkSize; 0
```

```
    char        *d_begin_p; 0
```

```
    char        *d_end_p; 0
```

```
    Link        *d_freeList_p; 0
```

```
public:
```

```
    explicit Pool(size_t blockSize);
```

```
    void *allocate();
```

```
    void deallocate(void *blockAddress);
```

```
};
```



Empty List

56

0



Created with 56
as Constructor
Argument

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
class Pool {  
    BlockList d_blockList;  
    size_t     d_blockSize;    56  
    size_t     d_chunkSize;   0  
    char      *d_begin_p;  
    char      *d_end_p;  
    Link      *d_freeList_p;  
  
public:  
    explicit Pool(size_t blockSize);  
    void *allocate();  
    void deallocate(void *blockAddress);  
};
```

The diagram illustrates the structure of the `Pool` class. It shows the following members:

- `BlockList d_blockList;`: Represented by a light blue box.
- `size_t d_blockSize; 56`: Represented by a light blue box with a green highlighted value `56`.
- `size_t d_chunkSize; 0`: Represented by a light blue box with a green highlighted value `0`.
- `char *d_begin_p;`: Represented by a light blue box.
- `char *d_end_p;`: Represented by a light blue box.
- `Link *d_freeList_p;`: Represented by a light blue box.

Annotations with red lines point from specific values to boxes:

- A red line points from the `56` value to a blue box labeled "Created with 56 as Constructor Argument".
- A red line points from the `0` value to a blue box labeled "Arbitrary Sentinel Value (current chunk size)".
- A red line points from the empty `d_blockList` pointer to a blue box labeled "Empty List".

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
class Pool {
```

```
    BlockList d_blockList;
```

```
    size_t      d_blockSize; 56
```

```
    size_t      d_chunkSize; 1
```

```
    char        *d_begin_p;
```

```
    char        *d_end_p;
```

```
    Link        *d_freeList_p;
```

```
public:
```

```
    explicit Pool(size_t blockSize);
```

```
    void *allocate();
```

```
    void deallocate(void *blockAddress);
```

```
};
```



Empty List



56



1

Created with 56
as Constructor
Argument



Potentially Useful
Sentinel Value
(next chunk size)

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
class Pool {  
    BlockList d_blockList;      ◦  
    size_t     d_blockSize;     56  
    size_t     d_chunkSize;     1  
    char       *d_begin_p;      ◦  
    char       *d_end_p;        ◦  
    Link       *d_freeList_p;    ◦  
  
public:  
    explicit Pool(size_t blockSize);  
    void *allocate();  
    void deallocate(void *blockAddress);  
};
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
class Pool {  
    BlockList d_blockList;      ◦  
    size_t     d_blockSize;    56  
    size_t     d_chunkSize;   1  
    char      *d_begin_p;      ◦  
    char      *d_end_p;        ◦  
    Link      *d_freeList_p;    ◦  
  
public:  
    explicit Pool(size_t blockSize);  
    void *allocate();  
    void deallocate(void *blockAddress);  
};
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
class Pool {
```

```
    BlockList d_blockList;
```

```
    size_t      d_blockSize;  56
```

```
    size_t      d_chunkSize;  2
```

```
    char       *d_begin_p;
```

```
    char       *d_end_p;
```

```
    Link      *d_freeList_p;
```

Returned

```
public:
```

```
    explicit Pool(size_t blockSize);
```

```
    void *allocate();
```

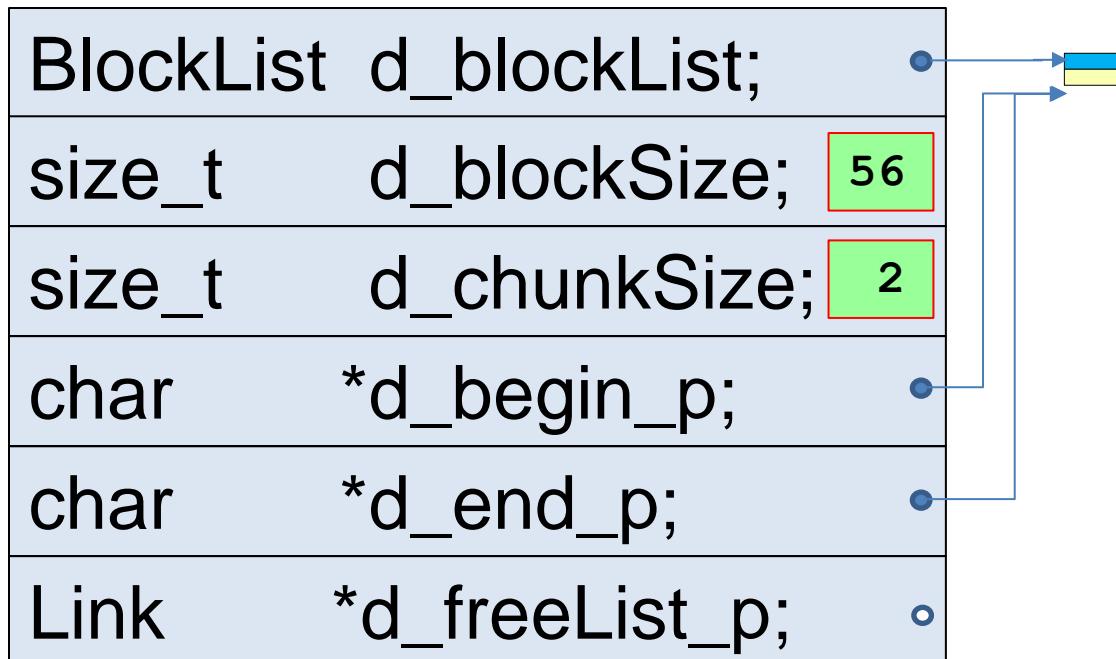
```
    void deallocate(void *blockAddress);
```

```
} ;
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
class Pool {
```



```
public:
```

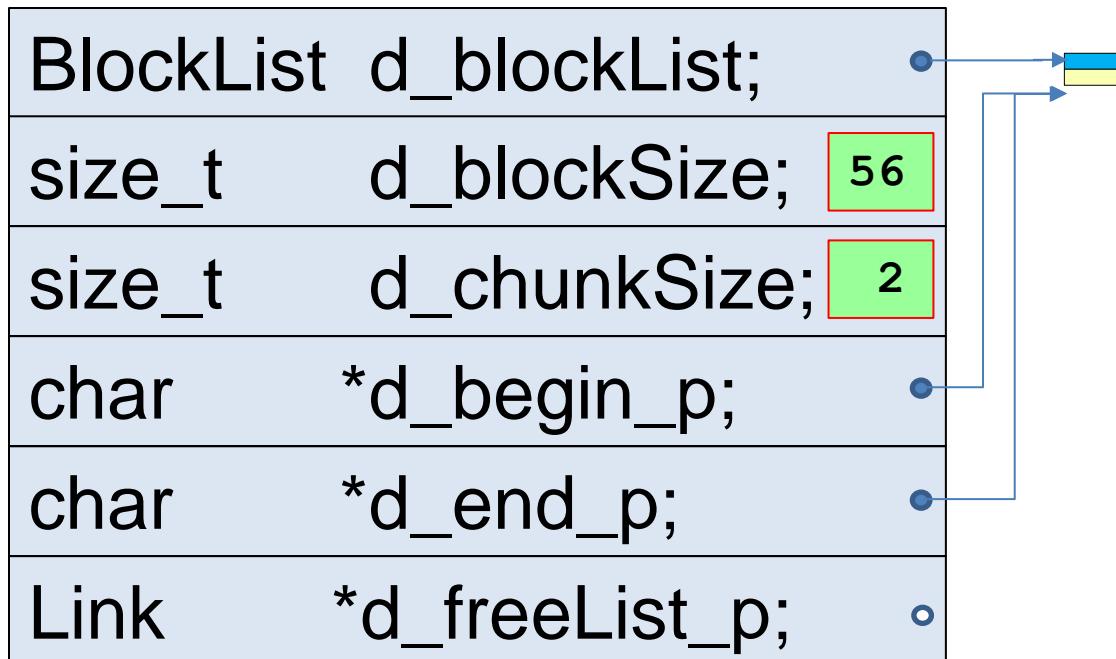
```
    explicit Pool(size_t blockSize);  
    void *allocate();  
    void deallocate(void *blockAddress);
```

```
} ;
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
class Pool {
```



```
public:
```

```
    explicit Pool(size_t blockSize);
```

```
    void *allocate();
```

```
    void deallocate(void *blockAddress);
```

```
};
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
class Pool {
```

```
    BlockList d_blockList;
```

```
    size_t      d_blockSize; 56
```

```
    size_t      d_chunkSize; 4
```

```
    char       *d_begin_p;
```

```
    char       *d_end_p;
```

```
    Link      *d_freeList_p;
```

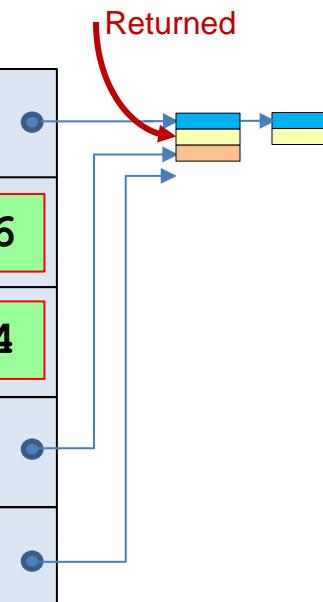
```
public:
```

```
    explicit Pool(size_t blockSize);
```

```
    void *allocate();
```

```
    void deallocate(void *blockAddress);
```

```
} ;
```



3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
class Pool {
```



```
public:
```

```
    explicit Pool(size_t blockSize);  
    void *allocate();  
    void deallocate(void *blockAddress);
```

```
} ;
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
class Pool {
```



```
public:
```

```
    explicit Pool(size_t blockSize);
```

```
    void *allocate();
```

```
void deallocate(void *blockAddress);
```

```
};
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
class Pool {
```



```
public:
```

```
    explicit Pool(size_t blockSize);
```

```
    void *allocate();
```

```
void deallocate(void *blockAddress);
```

```
} ;
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
class Pool {
```



```
public:
```

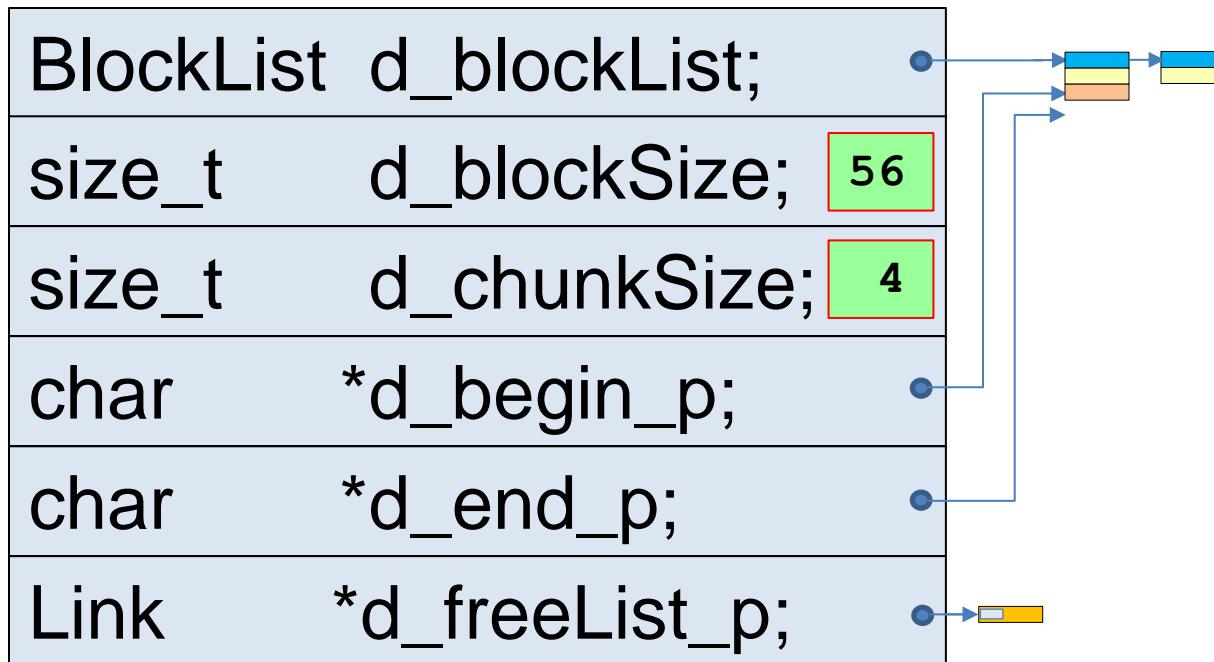
```
    explicit Pool(size_t blockSize);  
    void *allocate();  
    void deallocate(void *blockAddress);
```

```
} ;
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
class Pool {
```



```
public:
```

```
    explicit Pool(size_t blockSize);
```

```
    void *allocate();
```

```
    void deallocate(void *blockAddress);
```

```
} ;
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
class Pool {
```

```
    BlockList d_blockList;
```

```
    size_t      d_blockSize; 56
```

```
    size_t      d_chunkSize; 4
```

```
    char       *d_begin_p;
```

```
    char       *d_end_p;
```

```
    Link      *d_freeList_p;
```

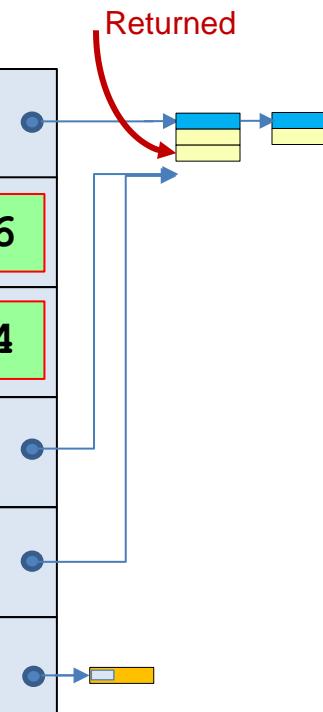
```
public:
```

```
    explicit Pool(size_t blockSize);
```

```
    void *allocate();
```

```
    void deallocate(void *blockAddress);
```

```
} ;
```



3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
class Pool {
```



```
public:
```

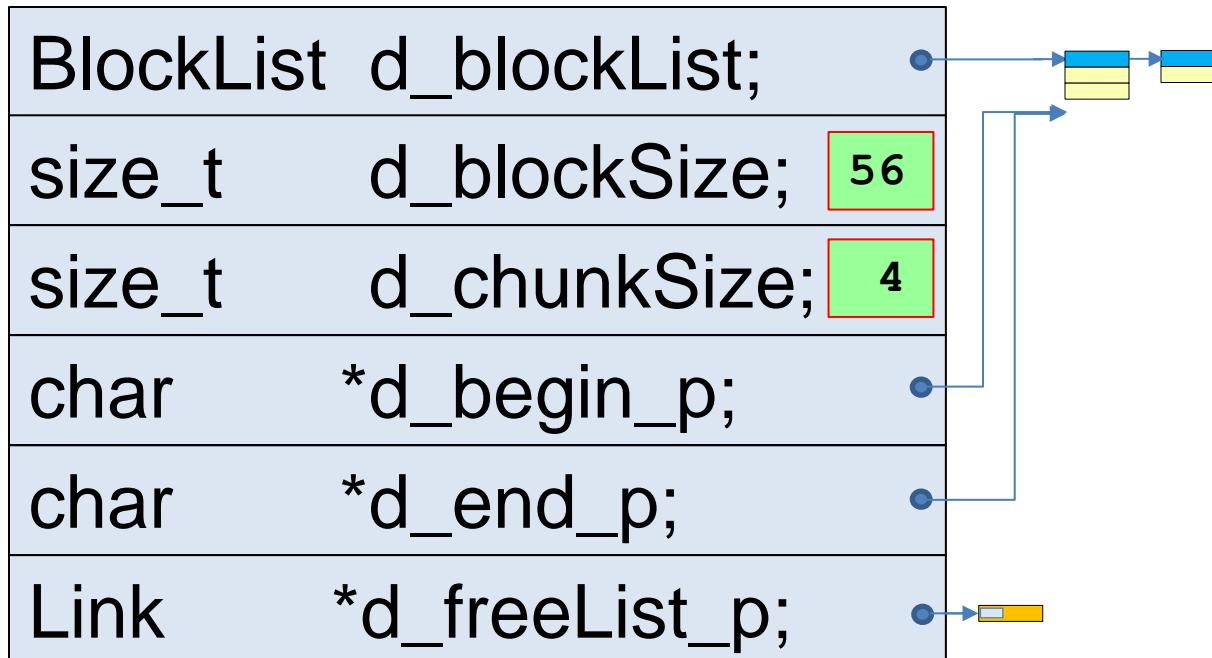
```
    explicit Pool(size_t blockSize);  
    void *allocate();  
    void deallocate(void *blockAddress);
```

```
} ;
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
class Pool {
```



```
public:
```

```
    explicit Pool(size_t blockSize);
```

```
    void *allocate();
```

```
    void deallocate(void *blockAddress);
```

```
};
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
class Pool {
```

```
    BlockList d_blockList;
```

```
    size_t      d_blockSize; 56
```

```
    size_t      d_chunkSize; 4
```

```
    char       *d_begin_p;
```

```
    char       *d_end_p;
```

```
    Link      *d_freeList_p;
```

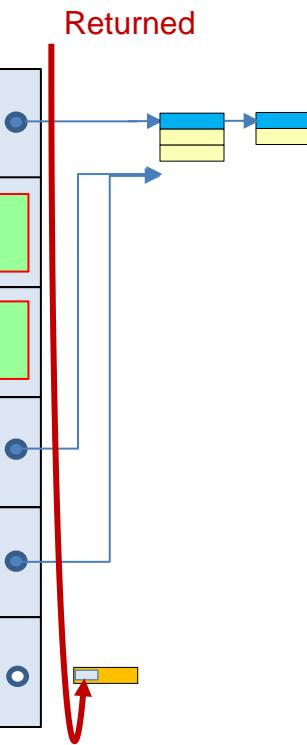
```
public:
```

```
    explicit Pool(size_t blockSize);
```

```
    void *allocate();
```

```
    void deallocate(void *blockAddress);
```

```
} ;
```



3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
class Pool {
```



```
public:
```

```
    explicit Pool(size_t blockSize);  
    void *allocate();  
    void deallocate(void *blockAddress);
```

```
} ;
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
class Pool {
```



```
public:
```

```
    explicit Pool(size_t blockSize);
```

```
    void *allocate();
```

```
    void deallocate(void *blockAddress);
```

```
} ;
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

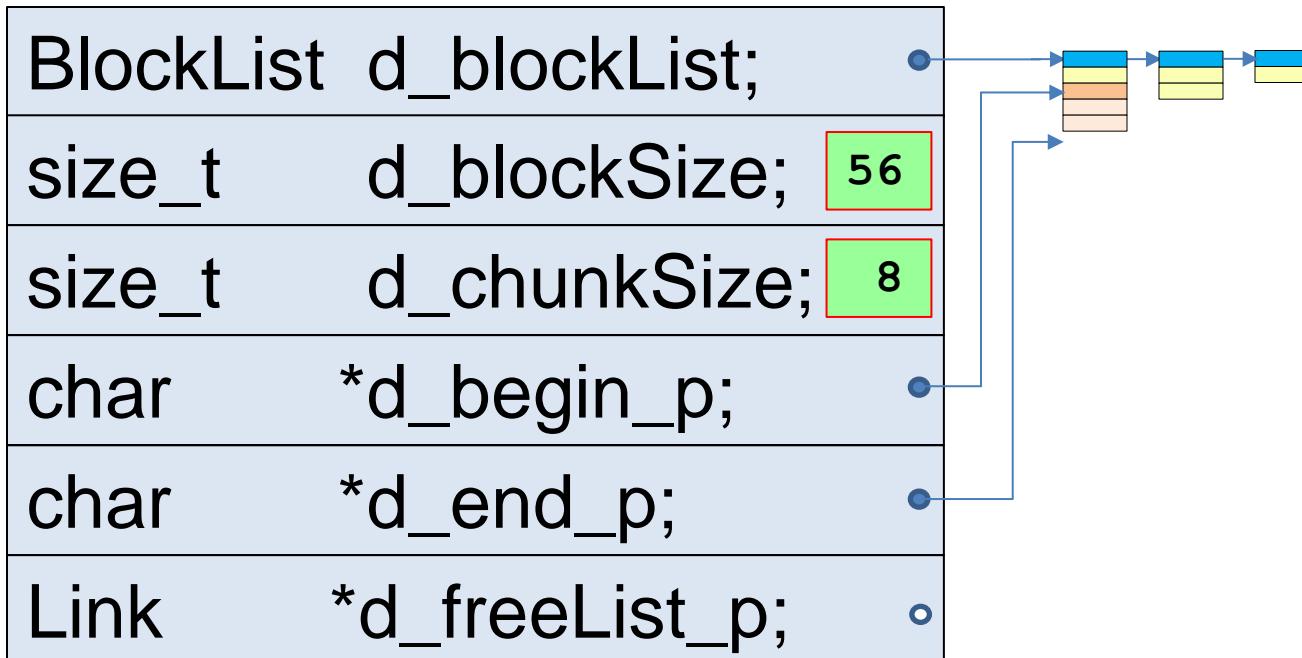
```
class Pool {  
    BlockList d_blockList;  
    size_t      d_blockSize;    56  
    size_t      d_chunkSize;   8  
    char       *d_begin_p;  
    char       *d_end_p;  
    Link       *d_freeList_p;  ○  
  
public:  
    explicit Pool(size_t blockSize);  
    void *allocate();  
    void deallocate(void *blockAddress);  
};
```

The diagram illustrates the internal structure of the `Pool` class and its usage. On the left, the `Pool` class definition is shown with its private members: `d_blockList`, `d_blockSize` (containing the value 56), `d_chunkSize` (containing the value 8), `d_begin_p`, `d_end_p`, and `d_freeList_p`. On the right, a linked list of memory blocks is depicted. The list consists of several nodes, each containing a blue header and a yellow body. A red arrow labeled "Returned" points from the `d_freeList_p` pointer in the `Pool` object to the first node in the list, indicating that a previously allocated block has been returned to the pool.

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
class Pool {
```



```
public:
```

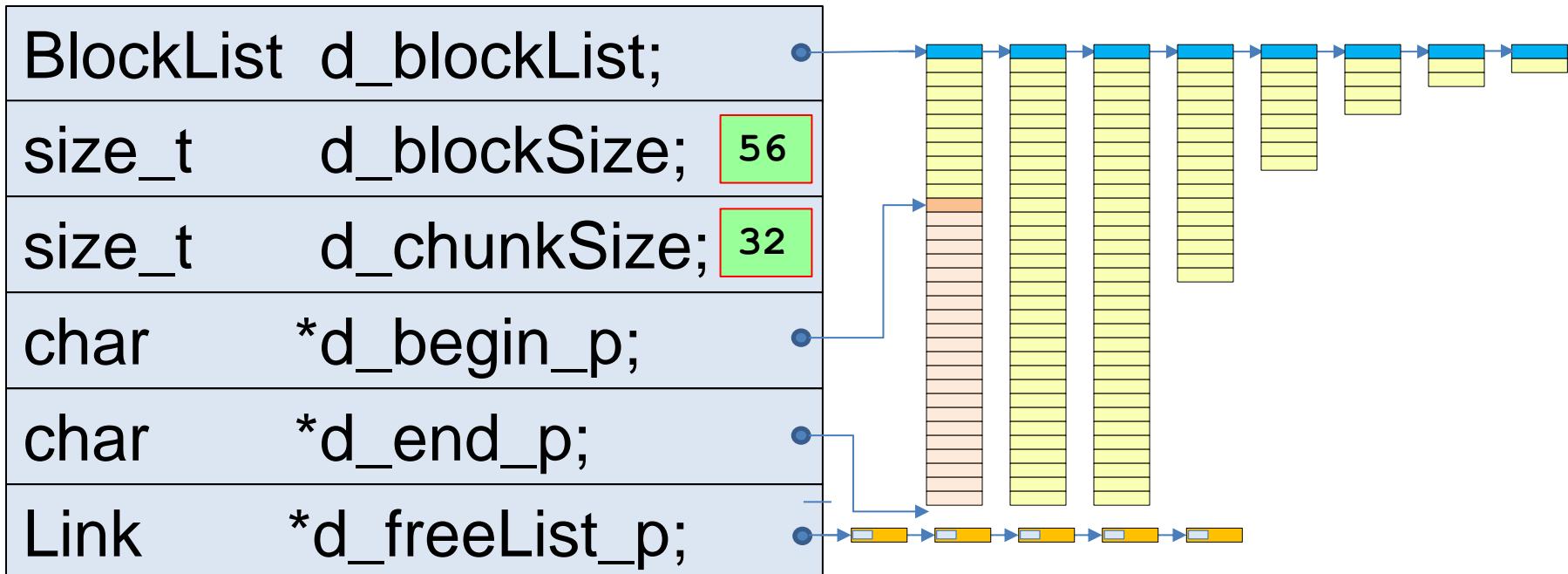
```
    explicit Pool(size_t blockSize);  
    void *allocate();  
    void deallocate(void *blockAddress);
```

```
} ;
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
class Pool {
```



```
public:
```

```
    explicit Pool(size_t blockSize);  
    void *allocate();  
    void deallocate(void *blockAddress);
```

```
} ;
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h
#include <blocklist.h>
class Pool {
    Blocklist  d_blockList;
    size_t      d_blockSize;
    size_t      d_chunkSize;
    char        *d_begin_p;
    char        *d_end_p;
    struct Link { d_next_p; }
                 *d_freeList_p;

public:
    Pool(size_t blockSize);
    void *allocate();
    void deallocate(void *a);
};
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h
#include <blocklist.h>
class Pool {
    Blocklist  d_blockList;
    size_t      d_blockSize;
    size_t      d_chunkSize;
    char        *d_begin_p;
    char        *d_end_p;
    struct Link { d_next_p; }
                 *d_freeList_p;

public:
    Pool(size_t blockSize);
    void *allocate();
    void deallocate(void *a);
};
```

```
// pool.h (continued)
inline
Pool::Pool(size_t blockSize)
: d_blockList(blockSize)
, d_blockSize(blockSize)
, d_chunkSize(1) // special!
, d_begin_p(0)
, d_end_p(0)
, d_freeList_p(0) { }
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h
#include <blocklist.h>
class Pool {
    Blocklist  d_blockList;
    size_t      d_blockSize;
    size_t      d_chunkSize;
    char        *d_begin_p;
    char        *d_end_p;
    struct Link { d_next_p; }
                 *d_freeList_p;

public:
    Pool(size_t blockSize);
    void *allocate();
    void deallocate(void *a);
};
```

```
// pool.h (continued)
inline
Pool::Pool(size_t blockSize)
: d_blockList(blockSize)
, d_blockSize(blockSize)
, d_chunkSize(1) // special!
, d_begin_p(0)
, d_end_p(0)
, d_freeList_p(0) { }
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h
#include <blocklist.h>
class Pool {
    Blocklist  d_blockList;
    size_t      d_blockSize;
    size_t      d_chunkSize;
    char        *d_begin_p;
    char        *d_end_p;
    struct Link { d_next_p; }
                 *d_freeList_p;

public:
    Pool(size_t blockSize);
    void *allocate();
    void deallocate(void *a);
};
```

```
// pool.h (continued)
inline
Pool::Pool(size_t blockSize)
: d_blockList(blockSize)
, d_blockSize(blockSize)
, d_chunkSize(1) // special!
, d_begin_p(0)
, d_end_p(0)
, d_freeList_p(0) { }
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h
#include <blocklist.h>
class Pool {
    Blocklist  d_blockList;
    size_t      d_blockSize;
    size_t      d_chunkSize;
    char        *d_begin_p;
    char        *d_end_p;
    struct Link { d_next_p; }
                 *d_freeList_p;

public:
    Pool(size_t blockSize);
    void *allocate();
    void deallocate(void *a);
};
```

```
// pool.h (continued)
inline
Pool::Pool(size_t blockSize)
: d_blockList(blockSize)
, d_blockSize(blockSize)
, d_chunkSize(1) // special!
, d_begin_p(0)
, d_end_p(0)
, d_freeList_p(0) { }
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h
#include <blocklist.h>
class Pool {
    Blocklist  d_blockList;
    size_t      d_blockSize;
    size_t      d_chunkSize;
    char        *d_begin_p;
    char        *d_end_p;
    struct Link { d_next_p; }
                 *d_freeList_p;

public:
    Pool(size_t blockSize);
    void *allocate();
    void deallocate(void *a);
};
```

```
// pool.h (continued)
inline
Pool::Pool(size_t blockSize)
: d_blockList(blockSize)
, d_blockSize(blockSize)
, d_chunkSize(1) // special!
, d_begin_p(0)
, d_end_p(0)
, d_freeList_p(0) { }

inline void
Pool::deallocate(void *a) {
    static_cast<Link *>(a)->
        d_next_p = d_freeList_p;
    d_freeList_p =
        static_cast<Link *>(a);
}
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h
#include <blocklist.h>
class Pool {
    Blocklist  d_blockList;
    size_t      d_blockSize;
    size_t      d_chunkSize;
    char        *d_begin_p;
    char        *d_end_p;
    struct Link { d_next_p; }
                 *d_freeList_p;

public:
    Pool(size_t blockSize);
    void *allocate();
    void deallocate(void *a);
};
```

```
// pool.h (continued)
inline
Pool::allocate() {
    if (d_end_p - d_begin_p >= d_chunkSize) {
        // ...
        , d_end_p(0)
        , d_freeList_p(0) { }
    }
    inline void
    Pool::deallocate(void *a) {
        static_cast<Link *>(a)->
            d_next_p = d_freeList_p;
        d_freeList_p =
            static_cast<Link *>(a);
    }
}
```

Link the block
onto the front of
the free list

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h
#include <blocklist.h>
class Pool {
    Blocklist  d_blockList;
    size_t      d_blockSize;
    size_t      d_chunkSize;
    char        *d_begin_p;
    char        *d_end_p;
    struct Link { d_next_p; }
                *d_freeList_p;
    void *getBlockFromNewChunk();
public:
    Pool(size_t blockSize);
    void *allocate();
    void deallocate(void *a);
};
```

```
// pool.h (continued again)
inline
void *Pool::allocate() {
    void *p;
    if (d_begin_p != d_end_p) {
        p = d_begin_p;
        d_begin_p += d_blockSize;
    }
    else if (d_freeList_p) {
        p = d_freeList_p;
        d_freeList_p = p->d_next_p;
    }
    else
        p = getBlockFromNewChunk();
    return p;
}
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h (continued again)
inline
void *Pool::allocate() {
    void *p;
    if (d_begin_p != d_end_p) {
        p = d_begin_p;
        d_begin_p += d_blockSize;
    }
    else if (d_freeList_p) {
        p = d_freeList_p;
        d_freeList_p = p->d_next_p;
    }
    else
        p = getBlockFromNewChunk();
    return p;
}
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h (continued again)
inline
void *Pool::allocate() {
    void *p;
    if (d_begin_p != d_end_p) {
        p = d_begin_p;
        d_begin_p += d_blockSize;
    }
    else if (d_freeList_p) {
        p = d_freeList_p;
        d_freeList_p = p->d_next_p;
    }
    else
        p = getBlockFromNewChunk();
    return p;
}
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h (continued again)
inline
void *Pool::allocate() {
    void *p;
    if (d_begin_p != d_end_p) {
        p = d_begin_p;
        d_begin_p += d_blockSize;
    }
    else if (d_freeList_p) {
        p = d_freeList_p;
        d_freeList_p = p->d_next_p;
    }
    else
        p = getBlockFromNewChunk();
    return p;
}
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h
#include <blocklist.h>
class Pool {
    Blocklist  d_blockList;
    size_t      d_blockSize;
    size_t      d_chunkSize;
    char        *d_begin_p;
    char        *d_end_p;
    struct Link { d_next_p; }
    I           *d_freeList_p;
void *getBlockFromNewChunk();
public:
    Pool(size_t blockSize);
    void *allocate();
    void deallocate(void *a);
};
```

```
// pool.h (continued again)
inline
void *Pool::allocate() {
    void *p;
    if (d_begin_p != d_end_p) {
        p = d_begin_p;
        d_begin_p += d_blockSize;
    }
    else if (d_freeList_p) {
        p = d_freeList_p;
        d_freeList_p = p->d_next_p;
    }
else
    p = getBlockFromNewChunk();
    return p;
}
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h (continued again)
inline
void *Pool::allocate() {
    void *p;
    if (d_begin_p != d_end_p) {
        p = d_begin_p; 
        d_begin_p += d_blocksize;
    }
    else if (d_freeList_p) {
        p = d_freeList_p;
        d_freeList_p = p->d_next_p;
    }
    else
    { 
        p = getBlockFromNewChunk();
    }
    return p;
}
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h (continued again)
inline
void *Pool::allocate() {
    void *p;
    if (d_begin_p != d_end_p) {
        p = d_begin_p;
        d_begin_p += a_BLOCKSIZE;
    }
    else if (d_freeList_p) {
        p = d_freeList_p;
        d_freeList_p = p->d_next_p;
    }
    else
        p = getBlockFromNewChunk();
    return p;
}
```

calling into .cpp file

```
// pool.cpp
#include <pool.h>

// ... (Include any extra header files that
// ... are needed.)

enum { k_MAX_CHUNK_SIZE = 32 };
const double k_GROWTH_FACTOR = 2.0;

void *Pool::getBlockFromNewChunk() {
    size_t size = d_blockSize*d_chunkSize;
    char *p = static_cast<char *>
        (::operator new(size));
    d_begin = p + d_blockSize;
    d_end   = p + size;
    d_chunkSize *= k_GROWTH_FACTOR;
    if (d_chunkSize > k_MAX_CHUNK_SIZE)
        d_chunkSize = k_MAX_CHUNK_SIZE;
    return p;
}
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h (continued again)
inline
void *Pool::allocate() {
    void *p;
    if (d_begin_p != d_end_p) {
        p = d_begin_p;
        d_begin_p += a_BLOCKSIZE;
    }
    else if (d_freeList_p) {
        p = d_freeList_p;
        d_freeList_p = p->d_next_p;
    }
    else
        p = getBlockFromNewChunk();
    return p;
}
```

calling into .cpp file

```
// pool.cpp
#include <pool.h>

// ... (Include any extra header files that
// ... are needed.)

enum { k_MAX_CHUNK_SIZE = 32 };
const double k_GROWTH_FACTOR = 2.0;

void *Pool::getBlockFromNewChunk() {
    size_t size = d_blockSize*d_chunkSize;
    char *p = static_cast<char *>
        (::operator new(size));
    d_begin = p + d_blockSize;
    d_end   = p + size;
    d_chunkSize *= k_GROWTH_FACTOR;
    if (d_chunkSize > k_MAX_CHUNK_SIZE)
        d_chunkSize = k_MAX_CHUNK_SIZE;
    return p;
}
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h (continued again)
inline
void *Pool::allocate() {
    void *p;
    if (d_begin_p != d_end_p) {
        p = d_begin_p;
        d_begin_p += a_BLOCKSIZE;
    }
    else if (d_freeList_p) {
        p = d_freeList_p;
        d_freeList_p = p->d_next_p;
    }
    else
        p = getBlockFromNewChunk();
    return p;
}
```

calling into .cpp file

```
// pool.cpp
#include <pool.h>

// ... (Include any extra header files that
// ... are needed.)

enum { k_MAX_CHUNK_SIZE = 64 };
const double k_GROWTH_FACTOR = 2.0;

void *Pool::getBlockFromNewChunk() {
    size_t size = d_blockSize*d_chunkSize;
    char *p = static_cast<char *>
        (::operator new(size));
    d_begin = p + d_blockSize;
    d_end   = p + size;
    d_chunkSize *= k_GROWTH_FACTOR;
    if (d_chunkSize > k_MAX_CHUNK_SIZE)
        d_chunkSize = k_MAX_CHUNK_SIZE;
    return p;
}
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h (continued again)
inline
void *Pool::allocate() {
    void *p;
    if (d_begin_p != d_end_p) {
        p = d_begin_p;
        d_begin_p += a_blockSize;
    }
    else if (d_freeList_p) {
        p = d_freeList_p;
        d_freeList_p = p->d_next_p;
    }
    else
        p = getBlockFromNewChunk ();
    return p;
}
```

calling into .cpp file

```
// pool.cpp
#include <pool.h>

// ... (Include any extra header files that
// ... are needed.)

enum { k_MAX_CHUNK_SIZE = 64 };
const double k_GROWTH_FACTOR = 2.0;

void *Pool::getBlockFromNewChunk () {
    size_t size = d_blockSize*d_chunkSize;
    char *p = static_cast<char *>
        (::operator new(size));
    d_begin = p + d_blockSize;
    d_end   = p + size;
    d_chunkSize *= k_GROWTH_FACTOR;
    if (d_chunkSize > k_MAX_CHUNK_SIZE)
        d_chunkSize = k_MAX_CHUNK_SIZE;
    return p;
}
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h (continued again)
inline
void *Pool::allocate() {
    void *p;
    if (d_begin_p != d_end_p) {
        p = d_begin_p;
        d_begin_p += a_blockSize;
    }
    else if (d_freeList_p) {
        p = d_freeList_p;
        d_freeList_p = p->d_next_p;
    }
    else
        p = getBlockFromNewChunk();
    return p;
}
```

calling into .cpp file

```
// pool.cpp
#include <pool.h>

// ... (Include any extra header files that
// ... are needed.)

enum { k_MAX_CHUNK_SIZE = 64 };
const double k_GROWTH_FACTOR = 2.0;

void *Pool::getBlockFromNewChunk() {
    if (1 == d_chunksize) d_chunksize = 8;

    size_t size = d_blockSize*d_chunkSize;
    char *p = static_cast<char *>
        (::operator new(size));
    d_begin = p + d_blockSize;
    d_end   = p + size;
    d_chunkSize *= k_GROWTH_FACTOR;
    if (d_chunkSize > k_MAX_CHUNK_SIZE)
        d_chunkSize = k_MAX_CHUNK_SIZE;
    return p;
}
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h (continued again)
inline
void *Pool::alloc(size_t size) {
    void *p;
    if (d_begin_p == d_end_p) {
        p = d_begin_p;
        d_begin_p += size;
    }
    else if (d_freeList_p) {
        p = d_freeList_p;
        d_freeList_p = p->d_next_p;
    }
    else {
        p = getBlockFromNewChunk();
    }
    return p;
}
```

More Speed Efficient?

calling into .cpp file

```
// pool.cpp
#include <pool.h>

// ... (Include any extra header files that
// ... are needed.)

enum { k_MAX_CHUNK_SIZE = 64 };
const double k_GROWTH_FACTOR = 2.0;

void *Pool::getBlockFromNewChunk() {
    if (1 == d_chunkSize) d_chunkSize = 8;

    size_t size = d_blockSize*d_chunkSize;
    char *p = static_cast<char *>
        (::operator new(size));
    d_begin = p + d_blockSize;
    d_end = p + size;
    d_chunkSize *= k_GROWTH_FACTOR;
    if (d_chunkSize > k_MAX_CHUNK_SIZE)
        d_chunkSize = k_MAX_CHUNK_SIZE;
    return p;
}
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h (continued again)
inline
void *Pool::allocate() {
    void *p;
    if (d_begin_p != d_end_p) {
        p = d_begin_p;
        d_begin_p += a_BLOCKSIZE;
    }
    else if (d_freeList_p) {
        p = d_freeList_p;
        d_freeList_p = p->d_next_p;
    }
    else
        p = getBlockFromNewChunk();
    return p;
}
```

calling into .cpp file

```
// pool.cpp
#include <pool.h>

// ... (Include any extra header files that
// ... are needed.)

enum { k_MAX_CHUNK_SIZE = 32 };
const double k_GROWTH_FACTOR = 2.0;

void *Pool::getBlockFromNewChunk() {
    size_t size = d_blockSize*d_chunkSize;
    char *p = static_cast<char *>
        (::operator new(size));
    d_begin = p + d_blockSize;
    d_end   = p + size;
    d_chunkSize *= k_GROWTH_FACTOR;
    if (d_chunkSize > k_MAX_CHUNK_SIZE)
        d_chunkSize = k_MAX_CHUNK_SIZE;
    return p;
}
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h (continued again)
inline
void *Pool::allocate() {
    void *p;
    if (d_begin_p != d_end_p) {
        p = d_begin_p;
        d_begin_p += a_BLOCKSIZE;
    }
    else if (d_freeList_p) {
        p = d_freeList_p;
        d_freeList_p = p->d_next_p;
    }
    else
        p = getBlockFromNewChunk();
    return p;
}
```

calling into .cpp file

```
// pool.cpp
#include <pool.h>

// ... (Include any extra header files that
// ... are needed.)

enum { k_MAX_CHUNK_SIZE = 32 };
const double k_GROWTH_FACTOR = 2.0;

void *Pool::getBlockFromNewChunk() {
    size_t size = d_blockSize*d_chunkSize;
    char *p = static_cast<char *>
        (::operator new(size));
    d_begin = p + d_blockSize;
    d_end   = p + size;
    d_chunkSize *= k_GROWTH_FACTOR;
    if (d_chunkSize > k_MAX_CHUNK_SIZE)
        d_chunkSize = k_MAX_CHUNK_SIZE;
    return p;
}
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h (continued again)
inline
void *Pool::allocate() {
    void *p;
    if (d_begin_p != d_end_p) {
        p = d_begin_p;
        d_begin_p += a_BLOCKSIZE;
    }
    else if (d_freeList_p) {
        p = d_freeList_p;
        d_freeList_p = p->d_next_p;
    }
    else
        p = getBlockFromNewChunk();
    return p;
}
```

calling into .cpp file

```
// pool.cpp
#include <pool.h>

// ... (Include any extra header files that
// ... are needed.)

enum { k_MAX_CHUNK_SIZE = 16 };
const double k_GROWTH_FACTOR = 2.0;

void *Pool::getBlockFromNewChunk() {
    size_t size = d_blockSize*d_chunkSize;
    char *p = static_cast<char *>
        (::operator new(size));
    d_begin = p + d_blockSize;
    d_end   = p + size;
    d_chunkSize *= k_GROWTH_FACTOR;
    if (d_chunkSize > k_MAX_CHUNK_SIZE)
        d_chunkSize = k_MAX_CHUNK_SIZE;
    return p;
}
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h (continued again)
inline
void *Pool::allocate() {
    void *p;
    if (d_begin_p != d_end_p) {
        p = d_begin_p;
        d_begin_p += a_BLOCKSIZE;
    }
    else if (d_freeList_p) {
        p = d_freeList_p;
        d_freeList_p = p->d_next_p;
    }
    else
        p = getBlockFromNewChunk();
    return p;
}
```

calling into .cpp file

```
// pool.cpp
#include <pool.h>

// ... (Include any extra header files that
// ... are needed.)

enum { k_MAX_CHUNK_SIZE = 16 };
const double k_GROWTH_FACTOR = 2.0;

void *Pool::getBlockFromNewChunk() {
    size_t size = d_blockSize*d_chunkSize;
    char *p = static_cast<char *>
        (::operator new(size));
    d_begin = p + d_blockSize;
    d_end   = p + size;
    d_chunkSize *= k_GROWTH_FACTOR;
    if (d_chunkSize > k_MAX_CHUNK_SIZE)
        d_chunkSize = k_MAX_CHUNK_SIZE;
    return p;
}
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h (continued again)
inline
void *Pool::allocate() {
    void *p;
    if (d_begin_p != d_end_p) {
        p = d_begin_p;
        d_begin_p += a_BLOCKSIZE;
    }
    else if (d_freeList_p) {
        p = d_freeList_p;
        d_freeList_p = p->d_next_p;
    }
    else
        p = getBlockFromNewChunk();
    return p;
}
```

calling into .cpp file

```
// pool.cpp
#include <pool.h>

// ... (Include any extra header files that
// ... are needed.)

enum { k_MAX_CHUNK_SIZE = 16 };
const double k_GROWTH_FACTOR = 1.25;

void *Pool::getBlockFromNewChunk() {
    size_t size = d_blockSize*d_chunkSize;
    char *p = static_cast<char *>
        (::operator new(size));
    d_begin = p + d_blockSize;
    d_end   = p + size;
    d_chunkSize *= k_GROWTH_FACTOR;
    if (d_chunkSize > k_MAX_CHUNK_SIZE)
        d_chunkSize = k_MAX_CHUNK_SIZE;
    return p;
}
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h (continued again)
inline
void *Pool::allocate() {
    void *p;
    if (d_begin_p != d_end_p) {
        p = d_begin_p;
        d_begin_p += a_BLOCKSIZE;
    }
    else if (d_freeList_p) {
        p = d_freeList_p;
        d_freeList_p = p->d_next_p;
    }
    else
        p = getBlockFromNewChunk ();
    return p;
}
```

calling into .cpp file

```
// pool.cpp
#include <pool.h>

// ... (Include any extra header files that
// ... are needed.)

enum { k_MAX_CHUNK_SIZE = 16 };
const double k_GROWTH_FACTOR = 1.25;

void *Pool::getBlockFromNewChunk() {
    size_t size = d_blockSize*d_chunkSize;
    char *p = static_cast<char *>
        (::operator new(size));
    d_begin = p + d_blockSize;
    d_end = p + size;
    d_chunkSize *= k_GROWTH_FACTOR;
}

if (d_chunkSize > k_MAX_CHUNK_SIZE)
    d_chunkSize = k_MAX_CHUNK_SIZE;
return p;
}
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h (continued again)
inline
void *Pool::allocate() {
    void *p;
    if (d_begin_p != d_end_p) {
        p = d_begin_p;
        d_begin_p += a_BLOCKSIZE;
    }
    else if (d_freeList_p) {
        p = d_freeList_p;
        d_freeList_p = p->d_next_p;
    }
    else
        p = getBlockFromNewChunk ();
    return p;
}
```

calling into .cpp file

```
// pool.cpp
#include <pool.h>

// ... (Include any extra header files that
// ... are needed.)

enum { k_MAX_CHUNK_SIZE = 16 };
const double k_GROWTH_FACTOR = 1.25;

void *Pool::getBlockFromNewChunk() {
    size_t size = d_blockSize*d_chunkSize;
    char *p = static_cast<char *>
        (::operator new(size));
    d_begin = p + d_blockSize;
    d_end = p + size;
    d_chunkSize *= k_GROWTH_FACTOR;
    if (d_chunksize < 4) ++d_chunksize;
    if (d_chunkSize > k_MAX_CHUNK_SIZE)
        d_chunkSize = k_MAX_CHUNK_SIZE;
    return p;
}
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h (continued again)
inline
void *Pool::allocate(size_t a_blockSize) {
    void *p;
    if (d_begin_ == 0) {
        p = d_begin_;
        d_begin_ += a_blockSize;
    }
    else if (d_freeList_ != 0) {
        p = d_freeList_;
        d_freeList_ = p->d_next_p;
    }
    else {
        p = getBlockFromNewChunk();
    }
    return p;
}
```

More
Space
Efficient?

calling into .cpp file

```
// pool.cpp
#include <pool.h>

// ... (Include any extra header files that
// ... are needed.)

enum { k_MAX_CHUNK_SIZE = 16 };
const double k_GROWTH_FACTOR = 1.25;

void *Pool::getBlockFromNewChunk() {
    size_t size = d_blockSize*d_chunkSize;
    char *p = static_cast<char *>
        (::operator new(size));
    d_begin = p + d_blockSize;
    d_end = p + size;
    d_chunkSize *= k_GROWTH_FACTOR;
    if (d_chunksize < 4) ++d_chunksize;
    if (d_chunkSize > k_MAX_CHUNK_SIZE)
        d_chunkSize = k_MAX_CHUNK_SIZE;
    return p;
}
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h (continued again)
inline
void *Pool::allocate() {
    void *p;
    if (p = getBlockFromExistingChunk()) {
        return p;
    } else {
        else {
            p = getBlockFromNewChunk();
            return p;
        }
    }
}
```

```
// pool.cpp
#include <pool.h>

// ... (Include any extra header files that
// ... are needed.)

enum { k_MAX_CHUNK_SIZE = 16 };
const float k_GROWTH_FACTOR = 1.25;

void *Pool::allocate(size_t size) {
    void *p;
    if (p = getBlockFromExistingChunk(size)) {
        return p;
    } else {
        else {
            p = getBlockFromNewChunk(size);
            return p;
        }
    }
}
```

Partial Insulation
Facilitates
Performance Tuning

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h
#include <blocklist.h>
class Pool {
    Blocklist  d_blockList;
    size_t      d_blockSize;
    size_t      d_chunkSize;
    char        *d_begin_p;
    char        *d_end_p;
    struct Link { d_next_p; }
                 *d_freeList_p;

public:
    Pool(size_t blockSize);
    void *allocate();
    void deallocate(void *a);
};
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h
#include <blocklist.h>
class Pool {
    Blocklist  d_blockList;
    size_t      d_blockSize;
    size_t      d_chunkSize;
    char        *d_begin_p;
    char        *d_end_p;
    struct Link { d_next_p; }
                *d_freeList_p;
    void *getBlockDarnIt();
public:
    Pool(size_t blockSize);
    void *allocate();
    void deallocate(void *a);
};
```

```
// pool.h (continued again/revised)

inline
void *Pool::allocate() {
    if (d_begin_p == d_end_p)
        return getBlockDarnIt();
    void *p = d_begin_p;
    d_begin_p += d_blockSize;
    return p;
}
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h
#include <blocklist.h>
class Pool {
    Blocklist  d_blockList;
    size_t      d_blockSize;
    size_t      d_chunkSize;
    char        *d_begin_p;
    char        *d_end_p;
    struct Link { d_next_p; }
                *d_freeList_p;
    void *getBlockDarnIt();
public:
    Pool(size_t blockSize);
    void *allocate();
    void deallocate(void *a);
};
```

```
// pool.h (continued again/revised)

inline
void *Pool::allocate() {
    if (d_begin_p == d_end_p)
        return getBlockDarnIt();
    void *p = d_begin_p;
    d_begin_p += d_blockSize;
    return p;
}
```



Small!

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h
#include <blocklist.h>
class Pool {
    Blocklist *d_blocklist;
    size_t d_size;
    size_t d_min_size;
    char *d_begin_p;
    char *d_end_p;
    struct {
        void *p;
        size_t size;
    } d_blocks[1];
public:
    Pool(size_t min_size);
    void *allocate();
    void deallocate(void *a);
};
```

```
// pool.h (continued again/revised)
inline
void *Pool::allocate() {
    if (d_begin_p == d_end_p)
        init();
    return d_begin_p + d_size;
}
```

Partial Insulation Reduces Code Bloat!

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h
#ifndef POOL_H
#include <assert.h>
class Pool {
public:
    char *d_begin_p;
    char *d_end_p;
    struct Link { d_next_p; } *d_freeList_p;
    void *getBlockDarnIt();
    public:
        Pool(size_t blockSize);
        void *allocate();
        void deallocate(void *a);
    };
}
```

If there is a
block in the
current chunk,
return it ASAP!

```
// pool.h (continued again/revised)
inline
void *Pool::allocate() {
    if (d_begin_p == d_end_p)
        return getBlockDarnIt();
    void *p = d_begin_p;
    d_begin_p += d_blockSize;
    return p;
}
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h
#ifndef POOL_H
#include <assert.h>
class Pool {
public:
    char *d_begin_p;
    char *d_end_p;
    struct Link { d_next_p; } *d_freeList_p;
    void *getBlockDarnIt();
    public:
        Pool(size_t blockSize);
        void *allocate();
        void deallocate(void *a);
    };
}
```

If there isn't,
whatever
happens next
is insulated!

```
// pool.h (continued again/revised)

inline
void *Pool::allocate() {
    if (d_begin_p == d_end_p)
        return getBlockDarnIt();
    void *p = d_begin_p;
    d_begin_p += d_blockSize;
    return p;
}
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h
#include <blocklist.h>
class Pool {
    Blocklist  d_blockList;
    size_t      d_blockSize;
    size_t      d_chunkSize;
    char        *d_begin_p;
    char        *d_end_p;
    struct Link { d_next_p; }
                 *d_freeList_p;
void *getBlockDarnIt();
public:
    Pool(size_t blockSize);
void *allocate();
    void deallocate(void *a);
};
```



3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h
#include <blocklist.h>
class Pool {
    Blocklist  d_blockList;
    size_t      d_blockSize;
    size_t      d_chunkSize;
    char        *d_begin_p;
    char        *d_end_p;
    struct Link { d_next_p; }
                *d_freeList_p;
void *getBlockDarnIt();
public:
    Pool(size_t blockSize);
void *allocate();
    void deallocate(void *a);
};
```

```
// pool.cpp (revised)
#include <blocklist>

// ... (Include any extra header files that
// ... are needed.)

enum { k_MAX_CHUNK_SIZE = 32 };
const double k_GROWTH_FACTOR = 2.0;

void *Pool::getBlockDarnIt() {
    if (d_freeList_p) {
        Link *q = d_freeList_p;
        d_freeList_p = q->d_next_p;
        return q;
    }
    size_t size = d_blockSize*d_chunkSize;
    char *p = static_cast<char *>
        (::operator new(size));
    d_begin = p + d_blockSize;
    d_end   = p + size;
    d_chunkSize *= k_GROWTH_FACTOR;
    if (d_chunkSize > k_MAX_CHUNK_SIZE)
        d_chunkSize = k_MAX_CHUNK_SIZE;
    return p;
}
```

calling into .cpp file

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.h
#ifndef _POOL_H_
#define _POOL_H_
#include <blocklist.h>
#include <new>

class Pool {
private:
    size_t d_blockSize;
    char *d_begin_p;
    char *d_end_p;
    struct Link {
        char *d_next_p;
        char *d_freeList_p;
    };
public:
    void *getBlockDarnIt();
    Pool(size_t blockSize);
    void *allocate();
    void deallocate(void *a);
};

// pool.cpp (revised)
#include <blocklist>

// ... (Include any extra header files that
// ... are needed.)

enum { k_MAX_CHUNK_SIZE = 32 };
const double k_GROWTH_FACTOR = 2.0;

void *Pool::getBlockDarnIt() {
    if (d_freeList_p) {
        Link *q = d_freeList_p;
        d_freeList_p = q->d_next_p;
        return q;
    }
    size_t size = d_blockSize*k_MAX_CHUNK_SIZE;
    char *p = static_cast<char *>(
        ::operator new(size));
    d_begin = p + d_blockSize;
    d_end = p + size;
    d_chunkSize *= k_GROWTH_FACTOR;
    if (d_chunkSize > k_MAX_CHUNK_SIZE)
        d_chunkSize = k_MAX_CHUNK_SIZE;
    return p;
}
```

Is there a block
available in the
free list?

file

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

```
// pool.cpp (revised)
#include <blocklist>

// ... (Include any extra header files that
// ... are needed.)

enum { k_MAX_CHUNK_SIZE = 32 };
const double k_GROWTH_FACTOR = 2.0;

void *Pool::getBlockDarnIt() {
    if (d_freeList_p) {
        Link *q = d_freeList_p;
        d_freeList_p = q->d_next_p;
        return q;
    }
    size_t size = d_blockSize*d_chunkSize;
    char *p = static_cast<char*>
        (::operator new(size));
    d_begin = p + d_blockSize;
    d_end = p + size;
    d_chunkSize *= k_GROWTH_FACTOR;
    if (d_chunkSize > k_MAX_CHUNK_SIZE)
        d_chunkSize = k_MAX_CHUNK_SIZE;
    return p;
}
```

3. Total and Partial *Insulation* Techniques

Adaptive Memory Pool (Simplified)

Discussion?

3. Total and Partial *Insulation* Techniques

End of Section

Questions?

3. Total and Partial *Insulation* Techniques

What Questions are we Answering?

- What do we mean by the term *Insulation*?
- How does *Insulation* compare with *Encapsulation*?
- Why/when would we put a `#include` directive in a `.h` file?
- What do we mean by *Total Insulation*?
- What are three general *Total Insulation* techniques?
- Which *Total-Insulation* technique is not Architectural?
- What do we mean by *Partial Insulation*?
- What are some examples of *Partial Insulation* techniques?
- What specific *benefits* do we get from insulation?
- What *costs* are generally associated with insulation?
- Why would one choose *Partial* over *Total* insulation?

3. Total and Partial *Insulation* Techniques

What Questions are we Answering?

- What do we mean by the term *Insulation*?
- How does *Insulation* compare with *Encapsulation*?
- Why/when would we put a `#include` directive in a `.h` file?
- What do we mean by *Total Insulation*?
- What are three general *Total Insulation* techniques?
- Which *Total-Insulation* technique is not Architectural?
- What do we mean by *Partial Insulation*?
- What are some examples of *Partial Insulation* techniques?
- What specific *benefits* do we get from insulation?
- What *costs* are generally associated with insulation?
- Why would one choose *Partial* over *Total* insulation?

3. Total and Partial *Insulation* Techniques

What Questions are we Answering?

- What do we mean by the term *Insulation*?
- How does *Insulation* compare with *Encapsulation*?
- Why/when would we put a `#include` directive in a `.h` file?
- What do we mean by *Total Insulation*?
- What are three general *Total Insulation* techniques?
- Which *Total-Insulation* technique is not Architectural?
- What do we mean by *Partial Insulation*?
- What are some examples of *Partial Insulation* techniques?
- What specific *benefits* do we get from insulation?
- What *costs* are generally associated with insulation?
- Why would one choose *Partial* over *Total* insulation?

3. Total and Partial *Insulation* Techniques

What Questions are we Answering?

- What do we mean by the term *Insulation*?
- How does *Insulation* compare with *Encapsulation*?
- Why/when would we put a `#include` directive in a `.h` file?
- What do we mean by *Total Insulation*?
- What are three general *Total Insulation* techniques?
- Which *Total-Insulation* technique is not Architectural?
- What do we mean by *Partial Insulation*?
- What are some examples of *Partial Insulation* techniques?
- What specific *benefits* do we get from insulation?
- What *costs* are generally associated with insulation?
- Why would one choose *Partial* over *Total* insulation?

Outline

1. Review of Elementary Physical Design
Components, Modularity, Physical Dependencies
2. Survey of Advanced *Levelization* Techniques
Avoiding Cyclic, Excessive, or Inappropriate Dependencies
3. Total and Partial *Insulation* Techniques
Avoiding Unnecessary Compile-Time Coupling
4. Present-Day, Real-World Design Examples
Prioritizing Physical Design to Improve Software Quality

Outline

1. Review of Elementary Physical Design
Components, Modularity, Physical Dependencies
2. Survey of Advanced *Levelization* Techniques
Avoiding Cyclic, Excessive, or Inappropriate Dependencies
3. Total and Partial *Insulation* Techniques
Avoiding Unnecessary Compile-Time Coupling
4. Present-Day, Real-World Design Examples
Prioritizing Physical Design to Improve Software Quality

4. Present-Day, Real-World Design Examples

Introduction

All of the software we write is governed
by a common overarching set of
Organizing Principles.

4. Present-Day, Real-World Design Examples

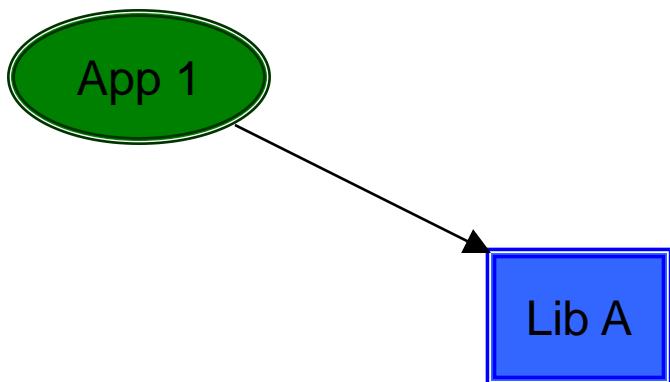
Introduction

All of the software we write is governed by a common overarching set of *Organizing Principles.*

Among the most central of which is achieving *Sound Physical Design.*

4. Present-Day, Real-World Design Examples

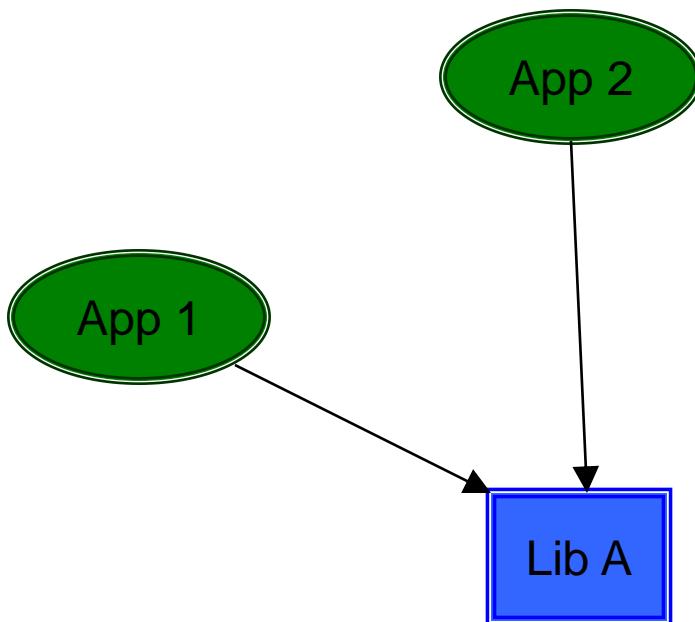
Creating a Big Ball of Mud



4. Present-Day, Real-World Design Examples

Creating a Big Ball of Mud

Where We Put Our Code Matters!

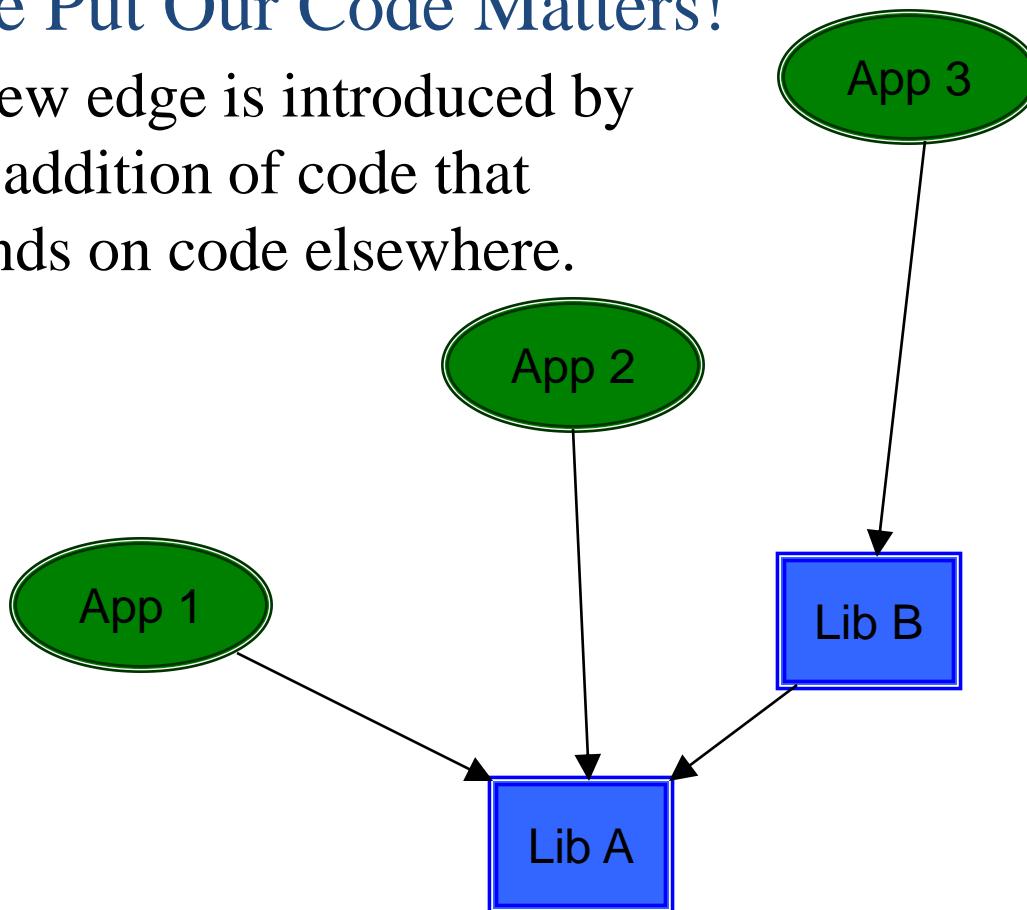


4. Present-Day, Real-World Design Examples

Creating a Big Ball of Mud

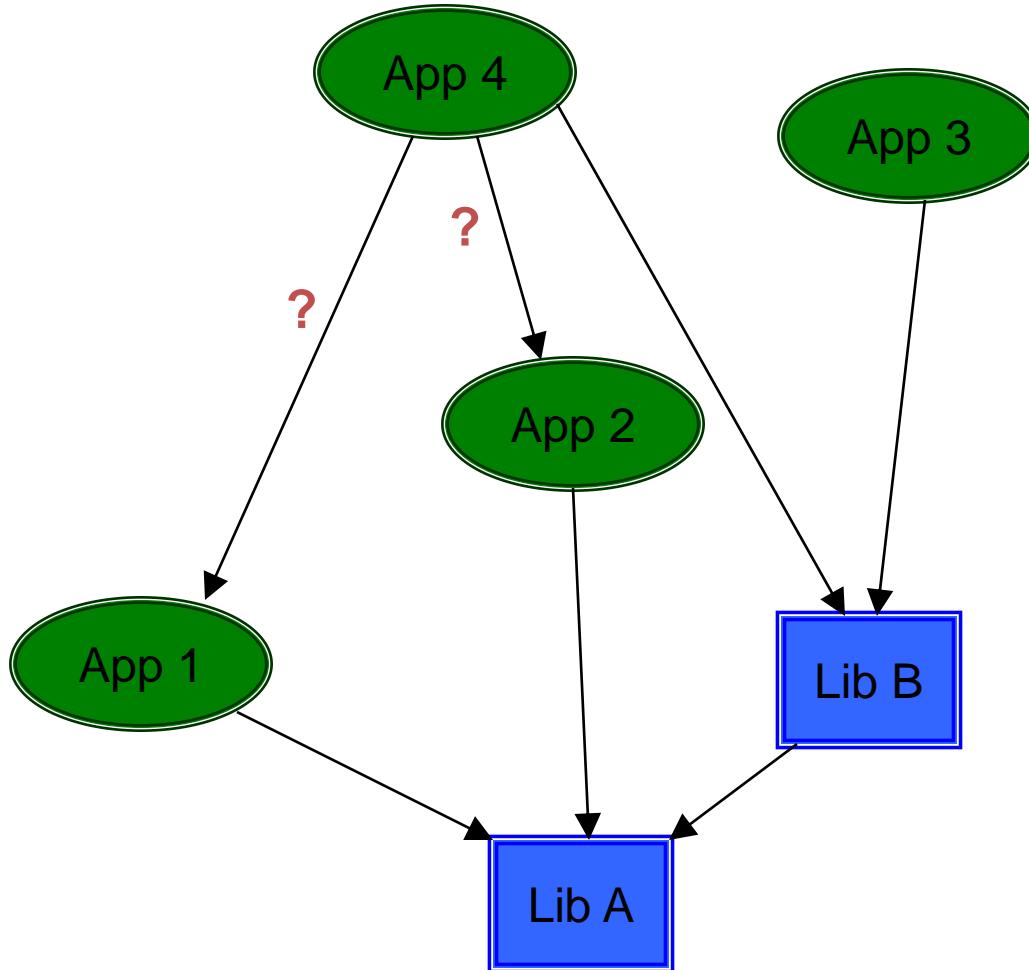
Where We Put Our Code Matters!

Each new edge is introduced by
the addition of code that
depends on code elsewhere.



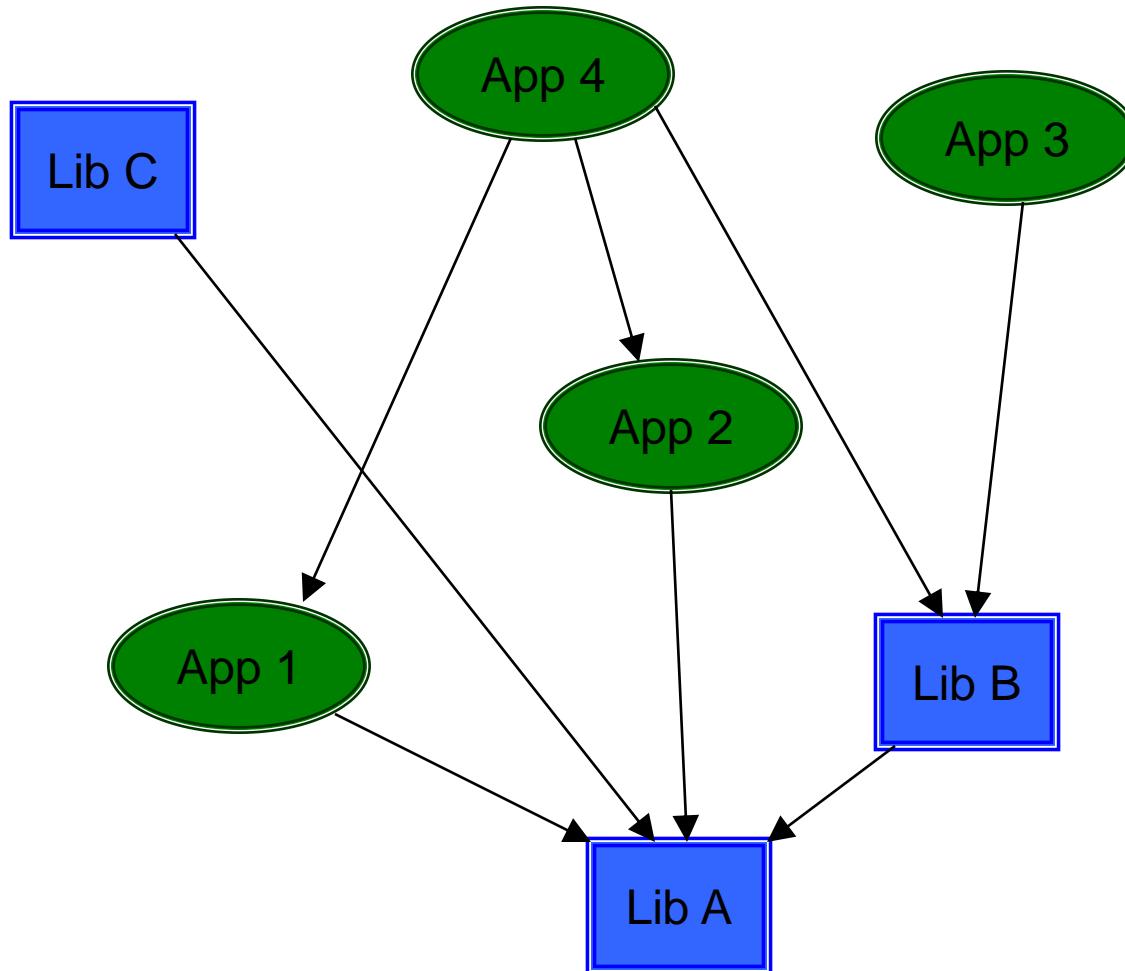
4. Present-Day, Real-World Design Examples

Creating a Big Ball of Mud



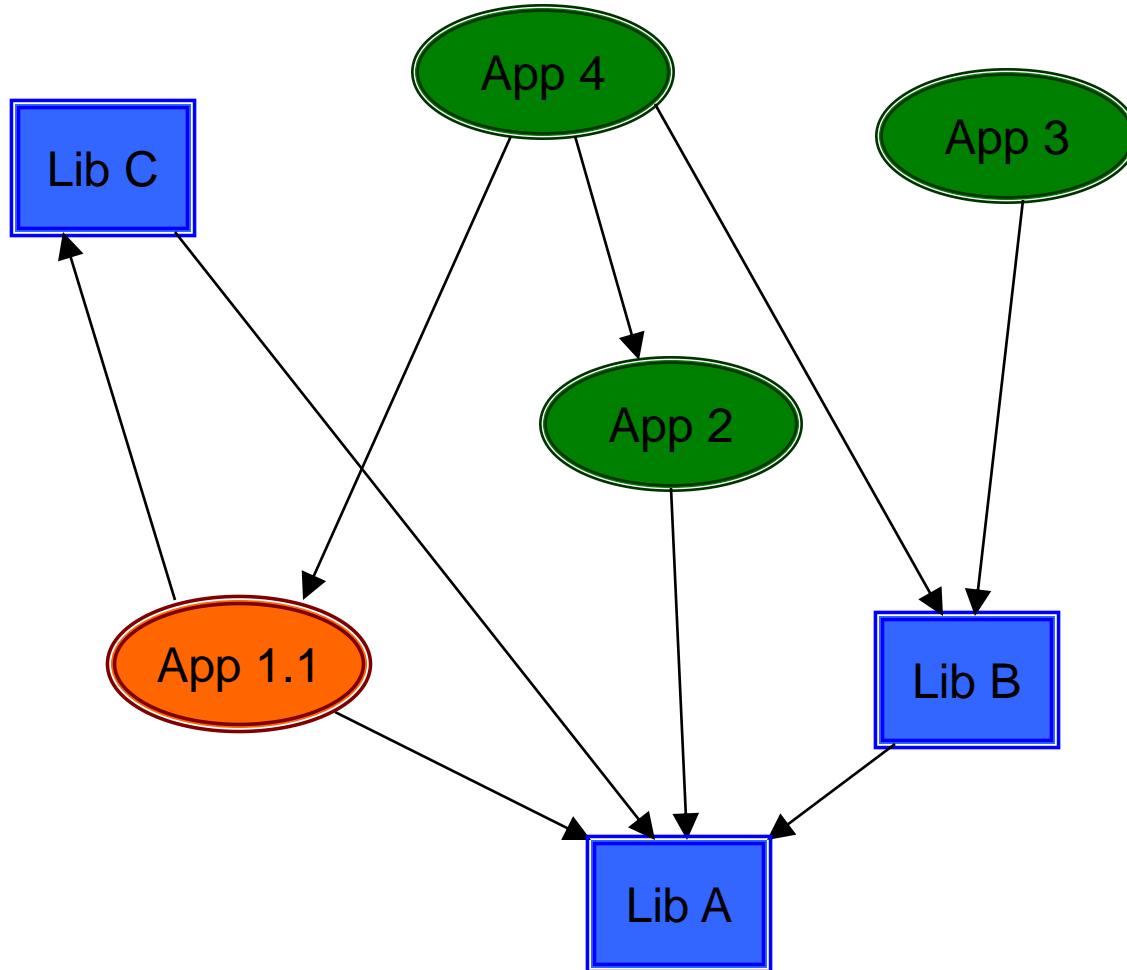
4. Present-Day, Real-World Design Examples

Creating a Big Ball of Mud



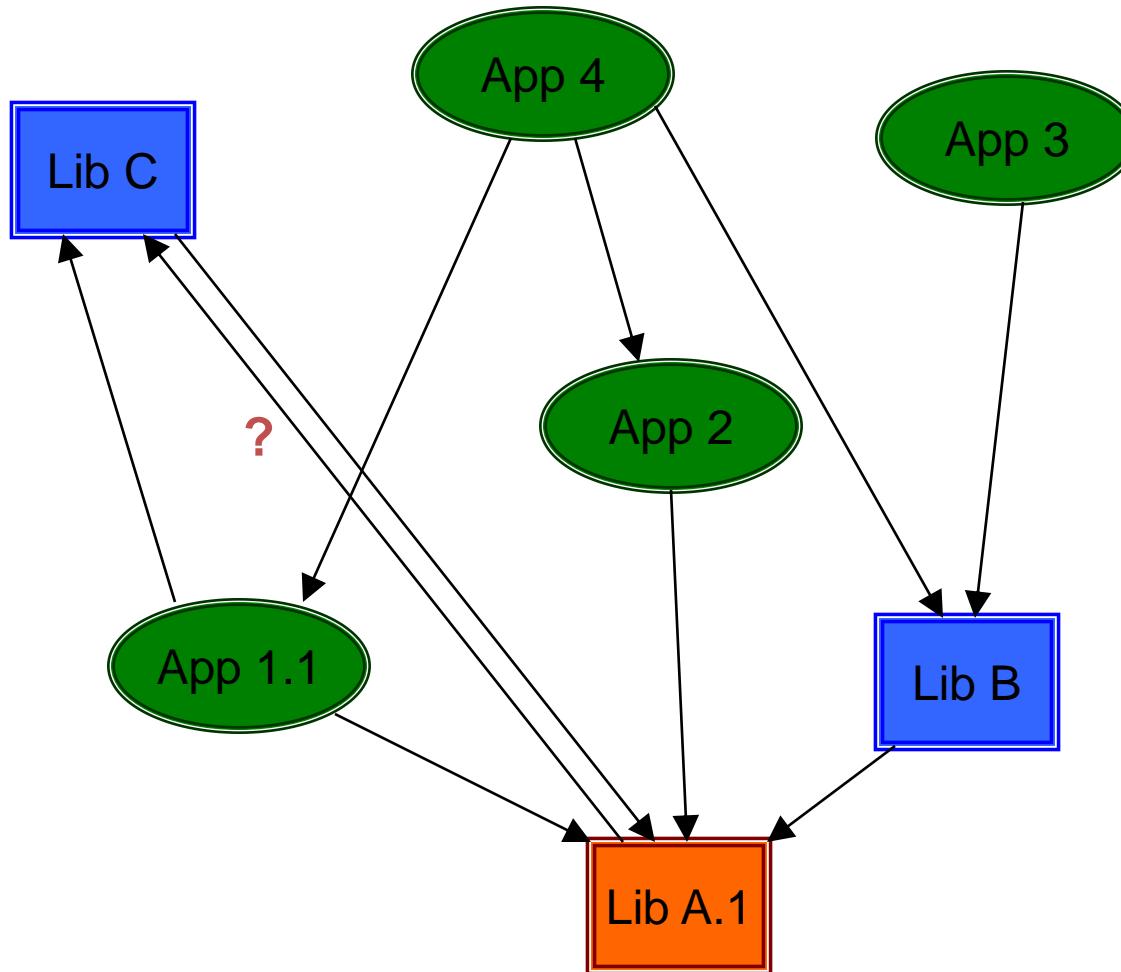
4. Present-Day, Real-World Design Examples

Creating a Big Ball of Mud



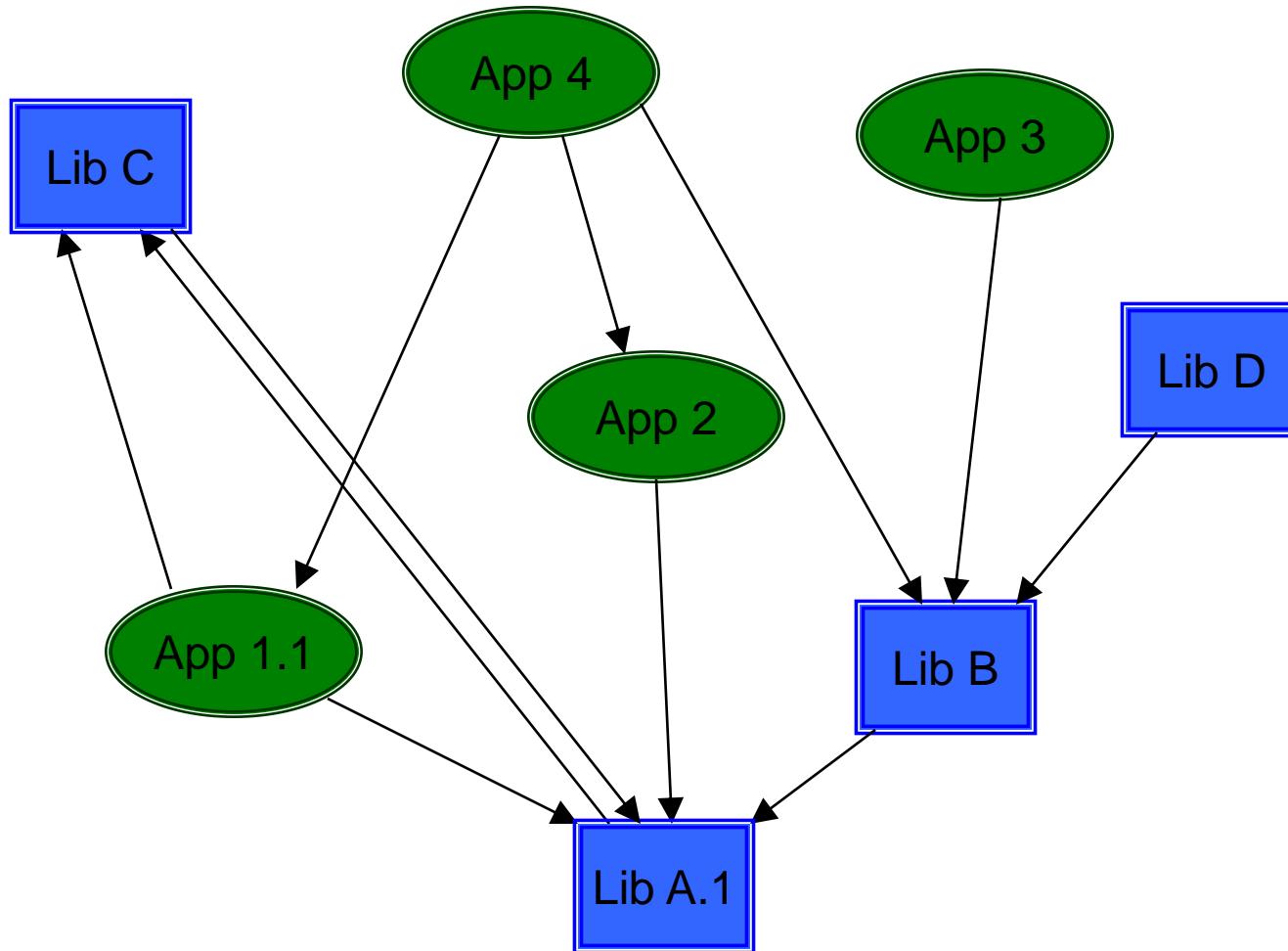
4. Present-Day, Real-World Design Examples

Creating a Big Ball of Mud



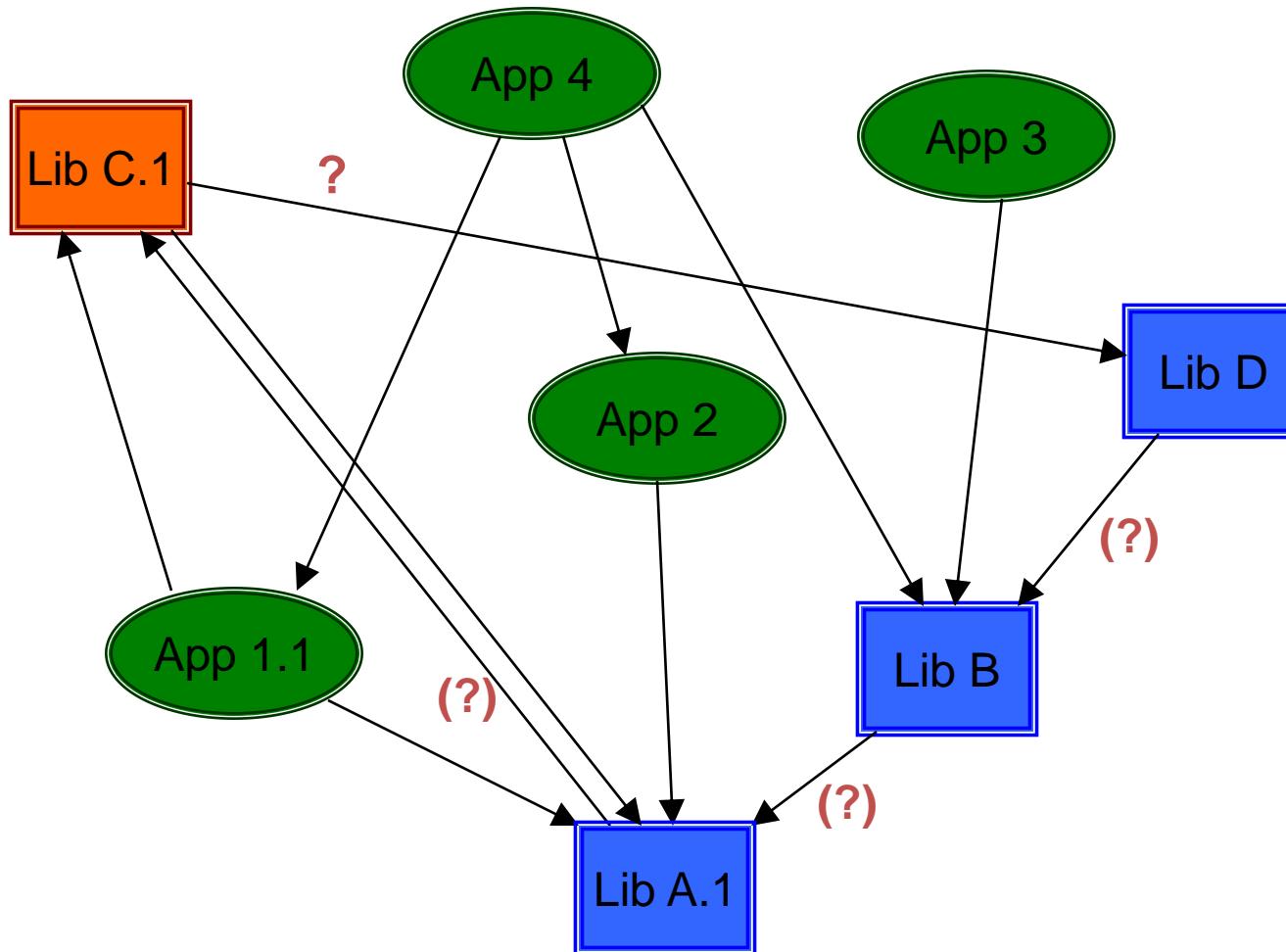
4. Present-Day, Real-World Design Examples

Creating a Big Ball of Mud



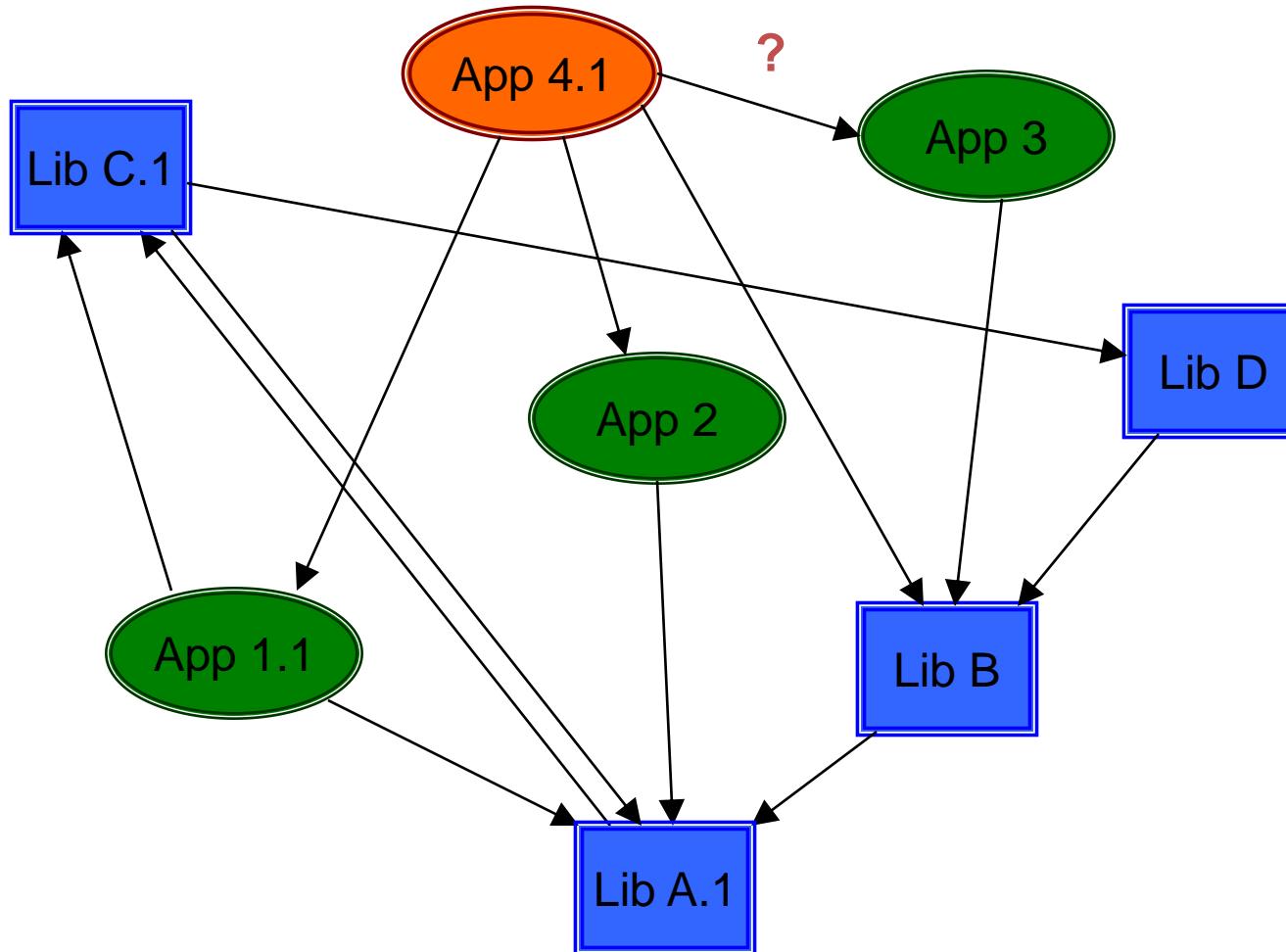
4. Present-Day, Real-World Design Examples

Creating a Big Ball of Mud



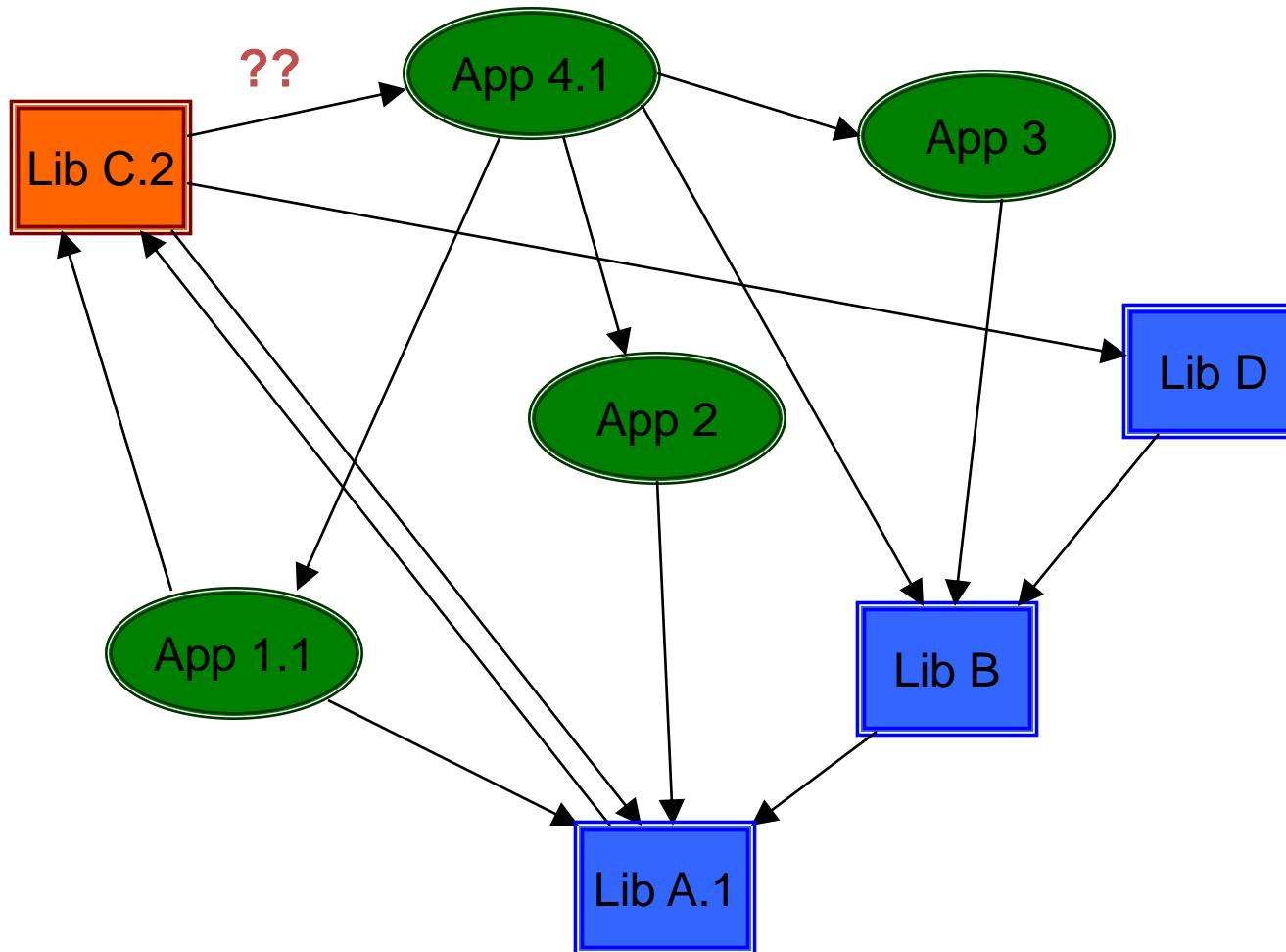
4. Present-Day, Real-World Design Examples

Creating a Big Ball of Mud



4. Present-Day, Real-World Design Examples

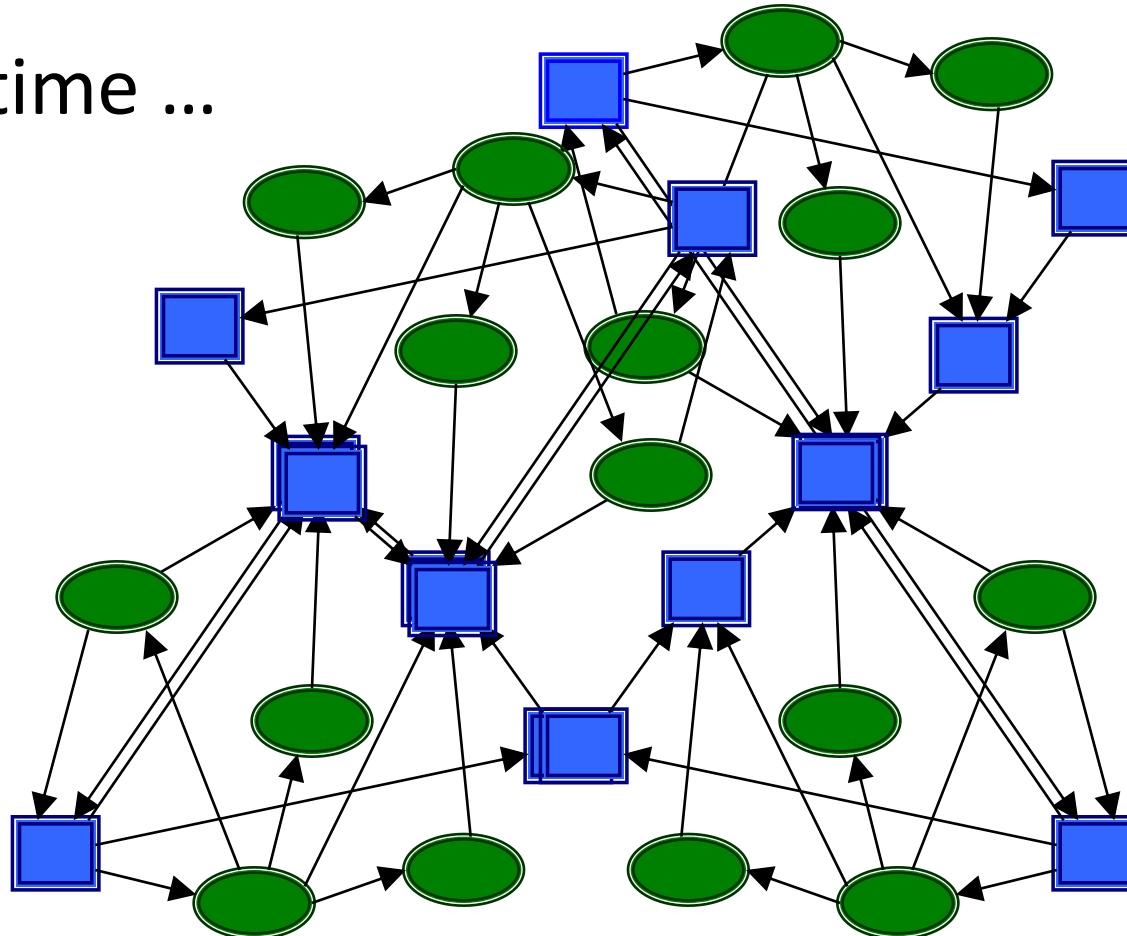
Creating a Big Ball of Mud



4. Present-Day, Real-World Design Examples

Creating a Big Ball of Mud

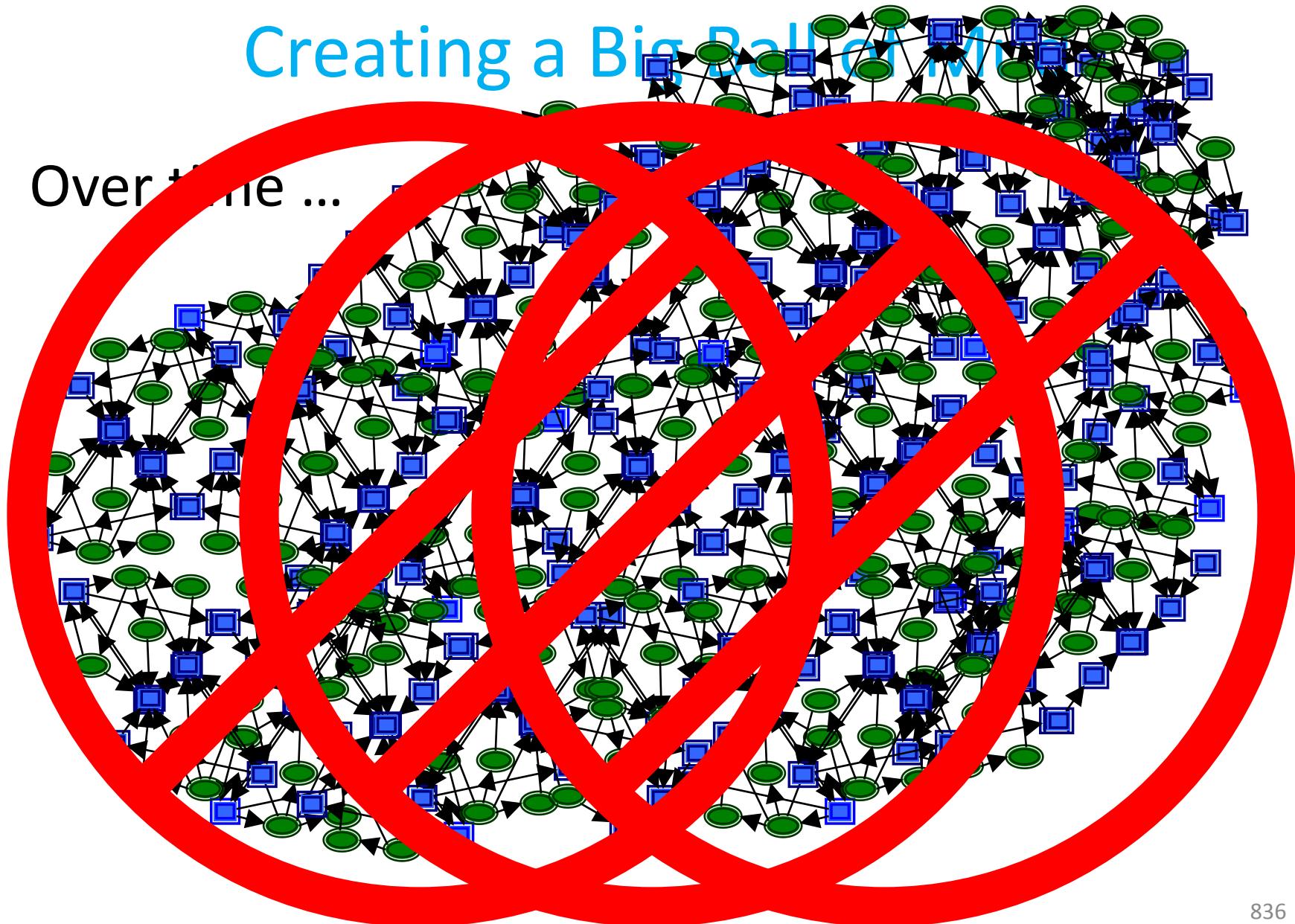
Over time ...

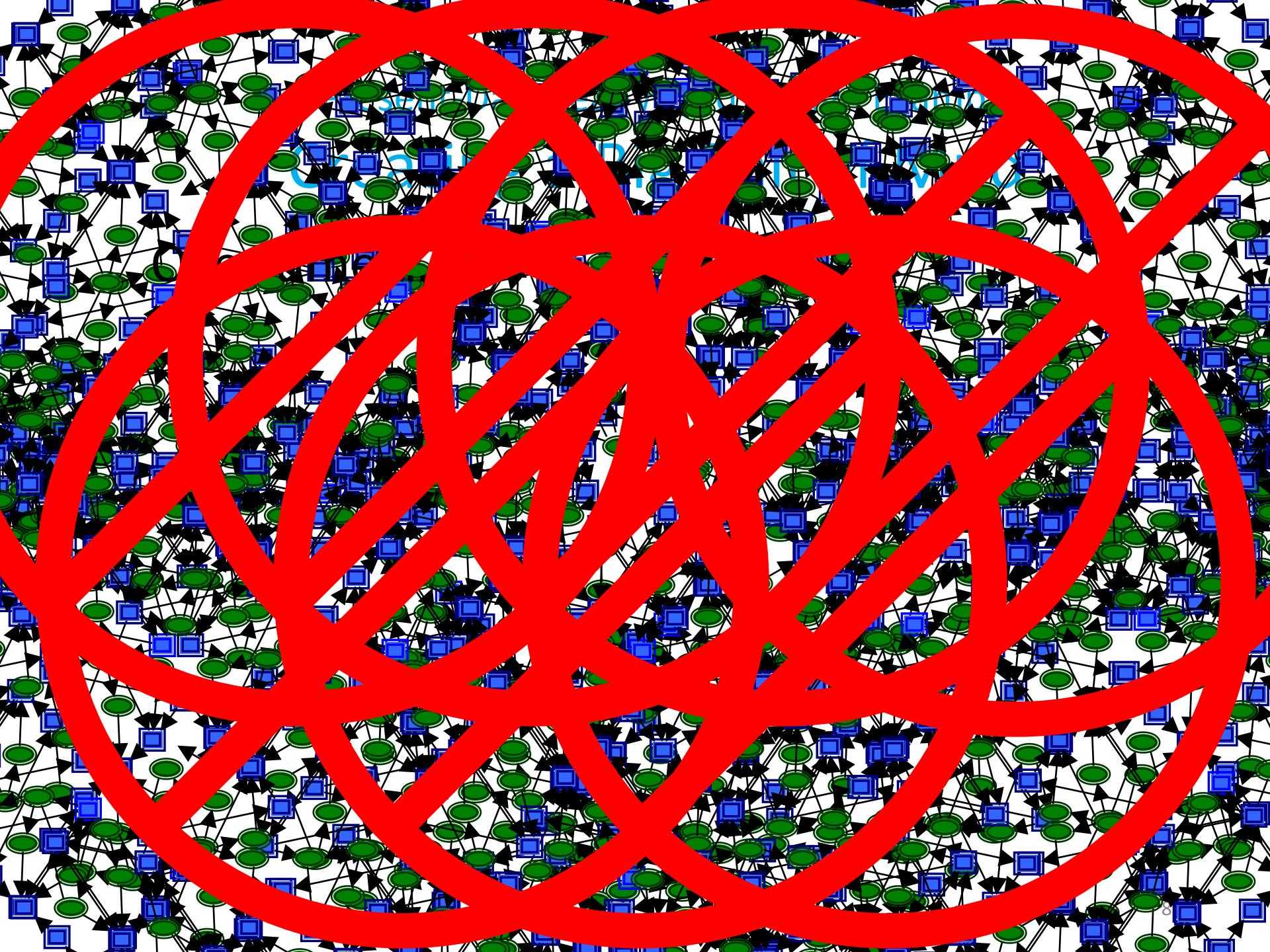


4. Present-Day, Real-World Design Examples

Creating a Big Ball of Mud

Over time ...





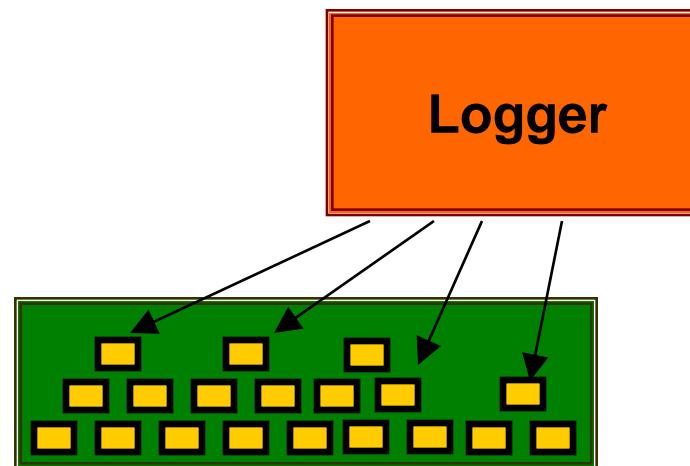
4. Present-Day, Real-World Design Examples

Large-Scale Physical Design

- Good physical design is an **engineering discipline, not** an afterthought.
- Good physical design must be introduced from the **inception** of an application.
- The physical design of our proprietary libraries should be **coherent** across the firm.

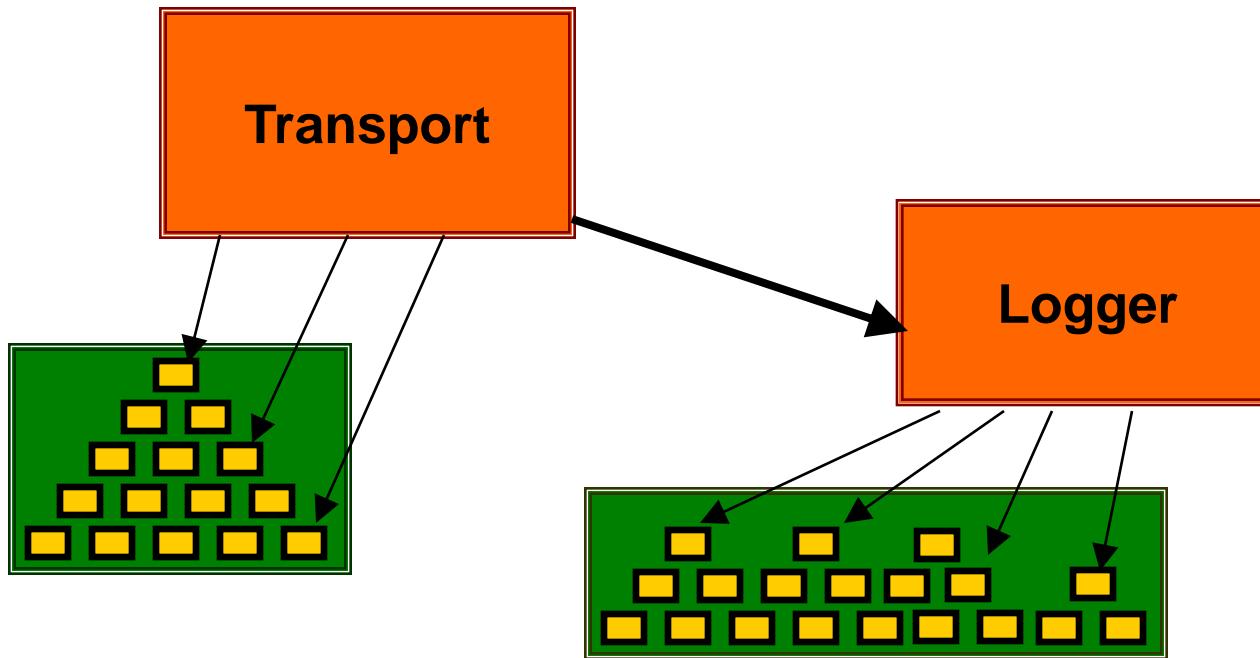
4. Present-Day, Real-World Design Examples

Cyclic Link-time Dependency



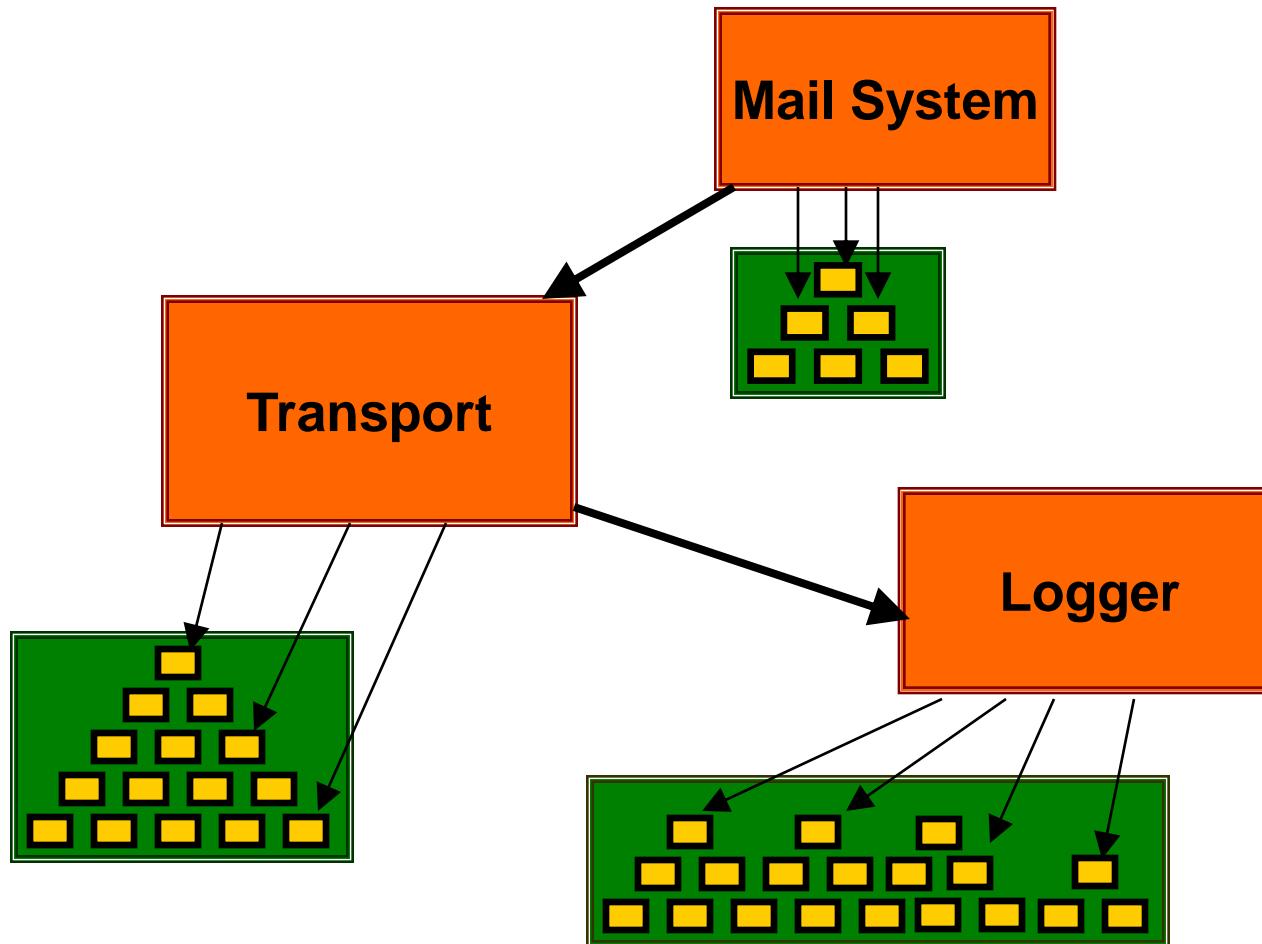
4. Present-Day, Real-World Design Examples

Cyclic Link-time Dependency



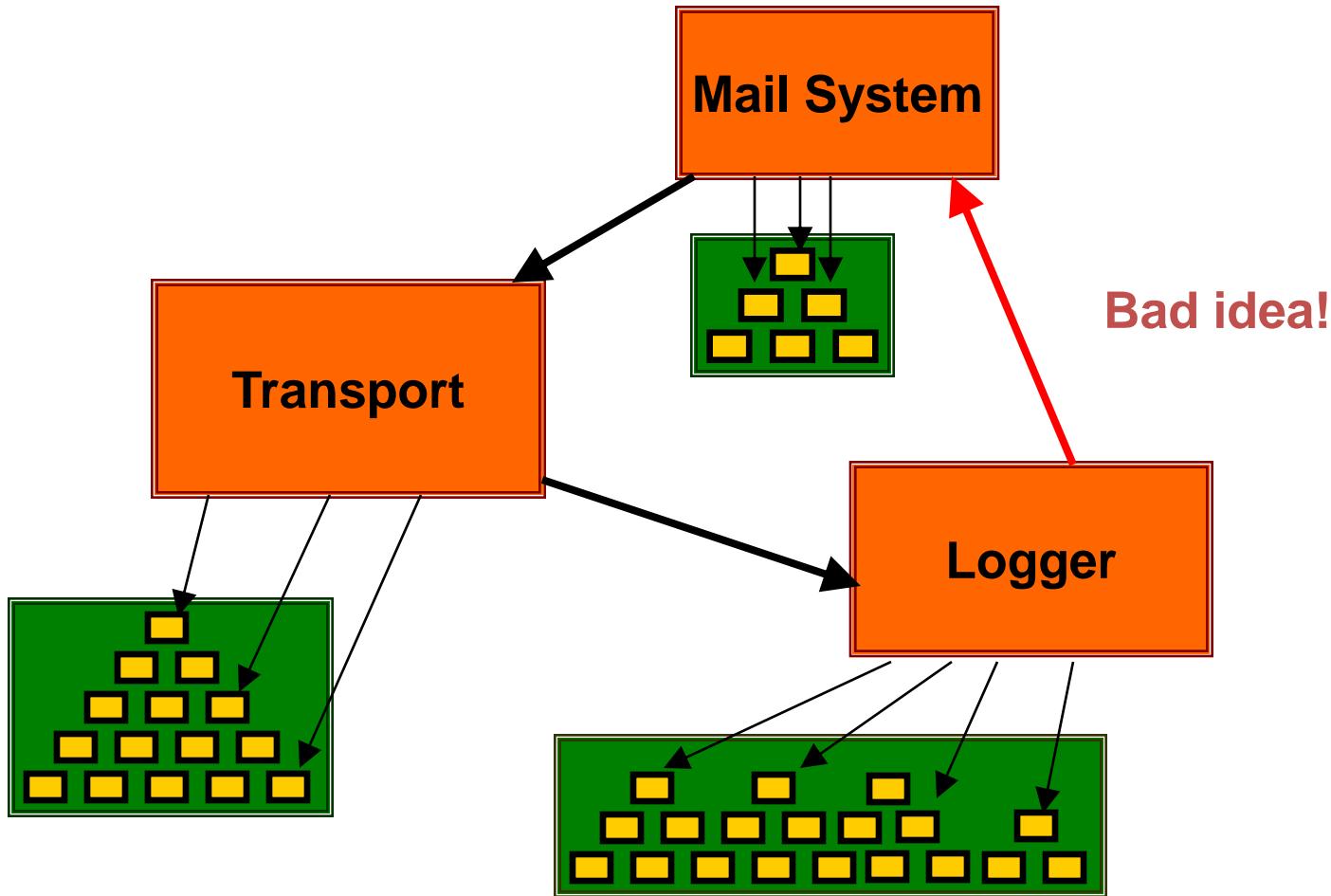
4. Present-Day, Real-World Design Examples

Cyclic Link-time Dependency



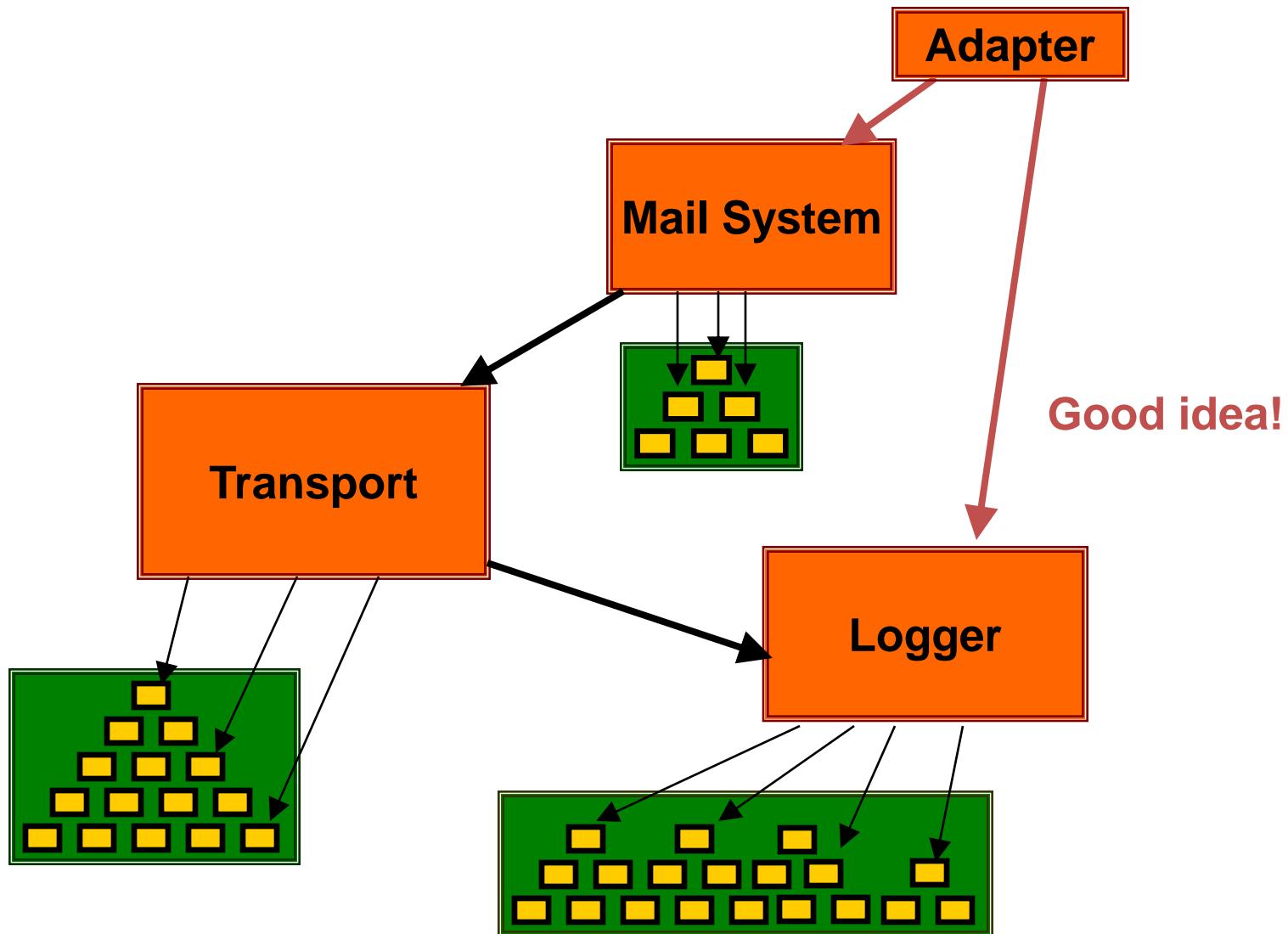
4. Present-Day, Real-World Design Examples

Cyclic Link-time Dependency



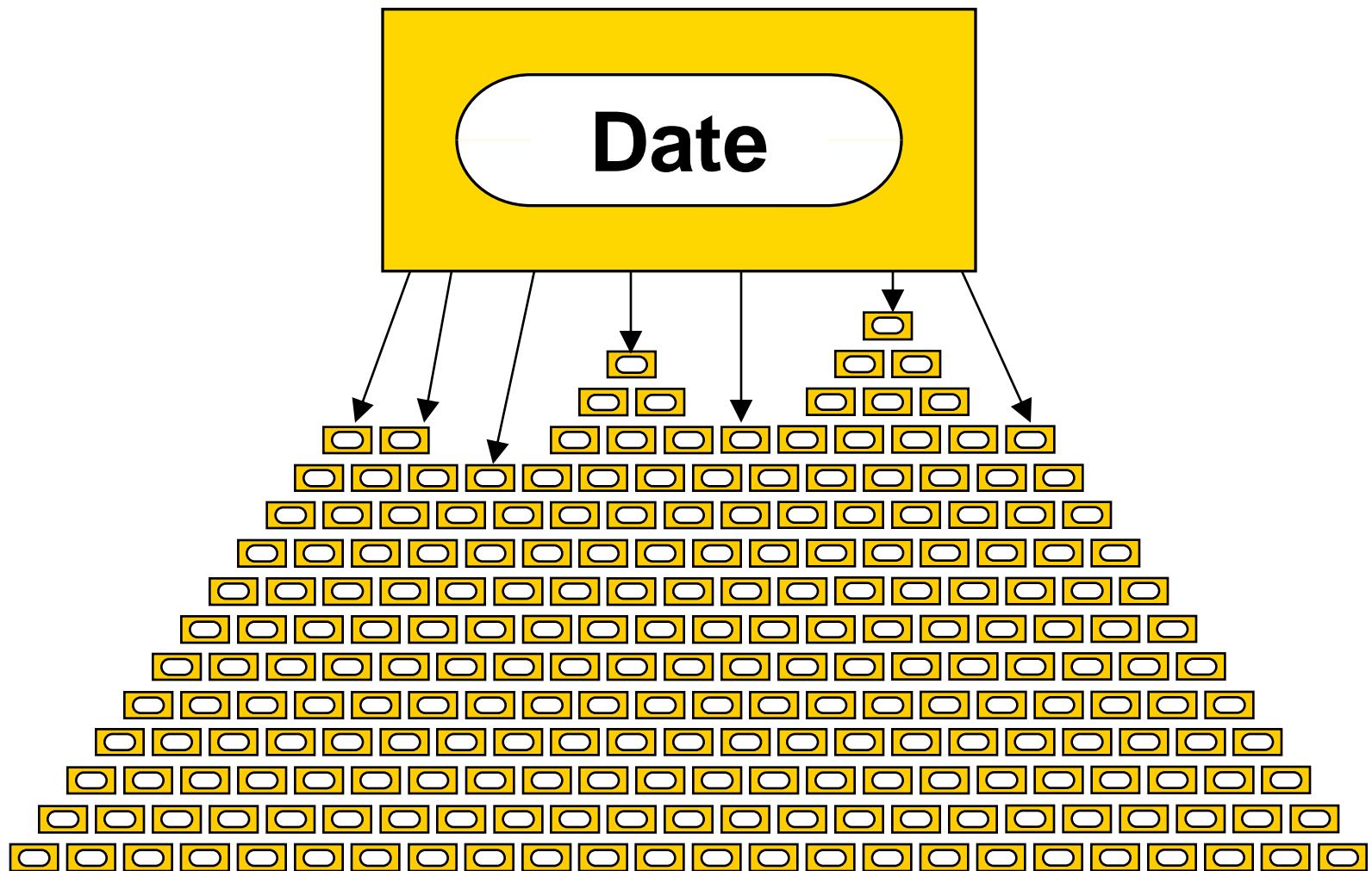
4. Present-Day, Real-World Design Examples

Cyclic Link-time Dependency



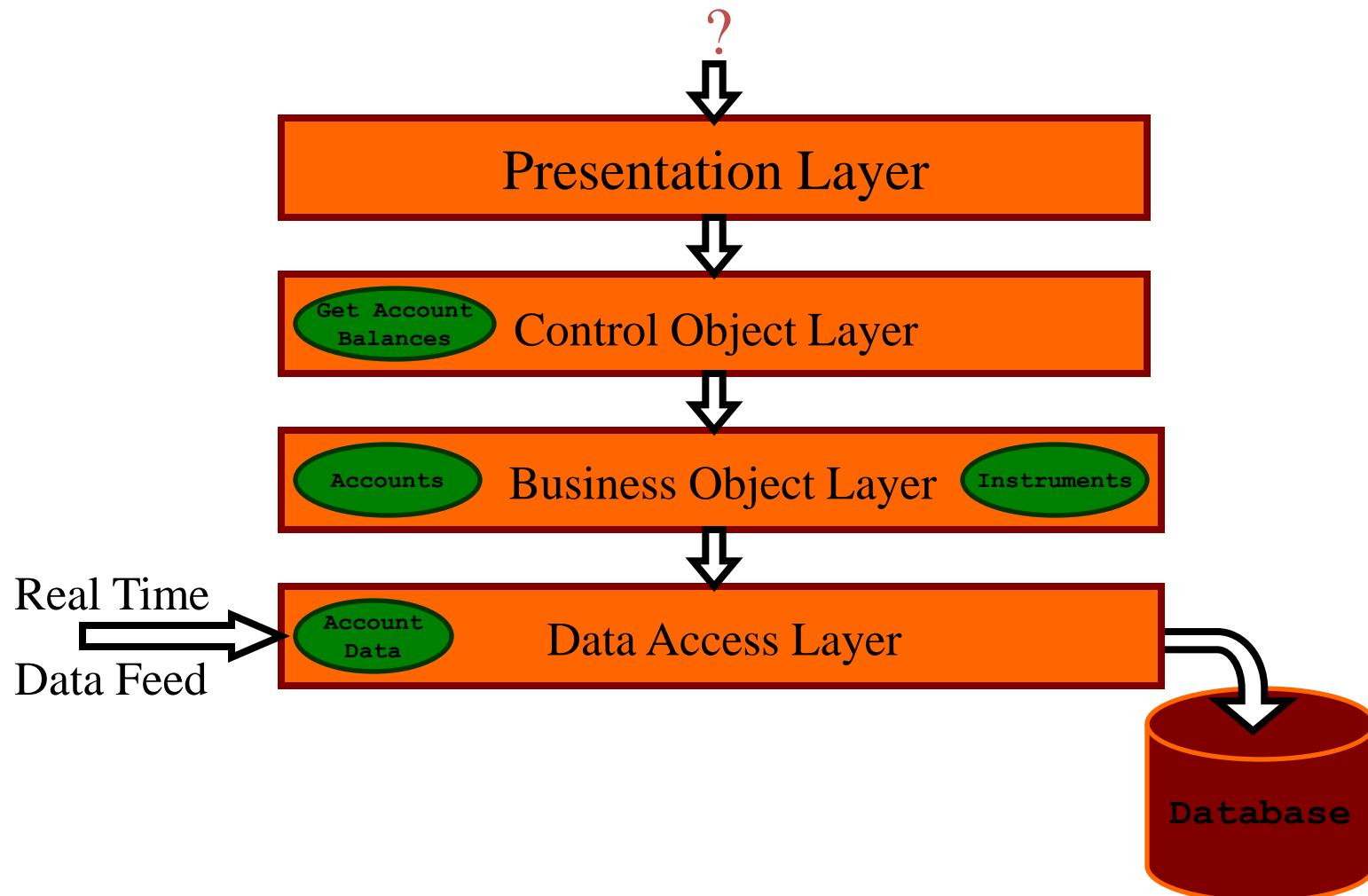
4. Present-Day, Real-World Design Examples

Excessive Link-time Dependency



4. Present-Day, Real-World Design Examples

Classical Layered Architecture

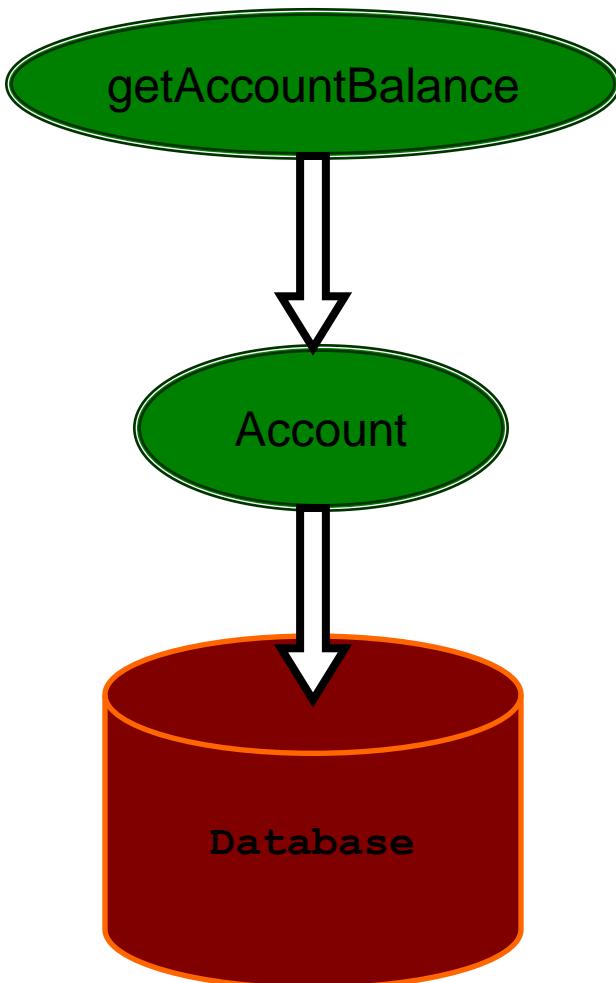


4. Present-Day, Real-World Design Examples

What Does Account Depend On?

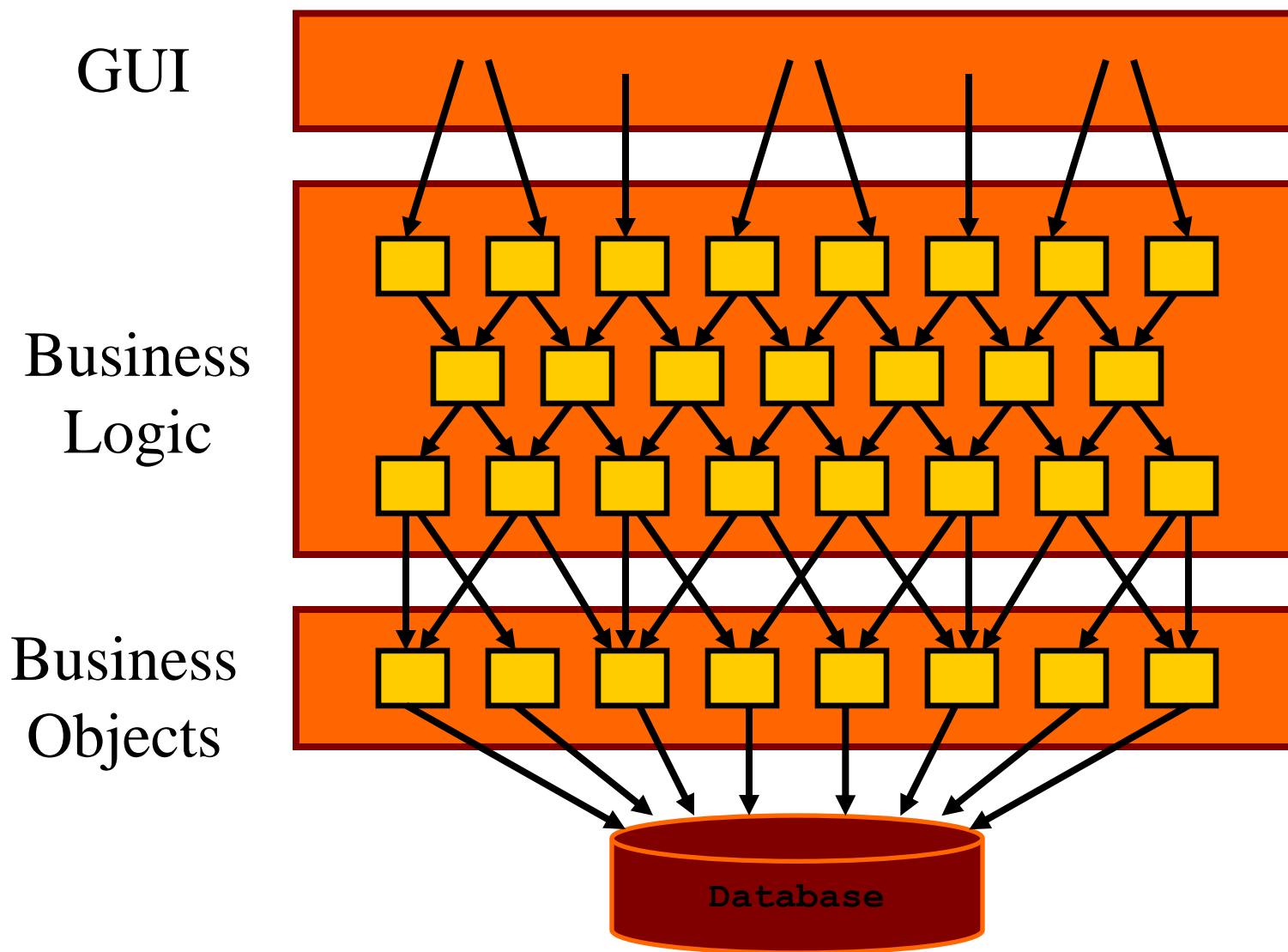
```
class Account {  
    // ...  
public:  
    Account(int accountNumber);  
        // Create an account  
        // corresponding to the  
        // specified 'accountNumber'  
        // in the database.  
    // ...  
};
```

4. Present-Day, Real-World Design Examples On the Database!



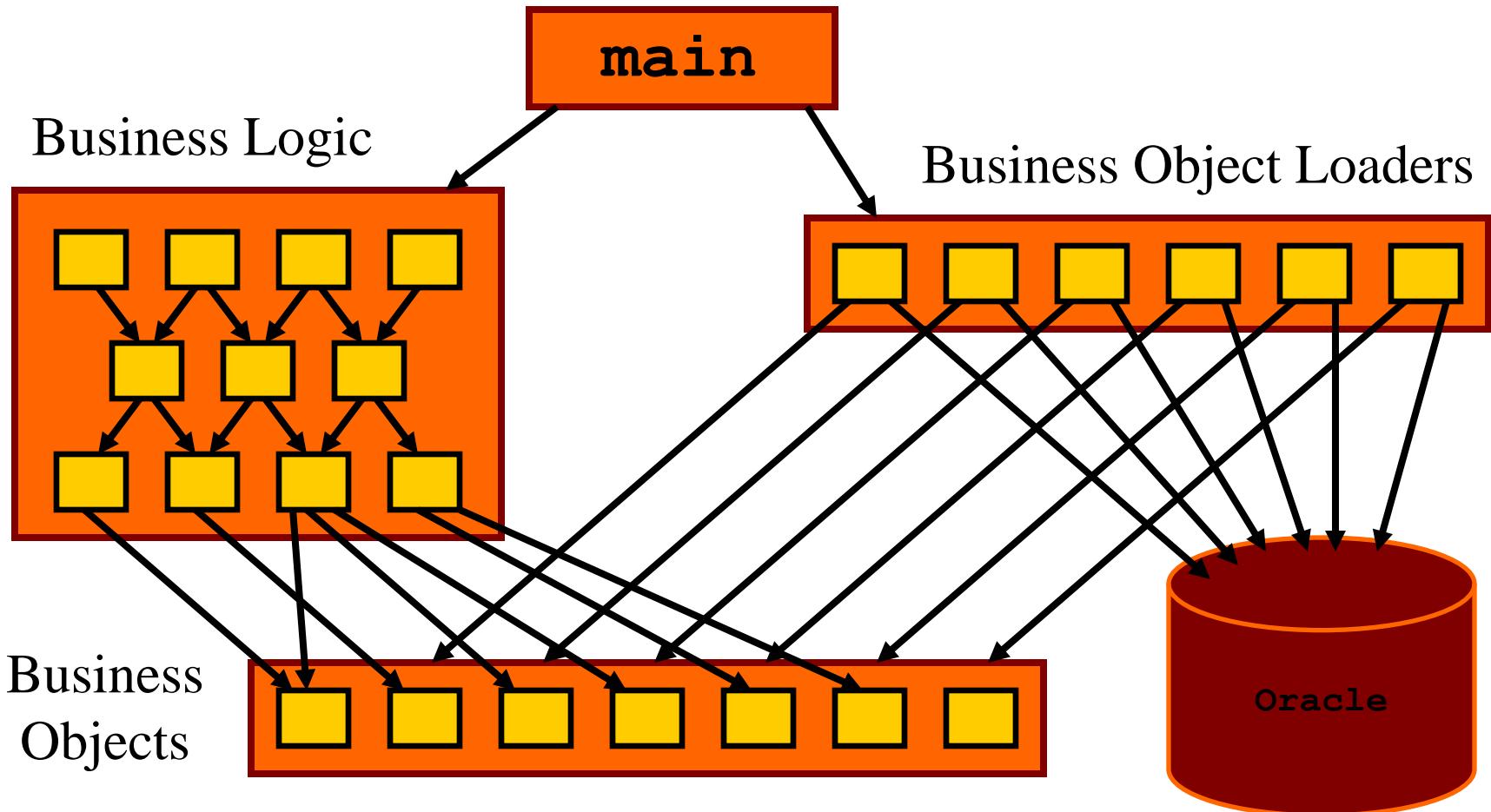
4. Present-Day, Real-World Design Examples

Everything Depends on the Database!



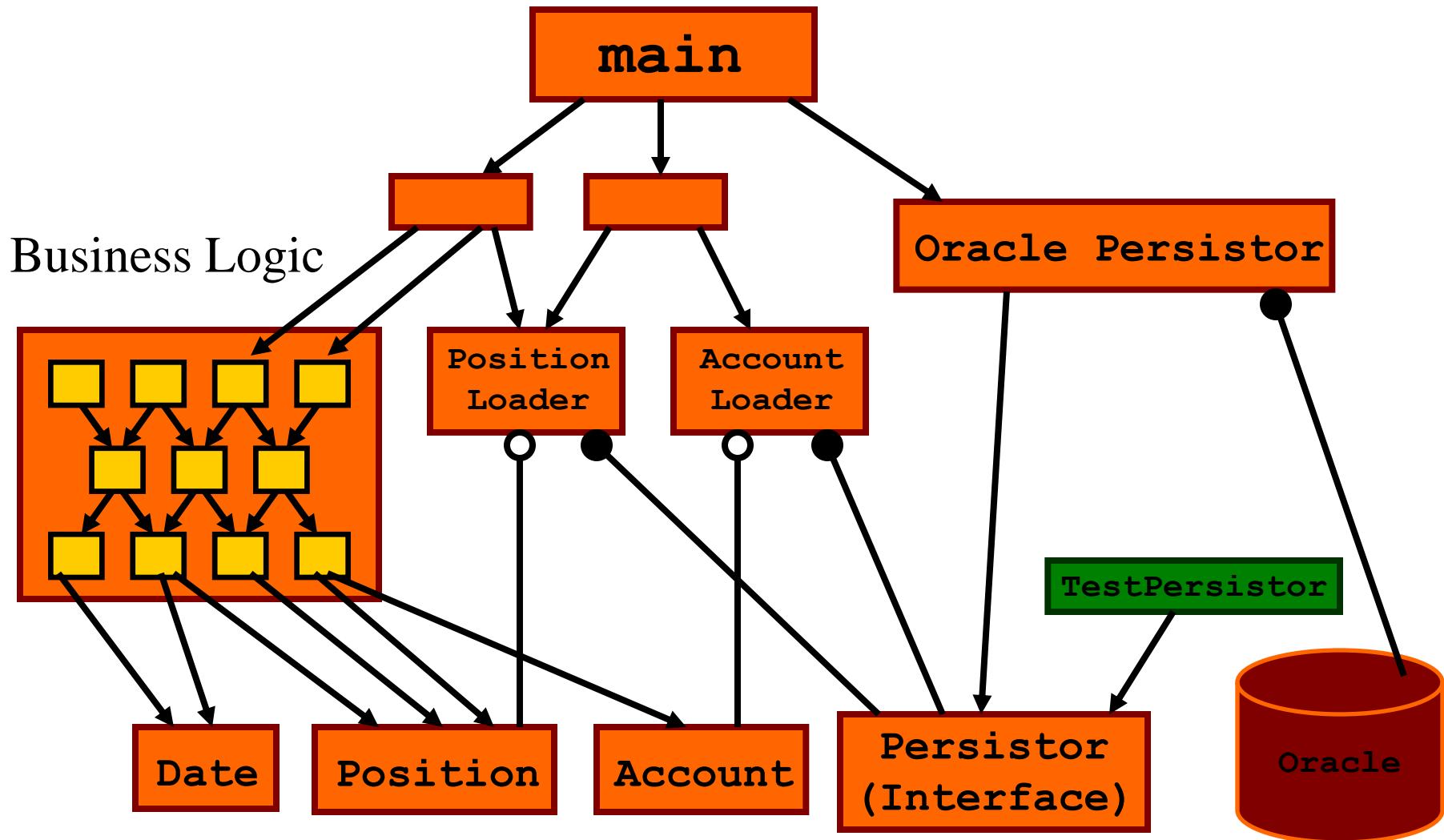
4. Present-Day, Real-World Design Examples

Escalating Heavy-Weight Dependencies



4. Present-Day, Real-World Design Examples

Breaking Dependencies Via Interfaces



4. Present-Day, Real-World Design Examples

A Business Request

Suppose you are asked to provide some business functionality:

"Write me a 'Date' class that tells me whether today is a business day."

4. Present-Day, Real-World Design Examples

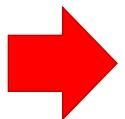
What's the Problem?

"Write me a 'Date' class that tells me whether **today** is a **business day**."

4. Present-Day, Real-World Design Examples

What's the Problem?

"Write me a 'Date' class that tells me whether **today** is a **business day**."

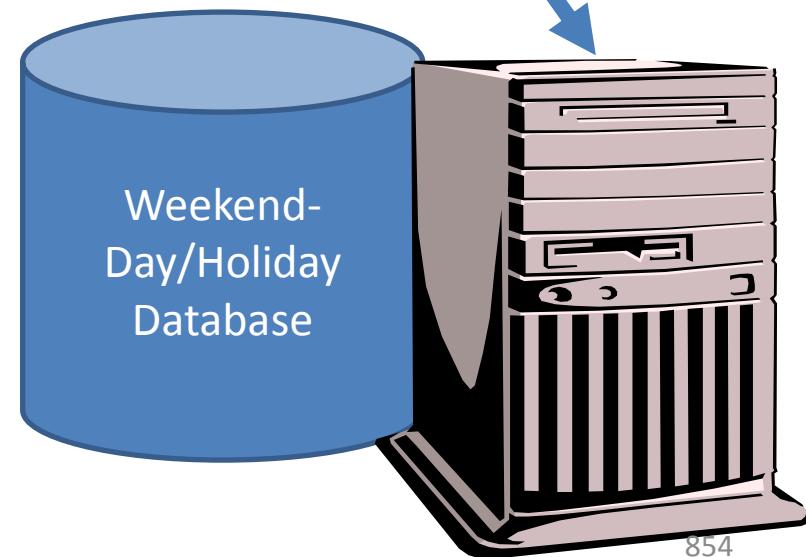
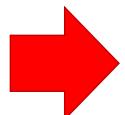


Date

4. Present-Day, Real-World Design Examples

What's the Problem?

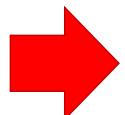
"Write me a 'Date' class that tells me whether **today** is a **business day**."



4. Present-Day, Real-World Design Examples

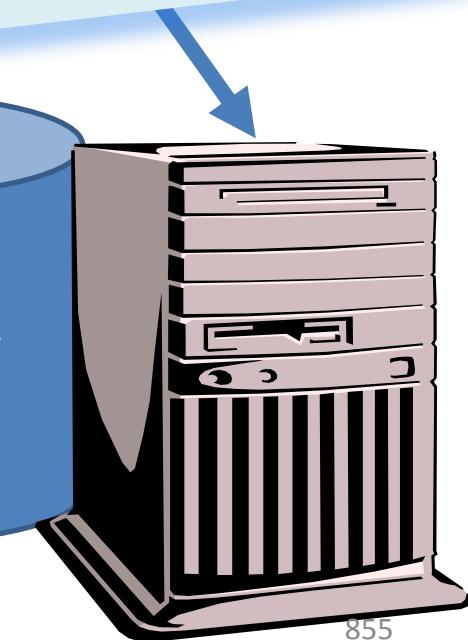
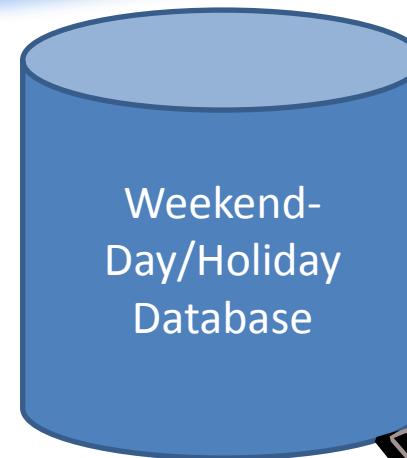
What's the Problem?

"Write me a 'Date' class that tells me whether **today** is a business day."



Date

Poor Logical Factoring



4. Present-Day, Real-World Design Examples

What's the Problem?

"Write me a 'Date' class that tells whether today is business day."

NOT

For Logical Factoring

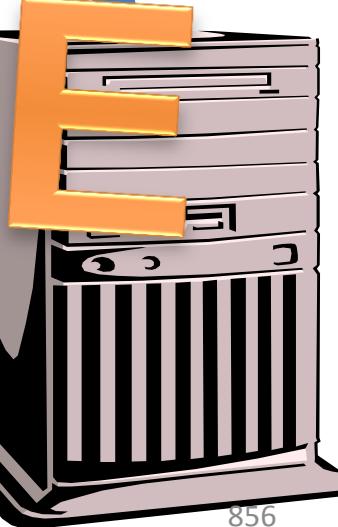
P

FLEXIBLE



FLEXIBLE

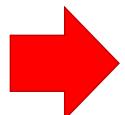
Worker
Day/Holiday
Database



4. Present-Day, Real-World Design Examples

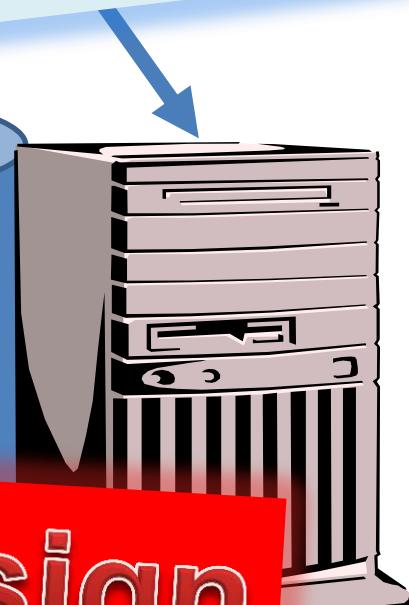
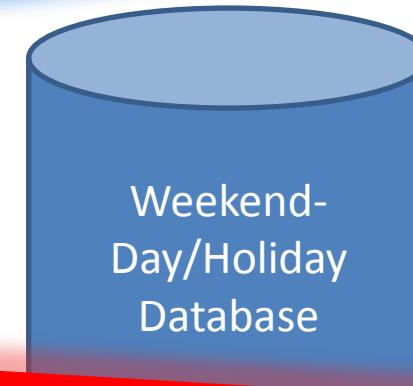
What's the Problem?

"Write me a 'Date' class that tells me whether **today** is a business day."



Date

Poor Logical Factoring



Poor Physical Design

4. Present-Day, Real-World Design Examples

What's the Problem?

"Write me a 'Date' class that tells me whether today is a business day."

NOT

Date

Poor Logical Factoring
MAINTAINABLE

Weekend-
Day/Holiday
Database

Poor Physical Design

4. Present-Day, Real-World Design Examples

The Original Request

"Write me a 'Date' class that tells me whether today is a business day."

What are the *real* requirements?

4. Present-Day, Real-World Design Examples

The Original Request

"Write me a 'Date' class that tells me whether today is a business day."

What are the *real* requirements?

1. Represent a *date value* as a C++ Type.

4. Present-Day, Real-World Design Examples

The Original Request

"Write me a 'Date' class that tells me whether today is a business day."

What are the *real* requirements?

1. Represent a *date value* as a C++ Type.
2. Determine what date value *today* is.

4. Present-Day, Real-World Design Examples

The Original Request

"Write me a 'Date' class that tells me whether today is a business day."

What are the *real* requirements?

1. Represent a *date value* as a C++ Type.
2. Determine what date value *today* is.
3. Determine if a date value is a *business day*.

4. Present-Day, Real-World Design Examples

The Original Request

"Write me a 'Date' class that tells me whether today is a business day."

What are the *real* requirements?

1. Represent a *date value* as a C++ Type.
2. Determine what date value *today* is.
3. Determine if a date value is a *business day*.
4. **Provide well-factored useful components that we'll need over and over again!**

4. Present-Day, Real-World Design Examples

The Original Request

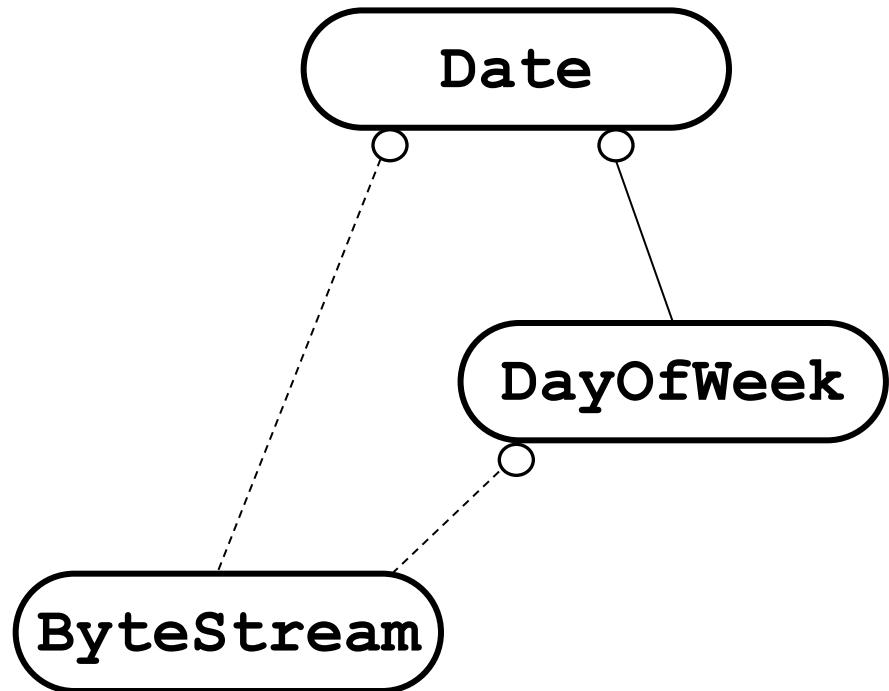
"Write me a 'Date' class that tells me whether today is a business day."

What are the *real* requirements?

1. **Represent a *date value* as a C++ Type.**
2. Determine what date value *today* is.
3. Determine if a date value is a *business day*.
4. **Provide well-factored useful components that we'll need over and over again!**

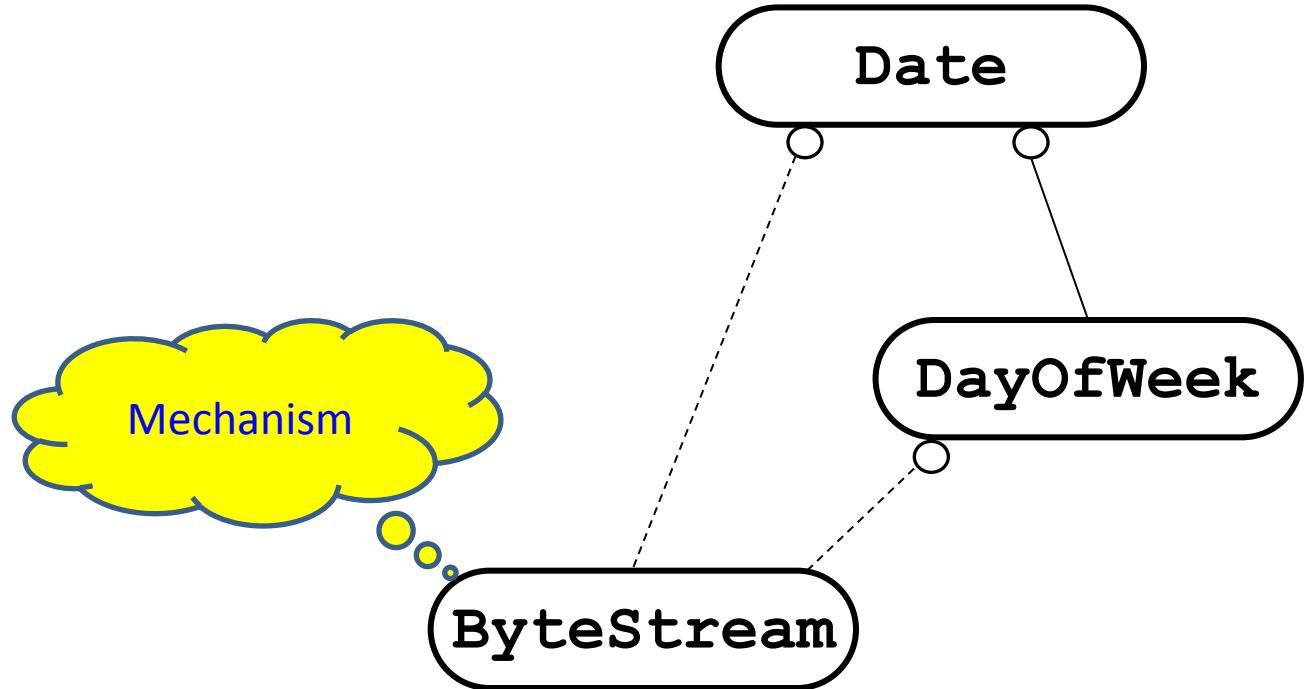
4. Present-Day, Real-World Design Examples

Represent a *Date Value* as a C++ Type



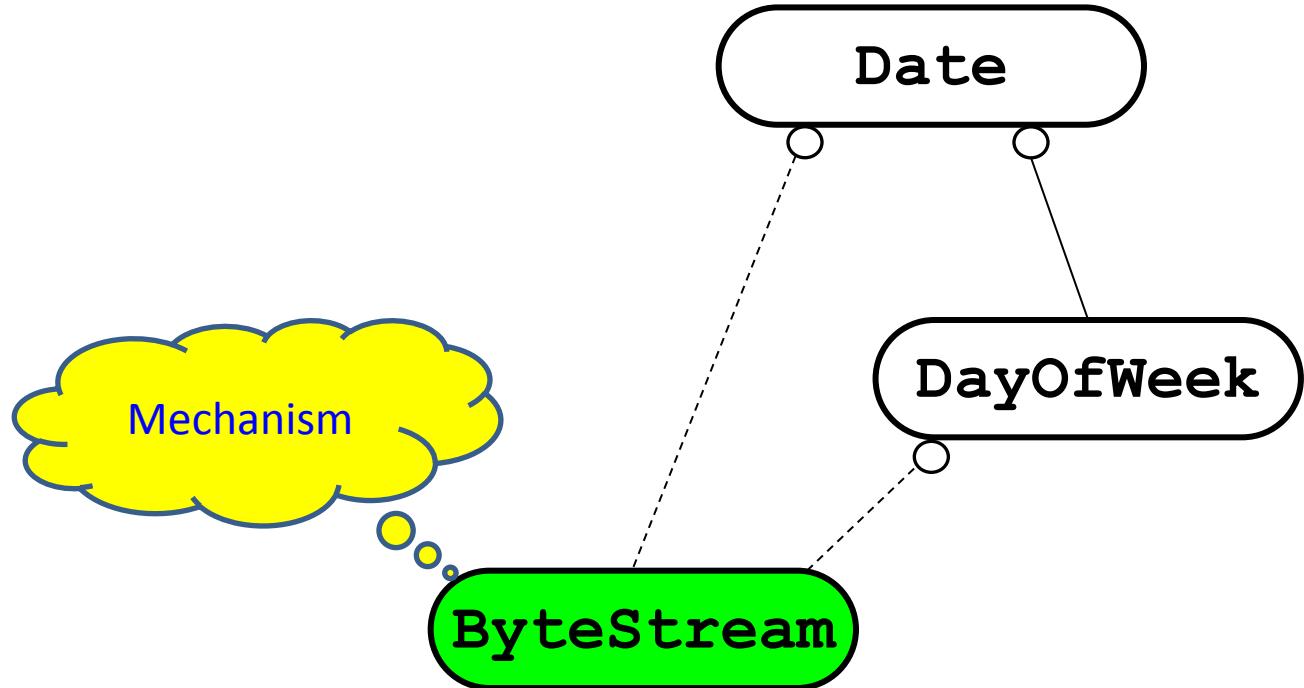
4. Present-Day, Real-World Design Examples

Represent a *Date Value* as a C++ Type



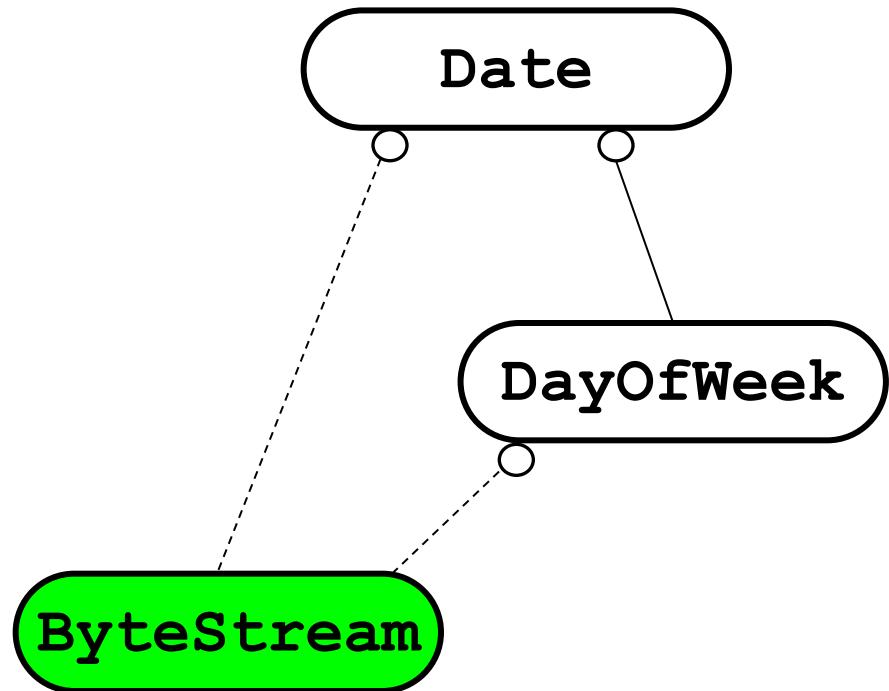
4. Present-Day, Real-World Design Examples

Represent a *Date Value* as a C++ Type



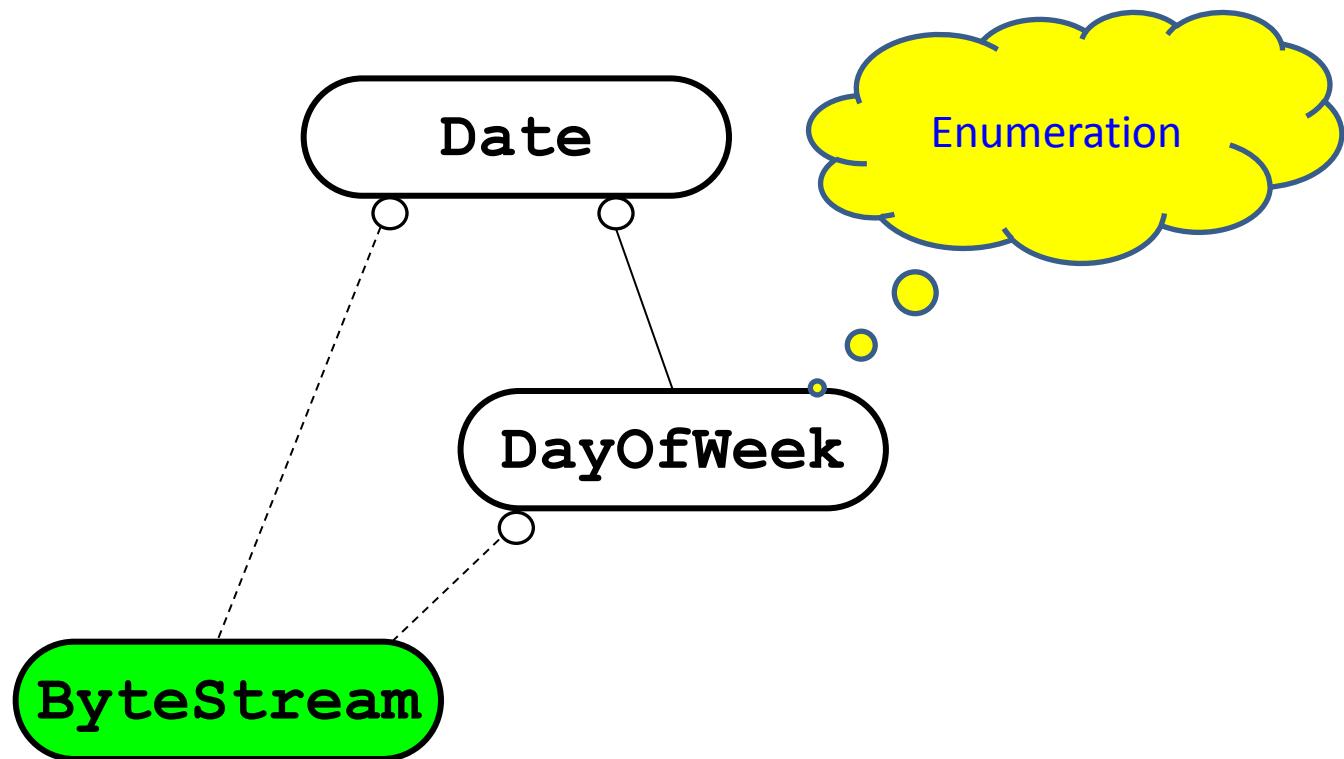
4. Present-Day, Real-World Design Examples

Represent a *Date Value* as a C++ Type



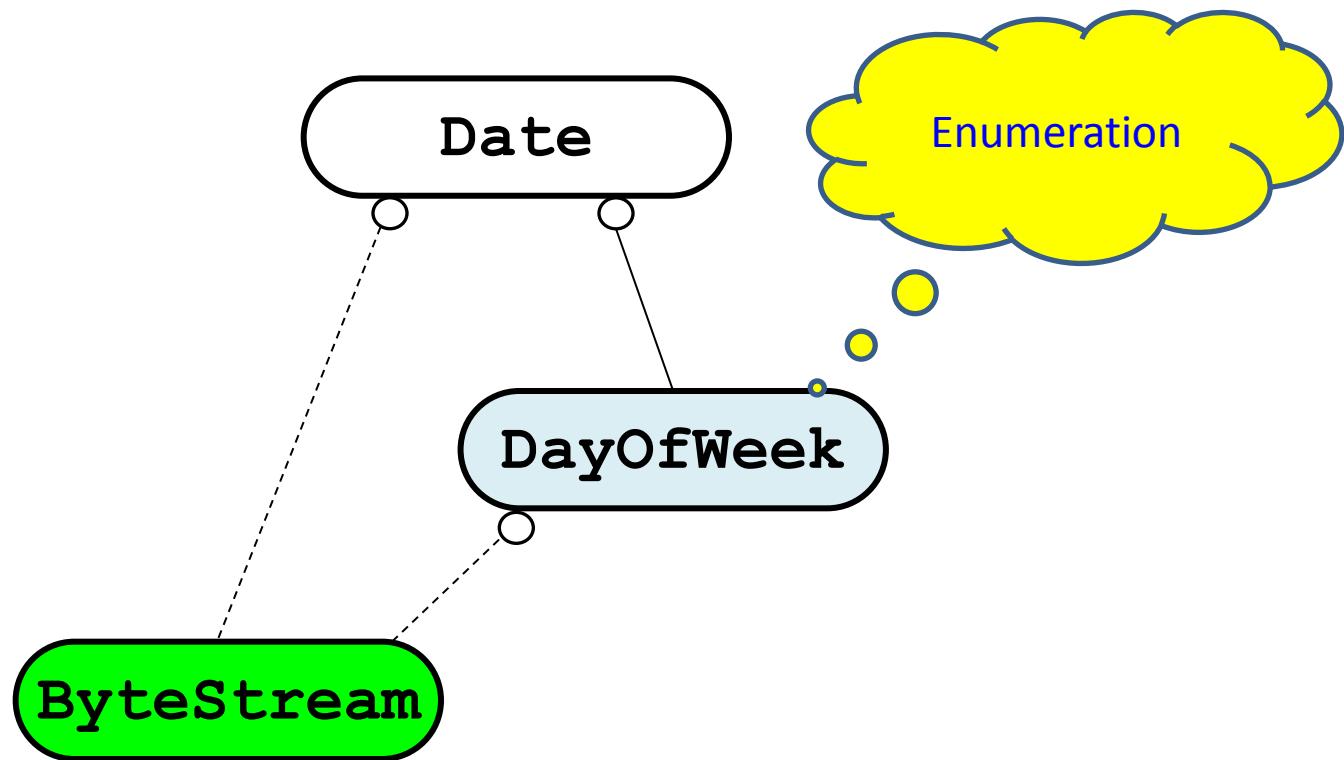
4. Present-Day, Real-World Design Examples

Represent a *Date Value* as a C++ Type



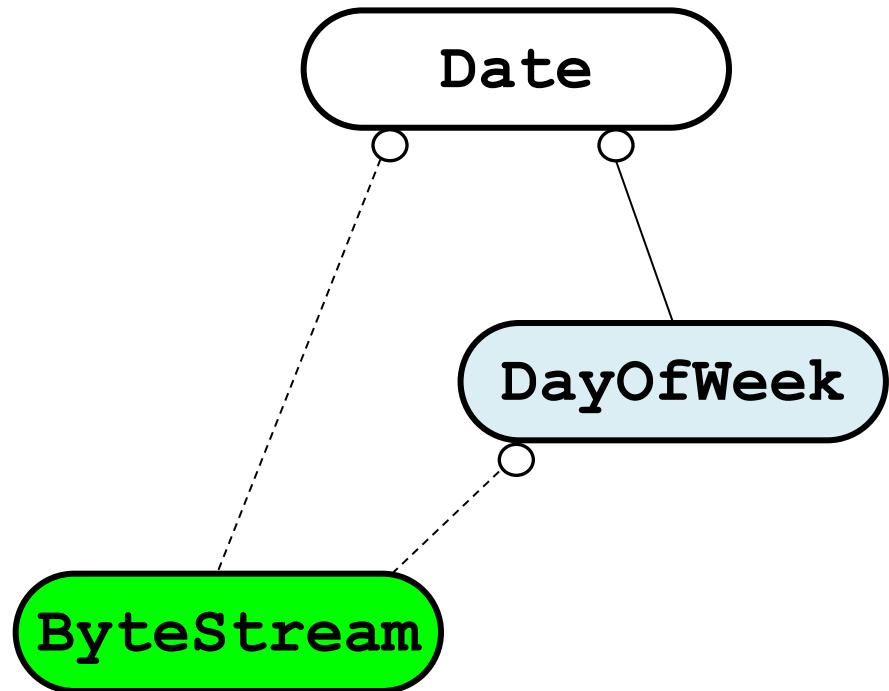
4. Present-Day, Real-World Design Examples

Represent a *Date Value* as a C++ Type



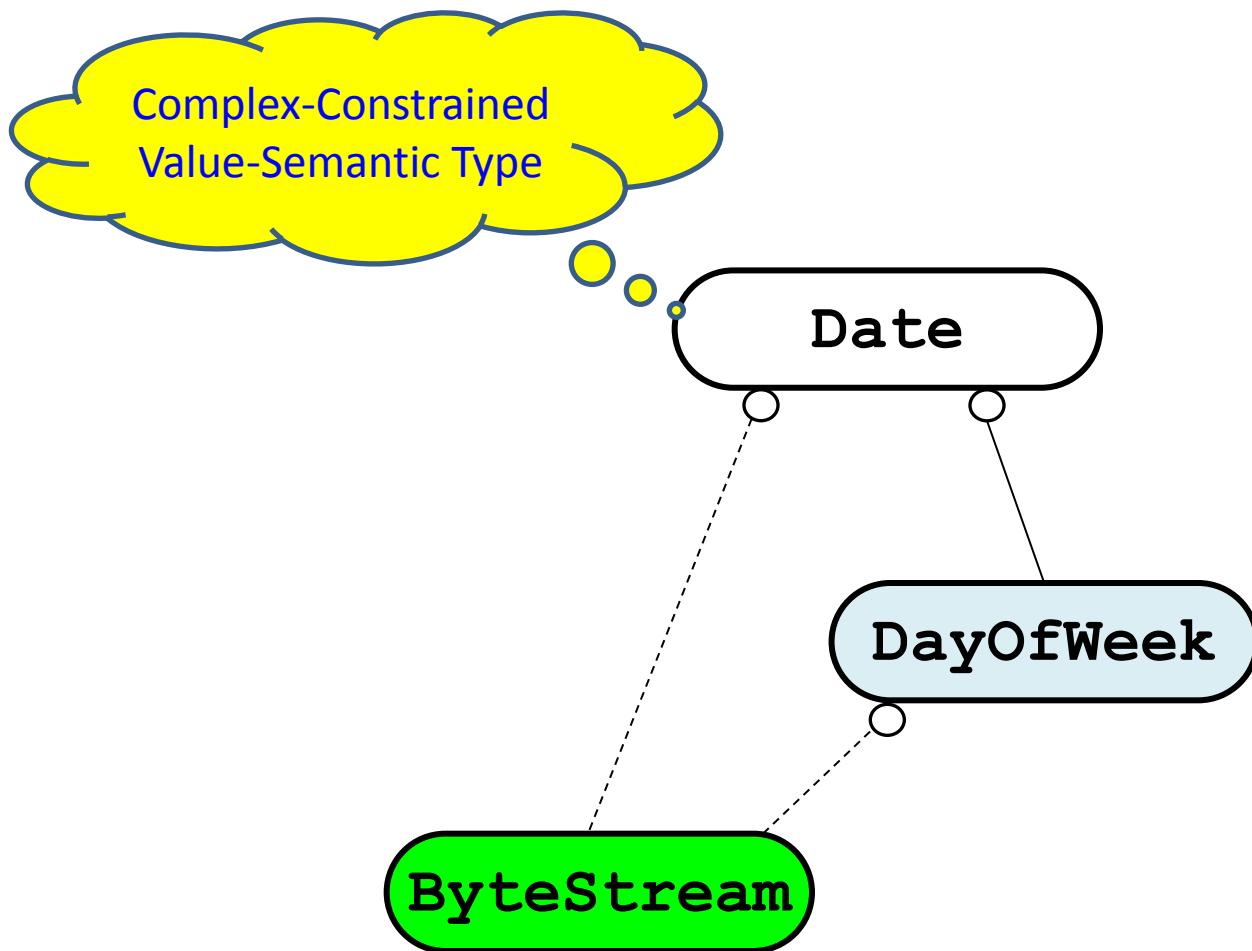
4. Present-Day, Real-World Design Examples

Represent a *Date Value* as a C++ Type



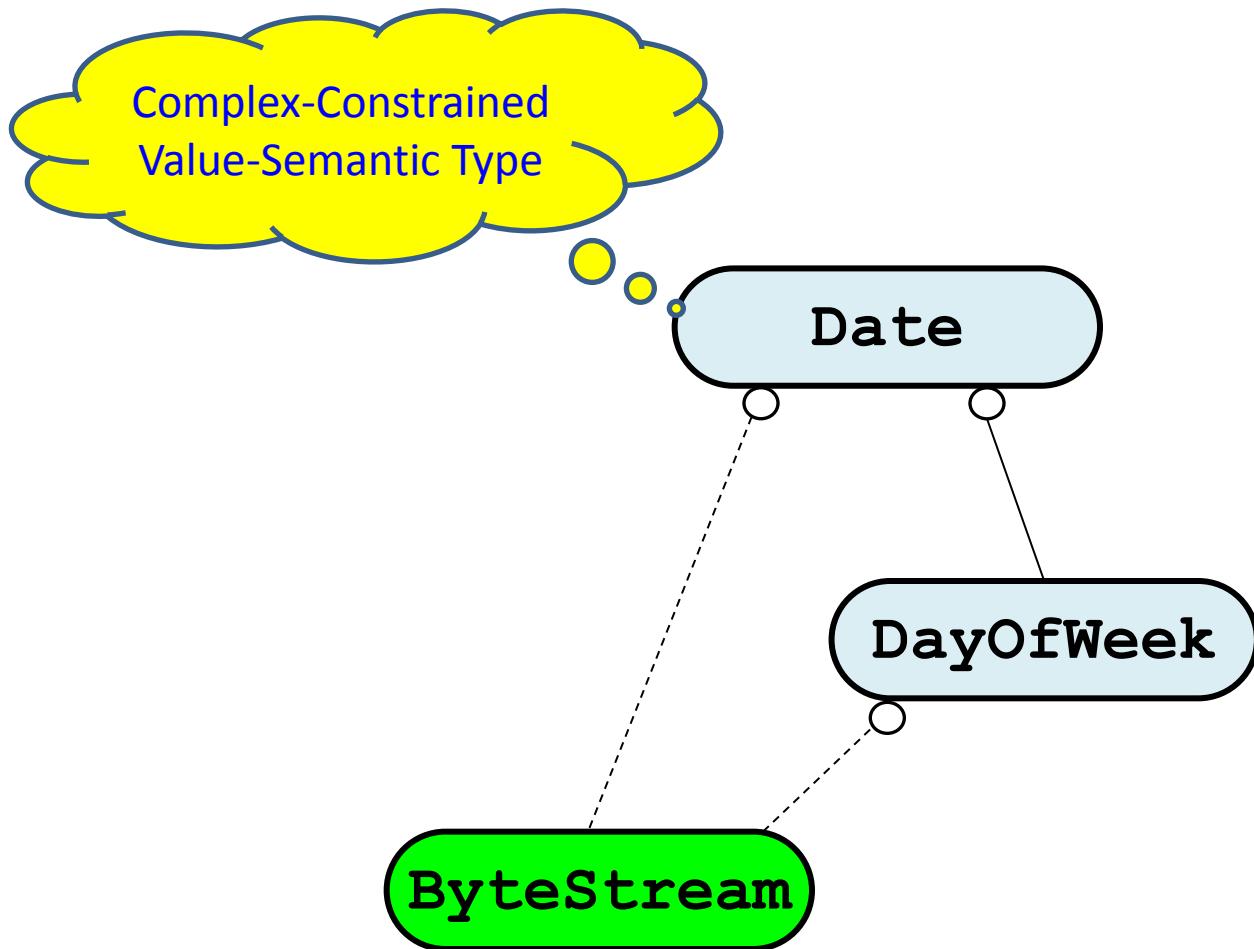
4. Present-Day, Real-World Design Examples

Represent a *Date Value* as a C++ Type



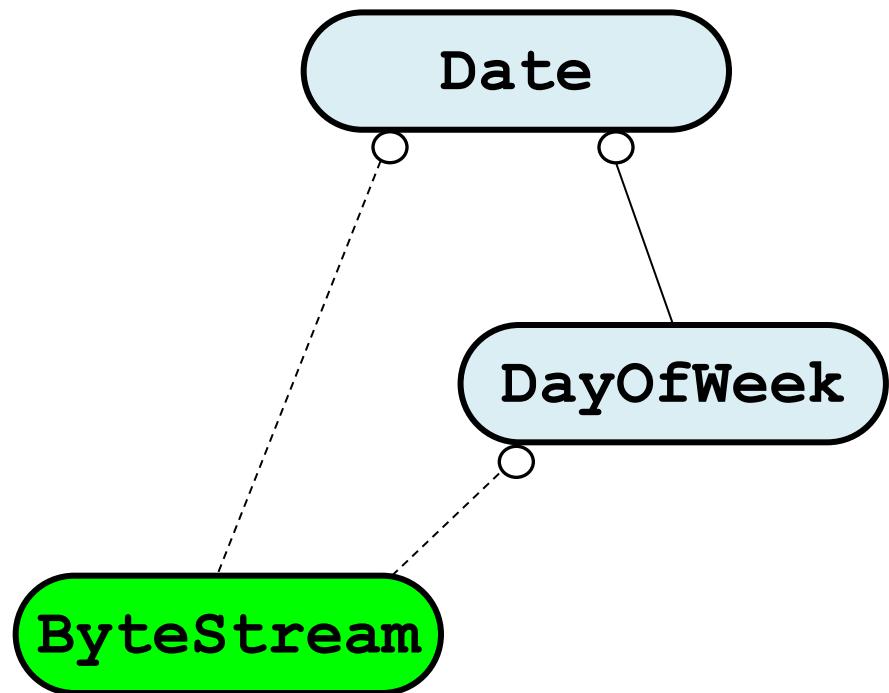
4. Present-Day, Real-World Design Examples

Represent a *Date Value* as a C++ Type



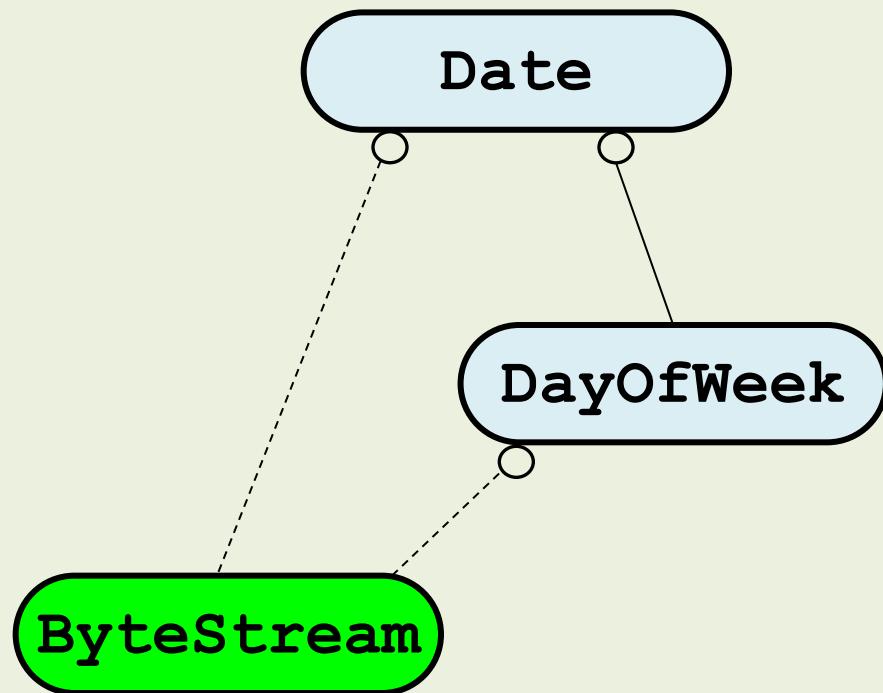
4. Present-Day, Real-World Design Examples

Represent a *Date Value* as a C++ Type



4. Present-Day, Real-World Design Examples

Solution 1: Represent a Date Value



4. Present-Day, Real-World Design Examples

The Original Request

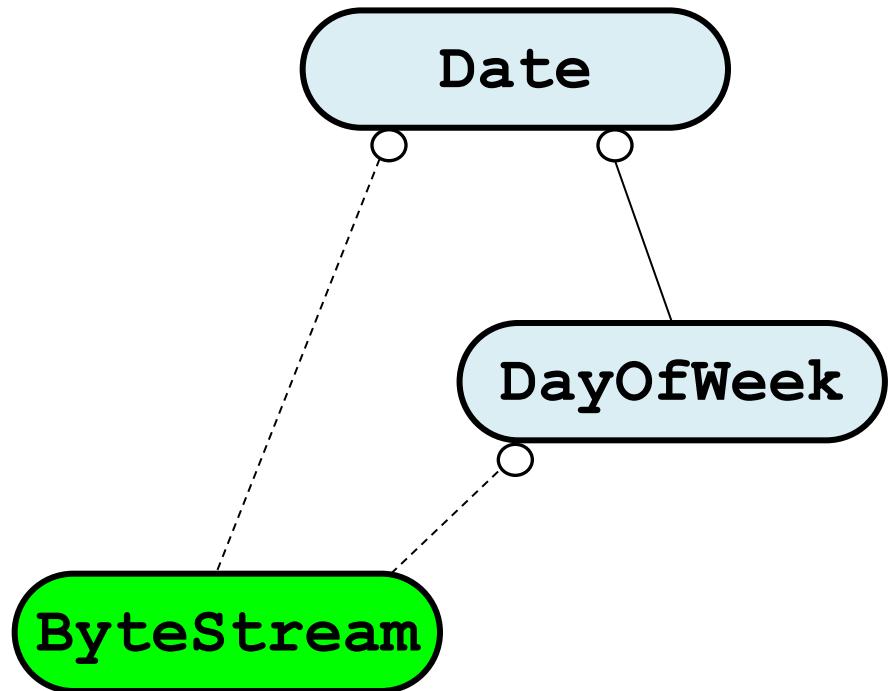
"Write me a 'Date' class that tells me whether today is a business day."

What are the *real* requirements?

1. Represent a *date value* as a C++ Type.
2. **Determine what date value *today* is.**
3. Determine if a date value is a *business day*.
4. **Provide well-factored useful components that we'll need over and over again!**

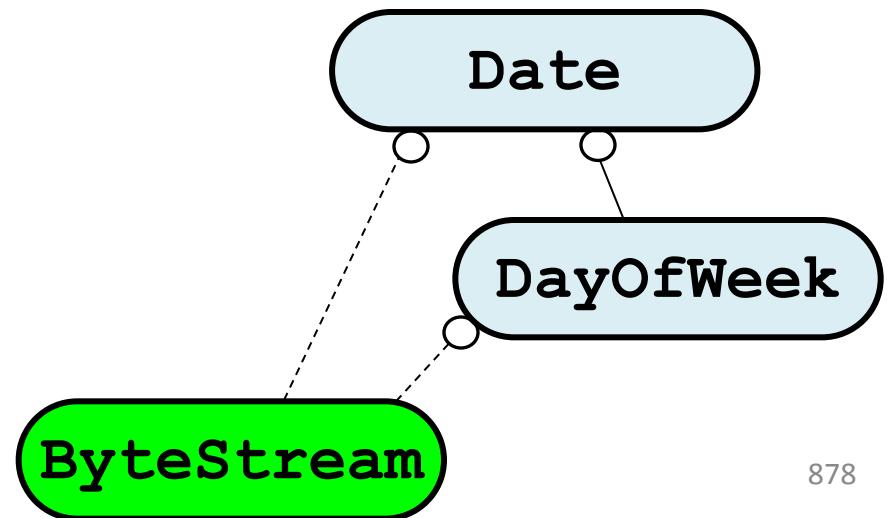
4. Present-Day, Real-World Design Examples

Determine what Date Value today is



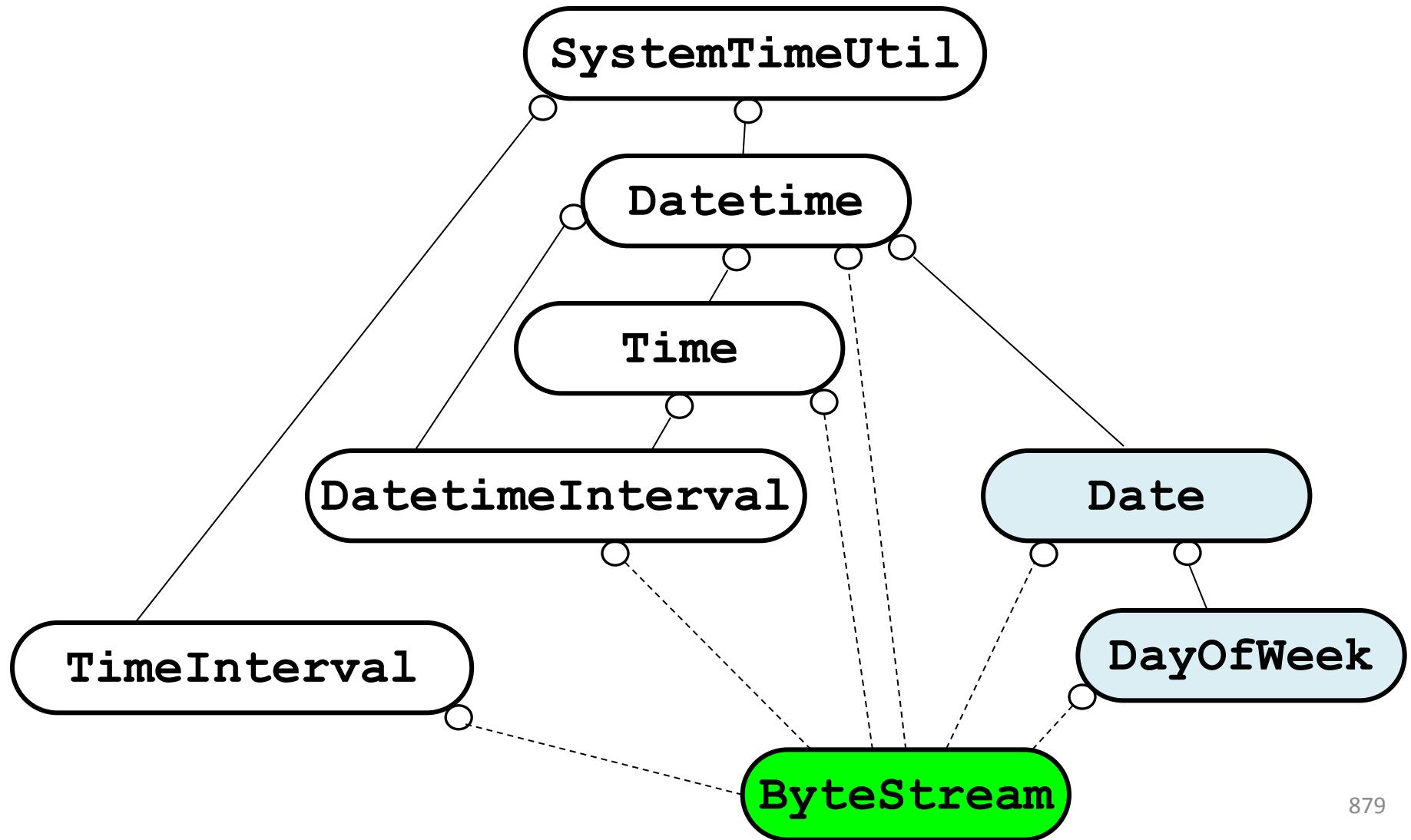
4. Present-Day, Real-World Design Examples

Determine what Date Value today is



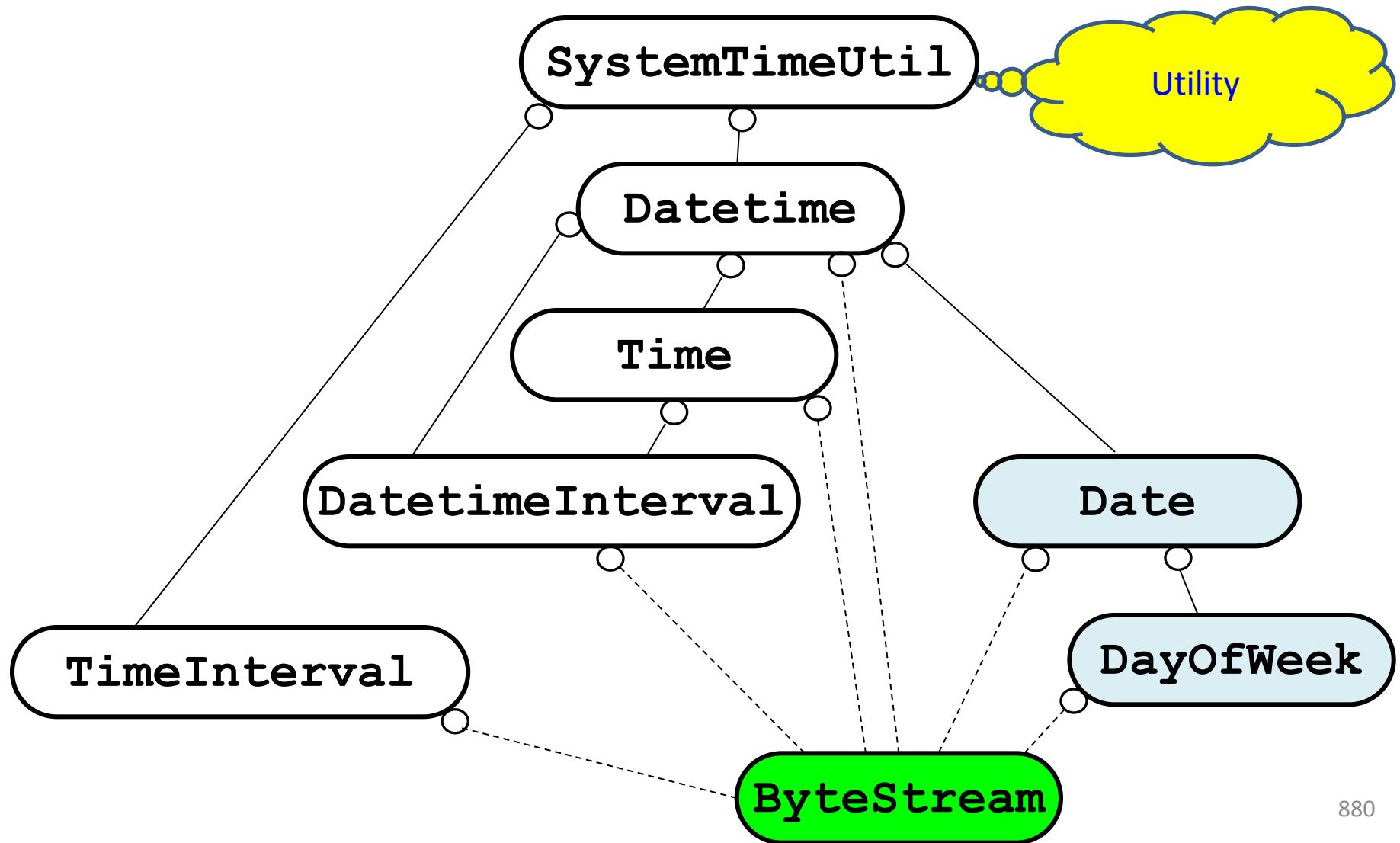
4. Present-Day, Real-World Design Examples

Determine what Date Value today is



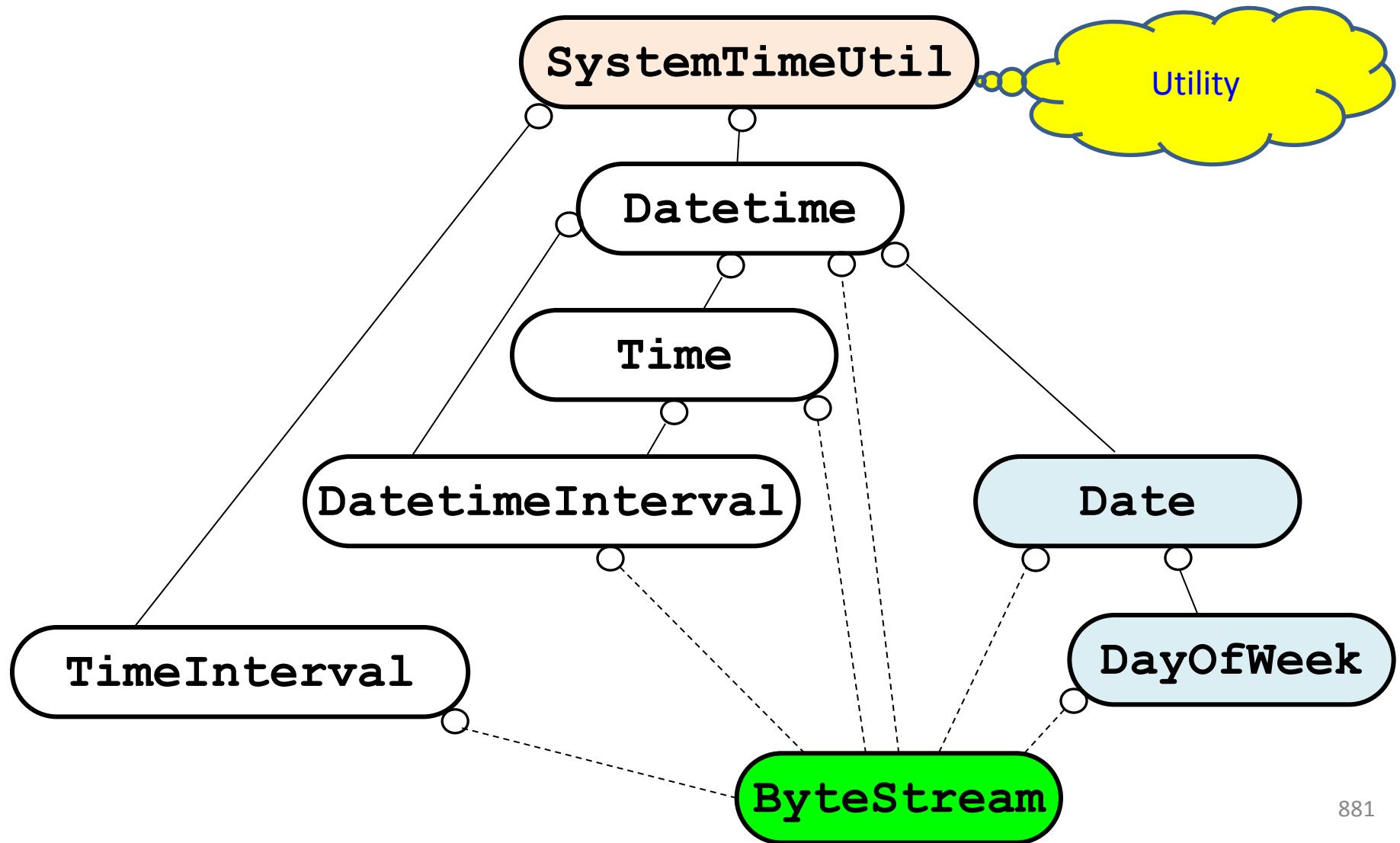
4. Present-Day, Real-World Design Examples

Determine what Date Value today is



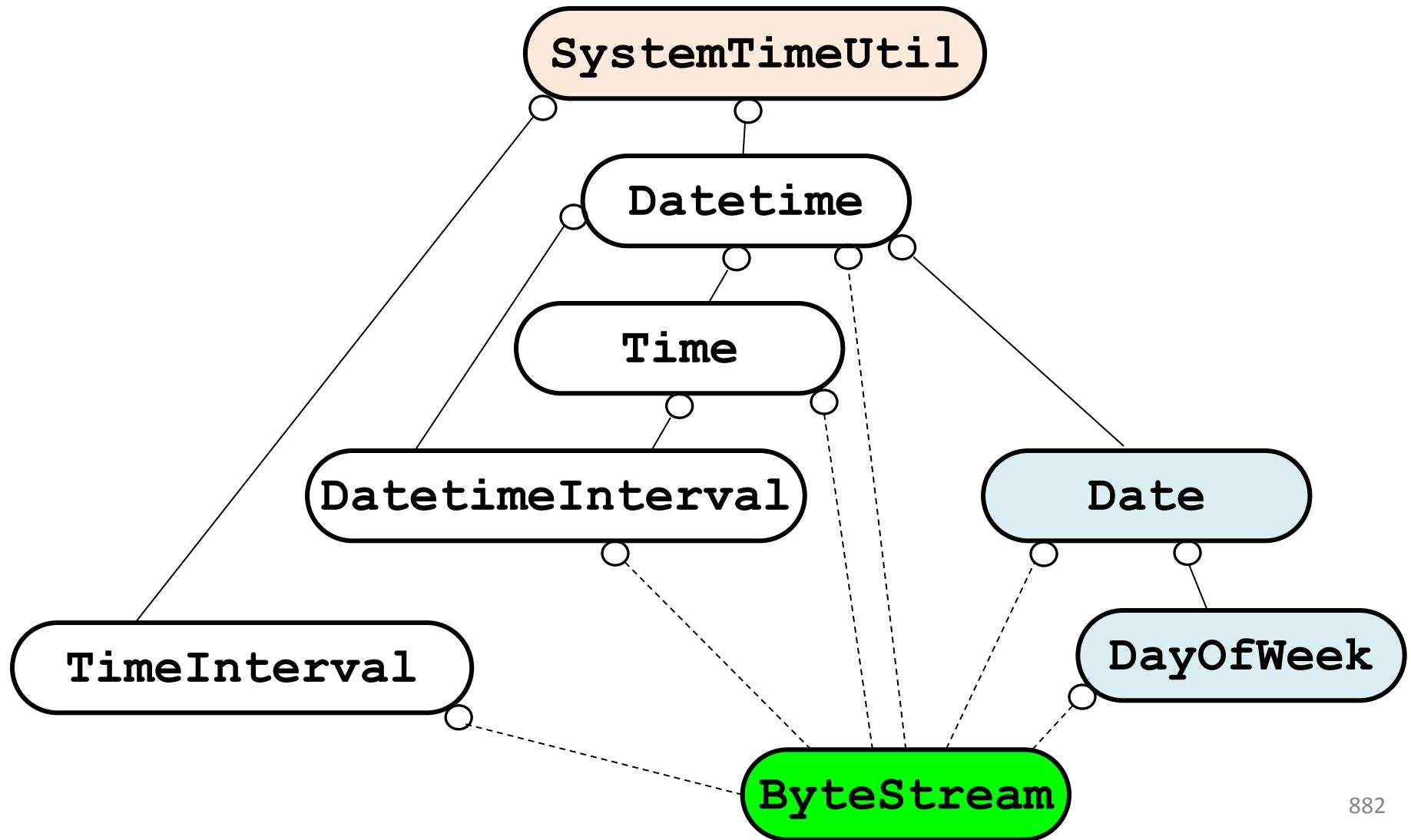
4. Present-Day, Real-World Design Examples

Determine what Date Value today is



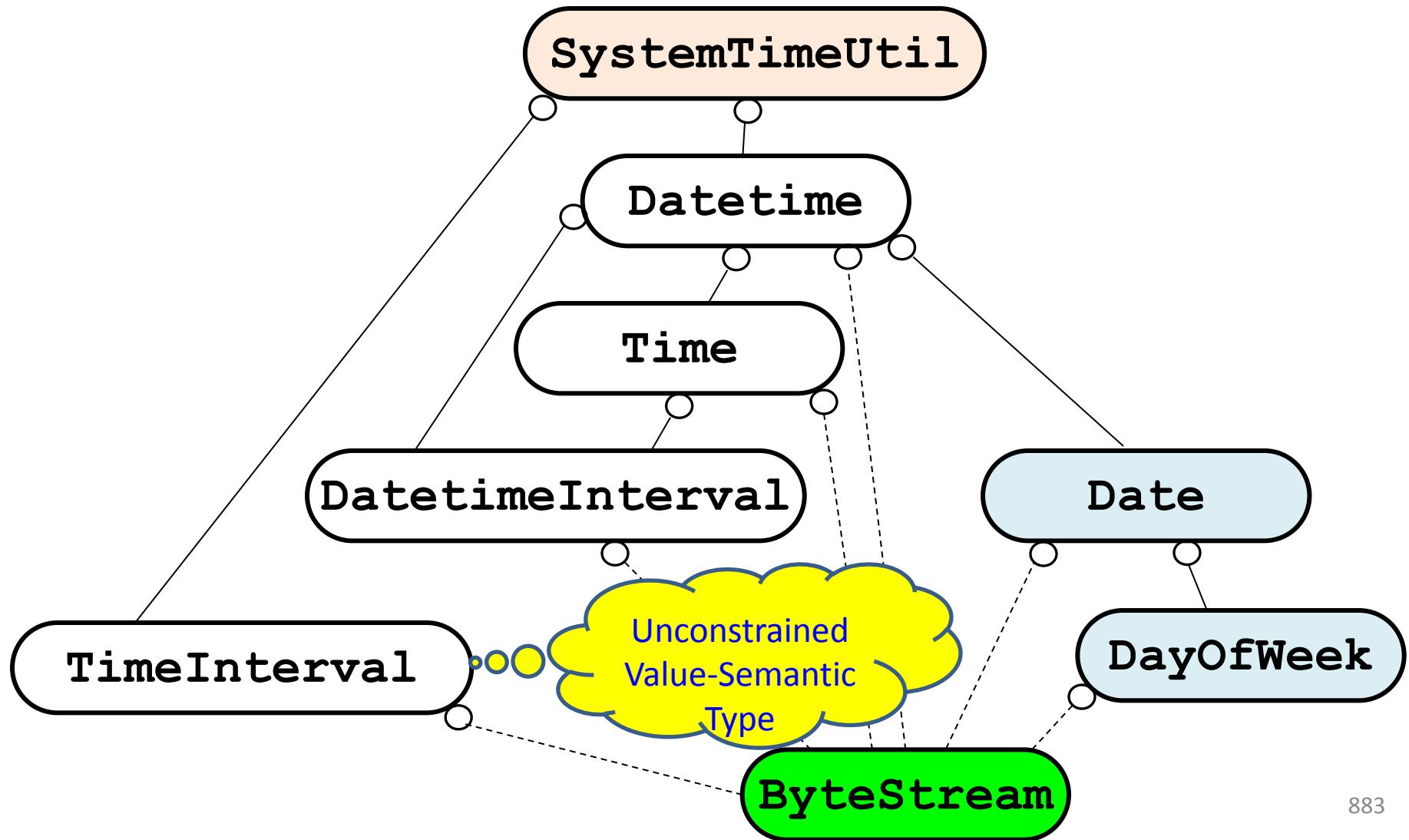
4. Present-Day, Real-World Design Examples

Determine what Date Value today is



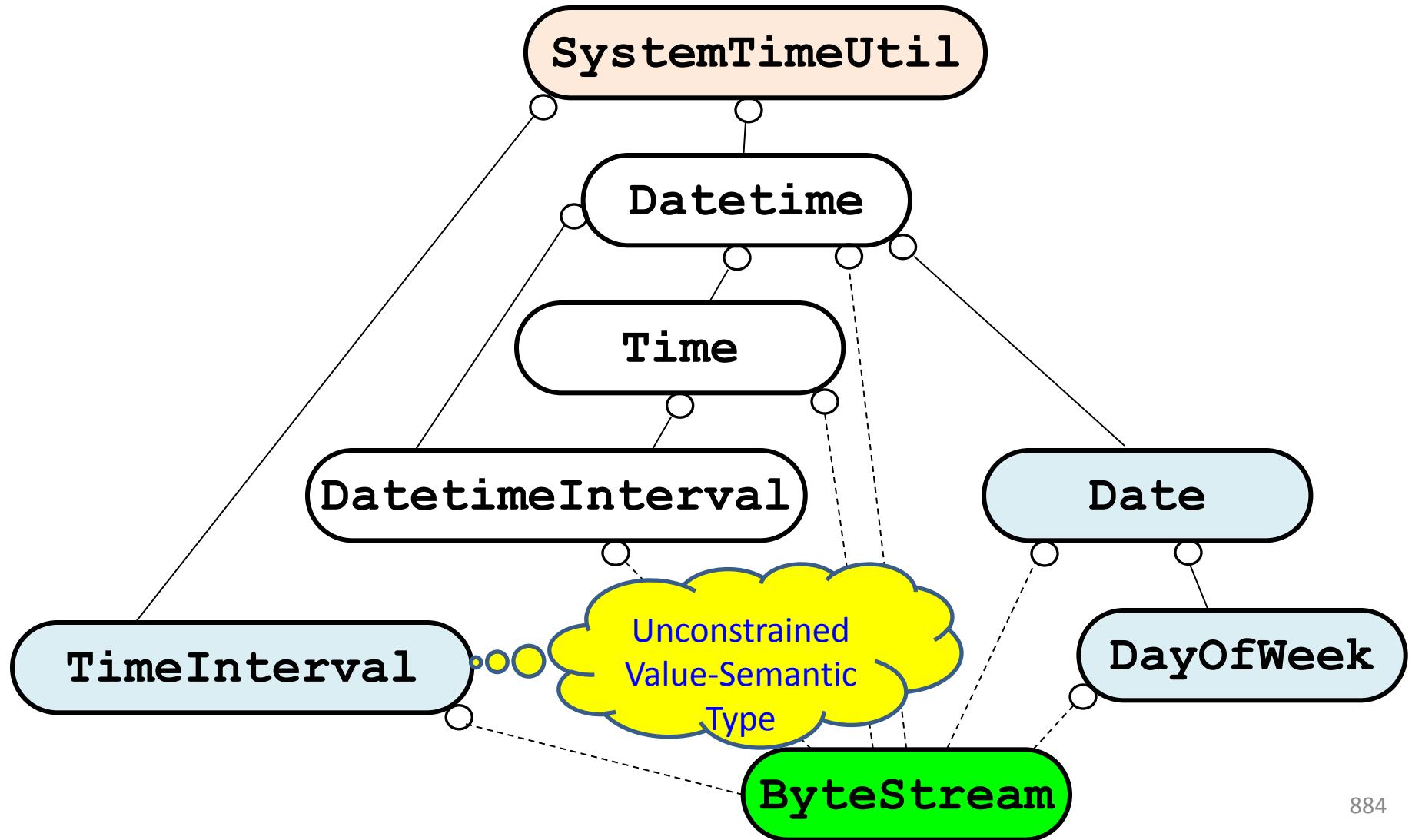
4. Present-Day, Real-World Design Examples

Determine what Date Value today is



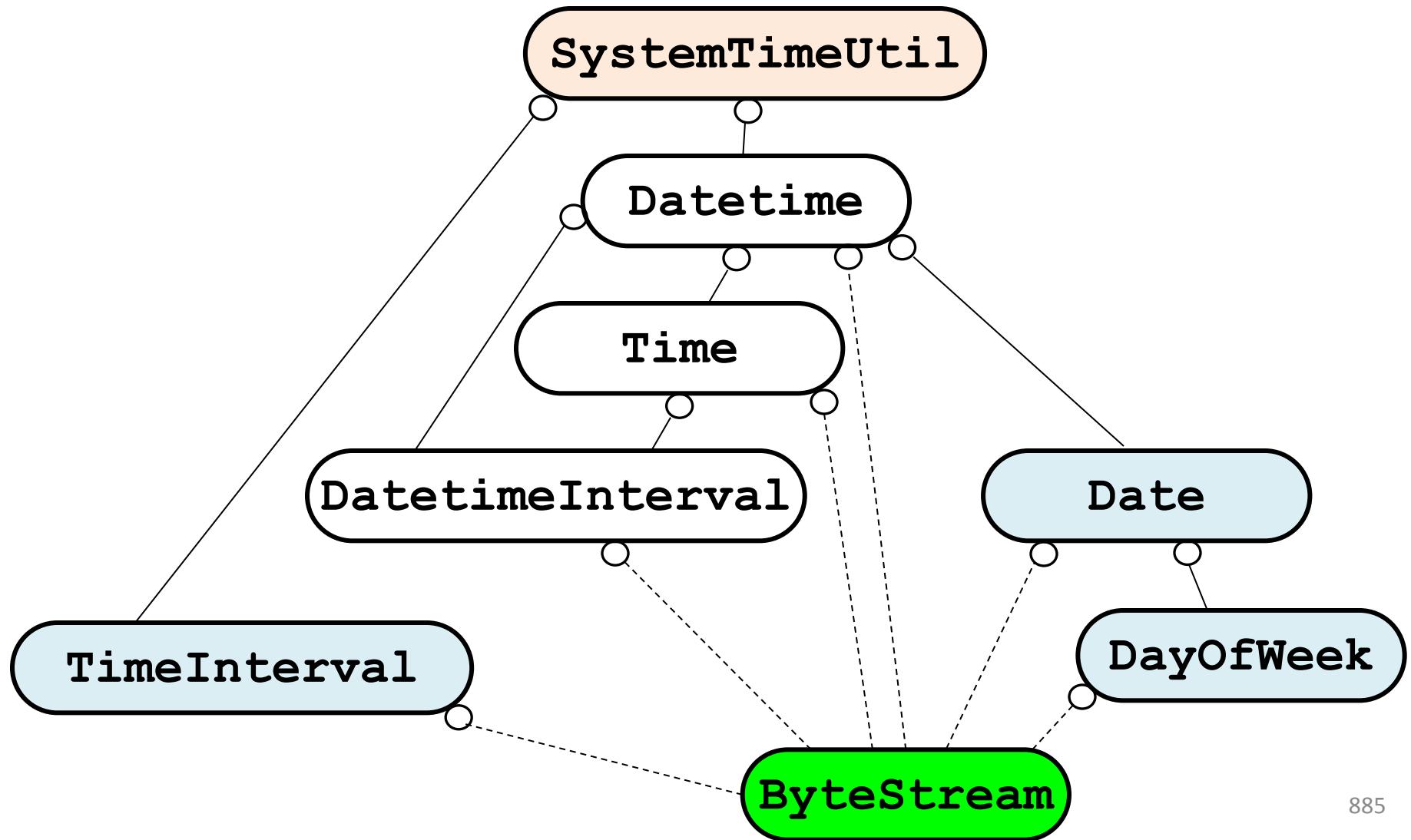
4. Present-Day, Real-World Design Examples

Determine what Date Value today is



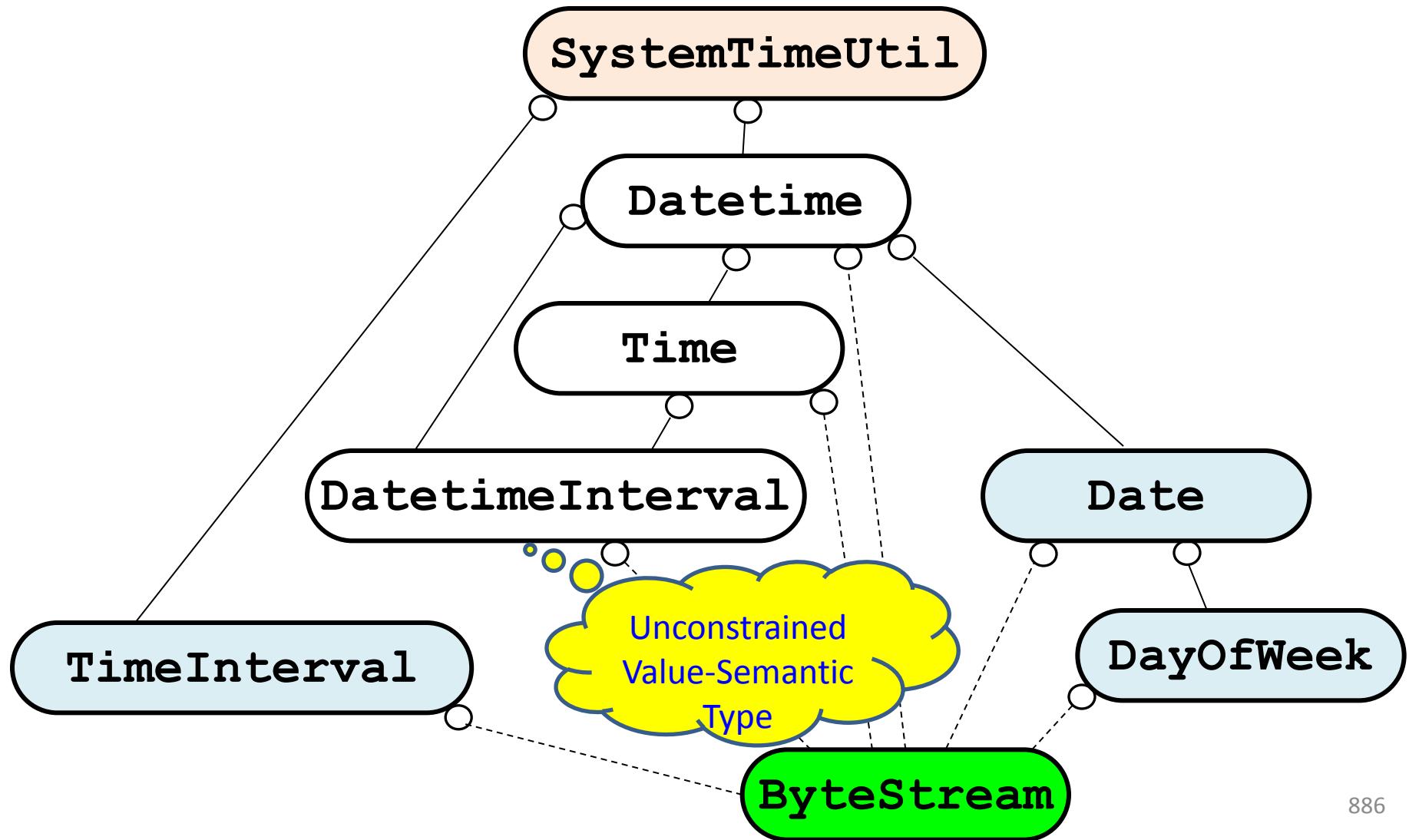
4. Present-Day, Real-World Design Examples

Determine what Date Value today is



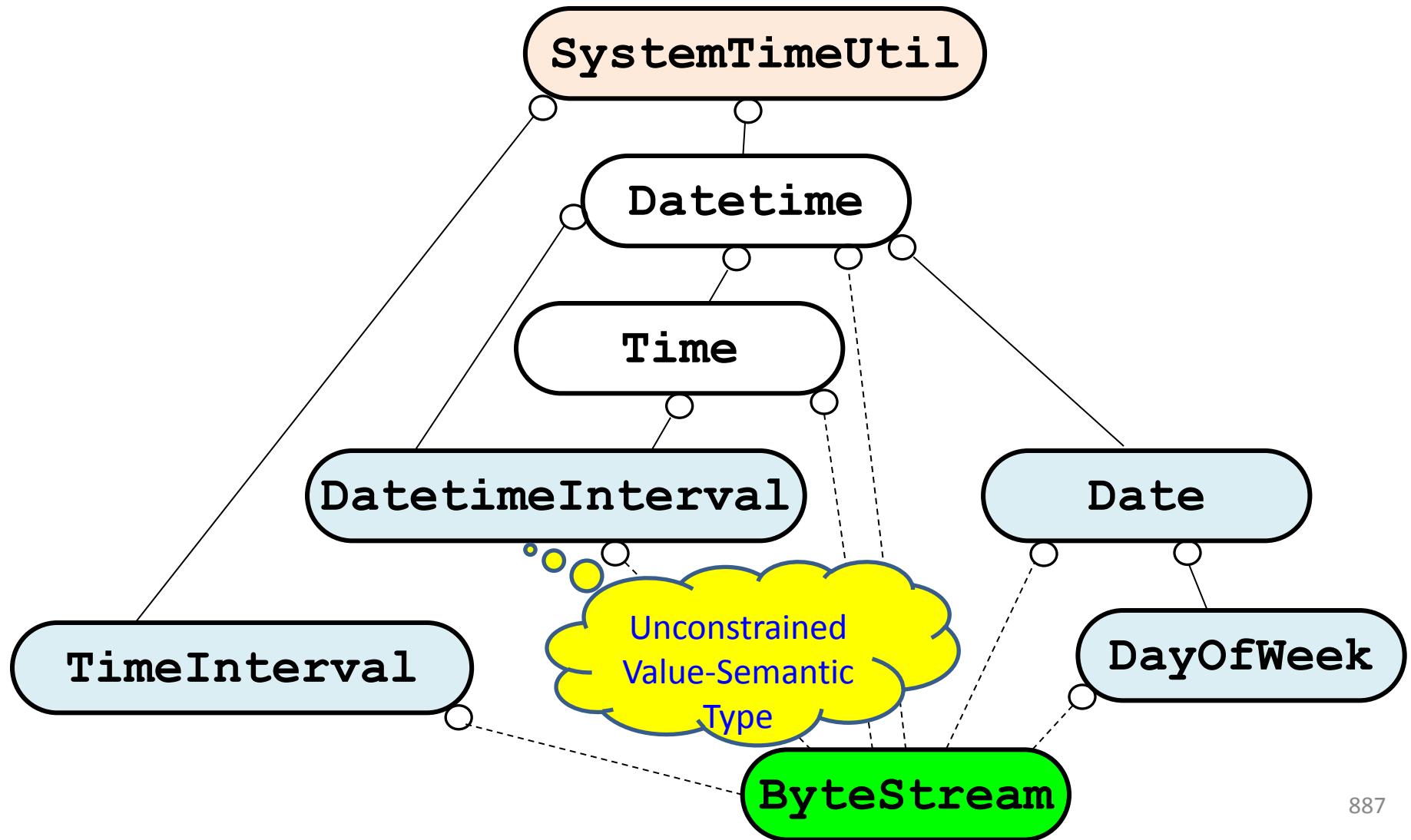
4. Present-Day, Real-World Design Examples

Determine what Date Value today is



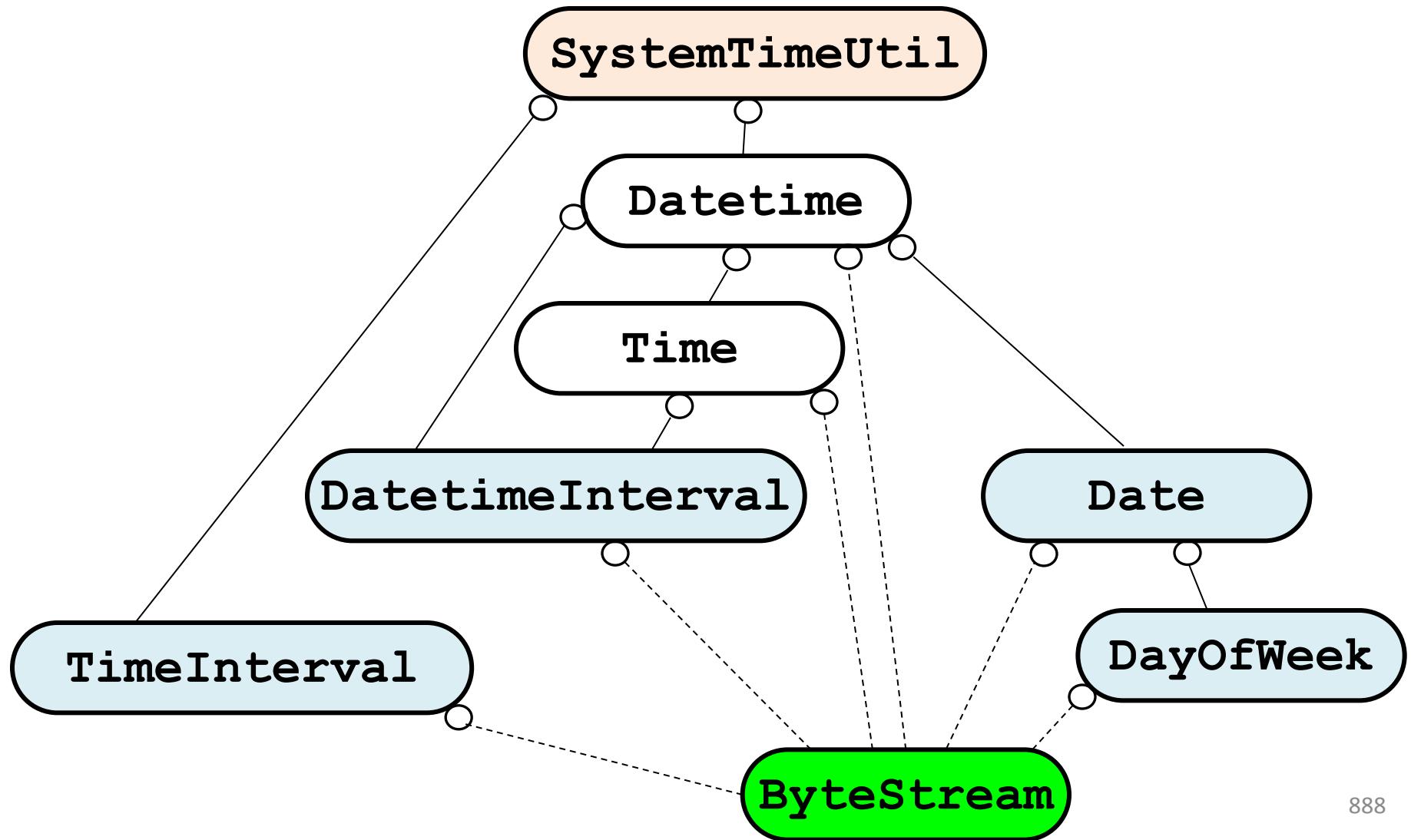
4. Present-Day, Real-World Design Examples

Determine what Date Value today is



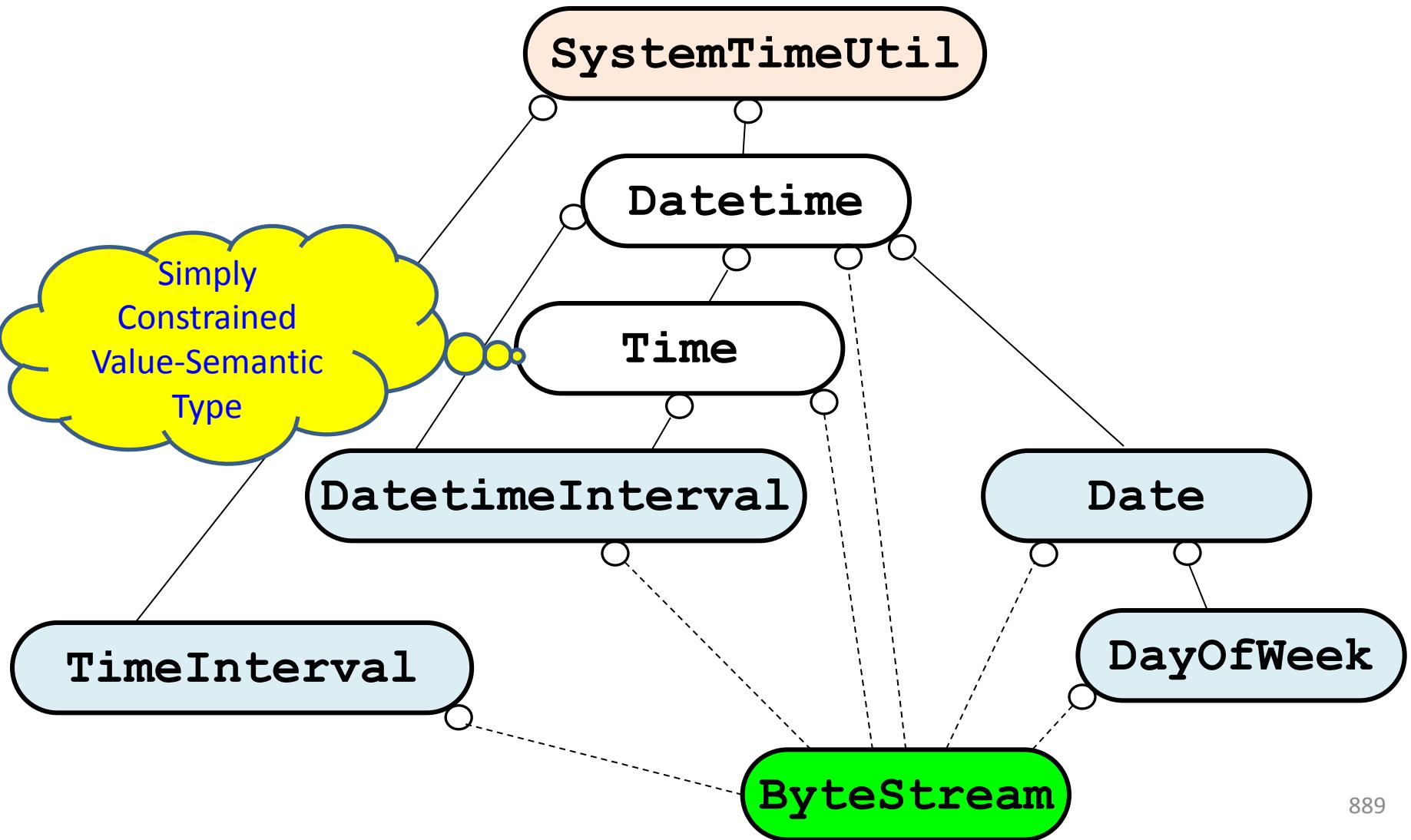
4. Present-Day, Real-World Design Examples

Determine what Date Value today is



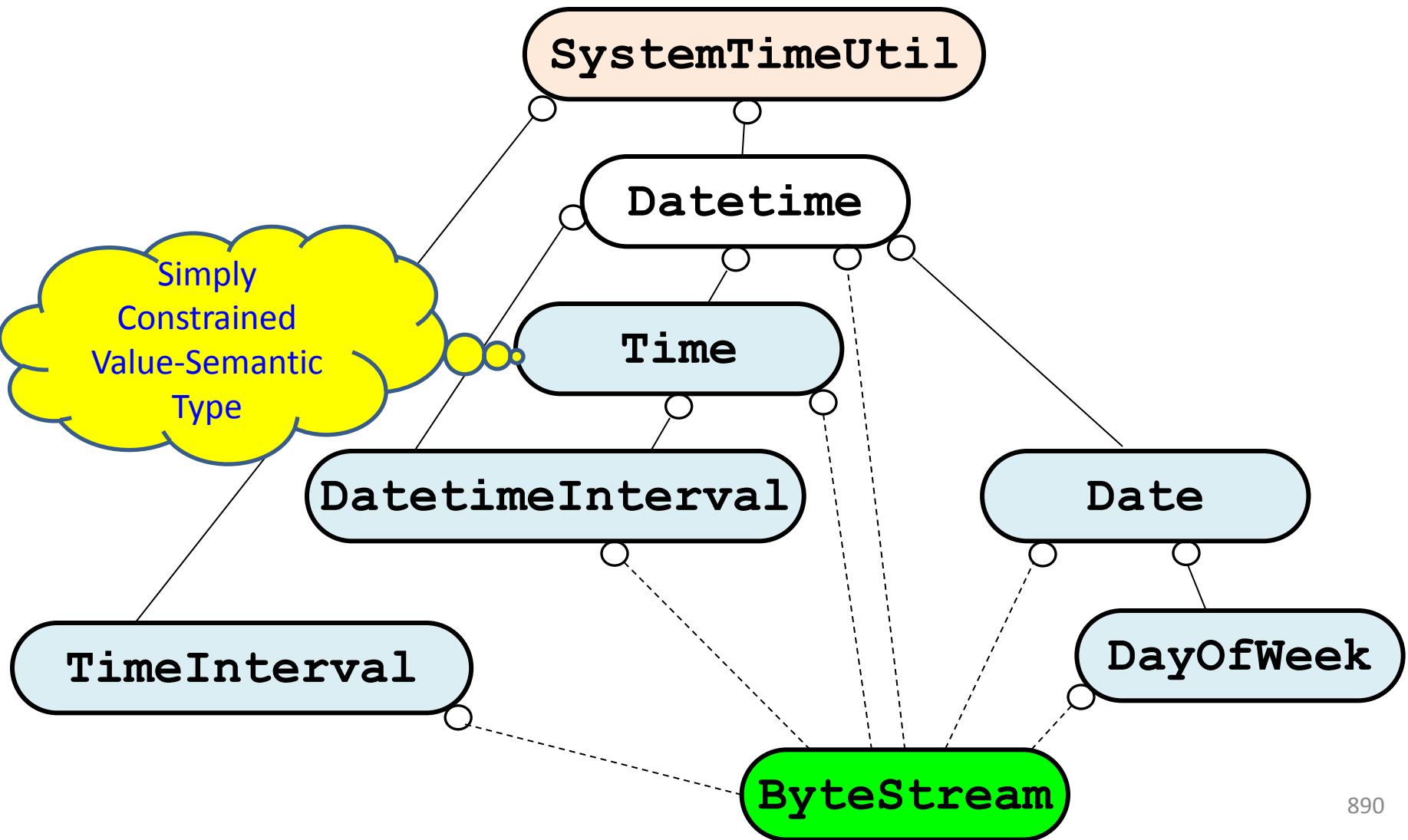
4. Present-Day, Real-World Design Examples

Determine what Date Value today is



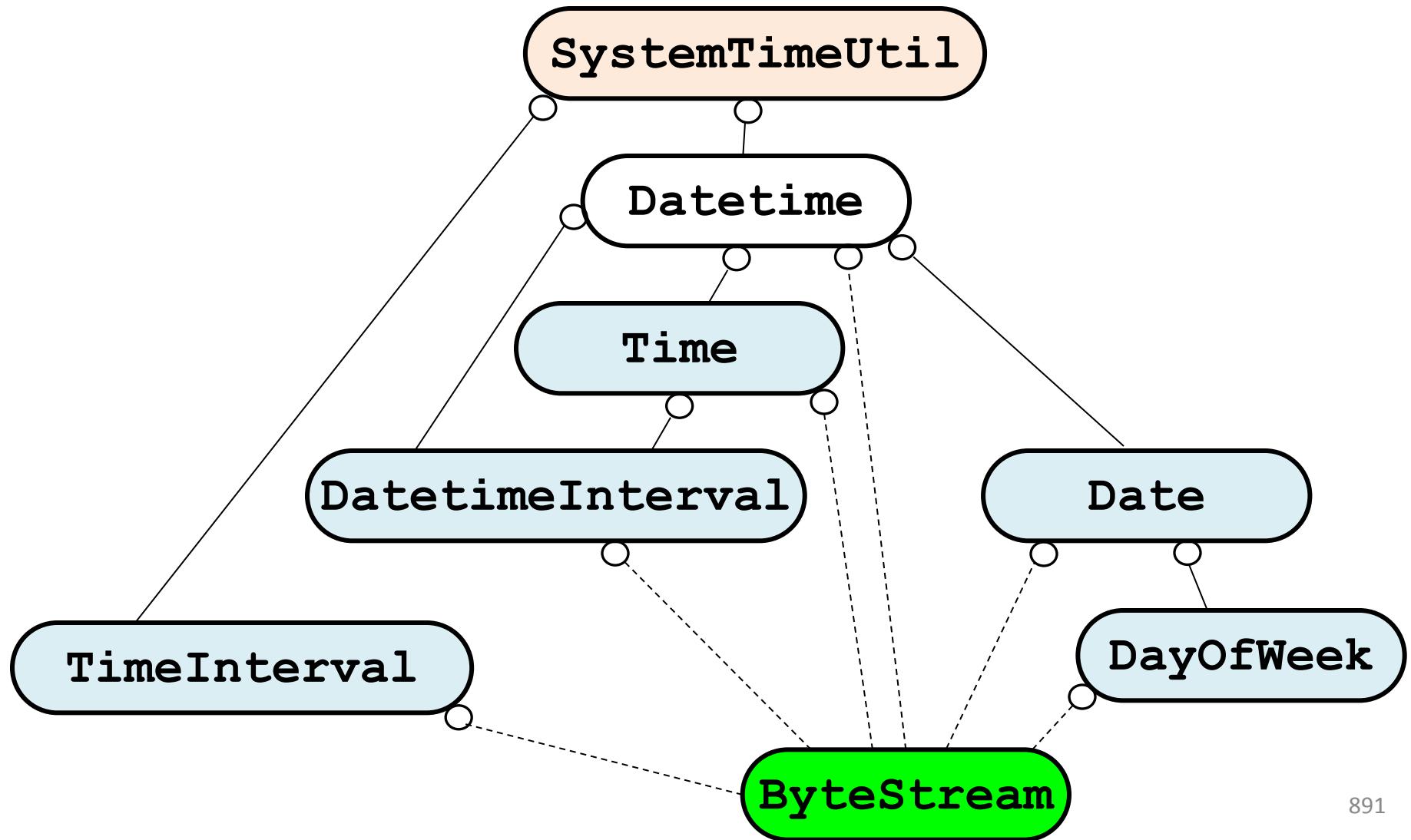
4. Present-Day, Real-World Design Examples

Determine what Date Value today is



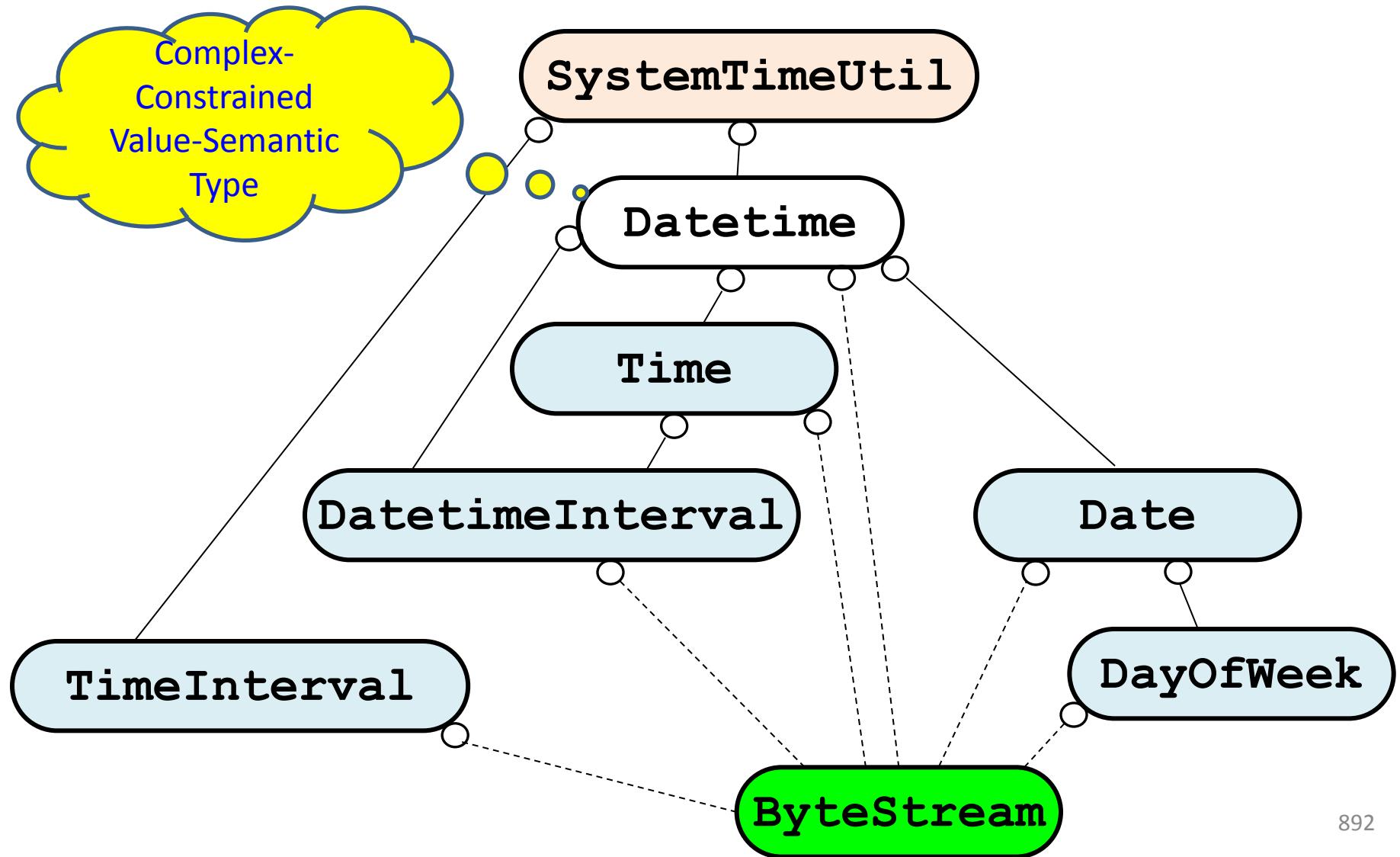
4. Present-Day, Real-World Design Examples

Determine what Date Value today is



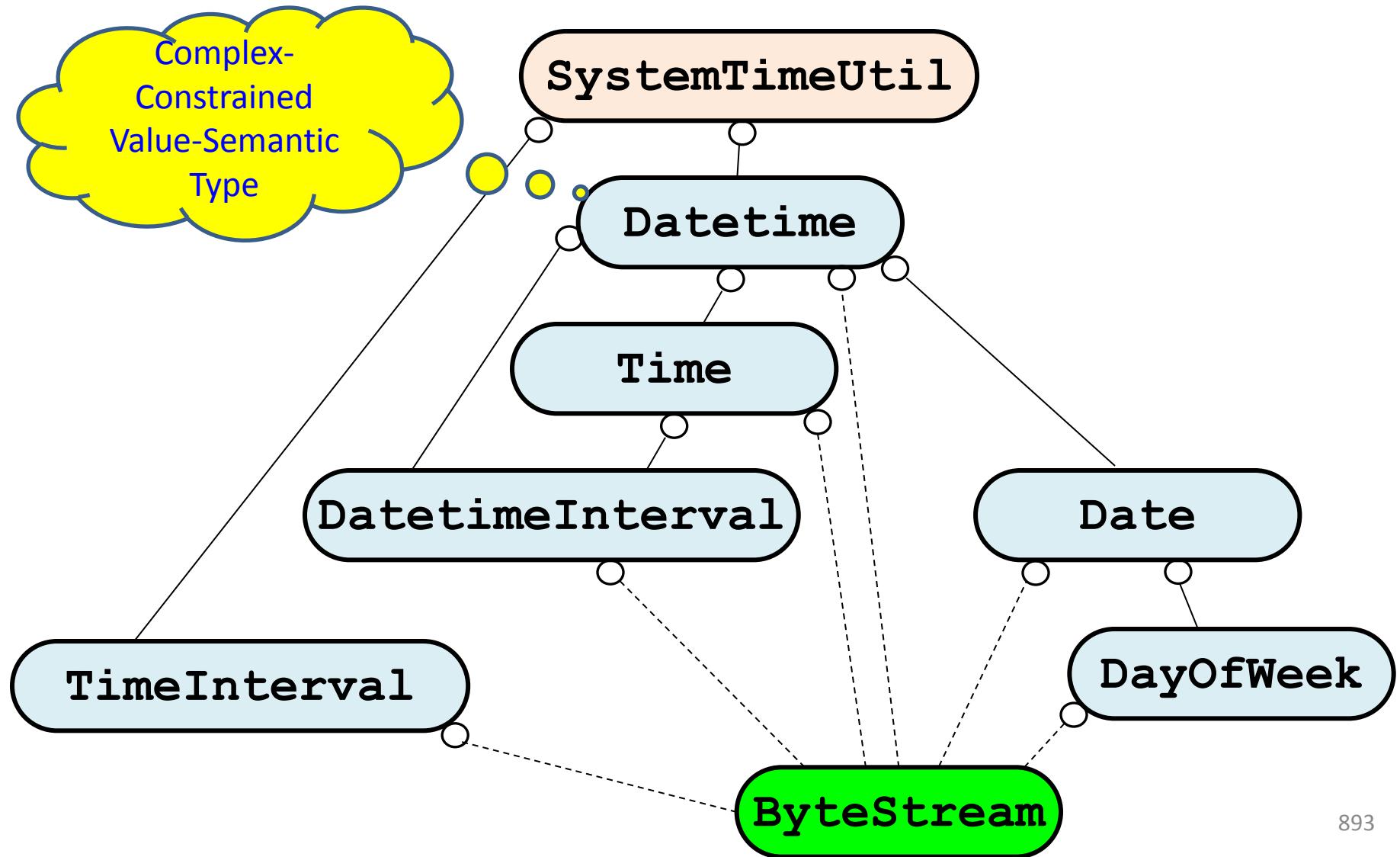
4. Present-Day, Real-World Design Examples

Determine what Date Value today is



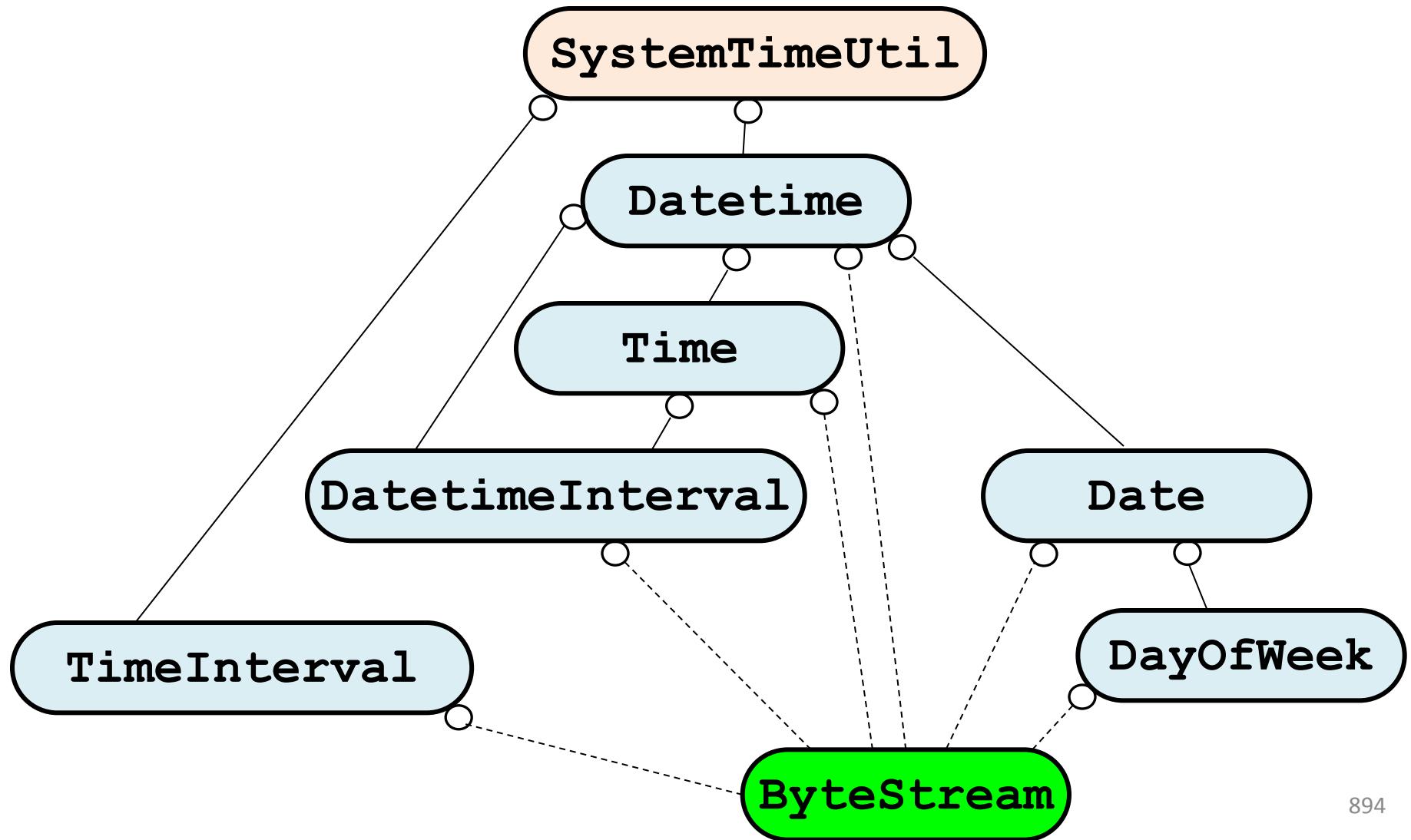
4. Present-Day, Real-World Design Examples

Determine what Date Value today is



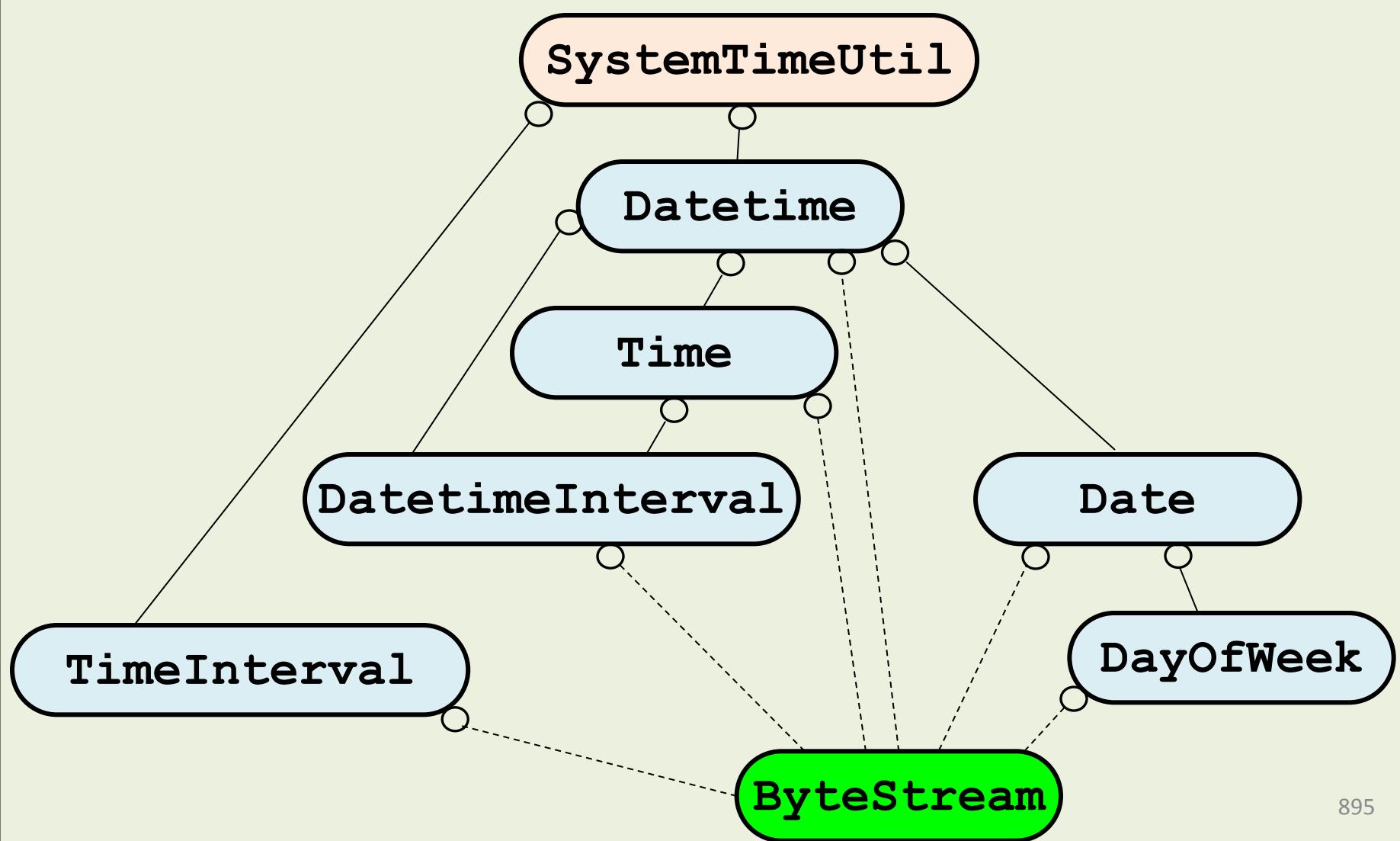
4. Present-Day, Real-World Design Examples

Determine what Date Value today is



4. Present-Day, Real-World Design Examples

Solution 2: What Date is Today?



4. Present-Day, Real-World Design Examples

The Original Request

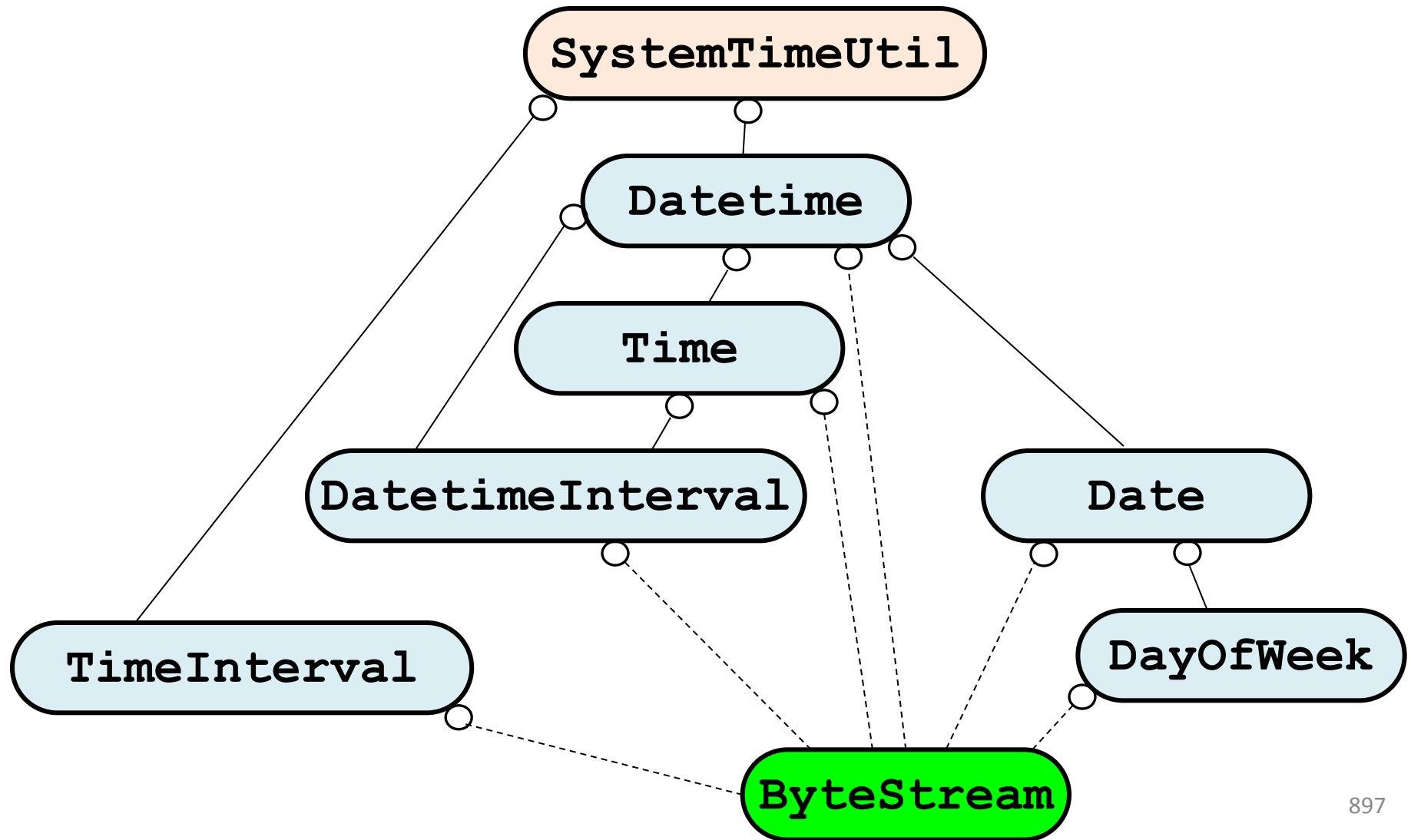
"Write me a 'Date' class that tells me whether today is a business day."

What are the *real* requirements?

1. Represent a *date value* as a C++ Type.
2. Determine what date value *today* is.
3. **Determine if a date value is a *business day*.**
4. Provide well-factored useful components that we'll need over and over again!

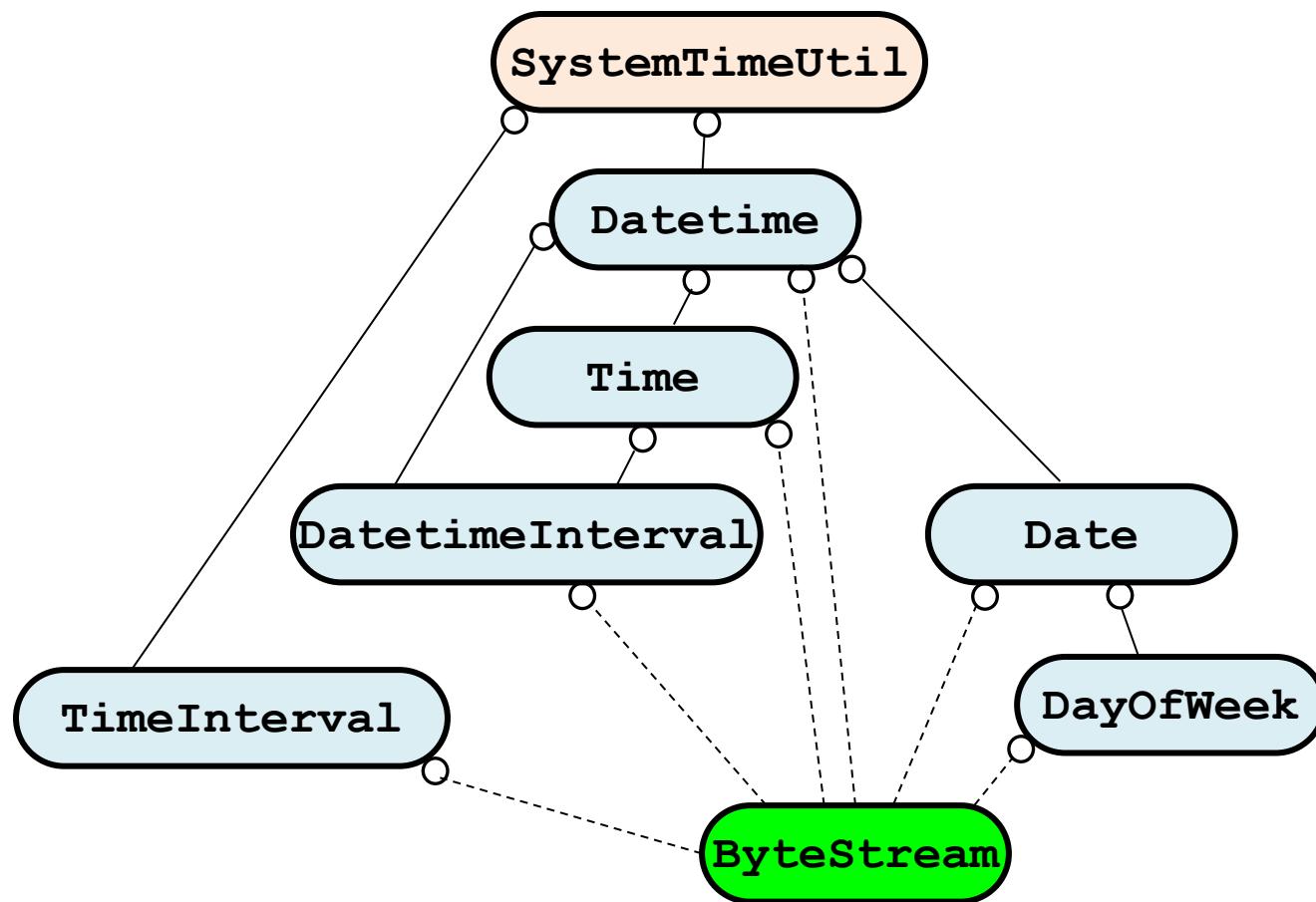
4. Present-Day, Real-World Design Examples

Determine if a Date Value is a *Business Day*



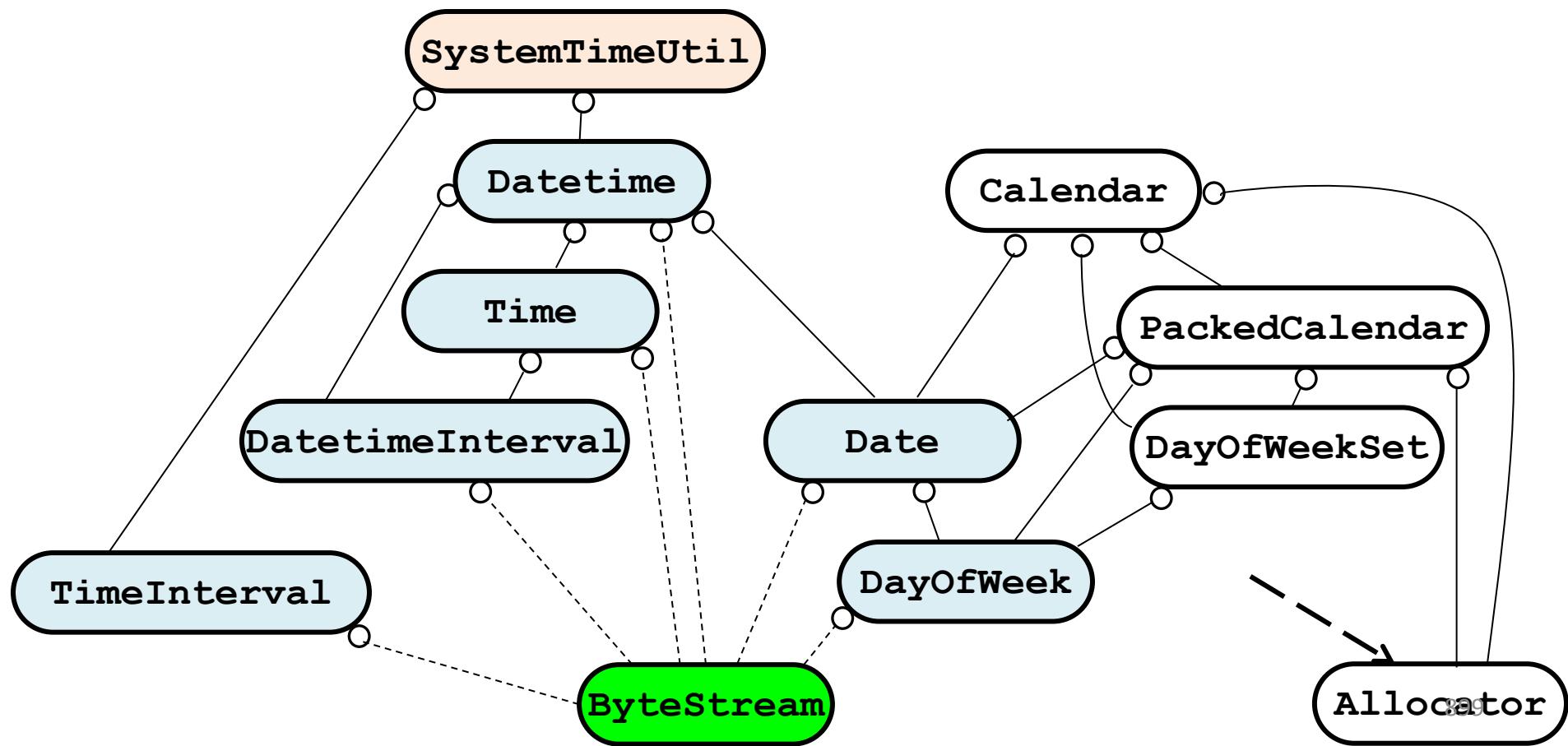
4. Present-Day, Real-World Design Examples

Determine if a Date Value is a *Business Day*



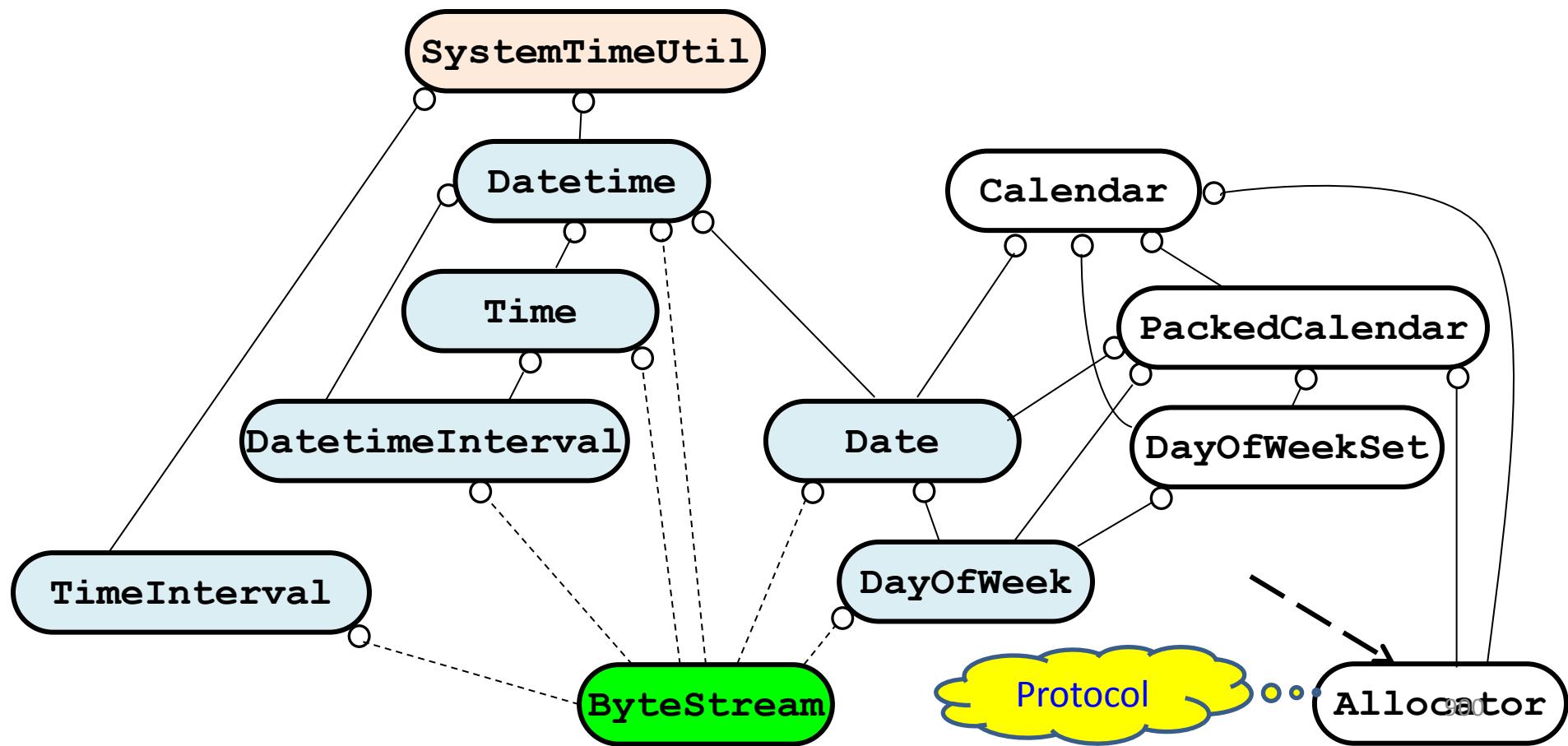
4. Present-Day, Real-World Design Examples

Determine if a Date Value is a *Business Day*



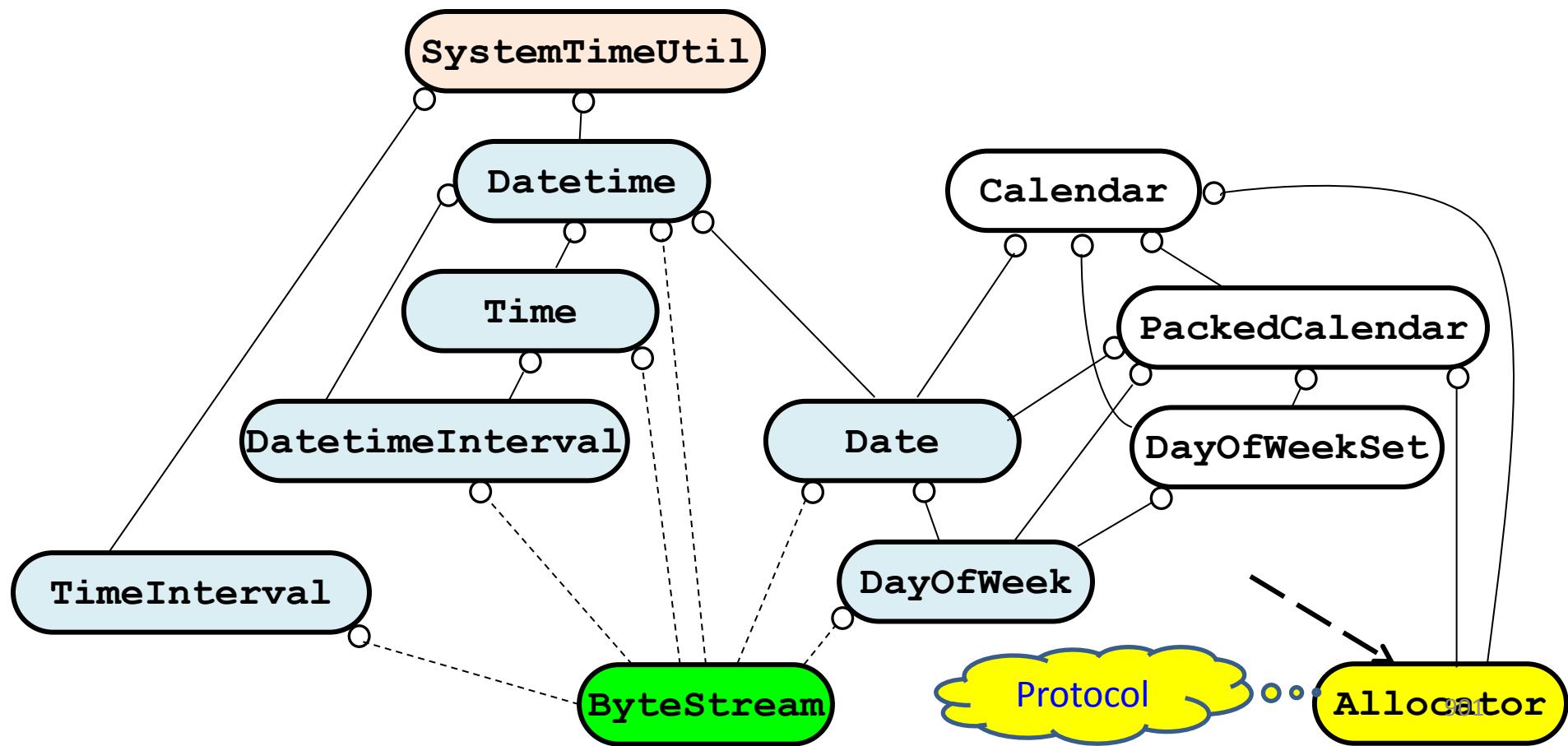
4. Present-Day, Real-World Design Examples

Determine if a Date Value is a *Business Day*



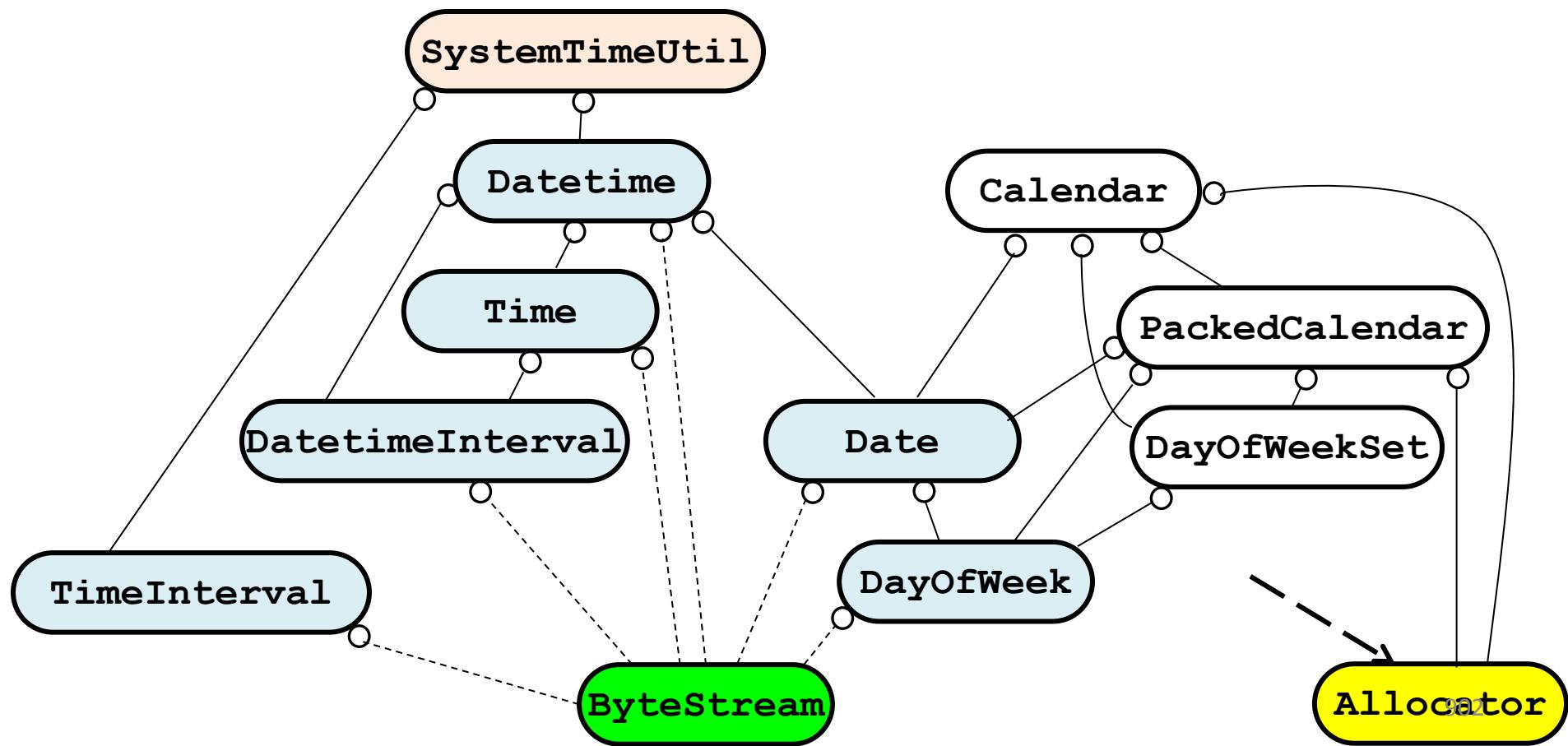
4. Present-Day, Real-World Design Examples

Determine if a Date Value is a *Business Day*



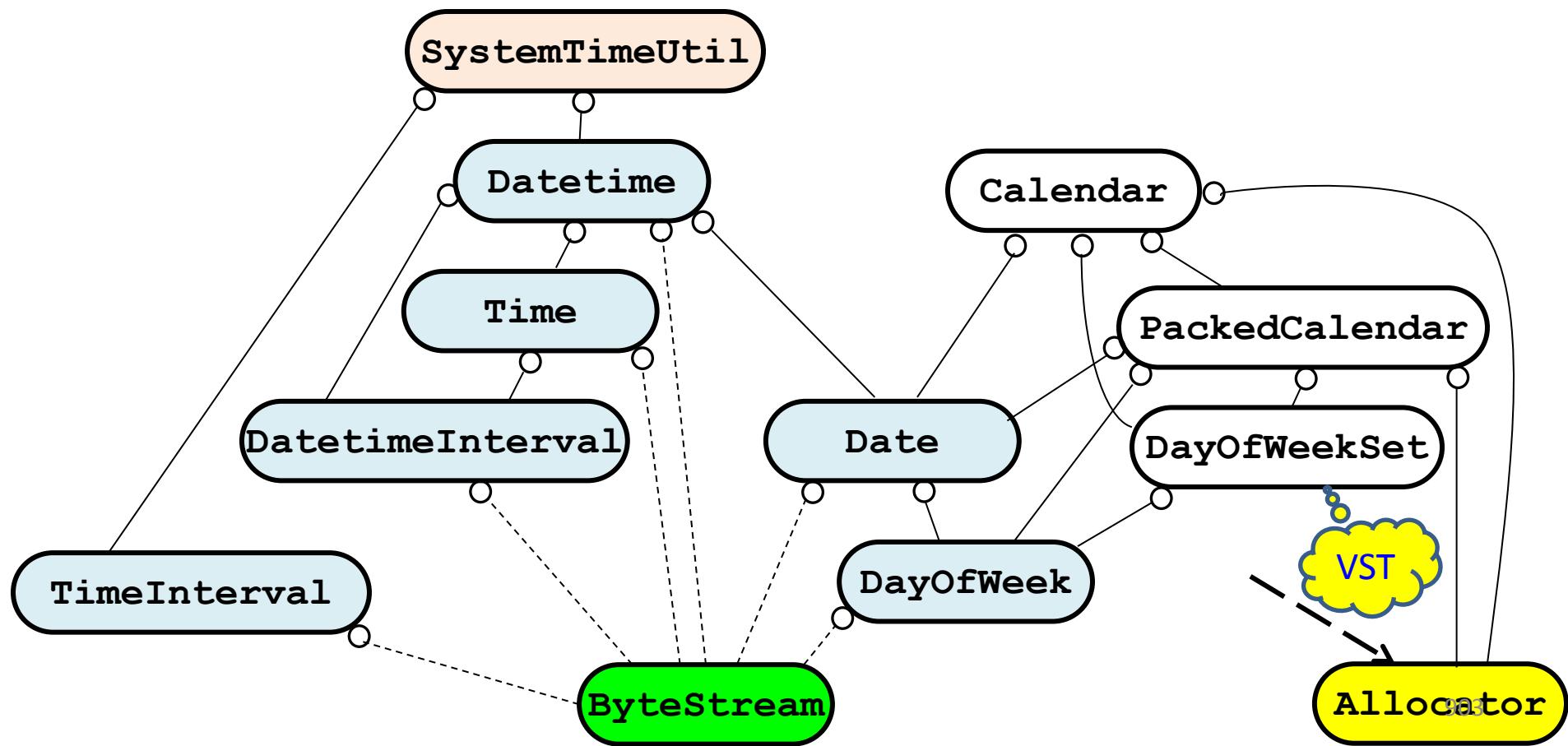
4. Present-Day, Real-World Design Examples

Determine if a Date Value is a *Business Day*



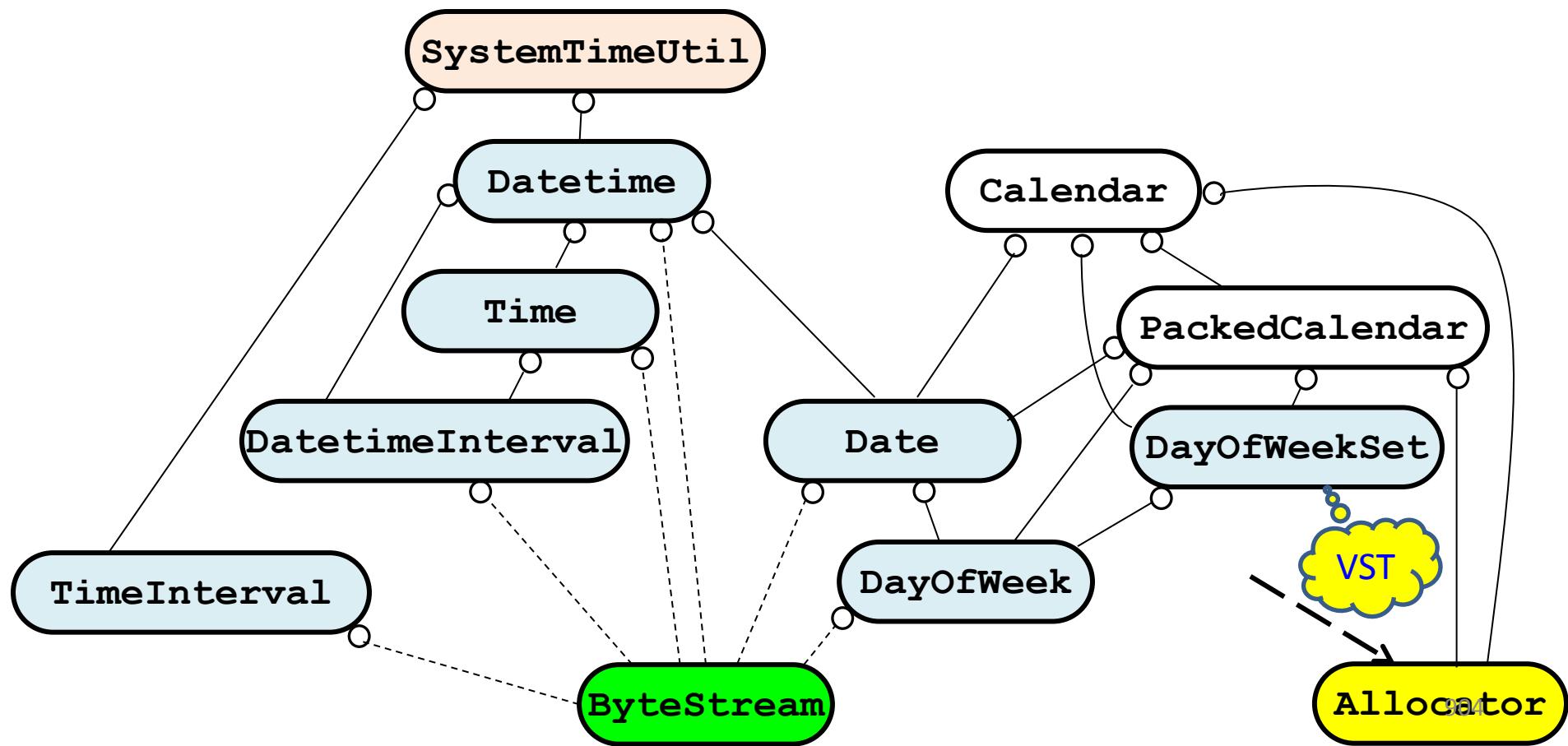
4. Present-Day, Real-World Design Examples

Determine if a Date Value is a *Business Day*



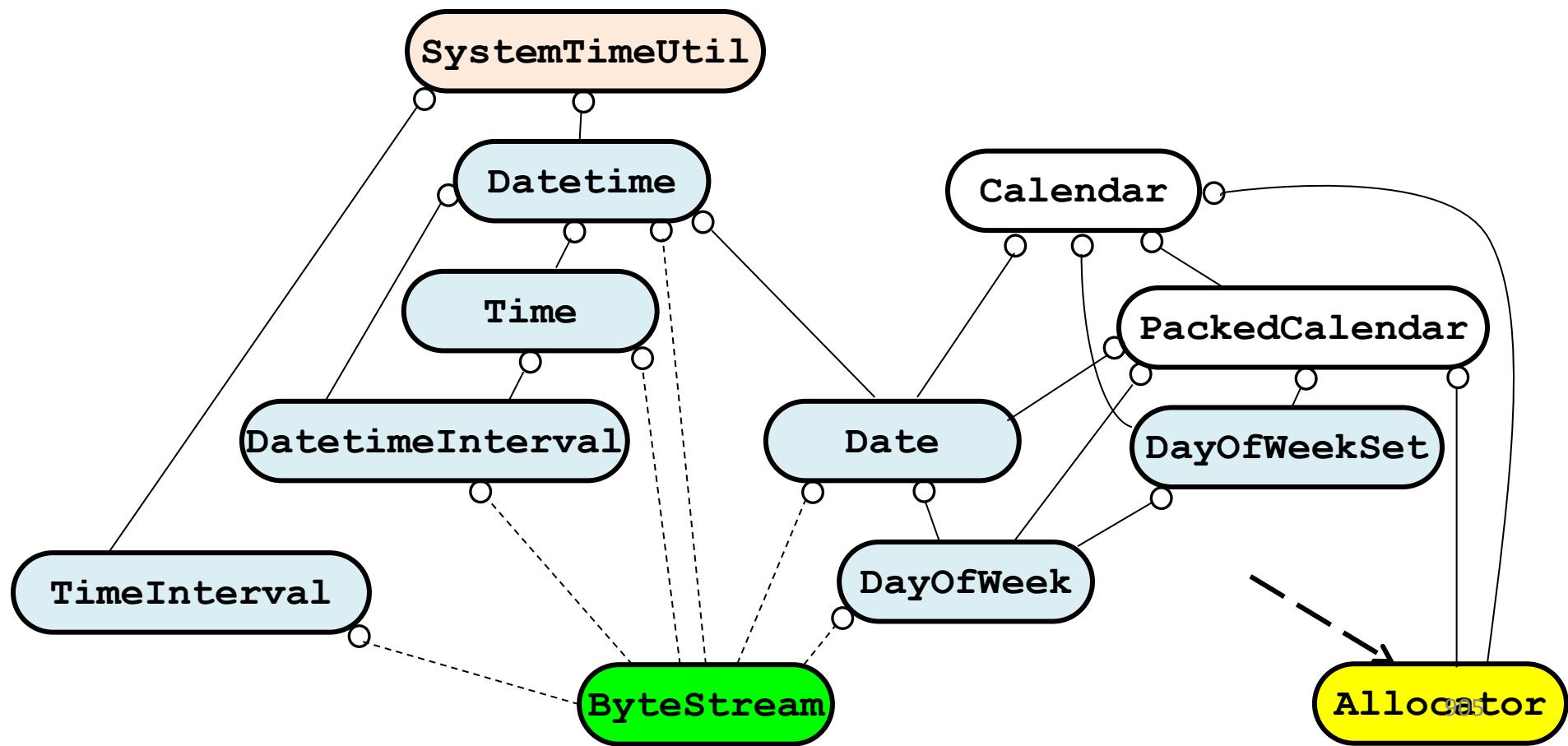
4. Present-Day, Real-World Design Examples

Determine if a Date Value is a *Business Day*



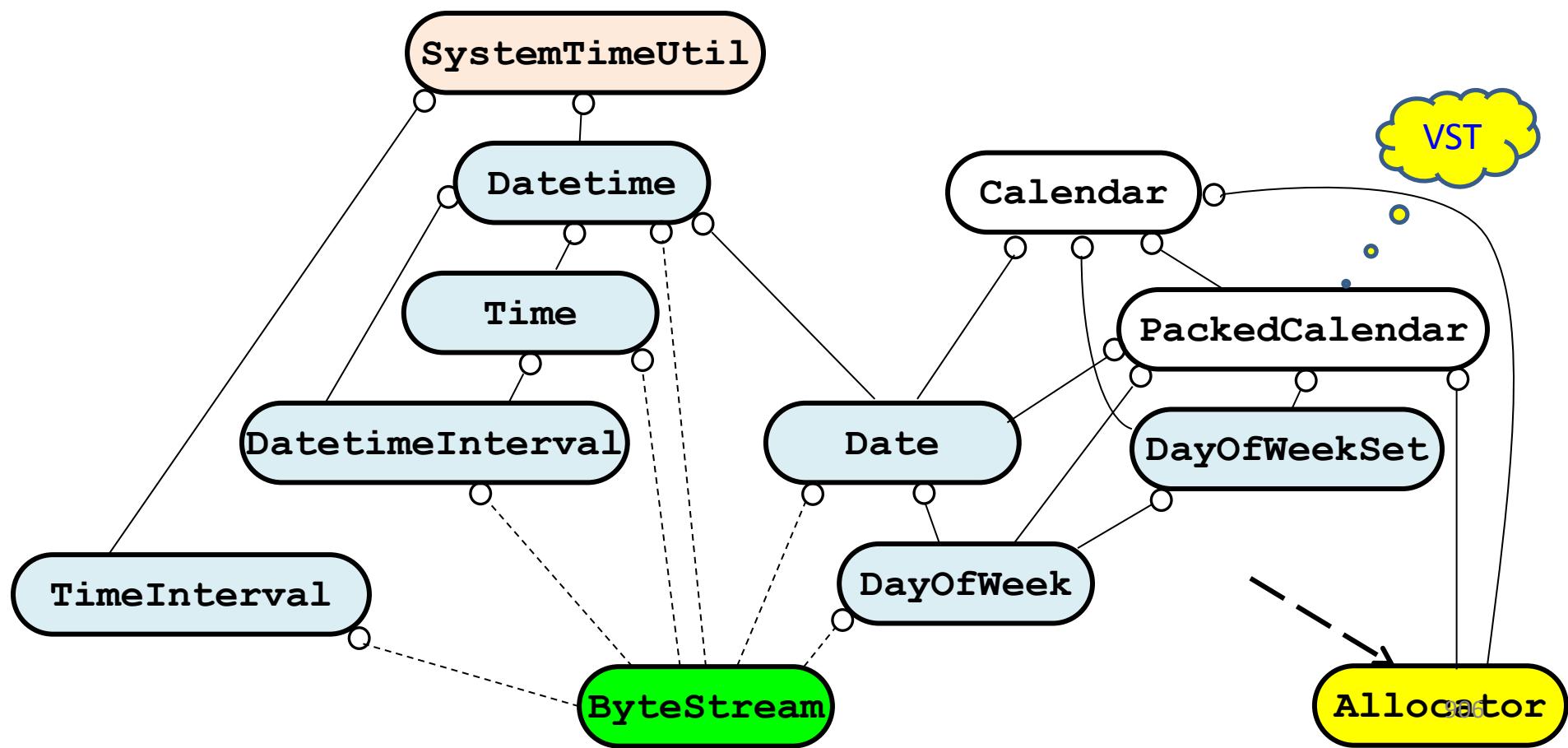
4. Present-Day, Real-World Design Examples

Determine if a Date Value is a *Business Day*



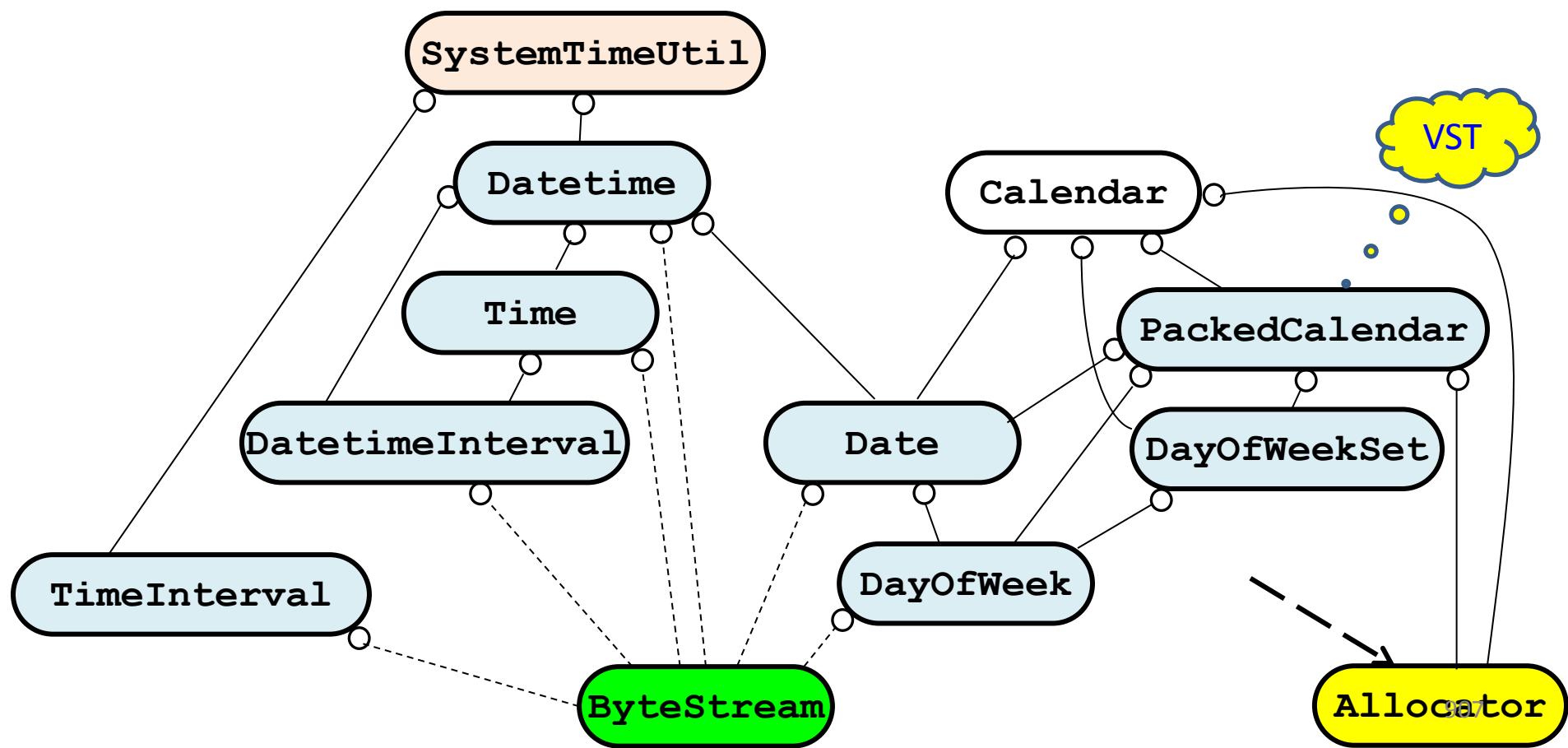
4. Present-Day, Real-World Design Examples

Determine if a Date Value is a *Business Day*



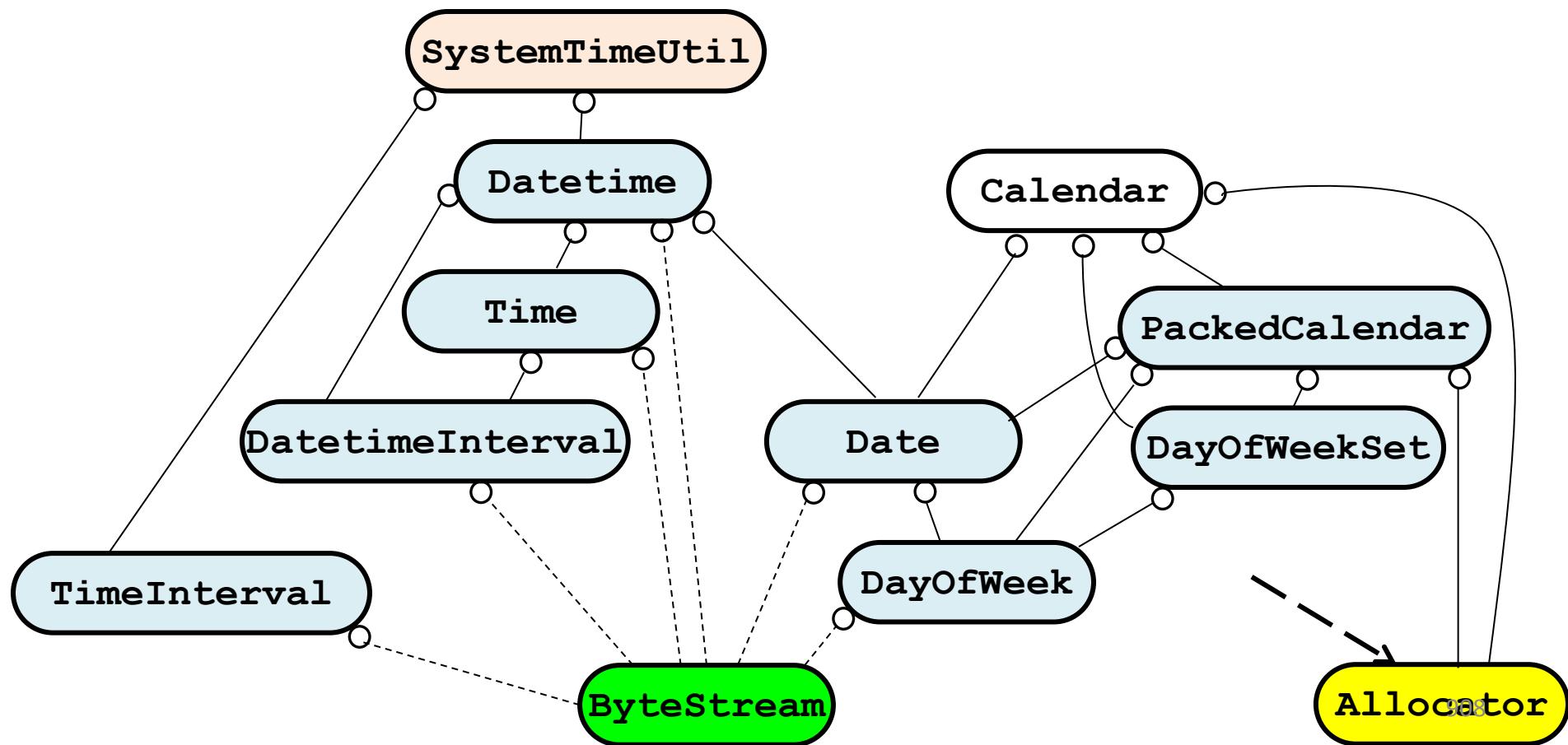
4. Present-Day, Real-World Design Examples

Determine if a Date Value is a *Business Day*



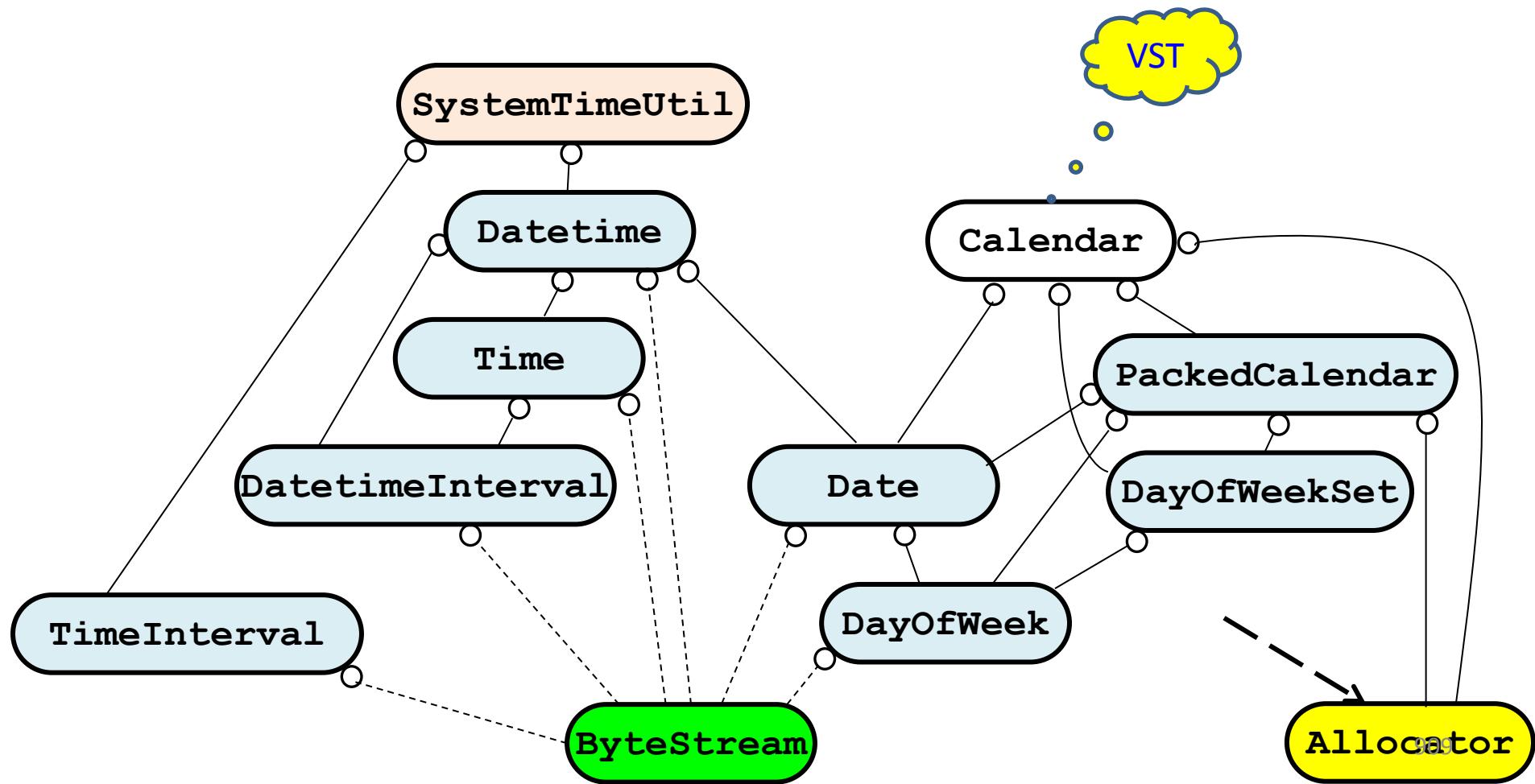
4. Present-Day, Real-World Design Examples

Determine if a Date Value is a *Business Day*



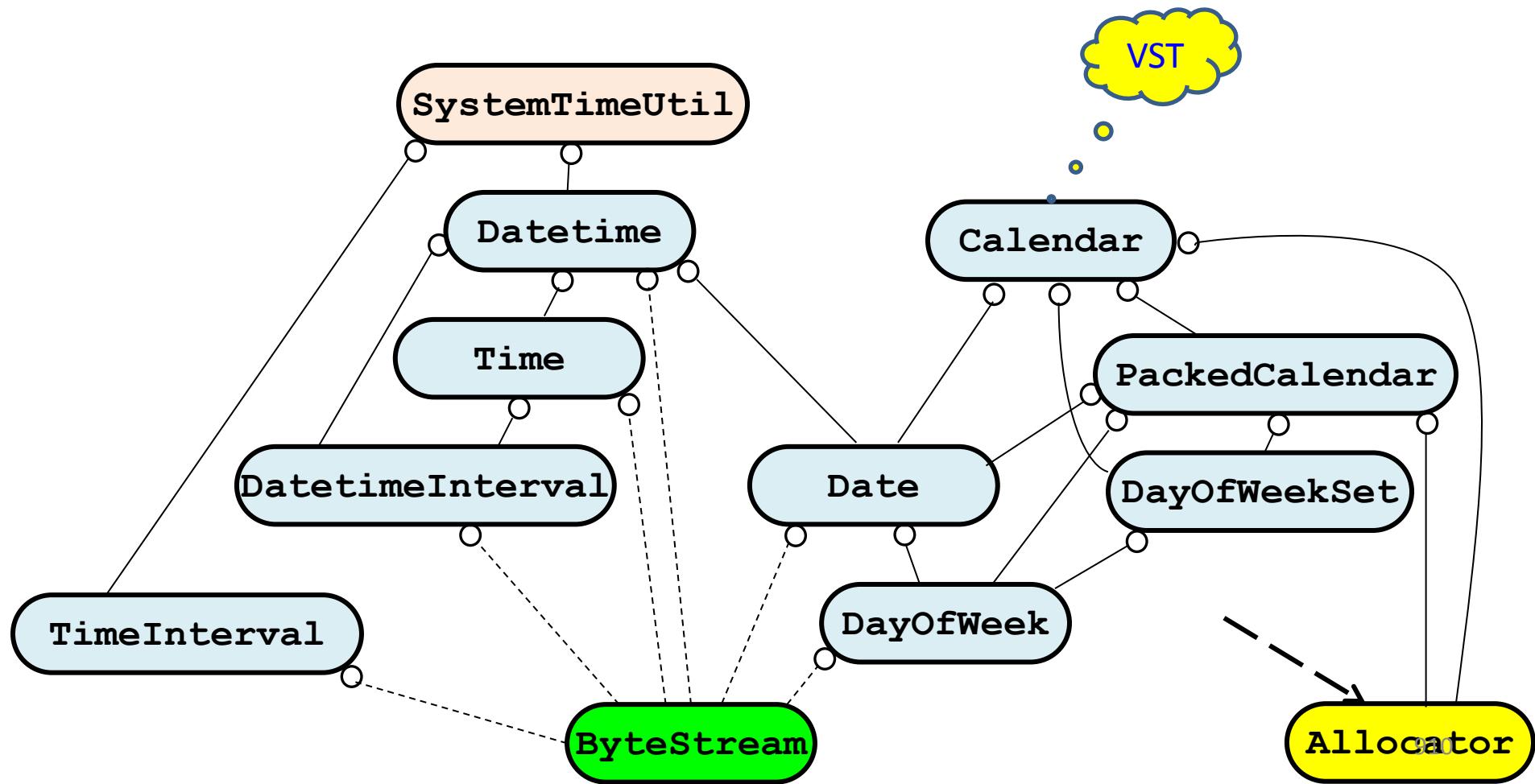
4. Present-Day, Real-World Design Examples

Determine if a Date Value is a *Business Day*



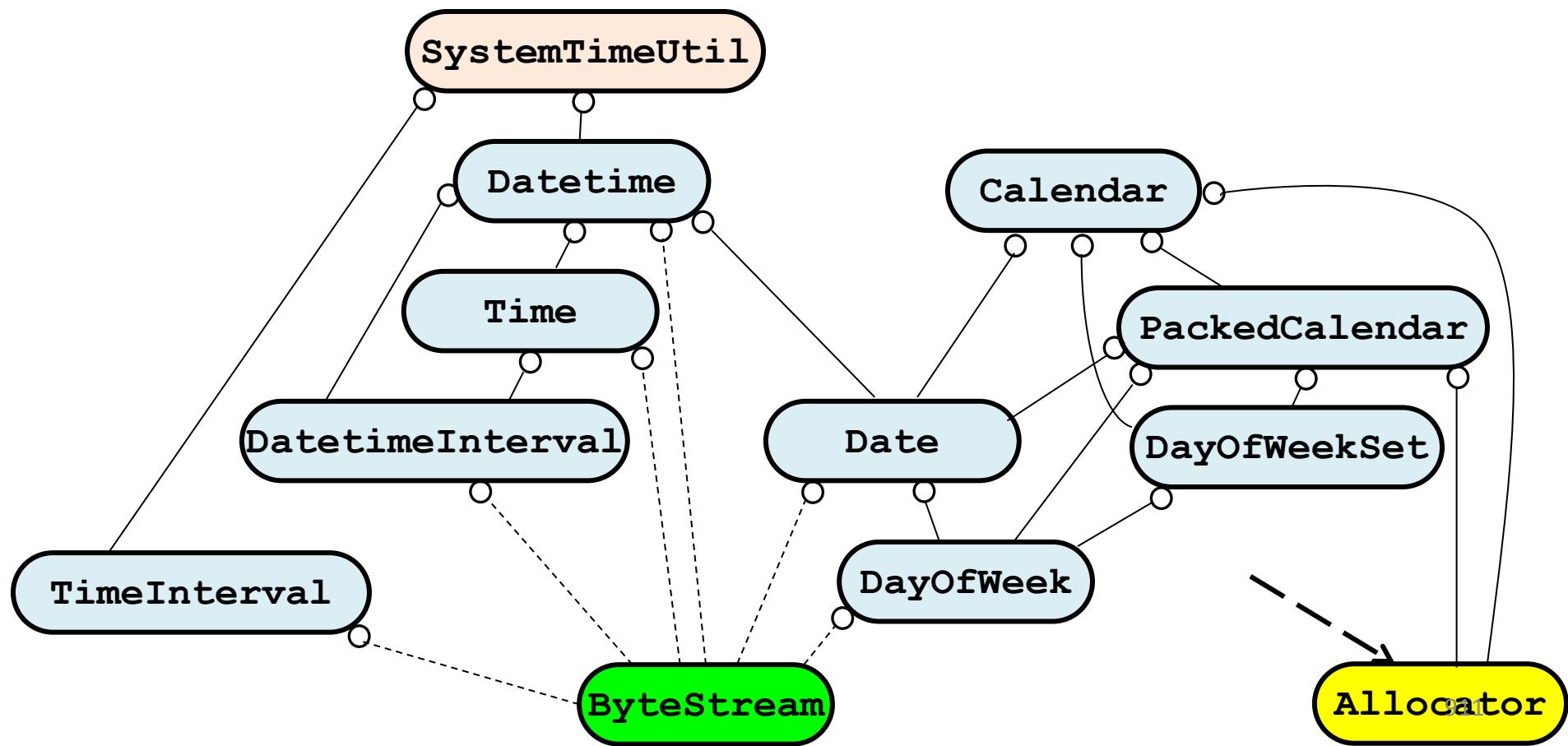
4. Present-Day, Real-World Design Examples

Determine if a Date Value is a *Business Day*



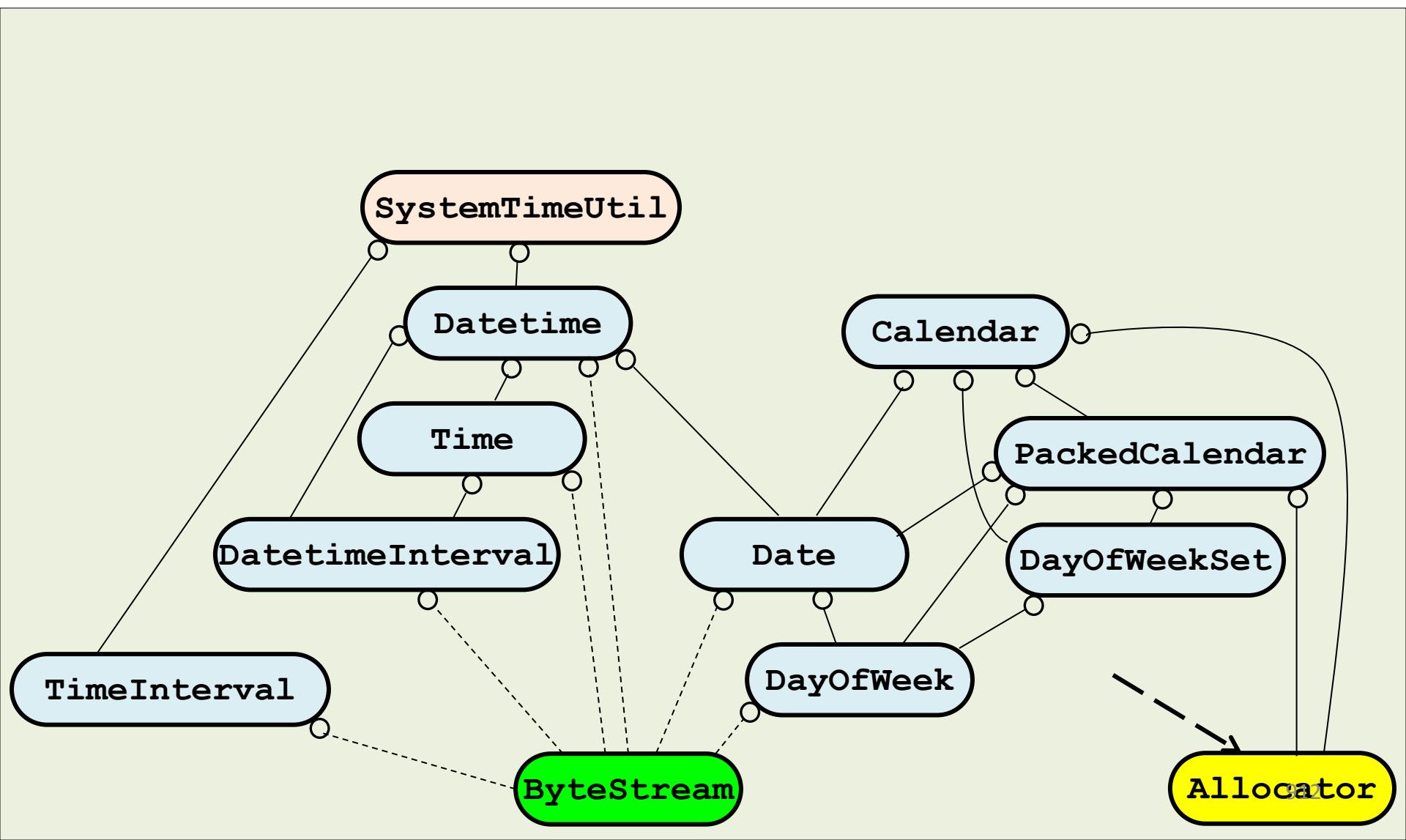
4. Present-Day, Real-World Design Examples

Determine if a Date Value is a *Business Day*



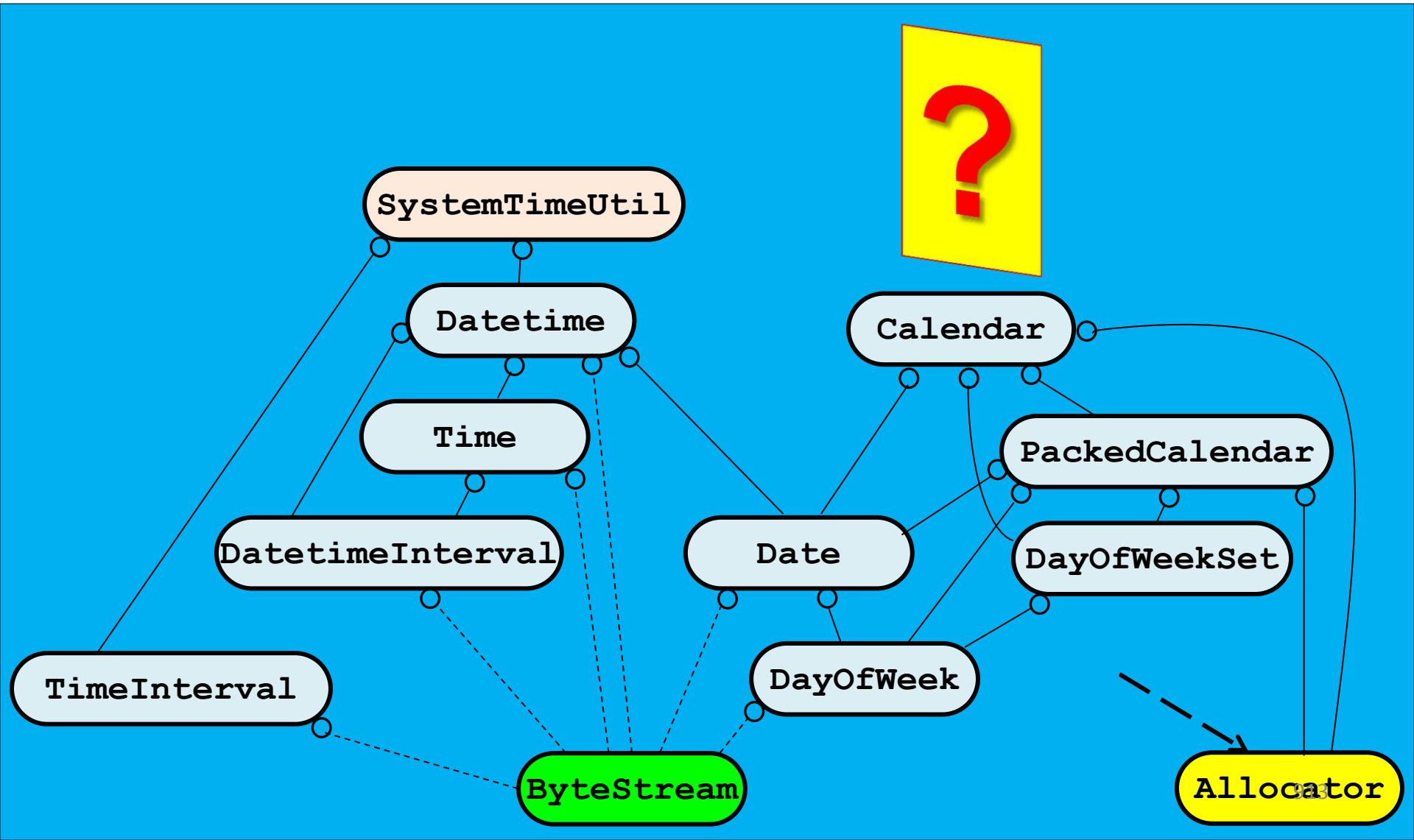
4. Present-Day, Real-World Design Examples

Determine if a Date Value is a *Business Day*



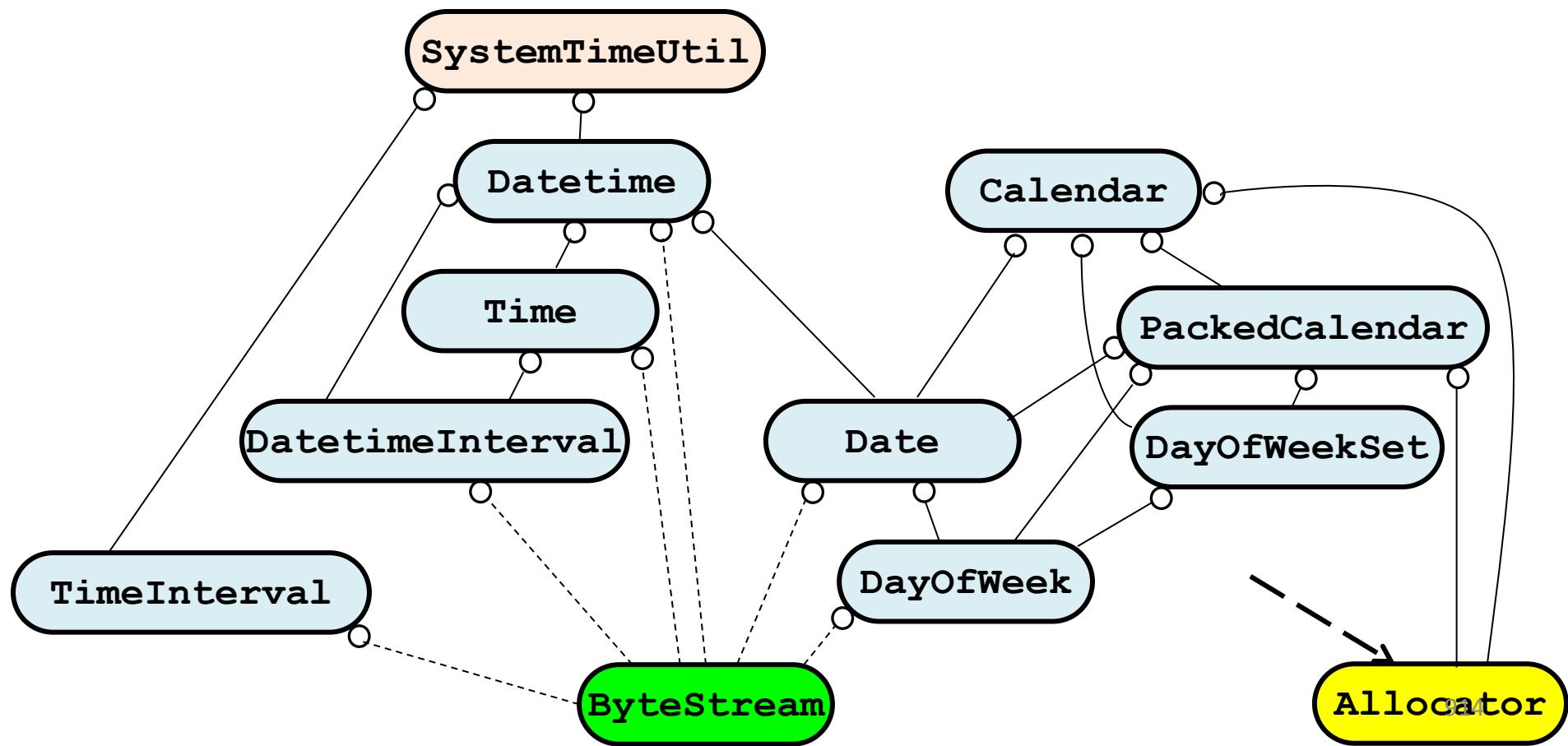
4. Present-Day, Real-World Design Examples

Wait a Minute: Where is the Data Source?



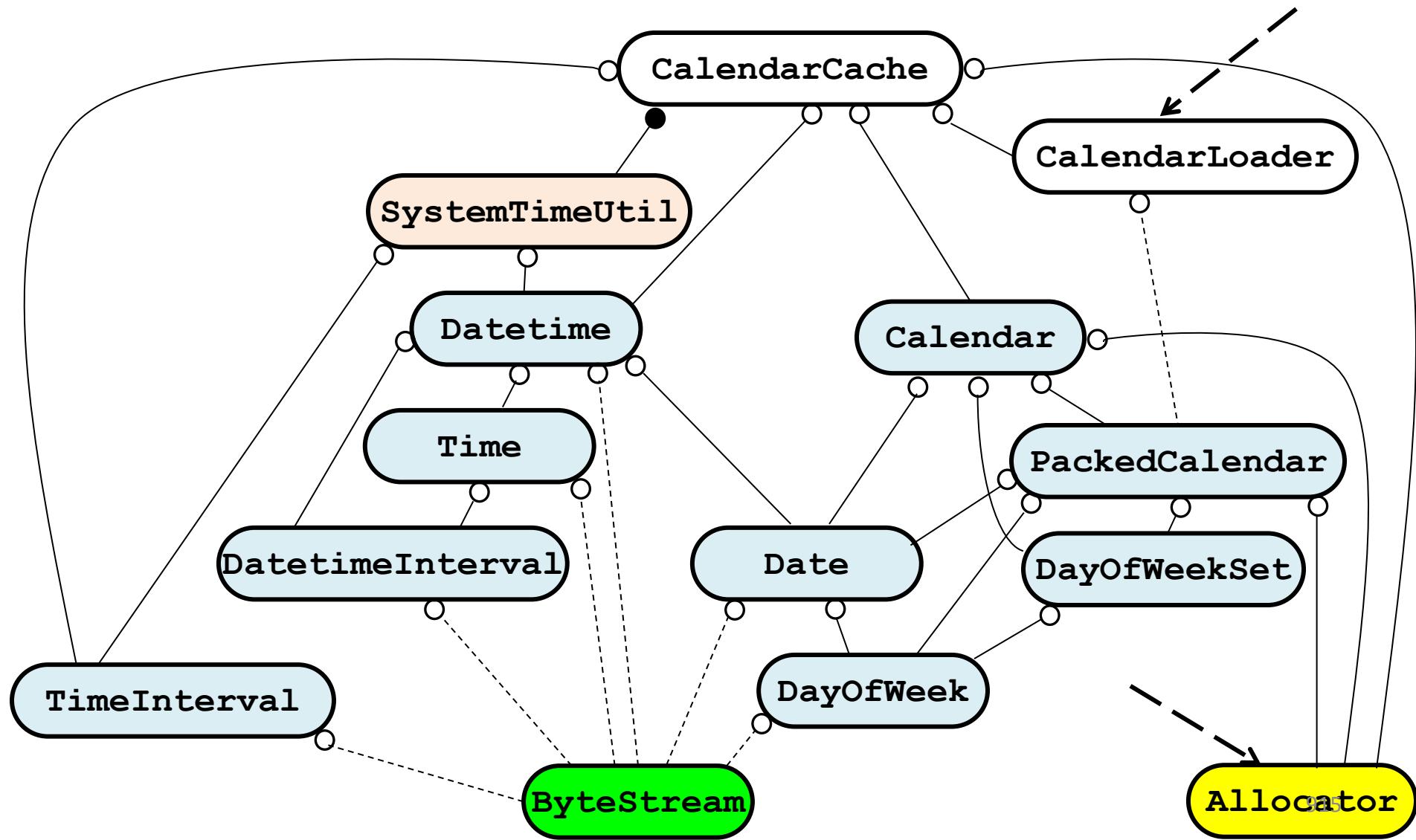
4. Present-Day, Real-World Design Examples

Wait a Minute: Where is the Data Source?



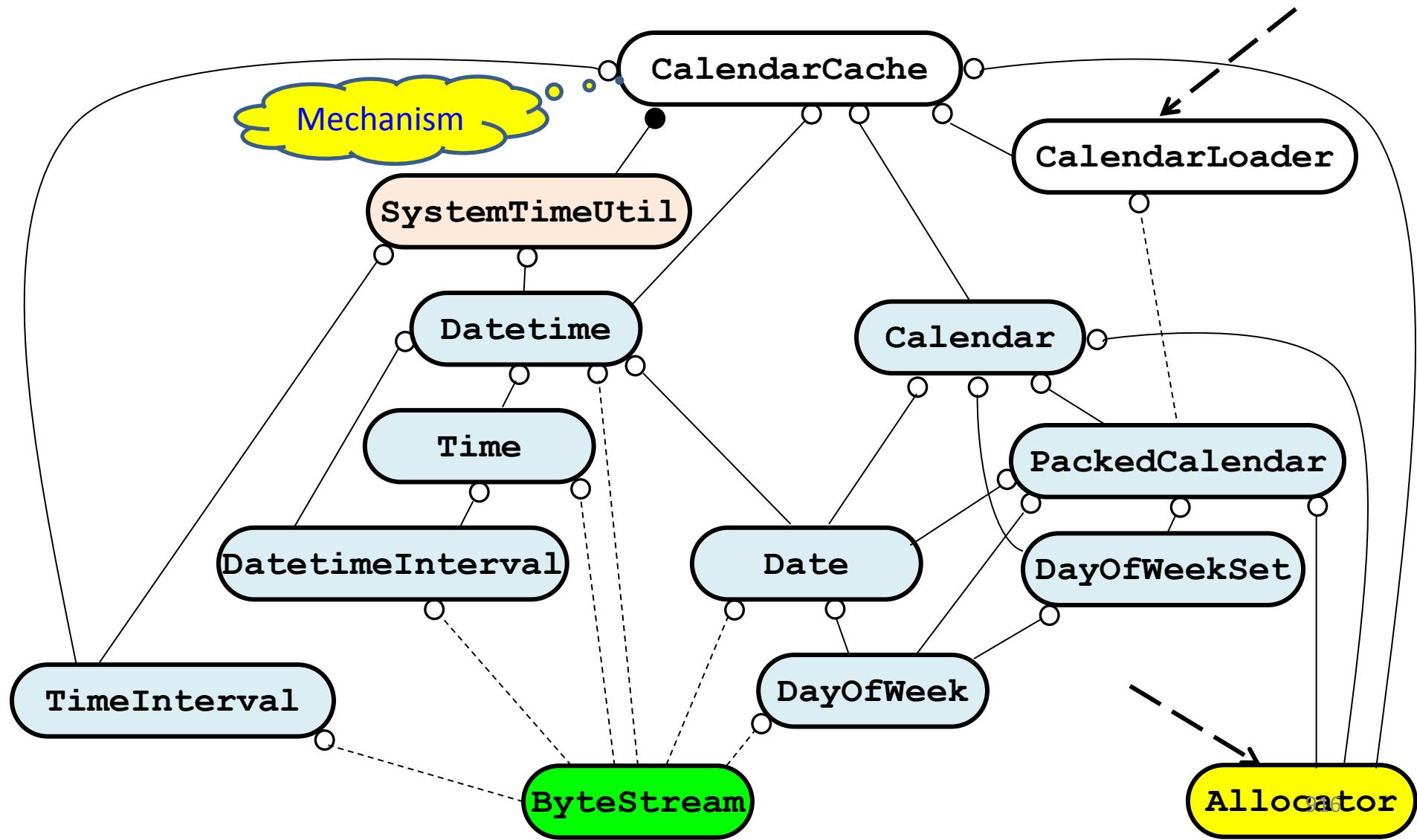
4. Present-Day, Real-World Design Examples

Wait a Minute: Where is the Data Source?



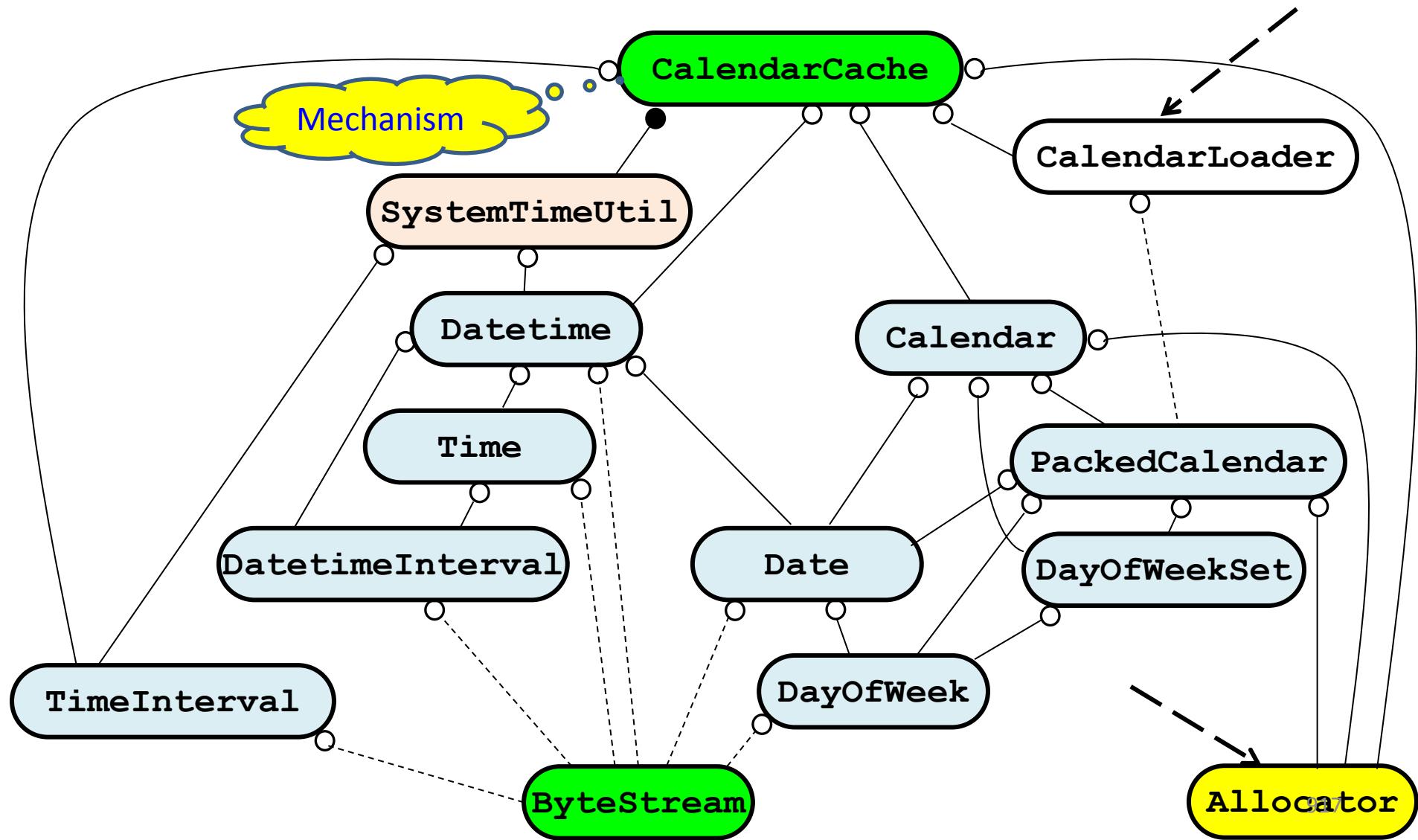
4. Present-Day, Real-World Design Examples

Wait a Minute: Where is the Data Source?



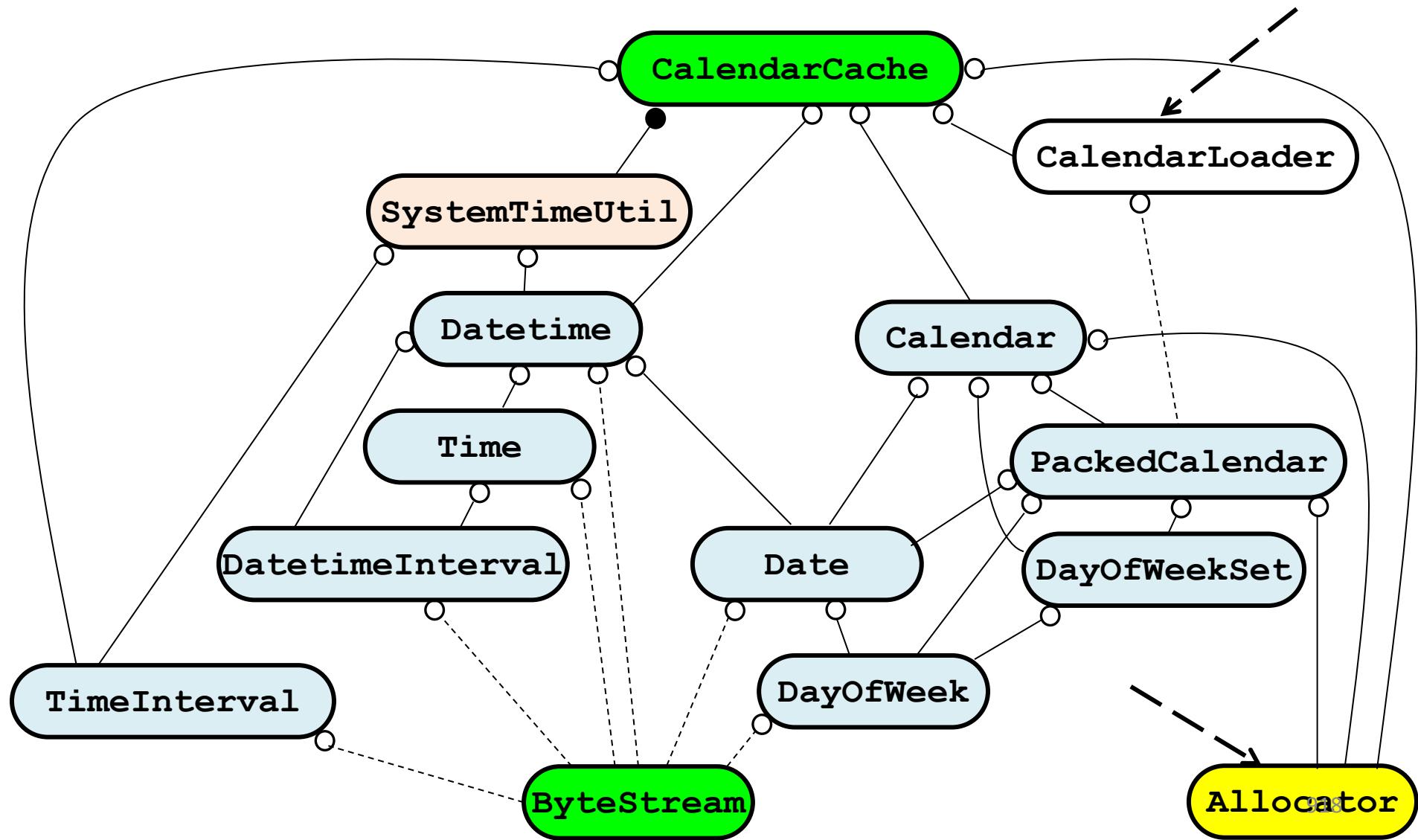
4. Present-Day, Real-World Design Examples

Wait a Minute: Where is the Data Source?



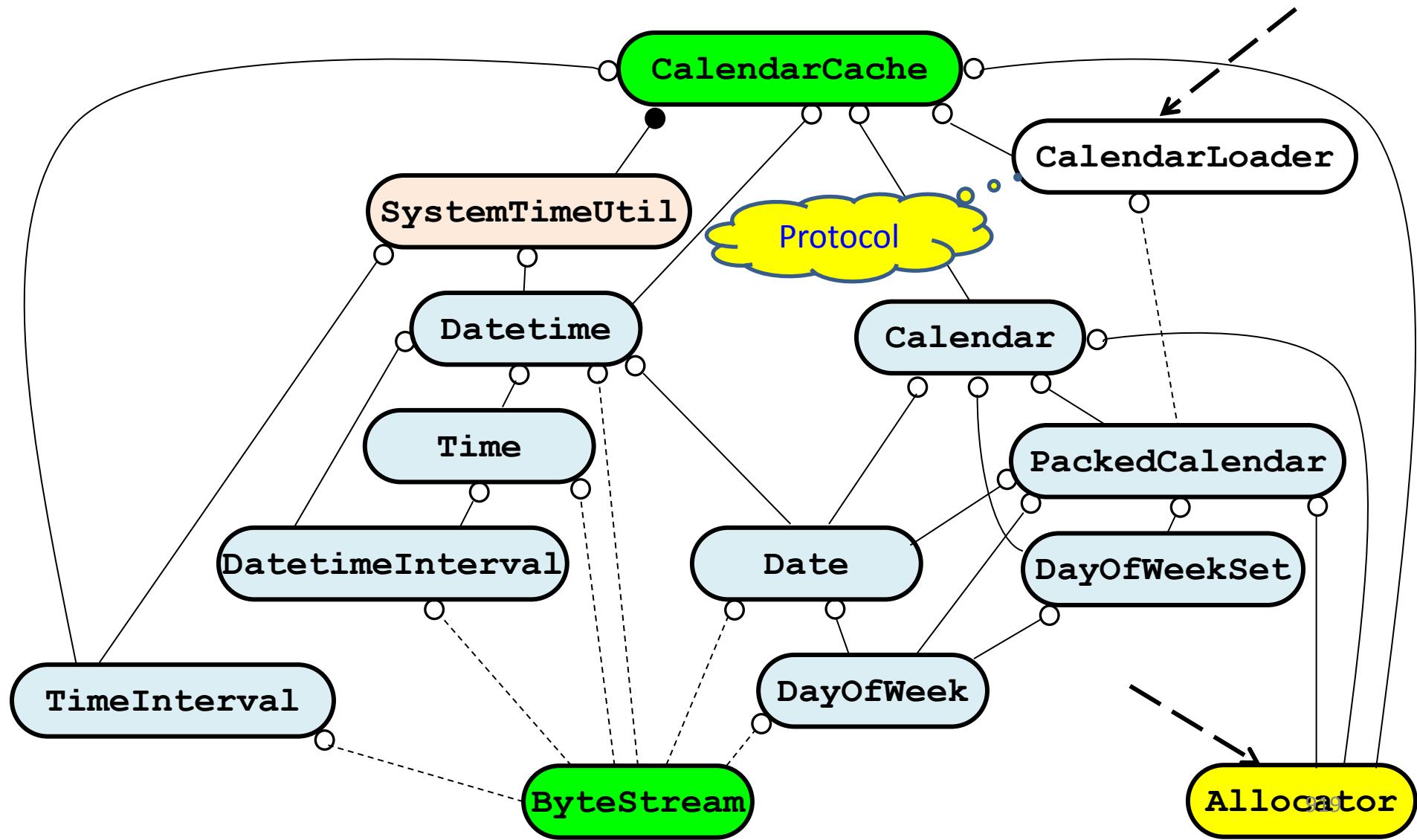
4. Present-Day, Real-World Design Examples

Wait a Minute: Where is the Data Source?



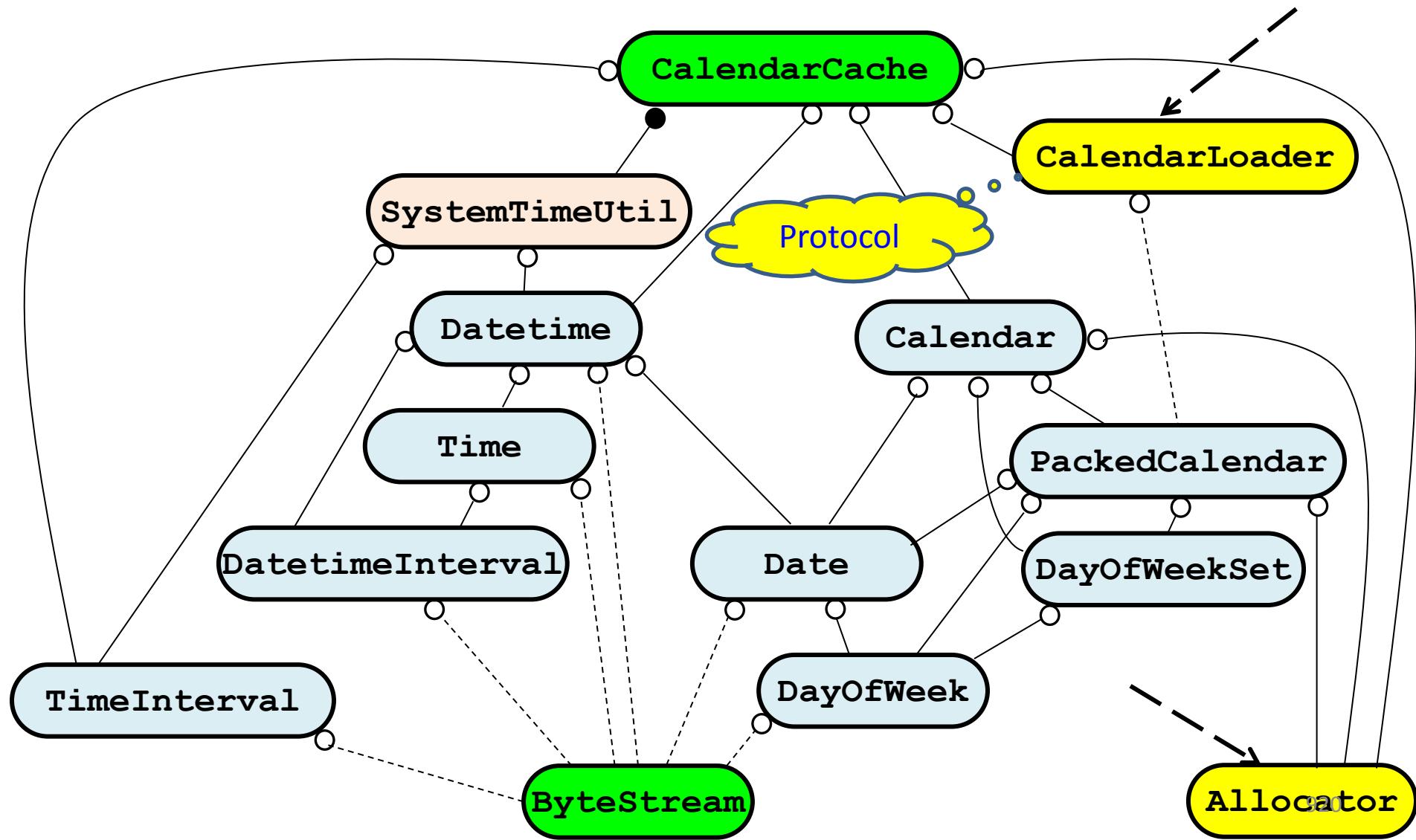
4. Present-Day, Real-World Design Examples

Wait a Minute: Where is the Data Source?



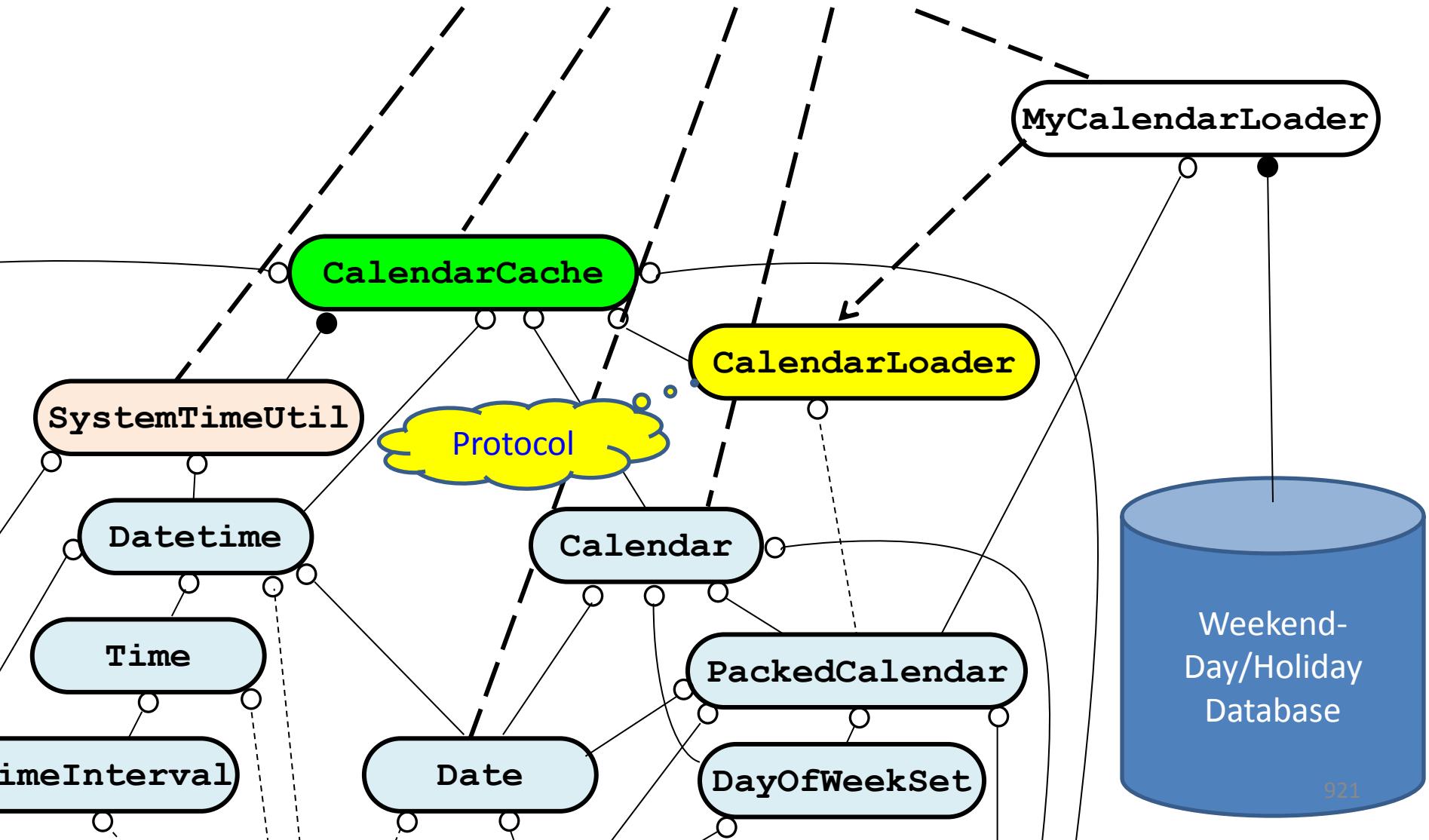
4. Present-Day, Real-World Design Examples

Wait a Minute: Where is the Data Source?



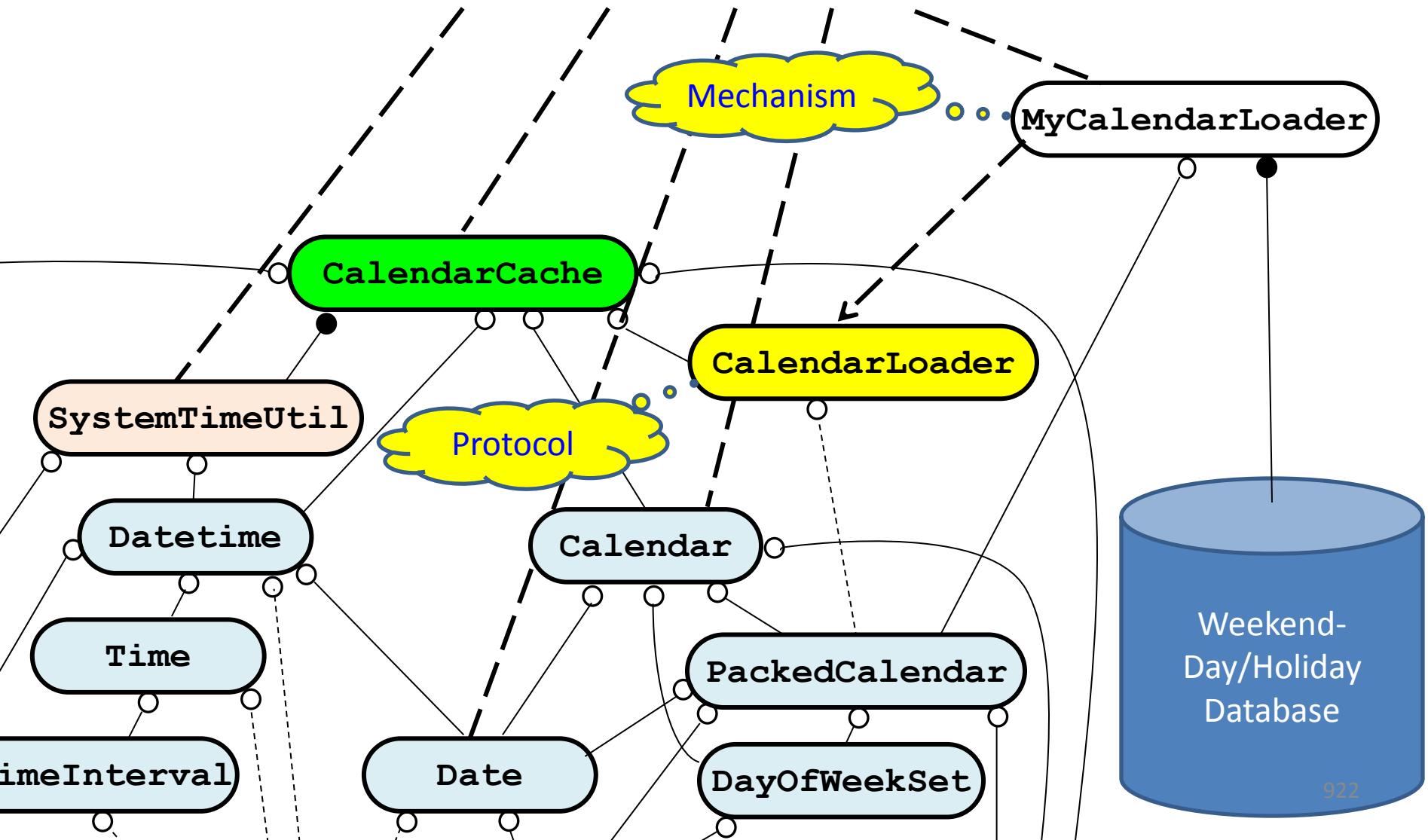
4. Present-Day, Real-World Design Examples

Wait a Minute: Where is the Data Source?



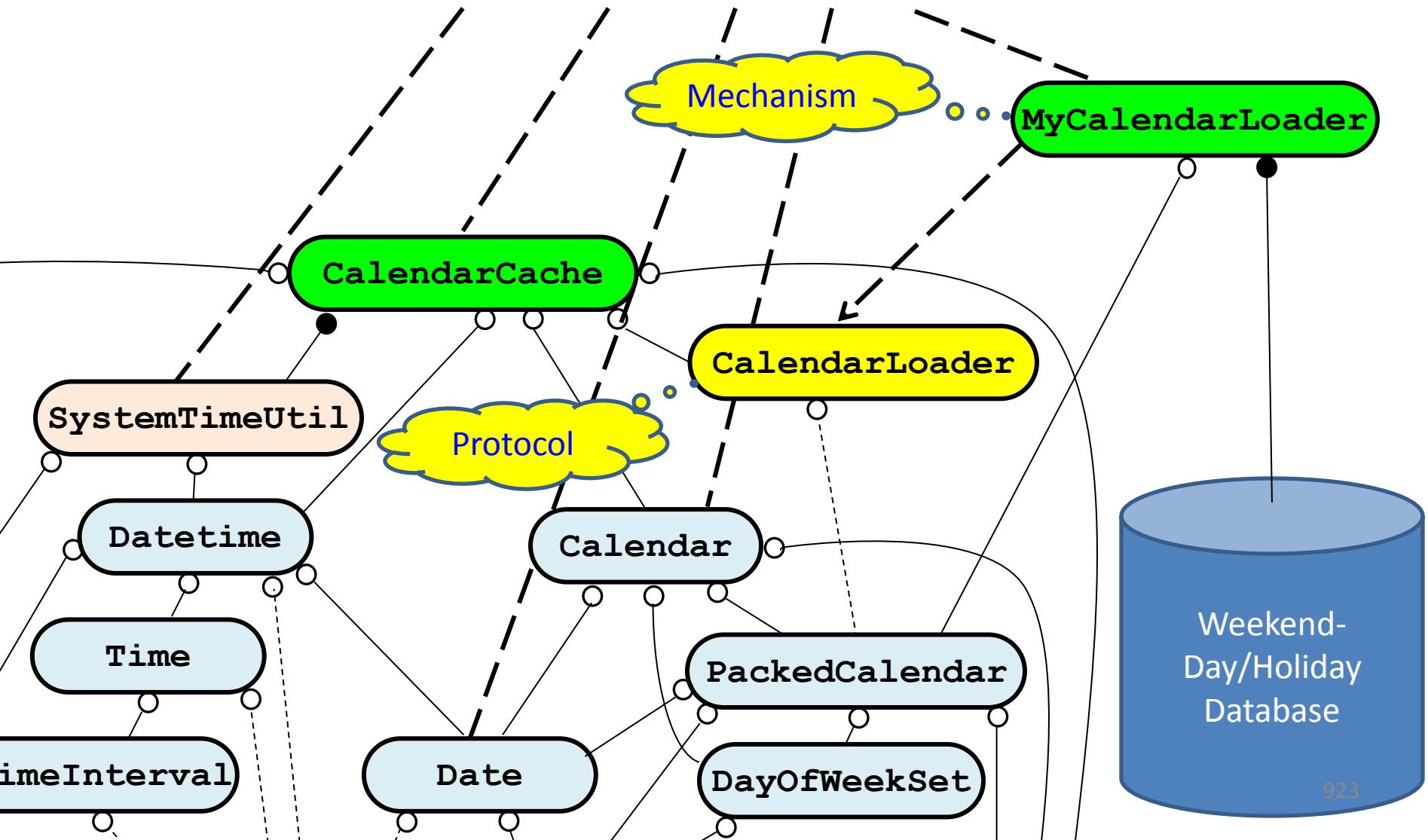
4. Present-Day, Real-World Design Examples

Wait a Minute: Where is the Data Source?



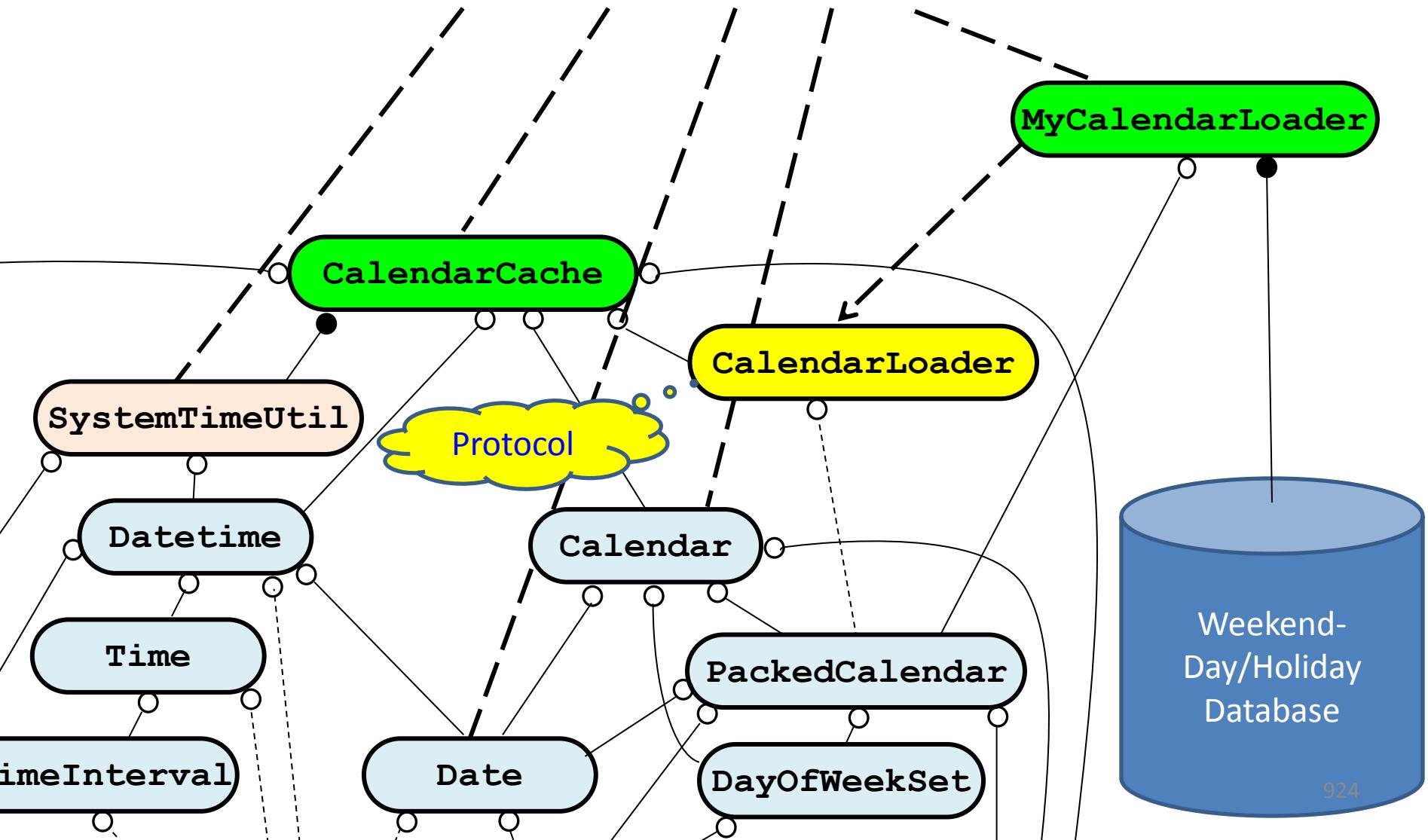
4. Present-Day, Real-World Design Examples

Wait a Minute: Where is the Data Source?



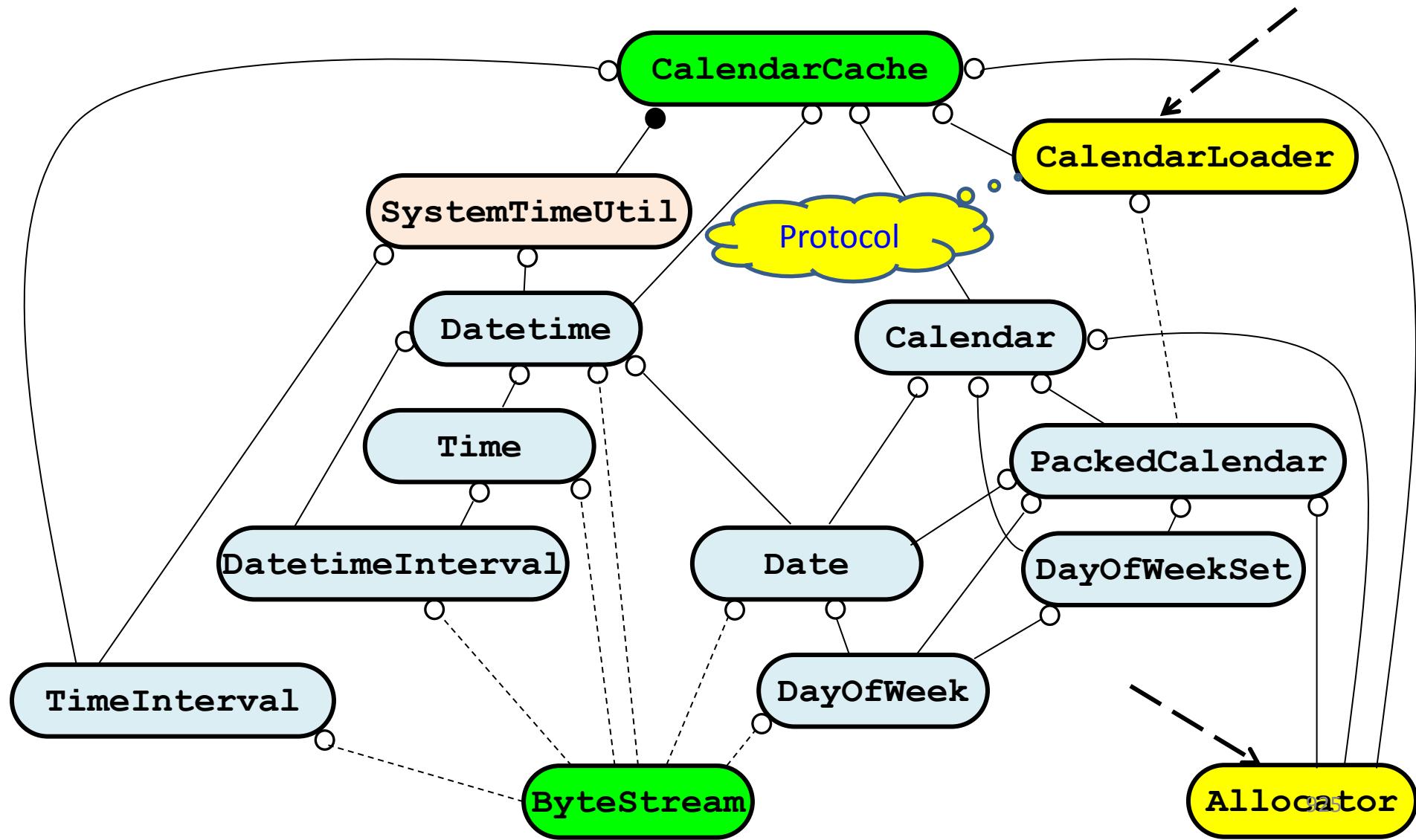
4. Present-Day, Real-World Design Examples

Wait a Minute: Where is the Data Source?



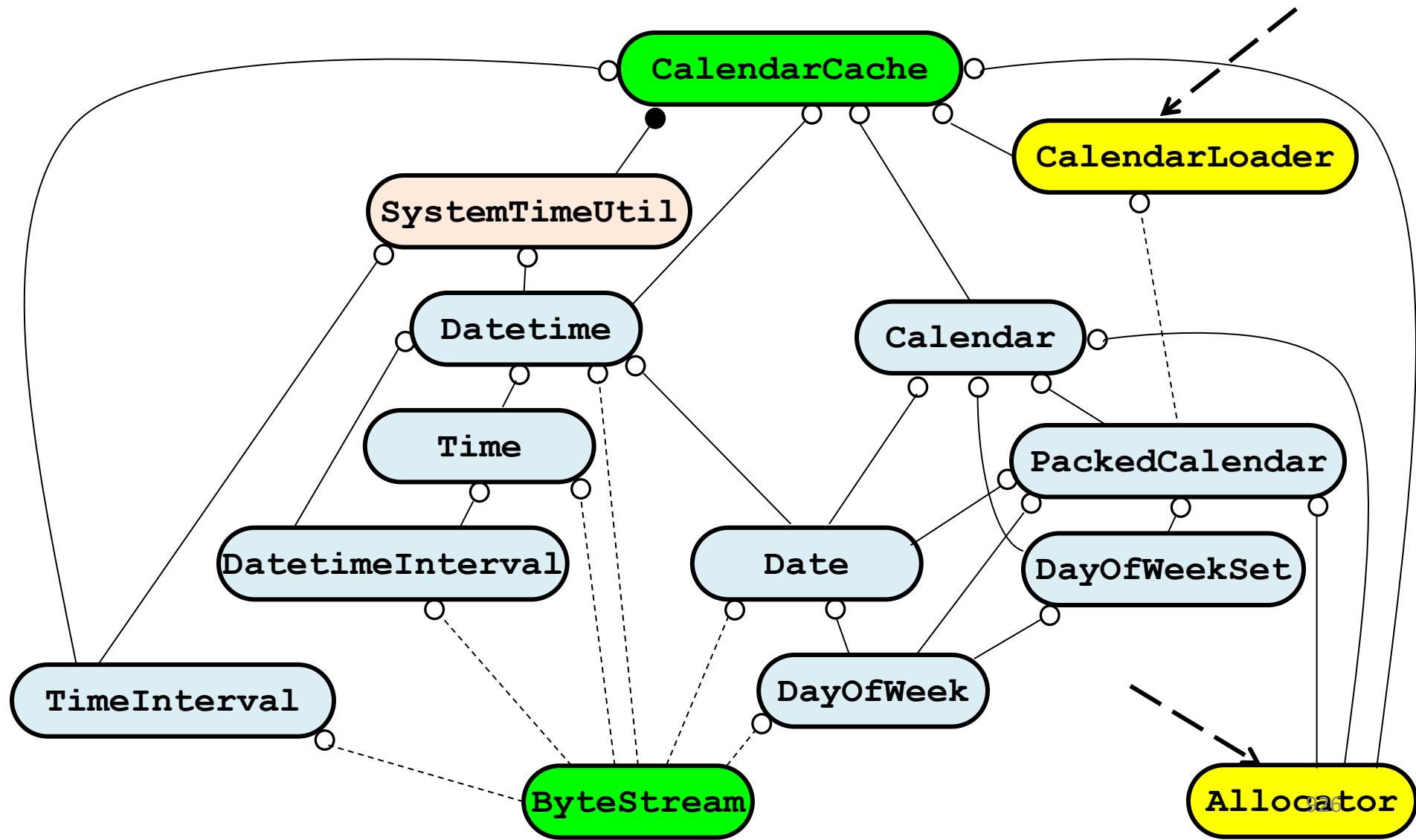
4. Present-Day, Real-World Design Examples

Wait a Minute: Where is the Data Source?



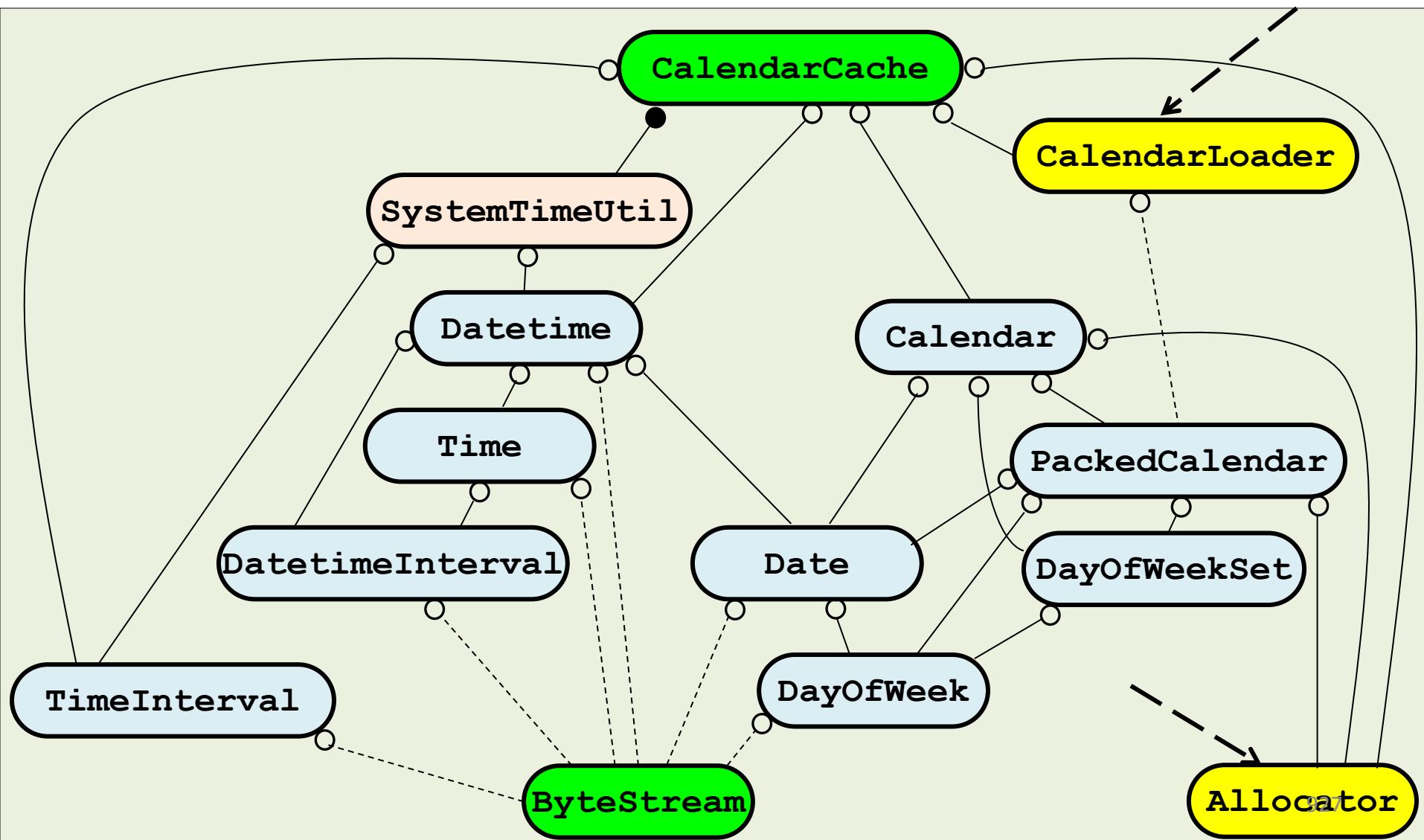
4. Present-Day, Real-World Design Examples

Wait a Minute: Where is the Data Source?



4. Present-Day, Real-World Design Examples

Solution 3: Is Date a Business Day?



4. Present-Day, Real-World Design Examples

The Original Request

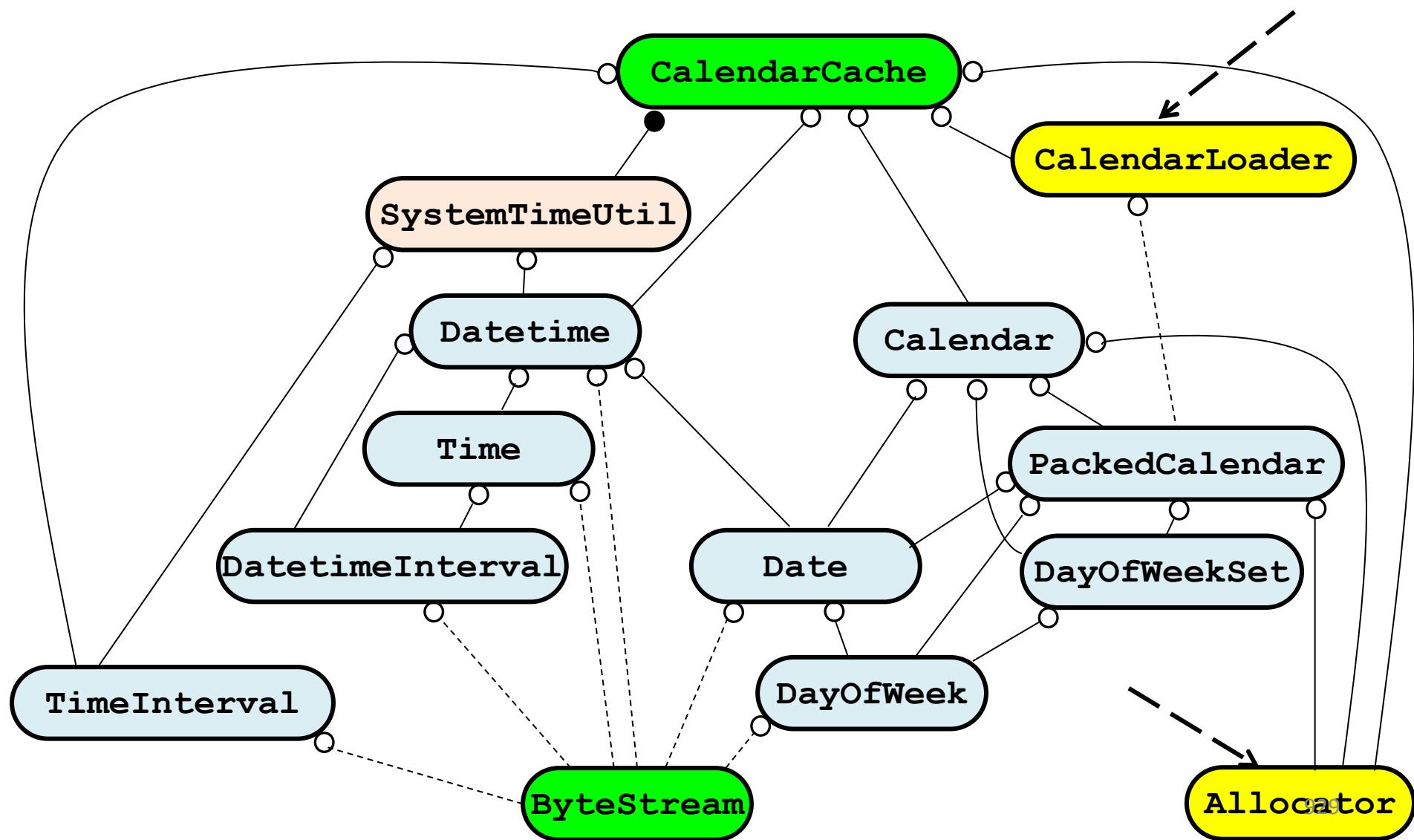
"Write me a 'Date' class that tells me whether today is a business day."

What are the *real* requirements?

1. Represent a *date value* as a C++ Type.
2. Determine what date value *today* is.
3. Determine if a date value is a *business day*.
4. **Provide well-factored useful components that we'll need over and over again!**

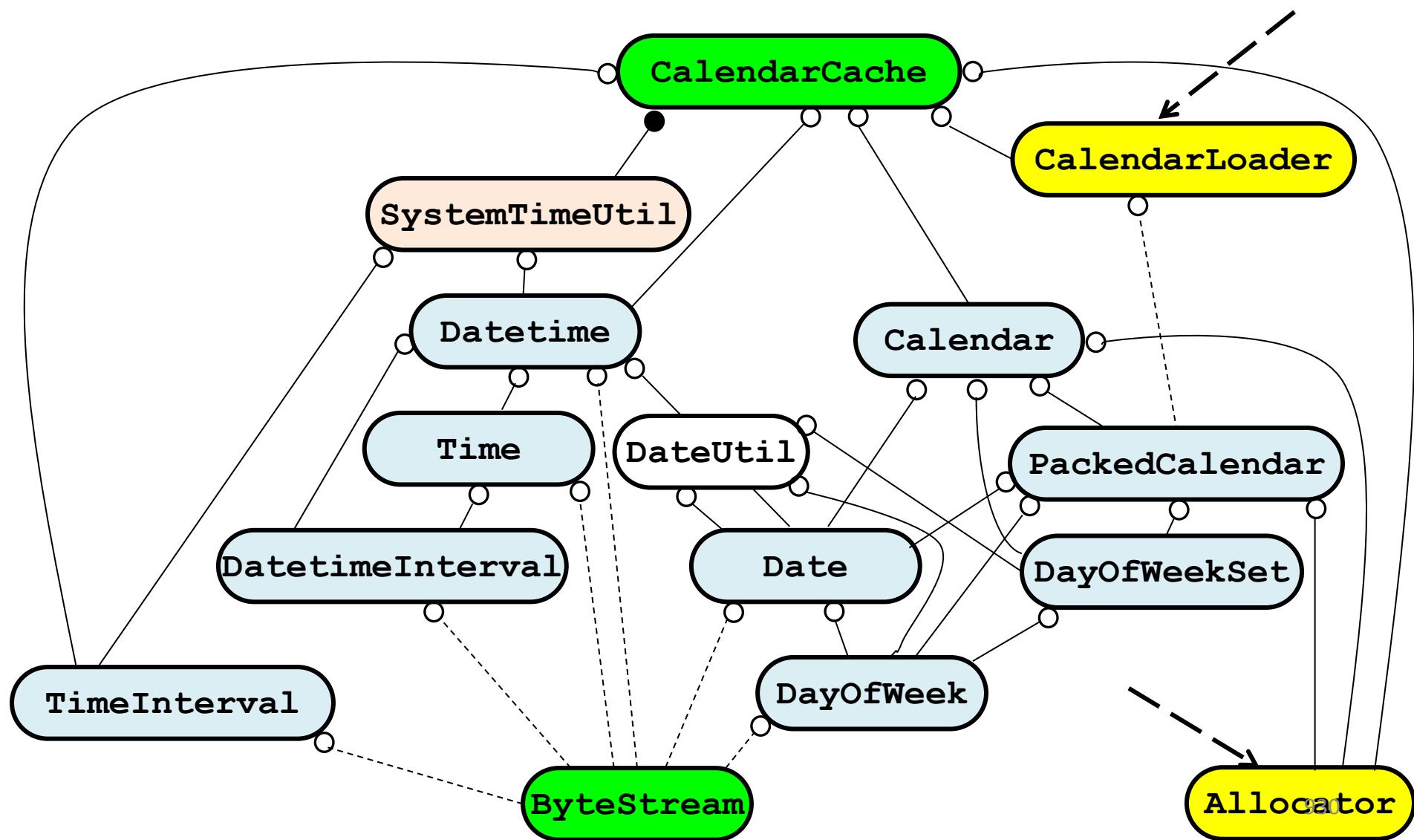
4. Present-Day, Real-World Design Examples

Non-Primitive Functionality



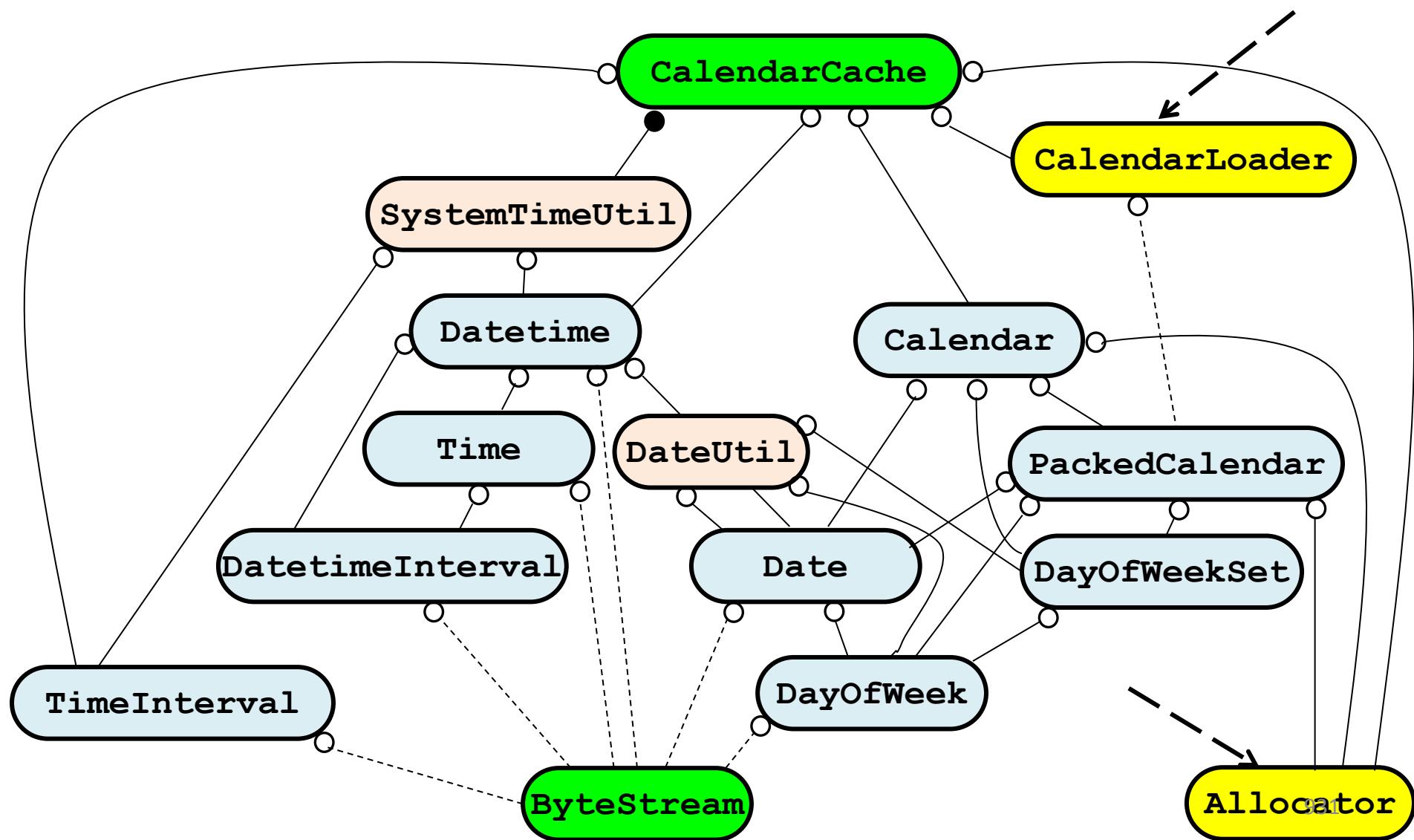
4. Present-Day, Real-World Design Examples

Non-Primitive Functionality



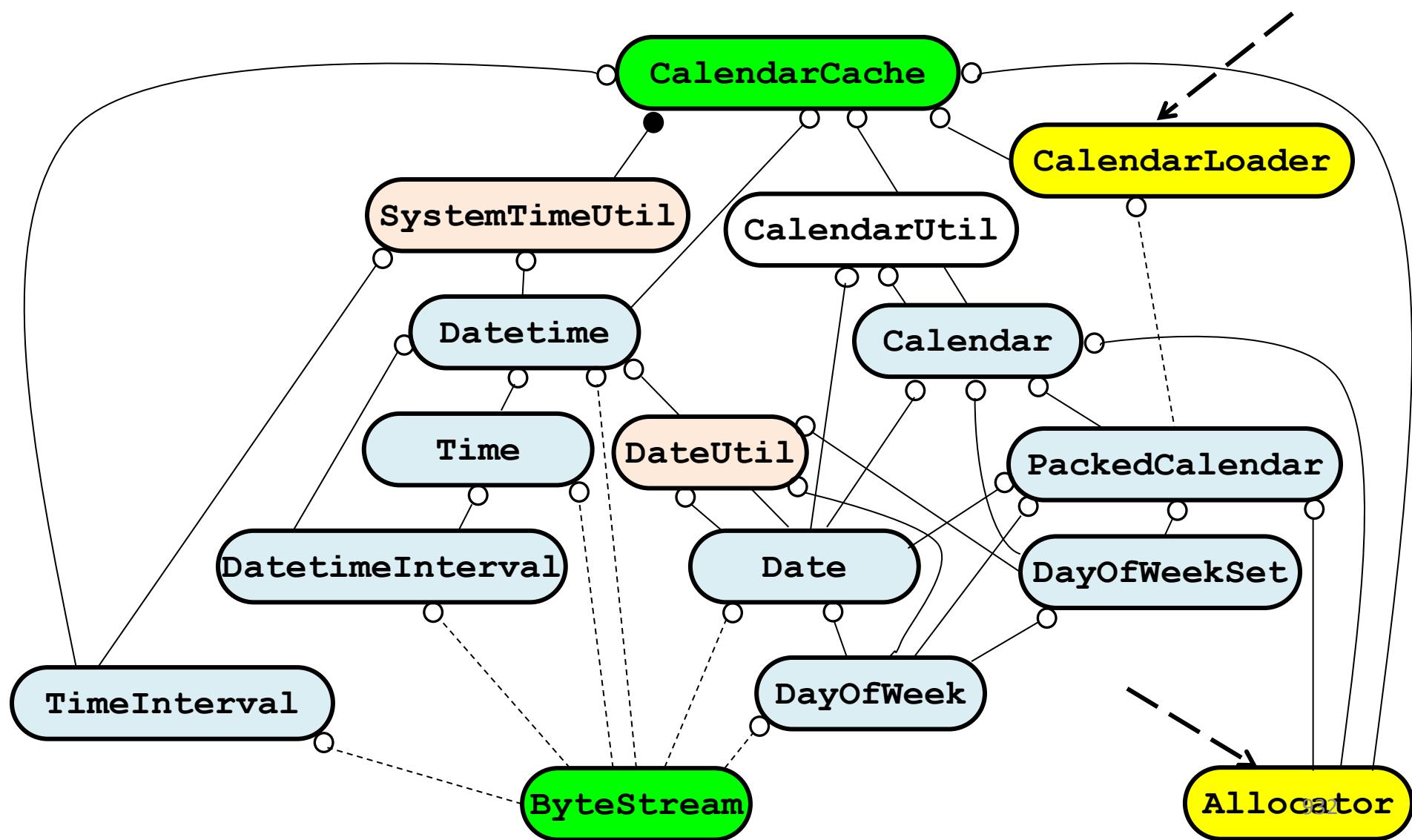
4. Present-Day, Real-World Design Examples

Non-Primitive Functionality



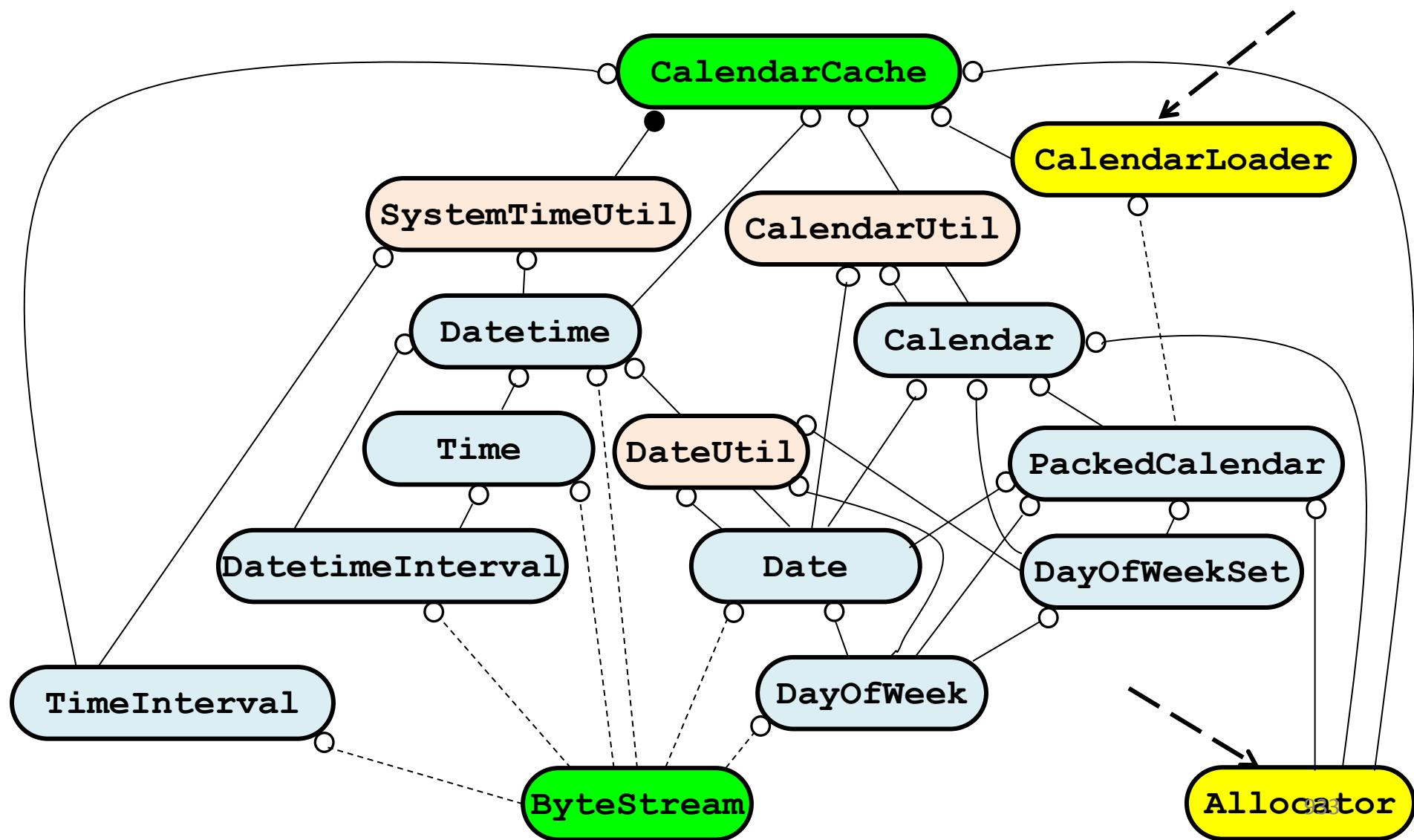
4. Present-Day, Real-World Design Examples

Non-Primitive Functionality



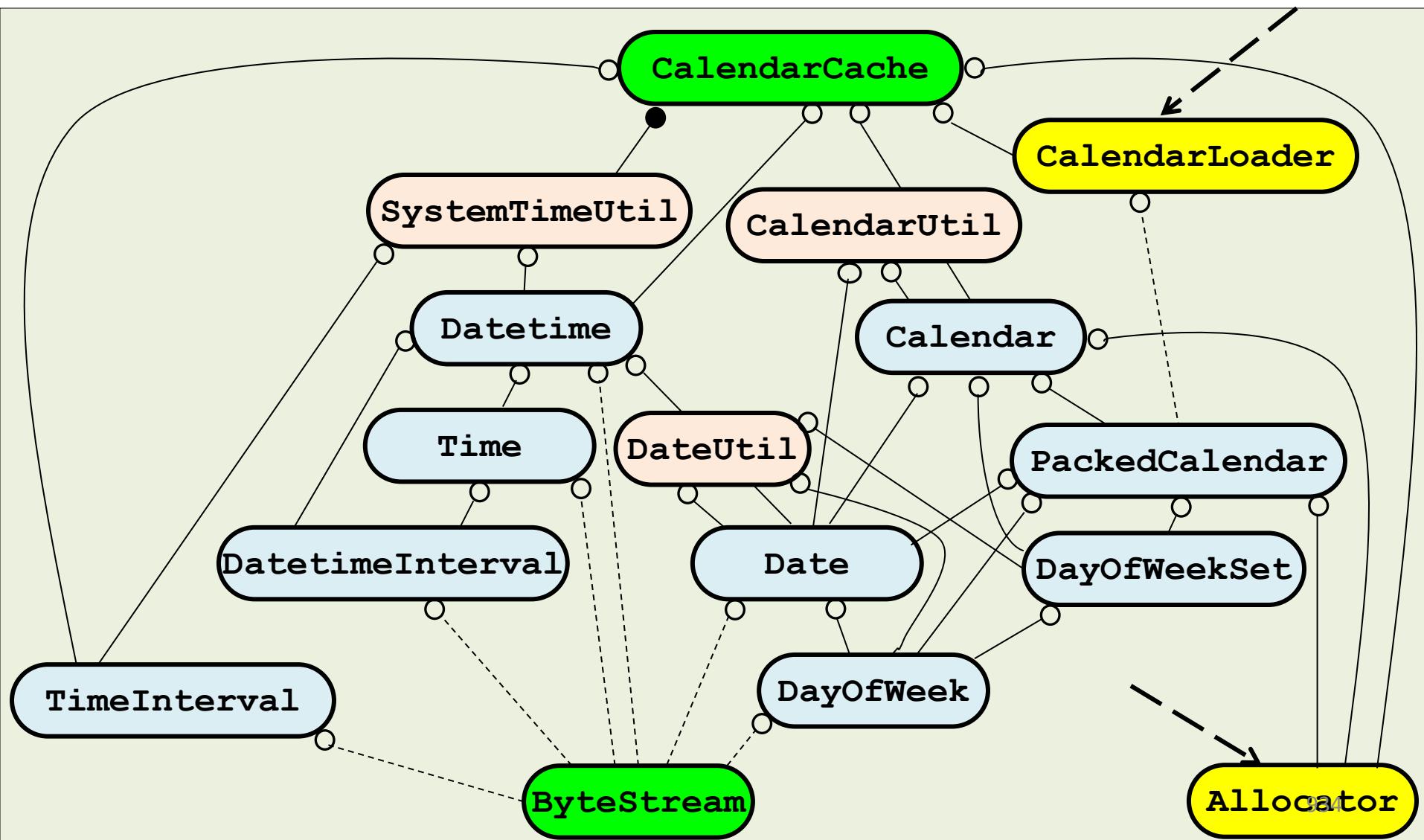
4. Present-Day, Real-World Design Examples

Non-Primitive Functionality



4. Present-Day, Real-World Design Examples

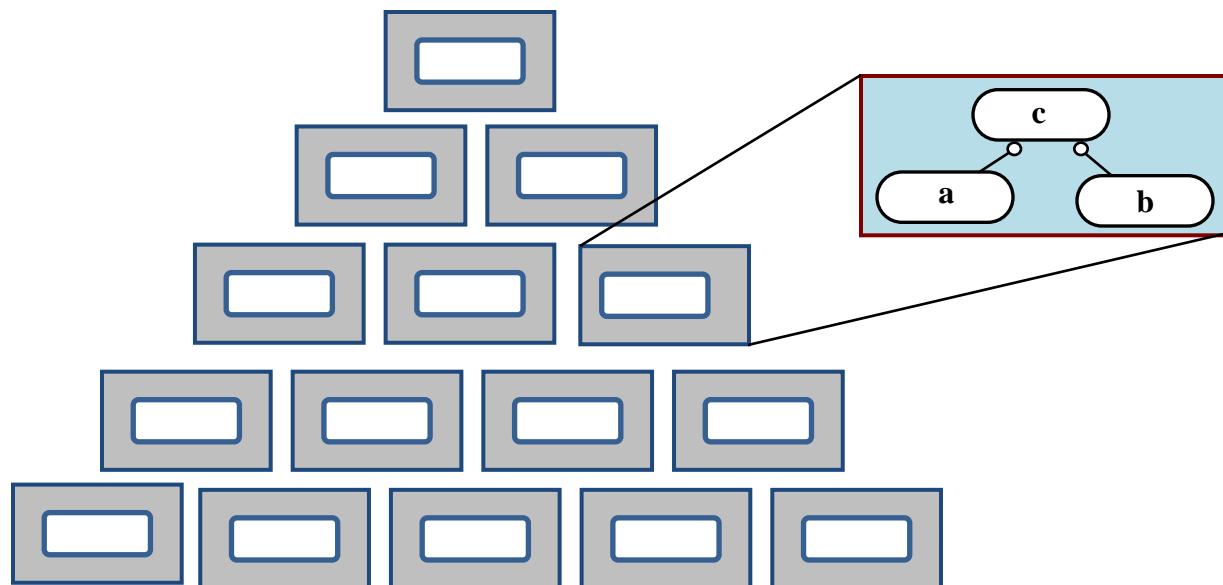
Fine-Grained Reusable Class Design



4. Present-Day, Real-World Design Examples

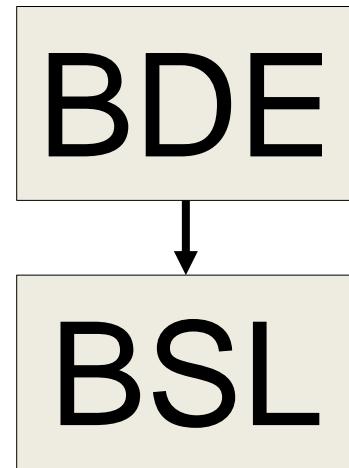
Rendering Software as Components

Logical content aggregated into a
Physical hierarchy of **components**



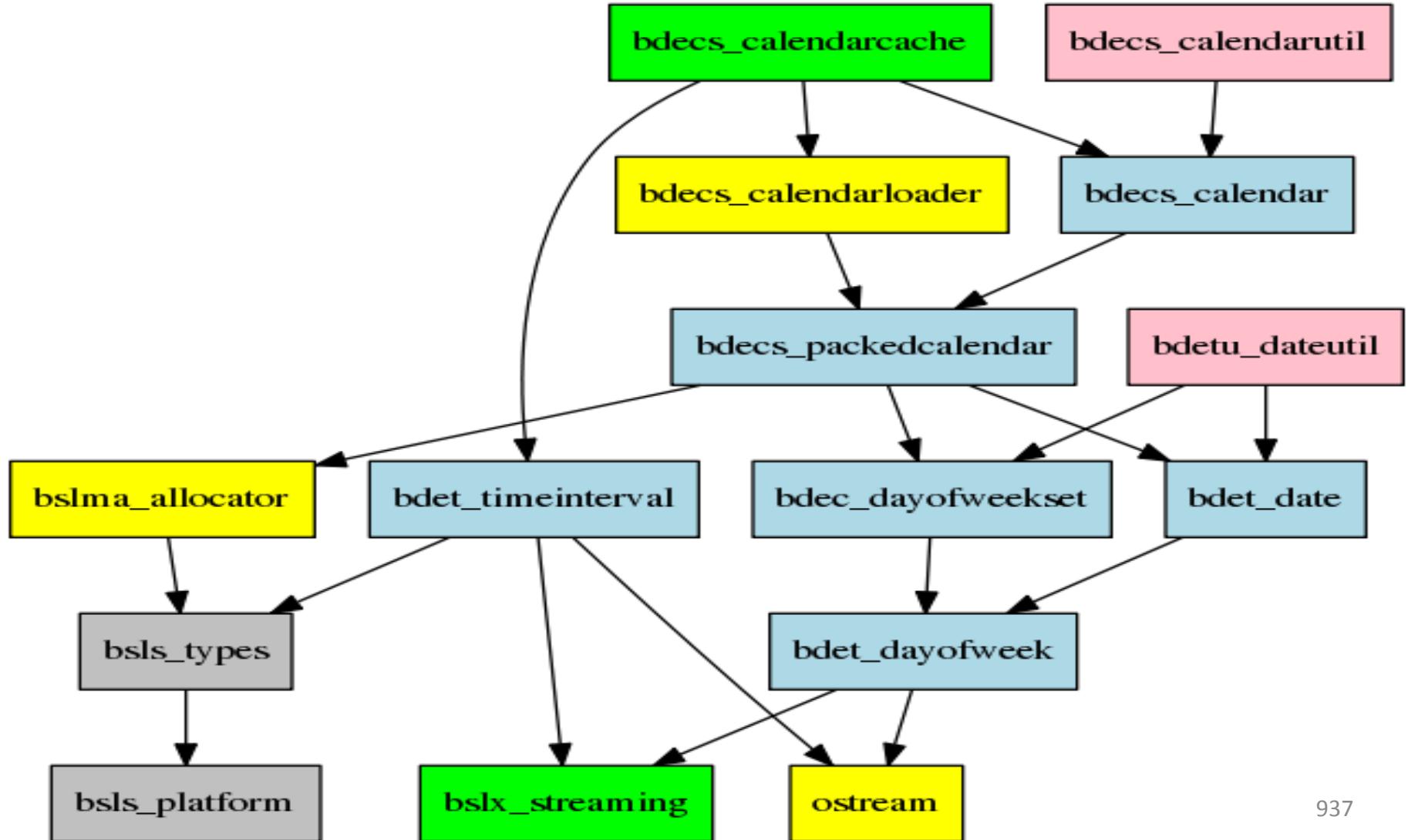
4. Present-Day, Real-World Design Examples

Package Group Dependencies



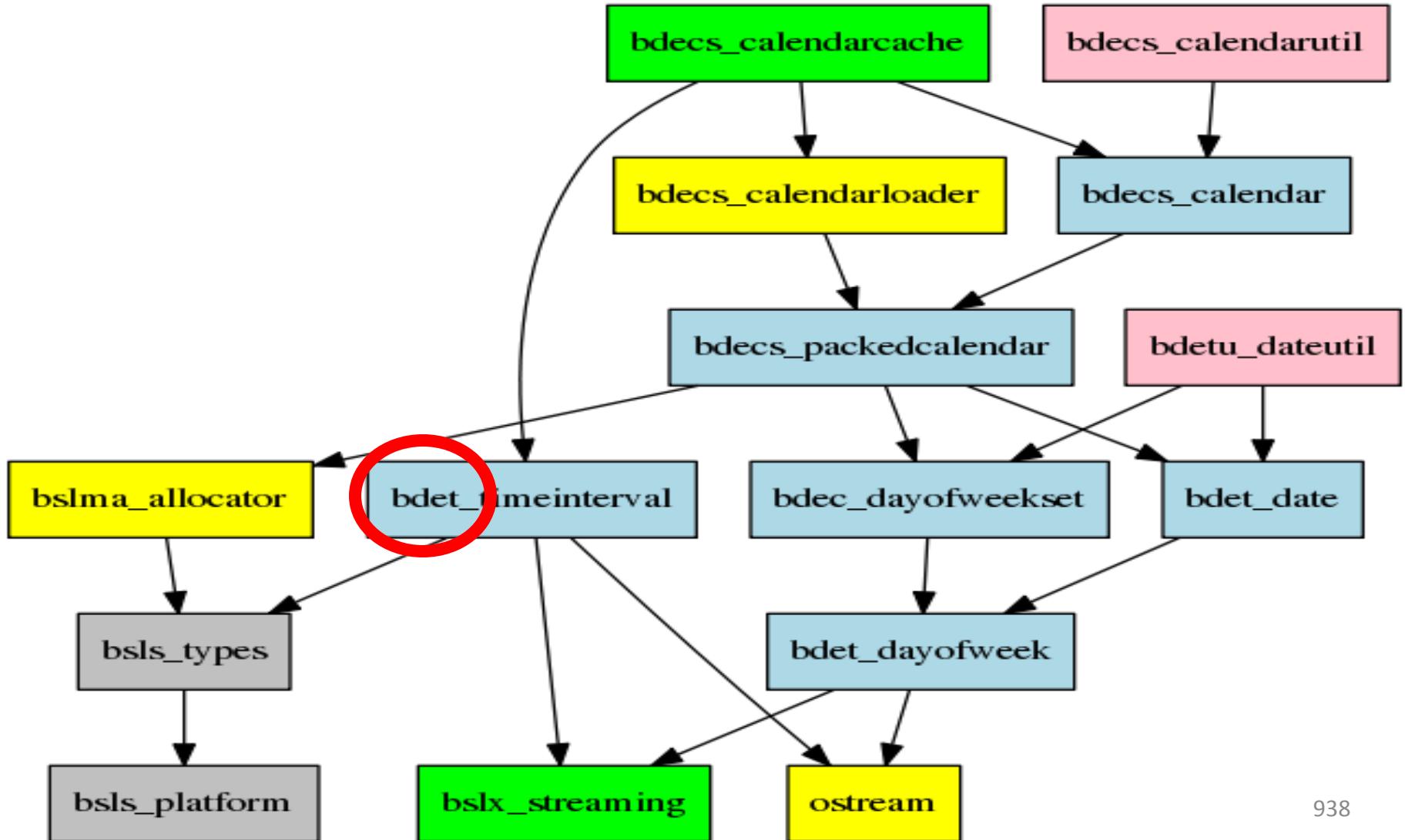
4. Present-Day, Real-World Design Examples

Client-Facing Component Diagram



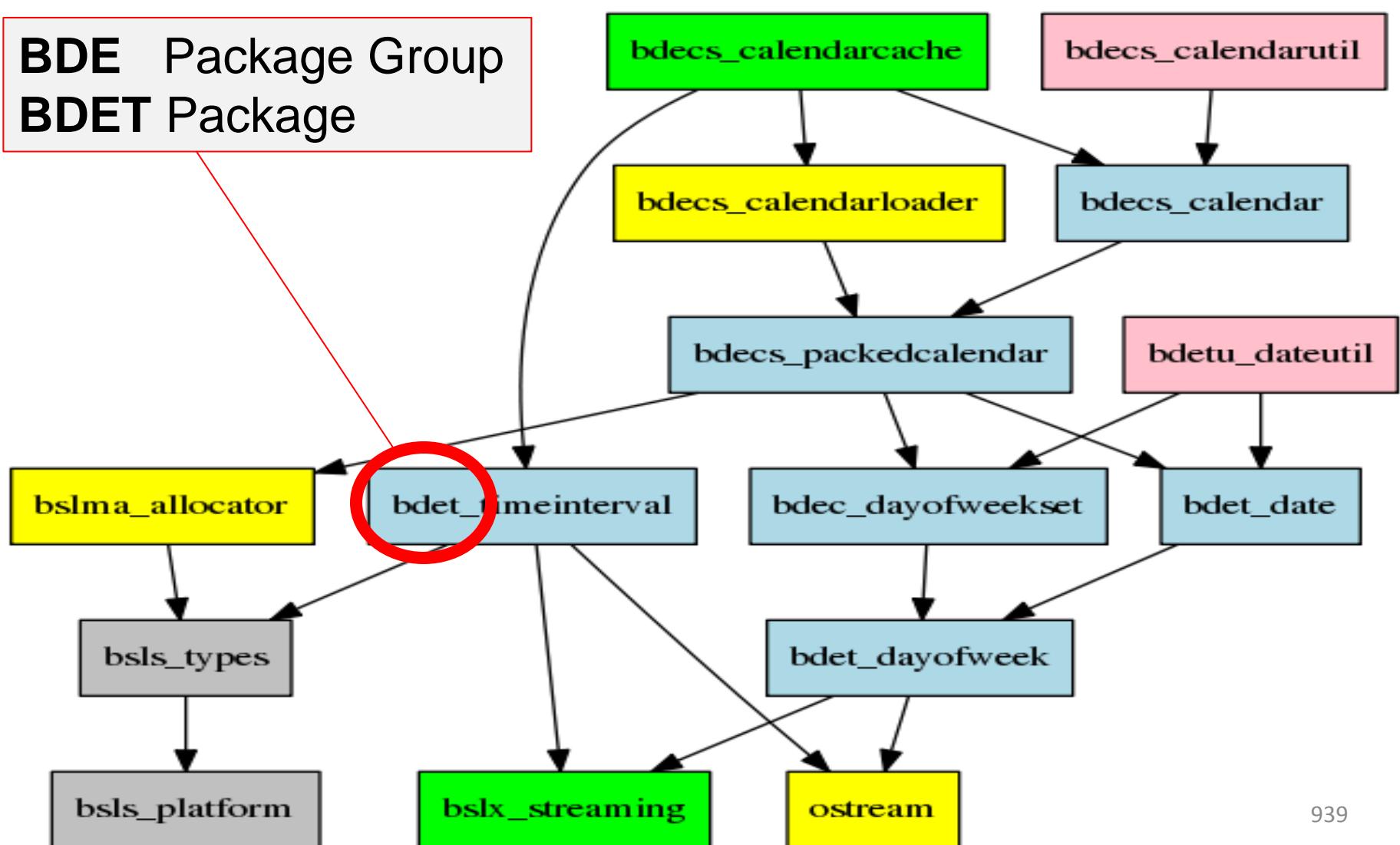
4. Present-Day, Real-World Design Examples

Client-Facing Component Diagram



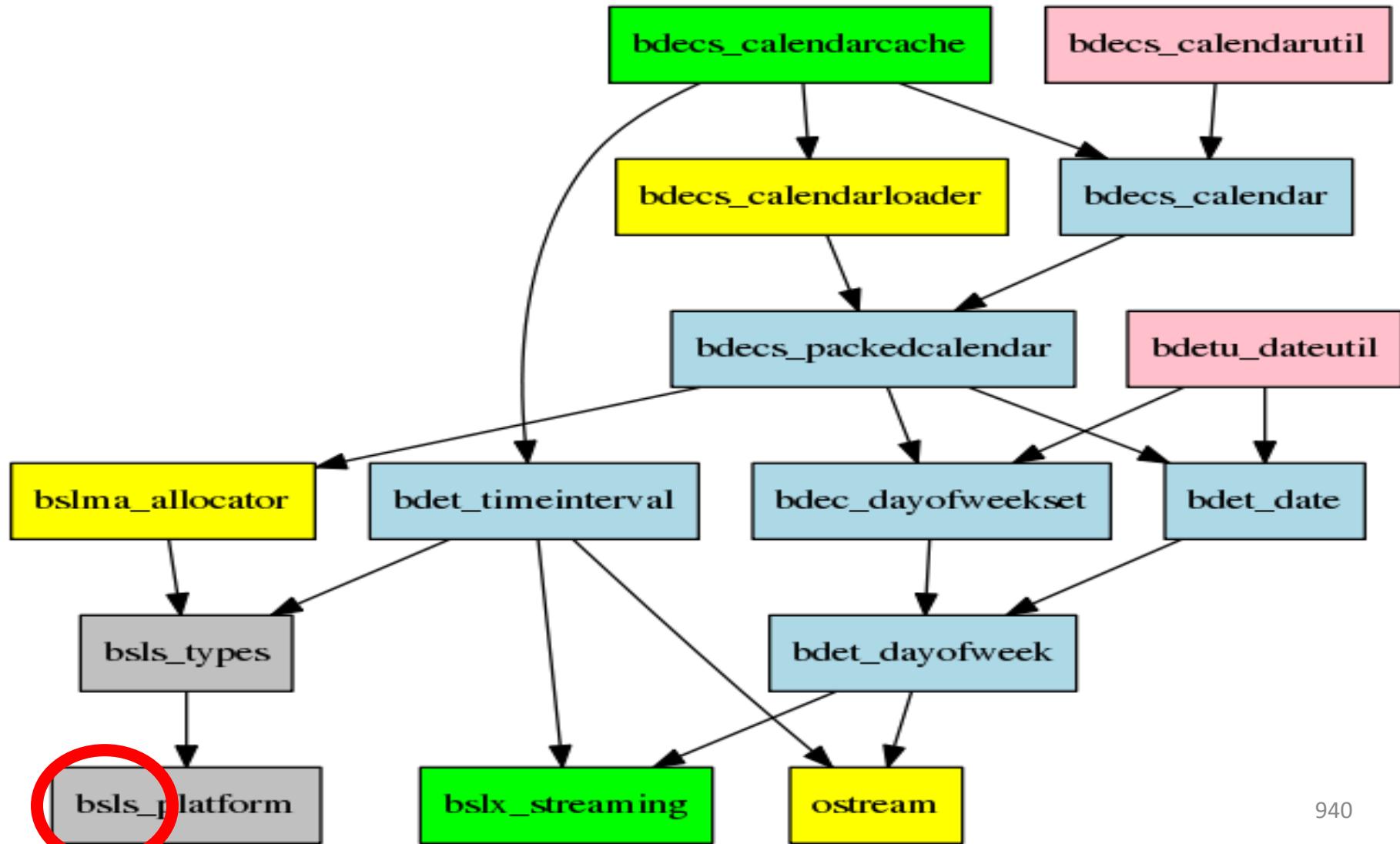
4. Present-Day, Real-World Design Examples

Client-Facing Component Diagram



4. Present-Day, Real-World Design Examples

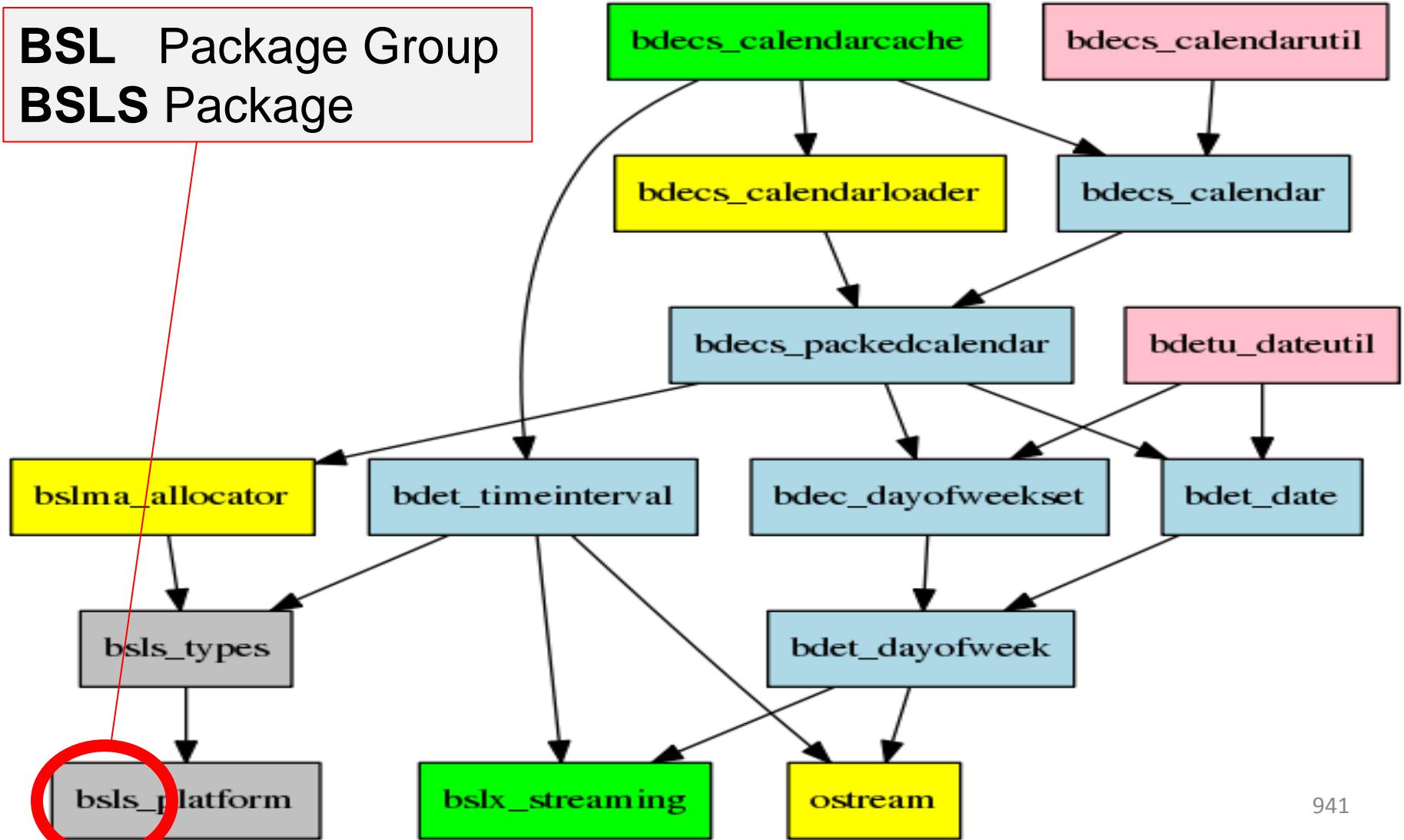
Client-Facing Component Diagram



4. Present-Day, Real-World Design Examples

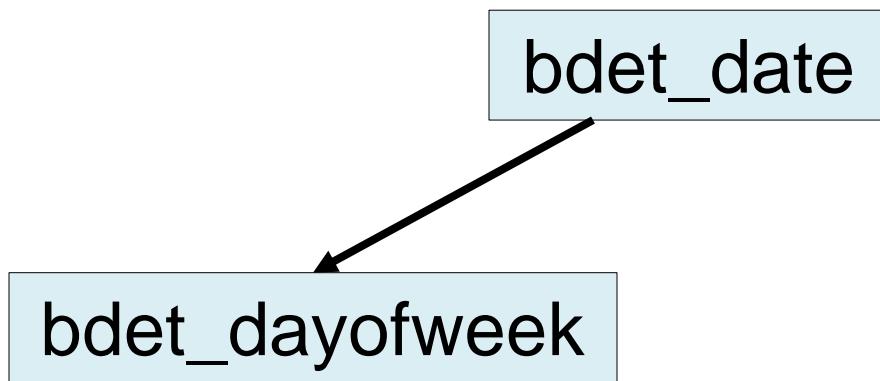
Client-Facing Component Diagram

BSL Package Group
BSLS Package



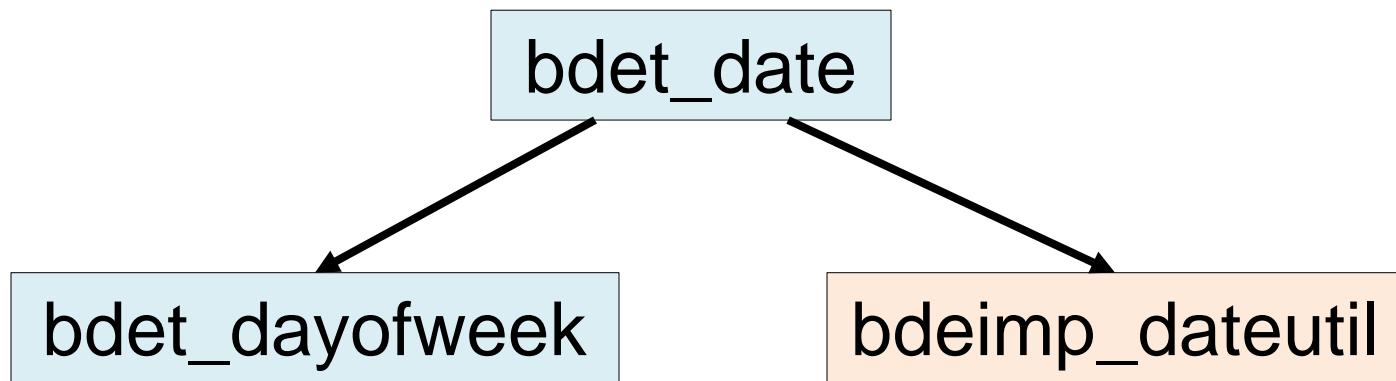
4. Present-Day, Real-World Design Examples

Implementing bdet_date



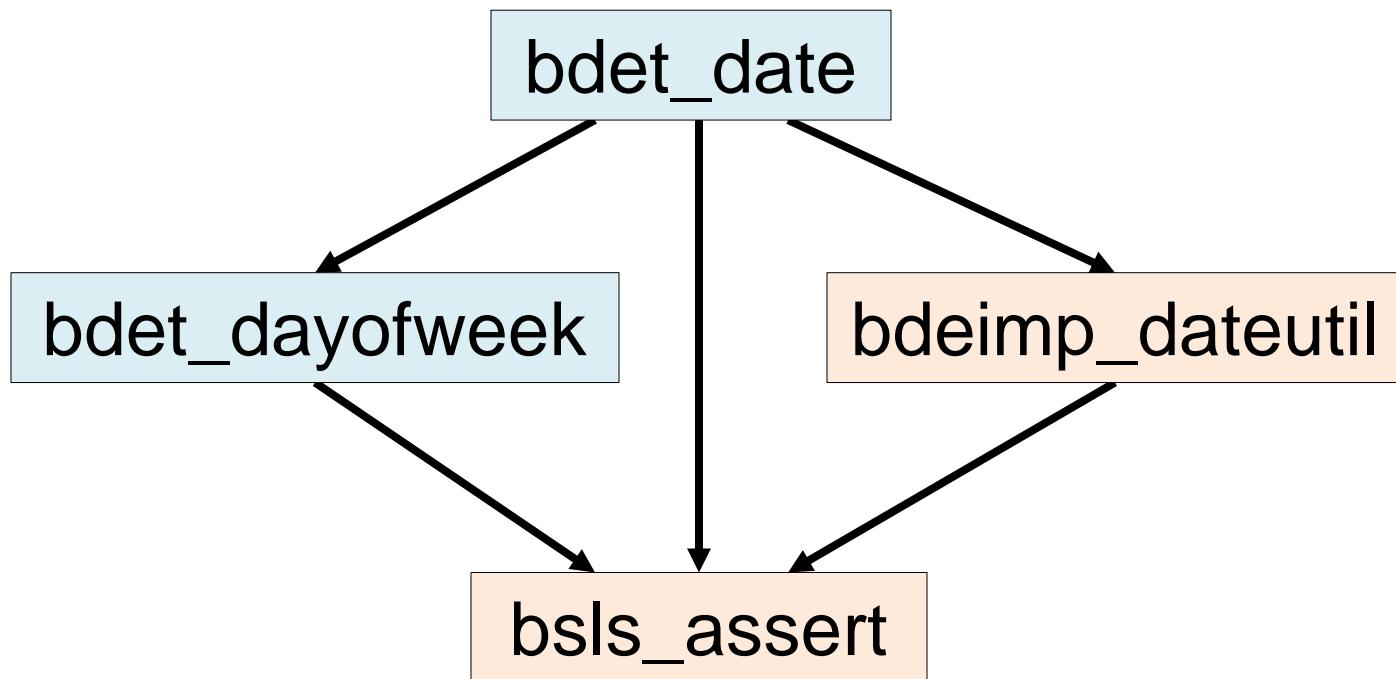
4. Present-Day, Real-World Design Examples

Implementing bdet_date



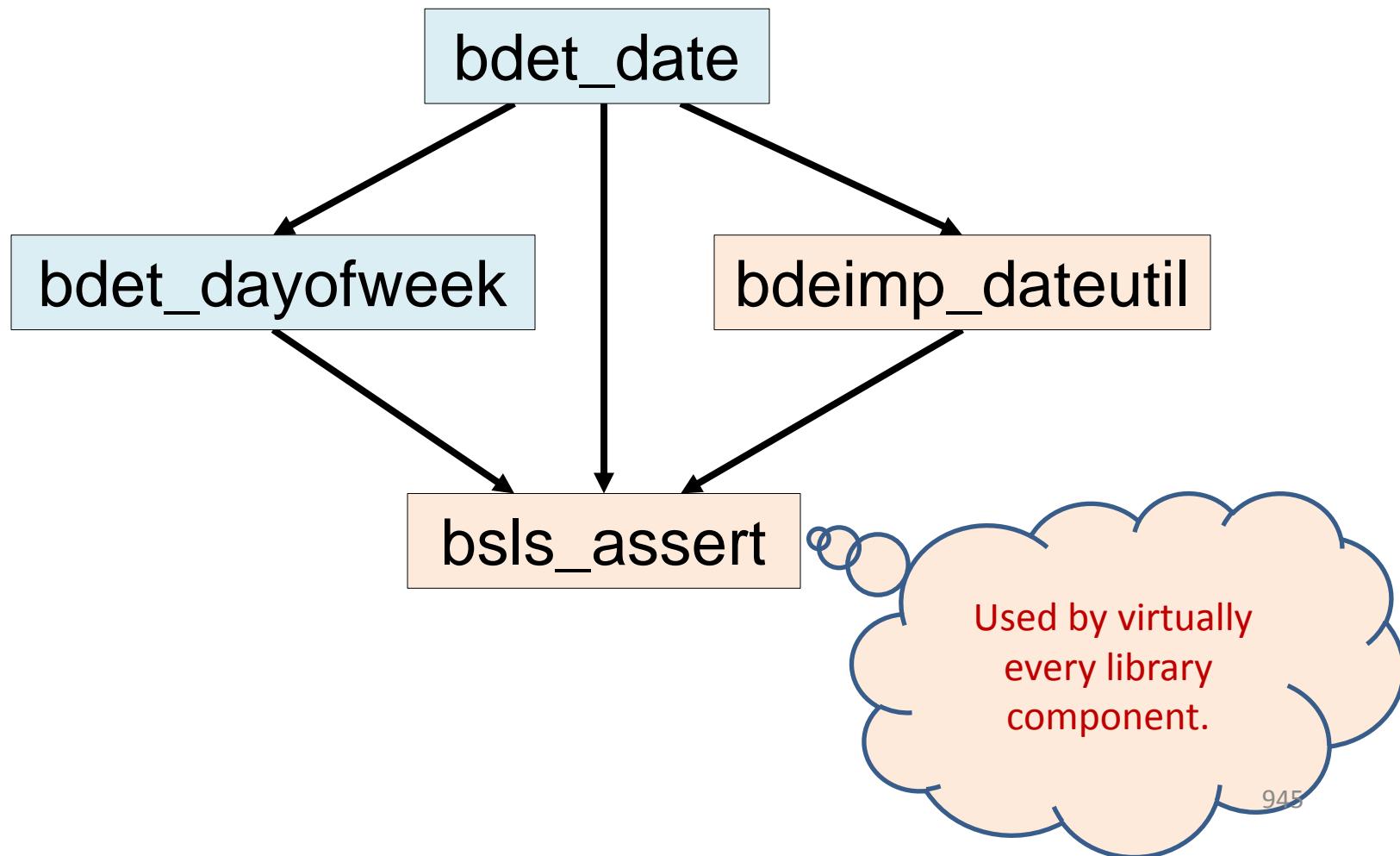
4. Present-Day, Real-World Design Examples

Implementing bdet_date



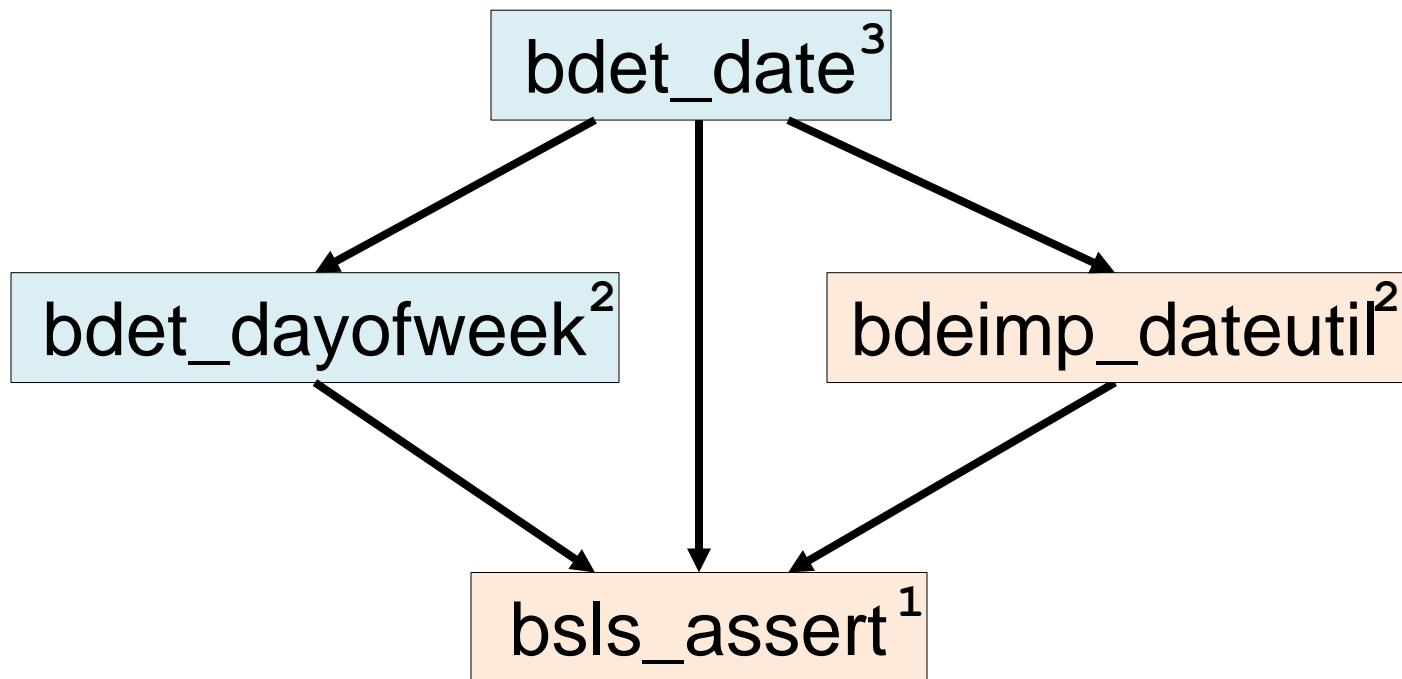
4. Present-Day, Real-World Design Examples

Implementing `bdet_date`



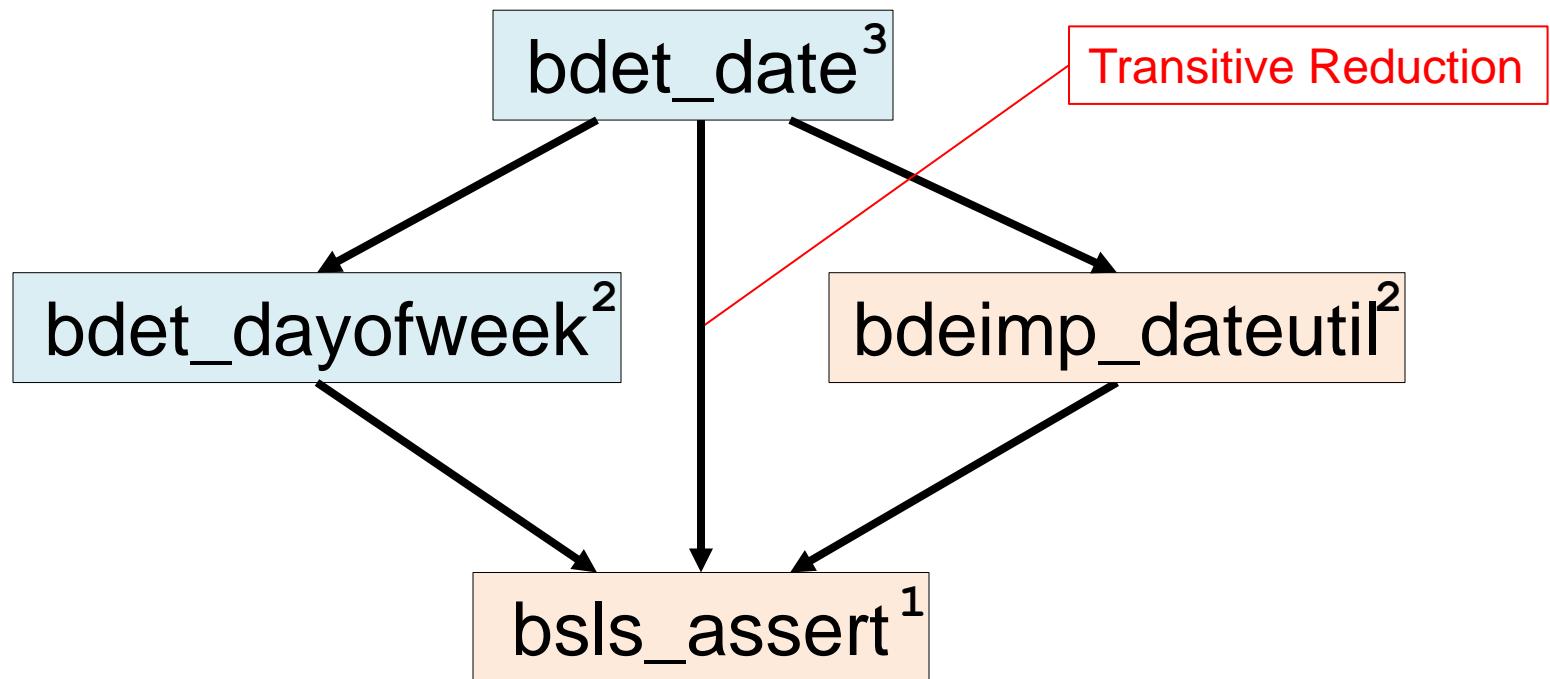
4. Present-Day, Real-World Design Examples

Implementing bdet_date



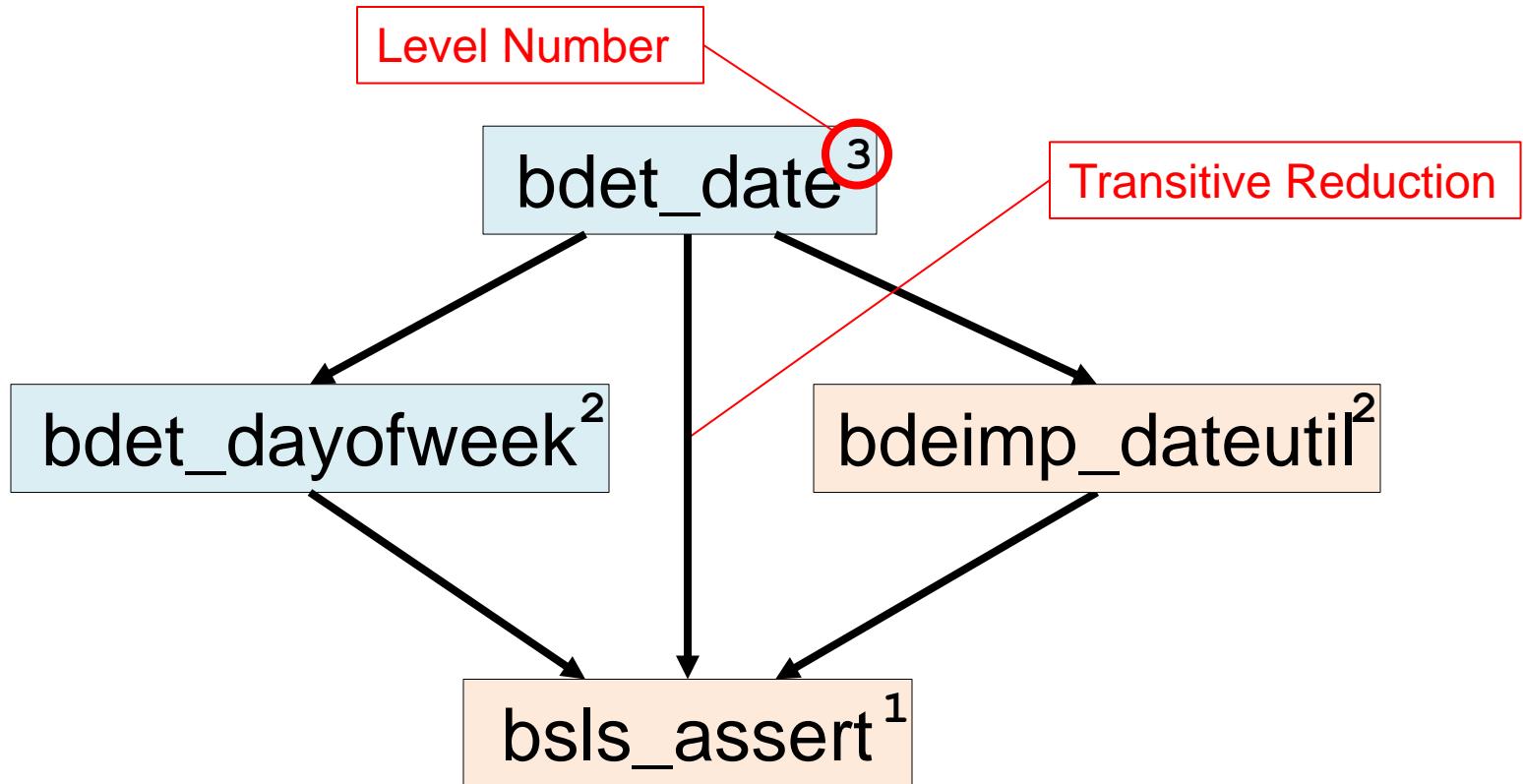
4. Present-Day, Real-World Design Examples

Implementing bdet_date



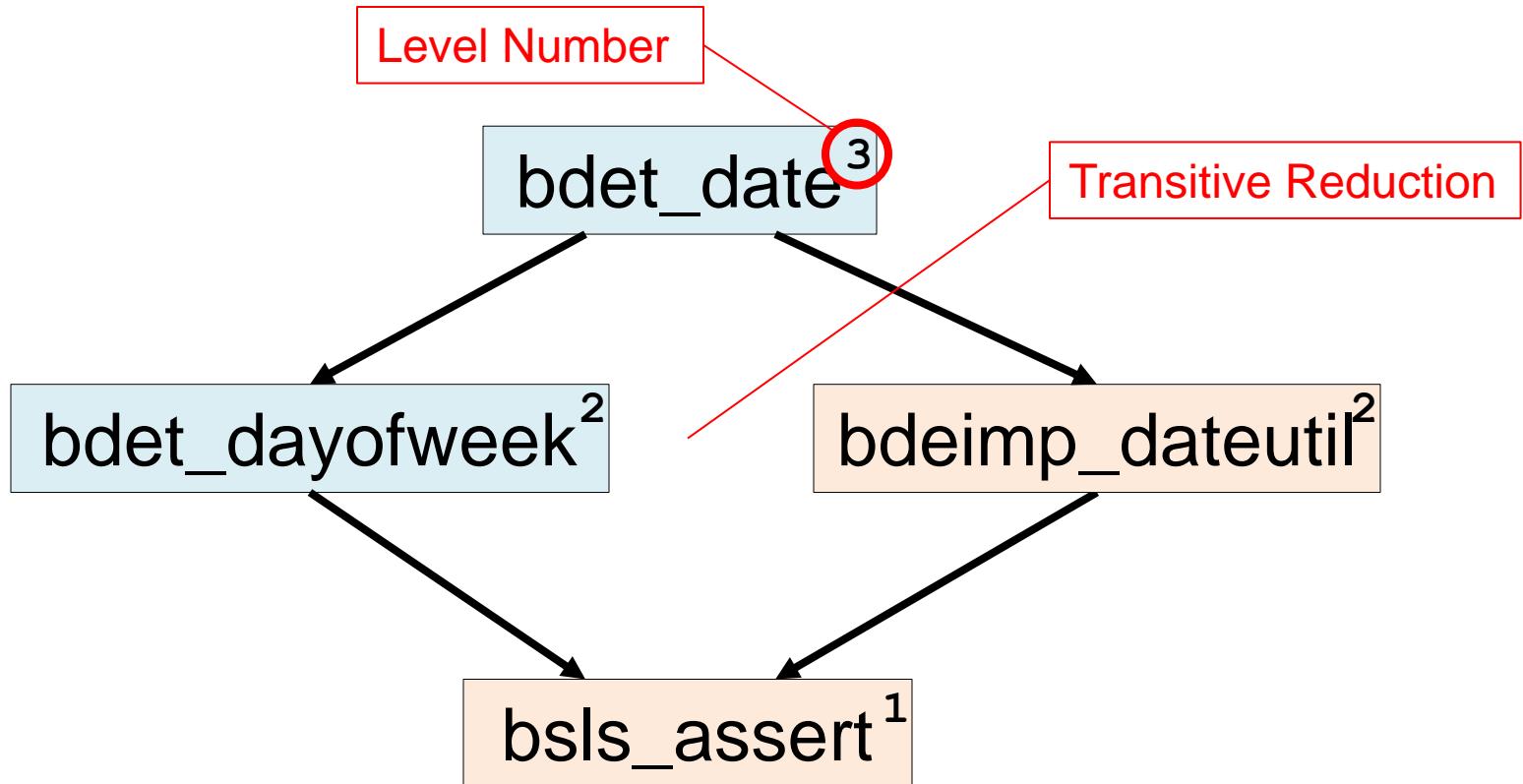
4. Present-Day, Real-World Design Examples

Implementing bdet_date



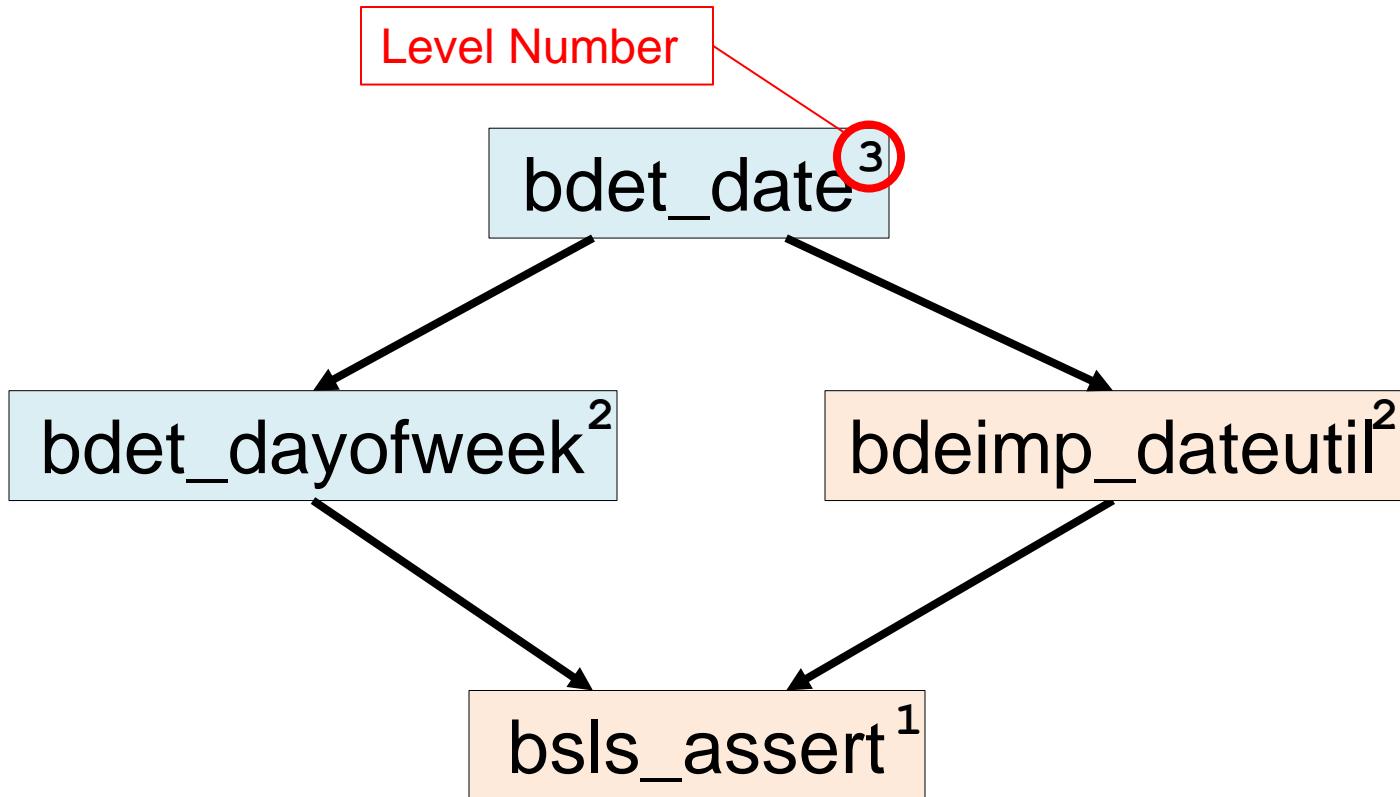
4. Present-Day, Real-World Design Examples

Implementing bdet_date



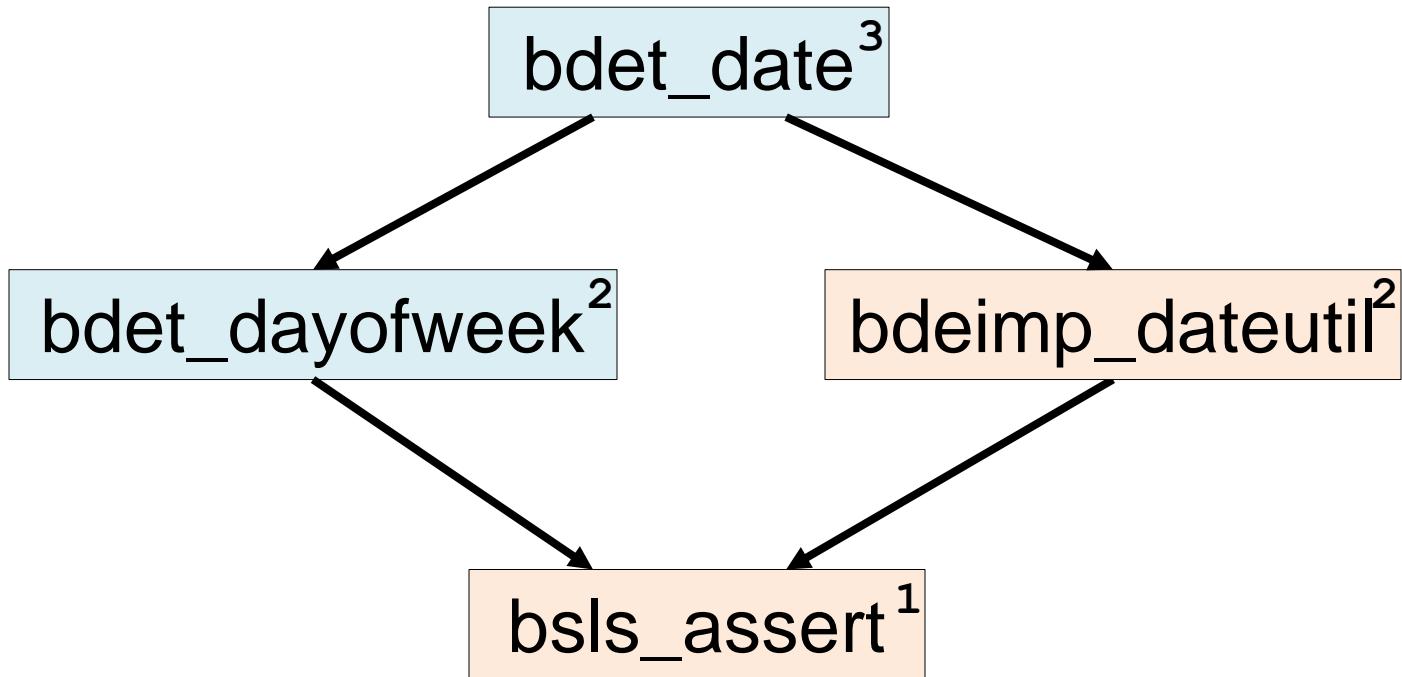
4. Present-Day, Real-World Design Examples

Implementing bdet_date



4. Present-Day, Real-World Design Examples

Implementing bdet_date

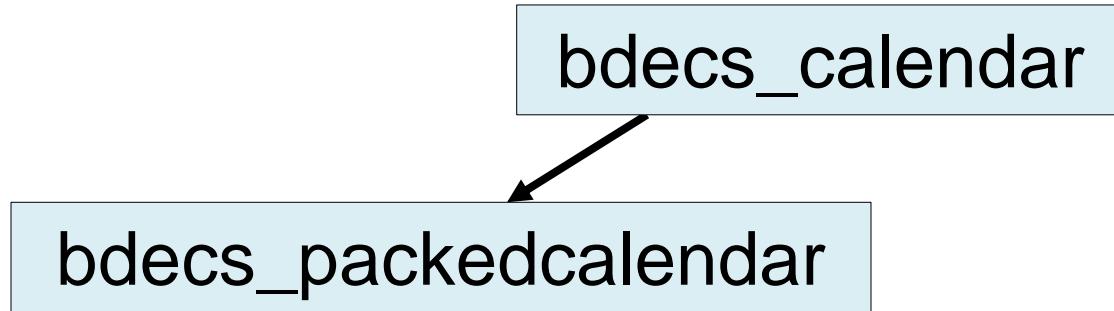


4. Present-Day, Real-World Design Examples

Implementing bdecs_calendar

4. Present-Day, Real-World Design Examples

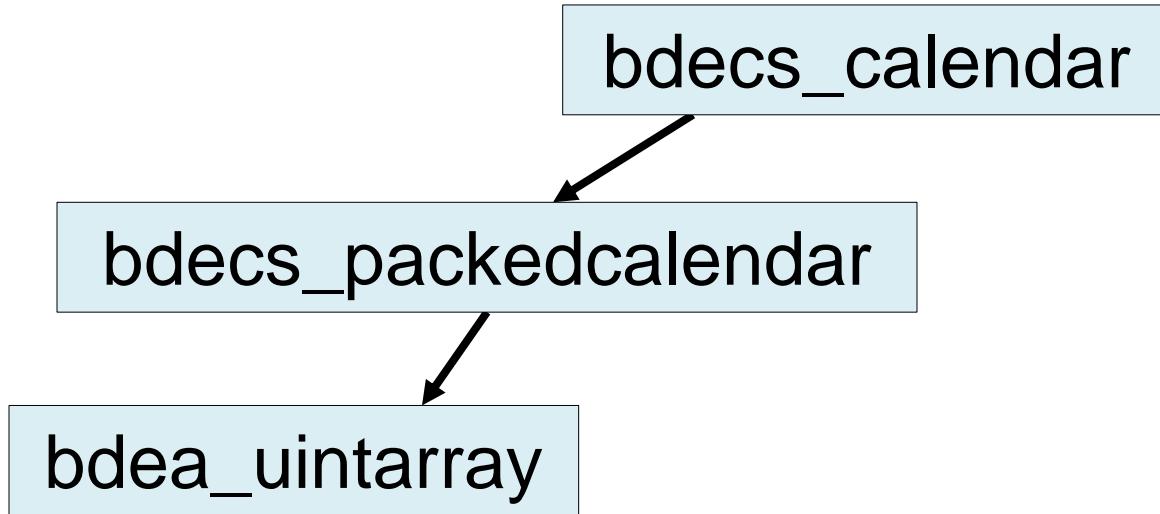
Implementing `bdecs_calendar`



bslma_allocator

4. Present-Day, Real-World Design Examples

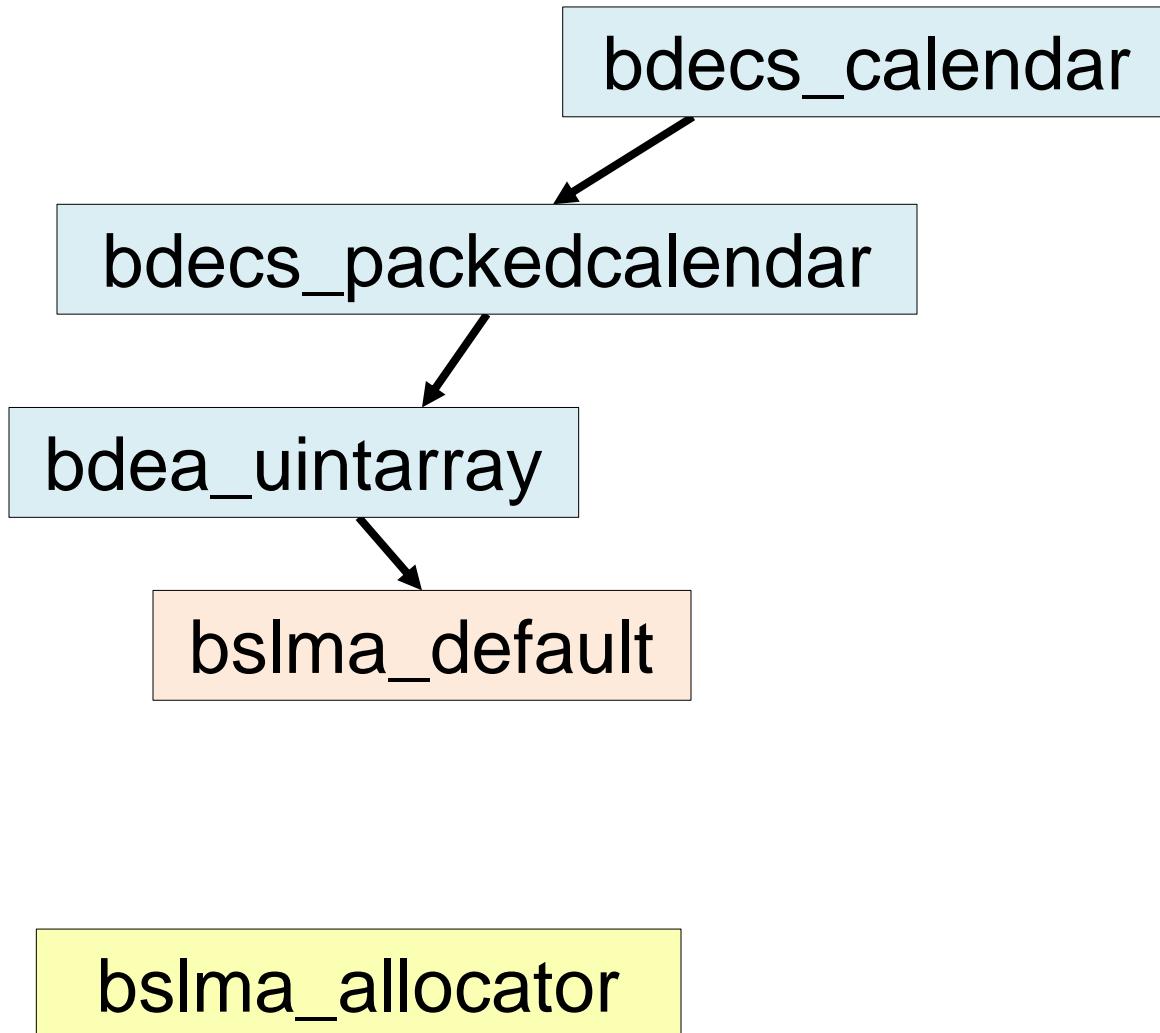
Implementing `bdecs_calendar`



`bslma_allocator`

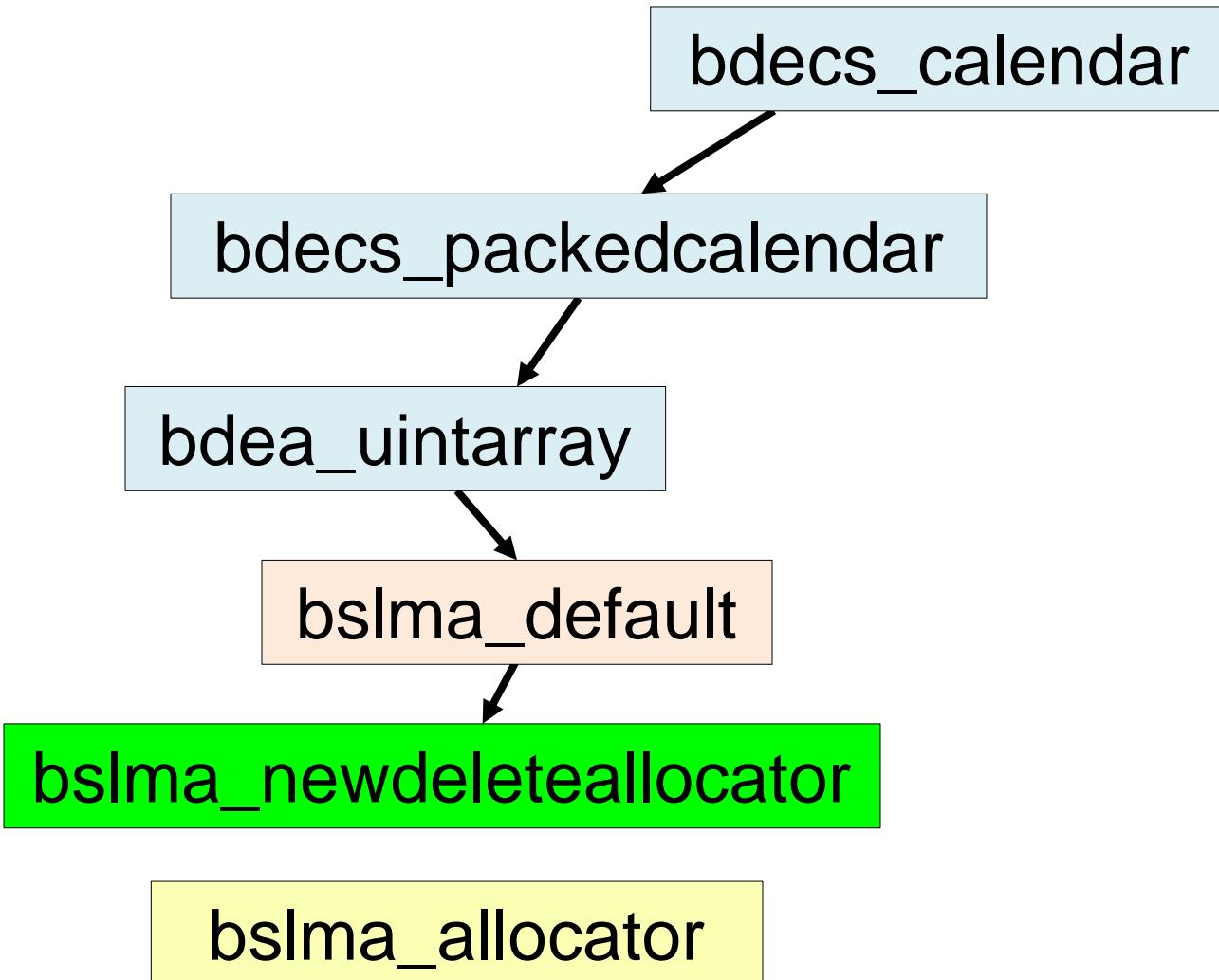
4. Present-Day, Real-World Design Examples

Implementing `bdecs_calendar`



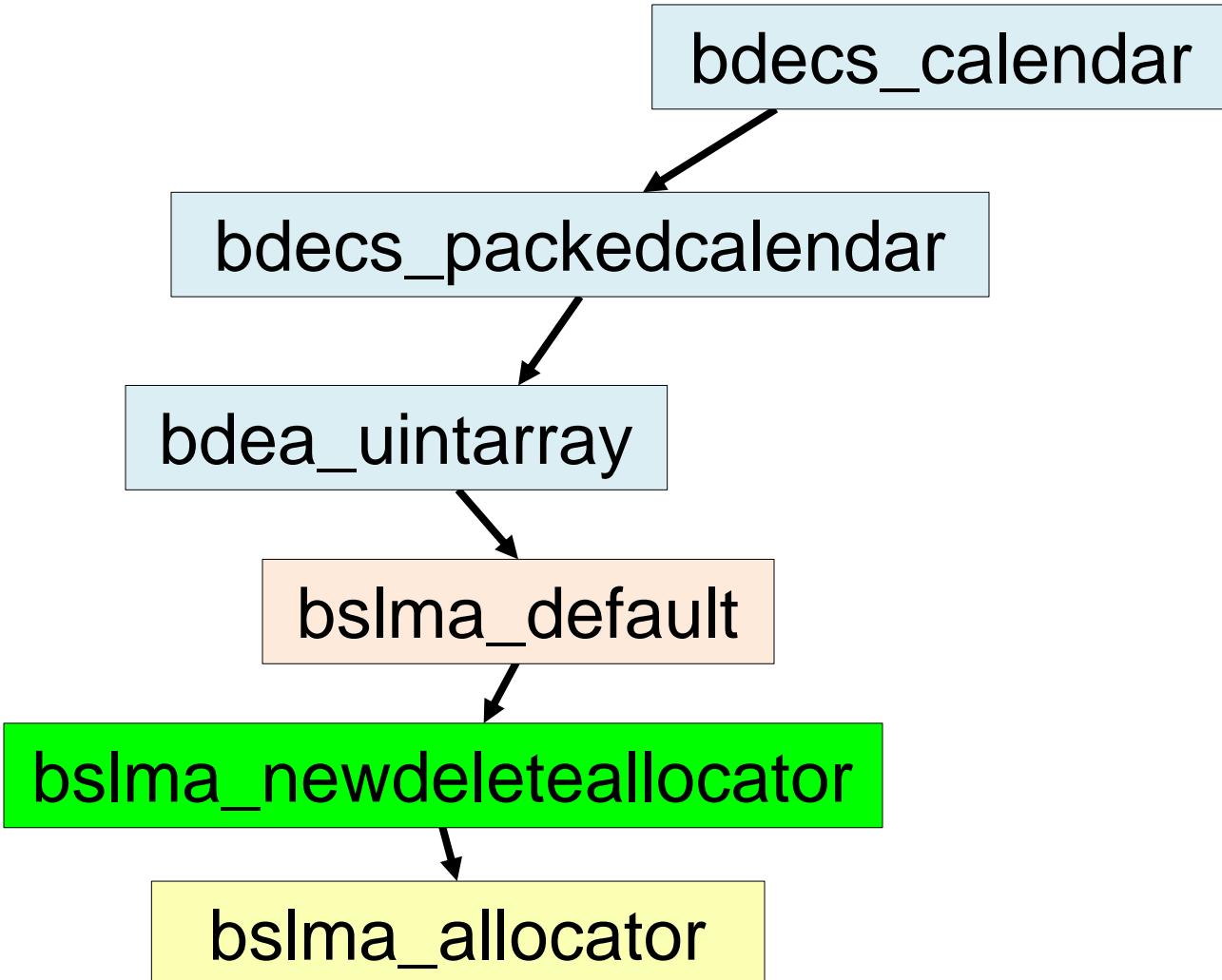
4. Present-Day, Real-World Design Examples

Implementing `bdecs_calendar`



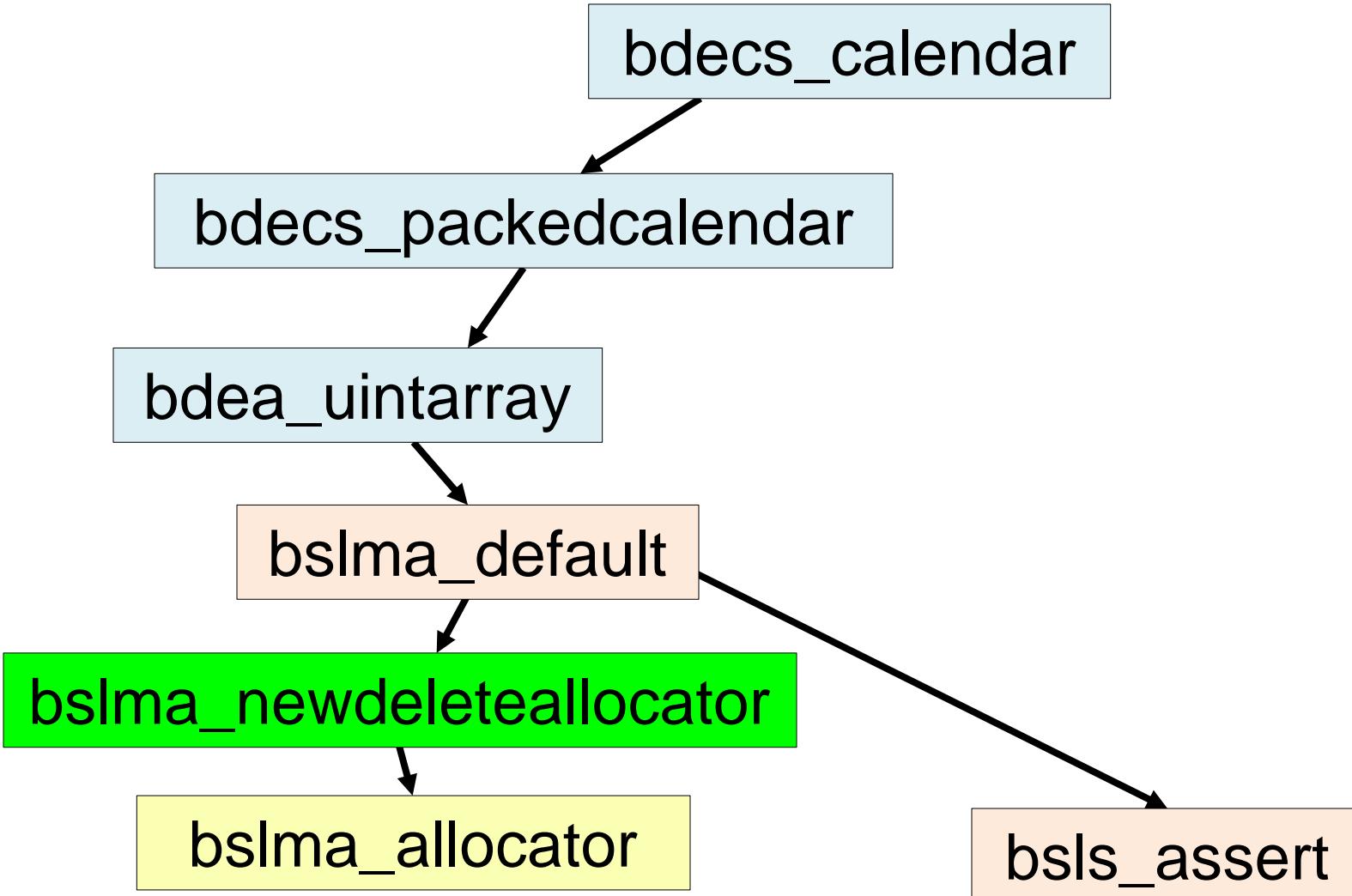
4. Present-Day, Real-World Design Examples

Implementing `bdecs_calendar`



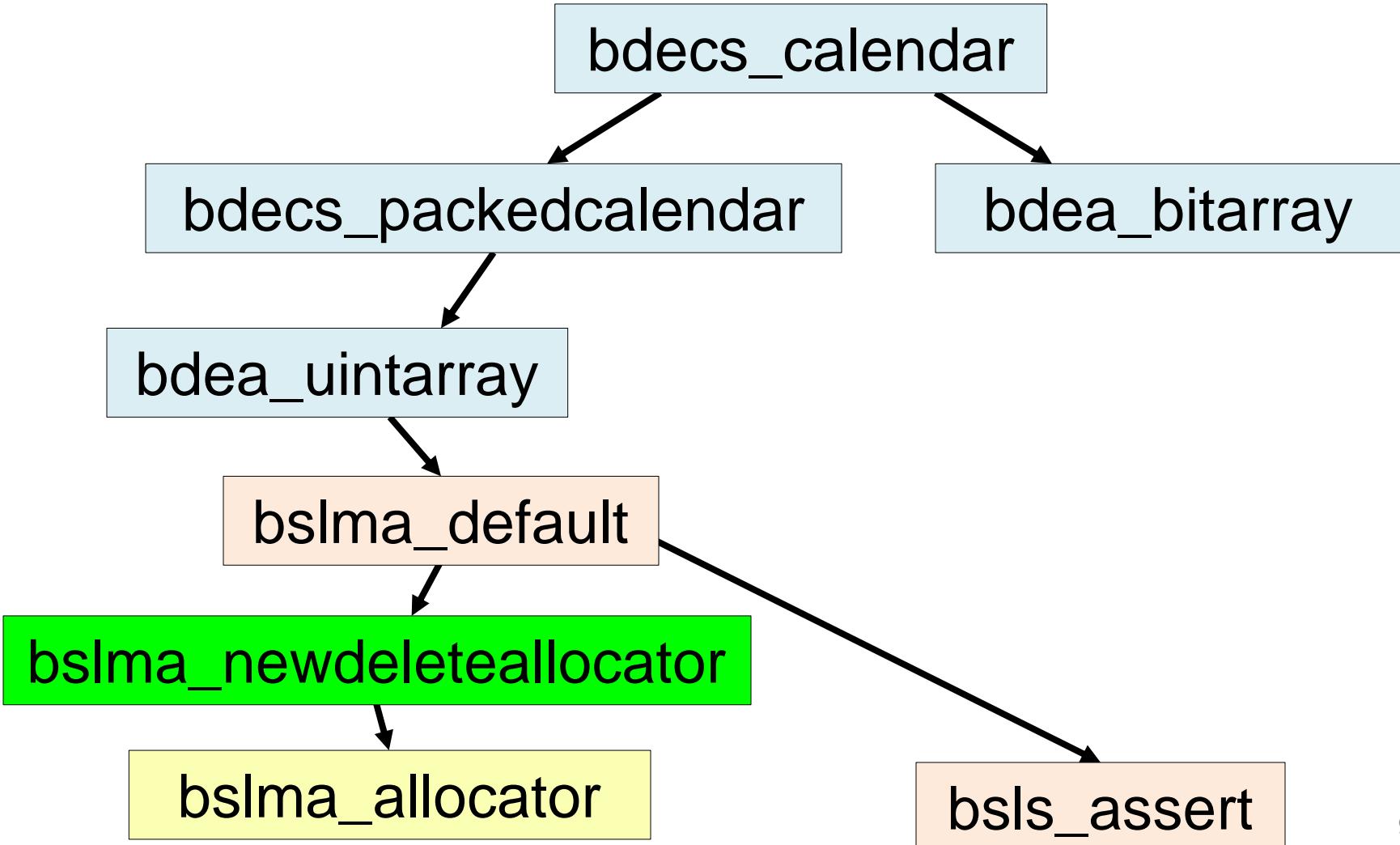
4. Present-Day, Real-World Design Examples

Implementing `bdecs_calendar`



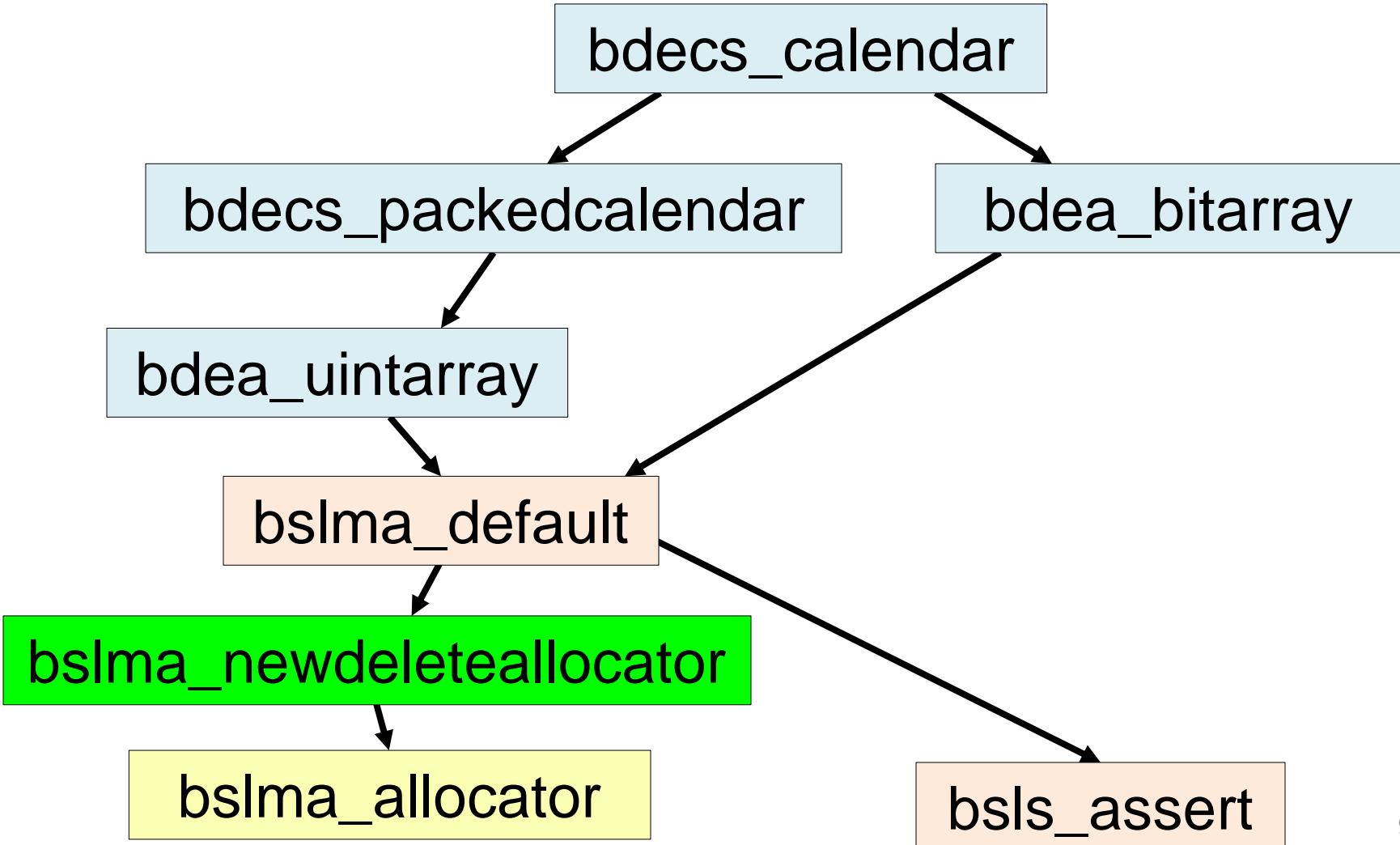
4. Present-Day, Real-World Design Examples

Implementing `bdecs_calendar`



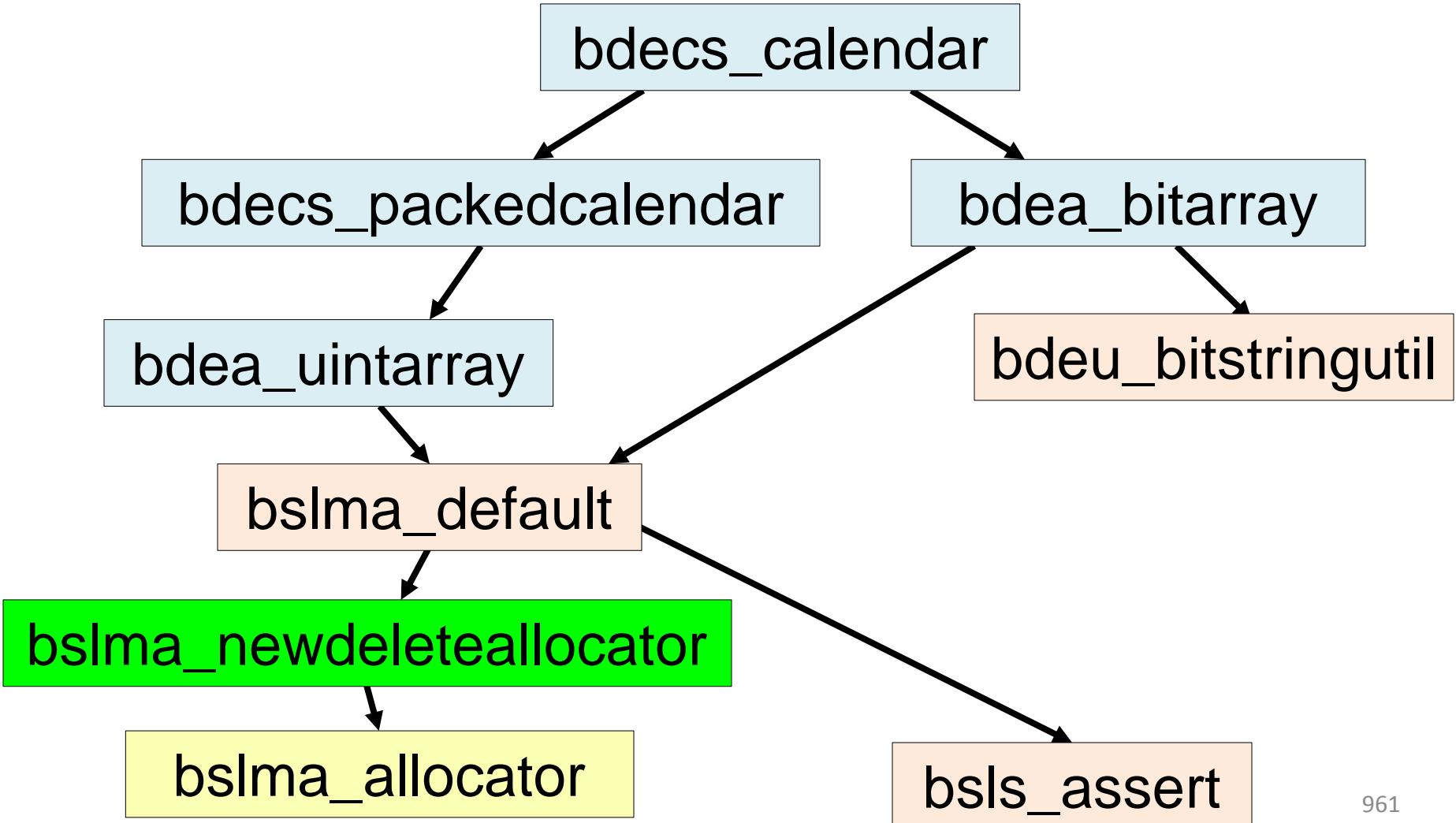
4. Present-Day, Real-World Design Examples

Implementing `bdecs_calendar`



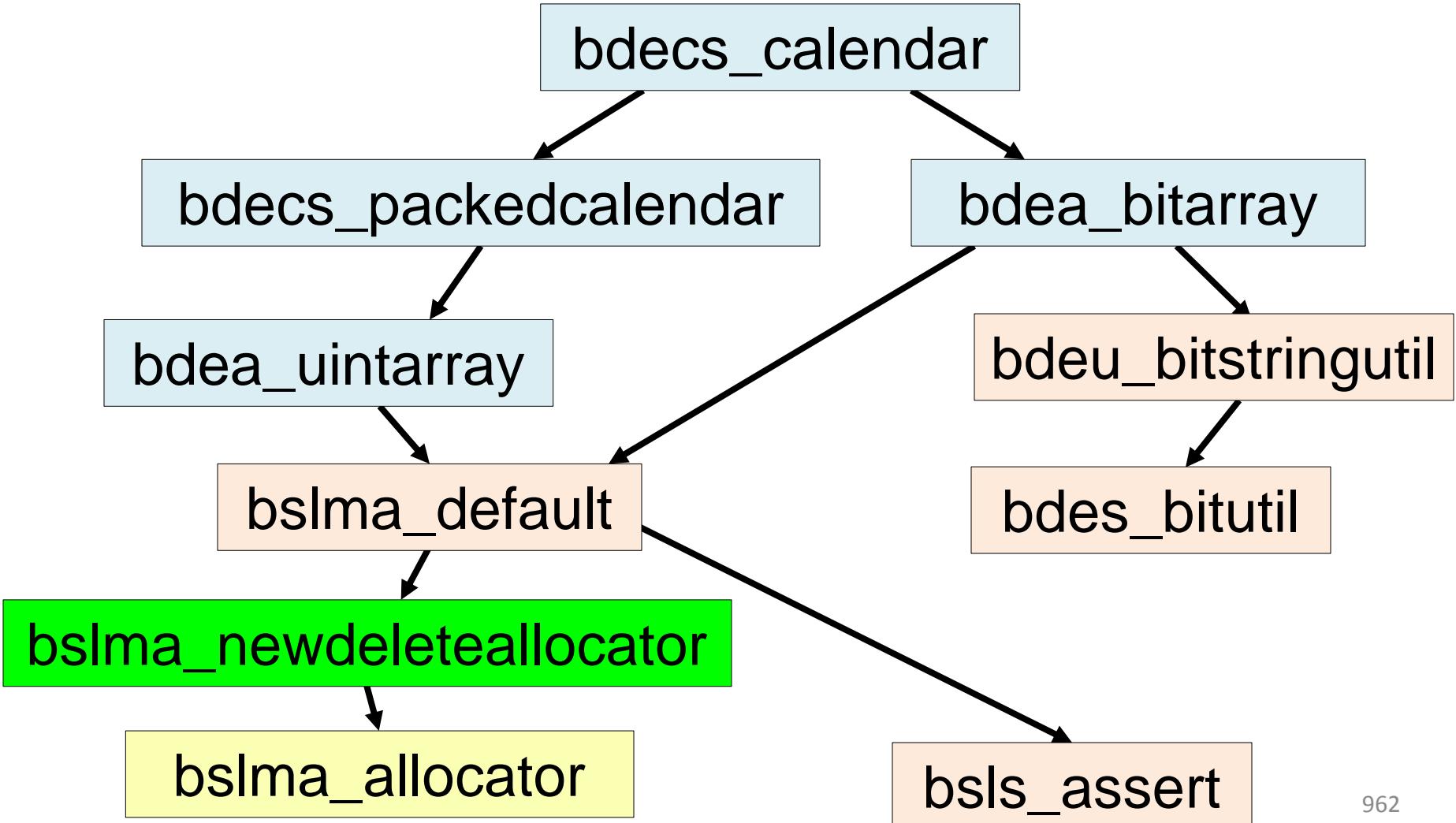
4. Present-Day, Real-World Design Examples

Implementing `bdecs_calendar`



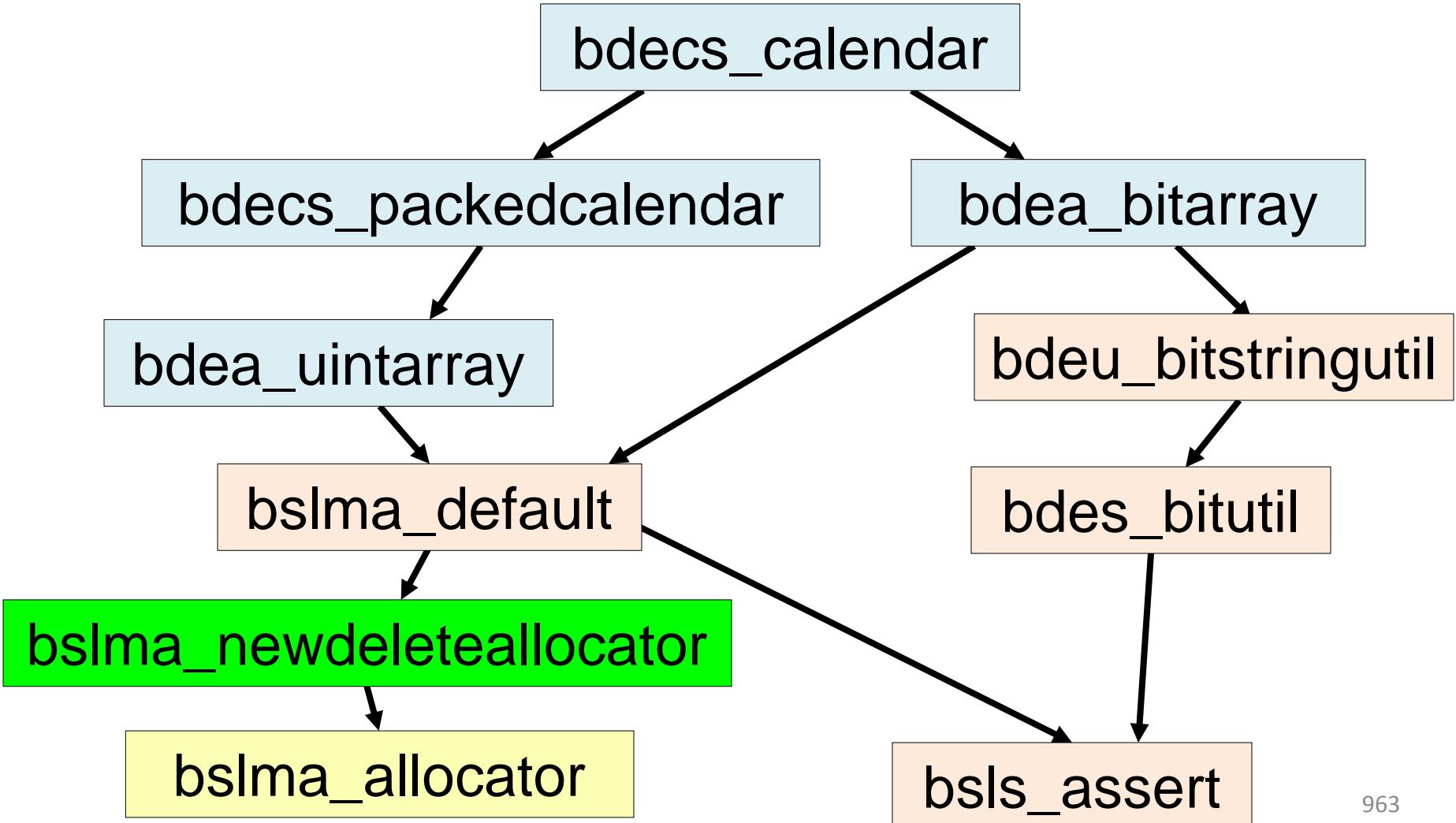
4. Present-Day, Real-World Design Examples

Implementing bdecs_calendar



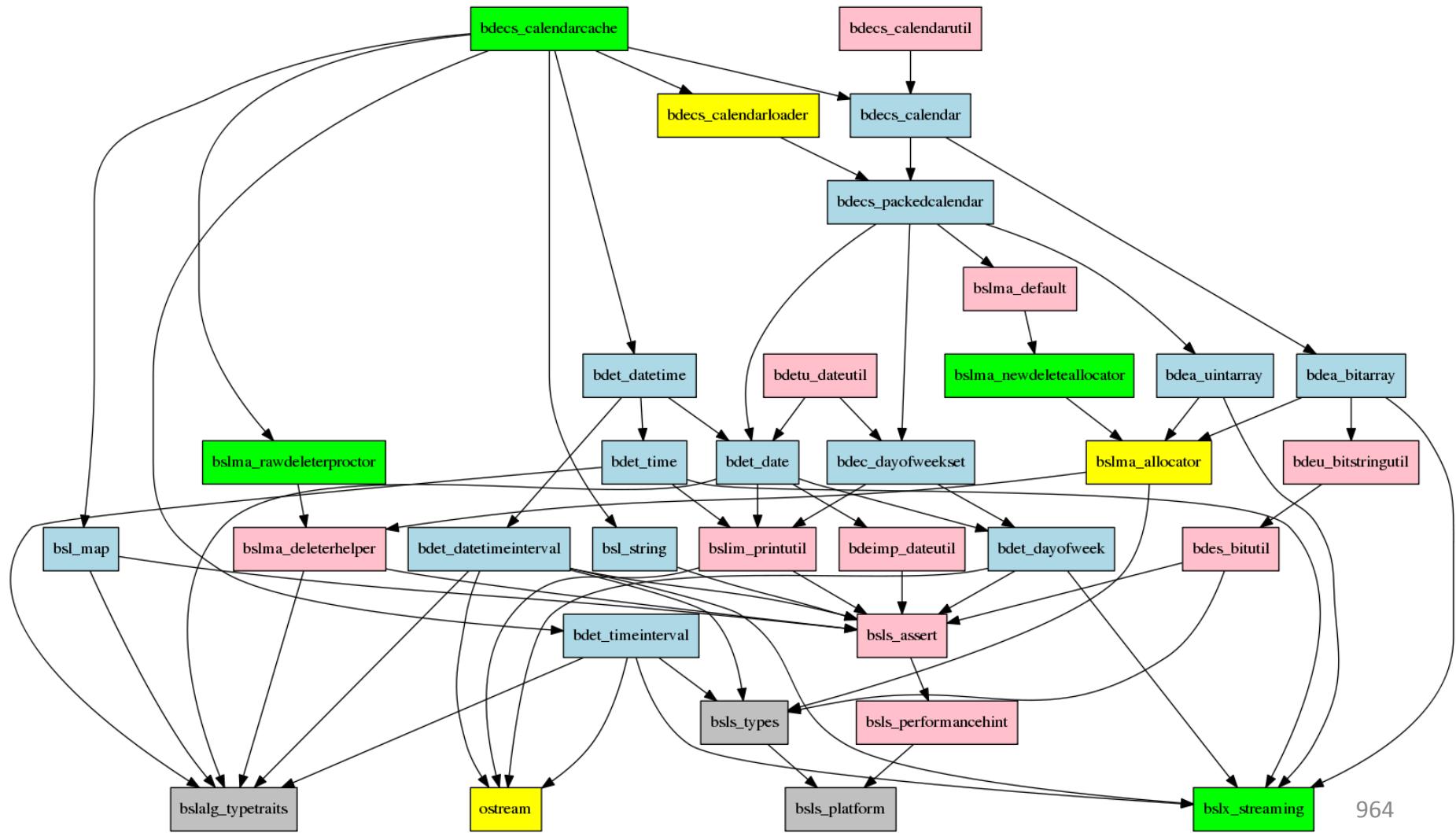
4. Present-Day, Real-World Design Examples

Implementing bdecs_calendar



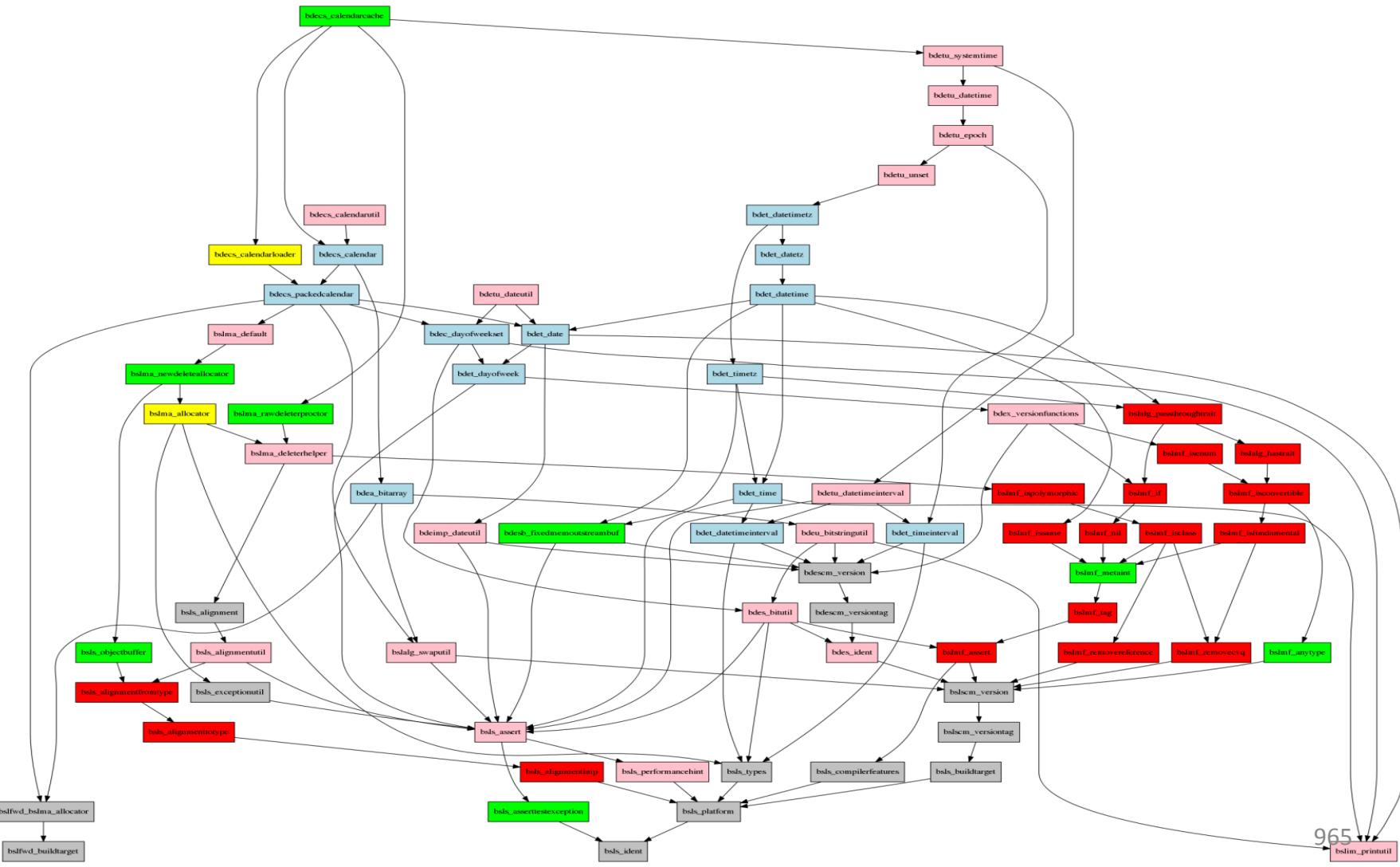
4. Present-Day, Real-World Design Examples

Hierarchically Reusable Implementation



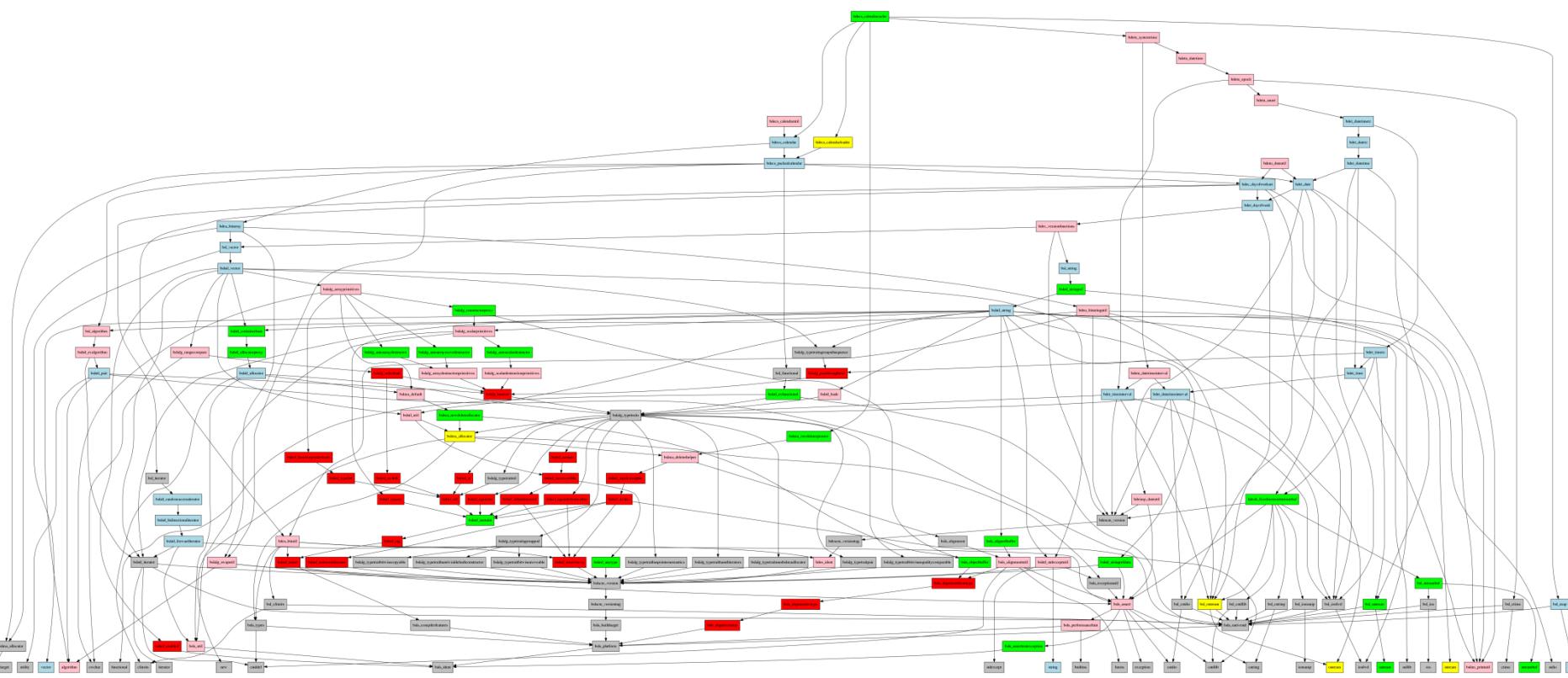
4. Present-Day, Real-World Design Examples

Hierarchically Reusable Implementation



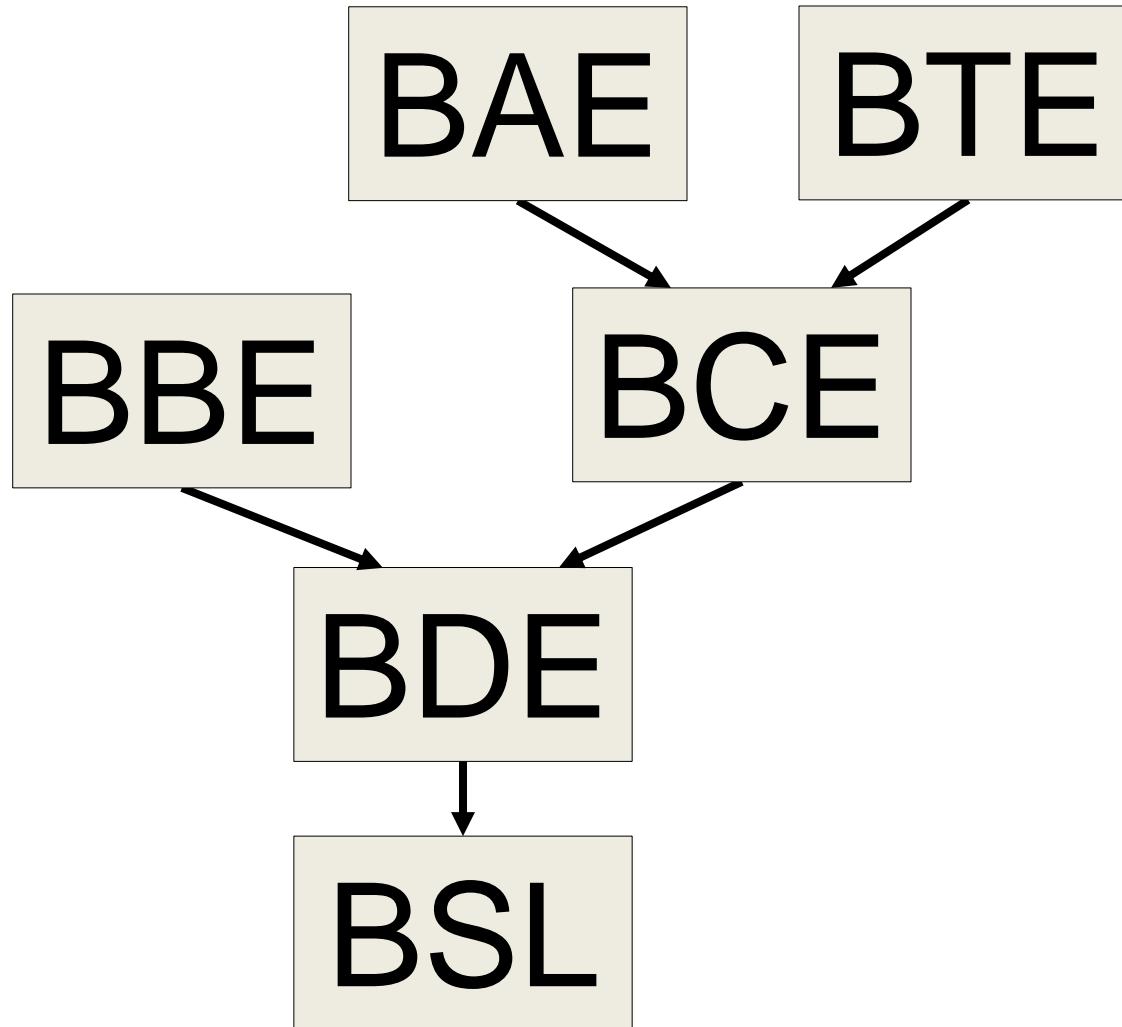
4. Present-Day, Real-World Design Examples

Hierarchically Reusable Implementation



4. Present-Day, Real-World Design Examples

Foundation “Package-Group” Libraries



4. Present-Day, Real-World Design Examples

End of Section

Questions?

Outline

1. Review of Elementary Physical Design
Components, Modularity, Physical Dependencies
2. Survey of Advanced *Levelization* Techniques
Avoiding Cyclic, Excessive, or Inappropriate dependencies
3. Total and Partial *Insulation* Techniques
Avoiding Unnecessary Compile-Time Coupling
4. Present-Day, Real-World Design Examples
Prioritizing Physical Design to Improve Software Quality

Conclusion

1. Review of Elementary Physical Design Components, Modularity, Physical Dependencies

Conclusion

1. Review of Elementary Physical Design
Components, Modularity, Physical Dependencies
 - A *Component* is the fundamental unit of both *logical* and *physical* software design.

Conclusion

1. Review of Elementary Physical Design

Components, Modularity, Physical Dependencies

- A *Component* is the fundamental unit of both *logical* and *physical* software design.
- No cyclic dependencies/*long-distance* friendships.

Conclusion

1. Review of Elementary Physical Design

Components, Modularity, Physical Dependencies

- A *Component* is the fundamental unit of both *logical* and *physical* software design.
- No cyclic dependencies/*long-distance* friendships.
- Colocate logical constructs only with good reason:
i.e., friendship; cycles; parts-of-whole; flea-on-elephant.

Conclusion

1. Review of Elementary Physical Design

Components, Modularity, Physical Dependencies

- A *Component* is the fundamental unit of both *logical* and *physical* software design.
- No cyclic dependencies/*long-distance* friendships.
- Colocate logical constructs only with good reason:
i.e., friendship; cycles; parts-of-whole; flea-on-elephant.
- Put a `#include` in a header only with good reason:
i.e., *Is-A*, *Has-A*, inline, enum, `typedef` -to-template.

Conclusion

The End