Onions and Spaghetti

Programming Lessons Learnt the Hard Way

Sara Falamaki
CSIRO Networking Technologies Lab

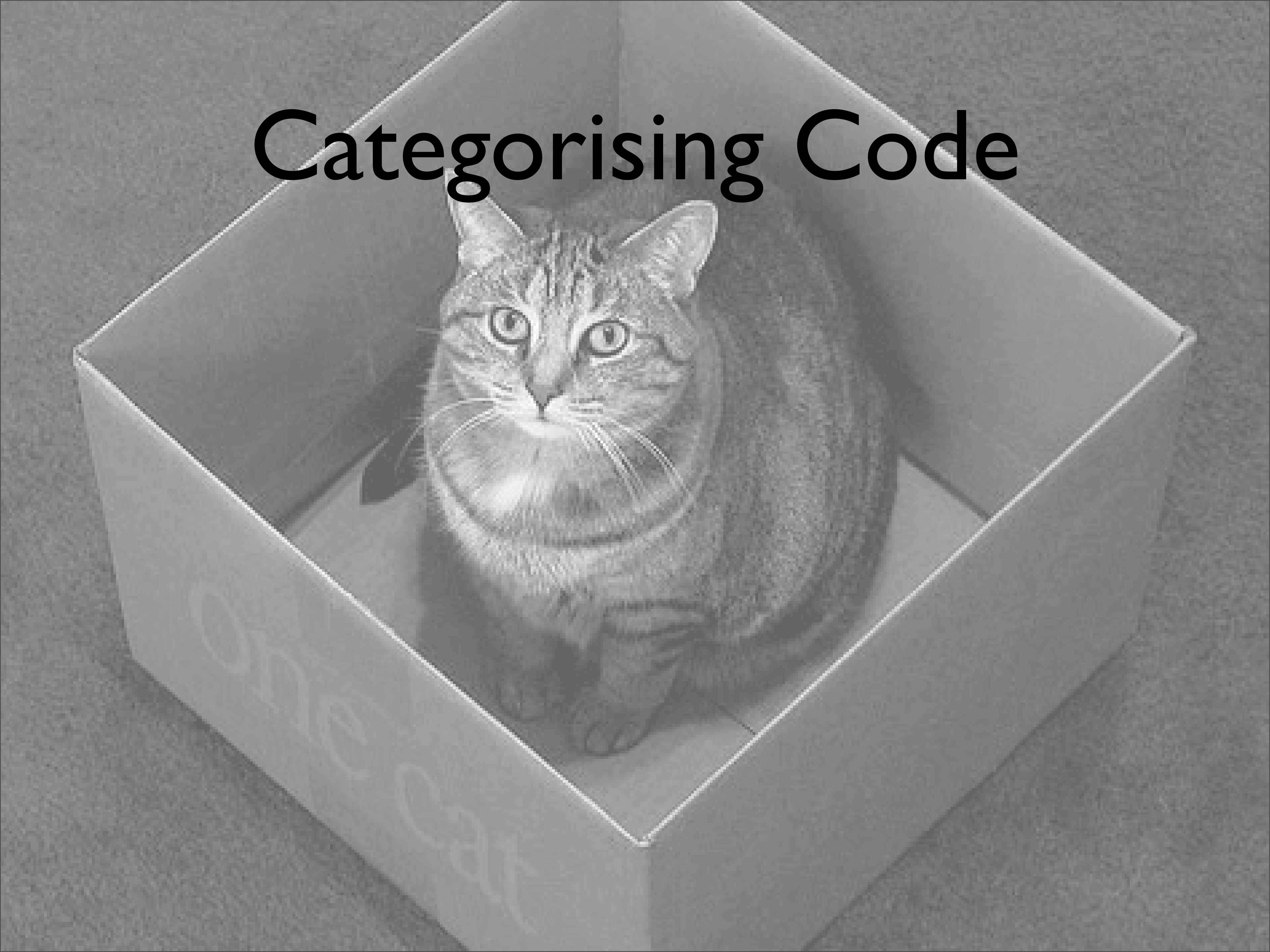
MSC Malaysia Open Source Developers Conference 2009

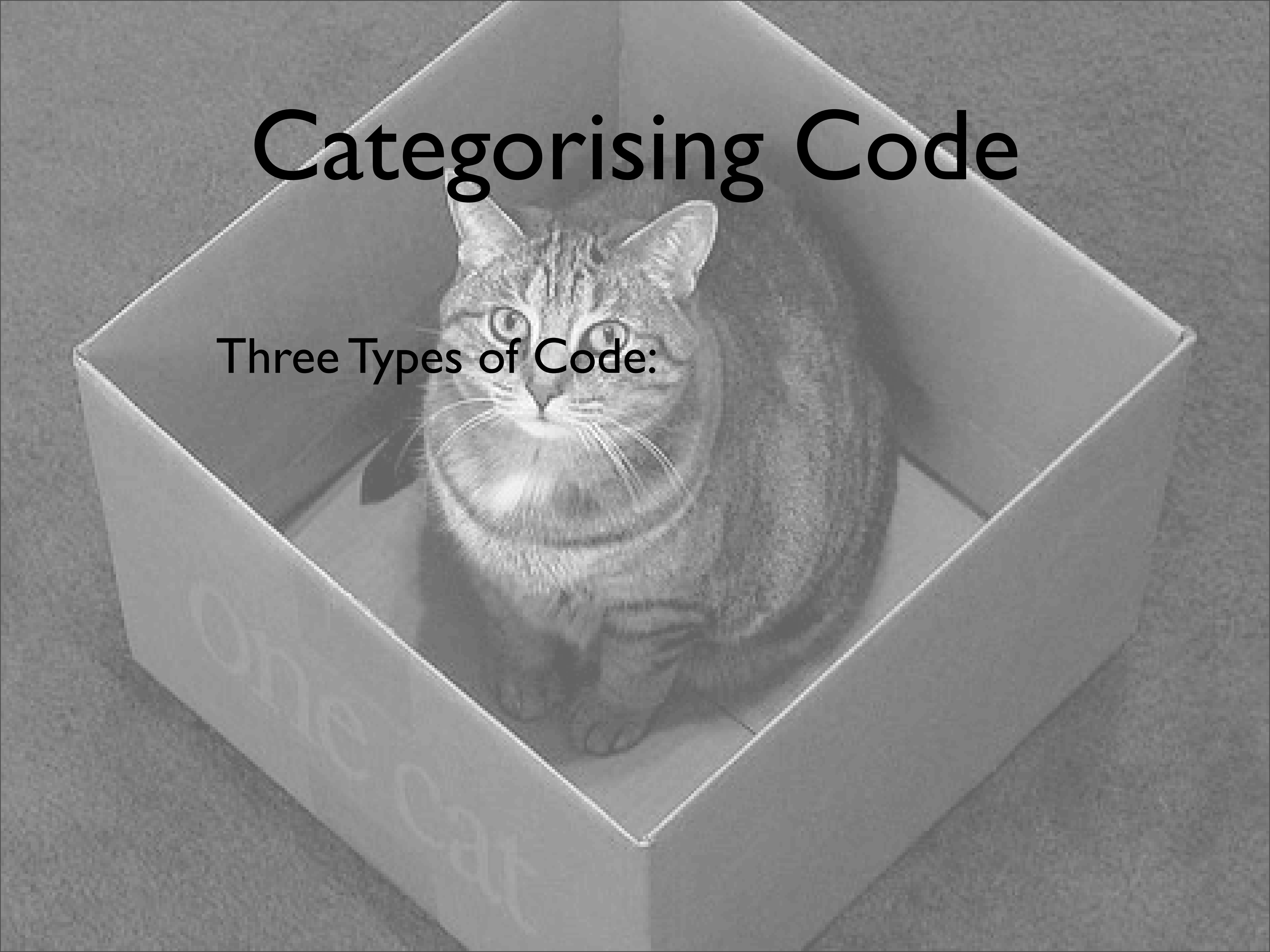
Onions and Spaghetti

Onions and Spaghetti

- Categorising Code
- Handling Errors and Exceptions
- Global Variables Are Considered Evil
- Unspaghettifying Your Objects
- Controlling Data Flow
- Encapsulation
- Threads are Hard!

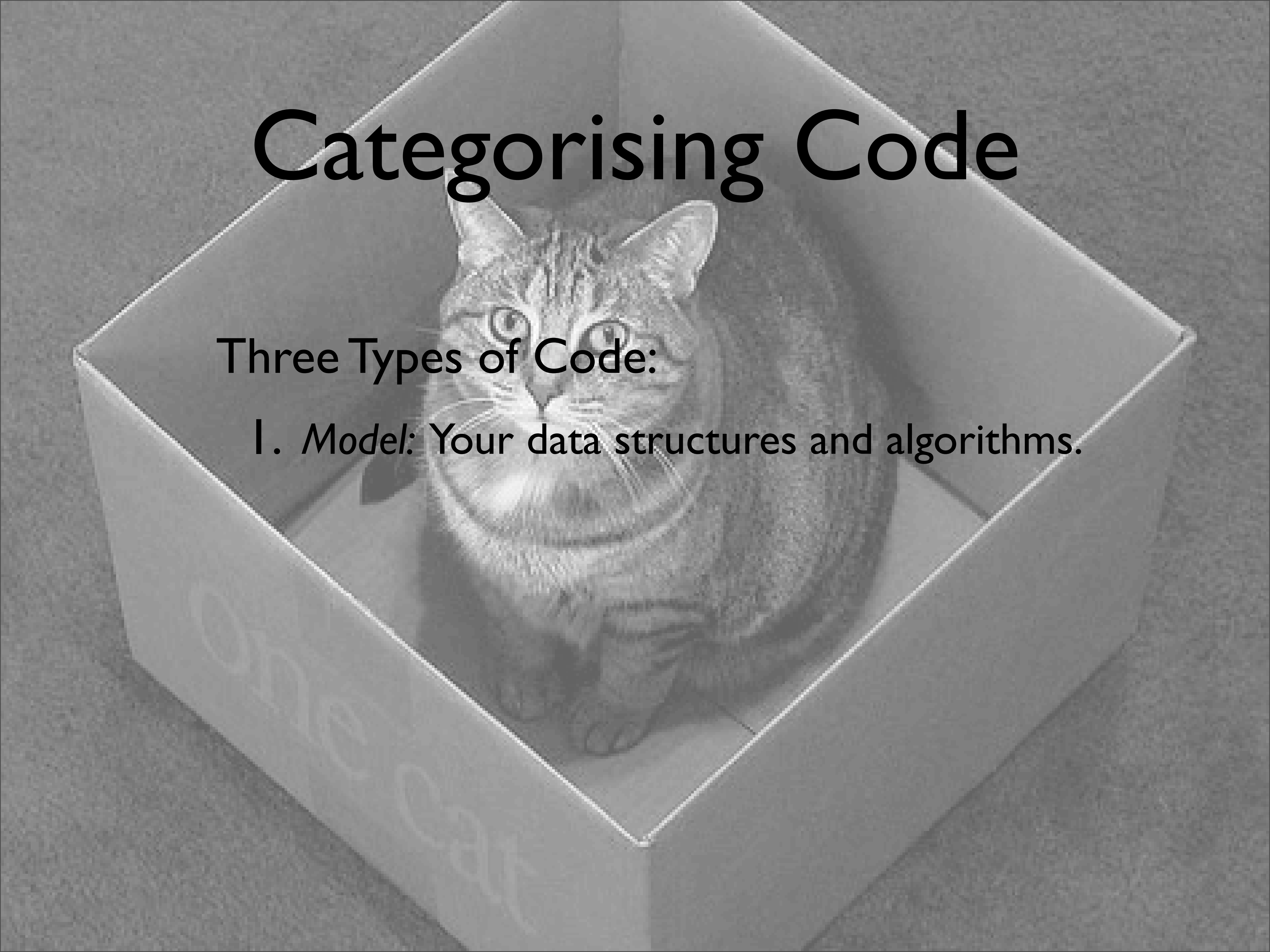
Categorising Code





Categorising Code

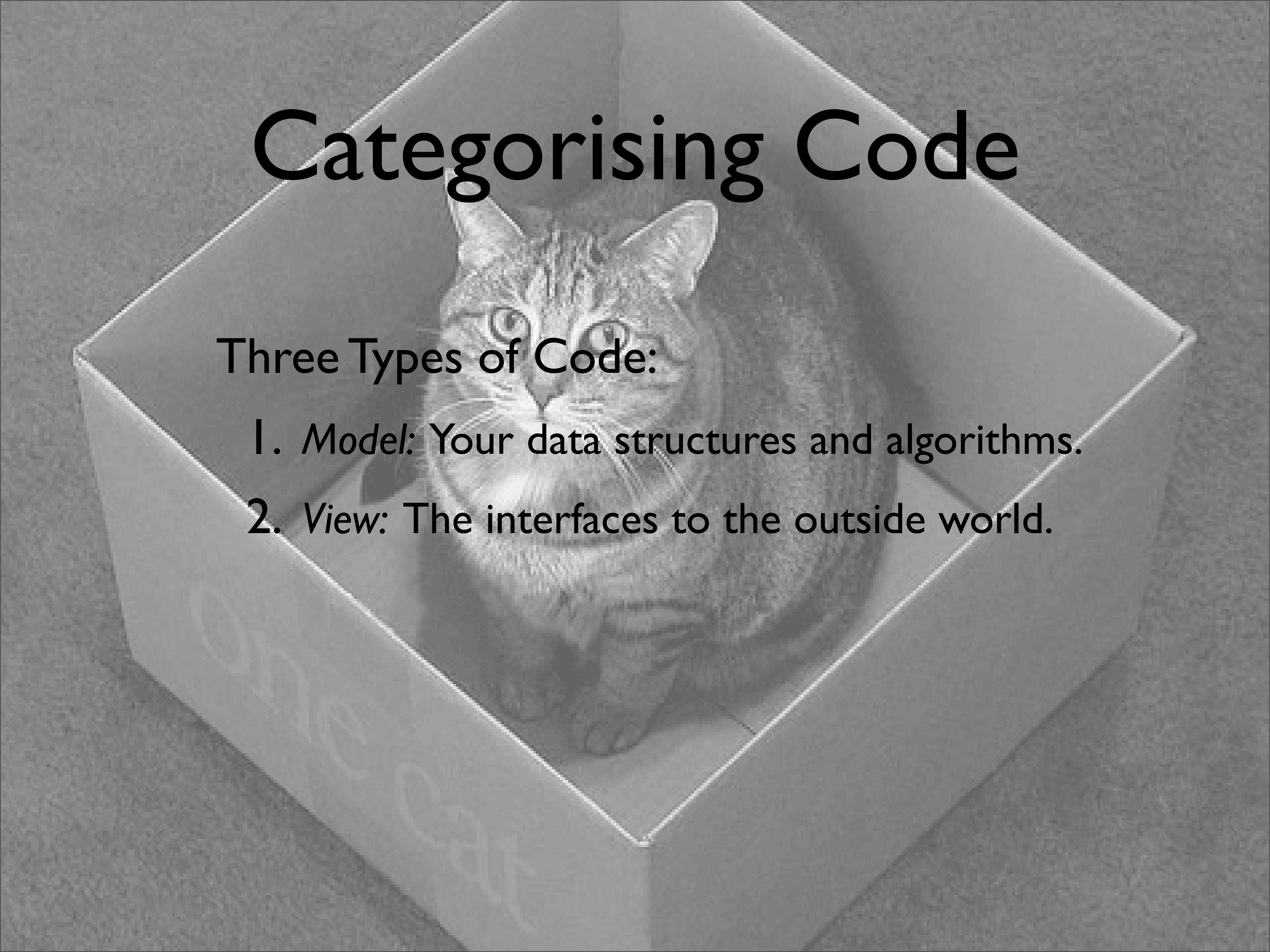
Three Types of Code:



Categorising Code

Three Types of Code:

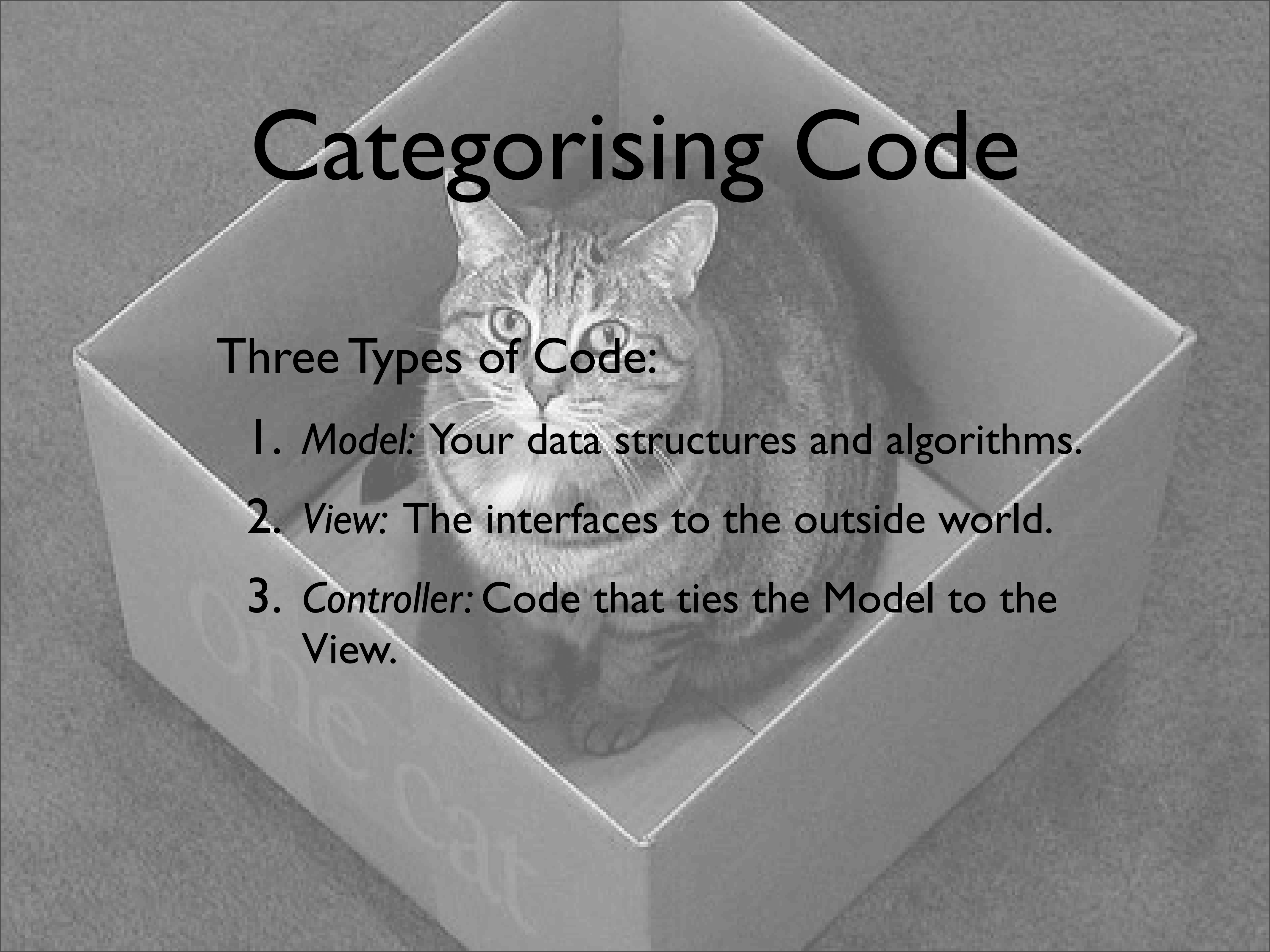
1. *Model:* Your data structures and algorithms.



Categorising Code

Three Types of Code:

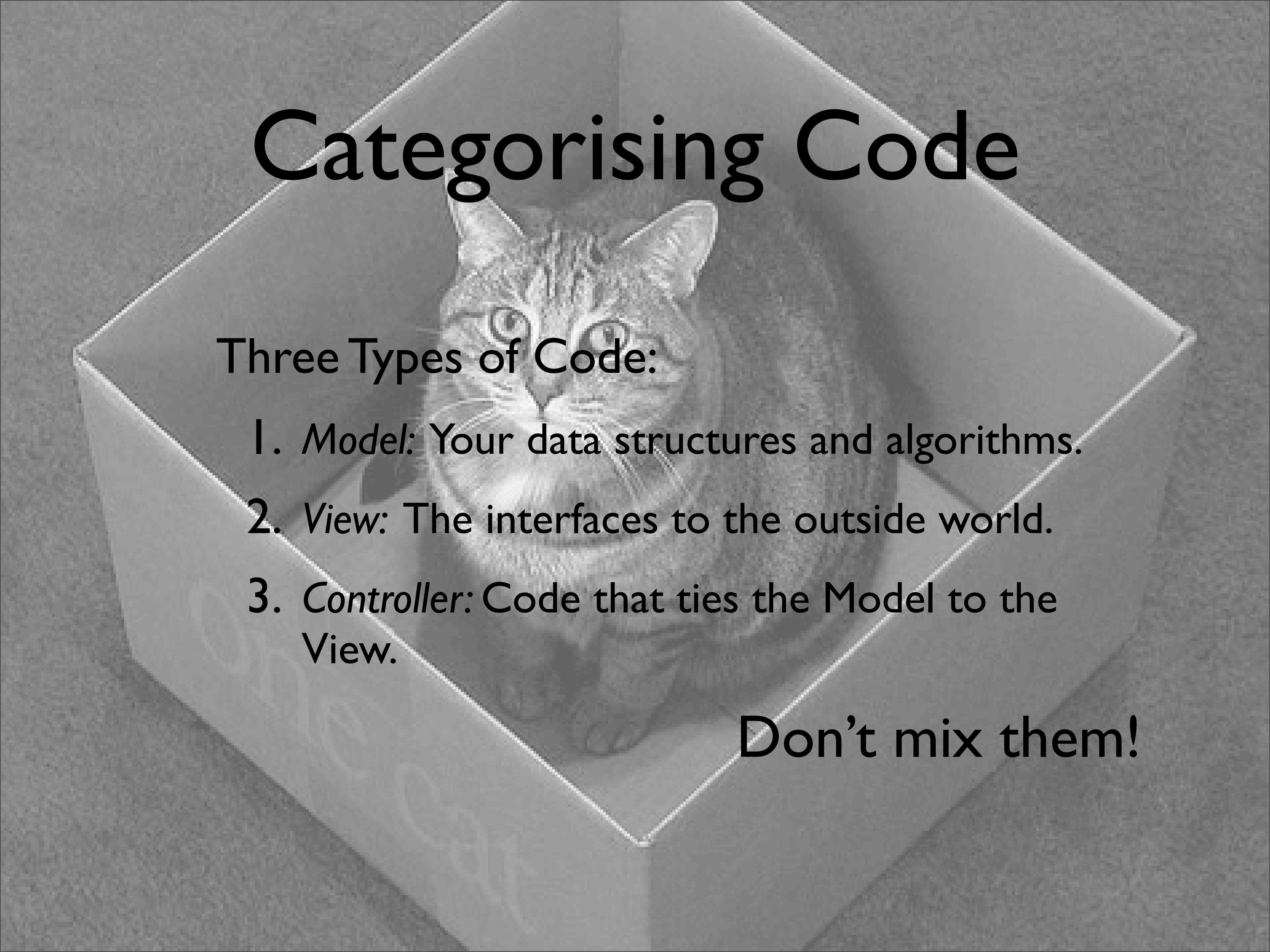
1. *Model:* Your data structures and algorithms.
2. *View:* The interfaces to the outside world.



Categorising Code

Three Types of Code:

1. *Model:* Your data structures and algorithms.
2. *View:* The interfaces to the outside world.
3. *Controller:* Code that ties the Model to the View.



Categorising Code

Three Types of Code:

1. *Model:* Your data structures and algorithms.
2. *View:* The interfaces to the outside world.
3. *Controller:* Code that ties the Model to the View.

Don't mix them!

Handling Exceptions

Handling Exceptions

Handling Exceptions

- Almost universally handled badly

Handling Exceptions

- Almost universally handled badly
- Only catch it if you can do something about it

Handling Exceptions

- Almost universally handled badly
- Only catch it if you can do something about it
- If you can't deal with it, let it bubble up and fail. **Fail-Fast** code is good!

Handling Exceptions

Handling Exceptions

- Don’t hot potato exceptions. Try to deal with each exception only once.

Handling Exceptions

- Don't hot potato exceptions. Try to deal with each exception only once.

Handling Exceptions

Handling Exceptions

- Exceptions can really complicate your program state when you catch them.

Handling Exceptions

- Exceptions can really complicate your program state when you catch them.
- If you intend to roll back state, make your try statements as small as possible.

Handling Exceptions

```
bool
Checker::isAvailable(){
    try{
        QueryResult result;
        checkQuery = "SELECT * FROM Devices WHERE available";
        _db.query(checkQuery, result);
        if(result.size()==0){
            rescheduleTimer(20);
            return false;
        }
        else{
            rescheduleTimer(100);
            propagateResults(result);
            return true;
        }
    }
    catch(Exception &e){
        cerr << "isAvailable: query failed!";
        return false;
    }
}
```

Handling Exceptions

```
bool
Checker::isAvailable(){
    QueryResult result;
    checkQuery = "SELECT * FROM Devices WHERE available";
    try{
        _db.query(checkQuery, result);
    }
    catch(DBTimeoutException &e){
        cerr << "isAvailable: query failed! " << e.what();
        rescheduleTimer(20);
        return false;
    }
    if(result.size()==0){
       rescheduleTimer(20);
       return false;
    }
    else{
       rescheduleTimer(100);
       propagateResults(result);
       return true;
    }

}
```

Global Variables are Considered Evil

Global Variables are Considered Evil

Global Variables are Considered Evil

- We all learn this in first year, but seem to forget it too frequently.

Global Variables are Considered Evil

- We all learn this in first year, but seem to forget it too frequently.
- Global variables masquerading as member variables are also Evil!

Why Are Global Variables Evil?

Why Are Global Variables Evil?

- They make tracking the program's state hard.

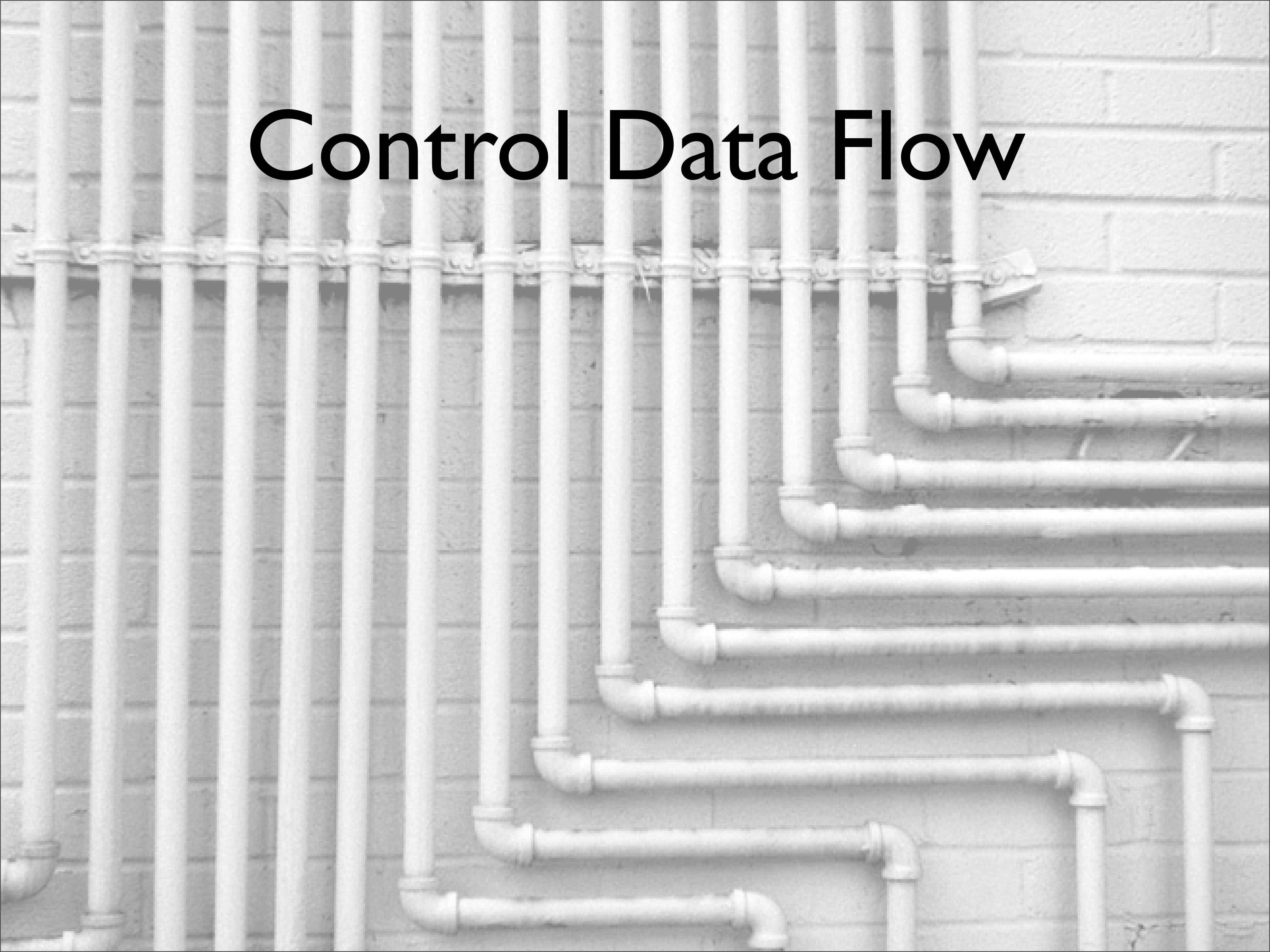
Why Are Global Variables Evil?

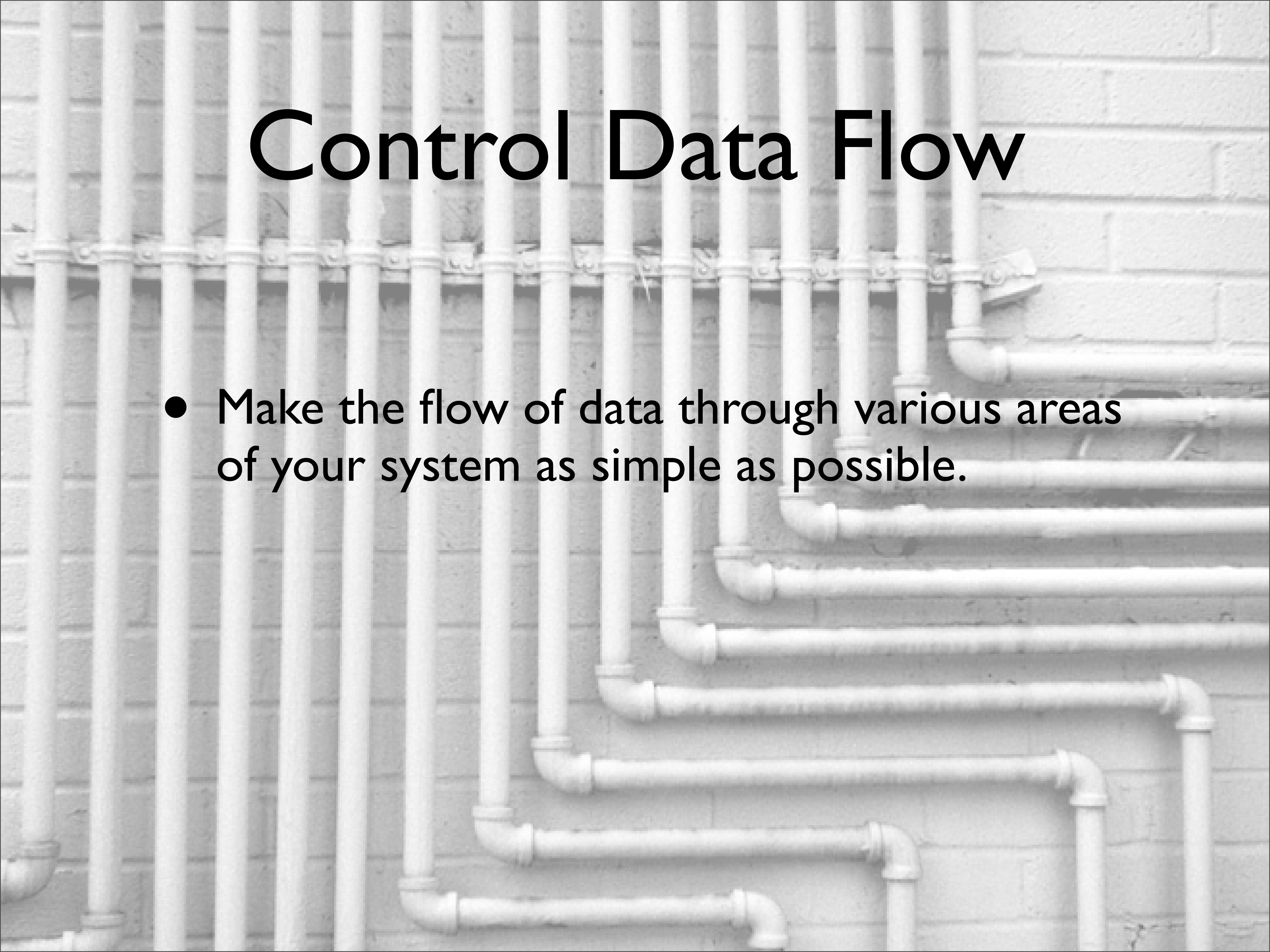
- They make tracking the program’s state hard.
- They make using multiple threads more dangerous and difficult.

Why Are Global Variables Evil?

- They make tracking the program's state hard.
- They make using multiple threads more dangerous and difficult.
- They make testing really really hard.

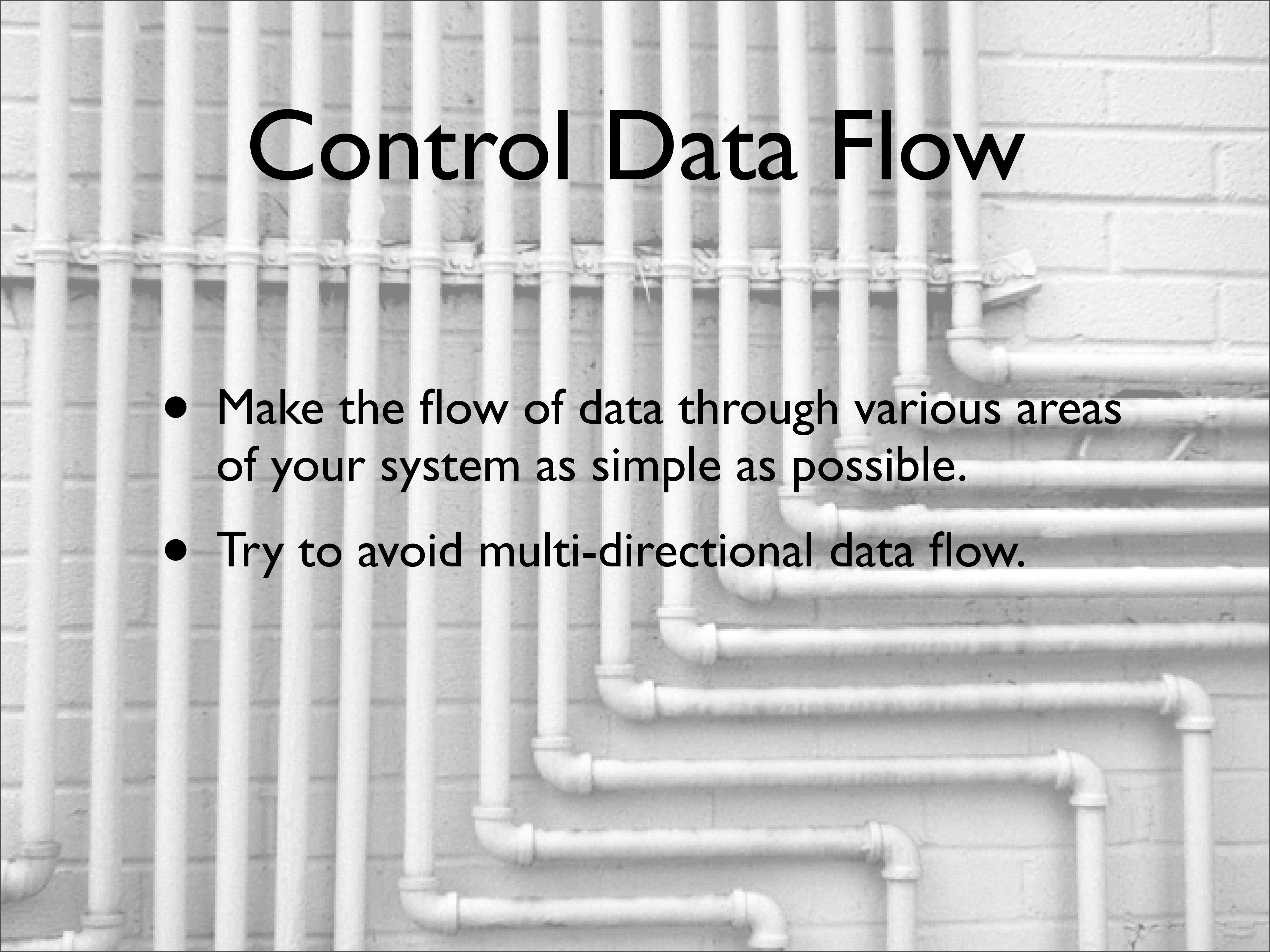
Control Data Flow





Control Data Flow

- Make the flow of data through various areas of your system as simple as possible.

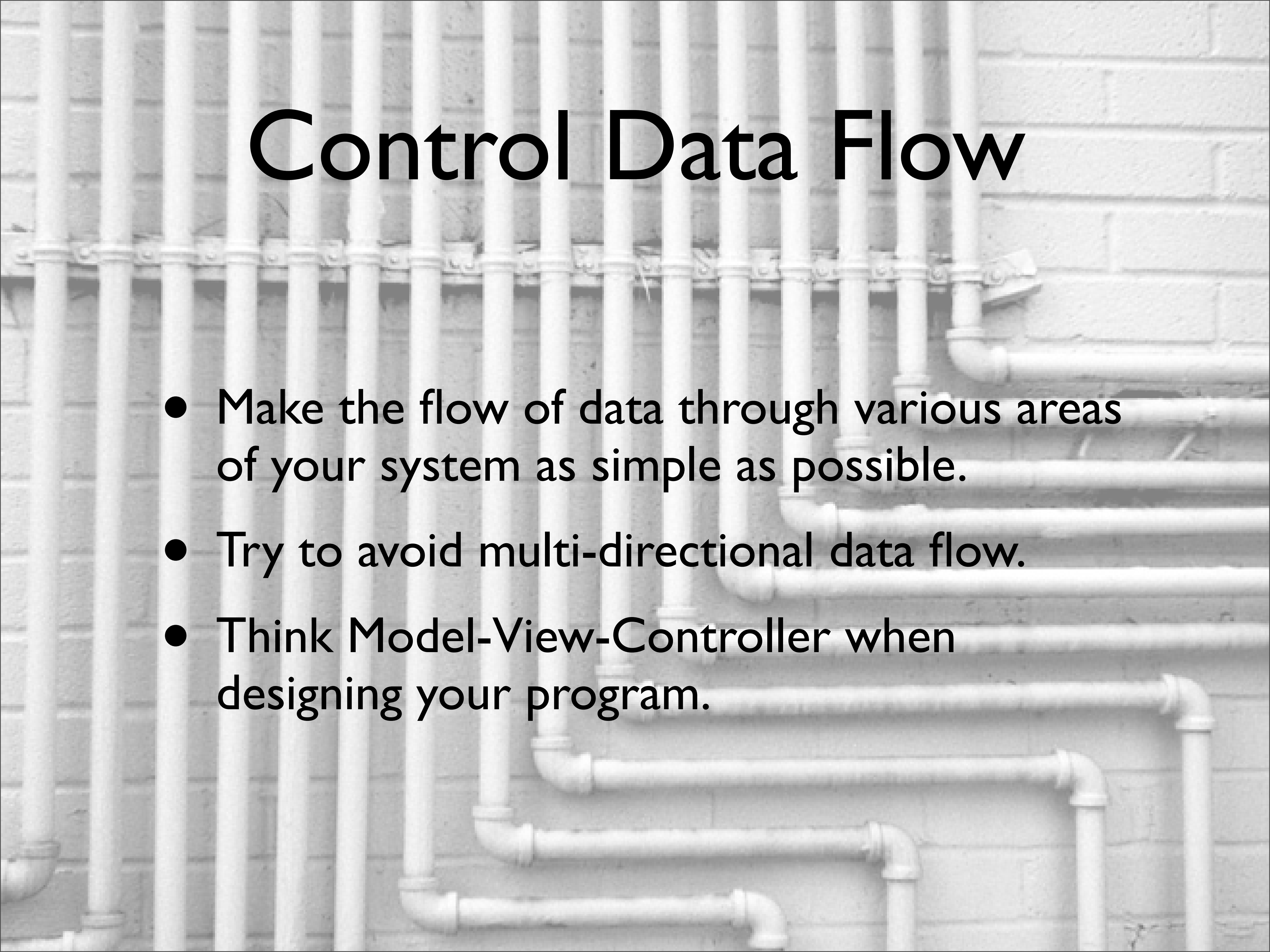


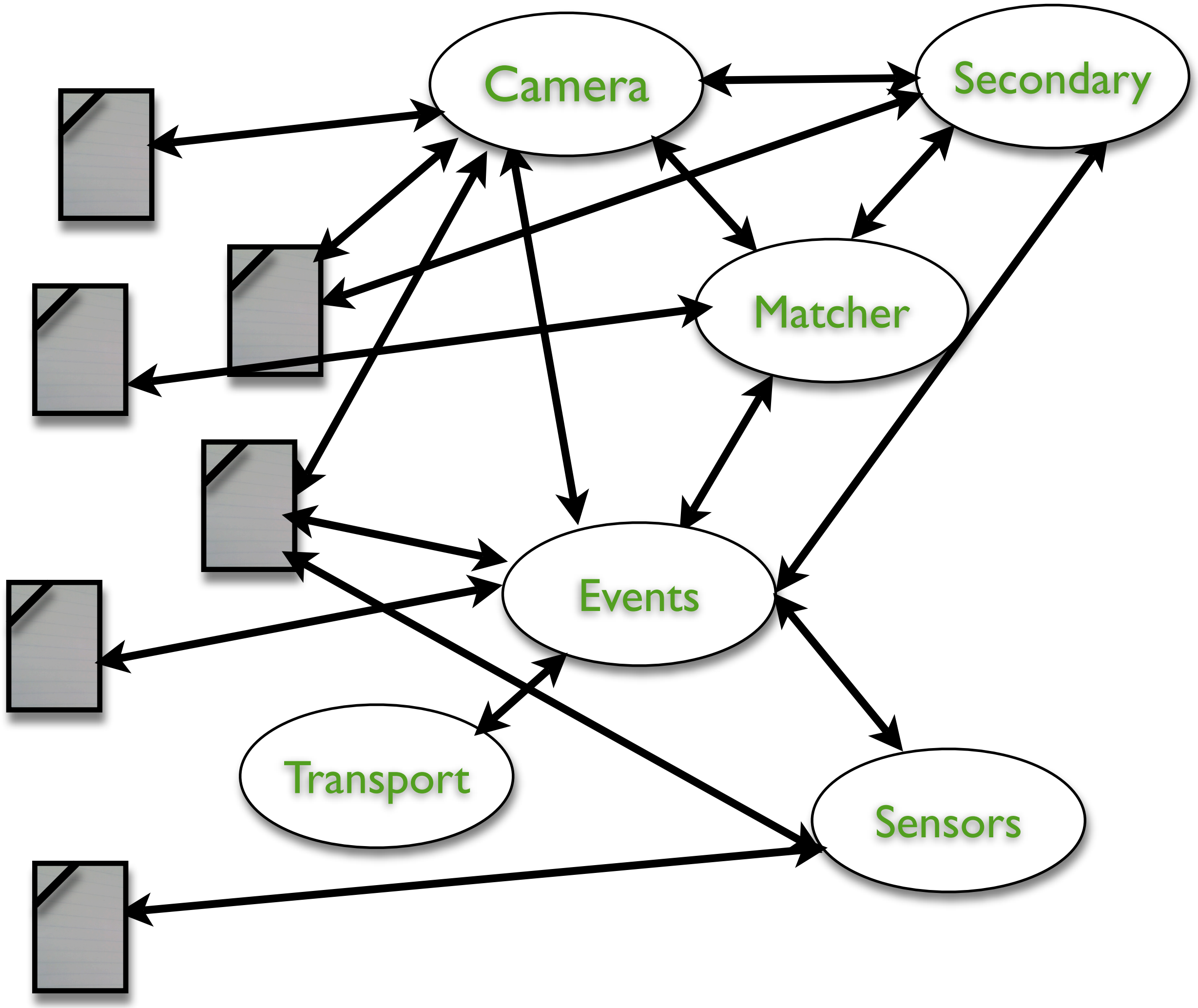
Control Data Flow

- Make the flow of data through various areas of your system as simple as possible.
- Try to avoid multi-directional data flow.

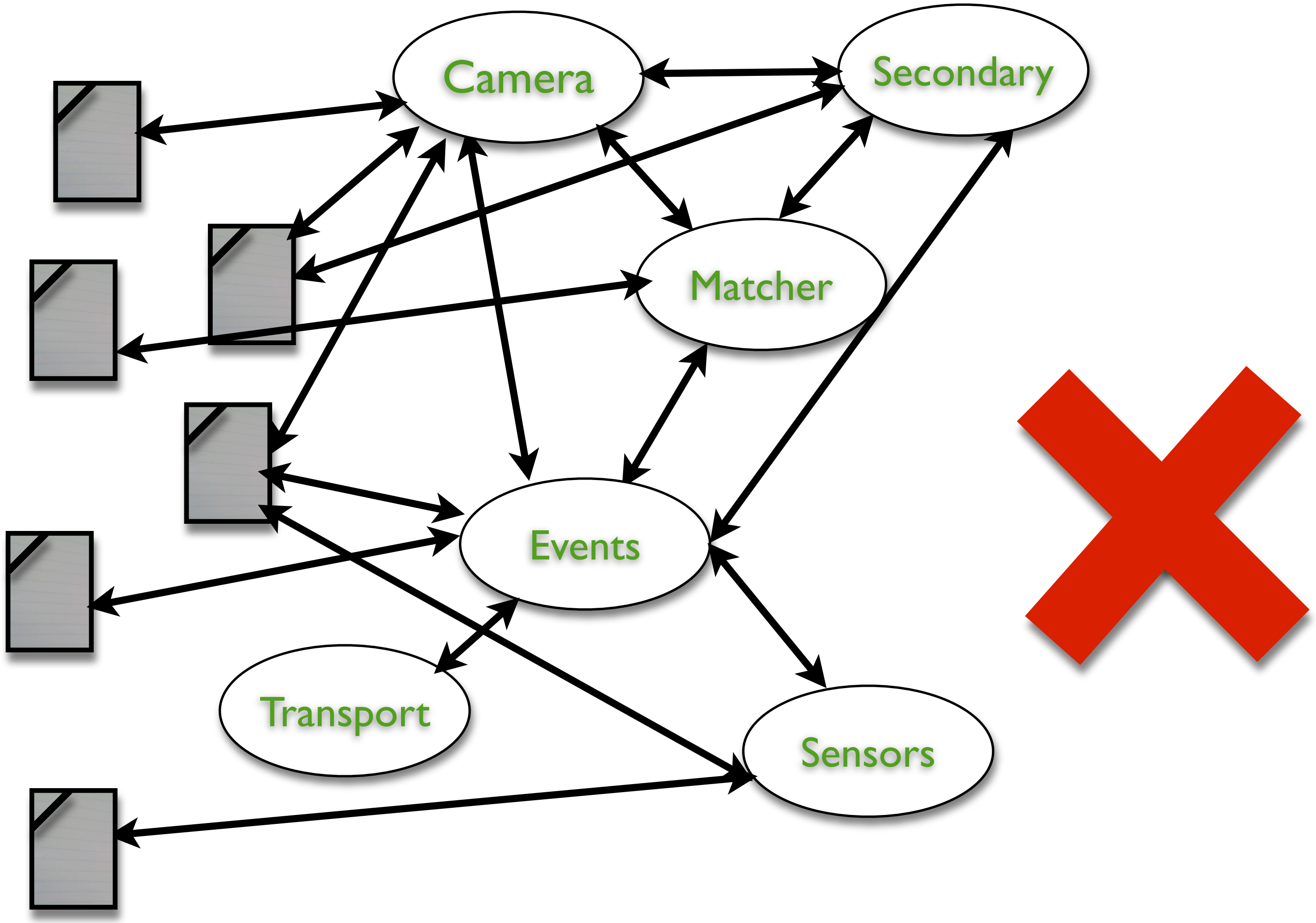
Control Data Flow

- Make the flow of data through various areas of your system as simple as possible.
- Try to avoid multi-directional data flow.
- Think Model-View-Controller when designing your program.

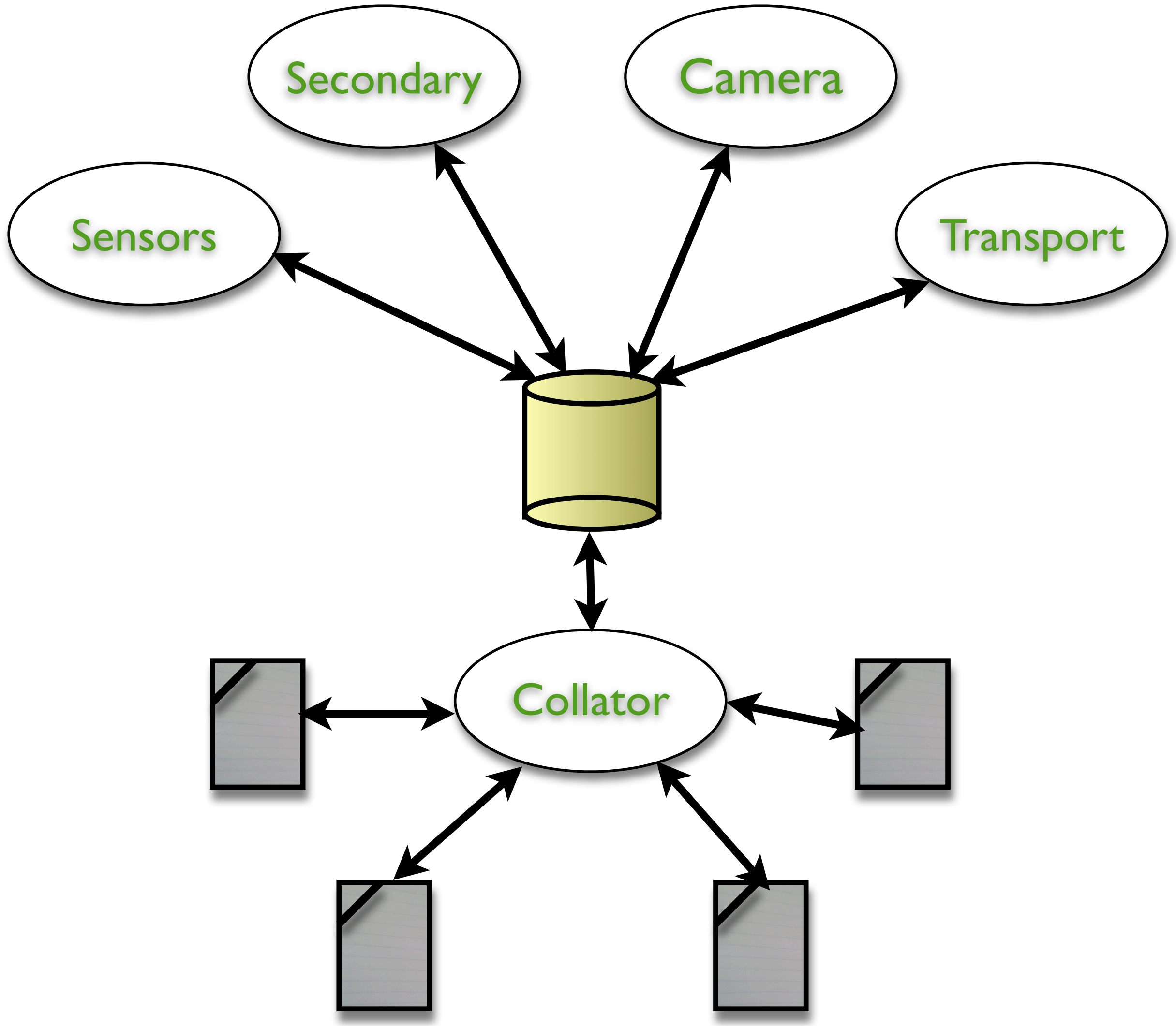




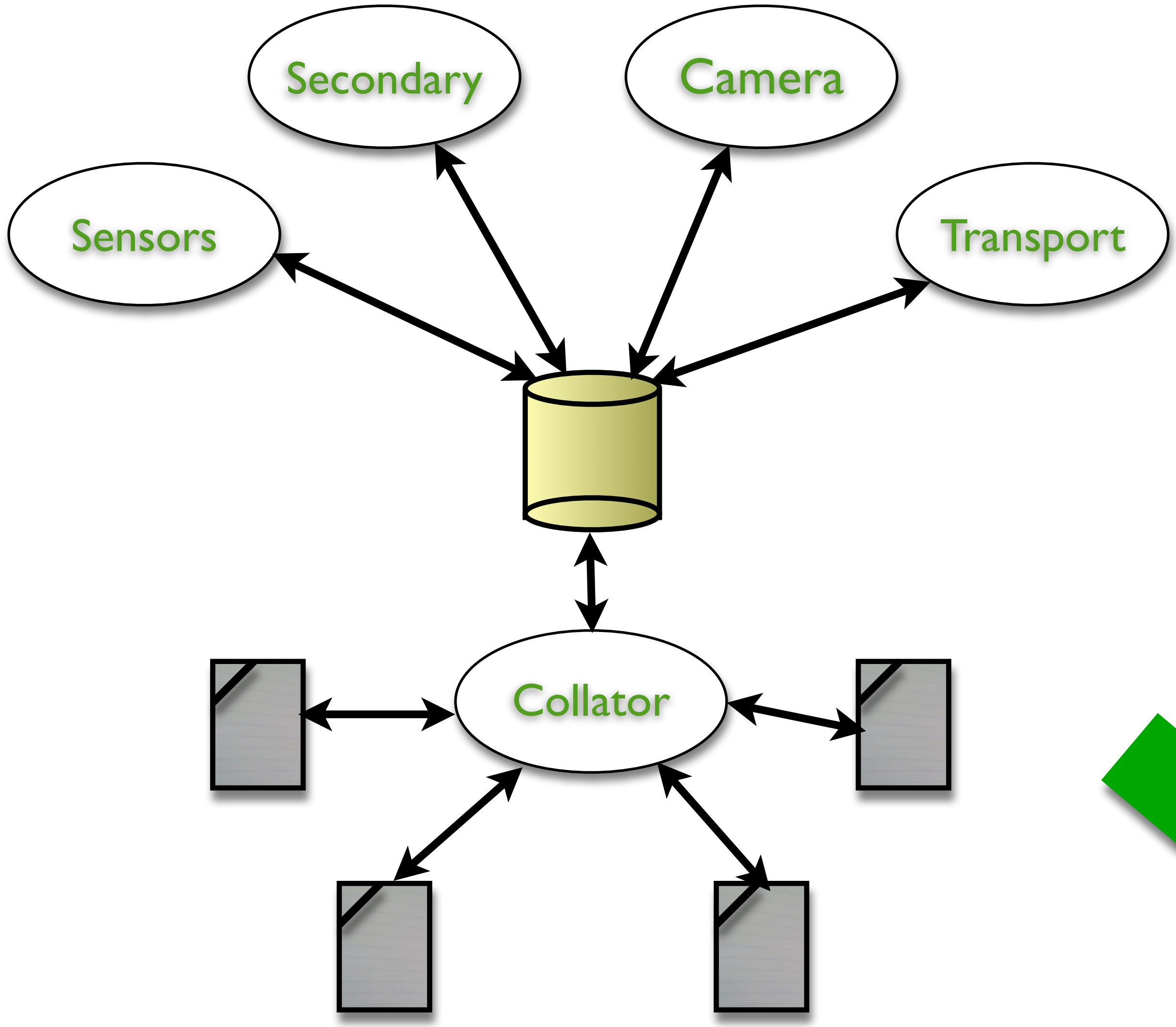
Original Design

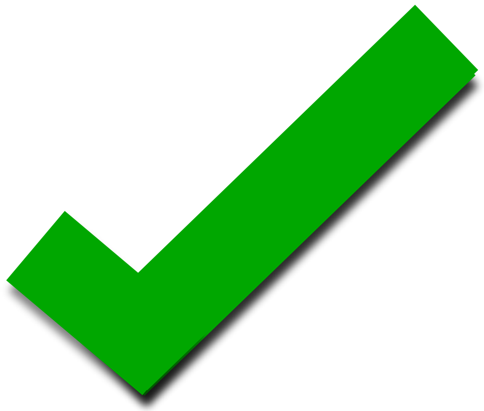


Original Design



New Design





New Design

Unspaghettifying Your Objects

Unspaghettifying Your Objects

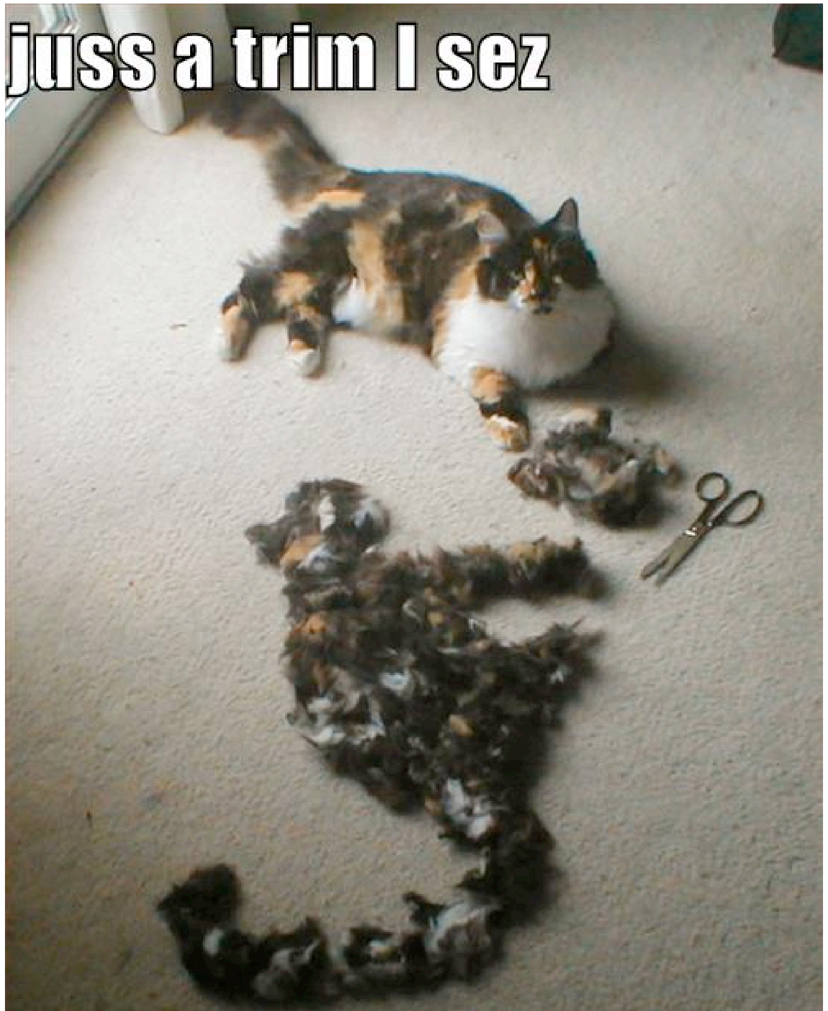
- Don’t use an object as a namespace and its members as global variables
- Don’t be afraid of having a library of static functions that do computation.

Unspaghettifying Your Objects

Keep your *object* graphs well trimmed

- Separate your logic from your object construction
- Avoid constructing *new* objects in your object - Use **Dependency Injection**

Unspaghettifying Your Objects



Unspaghettifying Your Objects

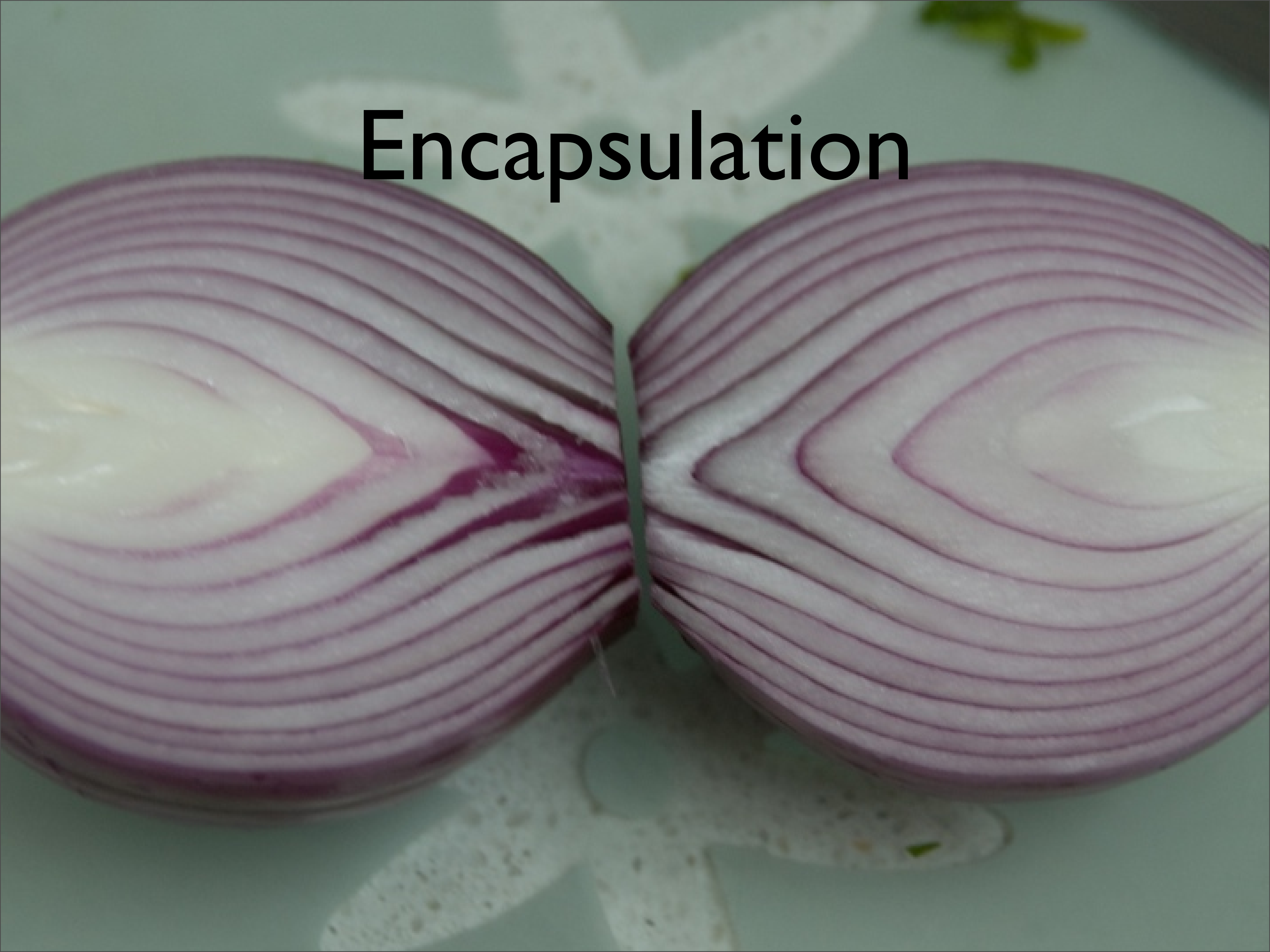
Unspaghettifying Your Objects

- Keep your *inheritance* graphs well trimmed - Don’t write Onion Code!

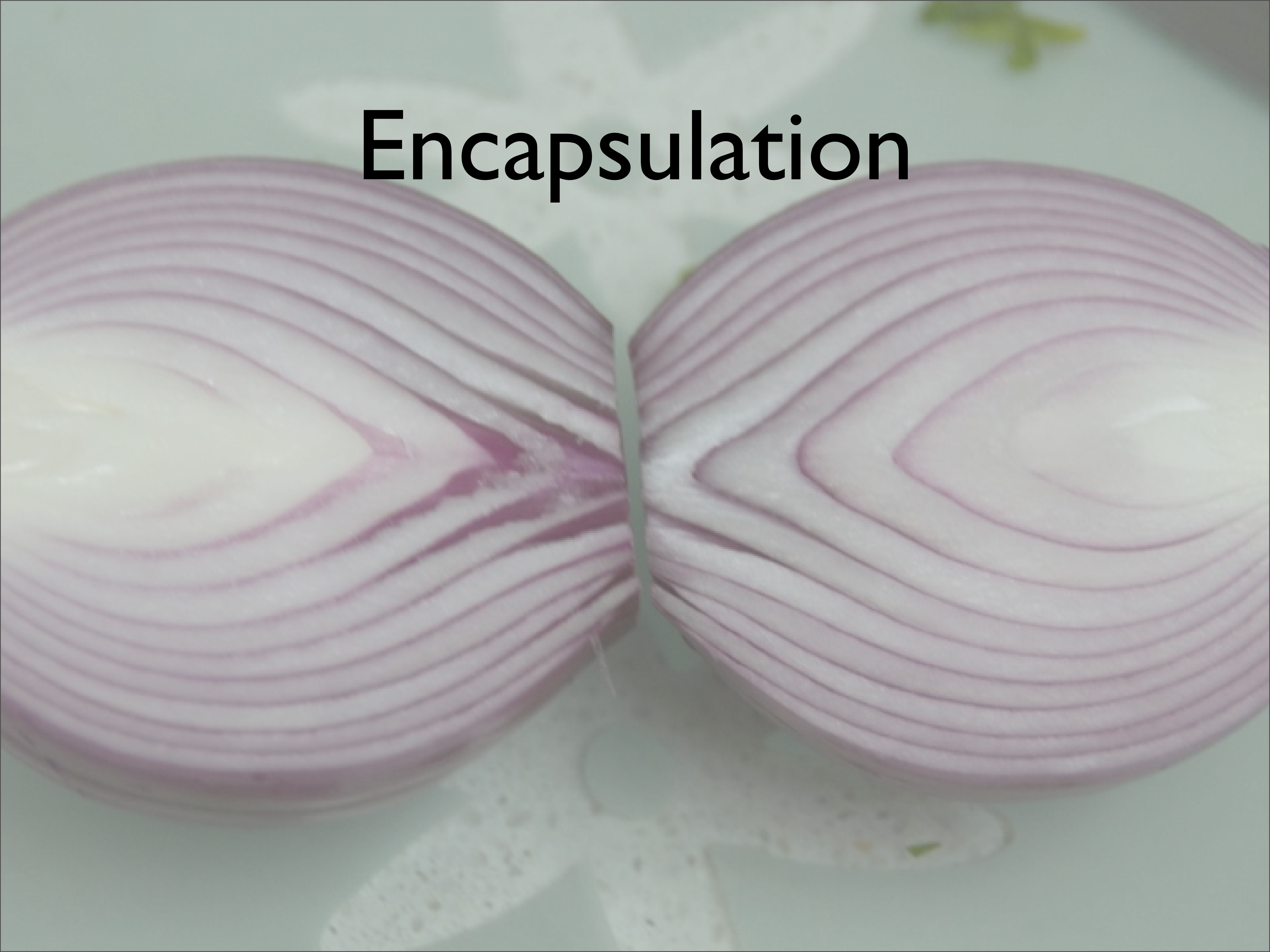
Unspaghettifying Your Objects

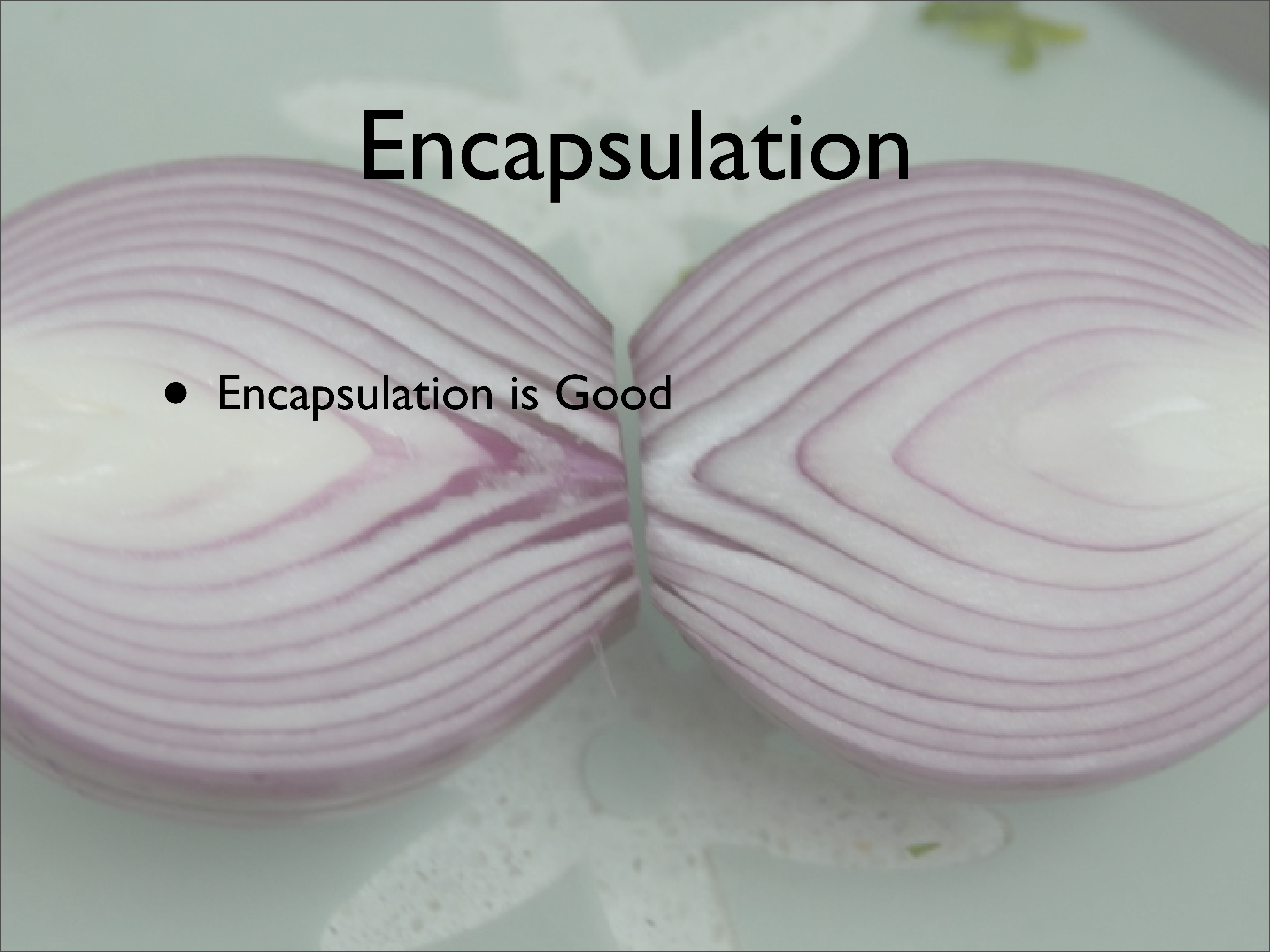
- Keep your *inheritance* graphs well trimmed - Don’t write Onion Code!
- Keep your *call* graphs well trimmed. Don’t have long chains of function calls.

Encapsulation



Encapsulation





Encapsulation

- Encapsulation is Good

Encapsulation

- Encapsulation is Good
- Excessive Encapsulation is Evil

Encapsulation

- Encapsulation is Good
- Excessive Encapsulation is Evil
- Excessive Encapsulation is Evil

Encapsulation

- Encapsulation is Good
- Excessive Encapsulation is Evil
- Excessive Encapsulation is Evil
- Don’t encapsulate the encapsulated. Don’t write Onion Code!

Encapsulation

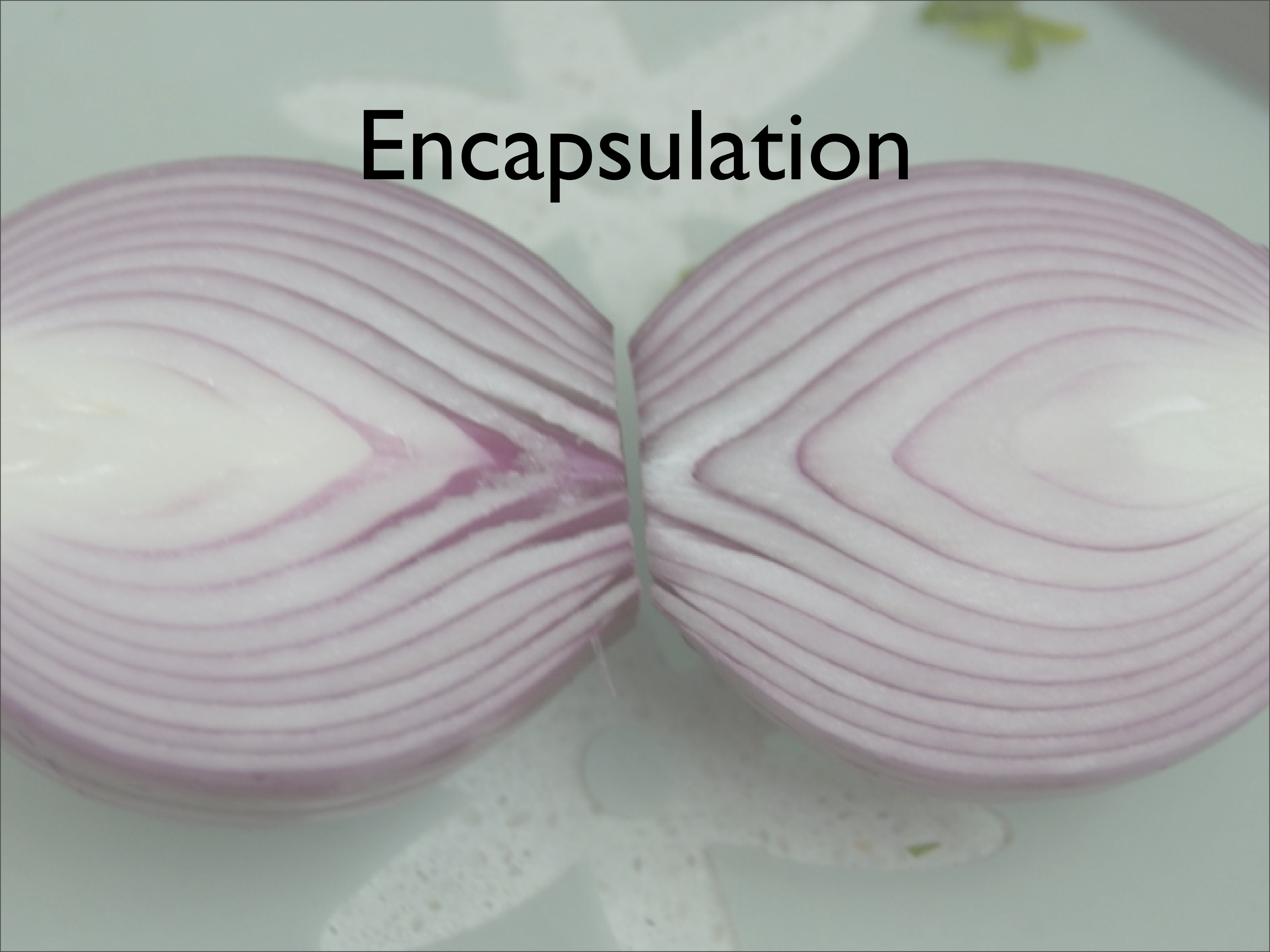
```
class Door{
    public:
        Door():_handle(Handle()){}

        void open(){
            if(!_handle.locked()){
                _opened = true;
            }
        }

        void close(){
            _opened = false;
        }
    protected:
        bool _opened;
        Handle _handle;
}
```

Encapsulation

```
class RedDoor:public Door{
    public:
        RedDoor():_colour("red"){}
    protected:
        string _colour;
}

class RedDoorWithAluminiumHandle: public RedDoor{
    RedDoorWithAluminiumHandle():_colour("red"),_handle(Handle("
aluminium")){}
}

RedDoorWithAluminiumHandle
makeRedDoorWithAluminiumHandle(){
    return RedDoorWithAluminiumHandle();
}
```

Encapsulation

```
class Door{
    public:
        Door(string colour, Handle handle):
            _colour(colour), _handle(handle){}
        void open(){
            if(!_handle.locked()){
                _opened = true;
        }
        void close(){
            _opened = false;
        }
    private:
        string _colour;
        Handle _handle;
}

Door
makeRedDoorWithAluminiumHandle(){
    handle = Handle("aluminium");
    return Door("red", handle);
}
```



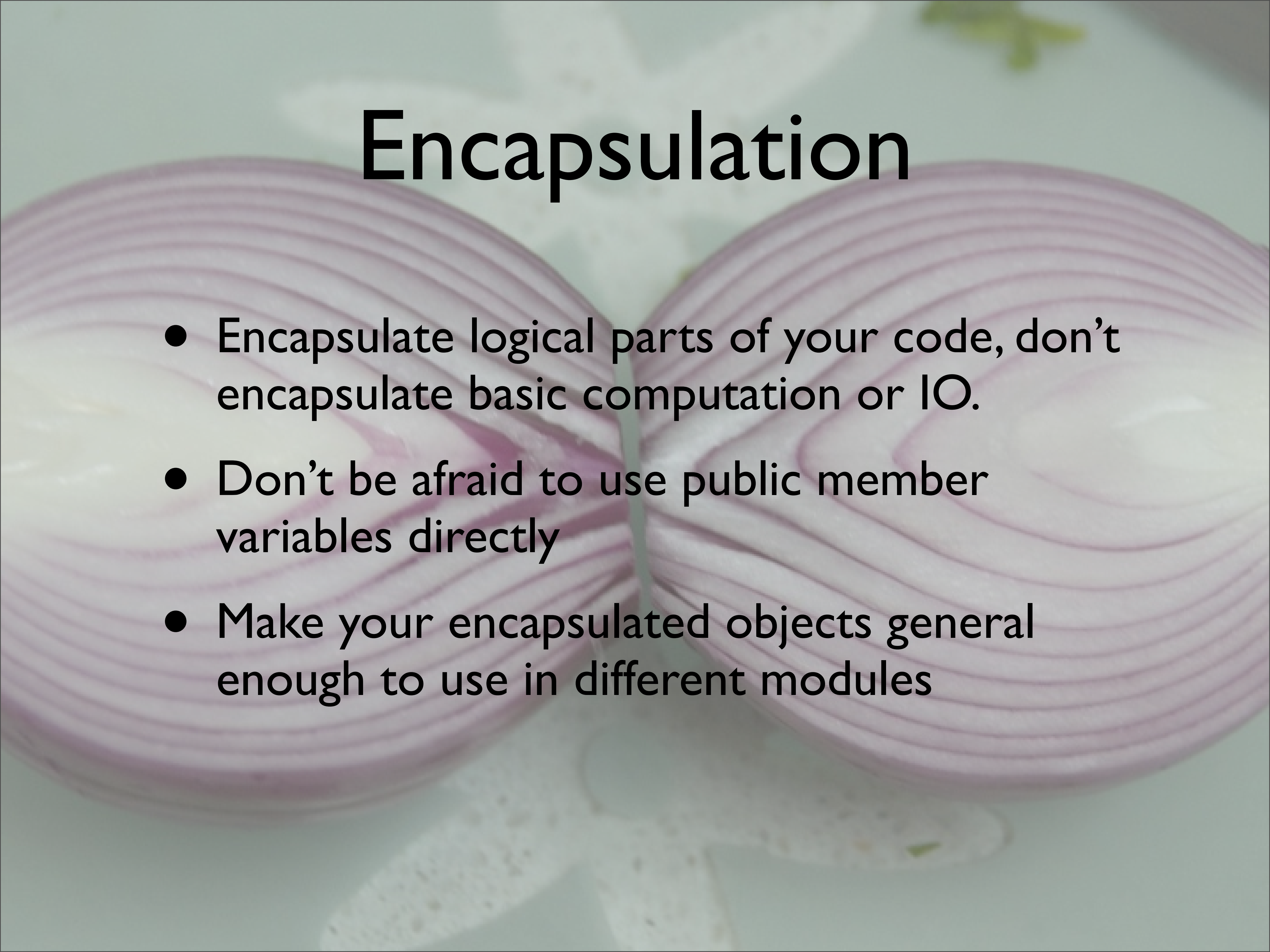
Encapsulation

Encapsulation

- Encapsulate logical parts of your code, don’t encapsulate basic computation or IO.

Encapsulation

- Encapsulate logical parts of your code, don't encapsulate basic computation or IO.
- Don't be afraid to use public member variables directly



Encapsulation

- Encapsulate logical parts of your code, don't encapsulate basic computation or IO.
- Don't be afraid to use public member variables directly
- Make your encapsulated objects general enough to use in different modules

Encapsulation

- Encapsulate logical parts of your code, don't encapsulate basic computation or IO.
- Don't be afraid to use public member variables directly
- Make your encapsulated objects general enough to use in different modules
- Composition or Inheritance?

Threads are Hard

"To offer another analogy, a folk definition of insanity is to do the same thing over and over again and expect the results to be different. By this definition, we in fact require that programmers of multithreaded systems be insane. Were they sane, they could not understand their programs."

-Edward A. Lee

Well see, it all started
with a loose thread
and just went downhill from there
ICANHASCHEEZBURGER.COM

Threads are Hard

Threads are Hard

- If it can be done in one thread, do it in one thread.
- Keep the multithreaded minority of your code separate from the single threaded majority.
- Use message passing rather than shared memory.

Summary

Summary

- Minimise your state.

Summary

- Minimise your state.
- Make your code as simple as possible but no simpler.

Summary

- Minimise your state.
- Make your code as simple as possible but no simpler.
- Control your data flow.

Simplicity is prerequisite for reliability

- Edsger W. Dijkstra

Questions?

More Information

Details of this talk and a copy of the slides are available at:

http://sara.falamaki.id.au/moin/Writing/ProgrammingTips

Photo Credits

http://www.flickr.com/photos/23232902@N05/2542353761/ (exception)

http://mine.icanhascheezburger.com/view.aspx?ciid=3900660 (trim)

http://mine.icanhascheezburger.com/view.aspx?ciid=4027009 (hot potato)

http://www.flickr.com/photos/vox/2208835201/sizes/o/ (pipes)

http://icanhascheezburger.com/2008/11/05/funny-pictures-see-it-all-started-with-a-loose-thread-and-just-went-downhill-from-there/ (threads)

http://www.flickr.com/photos/designedlykristi/122619504/sizes/o/

http://icanhascheezburger.com/2007/06/02/im-in-ur-quantum-box/