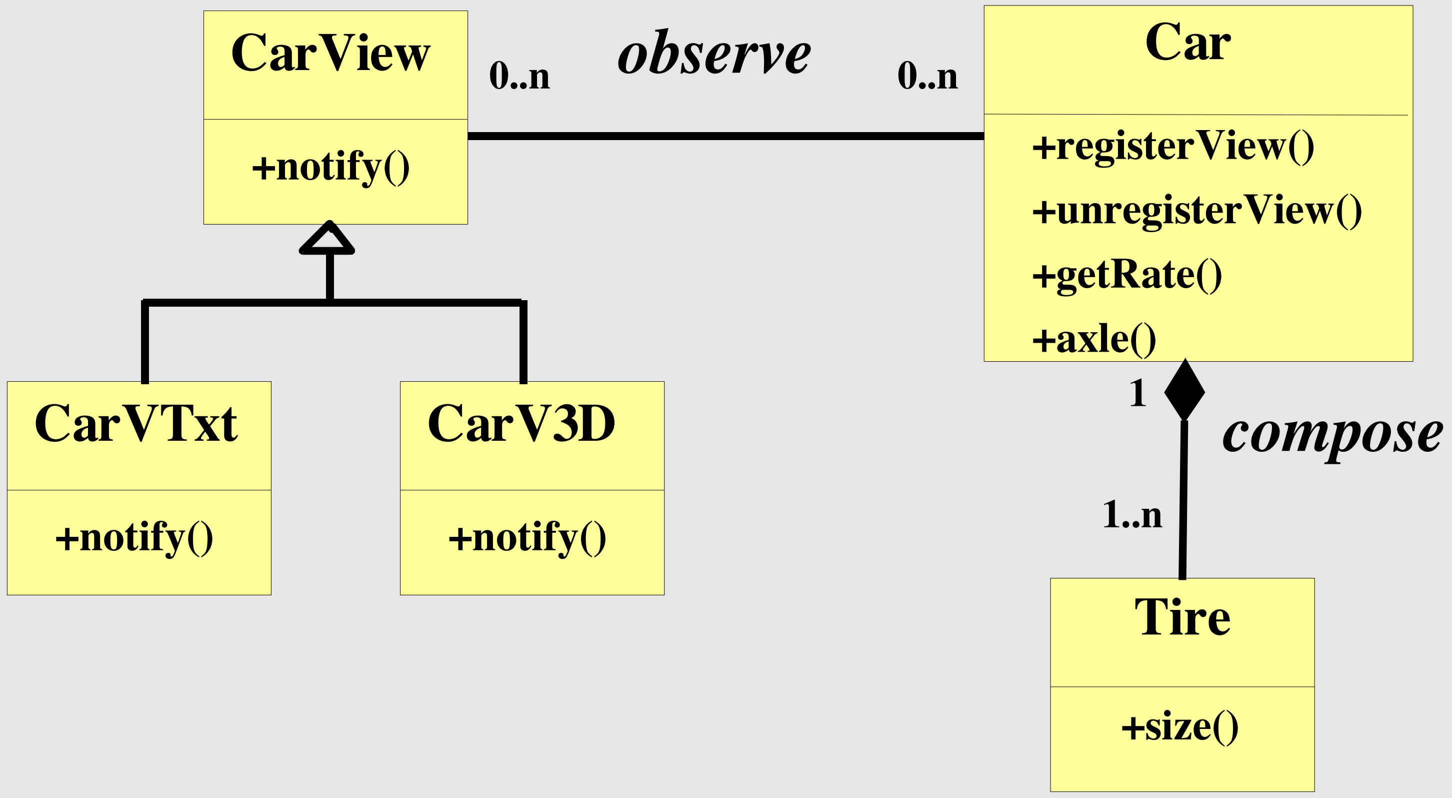
Interaction-based Programming in Classages

Y. David Liu

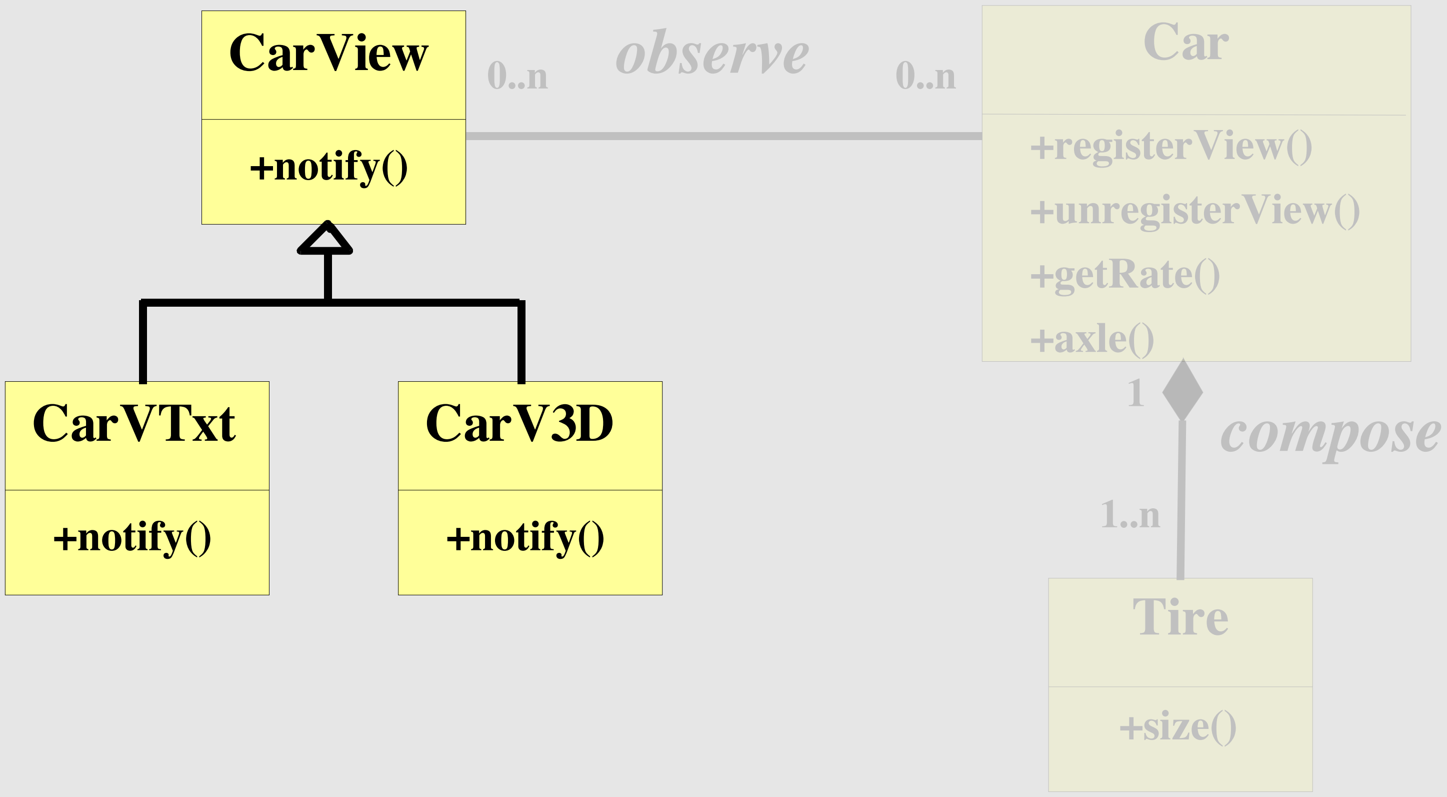
Scott Smith

Johns Hopkins University

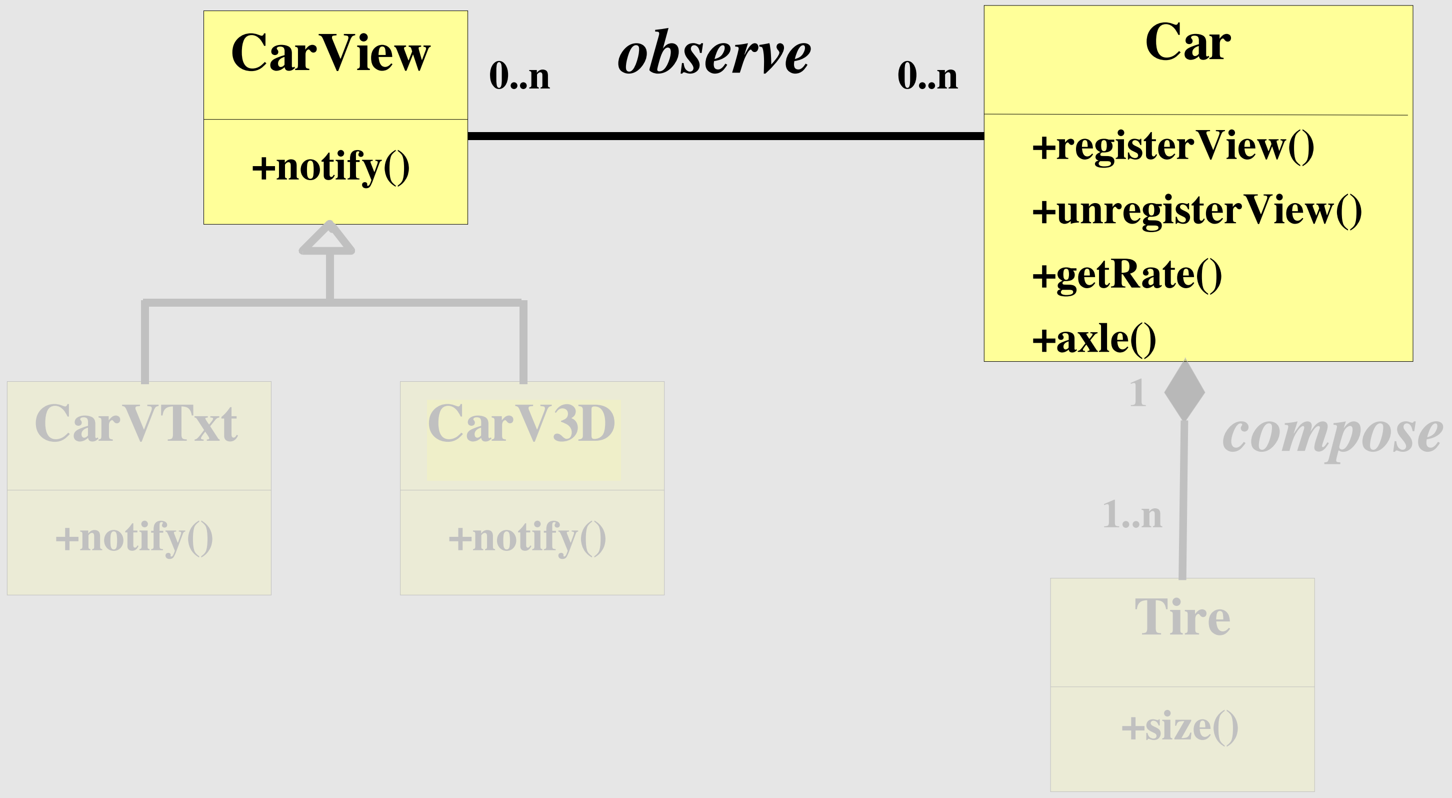
Interactions Illustrated by UML



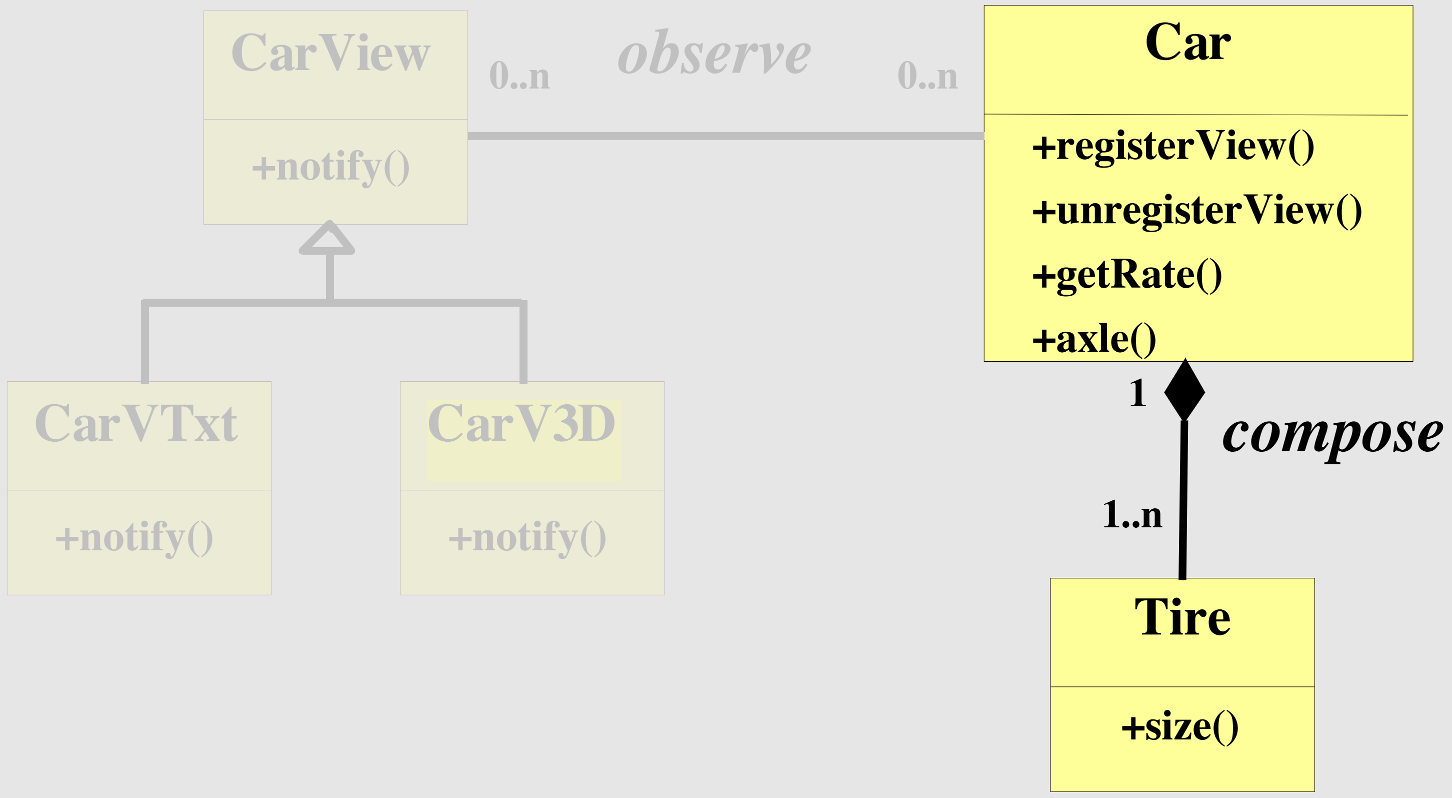
Interactions Illustrated by UML



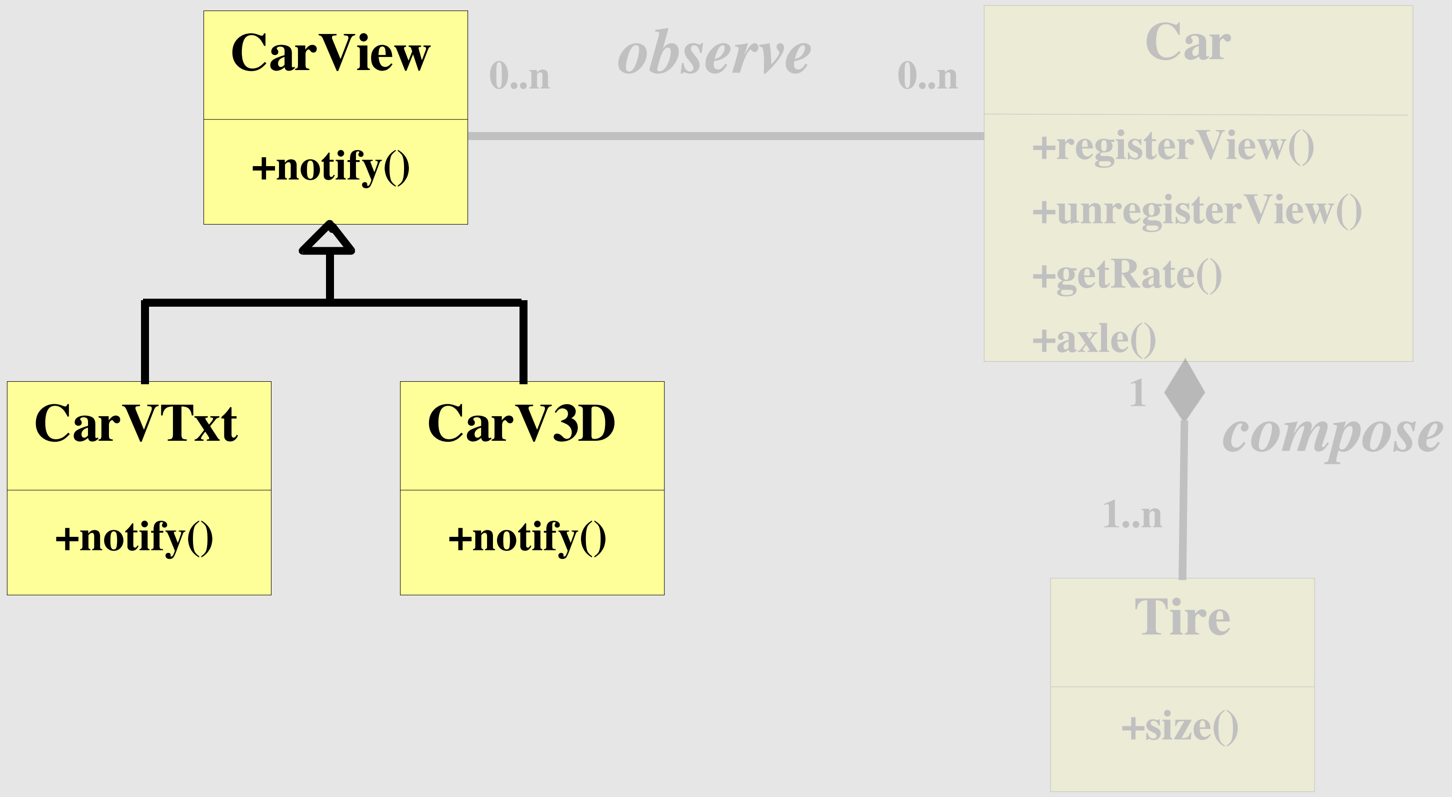
Interactions Illustrated by UML



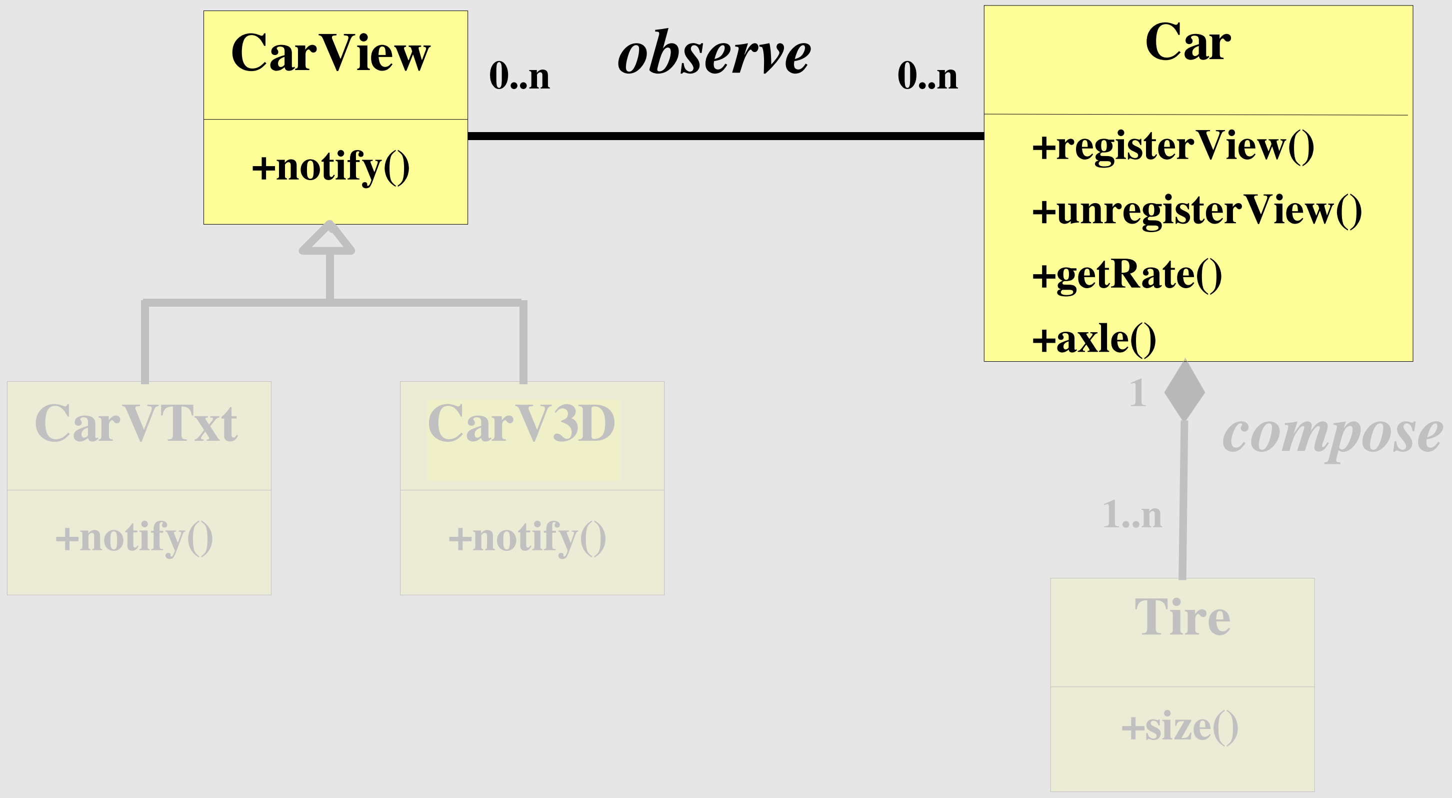
Interactions Illustrated by UML



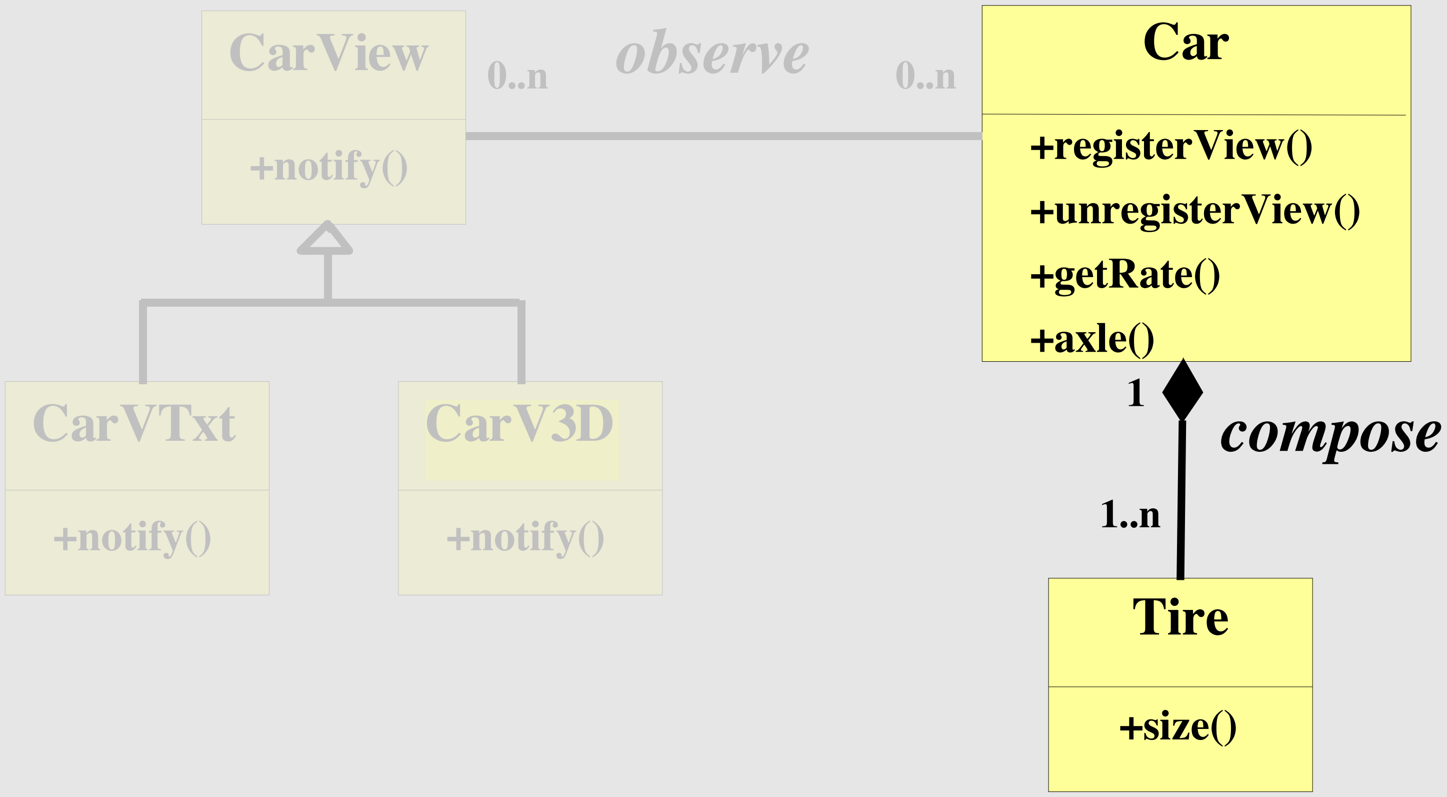
Class Interaction



Object Peer-to-Peer Interaction



Object Whole-Part Interaction

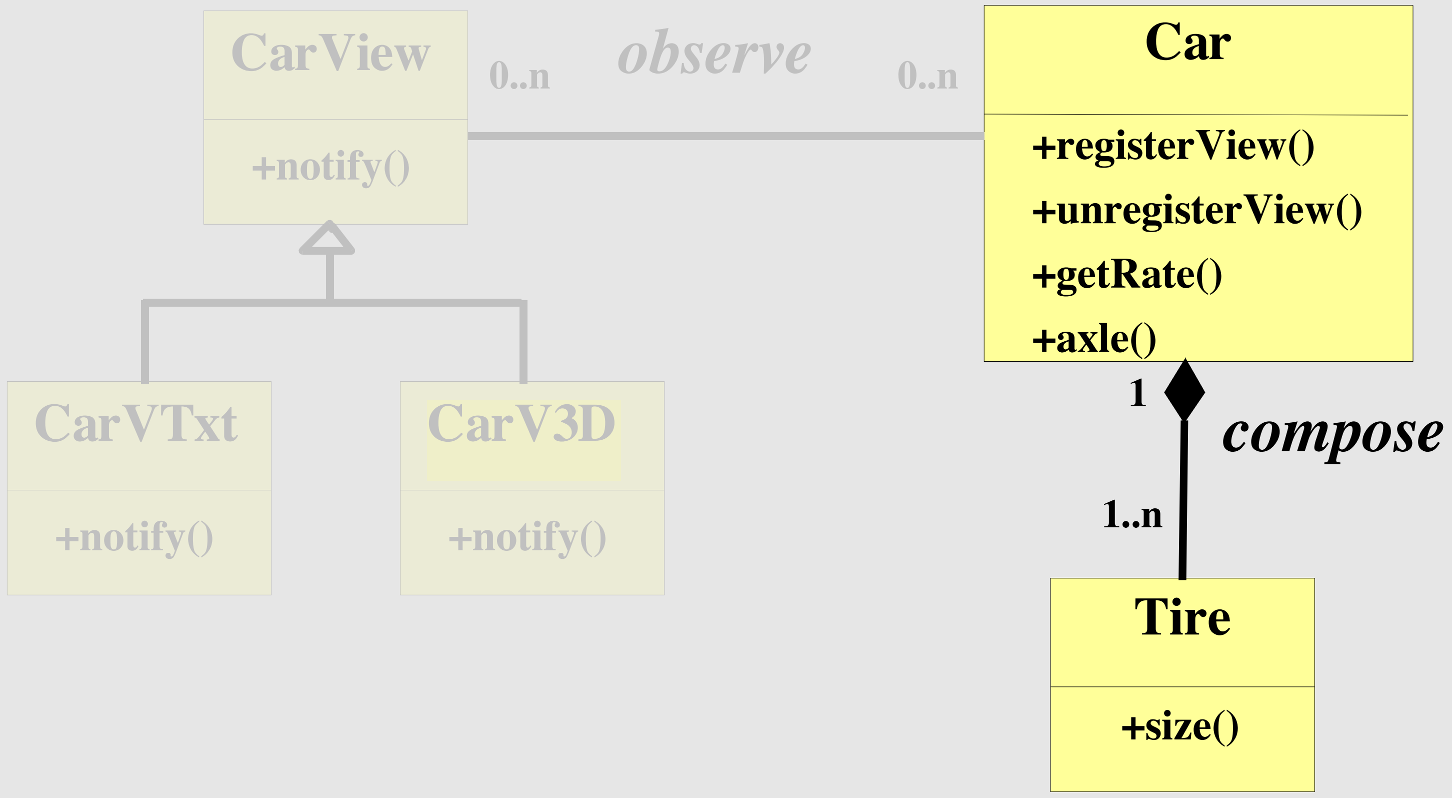


Limitations of Mainstream OOPLs on Object Interactions

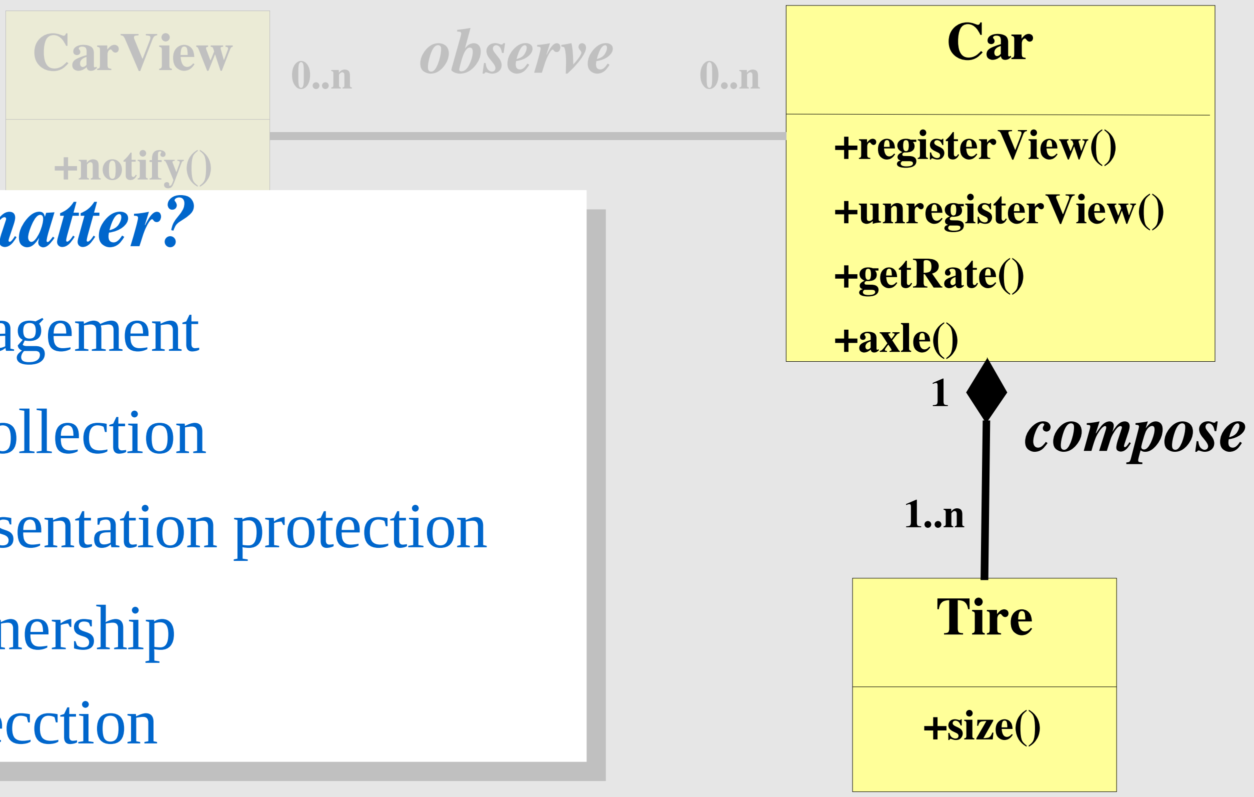
Limitations of Mainstream OOPLs on Object Interactions

- No explicit support for whole-part interactions.
- Inadequate support for coarse-grained interactions between objects.
- An object has only one encapsulation-enforceable interface for all interactions it might participate in.
- Interaction bi-directional dependencies are not explicit.

Object Whole-Part Interaction



Object Whole-Part Interaction



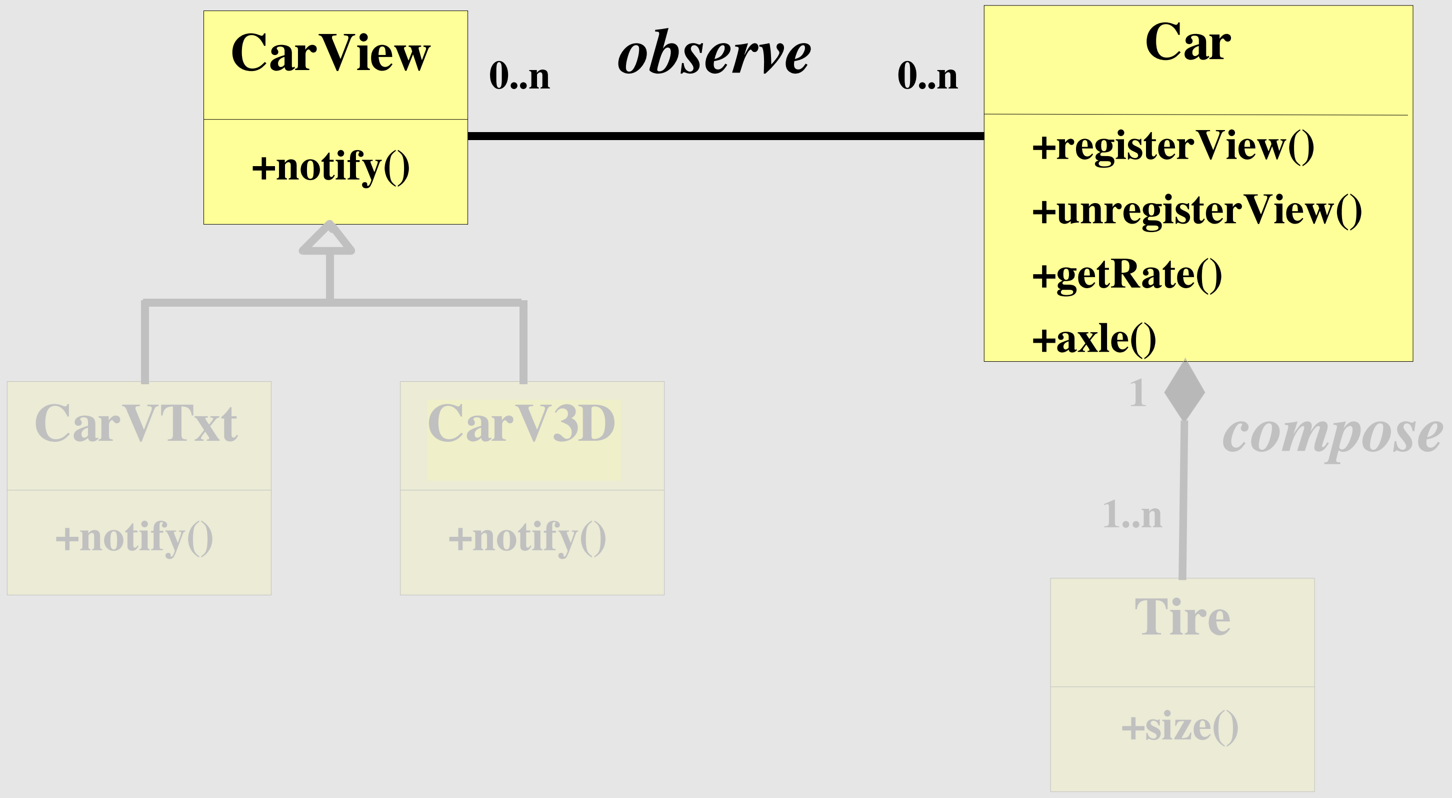
Why does it matter?

- memory management
  - garbage collection
- internal representation protection
  - object ownership
  - alias protecction

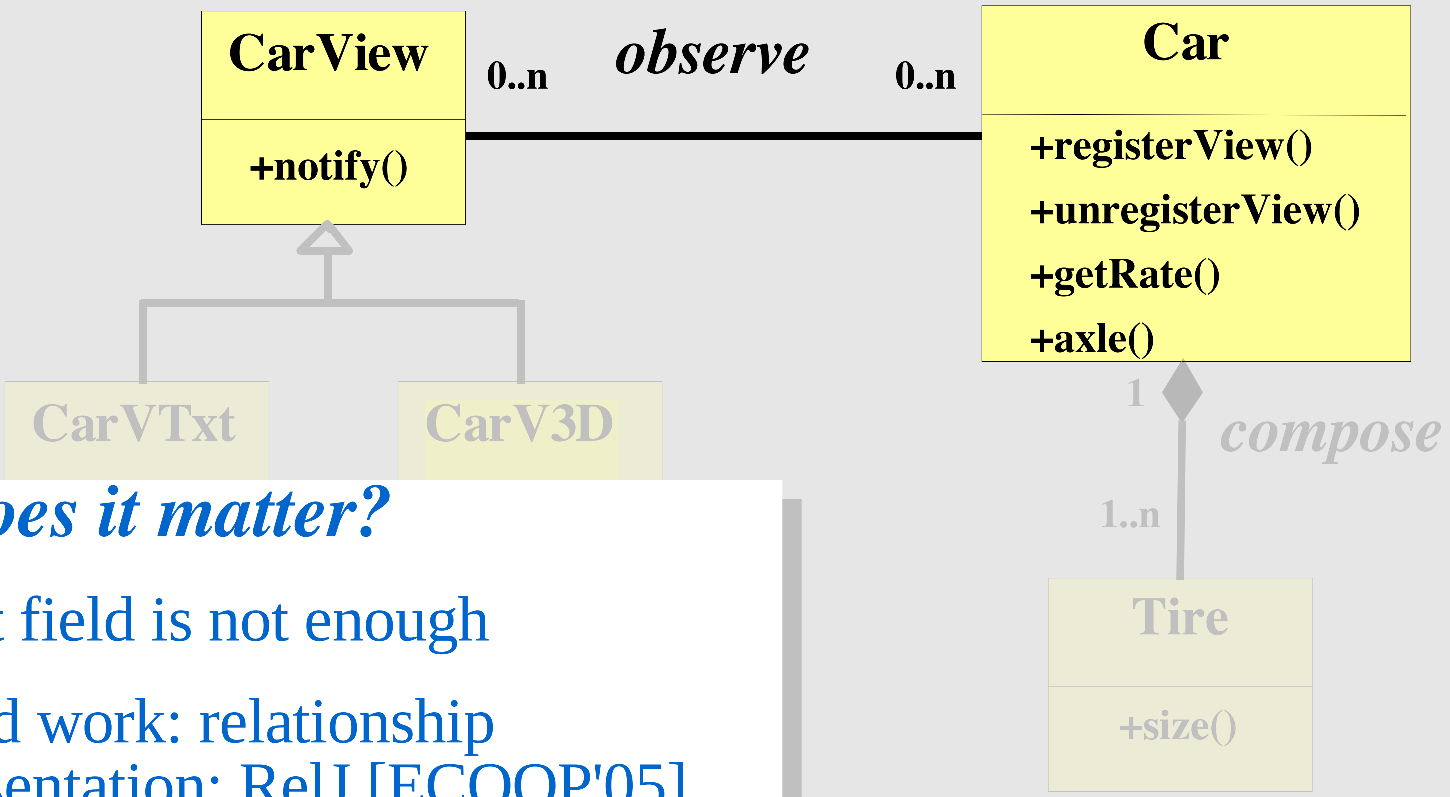
Limitations of Mainstream OOPLs on Object Interactions

- No explicit support for whole-part interactions.
- Inadequate support for coarse-grained interactions between objects.
- An object has only one encapsulation-enforceable interface for all interactions it might participate in.
- Interaction bi-directional dependencies are not explicit.

Representing the *observe* Interaction



Representing the *observe* Interaction



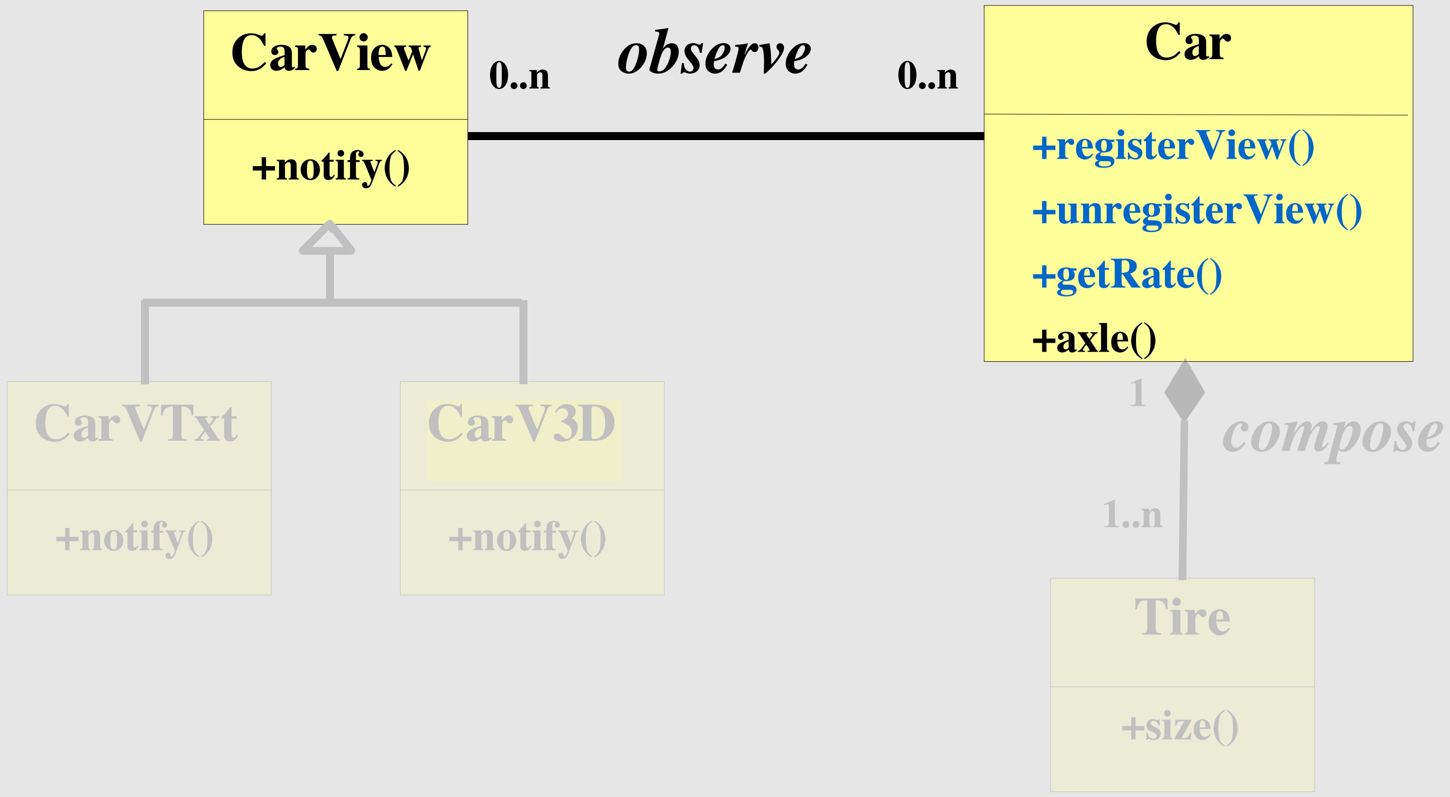
Why does it matter?

- object field is not enough
- related work: relationship representation: RelJ [ECOOP'05]

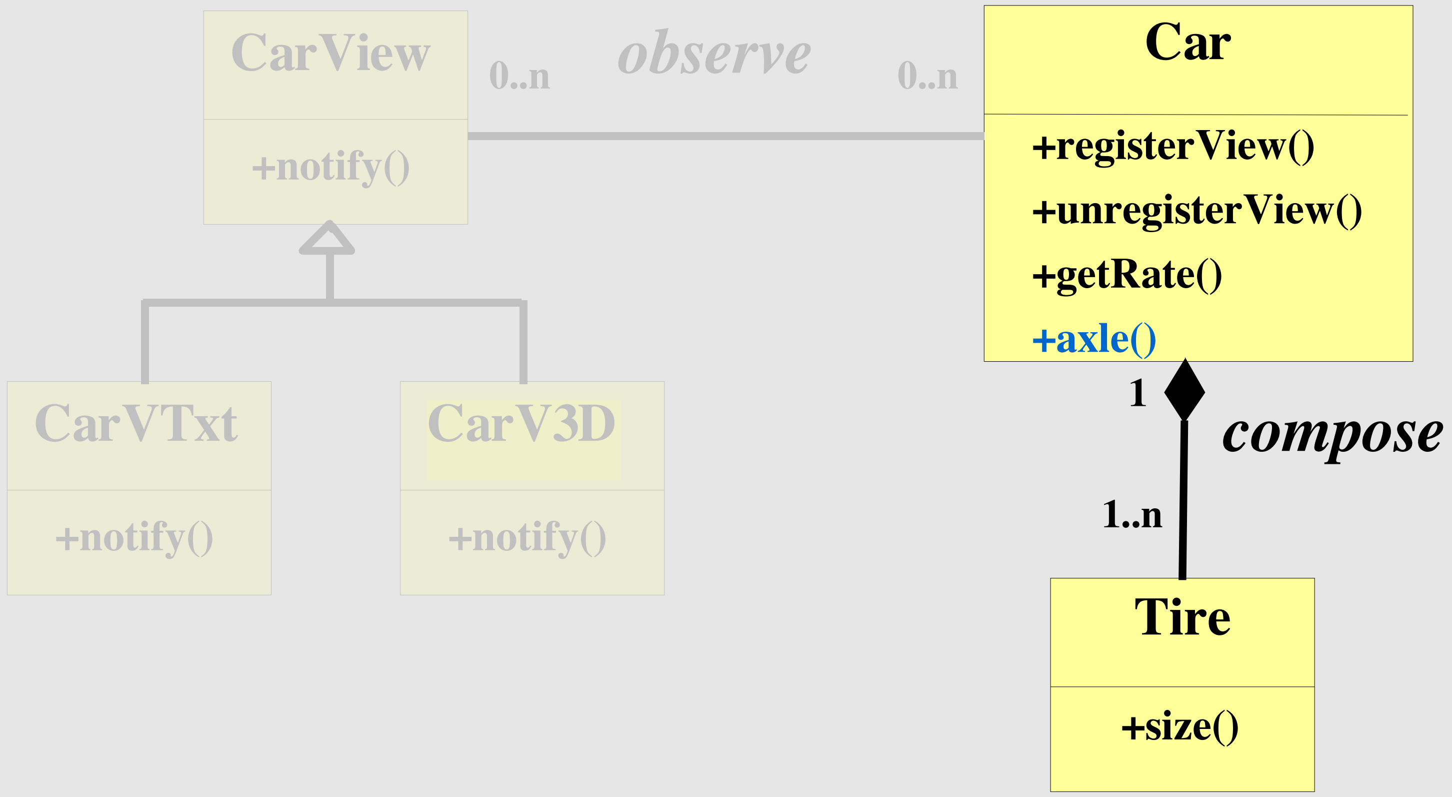
Limitations of Mainstream OOPLs on Object Interactions

- No explicit support for whole-part interactions.
- Inadequate support for coarse-grained interactions between objects.
- An object has only one encapsulation-enforceable interface for all interactions it might participate in.
- Interaction bi-directional dependencies are not explicit.

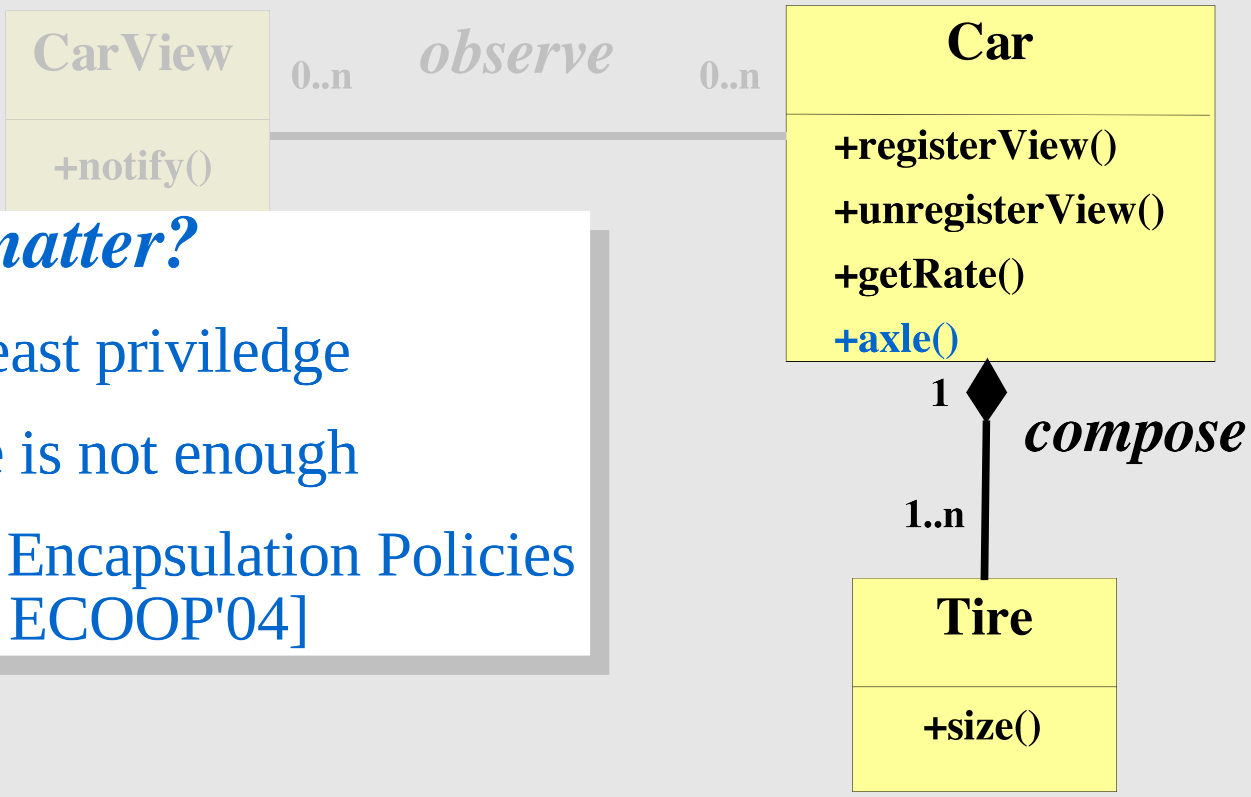
Only One Interface for All Ineractions



Only One Interface for All Interactions



Only One Interface for All Interactions



Why does it matter?

- principle of least priviledge
- Java interface is not enough
- related work: Encapsulation Policies [Scharli et al, ECOOP'04]

Limitations of Mainstream OOPLs on Object Interactions

- No explicit support for whole-part interactions.
- Inadequate support for coarse-grained interactions between objects.
- An object has only one encapsulation-enforceable interface for all interactions it might participate in.
- Interaction bi-directional dependencies are not explicit.
    - Callbacks

Limitations of Mainstream OOPLs on Class Interactions

Limitations of Mainstream OOPLs on Class Interactions

- Interaction bi-directional dependencies are not explicit.
  - Dependencies between superclasses and subclasses are fundamentally bi-directional.
- Interfaces for class interations are tangled with those for object interactions.
- A superclass has only one interface for all subclasses.

Limitations of Mainstream OOPLs on Class Interactions

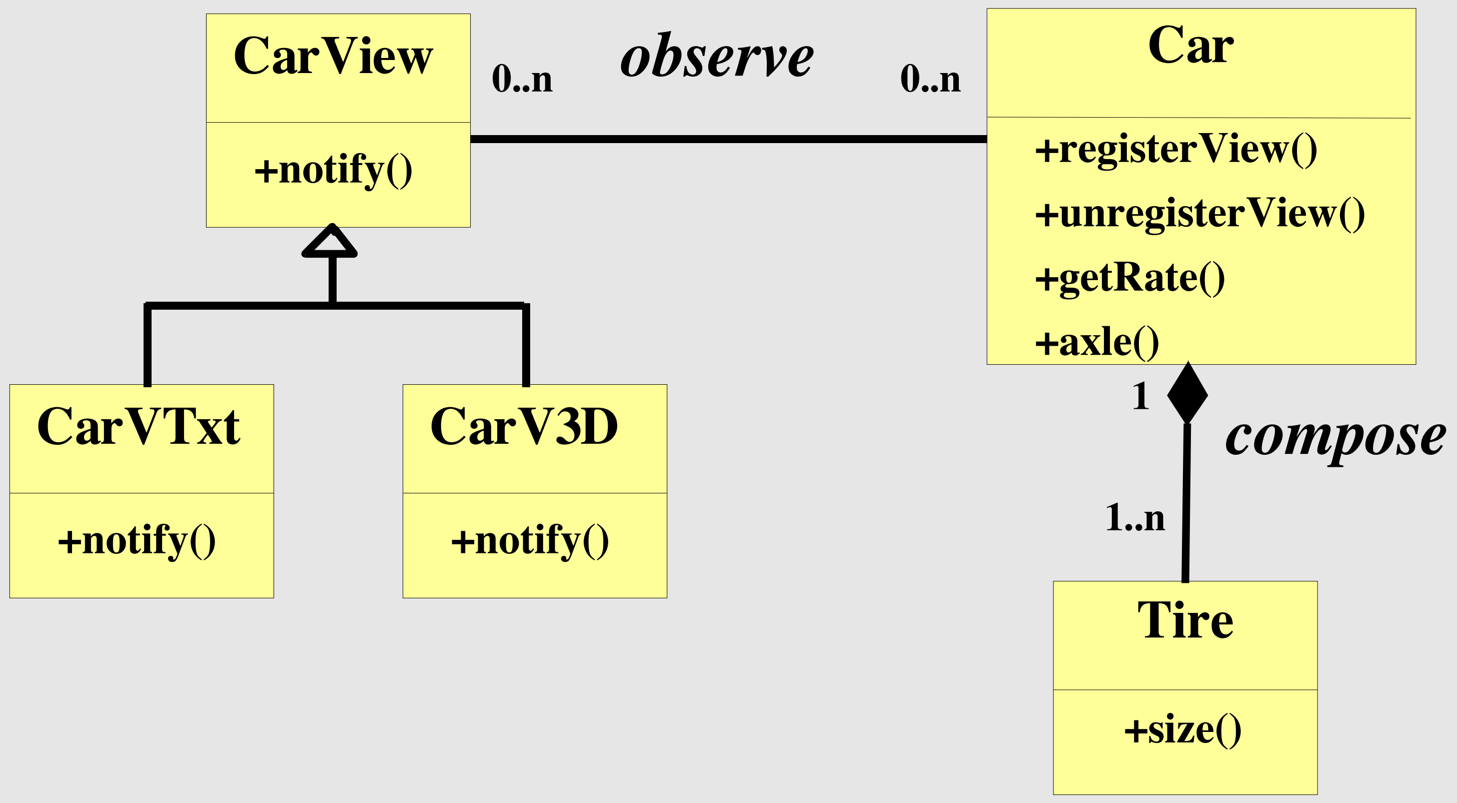
- Interaction bi-directional dependencies are not explicit.
- Interfaces for class interations are tangled with those for object interactions.
    - related work: Traits
- A superclass has only one interface for all subclasses.

Limitations of Mainstream OOPLs on Class Interactions

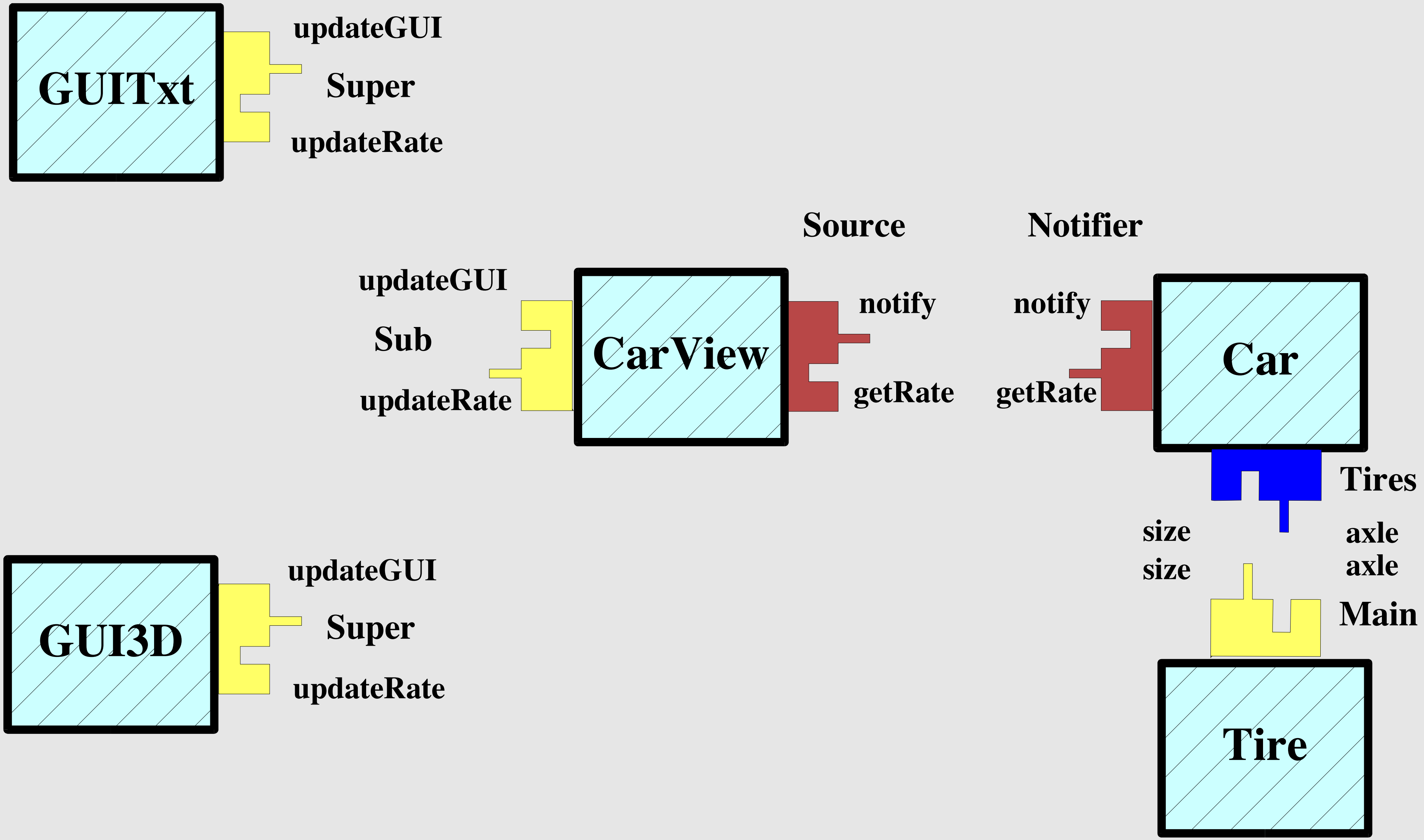
- Interaction bi-directional dependencies are not explicit.
- Interfaces for class interations are tangled with those for object interactions.
- A superclass has only one interface for all subclasses.

The Classages Solution

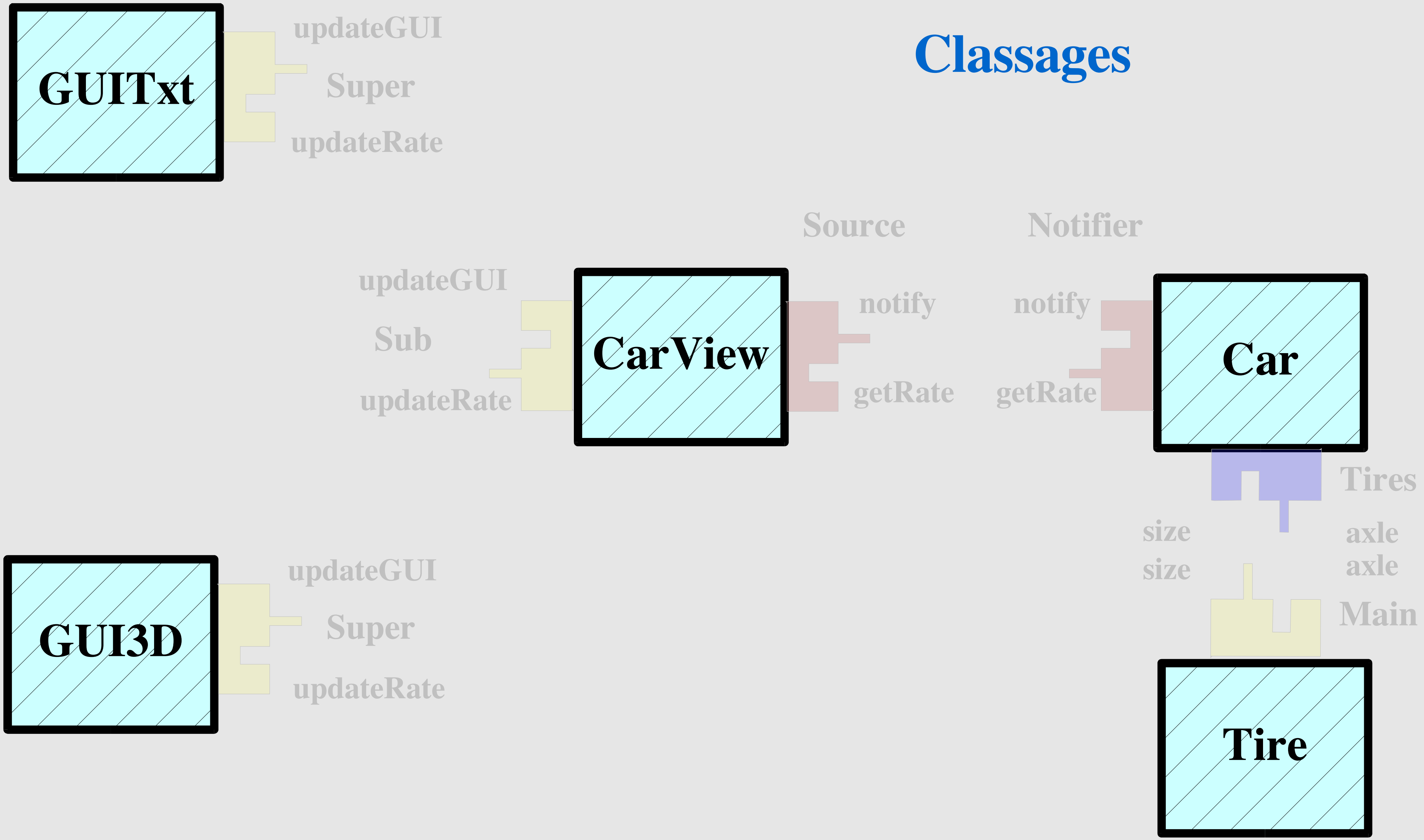
The Simple Example Revisited



The Same Example in Classages

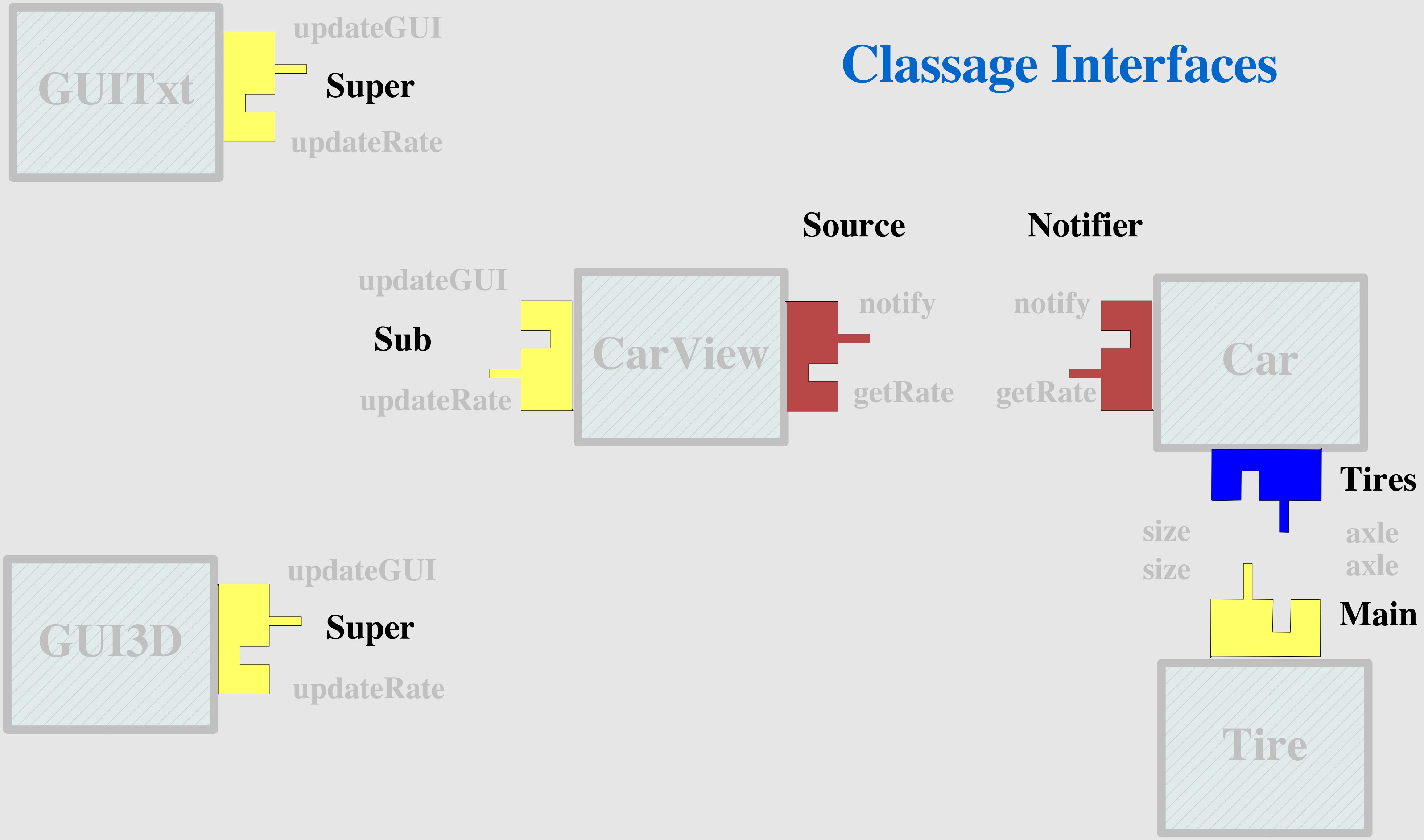


Classage Basics: Classages

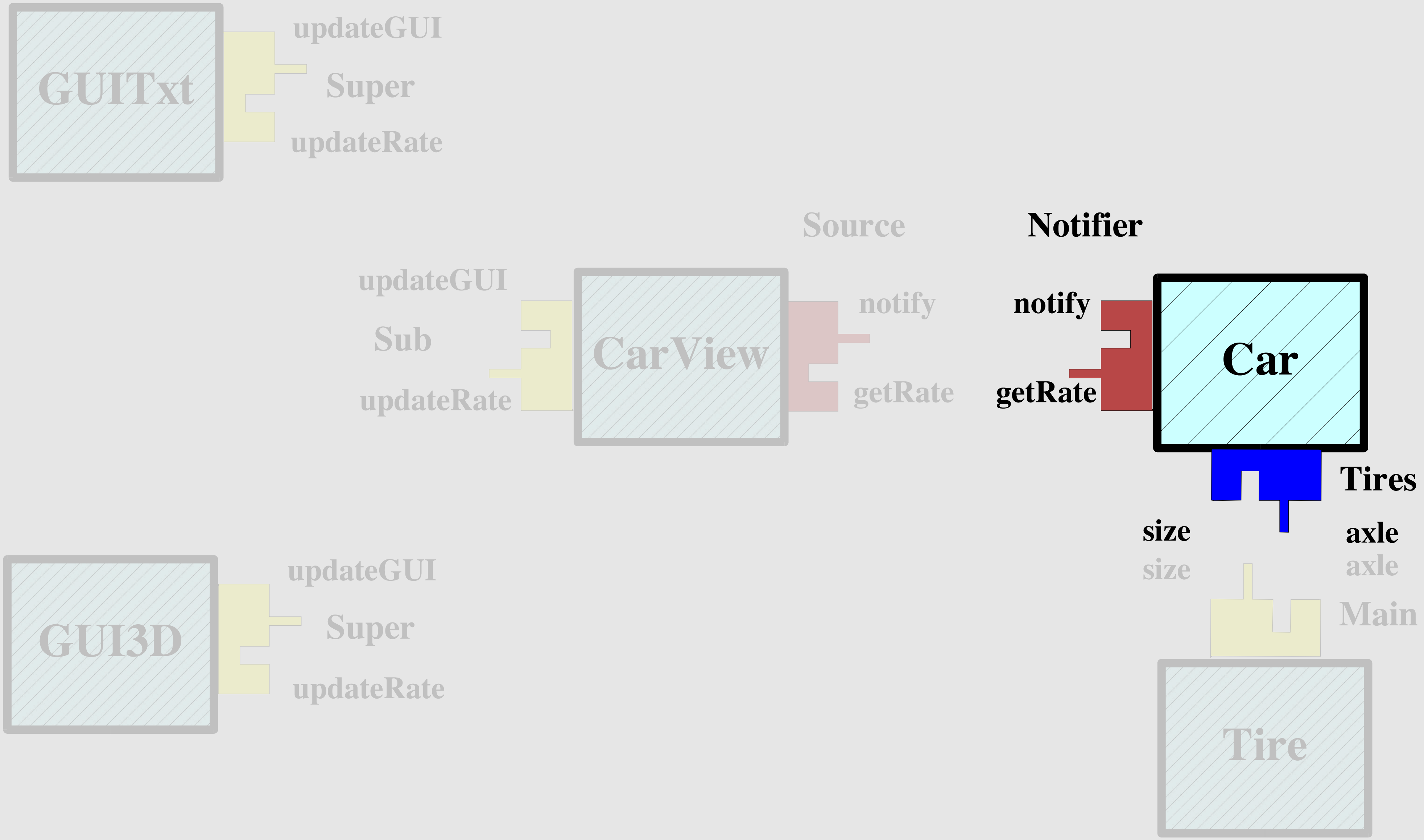


Classage Basics

Classage Interfaces

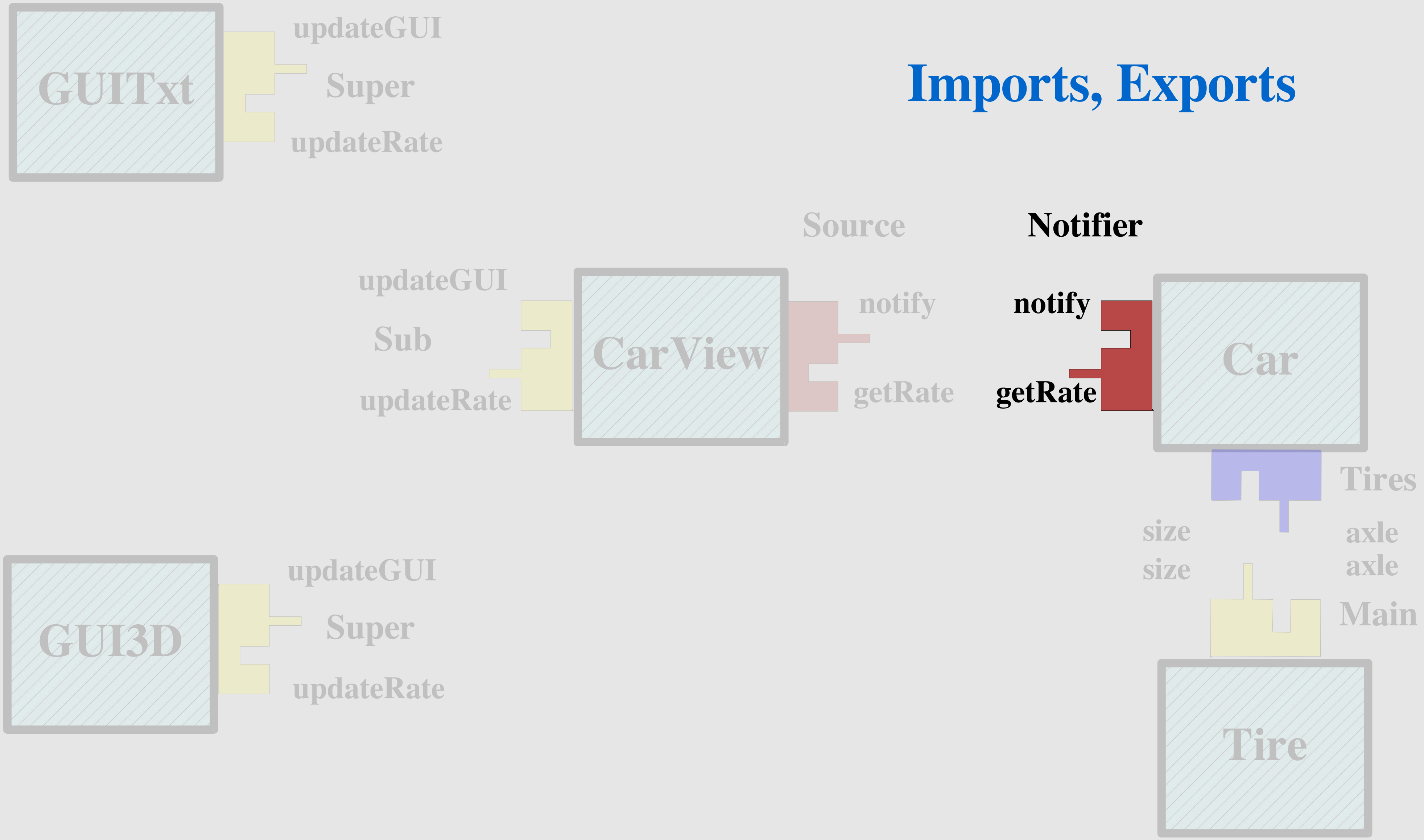


Classage Basics

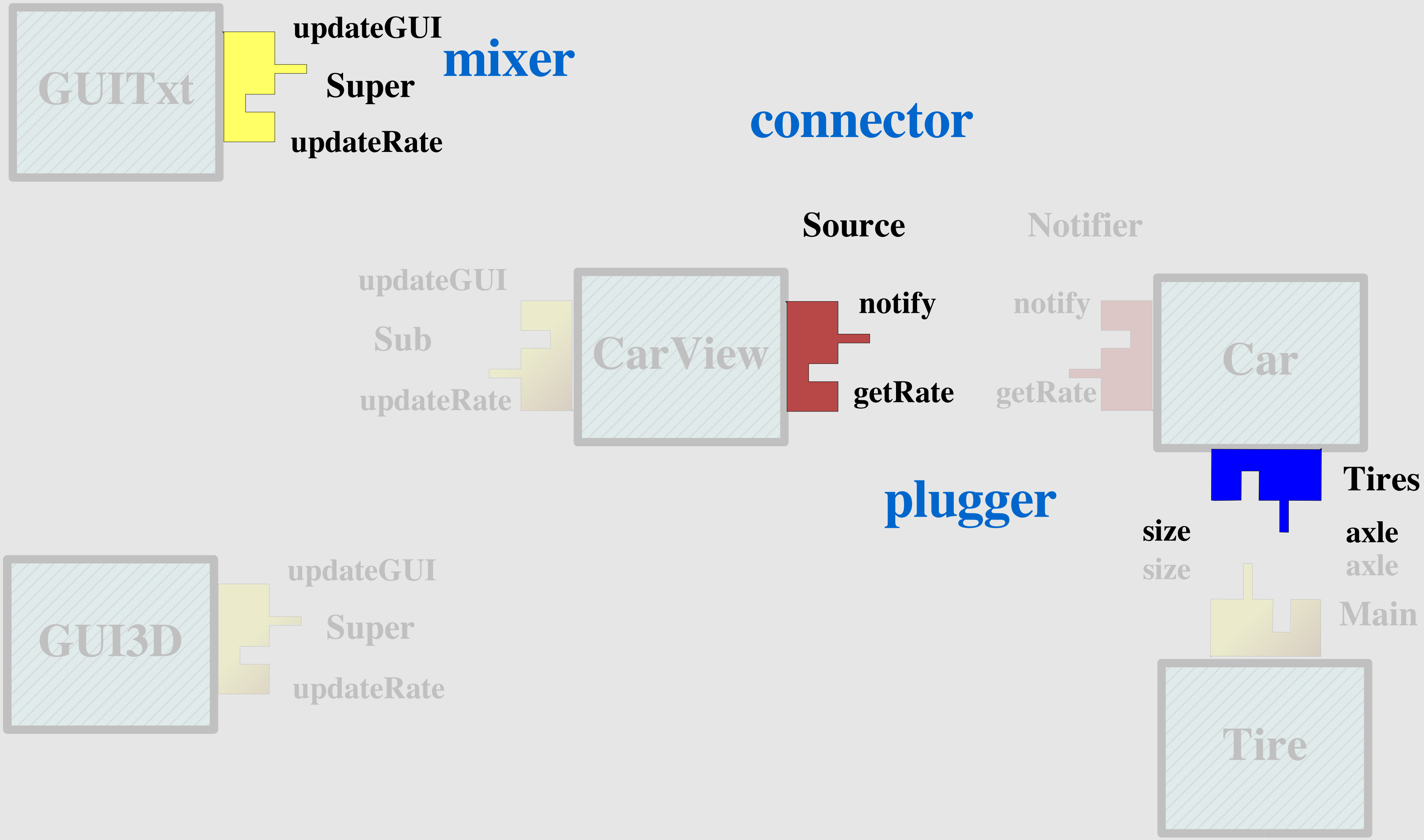


Classage Basics

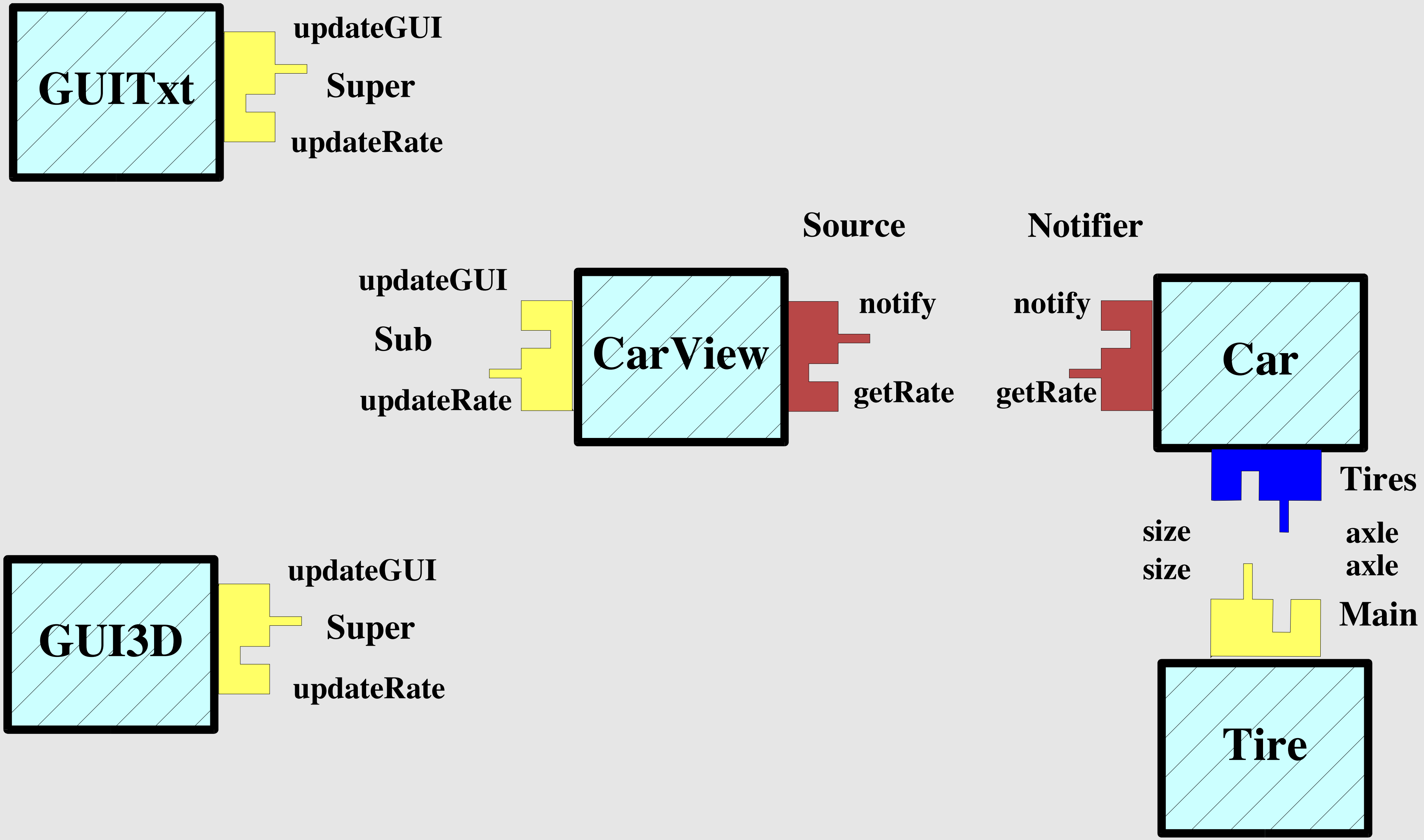
Imports, Exports



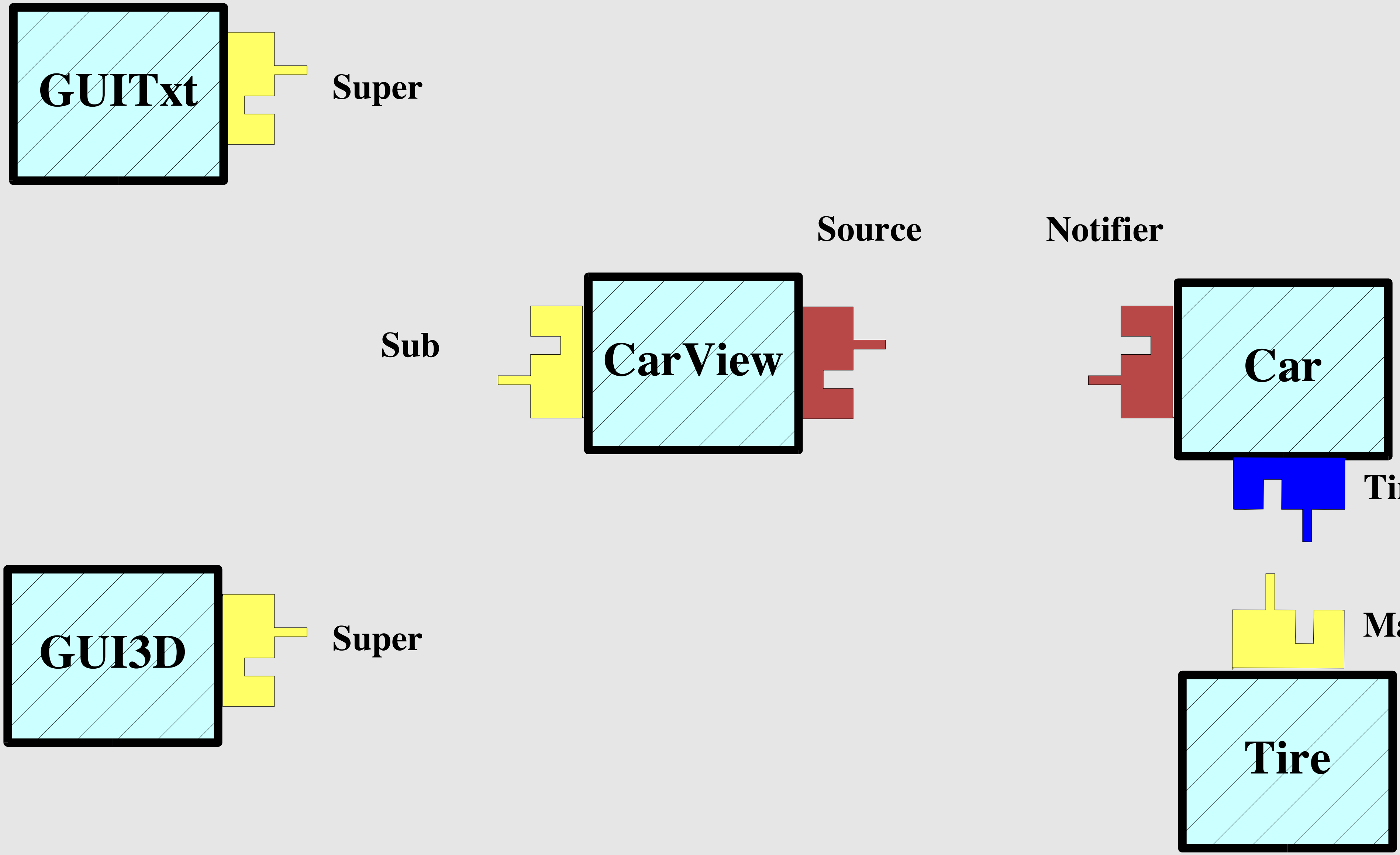
Classage Basics



A Demo

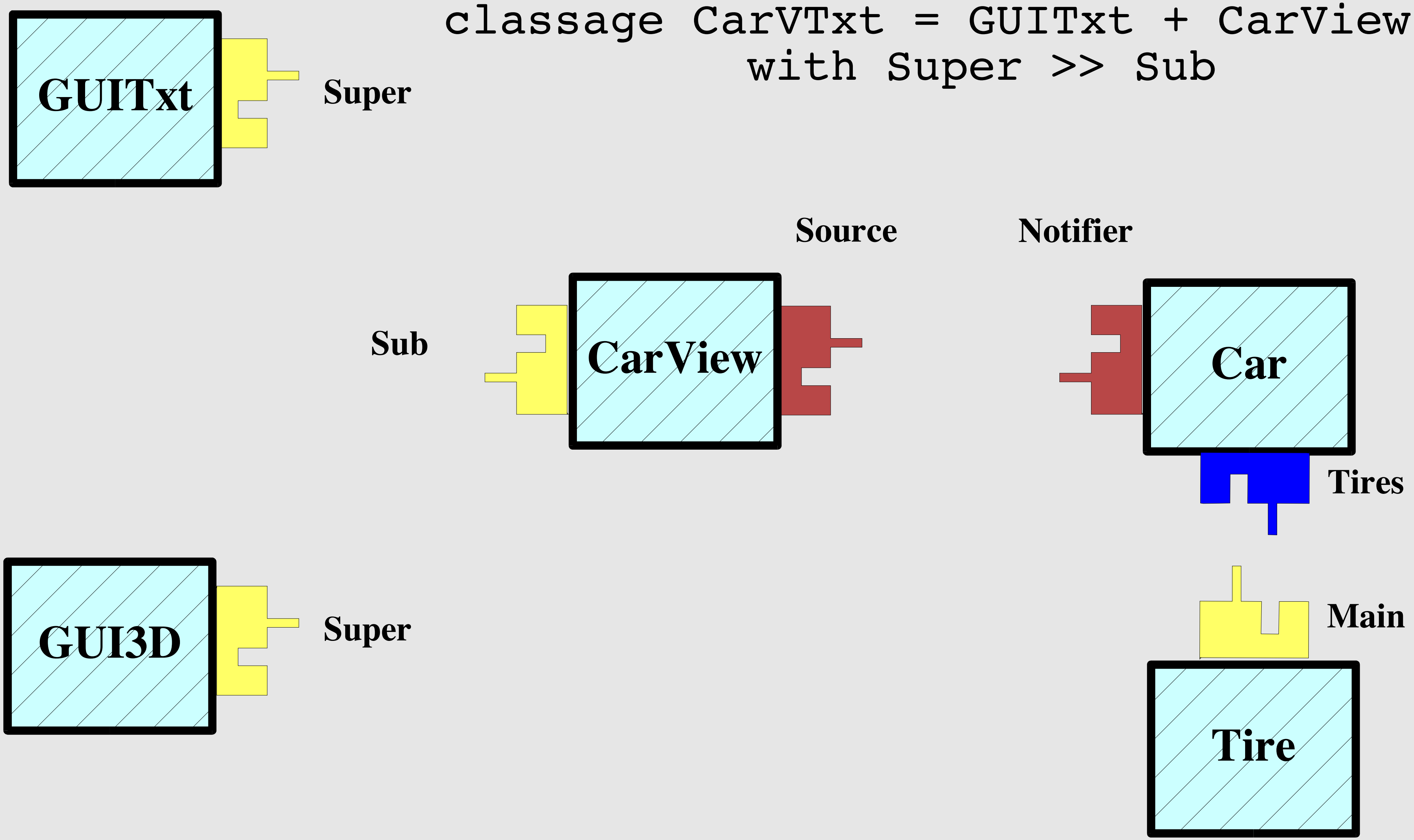

GUITxt
updateGUI
Super
updateRate
Source
Notifier
updateGUI
Sub
updateRate
CarView
notify
getRate
notify
getRate
Car
Tires
size
size
axle
axle
Main
Tire
GUI3D
updateGUI
Super
updateRate

A Demo

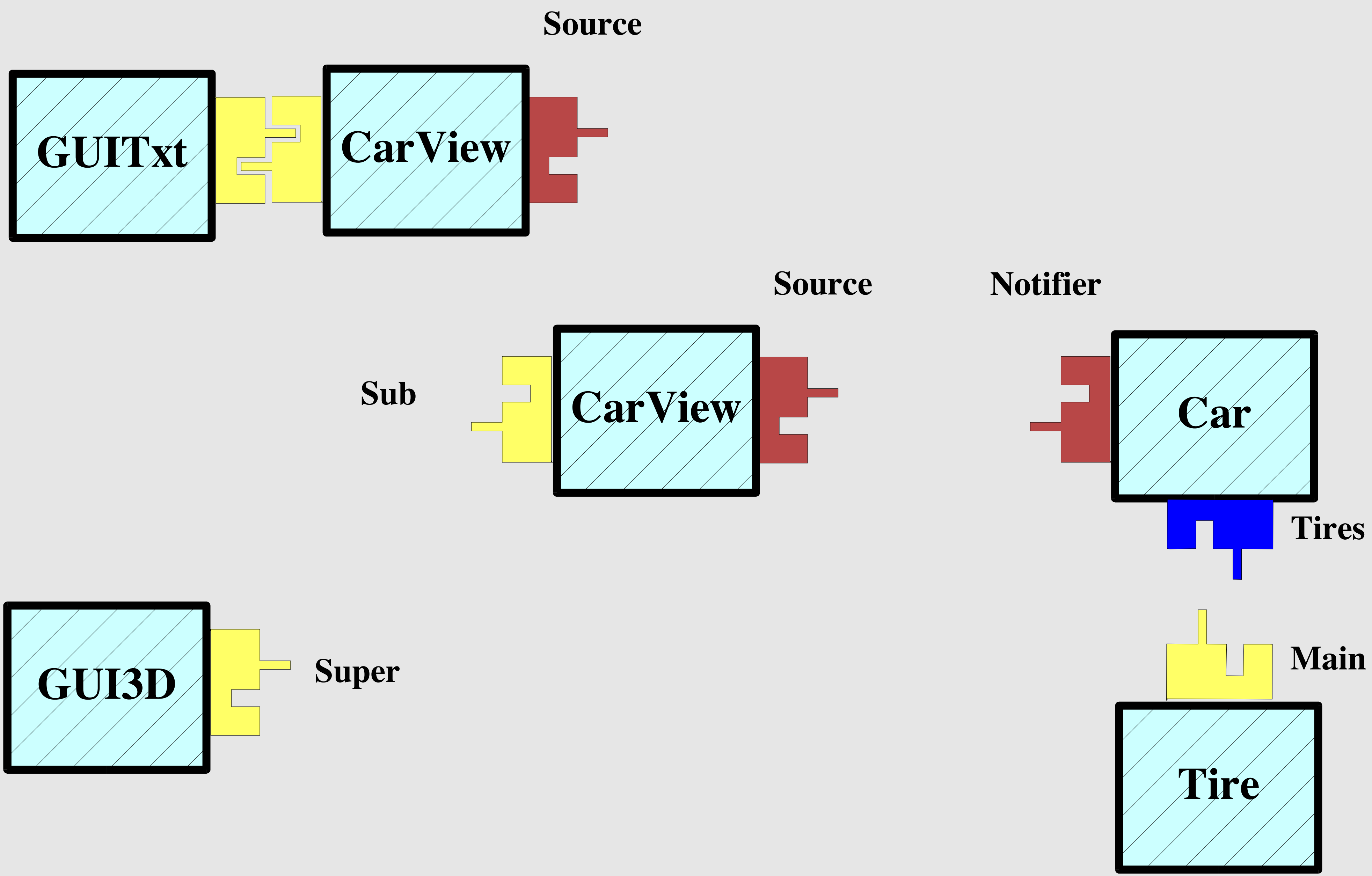

GUITxt
Super
Source
Notifier
Sub
CarView
Car
Tires
Main
GUI3D
Super
Tire

At Compile Time...

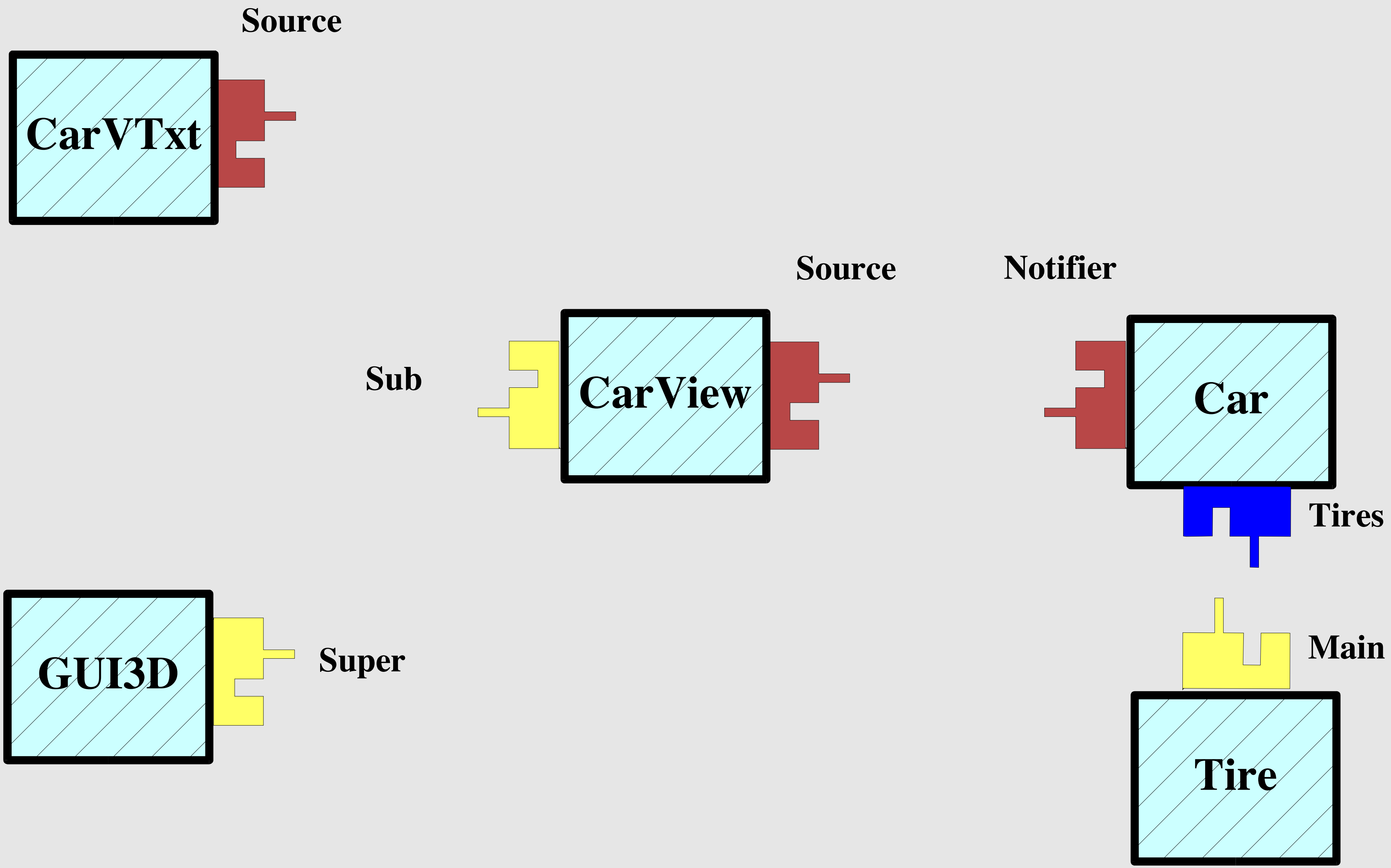
```
classage CarVTxt = GUITxt + CarView
          with Super >> Sub
```



Mixing: the Class Interaction

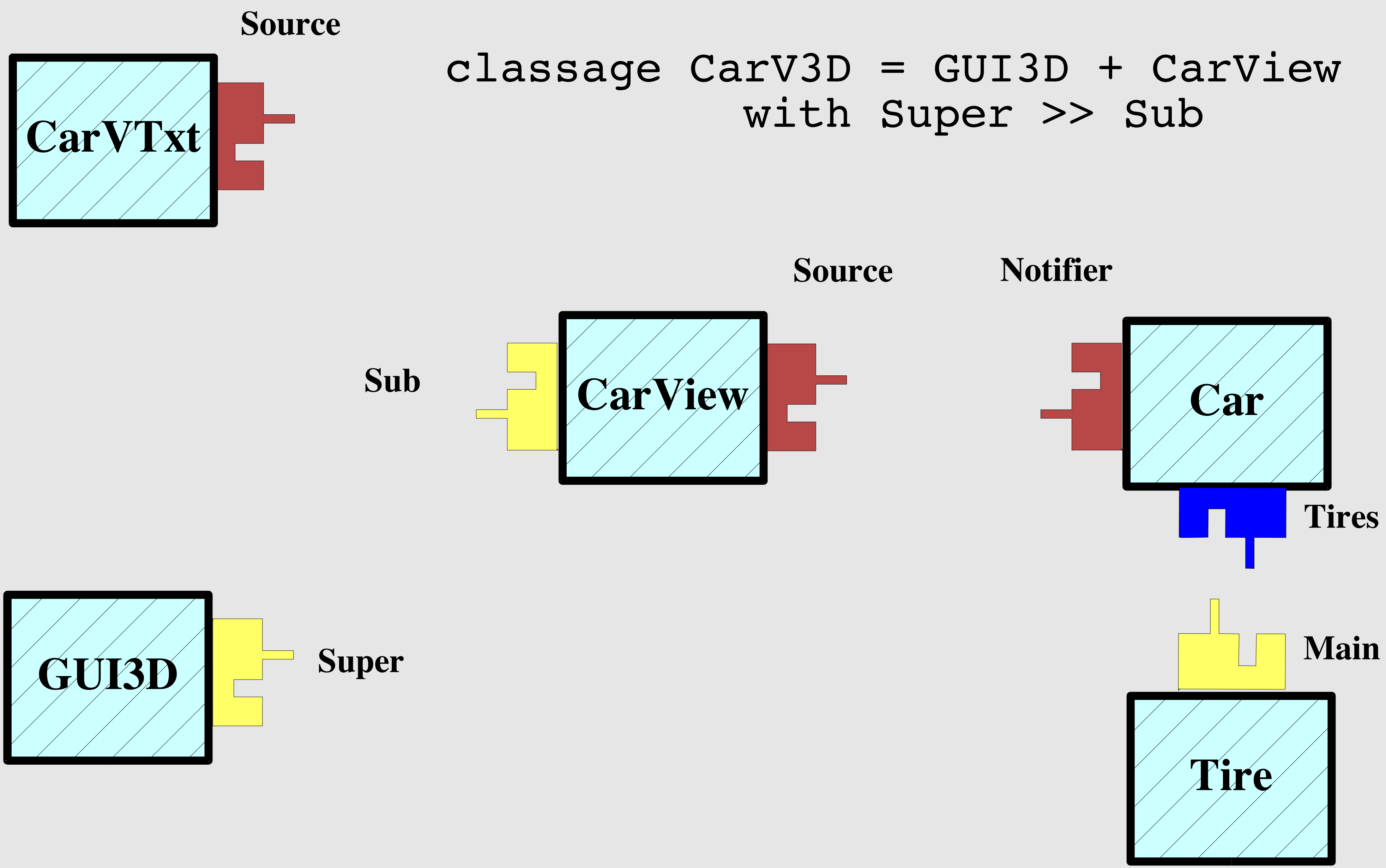


Mixing: the Class Interaction

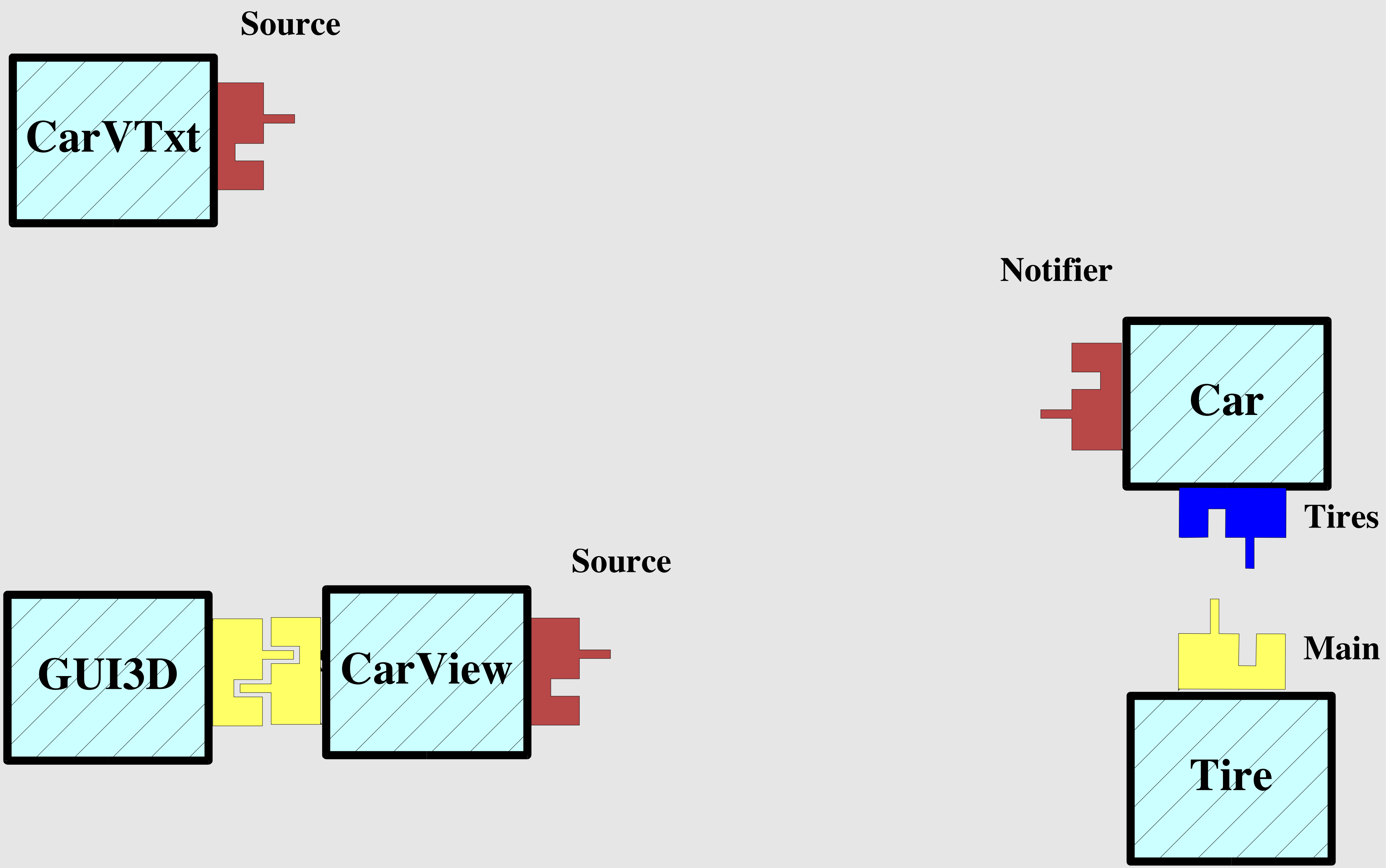


Mixing: the Class Interaction

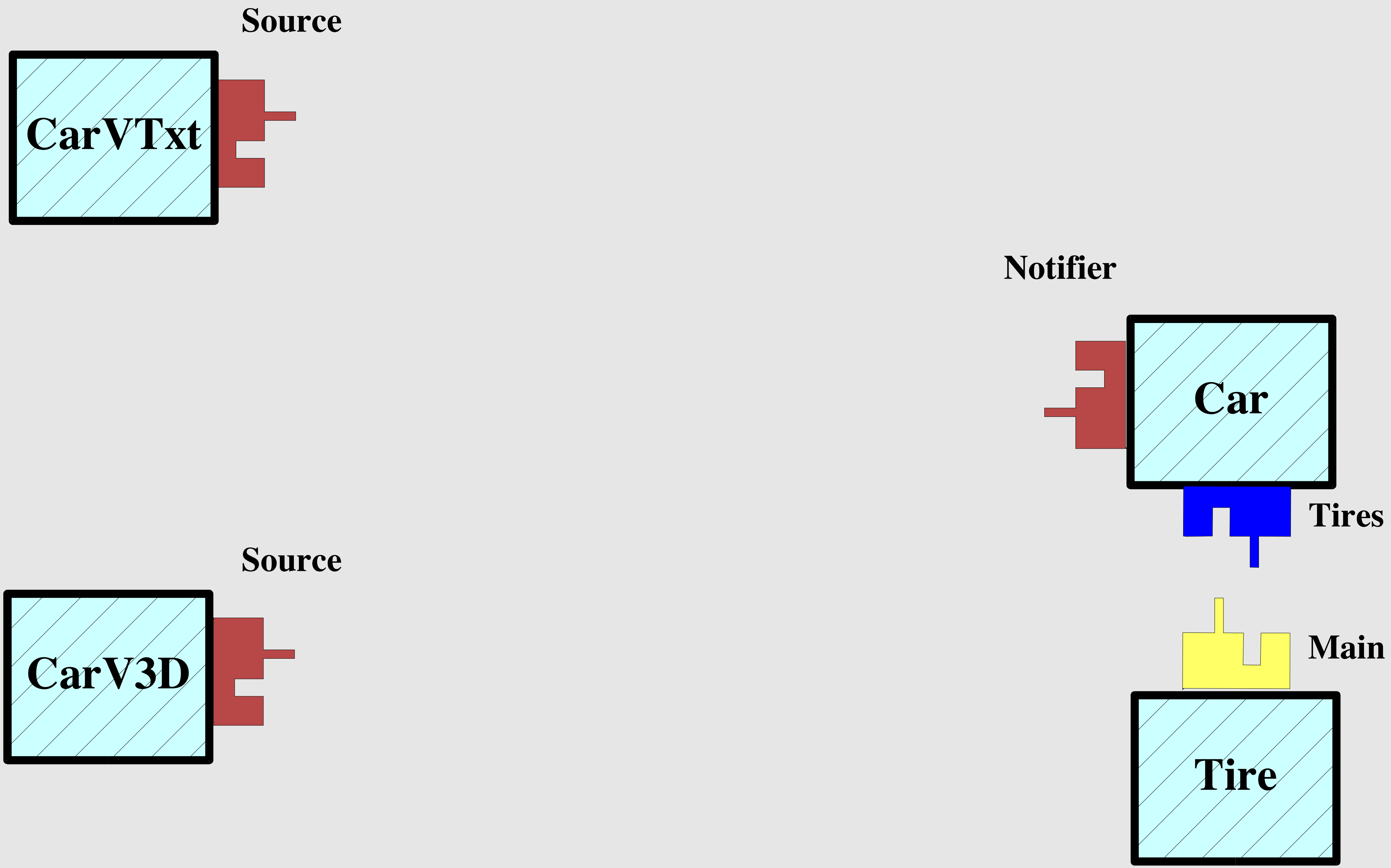
```
classage CarV3D = GUI3D + CarView
          with Super >> Sub
```



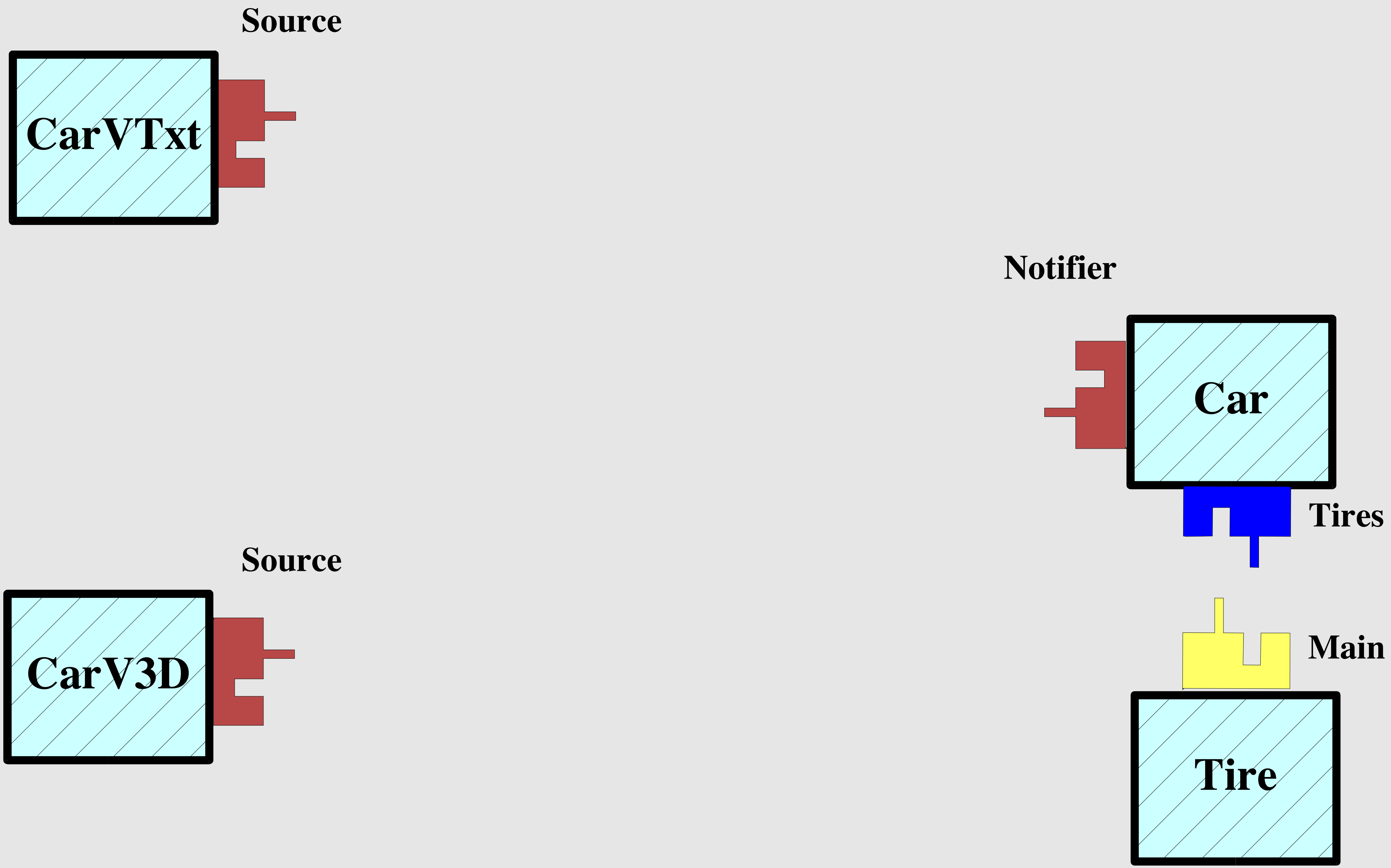
Mixing: the Class Interaction



Mixing: the Class Interaction

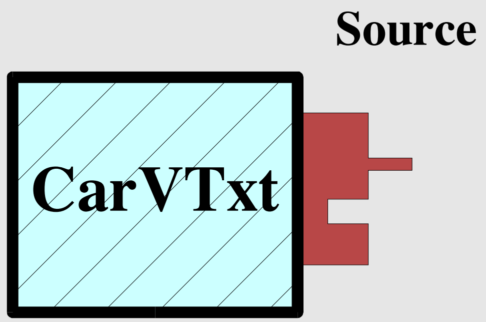


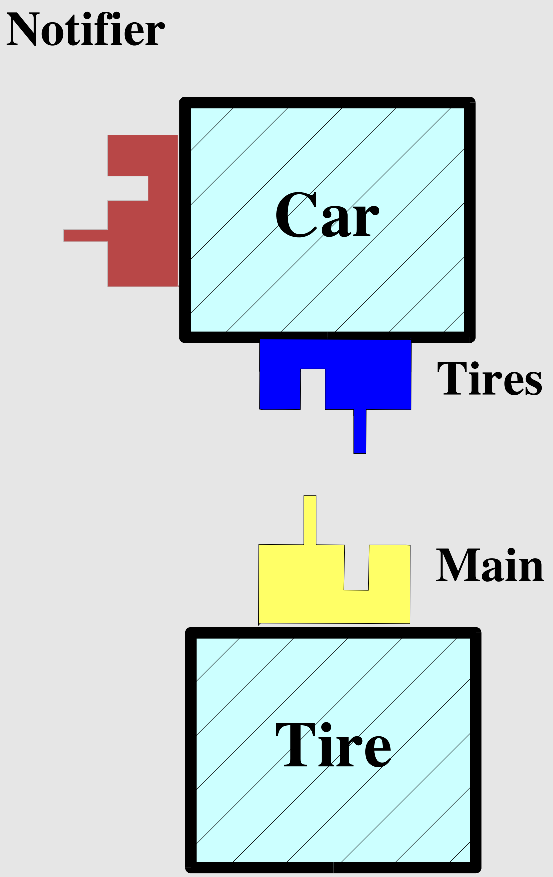
Now the program is running...

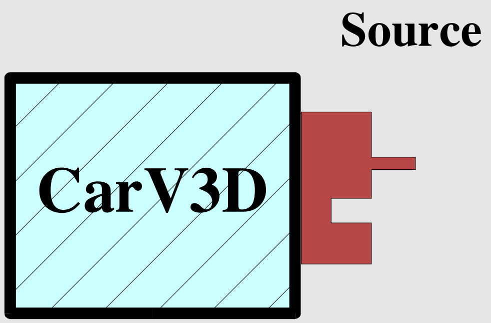


Objectage Instantiation

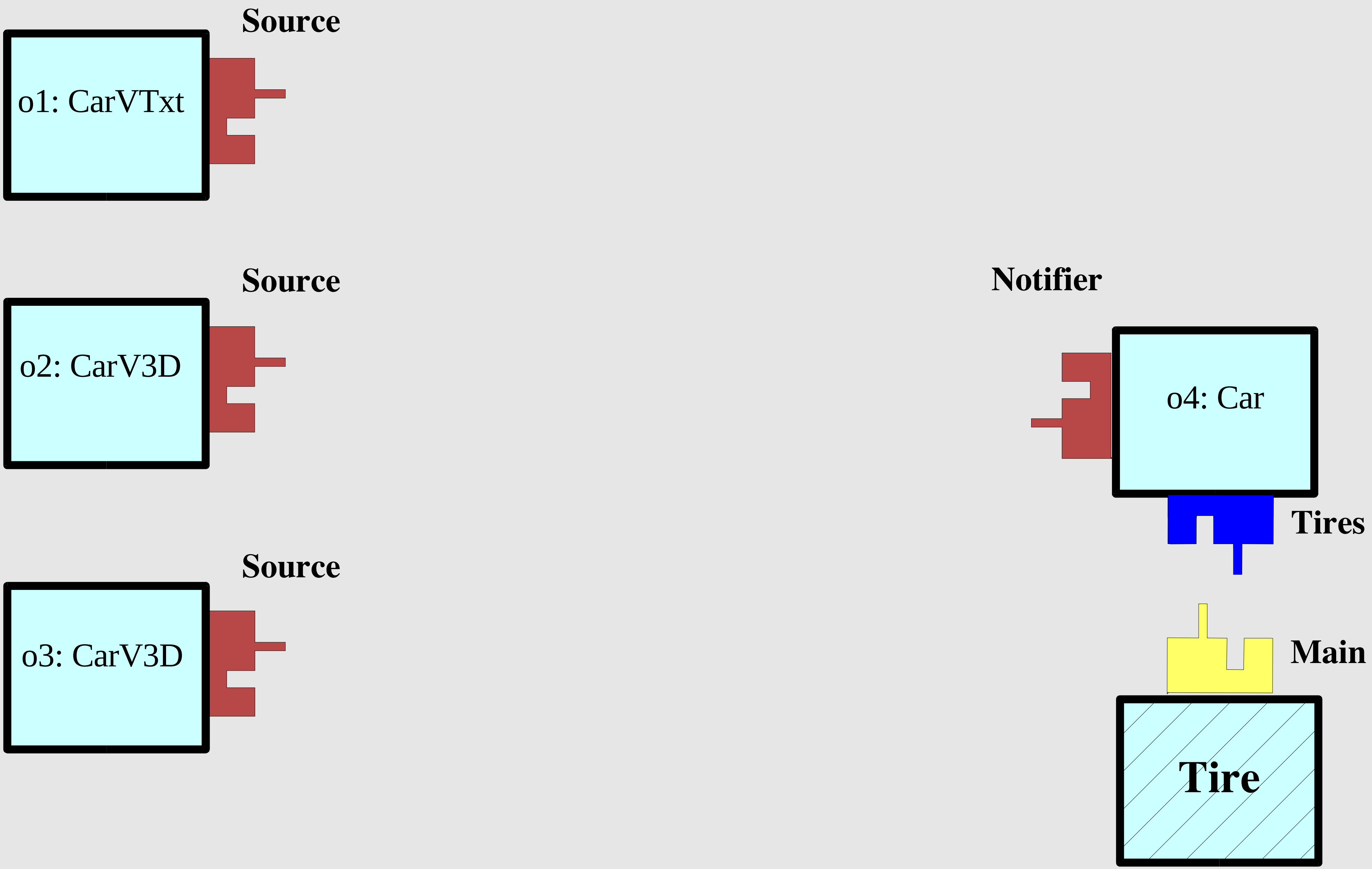
```
o1 = create CarVTxt();
o2 = create CarV3D();
o3 = create CarV3D();
o4 = create Car();
```



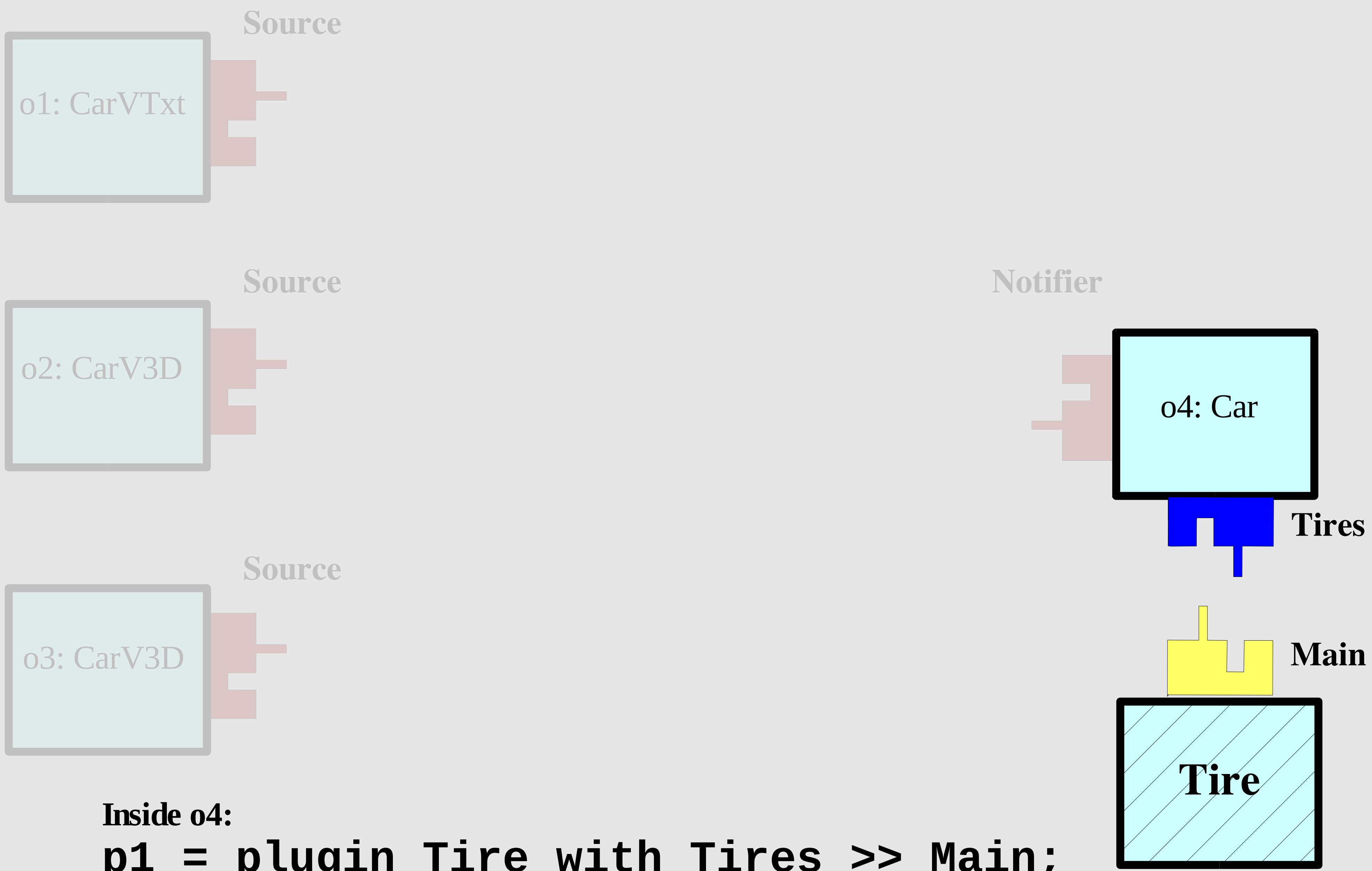




Objectage Instantiation



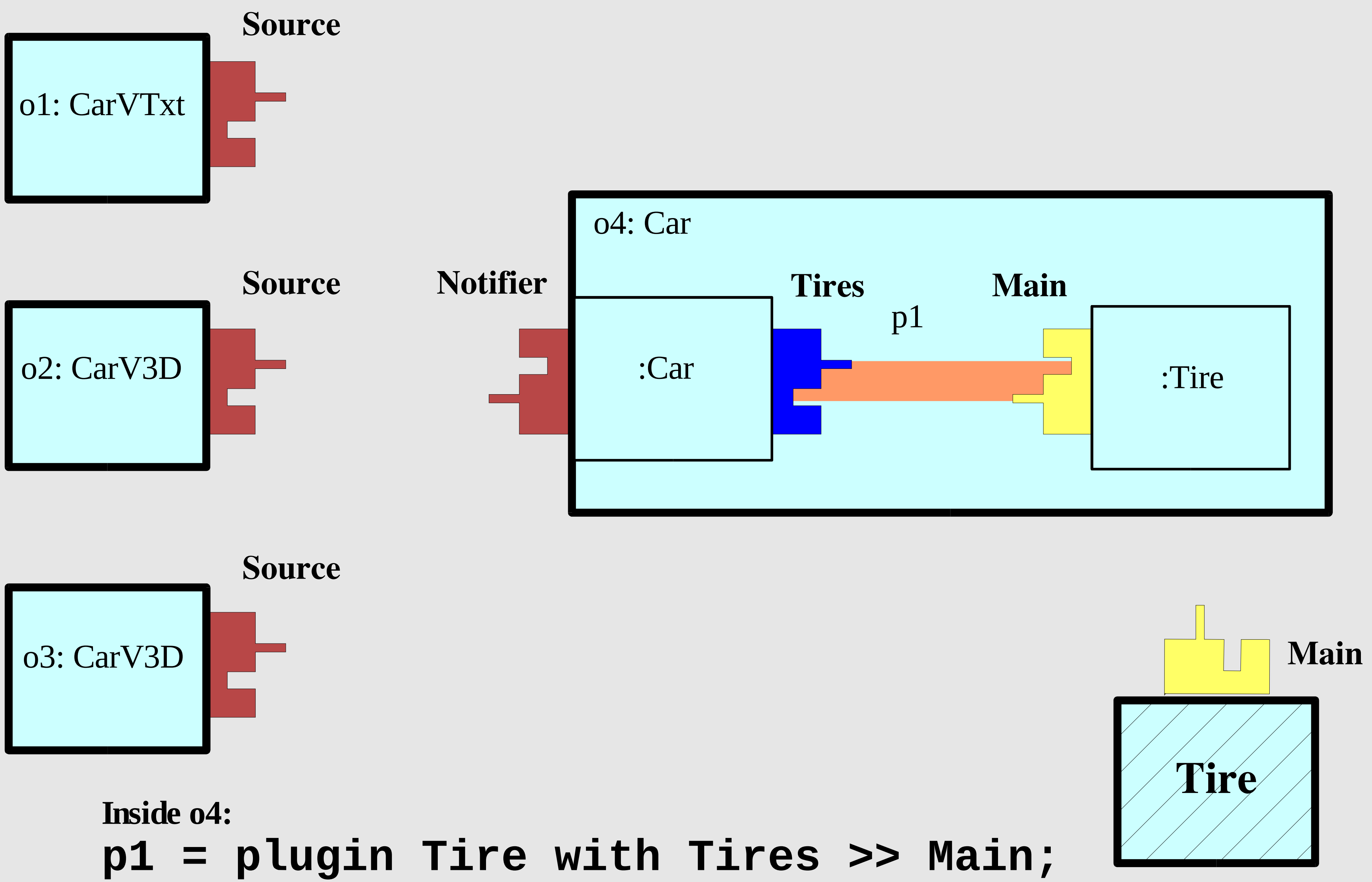
Plugging: Whole-Part Interaction



Inside o4:

```
p1 = plugin Tire with Tires >> Main;
```

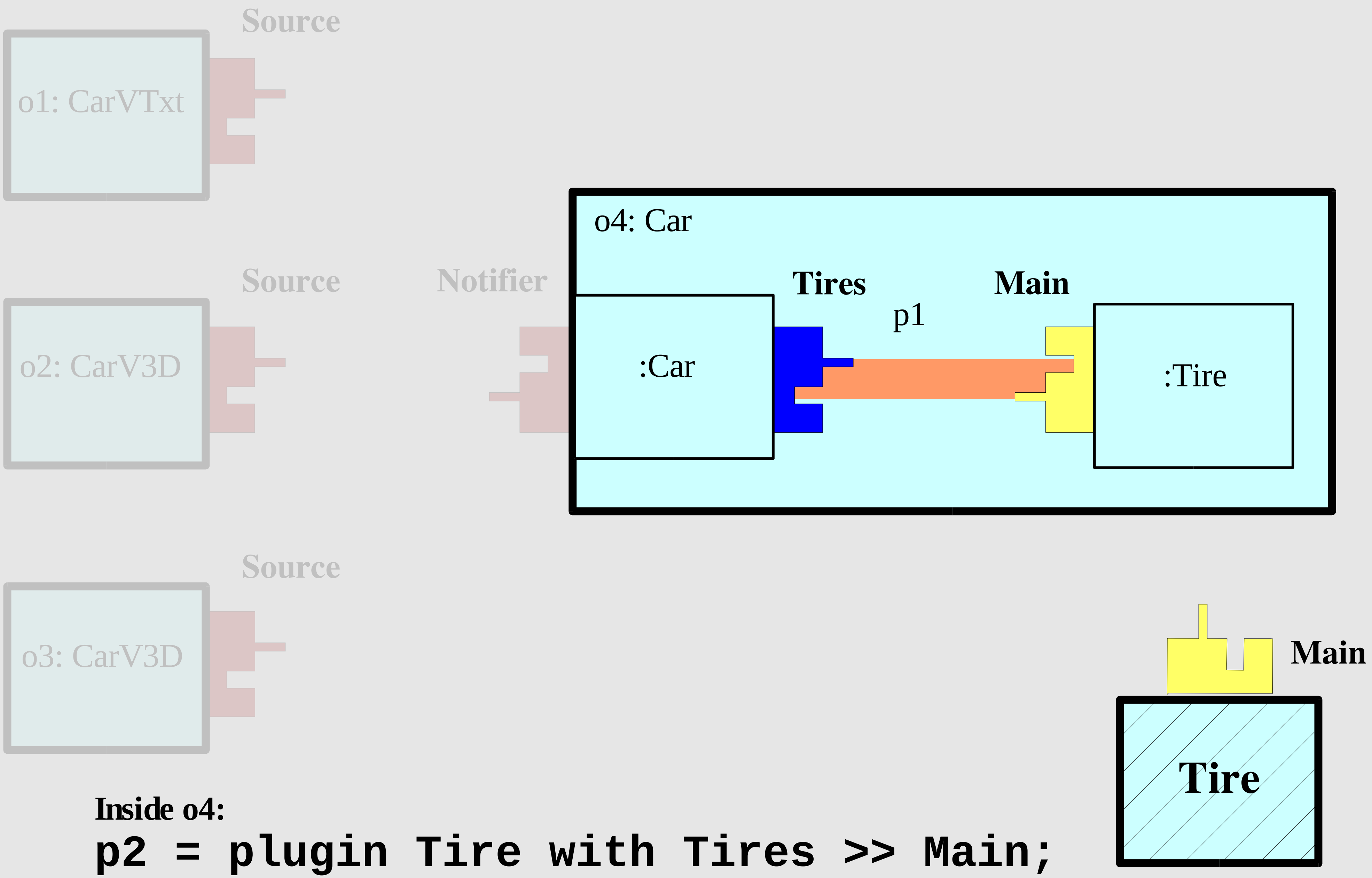
Plugging: Whole-Part Interaction



Inside o4:

```
p1 = plugin Tire with Tires >> Main;
```

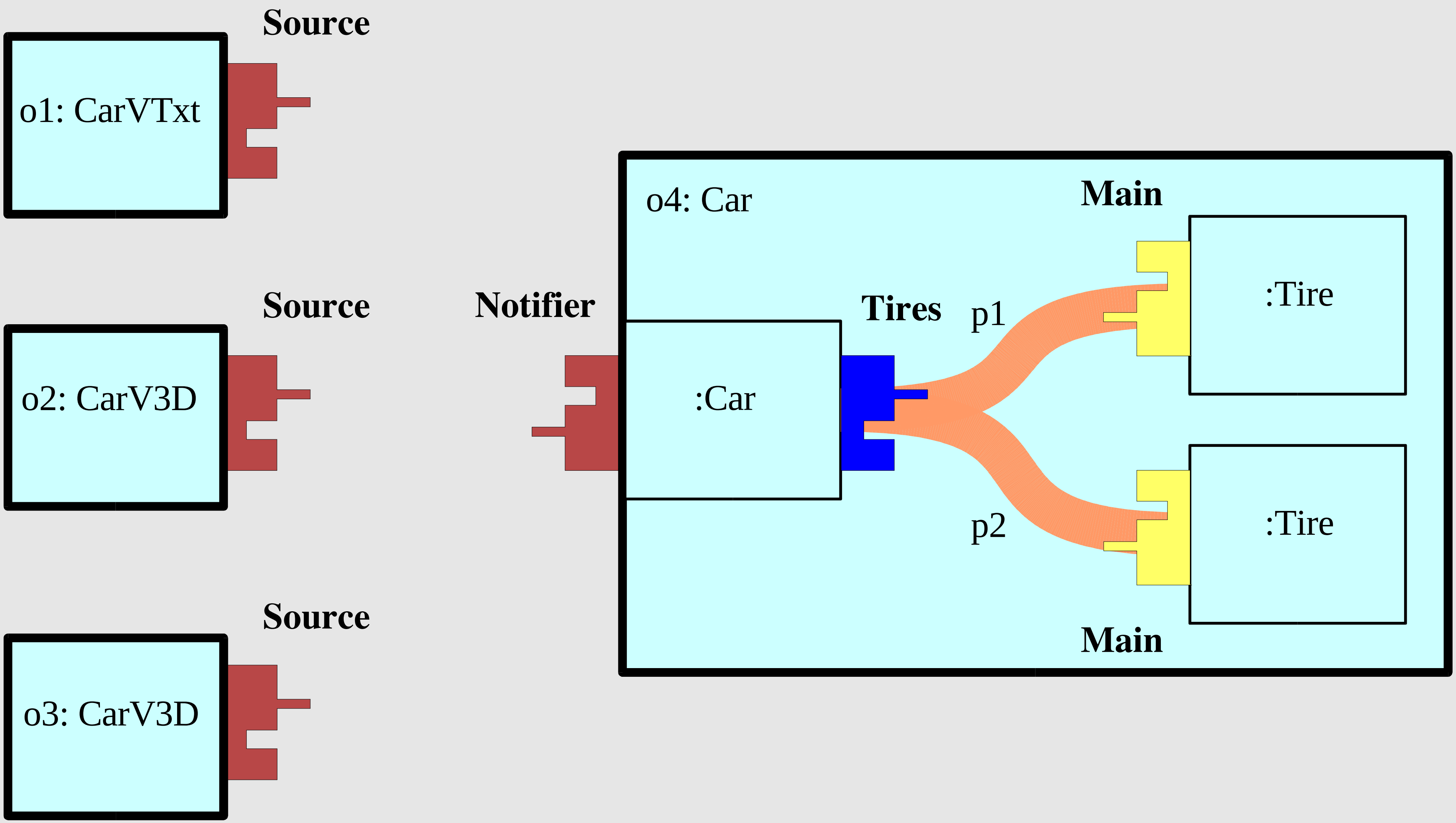
Plugging: Whole-Part Interaction



Inside o4:

```
p2 = plugin Tire with Tires >> Main;
```

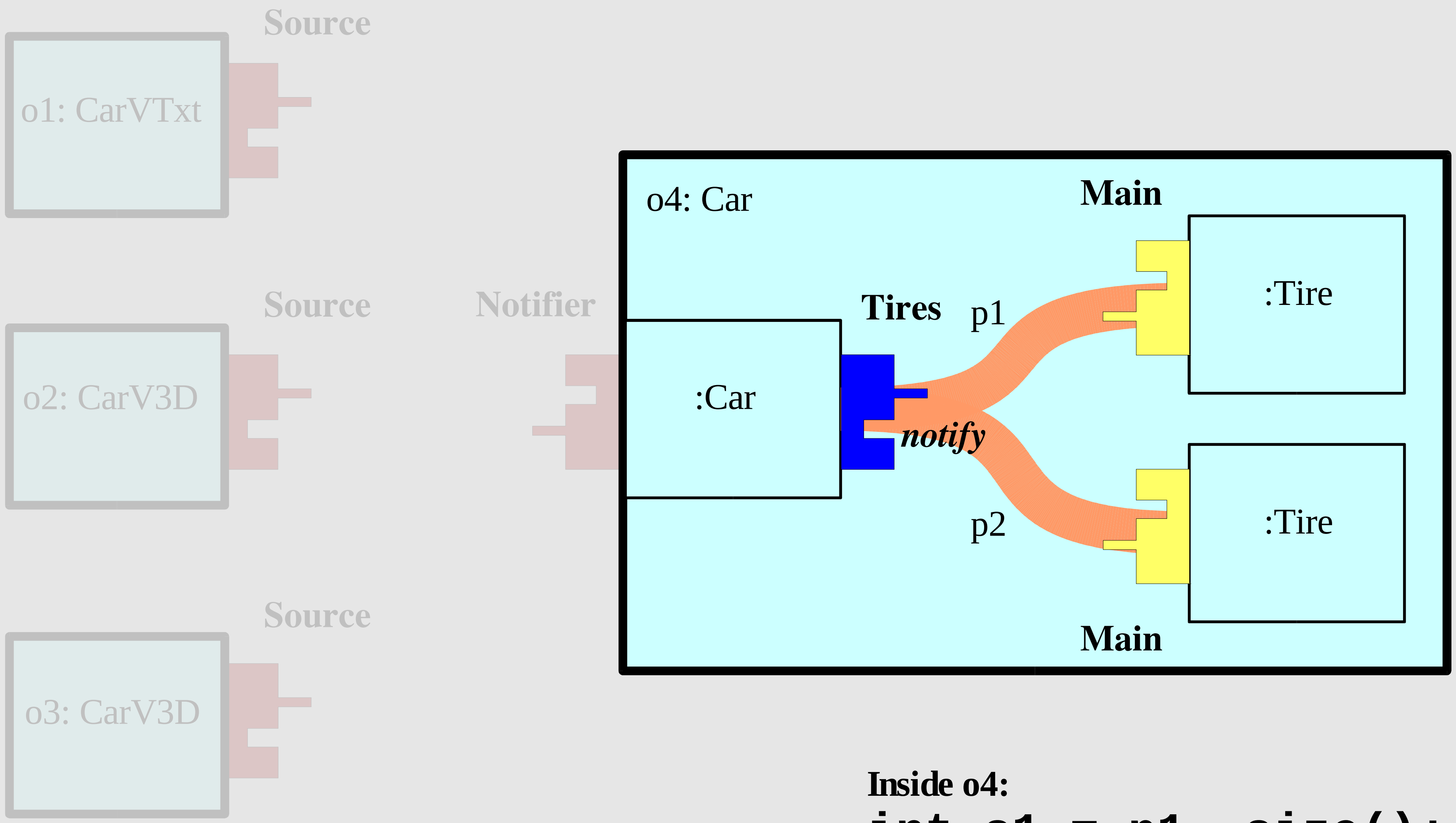
Plugging: Whole-Part Interaction



Inside o4:

```
p2 = plugin Tire with Tires >> Main;
```

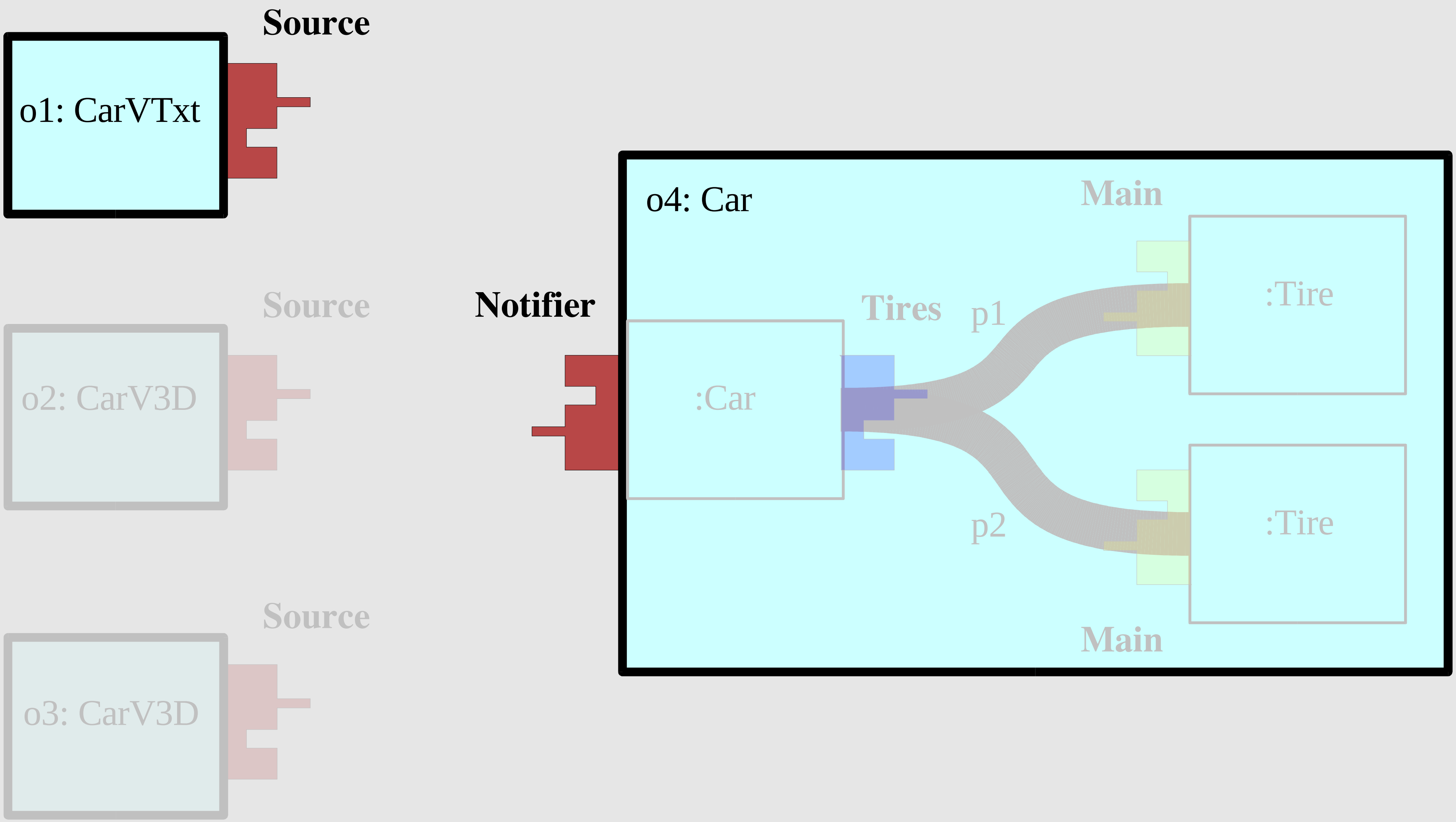
Plugging Handles



Inside o4:

```
int s1 = p1..size();
int s2 = p2..size();
```

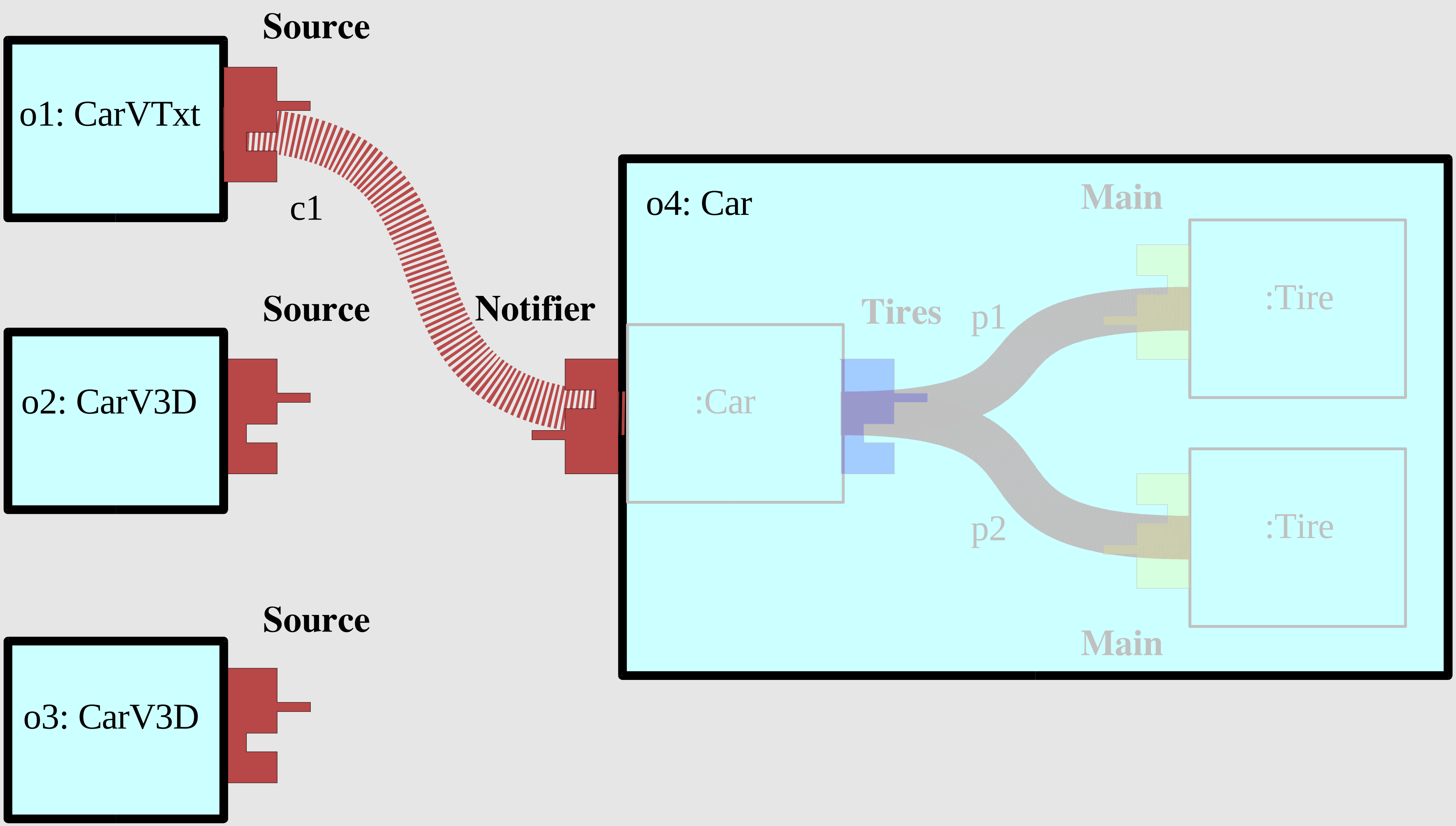
Connection: Peer-to-Peer Interaction



Inside o1:

```
c1 = connect o4 with Source >> Notifier;
```

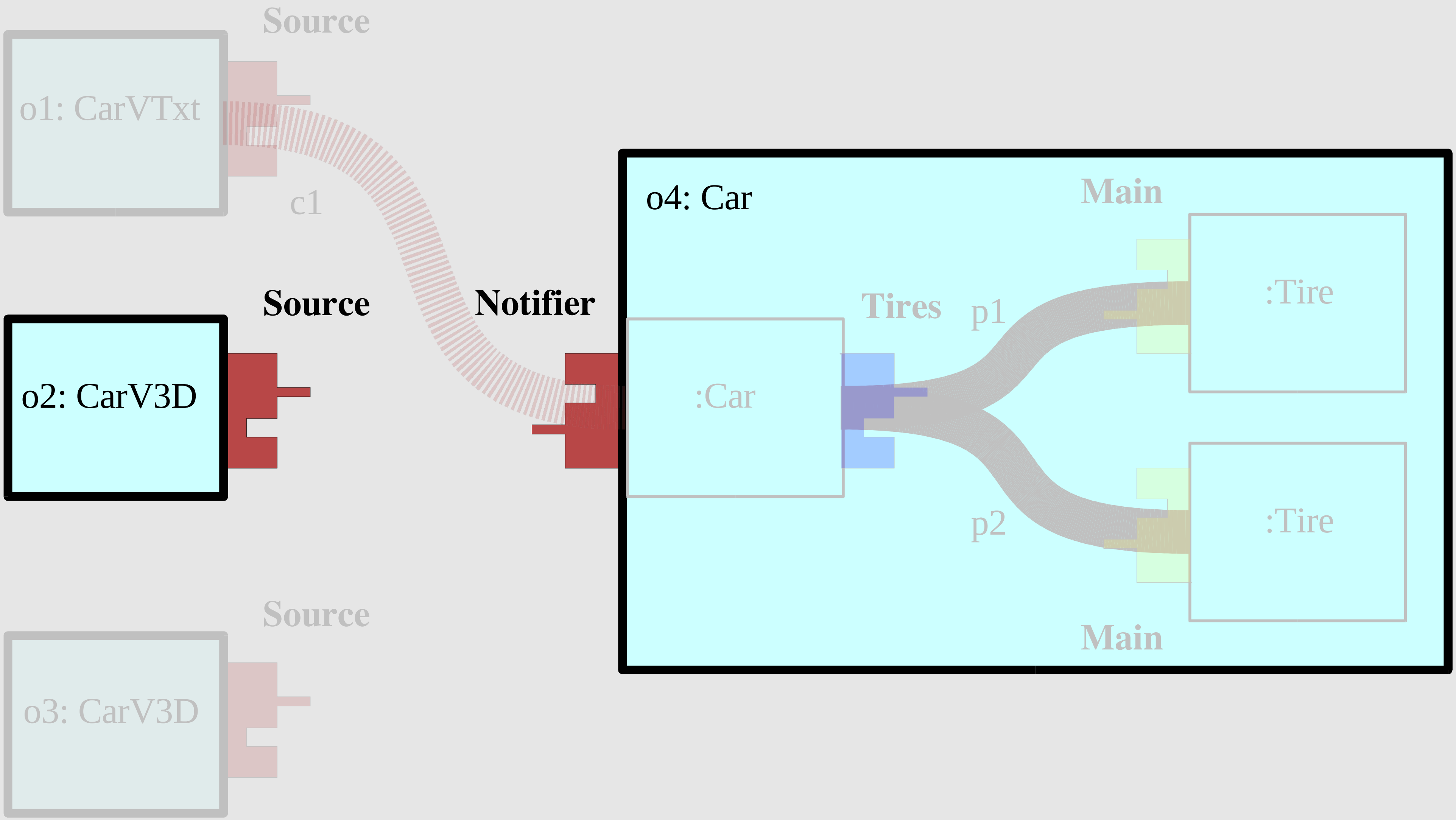
Connection Established



Inside o1:

```
c1 = connect o4 with Source >> Notifier;
```

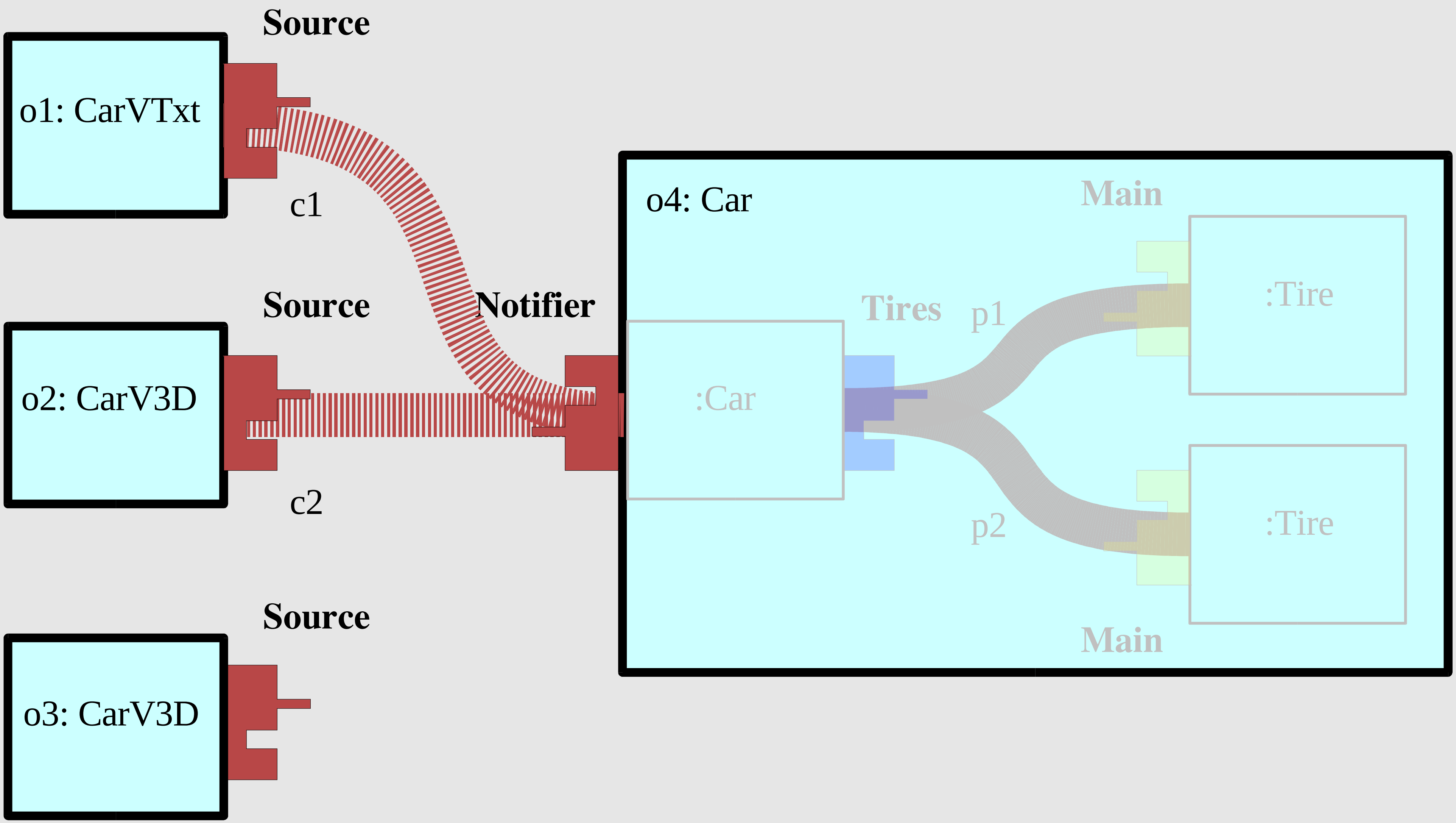
More Connections



Inside o2:

```
c2 = connect o4 with Source >> Notifier;
```

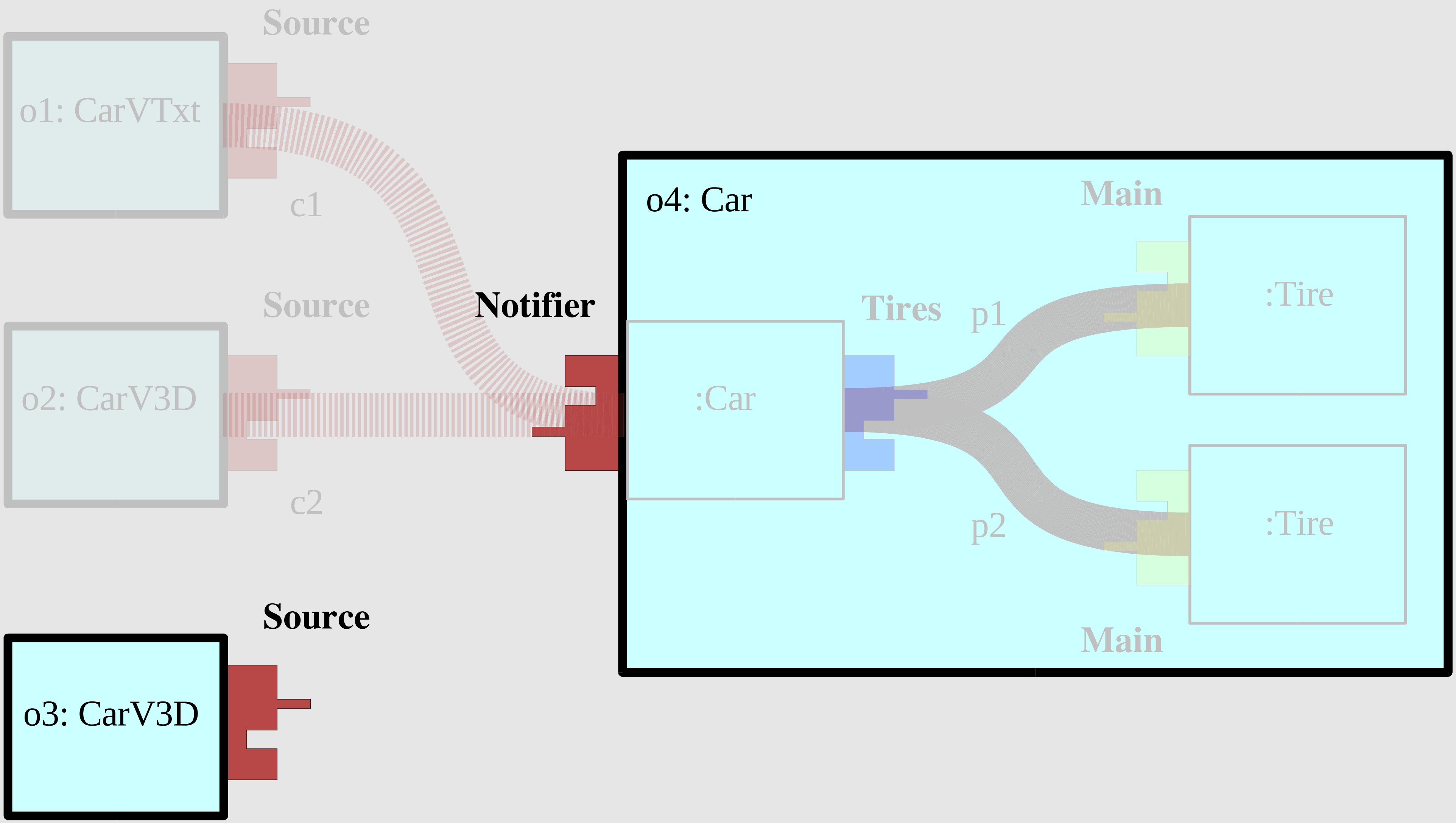
More Connections



Inside o2:

```
c2 = connect o4 with Source >> Notifier;
```

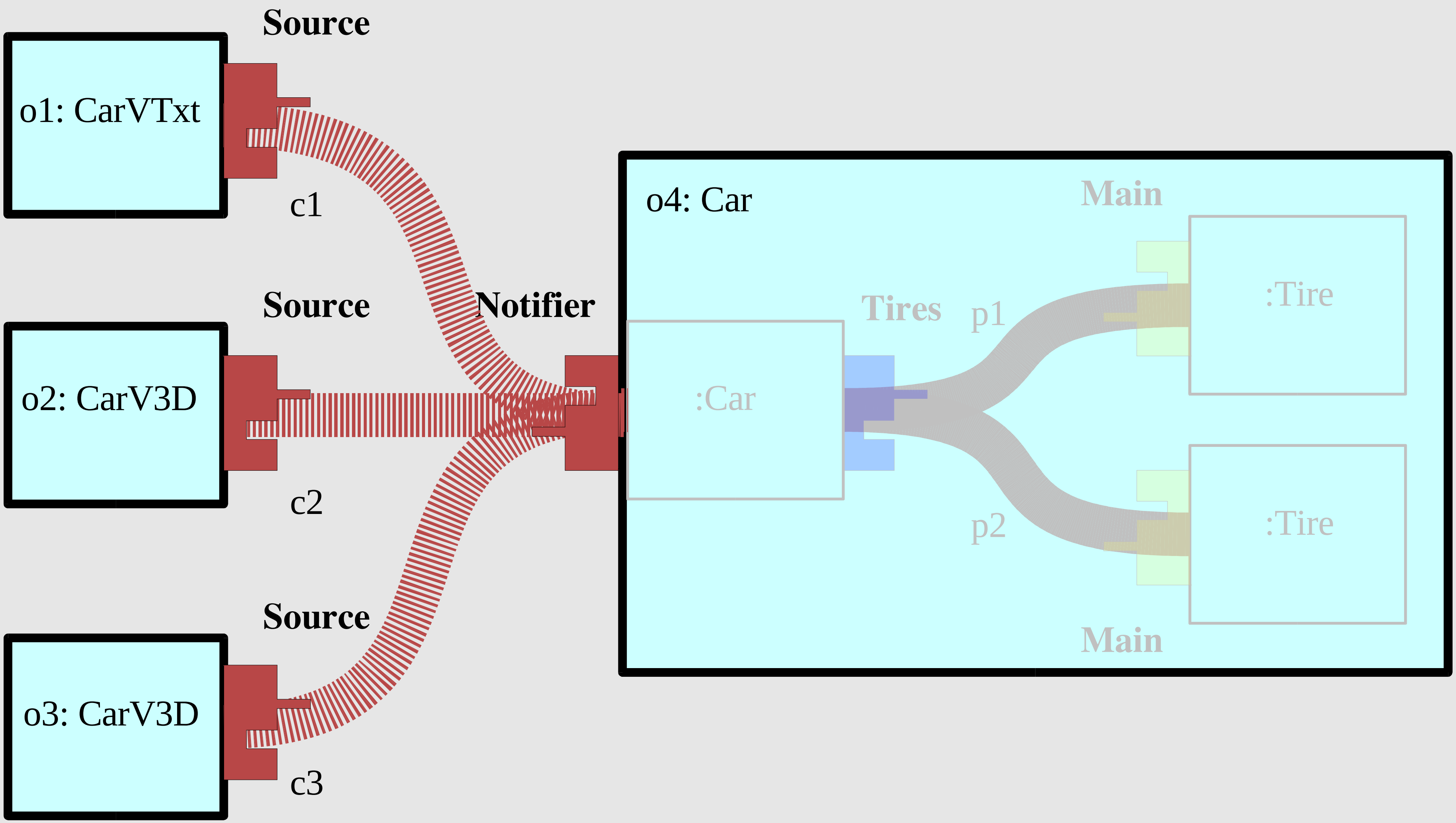
More Connections



Inside o3:

```
c3 = connect o4 with Source >> Notifier;
```

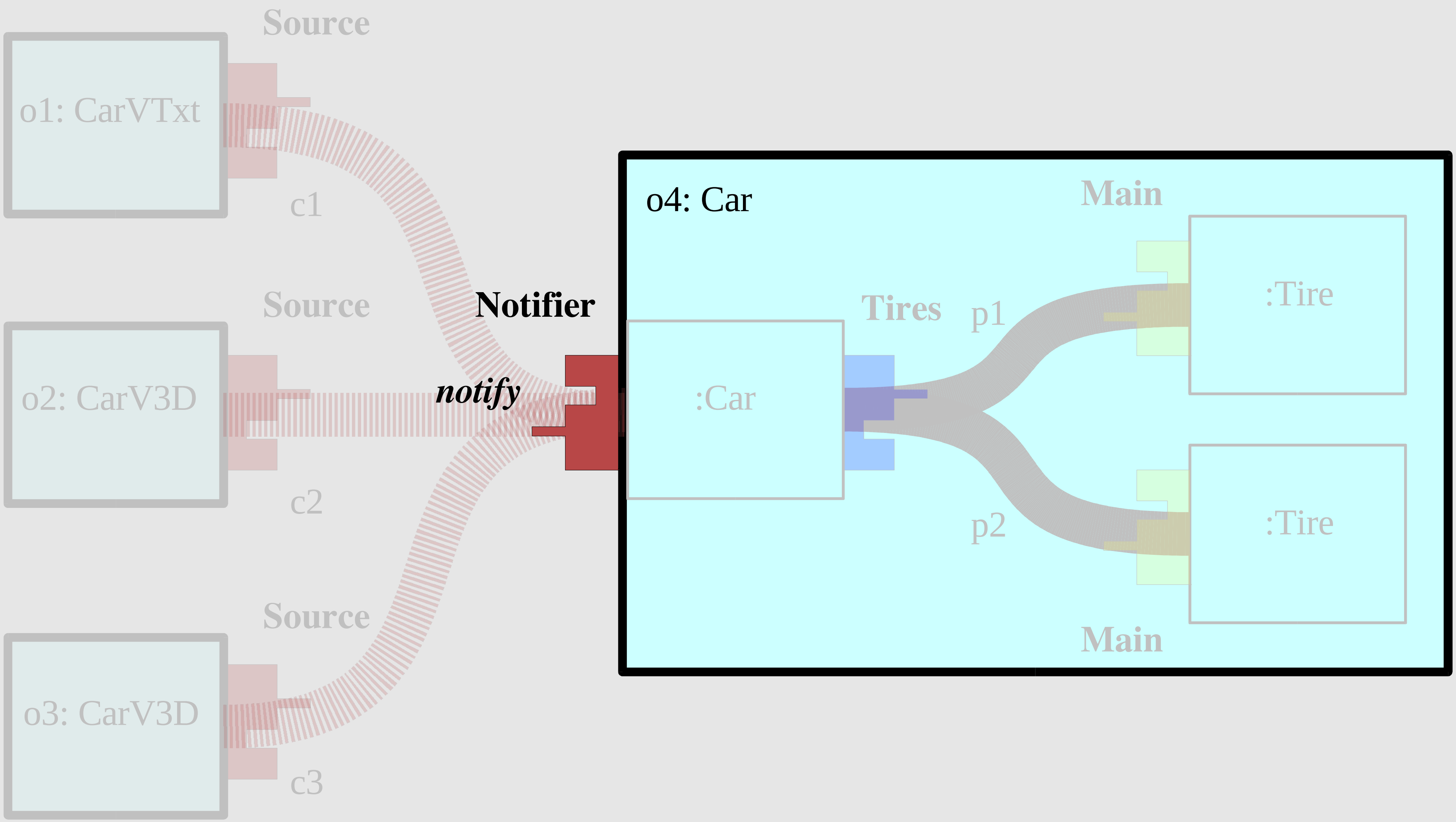
More Connections



Inside o3:

```
c3 = connect o4 with Source >> Notifier;
```

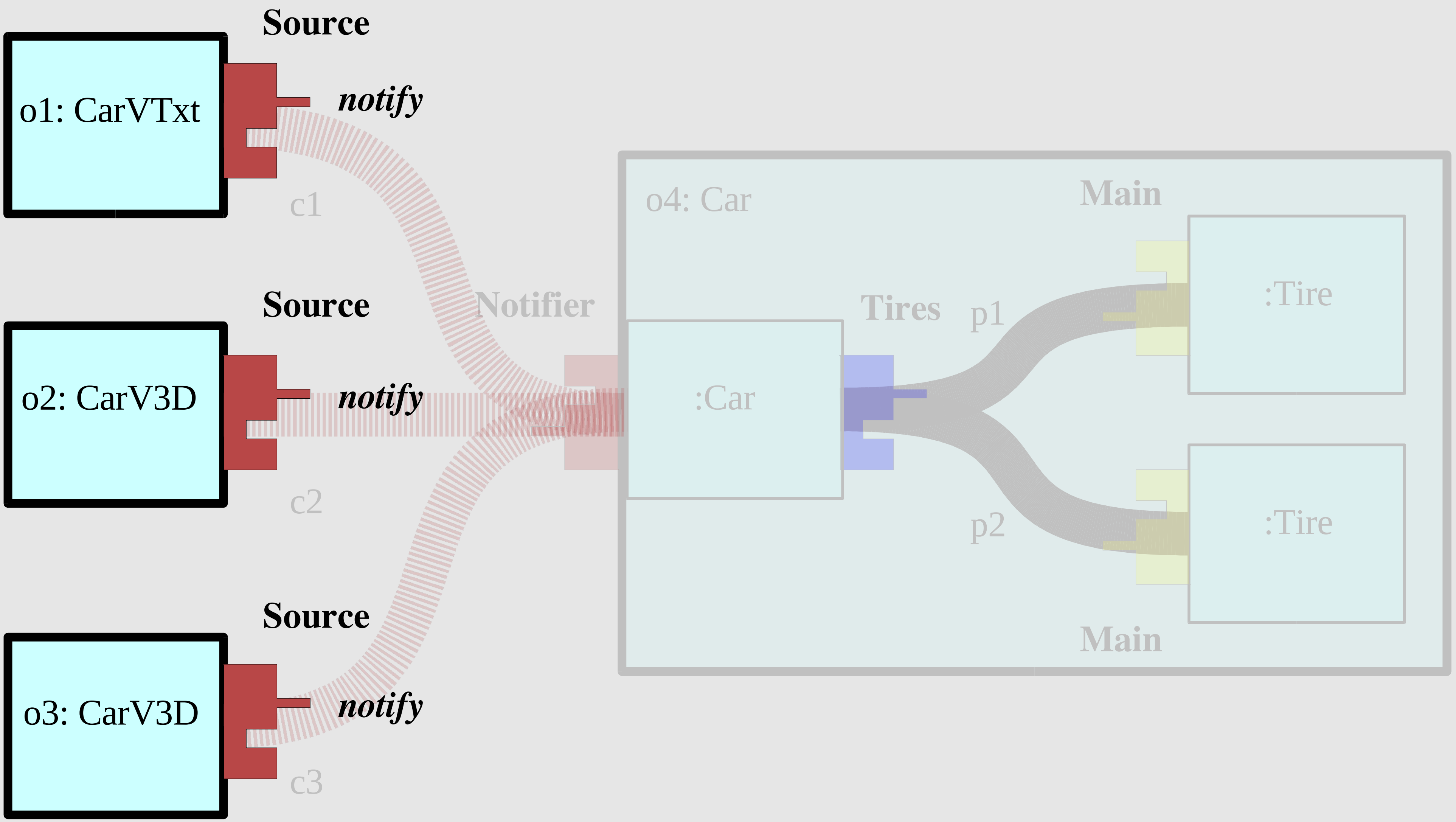
For All Connections



Inside o4:

```
forall(c: Notifier) {c->notify(); }
```

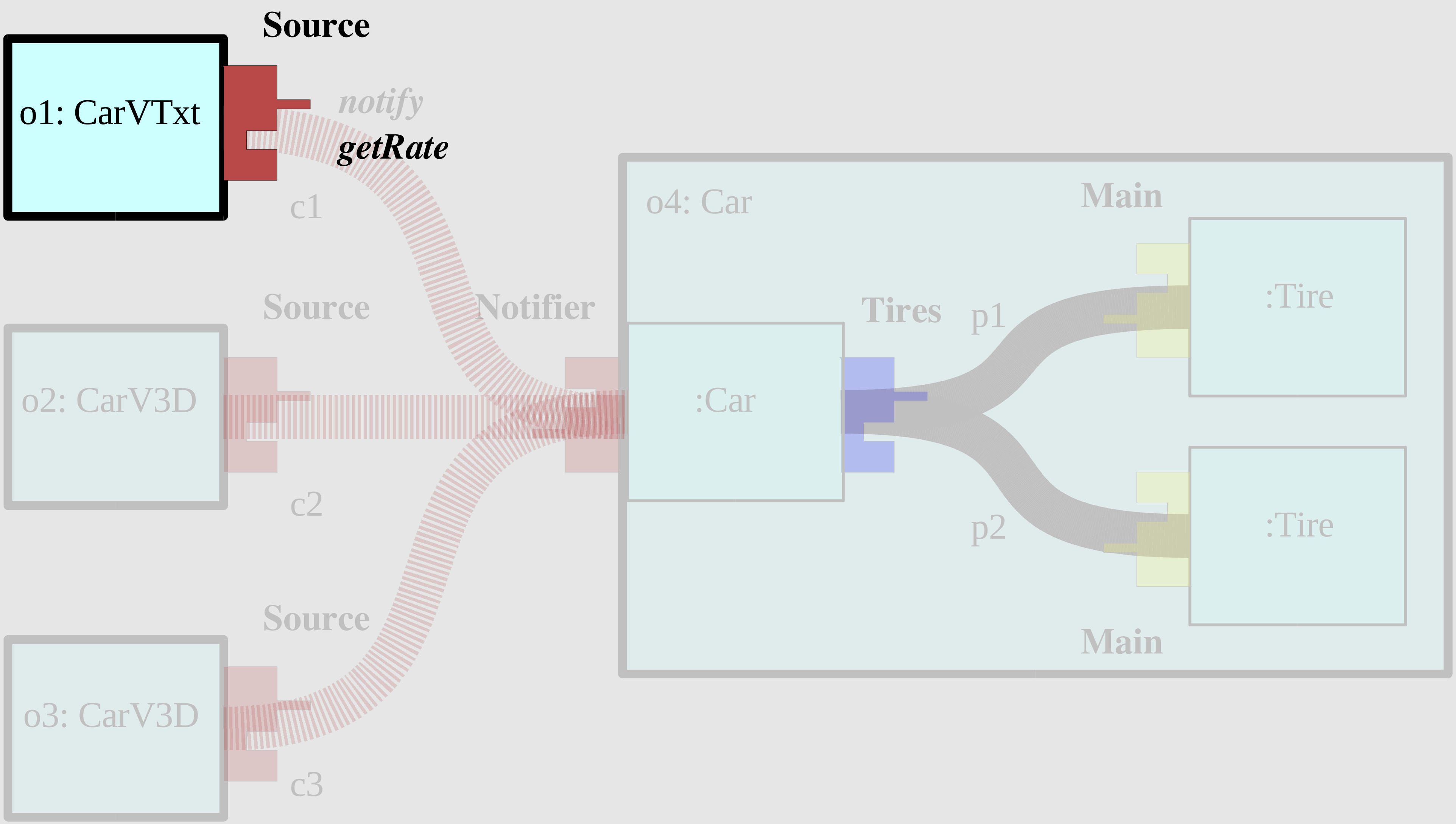
For All Connections



Inside o4:

```
forall(c: Notifier) {c->notify(); }
```

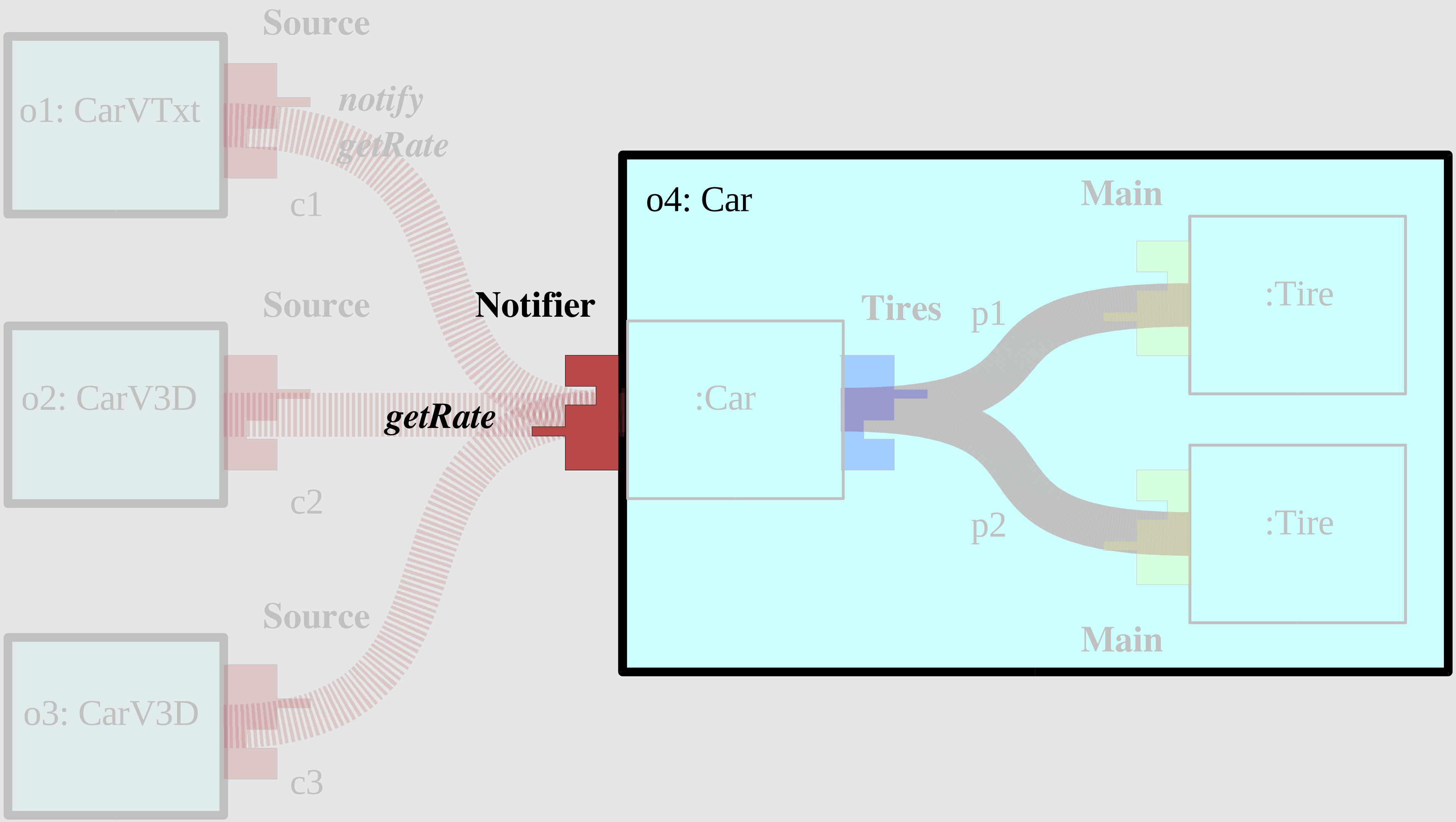
Callbacks



Inside *notify* of o1:

```
int r = getRate();
```

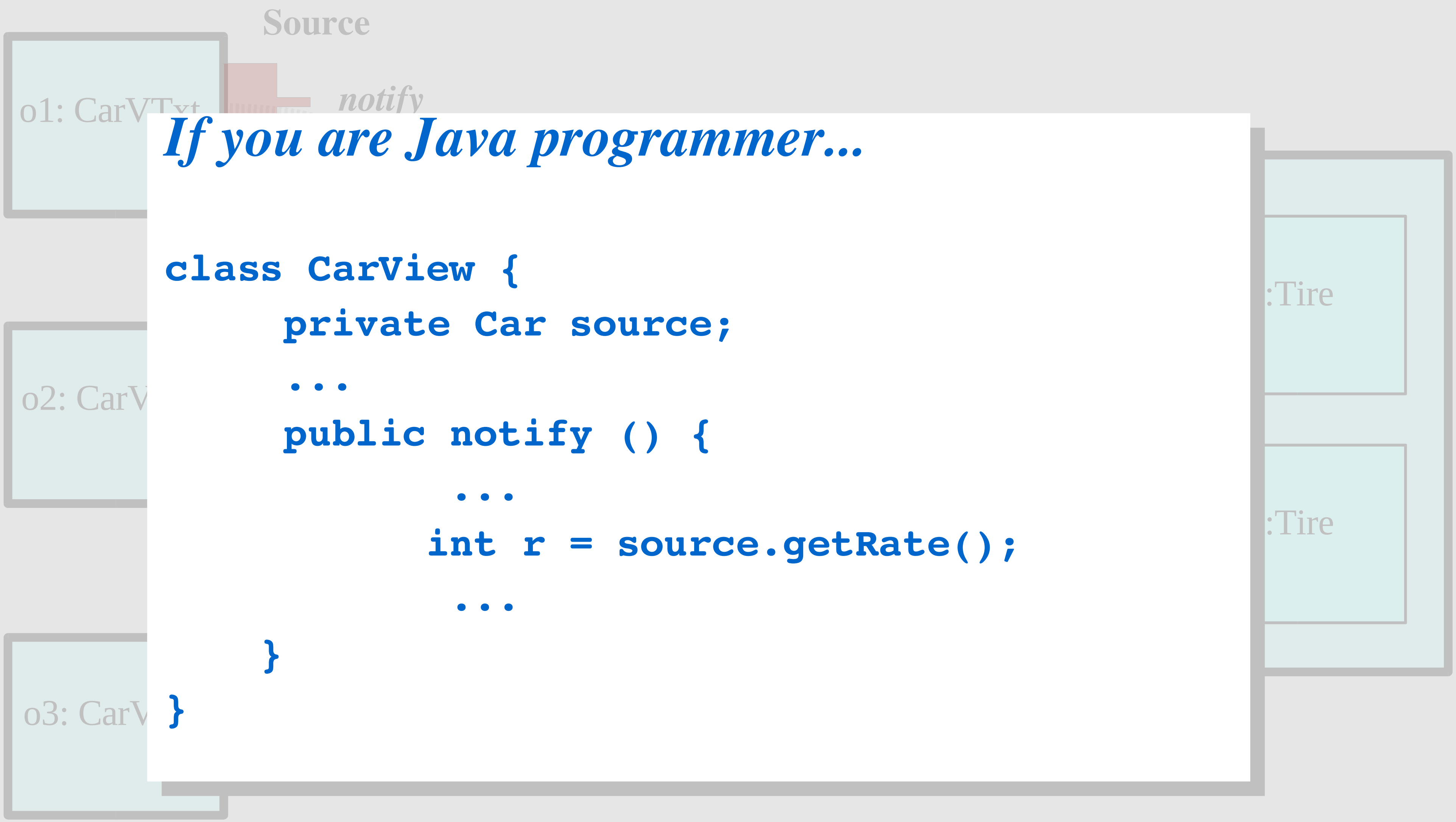
Callbacks



Inside *notify* of o1:

```
int r = getRate();
```

Callbacks



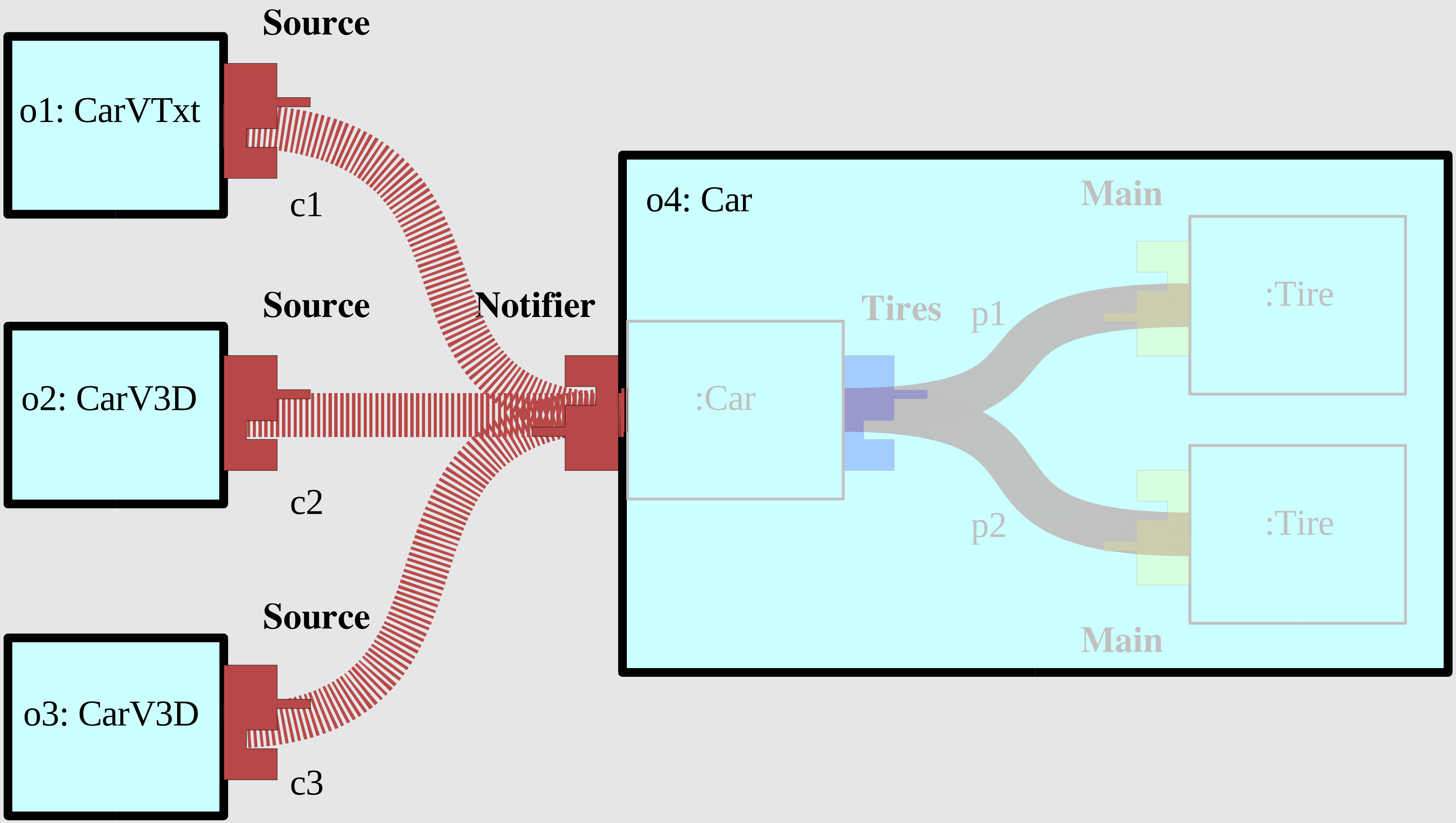
If you are Java programmer...

```
class CarView {
     private Car source;
     ...
     public notify () {
            ...
            int r = source.getRate();
            ...
     }
}
```

Inside *notify* of o1:

```
int r = getRate();
```

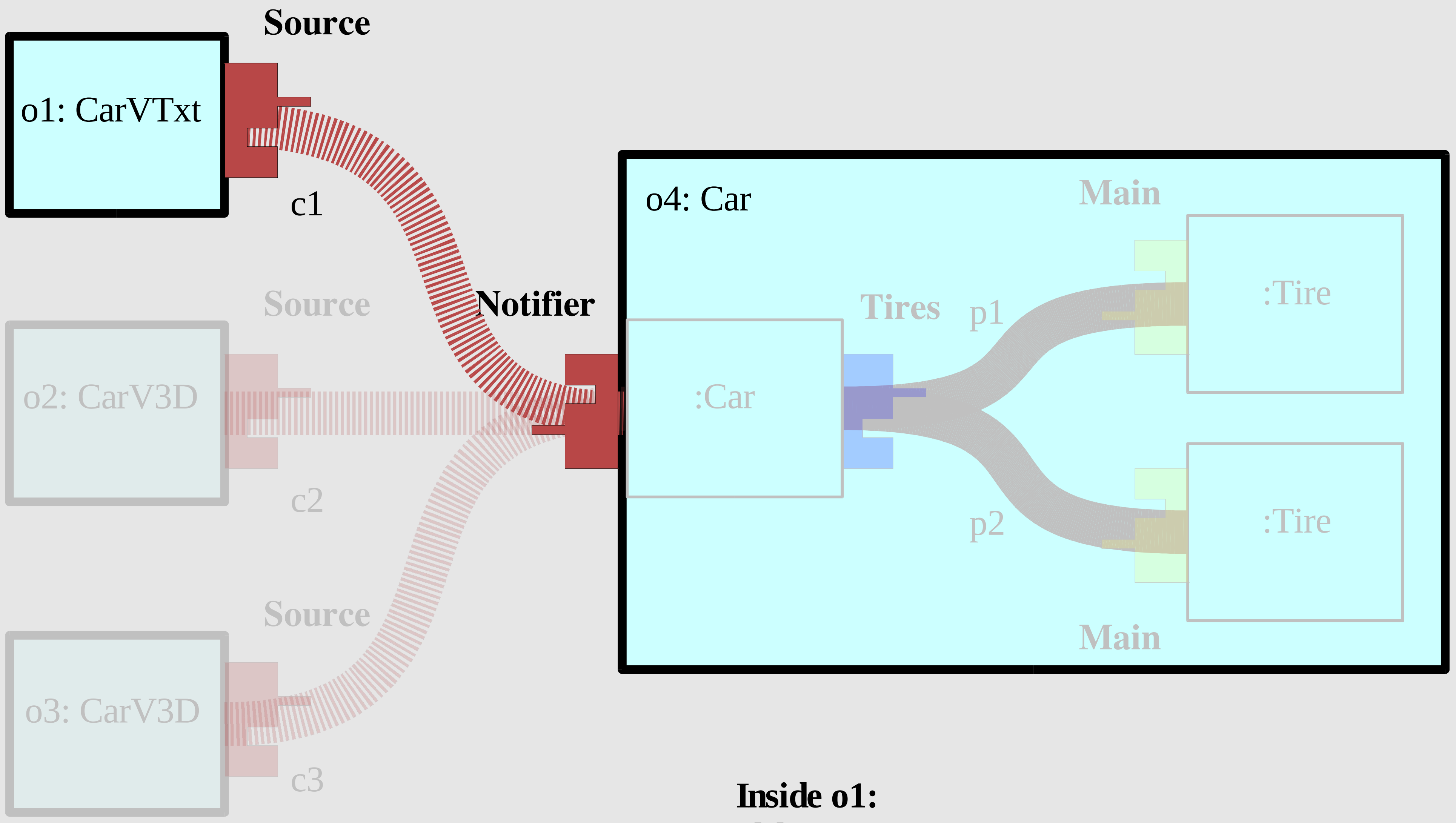
Stateful Connections



Inside *notify* of Source:

```
::counter = ::counter + 1;
```

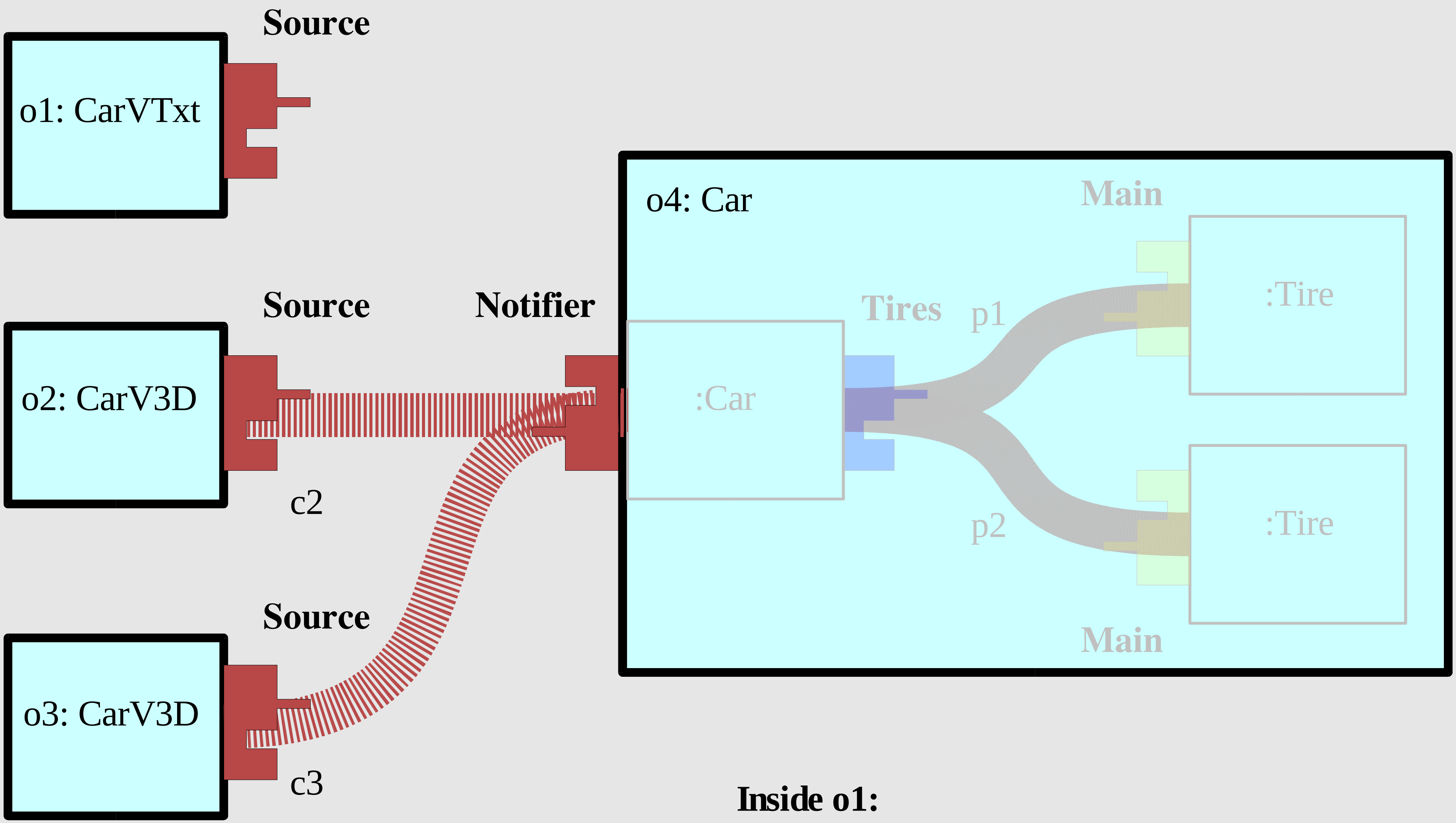
Disconnect



Inside o1:

```
disconnect c1;
```

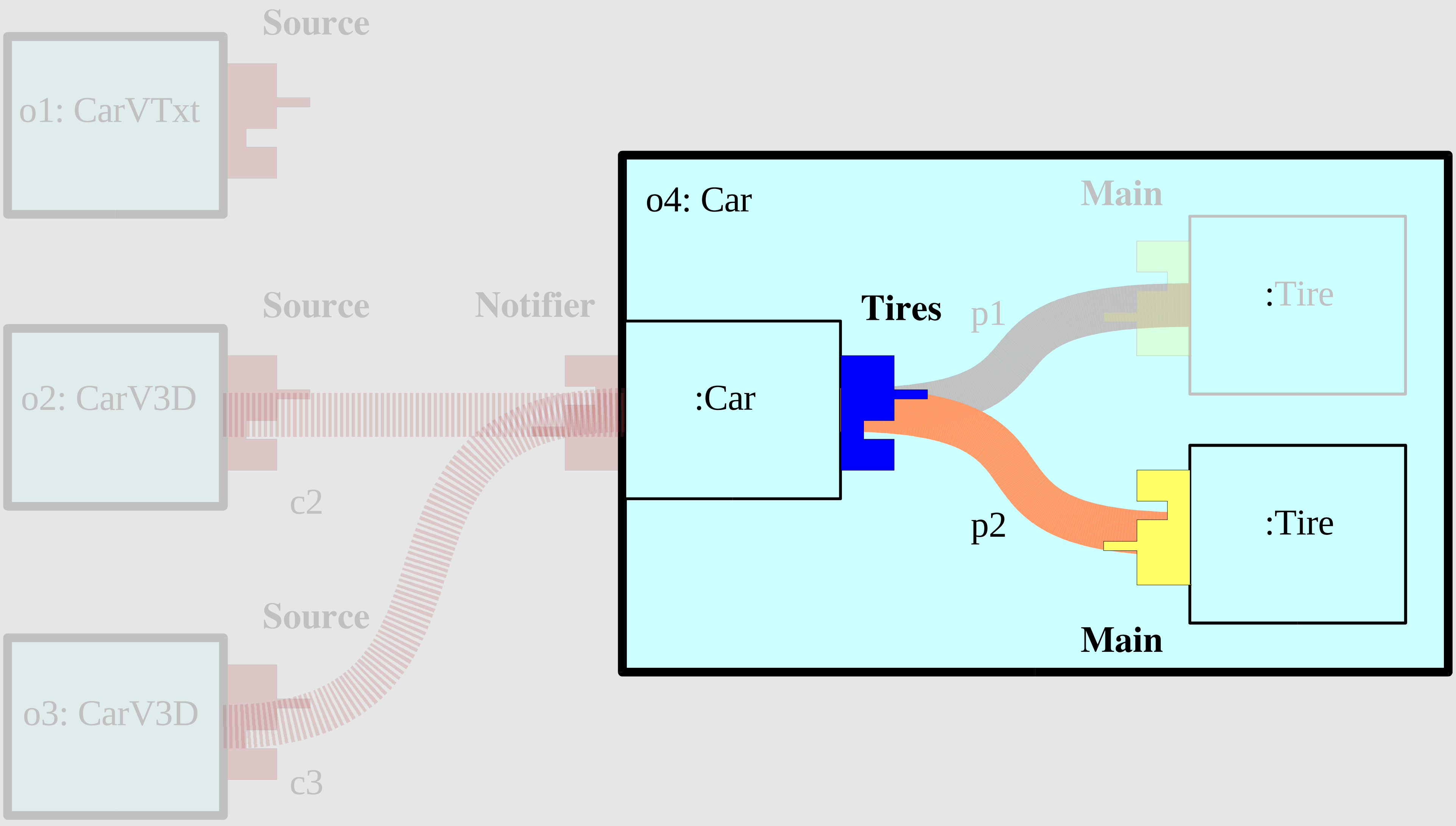
Disconnect



Inside o1:

```
disconnect c1;
```

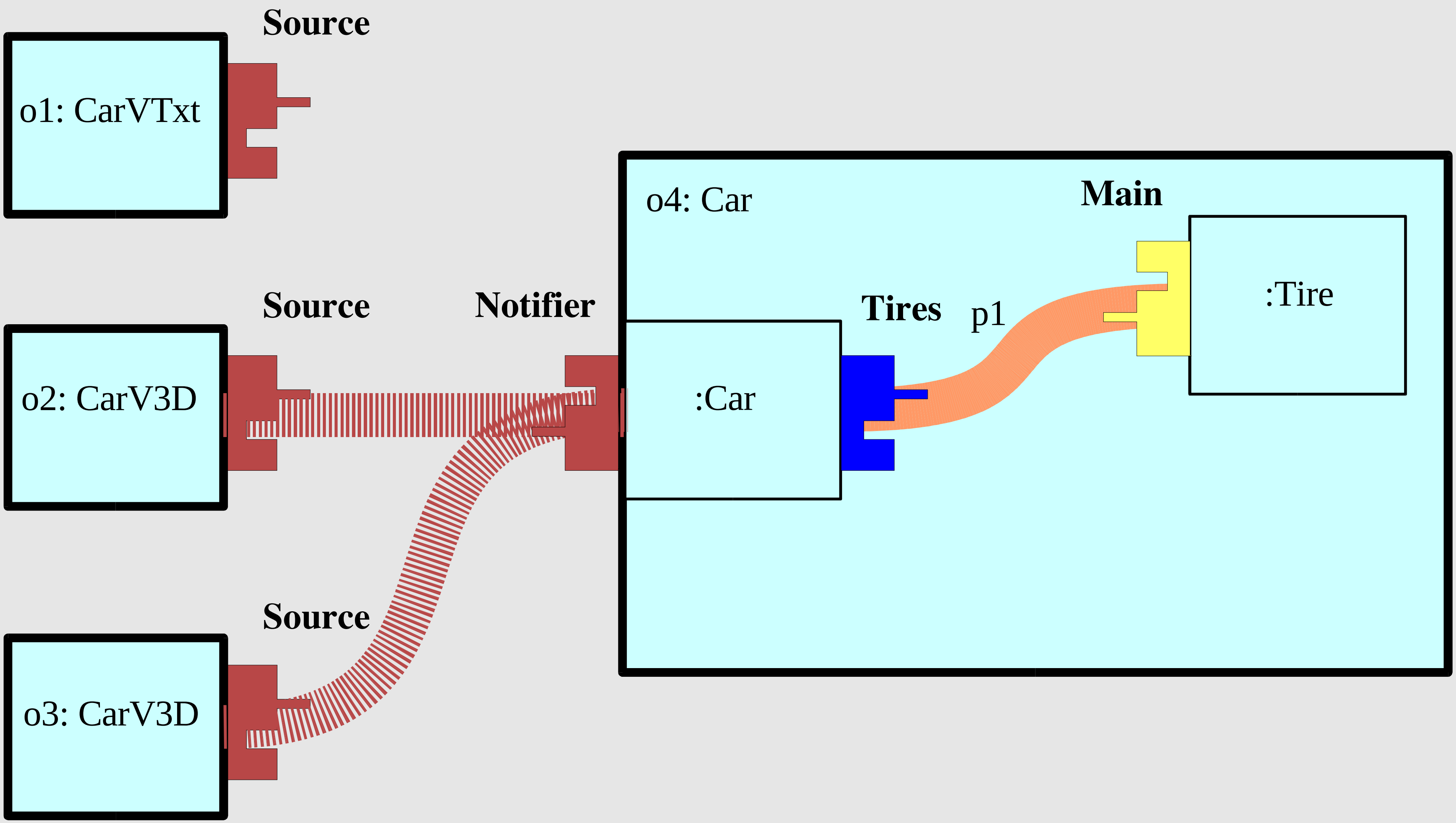
Unplug



Inside o4:

```
unplug p2;
```

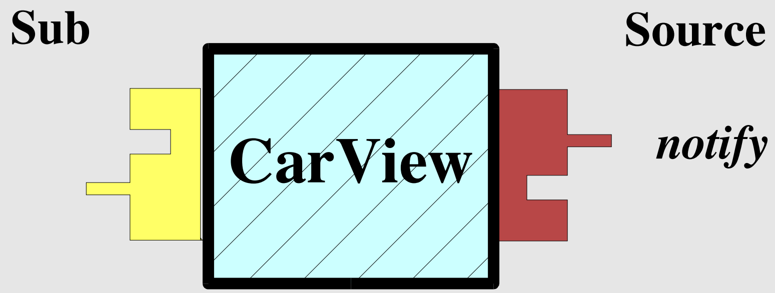
Unplug



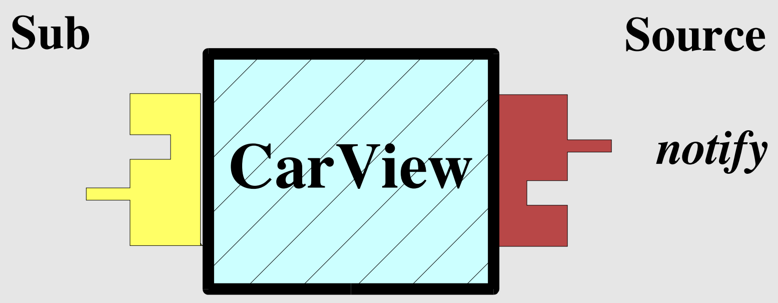
Inside o4:

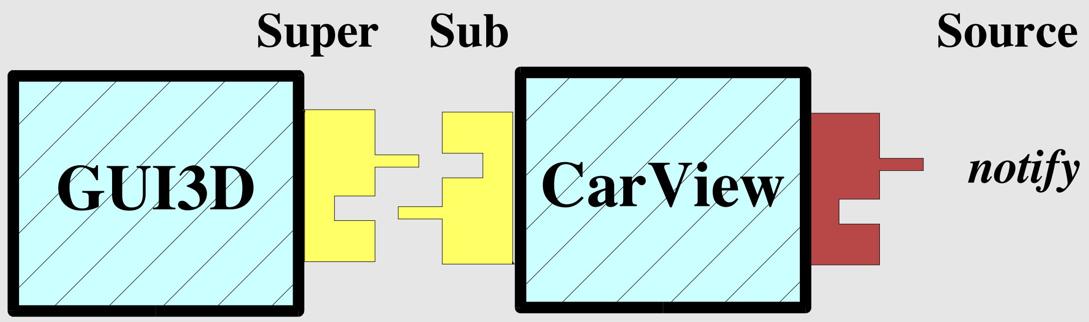
```
unplug p2;
```

Dynamic Dispatch

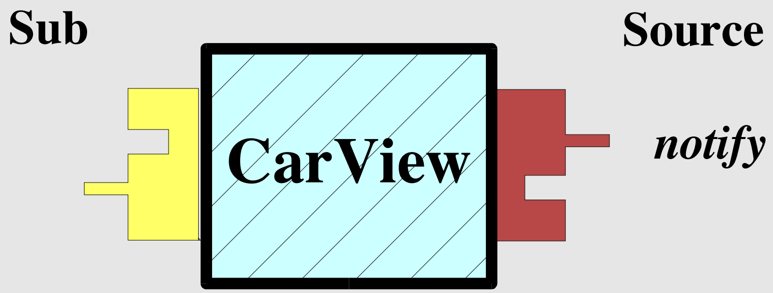


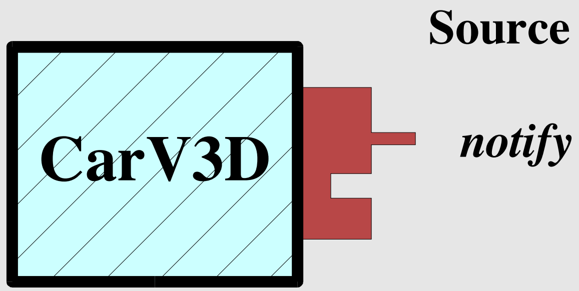
Dynamic Dispatch





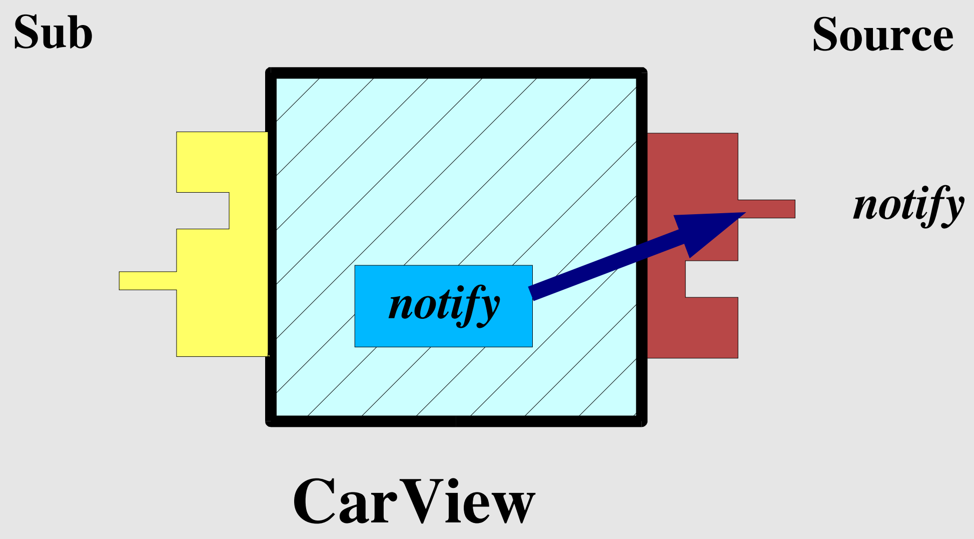
Dynamic Dispatch



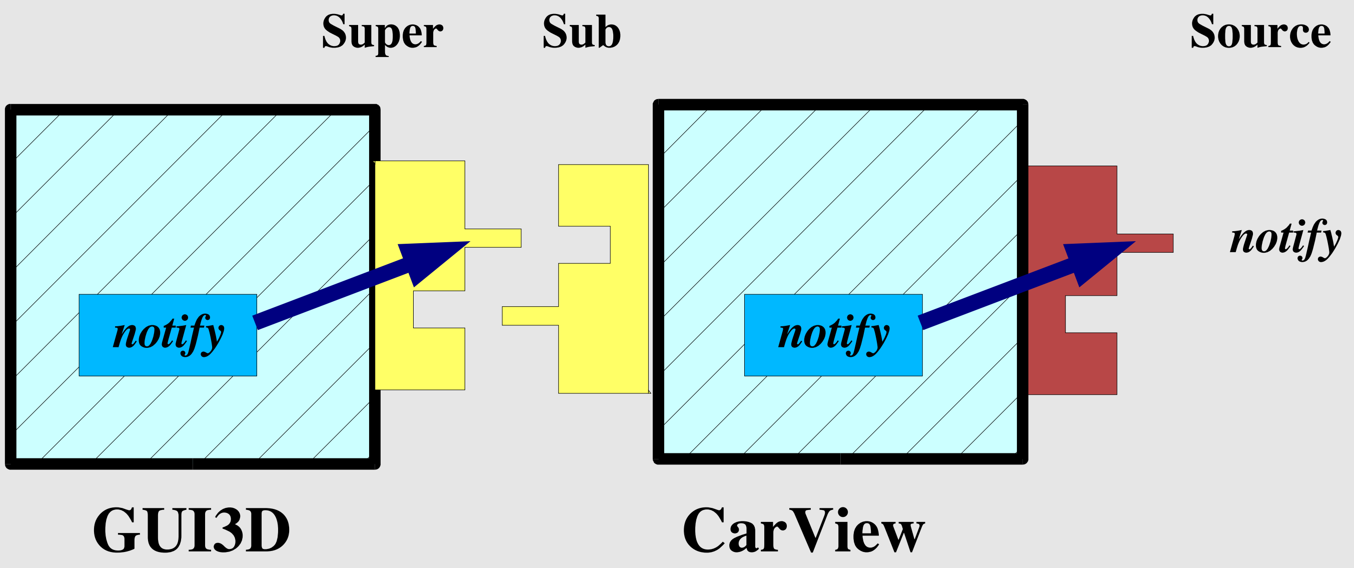


objectage type **CarV3D** **<:** objectage type **CarView**

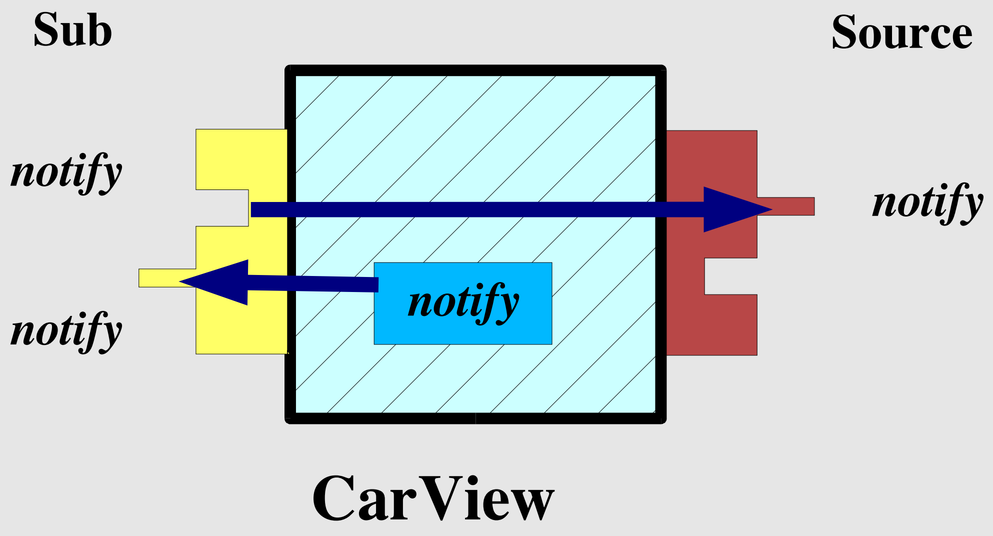
Static Dispatch



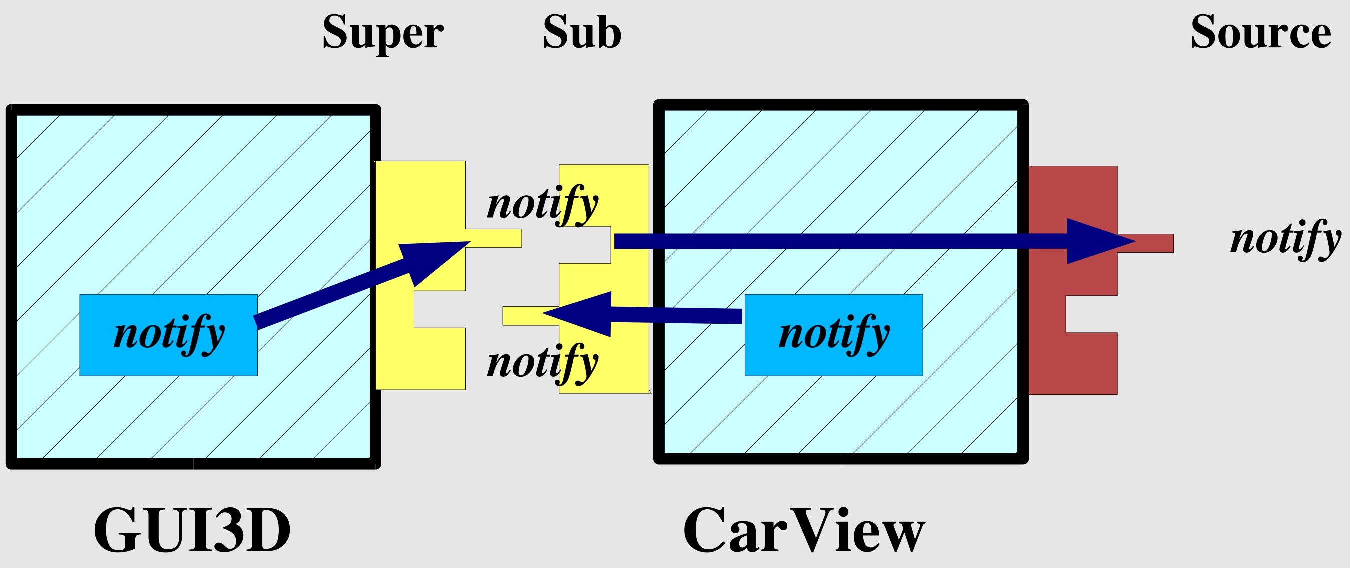
Static Dispatch



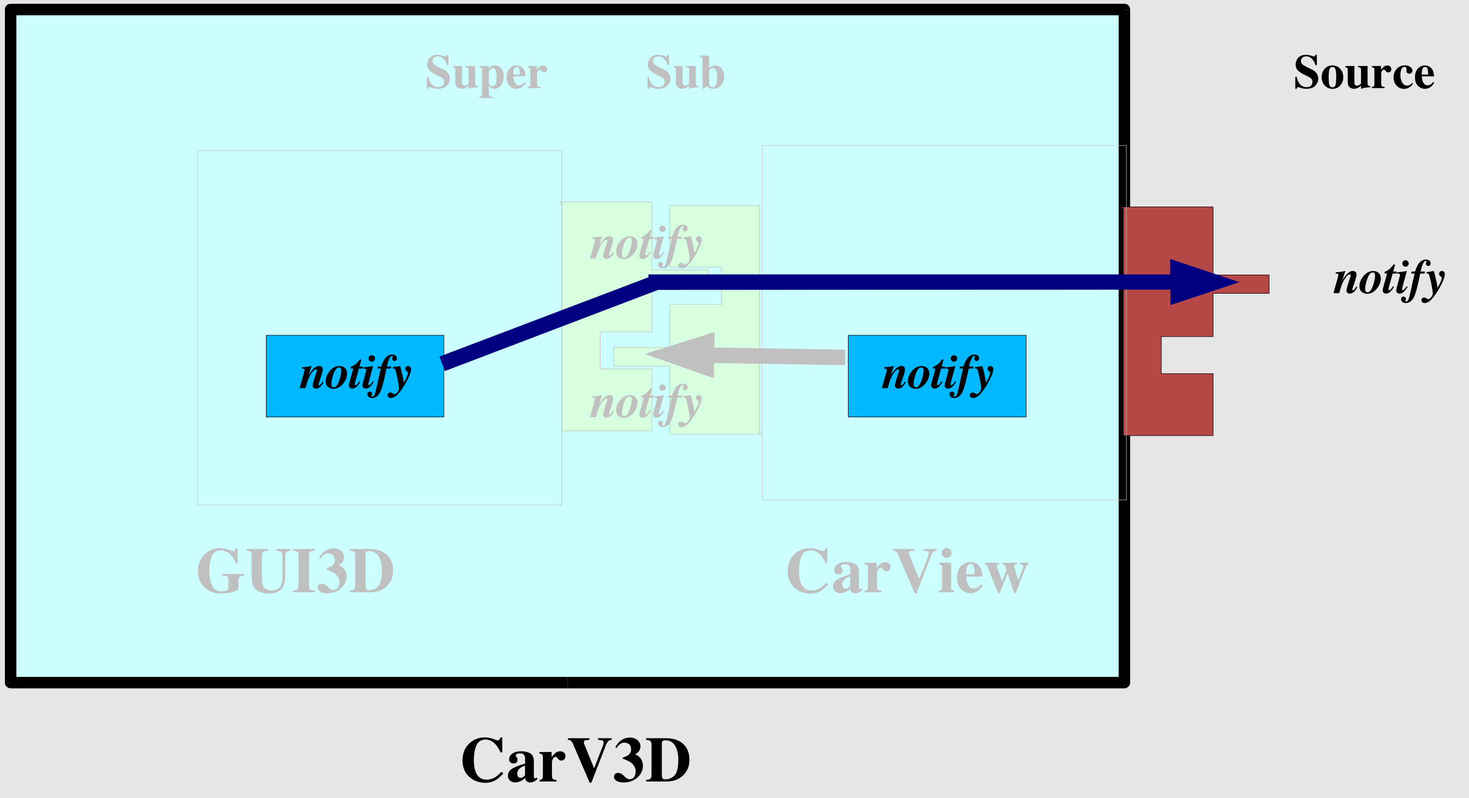
Overridable Method



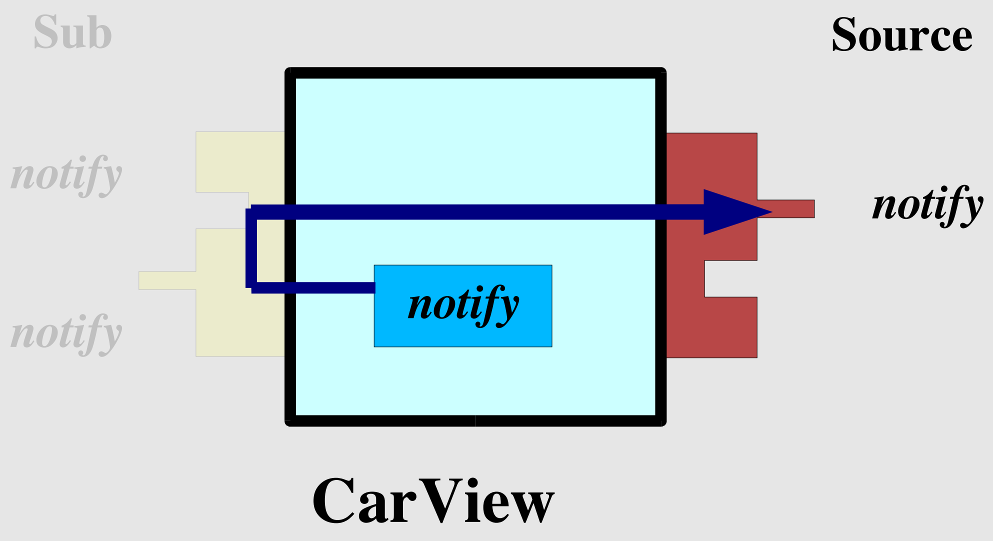
Dynamic Dispatch: CarV3D



Dynamic Dispatch: Objectage CarV3D



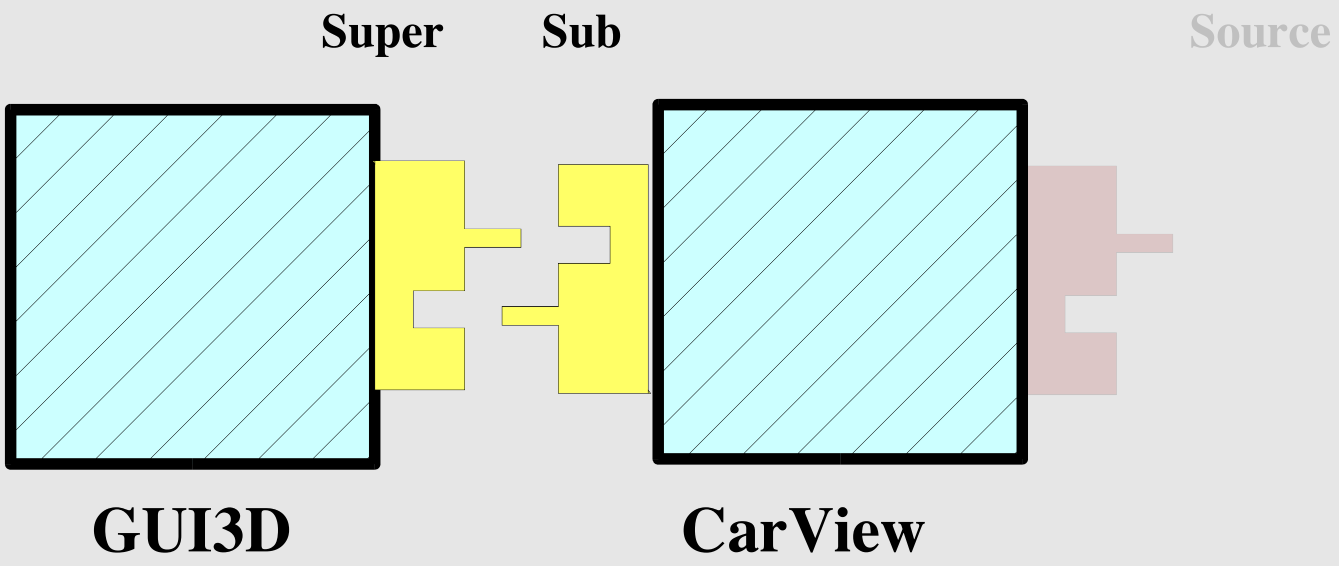
Dynamic Dispatch: Objectage CarView



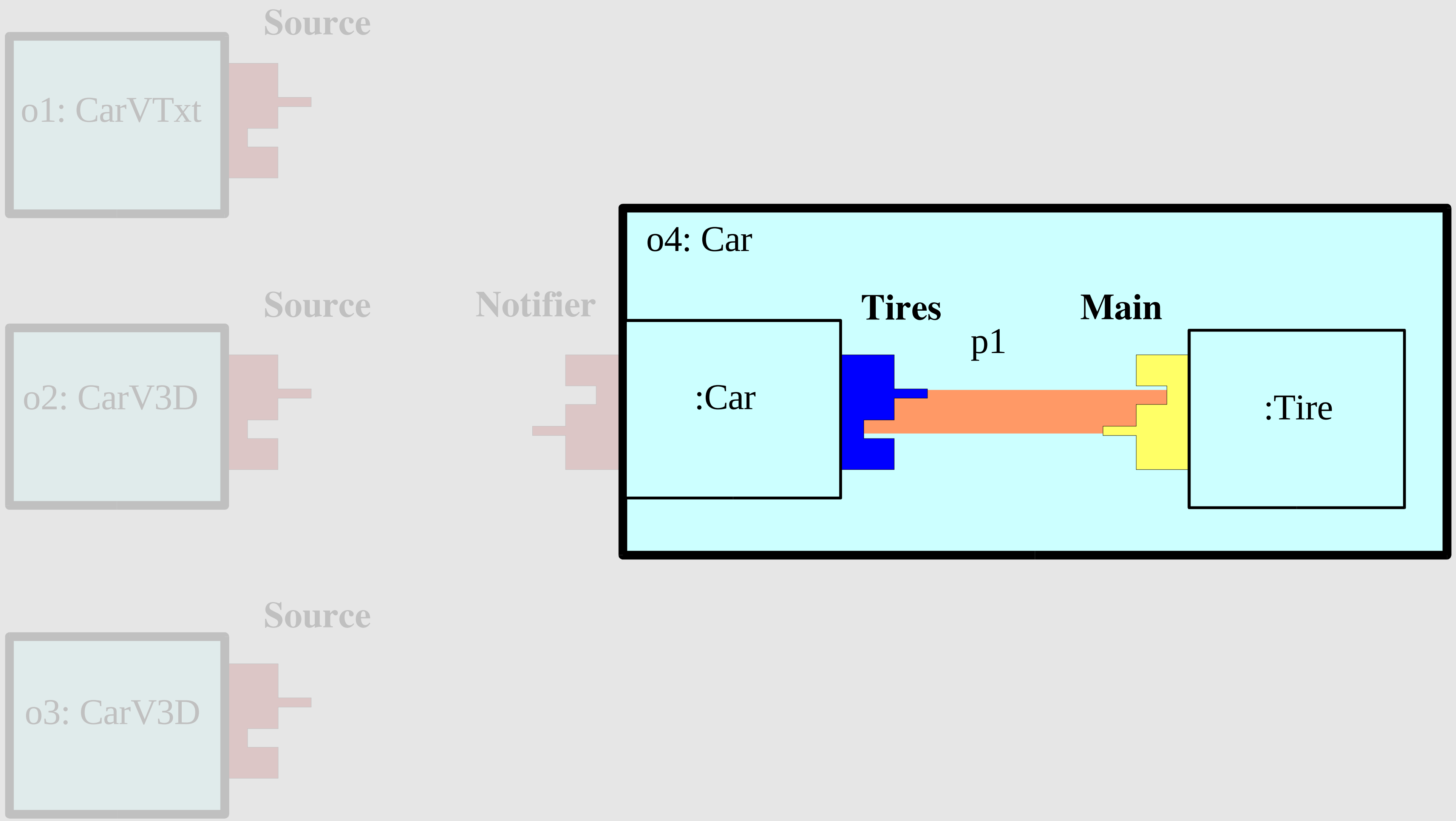
The Type System

- static typechecking for (dynamic) interactions: bi-directional interface match with subtyping.
- protecting internal representation: avoiding plugging handles to escape.
- no connection masquerading: avoiding connection handles to escape.

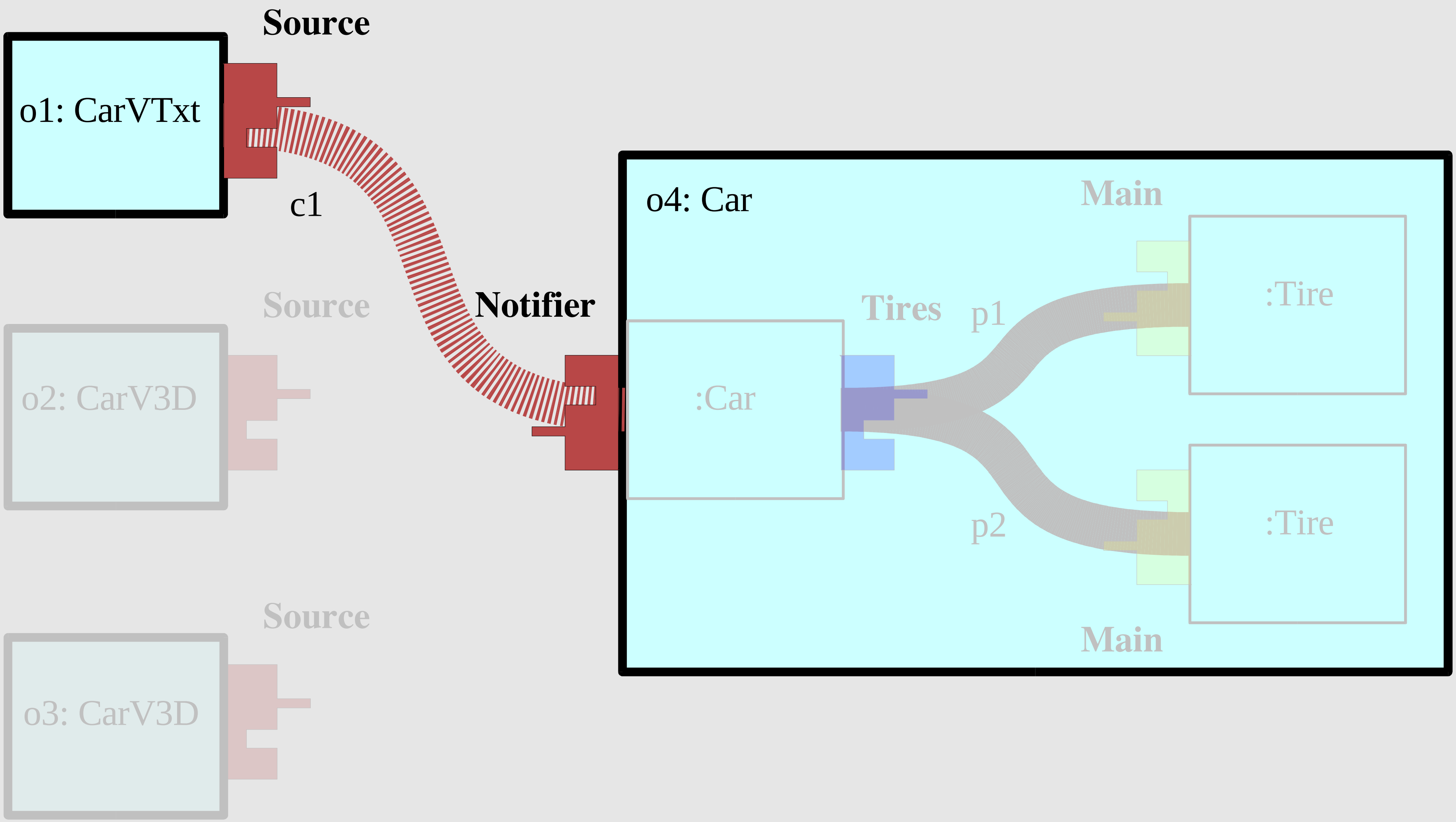
Mixing: The Class Interaction



Plugging: The Whole-Part Interaction



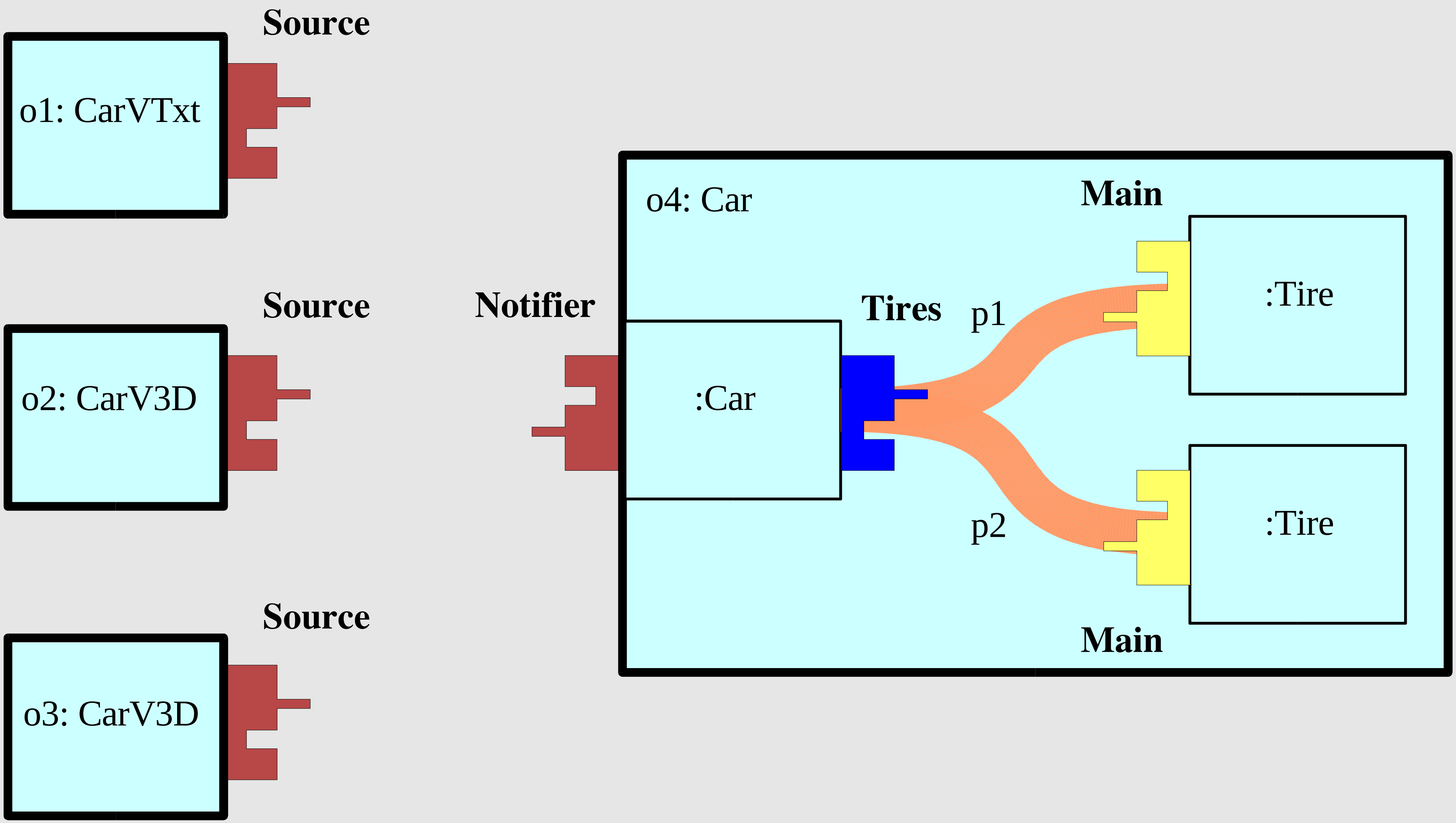
Connection: The Peer-to-Peer Interaction



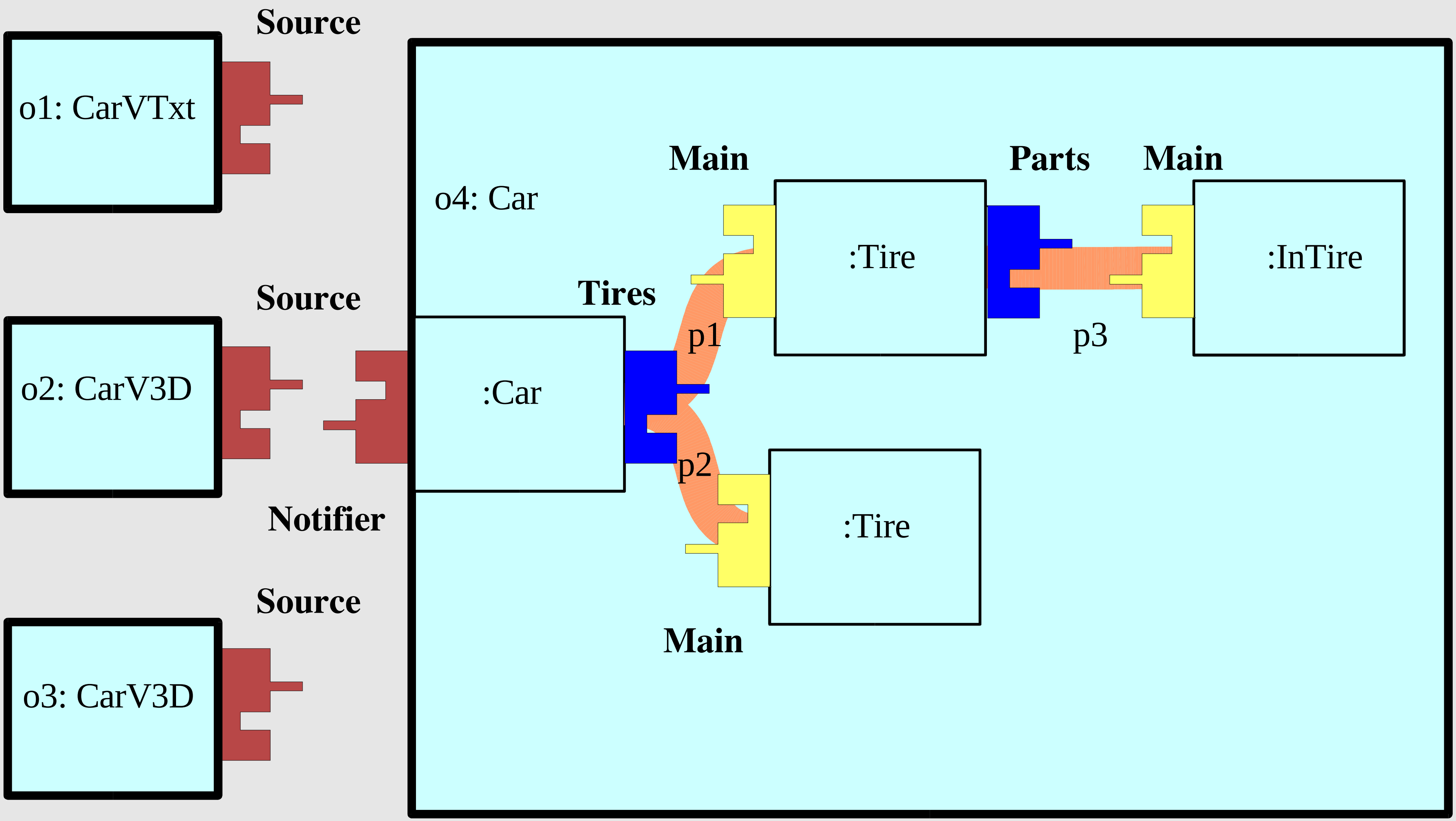
The Type System

- static typechecking for (dynamic) interactions: bi-directional interface match with subtyping.
- protecting internal representation: avoiding plugging handles to escape.
- no connection masquerading: avoiding connection handles to escape.

Protecting Internal Representation



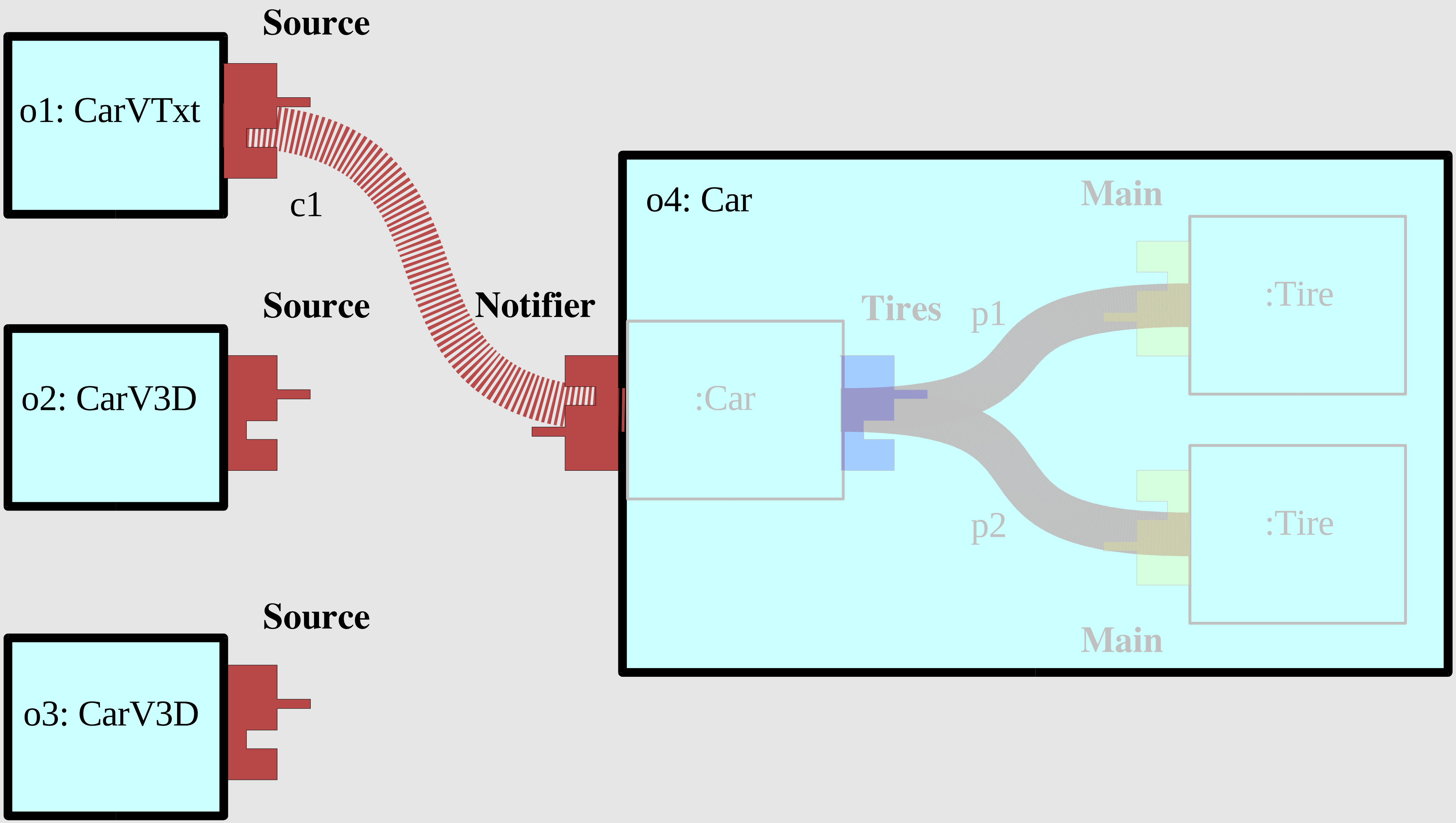
Protecting Internal Representation



The Type System

- static typechecking for (dynamic) interactions: bi-directional interface match with subtyping.
- protecting internal representation: avoiding plugging handles to escape.
- no connection masquerading: avoiding connection handles to escape.

No Connection Masquerading



Related Work

- explicit interfaces
    - component systems
    - architectural description languages
- object ownership, alias protection
- composition: mixins, Traits, module systems
- relationship representation
- environmental acquisition [Gil & Lorenz], [Cobbe & Felleisen]

Classages Design Principles

- Static interactions and dynamic interactions are fundamentally different.
- Internal interactions and external interactions are fundamentally different.
- Interactions fundamentally have a lifespan.
- Interactions are fundamentally bi-directional.
- Interactions always happen on explicitly defined interfaces.

Download

http://www.cs.jhu.edu/~yliu/Classages