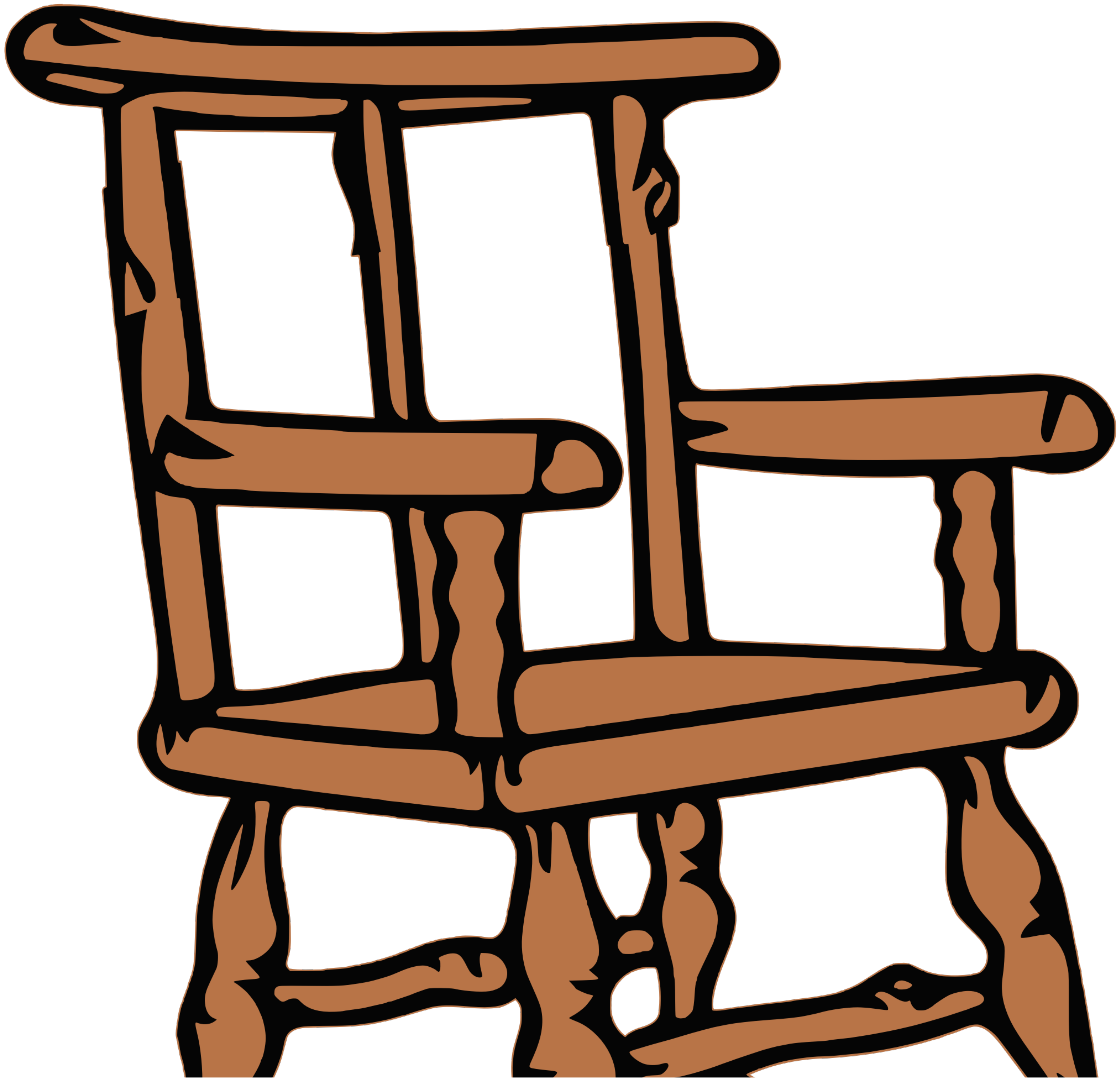
Prototypal Inheritance

Ambrose Bonnaire-Sergeant

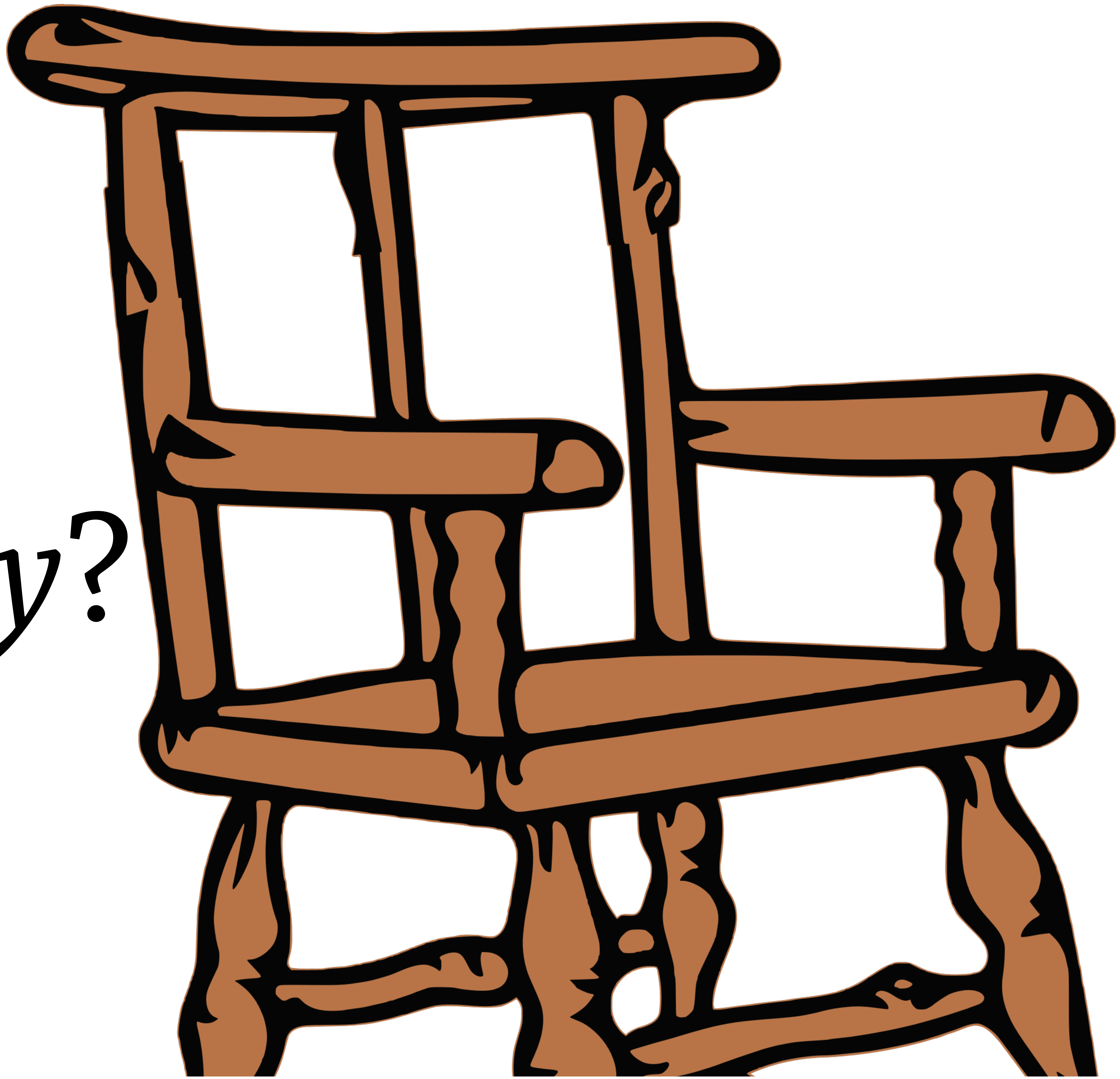
Some things are *more convenient* with computers…

Your chair…

I want one like that!



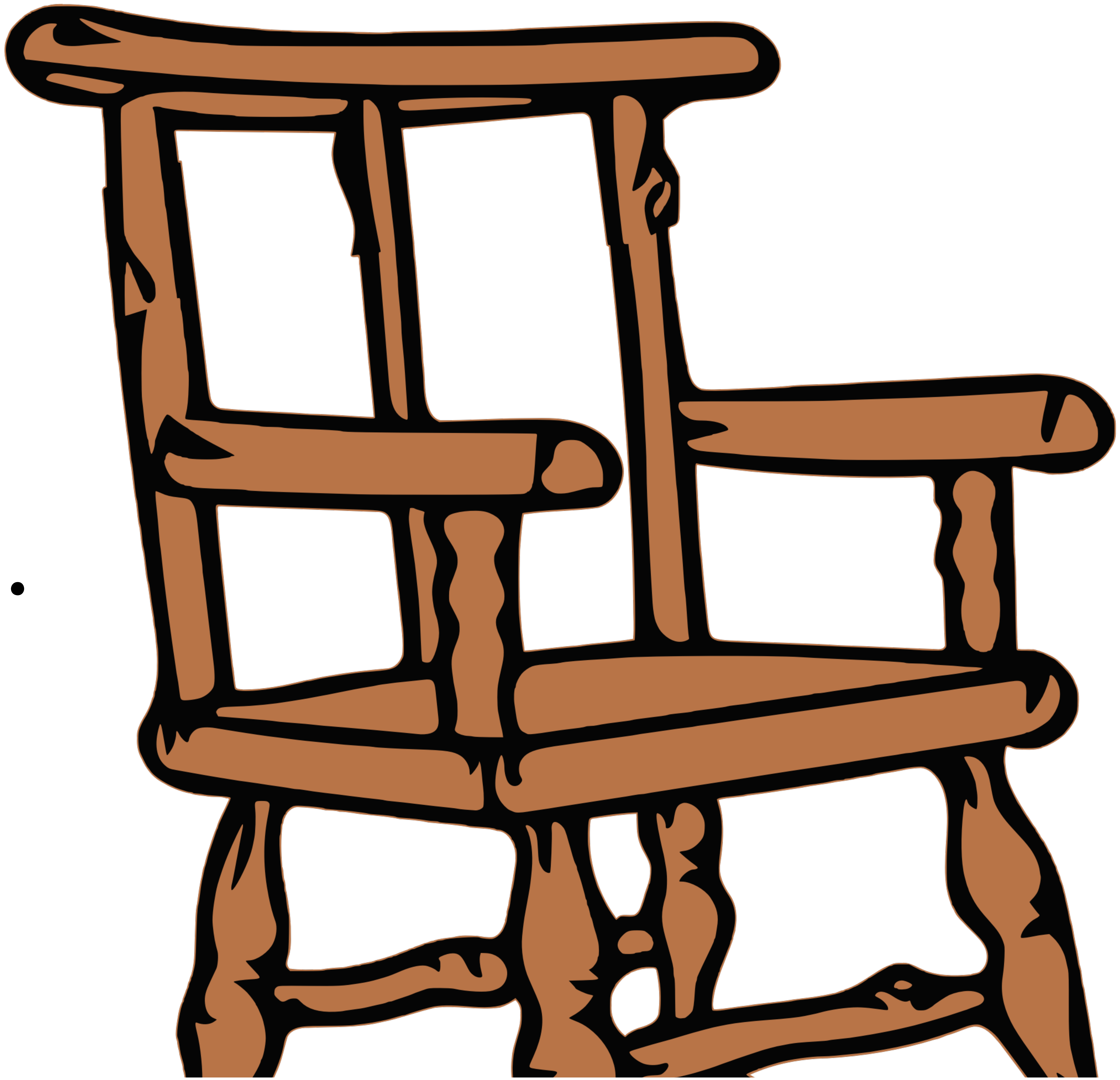
…can I’ve a *copy*?



Your chair…

You want to

attach rockers…



You want to
attach rockers…

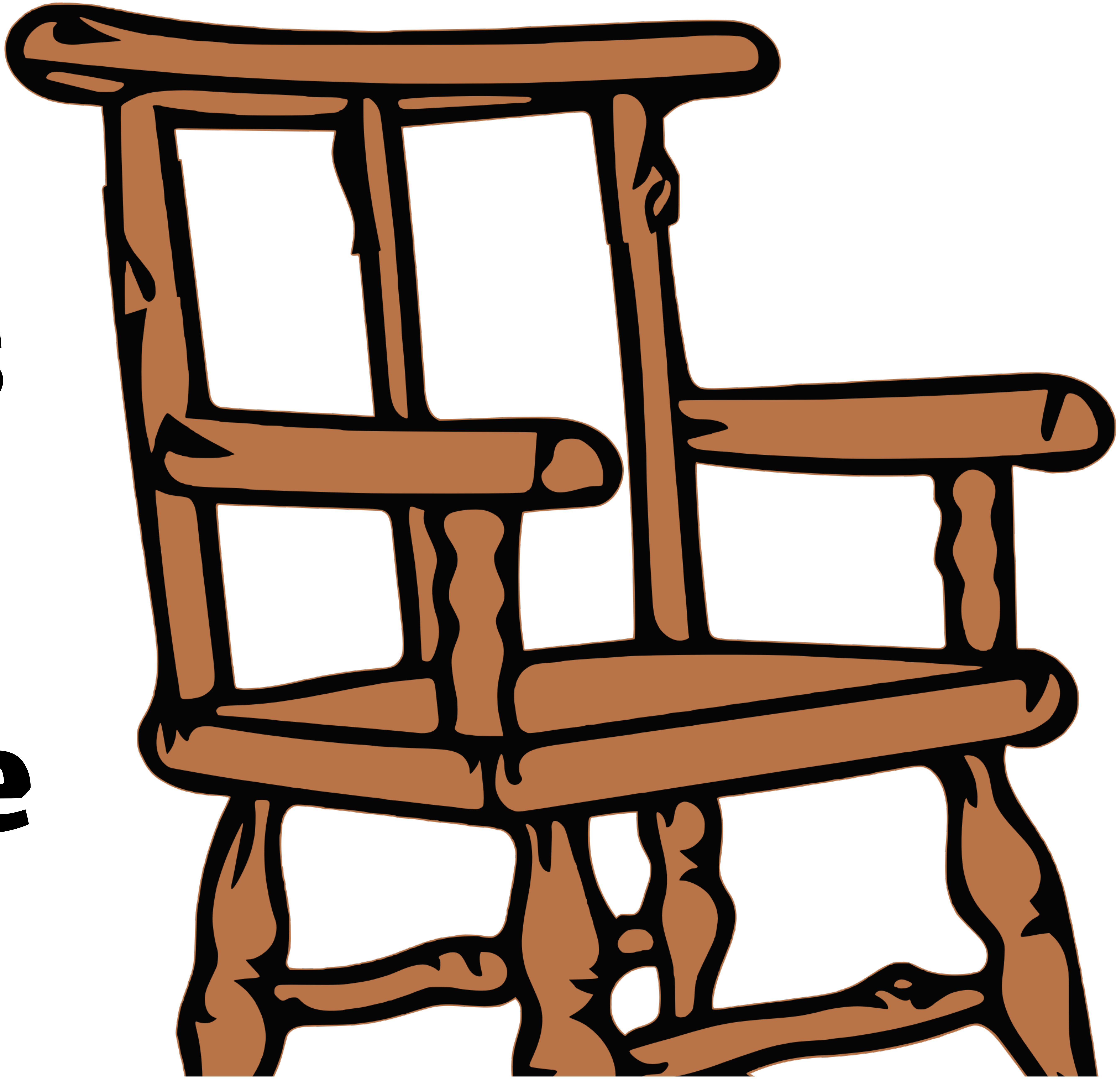
You want to

attach rockers…

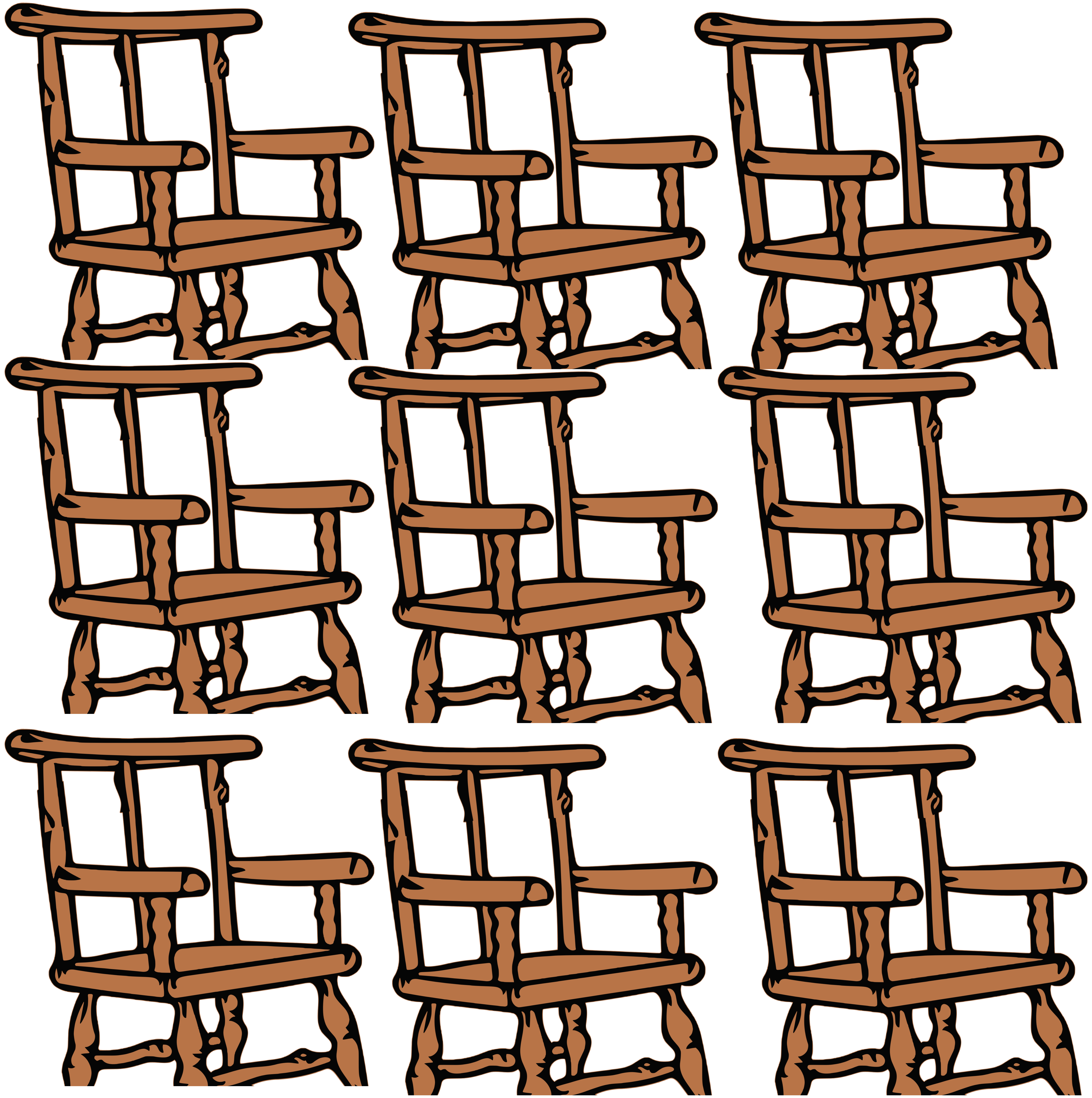
but *keep the original!*

Your chair…

Your chair's *model* is **defective**

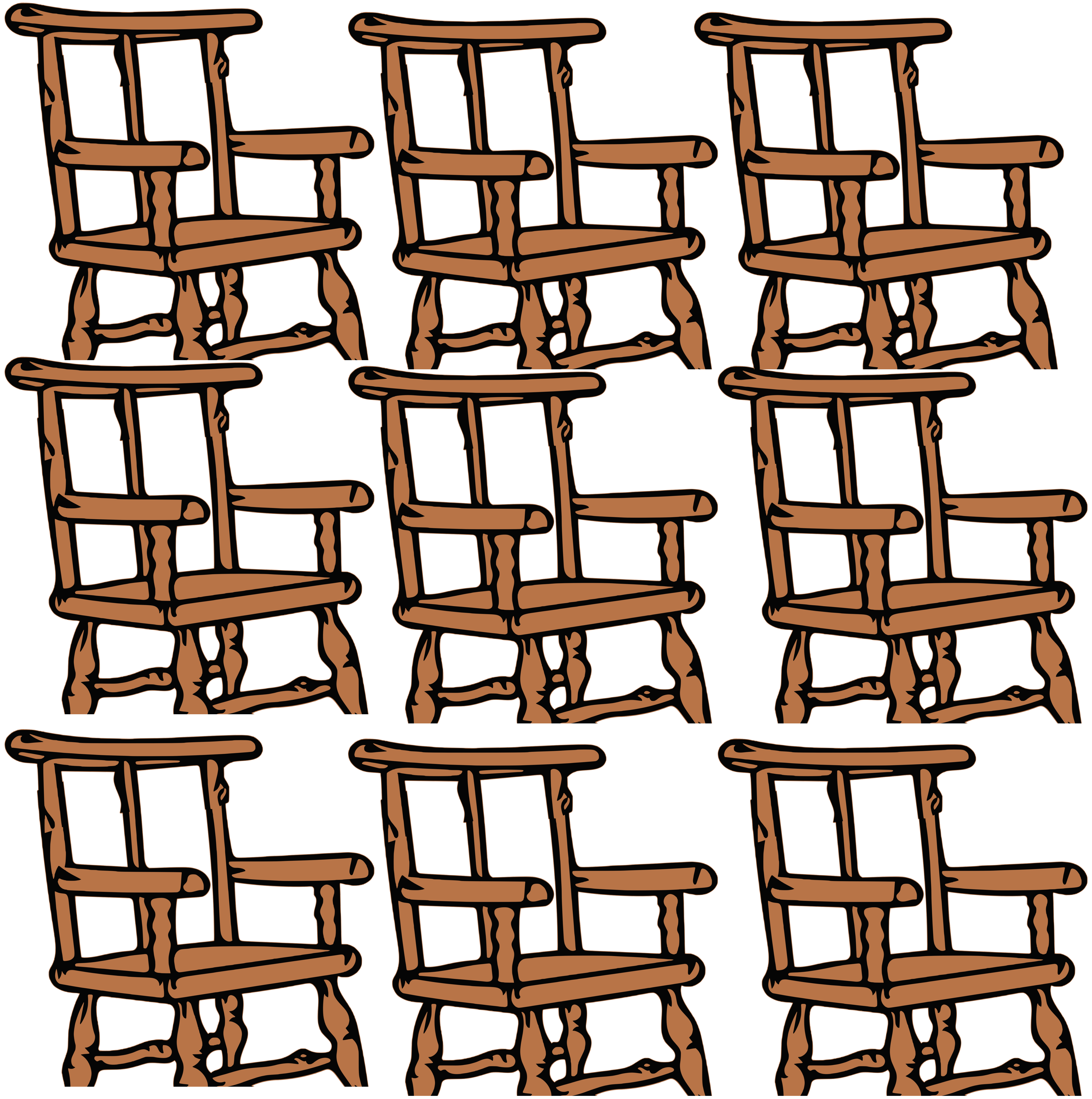


Your chair’s *model* is **defective**

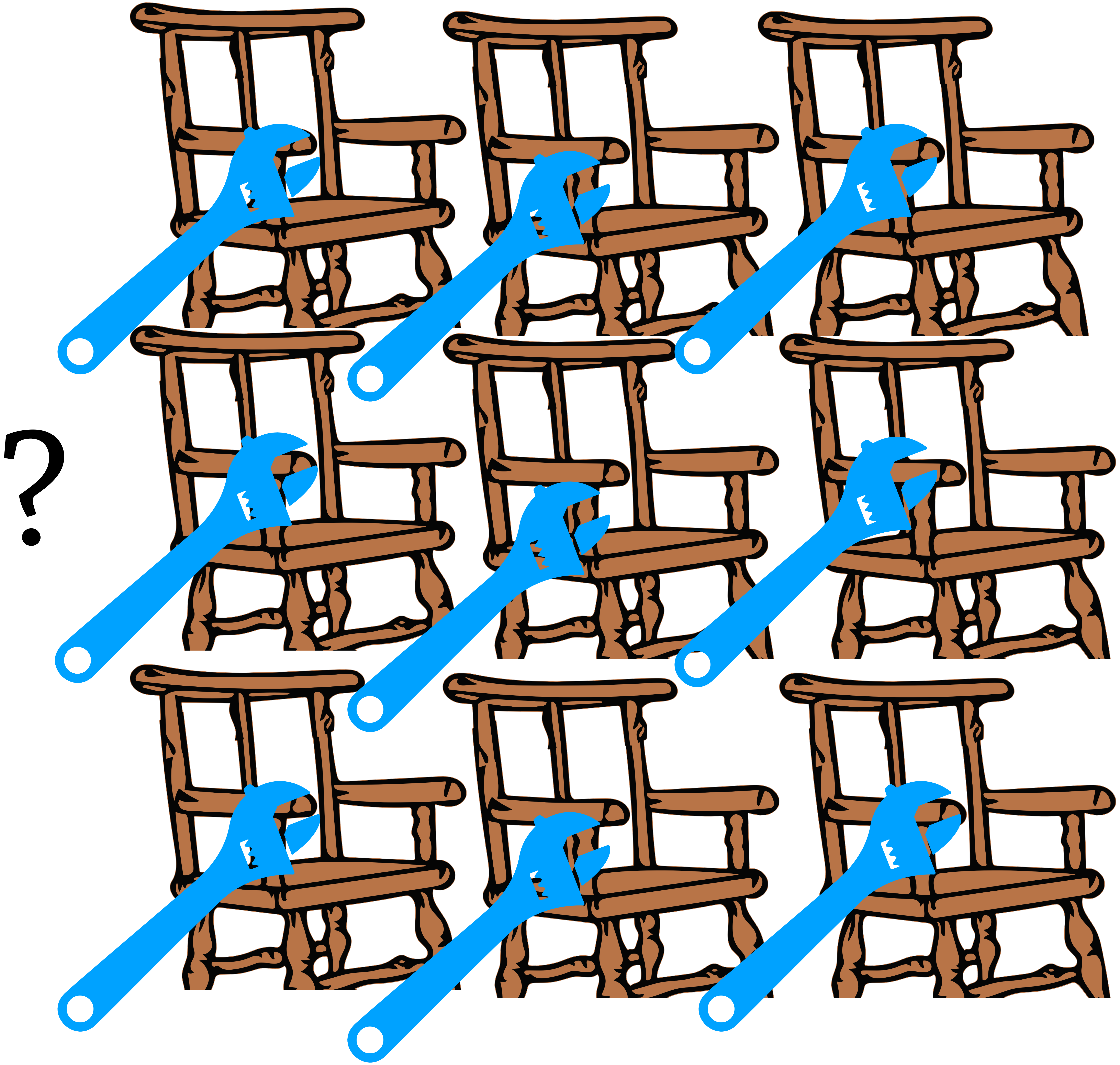


Your chair’s
model
is **defective**

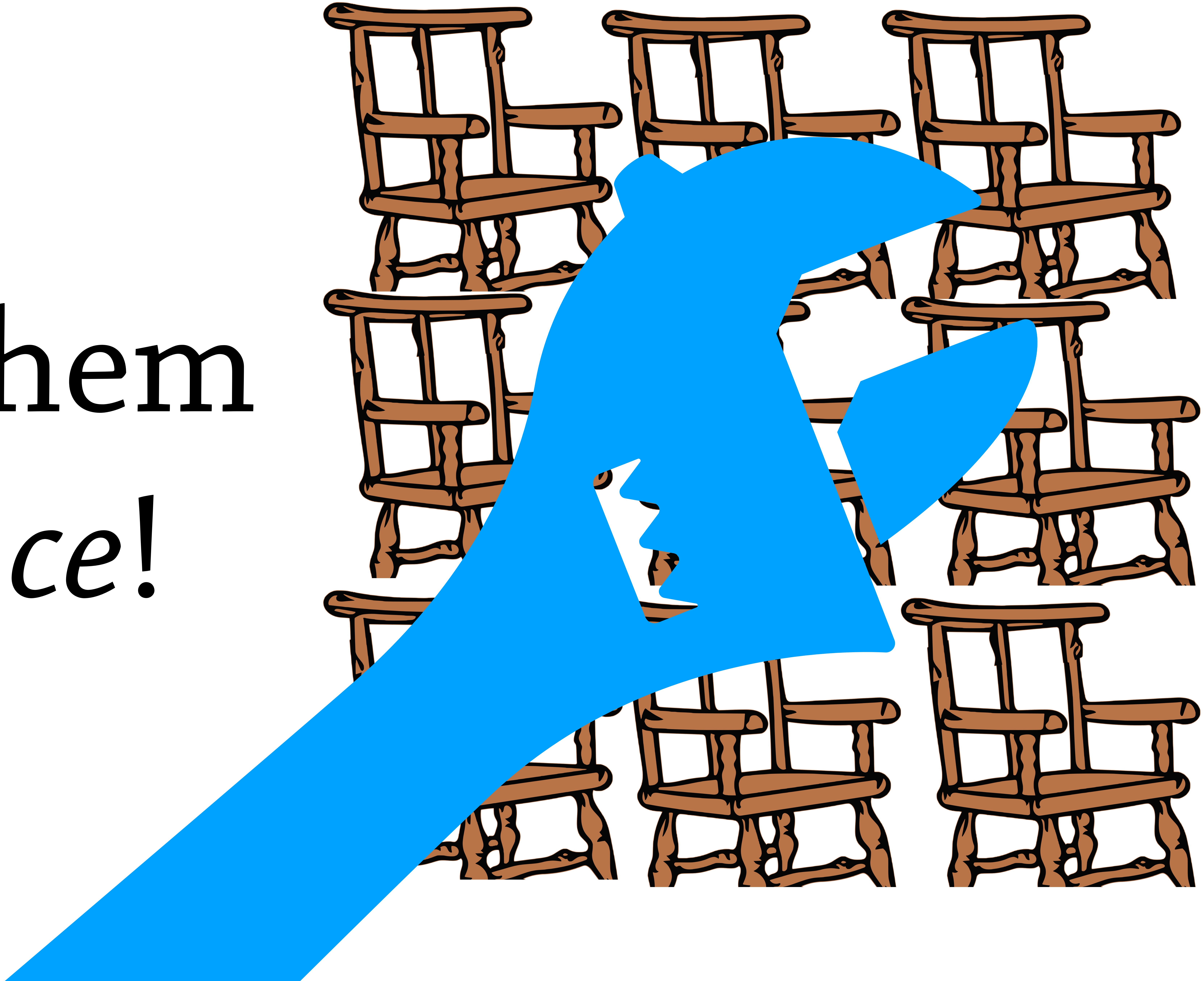
Fix each one?



Fix each one?



…or fix them *all at once!*



Physical objects don’t work like that!

Computer representations of objects are more *flexible*

This talk:

Prototypal Objects

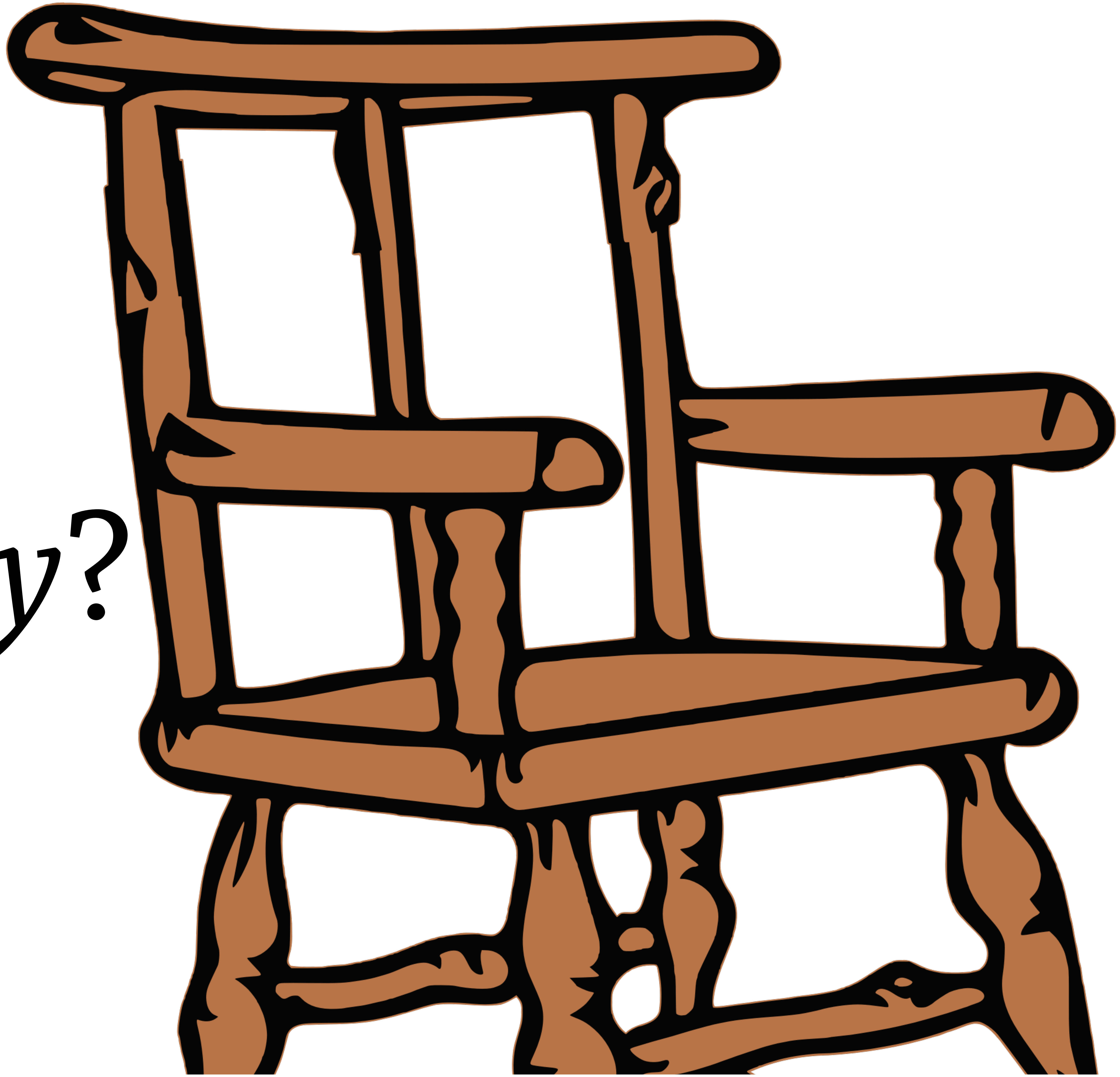
What is a *prototype*?

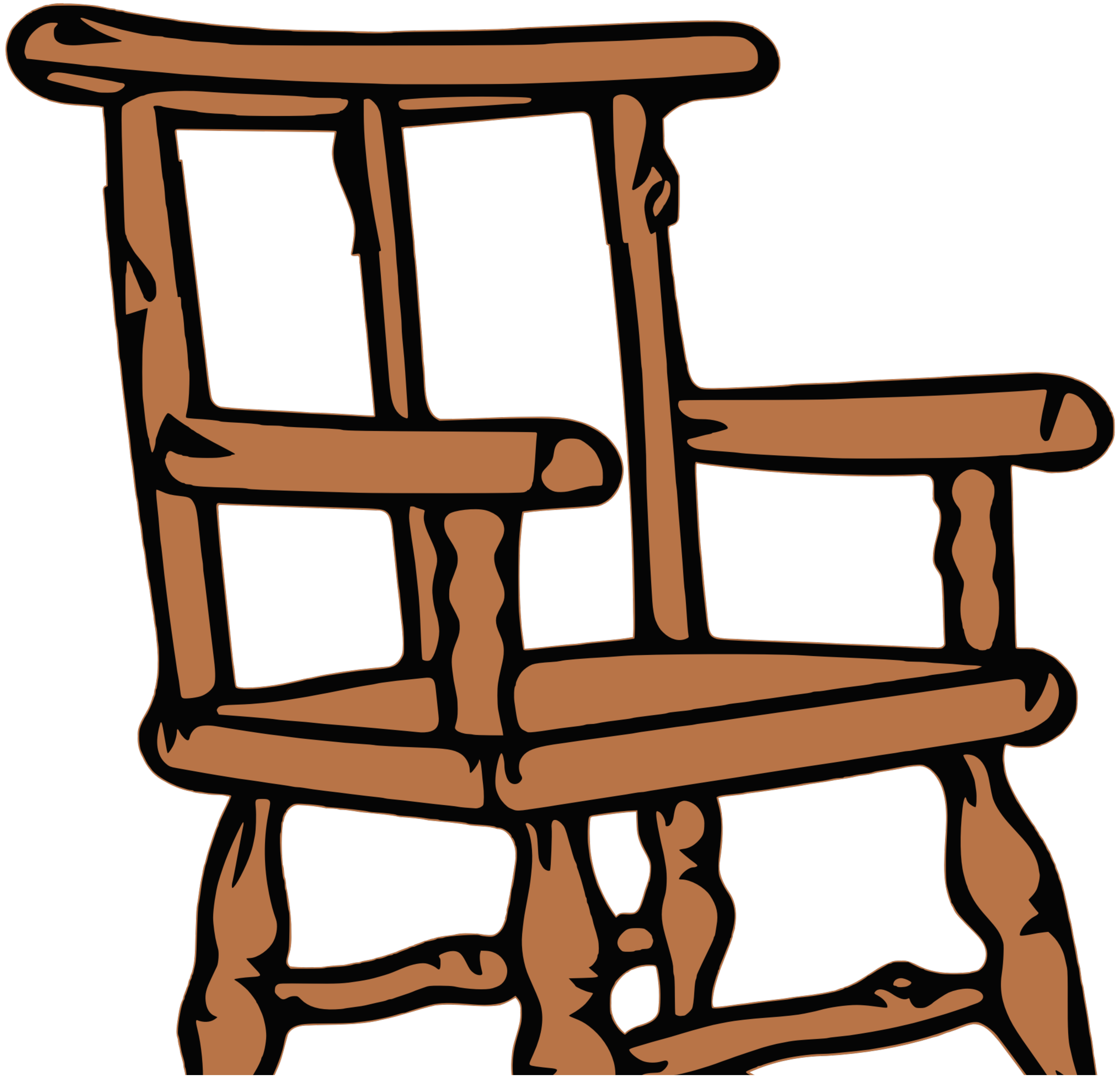
pro·to·type

noun

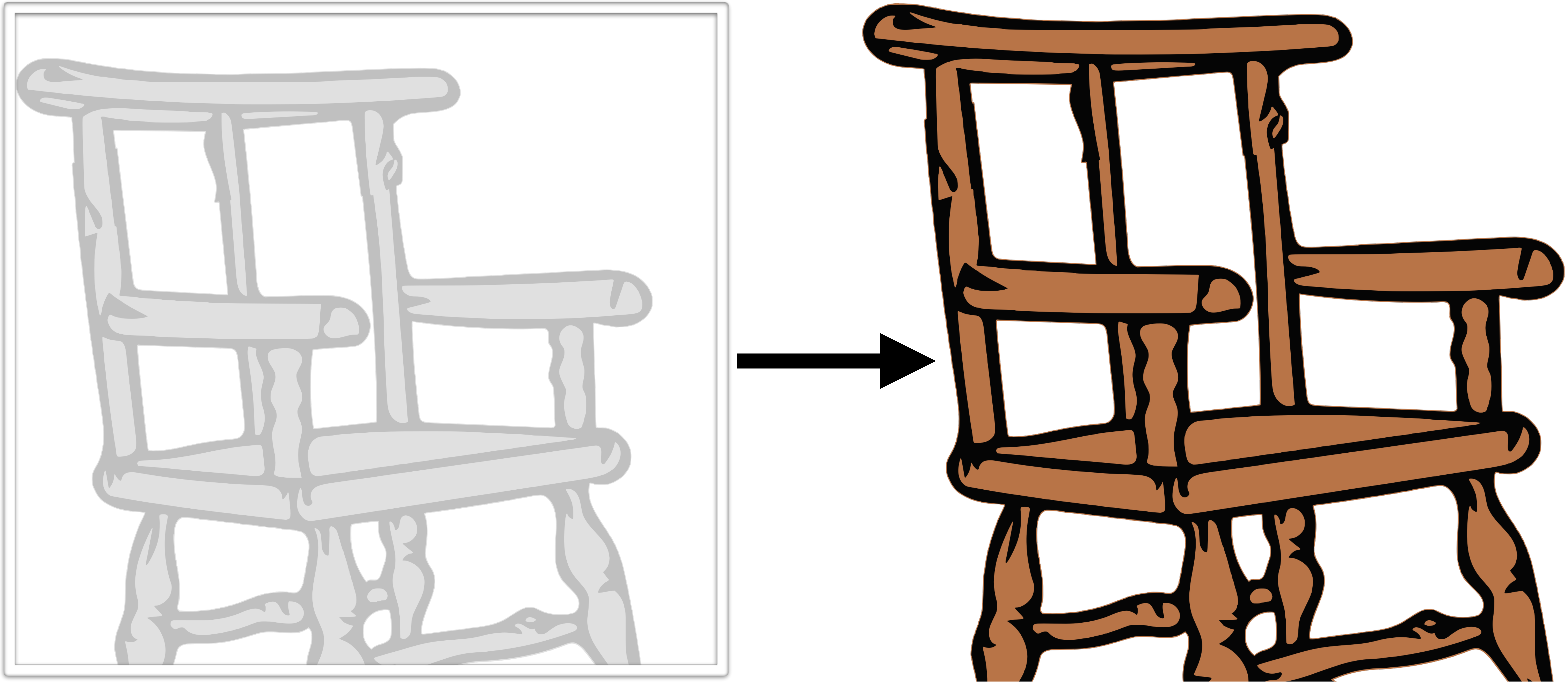
1. a first, typical or preliminary **model** of something, especially a machine, **from which** other forms are **developed or copied**.

…can I’ve a *copy*?





Prototype

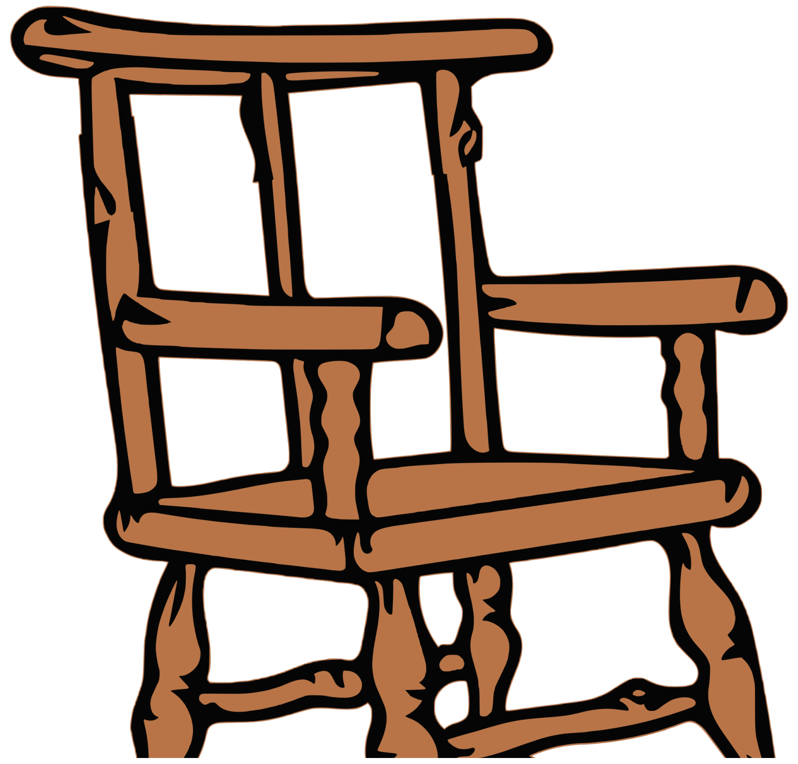


Copy

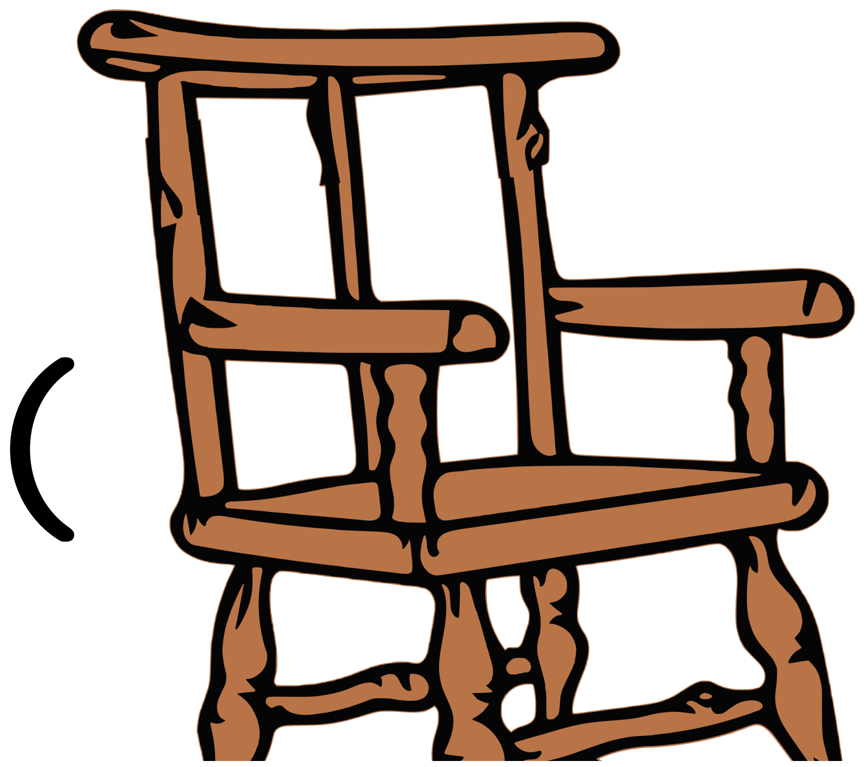
Prototype

Inheritance

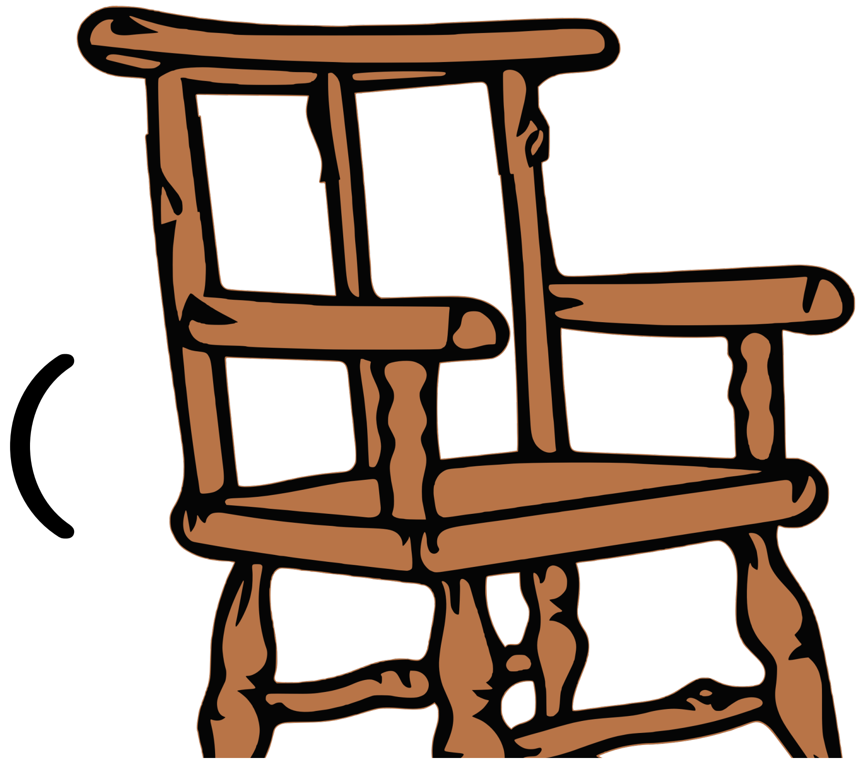
Notation



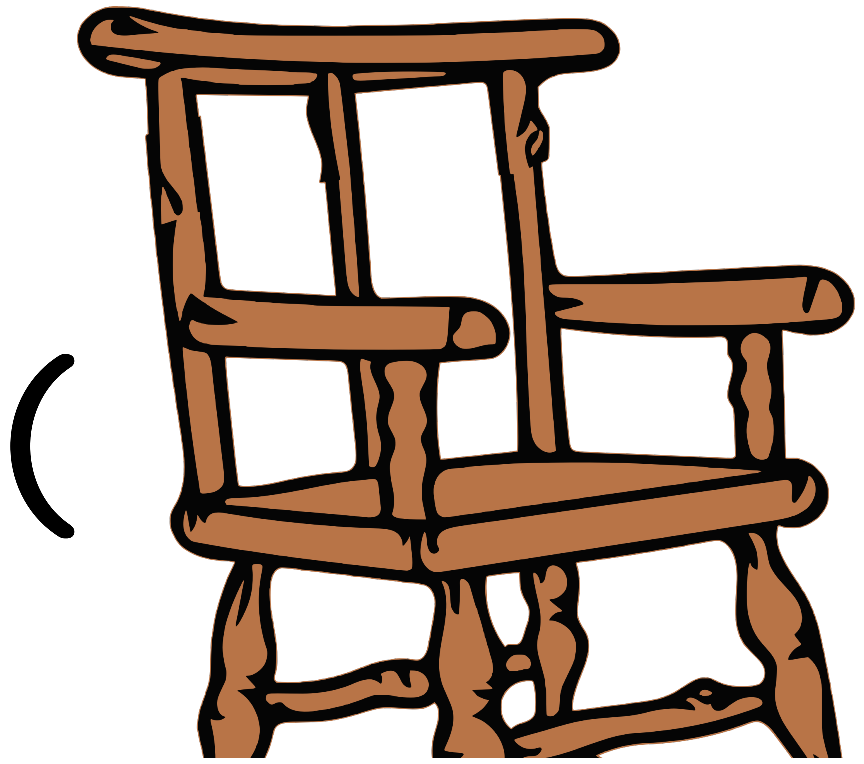
Notation

`Object.create(`  `)`

Notation

`= Object.create(`  `)`

Notation

= `Object.create(`  `)`

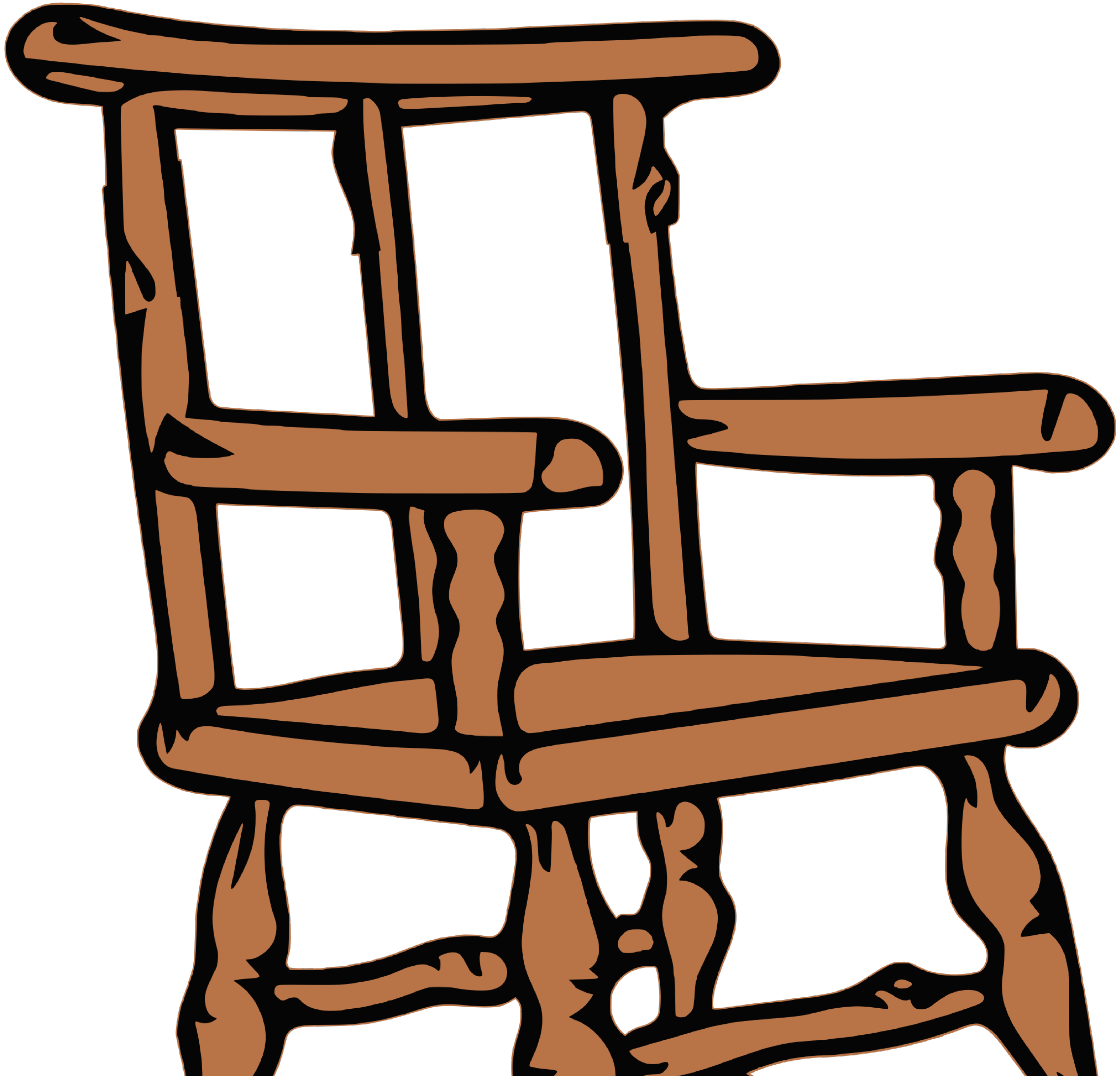
`Object.create(`*`<prototype>`*`)`

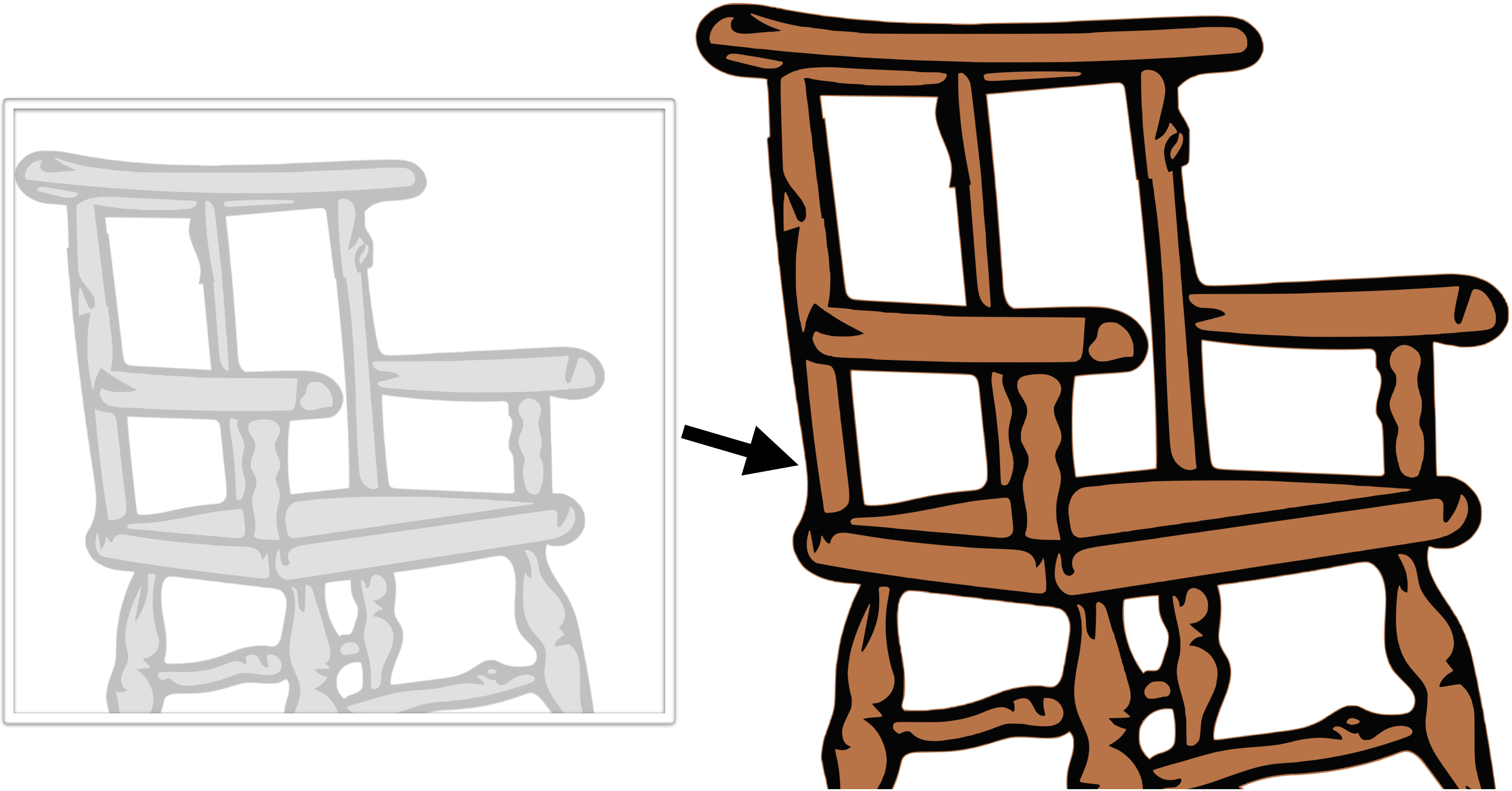
create a new object with given prototype

You want to

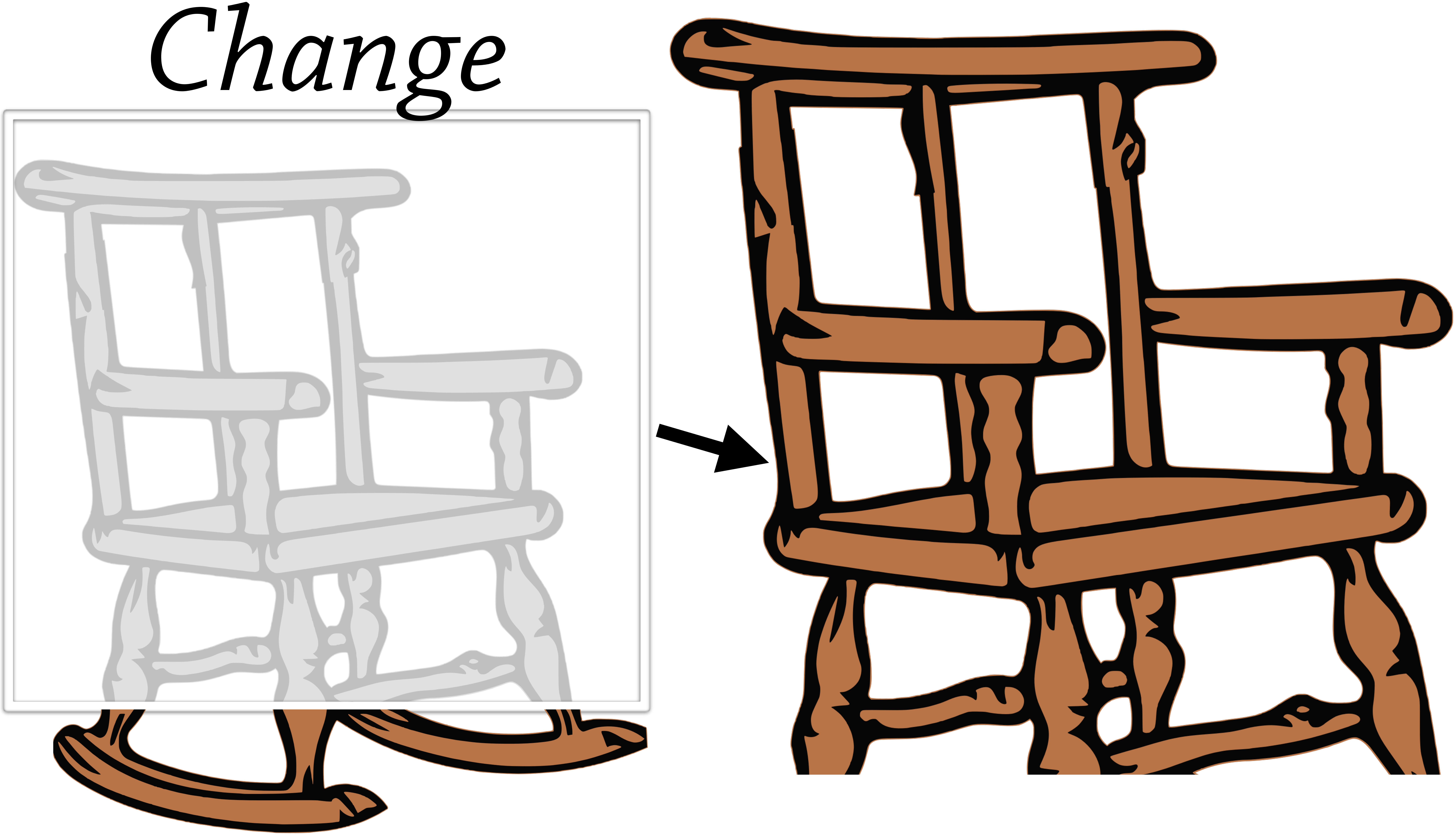
attach rockers…

but *keep the original!*

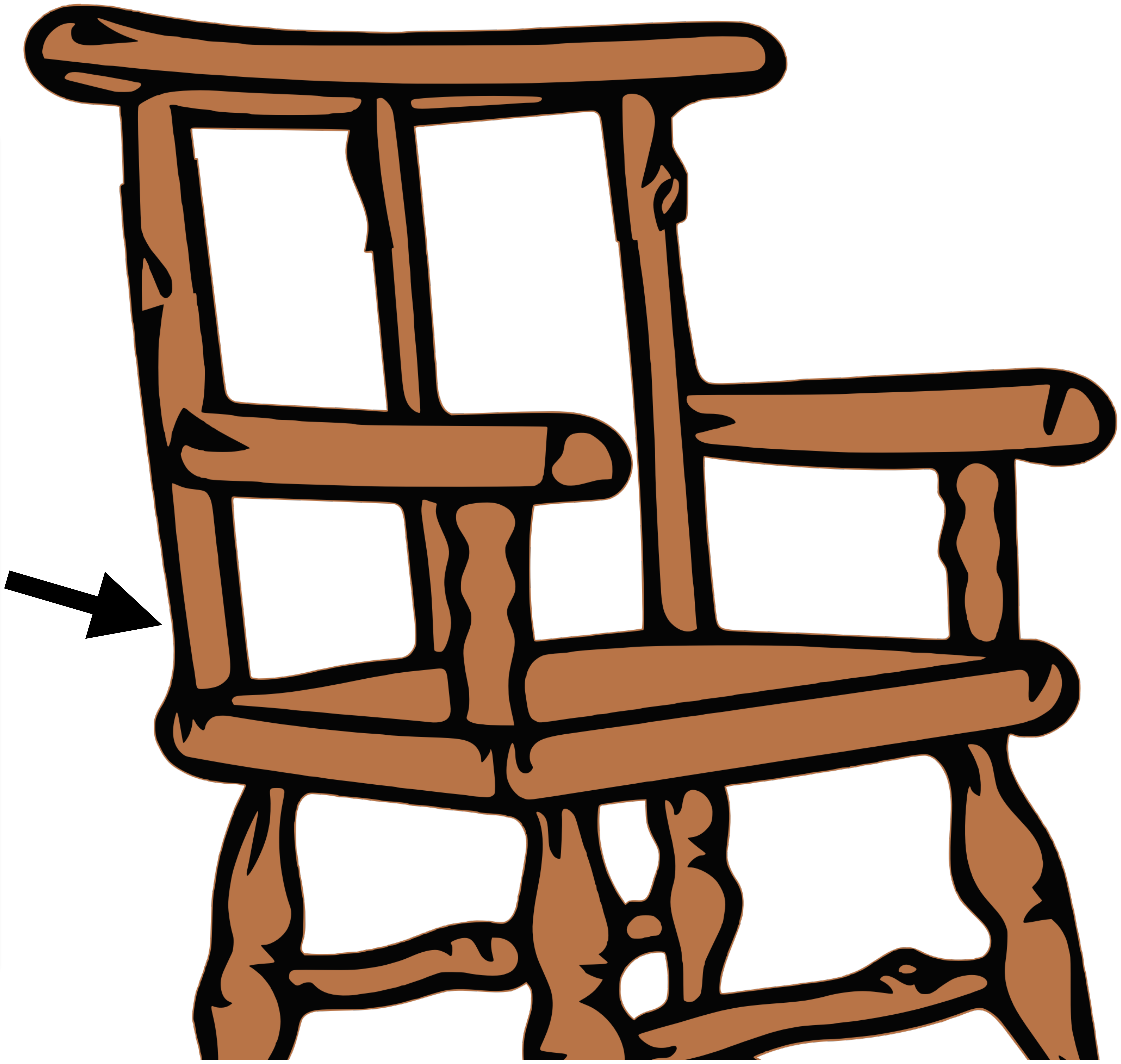


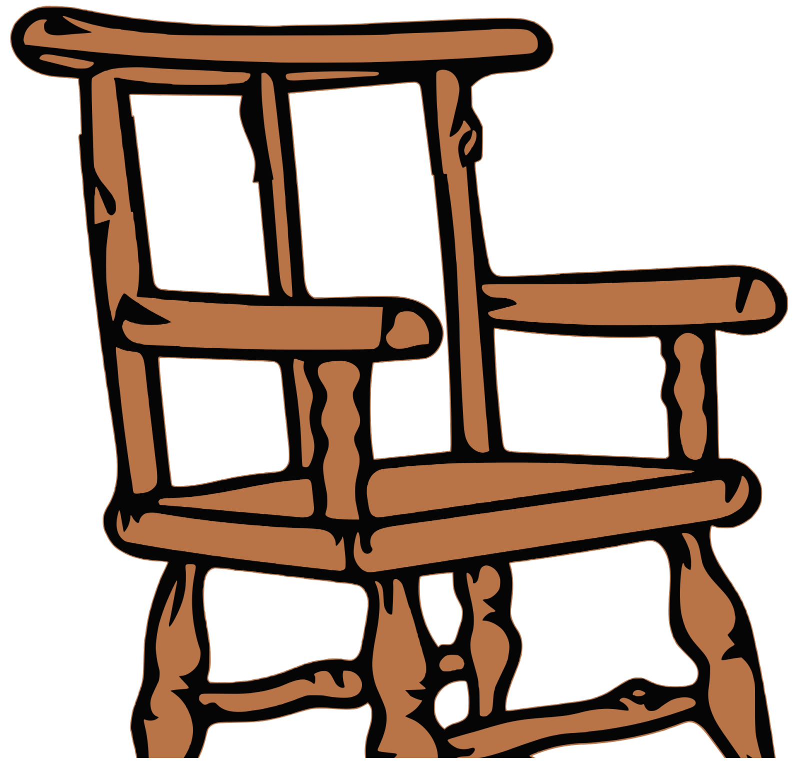


Change

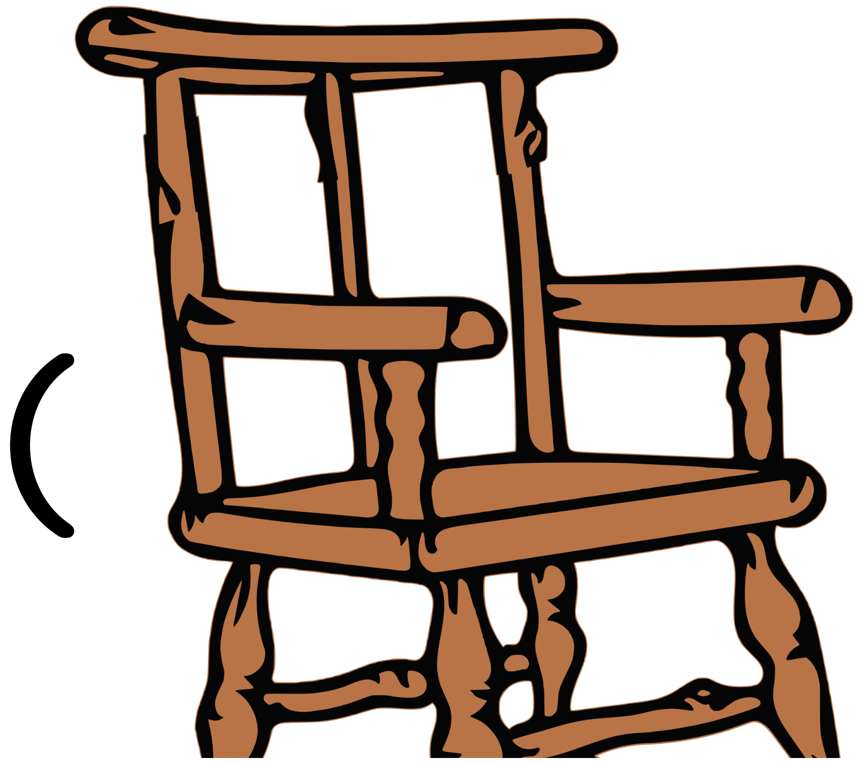


Change

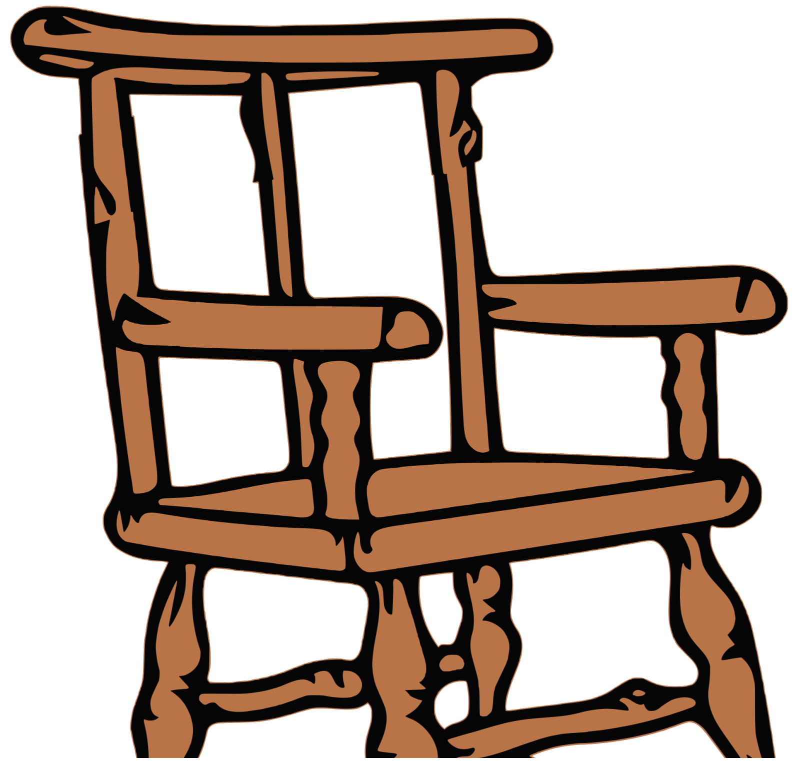


Commands	Result
	

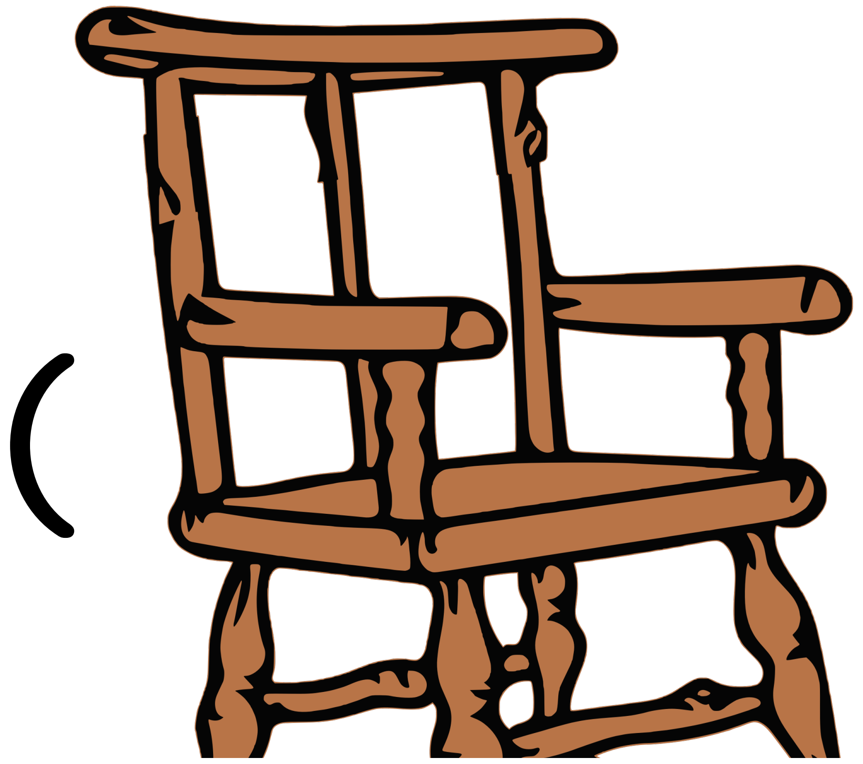
Commands

`= Object.create(`  `);`

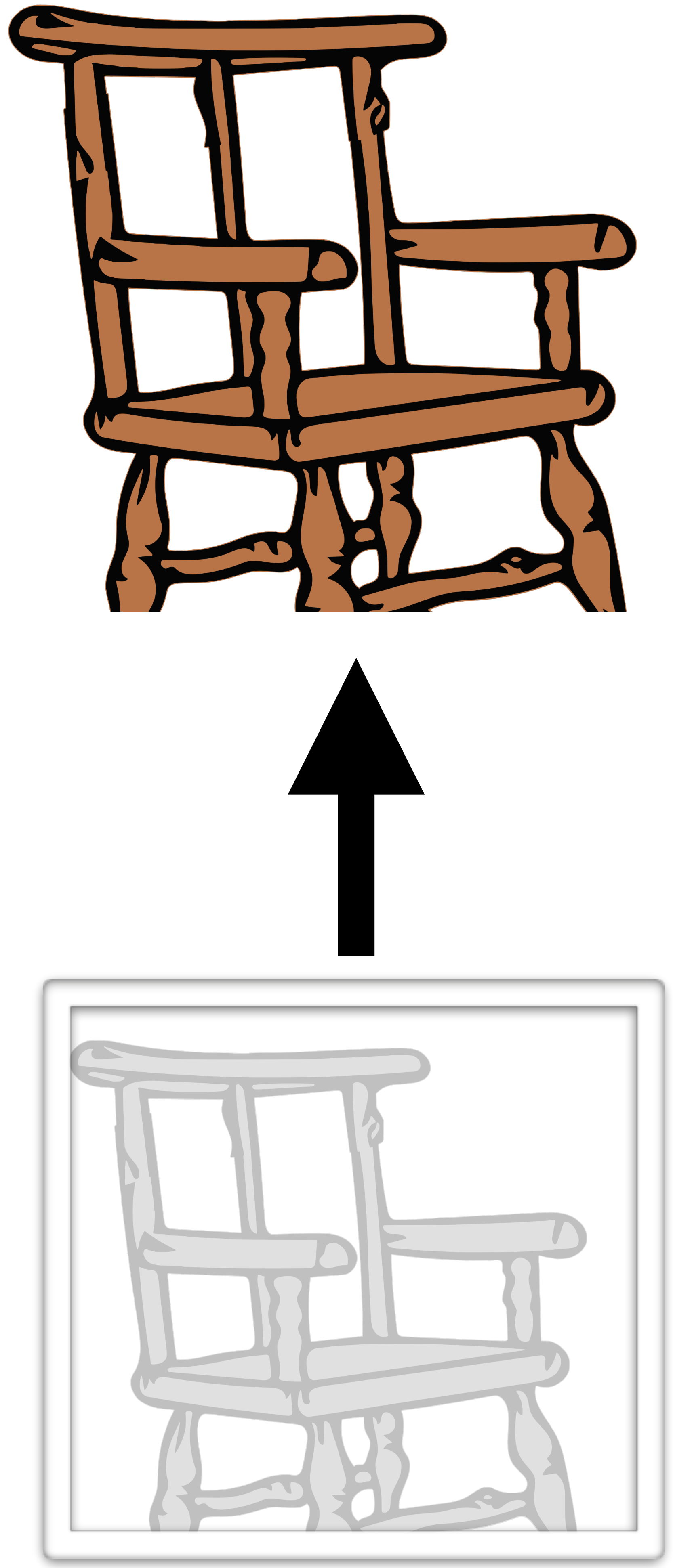
Result



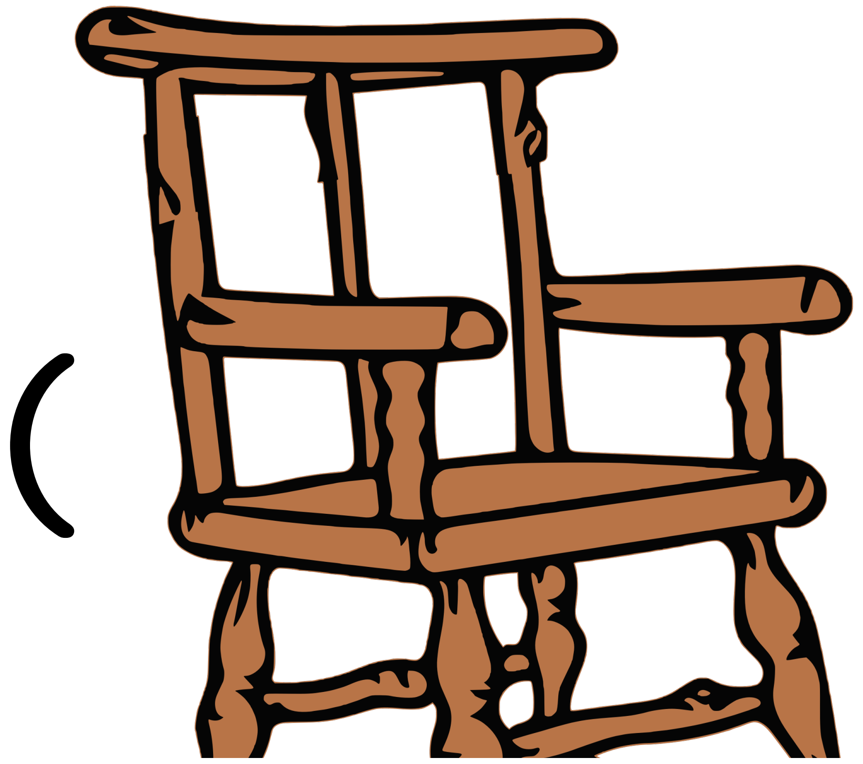
Commands

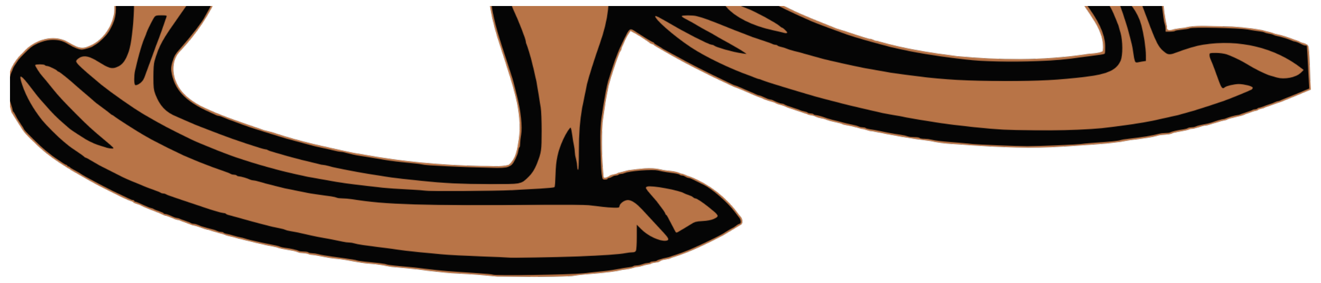
`= Object.create(`  `);`

Result

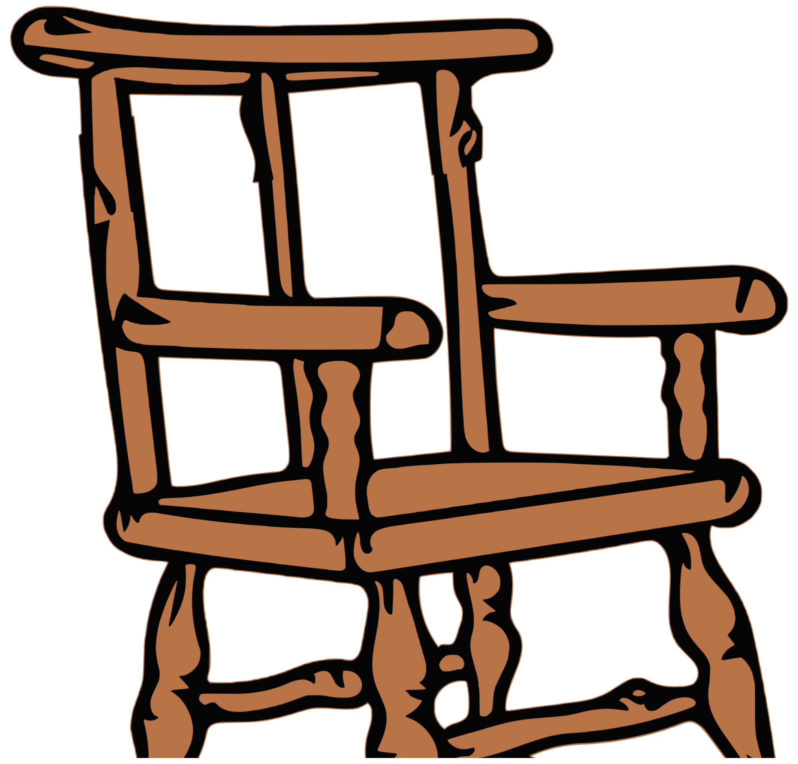


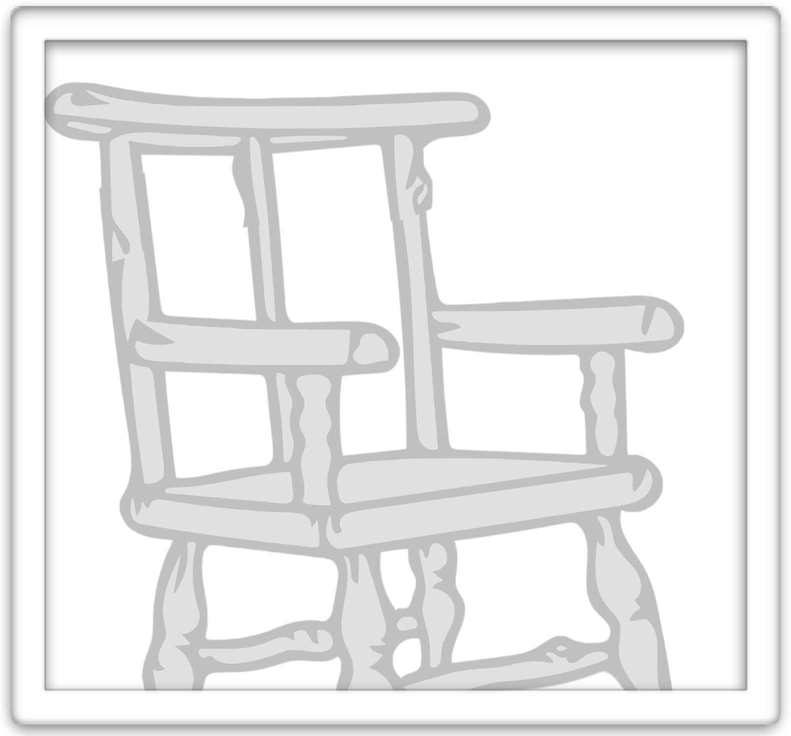
Commands

= Object.create();

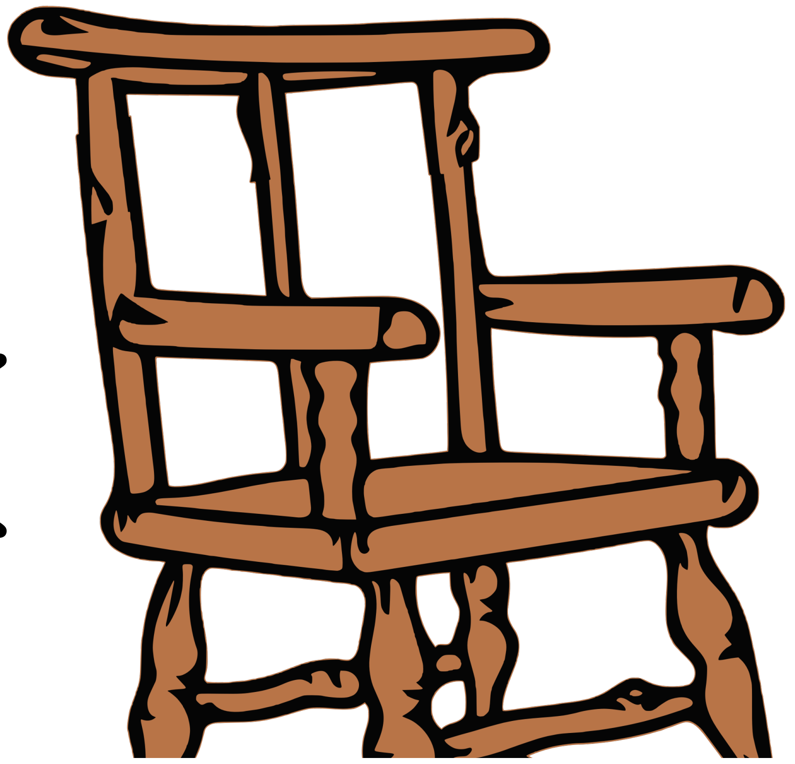
.rockers =  ;

Result



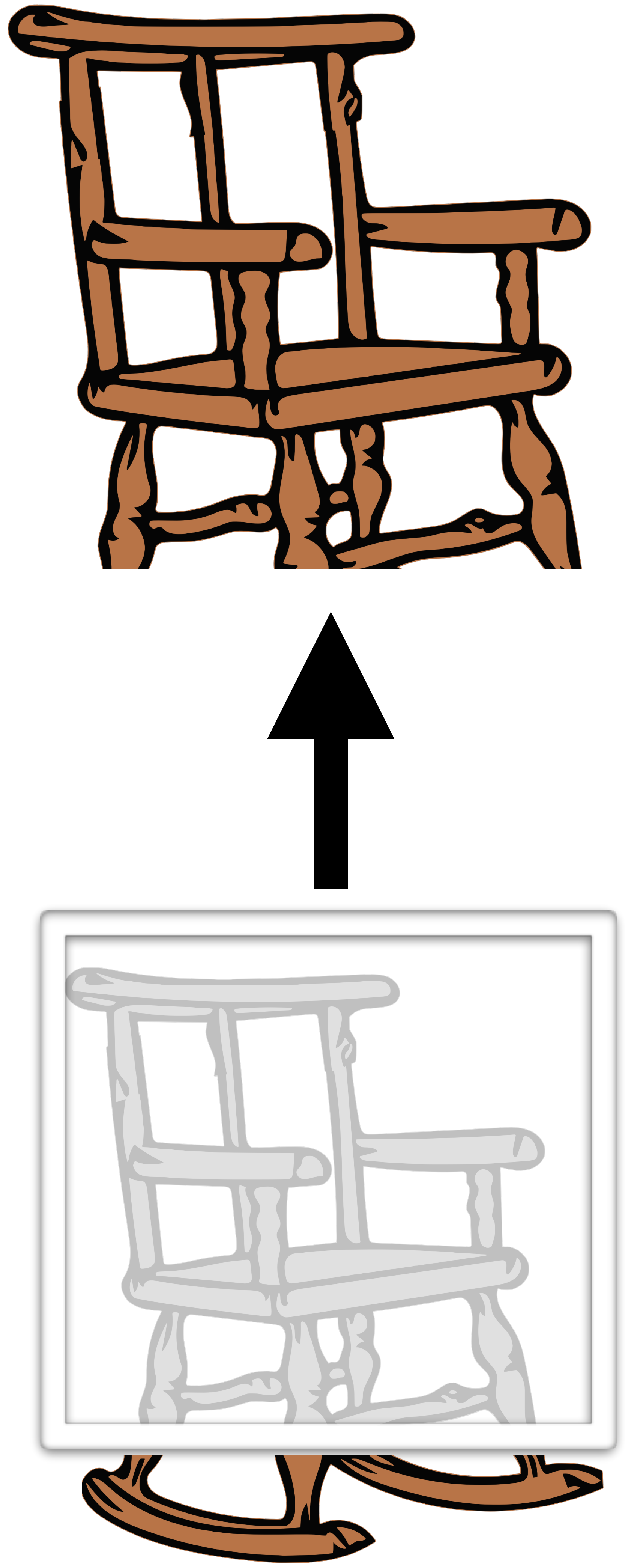


Commands

= Object.create();

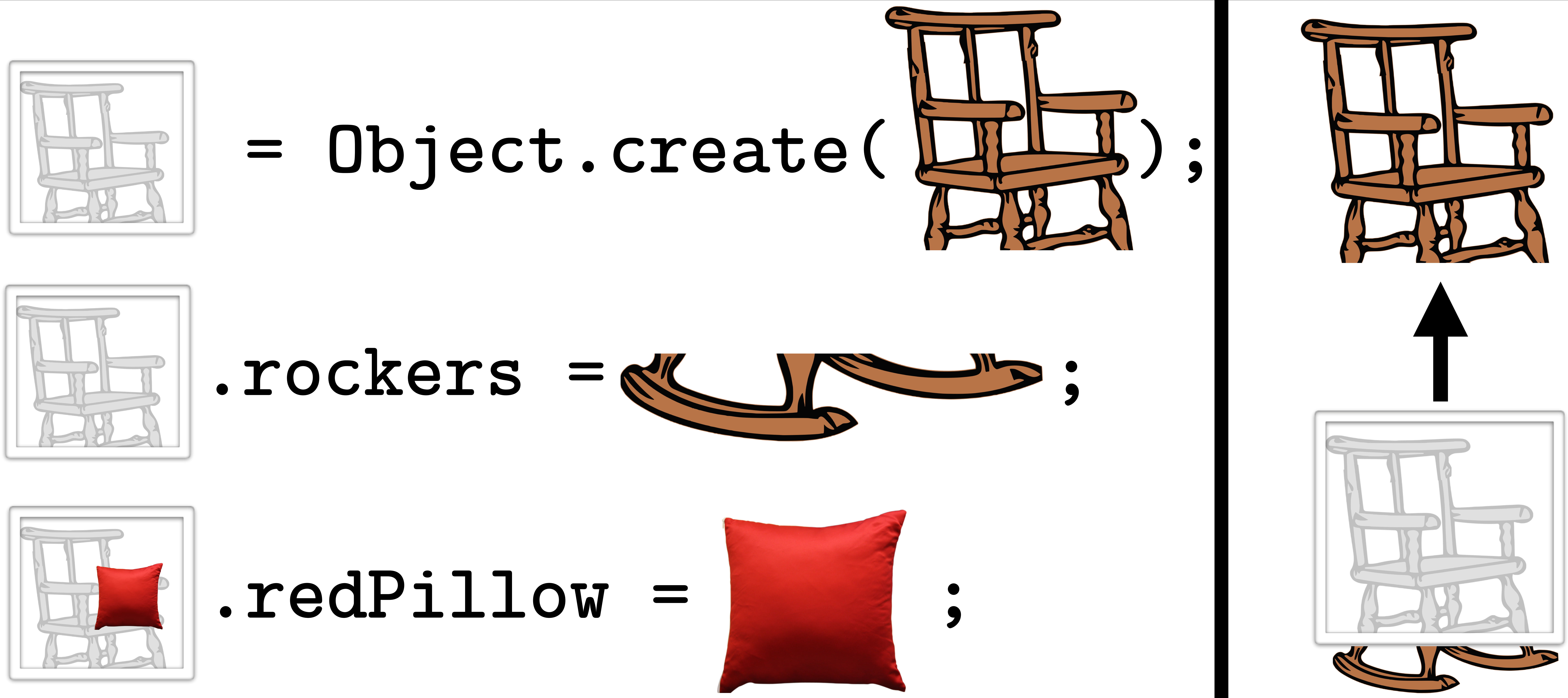
.rockers = ;

Result



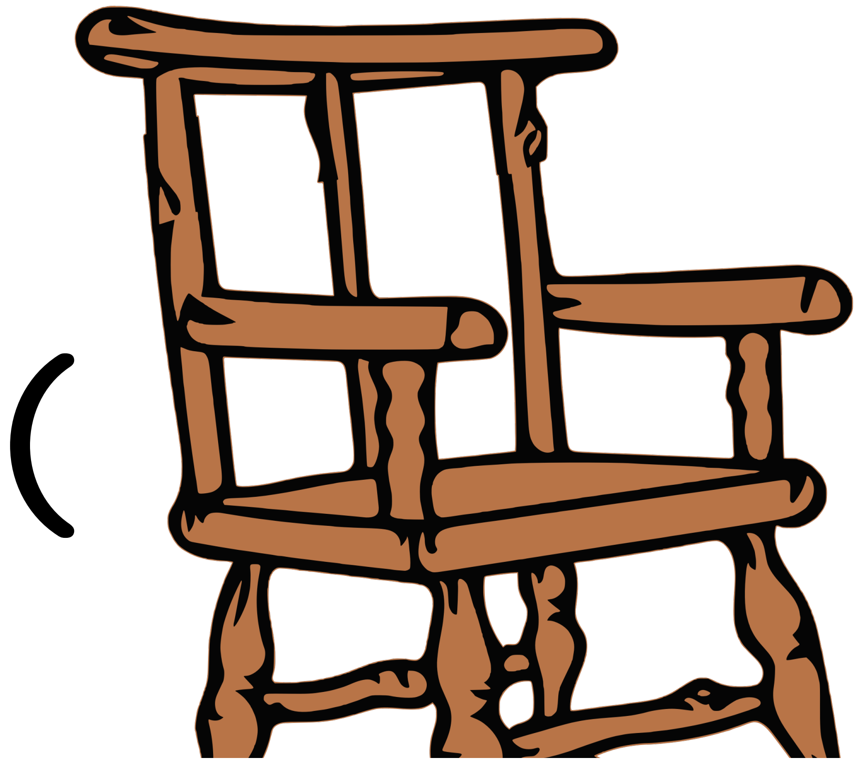
Commands

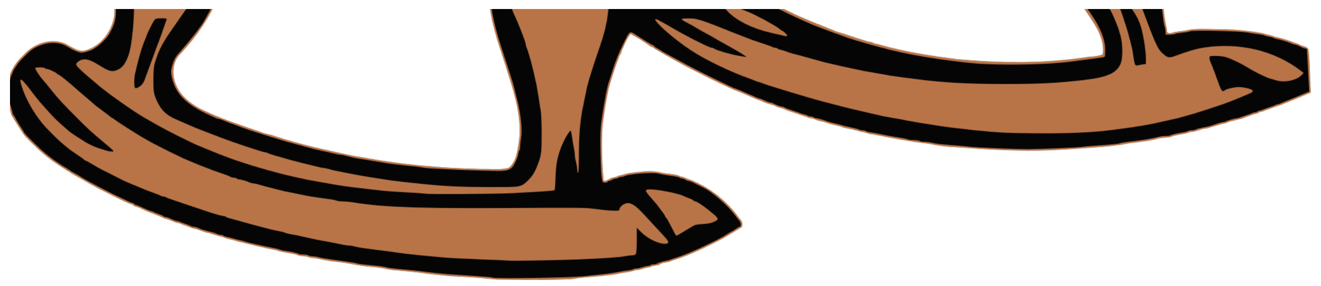
Result

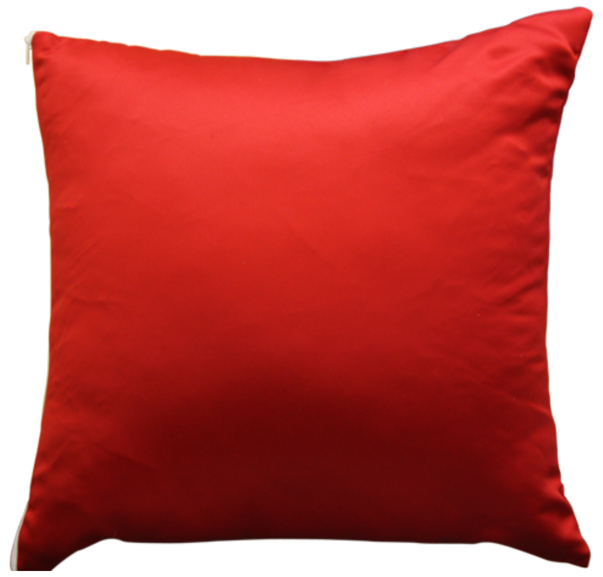


```
= Object.create( );
.rockers = ;
.redPillow = ;
```

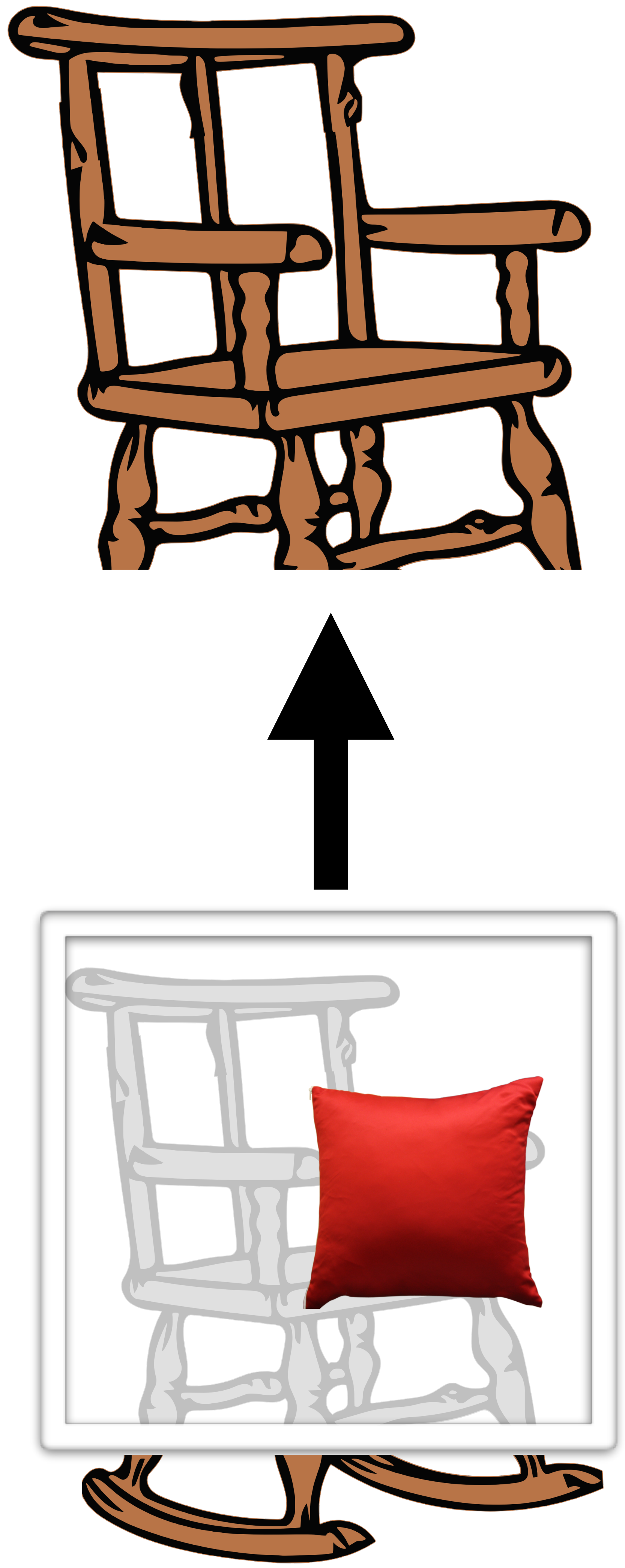
Commands

= Object.create();

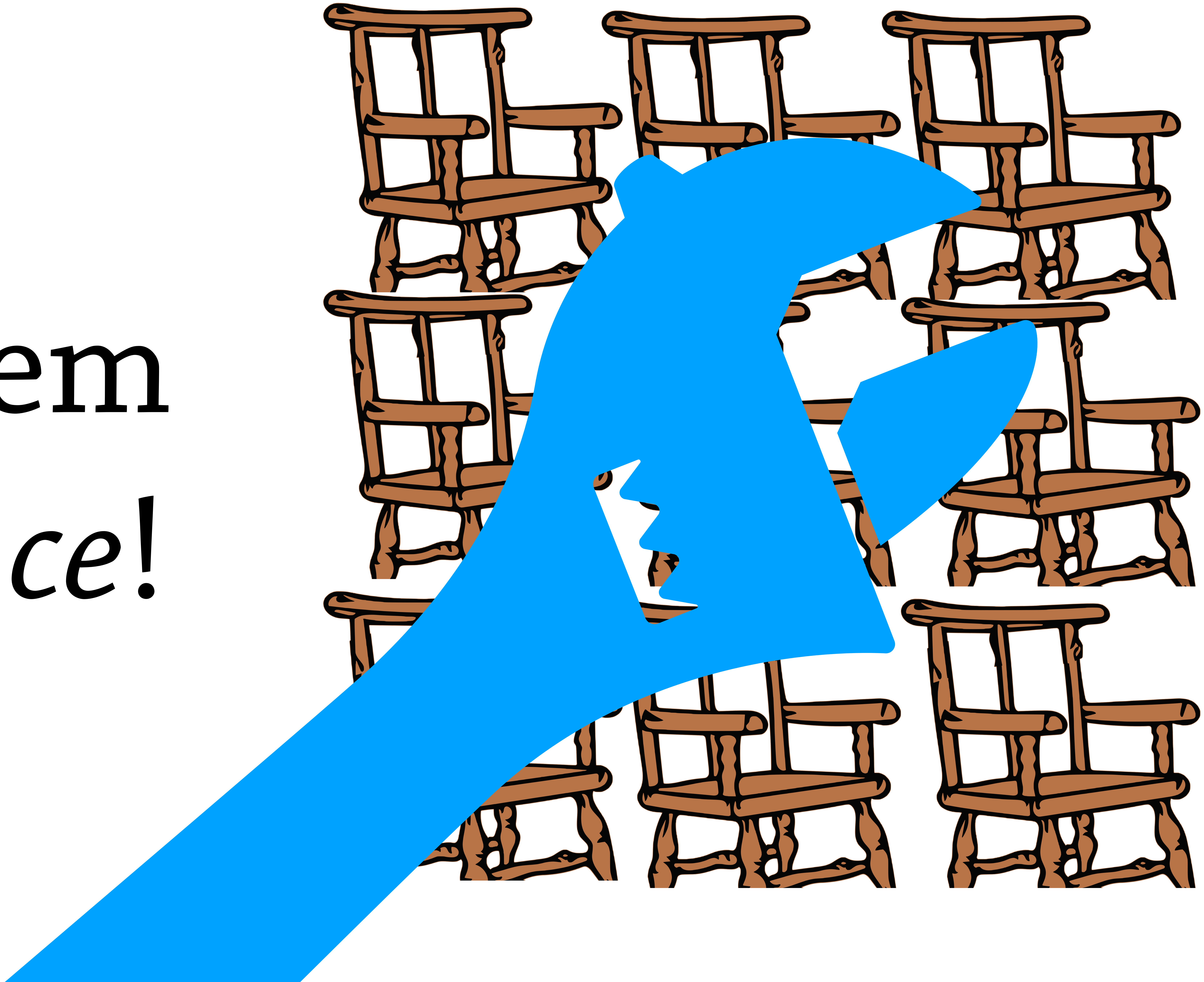
.rockers = ;

.redPillow = ;

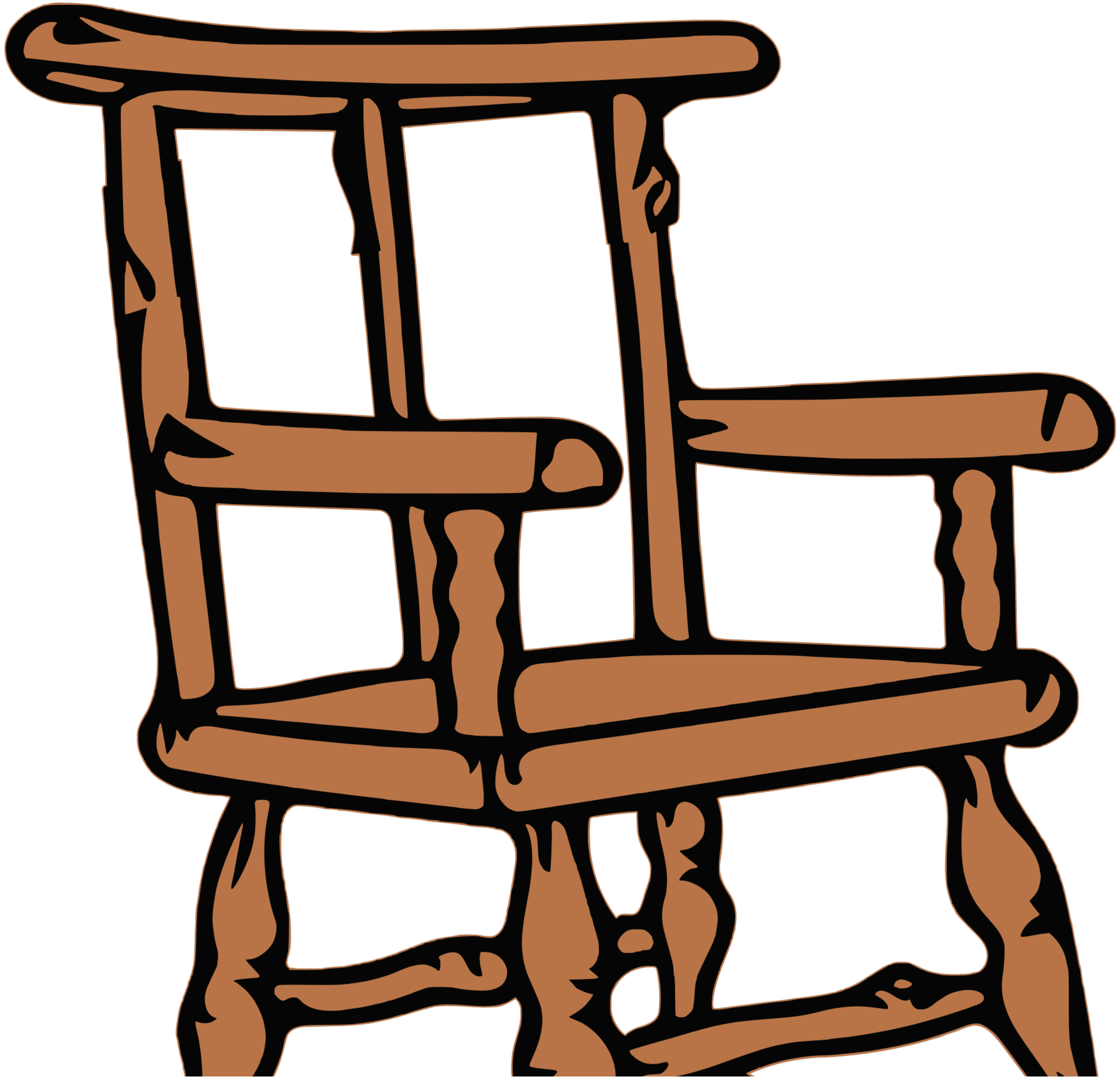
Result

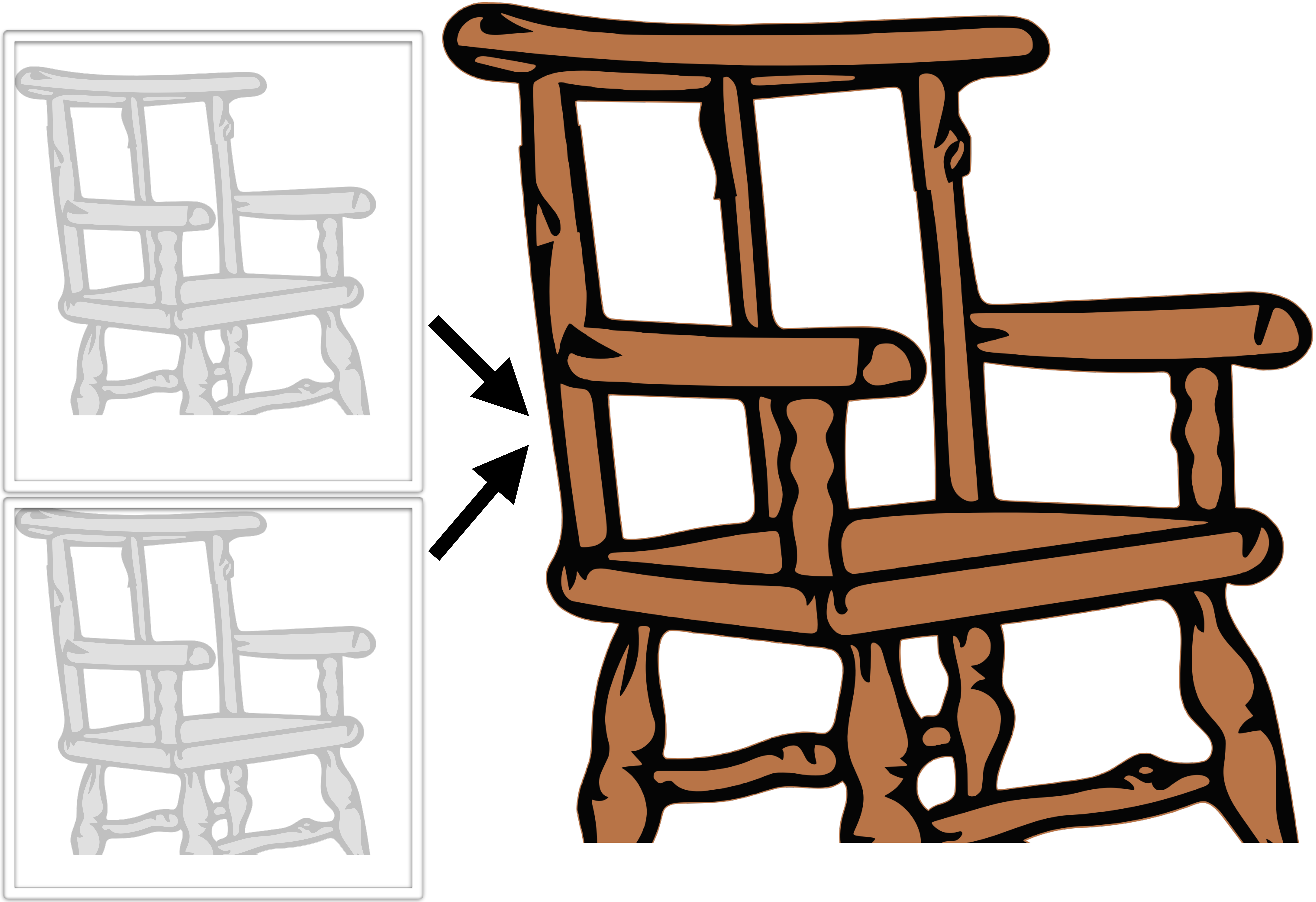


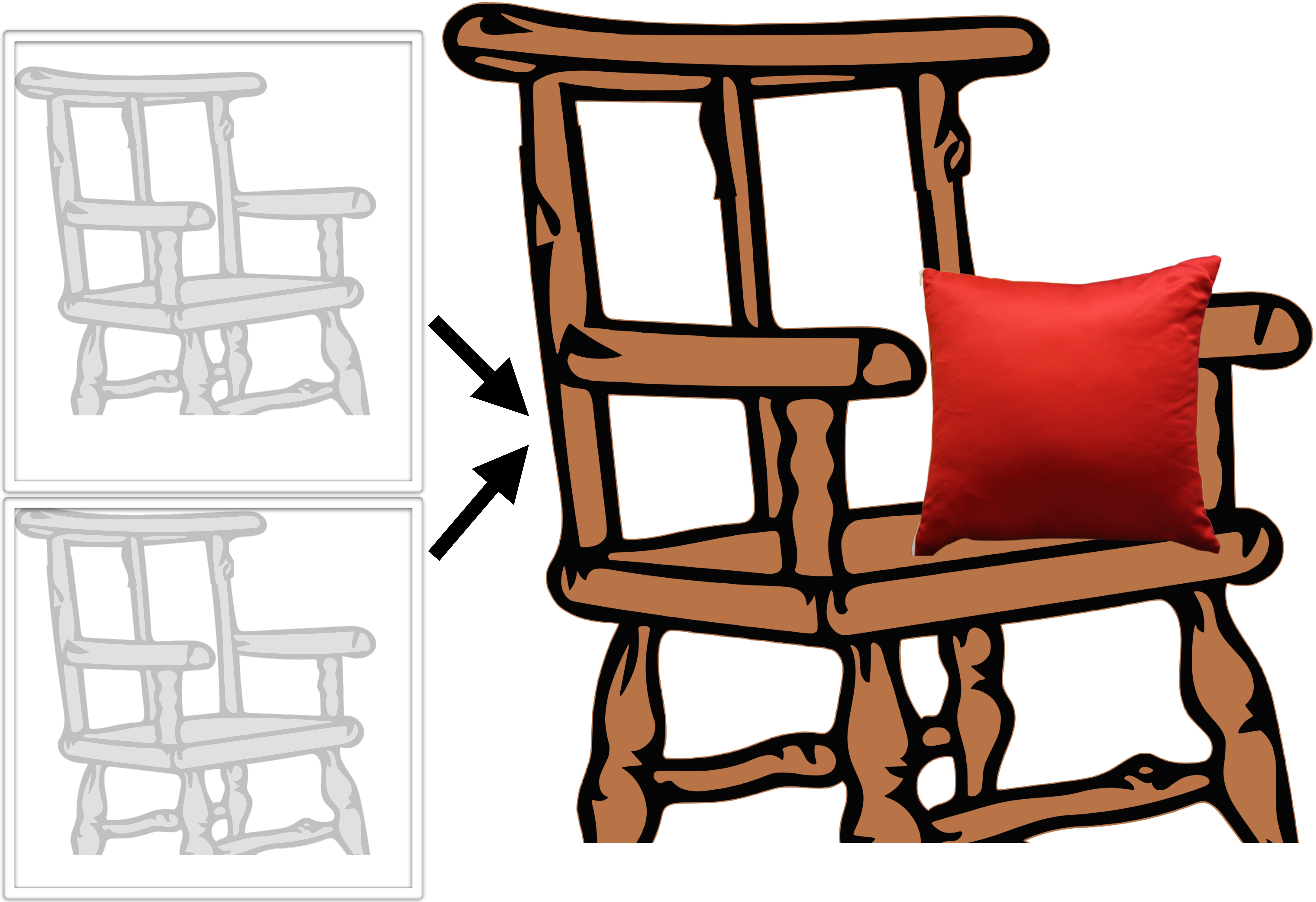
…fix them
all at once!



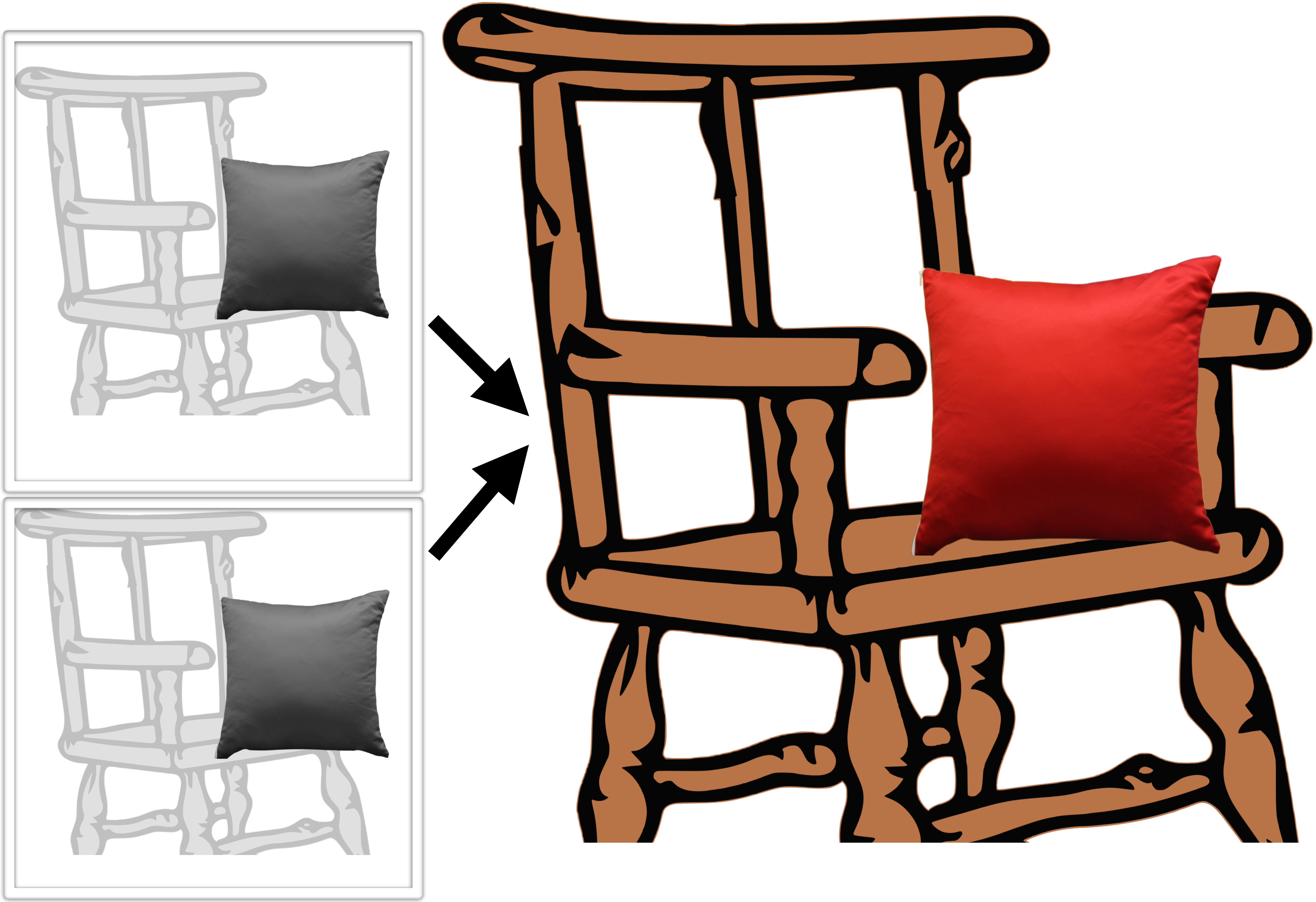
Shared State



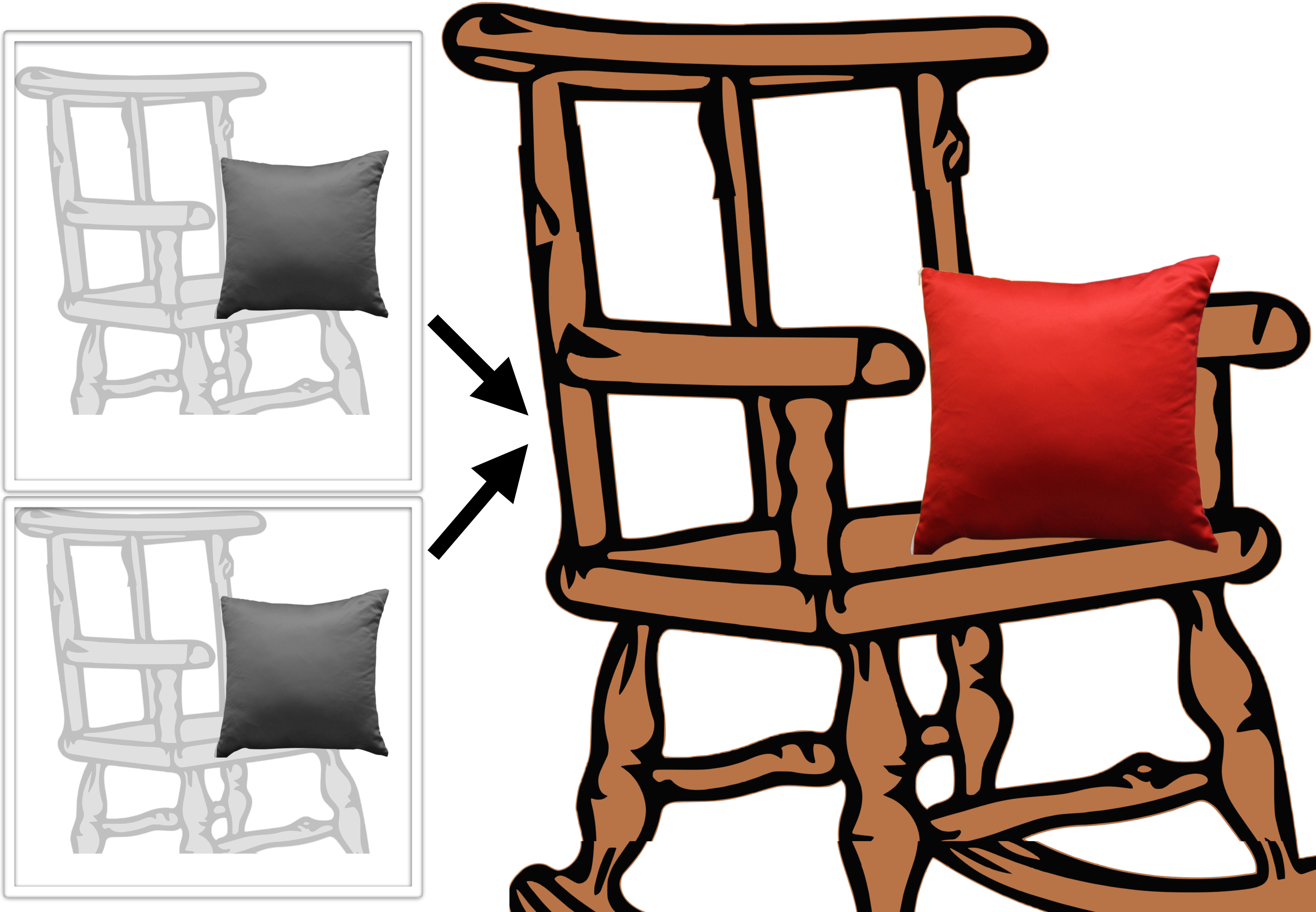




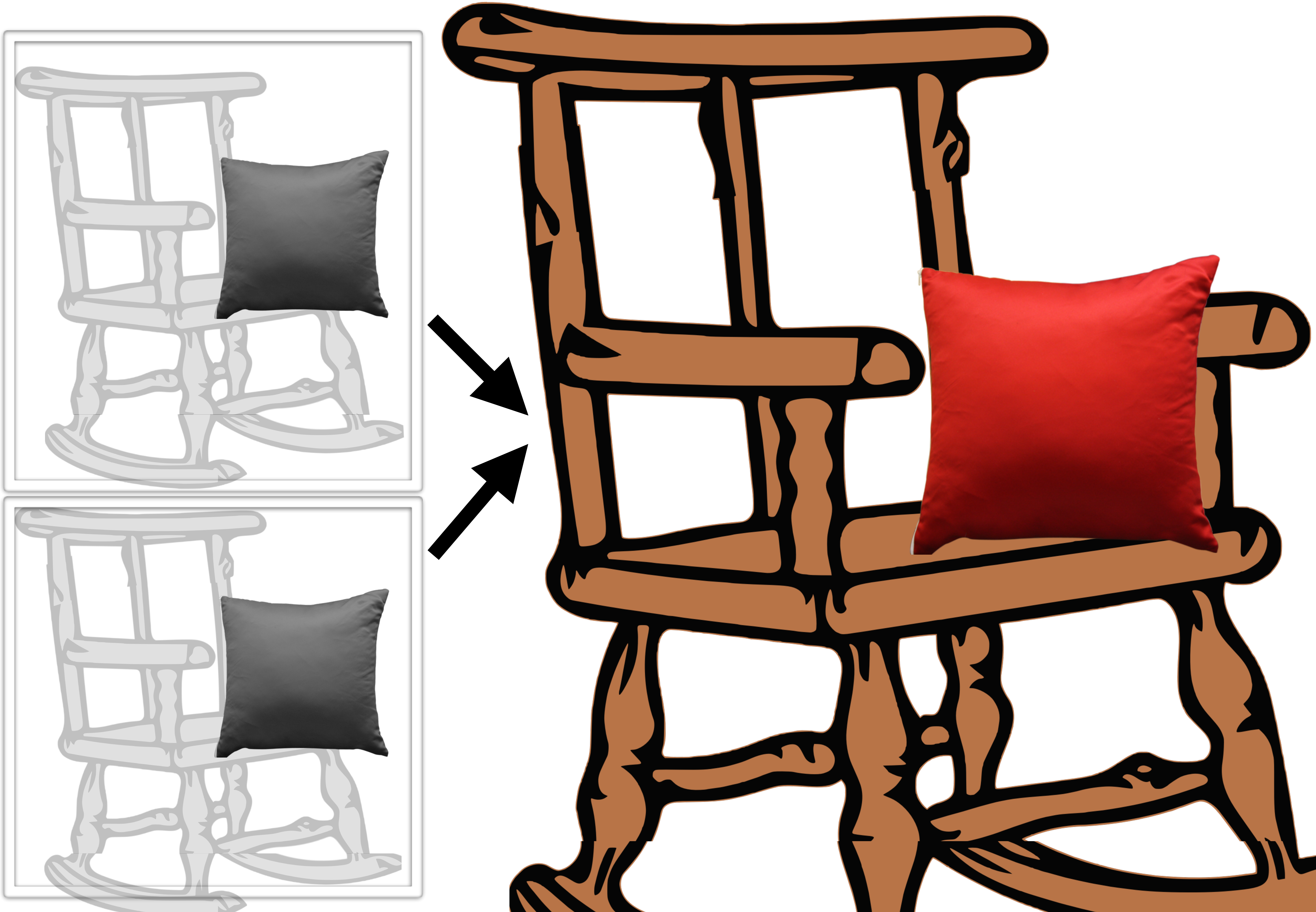
Share

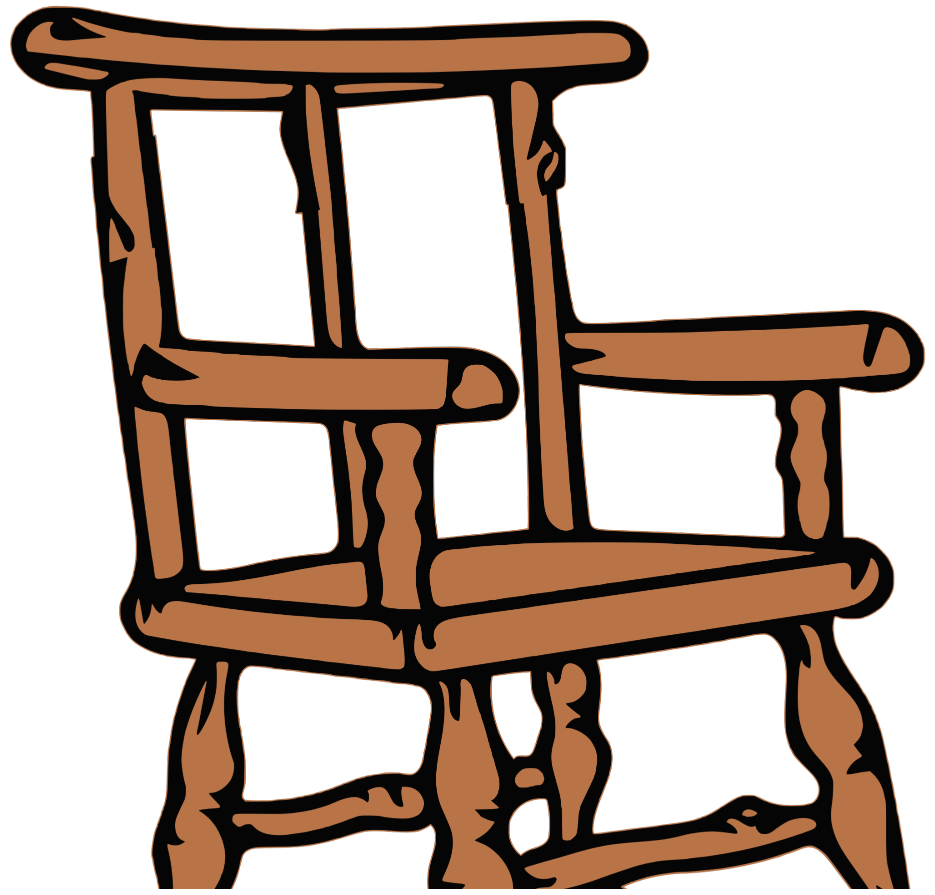


Share



Share

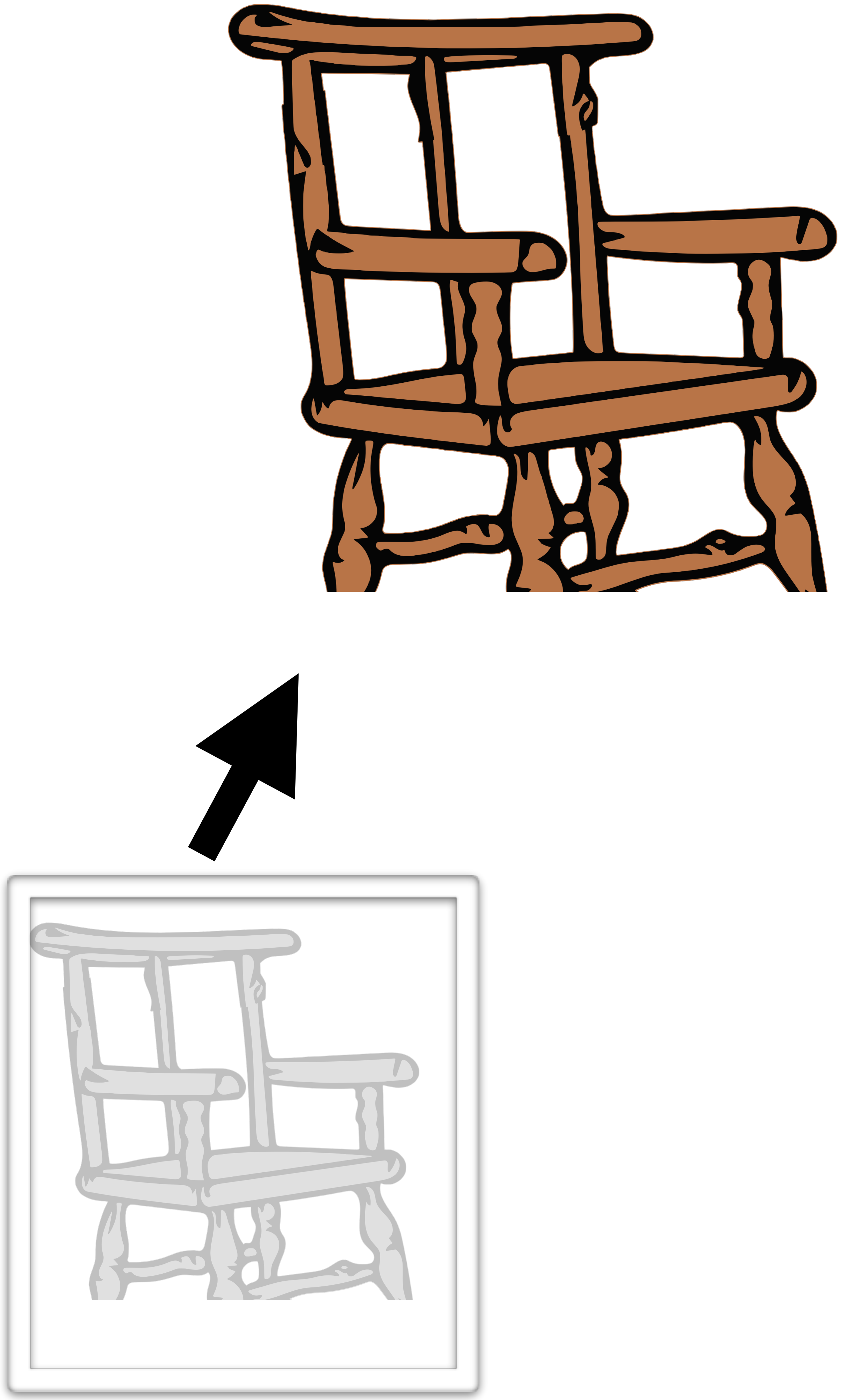


Commands	Result
	

Commands

`= Object.create(` `);`

Result

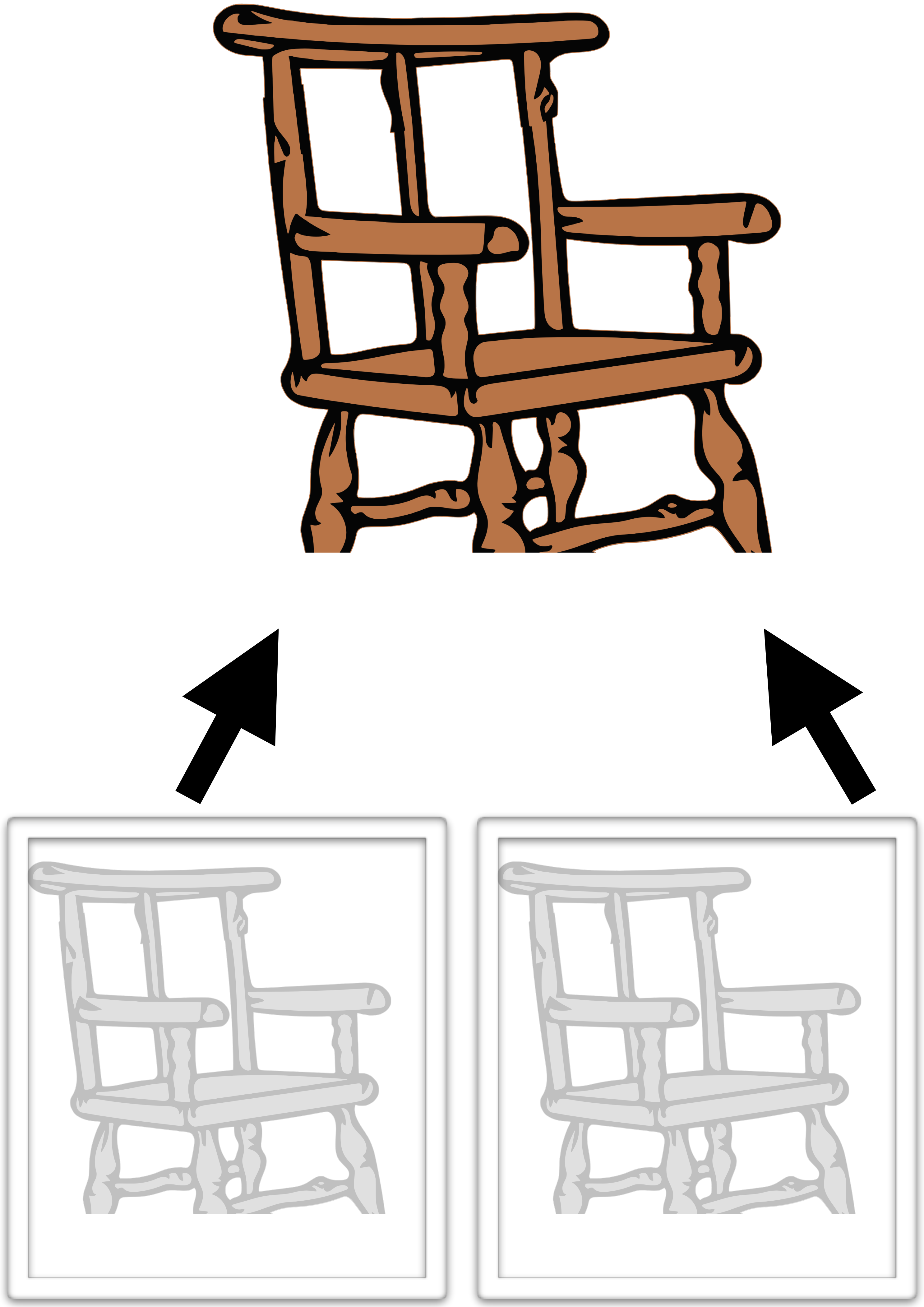


Commands

Result

= Object.create( );

= Object.create( );



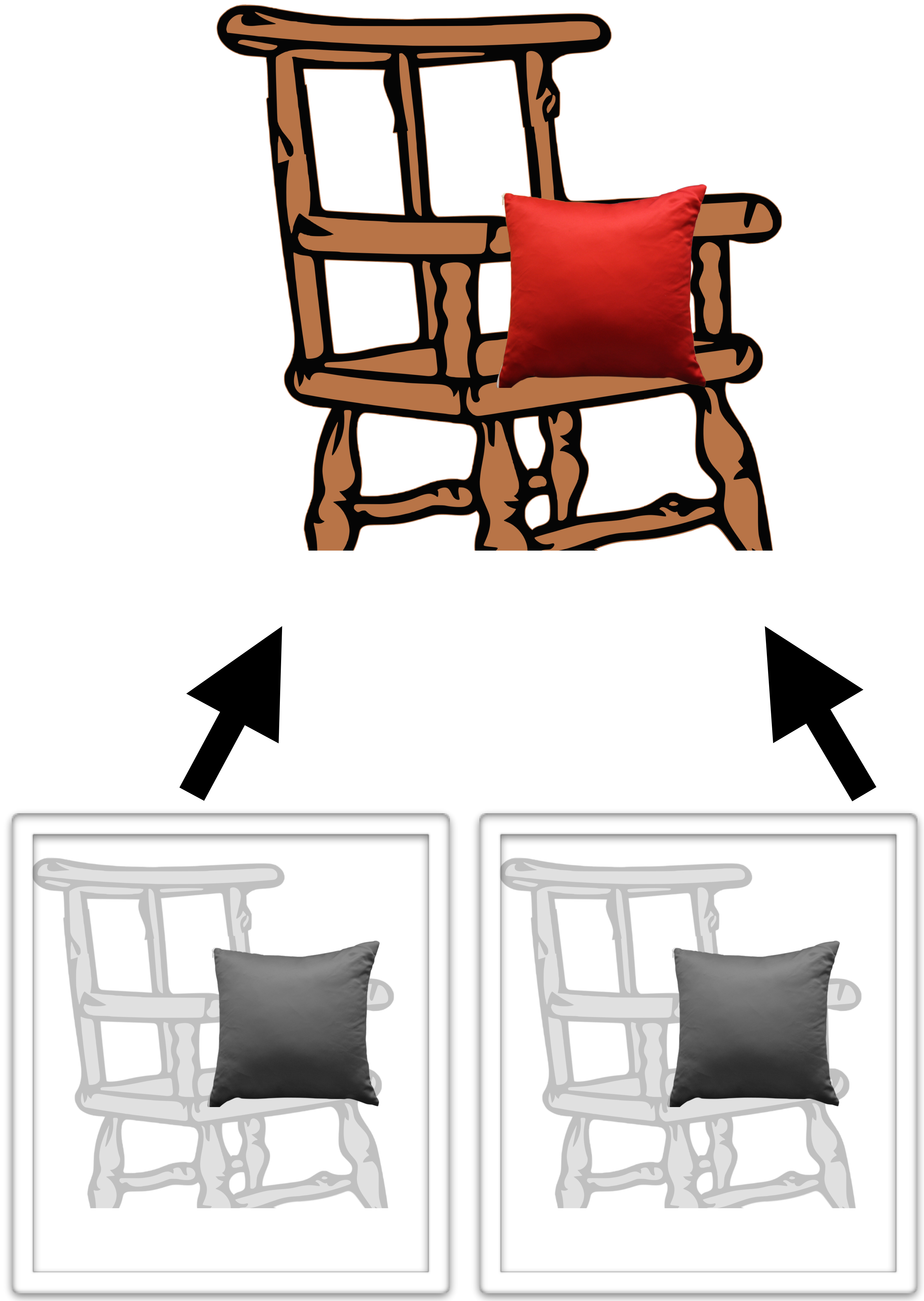
Commands

= Object.create( );

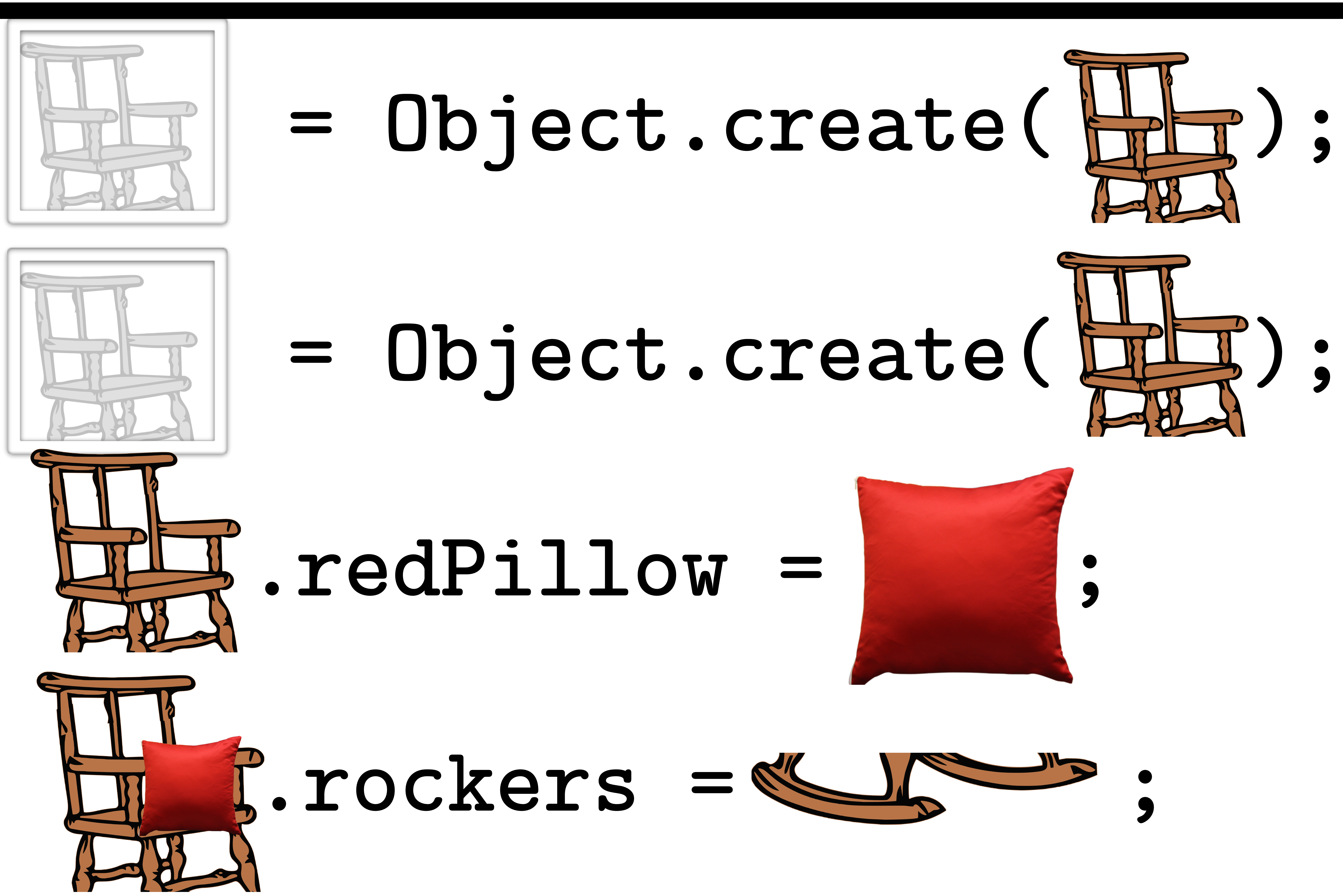
= Object.create( );

.redPillow = ;

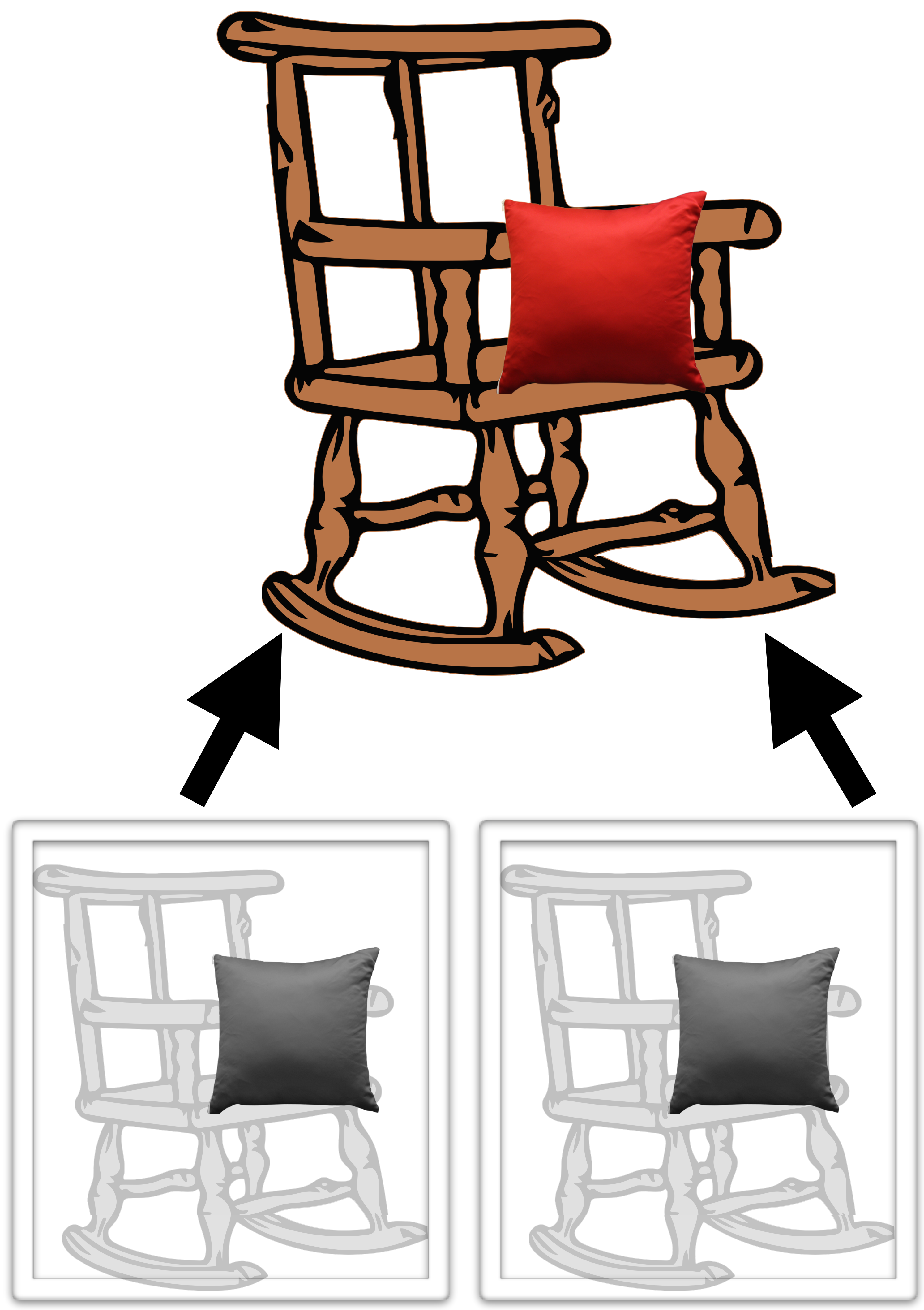
Result



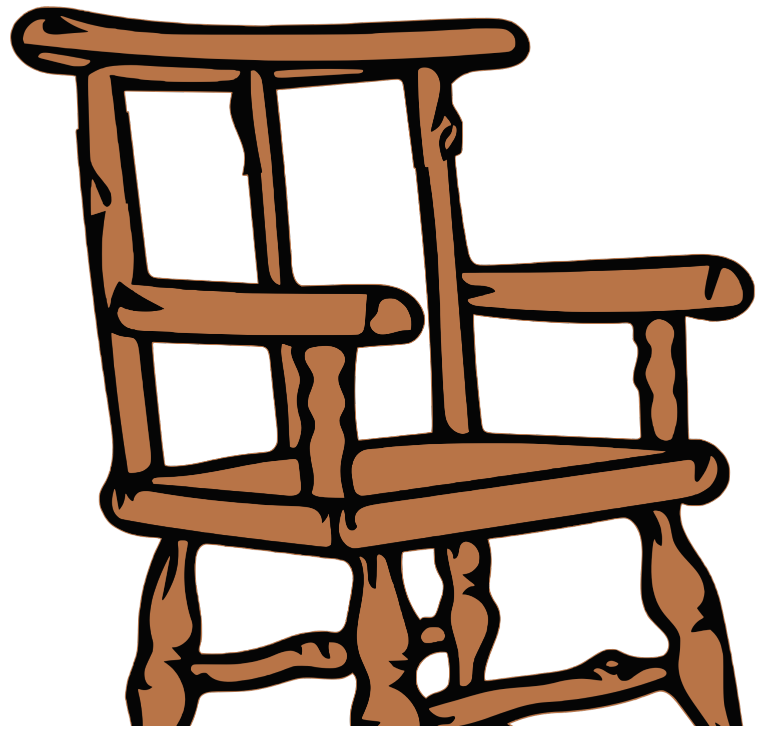
Commands | Result



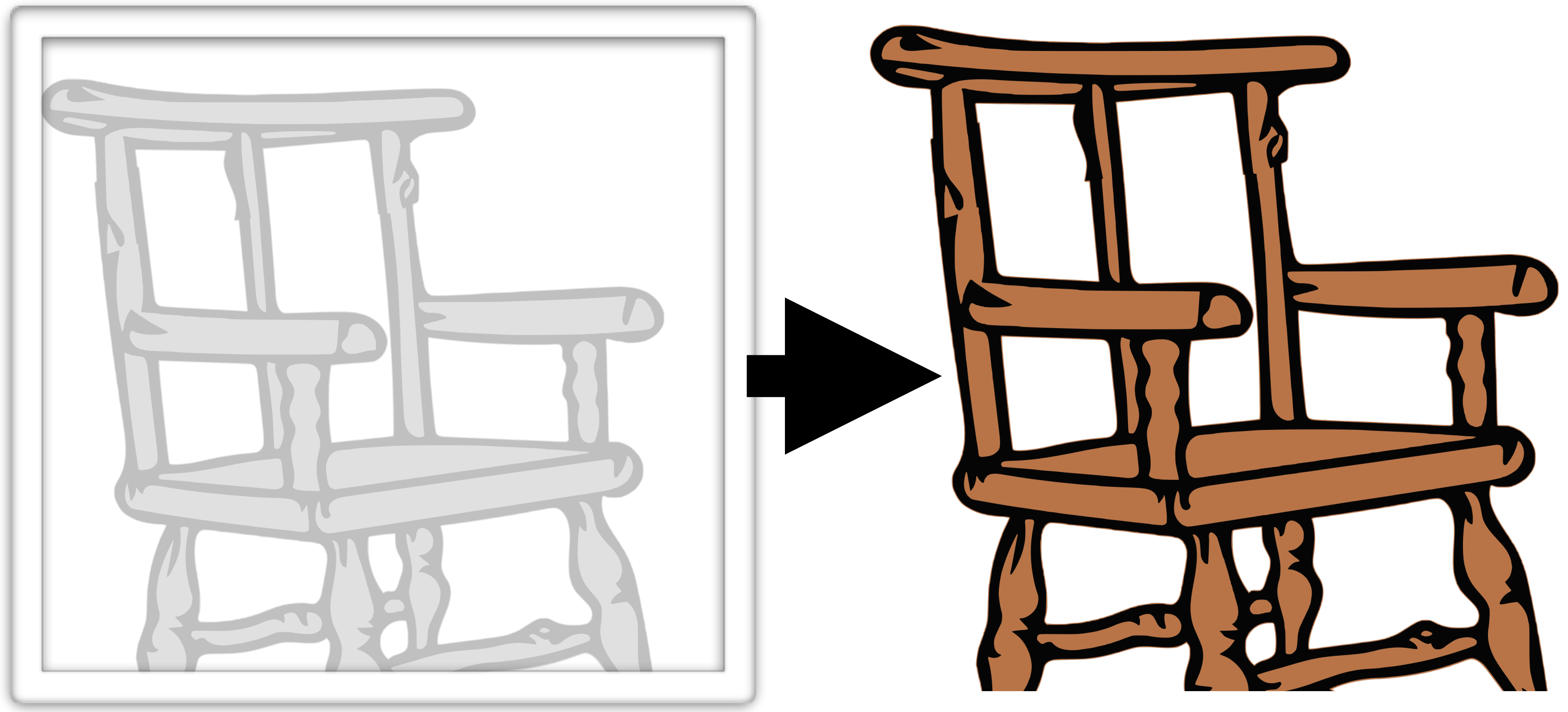
```
= Object.create( );
= Object.create( );
.redPillow = ;
.rockers = ;
```



Objects are **prototypes**

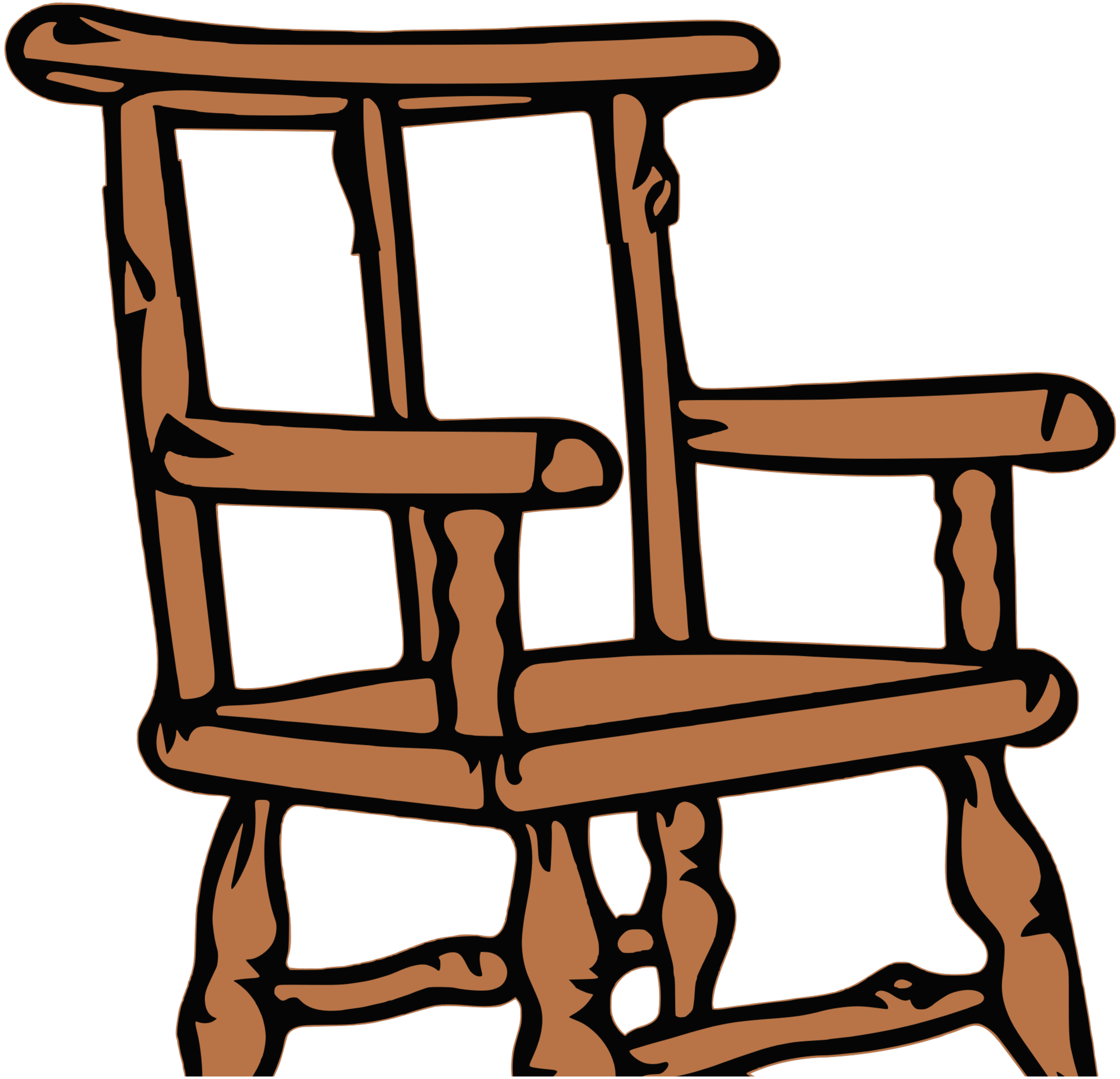


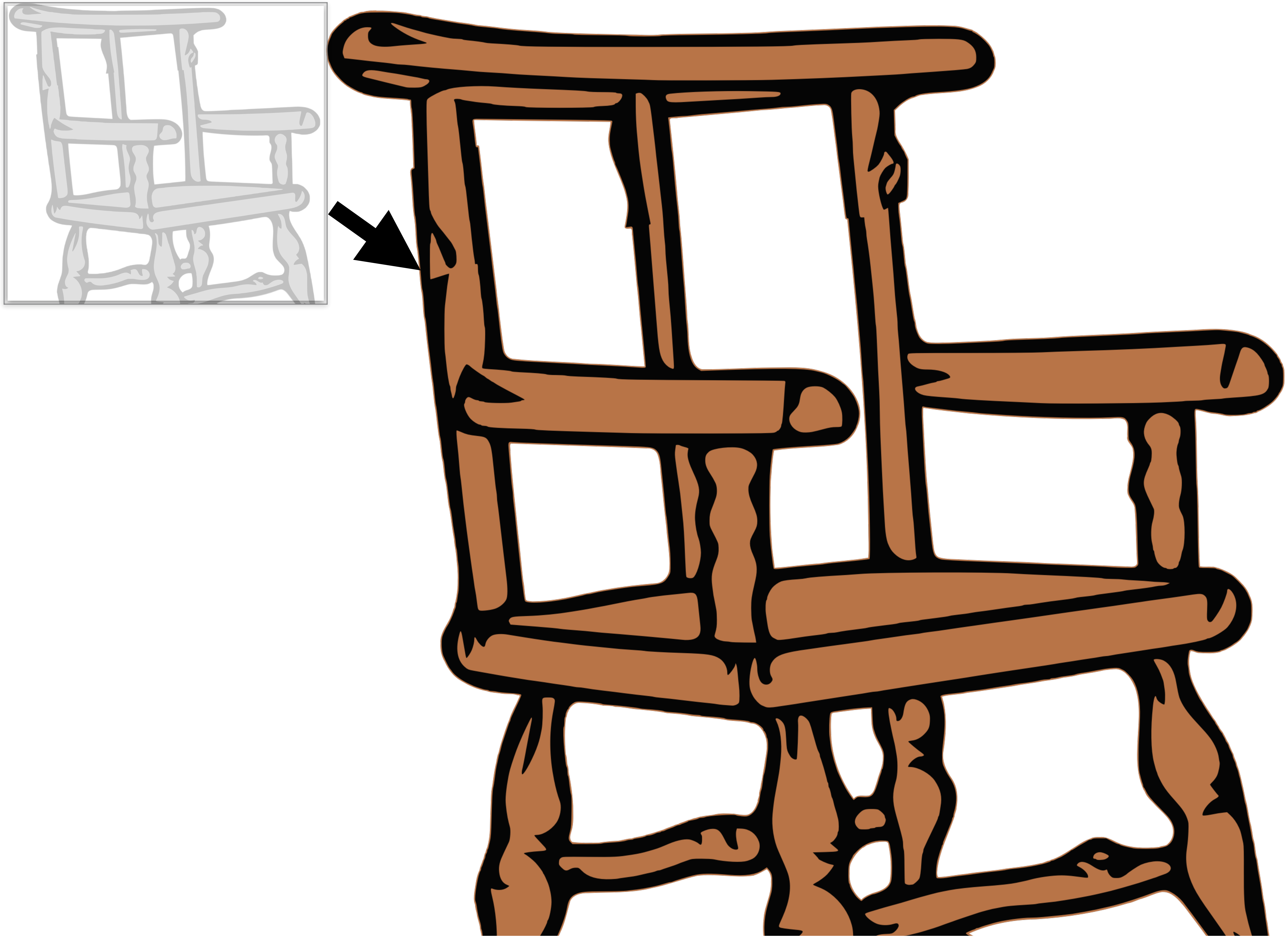
Objects are **prototypes**

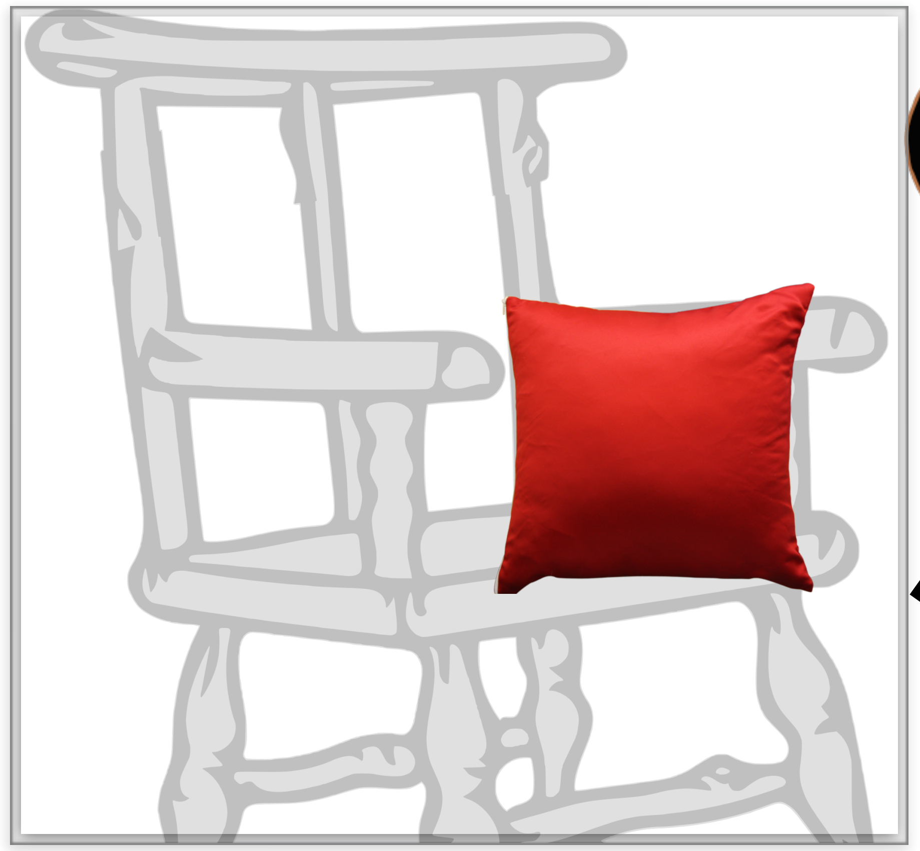


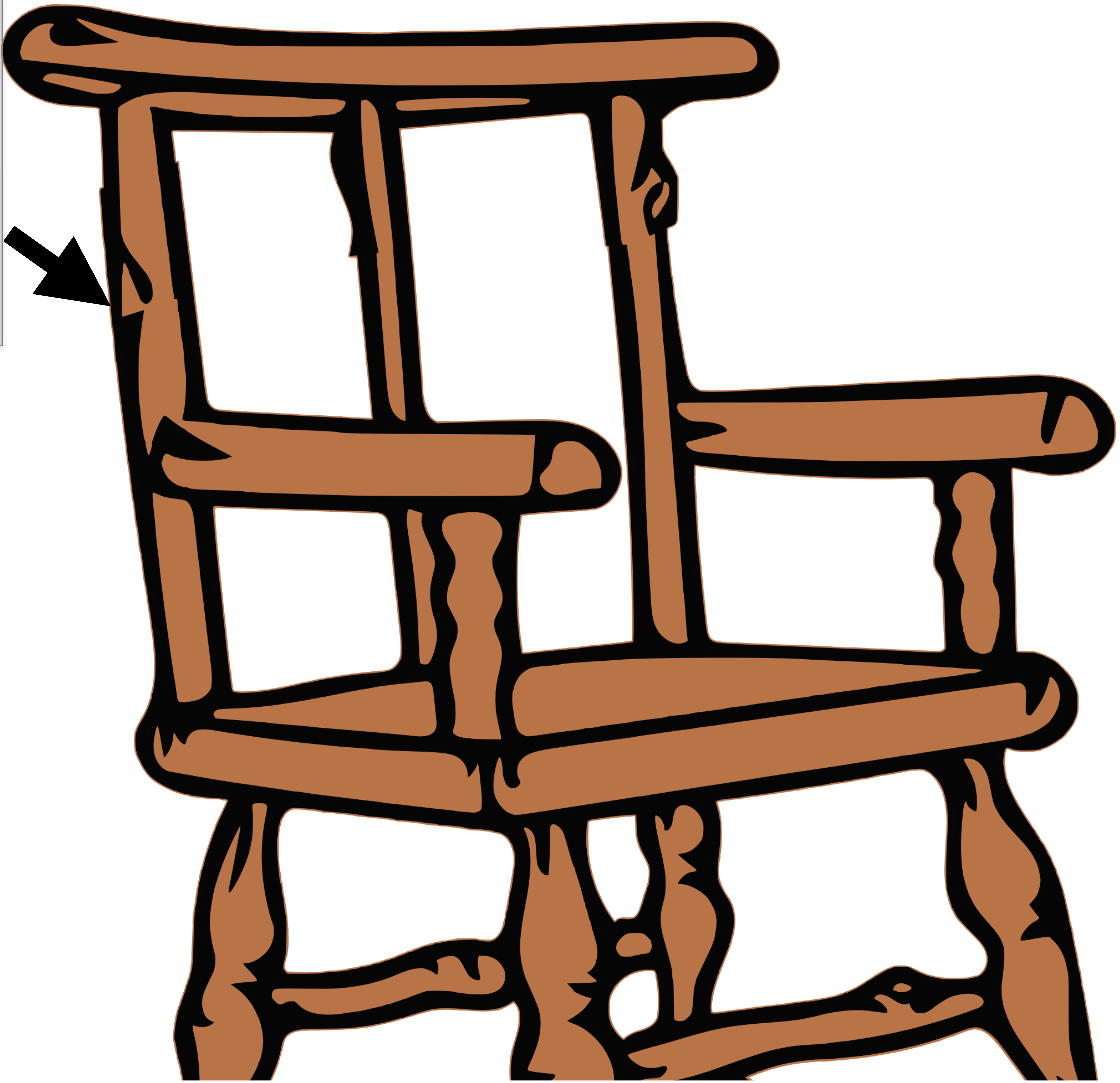
Objects =

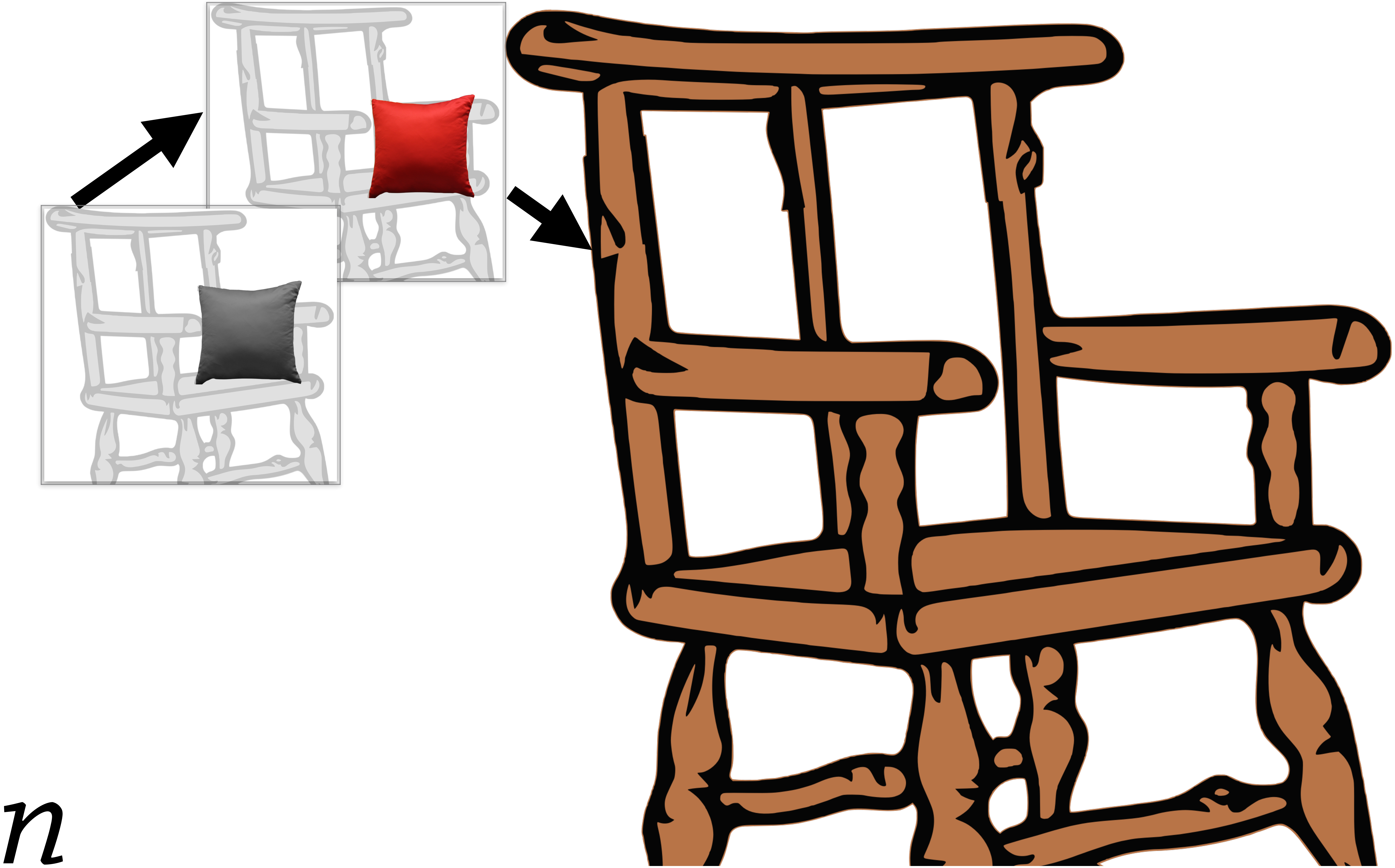
prototypes



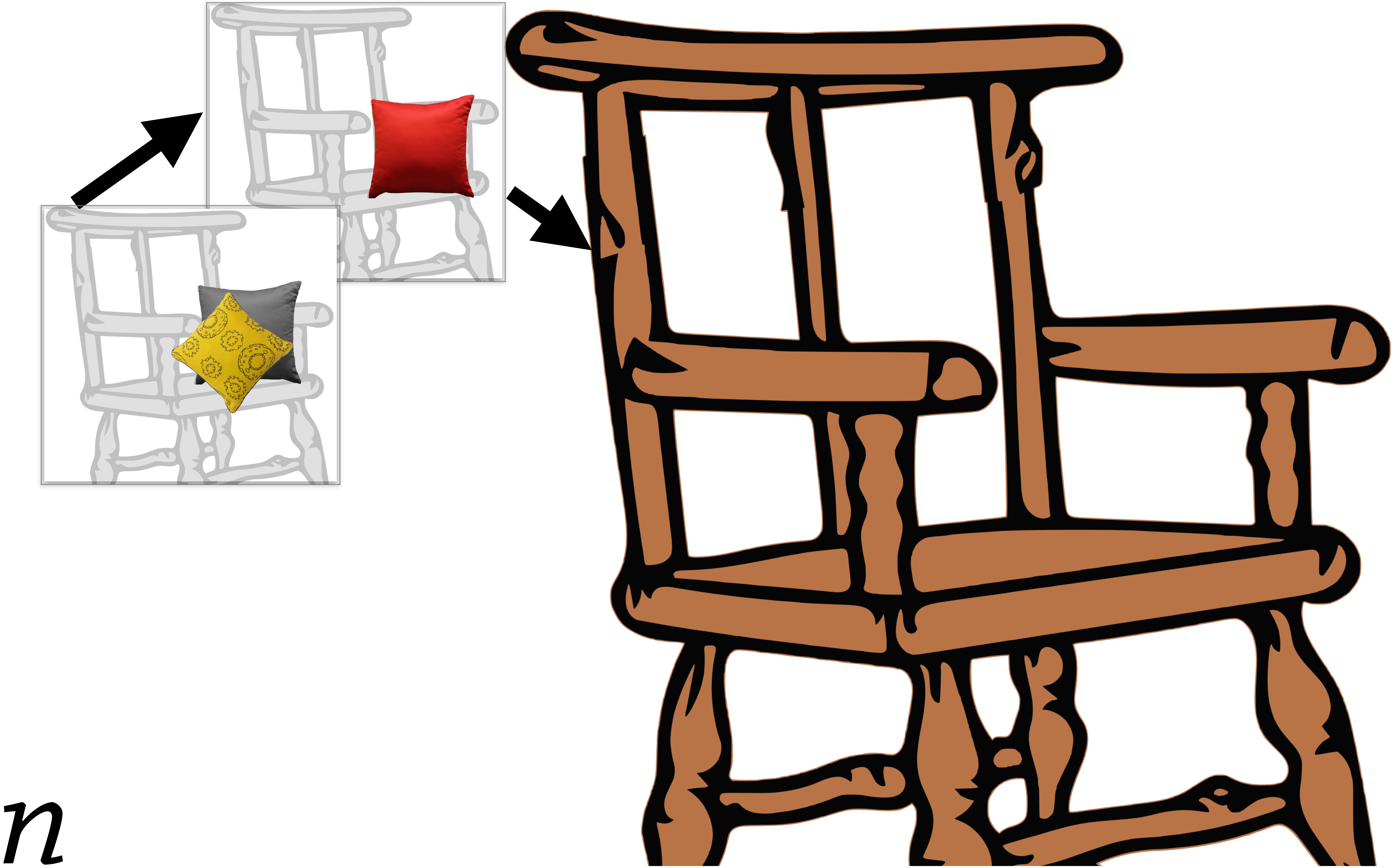




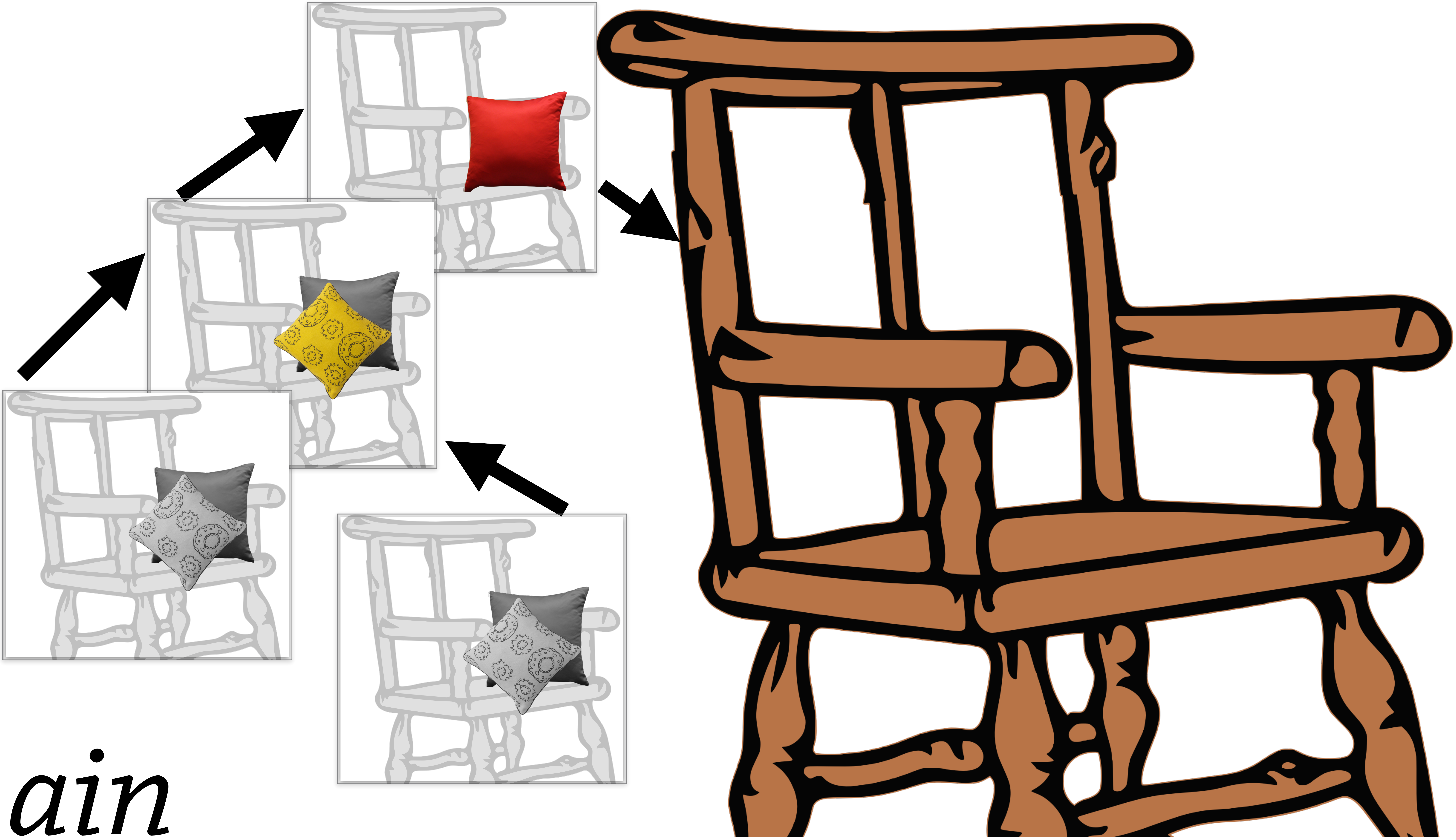




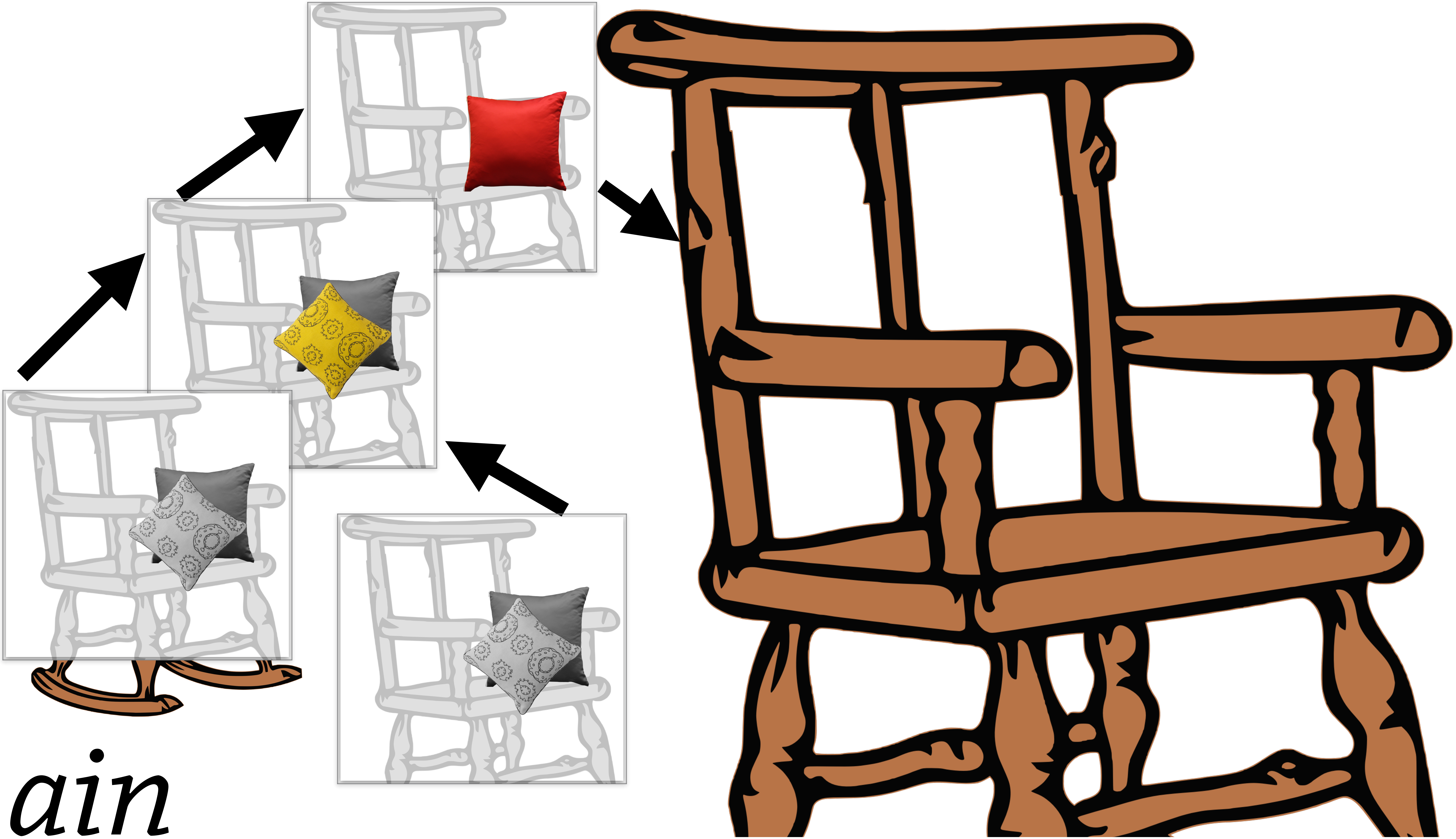
Chain



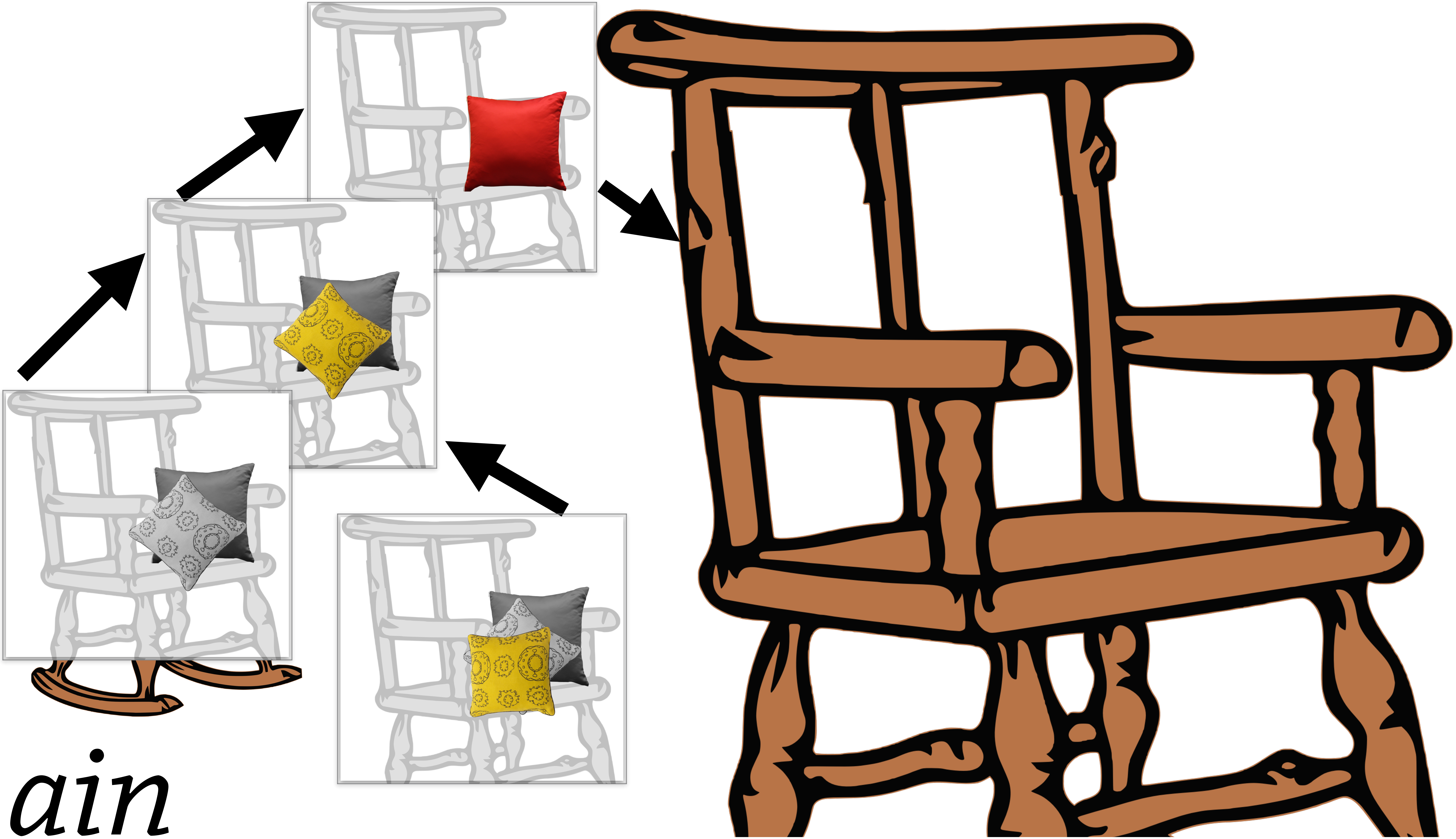
Chain



Chain



Chain

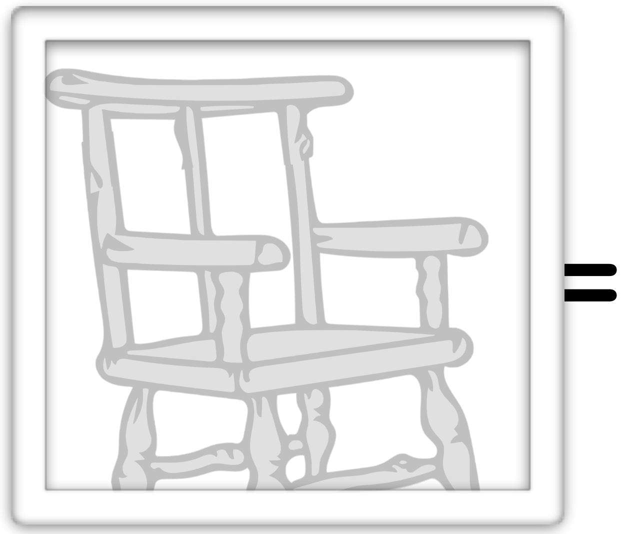
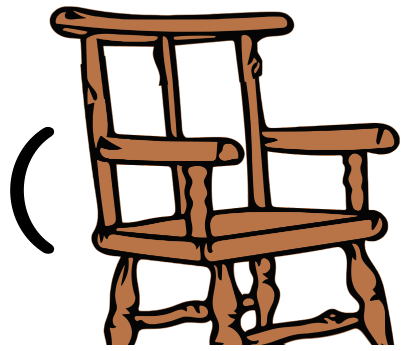


Chain

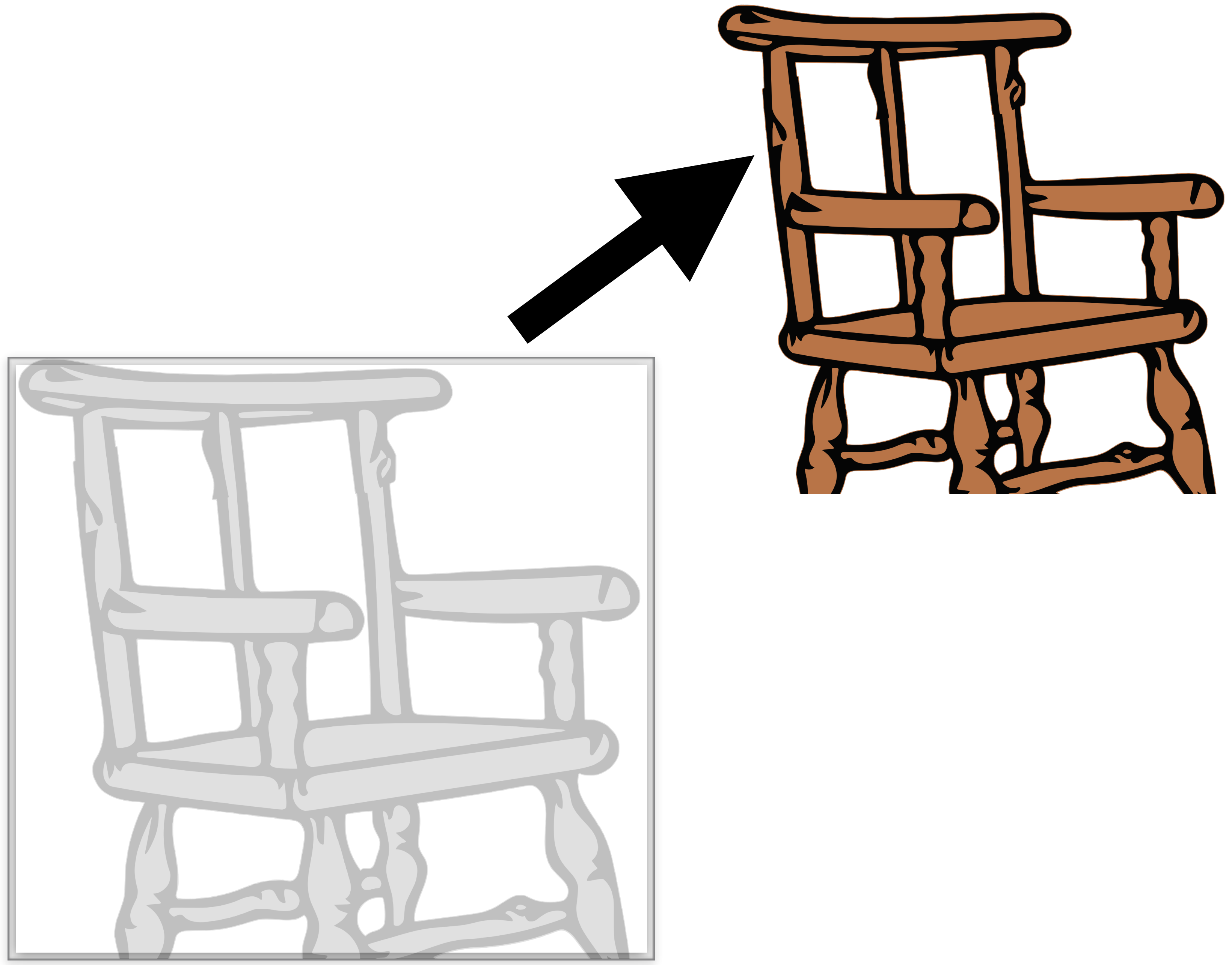
Differential Inheritance

Commands	Result

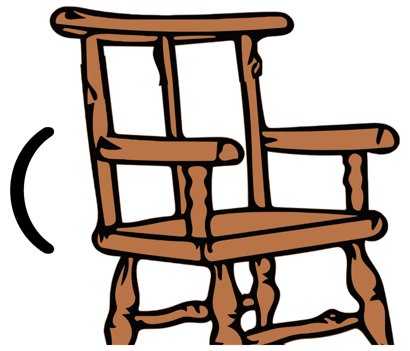
Commands

 = Object.create();

Result

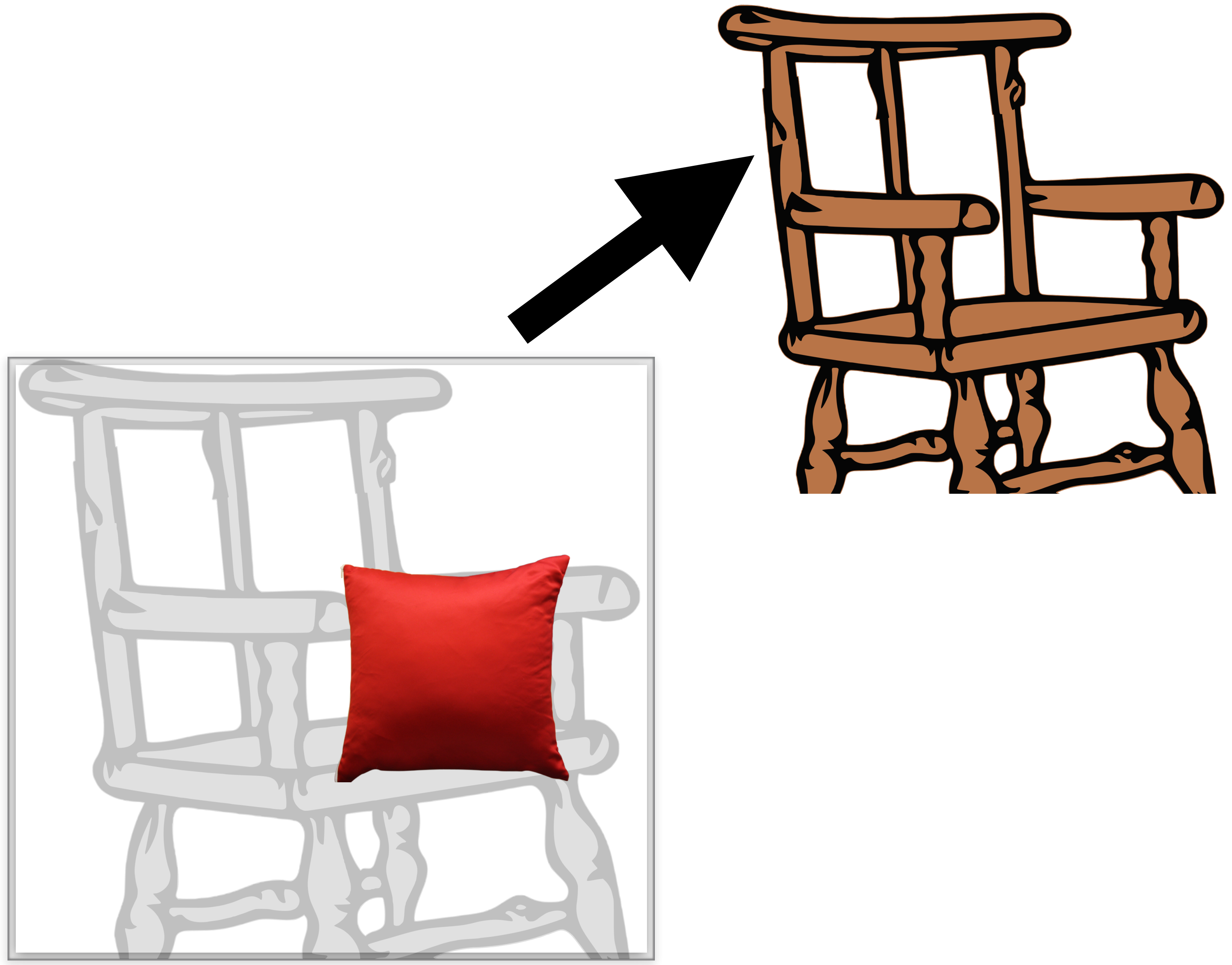


Commands

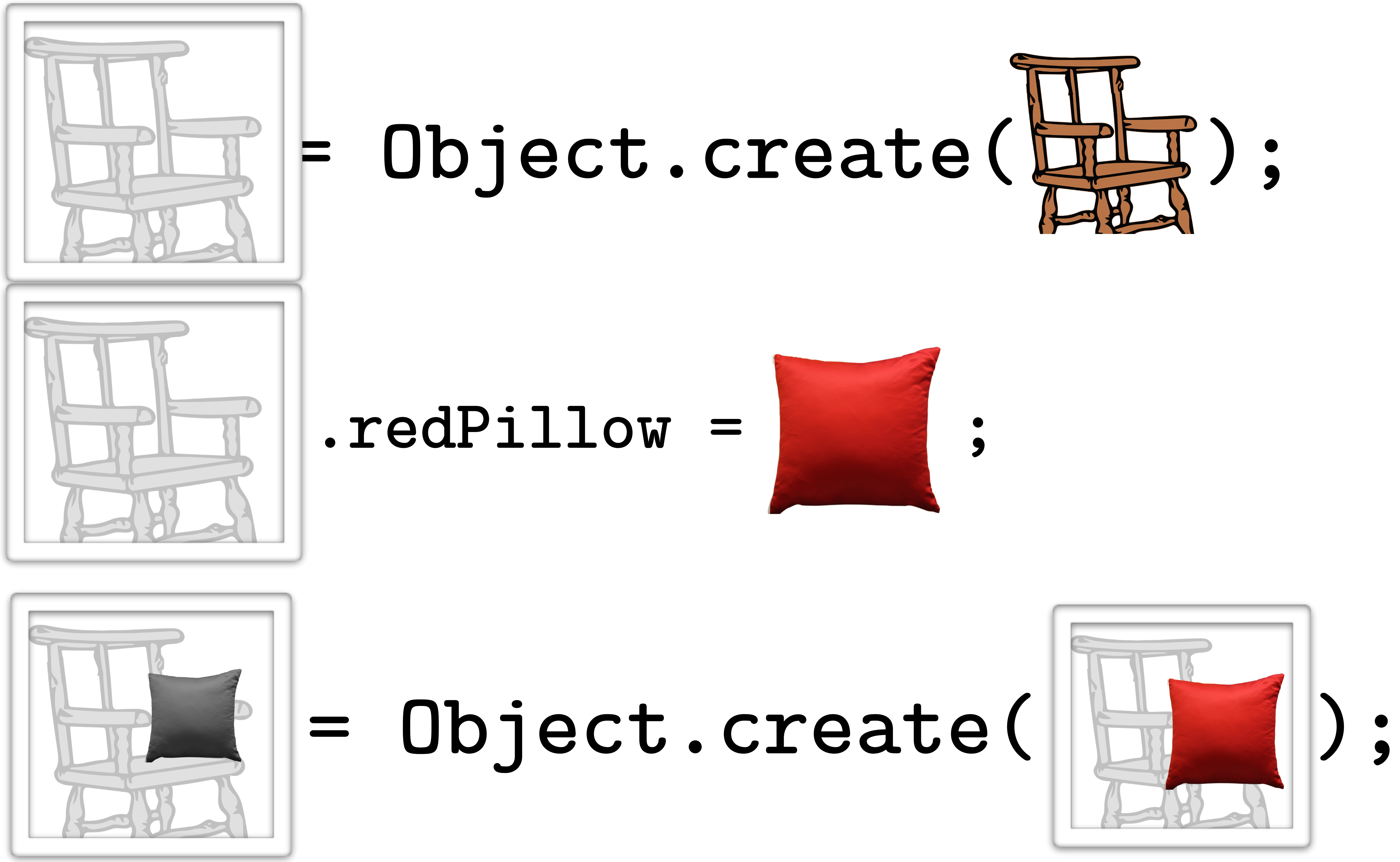
= Object.create();

.redPillow = ;

Result

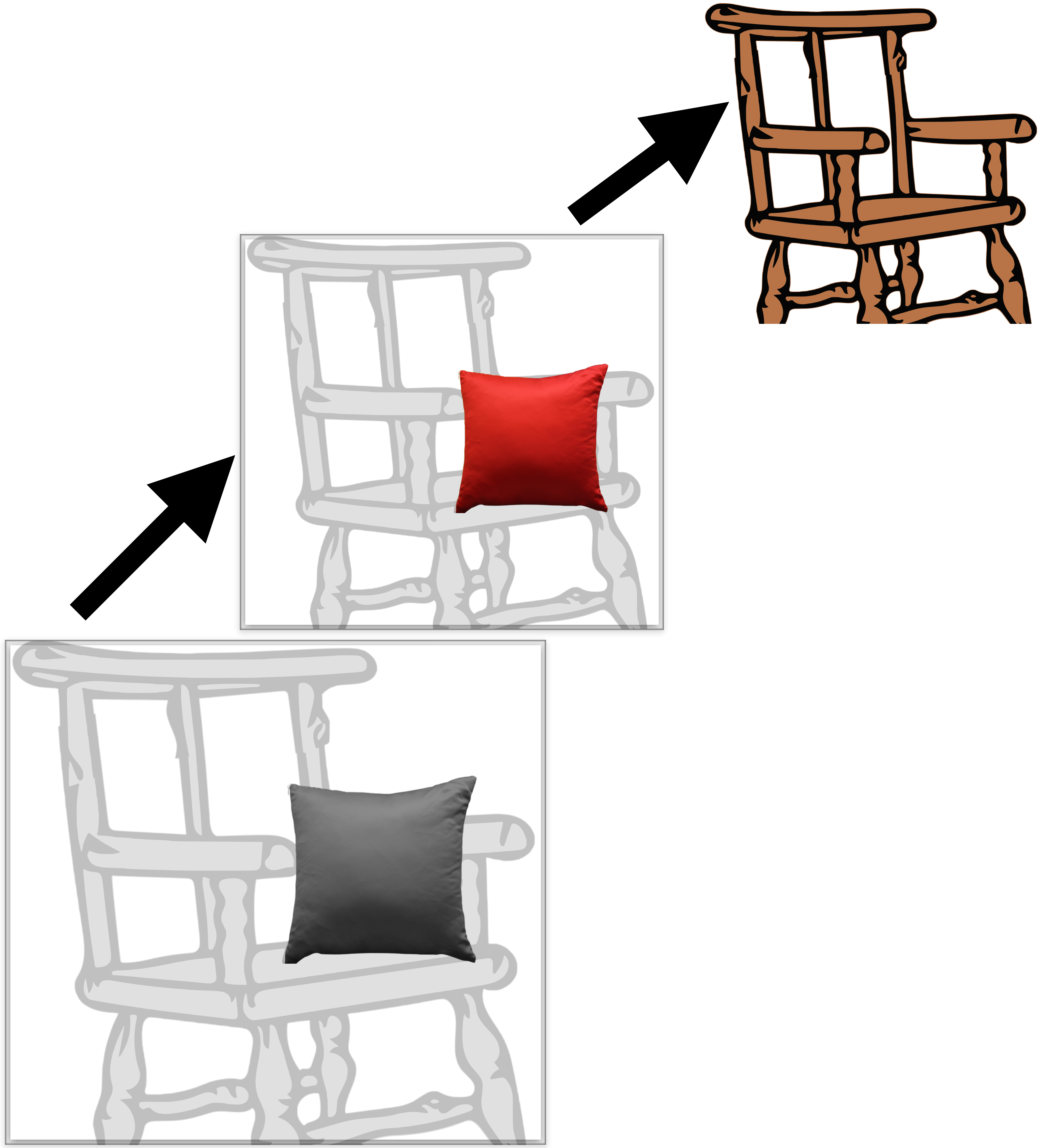


Commands

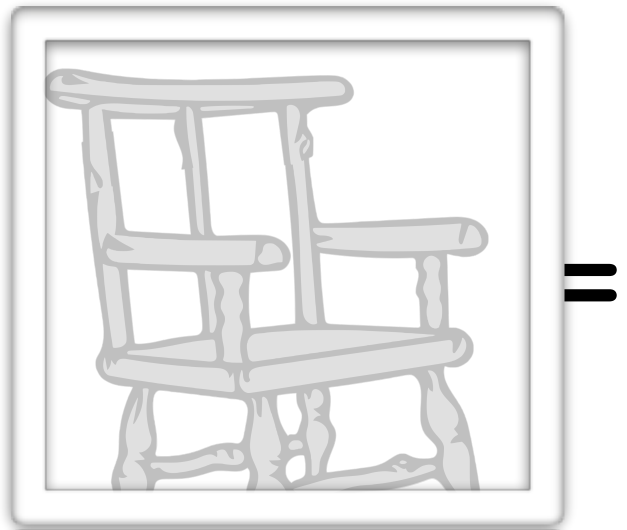
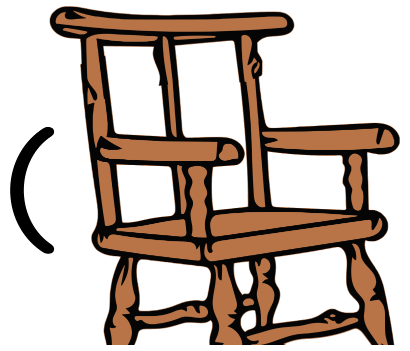


```
= Object.create(  );

.redPillow =   ;

= Object.create(  );
```

Result



Commands

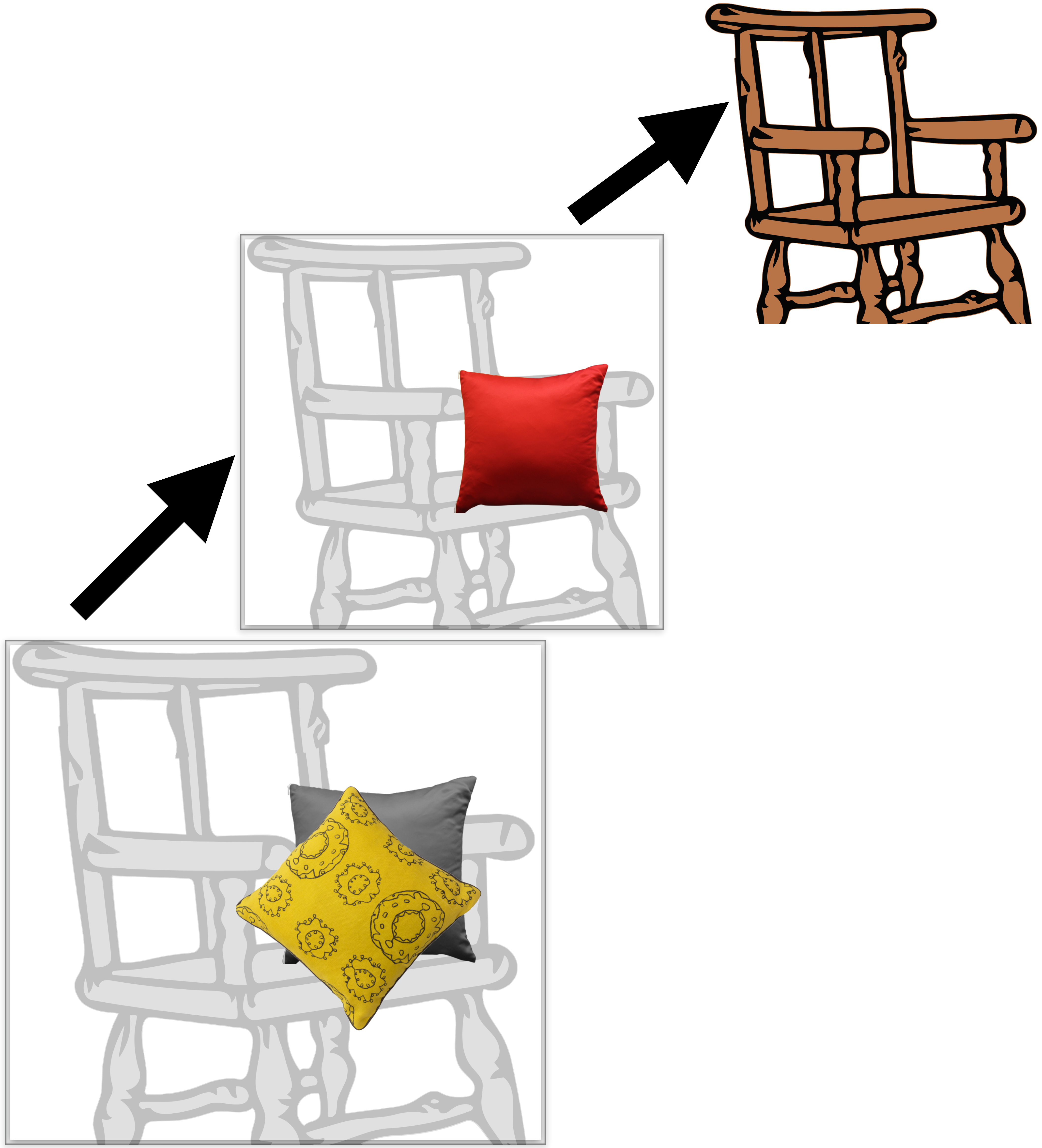
 = Object.create();

.redPillow = ;

= Object.create( );

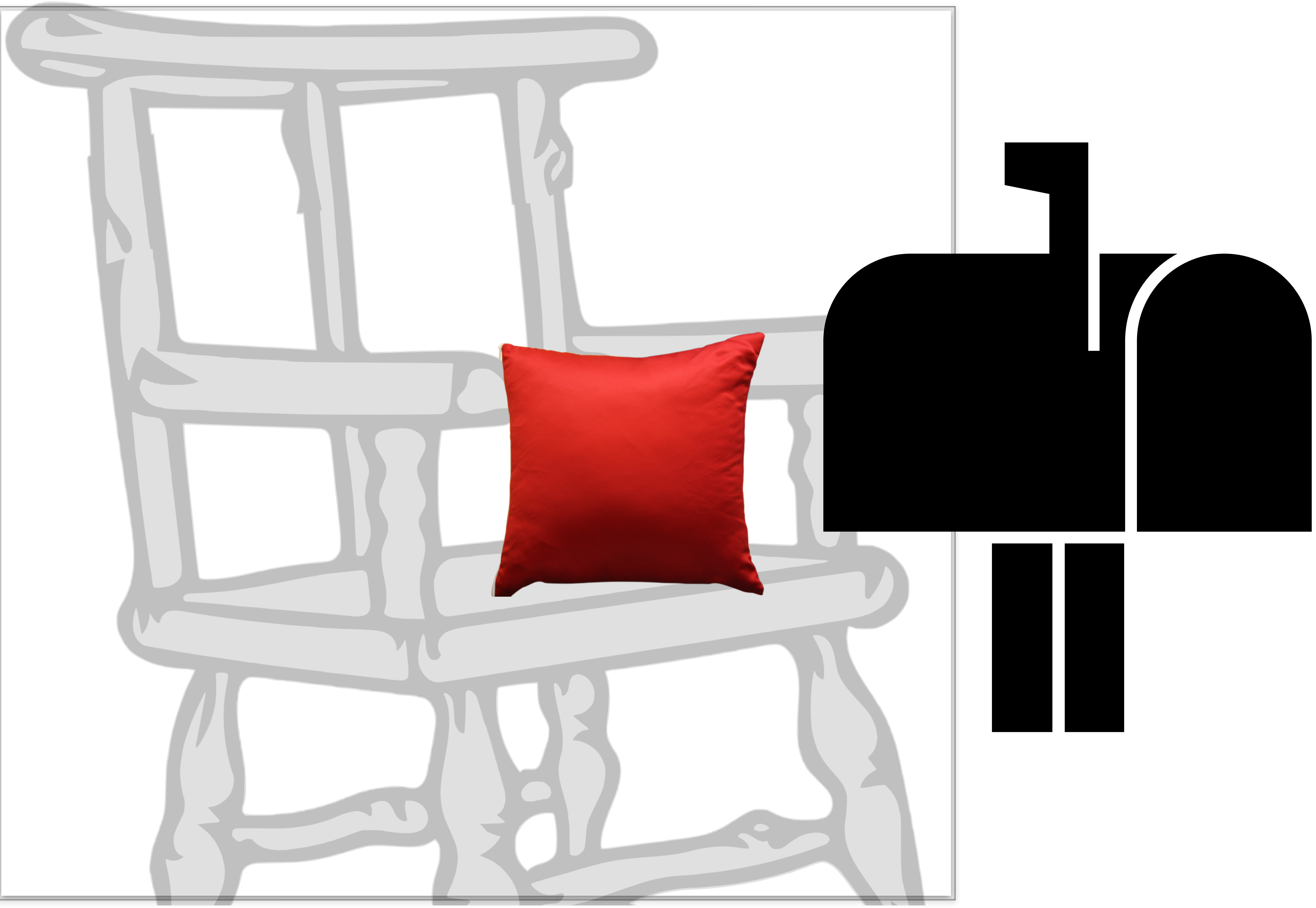
.yellowPillow = ;

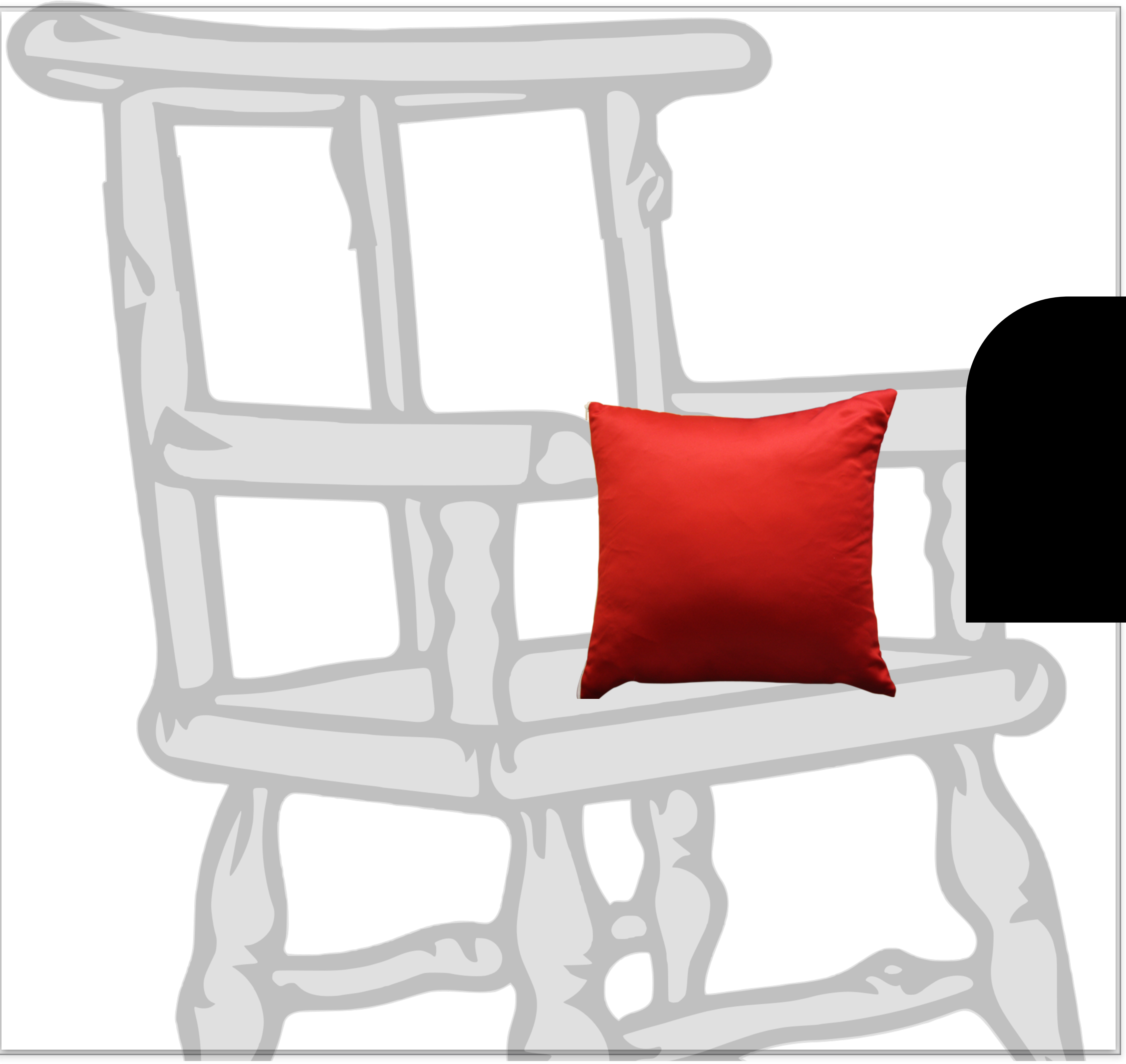
Result



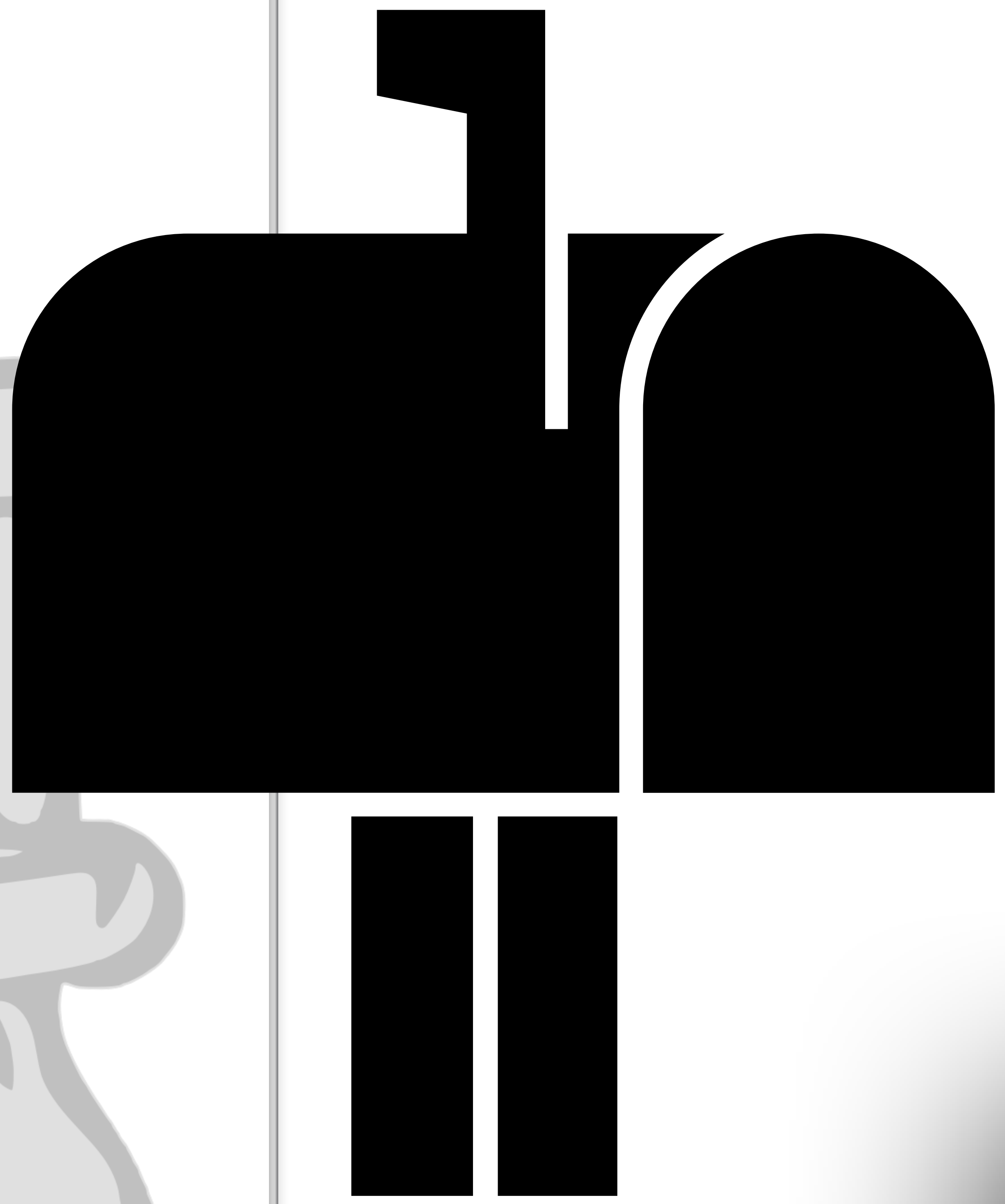
Objects *pass*

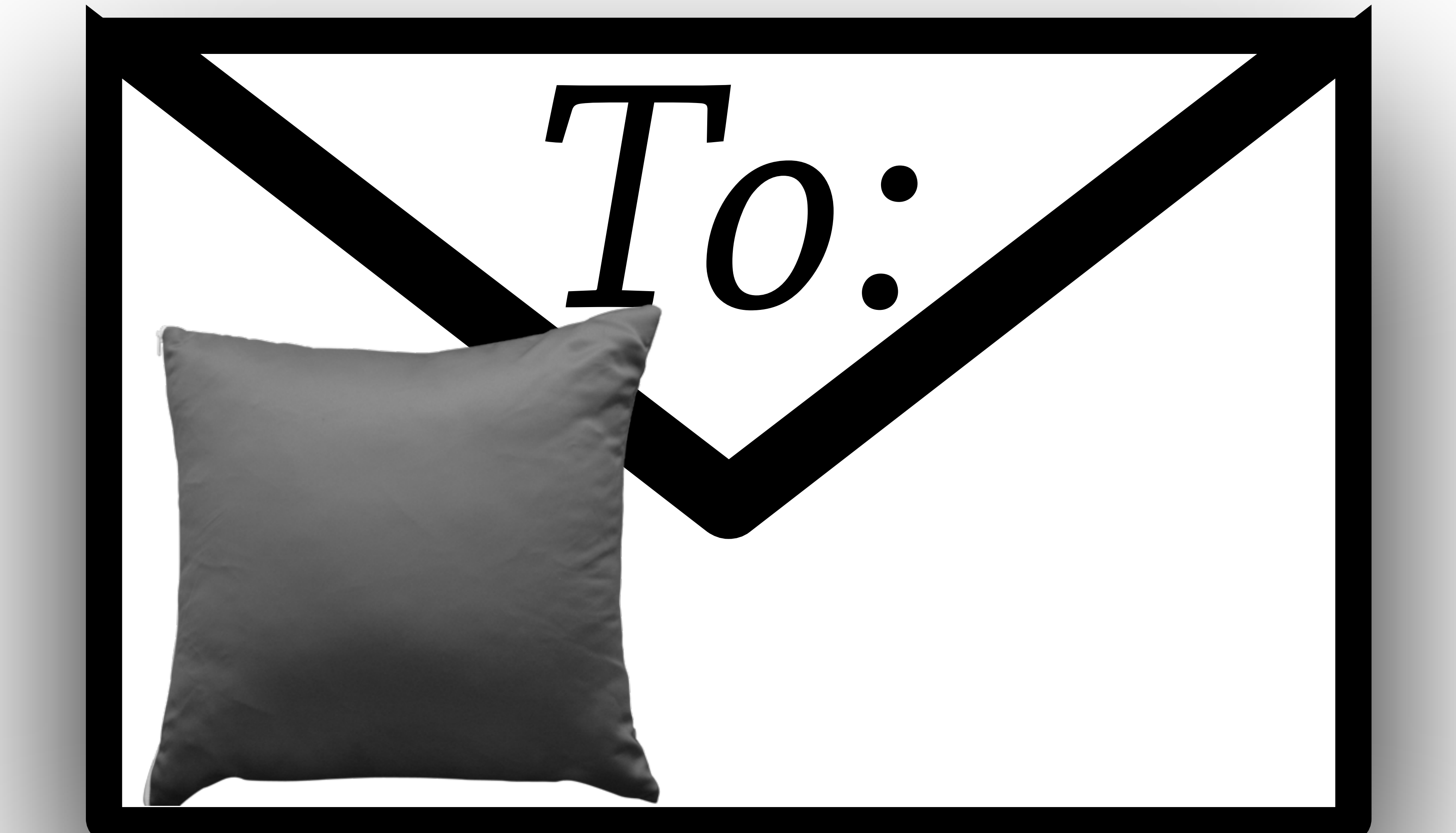
messages

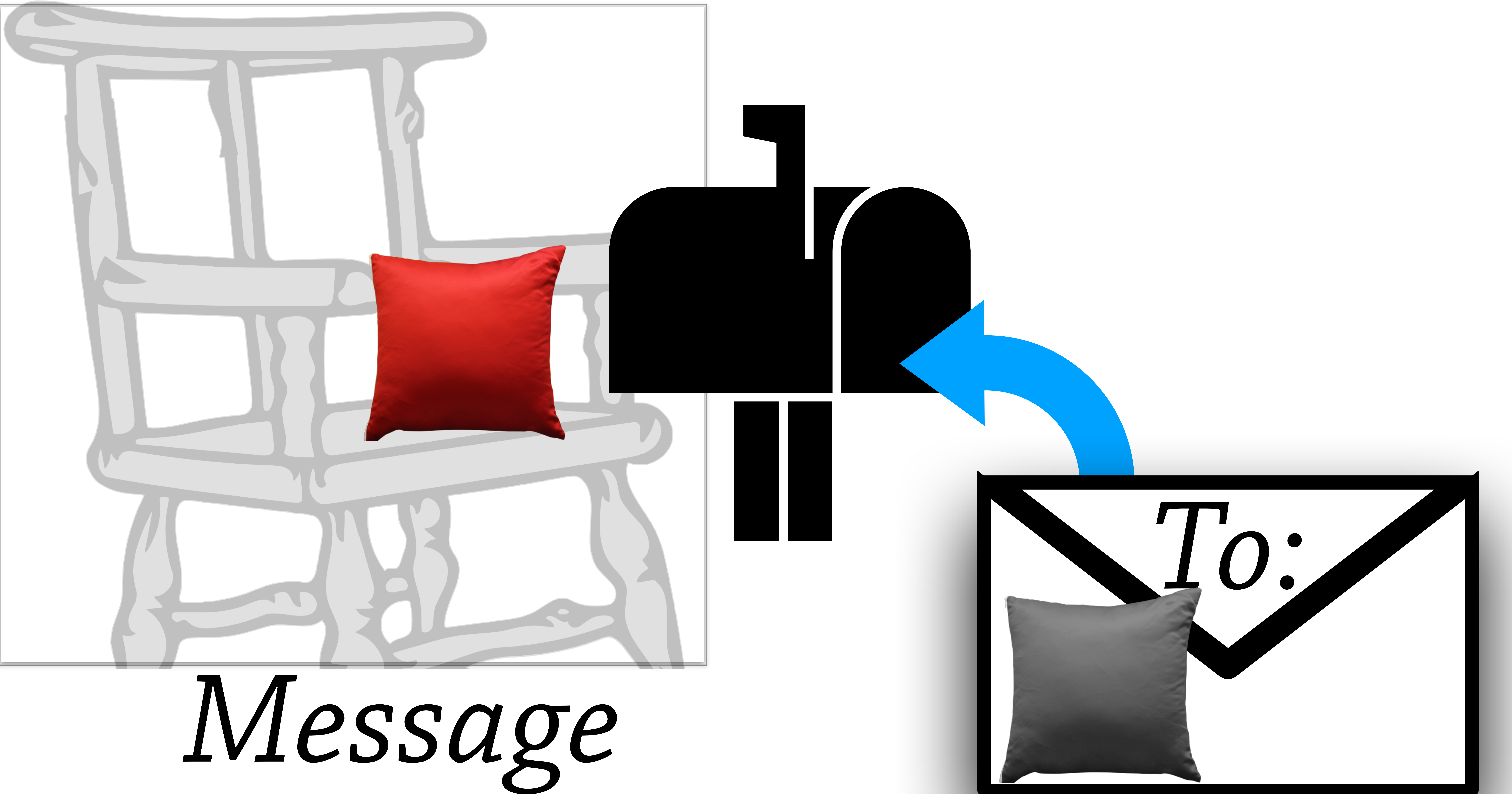




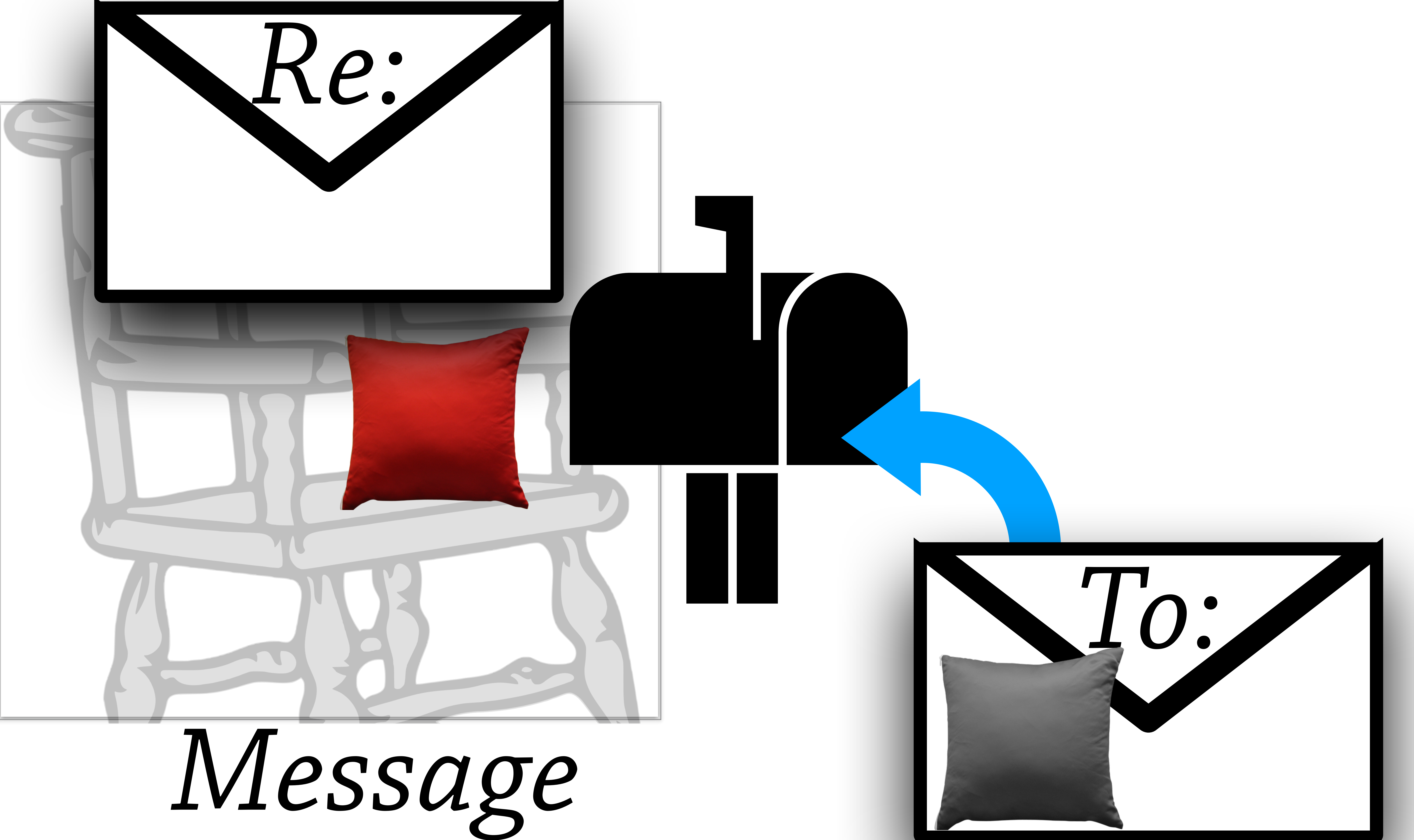
Message







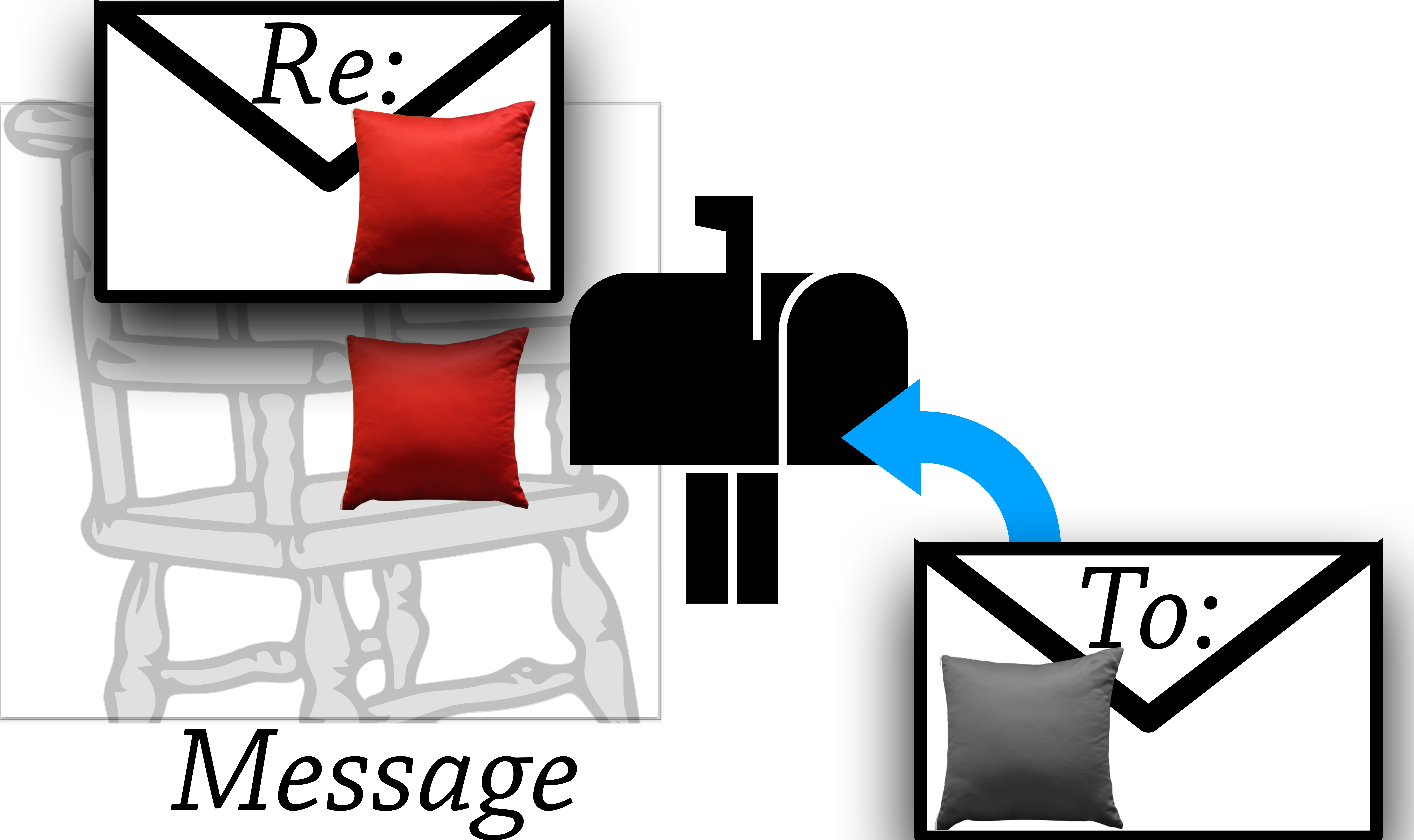
Message



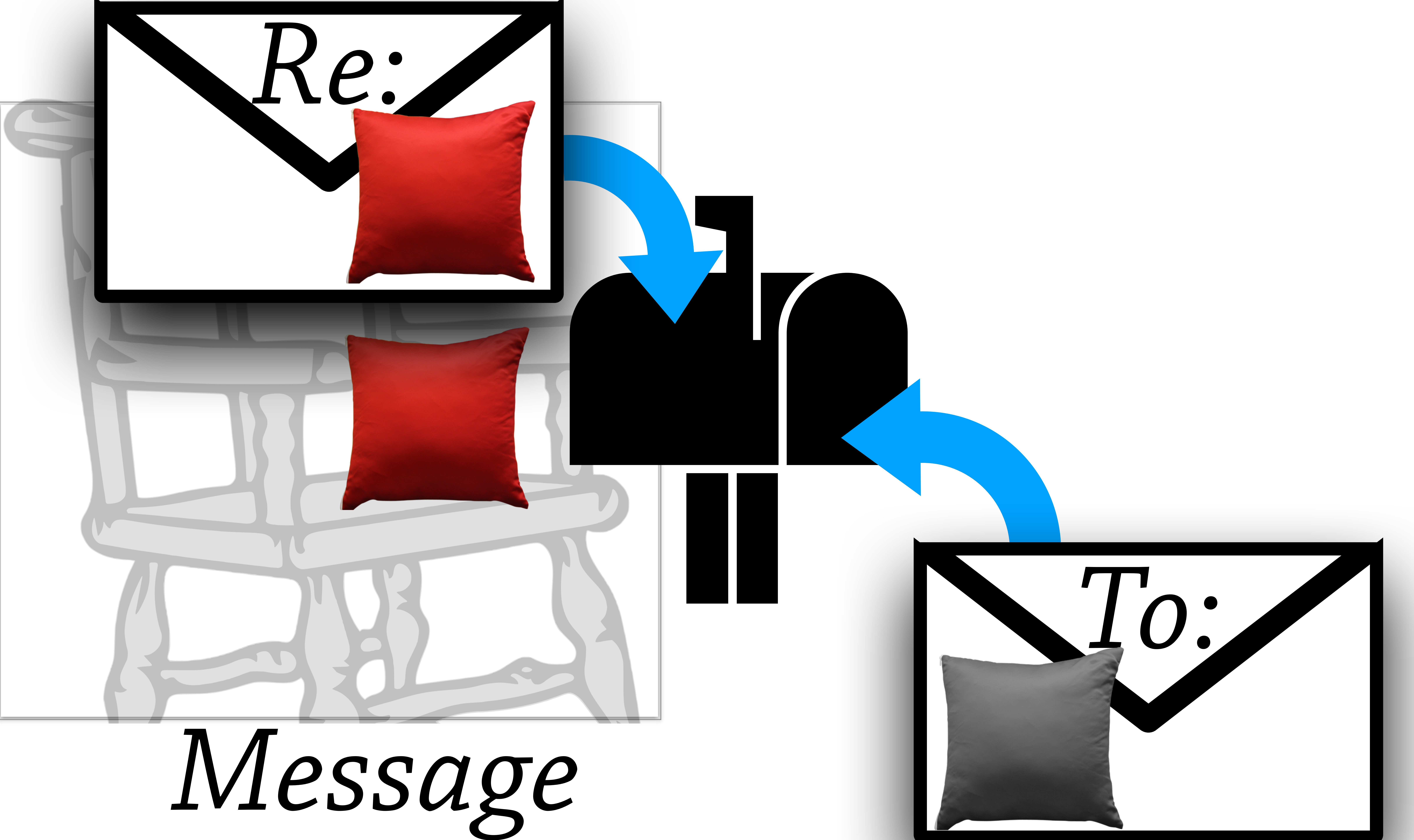
Re:

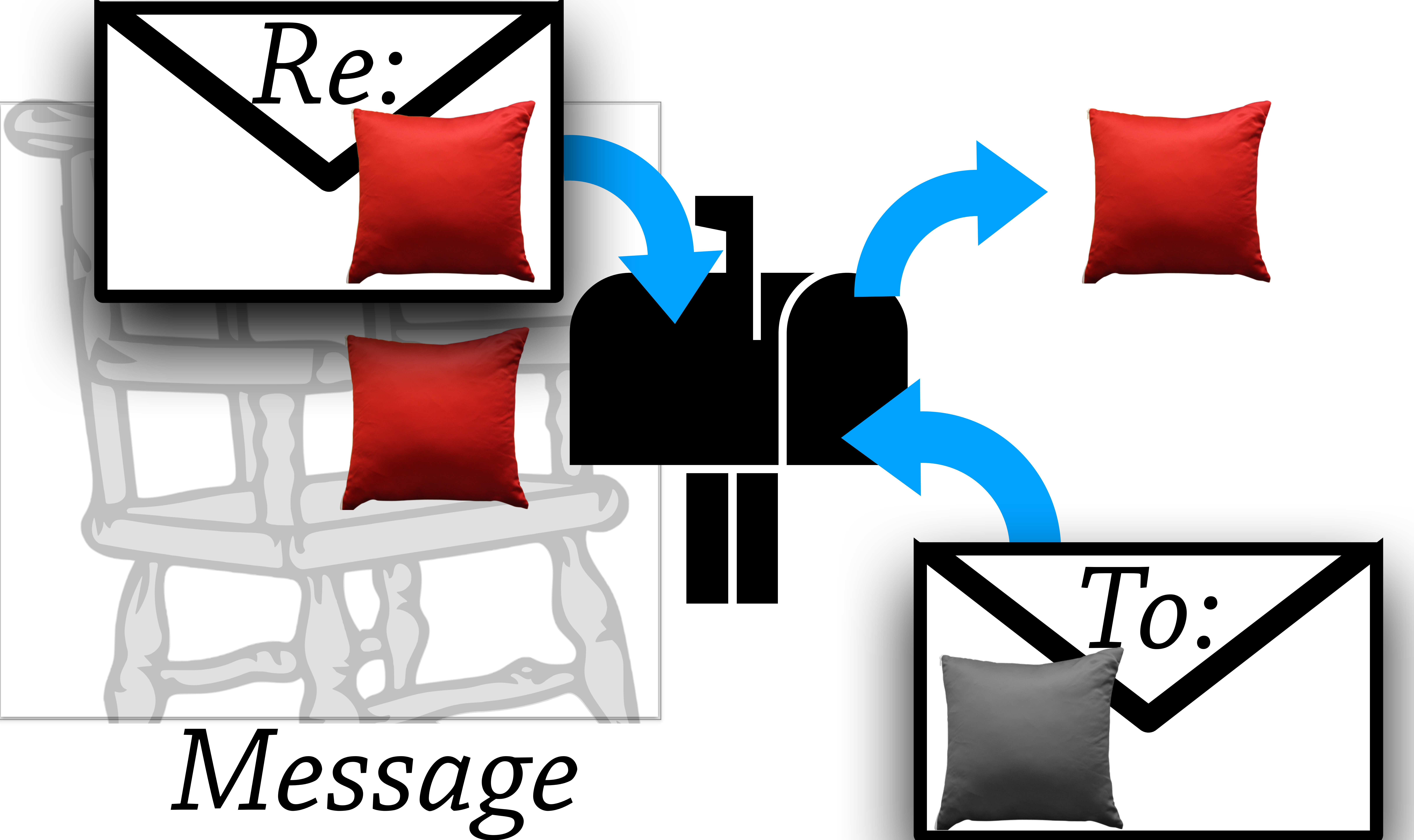
To:

Message



Message

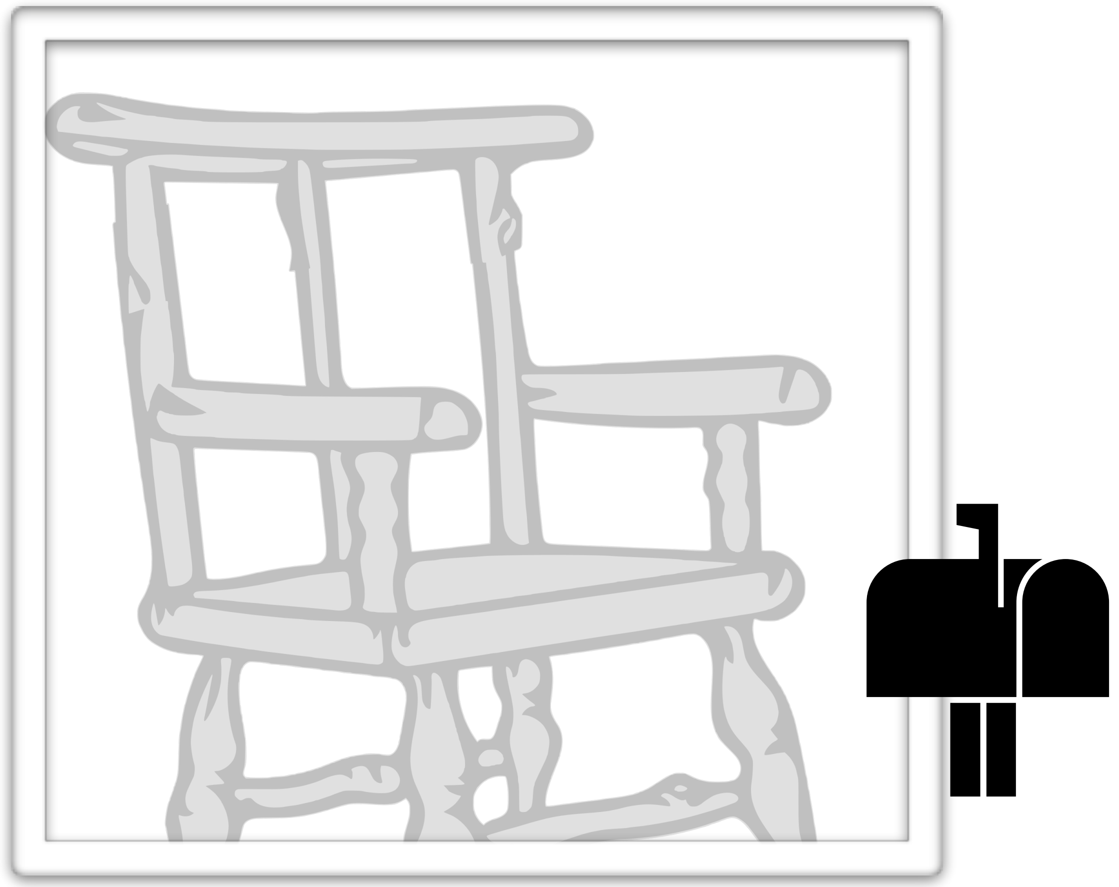

Re:
To:
Message



Message

Commands

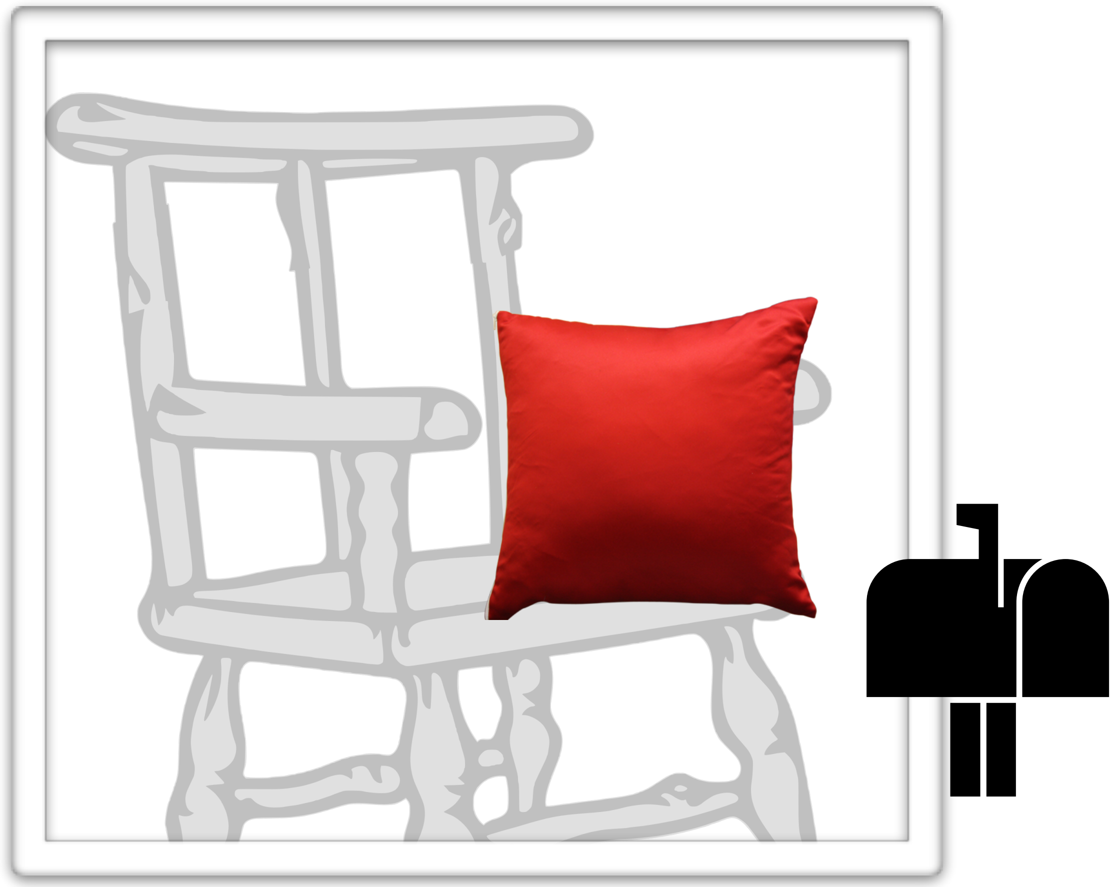
Result



Commands

`.redPillow =` `;`

Result

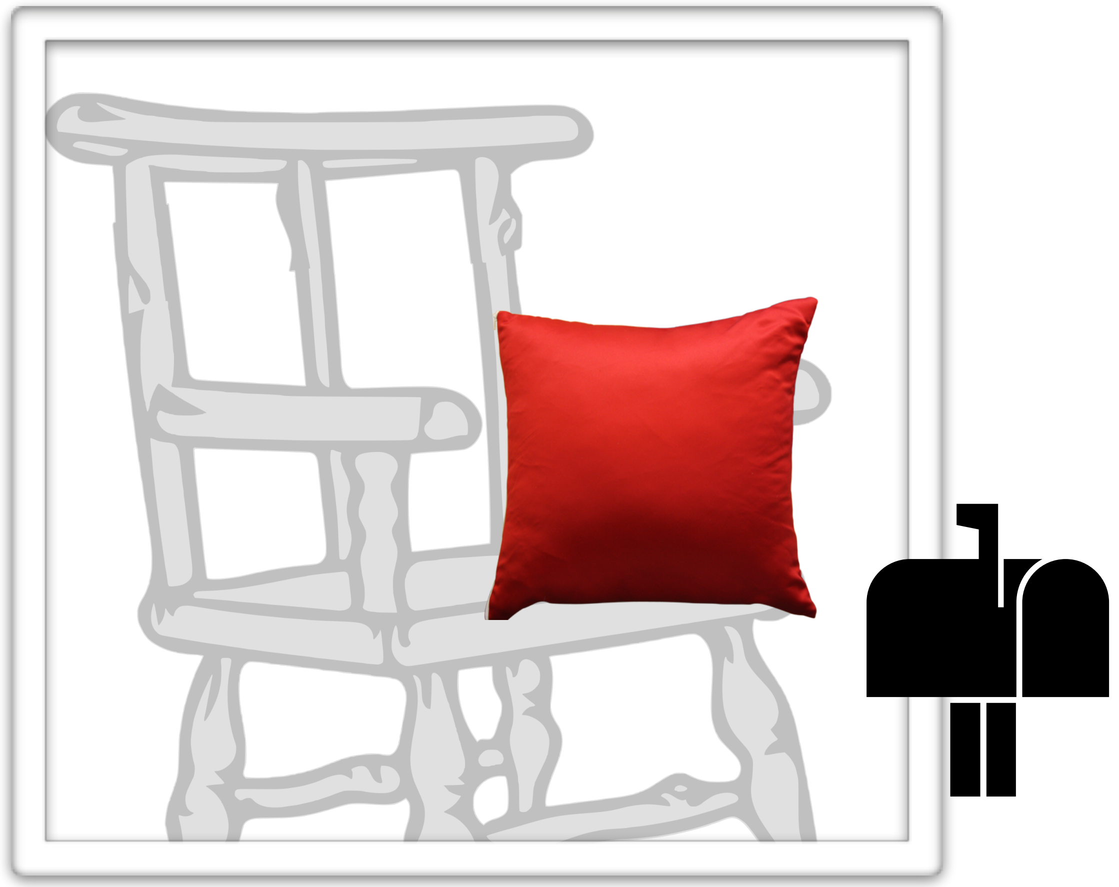


Commands

.redPillow = ;

.redPillow

Result

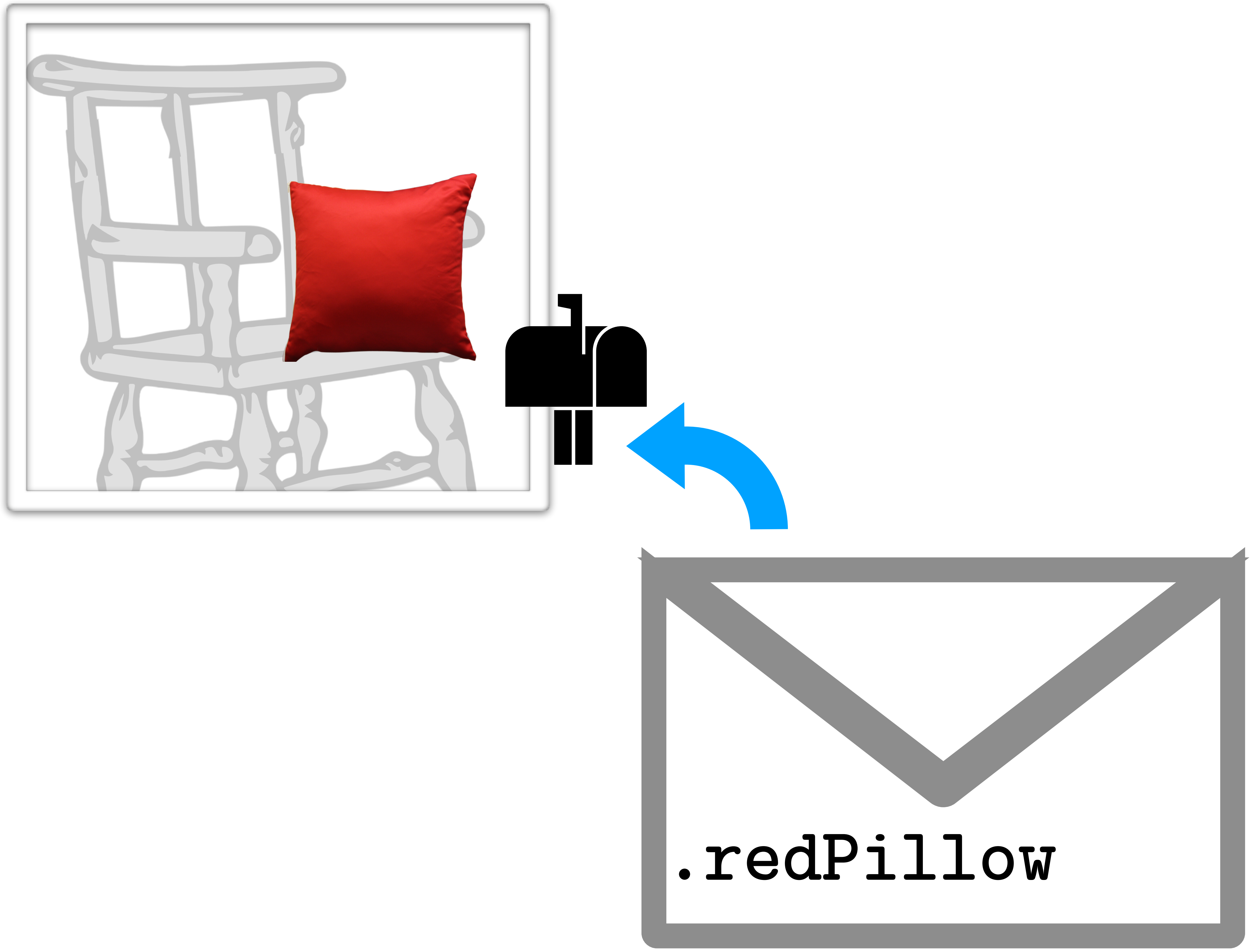


Commands

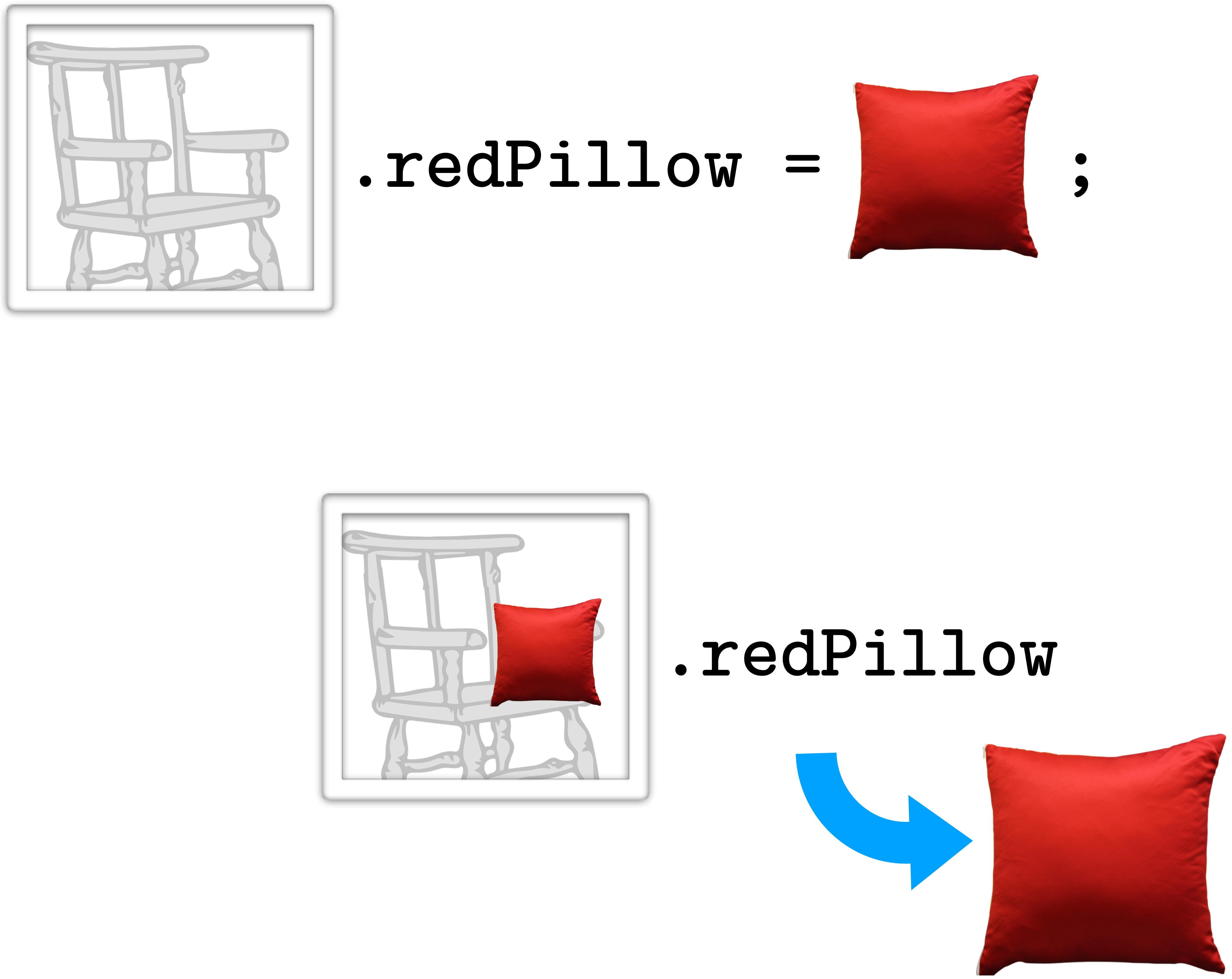
.redPillow = ;

.redPillow

Result



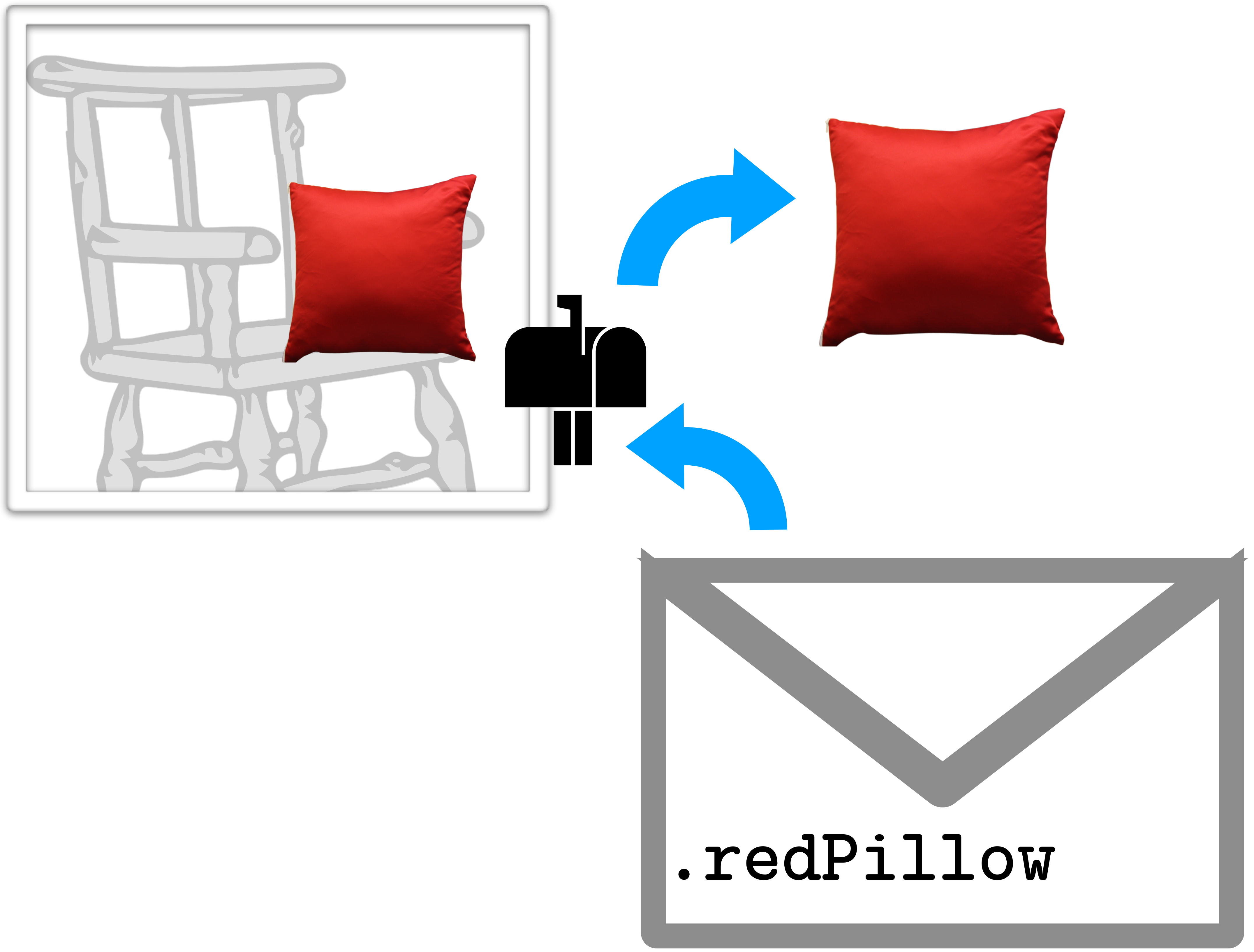
Commands



.redPillow = ;

.redPillow

Result

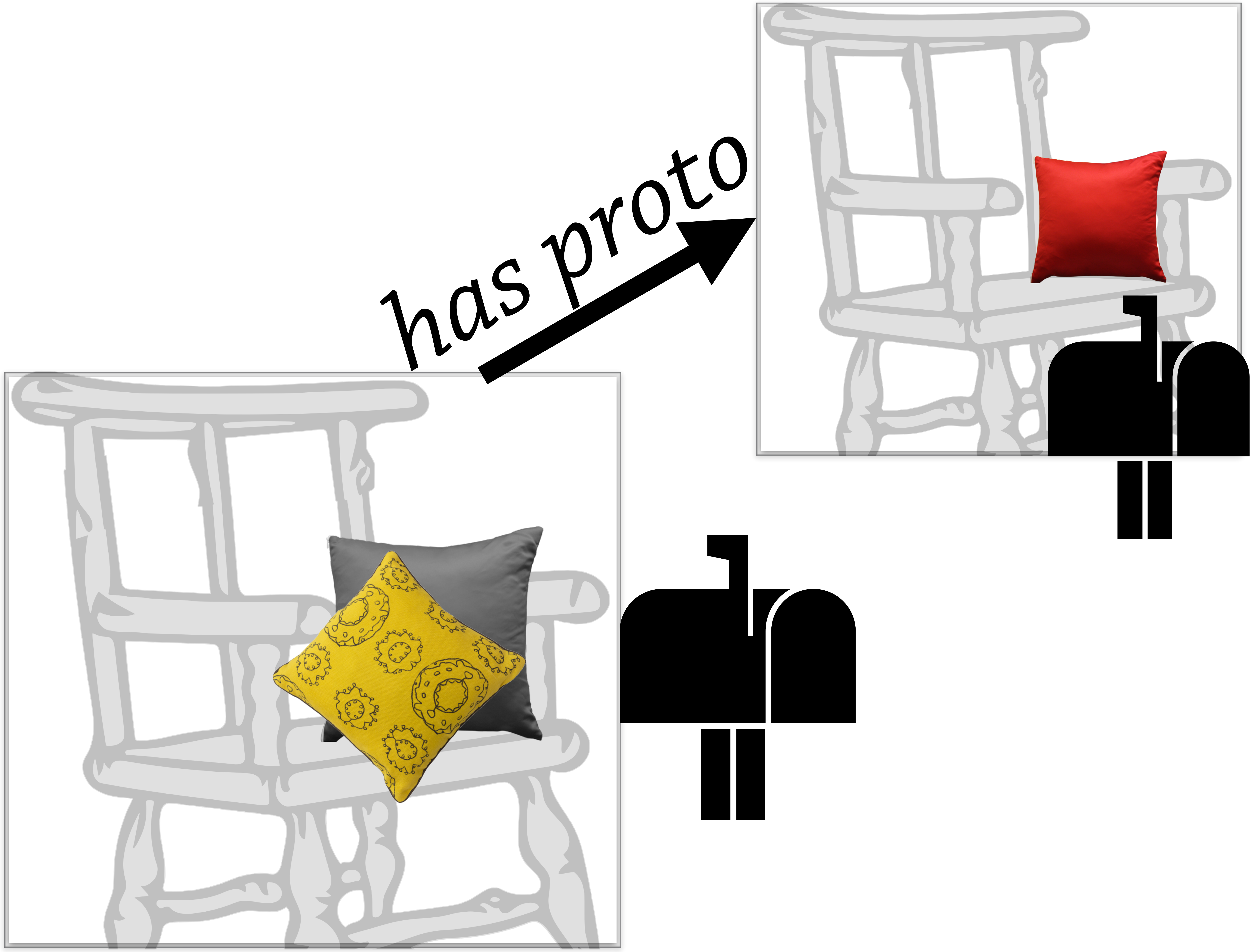


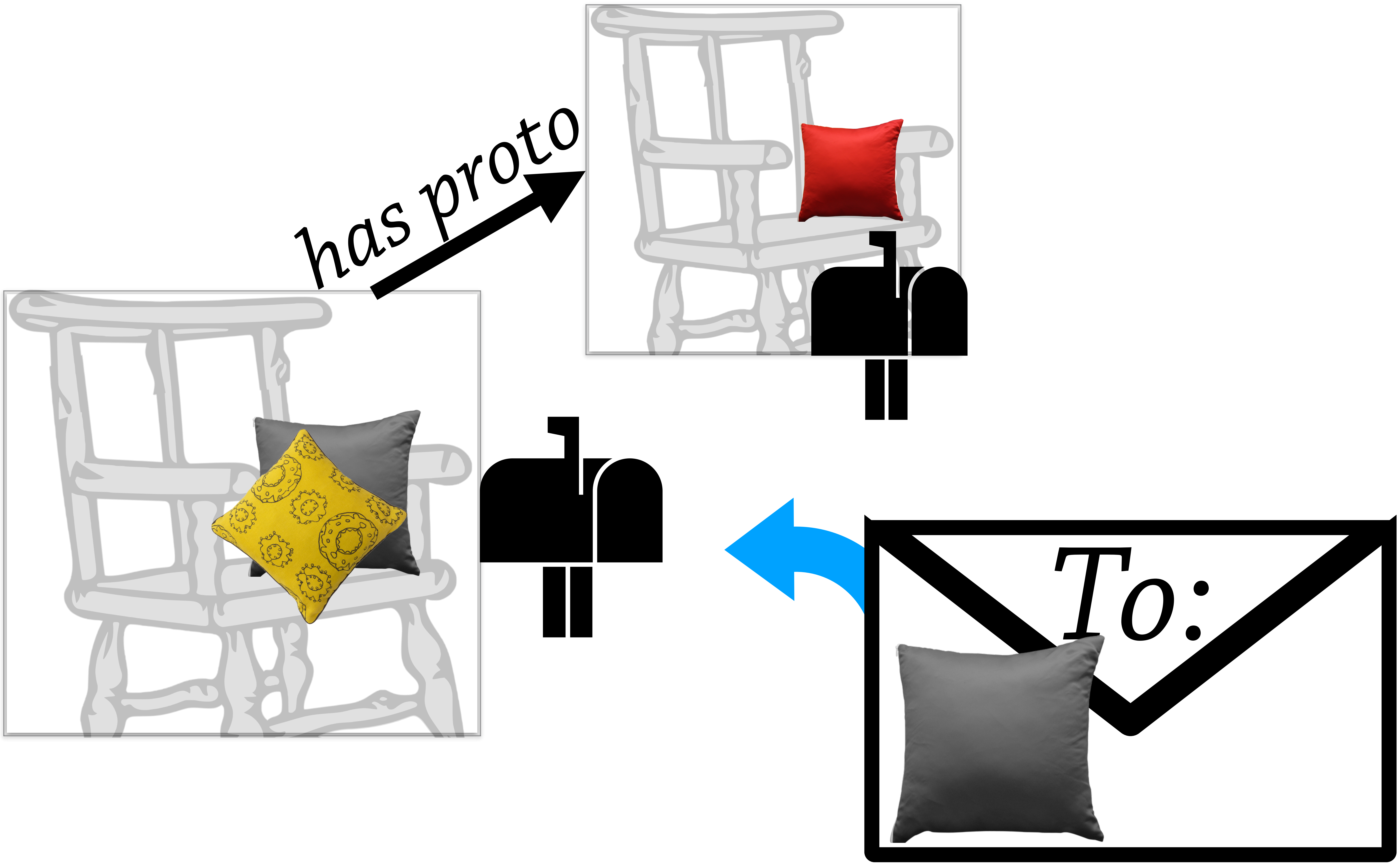
.redPillow

Objects

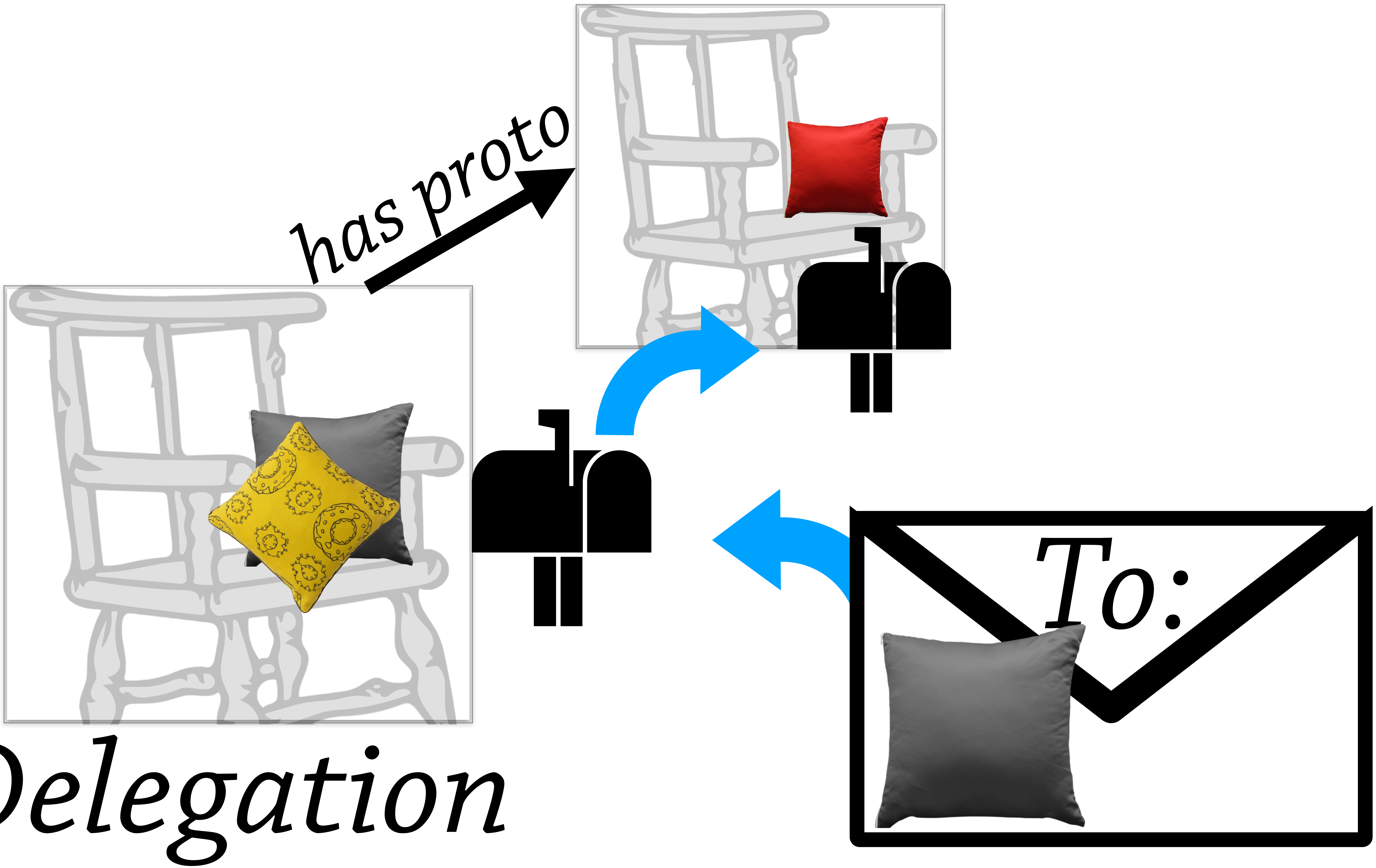
delegate to

prototypes


has proto

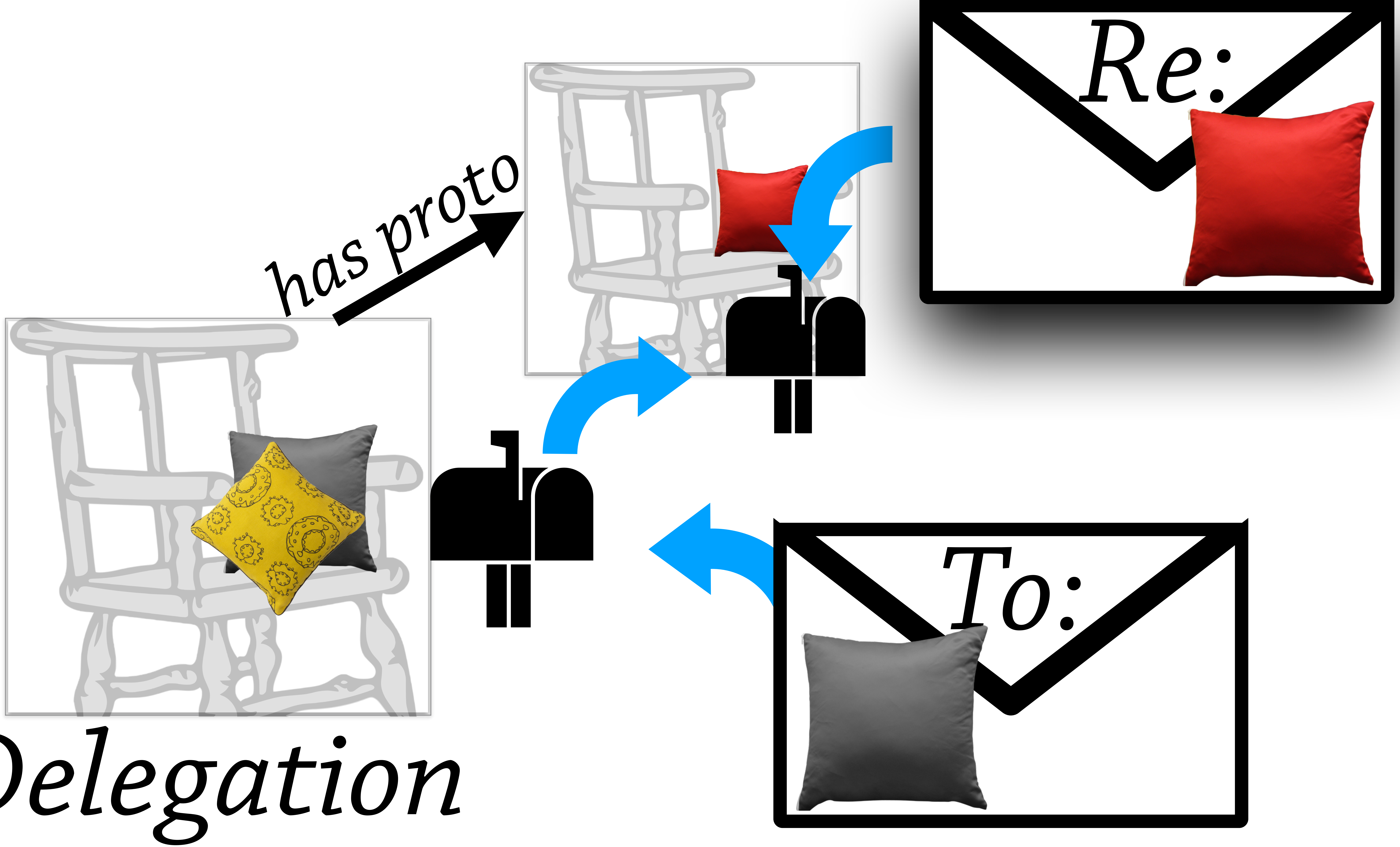

has proto
To:

Delegation



has proto

To:

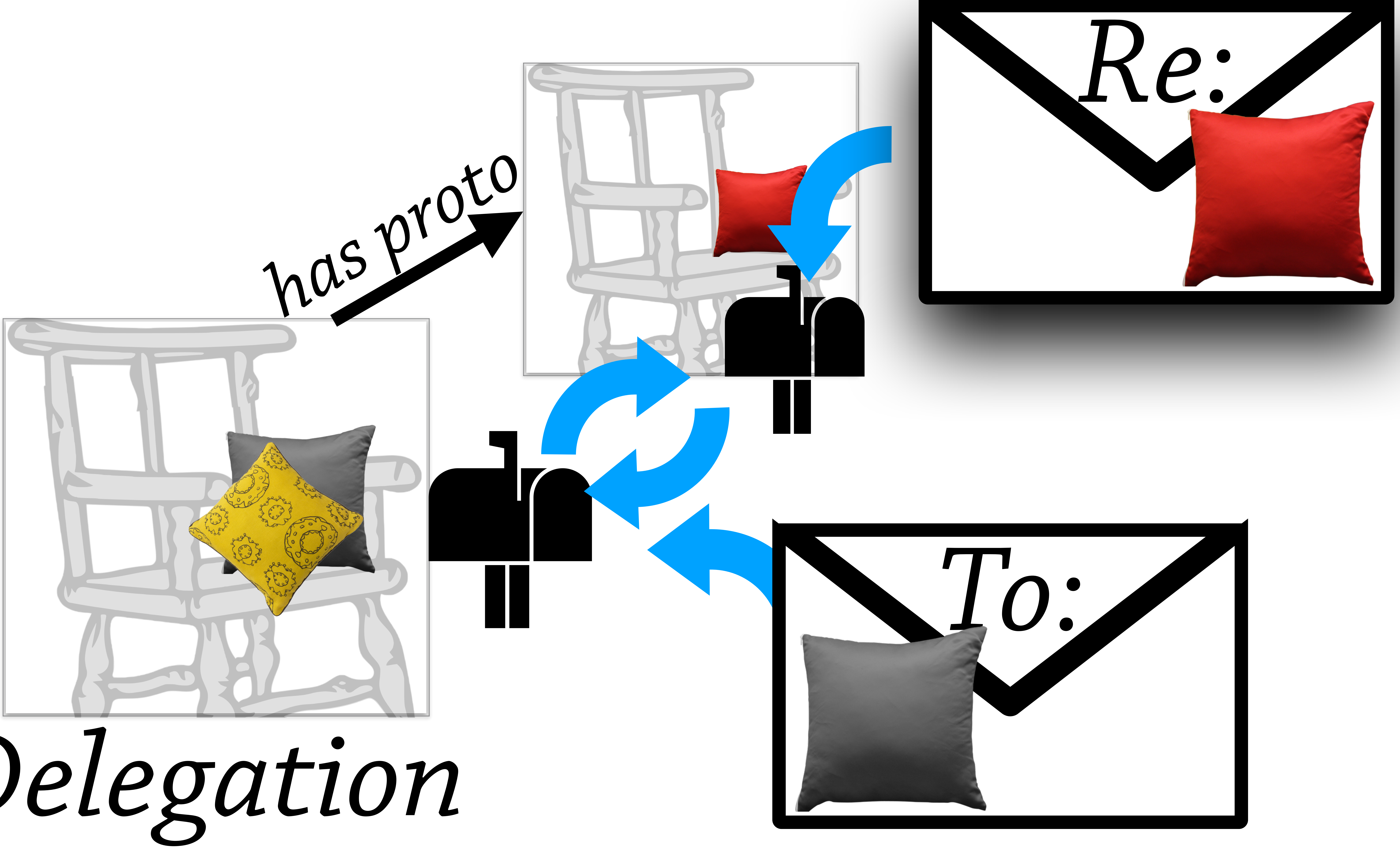


has proto

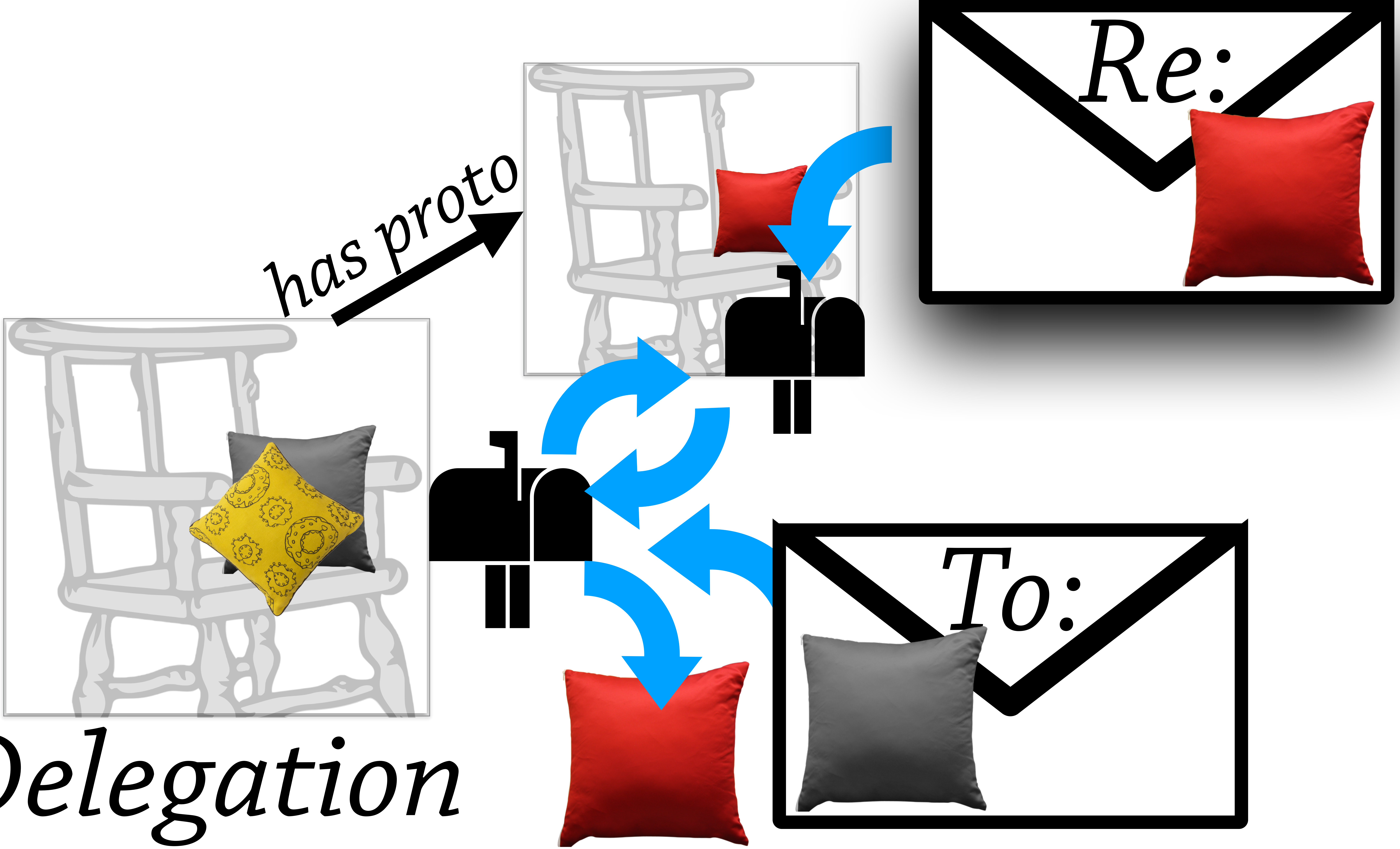
Re:

To:

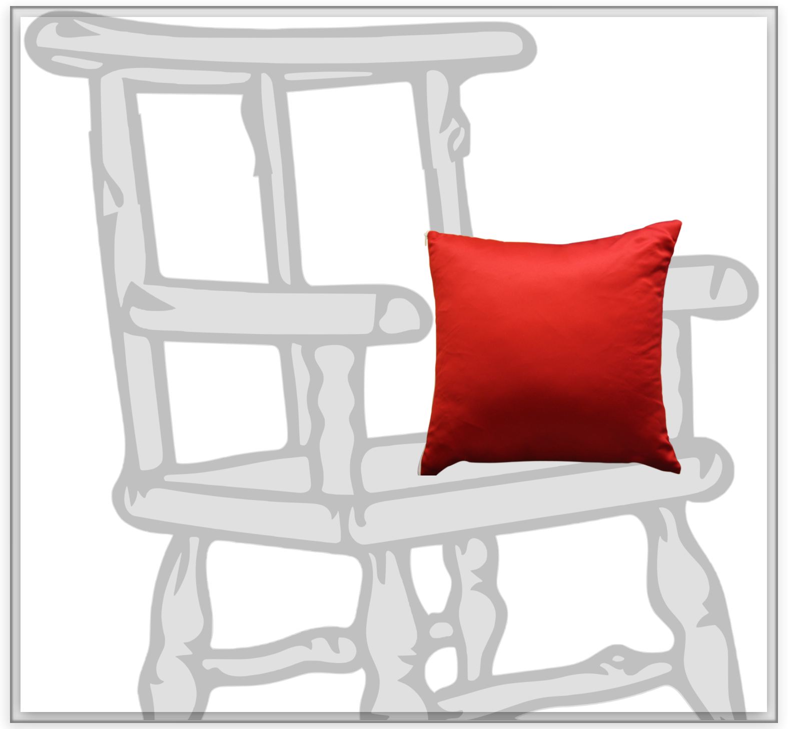
Delegation



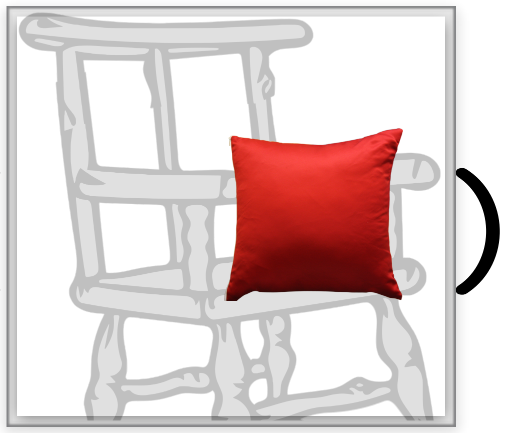
Delegation



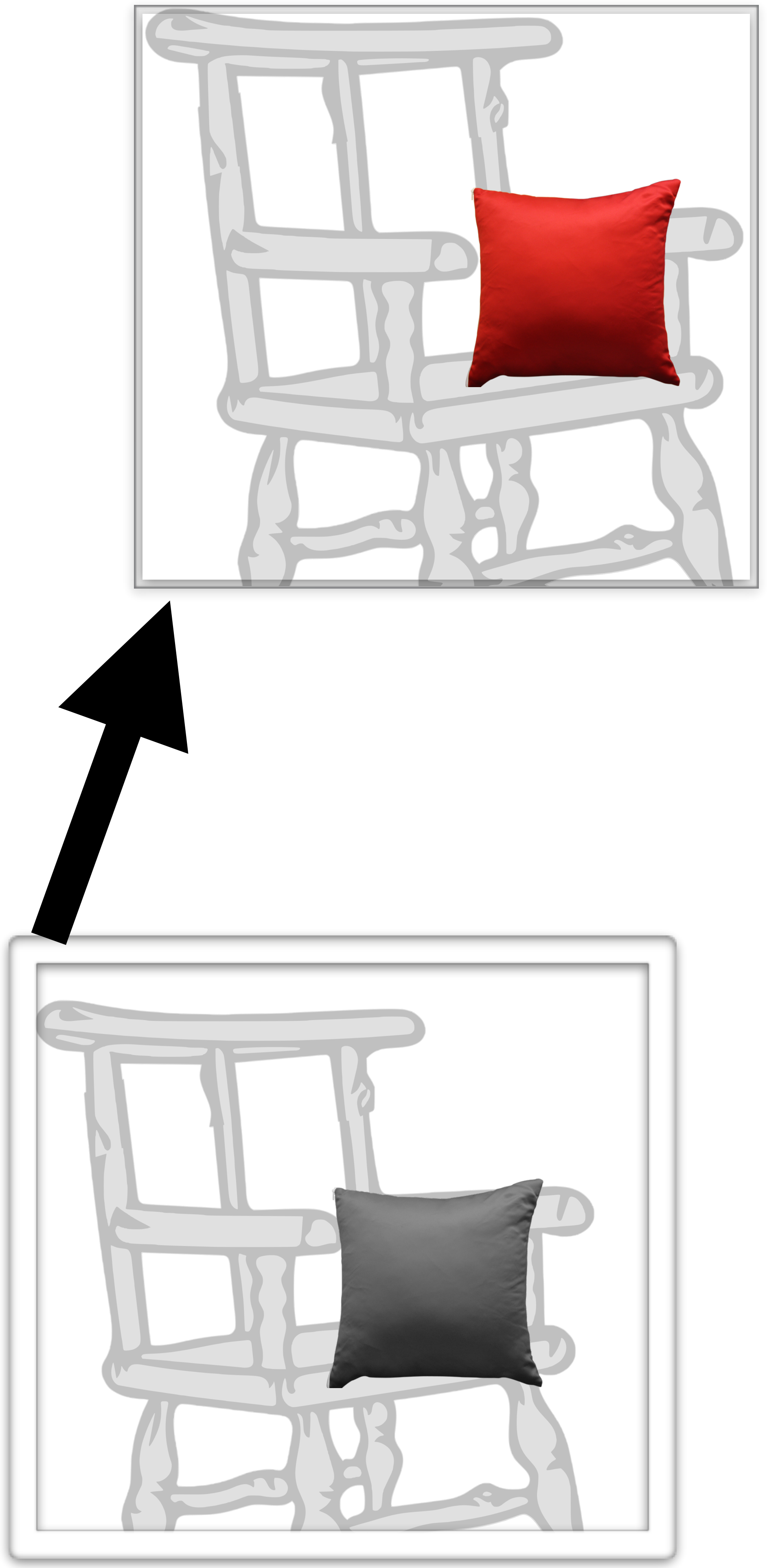
Delegation

Commands	Result
	

Commands

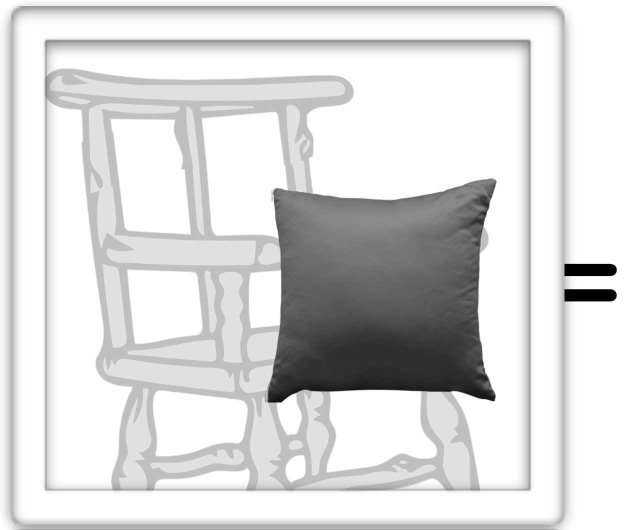
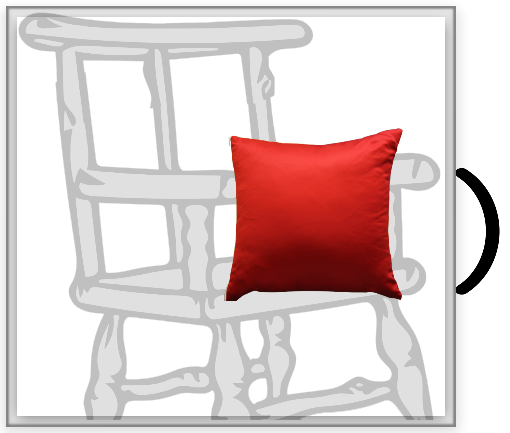
= Object.create();

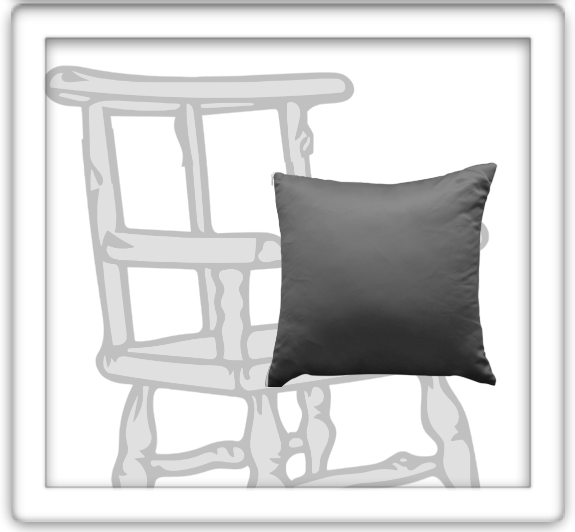
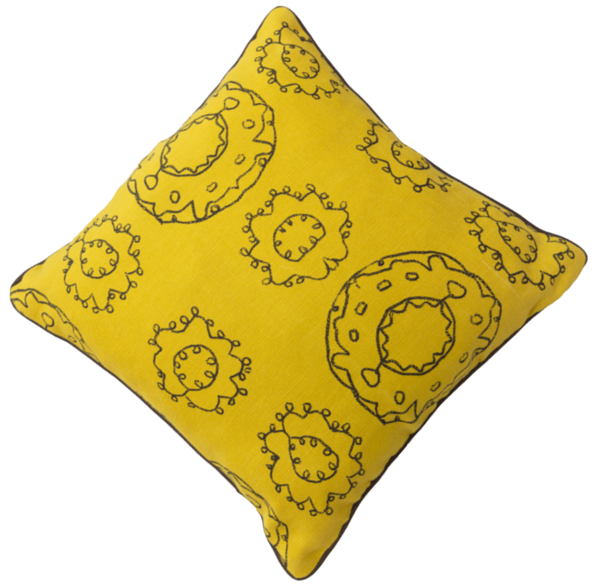
Result

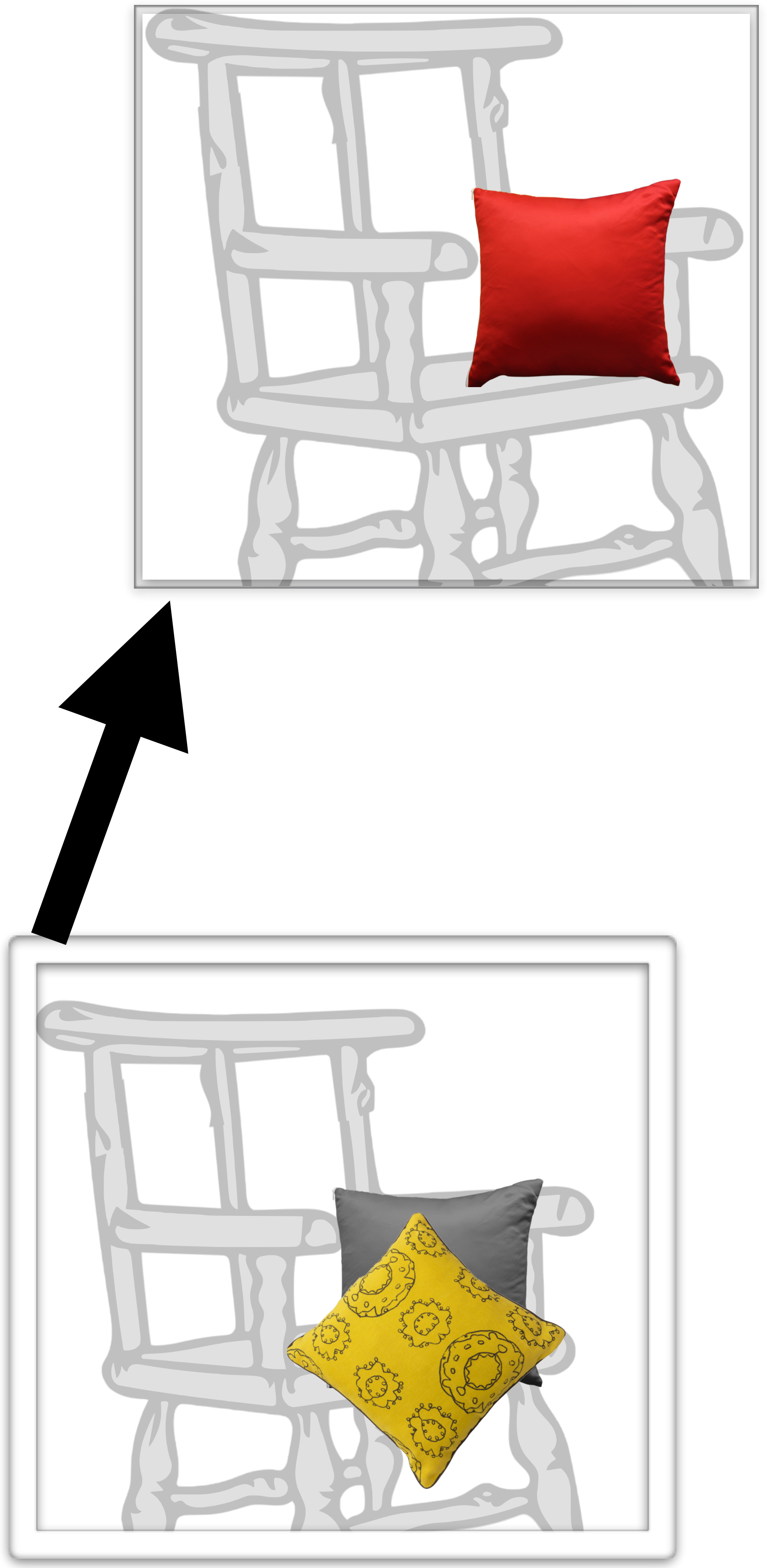


Commands

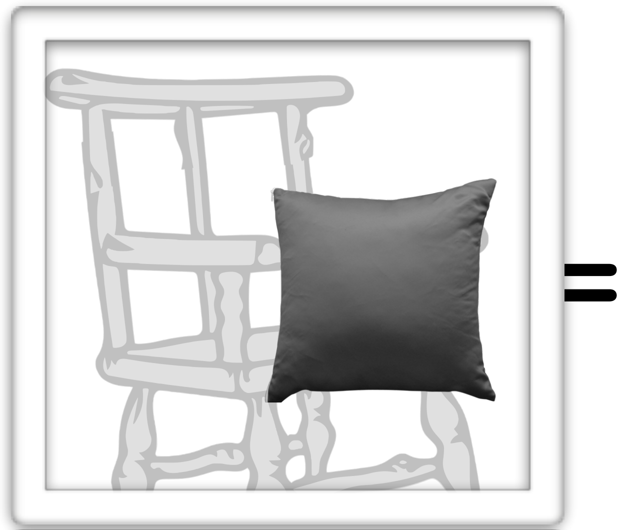
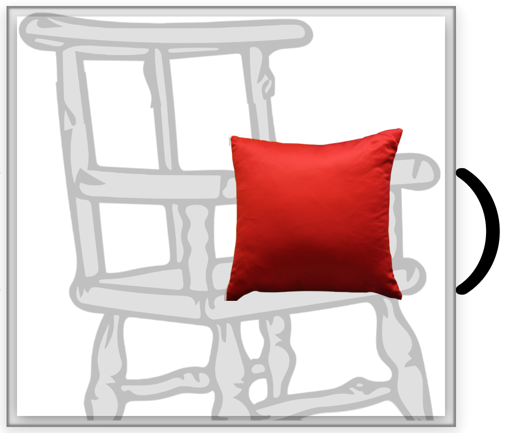
Result

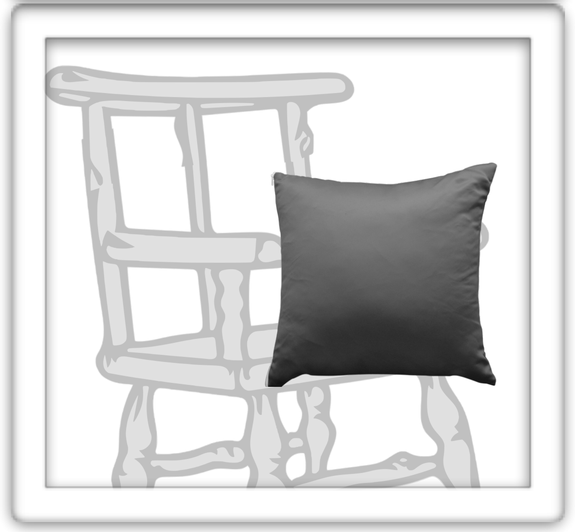
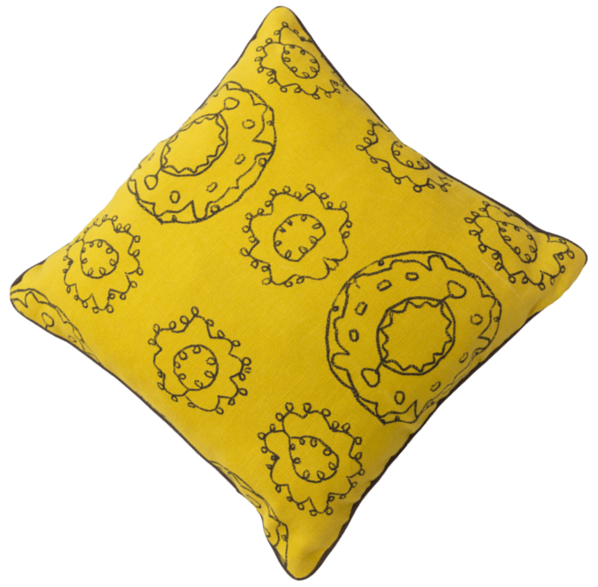
 = Object.create();

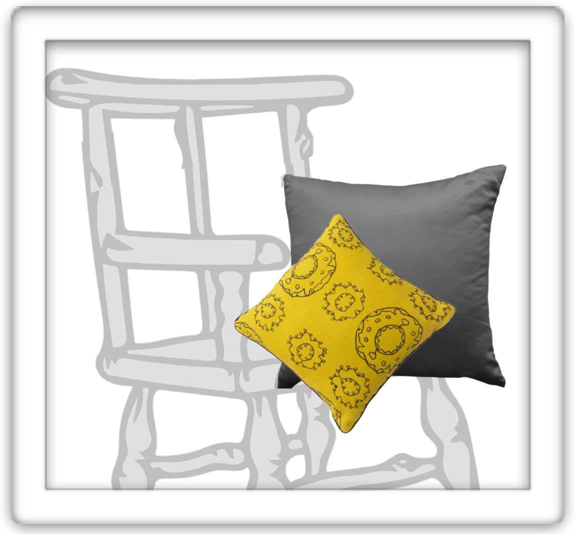
.yellowPillow = ;



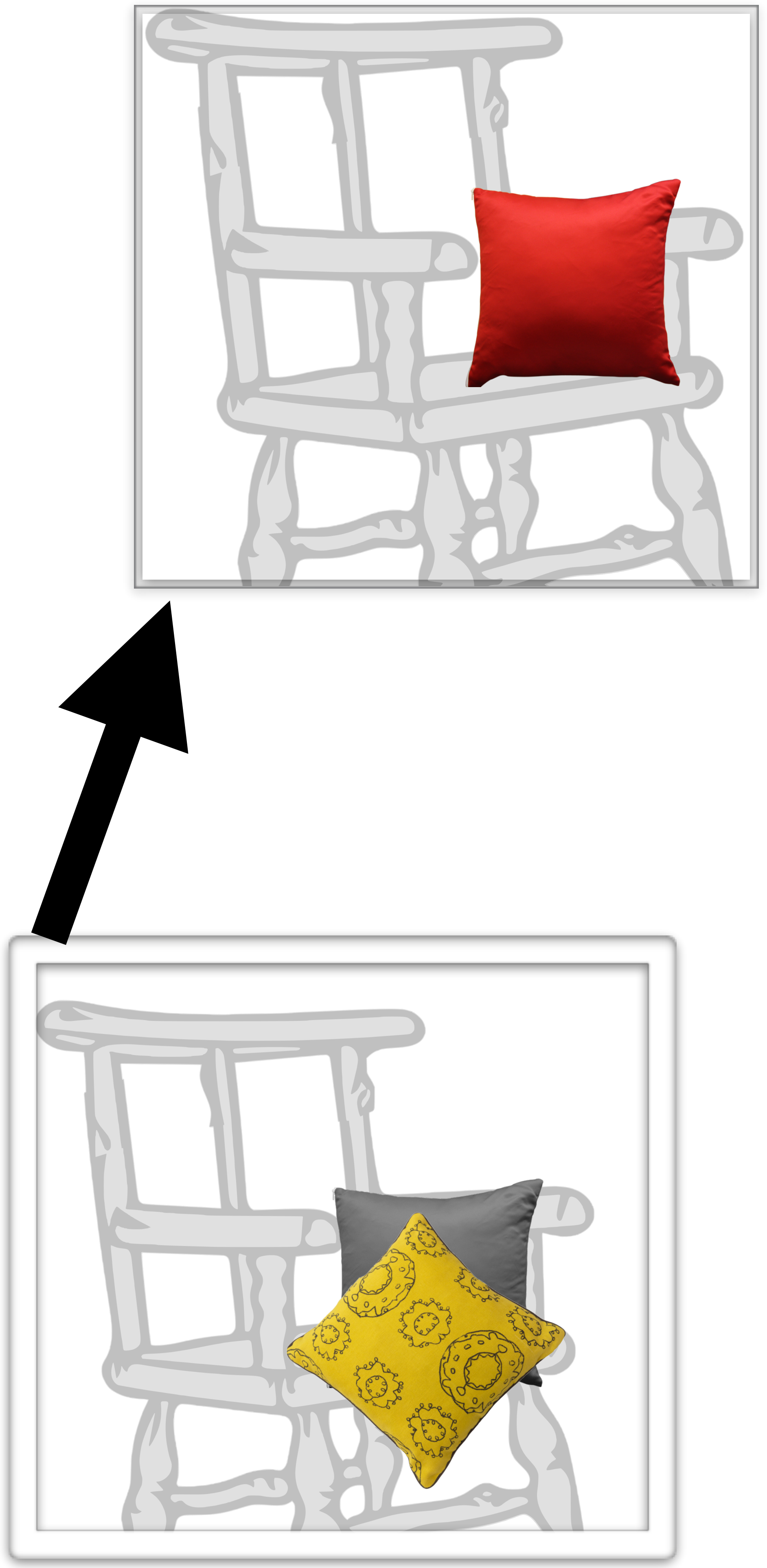
Commands

 = Object.create();

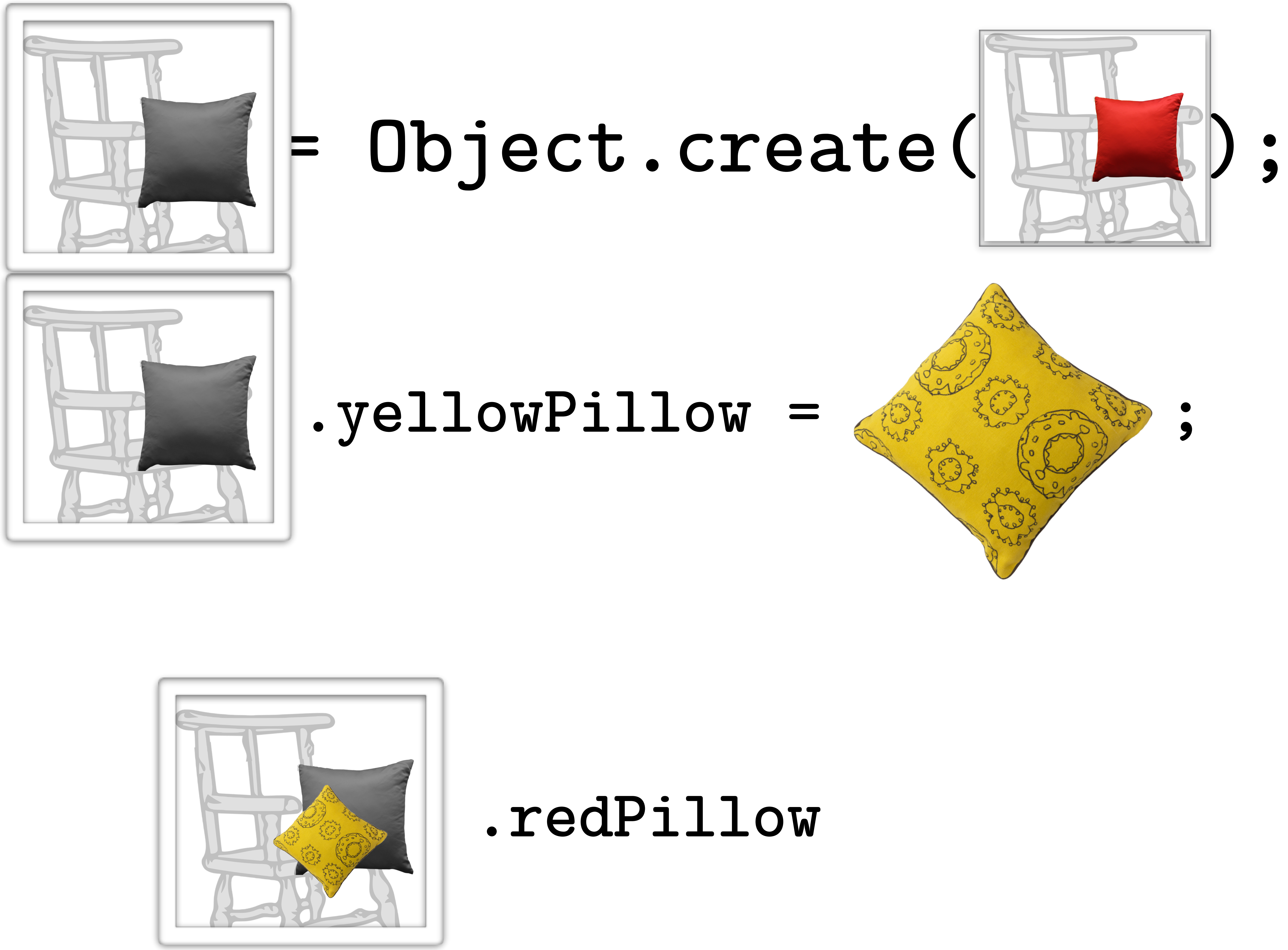
 .yellowPillow =  ;

 .redPillow

Result

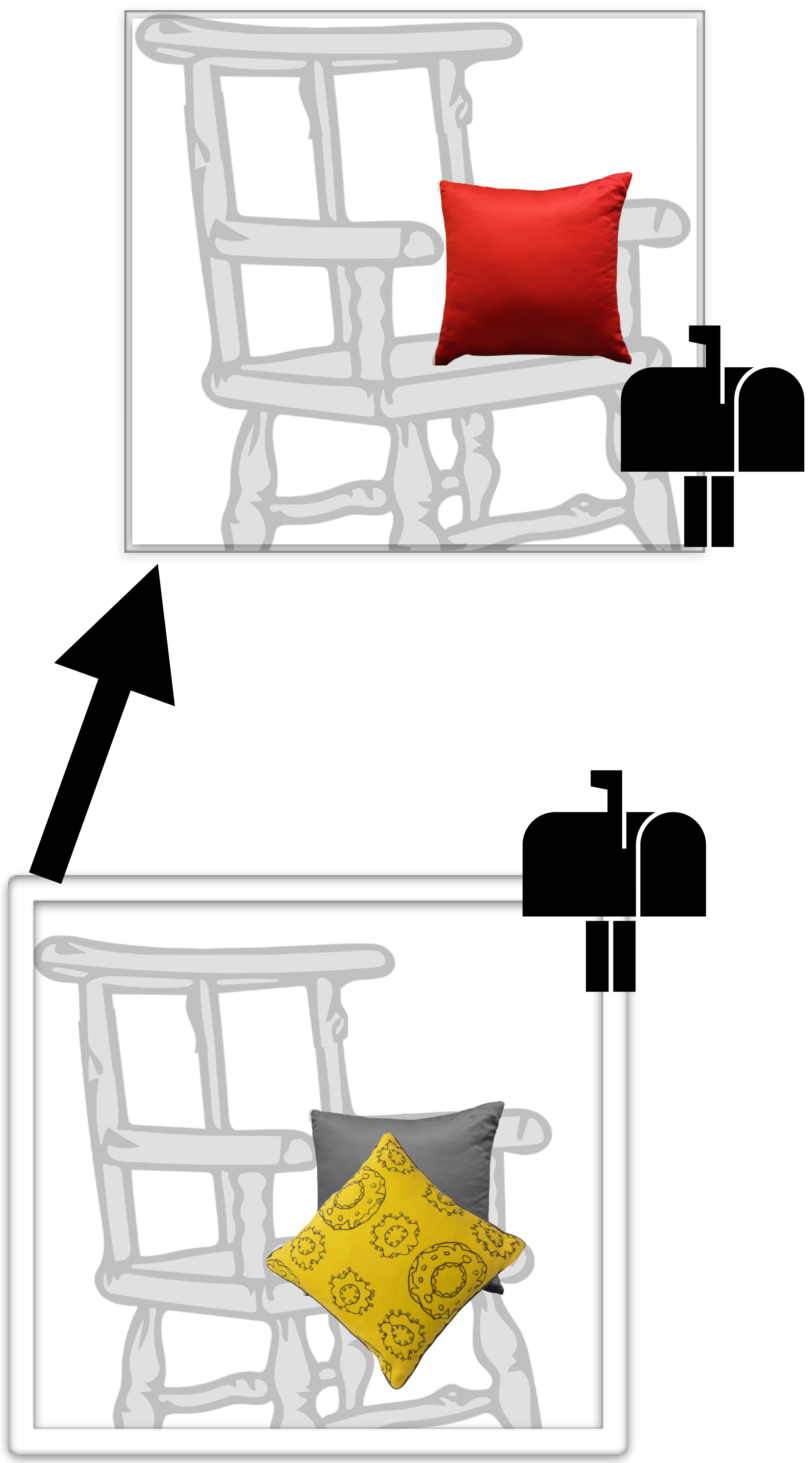


Commands

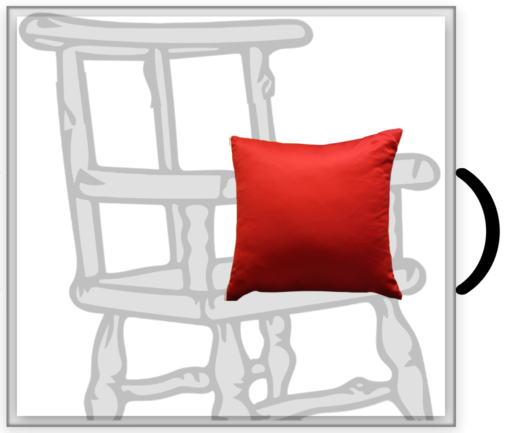


```
= Object.create(  );
.yellowPillow =  ;
.redPillow
```

Result



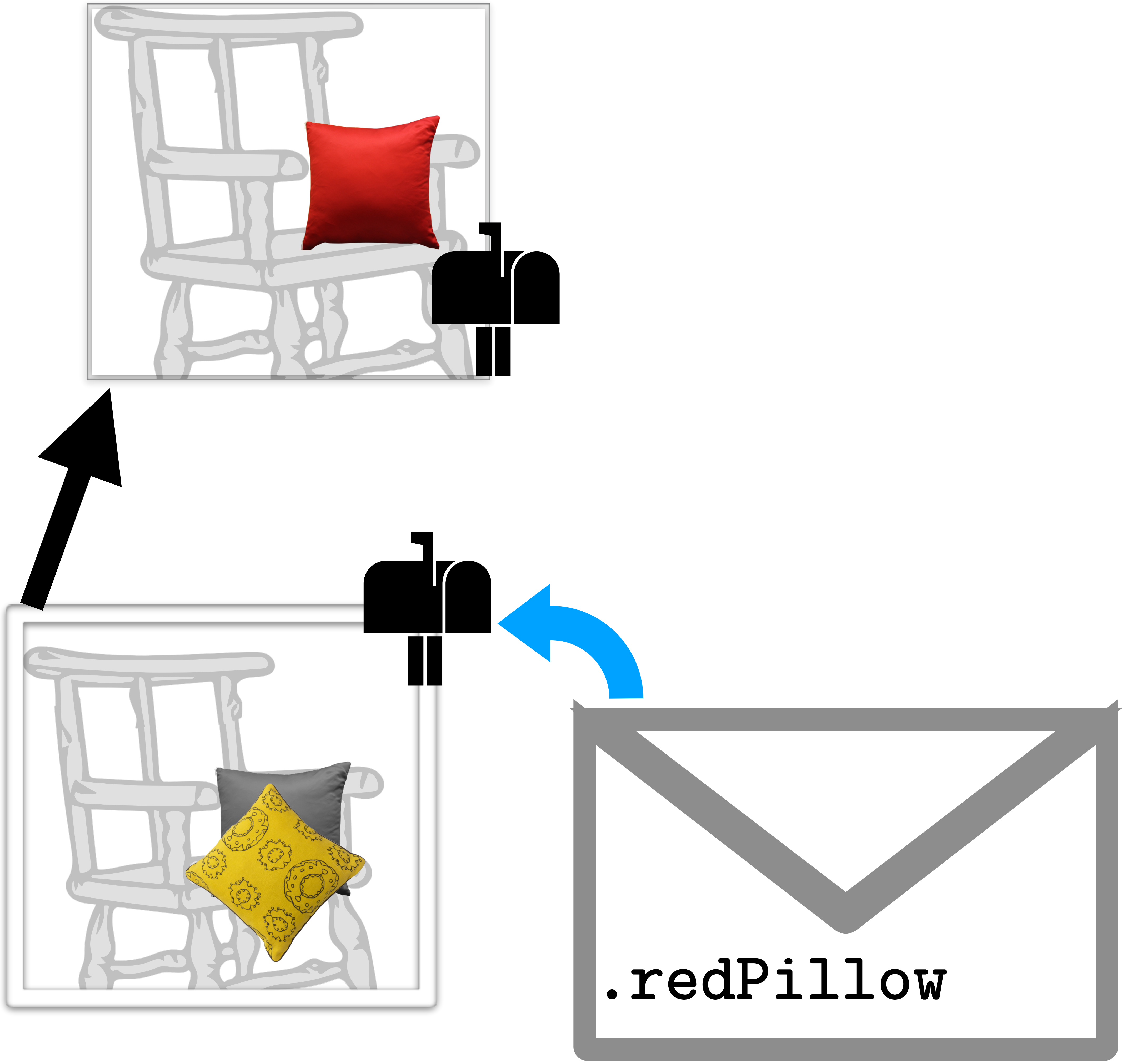
Commands

= Object.create();

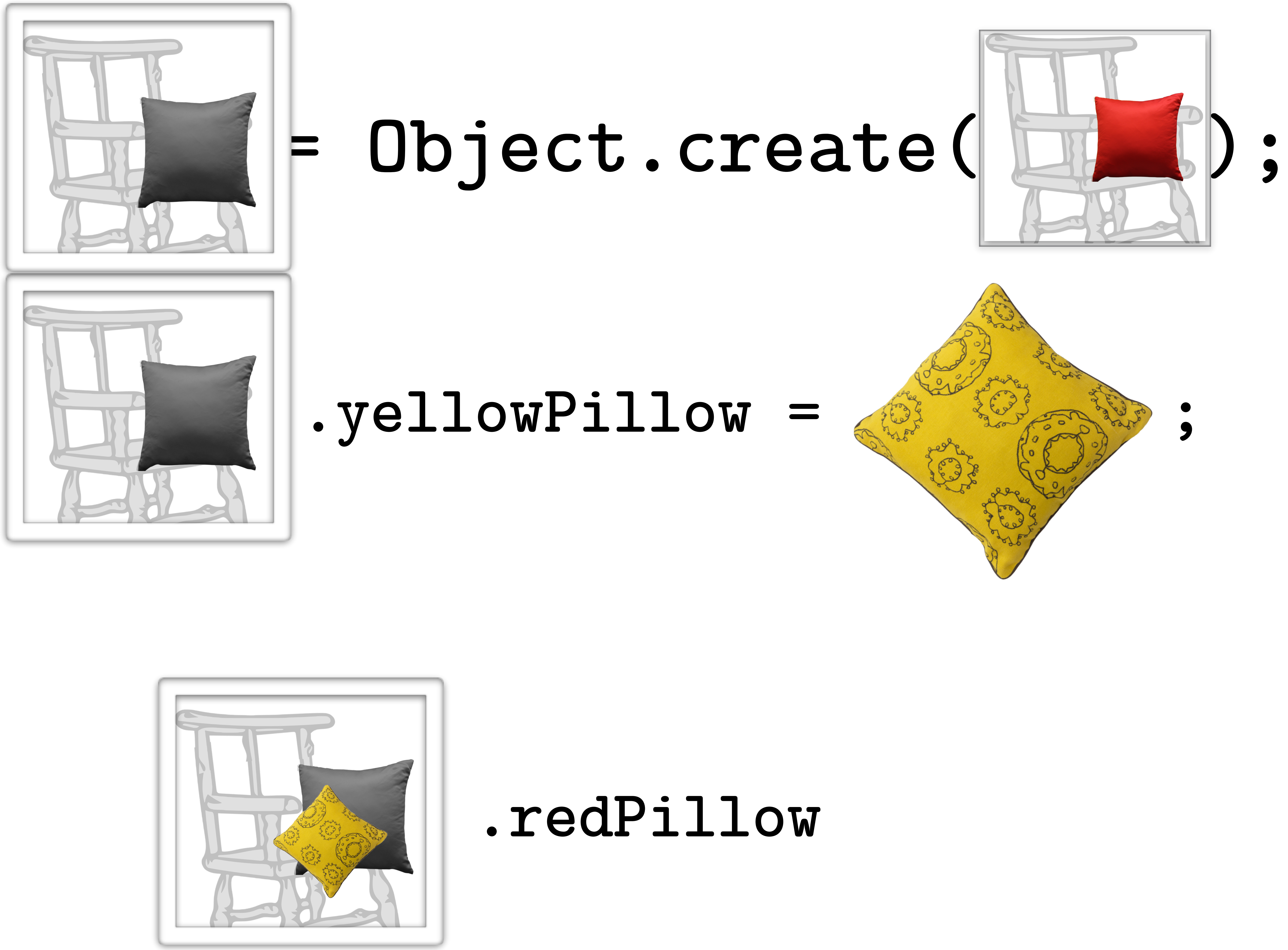
.yellowPillow = ;

.redPillow

Result

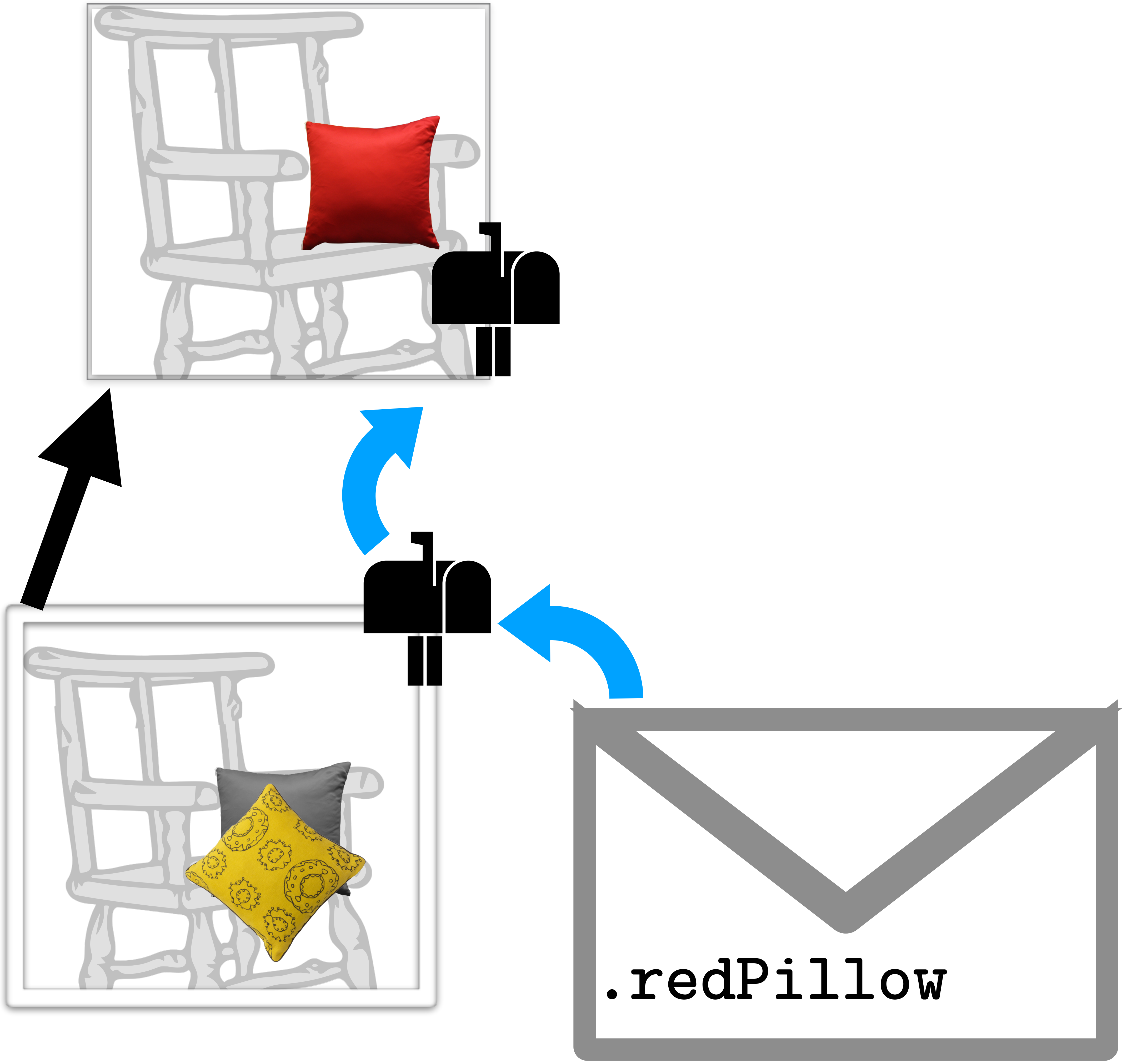


Commands

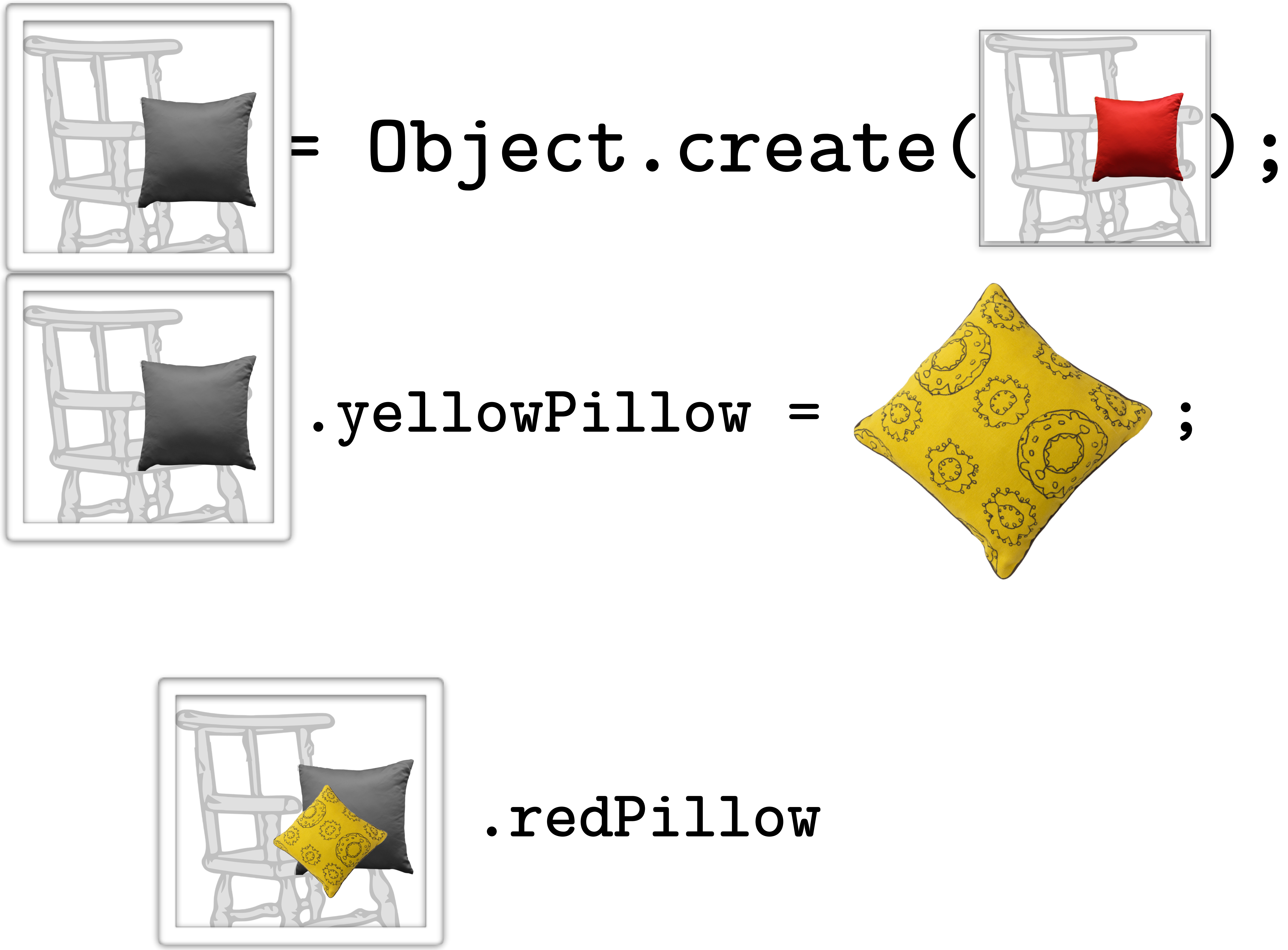


```
= Object.create( );

.yellowPillow = ;

.redPillow
```

Result

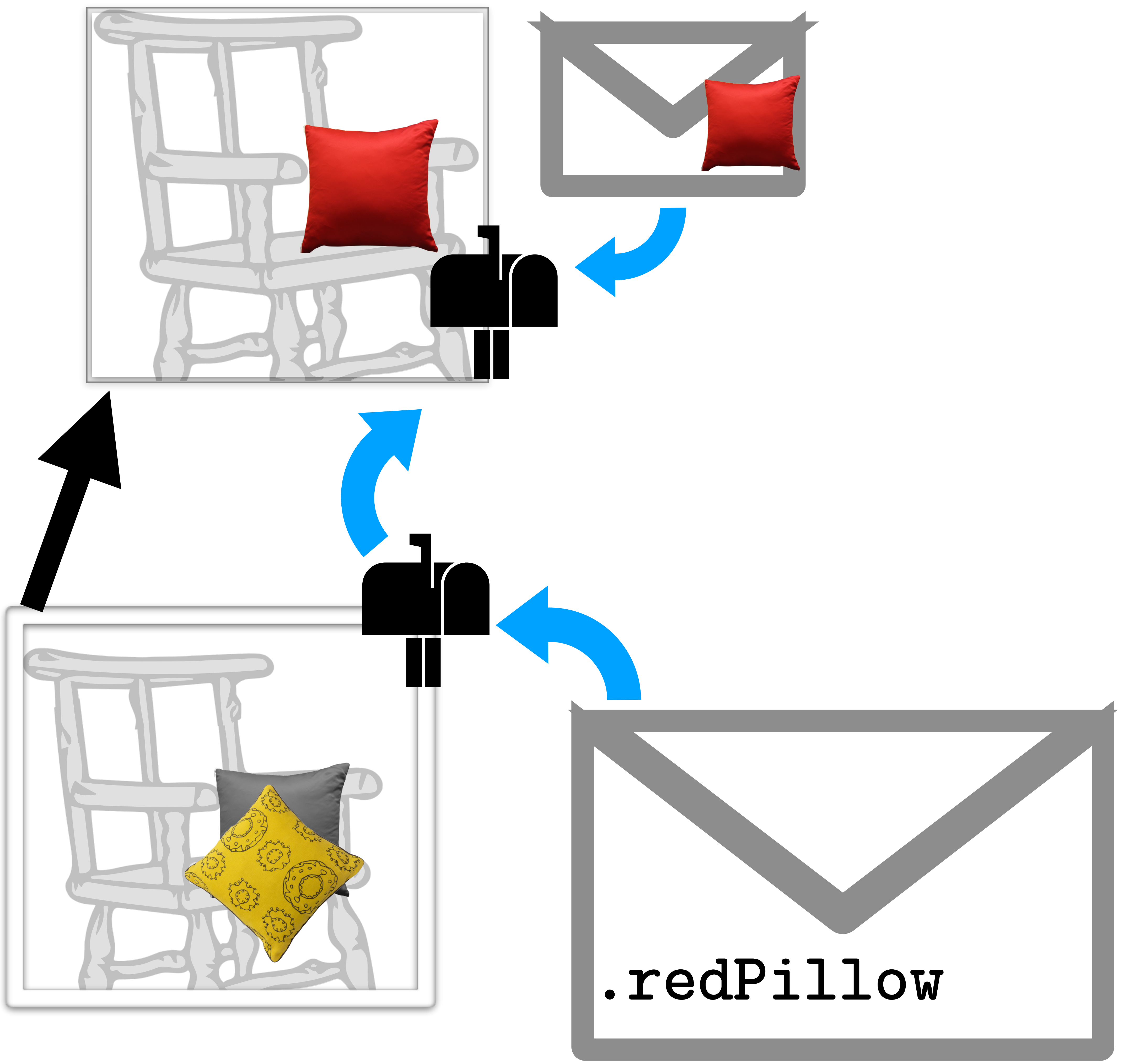


Commands

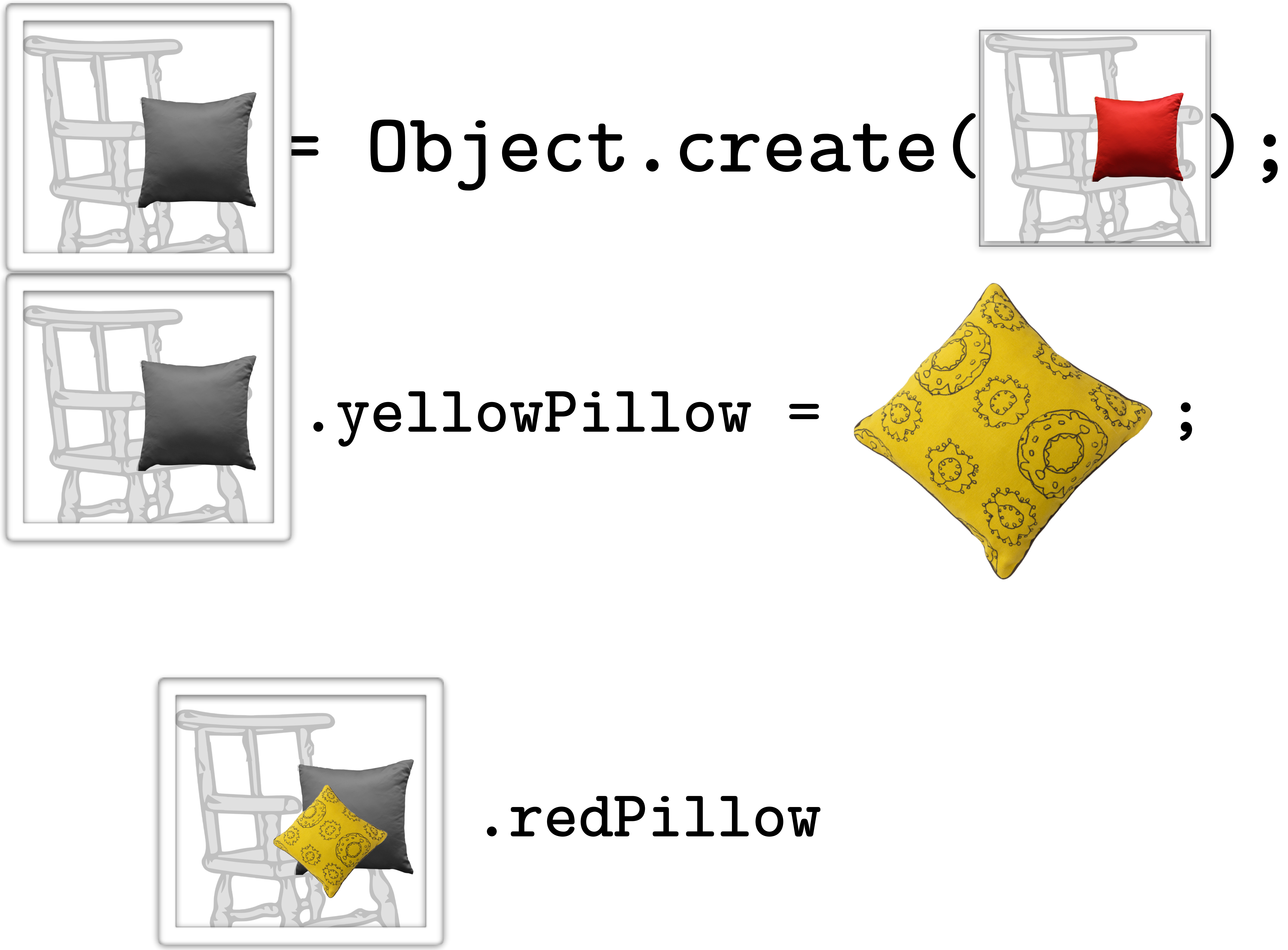


```
= Object.create( );
.yellowPillow = ;
.redPillow
```

Result

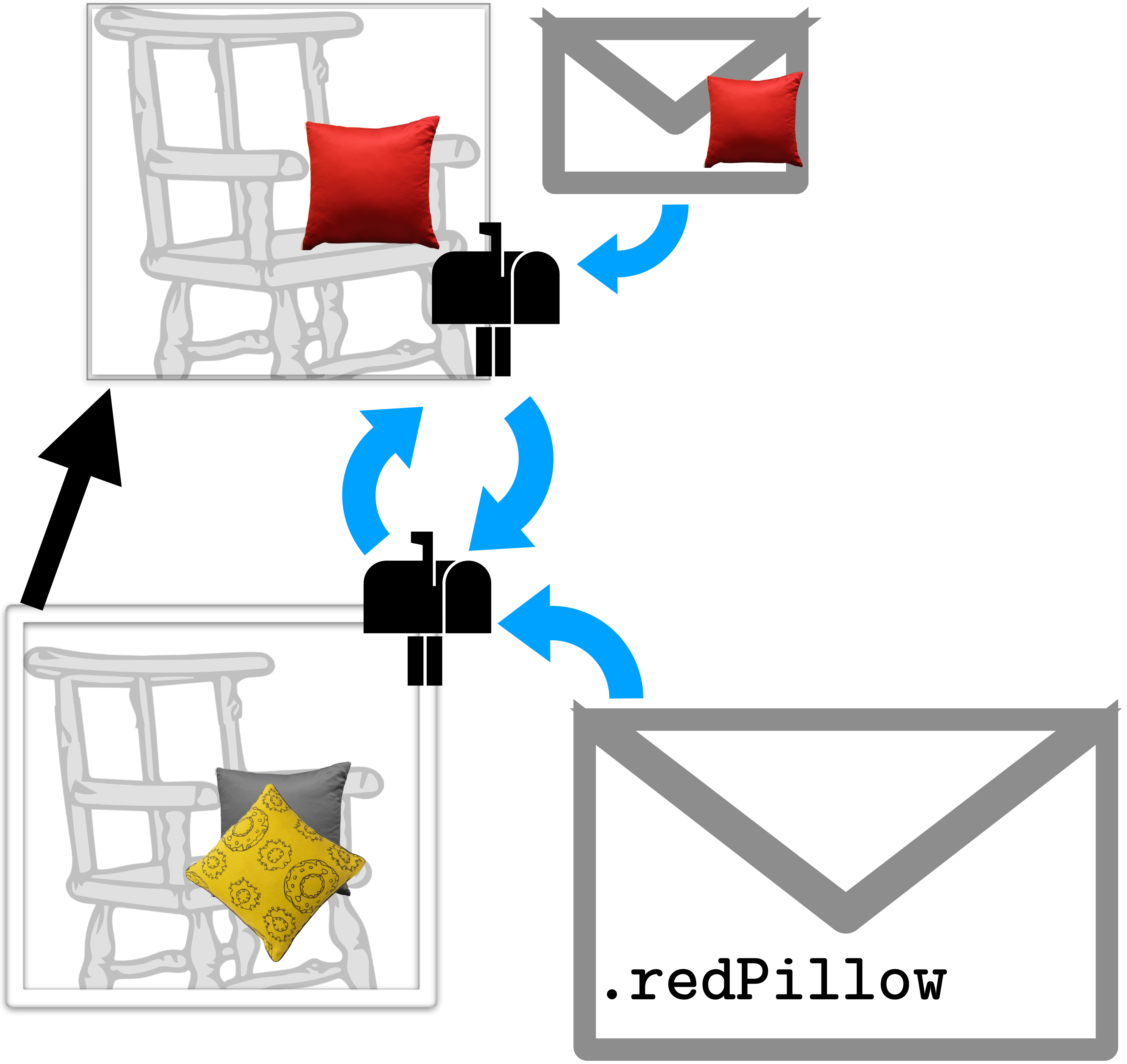


Commands

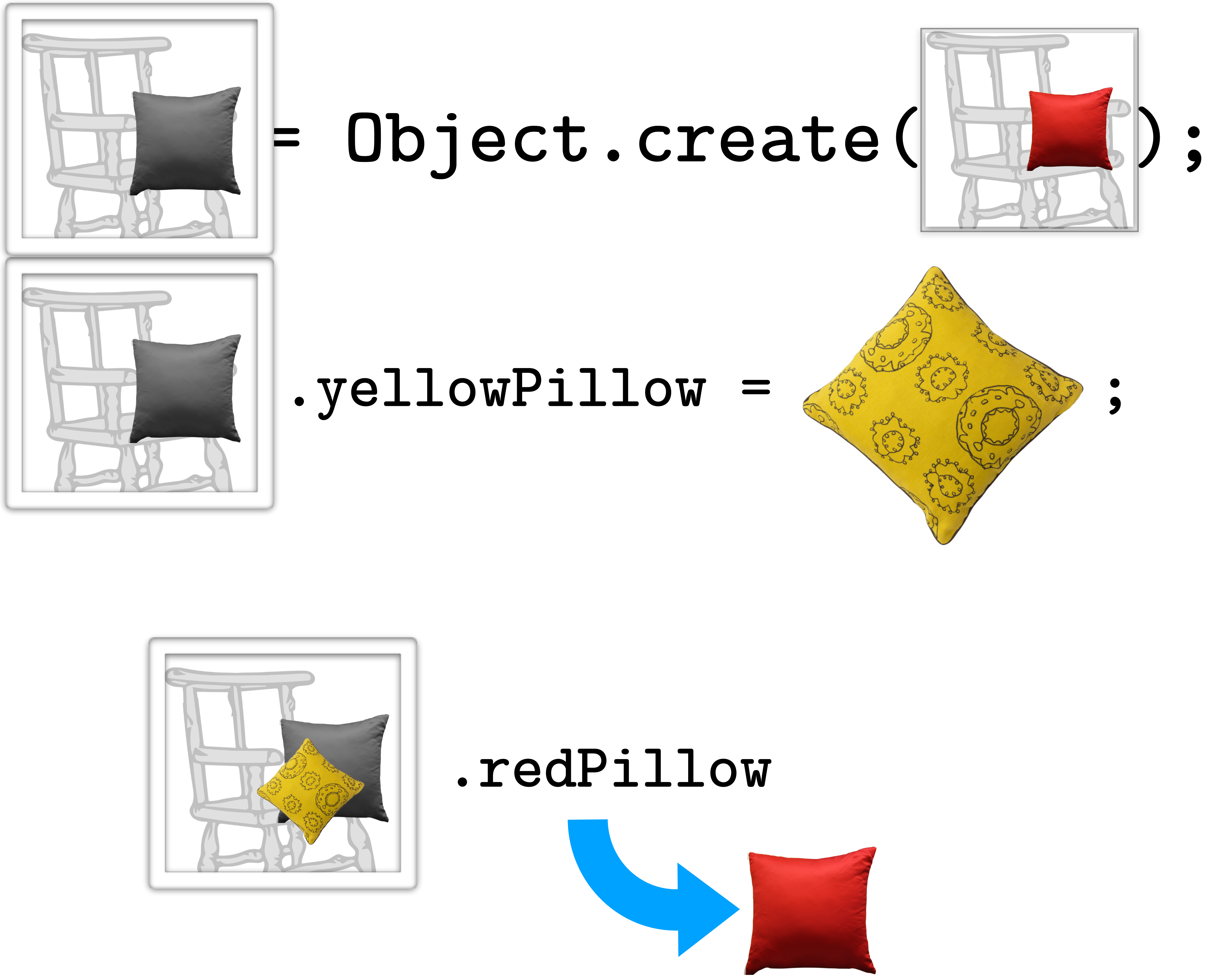


```
= Object.create( );
.yellowPillow = ;
.redPillow
```

Result

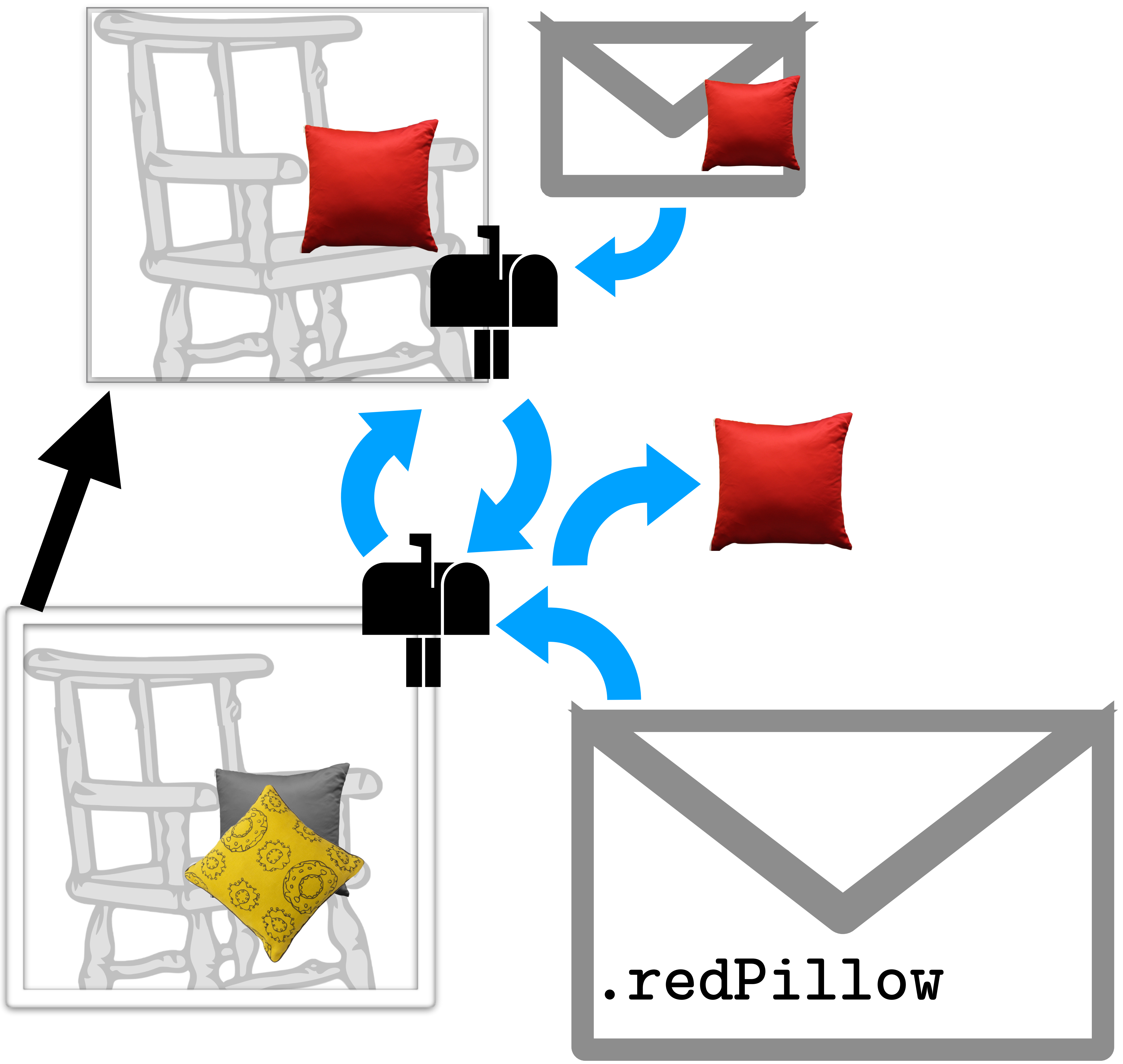


Commands



```
= Object.create(  );

.yellowPillow =  ;

.redPillow
```

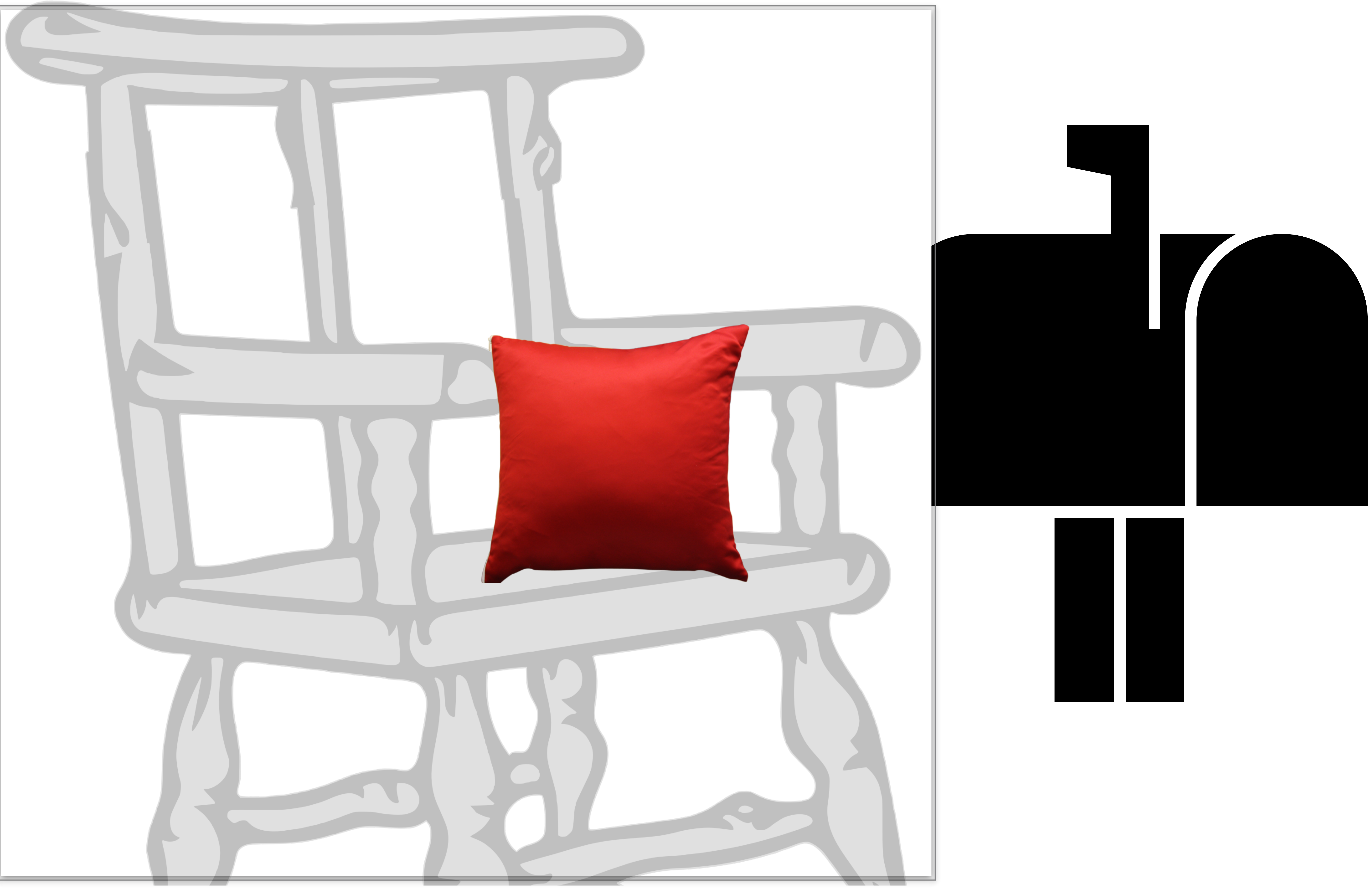
Result



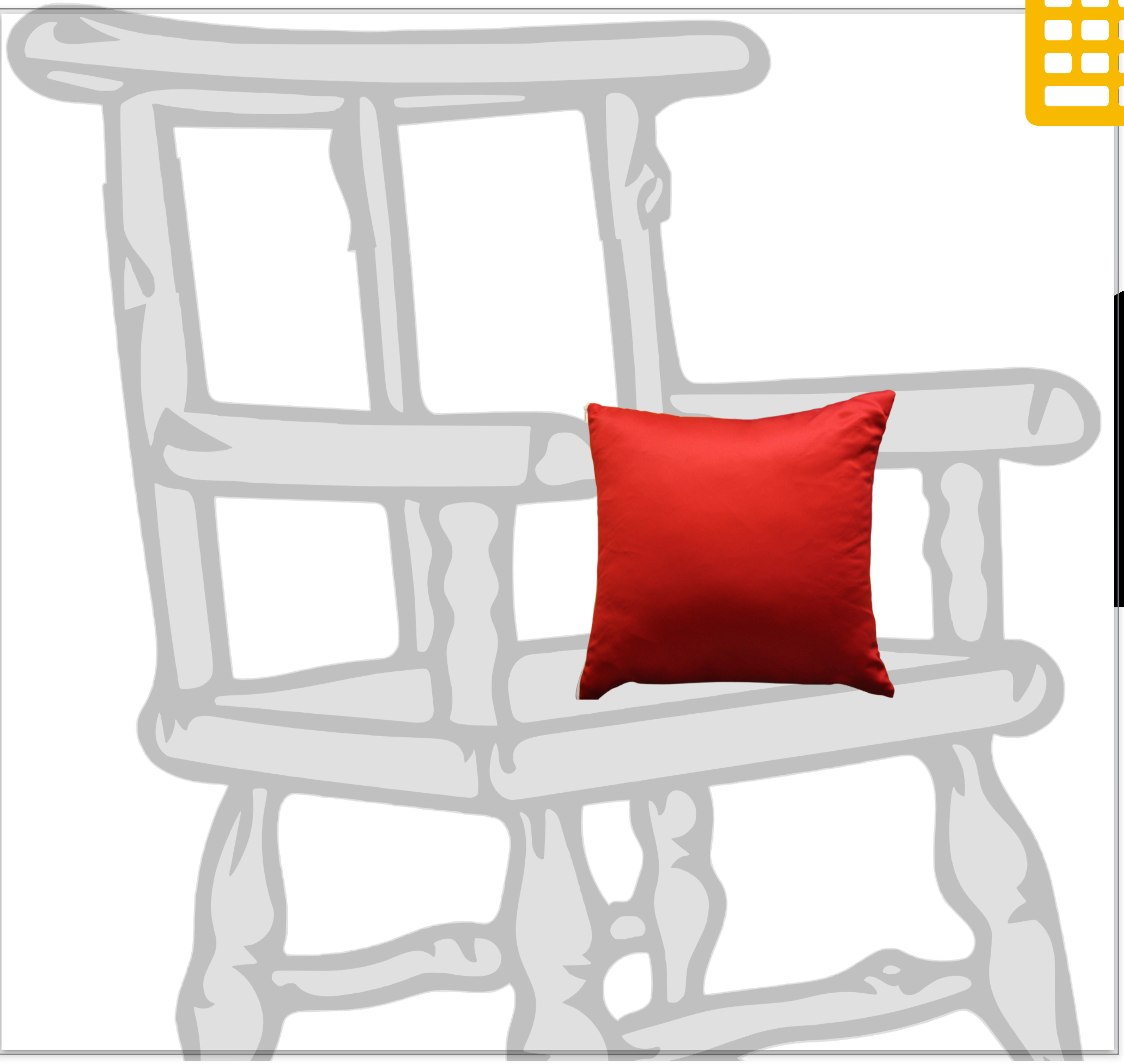
Objects

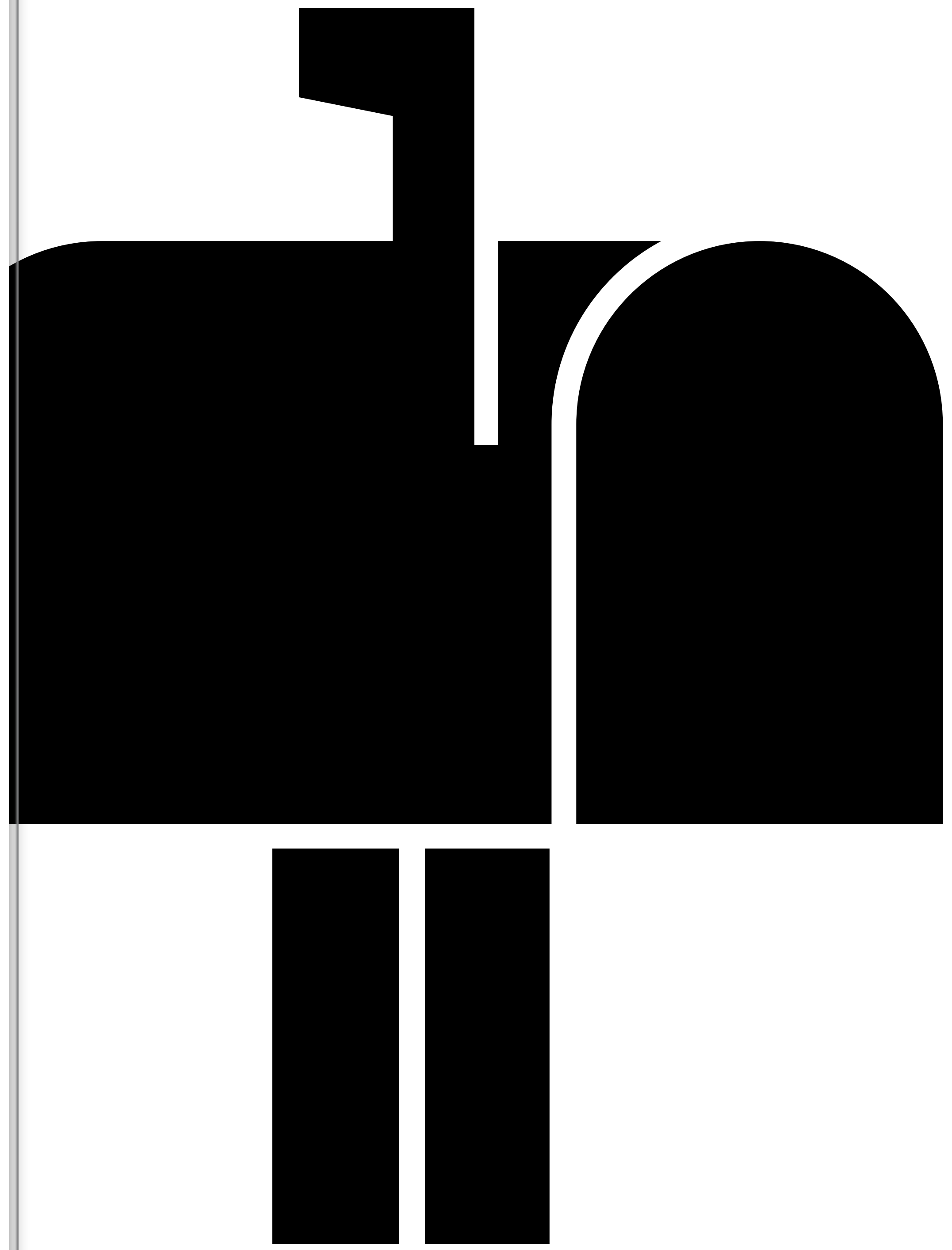
have

behavior



`pillowCalc() =`





`pillowCalc() =`

To:
Pillow
count?

`pillowCalc() =`

Behavior

To:

Pillow count?

Re:

1

Behavior

To:

Pillow count?

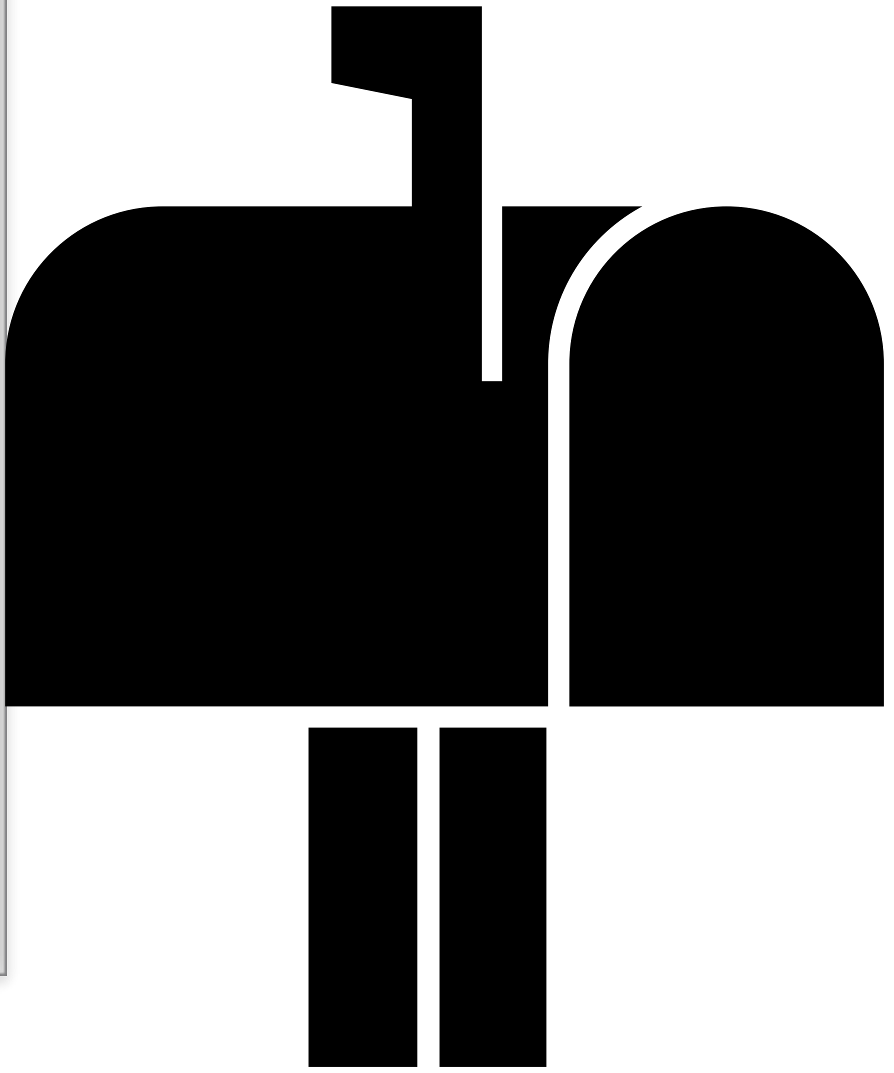
Re:
1
1
To:
Pillow
count?

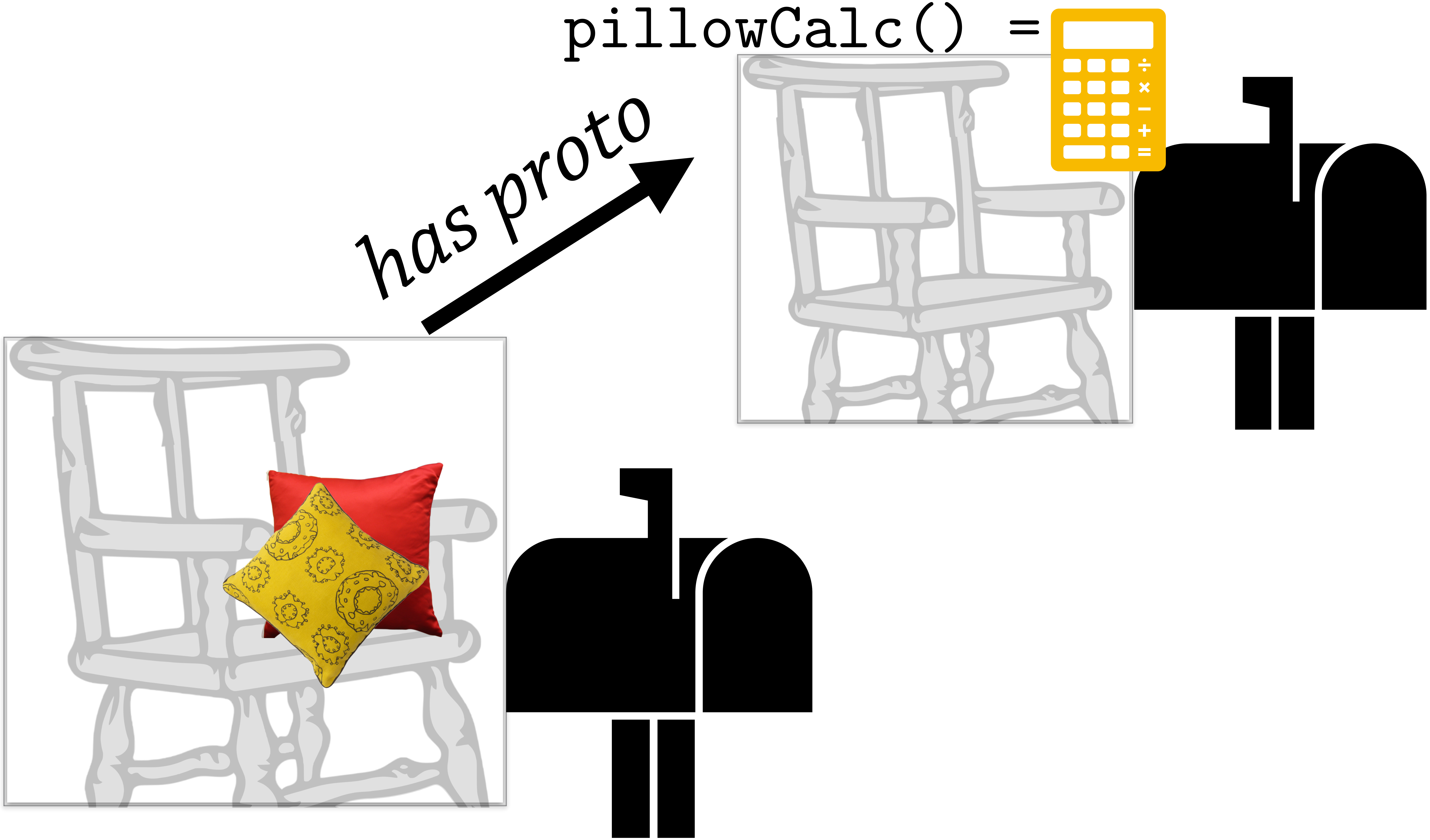
Behavior

Prototypes

serve

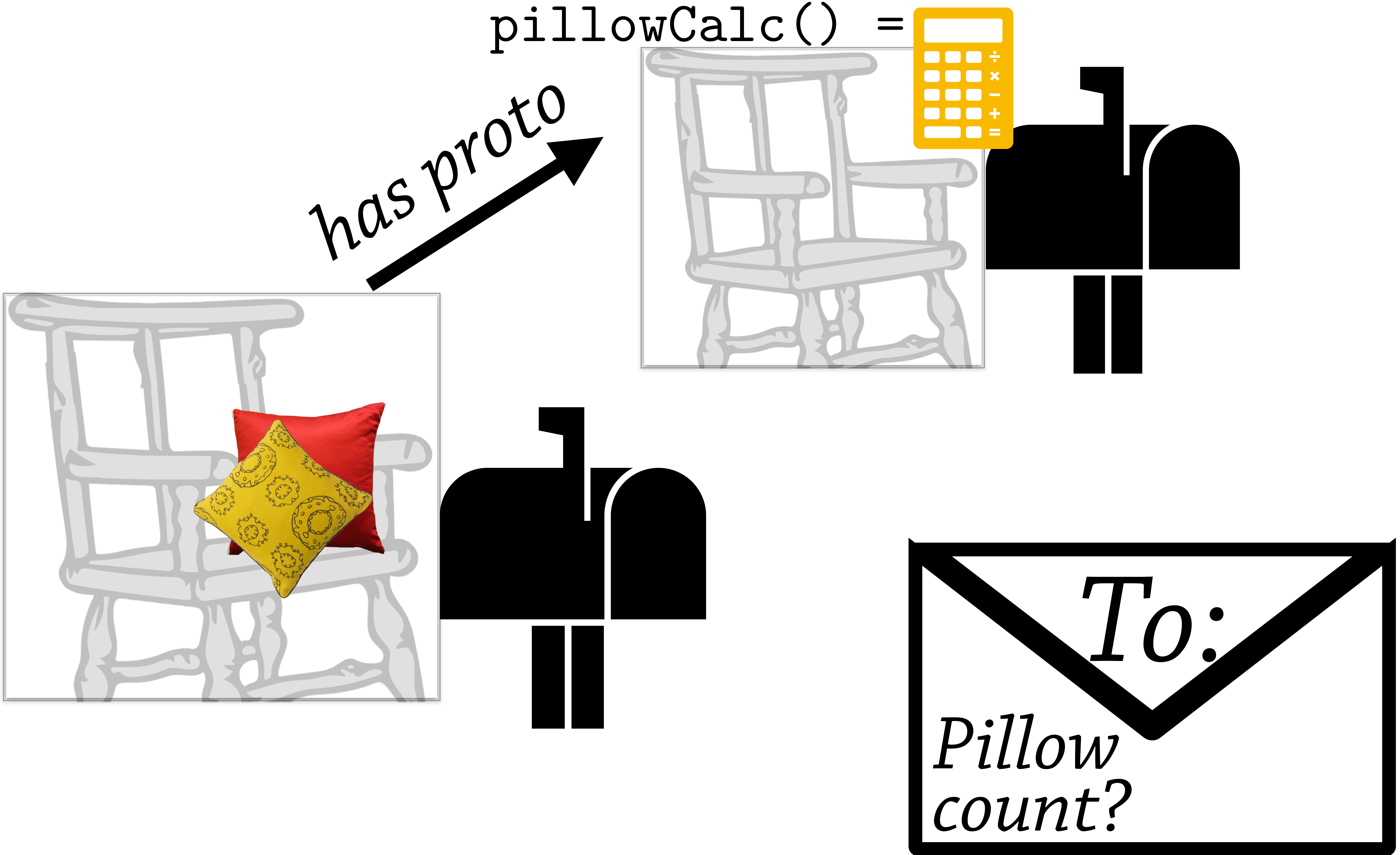
descendants





`pillowCalc() =`

has proto

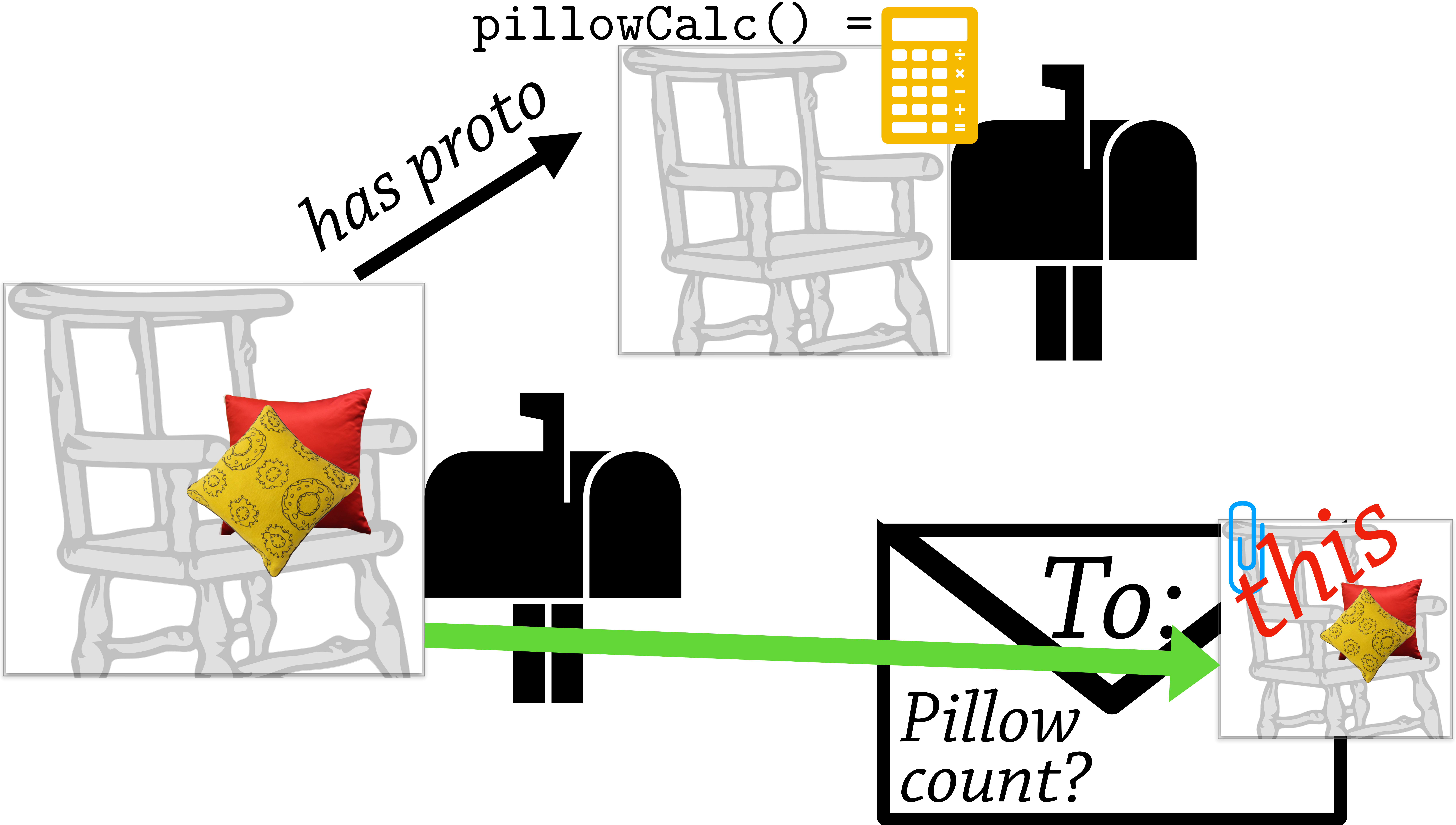


pillowCalc() =

has proto

To:

Pillow count?



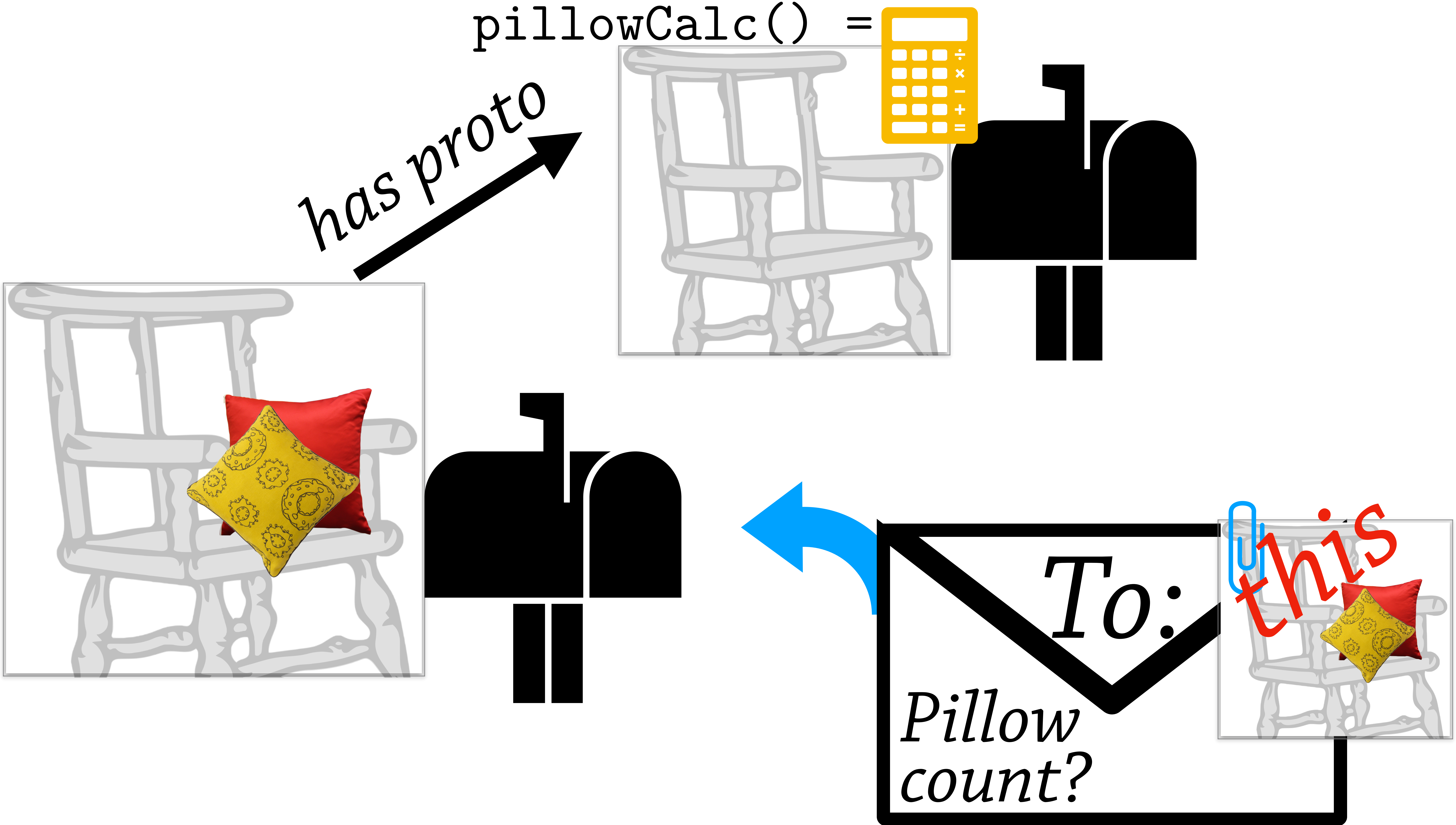
pillowCalc() =

has proto

To:

this

Pillow count?



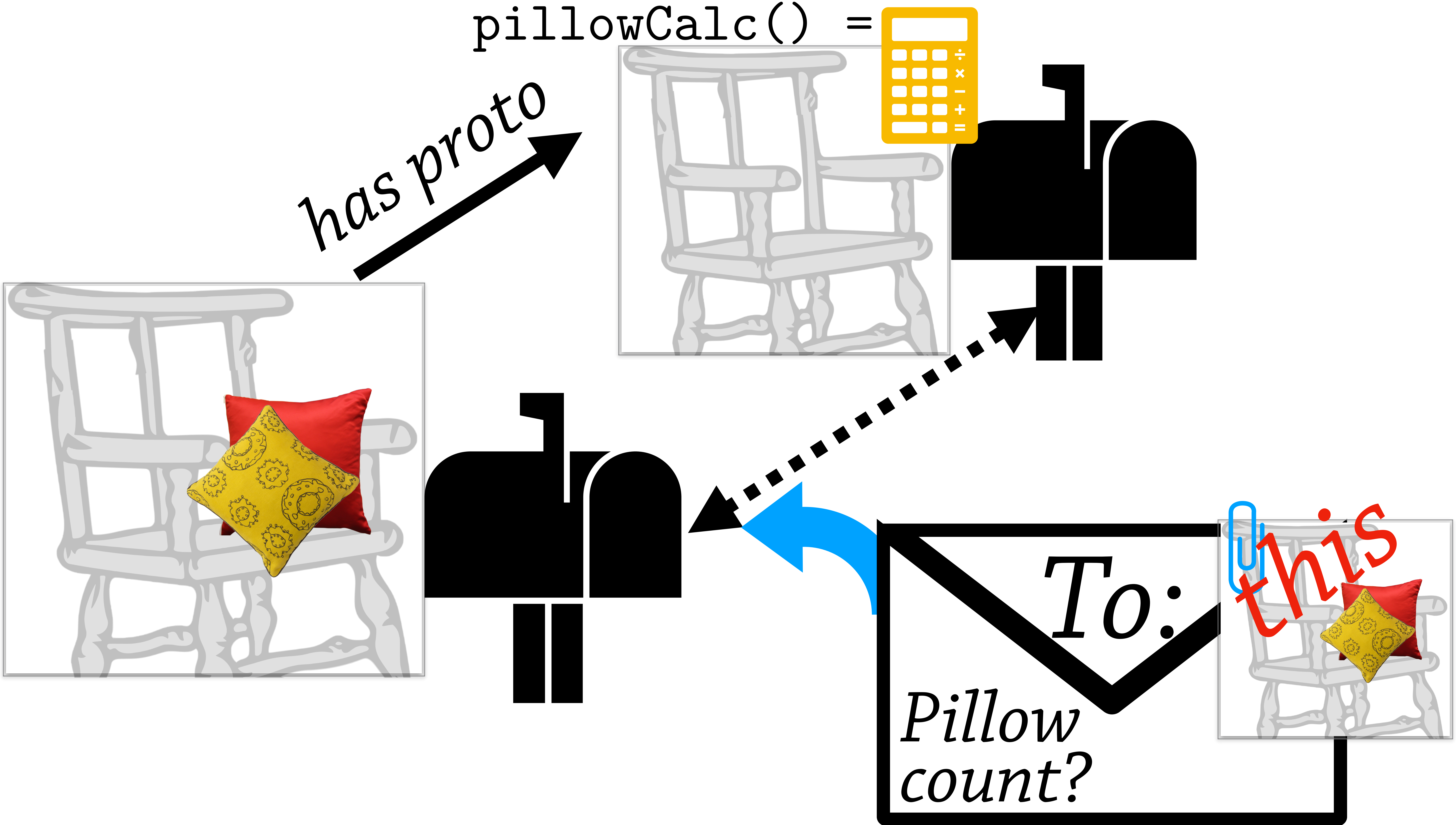
pillowCalc() =

has proto

To:

this

Pillow count?



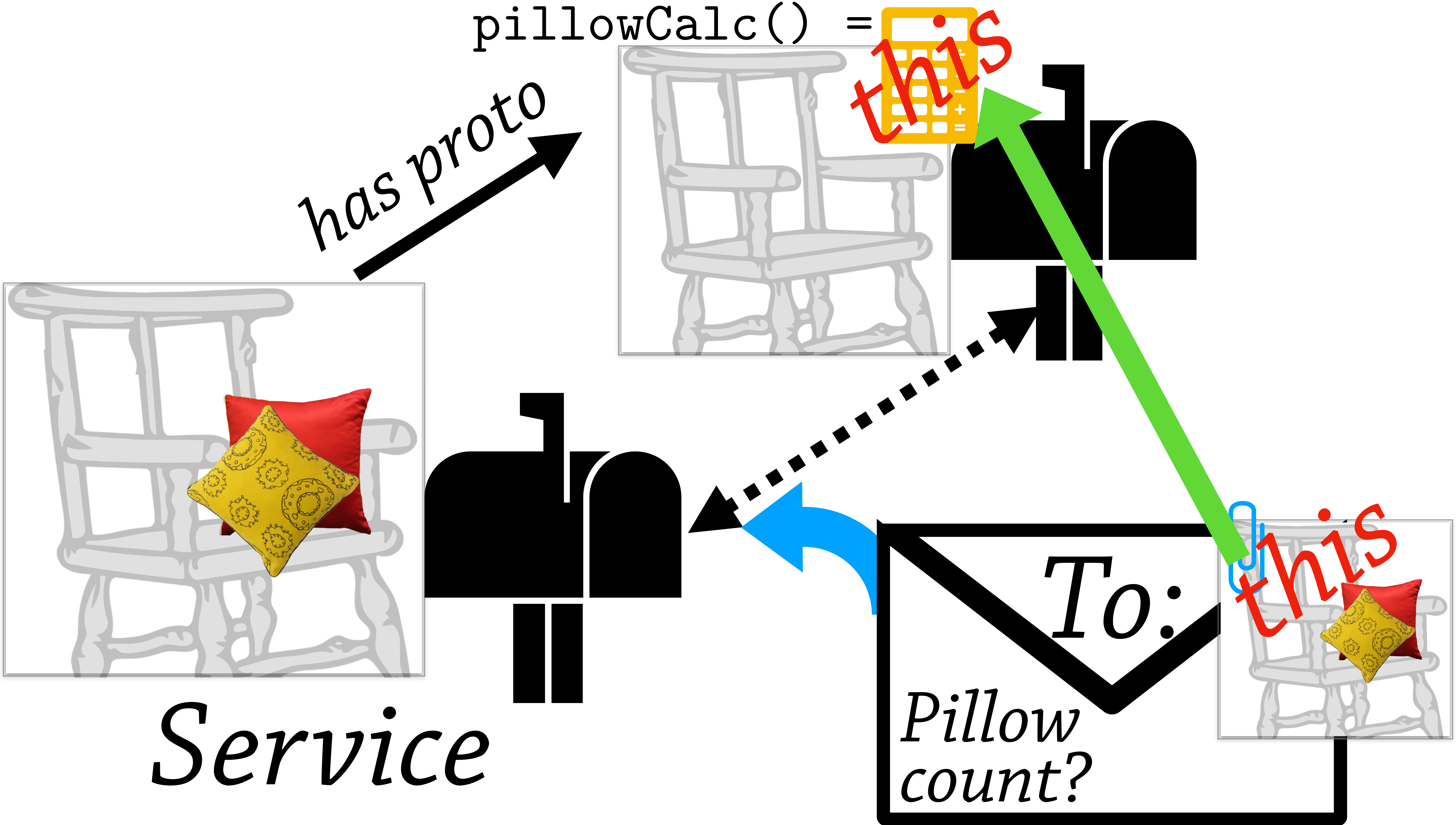
pillowCalc() =

has proto

To:

this

Pillow count?



`pillowCalc() =`

this

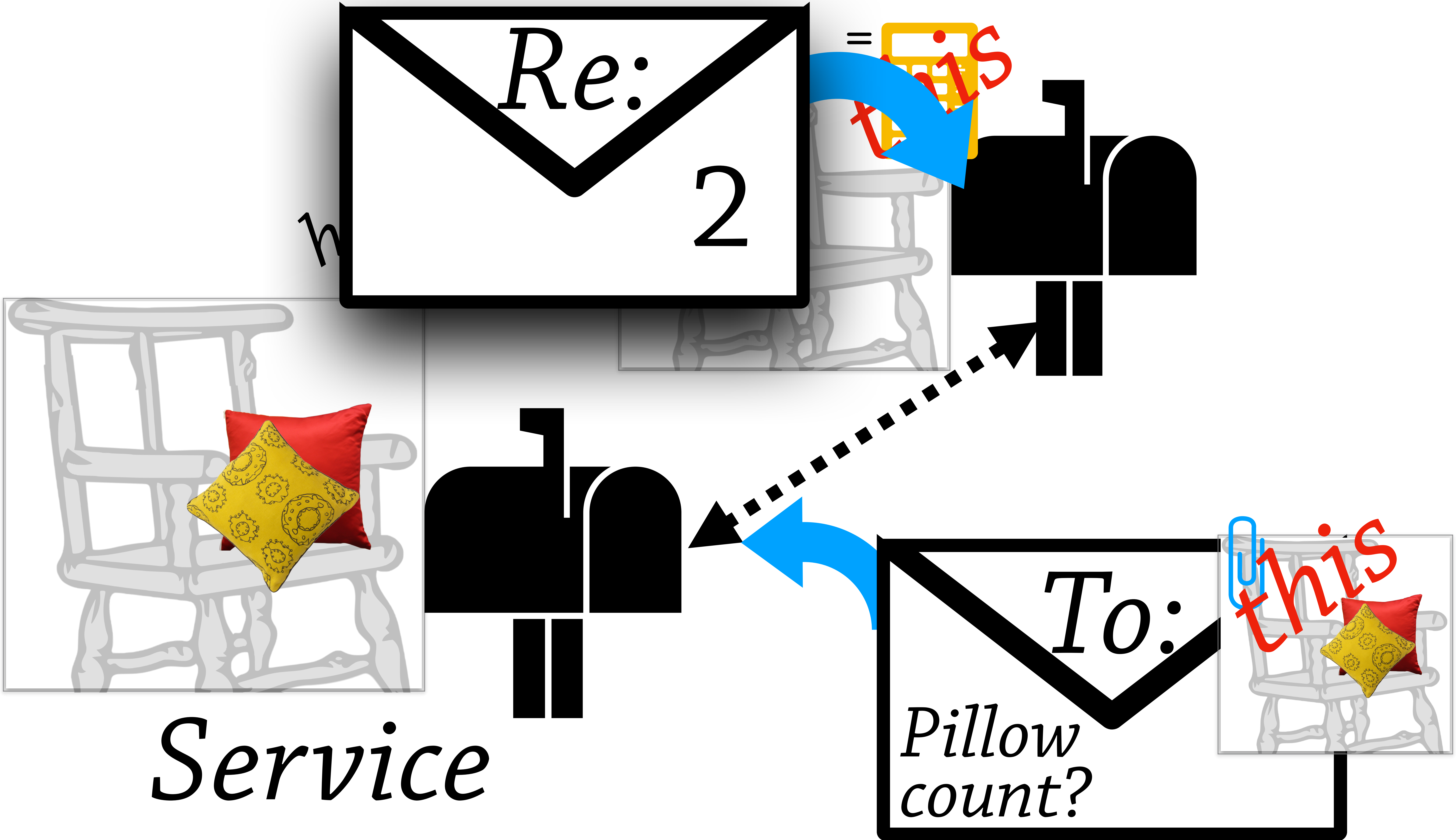
has proto

Service

To:

Pillow count?

this



Re:

2

=

this

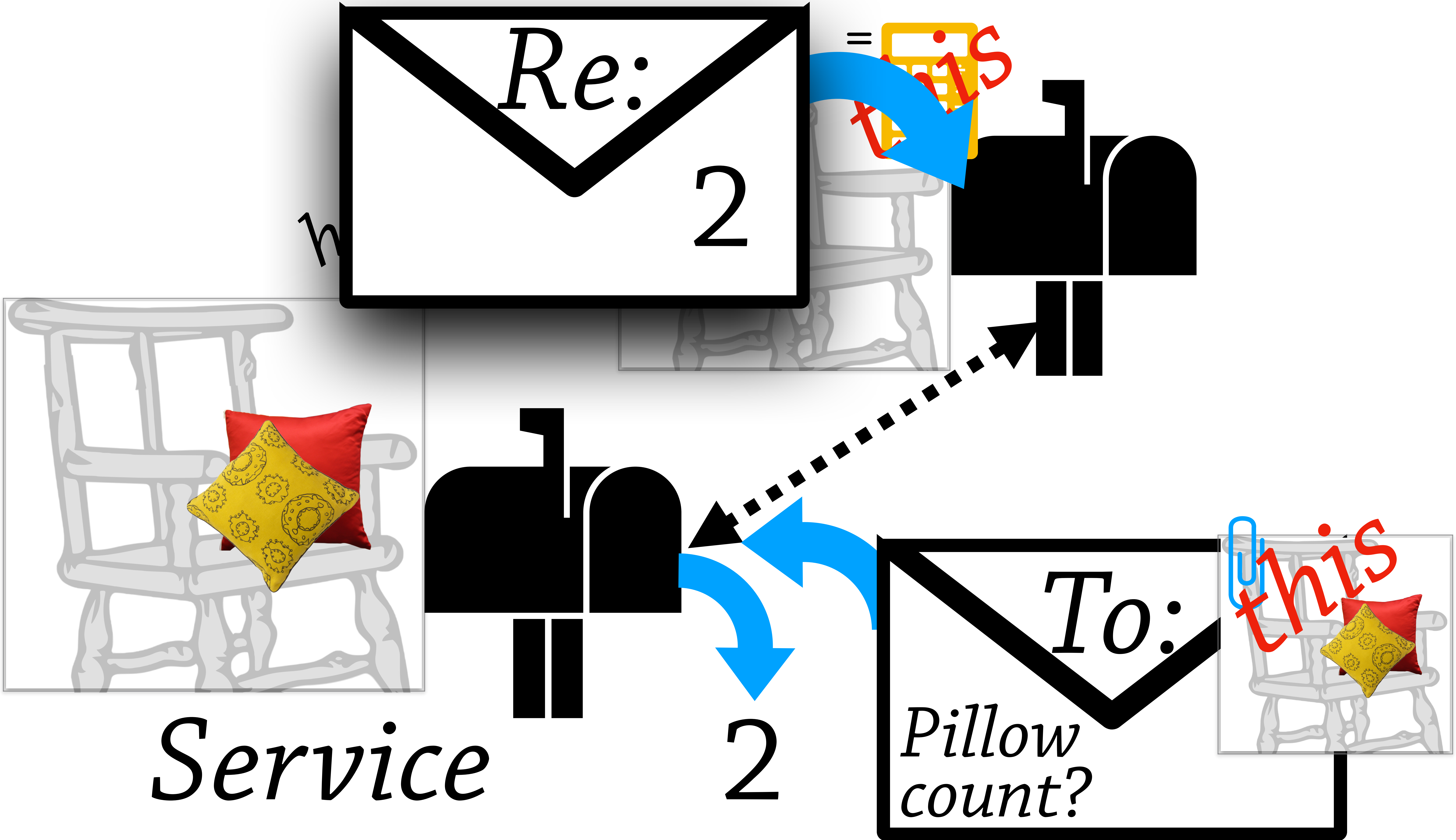
h

Service

To:

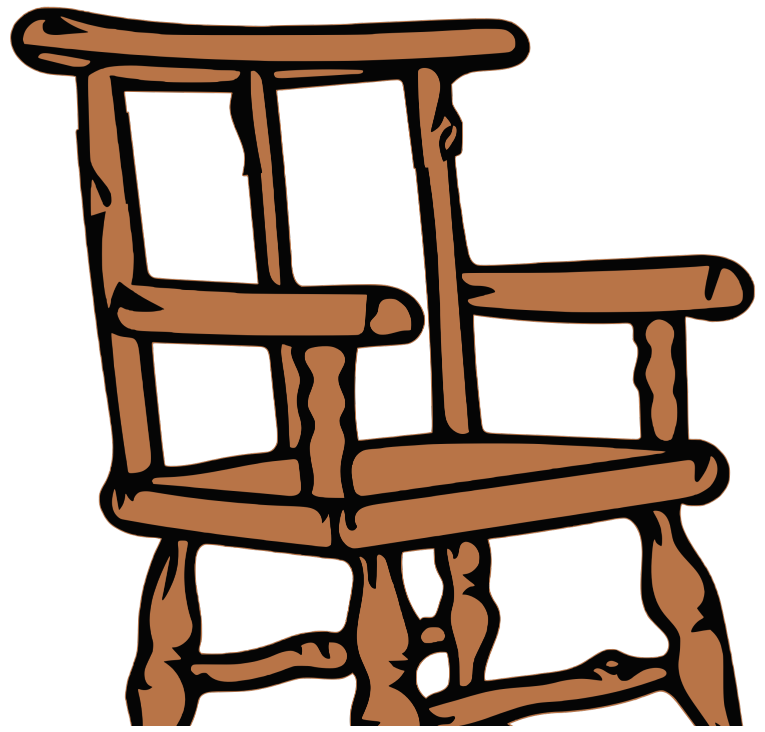
Pillow count?

this

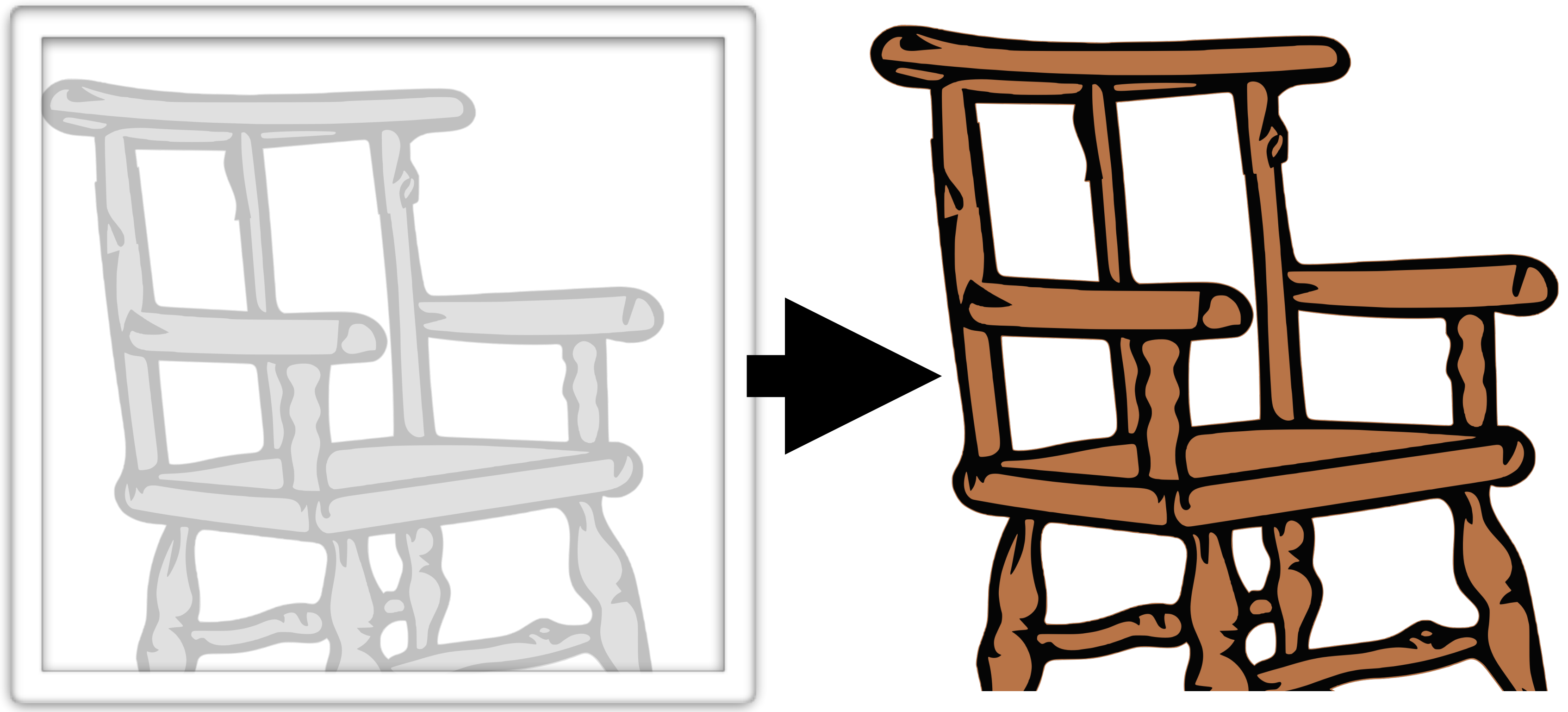

Re:
2
=
this
To:
Pillow count?
this
2
Service

The essence of *prototypal* **objects**:

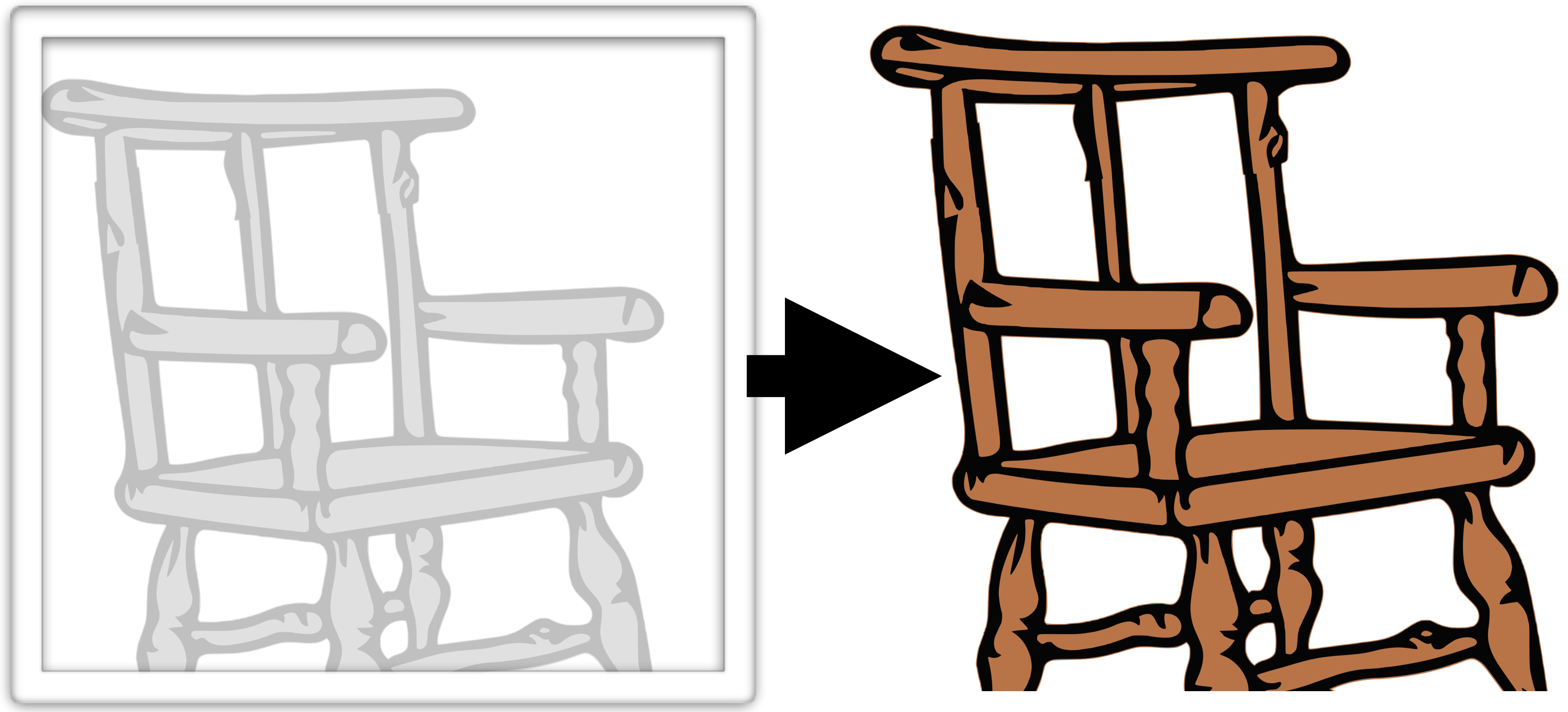
Objects are **prototypes**



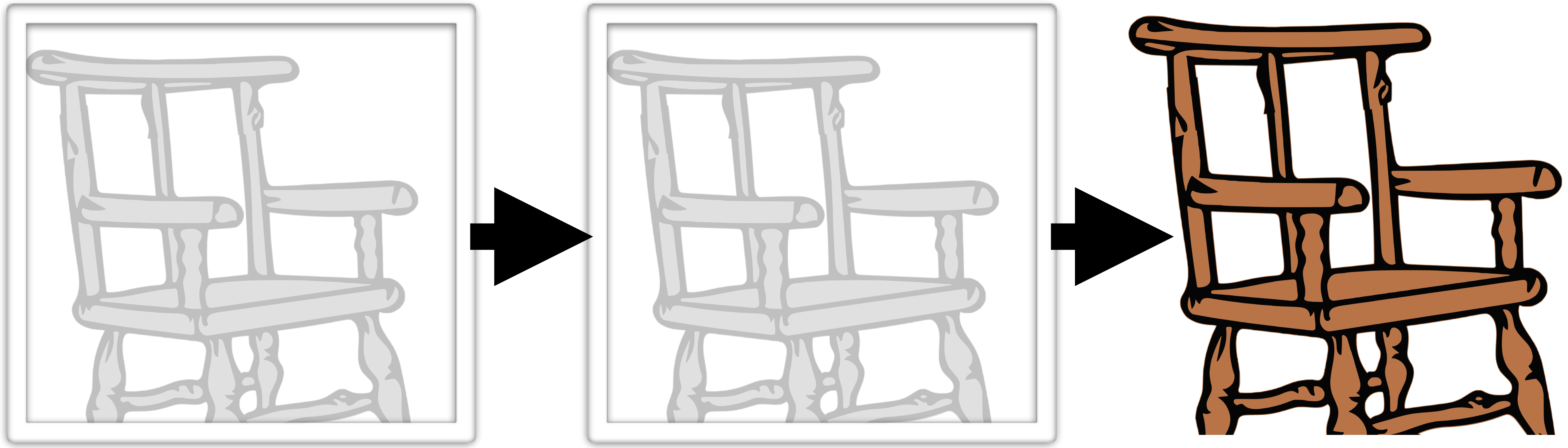
Objects are **prototypes**



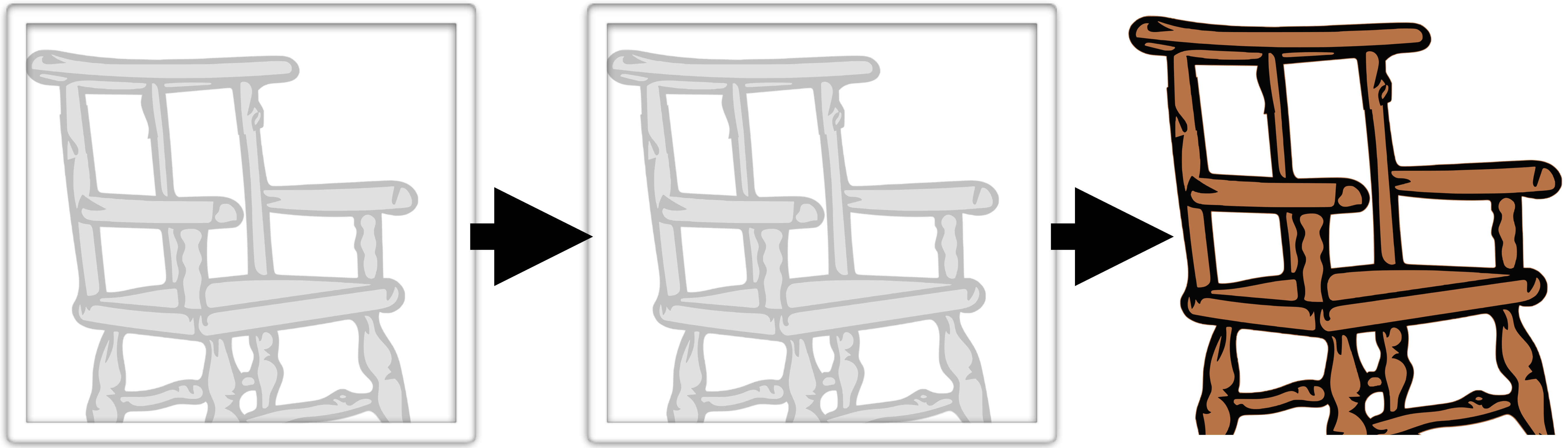
Objects = ***prototypes***



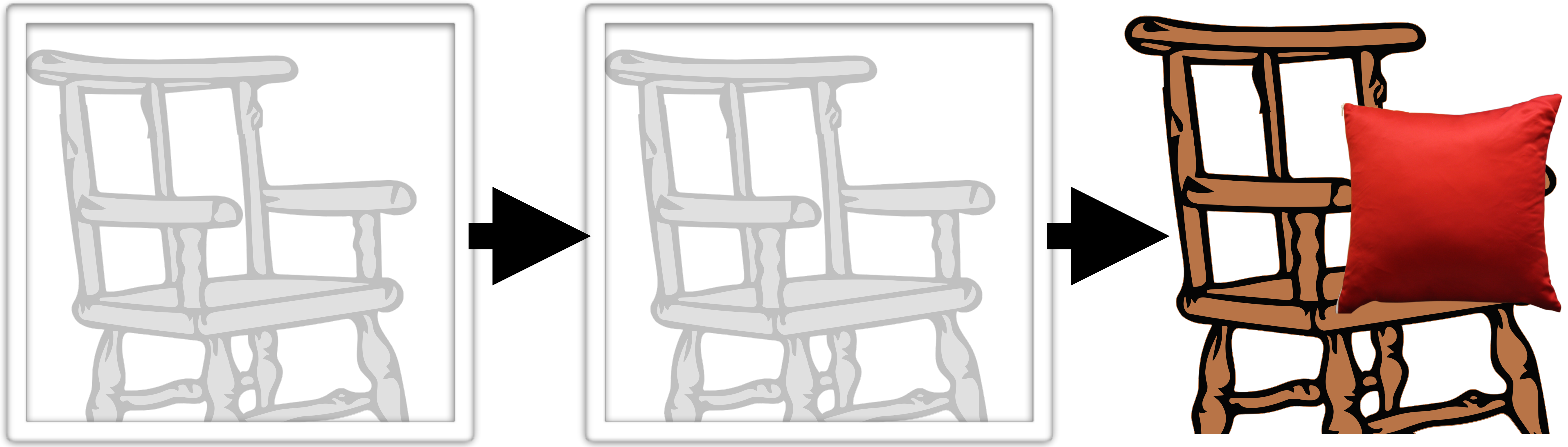
Objects = ***prototypes***



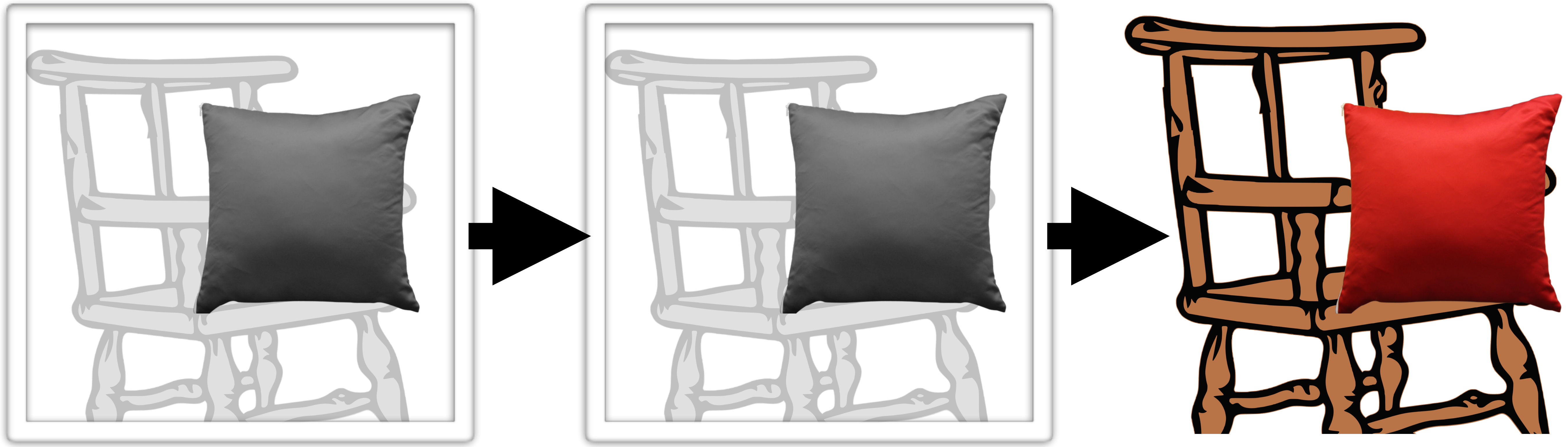
Prototypes *share state* with **children**



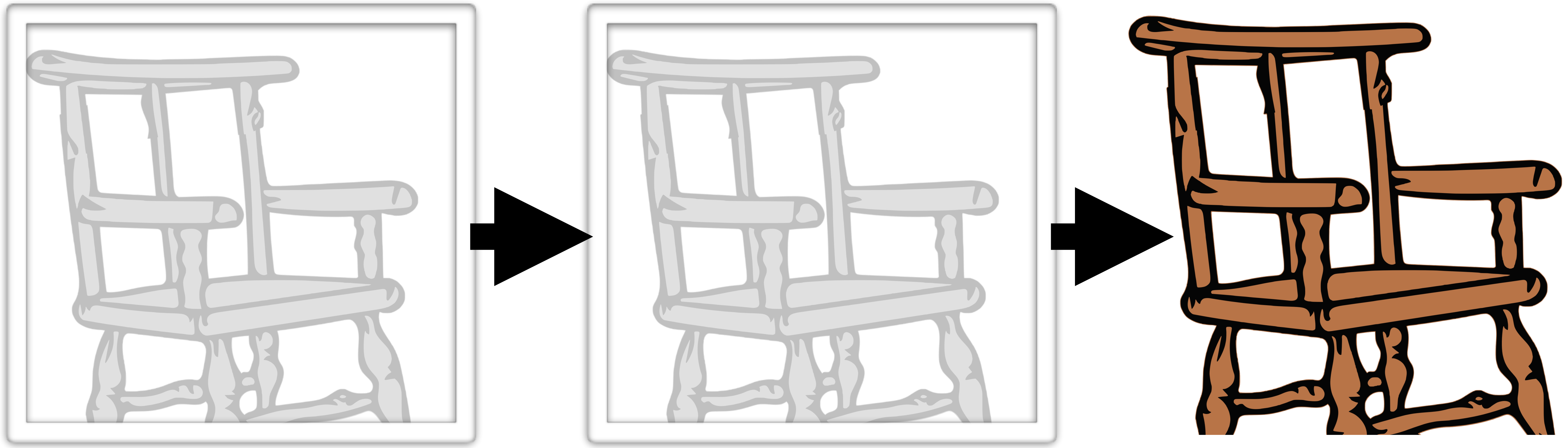
Prototypes *share state* with **children**



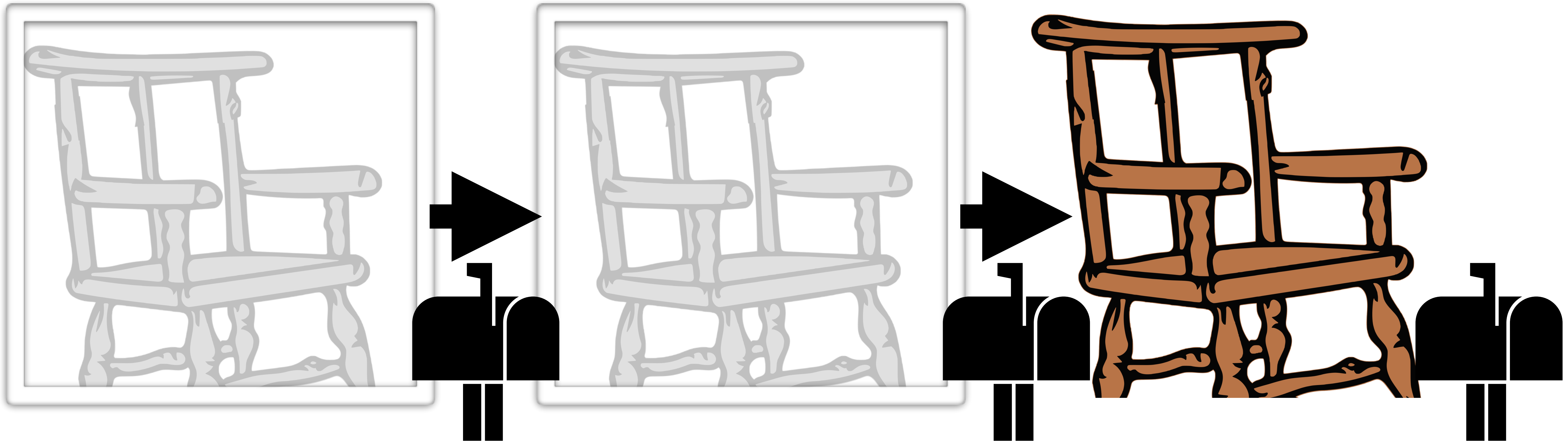
Prototypes *share state* with **children**



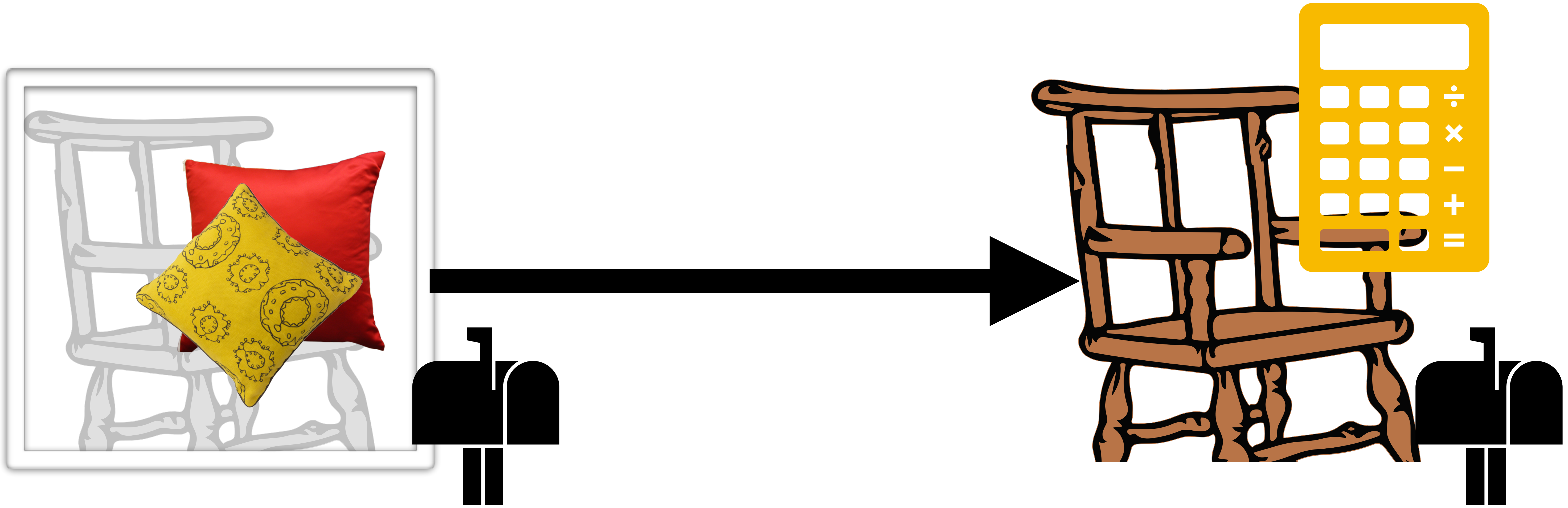
Objects communicate via **messages**



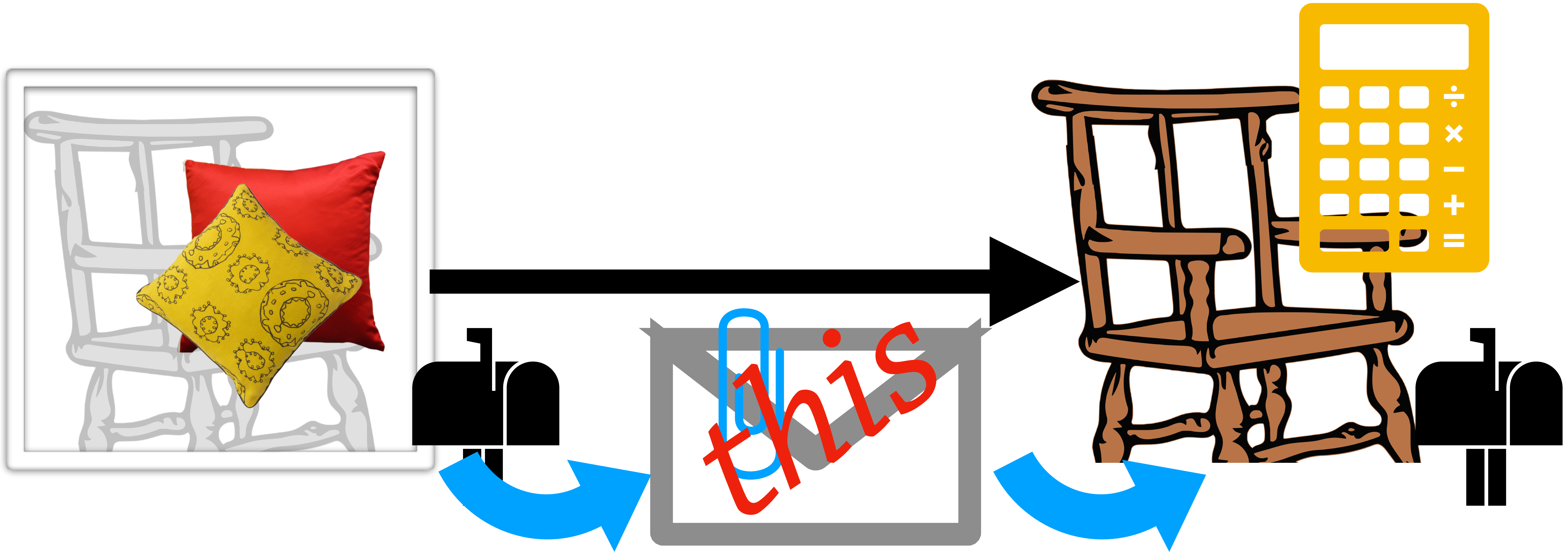
Objects communicate via **messages**



Prototypes *serve* **children**



Prototypes *serve* **children**



JavaScript is
prototypes
at the **bottom**

JavaScript

JavaScript

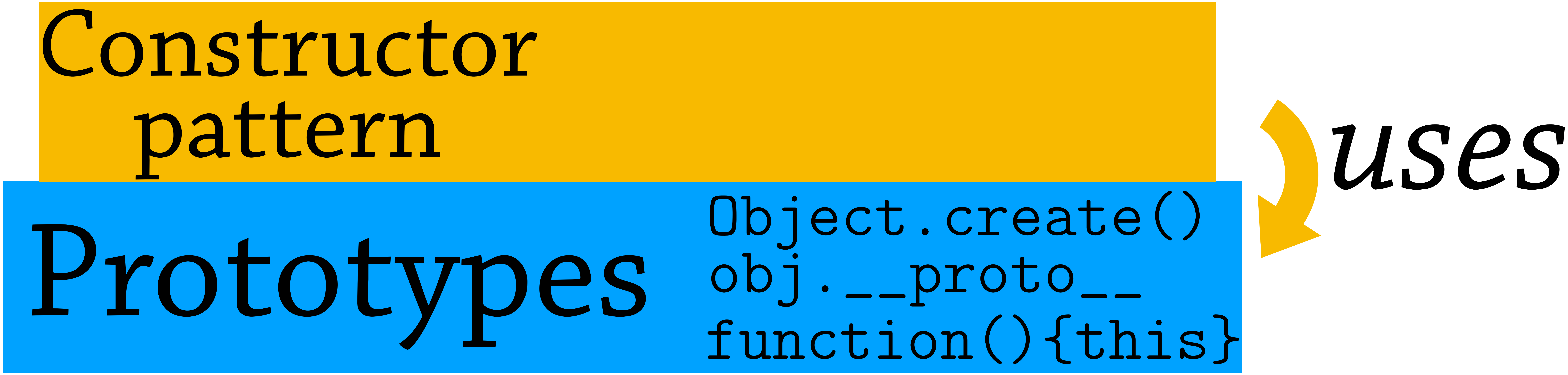
Prototypes

JavaScript

Prototypes

```
Object.create()
obj.__proto__
function(){this}
```

JavaScript



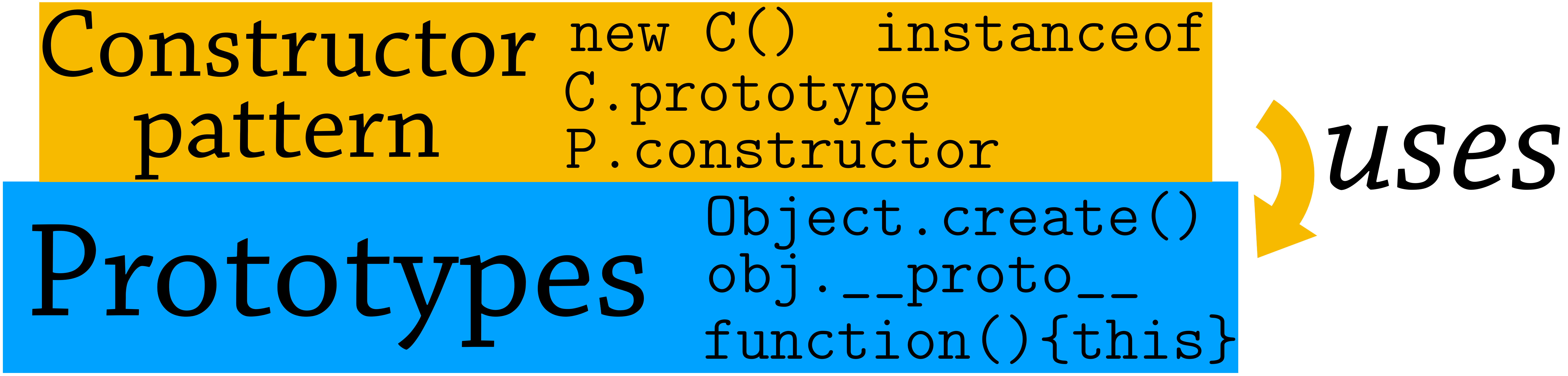
Constructor pattern

uses

Prototypes

```
Object.create()
obj.__proto__
function(){this}
```

JavaScript



Constructor pattern

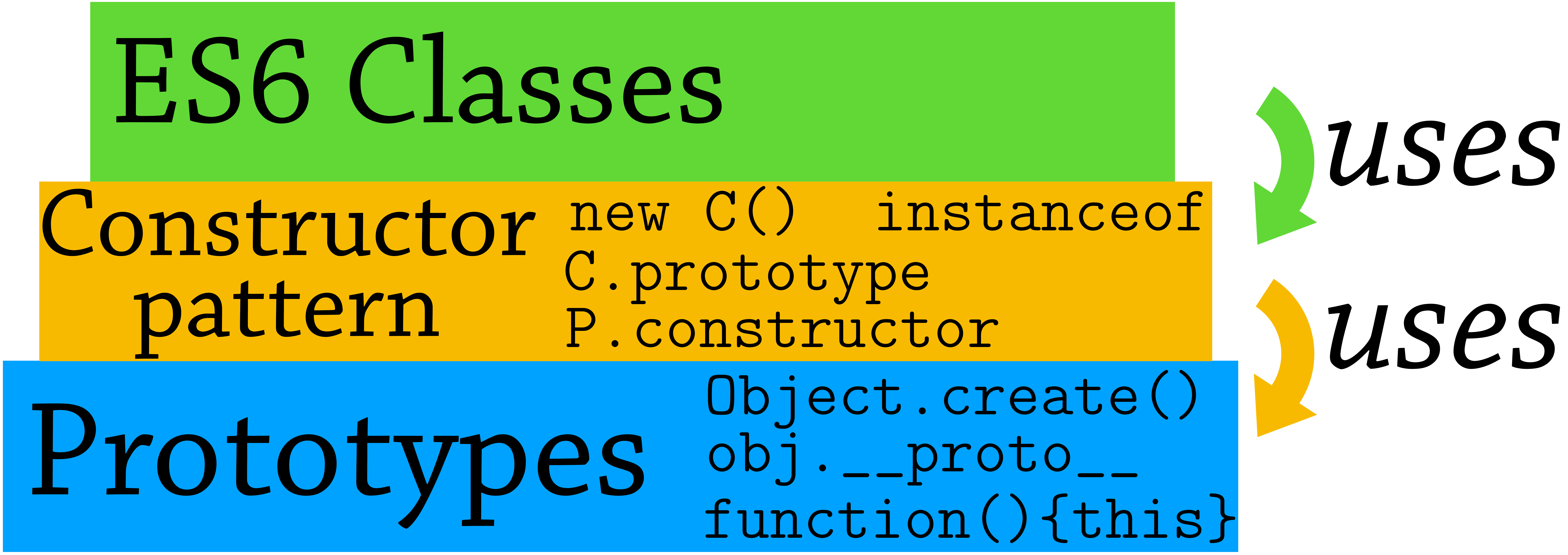
new C() instanceof
C.prototype
P.constructor

uses

Prototypes

Object.create()
obj.__proto__
function(){this}

JavaScript



ES6 Classes

uses

Constructor pattern

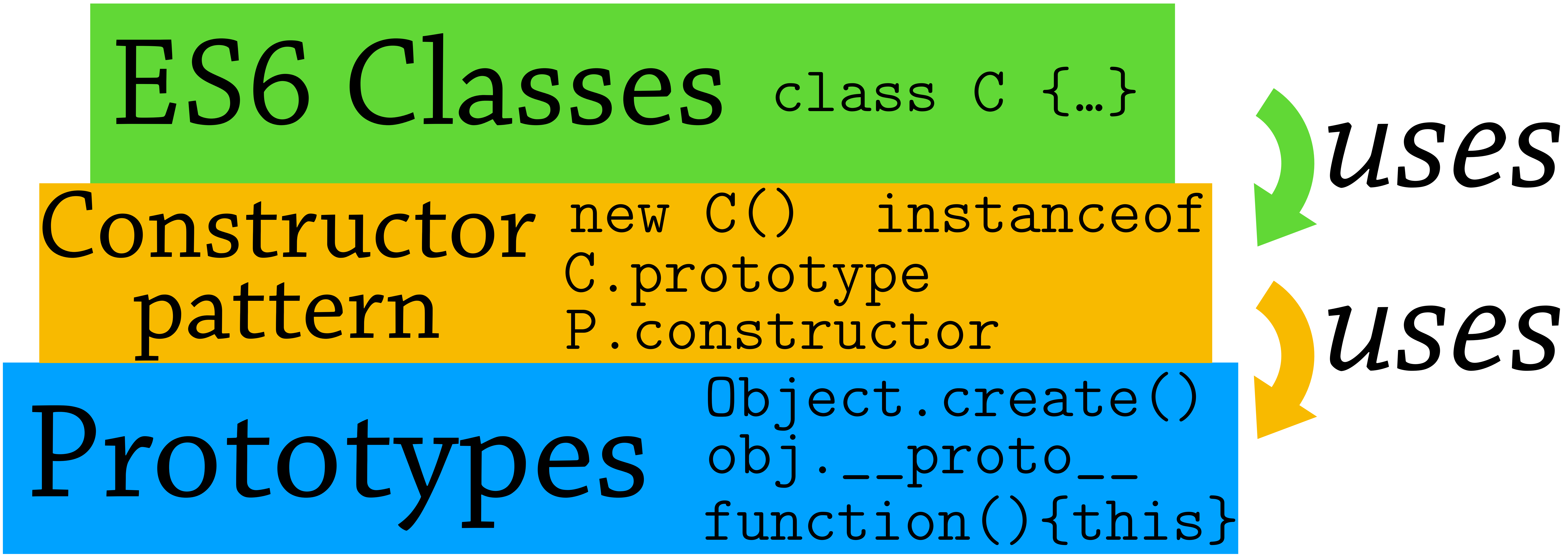
`new C()` `instanceof`
`C.prototype`
`P.constructor`

uses

Prototypes

`Object.create()`
`obj.__proto__`
`function(){this}`

JavaScript



ES6 Classes `class C {…}`

Constructor pattern `new C()` `instanceof` `C.prototype` `P.constructor`

Prototypes `Object.create()` `obj.__proto__` `function(){this}`

uses

uses

JavaScript

ES6 Classes `class C {…}`

Constructor pattern `new C()` `instanceof` `C.prototype` `P.constructor`

Prototypes `Object.create()` `obj.__proto__` `function(){this}`

uses

uses

This talk

JavaScript

Thanks!

ambrosebs.com

ES6 Classes `class C {…}`

Constructor pattern `new C()` `instanceof` `C.prototype` `P.constructor`

Prototypes `Object.create()` `obj.__proto__` `function(){this}`

uses

uses

This talk