Javascript
#03

Matthieu Nicolas
Licence Pro CIASIE
Slides par Christophe Bouthier

Plan

- Héritage

Héritage

Javascript
#03

Constructeur

- Fonction
  - appelée avec **new**
  - *this* = nouvel objet vide
  - défini le contenu de *this*
  - retourne *this*

Example

```
1   const Character = function () {
2       this.hp = 100
3       this.getHP = function () {
4           return this.hp
5       }
6   }
7
8   const charac = new Character()
9   charac.getHP()  // 100
10
```

En mémoire

```
const Character = function () {
    this.hp = 100
    this.getHP = function () {
        return this.hp
    }
}

const charac = new Character()
charac.getHP()  // 100
```

En mémoire

Character → fonction

charac → objet

```
1   const Character = function () {
2       this.hp = 100
3       this.getHP = function () {
4           return this.hp
5       }
6   }
7
8   const charac = new Character()
9   charac.getHP()  // 100
10
```

En mémoire

Character → fonction

charac → objet → hp = 100

objet → getHp = function () { …}

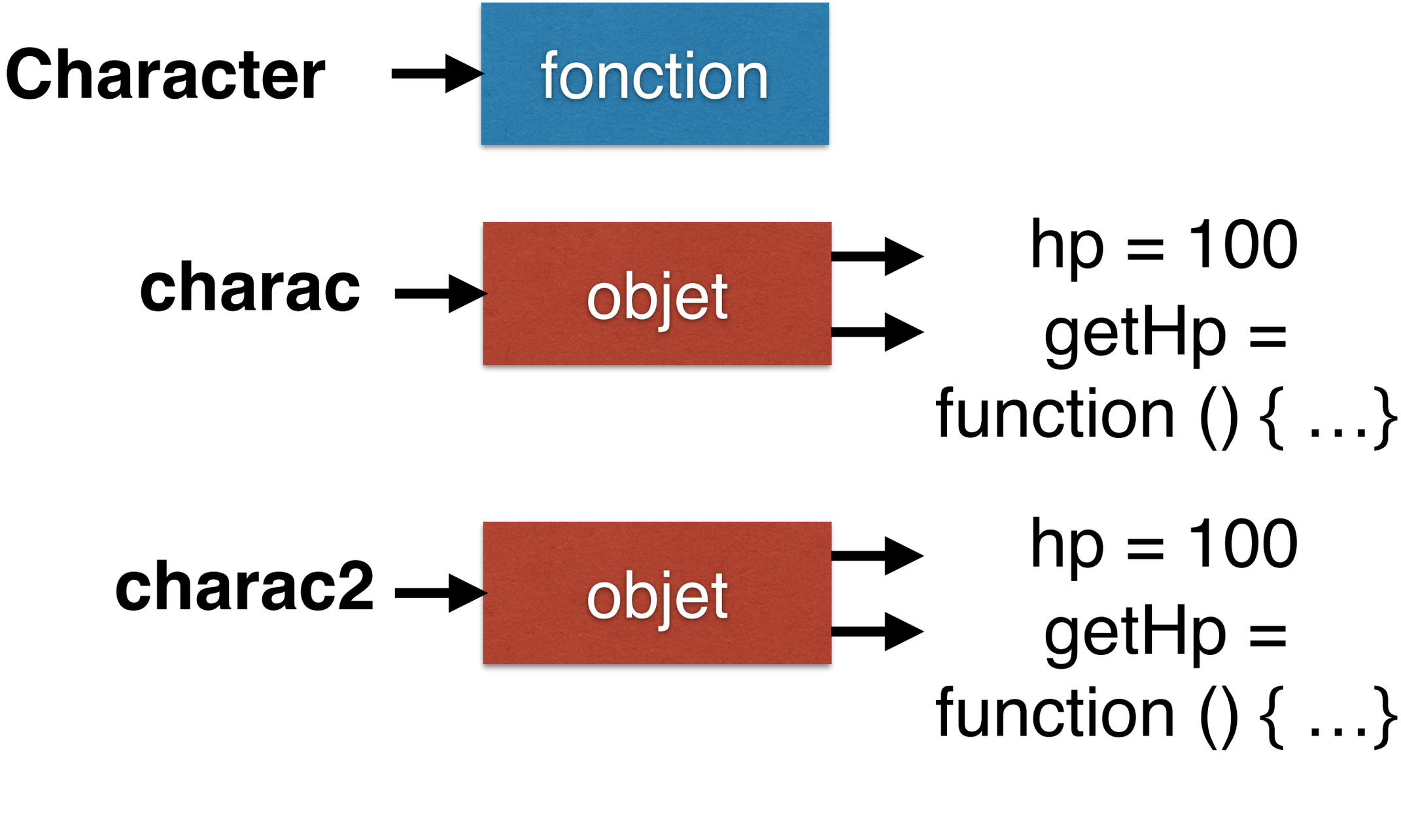
```
const Character = function () {
    this.hp = 100
    this.getHP = function () {
        return this.hp
    }
}

const charac = new Character()
charac.getHP()  // 100
```

Nouvelle instance

```
1   const Character = function () {
2       this.hp = 100
3       this.getHP = function () {
4           return this.hp
5       }
6   }
7
8   const charac = new Character()
9   charac.getHP()  // 100
10
11  const charac2 = new Character()
12
```

En mémoire


Character
fonction
charac
objet
hp = 100
getHp =
function () { …}
charac2
objet
hp = 100
getHp =
function () { …}

Duplication des propriétés

- Toutes les propriétés de **Character** sont dupliquées pour chaque instance
- C’est cool pour les attributs…
  - Permet de faire évoluer les instances différemment
- … ça l’est moins pour les méthodes
  - Gaspillage de ressources

Héritage de prototype

- Nous permet de définir une *structure* et un *comportement* commun à un ensemble d’objet
- Permet à chacun de ces objets de partager le même comportement…
- … tout en évoluant de façon différente

Héritage en JS

- 2 propriétés
  - *constructor*
  - *prototype*
- Complémentaires
- Automatiques

Propriété *constructor*

- Tout objet a une propriété *constructor*
  - type : fonction
  - celle utilisée avec **new** pour créer l'objet
- obj.constructor
  - créée automatiquement

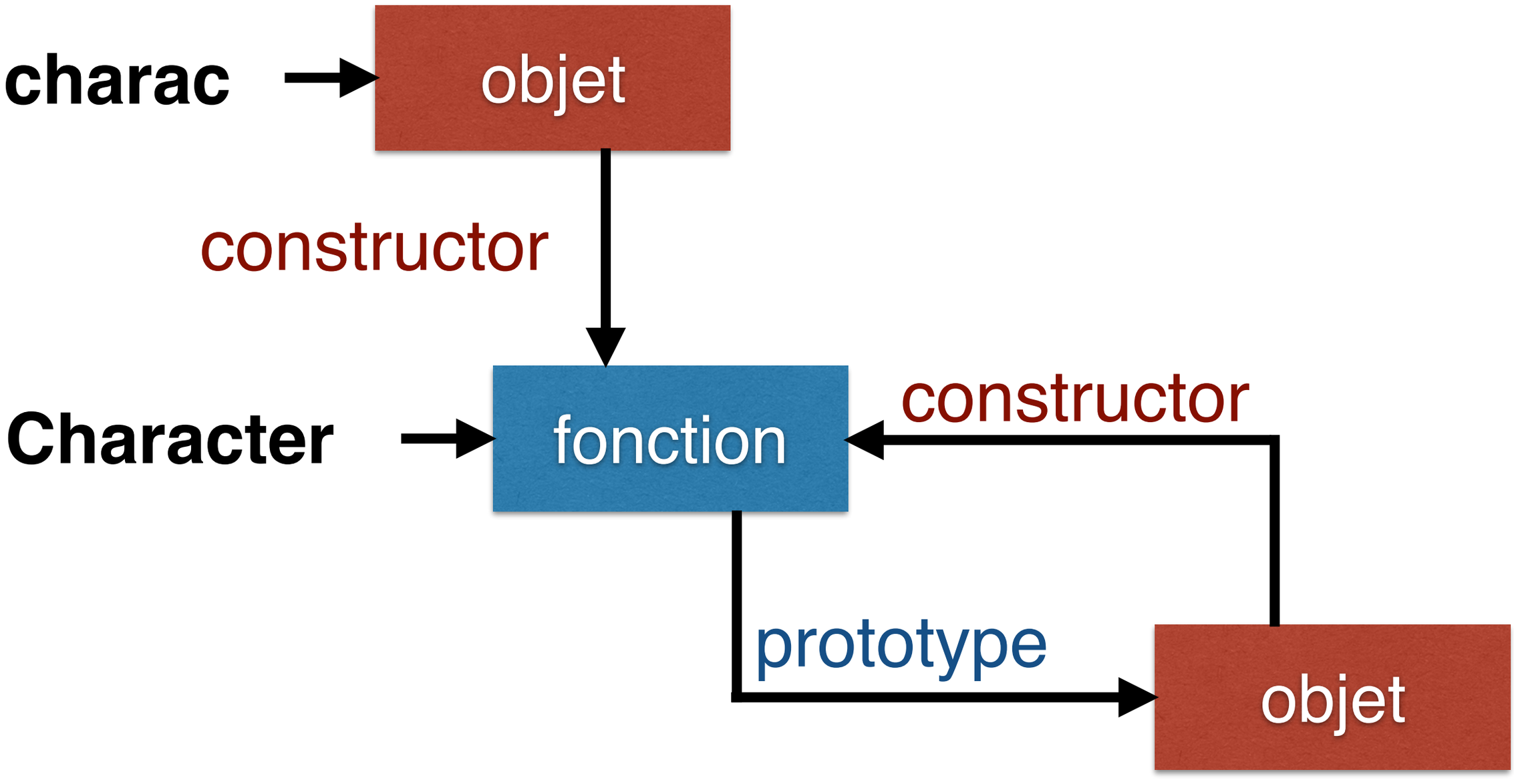
```
1   const charac = new Character()
2   charac.constructor  // Character
3
```

Propriété *prototype*

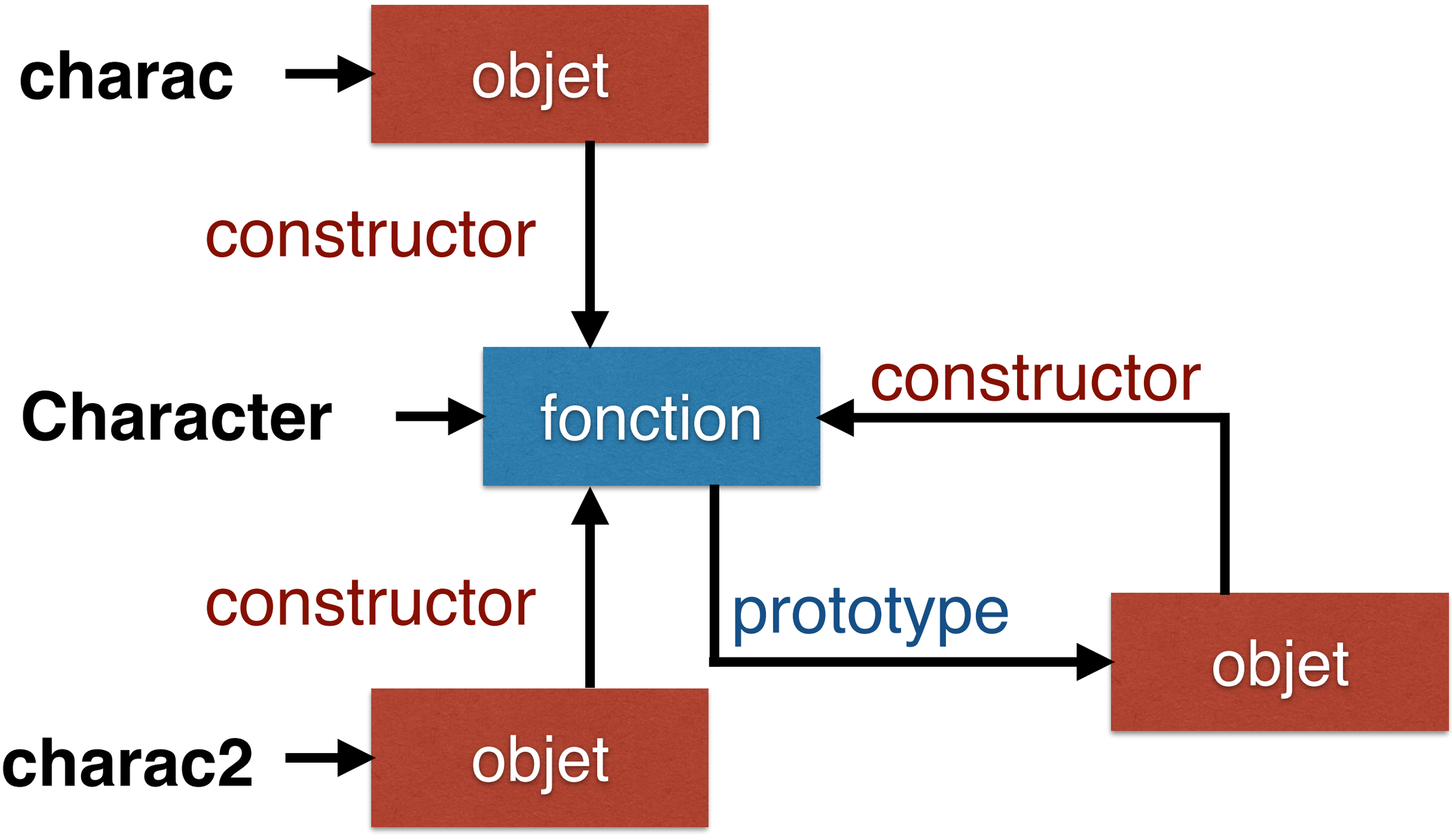
- Toute fonction a une propriété *prototype*
  - type : objet
  - valeur : *{ constructor: this }*
- MyObj.prototype
  - créée automatiquement

```
1  const charac = new Character()
2  charac.prototype  // { constructor: Character }
3
```

```
1   const Character = function () { ... }
2   const charac = new Character()
3
```



```
1   const Character = function () { ... }
2   const charac = new Character()
3   const charac2 = new Character()
4
```

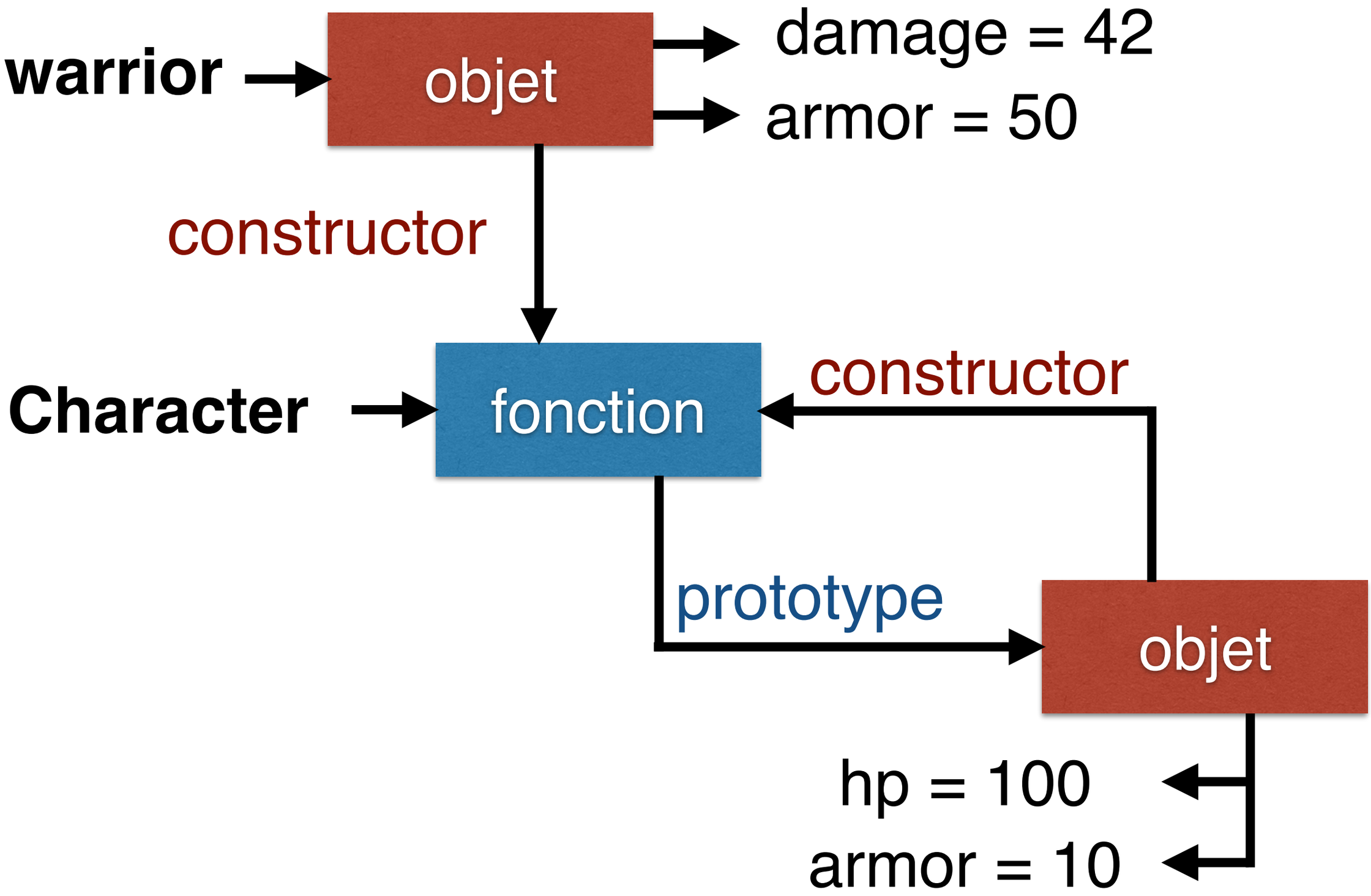


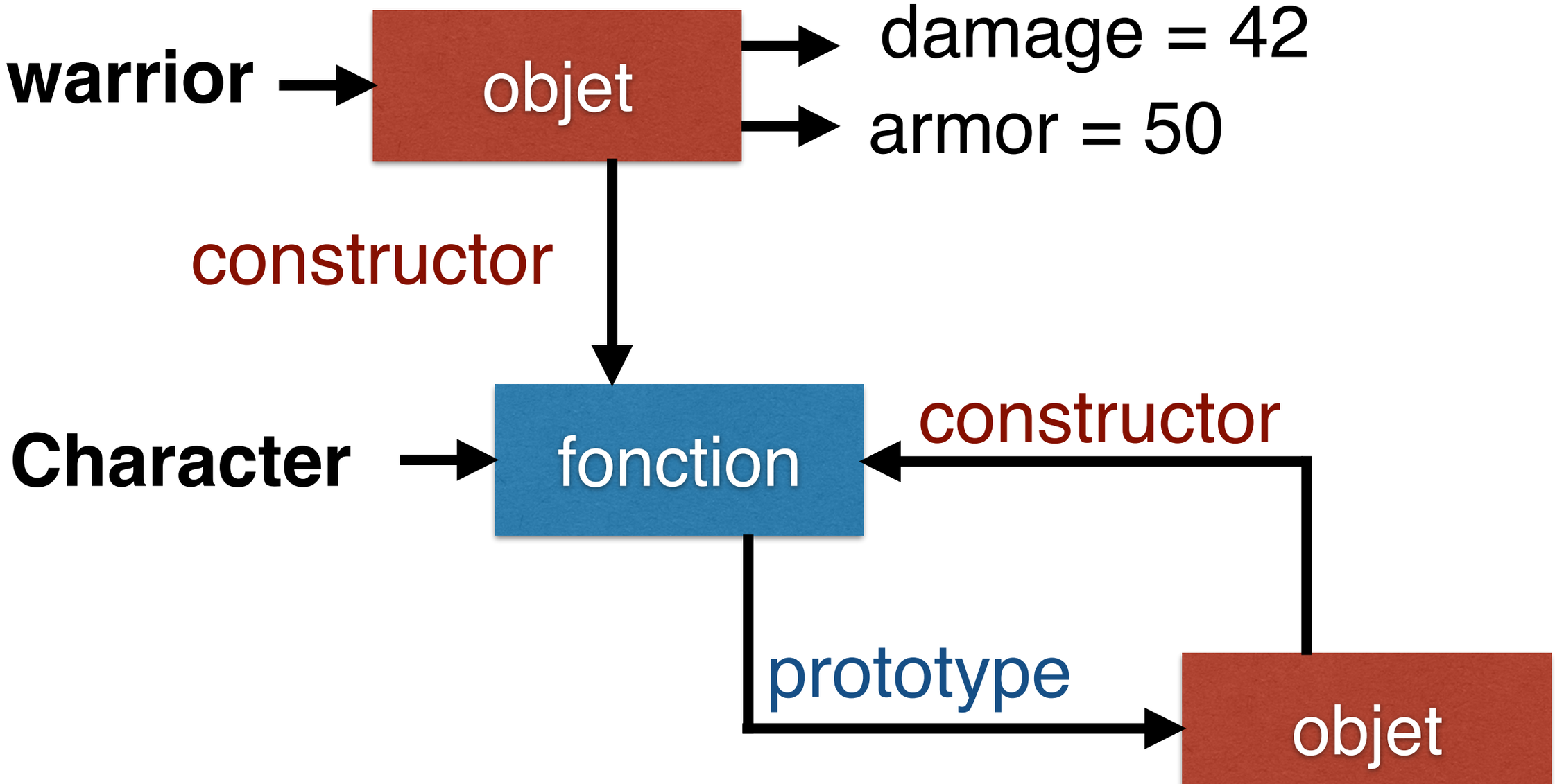
A quoi ça sert ?

- A définir un ensemble de propriétés partagées par un type d’objet
- Sorte de typage

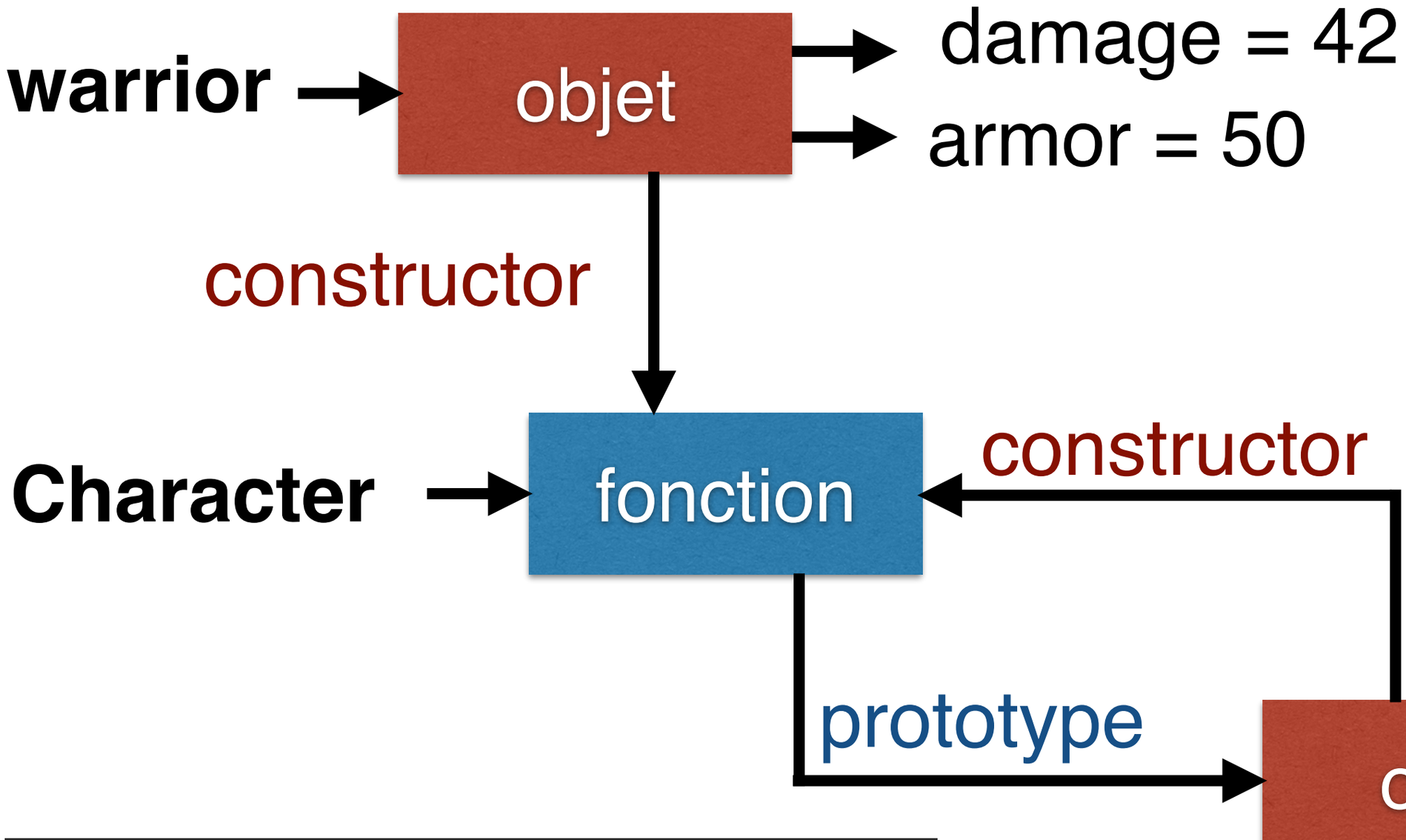
Example

```
1   const Character = function () {}
2   Character.prototype.hp = 100
3   Character.prototype.armor = 10
4
5   const warrior = new Character()
6   warrior.damage = 42
7   warrior.armor = 50
8
```


warrior
objet
damage = 42
armor = 50
constructor
Character
fonction
constructor
prototype
objet
hp = 100
armor = 10




```
1   warrior.damage·····//·?
2   warrior.hp·········//·?
3   warrior.armor·····//·?
4   warrior.mp·········//·?
5
```




```
1   warrior.damage·····//·42
2   warrior.hp·········//·100
3   warrior.armor·····//·50
4   warrior.mp·········//·undefined
5
```

Accès propriétés

- *warrior.damage*
  - accès à une propriété de l'objet
- *warrior.hp*
  - accès à une propriété du prototype
- *warrior.armor*
  - la propriété de l'objet masque le prototype
- *warrior.mp*
  - propriété non trouvée (ni objet ni prototype)

Tout est dynamique

- Possible de rajouter une propriété à un prototype d'un objet existant

```
1   const Character = function () {}
2   Character.prototype.hp = 100
3   Character.prototype.armor = 10
4
5   const warrior = new Character()
6   warrior.damage = 42
7   warrior.armor = 50
8
9   Character.prototype.mp = 50
10  warrior.mp          // 50
11
```

Tout est dynamique

- Possible de révéler une propriété du prototype en supprimant une de l'objet

```
1   const Character = function () {}
2   Character.prototype.hp = 100
3   Character.prototype.armor = 10
4
5   const warrior = new Character()
6   warrior.damage = 42
7   warrior.armor = 50
8
9   delete warrior.armor
10  warrior.armor          // 10
11
```

Jusque là…

- Tout objet a une propriété constructor
  - type : function
- Toute fonction a une propriété prototype
  - type : object
- Recherche propriété :
  - objet
  - puis *constructor.prototype*

Types existants

```
1   ({}).constructor····//·function·Object()·{…}
2   [].constructor·····//·function·Array()·{…}
3   "".constructor·····//·function·String()·{…}
4   (1).constructor····//·function·Number()·{…}
5   true.constructor···//·function·Boolean() {…}
6   /aaa/.constructor··//·function·RegExp()·{…}
7   f.constructor·····//·function·Function()·{…}
8
```

typeof

- Opérateur permettant de déterminer le type d’une valeur…

```
1  typeof 1            // "number"
2  typeof ""           // "string"
3  typeof true         // "boolean"
4
5  typeof Infinity     // "number"
6  typeof NaN          // "number"
7  typeof undefined    // "undefined"
8  typeof null         // "object"
9
```

typeof

- … mais peu utile avec les **Objects**

```
1   typeof warrior      // "object"
2   typeof [1, 2, 3, 4] // "object"
3   typeof {}           // "object"
4   typeof null         // "object"
5
```

instanceof

- Opérateur permettant de déterminer plus précisément le type d'un objet

```
1  warrior instanceof Character        // true
2  [1, 2, 3, 4] instanceof Character   // false
3  ({}) instanceof Character           // false
4  null instanceof Character           // false
5
```

Temps pour un peu d’exercice

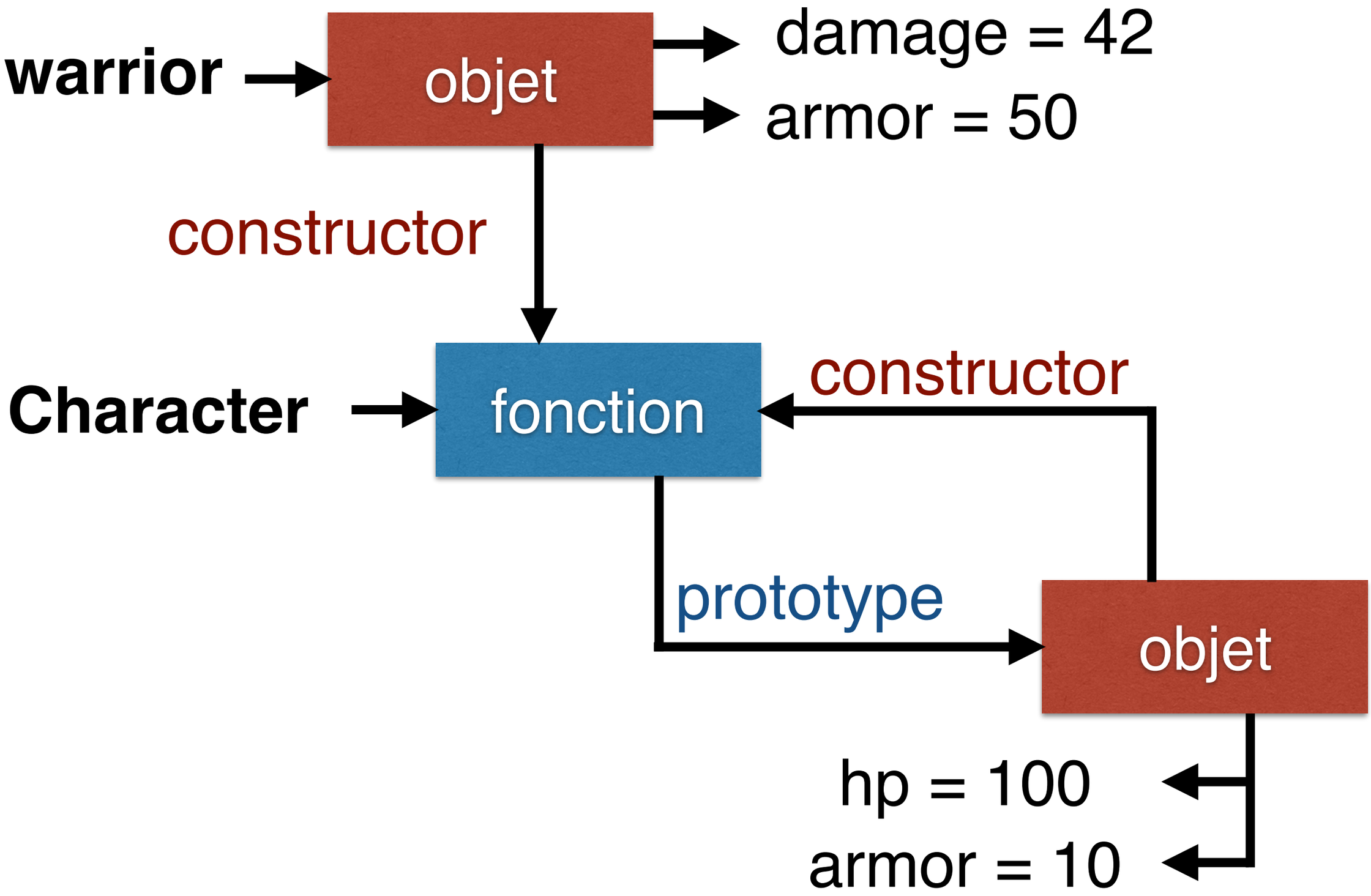
https://classroom.github.com/a/ZoKpjWsO
(lien dispo sur Arche)

Ça se complique

- Par défaut
  - *prototype* est créé automatiquement
    - *{ constructor: this }*
- Et si ce n’est pas le cas ?
  - Peut spécifier le prototype à la main
  - Création d’une chaine de prototypes

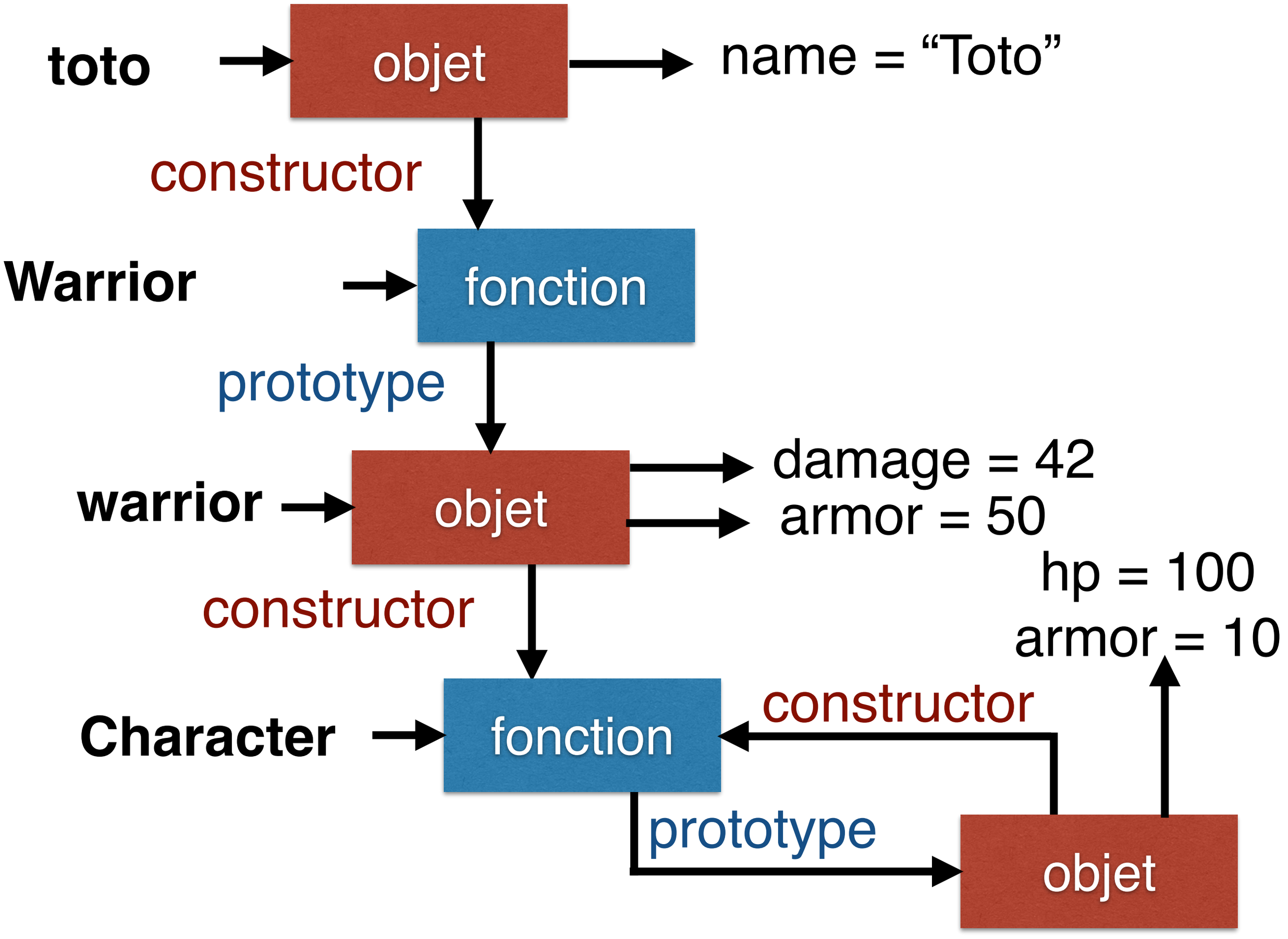
Example

```
1   const Character = function () {}
2   Character.prototype.hp = 100
3   Character.prototype.armor = 10
4
5   const warrior = new Character()
6   warrior.damage = 42
7   warrior.armor = 50
8
```


warrior
objet
damage = 42
armor = 50
constructor
Character
fonction
constructor
prototype
objet
hp = 100
armor = 10

Example

```
1   const Warrior = function () {}
2   Warrior.prototype = warrior
3
4   const toto = new Warrior()
5   toto.name = "Toto"
6
```


toto
objet
name = “Toto”
constructor
Warrior
fonction
prototype
warrior
objet
damage = 42
armor = 50
constructor
hp = 100
armor = 10
Character
fonction
constructor
prototype
objet

toto → objet → name = “Toto”

objet → constructor → **Warrior** → fonction → prototype → objet

```
1   toto.name   // ?
2   toto.damage // ?
3   toto.hp     // ?
4   toto.mp     // ?
5
```

warrior → objet → damage = 42, armor = 50

objet → constructor → **Character** → fonction → prototype → objet

objet → hp = 100, armor = 10

fonction → constructor → objet

Accès propriétés

- *toto.name = “Toto”*
  - accès à une propriété de l’objet
- *toto.damage = 42*
  - accès à une propriété du prototype
- *toto.hp = 100*
  - accès à une propriété du prototype du constructeur du prototype
- On remonte la chaine…

instanceof

- Fonctionne aussi avec les chaînes de prototypes

```
1   toto instanceof Warrior     // true
2   toto instanceof Character   // true
3
```

Object

- Tout est objet
  - « Tout les types dérivent du type Object »
    - Tous les types ont pour prototype le type **Object**
- *toto instanceof Object*
- toutes les propriétés d'**Object** se retrouvent dans tous les objets
  - *toString()*

Tout est Object ?

- Pas les Immuables

```
1   (1) instanceof Object   // false
2   "" instanceof Object    // false
3   true instanceof Object  // false
4
```

Tout est Object ?

- MAIS

```
1  new Number() instanceof Object     // true
2  new String() instanceof Object     // true
3  new Boolean() instanceof Object    // true
4
```

Specials

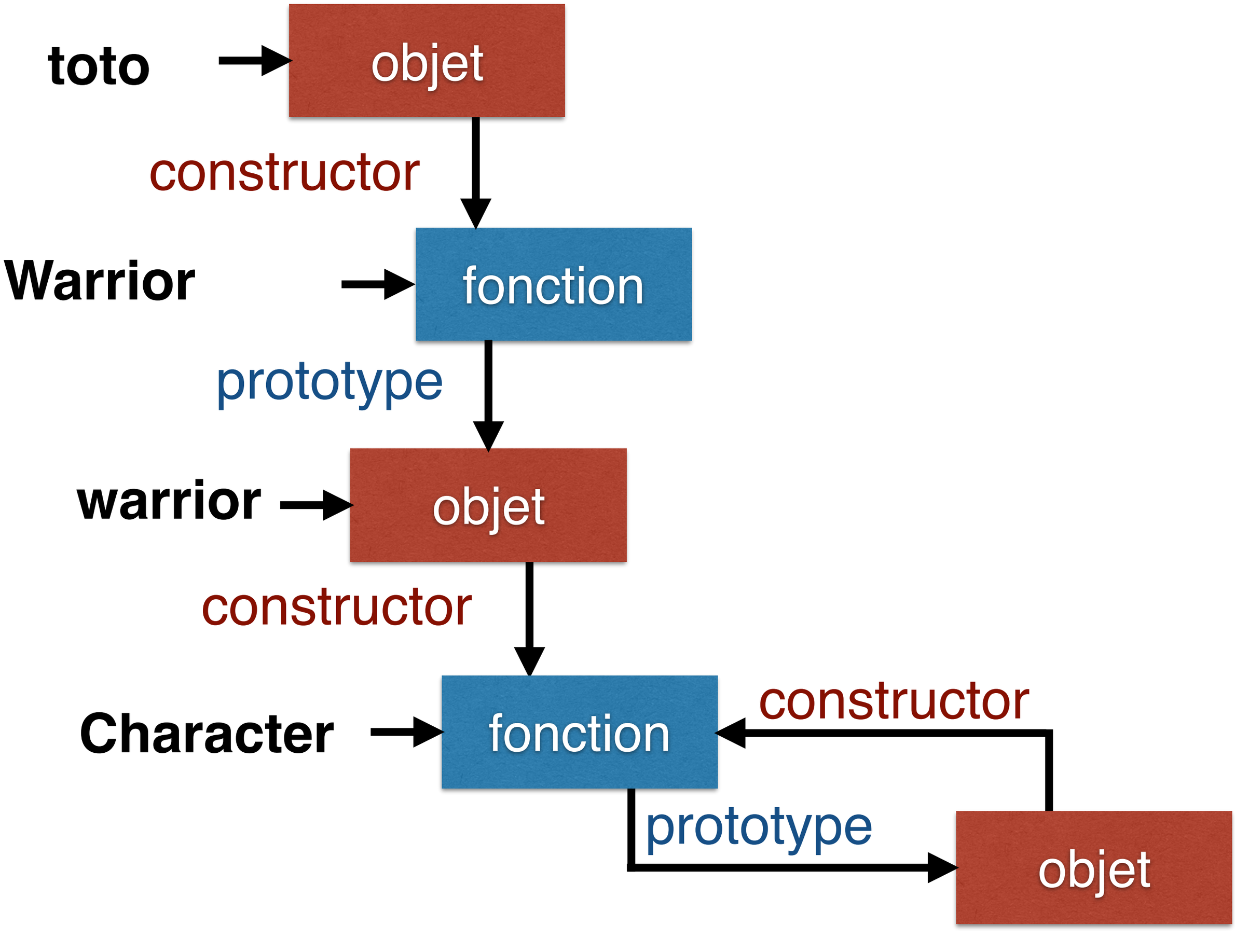
```
1   Infinity instanceof Object   // false
2   NaN instanceof Object        // false
3   undefined instanceof Object  // false
4   null instanceof Object       // false
5
```

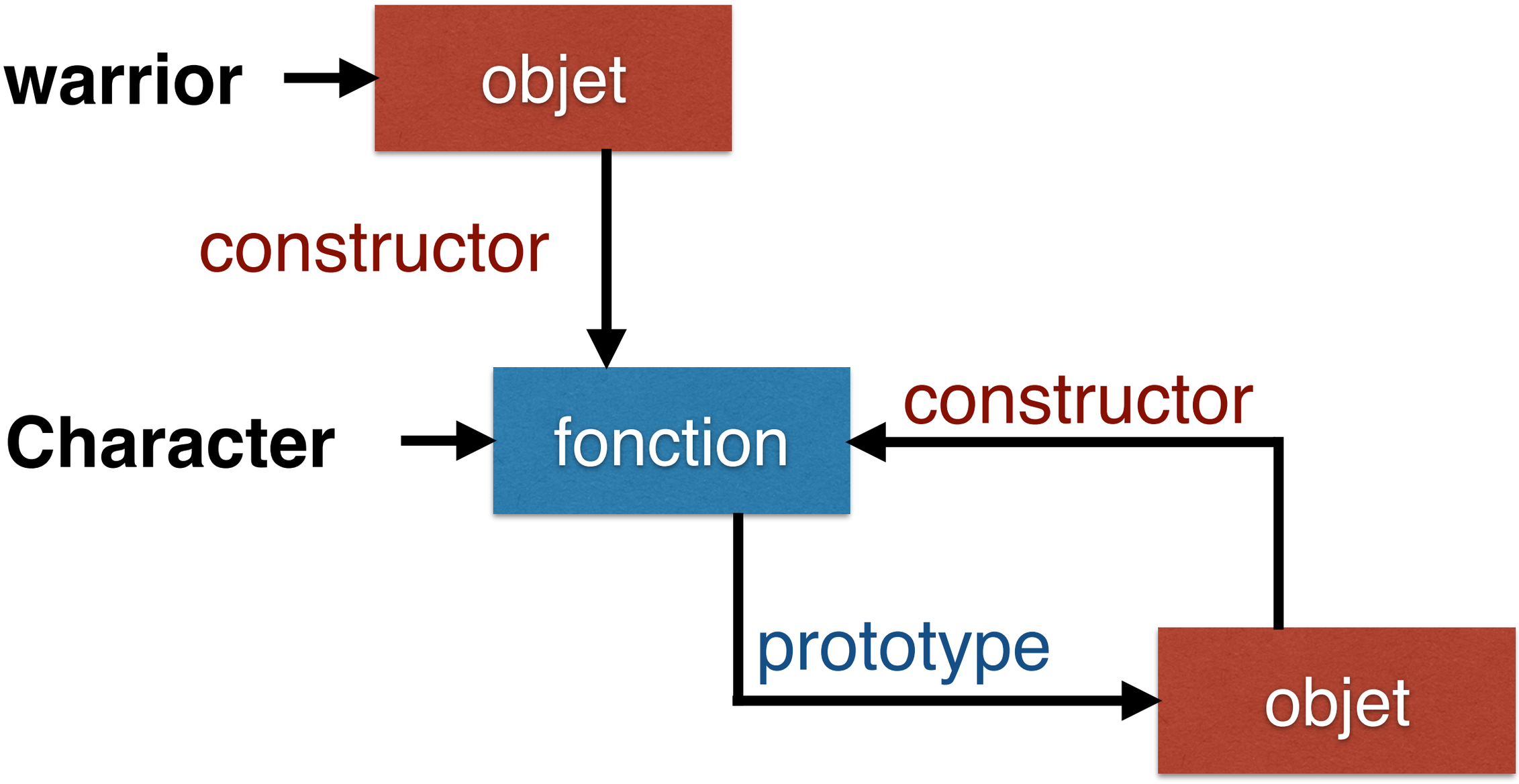
Vous reprendrez bien un peu de tests unitaires ?

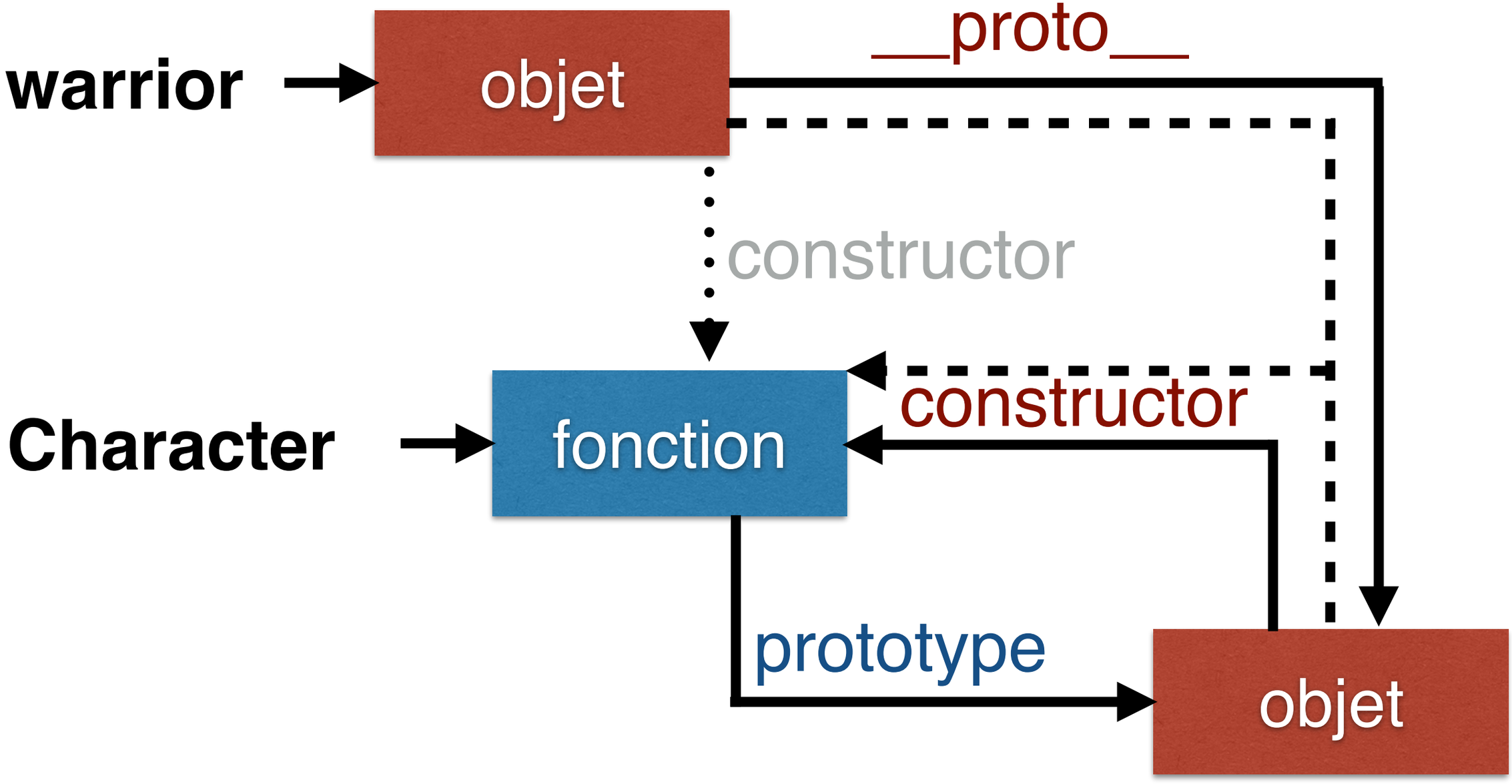
Récupérez la partie 2 à partir du repo distant *classroom*

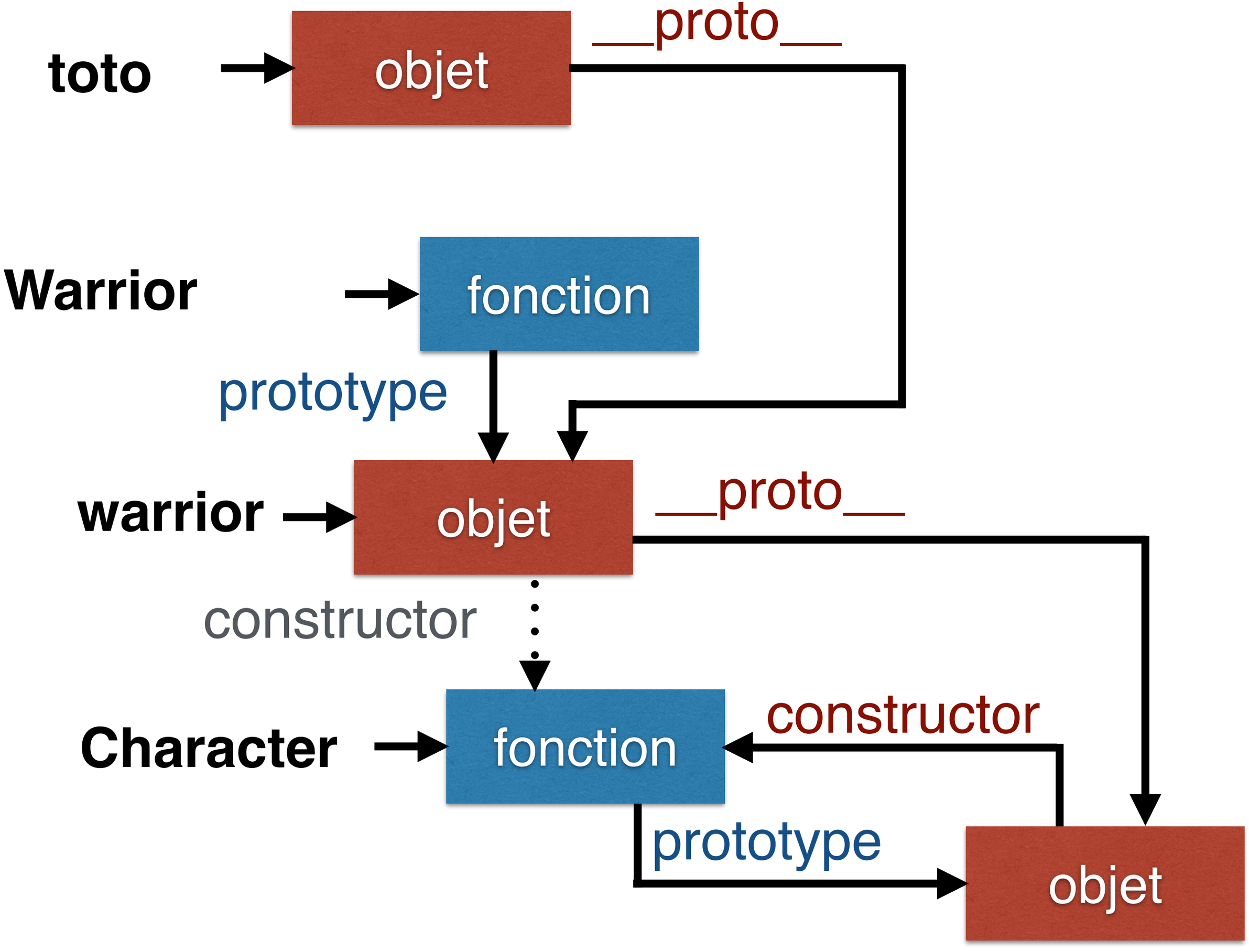
Sauf qu’en fait…

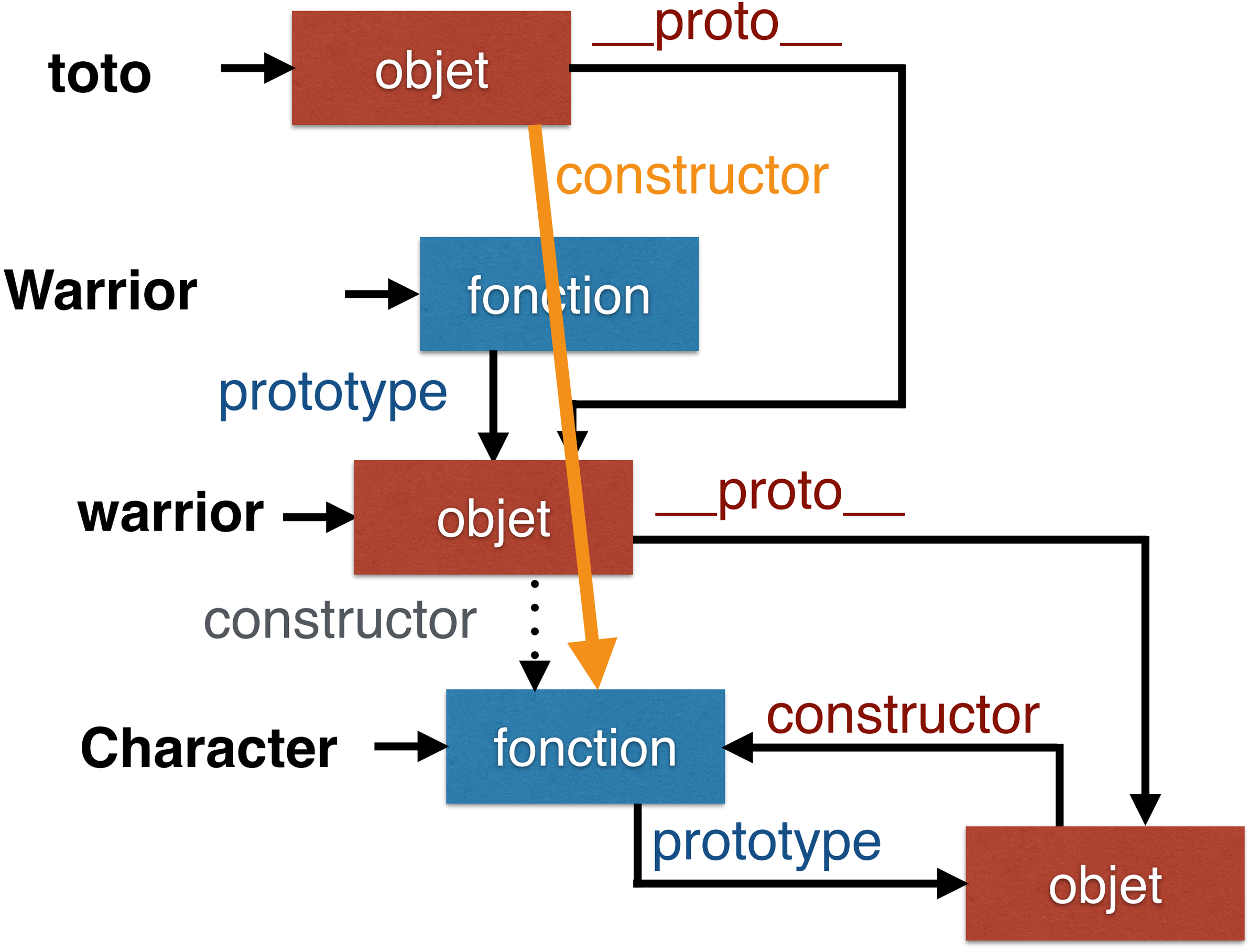
```
1   Character = function() {};
2   const warrior = new Character();
3
4   Warrior = function() {};
5   Warrior.prototype = warrior;
6
7   toto = new Warrior();
8
9   toto.constructor  // ?
10
```


toto
objet
constructor
Warrior
fonction
prototype
warrior
objet
constructor
Character
fonction
constructor
prototype
objet


warrior
objet
constructor
Character
fonction
constructor
prototype
objet


warrior
objet
__proto__
constructor
Character
fonction
constructor
prototype
objet


toto
objet
__proto__
Warrior
fonction
prototype
warrior
objet
__proto__
constructor
Character
fonction
constructor
prototype
objet


toto
objet
__proto__
constructor
Warrior
fonction
prototype
warrior
objet
__proto__
constructor
Character
fonction
constructor
prototype
objet

instanceof

- Ne se sert pas de “constructor”
- Parcours la chaine de prototypes
- À chaque étape, teste :
  - *obj.__proto__ === Class.prototype*

```
1   toto instanceof Warrior     // true
2   toto instanceof Character   // true
3   toto instanceof Object      // true
4
```

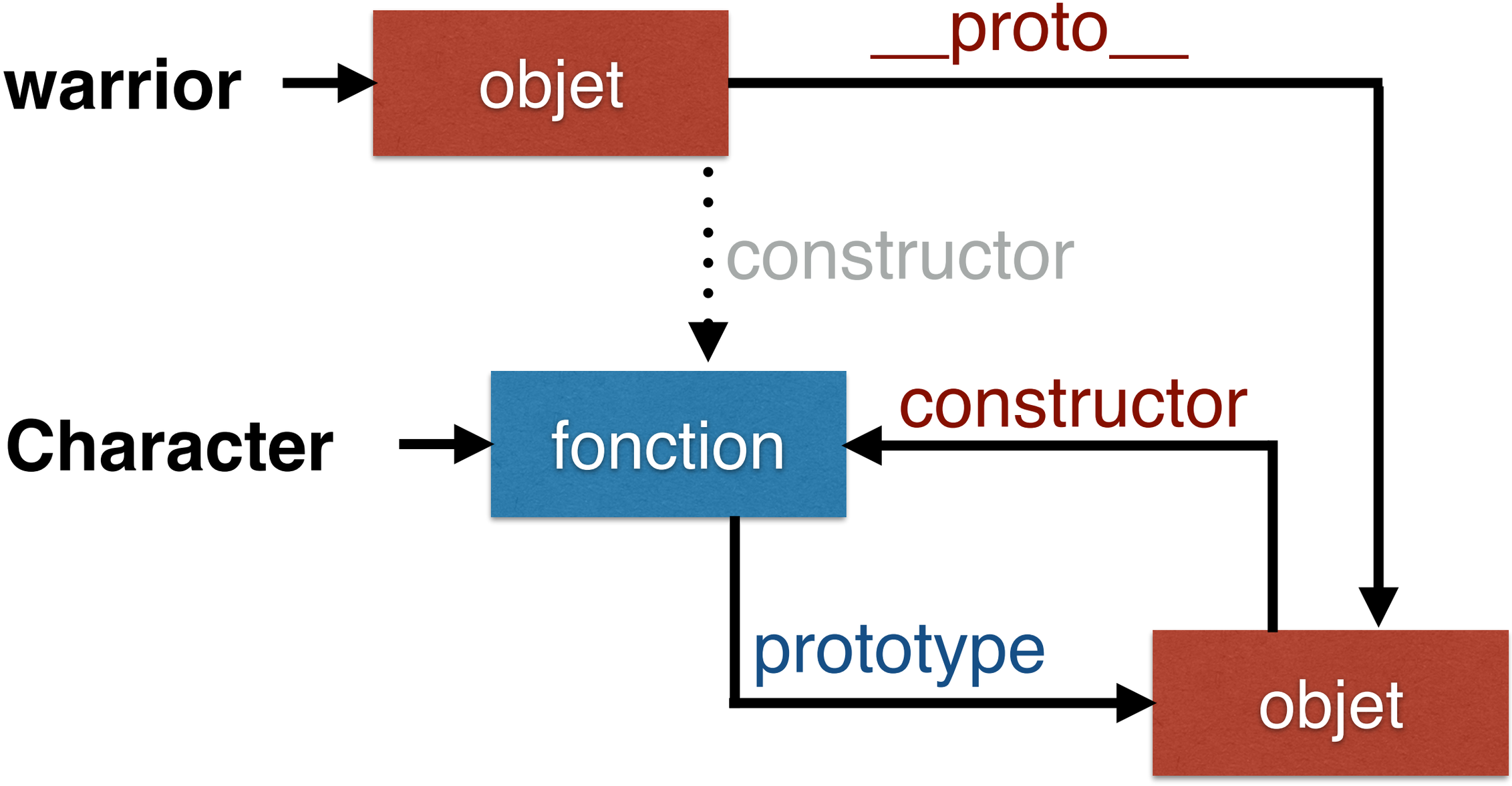
__proto__

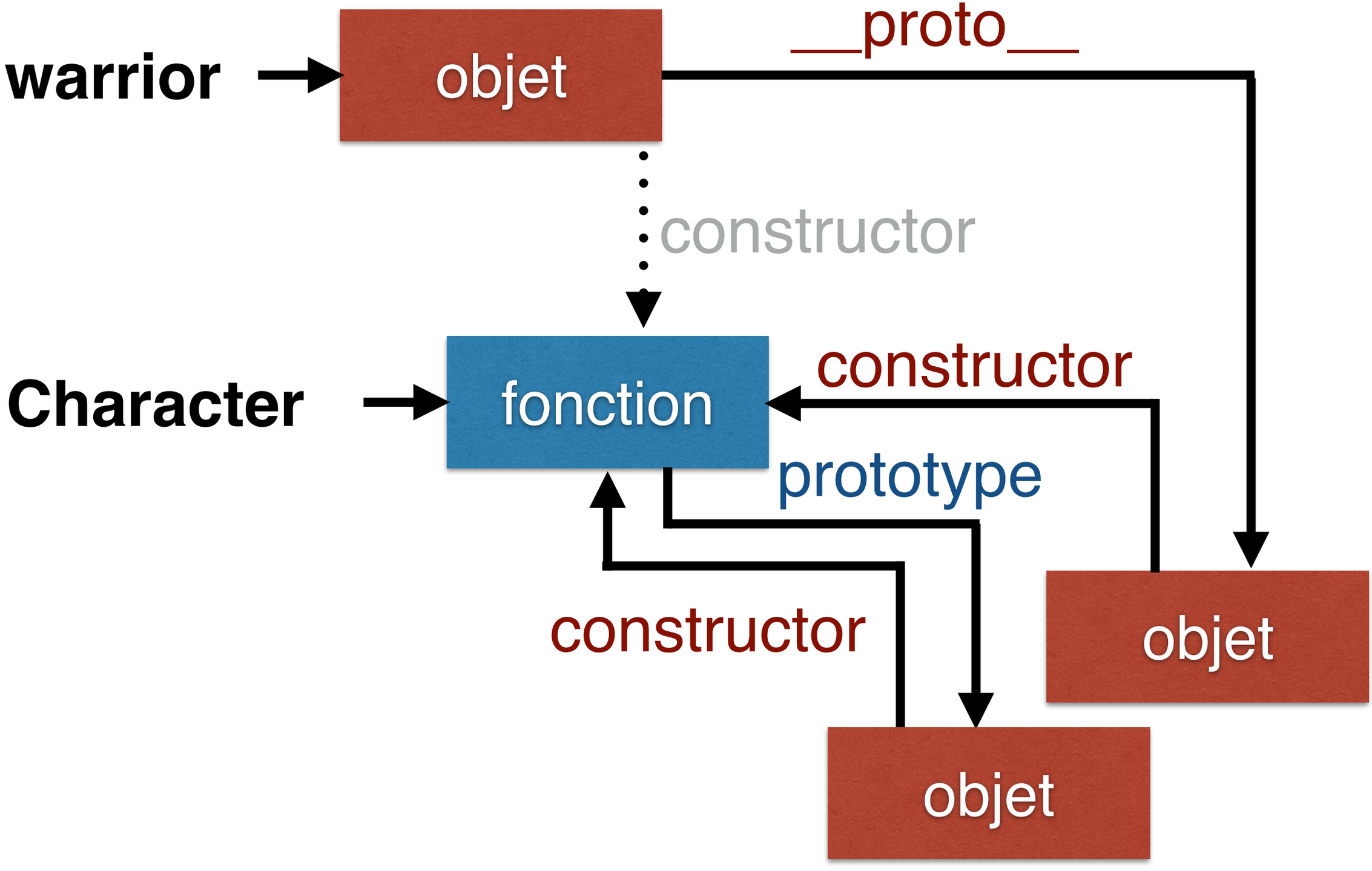
- Pourquoi ne pas apprendre directement le modèle avec __proto__ ?
  - __proto__ n'est PAS STANDARD
    - Pas implémenté dans IE
    - Implémentations différentes Mozilla/Webkit
- Fonctions spécifiées dans ECMASCript 6
  - Sous le nom **[[Prototype]]**
- MAIS : pas obligatoire !

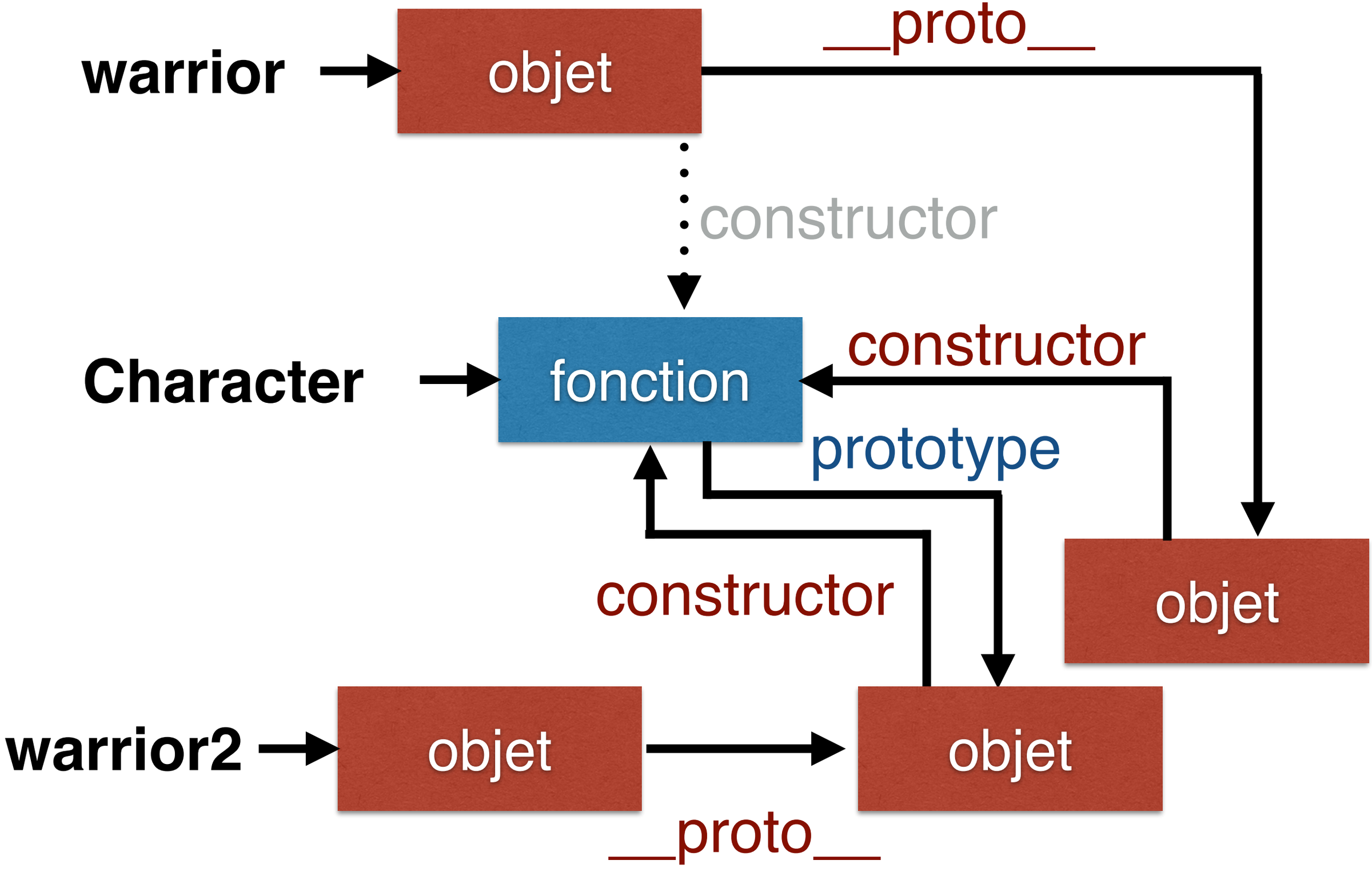
Tout est dynamique

- Possible de redéfinir le prototype d'une fonction à tout moment

```
1   const Character = function() {}
2   Character.prototype.hp = 100
3   Character.prototype.armor = 10
4
5   warrior = new Character()
6
7   Character.prototype = {
8       hp: 150,
9       shout: function () {
10          return "FUS RO DAH !"
11      }
12  }
13
14  warrior2 = new Character()
15
```


warrior
objet
__proto__
constructor
Character
fonction
constructor
prototype
objet


warrior
objet
__proto__
constructor
Character
fonction
constructor
prototype
constructor
objet
objet


warrior
objet
__proto__
constructor
Character
fonction
constructor
prototype
constructor
objet
warrior2
objet
objet
__proto__

```
const Character = function() {}
Character.prototype.hp = 100
Character.prototype.armor = 10

warrior = new Character()

Character.prototype = {
    hp: 150,
    shout: function () {
        return "FUS RO DAH !"
    }
}

warrior2 = new Character()

warrior.hp ..... // ?
warrior.armor ..... // ?
warrior.shout() ..... // ?

warrior2.hp ..... // ?
warrior2.armor ..... // ?
warrior2.shout() ..... // ?

```

Jamais 2 sans 3

Récupérez la partie 3 à partir du repo distant *classroom*