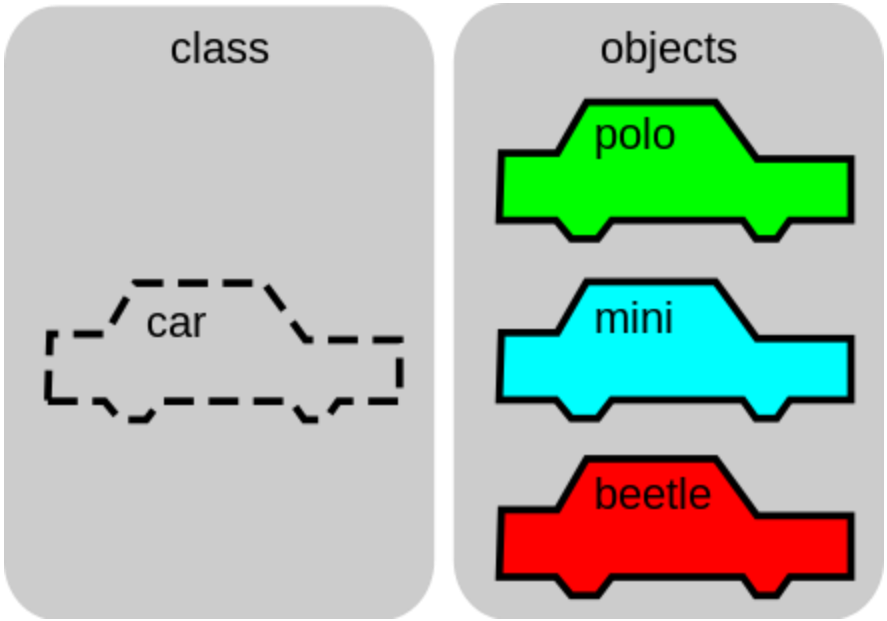
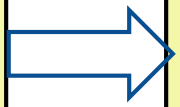
Objects



Constructing Object

- Constructor in JS is a function that is used to populate the fields of an existing object
- Naming convention : start the name of constructor with capital letter
- Constructor is passed to the new operator which constructs new object and calls constructor to populate/initialize the object.
- Every object has constructor property set to the constructor of the object.

```
function createPoint(x,y){
var pt = {};
pt.x= x;
Pt.y = y;
return pt;
}
var a = createPoint(2,3);
Var b = createPoint(4,5);
```



```
var Point = function(x, y){
   this.x = x;
  this.y =y;
}
var a = new Point(2, 3);
var b = new Point(3,4)

//where does this come from?
a.constructor === Point;
```

Where does this property come from?

- **`a.constructor === Point`**; **`//Both are function type`**
- **There are 5 different types that can contain values:**
  - string
  - number
  - boolean
  - object (includes null, Array, Date, Object)
  - Function
- **There are 2 datatype without values:**
  - null
  - undefined

Constructing Object

```
var a = new Object();
a.x = 2;
a.y = 3;
```

```
var a = new Object({x:2, y:3});
```

```
var b = Object.create(a);
```

‘this’ represents context object

- When used in constructor, ***this*** is bound to newly created object

```
var a = new Point(2,3); //this in Point=a
```

- When function is bound to an object, then ***this*** points to the binding object.

```
var a.foo = function(){return this.x};//this=a
```

Inheritance

- Each function object comes pre-equipped field called prototype
- The prototype field is initially an empty object
- When a field f is looked up in the object, the object's own fields are searched first.
- If object does not have that field, then object's constructor's prototype is searched to locate f.
- Thus any field added to the prototype field of the constructor is accessible to all objects built with that constructor.

Prototypal Inheritance: prototype

```
function Point(x,y){
this.x = x;
this.y = y;
}
var pt = new Point(2,3);
var pt2= new Point(4,5);
pt.constructor == Point;
pt.z == undefined;
//adding a field to prototype reflects in all inherited objects
Point.prototype.z = 10;
pt.z == 10;
pt2.z == 10;
Pt.hasOwnProperty(”z”) == false;

Point.prototype.hasOwnProperty(”z”) == true;
```

Prototypal Inheritance: __proto__

- prototype property:
  What can be inherited from me.

- __proto__ inner property :
  What have I inherited?

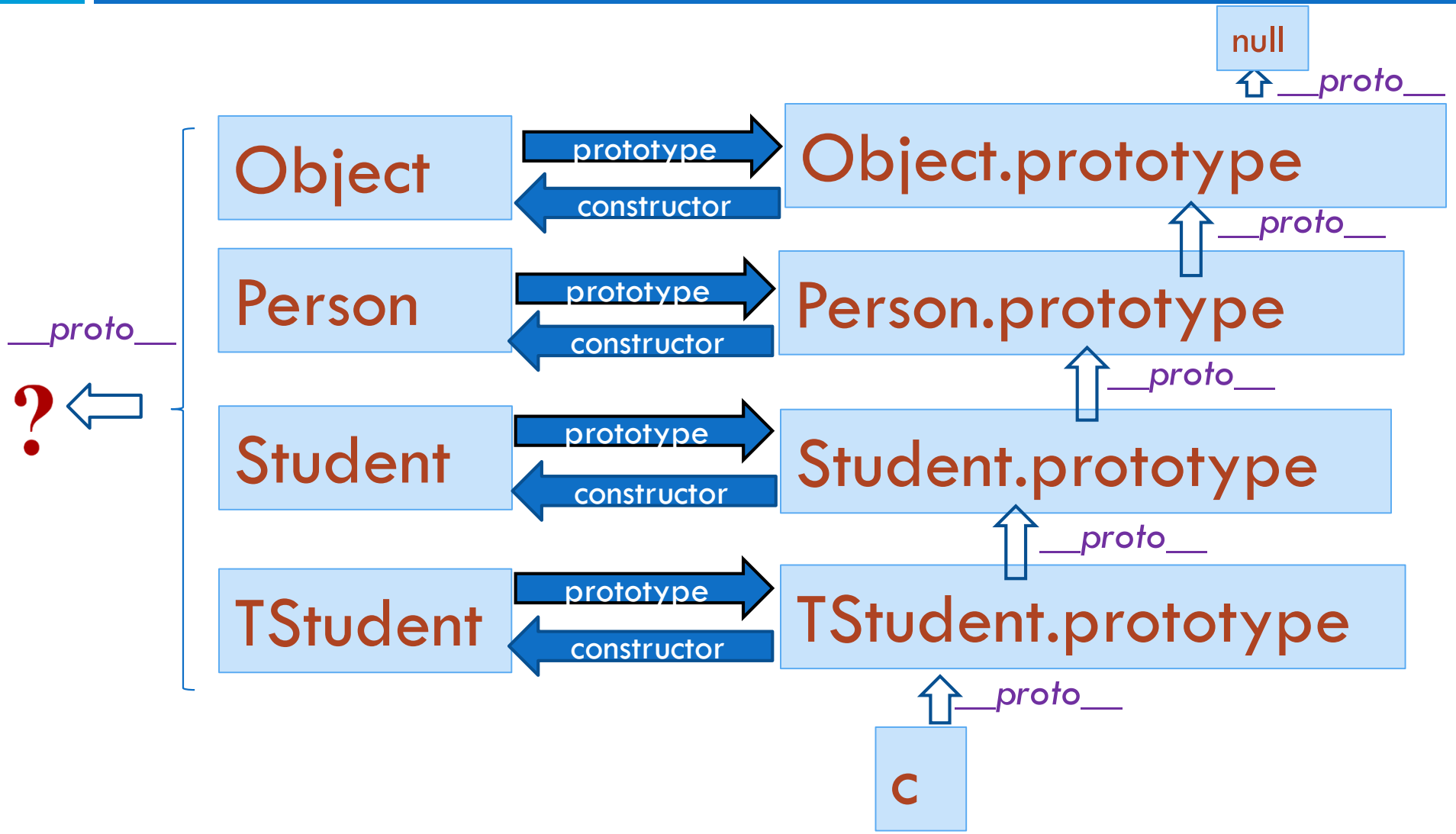
```
pt.__proto__ == pt.constructor.prototype; //== Point.prototype
pt.__proto__.__proto__ == Object.prototype;

pt.__proto__._proto__.__proto__ = null;
```

Prototypal Inheritance: __proto__

```
var Person =function(name,id) {
    this.name = name; this.id = id;
}
var Student = function(name, id, program){
     Person.call(this, name, id); this.program = program;
}
var TStudent = function(name, id, program, subject){
     Student.call(this, name, id, program); this.subject = subject;
}
//make the prototype chain
Student.prototype.__proto__ = Person.prototype;
Tstudent.prototype.__proto__ = Student.prototype;
```

```
var a = new Person("Foo",1);
var b = new Student("Bar",2, "ECE");
var c = new TStudent("Baz",3, "Maths", "ITW2");
Person.prototype.prop1 = "prop1 in person-prototype";
a.prop1 == b.prop1; b.prop1 == c.prop1;
c.prop1 == Person.prototype.prop1;
```

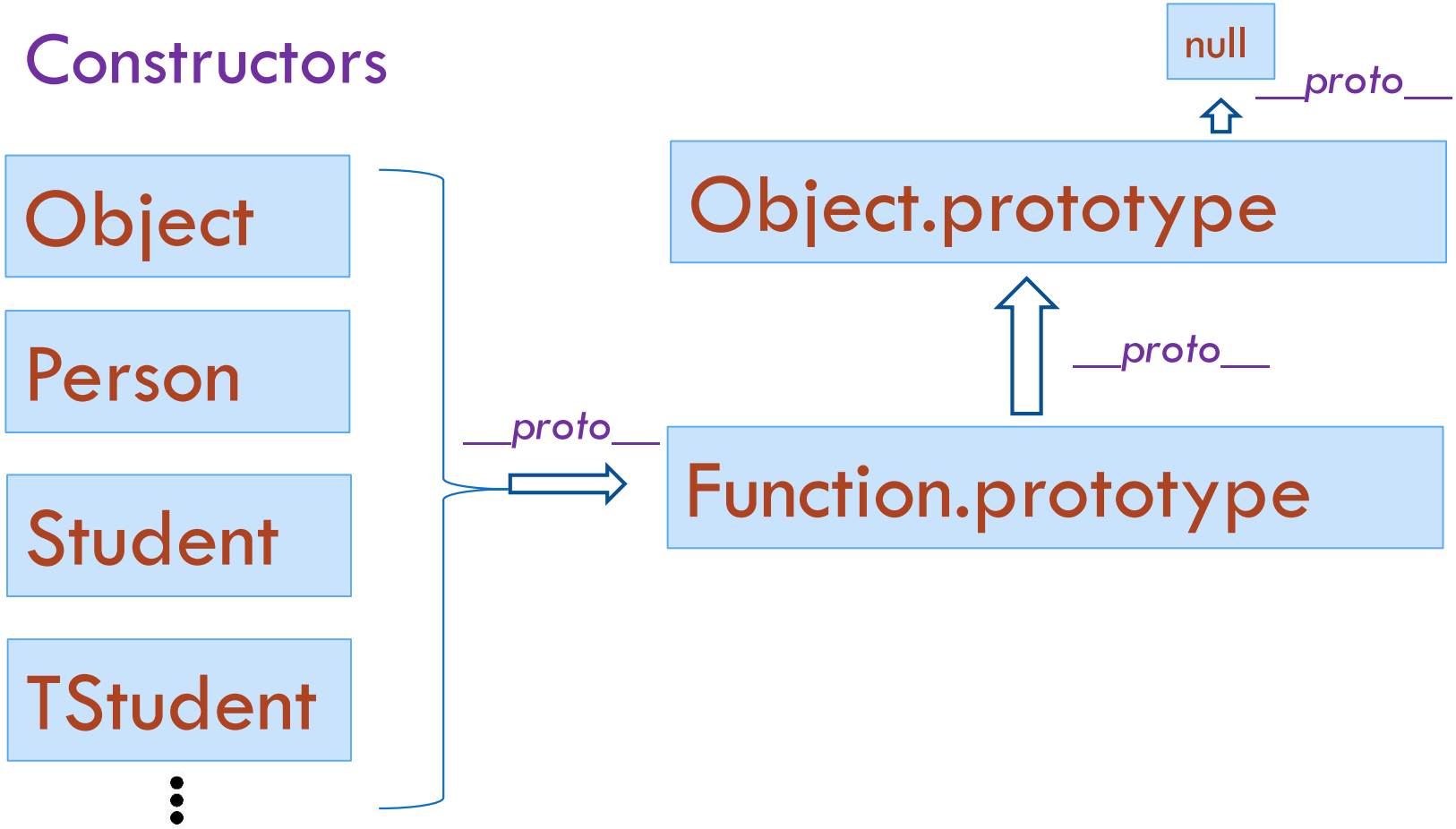
Protypal inheritance chain



Prototypal Inheritance chain


```
c.__proto__ == TStudent.prototype;
TStudent.prototype.__proto__ == Student.prototype;
Student.prototype.__proto__ == Person.prototype;
Person.prototype.__proto__ == Object.Prototype;
Object.prototype.__proto__ == null;
```

Protypal inheritance chain



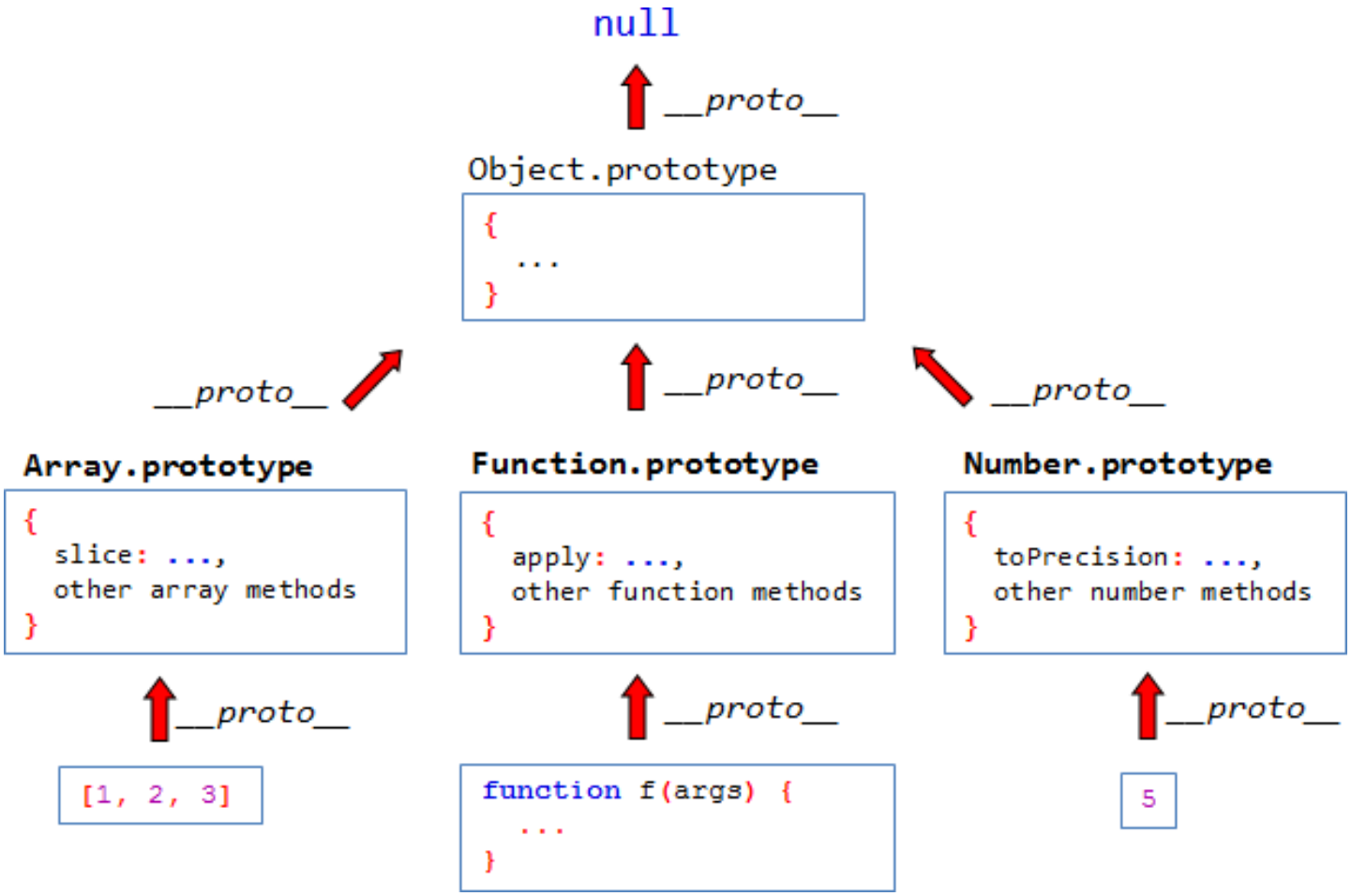
You can add more to the constructor list –

Array, Number, String, Map, Set, Function and user defined functions etc.

Prototypal Inheritance chain


```
//all constructors and functions derive from Function.prototype
Object.__proto__ == Function.prototype;
Array.__proto__ == Function.prototype;
String.__proto__ == Function.prototype;
Number.__proto__ == Function.prototype;
Set.__proto__ == Function.prototype;
Map.__proto__ == Function.prototype;
Function.__proto__ == Function.prototype;

//Function.prototype object derive from Object.prototype
Function.prototype.__proto__ == Object.prototype;
Object.prototype.__proto__ = null;
```

Inheritance in generic JS objects



Prototypal Inheritance chain


```
var ar = [”foo”, 2, 3];

ar.__proto__ == Array.prototype;
Array.prototype.__proto__ == Object.prototype;
Object.prototype.__proto__== null;

ar.__proto__ == Array.prototype;
ar.__proto__.__proto__ == Object.prototype;
ar.__proto__.__proto__.__proto__== null;
```