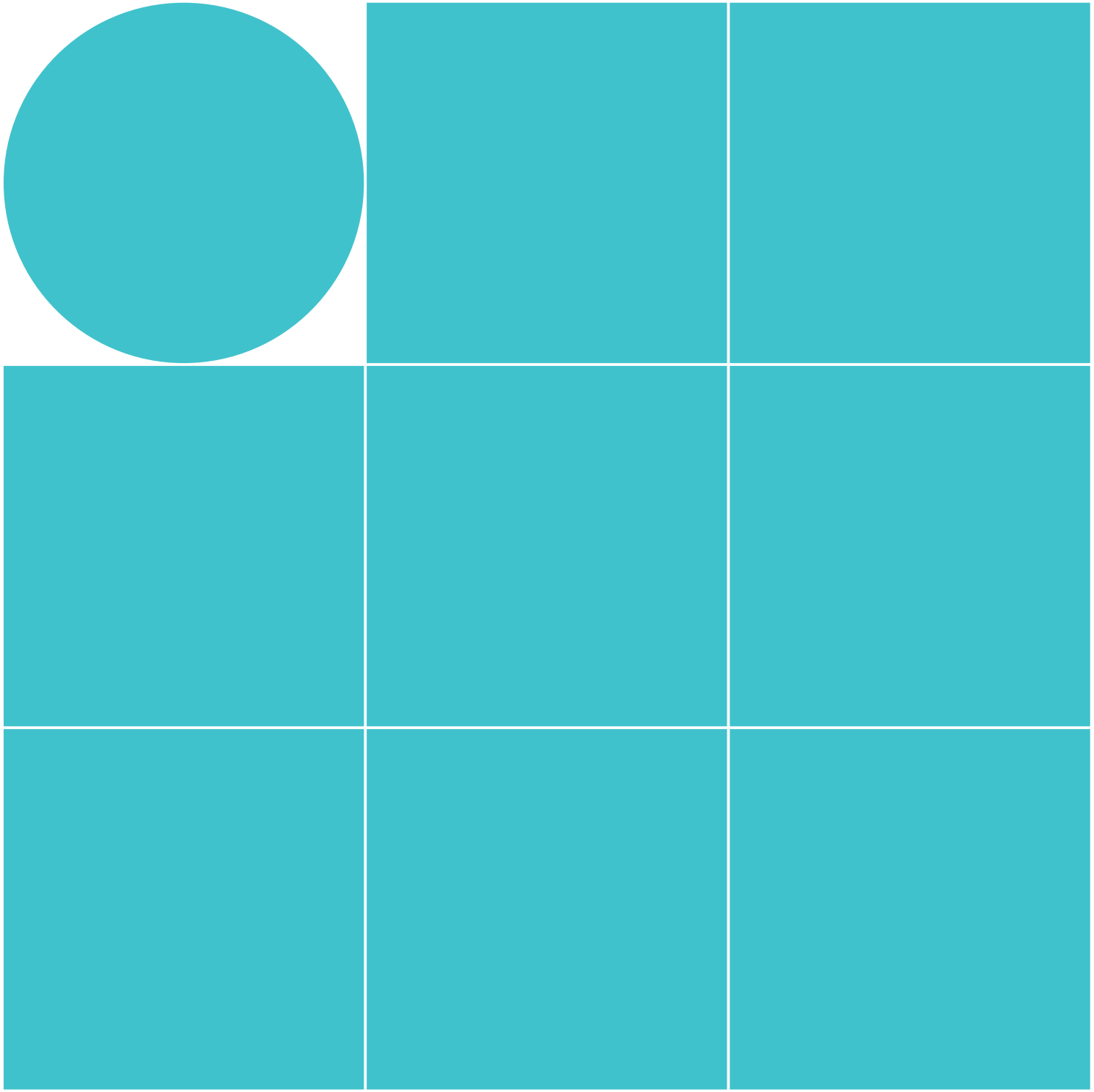
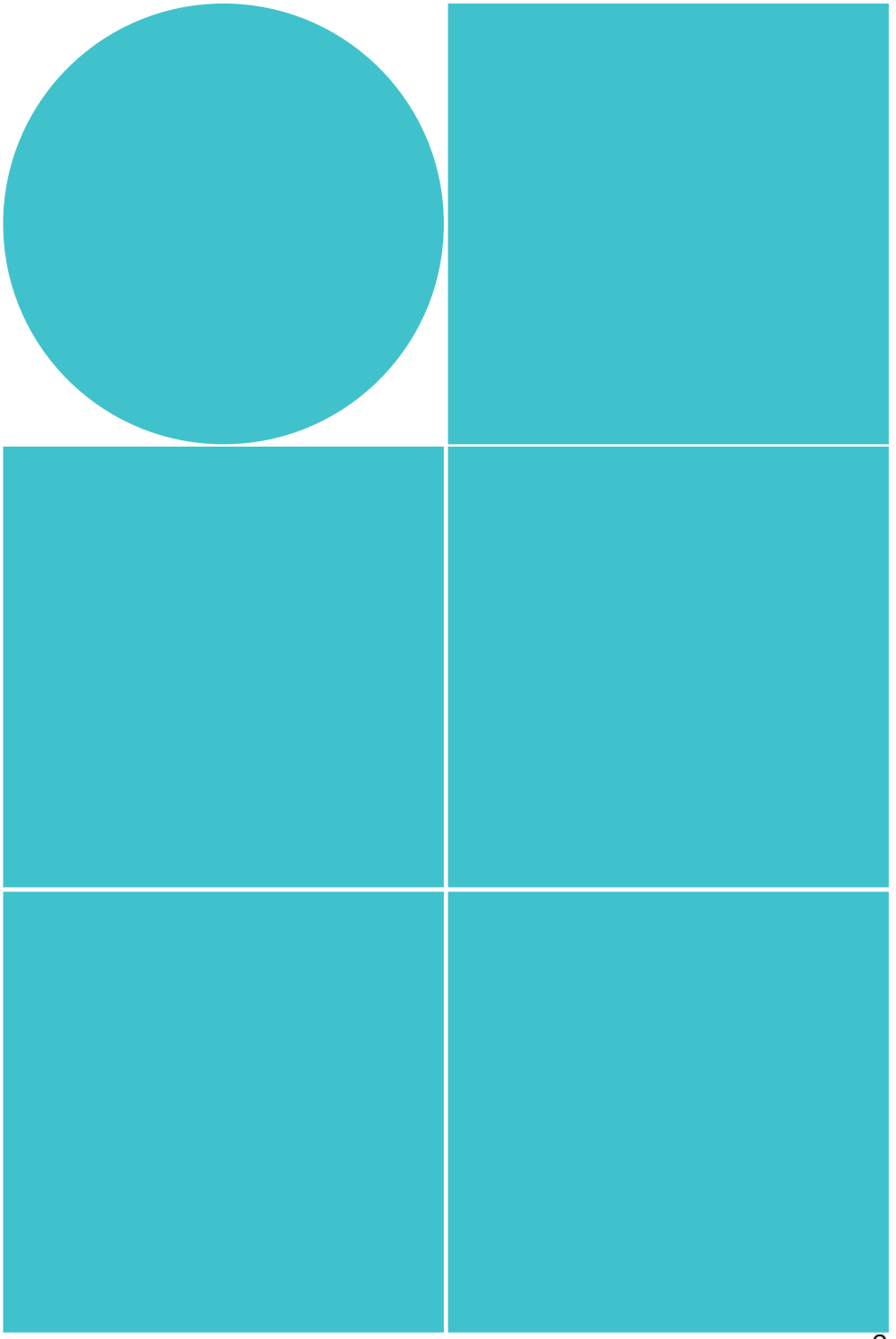


ANGULARJS

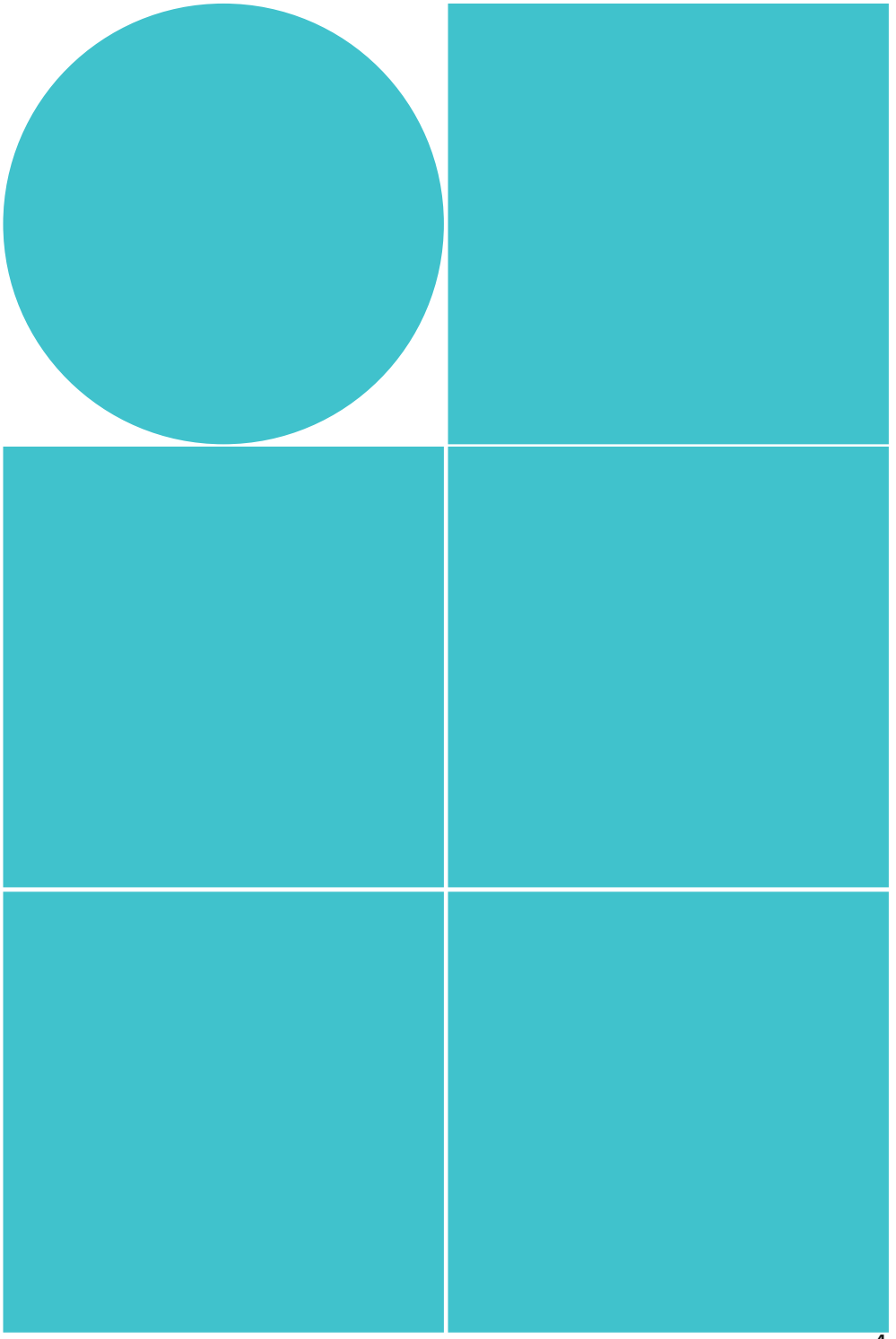
by Google

ekino.

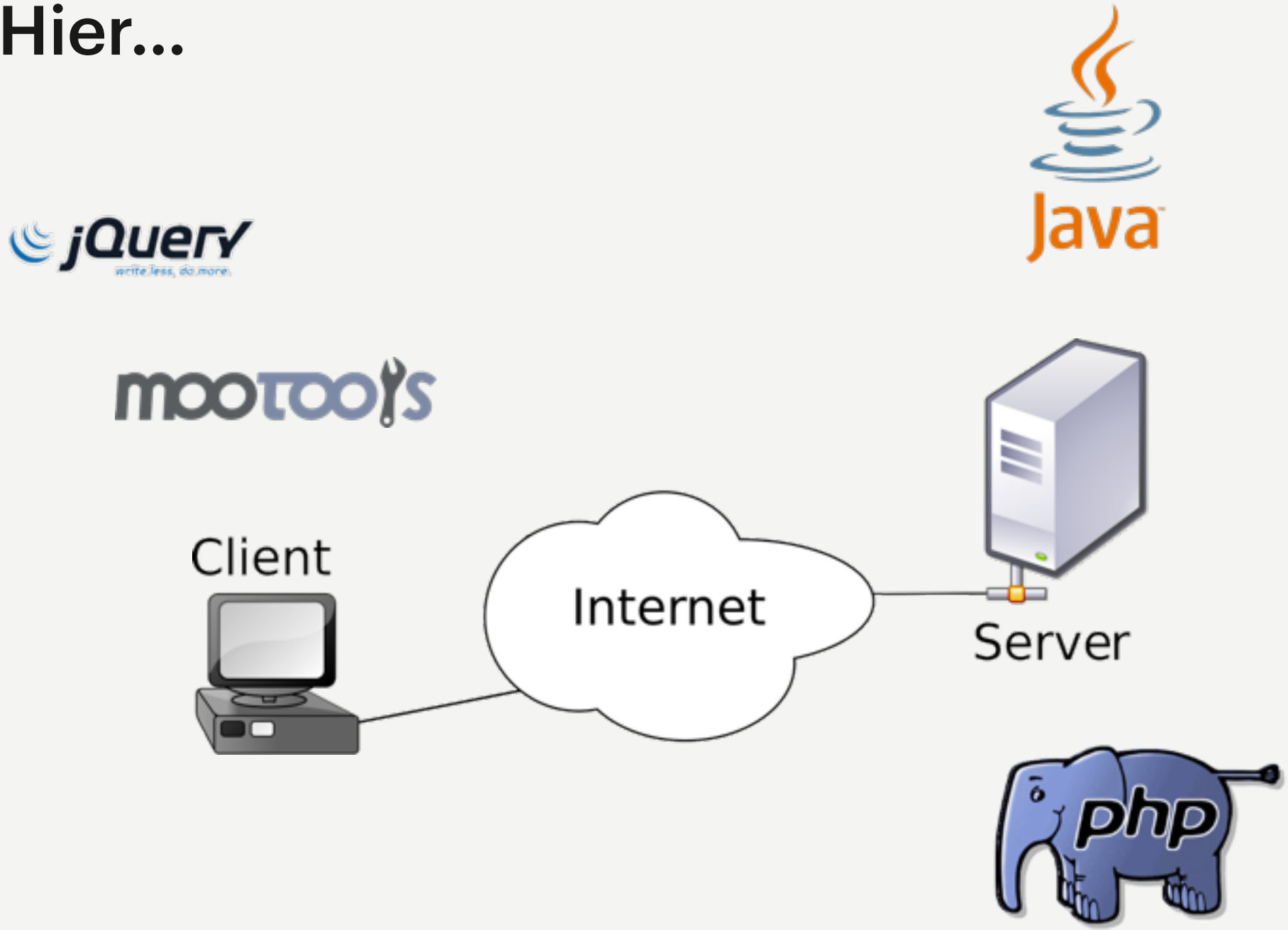
Présentation.



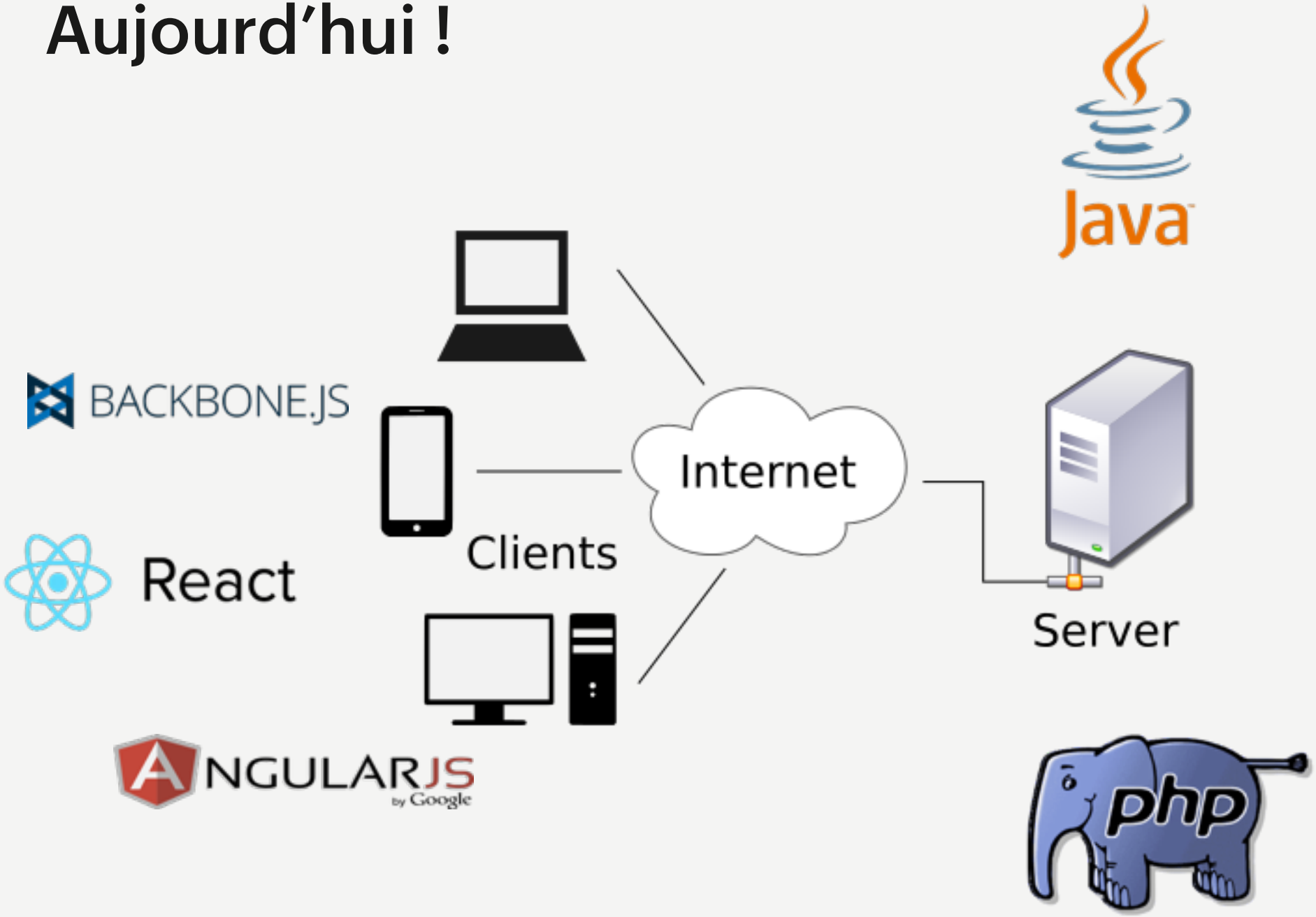
1. Introduction.



Hier...


jQuery
write less, do more.
Java
MOOTOOLS
Client
Internet
Server
php

Aujourd'hui !


Java
BACKBONE.JS
Internet
Clients
Server
React
ANGULARJS
by Google
php

AngularJS.

- **Single Page Application (SPA)**
- **Framework JavaScript libre et open-source**
- Développé par **Google**

- Créé par **Miško Hevery**
  - Pendant un projet Google
  - Utilisation framework personnel
  - 3 semaines refactoring
  - Simplification (17 000 lignes –> 1500 lignes)

AngularJS.

- Sépare l'application en couche de **présentation**, de **données** et de **logique**
- Encourage le **couplage lâche** entre ces composants

La vue c'est le HTML

Enrichissement du langage pour créer des composants personnalisés

---

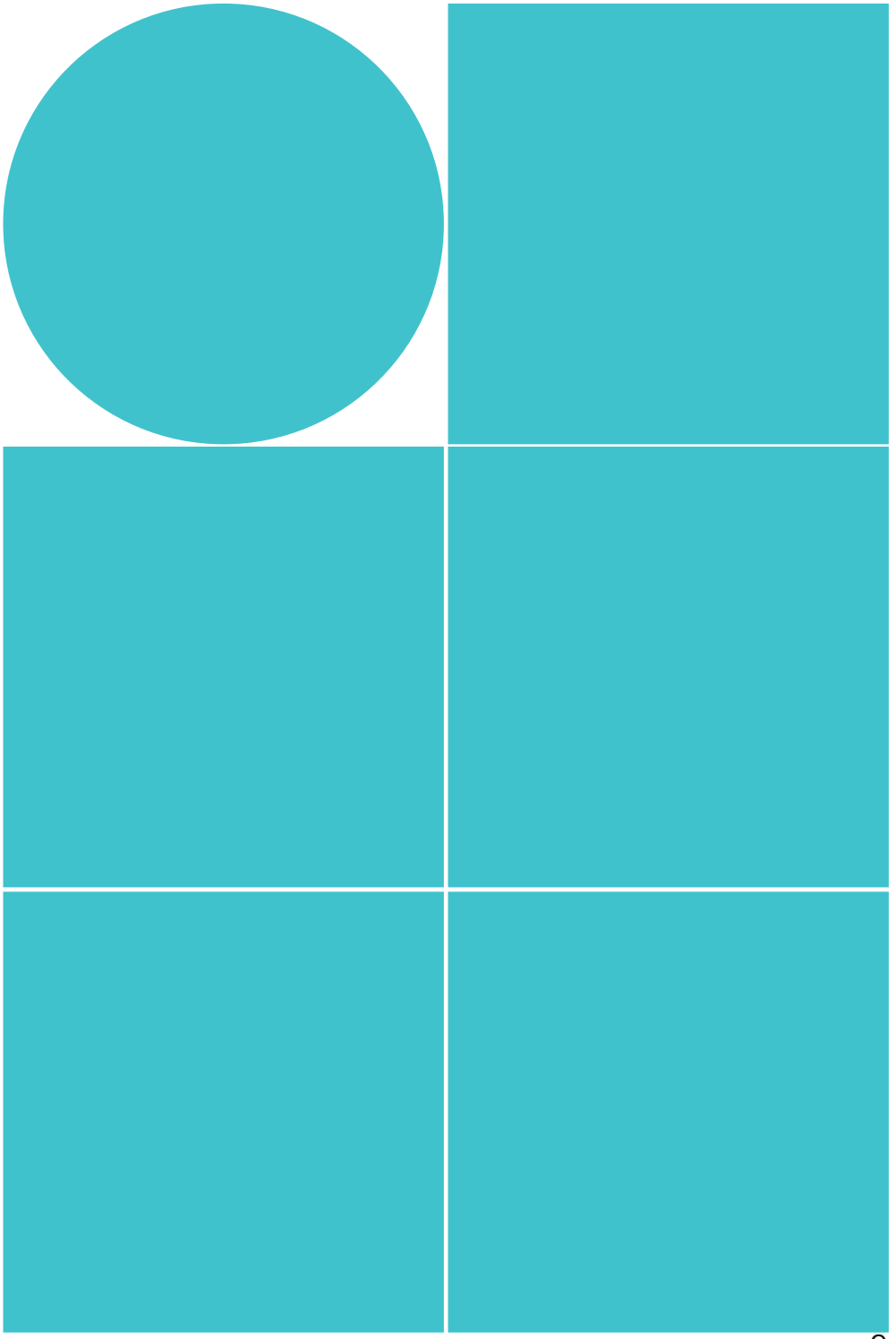
Supprime le code redondant

- DataBinding
- Injection de dépendance

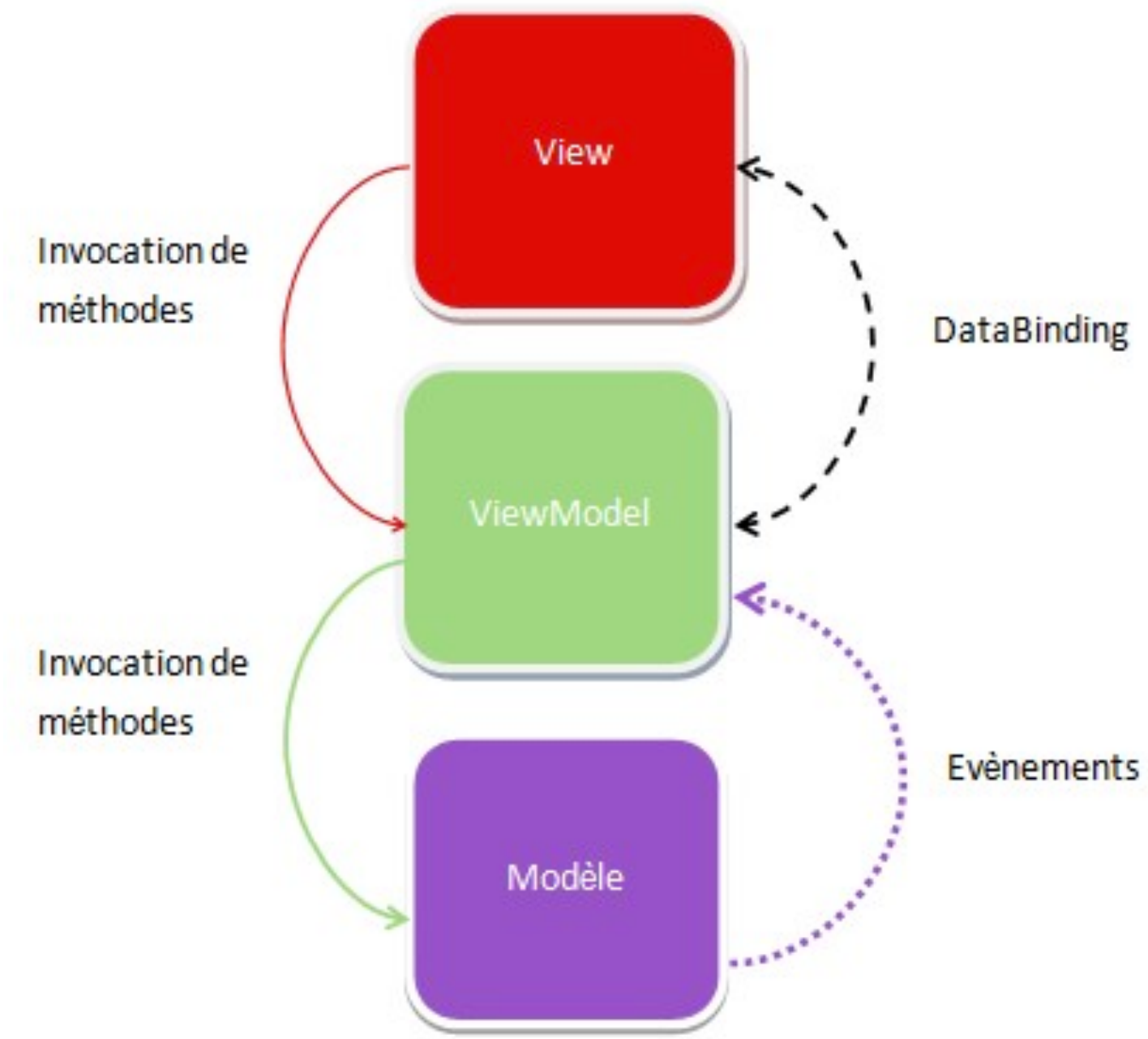
Compilateur HTML

- Directives
- Expressions

2. MVVM.
Model-View View-Model



Workflow MVVM
Lien direct de la vue View vers le WiewModel et du ViewModel vers le modèle. Evènements du Modèle au ViewModel et liaison DataBinding entre la vue View et le ViewModel.



La vue

- Le html (le markup).

Les expressions

- Code évalué dans la vue, utilisé pour la liaison de données (binding).

Le modèle

- N'importe quel objets JS représentant des données.

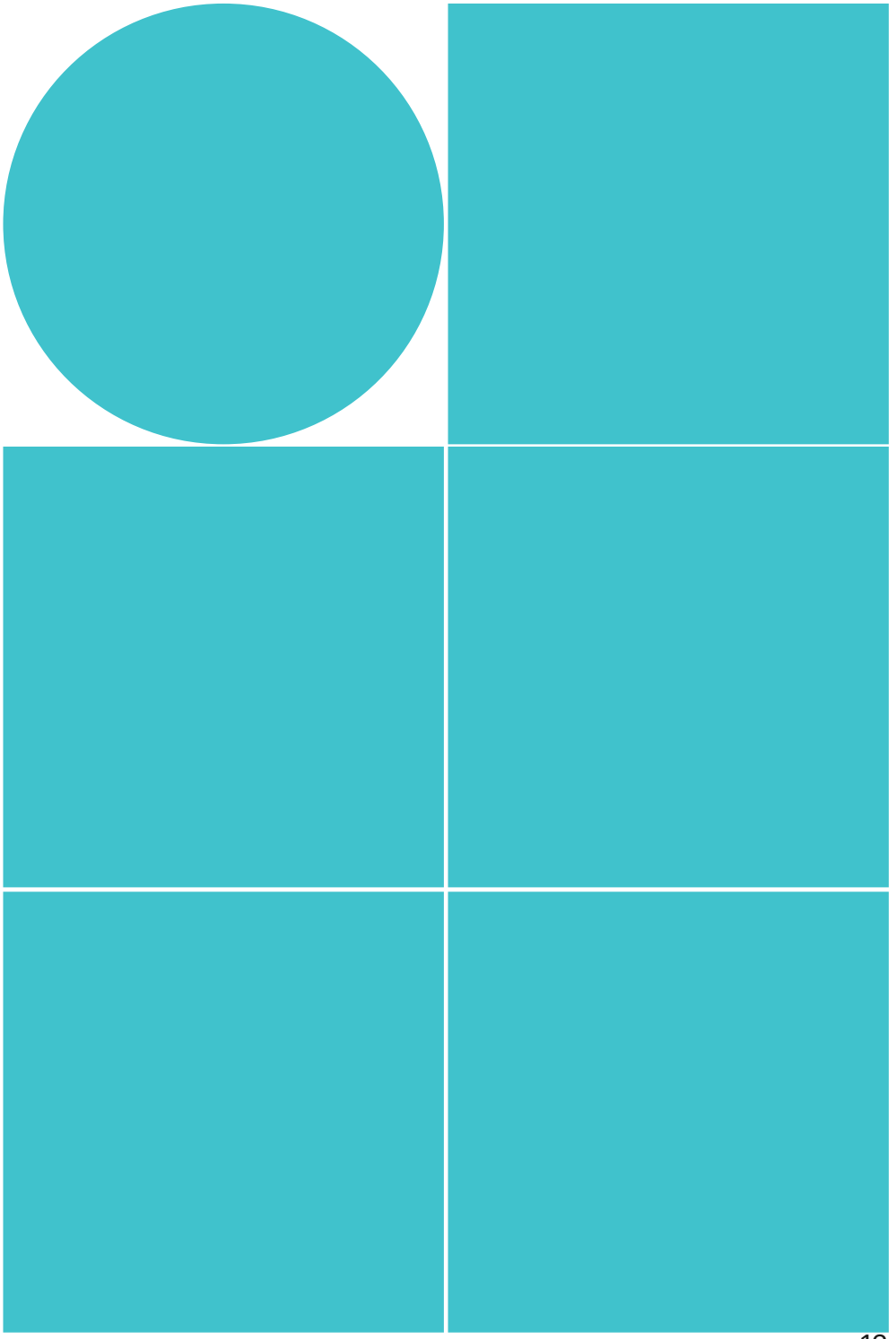
Le scope

- Glue entre la vue, le contrôleur et le modèle.

Le contrôleur

- Rajoute des comportements au scope.

3. Initialisation.



- **Un module dans un fichier js**

```
var avengersModule = angular.module(‘ avengersApp', []);
```

- **Un import de script**

```
<script src="lib/angular/angular.min.js"></script>
<script src="js/app.js"></script>
```

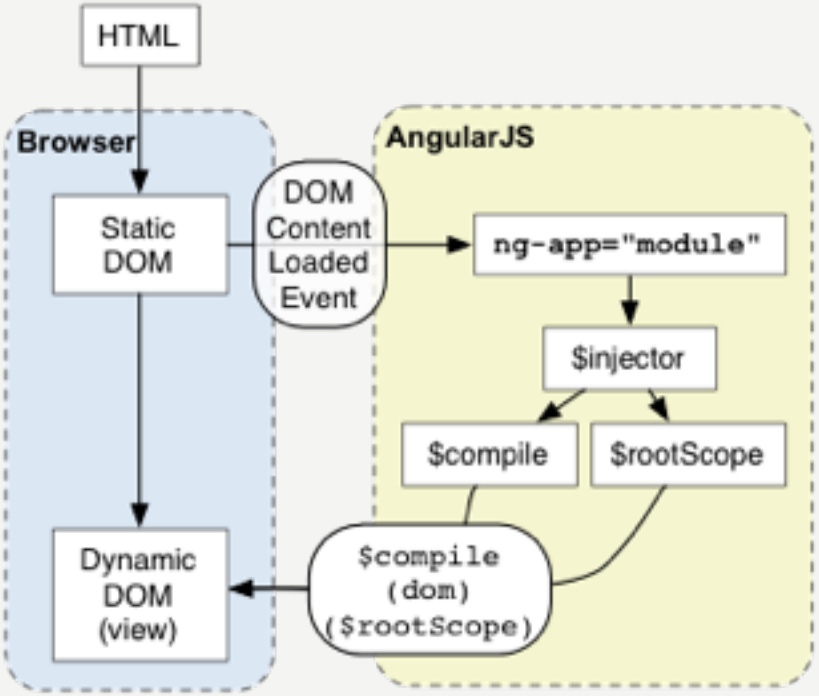
- **Pour instancier l’application:**
  - De façon automatique : une déclaration dans le HTML

```
<html ng-app="avengersApp“>
```

  - De façon manuelle : à l’aide de la méthode bootstrap

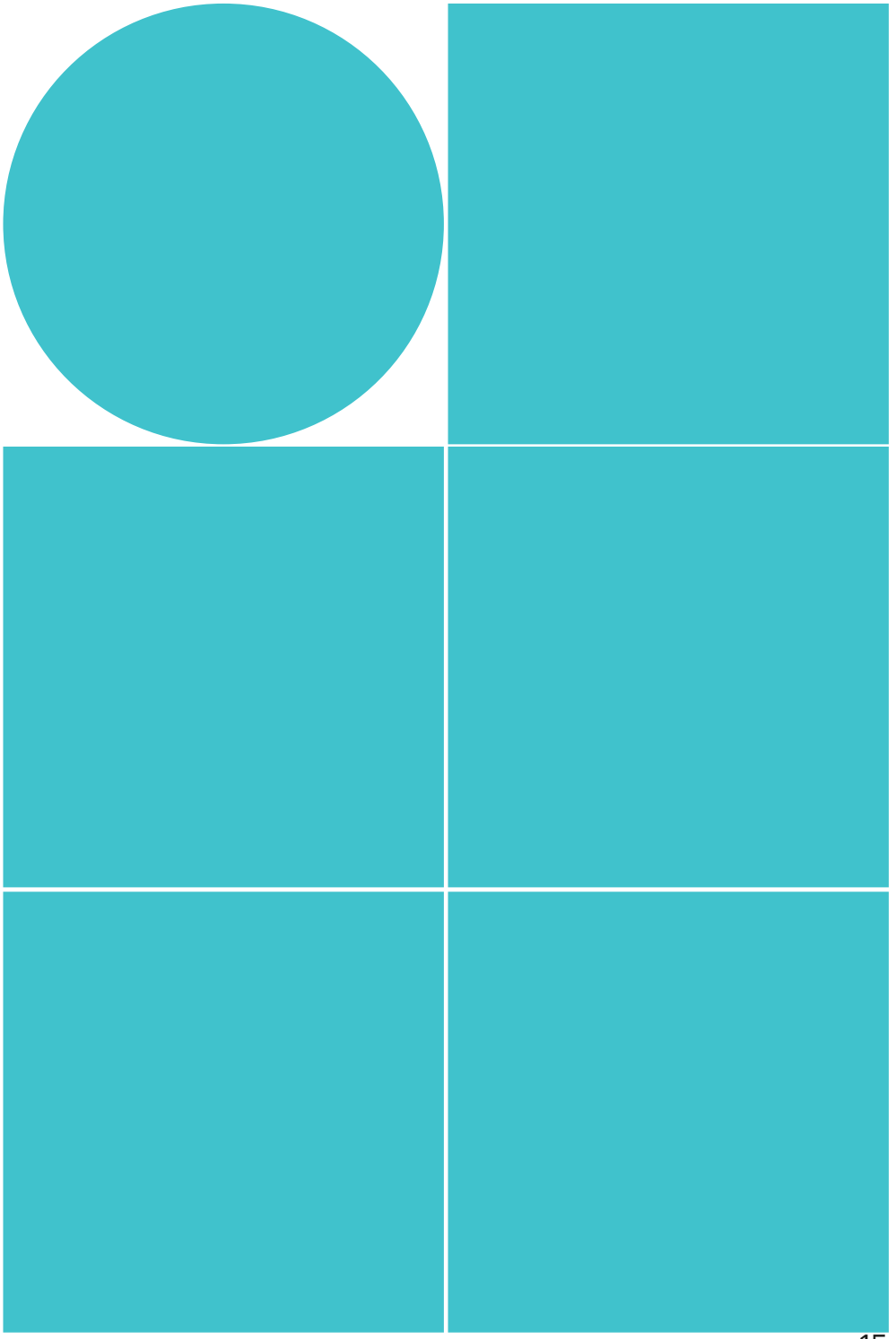
```
angular.element(document).ready(function() {
        angular.bootstrap(document);
});
```

```
<html ng-app>
  <head>
    <script src="angular.min.js">
   </script>
  </head>
  <body>
    <p ng-init=" name='World' ">Hello {{name}}!</p>
  </body>
</html>
```



1	Le navigateur charge le code HTML et l'analyse en un DOM
2	Le navigateur charge le script angular.js
3	Angular attend l'événement DOMContentLoaded
4	Angular cherche une directive ng-app, qui désigne les frontières de l'application
5	Le module spécifié dans ng-app est utilisé pour configurer le **$injector**
6	**$injector** est utilisé pour créer le service **$compile** ainsi que le **$rootScope**
7	le service **$compile** est utilisé pour compiler le DOM et le lier au **$rootScope**
8	La directive ng-init assigne « World » à la propriété « name » du scope
9	{{name}} interpole l'expression en « Hello World!"

4. Expressions.



Expressions

Execute du javaScript côté vue

- Notée entre accolades {{expression}}
- {{ 1+2 }}
- {{ "Hello"+ "Avengers"+2+2 }}
- {{ user.name }}

---

mais...

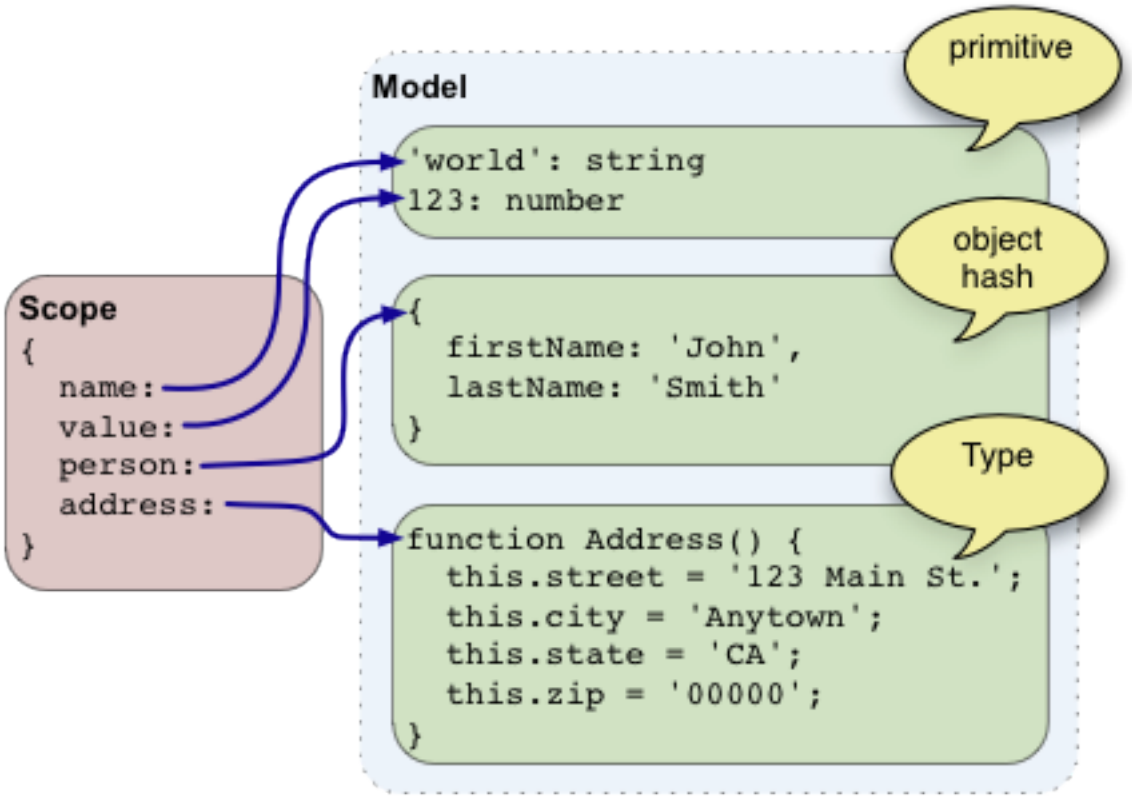
- Ne sont pas évaluées sur l'objet window mais sur le **scope**
- Ne provoquent pas de NPE
  (Null Pointer Exception)
- Peuvent être filtrées: {{ expression | filter }}

Tips

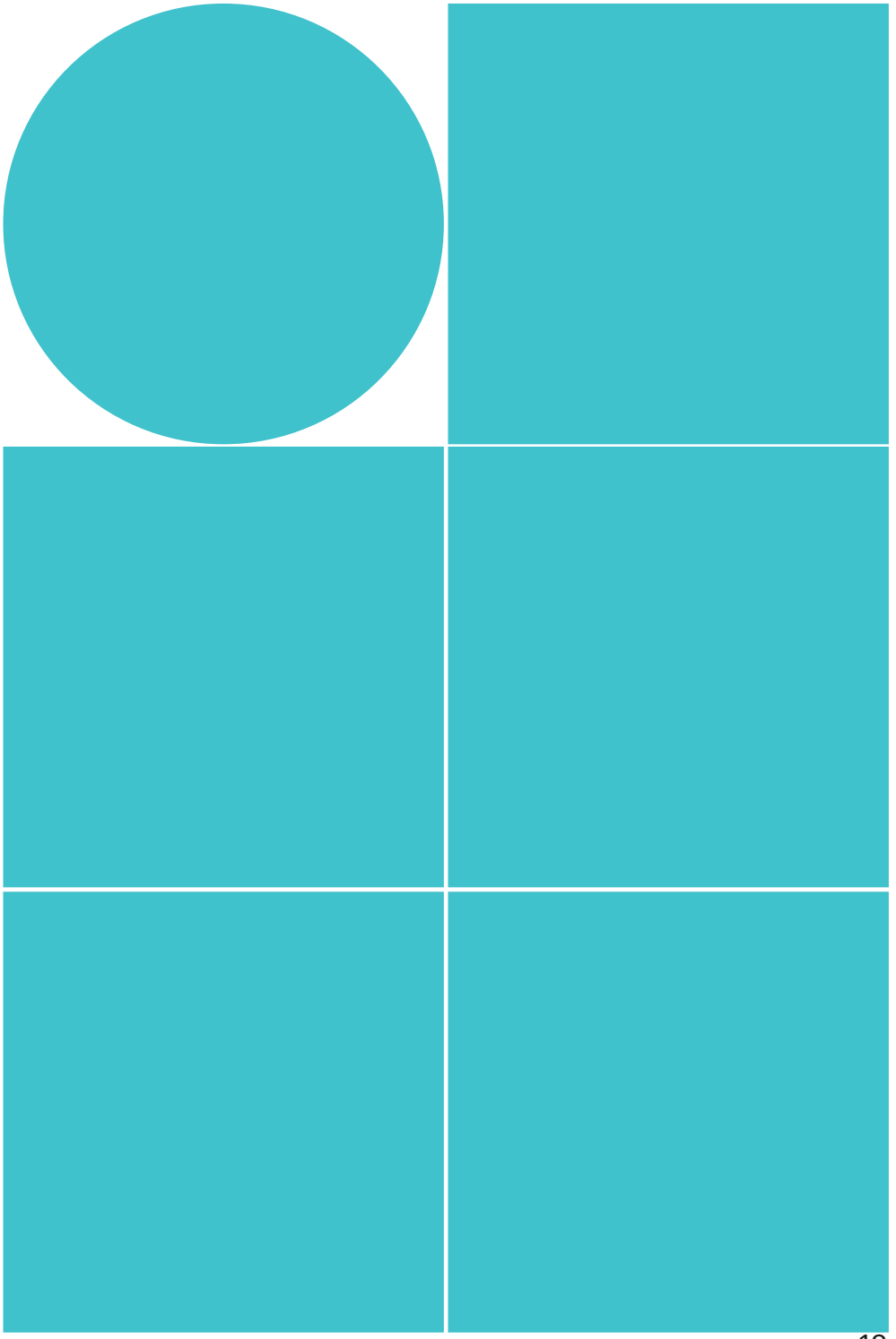
- $eval : exécution par le javaScript

```
scope.a = 1;
scope.b = 2;
expect(scope.$eval('a+b')).toEqual(3);
```

- **Type d'objet au sein d'une expression**



5. Contrôleur.



Contrôleur

Ce que fait le contrôleur:

- Donne l'état initial du scope
- Ajoute du comportement au scope
- Invoque des services

Dois contenir le minimum de logique possible!

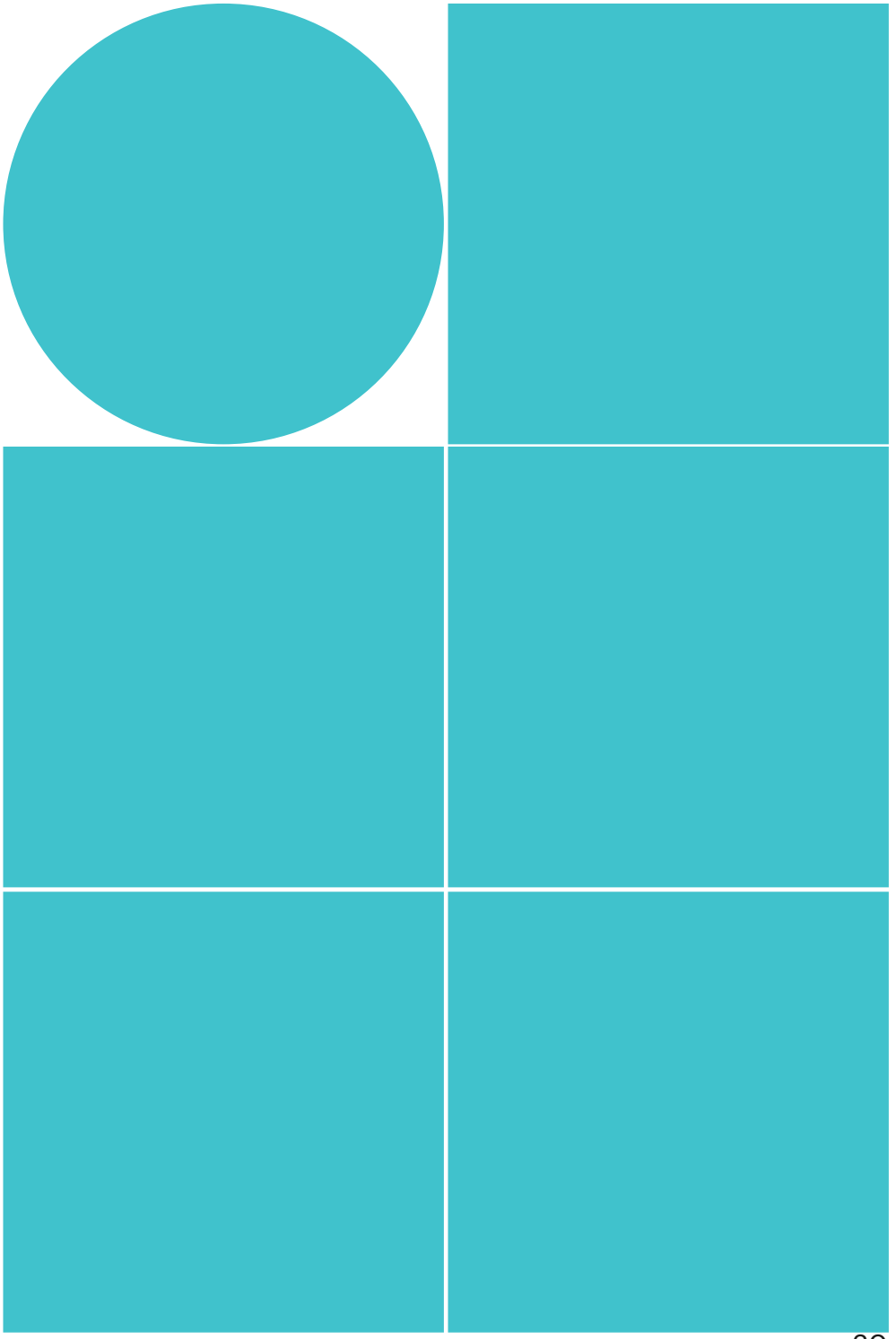
Création d'un contrôleur

```
angular.module("avengersApp")
.controller("MainCtrl",function($scope){
    $scope.title = "Hello Avengers !";
});
```

```
<div ng-controller="MainCtrl"></div>
```

PAS DE MANIPULATION DU DOM DANS LES CONTROLEURS ~~jQuery~~

6. Filtres.



Les filtres prédéfinis

- **currency**
  - Pour transformer le nombre en chaine de caractère avec un symbole monétaire
- **date**
  - Pour formater une date
- **filter**
  - Pour filtrer un tableau
- **json**
  - Pour transformer un objet en chaine de caractère
- **limitTo**
  - Pour retourner un ensemble connu d'éléments d'un tableau
- **lowercase/uppercase**
  - Pour transformer en minuscule ou majuscule la valeur filtrée
- **number**
  - Pour transformer le nombre en chaine de caractère
- **orderBy**
  - Pour trier un tableau

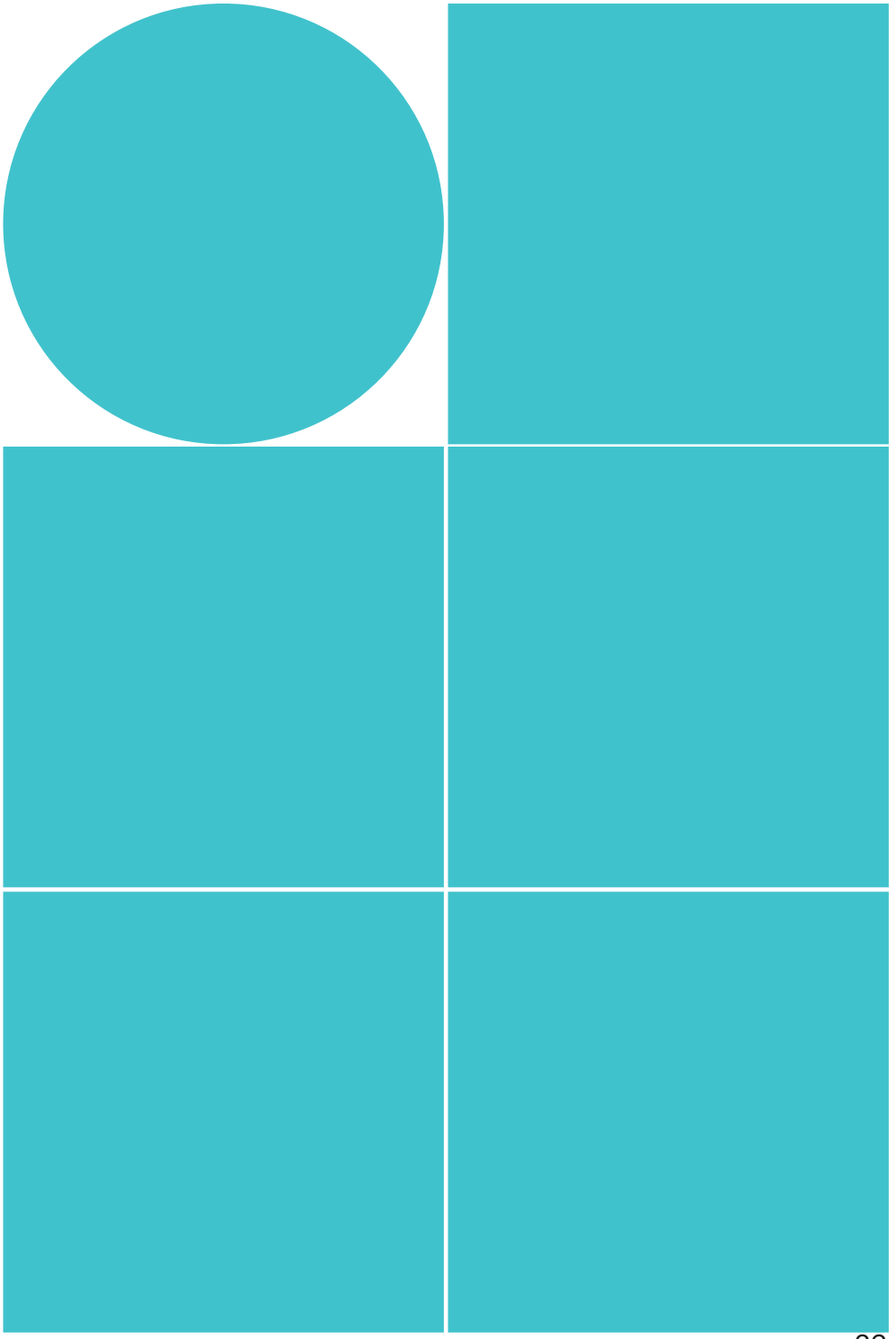
Filtres

- **Permettent de formater une donnée**
  - {{ expression | filter1 }}

- **On peut les chainer**
  - {{ expression | filter1 | filter2 }}

- **On peut leur passer des paramètres**
  - {{ expression | filter1:param1:param2 }}

7. Création de filtre.



Déclaration :

```
angular.module('monApp', []).
  .filter('myFilter', [function () {
        return function (input) {
              return input.charAt(0).toUpperCase() + input.slice(1);
        };
  }]);
```

Enregistrer un nouveau filtre avec une fonction de transformation.
Première lettre d'une chaîne de caractères en majuscule.

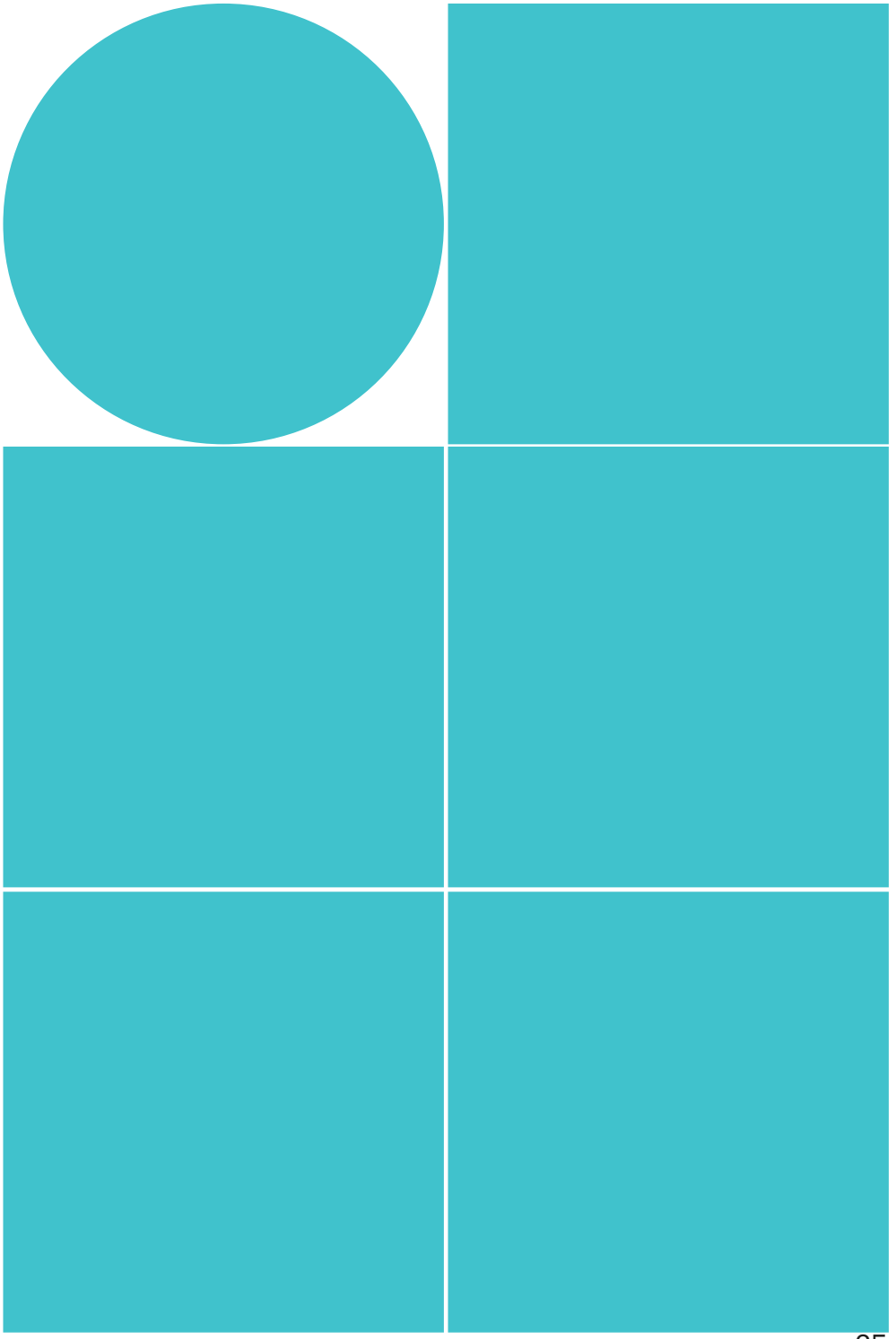
Instanciation dans la vue:

```
<span>{{ expression | myFilter}}</span>
```

Instanciation javascript:

```
$filter('myFilter')(array,expression)
```

8. Directives.



- **Utilisés pour attacher un comportement au Html**
- **Les directives peuvent être utilisées en :**
  - **Attribut** (recommandé)

```
<div ng-xxx="{expression}">
```

  - **Classe**

```
<div class="ng-xxx : {expression}">
```

  - **Element** (déconseillé)

```
<ng-xxx src=" {expression}"></ng-xxx>
```

  - **Commentaire** (déconseillé)
  - <!-- ng-xxx: {expression} -->

{{expression}}

- Une propriété
- Une chaine entre ‘’ ‘’
- Une fonction …

Applicative

- ng-app
  - Pour definir l'application angular
- ng-controller
  - Pour créer un nouveau scope et son contrôleur
- ng-repeat
  - Permet d'itérer et de répéter un block HTML pour chaque élément d'une collection
- ng-model
  - Pour établir un binding bidirectionnel
- ng-view
  - Pour setter le conteneur utilisé par le service de route

Affichage

- ng-class
  - ng-class="{ ‘isSelected’ : true }"
- ng-show, ng-hide, ng-if
  -
- ng-disabled
  - <button ng-model="button" ng-disabled="checked">Button</button>
- ng-attr
  -
- ng-style
  - Sample Text
- ng-class-even, ng-class-odd
  -

Traitements conditionnels

- ng-switch
  - Pour définir du contenu alternatif

```
<div ng-switch on="selection" >
    <div ng-switch-when="settings">Settings Div</div>
    <span ng-switch-when="home">Home Span</span>
    <span ng-switch-default>default</span>
</div>
```

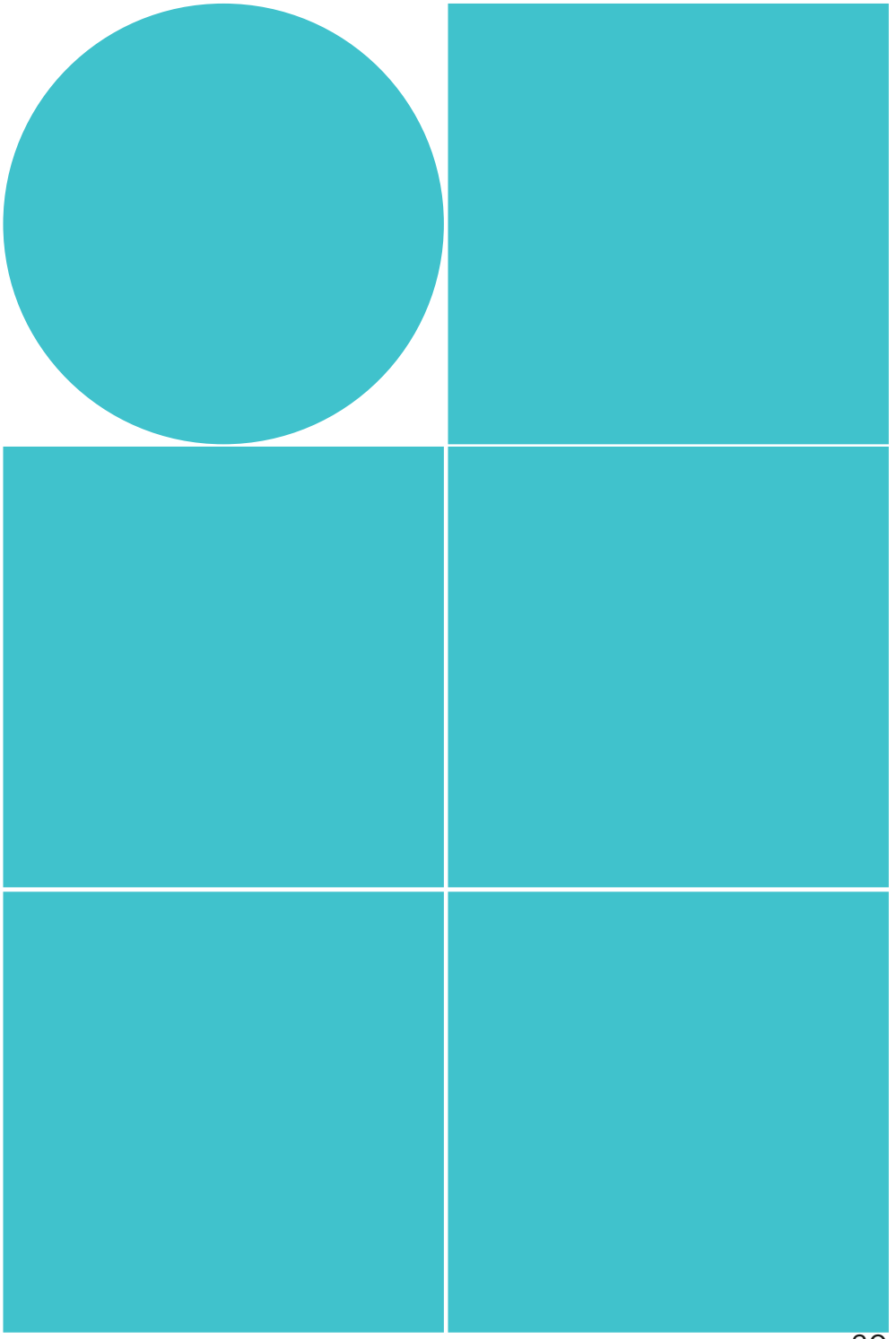
- ng-include
  - Pour intégrer un template

```
<div ng-include src="template.url"></div>
```

- ng-pluralize
  - Pour adapter des labels en fonction d'un nombre d'éléments

```
<ng-pluralize count="personCount"
              when="{ '0': 'Nobody is viewing.',
                      'one': '1 person is viewing.',
                      'other': '{} people are viewing.' }">
</ng-pluralize>
```

9. DOM events.



Gérer les actions utilisateur

- **Utiliser les directives natives**
  - ng-click / ng-dbl-click / ng-mouseenter / ...
  - ng-keydown / ...
  - ng-change
  - ng-focus / ng-blur
  - ng-submit
  - ...

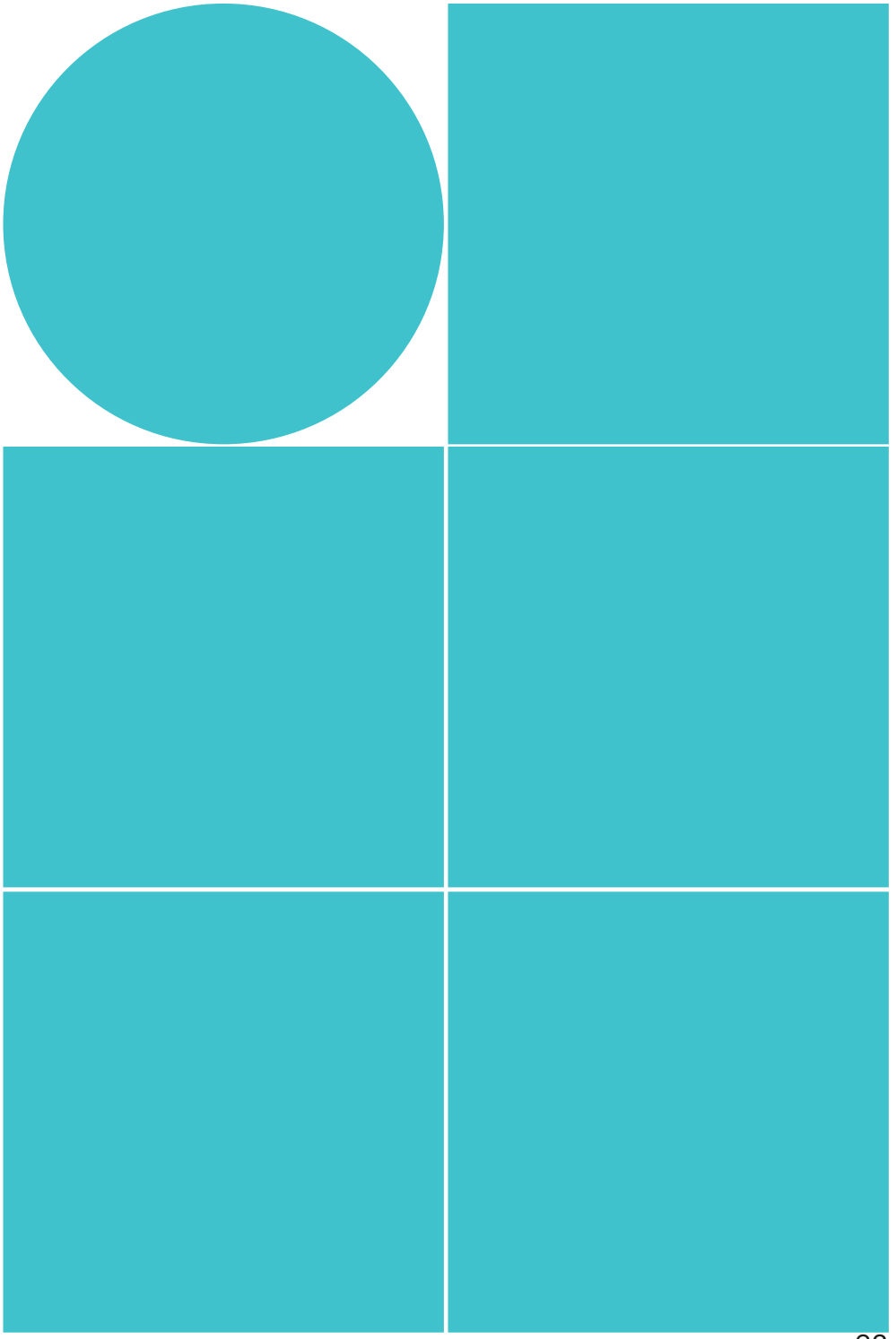
```
<button ng-click="hello('John')">Say hello</button>
<span>{{sentence}}</span>

$scope.hello = function(name) {
    $scope.sentence = ('Hello ' + (name || 'world') + '!';
}
```

  - $event

```
<button ng-click=”onClick($event)”>
$scope.onClick=function($event){$event.preventDefault()
```

10. Création de directive.



- **Pour enrichir le html**
  - **Modifier le DOM**
    - Créer un composant
    - Intégrer une librairie tiers (plugins externe)
  - **Créer de nouveaux comportements**
    - Ecouter des évènements (resize)
- **Réutilisable**
  - Notion de scope isolé

Pour invoquer une directive

- Le nom utilisé peut être (ex: maDirective)
    - ma-directive
    - ma:directive
    - ma_directive
    - x-ma-directive
    - data-ma-directive
- La directive peut être utilisé dans:
    - Un nom d'élément
        - <ma-directive>
    - Un attribut
        -
    - Un nom de class
        -
    - Un commentaire
        - <!-- directive: ma-directive exp -->

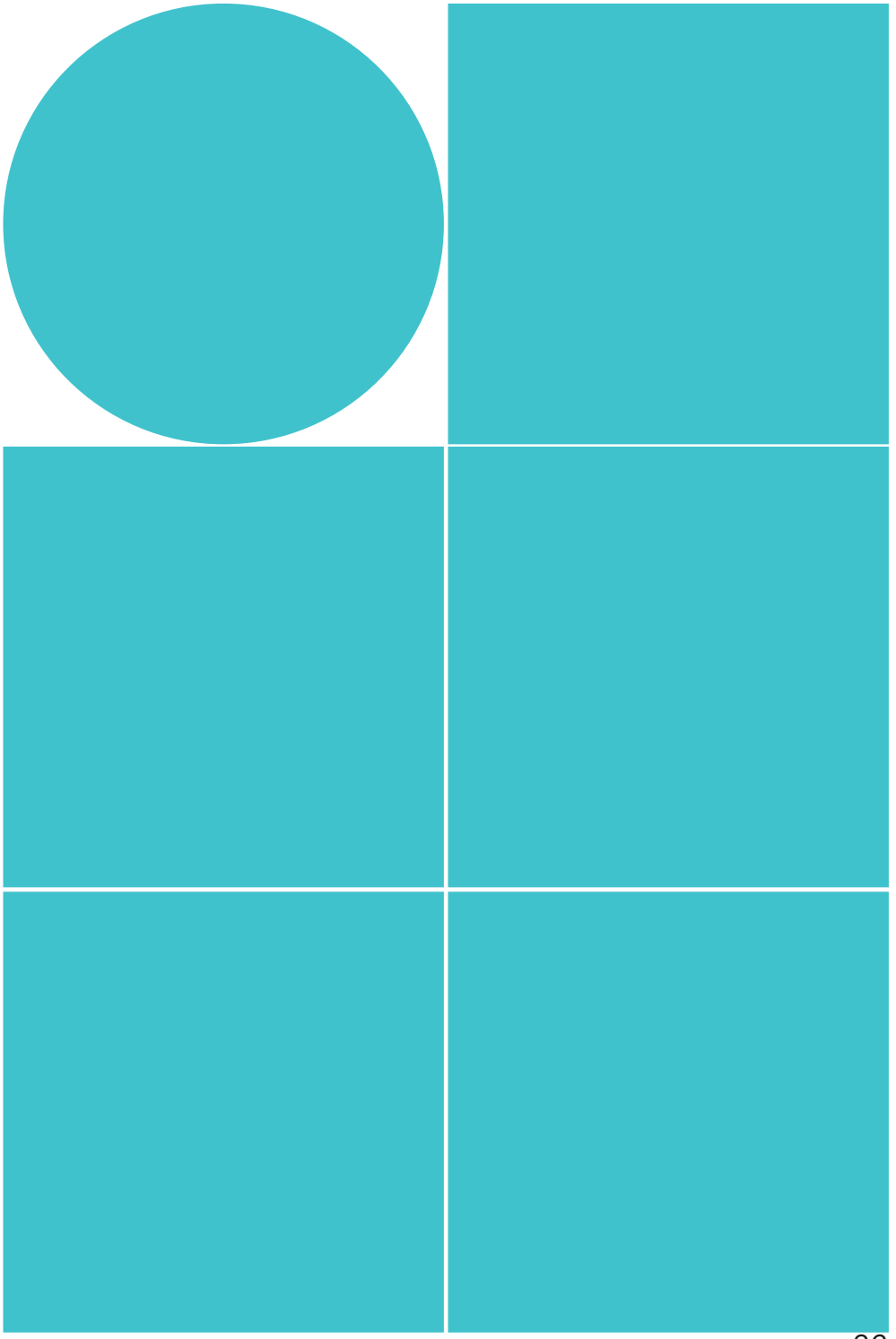
Paramètres d'une directive

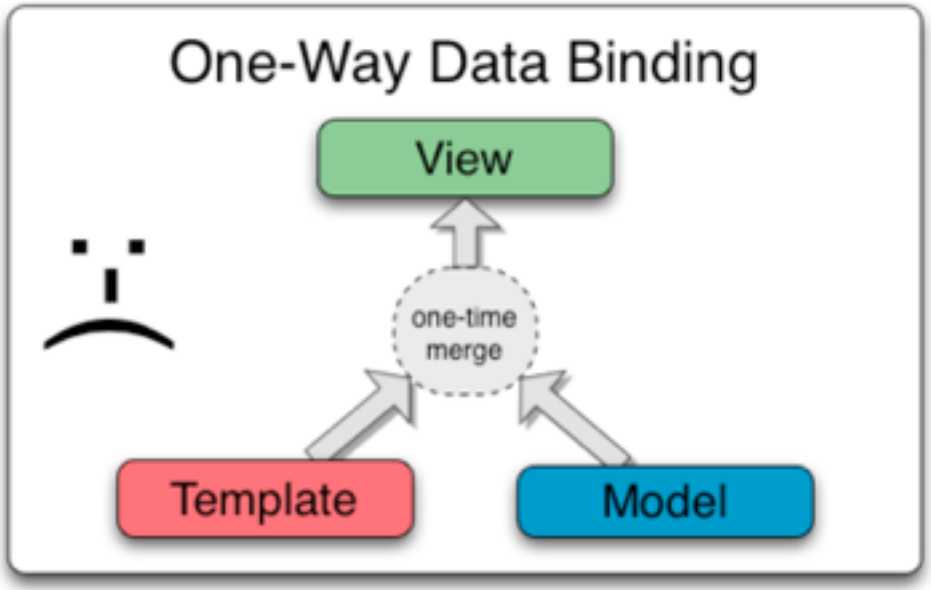
- **template : Compile le html avec une string**
- **templateUrl : Compile le html avec un fichier template**
- **link: Initialisation javascript de la directive**
- **replace: true Permet d'écraser l'élément DOM avec le template**
- **transclude : true Permet de cibler un élément enfant**

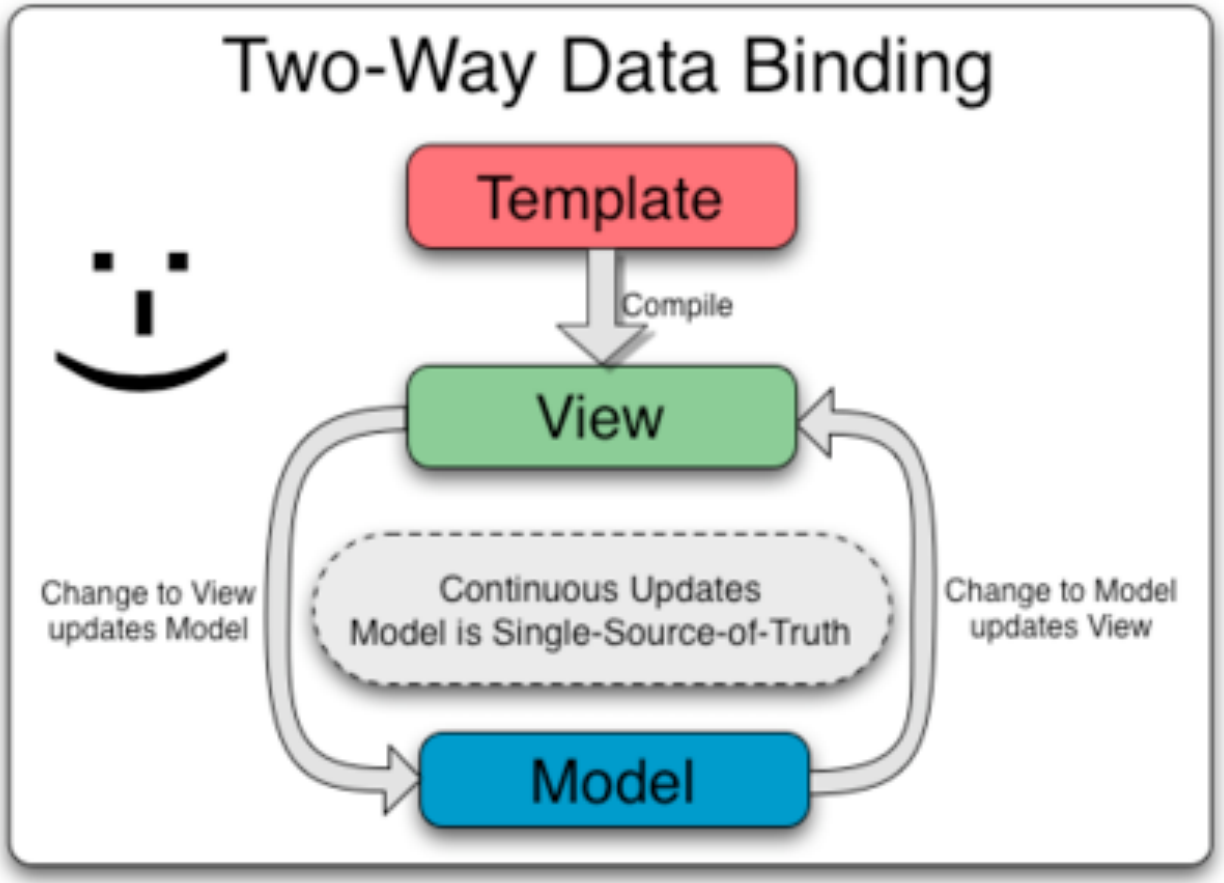
```
angular.module('avengersApp')
   .directive('maDirective', function() {
    return {
          template: '<span></span>'
          link: function(scope, element, attr) {

          }
      };
   })
```

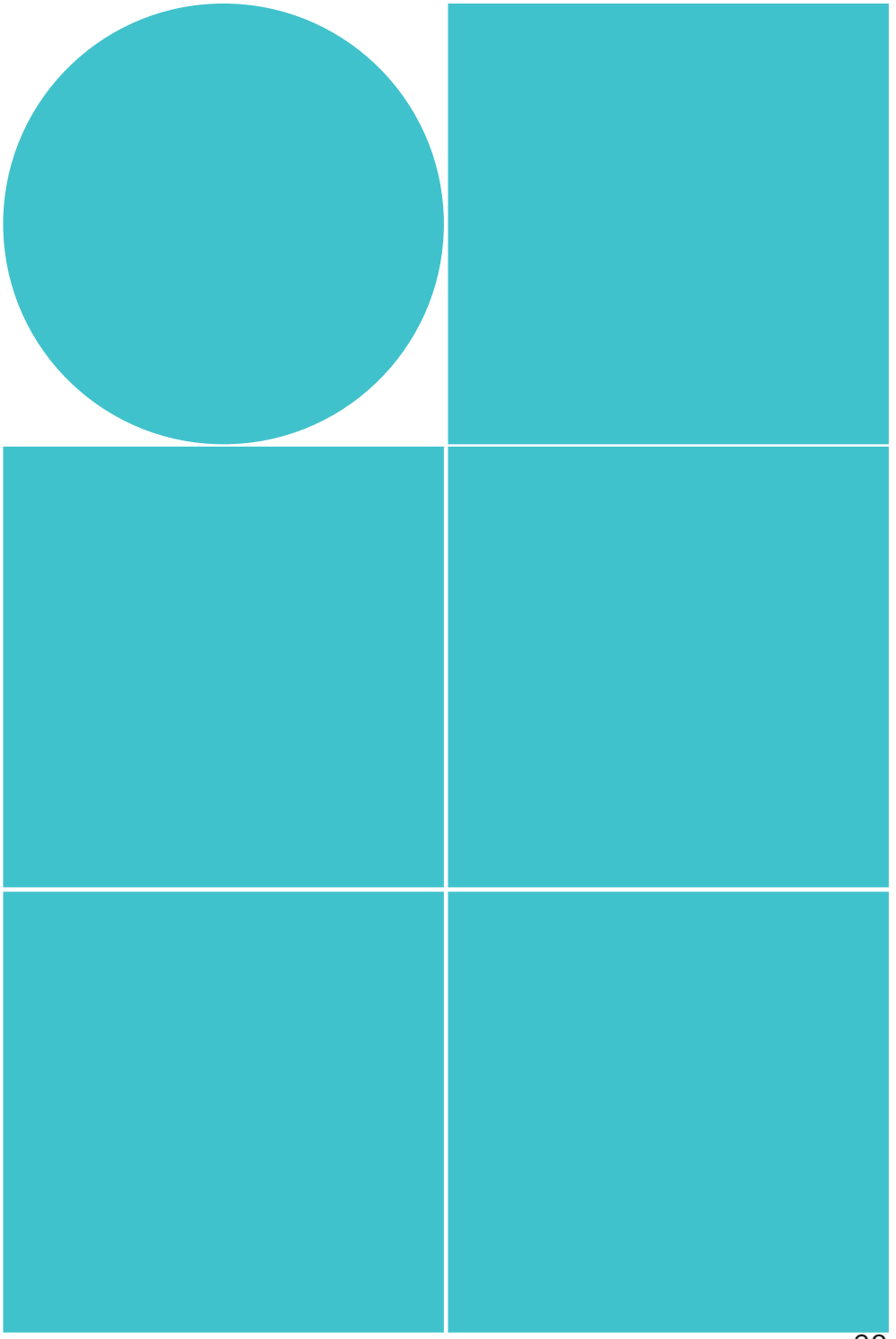
11. Data binding.



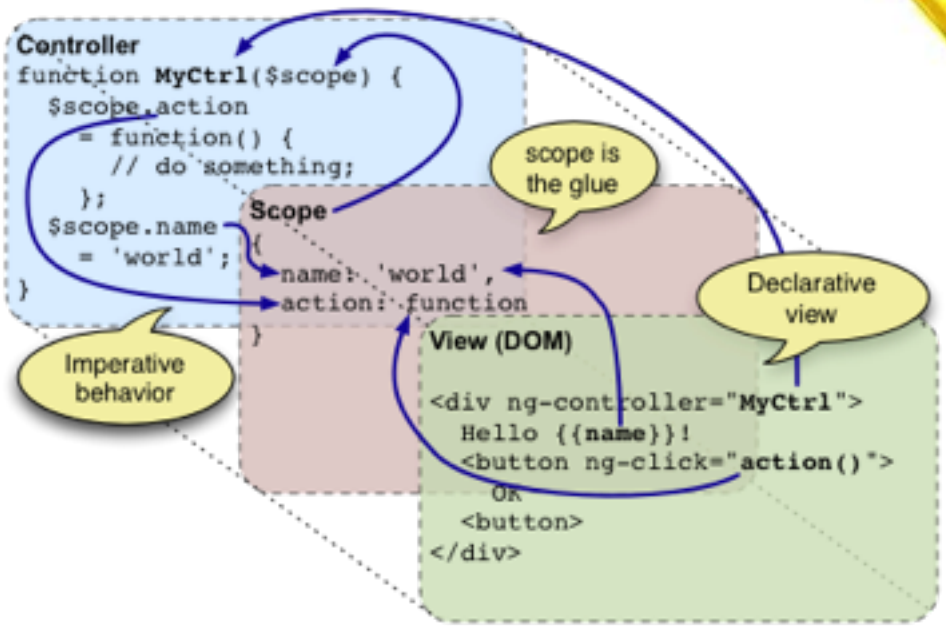

One-Way Data Binding
View
one-time
merge
Template
Model


Two-Way Data Binding
Template
Compile
View
Change to View
updates Model
Continuous Updates
Model is Single-Source-of-Truth
Change to Model
updates View
Model

12. Scope.



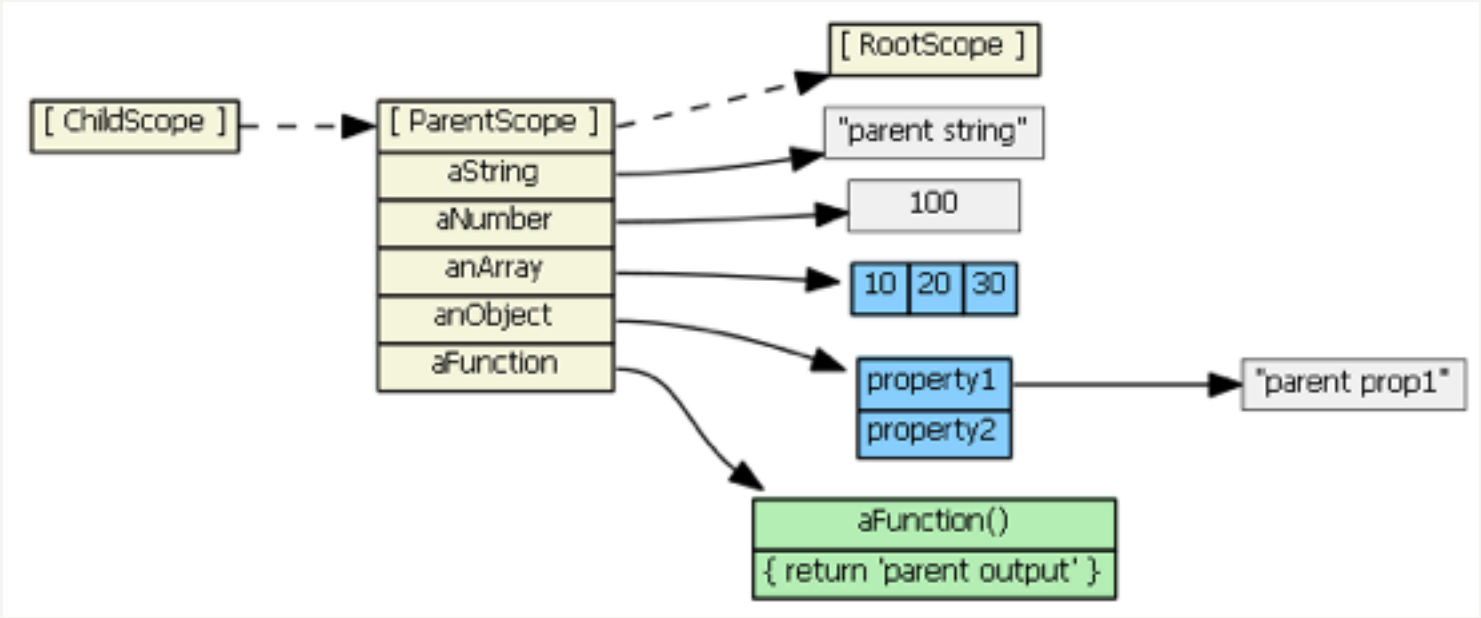
- **Pont entre la vue et le model**



- **$watch pour observer les changements du model**
- **$apply pour propager les changements**
- **Notion de hiérarchie : « child scope» « parent scope »**
- **Notion de scope implicite pour l'évaluation des expression {{expression}}**

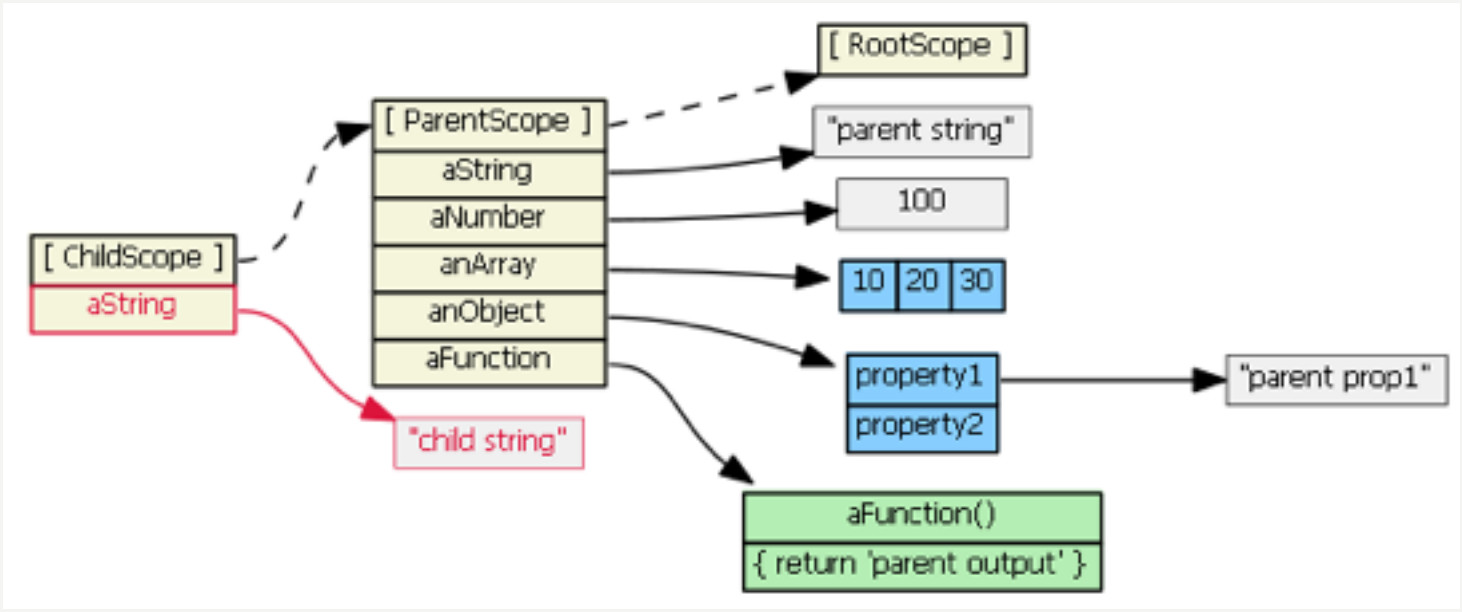
- **Structure hiérarchique**

Scopes & Héritage



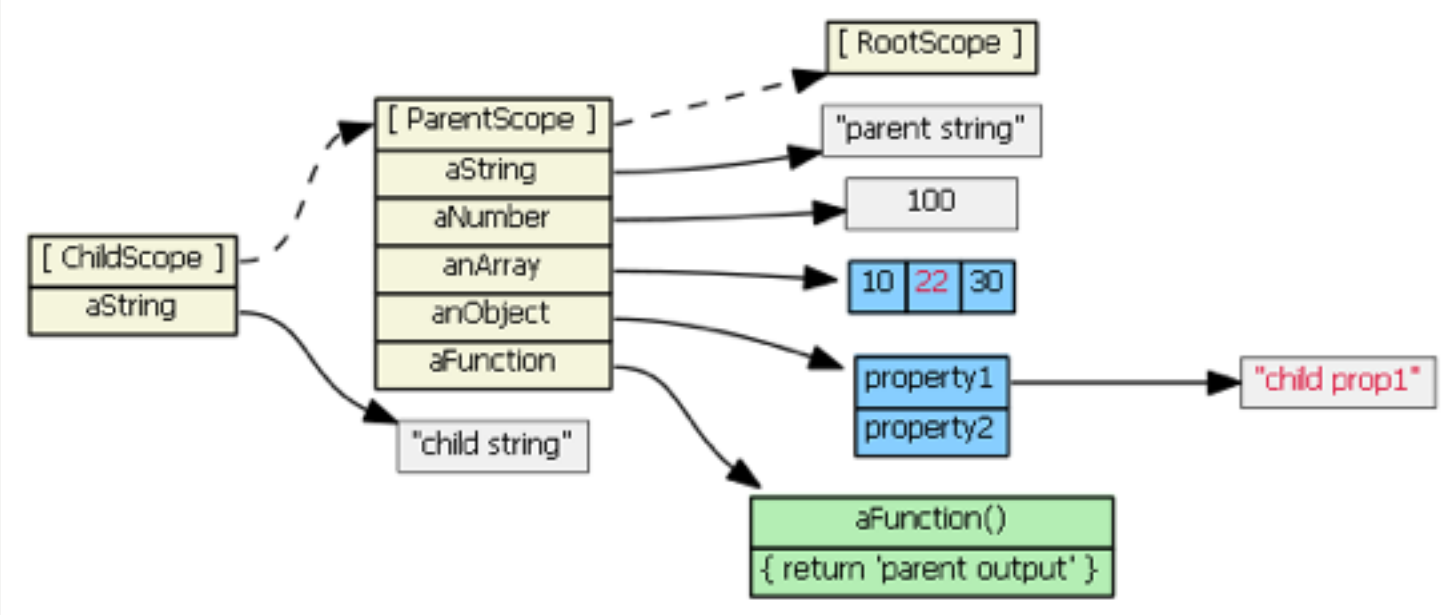
rootScope, parentScope, childScope

Scopes & Héritage



childScope.aString = 'child string'

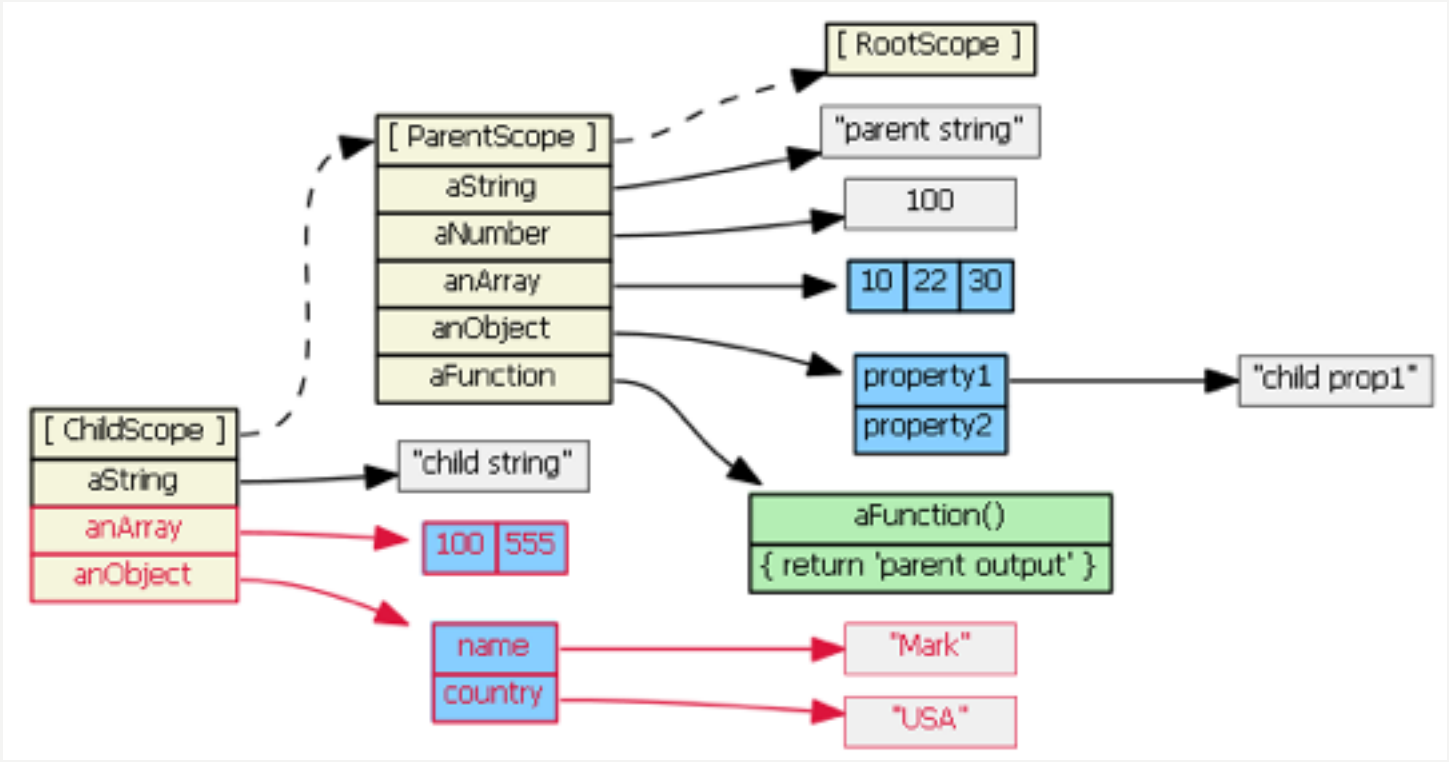
Scopes & Héritage



childScope.anArray[1] = '22'
childScope.anObject.property1 = 'child prop1'

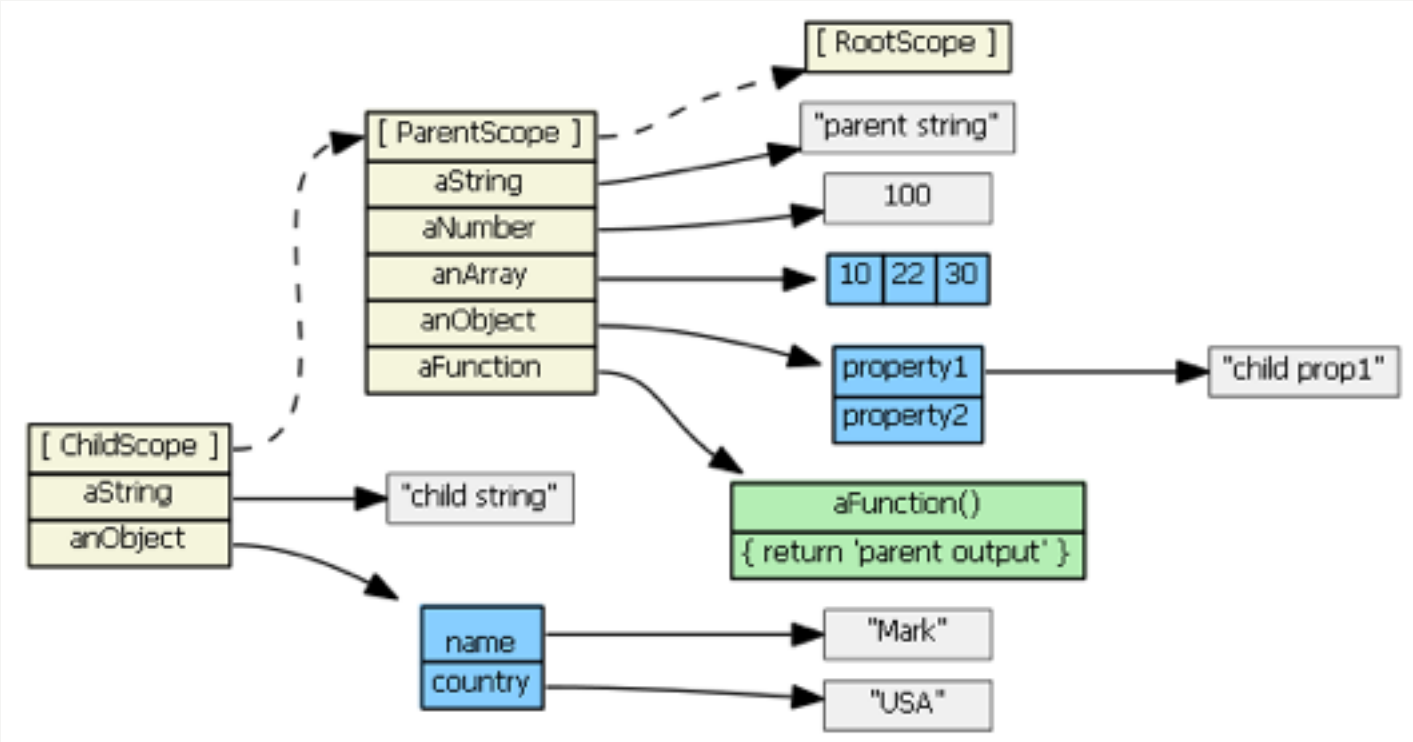
Ici on conserve l'héritage !

Scopes & Héritage



childScope.anArray = [100, 555]
childScope.anObject = { name: 'Mark', country: 'USA' }
Ici on casse l'héritage !
(on perd la référence à l'objet parent mais l'objet parent est conservé)

Scopes & Héritage



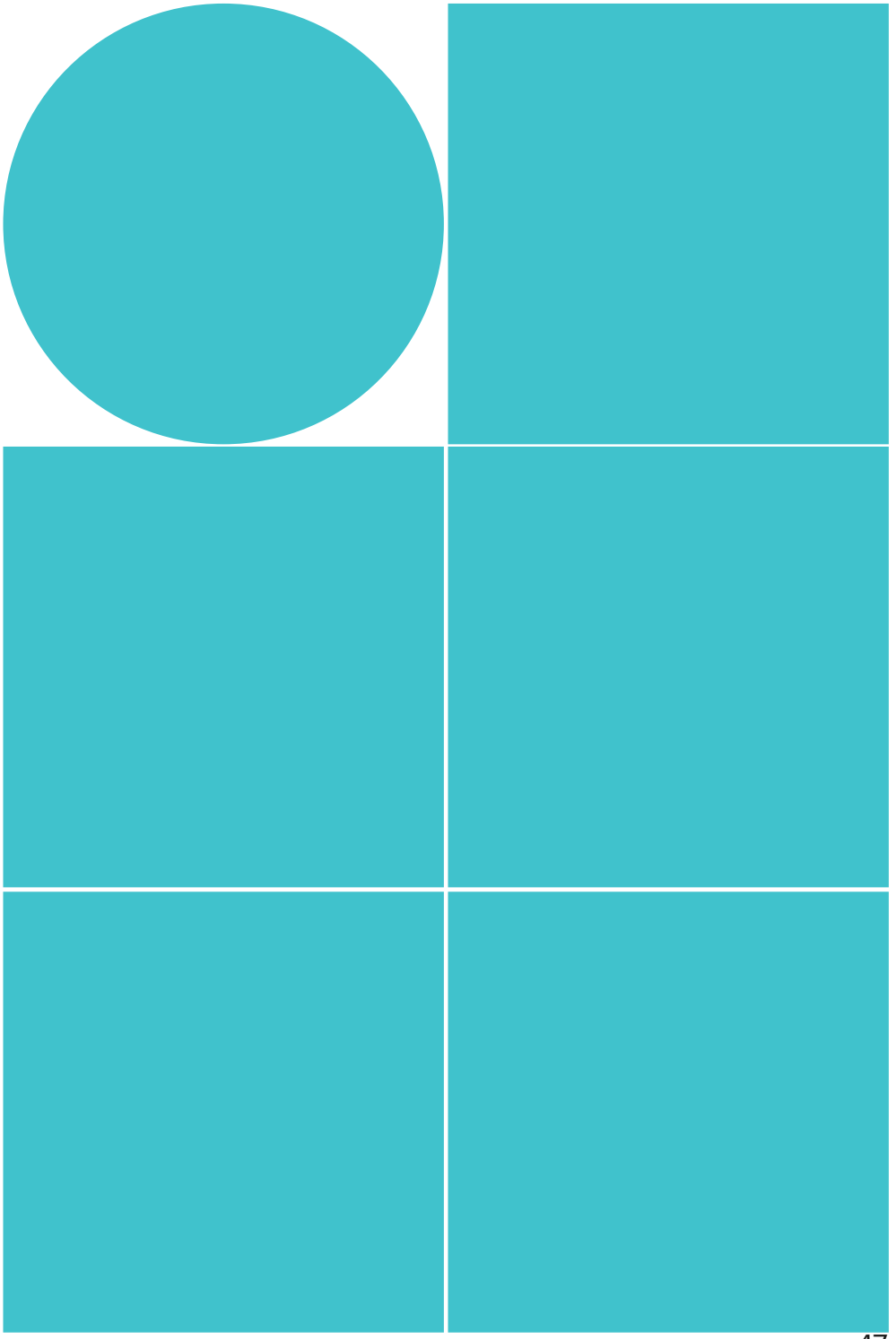
delete childScope.anArray
childScope.anArray[1] === 22 // true

(on remarque que l'on a pas casser l'héritage du scope)

- **Evènements**
  - $emit
    Pour propager un événement vers les scopes parents
  - $broadcast
    Pour propager un événement vers les scopes enfants
  - $on
    Pour écouter un événement

- **Debug**
  - angular.element($0).scope()

13. Injection de dépendances.



Injection de dépendances

- Pour une classe il y a 3 façons de gérer une dépendance

  1. L'instancier (new)
  2. La récupérer de façon définie (variable globale, singleton)
  3. **Se la faire fournir**

Dans Angular, DI privilégié : c'est le rôle de l'**injector**

- Une dépendance est référencée par son nom

- Utilisable dans:

  - Les blocs de configuration (provider/constant) et d'exécution
  - Les controller
  - Les directives
  - Les filters
  - Les méthodes factory

- Mise à plat depuis les différents modules

Injection de dépendances

```
angular.module('myModule', [])
.factory('serviceId', ['depService', function(depService) {
  // ...
}])
.directive('directiveName', ['depService', function(depService) {
  // ...
}])
.filter('filterName', ['depService', function(depService) {
  // ...
}]);
```

```
angular.module('myModule', [])
.config(['depProvider', function(depProvider) {
  // ...
}])
.run(['depService', function(depService) {
  // ...
}]);
```

Injection de dépendances

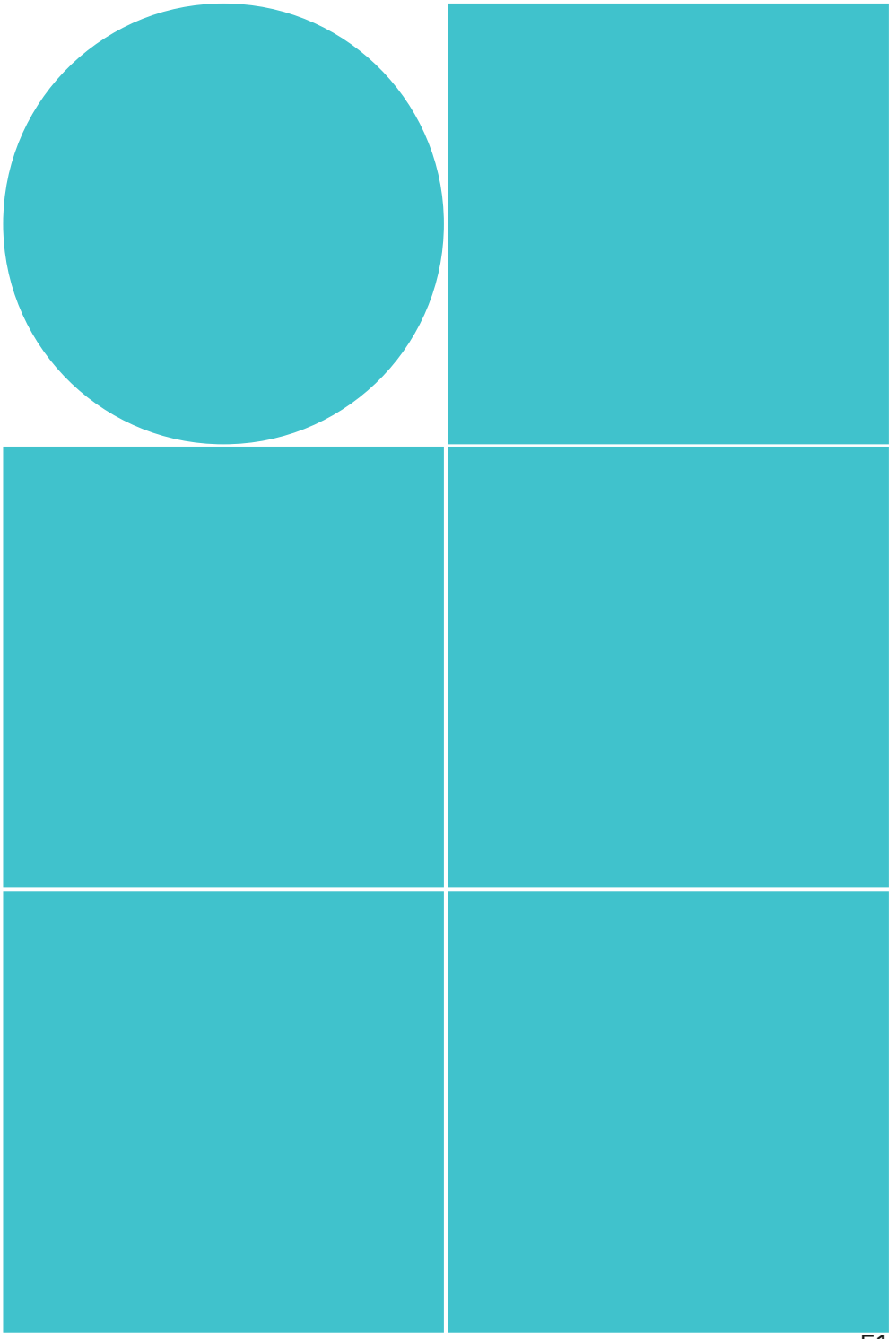
```
someModule.controller('MyController', ['$scope', 'dep1', 'dep2', function($scope, dep1, dep2) {
  ...
  $scope.aMethod = function() {
    ...
  }
  ...
}]);
```

- **Simplicité**
  Le framework Angular instancie pour nous les composants.
  L'injection de dépendance nous permet d'utiliser facilement un composant au sein d'un autre.

- **Fiabilité**
  Interdépendances des composants injectés.

- **Réutilisabilité**
  Incite le développeur à créer des composants unitaire permettant l'élaboration de système plus complexe.

14. Services.



- **Logique métier de l'application.**
- **Organisation et partage au sein de l'application.**
- **Instancier uniquement en cas de besoins vi l'injection de dépendance.**
- **Singleton.**
- **Possibilité de créer son propre service.**

Laquelle choisir ?

provider(name,provider)
- Si on veut configurer le service

factory(name,$getFn)
- Par défaut si on veut de l'injection

service(name,constructor)
- Si on veut typer notre service (instanceof)

value(name,value)
- Pour un service simple

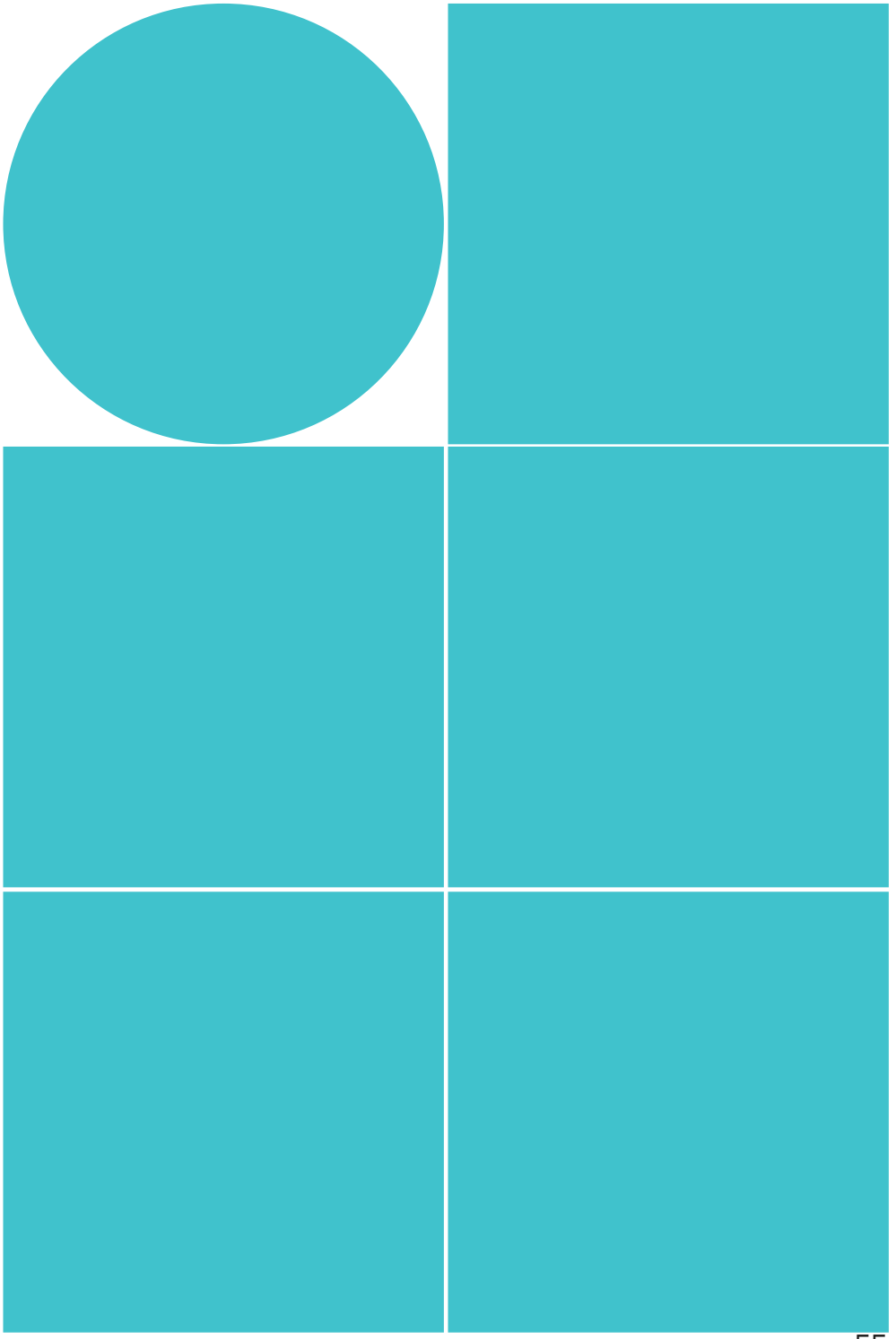
constant(name,value)
- Pour un service simple utilisable en phase Config

Méthode	Configurable	Injection de dépendances	Création du service	Nom du provider
module.provider()	Oui	Oui	Au premier accès, par appel de la méthode $get du provider	nom du service + 'Provider'
module.factory()	Non	Oui	Au premier accès, par appel de la fonction fournie	nom du service + 'Provider'
module.service()	Non	Oui	Au premier accès, en utilisant la fonction fournie comme un constructeur	nom du service + 'Provider'
module.value()	Non	Non	Le service est un objet préexistant	nom du service + 'Provider'
module.constant()	Oui	Non	Le service est un objet préexistant	nom du service

Services natifs

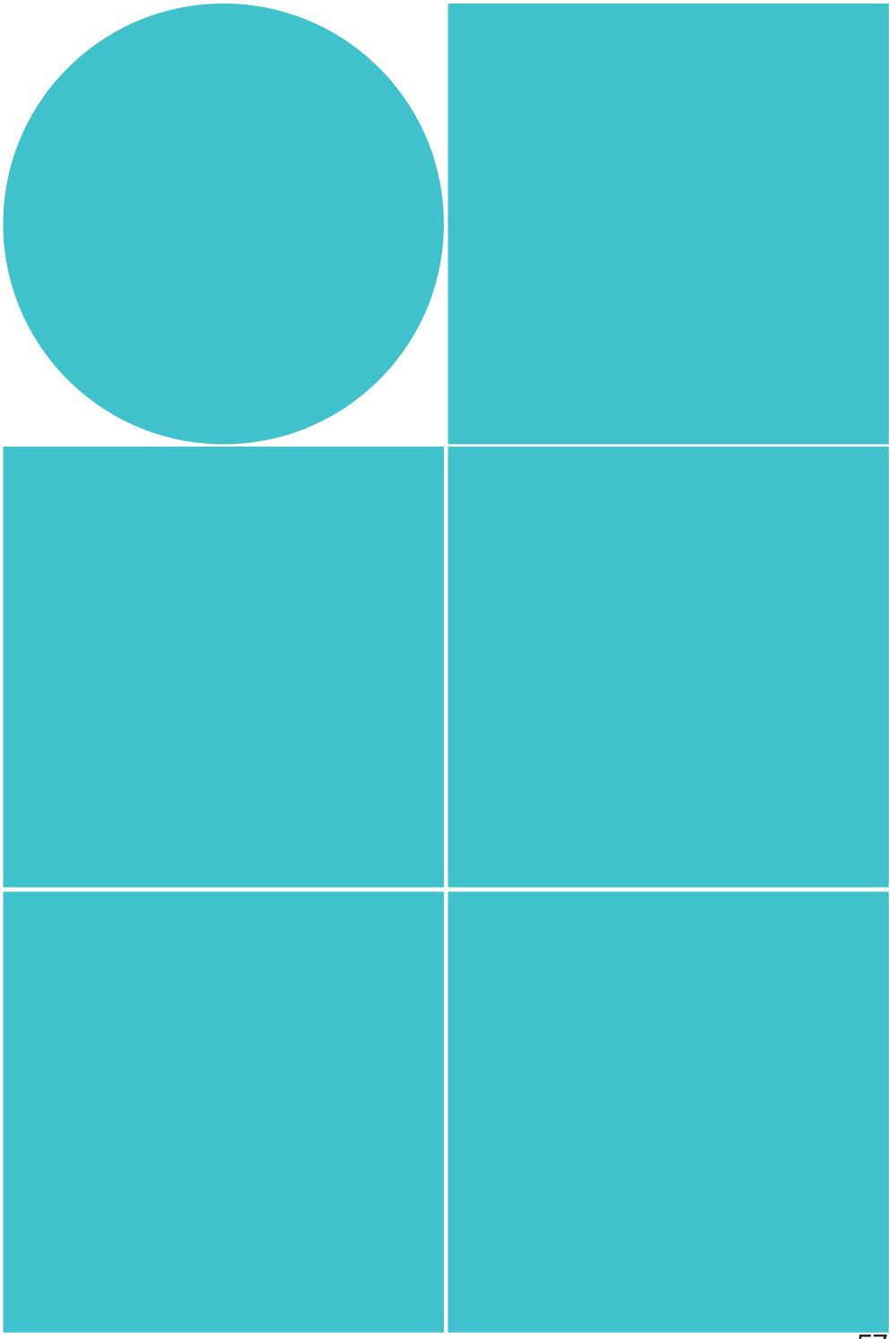
- $anchorScroll
- $animate
- $cacheFactory
- $compile
- $controller
- $document
- $exceptionHandler
- $filter
- $http
- $httpBackend
- $interpolate
- $interval
- $locale
- $location
- $log
- $parse
- $q
- $rootElement
- $rootScope
- $sce
- $sceDelegate
- $templateCache
- $timeout
- $window

15. Promesses.



- **Objet représentant le résultat futur d'une action exécutée de manière asynchrone**
- **Possibilité d'y greffer des callbacks (erreur,success,progress)**
- **reject()**
- **resolve()**
- **then()**
- **$q**
  Service aidant a exécuter des fonctions de manière asynchrone

16. Service $http.



Service utilisée pour faire des appels asynchrones

- **$http(config)**

```
var config = {
    method: 'POST',
    url: ’http://myService.action',
    headers: {
      'Content-Type': undefined
    },
    data: { test: 'test' },
}
```

Service $http

- **Retourne une promesse**
  - then(succesFn,erroFn) : Méthodes appelées avec la réponse
  - success(fn)
  - error(fn)

- **Ou fn est une fonction exécutée avec:**
  - data – {string|Object} – la réponse
  - status – {number} – la code du status HTTP de la response.
  - headers – {function([headerName])} – function pour récupérer les entêtes
  - config – {Object} – l'objet de configuration utilisé pour générer la requête.

```
$http({config}).
    success(function(data, status, headers, config) {
    }).
    error(function(data, status, headers, config) {
  });
};
```

17. Routing – ngRoute.

Utilisation de ngRoute

- **ngRoute est un module permettant à une application de configurer les URLs d'une application AngularJS**

- **$routeProvider**

  - when() : Ajoute une nouvelle route à l'aide des paramètres : **path** et **route**
    - route est un objet permettant de configurer une route
      - à controller
      - à controllerAs
      - à template
      - à templateUrl
      - à resolve
      - à redirectTo
      - à reloadOnSearch

  - otherwise() : Permet de configurer la route par défaut en cas d'échec

- **$routeParams**

  - Permet de récupérer les paramètres courant de l'url

Interagit avec la barre d'adresse du navigateur (lecture/écriture)

  - Basé sur window.location
  - Permet d'observer les changements d'url
  - Ne recharge pas complétement la page

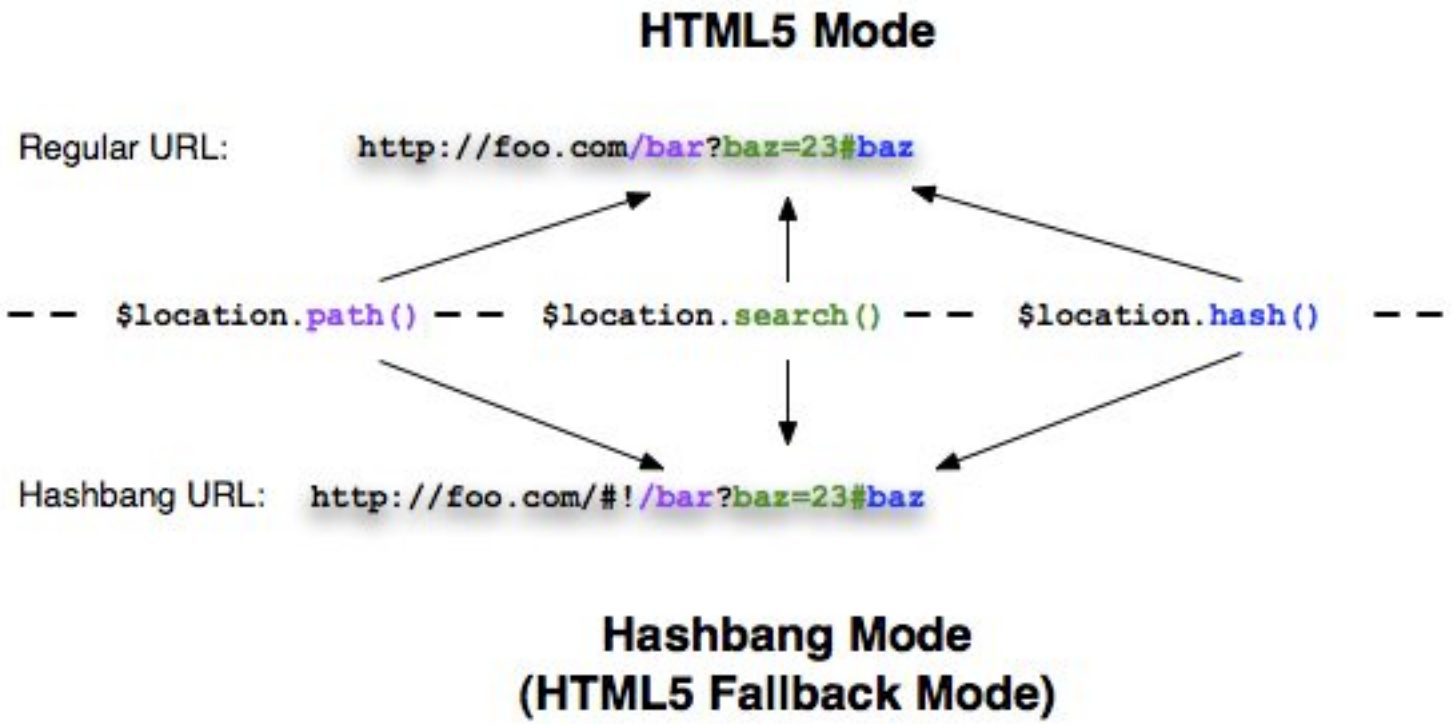
- **$location(config)**

  - html5Mode
  - hashPreffix

  Exemple :
  $locationProvider.html5Mode(true).hashPrefix('!');

- **$location ensemble de méthode**

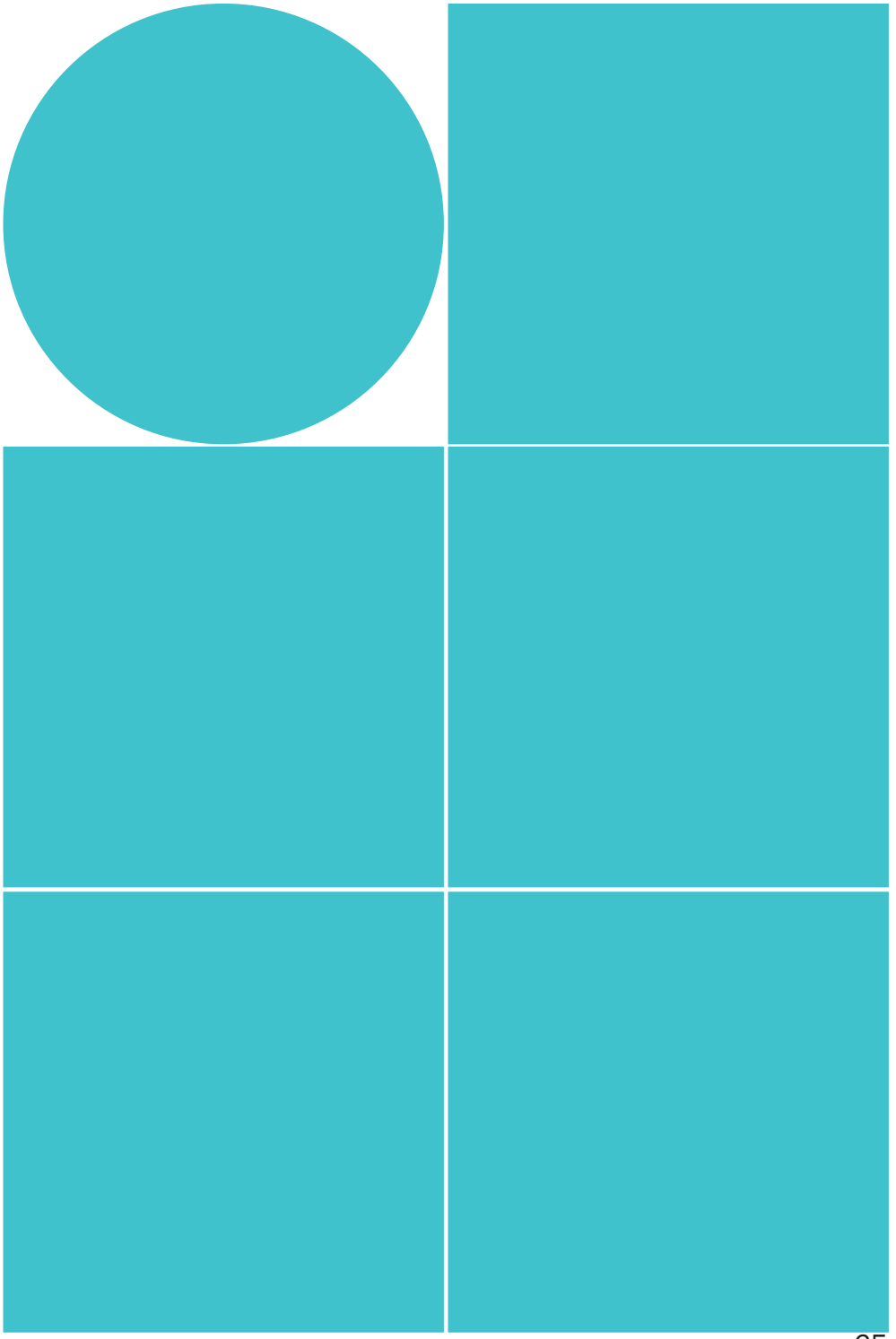
  - **$location.path()**
  - **$location.replace()**
  - **$location.search()**
  - $location.hash()
  - $location.port()
  - $location.protocol()


HTML5 Mode
Regular URL:
http://foo.com/bar?baz=23#baz
$location.path()
$location.search()
$location.hash()
Hashbang URL:
http://foo.com/#!/bar?baz=23#baz
Hashbang Mode
(HTML5 Fallback Mode)

Route et évènements

- **$routeChangeError** **($stateChangeError)**
  - Propagé si une erreur c'est produite lors d'un changement de route
- **$routeChangeStart**
  - Propagé avant un changement de route
- **$routeChangeSuccess**
  - Propagé après la résolution d'une route
- **$routeUpdate**
  - Propagé si le reloadOnSearch=false et que l'on est sur le même contrôleur

18. Modules.



- **Facile à déclarer**
- **Possibilité de packager des modules réutilisables**
- **Les modules peuvent être charger dans n'importe quel ordre**
- **Rend les tests unitaire plus simple et rapide**

Recommandation :

à **Un module pour chaque fonctionnalité**

à **Un module pour chaque composant réutilisable (par exemple pour les directives et les filtres)**

à **Un module pour l'initialisation de l'application**

Modules

```
angular.module('xmpl.service', [])

  .value('greeter', {
    salutation: 'Hello',
    localize: function(localization) {
      this.salutation = localization.salutation;
    },
    greet: function(name) {
      return this.salutation + ' ' + name + '!';
    }
  })

  .value('user', {
    load: function(name) {
      this.name = name;
    }
  });

angular.module('xmpl.directive', []);

angular.module('xmpl.filter', []);

angular.module('xmpl', ['xmpl.service', 'xmpl.directive', 'xmpl.filter'])

  .run(function(greeter, user) {
    // This is effectively part of the main method initialization code
    greeter.localize({
      salutation: 'Bonjour'
    });
    user.load('World');
  })

  .controller('XmplController', function($scope, greeter, user){
    $scope.greeting = greeter.greet(user.name);
  });
```

Modules

- **Un module c' est un collection de blocks qui vont être appliqués durant la phase d'initialisation.**
- **Il y a 2 sorte de blocks dans un modules**
    - Configuration block (injection et configuration des providers)
    - Run block (point d'entrée d'une module post configuration)

```
angular.module('myModule', []).
config(function(injectables) { // provider-injector
  // This is an example of config block.
  // You can have as many of these as you want.
  // You can only inject Providers (not instances)
  // into config blocks.
}).
run(function(injectables) { // instance-injector
  // This is an example of a run block.
  // You can have as many of these as you want.
  // You can only inject instances (not Providers)
  // into run blocks
});
```

Modules

- **Dépendances**

à **Soit le module B requis par le module A. Le block de configuration du module B sera executé avant le block de configuration du module A. Idem pour le block run.**

à **Chaque module ne peut être chargé qu'une seul fois mêle si plusieurs autres modules en sont dépendant.**

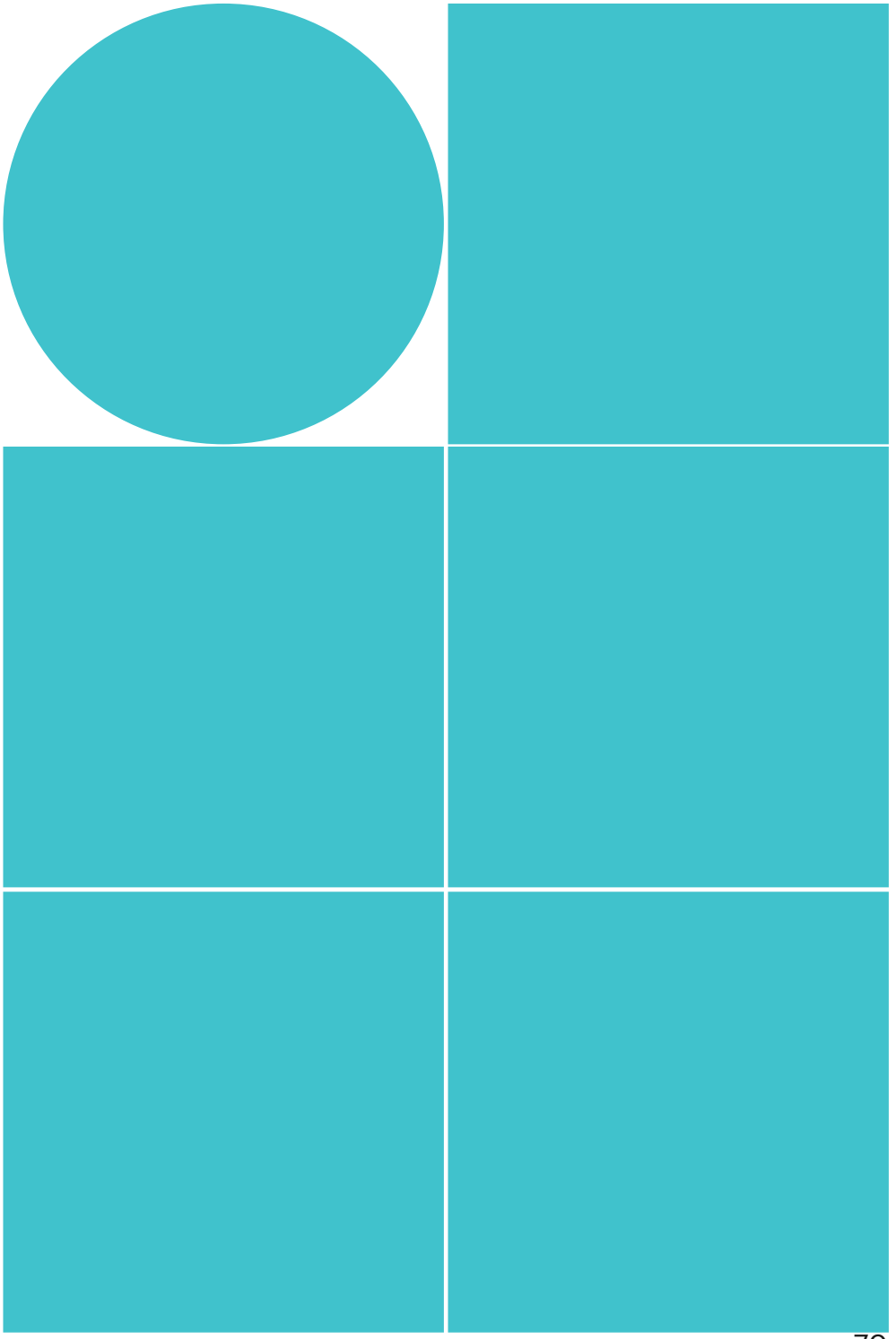
- **Création**

angular.module('myModule', [])

- **Récupération**

angular.module('myModule')

19. Questions.



Liens

https://angularjs.org/

https://github.com/angular/angular-phonecat

https://github.com/angular/angular.js

https://github.com/angular-ui/ui-router

https://github.com/mgonto/restangular

http://www.bennadel.com/

http://jonathancreamer.com/working-with-all-the-different-kinds-of-scopes-in-angular/

https://chrome.google.com/webstore/detail/angularjs-batarang/

ighdmehidhipcmcojjgiloacoafjmpfk

http://ionicframework.com/blog/angularjs-console/

https://www.airpair.com/angularjs/posts/top-10-mistakes-angularjs-developers-make

http://petermorlion.blogspot.fr/2014/11/ngeurope-angular-from-scratch.html