Dependency Injection Containers and Caching

Jiří Matula

Dependency Injection

Passing responsibility of object for initialization of dependent objects.

```php
class DependentClass {

    private $object;

    public __construct(Object $object) {
        $this->object = $object;
    }

}
```

```php
class DependentClass {

    private $object;

    public __construct() {
        $this->object = new Object();
    }

}
```

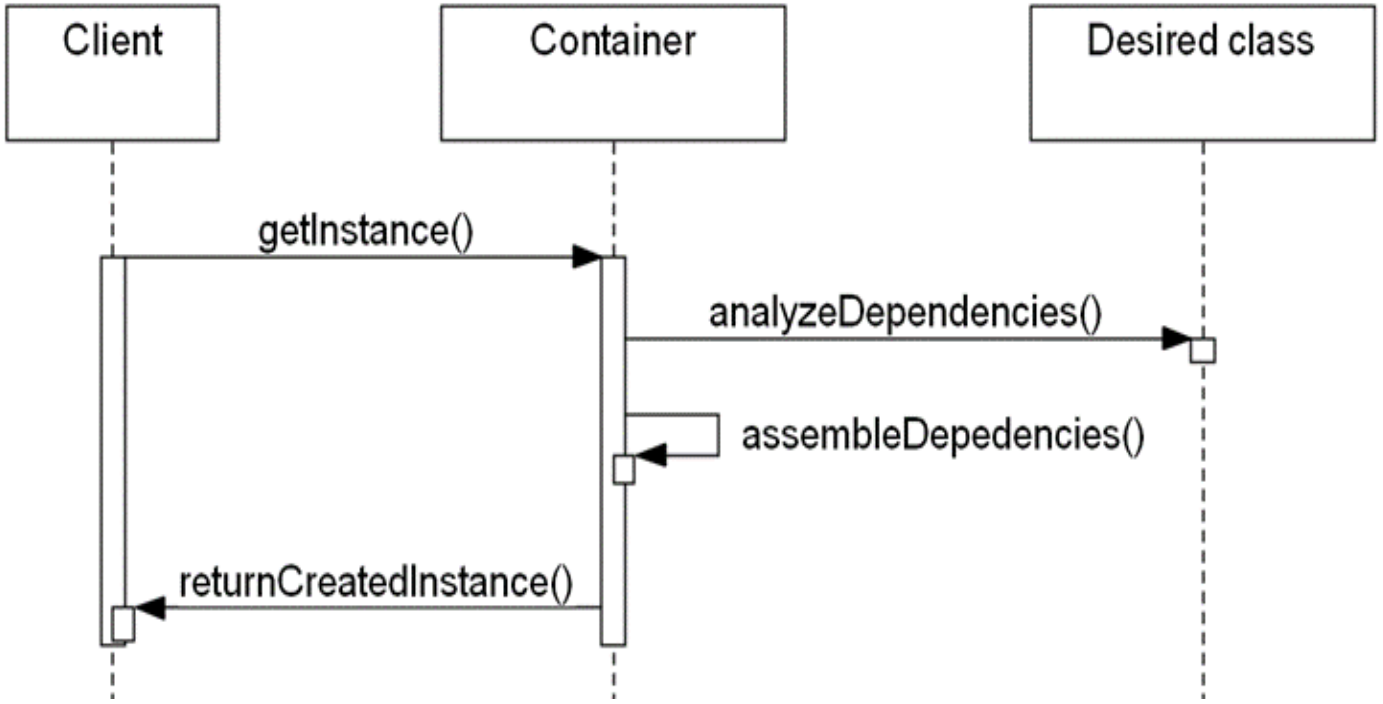
DI Containers Generally

- It facilitates object initializations.
- Increasing popularity of dependency injection.
- They becomes a core part of bigger frameworks and projects (e.g. Symfony, Spring and others).

Examples:

Pimple, PHP-DI, Google Guice.

DI Container Sequential Diagram



Features of DI Containers

- Recursive dependency injection,
- autowiring,
- responsibility for application configuration,
- lazy initialization of objects,
- substitution for some design patterns.

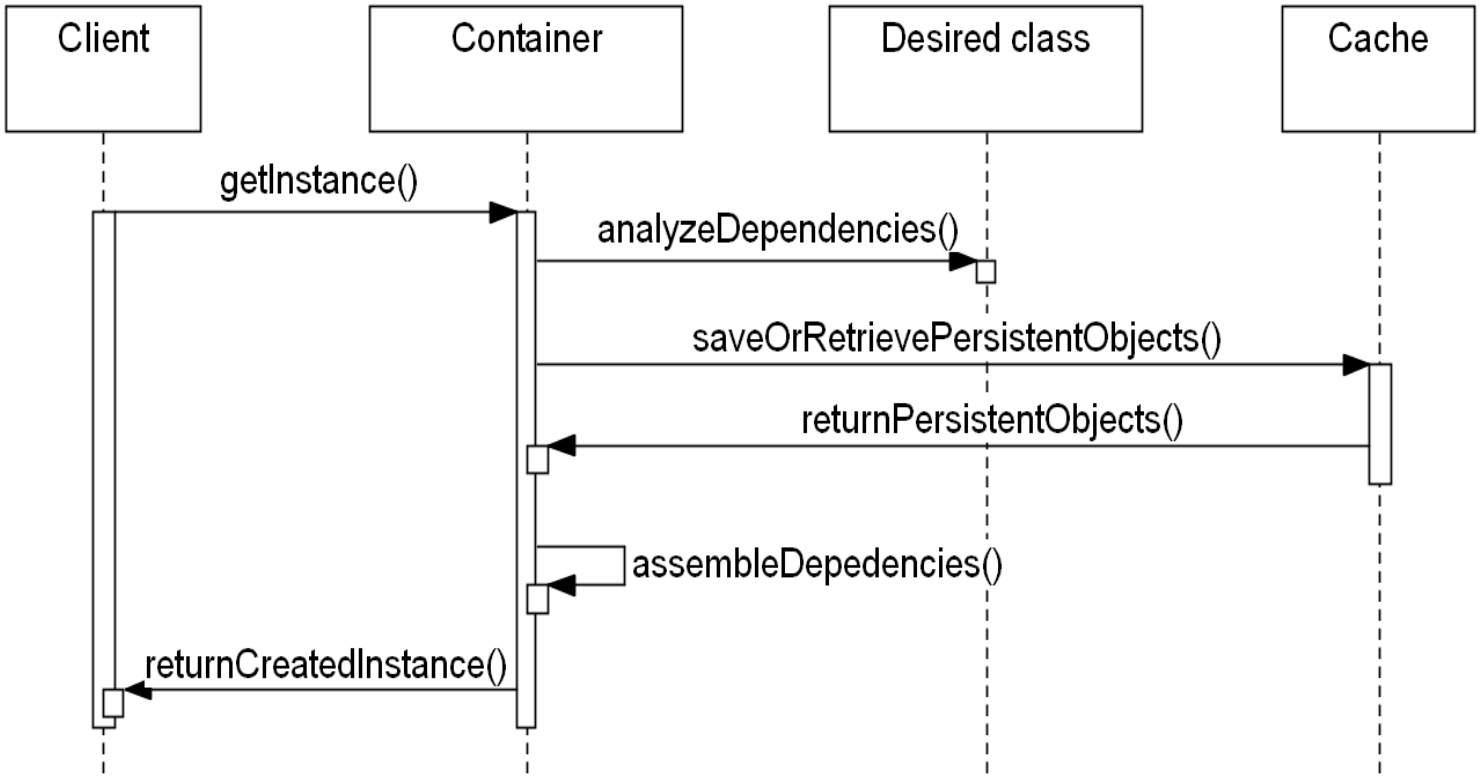
Cache – basic example

```
if (cache_key_exists(Object::class)) {
    $object = cache_fetch(Object::class)
} else {
    $object = new Object();
    cache_store(Object::class, $object)
}
```

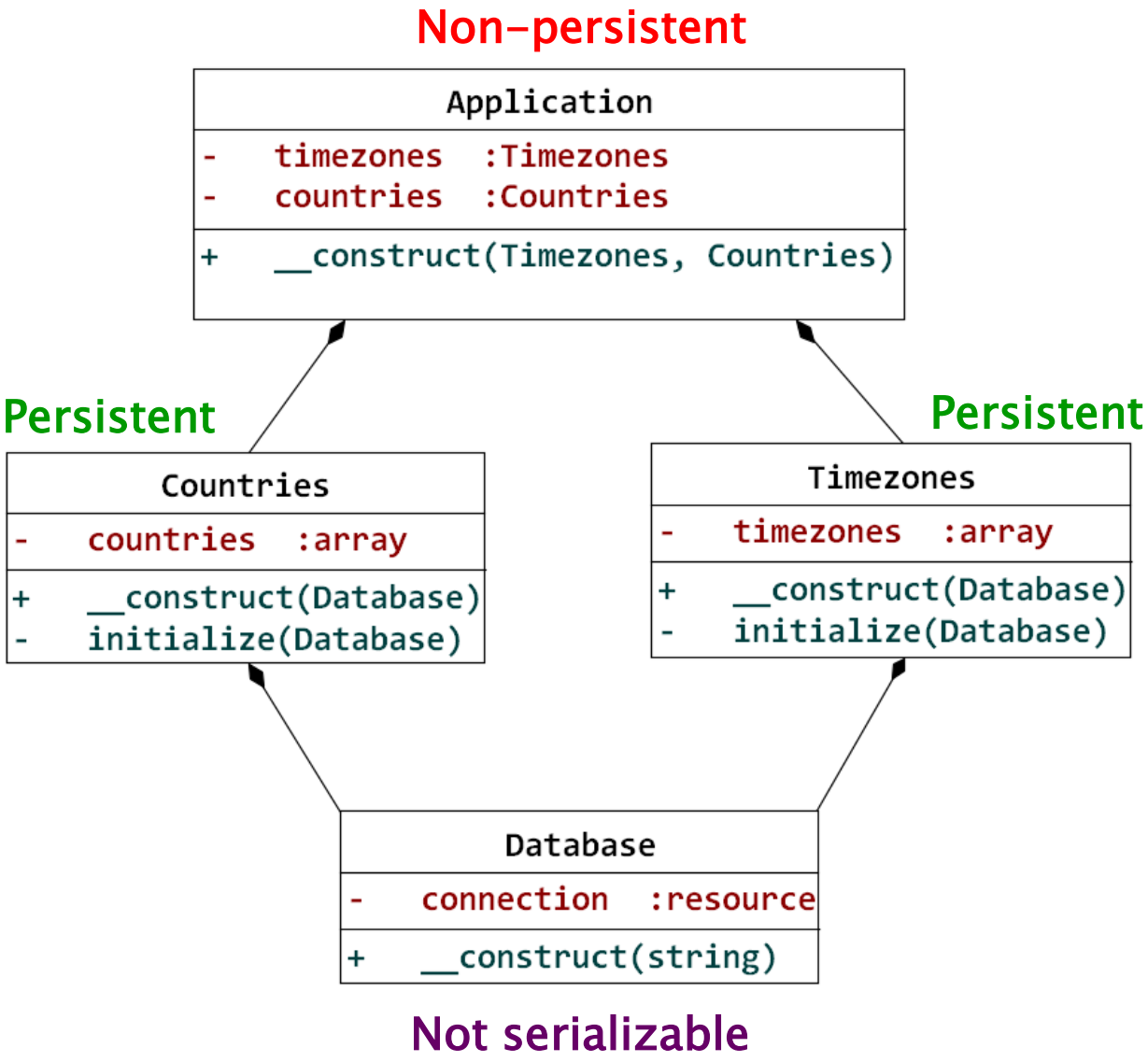
General idea

Let the container fully control lifecycle of class instances regardless the way how they are initialized.

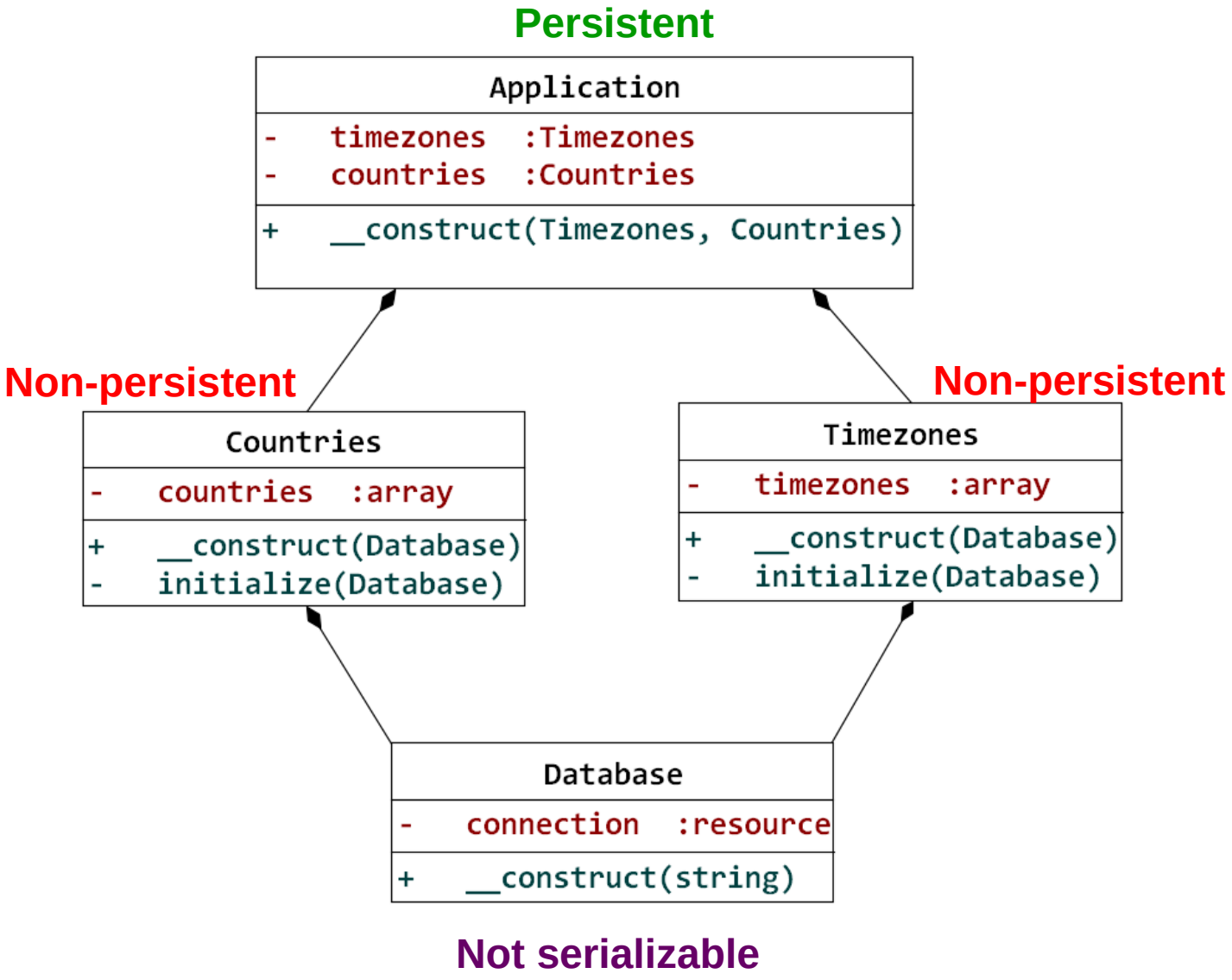
Sequence Diagram of Concept



Initialization process



Initialization process



UNIVERSITAS OSTRAVIENSIS

Discussion

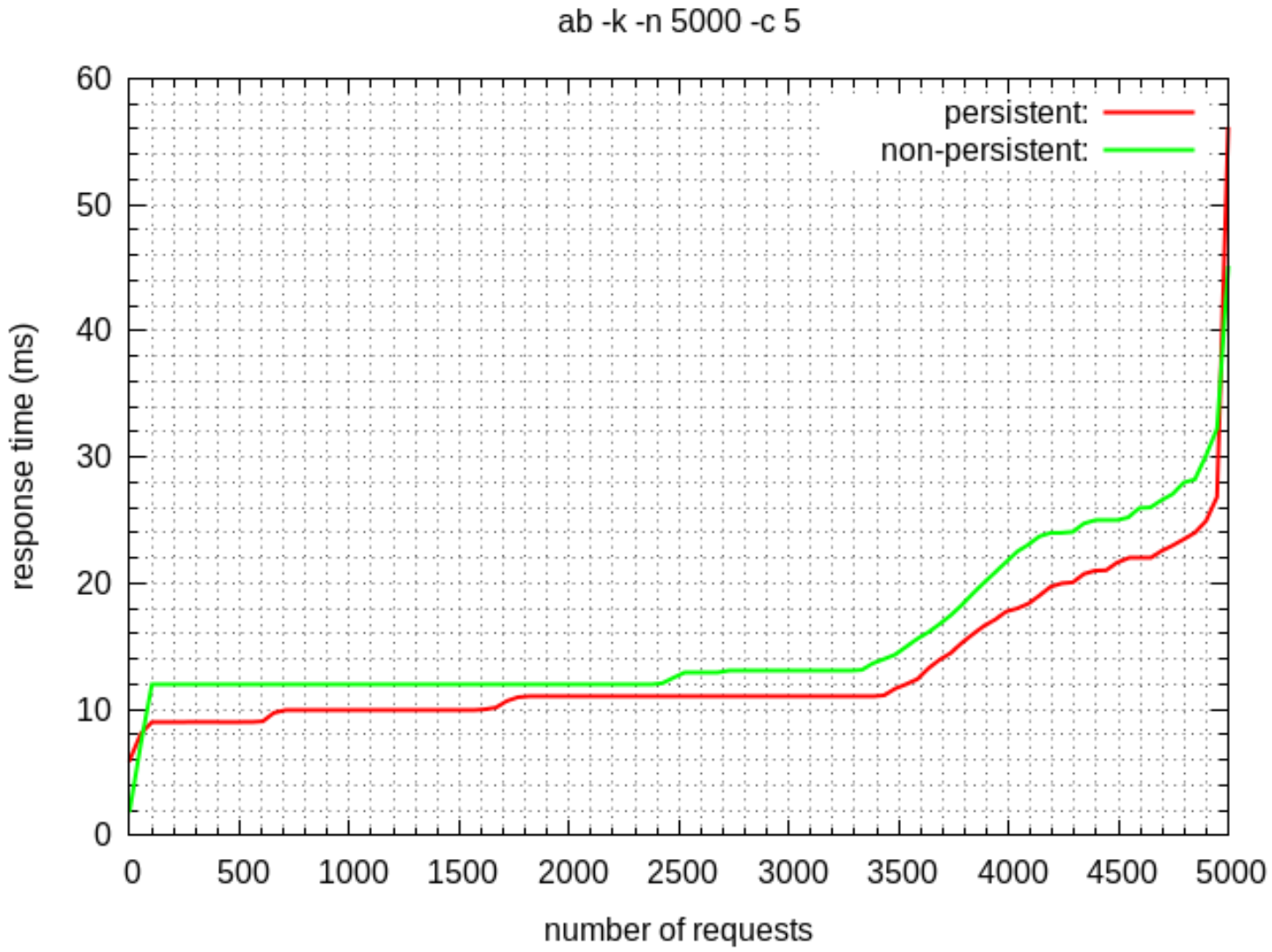
Advantages:

+ Caching of objects is not directly functional part of a class.
+ Improves reusability and testability of classes.
+ Application does not have to be fully initialized with every incoming request.

Disadvantages:

- Persistent class must not contain static and non-serializable class members.
- Developer has to be aware of application architecture, respectively class composition.

Experiment



UNIVERSITAS OSTRAVIENSIS

Experiment

	Non-persistent container	Persistent container
Time taken for tests	15.586 seconds	13.050 seconds
Complete requests	5 000	5 000
Failed request	55 (invalid length)	0
Request per second	320.81	383.15
Time per request (mean)	3.117 milliseconds	2.610 milliseconds
Percentage of the requests served within a certain time (ms)	50 % 13 66 % 13 75 % 18 80 % 22 90 % 25 95 % 27 98 % 30 99 % 32 100 % 45 (longest time)	50 % 11 66 % 11 75 % 15 80 % 18 90 % 22 95 % 23 98 % 25 99 % 27 100 % 56 (longest time)

UNIVERSITAS OSTRAVIENSIS

Summary

- ✓ By passing responsibility of object for construction, testability of classes is improved (mocking objects).
- ✓ Lifecycle of instances is controlled by container, hence initialization and configuration is not up to responsibility of classes.
- ✓ By following principles of DI it makes novice developers team more competitive.

Thank you for your attention.

Mgr. Jiří Matula
University of Ostrava
Faculty of Science
Department of Informatics and Computers
jiri.matula@osu.cz

UNIVERSITAS OSTRAVIENSIS