Concurrent Object Construction in Modern Object-oriented Programming Languages

Viktor Májer[1] Norbert Pataki[2]

[1]EPAM System European Headquarters, Budapest

[2]Dept. Programming Languages and Compilers
Eötvös Loránd University, Budapest, Hungary

Viktor_Majer@epam.com, patakino@elte.hu

ICAI 2014

Contents

Concurrency

- Hardware
  - multicore, manycore CPUs, GPU, FPGA, etc.
- Virtualization technics, clouds
  - xen, kvm
- Operating systems
- Programming languages, libraries
  - Object-oriented software development (C++, Java)
  - Threads, Intel TBB, FastFlow, etc.

Good old joke

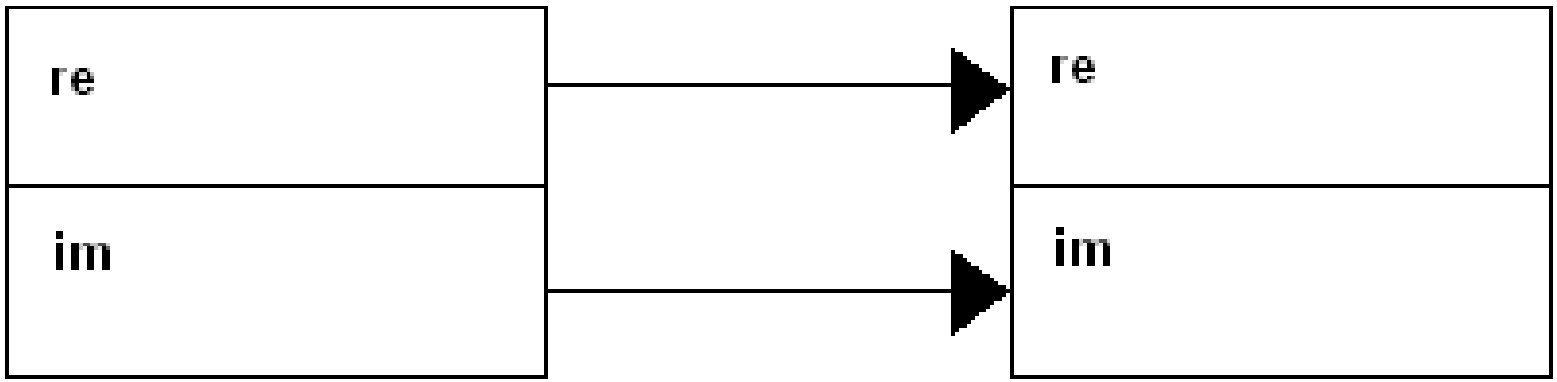
C++ objects

- value semantics

```
class Complex
{
  // ...
};

void f()
{
  Complex n;
  Complex i( 0.0, 1.0 );
  Complex c = i;
}
```

Copy constructors



Memberwise-copy semantics

- Class without copy constructor also can be copied
- Memberwise-copy semantics: sequentially copy the member of the class
- Pass-by-value parameters
- Ideal solution for concurrent execution: threads write to independent memory address, no locking issues.

Our approach

- Customize default copy constructor based on Clang
- Create parallel copy constructor: members copied in a parallel way
- Heuristics, thread pool

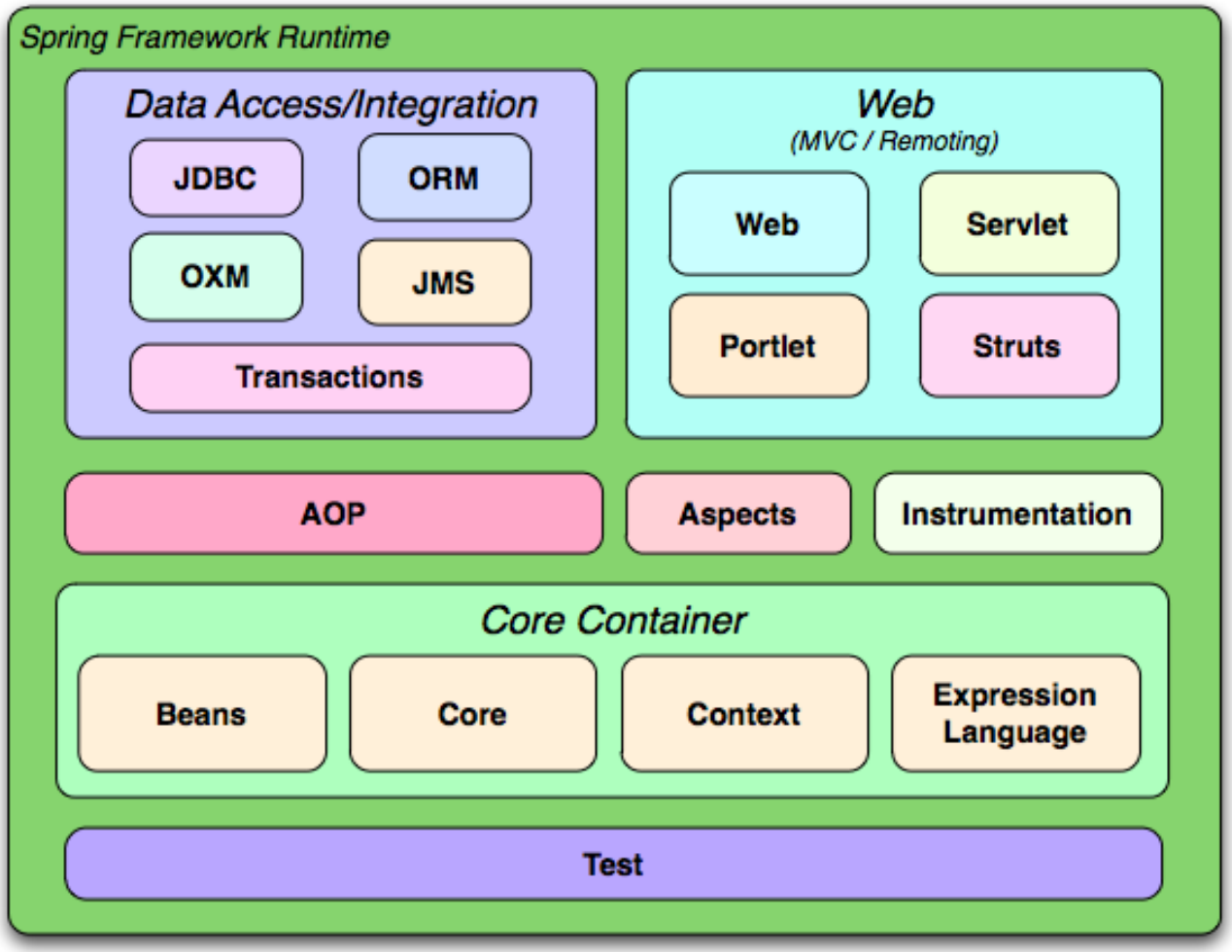
Heuristics

- Is it effective if two double values are copied concurrently?
- Is it effective if two vectors, strings or arrays are copied concurrently?
- Size of vector is not known at compilation-time
- How to group the members if a class has so many of them?
- How many threads are useful for this purpose?
- Thread pool, do not start threads at the beginning of all copy

Introduction to Spring Framework

- Java platform that provides comprehensive infrastructure support for developing applications
- Enables to build applications from POJOs and to apply enterprise services non-invasively to them
- Typical Java applications: consist of objects that collaborate to form the application $\Rightarrow$ objects have dependencies on each other
- Spring's Inversion of Control (IoC) component: formalization of composing components into a fully working application

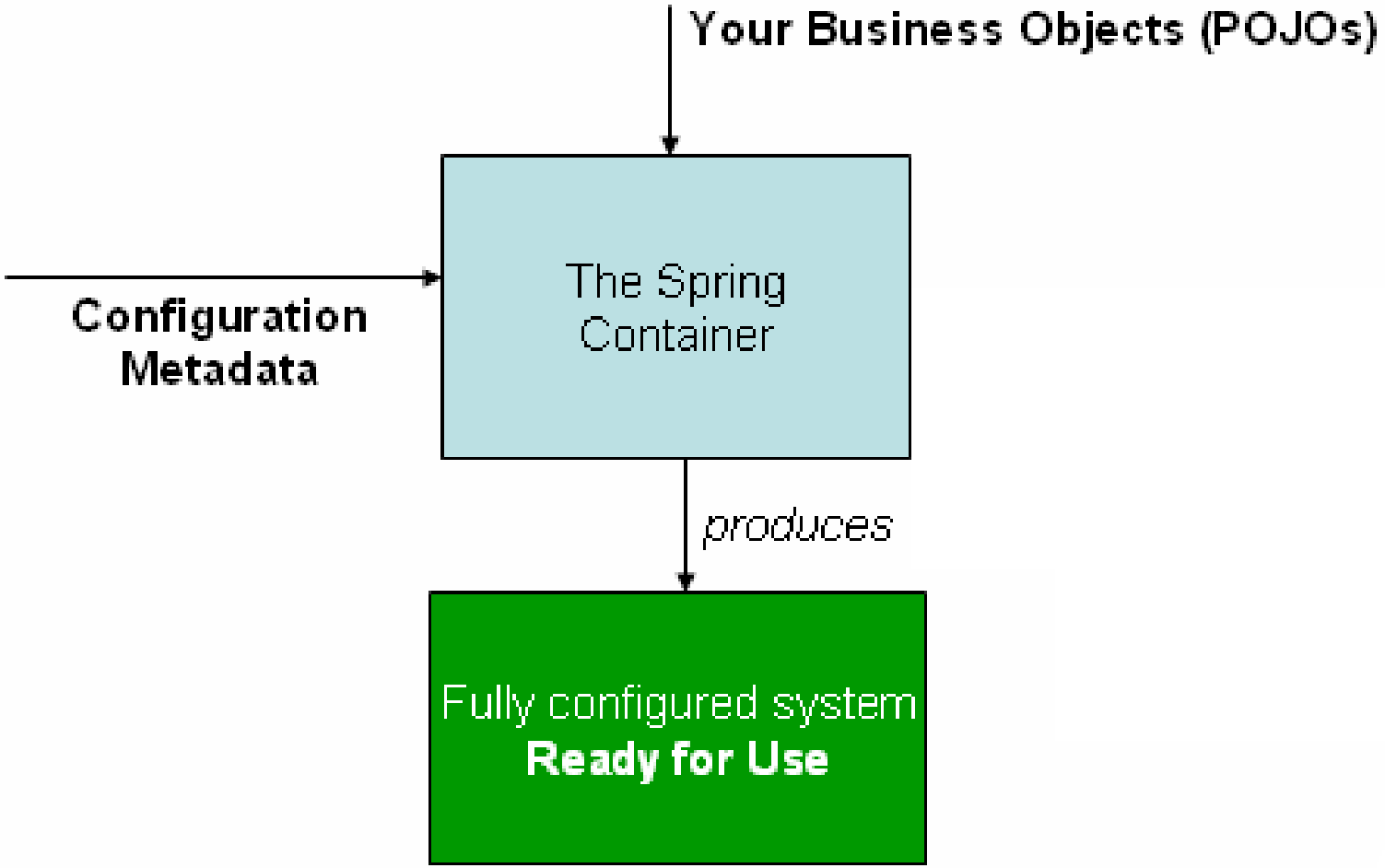
Spring Framework



Spring IoC

- One of Spring's core modules
- Uses Dependency Injection (DI) $\Rightarrow$ results in loose coupling
- Objects represented as beans, defined in Java, XML or by annotations $\Rightarrow$ provide flexibility
- Contexts representing an object graph, can be queried and manipulated even at run-time
- Decouples bean creation, configuration, and program logic
- Organizations use Spring to engineer robust, maintainable applications

Spring IoC


Your Business Objects (POJOs)
Configuration Metadata
The Spring Container
produces
Fully configured system
Ready for Use

Spring example

```
// services.xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans" ...>
    <bean id="petStore"
class="org.springframework.samples.PetStoreServiceImpl">
        <property name="accountDao" ref="accountDao"/>
        <property name="itemDao" ref="itemDao"/>
        <!-- additional collaborators and configuration go here -->
    </bean>
    <!-- more bean definitions for services go here -->
</beans>

// daos.xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans" ...>
    <bean id="accountDao"
class="org.springframework.samples.SqlMapAccountDao">
        <!-- additional collaborators and configuration go here -->
    </bean>

    <bean id="itemDao"
class="org.springframework.samples.SqlMapItemDao" />
</beans>
```

Spring example

```
// application.xml
<beans>
    <import resource="services.xml"/>
    <import resource="daos.xml"/>

    <bean id="bean1" class="..."/>
    <bean id="bean2" class="..."/>
</beans>
```

Using the container

```
// create and configure beans
ApplicationContext context = new ClassPathXmlApplicationContext(new
String[]
                                {"application.xml"});

// retrieve configured instance
PetStoreServiceImpl ps =
context.getBean("petStore",PetStoreServiceImpl.class);

// use configured instance
List<User> userList = ps.getUsernameList();
```

Problems

- Creation & initialization of beans happens in a single thread $\Rightarrow$ slowness
- Slow startup & teardown: can be minutes for an enterprise application
- In case of annotation-based config: startup time also increased by slow component scanning

Our approach

- Provide a way (BeanFactory) to initialize singleton non-lazy beans on startup in parallel
- Involve thread pools for better performance
- Our approach:
  1. Find all bean definitions that don't have any unresolved dependencies
  2. Schedule creation of each bean found in 1. in a separate concurrent task to allow parallel creation
  3. When any of the tasks scheduled in 2. is completed go to 1.

  The algorithm stops when all beans are created

Conclusion

- Modern OO languages are not a natural basis of concurrency
- Billions of objects are created at runtime
- Worth to be concurrent object construction
  - C++ copy constructors: Clang, heuristics
  - Java Spring Framework concurrent bean initialization