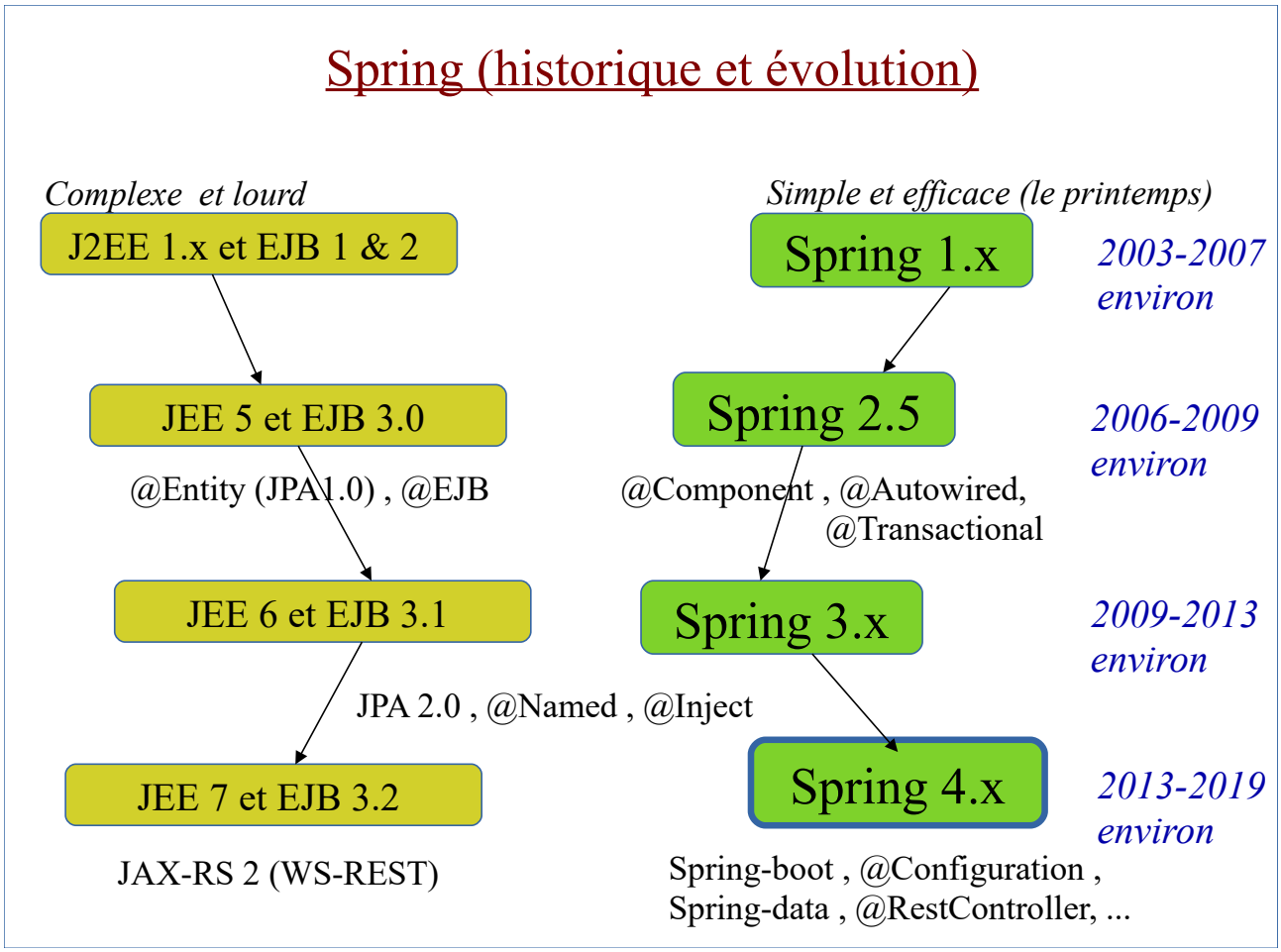
1. Configurations Spring – vue d'ensemble

1.1. Historique et évolution

Versions de Spring	*Possibilités au niveau de la configuration*
Depuis Spring 1.x	Configuration entièrement **XML** (avec entête DTD) **<bean >**
Depuis Spring 2.0	Configuration **XML** (avec entête XSD) + .properties
Depuis Spring 2.5	**Annotations spécifiques à Spring (@Component , @Autowired, ...)**
Depuis Spring 3.0	Compatibilité avec annotations DI (@Inject , @Named)
Depuis Spring 4.0	**Java Config (@Configuration , …)** et Spring boot (avec ou sans @EnableAutoConfiguration)



1.2. Avantages et inconvénients de chaque mode de configuration

Mode de config	Avantages	Inconvénients
XML	-Très explicite - Assez centralisé tout en étant flexible (import) . - utilisation possible de fichiers annexes ".properties	-Verbeux , plus à la mode - à maintenir / ajuster (si refactoring) - délicat (oblige à être très rigoureux "minuscules / majuscules" , noms des packages , namespaces XML , …)
Annotations au sein des composants (@Autowired, ...)	- très rapide / efficace - suffisament flexible (component-scan selon packages , @Qualifier , …) - réajustement automatique en cas de refactoring (sauf component-scan) .	- configuration dispersée dans le code de plein de composants - pour nos composants seulement (avec code source)
Classes de configuration "java" (@Configuration , ...)	-Très explicite - Assez centralisé tout en étant flexible (@Import) . - Auto completion java et détection des incompatibilités (types , configurations non prévues, …) - utilisation possible de fichiers annexes ".properties" pour les paramètres amenés à changer - à la mode ("hype")	- nécessite une compilation de la configuration java (heureusement souvent automatisée par maven ou autre)

Spring (vue d'ensemble sur formats de configuration)

Anciens projets (2004-2008 environ) → ***Configuration entièrement XML*** (<bean …><property ...> …) et souvent accompagné de hibernate (.hbm.xml)

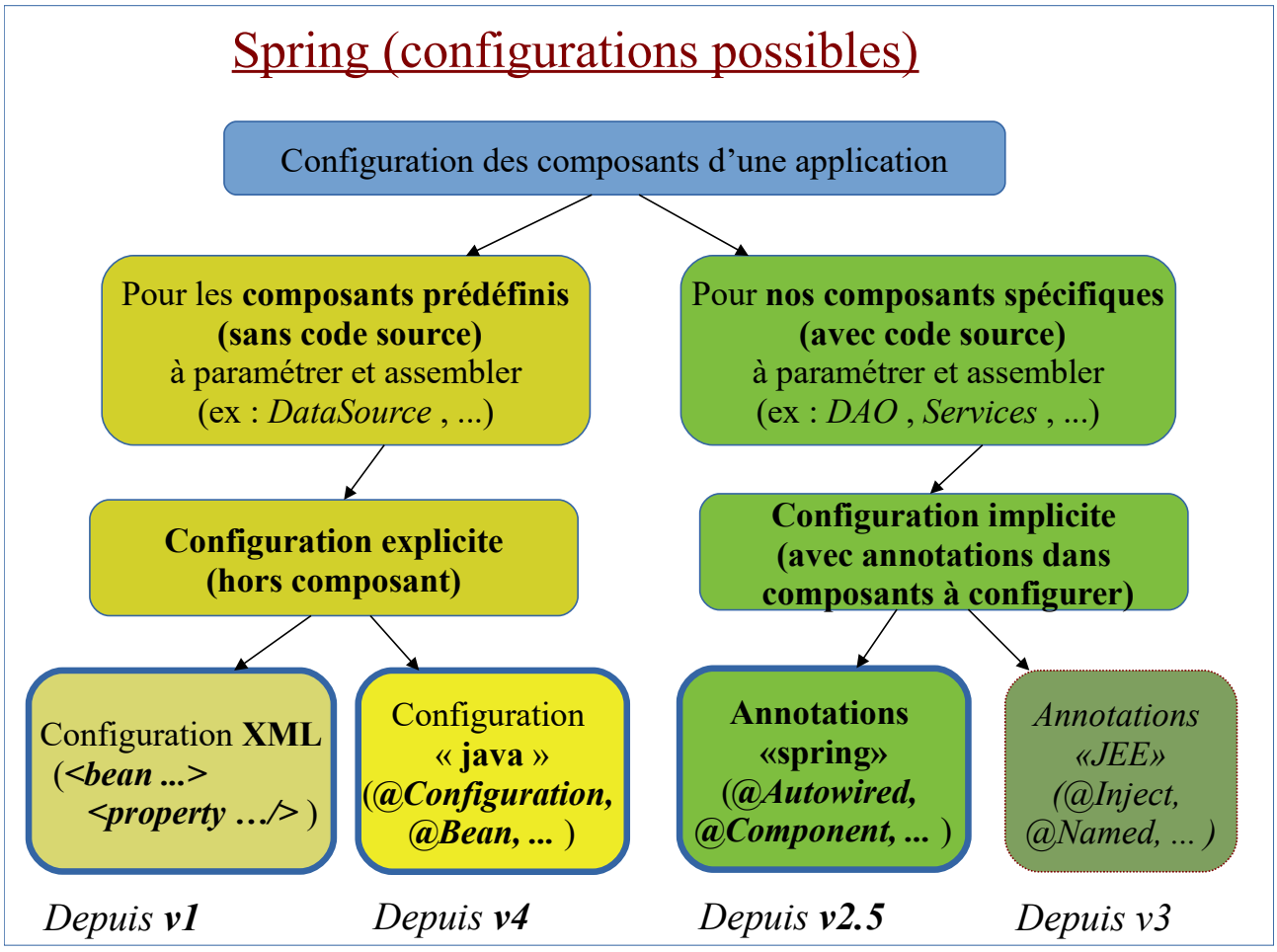
Projets commencés dans les années 2009-2014 environ → ***Configuration mixte XML*** + ***annotations*** (@Autowired , @Component) et souvent accompagné de JPA (@Entity , @Id)

(proche standard JEE / @Inject , ...)

Projets récents (depuis 2015 environ) → ***Configuration mixte javaConfig*** *(@Configuration, ...)* + ***annotations*** *(@Autowired, ...) ou éventuellement* ***auto-configuration*** *avec* ***application.properties*** et souvent avec ***Spring-boot*** , Spring-data , Spring-security et Spring-mvc (pour WS-REST)

1.3. Complémentarité nécessaire / configuration mixte

- **Les annotations @Component , @Autowired , ….** sont très pratiques pour configurer des relations entre composants (injection de dépendances) mais elles **ne peuvent être utilisées qu'au niveau de nos propres composants** (car il faut pouvoir un contrôle total sur le code source).
- Une **configuration XML (classique) ou bien** une **configuration "java config" (moderne) permet de configurer des composants génériques** (ex : DataSource , TransactionManager , ….) **dont on ne dispose pas du code source** .
- Dans tous les cas, il est éventuellement possible de s'appuyer sur des **fichiers annexes** au format **".properties"** pour simplifier l'édition de quelques paramètres clefs succeptibles de changer (ex : url JDBC , username, password , …)



1.4. Démarrages possibles depuis spring 2.5

Depuis méthode **main**() dans une application « standalone »	**ApplicationContext** ***springContext*** = **new ClassPathXmlApplicationContext("/*context.xml*") ;** **Cxy c = (Cxy)** ***springContext*****.getBean("***idBeanXy***"); *//ou bien c = springContext.getBean(Cxy.class);***
Depuis **test unitaire** (Junit + spring-test)	**@RunWith(SpringJUnit4ClassRunner.class)** **@ContextConfiguration(locations={"/context.xml"})** **public class TestCxy {** **@Autowired** **private Cxy c ;** *//+ méthodes prefixées par @Test* **}**
Depuis « listener web » ***(au démarrage d'une application web(.war) dans tomcat ou autre)***	<context-param> *<!-- dans WEB_INF/web.xml -->* **<param-name>*contextConfigLocation*</param-name>** **<param-value>classpath:/context.*xml*</param-value>** </context-param> **<listener><listener-class>** ***org.springframework.web.context.ContextLoaderListener*** **</listener-class></listener>** **----------** **... ctx = WebApplicationContextUtils .getWebApplicationContext(*application* ou *servletContext*) ; ... ctx.getBean(...) ; *//dans servlet ou jsp***

1.5. Variantes de démarrages possibles depuis spring 4

Depuis méthode **main**() dans une application « standalone »	***ApplicationContext springContext*** = **new AnnotationConfigApplicationContext(MyAppConfig.class, ConfigSupplementaire.class) ;** **Cxy c = (Cxy)** ***springContext*****.getBean("***idBeanXy***"); *//ou bien c = springContext.getBean(Cxy.class);***
Depuis **test unitaire** (Junit + spring-test)	**@RunWith (SpringJUnit4ClassRunner.class)** **@ContextConfiguration(classes={MyAppConfig.class})** **public class TestCxy {** **@Autowired** **private Cxy c ;** *//+ méthodes prefixées par @Test* **}**

Depuis « listener web »
(au démarrage d'une application web(.war) dans tomcat ou autre)
	class MyWebApplicationInitializer implements **WebApplicationInitializer** {
	public void **onStartup** (.. ***servletContext***)...{
	WebApplicationContext *context* = **new AnnotationConfigWebApplicationContext** ();
	context.**register** (***MyWebAppConfig.class***);
	servletContext .**addListener** (new **ContextLoaderListener** (***context***));
	//... }}

1.6. Configuration structurée (properties , import , profiles)

Spring (paramétrages indirects dans fichiers ".properties")

Quelque soit la version de Spring, en partant d'une configuration globale explicite ordinaire (xml/bean ou bien java/@Configuration) , il est possible de récupérer certaines valeurs variables (de paramètres clefs) dans un fichier annexe au format **".properties"**
Ceci s'effectue techniquement via "*PropertySourcesPlaceholderConfigurer*" ou un équivalent .

Configuration principale
(context.xml ou bien MyAppConfig.class)
avec
${database.url} ,
${database.username},
....

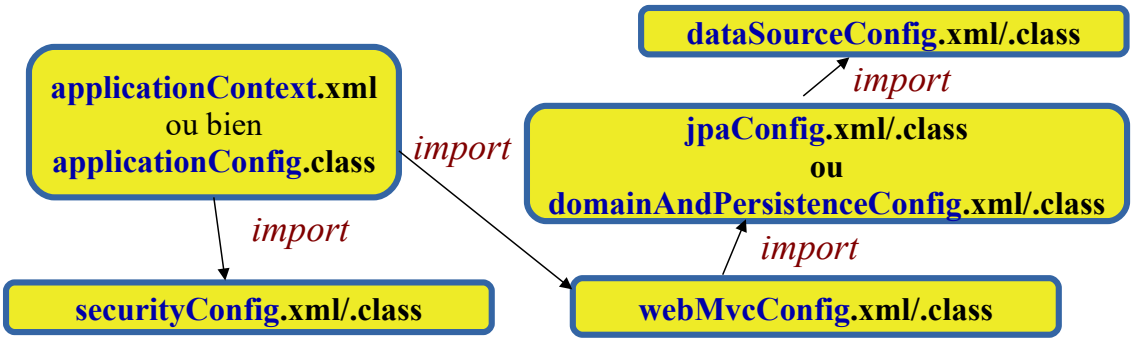
myApp.properties

database.url=jdbc:mysql://...
database.username=user1
database.password=pwd1
....

configuration technique et stable (quelquefois complexe)

sous configuration (indirecte) potentiellement variable et facile à modifier (claire, simple) .

Spring (Configuration structurée via "**import**")



Profiles (Variantes de configurations) depuis Spring4

@Profile({"!test"})
ou bien
@Profile({*"jta","test"*})
au dessus de **variantes de @Bean** dans **@Configuration**

context.getEnvironment().**setActiveProfiles**(...) ;
ou bien
springBootApp.**setAdditionalProfiles**(...);
ou bien
@ActiveProfiles(profiles = *{"test" , "jta"}*)
au dessus d'une classe de test (@RunWith, ...)

1.7. Spring boot et auto-configuration (depuis v4)