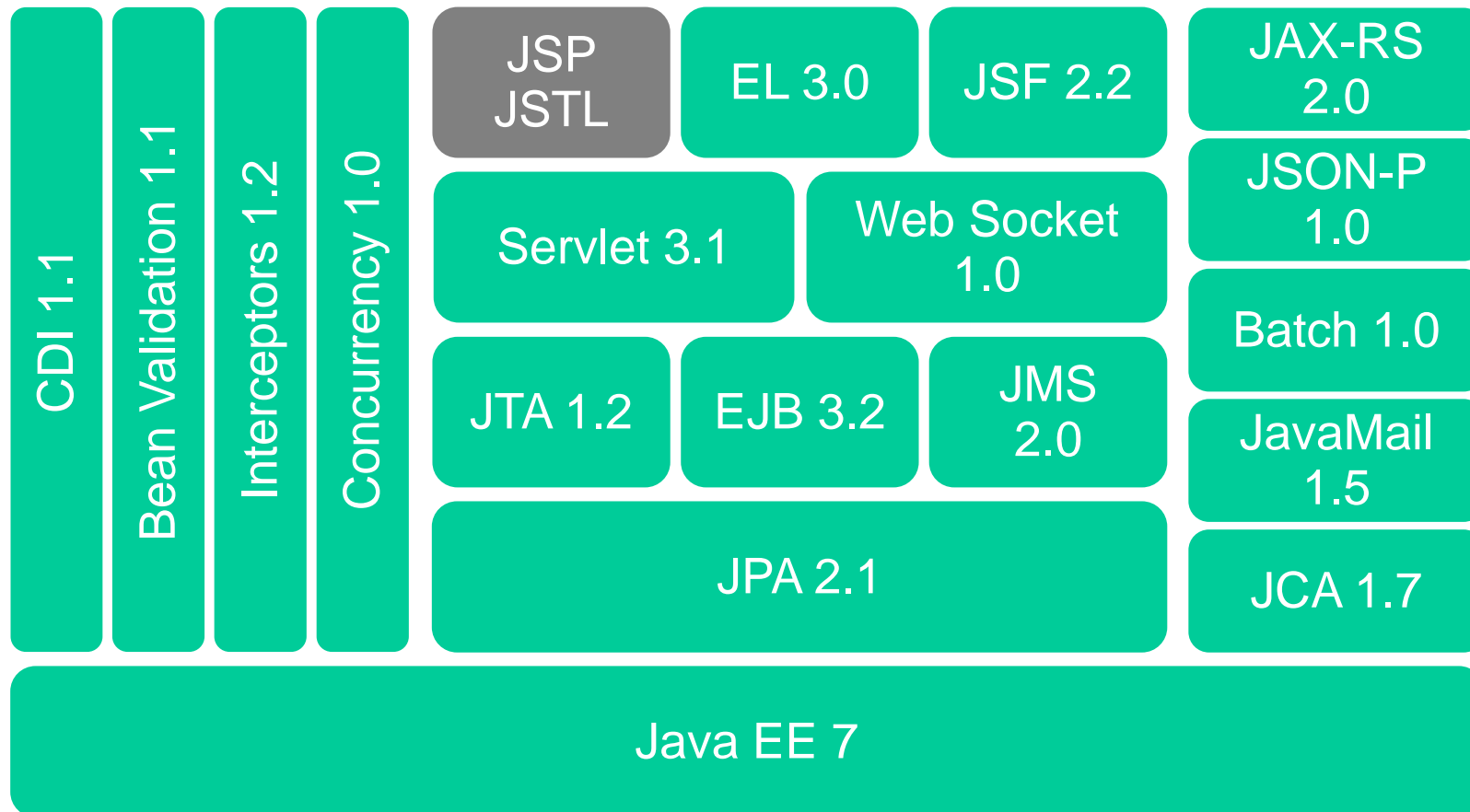


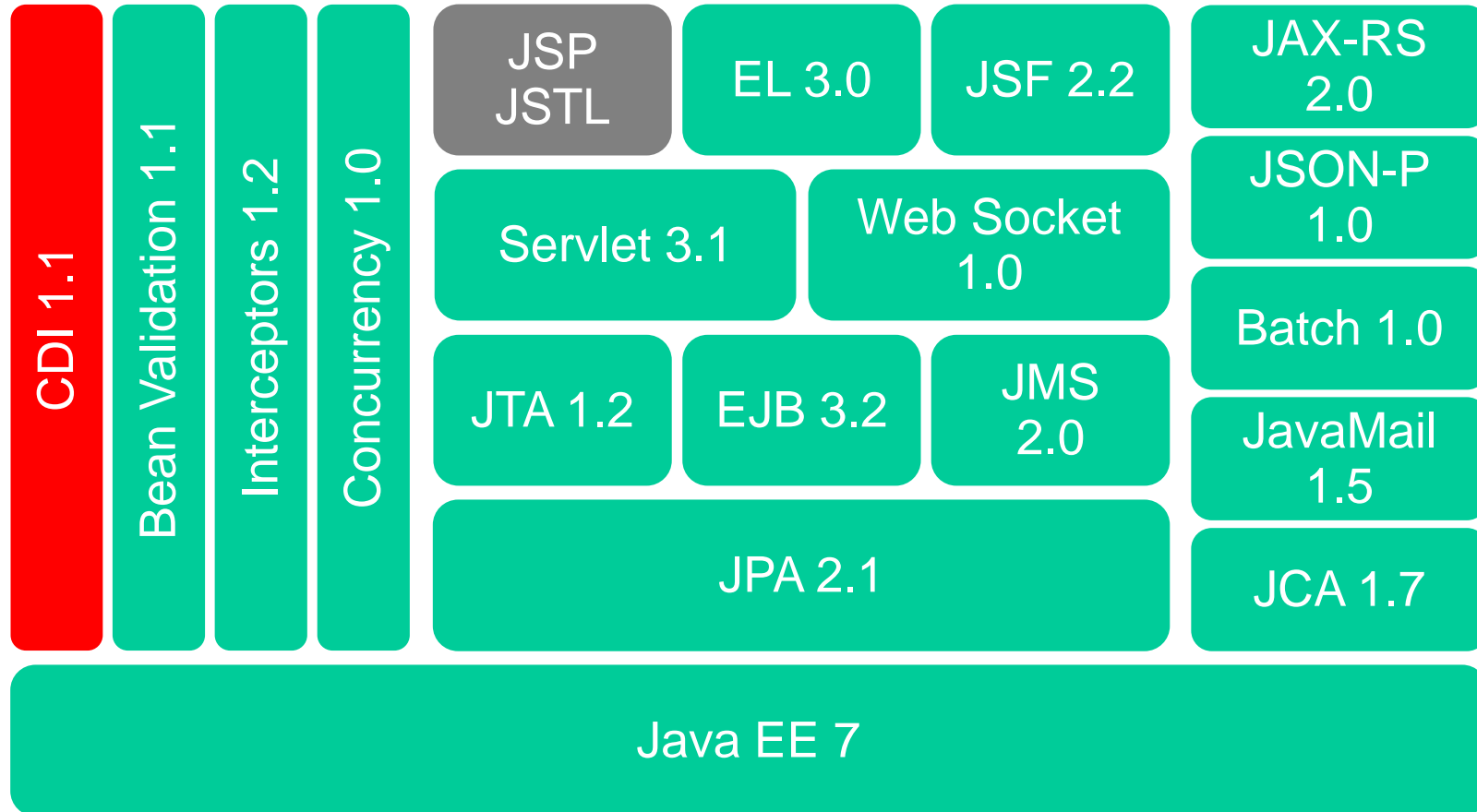
50 best features of Java EE 7 in 50 minutes

Markus Eisele, @myfear
Developer Advocate
September, 2014

#NN: <spec>: <feature>



CDI 1.1 (JSR 346)



#01: CDI: Default enabling

Finer scanning control

Possible values: `all`, `annotated`, `none`

`all` behaves like in Java EE 6 (default if not set)

```
<beans ... version="1.1" bean-discovery-mode="all">
  <alternatives>
    <class>org.agoncal.book.MockGenerator</class>
  </alternatives>
</beans>
```

#02: CDI: @Vetoed

Veto the processing of the class or package

@Vetoed

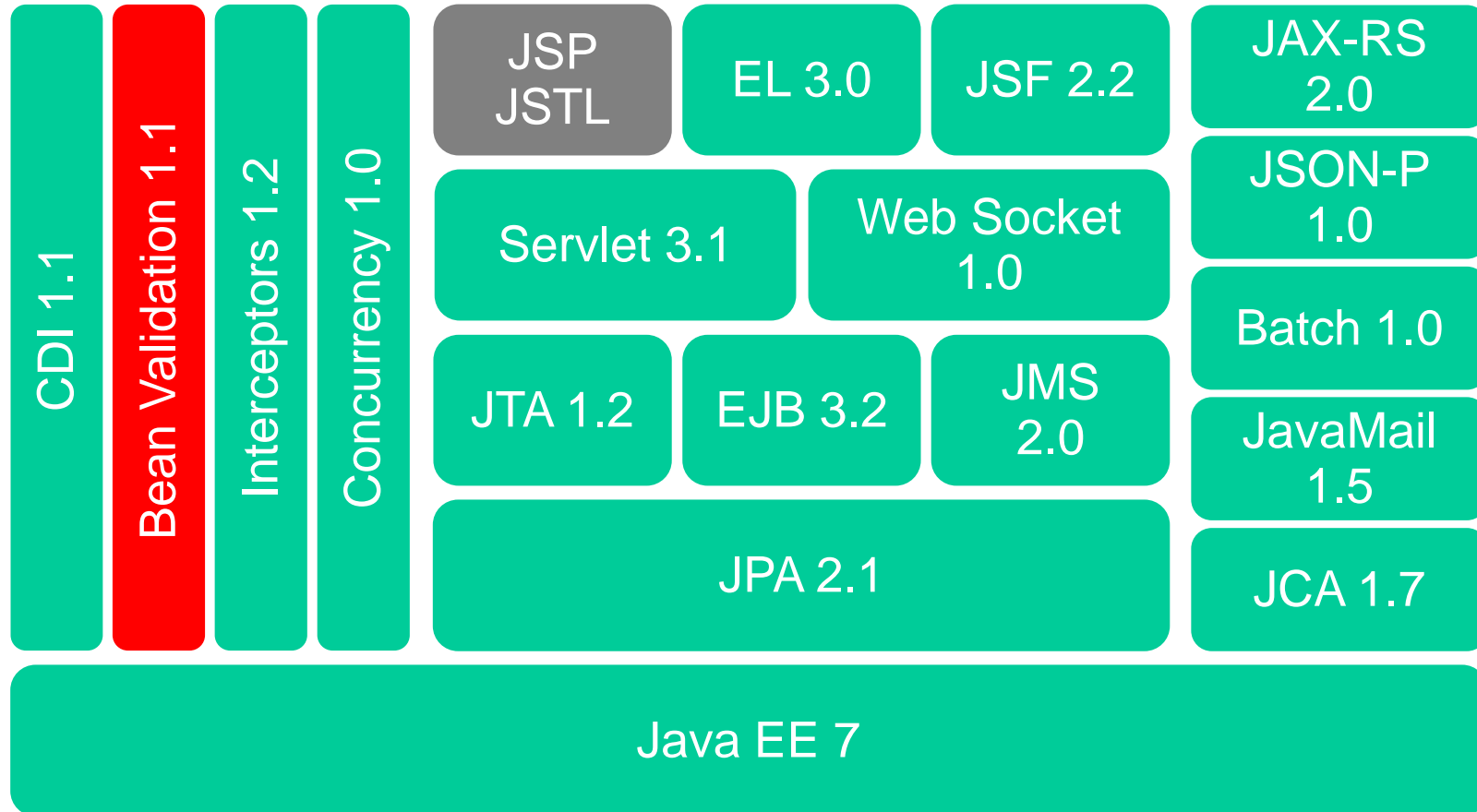
```
public class NonProcessedBean {  
    ...  
}
```

package-info.java

@Vetoed

```
package com.non.processed.package;
```

Bean Validation 1.1 (JSR 349)

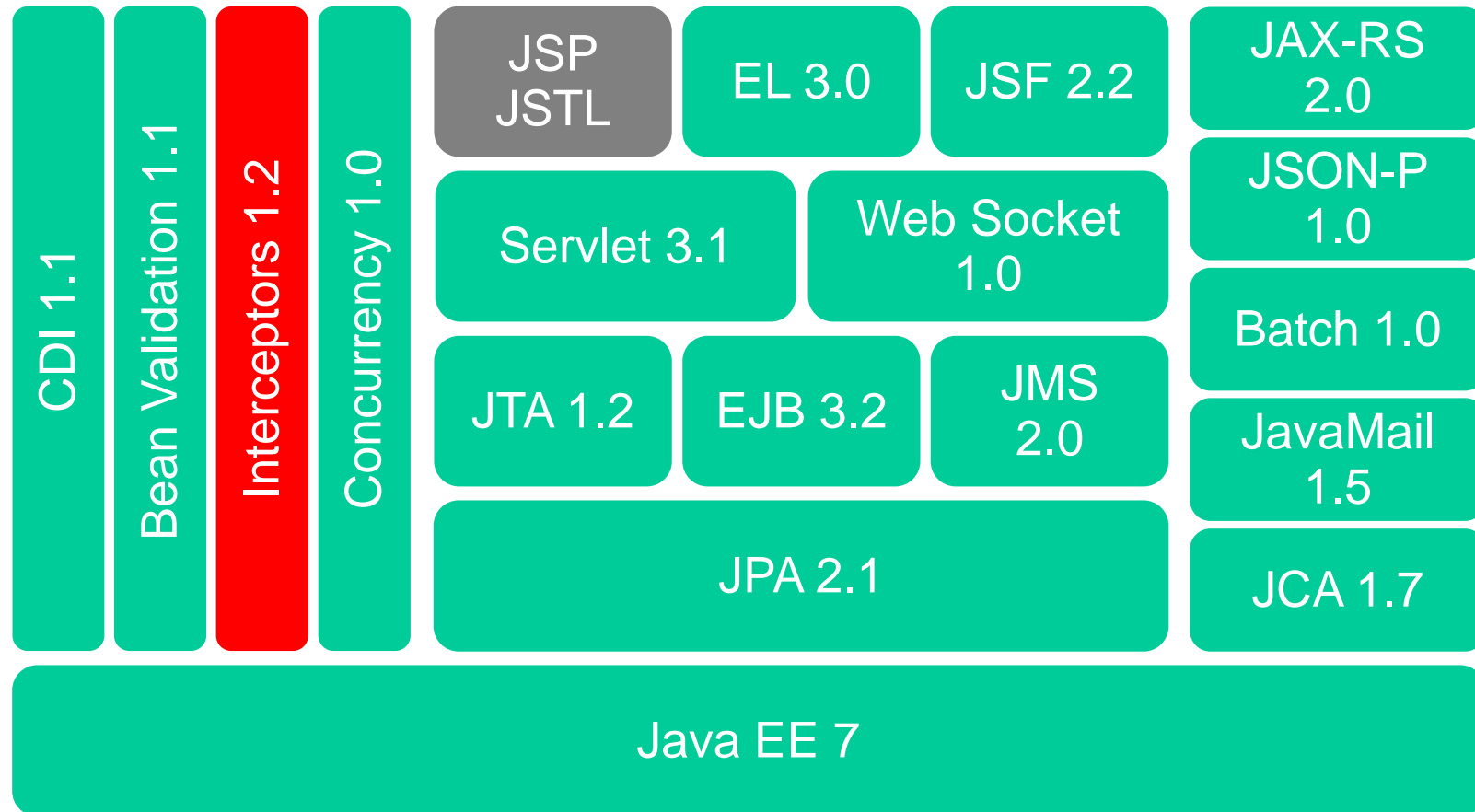


#03: Bean Validation: Method validation

Pre/post conditions on method and constructors

```
public class CardValidator {  
  
    public CardValidator(@NotNull Algorithm algorithm) {  
        this.algorithm = algorithm;  
    }  
  
    @AssertTrue  
    public Boolean validate(@NotNull CreditCard creditCard) {  
        return algorithm.validate(creditCard.getNumber());  
    }  
}
```


Interceptors 1.2 (JSR 318)



#04: Interceptors: AroundConstruct

Interceptor associated with a constructor

```
public class LoggingInterceptor {  
    @AroundConstruct  
    private void init(InvocationContext ic) throws  
Exception {  
        logger.fine("Entering constructor");  
        ic.proceed();  
        logger.fine("Exiting constructor");  
    }  
  
    @AroundInvoke  
    public Object logMethod(InvocationContext ic) ... {  
        // ...  
    }  
}
```

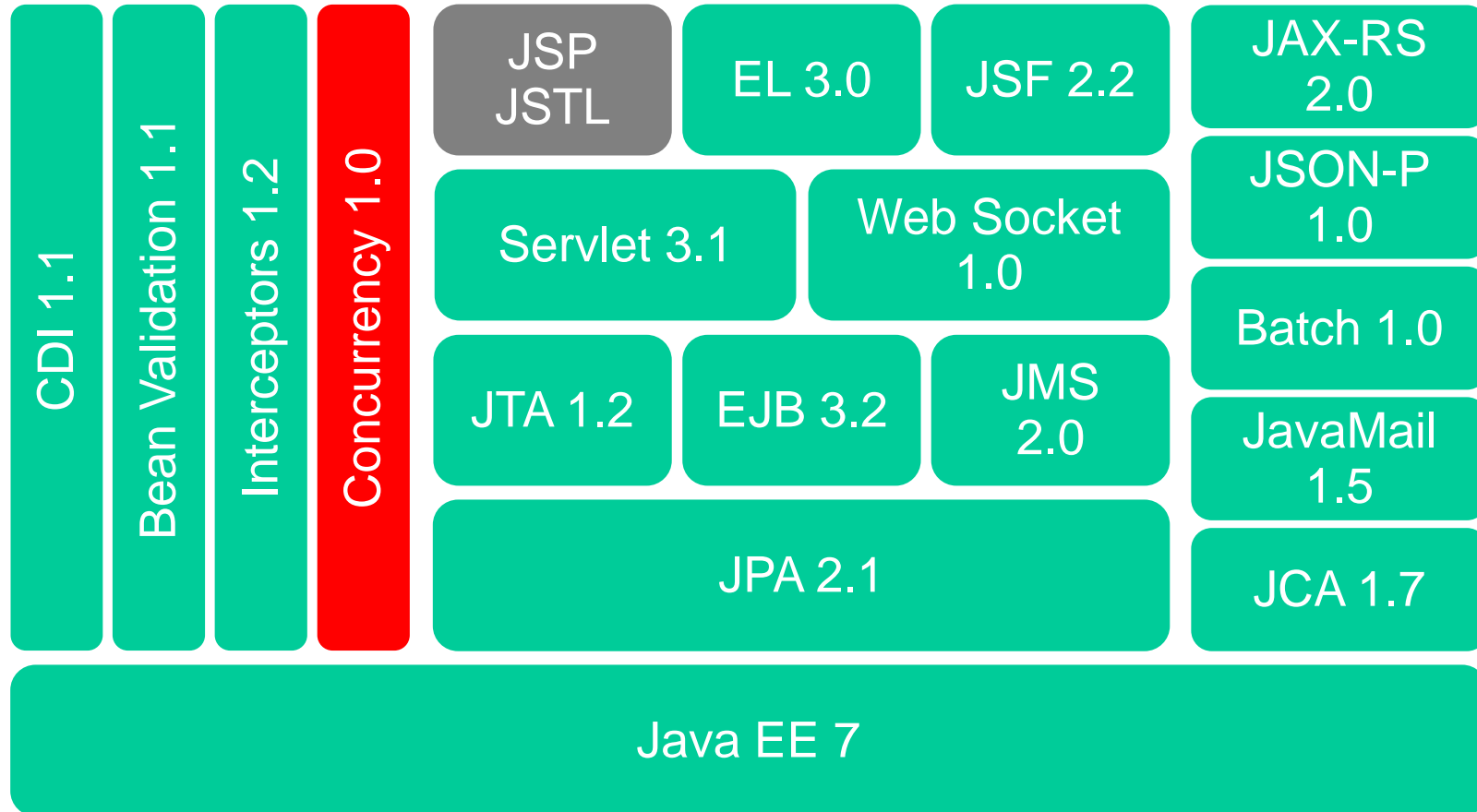
#05: Interceptors: @Priority

Prioritizing interceptor bindings

```
PLATFORM_BEFORE (0), LIBRARY_BEFORE (1000),  
APPLICATION (2000), LIBRARY_AFTER (3000),  
PLATFORM_AFTER (4000)
```

```
@Interceptor  
@Loggable  
@Priority(Interceptor.Priority.LIBRARY_BEFORE+10)  
public class LoggingInterceptor {  
  
    @AroundInvoke  
    ...  
}
```

Concurrency utilities 1.0 (JSR 236)



#06: Concurrency: ManagedExecutor

User threads in Java EE applications

Support simple and advance concurrency design patterns

Extend Concurrency Utilities API from Java SE (JSR 166y)

```
java.util.concurrent package
```

#06: Concurrency: ManagedExecutor

Default ManagedExectuor

```
@Resource
ManagedExecutorService executor;

ManagedExecutorService executor = (ManagedExecutorService)
ctx
    .lookup("java:comp/DefaultManagedExecutorService");
```

#06: Concurrency: ManagedExecutor

Specify in web.xml

```
<web-app ... version="3.1">
  <resource-env-ref>
    <resource-env-ref-name>
      concurrent/myExecutor
    </resource-env-ref-name>
    <resource-env-ref-type>
      javax.enterprise.concurrent.ManagedExecutorService
    </resource-env-ref-type>
  </resource-env-ref>
</web-app>
```

#07: Concurrency: ManagedScheduledExecutor

Managed version of `ScheduledExecutorService`

Submit delayed or periodic tasks

```
@Resource
```

```
ManagedScheduledExecutorService executor;
```


#07: Concurrency: ManagedScheduledExecutor

Access using JNDI

```
InitialContext ctx = new InitialContext();

ManagedScheduledExecutorService executor =
    (ManagedScheduledExecutorService) ctx.lookup(
        "java:comp/DefaultManagedScheduledExecutorService");
```

Can be defined in web.xml as well

#07: Concurrency: ManagedScheduledExecutor

```
executor.schedule(new MyCallableTask(), 5,  
TimeUnit.SECONDS);
```

```
executor.scheduleAtFixedRate(new  
MyRunnableTask(), 2, 3, TimeUnit.SECONDS);
```

```
executor.scheduleWithFixedDelay(new  
MyRunnableTask(), 2, 3, TimeUnit.SECONDS);
```

#08: Concurrency: ManagedThreadFactory

Extends ThreadFactory

```
@Resource(name = "DefaultManagedThreadFactory")  
ManagedThreadFactory factory;
```

```
ManagedThreadFactory factory =  
    (ManagedThreadFactory)  
    ctx.lookup("java:comp/DefaultManagedThreadFactory  
");
```

#08: Concurrency: ManagedThreadFactory

```
Thread thread = factory.newThread(new MyTask());
```

```
((ManageableThread) thread).isShutdown();
```

#09: Concurrency: DynamicProxy

Create dynamic proxy objects, adds contextual information available for applications running in Java EE environment

Classloading, JNDI, Security, ...

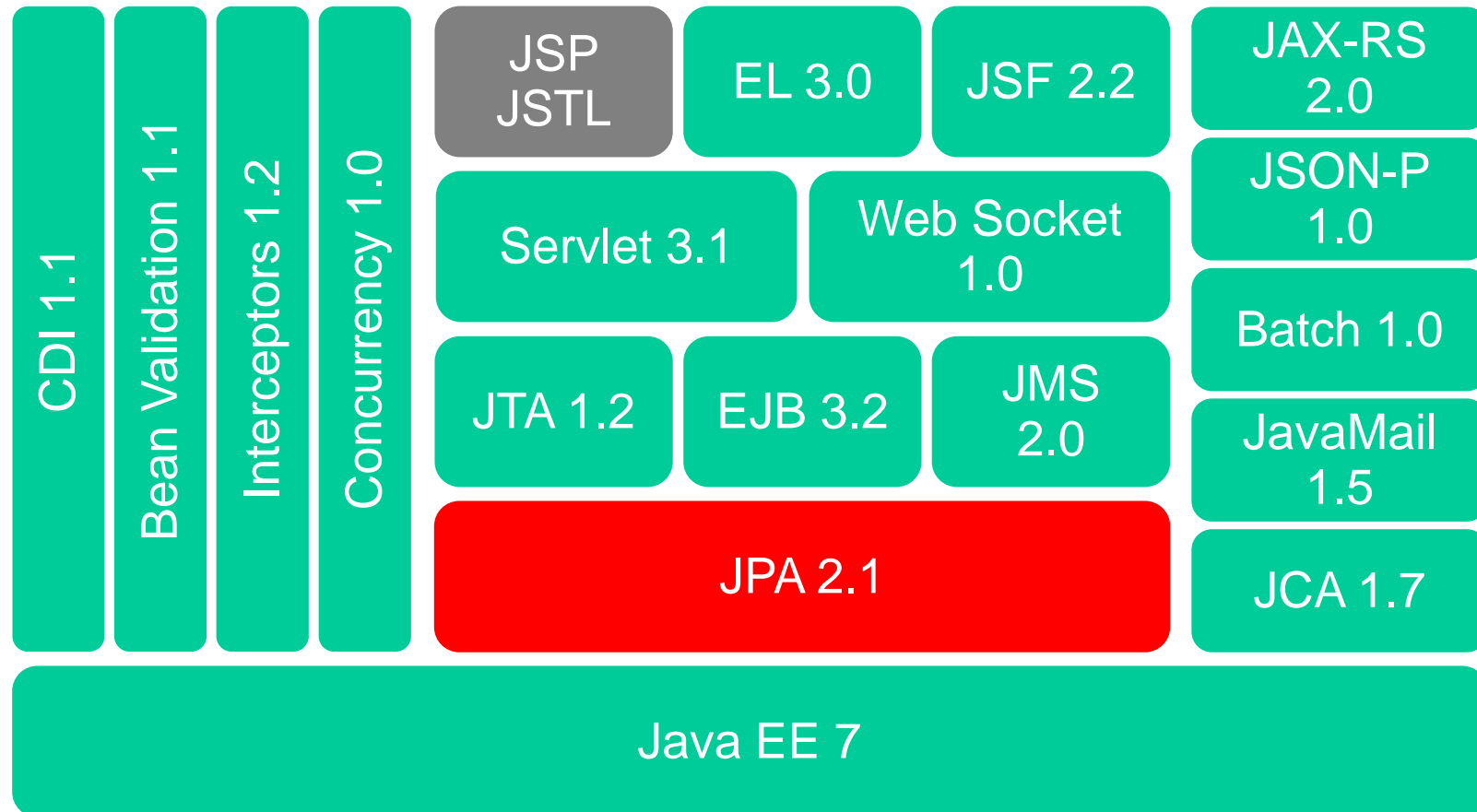
#09: Concurrency: DynamicProxy

```
@Resource  
ContextService service;
```

```
Runnable proxy =  
service.createContextualProxy(new  
MyRunnable(), Runnable.class);
```

```
Future f = executor.submit(proxy);
```

JPA 2.1 (JSR 338)



#10: JPA: Schema Generation

Standardized database schema generation

```
<persistence ... version="2.1">
  <persistence-unit ...>
    <properties>
      <property name="javax.persistence.schema-generation.scripts.action"
        value="drop-and-create"/>
      <property name="javax.persistence.schema-generation.scripts.create-
target"
        value="create.sql"/>
      <property name="javax.persistence.sql-load-script-source"
        value="insert.sql"/>
    </properties>
  </persistence-unit>
```


#11: JPA: @Index

Defines additional indexes in schema generation

```
@Entity
@Table(indexes = {
    @Index(columnList = "ISBN"),
    @Index(columnList = "NBOFPAGE")
})
public class Book {

    @Id @GeneratedValue
    private Long id;
    private String isbn;
    private Integer nbOfPage;
    ...
}
```

#12: JPA: Unsynchronized Persistence Context

Persistence context is not enlisted in any tx unless explicitly joined

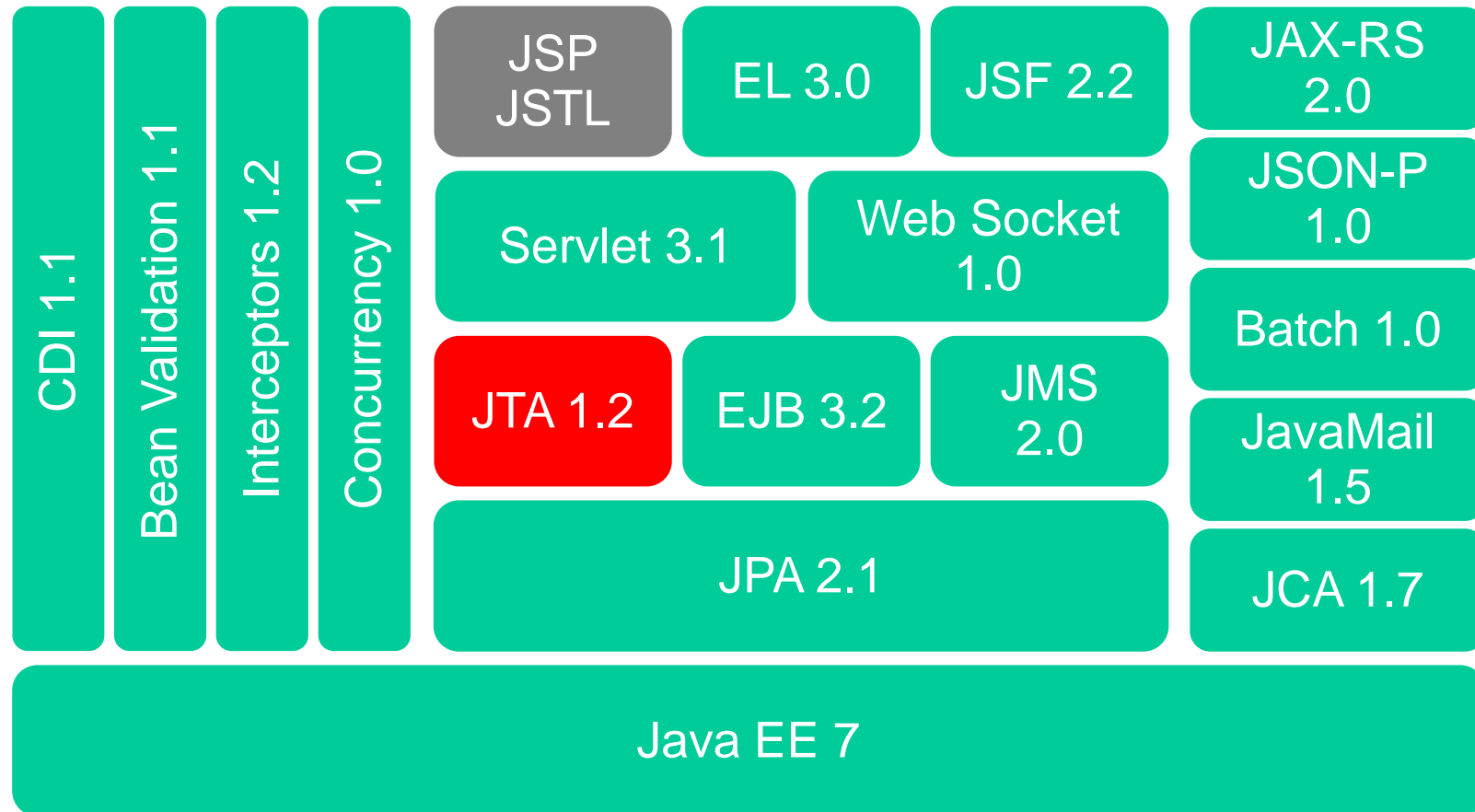
```
@PersistenceContext (synchronization =  
SynchronizationType.UNSYNCHRONIZED)  
private EntityManager em;  
...  
em.persist (book) ;  
  
...  
em.joinTransaction () ;
```

#13: JPA: Stored Procedure

Calling a stored procedure

```
@Entity
@NamedStoredProcedureQuery(name = "archiveOldBooks",
                           procedureName = "sp_archive_books",
                           parameters = {
                               @StoredProcedureParameter(name = "date", mode = IN,
                                                         type = Date.class),
                               @StoredProcedureParameter(name = "warehouse", mode = IN,
                                                         type = String.class)
                           })
public class Book {...}
```

JTA 1.2 (JSR 907)



#14: JTA: @Transactional

Transaction management on Managed Beans as CDI interceptor binding

```
@Path("book")
@Transactional(value = Transactional.TxType.REQUIRED,
               rollbackOn = {SQLException.class,
                             JMSEException.class},
               dontRollbackOn = SQLWarning.class)
public class BookRestService {

    @PersistenceContext
    private EntityManager em;

    @POST
    @Consumes(MediaType.APPLICATION_XML)
    public Response createBook(Book book) {...}
}
```

#15: JTA: @TransactionScoped

CDI scope whose lifecycle is scoped to the currently active JTA transaction

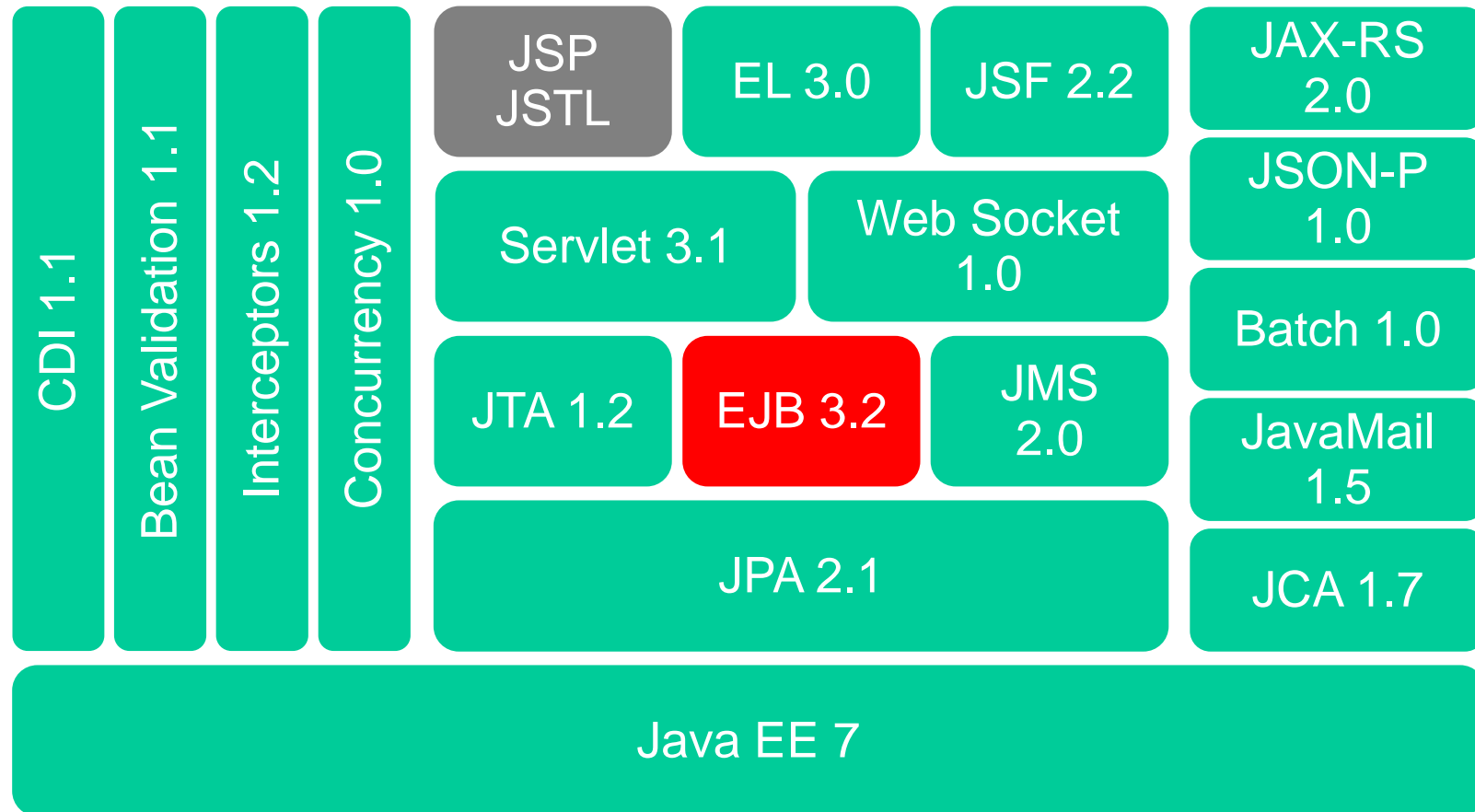
@TransactionScoped

```
public class BookBean {...}

@WebServlet
public class TxServlet extends HttpServlet {
    @Inject UserTransaction tx;
    @Inject BookBean b1;
    @Inject BookBean b2;

    protected void processRequest(...) {
        tx.begin();
        s_out.println(b1.getReference());
        s_out.println(b2.getReference());
        tx.commit();
    }
}
```

EJB 3.2 (JSR 345)



#16: EJB: Disable passivation of stateful

In some cases increases performance, scalability and robustness

```
@Stateful(passivationCapable = false)  
public class ShoppingCart {  
    ...  
}
```


#17: EJB-Lite: Async + Non-persistent timer

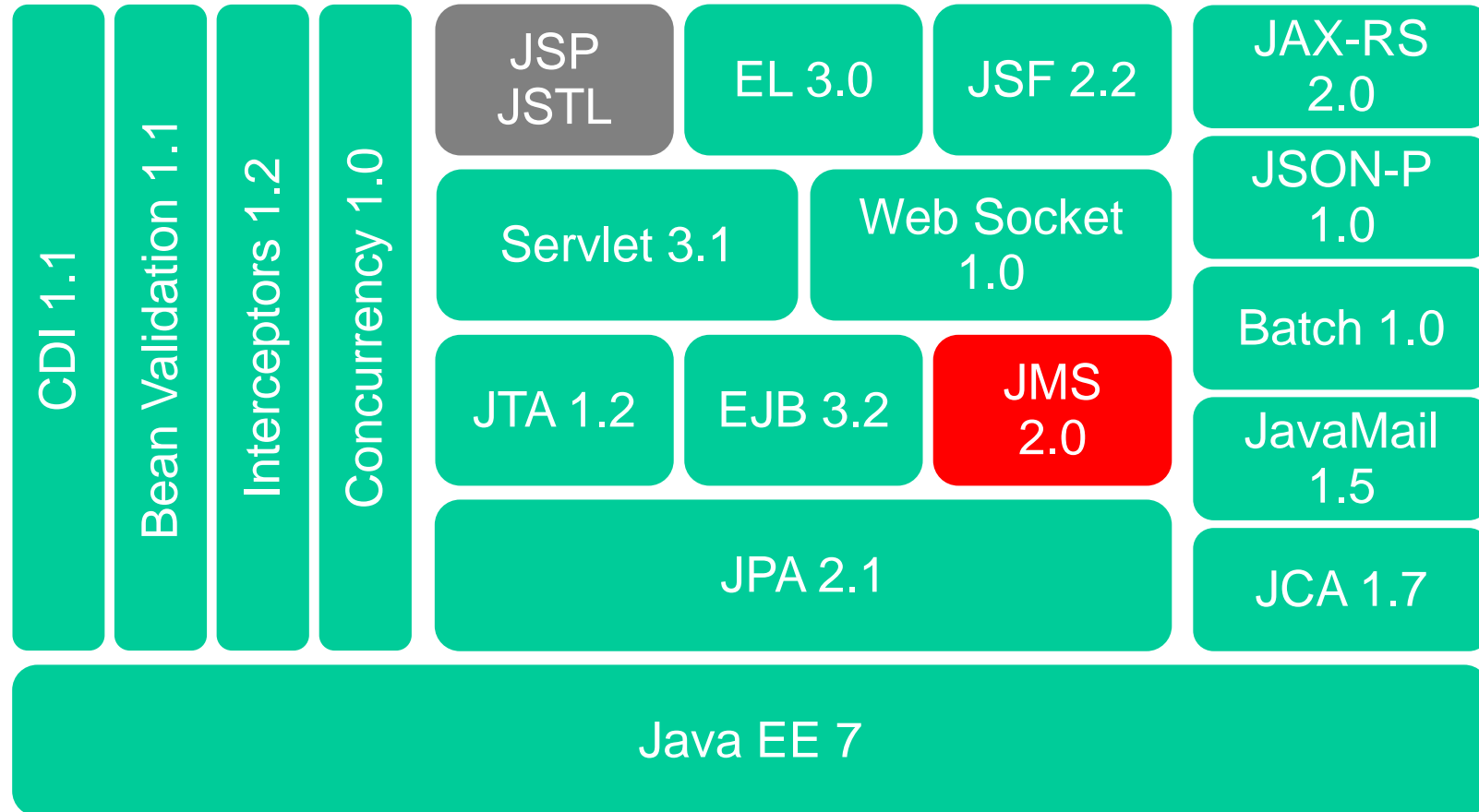
Extended the EJB Lite to include local asynchronous invocations and non-persistent EJB Timer Service

```
@Stateless
public class OrderEJB {

    @Asynchronous
    public void sendEmail (Order order) {
        // Very Long task
    }

    @Schedule(hour="2", persistent=false)
    public void createDailyReport() {
        // ...
    }
}
```

JMS 2.0 (JSR 343)



#18: JMS: JMSContext API

New simplified API to produce and consume messages

```
JMSContext ctx = connectionFactory.createContext();
```

```
ctx.createProducer().send(queue, "Text message sent");
```

```
ctx.createConsumer(queue).receiveBody(String.class);
```

```
ctx.createProducer()  
    .setPriority(2)  
    .setTimeToLive(1000)  
    .setDeliveryMode(DeliveryMode.NON_PERSISTENT)  
    .send(queue, message);
```

#19: JMS: Autocloseable

Several JMS interfaces implement Autocloseable

```
try (JMSContext ctx = connectionFactory.createContext()) {  
    ctx.createProducer().send(queue, "Text message sent");  
}
```

...

```
try (JMSContext ctx = connectionFactory.createContext()) {  
    while (true) {  
        String s =  
ctx.createConsumer(queue).receiveBody(String.class);  
    }  
}
```

#20: JMS: JMSConnectionFactoryDefinition

A JMS ConnectionFactory can be defined using an annotation on a container-managed class

```
@Stateless
@JMSConnectionFactoryDefinition(
    name = "java:app/jms/MyConnectionFactory",
    interfaceName =
"javax.jms.TopicConnectionFactory")

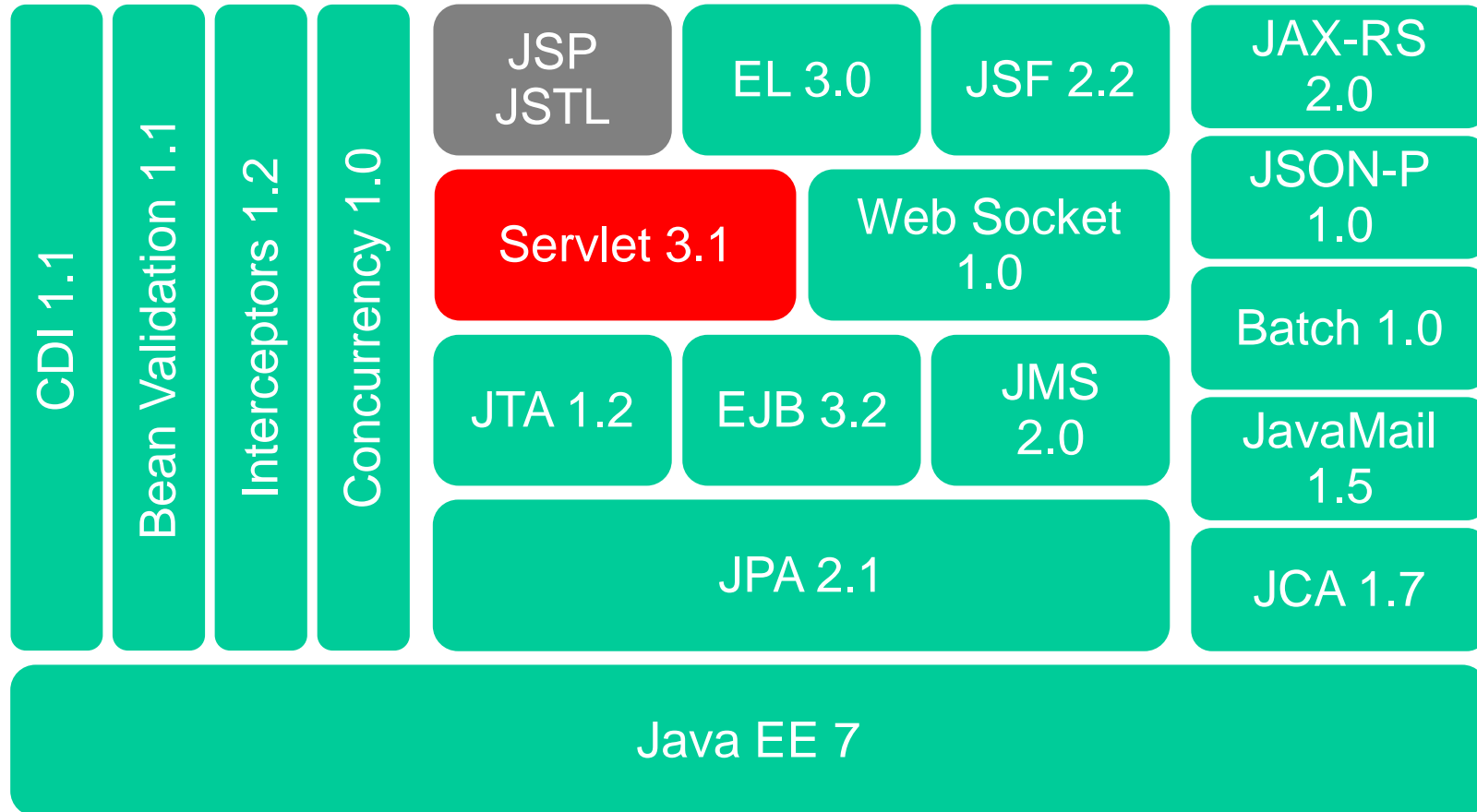
public class ExpensiveOrderEJB {...}
```

#21: JMS: JMSDestinationDefinition

A JMS queue or topic can be defined using an annotation

```
@Stateless
@JMSConnectionFactoryDefinition(
    name = "java:app/jms/MyConnectionFactory",
    interfaceName =
"javax.jms.TopicConnectionFactory")
@JMSDestinationDefinition(
    name = "java:app/jms/MyTopic",
    interfaceName = "javax.jms.Topic")
public class ExpensiveOrderEJB {...}
```

Servlet 3.1 (JSR 340)



#22: Servlet: Non-blocking I/O

```
public class TestServlet extends HttpServlet
    protected void doGet(HttpServletRequest request,
                          HttpServletResponse response)
                          throws IOException, ServletException {
    ServletInputStream input = request.getInputStream();
    byte[] b = new byte[1024];
    int len = -1;
    while ((len = input.read(b)) != -1) {
        . . .
    }
}
}
```


#22: Servlet: Non-blocking I/O

New methods to existing interfaces

ServletInputStream

```
public void setReadListener(ReadListener
listener);
```

```
public boolean isFinished();
```

```
public boolean isReady();
```

ServletOutputStream

```
public setWriteListener(WriteListener
listener);
```

```
public boolean canWrite();
```

#22: Servlet: Non-blocking I/O

New interfaces

```
public interface ReadListener extends
EventListener {
    public void onDataAvailable ();
    public void onAllDataRead ();
    public void onError ();
}
```

```
public interface WriteListener extends
EventListener {
    public void onWritePossible ();
    public void onError ();
}
```

#22: Servlet: Non-blocking I/O

Only for Asynchronous Servlets

```
AsyncContext context = request.startAsync();  
ServletInputStream input =  
request.getInputStream();  
input.setReadListener(  
    new MyReadListener(input, context));
```

#23: Servlet: Protocol Upgrade

```
<T extends HttpUpgradeHandler> T  
HttpServletRequest.upgrade(Class<T> class) throws  
IOException;
```

```
HttpUpgradeHandler  
    init(WebConnection wc);  
    destroy();
```

#23: Servlet: Protocol Upgrade

```
public interface WebConnection {  
    ServletInputStream getInputStream();  
    ServletOutputStream getOutputStream();  
}
```

#24: Servlet: Improved Security

Deny an HTTP method request for an uncovered HTTP method

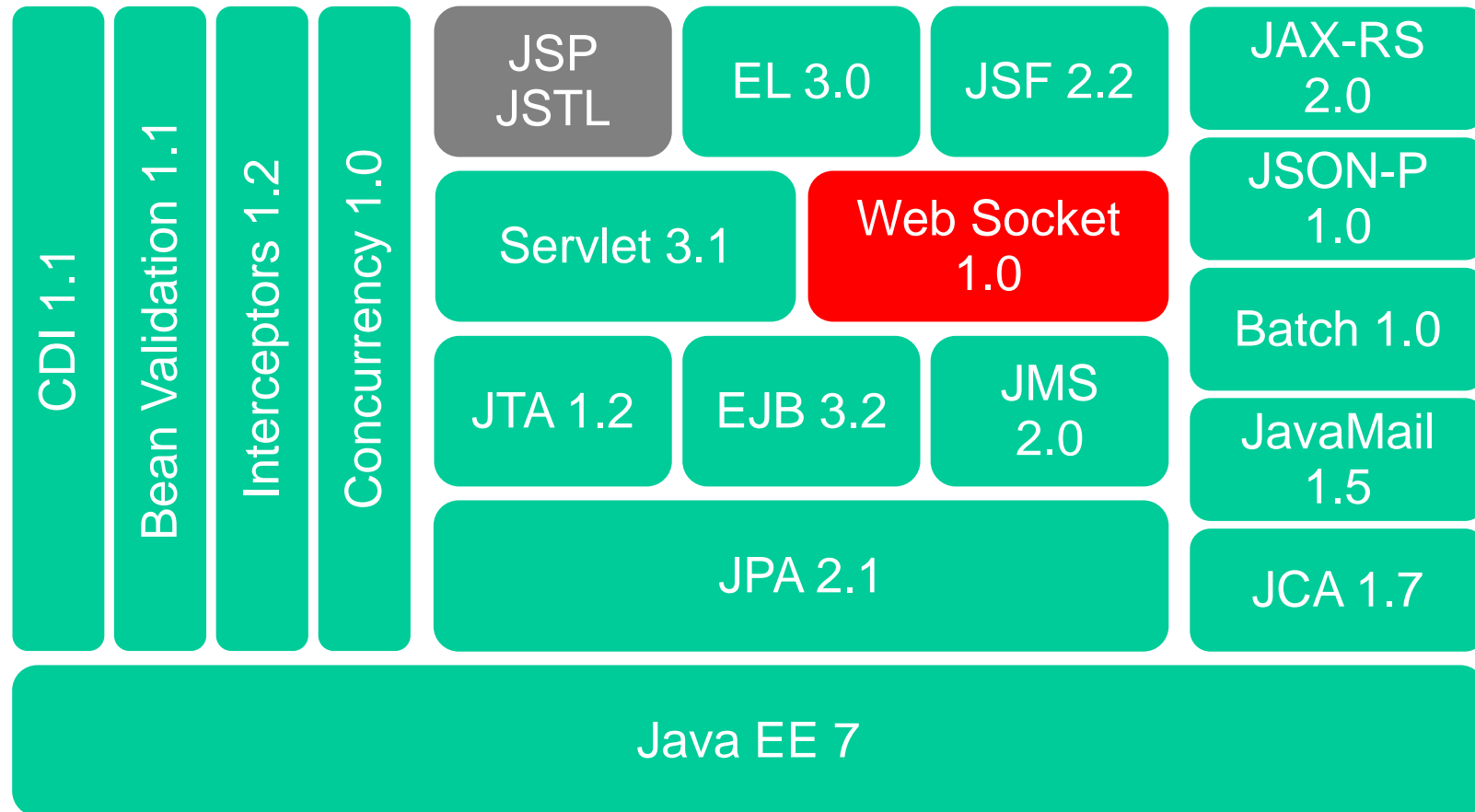
```
<web-app . . . version="3.1">
  <web-resource-collection>
    <url-pattern>/account/*</url-pattern>
    <http-method>GET</http-method>
  </web-resource-collection>
  <auth-constraint>
    . . .
  </auth-constraint>
</web-app>
```

#24: Servlet: Improved Security

Deny an HTTP method request for an uncovered HTTP method

```
<web-app . . . version="3.1">  
  <deny-uncovered-http-methods/>  
  <web-resource-collection>  
    <url-pattern>/account/*</url-pattern>  
    <http-method>GET</http-method>  
  </web-resource-collection>  
<auth-contraint>  
  . . .  
</auth-contraint>  
</web-app>
```

Web Socket 1.0 (JSR 356)



#25: WebSocket: Annotated server endpoint

Enables full-duplex bi-directional communication over single TCP connection

```
@javax.websocket.server.ServerEndpoint("/chat")  
public class ChatServer {  
  
    @OnMessage  
    public String chat(String name, Session session) {  
        for (Session peer : client.getOpenSessions()) {  
            peer.getBasicRemote().sendObject(message);  
        }  
    }  
}
```

#26: WebSocket: Lifecycle callbacks

@javax.websocket.OnOpen

```
public void open(Session s) { . . . }
```

@javax.websocket.OnClose

```
public void close(CloseReason c) { . . . }
```

@javax.websocket.OnError

```
public void error(Throwable t) { . . . }
```

#27: WebSocket: Annotated client endpoint

@javax.websocket.ClientEndpoint

```
public class MyClient {  
    @javax.websocket.OnOpen  
    public void open(Session session) { ... }  
  
    // Lifecycle callbacks  
}
```

#27: WebSocket: Annotated client endpoint

```
ContainerProvider
    .getWebSocketContainer()
    .connectToServer(
        MyClient.class,
        URI.create("ws://. . ."));
```

#28: WebSocket: Programmatic endpoints

```
public class ChatServer extends Endpoint {  
    @Override  
    public void onOpen(Session s, EndpointConfig ec) {  
        s.addMessageHandler(new MessageHandler.Whole<String>()  
    {  
        public void onMessage(String text) { . . . }  
    }  
    }  
  
    @Override  
    public void onClose(Session s, CloseReason cr) { . . . }  
  
    // . . .  
}
```

#28: WebSocket: Programmatic endpoints

```
public class MyApplicationConfig implements
ServerApplicationConfig {
    public Set<ServerEndpointConfig>
getEndpointConfigs(...) {
    ServerEndpointConfig.Builder
        .create(MyEndpoint.class, "/websocket")
        .configurator(new MyConfig())
        .build()
    }
}
```

#28: WebSocket: Programmatic endpoints

```
public class MyConfig extends  
ServerEndpointConfig.Configurator {  
  
    public <T> T getEndpointInstance (. . .) { . . . }  
  
    public void modifyHandshake (. . .) { . . . }  
  
    . . .  
}
```

#29: WebSocket: Encoder and Decoder

```
@javax.websocket.server.ServerEndpoint (
    value="/chat",
    decoders="MyDecoder.class",
    encoders="MyEncoder.class")
public class ChatServer {

    @OnMessage
    public String chat(ChatMessage name, Session session) {
        . . .
    }
}
```


#29: WebSocket: Encoder and Decoder

```
public class MyDecoder implements Decoder.Text<ChatMessage>
{
    public ChatMessage decode(String s) {
        // . . .
    }

    public boolean willDecode(String string) {
        // . . .
    }

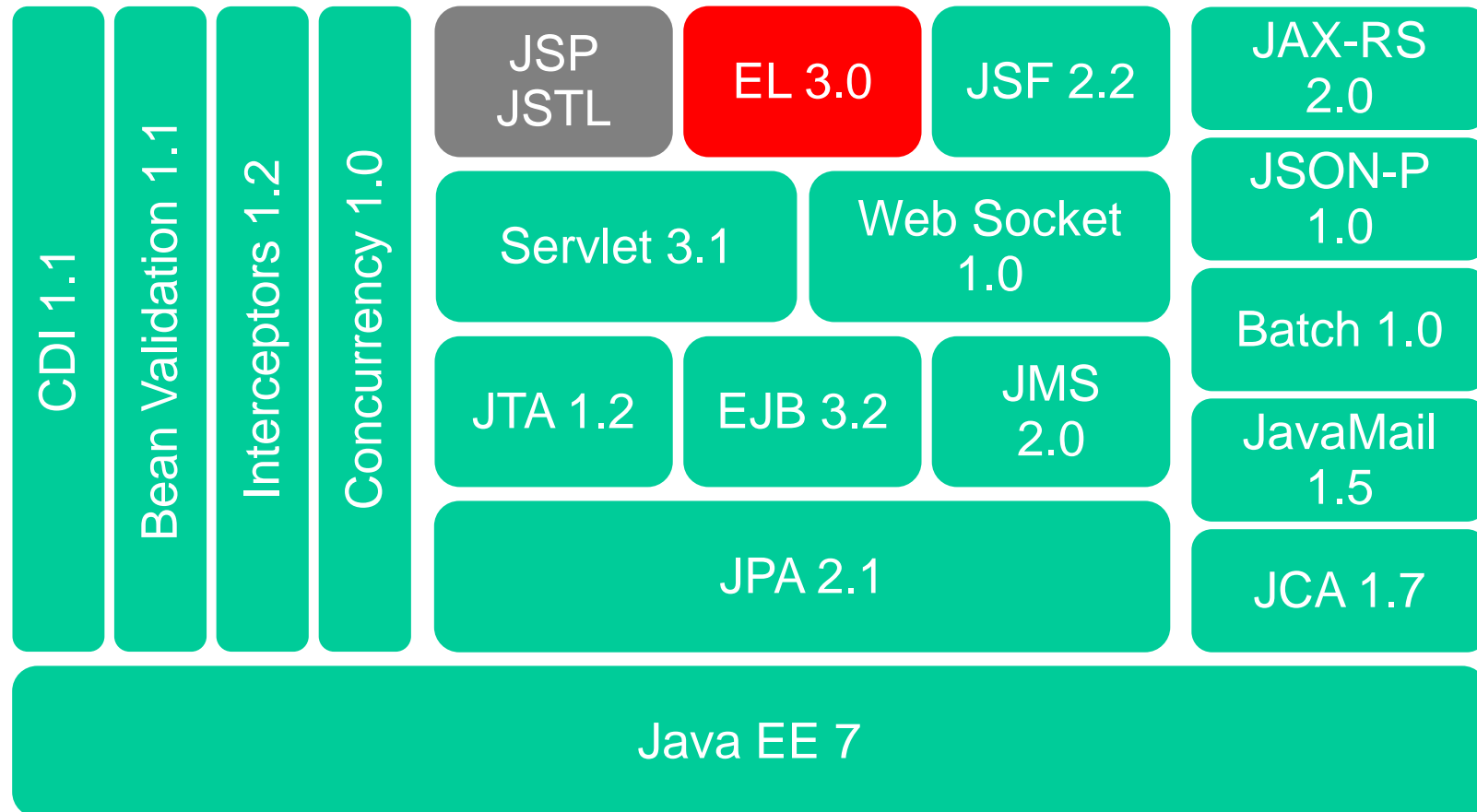
    // . . .
}
```

#29: WebSocket: Encoder and Decoder

```
public class MyEncoder implements Encoder.Text<ChatMessage>
{
    public String encode(ChatMessage chatMessage) {
        // . . .
    }

    // . . .
}
```

Expression Language 3.0 (JSR 341)



#30: Expression Language: ELProcessor

Use EL in a stand-alone environment

- Evaluate EL expressions

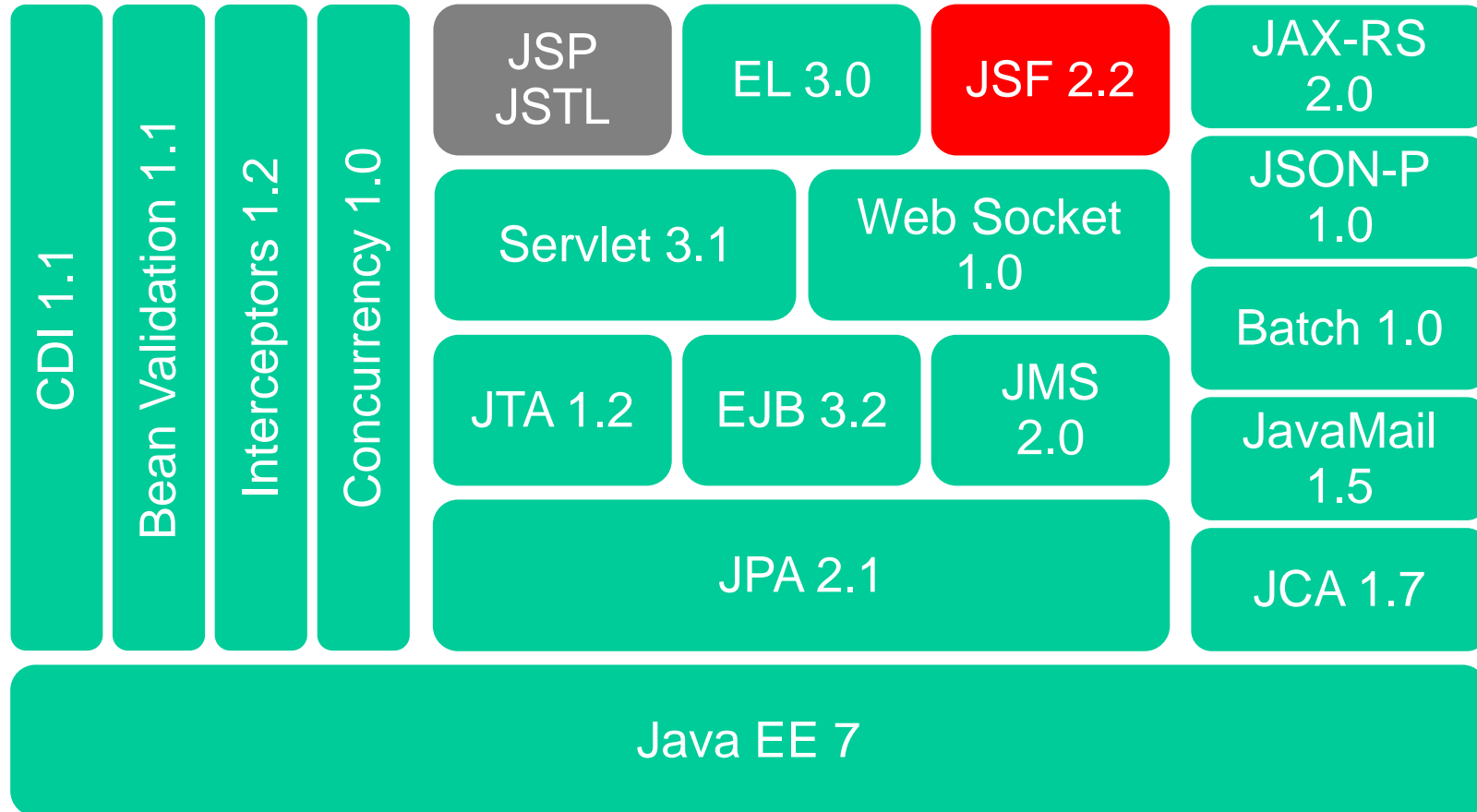
- Get/set bean properties

- Defining a static method as an EL function

- Defining an object instance as an EL name

```
ELProcessor elp = new ELProcessor();  
elp.defineBean("employee", new Employee("Charlie Brown"));  
String name = elp.eval("employee.name");
```

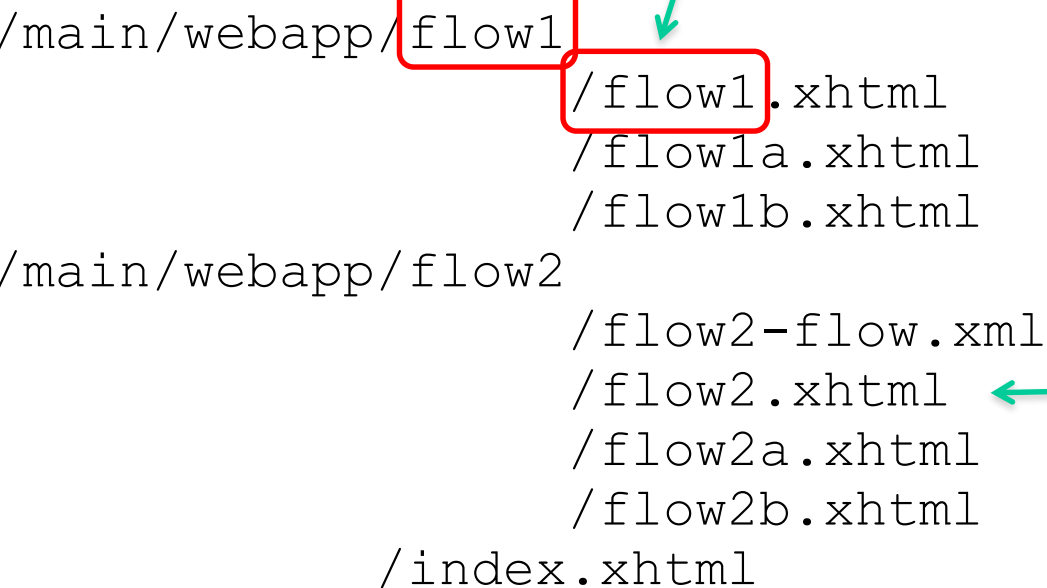
JSF 2.2 (JSR 344)



#31: JSF: Faces Flow

Package reusable flows in JAR

```
./src/main/webapp/flow1  
    /flow1.xhtml  
    /flow1a.xhtml  
    /flow1b.xhtml  
./src/main/webapp/flow2  
    /flow2-flow.xml  
    /flow2.xhtml  
    /flow2a.xhtml  
    /flow2b.xhtml  
/index.xhtml
```



#31: JSF: Faces Flow

Package reusable flows in JAR

```
@Named
@FlowScoped("flow1")
public class Flow1Bean implements Serializable {
}
```

```
@Produces @FlowDefinition
public Flow defineFlow(@FlowBuilderParameter FlowBuilder
fb) {
    String flowId = "flow1";
    // . . .
    return fb.getFlow();
}
```

#31: JSF: Faces Flow

Package reusable flows in JAR

`#{flowScope}`: Local flow storage

`#{facesContext.application.flowHandler.currentFlow}`
: Returns true if within a flow

#32: JSF: Resource Library Contract

Apply templates in a reusable and interchangeable manner

```
index-blue.xhtml
index-red.xhtml
WEB-INF/lib/contracts-library-1.0-SNAPSHOT.jar
    /META-INF/contracts/blue
        /style.css
        /javax.faces.contract.xml
        /template.xhtml
    /META-INF/contracts/red
        /style.css
        /javax.faces.contract.xml
        /template.xhtml
```

#32: JSF: Resource Library Contract

Apply templates in a reusable and interchangeable manner

```
<f:view contracts="red">  
  <ui:composition template="/template.xhtml">  
    . . .  
  </ui:composition>  
</f:view>
```

#33: JSF: Pass-through Attributes

HTML5-Friendly Markup

```
<h:inputText type="email" value="#{user.email}"/>
```

```
<input type="text" name="j_idt6:j_idt10"/>
```

```
<h:inputText p:type="email" value="#{user.email}"/>
```

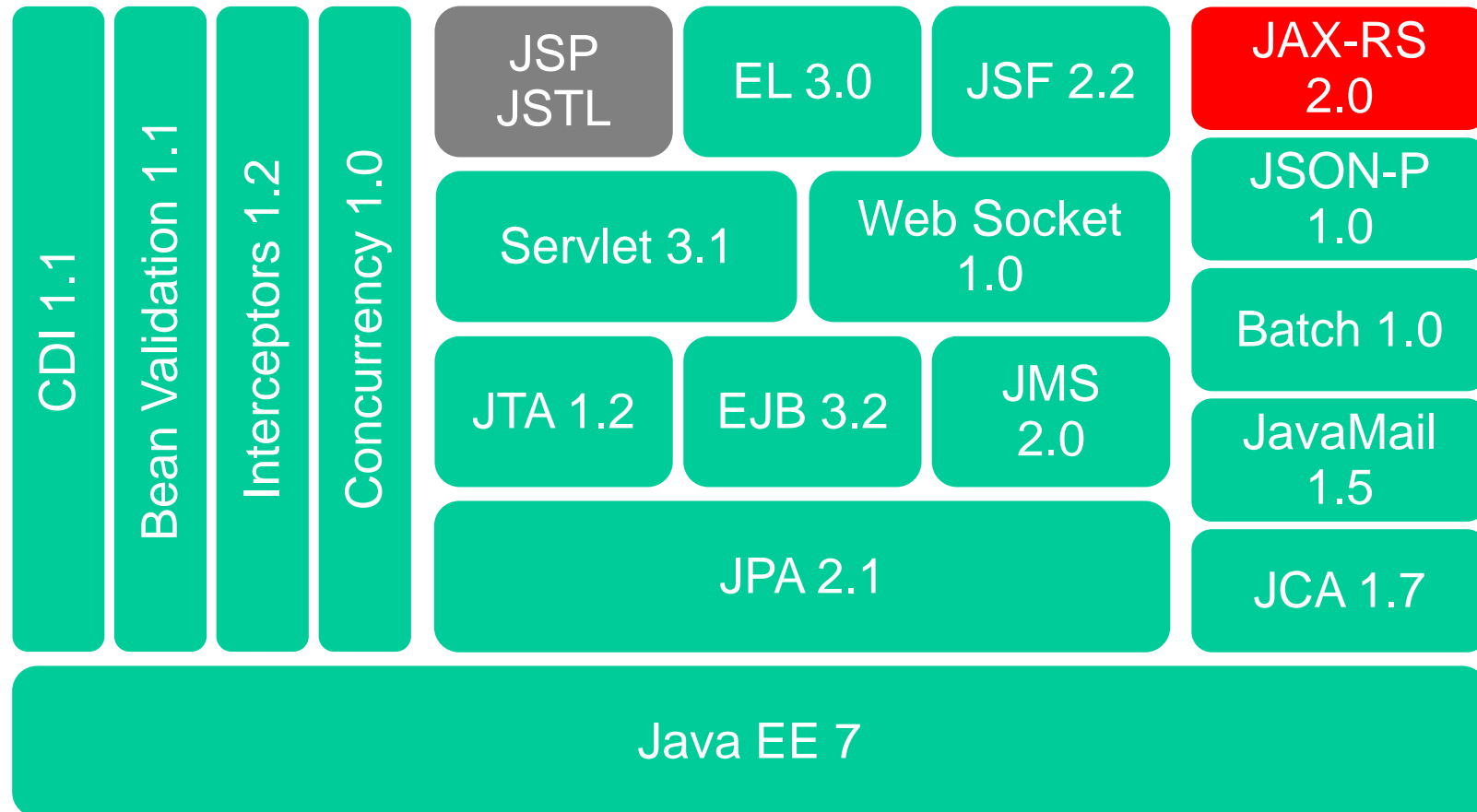
```
<input type="email" name="j_idt6:j_idt10"/>
```

#34: JSF: File Upload Component

```
<h:form enctype="multipart/form-data">  
  <h:inputFile value="#{fileUploadBean.file}"/><br/>  
  <h:commandButton value="Upload"/><p/>  
</h:form>
```

```
@Named @RequestScoped  
public class FileUploadBean {  
    private Part file;  
  
    //getter and setter  
}
```

JAX-RS 2.0 (JSR 339)



#35: JAX-RS: Client API

New API to consume rest services

```
Client client = ClientBuilder.newClient();  
WebTarget target =  
client.target("http://www.foo.com/book");  
Invocation invocation =  
target.request(TEXT_PLAIN).buildGet();  
Response response = invocation.invoke();
```

```
Response response = ClientBuilder.newClient()  
    .target("http://www.foo.com/book")  
    .request(MediaType.TEXT_PLAIN)  
    .get();
```

```
String body = ClientBuilder.newClient()  
    .target("http://www.foo.com/book")  
    .request()  
    .get(String.class);
```

#36: JAX-RS: Async Client

The client API also supports asynchronous invocation

```
Future<String> future = ClientBuilder.newClient()
    .target("http://www.foo.com/book")
    .request()
    .async()
    .get(String.class);

try {
    String body = future.get(1, TimeUnit.MINUTES);
} catch (InterruptedException | ExecutionException e)
{...}
```

#37: JAX-RS: Async Server

Asynchronous request processing on the server

```
@Path("/async")
public class AsyncResource {

    @GET
    public void asyncGet(@Suspended AsyncResponse
asyncResp) {

        new Thread(new Runnable() {

            public void run() {
                String result = veryExpensiveOperation();
                asyncResp.resume(result);
            }
        }).start();
    }
}
```


#38: JAX-RS: Message Filter

Used to process incoming and outgoing request or response headers

Filters on client side

`ClientRequestFilter`

`ClientResponseFilter`

Filters on server side

`ContainerRequestFilter`

`ContainerResponseFilter`

#38: JAX-RS: Message Filter

Used to process incoming and outgoing request or response headers

```
public class LogginFilter implements ClientRequestFilter {  
    public void filter(ClientRequestContext ctx) throws  
    IOException {  
        System.out.println(ctx.getMethod());  
        System.out.println(ctx.getUri());  
    }  
}
```

#39: JAX-RS: Entity Interceptors

Marshalling and unmarshalling HTTP message bodies

Intercepts inbound entity streams (reads from the “wire”)

`ReaderInterceptor`

Intercepts outbound entity streams (writes to the “wire”)

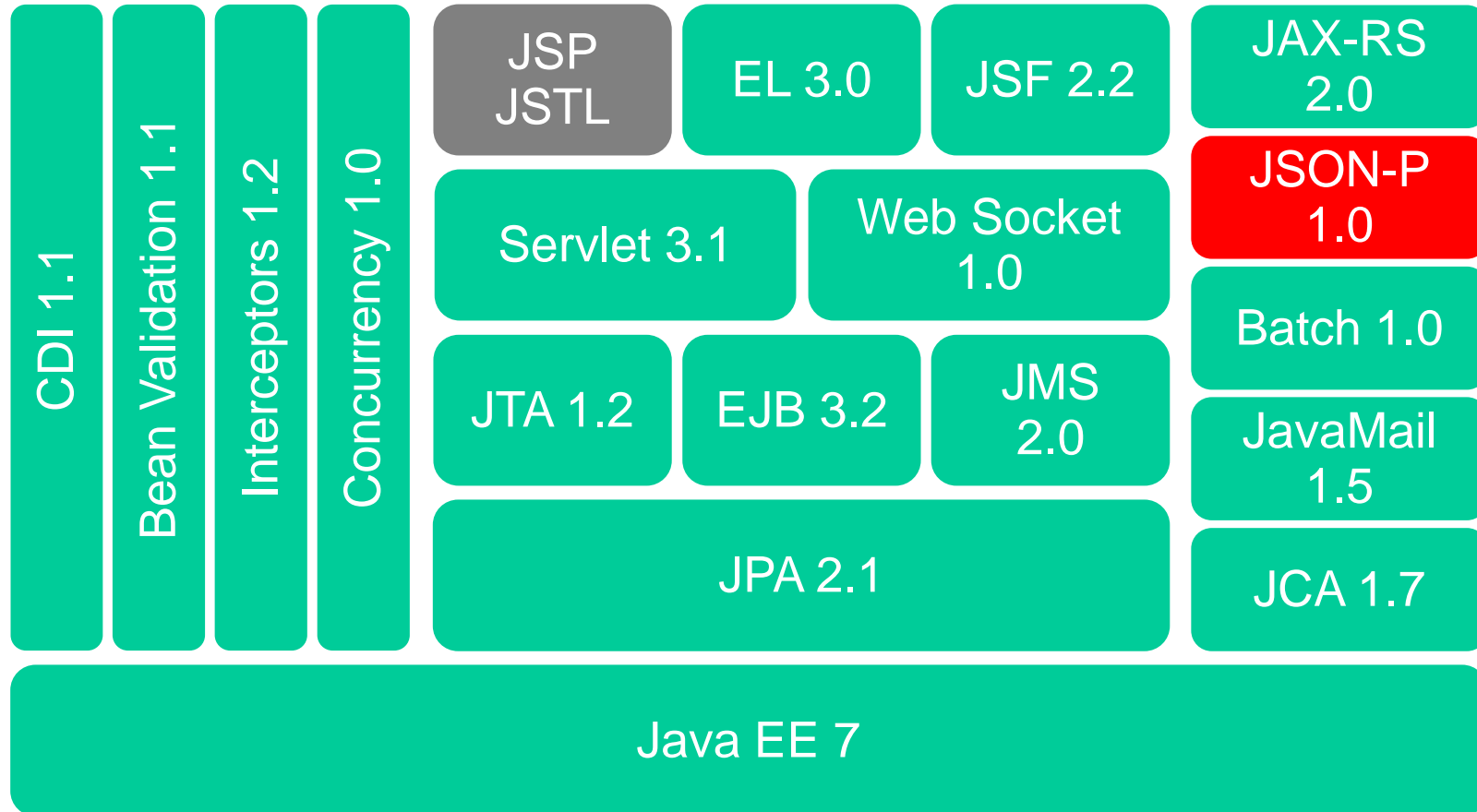
`WriterInterceptor`

#39: JAX-RS: Entity Interceptors

Marshalling and unmarshalling HTTP message bodies

```
public class GZipInterceptor implements
WriterInterceptor {
    public void aroundWriteTo(WriterInterceptorContext
ctx) {
        OutputStream os = ctx.getOutputStream();
        ctx.setOutputStream(new GZIPOutputStream(os));
        ctx.proceed();
    }
}
```

JSON-P 1.0 (JSR 353)



#40: JSON-P: JSON Builder

Creates an object model (or an array) in memory by adding elements

```
JsonObject value = Json.createObjectBuilder()
    .add("id", "1234")
    .add("date", "19/09/2012")
    .add("total_amount", "93.48")
    .add("customer", Json.createObjectBuilder()
        .add("first_name", "James")
        .add("last_name", "Rorrison")
        .add("email", "j.rorri@me.com")
        .add("phoneNumber", "+44 1234 1234")
    )
    .build();
```

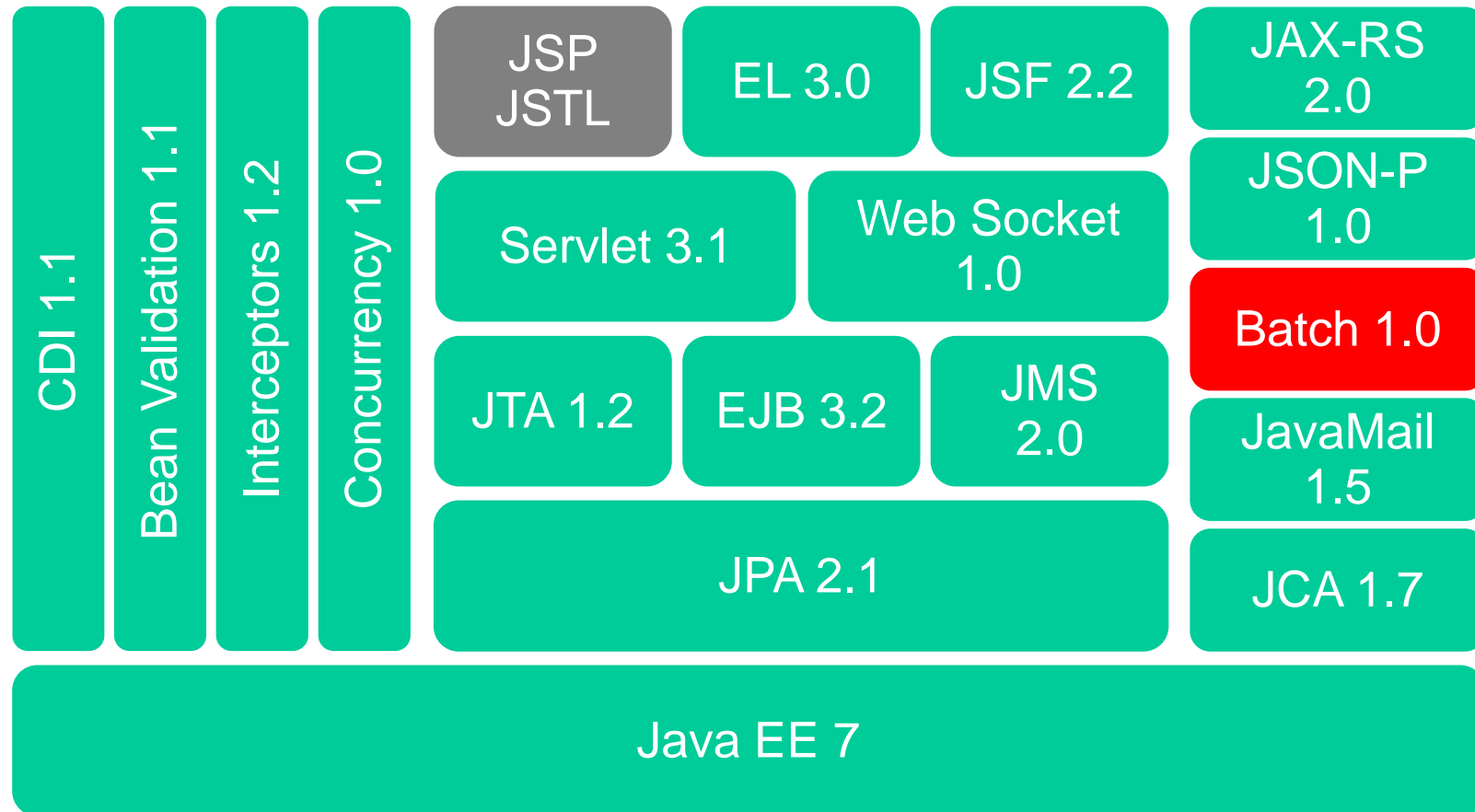
#41: JSON-P: JsonParser

Event-based parser that can read JSON data from a stream

```
JsonParser parser = Json.createParser(new
FileReader("order.json"));
while (parser.hasNext()) {
    JsonParser.Event event = parser.next();

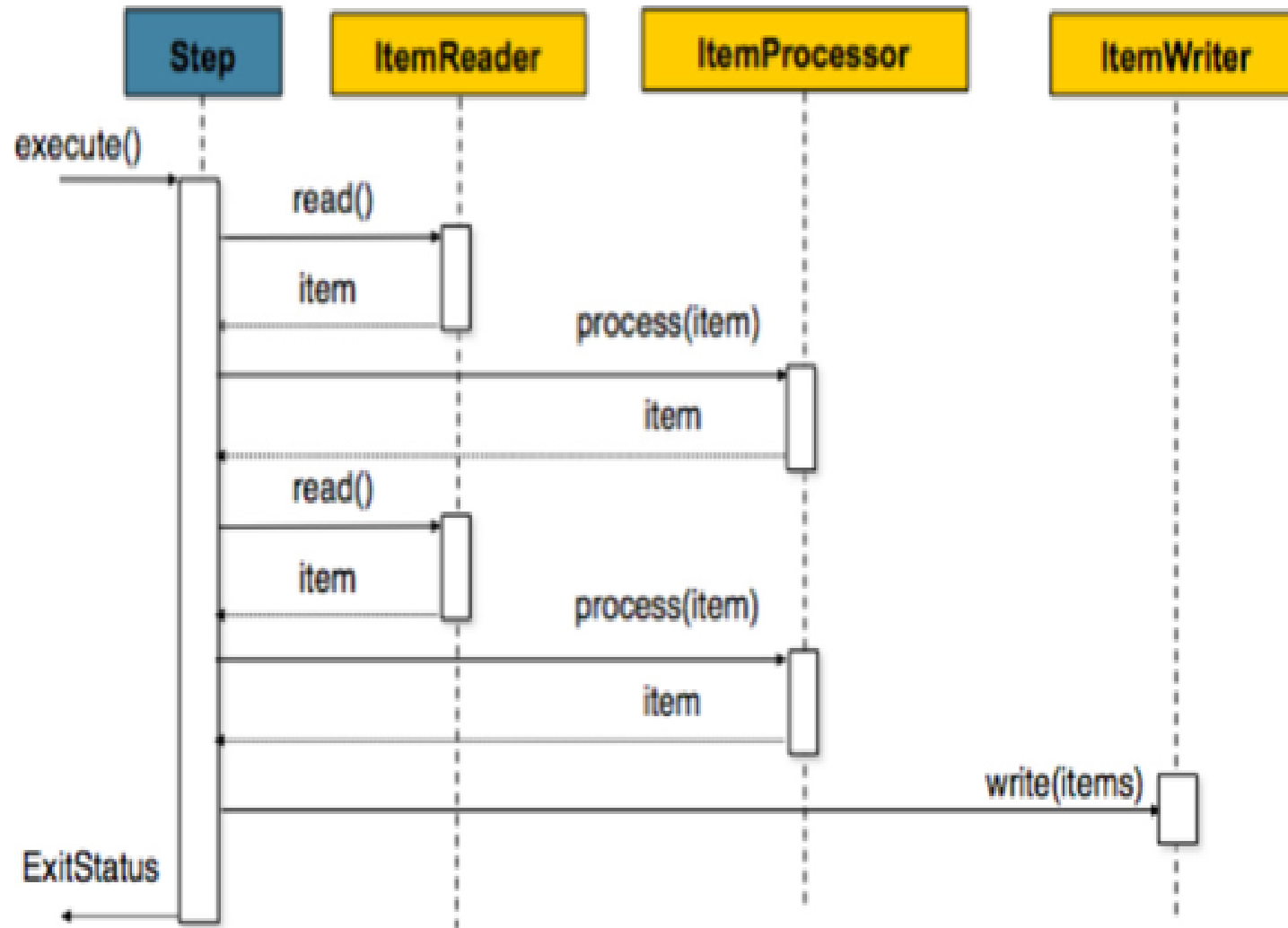
    if (event.equals(JsonParser.Event.KEY_NAME) &&
        parser.getString().matches("email")) {
        parser.next();
        email = parser.getString();
    }
}
```

Batch 1.0 (JSR 352)



#42: Batch: Chunk-style Processing

Item-oriented Processing Style (primary)



#42: Batch: Chunk-style Processing

```
<step id="sendStatements">
  <chunk item-count="3">
    <reader ref="accountReader"/>
    <processor ref="accountProcessor"/>
    <writer ref="emailWriter"/>
  </chunk>
</step>
```

...implements ItemReader {
public Object readItem() {
 // read account using JPA
}

...implements ItemProcessor {
public Object processItems(Object account)
 // read Account, return Statement
}

...implements ItemWriter {
public void writeItems(List accounts) {
 // use JavaMail to send email
}

#43: Batch: Batchlet-style Processing

Task-oriented processing style

```
<step id="transferFile">  
  <batchlet ref="MyFileTransfer" />  
</step>
```



```
...implements Batchlet {  
  @Override  
  public void process() {  
    // Transfer file  
  }  
}
```

#44: Batch: Job/Step/Chunk Listeners

```
<job id="myJob" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
version="1.0">
  <listeners>
    <listener ref="myJobListener"/>
  </listeners>
  <step id="myStep" >
    <listeners>
      <listener ref="myStepListener"/>
      <listener ref="myChunkListener"/>
      <listener ref="myItemReadListener"/>
      <listener ref="myItemProcessorListener"/>
      <listener ref="myItemWriteListener"/>
    </listeners>
    <chunk item-count="3">. . .</chunk>
  </step>
</job>
```

#44: Batch: Job/Step/Chunk Listeners

Interface	Abstract Classes
JobListener	AbstractJobListener
StepListener	AbstractStepListener
ChunkListener	AbstractChunkListener
ItemRead/Write/ProcessListener	AbstractItemRead/Write/ProcessListener
SkipRead/Write/ProcessListener	AbstractSkipRead/Write/ProcessListener
RetryRead/Write/ProcessListener	AbstractRetryRead/Write/ProcessListener

#44: Batch: Job/Step/Chunk Listeners

```
@Named
public class MyJobListener extends AbstractJobListener {

    @Override
    public void beforeJob() throws Exception { . . . }

    @Override
    public void afterJob() throws Exception { . . . }
}
```

#45: Batch: Partition

```
<step>
  <chunk item-count="3">
    <reader ref="myItemReader">
      <properties>
        <property name="start"
value="#{partitionPlan['start']}" />
        <property name="end" value="#{partitionPlan['end']}" />
      </properties>
    </reader>
    . . .
  </chunk>
```

#45: Batch: Partition

```
<partition>
  <plan partitions="2">
    <properties partition="0">
      <property name="start" value="1"/>
      <property name="end" value="10"/>
    </properties>
    <properties partition="1">
      <property name="start" value="11"/>
      <property name="end" value="20"/>
    </properties>
  </plan>
</partition>
</step>
```


#46: Batch: Creating Workflows

Flow: Elements that execute together as a unit

```
<flow id="flow1" next="step3">  
  <step id="step1" next="step2"> . . . </step>  
  <step id="step2"> . . . </step>  
</flow>  
<step id="step3"> . . . </step>
```

#46: Batch: Creating Workflows

Split: Concurrent execution of flows

```
<split id="split1" next=" . . . ">  
  <flow id="flow1">  
    <step id="step1"> . . . </step>  
  </flow>  
  <flow id="flow2">  
    <step id="step2"> . . . </step>  
  </flow>  
</split>
```

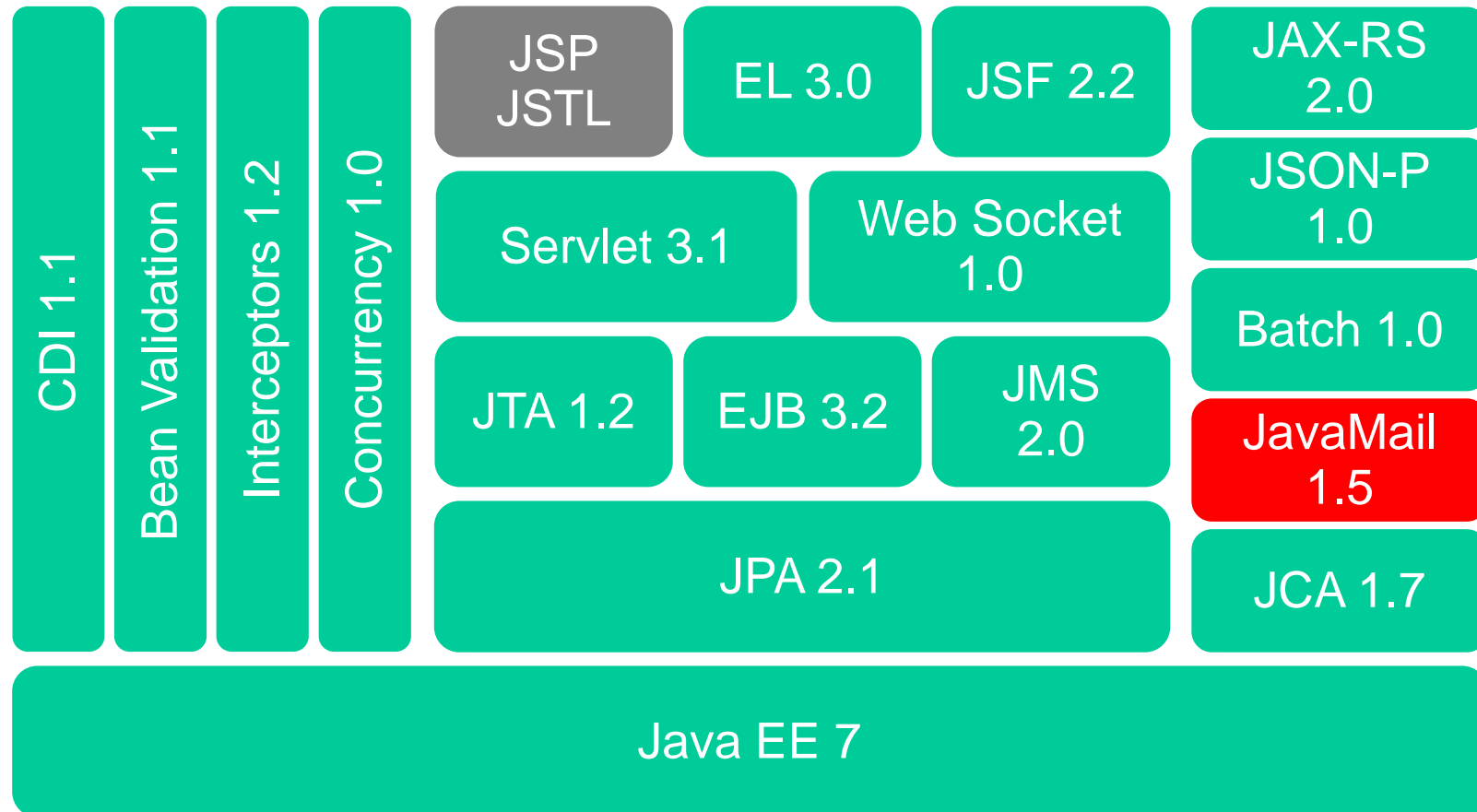
#46: Batch: Creating Workflows

Decision: Customized way of sequencing between steps, flows, splits

```
<step id="step1" next="decider1">. . .</step>
<decision id="decider1" ref="myDecider">
  <next on="DATA_LOADED" to="step2"/>
  <end on="NOT_LOADED"/> </decision>
<step id="step2">. . .</step>
```

```
@Named
public class MyDecider implements Decider {
    @Override
    public String decide(StepExecution[] ses) throws
Exception {
        . . .
        return "DATA_LOADED"; // or "NOT_LOADED"
    }
}
```

JavaMail 1.5 (JSR 919)

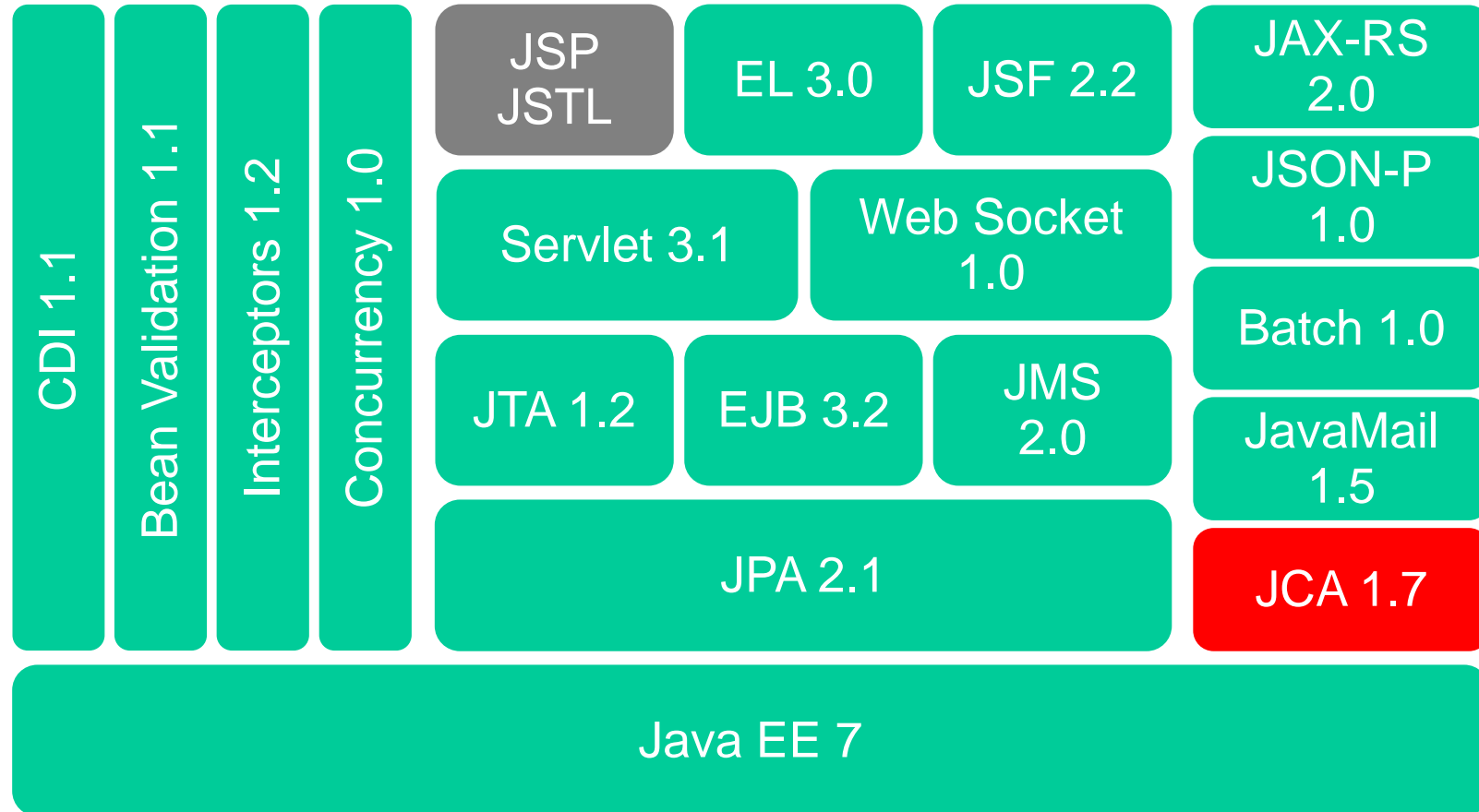


#47: JavaMail

```
@MailSessionDefinition(name = "java:comp/myMailSession",
    properties = {
        "mail.smtp.host=smtp.gmail.com",
        "mail.smtp.ssl.enable=true",
        "mail.smtp.auth=true",
        "mail.transport.protocol=smtp",
        "mail.debug=true"
    })
```

```
@Resource(lookup = "java:comp/myMailSession")
Session session;
```

JCA 1.7 (JSR 322)

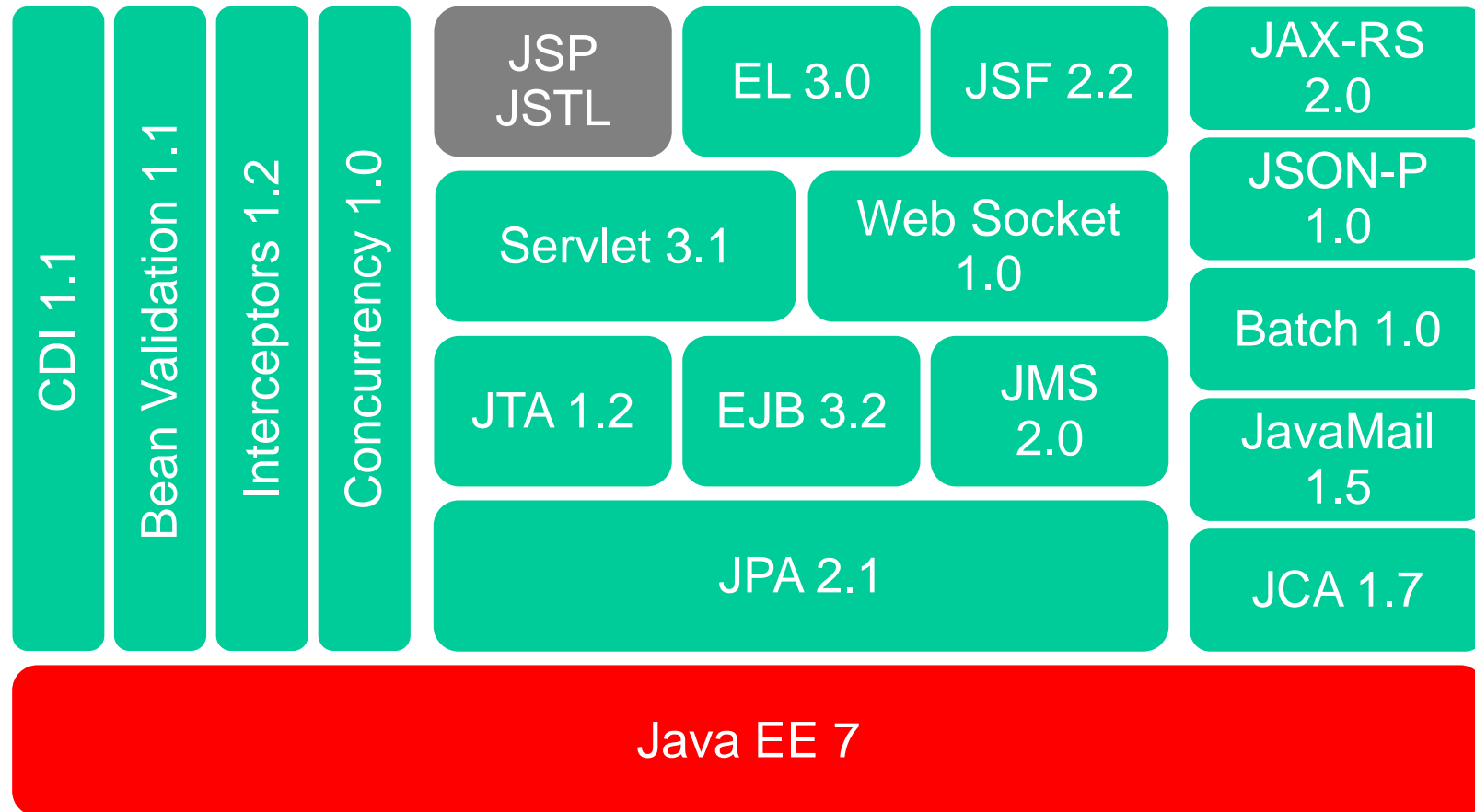


#48: Java Connector Architecture

```
@ConnectionDefinition(  
    connection="MyConnection.class",  
    connectionImpl="MyConnectionImpl.class",  
    connectionFactory="MyConnectionFactory.class",  
    connectionFactoryImpl="MyConnectionFactoryImpl.class"  
)
```

```
@AdministeredObjectDefinition(  
    className="MyQueueImpl.class",  
    name="java:comp/MyQueue",  
    resourceAdapter="myAdapter",  
)
```

Java EE 7 (JSR 342)



#49: Default Resources

Default Data Source

JNDI name: `java:comp/DefaultDataSource`

```
@Resource(lookup="java:comp/DefaultDataSource")  
DataSource myDS;
```

```
@Resource  
DataSource myDS;
```

#49: Default Resources

Default JMS Connection Factory

JNDI name:

```
java:comp/DefaultJMSConnectionFactory
```

```
@Resource(lookup="java:comp/DefaultJMSConnectionFactory")  
ConnectionFactory myCF;
```

```
@Resource  
ConnectionFactory myCF;
```

#49: Default Resources

Default Concurrency Utilities Objects

JNDI names

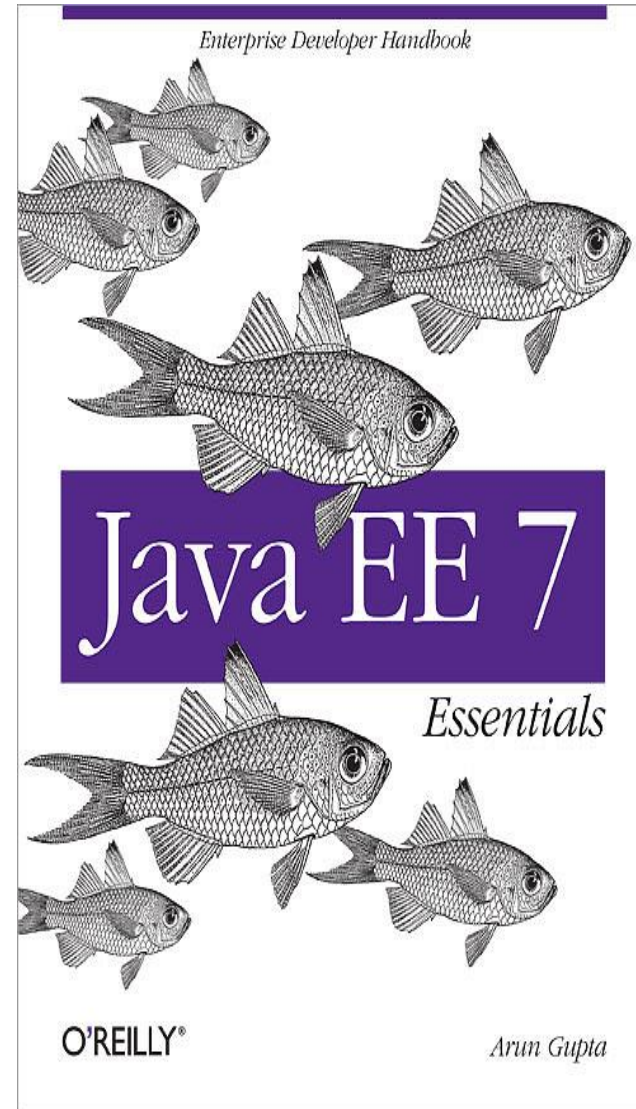
`java:comp/DefaultManagedExecutorService`

`java:comp/DefaultManagedScheduledExecutorService`

`java:comp/DefaultManagedThreadFactory`

`java:comp/DefaultContextService`

#50: Buy our books!



References

Java EE Samples Project:

github.com/javaee-samples/javaee7-samples

Ticket-Monster Examples Applications (EE 6!):

<https://github.com/jboss-developer/ticket-monster/tree/2.6.0.Final-with-tutorials>

Fly Fast, and Free.

WildFly = 8.1

Download now!