ORACLE®

Java EE – Past, Present and Future

Mike Keith, Oracle

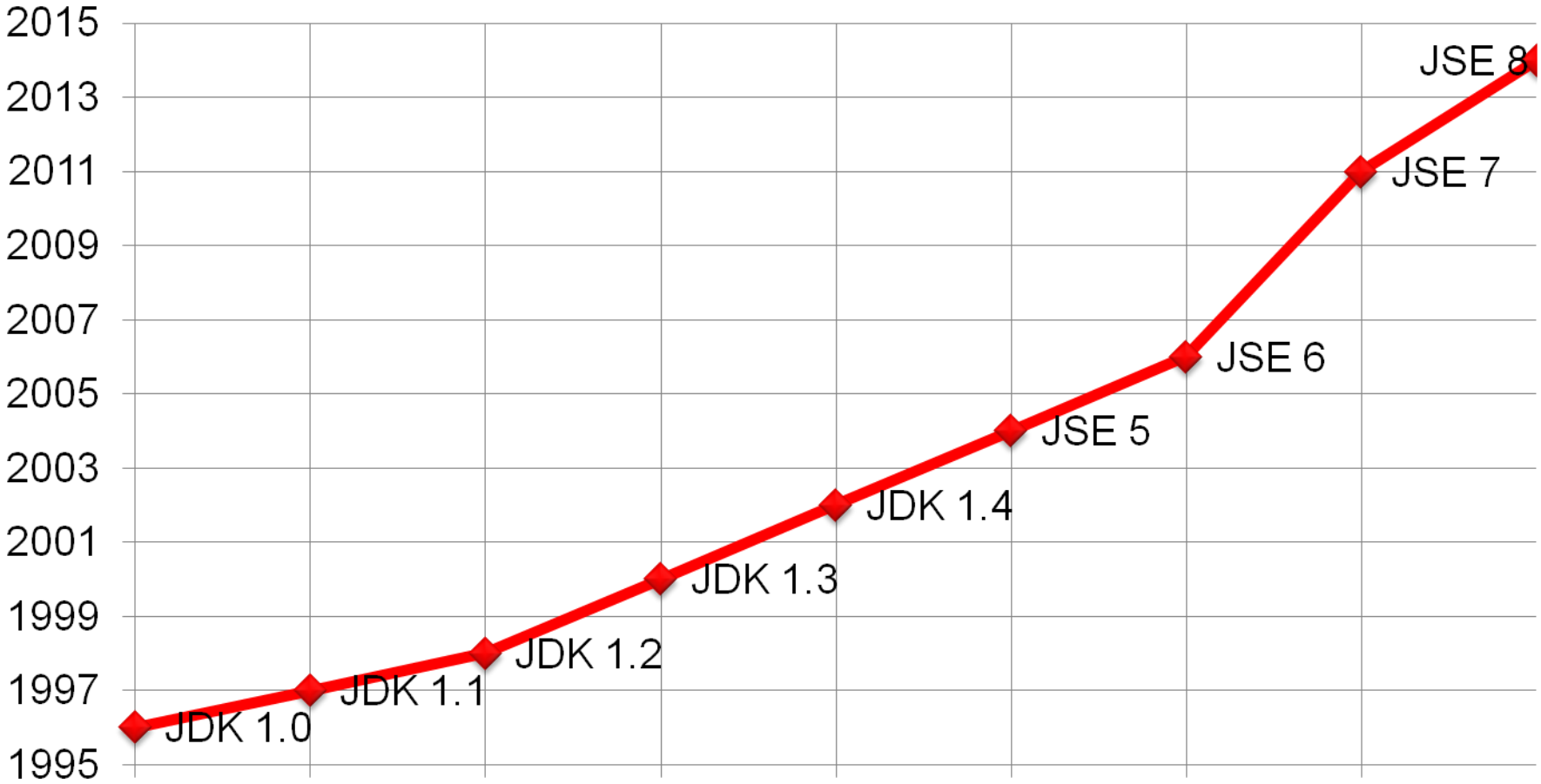
Standard Disclaimer

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

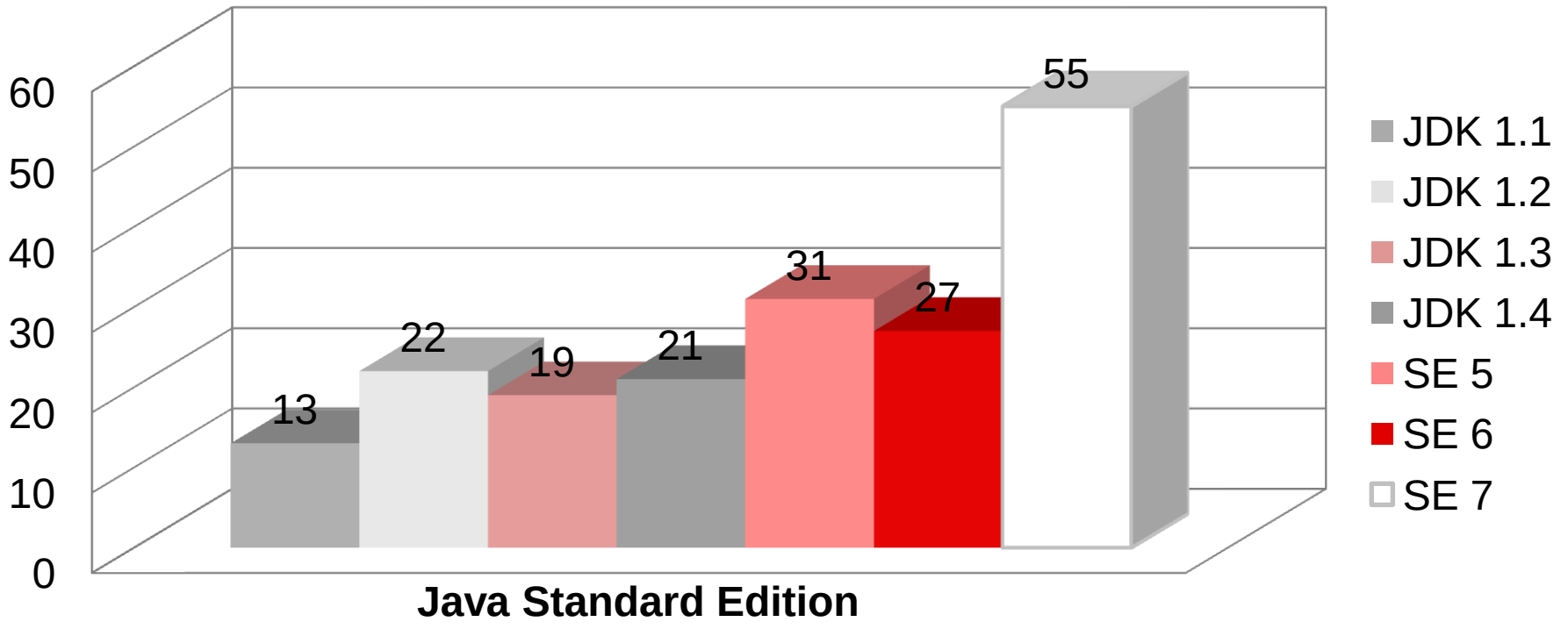
Agenda

- Explore the Past
- Analyze the Present
- Foresee the Future
- Claim victory and go eat lunch

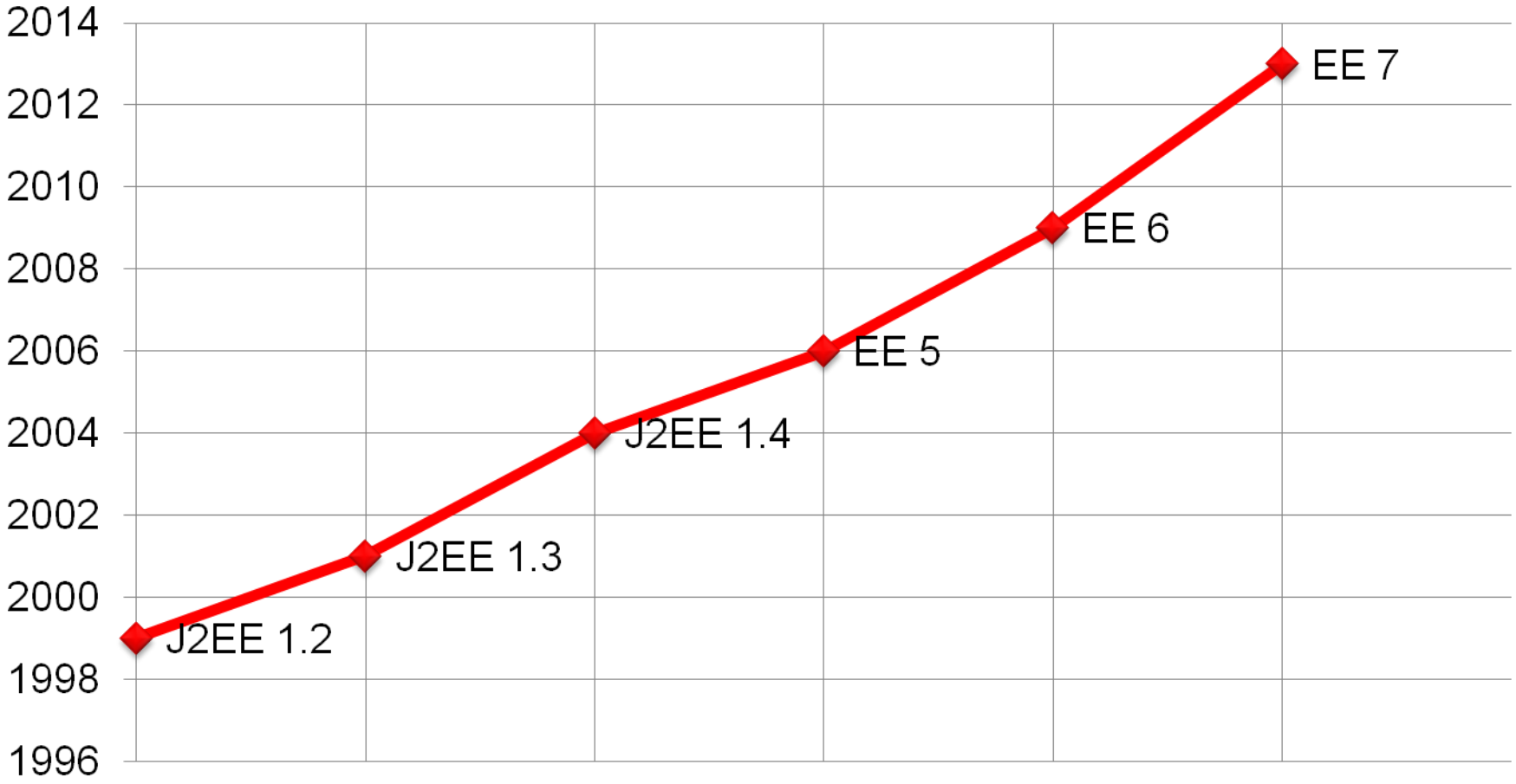
Java SE Timeline



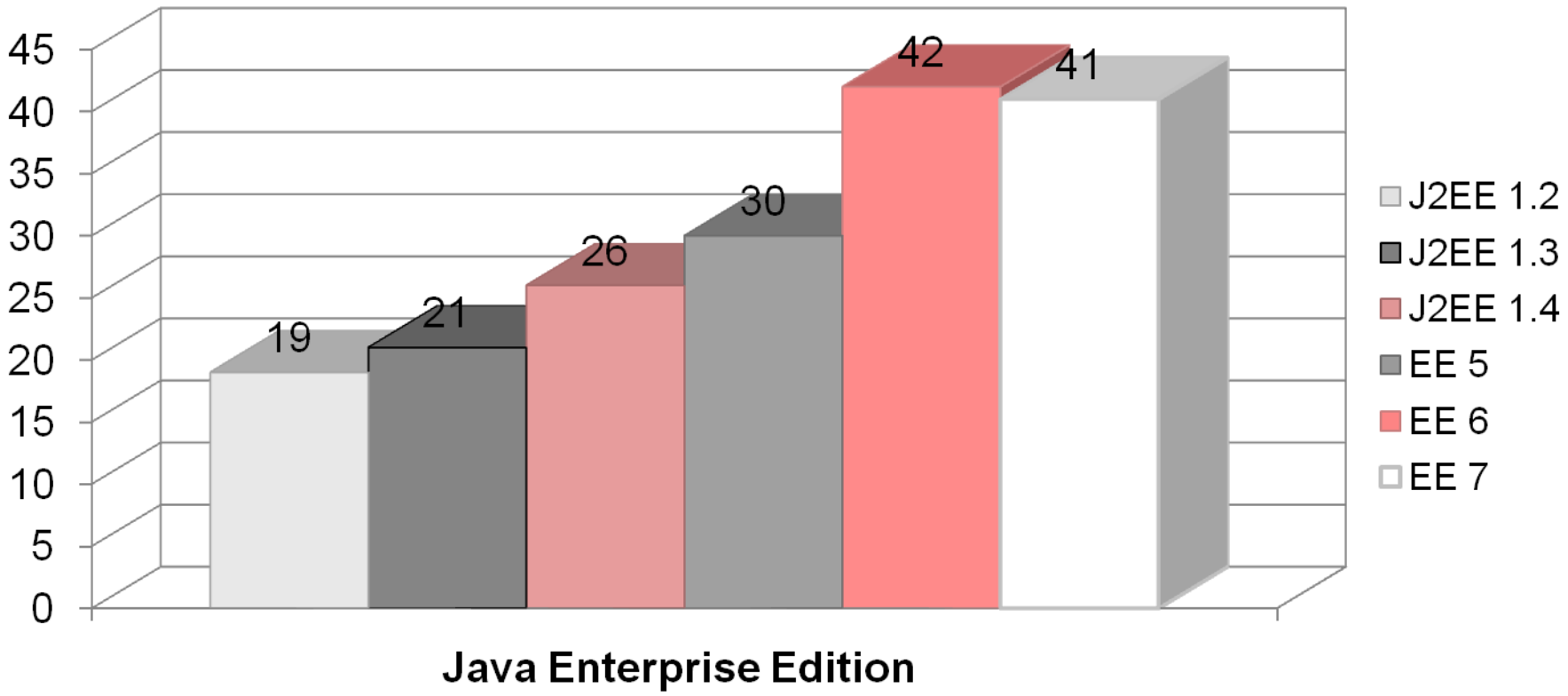
Java SE Release Cycle Length



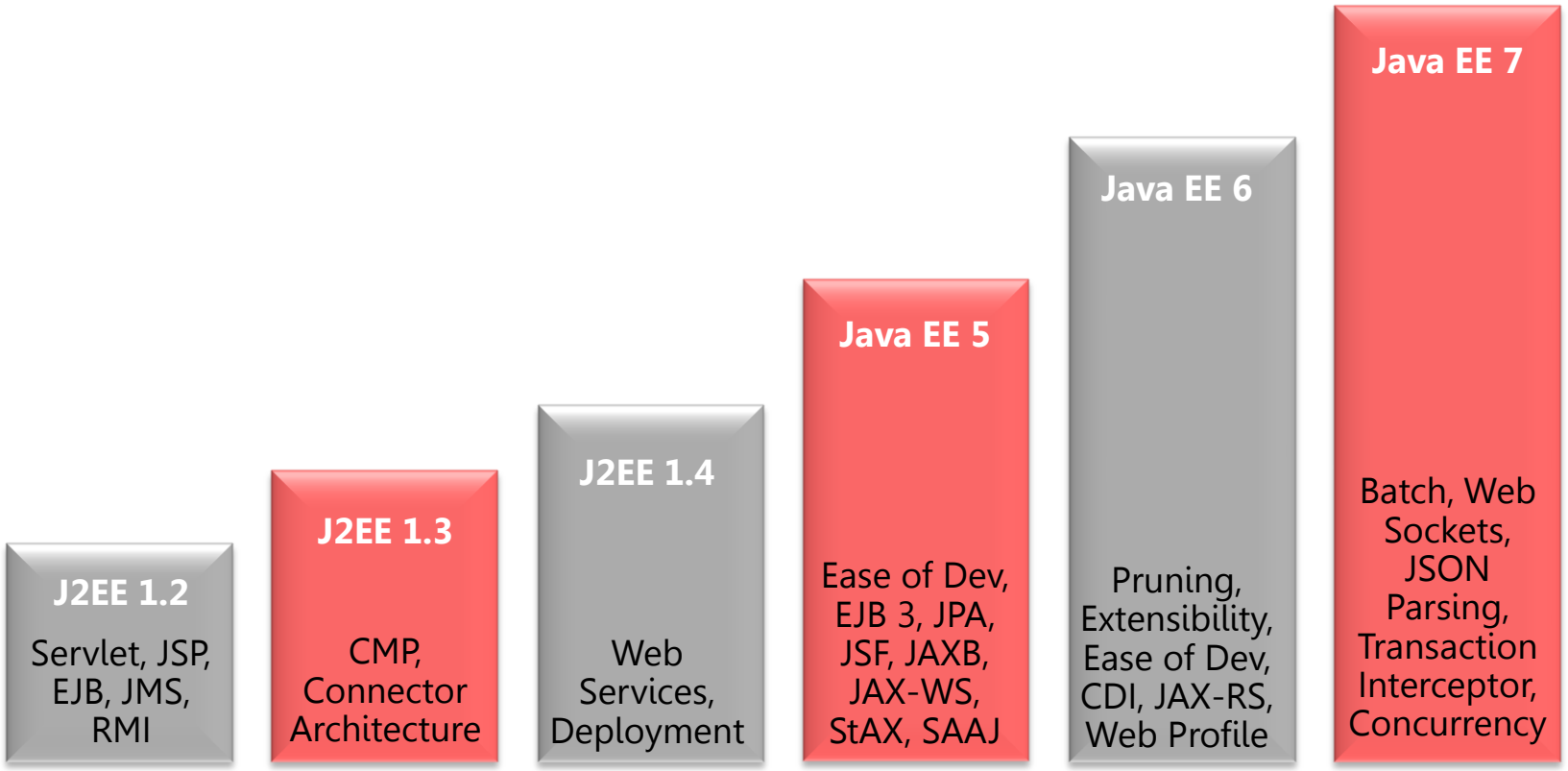
Java EE Timeline



Java EE Release Cycle Length



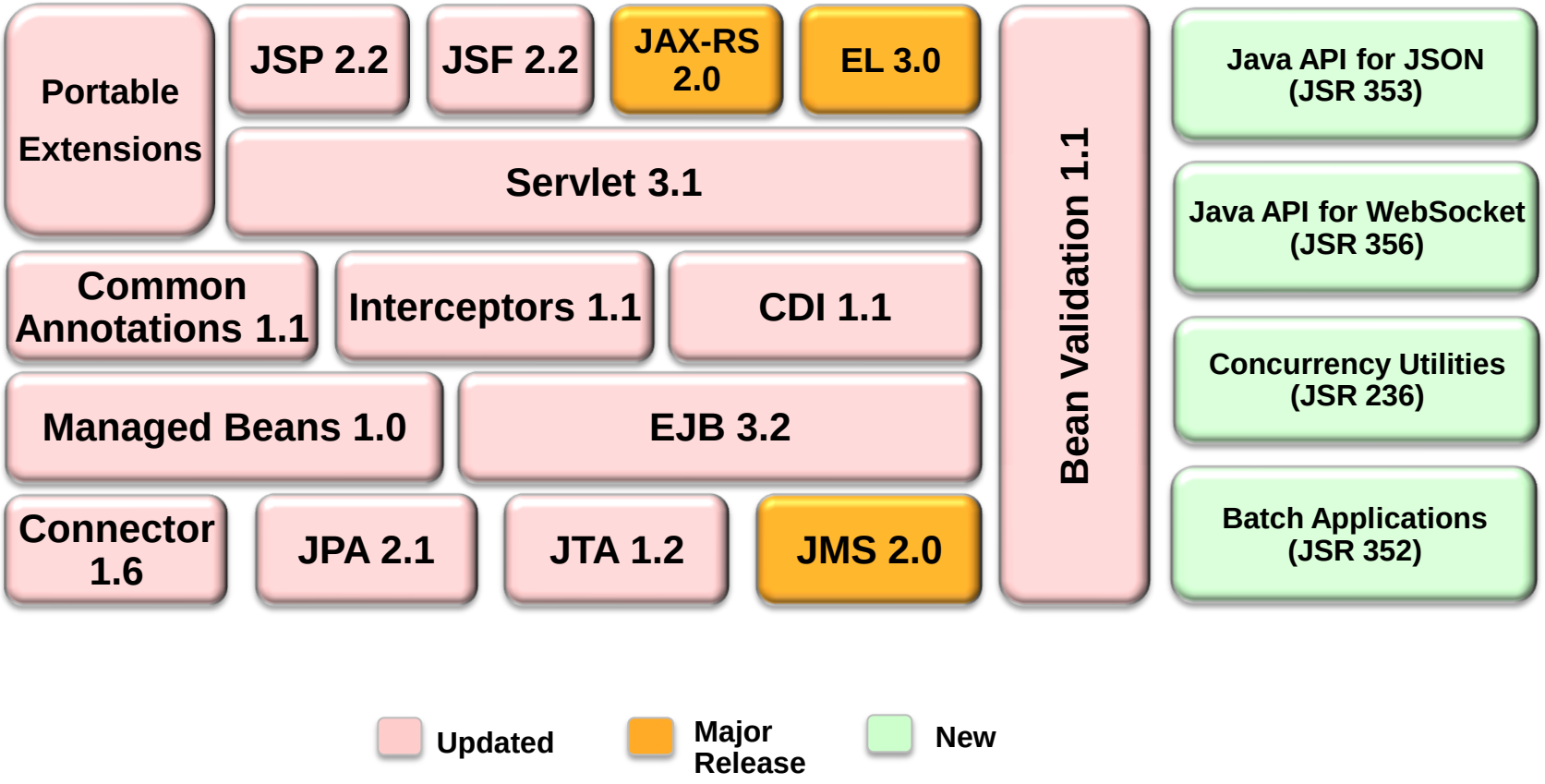
Java EE Release Features



Java EE 6 Success

- **40+ Million Java EE 6 Component Downloads**
- #1 Choice for Enterprise Developers
- #1 Application Development Platform
- Fastest implementation of a Java EE release

Java EE 7 Specifications



Java API for JSON Processing (JSON-P)

JavaScript Object Notation (JSON)

- Simple format for portable data
  - Fulfils same purpose as XML, but easier to read and write
- Started from JavaScript, now used by REST and Web Sockets, etc.
  - API's exported by social media (Twitter, Facebook Graph API, etc.)
- Limited set of types
  - Objects, arrays, key-value pairs, strings, numbers, booleans, null
- Simple syntax
  - { } around objects, [ ] around arrays, : between key and value
  - Comma separator

JSON Example

```
{
    "firstName": "Mike",
    "lastName": "Keith",
    "age": 21,
    "phoneNumbers": [
        { "type": "home",
          "number": "1-613-123-456" },
        { "type": "mobile",
          "number": "1-613-987-6543" }
    ]
}
```

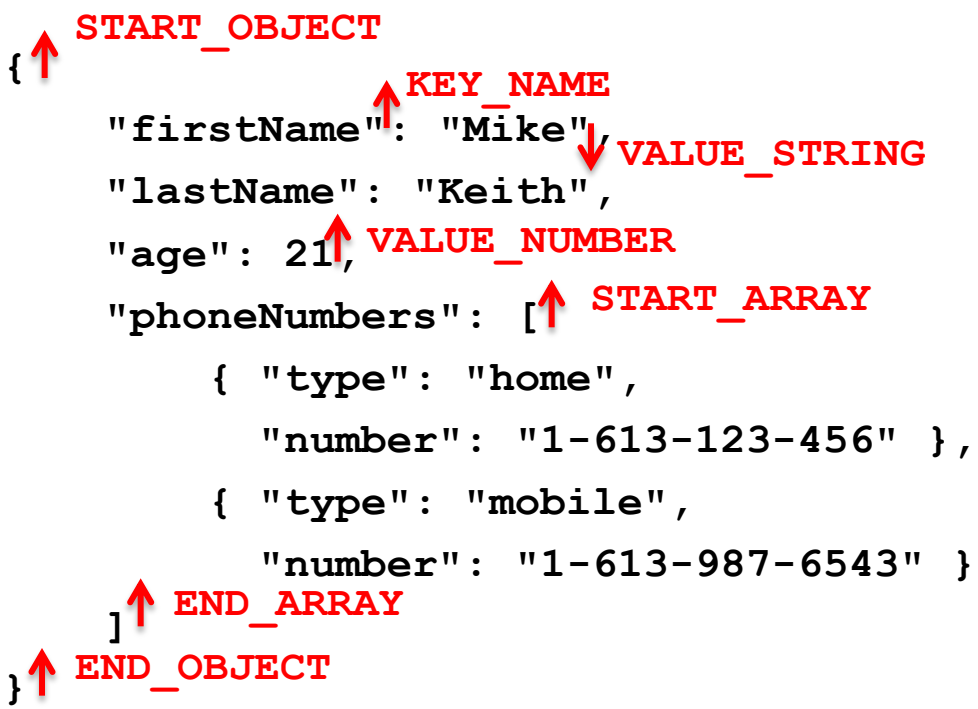
Java API for JSON Processing (JSON-P)

- API to parse and generate JSON
- Streaming API
  - Low-level, efficient way to parse/generate JSON
  - Provides pluggability for parsers/generators
- Object Model
  - Simple, easy-to-use high-level API (similar to XML DOM API)
  - Implemented on top of Streaming API
- JSON binding to Java objects (JSON-B) to come later

JSON-P 1.0 – Streaming API

- `JsonParser`
  - Parses JSON in a streaming way from input sources
    - Similar to StaX's `XMLStreamReader`, a pull parser
  - Created using
    - `Json.createParser(…)`
    - `Json.createParserFactory().createParser(…)`
  - Parser state events
    - `START_OBJECT, END_OBJECT`
    - `START_ARRAY, END_ARRAY`
    - `KEY_NAME, VALUE_STRING, VALUE_NUMBER, VALUE_TRUE, VALUE_FALSE, VALUE_NULL`

JSON-P 1.0 – Streaming API




```
{
    "firstName": "Mike",
    "lastName": "Keith",
    "age": 21,
    "phoneNumbers": [
        { "type": "home",
          "number": "1-613-123-456" },
        { "type": "mobile",
          "number": "1-613-987-6543" }
    ]
}
```

```
Iterator<Event> iter = parser.iterator();
iter.next(); iter.next();
String attributeName = parser.getString();
iter.next();
String attributeValue = parser.getString();
```

JSON-P 1.0 – Streaming API

- `JsonGenerator`
  - Generates JSON in a streaming way to output sources
    - Similar to StaX's XMLStreamWriter
  - Created using
    - `Json.createGenerator(…)`
    - `Json.createGeneratorFactory().createGenerator(…)`
  - Optionally, configured with features
    - E.g. for pretty printing

JSON-P 1.0 – Streaming API

```
{
    "firstName": "Mike",
    "lastName": "Keith",
    "age": 21
}
```

```
JsonGenerator jsonGen = Json.createGenerator(…);
jsonGen.writeStartObject()
       .write("firstname", "Mike")
       .write("lastname", "Keith")
       .write("age", 21)
       .writeEndObject();
jsonGen.close();
```

JSON-P 1.0 – Object Model API

- JsonObject/JsonArray – object and array structures
- JsonString/JsonNumber – string and number values
- JsonObjectBuilder – Builds JsonObjects
- JsonArrayBuilder – Builds JsonArrays
- JsonReader – Reads JsonObject and JsonArray from input source
- JsonWriter – Writes JsonObject and JsonArray to output source

JSON-P 1.0 – Object Model API

```
{
    "firstName": "Mike",
    "lastName": "Keith",
    "age": 21
}
```

```
JsonObject person = Json.createObjectBuilder()
   .add("firstName", "Mike")
   .add("lastName", "Keith")
   .add("age", 21)
   .build();
```

JSON-P 1.0 – Object Model API

- Reads JsonObject and JsonArray from input source
- Uses pluggable JsonParser
- Reads from InputStream or Reader

```
try (JsonReader reader = Json.createReader(input)) {
    JsonObject person = reader.readObject();
}
```

JSON-P 1.0 – Object Model API

- Writes JsonObject and JsonArray to output source
- Uses pluggable JsonGenerator
- Writes to an OutputStream or Writer

```
try (JsonWriter writer = Json.createWriter(output)) {
    writer.writeObject(person);
}
```

Culture Break – Beavers

- Can stay under water for 15 minutes at a time
- Use their tail as an alarm

- Second only to humans in ability to "modify" their environment

Java API for WebSockets

Java API for WebSockets 1.0

- API for WebSocket Client/Server Endpoints
- Annotation-driven or code-driven
- Server endpoint
  - `@ServerEndpoint` or subclass Endpoint class
- Client endpoint
  - `@ClientEndpoint` or subclass Endpoint class
- Integration with Java EE Web container
  - ServerContainer accessible as ServletContext attribute to permit dynamic registration of endpoints

Java API for WebSockets 1.0

```
import javax.websocket.OnMessage;
import javax.websocket.server.ServerEndpoint;

@ServerEndpoint("/hello")
public class HelloBean {

    @OnMessage
    public String sayHello(String name) {
        return “Hello “ + name;
    }
}
```

WebSocket Annotations

Annotation	Level	Purpose
`@ServerEndpoint`	class	Turns a POJO into a WebSocket Endpoint
`@ClientEndpoint`	class	Turns a POJO into a WebSocket Client
`@OnOpen`	method	Intercepts WebSocket connection Open events
`@OnClose`	method	Intercepts WebSocket connection Close events
`@OnMessage`	method	Intercepts WebSocket Message events
`@OnError`	method	Intercepts errors during a conversation
`@PathParam`	method parameter	Flags a matched path segment of a URI-template

@ServerEndpoint Elements

value	Relative URI or URI template e.g. “/hello” or “/chat/{subscriber-level}”
configurator	Custom configurator to use to create new instances of the endpoint
decoders	List of message decoder classes
encoders	List of message encoder classes
subprotocols	List of the names of the supported subprotocols

WebSocket Custom Payloads

```
@ServerEndpoint(
    value="/hello",
    encoders={MyMessage.class},
    decoders={MyMessage.class}
)
public class MyEndpoint {
    . . .
}
```

WebSocket Custom Payloads

```
public class MyMessage implements Decoder.Text<MyMessage>,
                                  Encoder.Text<MyMessage> {
  public JsonObject jsonObject;

  public void init(EndpointConfig config) {}
  public void destroy() {}

  public boolean willDecode(String string) { return true; }

  public MyMessage decode(String s) {
    jsonObject = Json.createReader(newStringReader(s))
                     .readObject();
    return this;

  }

  public String encode(MyMessage myMessage) {
    return myMessage.jsonObject.toString();
  }
}
```

WebSocket Client Side

```
@ClientEndpoint
public class HelloClient {
  @OnMessage
  public void message(String message, Session session) {
    // process message from server
  }
}


WebSocketContainer container = ContainerProvider
                                        .getWebSocketContainer();
container.connectToServer(HelloClient.class, "…/hello");
```

WebSocket Chat Example

```
@ServerEndpoint("/chat")
public class ChatBean {

    Set<Session> peers = Collections.synchronizedSet(…);

    @OnOpen
    public void open(Session peer) { peers.add(peer); }

    @OnClose
    public void close(Session peer) { peers.remove(peer); }

    @OnMessage
    public void message(String msg, Session client) {
        for (Session peer : peers) {
            peer.getRemote().sendObject(msg);
        }
    }
}
```

What is so good about Java EE?

- ✓ Very broad and healthy platform of technologies
- ✓ Mature and time-tested
- ✓ Standards offer application/knowledge portability
- ✓ Both corporate and grass roots support
- ✓ Continues to advance at a sustainable pace
- ✓ Web Profile offers relevant technology subset
- ✓ As easy to develop as any other technology

What is missing from Java EE?

- Temporary Caching (JSR 107)
- JSON Binding to Java objects
- No defined notion of a cluster
- Web framework still not very easy to use
- No support for NoSQL and Big Data
- Little in-container multitenancy support defined
- Still too hard for DevOps to deploy artifacts

What will be in Java EE 8 and Beyond?

Hopefully the expected stuff:

- JSON-B – Binding JSON to Java objects
- Temporary Caching
- Externalization of deployment configuration
- More features in the various subspecs

But also:

- Domain-specific services?
- Some support for cloud-based infrastructure?

Java EE – Still a Winner

Java EE still defines the most ubiquitous technologies used for enterprise software development on the planet!

More Information

- Java EE landing page
  oracle.com/us/javaee
- Java EE 7 Tutorial
  docs.oracle.com/javaee/7/tutorial/doc/
- Java EE 7 Reference Implementation
  glassfish.org
- The Aquarium
  blogs.oracle.com/theaquarium