

# Java API for XML Processing 1.1

## What's New

Edwin Goei  
Engineer, Sun Microsystems

# Introduction

- JAXP 1.0 emerged to fill deficiencies in existing industry standards: SAX 1.0 and DOM Level 1.
- Examples:
  - Bootstrapping a DOM tree
  - Controlling parser validation
- Since JAXP 1.0:
  - Industry standards changed: SAX 2.0, DOM Level 2
  - JAXP expanded to satisfy more needs: XSLT

# JAXP 1.1

- JAXP enables apps to parse and transform XML documents
- Allows apps to be independent of a particular implementation
- Augments existing SAX and DOM API standards

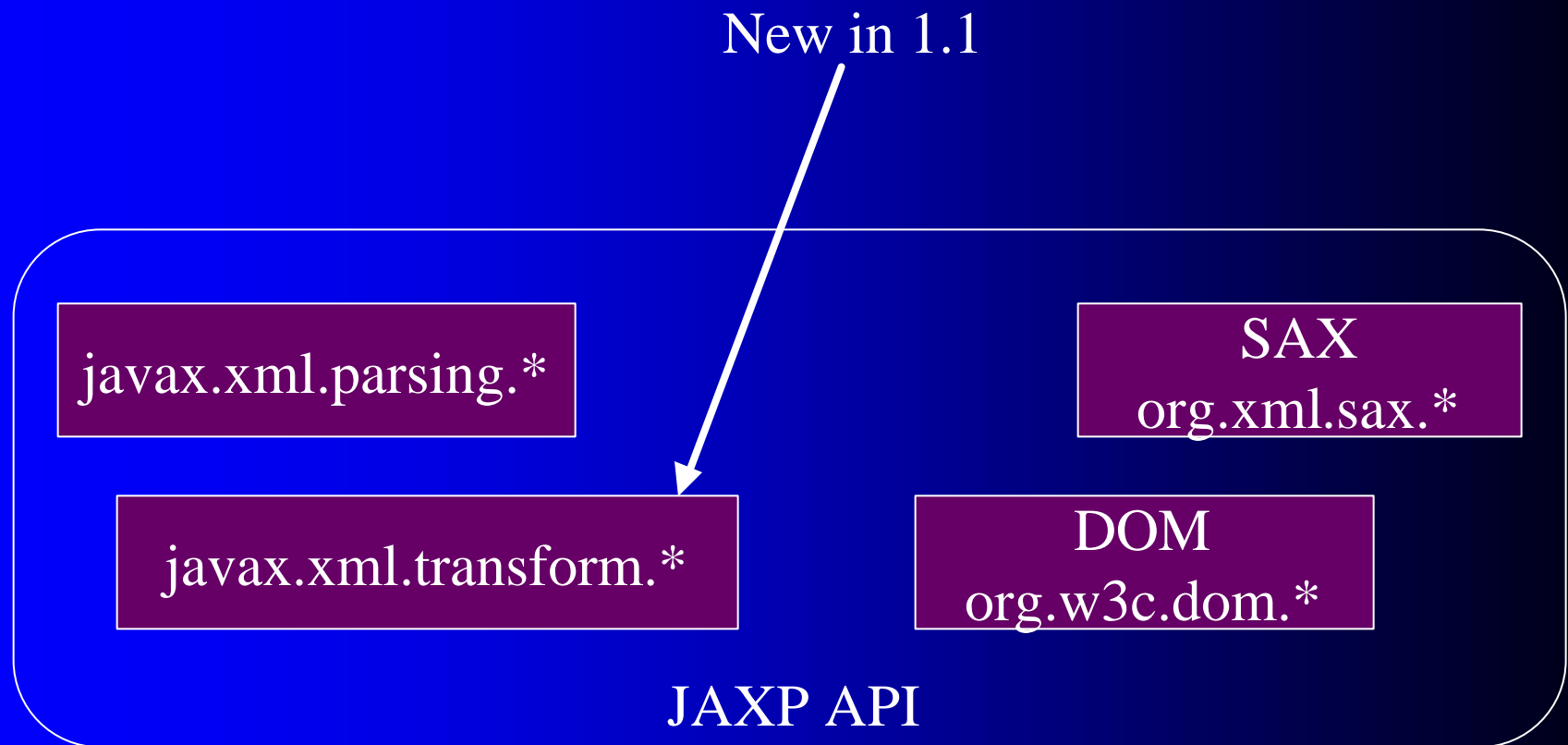
# New Features in 1.1

- Support for XSLT 1.0 based on TrAX (Transformation API for XML)
- Name change from “Parsing” to “Processing”
- Parsing API updated to SAX 2.0 and DOM Level 2
- Improved scheme to locate pluggable implementations

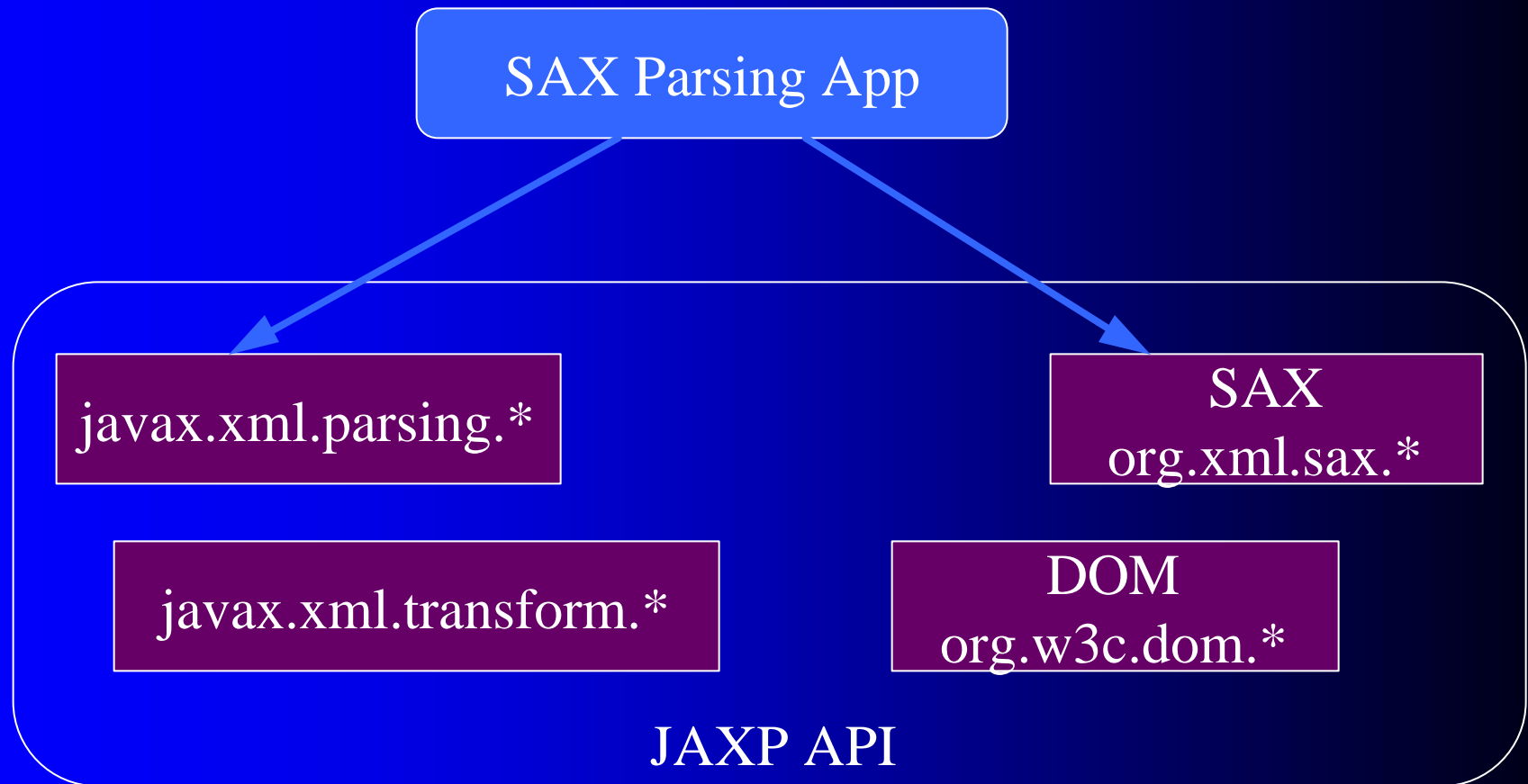
# Application Usage Categories

- Parsing using SAX 2.0
- Parsing using DOM Level 2
- Transformation using XSLT

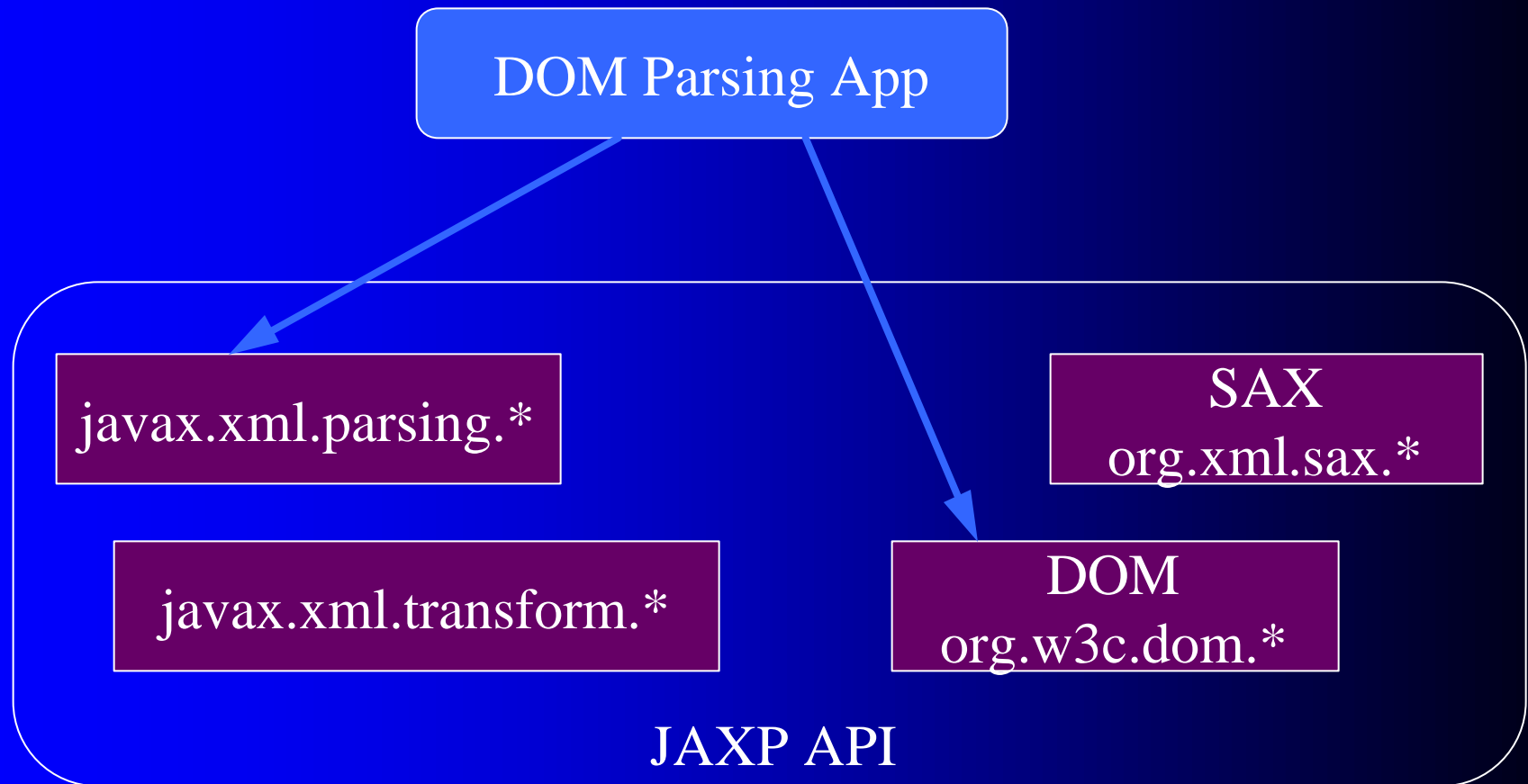
# JAXP 1.1 Components



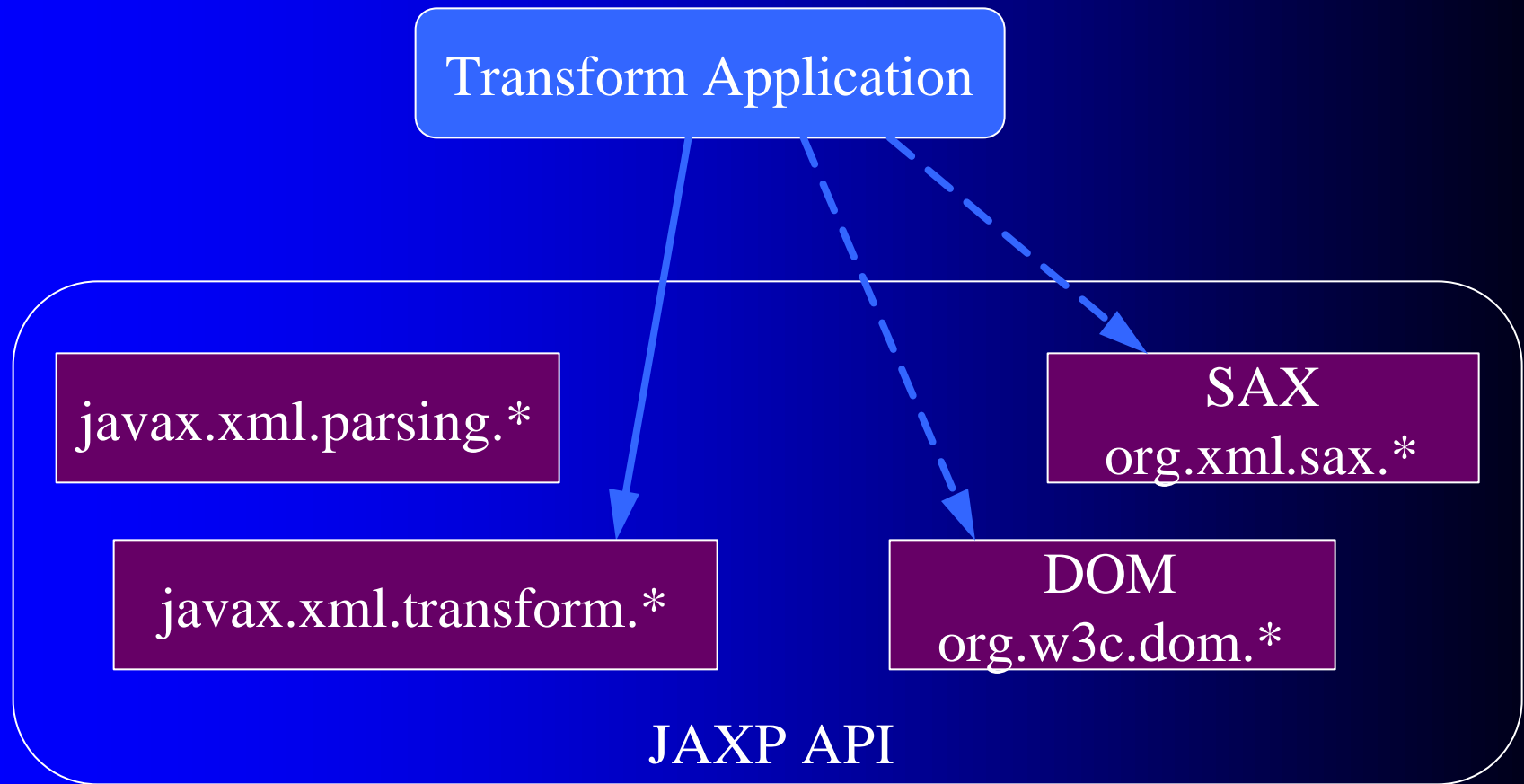
# 1) SAX Parsing Application



## 2) DOM Parsing Application



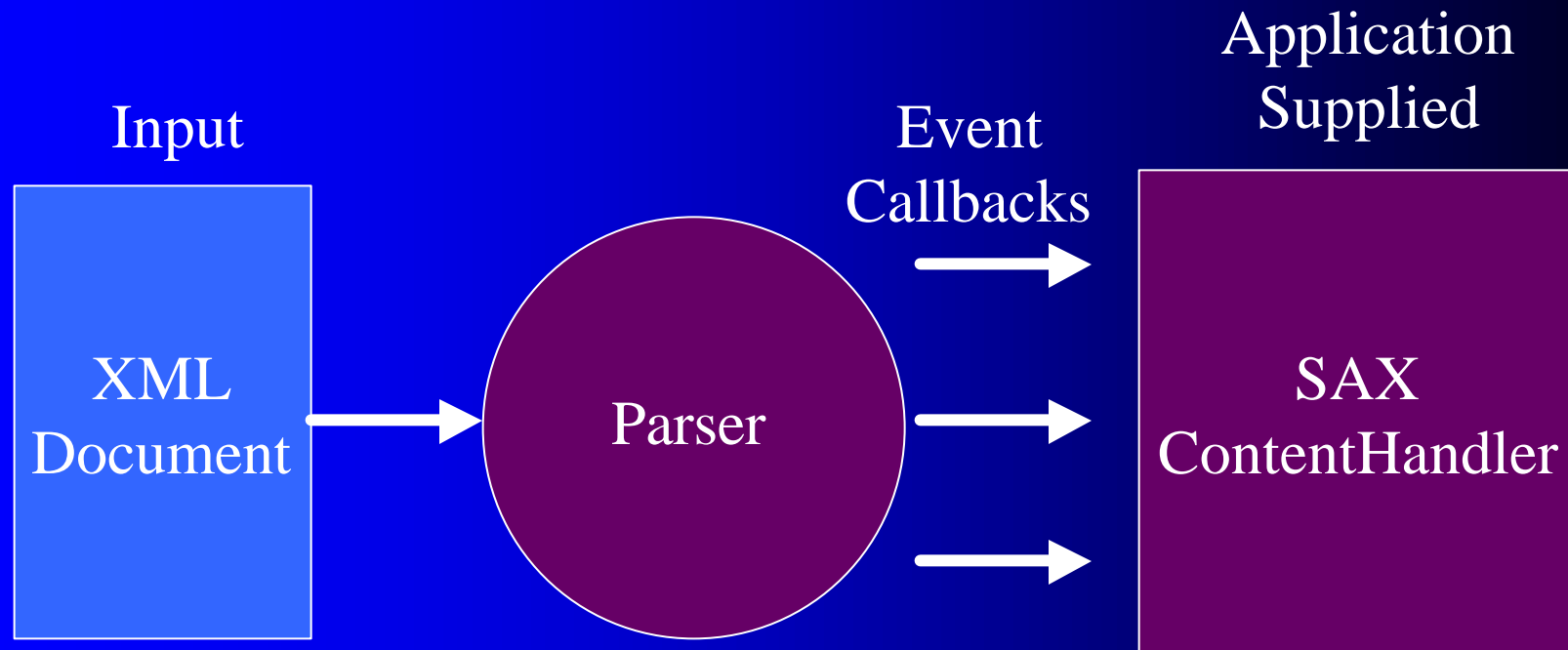
# 3) Transform Application



# 1) Parsing using SAX 2.0

- Application supplies a SAX ContentHandler to parser
- Application tells parser to start parsing a document
- Parser calls methods in the ContentHandler that application previously supplied during parse

# SAX 2.0 Parsing



# SAX ContentHandler

```
public interface ContentHandler {  
    void startElement(namespaceURI,  
        localName, qName, atts);  
    void endElement(namespaceURI,  
        localName, qName);  
    void characters(char[] ch, start, length);  
    ...  
}
```

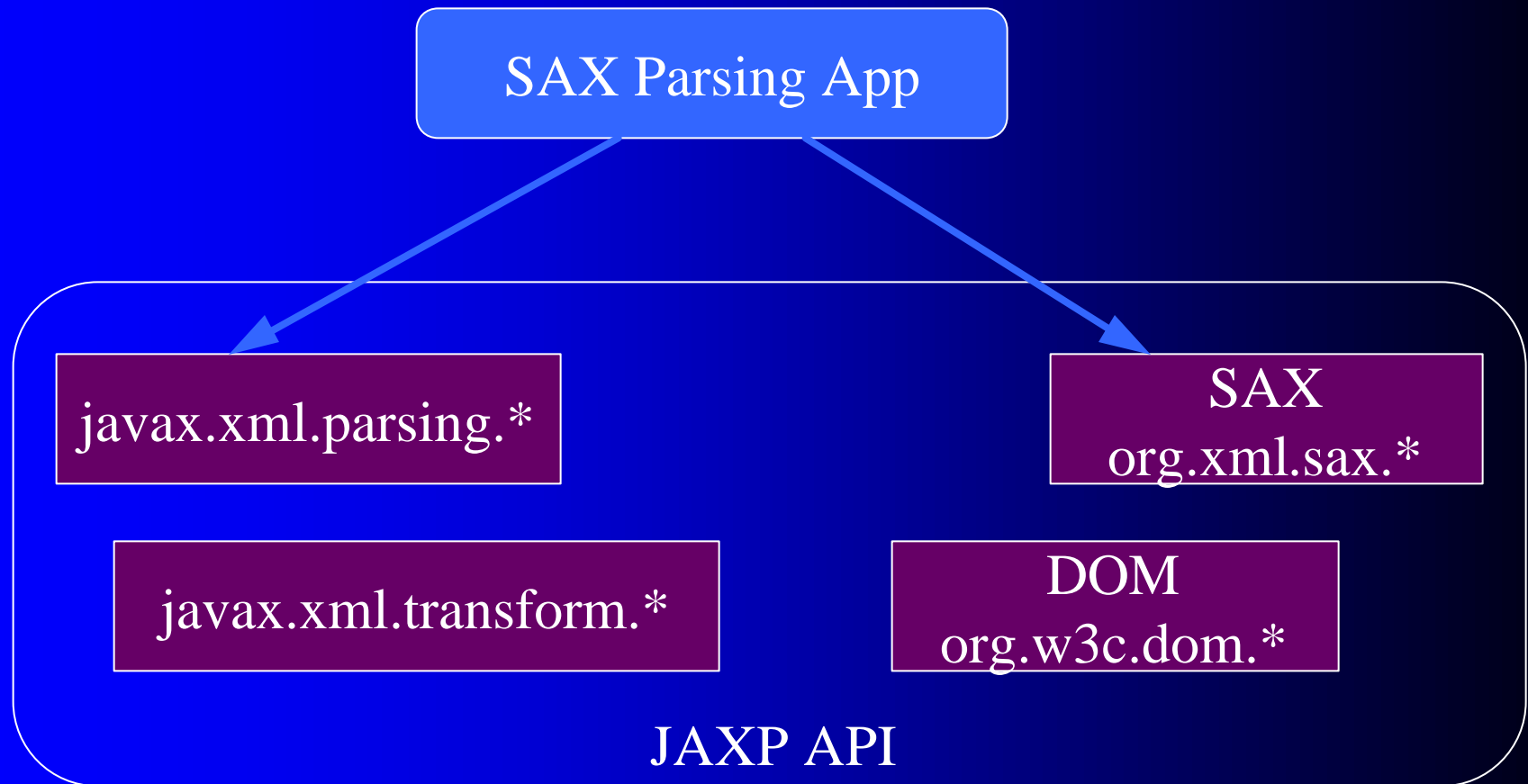
# Example: SAX2 Application

1. `XMLReader xmlReader = create SAX2 XMLReader instance`
2. `xmlReader.setContentHandler(myContentHandler);`
3. `xmlReader.parse(myInputSource);`

# Create XMLReader

```
SAXParserFactory spf =  
    SAXParserFactory.newInstance();  
  
// Change namespaces feature to SAX2 default  
spf.setNamespaceAware(true);  
  
// Create SAX XMLReader w/ SAX2 default features  
XMLReader xmlReader =  
    spf.newSAXParser().getXMLReader();  
  
// Use xmlReader as you would normally  
...
```

# SAX Parsing Application



# Changing the Implementation

- Define a system property:  
`javax.xml.parsers.SAXParserFactory`
- `$JAVA_HOME/jre/lib/jaxp.properties` file
- Jar Service Provider  
`META-INF/services/javax.xml.parsers.SAXParserFactory`
- Platform default (fallback)

# Jar Service Provider

- Jar file may contain a resource file called `.../javax.xml.parsers.SAXParserFactory` containing the name of a concrete class to instantiate
- JAXP static `SAXParserFactory.newInstance()` method searches classpath for resource and instantiates specified concrete class

# Examples: Using a Particular Implementation

With Java 2 version 1.3

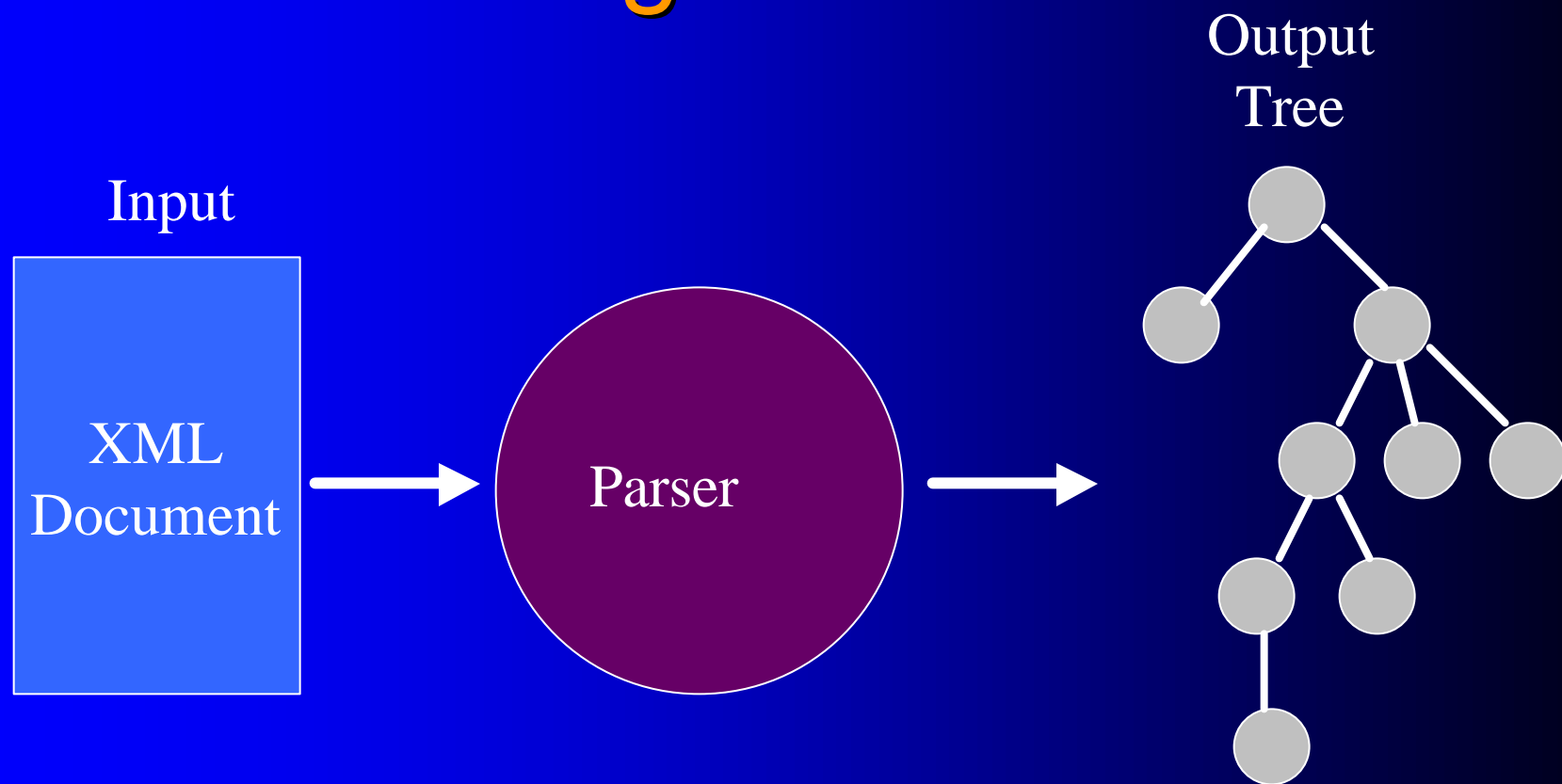
- To use Xerces, use classpath =
  - xerces.jar (contains all classes)
- To use Crimson, use classpath =
  - jaxp.jar (contains javax.xml.\*)
  - crimson.jar (contains sax, dom)
- You get Xerces, if classpath =
  - jaxp.jar
  - xerces.jar
  - crimson.jar

(Jar file names correct as of Feb 2001)

## 2) DOM Parsing Example

- Application gives DOM builder an XML document to parse
- Builder returns with a DOM Document object representing the DOM “tree”

# DOM Parsing



# JAXP Adds to DOM Level 2

- Method to “Load” a DOM Document object from an XML document\*
- Methods to control parser behavior such as validation and error handling\*
- Provides pluggable DOM parser implementation

\* Proposed for Level 3

# JAXP DOM Example

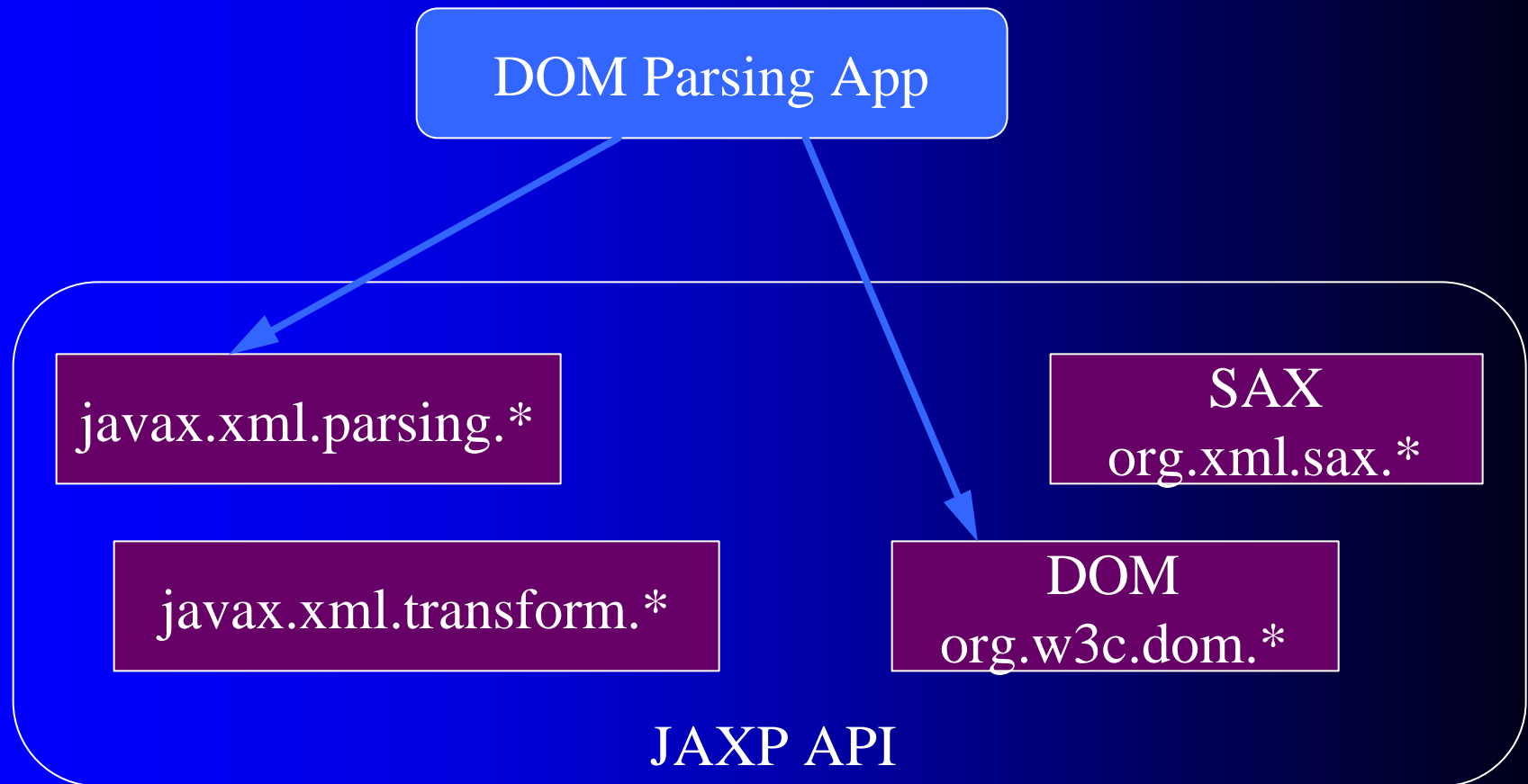
```
// Get platform default implementation
DocumentBuilderFactory dbf =
    DocumentBuilderFactory.newInstance();
// Set options
dbf.setNamespaceAware(true);
dbf.setValidating(true);

// Create new builder
DocumentBuilder db = dbf.newDocumentBuilder();

db.setErrorHandler(myErrorHandler);
Document doc =
    db.parse("http://server.com/foo.xml");

// Use doc with usual DOM methods
...
```

# DOM Parsing Application



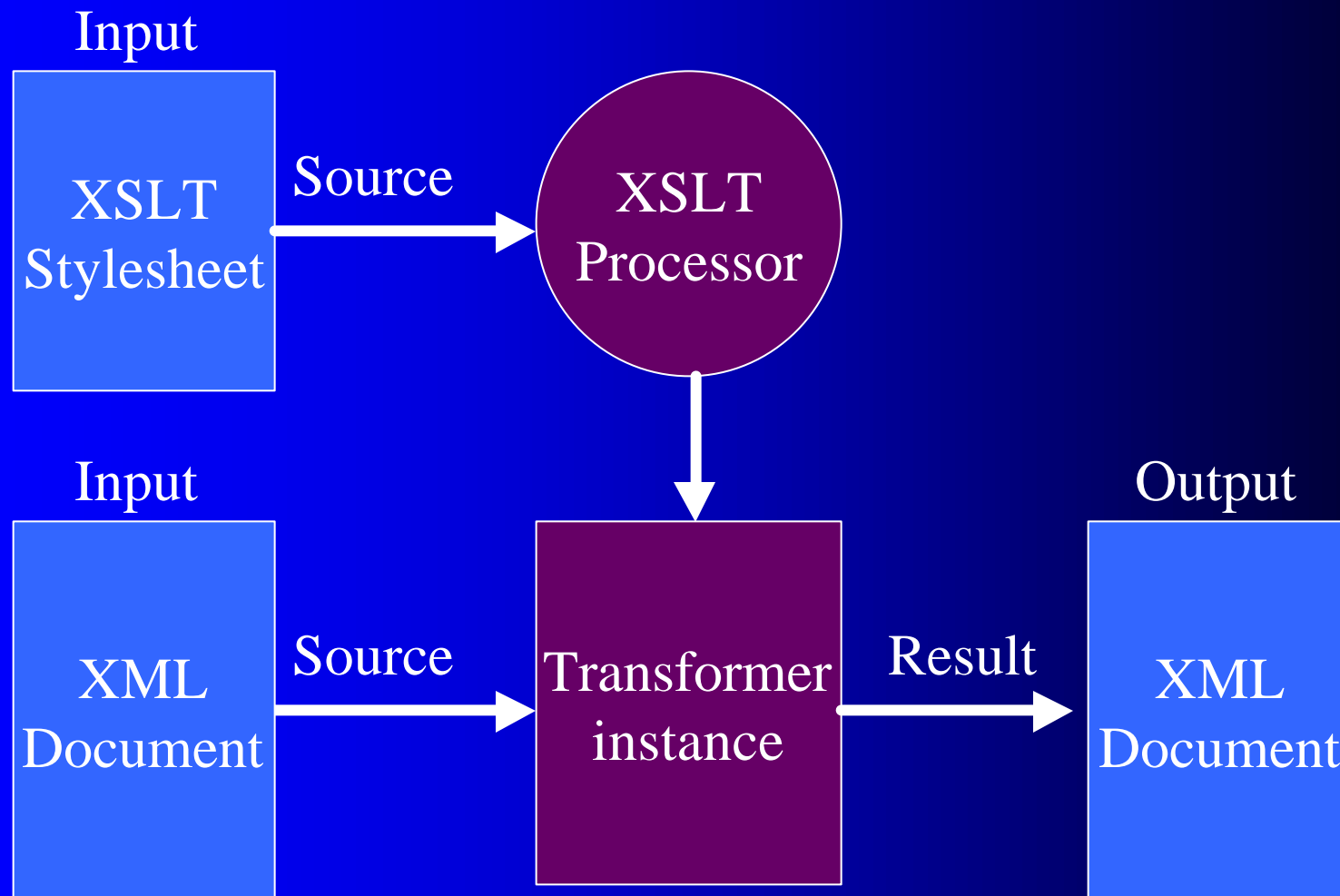
# JAXP DocumentBuilderFactory

- Has same pluggability mechanism as SAXParserFactory does

# 3) Transformation Using XSLT

- Major new feature
- Enables transformation of one XML document into another XML document using XSLT 1.0
- API based on TrAX, Transformation API for XML, initiated by Scott Boag, Michael Kay, and others. Incorporated into JAXP version 1.1.
- Provides pluggable Transform implementation similar to parsing

# Transformation Diagram



# Transformation Example

- Create Transform instance from XSLT stylesheet
- Use the Transform instance to transform the source document into a result document by calling:

```
Transform.transform(Source, Result)
```

# Transform Arguments

- Specialized implementations of generic `Source` and `Result` interfaces are in
  - `javax.xml.transform.stream`
  - `javax.xml.transform.sax`
  - `javax.xml.transform.dom`
- Different combinations of `Source` and `Result` can be passed to `transform()` method
- Examples: `StreamSource`, `DOMSource`, `SAXResult`, `StreamResult`

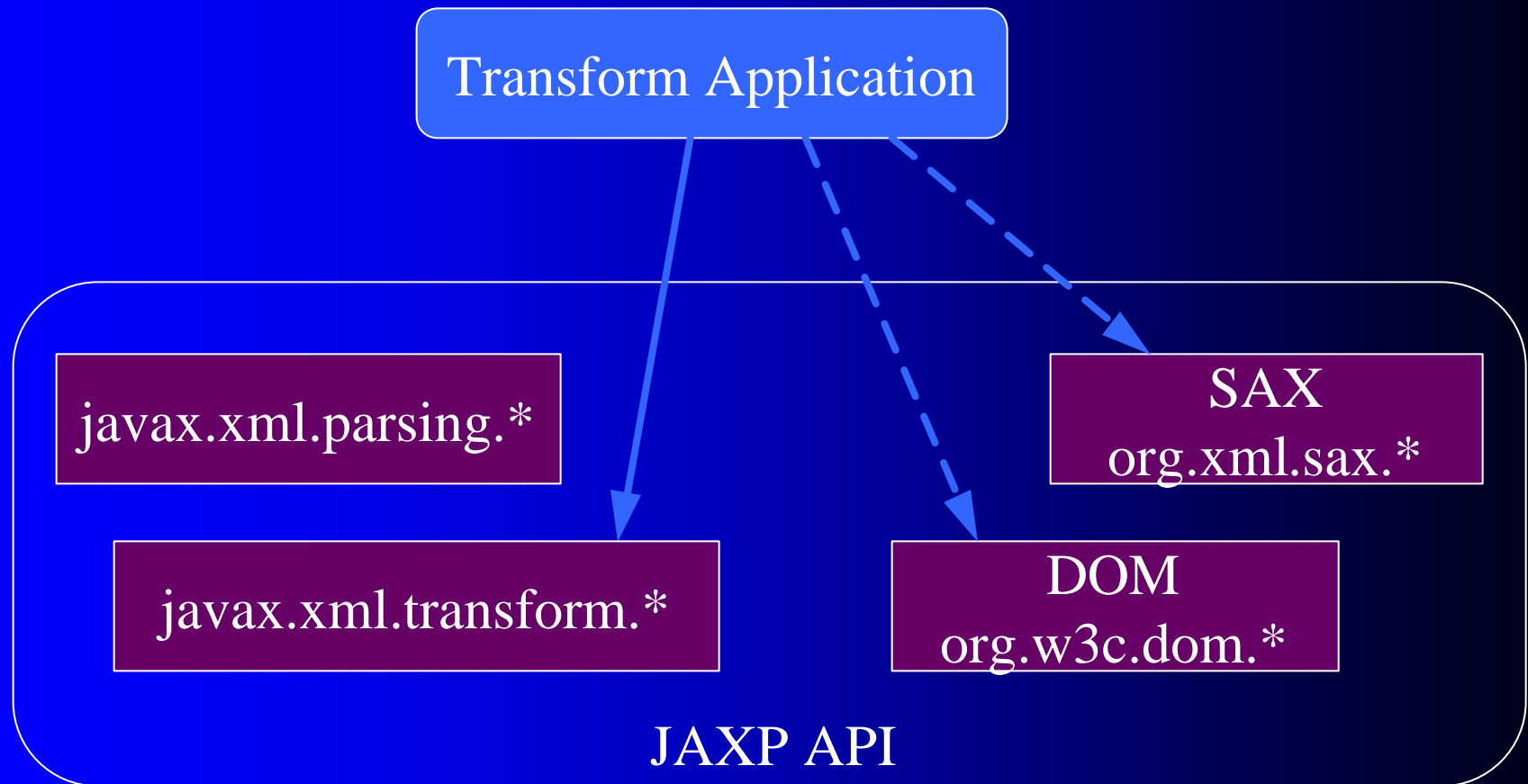
# Transform Example Code

```
// Get concrete implementation
TransformFactory tf =
    TransformFactory.newInstance();

// Create a transformer for a particular stylesheet
Transformer transformer = tf.newTransformer(
    new StreamSource(stylesheet));

// Transform input XML doc to System.out
transformer.transform(new StreamSource(sourceId),
    new StreamResult(System.out));
```

# Transform Application



# Example XSLT Stylesheet

```
<?xml version="1.0"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
  <xsl:template match="/">
    <html>
      <head>
        <title>Stock Quotes</title>
        <style type="text/css">
          <xsl:comment>
            body { background-color: white; }
          </xsl:comment>
        </style>
      </head>
```

# Example XSLT Stylesheet (cont)

```
<body>
  <center>
    <h1>Stock Quotes</h1>
  </center>
  <xsl:apply-templates/>
  
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

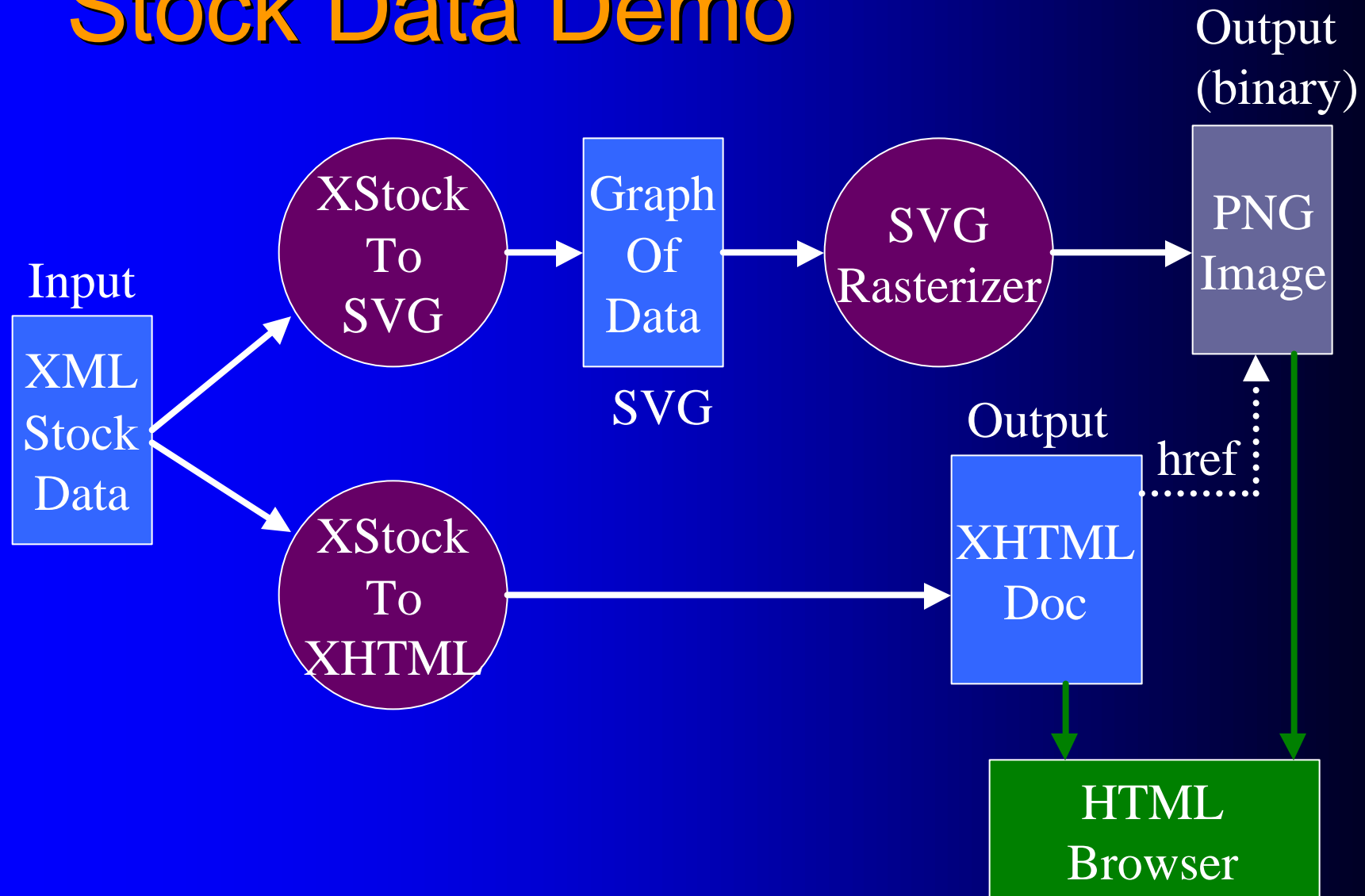
# Example: Output a DOM Tree

- How do I output a DOM tree as XML?
- Future: a requirement of DOM Level 3
- Can use a JAXP 1.1 transform to do this
  - Create an identity Transformer
  - Transform DOMSource to StreamResult:  
`identity.transform(domSource, streamResult)`

# Demonstration

- Transform stock data in some XML format into a document which can be viewed in an HTML browser.
- Use 2 transforms:
  - Stock data to SVG
  - Stock data to XHTML
- Rasterize SVG into image

# Stock Data Demo



# References

- My web page: [www.apache.org/~edwingo](http://www.apache.org/~edwingo)
- Apache XML projects: [xml.apache.org](http://xml.apache.org)
- SAX: [www.megginson.com/SAX](http://www.megginson.com/SAX)
- DOM: [www.w3c.org/DOM](http://www.w3c.org/DOM)
- JAXP specs, RI: [java.sun.com/xml](http://java.sun.com/xml)
- SAXON: [users.iclway.co.uk/mhkay/saxon](http://users.iclway.co.uk/mhkay/saxon)

# Questions