

Declarative Meta Programming

Andy Kellens (akellens@vub.ac.be)



In this lecture ...

- ▶ **Goal of Declarative Meta Programming**
- ▶ **Meta Programming**
- ▶ **SOUL**
- ▶ **Overview of the predicates in LiCoR**
- ▶ **Examples**
- ▶ **Detecting the composite design pattern**
- ▶ **Applications**

A bit of context

- ▶ **Ongoing research at VUB and UCL**
- ▶ **Started in 1996 (Wuyts)**
- ▶ **Group of about 10 people**
- ▶ **Common research artefact: the SOUL language**
- ▶ **But different goals:**
 - verifying architecture/design
 - program generation
 - declarative user interface specification
 - aspect-oriented language design
- ▶ **Common problem: supporting software evolution**

A bit of context

- ▶ **Ongoing research at VUB and UCL**
- ▶ **Started in 1996 (Wuyts)**
- ▶ **Group of about 10 people**
- ▶ **Common research artefact: the SOUL language**
- ▶ **But different goals:**
 - verifying architecture/design
 - program generation
 - declarative user interface specification
 - aspect-oriented language design
- ▶ **Common problem: supporting software evolution**

Documenting software ...

```
import java.io.*;
import java.util.*;

/**
 * Command line program to copy a file to another directory.
 * Author: Bruce Eckel
 */
public class CopyFile {
    // constant values for the override option
    public static final int OVERWRITE_ALWAYS = 1;

    // ... (rest of the code) ...
}

/**
 * Main method
 */
public static void main(String[] args) {
    // ... (rest of the main method) ...
}

/**
 * Copy file, optionally creating a checksum
 */
public static void copyFile(String srcFile, String destFile,
                           boolean overwrite, boolean checksum)
    throws IOException {
    // ... (rest of the copyFile method) ...
}

/**
 * Verify file
 */
public static boolean verifyFile(String srcFile, String destFile)
    throws IOException {
    // ... (rest of the verifyFile method) ...
}

/**
 * Print a message to standard output and read lines from
 * standard input until we see an EOF or the program
 * prints a message indicating that it is terminated by user.
 * Prints one message, then the file, then the file.
 */
public static void printMessage(String message)
    throws IOException {
    // ... (rest of the printMessage method) ...
}

/**
 * Read a message from standard input and read lines from
 * standard output until we see an EOF or the program
 * prints a message indicating that it is terminated by user.
 * Prints one message, then the file, then the file.
 */
public static void readMessage(String message)
    throws IOException {
    // ... (rest of the readMessage method) ...
}
}
```

```
public class FileDownload {
    public static void download(String address, String localFileName)
        throws IOException {
        OutputStream out = null;
        URLConnection conn = null;
        InputStream in = null;
        try {
            URL url = new URL(address);
            out = new BufferedOutputStream(new FileOutputStream(localFileName));
            conn = url.openConnection();
            in = conn.getInputStream();
            byte[] buffer = new byte[1024];
            int numRead;
            long numWritten = 0;
            while ((numRead = in.read(buffer)) != -1) {
                out.write(buffer, 0, numRead);
                numWritten += numRead;
            }
            System.out.println(localFileName + " (" + numWritten);
        } catch (Exception exception) {
            exception.printStackTrace();
        } finally {
            try {
                if (in != null)
                    in.close();
                if (out != null)
                    out.close();
            } catch (IOException ioe) {
            }
        }
    }

    // ... (rest of the class) ...
}

public static void main(String[] args) {
    for (int i = 0; i < args.length; i++)
        download(args[i]);
}
}
```

Calls to database
- after state change
- follow DB protocol

Documenting software ...

```
import java.io.*;
import java.util.*;

/**
 * Download the program to copy a file to another directory.
 * Author: Bruce Eckstein
 */
public class CopyFile {
    // constant values for the override option
    public static final int OVERWRITE_ALWAYS = 1;

    // ... (rest of the code) ...

    public static void main(String[] args) {
        if (args.length == 0) {
            System.out.println("Usage: CopyFile src-dir dest-dir");
            System.exit(1);
        }
        // ... (rest of the code) ...
    }
}
```

```
public class FileDownload {
    public static void download(String address, String localFileName) {
        OutputStream out = null;
        URLConnection conn = null;
        InputStream in = null;
        try {
            URL url = new URL(address);
            out = new BufferedOutputStream(new FileOutputStream(localFileName));
            conn = url.openConnection();
            in = conn.getInputStream();
            byte[] buffer = new byte[1024];
            int numRead;
            long numWritten = 0;
            while ((numRead = in.read(buffer)) != -1) {
                out.write(buffer, 0, numRead);
                numWritten += numRead;
            }
            System.out.println(localFileName + " (" + numWritten);
        } catch (Exception exception) {
            exception.printStackTrace();
        } finally {
            try {
                if (in != null) {
                    in.close();
                }
                if (out != null) {
                    out.close();
                }
            } catch (IOException ioe) {
            }
        }
    }

    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++) {
            download(args[i]);
        }
    }
}
```

Calls to database
- after state change
- follow DB protocol

Factory methods
- create objects
consistently

Documenting software ...

```
import java.io.*;
import java.util.*;

/**
 * Summed the program to copy a file to another directory.
 * Author: Bruce Eckel
 */
public class CopyFile { // constant values for the override option
    public static final int OVERWRITE_ALWAYS = 1;

    // ... (more code) ...

    // ... (more code) ...

    // ... (more code) ...

    // ... (more code) ...

    // ... (more code) ...

    // ... (more code) ...
}
```

```
public class FileDownload {
    public static void download(String address, String localFileName) {
        OutputStream out = null;
        URLConnection conn = null;
        InputStream in = null;

        // ... (more code) ...

        public static void m
        Cor
    }
}
```

Calls to database
er state change
ow DB protocol

Discover/document:
Regularities
Architecture
Idioms
Patterns
Conventions

Bugs
Bad Smells

actory methods
ate objects
istently

ding conventions
. all events prefixed
t*
ow for consistency

Evolution ...

```
import java.io.*;
import java.util.*;

/**
 * Command line program to copy a file to another directory.
 *
 * @author Bruce Eckel
 */
public class CopyFile {
    // constant values for the override option
    public static final int OVERRIDE_OPTION = 1;
    public static final int OVERRIDE_FORCE = 2;

    // program options initialized to default values
    private static int overrideOption = 0;
    private static boolean copyTimestamp = true;
    private static int overrideForce = OVERRIDE_FORCE;

    public static long copyFile(String srcFile, File destDir)
        throws IOException {
        // check that srcFile exists
        if (!new File(srcFile).exists())
            throw new FileNotFoundException(srcFile);
        // check that destDir exists
        if (!destDir.exists())
            destDir.mkdirs();

        // get the file name
        String fileName = srcFile.substring(srcFile.lastIndexOf("/") + 1);

        // get the destination file name
        String destFileName = destDir.getAbsolutePath() + fileName;

        // check if dest file exists
        File destFile = new File(destFileName);
        if (destFile.exists()) {
            if (overrideOption == OVERRIDE_FORCE) {
                // force override
                destFile.delete();
            } else if (overrideOption == OVERRIDE_OPTION) {
                // ask for override
                System.out.print("File exists. " +
                    "Override? (y/n): ");
                char ch = '\0';
                while (ch != 'y' && ch != 'n')
                    ch = Character.toLowerCase(System.in.read());
                if (ch == 'y')
                    destFile.delete();
            } else {
                // do not override
                return new IOException("File exists. " +
                    "Override required.");
            }
        }

        // copy the file
        try {
            FileInputStream src = new FileInputStream(srcFile);
            FileOutputStream dest = new FileOutputStream(destFileName);
            byte[] buffer = new byte[1024];
            int numRead;
            while ((numRead = src.read(buffer)) != -1)
                dest.write(buffer, 0, numRead);
            src.close();
            dest.close();

            // optionally copy timestamp
            if (copyTimestamp)
                copyTimestamp(srcFile, destFileName);
        } catch (IOException e) {
            throw e;
        }

        // optionally verify file
        if (overrideOption == OVERRIDE_OPTION)
            verifyFile(srcFile, destFileName);

        // print a message to standard output and read lines from
        // standard input until the user enters 'q' or the program
        // prints the source file size for the
        public static boolean readStandardOutputAndReadLinesFromStandardInput()
            throws IOException {
                System.out.println("Source file size: " +
                    new File(srcFile).length());
                System.out.println("Destination file size: " +
                    new File(destFileName).length());
                System.out.println("Copy timestamp: " +
                    copyTimestamp);
                System.out.println("Override option: " +
                    overrideOption);
                System.out.println("Override force: " +
                    overrideForce);
                System.out.println("Please enter 'q' to quit or 'c' for help.");
                while ((ch = System.in.read()) != 'q')
                    continue;
                return true;
            }

        // print a message to standard output and read lines from
        // standard input until the user enters 'q' or the program
        // prints the source file size for the
        public static boolean readStandardOutputAndReadLinesFromStandardInput()
            throws IOException {
                System.out.println("Source file size: " +
                    new File(srcFile).length());
                System.out.println("Destination file size: " +
                    new File(destFileName).length());
                System.out.println("Copy timestamp: " +
                    copyTimestamp);
                System.out.println("Override option: " +
                    overrideOption);
                System.out.println("Override force: " +
                    overrideForce);
                System.out.println("Please enter 'q' to quit or 'c' for help.");
                while ((ch = System.in.read()) != 'q')
                    continue;
                return true;
            }
    }

    // copy timestamp from srcFile to destFile
    private static void copyTimestamp(String srcFile, String destFile)
        throws IOException {
        File src = new File(srcFile);
        File dest = new File(destFile);
        if (src.exists() && dest.exists())
            dest.setLastModified(src.getLastModified());
    }

    // verify file
    private static void verifyFile(String srcFile, String destFile)
        throws IOException {
        File src = new File(srcFile);
        File dest = new File(destFile);
        if (src.exists() && dest.exists())
            if (src.length() != dest.length())
                System.out.println("File sizes do not match.");
            else if (src.lastModified() != dest.lastModified())
                System.out.println("File timestamps do not match.");
            else
                System.out.println("Files match.");
    }
}
```

```
public class FileDownload {
    public static void download(String address, String localFileName)
        throws IOException {
        URL url = new URL(address);
        HttpURLConnection conn = null;
        InputStream in = null;
        try {
            conn = (HttpURLConnection) url.openConnection();
            in = conn.getInputStream();
            byte[] buffer = new byte[1024];
            int numRead;
            long numWritten = 0;
            while ((numRead = in.read(buffer)) != -1) {
                out.write(buffer, 0, numRead);
                numWritten += numRead;
            }
            System.out.println(localFileName + " (" + numWritten +
                " bytes)");
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                in.close();
            } catch (IOException ioe) {
            }
        }
    }

    public static void download(String address)
        throws IOException {
        int lastSlashIndex = address.lastIndexOf('/');
        if (lastSlashIndex >= 0 &&
            lastSlashIndex < address.length() - 1)
            download(address, address.substring(lastSlashIndex + 1));
        else
            System.err.println("Could not figure out local file name for " +
                address);
    }

    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++)
            download(args[i]);
    }
}
```

Calls to database
- after state change
- follow DB protocol

Factory methods
- create objects
consistently

Coding conventions
- e.g. all events prefixed
Event*
- follow for consistency

Evolution ...

```
import java.io.*;
import java.util.*;
/**
 * named the program to copy a file to another directory.
 * Author: M. J. Heule
 */
public class CopyFile {
    // constant values for the overwrite option
    public static final int overwriteAlways = 1;
    public static final int overwriteNever = 2;
    public static final int overwriteIfNewer = 3;
    // program options initialized to default values
    private static int overwrite = overwriteAlways;
    private static boolean copyTimestamp = true;
    private static boolean copyPermissions = true;
    private static boolean copyOwnership = true;
    private static long overwriteTime = 0L;
    public static long overwriteTime(long srcTime, File destFile) {
        if (destFile.exists()) {
            long destTime = destFile.lastModified();
            if (overwrite == overwriteAlways) {
                return srcTime;
            } else if (overwrite == overwriteNever) {
                return destTime;
            } else if (overwrite == overwriteIfNewer) {
                return srcTime > destTime ? srcTime : destTime;
            }
        } else {
            return srcTime;
        }
    }
    private static boolean checkPermissions(File srcFile, File destFile) {
        if (!srcFile.canRead()) {
            System.err.println("Cannot read from source file.");
            return false;
        }
        if (!destFile.canWrite()) {
            System.err.println("Cannot write to destination file.");
            return false;
        }
        return true;
    }
    private static void copyFile(File srcFile, File destFile) {
        if (!checkPermissions(srcFile, destFile)) {
            return;
        }
        try {
            FileInputStream in = new FileInputStream(srcFile);
            FileOutputStream out = new FileOutputStream(destFile);
            byte[] buffer = new byte[1024];
            int numRead;
            while ((numRead = in.read(buffer)) != -1) {
                out.write(buffer, 0, numRead);
            }
            in.close();
            out.close();
        } catch (IOException ioe) {
            System.err.println("Cannot copy file: " + ioe);
        }
    }
    private static void copyPermissions(File srcFile, File destFile) {
        if (!copyOwnership(srcFile, destFile)) {
            return;
        }
        if (!copyPermissions(srcFile, destFile)) {
            return;
        }
        if (!copyTimestamp(srcFile, destFile)) {
            return;
        }
    }
    private static boolean copyOwnership(File srcFile, File destFile) {
        return true;
    }
    private static boolean copyPermissions(File srcFile, File destFile) {
        return true;
    }
    private static boolean copyTimestamp(File srcFile, File destFile) {
        return true;
    }
    public static void main(String[] args) {
        if (args.length < 2) {
            System.err.println("Usage: CopyFile src-dir dest-dir");
            return;
        }
        File srcDir = new File(args[0]);
        File destDir = new File(args[1]);
        if (!srcDir.isDirectory() || !destDir.isDirectory()) {
            System.err.println("Both a readable file and a writable file.");
            return;
        }
        File[] srcFiles = srcDir.listFiles();
        if (srcFiles == null) {
            System.err.println("Cannot list source directory.");
            return;
        }
        for (File srcFile : srcFiles) {
            File destFile = new File(destDir, srcFile.getName());
            copyFile(srcFile, destFile);
            copyPermissions(srcFile, destFile);
            copyOwnership(srcFile, destFile);
            copyTimestamp(srcFile, destFile);
        }
    }
}
```



```
public class FileDownload {
    public static void download(String address, String localFileName) {
        OutputStream out = null;
        URLConnection conn = null;
        InputStream in = null;
        try {
            URL url = new URL(address);
            out = new BufferedOutputStream(new FileOutputStream(localFileName));
            conn = url.openConnection();
            in = conn.getInputStream();
            byte[] buffer = new byte[1024];
            int numRead;
            long numWritten = 0;
            while ((numRead = in.read(buffer)) != -1) {
                out.write(buffer, 0, numRead);
                numWritten += numRead;
            }
            System.out.println(localFileName + " (" + numWritten +
                " bytes) downloaded.");
        } catch (Exception exception) {
            exception.printStackTrace();
        } finally {
            try {
                if (in != null) {
                    in.close();
                }
                if (out != null) {
                    out.close();
                }
            } catch (IOException ioe) {
                ioe.printStackTrace();
            }
        }
    }
    public static void download(String address) {
        int lastSlashIndex = address.lastIndexOf('/');
        if (lastSlashIndex >= 0 && lastSlashIndex < address.length() - 1) {
            download(address, address.substring(lastSlashIndex + 1));
        } else {
            System.err.println("Could not figure out local file name for " +
                address);
        }
    }
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++) {
            download(args[i]);
        }
    }
}
```

Calls to database
- after state change
- follow DB protocol

Factory methods
- create objects consistently

Coding conventions
- e.g. all events prefixed Event*
- follow for consistency

```
/**
 * named the program to copy a file to another directory.
 * Author: M. J. Heule
 */
public class CopyFile {
    // constant values for the overwrite option
    public static final int overwriteAlways = 1;
    public static final int overwriteNever = 2;
    public static final int overwriteIfNewer = 3;
    // program options initialized to default values
    private static int overwrite = overwriteAlways;
    private static boolean copyTimestamp = true;
    private static boolean copyPermissions = true;
    private static boolean copyOwnership = true;
    private static long overwriteTime = 0L;
    public static long overwriteTime(long srcTime, File destFile) {
        if (destFile.exists()) {
            long destTime = destFile.lastModified();
            if (overwrite == overwriteAlways) {
                return srcTime;
            } else if (overwrite == overwriteNever) {
                return destTime;
            } else if (overwrite == overwriteIfNewer) {
                return srcTime > destTime ? srcTime : destTime;
            }
        } else {
            return srcTime;
        }
    }
    private static boolean checkPermissions(File srcFile, File destFile) {
        if (!srcFile.canRead()) {
            System.err.println("Cannot read from source file.");
            return false;
        }
        if (!destFile.canWrite()) {
            System.err.println("Cannot write to destination file.");
            return false;
        }
        return true;
    }
    private static void copyFile(File srcFile, File destFile) {
        if (!checkPermissions(srcFile, destFile)) {
            return;
        }
        try {
            FileInputStream in = new FileInputStream(srcFile);
            FileOutputStream out = new FileOutputStream(destFile);
            byte[] buffer = new byte[1024];
            int numRead;
            while ((numRead = in.read(buffer)) != -1) {
                out.write(buffer, 0, numRead);
            }
            in.close();
            out.close();
        } catch (IOException ioe) {
            System.err.println("Cannot copy file: " + ioe);
        }
    }
    private static void copyPermissions(File srcFile, File destFile) {
        if (!copyOwnership(srcFile, destFile)) {
            return;
        }
        if (!copyPermissions(srcFile, destFile)) {
            return;
        }
        if (!copyTimestamp(srcFile, destFile)) {
            return;
        }
    }
    private static boolean copyOwnership(File srcFile, File destFile) {
        return true;
    }
    private static boolean copyPermissions(File srcFile, File destFile) {
        return true;
    }
    private static boolean copyTimestamp(File srcFile, File destFile) {
        return true;
    }
    public static void main(String[] args) {
        if (args.length < 2) {
            System.err.println("Usage: CopyFile src-dir dest-dir");
            return;
        }
        File srcDir = new File(args[0]);
        File destDir = new File(args[1]);
        if (!srcDir.isDirectory() || !destDir.isDirectory()) {
            System.err.println("Both a readable file and a writable file.");
            return;
        }
        File[] srcFiles = srcDir.listFiles();
        if (srcFiles == null) {
            System.err.println("Cannot list source directory.");
            return;
        }
        for (File srcFile : srcFiles) {
            File destFile = new File(destDir, srcFile.getName());
            copyFile(srcFile, destFile);
            copyPermissions(srcFile, destFile);
            copyOwnership(srcFile, destFile);
            copyTimestamp(srcFile, destFile);
        }
    }
}
```

Evolution ...

```
import java.io.*;
import java.net.*;

/** Read the program to copy a file to another directory.
 * Factory Method example
 */
public class CopyFile {
    // constant values for the override option
    public static final int OVERWRITE_OPTION = 1;
    public static final int OVERWRITE_COPY = 2;
    public static final int OVERWRITE_COPY_TIMESTAMP = 3;

    // program options initialized to default values
    private static int overwriteOption = 1;
    private static boolean copyTimestamp = true;
    private static boolean overwriteCopy = false;
    private static int overwriteOption = OVERWRITE_COPY;

    public static void copyFile(String srcFile, String destFile) {
        // check if source file exists
        File srcFileObj = new File(srcFile);
        if (!srcFileObj.exists()) {
            System.out.println("Source file does not exist.");
            return;
        }

        // check if destination file exists
        File destFileObj = new File(destFile);
        if (destFileObj.exists()) {
            if (overwriteOption == OVERWRITE_COPY) {
                System.out.println("Destination file exists. Overwriting copy.");
            } else if (overwriteOption == OVERWRITE_COPY_TIMESTAMP) {
                System.out.println("Destination file exists. Overwriting copy with timestamp.");
            } else if (overwriteOption == OVERWRITE_OPTION) {
                System.out.println("Destination file exists. Overwriting option.");
            }
        }

        // create a new file object for the destination
        File destFileObj = new File(destFile);
        if (!destFileObj.exists()) {
            System.out.println("Destination file does not exist. Creating.");
            destFileObj.createNewFile();
        }

        // copy the file
        try {
            FileInputStream fis = new FileInputStream(srcFile);
            FileOutputStream fos = new FileOutputStream(destFile);
            byte[] buffer = new byte[1024];
            int bytesRead;
            while ((bytesRead = fis.read(buffer)) != -1) {
                fos.write(buffer, 0, bytesRead);
            }
            fis.close();
            fos.close();
        } catch (IOException e) {
            System.out.println("Error copying file: " + e.getMessage());
        }

        // optionally verify the copy
        if (overwriteOption == OVERWRITE_COPY_TIMESTAMP) {
            verifyCopy(srcFile, destFile);
        }
    }

    private static void verifyCopy(String srcFile, String destFile) {
        File srcFileObj = new File(srcFile);
        File destFileObj = new File(destFile);
        if (!srcFileObj.exists() || !destFileObj.exists()) {
            System.out.println("One or both files do not exist.");
            return;
        }
        if (srcFileObj.length() != destFileObj.length()) {
            System.out.println("Files do not have the same size.");
            return;
        }
        if (!srcFileObj.equals(destFileObj)) {
            System.out.println("Files do not have the same content.");
            return;
        }
        System.out.println("Files are identical.");
    }

    public static void main(String[] args) {
        if (args.length != 2) {
            System.out.println("Usage: java CopyFile <srcFile> <destFile>");
            return;
        }
        copyFile(args[0], args[1]);
    }
}
```



```
public class FileDownload {
    public static void download(String address, String localFileName) {
        OutputStream out = null;
        URLConnection conn = null;
        InputStream in = null;
        try {
            URL url = new URL(address);
            out = new BufferedOutputStream(new FileOutputStream(localFileName));
            conn = url.openConnection();
            in = conn.getInputStream();
            byte[] buffer = new byte[1024];
            int numRead;
            long numWritten = 0;
            while ((numRead = in.read(buffer)) != -1) {
                out.write(buffer, 0, numRead);
                numWritten += numRead;
            }
            System.out.println(localFileName + "\t" + numWritten);
        } catch (Exception exception) {
            exception.printStackTrace();
        } finally {
            try {
                if (in != null) in.close();
                if (out != null) out.close();
            } catch (IOException ioe) {
                ioe.printStackTrace();
            }
        }
    }

    public static void download(String address, String localFileName) {
        int lastSlashIndex = address.lastIndexOf('/');
        if (lastSlashIndex == -1) {
            lastSlashIndex = address.length() - 1;
        } else {
            lastSlashIndex = address.substring(lastSlashIndex + 1);
        }
        System.out.println("Could not figure out local file name for " + address);
    }

    public static void main(String[] args) {
        if (args.length != 2) {
            System.out.println("Usage: java FileDownload <address> <localFileName>");
            return;
        }
        download(args[0], args[1]);
    }
}
```

Calls to database
- after state change
- follow DB protocol

Factory methods
- create objects consistently

Coding conventions
- e.g. all events prefixed Event*
- follow for consistency

???

???

???

Evolution ...

```

import java.io.*;
import java.util.*;

/**
 * Revised the program to copy a file to another directory.
 * @author Bruce Eckel
 */
public class CopyFile {
    // constant values for the override option
    public static final int OVERWRITE_OPTION = 1;
    public static final int COMMIT_NAME = 2;
    public static final int COMMIT_COPY = 3;

    // program options initialized to default values
    private static int overrideOption = 0;
    private static String commitName = null;
    private static boolean copyWithTimestamp = true;
    private static boolean commitCopy = true;
    private static int commitOption = OVERWRITE_OPTION;

    public static long copyFile(String srcFile, File destDir)
        throws IOException {
        // check if destDir is a directory
        if (!destDir.isDirectory())
            throw new IOException("Destination is not a directory");
        // check if srcFile is a file
        if (!srcFile.isFile())
            throw new IOException("Source is not a file");

        // check if srcFile is readable
        if (!srcFile.canRead())
            throw new IOException("Source is not readable");

        // get the file name
        String fileName = srcFile.getName();

        // get the file's last modification time
        long lastModTime = srcFile.lastModified();

        // get the file's length
        long fileLength = srcFile.length();

        // get the file's MIME type
        String mimeType = srcFile.getMimeType();

        // get the file's content type
        String contentType = srcFile.getContentType();

        // get the file's charset
        String charset = srcFile.getCharset();

        // get the file's encoding
        String encoding = srcFile.getEncoding();

        // get the file's magic number
        long magicNumber = srcFile.getMd5();

        // get the file's CRC32
        long crc32 = srcFile.getCrc32();

        // get the file's SHA1
        byte[] sha1 = srcFile.getSha1();

        // get the file's MD5
        byte[] md5 = srcFile.getMd5();

        // get the file's CRC32
        long crc32 = srcFile.getCrc32();

        // get the file's SHA1
        byte[] sha1 = srcFile.getSha1();

        // get the file's MD5
        byte[] md5 = srcFile.getMd5();

        // ...
    }
}

```



```

/**
 * Downloads a file from a URL to a local file.
 * @author Bruce Eckel
 */
public class FileDownload {
    public static void download(String address, String localFileName)
        throws IOException {
        OutputStream out = null;
        URLConnection conn = null;
        InputStream in = null;
        try {
            URL url = new URL(address);
            out = new BufferedOutputStream(new FileOutputStream(localFileName));
            conn = url.openConnection();
            in = conn.getInputStream();
            byte[] buffer = new byte[1024];
            int numRead;
            long numWritten = 0;
            while ((numRead = in.read(buffer)) != -1) {
                out.write(buffer, 0, numRead);
                numWritten += numRead;
            }
            System.out.println("Local file name = " + localFileName);
            System.out.println("Bytes written = " + numWritten);
        } catch (Exception exception) {
            exception.printStackTrace();
        } finally {
            try {
                in.close();
            } catch (IOException ioe) {}
            try {
                out.close();
            } catch (IOException ioe) {}
        }
    }

    public static void download(String address, String localFileName,
        int lastIndexOfSlash)
        throws IOException {
        int lastIndexOfSlash = address.lastIndexOf('/');
        if (lastIndexOfSlash < 0)
            lastIndexOfSlash = address.length() - 1;
        else
            lastIndexOfSlash = address.lastIndexOf(lastIndexOfSlash);
        System.err.println("Could not figure out local file name for " +
            address);
    }

    public static void main(String[] args) {
        if (args.length != 2)
            System.out.println("Usage: java FileDownload <url> <local file name>");
    }
}

```

```

import java.awt.*;
import java.awt.event.*;
import java.util.*;

/**
 * A Runnable interface.
 * @author Bruce Eckel
 */
public interface Runnable {
    void run();
}

/**
 * A class that implements the Runnable interface.
 * @author Bruce Eckel
 */
public class HappyNewYear implements Runnable {
    private static NumberFormat formatter = NumberFormat.getInstance();
    private JFrame frame;
    private JLabel label;
    private long millis;
    private String message;

    public HappyNewYear(JFrame frame, JLabel label) {
        // store argument GUI elements
        this.frame = frame;
        this.label = label;
        // compute beginning of next year
        Calendar cal = new GregorianCalendar();
        int nextYear = cal.get(Calendar.YEAR) + 1;
        cal.set(Calendar.YEAR, nextYear);
        cal.set(Calendar.DAY_OF_MONTH, 1);
        cal.set(Calendar.HOUR_OF_DAY, 0);
        cal.set(Calendar.MINUTE, 0);
        cal.set(Calendar.SECOND, 0);
        millis = cal.getTime().getTime();
        // prepare a message
        message = "Happy " + nextYear + "!";
    }

    public static int determineFontSize(JFrame frame,
        int componentWidth, String fontName, int fontStyle,
        String text) {
        int fontSize = componentWidth * 2 / text.length();
        Font font = new Font(fontName, fontStyle, fontSize);
        FontMetrics fontMetrics = frame.getFontMetrics(font);
        int stringWidth = fontMetrics.stringWidth(text);
        return (int) (fontSize * 0.95 * componentWidth / stringWidth);
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.addKeyListener(new KeyListener() {
            public void keyPressed(KeyEvent event) {
                public void keyReleased(KeyEvent event) {
                    if (event.getKeyChar() == KeyEvent.VK_ESCAPE)
                        System.exit(0);
                }
            }
            public void keyTyped(KeyEvent event) {}
        });
        frame.setUndecorated(true);
        label.setForeground(Color.WHITE);
        label.setBackground(Color.BLACK);
        label.setOpaque(true);
        frame.getContentPane().add(label);
        final String fontName = "SansSerif";
        int fontSizeNumber = determineFontSize(frame,
            Toolkit.getDefaultToolkit().getScreenSize().width,
            fontName, fontStyle, fontMetrics.format(99999999));
        int fontSize = determineFontSize(frame,
            Toolkit.getDefaultToolkit().getScreenSize().width,
            fontName, fontStyle, "Happy 9999!");
    }
}

```



```

import java.io.*;
import java.util.*;

/**
 * Revised the program to copy a file to another directory.
 * @author Bruce Eckel
 */
public class CopyFile {
    // constant values for the override option
    public static final int OVERWRITE_OPTION = 1;
    public static final int COMMIT_NAME = 2;
    public static final int COMMIT_COPY = 3;

    // program options initialized to default values
    private static int overrideOption = 0;
    private static String commitName = null;
    private static boolean copyWithTimestamp = true;
    private static boolean commitCopy = true;
    private static int commitOption = OVERWRITE_OPTION;

    public static long copyFile(String srcFile, File destDir)
        throws IOException {
        // check if destDir is a directory
        if (!destDir.isDirectory())
            throw new IOException("Destination is not a directory");
        // check if srcFile is a file
        if (!srcFile.isFile())
            throw new IOException("Source is not a file");

        // check if srcFile is readable
        if (!srcFile.canRead())
            throw new IOException("Source is not readable");

        // get the file name
        String fileName = srcFile.getName();

        // get the file's last modification time
        long lastModTime = srcFile.lastModified();

        // get the file's length
        long fileLength = srcFile.length();

        // get the file's MIME type
        String mimeType = srcFile.getMimeType();

        // get the file's content type
        String contentType = srcFile.getContentType();

        // get the file's charset
        String charset = srcFile.getCharset();

        // get the file's encoding
        String encoding = srcFile.getEncoding();

        // get the file's magic number
        long magicNumber = srcFile.getMd5();

        // get the file's CRC32
        long crc32 = srcFile.getCrc32();

        // get the file's SHA1
        byte[] sha1 = srcFile.getSha1();

        // get the file's MD5
        byte[] md5 = srcFile.getMd5();

        // ...
    }
}

```

Calls to database
- after state change
- follow DB protocol

???

Factory methods
- create objects consistently

???

Coding conventions
- e.g. all events prefixed Event*
- follow for consistency

???

Software Erosion



```
package org.ej44.de.jmaki;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

/**
 * Instead of subclassing JRadioButton directly, you should now write an
 * action and file.
 *
 * @author Steve Nouri
 * @version 1.0 (2002/02/28 02:17:13) operator step 3
 */
public abstract class JRadioButton {

    /**
     * Creates a new JRadioButton with the specified name.
     * @param name the action name
     */
    public JRadioButton(String name) {
        this.name = name;
    }

    /**
     * Returns the internal name of this action.
     */
    public String getName() {
        return name;
    }

    /**
     * Returns the label of this button. The default implementation returns the
     * value of the property named by the action's internal name suffixed
     * with ".label".
     */
    public String getLabel() {
        return JRadioButton.getAction(name + ".label");
    }

    /**
     * Returns the text that should be shown when the mouse is placed over
     * this button's icon. Use of this method is currently only used by
     * the Swing system.
     * @param name the action name
     */
    public String getToolTipText() {
        return name;
    }

    /**
     * Returns the icon of this button. The default implementation returns the
     * value of the property named by the action's internal name suffixed
     * with ".icon".
     */
    public Image getIcon() {
        return JRadioButton.getAction(name + ".icon");
    }

    /**
     * Returns the text that should be shown when the mouse is placed over
     * this button's icon. Use of this method is currently only used by
     * the Swing system.
     * @param name the action name
     */
    public String getToolTipText() {
        return name;
    }

    /**
     * Returns the icon of this button. The default implementation returns the
     * value of the property named by the action's internal name suffixed
     * with ".icon".
     */
    public Image getIcon() {
        return JRadioButton.getAction(name + ".icon");
    }

    /**
     * Returns the text that should be shown when the mouse is placed over
     * this button's icon. Use of this method is currently only used by
     * the Swing system.
     * @param name the action name
     */
    public String getToolTipText() {
        return name;
    }

    /**
     * Returns the icon of this button. The default implementation returns the
     * value of the property named by the action's internal name suffixed
     * with ".icon".
     */
    public Image getIcon() {
        return JRadioButton.getAction(name + ".icon");
    }
}

//end, returns false
//by default,
//return JRadioButton.getAction(name + ".label");
```



```
package org.ej44.de.jmaki;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

/**
 * Instead of subclassing JRadioButton directly, you should now write an
 * action and file.
 *
 * @author Steve Nouri
 * @version 1.0 (2002/02/28 02:17:13) operator step 3
 */
public abstract class JRadioButton {

    /**
     * Creates a new JRadioButton with the specified name.
     * @param name the action name
     */
    public JRadioButton(String name) {
        this.name = name;
    }

    /**
     * Returns the internal name of this action.
     */
    public String getName() {
        return name;
    }

    /**
     * Returns the label of this button. The default implementation returns the
     * value of the property named by the action's internal name suffixed
     * with ".label".
     */
    public String getLabel() {
        return JRadioButton.getAction(name + ".label");
    }

    /**
     * Returns the text that should be shown when the mouse is placed over
     * this button's icon. Use of this method is currently only used by
     * the Swing system.
     * @param name the action name
     */
    public String getToolTipText() {
        return name;
    }

    /**
     * Returns the icon of this button. The default implementation returns the
     * value of the property named by the action's internal name suffixed
     * with ".icon".
     */
    public Image getIcon() {
        return JRadioButton.getAction(name + ".icon");
    }
}

//end, returns false
//by default,
//return JRadioButton.getAction(name + ".label");
```

```
package org.ej44.de.jmaki;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

/**
 * Instead of subclassing JRadioButton directly, you should now write an
 * action and file.
 *
 * @author Steve Nouri
 * @version 1.0 (2002/02/28 02:17:13) operator step 3
 */
public abstract class JRadioButton {

    /**
     * Creates a new JRadioButton with the specified name.
     * @param name the action name
     */
    public JRadioButton(String name) {
        this.name = name;
    }

    /**
     * Returns the internal name of this action.
     */
    public String getName() {
        return name;
    }

    /**
     * Returns the label of this button. The default implementation returns the
     * value of the property named by the action's internal name suffixed
     * with ".label".
     */
    public String getLabel() {
        return JRadioButton.getAction(name + ".label");
    }

    /**
     * Returns the text that should be shown when the mouse is placed over
     * this button's icon. Use of this method is currently only used by
     * the Swing system.
     * @param name the action name
     */
    public String getToolTipText() {
        return name;
    }

    /**
     * Returns the icon of this button. The default implementation returns the
     * value of the property named by the action's internal name suffixed
     * with ".icon".
     */
    public Image getIcon() {
        return JRadioButton.getAction(name + ".icon");
    }
}

//end, returns false
//by default,
//return JRadioButton.getAction(name + ".label");
```



time

Small vs. large software

```
import java.io.*;
import java.util.*;

/** Command line program to copy a file to another directory.
 * @author Bruce Eckel
 */
public class CopyFile {
    // constants values for the override option
    public static final int OVERRIDE_OPTION = 1;
    public static final int NO_OVERRIDE_OPTION = 0;

    // program options defined in default values
    private static int overrideOption = NO_OVERRIDE_OPTION;
    private static boolean copyFileToAnotherDir = true;
    private static boolean copyFileToCurrentDir = false;
    private static int overrideOption = NO_OVERRIDE_OPTION;

    public static void copyFile(String srcFile, String destDir)
        throws IOException {
        // check if srcFile is a directory
        if (new File(srcFile).isDirectory())
            return;

        // check if destDir is a directory
        if (!new File(destDir).isDirectory())
            return;

        // get the file name
        String fileName = new File(srcFile).getName();

        // get the destination file name
        String destFileName = destDir + fileName;

        // check if the destination file already exists
        if (new File(destDir + fileName).exists())
            return;

        // copy the file
        try {
            FileInputStream in = new FileInputStream(srcFile);
            FileOutputStream out = new FileOutputStream(destDir + fileName);
            byte[] buffer = new byte[1024];
            int numRead;
            while ((numRead = in.read(buffer)) != -1)
                out.write(buffer, 0, numRead);
            in.close();
            out.close();
        } catch (IOException e) {
            System.err.println("Error copying file: " + e.getMessage());
        }
    }

    // main method
    public static void main(String[] args) {
        if (args.length != 2)
            System.out.println("Usage: java CopyFile <srcFile> <destDir>");
        else {
            copyFile(args[0], args[1]);
        }
    }
}
```

```
public class FileDownload {
    public static void download(String address, String localFileName)
        throws IOException {
        URL url = new URL(address);
        HttpURLConnection conn = null;
        InputStream in = null;
        OutputStream out = null;

        try {
            conn = (HttpURLConnection) url.openConnection();
            in = conn.getInputStream();
            out = new FileOutputStream(localFileName);
            byte[] buffer = new byte[1024];
            int numRead;
            long numWritten = 0;
            while ((numRead = in.read(buffer)) != -1)
                out.write(buffer, 0, numRead);
            numWritten += numRead;
        } catch (Exception e) {
            System.out.println("Error downloading file: " + e.getMessage());
        } finally {
            if (in != null)
                in.close();
            if (out != null)
                out.close();
        }
    }

    public static void main(String[] args) {
        if (args.length != 2)
            System.out.println("Usage: java FileDownload <address> <localFileName>");
        else {
            download(args[0], args[1]);
        }
    }
}
```

```
public class HappyNewYear implements Runnable {
    // constants
    private static final int YEAR = 2000;
    private static final int MONTH = 1;
    private static final int DAY = 1;
    private static final int HOUR = 0;
    private static final int MINUTE = 0;
    private static final int SECOND = 0;

    // instance variables
    private JFrame frame;
    private JLabel label;
    private long newYearMillis;
    private String message;

    public HappyNewYear(JFrame frame, JLabel label) {
        this.frame = frame;
        this.label = label;
        // compute beginning of next year
        Calendar cal = new GregorianCalendar();
        int nextYear = cal.get(Calendar.YEAR) + 1;
        cal.set(Calendar.YEAR, nextYear);
        cal.set(Calendar.MONTH, Calendar.JANUARY);
        cal.set(Calendar.DAY_OF_MONTH, 1);
        cal.set(Calendar.HOUR_OF_DAY, 0);
        cal.set(Calendar.MINUTE, 0);
        cal.set(Calendar.SECOND, 0);
        newYearMillis = cal.getTime().getTime();
        // prepare a message
        message = "Happy " + nextYear + "!";
    }

    public static int determineFontSize(JFrame frame,
        int componentWidth, String fontName, int fontStyle,
        String text) {
        int fontSize = componentWidth * 2 / text.length();
        Font font = new Font(fontName, fontStyle, fontSize);
        FontMetrics fontMetrics = frame.getFontMetrics(font);
        int stringWidth = fontMetrics.stringWidth(text);
        return (int) (fontSize * (1 - (stringWidth / componentWidth)));
    }

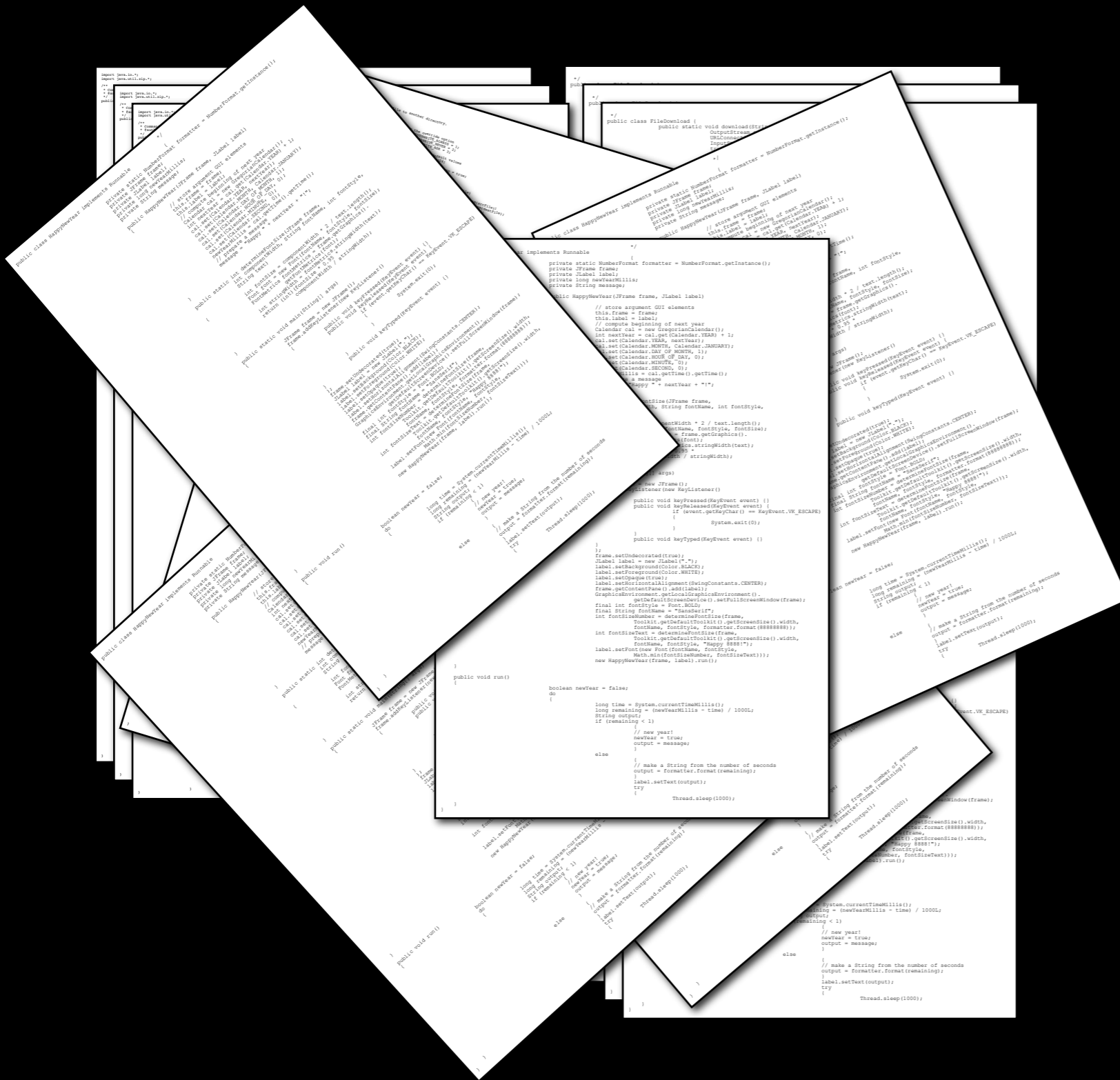
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.addKeyListener(new KeyListener() {
            public void keyPressed(KeyEvent event) {
                public void keyReleased(KeyEvent event) {
                    if (event.getKeyChar() == KeyEvent.VK_ESCAPE)
                        System.exit(0);
                }
            }
            public void keyTyped(KeyEvent event) {
            }
        });
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        label = new JLabel(" ");
        label.setBackground(Color.BLACK);
        label.setForeground(Color.WHITE);
        label.setOpaque(true);
        label.setHorizontalAlignment(Constants.CENTER);
        frame.getContentPane().add(label);
        GraphicsEnvironment ge = GraphicsEnvironment.getLocalGraphicsEnvironment();
        GraphicsDevice gd = ge.getDefaultScreenDevice();
        gd.setFullScreenWindow(frame);
        final String fontName = "SansSerif";
        final int fontStyle = Font.BOLD;
        int fontSize = determineFontSize(frame,
            Toolkit.getDefaultToolkit().getScreenSize().width,
            fontName, fontStyle, Constants.FORMAT(88888888));
        int fontSizeText = determineFontSize(frame,
            Toolkit.getDefaultToolkit().getScreenSize().width,
            fontName, fontStyle, "Happy 8888!");
        label.setFont(new Font(fontName, fontStyle,
            Math.min(fontSize, fontSizeText)));
        new HappyNewYear(frame, label).run();
    }

    public void run() {
        boolean newYear = false;
        do {
            long time = System.currentTimeMillis();
            long remaining = (newYearMillis - time) / 1000L;
            if (remaining < 1)
                // new year!
                newYear = true;
                output = message;
            else
                // make a String from the number of seconds
                output = formatter.format(remaining);
            label.setText(output);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
            }
        } while (true);
    }
}
```

Manual
does not
scale

Tools
needed

Small vs. large software



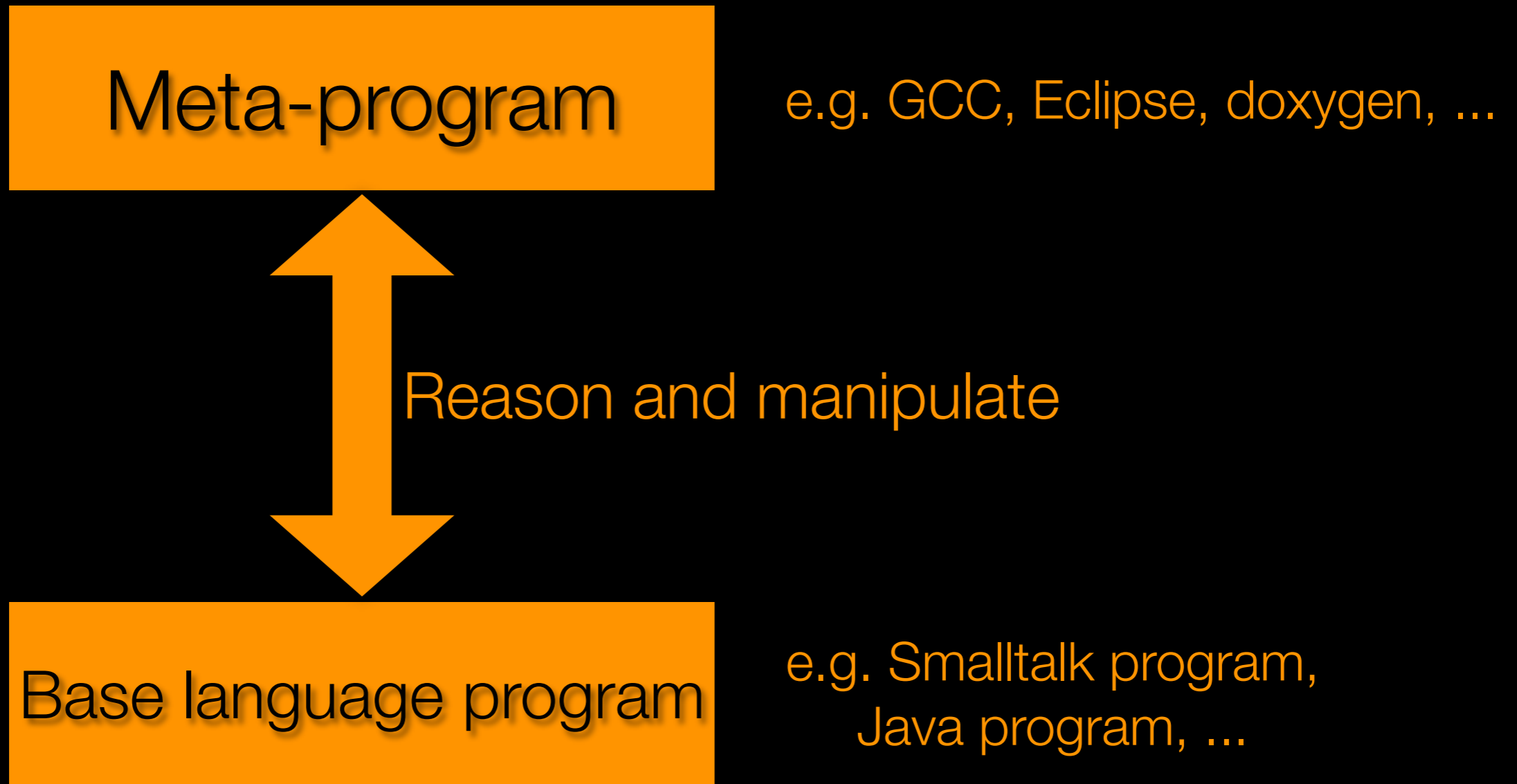
Manual
does not
scale

Tools
needed

Declarative meta programming

- ▶ **Prevent software erosion**
- ▶ **Querying the source code**
 - Structure of the source code
 - Static analysis (approximate what the source code will do)
 - mix of logic programming and imperative/OO programming
- ▶ **Causal connection**
 - Keep the link with the source code
 - If code changes, queries work over new code

Meta programming



Example

```
Transcript show: 'Hello, World!'
```

In

Compiler

Out

```
short CompiledBlock numArgs=0 numTemps=0 frameSize=12
```

```
literals: ({Transcript} 'Hello, World!' )
```

```
1 <34> push Transcript
```

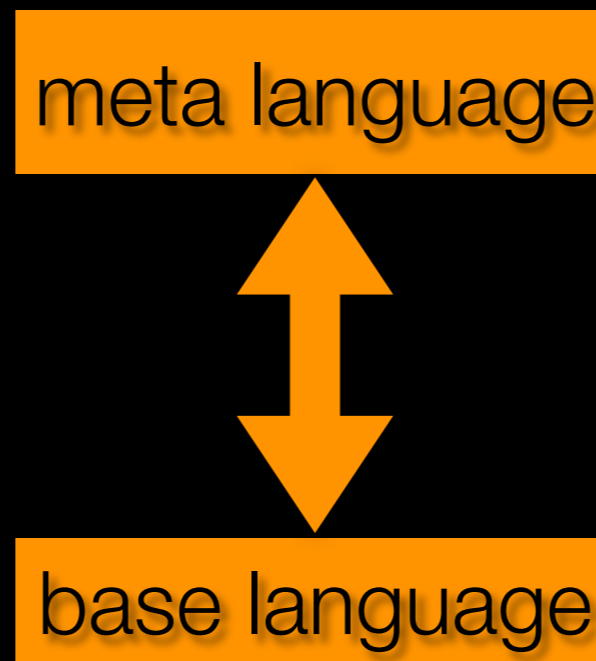
```
2 <1D> push 'Hello, World!'
```

```
3 <F0 80> send show:
```

```
5 <65> return
```

Meta programming language

- ▶ Programming language used to write meta programs
- ▶ Reason over/manipulate programs in *base* language



- ▶ Meta and base language can be the same

Example meta program in Smalltalk

```
Smalltalk allClasses
  do:[:class |
    class selectors
      do:[:selector |
        Transcript show: class asString, '>>', selector asString; cr.
      ]
  ]
```

```
SmalltalkMethod>>logicParseTree
SmalltalkMethod>>classesReferenced
SmalltalkMethod>>printOn:
F03_Streams_03>>codeExample
F03_Streams_03>>how
F03_Streams_03>>what
F03_Streams_03>>sortArray
F03_Streams_03>>why
F03_Streams_03>>defaultWorkspaceText
F03_Streams_03>>displayString
.....
```

meta: Smalltalk



base: Smalltalk

Declarative meta programming

- ▶ Use of declarative language as meta language
- ▶ In our case: Prolog

declarative meta language



base language

Why Prolog?

- ▶ **As mentioned yesterday:**
 - Focus on *what*, rather than *how*
- ▶ **Source code is the fact base**
- ▶ **Use queries in order to retrieve information from that code**
- ▶ **Has proven to be very effective!**

SOUL

- ▶ **Smalltalk Open *Unification Language***
- ▶ **Implemented on top of Smalltalk**
- ▶ **Prolog derivative**
- ▶ **Symbiosis with Smalltalk**
- ▶ **Libraries of predicates to reason about software**



SOUL vs. Prolog

- ▶ **Variables indicated by ?**
- ▶ **Lists using < and > instead of [and]**
- ▶ ***equals* predicate instead of =**
- ▶ **Keyword if instead of :-**

```
%append(?list1, ?list2, ?list)
```

```
append(<>, ?list, ?list)
```

```
append(<?first | ?rest>, ?list2, <?first | ?list>) if  
    append(?rest, ?list2, ?list).
```

Smalltalk terms

- ▶ **Symbiosis with underlying Smalltalk**
- ▶ **Use Smalltalk objects in SOUL**
- ▶ **Indicated using [and]**
- ▶ **Use of Smalltalk values:**

```
equals(?string, ['Hello, World'])
```

- ▶ **Use of Smalltalk expressions:**

```
add(?number1, ?number2, [?number1 + ?number2])
```

Smalltalk clauses

- ▶ Use a Smalltalk block as a logic condition
- ▶ Must evaluate to *true* or *false*
- ▶ Variables must be bound when evaluated
- ▶ Unification = evaluation

```
even(?number) if  
  [(?number rem: 2) = 0]
```

Resolution of Smalltalk clauses

if even(10)

if even(10)

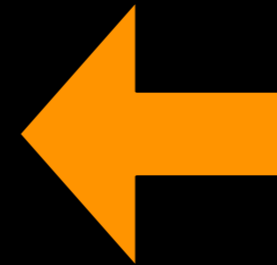


SOUL

Resolution of Smalltalk clauses

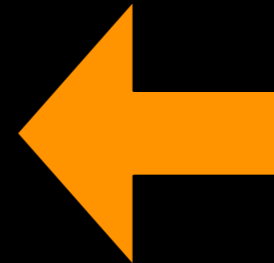
if even(10)

if even(10)



SOUL

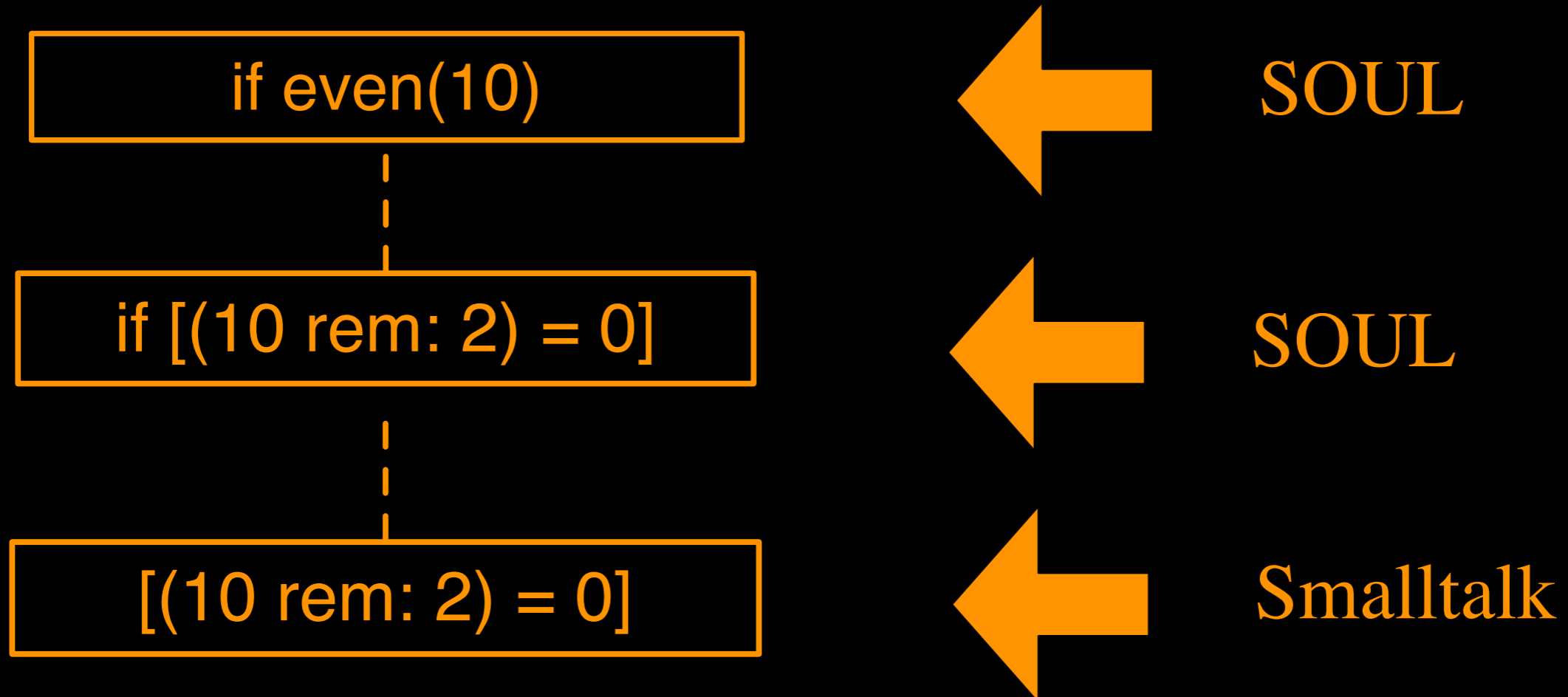
if [(10 rem: 2) = 0]



SOUL

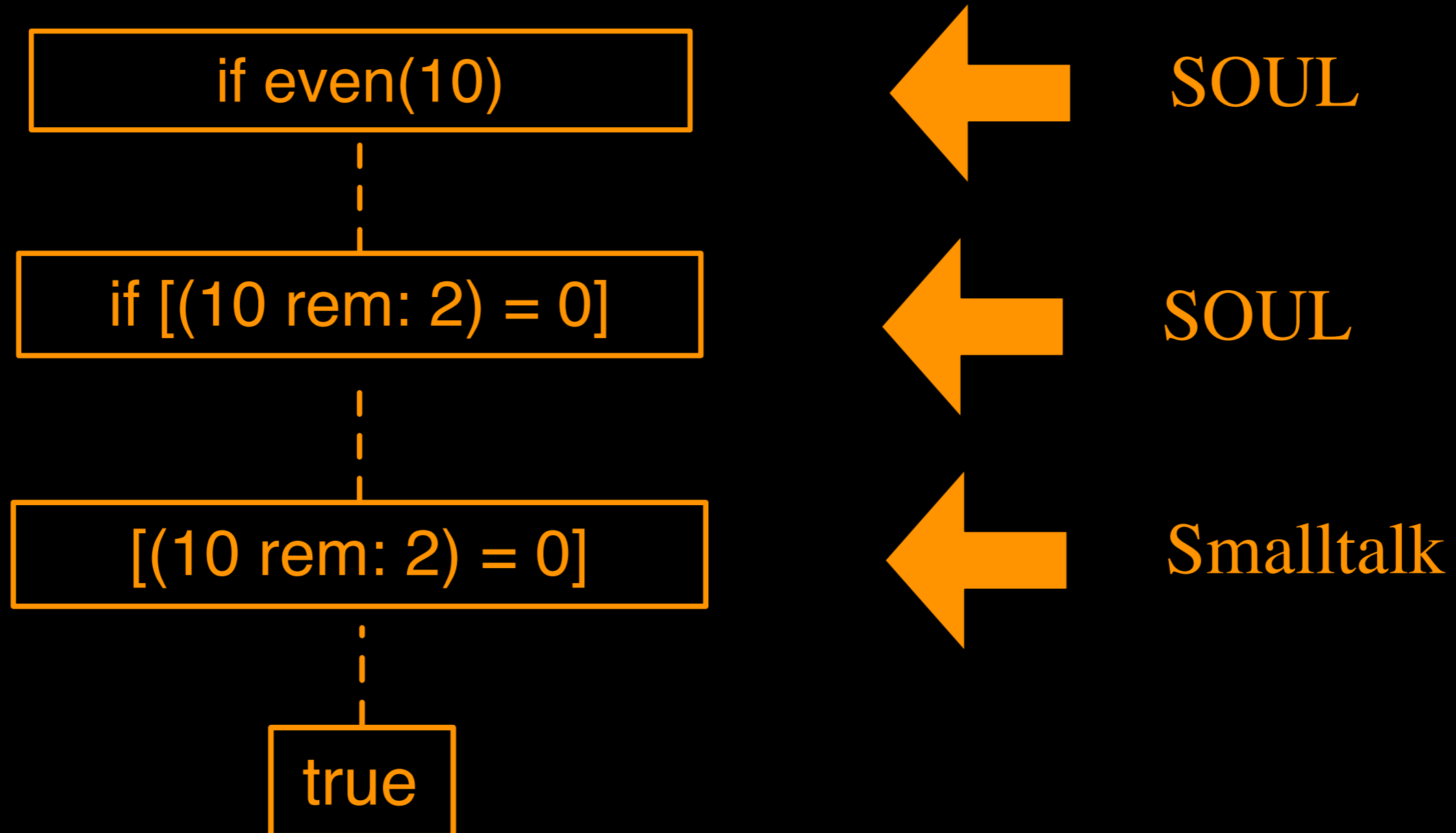
Resolution of Smalltalk clauses

`if even(10)`



Resolution of Smalltalk clauses

`if even(10)`



Example usage of symbiosis

```
if member(?number, <1,2,3>)
```

```
if member(?number, [Array with: 1 with: 2 with: 3])
```

```
?number -> 1
```

```
?number -> 2
```

```
?number -> 3
```

Smalltalk collection

```
member(?element, ?list) if  
  [?list isKindOf: Collection],  
  memberCollection(?element, ?list)
```

SOUL list

```
member(?element, ?list) if  
  list(?list),  
  memberList(?element, ?list)
```

Quoted code

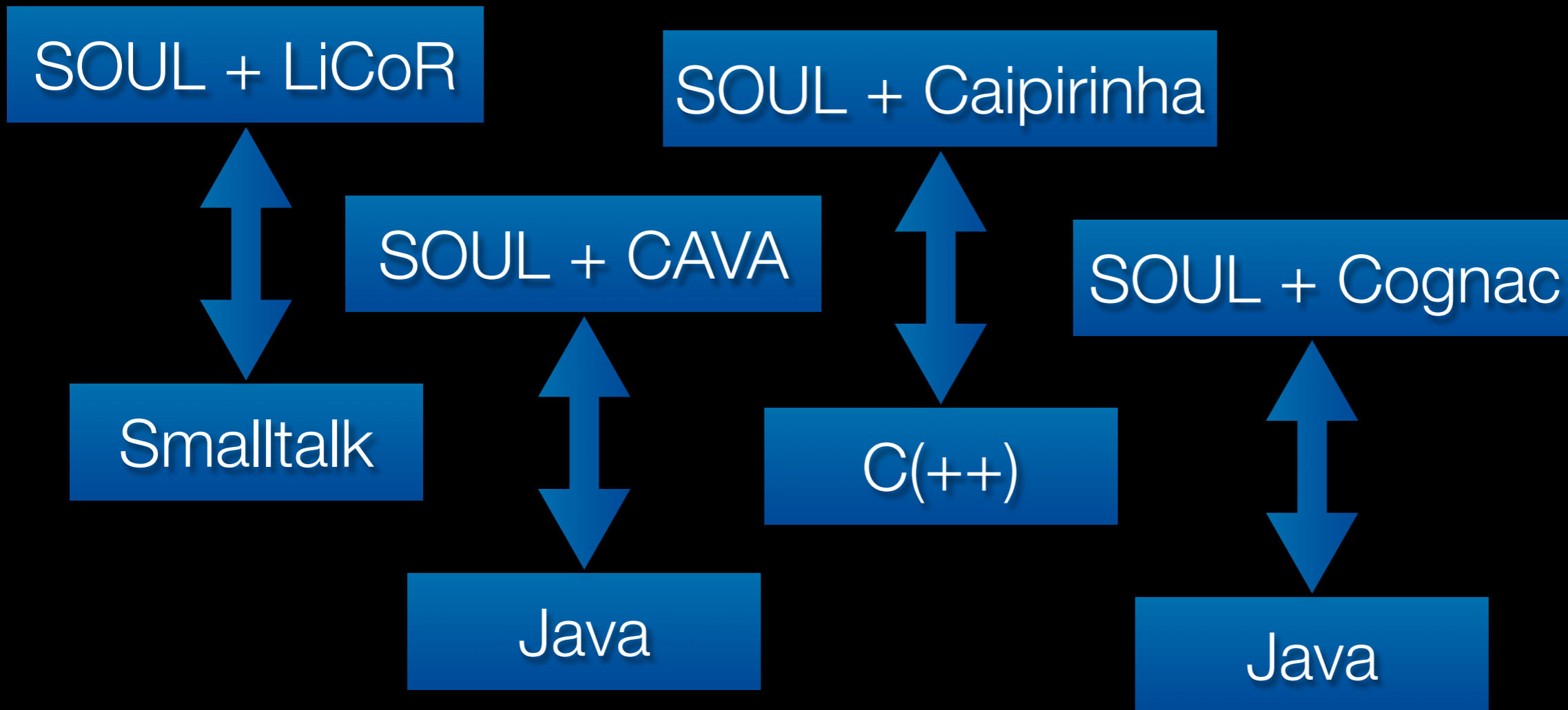
- ▶ Consider it a string with variables
- ▶ Useful for code generation
- ▶ Not evaluated by SOUL
- ▶ Indicated by { and }

```
equals(?code, {Array with: ?x with: ?y}),  
equals(?x, 1),  
equals(?y, 2),  
equals(?array, [Compiler evaluate: ?code])
```

```
?code -> {Array with: 1 with: 2}  
?array -> #(1 2)
```

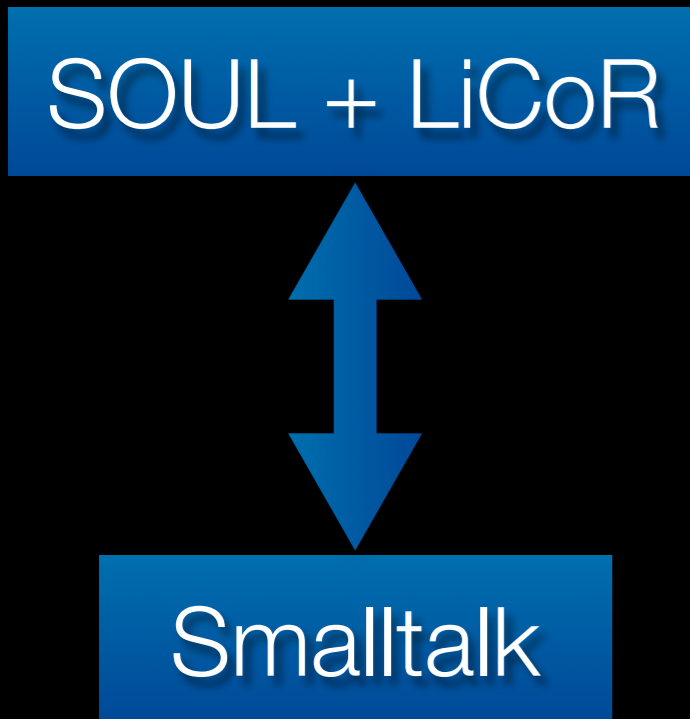
Reasoning about actual programs

- ▶ Libraries of predicates for SOUL
- ▶ For Smalltalk, Java, C(++) and Cobol

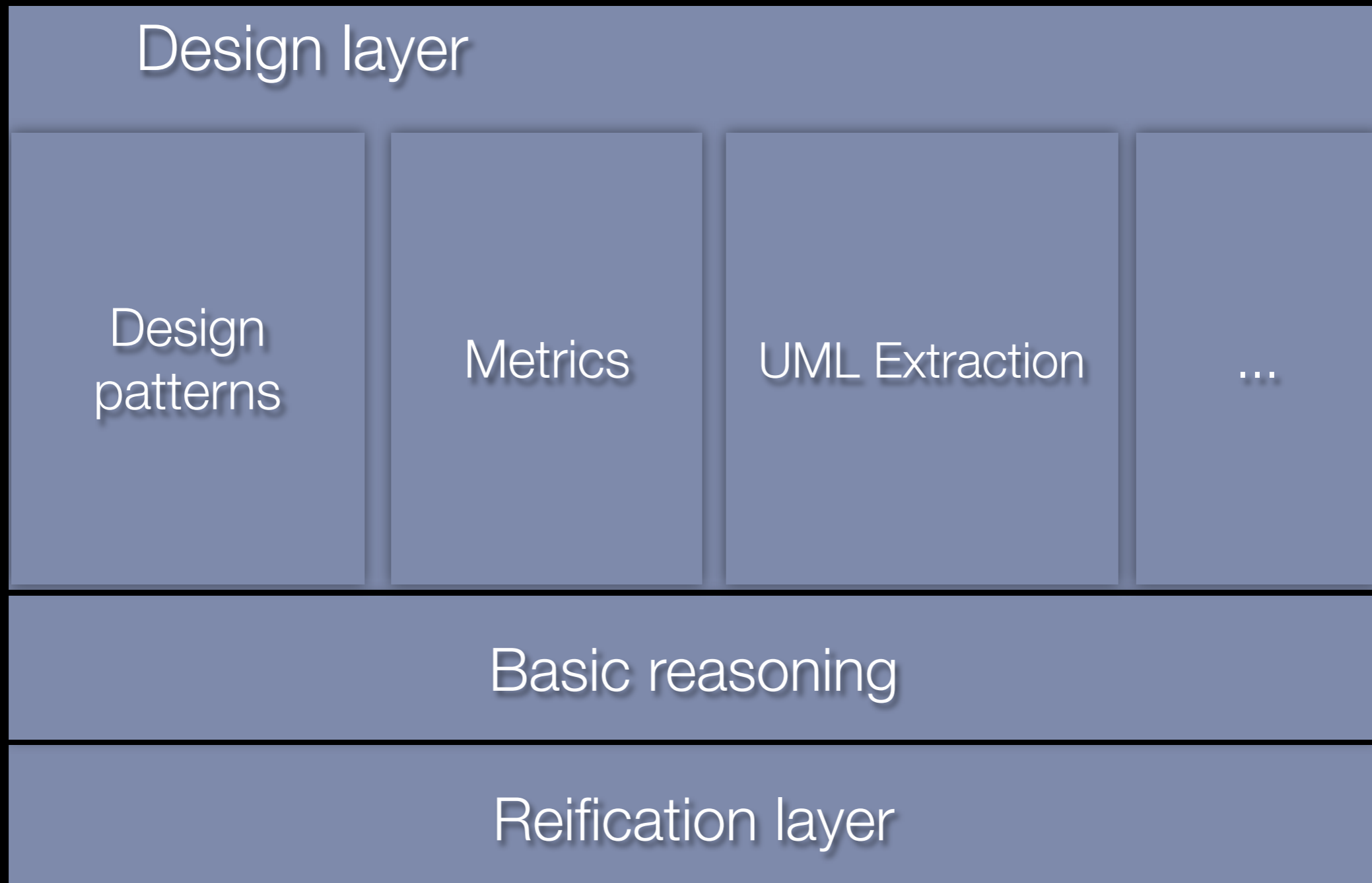


Reasoning about actual programs

- ▶ **Libraries of predicates for SOUL**
- ▶ **For Smalltalk, Java, C(++) and Cobol**



Layered design



Reification layer

- ▶ **Represent program entities in SOUL**
 - E.g. classes, methods, bundles, packages, ...
- ▶ **Basic set of predicates**

`class(?class)`

`method(?method)`

`methodInClass(?method)`

`classInPackage(?class, ?package)`

Reification layer

Example

The screenshot shows the SOUL Querybrowser application window. The title bar reads "SOUL Querybrowser". The main query input field contains the text "if class(?x)". Below the input field, there are controls for "Lookup in:" (set to "default") and "Evaluator" (set to "Evaluator"), along with a "Configure" button. To the right of the input field are several buttons: "All Results", "All Results Int.", "Next Result", and "Basic Inspect". Below these buttons is a "Variable View Ordering" section with a list containing "?x" and buttons for "Apply" and "Clear". At the bottom of the window, there are three view mode buttons: "Browser View", "Tree View", and "Text View", with "Text View" currently selected. The main display area shows the following output:

```
SOUL found
7734 solutions in 103 ms for:
if class(?x)

[?x-->[Weaklings.WeakProxy]]

[?x-->[Object]]

[?x-->[Layout]]

[?x-->[LayoutOrigin]]

[?x-->[LayoutFrame]]

[?x-->[MovingLayoutFrame]]
```


Implemented using symbiosis

- ▶ Extensive meta programming facilities in Smalltalk
- ▶ Use them to implement reification layer

```
class(?class) if  
    member(?class, [Smalltalk allClasses])
```

```
methodInClass(?method,?class) if  
    member(?method, [?class selectors collect:[:sel |  
        ?class compiledMethodAt: sel]])
```

Implemented using symbiosis

- ▶ Extensive meta programming facilities in Smalltalk
- ▶ Use them to implement reification layer

```
class(?class) if  
  member(?class, [Smalltalk allClasses])
```

Link to actual
implementation

```
methodInClass(?method, ?class) if  
  member(?method, [?class selectors collect:[:sel |  
    ?class compiledMethodAt: sel]])
```

Via Smalltalk

Basic reasoning

- ▶ **Built on top of reification layer**

- ▶ **Extract:**

- basic relationships
- abstract class predicate
- message sends

```
classInHierarchyOf(?class, ?parent)
methodNameInClass(?method, ?name, ?
class)
classImplements(?class, ?selector)
abstractMethod(?method)
methodSendsSelector(?method, ?selector)
....
```

Example

The screenshot shows the SOUL Querybrowser application window. The title bar reads "SOUL Querybrowser". The main query area contains the text: `if classInHierarchyOf(?class,[ApplicationModel])`. Below the query area, there are controls for "Lookup in:" (set to "default") and "Evaluator" (set to "Evaluator"), with a "Configure" button. To the right of the query area are buttons for "All Results", "All Results Int.", "Next Result", and "Basic Inspect". Below these is a "Variable View Ordering" section with a list containing "?class" and buttons for "Apply" and "Clear". At the bottom of the window, there are tabs for "Browser View", "Tree View", and "Text View", with "Text View" selected. The main display area shows the results of the query: "SOUL found 405 solutions in 9 ms for: if classInHierarchyOf(?class,[ApplicationModel])". The results are listed as follows:

- [?class-->[ApplicationModel]]
- [?class-->[SimpleDialog]]
- [?class-->[SimpleListEditor]]
- [?class-->[Differator]]
- [?class-->[Inspector]]
- [?class-->[UISettings]]

A more complex example

```
if methodInClass(?method, ?class),  
    methodWithReturnStatement(?method, variable(?field)),  
    instanceVariableInClass(?field, ?class)
```

```
Person>>name
```

```
^name
```

```
Person>>address
```

```
^address
```



Find
accessors

Accessors

The screenshot shows the SOUL Querybrowser interface. At the top, the window title is "SOUL Querybrowser". The main query area contains the following code:

```
if methodInClass(?method,?class),
methodWithReturnStatement(?method,variable(?field)),
instanceVariableInClass(?field,?class)
```

Below the query area, there are controls for the lookup process:

- Lookup in: 16 solutions in 6552 ms
- Evaluator:

On the right side, there are several buttons for navigation and inspection:

- All Results
- Next Result
- Next x Results
- Debug
- Basic Inspect

Below these buttons is a "Variable View Ordering" section with a list box containing:

- ?method
- ?field
- ?class

Next to the list box are "Apply" and "Clear" buttons.

At the bottom, there are three view mode buttons: "Browser View", "Tree View", and "Text View". The "Text View" button is currently selected.

The main results area displays the following output:

```
[?class-->[LayoutOrigin],
?field-->[#topFraction],
?method-->[LayoutOrigin>>topFraction]]

[?class-->[LayoutOrigin],
?field-->[#left],
?method-->[LayoutOrigin>>leftOffset]]

[?class-->[LayoutOrigin],
?field-->[#leftFraction],
?method-->[LayoutOrigin>>leftFraction]]

[?class-->[LayoutFrame],
?field-->[#right],
?method-->[LayoutFrame>>rightOffset]]
```

Design layer

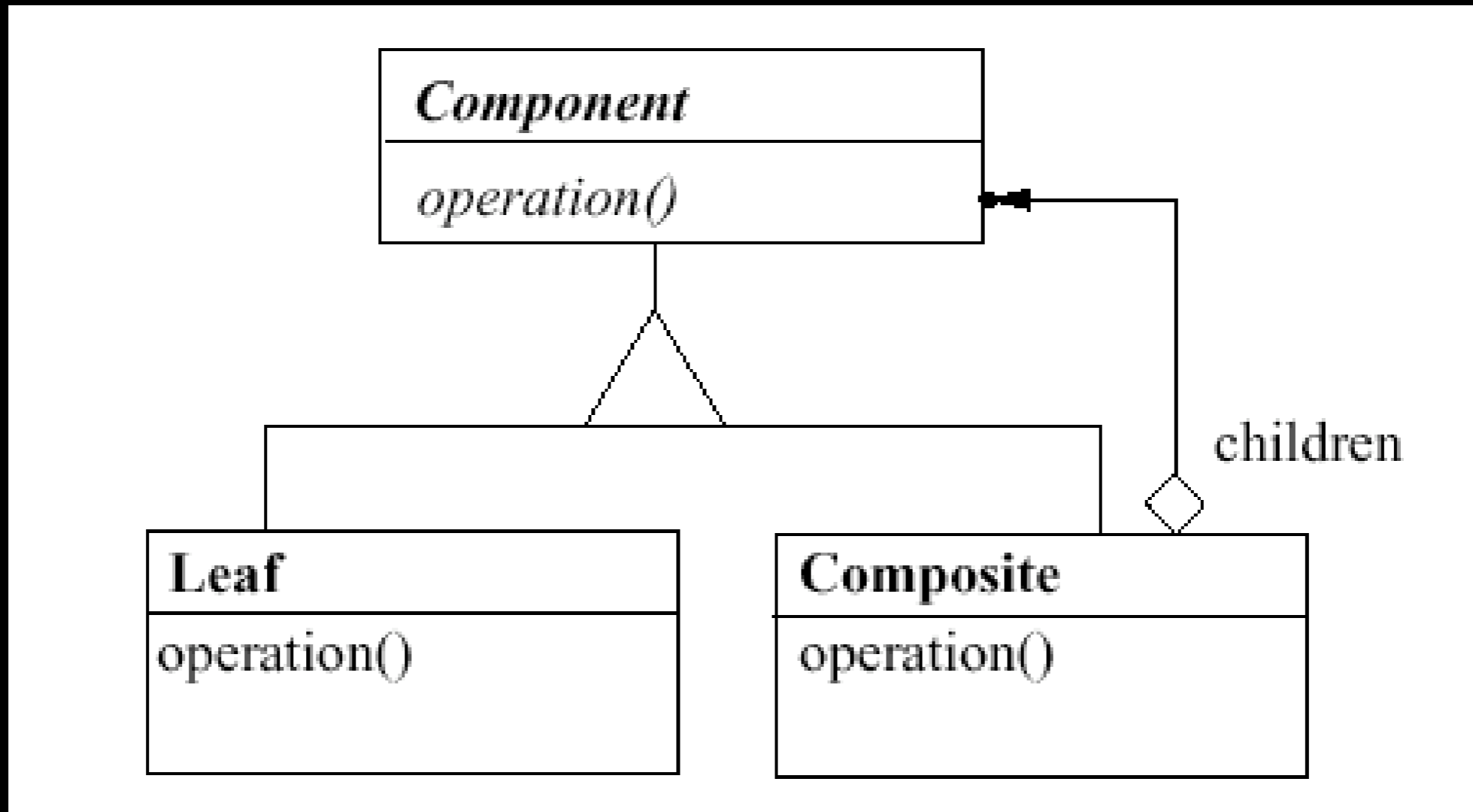
▶ Set of predicates to:

- detect design patterns
- detect bad smells
- calculate metrics
- ...

▶ Using the predicates from the underlying layers

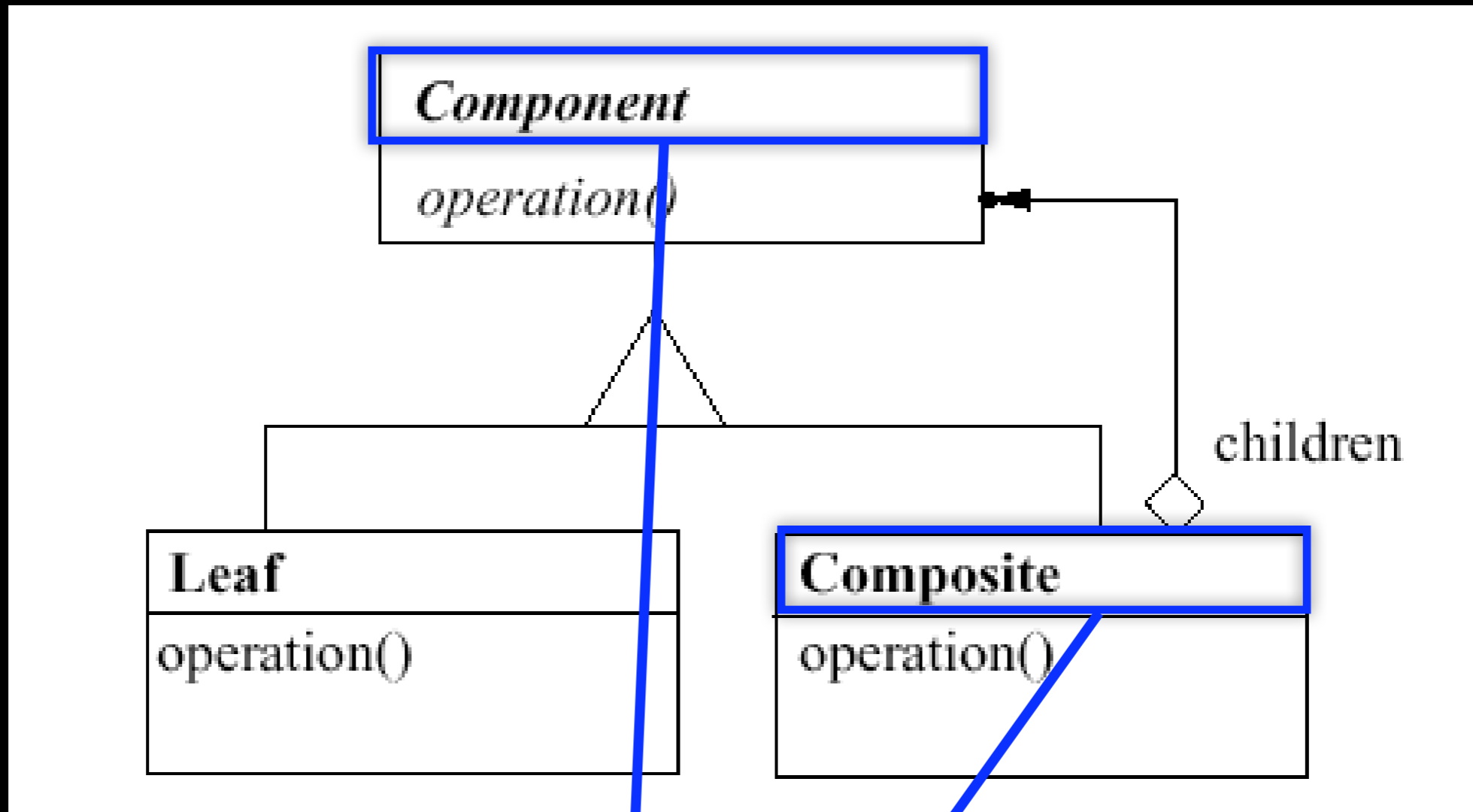


Example: composite design pattern



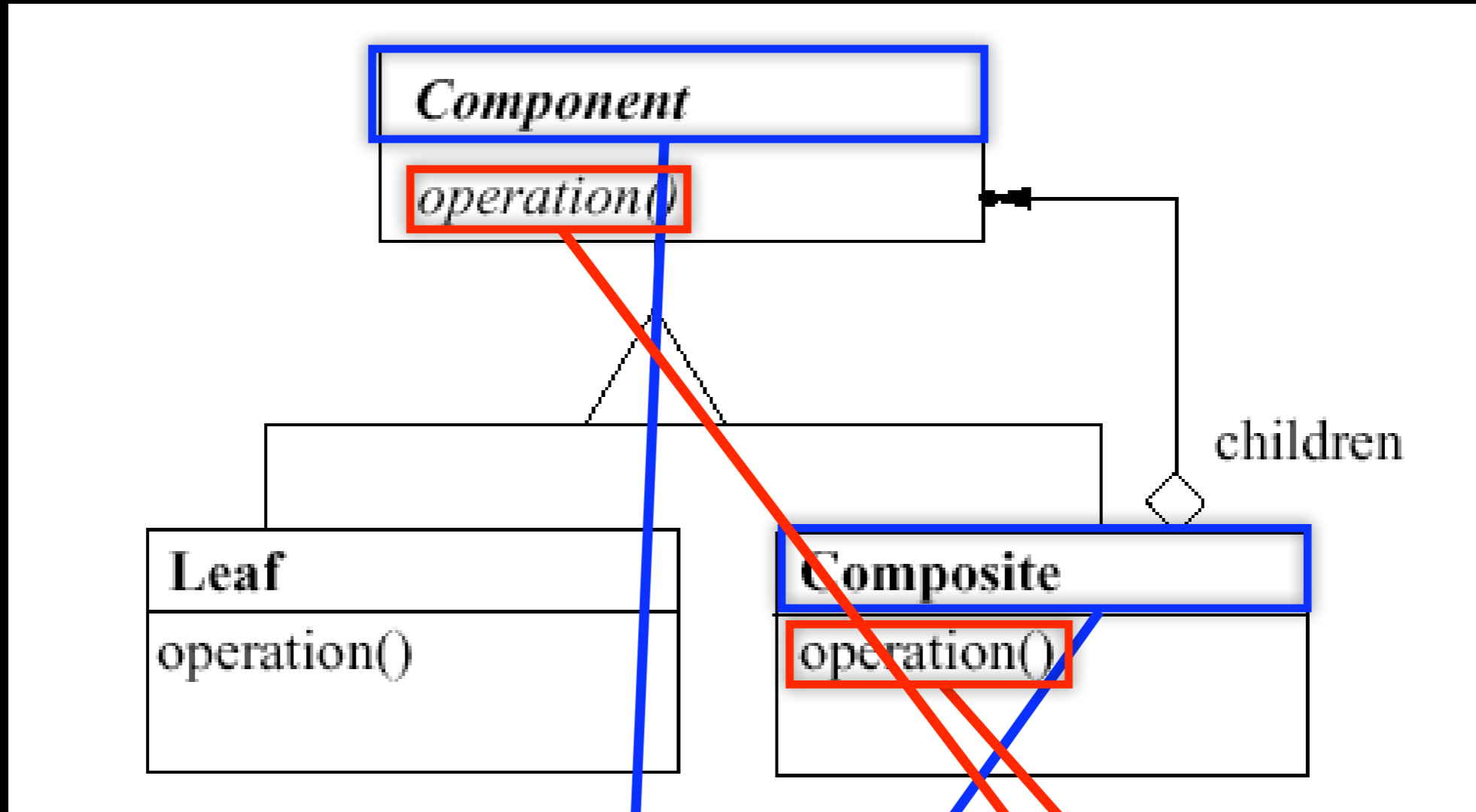
```
compositePattern(?comp, ?composite, ?msg) if  
  compositeStructure(?comp, ?composite),  
  compositeAggregation(?comp, ?composite, ?msg)
```


Example: composite design pattern



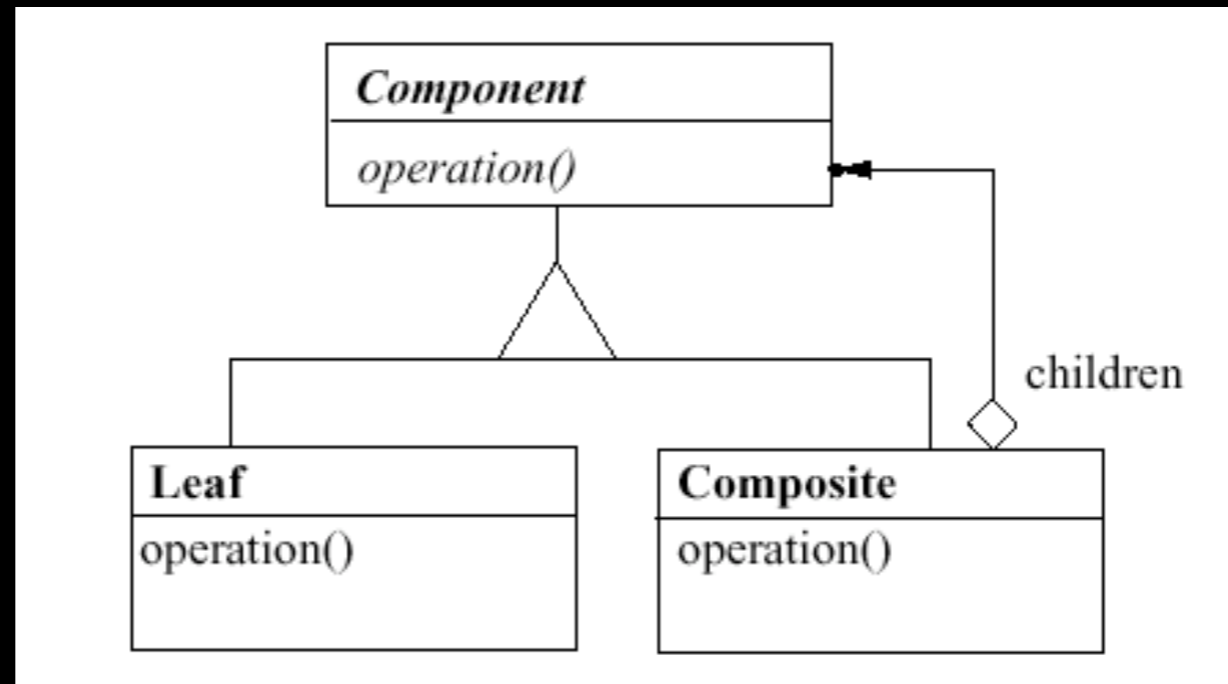
```
compositePattern(?comp, ?composite, ?msg) if
  compositeStructure(?comp, ?composite),
  compositeAggregation(?comp, ?composite, ?msg)
```

Example: composite design pattern



```
compositePattern(?comp, ?composite, ?msg) if  
  compositeStructure(?comp, ?composite),  
  compositeAggregation(?comp, ?composite, ?msg)
```

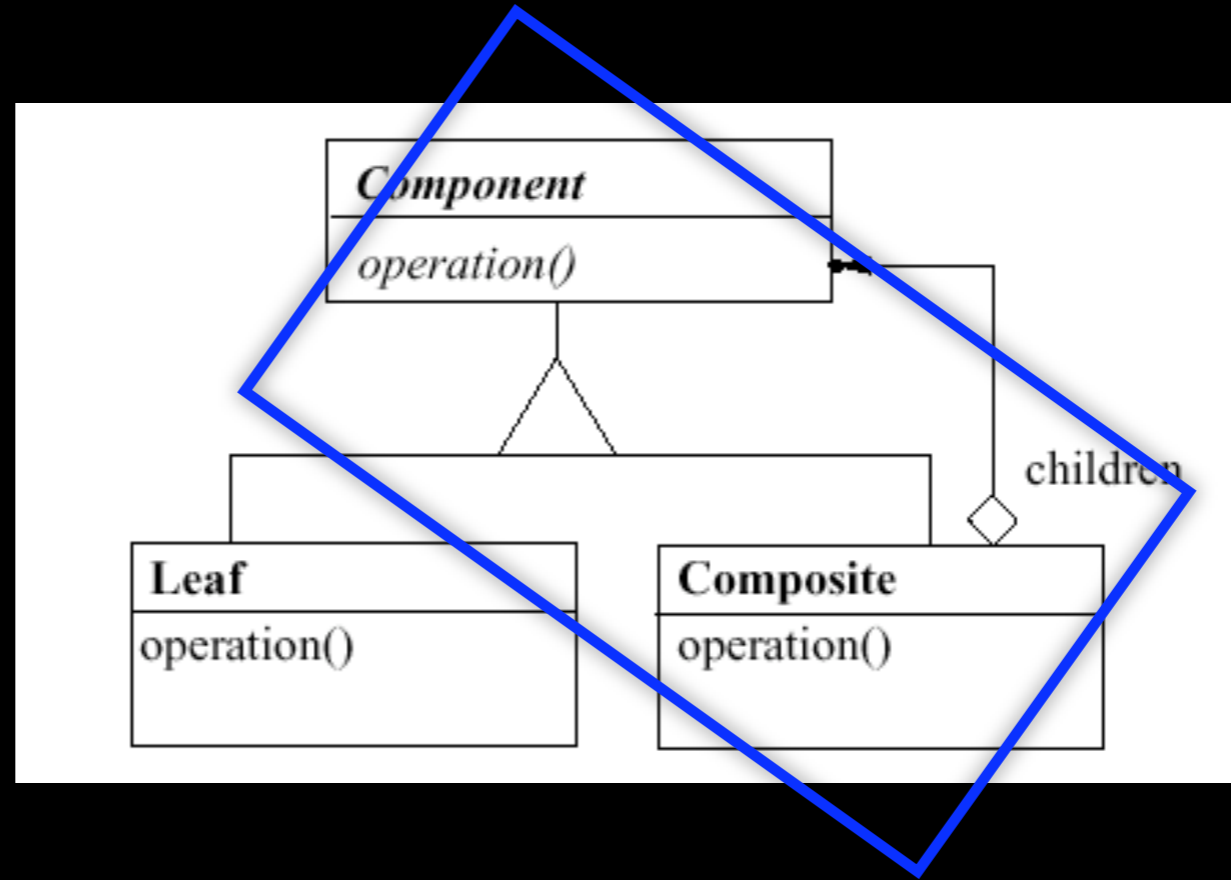
Example continued



Find
structure

```
compositeStructure(?comp, ?composite) if
class(?comp),
classInHierarchyOf(?composite, ?comp)
```

Example continued



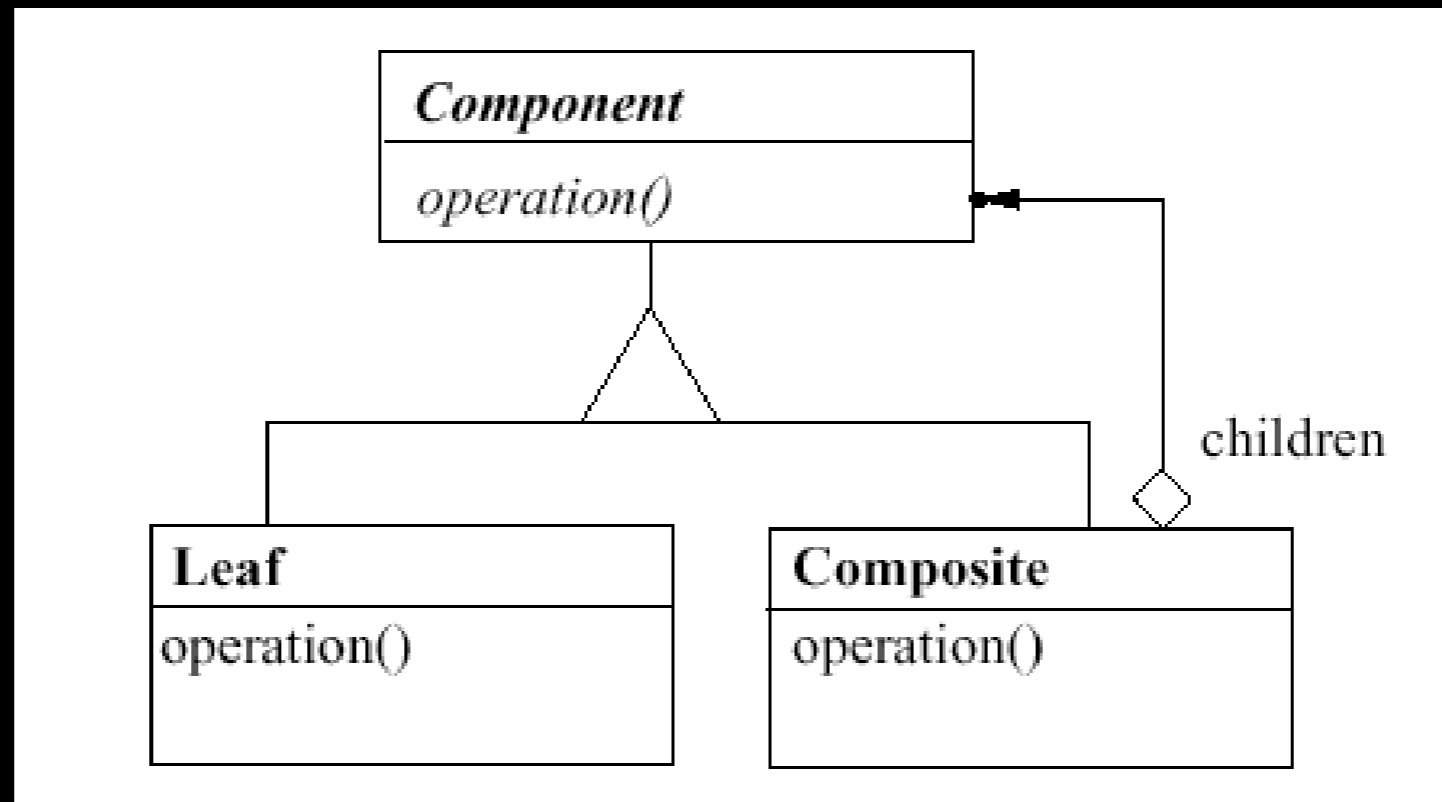
Find
structure

```
compositeStructure(?comp, ?composite) if
  class(?comp),
  classInHierarchyOf(?composite, ?comp)
```

Example continued

```
Composite>>operation  
self children do[:child |  
  child operation]
```

Find
aggregate

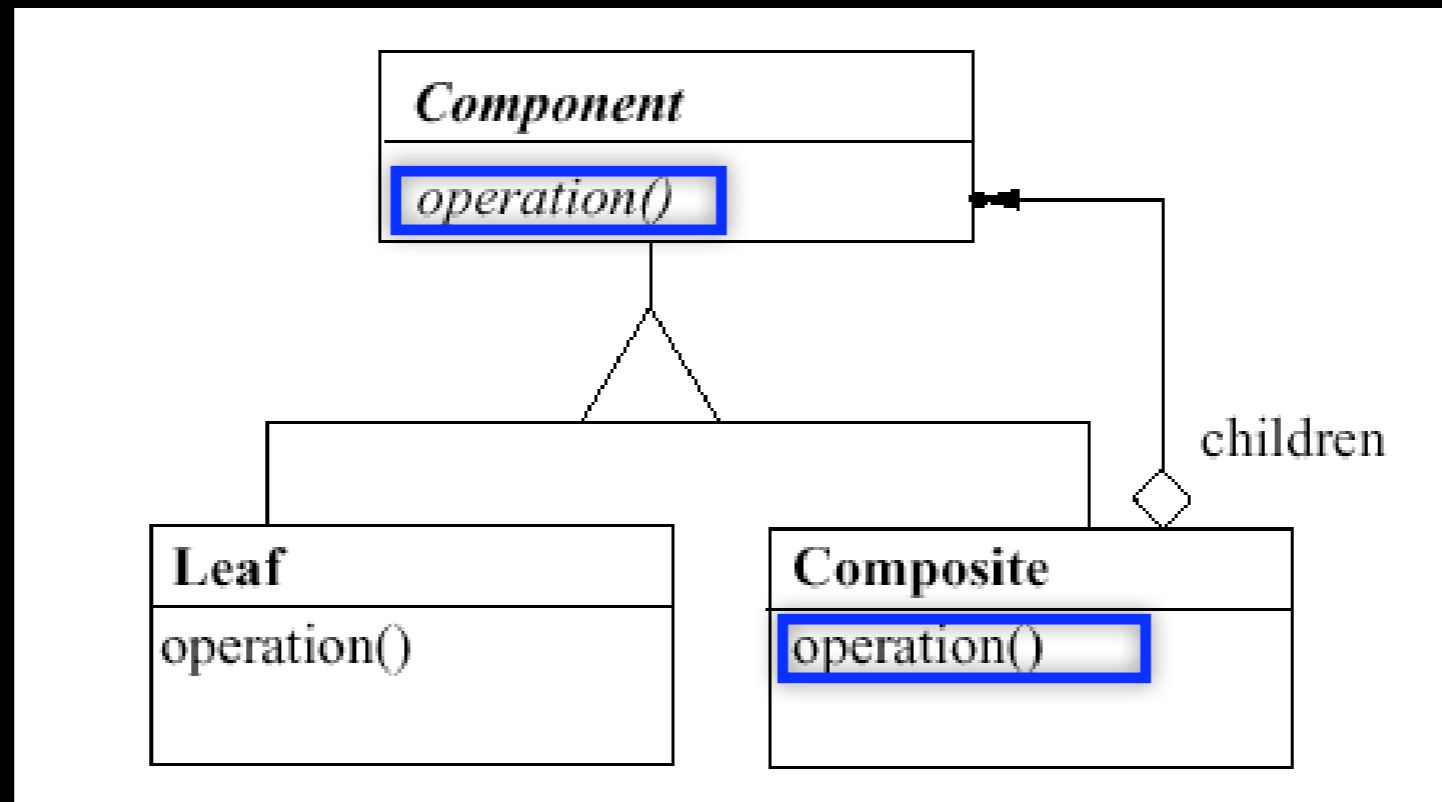


```
compositeAggregation(?comp, ?composite, ?msg) if  
  classesImplementCommonSelector(?comp, ?composite),  
  methodNameInClass(?method, ?msg, ?composite),  
  statementsOfMethod(?statement, ?method),  
  iterationStatementWithBlock(?statement, ?iterationBlock),  
  blockIteratesMessage(?iterationBlock, ?msg)
```

Example continued

```
Composite>>operation  
self children do[:child |  
  child operation]
```

Find
aggregate

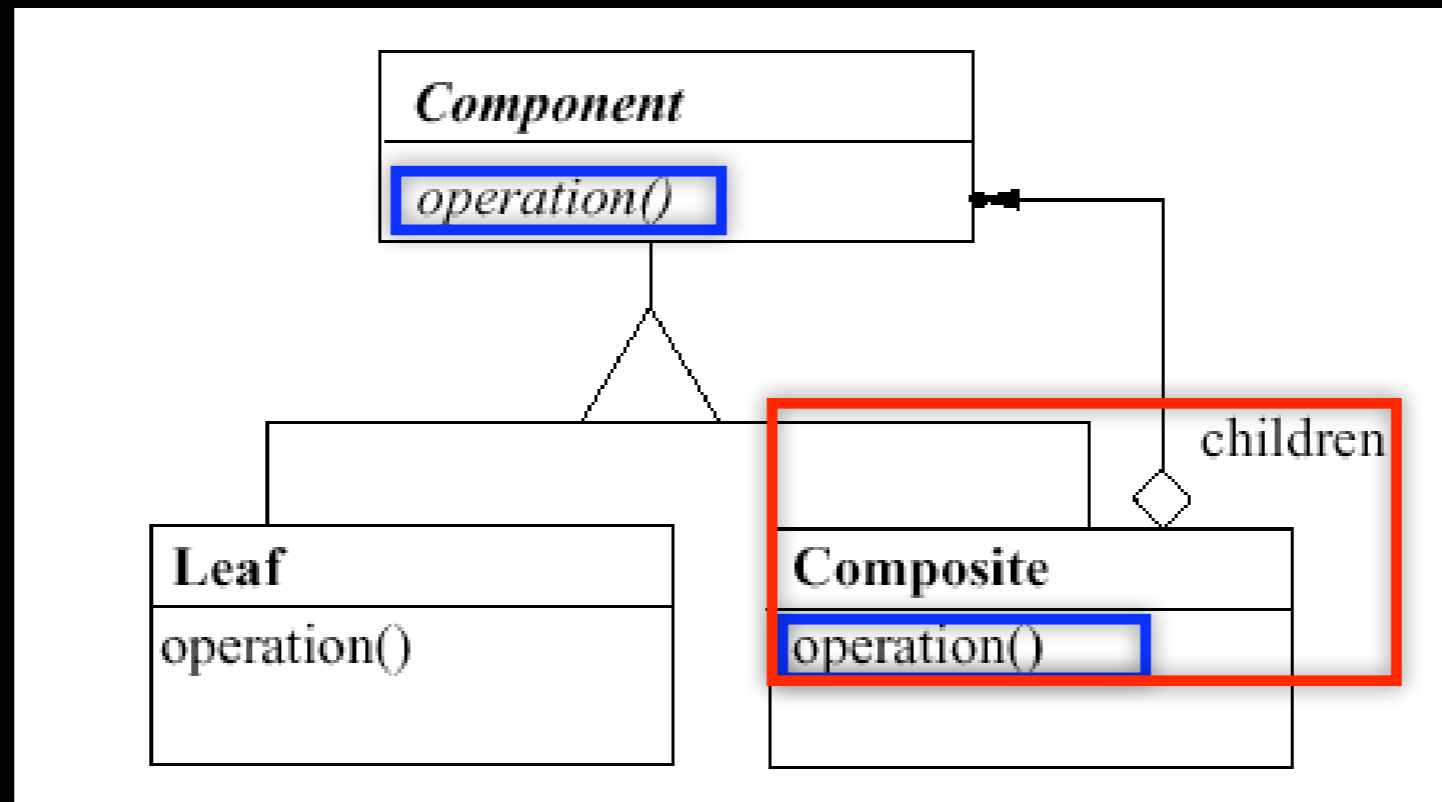


```
compositeAggregation(?comp, ?composite, ?msg) if  
classesImplementCommonSelector(?comp, ?composite),  
methodNameInClass(?method, ?msg, ?composite),  
statementsOfMethod(?statement, ?method),  
iterationStatementWithBlock(?statement, ?iterationBlock),  
blockIteratesMessage(?iterationBlock, ?msg)
```

Example continued

```
Composite>>operation  
self children do[:child |  
  child operation]
```

Find
aggregate



```
compositeAggregation(?comp, ?composite, ?msg) if  
classesImplementCommonSelector(?comp, ?composite),  
methodNameInClass(?method, ?msg, ?composite),  
statementsOfMethod(?statement, ?method),  
iterationStatementWithBlock(?statement, ?iterationBlock),  
blockIteratesMessage(?iterationBlock, ?msg)
```

Composites in Smalltalk

```
[?comp-->[Refactory.Browser.LintRule],  
?composite-->[Refactory.Browser.CompositeLintRule],  
?msg-->[#resetResult]]
```

```
[?comp-->[VisualComponent],  
?composite-->[CompositePart],  
?msg-->[#updateSpot:]]
```

```
[?comp-->[Refactory.HotDraw.Figure],  
?composite-->[Refactory.HotDraw.CompositeFigure],  
?msg-->[#release]]
```

```
[?comp-->[XML.Node],  
?composite-->[XML.Document],  
?msg-->[#printOn:]]
```

```
[?comp-->[Refactory.Browser.NavigatorPart],  
?composite-->[Refactory.Browser.TabNavigatorPart],  
?msg-->[#navigator:]]
```

**Queries through 7734 classes and 72962 methods
in 13 minutes to find 81 solutions**

Applications

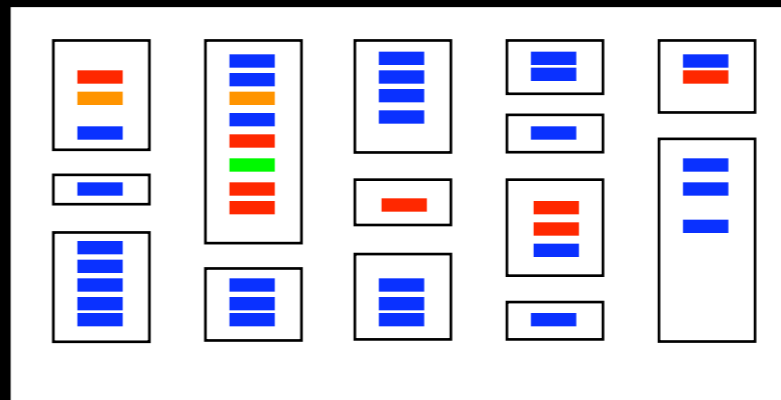
- ▶ **Applied to many problems:**
 - Co-evolution of design and source code
 - Reasoning about traces
 - Composing program generators
 - User interface specification
 - Aspect-oriented programming
 - Template querying

Applications

▶ Applied to many problems:

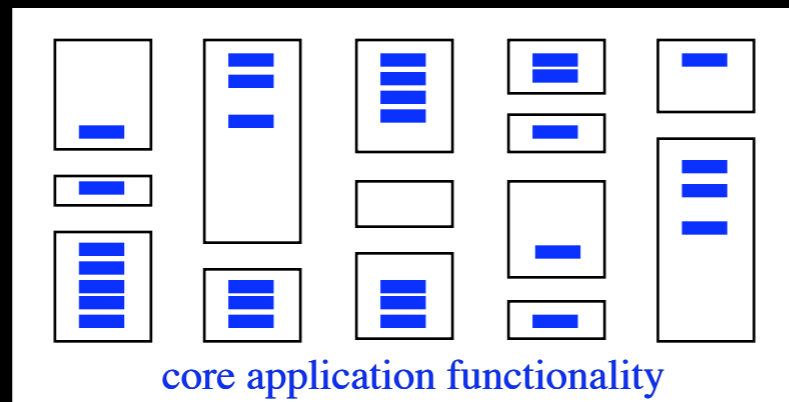
- Co-evolution of design and source code
- Reasoning about traces
- Composing program generators
- User interface specification
- Aspect-oriented programming
- Template querying

Aspect-oriented programming

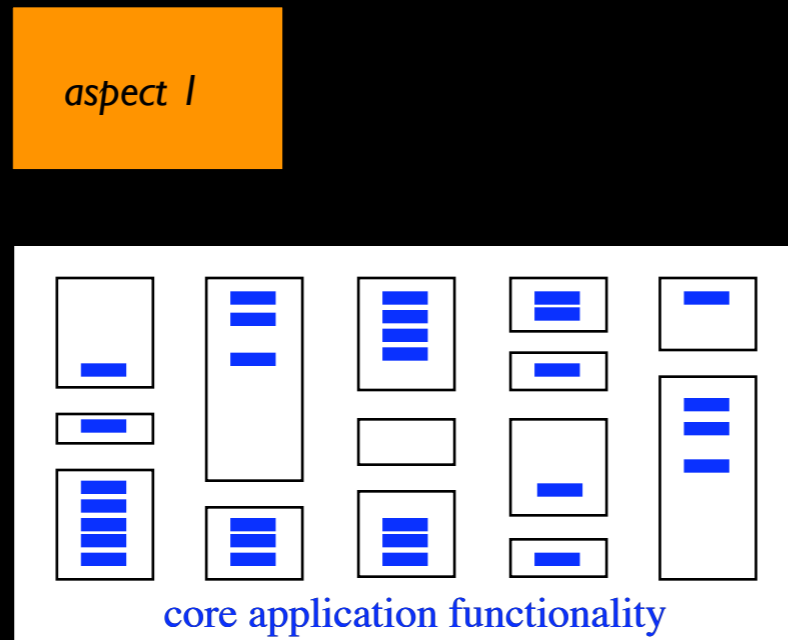


Aspect-oriented programming

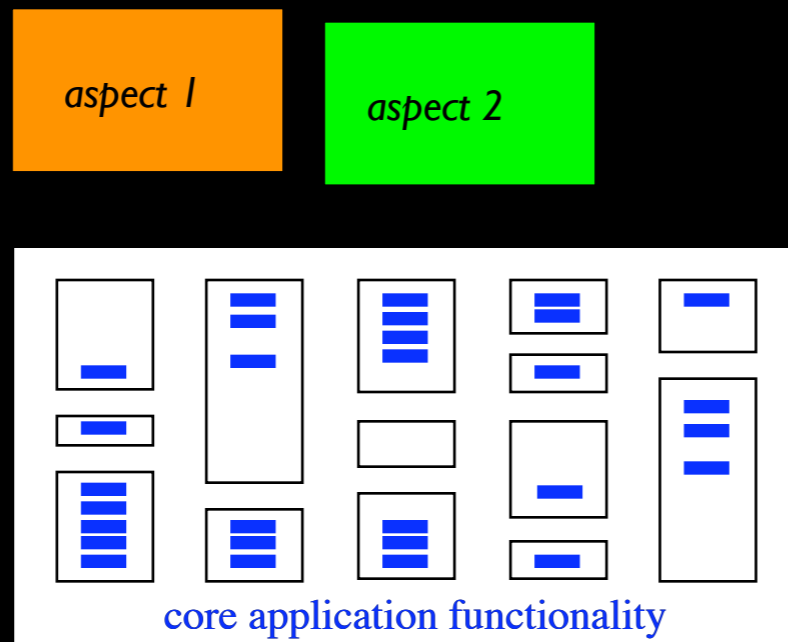
Aspect-oriented programming



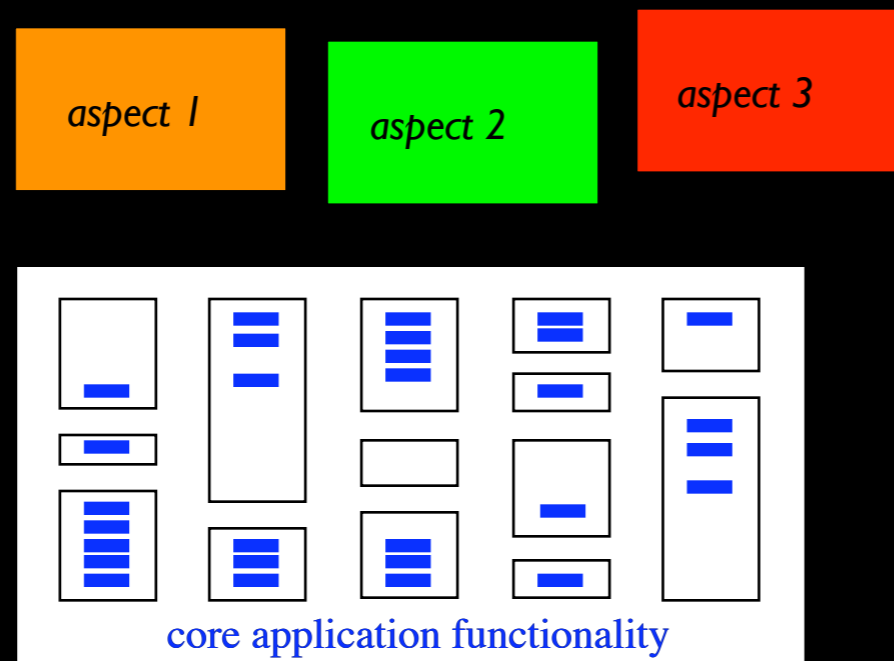
Aspect-oriented programming



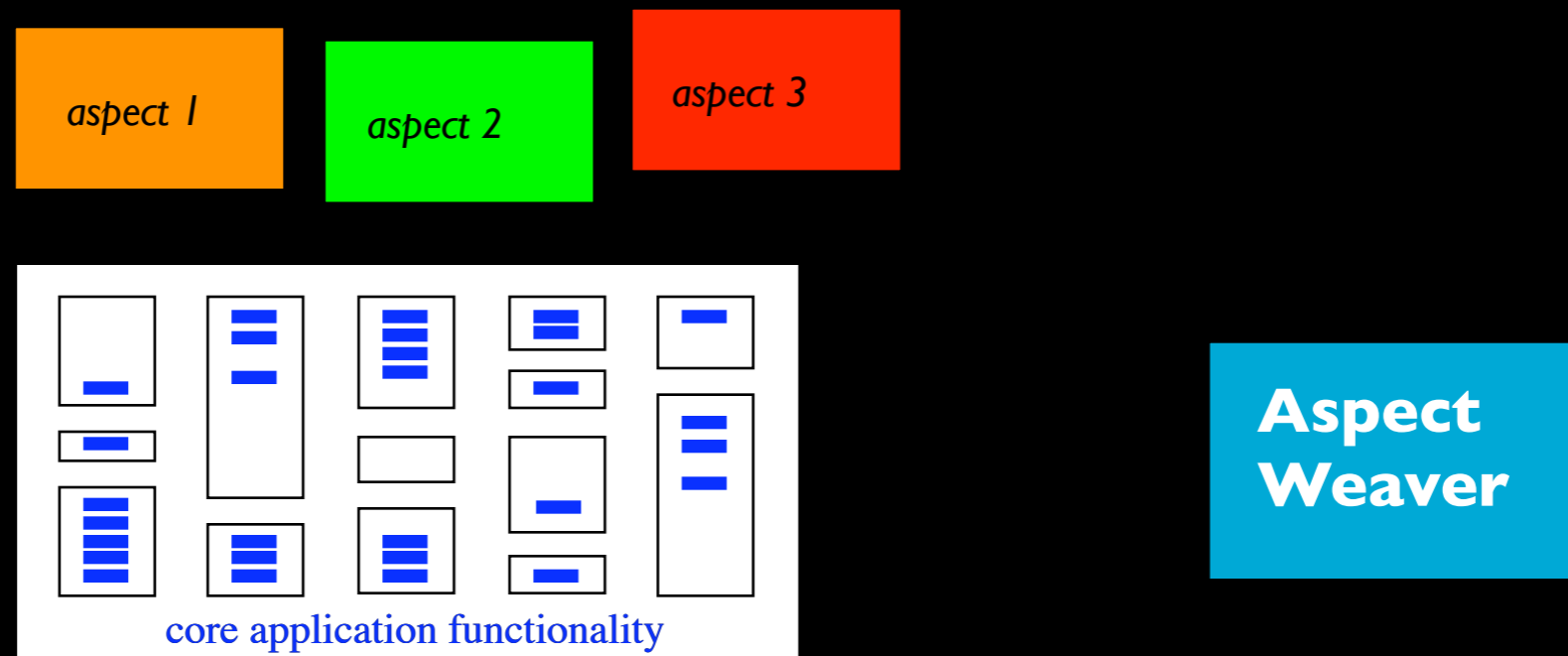
Aspect-oriented programming



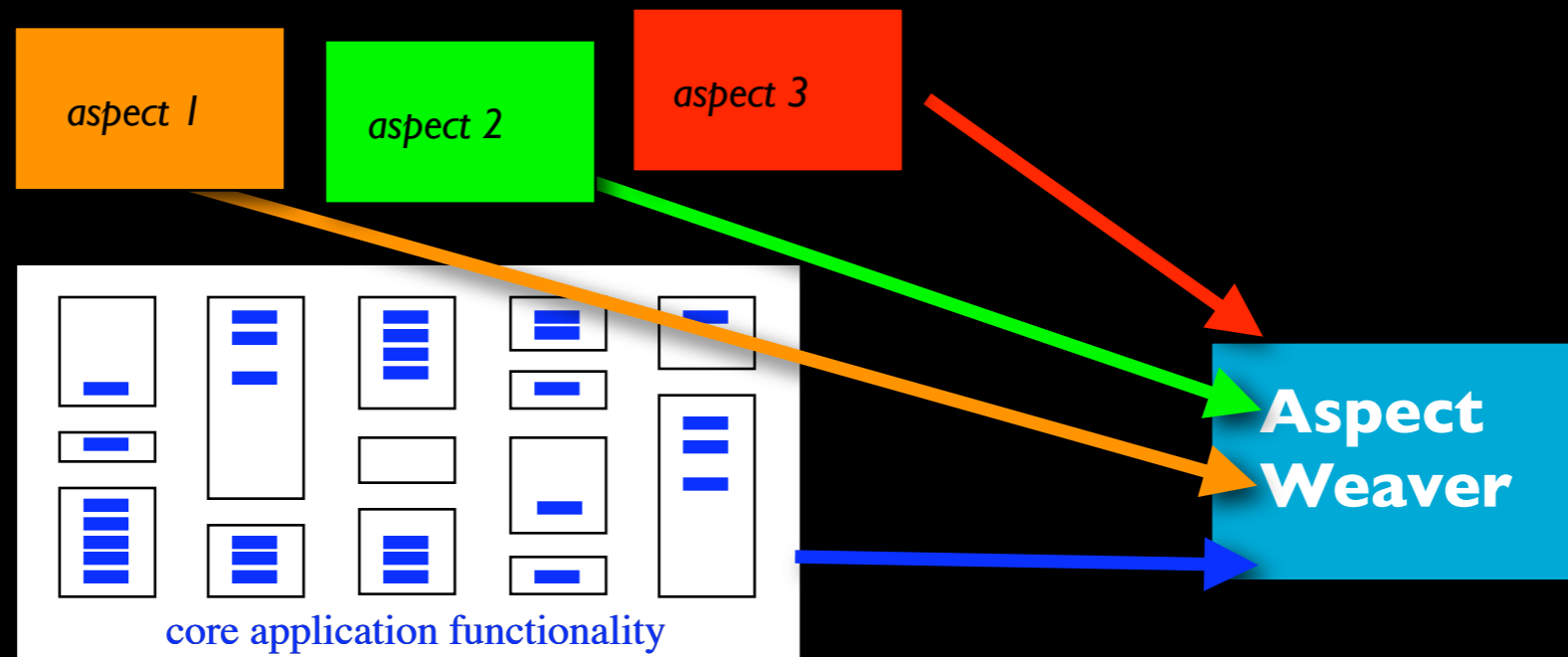
Aspect-oriented programming



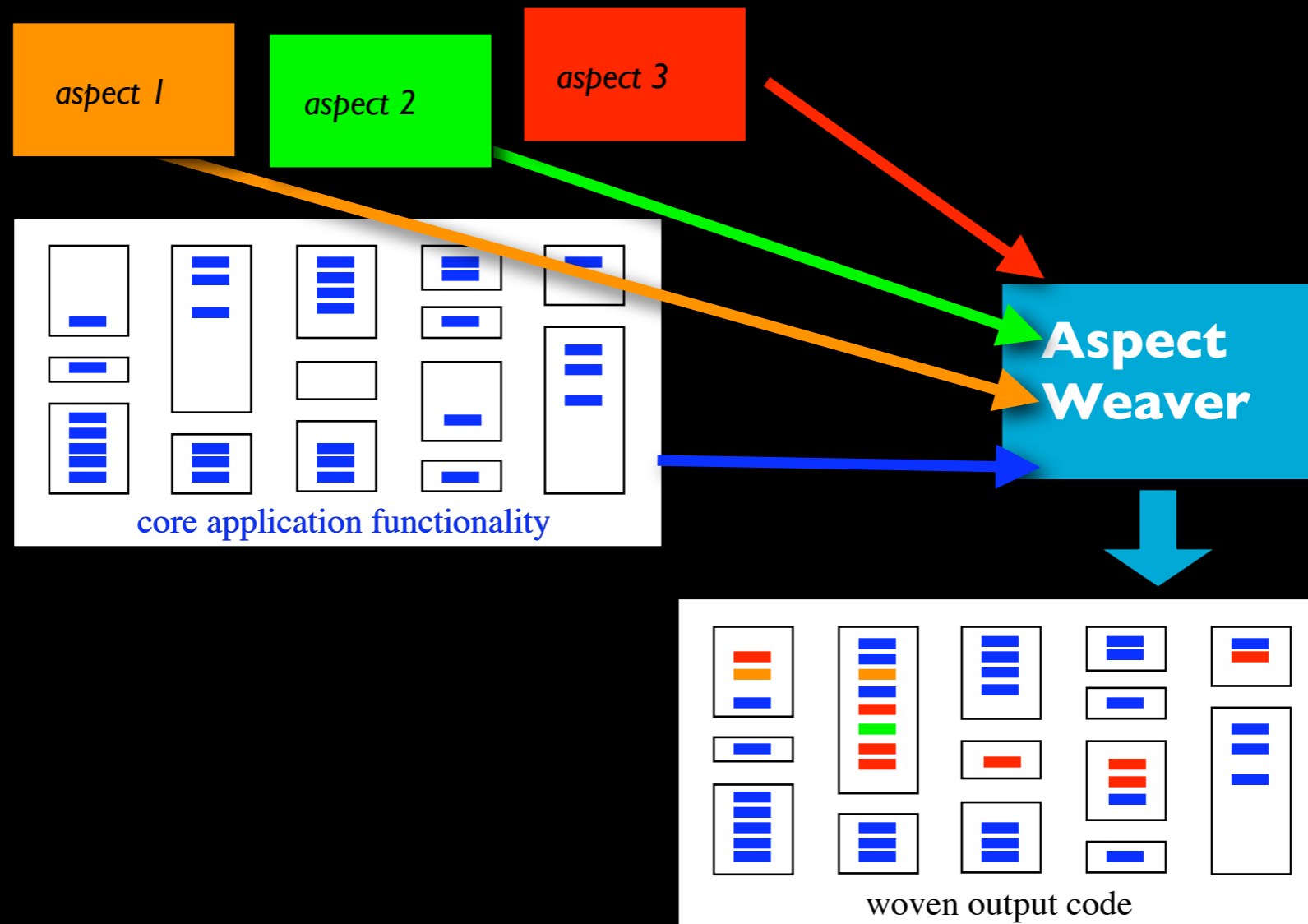
Aspect-oriented programming



Aspect-oriented programming



Aspect-oriented programming



CARMA [Gybels 03]

► Logic meta programs as pointcuts

Observer

```
changesState(?class, ?methodName) if
  shadowIn(?class, ?methodName, ?sp),
  assignmentShadow(?sp, ?variable)

changesState(?class, ?methodName) if
  shadowIn(?class, ?methodName, ?sp),
  messageShadow(?sp, ?rcvr, ?msg),
  selfReceiver(?rcvr),
  changesState(?class, ?msg)

after ?jp matching
  reception(?jp, ?msg, ?args),
  inObject(?jp, ?obj),
  objectClass(?obj, ?class),
  changesState(?class, ?msg),
  not(caller(?jp, ?obj))
do
  observers notify
```

CARMA [Gybels 03]

► Logic meta programs as pointcuts

Observer

```
changesState(?class, ?methodName) if  
  shadowIn(?class, ?methodName, ?sp),  
  assignmentShadow(?sp, ?variable)
```

```
changesState(?class, ?methodName) if  
  shadowIn(?class, ?methodName, ?sp),  
  messageShadow(?sp, ?rcvr, ?msg),  
  selfReceiver(?rcvr),  
  changesState(?class, ?msg)
```

```
after ?jp matching  
  reception(?jp, ?msg, ?args),  
  inObject(?jp, ?obj),  
  objectClass(?obj, ?class),  
  changesState(?class, ?msg),  
  not(caller(?jp, ?obj))  
do  
  observers notify
```

All methods that
change the state of
an object

CARMA [Gybels 03]

► Logic meta programs as pointcuts

Observer

```
changesState(?class, ?methodName) if  
  shadowIn(?class, ?methodName, ?sp),  
  assignmentShadow(?sp, ?variable)
```

```
changesState(?class, ?methodName) if  
  shadowIn(?class, ?methodName, ?sp),  
  messageShadow(?sp, ?rcvr, ?msg),  
  selfReceiver(?rcvr),  
  changesState(?class, ?msg)
```

```
after ?jp matching  
  reception(?jp, ?msg, ?args),  
  inObject(?jp, ?obj),  
  objectClass(?obj, ?class),  
  changesState(?class, ?msg),  
  not(caller(?jp, ?obj))
```

```
do  
  observers notify
```

All methods that
change the state of
an object

Pointcut: intervene at
which run-time points?

CARMA [Gybels 03]

► Logic meta programs as pointcuts

Observer

```
changesState(?class, ?methodName) if  
  shadowIn(?class, ?methodName, ?sp),  
  assignmentShadow(?sp, ?variable)  
  
changesState(?class, ?methodName) if  
  shadowIn(?class, ?methodName, ?sp),  
  messageShadow(?sp, ?rcvr, ?msg),  
  selfReceiver(?rcvr),  
  changesState(?class, ?msg)
```

```
after ?jp matching  
  reception(?jp, ?msg, ?args),  
  inObject(?jp, ?obj),  
  objectClass(?obj, ?class),  
  changesState(?class, ?msg),  
  not(caller(?jp, ?obj))
```

```
do  
  observers notify
```

All methods that
change the state of
an object

Pointcut: intervene at
which run-time points?

Advice: what to do?

Template querying

- ▶ Write down a piece of code with variables in it as a query
- ▶ Use static analysis to increase precision

```
if jtClassDeclaration(?c){
class ?c {
    private ?type ?field;
    public ?type ?name() { return ?field; }}
}
if jtClassDeclaration(?c){
class ?c {
    private ?type ?field;
    public ?rt ?name(?type ?var) { ?field = ?var; }}
}
```


Declarative Meta Programming

▶ For more information:

- <http://prog.vub.ac.be/DMP>
- <http://prog.vub.ac.be/SOUL>