Engineering Abstractions in Model Checking and Testing

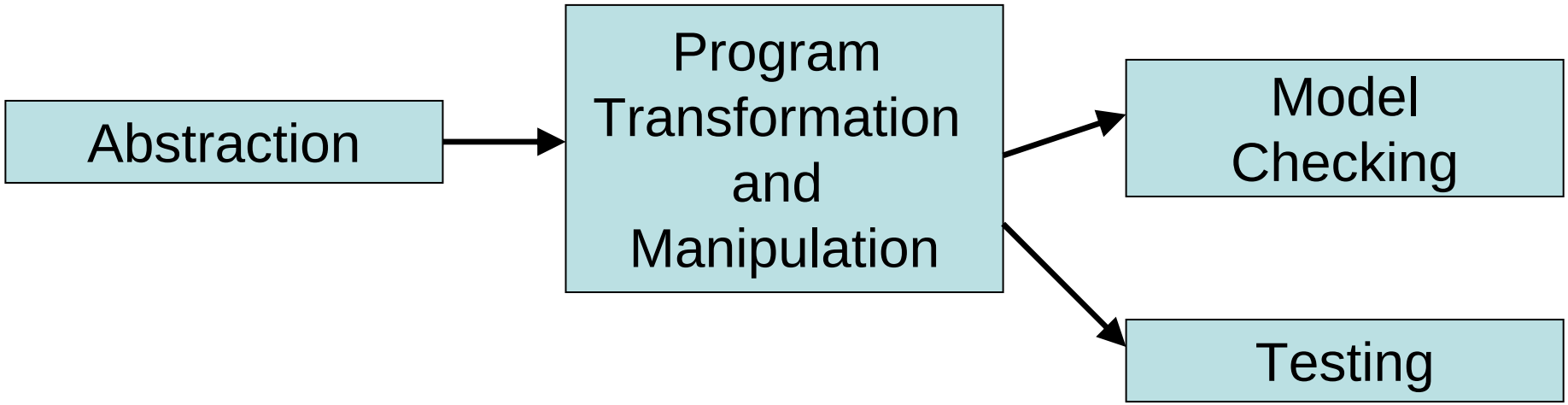
Michael Achenbach

Klaus Ostermann

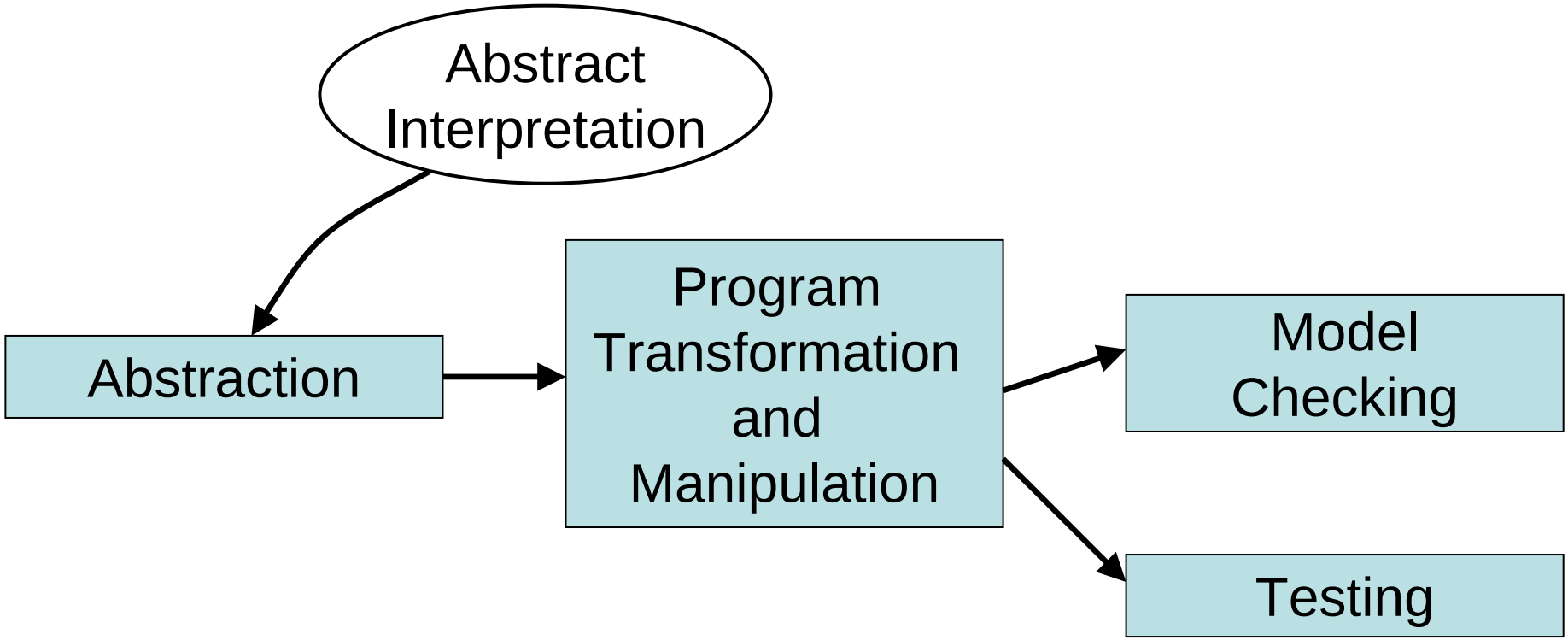
This Talk

- What is abstraction engineering?
- How can we integrate abstractions with current tools?
- What more is required in the future?

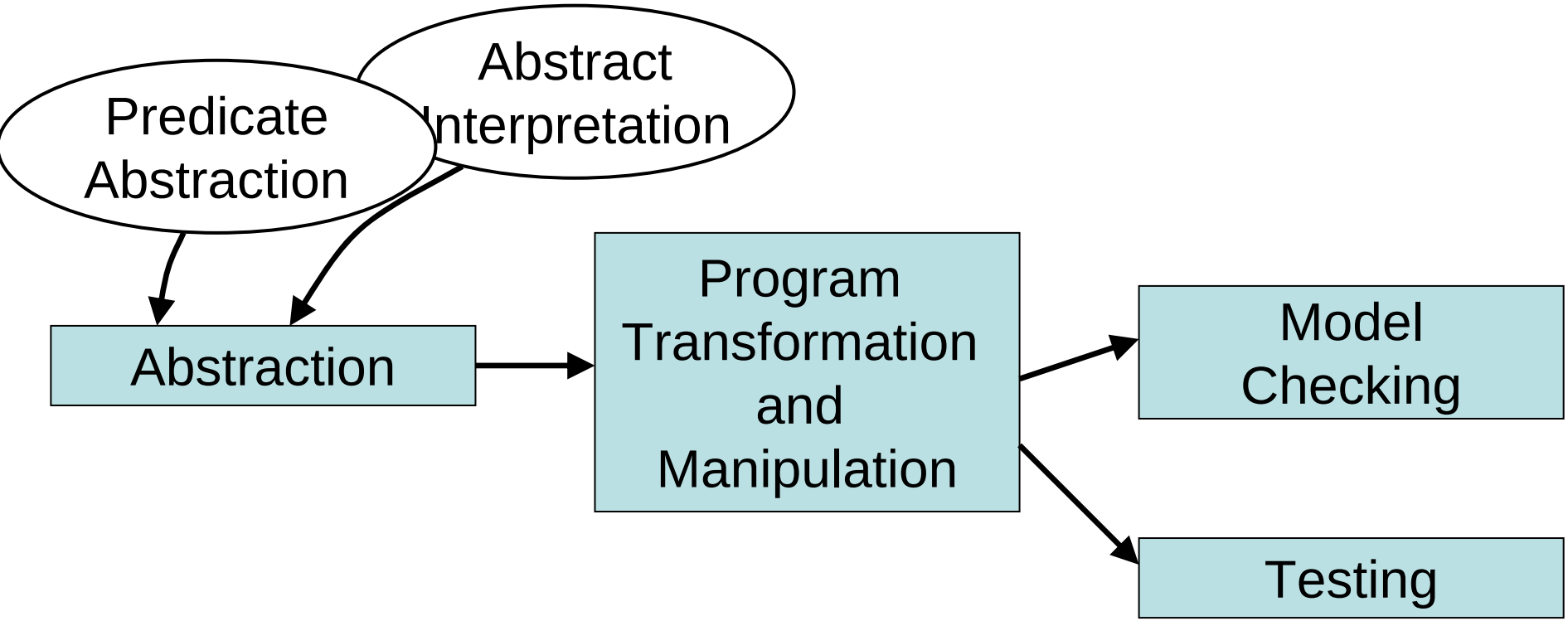
Abstraction Engineering



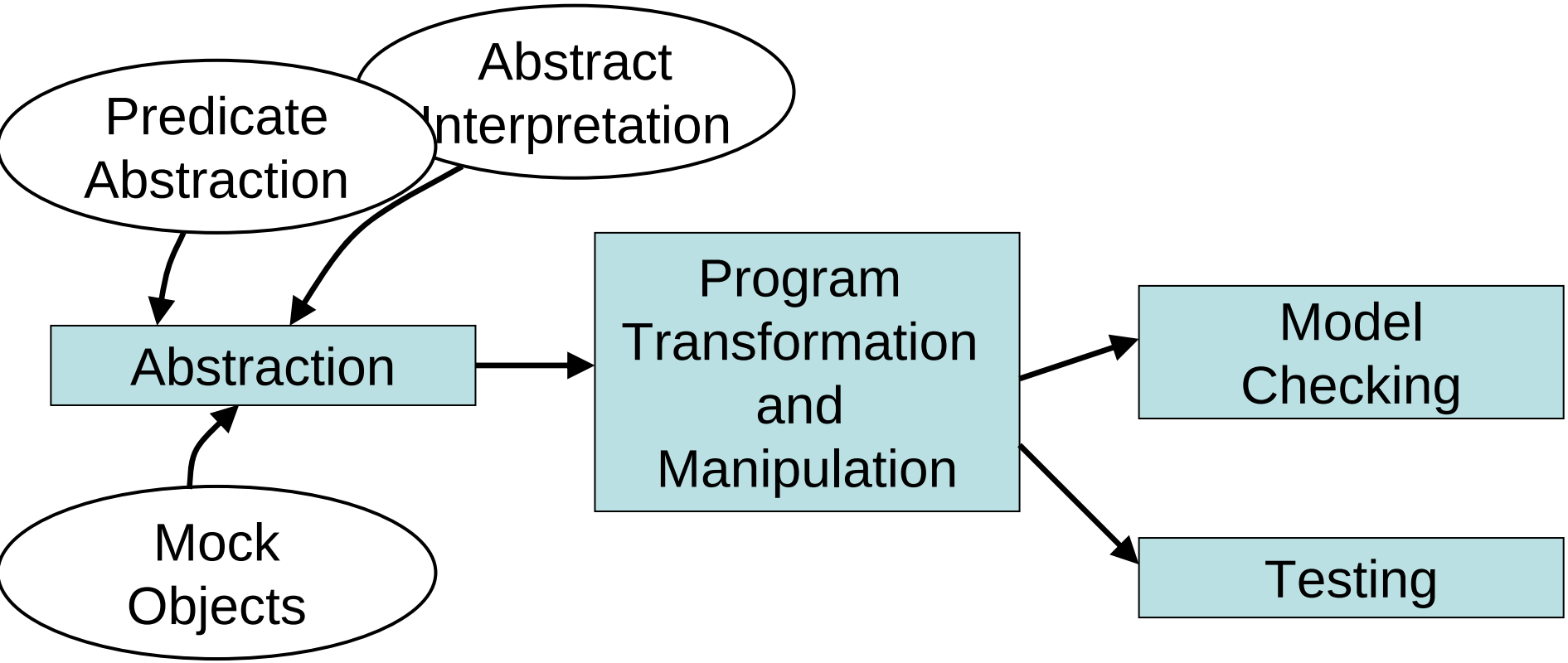
Abstraction Engineering



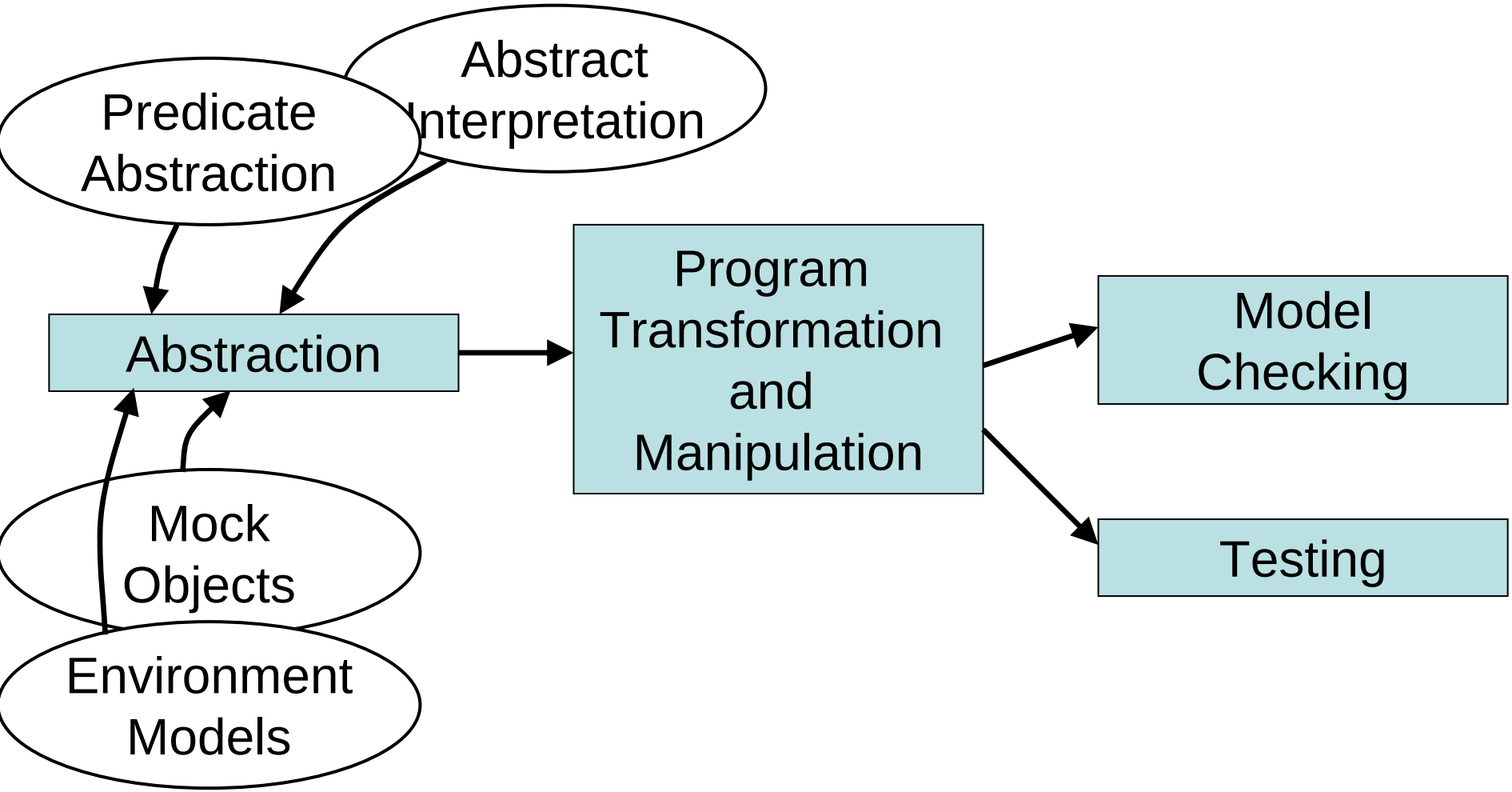
Abstraction Engineering



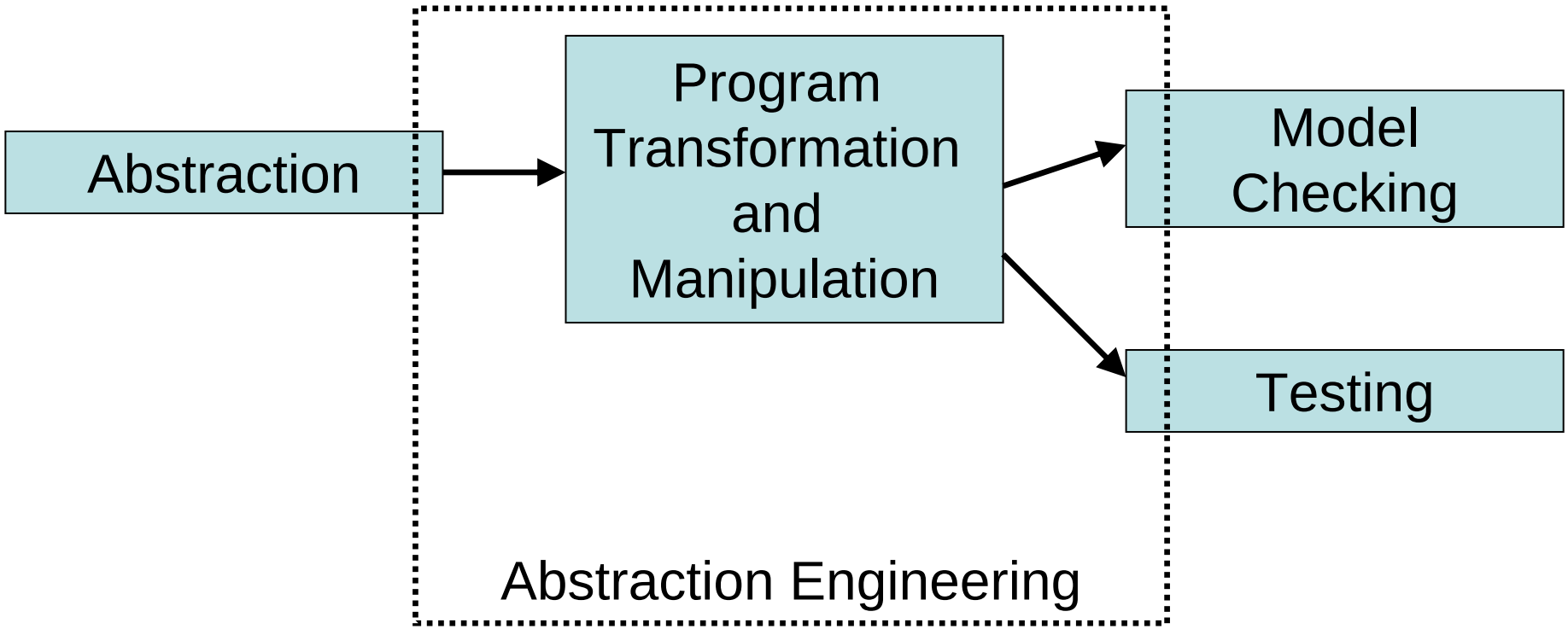
Abstraction Engineering



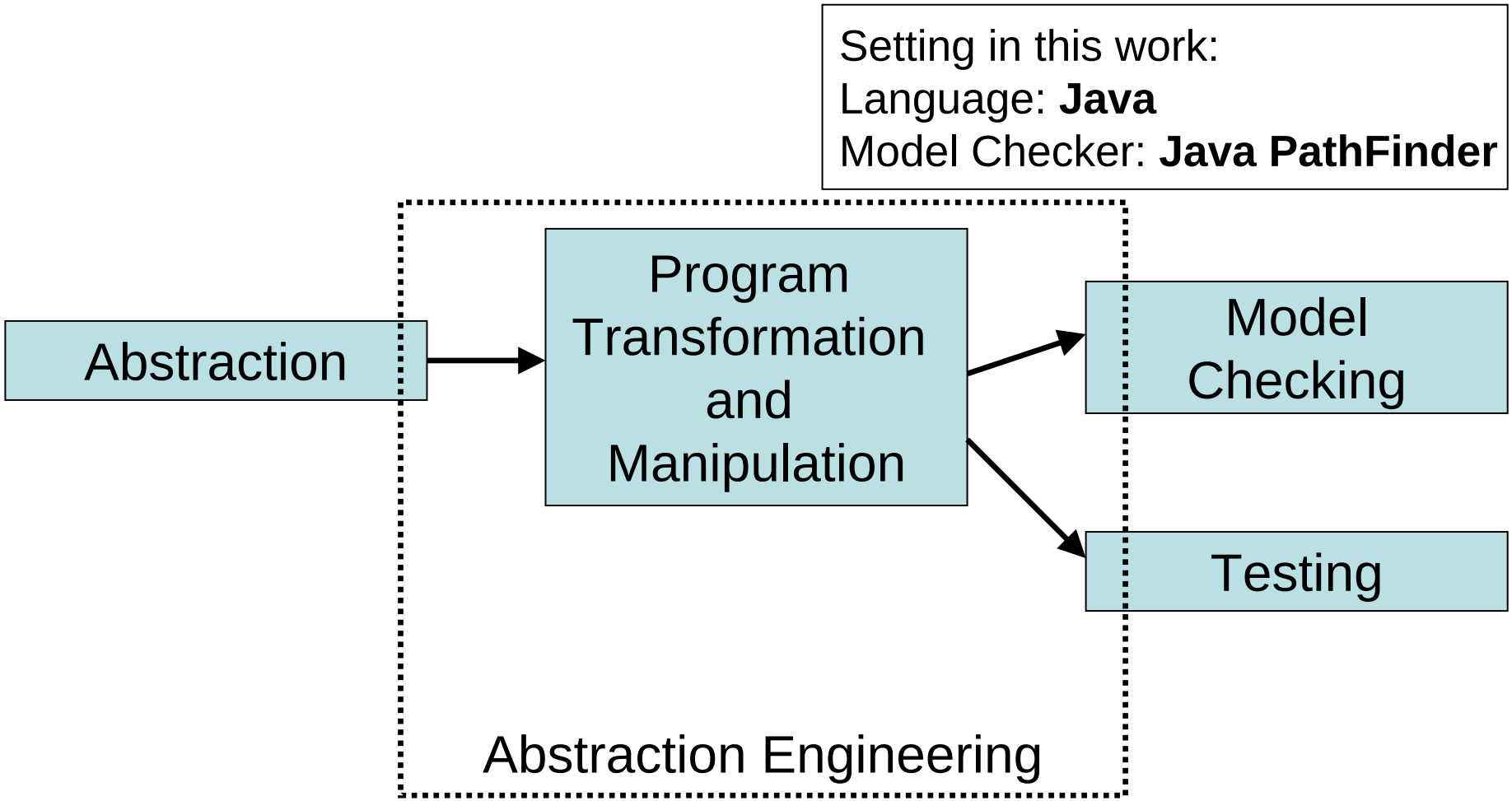
Abstraction Engineering



Abstraction Engineering



Abstraction Engineering



Our Notion of Abstraction

- Implementation variant of a class
  - Has bigger or smaller interface
  - Checks also specifications
  - Generates test input
  - Uses non-determinism, pruning
- Engineering difficulties in integration
  - Exposure of local variables
  - Access to private data
  - Unwanted auxiliary methods

How to integrate abstractions into a software system?

Example: Reading Files – Test Robustness

```
FileReader fileReader = null;
try {
 fileReader = new FileReader(inputFile);
 BufferedReader in = new
  BufferedReader(fileReader);
 try {
  while (in.readLine() != null) ...;
 } finally { in.close(); }
} catch (IOException e) {log(e)}
finally {
 try { fileReader.close(); }
 catch (IOException e) {log(e)}
}
```

Example: Reading Files – Test Robustness

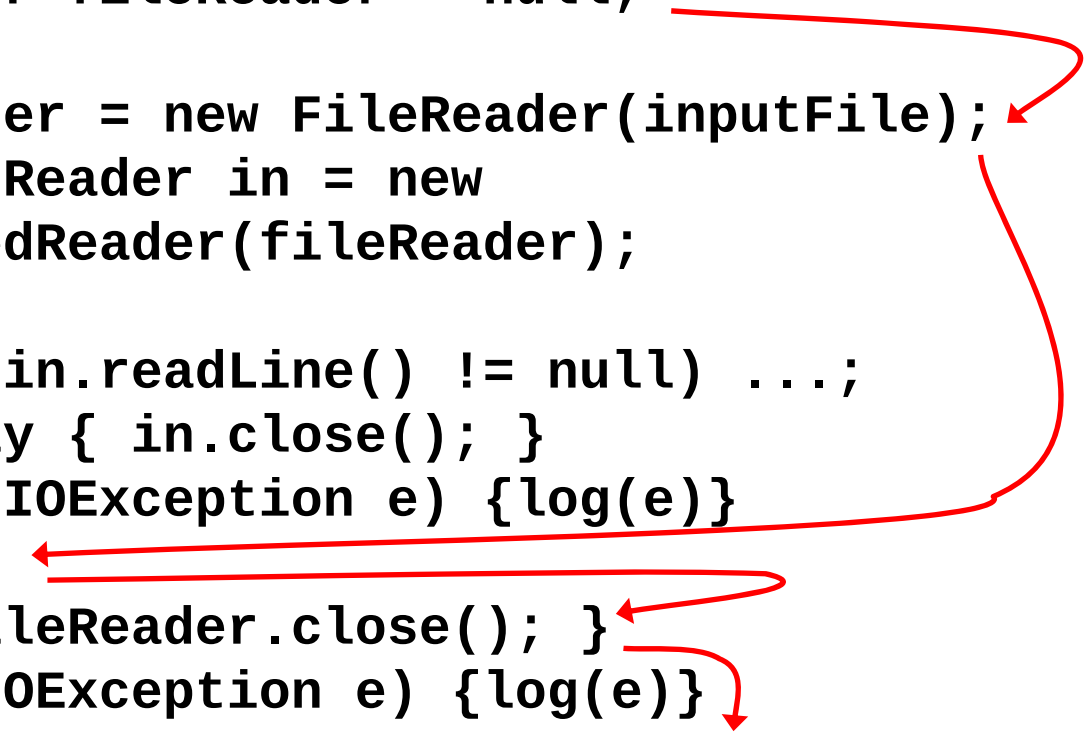
```
FileReader fileReader = nu
try {
 fileReader = new FileReader(i
 BufferedReader in = new
  BufferedReader(fileReader);
 try {
  while (in.readLine() != null) ...;
 } finally { in.close(); }
} catch (IOException e) {log(e)}
finally {
 try { fileReader.close(); }
 catch (IOException e) {log(e)}
}
```

Use explorative method:

- Check all possible exceptions
- Check if streams are always closed

Example: Reading Files – Test Robustness

```
FileReader fileReader = null;
try {
 fileReader = new FileReader(inputFile);
 BufferedReader in = new
  BufferedReader(fileReader);
 try {
  while (in.readLine() != null) ...;
 } finally { in.close(); }
} catch (IOException e) {log(e)}
finally {
 try { fileReader.close(); }
 catch (IOException e) {log(e)}
}
```



Reveal possible null pointer exception

Example: Reading Files – Test Robustness

```
FileReader fileReader = null;
try {
 fileReader = new FileReader(inputFile);
 BufferedReader in = new
  BufferedReader(fileRead
 try {
  while (in.readLine() != null)
 } finally { in.close(); }
} catch (IOException e) {log(e)}
finally {
 try { fileReader.close(); }
 catch (IOException e) {log(e)}
}
```

Hard to test with concrete file reader…

Use abstraction!

Example: Reading Files – Test Robustness

```
FileReader fileReader = null;
try {
 fileReader = new FileReader(inpu
 BufferedReader in = new
  BufferedReader(fileReader);
 try {
  while (in.readLine() != null) ...;
 } finally { in.close(); }
} catch (IOException e) {log(e)}
finally {
 try { fileReader.close(); }
 catch (IOException e) {log(e)}
}
```

Avoid invasive modification of local variables and inheritance hierarchy

Example: Reading Files – Test Robustness

```
FileReader fileReader = null;
try {
 fileReader = new FileReader(inpu
 BufferedReader in = new
  BufferedReader(fileReader);
 try {
  while (in.readLine() != null) ...;
 } finally { in.close(); }
} catch (IOException e) {log(e)}
finally {
 try { fileReader.close(); }
 catch (IOException e) {log(e)}
}
```

Avoid invasive modification of local variables and inheritance hierarchy

More than one concrete FileReader is in use

Example: Reading Files – Test Robustness

```
FileReader fileReader = null;
try {
 fileReader = new FileReader(inpu
 BufferedReader in = new
  BufferedReader(fileReader);
 try {
  while (in.readLine() != null) ...;
 } finally { in.close(); }
} catch (IOException e) {log(e)}
finally {
                          (); }
                          {log(e)}
```

Avoid invasive modification of local variables and inheritance hierarchy

Advanced dynamic scoping mechanisms required

More than one concrete FileReader is in use

Abstraction of File and FileReader

```
class MyFile abstracts File {
 boolean exists(){
  return ?;
 }
}

class MyFileReader abstracts FileReader {
 public MyFileReader (MyFile f) {
  if(!f.exists()) throw new FileNotFoundException();
 }

 public void close() {
  if(?) throw new IOException();
 }
}
```

Abstraction of File and FileReader

```
class MyFile abstracts File {
 boolean exists(){
  return ?;
 }
}

class MyFileReader abstracts FileReader {
 public MyFileReader (MyFile f) {
  if(!f.exists()) throw new FileNotFoundException();
 }

 public void close() {
  if(?) throw new IOException();
 }
}
```

Non-deterministic choice of the model checker

Abstraction of File and FileReader

```
class MyFile abstracts File {
 boolean exists(){
  return ?;
 }
}

class MyFileReader abstracts FileReader {
 public MyFileReader (MyFile f) {
  if(!f.exists()) throw new FileNotFoundException();
 }

 public void close() {
  if(?) throw new IOException();
 }
}
```

Non-deterministic choice of the model checker

Bigger interface

Analysis of Current Tools

- Setting:
  - Finding bugs in Java programs
  - Program exploration with Java PathFinder (JPF)
- Tools for modularization:
  - AspectJ
  - Javassist
  - Java Model Interface (JMI) of JPF

AspectJ

- Inter-type declarations allow bigger interfaces
- Type-safe
- Mandatory super constructor problem: No class exchange
- Number of inter-type fields grows with each abstraction

Javassist and JMI

- Allow implementation alternative of whole class
- No static safety
- Only one abstraction at runtime

Future Work

- Formalize requirements of abstraction engineering
  - Scoping / multiple class versions
  - Type safety
- Evaluate dynamic language features:
  - Dynamic class exchange
  - Dynamic abstraction conversion
- Analyze dynamic languages: Ruby, Python, LUA, etc.

Conclusions

- Engineering problem: How to integrate abstractions into a software system?
- Useful abstraction engineering tasks supported by current tools **but**
- Tools lack expressiveness/local scoping

Model checking without abstraction engineering has no future in practice!

Thank You!