Algorithmic Verification of Procedural Programs in the Presence of Code Variability

Siavash Soleimanifard

School of Computer Science and Communication
KTH Royal Institute of Technology
Stockholm

Doctoral Thesis Presentation
19 September 2014

Problem Statement

Goal

- Algorithmic and practical verification

Properties

- Temporal properties (we focus on safety properties)

Systems under consideration

- Sequential procedural programs with unbounded recursive calls
- In the presence of code variability

Variability Scenarios

Incomplete Programs

- E.g., open platforms

Mobile Code

- E.g., add-ons, extensions

Code Evolution

- E.g., application updates, self adaptive systems

Multiple Implementations

- E.g., product families

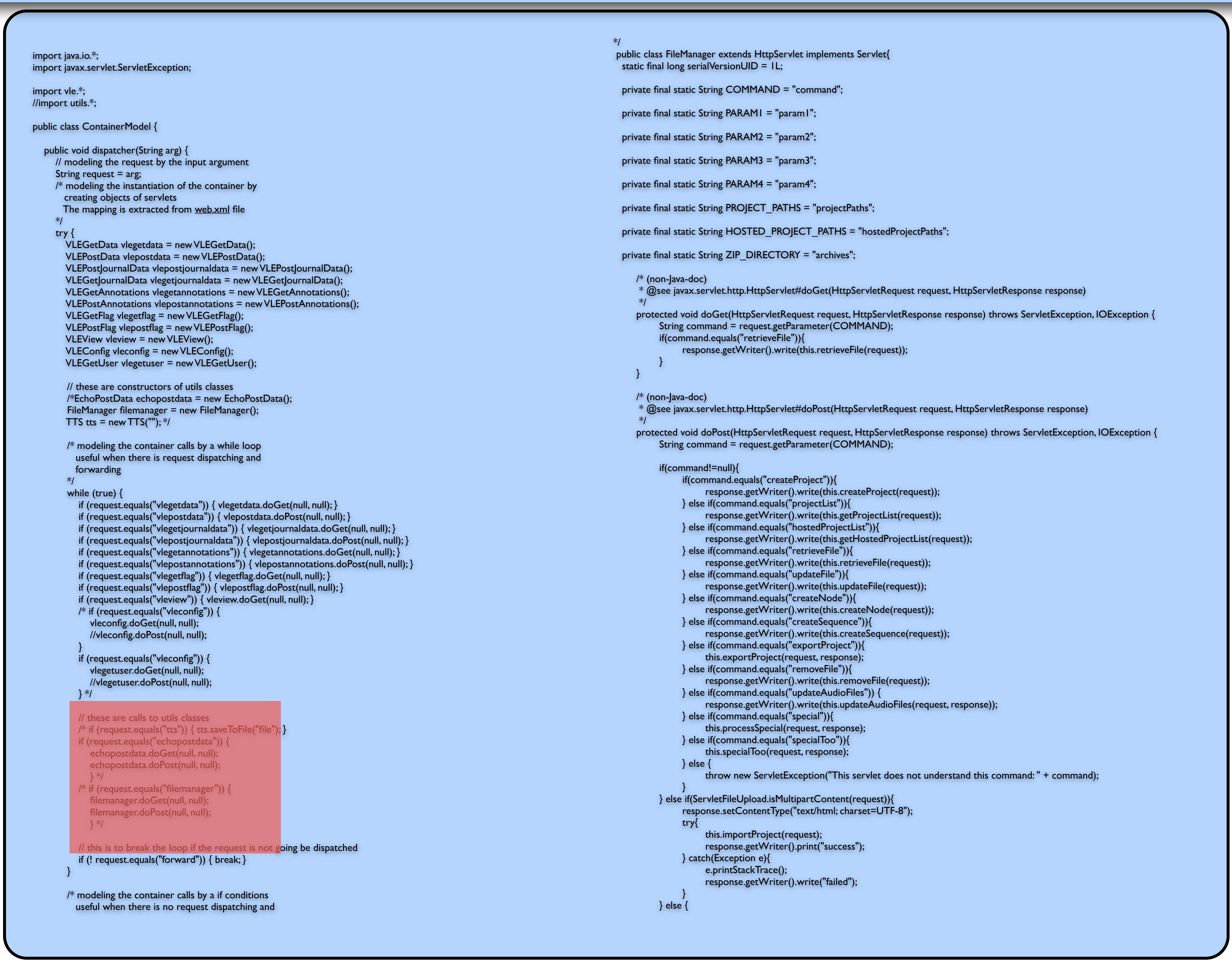
Variability

```
import java.io.*;
import javax.servlet.ServletException;

import vle.*;
//import utils.*;

public class ContainerModel {

    public void dispatcher(String arg) {
        // modeling the request by the input argument
        String request = arg;
        /* modeling the instantiation of the container by
           creating objects of servlets
          The mapping is extracted from web.xml file
        */
        try {
          VLEGetData vlegetdata = new VLEGetData();
          VLEPostData vlepostdata = new VLEPostData();
          VLEPostJournalData vlepostjournaldata = new VLEPostJournalData();
          VLEGetJournalData vlegetjournaldata = new VLEGetJournalData();
          VLEGetAnnotations vlegetannotations = new VLEGetAnnotations();
          VLEPostAnnotations vlepostannotations = new VLEPostAnnotations();
          VLEGetFlag vlegetflag = new VLEGetFlag();
          VLEPostFlag vlepostflag = new VLEPostFlag();
          VLEView vleview = new VLEView();
          VLEConfig vleconfig = new VLEConfig();
          VLEGetUser vlegetuser = new VLEGetUser();

          // these are constructors of utils classes
          /*EchoPostData echopostdata = new EchoPostData();
          FileManager filemanager = new FileManager();
          TTS tts = new TTS(""); */

          /* modeling the container calls by a while loop
             useful when there is request dispatching and
             forwarding
          */
          while (true) {
              if (request.equals("vlegetdata")) { vlegetdata.doGet(null, null); }
              if (request.equals("vlepostdata")) { vlepostdata.doPost(null, null); }
              if (request.equals("vlegetjournaldata")) { vlegetjournaldata.doGet(null, null); }
              if (request.equals("vlepostjournaldata")) { vlepostjournaldata.doPost(null, null); }
              if (request.equals("vlegetannotations")) { vlegetannotations.doGet(null, null); }
              if (request.equals("vlepostannotations")) { vlepostannotations.doPost(null, null); }
              if (request.equals("vlegetflag")) { vlegetflag.doGet(null, null); }
              if (request.equals("vlepostflag")) { vlepostflag.doPost(null, null); }
              if (request.equals("vleview")) { vleview.doGet(null, null); }
              /* if (request.equals("vleconfig")) {
                  vleconfig.doGet(null, null);
                  //vleconfig.doPost(null, null);
              }
              if (request.equals("vleconfig")) {
                  vlegetuser.doGet(null, null);
                  //vlegetuser.doPost(null, null);
              } */

              // these are calls to utils classes
              /* if (request.equals("tts")) { tts.saveToFile("file"); }
              if (request.equals("echopostdata")) {
                  echopostdata.doGet(null, null);
                  echopostdata.doPost(null, null);
                  } */
              /* if (request.equals("filemanager")) {
                  filemanager.doGet(null, null);
                  filemanager.doPost(null, null);
                  } */

              // this is to break the loop if the request is not going be dispatched
              if (! request.equals("forward")) { break; }
          }

          /* modeling the container calls by a if conditions
             useful when there is no request dispatching and
```

```
*/
 public class FileManager extends HttpServlet implements Servlet{
   static final long serialVersionUID = 1L;

   private final static String COMMAND = "command";

   private final static String PARAM1 = "param1";

   private final static String PARAM2 = "param2";

   private final static String PARAM3 = "param3";

   private final static String PARAM4 = "param4";

   private final static String PROJECT_PATHS = "projectPaths";

   private final static String HOSTED_PROJECT_PATHS = "hostedProjectPaths";

   private final static String ZIP_DIRECTORY = "archives";

      /* (non-Java-doc)
       * @see javax.servlet.http.HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
       */
      protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
              String command = request.getParameter(COMMAND);
              if(command.equals("retrieveFile")){
                      response.getWriter().write(this.retrieveFile(request));
              }
      }

      /* (non-Java-doc)
       * @see javax.servlet.http.HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
       */
      protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
              String command = request.getParameter(COMMAND);

              if(command!=null){
                      if(command.equals("createProject")){
                              response.getWriter().write(this.createProject(request));
                      } else if(command.equals("projectList")){
                              response.getWriter().write(this.getProjectList(request));
                      } else if(command.equals("hostedProjectList")){
                              response.getWriter().write(this.getHostedProjectList(request));
                      } else if(command.equals("retrieveFile")){
                              response.getWriter().write(this.retrieveFile(request));
                      } else if(command.equals("updateFile")){
                              response.getWriter().write(this.updateFile(request));
                      } else if(command.equals("createNode")){
                              response.getWriter().write(this.createNode(request));
                      } else if(command.equals("createSequence")){
                              response.getWriter().write(this.createSequence(request));
                      } else if(command.equals("exportProject")){
                              this.exportProject(request, response);
                      } else if(command.equals("removeFile")){
                              response.getWriter().write(this.removeFile(request));
                      } else if(command.equals("updateAudioFiles")) {
                              response.getWriter().write(this.updateAudioFiles(request, response));
                      } else if(command.equals("special")){
                              this.processSpecial(request, response);
                      } else if(command.equals("specialToo")){
                              this.specialToo(request, response);
                      } else {
                              throw new ServletException("This servlet does not understand this command: " + command);
                      }
              } else if(ServletFileUpload.isMultipartContent(request)){
                      response.setContentType("text/html; charset=UTF-8");
                      try{
                              this.importProject(request);
                              response.getWriter().print("success");
                      } catch(Exception e){
                              e.printStackTrace();
                              response.getWriter().write("failed");
                      }
              } else {
```

$: \theta$

Variability

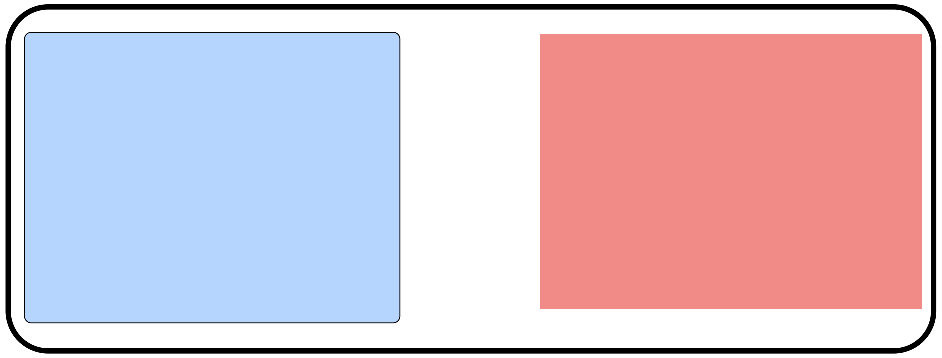


```
import java.io.*;
import javax.servlet.ServletException;

import vle.*;
//import utils.*;

public class ContainerModel {

    public void dispatcher(String arg) {
        // modeling the request by the input argument
        String request = arg;
        /* modeling the instantiation of the container by
           creating objects of servlets
          The mapping is extracted from web.xml file
        */
        try {
           VLEGetData vlegetdata = new VLEGetData();
           VLEPostData vlepostdata = new VLEPostData();
           VLEPostJournalData vlepostjournaldata = new VLEPostJournalData();
           VLEGetJournalData vlegetjournaldata = new VLEGetJournalData();
           VLEGetAnnotations vlegetannotations = new VLEGetAnnotations();
           VLEPostAnnotations vlepostannotations = new VLEPostAnnotations();
           VLEGetFlag vlegetflag = new VLEGetFlag();
           VLEPostFlag vlepostflag = new VLEPostFlag();
           VLEView vleview = new VLEView();
           VLEConfig vleconfig = new VLEConfig();
           VLEGetUser vlegetuser = new VLEGetUser();

           // these are constructors of utils classes
           /*EchoPostData echopostdata = new EchoPostData();
           FileManager filemanager = new FileManager();
           TTS tts = new TTS(""); */

           /* modeling the container calls by a while loop
              useful when there is request dispatching and
              forwarding
           */
           while (true) {
               if (request.equals("vlegetdata")) { vlegetdata.doGet(null, null); }
               if (request.equals("vlepostdata")) { vlepostdata.doPost(null, null); }
               if (request.equals("vlegetjournaldata")) { vlegetjournaldata.doGet(null, null); }
               if (request.equals("vlepostjournaldata")) { vlepostjournaldata.doPost(null, null); }
               if (request.equals("vlegetannotations")) { vlegetannotations.doGet(null, null); }
               if (request.equals("vlepostannotations")) { vlepostannotations.doPost(null, null); }
               if (request.equals("vlegetflag")) { vlegetflag.doGet(null, null); }
               if (request.equals("vlepostflag")) { vlepostflag.doPost(null, null); }
               if (request.equals("vleview")) { vleview.doGet(null, null); }
               /* if (request.equals("vleconfig")) {
                   vleconfig.doGet(null, null);
                   //vleconfig.doPost(null, null);
               }
               if (request.equals("vleconfig")) {
                   vlegetuser.doGet(null, null);
                   //vlegetuser.doPost(null, null);
               } */

               // these are calls to utils classes
               /* if (request.equals("tts")) { tts.saveToFile("file"); }
               if (request.equals("echopostdata")) {
                   echopostdata.doGet(null, null);
                   echopostdata.doPost(null, null);
                   } */
               /* if (request.equals("filemanager")) {
                   filemanager.doGet(null, null);
                   filemanager.doPost(null, null);
                   } */

               // this is to break the loop if the request is not going be dispatched
               if (! request.equals("forward")) { break; }
           }

           /* modeling the container calls by a if conditions
              useful when there is no request dispatching and
```

```
*/
 public class FileManager extends HttpServlet implements Servlet{
   static final long serialVersionUID = 1L;

   private final static String COMMAND = "command";

   private final static String PARAM1 = "param1";

   private final static String PARAM2 = "param2";

   private final static String PARAM3 = "param3";

   private final static String PARAM4 = "param4";

   private final static String PROJECT_PATHS = "projectPaths";

   private final static String HOSTED_PROJECT_PATHS = "hostedProjectPaths";

   private final static String ZIP_DIRECTORY = "archives";

      /* (non-Java-doc)
       * @see javax.servlet.http.HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
       */
      protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
              String command = request.getParameter(COMMAND);
              if(command.equals("retrieveFile")){
                      response.getWriter().write(this.retrieveFile(request));
              }
      }

      /* (non-Java-doc)
       * @see javax.servlet.http.HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
       */
      protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
              String command = request.getParameter(COMMAND);

              if(command!=null){
                      if(command.equals("createProject")){
                              response.getWriter().write(this.createProject(request));
                      } else if(command.equals("projectList")){
                              response.getWriter().write(this.getProjectList(request));
                      } else if(command.equals("hostedProjectList")){
                              response.getWriter().write(this.getHostedProjectList(request));
                      } else if(command.equals("retrieveFile")){
                              response.getWriter().write(this.retrieveFile(request));
                      } else if(command.equals("updateFile")){
                              response.getWriter().write(this.updateFile(request));
                      } else if(command.equals("createNode")){
                              response.getWriter().write(this.createNode(request));
                      } else if(command.equals("createSequence")){
                              response.getWriter().write(this.createSequence(request));
                      } else if(command.equals("exportProject")){
                              this.exportProject(request, response);
                      } else if(command.equals("removeFile")){
                              response.getWriter().write(this.removeFile(request));
                      } else if(command.equals("updateAudioFiles")) {
                              response.getWriter().write(this.updateAudioFiles(request, response));
                      } else if(command.equals("special")){
                              this.processSpecial(request, response);
                      } else if(command.equals("specialToo")){
                              this.specialToo(request, response);
                      } else {
                              throw new ServletException("This servlet does not understand this command: " + command);
                      }
              } else if(ServletFileUpload.isMultipartContent(request)){
                      response.setContentType("text/html; charset=UTF-8");
                      try{
                              this.importProject(request);
                              response.getWriter().print("success");
                      } catch(Exception e){
                              e.printStackTrace();
                              response.getWriter().write("failed");
                      }
              } else {
```

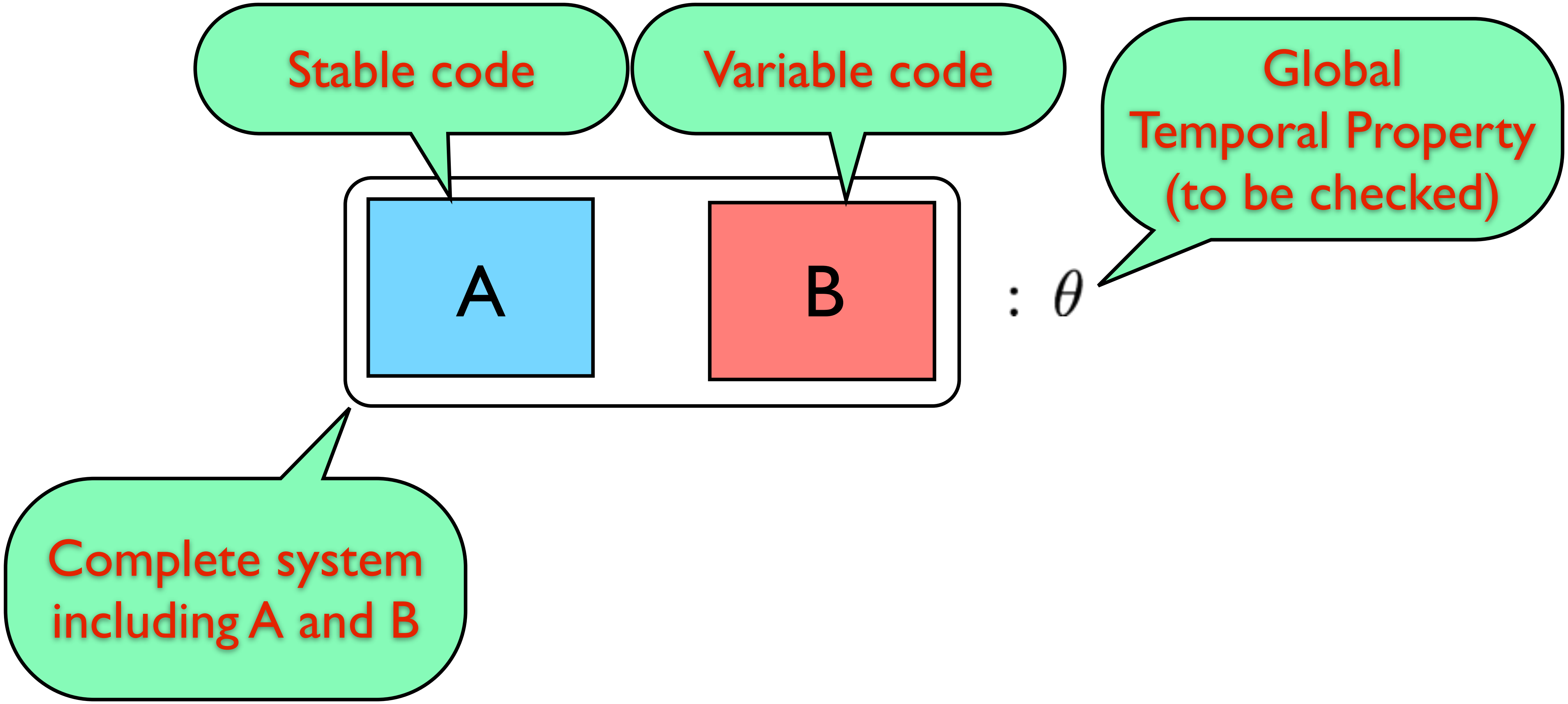
$: \theta$

Variability



$: \ \theta$

Verifying Variable Systems


Stable code
Variable code
Global Temporal Property (to be checked)
A
B
$: \theta$
Complete system including A and B

Running Example: Pointer Language

```
decl x = null;
decl y = null;

Main() {
   while(*) {
        new x;
        y = x;
        Foo();
        delete x;
    }
}

Foo() {
  .....
}
```

Running Example: Pointer Language

```
decl x = null;
decl y = null;

Main() {
   while(*) {
      new x;
      y = x;
      Foo();
      delete x;
   }
}

Foo() {
  .....
}
```

Running Example: Global Prop.

```
decl x = null;
decl y = null;

Main() {
  while(*) {
    new x;
    y = x;
    Foo();
    delete x;
  }
}
```

```
Foo() {
  .....
}
```

Running Example: Global Prop.

```
decl x = null;
decl y = null;

Main() {
  while(*) {
    new x;
    y = x;
    Foo();
    delete x;
  }
}
```

```
Foo() {
  .....
}
```

Global Property

"always a **delete** between two **new**"

Running Example: Global Prop.

```
decl x = null;
decl y = null;

Main() {
  while(*) {
    new x;
    y = x;
    Foo();
    delete x;
  }
}
```

```
Foo() {
  new x;
}
```

Global Property

“always a **delete** between two **new**”

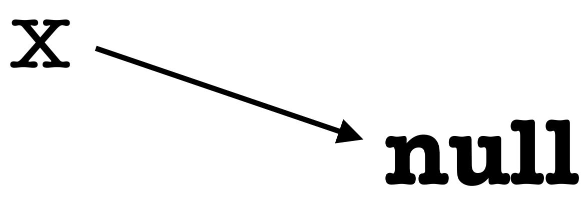
Running Example: Global Prop.

```
decl x = null;
decl y = null;

Main() {
   while(*) {
     new x;
     y = x;
     Foo();
     delete x;
   }
}
```

```
Foo() {
  new x;
}
```

Global Property

“always a **delete** between two **new**”



x

null

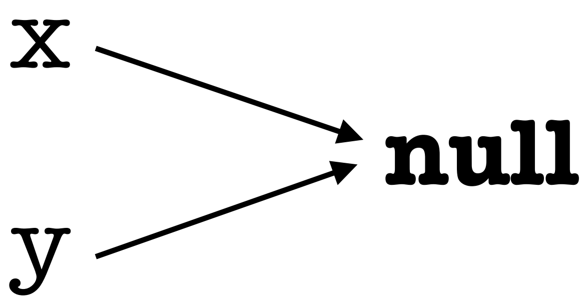
Running Example: Global Prop.

```
decl x = null;
decl y = null;

Main() {
  while(*) {
    new x;
    y = x;
    Foo();
    delete x;
  }
}
```

```
Foo() {
  new x;
}
```

Global Property

“always a **delete** between two **new**”



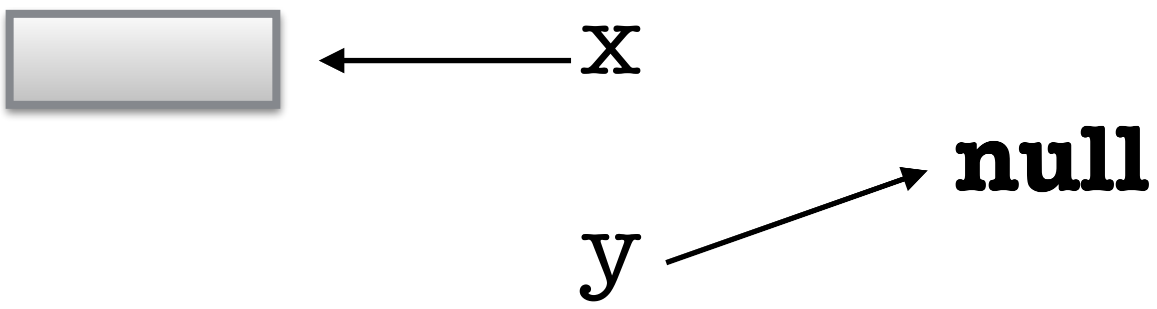
Running Example: Global Prop.

```
decl x = null;
decl y = null;

Main() {
  while(*) {
    new x;
    y = x;
    Foo();
    delete x;
  }
}
```

```
Foo() {
  new x;
}
```

Global Property

“always a **delete** between two **new**”



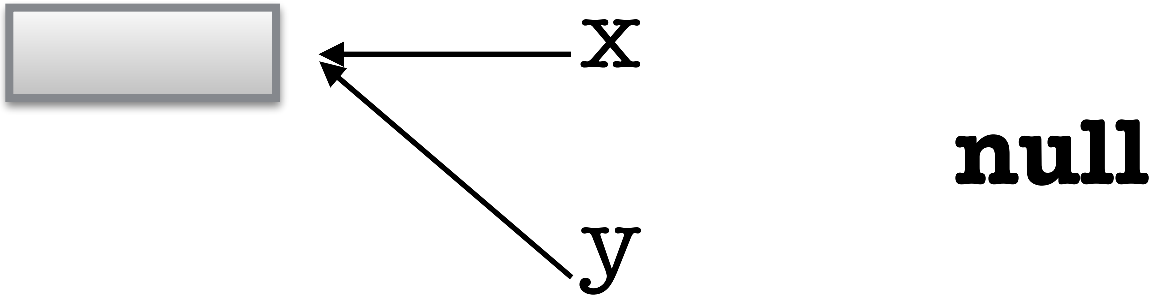
Running Example: Global Prop.

```
decl x = null;
decl y = null;

Main() {
  while(*) {
    new x;
    y = x;
    Foo();
    delete x;
  }
}
```

```
Foo() {
  new x;
}
```

Global Property

“always a **delete** between two **new**”



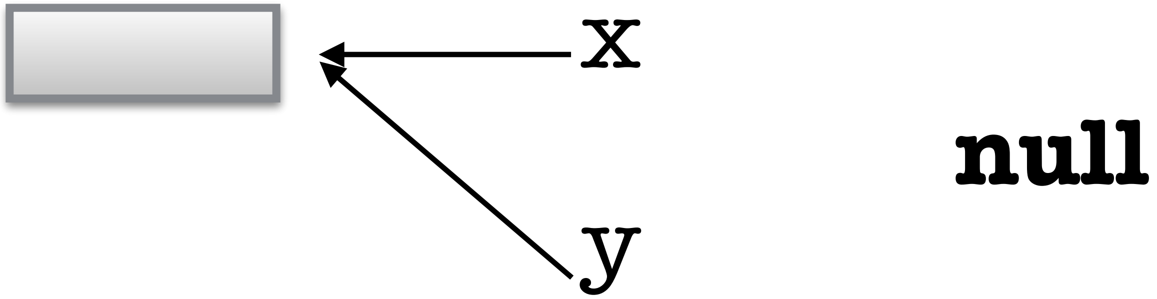
Running Example: Global Prop.

```
decl x = null;
decl y = null;

Main() {
   while(*) {
     new x;
     y = x;
     Foo();
     delete x;
   }
}
```

```
Foo() {
  new x;
}
```

Global Property

“always a **delete** between two **new**”



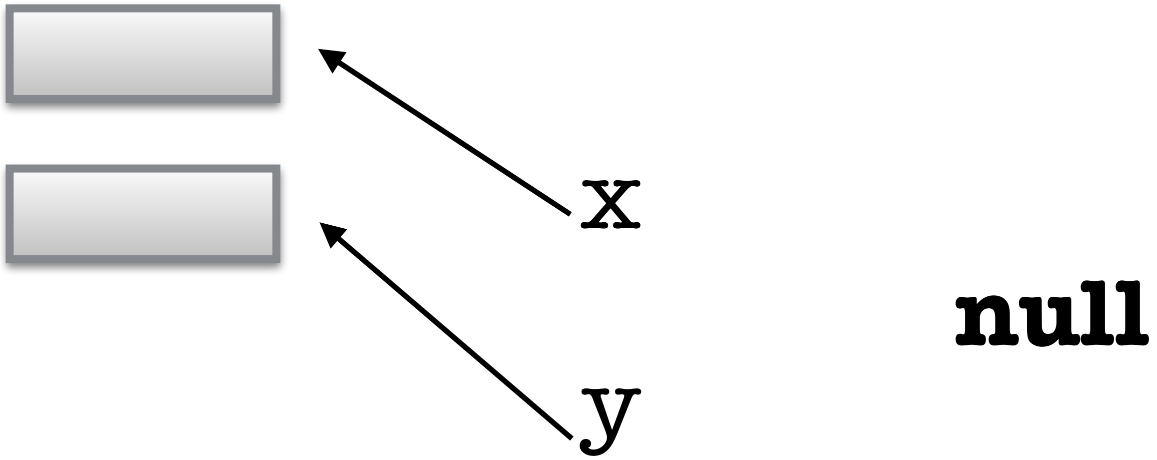
Running Example: Global Prop.

```
decl x = null;
decl y = null;

Main() {
  while(*) {
    new x;
    y = x;
    Foo();
    delete x;
  }
}
```

```
Foo() {
  new x;
}
```

Global Property

"always a **delete** between two **new**"



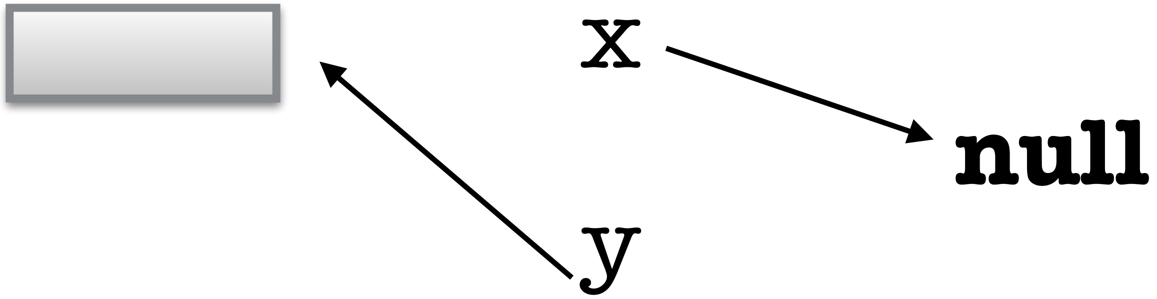
Running Example: Global Prop.

```
decl x = null;
decl y = null;

Main() {
  while(*) {
    new x;
    y = x;
    Foo();
    delete x;
  }
}
```

```
Foo() {
  new x;
}
```

Global Property

"always a **delete** between two **new**"



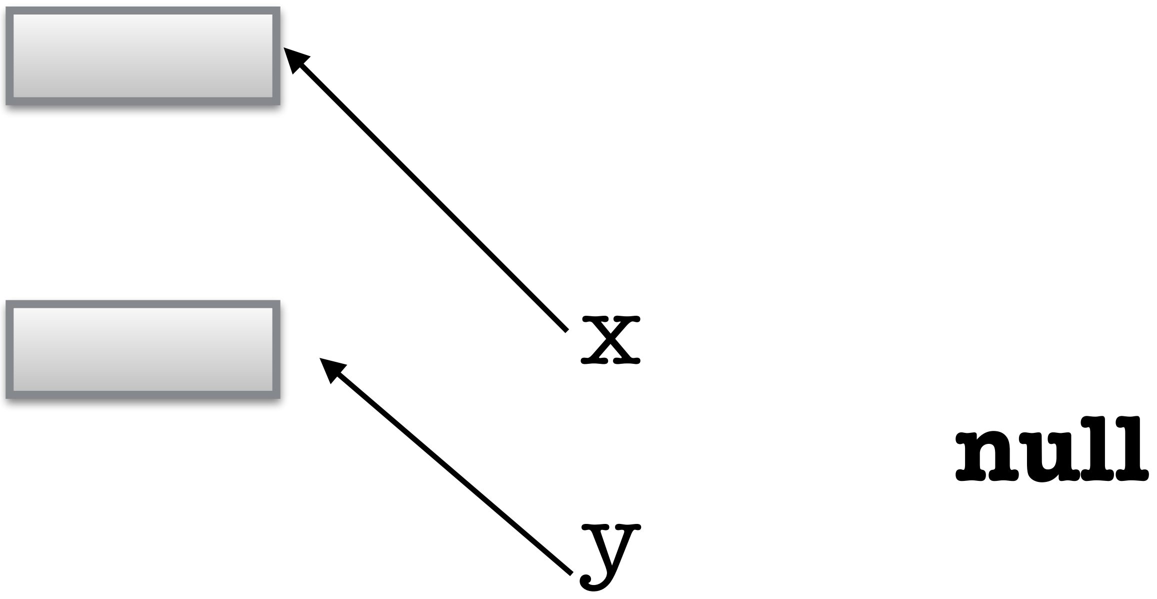
Running Example: Global Prop.

```
decl x = null;
decl y = null;

Main() {
  while(*) {
    new x;
    y = x;
    Foo();
    delete x;
  }
}
```

```
Foo() {
  new x;
}
```

Global Property

“always a **delete** between two **new**”



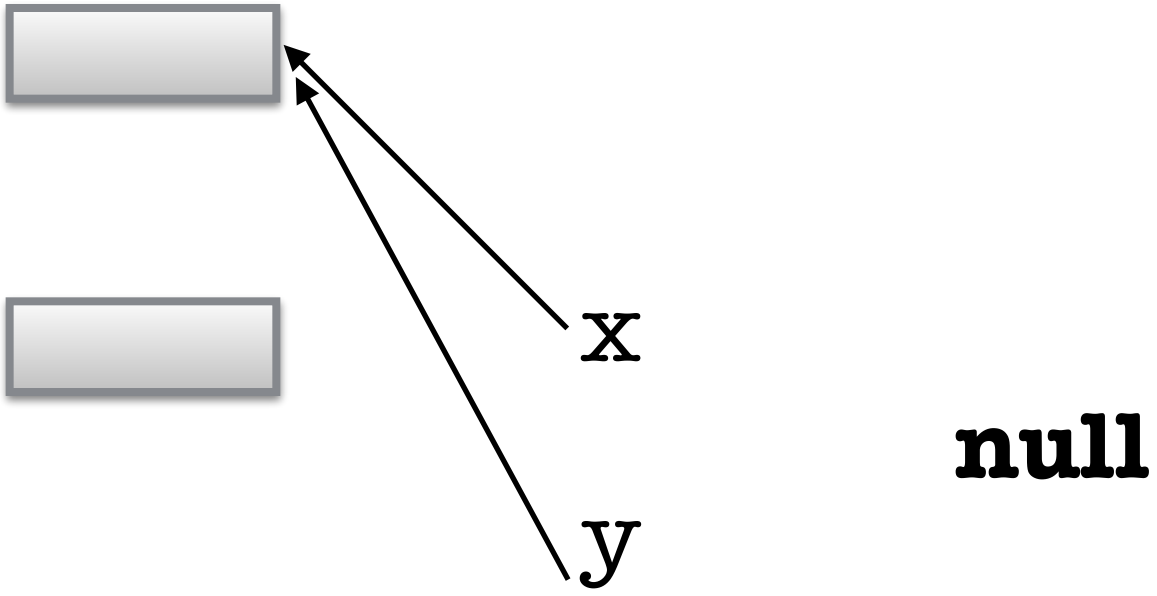
Running Example: Global Prop.

```
decl x = null;
decl y = null;

Main() {
  while(*) {
    new x;
    y = x;
    Foo();
    delete x;
  }
}
```

```
Foo() {
  new x;
}
```

Global Property

"always a **delete** between two **new**"



Running Example: Specification

```
decl x = null;
decl y = null;

Main() {
   while(*) {
     new x;
     y = x;
     Foo();
     delete x;
   }
}
```

```
Foo() {
   .....
}
```

Global Property

"always a **delete** between two **new**"

Running Example: Specification

```
decl x = null;
decl y = null;

Main() {
  while(*) {
    new x;
    y = x;
    Foo();
    delete x;
  }
}
```

```
Foo() {
  .....
}
```

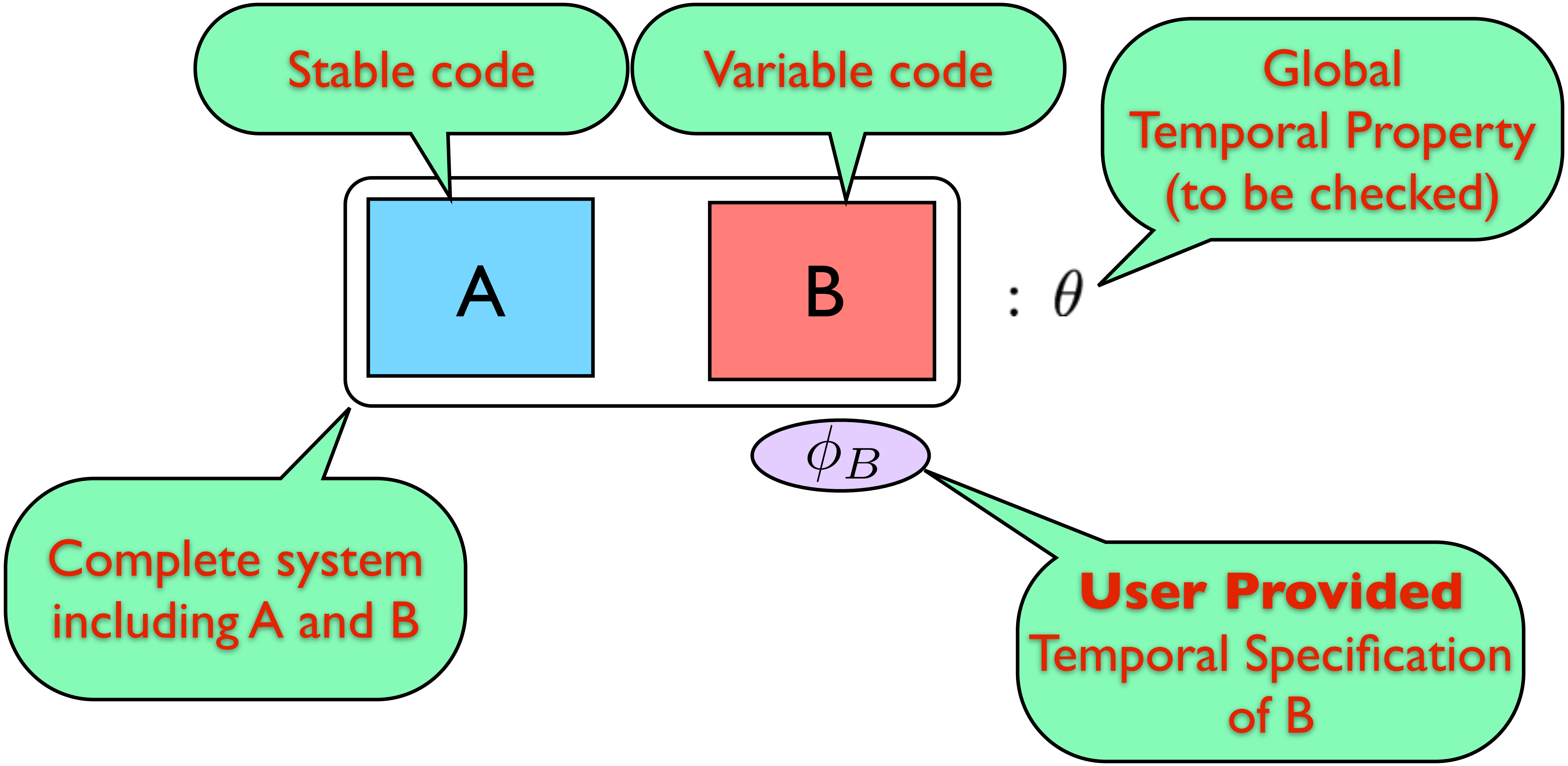
Global Property

"always a **delete** between two **new**"

Constraint on the Implementation of Foo

"no **new** statement"

Verification Setup


Stable code
Variable code
Global Temporal Property (to be checked)
A
B
$: \theta$
ϕ_B
Complete system including A and B
User Provided Temporal Specification of B

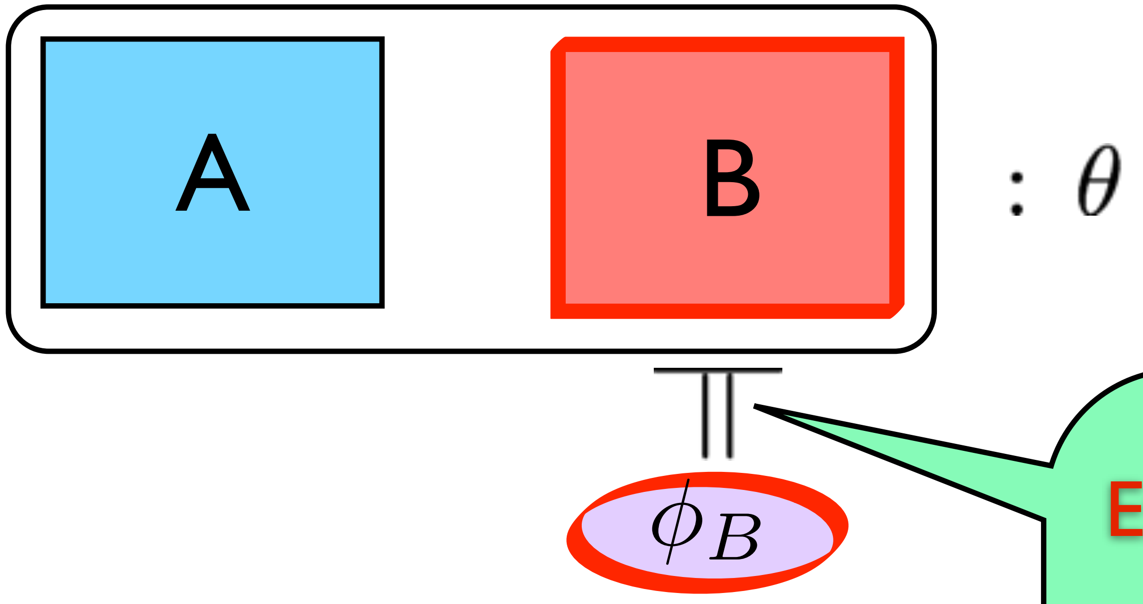
Modular Verification

Verification Subtasks

I. check that each variable component satisfies its local specification

II. check that the **composition** of the specification of variable components together with the implementation of the stable ones satisfies the global property

Task I: Local Check



Existing Verification Techniques, e.g., **Model Checking**

Modular Verification

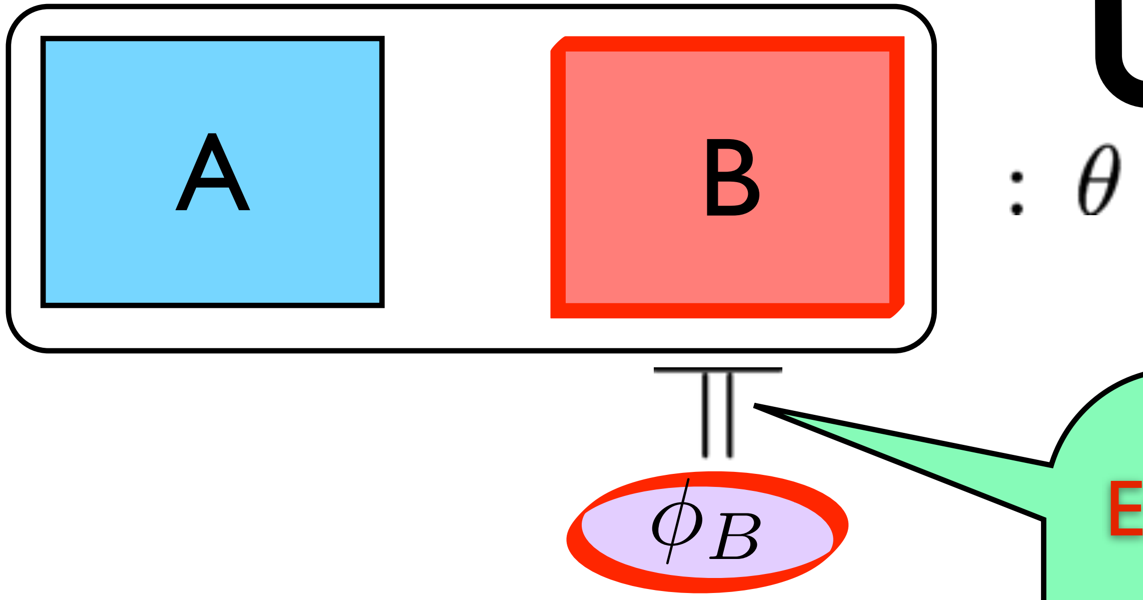
Verification Subtasks

I. check that each variable component satisfies its local specification

II. check that the **composition** of the specification of variable components together with the implementation of the stable ones satisfies the global property

Task I: Local Check

Customer Side



Existing Verification Techniques, e.g., **Model Checking**

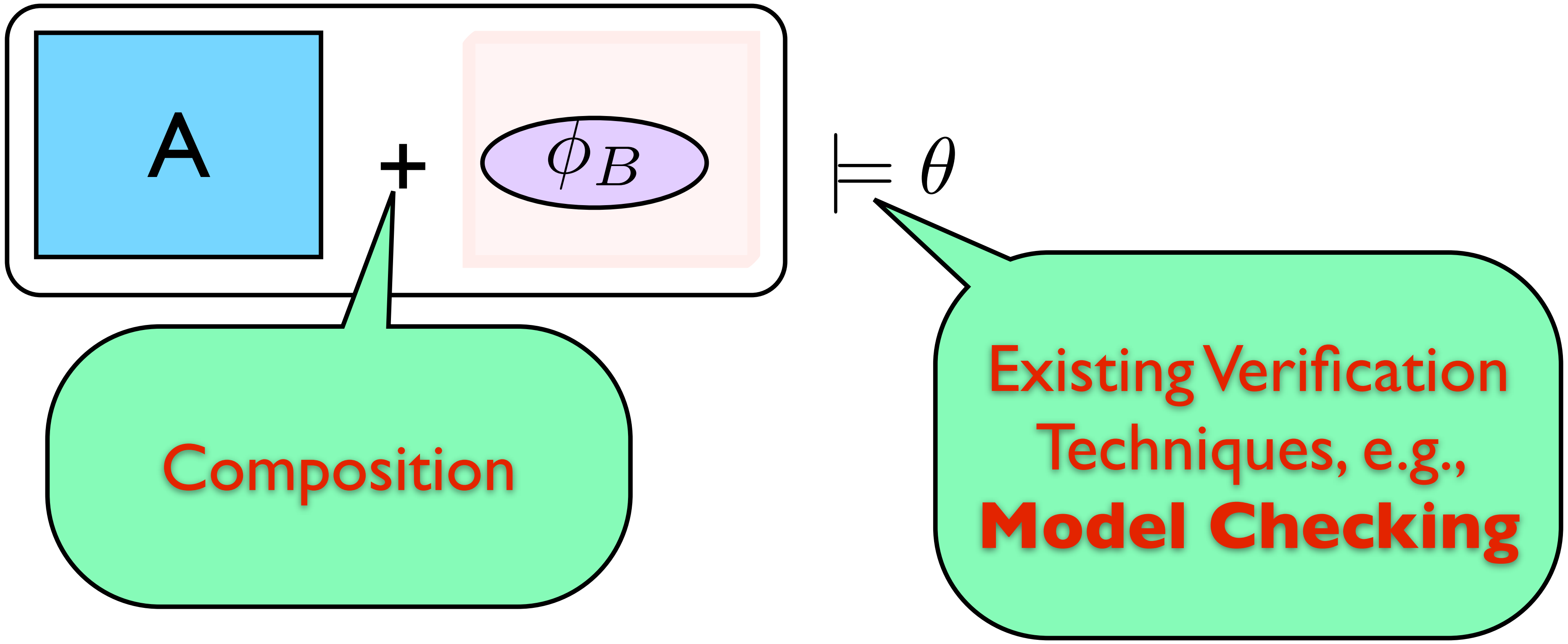
Modular Verification

Verification Subtasks

I. check that each **variable component** satisfies **its local specification**

II. check that the **composition** of the specification of variable components together with the implementation of the stable ones satisfies the global property

Task II: Global Check



Modular Verification

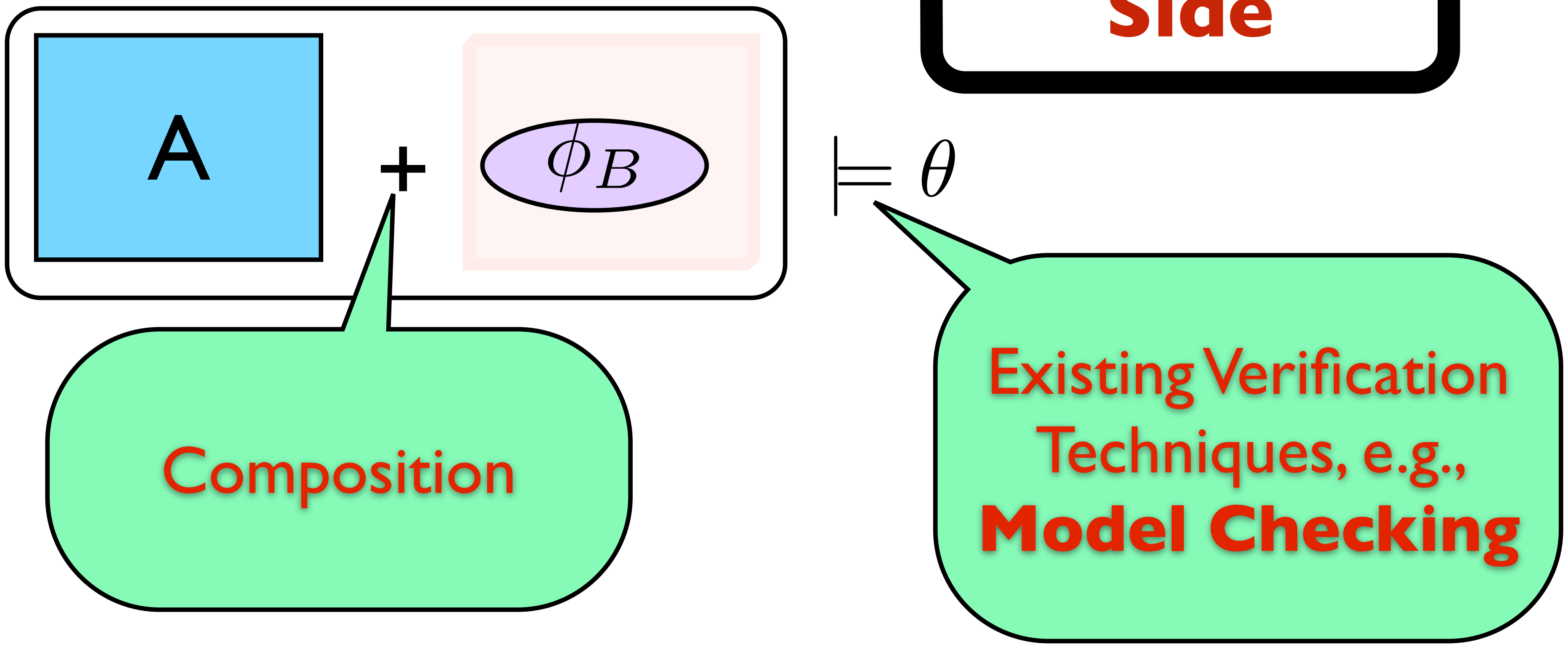
Verification Subtasks

I. check that each **variable component** satisfies **its local specification**

II. check that the **composition** of the specification of variable components together with the implementation of the stable ones satisfies the global property

Task II: Global Check

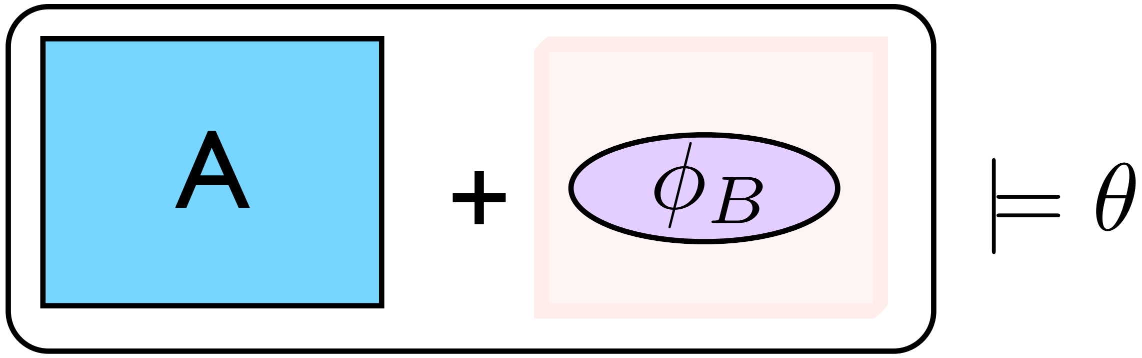
Developer Side



Composition

Existing Verification Techniques, e.g., **Model Checking**

Modular Verification



A + ϕ_B $\models \theta$

Existing Approaches

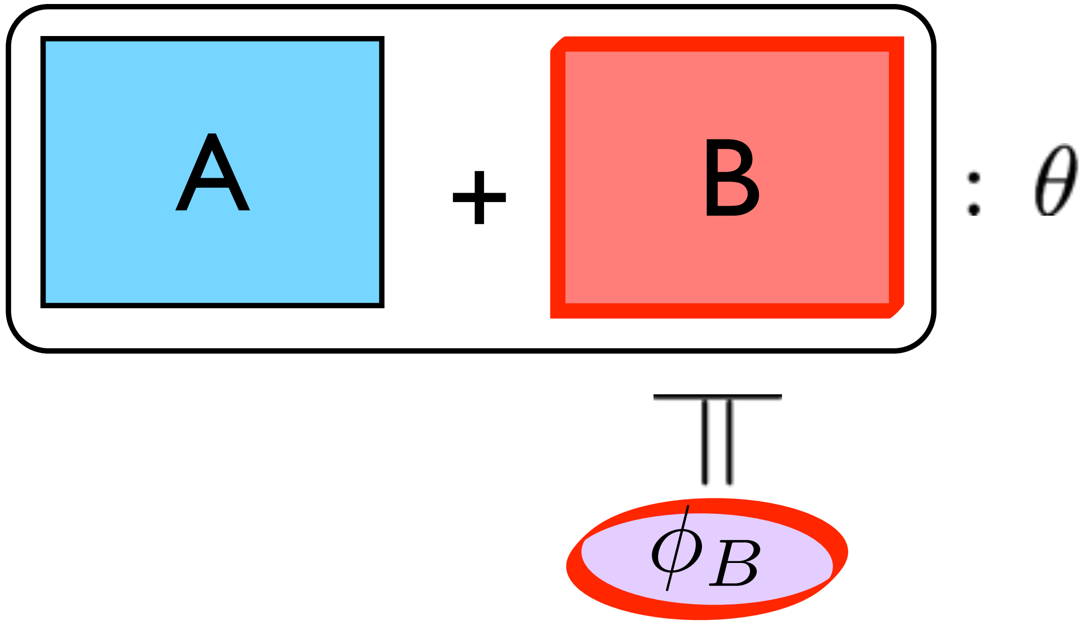
- Modular verification of procedural programs: "built-in" for **Hoare-logic** based approaches

Not for Temporal Properties

- Modular model checking: based on **maximal model construction**
  Grumberg & Long 1994: ACTL
  Kupferman & Vardi 2000: ACTL*

Finite Systems (not procedural programs)

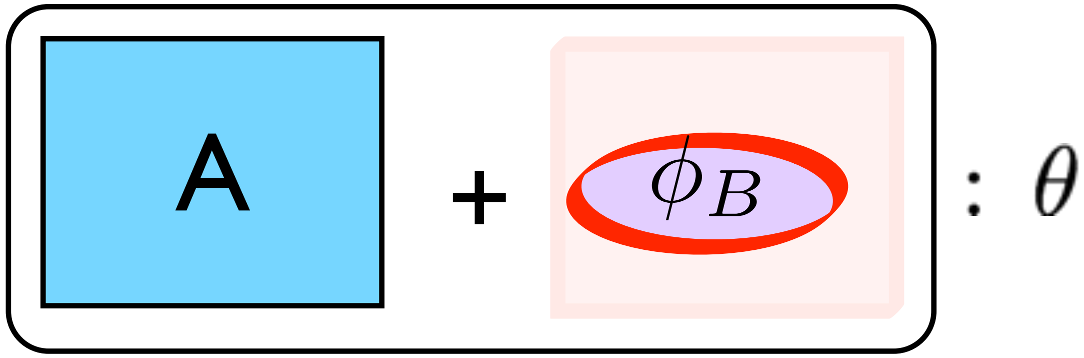
Verification based on Maximal Models



Task I

1. Model check the code of the variable components against their local specifications

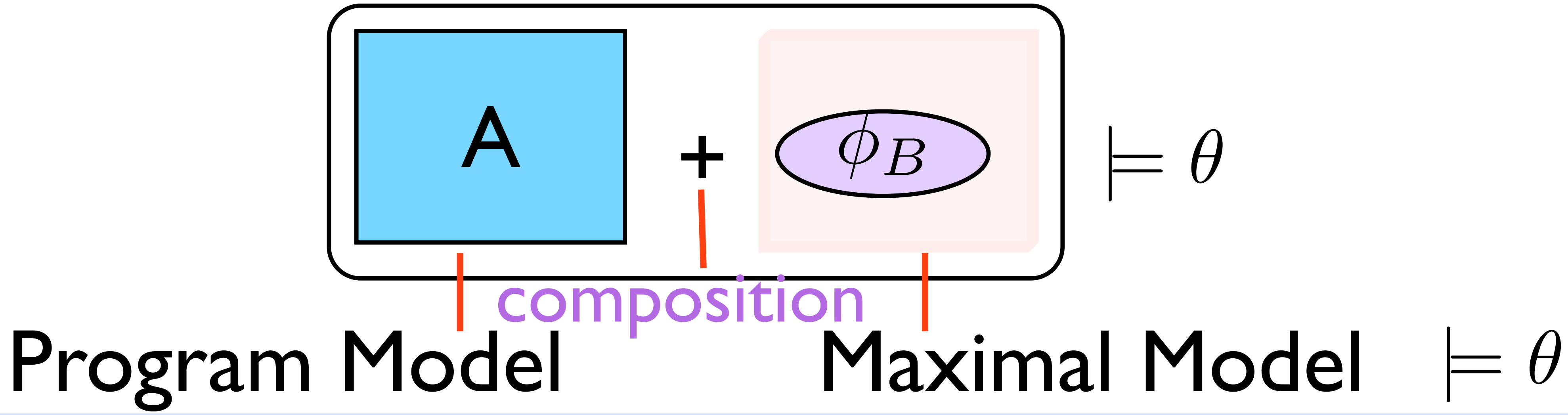
Verification based on Maximal Models



Task I

1. Model check the code of the variable components against their local specifications

Verification based on Maximal Models



A + ϕ_B $\models \theta$

composition

Program Model

Maximal Model $\models \theta$

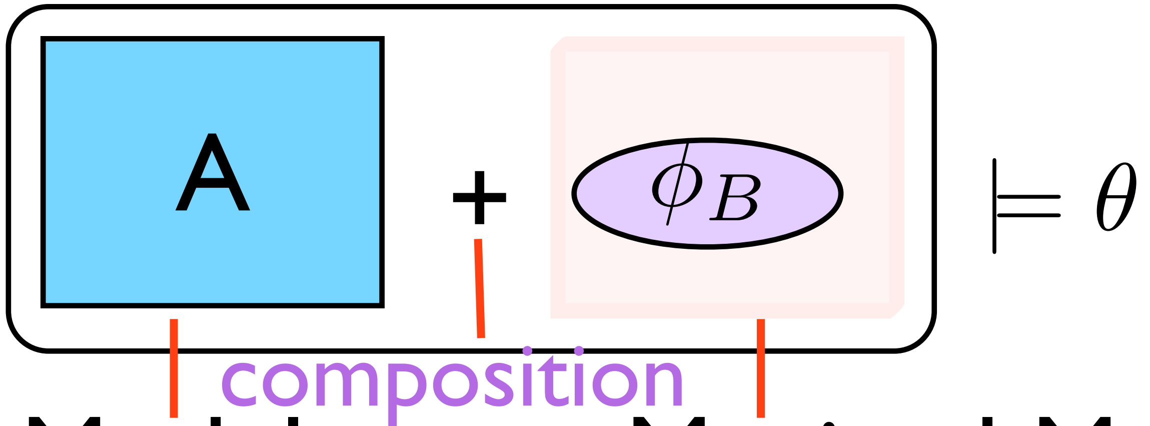
Task I

1. Model check the code of the variable components against their local specifications

Task II

1. Model extraction from stable code

Verification based on Maximal Models



Program Model Maximal Model $\models \theta$

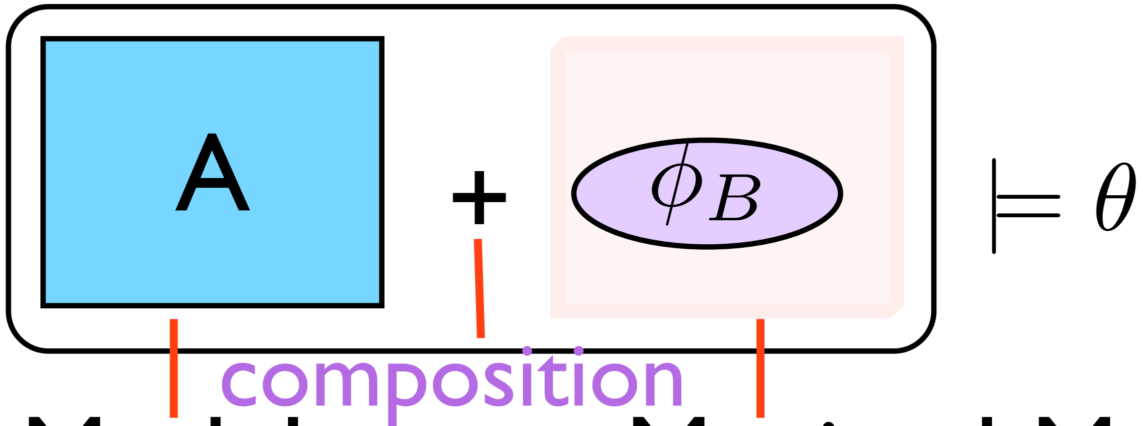
Task I

1. Model check the code of the variable components against their local specifications

Task II

1. Model extraction from stable code
2. Maximal model construction from local specification

Verification based on Maximal Models



composition

Program Model Maximal Model $\models \theta$

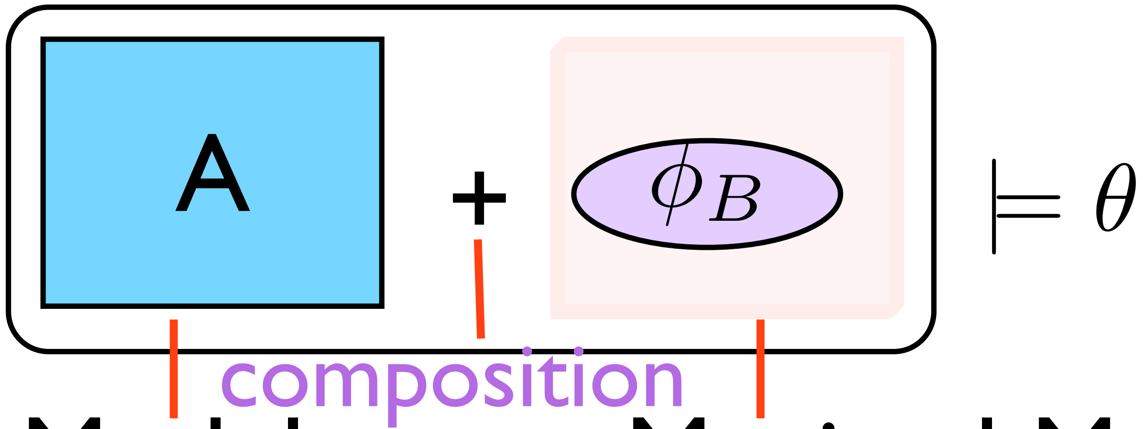
Task I

1. Model check the code of the variable components against their local specifications

Task II

1. Model extraction from stable code
2. Maximal model construction from local specification

Verification based on Maximal Models



A

+

ϕ_B

$\models \theta$

composition

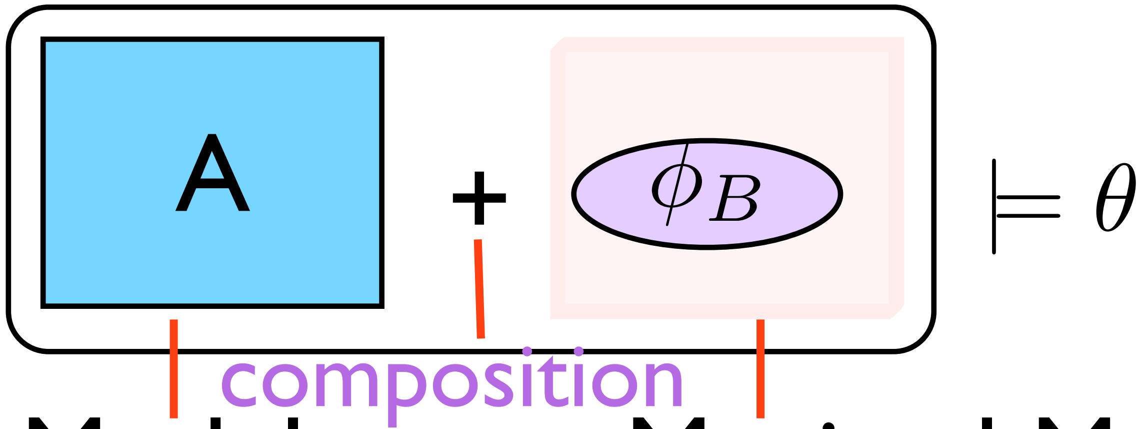
Program Model

Maximal Model $\models \theta$

Maximal Models

- A maximal model for ϕ_B is the most general model that satisfies it
  - represents all models that satisfy ϕ_B
  - models represent code, thus a maximal model for ϕ_B represents any code that its model satisfies ϕ_B

Verification based on Maximal Models



A + ϕ_B $\models \theta$

composition

Program Model

Maximal Model $\models \theta$

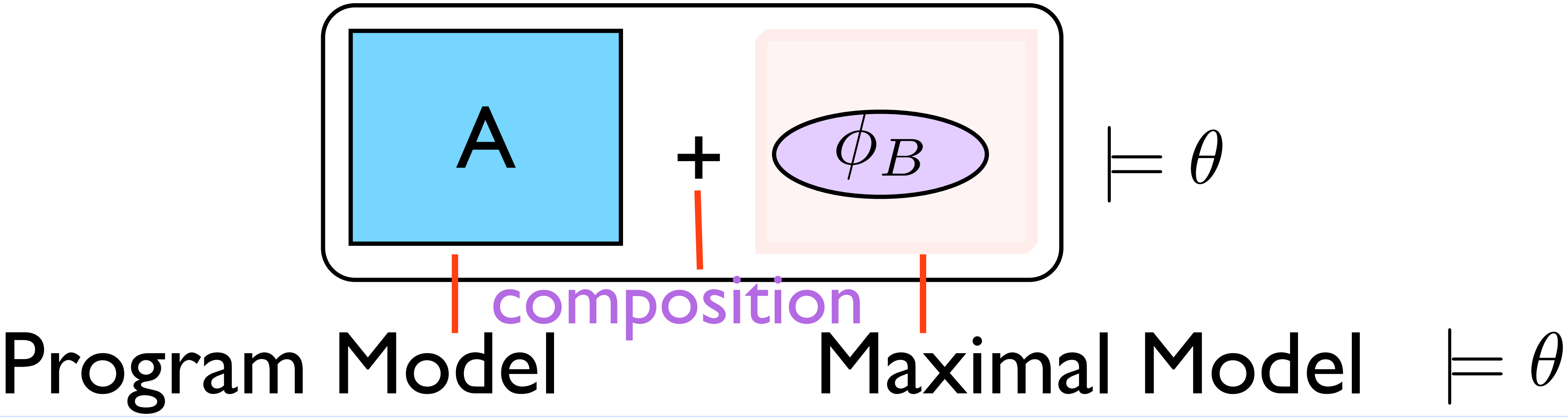
Task I

1. Model check the code of the variable components against their local specifications

Task II

1. Model extraction from stable code
2. Maximal model construction from local specification

Verification based on Maximal Models



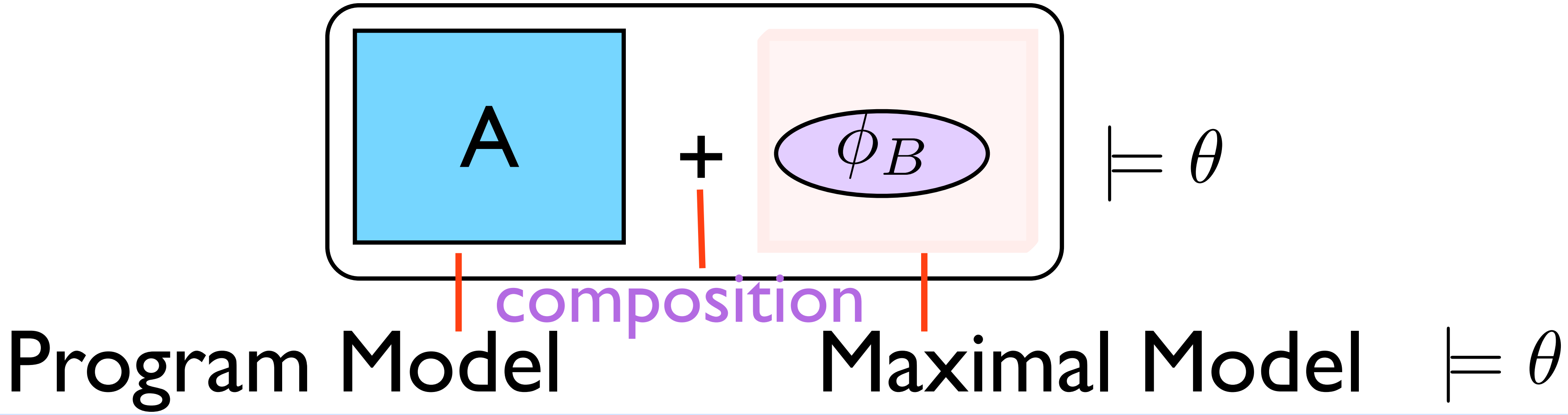
Task I

1. Model check the code of the variable components against their local specifications

Task II

1. Model extraction from stable code
2. Maximal model construction from local specification

Verification based on Maximal Models



A + ϕ_B $\models \theta$

composition

Program Model

Maximal Model $\models \theta$

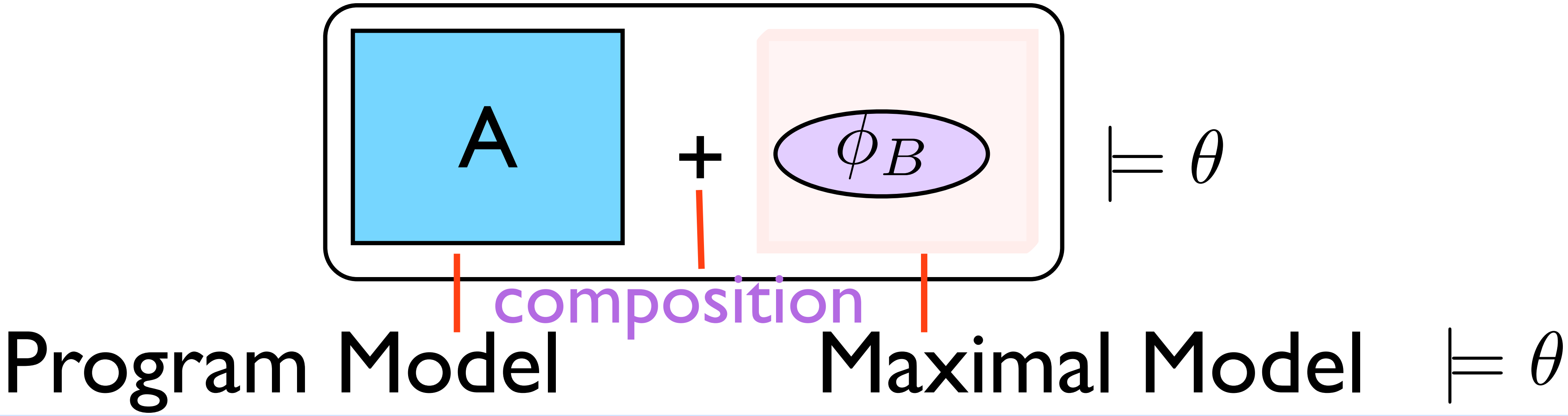
Task I

1. Model check the code of the variable components against their local specifications

Task II

1. Model extraction from stable code
2. Maximal model construction from local specification
3. Compose extracted and constructed models

Verification based on Maximal Models



A + ϕ_B $\models \theta$

composition

Program Model

Maximal Model $\models \theta$

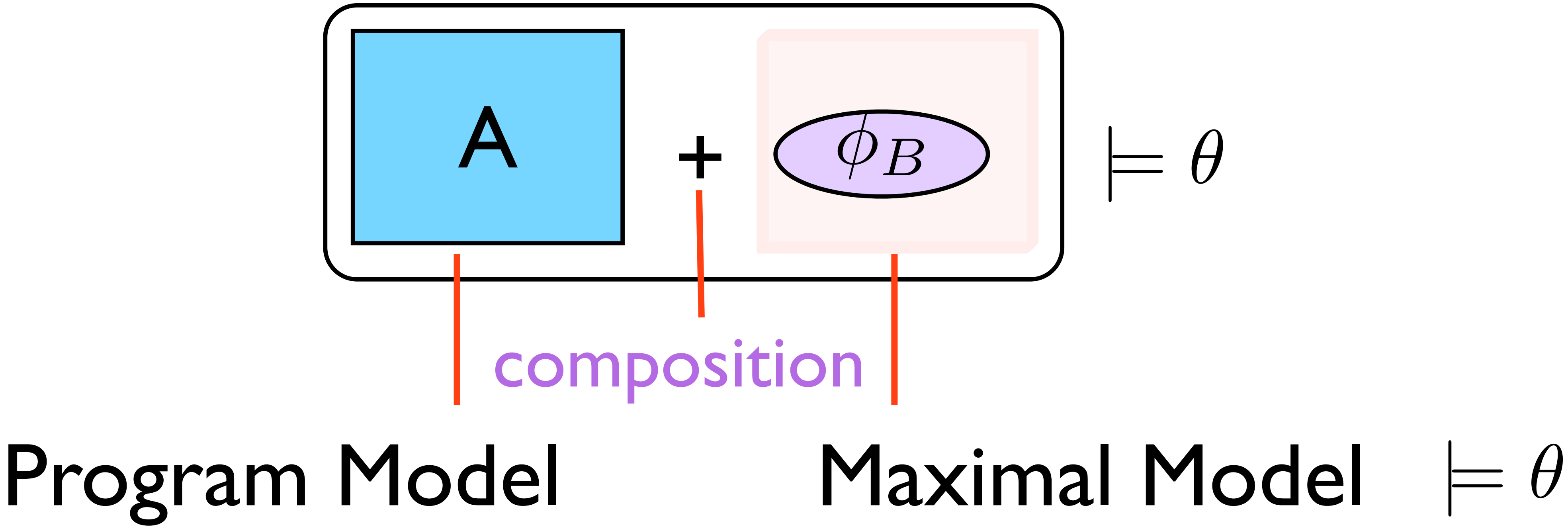
Task I

1. Model check the code of the variable components against their local specifications

Task II

1. Model extraction from stable code
2. Maximal model construction from local specification
3. Compose extracted and constructed models
4. Model check the resulting model

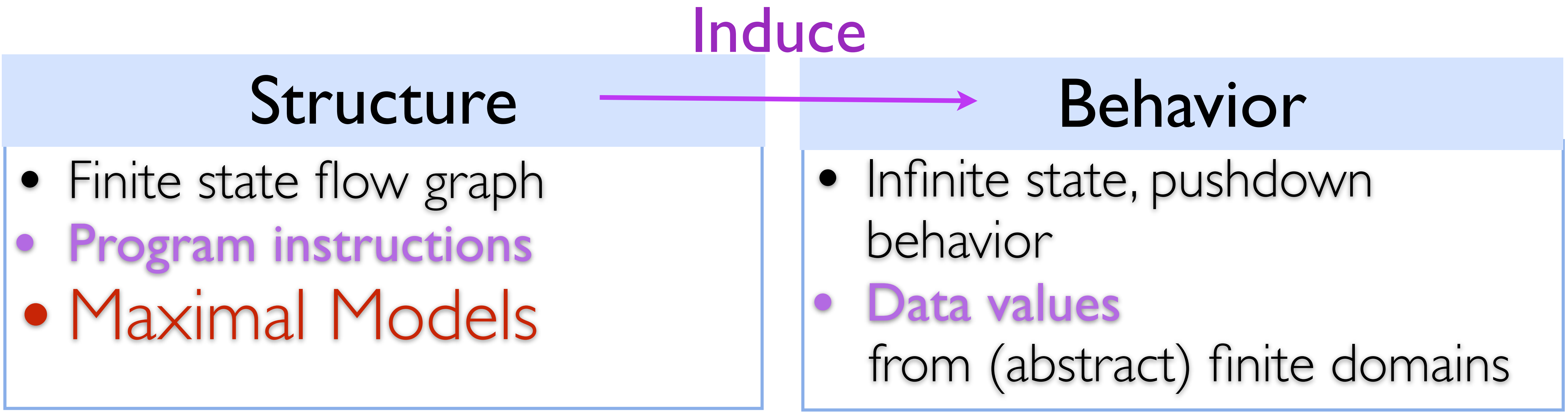
Framework



Challenges

- Algorithmic solution (procedural programs, i.e., pushdown systems)
- Practical
  - Task I (user side task) efficient and with minimal manual effort
  - complexity of the maximal model construction
  - difficulty and complexity of specifying specifications and models
  - Independent from programming languages

Framework



Induce

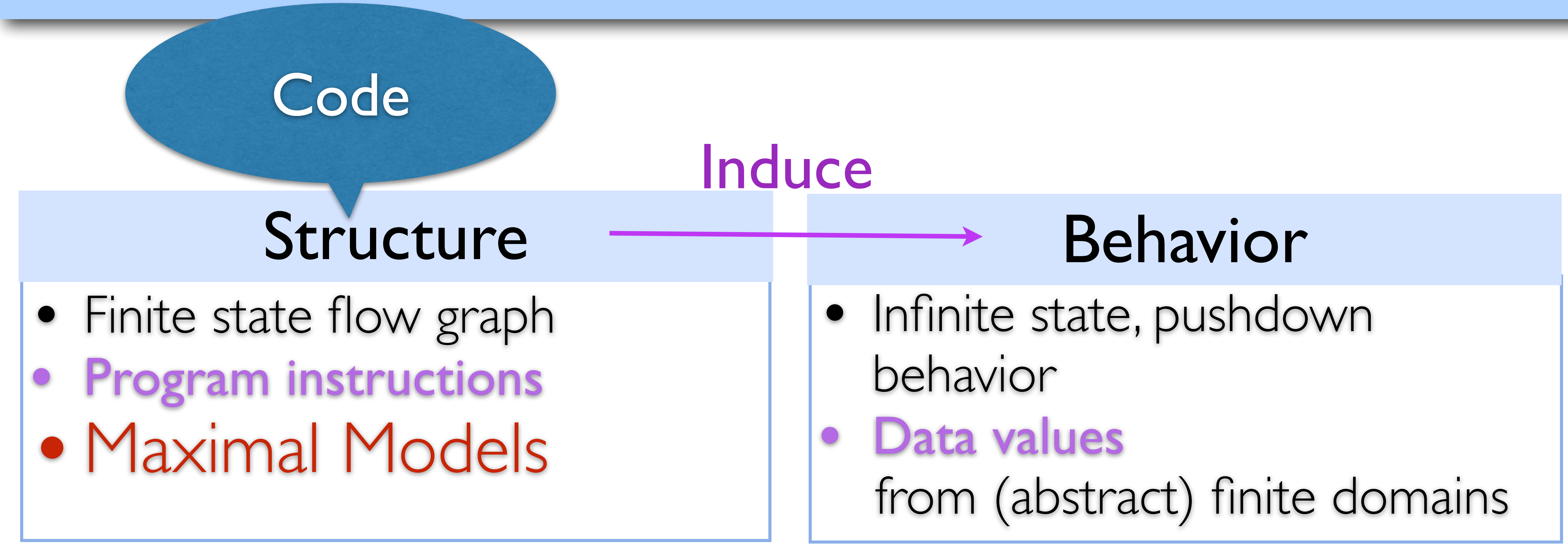
Structure

- Finite state flow graph
- Program instructions
- Maximal Models

Behavior

- Infinite state, pushdown behavior
- Data values
  from (abstract) finite domains

Framework



Code

Induce

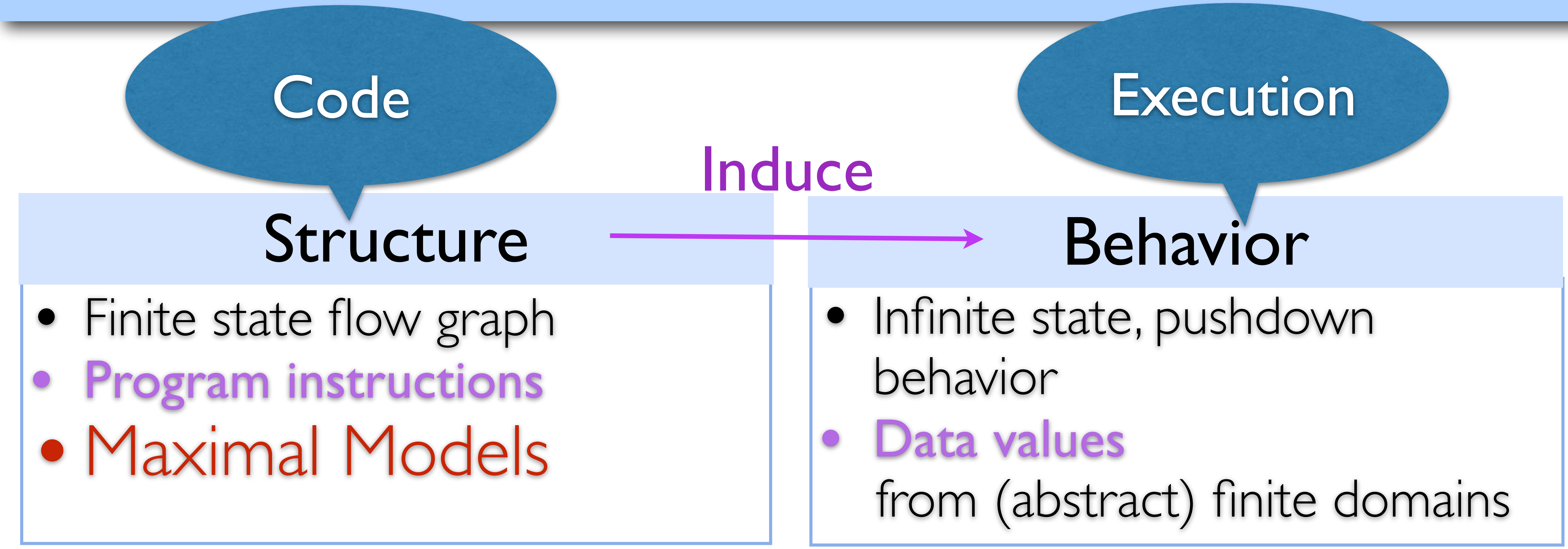
Structure

- Finite state flow graph
- Program instructions
- Maximal Models

Behavior

- Infinite state, pushdown behavior
- Data values from (abstract) finite domains

Framework



Code

Execution

Induce

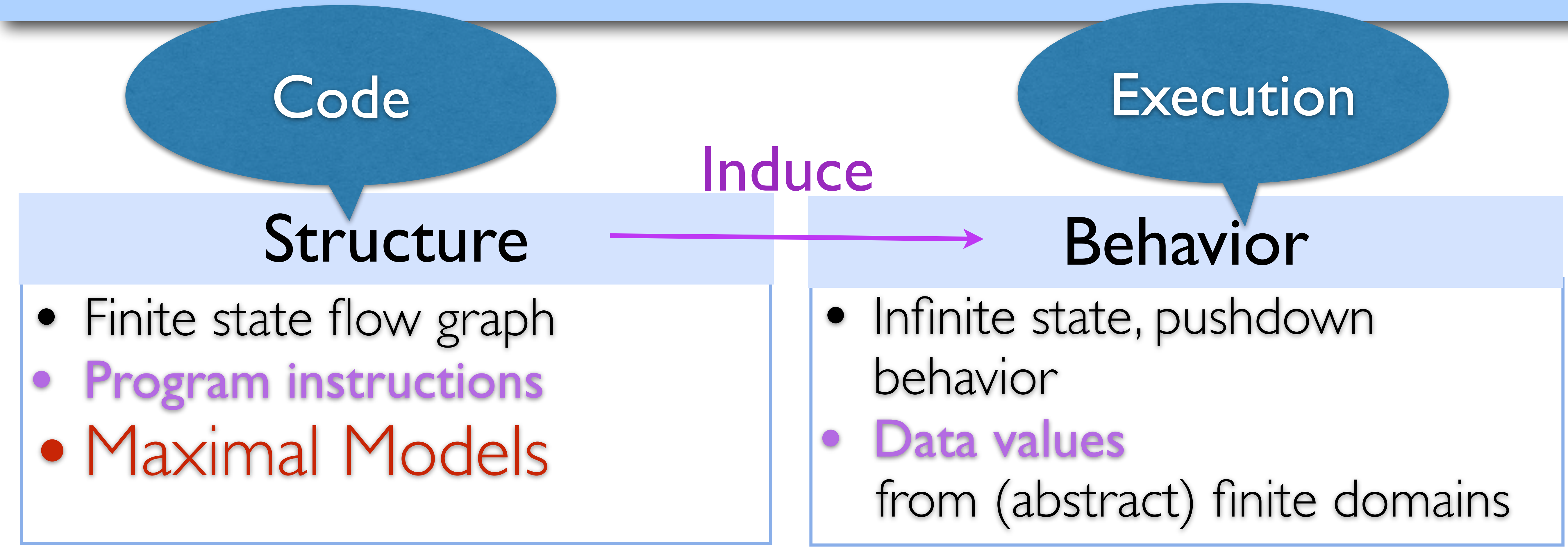
Structure

- Finite state flow graph
- Program instructions
- Maximal Models

Behavior

- Infinite state, pushdown behavior
- Data values from (abstract) finite domains

Framework



Code

Execution

Induce

Structure

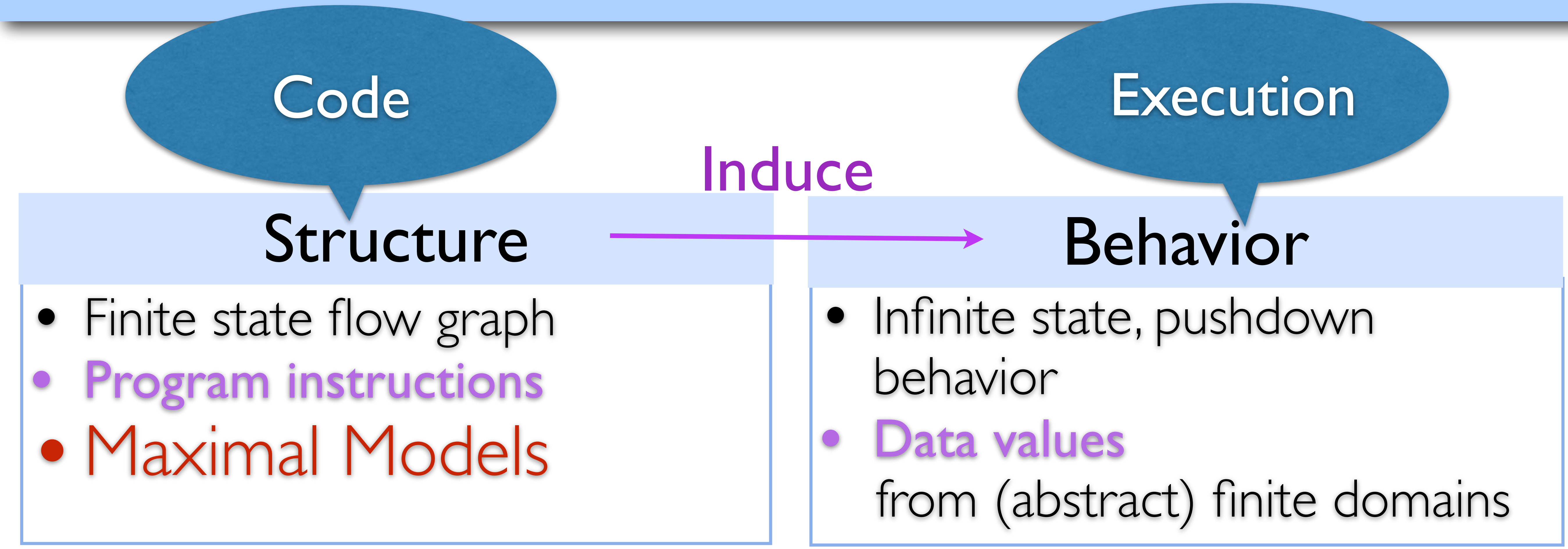
- Finite state flow graph
- Program instructions
- Maximal Models

Behavior

- Infinite state, pushdown behavior
- Data values
  from (abstract) finite domains

Practical Specification

Framework



Code

Execution

Induce

Structure

- Finite state flow graph
- Program instructions
- Maximal Models

Behavior

- Infinite state, pushdown behavior
- Data values from (abstract) finite domains

Practical Specification

Completeness

Overview of the Approach

User Tasks

1. Specification: Local specification (variable) & Global property
2. Define observed instructions

Task I

1. Model check the code of the variable components against their local specifications

Task II

1. Model extraction from stable code
2. Maximal model construction from local specification
3. Compose models and induce the behavior of the system
4. Model check the behavior

Specification

```
decl x = null;
decl y = null;

Main() {
  while(*) {
    new x;
    y = x;
    Foo();
    delete x;
  }
}
```

```
Foo() {
  .....
}
```

Global Behavioral Property

“always a **delete** between two **new**”

Local Structural Specification of Foo

“no **new** statement”

Specification

```
decl x = null;
decl y = null;

Main() {
  while(*) {
    new x;
    y = x;
    Foo();
    delete x;
  }
}
```

```
Foo() {
  .....
}
```

Global Behavioral Property

"always a **delete** between two **new**"

Local Structural Specification of Foo

"no **new** statement"

Observed Instructions

- Observed instructions: **new**, **delete**
- Capture the effect of other instructions through logical conditions of the form $v = v' \quad v \neq v'$

Specification

```
decl x = null;
decl y = null;

Main() {
  while(*) {
    new x;
    y = x;
    Foo();
    delete x;
  }
}
```

```
Foo() {
  .....
}
```

Global Behavioral Property

"always a **delete** between two **new**"

Local Structural Specification of Foo

"no **new** statement"

Simulation Logic

$$\phi ::= p \mid \neg p \mid X \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid [a]\phi \mid \nu X.\, \phi$$

Specification

```
decl x = null;
decl y = null;

Main() {
  while(*) {
    new x;
    y = x;
    Foo();
    delete x;
  }
}
```

```
Foo() {
  .....
}
```

Global Behavioral Property

"always a **delete** between two **new**"

Local Structural Specification of Foo

"no **new** statement"

Simulation Logic

$$\phi ::= p \mid \neg p \mid X \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid [a]\phi \mid \nu X.\, \phi$$

entry, return, method names

$v = v' \quad v \neq v'$

Overview of the Approach

User Tasks

1. Specification: Local specification (variable) & Global property
2. Define observed instructions

Task I

1. Model check the code of the variable components against their local specifications

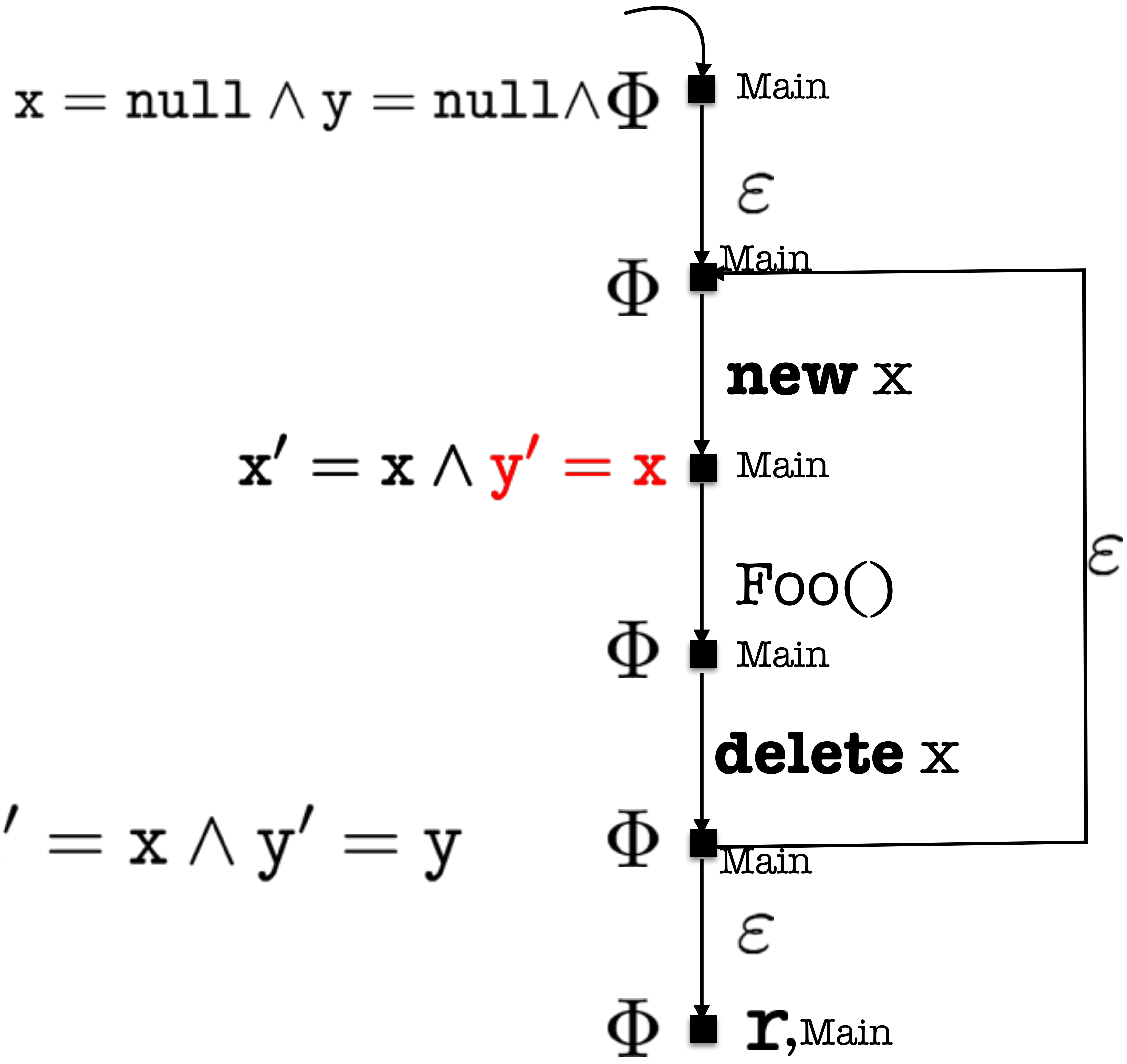
Task II

1. Model extraction from stable code
2. Maximal model construction from local specification
3. Compose models and induce the behavior of the system
4. Model check the behavior

Structural Models

```
decl x = null;
decl y = null;

Main() {
   while(*) {
     new x;
     y = x;
     Foo();
     delete x;
   }
}
```

```
Foo() {
   .....
}
```



Φ is $x' = x \wedge y' = y$

Structural Models

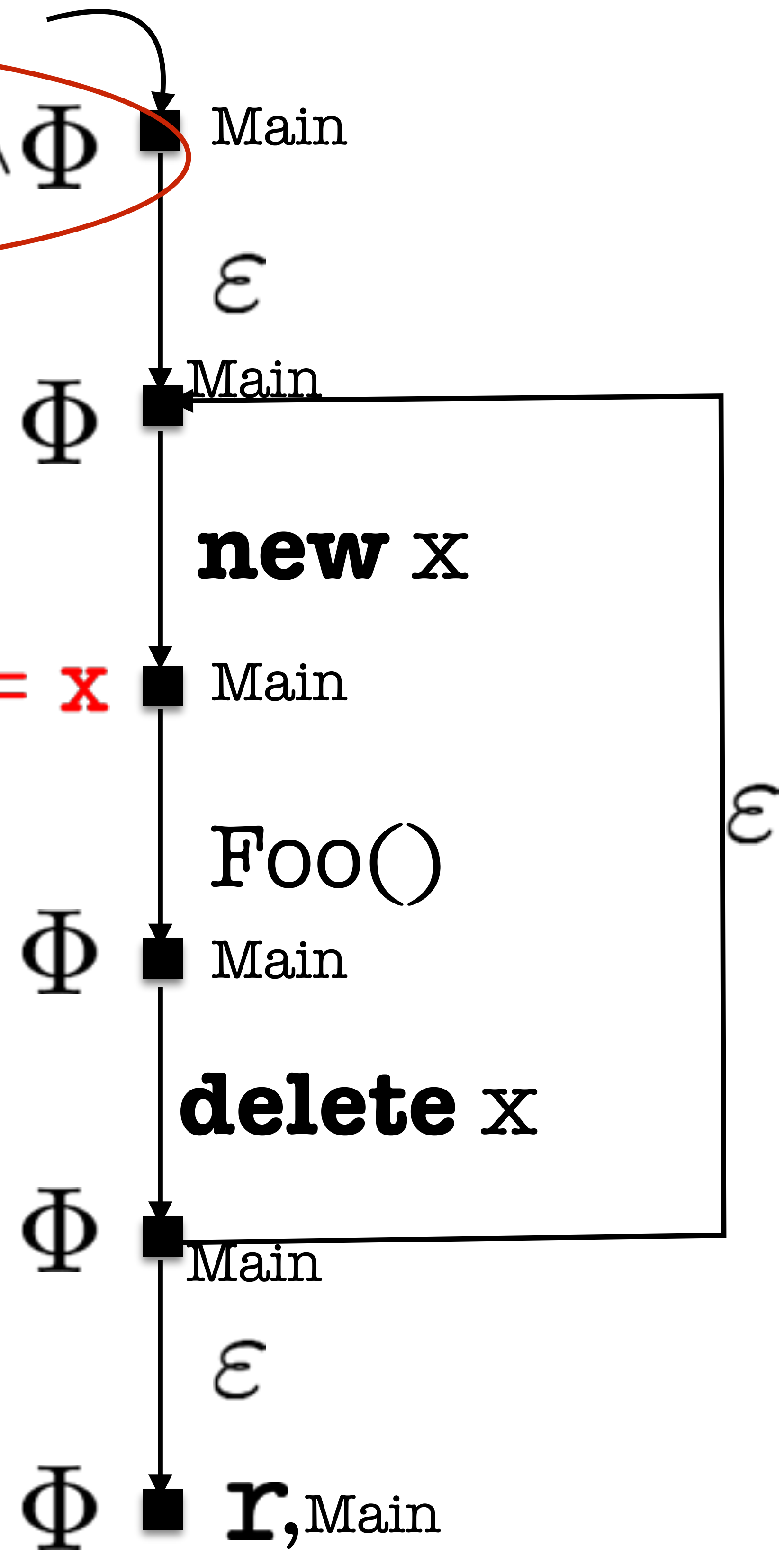
```
decl x = null;
decl y = null;

Main() {
   while(*) {
     new x;
     y = x;
     Foo();
     delete x;
   }
}
```

```
Foo() {
   .....
}
```

$\mathtt{x} = \mathtt{null} \wedge \mathtt{y} = \mathtt{null} \wedge \Phi$

$\mathtt{x}' = \mathtt{x} \wedge \mathtt{y}' = \mathtt{x}$

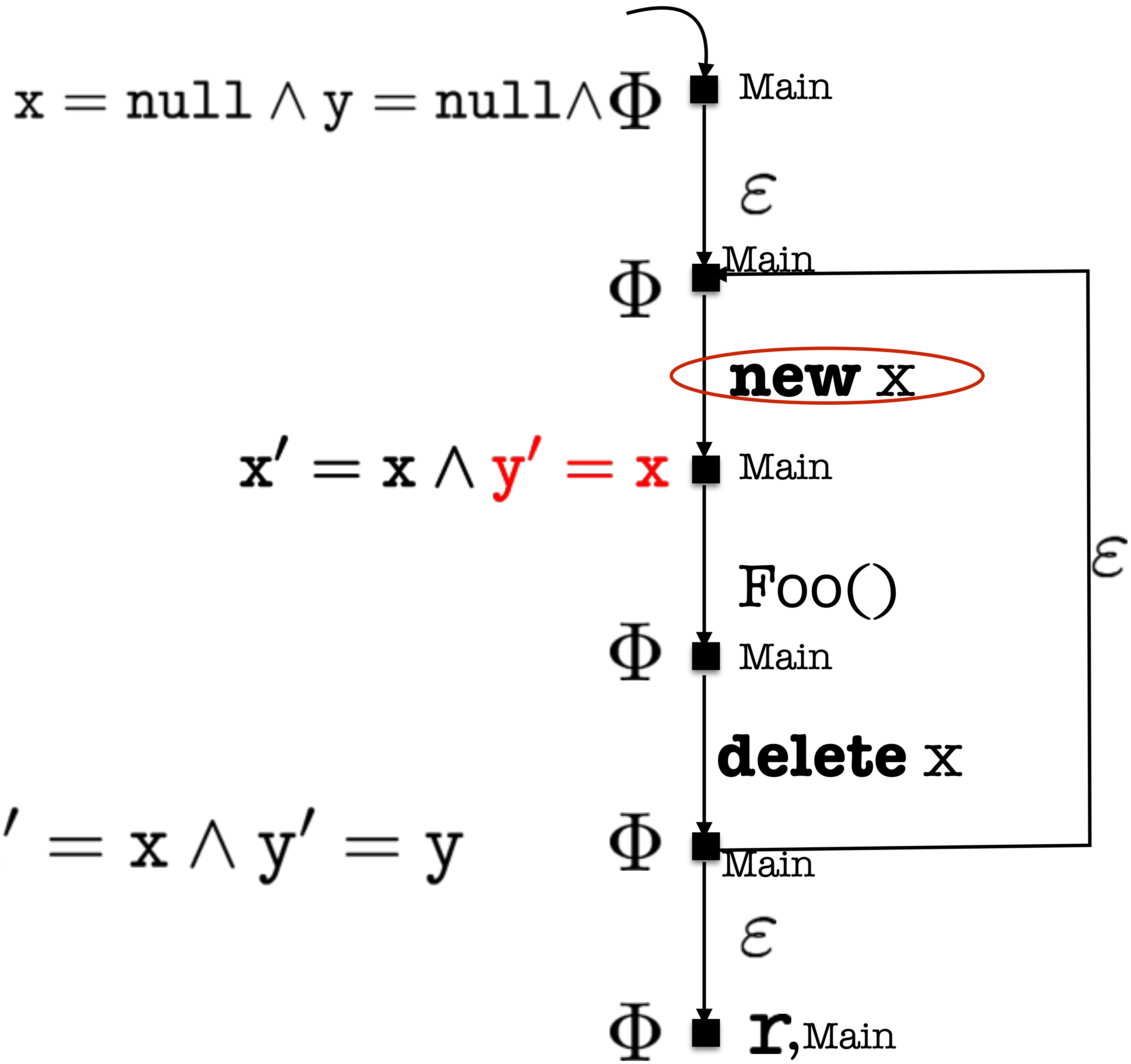
Φ is $\mathtt{x}' = \mathtt{x} \wedge \mathtt{y}' = \mathtt{y}$



Structural Models

```
decl x = null;
decl y = null;

Main() {
   while(*) {
     new x;
     y = x;
     Foo();
     delete x;
   }
}
```

```
Foo() {
   .....
}
```

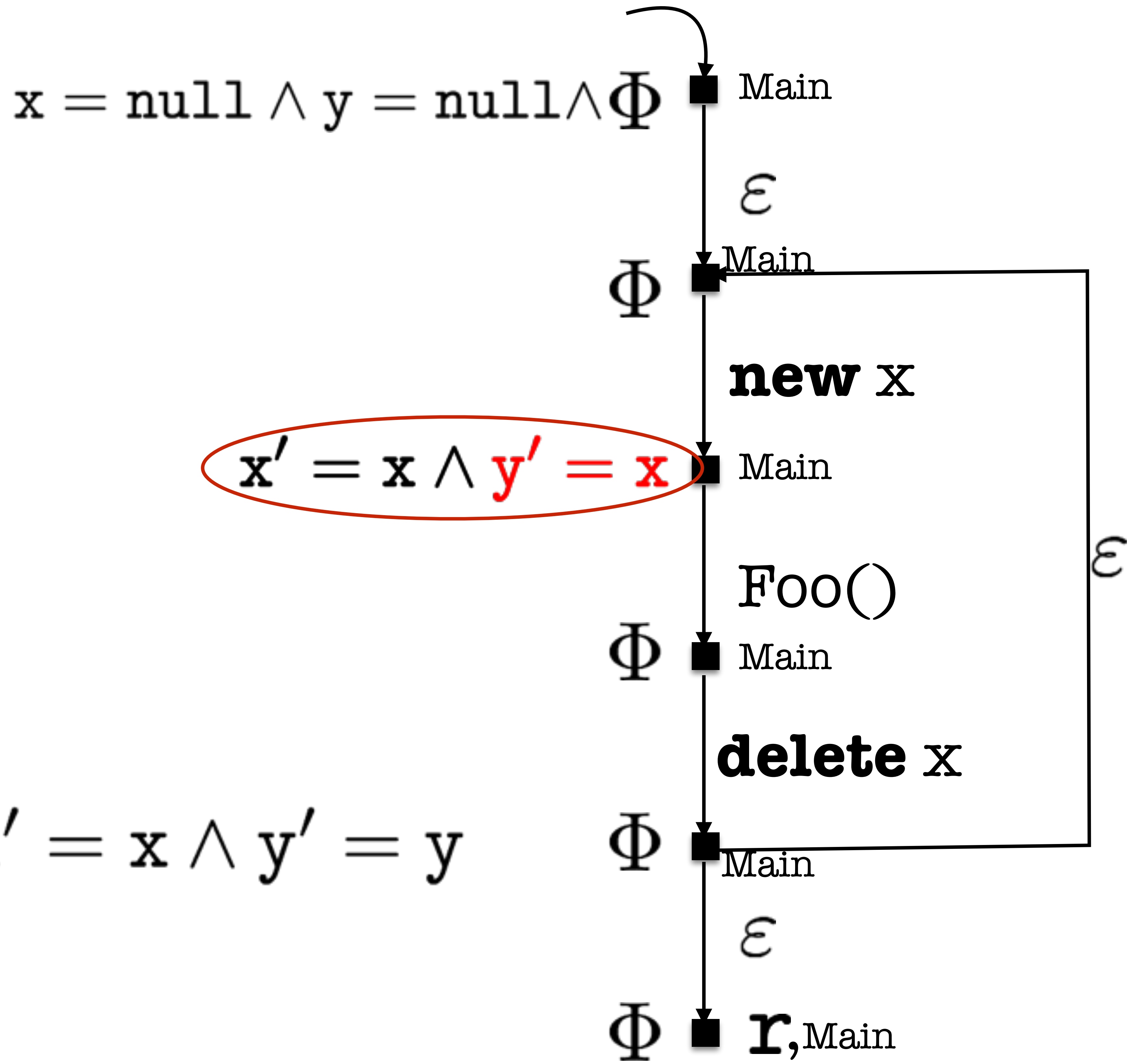


Φ is $x' = x \wedge y' = y$

Structural Models

```
decl x = null;
decl y = null;

Main() {
  while(*) {
    new x;
    y = x;
    Foo();
    delete x;
  }
}
```

```
Foo() {
  .....
}
```

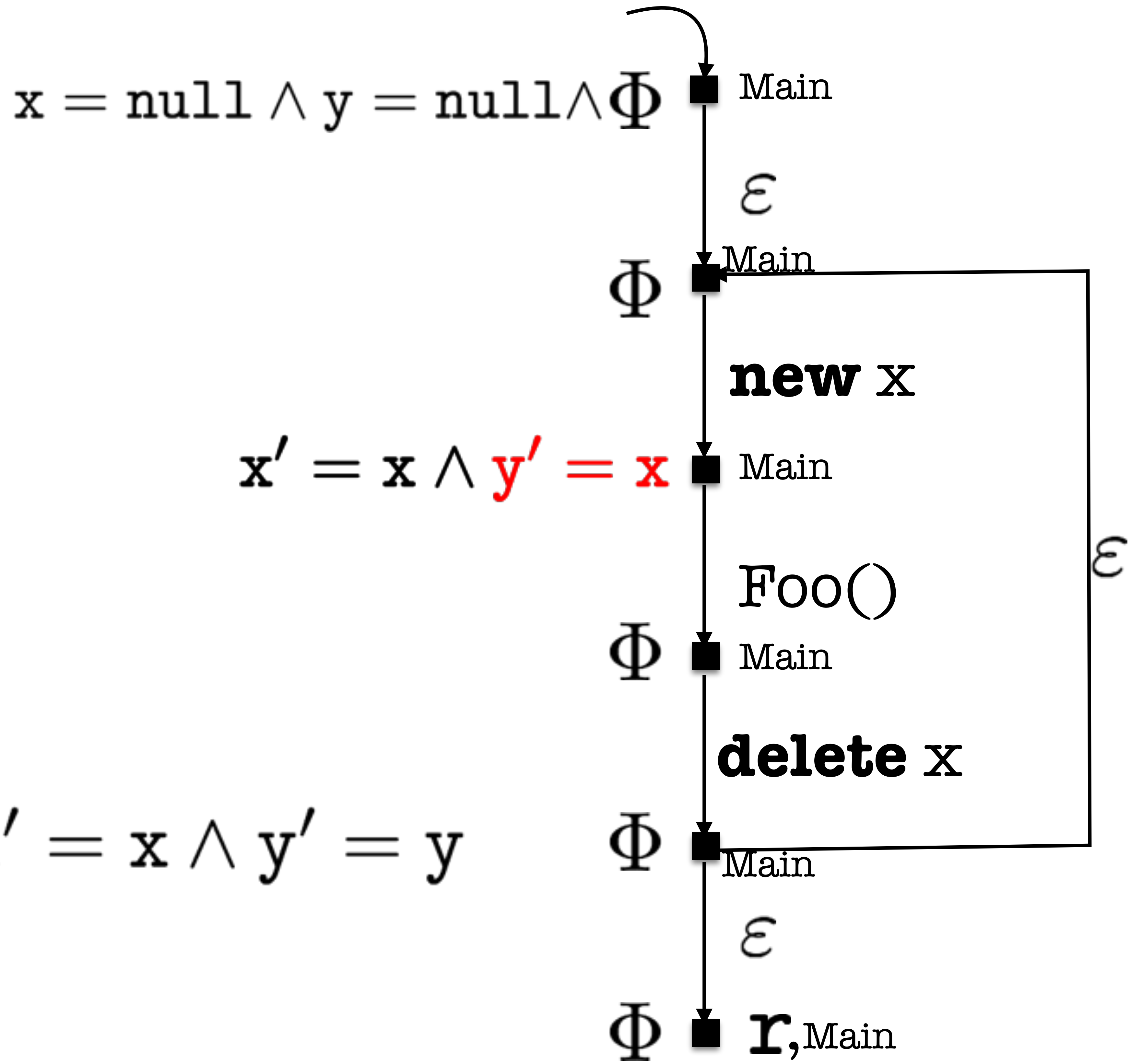


Φ is $x' = x \wedge y' = y$

Structural Models

```
decl x = null;
decl y = null;

Main() {
   while(*) {
     new x;
     y = x;
     Foo();
     delete x;
   }
}
```

```
Foo() {
   .....
}
```



Φ is $x' = x \wedge y' = y$

Overview of the Approach

User Tasks

1. Specification: Local specification (variable) & Global property
2. Define observed instructions

Task I

1. Model check the code of the variable components against their local specifications

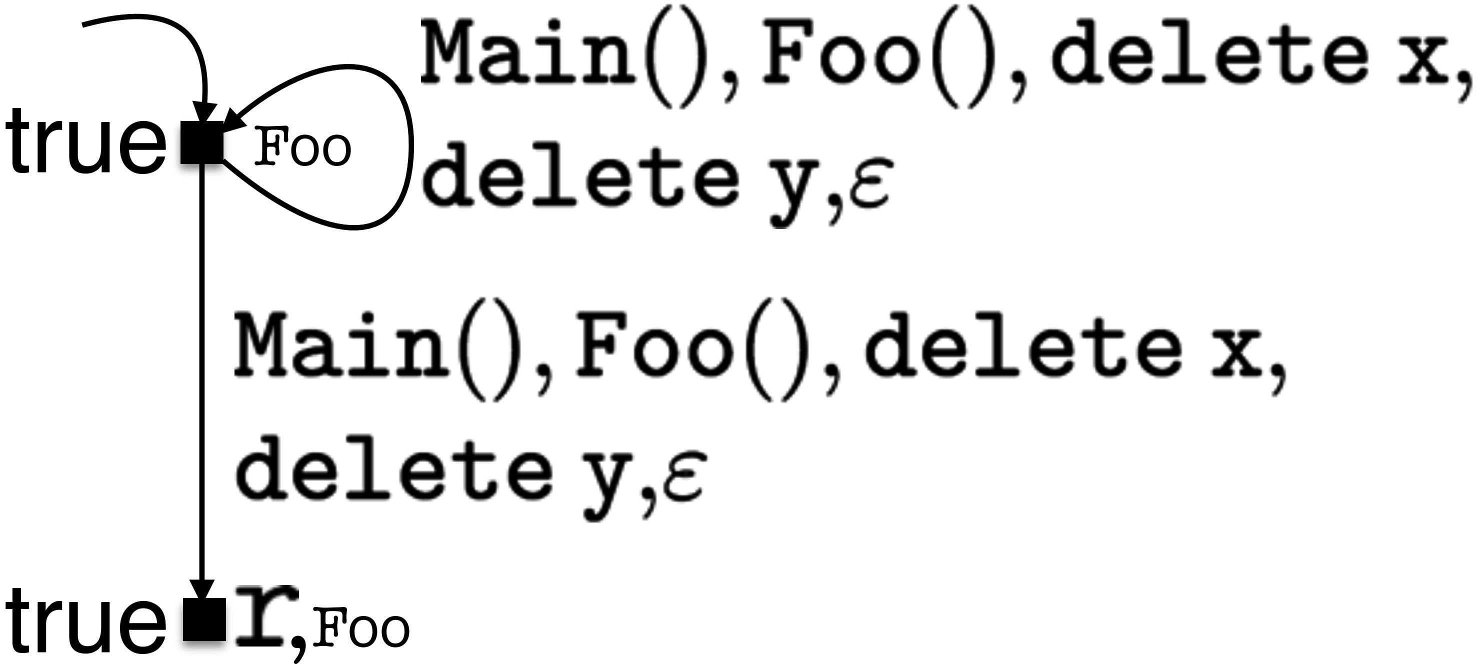
Task II

1. Model extraction from stable code
2. Maximal model construction from local specification
3. Compose models and induce the behavior of the system
4. Model check the behavior

Structural Maximal

```
decl x = null;
decl y = null;

Main() {
  while(*) {
    new x;
    y = x;
    Foo();
    delete x;
  }
}
```

```
Foo() {
  .....
}
```



true

Foo

Main(), Foo(), delete x,
delete y,ε

Main(), Foo(), delete x,
delete y,ε

true

r,Foo

Local Structural Specification of Foo

“no **new** statement”

Overview of the Approach

User Tasks

1. Specification: Local specification (variable) & Global property
2. Define observed instructions

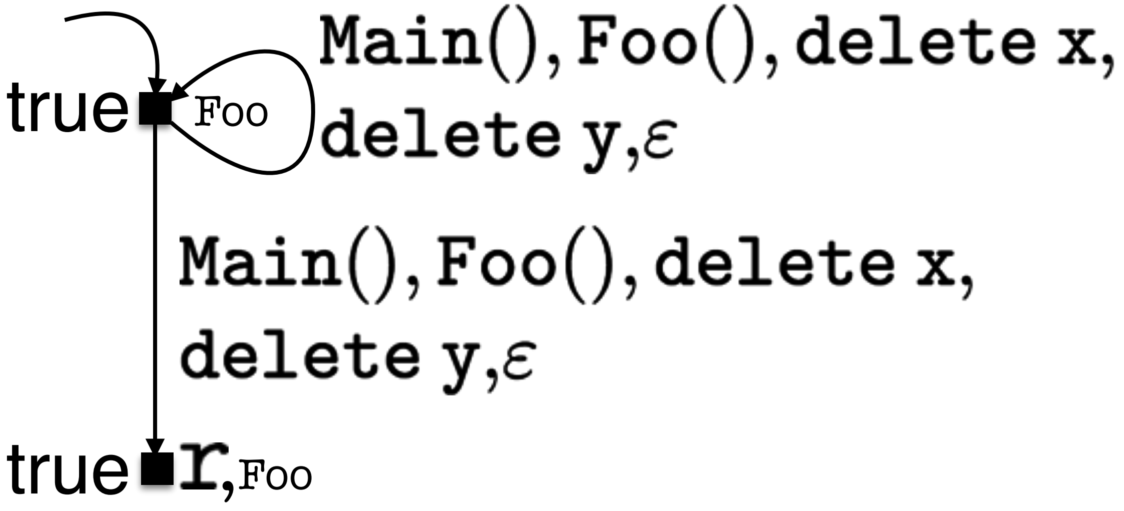
Task I

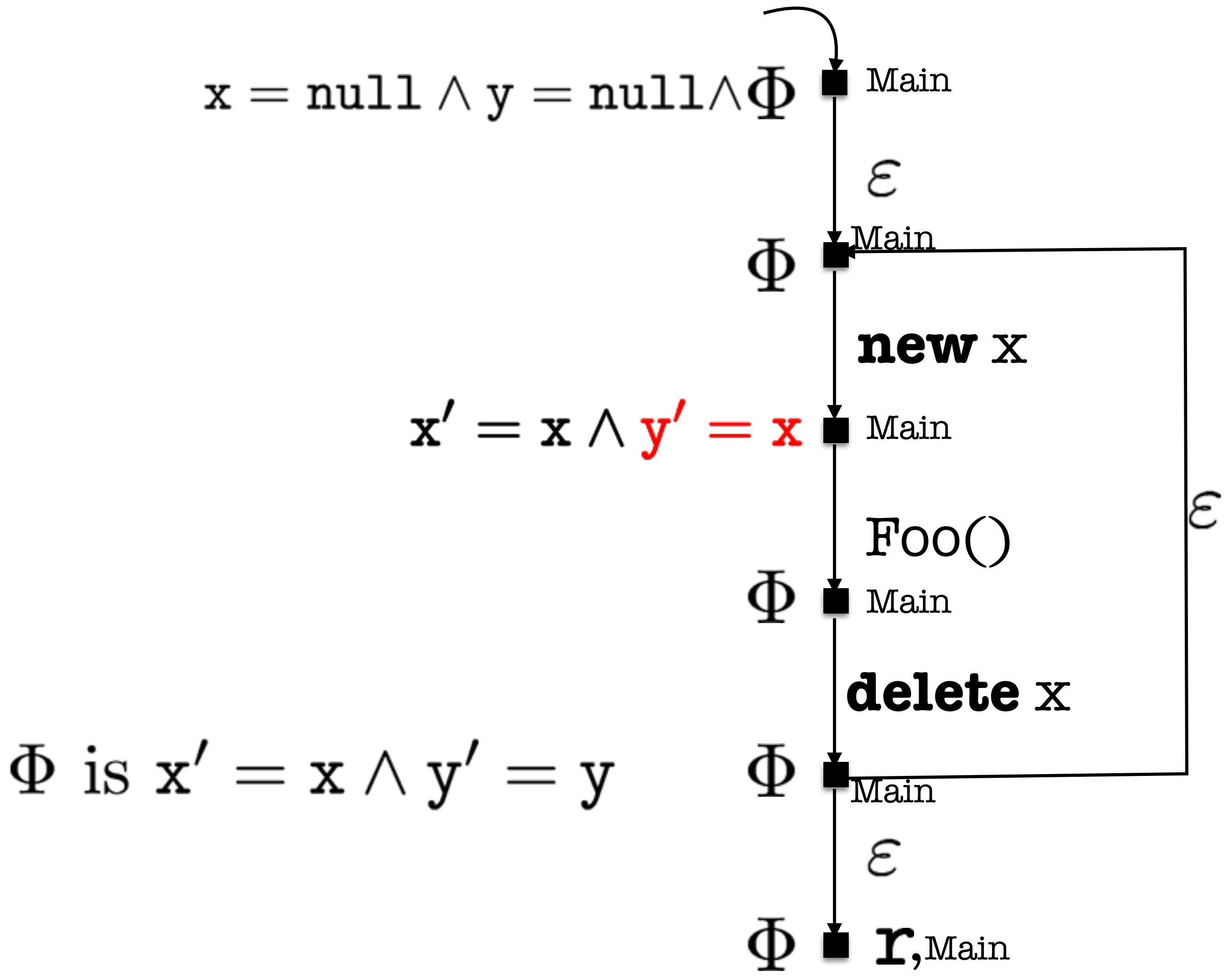
1. Model check the code of the variable components against their local specifications

Task II

1. Model extraction from stable code
2. Maximal model construction from local specification
3. Compose models and induce the behavior of the system
4. Model check the behavior

Composition of Structures


true
Foo
$\mathtt{Main}(), \mathtt{Foo}(), \mathtt{delete}\ \mathtt{x}, \mathtt{delete}\ \mathtt{y}, \varepsilon$
$\mathtt{Main}(), \mathtt{Foo}(), \mathtt{delete}\ \mathtt{x}, \mathtt{delete}\ \mathtt{y}, \varepsilon$
true
$\mathtt{r}, \mathrm{Foo}$


$\mathtt{x} = \mathtt{null} \wedge \mathtt{y} = \mathtt{null} \wedge \Phi$
Main
ε
Φ
Main
new x
$\mathtt{x}' = \mathtt{x} \wedge \mathtt{y}' = \mathtt{x}$
Main
Foo()
ε
Φ
Main
delete x
Φ
Main
ε
Φ
$\mathtt{r}, \mathrm{Main}$

Φ is $\mathtt{x}' = \mathtt{x} \wedge \mathtt{y}' = \mathtt{y}$

Overview of the Approach

User Tasks

1. Specification: Local specification (variable) & Global property
2. Define observed instructions

Task I

1. Model check the code of the variable components against their local specifications

Task II

1. Model extraction from stable code
2. Maximal model construction from local specification
3. Compose models and induce the behavior of the system
4. Model check the behavior

Overview of the Approach

User Tasks

1. Specification: Local specification (variable) & Global property
2. Define observed instructions

Task I

1. Model check the code of the variable components against their local specifications (**Quick and Easy**)

Task II

1. Model extraction from stable code
2. Maximal model construction from local specification
3. Compose models and induce the behavior of the system
4. Model check the behavior

Instantiations of the Framework

Instantiation for Program Models without Data

- No observed instruction
  - Labels: function calls and ε
- Abstract away all program data (no assertions, semantic entailment)

Instantiation for Program Models with Boolean Data

- No observed instruction
  - Labels: function calls and ε
- Assertions capture the effect of assignments and conditions

Instantiation for Program Models with Pointer Data

- Observed instruction: **`new`**, **`delete`**
  - Labels: function calls, new, delete and ε
- Assertions capture the effect of assignments and conditions

Contributions

Contributions

Tool Support:

1. A set of stand alone tools, CVPP toolset (existed before)
2. A fully automated tool for procedure-modular verification of programs with full data abstraction
   PROMOVER

- Siavash Soleimanifard, Dilian Gurov, and Marieke Huisman. *Procedure-modular verification of control flow safety properties.* In FTfJP '10
- Siavash Soleimanifard, Dilian Gurov, and Marieke Huisman. *ProMoVer: Modular verification of temporal safety properties.* In SEFM '11
- Siavash Soleimanifard, Dilian Gurov, and Marieke Huisman. *Procedure-modular specification and verification of temporal safety properties.* In Journal of Software and System Modelling. **(Paper I)**

Contributions cont.

Verification of product families:

1. A hierarchical variable model for capturing commonality and variability of products
2. A extension of our verification technique for the efficient verification of product families represented by hierarchical variability models

- Ina Schaefer, Dilian Gurov, and Siavash Soleimanifard. *Compositional algorithmic verification of software product lines.* In FMCO '10.
- Siavash Soleimanifard, Dilian Gurov, Bjarte Østvold, and Minko Markov.
  Model Mining and Efficient Verification of Software Product Lines.
  Submitted to Journal of Logic and Algebraic Programming. **(Paper II)**

Contributions cont.

Development of the framework:

1. A framework for compositional verification with full data abstraction (existed before)
2. A generic framework for verification of procedural programs in the presence of variability
    - Three instantiations of the framework

- Siavash Soleimanifard, and Dilian Gurov. *Algorithmic verification of procedural programs in the presence of code variability*. In *FACS '14*.

- Siavash Soleimanifard, and Dilian Gurov. *Algorithmic verification of procedural programs in the presence of code variability. Technical Report* **(Paper III)**

ProMoVer

ProMoVer

Fully automated tool for **procedure-modular** verification of **Java** programs with **data-less instantiation** of the generic framework

ProMoVer -- Usage

```
/**
 *  @global_ltl_prop:
 *    even -> X ((even && ! entry) W odd)
 */
public class Number {
    /** @local_interface:  required odd
     *  @local_ltl_prop:
     *    G(X (!even || !entry) && (odd -> X G even))
     */
    public boolean even(int n) {
        if (n == 0) return true;
        else return odd(n-1);
    }
    /** @local_interface:  required even
     *  @local_ltl_prop:
     *    G(X (!odd || !entry) && (even -> X G odd))
     */
    public boolean odd(int n) {
        if (n == 0) return false;
        else return even(n-1);
    }
}
```

ProMoVer -- Usage

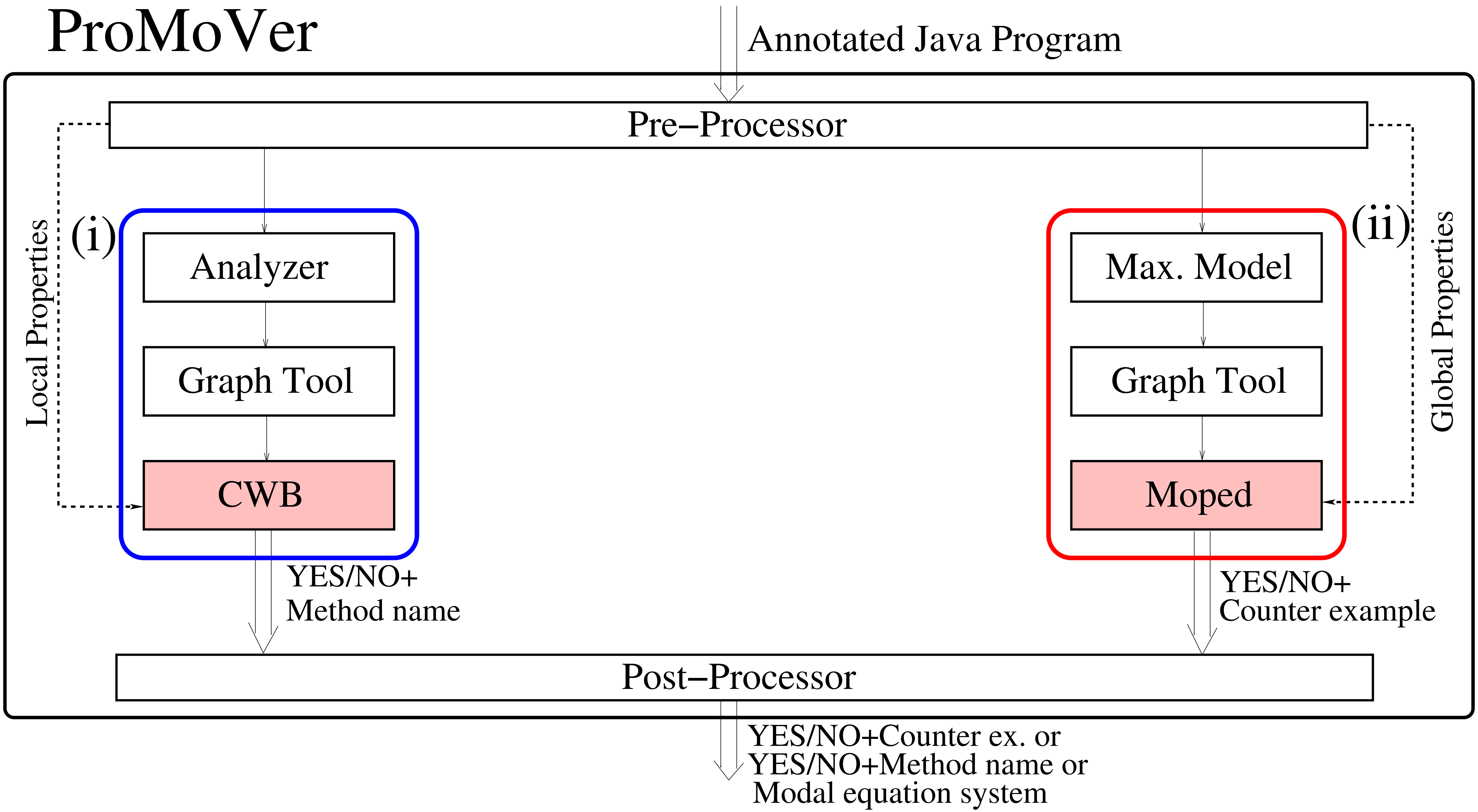
```
/**
 *  @global_ltl_prop:
 *    even -> X ((even && ! entry) W odd)
 */
public class Number {
    /** @local_interface: re
     *  @local_ltl_prop:
     *    G(X (!even || !entry
     */
    public boolean even(int
        if (n == 0) return tru
        else return odd(n-1);
    }
    /** @local_interface: required even
     *  @local_ltl_prop:
     *    G(X (!odd || !entry) && (even -> X G odd))
     */
    public boolean odd(int n) {
        if (n == 0) return false;
        else return even(n-1);
    }
}
```

Verification Result:

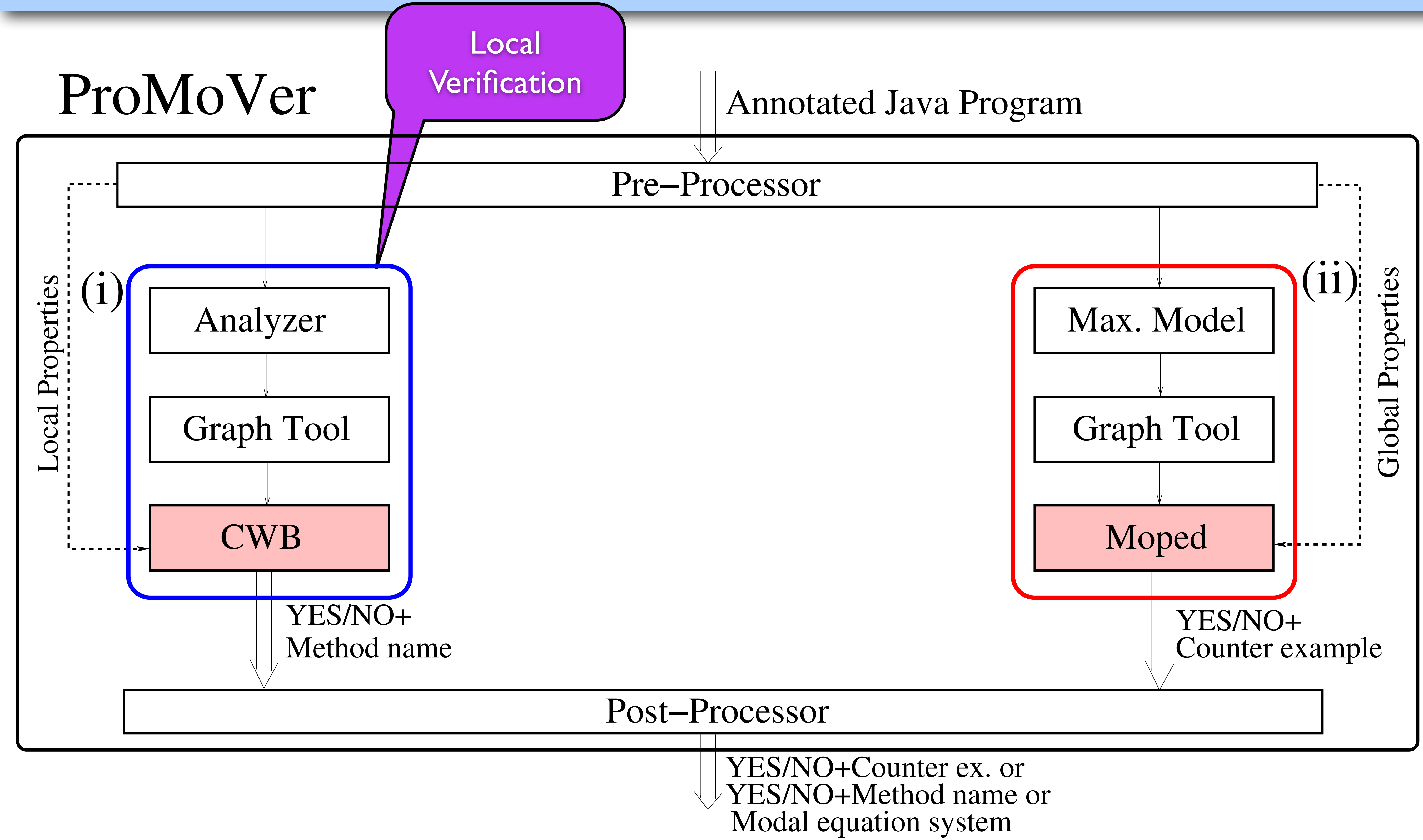
YES/NO

$$(\texttt{even}, \varepsilon) \xrightarrow{\texttt{even call odd}} (\texttt{odd}, \texttt{even}) \xrightarrow{\texttt{odd ret even}} (\texttt{even}, \varepsilon)$$

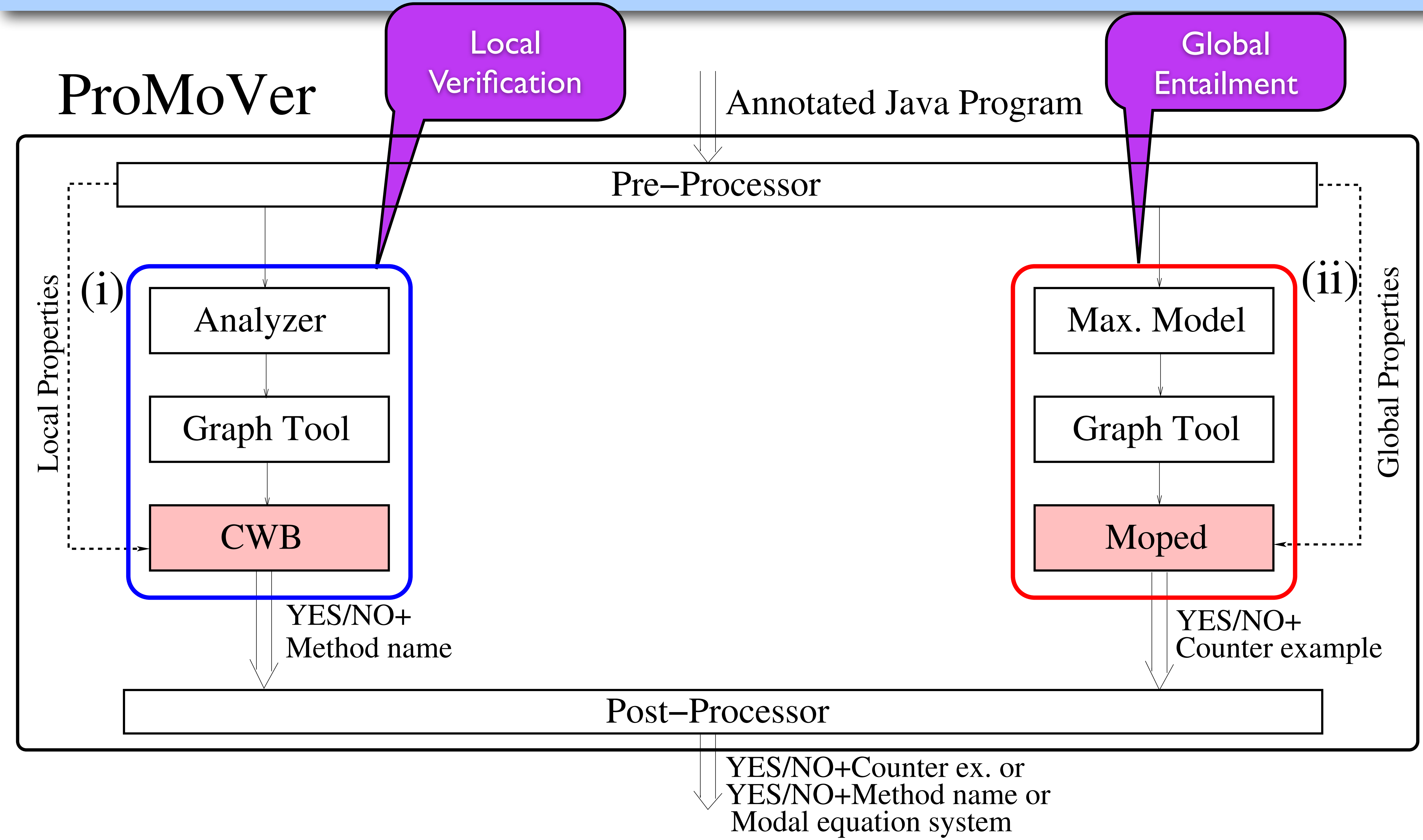
ProMoVer


ProMoVer
Annotated Java Program
Pre–Processor
Local Properties
(i)
Analyzer
Graph Tool
CWB
(ii)
Max. Model
Graph Tool
Moped
Global Properties
YES/NO+
Method name
YES/NO+
Counter example
Post–Processor
YES/NO+Counter ex. or
YES/NO+Method name or
Modal equation system

ProMoVer


Local Verification
ProMoVer
Annotated Java Program
Pre–Processor
Local Properties
(i)
Analyzer
Graph Tool
CWB
YES/NO+
Method name
(ii)
Max. Model
Graph Tool
Moped
Global Properties
YES/NO+
Counter example
Post–Processor
YES/NO+Counter ex. or
YES/NO+Method name or
Modal equation system

ProMoVer



Local Verification

Global Entailment

ProMoVer

Annotated Java Program

Pre–Processor

Local Properties

(i)

Analyzer

Graph Tool

CWB

YES/NO+
Method name

(ii)

Global Properties

Max. Model

Graph Tool

Moped

YES/NO+
Counter example

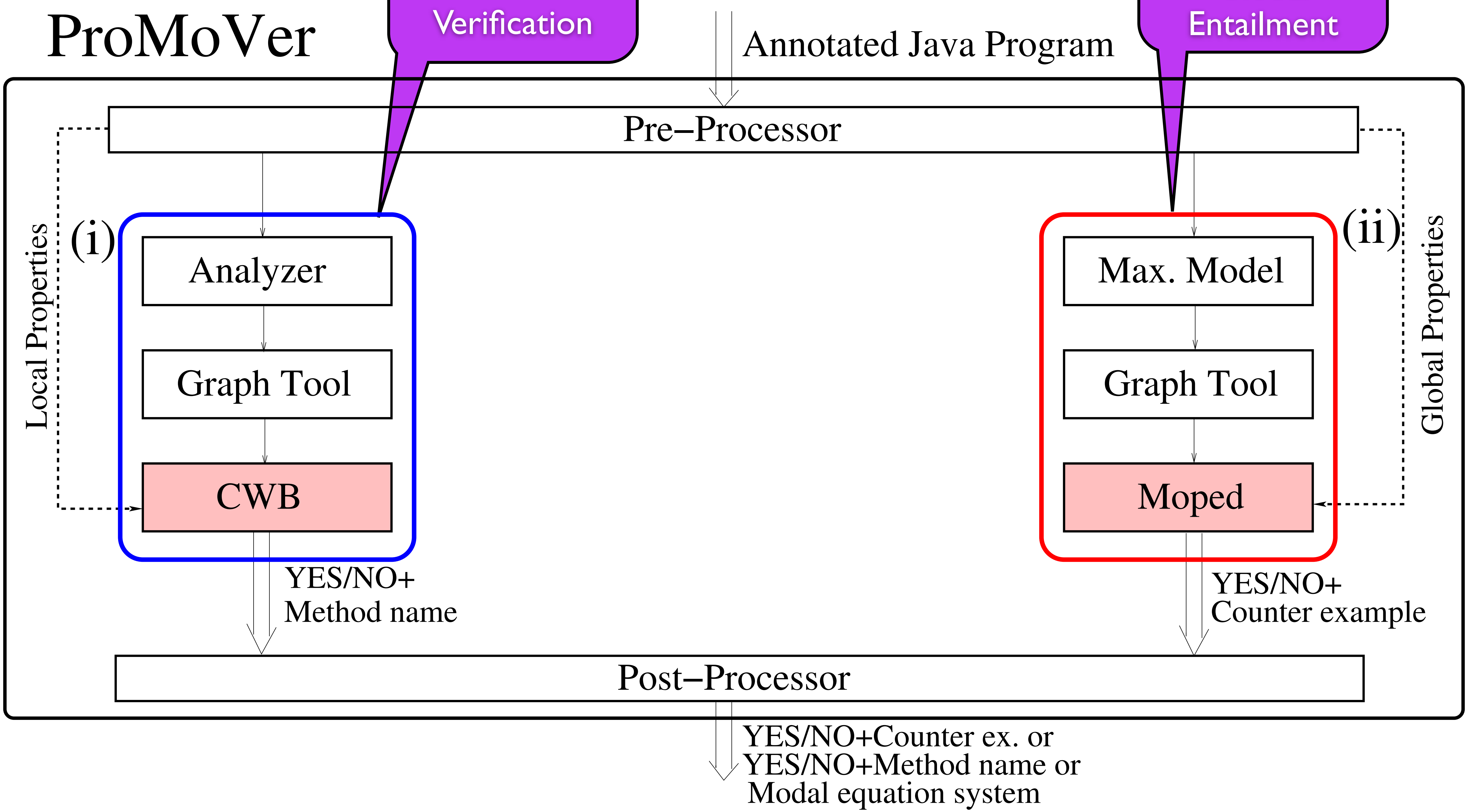
Post–Processor

YES/NO+Counter ex. or
YES/NO+Method name or
Modal equation system

ProMoVer -- Advanced Features

- Different specification languages
  - Safety fragment of modal mu-calculus
  - Modal equation systems
  - Safety LTL
  - Safety automata
- Specification extractor
- Proof storage and reuse mechanism

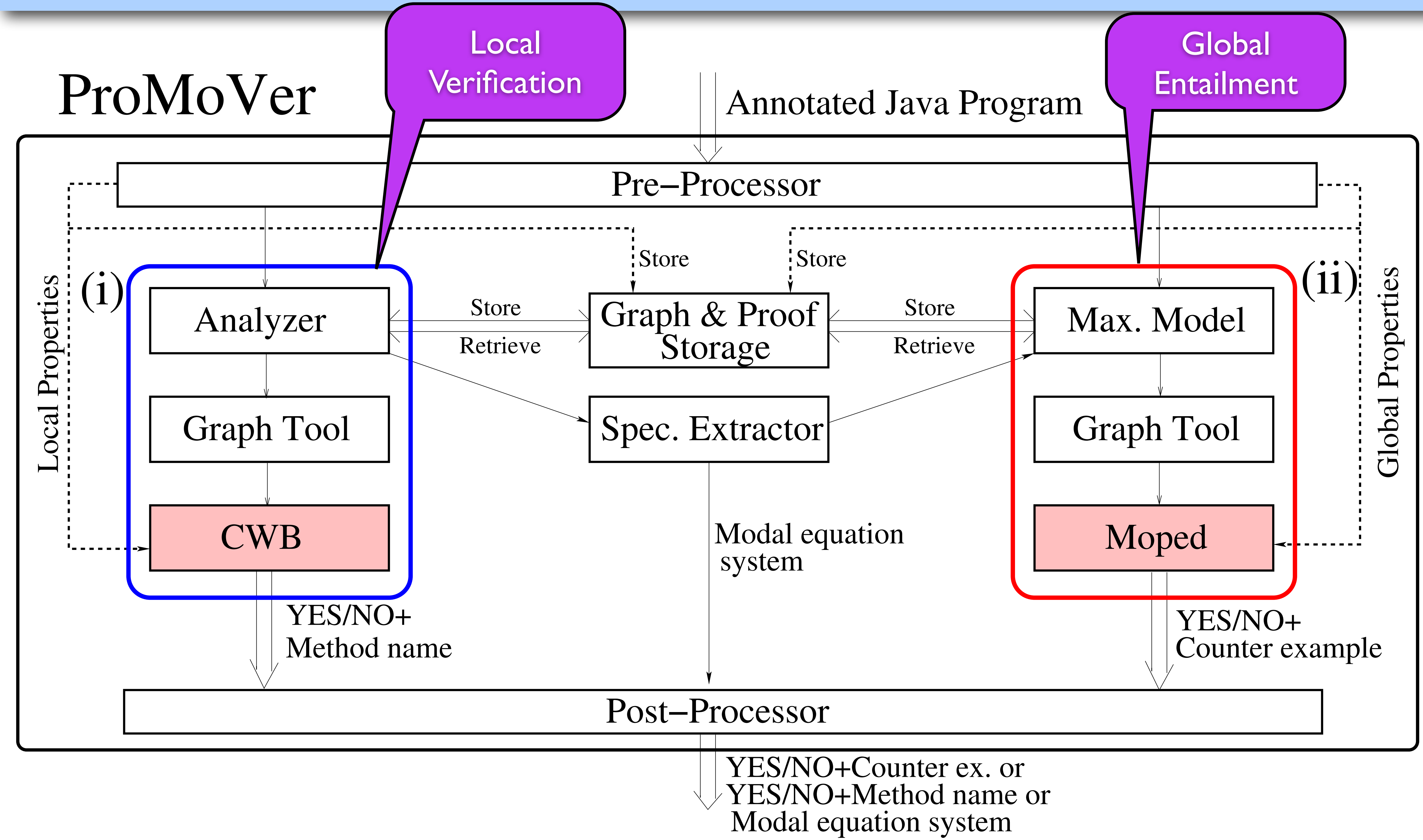
ProMoVer



Local Verification

Global Entailment

ProMoVer -- Advanced Features



A Case Study

Global Property

“only a single database connection should be created for each request and it should be properly closed”

A Case Study

Global Property

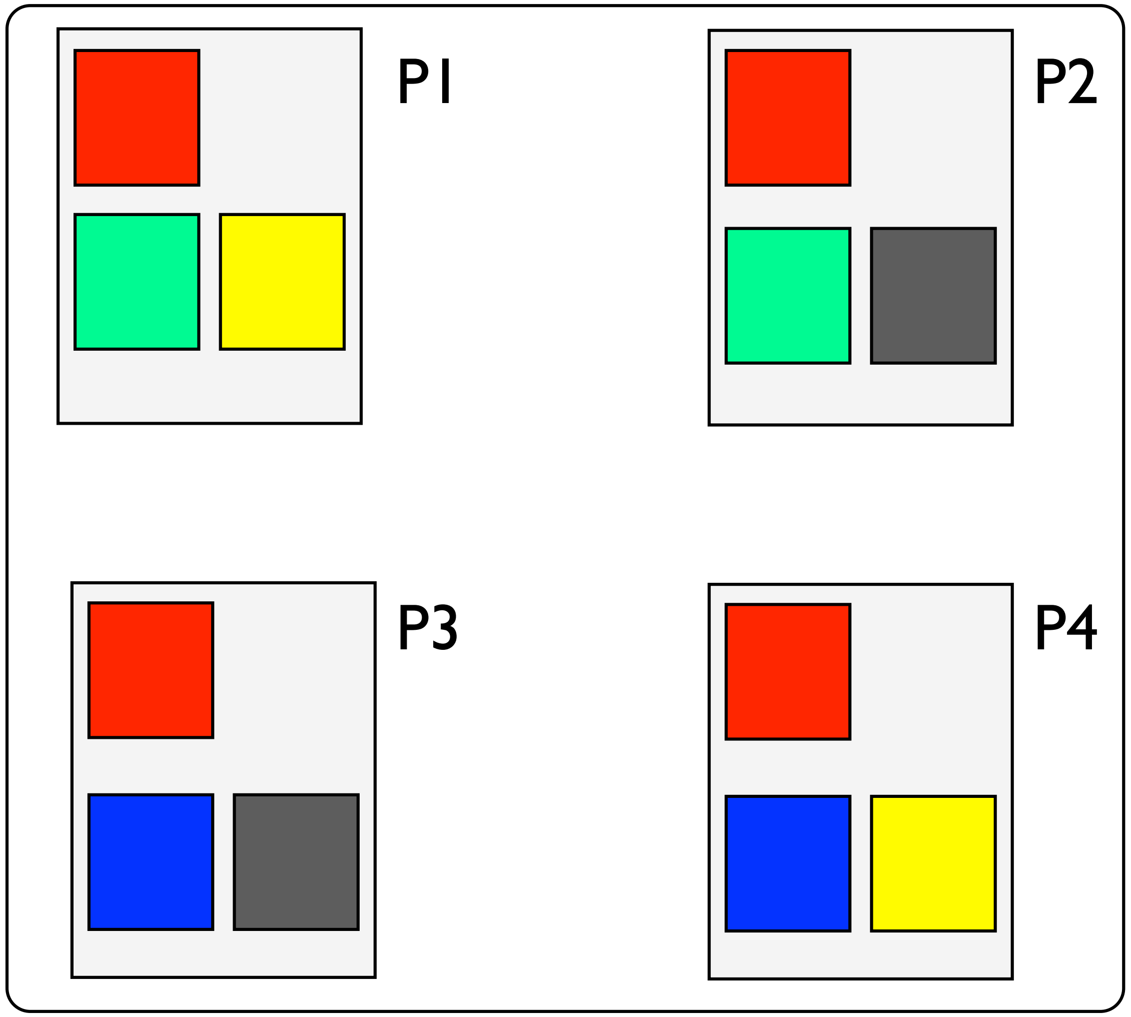
“only a single database connection should be created for each request and it should be properly closed”

Application	Lines of Code	Task I	Maximal Model Cons.	Global Model Check	Total Time
Sail-Web app.	3038	NA	8 sec	1 sec	71 sec
Sail-Web app. Improved	3057	NA	NA	1 sec	22 sec
Sail-Web app. Full-version	10844	32 sec	NA	NA	32 sec

Product Families

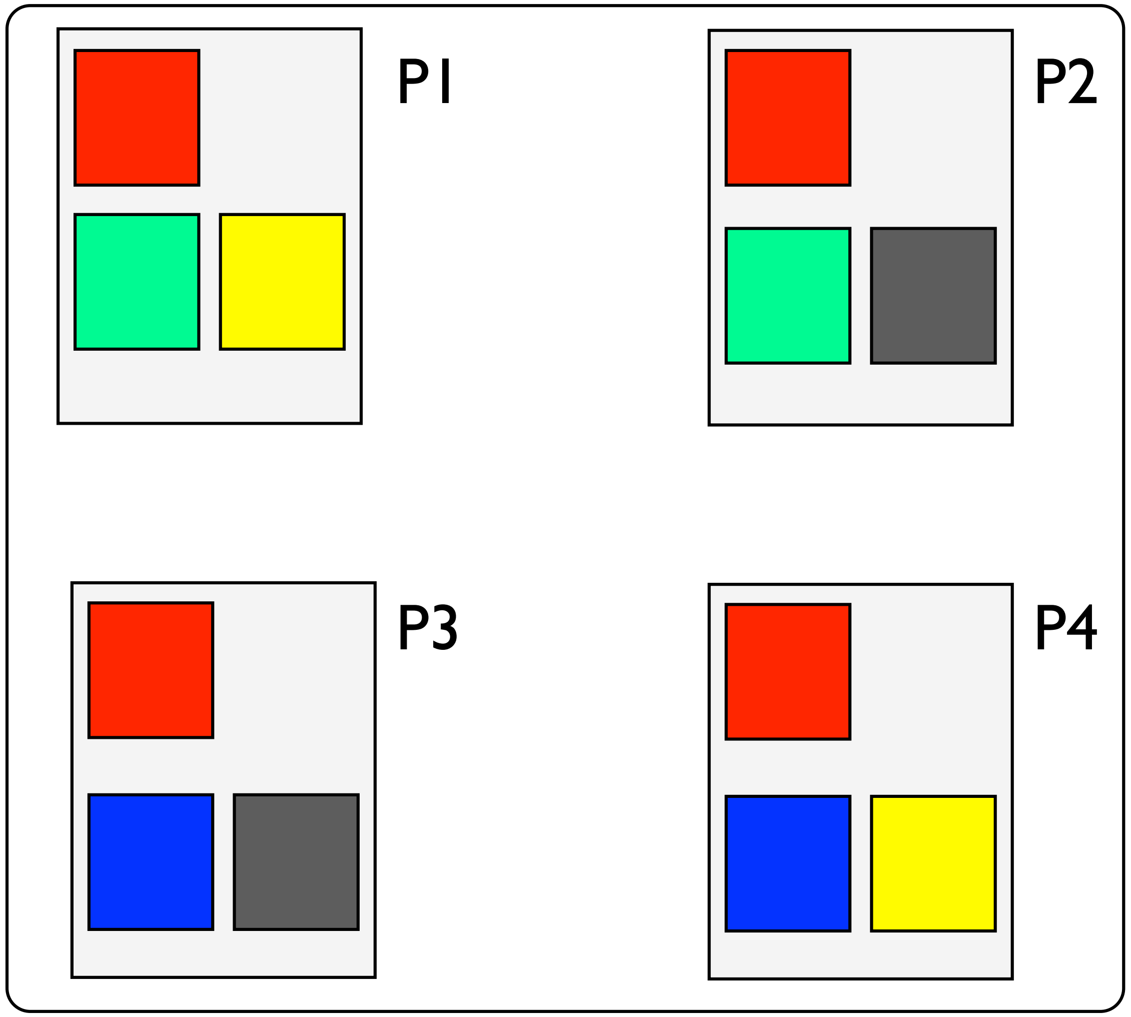
Product Families

Set of products with well-defined **commonalities** and **variabilities**



Product Families

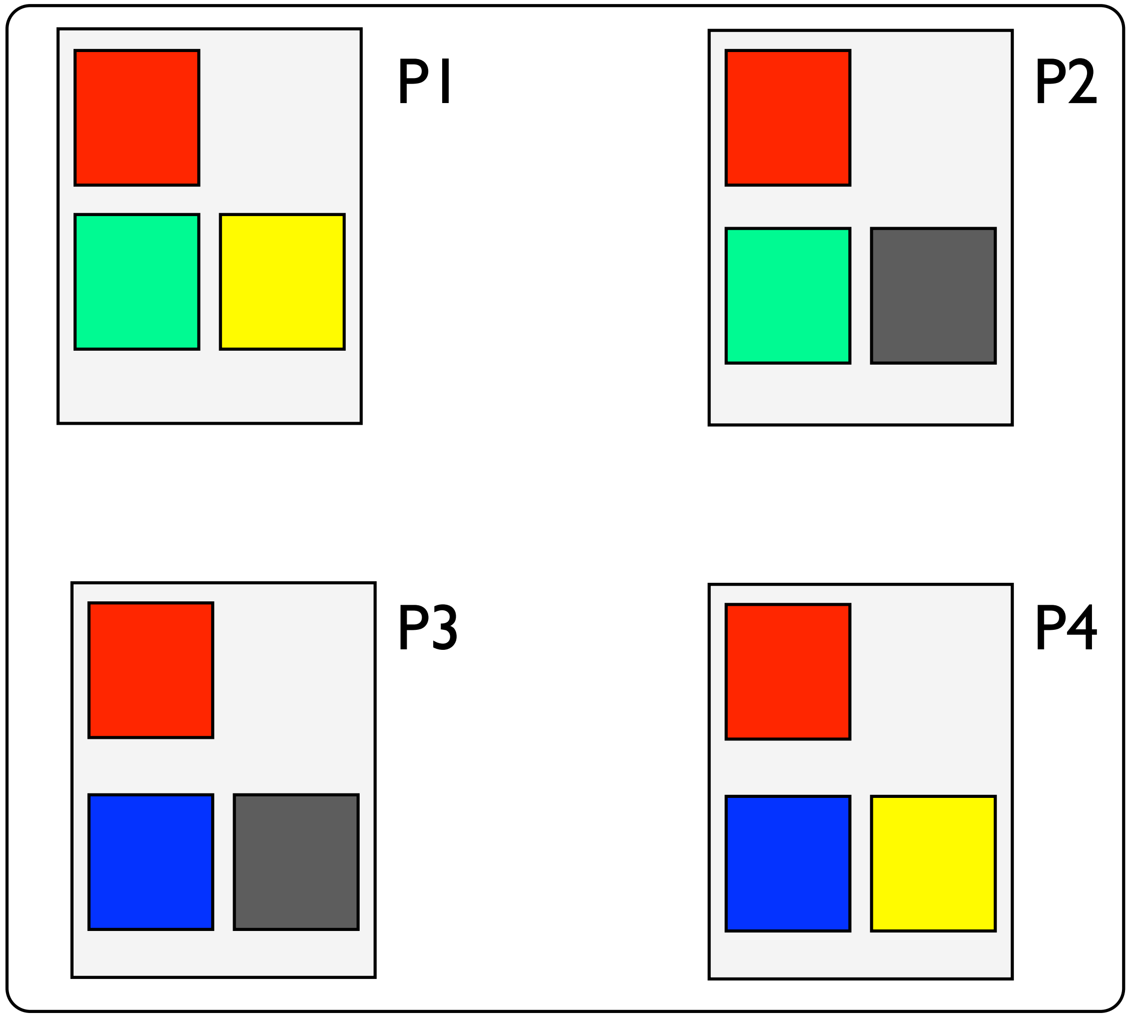
Set of products with well-defined **commonalities** and **variabilities**



$: \ \theta$

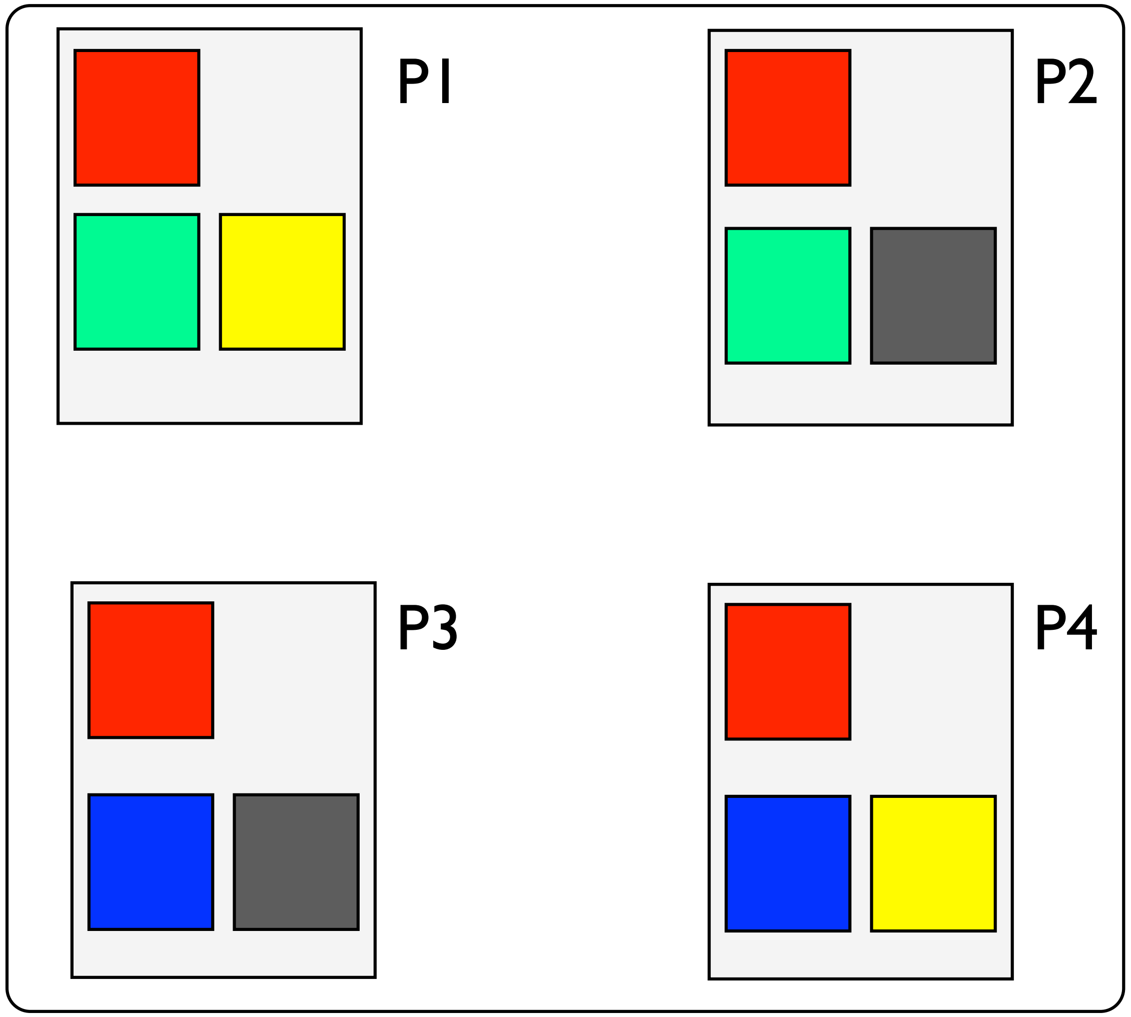
Product Families

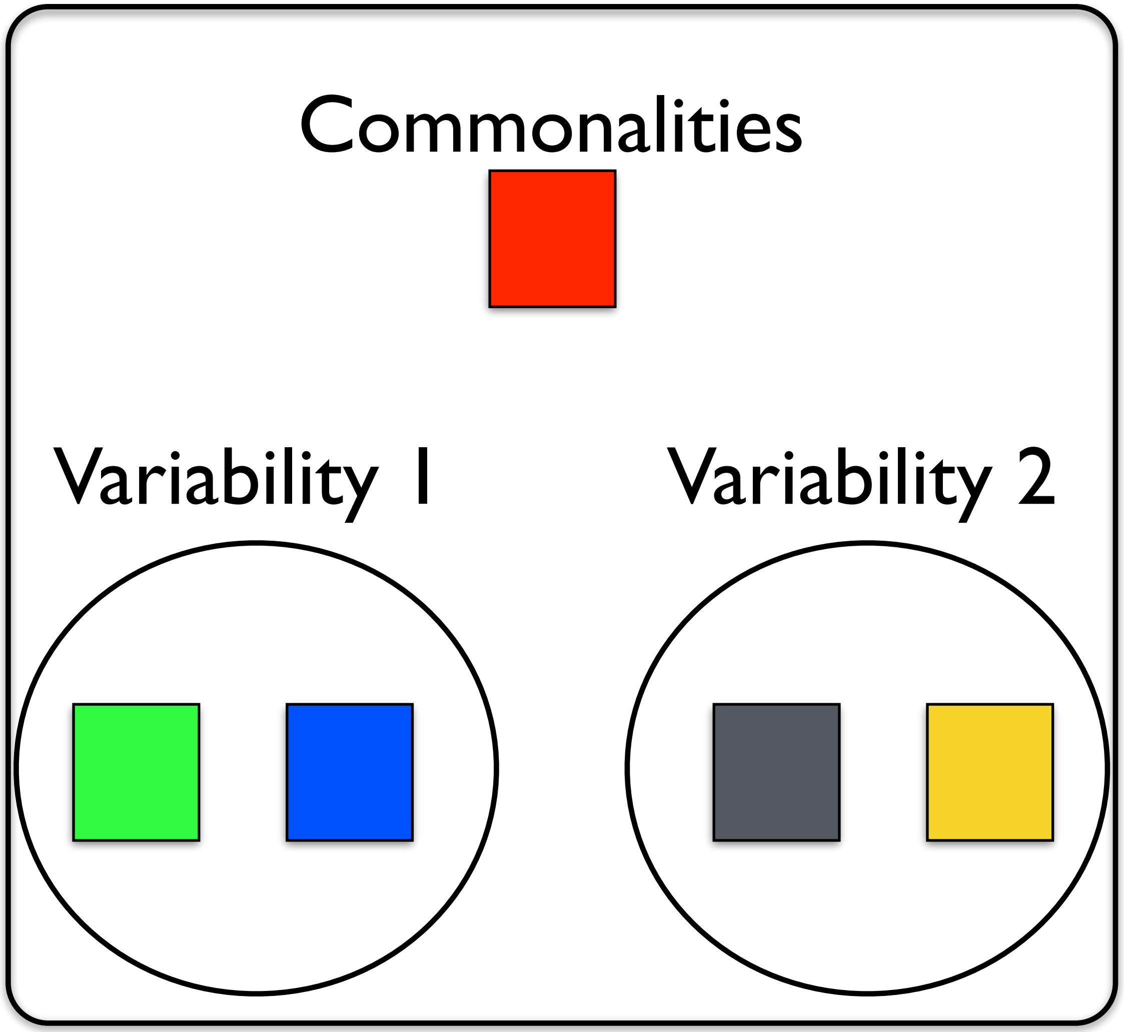
Set of products with well-defined **commonalities** and **variabilities**



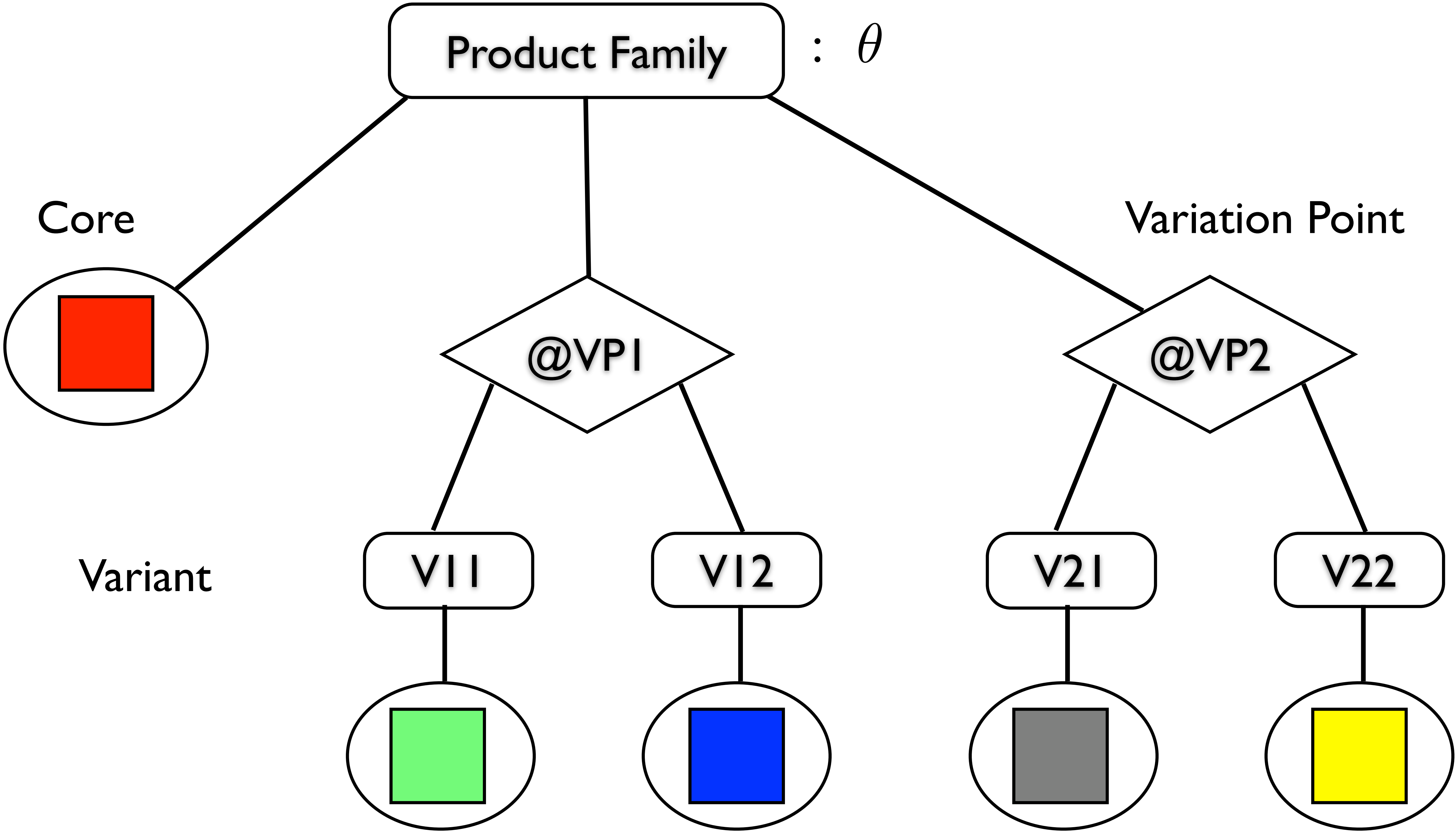
Product Families

Set of products with well-defined **commonalities** and **variabilities**

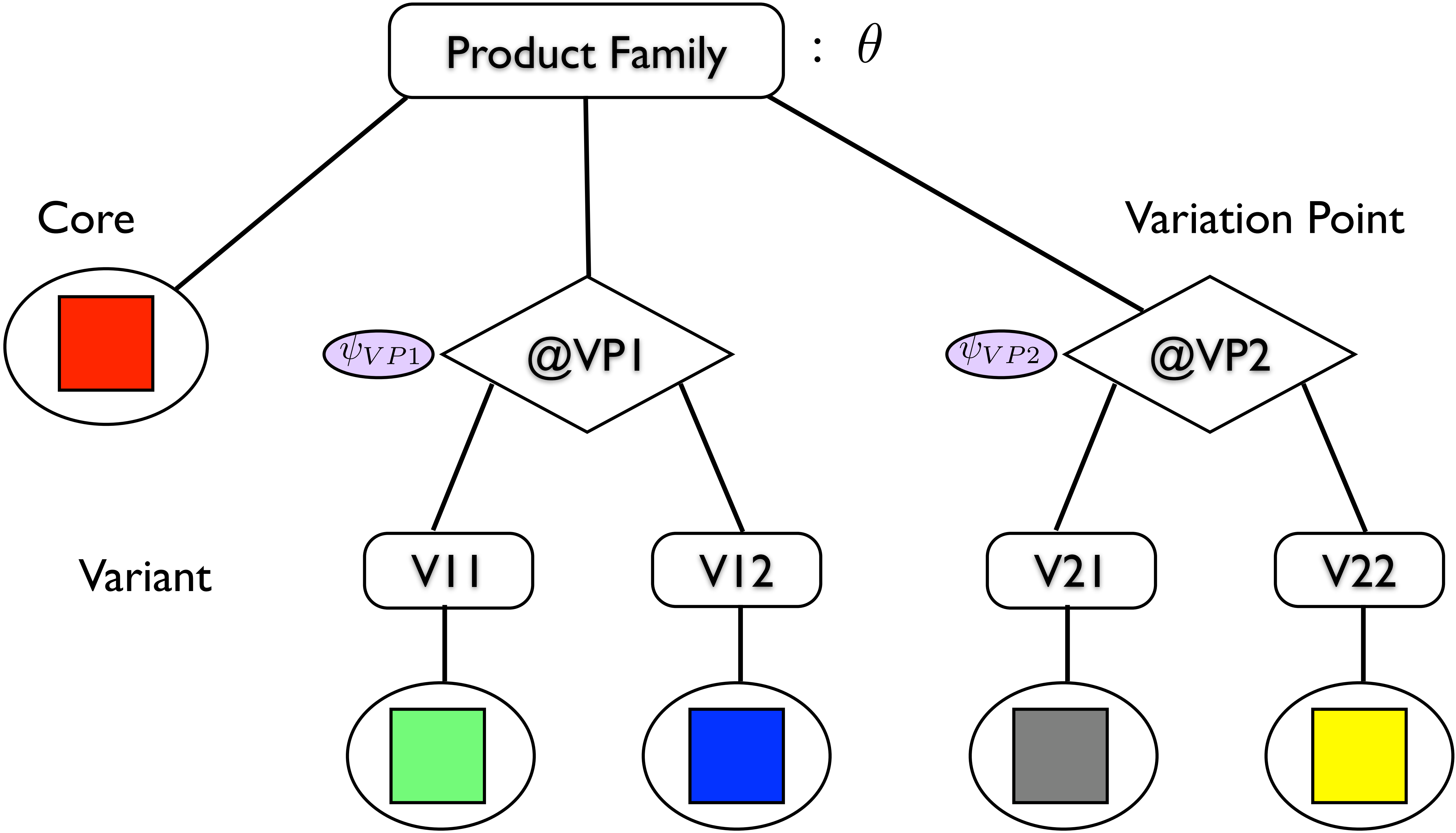




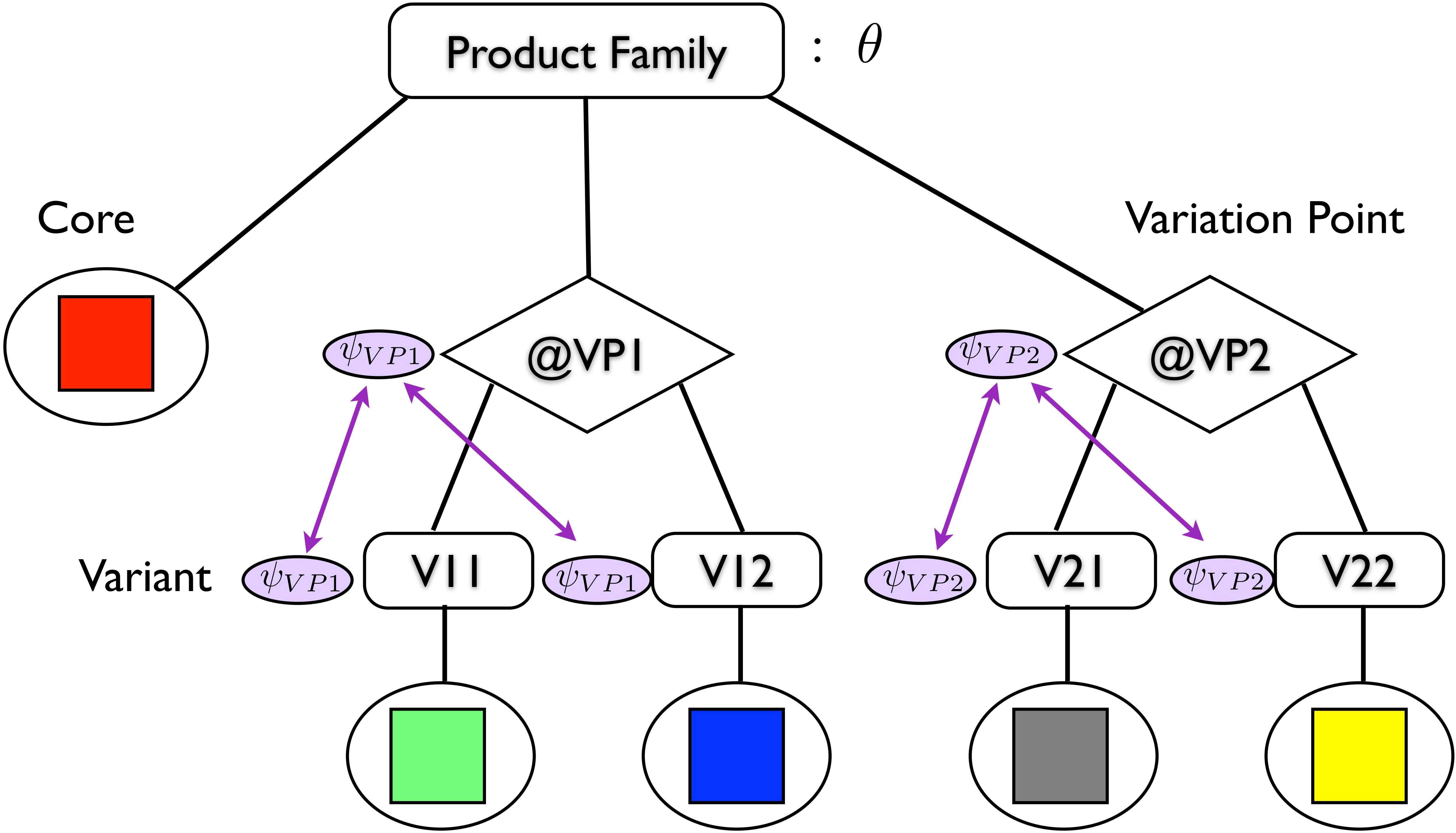
Hierarchical Variability



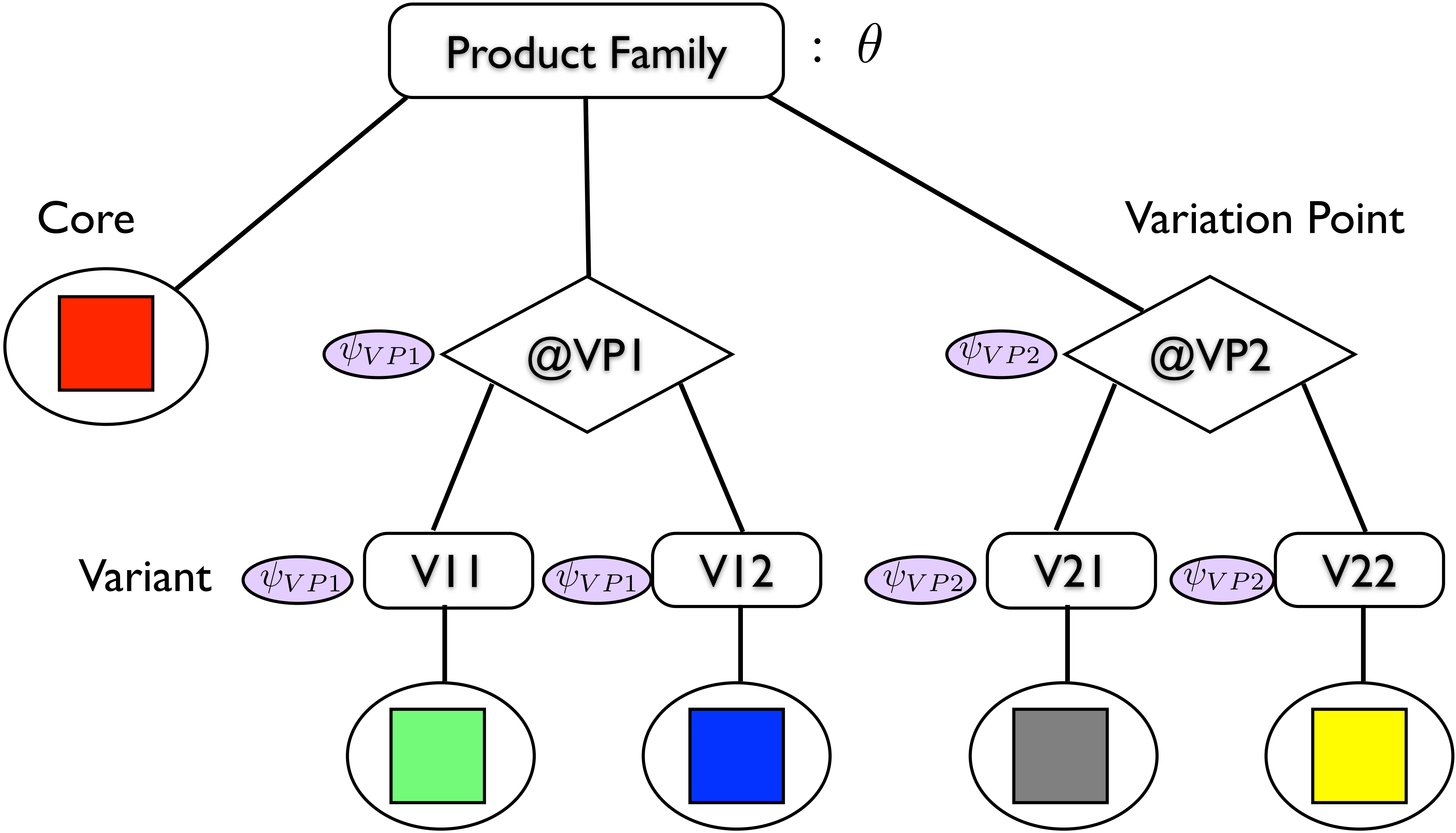
Hierarchical Variability



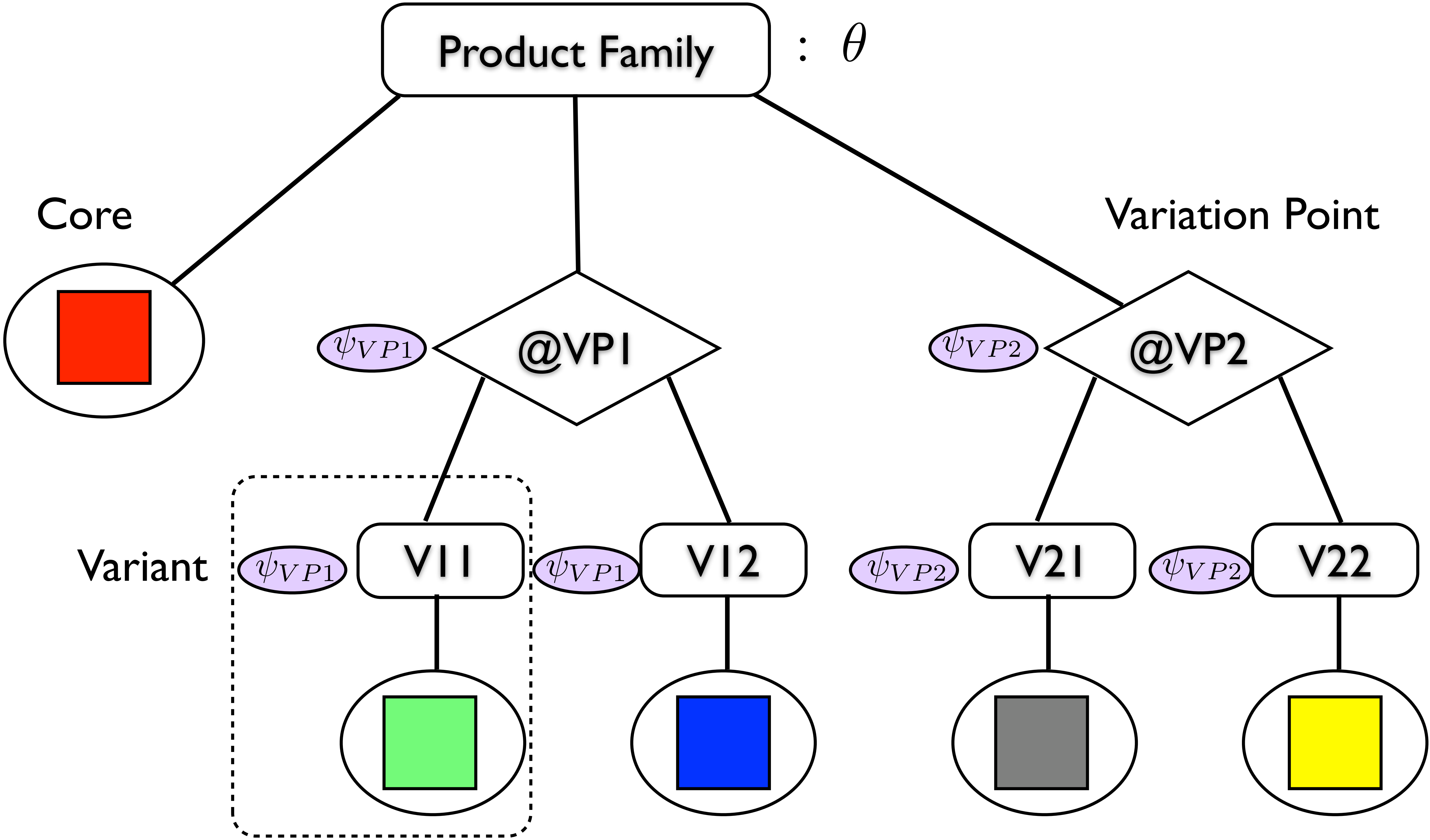
Hierarchical Variability



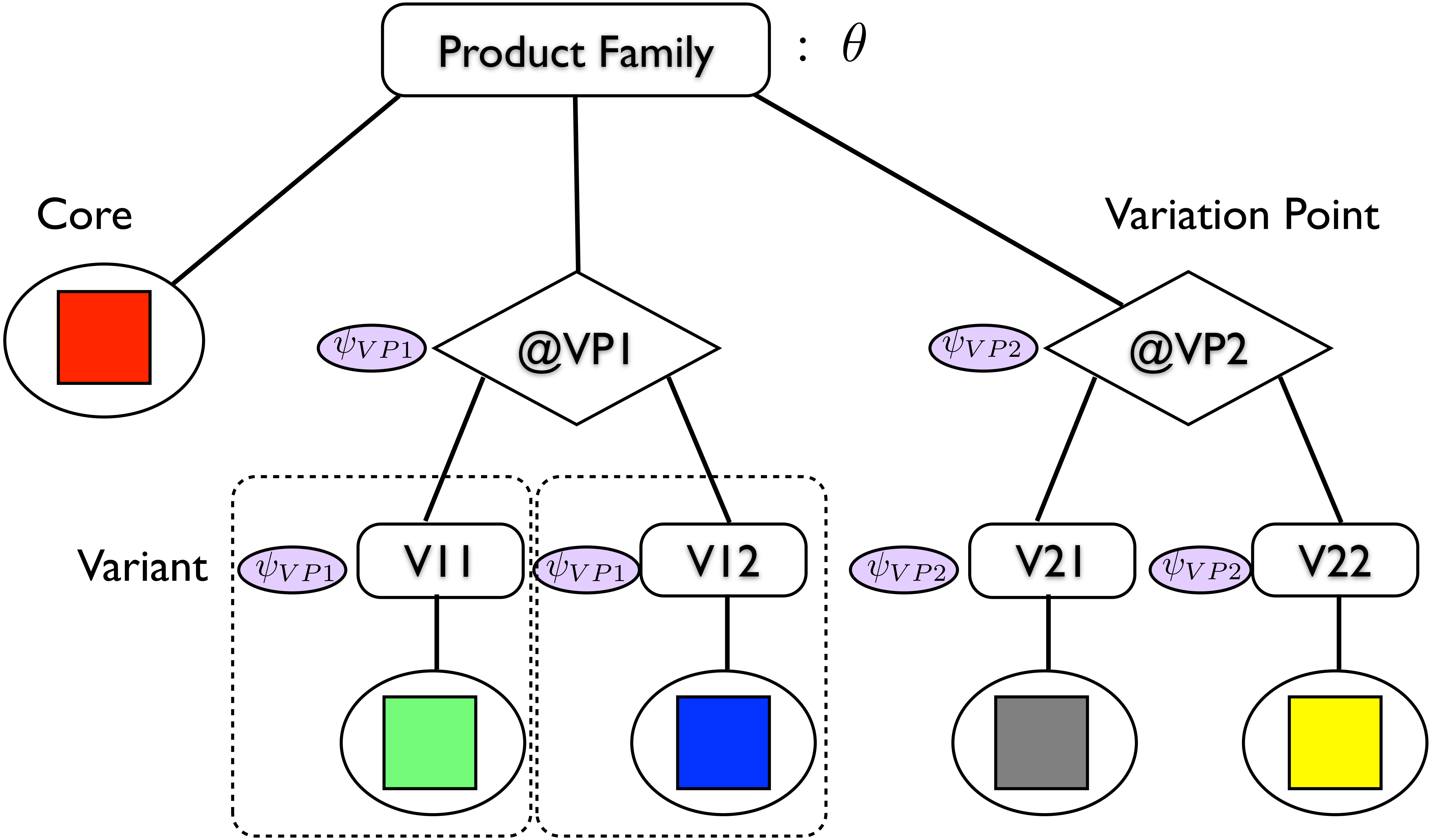
Hierarchical Variability



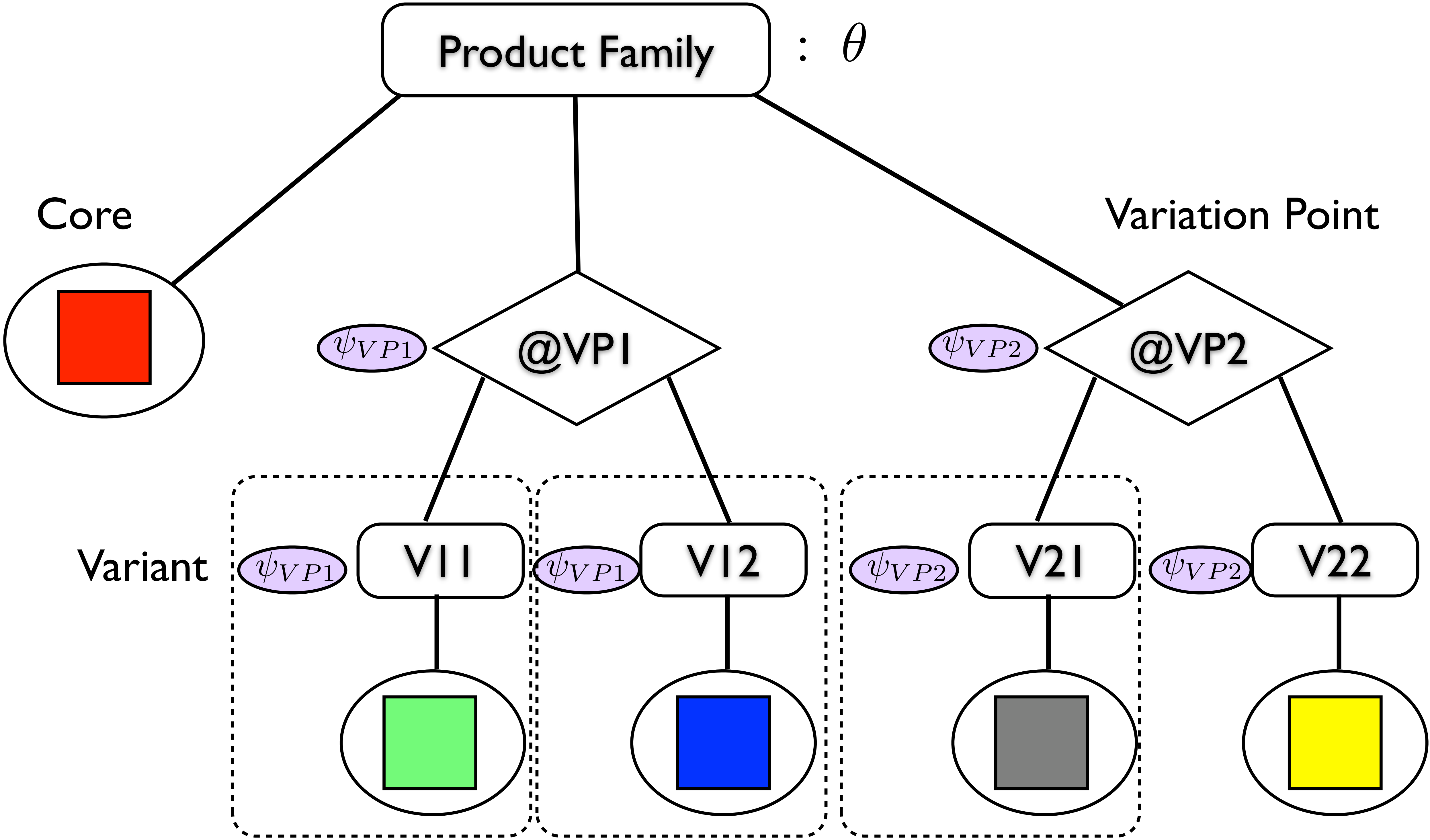
Hierarchical Variability



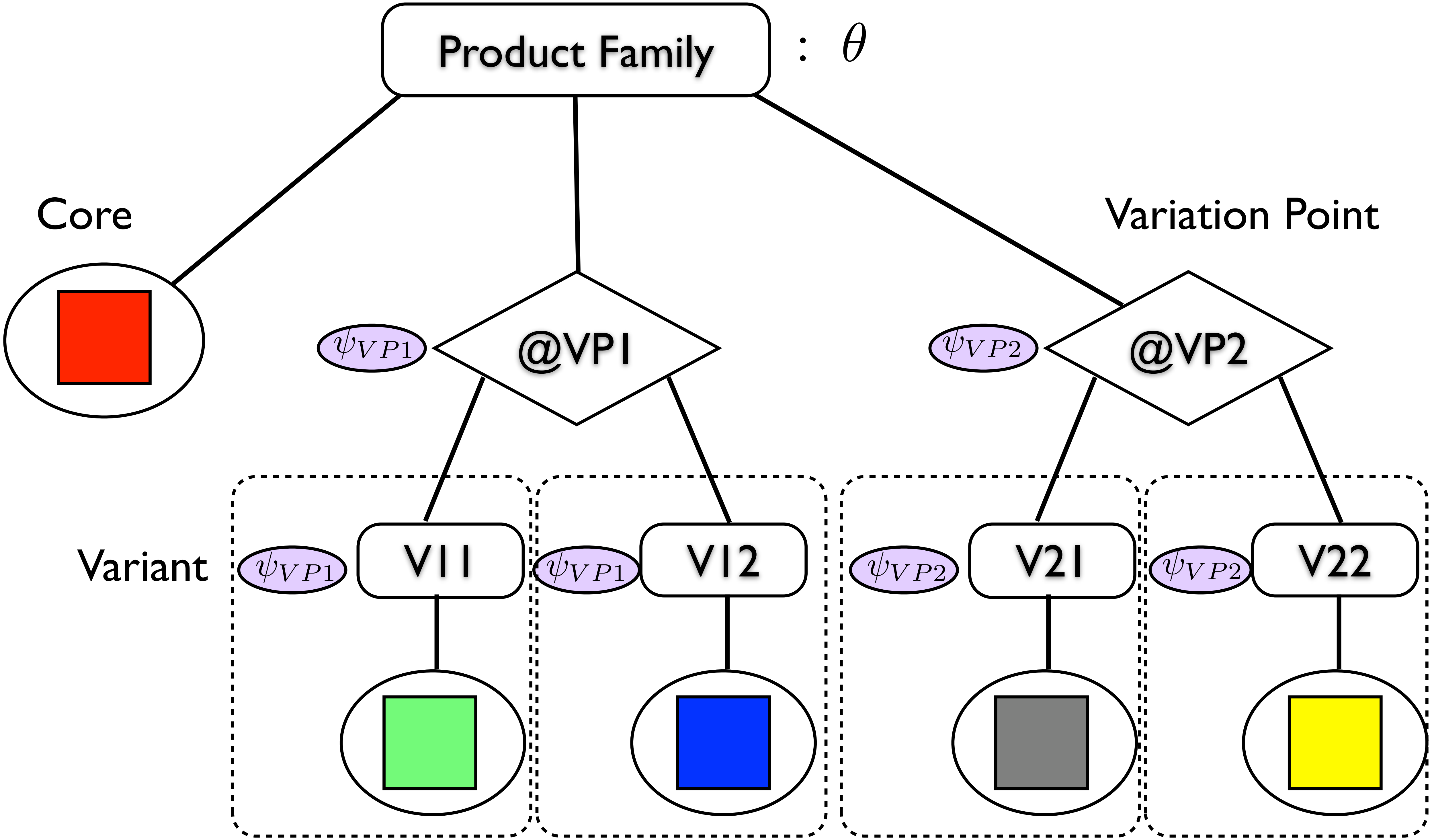
Hierarchical Variability



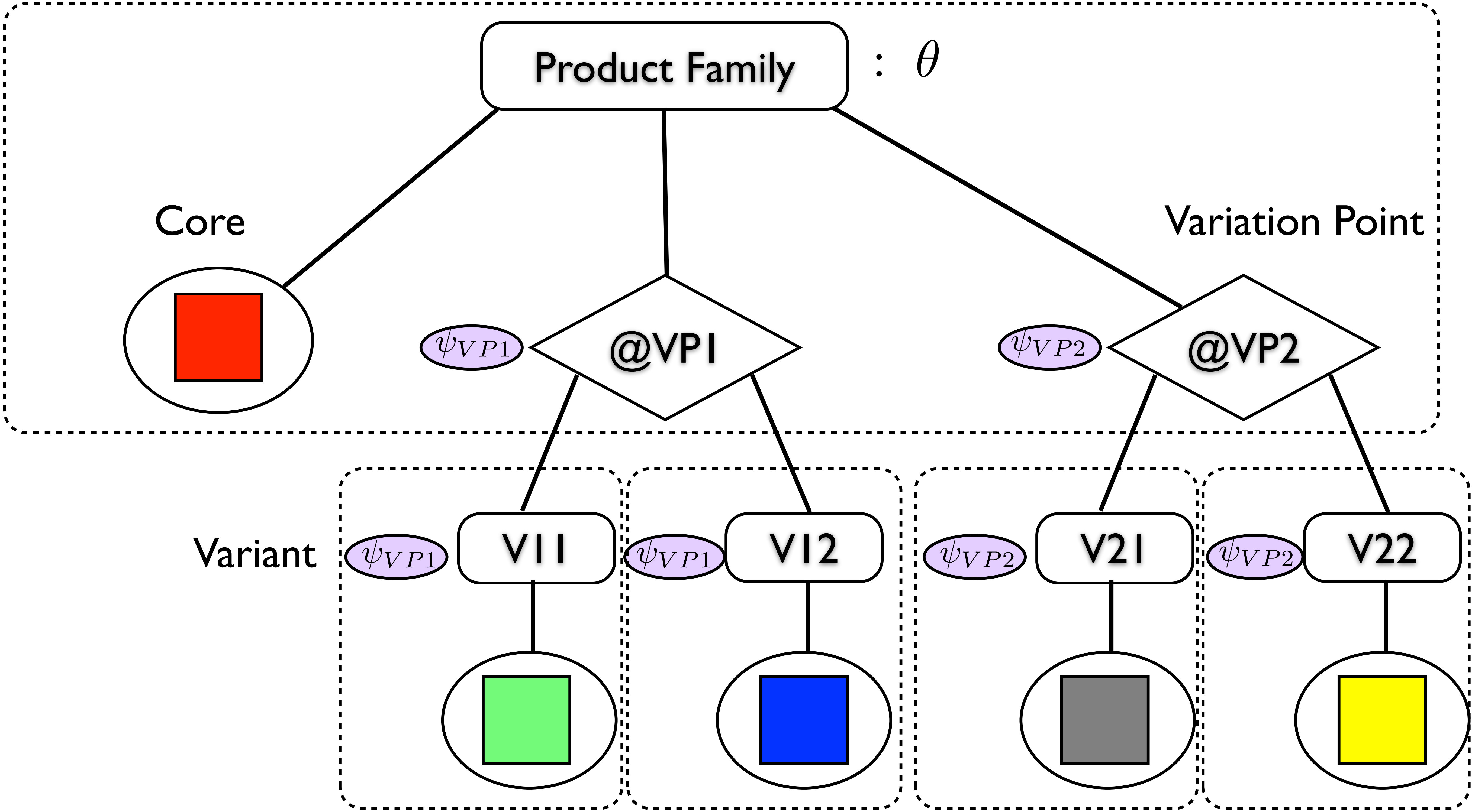
Hierarchical Variability



Hierarchical Variability



Hierarchical Variability



Verification of Product Families

Verification of Product Families

Defining variability models

Verification of Product Families

Defining variability models

Construction procedure

Verification of Product Families

Defining variability models

Construction procedure

Automation of the construction

Verification of Product Families

Defining variability models

Construction procedure

Automation of the construction

Verification procedure

Verification of Product Families

Defining variability models

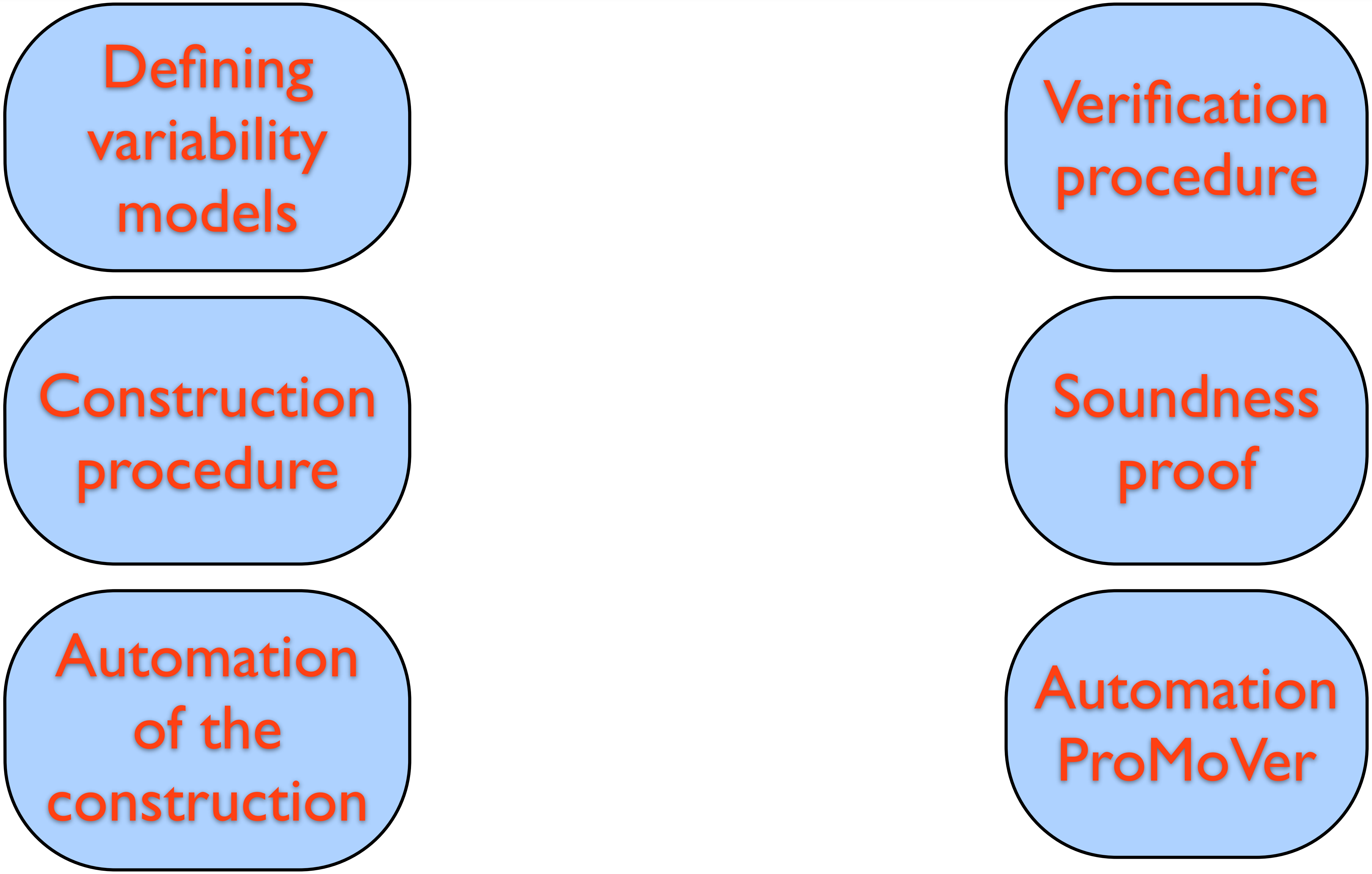
Construction procedure

Automation of the construction

Verification procedure

Soundness proof

Verification of Product Families



Defining variability models

Construction procedure

Automation of the construction

Verification procedure

Soundness proof

Automation ProMoVer

Verification of Product Families

Defining variability models

Construction procedure

Automation of the construction

Verification procedure

Soundness proof

Automation ProMoVer

Case study

Case Studies

Case Studies

Application	Depth	Products	non-comp. Time	comp. Time
Cash Desk	1	9	79 sec	9 sec
Cash Desk with Coupons	1	18	117 sec	10 sec
Cash Desk with Cards	2	27	278 sec	11 sec
Cash Desk with Cards & Coupon	2	54	652 sec	12 sec

Summary

Summary

- Generic framework for modular verification

Summary

- Generic framework for modular verification
- Modularity allows dealing with variability

Summary

- Generic framework for modular verification
- Modularity allows dealing with variability
- Instantiation of the framework

Summary

- Generic framework for modular verification
- Modularity allows dealing with variability
- Instantiation of the framework
    - models without data

Summary

- Generic framework for modular verification
- Modularity allows dealing with variability
- Instantiation of the framework
  - models without data
    - ✶tools and wrappers for various variability scenarios

Summary

- Generic framework for modular verification
- Modularity allows dealing with variability
- Instantiation of the framework
    - models without data
        - ✴tools and wrappers for various variability scenarios
        - ✴case studies and experiments

Summary

- Generic framework for modular verification
- Modularity allows dealing with variability
- Instantiation of the framework
    - models without data
        - ✶tools and wrappers for various variability scenarios
        - ✶case studies and experiments
    - Pointer programs

Summary

- Generic framework for modular verification
- Modularity allows dealing with variability
- Instantiation of the framework
  - models without data
    - ✶ tools and wrappers for various variability scenarios
    - ✶ case studies and experiments
  - Pointer programs
- Efficient verification of product families