

# Static Conformance Checking of Runtime Architectures – Tool Demonstration

---

Marwan Abi-Antoun     Jonathan Aldrich  
School of Computer Science  
Carnegie Mellon University



# The problem: architectural conformance

---

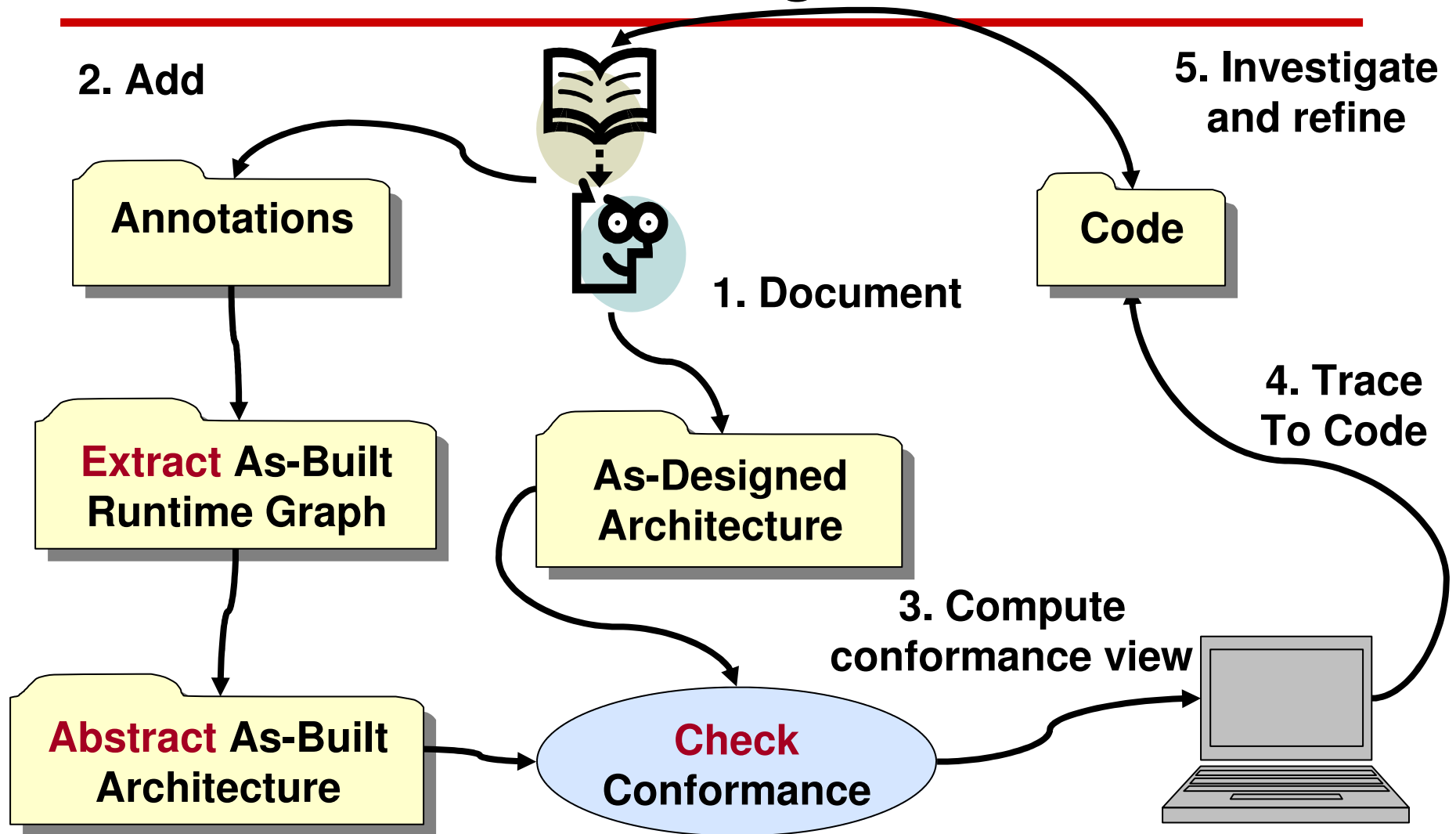
- Architects think in terms of **as-designed architecture**
- Developers implement and evolve code, i.e., **as-built architecture**
- How to **check conformance** between **as-built** and **as-designed** architectures?
  - Intuitive definition: two components communicate only when the architecture allows them to do so
- Architectural **violations** could be serious defects, e.g., lead to security breaches

# This tool demonstration

---

Tools to support a **semi-automated** approach to **statically check** a system's **structural conformance** to an as-designed **runtime architecture**

# Conformance Checking Process



# Key aspects of our approach

---

- Focus on **runtime architecture**
- Models runtime entities and their interactions
  - Influences **quality attributes**, e.g., **security**, reliability
  - a.k.a. Component-and-Connector (C&C) view
- Component: unit of computation and state
  - an object or a group of objects in O-O system
- Connector: abstraction of runtime interaction
  - E.g., field reference or method call in O-O system
- Complements **code architecture**
  - UML class diagram
  - Deals with quality attributes like **maintainability**

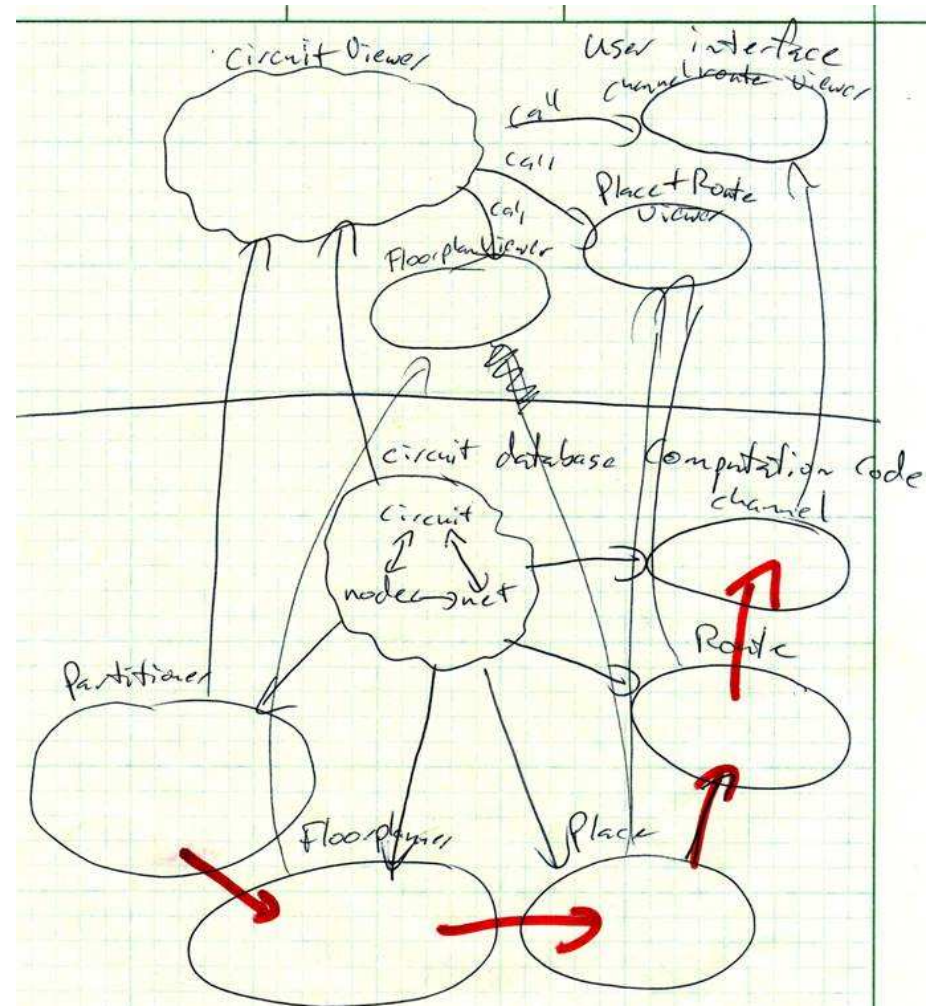
# Key aspects of our approach

---

- Handle **existing languages** and designs
  - No radical language extensions
  - E.g., ArchJava specifies components in code
  - **Annotations OK**
- Use **static analyses**
  - Dynamic analysis cannot prove program always satisfies particular property
  - Must be **sound**, i.e., reveal all entities and relations that could possibly exist at runtime

# Running Example: Aphyds

- 8-KLOC Java system
- As-designed architecture by original developer
- Two-tiered system
- Hierarchical decomposition



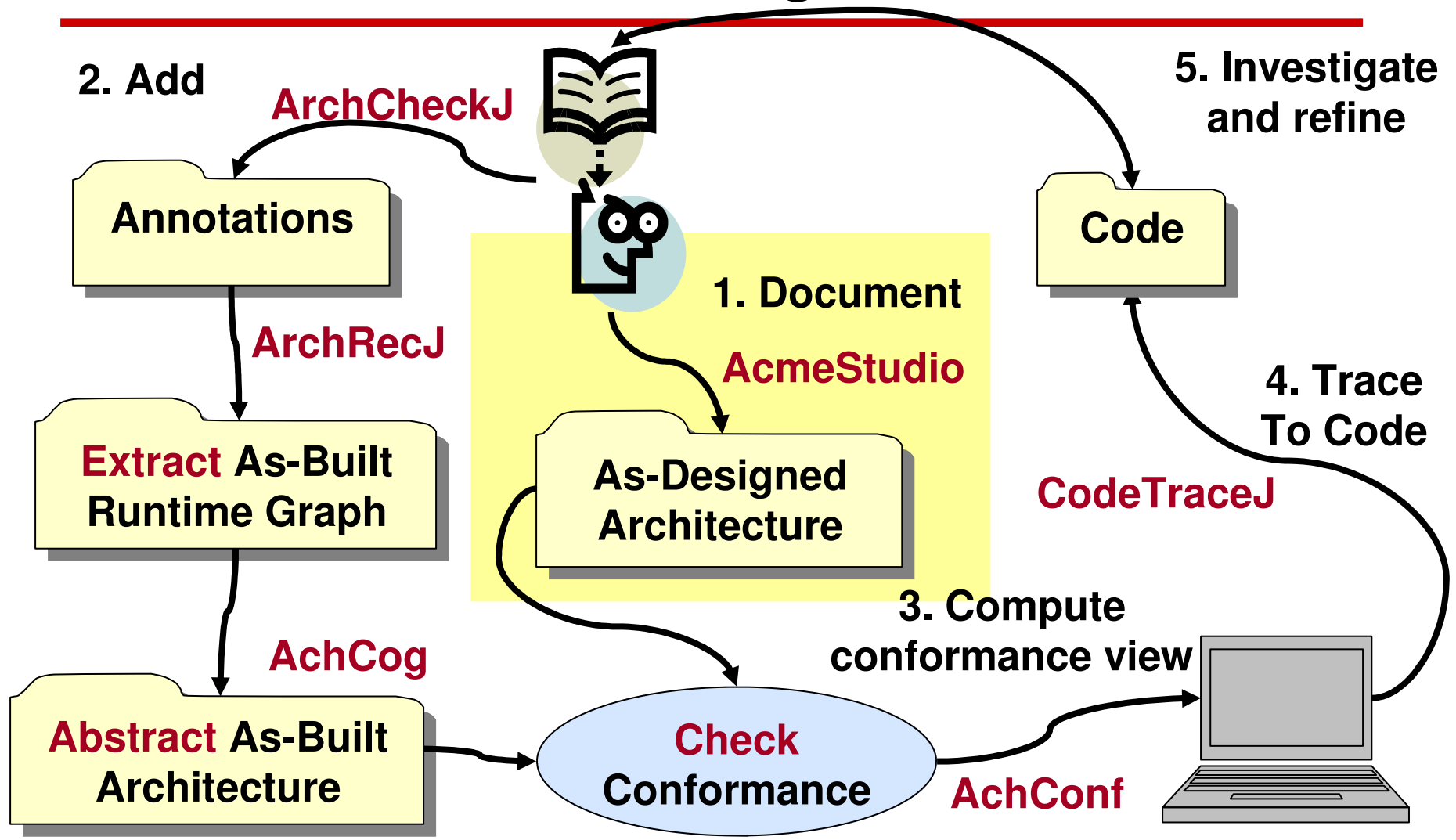
# Check conformance using the strategy

## Extract-Abstract-Check

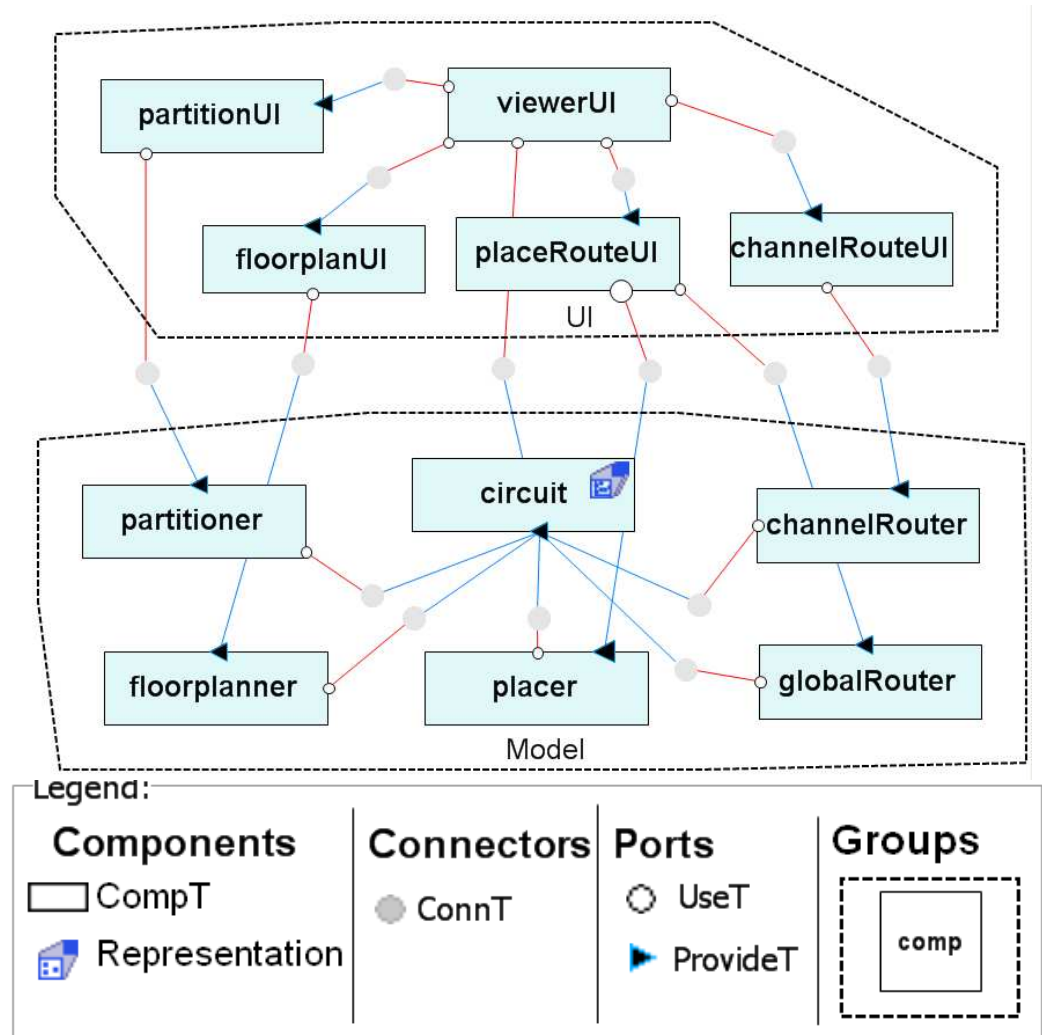
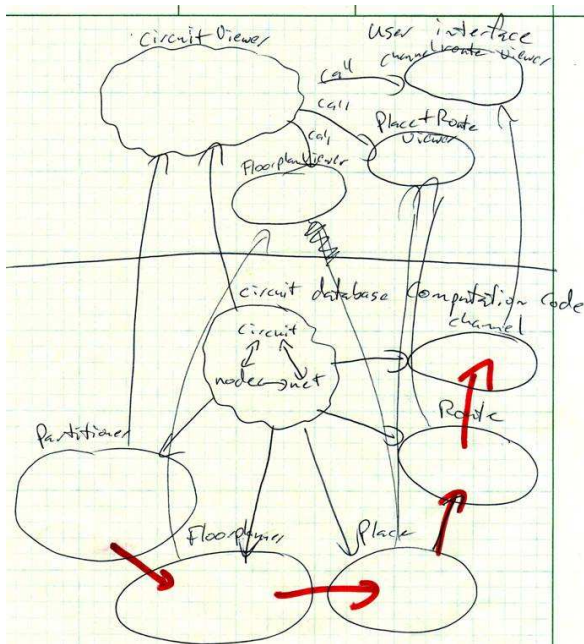
---

1. Document **as-designed** architecture
2. Abstract **as-built** architecture from code
  - Add annotations to code
  - **Extract** instance structure
  - **Abstract** into **as-built architecture**
3. **Check** conformance
  - **Compare** as-built and as-designed
  - Display results graphically
  - Trace finding to code

# Conformance Checking Process



# AcmeStudio: Document as-designed architecture

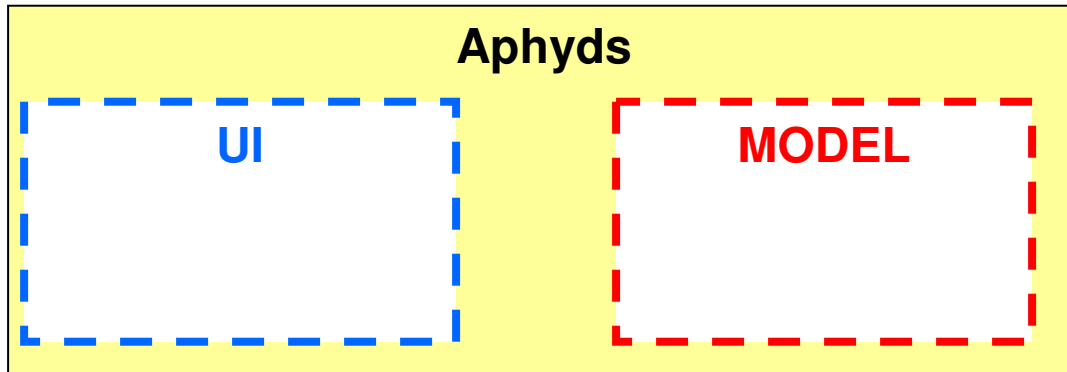


# Extract as-built architecture

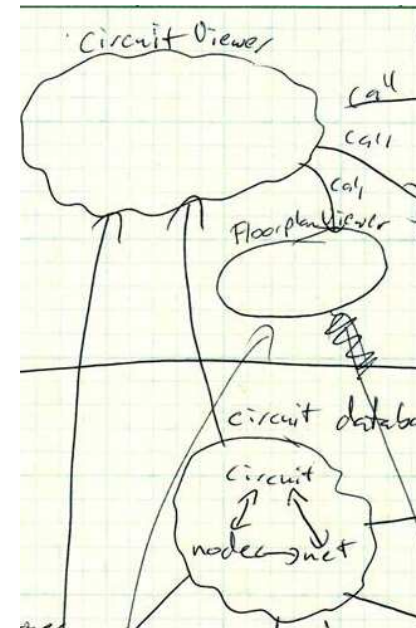
---

- Add **annotations** to code
  - Not discussed here in depth
  - See **related tool demonstration**
  - Currently, annotations done manually
  - Room for future automation
- Extract **hierarchical** runtime structure
  - See **related tool demonstration**

# Extracting runtime structure using ...



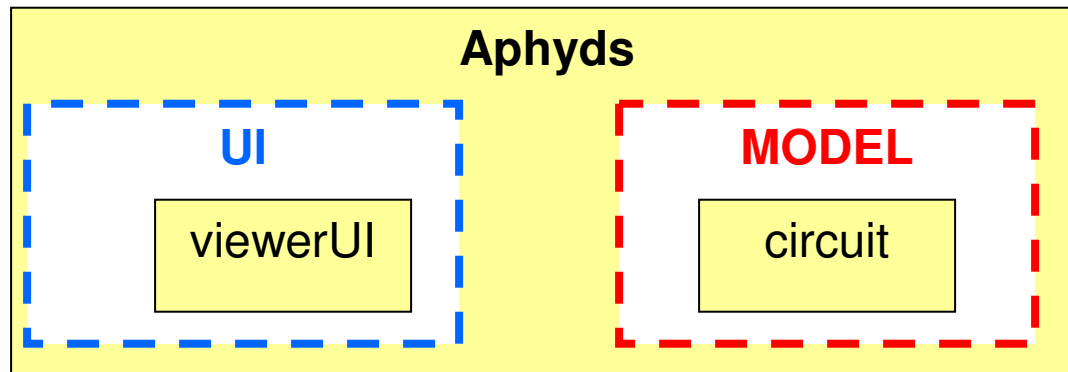
```
class Aphyds {  
    domain UI, MODEL;  
    ...  
}
```



*Declarations  
are simplified*

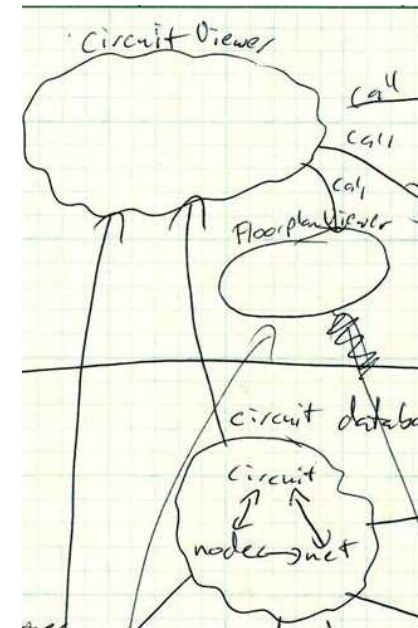
**Ownership domain = conceptual group of objects**

# ... ownership domain annotations



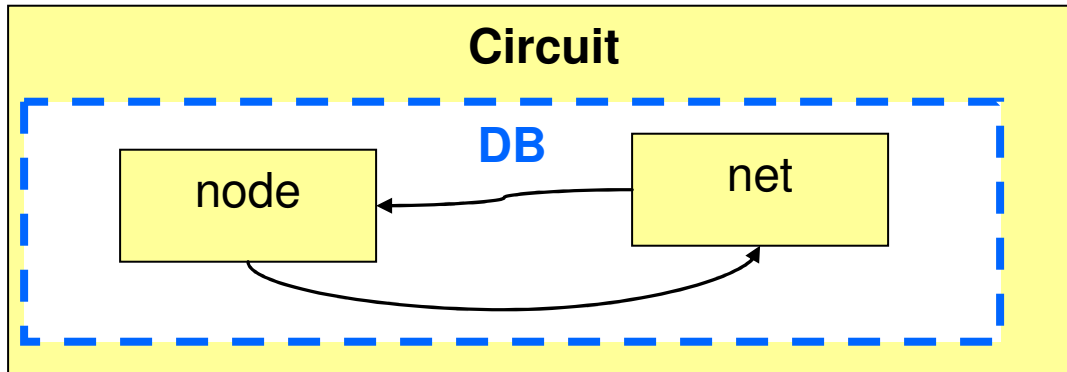
```
class Aphyds {  
    domain UI, MODEL;  
  
    UI viewer viewerUI;  
    MODEL Circuit circuit;  
    ...  
}
```

**Domains can be defined at the top-level**



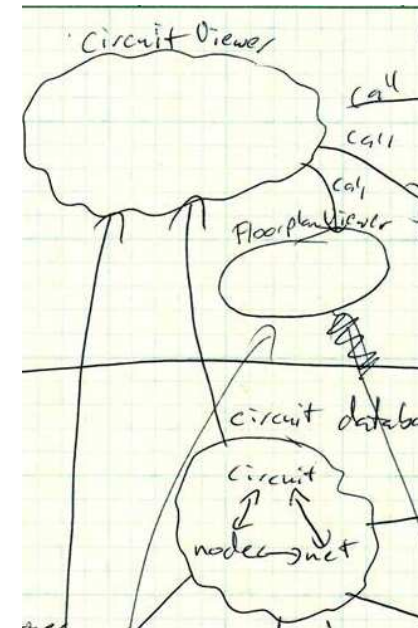
*Declarations  
are simplified*

# Representing system decomposition



```
class Circuit {  
  domain DB;  
  
  DB Node node;  
  DB Net net;  
  ...  
}
```

**Domains can be declared inside each object**



*Declarations are simplified*

# Why use annotations?

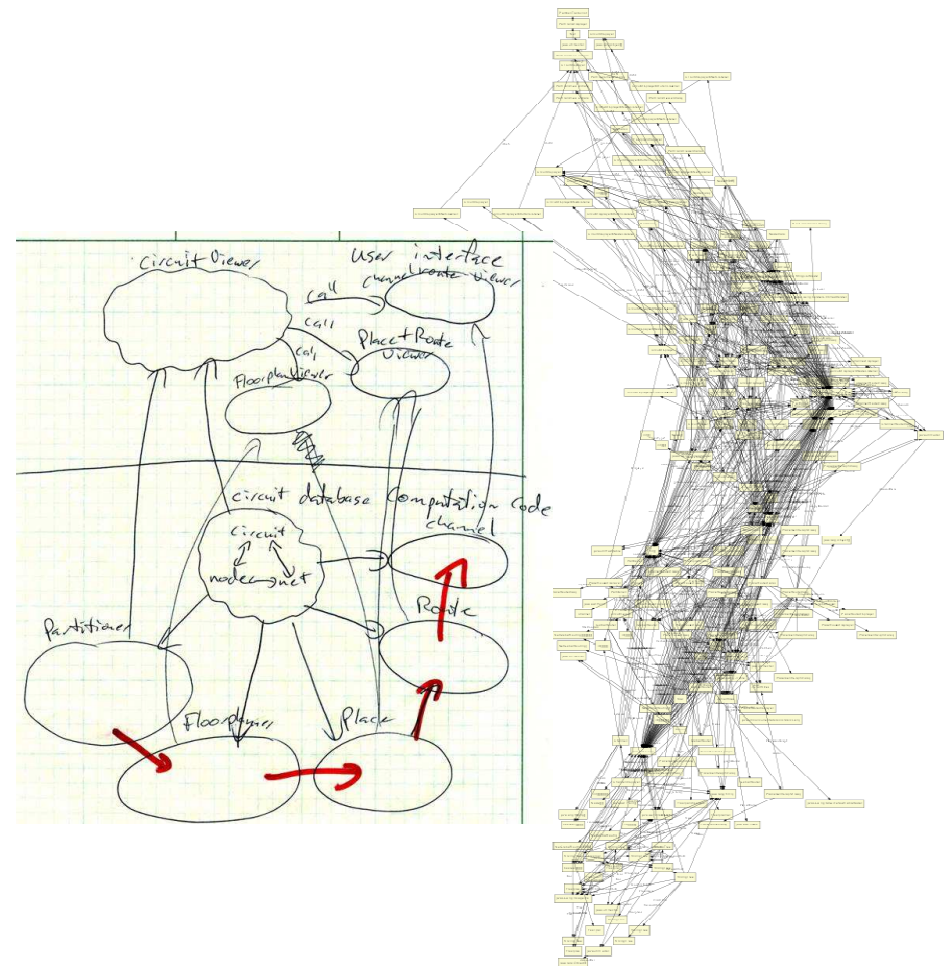
---

- Annotations specify in code
  - **object encapsulation**
  - **logical containment**
  - **tiers**
- Not explicit constructs in general purpose programming languages
- Avoid extracting abstractions that architects do not recognize
- Make **as-built** architecture **comparable** to **as-designed** architecture

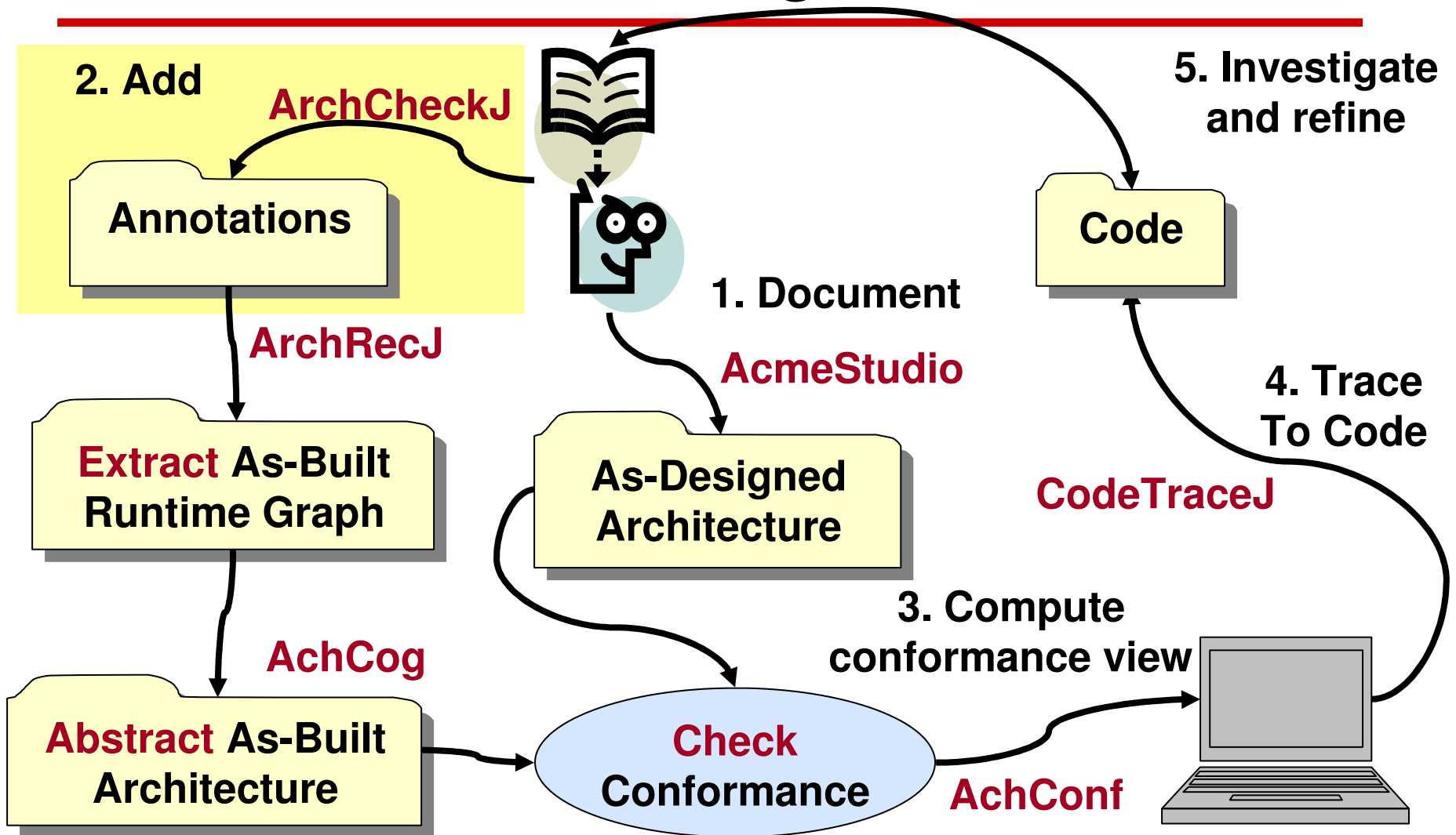
# Aphyds object graph without annotations

Using Womble tool [Jackson and Waingold, TSE 2001]

- Non-hierarchical object graph
- No architectural abstraction
  - Low-level objects mixed in with important objects
  - Cannot easily tell them apart
- Same runtime object may appear as **multiple** components



# Conformance Checking Process



# ArchCheckJ: Check annotations

---

- Add Java 1.5 annotations
- Check ownership domain annotations

# ArchCheckJ

The screenshot shows the Eclipse IDE with the ArchCheckJ plugin. The Package Explorer on the left shows a project structure with various Java files. The Main.java file is open in the editor, showing the following code:

```
@import edu.cmu.cs.aliasjava.annotations.Domain;

@Domains( { "MODEL", "UI" })
public class Main {

    @Domain("UI<UI,MODEL>")Viewer circuitViewer = new Viewer();

    public Main() {
    }

    public void run() {
        try {
            // Add the following code if you want the Look and Feel
            // to be set to the Look and Feel of the native system.
            /*
             * try { UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName()); } catch (Exception e) { }
             */

            // Create a new instance of our application's frame, and make it visible.

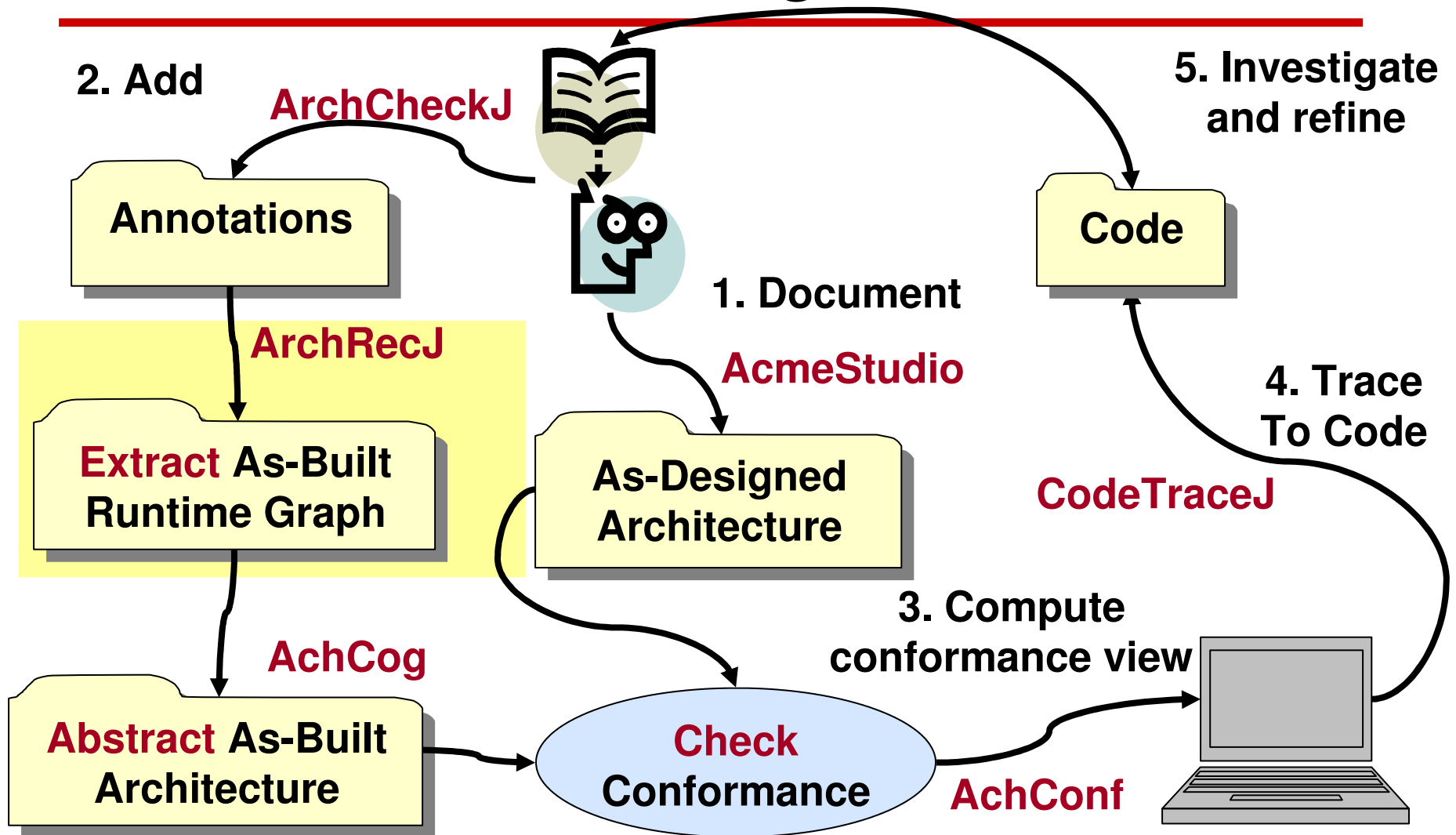
            circuitViewer.setVisible(true);
        }
        catch (@Domain("unique") Throwable t) {
            t.printStackTrace();
            // Ensure the application exits with an error condition.
            System.exit(1);
        }
    }

    public static void main(@Domain("lent[shared]")String args[]) {
        @Domain("lent")Main system = new Main();
        system.run();
    }
}
```

The Problems view at the bottom shows 64 items, all of which are "Problem" type. The first few items are:

Description	Resource	Path	Locat...	Type
Alias annotation does not match expected annotation lent at assignment return this;	FloorplanUI...	Aphyds_PublicDo...	line 82	Problem
Alias annotation circuitdb does not match expected annotation circuit.DB at assignment targ=circuit.getNet(netname)	CircuitDispla...	Aphyds_PublicDo...	line 205	Problem
Alias annotation circuitdb does not match expected annotation circuit.DB at assignment targ=circuit.getActionCommand()	CircuitDispla...	Aphyds_PublicDo...	line 146	Problem
Alias annotation circuitdb does not match expected annotation circuit.DB at assignment targ=circuit.getNode(nodename)	CircuitDispla...	Aphyds_PublicDo...	line 177	Problem
Alias annotation fpdb does not match expected annotation fp.DB at assignment fp.getBestFloorplan()	FloorplanUI...	Aphyds_PublicDo...	line 108	Problem
Alias annotation fpdb does not match expected annotation fp.DB at assignment fp.getTreeRoot()	FloorplanUI...	Aphyds_PublicDo...	line 100	Problem
Alias annotation fpdb does not match expected annotation null at assignment (SlicingTree)node.getUserObject()	FloorplanUI...	Aphyds_PublicDo...	line 197	Problem
Alias annotation owner does not match expected annotation lent at assignment newDest.Source=this	NetGlobalRo...	Aphyds_PublicDo...	line 492	Problem
Alias annotation ptdb does not match expected annotation partitioner.DB at assignment pt=partitioner.partitionCircuit(pdiag)	PartUI.java	Aphyds_PublicDo...	line 417	Problem
Alias annotation shared does not match expected annotation null at assignment e.getActionCommand()	CircuitDispla...	Aphyds_PublicDo...	line 146	Problem

# Conformance Checking Process

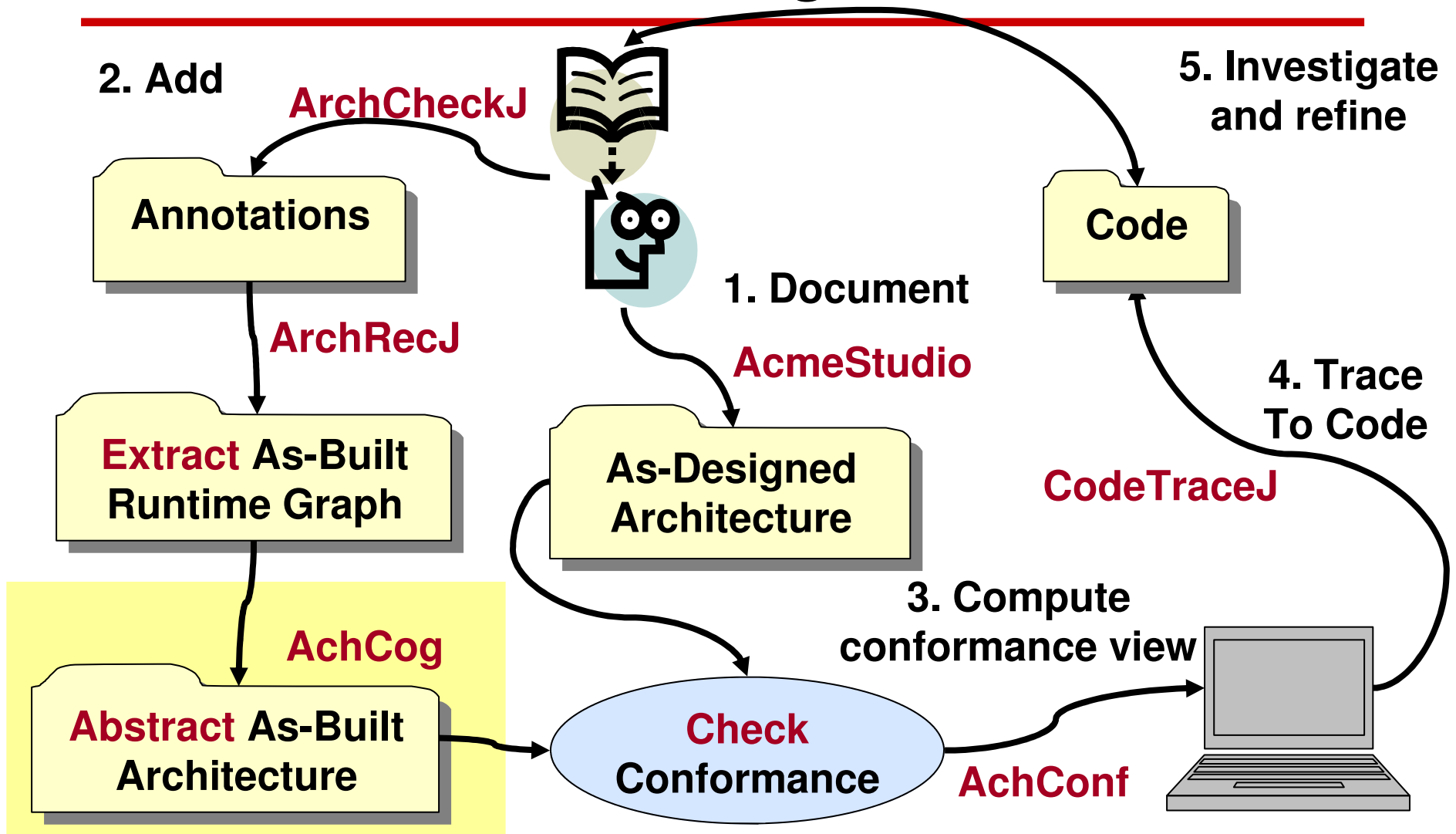


## Step 2.2 Extract **runtime structure**

---

- **Hierarchical representation of runtime object graphs**
  - Show **runtime** entities and their relations
  - *Not* classes, interfaces, inheritance, etc.
- Control abstraction by:
  - ownership hierarchy
  - types

# Conformance Checking Process

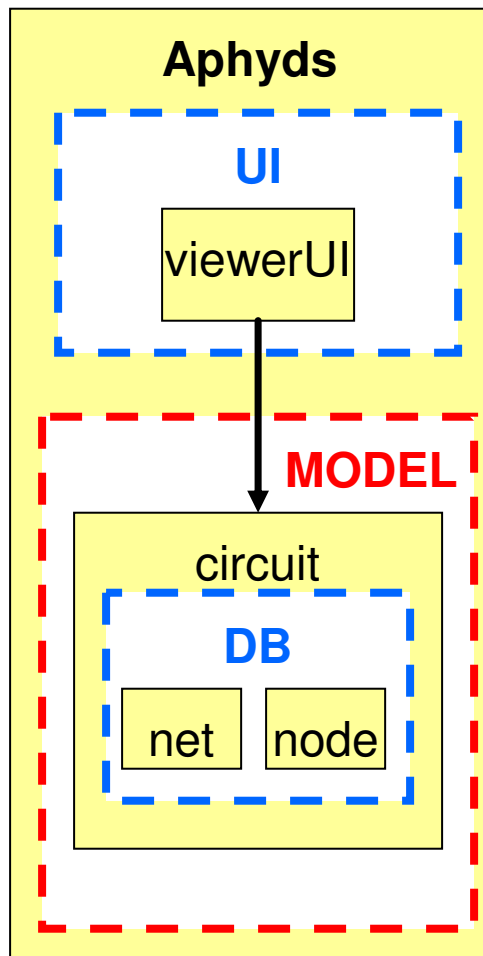


# ArchCog: Abstract OOG into **as-built** arch

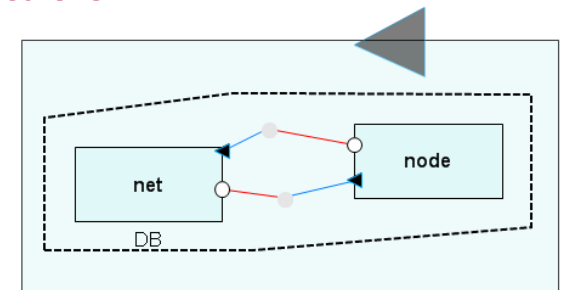
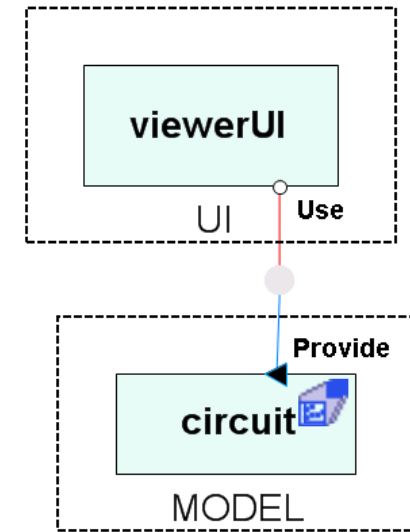
---

- Conversion from OOG to C&C view
  - Base transformation
  - Additional abstraction

# Abstracting OOG into **as-built** architecture: base transformation



- | OOG                | ↔ | C&C view              |
|--------------------|---|-----------------------|
| • Top-level object | ↔ | System                |
| • Object           | ↔ | Component             |
| • Domain           | ↔ | Group                 |
| • Interface        | ↔ | <b>Provide</b> port   |
| • Field reference  | ↔ | <b>Use</b> port       |
| • Object relation  | ↔ | Connector             |
| • Substructure     | ↔ | <b>Representation</b> |



# Abstracting OOG into **as-built** architecture: **additional abstraction**

---

- Control projection depth
- Elide private domains
- Elide single domains
- Add types and properties
- Merge objects

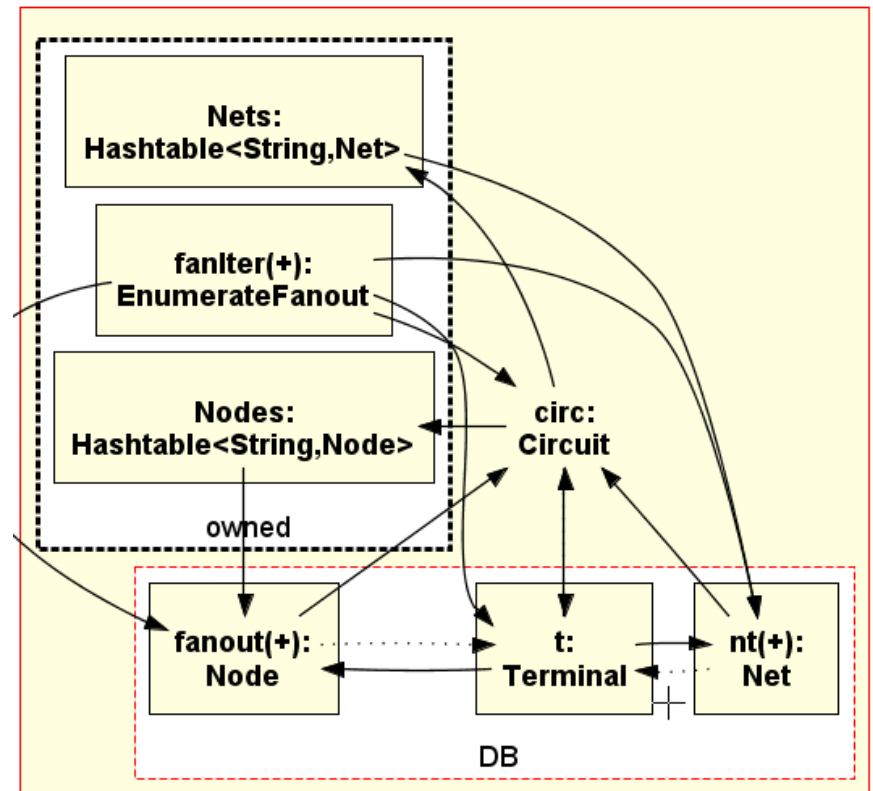
# Control projection depth

---

- Change uniformly across all objects
- Exclude substructure of selected object
- Skip objects beyond a certain depth
  - OOG deep hierarchy
  - As-designed view shallow hierarchy
  - Convert to depth of hierarchical decomposition in as-designed view
  - Speeds up structural comparison

# Elide private domains

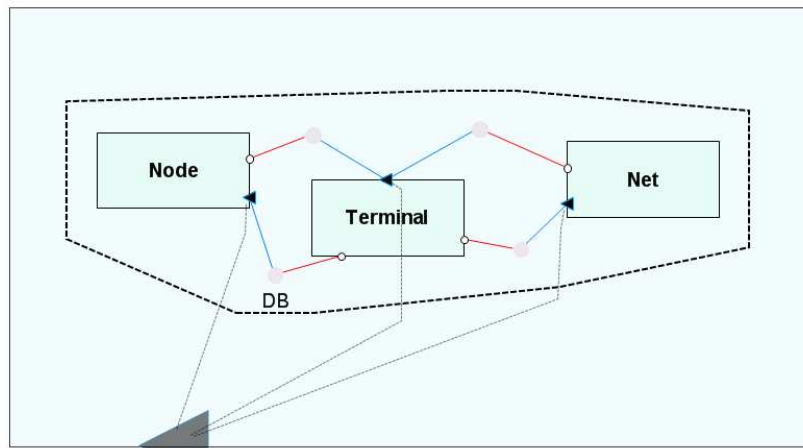
- Private domains hold low-level objects
- Public domains hold externally visible state
- Exclude implementation details at once



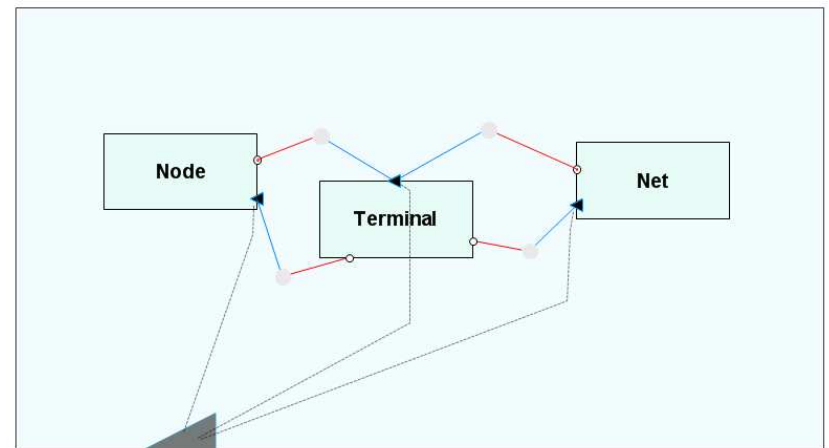
# Elide single domains to match the hierarchical decompositions

---

- In OOG, each object is in a domain
- Systematic conversion would create each Component in a Group
- Architects typically use only top-level tiers



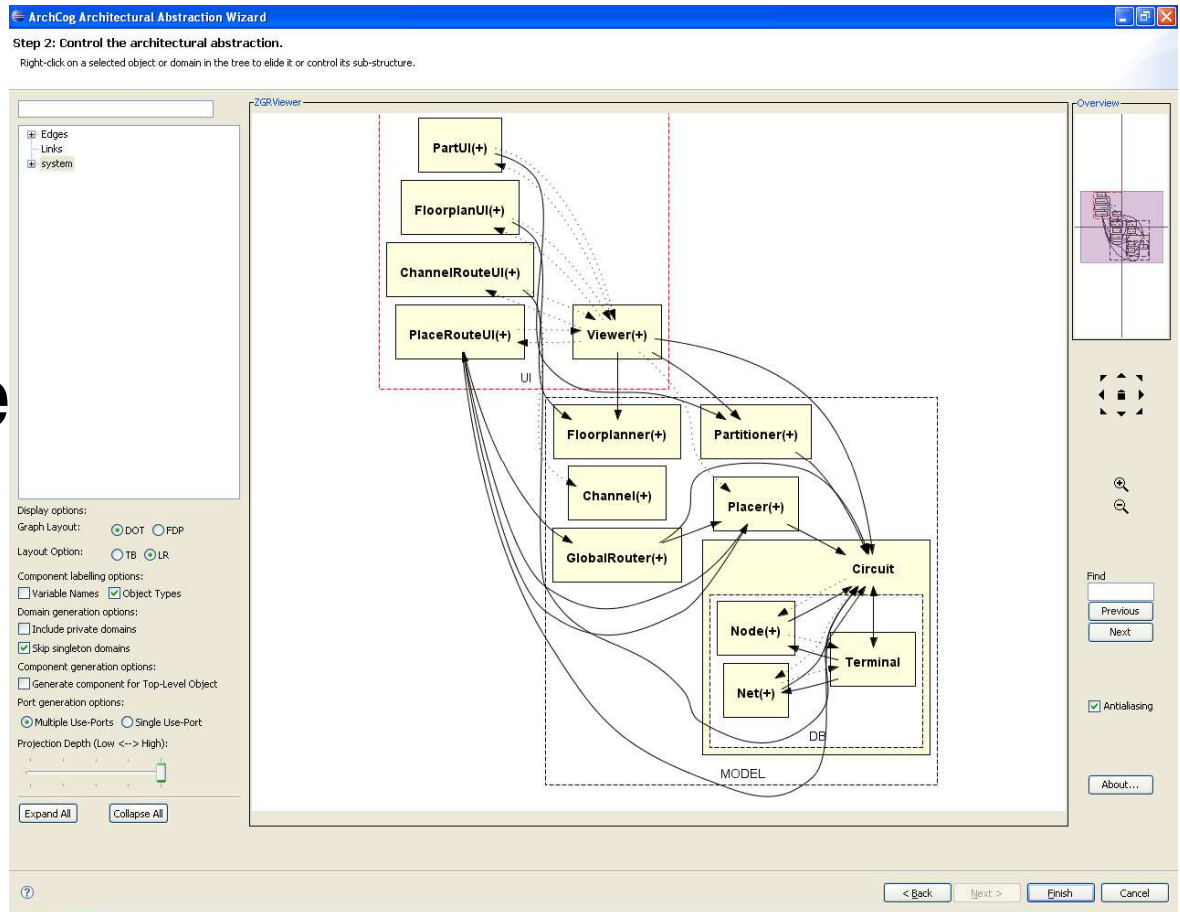
**Not eliding DB**



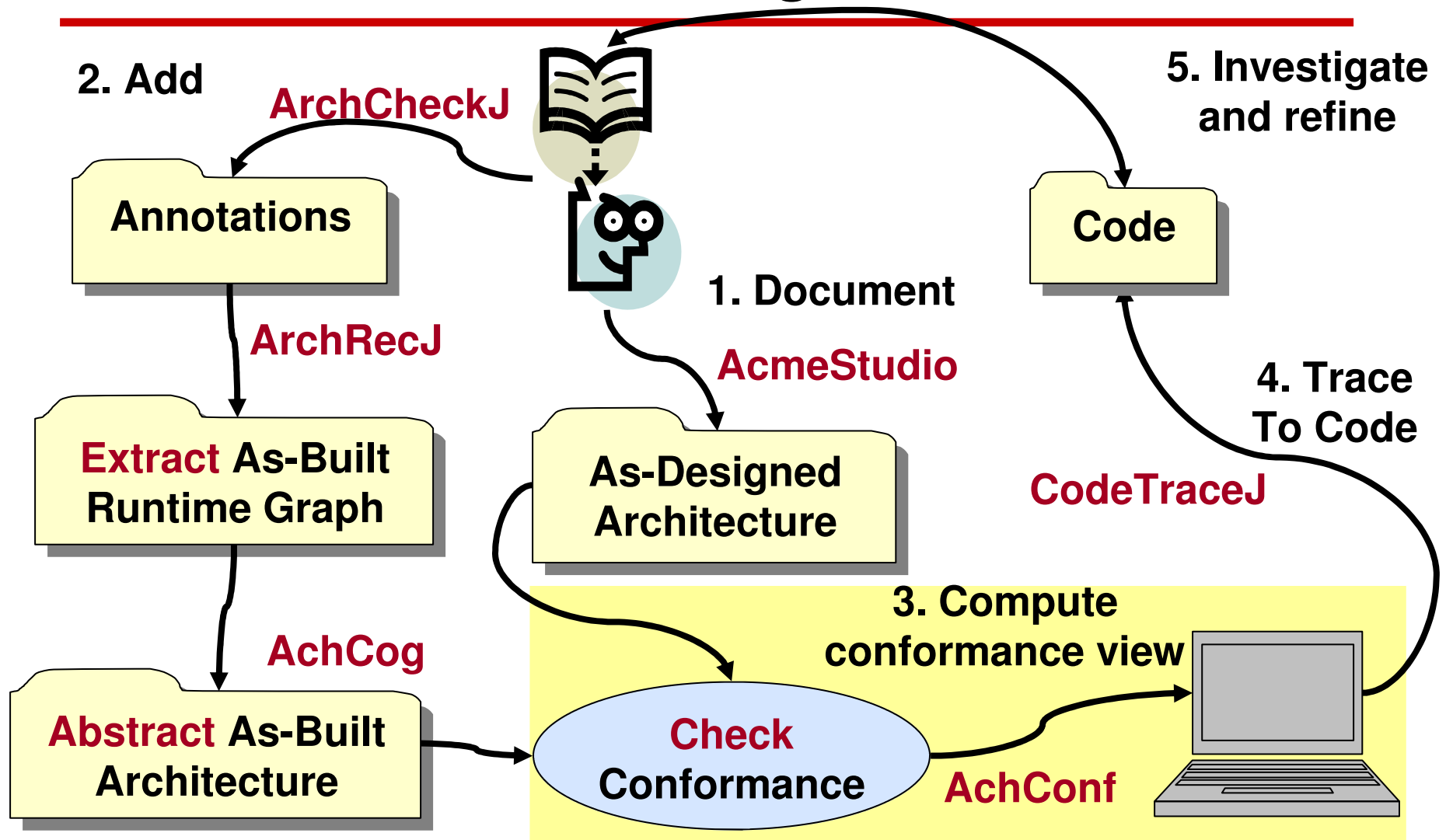
**Eliding DB**

# ArchCog: Abstract **as-built C&C** view

- Control projection depth
- Elide private domains
- Elide single domains



# Conformance Checking Process



# Checking Conformance

---

- ***Definition:*** A system conforms to as-designed architecture if the latter is conservative abstraction of system's runtime structure
- ***Communication integrity:*** each component in the implementation may only communicate directly with the components to which it is connected in the architecture

# Relation to view synchronization

---

- Conformance checking **differs** from view synchronization
  - Goal is **not** to make views **identical**
  - **Extra sub-structure** in as-built architecture
  - **Innocuous differences**, e.g., renames
- **As-designed view** more authoritative
  - Included components more relevant than those omitted
  - Names convey some architectural intent

# Using structural comparison to compare architectures

---

- Does not assume unique node identifiers
- Can detect renames
  - Names cannot be expected to match
  - Treating rename or move as insert/delete
  - Produce structurally equivalent views
  - But **lose properties** associated with elements
- Some limitations:
  - May require forcing some matches **manually**
  - Scales up to thousands of nodes




# Forcing/preventing matches manually

---

- Limitation of structural comparison
  - Node not always matched correctly
  - Manually force matches between nodes
  - Usually happens on small graphs
- Example in Aphyds:
  - Node, Net and Terminal

# As-built vs. as-designed key differences

---

- **Convergence**: node or edge **in both** as-built and in as-designed view 
- **Divergence**: node or edge in as-built, but **not in as-designed** view 
- **Absence**: node or edge in as-designed view, but **not in as-built** view 

# Conformance checking analysis

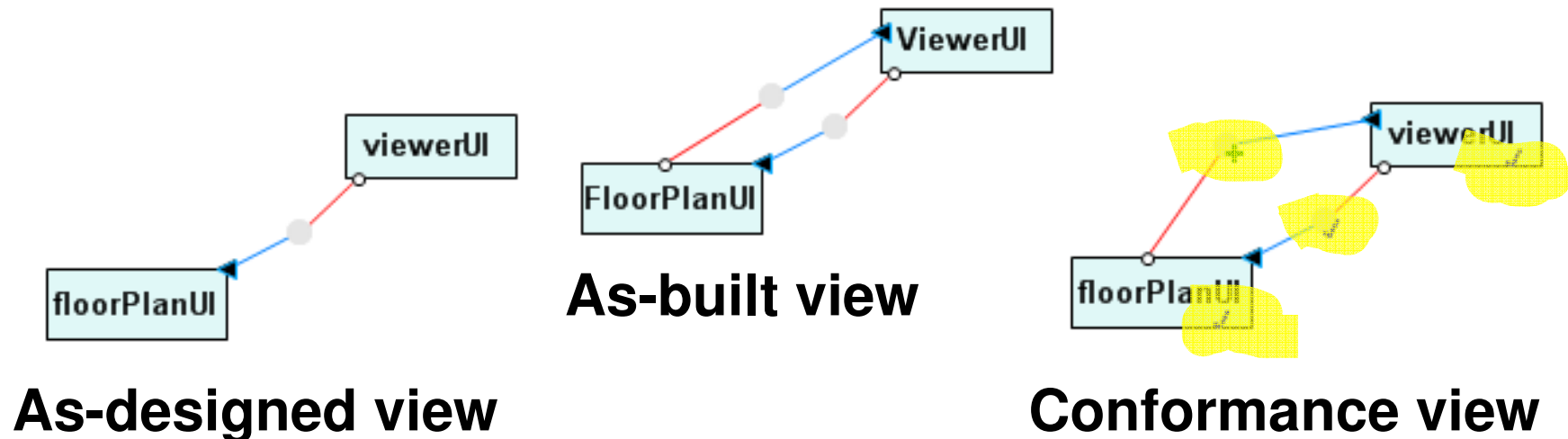
---

- **Highlight differing connections** between as-built and as-designed views
- **Use as-designed view names**
- **Summarize divergent** components without adding them directly
- Check only **matching sub-structures**

# Highlight differing connections

---

- Structurally match components in as-built view to those in as-designed view
- Show differing connections as divergences or absences



# Use as-designed view names

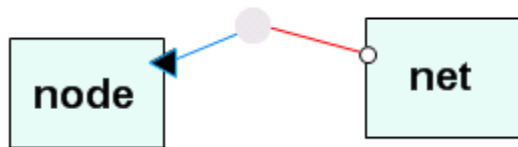
---

- Element **names** in as-designed and as-built views may not match exactly
- **Structural comparison catches renames**
- **Use as-designed view names** to show additional communication between as-built components without renaming them

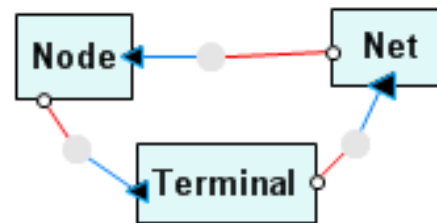
# Summarize divergent components

---

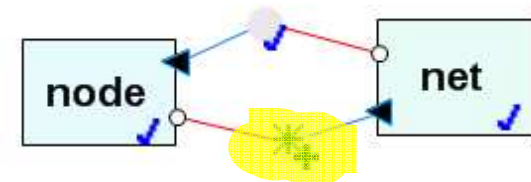
- Avoid cluttering as-designed architecture
- Account for any communication in as-built view **that is not** in as-designed view including communication through divergent components.
- Decorate summary connector with ✱



**As-designed view**



**As-built view**

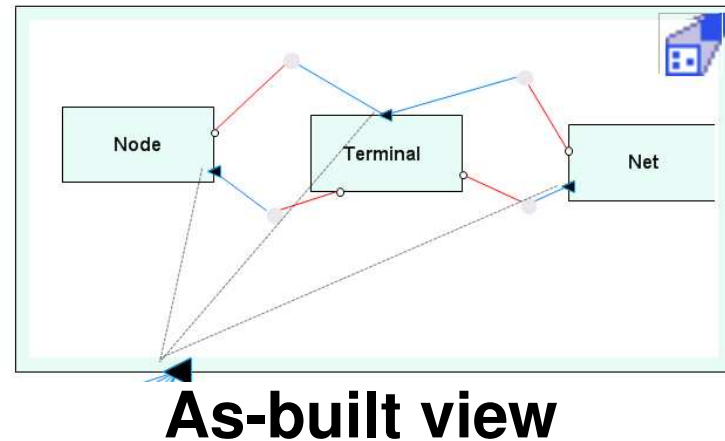
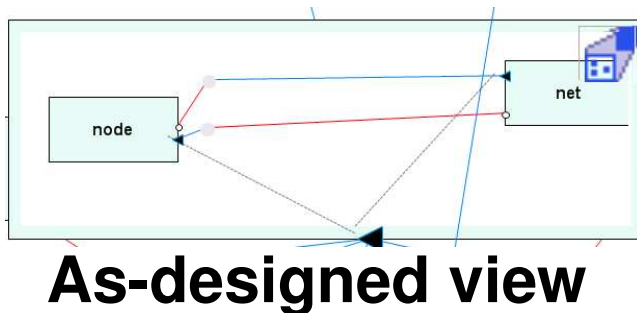


**Conformance view**

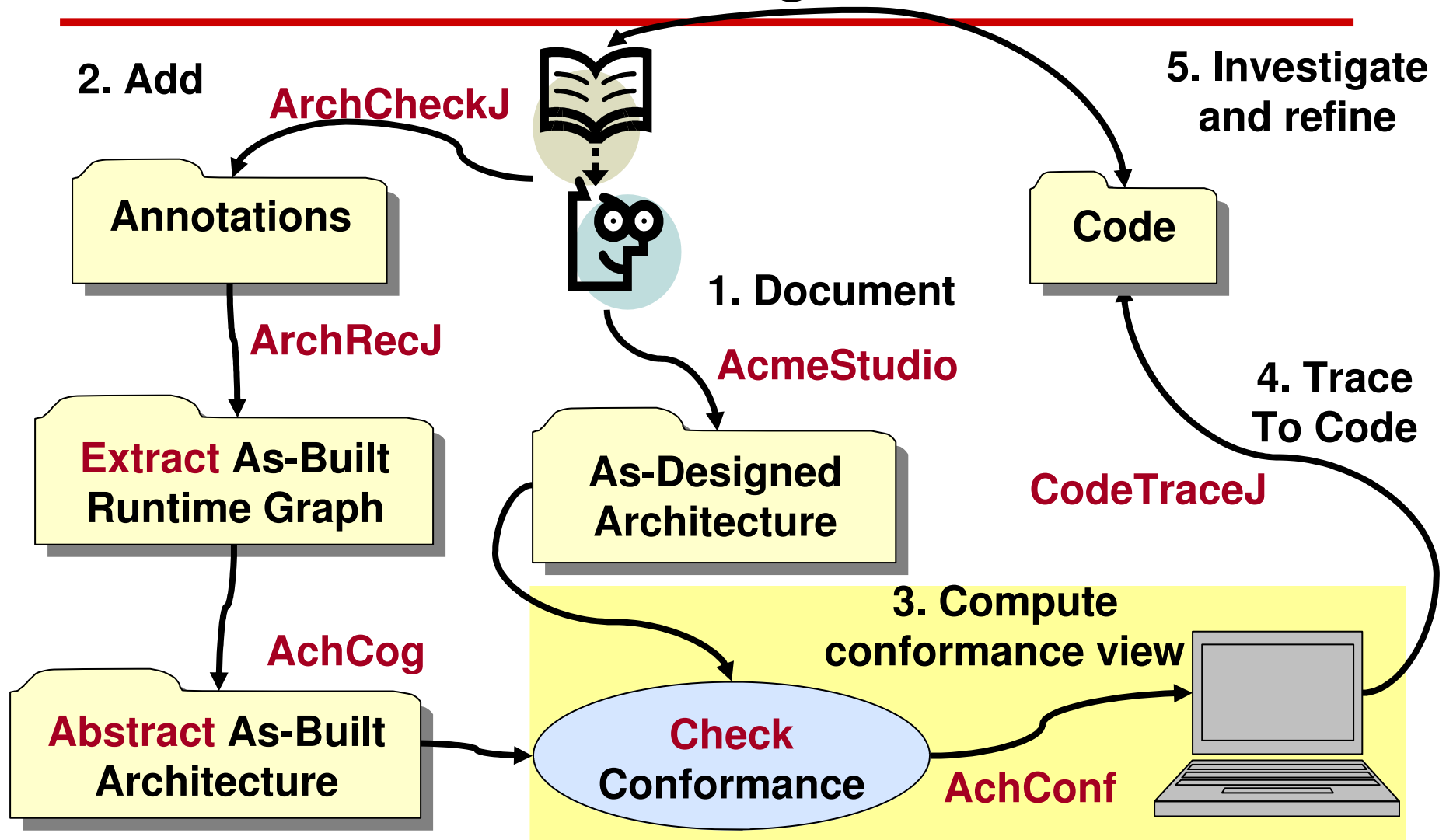
# Check only matching sub-structures

---

- In most cases, as-built and as-designed views have similar depth
- If not, ignore substructure if it exists in **as-built view** but not in the as-designed view, to avoid many false positives



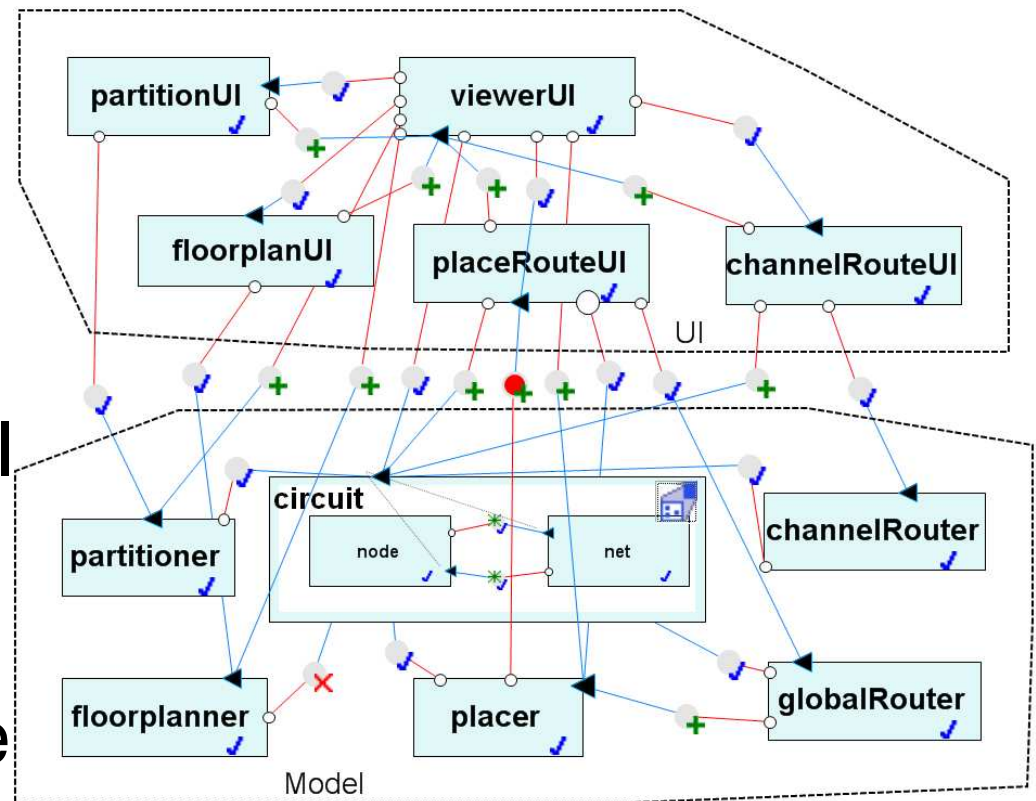
# Conformance Checking Process



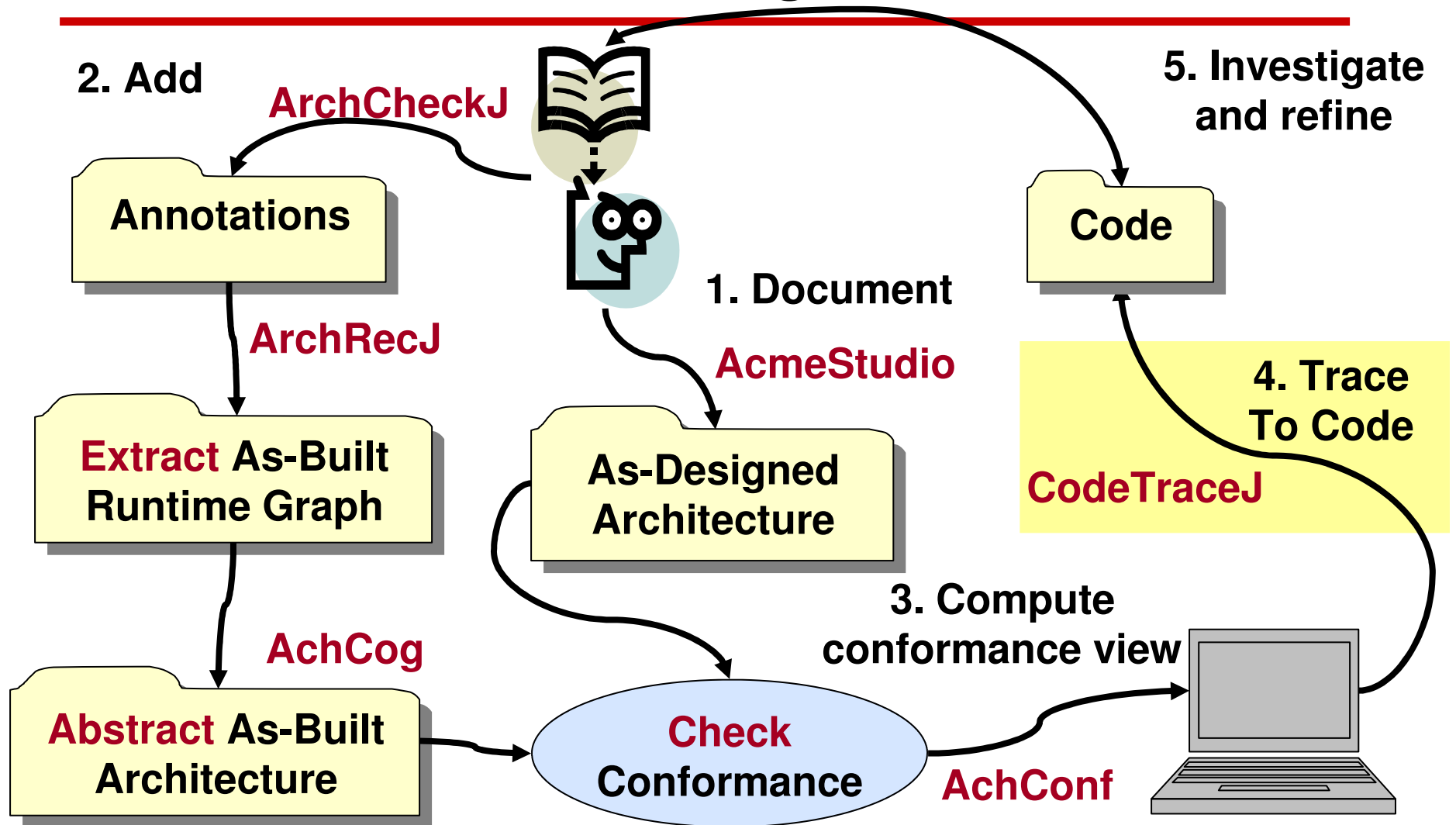


# Aphyds conformance results

- **Missing** top-level component partitionUI
- **Callback** from placer in Model to placeRouteUI in UI
- Many connections thought to be unidirectional were **bi-directional**



# Conformance Checking Process



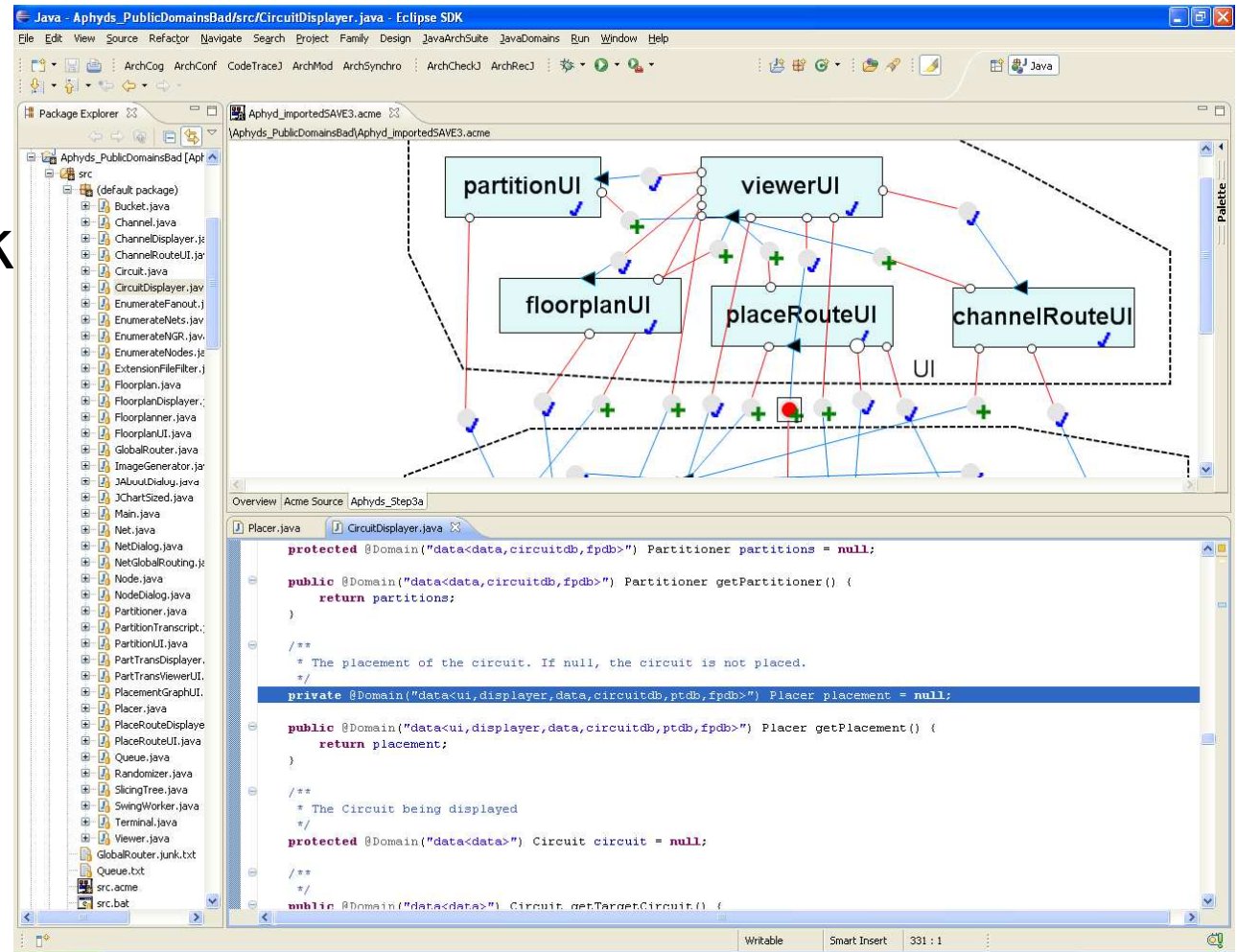
# Relate architectural element to code

---

- **CodeTraceJ** loads element's traceability from architecture:
  - opens corresponding Java files
  - highlights appropriate lines of code
- Analyze conformance finding without potentially reviewing entire code base

# CodeTraceJ: Trace conformance finding to code

- Aphyds
  - Trace callback to code



# Future Work

---

- Tool to convert OOG to C&C view
  - Support more abstraction rules
  - E.g., merge two components by name
  - E.g., map entire domain to component
- Annotation tool support
  - Easier to add annotations to large code bases

# Summary

---

- Approach can find interesting **structural non-conformities** between as-designed and as-built architectures
- Approach provides **positive assurance** that code conforms to architecture