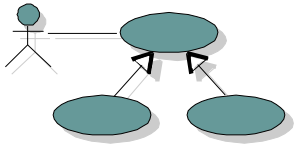
Politecnico
di Milano

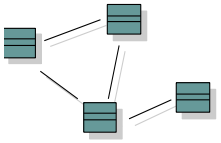
Tips and Tricks for Supporting Architecture Description with UML*

*Kruchten, P.; Selic, B.; Kozaczynski, W.; Larsen, G. & Brown, A. W. (2001), Describing Software Architecture with UML., *in* Hausi A. Müller; Mary Jean Harrold & Wilhelm Schäfer, ed., 'ICSE' , IEEE Computer Society, , pp. 777 .

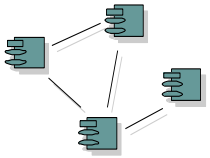
Architecture and Iterations



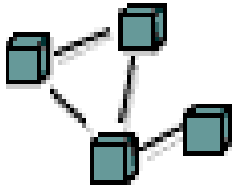
Use case Model



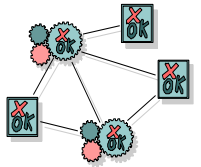
Design Model



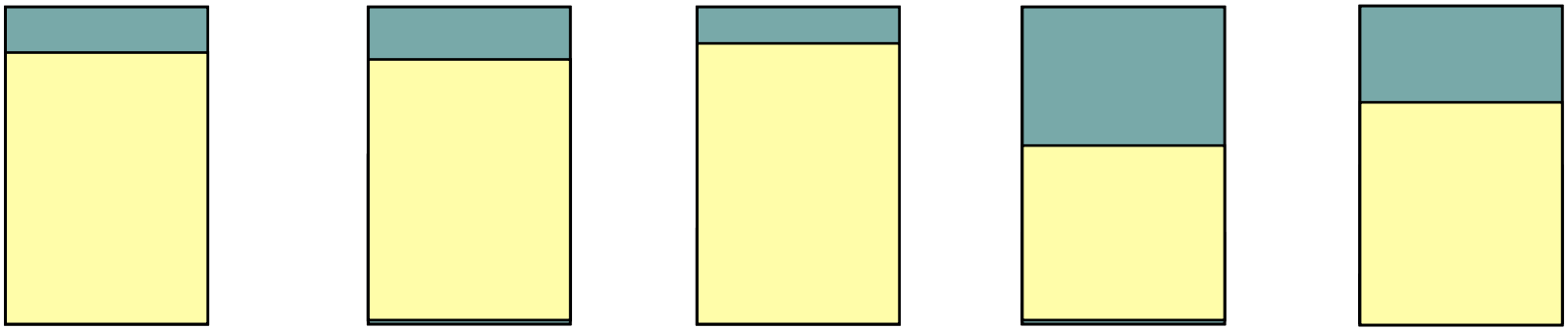
Implementation Model



Deployment Model

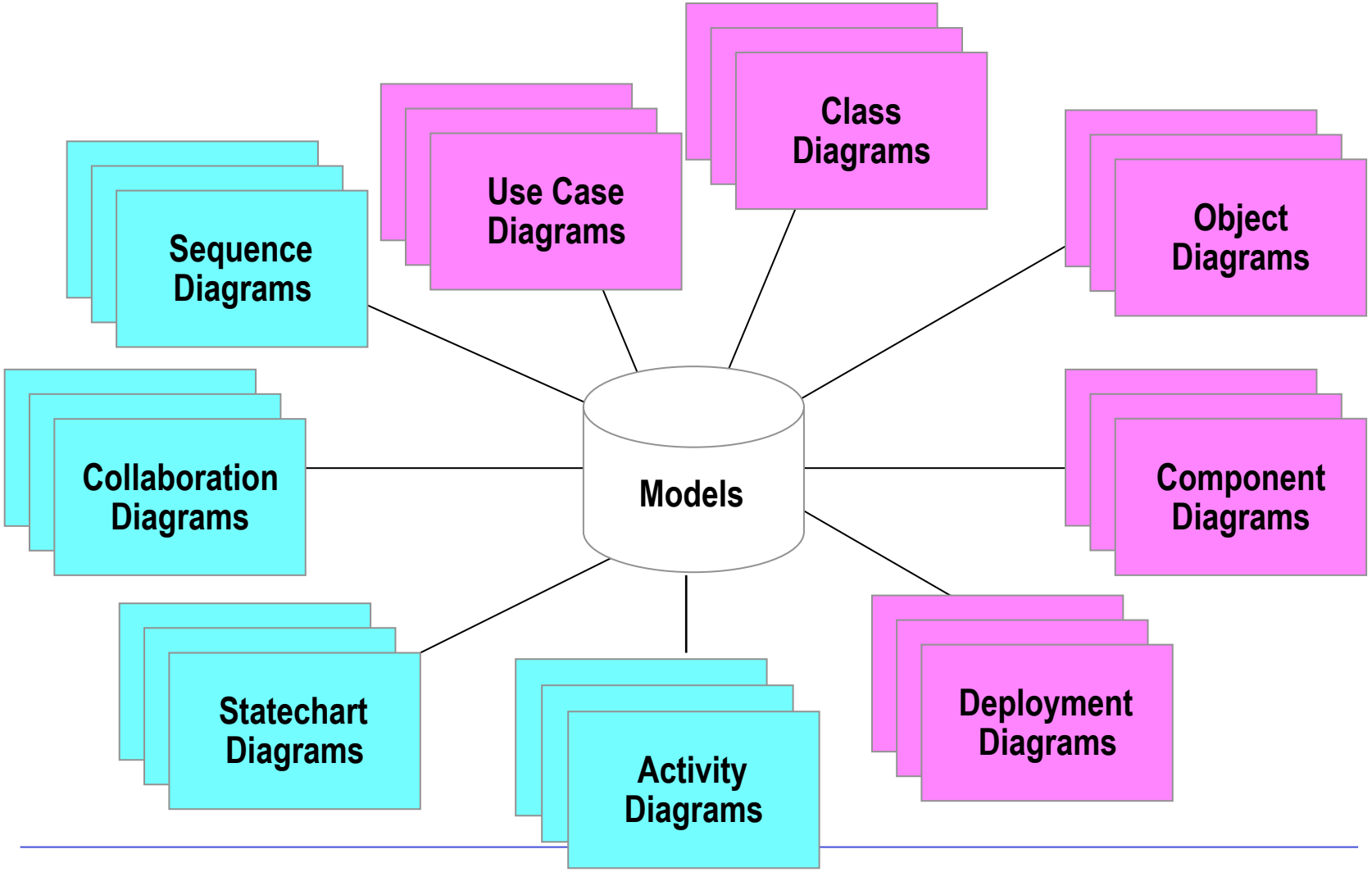


Test Model



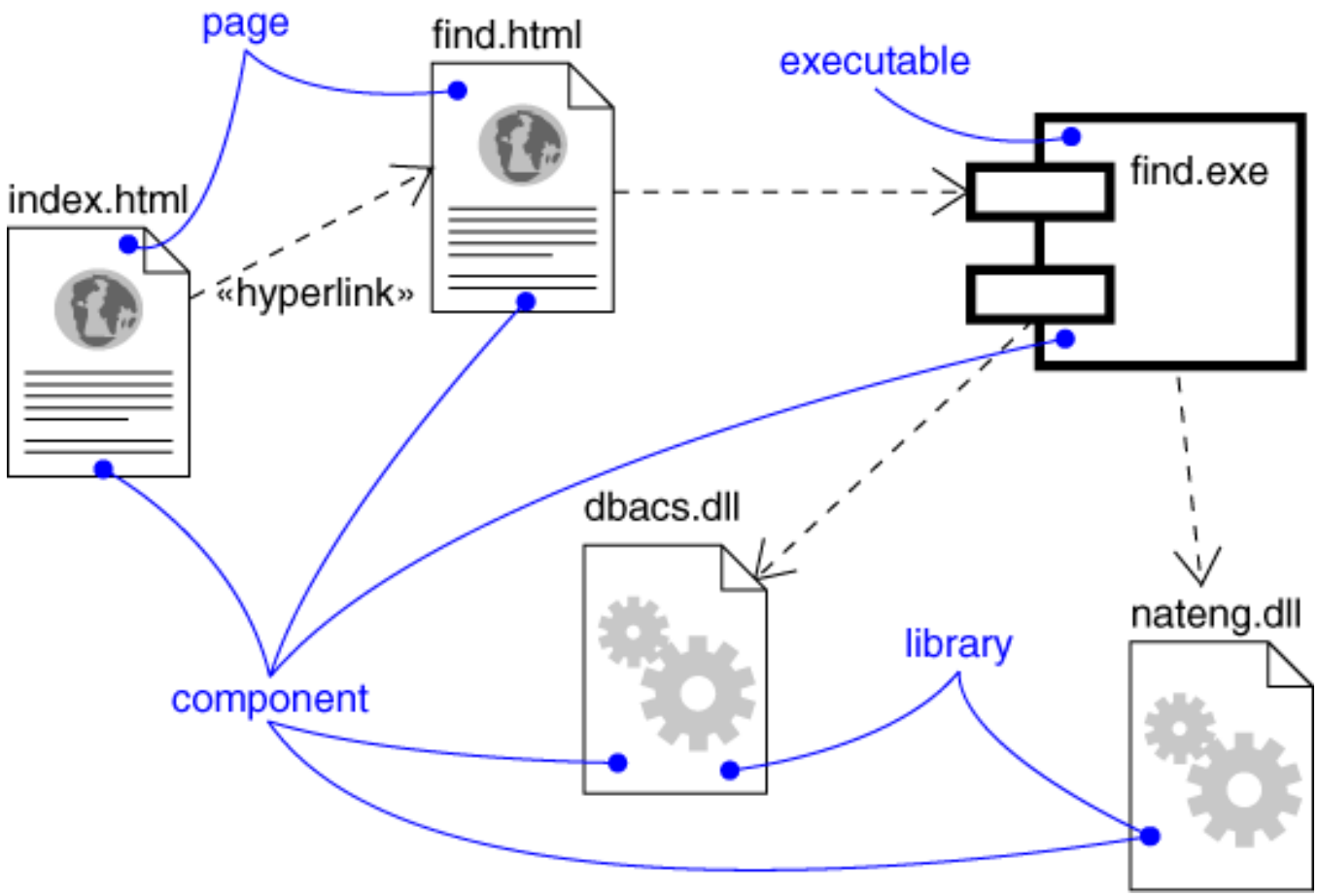
Architectural Content

UML Models, Views, and Diagrams



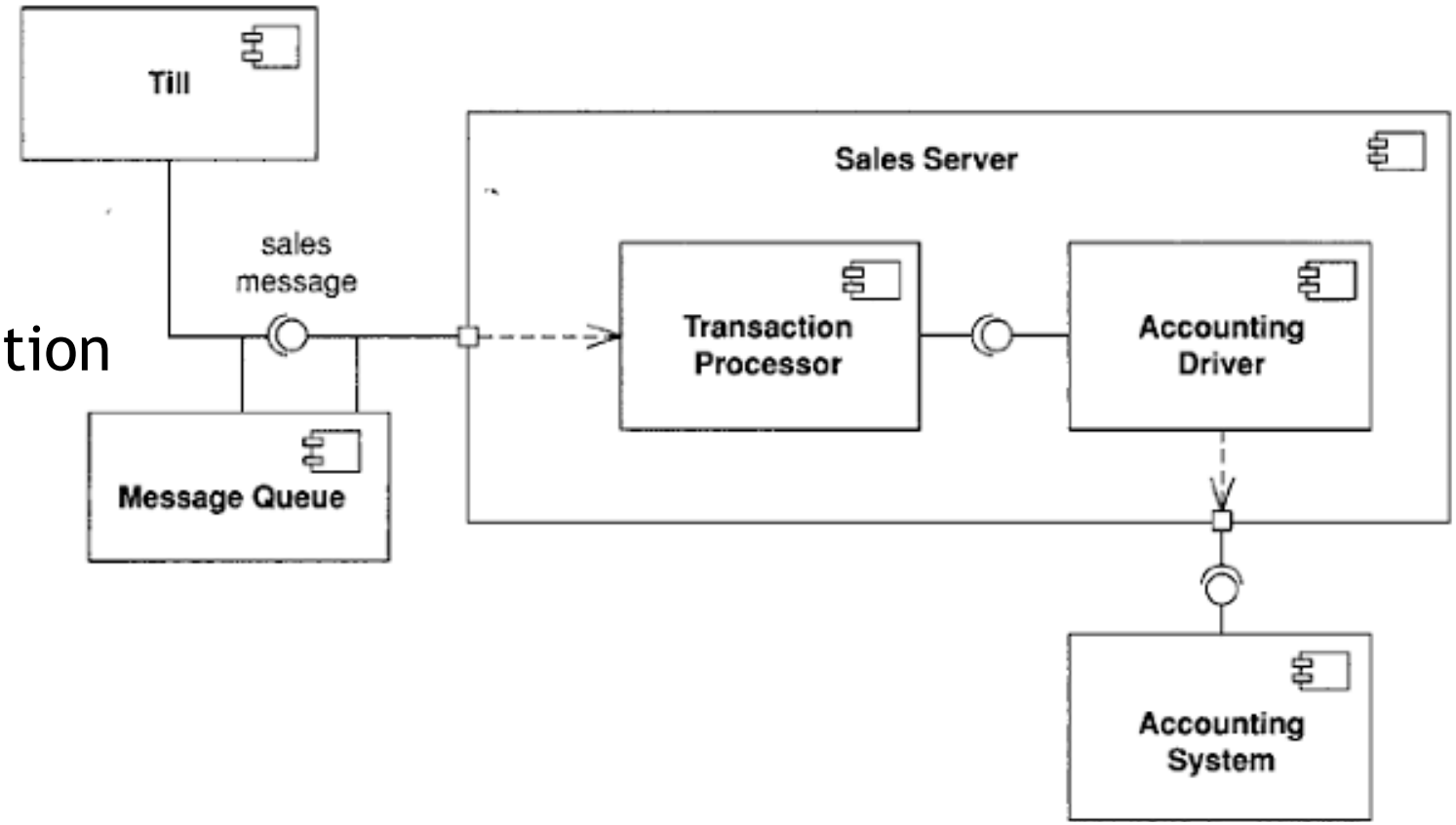
Component Diagram

- Captures physical structure of system implementation



Component Diagram

- Captures physical structure of system implementation

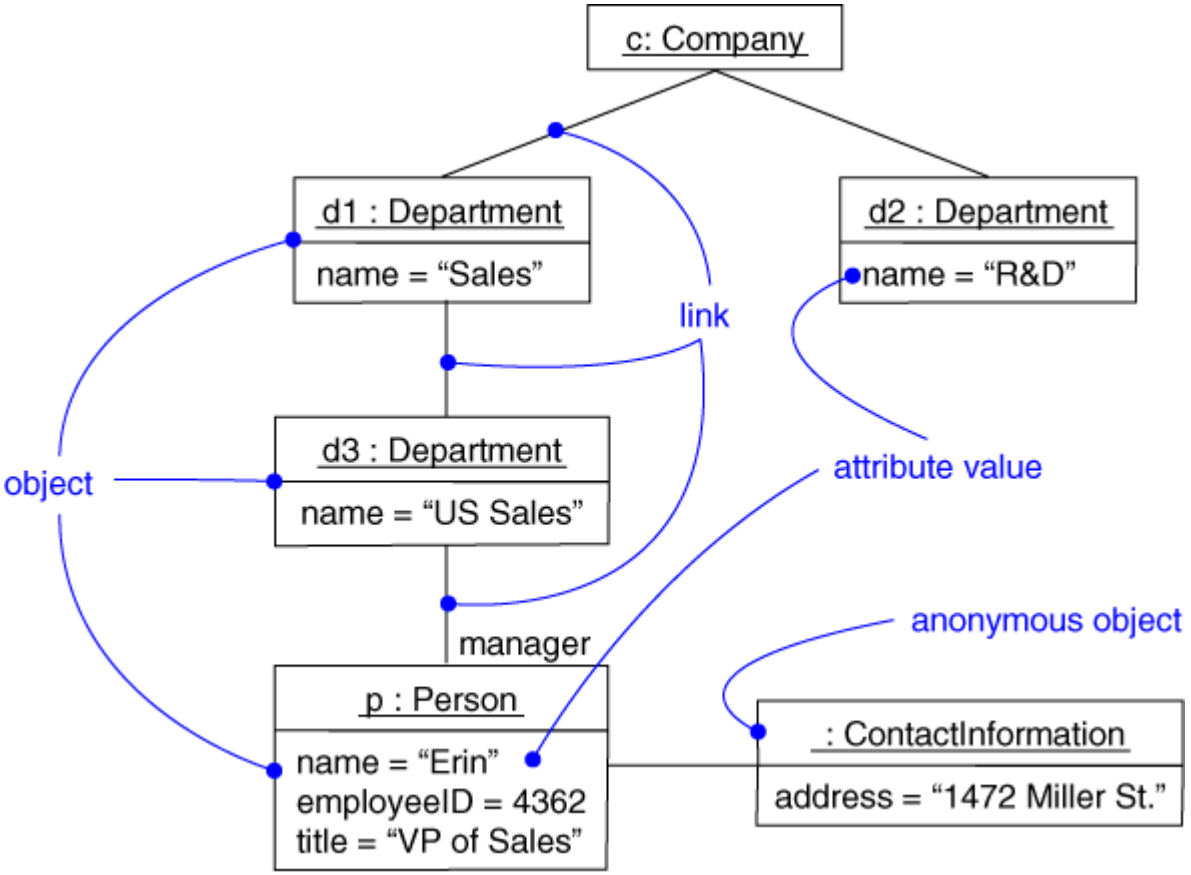


Component Diagram

- Captures the physical structure of the implementation
- Built as part of architectural specification
- Purpose
  - Organize source code
  - Construct an executable release
  - Specify a physical database
- Developed by architects and programmers

Object Diagram

- Captures instances and links

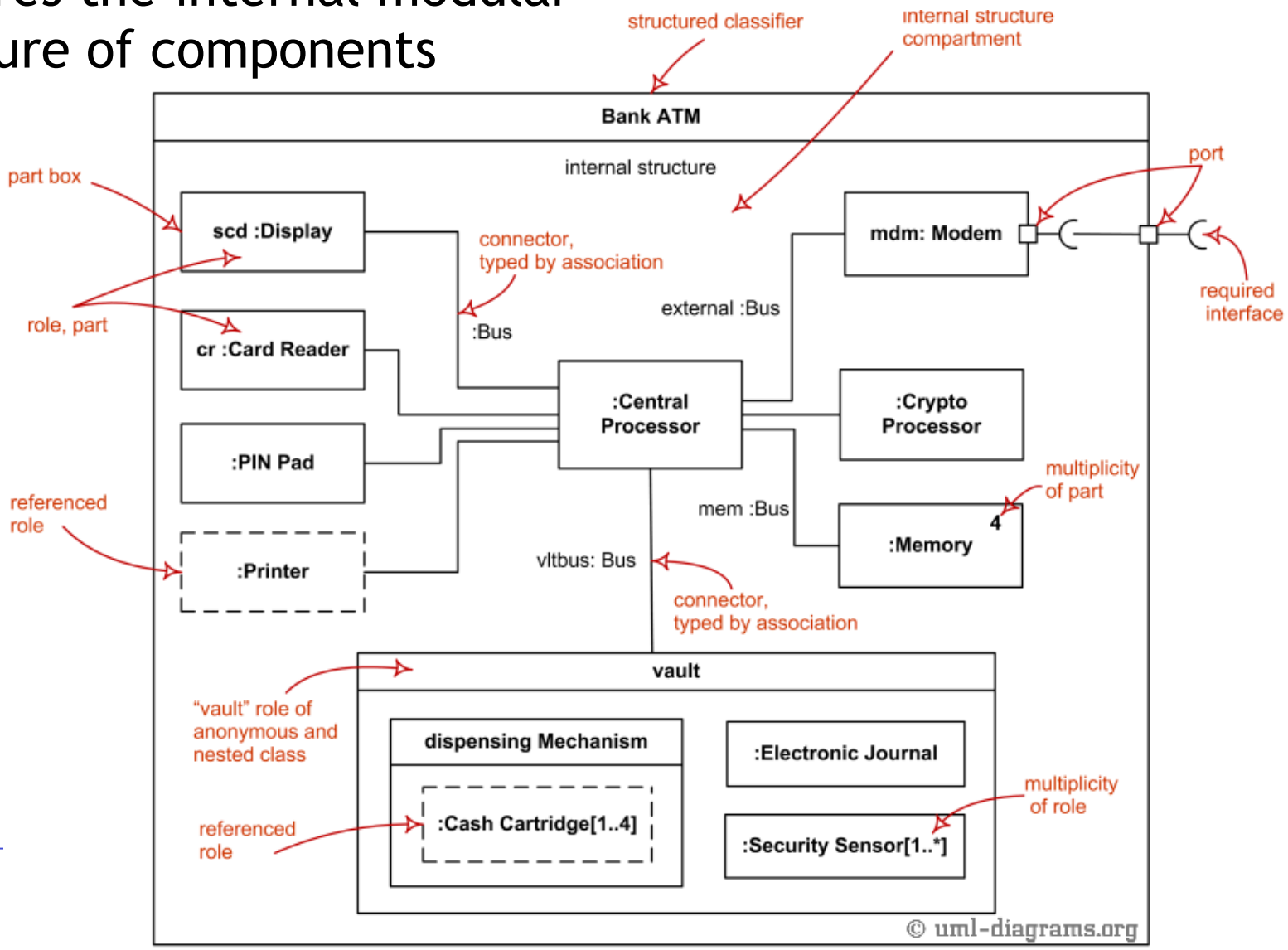


Object Diagram

- Shows instances and links that architecture elements maintain across functions
- Built during analysis and design
- Purpose
  - Illustrate data/object structures
  - Specify architecture runtime snapshots
- Developed by analysts, designers, and implementers

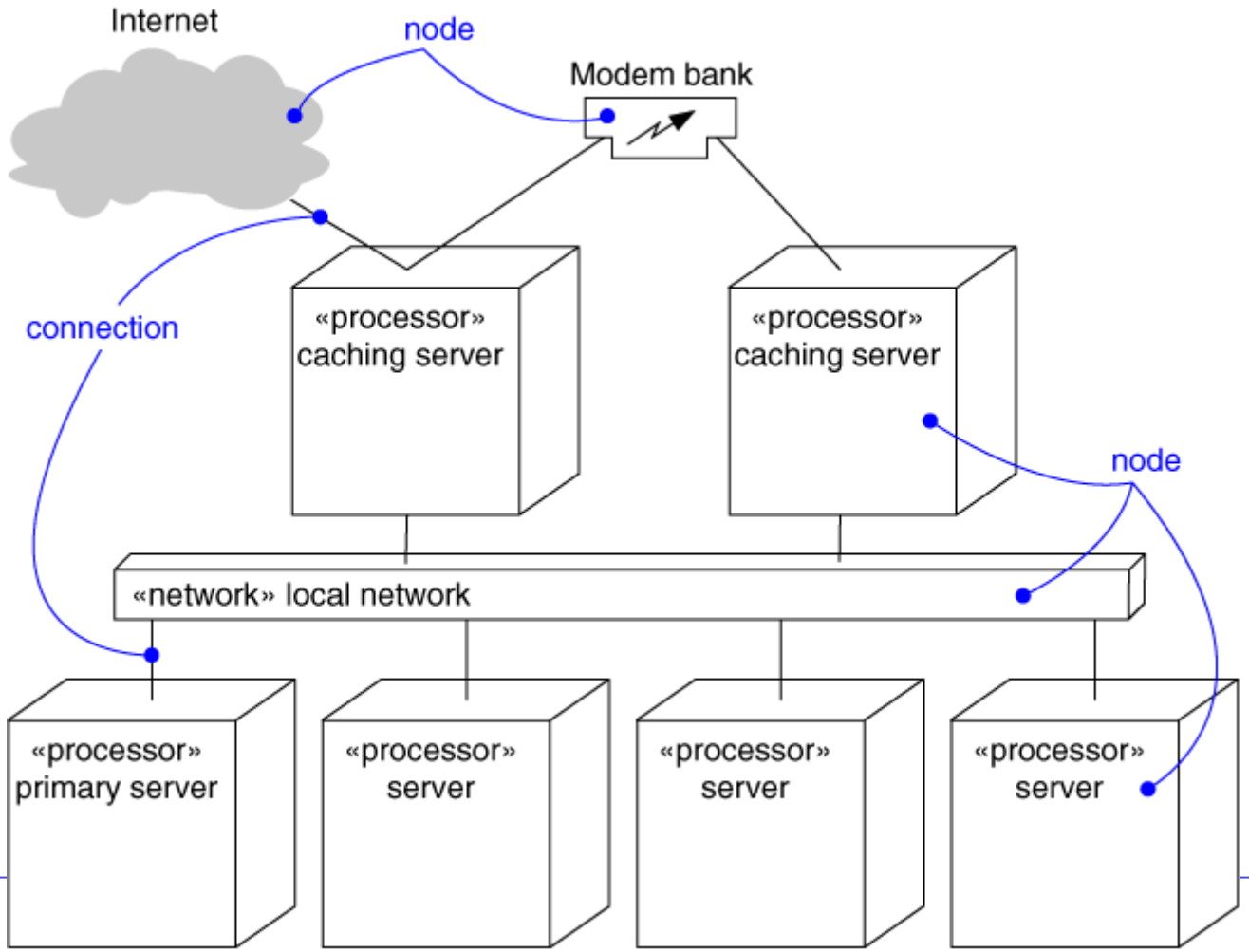
Composite Structure Diagram

- Captures the internal modular structure of components



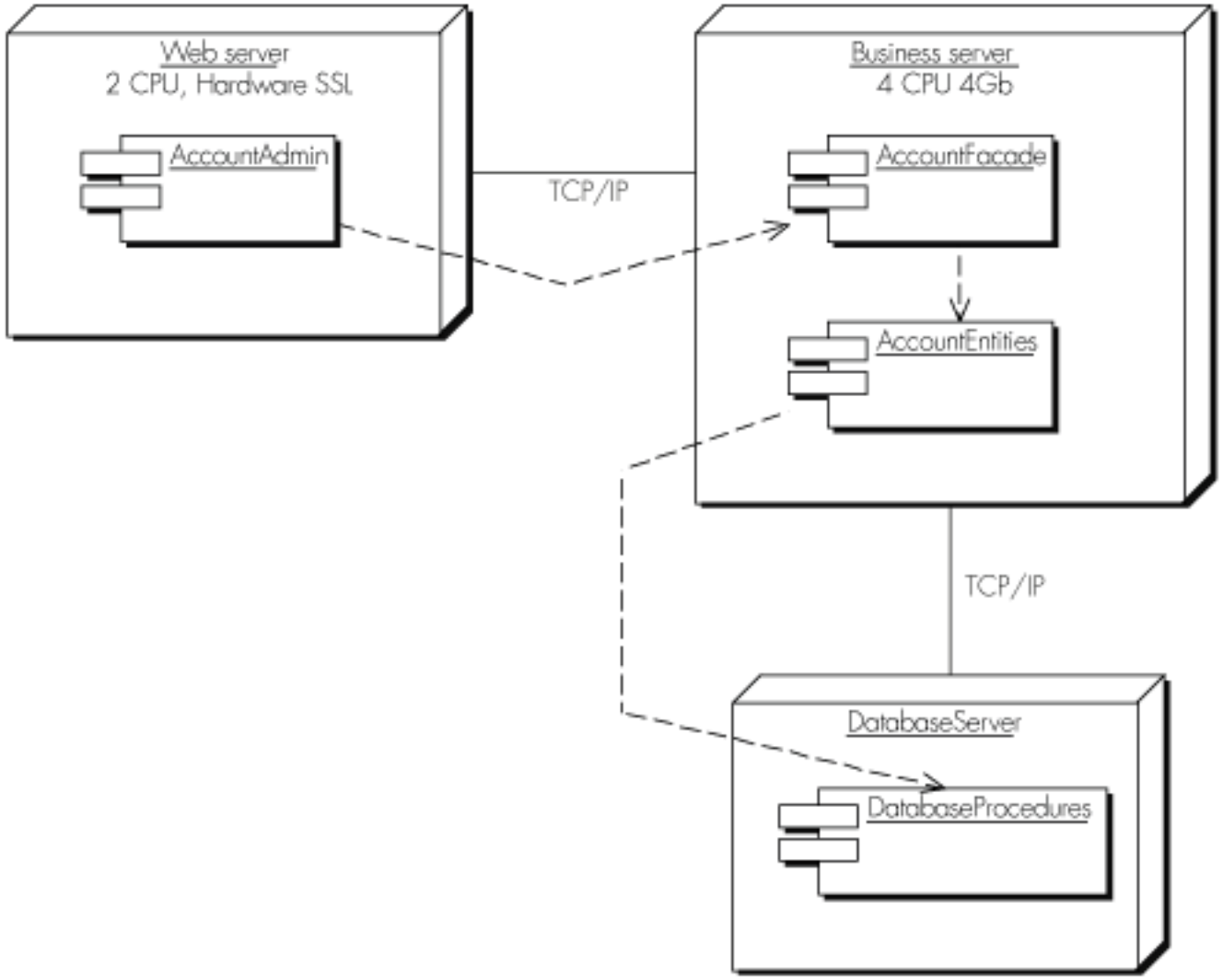
Deployment Diagram

- Captures the topology of a system's hardware



Deployment Diagram

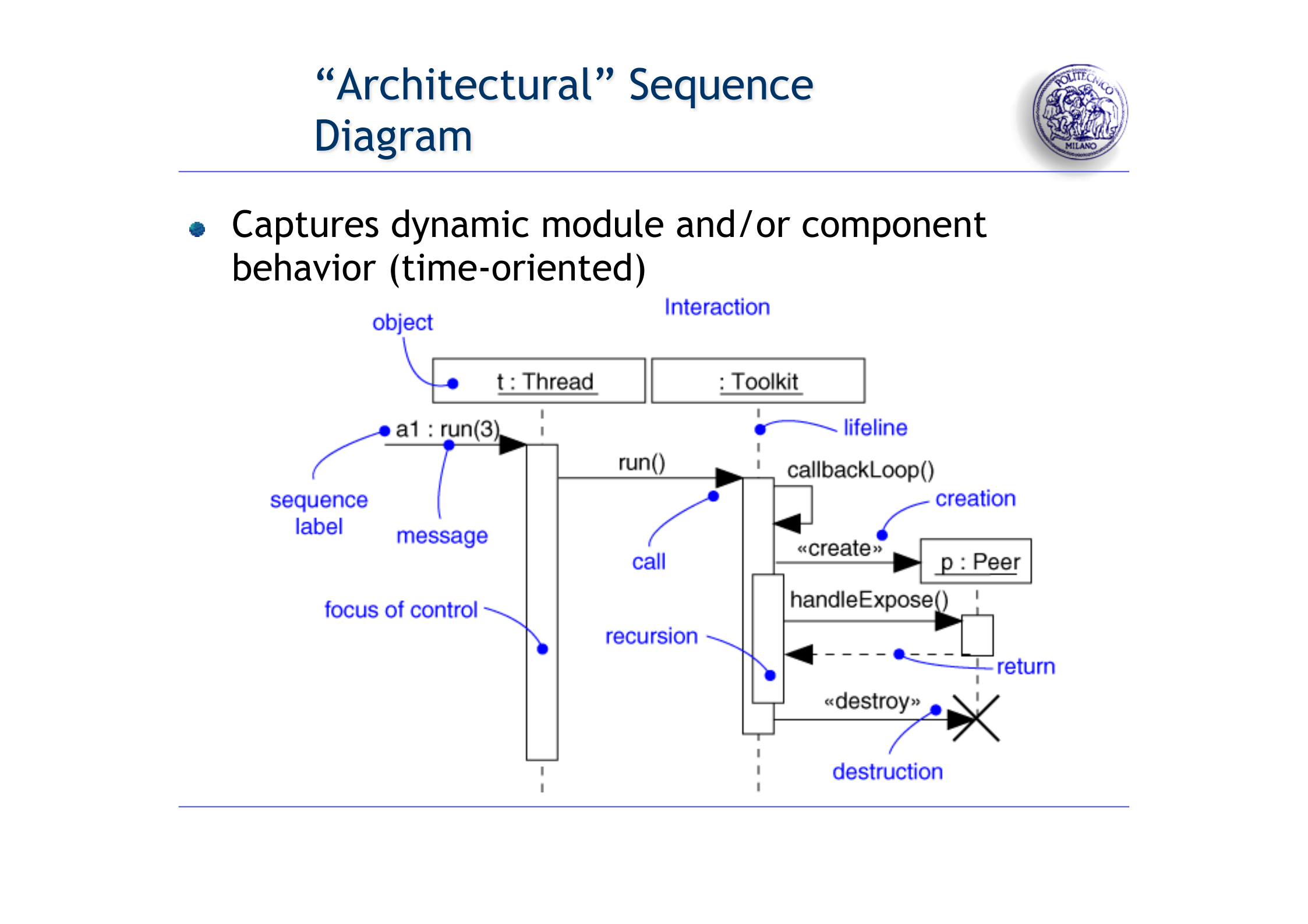
- Captures the topology of a system's hardware… **but not only!**



Deployment Diagram

- Captures the topology of a system's hardware
- Built as part of architectural specification
- Purpose
  - Specify the distribution of components
  - Identify performance bottlenecks
- Developed by architects, networking engineers, and system engineers

“Architectural” Sequence Diagram



- Captures dynamic module and/or component behavior (time-oriented)

“Architectural” Sequence Diagram

- Captures dynamic behavior (time-oriented)
- Purpose
  - Model flow of control
  - Illustrate typical scenarios
  - Analyse architecture -ilities

Politecnico
di Milano

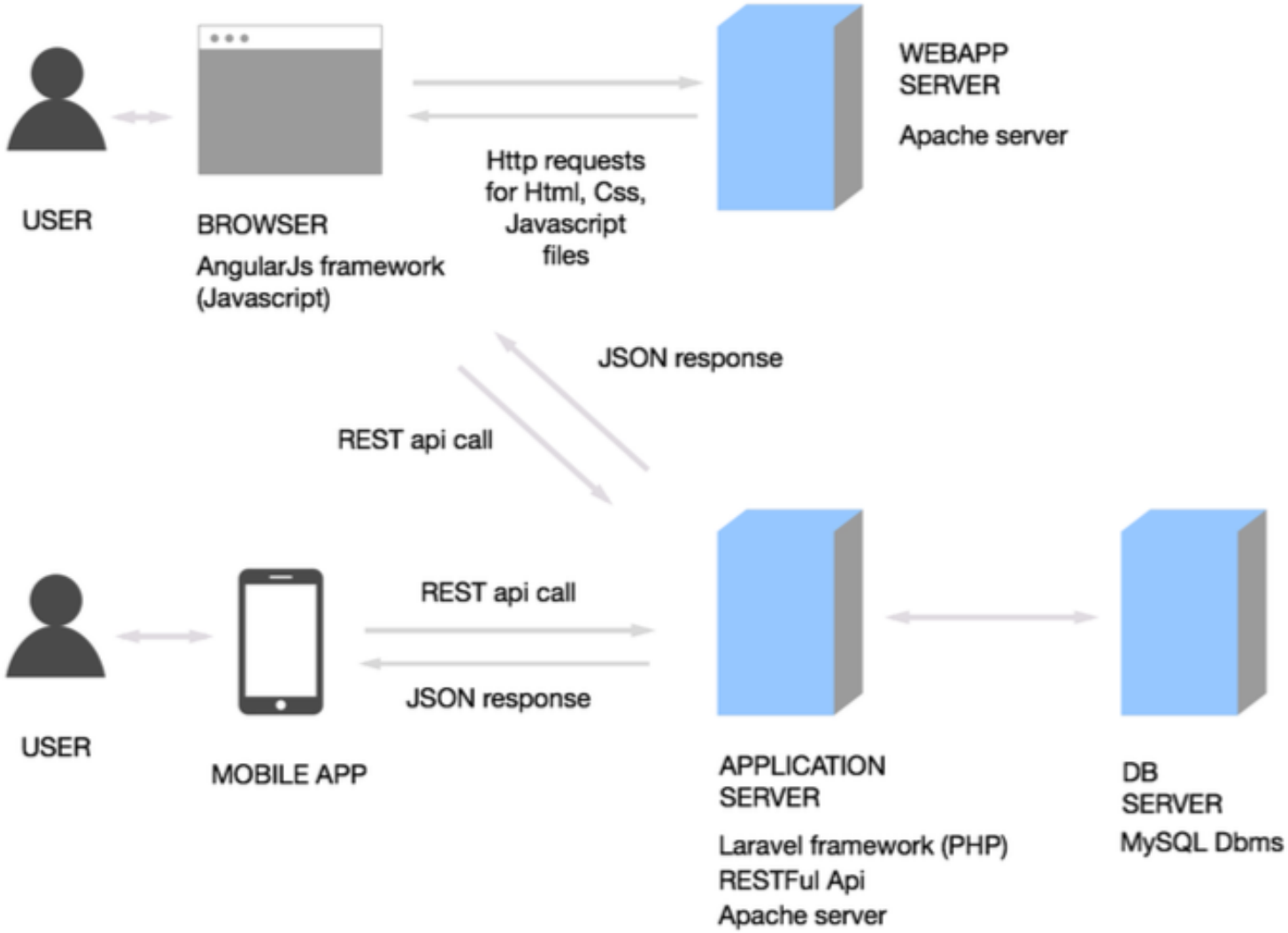
Putting it all at work… a sample scenario from last year's project!

Let’s see... Solution description!

- “We will implement a client-server architecture (Fig. 2) based on common REST API and MVC pattern, so with just one server application we manage both web application and mobile application, obviously we will have a version for taxi driver and a version for clients covering cases for handling login and other requests from drivers and clients.”

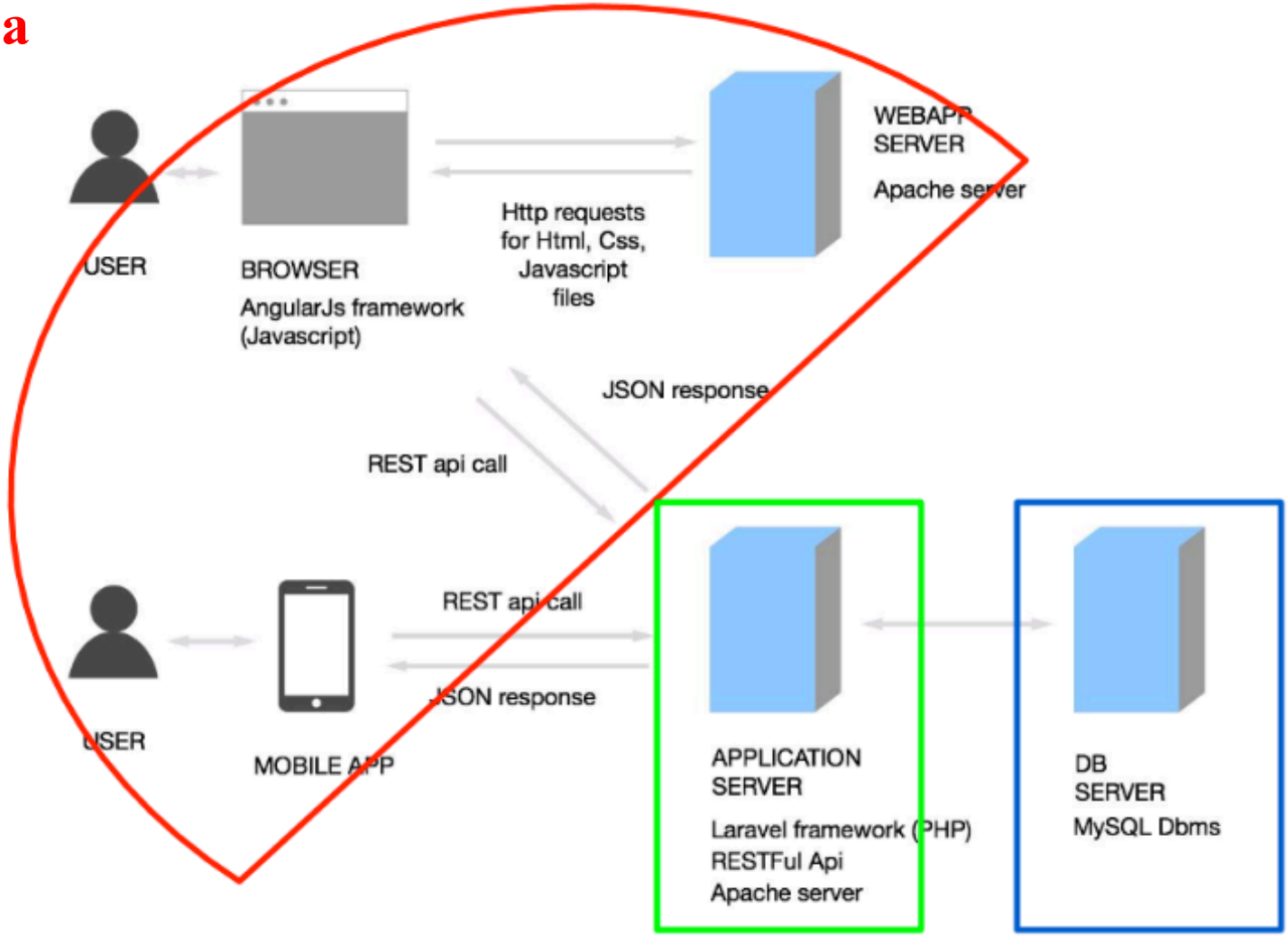
Cit. [the student’s interpretation of the textual specification]

Meanwhile, back in the requirements... Architecture Draft?!



and... First Architecture Decision!

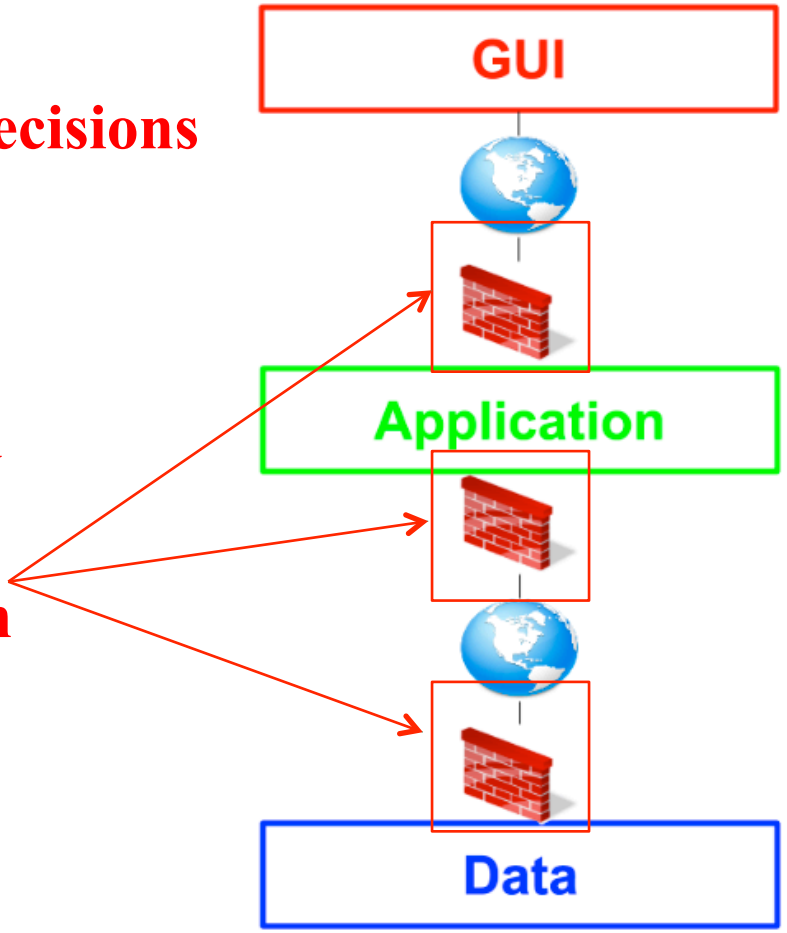
*** Architecture is a set of Design Decisions... Decision 1, three tiers!**



Architecture With Style

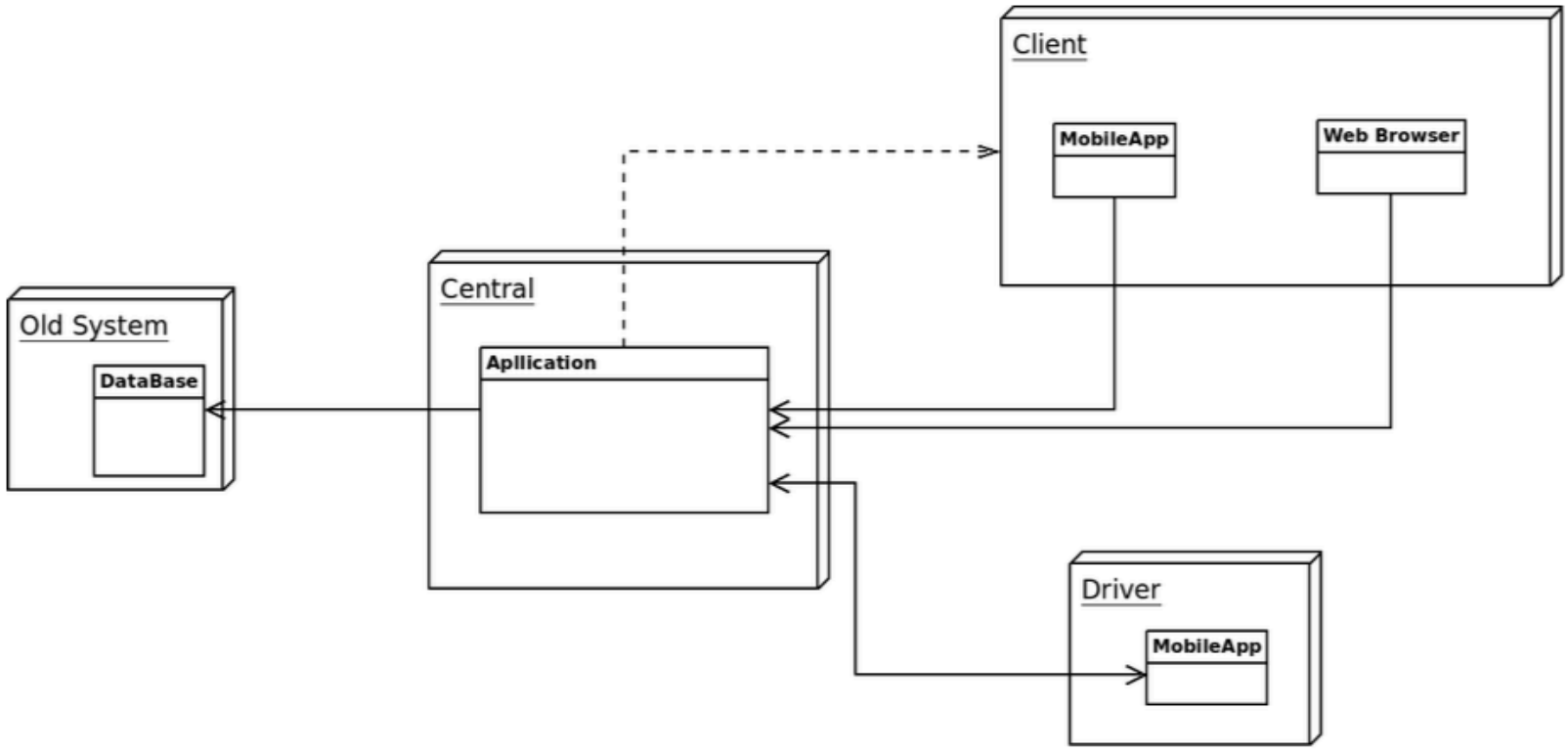
Hint: Document your decisions
And their reasons...

e.g., here you may want a secure distributed system



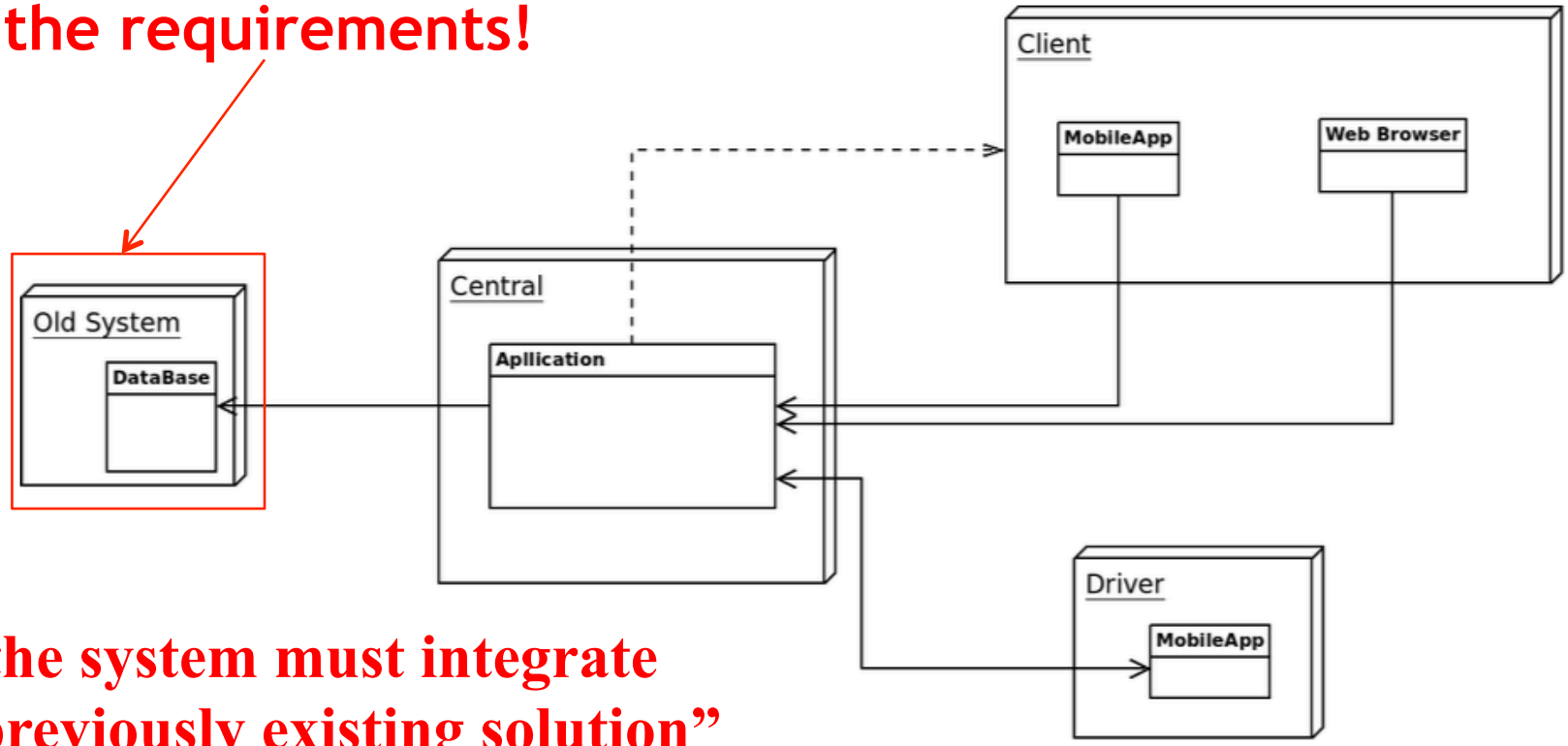
High-Level Components

- **You need to map your decisions to high-level physical representations**



High-Level Components

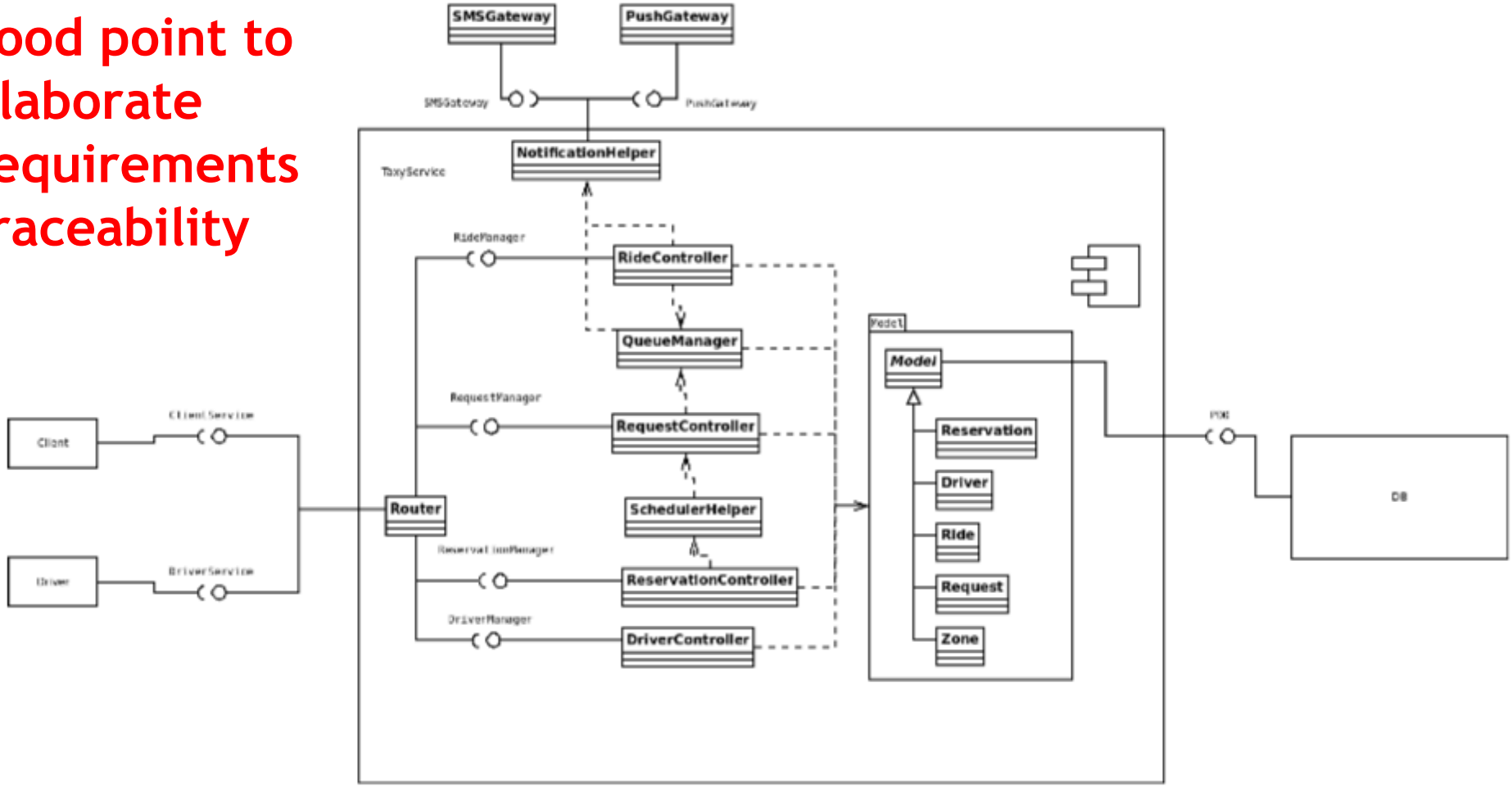
- **You need to map your decisions to high-level physical representations... And don't forget the requirements!**



R x.y: "the system must integrate With a previously existing solution"

A composite structure to elaborate modules...

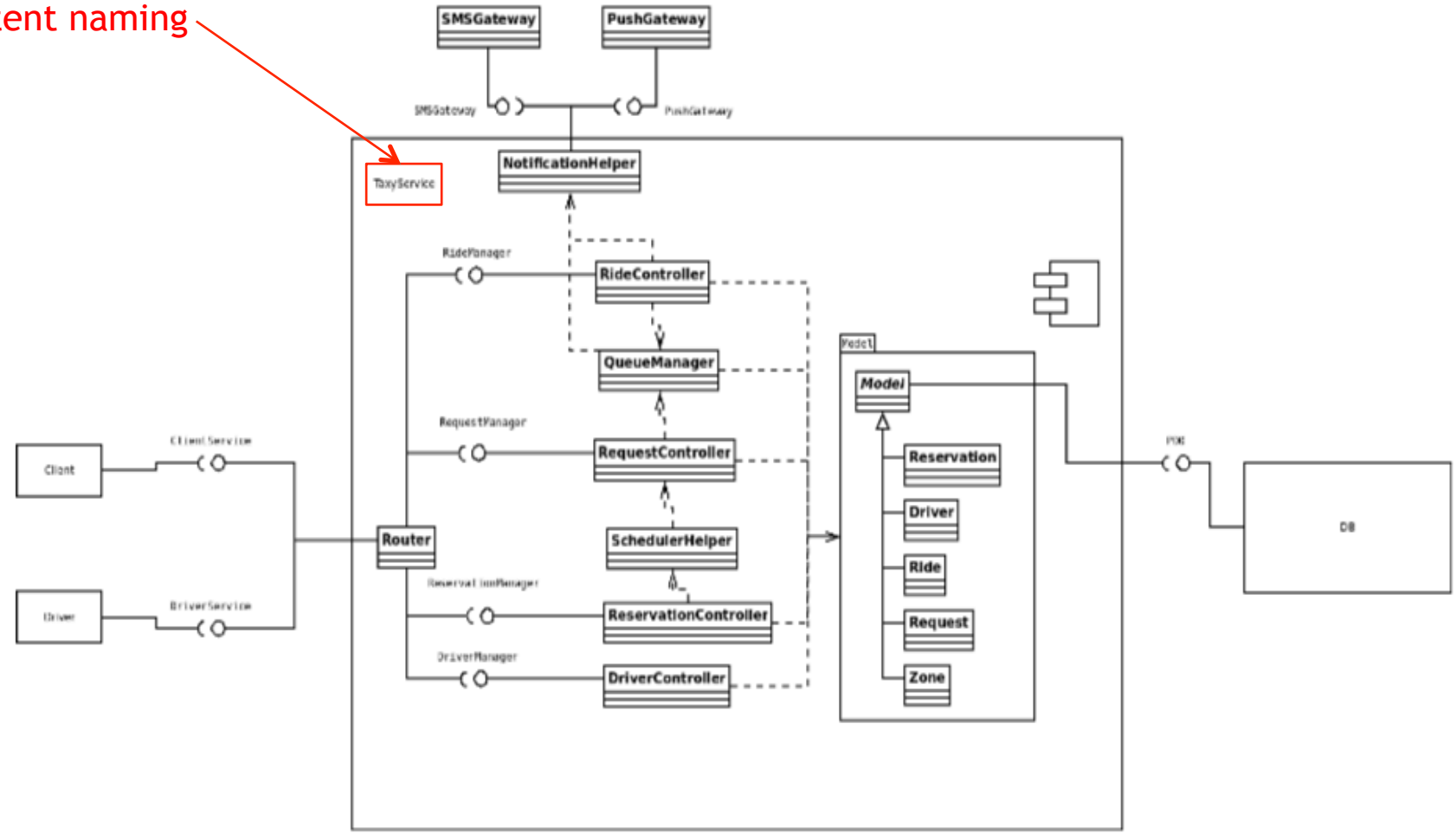
This would be a good point to elaborate requirements traceability



A composite structure to elaborate modules...

Remember:

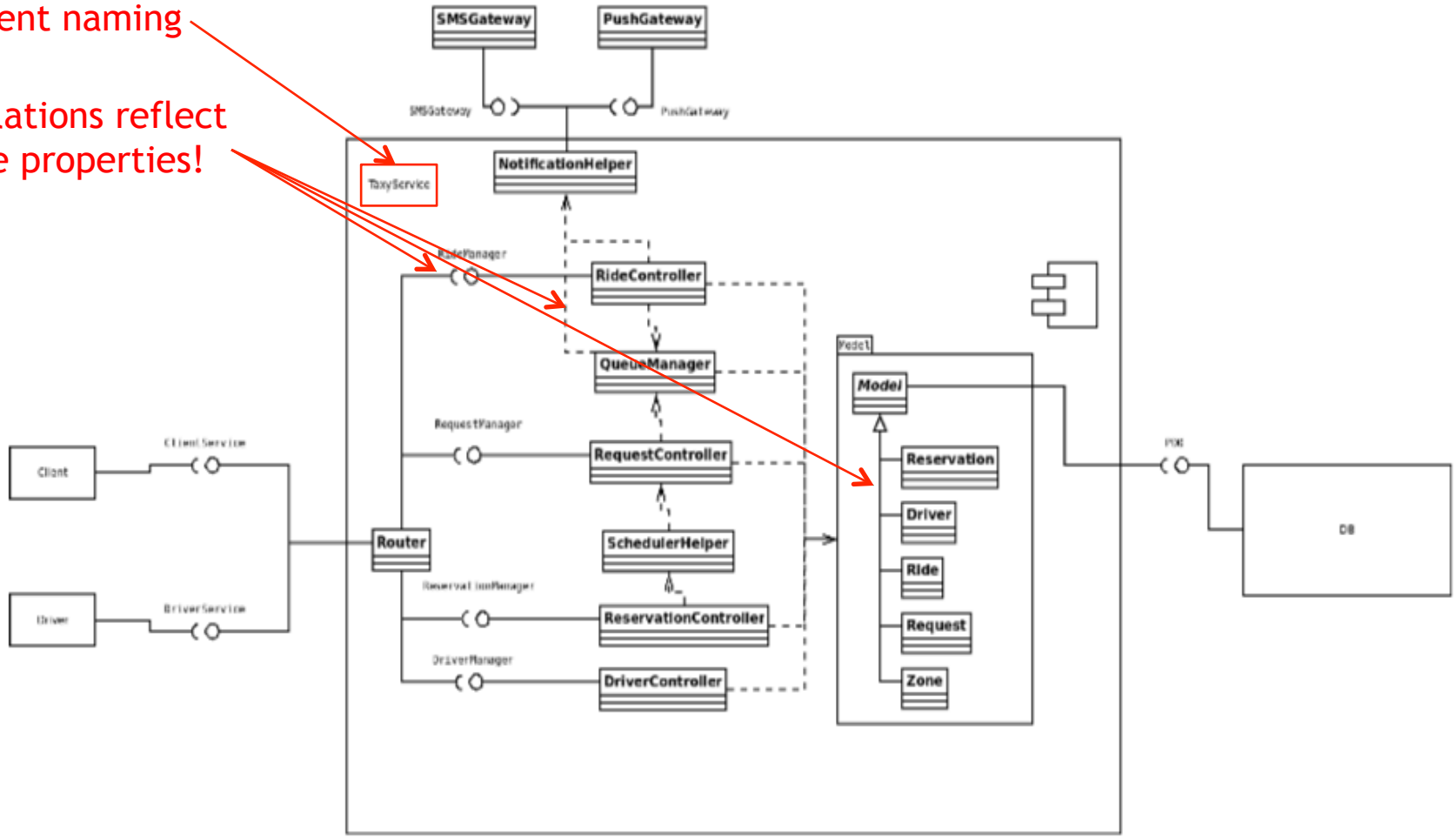
- Consistent naming



A composite structure to elaborate modules...

Remember:

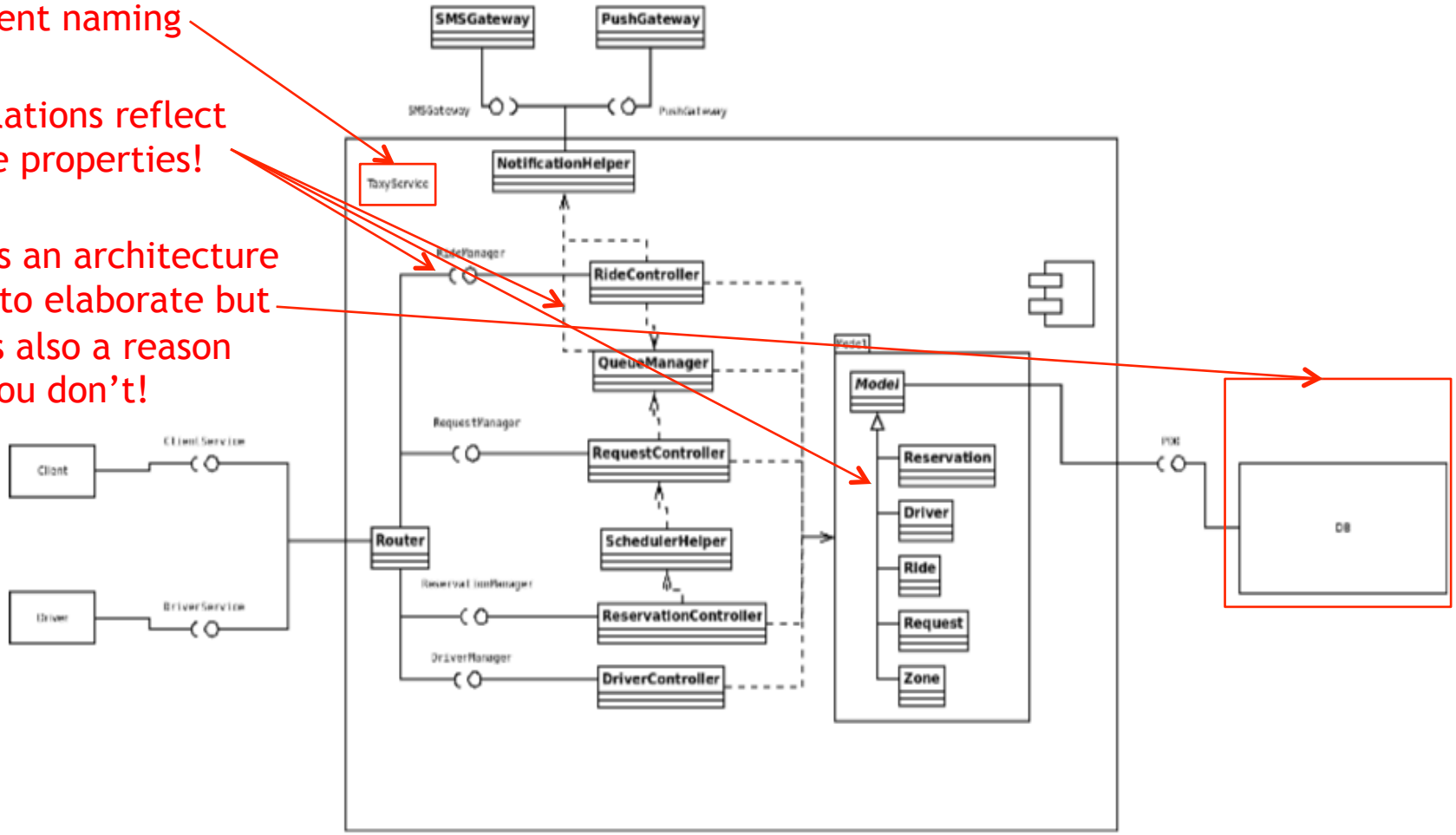
- Consistent naming
- UML relations reflect runtime properties!



A composite structure to elaborate modules...

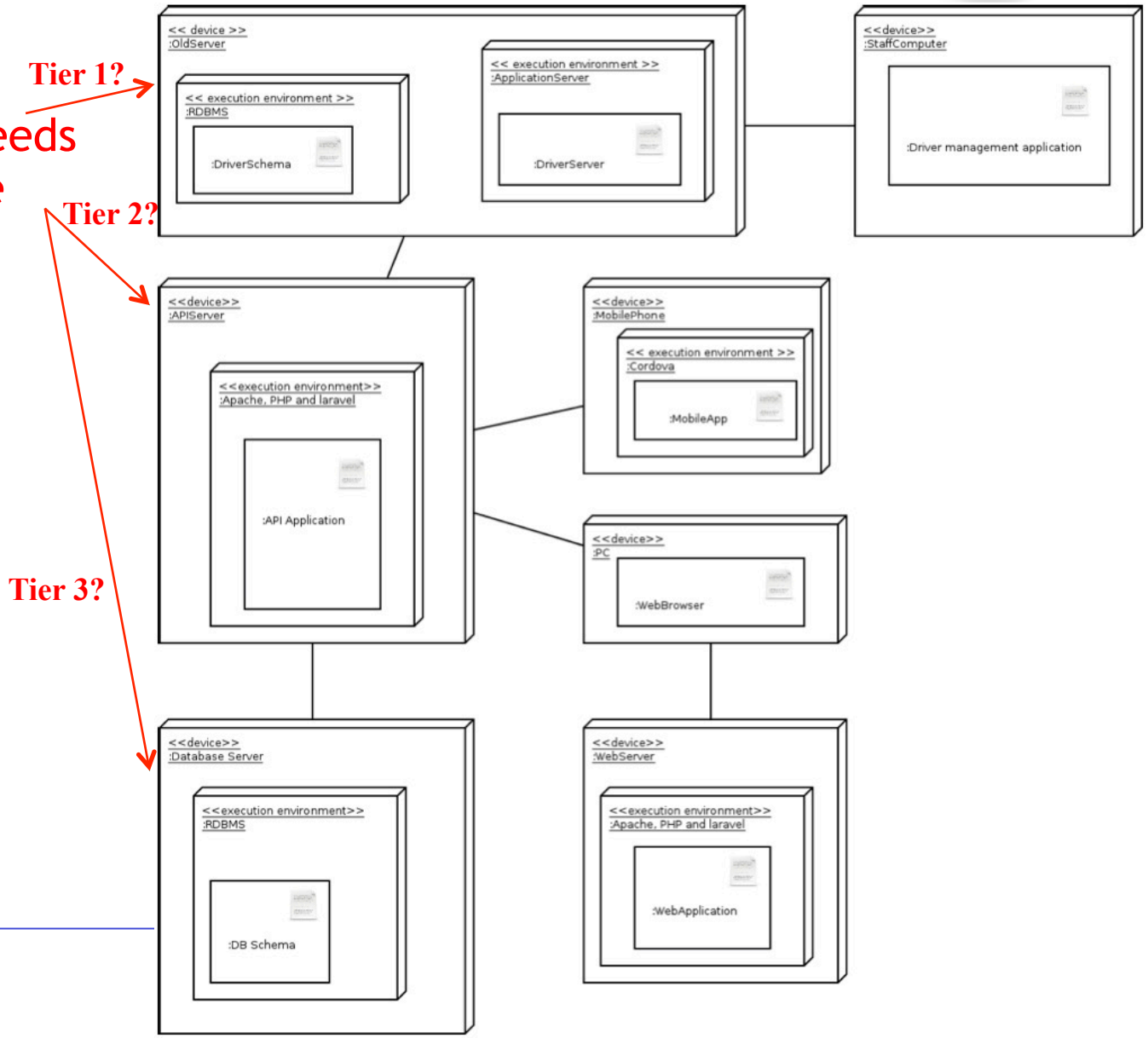
Remember:

- Consistent naming
- UML relations reflect runtime properties!
- There is an architecture reason to elaborate but there is also a reason when you don't!



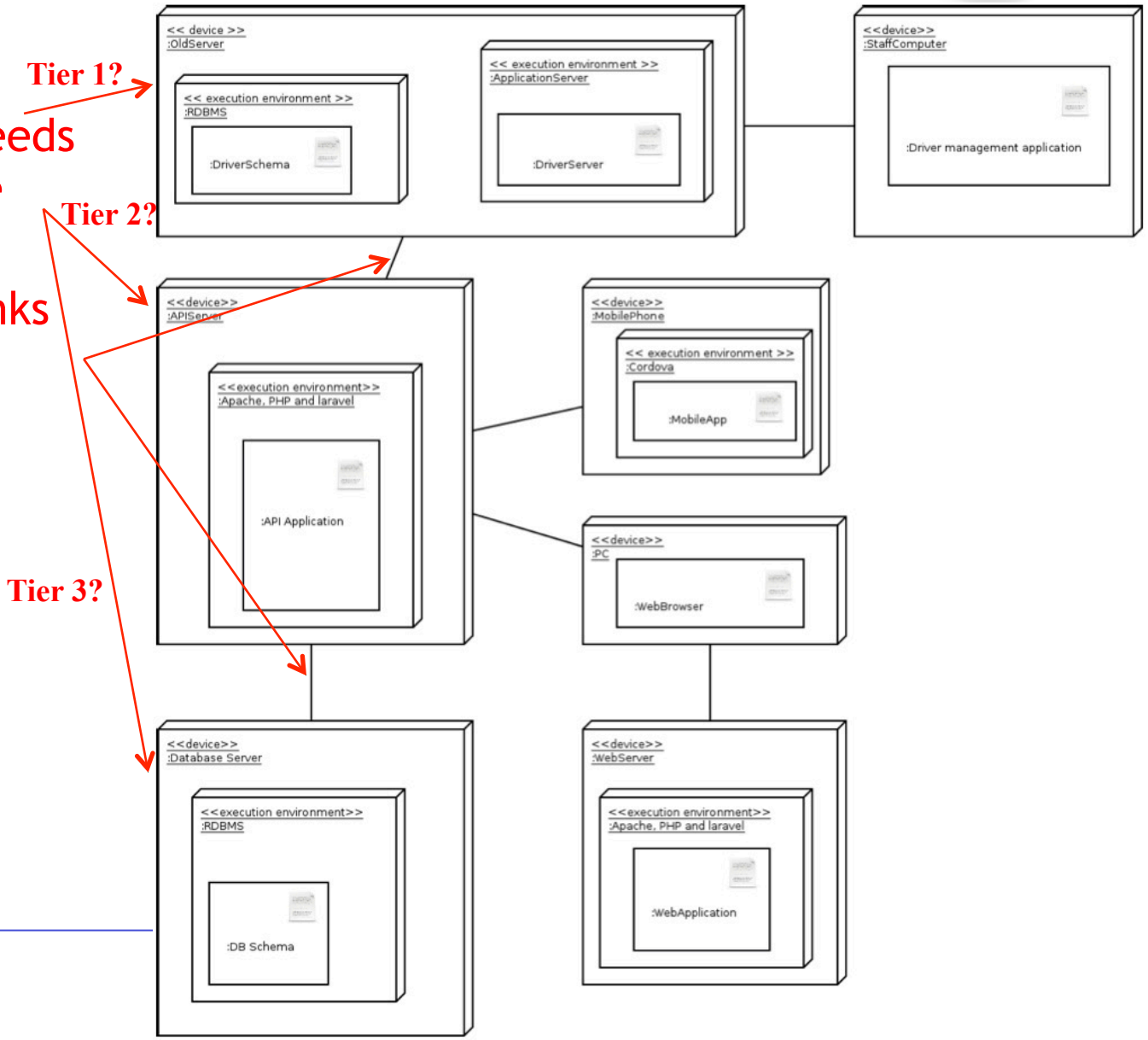
A deployment diagram to elaborate physical components allocation

- remember:
  - Deployment needs to reflect style restrictions



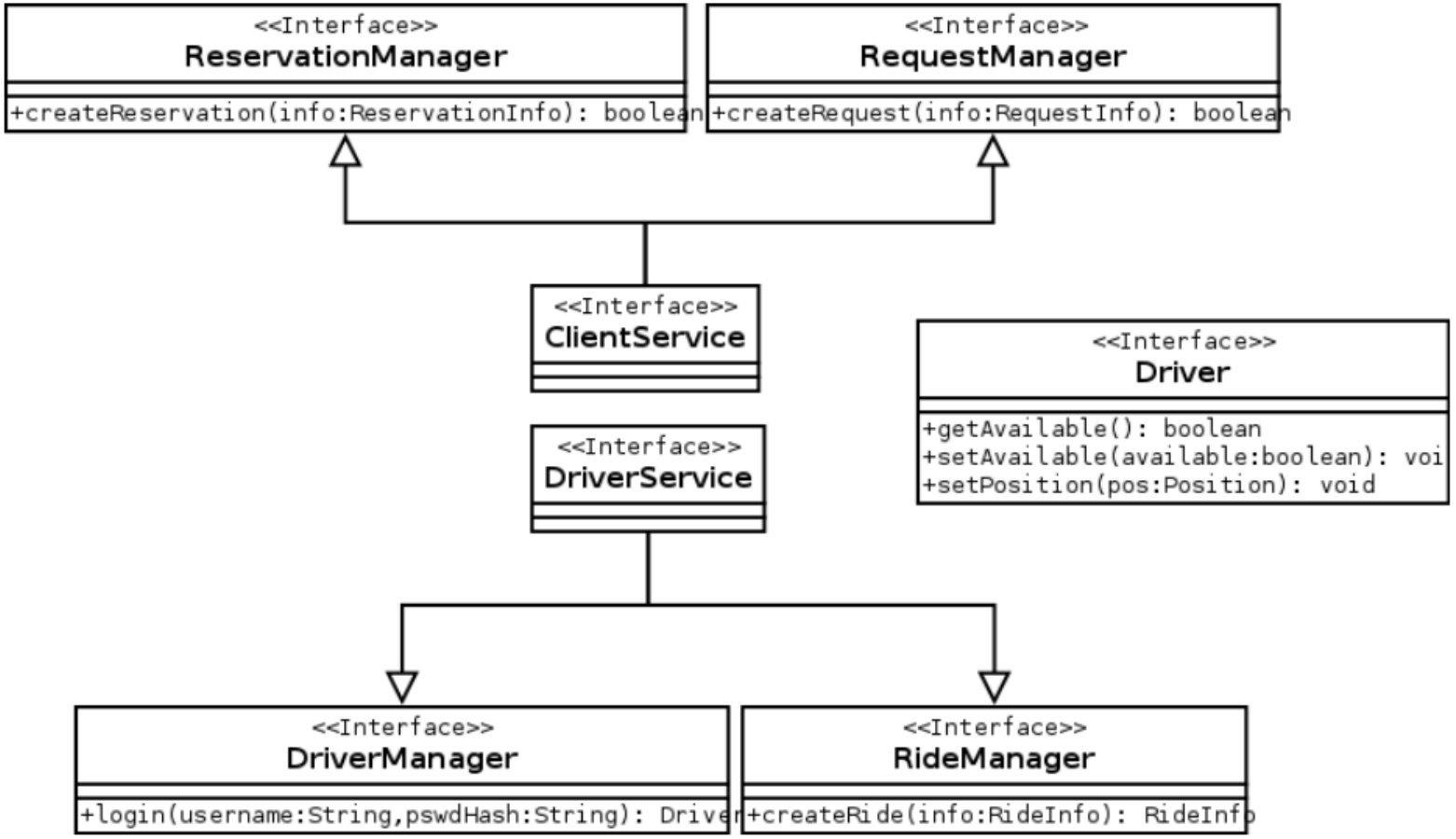
A deployment diagram to elaborate physical components allocation

- remember:
  - Deployment needs to reflect style restrictions
  - Deployment links need to follow composite structure



But... I also have a class diagram!

- Try to provide at least the same level of detail for each view...



Process View... Activities & States!

<Omitted for the sake of brevity*>

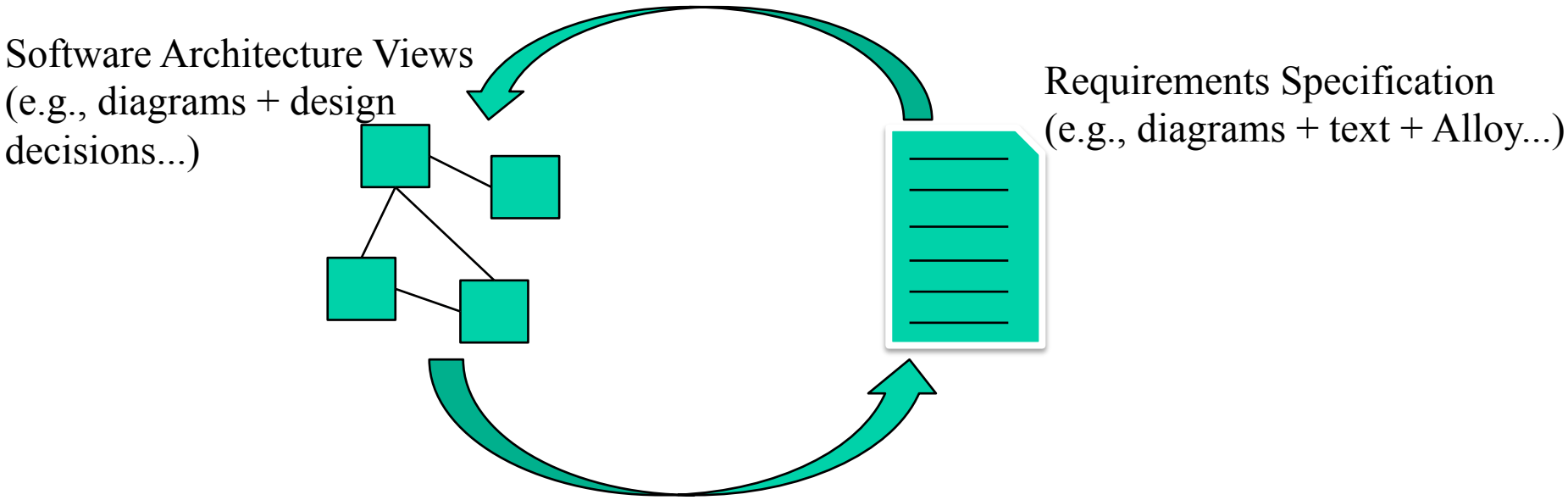
Remember to provide behavioral architecture views that:

1. Convey meaningful architecture properties (find them in the requirements), e.g., for analysis;
2. Reflect implementable behavior (functional);
3. Make behavioral assumptions explicit (requirements and domain assumptions!);

*can be found on the project deliverable which will be released! ☺

So, in practice...

- Iteration between **architecture** (and architecture views/diagrams) and **requirements** is key!



Thanks for your endurance…

- Break?

Coming up next… More on requirements and also other examples!

Yet another example... Artificial Neural Network*!

Usual drill... Some text!

- “An ANN is a structure consisting of interconnected nodes that communicate with one another. The structure and its functionality are inspired by neural networks found in a biological brain. Hebbian Theory explains [...] Similarly, a typical ANN has connections between nodes that have a weight which is updated as the network learns. The nodes labeled "+1" are called *biases*. The leftmost blue column of nodes are *input nodes*, the middle column contains *hidden nodes*, and the rightmost column contains *output nodes*. There may be many columns of hidden nodes, known as *hidden layers*...”

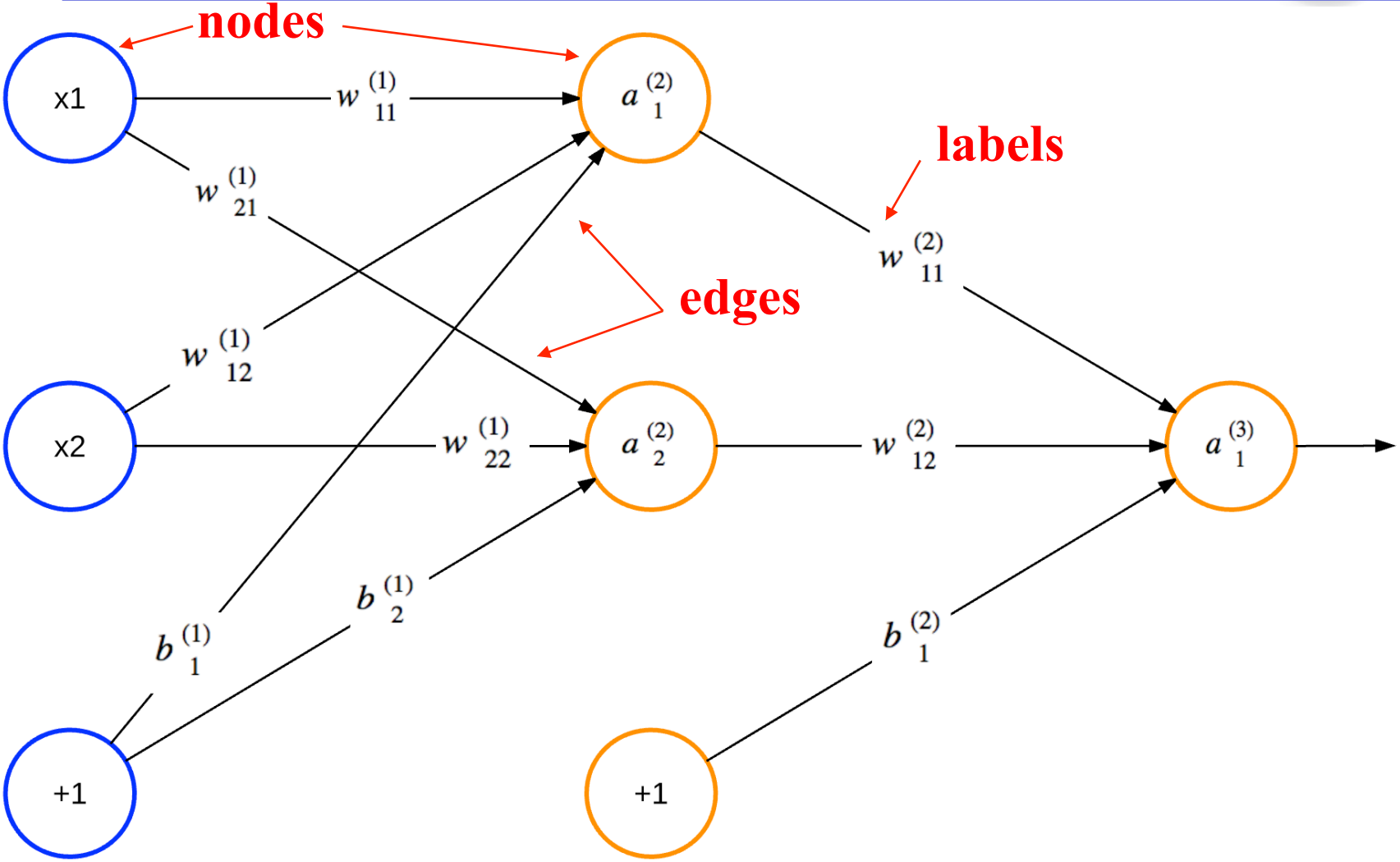
Goal: devise a system to support ANNs:

→ draw a single digit;

→ OCR training with digit;

→ request the system for a prediction;

ANN, let's visualize an example



Now... Let's focus on the top-most goal(s)

- "An ANN is a structure consisting of interconnected nodes that communicate with one another. The structure and its functionality are inspired by neural networks found in a biological brain. Hebbian Theory explains [...] Similarly, a typical ANN has connections between nodes that have a weight which is updated as the network learns. The nodes labeled "+1" are called *biases*. The leftmost blue column of nodes are *input nodes*, the middle column contains *hidden nodes*, and the rightmost column contains *output nodes*. There may be many columns of hidden nodes, known as *hidden layers*..."

Goal: devise a **system** to support ANNs:

→ **draw** a single digit;

→ OCR **training** with digit;

→ **request** the system for a prediction;

Now... Let's focus on the top-most goal(s)

- So... We need:
  - ANN system;
  - An interface with that system;
  - ANN Design logic;
  - ANN Training logic;

Goal: devise a **system** to support ANNs:

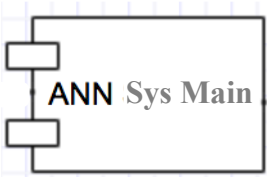
→ **draw** a single digit;

→ OCR **training** with digit;

→ **request** the system for a prediction;

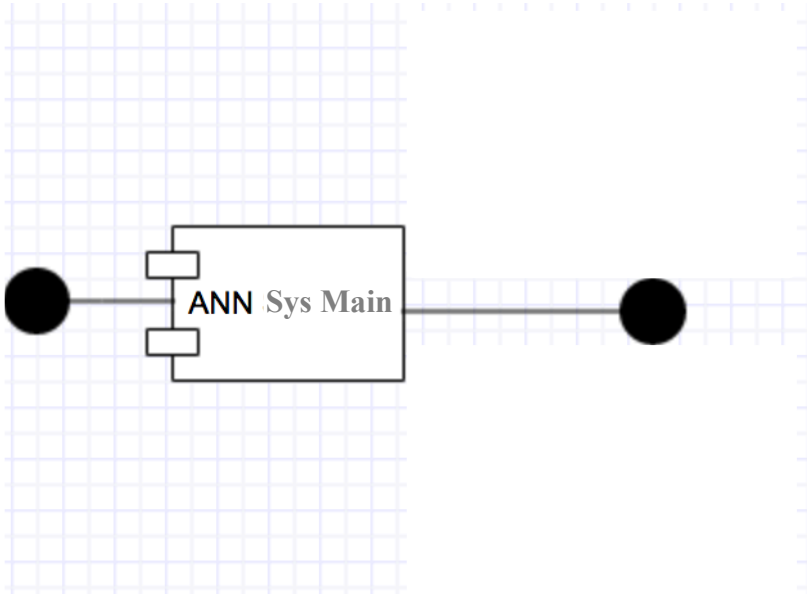
Now... Let's focus on the top-most goal(s)

- So... We need:
  - **ANN system;**
  - ANN interface;
  - ANN Design logic;
  - ANN Training logic;



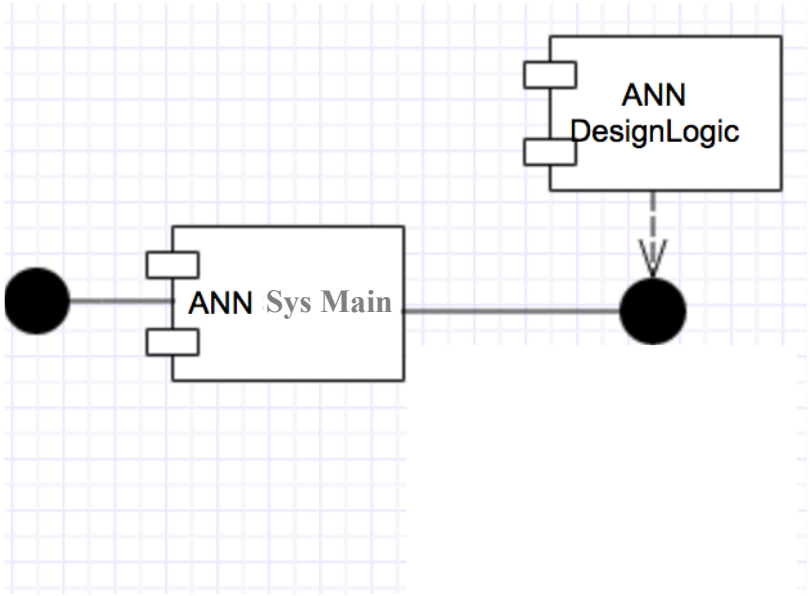
Now... Let's focus on the top-most goal(s)

- So... We need:
  - **ANN system;**
  - **ANN interface;**
  - ANN Design logic;
  - ANN Training logic;



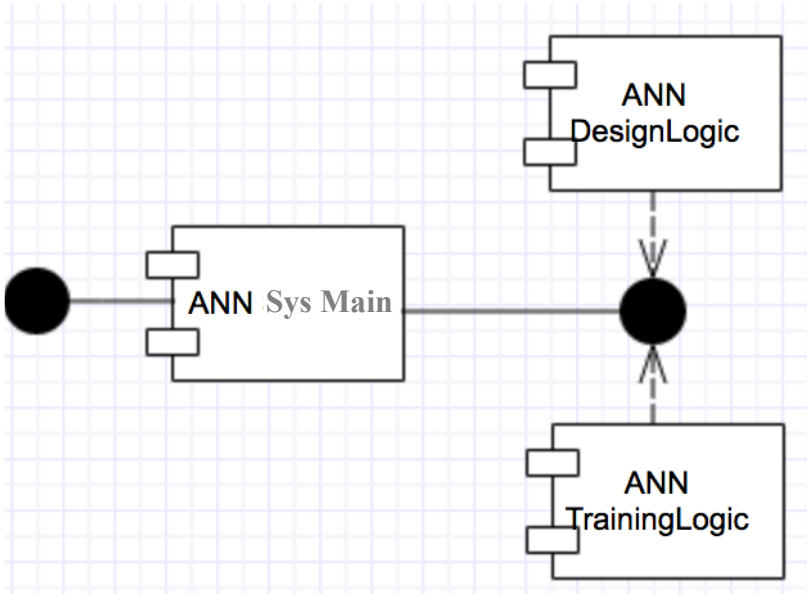
Now... Let's focus on the top-most goal(s)

- So... We need:
  - **ANN system;**
  - **ANN interface;**
  - **ANN Design logic;**
  - ANN Training logic;



Now... Let's focus on the top-most goal(s)

- So... We need:
  - **ANN system;**
  - **ANN interface;**
  - **ANN Design logic;**
  - **ANN Training logic**

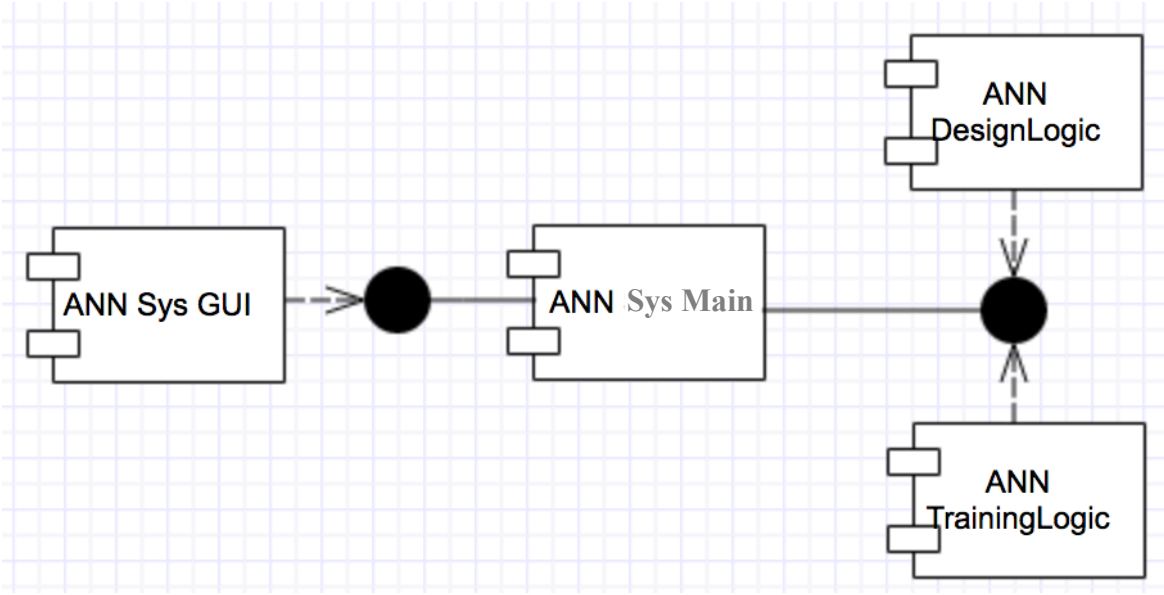


...Enough?

Now... Let's focus on the top-most goal(s)

- So... We need:
  - ANN system;
  - ANN interface;
  - ANN Design logic;
  - ANN Training logic;

Is this good?



Goal: devise a **system** to support ANNs:

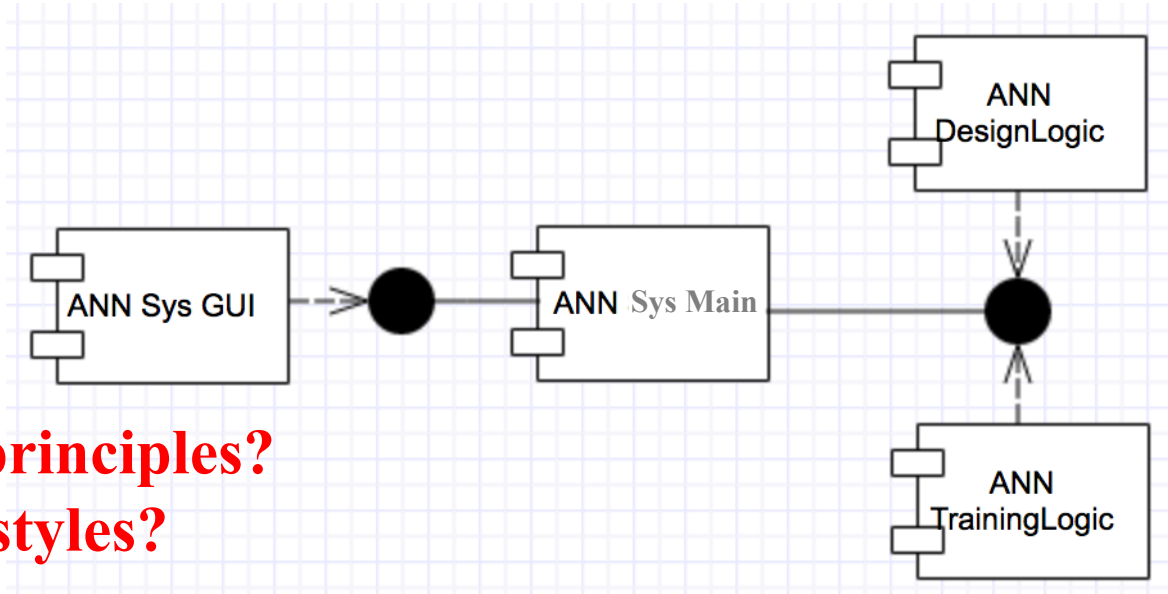
→ **draw** a single digit;

→ OCR **training** with digit;

→ **request** the system for a prediction;

Now... Let's focus on the top-most goal(s)

- So... We need:
  - ANN system;
  - ANN interface;
  - ANN Design logic;
  - ANN Training logic;



Is this good? Depends...

1. **Does it follow design principles?**
2. **Does it match known styles?**
3. **...**

Goal: devise a **system** to support ANNs:

→ **draw** a single digit;

→ OCR **training** with digit;

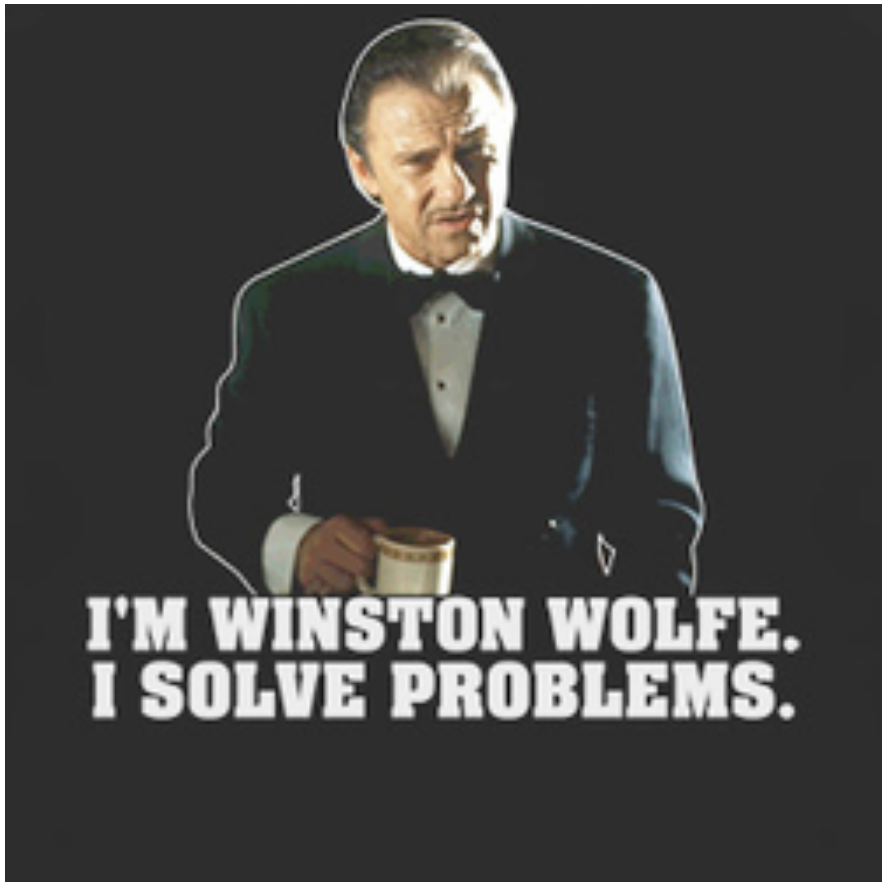
→ **request** the system for a prediction;

Always remember...

Theorem: “Software Architects Solve Problems”

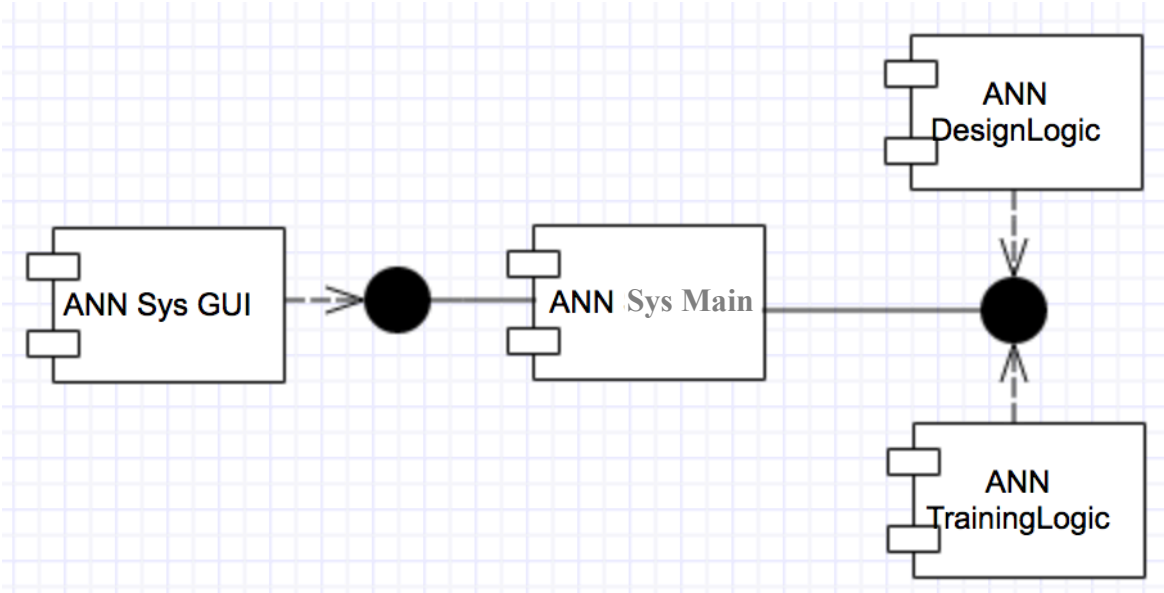
Proof - <internet still exists>

Corollary – <software is a problem>



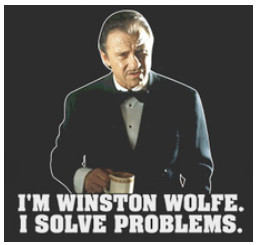
Now... Let's focus on the top-most goal(s)

- So... We need:
  - ANN system;
  - ANN interface;
  - ANN Design logic;
  - ANN Training logic;



Is this good? Probably:

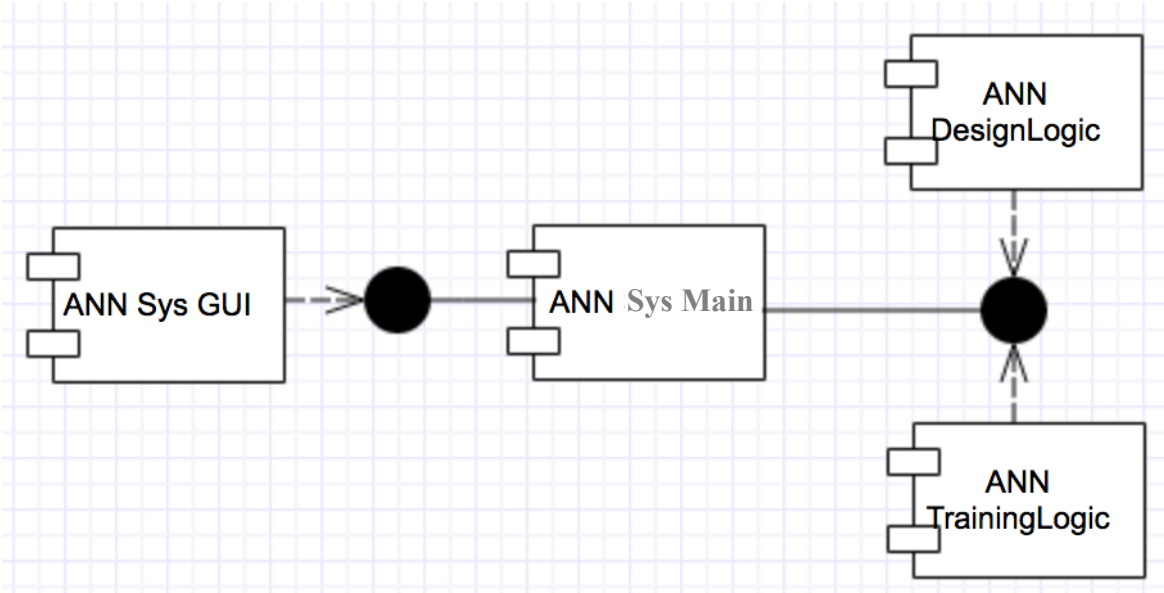
1. **Loose coupling**
2. **Low cohesion**
3. **High modularity**
4. **consistent with Client/server**
5. **...**



Decision 1: this architecture is good!

Now... Let's focus on the top-most goal(s)

- So... We need:
  - ANN system;
  - ANN interface;
  - ANN Design logic;
  - ANN Training logic;



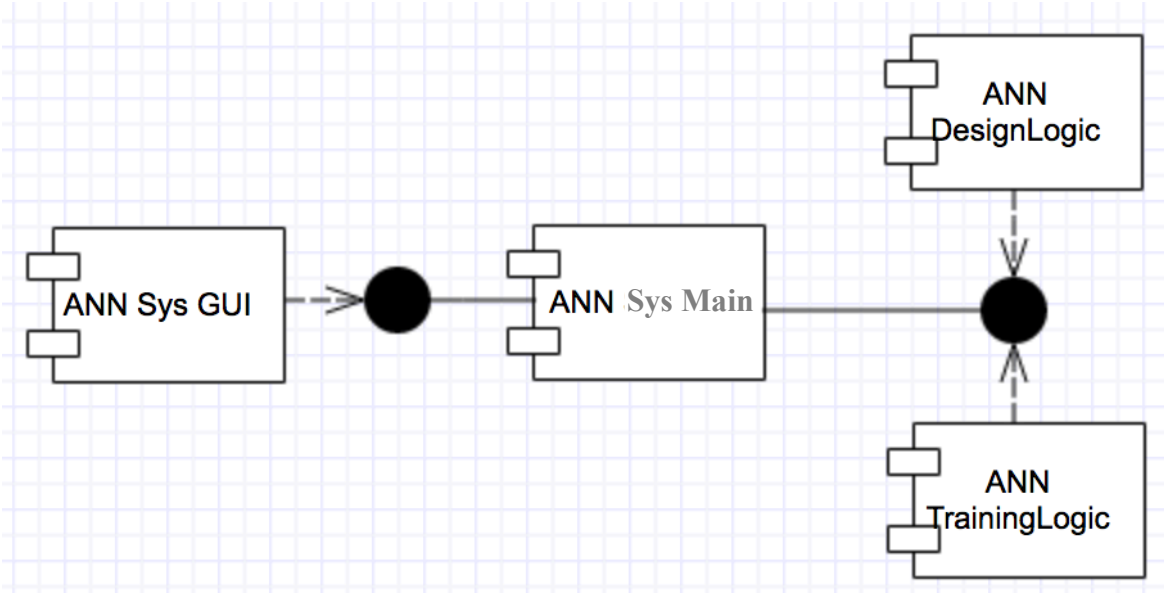
Is this good? Probably.

1. Loose coupling
2. Low cohesion
3. High modularity
4. **consistent with Client/server**
5. **...**

Decision 1: this architecture is good!

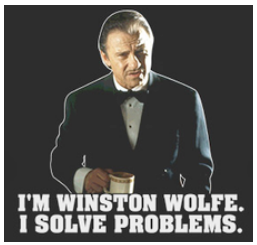
Now... Let's focus on the top-most goal(s)

- So... We need:
  - ANN system;
  - ANN interface;
  - ANN Design logic;
  - ANN Training logic;



Is this good? Probably.

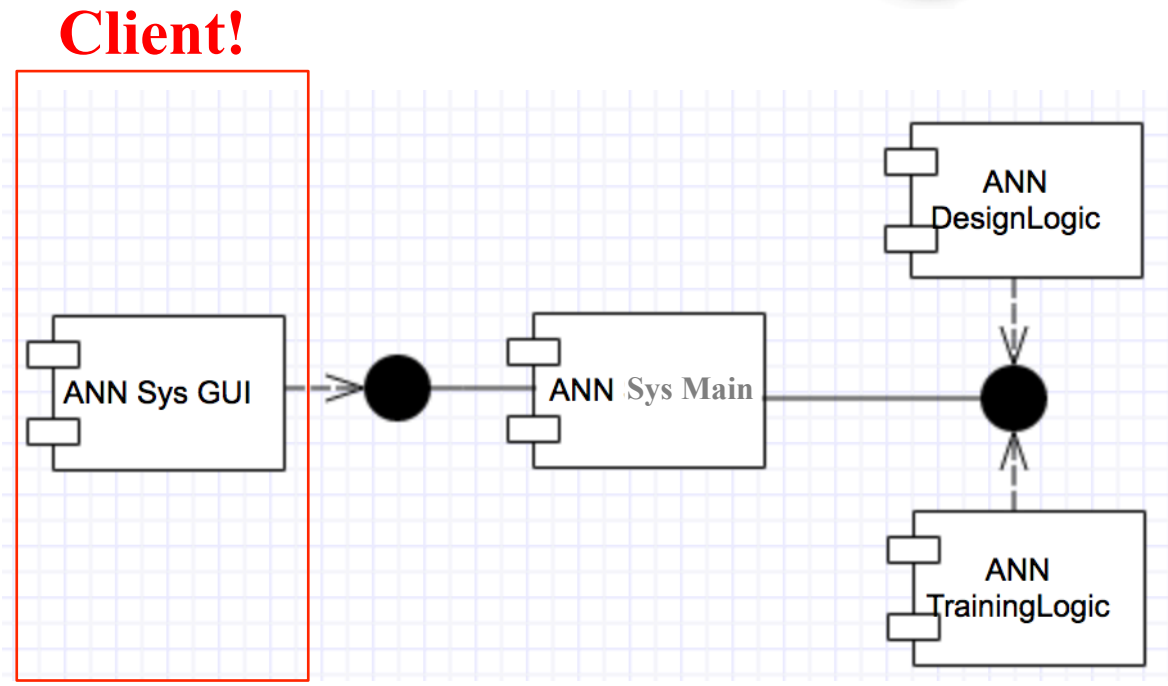
1. **Loose coupling**
2. **Low cohesion**
3. **High modularity**
4. **consistent with Client/server**
5. **...**



Decision 2: let's use Client/Server!

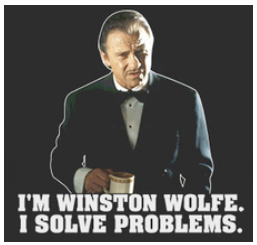
Now... Let's focus on the top-most goal(s)

- So... We need:
  - ANN system;
  - ANN interface;
  - ANN Design logic;
  - ANN Training logic;



Is this good? Probably.

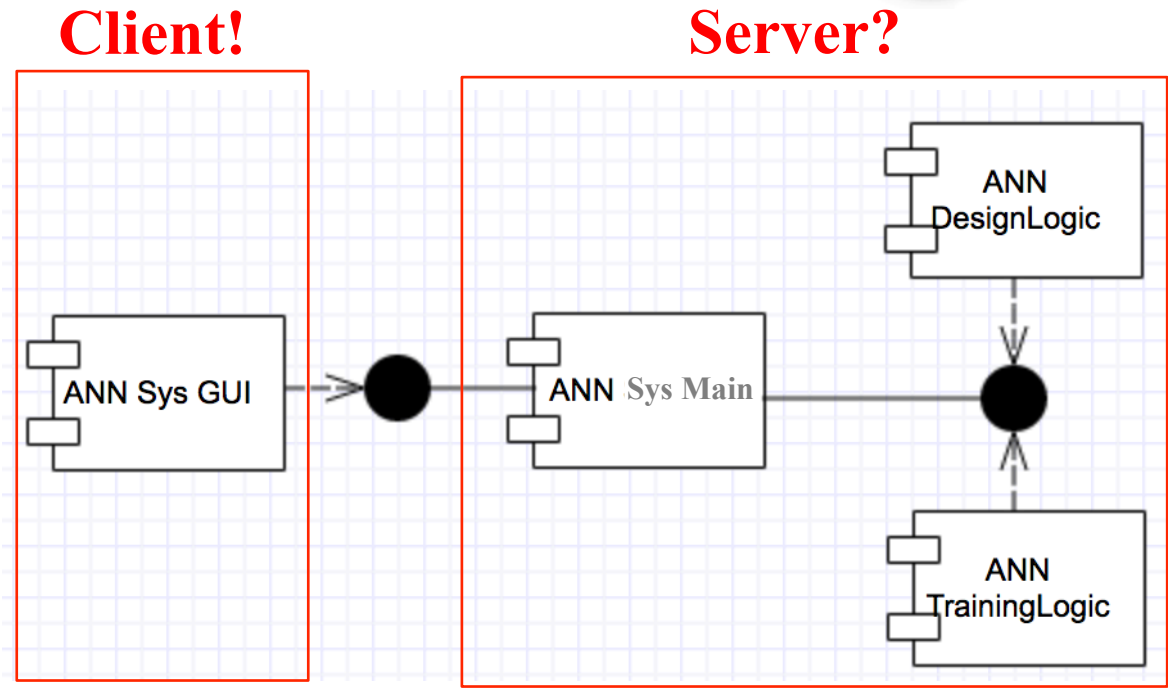
1. **Loose coupling**
2. **Low cohesion**
3. **High modularity**
4. **consistent with Client/server**
5. **...**



Decision 2: let's use Client/Server!

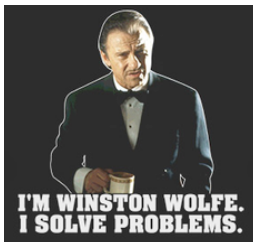
Now... Let's focus on the top-most goal(s)

- So... We need:
  - ANN system;
  - ANN interface;
  - ANN Design logic;
  - ANN Training logic;



Is this good? Probably.

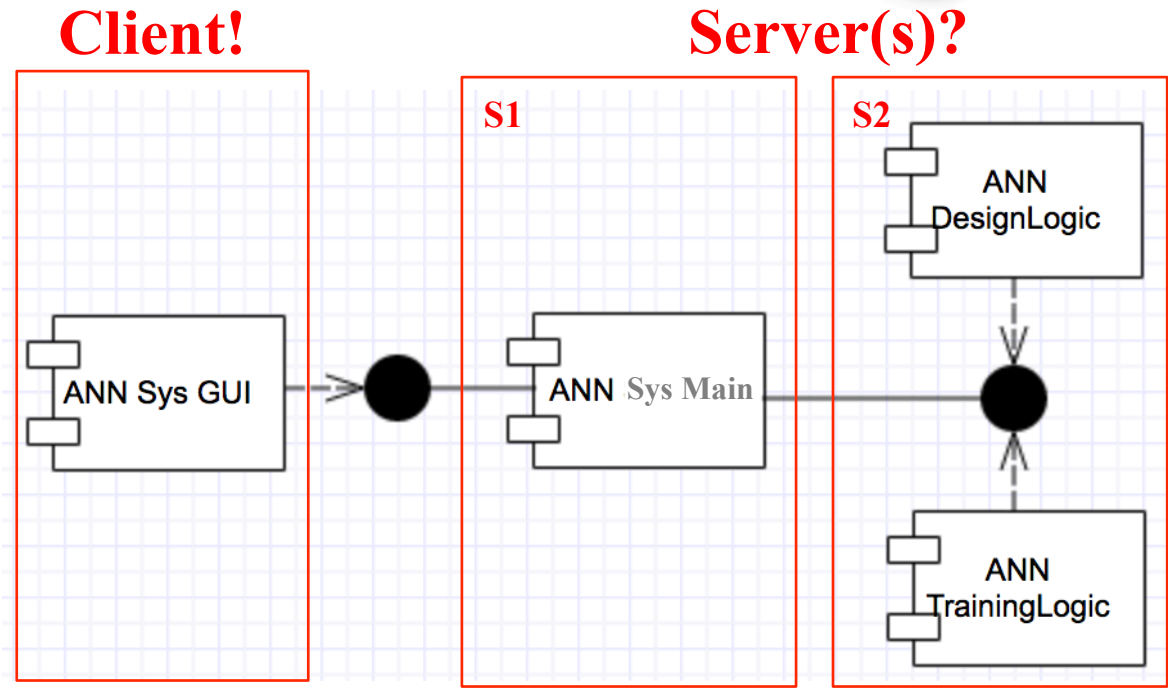
1. **Loose coupling**
2. **Low cohesion**
3. **High modularity**
4. **consistent with Client/server**
5. **...**



Decision 2: let's use Client/Server!

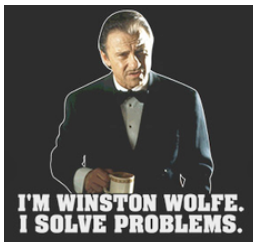
Now… Let's focus on the top-most goal(s)

- So… We need:
  - ANN system;
  - ANN interface;
  - ANN Design logic;
  - ANN Training logic;



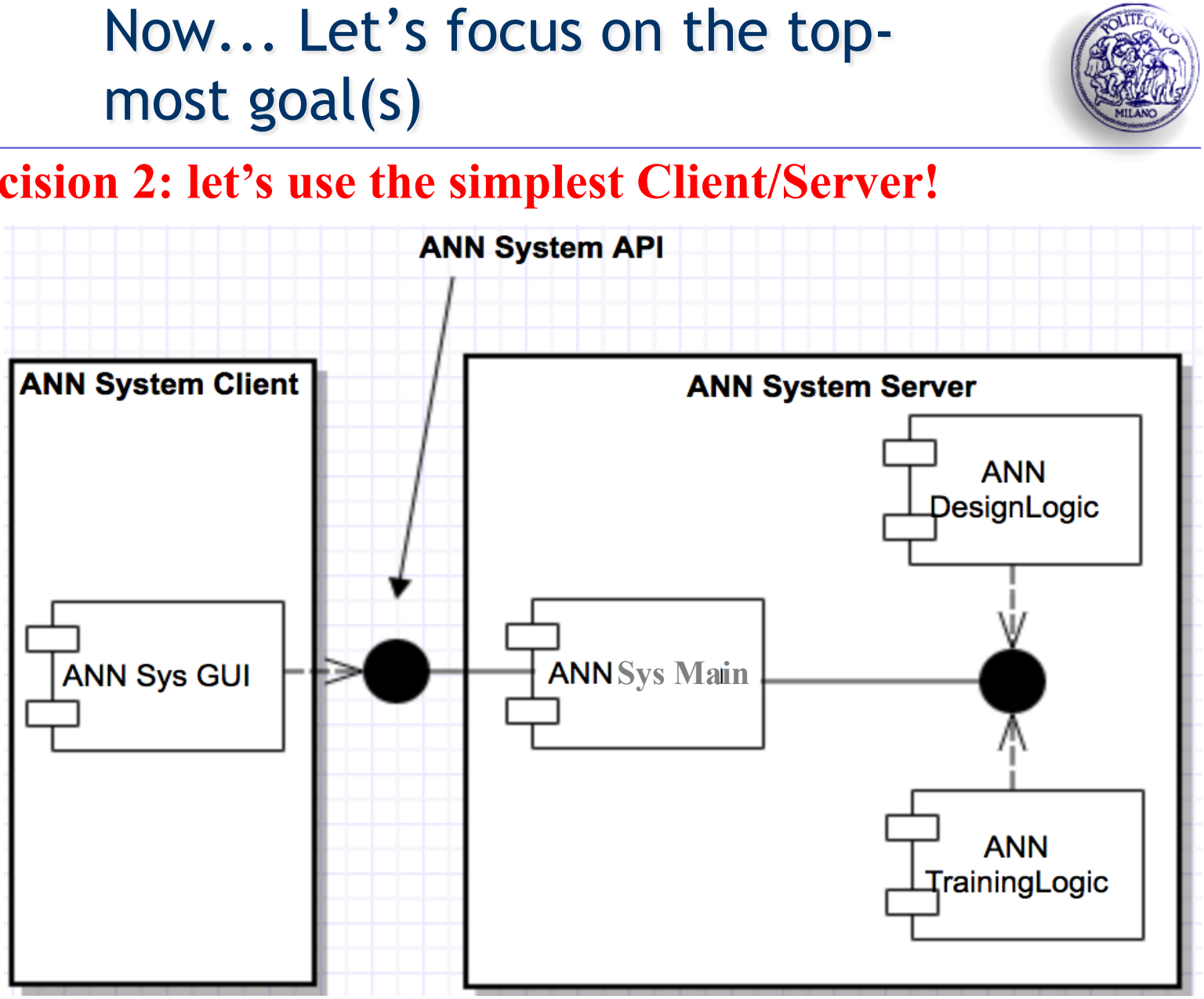
Is this good? Probably.

1. **Loose coupling**
2. **Low cohesion**
3. **High modularity**
4. **consistent with Client/server**
5. **…**



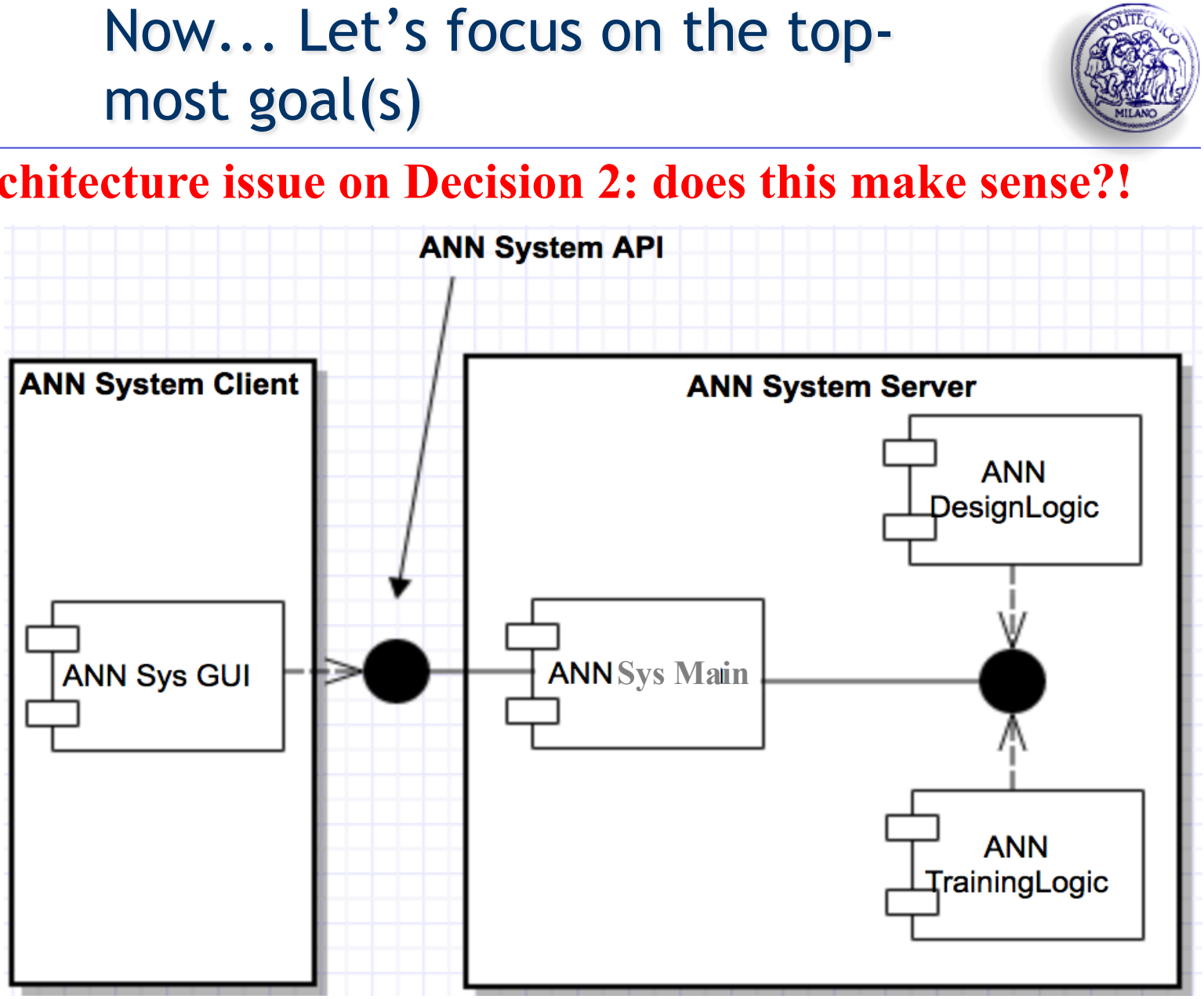
Decision 2: let's use Client/Server!

Now... Let's focus on the top-most goal(s)



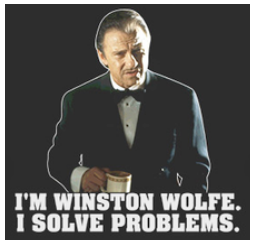
Decision 2: let's use the simplest Client/Server!

Now... Let's focus on the top-most goal(s)

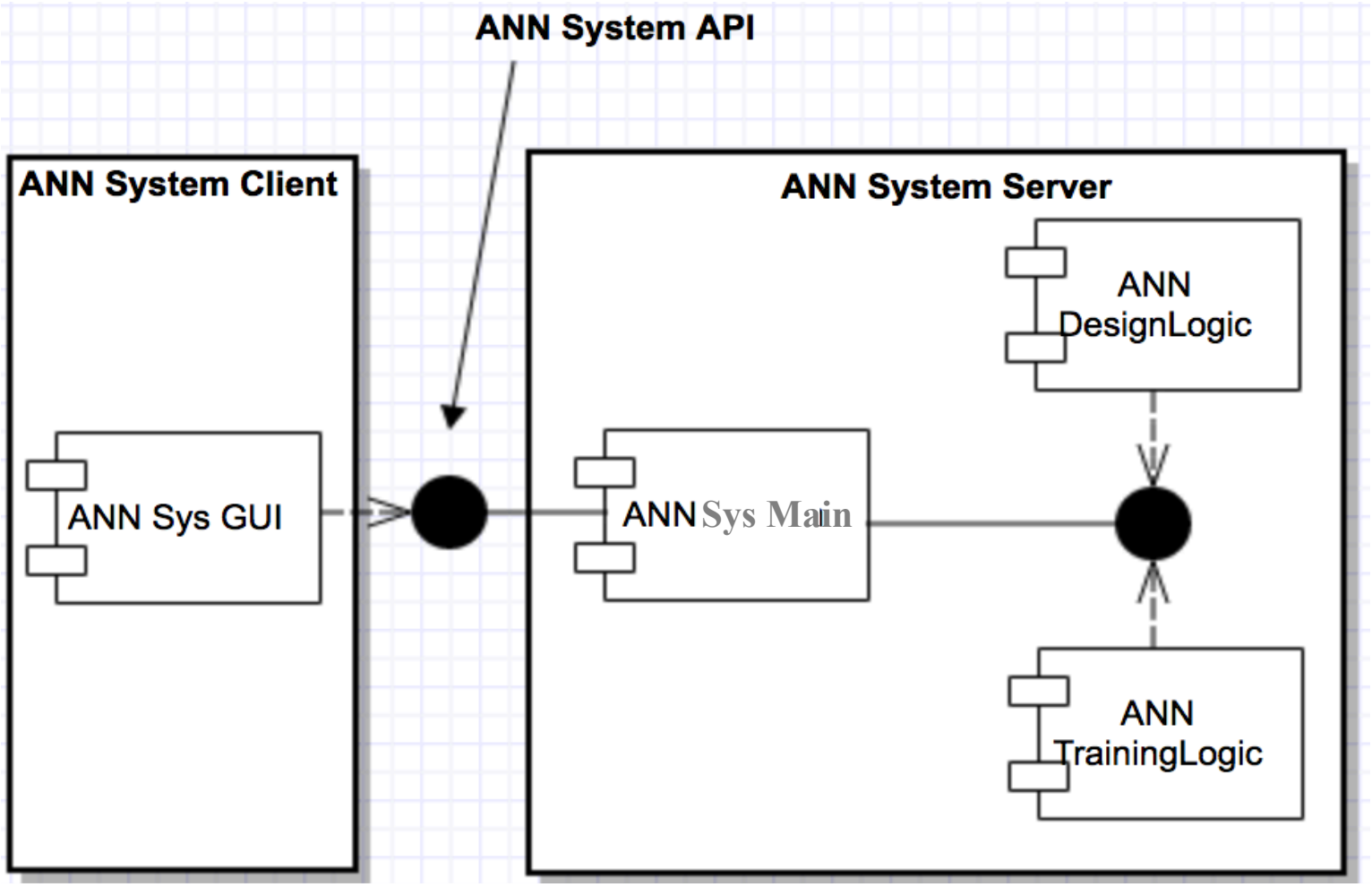


Architecture issue on Decision 2: does this make sense?!

Now... Let's focus on the top-most goal(s)



Decision 3: ...it does! Simple, cohesive, reflects sys. goals...



Now... Let's focus on the top-most goal(s)

Architecture Implementation Issue:

→ **how do I implement ANN System in Client/Server?**

- I'm very good with Python so I'll use that for logic;
- The Client/Server style comes with technology!
  a. HTTP/REST for interfacing;
  b. JavaScript for client logic;
  ...

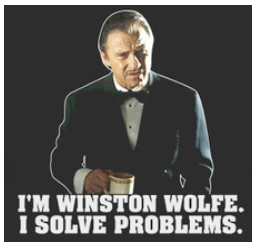
Now... Let's focus on the top-most goal(s)

Architecture Implementation Issue:

→ **how do I implement ANN System in Client/Server?**

- I'm very good with Python so I'll use that for logic;
- The Client/Server style comes with technology!
  a. HTTP/REST for interfacing;
  b. JavaScript for client logic;
  ...

Decision 4: Let's implement with HTTP for interfacing, Python for server logic and JS for client logic



All said and done

- 5 modules:
  - a client (e.g., ocr.js)
  - a server (e.g., server.py)
  - a simple user interface (e.g., ocr.html)
  - an ANN trained via backpropagation (e.g., ocr.py)
  - an ANN design script (e.g., neural_network_design.py)

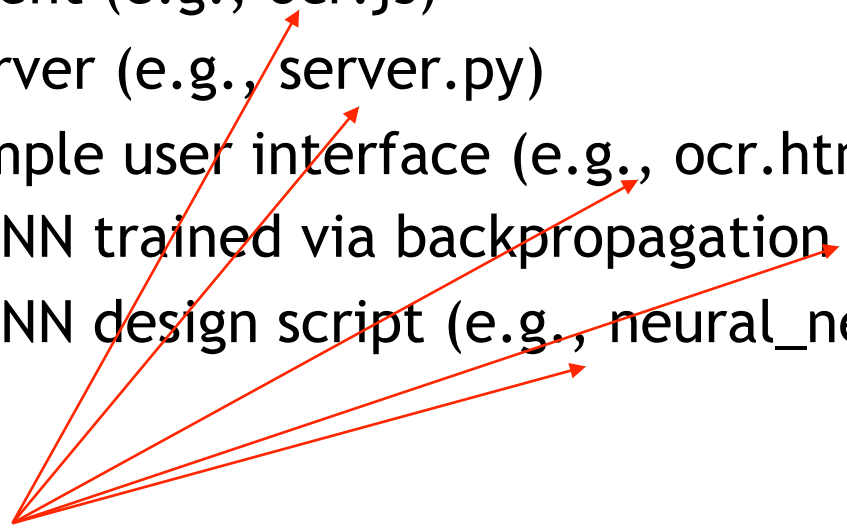
All said and done

- 5 modules:
  - a client (e.g., ocr.js)
  - a server (e.g., server.py)
  - a simple user interface (e.g., ocr.html)
  - an ANN trained via backpropagation (e.g., ocr.py)
  - an ANN design script (e.g., neural_network_design.py)

Hint: always use meaningful names!

All said and done

- 5 modules:
  - a client (e.g., ocr.js)
  - a server (e.g., server.py)
  - a simple user interface (e.g., ocr.html)
  - an ANN trained via backpropagation (e.g., ocr.py)
  - an ANN design script (e.g., neural_network_design.py)



Hint: * Always * use coding standards!

OCR Interface

```
<html>
<head>
   <script src="ocr.js"></script>
   <link rel="stylesheet" type="text/css" href="ocr.css">
</head>
<body onload="ocrDemo.onLoadFunction()">
   <div id="main-container" style="text-align: center;">
      <h1>OCR Demo</h1>
      <canvas id="canvas" width="200" height="200"></canvas>
      <form name="input">
         <p>Digit: <input id="digit" type="text"> </p>
         <input type="button" value="Train" onclick="ocrDemo.train()">
         <input type="button" value="Test" onclick="ocrDemo.test()">
         <input type="button" value="Reset" onclick="ocrDemo.resetCanvas();"/>
      </form>
   </div>
</body>
</html>
```

OCR Client

```
var ocrDemo = {
    CANVAS_WIDTH: 200,
    TRANSLATED_WIDTH: 20,
    PIXEL_WIDTH: 10, // TRANSLATED_WIDTH = CANVAS_WIDTH / PIXEL_WIDTH
     drawGrid: function(ctx) {
         for (var x = this.PIXEL_WIDTH, y = this.PIXEL_WIDTH;
                  x < this.CANVAS_WIDTH; x += this.PIXEL_WIDTH,
                  y += this.PIXEL_WIDTH) {
             ctx.strokeStyle = this.BLUE;
             ctx.beginPath();
             ctx.moveTo(x, 0);
             ctx.lineTo(x, this.CANVAS_WIDTH);
             ctx.stroke();

             ctx.beginPath();
             ctx.moveTo(0, y);
             ctx.lineTo(this.CANVAS_WIDTH, y);
             ctx.stroke();
         }
     },
```

Init signature

Function:
Draw a blue grid;

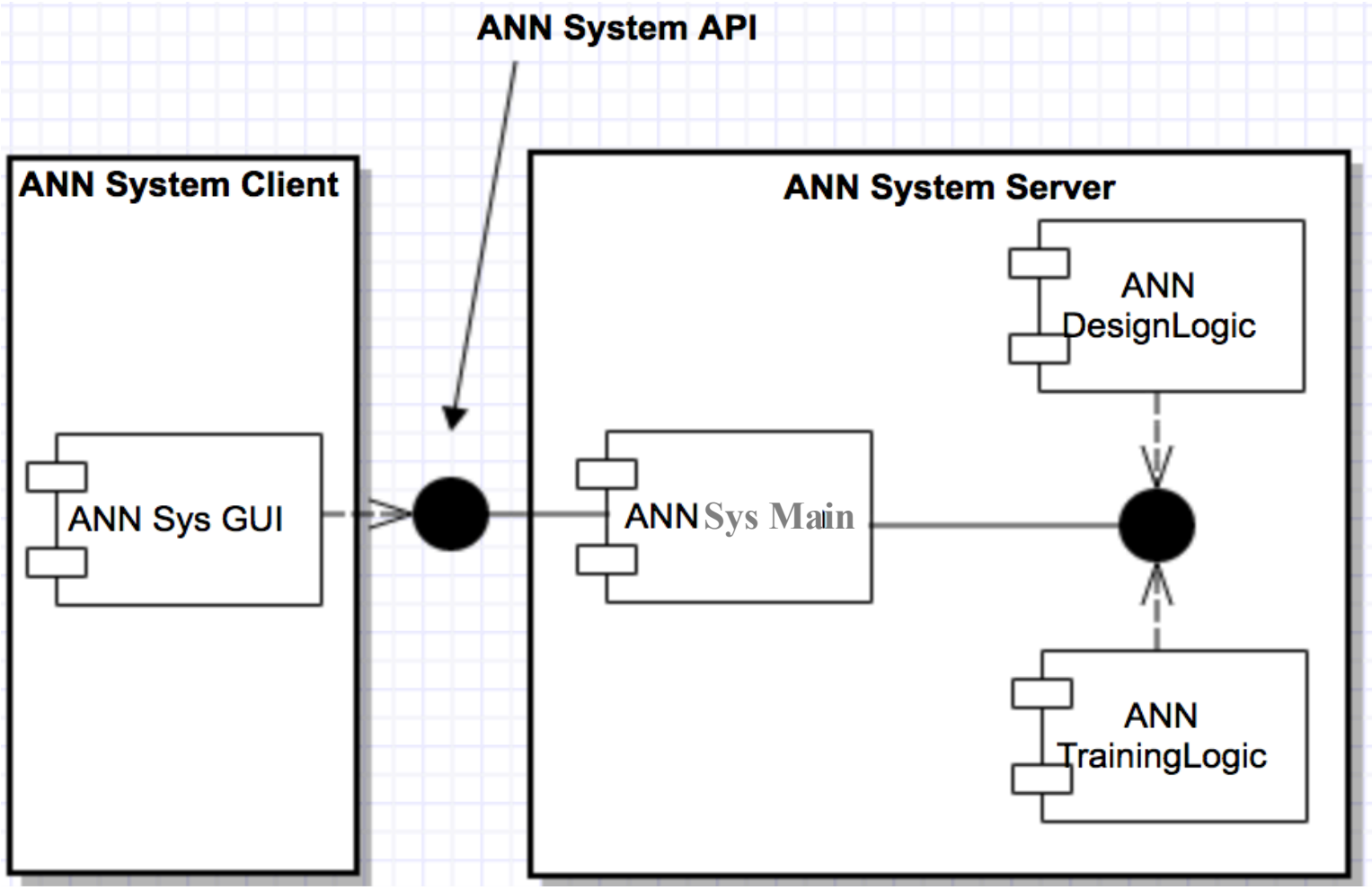
...

OCR Client

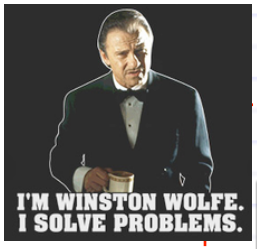
```
var ocrDemo = {
    CANVAS_WIDTH: 200,
    TRANSLATED_WIDTH: 20,
    PIXEL_WIDTH: 10, // TRANSLATED_WIDTH = CANVAS_WIDTH / PIXEL_WIDTH
     drawGrid: function(ctx) {
         for (var x = this.PIXEL_WIDTH, y = this.PIXEL_WIDTH;
                  x < this.CANVAS_WIDTH; x += this.PIXEL_WIDTH,
                  y += this.PIXEL_WIDTH) {
             ctx.strokeStyle = this.BLUE;
             ctx.beginPath();
             ctx.moveTo(x, 0);
             ctx.lineTo(x, this.CANVAS_WIDTH);
             ctx.stroke();

             ctx.beginPath();
             ctx.moveTo(0, y);
             ctx.lineTo(this.CANVAS_WIDTH, y);
             ctx.stroke();
         }
     },
```

Coding Standard!
Using ESLint here

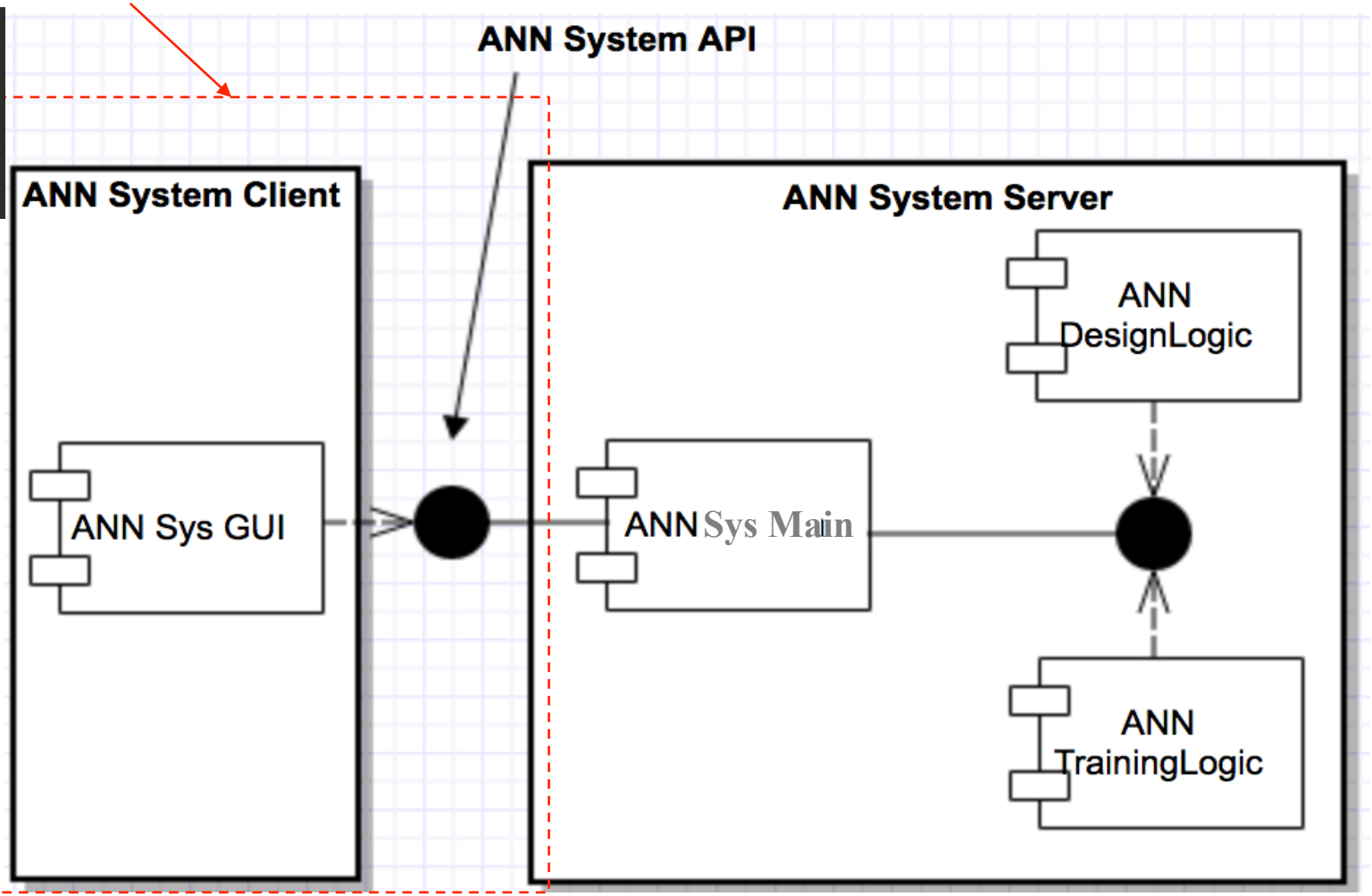
ANN Stakeholder: where are we?



Stakeholder: where are we?

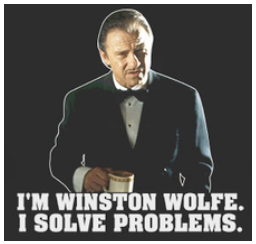


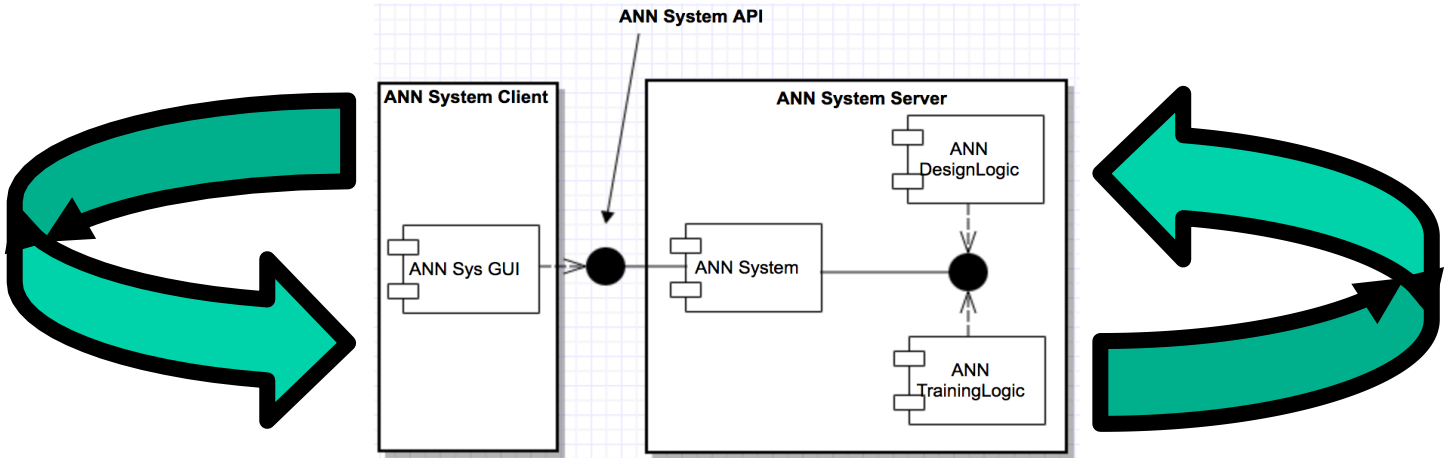
Architect: we are here.



What's next?

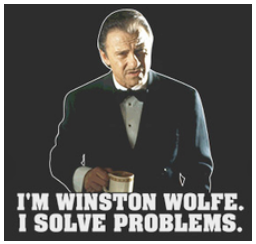
Iterate! Iterate! Iterate!

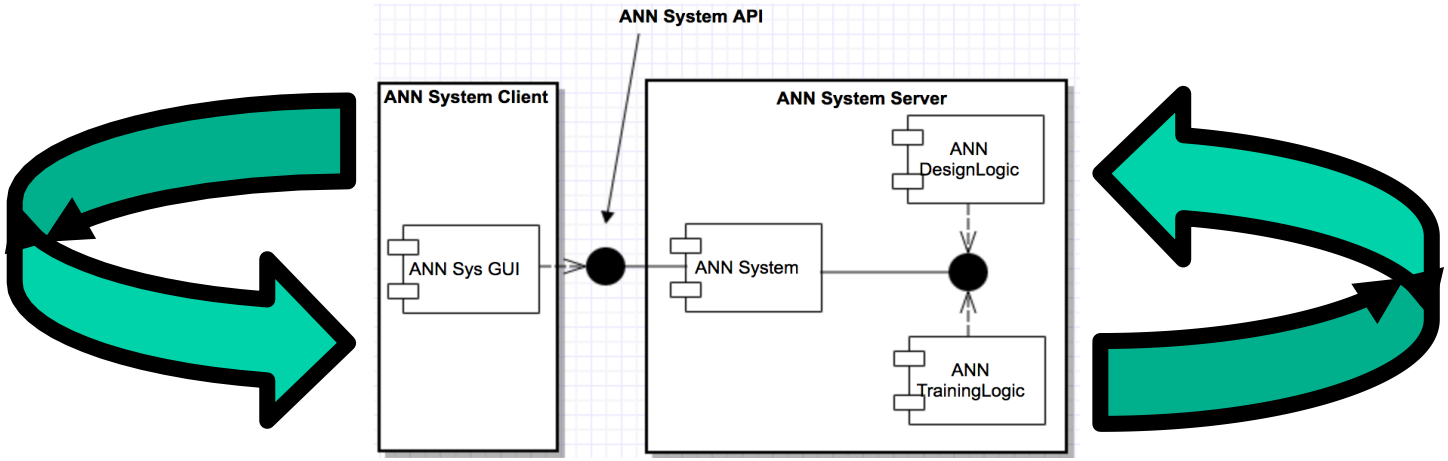




What's next?

Iterate! Iterate! Iterate!





Until we're done developing/integrating/testing...

- Any Questions?

References (Architecture)

- Len Bass, Paul Clements & Rick Kazman, *Software Architecture in Practice*, Addison-Wesley, 1998.
- Frank Buschmann, Régine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stahl, *Pattern-Oriented Software Architecture - A System of Patterns*, Wiley and Sons, 1996.
- Christine Hofmeister, Robert Nord, Dilip Soni, *Applied Software Architecture*, Addison-Wesley 1999.
- Eric Gamma, John Vlissides, Richard Helm, Ralph Johnson, *Design Patterns*, Addison-Wesley 1995.
- Philippe Kruchten, "The 4+1 View Model of Architecture," *IEEE Software*, 12 (6), November 1995, IEEE.
  - http://www.rational.com/support/techpapers/ieee/
- Eberhardt Rechtin, *Systems Architecting: Creating and Building Complex Systems*, Englewood Cliffs NJ, Prentice-Hall, 1991.

References (Architecture)

- Eberhardt Rechtin & Mark Maier, *The Art of System Architecting*, CRC Press, 1997.
- *Recommended Practice for Architectural Description*, Draft 2.0 of IEEE P1471, May 1998
  - http://www.pithecanthropus.com/~awg/
- Mary Shaw, and David Garlan, *Software Architecture—Perspectives on an Emerging Discipline*, Upper Saddle River, NJ, Prentice-Hall, 1996.
- Bernard I. Witt, F. Terry Baker, and Everett W. Merritt, *Software Architecture and Design—Principles, Models, and Methods*, New York NY, Van Nostrand Reinhold, 1995.
- The World-wide Institute of Software Architects
  - http://www.wwisa.org

References (UML4Arch)

- Grady Booch, James Rumbaugh, Ivar Jacobson, *The Unified Modeling Language User Guide*, Addison-Wesley, 1999.