WADS-DSN 2004 Invited Talk

From Dependable Architectures To Dependable Systems

Nenad Medvidovic

Computer Science Department
University of Southern California
Los Angeles, U.S.A.
neno@usc.edu

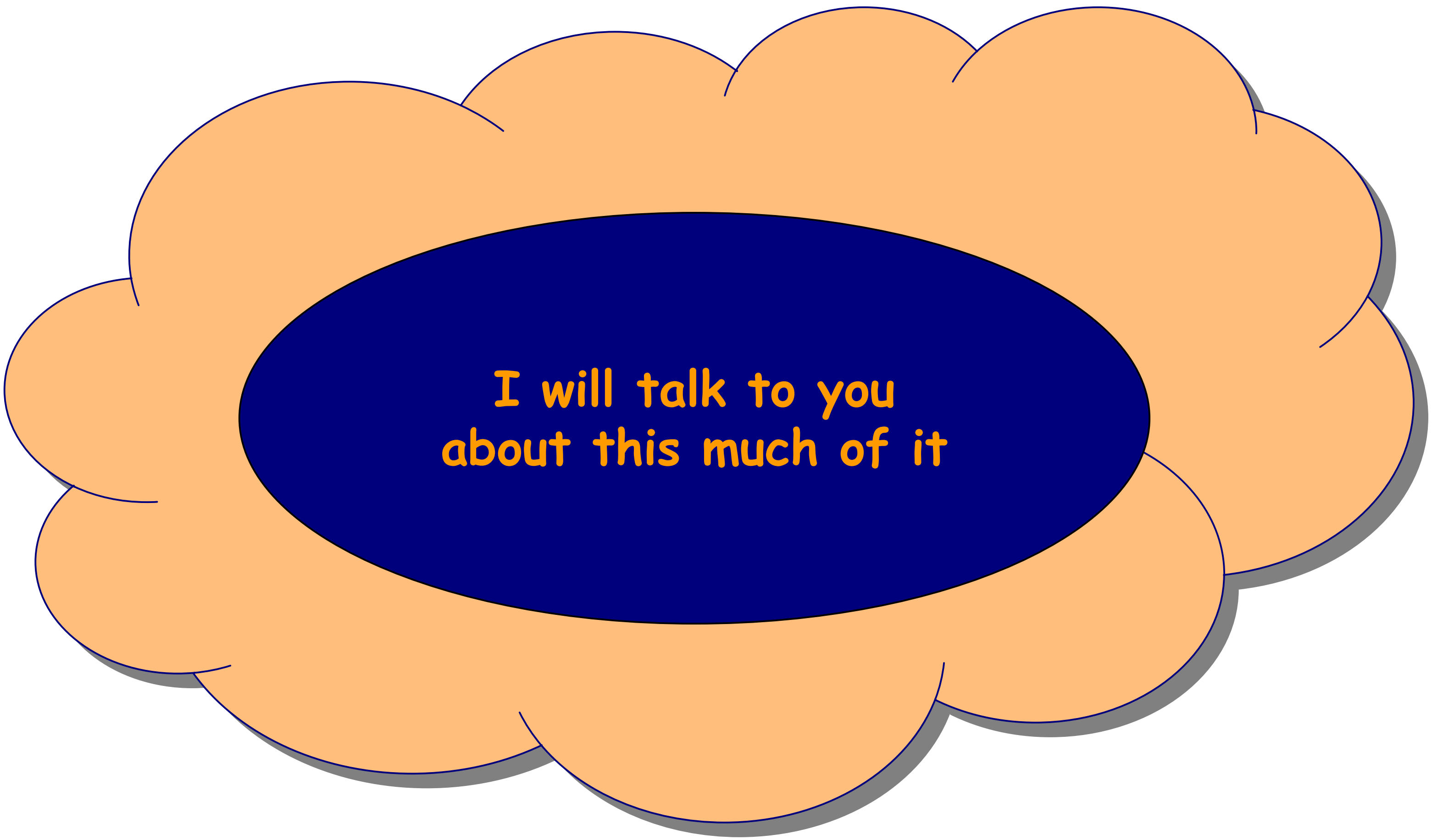
Goals of the Talk

- Identify some challenges
- Suggest some solutions
- Motivate future research
- Invite dissenting opinions

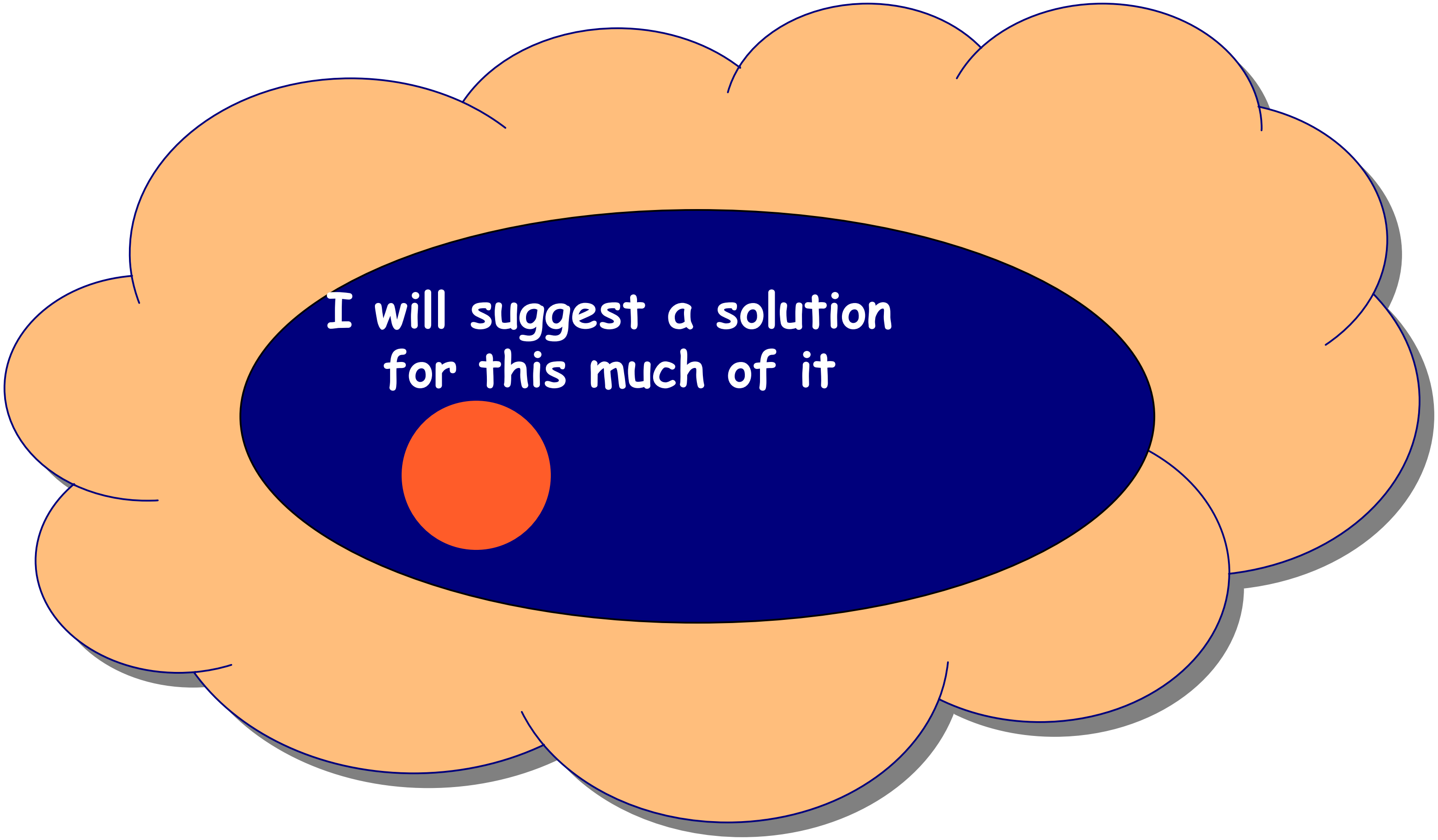
Goals of the Talk

The problem is this big and sometimes ill-defined

Goals of the Talk



Goals of the Talk



Go

But, hopefully, we will have this much fun!

What Is Dependability?

- Degree of user confidence that the system will operate as expected
- Key dimensions
  - Availability
  - Reliability
  - Security
  - Safety
- But also
  - Repairability
  - Maintainability
  - Survivability
  - Fault tolerance

What Is Architecture?

- A high-level model of a system
  - The system's "blueprint"
- Represents system organization
  - Data
  - Computation
  - Interaction
  - Structure
- Embodies system properties
  - Communication integrity, performance, throughput, liveness, . . .
  - Can/does it embody dependability?
  - (how) Can those properties be transferred to the system itself?

A "Traditional" Architectural Model

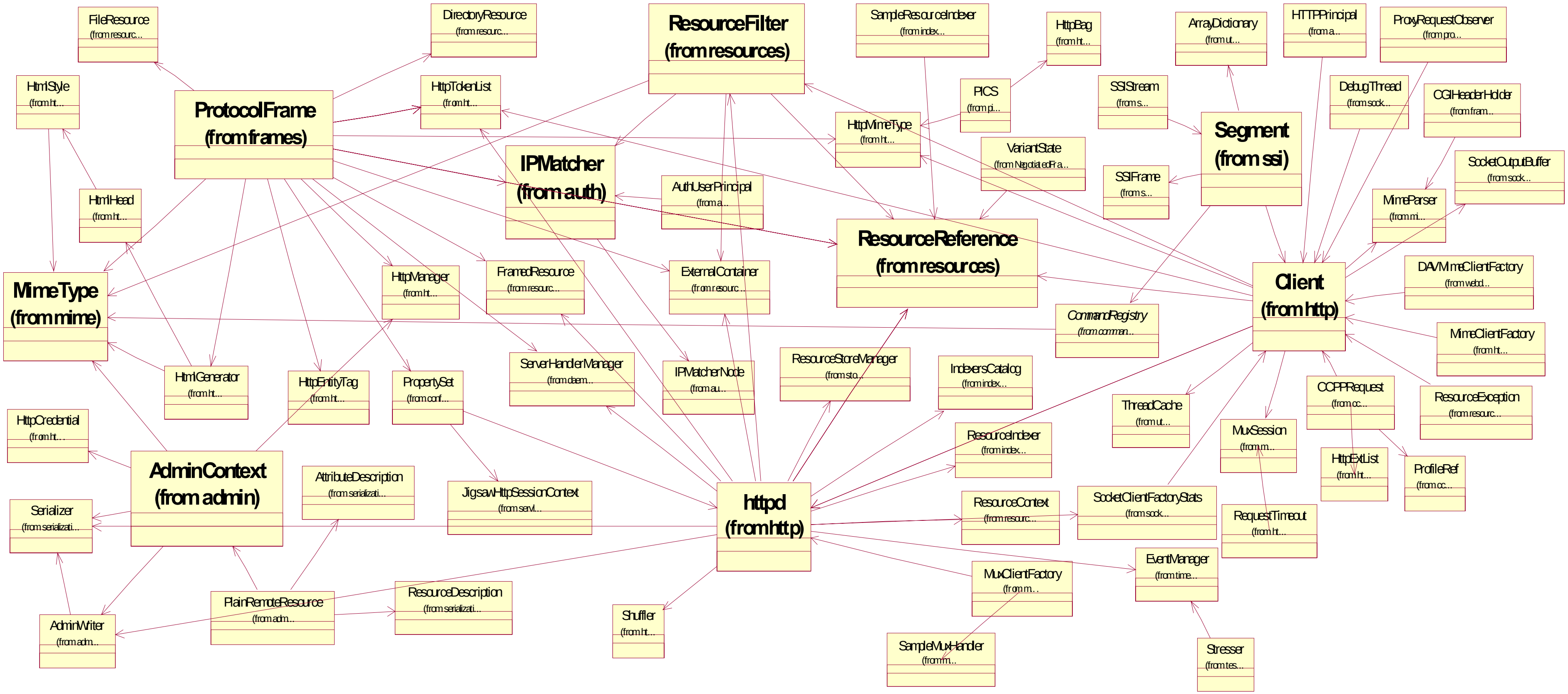
```
connector Pipe =
  role Writer = write → Writer ⊓ close → √
  role Reader =
    let ExitOnly = close → √
    in let DoRead = (read → Reader
                  [] read-eof → ExitOnly)
    in DoRead ⊓ ExitOnly
  glue = let ReadOnly = Reader.read → ReadOnly
                  [] Reader.read-eof → Reader.close → √
                  [] Reader.close → √
       in let WriteOnly = Writer.write → WriteOnly
                  [] Writer.close → √
       in Writer.write → glue
        [] Reader.read → glue
        [] Writer.close → ReadOnly
        [] Reader.close → WriteOnly
```

A "Traditional" Architectural Model

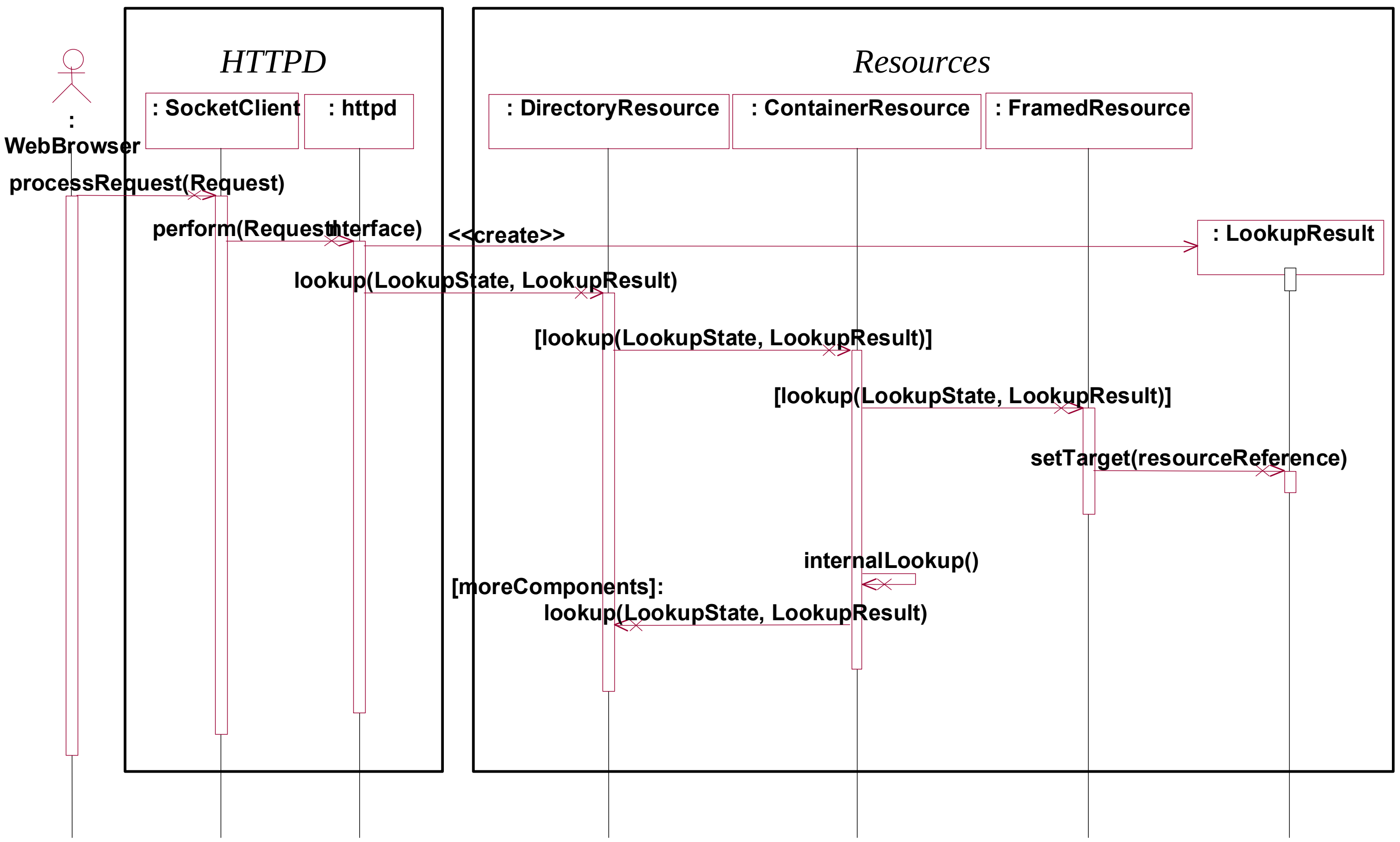
```
connector Pipe =
  role Writer = write → Writer ⊓ close → √
  role Reader =
    let ExitOnly = close → √
    in let DoRead = (read → Reader
                  [] read-eof → ExitOnly)
    in DoRead ⊓ ExitOnly
  glue = let ReadOnly = Reader.read → ReadOnly
                  [] Reader.read-eof → Reader.close → √
                  [] Reader.close → √
      in let WriteOnly = Writer.write → WriteOnly
                  [] Writer.close → √
      in Writer.write → glue
       [] Reader.read → glue
       [] Writer.close → ReadOnly
       [] Reader.close → WriteOnly
```

What/where is the dependability?

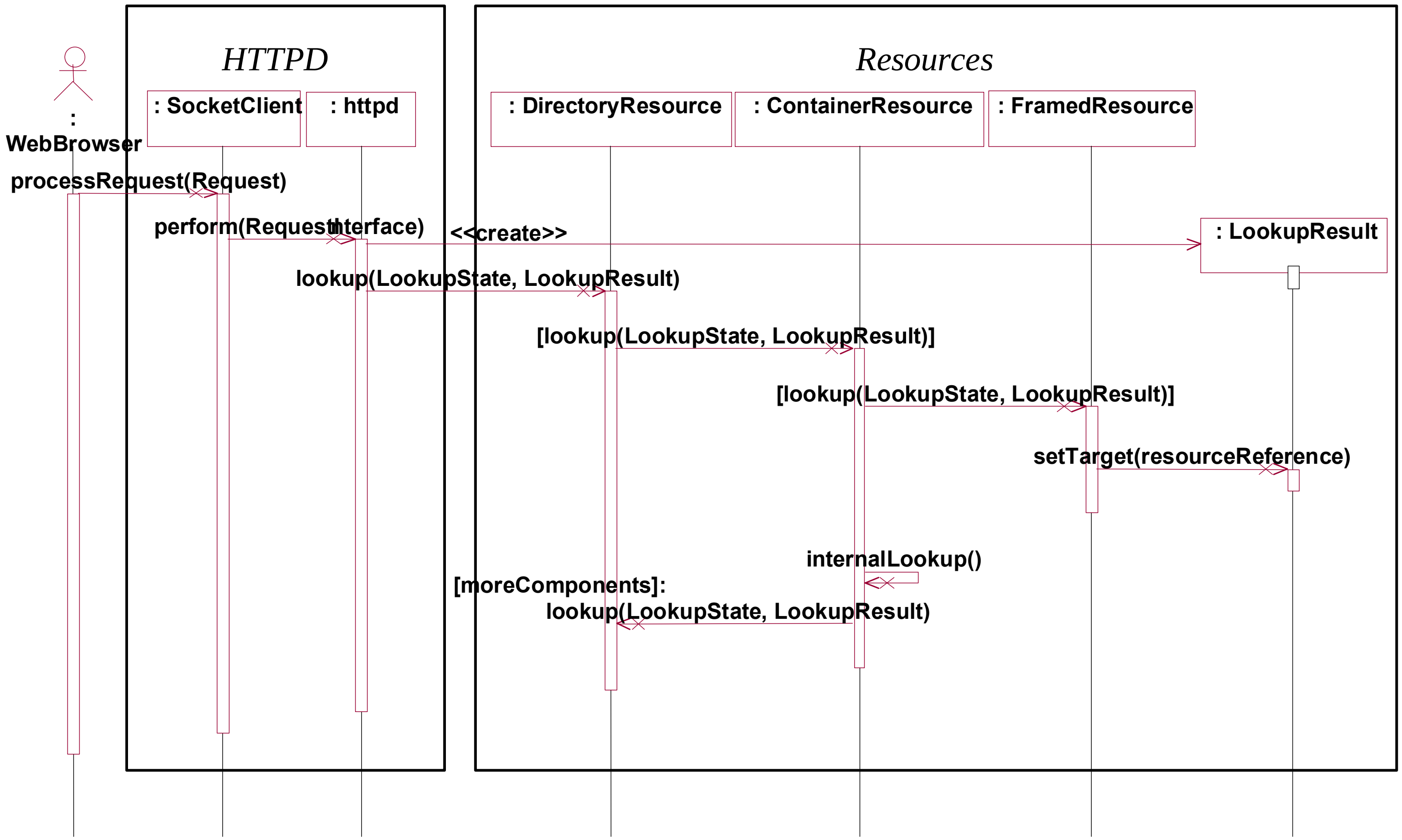
A "Standard" Architectural Model



A "Standard" Architectural Model



A "Standard" Architectural Model



What/where is the dependability?

But, We Can Model Anything

- Meta-H, ROOM, UniCon, etc. can help ensure real-time properties in models
- Markov chains can help ensure reliability in models
- Multi-versioning connectors can help ensure fault tolerance
- Code/data mobility and replication formalisms can help ensure availability
- . . .

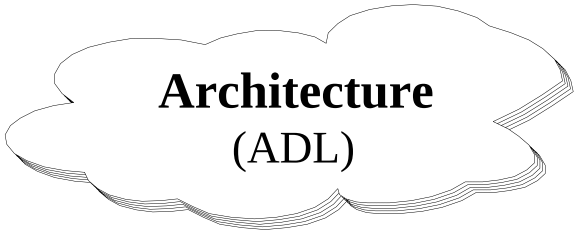
But, We Can Model Anything

- Meta-H, ROOM, UniCon, etc. can help ensure real-time properties in models
- Markov chains can help ensure reliability in models
- Multi-versioning connectors can help ensure fault tolerance
- Code/data mobility and replication formalisms can help ensure availability
- . . .

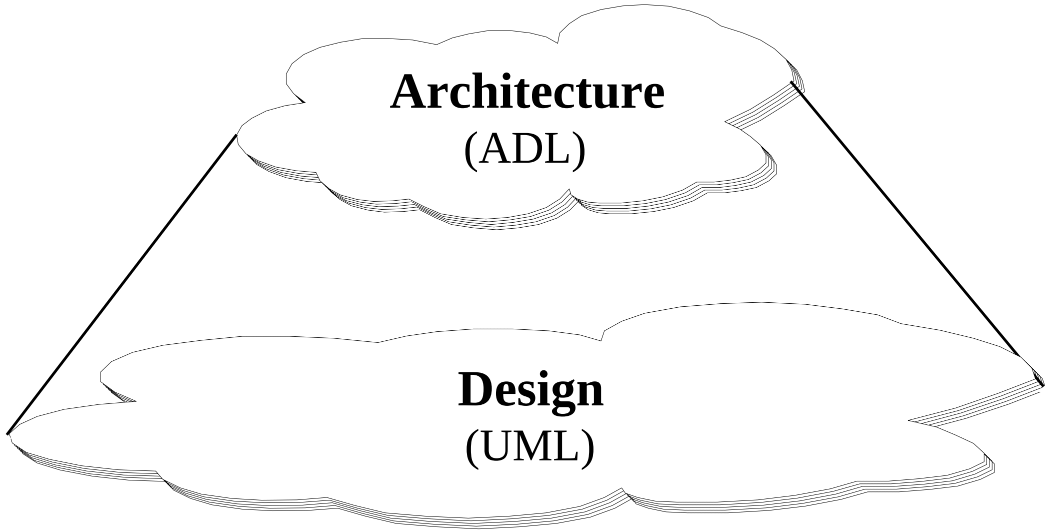
So then the problem is solved, right?

Why the Problem Isn't Solved

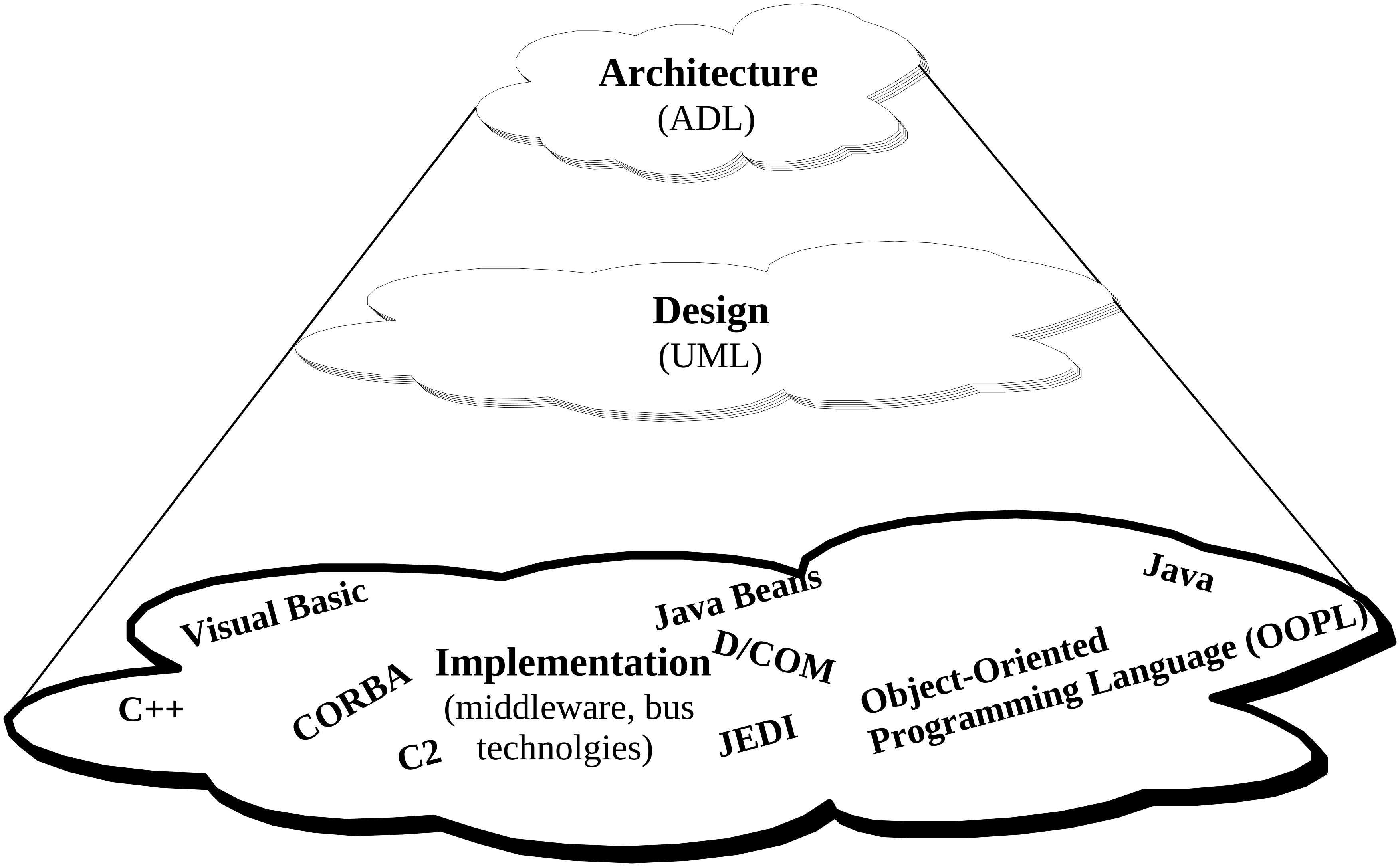
Why the Problem Isn't Solved



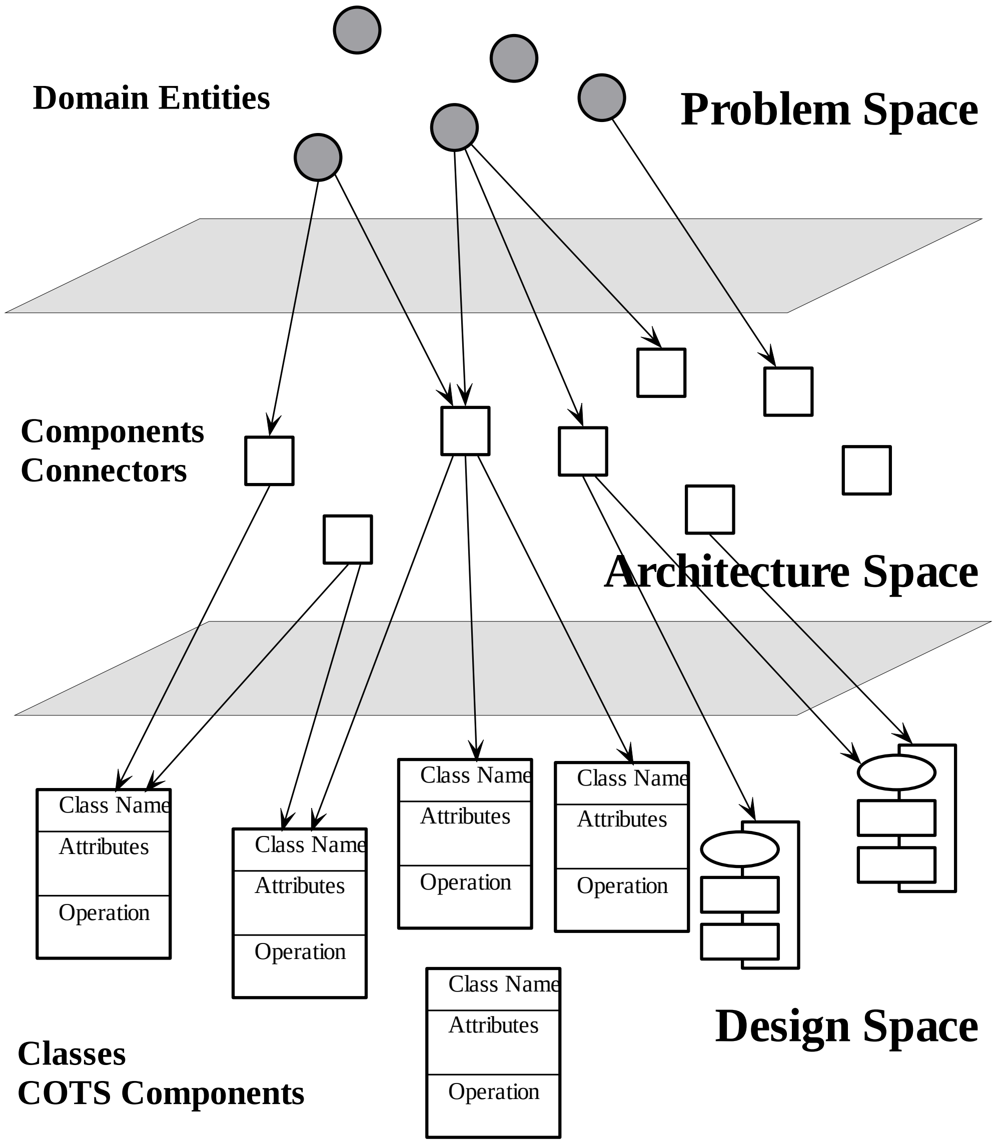
Why the Problem Isn't Solved



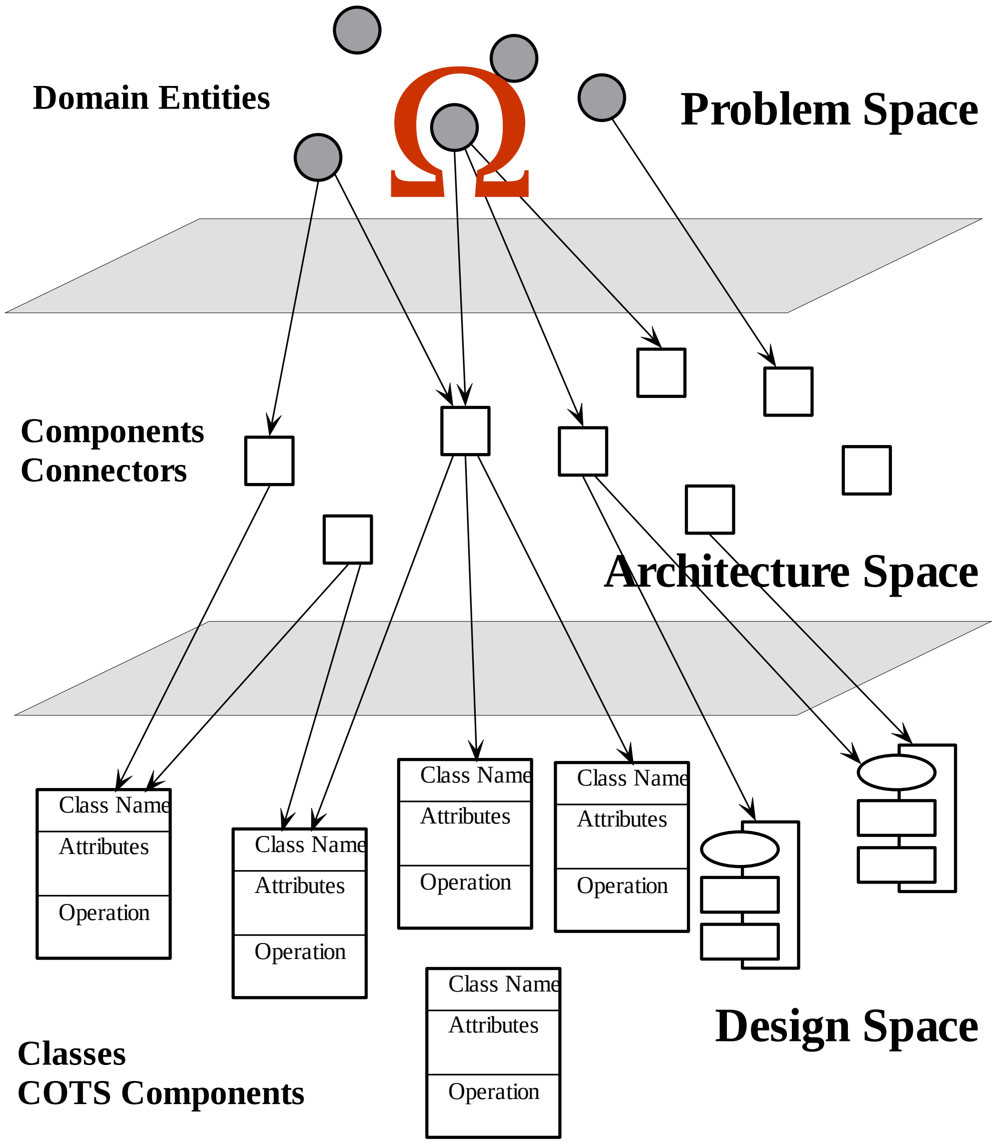
Why the Problem Isn't Solved



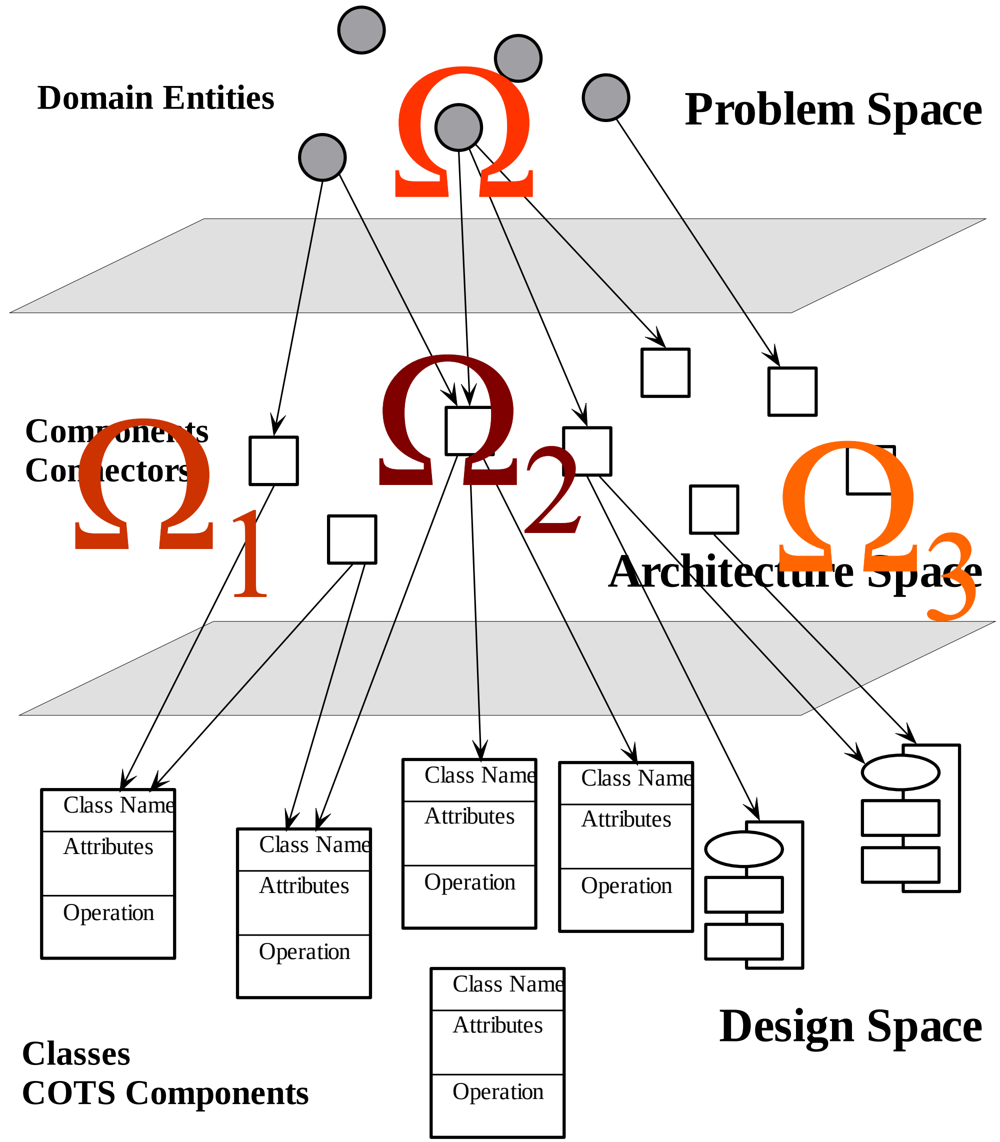
Why the Problem Isn't Solved



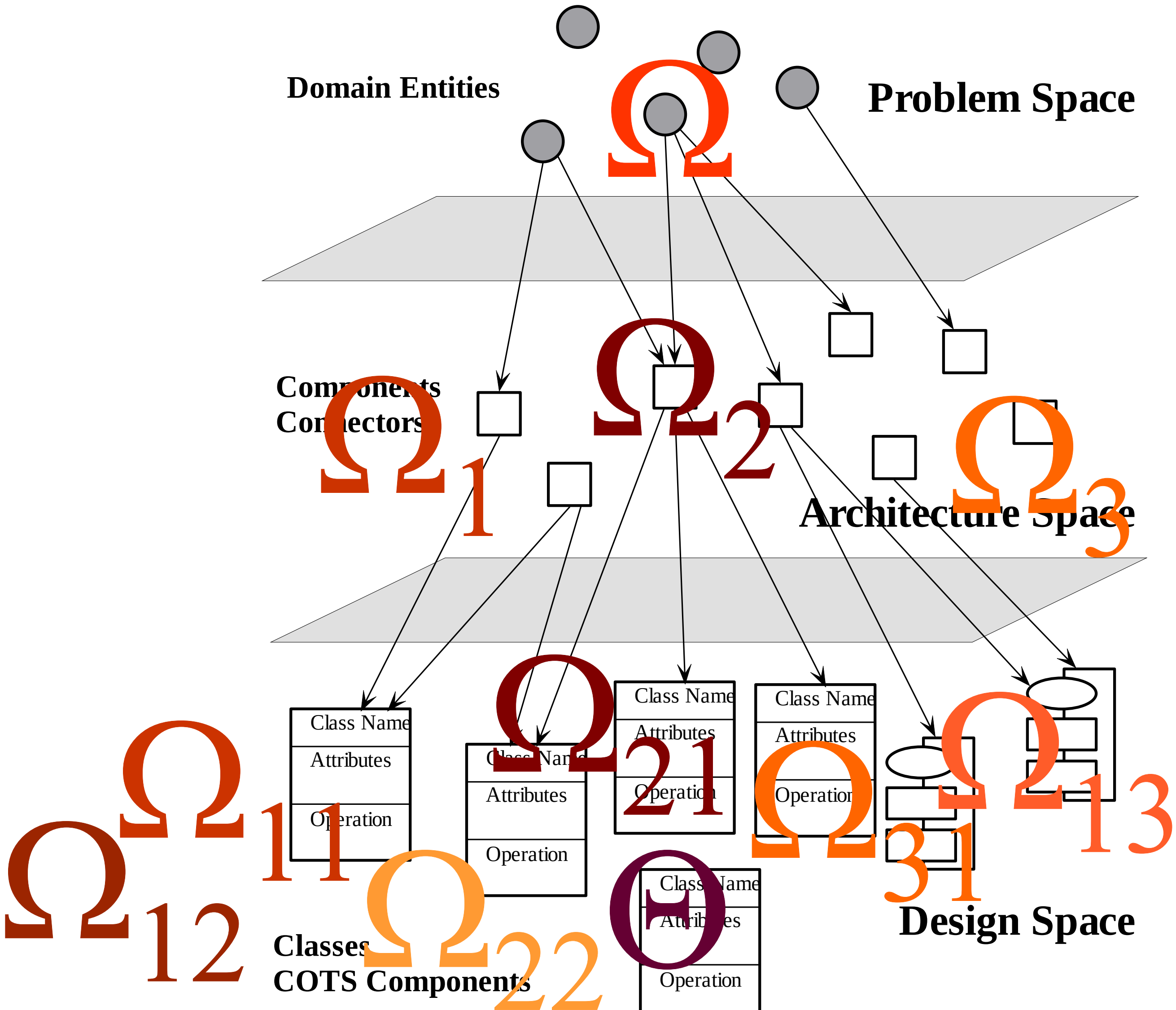
Why the Problem Isn't Solved



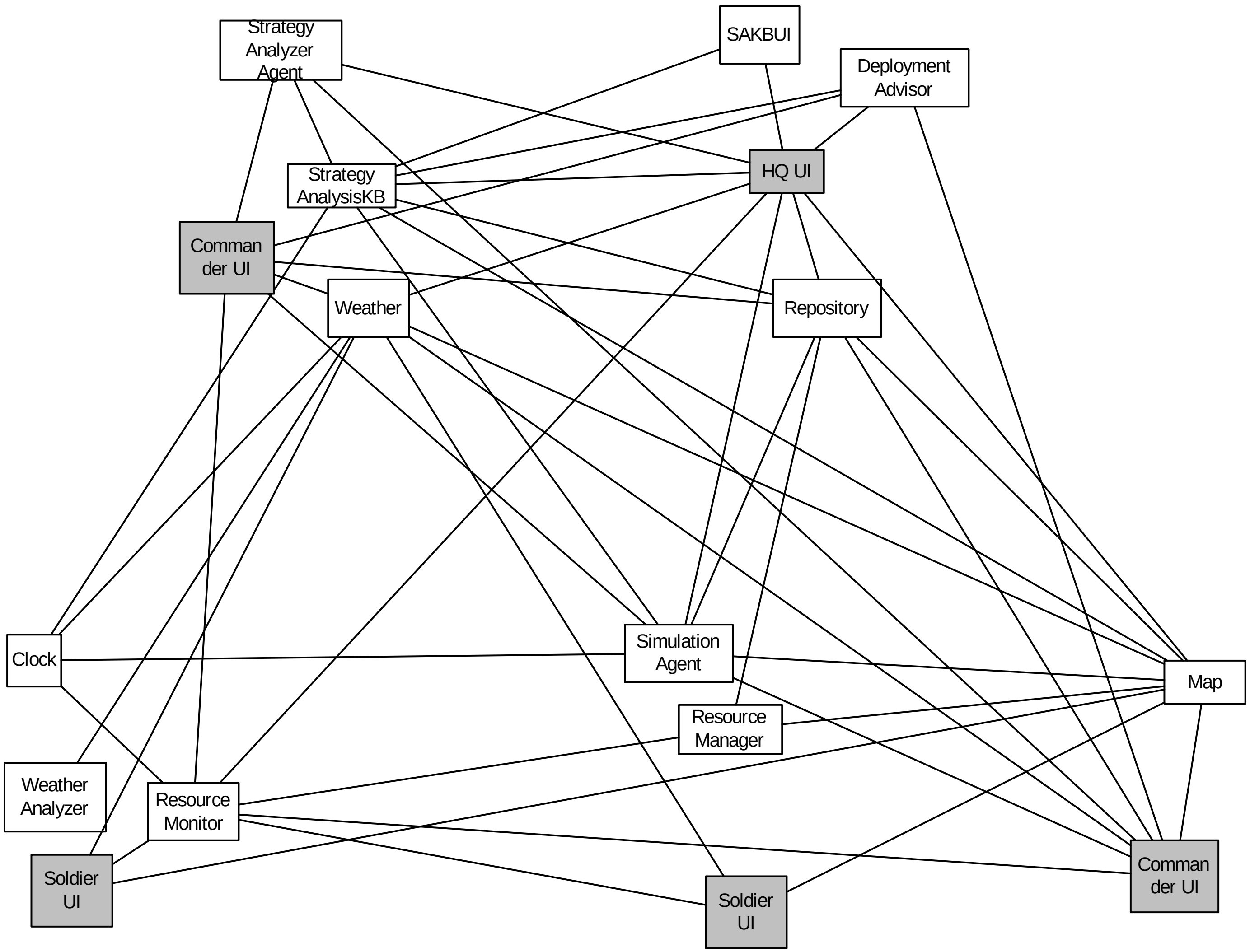
Why the Problem Isn't Solved



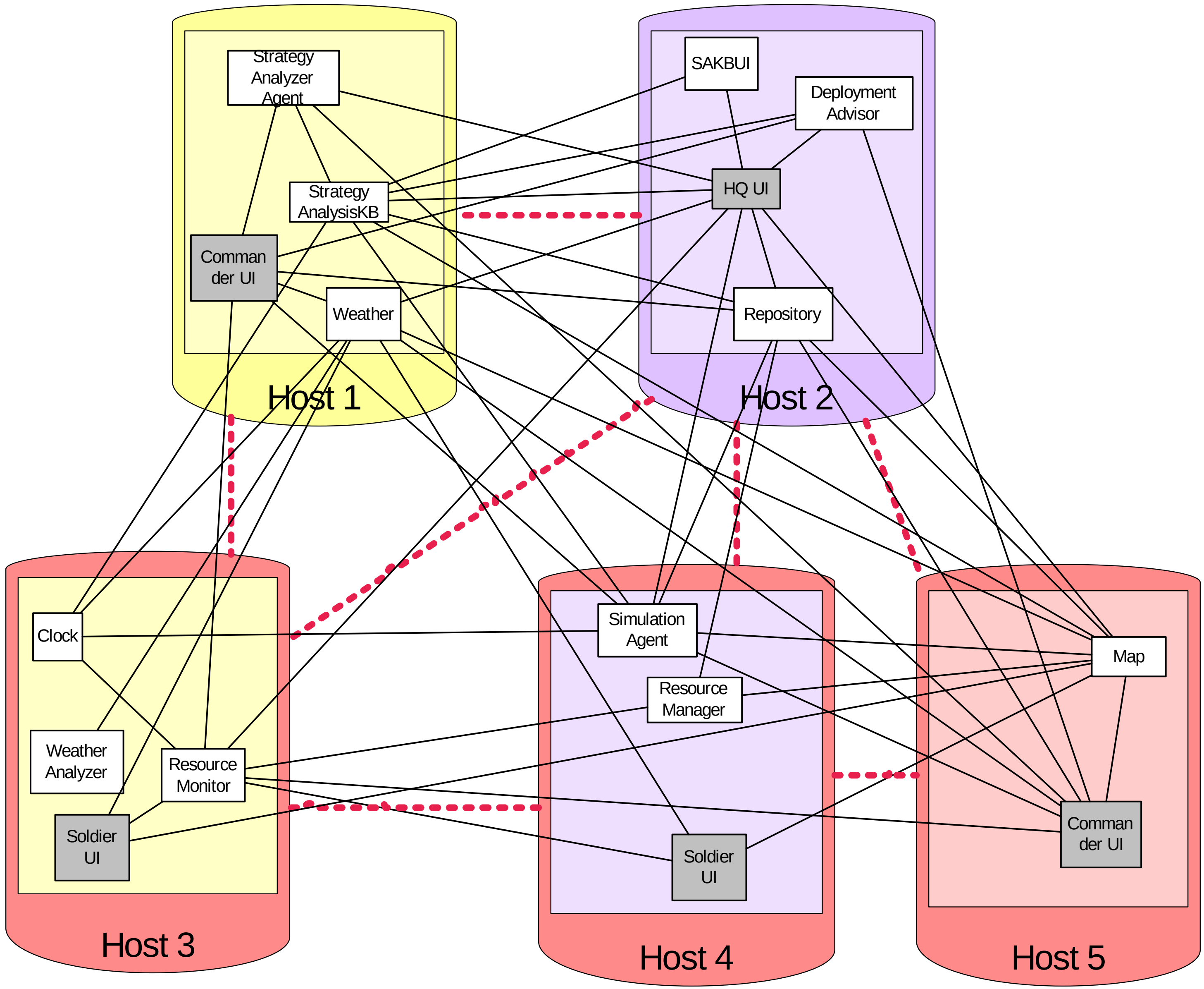
Why the Problem Isn't Solved



From Models to Systems



From Models to Systems

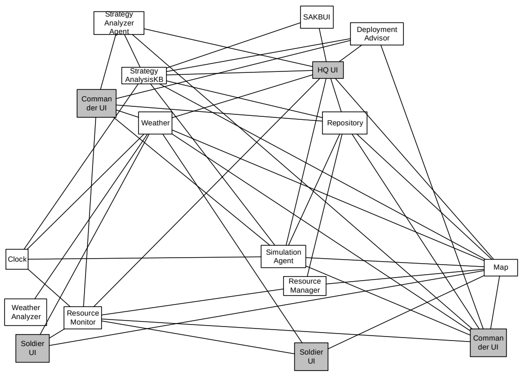


From Models to Systems

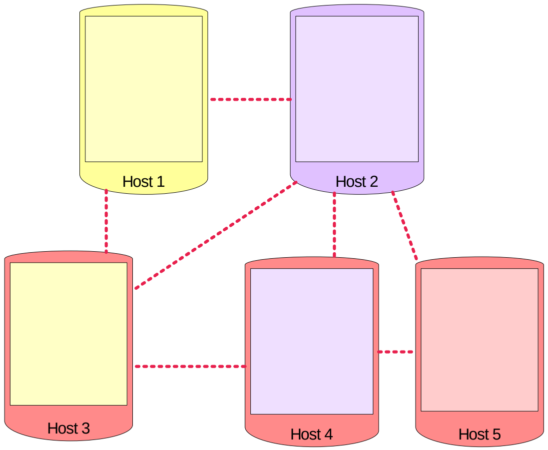
The remainder of this talk will focus on two key questions:

1. How do we get from

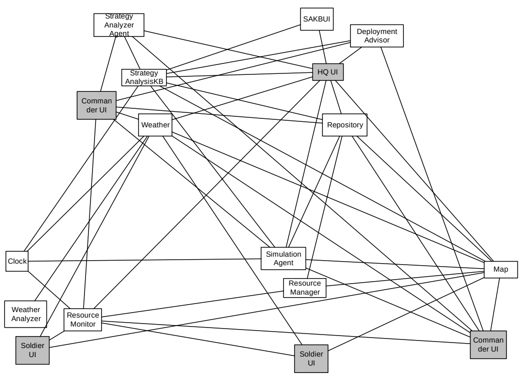
1. How do we get from



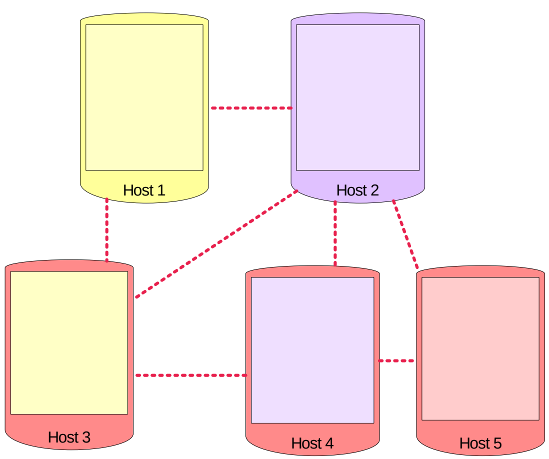
and



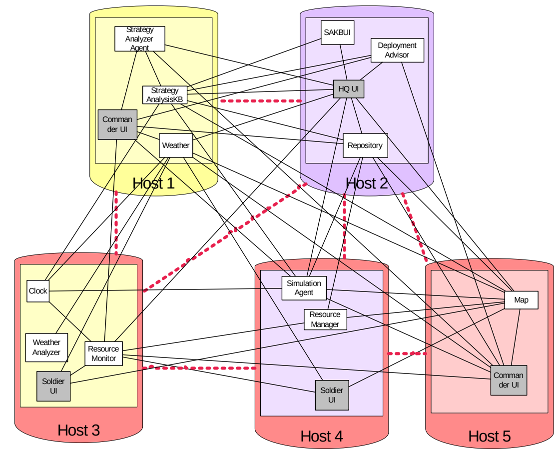
1. How do we get from



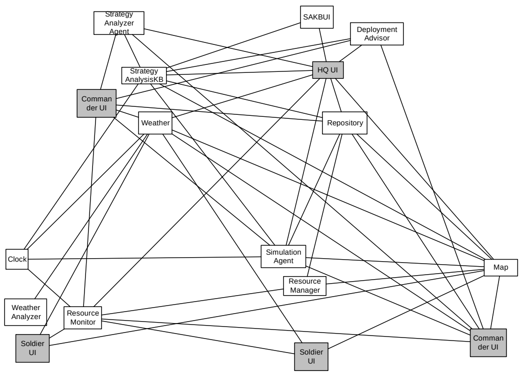
and



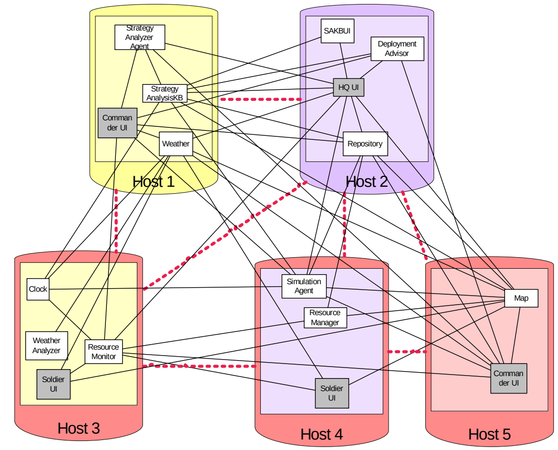
to



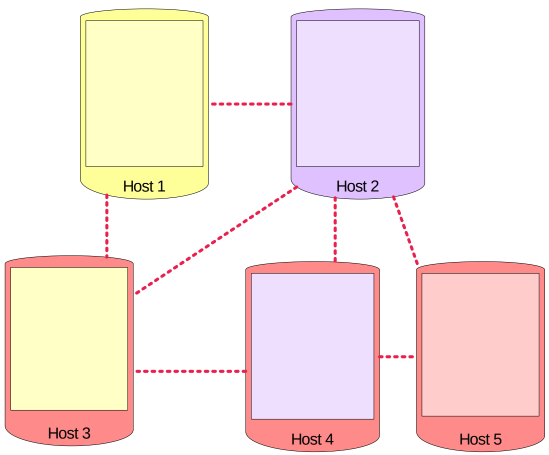
1. How do we get from



to



and

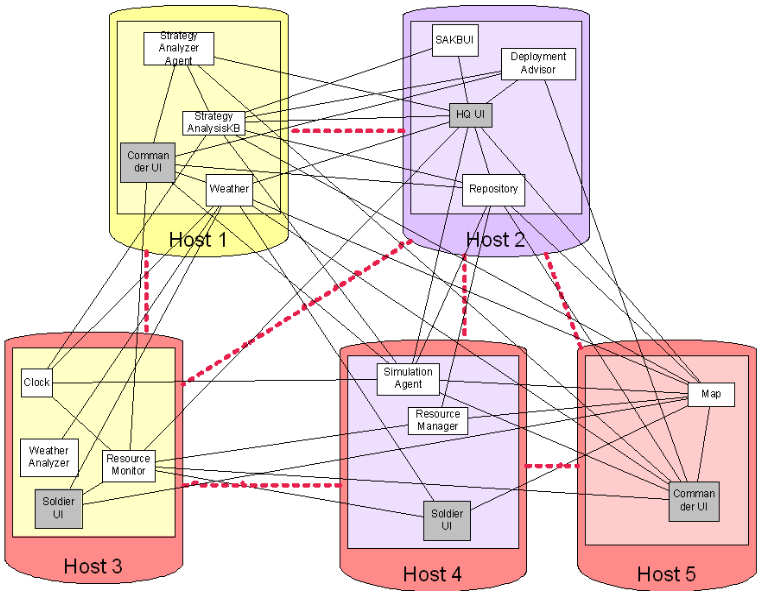


to

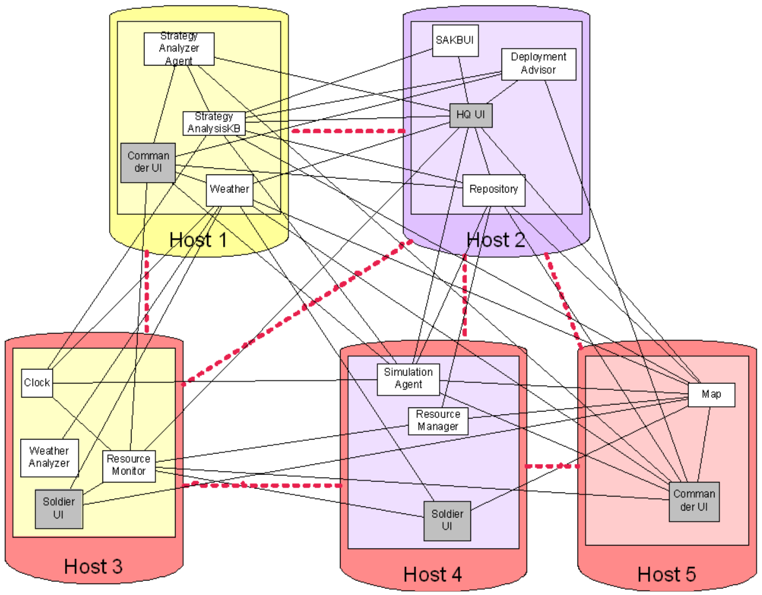
?

2. How do we know

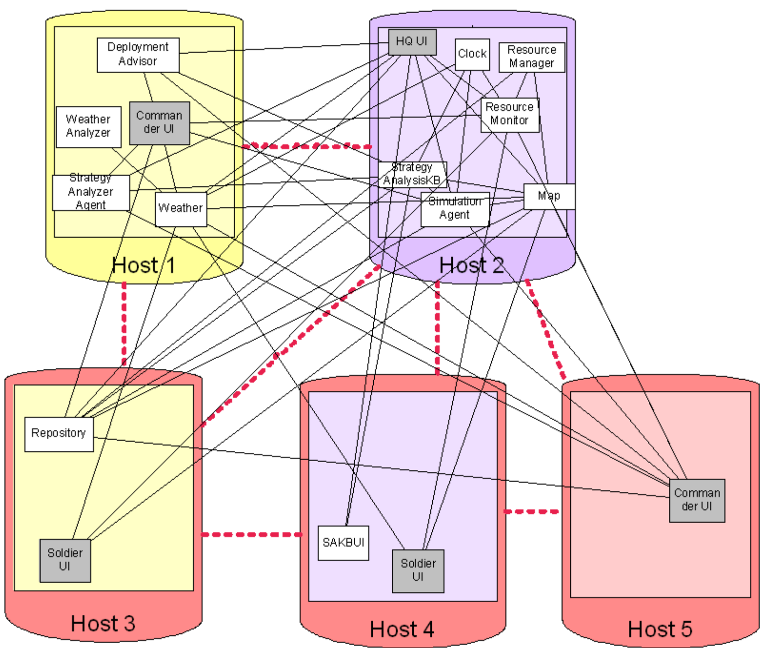
2. How do we know



2. How do we know



is “better” than



?

Outline

- From architectures to systems
- Ensuring dependability
  - Problem definition
  - Proposed solution
- Concluding remarks

Outline

- From architectures to systems
- Ensuring dependability
  - Problem definition
  - Proposed solution
- Concluding remarks

How Do I Dependably Implement an Architecture?

- Architectures provide *high-level* concepts
  - Components, connectors, ports, events, configurations
- Programming languages provide *low-level* constructs
  - Variables, arrays, pointers, procedures, objects
- Bridging the two often is an art-form
  - Middleware can help "split the difference"
- Existing middleware technologies
  - Support some architectural concepts (e.g., components, events)
  - but not others (e.g., connectors, configurations)
  - Impose particular architectural styles

How Do I Dependably Implement an Architecture?

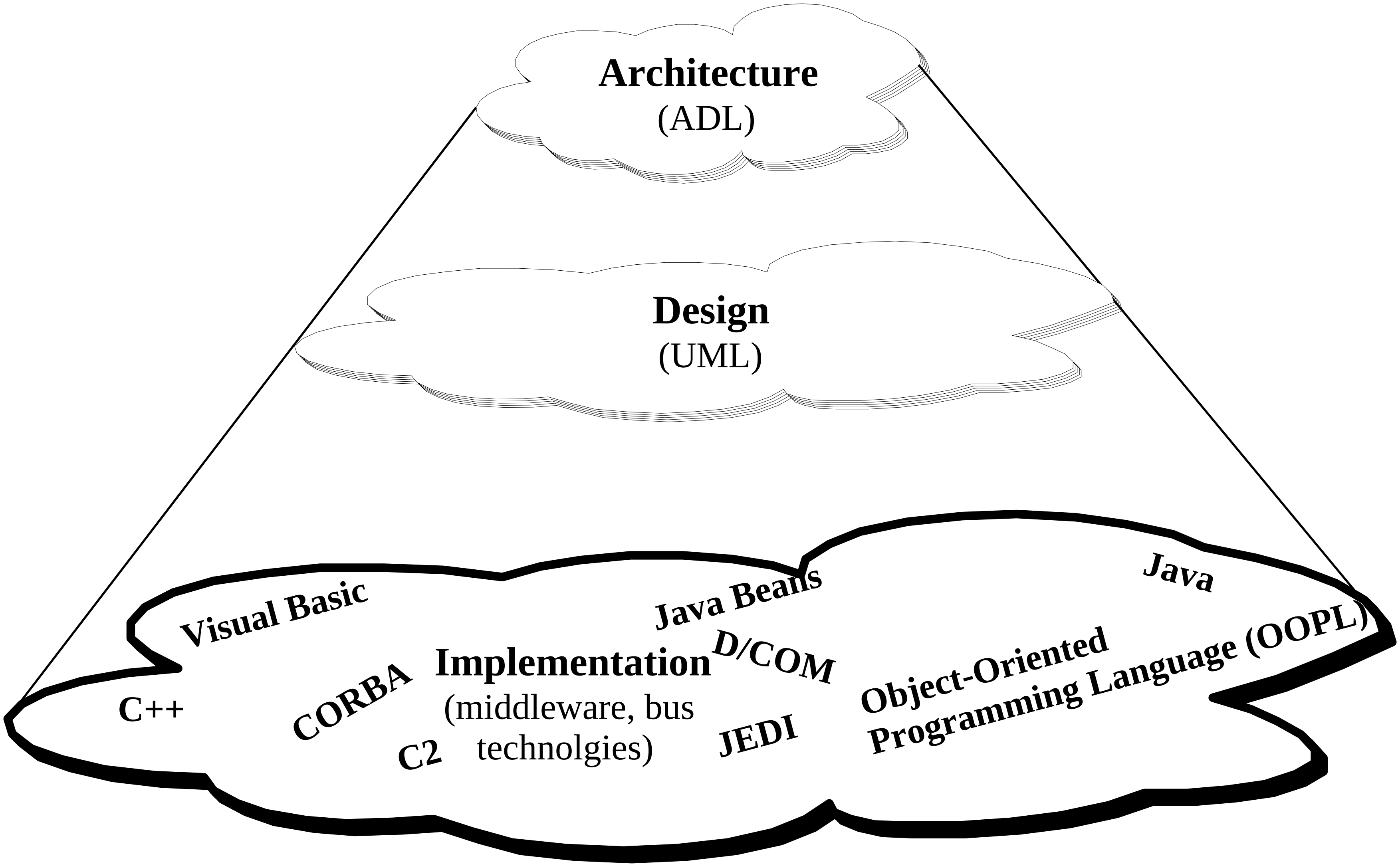
- Architectures provide *high-level* concepts
  - Components, connectors, ports, events, configurations
- Programming languages provide *low-level* constructs
  - Variables, arrays, pointers, procedures, objects
- Bridging the two often is an art-form
  - Middleware can help "split the difference"
- Existing middleware technologies
  - Support some architectural concepts (e.g., components, events)
  - but not others (e.g., connectors, configurations)
  - Impose particular architectural styles

What is needed is "architectural middleware"

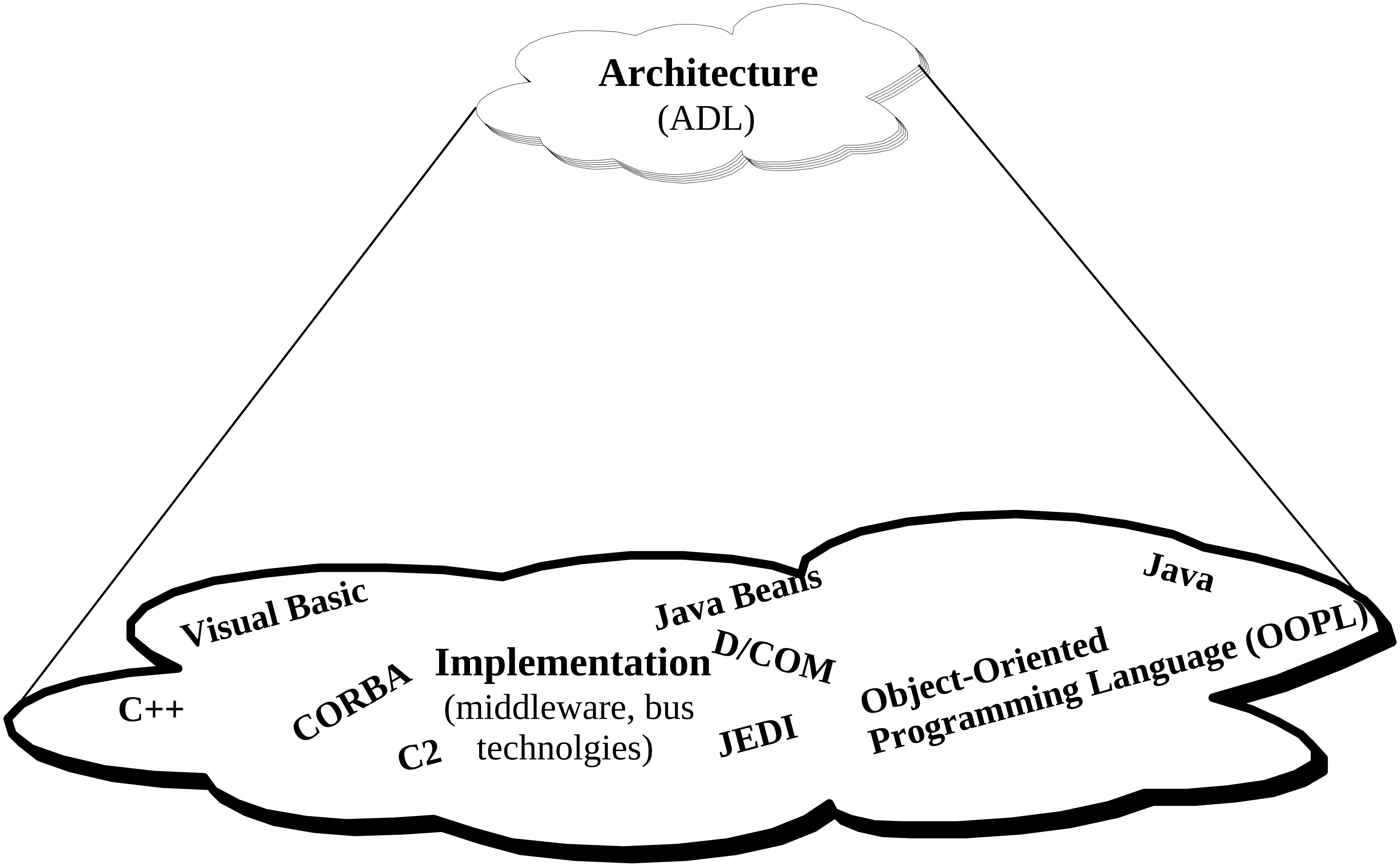
Architectural Middleware

- Natively support architectural concepts as middleware constructs
- Include system design support
  - Typically via an accompanying ADL and analysis tools
  - May support explicit architectural styles
- Support round-trip development
  - From architecture to implementation and back
- Support automated transformation of architectural models to implementations
  - i.e., dependable implementation
- Examples
  - ArchJava
  - Aura
  - c2.fw
  - Prism-MW

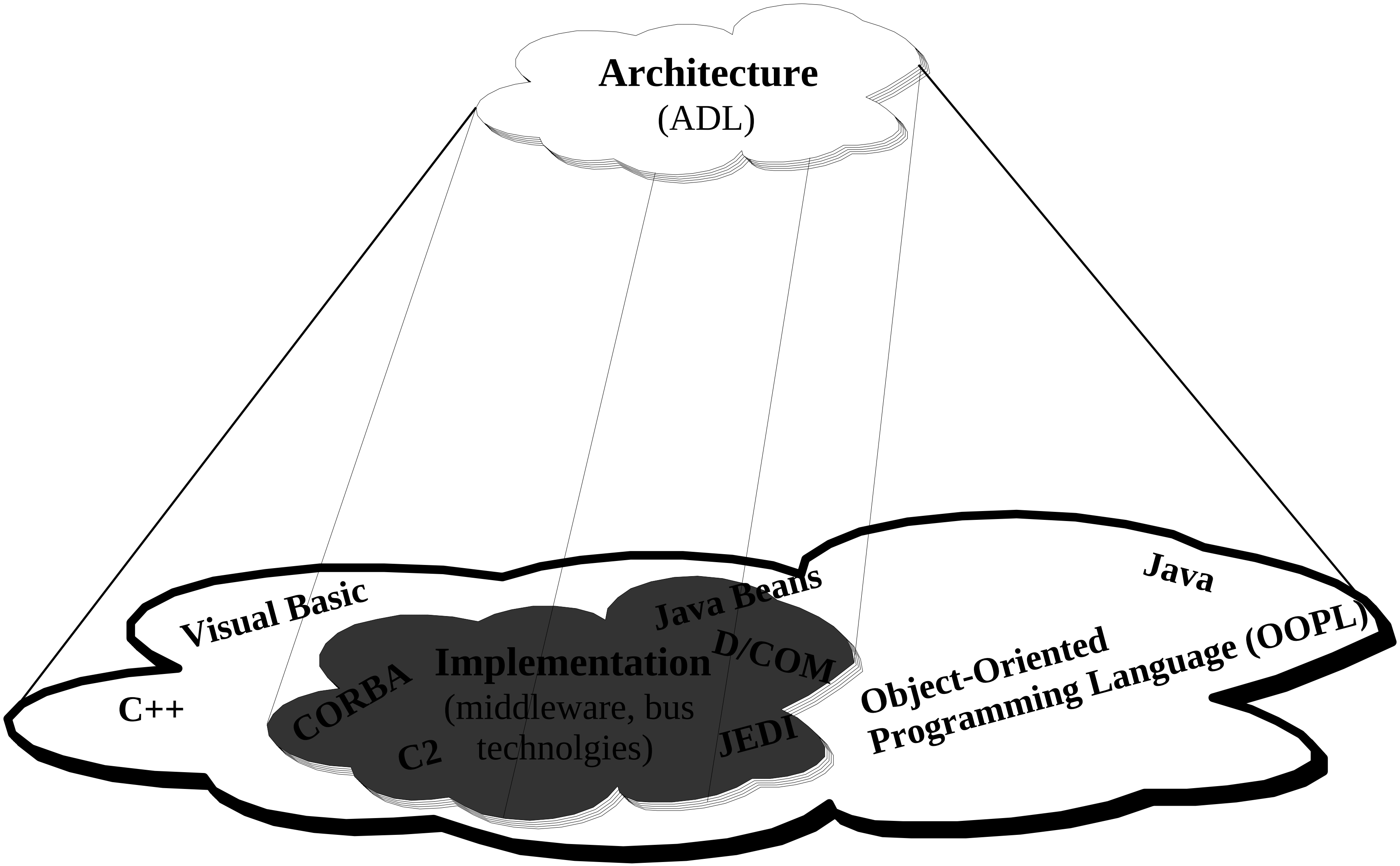
Dependable Implementation



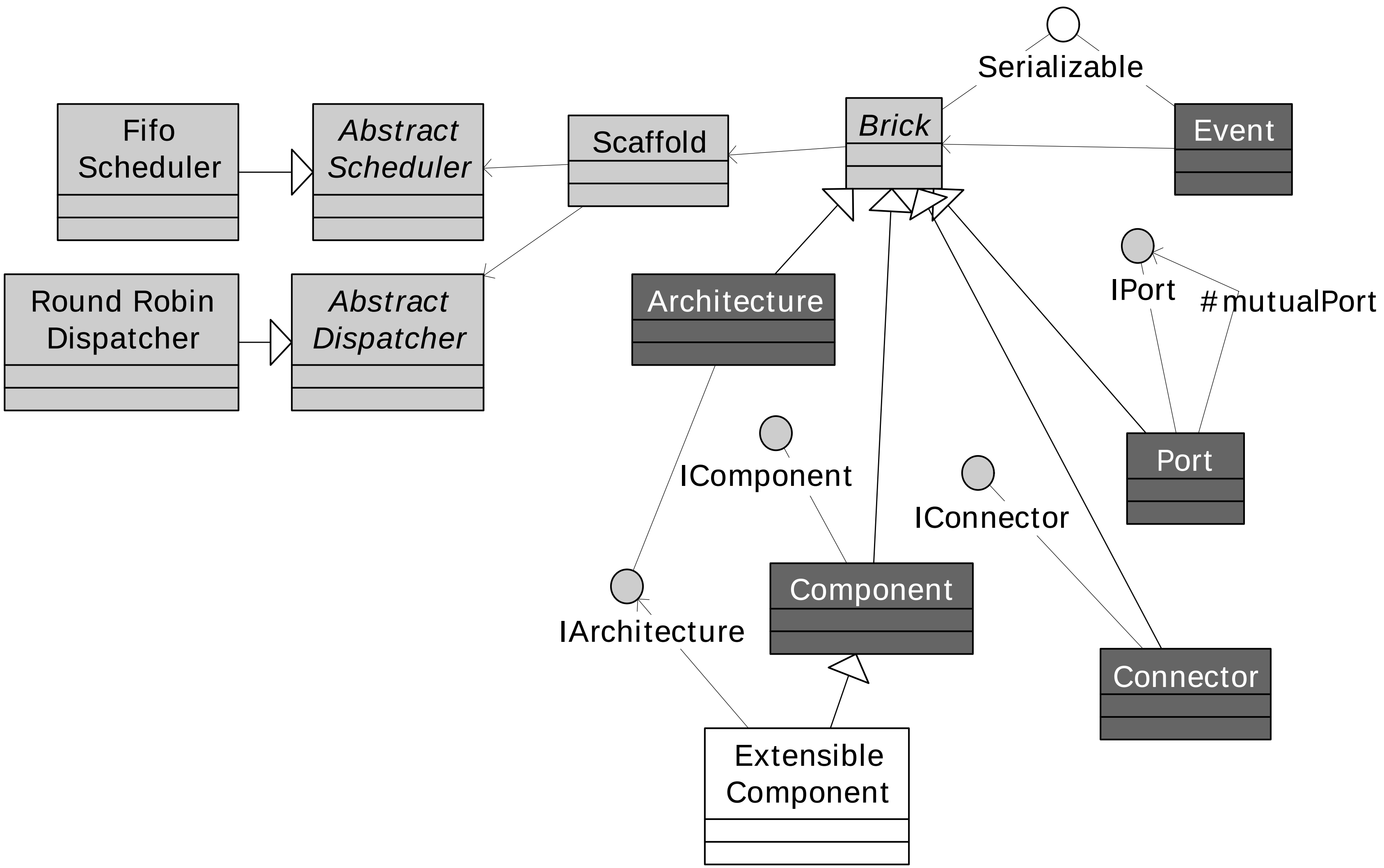
Dependable Implementation



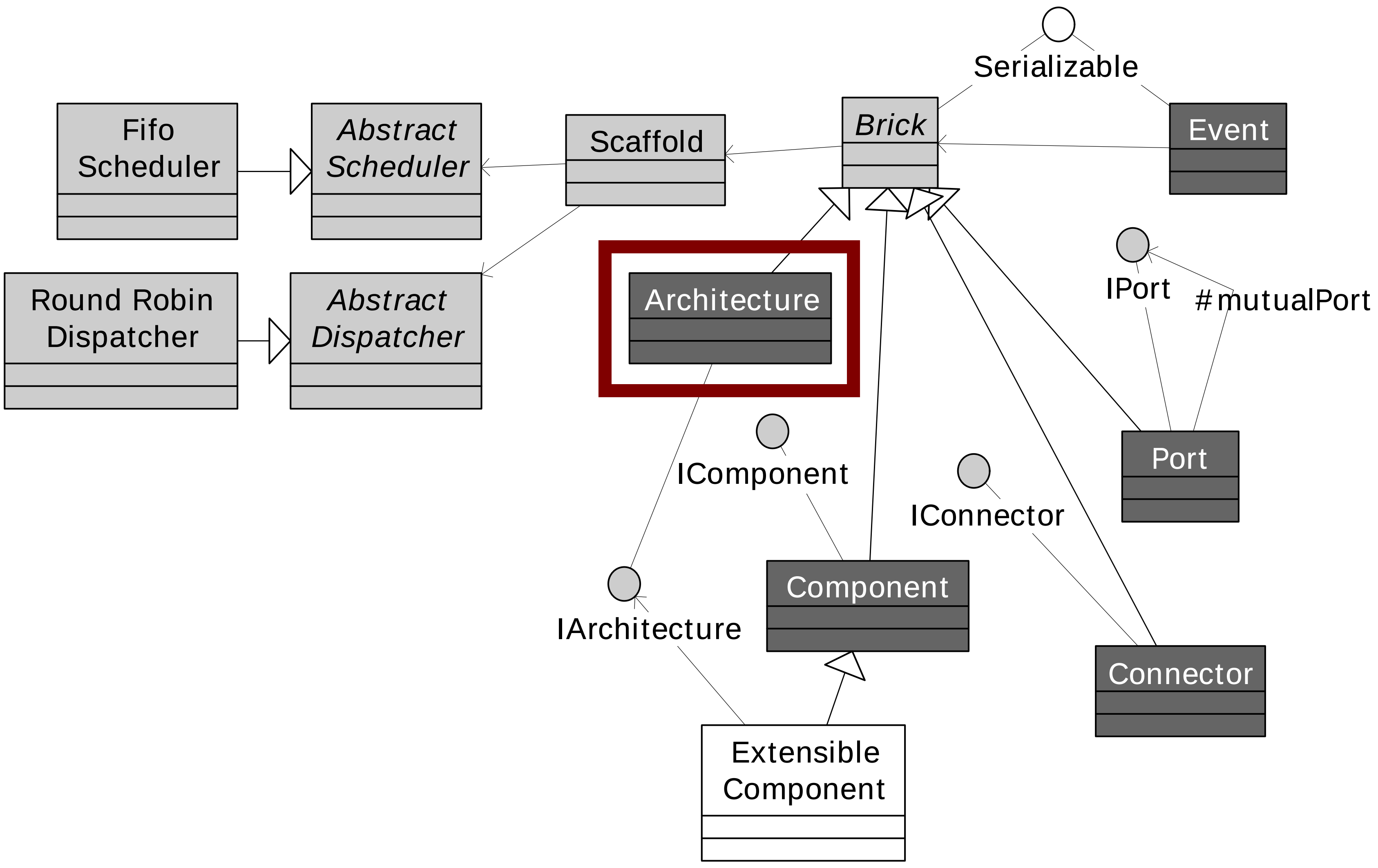
Dependable Implementation



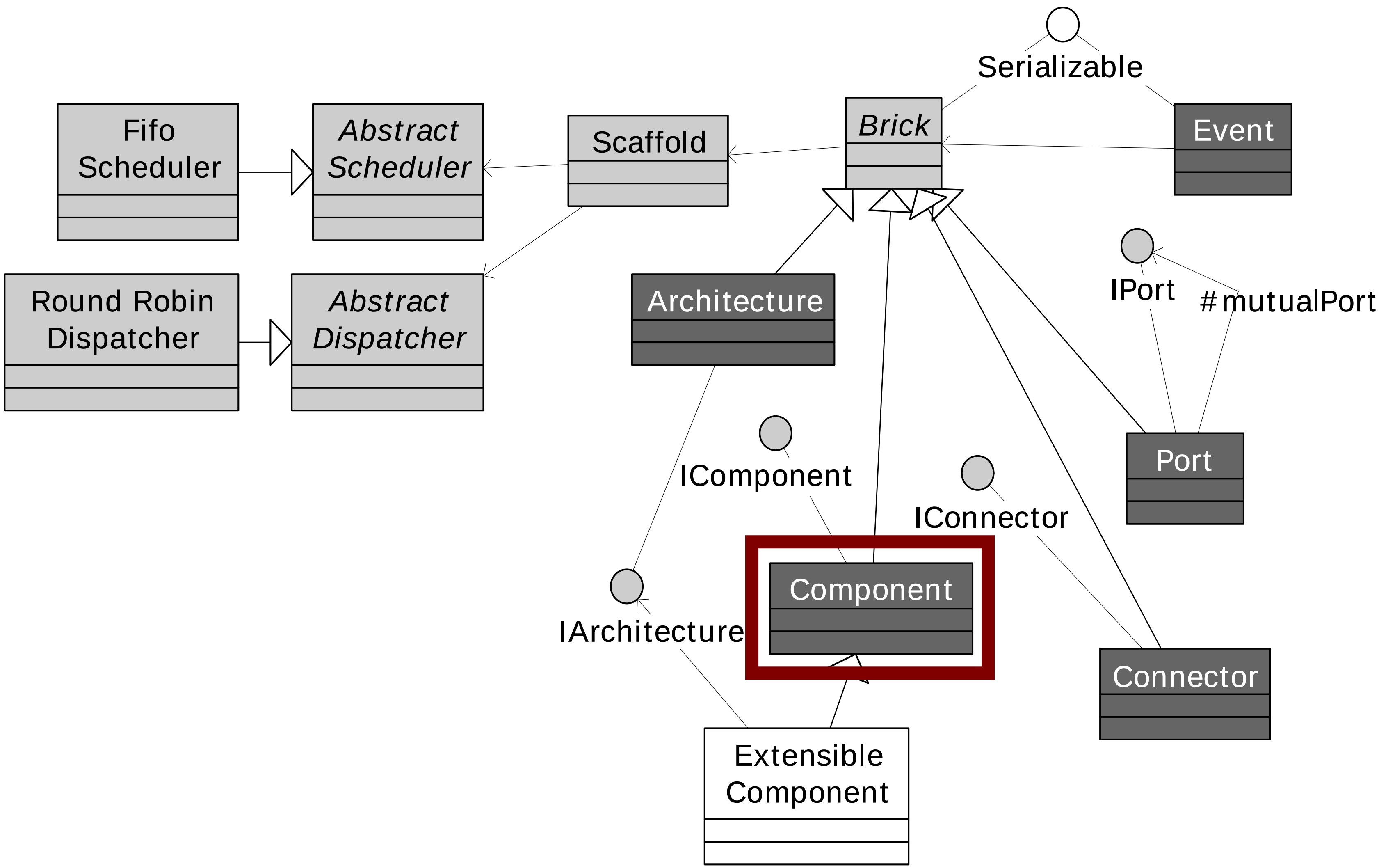
Example: Prism-MW



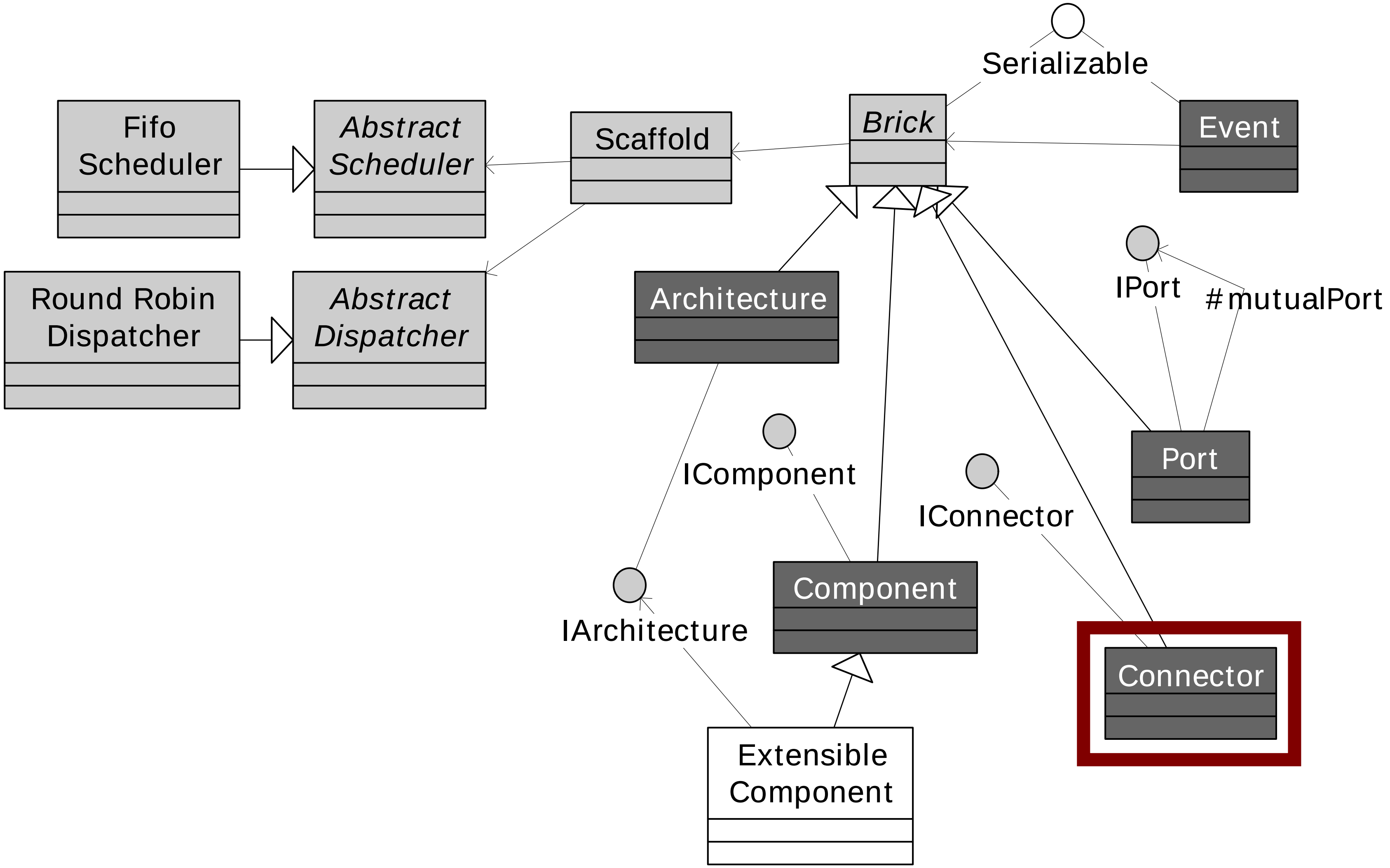
Example: Prism-MW



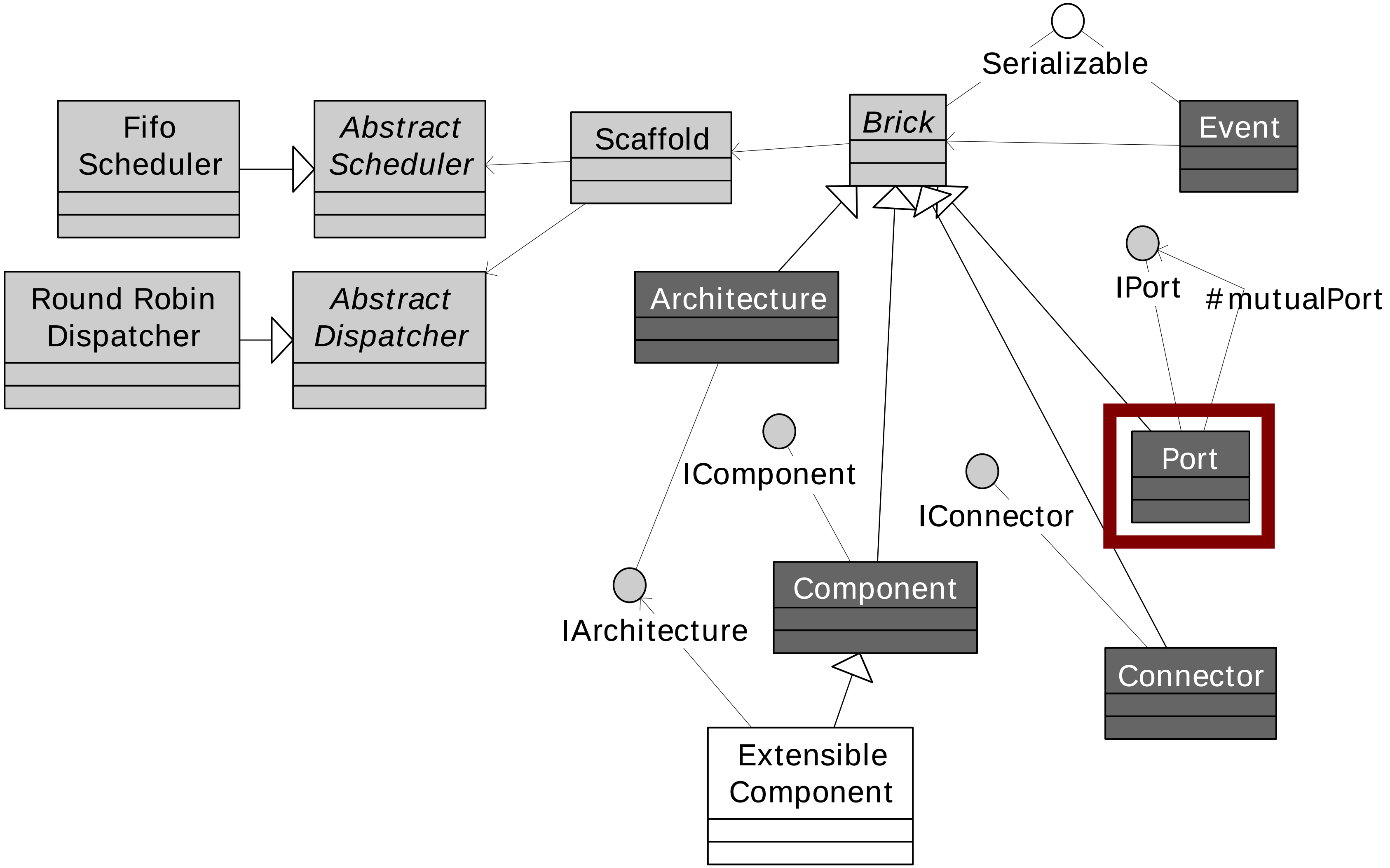
Example: Prism-MW



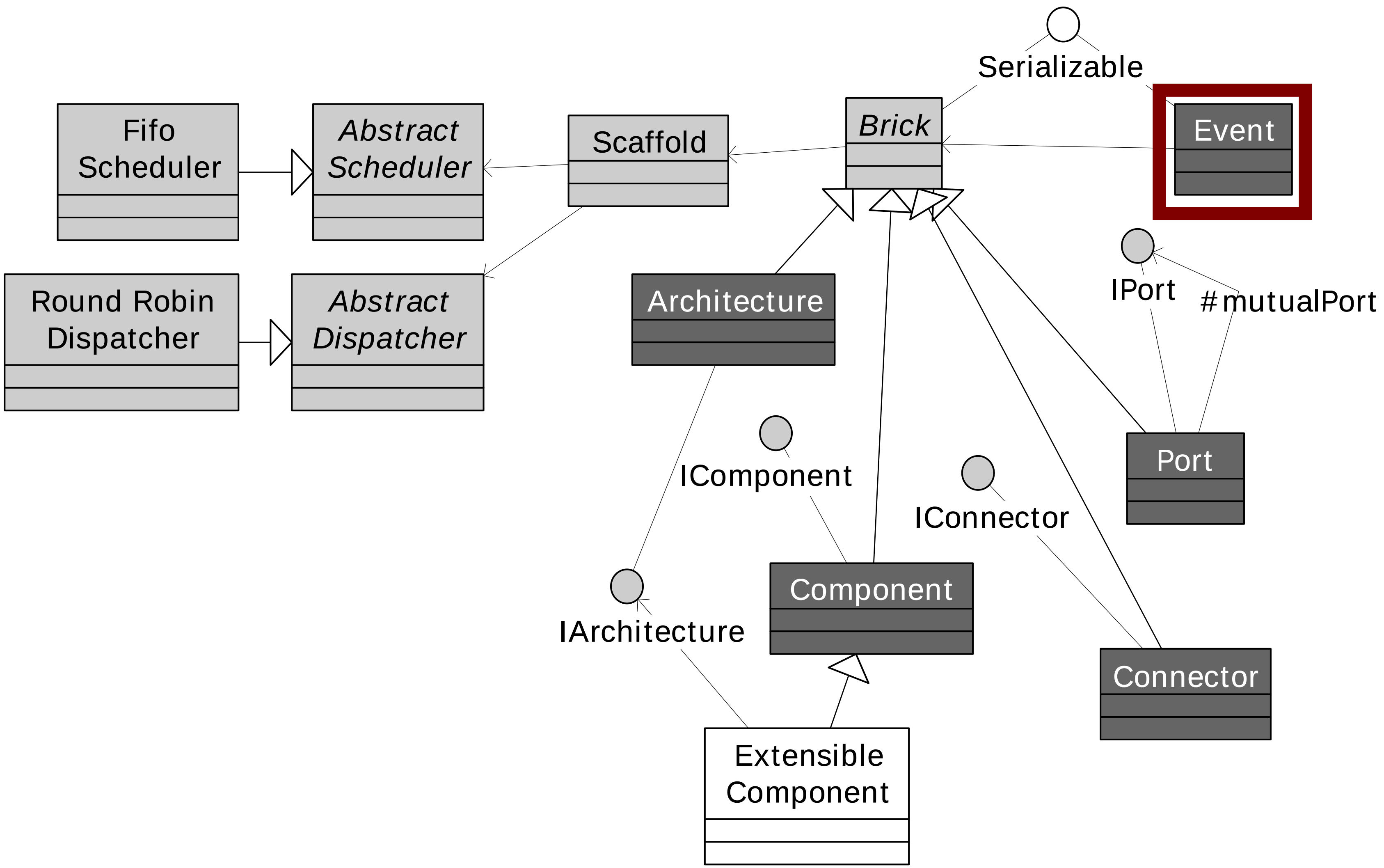
Example: Prism-MW



Example: Prism-MW



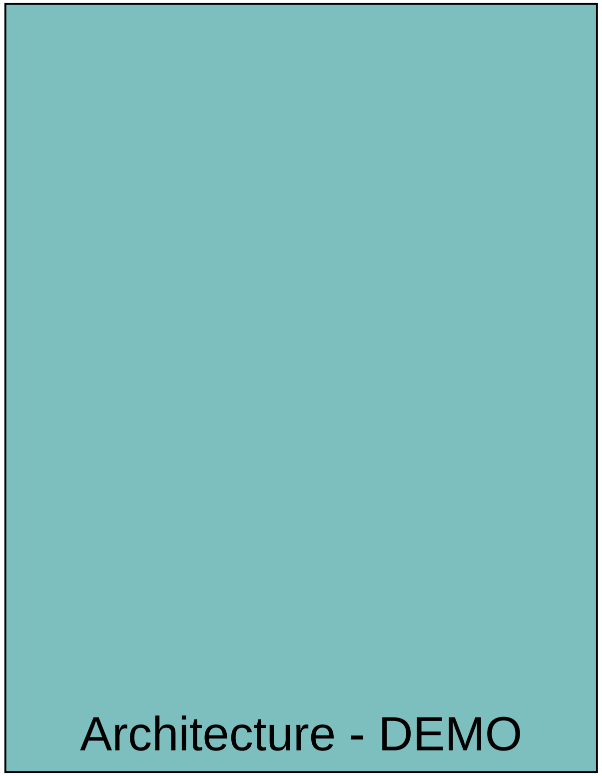
Example: Prism-MW



Using Prism-MW

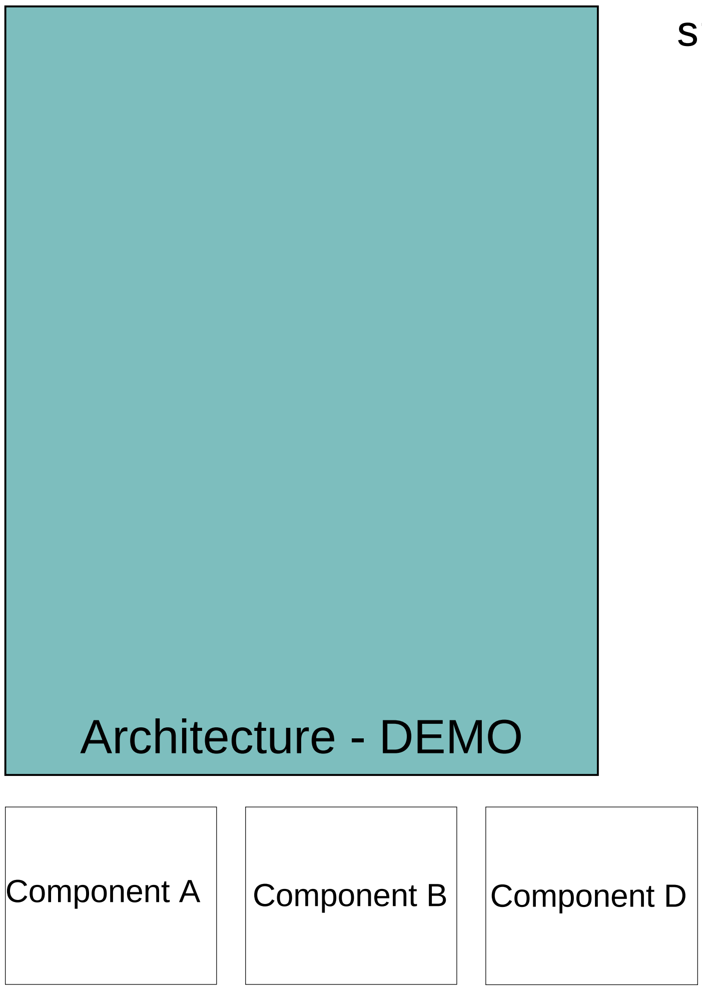
Using Prism-MW

```
class DemoArch {
    static public void main(String argv[]) {
        Architecture arch = new Architecture ("DEMO");
```



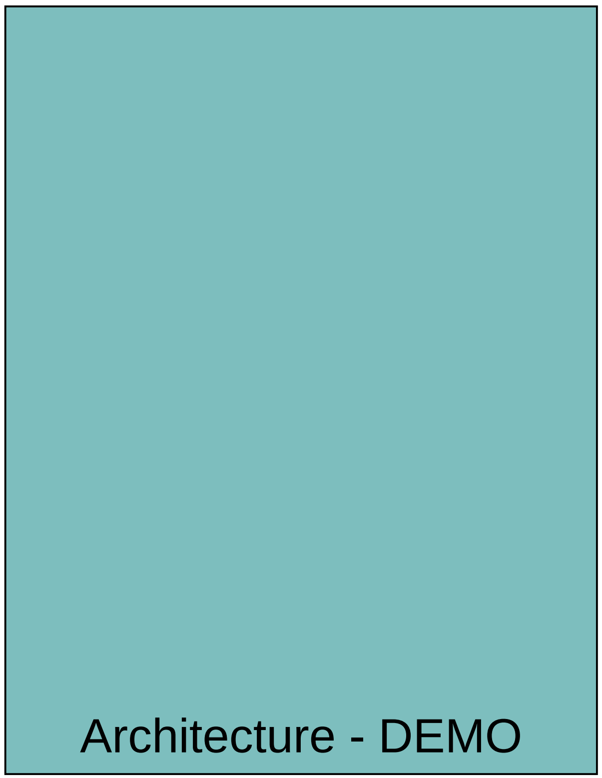
Using Prism-MW

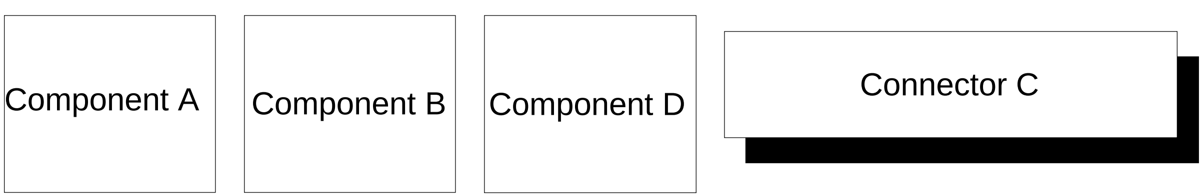
```
class DemoArch {
    static public void main(String argv[]) {
        Architecture arch = new Architecture ("DEMO");
        // create components
        ComponentA a = new ComponentA ("A");
        ComponentB b = new ComponentB ("B");
        ComponentD d = new ComponentD ("D");
```



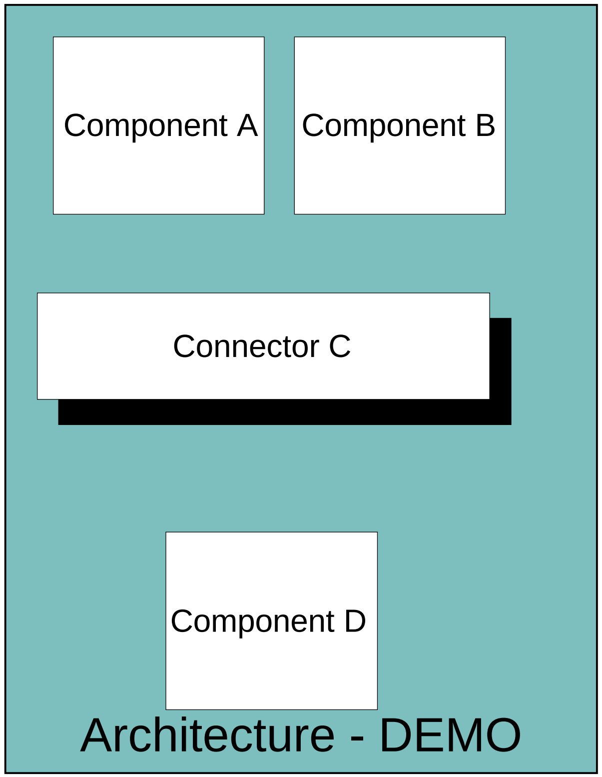
Using Prism-MW

```
class DemoArch {
    static public void main(String argv[]) {
        Architecture arch = new Architecture ("DEMO");
        // create components
        ComponentA a = new ComponentA ("A");
        ComponentB b = new ComponentB ("B");
        ComponentD d = new ComponentD ("D");
        // create connectors
        Connector conn = new Connector("C");
```



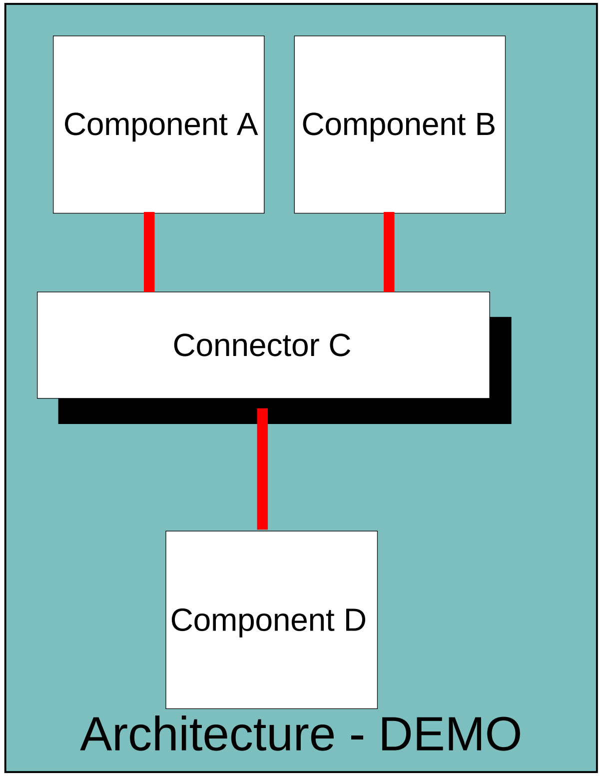


Using Prism-MW




```
class DemoArch {
    static public void main(String argv[]) {
        Architecture arch = new Architecture ("DEMO");
        // create components
        ComponentA a = new ComponentA ("A");
        ComponentB b = new ComponentB ("B");
        ComponentD d = new ComponentD ("D");
        // create connectors
        Connector conn = new Connector("C");
        // add components and connectors
        arch.addComponent(a);
        arch.addComponent(b);
        arch.addComponent(d);
        arch.addConnector(conn);
```

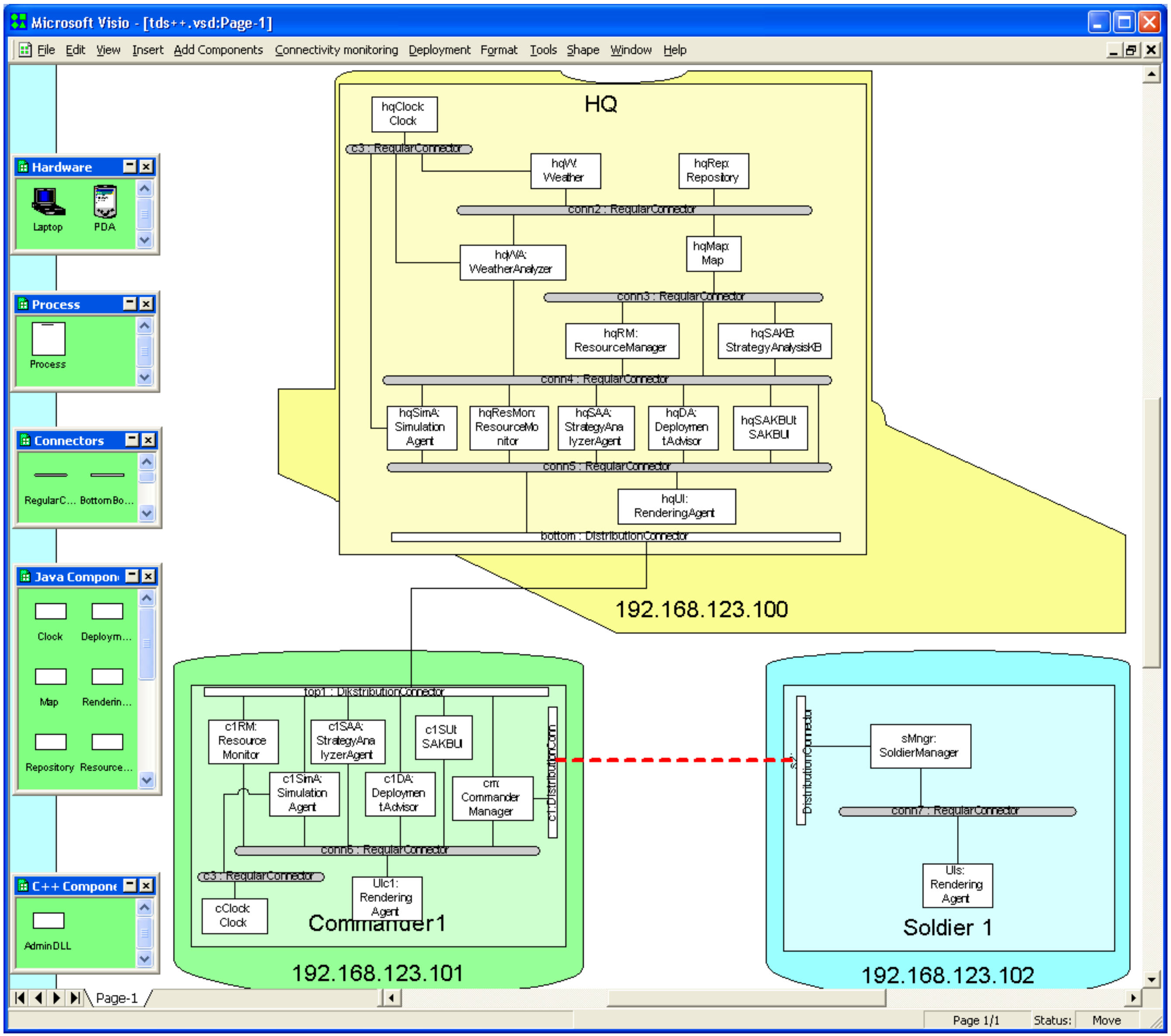
Using Prism-MW



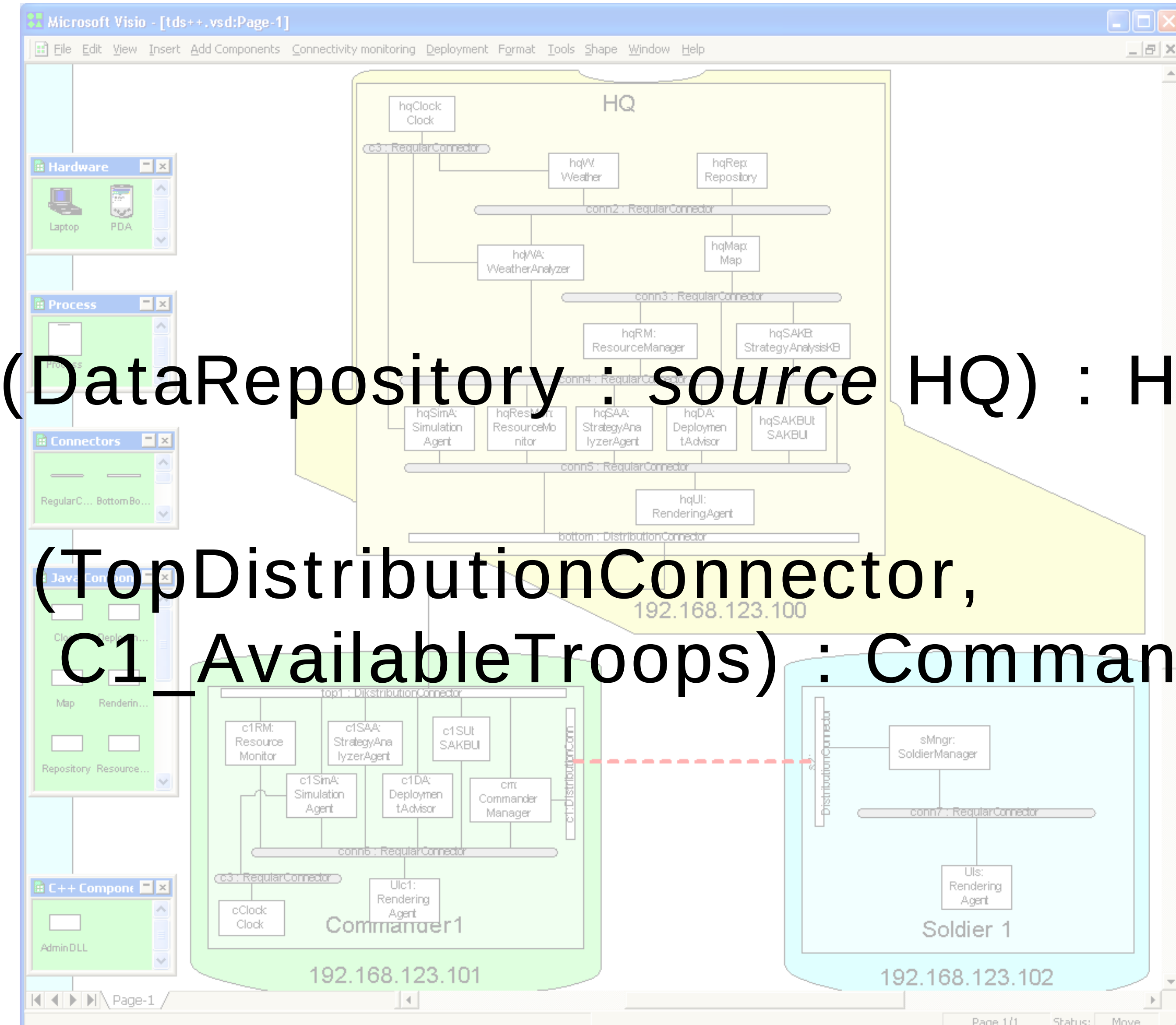

```
class DemoArch {
    static public void main(String argv[]) {
        Architecture arch = new Architecture ("DEMO");
        // create components
        ComponentA a = new ComponentA ("A");
        ComponentB b = new ComponentB ("B");
        ComponentD d = new ComponentD ("D");
        // create connectors
        Connector conn = new Connector("C");
        // add components and connectors
        arch.addComponent(a);
        arch.addComponent(b);
        arch.addComponent(d);
        arch.addConnector(conn);

        // establish the interconnections
        arch.weld(a, conn);
        arch.weld(b, conn);
        arch.weld(conn, d)
    }
}
```

Deploying a Prism-MW Architecture



Deploying a Prism-MW Architecture



add (DataRepository : *source* HQ) : HQ;

weld (TopDistributionConnector,
C1_AvailableTroops) : Commander1;

Outline

- From architectures to systems
- Ensuring dependability
  - Problem definition
  - Proposed solution
- Concluding remarks

Outline

- From architectures to systems
- Ensuring dependability
  - Problem definition
  - Proposed solution
- Concluding remarks

Availability

- The degree to which a system is operational and accessible when required for use [IEEE]
- Deployment architecture influences availability
  - Components on the same host can communicate regardless of the network's status
  - Components on different hosts are insulated from each other's failures
- Quantifying availability
  - **Ratio of the**
    number of successfully completed interactions
    in the system to the
    total number of attempted interactions

Maximizing Availability *a priori*

- We may not know many relevant system parameters
  - Dependability of each component
  - Frequency of component interactions
  - Volume of component interactions
  - Dependability of component interactions
  - CPU usage on each host
  - Dependability of each host
  - Effective bandwidth of each network connection
  - Dependability of each network connection
  - . . .

Maximizing Availability *a priori*

- We may not know many relevant system parameters
  - Dependability of each component
  - Frequency of component interactions
  - Volume of component interactions
  - Dependability of component interactions
  - CPU usage on each host
  - Dependability of each host
  - Effective bandwidth of each network connection
  - Dependability of each network connection
  - . . .

The current deployment architecture may not work well

Simplified Problem Definition

Given:

(1) a set C of n components ($n = |C|$), a relation $freq : C \times C \to \Re$, and a function $mem_{comp} : C \to \Re$

Simplified Problem Definition

Given:

(1)a set C of n components ($n = |C|$), a relation $freq : C \times C \rightarrow \Re$, and a function $mem_{comp} : C \rightarrow \Re$

$$freq(c_i, c_j) = \begin{cases} 0 \quad if \;\; c_i = c_j \\ frequency\ of\ \ comm\ between\ c_i\ \ and\ \ c_j \quad\quad if \quad\quad c_i \neq c_j \end{cases}$$

Simplified Problem Definition

Given:

(1) a set C of n components ($n = |C|$), a relation $freq : C \times C \rightarrow \Re$, and a function $mem_{comp} : C \rightarrow \Re$

$$freq(c_i, c_j) = \left\{ \begin{array}{ll} 0 \quad if \ \ c_i = c_j & \\ frequency \ of \ \ comm \ between \ c_i \ \ and \ \ c_j & \quad if \quad c_i \neq c_j \end{array} \right\}$$

$$mem_{comp}(c) = required \ memory \ \ for \ c$$

Simplified Problem Definition

Given:

(2) a set H of *k* hardware nodes ($k = |H|$), a relation

$rel : H \times H \rightarrow \Re$, and a function $mem_{host} : H \rightarrow \Re$

Simplified Problem Definition

Given:

(2) a set H of *k* hardware nodes ($k = |H|$), a relation

$rel : H \times H \rightarrow \Re$, and a function $mem_{host} : H \rightarrow \Re$

$$rel(h_i, h_j) = \left\{ \begin{array}{l} 1 \quad if \quad h_i = h_j \\ 0 \quad if \quad h_i \ \ is\ not\ connected\ to\ h_j \\ reliability\ of\ the\ link\ between\ h_i\ and\ h_j \quad if \quad h_i \neq h_j \end{array} \right\}$$

Simplified Problem Definition

Given:

(2) a set H of *k* hardware nodes ($k = |H|$), a relation

$rel : H \times H \rightarrow \Re$, and a function $mem_{host} : H \rightarrow \Re$

$$rel(h_i, h_j) = \begin{cases} 1 \quad if \quad h_i = h_j \\ 0 \quad if \quad h_i \ \ is \ not \ connected \ to \ \ h_j \\ reliability \ of \ \ the \ link \ between \ h_i \ \ and \ \ h_j \quad if \quad h_i \neq h_j \end{cases}$$

$$mem_{host}(h) = available \ \ memory \ \ on \ \ host \ \ h$$

Simplified Problem Definition

Given:

(3) Two relations that restrict locations of software components

$loc : C \times H \rightarrow \{0,1\}$ $\qquad$ $colloc : C \times C \rightarrow \{-1,0,1\}$

Simplified Problem Definition

Given:

(3) Two relations that restrict locations of software components

$$loc: C \times H \rightarrow \{0,1\} \qquad colloc: C \times C \rightarrow \{-1,0,1\}$$

$$loc(c_i, h_j) = \begin{cases} 1 & if \quad c_i \; can \; be \; deployed \; onto \; h_j \\ 0 & if \quad c_i \; cannot \; be \; deployed \; onto \; h_j \end{cases}$$

Simplified Problem Definition

Given:

(3) Two relations that restrict locations of software components

$$loc : C \times H \to \{0,1\} \qquad colloc : C \times C \to \{-1,0,1\}$$

$$loc(c_i, h_j) = \begin{cases} 1 & if \quad c_i \ can\ be\ deployed\ onto\ h_j \\ 0 & if \quad c_i \ cannot\ be\ deployed\ onto\ h_j \end{cases}$$

$$colloc(c_i, c_j) = \begin{cases} -1 & if \quad c_i \ cannot\ be\ on\ the\ same\ host\ as\ c_j \\ 1 & if \quad c_i \ has\ to\ be\ on\ the\ same\ host\ as\ c_j \\ 0 & if \quad there\ are\ no\ restrictions\ on\ collocation\ of\ c_i\ and\ c_j \end{cases}$$

Simplified Problem Definition

Find a function $f : C \rightarrow H$ such that the system's overall availability

$$A = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}\left(freq(c_i, c_j) * rel(f(c_i), f(c_j)) \right)}{\sum_{i=1}^{n}\sum_{j=1}^{n} freq(c_i, c_j)}$$

is maximized

Simplified Problem Definition

Find a function $f : C \rightarrow H$ such that the system's overall availability

$$A = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}\left(freq(c_i, c_j) * rel(f(c_i), f(c_j))\right)}{\sum_{i=1}^{n}\sum_{j=1}^{n} freq(c_i, c_j)}$$

is maximized, and the following three conditions are satisfied:

Simplified Problem Definition

Find a function $f : C \rightarrow H$ such that the system's overall availability

$$A = \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} \left(freq(c_i, c_j) * rel(f(c_i), f(c_j)) \right)}{\sum_{i=1}^{n} \sum_{j=1}^{n} freq(c_i, c_j)}$$

is maximized, and the following three conditions are satisfied:

$$\forall i \in [1, k] \left\{ \forall j \in [1, n] \quad f(c_j) = h_i \;\middle|\; \sum_j mem_{comp}(c_j)) \leq mem_{host}(h_i) \right\}$$

Simplified Problem Definition

Find a function $f : C \rightarrow H$ such that the system's overall availability

$$A = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}\left(freq(c_i, c_j) * rel(f(c_i), f(c_j))\right)}{\sum_{i=1}^{n}\sum_{j=1}^{n} freq(c_i, c_j)}$$

is maximized, and the following three conditions are satisfied:

$$\forall j \in [1, n] \qquad loc(c_j, f(c_j)) = 1$$

Simplified Problem Definition

Find a function $f : C \rightarrow H$ such that the system's overall availability

$$A = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}\left(freq(c_i, c_j) * rel(f(c_i), f(c_j))\right)}{\sum_{i=1}^{n}\sum_{j=1}^{n} freq(c_i, c_j)}$$

is maximized, and the following three conditions are satisfied:

$$\forall k \in [1, n] \quad \forall l \in [1, n]$$

$$(colloc\ (c_k, c_l) = 1) \Rightarrow (f(c_k) = f(c_l))$$

$$(colloc\ (c_k, c_l) = -1) \Rightarrow (f(c_k) \neq f(c_l))$$

Simplified Problem Definition

Find a function $f : C \rightarrow H$ such that the system's overall availability

$$A = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}\left(freq(c_i, c_j) * rel(f(c_i), f(c_j))\right)}{\sum_{i=1}^{n}\sum_{j=1}^{n} freq(c_i, c_j)}$$

is maximized, and the following three conditions are satisfied:

Note that the possible number of different functions f is k^n

Outline

- From architectures to systems
- Ensuring dependability
  - Problem definition
  - Proposed solution
- Concluding remarks

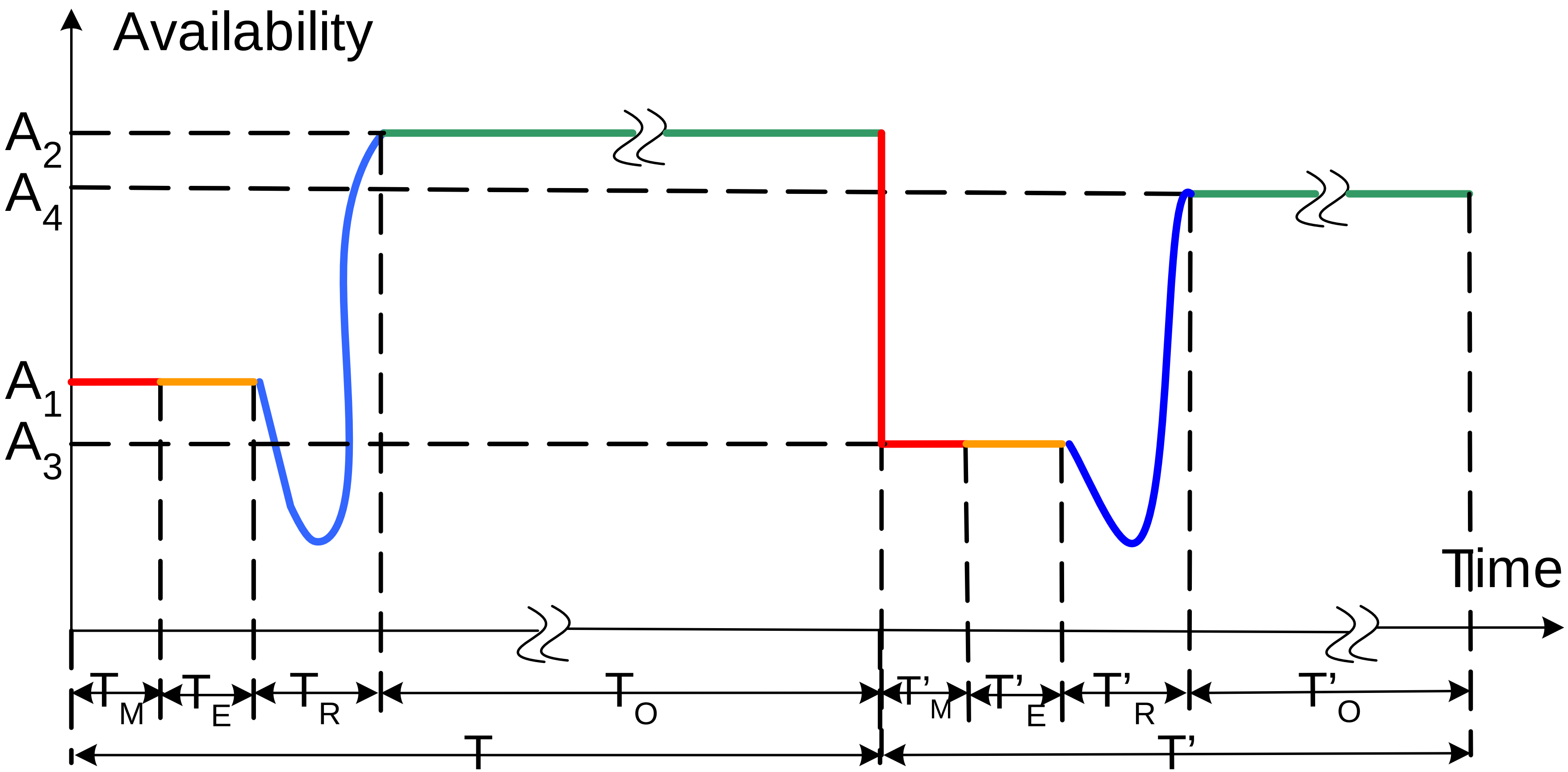
Overview of the Approach

- Objective
  - Identify the problem
    - Log and examine system events
    - Actively monitor the system during runtime
  - Develop a solution
    - Decide which data to cache
    - Decide which components to replicate
    - Introduce multiple execution modes
    - Calculate an improved system deployment
  - Apply the solution to eliminate the problem
    - Cache or hoard data
    - Replicate data or code
    - Redeploy (parts of) the system

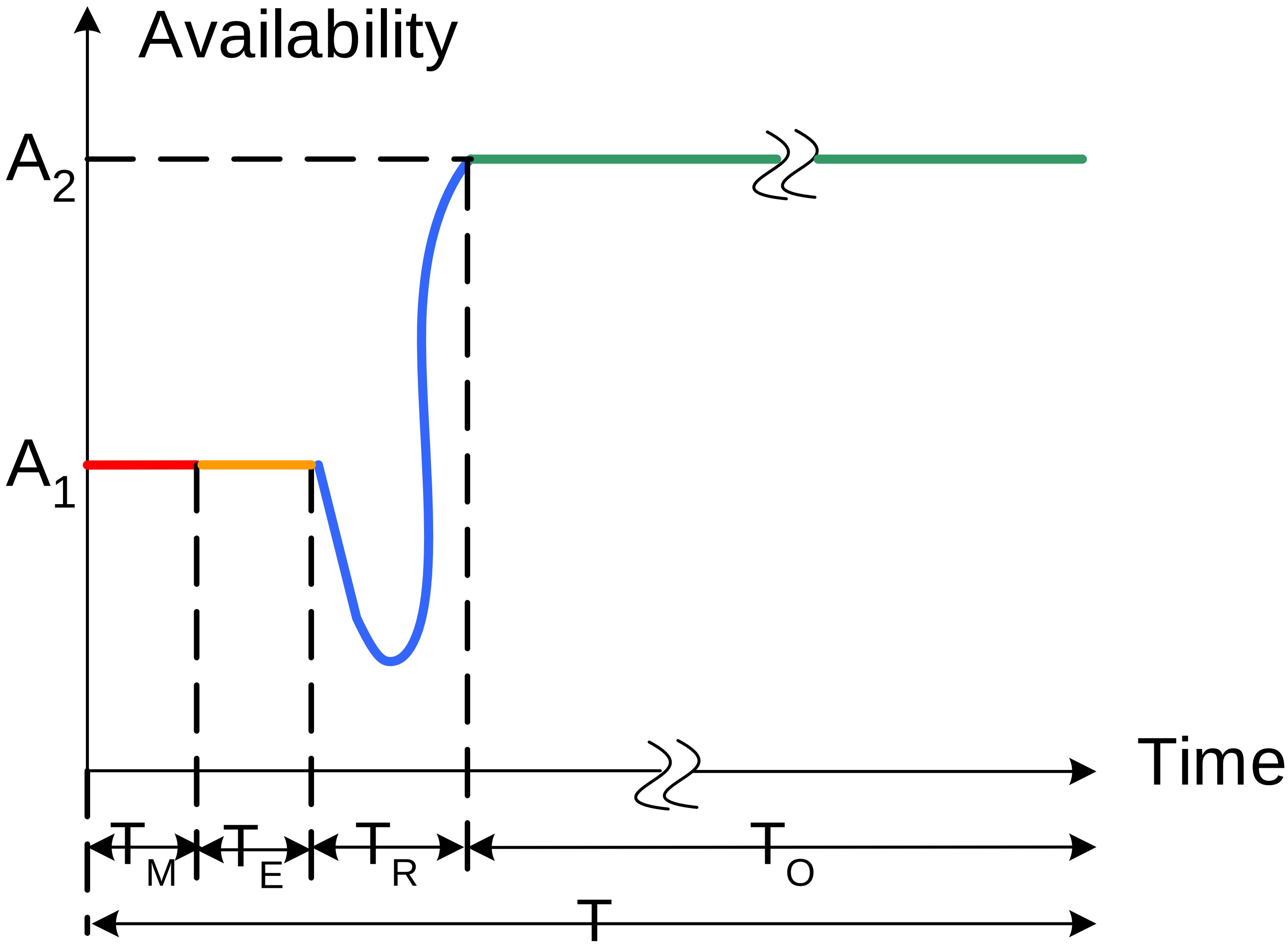
Overview of the Approach

- Objective
  - Identify the problem
    - Log and examine system events
    - Actively monitor the system during runtime
  - Develop a solution
    - Decide which data to cache
    - Decide which components to replicate
    - Introduce multiple execution modes
    - Calculate an improved system deployment
  - Apply the solution to eliminate the problem
    - Cache or hoard data
    - Replicate data or code
    - Redeploy (parts of) the system

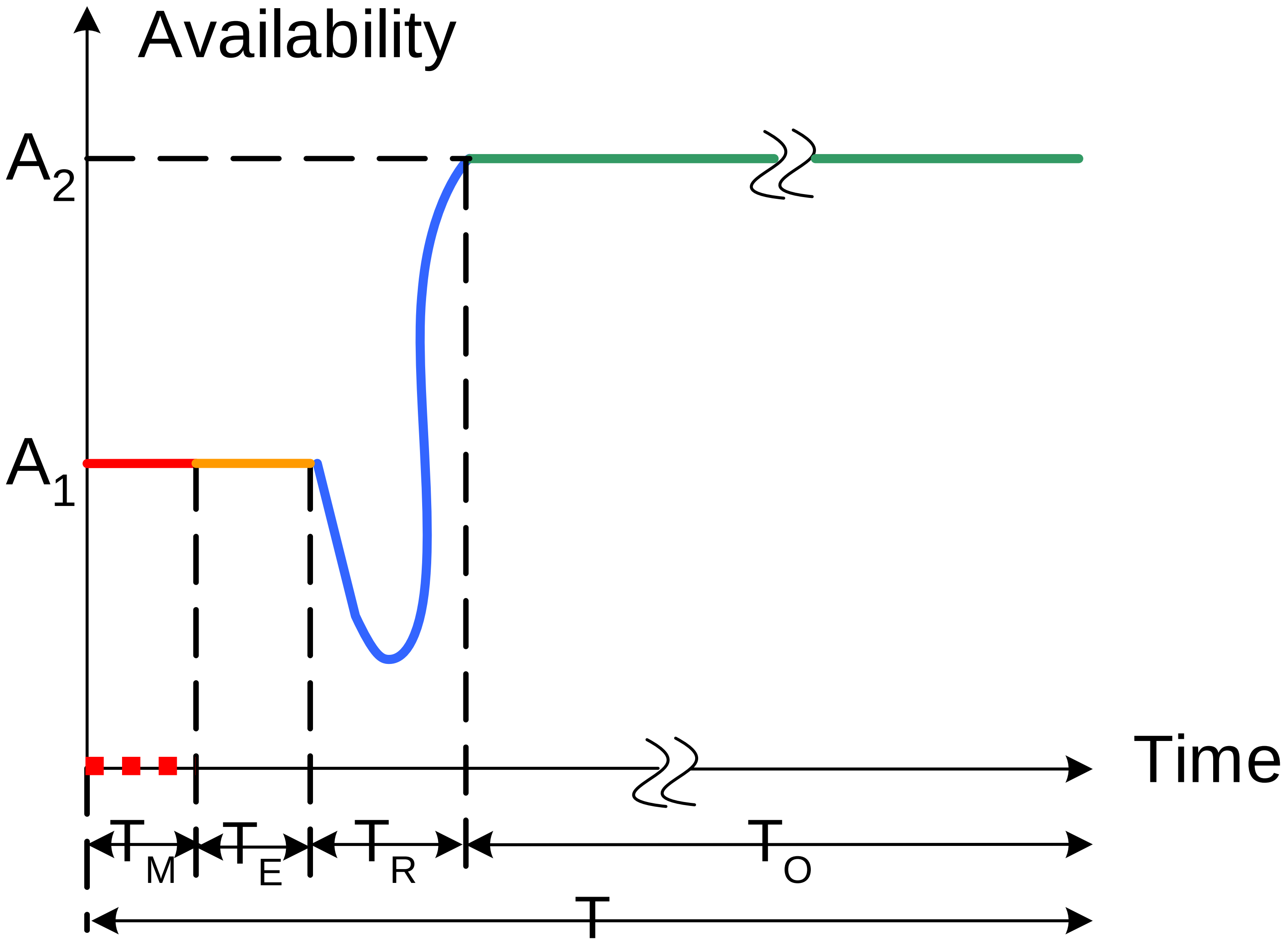
Improving Availability via Redeployment



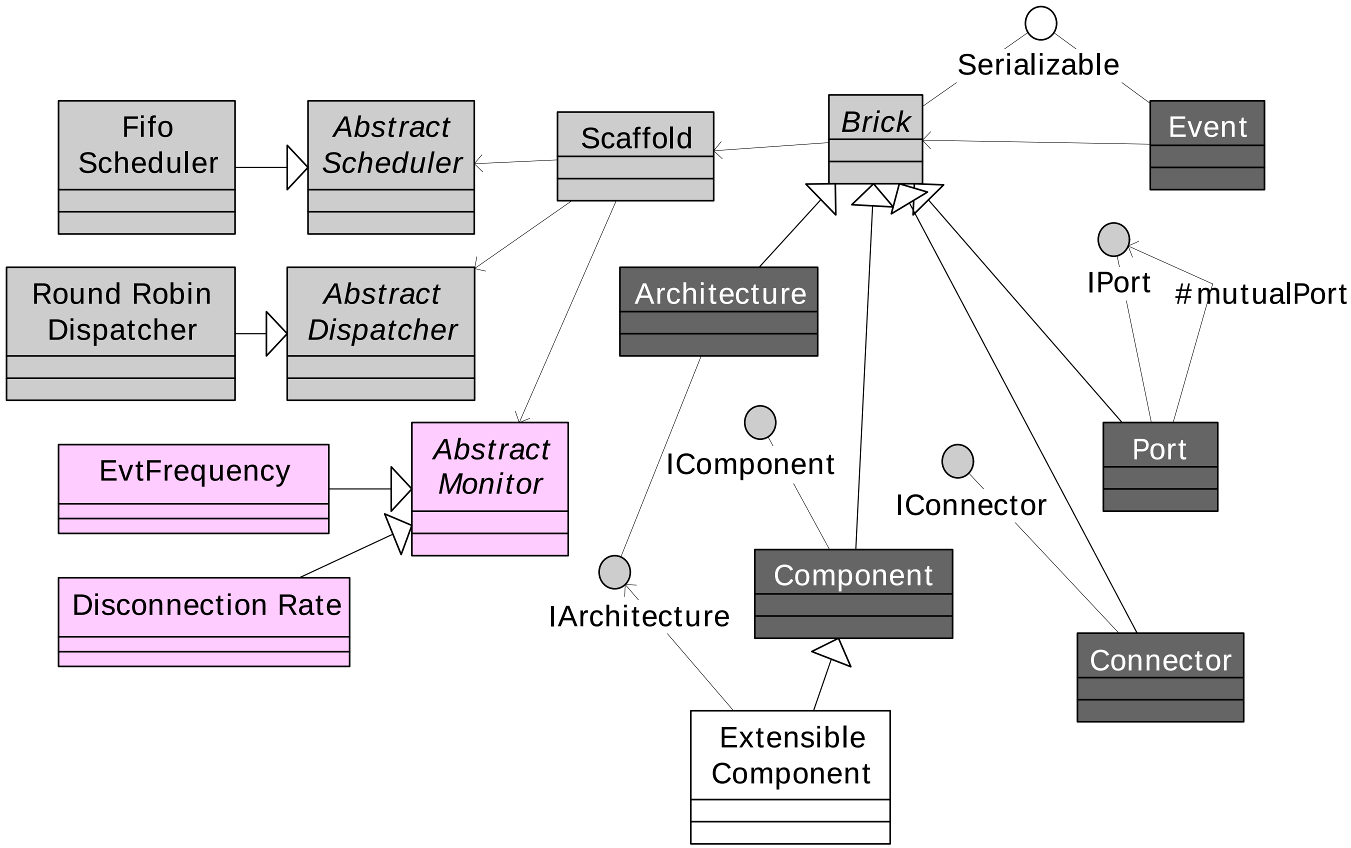
First Identify the Problem



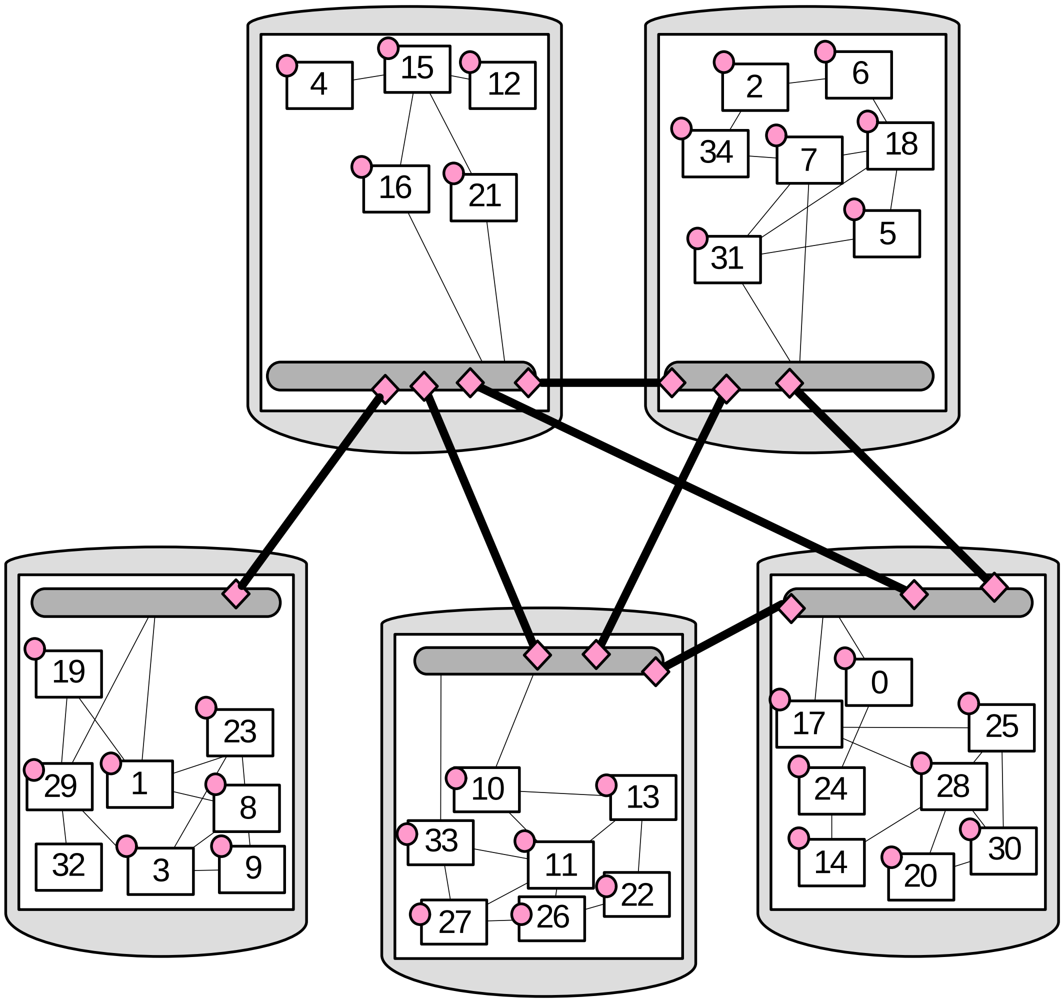
First Identify the Problem



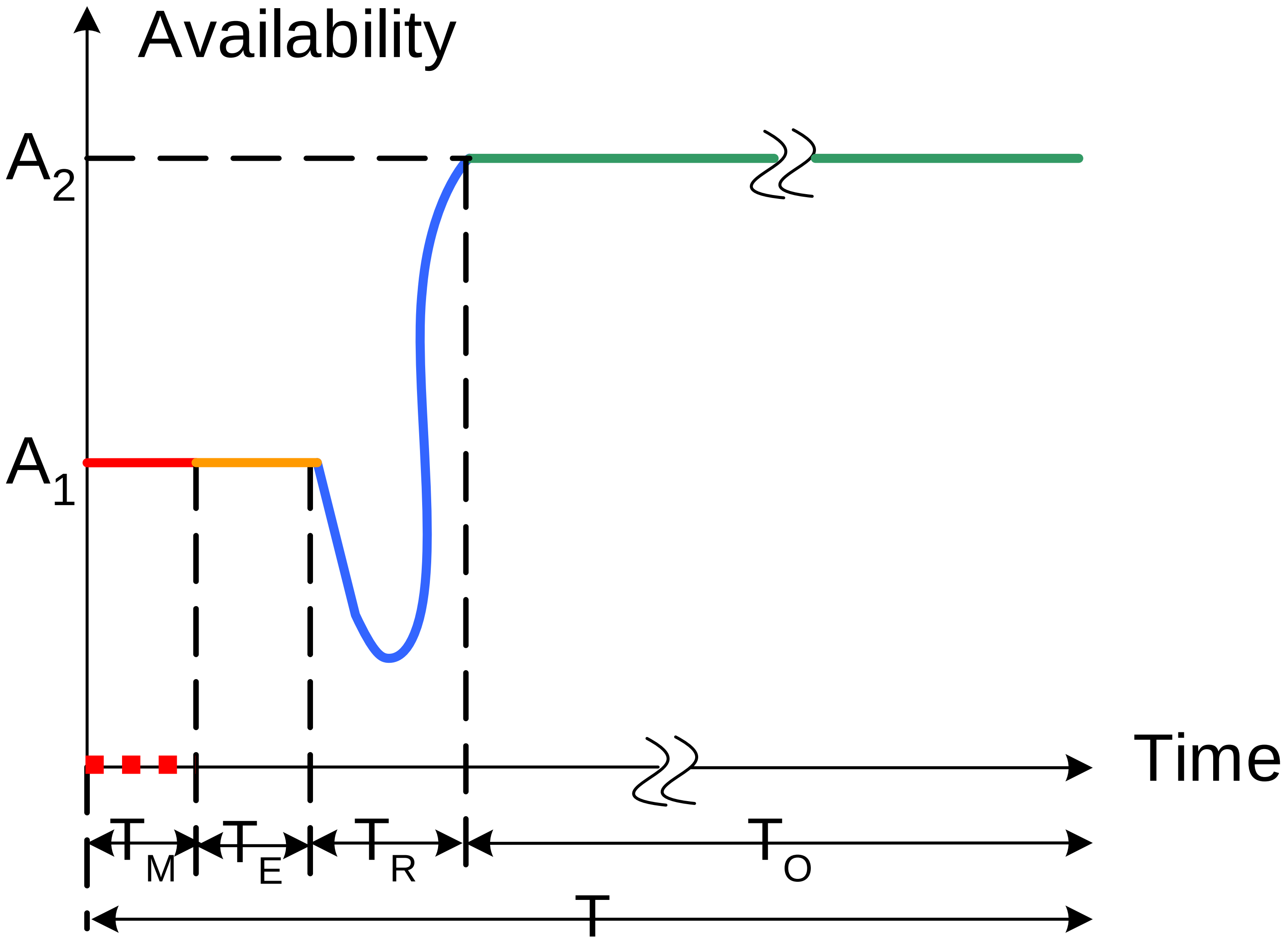
Monitoring in Prism-MW



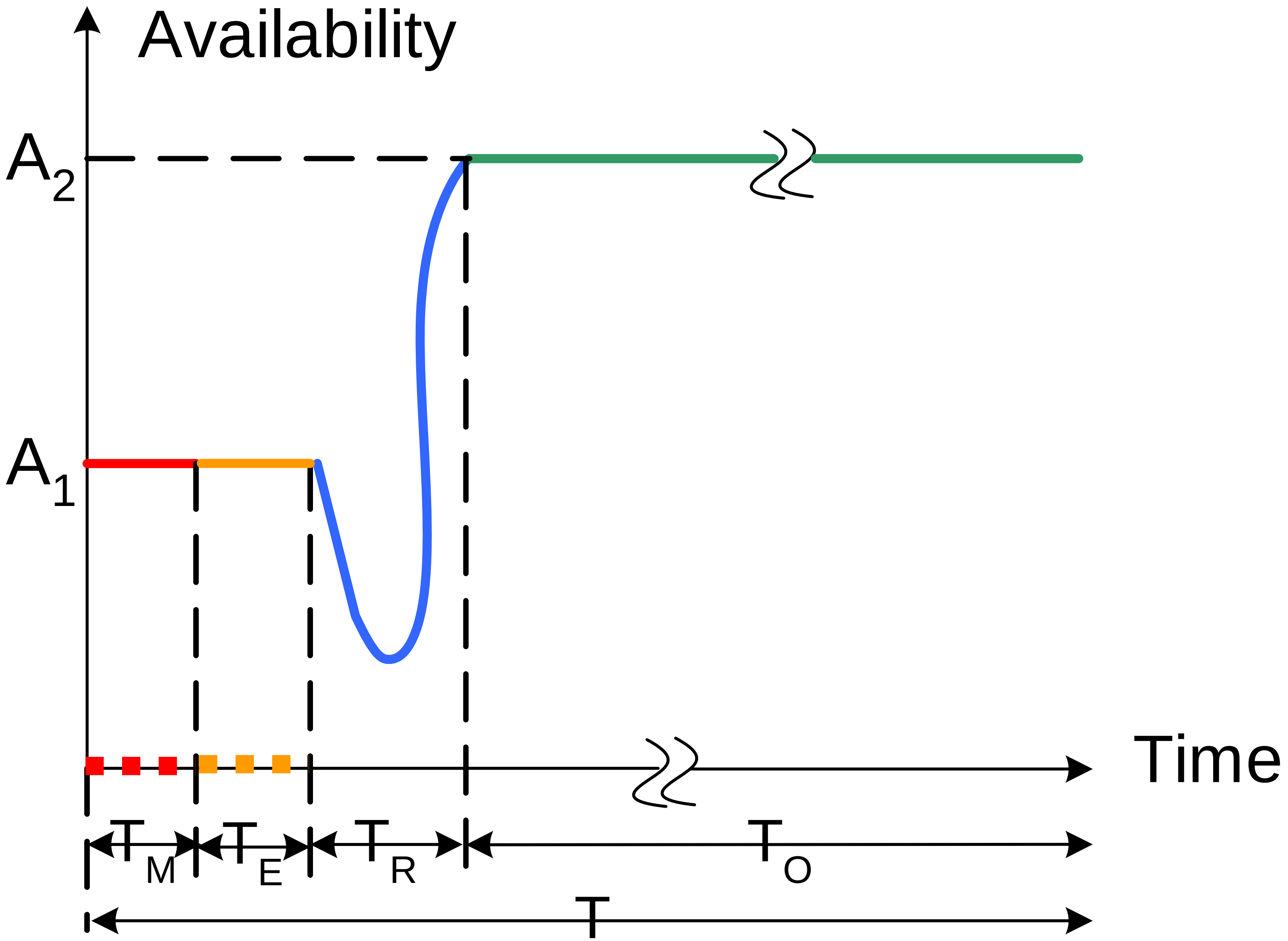
Monitoring in Action



Then Develop a Solution



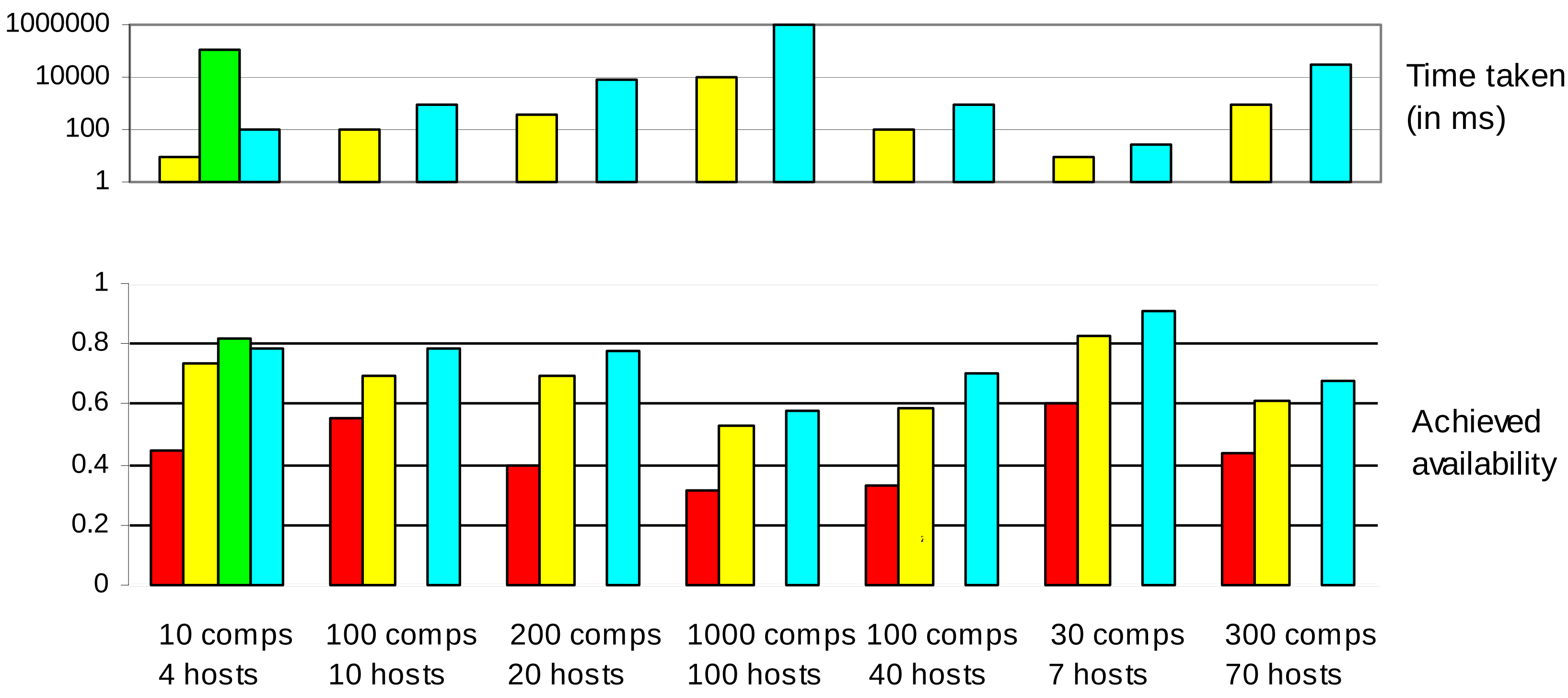
Then Develop a Solution



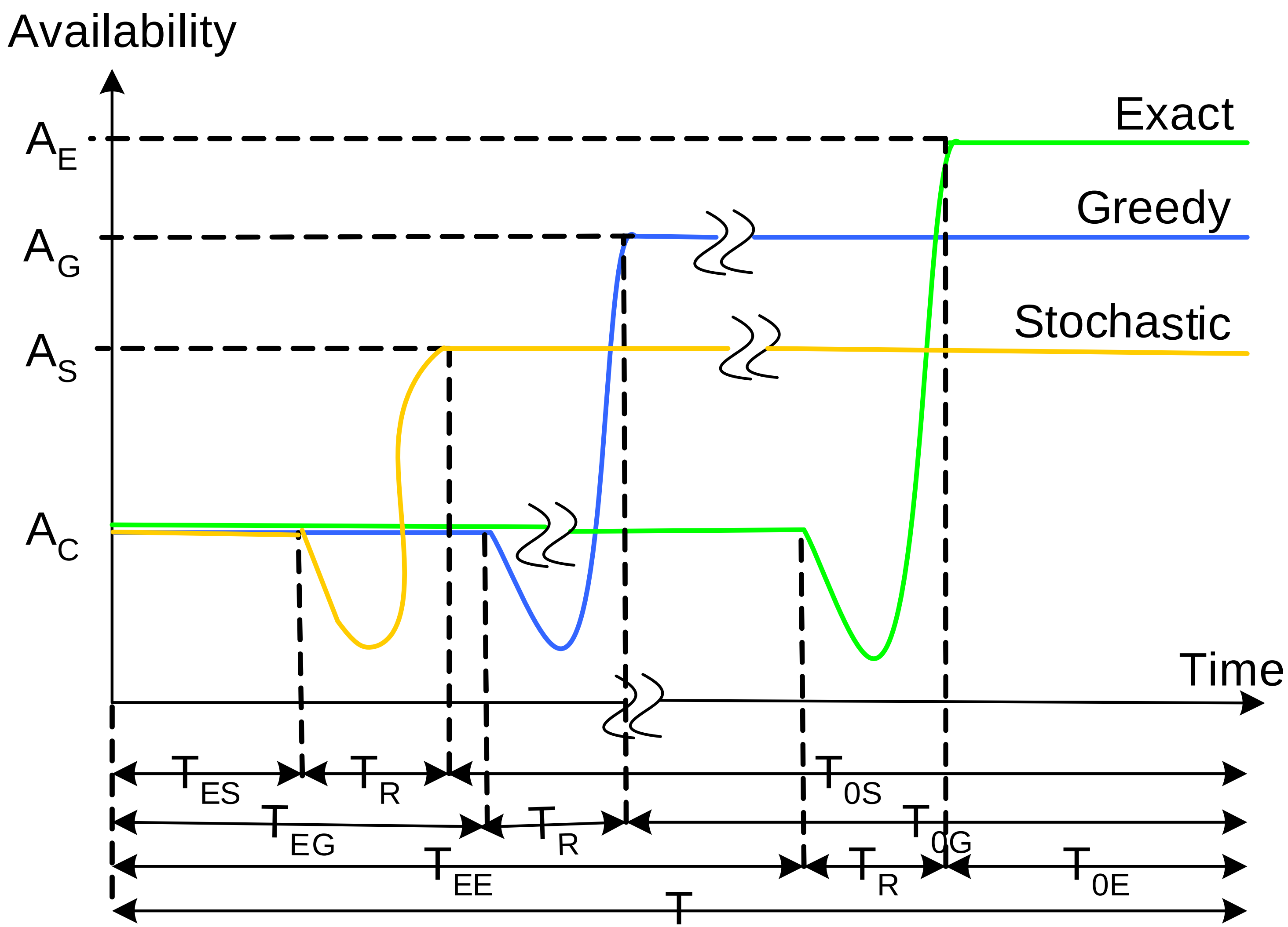
Estimating in Action

Suite of algorithms:

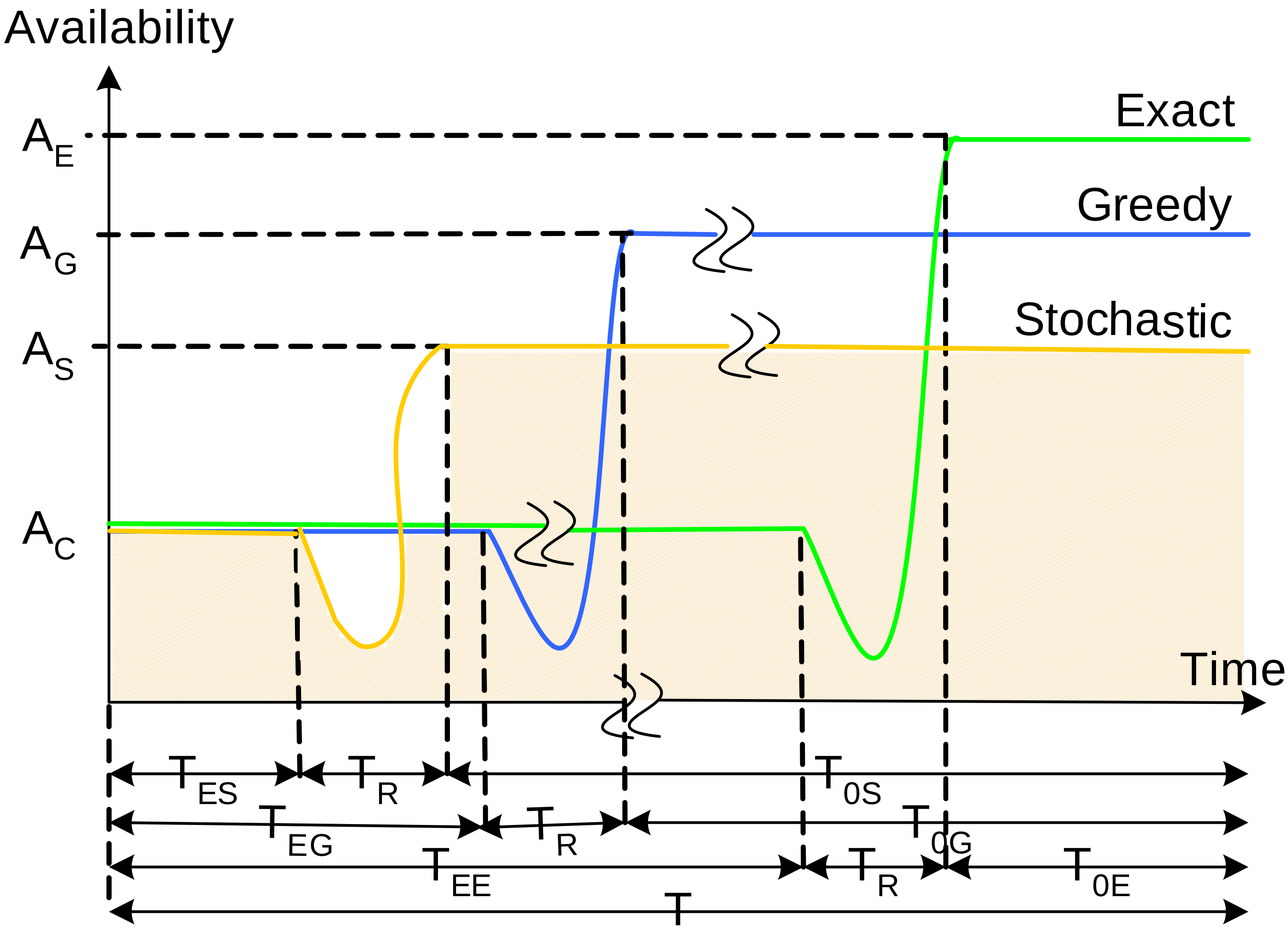
- *Exact – exponential complexity*
- Stochastic – quadratic complexity
- Adaptive greedy – cubic complexity
- *Decentralized – quadratic complexity*



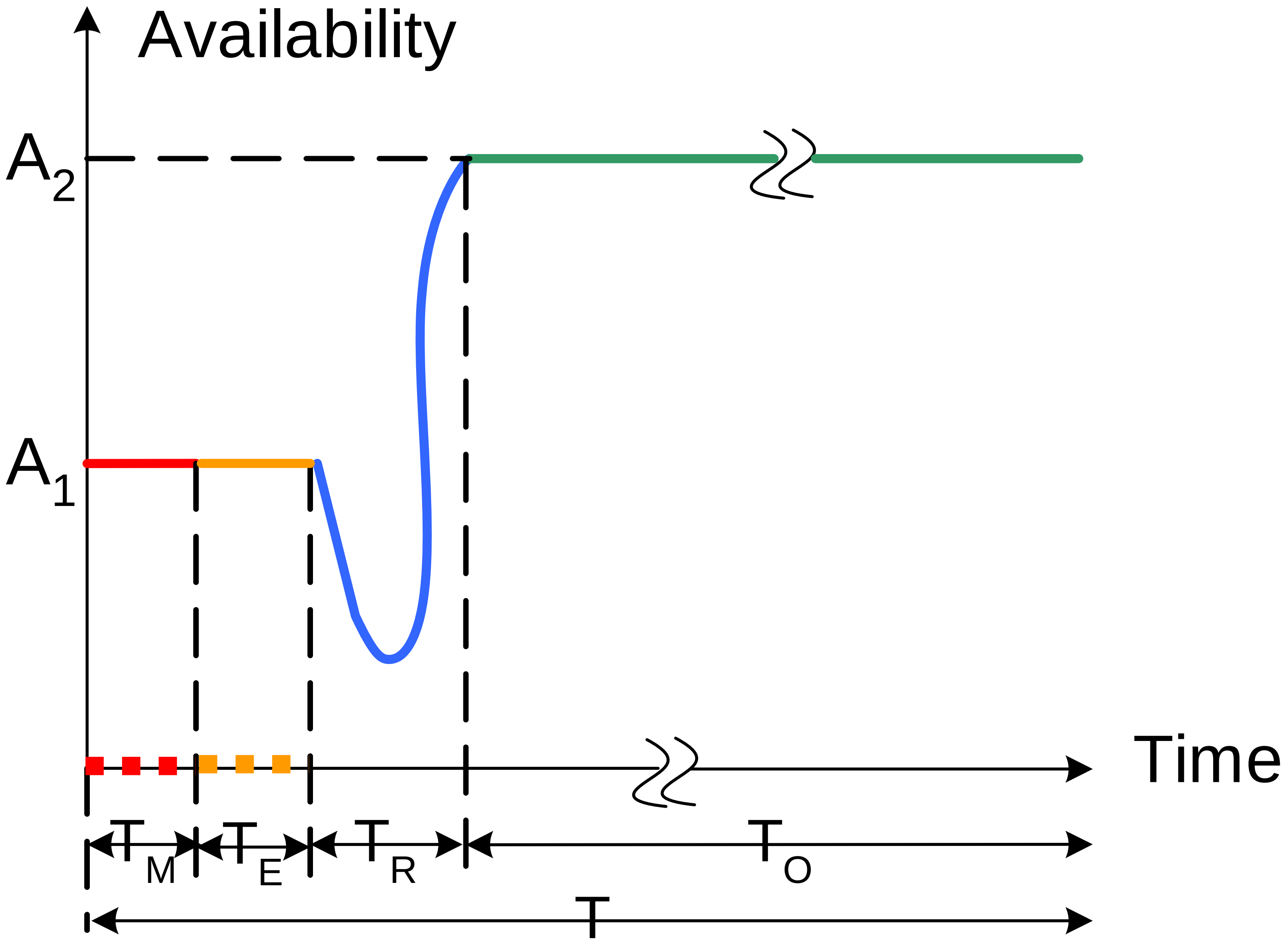
Automatic Algorithm Selection



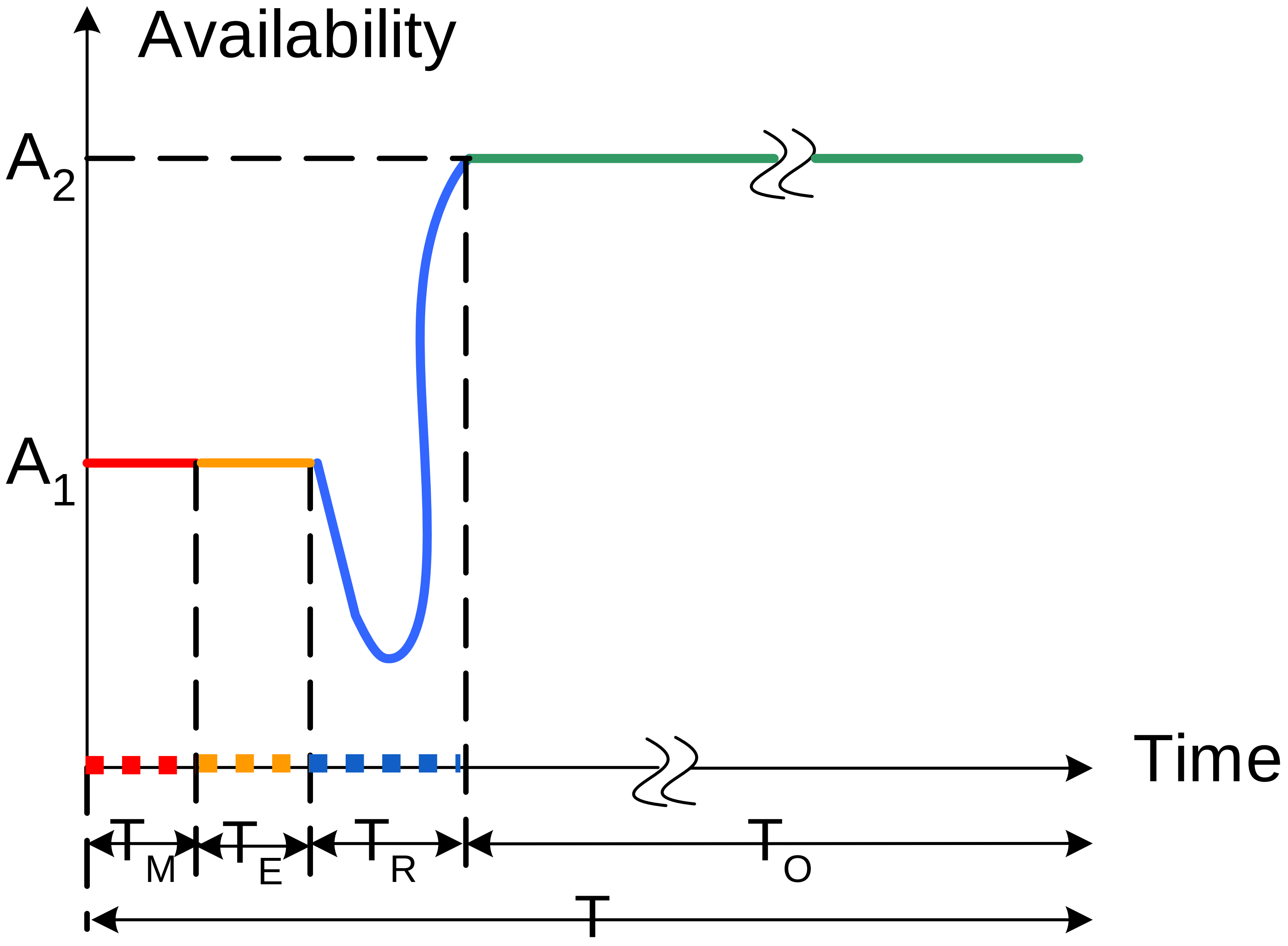
Automatic Algorithm Selection



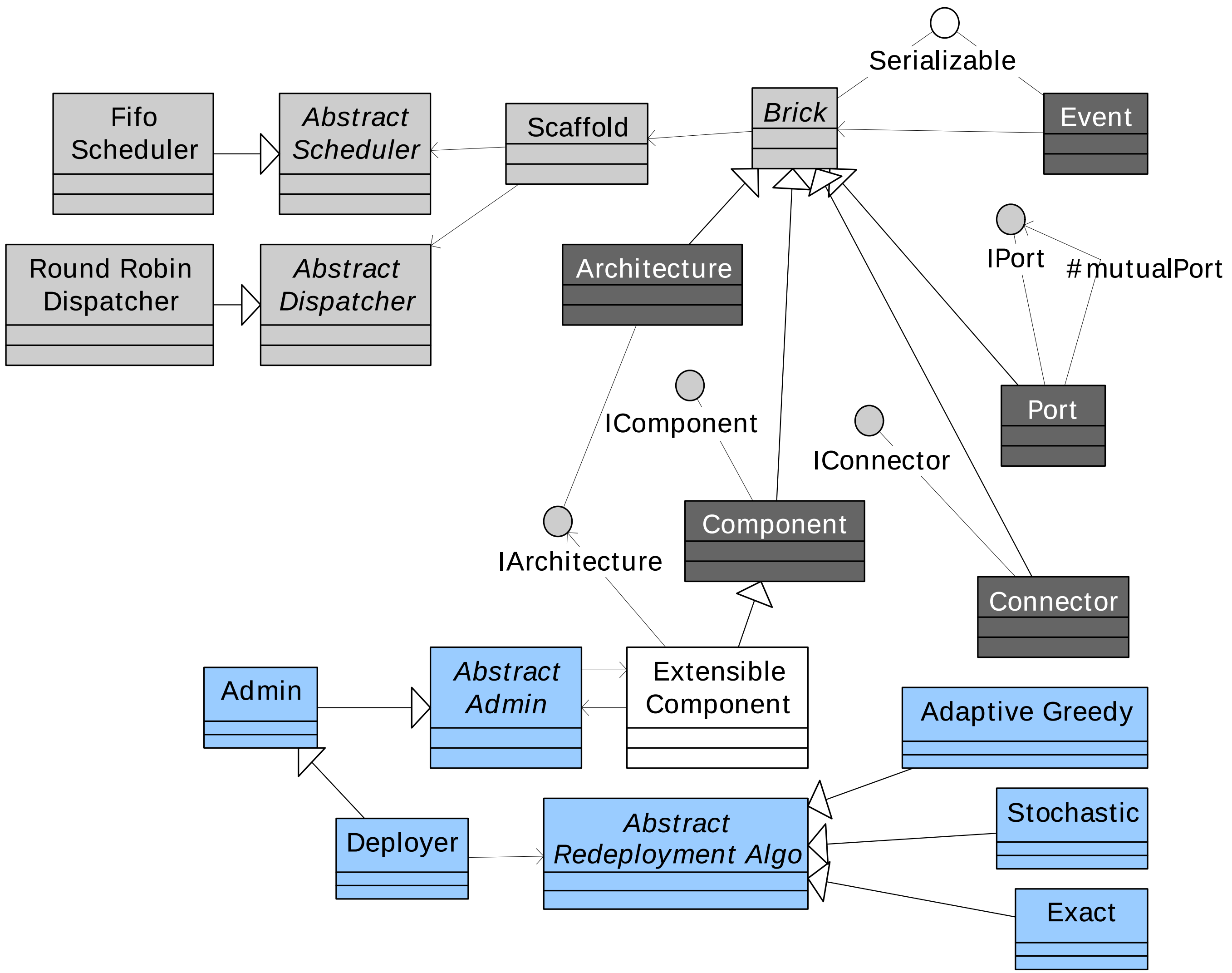
Then Apply the Solution



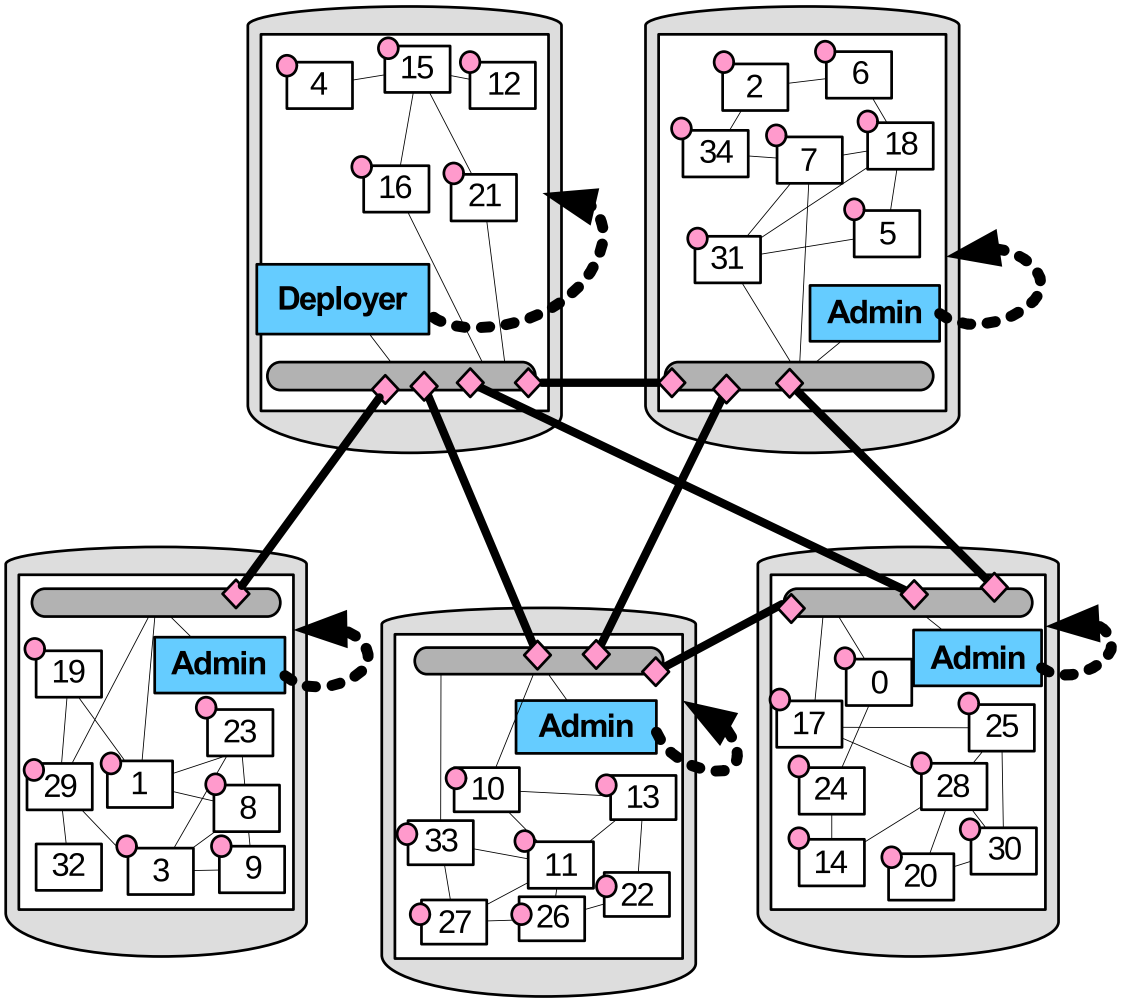
Then Apply the Solution



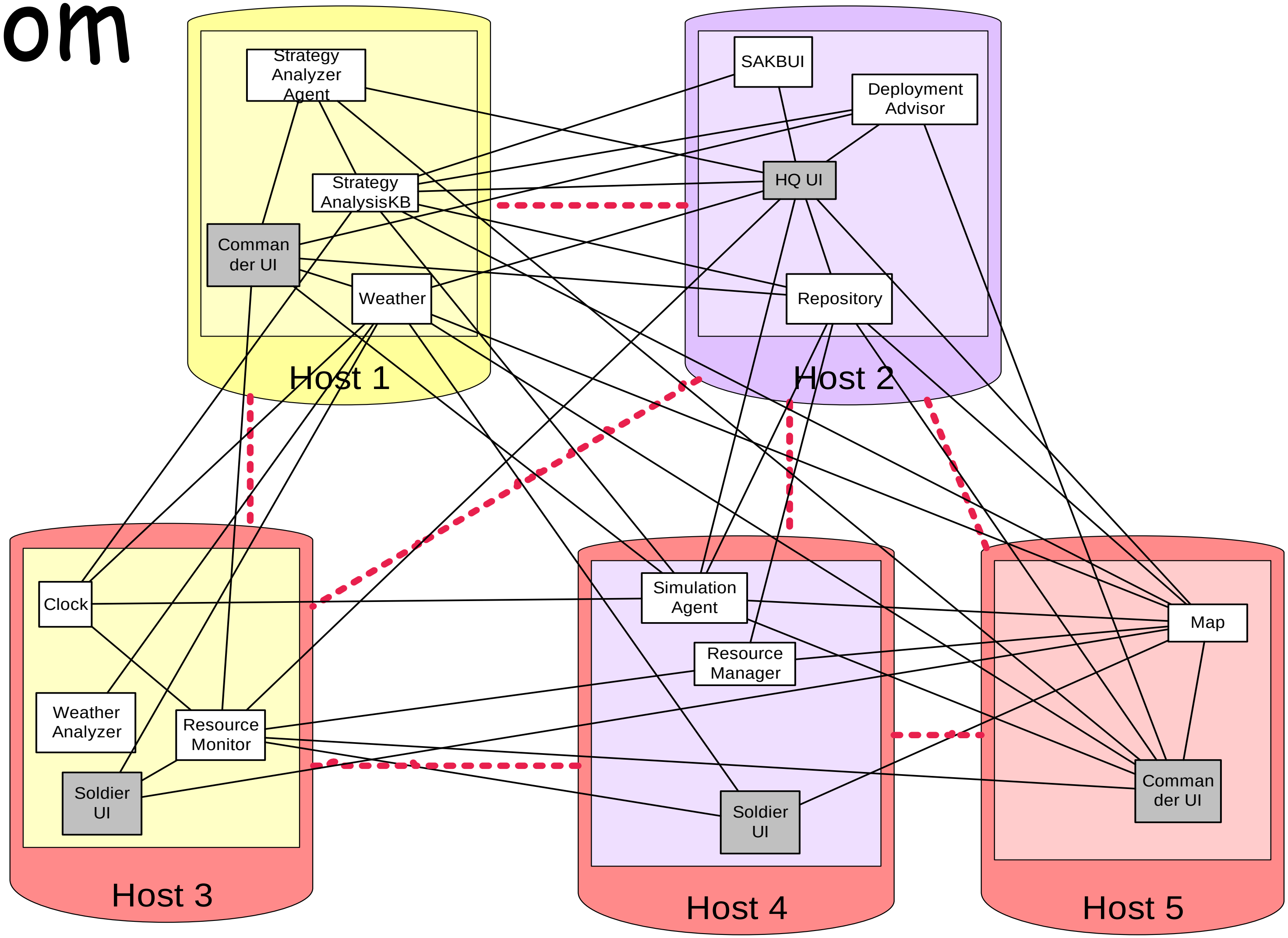
Redeployment in Prism-MW



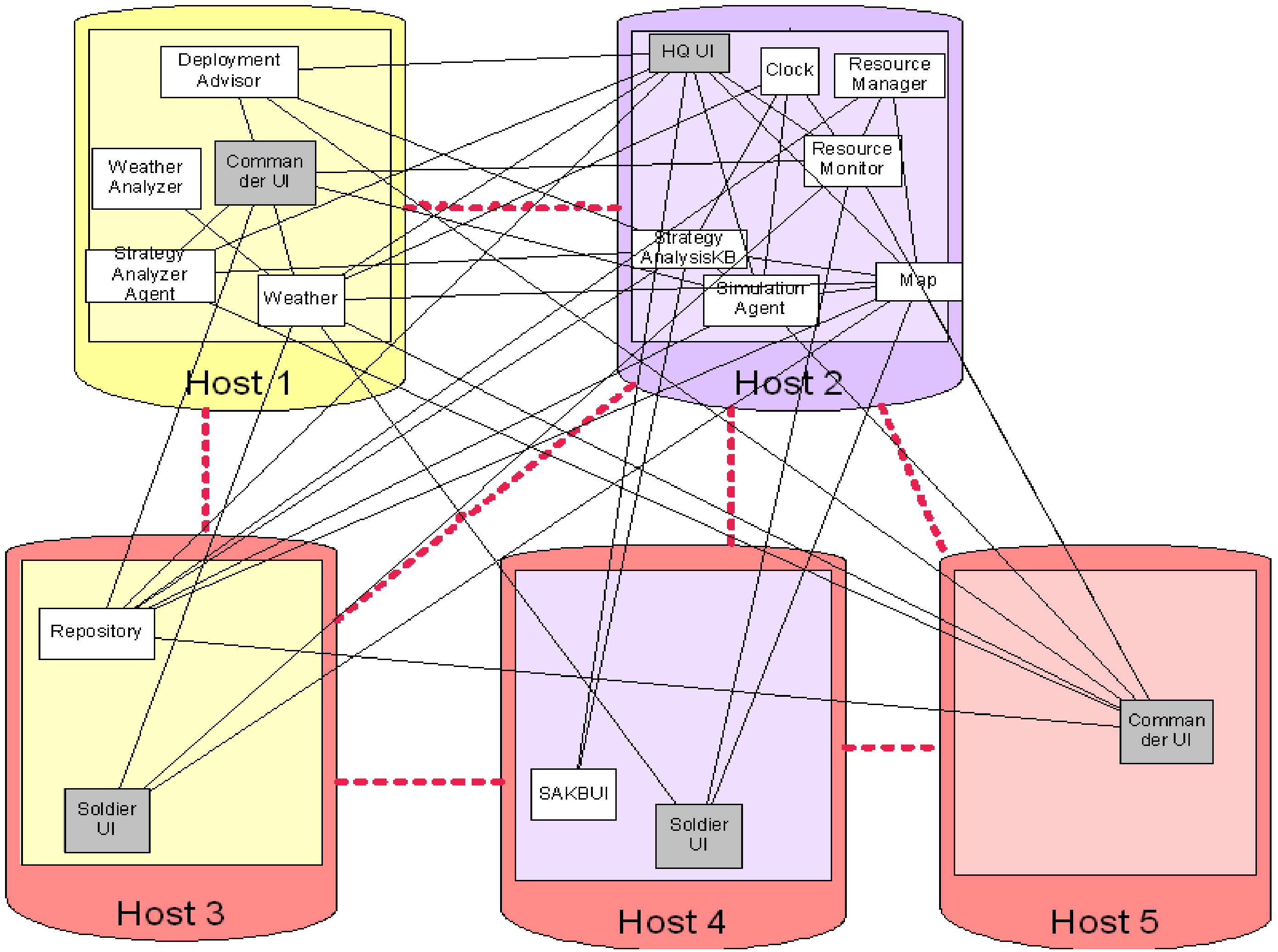
Redeployment in Action



Going from


Strategy Analyzer Agent
Strategy AnalysisKB
Commander UI
Weather
Host 1
SAKBUI
Deployment Advisor
HQ UI
Repository
Host 2
Clock
Weather Analyzer
Resource Monitor
Soldier UI
Host 3
Simulation Agent
Resource Manager
Soldier UI
Host 4
Map
Commander UI
Host 5

to



Deployment Advisor
Weather Analyzer
Commander UI
Strategy Analyzer Agent
Weather
Host 1
HQ UI
Clock
Resource Manager
Resource Monitor
Strategy AnalysisKB
Simulation Agent
Map
Host 2
Repository
Soldier UI
Host 3
SAKBUI
Soldier UI
Host 4
Commander UI
Host 5

Outline

- From architectures to systems
- Ensuring dependability
  - Problem definition
  - Proposed solution
- Concluding remarks

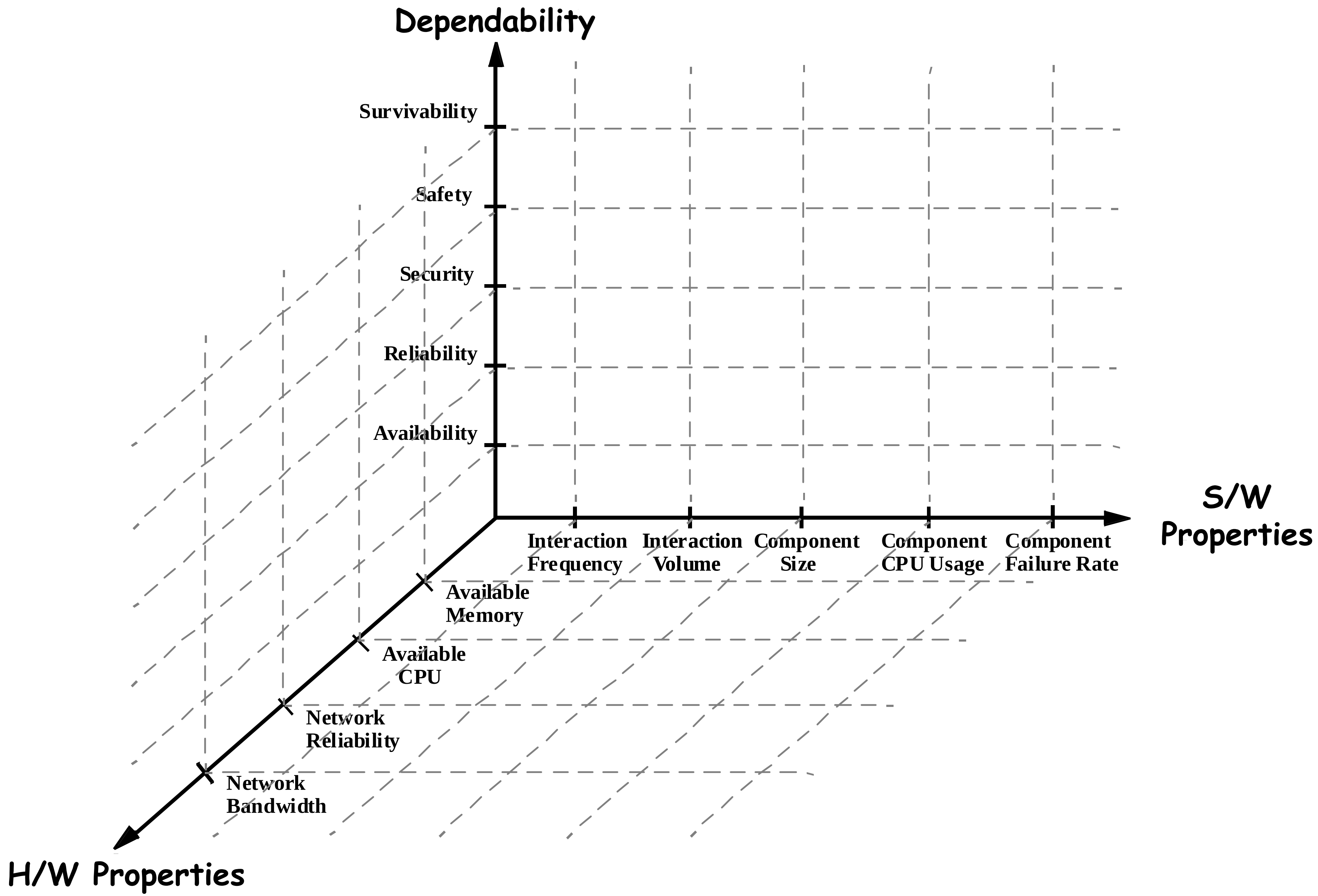
Concluding Remarks

- Still so much to do
  - Enriching/completing the models
  - Focusing on additional aspects of dependability
  - Addressing feature interactions
  - Addressing emergent properties
  - Determining which concerns are (not) architectural

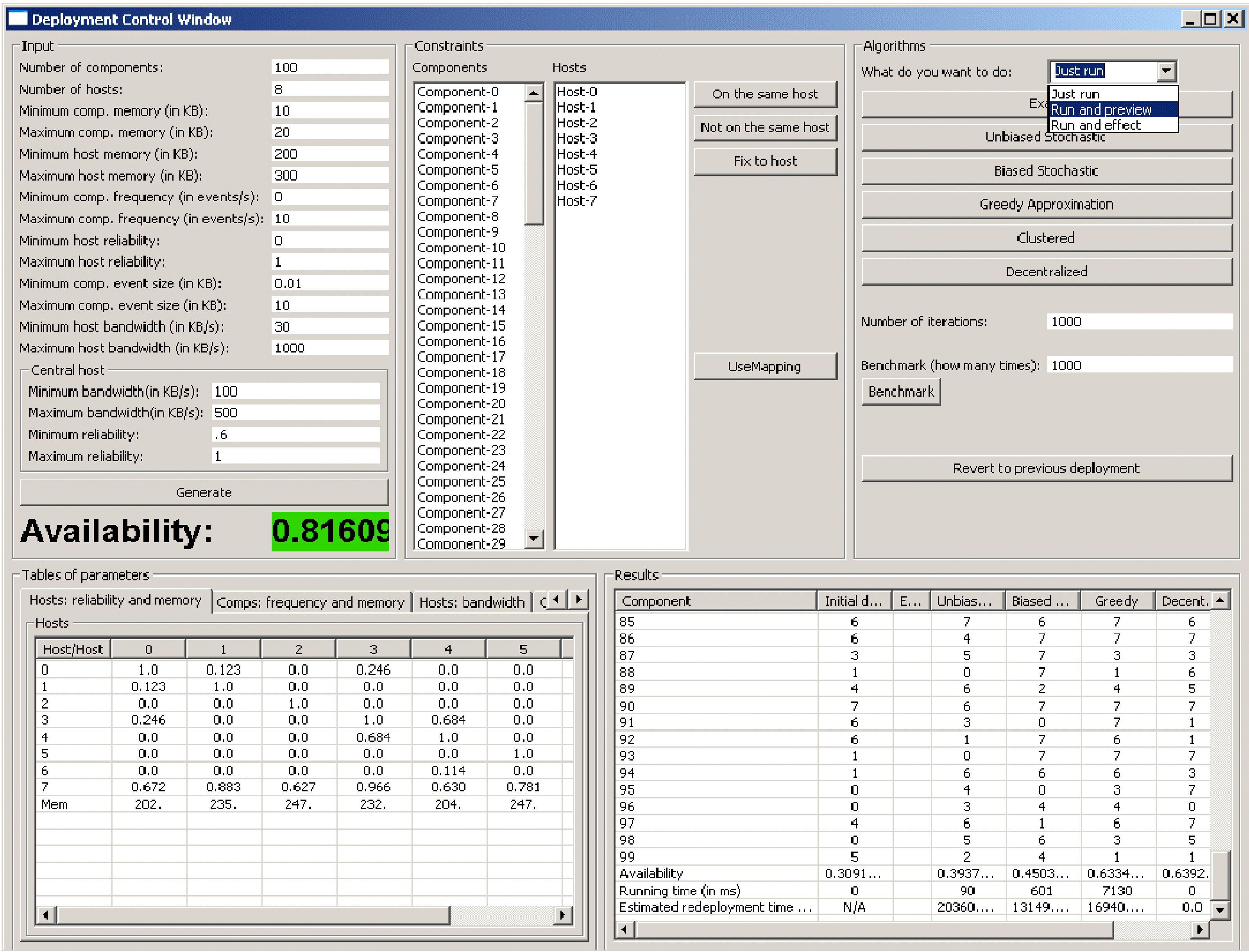
Promise or Illusion?

- If we can define it, we should be able to
  - Analyze for it
  - Build it
  - Measure it
  - Act on it
- So, why haven't we done it yet?

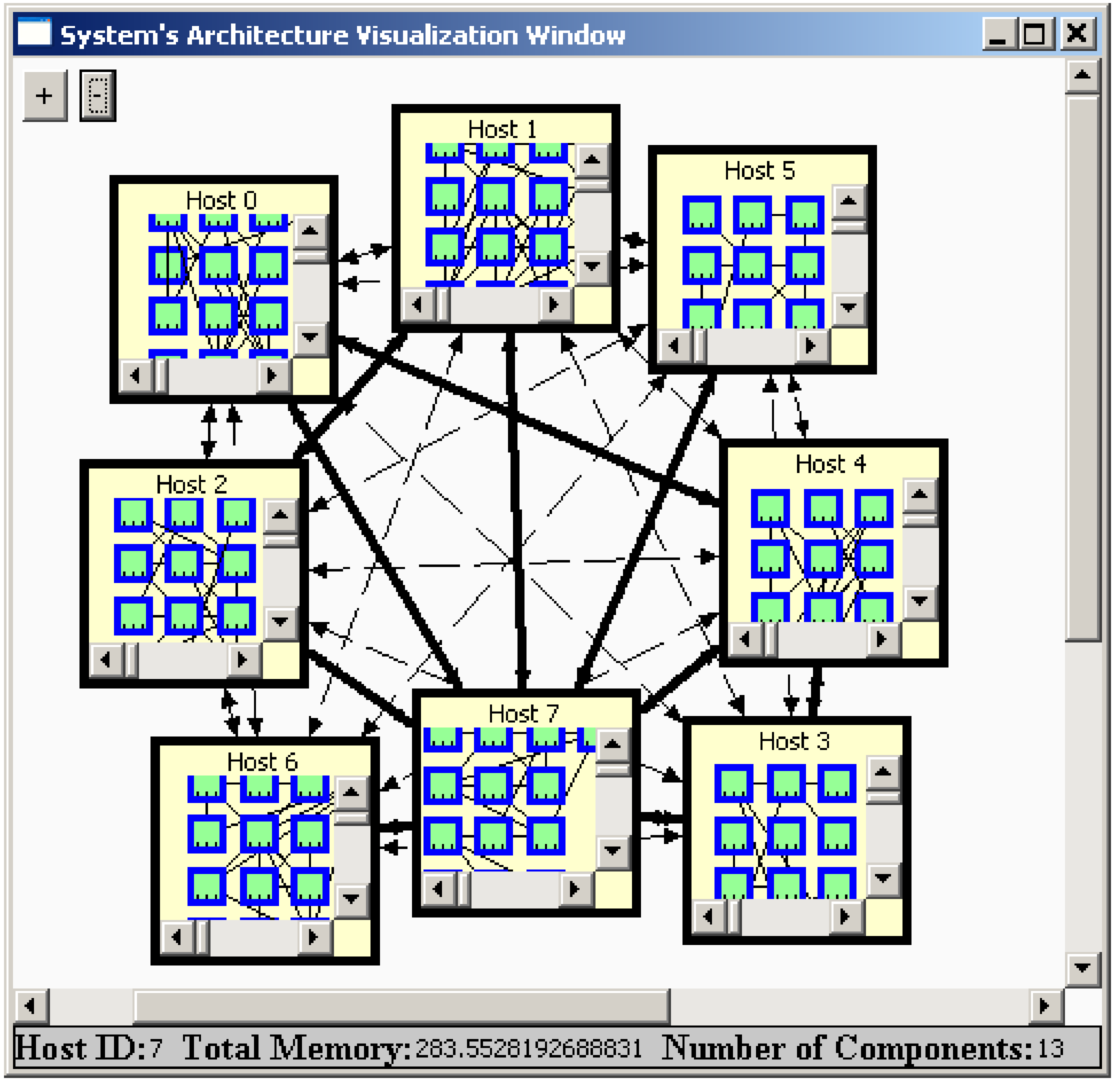
The Playing Field



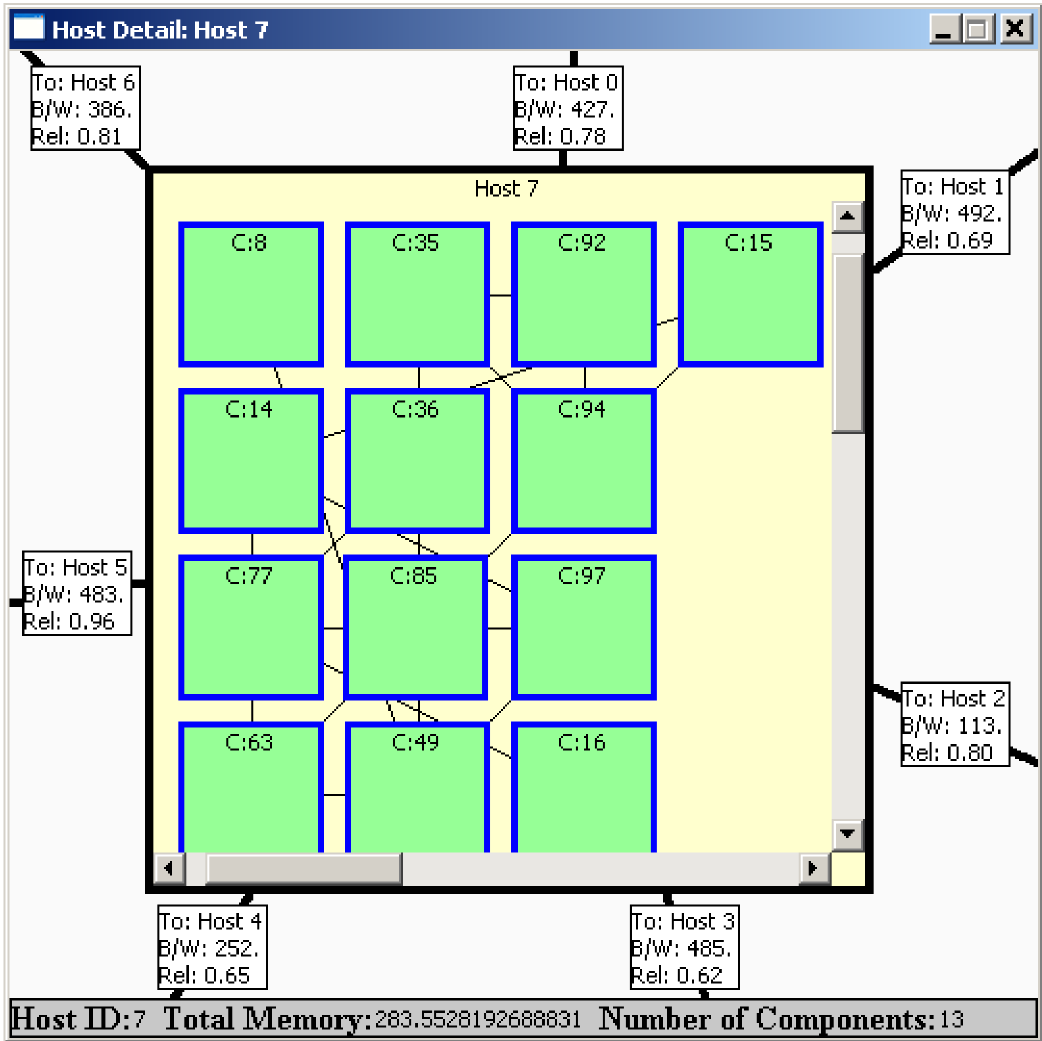
Exploring the Problem Space



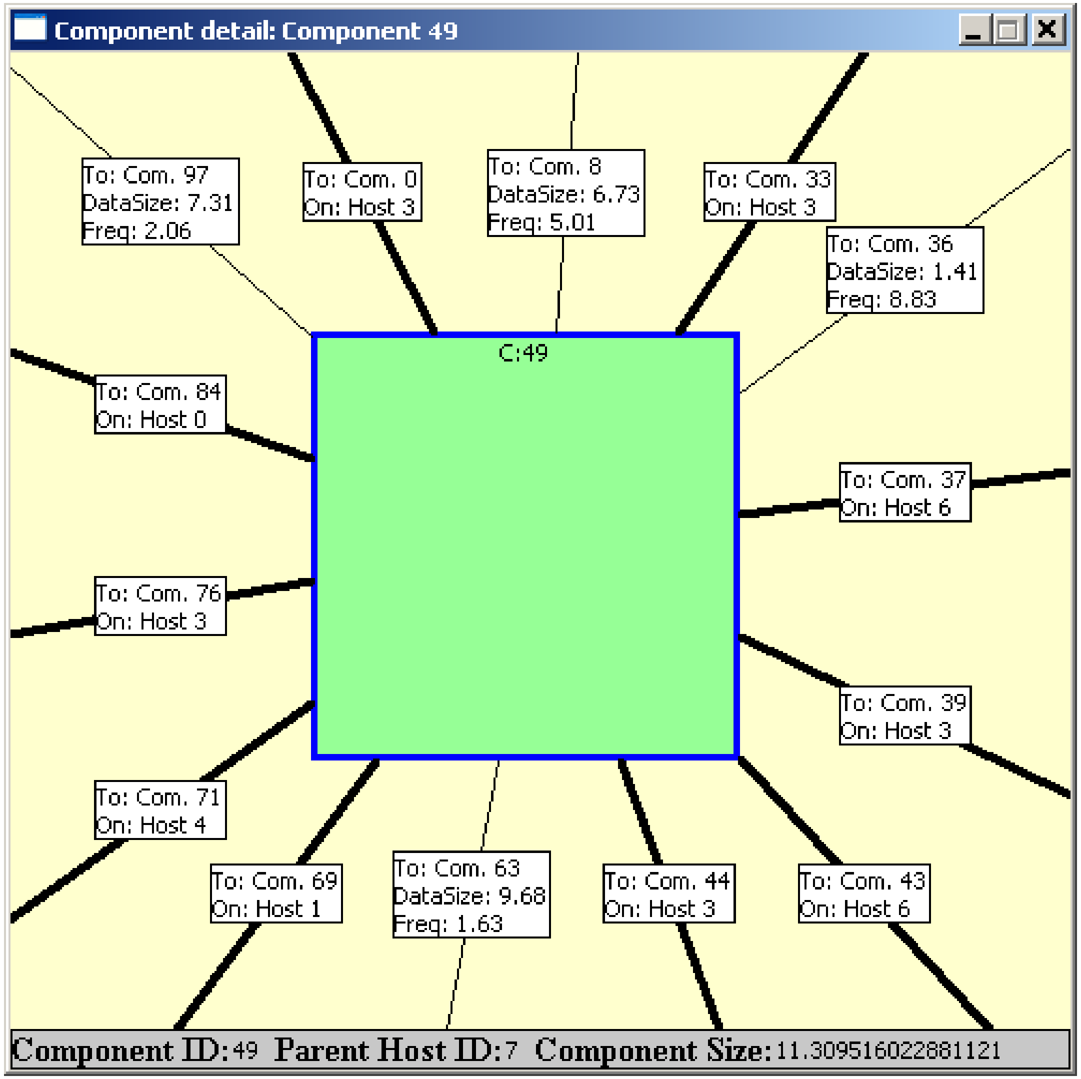
Exploring the Problem Space



Exploring the Problem Space

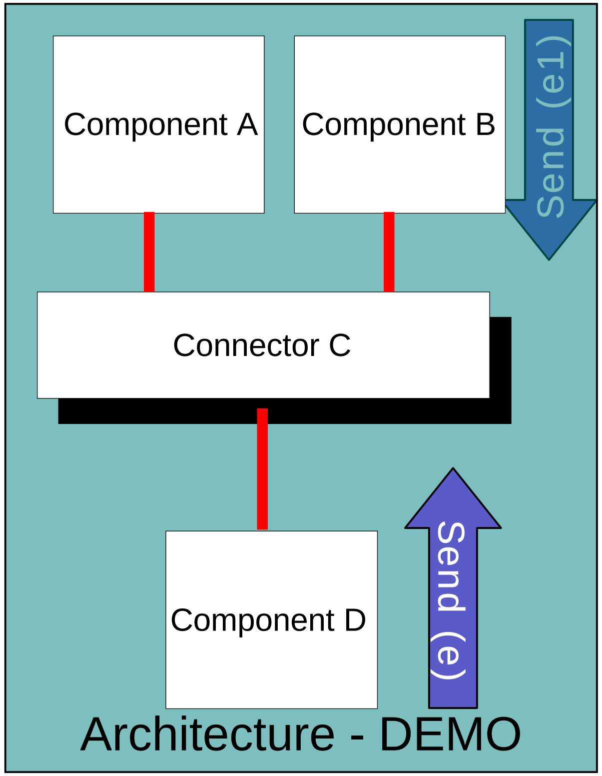


Exploring the Problem Space



Questions?

Using Prism-MW



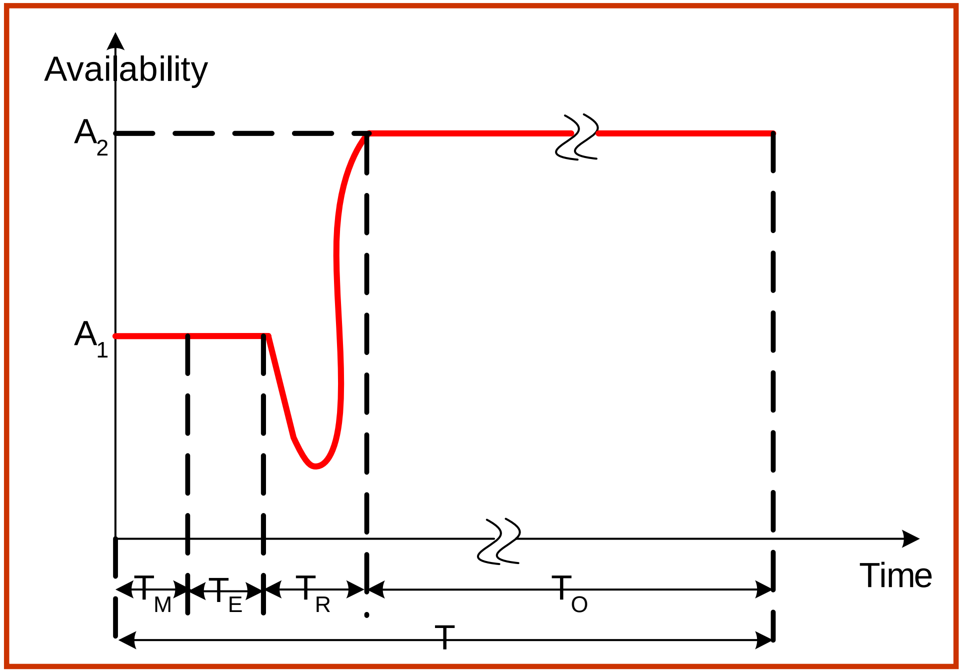
Component D sends an event

```
Event e = new Event ("Event_D");
e.addParameter("param_1", p1);
send (e);
```

Component B handles the event and sends a response

```
public void handle(Event e)
{
        if (e.equals("Event_D")) {
        ...
        Event e1= new Event("Response_to_D");
        e1.addParameter("response", resp);
        send(e1);
        }...
    }
```

Assumptions



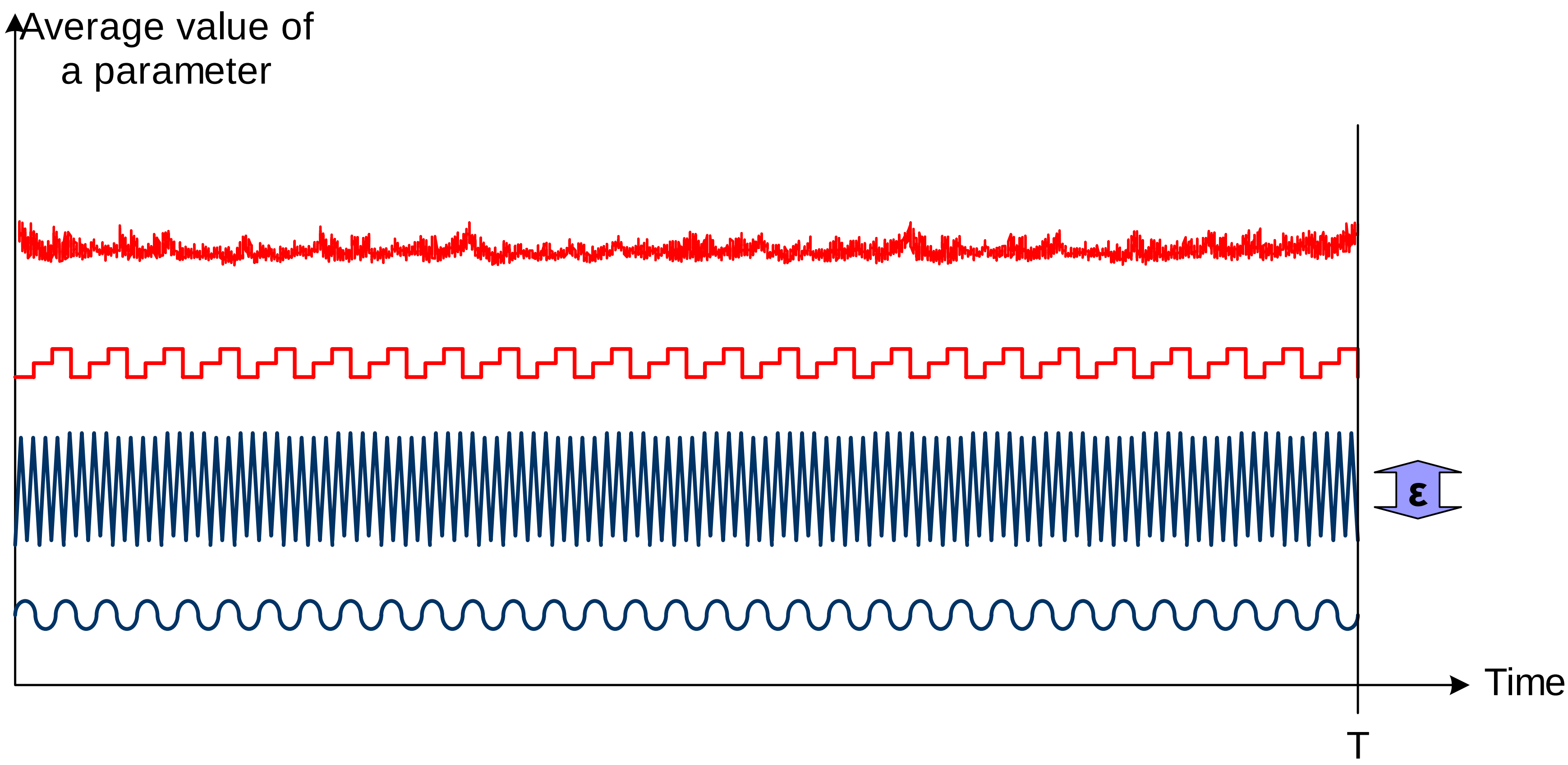
$$T_M + T_E + T_R << T_0$$

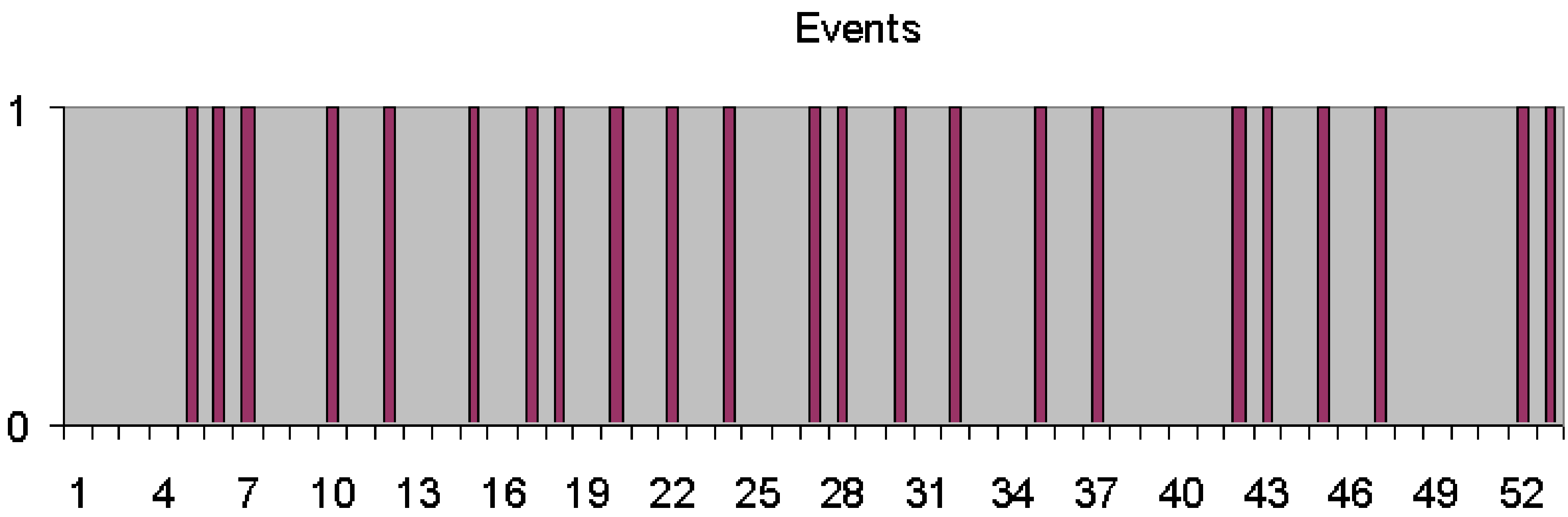
- *Possible in small amounts of time even for slow links*

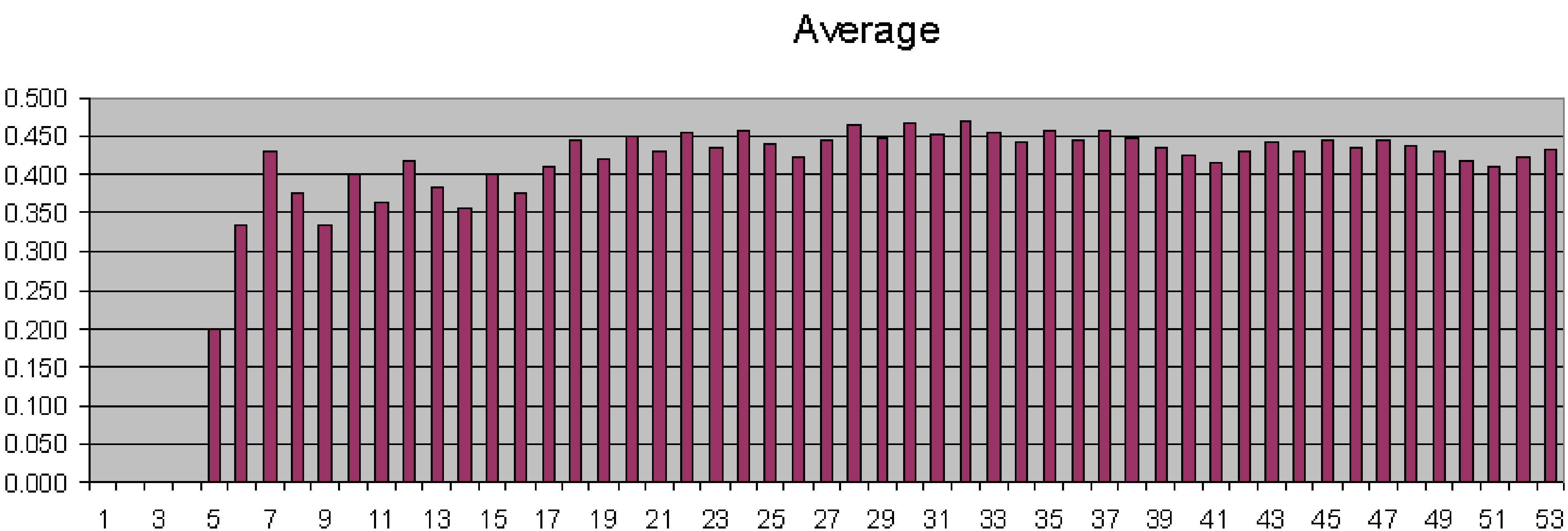
For example,

- *dialup (56K), 50% reliability, 100K in 28 sec*
- *1M, 50% reliability, 100K in less than 0.7 sec*

Assumptions




Events
1
0
1
4
7
10
13
16
19
22
25
28
31
34
37
40
43
46
49
52


Average
0.500
0.450
0.400
0.350
0.300
0.250
0.200
0.150
0.100
0.050
0.000
1
3
5
7
9
11
13
15
17
19
21
23
25
27
29
31
33
35
37
39
41
43
45
47
49
51
53

What Is Needed?

- Scalable and lightweight support for
  - distributed architectures
  - with arbitrary topologies
- Support for automatic monitoring
- Efficient and precise solutions for the exponentially complex redeployment problem
- Support for automatic redeployment

Problem breakdown

Problem breakdown

1) **Lack of knowledge about runtime system parameters**
   - unknown at the time of initial deployment
   - change at runtime
   - **architectural middleware with monitoring support**

Problem breakdown

1) **Lack of knowledge about runtime system parameters**
   - unknown at the time of initial deployment
   - change at runtime
   - **architectural middleware with monitoring support**

2) **Exponentially complex problem**
   - n components and k hosts = k^n possible deployments
   - system constraints further complicate the problem
   - **polynomial-time approximating algorithms**

Problem breakdown

1) **Lack of knowledge about runtime system parameters**
   - unknown at the time of initial deployment
   - change at runtime
   - **architectural middleware with monitoring support**

2) **Exponentially complex problem**
   - n components and k hosts = k^n possible deployments
   - system constraints further complicate the problem
   - **polynomial-time approximating algorithms**

3) **Assessing deployment architectures**
   - comparison of different algorithms
   - performance vs. complexity, sensitivity analysis, etc.
   - **analysis and simulation environment**

Problem breakdown

1) **Lack of knowledge about runtime system parameters**
   - unknown at the time of initial deployment
   - change at runtime
   - **architectural middleware with monitoring support**

2) **Exponentially complex problem**
   - n components and k hosts = k^n possible deployments
   - system constraints further complicate the problem
   - **polynomial-time approximating algorithms**

3) **Assessing deployment architectures**
   - comparison of different algorithms
   - performance vs. complexity, sensitivity analysis, etc.
   - **analysis and simulation environment**

4) **Effecting the selected solution**
   - redeploying components
   - requires an automated solution
   - **architectural middleware with deployment support**

Why the Problem Isn't Solved