Improving Availability of Distributed Event-Based Systems via Run-Time Monitoring and Analysis

Sam Malek
and
Marija Mikic-Rakic
Nels Beckman
Nenad Medvidovic

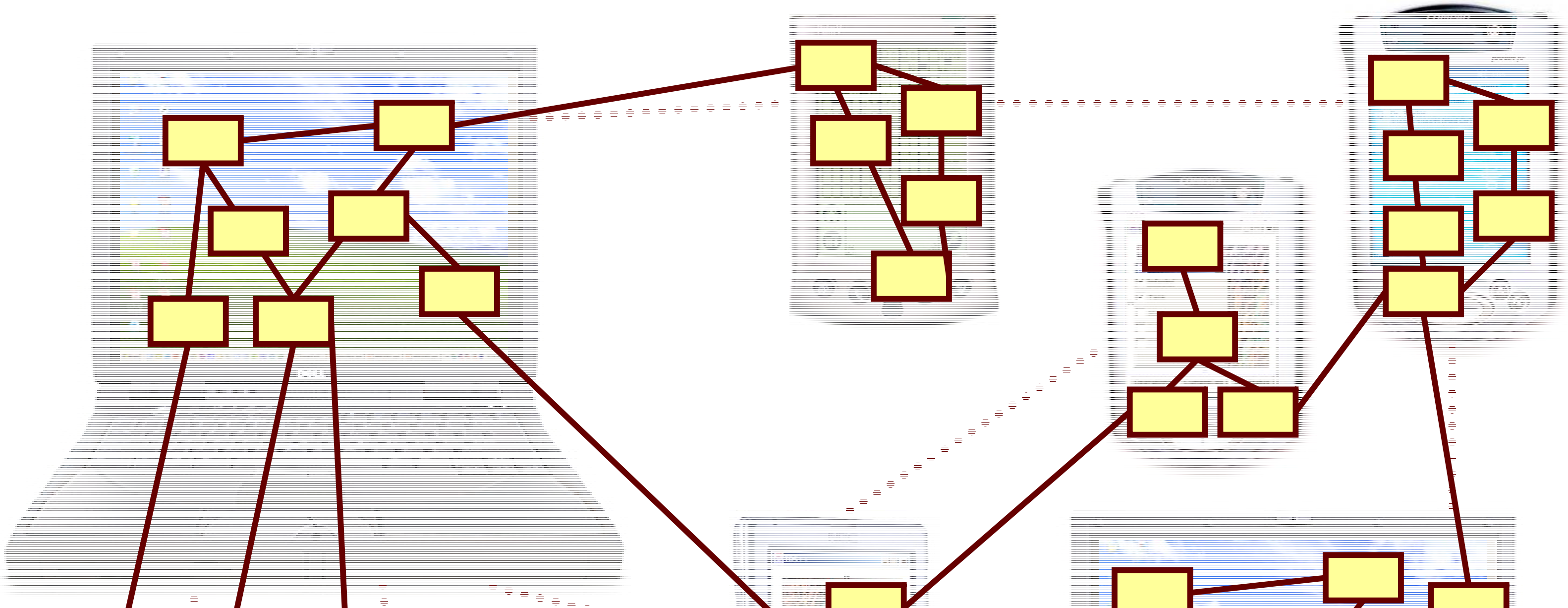
University of Southern California

SoftArch
USC-CSE

Outline

- Motivation
- Problem description
- Prism-MW
- DeSi
- Algorithms
- Concluding remarks

Motivation



How good is this deployment architecture?

What are its properties?

How should it be modified to ensure higher availability?

Outline

- Motivation
- Problem description
- Prism-MW
- DeSi
- Algorithms
- Concluding remarks

Problem description

Given system model parameters:

- Software component properties
  - Memory requirements
  - Frequency of interaction
  - Size of the exchanged data
- Hardware host properties
  - Memory capacity
  - Network reliability
  - Network bandwidth
- Constraints
  - Location
  - Co-location

Problem description

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

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

is maximized, and the deployment is valid.

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

Problem breakdown

1) **Lack of knowledge about runtime system model parameters**
   - System model parameters not known at the time of initial deployment
   - System model parameters change at runtime
   - **Middleware with monitoring support**

2) **Exponentially complex problem**
   - n components and k hosts = k^n possible deployments
   - **Polynomial time approximating algorithms**

3) **Environment for assessing deployments**
   - Comparison of different solutions and algorithms
   - performance vs. complexity, sensitivity analysis, etc
   - **Analysis and visualization utilities**

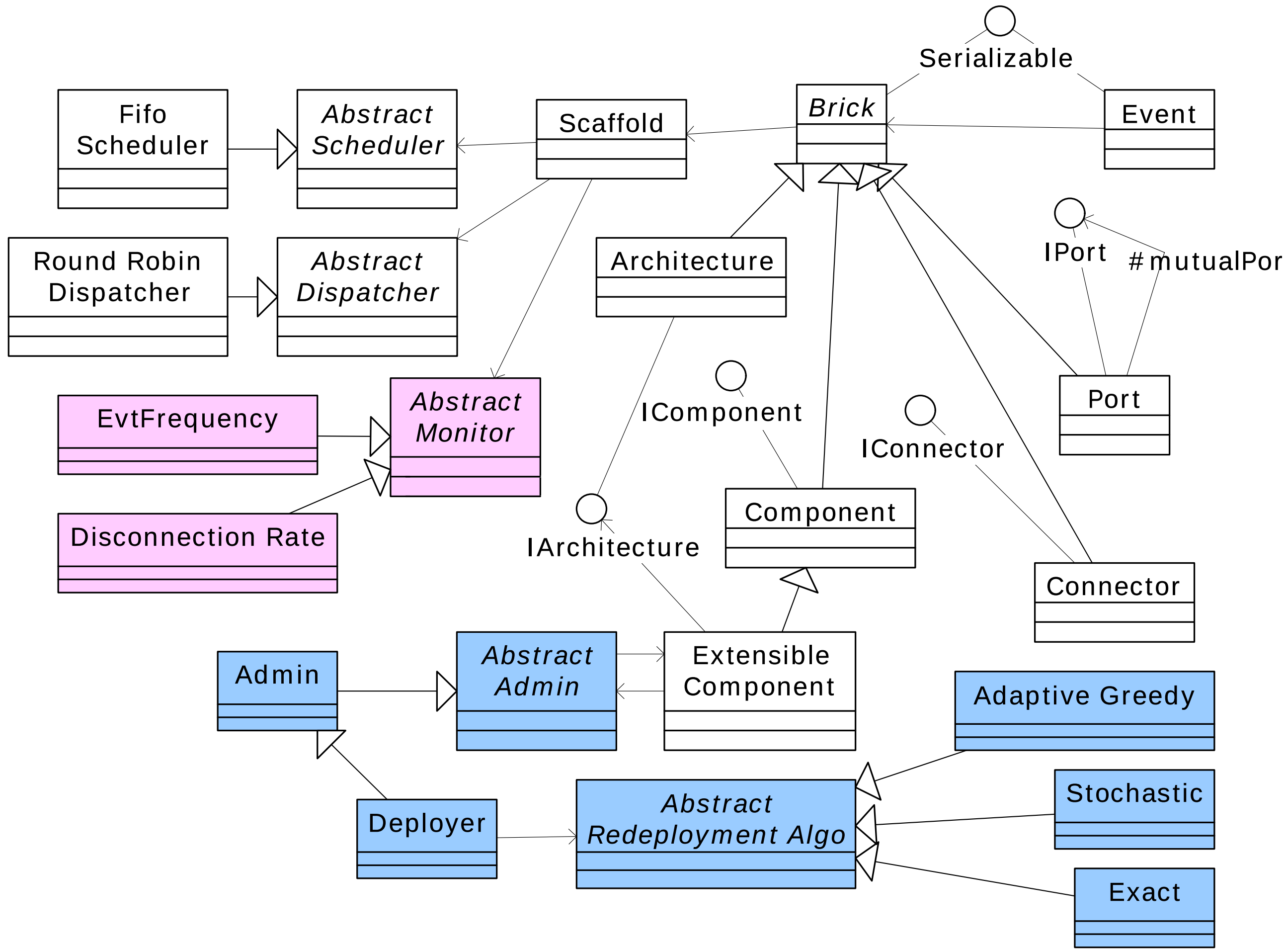
4) **Effecting the selected solution**
   - Redeploying components
   - Requires an automated solution
   - **Middleware with deployment support**

Outline

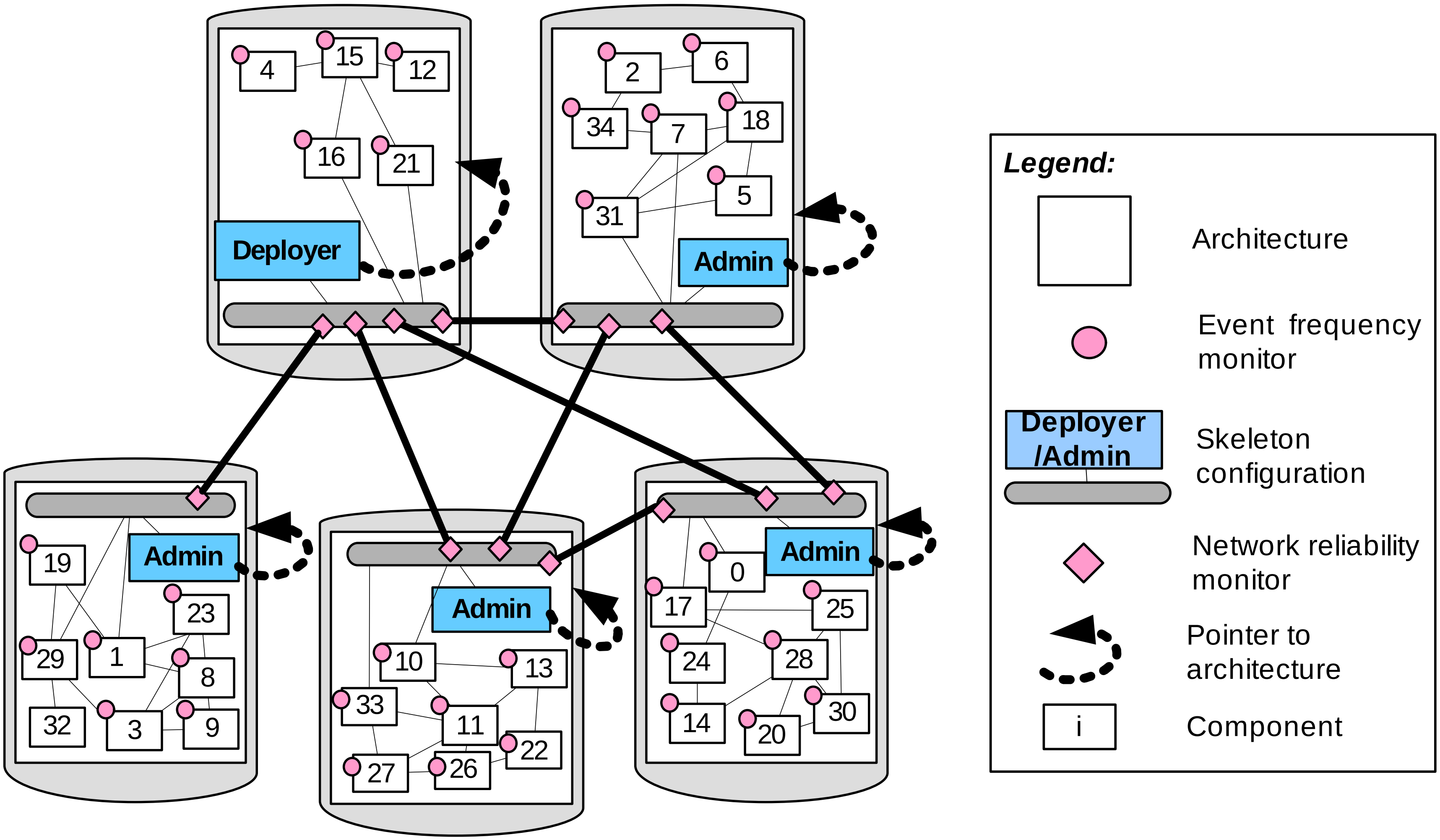
- Motivation
- Problem description
- Prism-MW
- DeSi
- Algorithms
- Concluding remarks

Prism middleware

- An architectural middleware
- Enables implementation and deployment of distributed systems in terms of their architectural elements
- Support for monitoring and redeployment

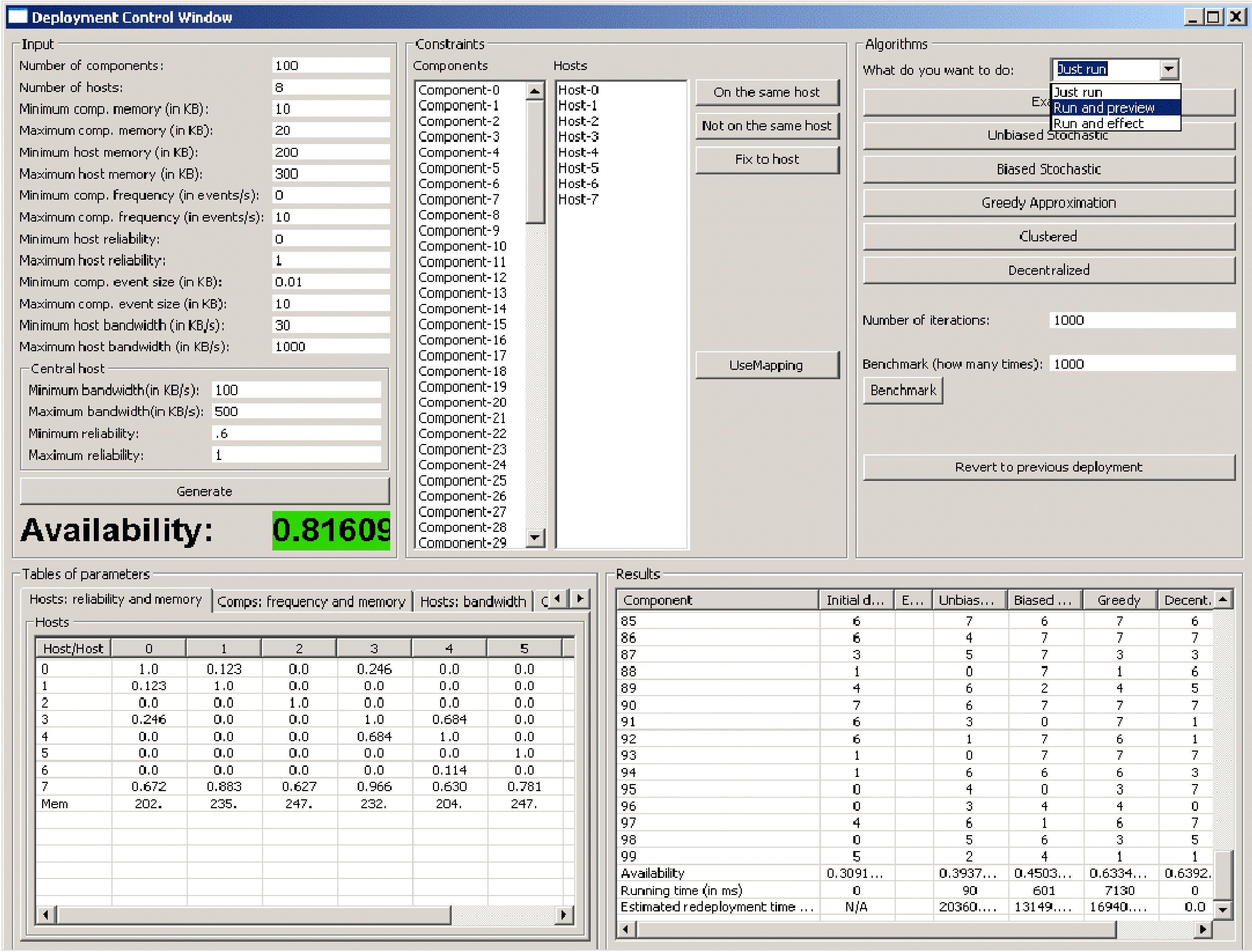


Monitoring and redeploying

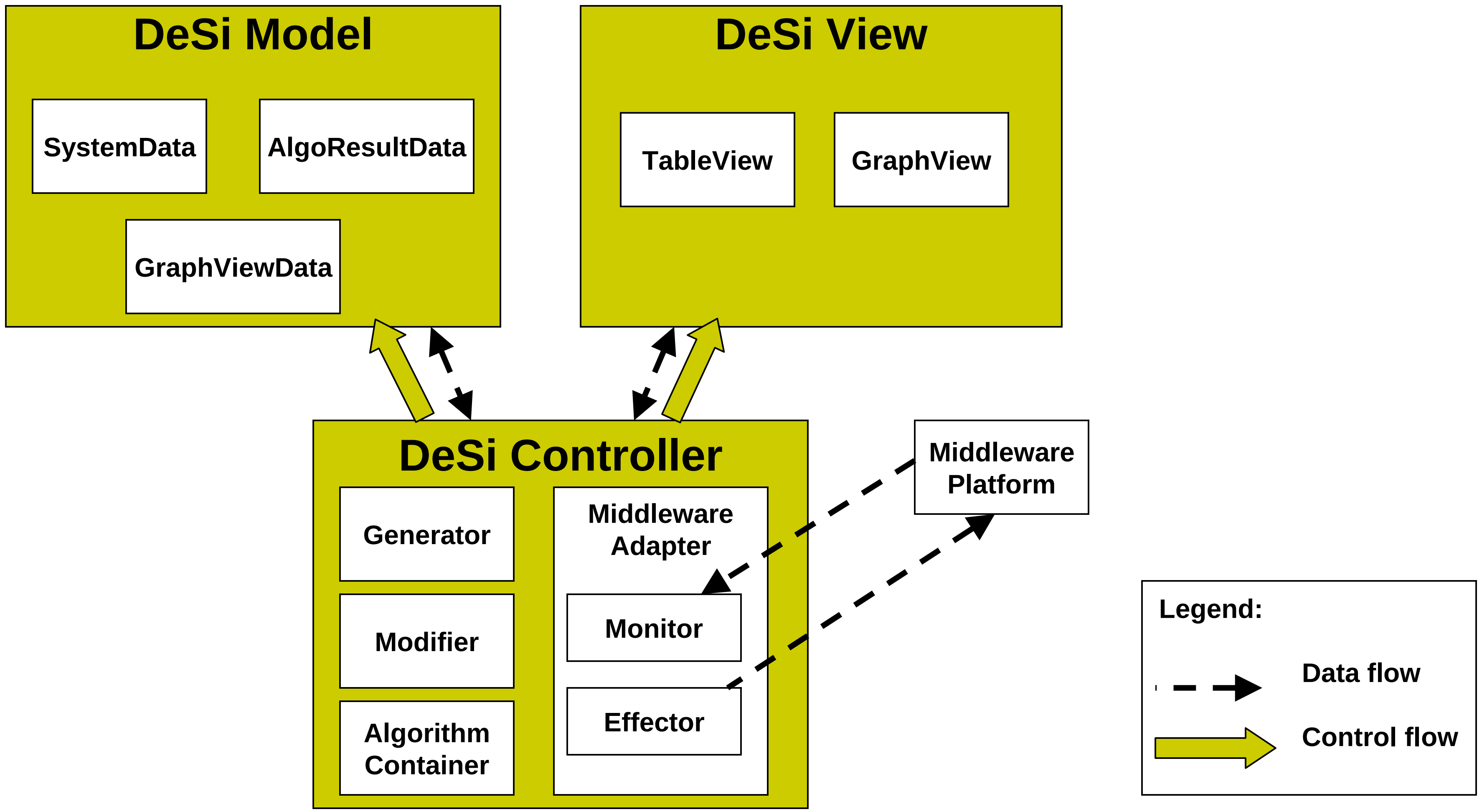


DeSi

- Deployment simulation environment
  - Specification and generation of deployment architectures
  - Visualization and analysis of distributed system
  - Estimation of the quality of deployment
  - Facilitation of rapid development and comparison of algorithms


Deployment Control Window
Input
Number of components: 100
Number of hosts: 8
Minimum comp. memory (in KB): 10
Maximum comp. memory (in KB): 20
Minimum host memory (in KB): 200
Maximum host memory (in KB): 300
Minimum comp. frequency (in events/s): 0
Maximum comp. frequency (in events/s): 10
Minimum host reliability: 0
Maximum host reliability: 1
Minimum comp. event size (in KB): 0.01
Maximum comp. event size (in KB): 10
Minimum host bandwidth (in KB/s): 30
Maximum host bandwidth (in KB/s): 1000
Central host
Minimum bandwidth(in KB/s): 100
Maximum bandwidth(in KB/s): 500
Minimum reliability: .6
Maximum reliability: 1
Generate
Availability: 0.81609
Constraints
Components
Component-0 … Component-29
Hosts
Host-0 … Host-7
On the same host
Not on the same host
Fix to host
UseMapping
Algorithms
What do you want to do: Just run
Just run
Run and preview
Run and effect
Unbiased Stochastic
Biased Stochastic
Greedy Approximation
Clustered
Decentralized
Number of iterations: 1000
Benchmark (how many times): 1000
Benchmark
Revert to previous deployment
Tables of parameters
Hosts: reliability and memory
Comps: frequency and memory
Hosts: bandwidth
Hosts
Host/Host 0 1 2 3 4 5
0 1.0 0.123 0.0 0.246 0.0 0.0
1 0.123 1.0 0.0 0.0 0.0 0.0
2 0.0 0.0 1.0 0.0 0.0 0.0
3 0.246 0.0 0.0 1.0 0.684 0.0
4 0.0 0.0 0.0 0.684 1.0 0.0
5 0.0 0.0 0.0 0.0 0.0 1.0
6 0.0 0.0 0.0 0.0 0.114 0.0
7 0.672 0.883 0.627 0.966 0.630 0.781
Mem 202. 235. 247. 232. 204. 247.
Results
Component Initial d… E… Unbias… Biased … Greedy Decent.
85 6 7 6 7 6
86 6 4 7 7 7
87 3 5 7 3 3
88 1 0 7 1 6
89 4 6 2 4 5
90 7 6 7 7 7
91 6 3 0 7 1
92 6 1 7 6 1
93 1 0 7 7 7
94 1 6 6 6 3
95 0 4 0 3 7
96 0 3 4 4 0
97 4 6 1 6 7
98 0 5 6 3 5
99 5 2 4 1 1
Availability 0.3091… 0.3937… 0.4503… 0.6334… 0.6392.
Running time (in ms) 0 90 601 7130 0
Estimated redeployment time … N/A 20360.… 13149.… 16940.… 0.0

DeSi's architecture



Suite of algorithms

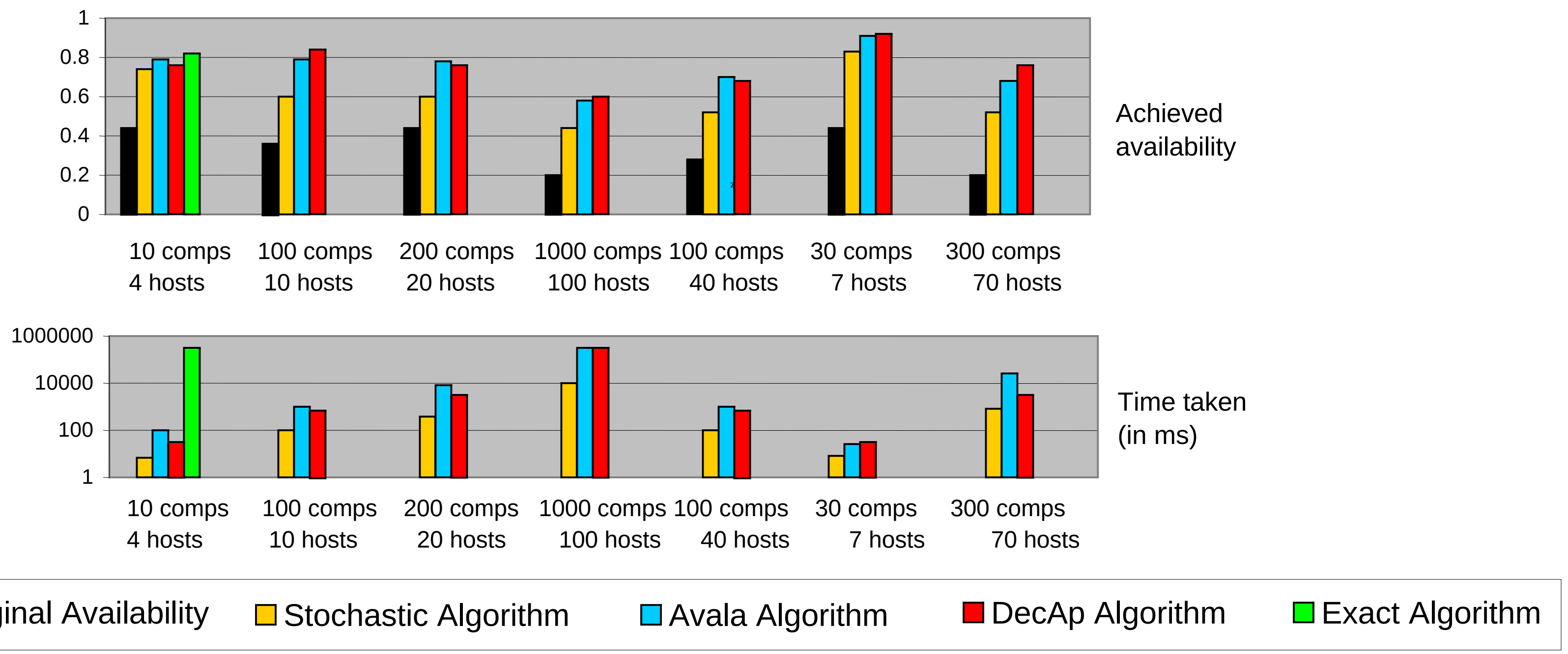
Exact – finds optimal solution *O(k^n)*

Biased/Unbiased stochastic – random selection *O(n^2)*

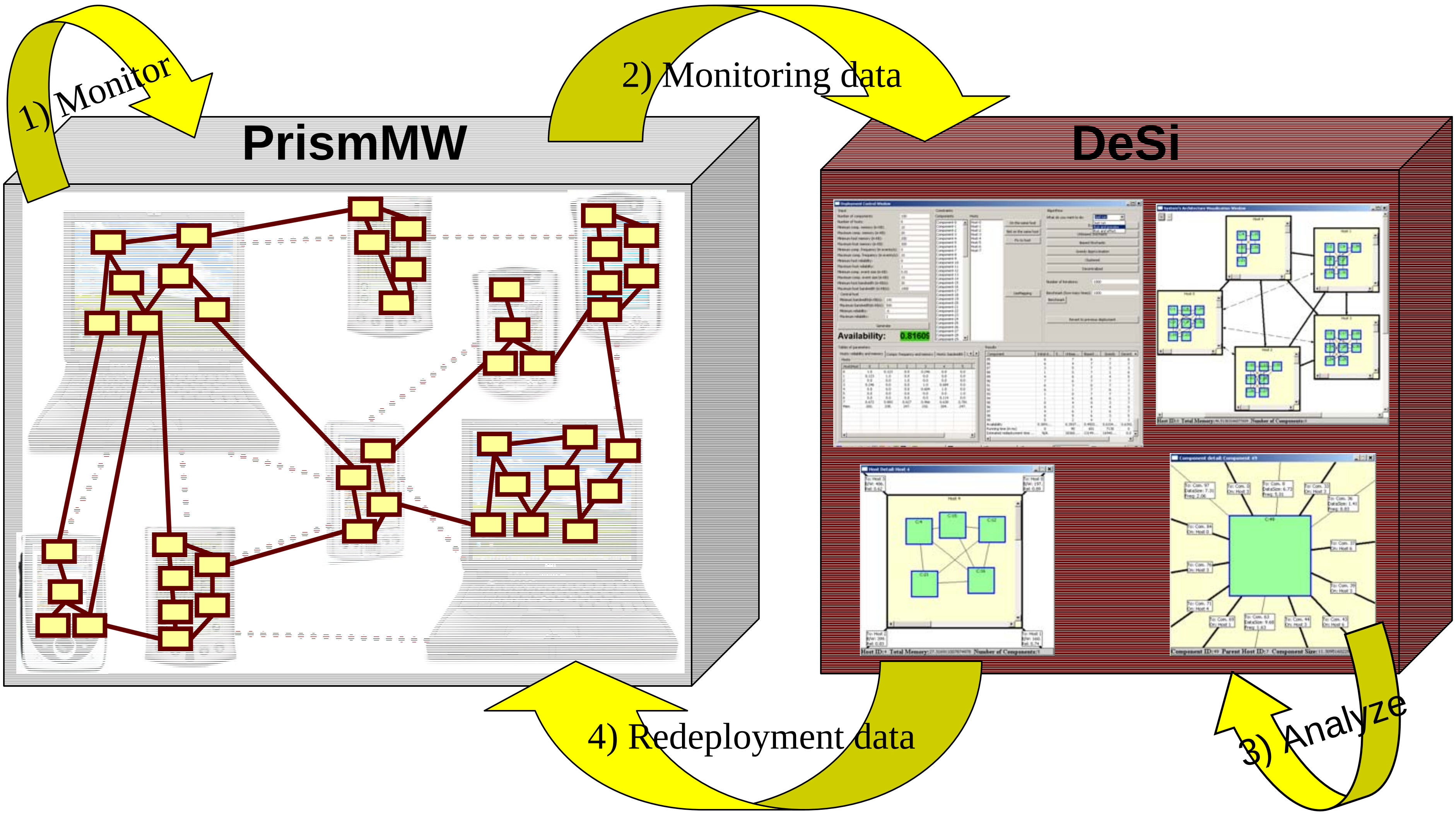
Avala – greedy approximation *O(n^3)*

DecAp – decentralized auction based *O(n^3)*

Clustering – decreases complexity



Integration



Conclusion and future work

- Quality of deployment architectures
- Techniques/tools for improving availability

On-going/future work:

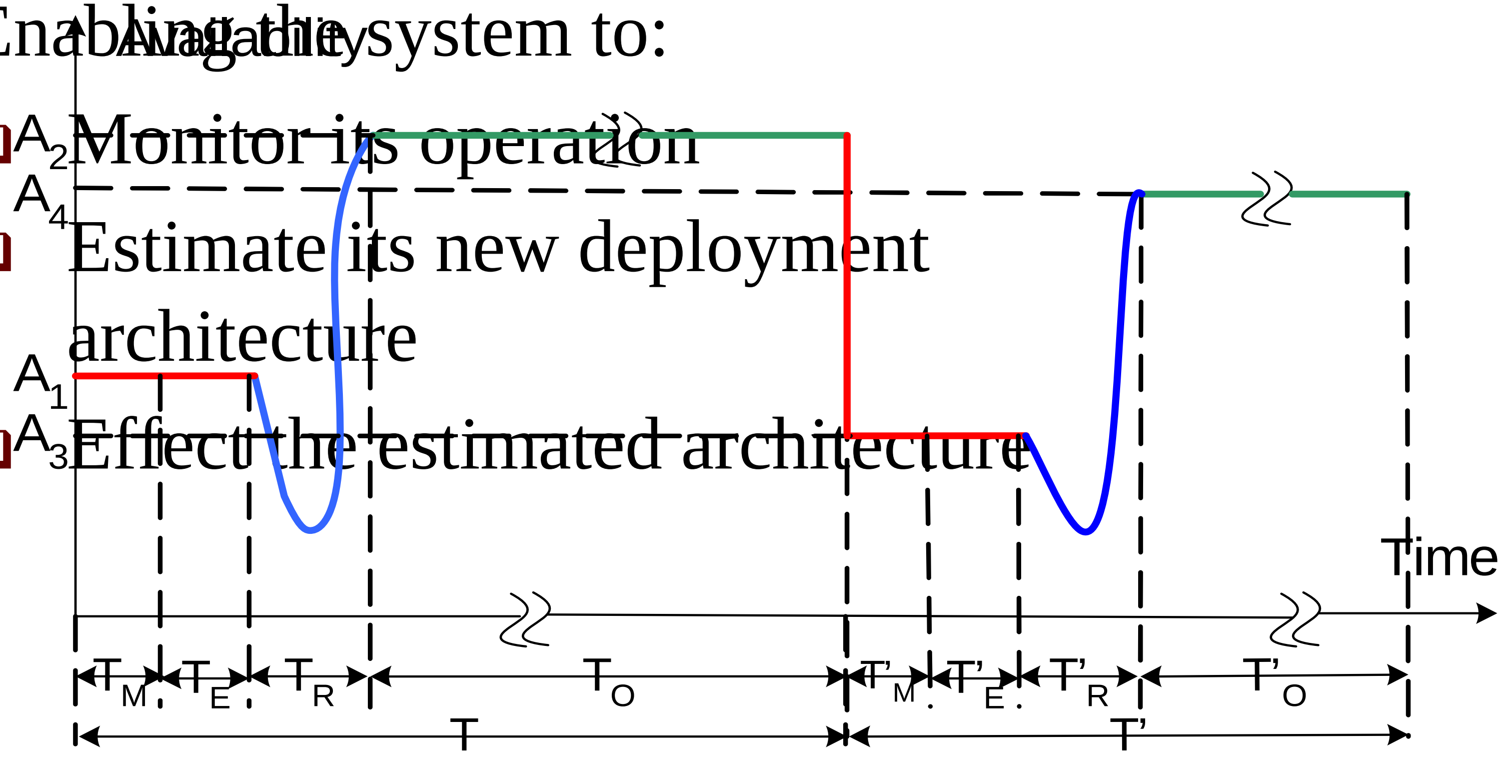
- Modeling other system properties
- Integrating DeSi with other platforms
- Decentralization and trust

Questions?

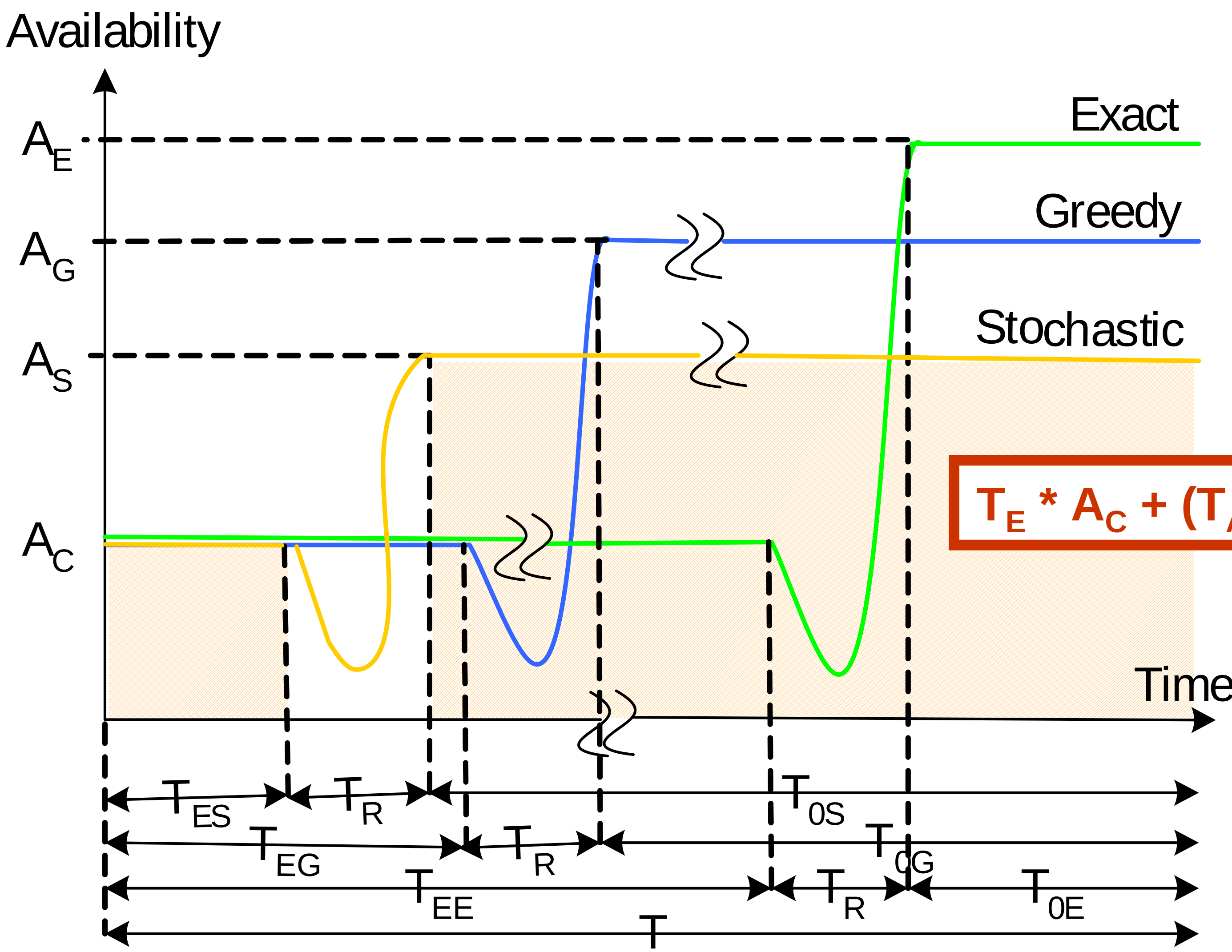
Approach - overview

Enabling the system to:

- Monitor its operation
- Estimate its new deployment architecture
- Effect the estimated architecture

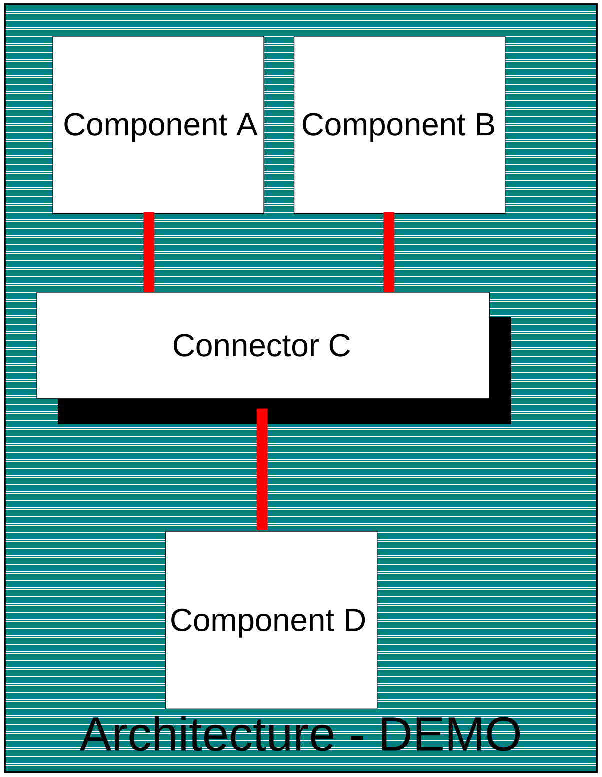


Automatic algorithm selection



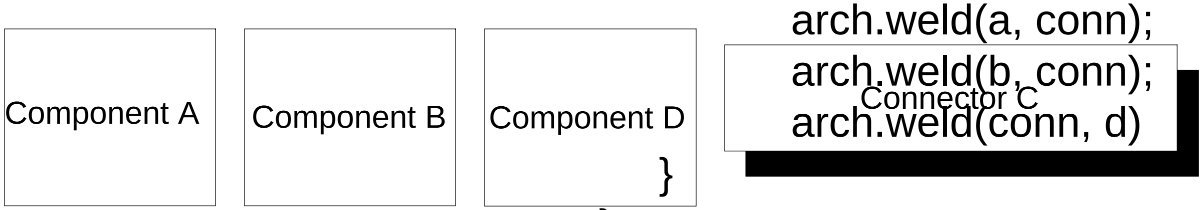
$$T_E * A_C + (T_{AVG} - T_E) * A_{EXP}$$

Using Prism-MW

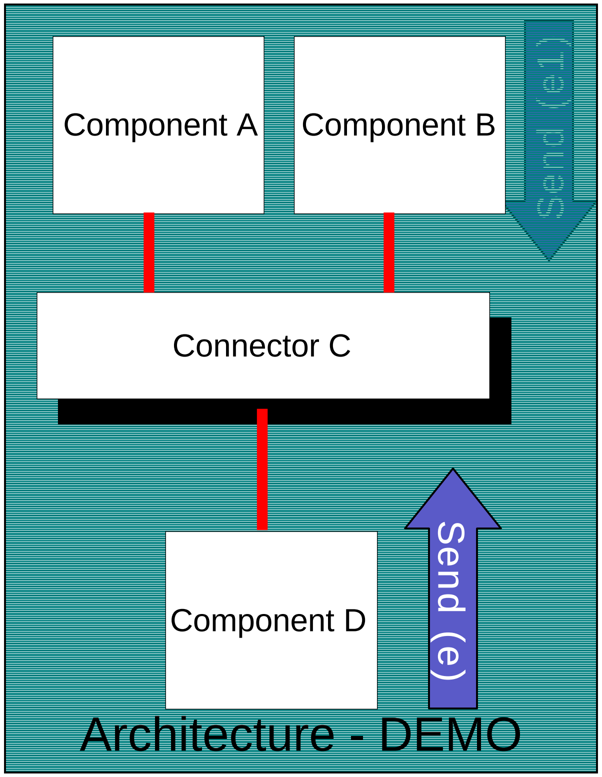



```
class DemoArch {
    static public void main(String argv[]) {
        Architecture arch = new Architecture ("DEMO ");
        // create components
        ComponentA a = new ComponentA ("A");
        ComponentB b = new ComponentB ("B");
        ComponentD d = new ComponentD (“D”);
        // create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



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);
        }...
    }
```