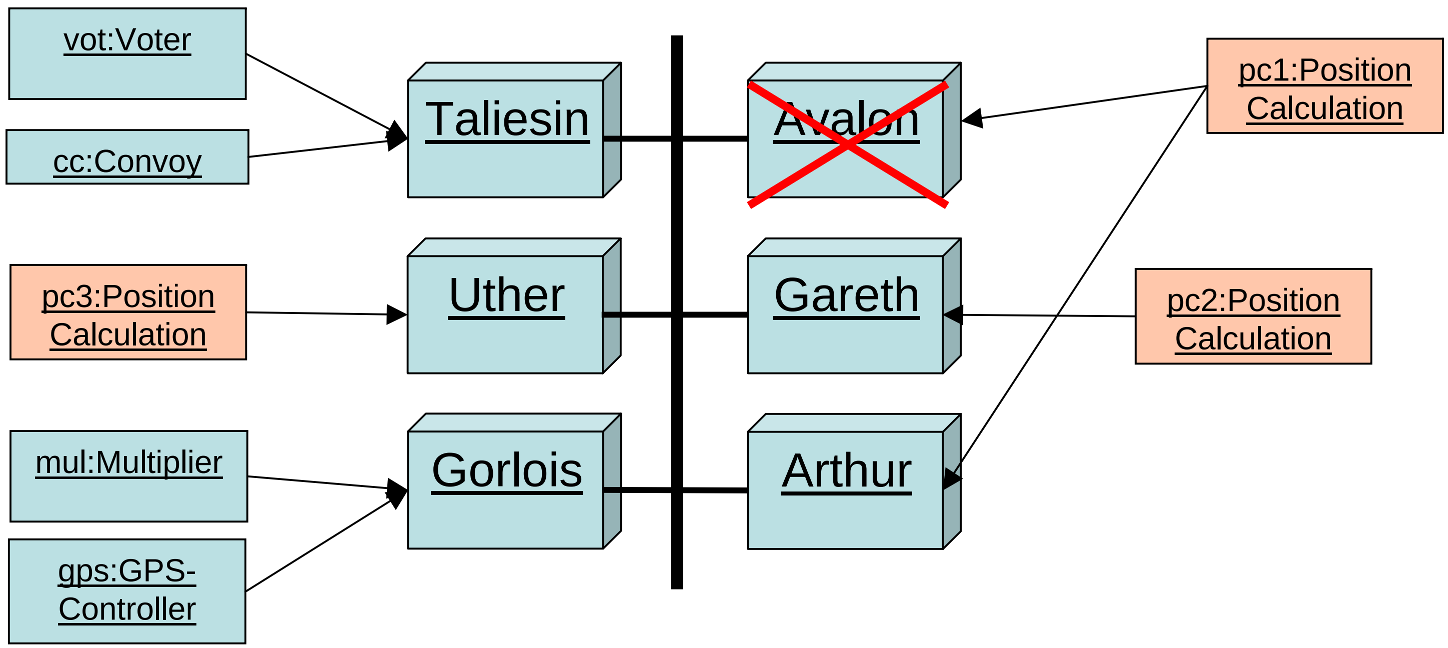
Computing Optimal Self-Repair Actions: Damage Minimization versus Repair Time

Matthias Tichy, Holger Giese, Daniela Schilling, Wladimir Pauls

Motivation

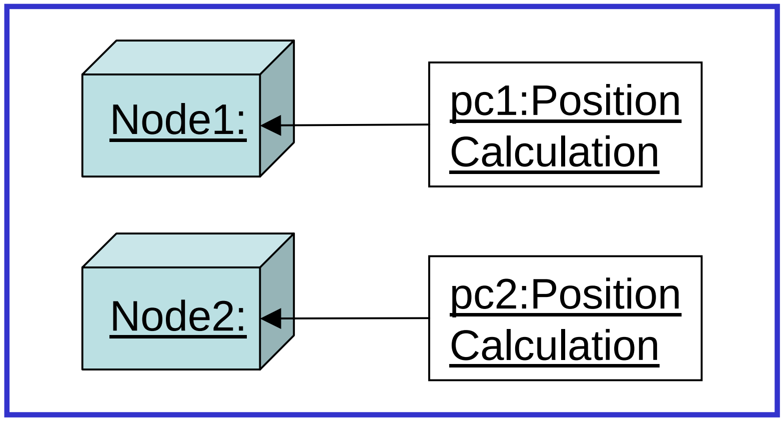
Motivation

- Redundant implementations of important software components



- Required: reconfiguration
- Given: automatism to detect failed components
- Self-Repair Actions: automatic calculation of redeployment for failed components

Initial Deployment



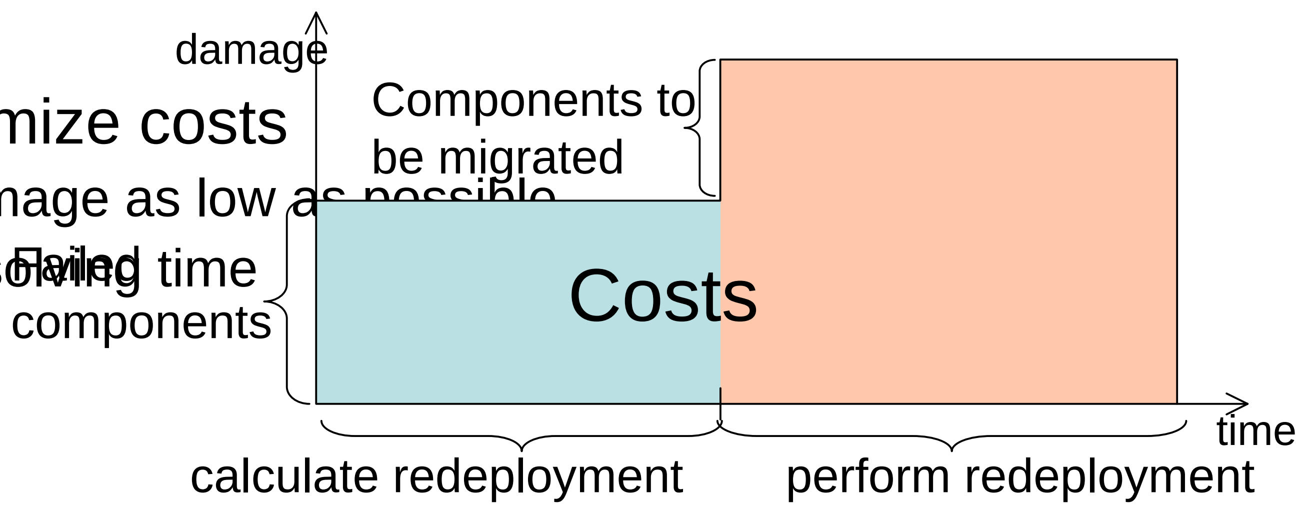
pc1.mem=2.0Mb

- Map deployment constraints given as extended UML Deployment Diagrams to inequalities over boolean and integer variables
- Use constraint solver to calculate initial deployment

WOSS/FSE 2004:
Matthias Tichy, Daniela Schilling, Holger Giese:
Design of Self-Managing Dependable Systems with UML and Fault Tolerance Patterns

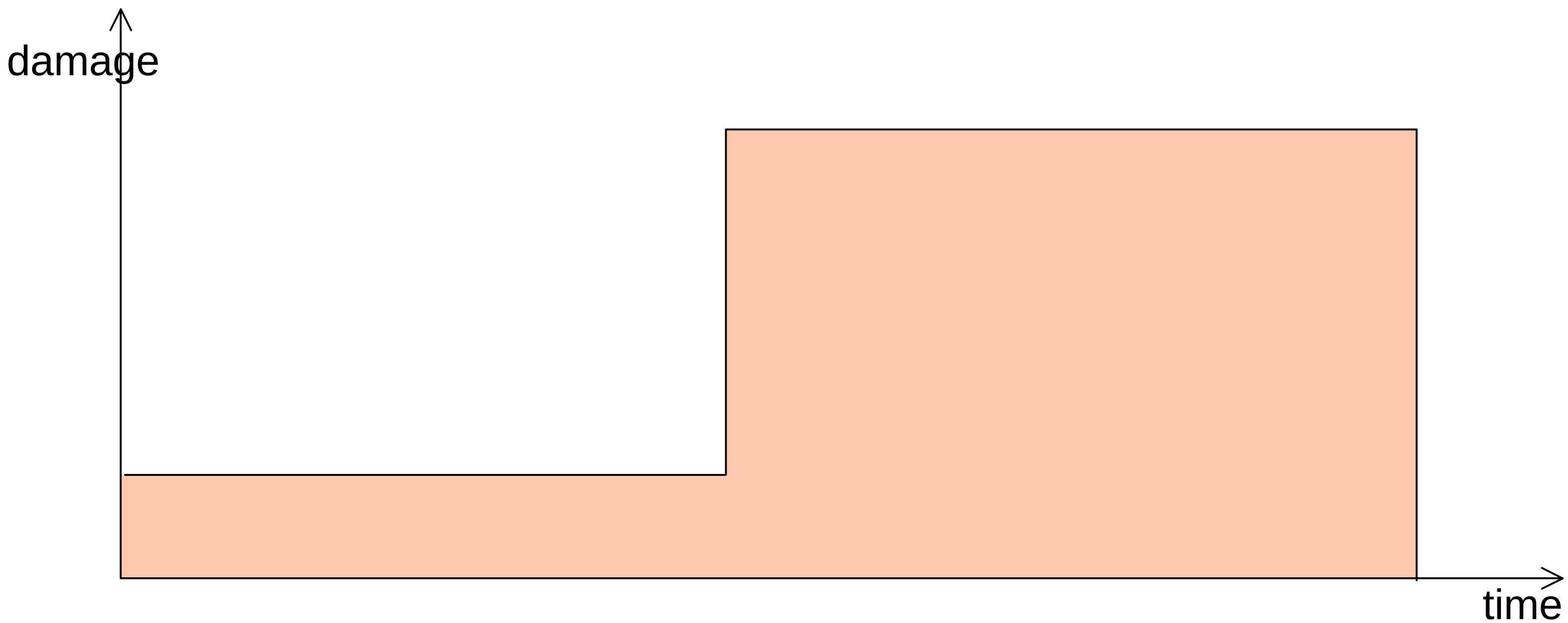
Online Redeployment

- Node crash failure ⇒ all components running on this node fail too
- Compute Self-Repair Action
  - -> Find suitable nodes to redeploy failed components
- How to find suitable nodes?
- What to do if there is no suitable node?
  - Redeploy further (still running) components
- Damage: negative effects of unavailable components
- Costs
- Goal: minimize costs
  - Keep damage as low as possible
  - Reduce solving time



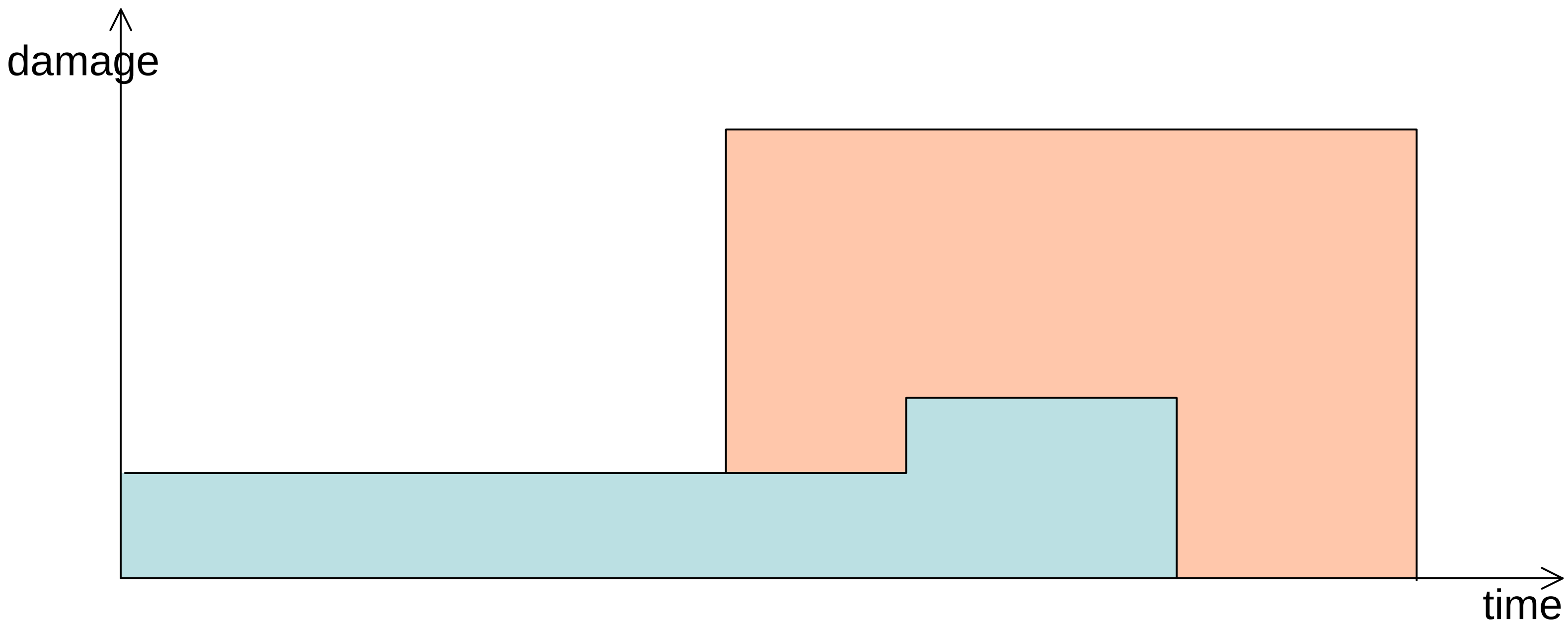
Online Redeployment - 1.Solution -

- Remove crashed nodes from constraint system
- Solve complete constraint system again



Online Redeployment - 2.Solution -

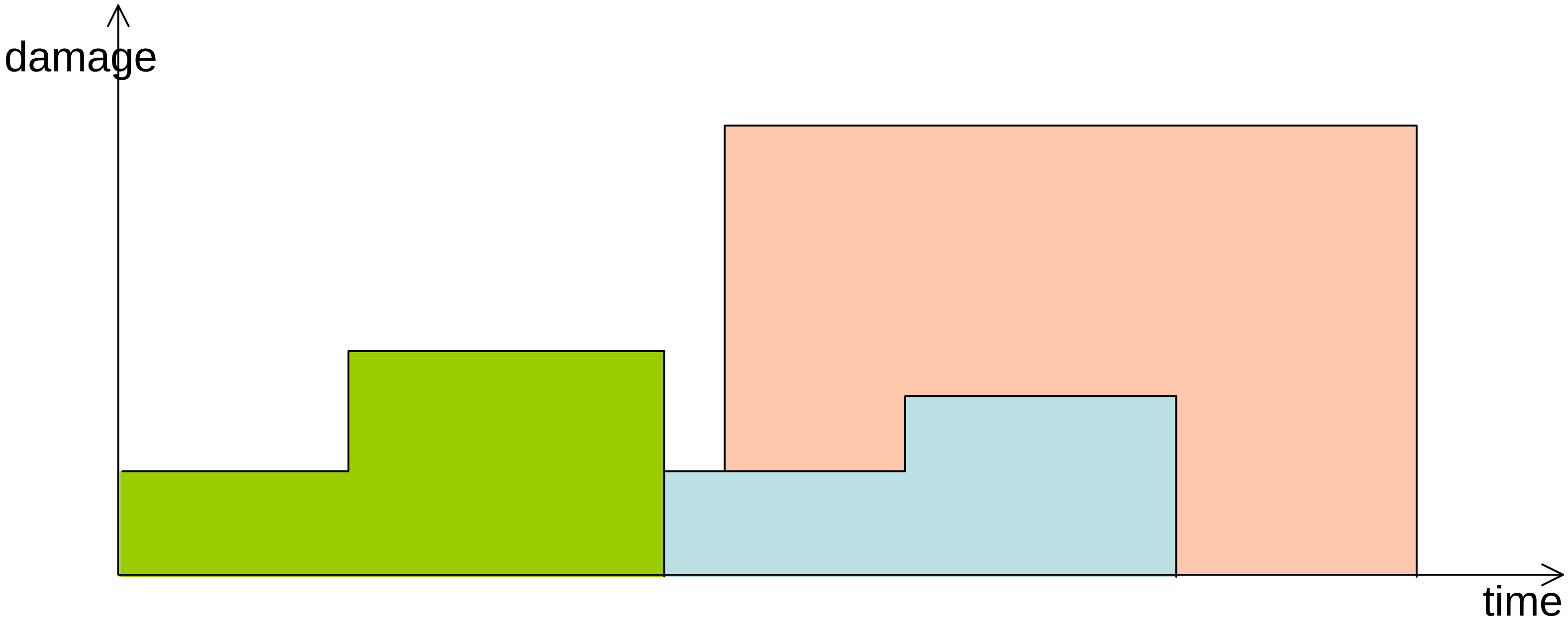
- Remove crashed nodes from constraint system
- Add objective function (minimize damage caused by migration of running componets) to the constraint system
- Solve complete system again



Online Redeployment - Our Approach -

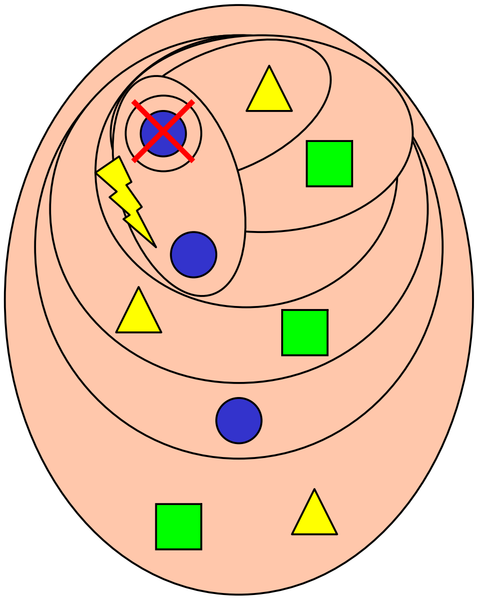
- Remove crashed nodes from constraint system
- Add objective function (minimize damage) to the constraint system
- Try to solve constraint systems for failed components only
- Until a solution is found: extend set of components that have to be redeployed/migrated
- Use Constraint solver
- Heuristic approach

Online Redeployment - Our Approach -



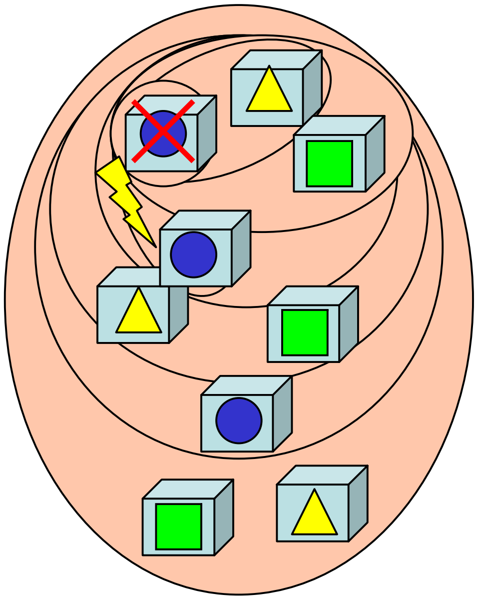
Choosing Components for Redeployment

- Example: 3 redundant copies of important components
- Algorithm:
  - Try to redeploy failed component
  - Until redeployment is possible:
    1. Choose components which are no redundant copies of failed components
    2. Choose components where only one of three redundant copies already failed
    3. Choose arbitrary components



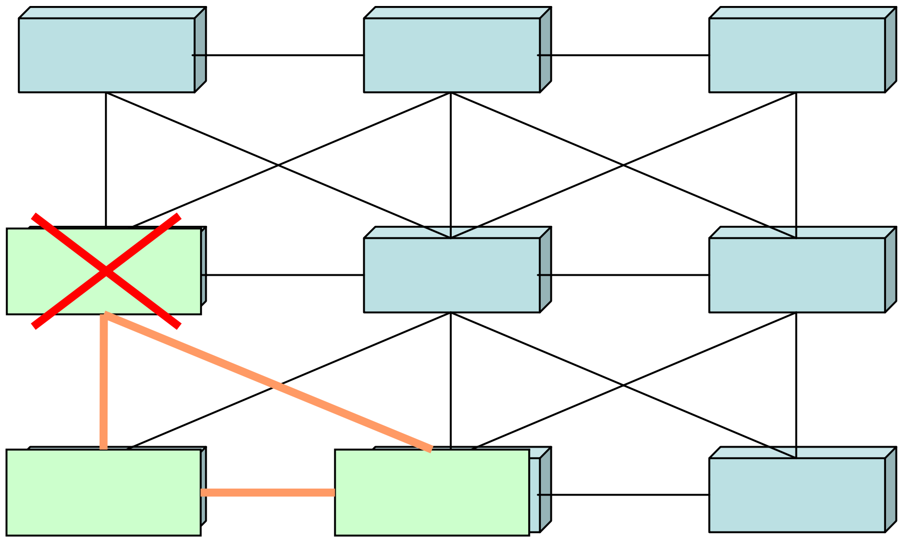
Choosing Components for Redeployment

- Example: 3 redundant copies of important components
- Algorithm:
  - Try to redeploy failed component
  - Until redeployment is possible:
    1. Choose components which are no redundant copies of failed components
    2. Choose components where only one of three redundant copies already failed
    3. Choose arbitrary components



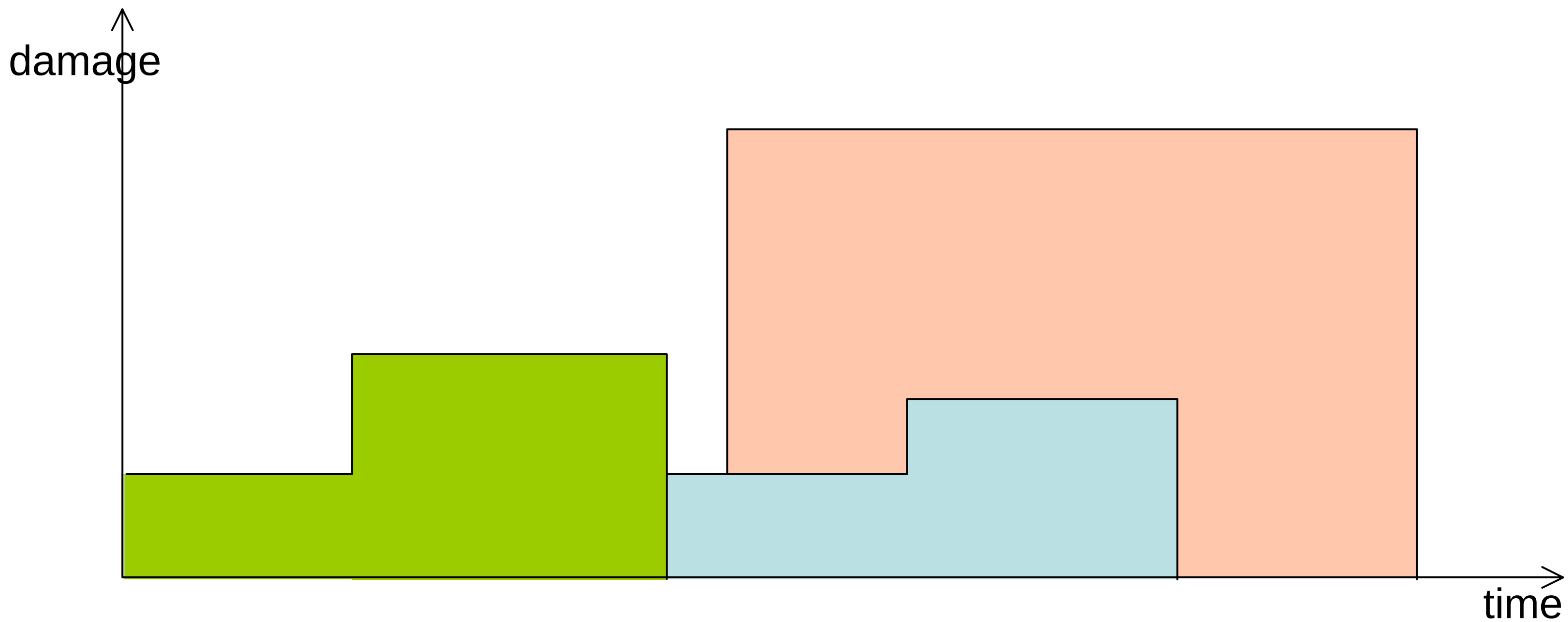
Experiment

- Scenario:
  - 36 nodes with 114 links
  - 72 components with 99 connectors
  - 5 node-specific (CPU, OS, Memory, Utilization, HDD) and 2 link-specific (Bandwidth, Loss) deployment restrictions
  - set of deployment constraints on components and connectors
- Experiment:
  - Randomly selected a node and let it fail



Experimental Results

Test Nr.	1. Solution		2. Solution		Our Algorithm	
	Time (ms)	Damage	Time (ms)	Damage	Time (ms)	Damage
1	13630	773	> 1h	N/A	50	7
2	14890	97	56060	29	30	30
3	13790	4	14920	1	10	5
4	13660	34	16430	31	50	34



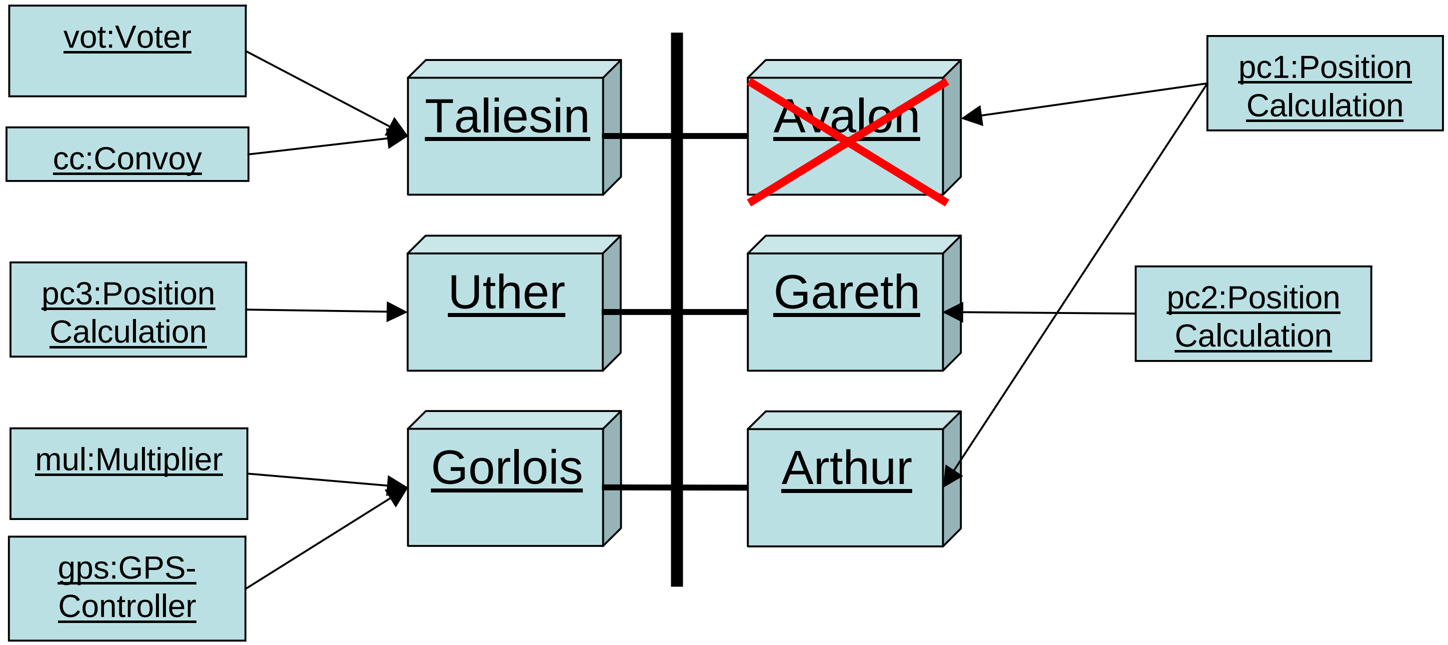
Conclusion & Future Work

- Algorithm to calculate optimal self-repair actions
- Deployment constraints solved by standard constraint solver
- Experiment showed that algorithm is nearly optimal in damage minimization and time consumption
- Not presented: pre-solving step

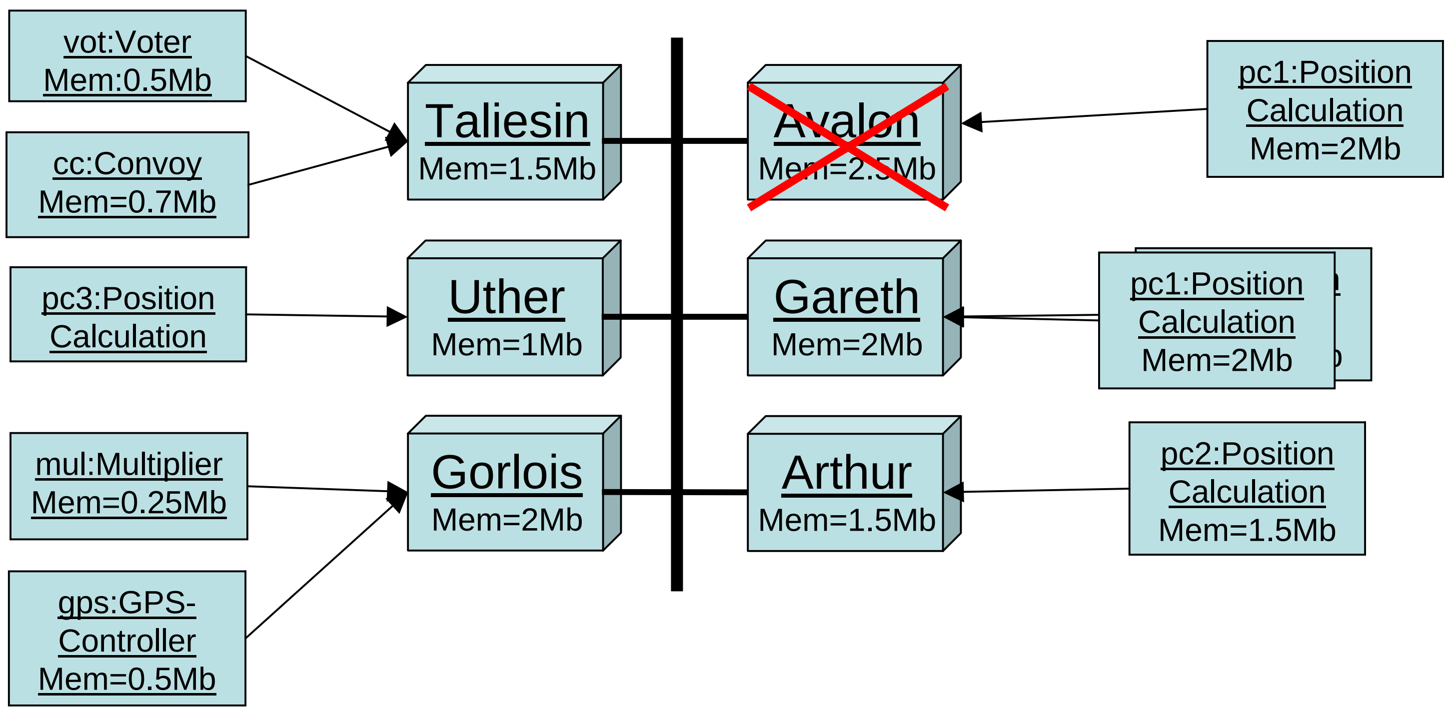
- Communication and monitoring framework
- Describe repair rules by graph transformation systems

Appendix

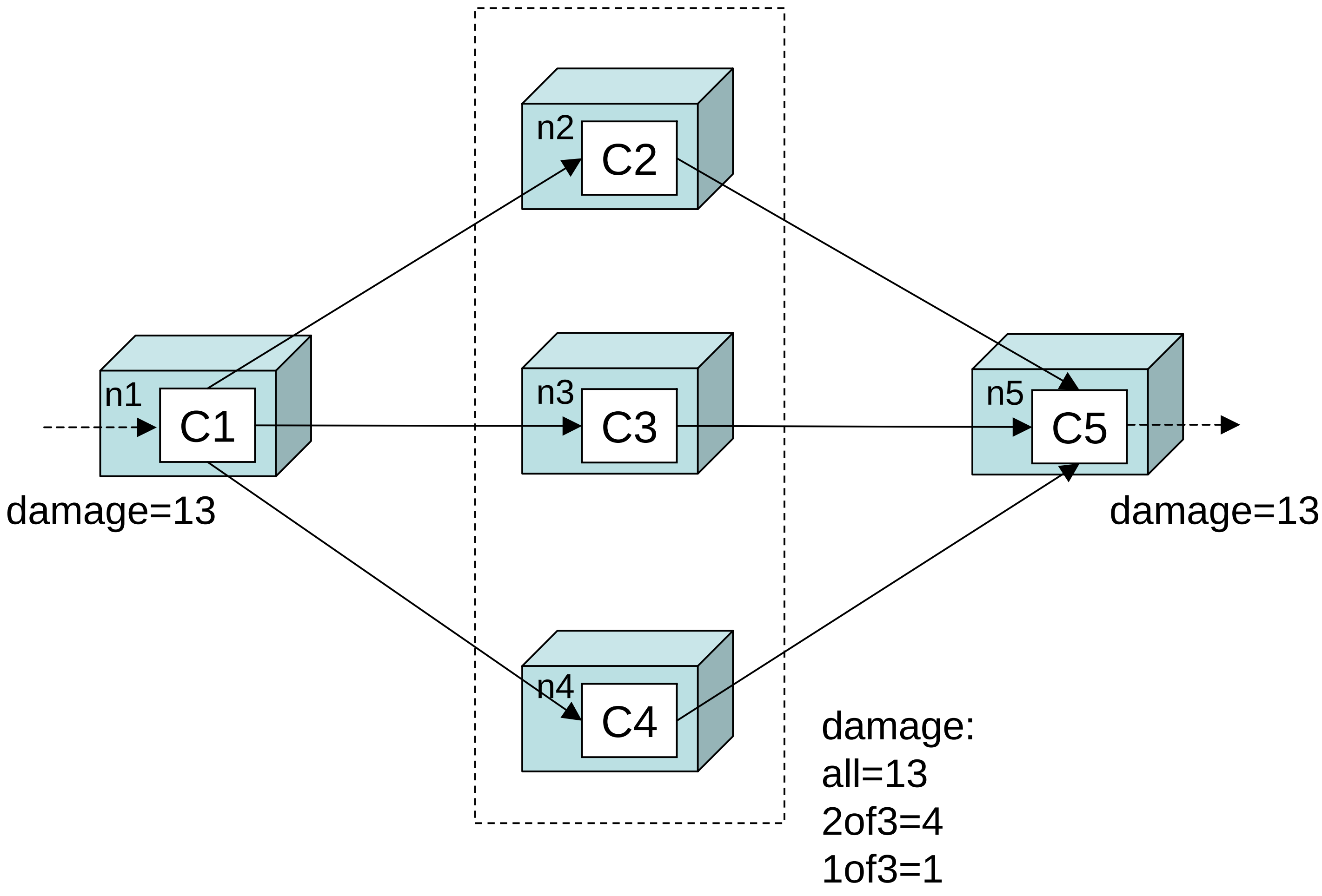
Simple Redeployment



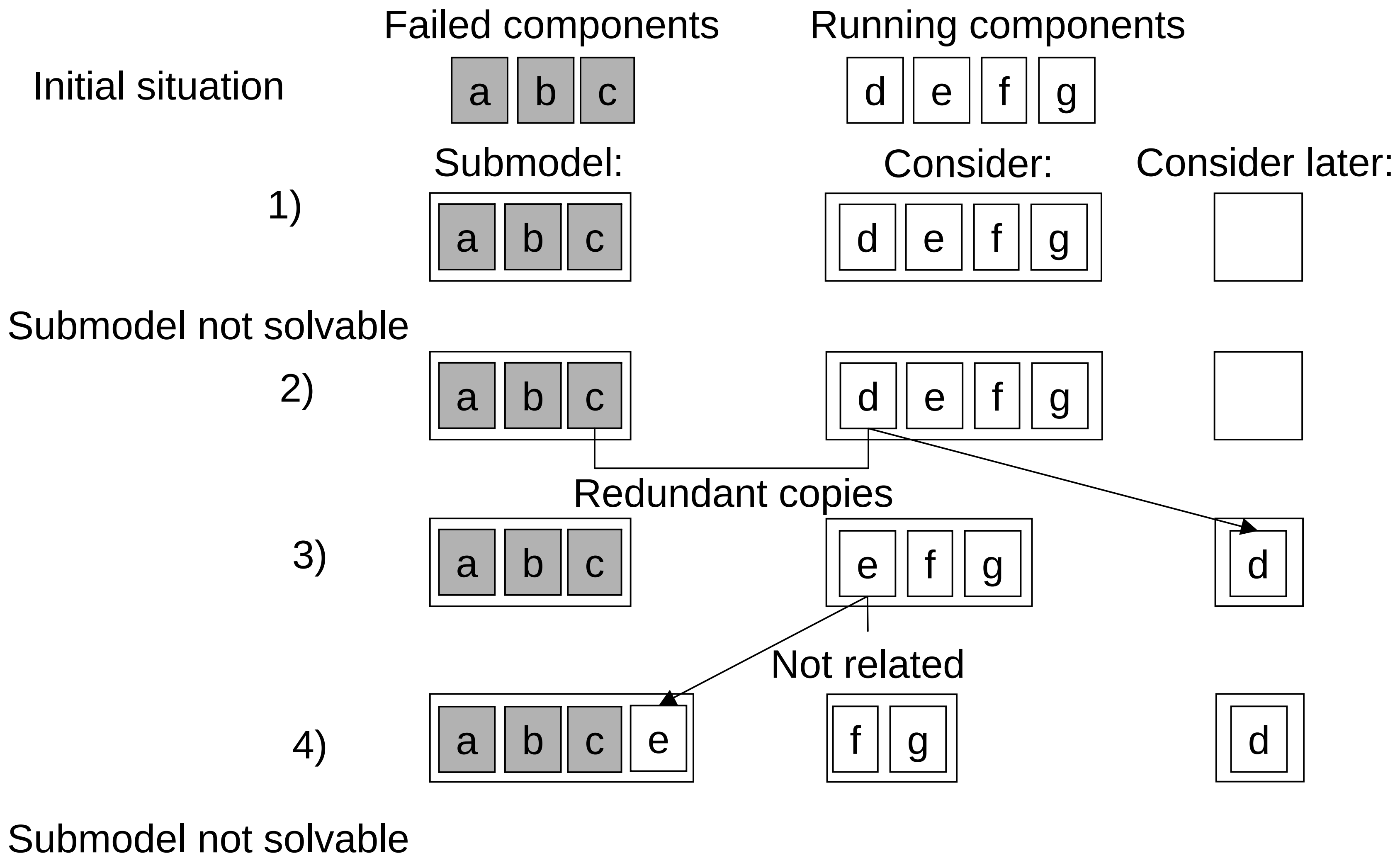
Example


vot:Voter
Mem:0.5Mb
cc:Convoy
Mem=0.7Mb
pc3:Position
Calculation
mul:Multiplier
Mem=0.25Mb
gps:GPS-
Controller
Mem=0.5Mb
Taliesin
Mem=1.5Mb
Uther
Mem=1Mb
Gorlois
Mem=2Mb
Avalon
Mem=2.5Mb
Gareth
Mem=2Mb
Arthur
Mem=1.5Mb
pc1:Position
Calculation
Mem=2Mb
pc1:Position
Calculation
Mem=2Mb
pc2:Position
Calculation
Mem=1.5Mb

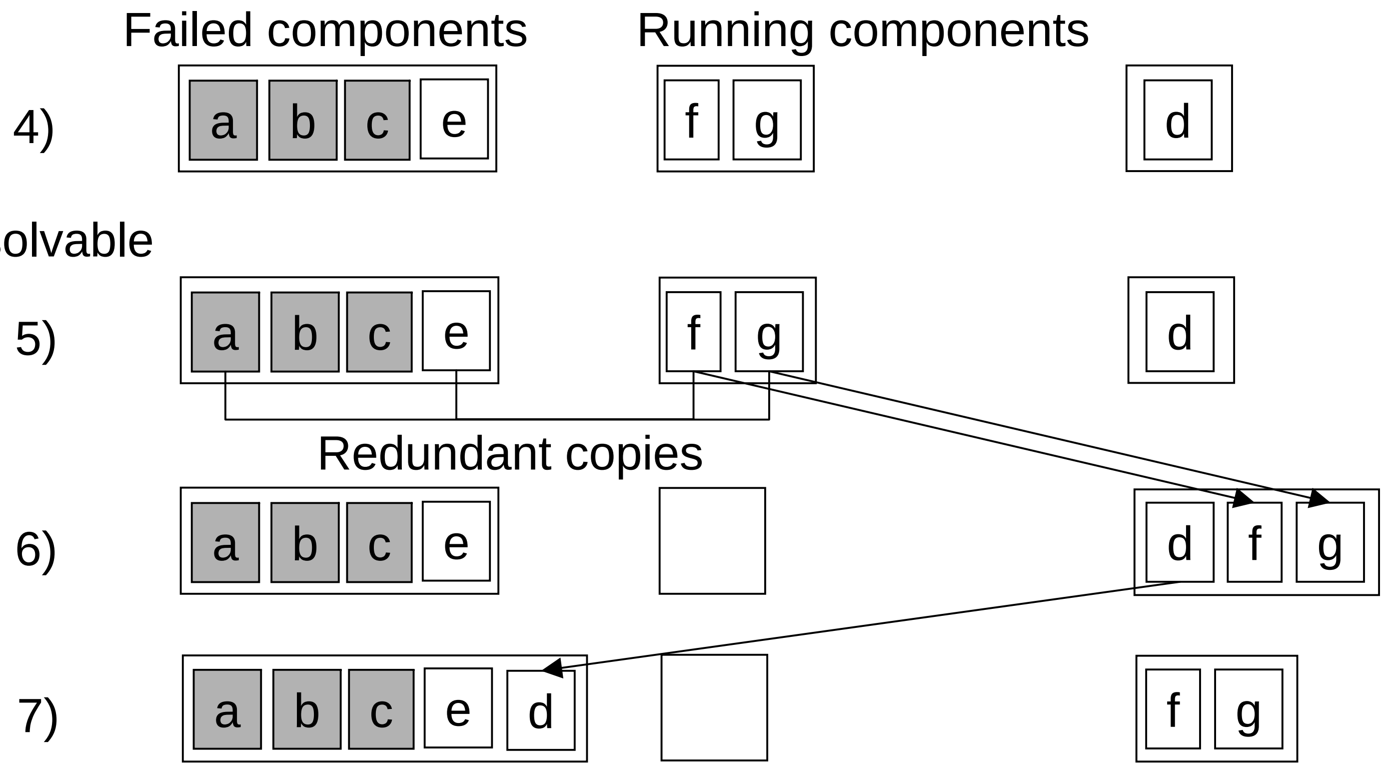
Damage Calculation



Submodel Expansion



Submodel Expansion(2)



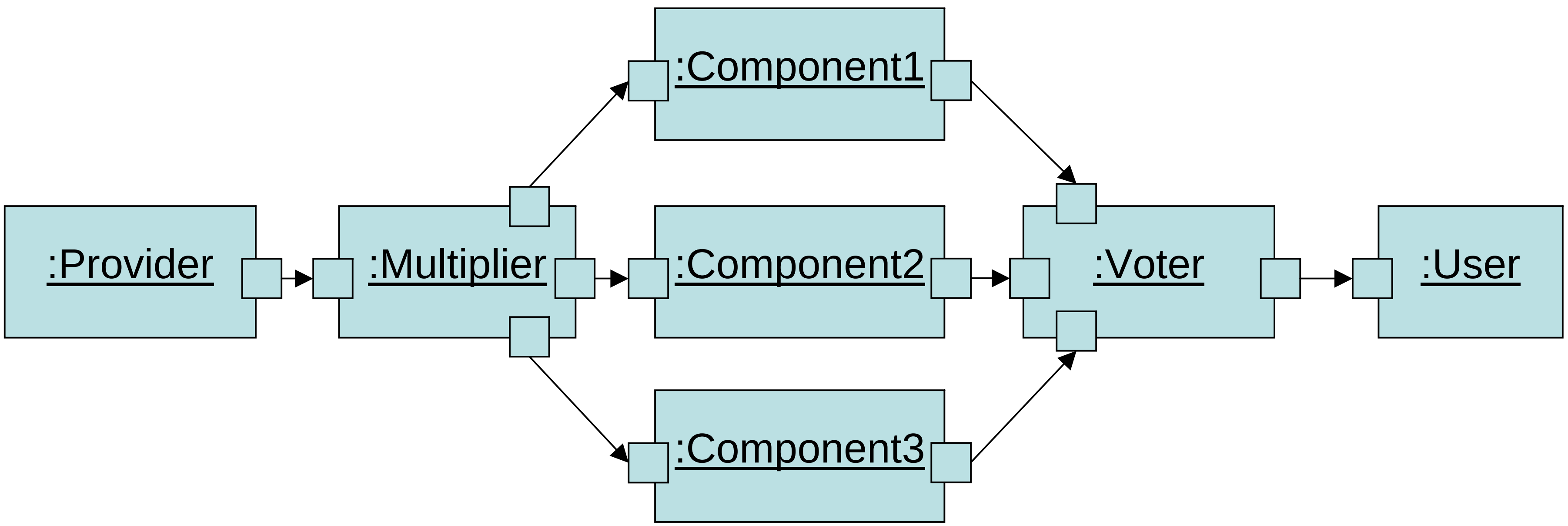
Submodel not solvable

Submodel solvable

Pre-Solving

Foundations (TMR)

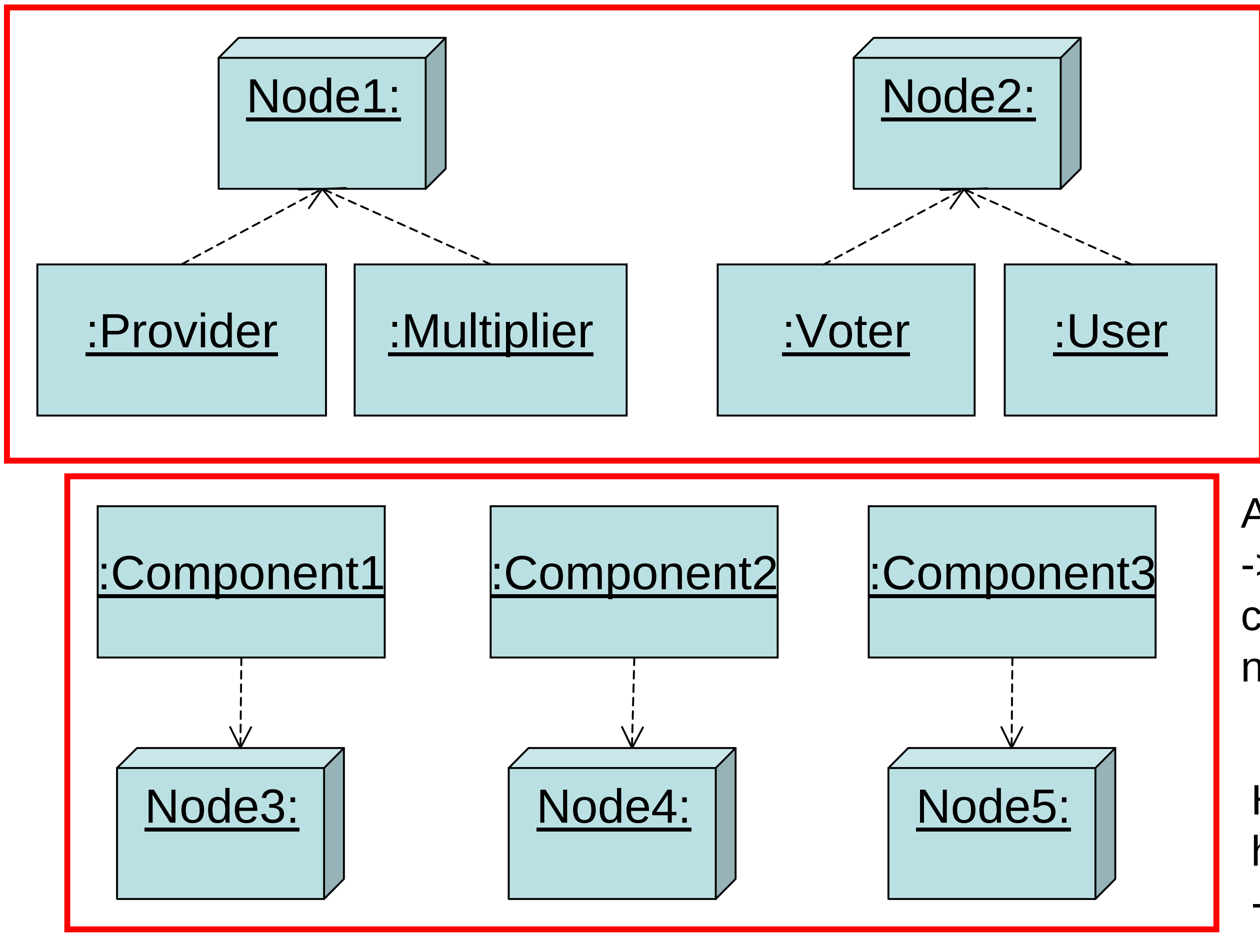
- Use fault tolerance techniques to ensure dependability
- Triple Modular Redundancy (TMR)



University of Paderborn
Software Engineering Group
Prof. Dr. Wilhelm Schäfer

Foundations (TMR)

- Deployment constraints for TMR



Avoid single-point-of-failure of voter / multiplier
-> Deploy voter and user to same node
(if the user fails, the failure of the voter is no problem)

Avoid crash failures
-> Deploy redundant components to distinct nodes

Heterogeneous hardware platform
-> require different CPU

{ Node3.CPU ⑤ Node4.CPU ❖ Node4.CPU ⑤ Node5.CPU ❖ Node3.CPU ⑤ Node 5.CPU }

www.FUJABA.de
Tool Suite

Online Redeployment - Our Solution -

- Compute Self-Repair Action
  - -> Find suitable nodes to redeploy failed components
- How to find suitable nodes?
- What to do if there is no suitable node?
  - 2) Redeploy further (still running) components
  - Goal: reduce costs
    - Redeployment should not decrease dependability (reduce damage)
    - Reduce solving time


BahnCam @ www.MLaP.de
2005-05-12 CEST 15:49:23