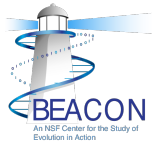
Automated Generation of Adaptive Test Plans for Self-Adaptive Systems

Erik Fredericks and Betty H. C. Cheng

May 19th, 2015



Motivation

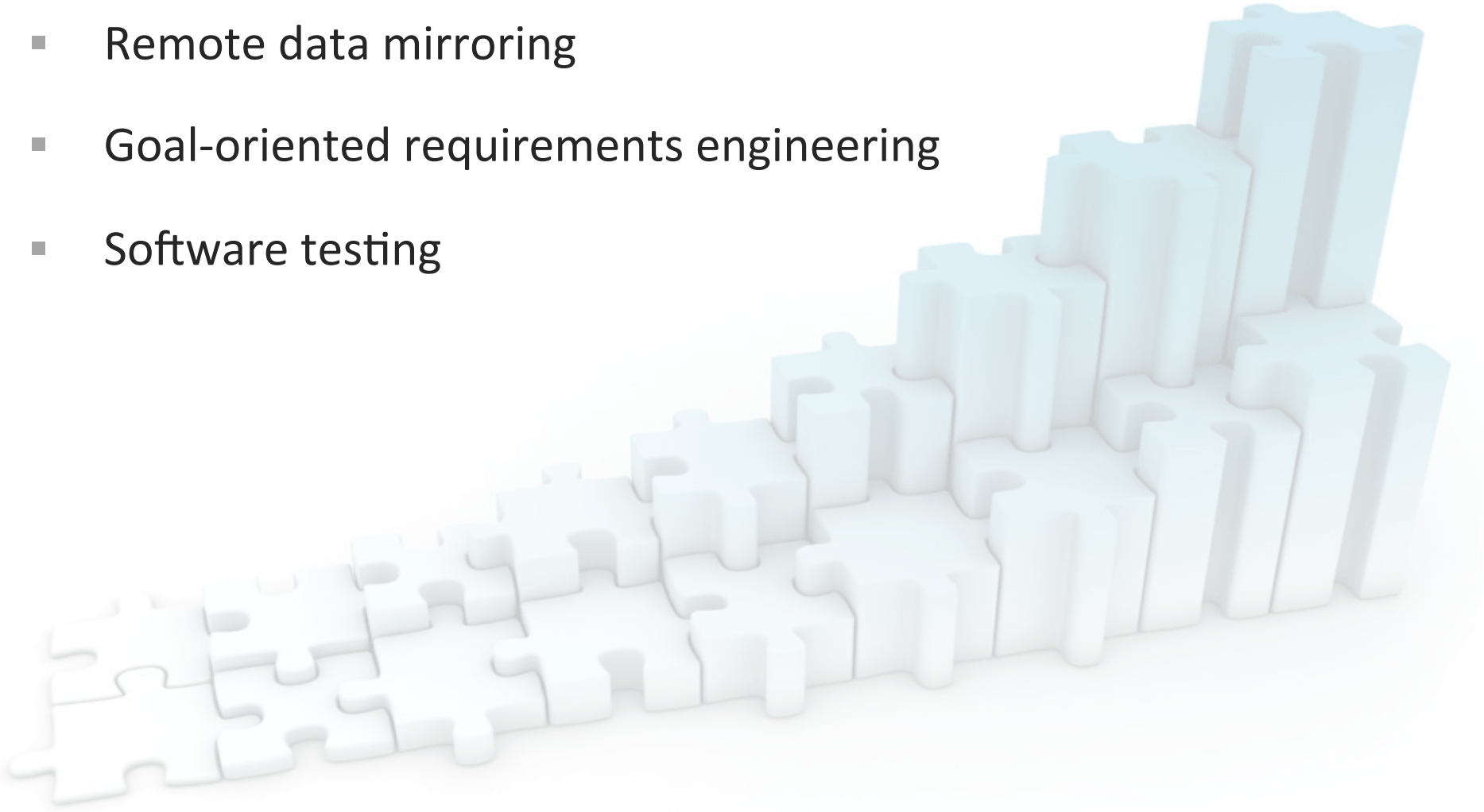
- Run-time testing provides assurance for self-adaptive systems (SAS)
- An SAS can experience uncertainty, possibly rendering test cases created at design time irrelevant
- **Proteus** ensures that test suites and test cases remain relevant throughout SAS execution

Agenda

- Background
- Proteus approach
- Case study
  - Discussion
- Impact of run-time testing
  - Discussion
- Related work

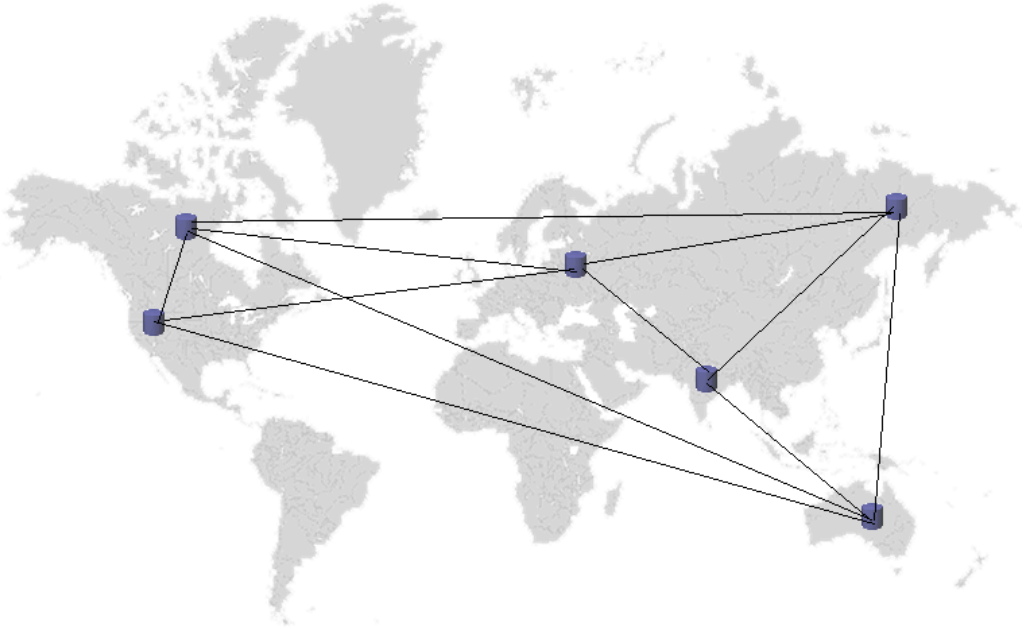
Background

- Remote data mirroring
- Goal-oriented requirements engineering
- Software testing

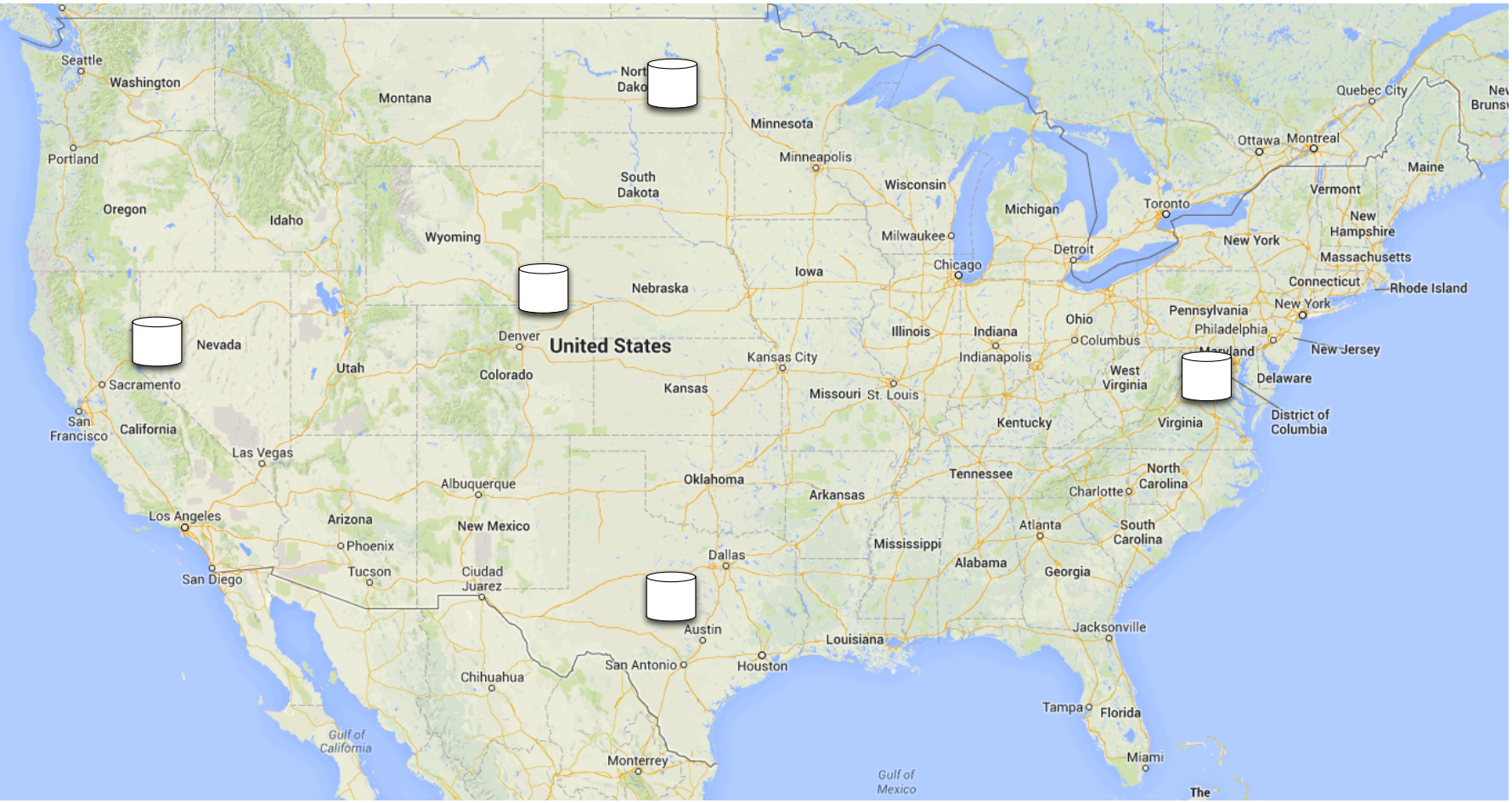


Remote Data Mirroring Application

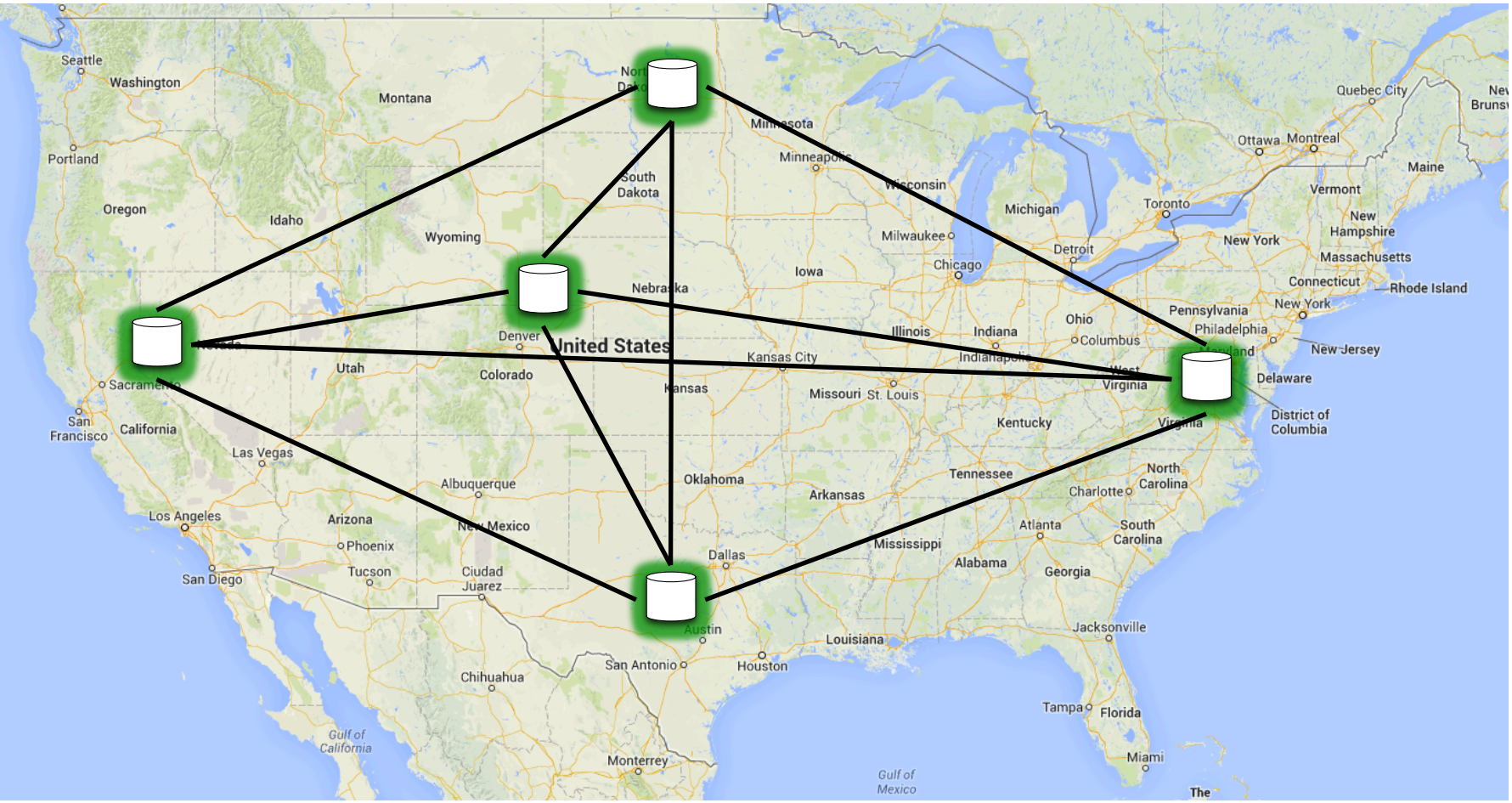
- RDM [Veitch2003,Keeton2004] provides:
  - Data protection
  - Prevents data loss and maintains availability
  - Stores data in physically remote locations
  - Represented as an SAS



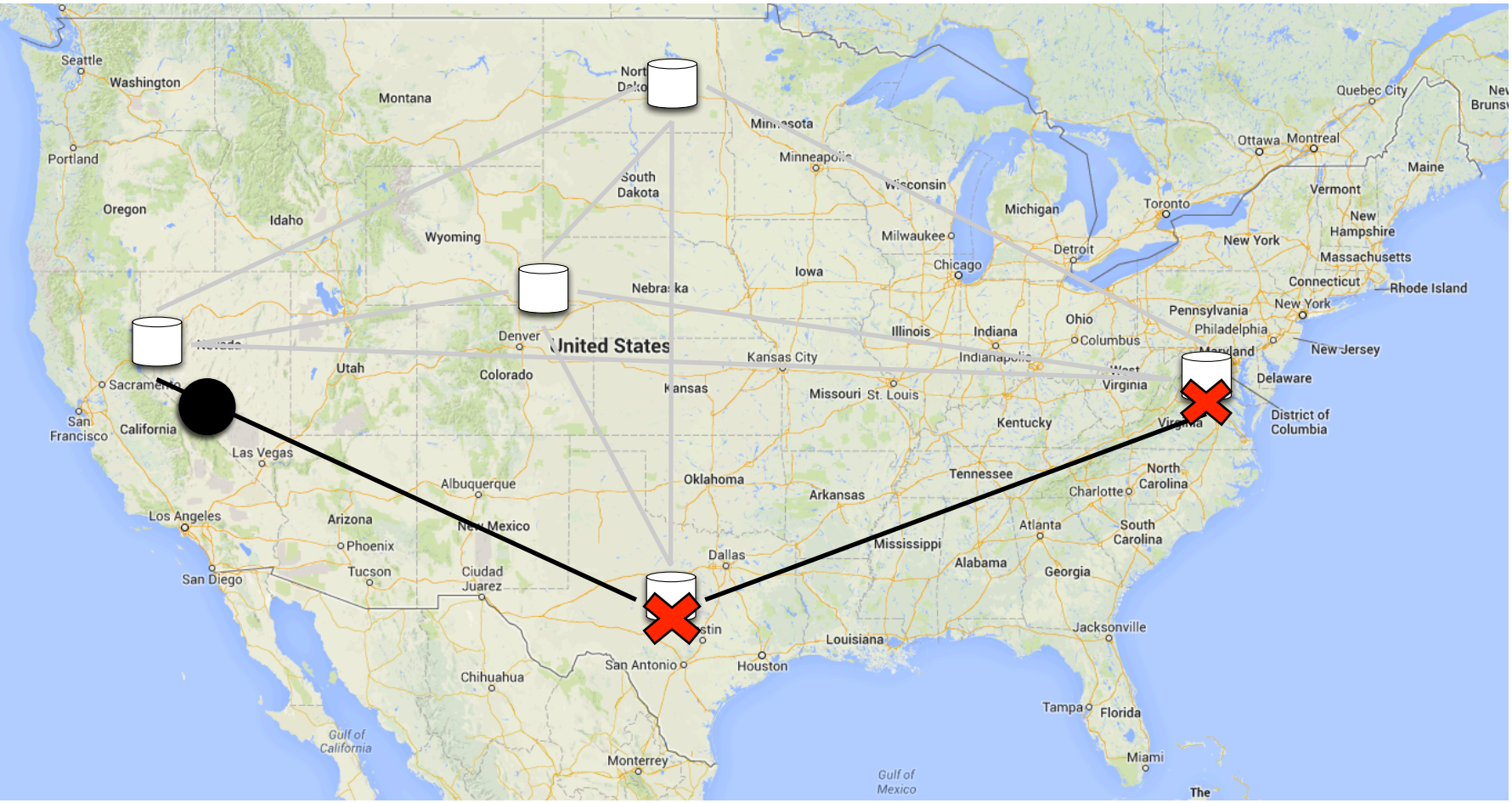
Remote Data Mirroring Application



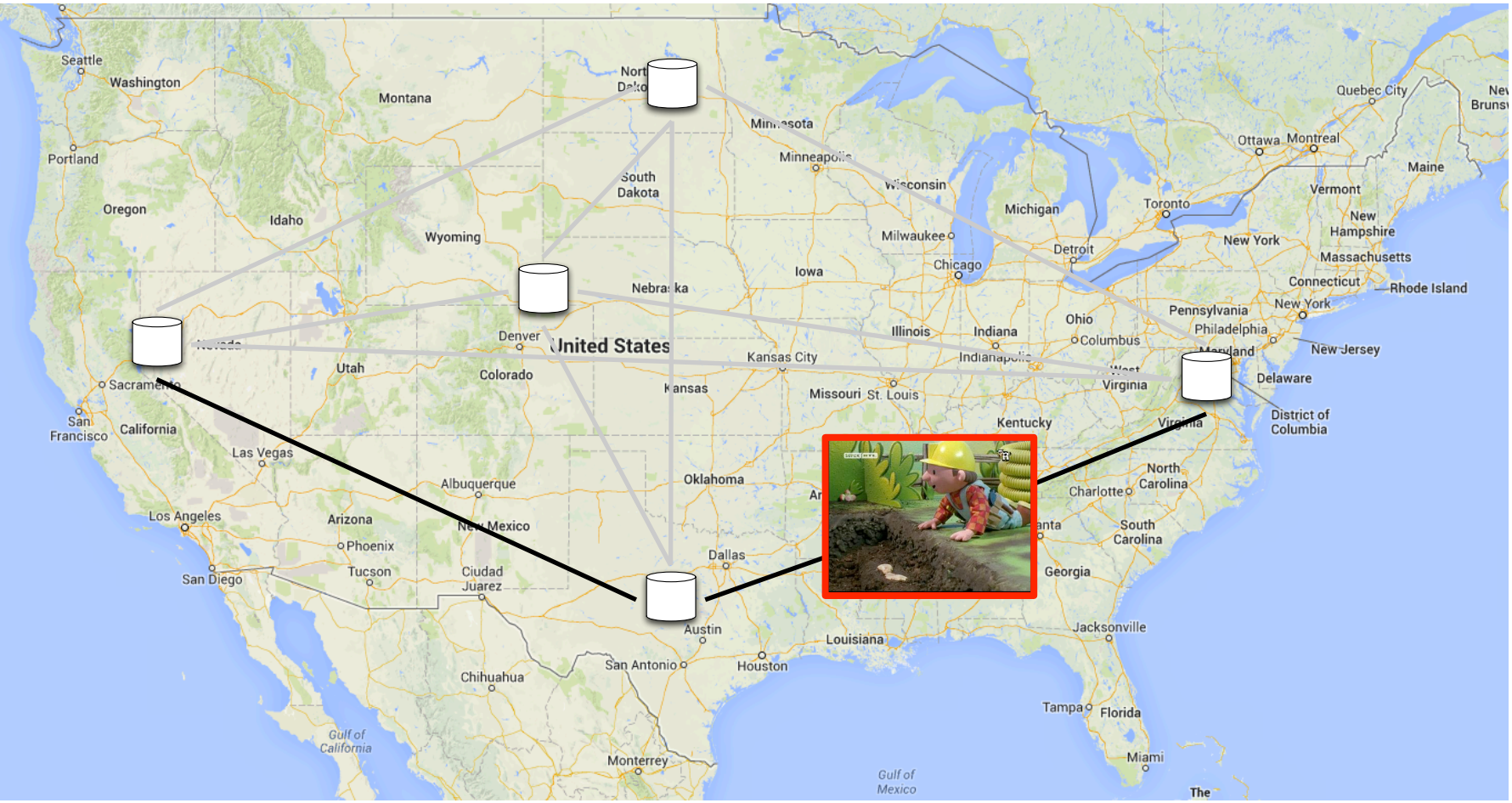
Network Connections



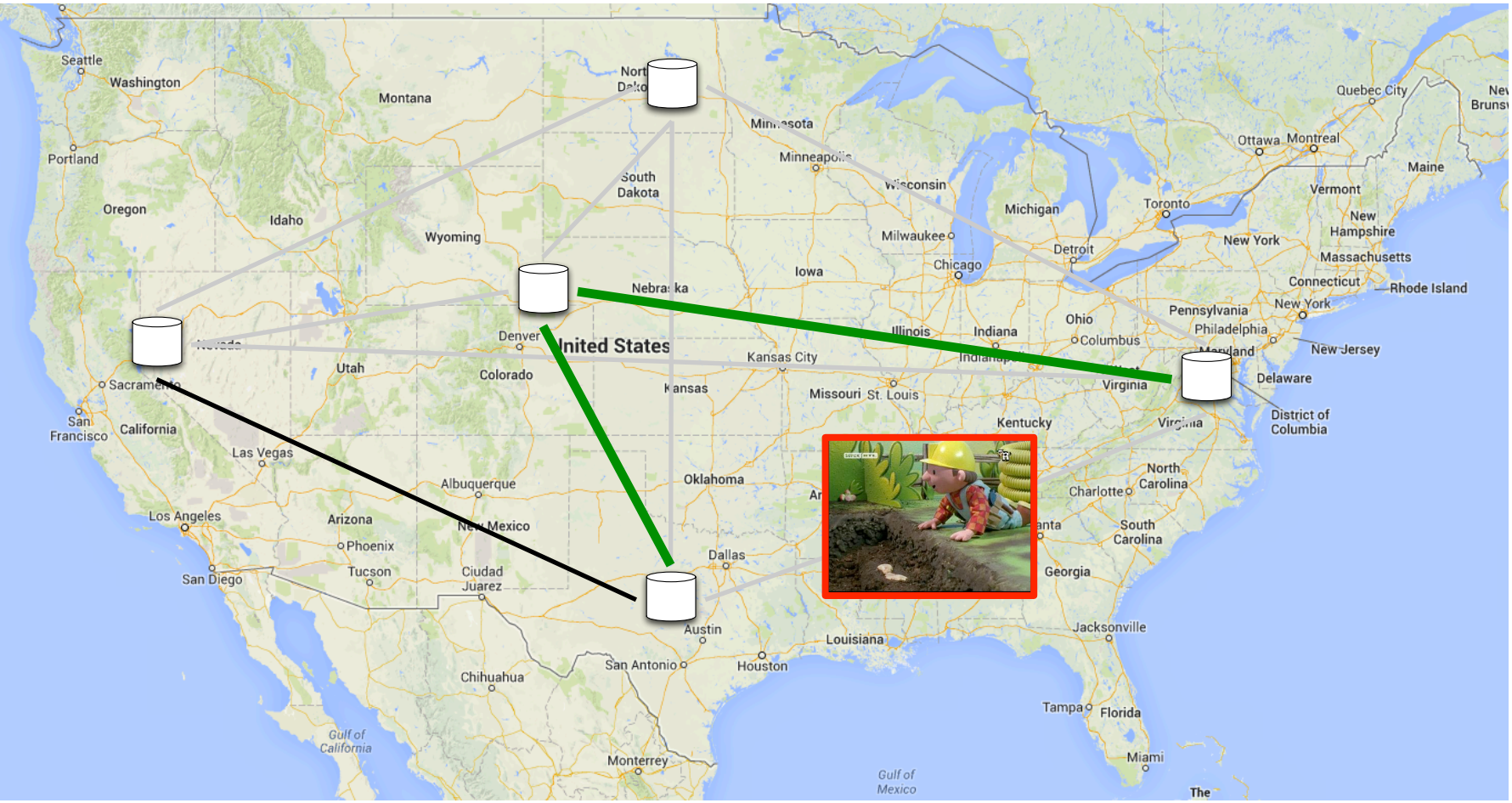
Dropped Message



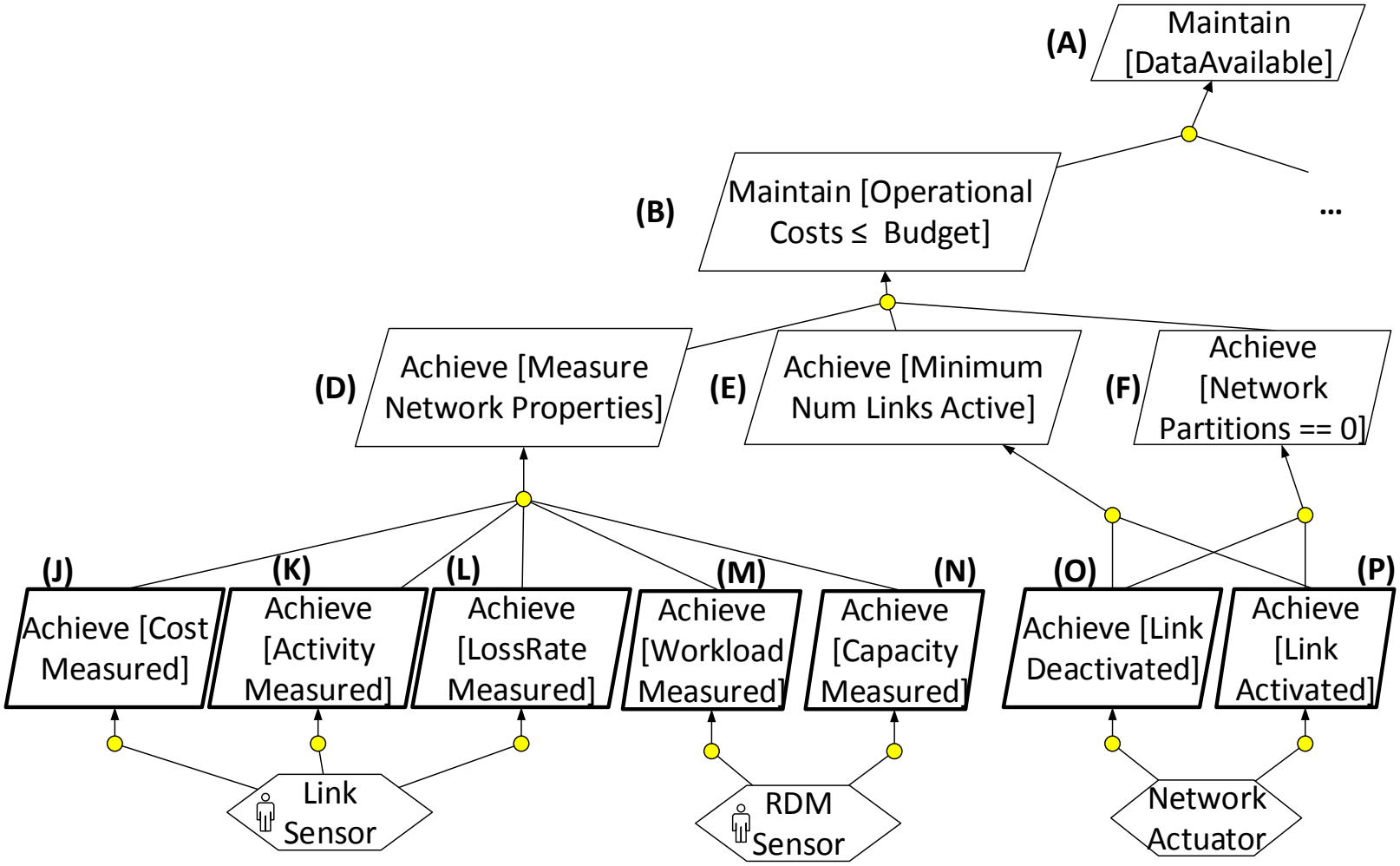
Disrupted Connection



Reconfiguration

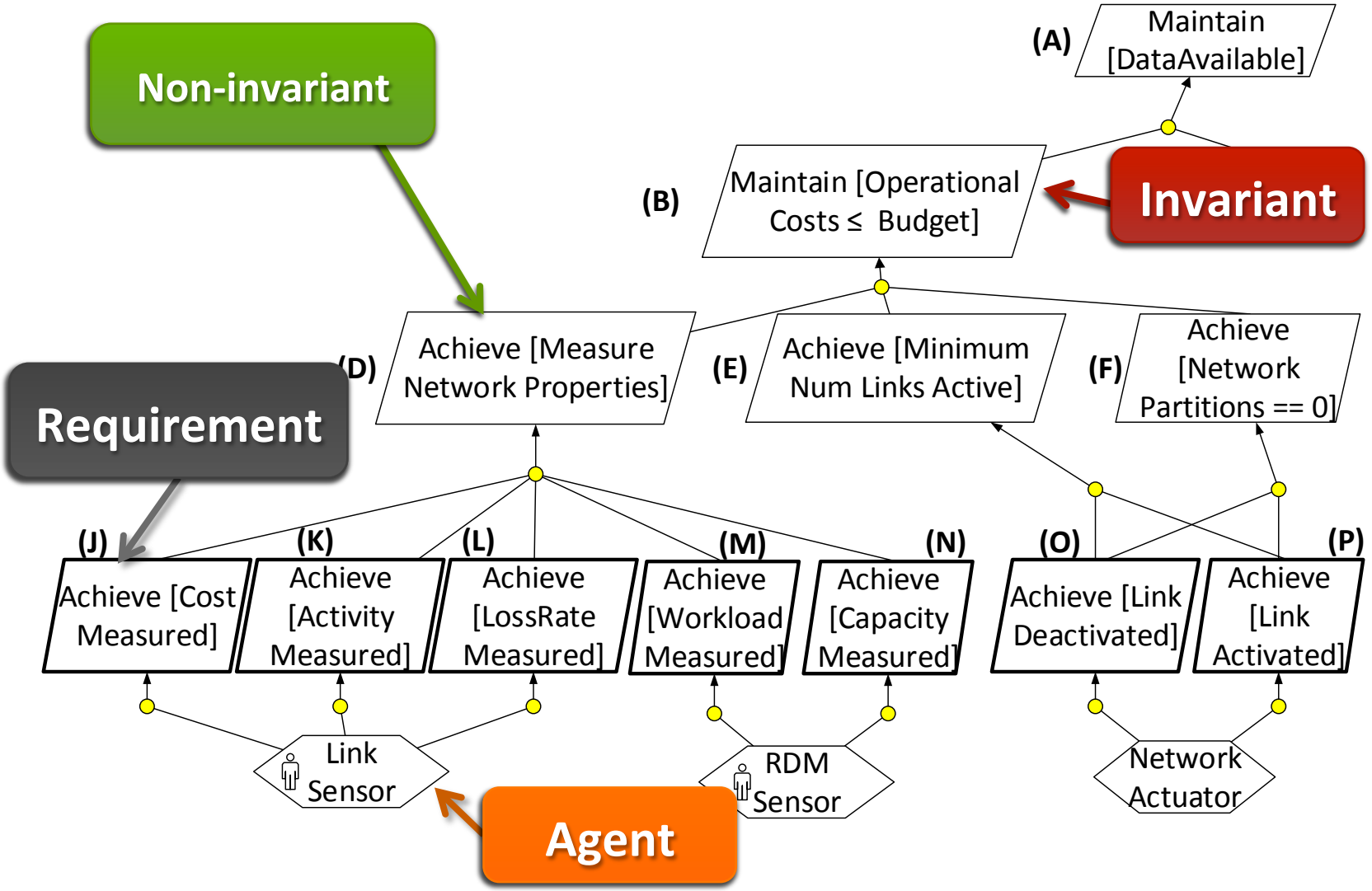


Goal-Oriented Requirements Engineering



Partial KAOS [Dardenne1993,vanLamsweerde2009] goal model of RDM

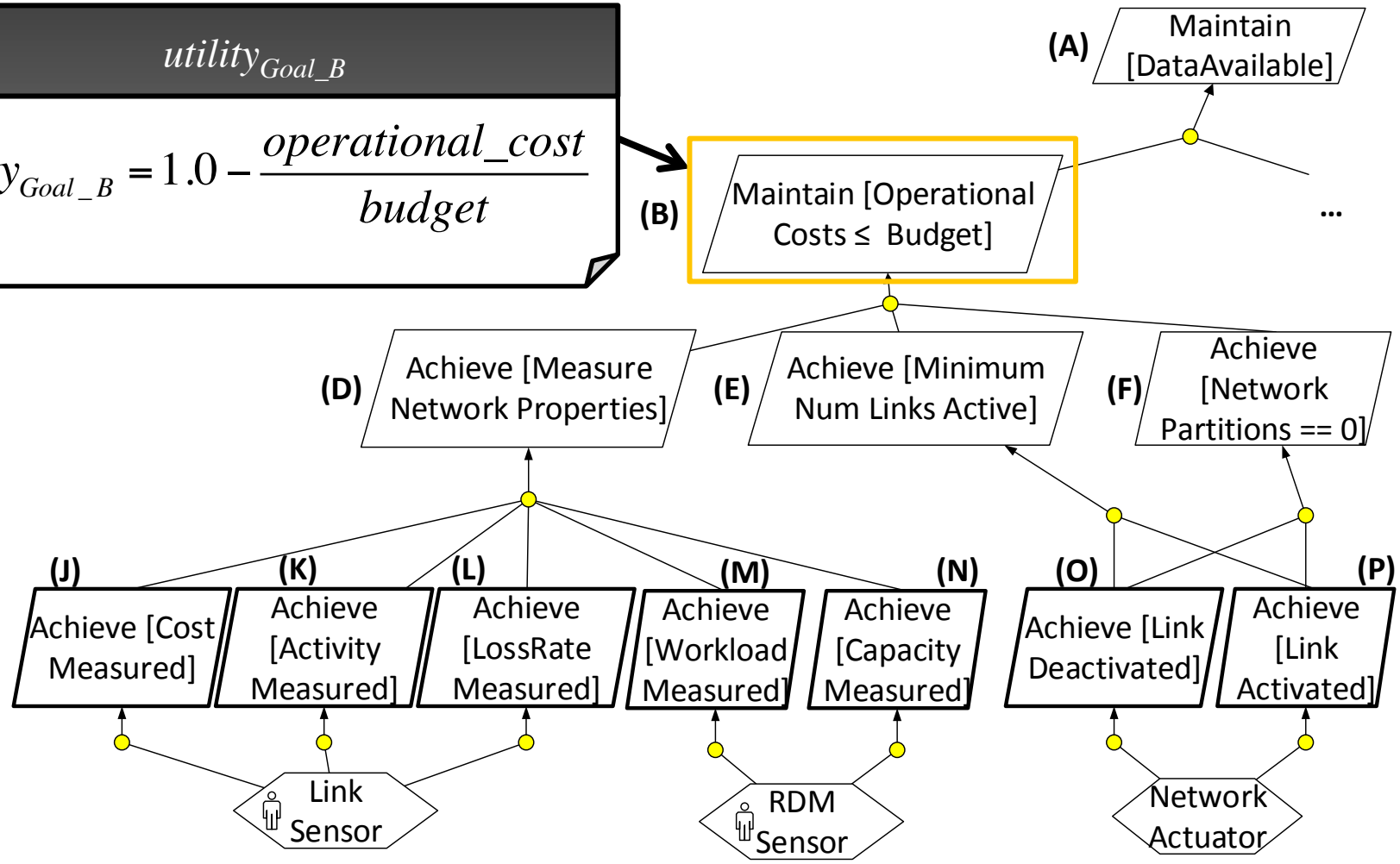
Goal-Oriented Requirements Engineering



Utility Functions

$$utility_{Goal\_B}$$

$$utility_{Goal\_B} = 1.0 - \frac{operational\_cost}{budget}$$

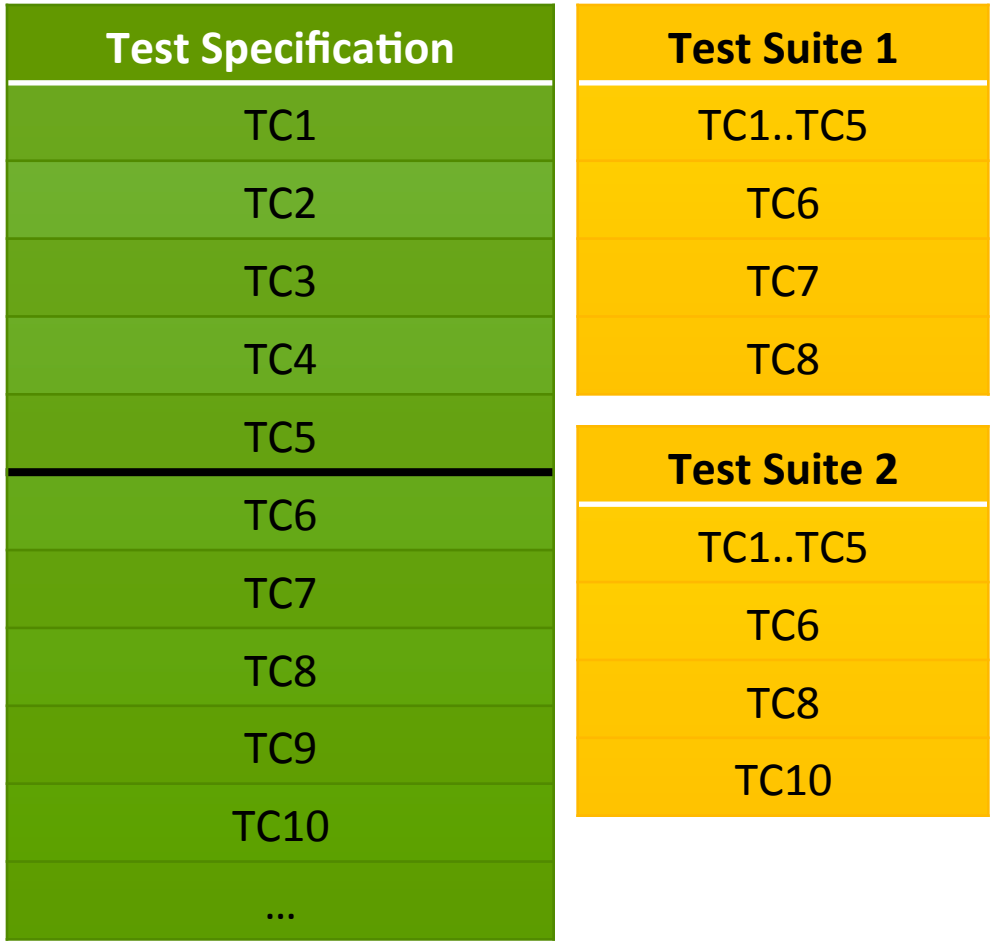


Software Testing

- Requirements-based testing
  - Validate that system under test is satisfying requirements
    [Myers2011,IEEE2010]
- Regression testing
  - Re-validate system following major change to software
    [Myers2011,IEEE2010]

Software Testing

- Terminology
  - Test case
    - Single test to assess all or a portion of a requirement
  - Test specification
    - Set of all possible test cases derived for a software system
  - Test suite
    - Subset of test cases from the test specification
    - Typically derived to be executed under a particular operating context



Software Testing

- Terminology

Test Suite 1
TC1..TC5
TC6
TC7
TC8

o assess all or a
requirement

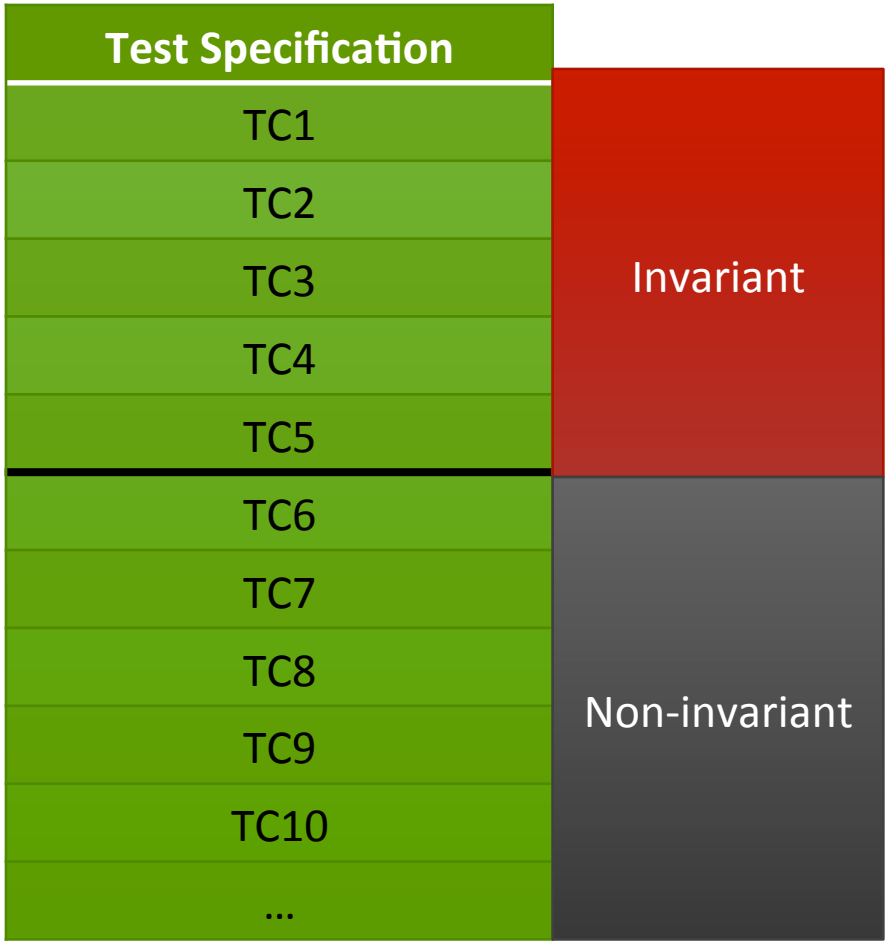
on

ssible test cases
derived for a software system

Test Suite 2
TC1..TC5
TC6
TC8
TC10

st cases from the
ation

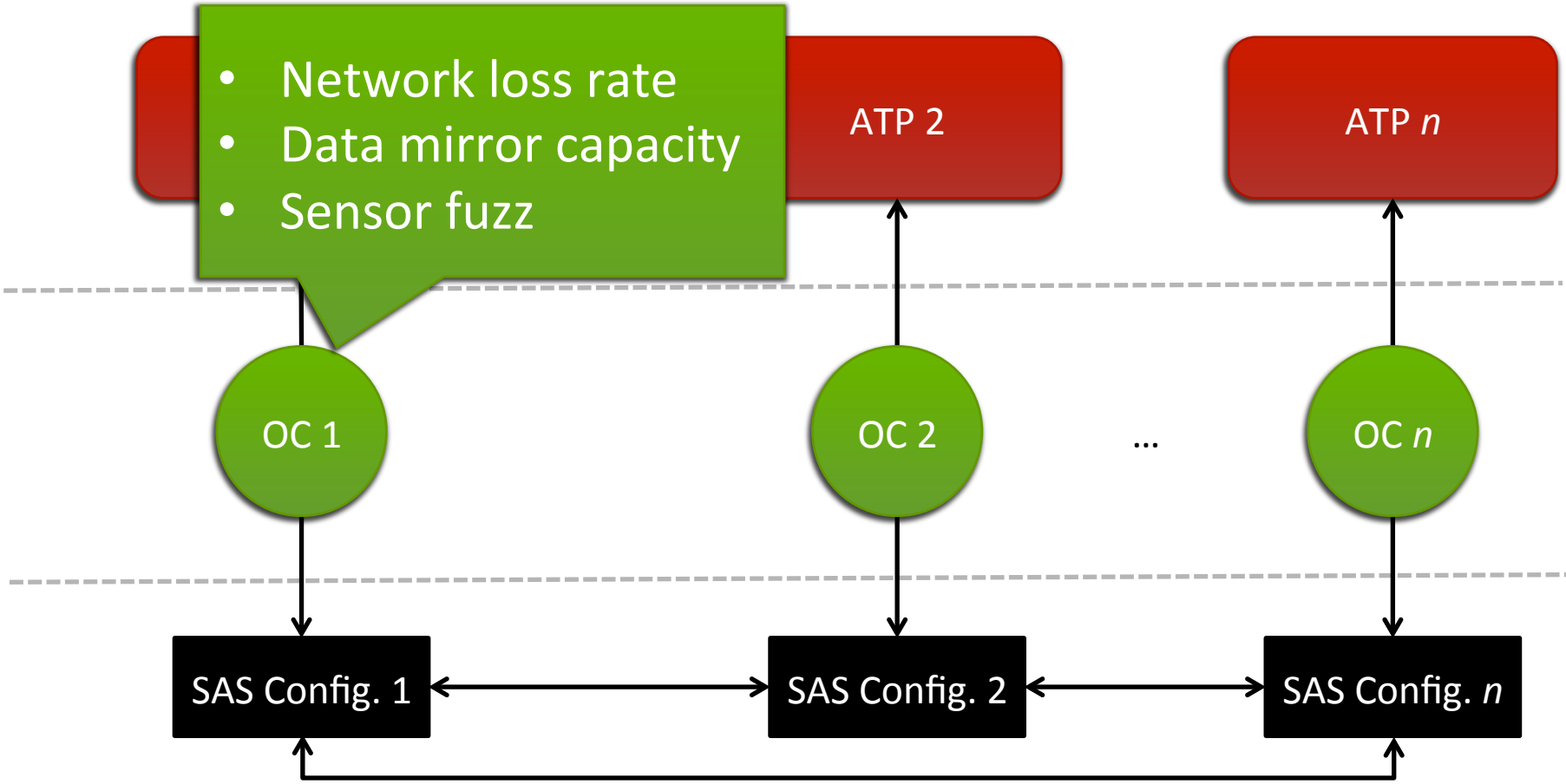
ived to be
der a particular
operating context



Proteus Approach

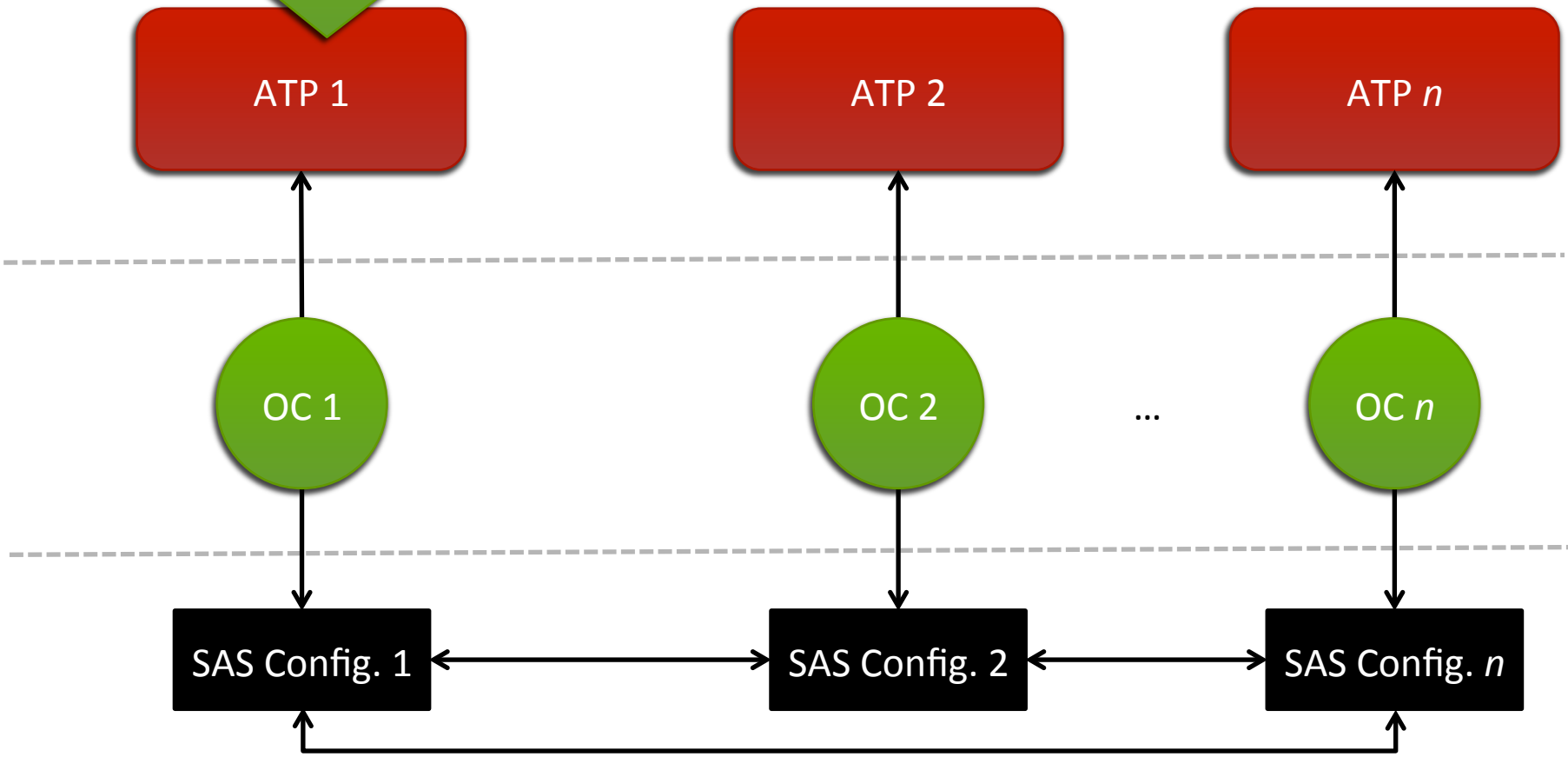
- Requirements-driven approach for managing run-time testing
- Defines an **adaptive test plan** for each SAS configuration
  - Each configuration corresponds to a particular set of environmental conditions, or *operating context*
- Performs a testing cycle during each timestep of SAS execution

Adaptive Test Plans



Adaptive Test Plans

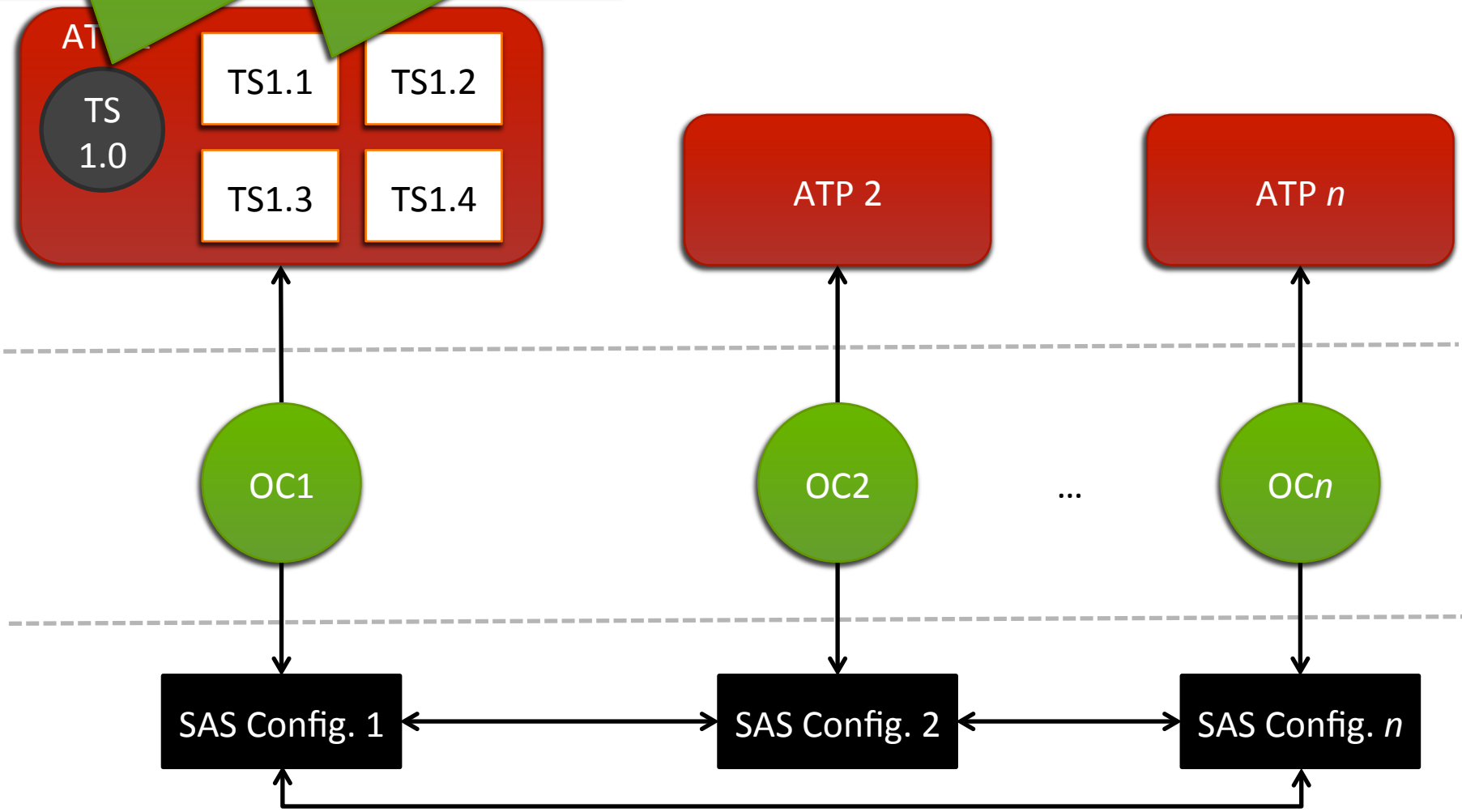
Comprises all test suites for given operating context



daptive Test Plans

Defined

Automatically derived by Proteus



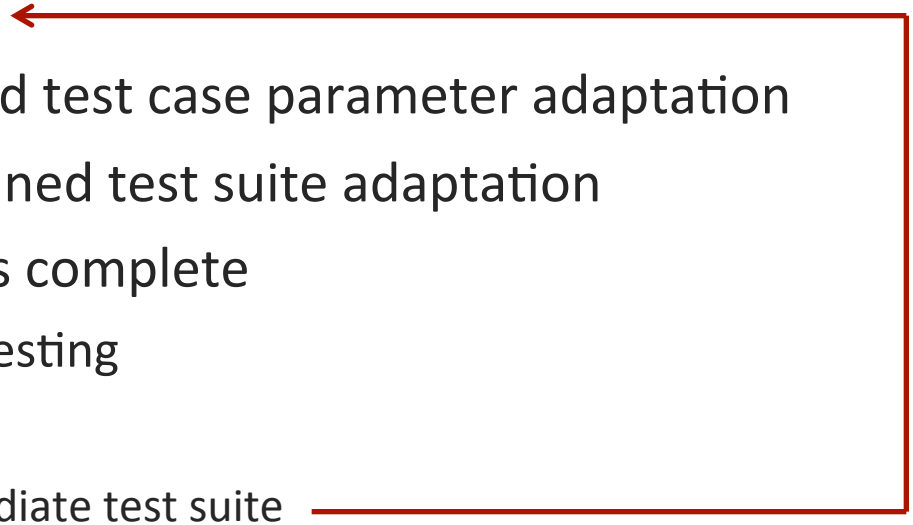
Test Case Activation State

- Test cases within test suite have an activation state:
  - **ACTIVE**: Executed when current test suite is performed
  - **INACTIVE**: Not executed when current test suite is performed
  - **N/A**: Not executed, as it is not relevant to current operating context

- Default test suite ($TS_{k.0}$):
  - Relevant to operating context: **ACTIVE**
  - Irrelevant test cases labeled: **N/A**

Testing Cycle

- Testing cycle at each step of SAS execution
  1. Execute default test suite
  2. Analyze test results
  3. Perform fine-grained test case parameter adaptation
  4. Perform coarse-grained test suite adaptation
  5. Determine if cycle is complete
     1. If complete: halt testing
     2. If not complete:
        1. Execute intermediate test suite



Test Case Fitness

- Test case fitness (**relevance**) is defined as:

$$relevance_{TC} = 1.0 - \frac{\left| value_{measured} - value_{expected} \right|}{value_{expected}}$$

- For example:

High relevance to environment

- Test case expected value = **0.50**
- Test case measured value = **0.45**
- **Fitness = 0.90**

Low relevance to environment

Test case expected value = **0.50**

Test case measured value = **0.01**

Fitness = 0.02

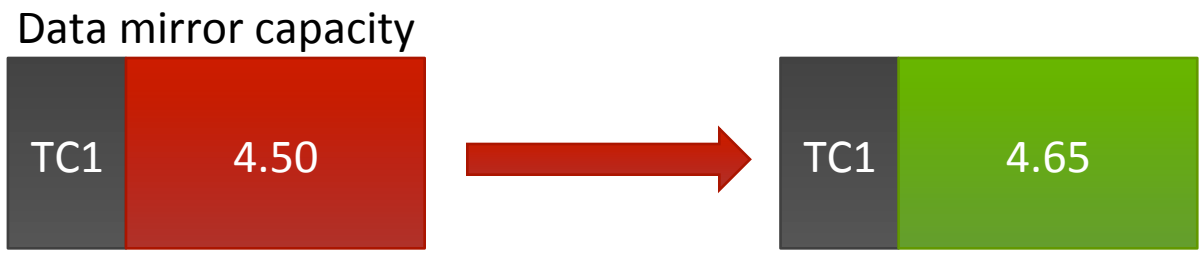
Results Analysis

- Each test case is correlated to at least one goal for validation
  - Test result validated against utility value

- **True positive** → No action necessary
  - Test case relevance = [Threshold, 1.0]
  - Utility value(s) > 0.0
- **True negative** → Error detected in SAS , perform reconfiguration
  - Test case relevance = [0.0, Threshold)
  - Utility value(s) = 0.0
- **False positive** → Error detected in both SAS and test case, adapt both
  - Test case relevance = [Threshold, 1.0]
  - Utility value(s) = 0.0
- **False negative** → Error detected in test case, adapt test case
  - Test case relevance [0.0, Threshold)
  - Utility value(s) > 0.0

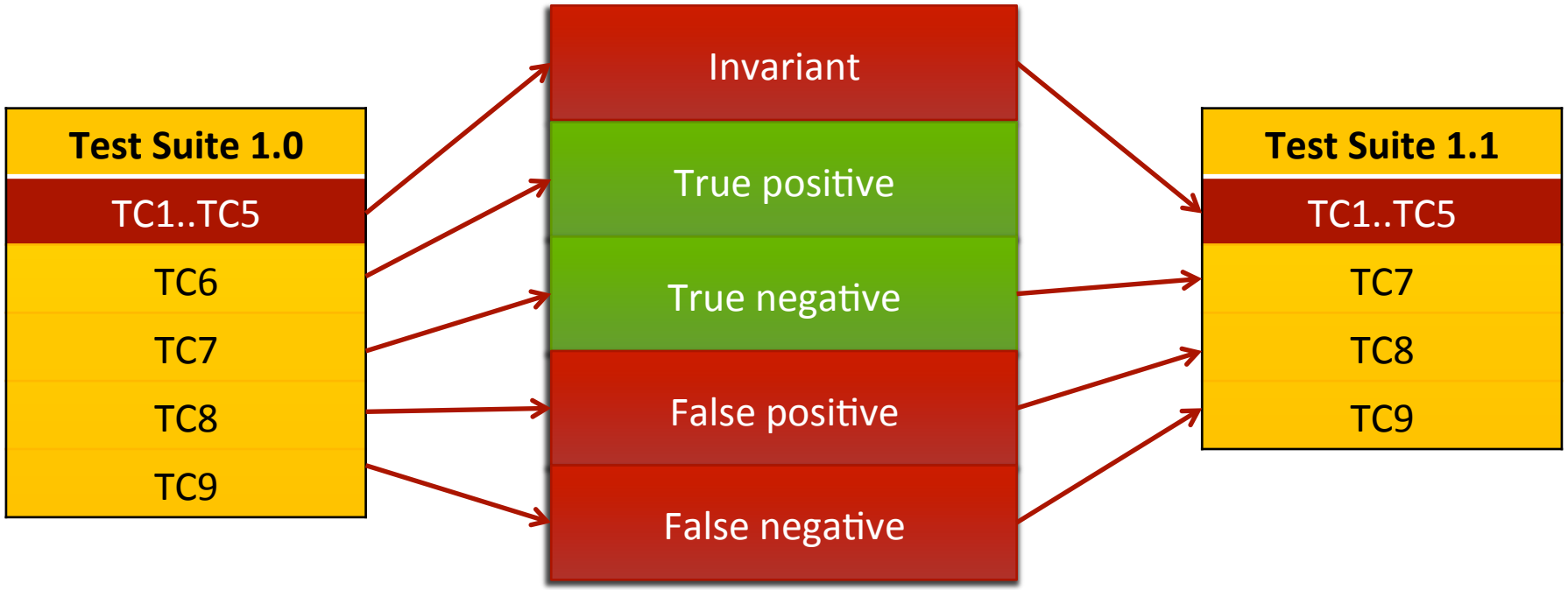
Fine-grained Adaptation

- **Veritas** [Fredericks2014.SEAMS]
  - Adapts non-invariant false positive and false negative test cases
  - Online evolutionary algorithm
  - Searches for a better test case *expected value*
    - Addresses system or environmental uncertainty for each operating context



Coarse-grained Adaptation

- Dynamically generate test suites based on test results



End of Testing Cycle

- Testing cycle terminates when:
  - New SAS configuration is invoked
    - New testing cycle initiated
  - All test cases result in true positives

- If the cycle continues, then the dynamically-generated test suite is executed instead of the default test suite

Case Study

- Simulated RDM network
  - **[15, 30]** data mirrors
  - **[100, 200]** data messages
  - **300** timesteps

- Uncertainty at each timestep:
  - Unpredictable network link failures
  - Randomly dropped or delayed messages
  - Noise applied to data mirror sensors / network links

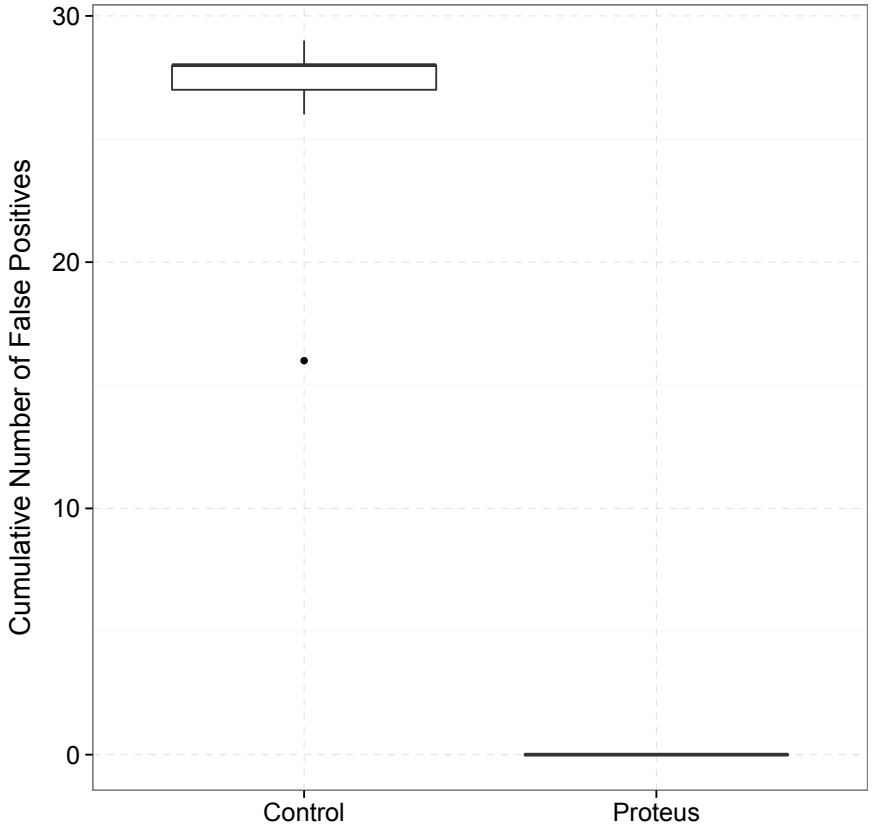
Case Study

- Test specification:
  - 36 test cases
    - 7 invariant [precluded from adaptation]
    - 29 non-invariant [can be adapted]

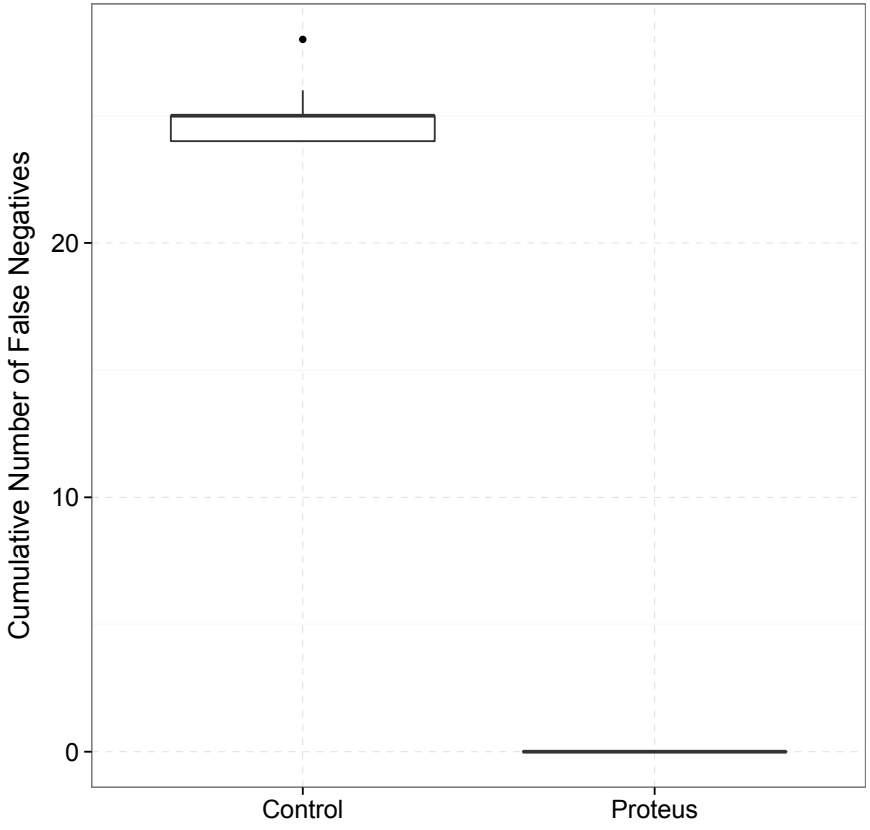
- Compared Proteus adaptive test plans to a manually-derived Control test plan
  - Control test suite:
    - All test cases from test specification relevant to each operating context

Proteus and Veritas

Executed false positive test cases, i.e., **test case relevance = [Threshold, 1.0], utility value = 0.0**

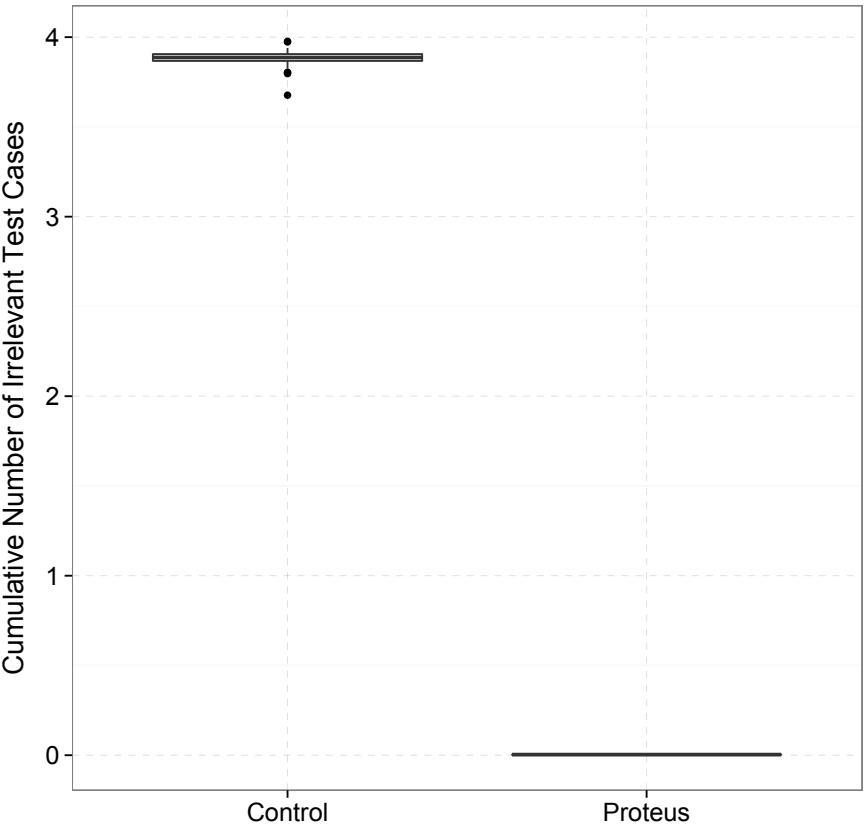


Executed false negative test cases, i.e., **test case relevance = [0.0, Threshold), utility value > 0.0**

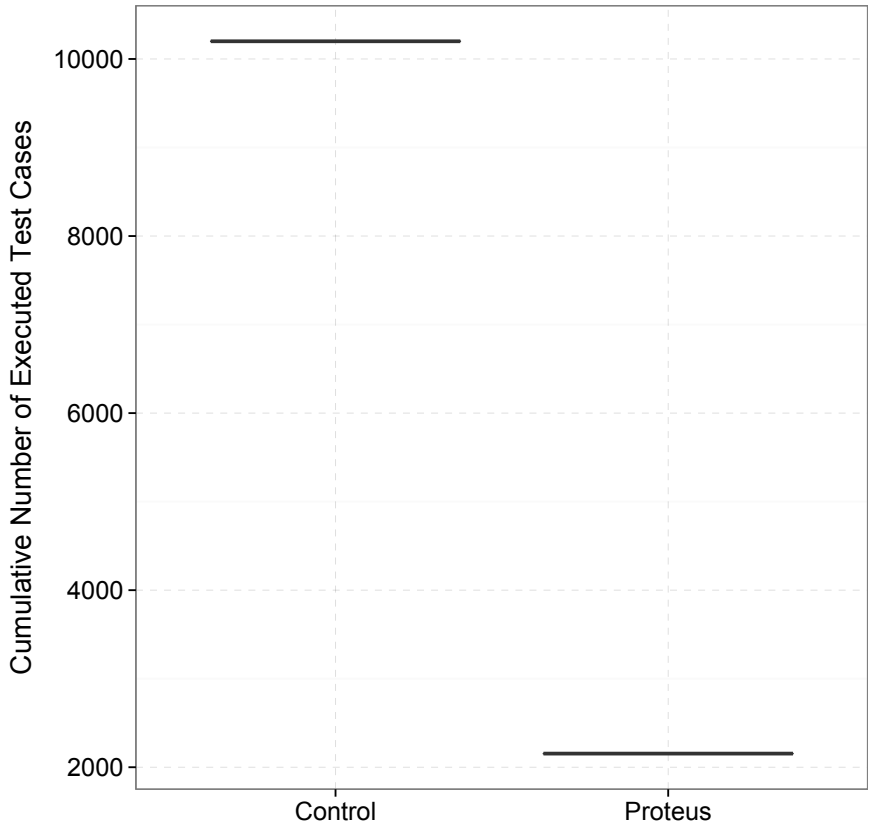


Experimental Results

Executed irrelevant test cases, i.e., **test case relevance = 0.0**



Total number of executed test cases



Case Study Discussion

- Adaptive testing provided by Proteus framework supported by Veritas
- Test suites and test cases remain relevant to changing environmental conditions
- Reduces number of irrelevant test cases executed at run time

Impact of Run-Time Testing

- Analyzed impact of our framework:
  - Execution time
  - Memory overhead
  - Requirements satisficement
  - Number of reconfigurations

Impact Study

- Tested **three** states:
  - (S1): All run-time testing activities enabled
    - Proteus+Veritas enabled
  - (S2): All run-time testing activities disabled
    - Proteus+Veritas disabled
  - (S3): Run-time testing framework removed
    - Proteus+Veritas code / data structures removed from implementation

Impact

- Execution time
  - Measured total execution time of RDM simulation

	(S1)	(S2)	(S3)
Average execution time (seconds)	23.030	13.901	13.785

Impact

- Execution time
  - Measured total execution time of RDM simulation

	(S1)	(S2)	(S3)
Average execution time (seconds)	23.030	13.901	13.785

Significant ($p < 0.05$)

Impact

- Execution time
  - Measured total execution time of RDM simulation

	(S1)	(S2)	(S3)
Average execution time (seconds)	23.030	13.901	13.785

- Memory footprint
  - Measured maximum memory usage of RDM simulation

	(S1)	(S2)	(S3)
Average memory usage (megabytes)	65.234	65.332	65.020

Impact

- Execution time
  - Measured total execution time of RDM simulation

	(S1)	(S2)	(S3)
Average execution time (seconds)	23.030	13.901	13.785

Not significant (p > 0.05)

- Memory footprint
  - Measured maximum memory usage of RDM simulation

	(S1)	(S2)	(S3)
Average memory usage (megabytes)	65.234	65.332	65.020

Impact

- Requirements satisficement
  - Calculated average utility value over simulation

	(S1)	(S2)	(S3)
Average utility value	0.7717	0.7656	0.7656

Impact

- Requirements satisficement
  - Calculated average utility value over simulation

	(S1)	(S2)	(S3)
Average utility value	0.7717	0.7656	0.7656

Not significant (p > 0.05)

Impact

- Requirements satisficement
  - Calculated average utility value over simulation

	(S1)	(S2)	(S3)
Average utility value	0.7717	0.7656	0.7656

- Number of reconfigurations
  - Averaged number of triggered RDM reconfigurations

	(S1)	(S2)	(S3)
Average number of reconfigurations	23.00	17.28	19.16

Impact

- Requirements satisficement
  - Calculated average utility value over simulation

	(S1)	(S2)	(S3)
Average utility value	0.7717	0.7656	0.7656

Not significant (p > 0.05)

- Number of reconfigurations
  - Averaged number of triggered RDM reconfigurations

	(S1)	(S2)	(S3)
Average number of reconfigurations	23.00	17.28	19.16

Discussion of Testing Impact

- Run-time, adaptive testing only **significantly** impacts RDM in terms of execution time
  - Exploring parallelization strategies to reduce time impact
- While not significant, a clear difference in mean utility values exist in requirements satisficement
  - Timing of measurement causes sampling times to be slightly different

Related Work

- **Search-based software testing**
  - Techniques such as evolutionary computation, hill climbing, simulated annealing used for different testing approaches in model testing [Harman2009], regression testing [Harman2012], and structural testing [McMinn2011]
  - EvoSuite [Fraser2011] and Nighthawk [Andrews2011] are evolutionary frameworks for generating test suites and instantiating unit tests
  - Veritas uses a run-time evolutionary algorithm, whereas the other techniques focus on design time search

- **Run-time testing**
  - Implemented using reinforcement learning [Veanes2006], recording & replaying [Tsai1990], and Markov modeling [Filieri2011] approaches
  - Veritas combines evolutionary search for test parameters with utility-based validation
  - Proteus maintains relevance of test suites as conditions change

Related Work

- **Test suite generation**
  - Requirements specification used to generate formal grammars [Bauer1979]
    - Proteus generates new test suites based upon a pre-defined default test suite and executes based on monitored conditions
  - Artificial intelligence used to automatically generate test plans for graphical user interfaces [Memon2001]
    - Proteus analyzes monitoring information to select appropriate test suite
- **Test case selection and prioritization**
  - Select a representative set of test cases and prioritize their execution [Harman2009]
    - Proteus selects and executes tests at run time
  - Tropos [Nguyen2008] uses agent-based randomized testing to validate multi-agent systems
    - Proteus generates test suites targeted towards specific DAS operating contexts

Acknowledgements

- NSF BEACON Center (www.beacon-center.org)
- NSF grants CCF-0820220, DBI-0939454, CNS-0854931, CNS-1305358
- Ford Motor Company
- General Motors

Q&A

References

- **[Veitch2003]** M. Ji, A. Veitch, and J. Wilkes, "Seneca: Remote mirroring done write," in *USENIX 2003 Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, June 2003, pp. 253–268.
- **[Keeton2004]** K. Keeton, C. Santos, D. Beyer, J. Chase, and J. Wilkes, "Designing for disasters," in *Proc. of the 3rd USENIX Conference on File and Storage Technologies*. Berkeley, CA, USA: USENIX Association, 2004, pp. 59–62.
- **[Dardenne1993]** A. Dardenne, A. Van Lamsweerde, and S. Fickas, "Goal-directed re- quirements acquisition," *Science of computer programming*, vol. 20, no. 1, pp. 3–50, 1993.
- **[vanLamsweerde2009]** A. van Lamsweerde, *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, 2009.
- **[deGrandis2009]** P. deGrandis and G. Valetto, "Elicitation and utilization of application- level utility functions," in *Proc. of the 6th International Conference on Autonomic Computing*, ser. ICAC '09. ACM, 2009, pp. 107–116.
- **[Ramirez2011]** A. J. Ramirez and B. H. C. Cheng, "Automatically deriving utility functions for monitoring software requirements," in *Proc. of the 2011 International Conference on Model Driven Engineering Languages and Systems Conference*, Wellington, New Zealand, 2011, pp. 501–516.
- **[Walsh2004]** W. E. Walsh, G. Tesauro, J. O. Kephart, and R. Das, "Utility functions in autonomic systems," in *Proc. of the First IEEE International Conference on Autonomic Computing*. IEEE Computer Society, 2004, pp. 70–77.
- **[Myers2011]** G. J. Myers, C. Sandler, and T. Badgett, *The art of software testing*. John Wiley & Sons, 2011.
- **[IEEE2010]** IEEE, "Systems and software engineering – vocabulary," *ISO/IEC/IEEE 24765:2010(E)*, pp. 1–418, Dec 2010.

References

- **[Fredericks2014.SEAMS]** E. M. Fredericks, B. DeVries, and B. H. C. Cheng, "Towards run- time adaptation of test cases for self-adaptive systems in the face of uncertainty," in *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '14, 2014.

- **[Harman2009]** M. Harman, S. A. Mansouri, and Y. Zhang, "Search based software engineering: A comprehensive analysis and review of trends techniques and applications," *Department of Computer Science, King's College London, Tech. Rep. TR-09-03*, 2009.

- **[Fraser2011]** Gordon Fraser and Andrea Arcuri. Evosuite: automatic test suite generation for object-oriented software. In Proc. of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering, ES- EC/FSE '11, pages 416–419, Szeged, Hungary, 2011. ACM.

- **[Andrews2011]** James H. Andrews, Tim Menzies, and Felix C.H. Li. Genetic algorithms for randomized unit testing. IEEE Trans. on Software Engineering, 37(1):80–94, January 2011.

- **[Veanes2006]** Margus Veanes, Pritam Roy, and Colin Campbell. Online testing with rein- forcement learning. In Formal Approaches to Software Testing and Runtime Verification, pages 240–253. Springer, 2006.

- **[Tsai1990]** J.J.-P. Tsai, K.-Y. Fang, Horng-Yuan Chen, and Yao-Dong Bi. A noninter- ference monitoring and replay mechanism for real-time software testing and debugging. Software Engineering, IEEE Transactions on, 16(8):897–916, 1990.

- **[Filieri2011]** Antonio Filieri, Carlo Ghezzi, and Giordano Tamburrelli. Run-time efficient probabilistic model checking. In Proc. of the 33rd International Conference on Software Engineering, pages 341–350, Waikiki, Honolulu, Hawaii, USA, 2011. ACM.