Universität Rostock Traditio et Innovatio

Algorithm evaluation - Steps towards more quality in M & S

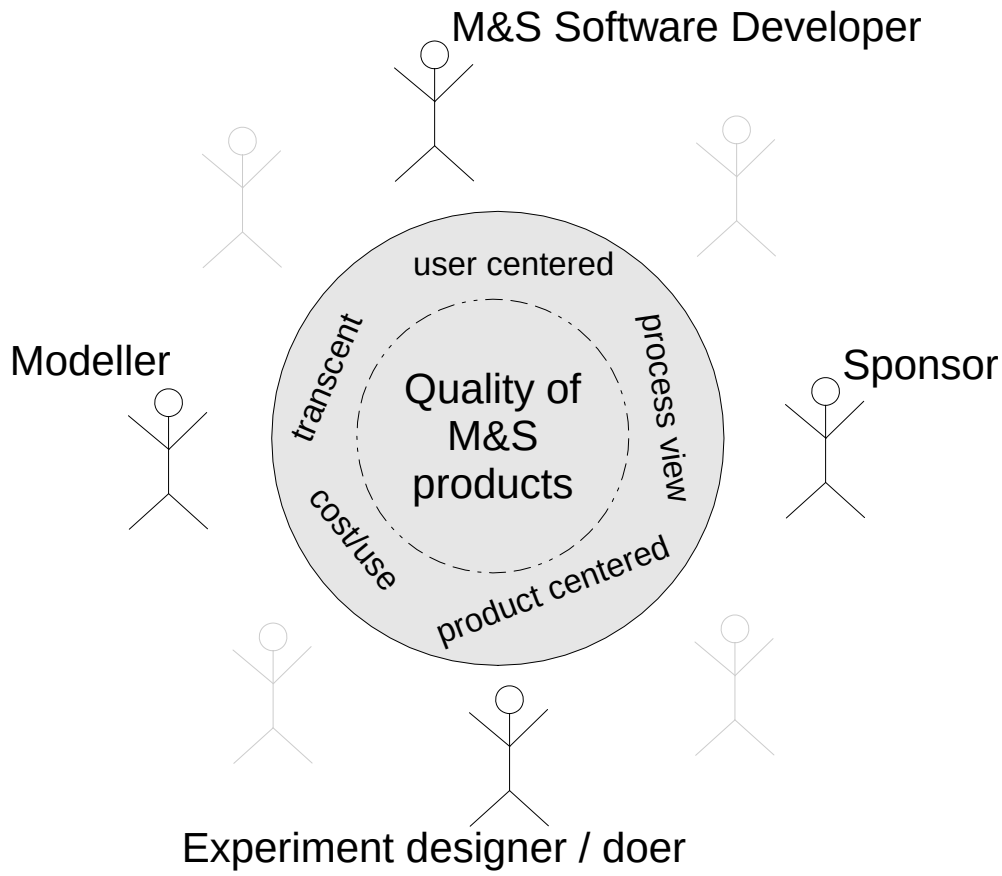
Perspektiven der Modellbildung und Simulation

Roland Ewald, Jan Himmelspach, Stefan Leye, and Stefan Rybacki, *Adelinde M. Uhrmacher*
University of Rostock, Faculty of Computer Science and Electrical Engineering

What does quality of M&S refer to?

- Quality of M&S results
  - Quality of models (right abstraction - right modeling means)
  - Quality of experiments (right overall process, with the right algorithms with sufficient number of runs)
- Quality of software (Well defined development process)
  - “Bug free software” (Well tested, software reuse)
  - Efficiency of the software / scalability (Alternative algorithms, different hardware, ...)

Perspectives on quality



Software development for M&S

"[...] to support practitioners, who (because of lack of background and or time) should be protected from everything other than stating the problem and its context" [Schmeiser 09]

- (not only) "by developing theory, methods, and algorithms" but
- by providing these in an accessible manner
- by supporting and guiding users,
- and (thus) "reducing the sources of errors"

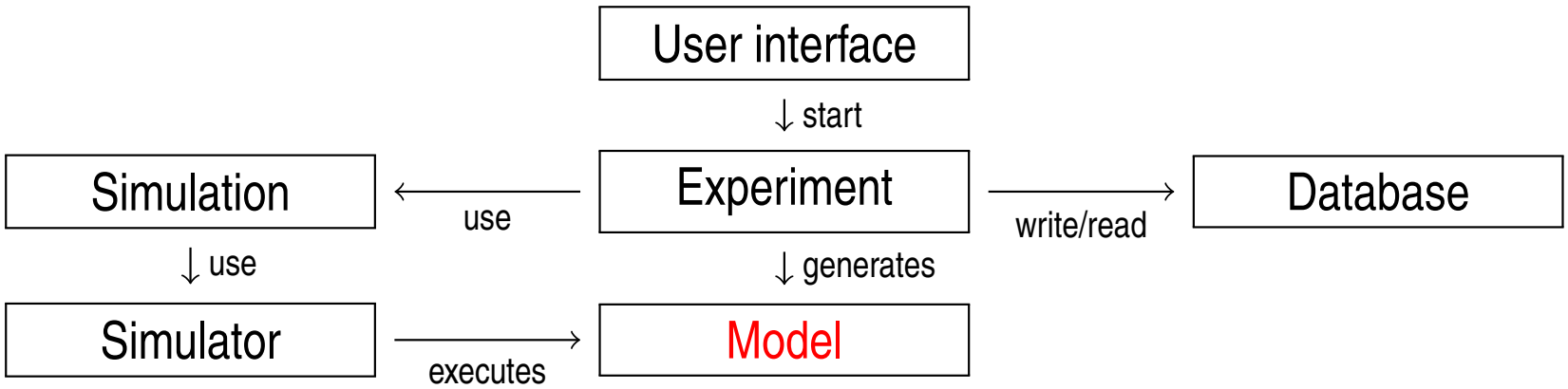
A few first answers from James II

A Java-Based Multi Purpose Modeling Environment for Simulation

- There are (common) principles, techniques, and elements shareable by all M&S software
- why not use an integrative approach and allow intensive reuse (cross tasks / techniques / domains)
  - facilitates development and evaluation of new methods
  - more reliable simulation results
  - currently more than 600 plug-ins
- *Plug 'n simulate*: Base (framework) + plug-ins per task/technique

Modelling in JAMES II

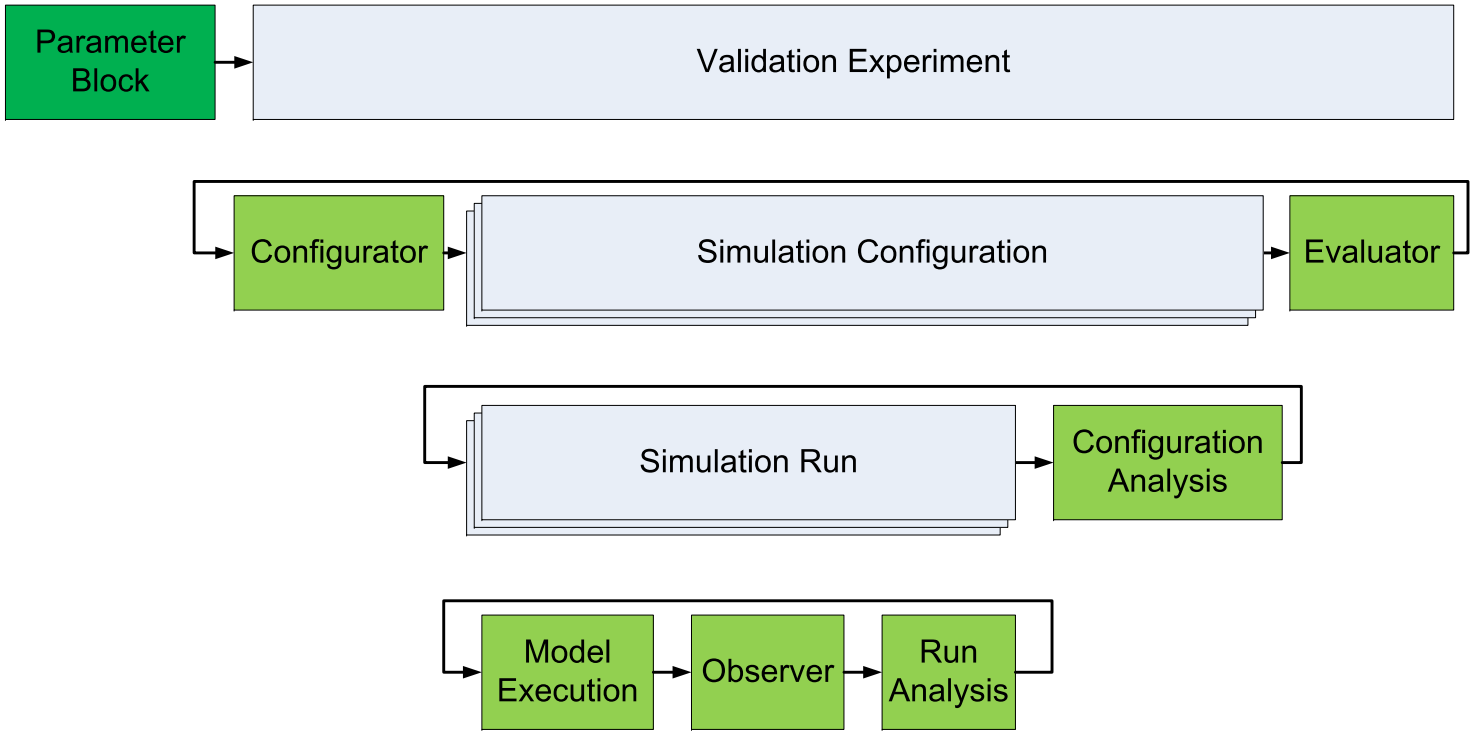
Quality of models: right modeling means



- DEVS-based formalisms
- π-based formalisms
- in addition for teaching: CA, Petri Nets, ...
- + means for composition and model reuse

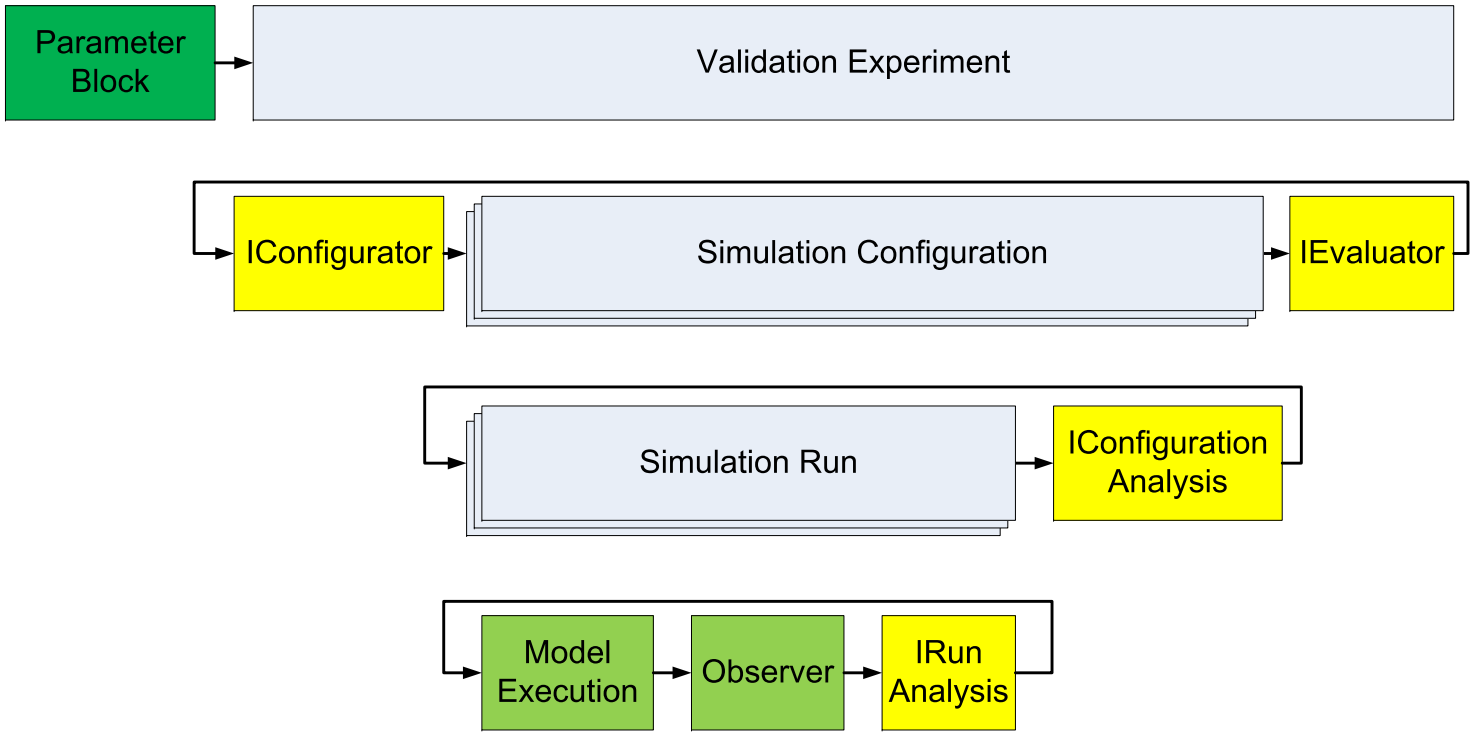
Model validation in JAMES II

Quality of models: right abstraction



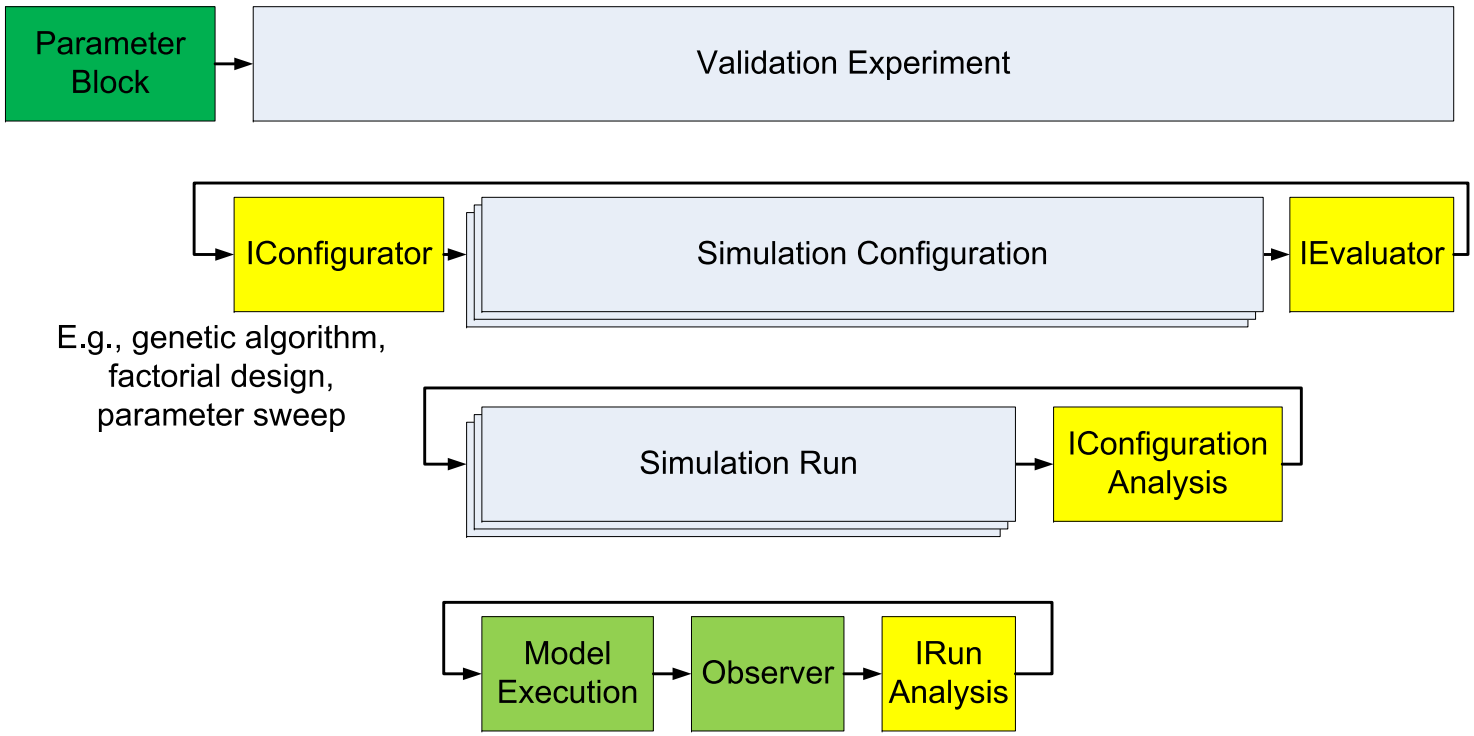
Model validation in JAMES II

Quality of models: right abstraction



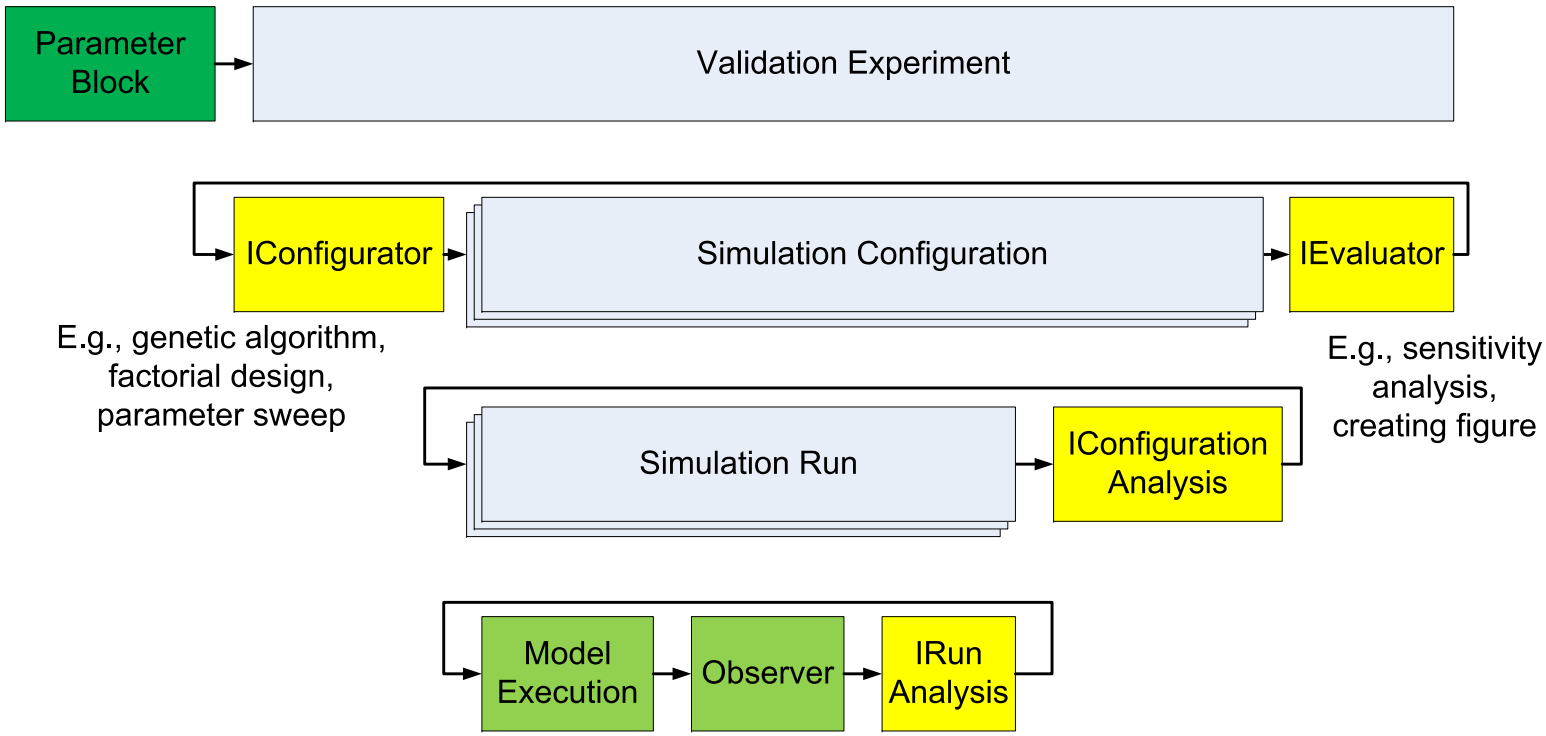
Model validation in JAMES II

Quality of models: right abstraction



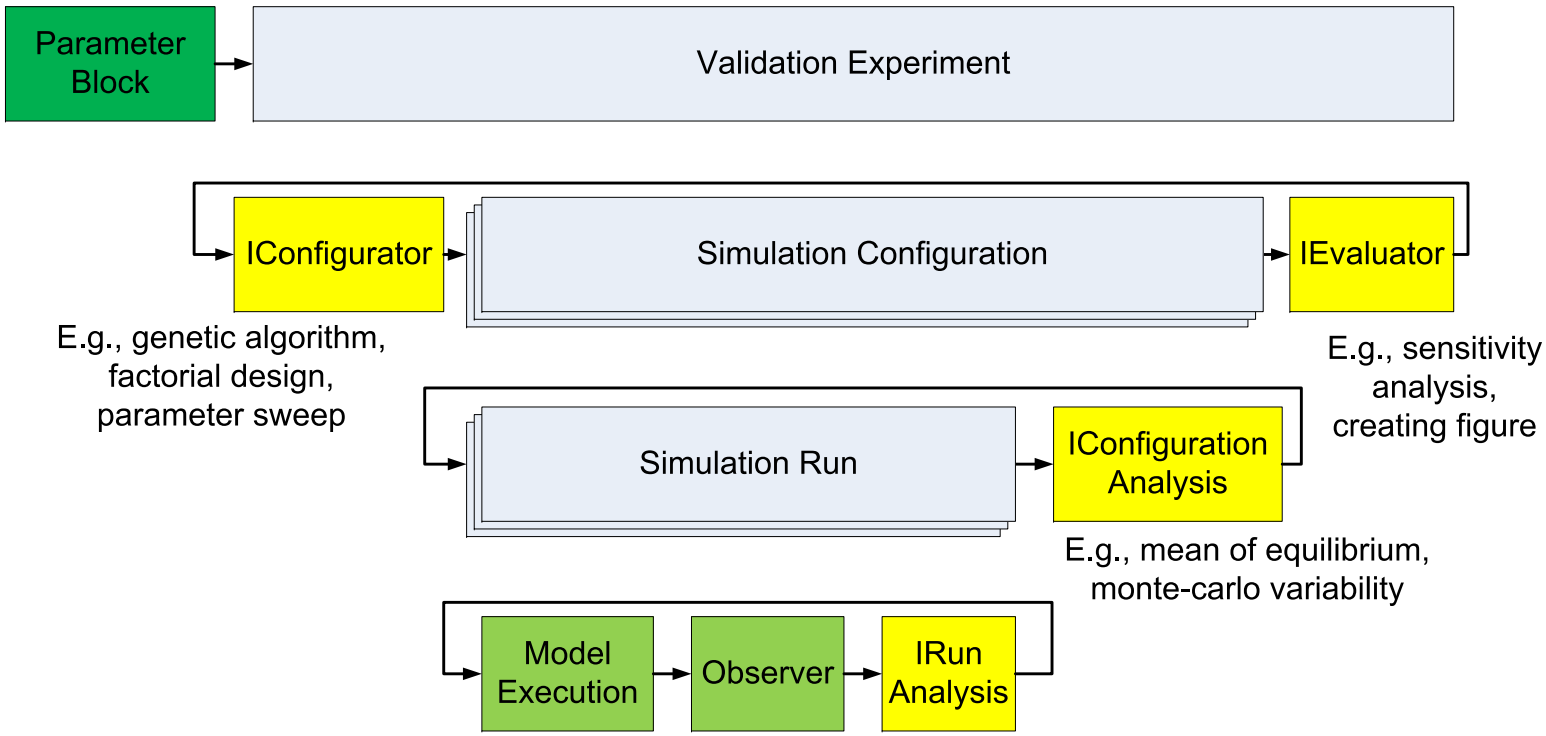
Model validation in JAMES II

Quality of models: right abstraction



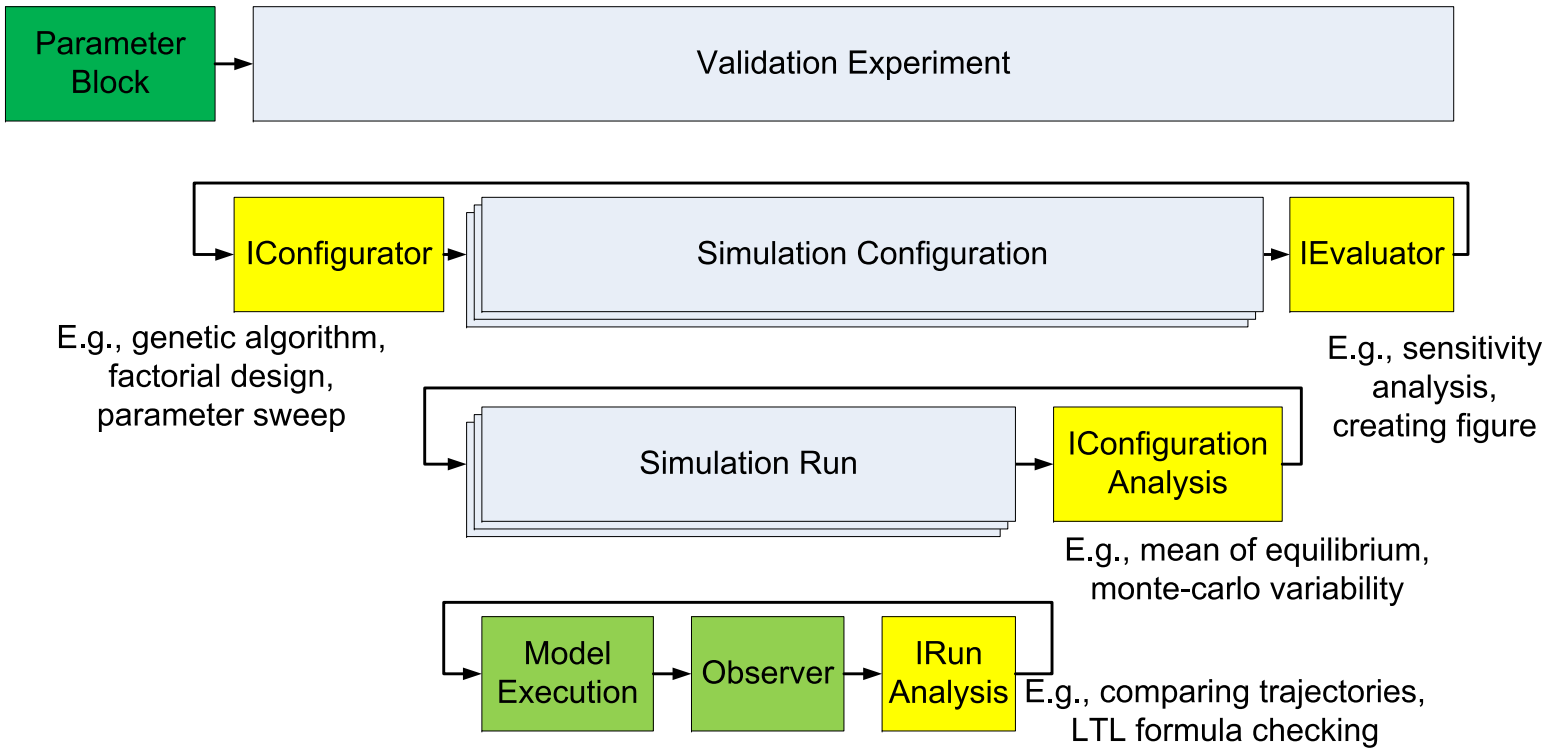
Model validation in JAMES II

Quality of models: right abstraction



Model validation in JAMES II

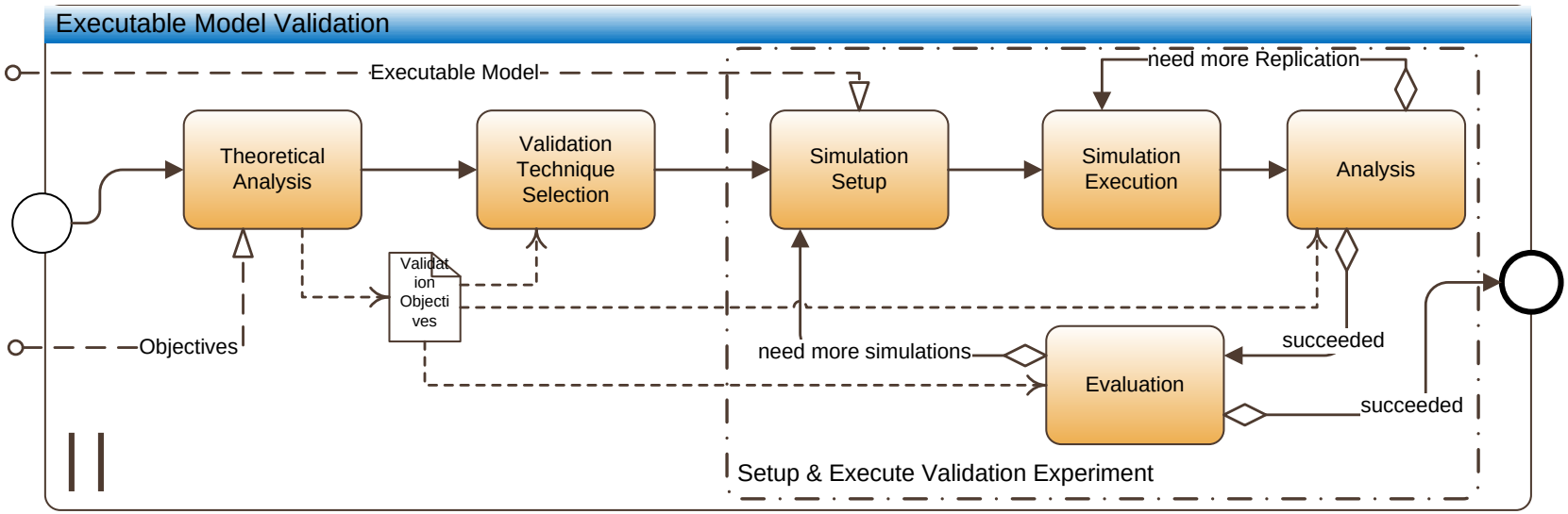
Quality of models: right abstraction



Experimentation support in JAMES II

Quality of experiments: the right process

Quality of models: the right model abstraction

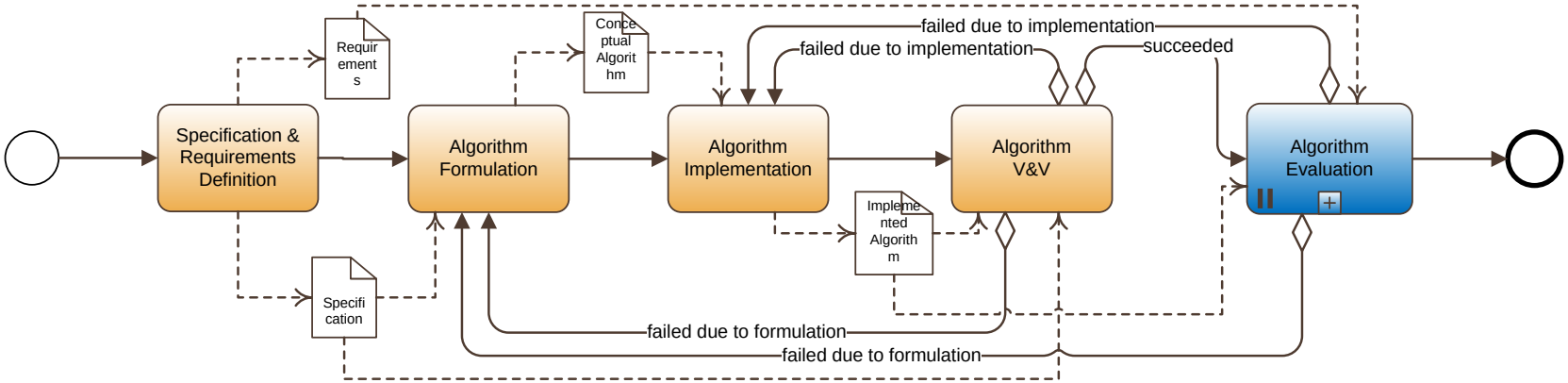


(Sub-) Workflow description for validation of an executable model

Experimentation support in JAMES II

Quality of experiments: the right process

Quality of experiments: the right algorithm



Workflow description for the evaluation of an algorithm

Evaluating (Implemented) M & S Algorithms

Correctness

- (Adapting) Methods from software inspection and testing
- Experimental validation

Efficiency

- Benchmarking

Requirements

- Standardized Test Cases (like SATlib, UCI machine learning repository . . .)
- Alternative algorithms
- Automated testing and evaluation (user transparency)

Sufficient efficiency is the precondition for a sufficient validation!

"`bug-free"' M & S Algorithms?

- Unit tests: essential tool for developers
- Automated testing of corner cases, which need to be defined specifically for specific algorithm classes
- Allows continuous integration
- Facilitates (test-driven) development of new algorithms

Standard Tests for Event Queues

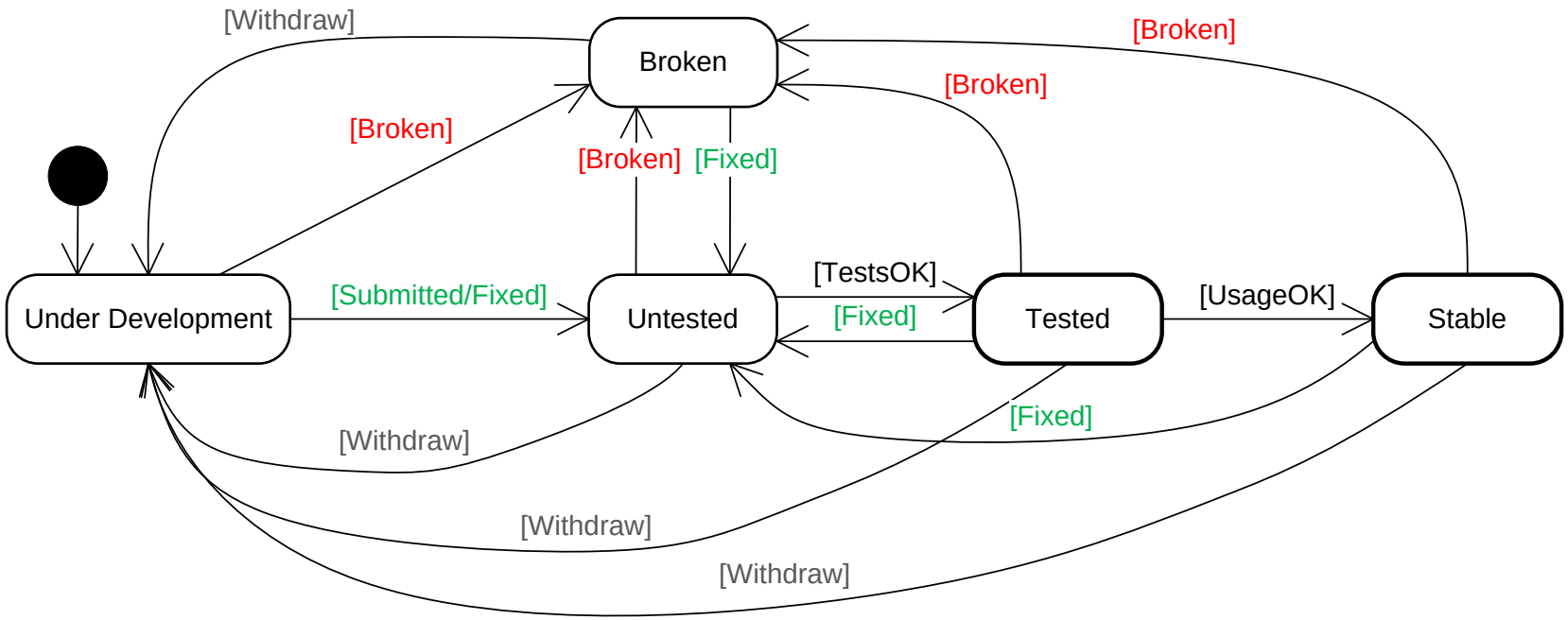
- Test methods for the interface of event queues
- Classical (black box) unit testing
- Can be enriched with further test scenarios (gray box tests) which reflect internals of the implementations

Basic set of tests ensures a minimal testing; all queues have to pass these at least ($\rightarrow$ minimal quality).

Managing Data on Algorithm Validity

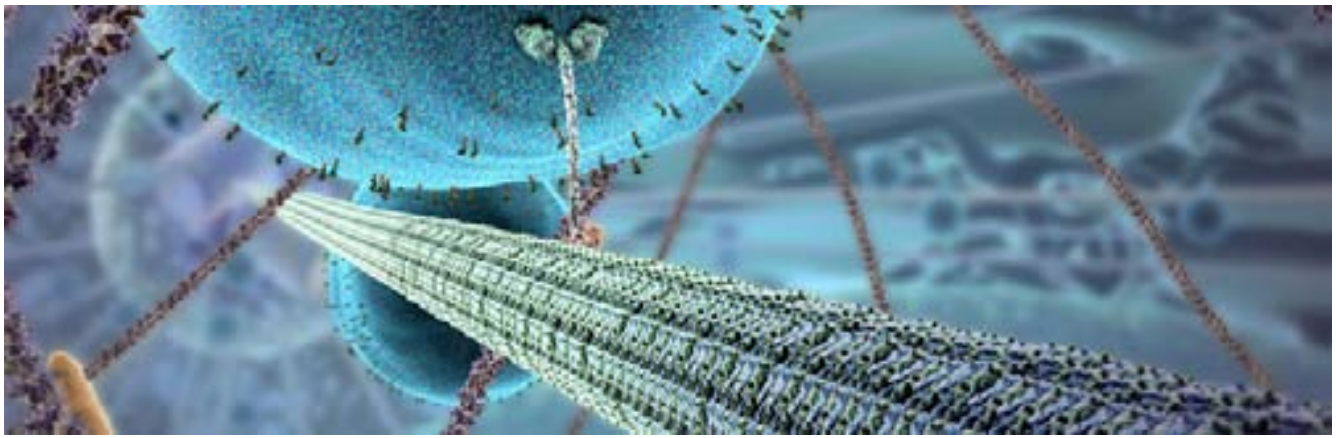
- Validity is important to users
- May be used for deciding upon a certain setup
- Can be managed (and updated) automatically

Example: Plug-in Life Cycle in JAMES II



Efficiency of (implemented) M & S Algorithms

- Scalable
- Flexible
- and transpartent to the user



The inner life of the cell, `http://multimedia.mcb.harvard.edu/`

Bench Models

required to ensure:

- Comparability
- Reproducibility

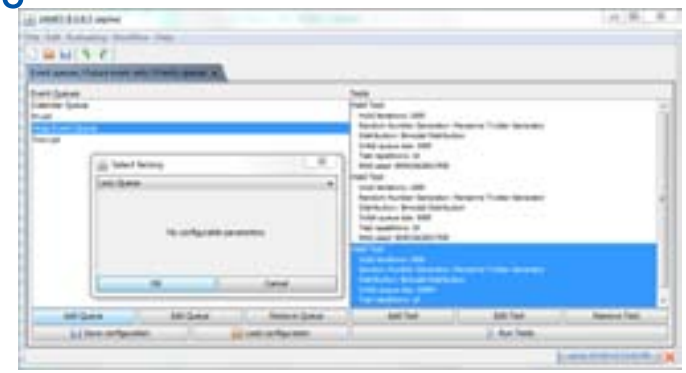
important properties:

- Parameterizability
- Scalability
- Simplicity
- Representativeness

Commonplace in other domains, e.g.,

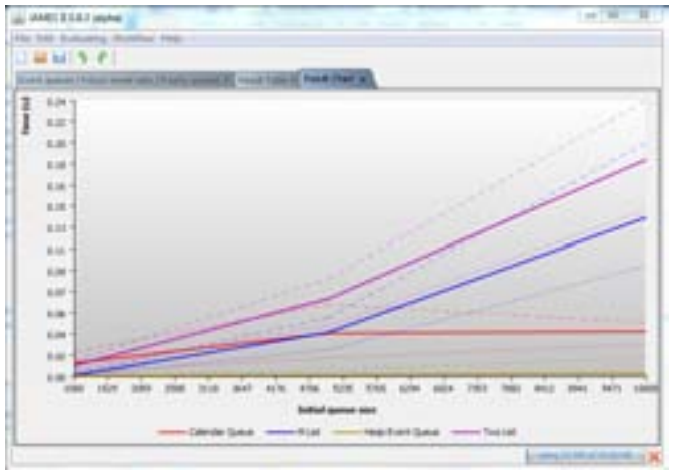
- SAT-solvers: SATLib `http://www.satlib.org`
- UCI machine learning repository `http://archive.ics.uci.edu/ml`

Benchmarking Event Queues



Problems (Std tests)

- Up/Down
- Hold

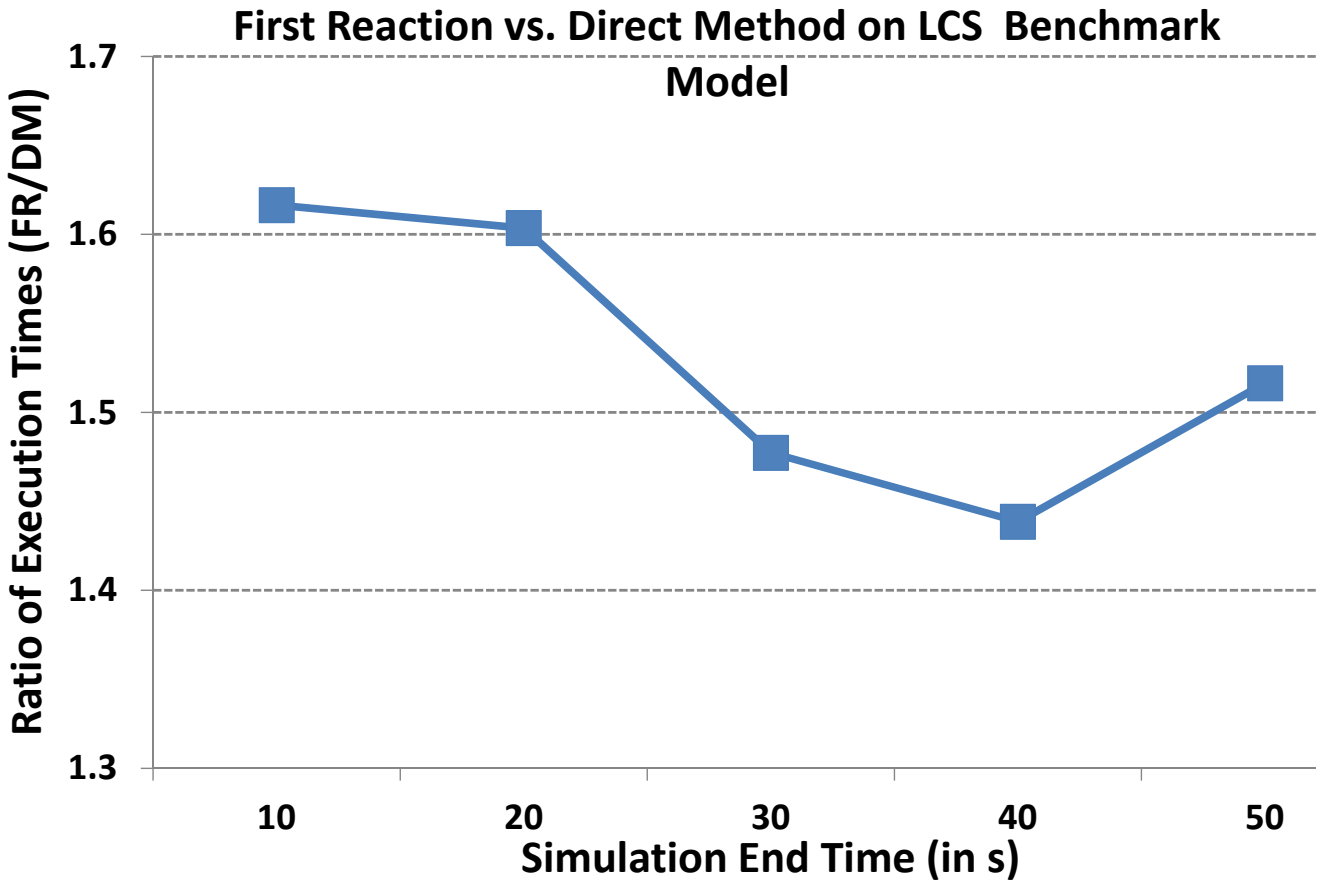


Parameters

- initial distribution of entries
- distributions for enqueue

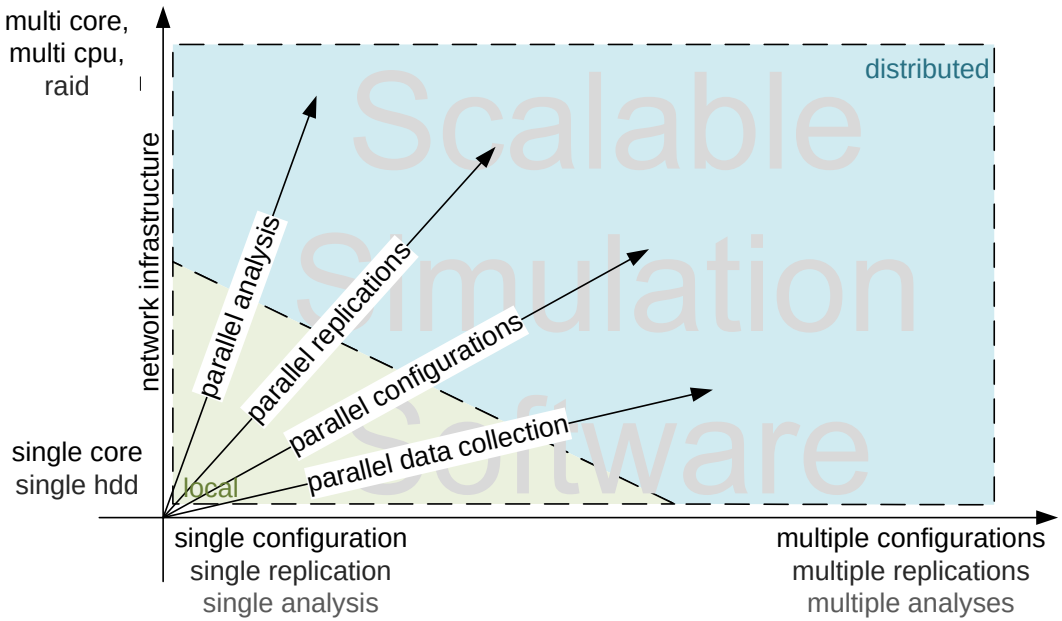
Being in use for more than three decades.

Pitfall: Load of Benchmark Model may *vary* over Time



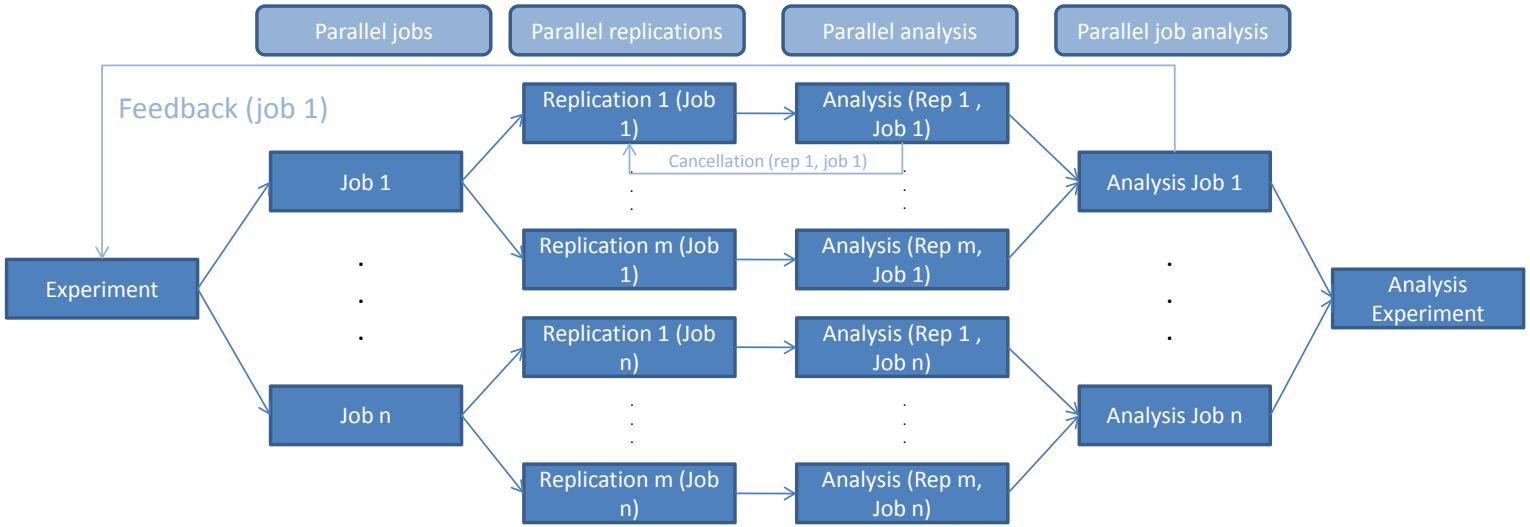
Scalability

- Large number of CPUs / cores per CPU
- Large models (memory, computational demands)
- Large parameter spaces (many trajectories need to be computed)
- Large amounts of (observed) data
- Complex analysis of data



Flexibility

Efficiency of the software: Alternative algorithms

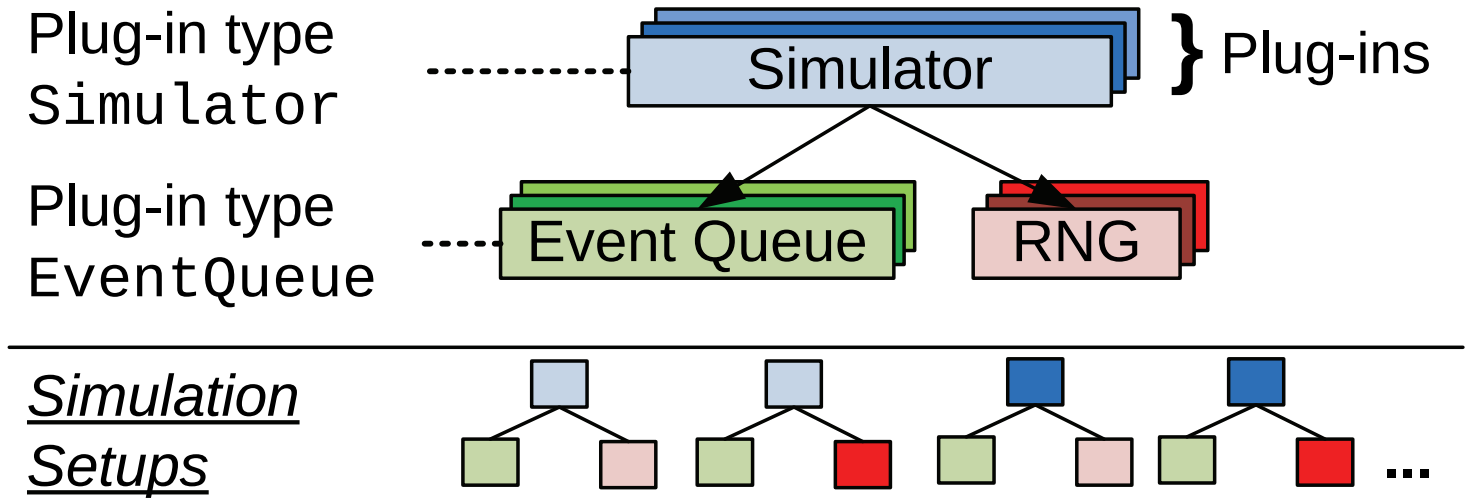


User Transparency and support

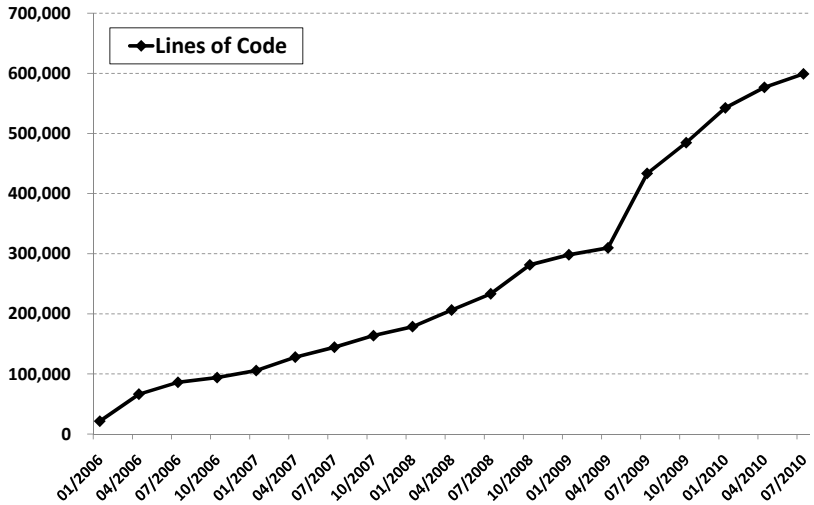
- Flexible and scalable simulation systems are hard to...
  - build
  - evaluate
  - **configure**
- *Simulation Algorithm Selection:*
  What is the best algorithm combination for a given problem?

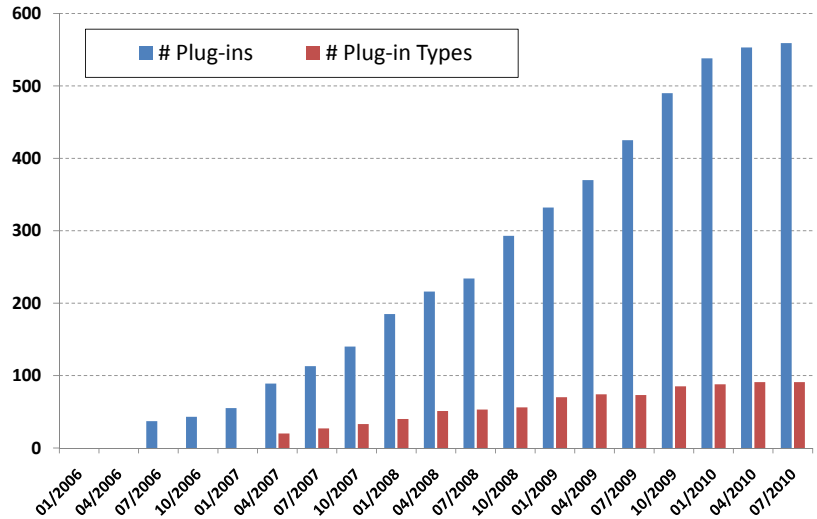
Drew Endy & Isadora Deese, Adventures in Synthetic Biology, Nature, 2005

Algorithm Selection on JAMES II Plug-ins



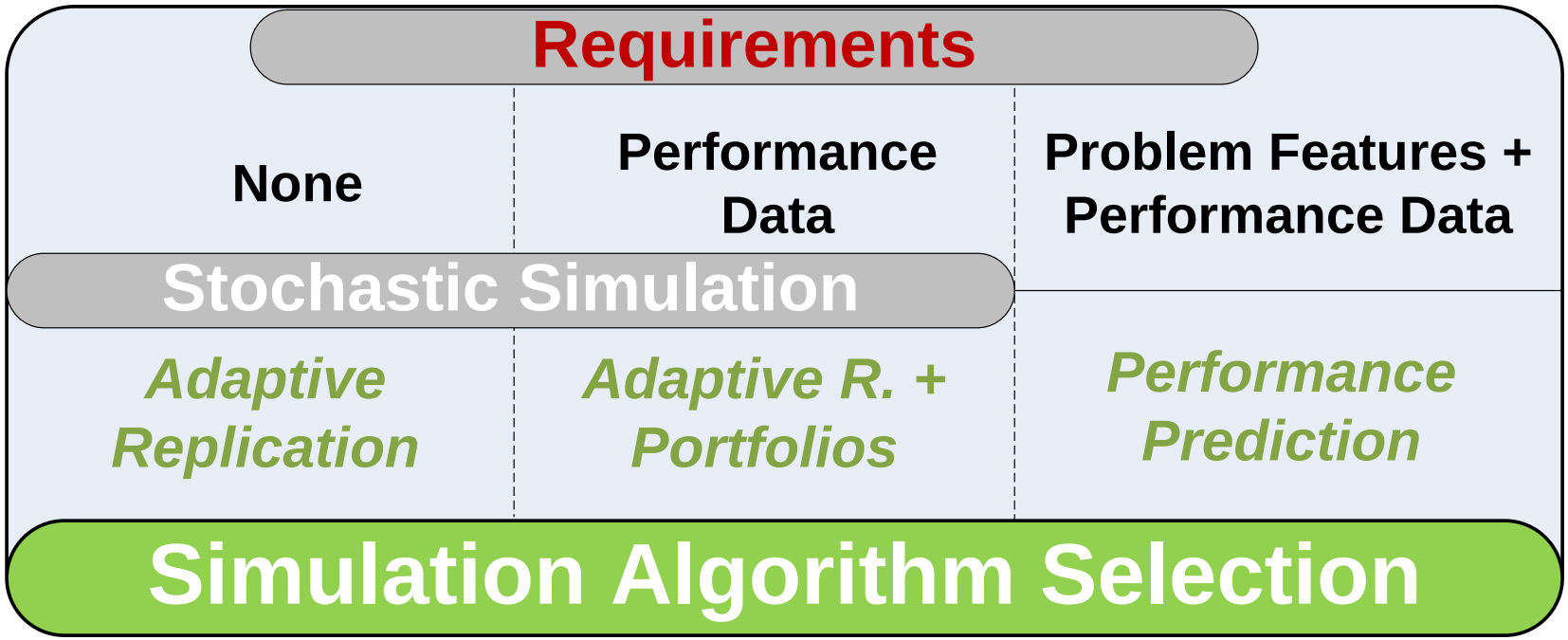
JAMES II Flexibility increases Difficulty



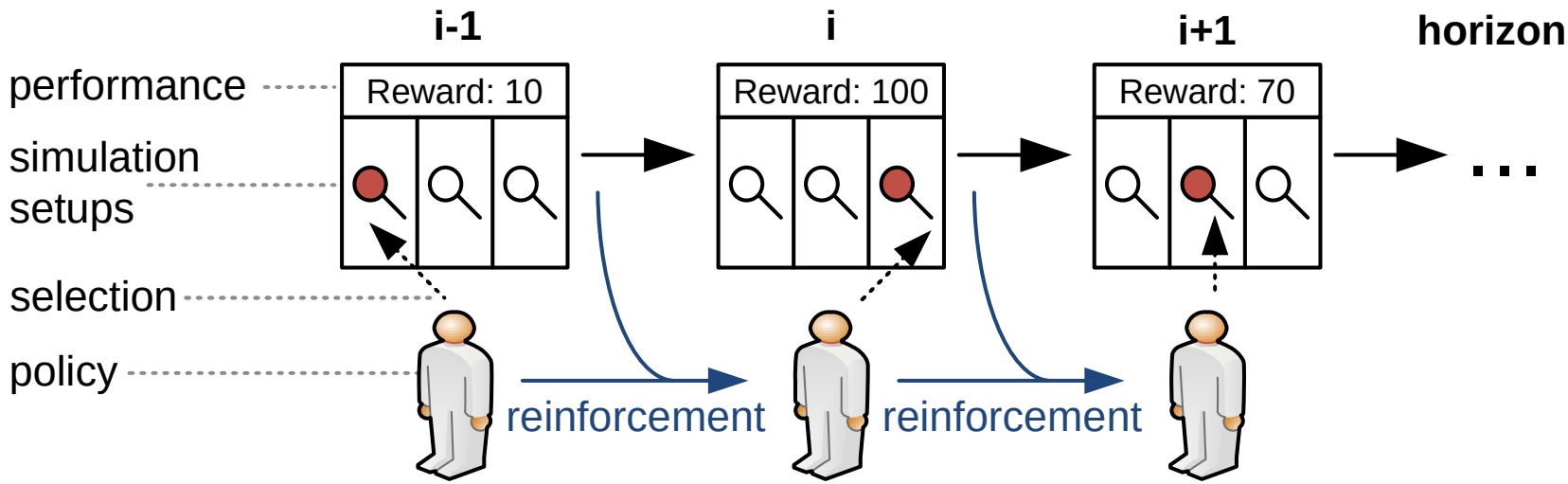


- Code base grows steadily
- Combinatorial explosion
- Algorithm parameters must not be neglected

Algorithm Selection: Different Circumstances

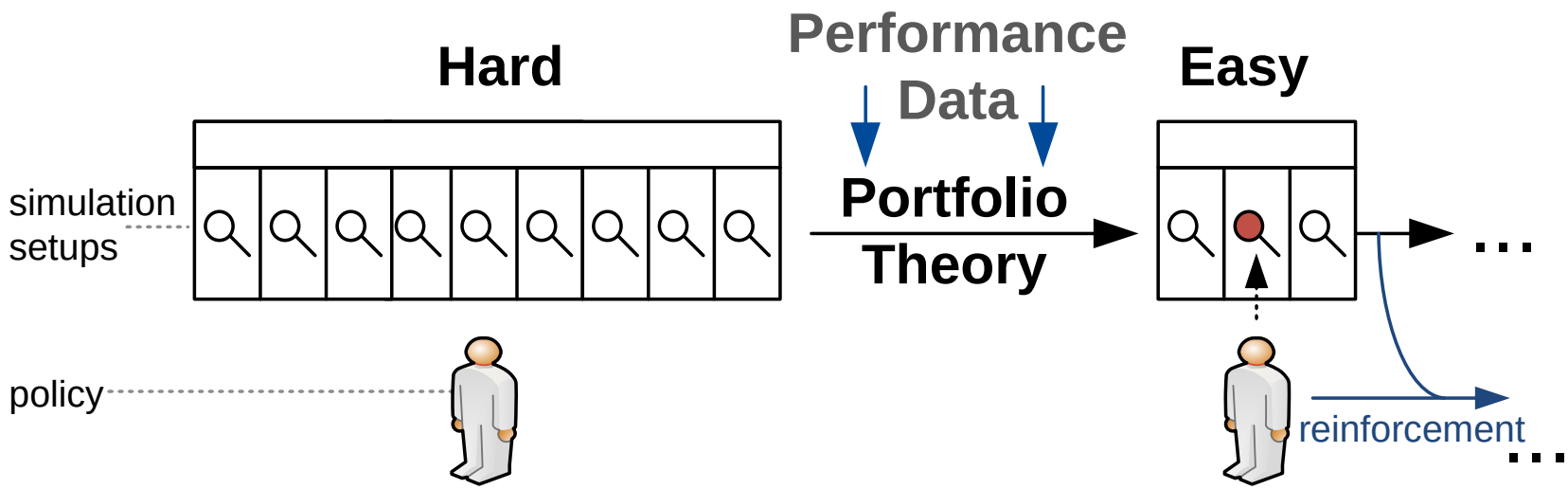


No Prior Knowledge: Adaptive Replication



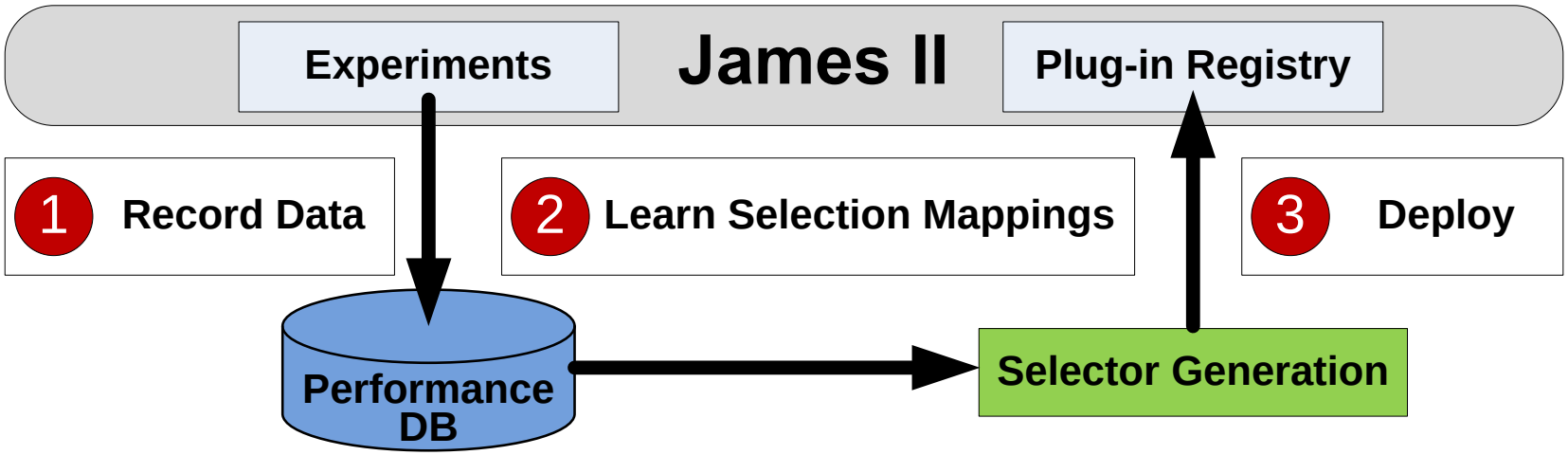
- Restricted to stochastic simulation
- *Policies* to solve the Multi-Armed Bandit Problem
- Requires *sufficiently many replications*
- **Results: speedup up to 3.2** (w.r.t. avg. performance)

Just Performance Data: Algorithm Portfolios



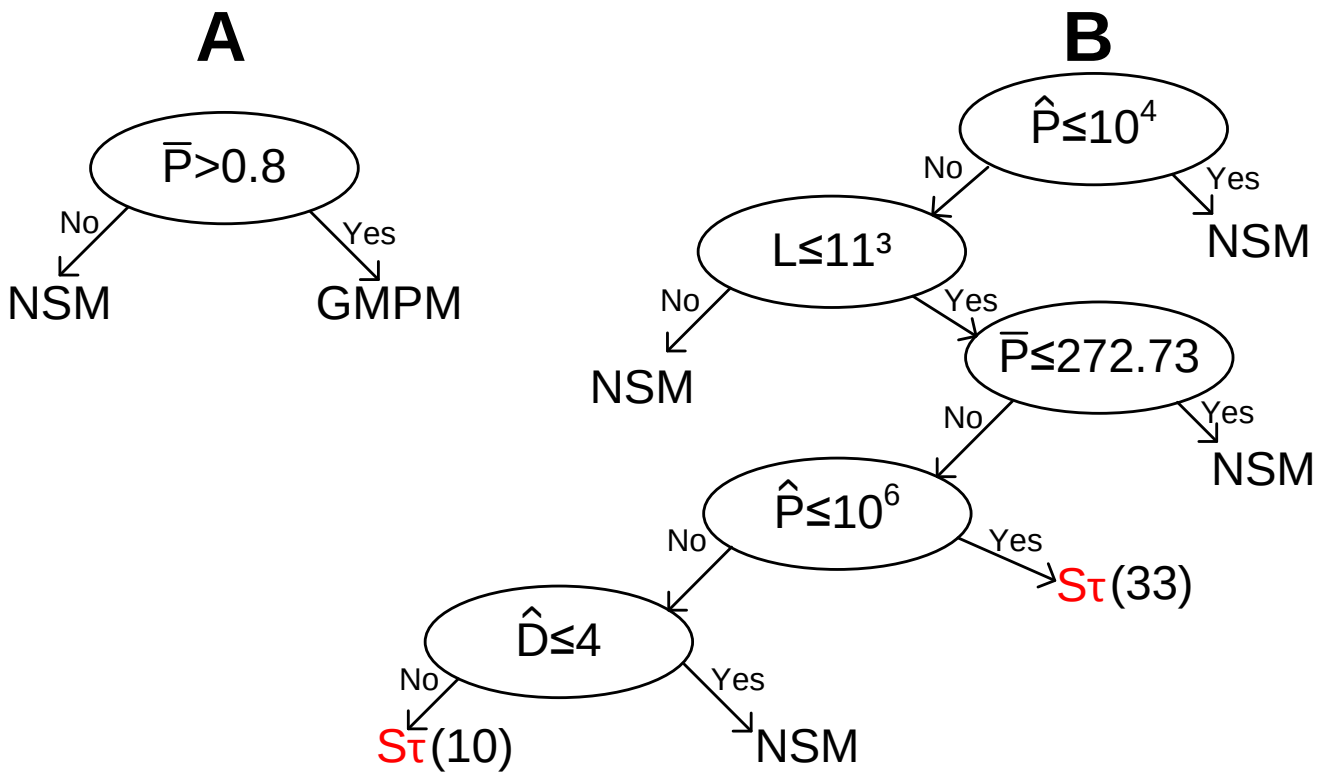
- Portfolio theory allows to *rank* algorithms $\Rightarrow$ implicit quality measure
- Combines set of algorithm to improve overall performance
- **Results: additional speedup of 3**

Perf. Data & Problem Features: Performance Prediction



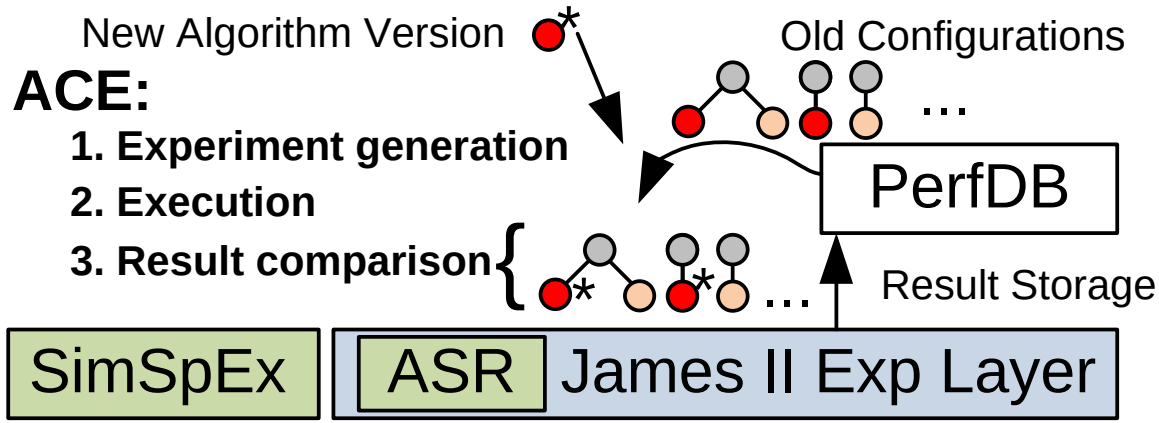
- Inductive learning (e.g., decision trees)
- Auxiliary components help quality management:
  - Performance database: stores past performance data
  - Enhanced registry: stores life cycle state of plug-ins (e.g., broken, stable, etc.)

Example: Decision Trees Generated for S_τ



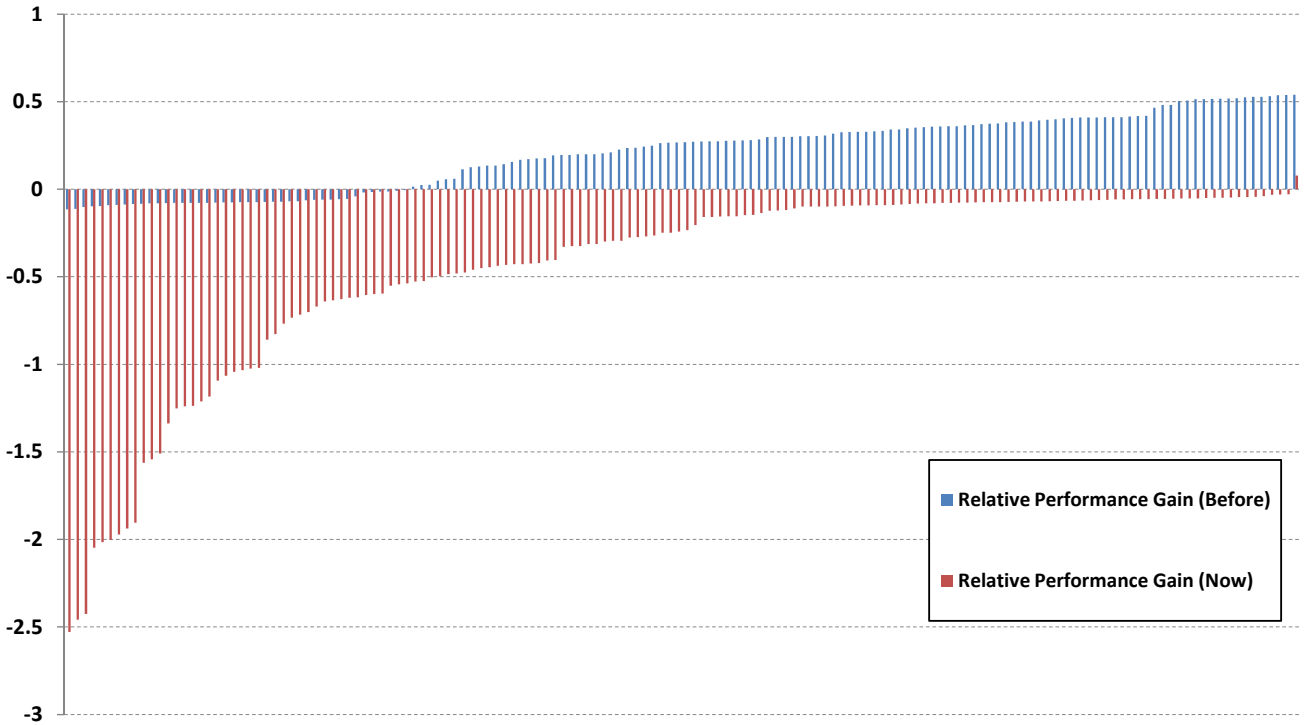
Obtainable speed-up: **several orders of magnitude**.

Coping with Change: Algorithmic Change Evaluator



- Hard to test: What is the *overall* performance impact of a code change?
- Requires Simulation Space Exploration components (SimSpEx)
- ACE allows a *fast* evaluation of changes because:
  - Adaptive replication (ASR) keeps focus on *fastest algorithms*
  - Data from PerfDB *steers experiments*

Sample Results: a bad code change



Idea: compare *best* (here: fastest) simulation setup that contains the changed algorithm, with *best* setup that does *not* contain the changed algorithm — before (from PerfDB) and after the code change, on a *set* of benchmark models (x-axis).

Quality of M& S: Evaluating Algorithms

- Quality of experiments: right algorithms
  alternatives + selection of suitable algorithms
- M & S software quality : “Bug free software”
  alternatives + support for evaluation and report of life cycle
- M & S software quality: Efficiency of the software
  alternatives + selection of efficient algorithms

Quality of M& S: Evaluating Algorithms

- Quality of experiments: right algorithms
  alternatives + selection of suitable algorithms
- M & S software quality : “Bug free software”
  alternatives + support for evaluation and report of life cycle
- M & S software quality: Efficiency of the software
  alternatives + selection of efficient algorithms

Potential for Standardization

- Software component interfaces (Reuse)
- Benchmark models (Repeatability)
- Experiment descriptions (Repeatability)
- Data storage formats (Usability)
- Knowledge for algorithm selection (Usability)
- Knowledge for modeling formalisms selection (Usability)

Thank you for your attention.

Questions?