Information and Communication Technologies Institute
Carnegie Mellon | PORTUGAL
AN INTERNATIONAL PARTNERSHIP

Architecting Resilience: Handling Malicious and Accidental Threats

Rogério de Lemos
University of Kent, UK

Javier Cámara
CMU, USA

Chris Bailey
University of Kent, UK

Carlos Eduardo da Silva
UFRN, Brazil

EASSy 2013
NII Shonan Village Centre. April 2013

ADAAS | ASSURING DEPENDABILITY IN ARCHITECTURE-BASED ADAPTIVE SYSTEMS

Context

- **Resilience** according to *Laprie*
  - persistence of service delivery that can justifiably be trusted, when facing changes
  - attaining dependability or security considering change
    - accidental and malicious threats

- "**Architecting**" according to *Rechting*
  - the process of creating and building architectures
    - the architect influences the whole development process
  - the art and science of creating and building complex systems
    - scope, structure and certification

Resilience: Justification of Trust

- Related to **provision of assurances**
- Based on building **arguments** about system resilience
- Development- and run-time **evidence**
  - collection
    - relies on verification and validation techniques, rigorous design, etc.
      - e.g., model checking, sat solvers, testing, etc.
  - structuring
    - in the from of resilience cases (dependability or safety cases)
  - analysis
    - build the arguments

Outline of the Talk

- **Integration Testing**
  - generation of plans for managing testing
- **Evaluating Resilience**
  - controller and system through stimulation of changeload
- **Self-Adaptive Authorisation Infrastructures**
  - insider threats

Let's have break… Let's be reflective…

- ***Does it make sense to test self-adaptive system at run-time?***
  - *If positive, what would be needed?*
- ***How effective is empirical evidence when evaluating the resilience of self-adaptive systems?***
  - *Should it be done at development-time or run-time?*
- ***How useful is self-adaptation in dealing with insider threats?***
  - *What should be monitored, analysed and controlled?*

Information and Communication Technologies Institute
Carnegie Mellon | PORTUGAL
AN INTERNATIONAL PARTNERSHIP

EASSy 2013
Architecting Resilience: Handling Malicious and Accidental Threats

Part 1: Integration Testing

ADAAS | ASSURING DEPENDABILITY IN ARCHITECTURE-BASED ADAPTIVE SYSTEMS

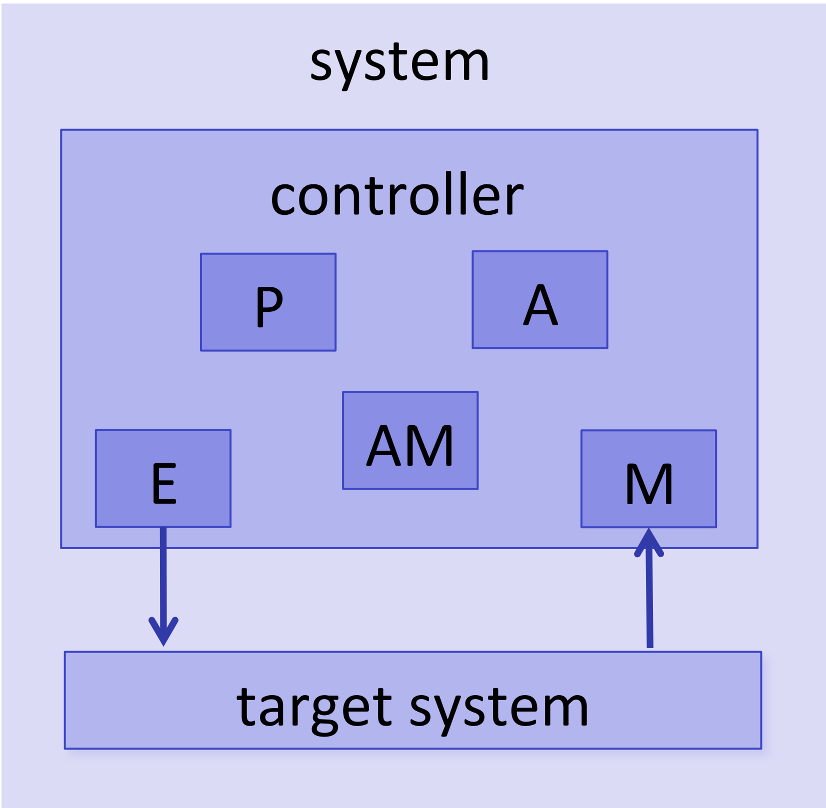
Integration Testing in Self-adaptive Systems

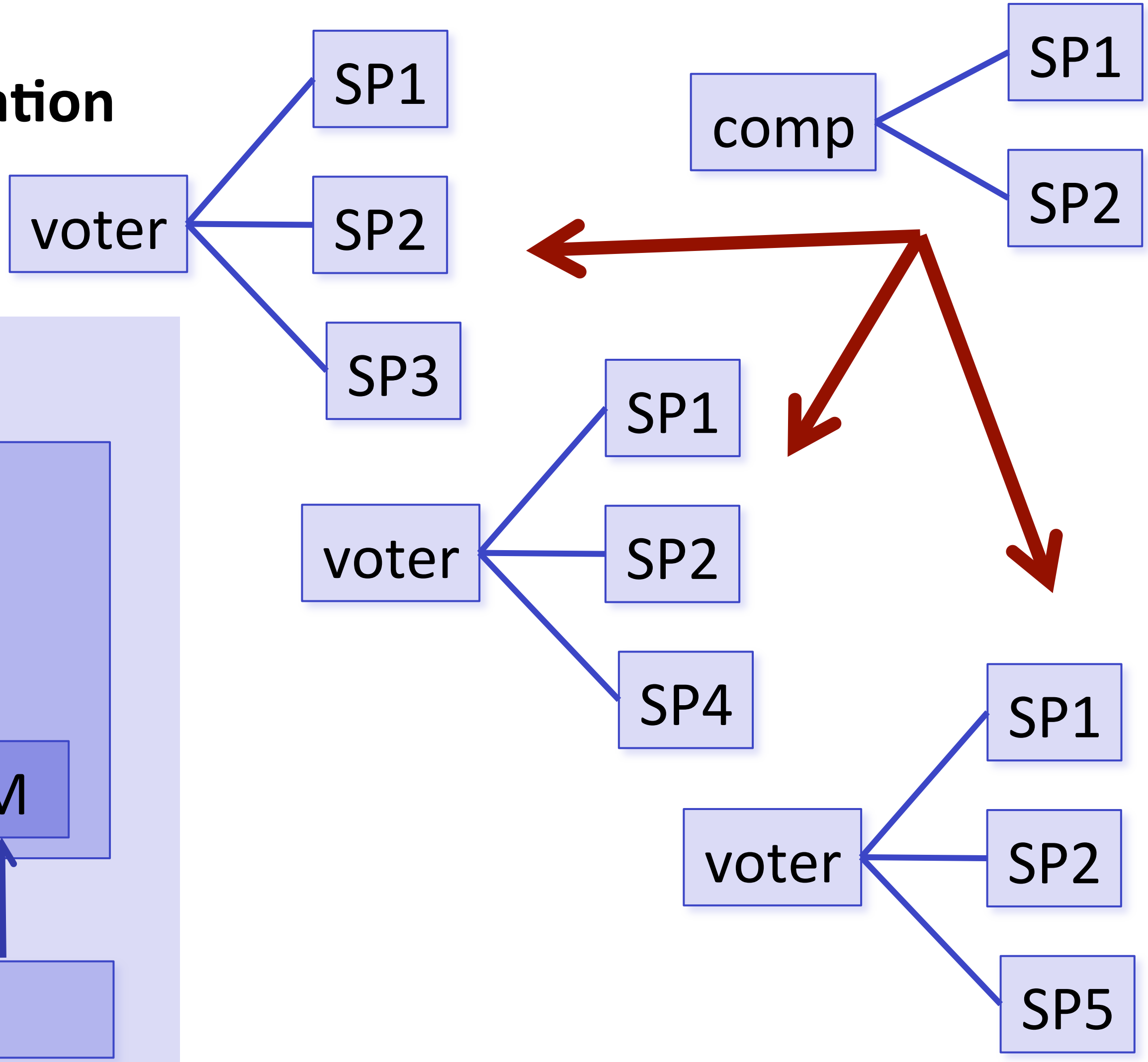
- **During architectural reconfiguration software testing is usually neglected**
  - challenging to test all possible architectural configurations at development-time
  - new components allow configurations not envisioned at development-time

- **Uncertainty and variability affects adaptation process**
  - changing goals, unexpected resource conditions, and unpredictable environments

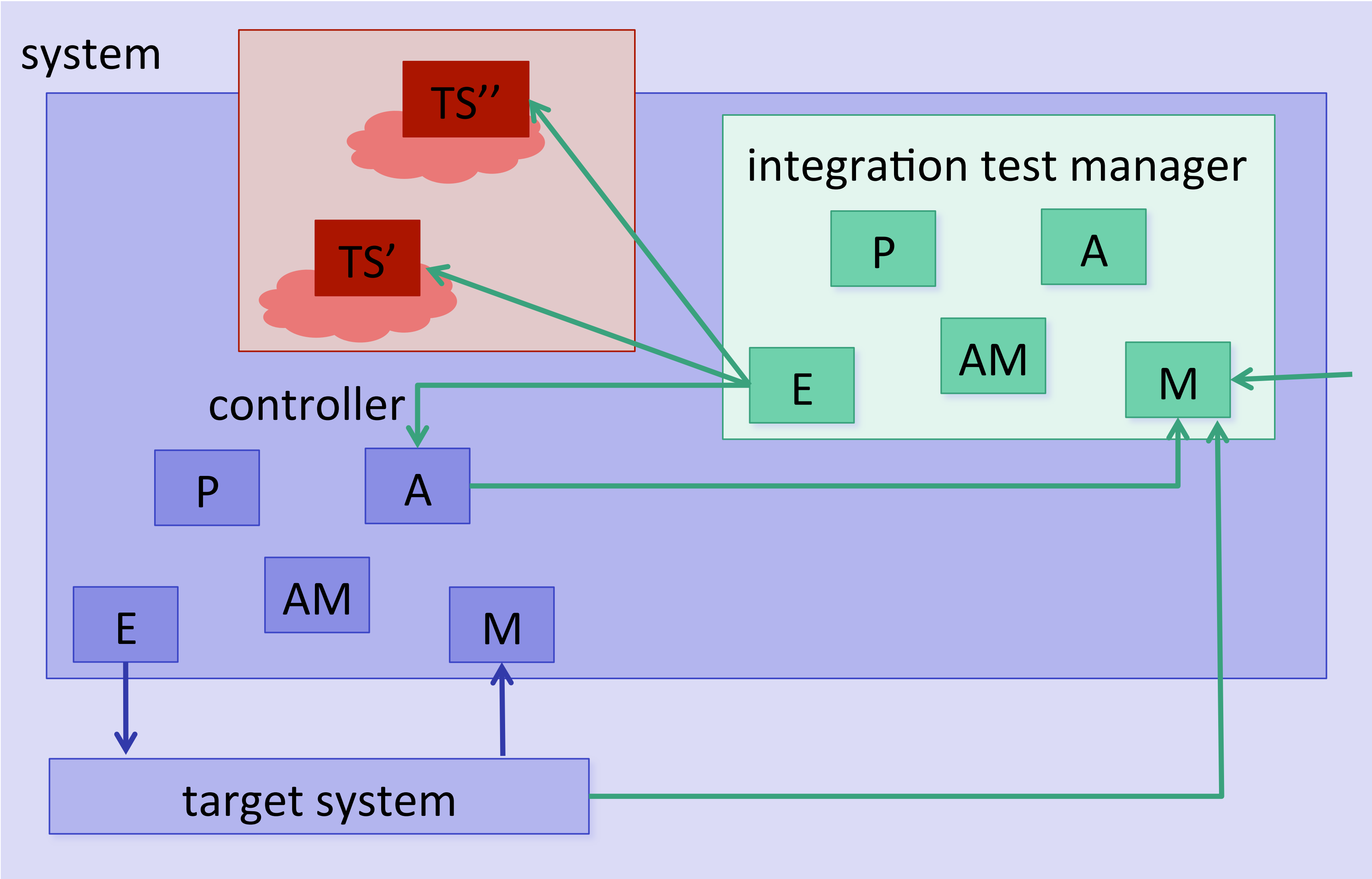
Integration Testing in Self-adaptive Systems

Architectural reconfiguration





Integration Testing in Self-adaptive Systems

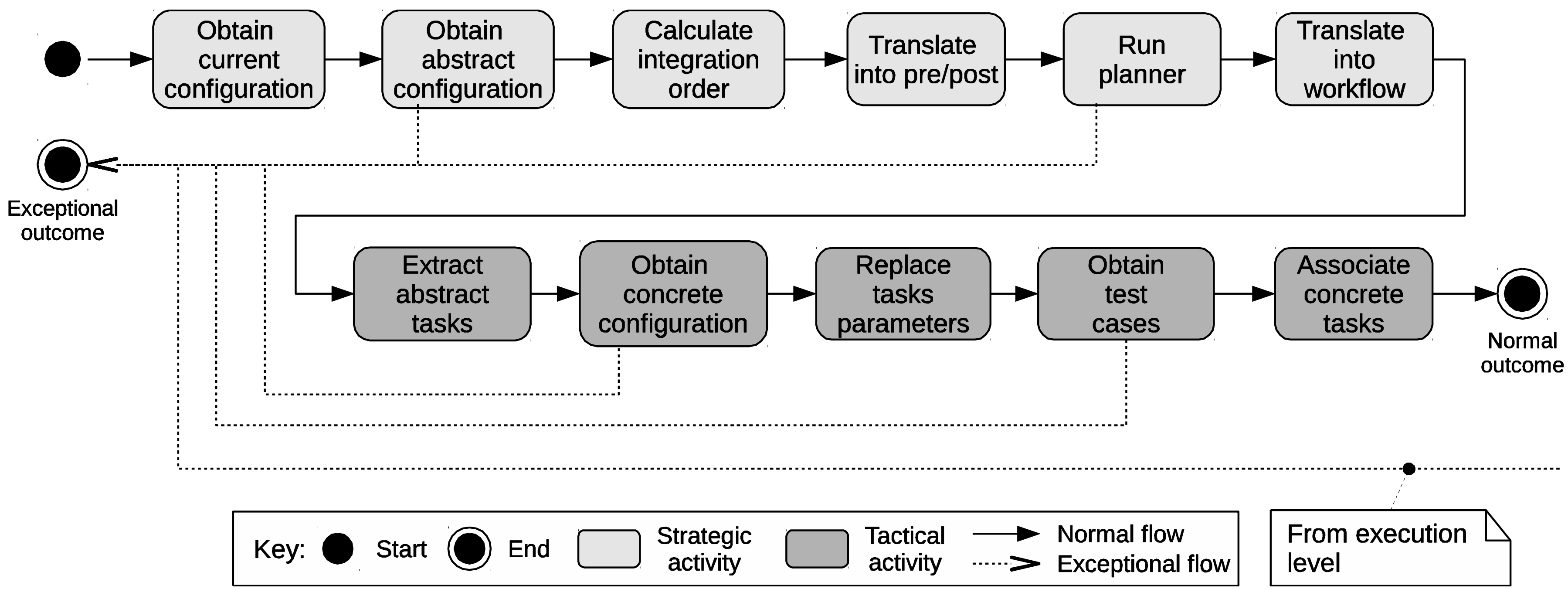


Integration Testing in Self-adaptive Systems

- **A framework and tool support for the dynamic generation of plans for integration testing [SEAMS 2011]**
  - workflows
    - represent and execute the plan
  - AI planning
    - AI planner generates a sequence of tasks to achieve a goal
  - MDE transformations
    - translate domain specific models into planning problems

Integration Testing in Self-adaptive Systems

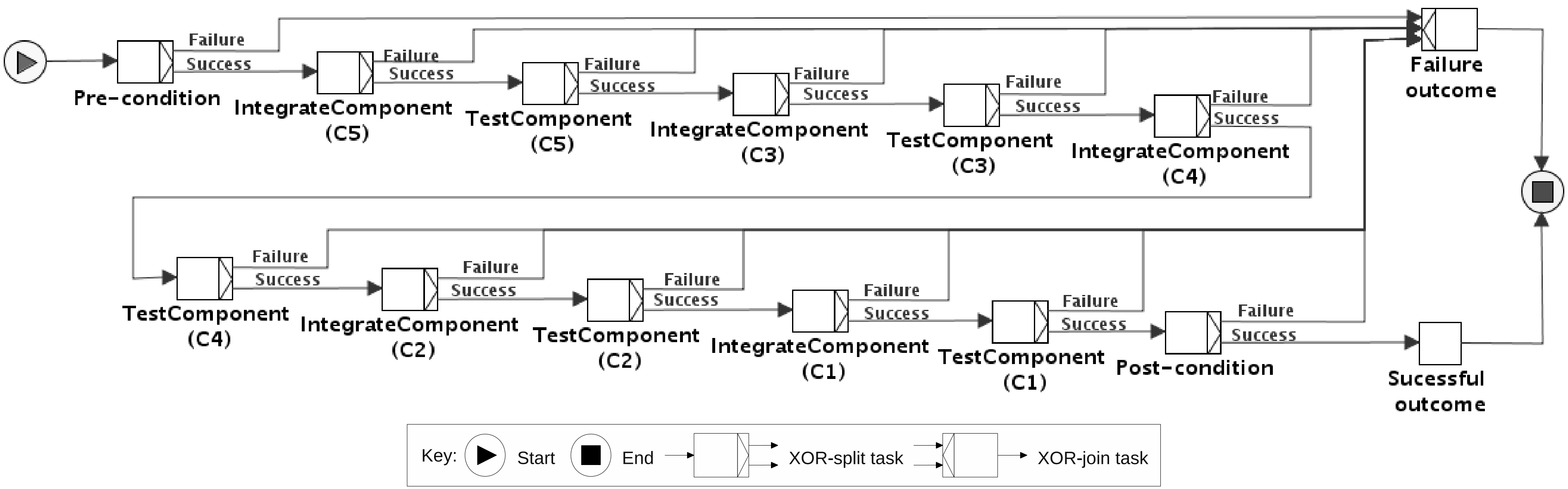
- **Process for generating dynamic plans**



Information and Communication Technologies Institute
Carnegie Mellon | PORTUGAL

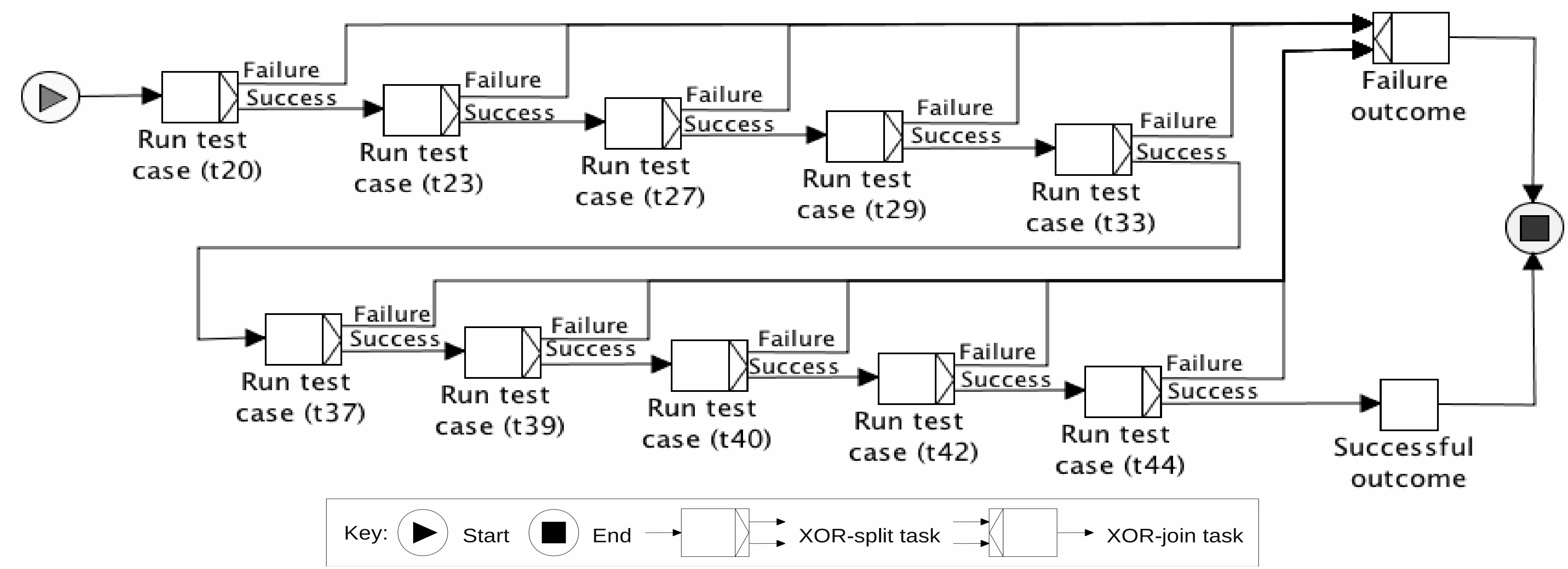
Integration Testing in Self-adaptive Systems

- **Generated workflow with several sub-workflows**
  - IntegrateComponent: reconfiguration
  - TestComponent: test cases



Integration Testing in Self-adaptive Systems

- **Example of sub-workflow**
  - TestComponent: test cases selected based on goals



Information and Communication Technologies Institute
Carnegie Mellon | PORTUGAL
AN INTERNATIONAL PARTNERSHIP

EASSy 2013
Architecting Resilience: Handling Malicious and Accidental Threats

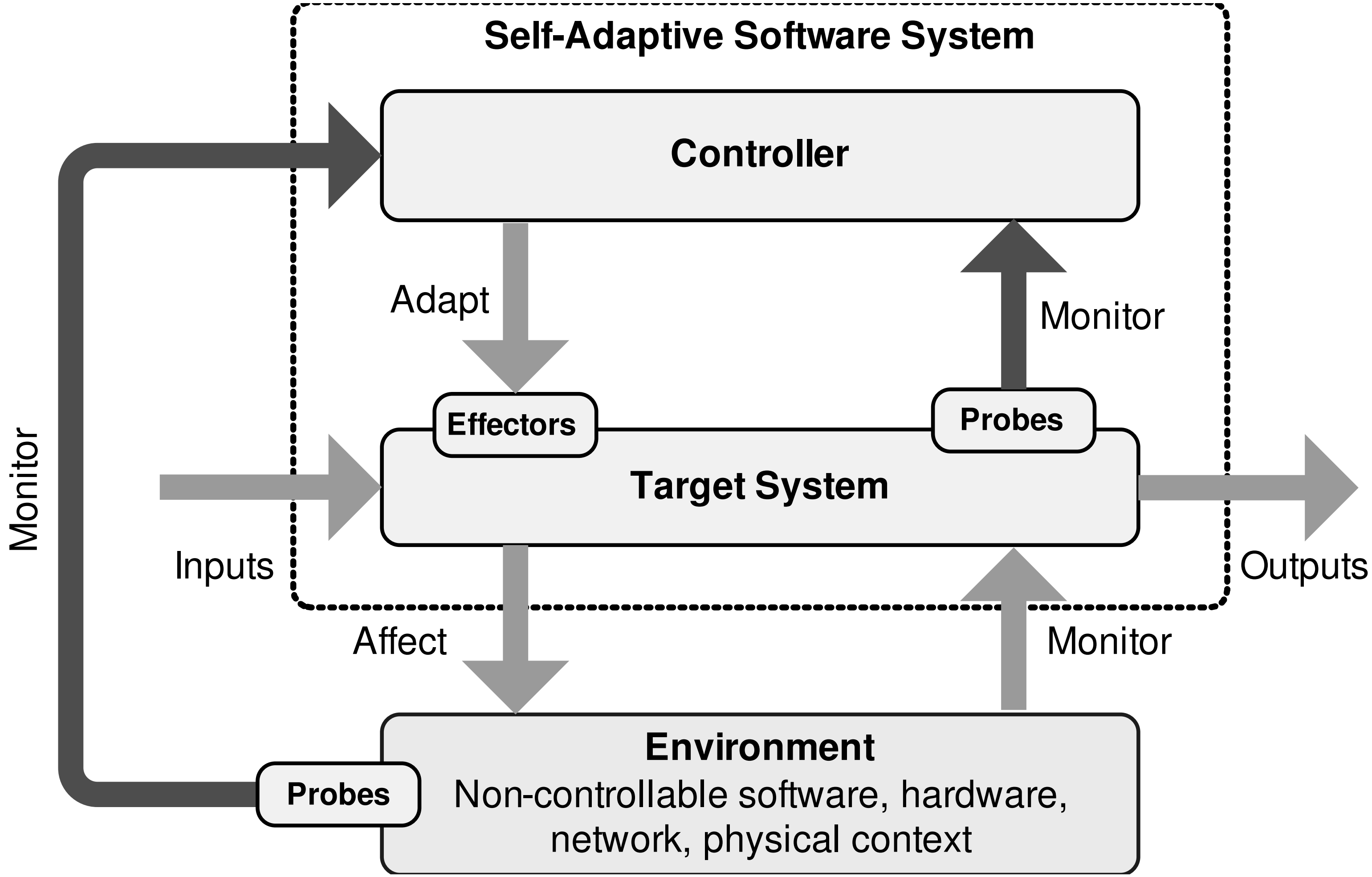
Part 2: Evaluating Resilience

ADAAS | ASSURING DEPENDABILITY IN ARCHITECTURE-BASED ADAPTIVE SYSTEMS

Evaluating Resilience

- **Stepwise progress for the provision of assurances about the resilience of self-adaptive software systems**
  - Resilience evaluation based on environmental stimuli **[SEAMS 2012]**
    - probabilistic model-checking for obtaining levels of confidence
  - Resilience evaluation by comparing adaptation mechanisms of self-adaptive software systems **[Computing 2013]**
    - framework and changeload
  - Robustness evaluation of controllers **[LADC 2013]**
    - injection of faults for evaluating Rainbow
  - Effectiveness of architecture-based self-adaptation **[SEAMS 2013]**
    - effort in evolving industrial middleware
  - Robustness-driven resilience evaluation of self-adaptive software systems
    - evaluating system properties by injecting faults

Self-Adaptive Software System – Our Model



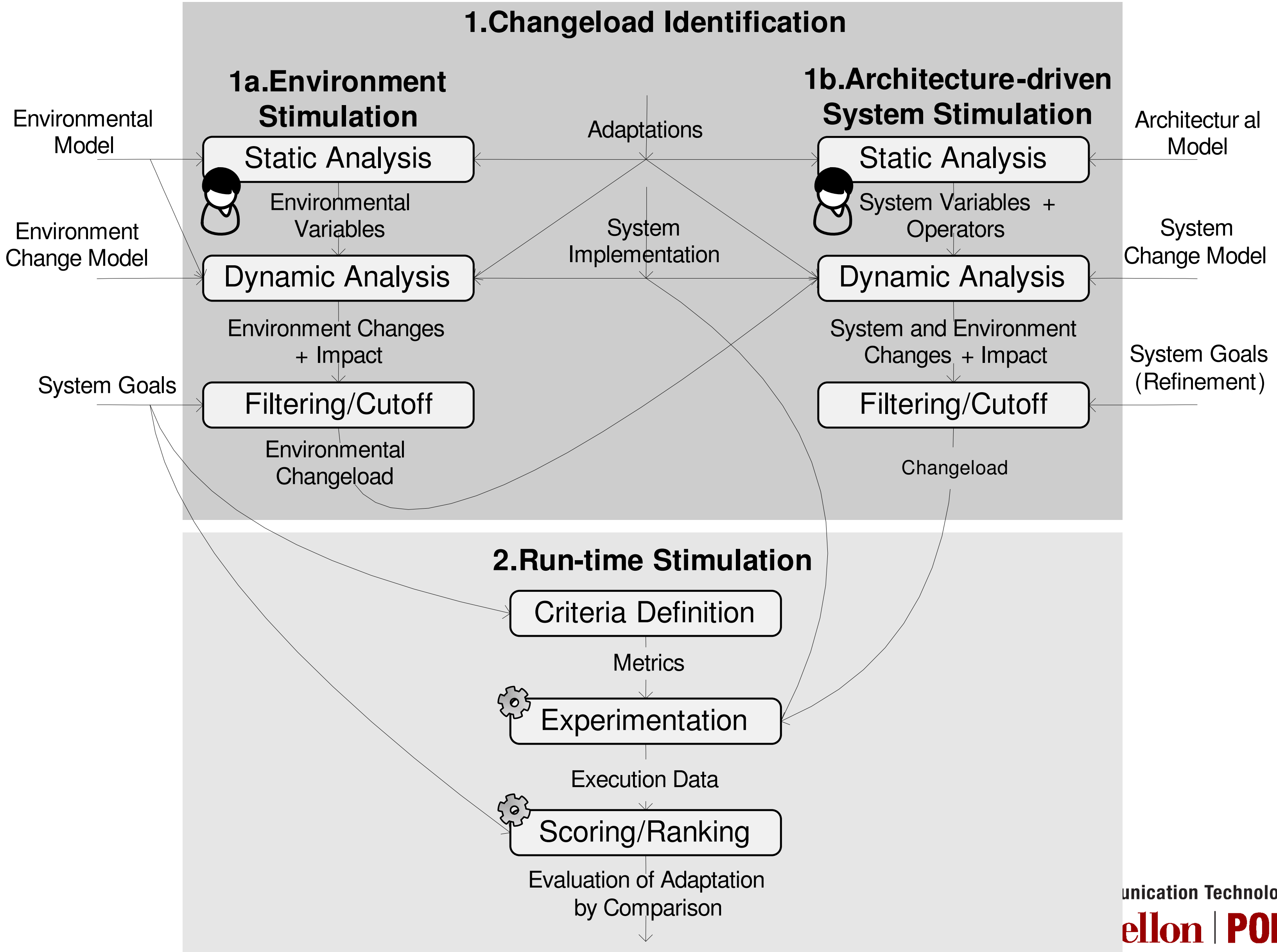
Evaluating Resilience

- **Stepwise progress for the provision of assurances about the resilience of self-adaptive software systems**
  - Resilience evaluation based on environmental stimuli **[SEAMS 2012]**
    - probabilistic model-checking for obtaining levels of confidence
  - Resilience evaluation by comparing adaptation mechanisms of self-adaptive software systems **[Computing 2013]**
    - framework and changeload
  - Robustness evaluation of controllers **[LADC 2013]**
    - injection of faults for evaluating Rainbow
  - Effectiveness of architecture-based self-adaptation **[SEAMS 2013]**
    - effort in evolving industrial middleware
  - Robustness-driven resilience evaluation of self-adaptive software systems
    - evaluating system properties by injecting faults

Changeload – Basis for Evaluation

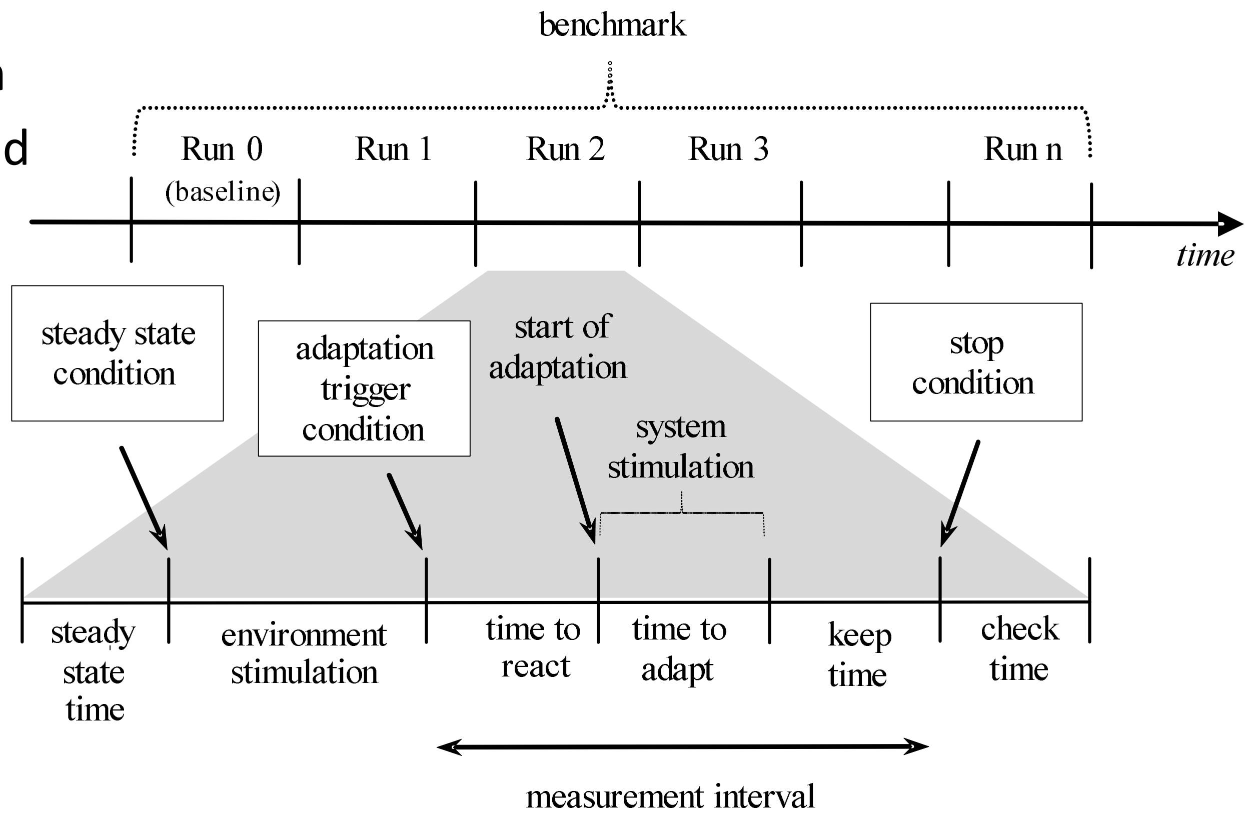
- A **changeload** is a set of representative change scenarios
  - specification to stimulate system and environment
- A **change scenario**
  - **system state**
  - **environment state** under which tasks are performed
    - including hardware/software resource conditions needed to provide service
  - set of **system goals**
  - set of **changes** in
    - system
    - environment
    - system goals
  - an implicit time frame

Framework Overview



Run-Time System Stimulation

- **Goal:** Exercising the different adaptation alternatives in the running system to obtain evidence for comparison
- **Three stages**
  - **criteria definition**
    - metrics defined according to **system goals**
  - **experimentation**
  - **scoring/Ranking**
    - evaluation by comparison



Evaluating Resilience

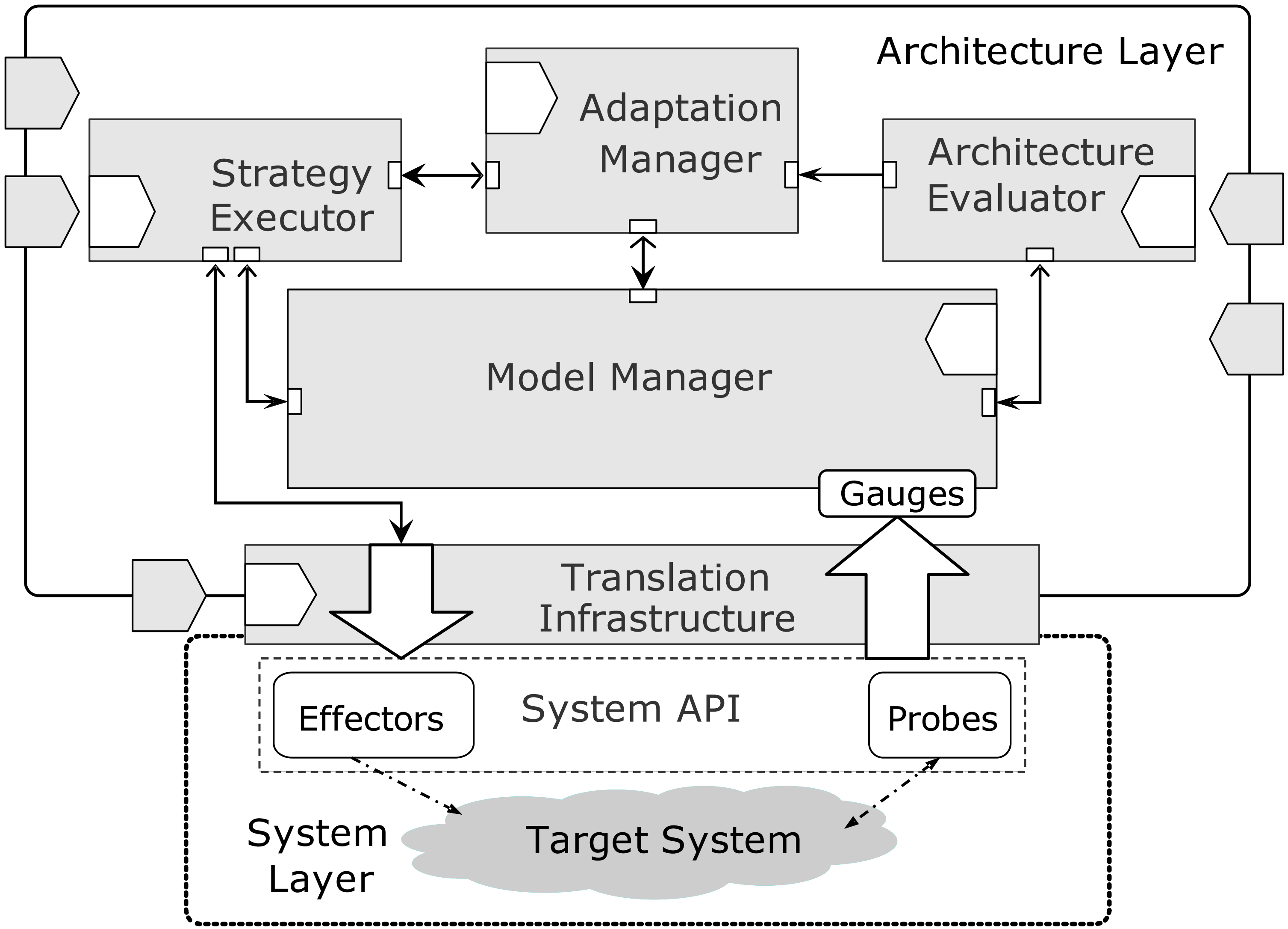
- **Stepwise progress for the provision of assurances about the resilience of self-adaptive software systems**
  - Resilience evaluation based on environmental stimuli **[SEAMS 2012]**
    - probabilistic model-checking for obtaining levels of confidence
  - Resilience evaluation by comparing adaptation mechanisms of self-adaptive software systems **[Computing 2013]**
    - framework and changeload
  - Robustness evaluation of controllers **[LADC 2013]**
    - injection of faults for evaluating Rainbow
  - Effectiveness of architecture-based self-adaptation **[SEAMS 2013]**
    - effort in evolving industrial middleware
  - Robustness-driven resilience evaluation of self-adaptive software systems
    - evaluating system properties by injecting faults

Robustness Tests

- Based on mutation of input received from probes
- General structure of controller input coming from probes
  - Name of variable being updated
  - New variable value
  - Timestamp providing temporal context
- Rainbow: messages encoded as text strings sent back to the controller

[timestamp] variable name : variable value

The Rainbow Framework



Controller Failure Mode Classification

Adapted version of the CRASH* Scale

- **Catastrophic:** whole controller crashes or becomes corrupted
  - Might include the OS or machine on which the controller runs
- **Restart:** controller hangs and needs external reboot
  - Within worst case execution time of adaptation cycle
- **Abort:** abnormal behavior due to an exception raised at run-time in the controller
- **Silent:** controller fails to acknowledge an error
- **Hindering:** Controller fails to return correct error code

* P. Koopman and J. DeVale. Comparing the robustness of posix operating systems. In Proceedings of the Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing, FTCS '99, Washington, DC, USA, 1999. IEEE Computer Society.

Robustness Tests - Mutation Rules

Type	Rule Name	Description
Message	MsgNull	Replace by null value
	MsgEmpty	Replace by empty string
	MsgPredefined	Replace by predefined string
	MsgNonPrintable	Replace by string with non-printable characters
	MsgAddNonPrintable	Add non-printable characters to the string
	MsgOverflow	Add characters to overflow max string size
Timestamp	TSEmpty	Replace by empty timestamp
	TSRemove	Remove timestamp from response
	TSInvalidFormat	Replace by timestamp with invalid format
	TSDateMaxRange	Replace date in timestamp by maximum valid
	TSDateMinRange	Replace date in timestamp by minimum valid
	TSDateMaxRangePlusOne	Replace date in timestamp by maximum valid plus one
	TSDateMinRangeMinusOne	Replace date in timestamp by minimum valid minus one
	TSDateAdd100	Add 100 years to date in timestamp
	TSDateSubtract100	Subtract 100 years from date in timestamp
	TSInvalidDate	Replace date in timestamp by invalid date (*e.g.*, 2/29/1984)
Variable Name	VNRemove	Remove variable name
	VNSwap	Replace by different valid variable name of same type
	VNSwapType	Replace by different valid variable name of different type
	VNInvalidFormat	Replace by variable name with invalid format
	VNNotExist	Replace by non-existing variable name

Type	Rule Name	Description
Variable Value	VVRemove	Remove variable value
	VVInvalidFormat	Replace value by one with invalid format
	Number	
	VVNumAbsoluteMinusOne	Replace by -1
	VVNumAbsoluteOne	Replace by 1
	VVNumAbsoluteZero	Replace by 0
	VVNumAddOne	Add 1
	VVNumSubtractOne	Subtract 1
	VVNumMax	Replace by maximum number valid for type
	VVNumMin	Replace by minimum number valid for type
	VVNumMaxPlusOne	Replace by maximum number valid for type plus one
	VVNumMinMinusOne	Replace by minimum number valid for type minus one
	VVNumMaxRange	Replace by maximum number valid for variable
	VVNumMinRange	Replace by minimum number valid for variable
	VVNumMaxRangePlusOne	Replace by maximum number valid for variable plus one
	VVNumMinRangeMinusOne	Replace by minimum number valid for variable minus one
	Boolean	
	VVBoolPredefined	Replace by predefined value

Changeload Identification

1. Identify workload and operating conditions
   - Able to drive the controller through its different operational stages

2. Identify set of probes used through different controller stages

3. Identify set of applicable mutation rules
   - According to probes identified

4. Instantiate changes using mutation rules
   - Incorporated into change scenarios
   - One change scenario per mutation rule+controller stage

Experimental Evaluation

- 96 tests (32 applicable mutations X 3 controller operational stages)
- 48 tests (50%) uncovered robustness issues
- No Catastrophic, Restart, or Hindering issues detected
  - Limited observability
    - Build specific error detectors
  - Additional controller input
- Similar pattern of robustness issues identified at different controller stages
  - Justified by Rainbow's architecture
  - Some variations relevant when considering ensemble controller plus system
- Additional robustness issues raised outside of the controller

Mutation Rule	**Failures**					
	Analysis		**Planning**		**Execution**	
	A	S	A	S	A	S
MsgNull	1	1	1	1	1	1
MsgEmpty		1		1		1
MsgPredefined		1		1		1
MsgNonPrintable		1		1		1
MsgAddNonPrintable		1		1		1
TSEmpty		1		1		1
TSRemove		1		1		1
VNRemove		1		1		1
VVRemove		1		1		1
VVInvalidFormat		1		1		1
VVNumAbsoluteMinusOne		1		1		1
VVNumMax		1		1		1
VVNumMin		1		1		1
VVNumMaxPlusOne		1		1		1
VVNumMinMinusOne		1		1		1
VVNumMinRangeMinusOne		1		1		1
TOTAL	1	16	1	16	1	16

Evaluating Resilience

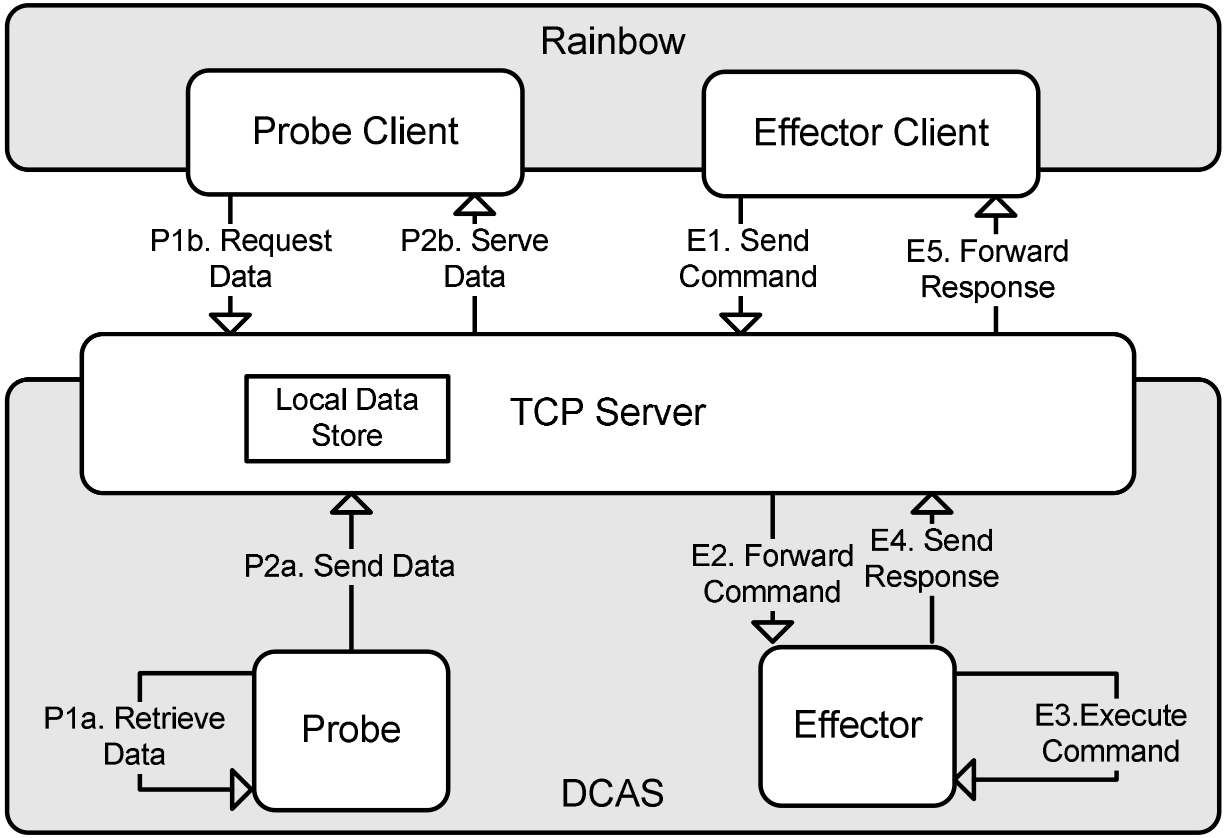
- **Stepwise progress for the provision of assurances about the resilience of self-adaptive software systems**
  - Resilience evaluation based on environmental stimuli **[SEAMS 2012]**
    - probabilistic model-checking for obtaining levels of confidence
  - Resilience evaluation by comparing adaptation mechanisms of self-adaptive software systems **[Computing 2013]**
    - framework and changeload
  - Robustness evaluation of controllers **[LADC 2013]**
    - injection of faults for evaluating Rainbow
  - Effectiveness of architecture-based self-adaptation **[SEAMS 2013]**
    - effort in evolving industrial middleware
  - Robustness-driven resilience evaluation of self-adaptive software systems
    - evaluating system properties by injecting faults

Integrating Rainbow and DCAS

Remove hardwired adaptation logic from DCAS, replacing it by external control exerted by Rainbow

- **Evolution of DCAS**
  - Remove built-in adaptation logic
  - Expose part of DCAS functionality
    - Public interface to probes and effectors
  - Enable Rainbow-DCAS communication
    - Lightweight TCP Server
- **Customizing the Rainbow Framework**
  - Model DCAS architecture
  - Implementing probes, effectors, gauges
  - Scripting adaptation

Rainbow-DCAS Translation Infrastructure



Evaluation – Implementation Effort

Rainbow Customization

Task	Time (h)	%
Architecture modeling	20	21.9
Implementing client probes and gauges	22	24.1
Implementing client effectors	12	13.1
Scripting adaptation (tactics and strategies)	35	38.4
Miscellaneous configurations	2	2.1
Total	**91**	**100**

DCAS Evolution

Task	Time (h)	%
Implementing TCP server	15	10.3
Identifying and removing built-in adaptation	40	27.5
Implementing probes	45	31
Implementing effectors	35	24.1
Miscellaneous configurations	10	6.8
Total	**145**	**100**

Evaluating Resilience

- **Stepwise progress for the provision of assurances about the resilience of self-adaptive software systems**
  - Resilience evaluation based on environmental stimuli **[SEAMS 2012]**
    - probabilistic model-checking for obtaining levels of confidence
  - Resilience evaluation by comparing adaptation mechanisms of self-adaptive software systems **[Computing 2013]**
    - framework and changeload
  - Robustness evaluation of controllers **[LADC 2013]**
    - injection of faults for evaluating Rainbow
  - Effectiveness of architecture-based self-adaptation **[SEAMS 2013]**
    - effort in evolving industrial middleware
  - Robustness-driven resilience evaluation of self-adaptive software systems
    - evaluating system properties by injecting faults

Information and Communication Technologies Institute
Carnegie Mellon | PORTUGAL
AN INTERNATIONAL PARTNERSHIP

EASSy 2013
Architecting Resilience: Handling Malicious and Accidental Threats

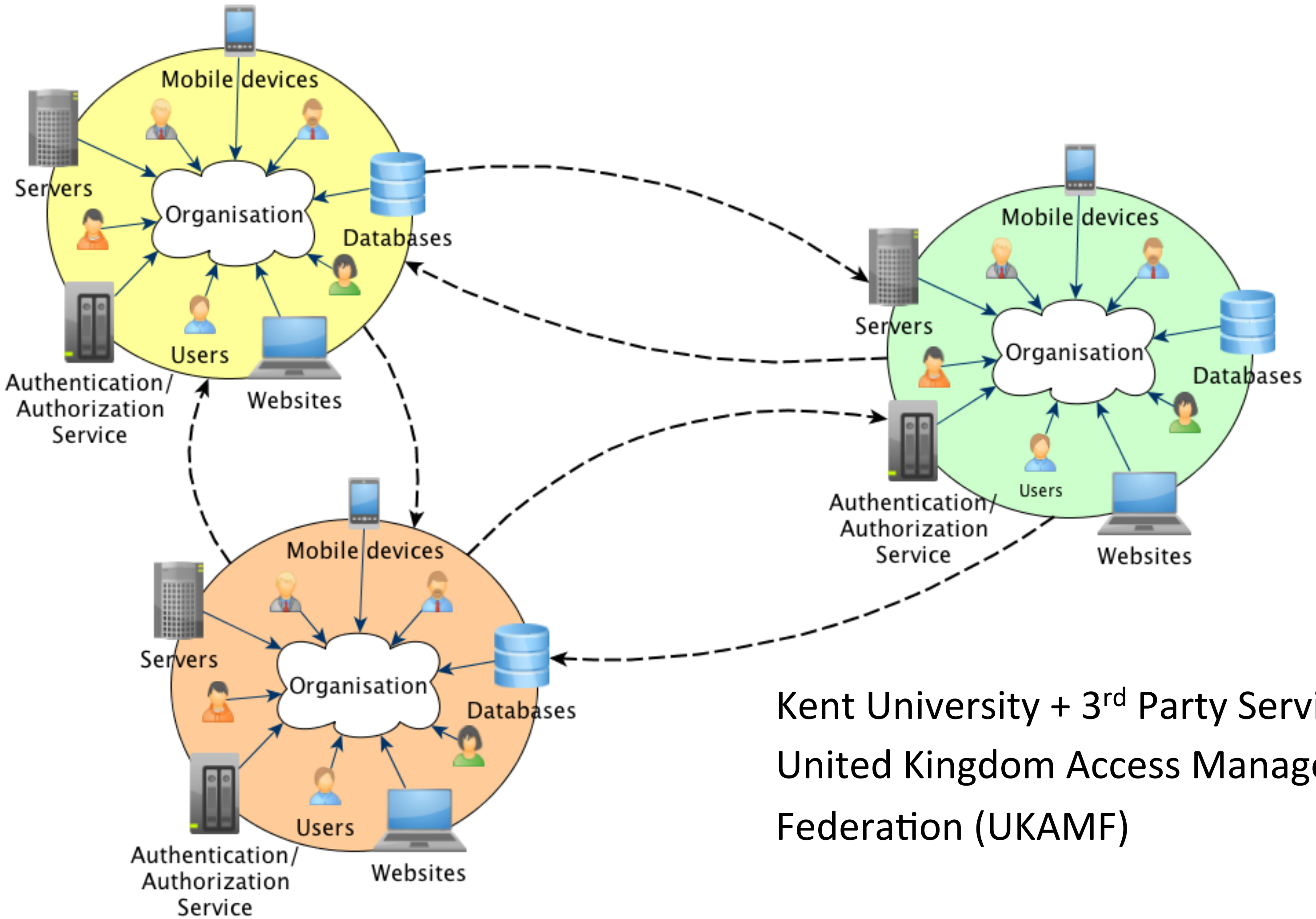
Part 3: Self-Adaptive Authorisation Infrastructures

ADAAS | ASSURING DEPENDABILITY IN ARCHITECTURE-BASED ADAPTIVE SYSTEMS

Self-Adaptive Authorisation Infrastructures

- **Authorization infrastructures protect, control and monitor access to electronic resources**
  - federated authorization infrastructures become increasingly difficult to manage
- **Malicious behavior** can be seen as abuse of access caused by **insider threat** within federated authorization infrastructures
- **Self-adaptation** is seen as a means to improve the management of malicious behavior, by adapting authorization policies and access rights
  - Adapt to mitigate malicious behavior, and prevent future attacks

Federated Authorization



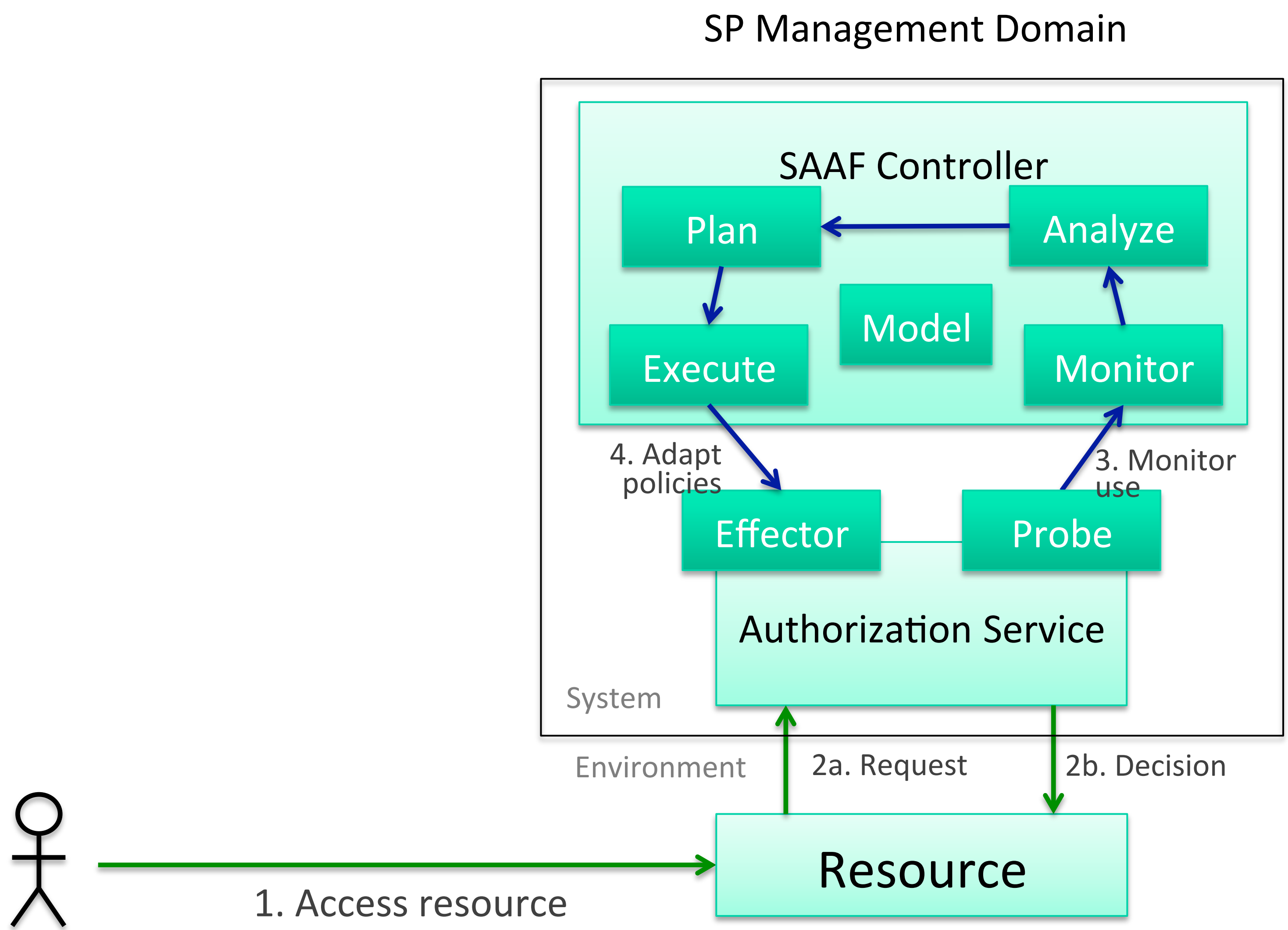
Kent University + 3rd Party Services

United Kingdom Access Management Federation (UKAMF)

Challenges of Managing Federations

- Heavily dependent on manual processes to identify and resolve malicious behaviour
- Involves multiple management domains
- Privacy protected users and large unknown user base
- Built on authorization infrastructures which are not typically designed to reflect on user behaviour

Self-Adaptive Authorization



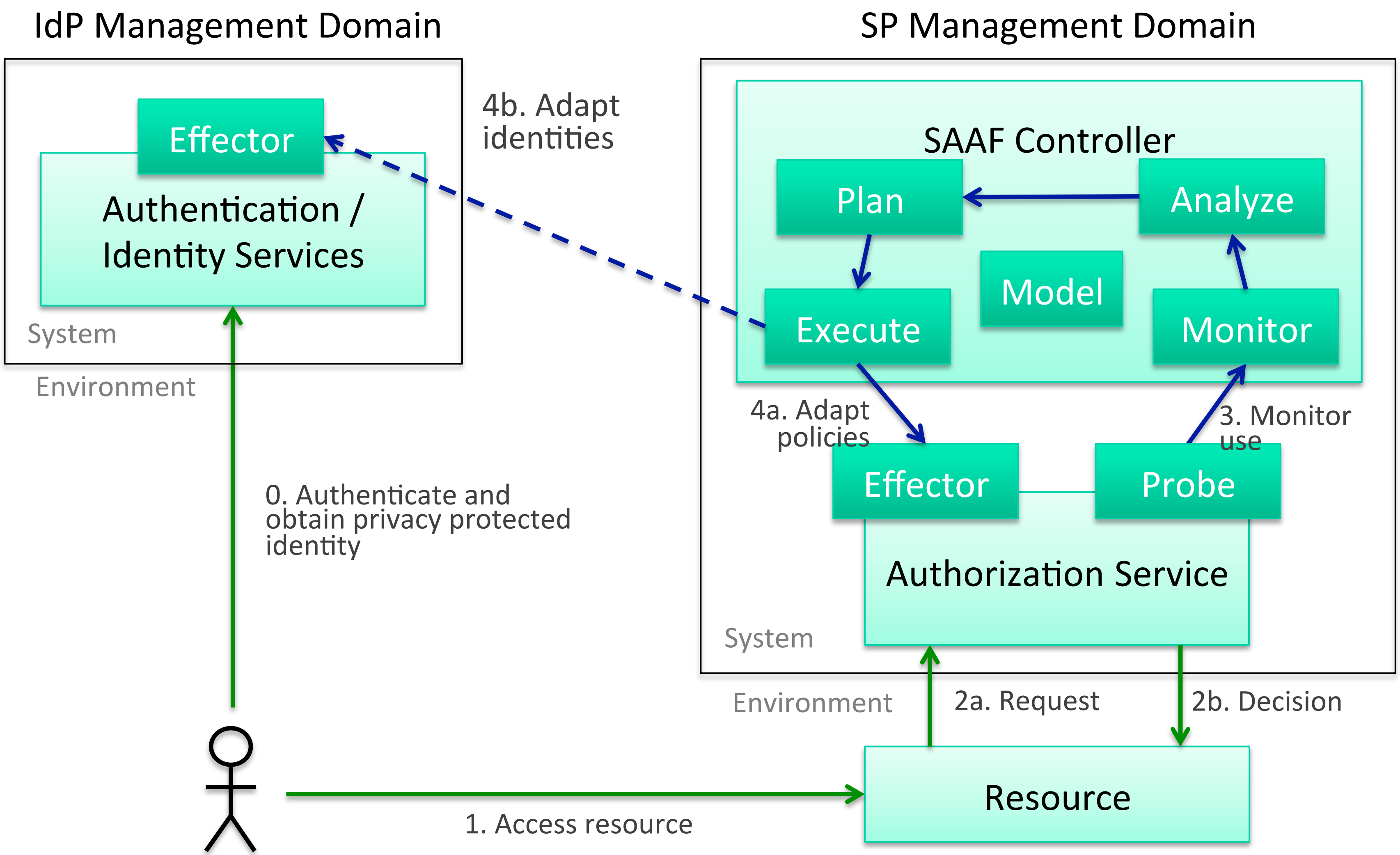
Cross-Domain Adaptation

- **Federations imply distributed authorization infrastructures with multiple management domains**

- **Identity Providers (IdPs) and Service Providers (SPs) have to work together to resolve malicious activity, else jeopardise trust**

- **Solution: Deploy an Identity Provider (IdP) Effector**
  - facilitates adaptation of IdP assets
  - allows IdP organisation to govern who can adapt and the extent of adaptation

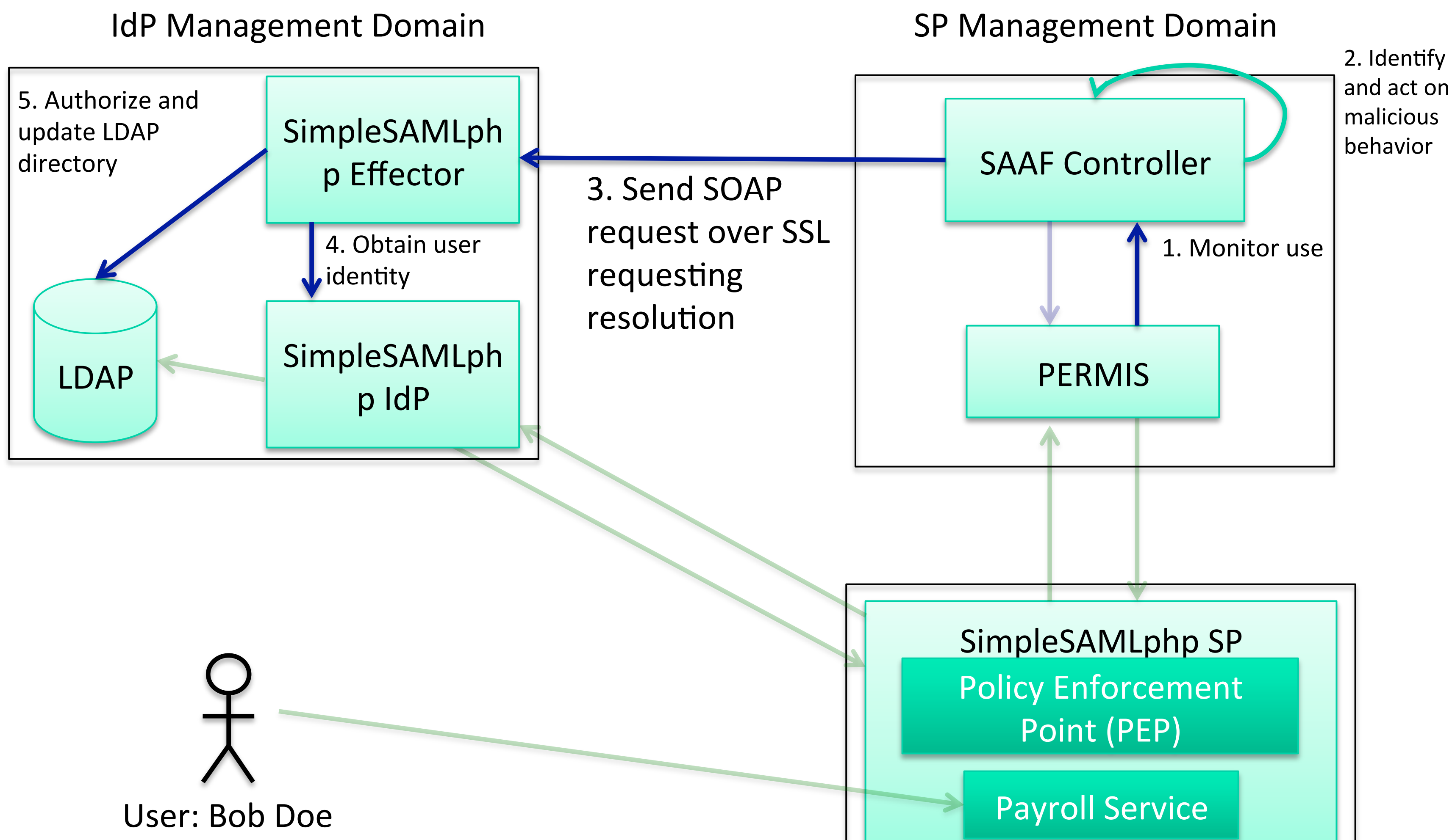
Self-Adaptive Federated Authorization



Simulating Malicious Behaviour

- **The SAAF controller classifies malicious behaviour with a behaviour rule**
  - No single subject should access the payroll service with a greater rate of 10 access requests per minute
- **User is allowed to access the payroll resource using his/her IdP identity assigned attribute 'permisRole=Contractor'**
- **User 'Bob Doe' breaks this rule by executing a high rate of requests to the payroll resource**

Self-Adaptation affecting IdP Service

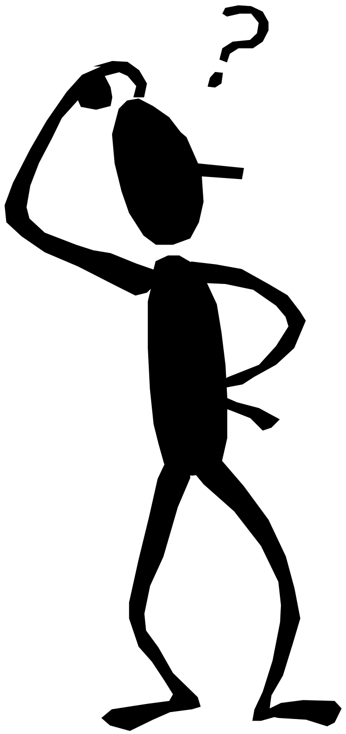


In Summary…

- **Progress on the resilience evaluation of self-adaptive systems**
  - evidence can be obtained by testing (run-time) or stimulations (development-time)
  - challenge is how to collect, structure and analyse evidence

- **Protection against insider attacks based on self-adaptation**
  - evaluation of SAAF through malicious changeload
  - look into more sophisticated ways of detecting malicious behaviour

Questions?



Thank you!

Thanks to

- Javier Camara
- Chris Bailey
- Carlos Eduardo da Silva