NICTA

Problem Detection and Diagnosis under Sporadic Operations and Changes

Liming Zhu, Research Group Leader, NICTA

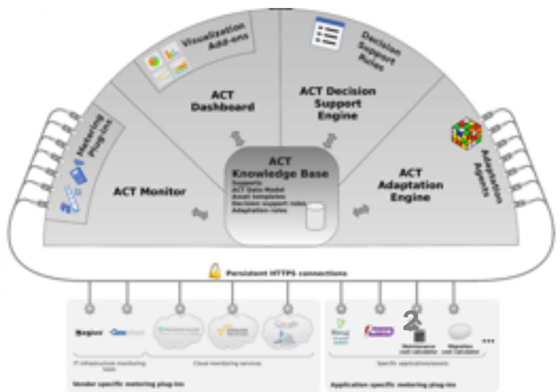
SPEC DevOps WG, 2015 May.

Australian Government

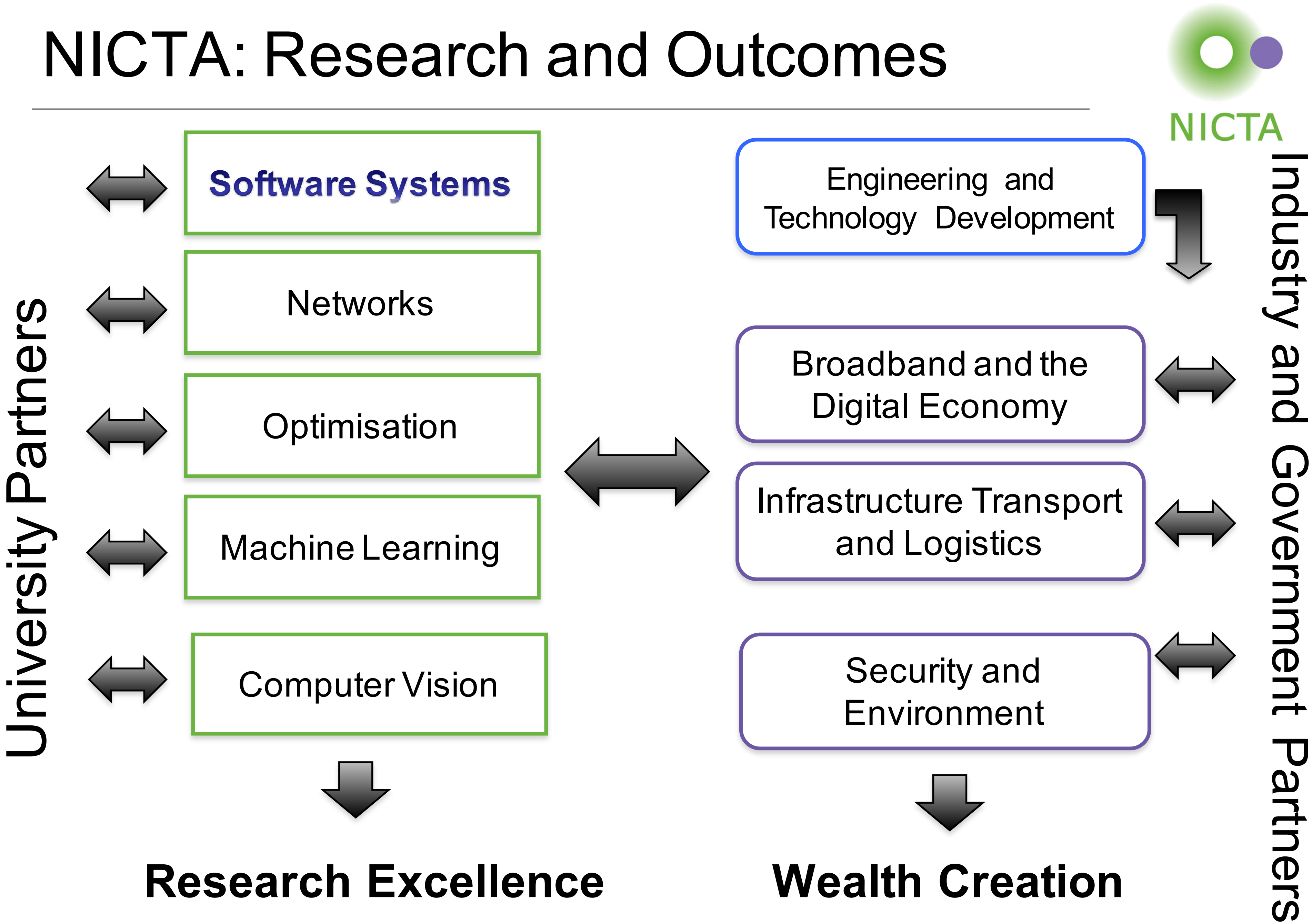
NICTA (National ICT Australia)

- Australia's National Centre of Excellence in Information and Communication Technology
- Five Research Labs:
  - ATP: Australian Technology Park, **Sydney**
  - NRL: UNSW, **Sydney**
  - CRL: ANU, **Canberra**
  - VRL: Various universities, **Melbourne**
  - QRL: 70 Bowen street, **Brisbane**

- 700 staff including 270 PhD students
- Budget: ~$90M/yr from Fed/State Gov and industry
- ~600 research papers/year, ~150 patents total



NICTA: Research and Outcomes



Software Systems Research Group

NICTA

- Vision: Cost Effective Dependable Systems
  - Single machine systems on trusted microkernel
  - **Distributed systems on untrusted platforms**
- Core area: dependable applications on cloud platform
  - Dependability = Performance, Reliability, Availability, Security
  - Focus: cloud consumers (not providers) and (sporadic) operations
    - Consumer: limited visibility (monitoring) and control
    - Operation: during deployment/upgrade/re-configuration time
  - Continuous deployment pipelines and DevOps
    - DevOps impact on performance, error detection and diagnosis
    - Design time analysis of operation processes using logs
    - Run time use of process contexts, logs and metrics for performance monitoring, error detection and diagnosis.

Motivation: System Operations Matter

NICTA

- Gartner predicts:
  - “Through 2015, **80% of outages** impacting mission-critical services will be caused by people and process issues, and more than 50% of those outages will be caused by **change / configuration / release integration** and hand-off issues.”*
- The case of Knight Capital – from Wikipedia:
  - The Knight Capital Group was an American global financial services firm […].[…] Knight was **the largest trader** in U.S. equities, with a market share of 17.3% on NYSE and 16.9% on NASDAQ.[2] The company agreed to be acquired by Getco LLC in Dec 2012 after an **trading error lost $460 million.**
  - This took 45 minutes and was an upgrade error
- Two empirical studies
  - Big data analytics application (Yuan, OSDI14) and cloud application (Gunawi, SoCC14) performance issues are largely caused by operational protocols

Challenges: Continuous Changes & Uncertainty

NICTA

- Significantly shorter release cycles
  - Continuous delivery/deployment: from months / scheduled downtime to hours / any time
    - Etsy.com: 25 full deployments per day at 10 commits per deploy
  - Baseline-based anomaly detection no longer works!
- Continuous changes
  - Multiple sporadic operations at all times
  - Scaling in/out, snapshot, migration, reconfiguration, (rolling) upgrade, cron-jobs, backup, recovery…
- Cloud uncertainty
  - Limited visibility/monitoring, indirect control, small-scale failures are the norm

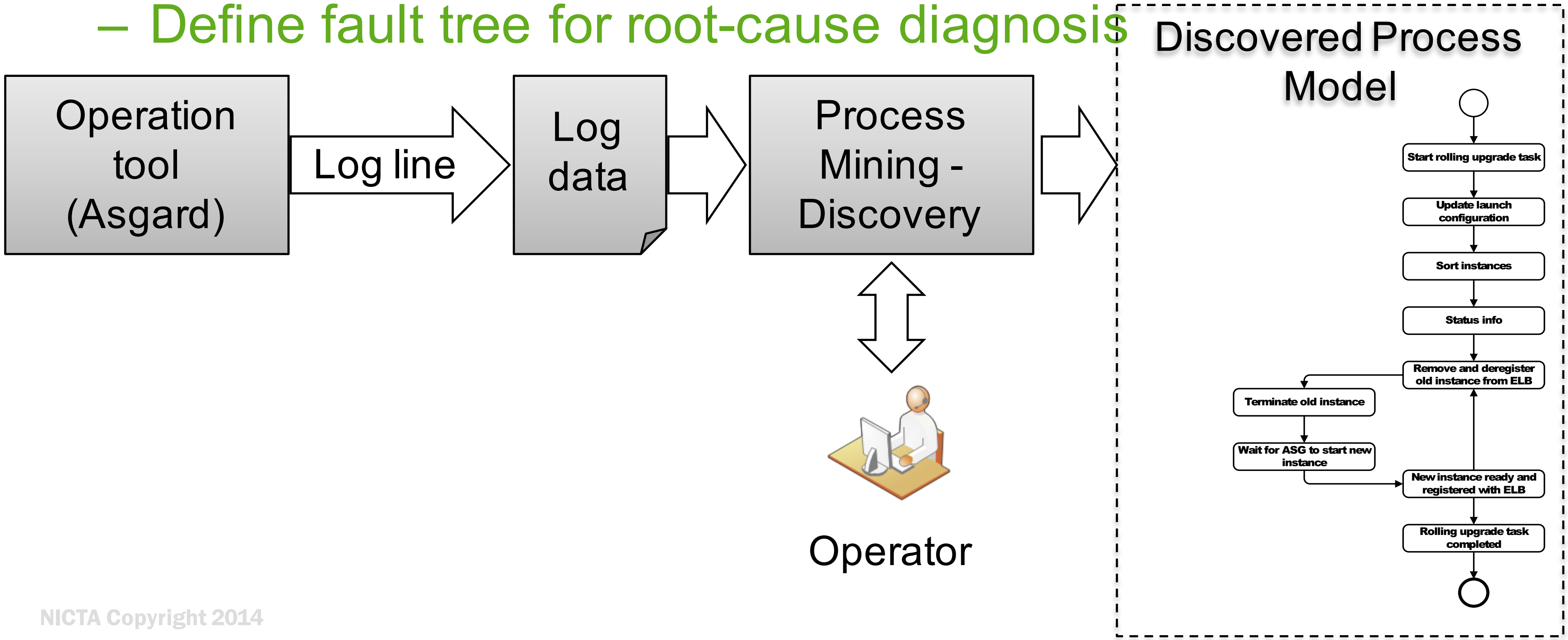
Our Approach – High-level View

NICTA

- Increasing dependability during Operation time e.g., through:
  - More accurate performance monitoring
  - Faster error detection
  - Fast or autonomous healing (quick fix)
  - Root cause diagnosis
  - Guided or autonomous recovery
- Incorporating change-related knowledge into system management
  - Knowledge about sporadic operation in **Process-Oriented Dependability** (POD): error detection and diagnosis using **process model & context**

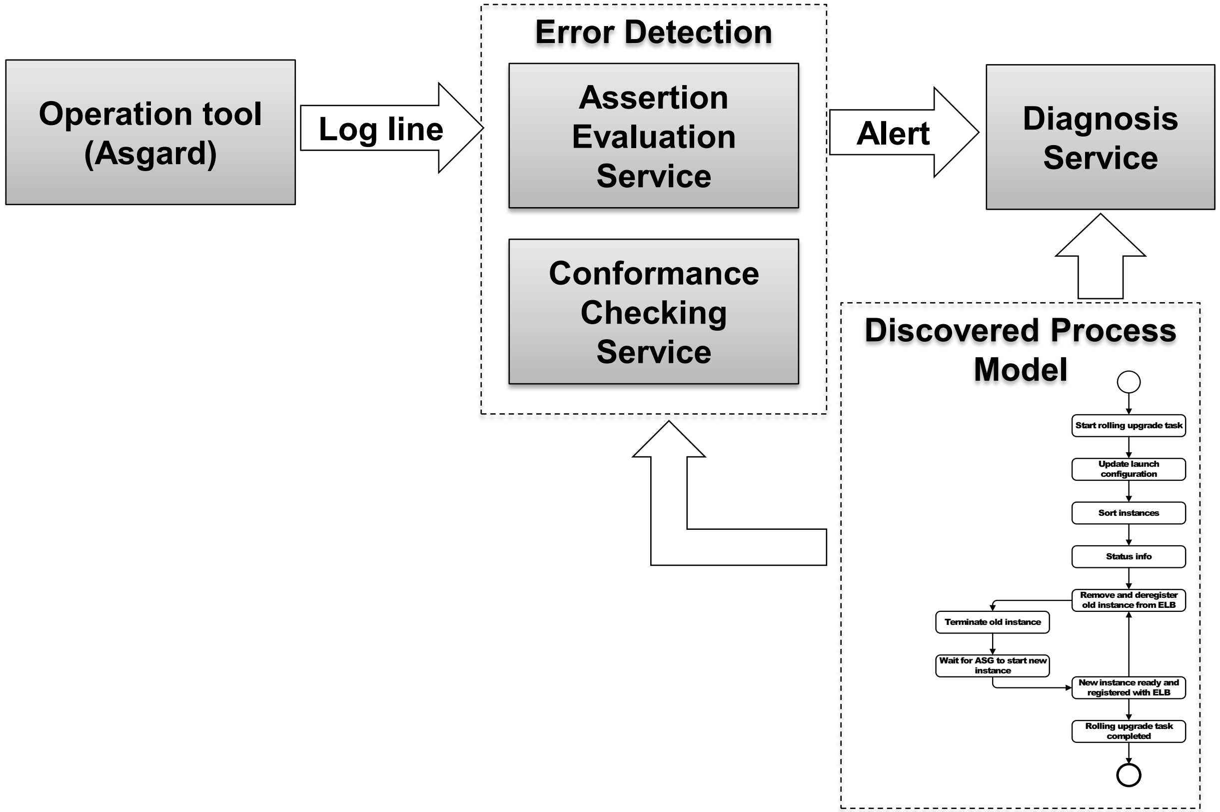
Our Approach: Use Process Context

- Offline: treat an operation as a *process*
  - Process discovery from logs/scripts
    - Clustering of log lines and process mining
  - Expected outcomes of steps specified as assertions
    - Including performance metrics-related assertions
  - Define fault tree for root-cause diagnosis



Our Approach: Use Process Context

- Online: use process context
  - Process context: process model/instance/step, expected states
  - Errors are detected by examining logs and monitoring data
    - Conformance checking against expected processes using logs
    - Assertions evaluation, directly or using **monitoring facilities**
  - Detected errors are further diagnosed for (root) causes
    - Examining a fault tree to locate potential root causes



Research Results Realized in Tools

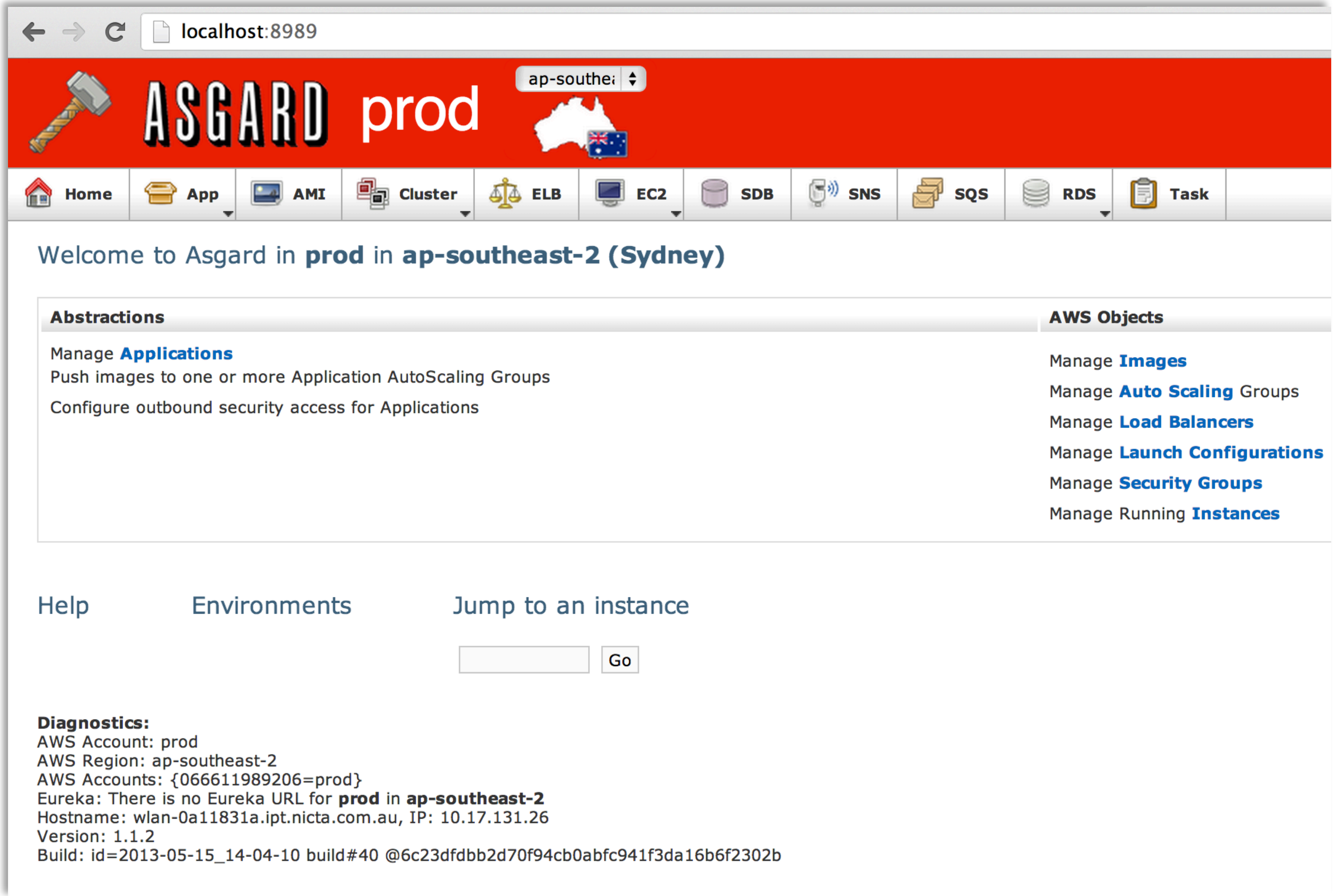
NICTA

Process-Oriented Dependability (POD) Suite

- Offline: treat an operation as a process and discovery it
  - **POD-Discovery**: Process discovered automatically from logs/scripts
- Online: use the discovered processes
  - Process model and context: process/instance/step, expected states
  - **POD-Detection**: Errors detected by
    - process conformance checking and assertions on step outcomes
  - **POD-Diagnosis**: Detected errors further diagnosed for (root) causes
    - Using fault trees, Bayesian networks and automated diagnostic testing
  - POD-Viz: visualization of process progression along monitoring
    - False monitoring alarms suppressed using process context

1. X. Xu, L. Zhu, et. al., "POD-Diagnosis: Error Diagnosis of Sporadic Operations on Cloud Applications," in 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2014.
2. Xiwei Xu, L. Zhu, et al., "Crying Wolf and Meaning it: Reducing False Alarms in Monitoring of Sporadic Operations" in International Workshop on Complex faUlts and Failures in LargE Software Systems (COUFLESS) at ICSE, 2015.

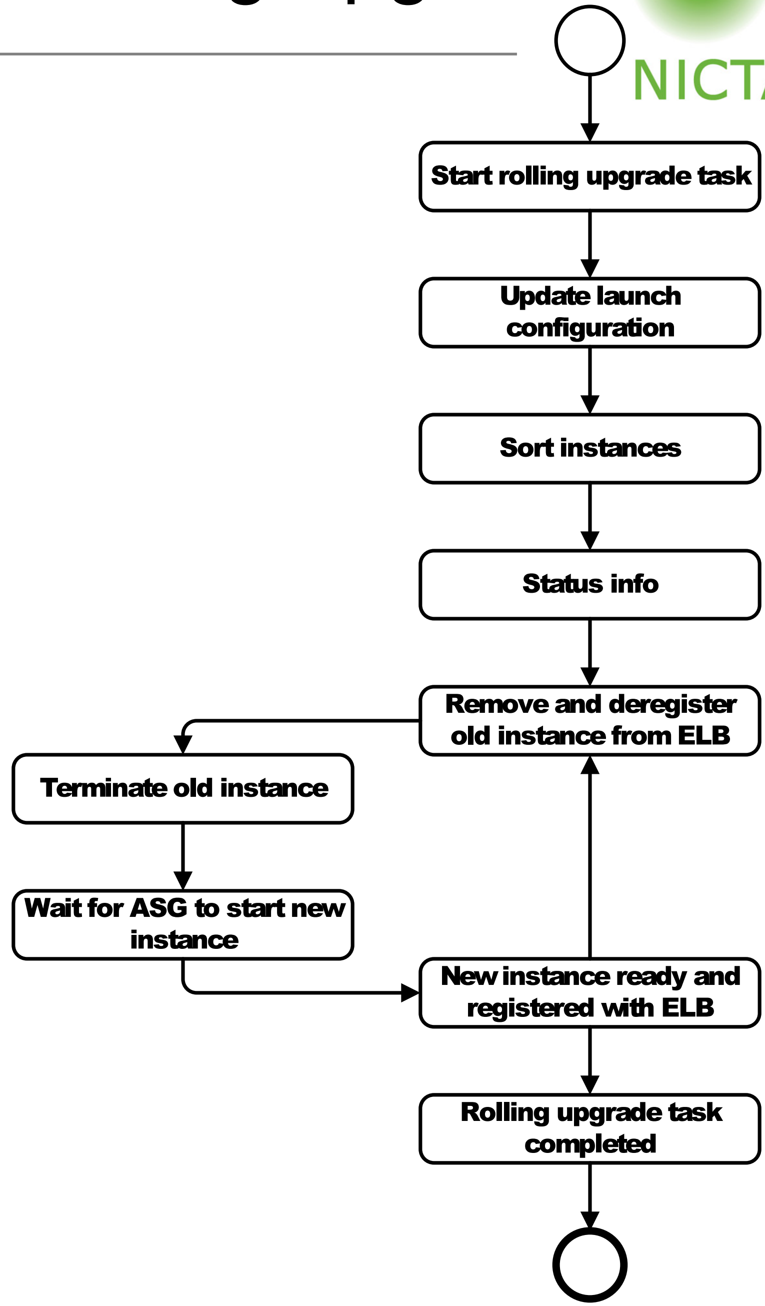
Sporadic Operation Example: Rolling Upgrade



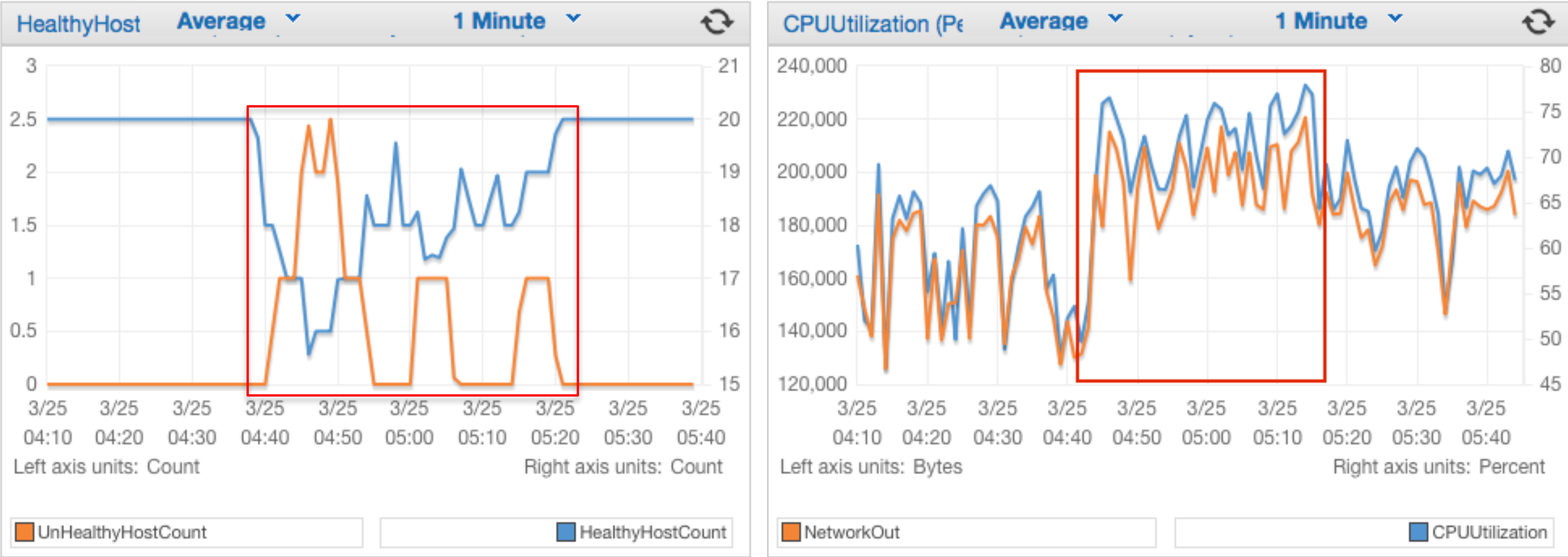
Sporadic Operation Example: Rolling Upgrade

- Rolling upgrade: upgrade software on many virtual machines without downtime or significant additional cost

- 100 servers running in the cloud with version 1 software
- Upgrade 10 servers at a time to version 2 software
- Potentially takes a long time to complete with errors *during* the operation with other interfering operations



System Monitoring During Rolling Upgrade



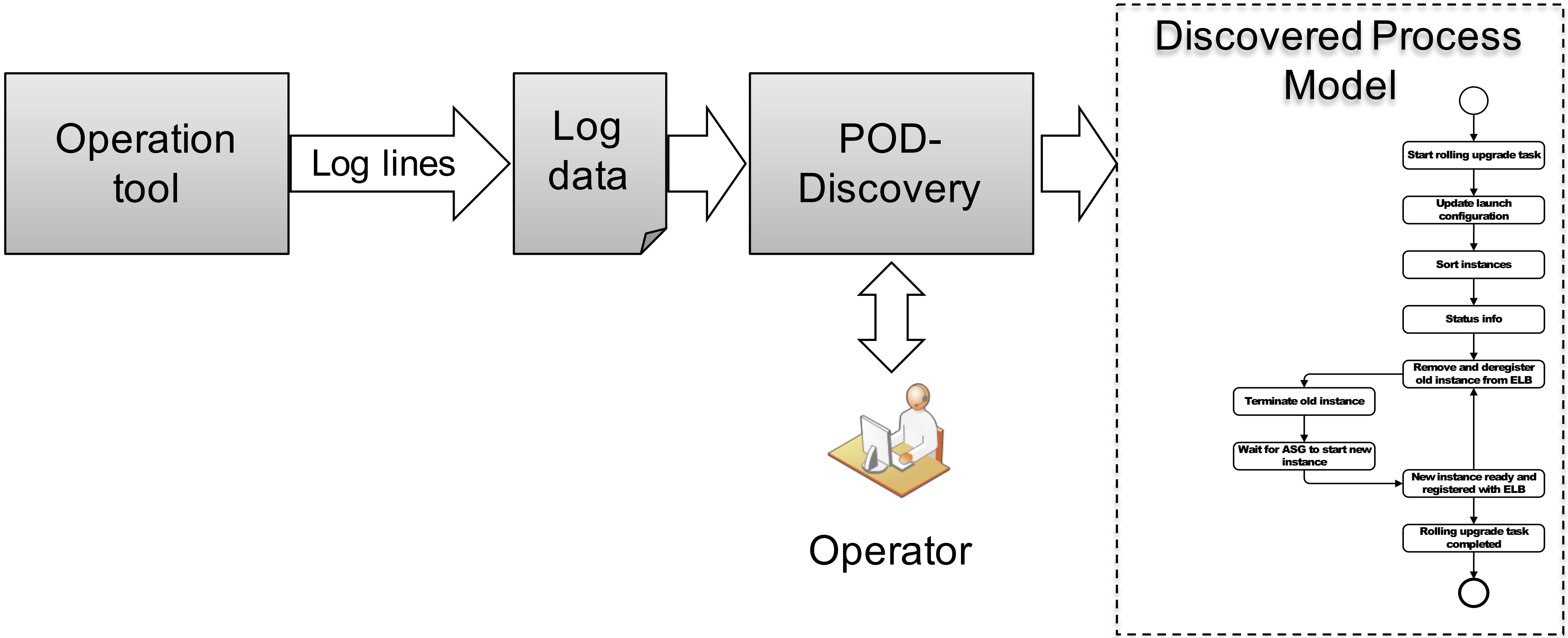
Standard anomaly detection raises lots of alerts

→ Operators switch it off during sporadic operations or ignore the alerts

- Not a good idea if done 25x a day

Part: Operation Process Discovery

POD-Discovery

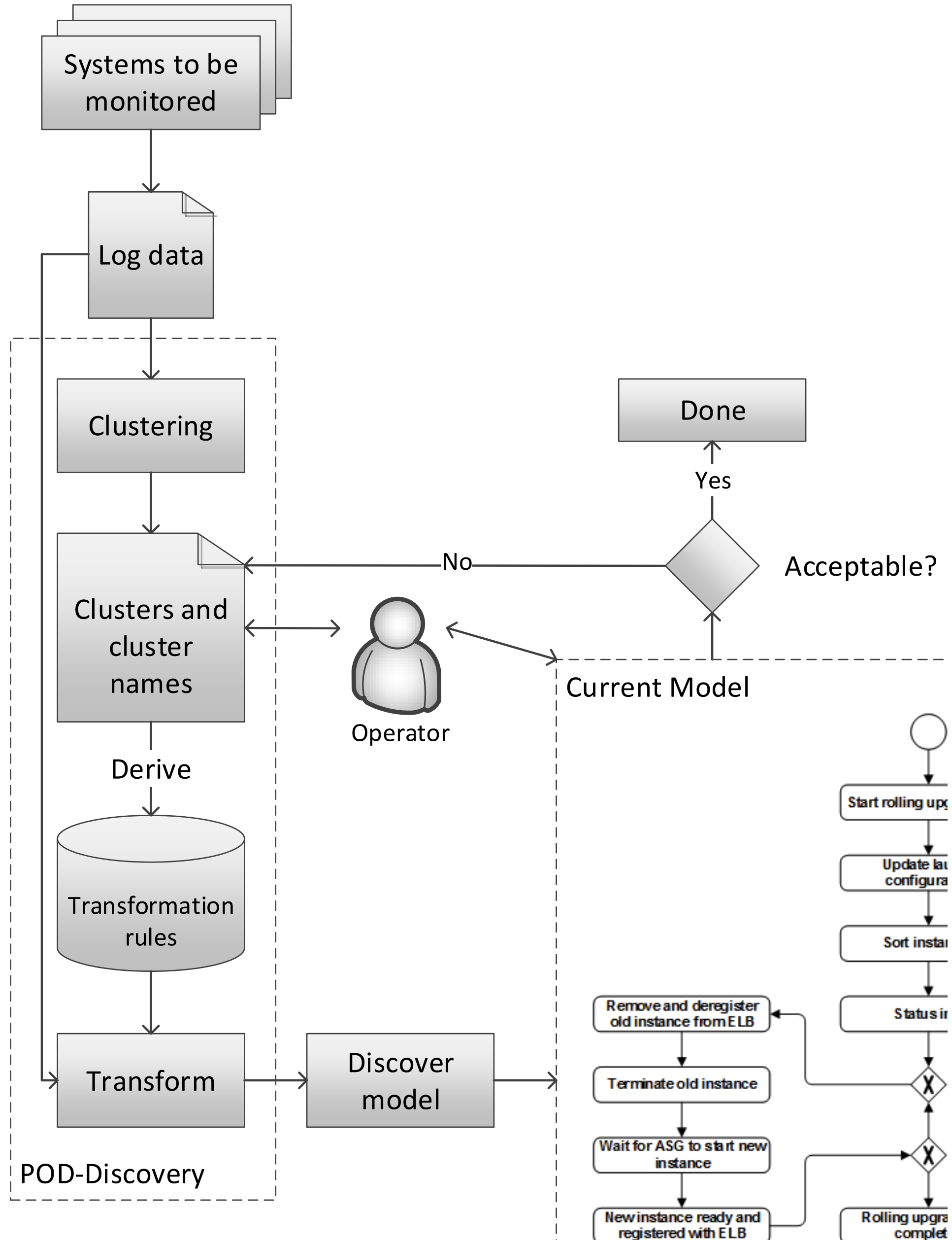


NICTA

POD-Discovery

NICTA

1. Collect & filter logs (using Logstash)
2. Cluster the log lines (HAC)
3. Decide on clusters & name them (Operator)
4. Automatically derive regular expressions for the clusters & formulate transformation rules
5. Apply transformation rules to original log, annotate cluster names
6. Import annotated log into process mining tools & apply process discovery algorithms
7. If anything requires changes, go back to the respective steps and redo from there



POD-Discovery: distance functions

NICTA

- Tokens and distance functions
  - Tokenize log line on separators like {}, [], “ “, …
  - Detect token types:

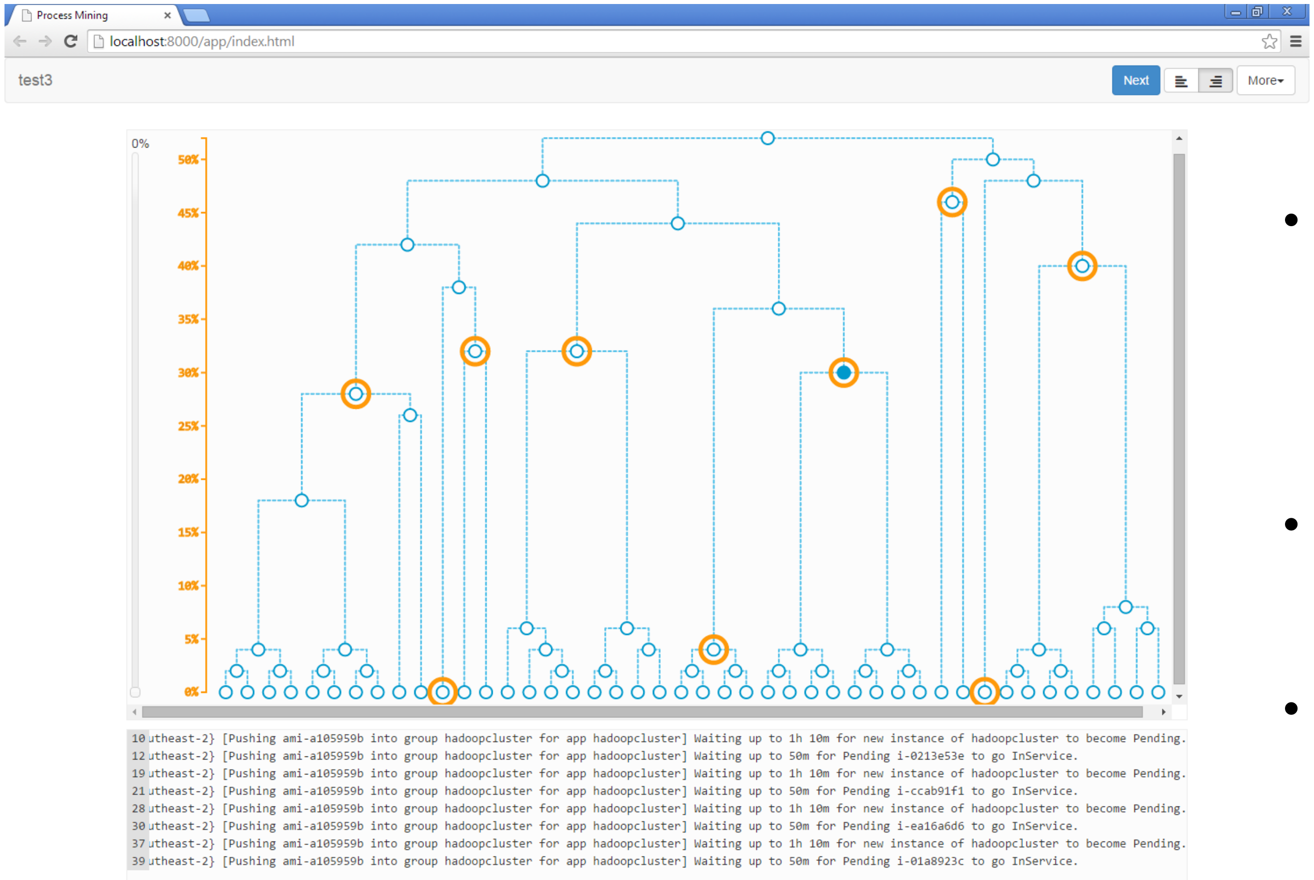
Log line:	[2013-07-12 16:11:05,640] ... {Client: localhost 127.0.0.1} ... It took 3m 28s for instance i-3fb96c03 to terminate and be replaced by i-0213e53e							
L0 Tokens:	date	time	text	IP address	text	instance ID	text	instance ID
L1 Tokens:	date-time		"{" + text + IP address + "}"		text + instance ID + text + instance ID			

- Calculate distance per log line pair
  - Same token type: custom distance function, some statically **0**
  - Different token type: max. distance, i.e. **1**

Log line 1:	[2013-07-12 16:11:05,640] ... {Client: localhost 127.0.0.1} ... [16:11:05,640] ... It took 3m 28s for instance i-3fb96c03 to terminate and be replaced by i-0213e53e								
L0 Tokens:	date	time	text	IP address	time	text	instance ID	text	instance ID
Distances:	0	0.024	0.473	0	1	0	0	0	0
L0 Tokens:	date	time	text	IP address	AMI ID	text	instance ID	text	instance ID
Log line 1:	[2013-07-12 16:14:36,912] ... {Client: control-node 10.10.21.153} ... ami-a105959b ... It took 2m 23s for instance i-3663730b to terminate and be replaced by i-ccab91f1								

POD-Discovery: Clustering

- Hierarchical agglomerative clustering (HAC)
  - Does not require a-priori knowledge of the number of clusters we search for (as opposed to *k*-means, e.g.)
  - Based on log line distances produce interactive dendrogram:



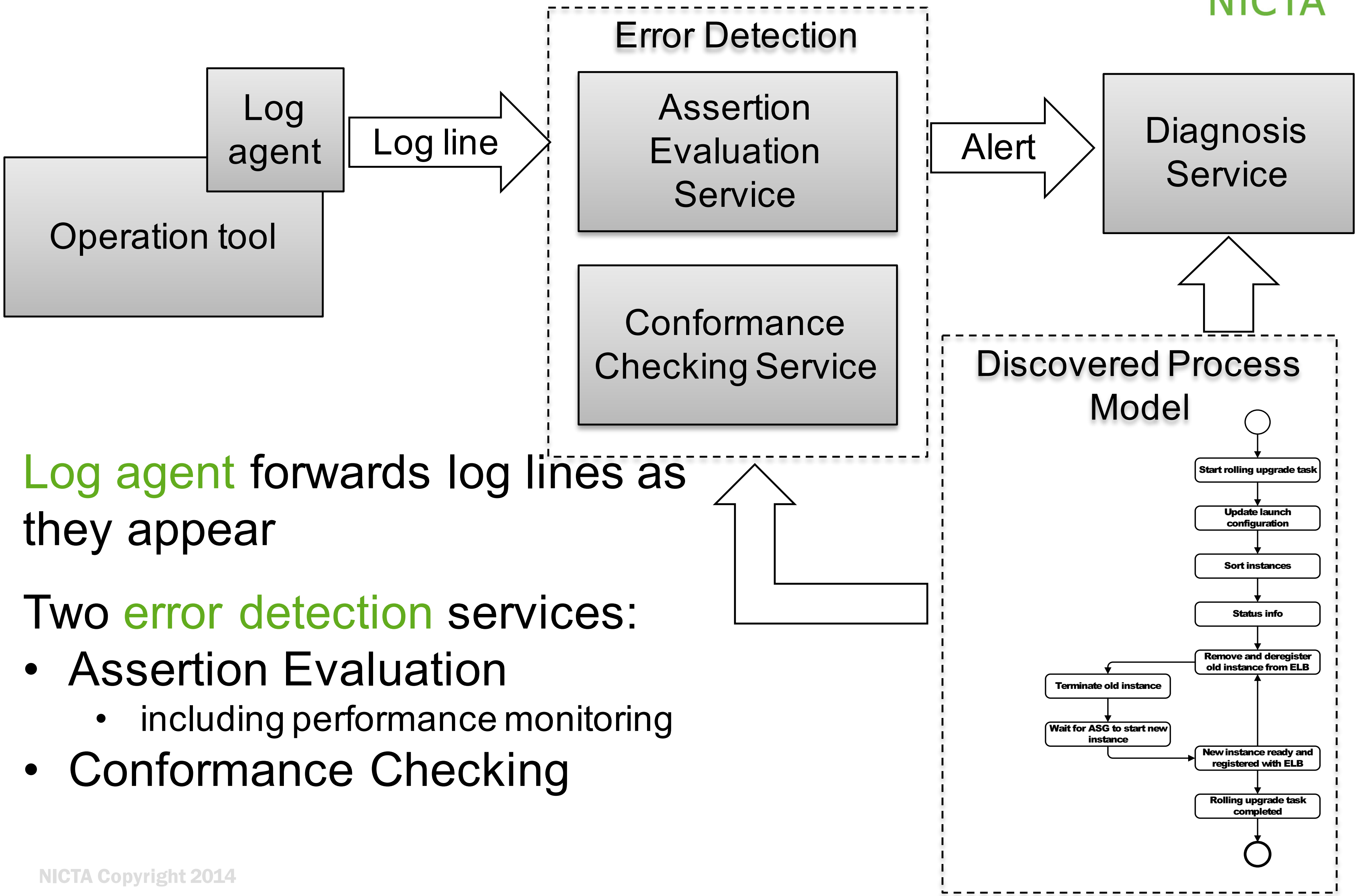
- Inspect content of clusters, e.g., for the filled node log lines are shown below
- Select clusters, shown with orange circles
- Can fine-tune afterwards

NICTA

Part:
POD-Detection & POD-Diagnosis

Australian Government

Error Detection & Diagnosis Overview



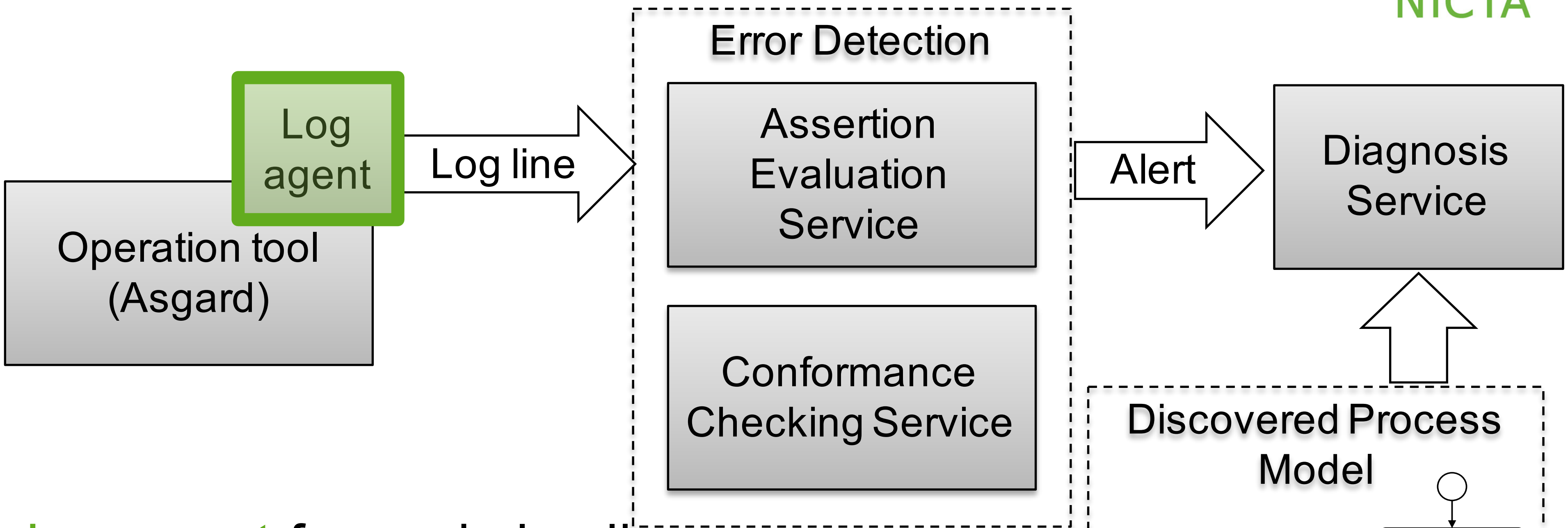
Log agent forwards log lines as they appear

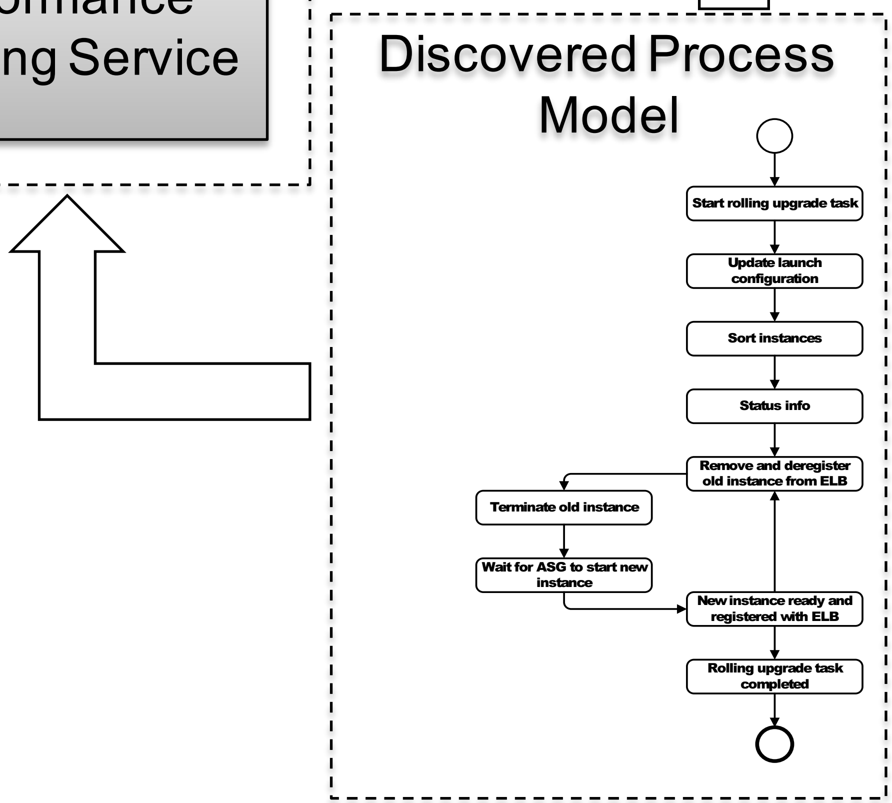
Two error detection services:

- Assertion Evaluation
  - including performance monitoring
- Conformance Checking

Error Detection & Diagnosis Overview

NICTA





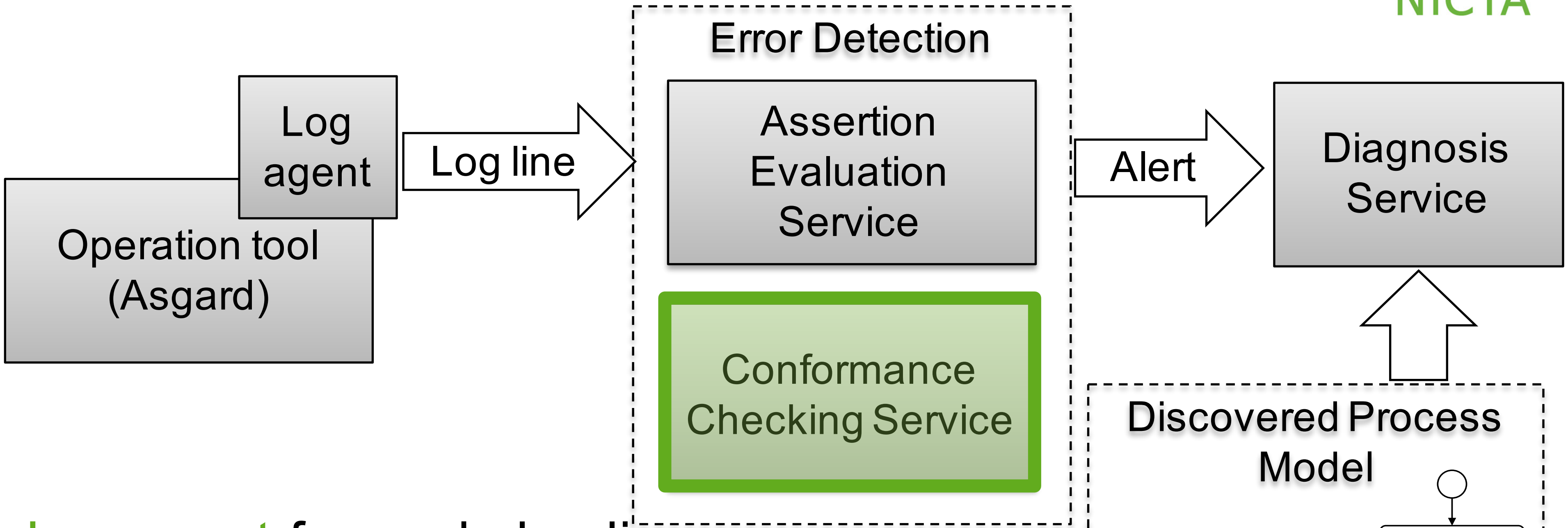
Log agent forwards log lines as they appear

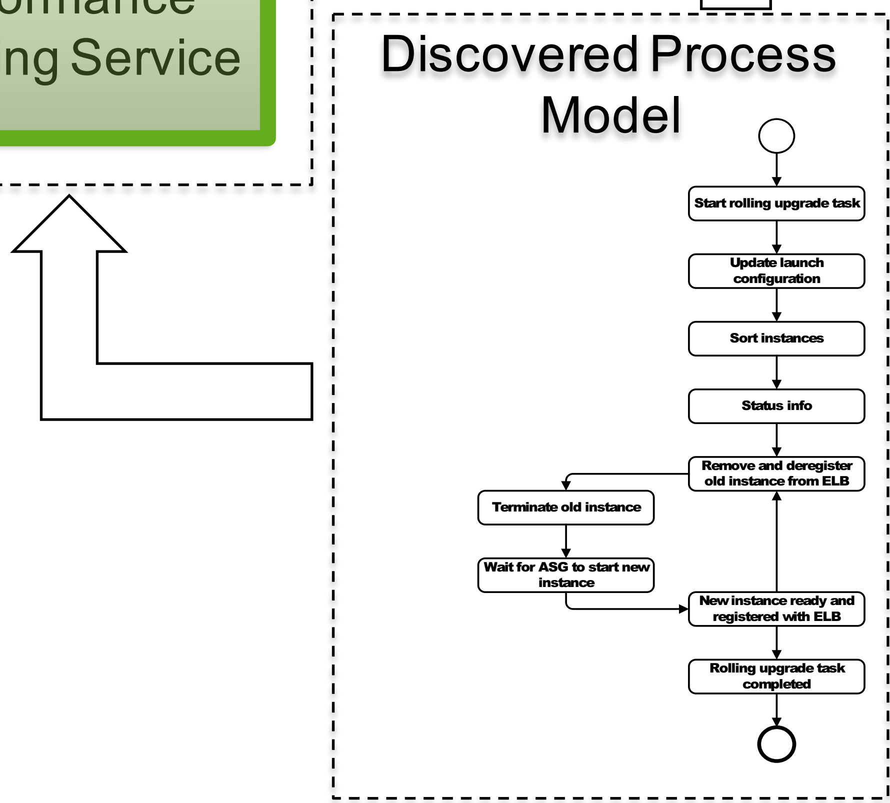
Two error detection services:

- Assertion Evaluation
  - including performance monitoring
- Conformance Checking

Error Detection & Diagnosis Overview

NICTA



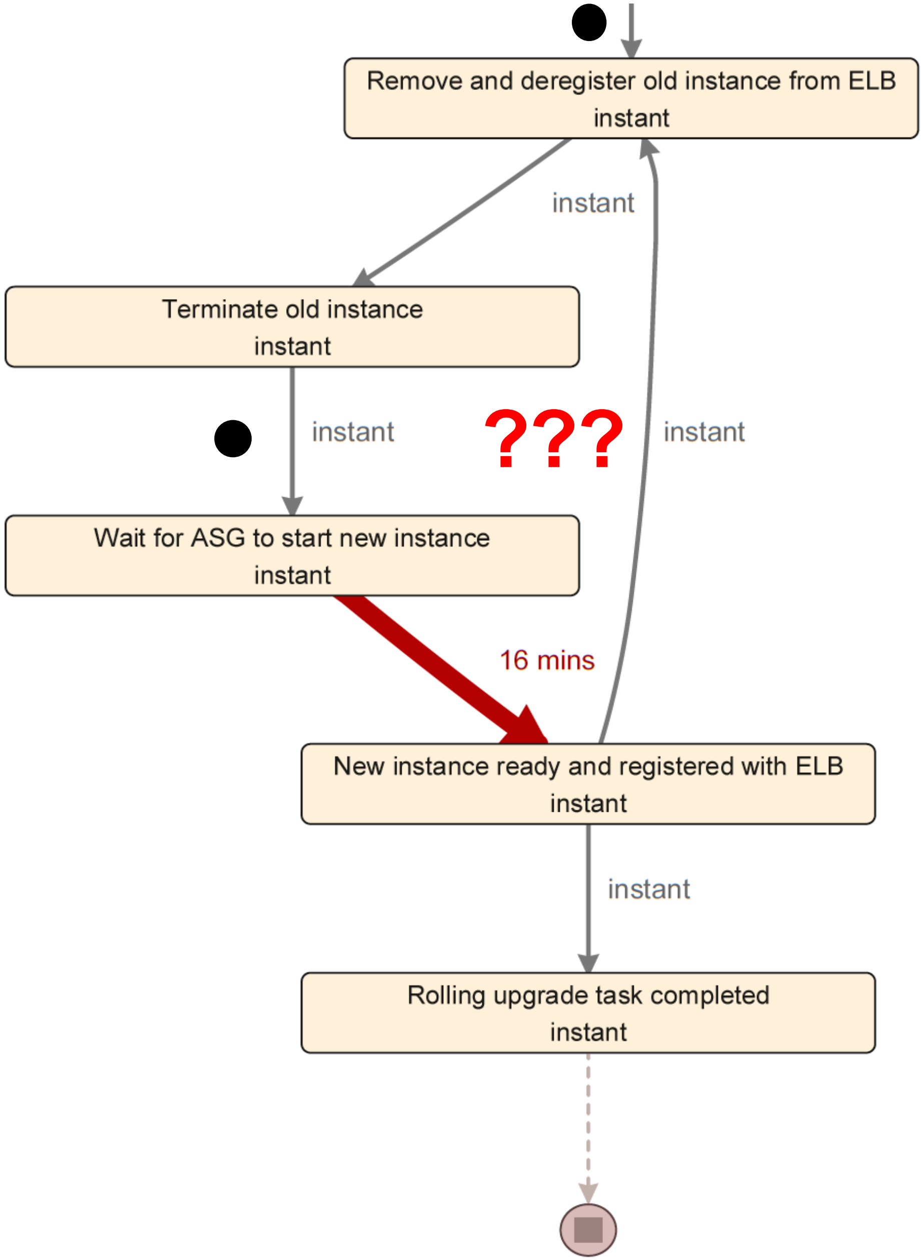


Log agent forwards log lines as they appear

Two error detection services:

- Assertion Evaluation
  - including performance monitoring
- Conformance Checking

Conformance Checking: how it works



Log lines:

- `Remove ...`
- `Terminate ...`
- `Wait ...`
- `Terminate ...`

Raise alert

Error count +1

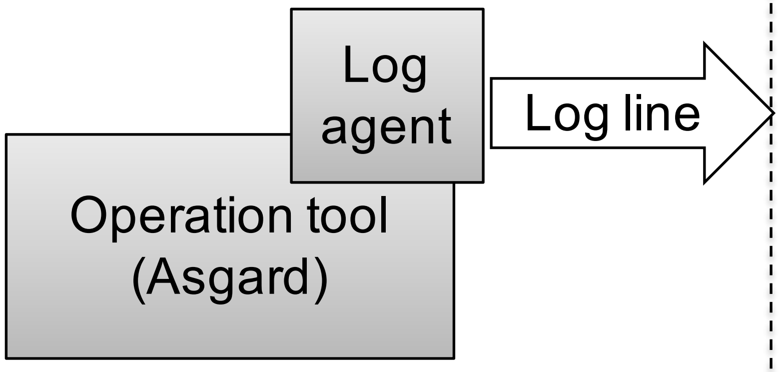
Conformance Checking: outcomes

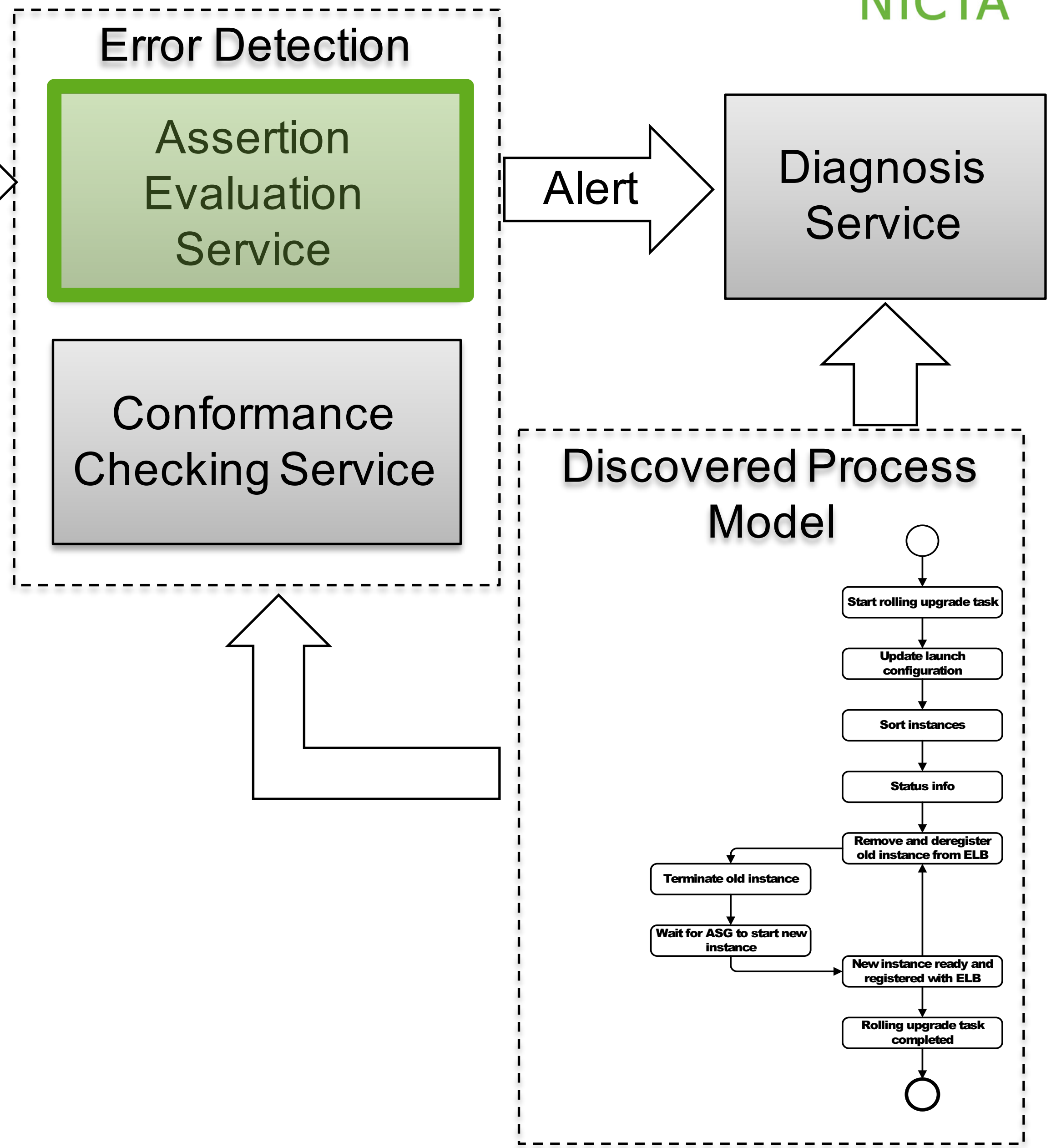
NICTA

- Conformance checking can detect the following types of errors:
  - *Unknown / error log line*: a log line that corresponds to a known error, or is simply unknown.
  - *Unfit*: a log line corresponds to a known activity, but said activity should not happen in the current execution state of the process instance.
- All other log lines are deemed *fit*
- Goal: 100% fit, else raise alert
  - Learn from false alerts → improve classification and/or model

Error Detection & Diagnosis Overview

NICTA





Log agent forwards log lines as they appear

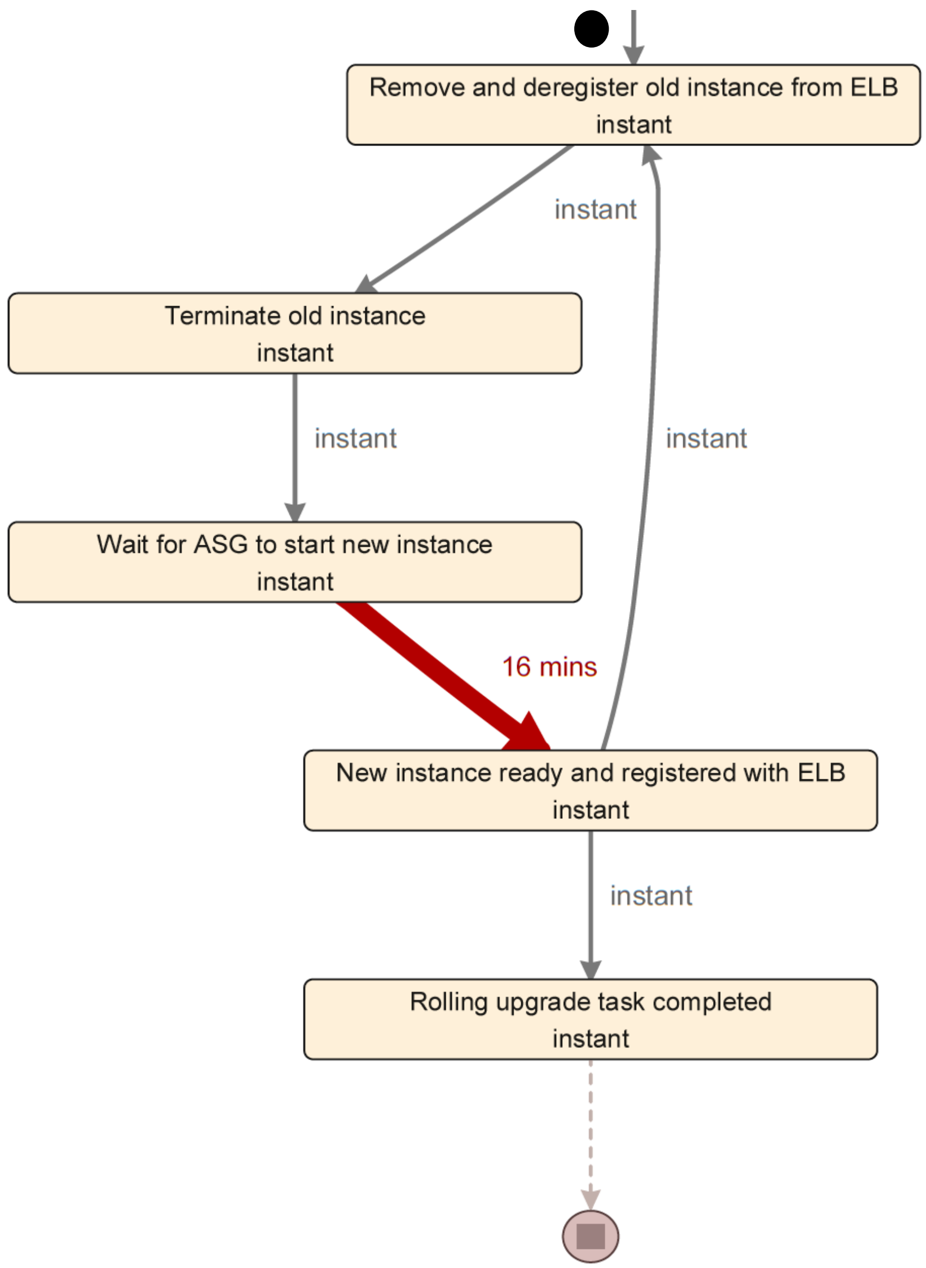
Two error detection services:

- Assertion Evaluation
- Conformance Checking

Creating Assertions

- Assertions check if the actual state at some point is the expected state
  - Coded against Cloud APIs – can find out the true state of resources directly
- Currently, some assertions are automatically generated while others are written manually
  - API call logs and **statistical analysis of metrics**
- Low level assertion
  - Instance *i* is terminated successfully
- High level assertion
  - There are *n* instances running version *x*

Assertion Evaluation: how it works



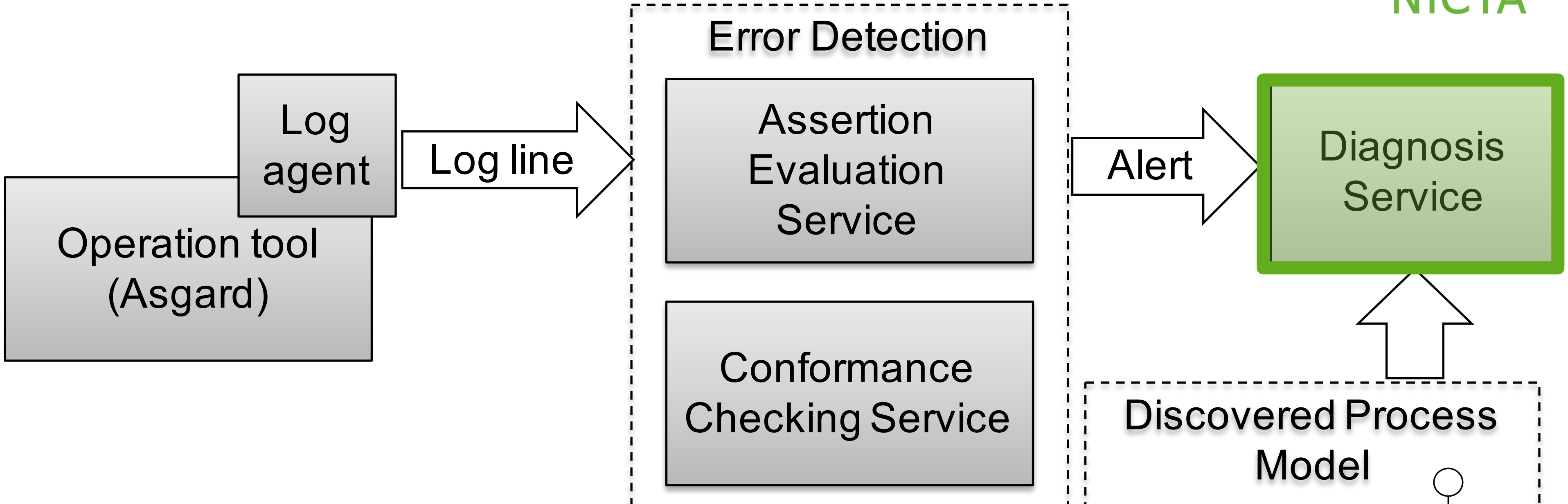
Log line:

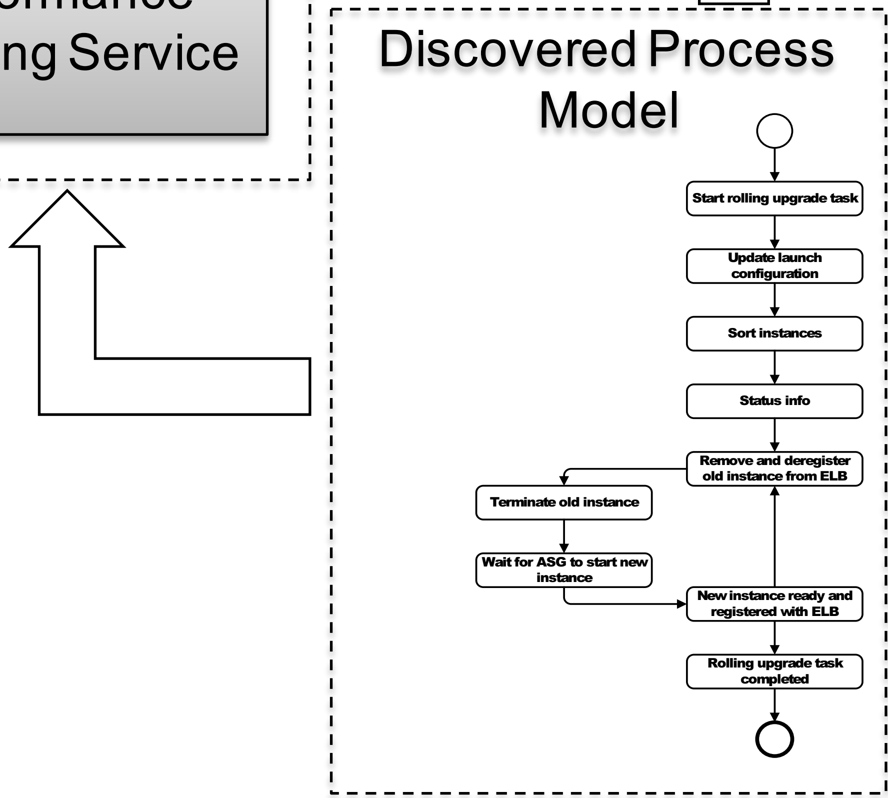
- `Remove ...`
- `Terminate ...`
- `Wait ...`
- `New instance ...`

Assertions:

- i‘ … and … registered with ELB
- i has been removed from ASG

Error Detection & Diagnosis Overview



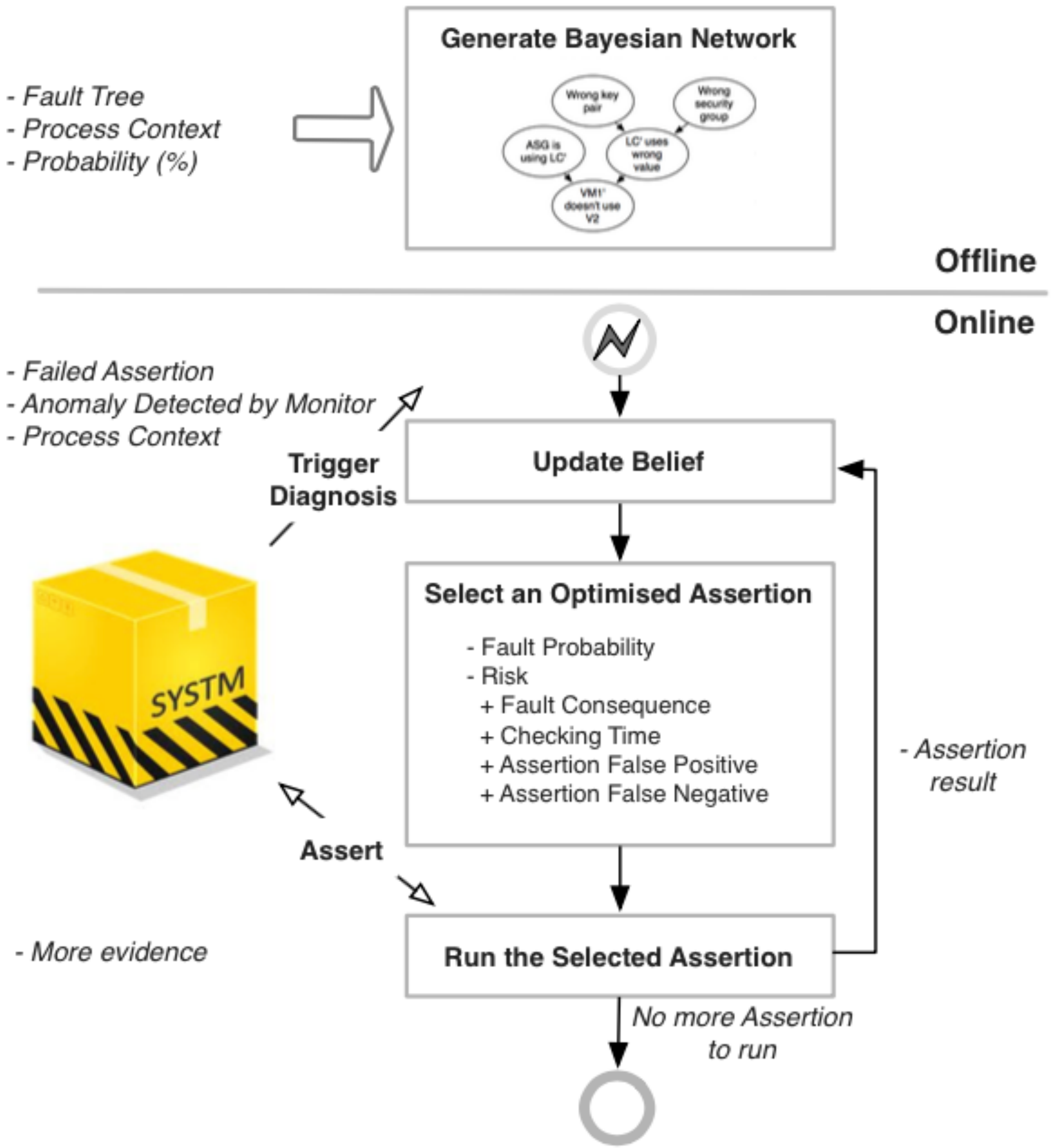


Log agent forwards log lines as they appear

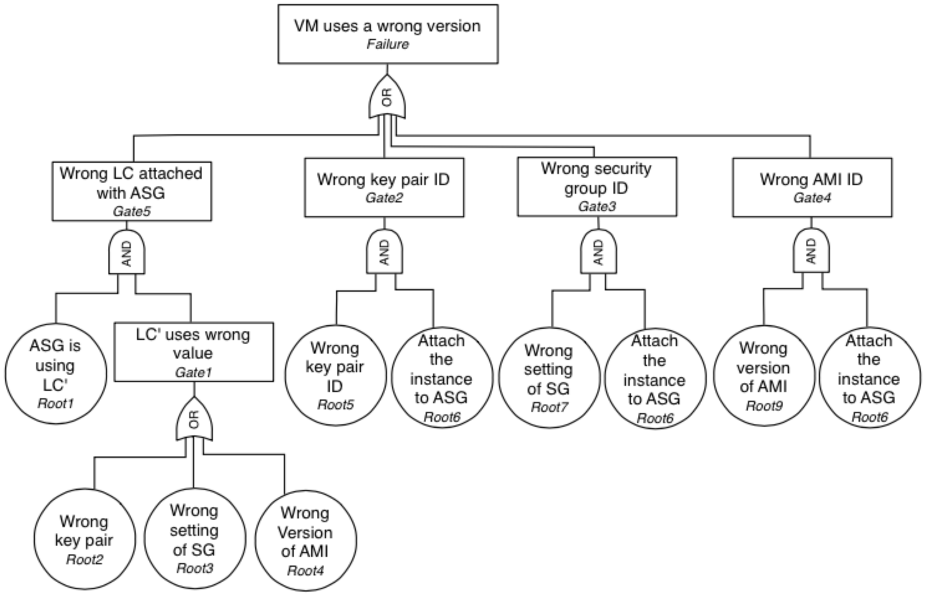
Two error detection services:

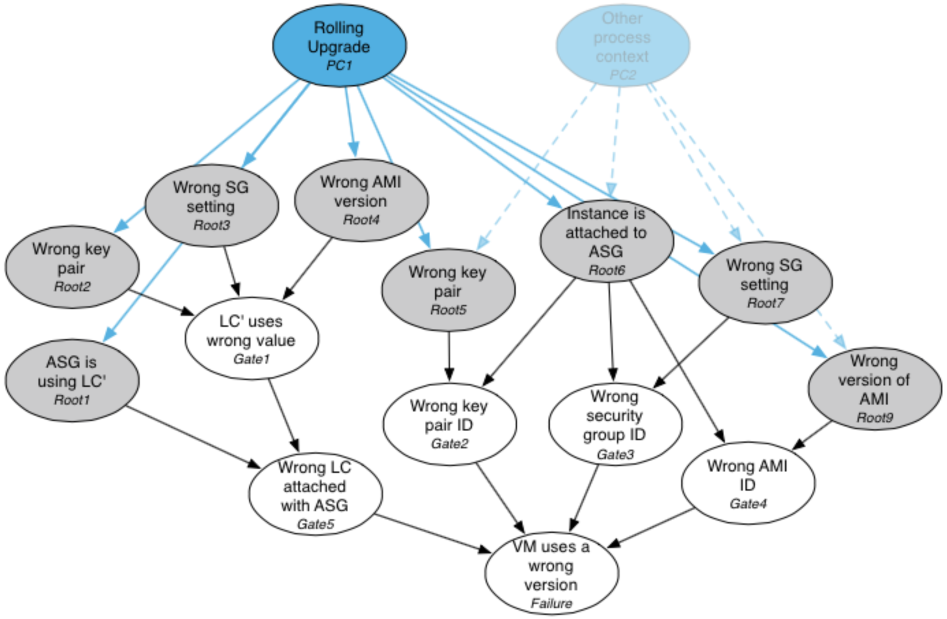
- Assertion Evaluation
  - including performance monitoring
- Conformance Checking

POD-Diagnosis: how it works



POD-Diagnosis: how it works





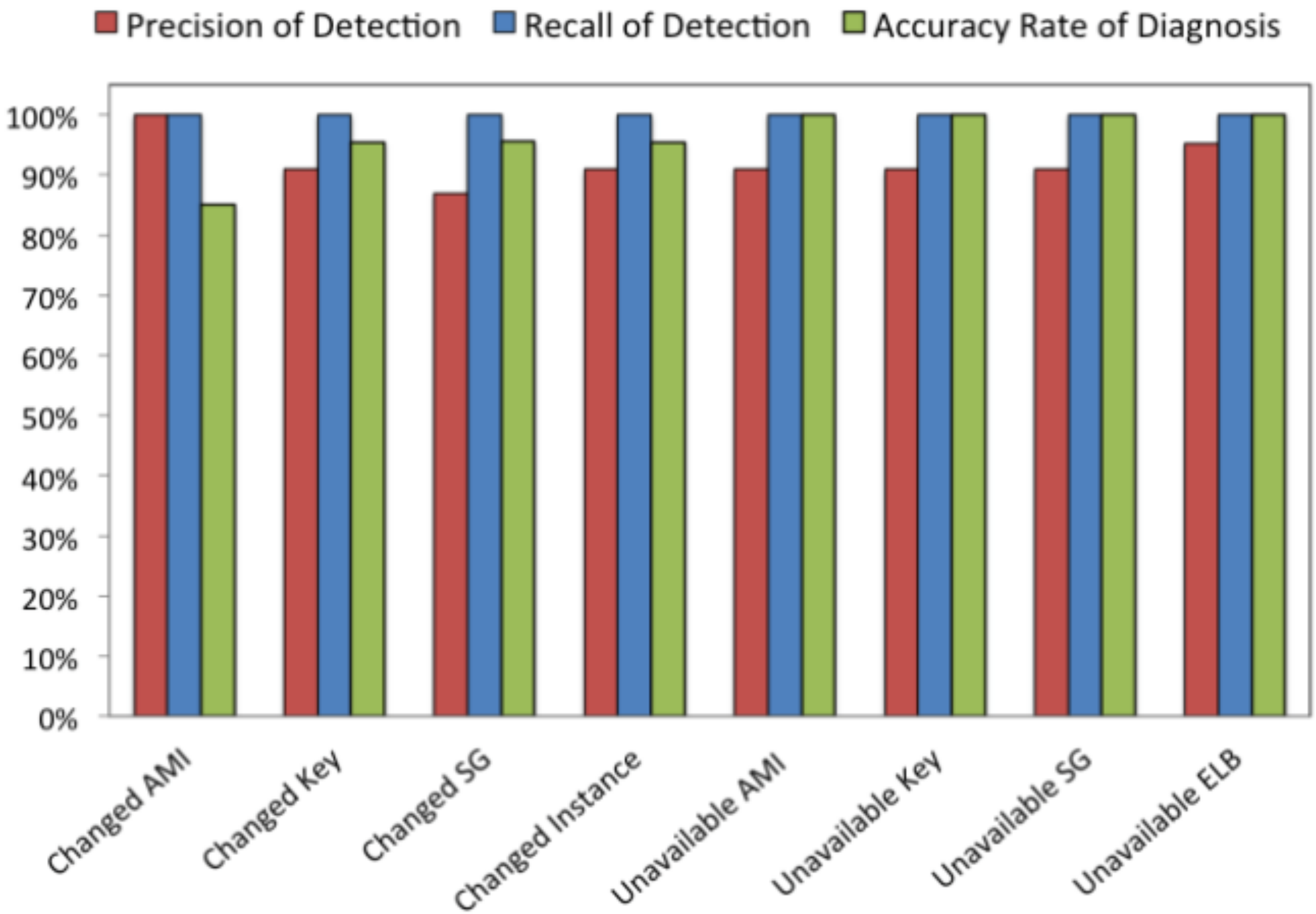
- Fault trees built as knowledge base and converted to Bayesian network
- Nodes in BN represents potential faults and causes
- On-demand diagnosis tests to update the belief and locate the causes
    - Probability-based
    - Online optimization-based

Evaluation: POD-Detection/Diagnosis

NICTA

- Experiments
  - Rolling upgrade of 100+ node cluster in AWS
    - Fault injection+ confounding processes: random kill, scaling-in..
- Detected errors
  - Assertion checking: known errors and global errors
    - Examples: key management, launch configuration, images…
  - Compliance checking: unknown errors
    - skipping activities or undone activities
- Diagnosed faults and root causes
- Time and precision
  - Compared with Asgard/Monitoring internal mechanisms
    - Detected **more** errors **earlier**
  - Diagnosis: limited to known causes in the fault tree
    - **Faster** diagnosis

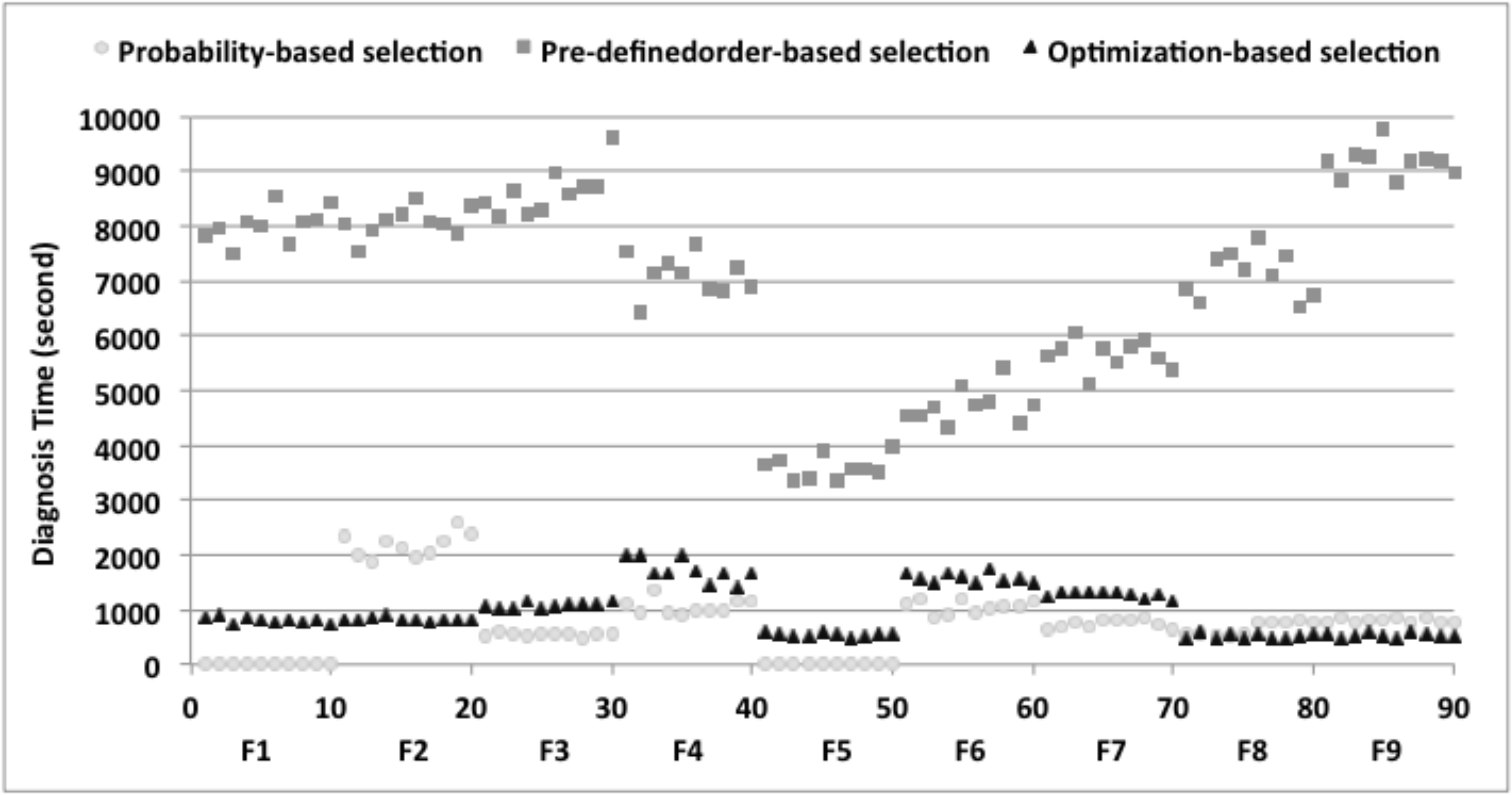
Error Detection Accuracy



** Diagnosis accuracy rate is based on an pre-defined diagnosis order*

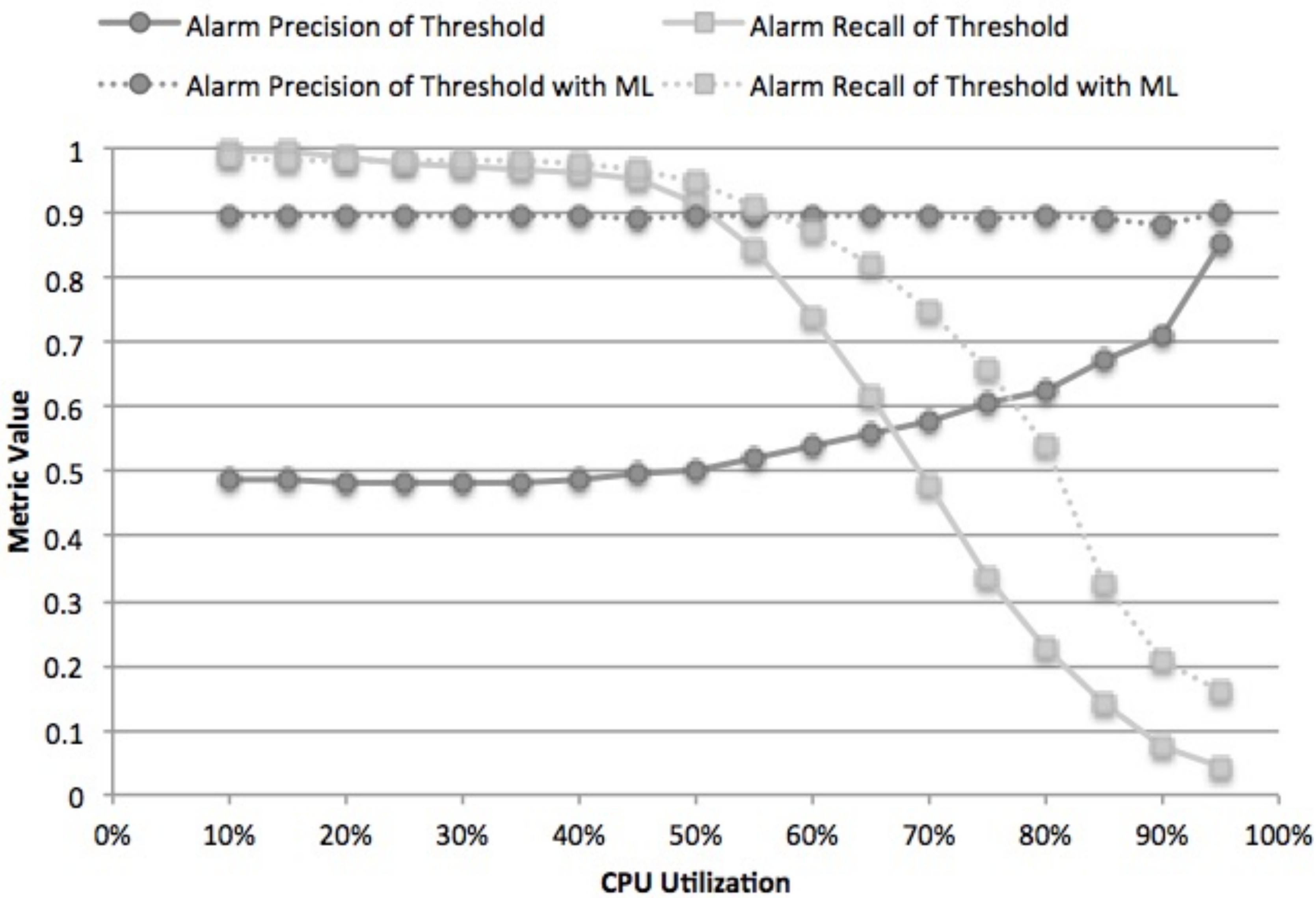
Evaluation: POD Diagnosis Time

NICTA



Evaluation: False Alarm Suppression

NICTA



Evaluation: Metrics Threshold Generation

Validation of approach with 22 rounds of rolling upgrade with multiple operations and fault injection

	Basic Detection			Final classification
Time window	**1 min**	**3 min**	**5 min**	**3 min**
Precision	**0.567**	**0.712**	**0.742**	**0.904**
Recall	**0.933**	**0.984**	**0.984**	**1.000**
Accuracy	**0.845**	**0.912**	**0.942**	**0.981**
F-Score	**0.706**	**0.826**	**0.846**	**0.949**

From basic to final:

#	Decision	Fix Explanation
21	valid	Asgard attempts to terminate a VM that has been terminated earlier by fault injection
11	not counted	Missing data
5	valid	Fault injection's attempt fails due to VM being already terminated by Asgard earlier
5	not valid	Wrong prediction due to discretization function
2	not valid	Wrong prediction (FP) due to more than one instance termination the minute before
2	not valid	More than two minute delay in termination (requires time-window more than 2 minute)
2	valid	VM terminated by fault injection while pending (booting)

Summary

- Core area: Dependable Applications on Cloud
  - Process-Oriented Dependability (POD)
    - process context for error diagnosis and detection
      - Machine Learning: false alarm suppression
      - Automatic metrics threshold/assertion generation
      - DTMC-based analysis: better predictability
  - DevOps: a new book on Amazon
- Connections with SPEC DevOps WG
  - Solving continuous change challenge..
    - No easy baseline or benchmarking
  - Log analysis for operation process context
    - Runtime log collection and analysis; log as monitoring
  - Context used in both SPE time and APM time
    - Performance impact of confounding "changes"
  - Other past work and current collaborations
    - J2EE micro-benchmarking + queuing models for prediction
    - Hadoop performance optimization and prediction