# DICE: a Model-Driven DevOps Framework for Big Data

Giuliano Casale
*Imperial College London*

DICE Horizon 2020 Project
Grant Agreement no. 644869
http://www.dice-h2020.eu

Funded by the Horizon 2020
Framework Programme of the European Union

# DevOps & Big data

- DevOps for Big data
  - DataOps: DevOps & data engineering
  - DevOps for data intensive applications
- DevOps with Big data
  - Mining monitoring data for feedbacks/RCA
  - Novel uses of data science to address software engineering problems

# DICE: DevOps for DIAs

*Mission*: support SMEs in developing high-quality cloud-based data-intensive applications (DIAs)

- Horizon 2020 research project (4M€, 2015-18)
- 9 partners (Academia & SMEs), 7 EU countries

Imperial College London

XLAB NOT IDLE

@ IeAT

Universidad Zaragoza

NETFECTIVE Creative Technology

ATC ATHENS TECHNOLOGY CENTER

POLITECNICO DI MILANO

prodevelop Integración de tecnologías

flexiOPS

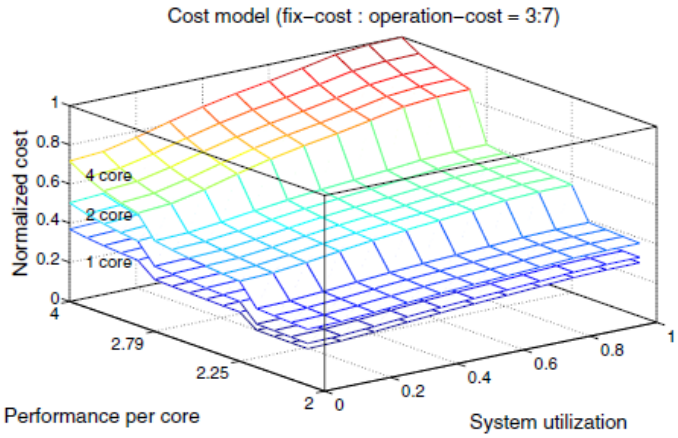
# Challenges

1. Deliver a DevOps toolchain for DIAs
   - From requirements to CI/CD
   - Shared view of the application across roles

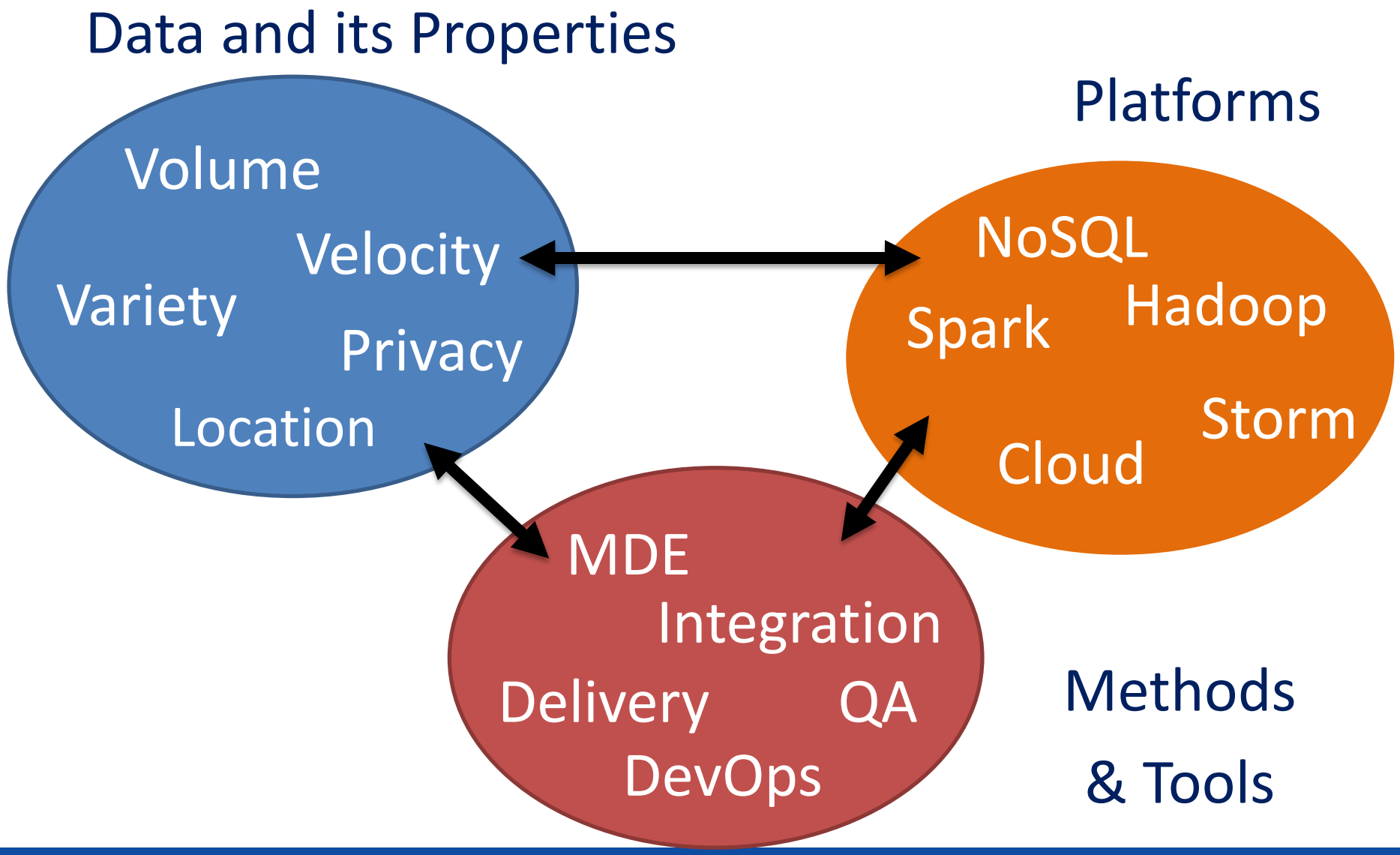
2. Increase automation of quality analysis during DIA development and operation
   - Analysis of efficiency, reliability, correctness
   - Architectural optimisation for DIAs

# What do we mean by Quality?

- Reliability
  - Availability
  - Fault-tolerance
- Efficiency
  - Performance
  - Costs
- Correctness
  - Privacy & security
  - Temporal metrics



# DIA dimensions

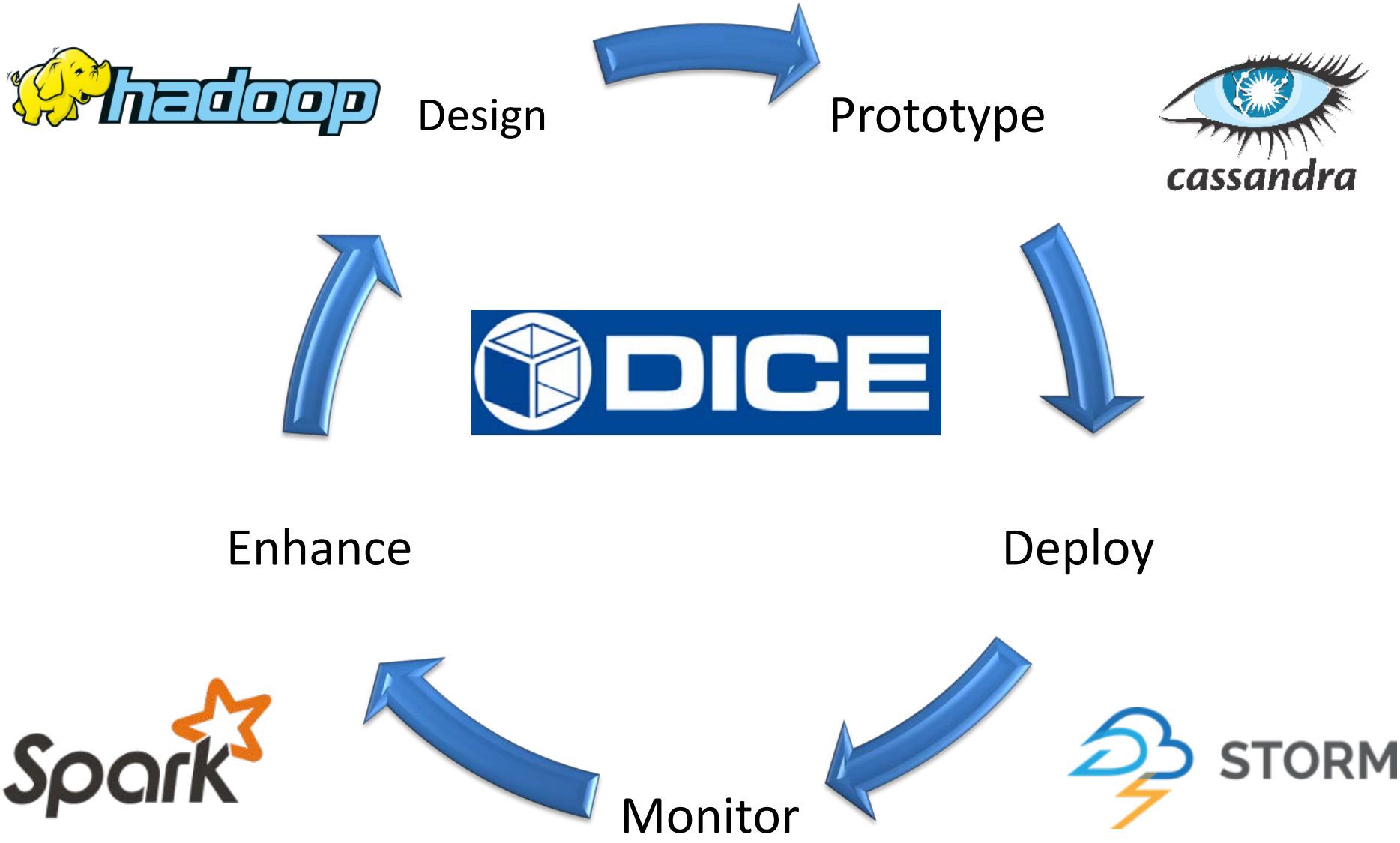


# Our case for model-driven DevOps

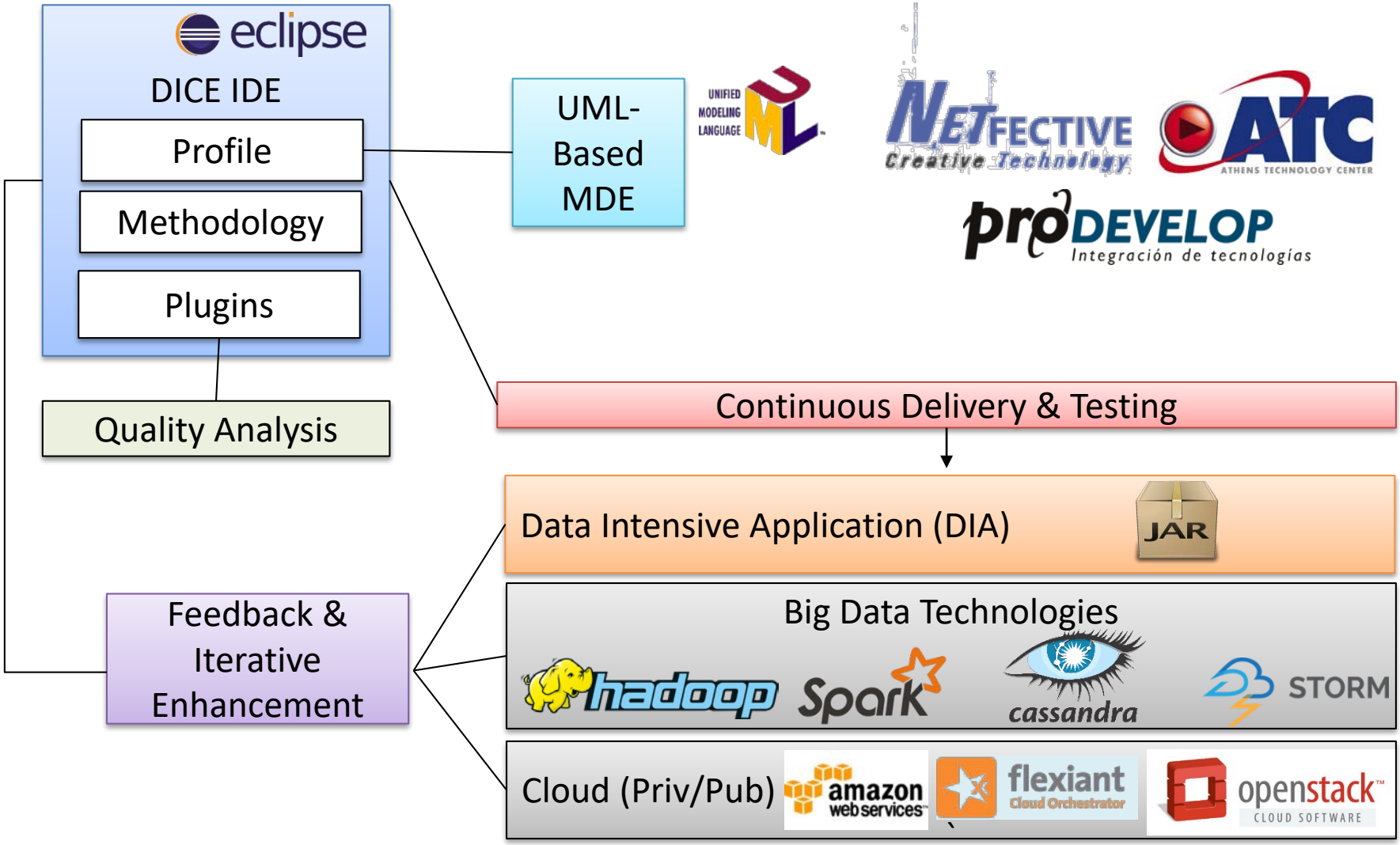
1. Model-driven orchestration
   - Big Data technologies are in movement
2. QA through M2M transformation of requirements/SLAs/architectural information
   - Test (and stress test) generation
   - Simulation
   - Verification
   - Optimization (costs, SLAs, config)
   - Architectural anti-pattern detection

Eases QA skill shortage

# DICE: Quality-Aware DevOps for Big Data



# DICE Framework

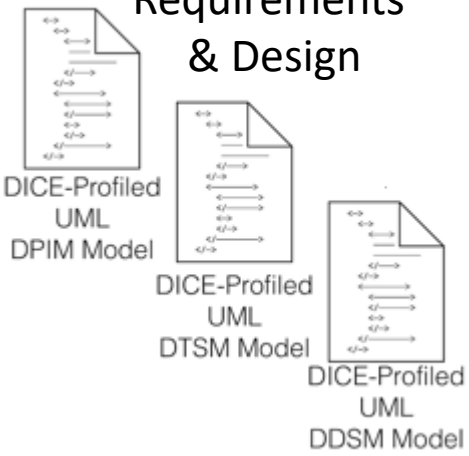


# DICE Workflow - Dev

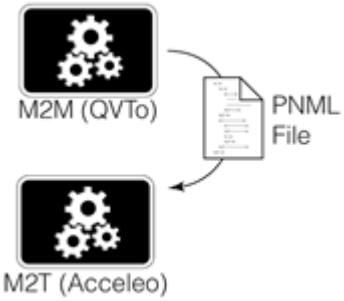

DICE Eclipse IDE

Enhance Architecture

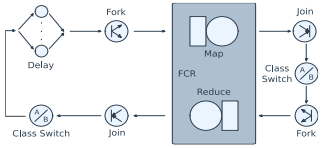
Requirements & Design



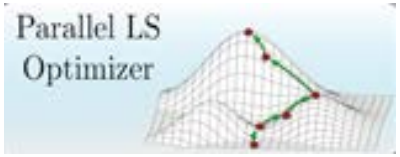

Transform to Formal Models




Simulate & Verify




Optimize & Anti-patterns

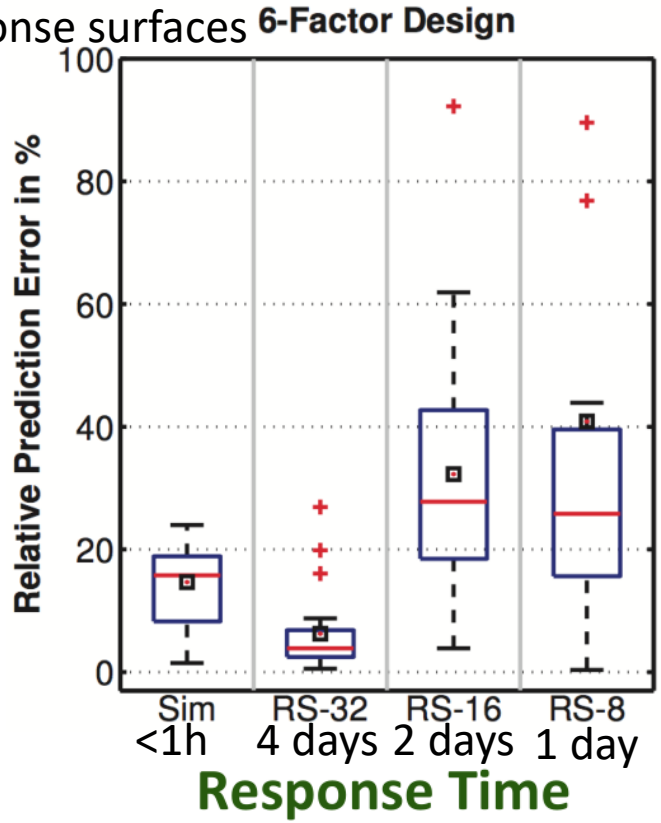


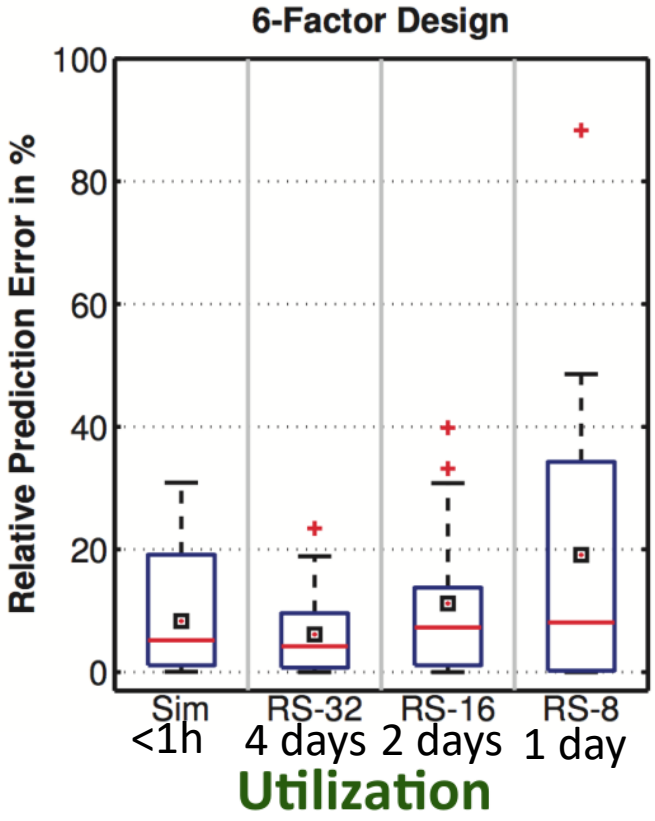
# Why simulating DIAs?

- Early-stage scalability analysis
- Effective what-if analysis for in-memory wklds

Example: SAP HANA

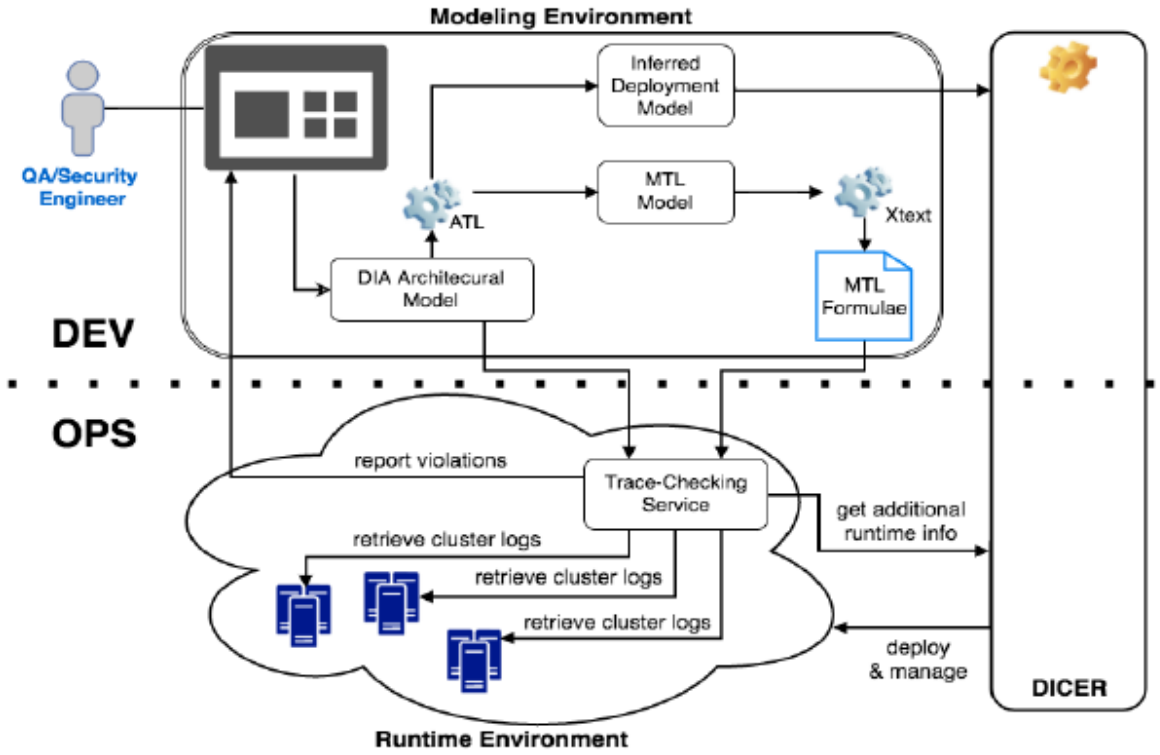
Sim vs Response surfaces



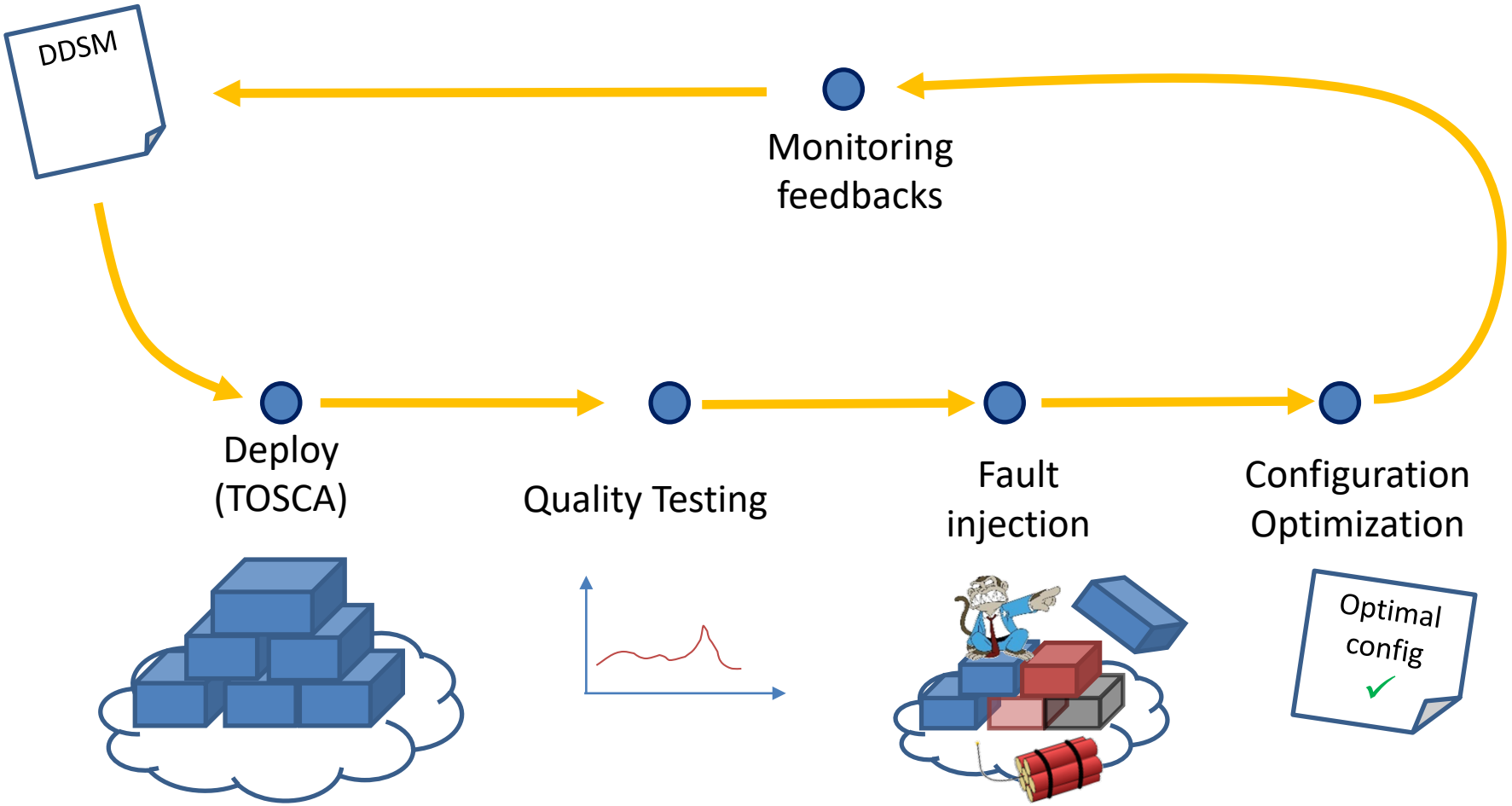



# Why verification in DIAs?

- Privacy-by-design (GDPR)
- M2M transformation propagates changes to runtime log verification

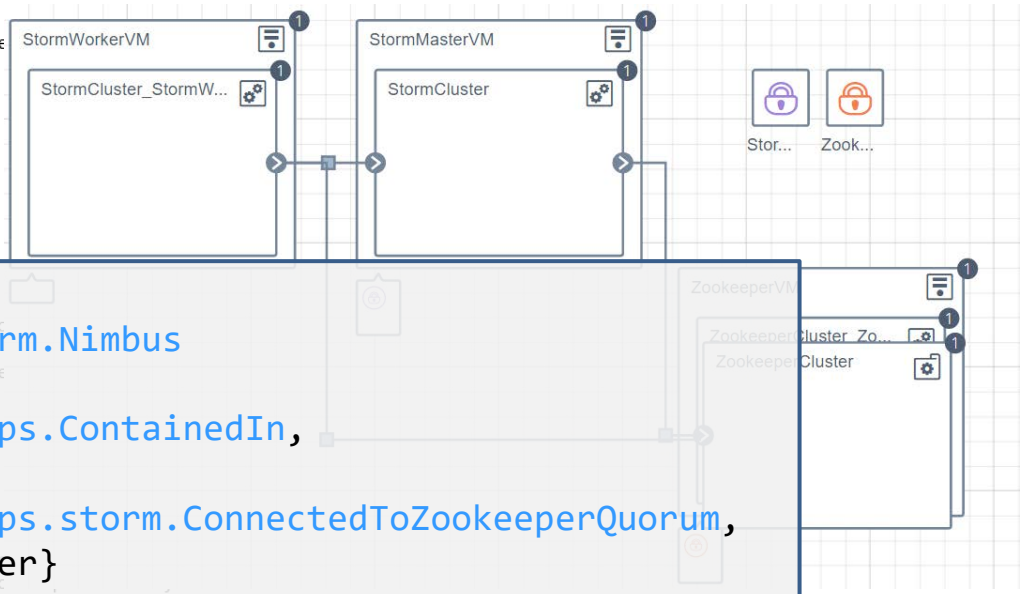


# DICE Workflow - Ops



# TOSCA technology library for DIA

```
tosca_definitions_version: cloudify_dsl_1_3
imports: [
  'http://dice-project.github.io/DICE-Deployment-Cloudify/spec/openstack/0.1.4/plugin.yaml']
node_templates:
  ZookeeperVM:
    type: dice.hosts.Small
    instances: {deploy: 1}
    relationships:
    - {type: dice.relationships.ProtectedBy, target: ZookeeperCluster_Zookee
  StormMasterVM:
    type: dice.hosts.Small
    relationships:
    - {type: dice.relationships.ProtectedBy, target: StormCluster_firewall}
  StormWorkerVM:
    type: dice.hosts.Small
    instances: {deploy: 1}
  StormCluster:
    type: dice.components.storm.Nimbus
    relationships:
    - {type: dice.r
    - {type: dice.r
    properties:
      configuration
        queueSize:
  StormCluster_firewall:
    type: dice.firewall_rules.storm.Nimbus
  StormCluster_StormWorkerVM_slave:
    type: dice.components.storm.Worker
    relationships:
    - {type: dice.r
    - {type: dice.r
    - {type: dice.r
    properties:
      configuration
        memoryCapacity: '4096'}
  ZookeeperCluster_ZookeeperVM_worker:
    type: dice.components.zookeeper.Server
    relationships:
    - {type: dice.r
    - {type: dice.r
    properties:
      configuration: {tickTime: '1500', initLimit: '10', syncLimit: '5'}
  ZookeeperCluster_ZookeeperVM_worker_firewall:
    type: dice.firewall_rules.zookeeper.Server
  ZookeeperCluster:
    type: dice.components.zookeeper.Quorum
    relationships:
    - {type: dice.relationships.zookeeper.QuorumContains, target: ZookeeperVM}
```




```
StormCluster:
    type: dice.components.storm.Nimbus
    relationships:
    - {type: dice.relationships.ContainedIn,
       target: StormMasterVM}
    - {type: dice.relationships.storm.ConnectedToZookeeperQuorum,
       target: ZookeeperCluster}
    properties:
      configuration: {taskTimeout: '30', supervisorTimeout: '60',
                      monitorFrequency: '10', queueSize: '100000',
                      retryTimes: '5', retryInterval: '2000'}
```

# DevOps for BD: Lessons learned

1. Choosing the right Big data technology upfront remains an open problem.

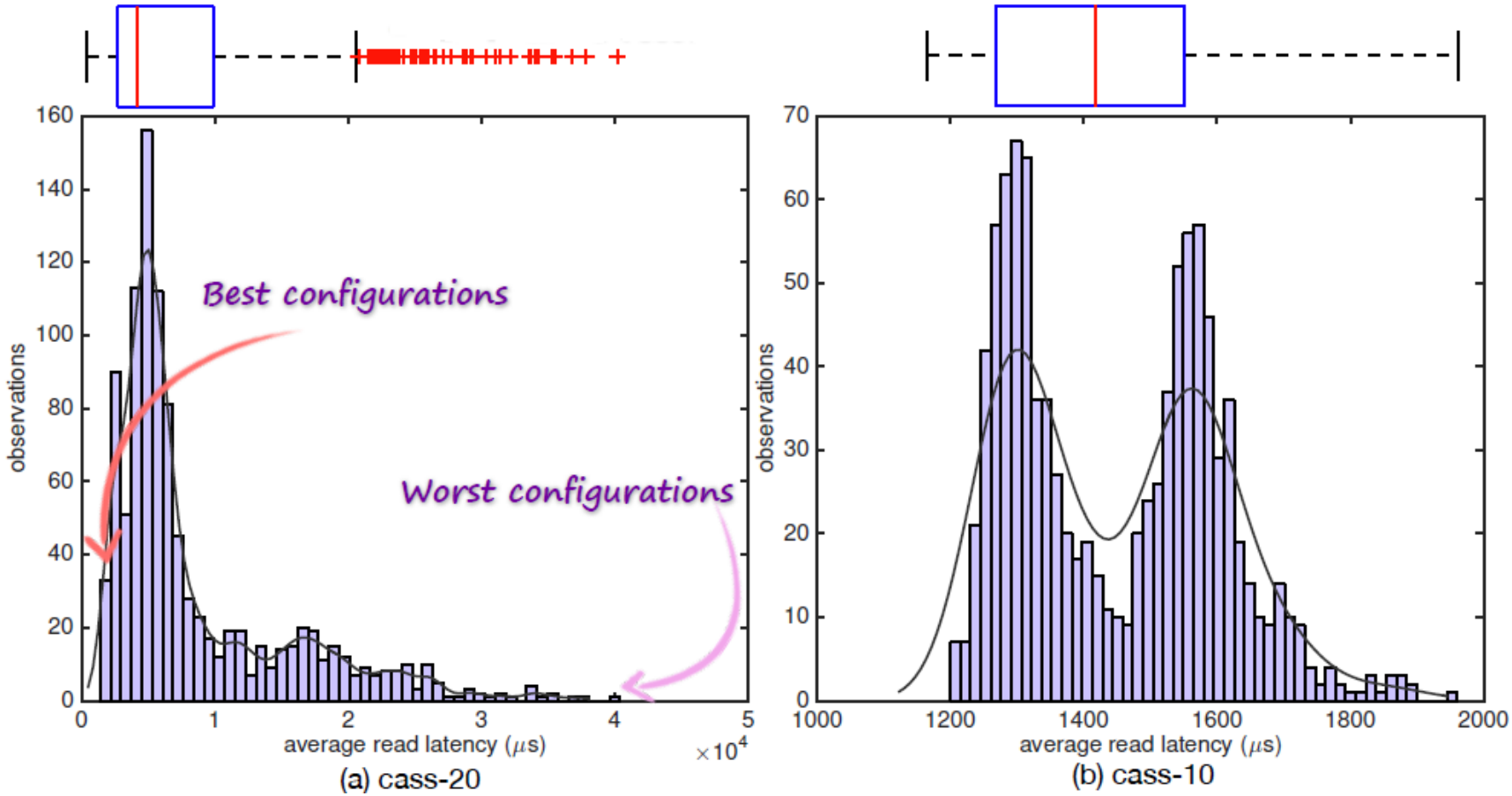
2. Model-based orchestration an easier sell than model-based design.

3. We are still far apart from convergence of DevOps with DataOps.

4. Number of tools can be a problem, methodology is key.

# DevOps with BD: Configuration

- Code changes mean that workloads change
- Wrong configurations can have high performance costs



# Configuring a DIA architecture

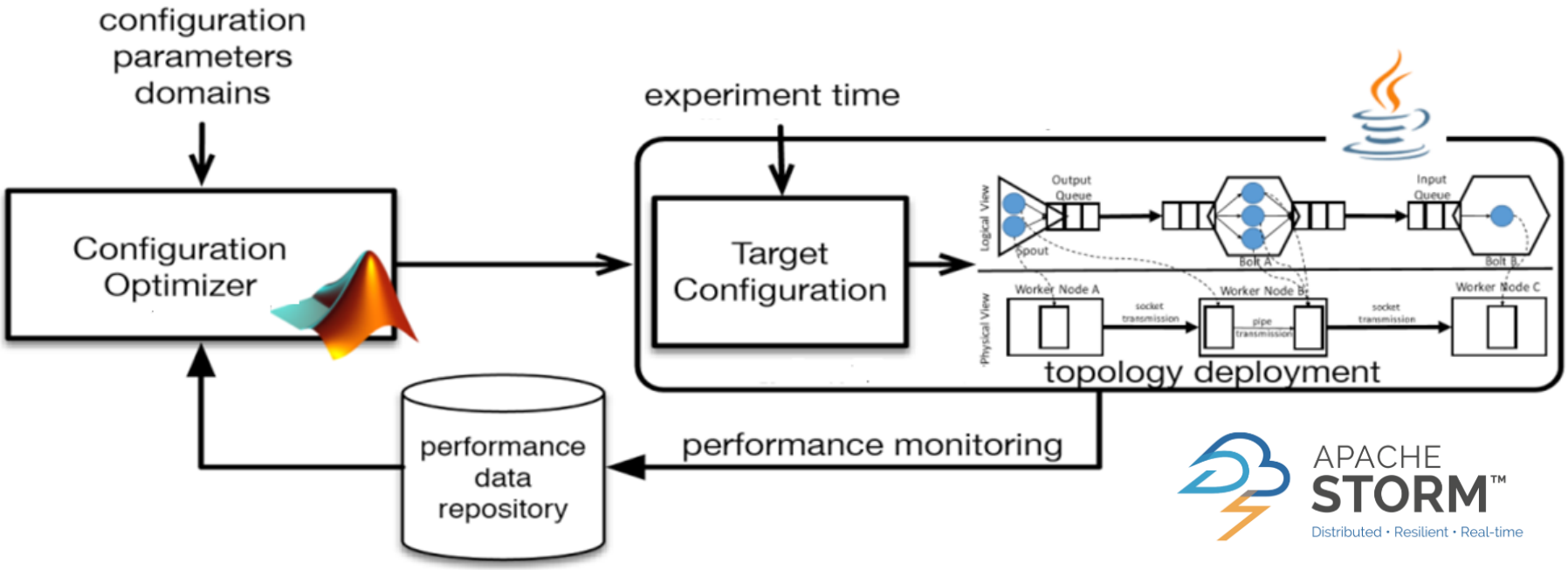
- Example: a Storm-based application

```
102
103 drpc.port: 3772
104 drpc.worker.threads: 64
105 drpc.max_buffer_size: 1048576
106 drpc.queue.size: 128
107 drpc.invocations.port: 3773
108 drpc.invocations.threads: 64
109 drpc.request.timeout.secs: 600
110 drpc.childopts: "-Xmx768m"
111 drpc.http.port: 3774
112 drpc.https.port: -1
113 drpc.https.keystore.password: ""
114 drpc.https.keystore.type: "JKS"
115 drpc.http.creds.plugin: org.apache.storm.security.auth.DefaultHttpCredentialsPlugi
116 drpc.authorizer.acl.filename: "drpc-auth-acl.yaml"
117 drpc.authorizer.acl.strict: false
118
119 transactional.zookeeper.root: "/transactional"
120 transactional.zookeeper.servers: null
121 transactional.zookeeper.port: null
122
123 ## blobstore configs
124 supervisor.blobstore.class: "org.apache.storm.blobstore.NimbusBlobStore"
125 supervisor.blobstore.download.thread.count: 5
126 supervisor.blobstore.download.max_retries: 3
127 supervisor.localizer.cache.target.size.mb: 10240
128 supervisor.localizer.cleanup.interval.ms: 600000
129
```

!

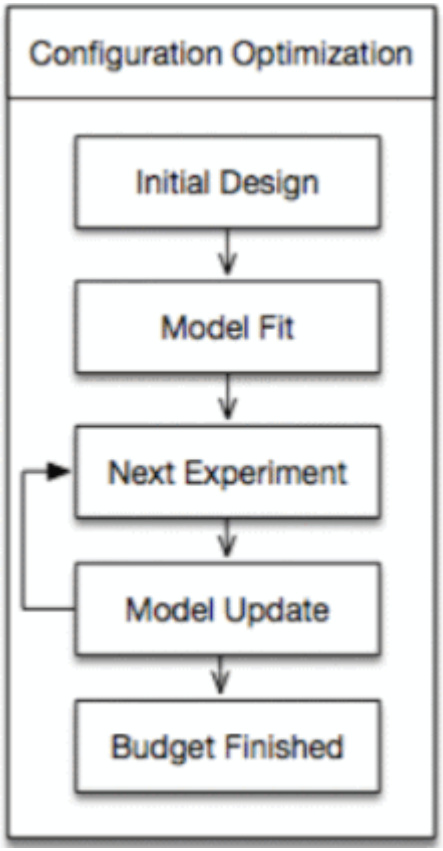
# How to evolve configuration?

- Batch exploration of alternative DIA configurations
- Algorithmic use of automated deployment



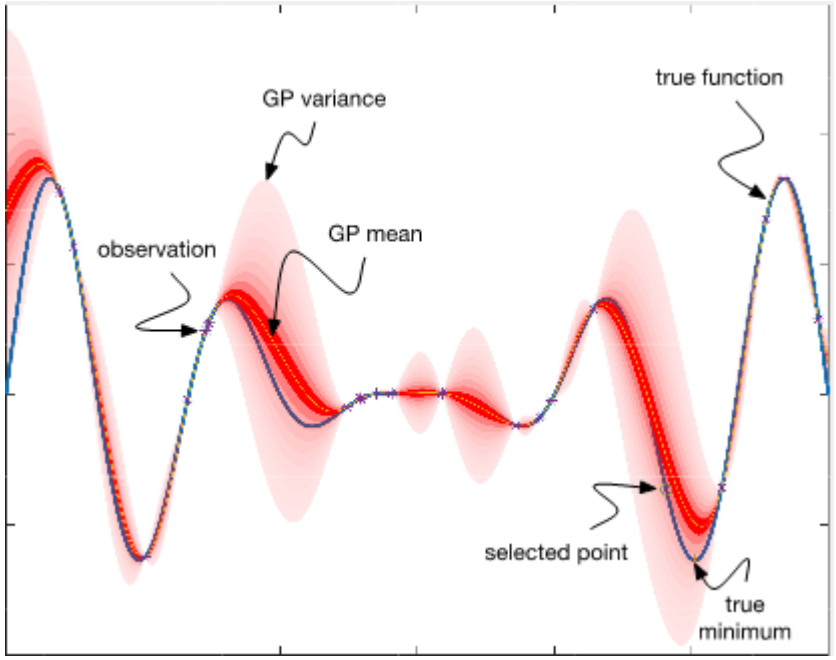
# Configuration optimization (CO)

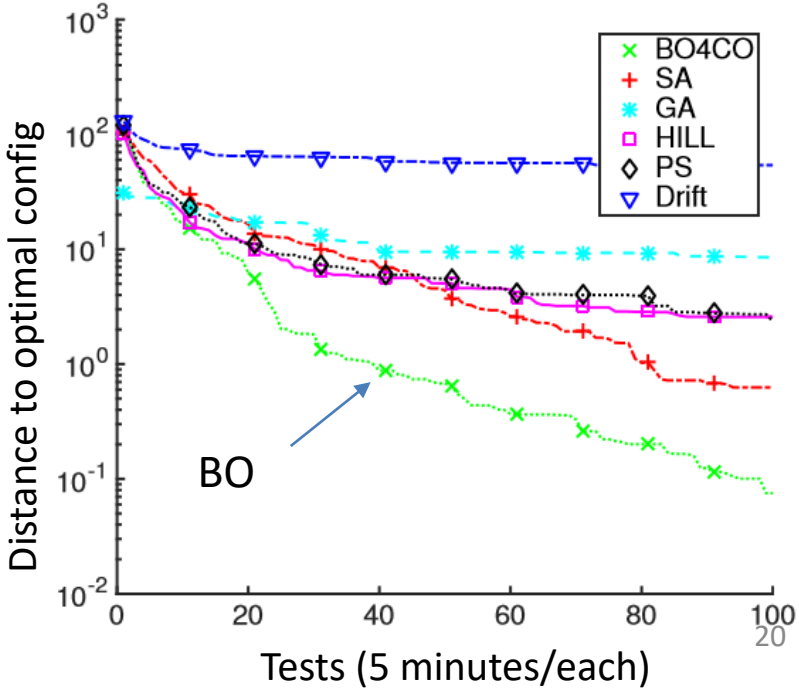
- Formally, a sequential blackbox optimization
  - Objective function is unknown before the experiments (e.g., response time)
  - Fixed budget of experiments
  - User inputs range of parameters
- Several techniques available
  - Design of experiments
  - Bayesian optimization
  - …



# CO case study: Storm-based DIA

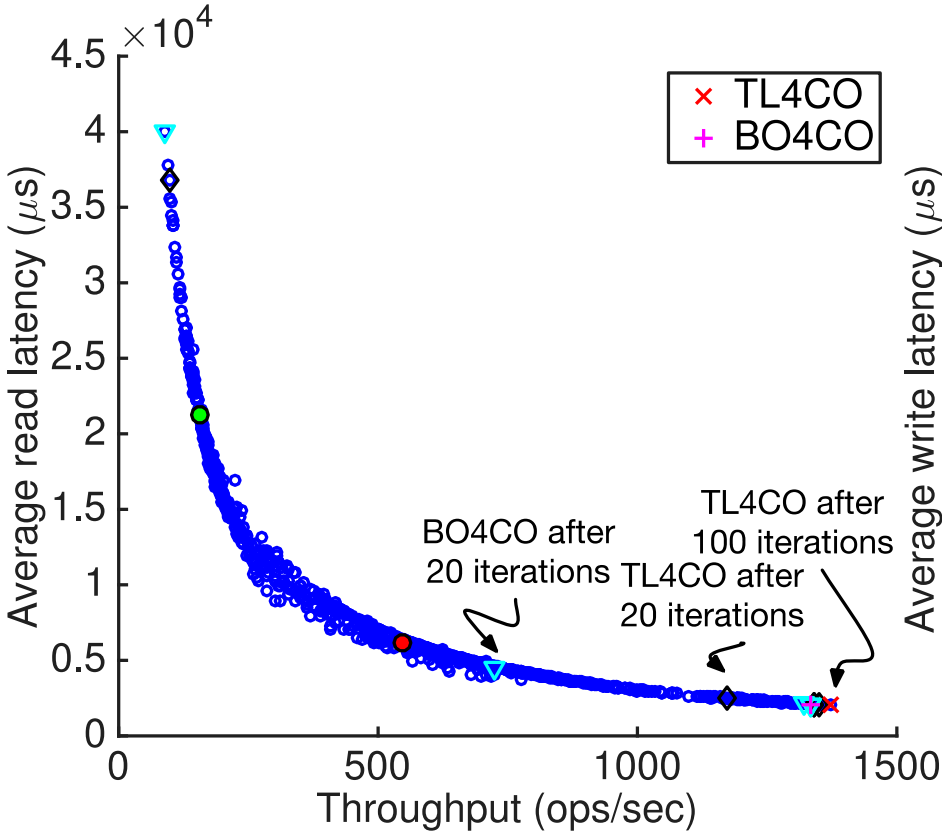
- 2.5 months to exhaustively test 4000 alternatives
- Bayesian optimization can find good configurations in just a few tens of tests



# Cassandra example

- Transfer learning methods helps reusing performance data from prior DIA releases



# DevOps with BD: Lessons learned

1. Works out of the box, little expertise needed
2. Risk needs to be a tunable parameter
3. People still want to understand how it works
4. Scalability to tens, not hundreds of parameters

# Thanks!

## Questions?