[simula . research laboratory]

SAGAR SEN, STEFANO DI ALESIO, DUSICA MARIJAN, ARNAB SARKAR, **SIMULA RESEARCH LABORATORY, OSLO, NORWAY**

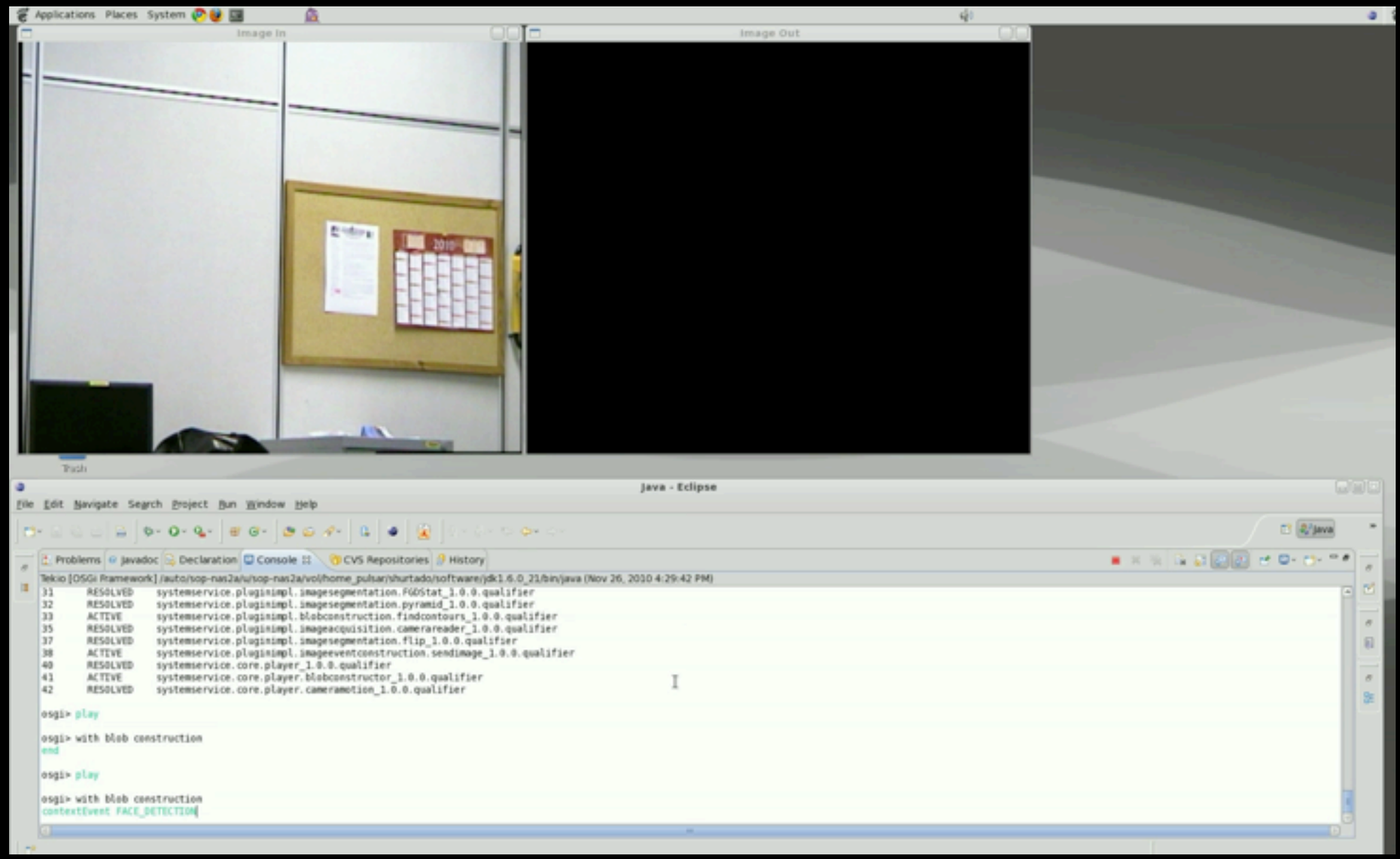
EVALUATING RECONFIGURATION IMPACT IN SELF-ADAPTIVE SYSTEMS
AN APPROACH BASED ON COMBINATORIAL INTERACTION TESTING

August 27, 2015 12h30-12h45, SEAA 2015, Funchal Madeira

OUTLINE

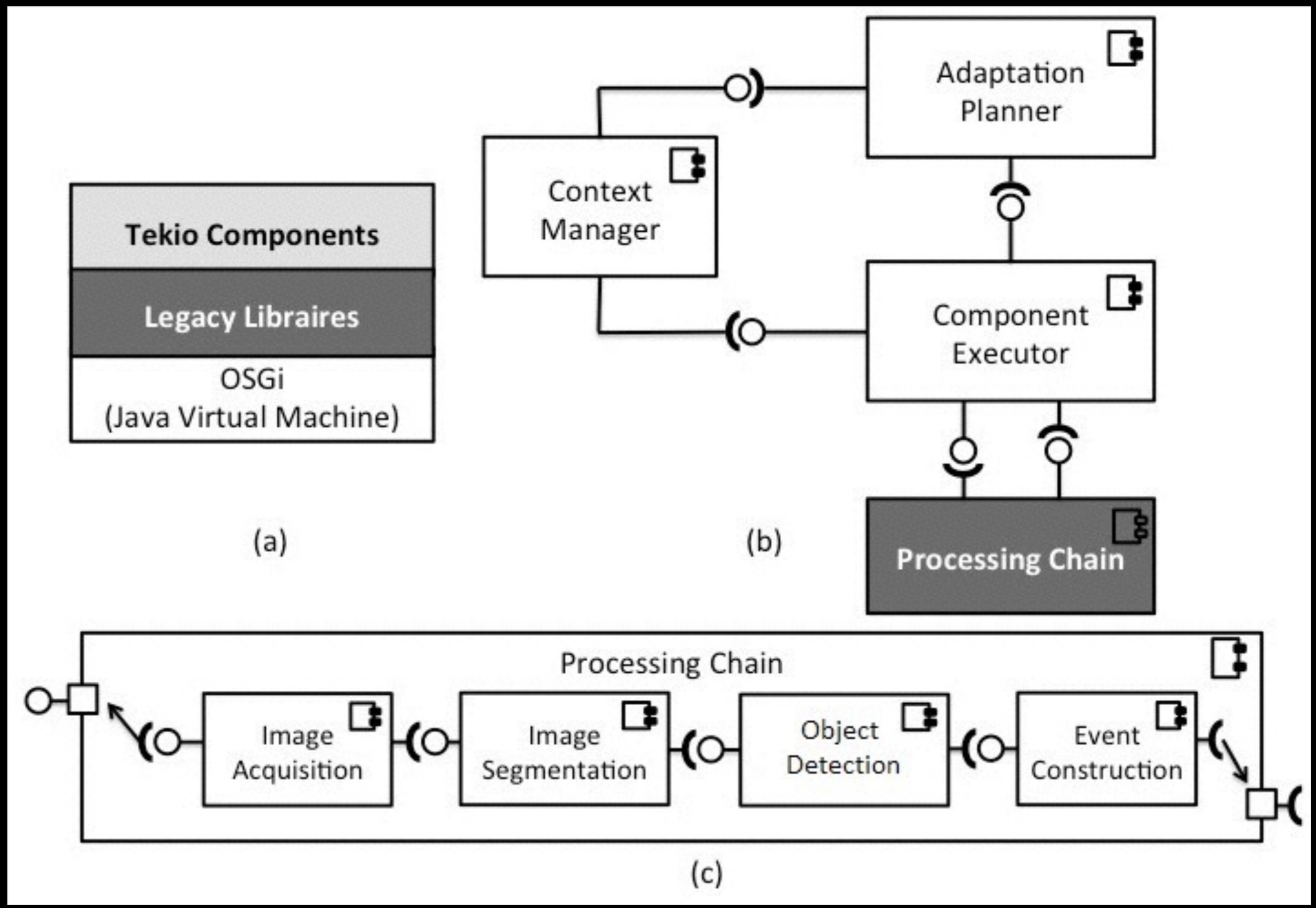
- **problem context**
- approach to generate test sequence of reconfigurations
- preliminary validation
- what impact can this work have?

TEKIO: A SELF-ADAPTIVE VISION SYSTEM



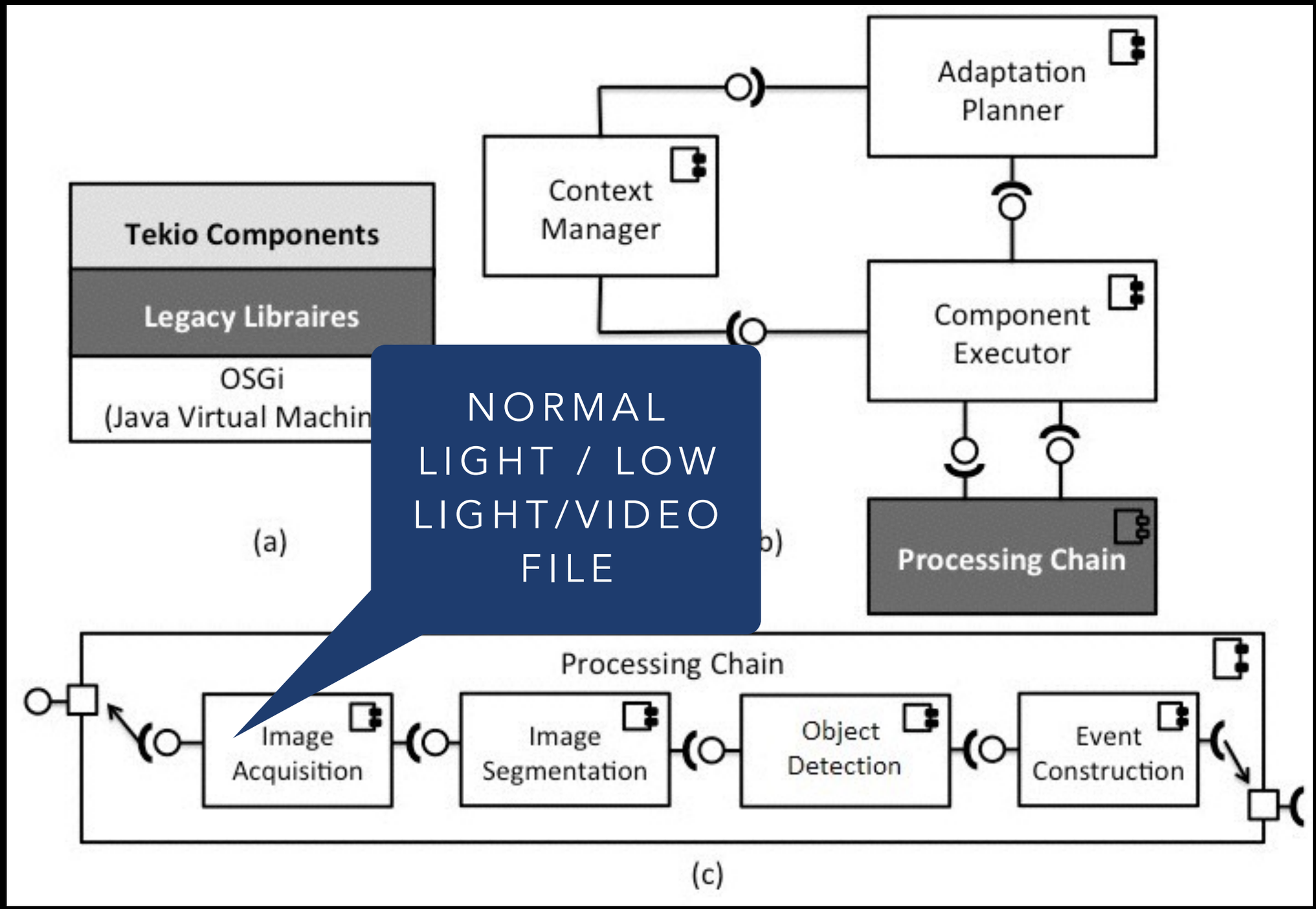
Software system that adapts due to change in operational context

TEKIO'S ARCHITECTURE

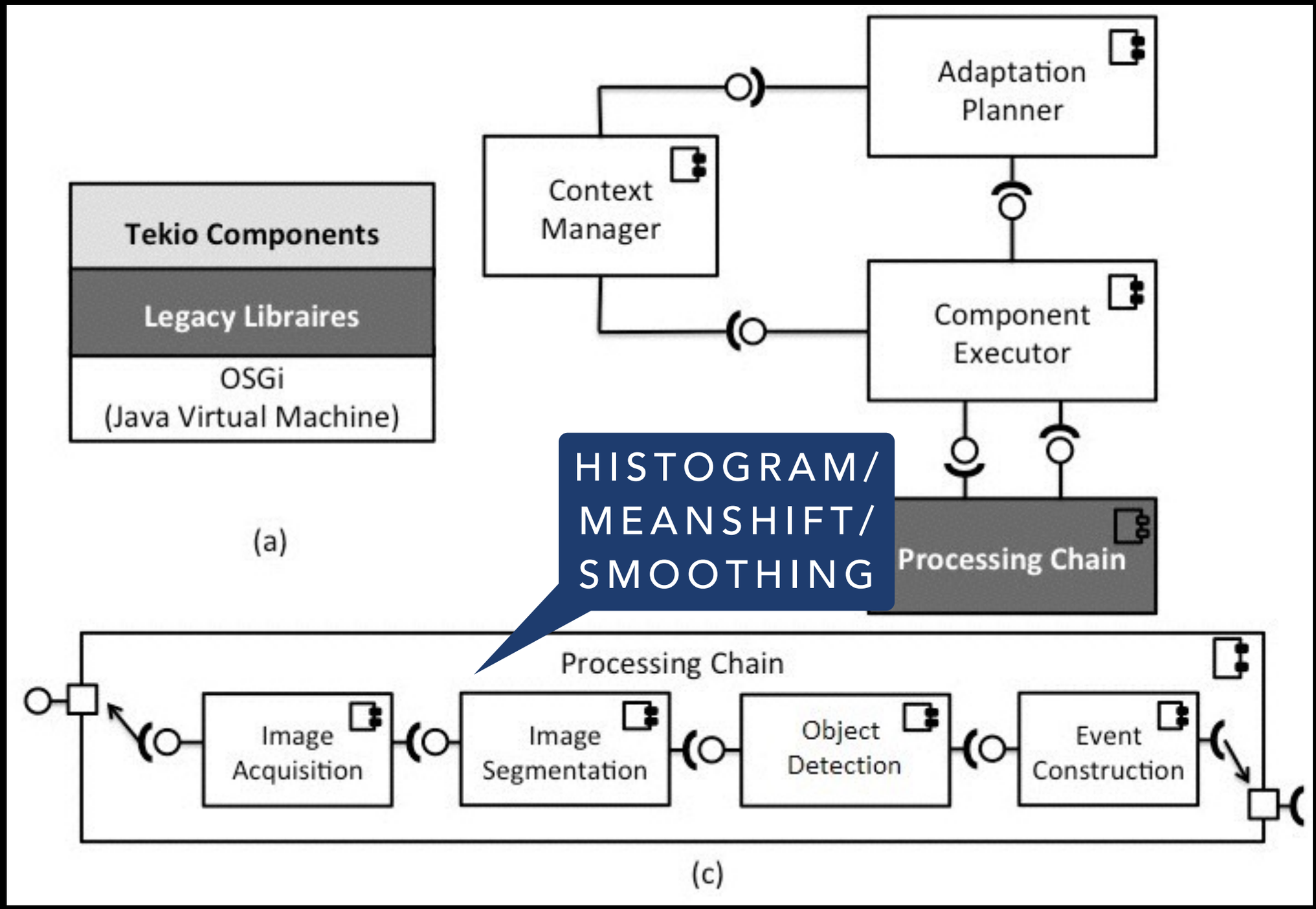


Santiago Hurtado, Sagar Sen, and Rubby Casallas. 2011. Reusing legacy software in a self-adaptive middleware framework. (ARM '11). ACM, New York, NY, USA, 29-35.

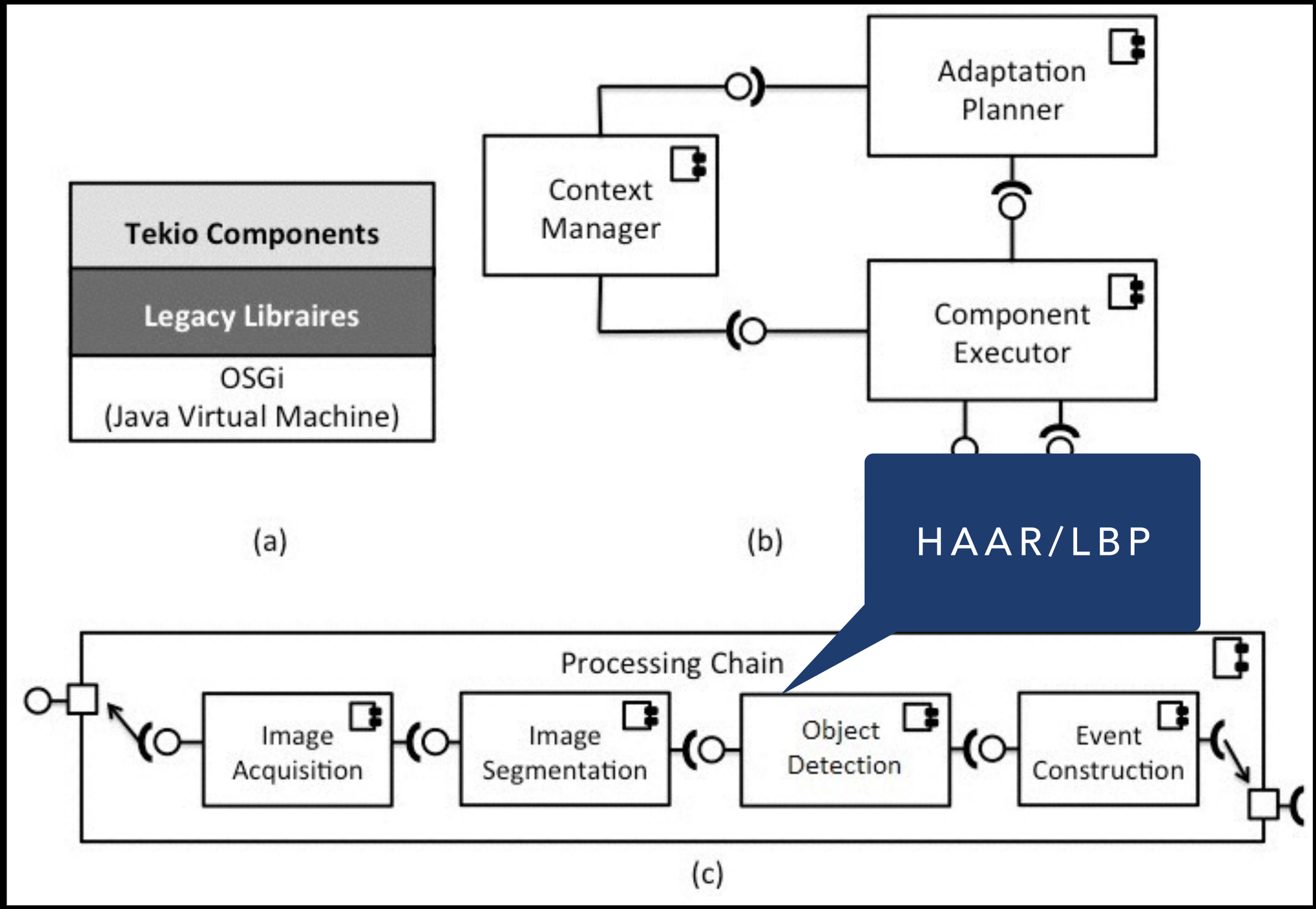
SEVERAL POSSIBLE WAYS TO CONFIGURE



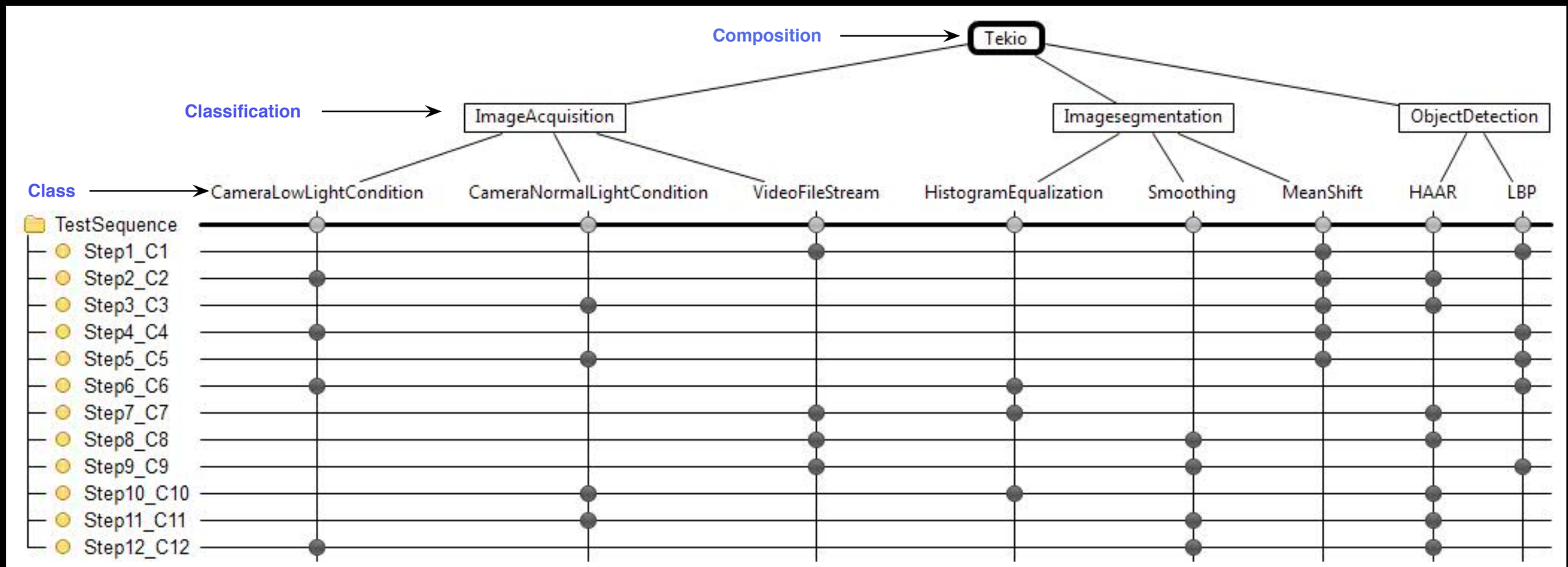
SEVERAL POSSIBLE WAYS TO CONFIGURE



SEVERAL POSSIBLE WAYS TO CONFIGURE



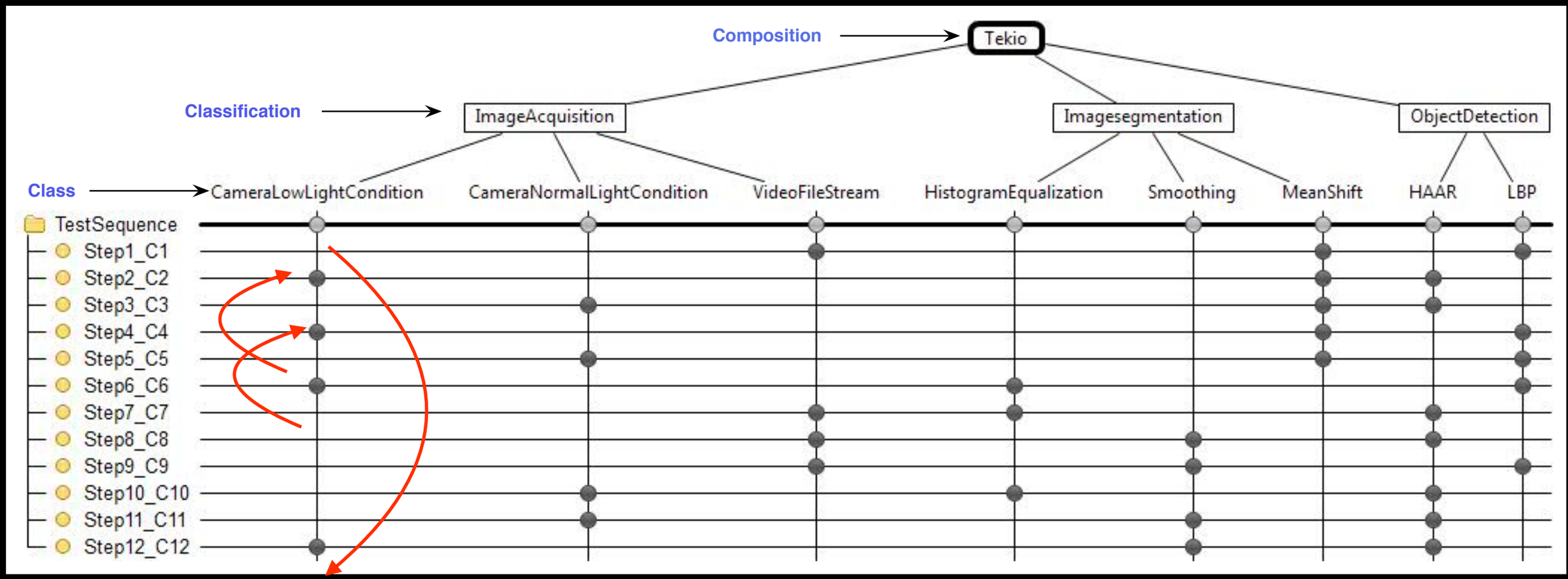
MODELLING VARIABILITY WITH CLASSIFICATION TREES



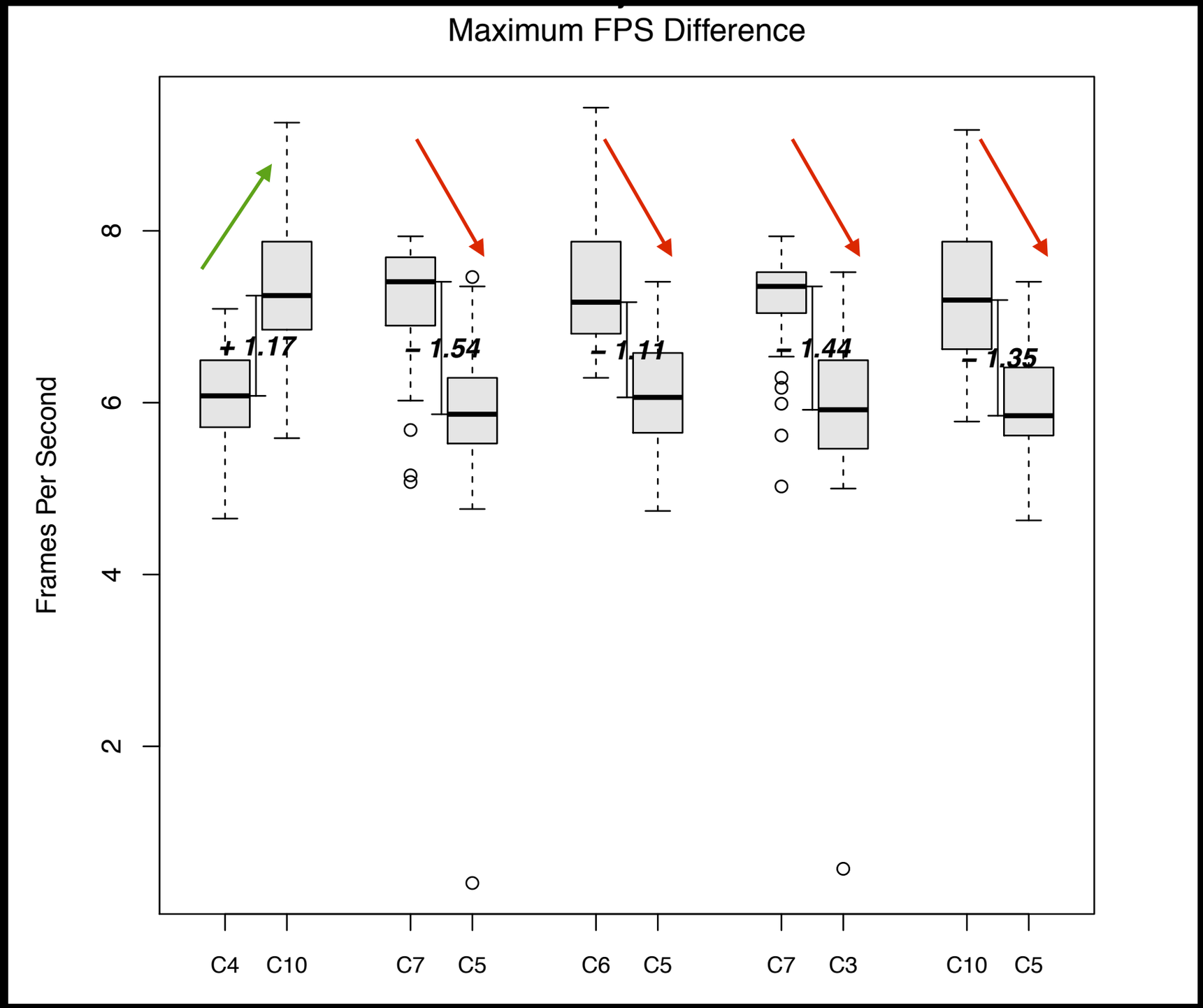
configurations

- What can happen when we **arbitrarily reconfigure** the self-adaptive system based on contextual changes?

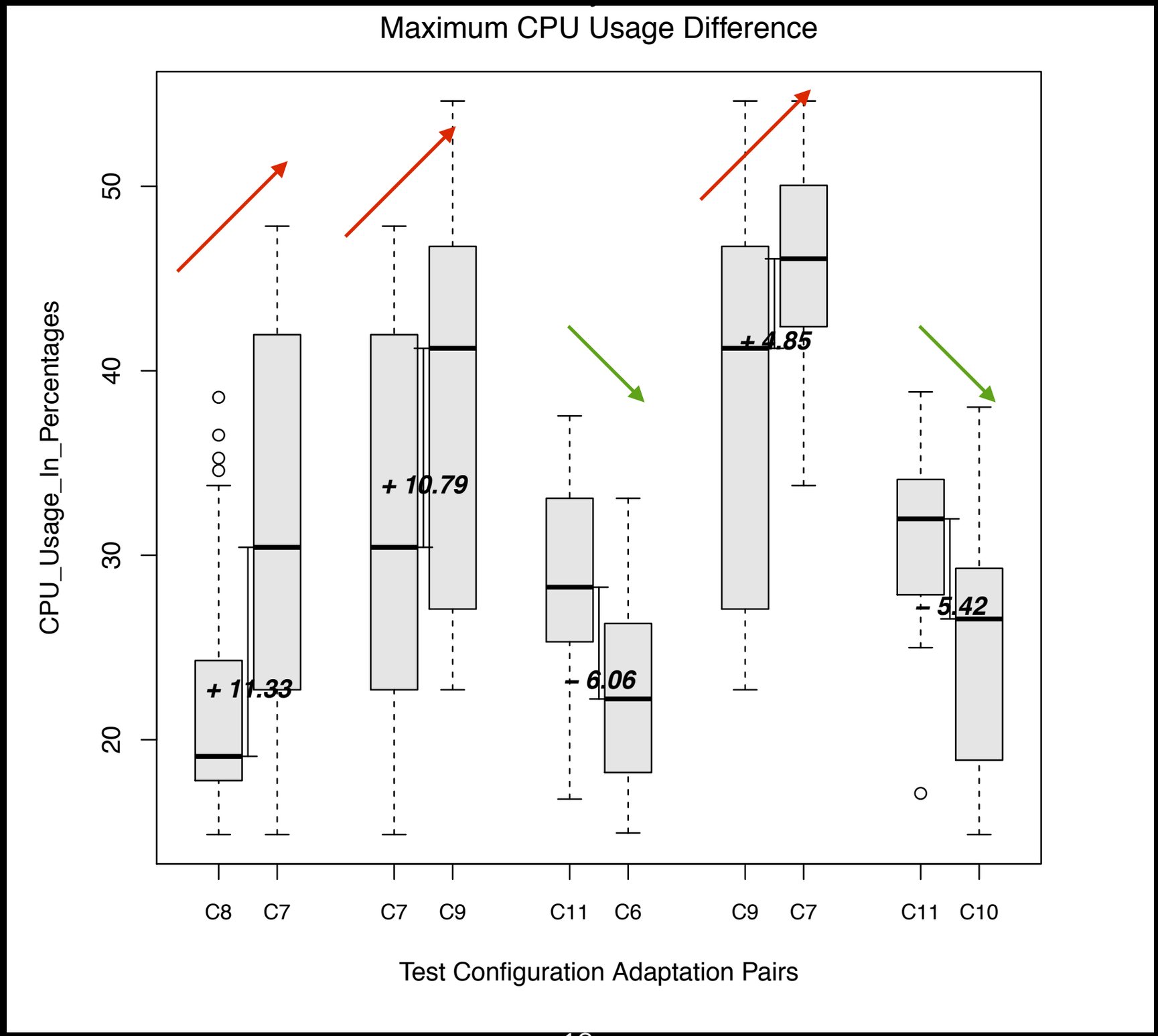
RECONFIGURATION



IMPACT ON QOS (FRAME RATE)



IMPACT OF QOS (CPU USAGE)



CHALLENGE

- How can we generate an **adequate test sequence** of reconfigurations such that we can **understand and evaluate reconfiguration impact** on QoS?

OUTLINE

- a challenge in testing self-adaptive systems
- **approach to generate test sequence of reconfigurations**
- preliminary validation
- what impact can this work have?

CONDITIONS FOR THE TEST SEQUENCE ADEQUACY

- It is a minimal sequence of **repeated configurations** of an adaptive system

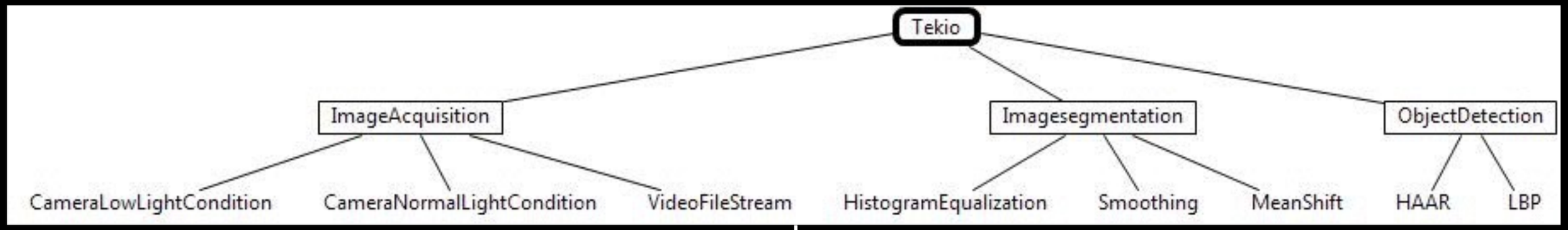
- Configurations in the sequence must cover **T-wise** (pairwise in most cases) interactions between variable/mutable features. Eg. interaction between object detection and image segmentation

- Sequence must **satisfy constraints between adaptable features** (such as only one of two possible object detection algorithms can be used in a configuration)

- The sequence must cover all valid **R-wise interactions** (hops) between configurations.

Step 1: Generating **configurations** covering T-wise interactions

TRANSFORMATION OF CLASSIFICATION TREE TO ALLOY



transform

constraint satisfaction problem in a lightweight formal method Alloy

ALLOY

- The Alloy language can represent a modelling domain such as classification trees in **first-order relational logic with quantifiers.**

- Alloy **signatures** define a finite set of **“atoms”** immutable named entities. Eg. The signature Integer has atoms -2,-1,0,1,2,..etc. in the a finite scope of 2.

- Alloy **facts, predicates, and functions** specify constraints between atoms as **“relations”.** Eg. The relation **“xor”** between two signatures is a 2-tuple relation

- Alloy Analyser transforms a modelling domain to 3-Conjunctive Normal Form to be solved by a SAT solver such as MiniSAT in a **finite scope.** The solver create atoms in the scope and determines **an assignment for all relations** called an **Alloy instance.**

- Alloy instances can then be **transformed back to a a set of configurations**

HIGHLIGHTS OF TRANSFORMATION

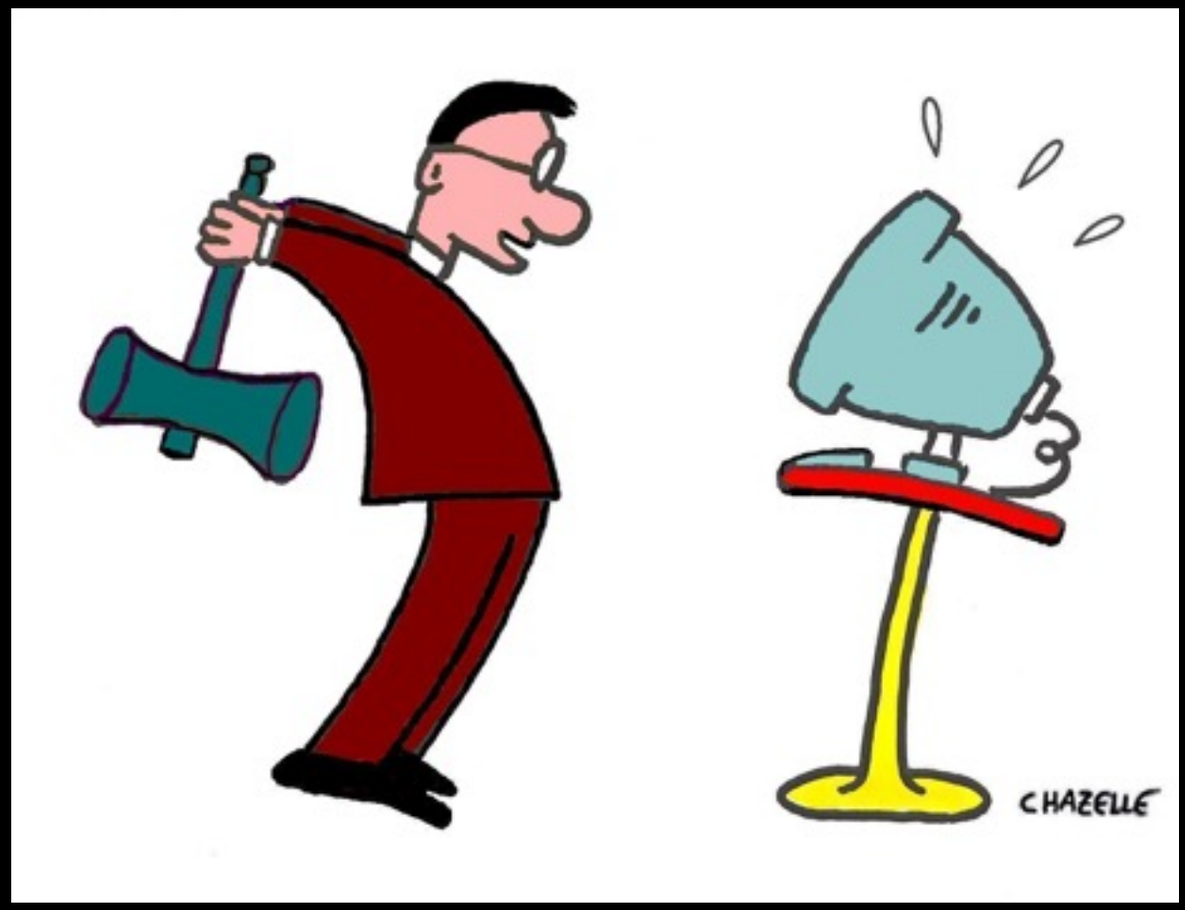
- **Features** of the adaptive system are Alloy signatures
- A **Configuration** signature is a relation towards a set of features
- A **ConfigurationSet** signature is relation to a set of configurations
- Constraints between features are transformed to **Alloy facts**
- T-wise combinatorial interactions between features are transformed to a set of 2^T C(N,T) **Alloy predicates**

(Details of transformation in the paper)

- Goal is to solve the Alloy model to obtain a **ConfigurationSet**

- With a **minimal** number of configurations that **satisfies all valid T-wise interaction predicates**

- **Intractable**



INCREMENTAL GROWTH OF CONFIGURATION SET

- We **incrementally grow** configuration sets until all T-wise predicates are solved in a **finite scope.**

- We **merge all configuration sets** into a one set of configurations that cover all T-wise predicates.

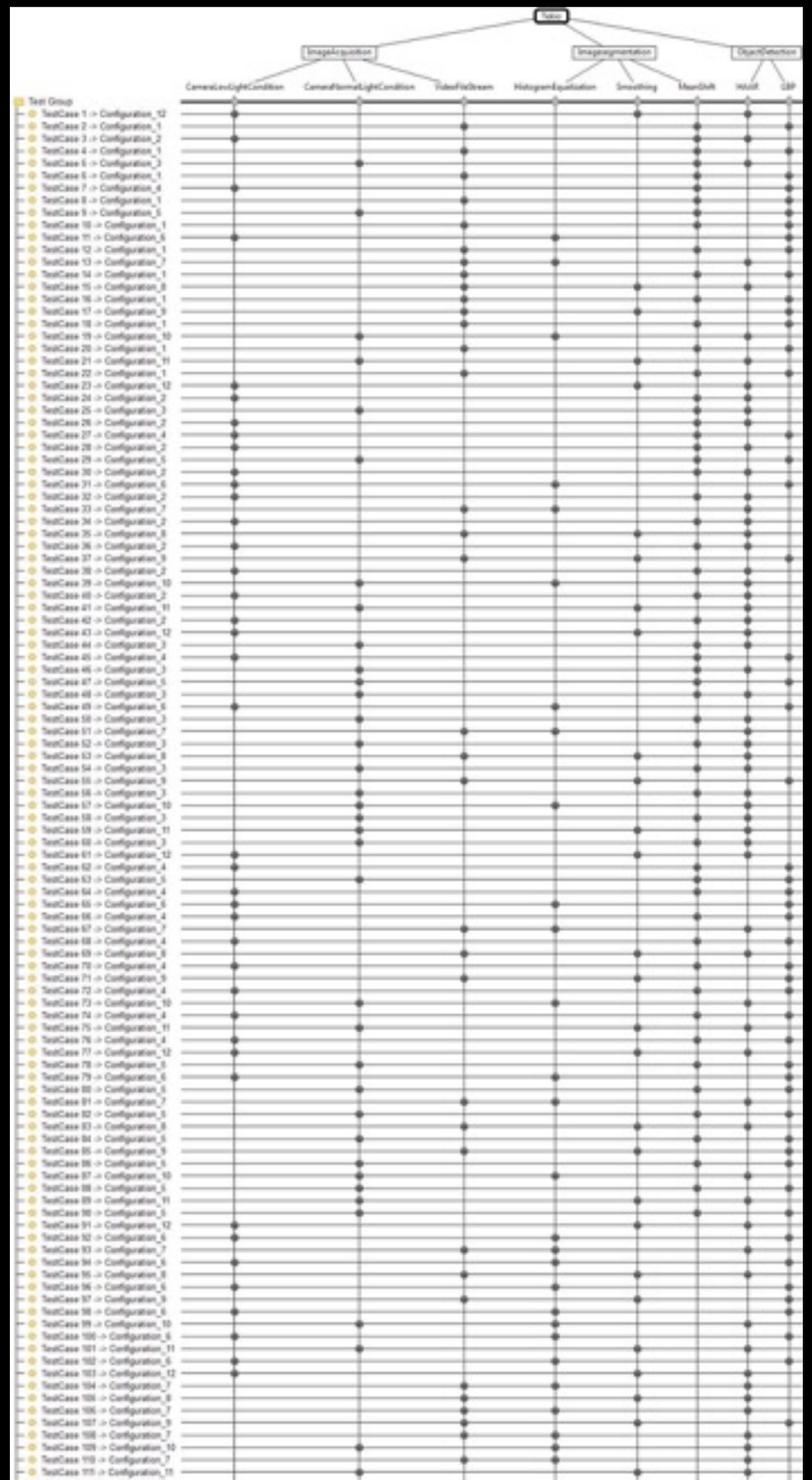
Step 2: Generating a **test sequence** covering R-wise hops between configurations from Step 1

TEST SEQUENCE THAT COVERS ALL R-WISE

- Given a set of **K** configurations that cover T-wise interactions between features we generate all **R-permutations** of the set of K configurations.

- Test sequence satisfying all R-wise permutations contains **K!/(K-R)!** reconfigurations

RESULT LOOKS LIKE THIS FOR TEKIO



132 Reconfigurations covering all **pairwise feature interactions** and **pairwise reconfigurations** between configurations for Tekio

OUTLINE

- a challenge in testing self-adaptive systems
- approach to generate test sequence of reconfigurations
- **preliminary validation**
- what impact can this work have?

PRELIMINARY VALIDATION

- We measured CPU Usage, Frame rate, and Memory usage in Tekio by running the sequence of 132 reconfigurations about 100 times

- We discovered **several critical reconfigurations** that consistently led to fluctuations in **CPU usage** and **frame rate.**

- These reconfigurations eventually helped develop **adaptation rules** for **stable QoS**

OUTLINE

- a challenge in testing self-adaptive systems
- approach to generate test sequence of reconfigurations
- preliminary validation
- **what impact can this work have?**

IMPACT

- (Self-)adaptive component-based systems empower **software reuse** and **evolution** and change behaviour at runtime

- However, this comes the “cost” of **unknown and unexpected behaviour**

- We aim to tame this unpredictability by understanding **reconfiguration impact** with **combinatorial interaction testing**

- **Combinatorial interaction testing** gives good **coverage** of behaviour and is successful in detecting unpredictability

- Not much better than **random testing**

Thank you!

sagar@simula.no