A summary of Runtime Verification @ UM

Christian Colombo

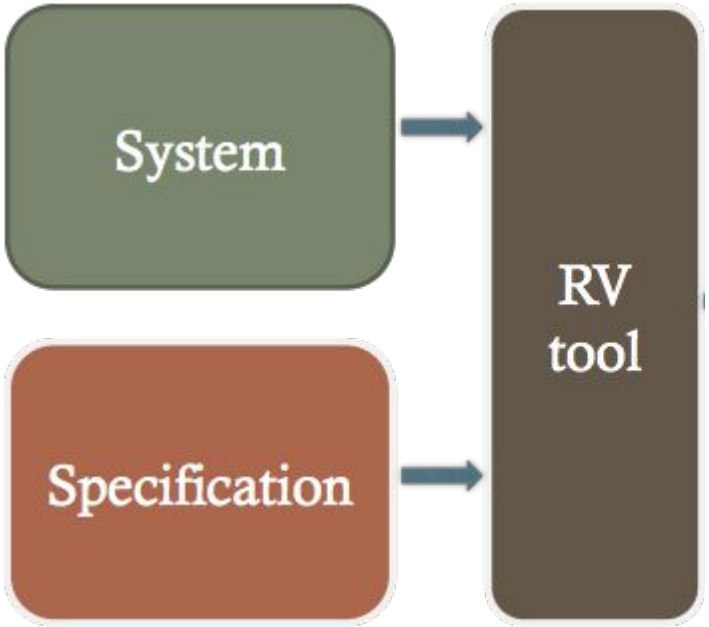
Kickoff Meeting October 2018

SPS G5448

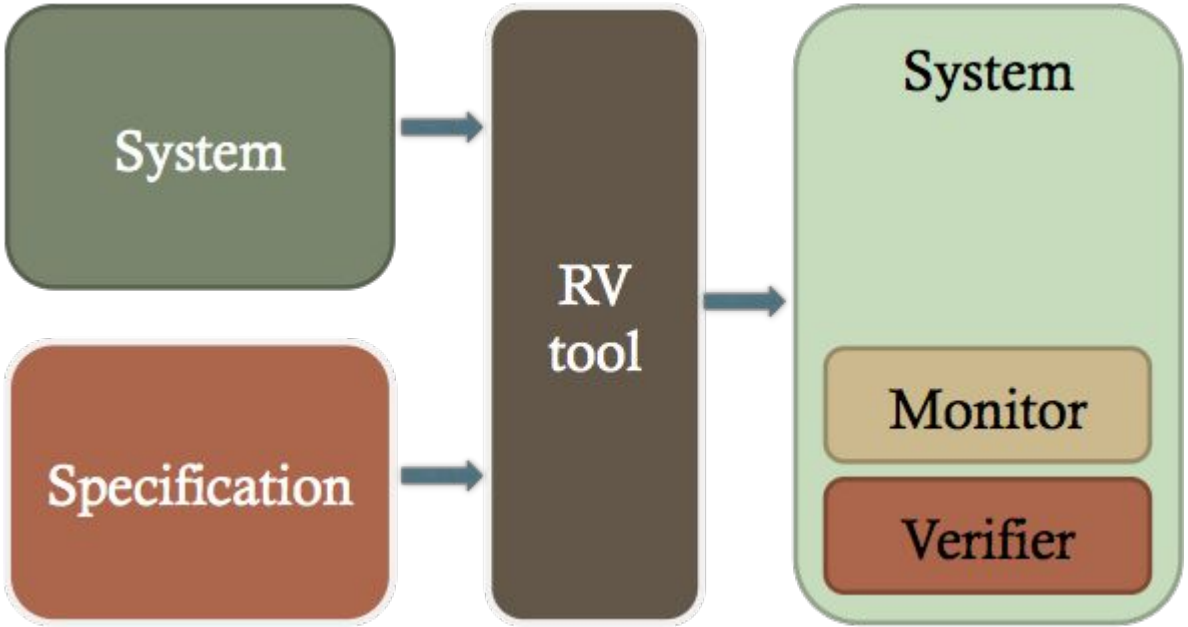
Secure Communication in the Quantum Era

Runtime Verification

Runtime Verification

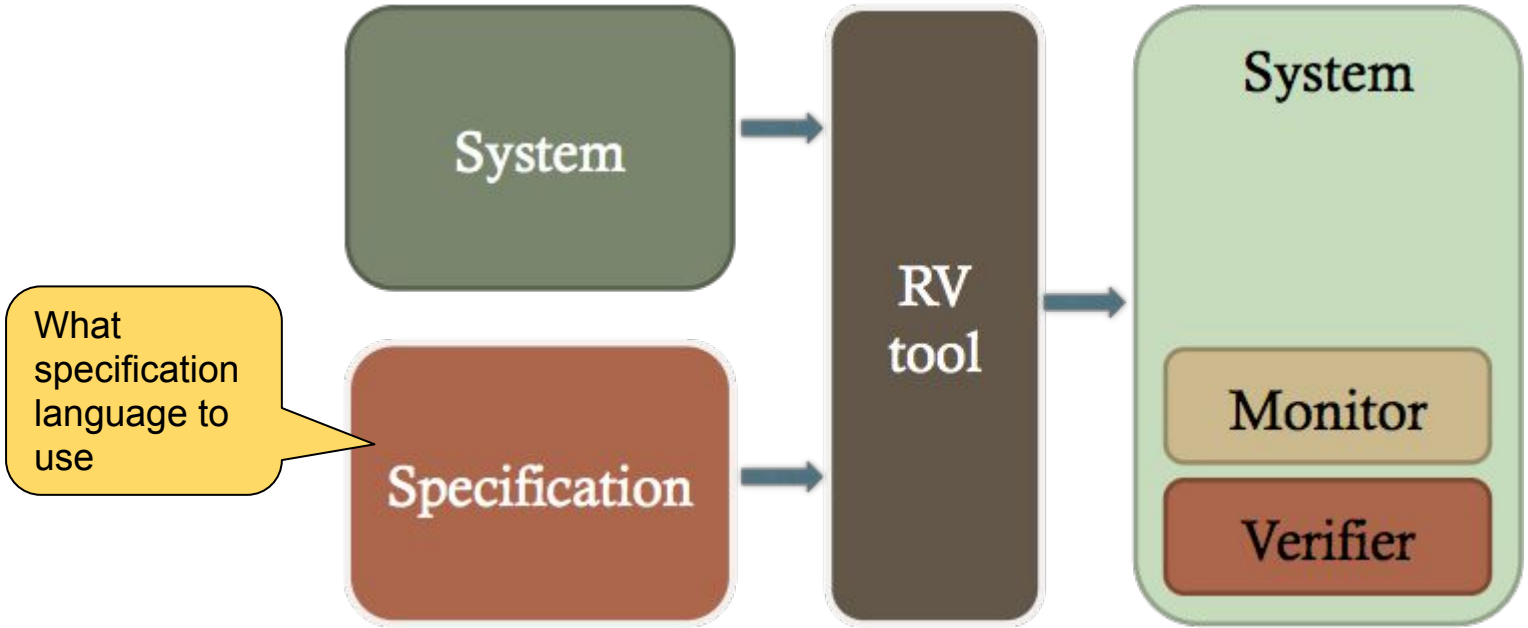


Runtime Verification

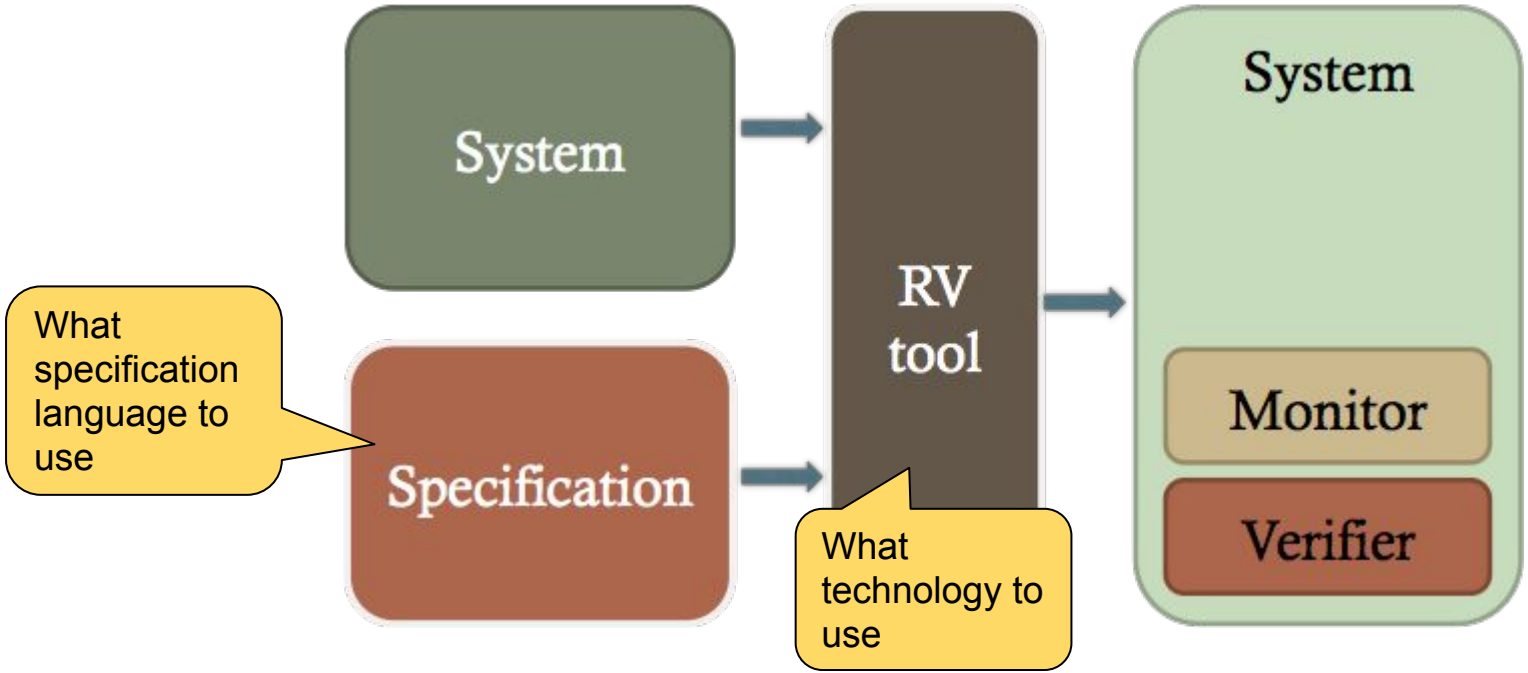


Design aspects

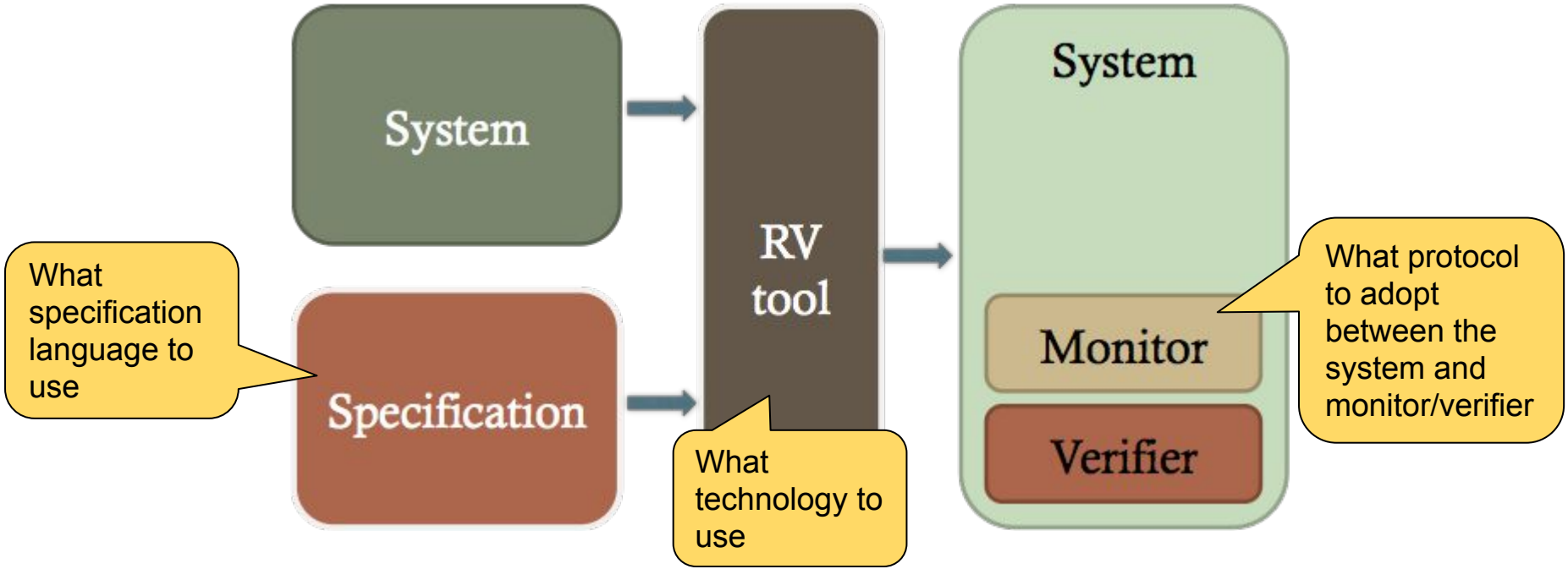
Runtime Verification



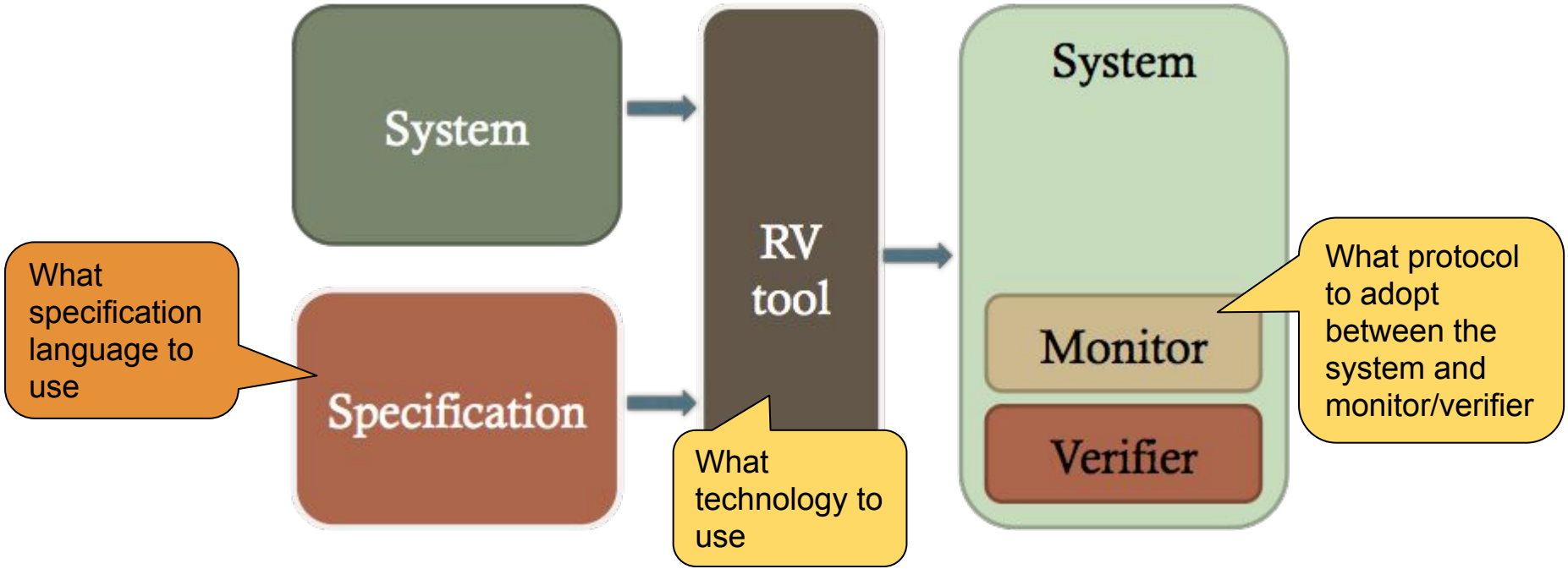
Runtime Verification



Runtime Verification



Runtime Verification



Specification Languages - Expressivity

Support for:

- Sequencing of events **“No write before a login”**
- Real-time **“Never more than 5 bad logins in 1 minute”**
- Per-object **“For each user, total spending cannot exceed €100 per day”**

Specification Languages - Understandability

Formats:

- Logics **(!login)* write**
- Automata (finite state machines)
- Domain-specific languages (sometimes as controlled natural languages)

Domain-specific languages

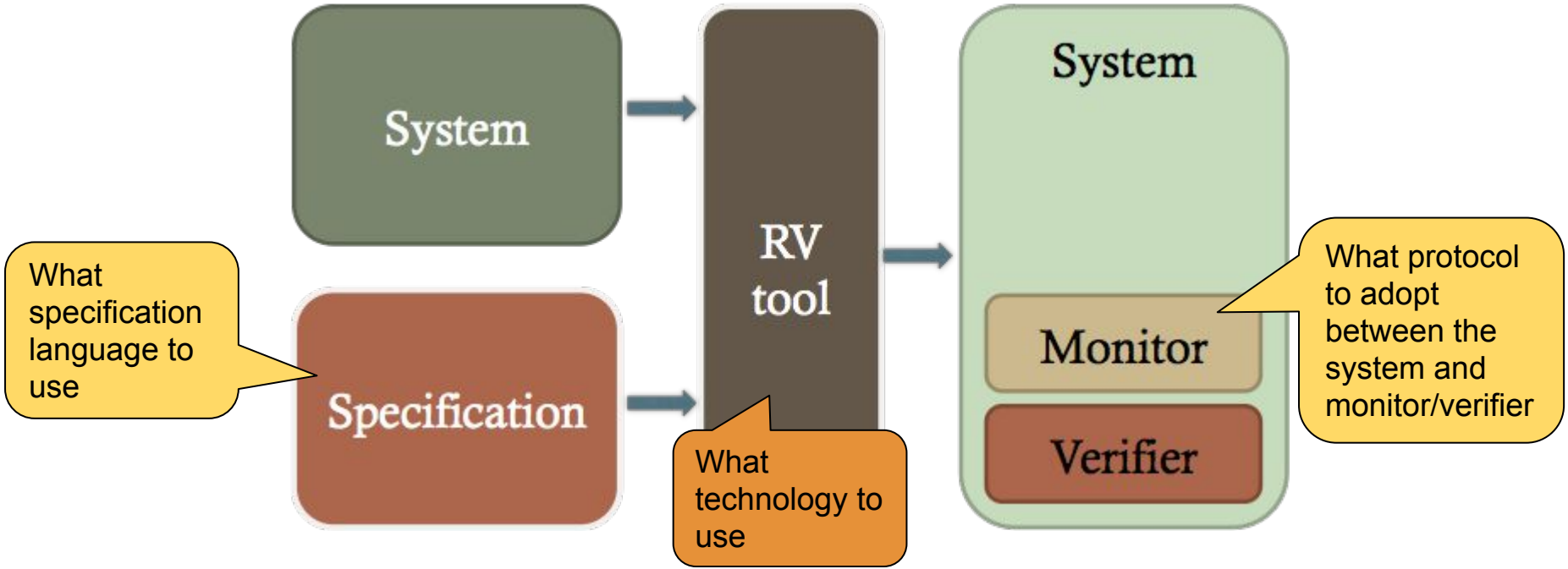
VISA regulations **“A non-verified cardholder can only spend €X per month”**

Tax fraud **“Tax payers declaring an income less than X% of the last Y-year average”**

Business intelligence **“Alert me whenever a post gets more than X negative comments in Y minutes”**

Fraud risk **“Increase risk score by X% for each transfer from country Y with amount greater than Z”**

Runtime Verification



Technologies

Java + AspectJ

Java + Kafka/RabbitMQ/etc

Erlang

C

A mixture of technologies

Architectures

Monolith systems

Distributed systems with a global clock

Distributed systems

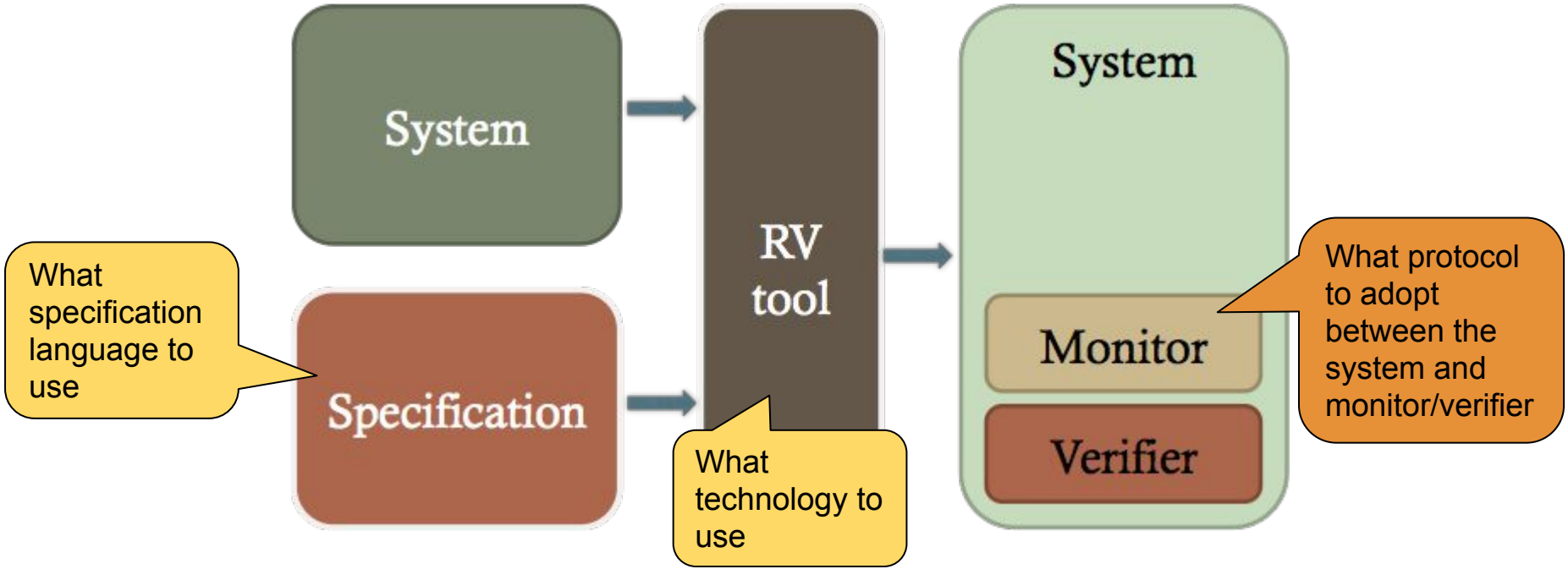
Ways of obtaining events

By modifying the code

By intercepting communication

From a data source (eg: database)

Runtime Verification



Monitor and System work in parallel?

System and monitor wait for each other

System runs independently of the monitor

What happens when a problem is detected?

The monitor simply raises an alert

The monitor can “fix” the situation

Objectives

Easy expression of specifications

Easy to modify

No experts needed

Correctness

The behaviour of the system (with the introduction of the monitor) should not be modified

The monitor only/always raises an alert when it should

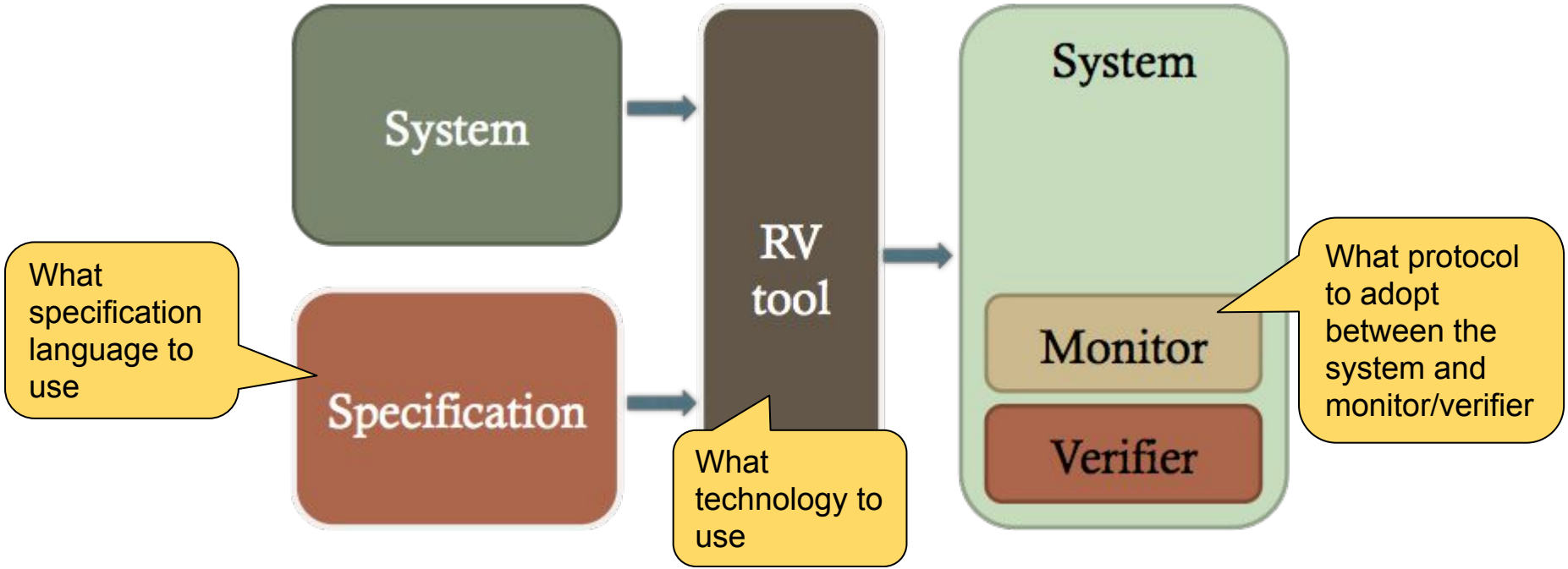
Reducing overheads

Make it as non-intrusive as possible

Acceptable slowdown

Security?

Runtime Verification



What specification language to use

What kind of properties do we need to check for?

Who will write these specifications?

How frequently they are likely to change?

What technology to use

Which technologies will the monitor be interacting with?

Is distribution an issue?

How can events be intercepted and communicated (securely!)?

How can the monitor be secured?

Protocol

Do we need to respond (timely) to unexpected behaviour?

If yes, what kind of response is envisaged?