

Runtime Verification as a Toolkit of Techniques for Cyber Security Monitoring

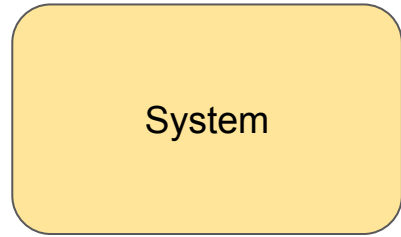
Christian Colombo

University of Malta


Buenos Aires, November 2023

What is Runtime Verification?

A General Picture

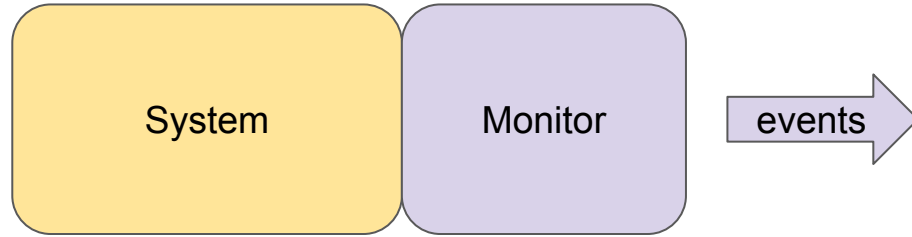


A General Picture

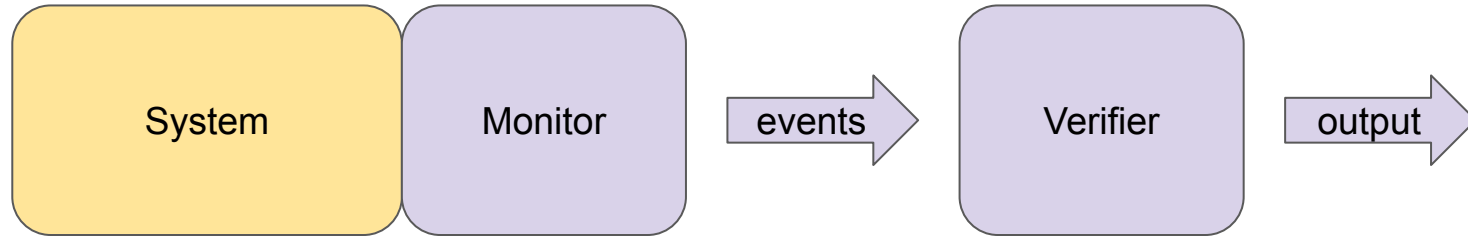


Very loose definition of
a “system” ...
server/client
banking/robotics/etc
sequential/concurrent/
distributed

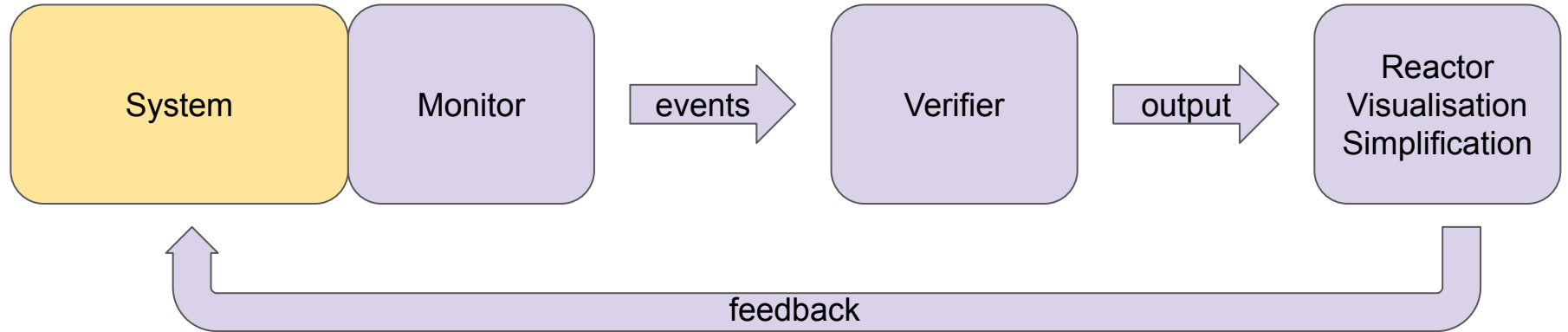
A General Picture



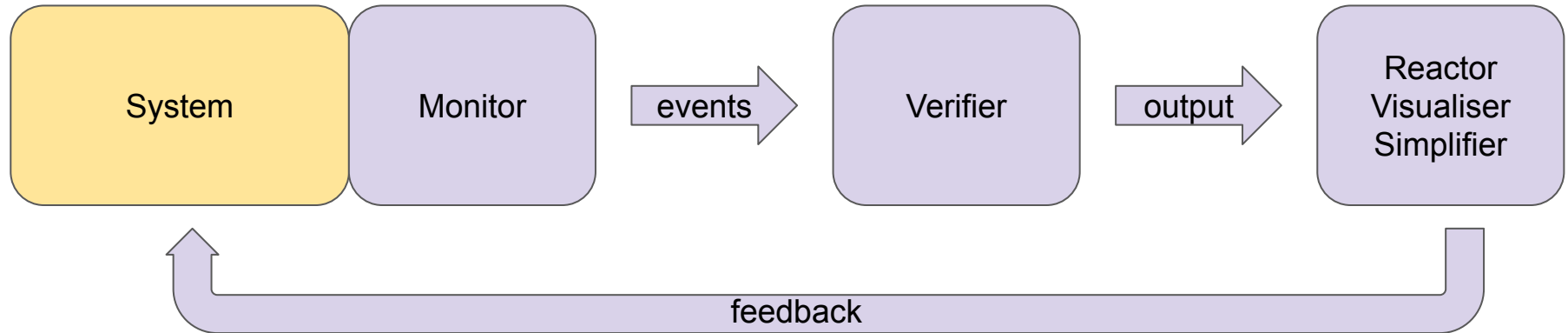
A General Picture



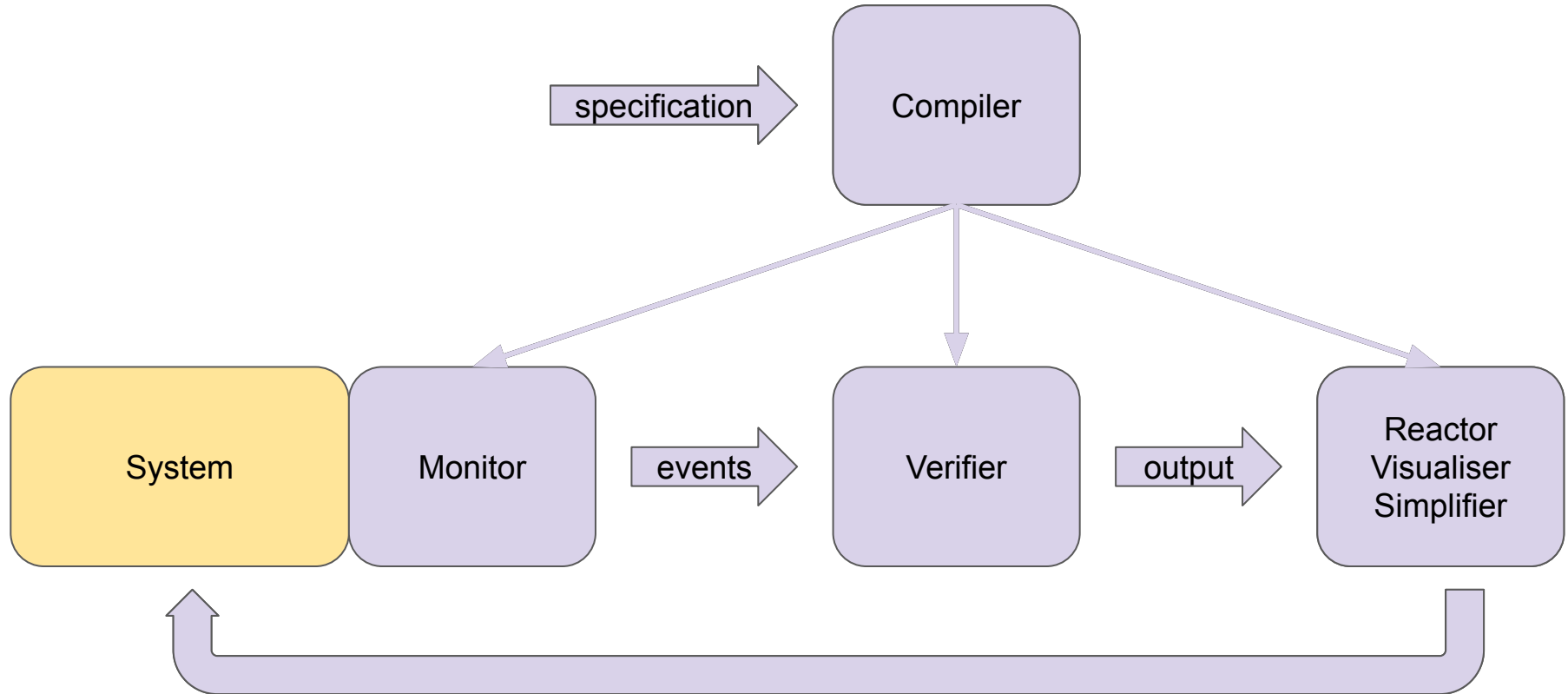
A General Picture



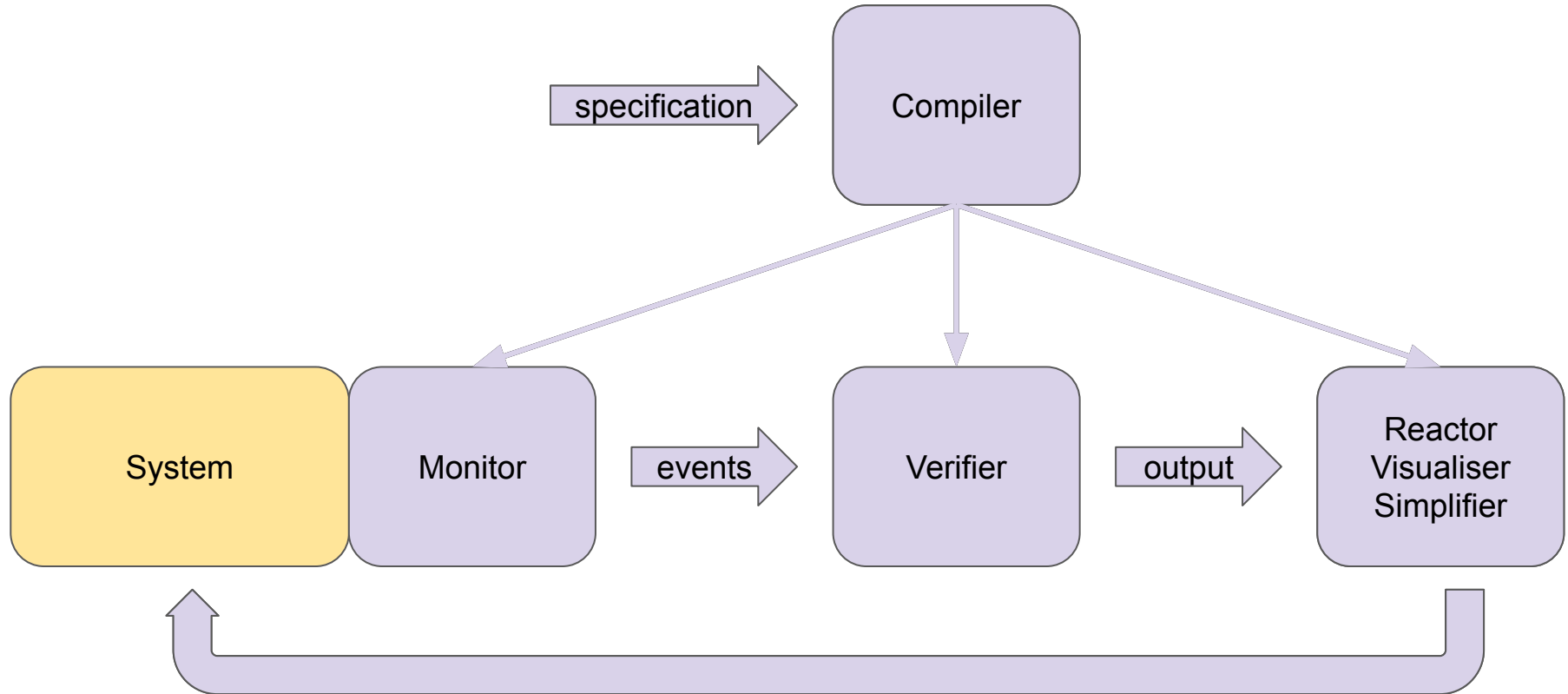
What is Special?



What is Special? (1) - Automatic Synthesis



What is Special? (2) - Separation of Concerns



Why Runtime Verify?

As a support tool during testing

- Some oracles might be hard to implement

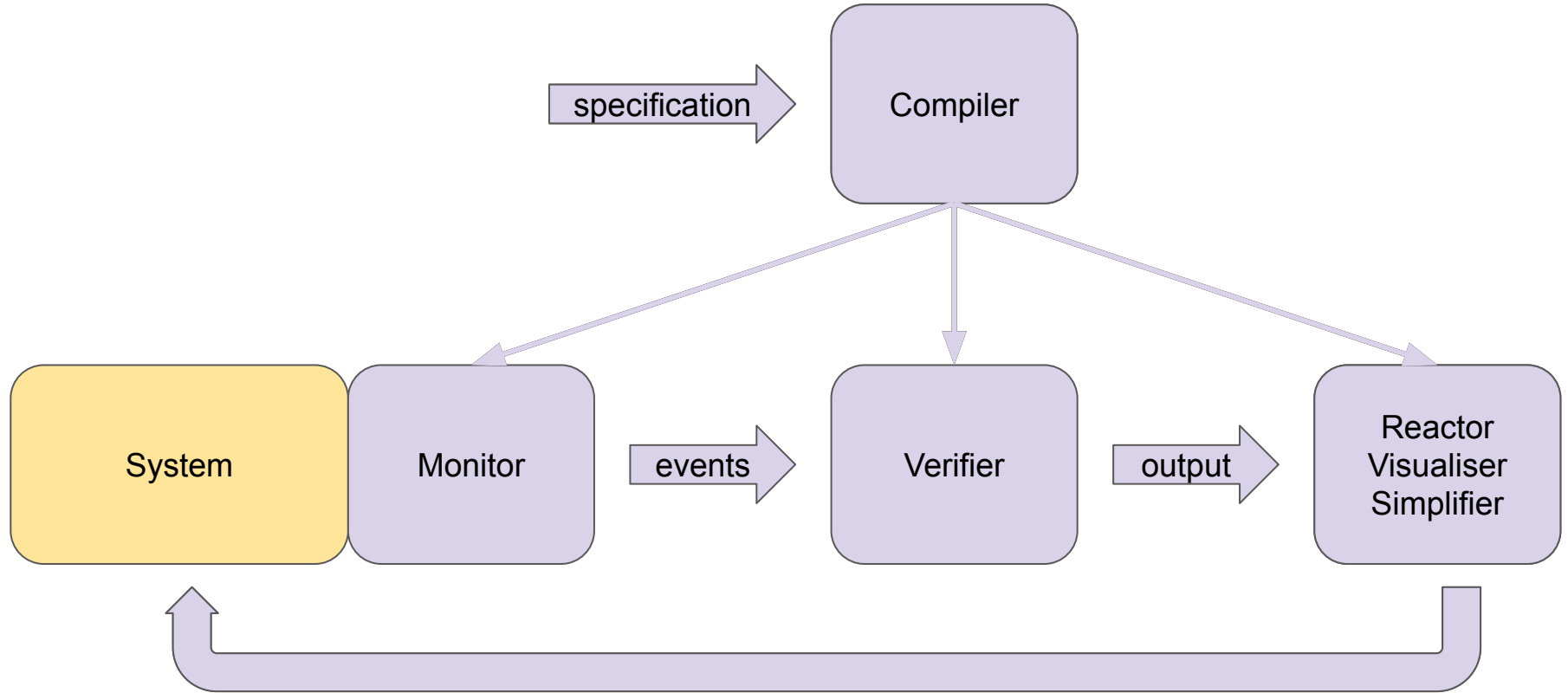
For detecting bugs

- Not always feasible to fully verify/test system before runtime

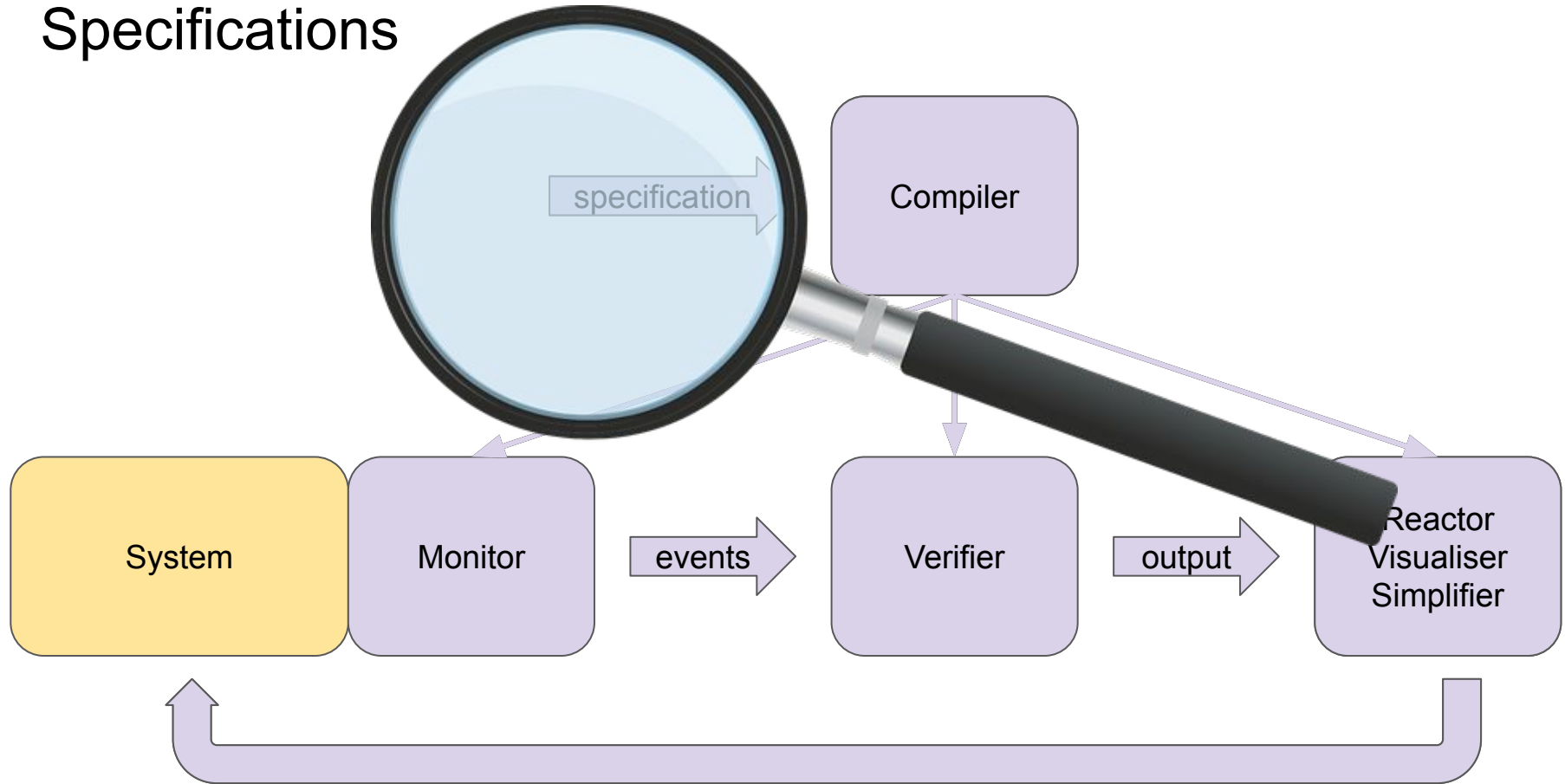
For general stream processing/pattern recognition

- Sometimes it's convenient

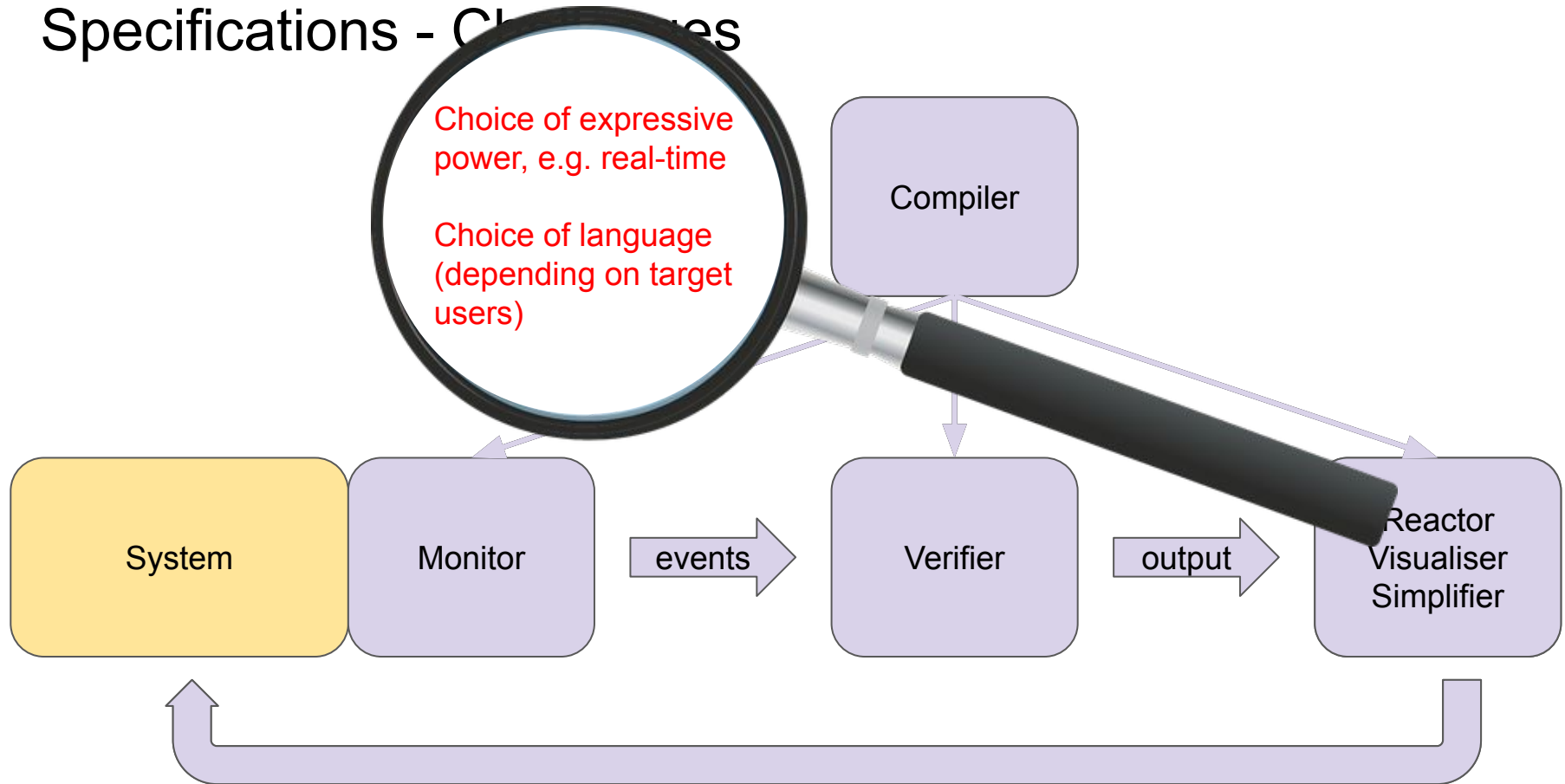
Zooming in



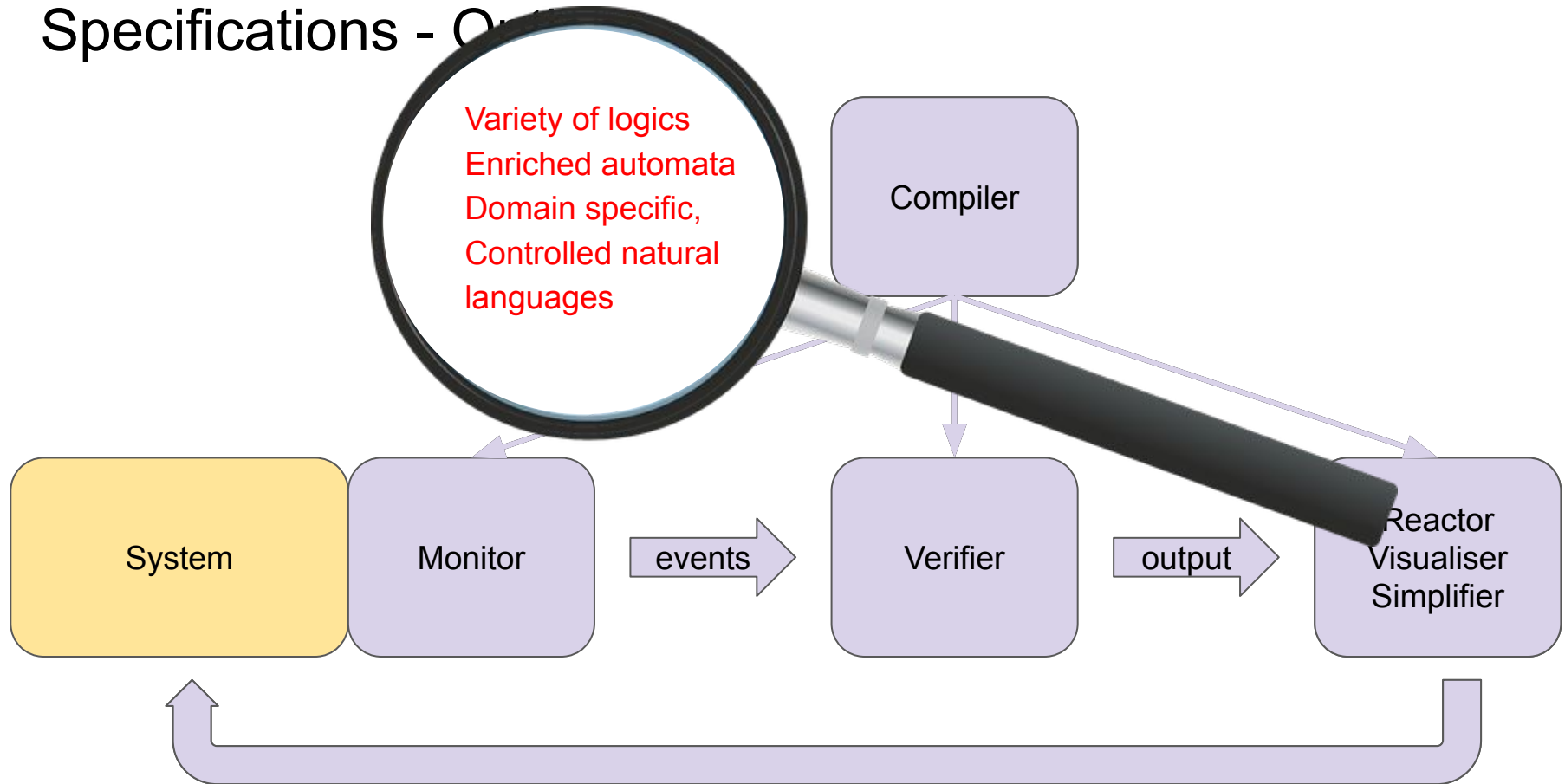
Specifications



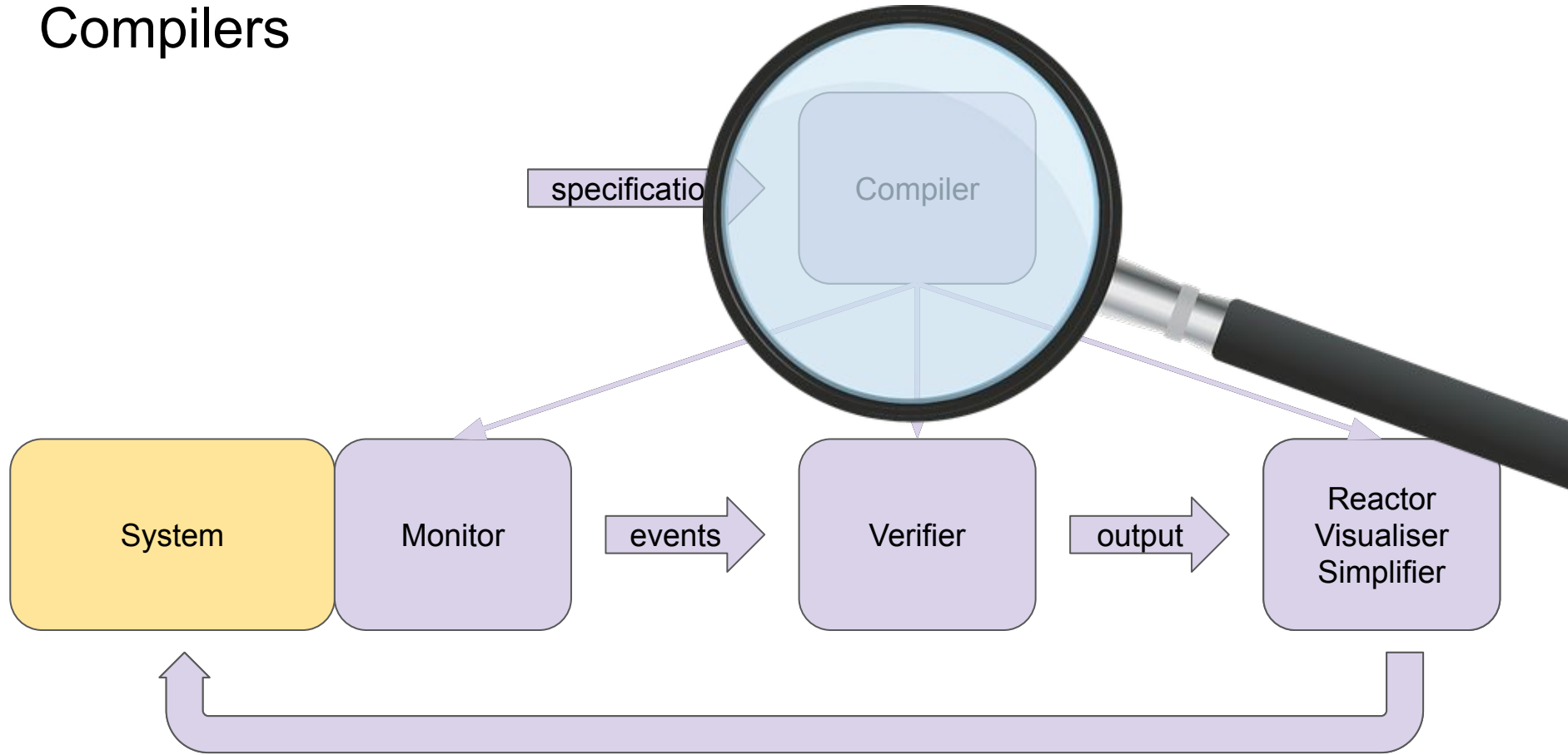
Specifications - Choices



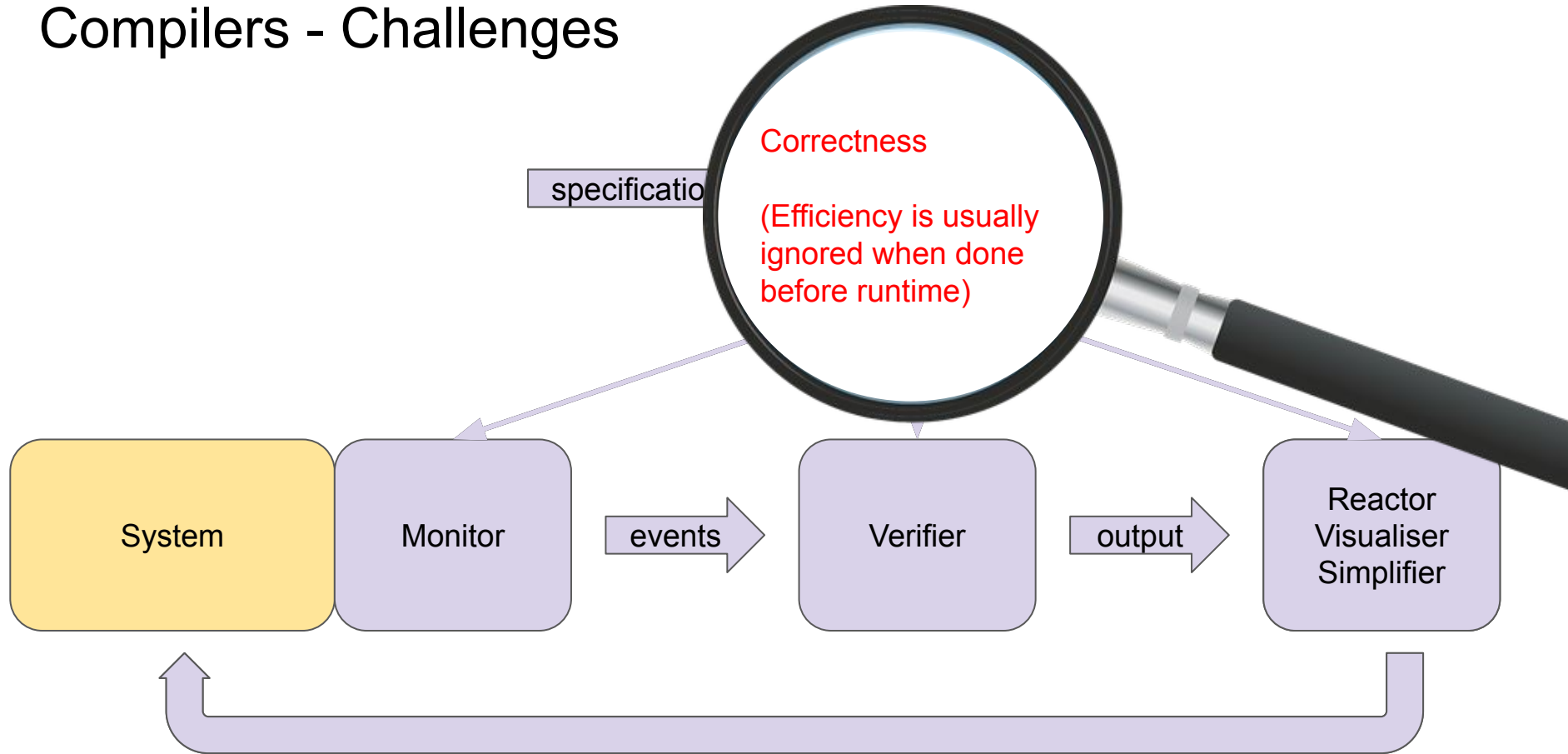
Specifications - Overview



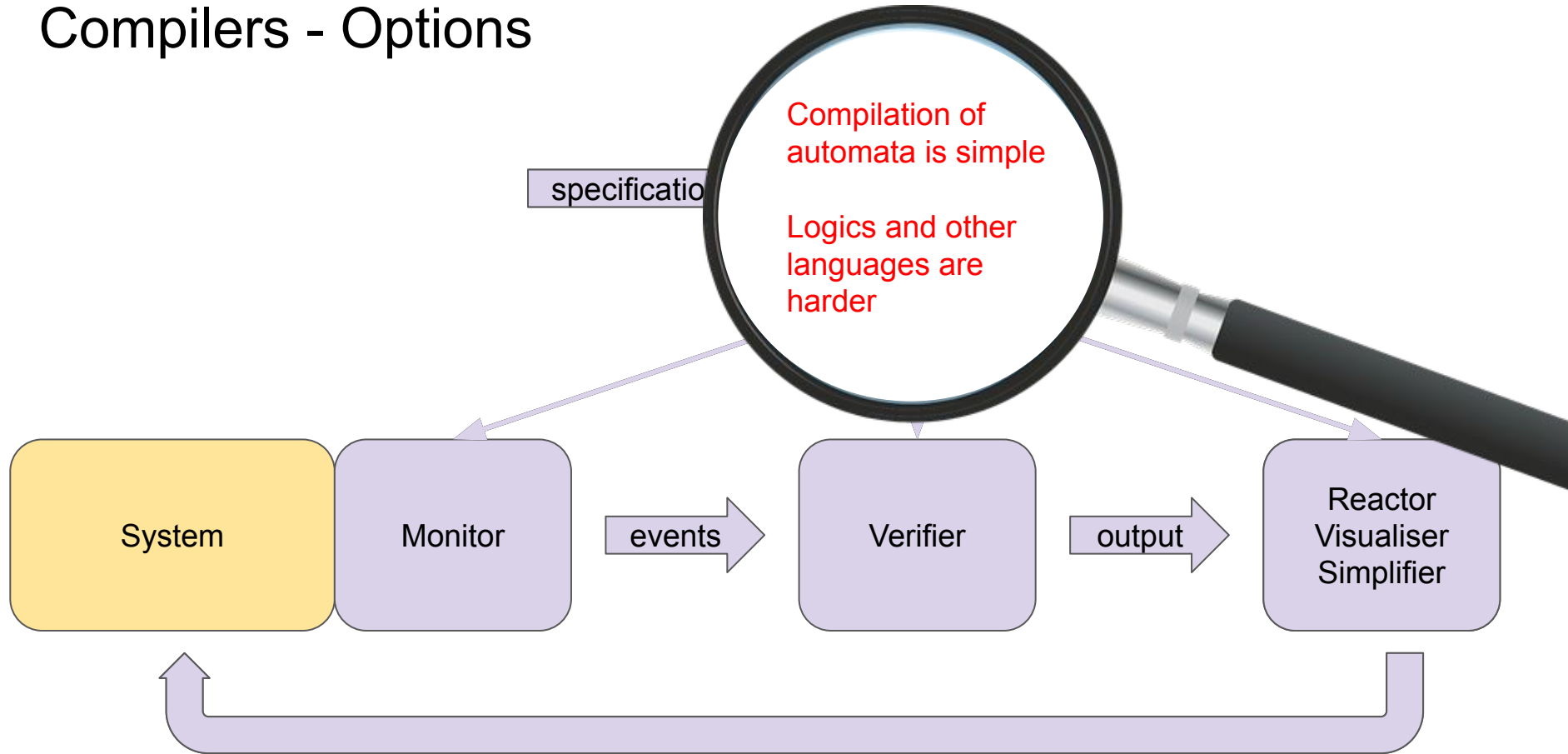
Compilers



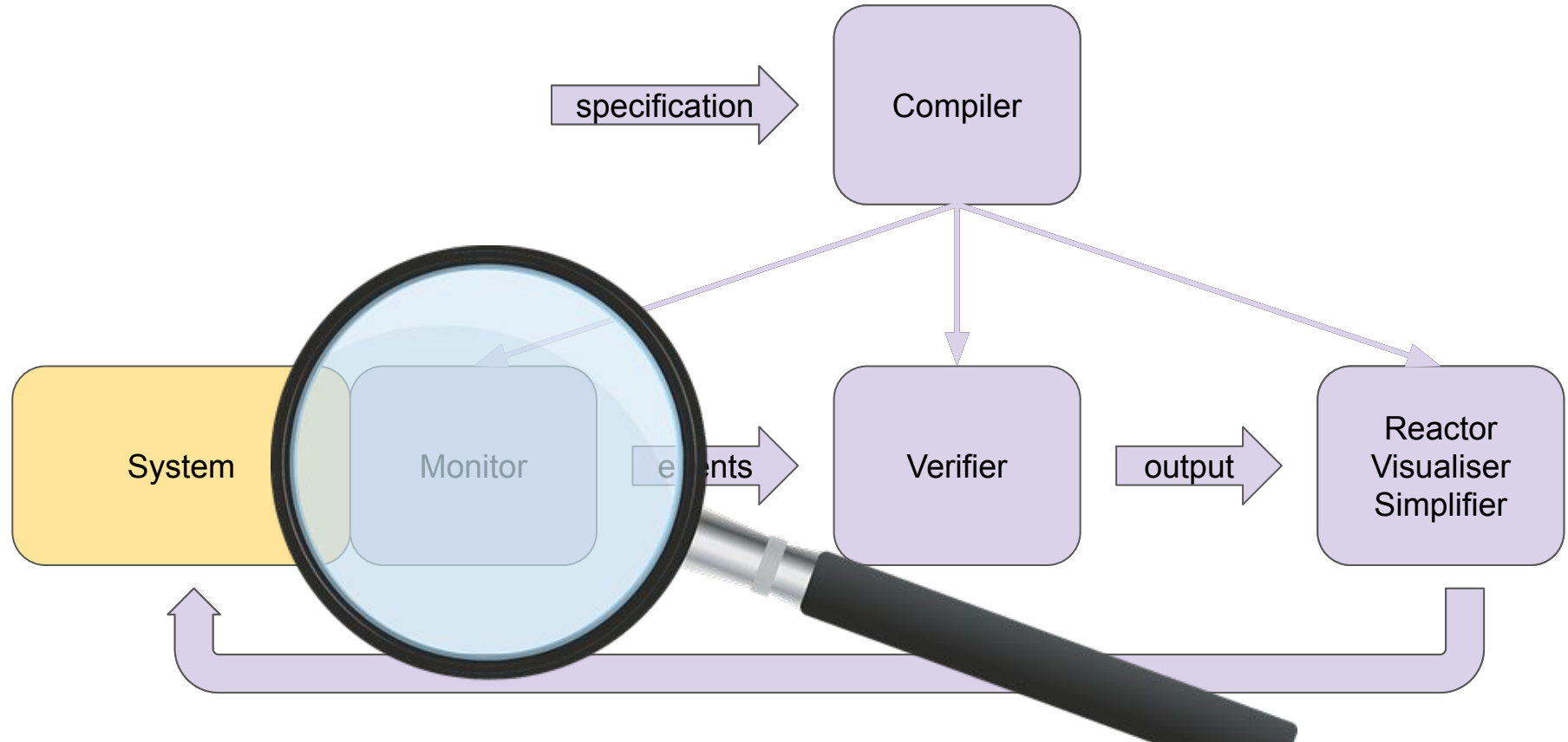
Compilers - Challenges



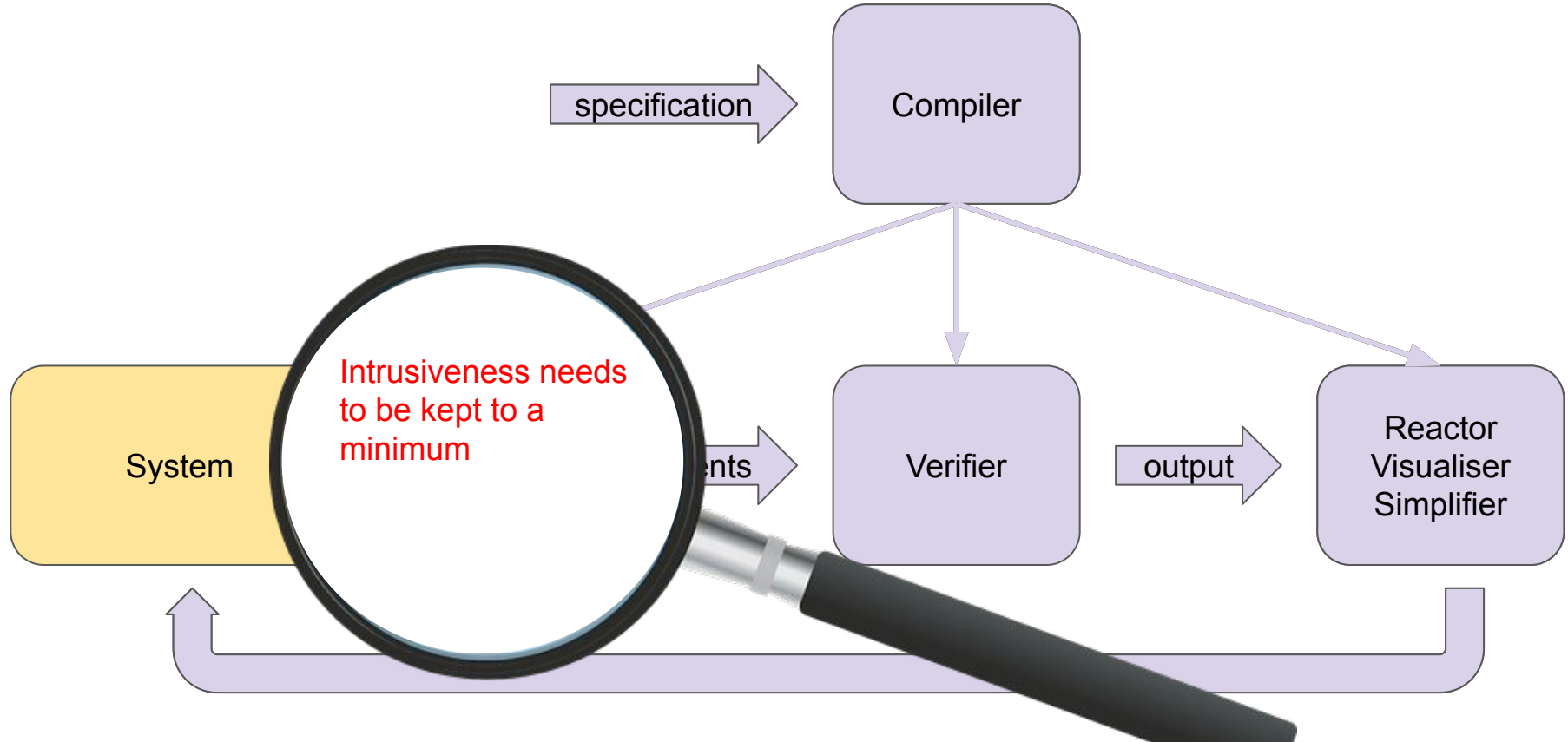
Compilers - Options



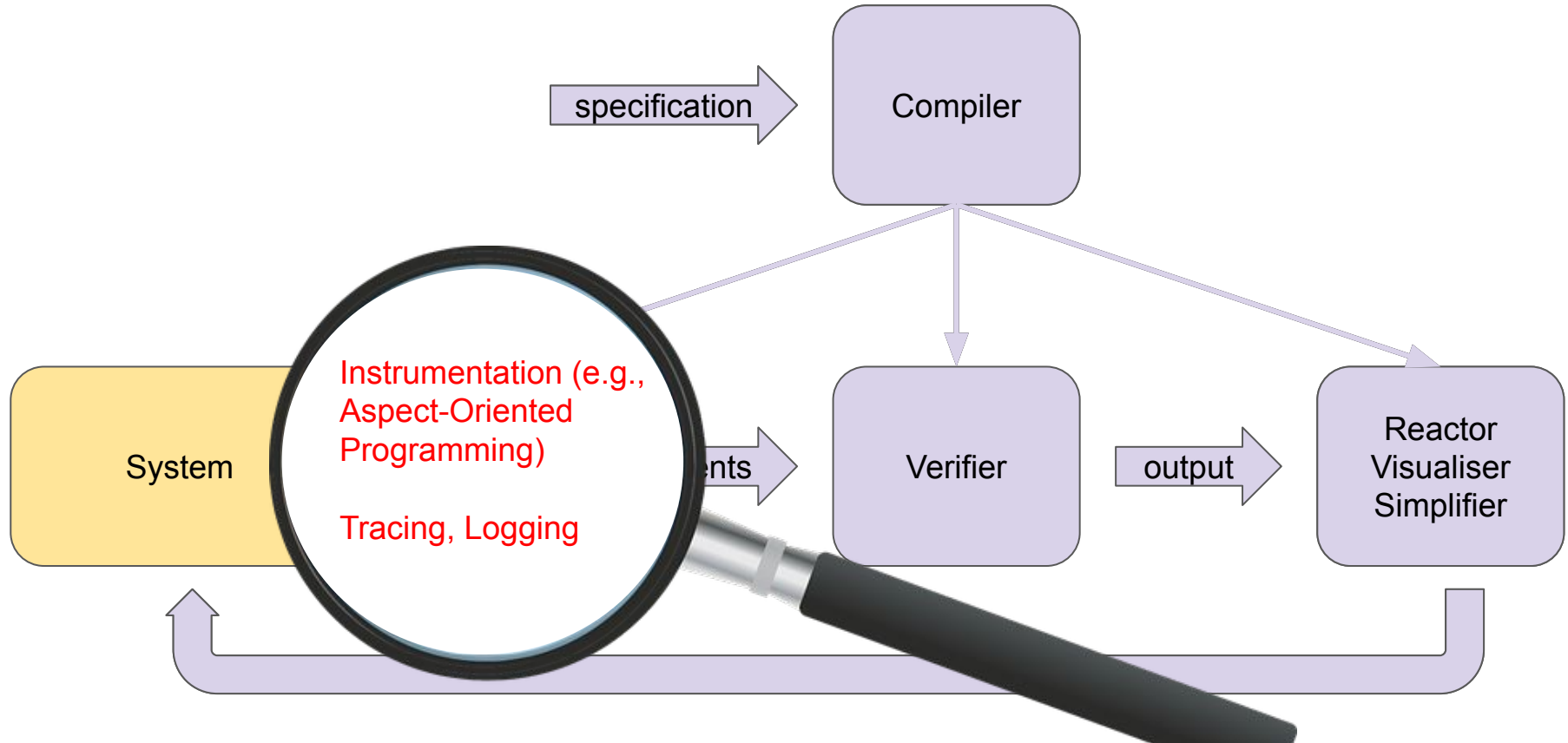
Monitors



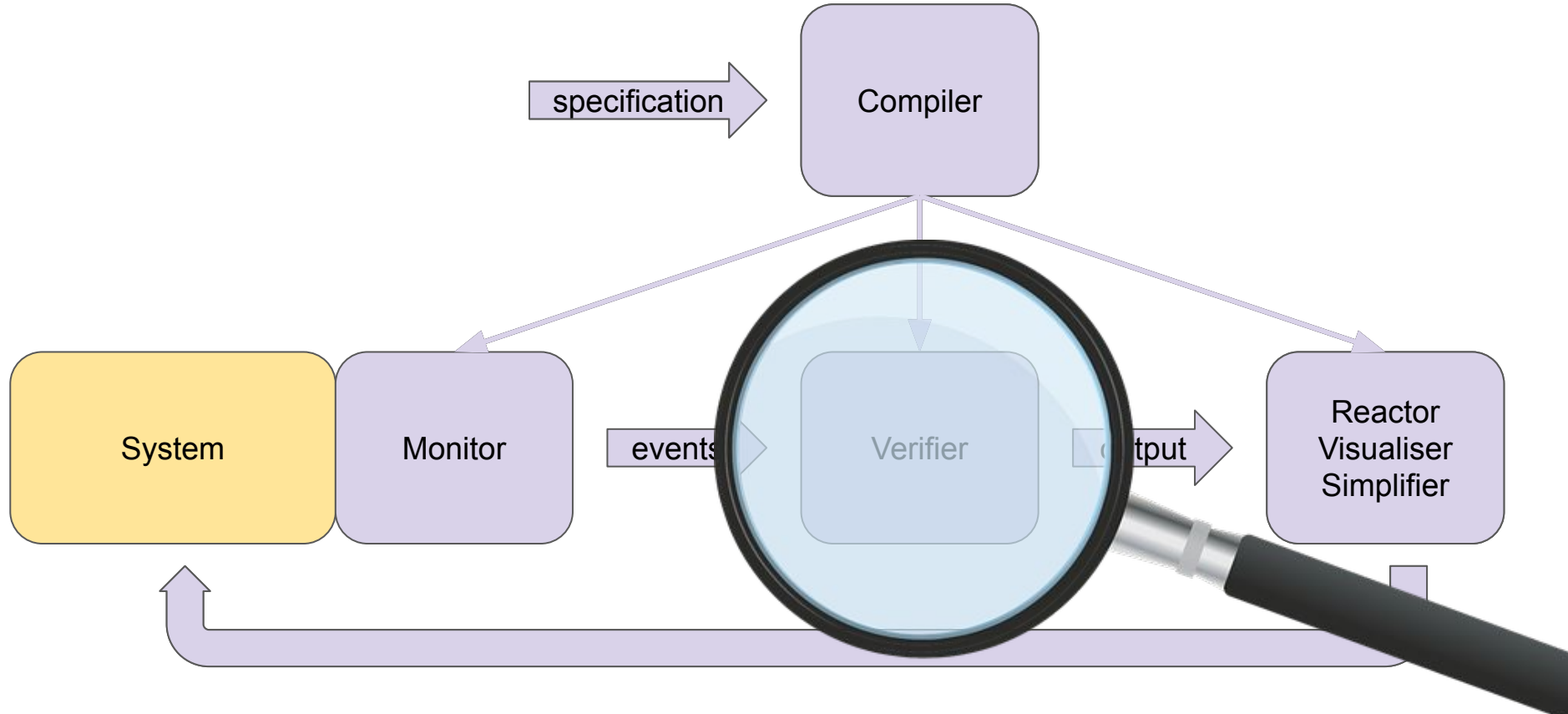
Monitors - Challenges



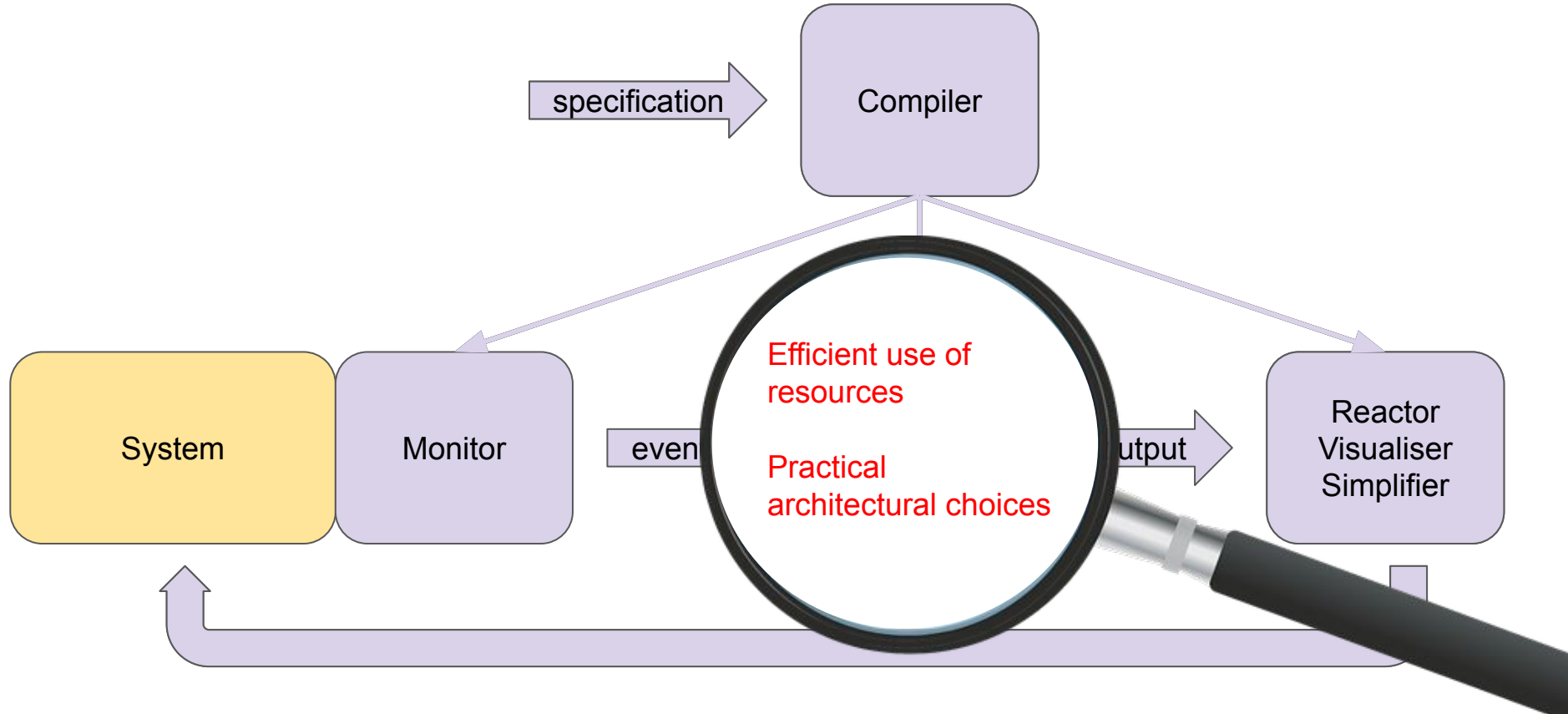
Monitors - Options



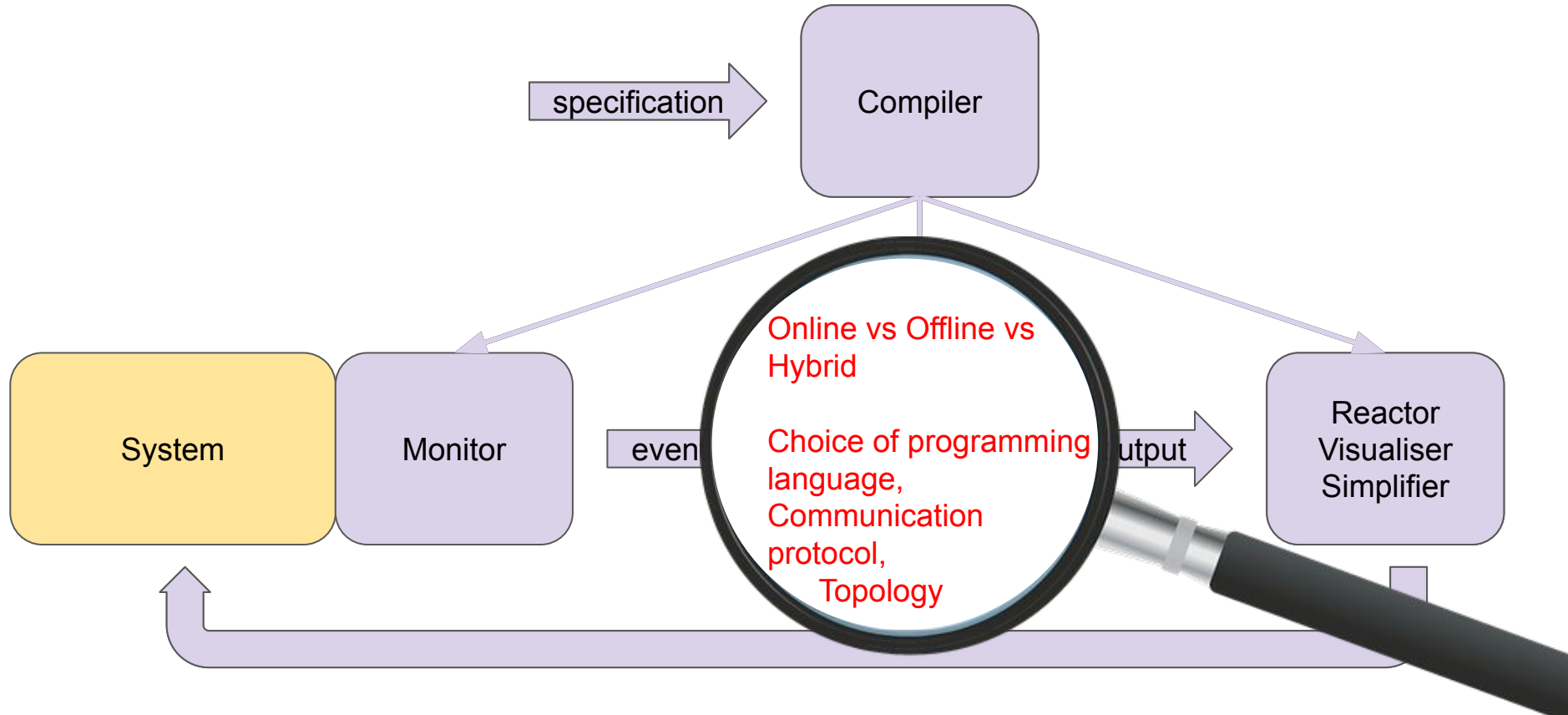
Verifiers



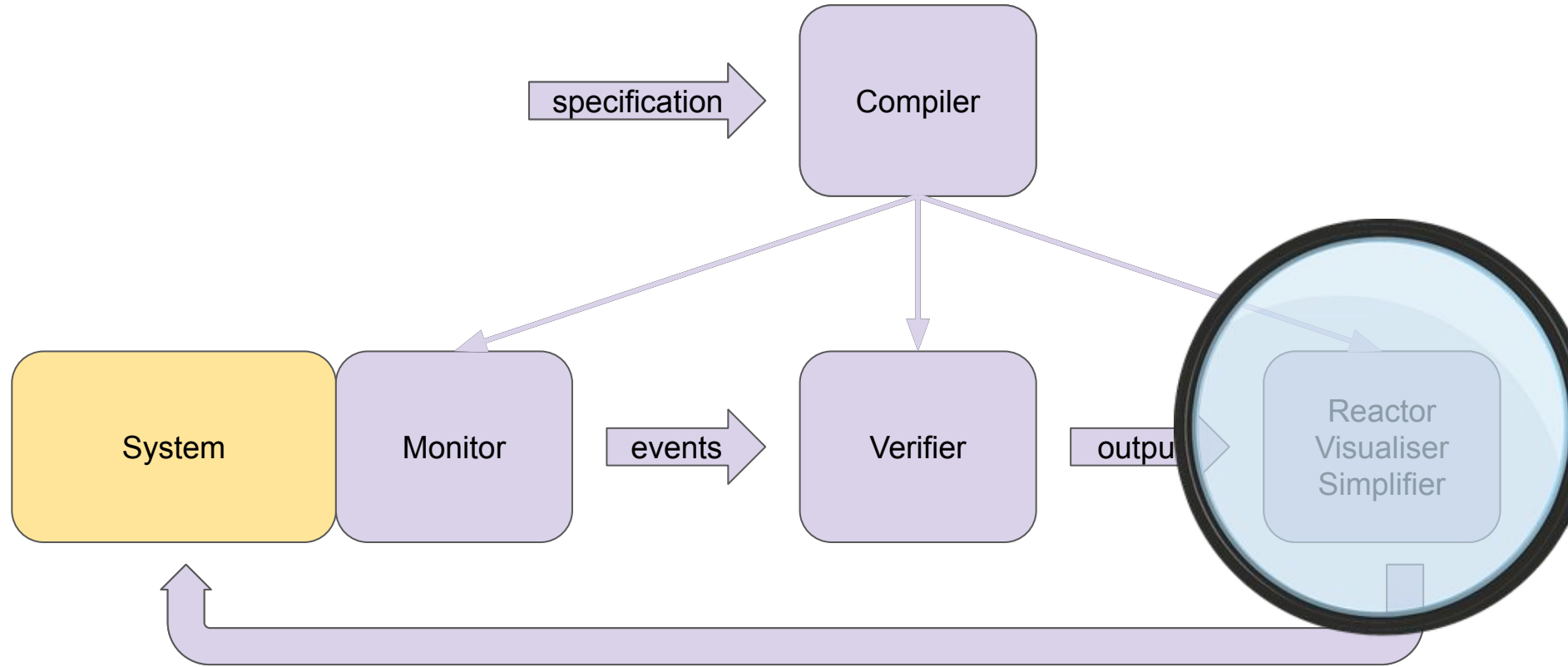
Verifiers - Challenges



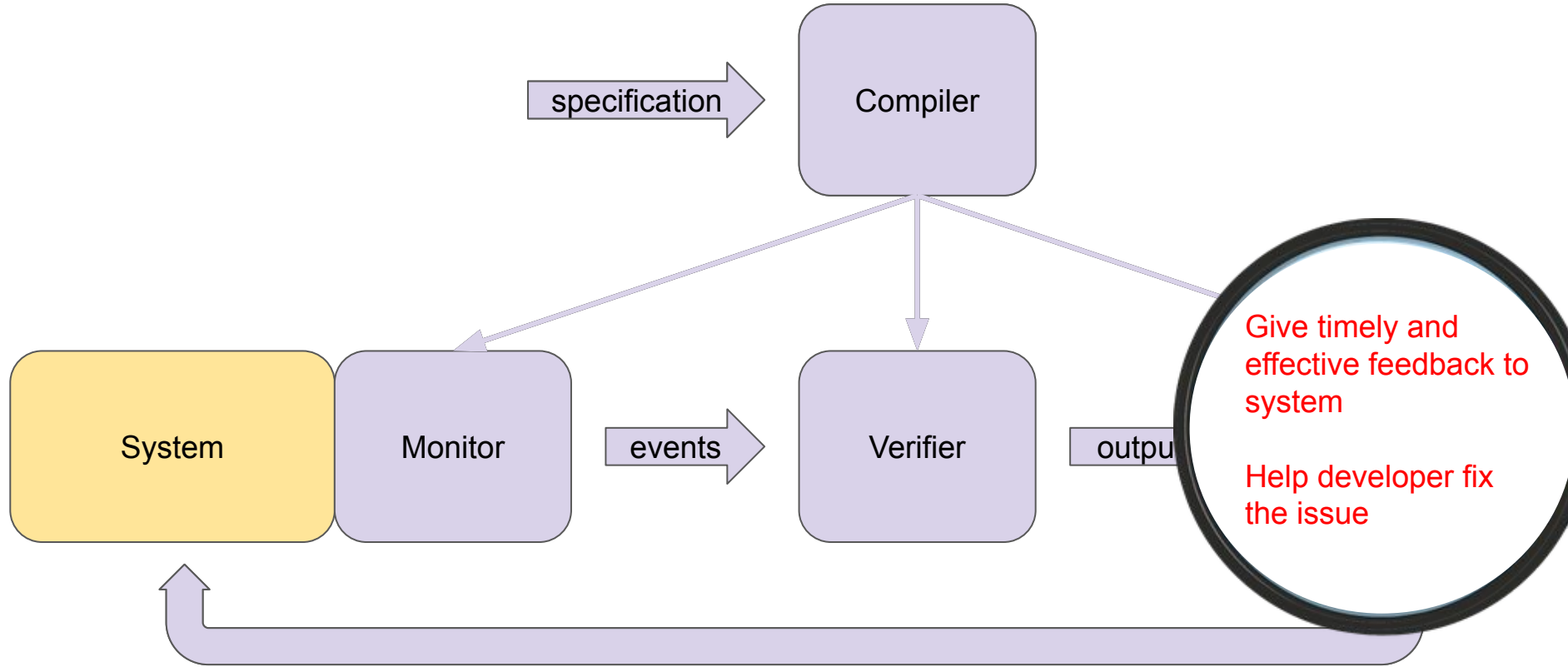
Verifiers - Options



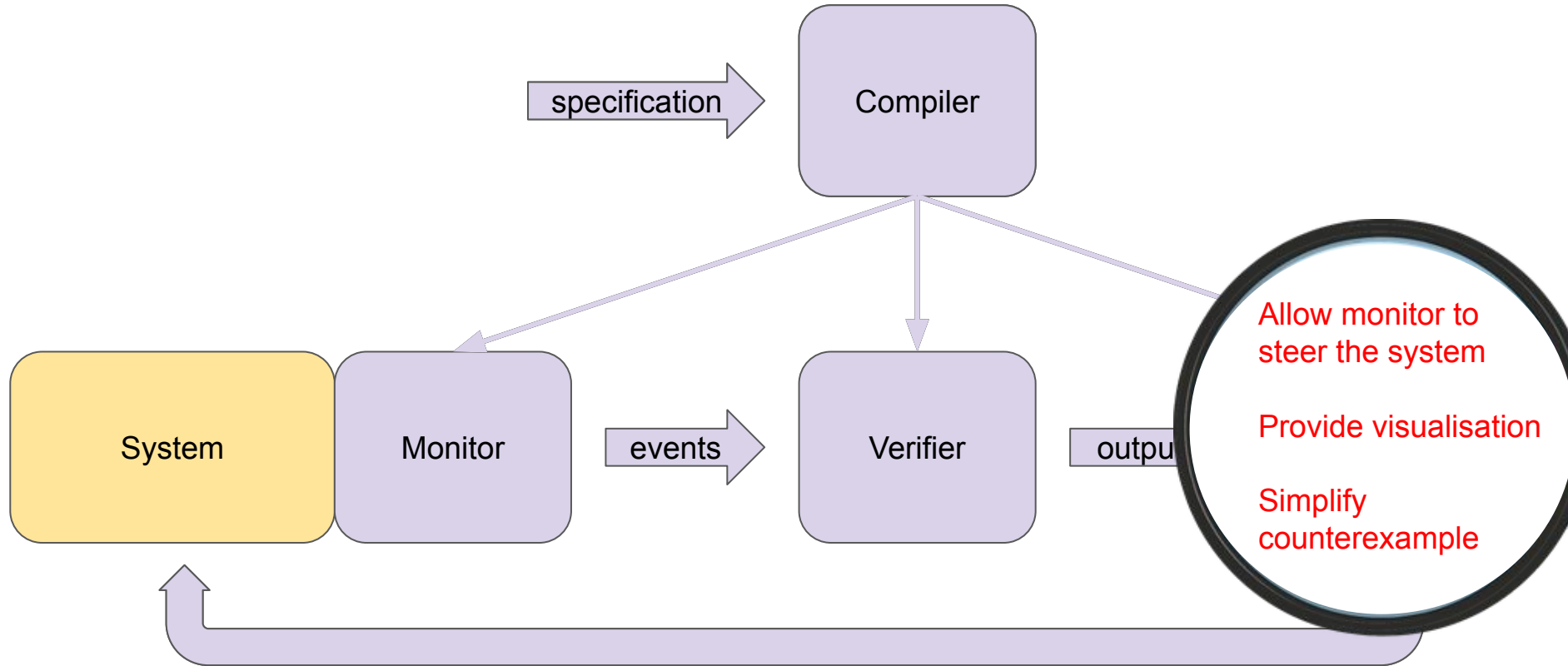
Reactions



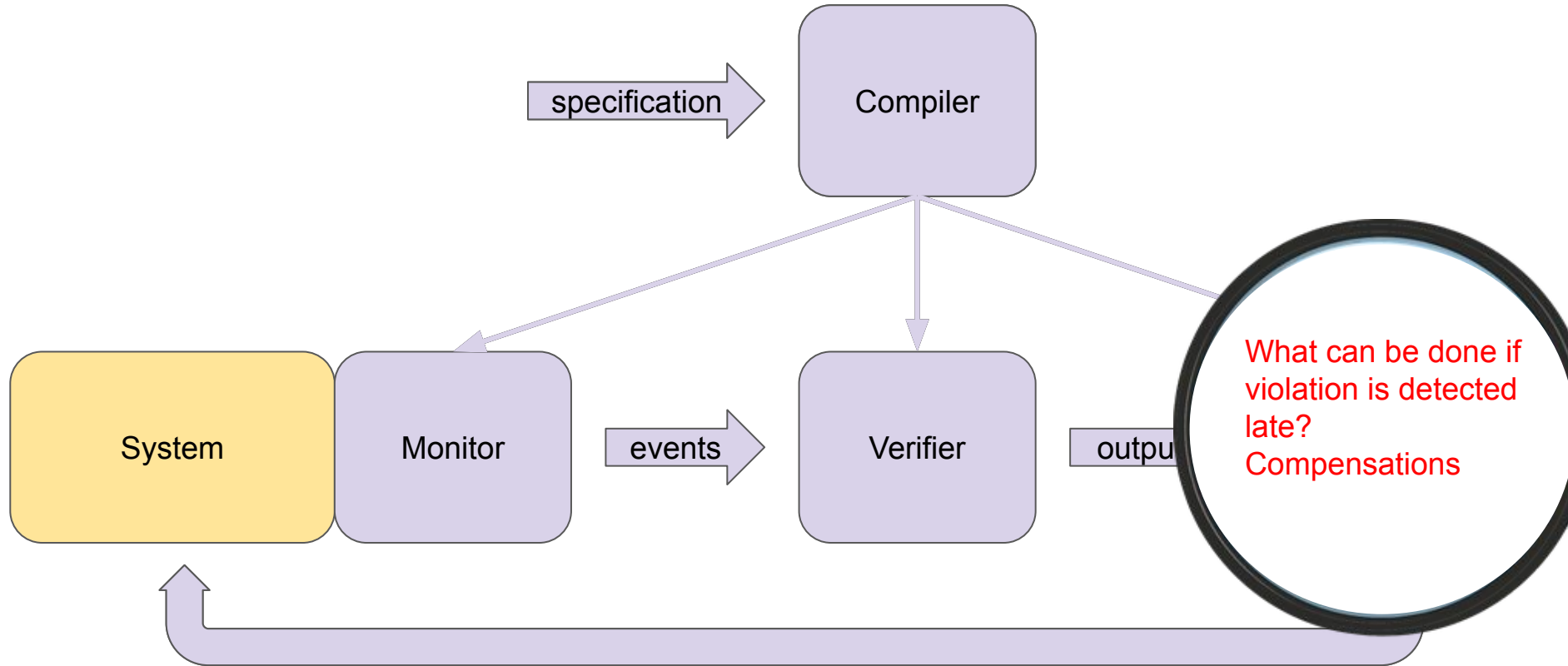
Reactions - Challenges



Reactions - Options



Reactions - More Options



Cyber Security (1)

Securing a Group Key Exchange Protocol Implementation

(part of NATO-funded project)

Secure Communication in the Quantum Era

Quantum computers (when they become practical) pose a threat to cryptographic communication protocols.

This project aimed to design a new “Quantum-safe” Group Key Exchange Protocol

And provide a proof of concept implementation

Case study: a chat application using the protocol to establish the secret keys



L-Università
ta' Malta

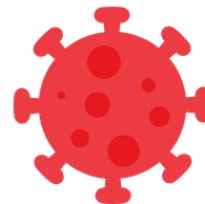
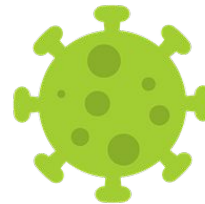


The need for secure communication

As the COVID-19 pandemic lockdown forced most employees into remote working, serious weaknesses in Zoom were exposed. Issues ranged from insecure key establishment to inadequate block cipher mode usage.

Other previous high-profile incidents concerning insecure cryptographic protocol implementation were caused by:

- Weak randomness.
- Insufficient checks on protocol compliance.
- Memory corruption bugs.



L-Università
ta' Malta



Many things can go wrong on many different levels of abstraction

(High level) Wrong protocol
implementation

The protocol implementation might deviate from the
verified (theoretical) design

Medium level threats

Malware, Data leaks, etc

Low level threats

Arithmetic overflows, undefined downcasts, and
invalid pointer references

Hardware

Can hardware be trusted?
Side Channel attacks?



L-Università
ta' Malta



Many things can go wrong on many different levels of abstraction

(High level) Wrong protocol implementation

The protocol implementation might deviate from the verified (the specification)

Medium level threats

Malware

How can RV help?

Low level threats

Arithmetic errors, buffer overflows, and downcasts, and invalid pointer references

Hardware

Can hardware be trusted?
Side Channel attacks?



L-Università
ta' Malta



Many things can go wrong on many different levels of abstraction

(High level) Wrong protocol implementation

The protocol implementation might deviate from the verified (the specification)

Medium level threats

Malware

How can RV help?

Low level threats

Arithmetic overflows, buffer overflows, and invalid memory accesses

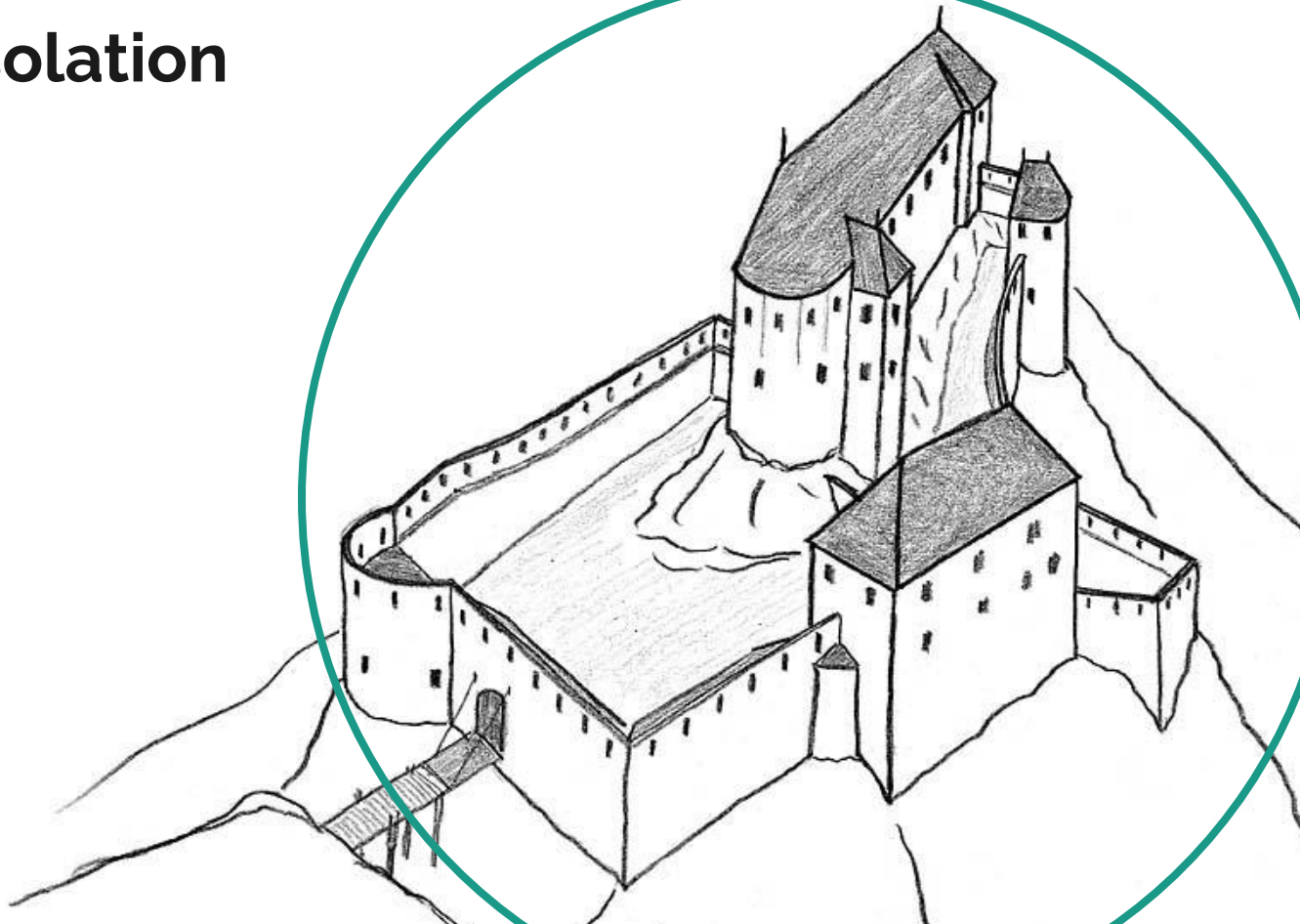
Hardware

Side Channel

A mix of RV and Hardware



Concept 1: Isolation



Concept 1: Isolation

Medium level

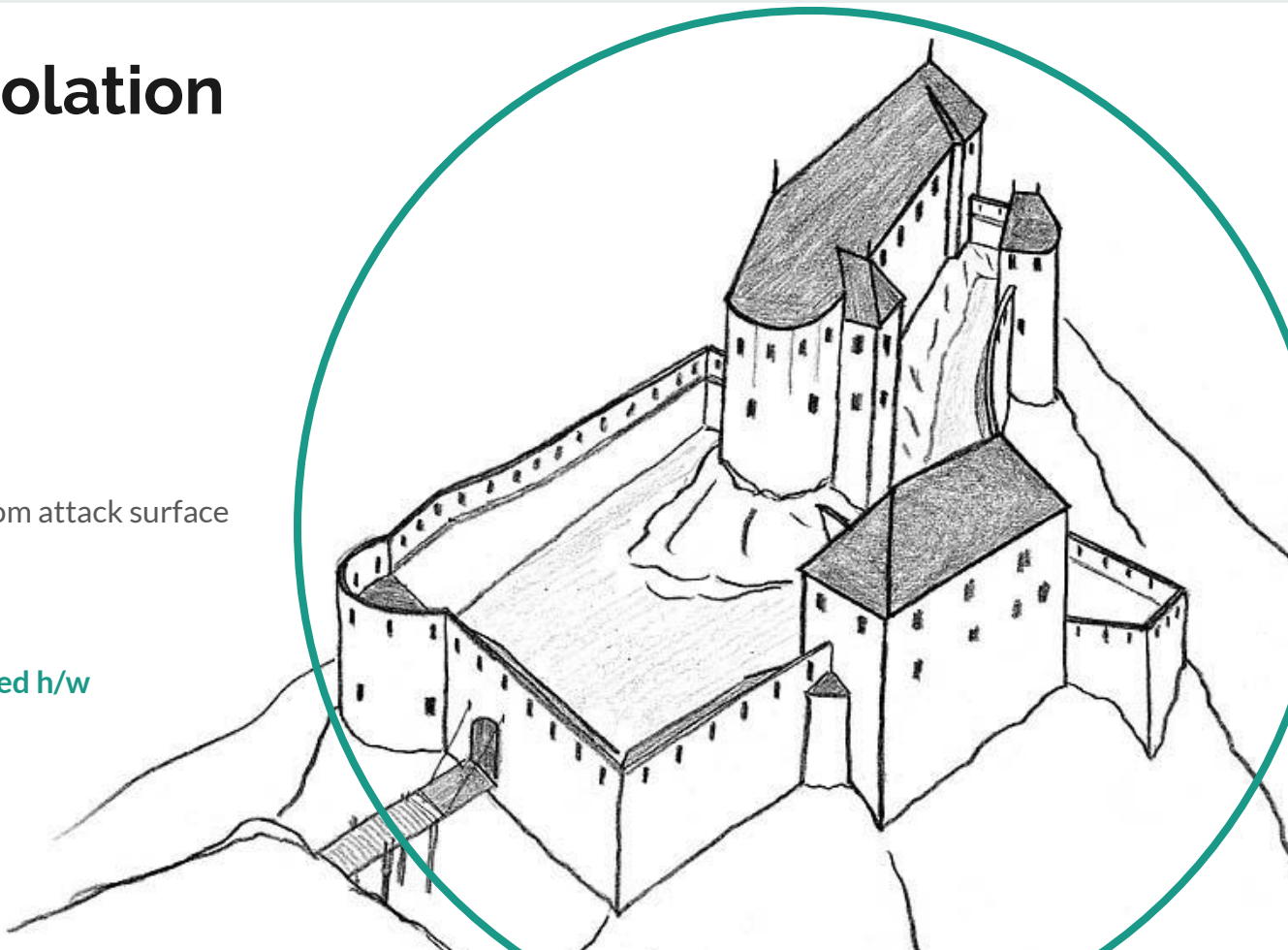
Crypto keys

Low level

Hide memory errors from attack surface

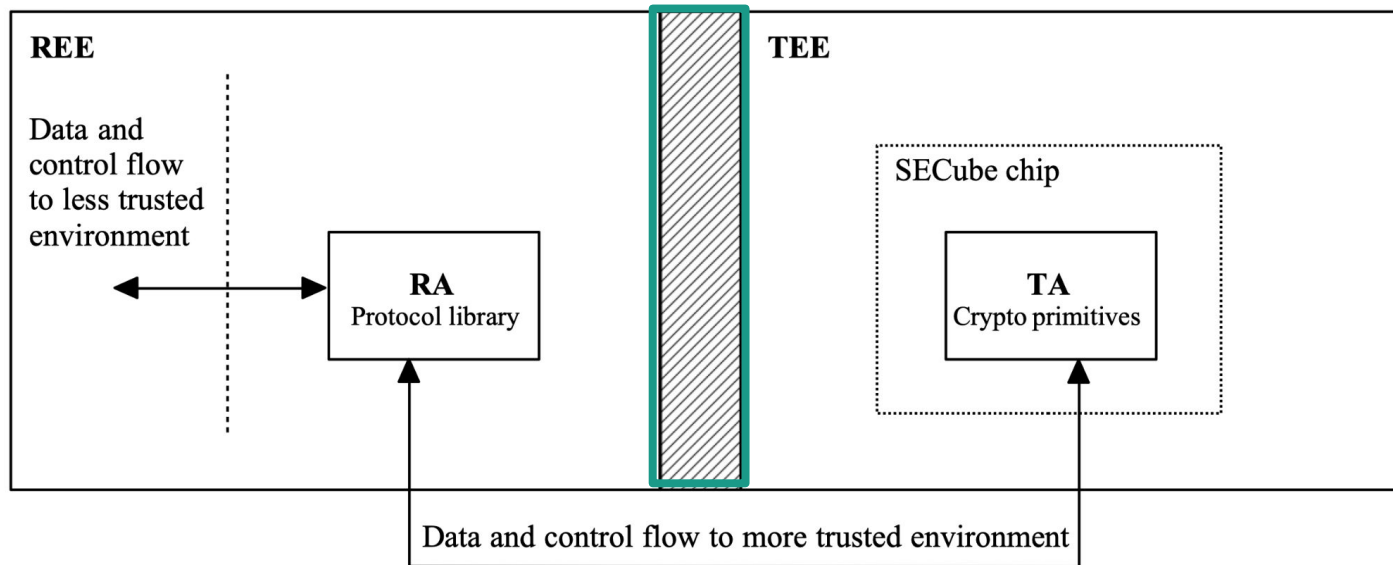
Hardware

Bring-your-own certified h/w

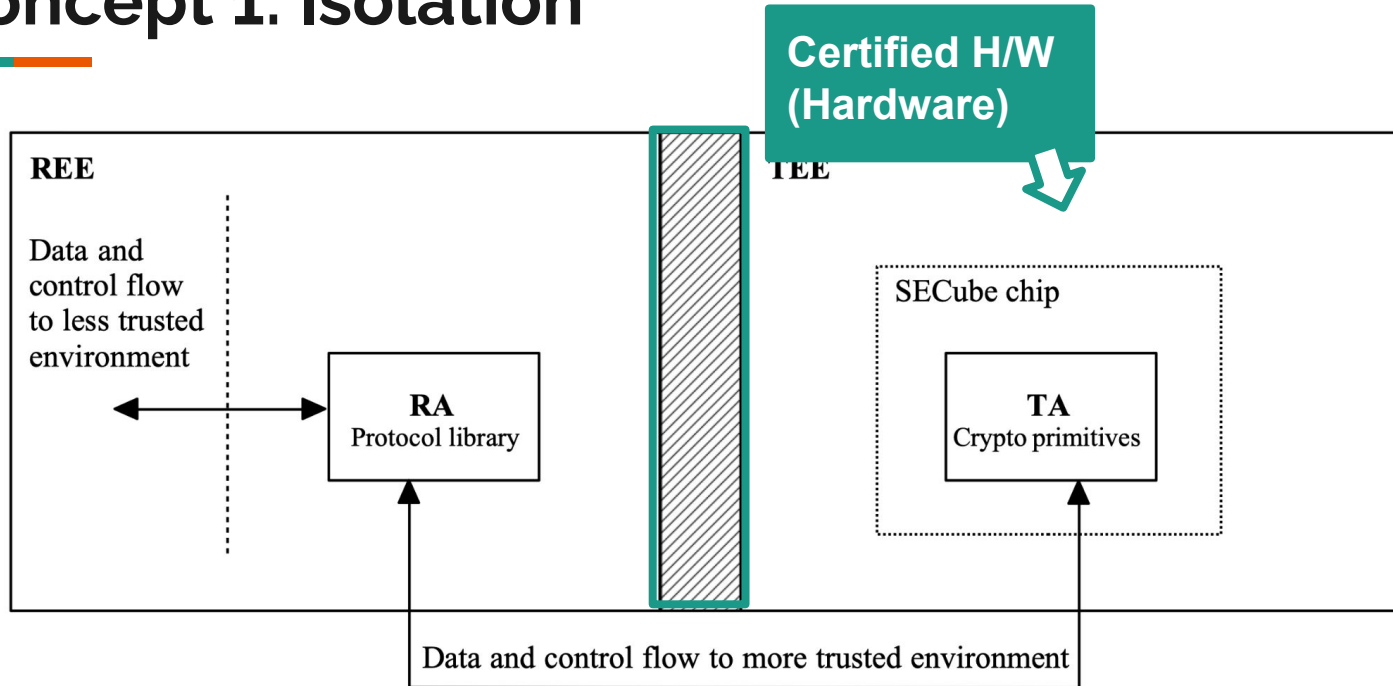


L-Università
ta' Malta

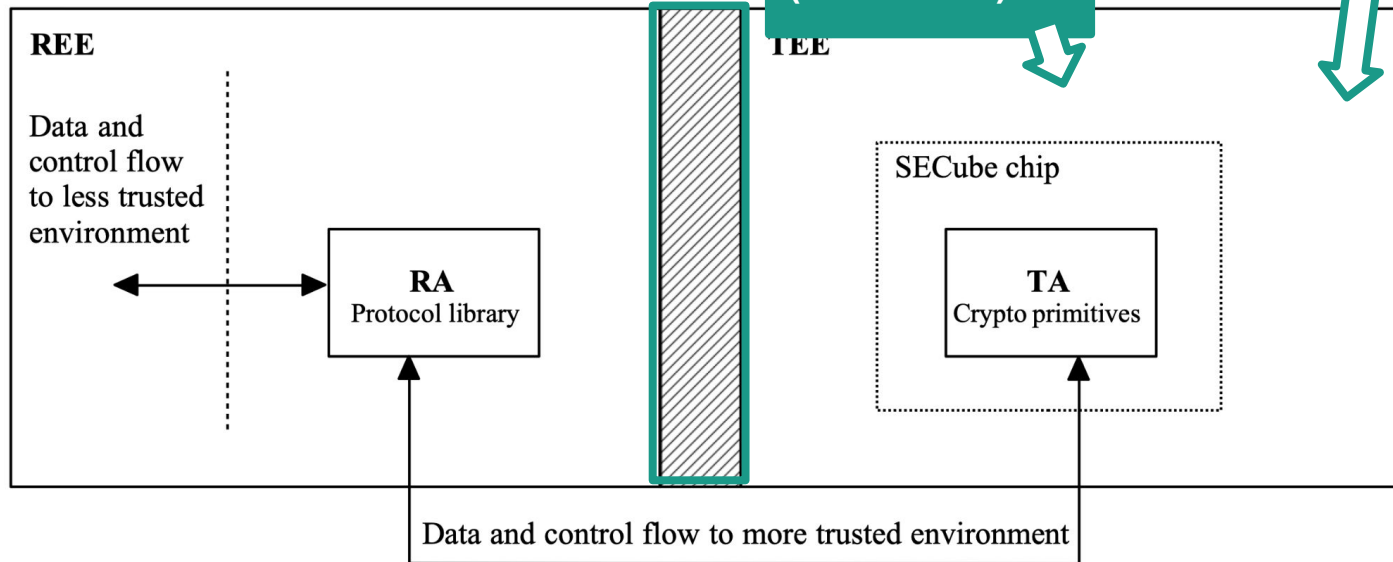
Concept 1: Isolation



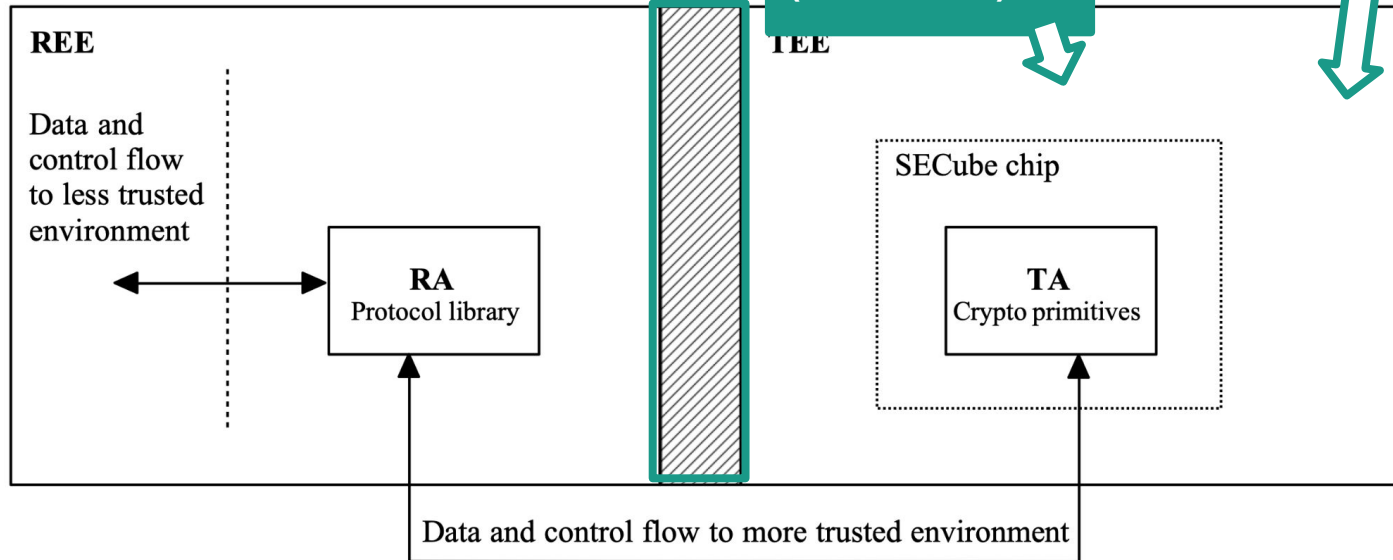
Concept 1: Isolation



Concept 1: Isolation



Concept 1: Isolation

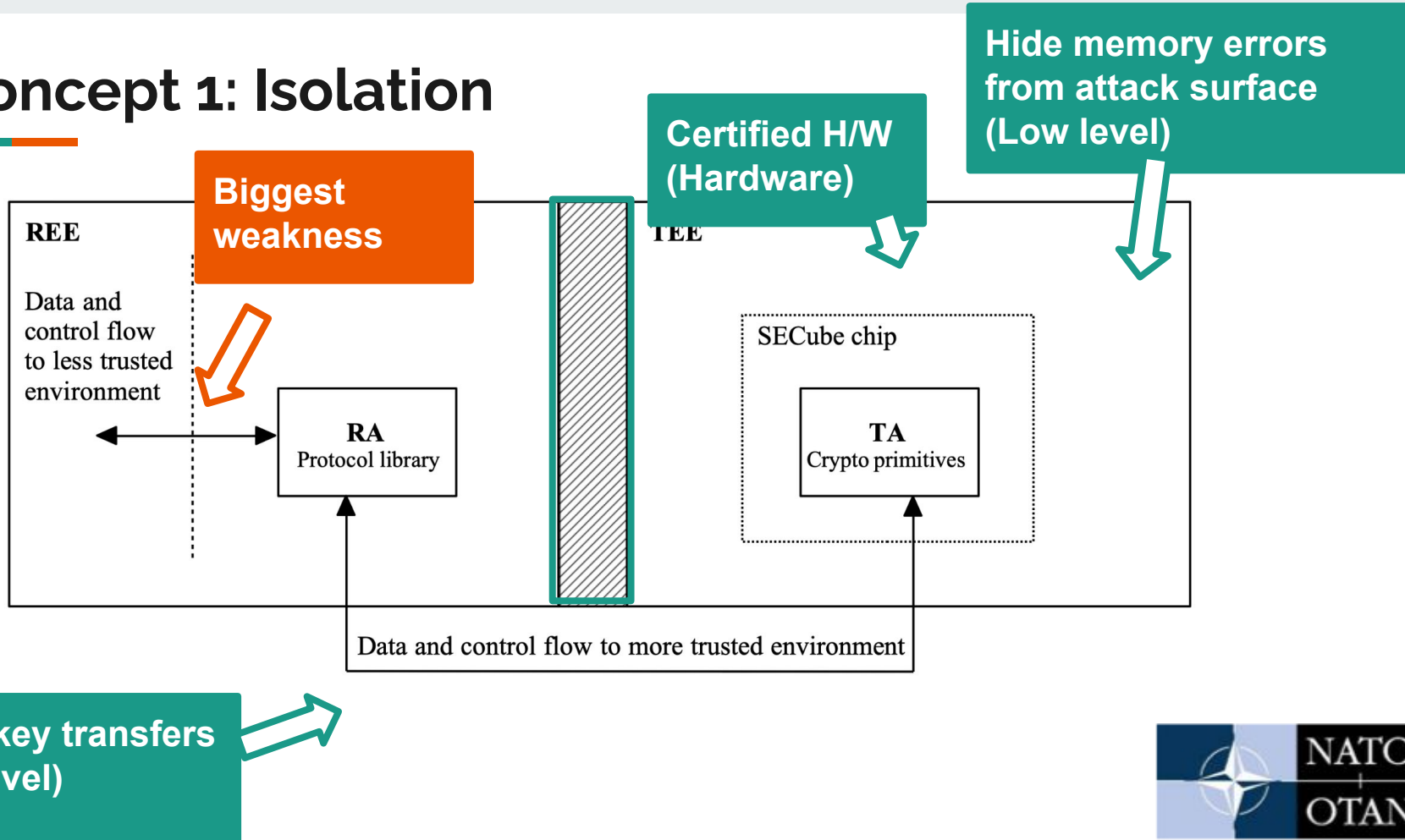


**Certified H/W
(Hardware)**

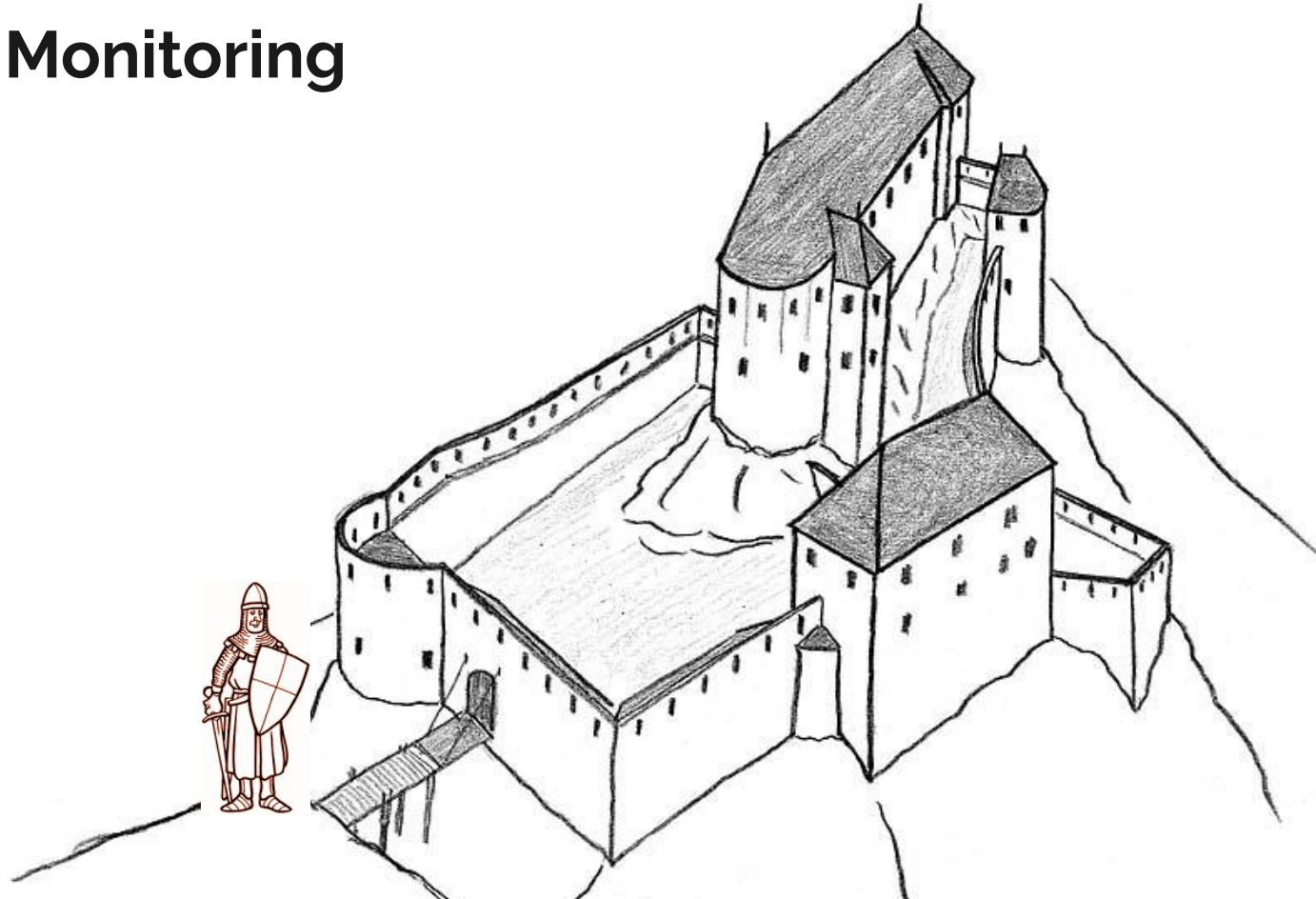
**Hide memory errors
from attack surface
(Low level)**

**No crypto key transfers
(Medium level)**

Concept 1: Isolation



Concept 2: Monitoring



Concept 2: Monitoring

High level

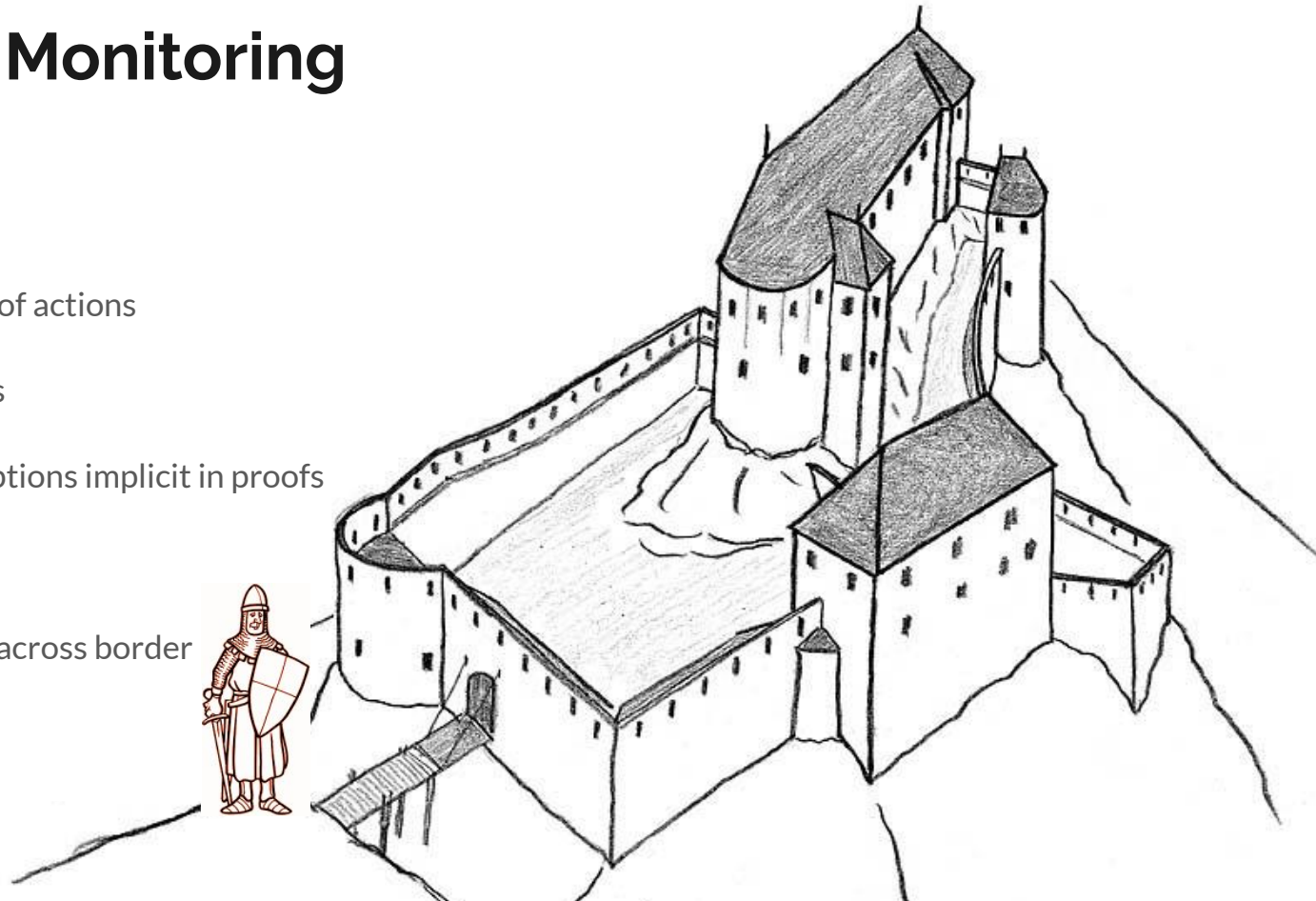
- Check sequences of actions

- Check parameters

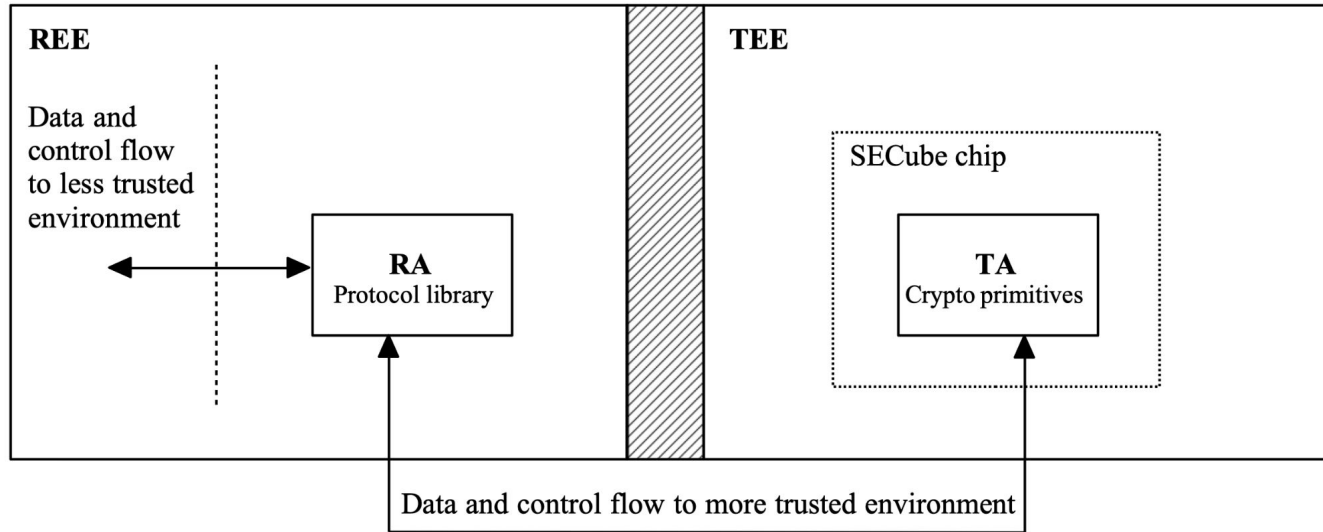
- Check any assumptions implicit in proofs

Low level

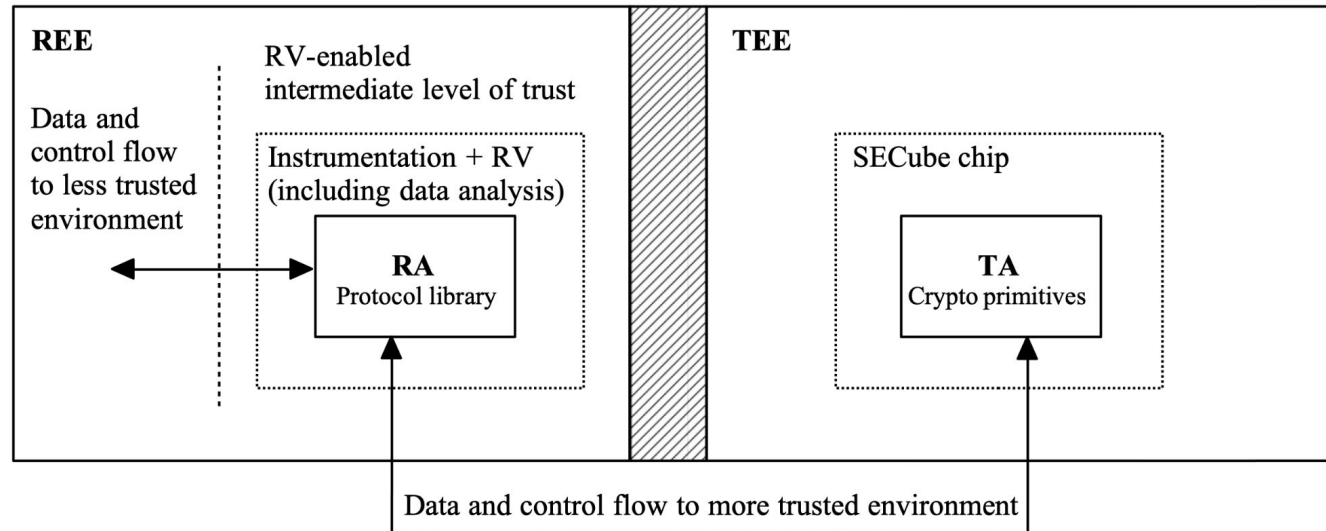
- Plaintext leakage across border



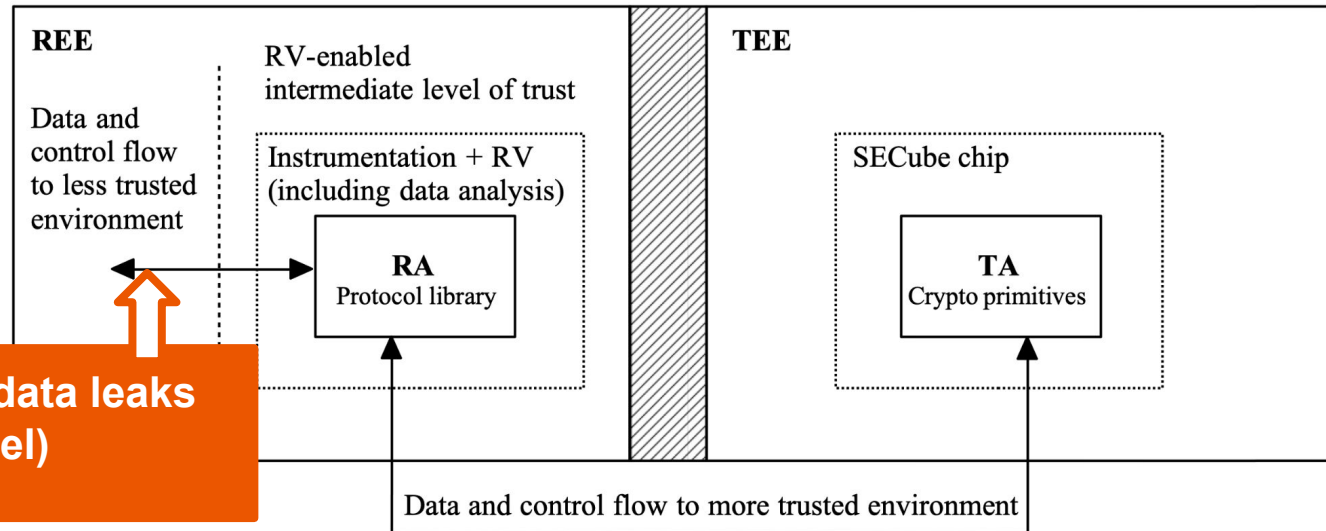
Concept 2: Monitoring



Concept 2: Monitoring



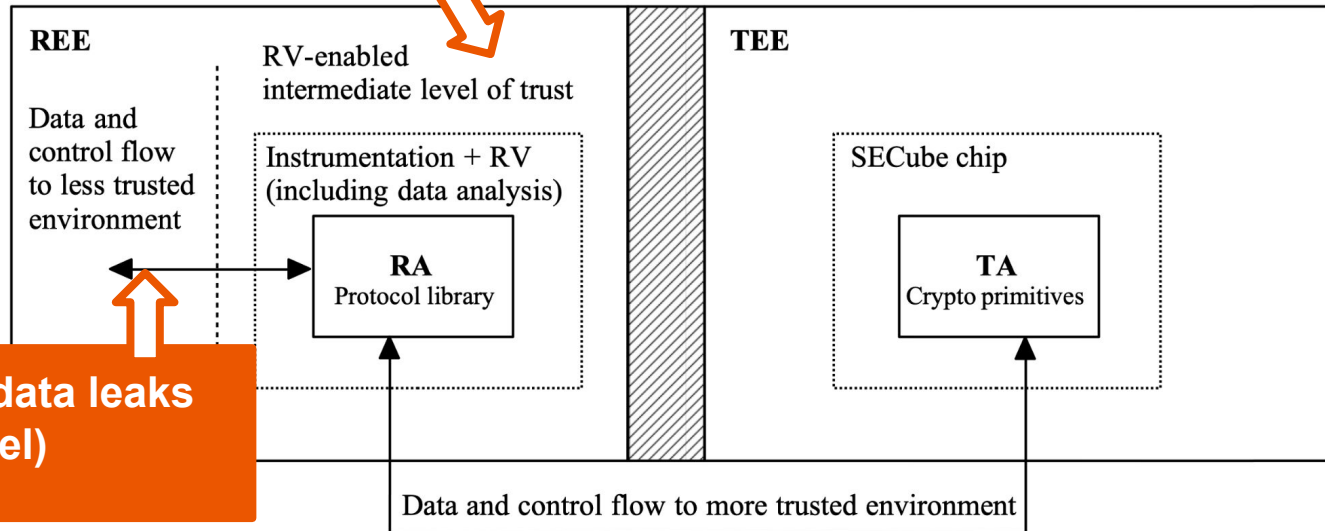
Concept 2: Monitoring



Monitor for data leaks
(Medium level)

**Monitor code while executing
(High level)**

Monitoring



**Monitor for data leaks
(Medium level)**



L-Università
ta' Malta

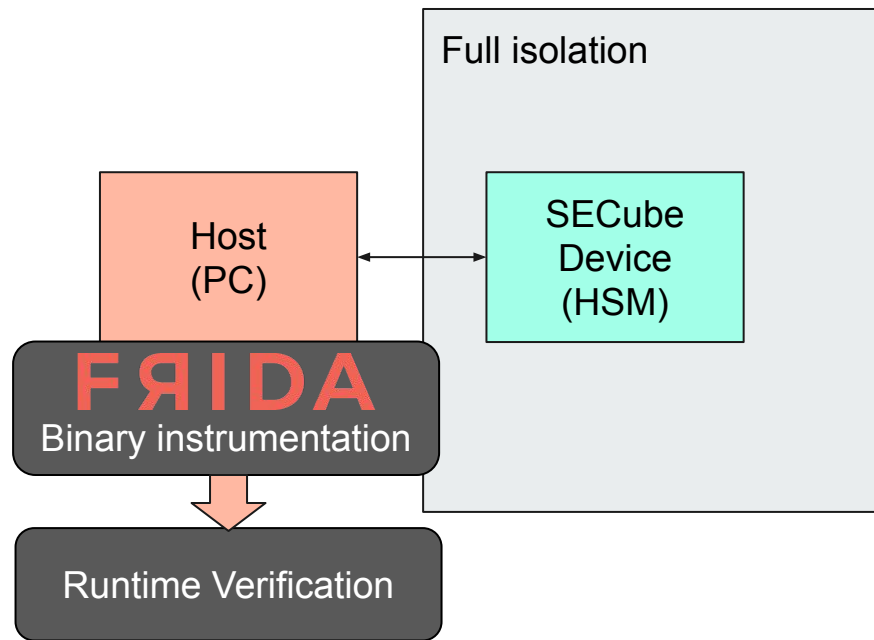


How does it look in practice?

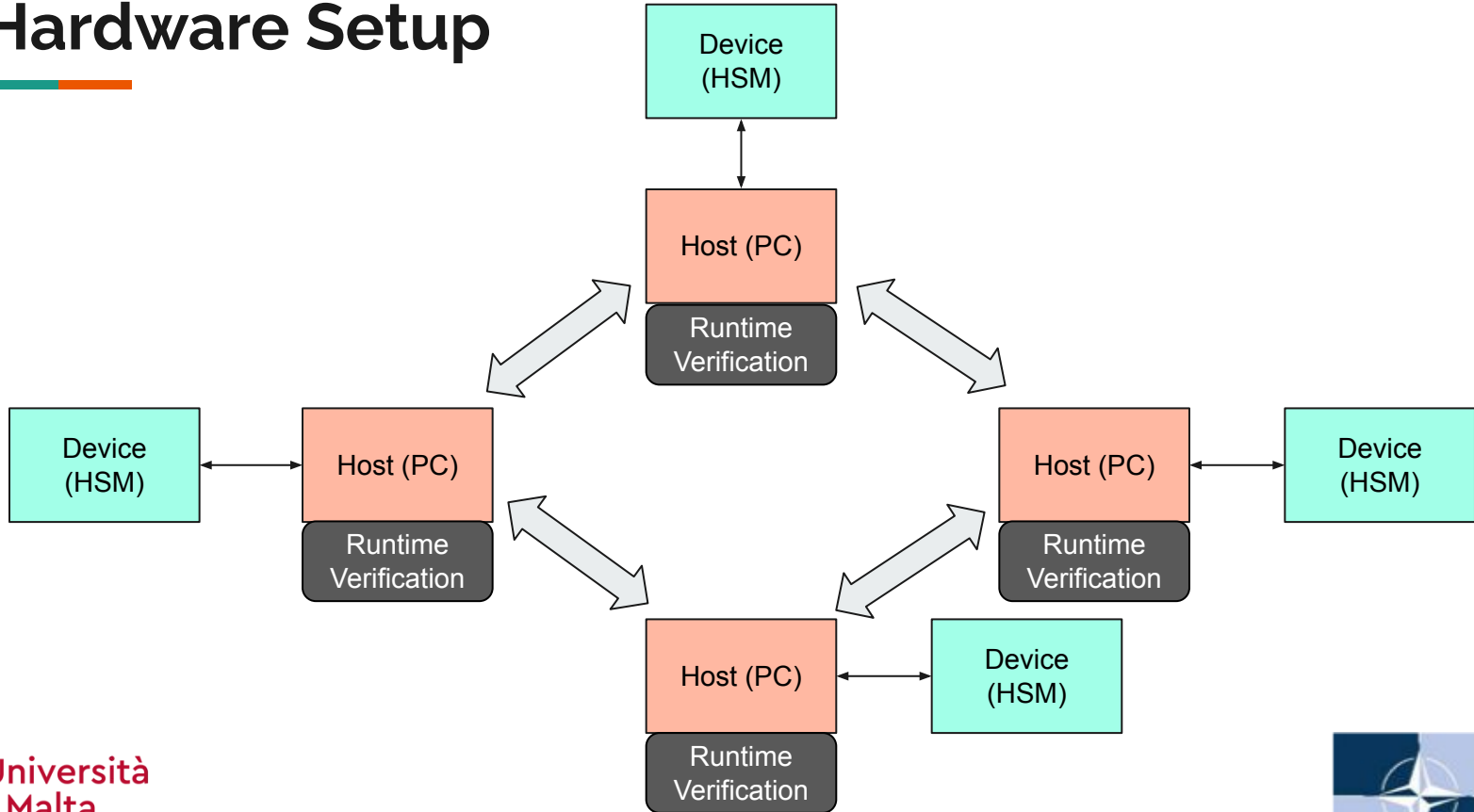
Hardware Security Module - SECube



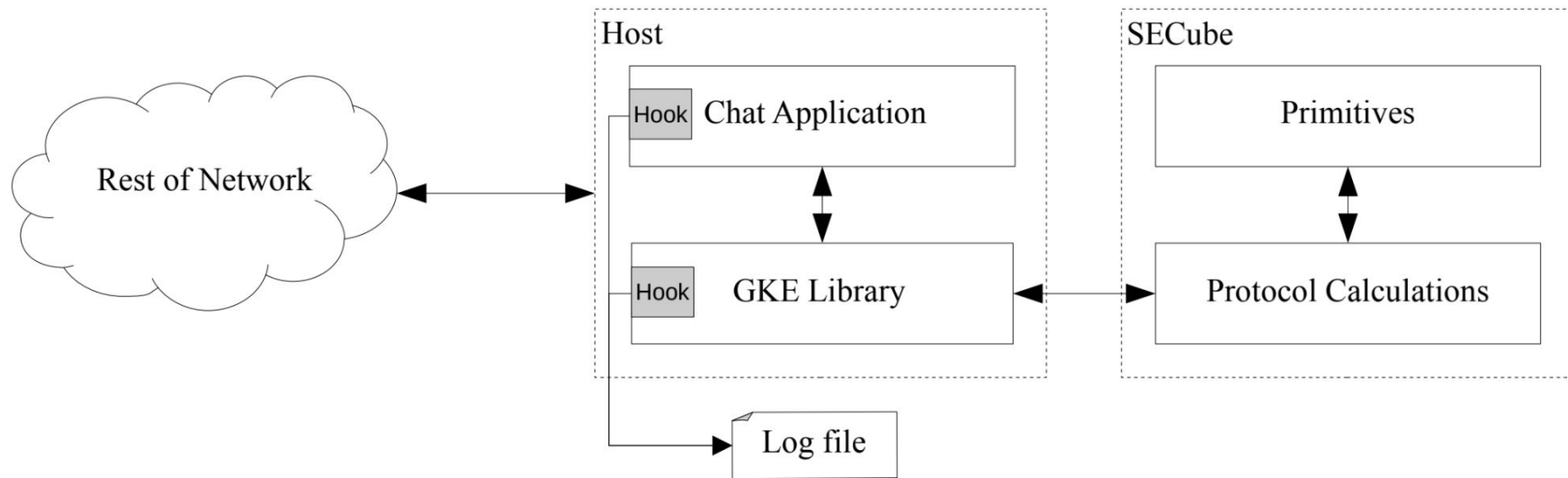
Hardware



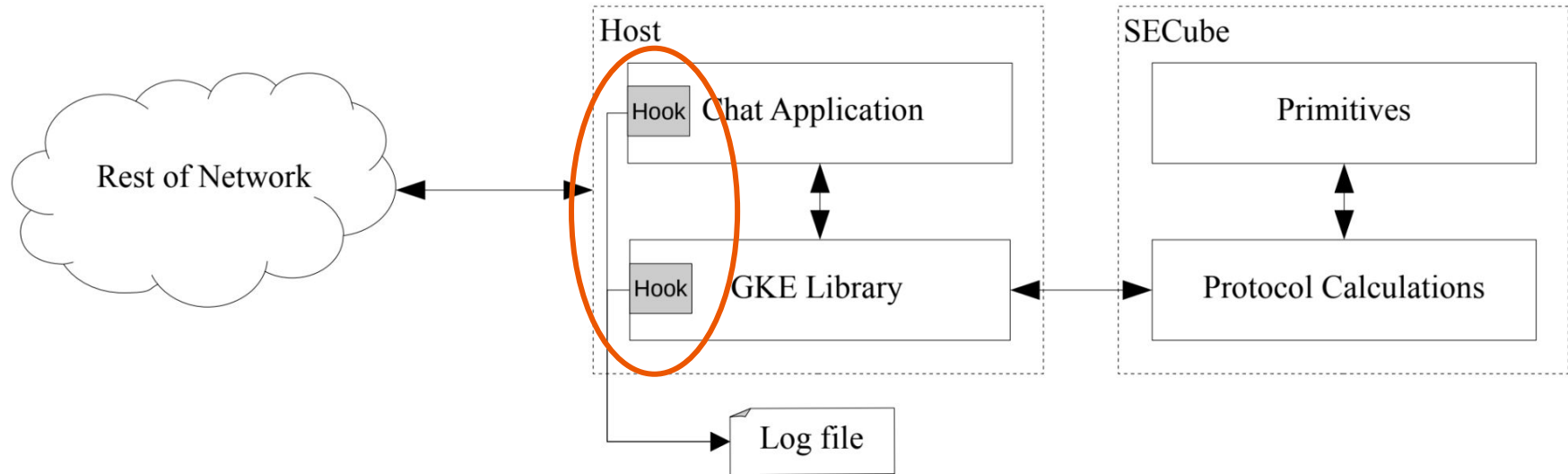
Hardware Setup



Looking Deeper at a Single Client



Instrumentation

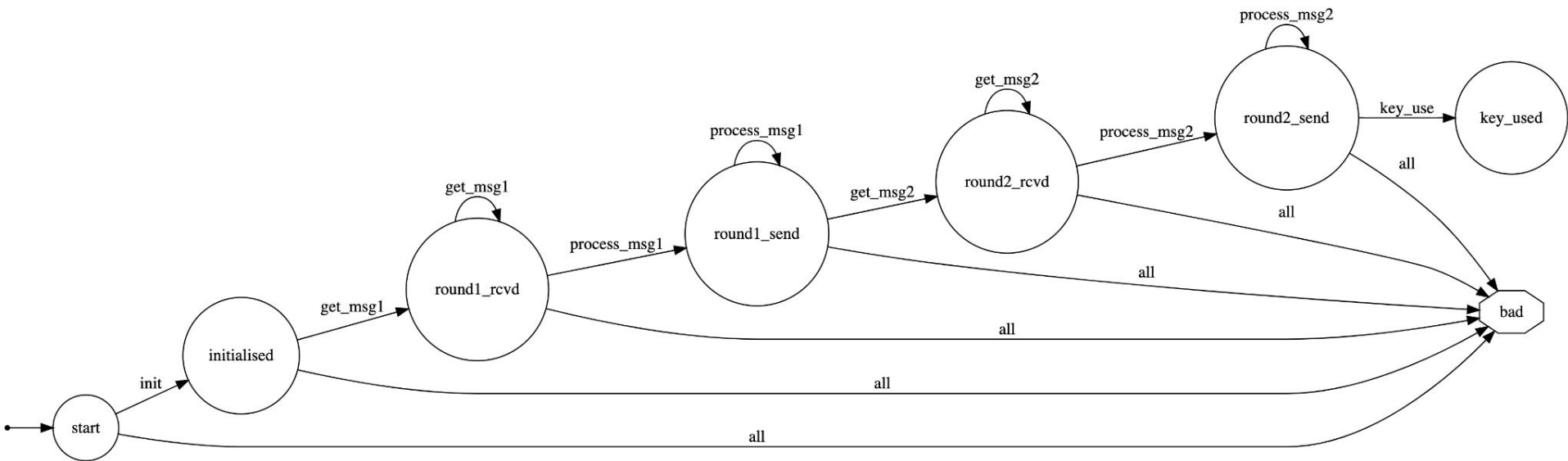


Log Output

```
[1935] 5642f6415b70 47 4b 45 7c 3e 20 GKE|>
[1935] 1625088325476 libpthread!read(fd=0x0, buf=0x7ffc1a22736f, count=0x1)
[1935] 1625088325476 libpthread!read() retVal: 0x1
[1935] 1625088325476 [HEXDUMP] out...
[1935] 7ffc1a22736f 2f /
[1935] 1625088325476 libpthread!read(fd=0x0, buf=0x7ffc1a2272df, count=0x1)
[1935] 1625088325476 libpthread!read() retVal: 0x1
[1935] 1625088325476 [HEXDUMP] out...
[1935] 7ffc1a2272df 72 r
[1935] 1625088325476 libpthread!read(fd=0x0, buf=0x7ffc1a2272df, count=0x1)
[1935] 1625088325476 libpthread!read() retVal: 0x1
[1935] 1625088325476 [HEXDUMP] out...
[1935] 7ffc1a2272df 6f o
[1935] 1625088325476 libpthread!read(fd=0x0, buf=0x7ffc1a2272df, count=0x1)
[1935] 1625088325476 libpthread!read() retVal: 0x1
[1935] 1625088325476 [HEXDUMP] out...
[1935] 7ffc1a2272df 6f o
[1935] 1625088325476 libpthread!read(fd=0x0, buf=0x7ffc1a2272df, count=0x1)
[1935] 1625088325476 libpthread!read() retVal: 0x1
[1935] 1625088325476 [HEXDUMP] out...
[1935] 7ffc1a2272df 6d m
[1935] 1625088325476 libpthread!read(fd=0x0, buf=0x7ffc1a2272df, count=0x1)
[1935] 1625088325476 libpthread!read() retVal: 0x1
[1935] 1625088325476 [HEXDUMP] out...
[1935] 7ffc1a2272df 20
[1935] 1625088325476 libpthread!read(fd=0x0, buf=0x7ffc1a2272df, count=0x1)
[1935] 1625088325476 libpthread!read() retVal: 0x1
```

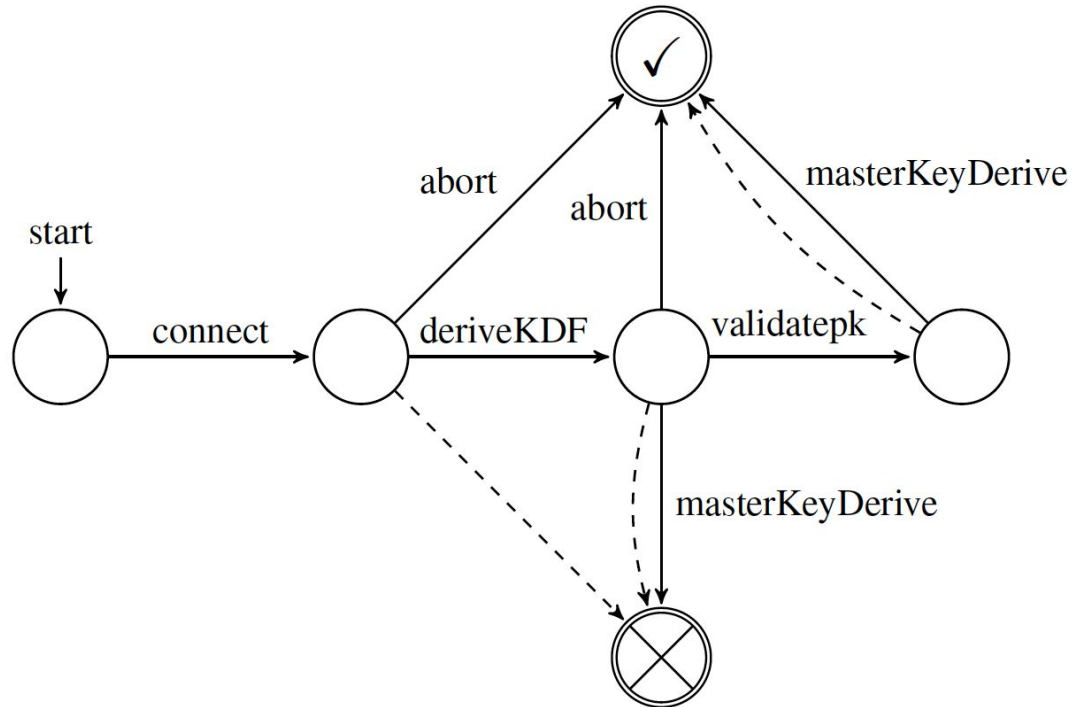


Formal Specification



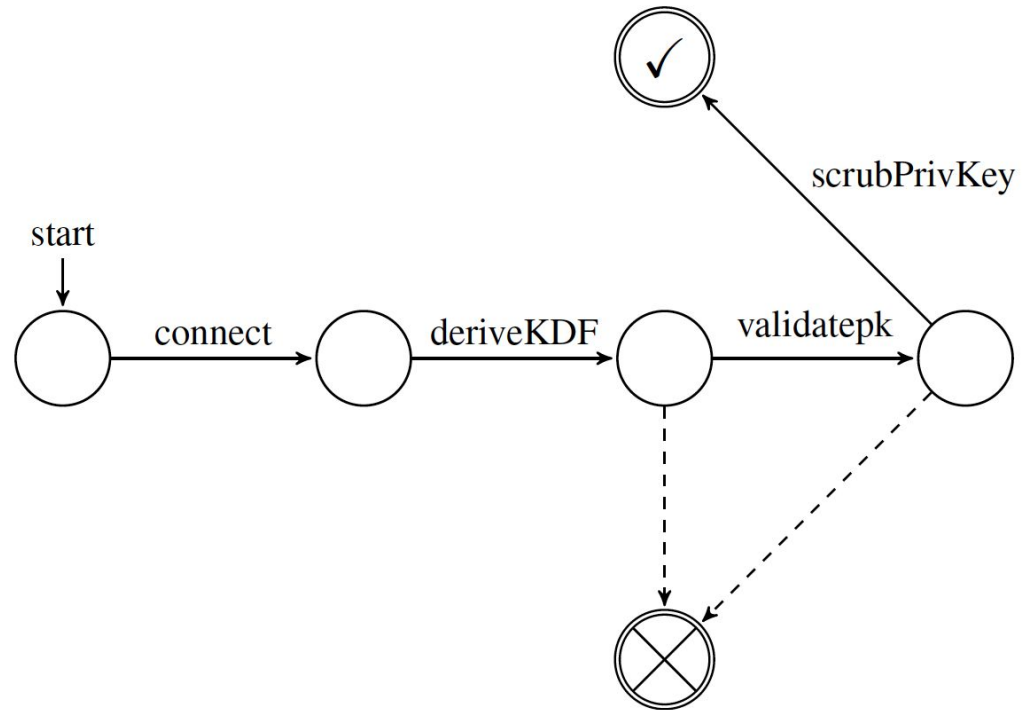
Properties verified (High level) on ECDHE

Remote peer's **public key is validated** on each exchange (unless the session is aborted)



Properties verified (High level) on ECDHE

Once master secret is established,
private keys should be **scrubbed**
from memory

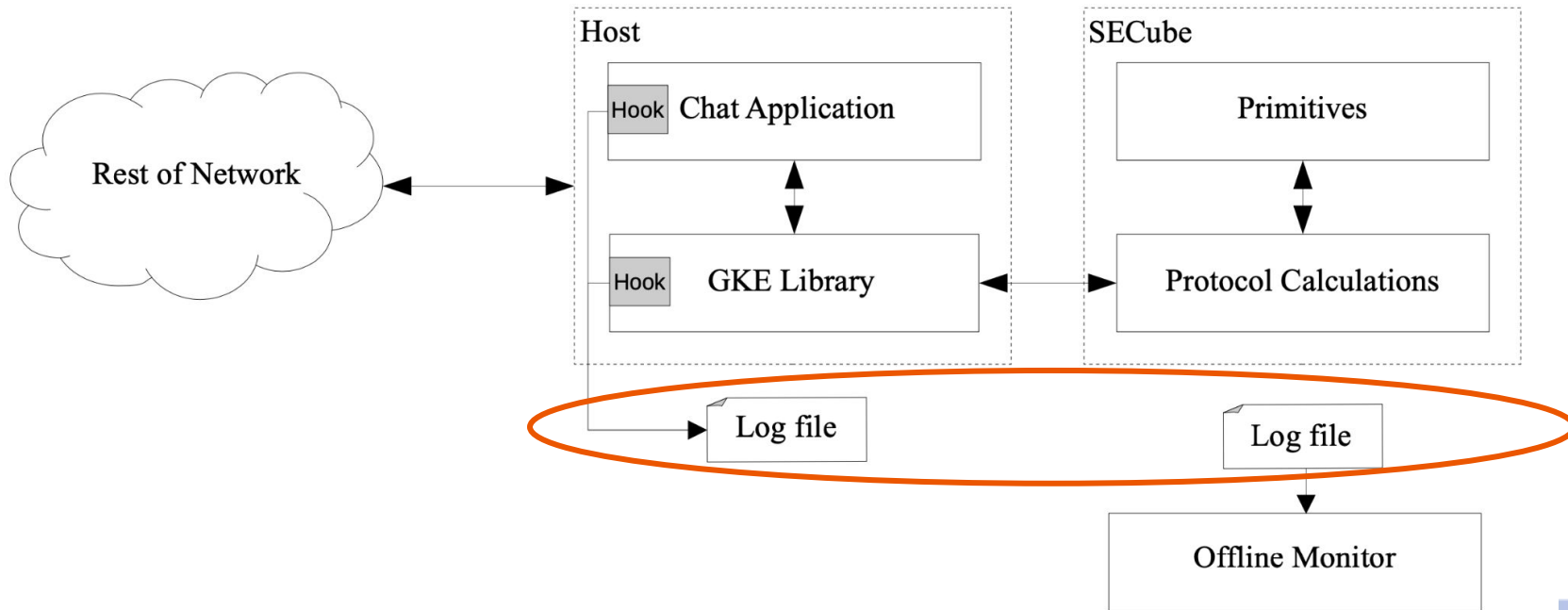


High level monitors

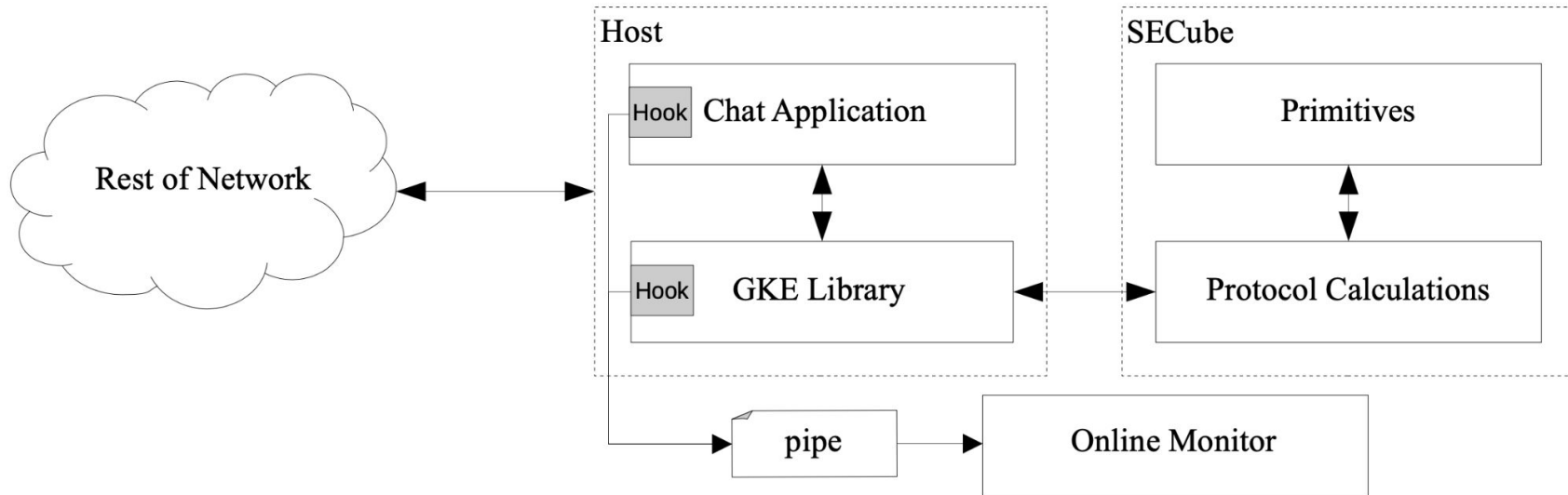


| Property layers | Chat app | GKE Library | All (incl. Primitives) |
|-----------------|--------------------------------------|-------------------------|---------------------------------------|
| Assertion | Printable decrypted characters | Sensitive data scrubbed | Valid function parameters and returns |
| Temporal | Chatroom lifecycle, standard sockets | | Correct function call sequence |
| Hyper | | Randomness quality | |

Running the Monitor Offline



Switching to Online Monitoring



```

natosps@natosps-Z87-HD3:~/git/GKEdemo/GKE/Instrumentation/Injection$ python3 injectRV.py ../../bin/chat --id 3 --repeater 147.175.106.130
FIFO named '/tmp/pipe 241798 to larva' is created successfully.
*****
* You are now connected to the server. *
* You can now create a chat room or enter an existing chat room. *
* For available commands, please type /help. *
*****
GKE|> waiting for chat app
/room new U 4
Creating room 'U'
-----
CONGRATULATIONS! The room 'U' was created for users {3,4}.
A shared secret key for all room users was established by the Quantum-Future Group Authenticated Key Exchange Protocol.
From now on, the shared secret key will be used with AES CCM 128 to encrypt communication between room users.
You can now enter the room and start sending encrypted messages.
-----
GKE|> RV:: *1* Initialised
RV:: *a* init_protocol_run_env called as expected
RV:: *b0* init_participant called as expected
RV:: *b1* init_participant called as expected
RV:: *c* round_one called as expected
RV:: *d* load_pw called as expected
RV:: *e* generate_beta called as expected
RV:: *f* calculate_g called as expected
RV:: *2* During the key exchange protocol (executed during room creation) the correct number of messages were received in round 1.
RV:: *g* round_two called as expected
RV:: *h* generate_k called as expected
RV:: *i* extract_result called as expected
RV:: *j1* kem_enc called as expected
RV:: *k1* calculate_shared_value called as expected
RV:: *l1* generate_MAC_init called as expected
RV:: *3* During the key exchange protocol (executed during room creation) the correct number of messages were sent in round 1.
RV:: *4* During the key exchange protocol (executed during room creation) the correct number of messages were received in round 2.
RV:: *5* During the key exchange protocol (executed during room creation) the correct number of messages were sent in round 2.
RV:: *m* round_two_finalize called as expected
RV:: *n* generate_MAC_non_init called as expected
/room enter U
-----
You have entered the room 'U'
You can now type a message directly to the command line.
The message will be encrypted and will be sent to all room users.
For other available actions, please type /roomhelp
-----
GKE|U> test
[U13: test
GKE|U> RV:: *6* The previously received message was decrypted using the shared secret key established during the creation of the room.

```



L-
ta

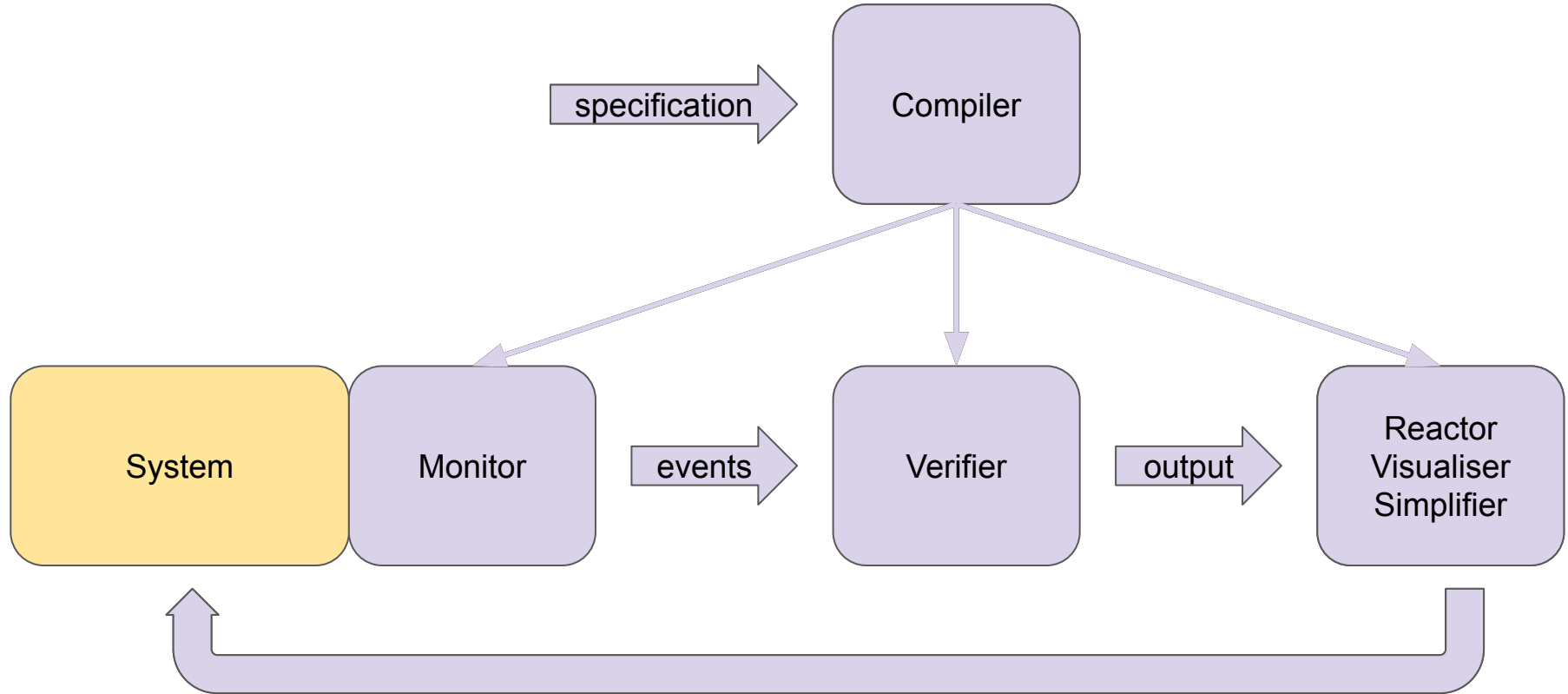


Monitor Alert

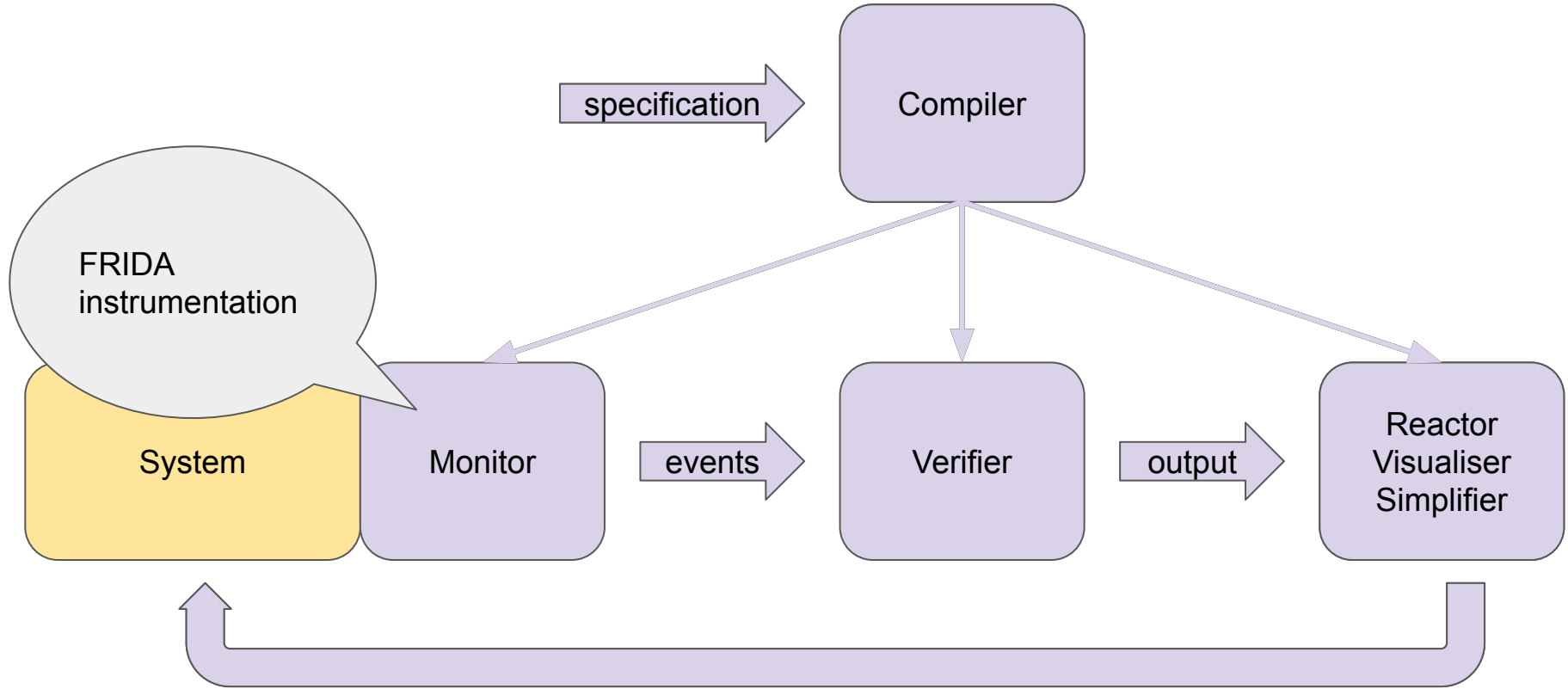
```
GKE|room1> hello
[room1]3: hello
GKE|room1> RV:: *6* The previously received message was decrypted using the shared secret key established during the creation of the room.
máme nový protokol!
GKE|room1> RV::*!WRONG!* Found non-ASCII characters: [109, -61, -95, 109, 101, 32, 110, 111, 118, -61, -67, 32, 112, 114, 111, 116, 107, 108, 33]
[room1]3: máme nový protokol!
```



Where does RV fit in?



Where does RV fit in?



Where is the problem?

Protocol
correctness
checks in terms
of Larva specs

specification

Compiler

FRIDA
instrumentation

System

Monitor

events

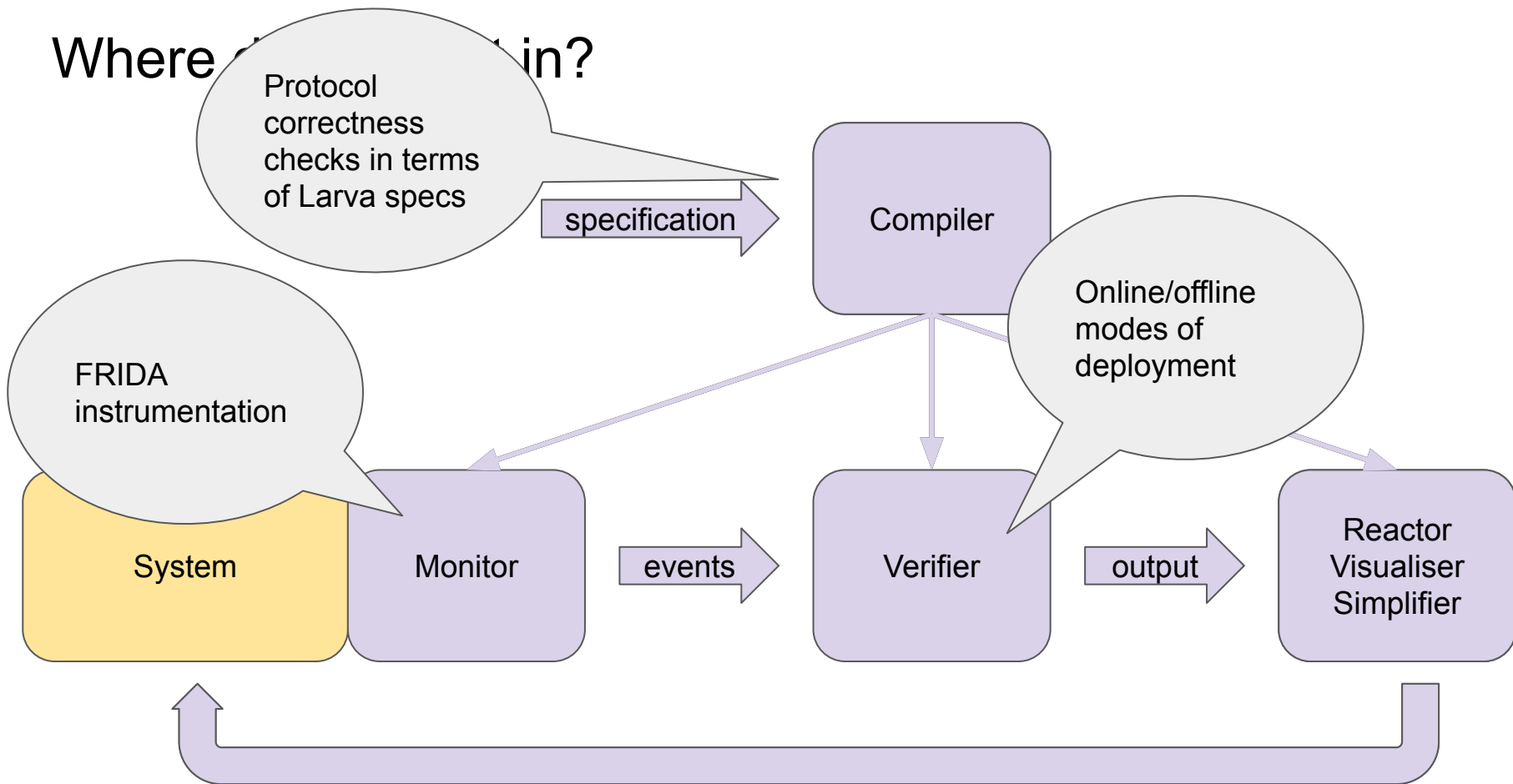
Verifier

output

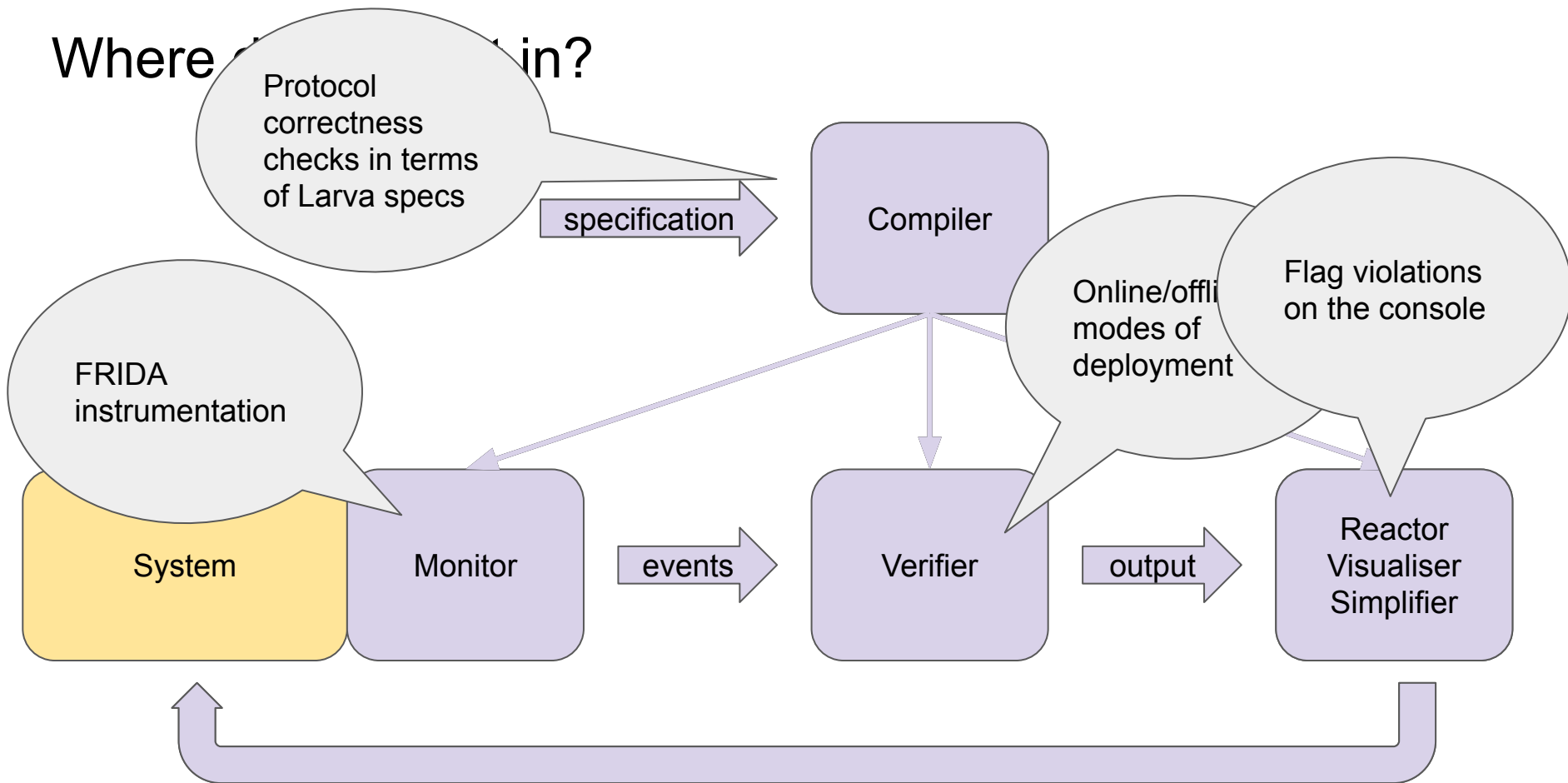
Reactor
Visualiser
Simplifier



Where do we fit in?



Where do we fit in?



Monitoring Overheads

- Scenario A: 3 clients involved, with client `id=1` creating a room (following the protocol steps for an initiator participant U_0).
- Scenario B: 3 clients involved, with client `id=1` joining the room (following the protocol steps for a non-initiator participant $U_{1 \leq i \leq n}$).

The scenarios include 20 and 13 seconds of thread sleeps respectively to mimic a realistic chat. This will be factored in in the results discussion.

Monitoring Overheads

A - Creating a chat room
B - Joining a chat room

| Time (s) | Without SEcube™ | | | Using SEcube™ | | |
|------------------|-----------------|-------|-------|---------------|-------|-------|
| Scenario | A | B | All | A | B | All |
| Non-instrumented | 20.02 | 13.01 | 33.03 | 20.18 | 13.27 | 33.45 |
| Instrumented | 20.44 | 14.39 | 34.83 | 21.30 | 13.68 | 34.98 |



Monitoring Overheads

A - Creating a chat room
B - Joining a chat room

| Time (s) | Without SEcube™ | | | Using SEcube™ | | | |
|------------------|-----------------|-------|-------|---------------|-------|-------|------|
| Scenario | A | B | All | A | B | All | |
| Non-instrumented | 20.02 | 13.01 | 33.03 | 20.18 | 13.27 | 33.45 | 1% |
| Instrumented | 20.44 | 14.39 | 34.83 | 21.30 | 13.68 | 34.98 | 0.4% |
| Increase | 2% | 11% | 5% | 6% | 3% | 5% | |

Instrumentation
is more
expensive than
HSM



Past Work

| # | Context | Tech | Instrumentation | Data aspect | On/offline |
|---|----------|--------|-----------------|----------------------------------|------------|
| 1 | Firefox | C++ | DBI (Frida) | Taint inference on outgoing data | off |
| 2 | Paramiko | Python | AOP (aspectlib) | Limited to parameter checking | off |
| 3 | Chat app | C++ | DBI (Frida) | Monitoring incoming data | on (async) |

Future/Ongoing Work

Protecting the monitor

- The monitor is executed in a protected environment (RunC Container)
- The monitoring logs are encrypted and stored in temper-evident file system (SEALFS)

Cyber Security (2)

Extracting Evidence

(part of Horizon 2020
LOCARD project)

Stealthy Malware - Living Off the Land (LOtL)

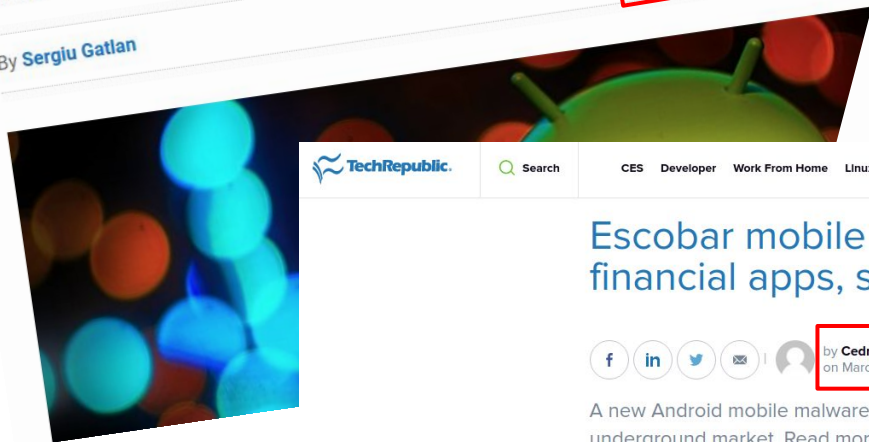
- You don't know you are infected
- Malware can observe what you are doing e.g., through accessibility permission
- Can **unlock** your phone
- Send **messages** from your phone without your knowledge
- Modify **text fields** as you press submit

Home > News > Security > Finland warns of Flubot malware heavily targeting Android users

Finland warns of Flubot malware heavily targeting Android users

November 30, 2021 09:06

By [Sergiu Gatlan](#)



SharkBot: a “new” generation Android banking Trojan being distributed on Google Play Store

RIFT: Research and Intelligence Fusion Team

play.google.com/store/apps/details?id=com...
Google Play
Uncategorized
March 3, 2022 10 Minutes

TechRepublic.

Search

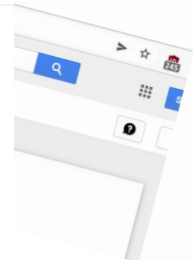
CES Developer Work From Home Linux 5G IT Policy Downloads Security Top DaaS Providers Excel Tips More ▾ Newsletters Forums Res

Escobar mobile malware targets 190 banking and financial apps, steals 2FA codes



by [Cedric Pernet](#) in [Security](#)
on March 17, 2022, 7:18 AM PDT

A new Android mobile malware dubbed Escobar has hit the cybercrime underground market. Read more about it and see how to protect yourself from this threat.



Stealthy Malware - Living Off the Land (LOtL)

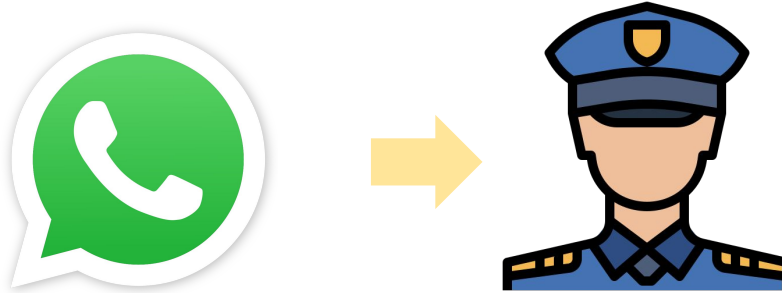
- Delegate sensitive tasks (e.g. sending messages) to benign apps
- Leave little to no evidence behind (no suspicious permissions needed)
- **BUT Cannot** avoid executing in memory

Assumptions

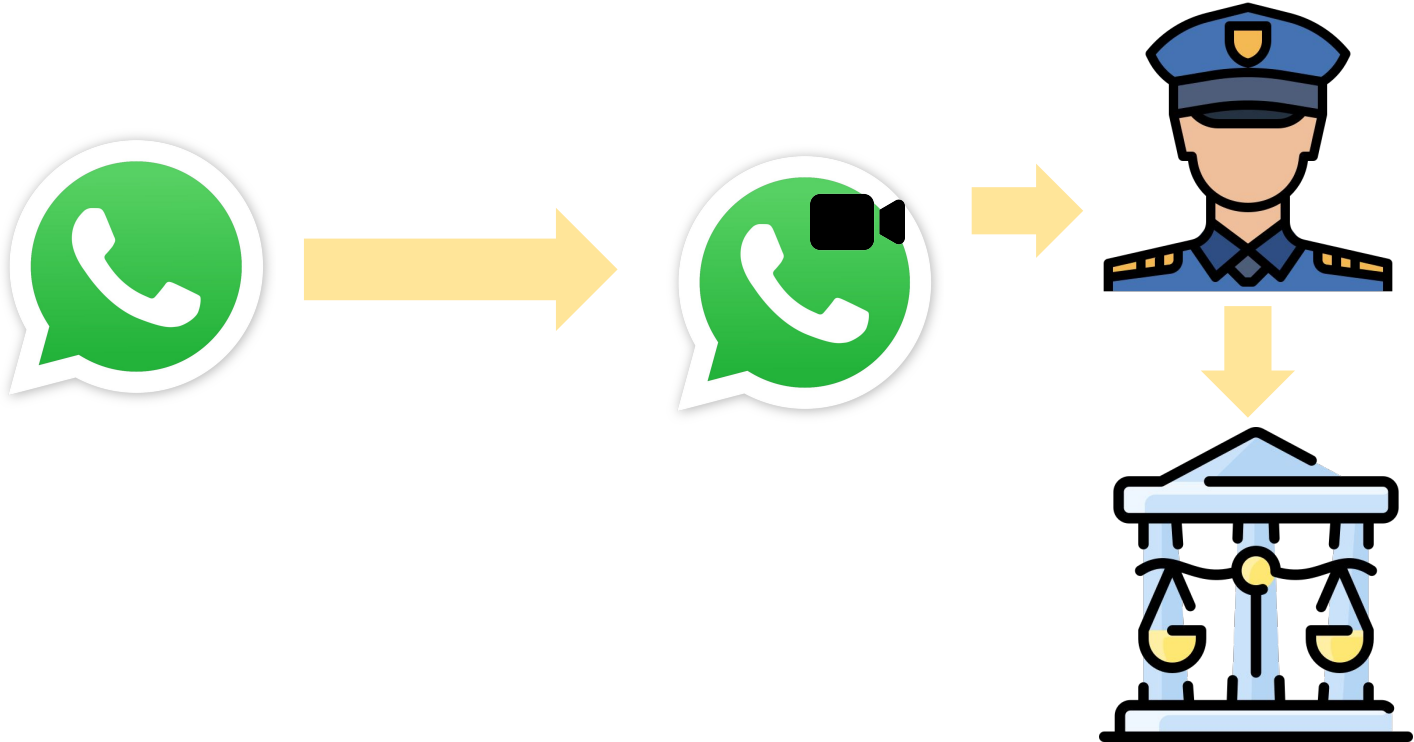
- We don't modify Android
- We don't modify the app
- We want an approach which is easy to use across apps and app versions

Whatsapp Example

Could Whatsapp be sending messages without me knowing it?



Instrumenting Whatsapp



MobFor (ii)

Forensic
readiness

*1. Asset
management*

Targeted

- > Apps
- > Devices
- > Users



MobFor (ii)

Forensic
readiness

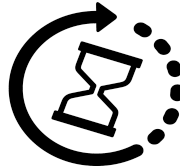
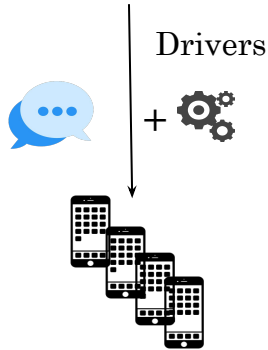
*1. Asset
management*

Targeted

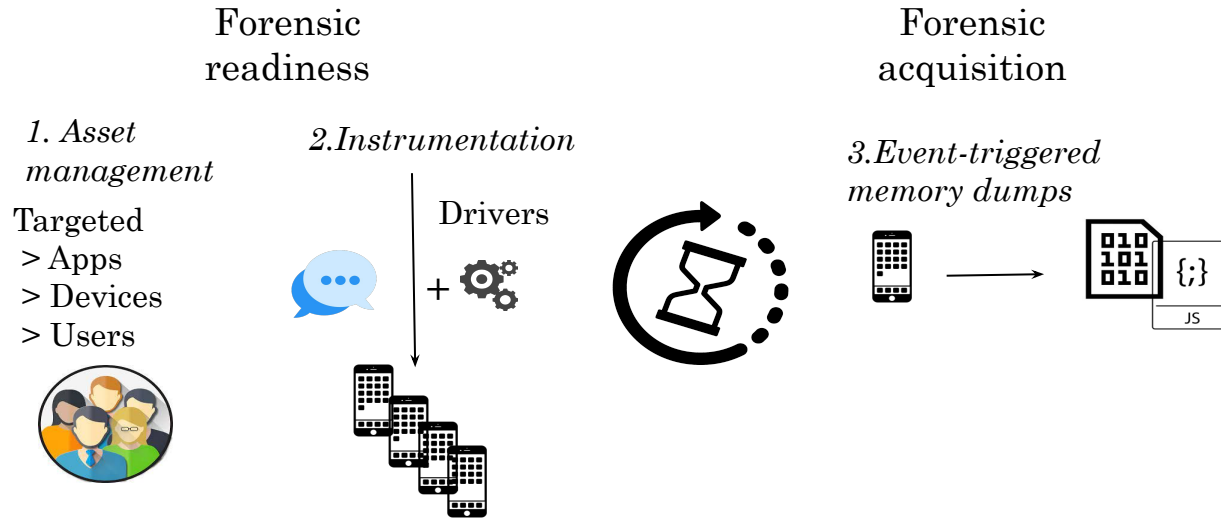
- > Apps
- > Devices
- > Users



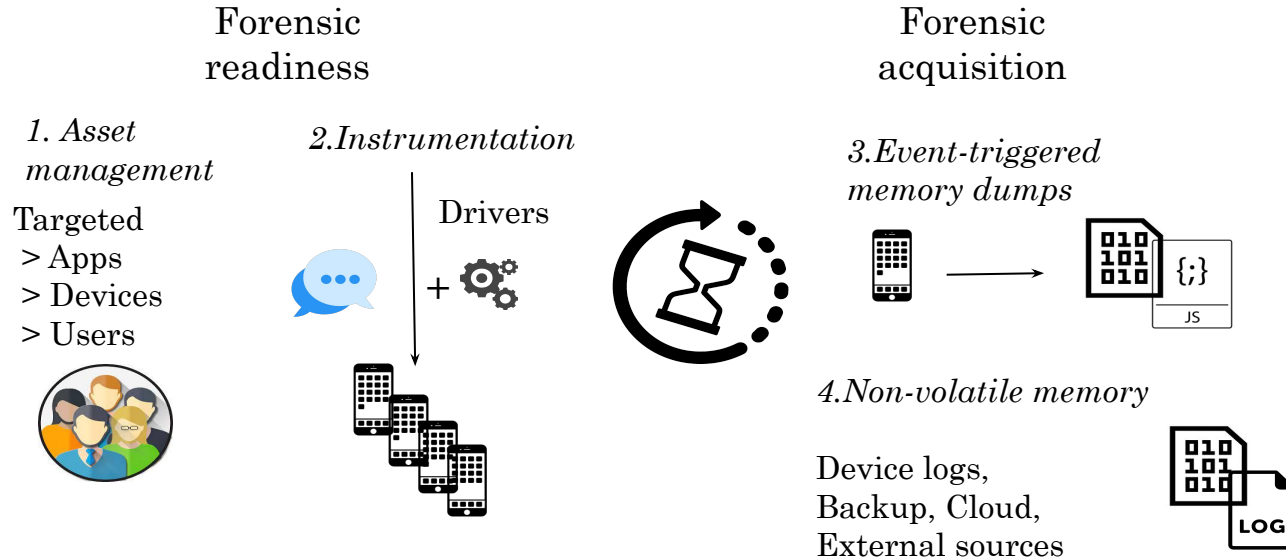
2. Instrumentation



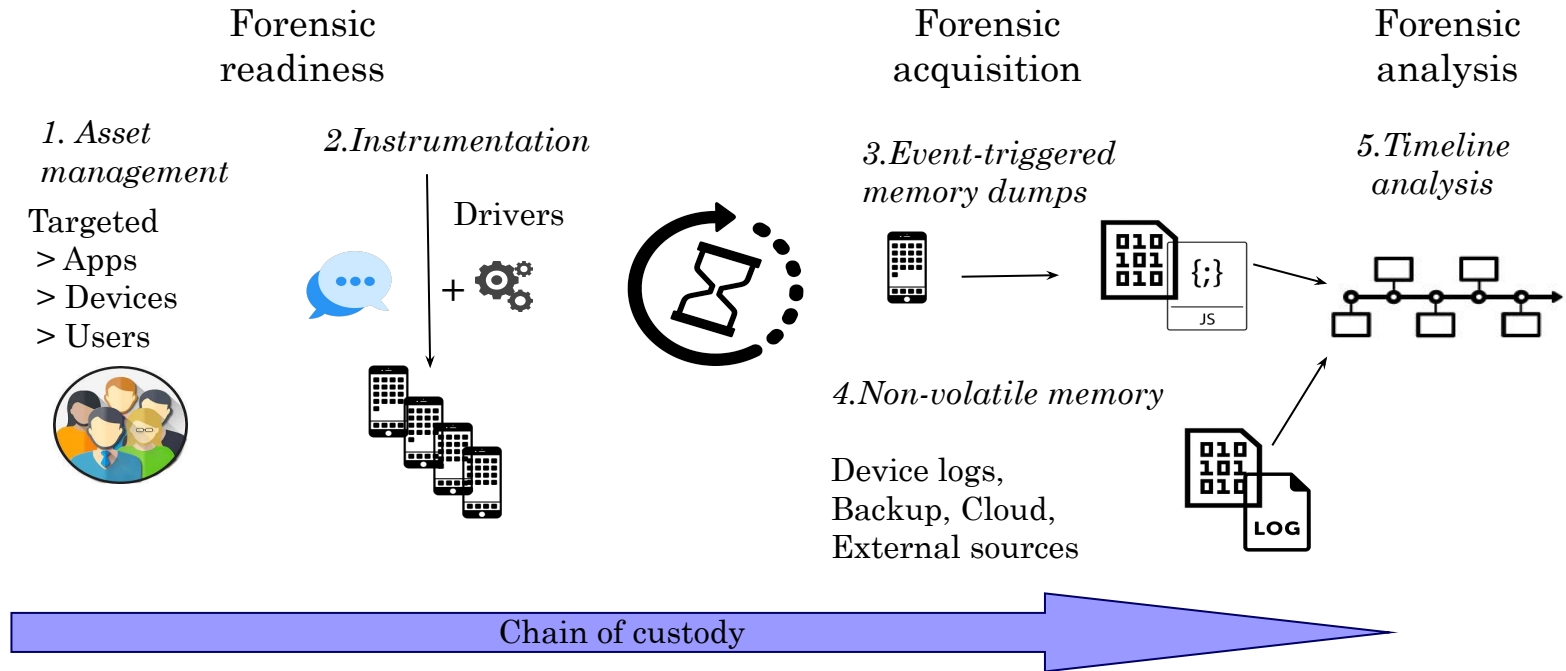
MobFor (ii)



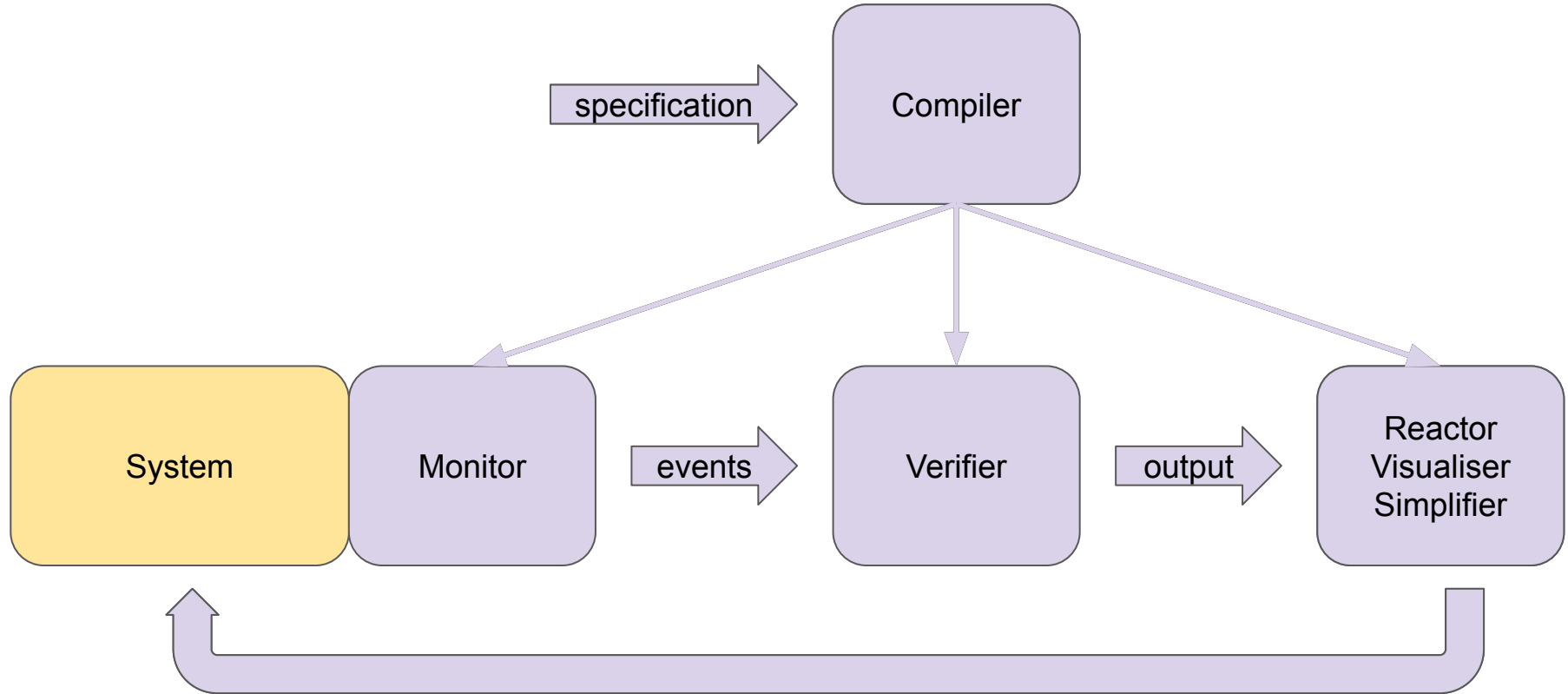
MobFor (ii)



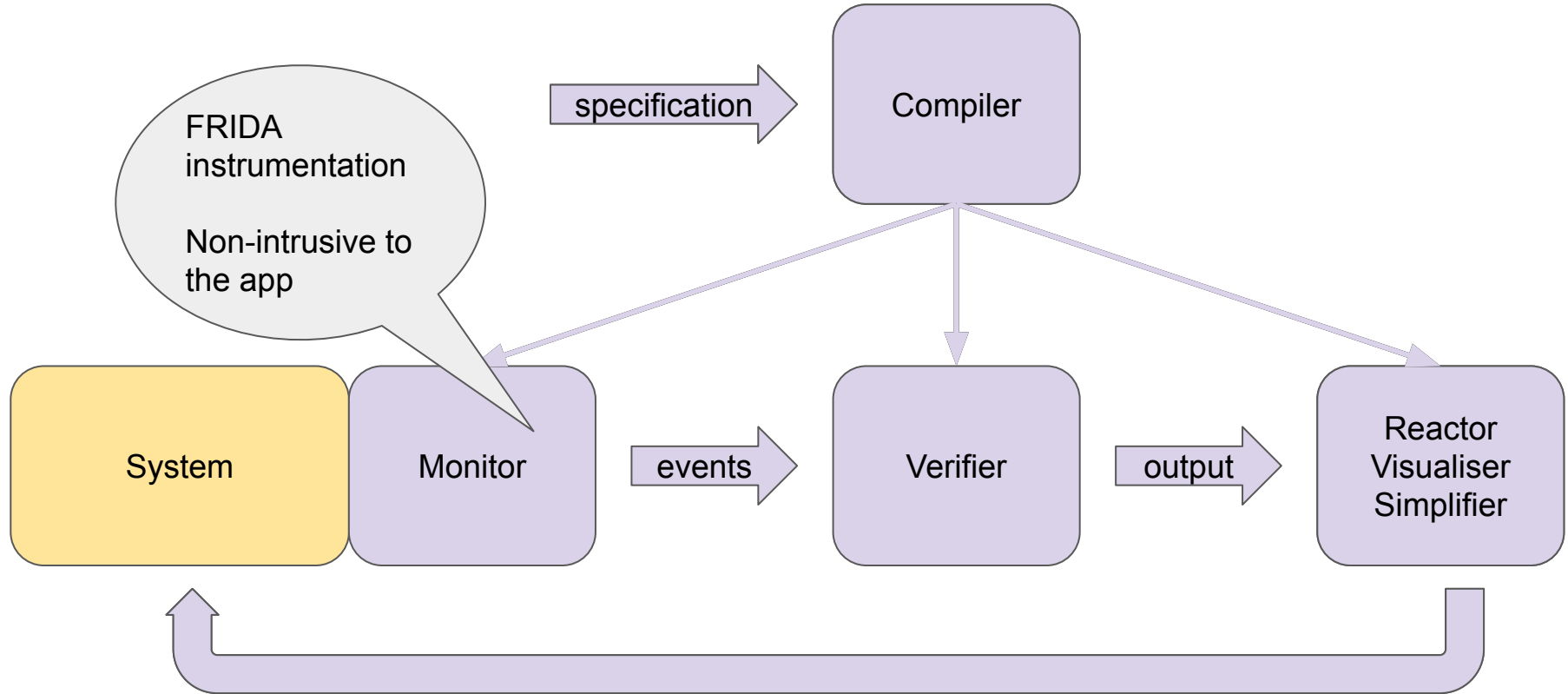
MobFor (ii)



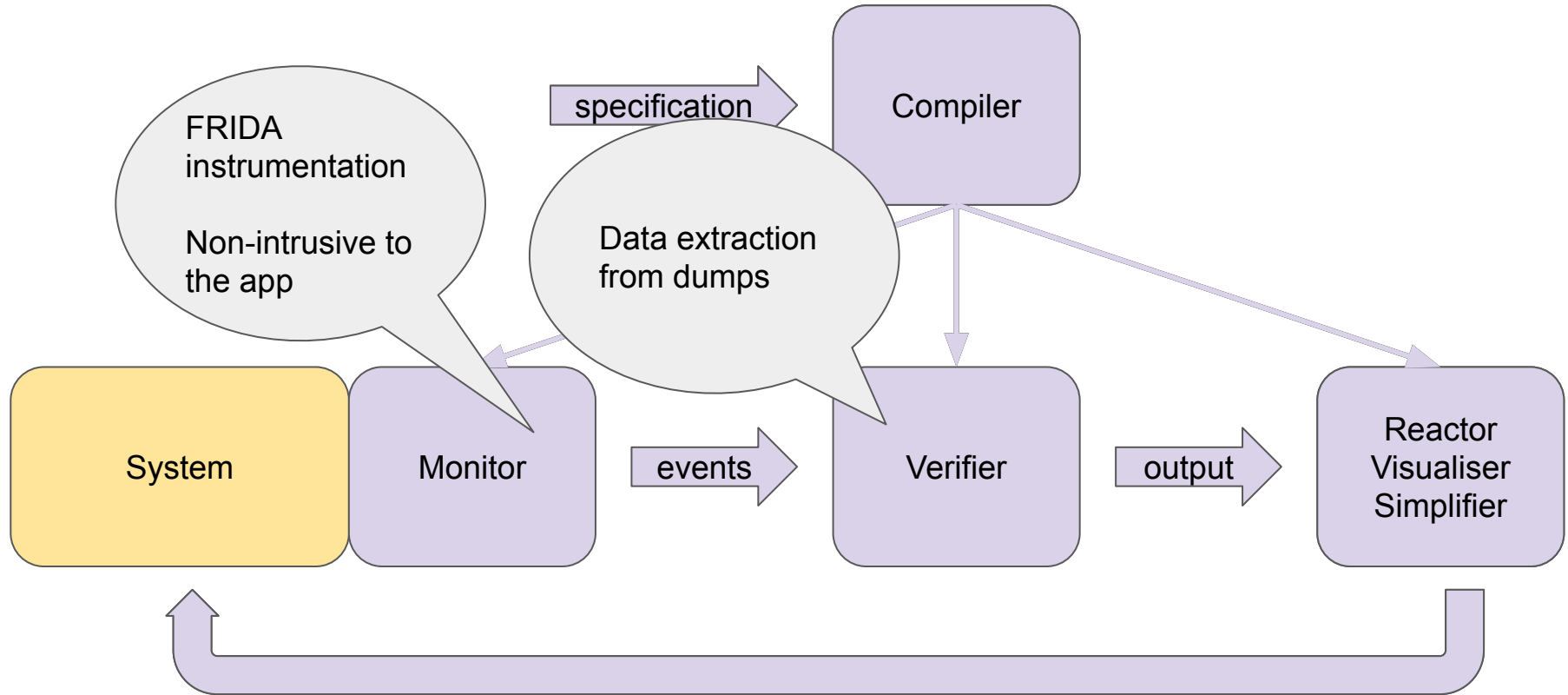
Where does RV fit in?



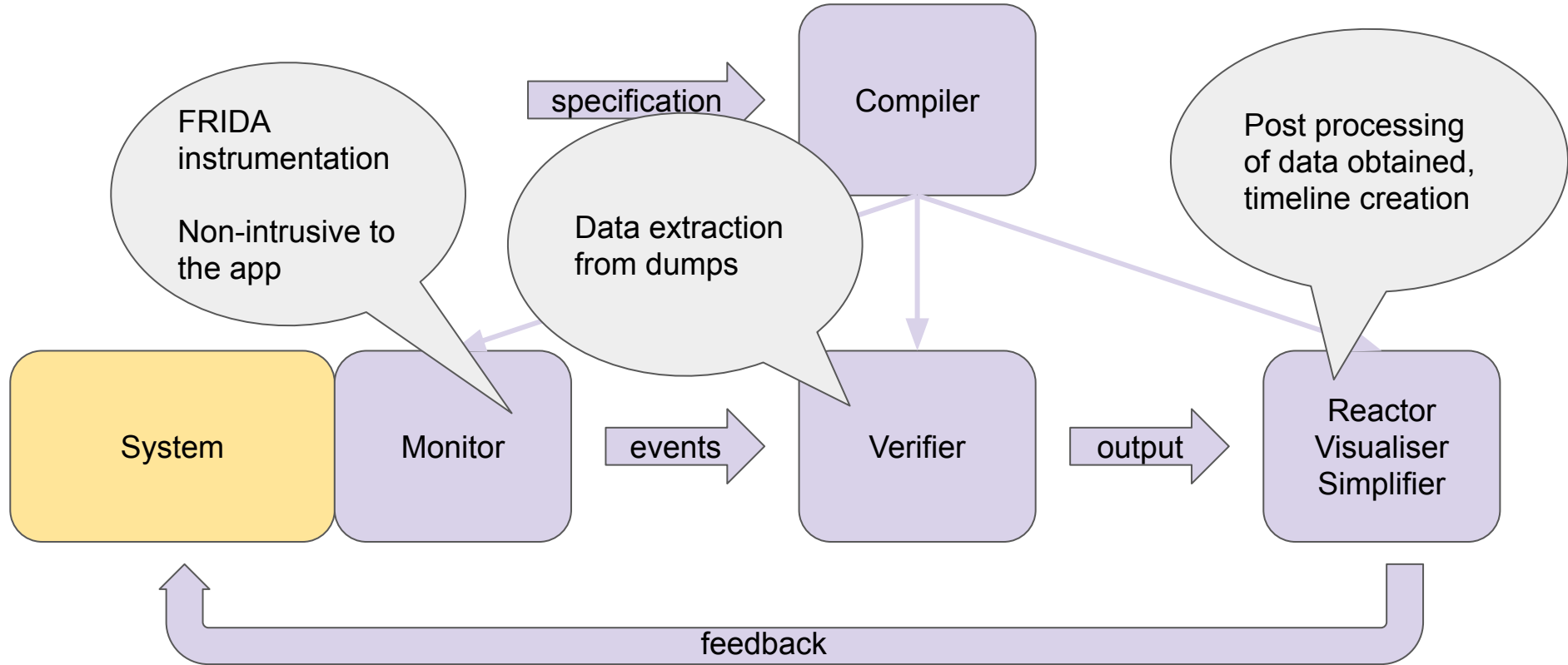
Where does RV fit in?



Where does RV fit in?



Where does RV fit in?



Recent Work

Making instrumentation “plug-and-play”

- Hosting app in virtual app instead of repackaging

- Using infrastructure-based trigger points

Adding anomaly detection to show value added of logged events

Future/Ongoing Work

Trying the same approach on financial apps

Improving the virtualisation system

Conclusions

Conclusions

RV in itself offers two main ideas:

- Formal specifications
- Separation of concerns

The nice thing about RV is that it has a lot to offer to different areas

Two main projects:

- Securing a cryptographic protocol (in conjunction with hardware)
- Extracting events from memory

Questions