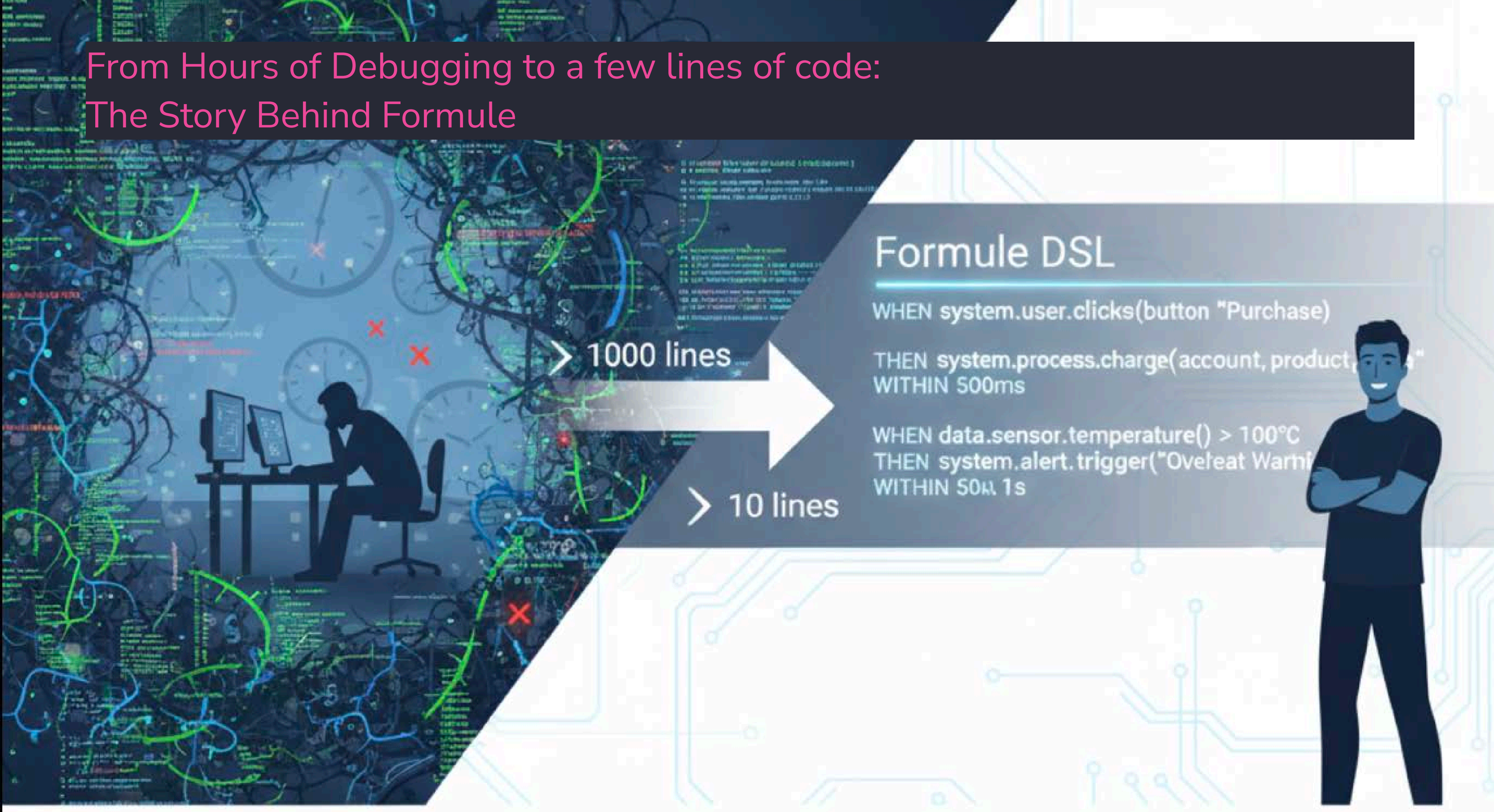
Event-Driven Data Analysis for System Verification and Operation

Requirements as code

Get Started

Ákos Nagy, Márton Szabó



From Hours of Debugging to a few lines of code:
The Story Behind Formule
Formule DSL
WHEN system.user.clicks(button "Purchase)
THEN system.process.charge(account, product,
WITHIN 500ms
WHEN data.sensor.temperature() > 100°C
THEN system.alert.trigger("Oveleat
WITHIN
1000 lines
10 lines

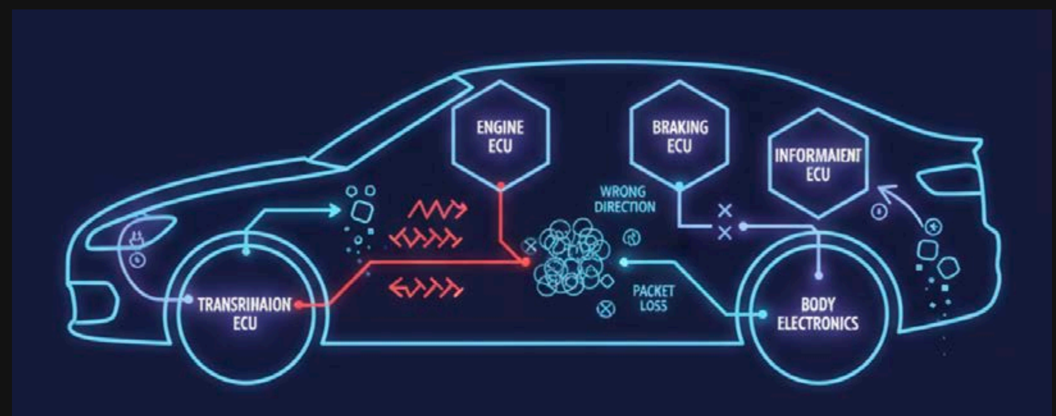
The Problem: A Car That Wouldn't Start

Our story began with a critical issue: brand new test vehicles were failing to start. 🚗 💨

Root cause Tiny communication delays between ECUs.

Effect: A cascade of failures aborting the car's startup.

A **major bottleneck**: analysts spent **hours** manually digging through *massive log files* for each failed car just to find the problem.



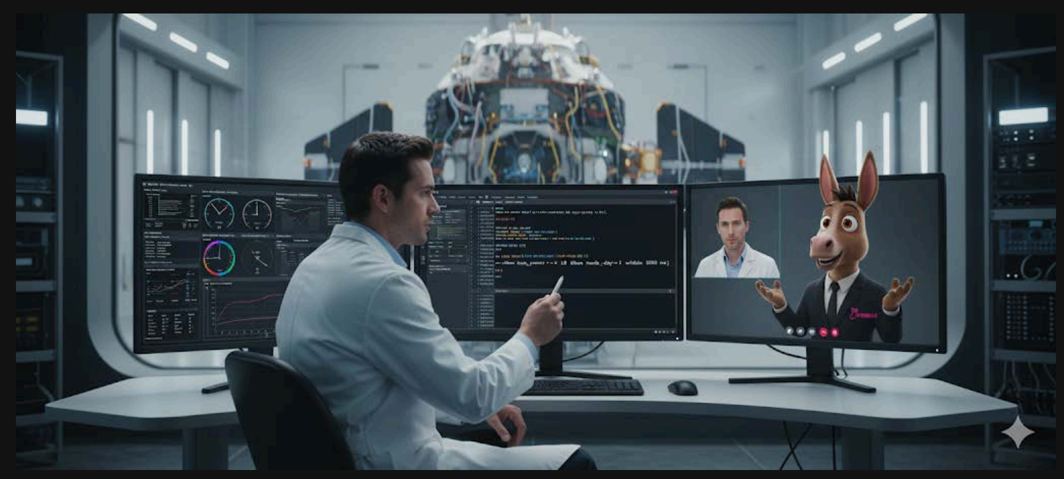
The First Attempt: A Brute-Force Automation

- A **custom** tool in *Java and Groovy* to implement the same checks the analysts were performing
- A *partial victory*, successfully cutting diagnosis time **from hours down to minutes** ✅
- The *catch*: it only worked for **already known root causes**
- Created a new monster: a codebase with **thousands of lines of complex logic**.
- *Slow adaptation*: implementing checks for new issues and requirements **significant software test engineering effort**.
- **No adoptation**: while the developers worked on the tests, the analysts resolved the issues *manually*. By the time the automation was done, the analyst team already moved on to the next issue.

The Breakthrough:

- Shift our thinking from **coding the hunt for errors** to simply **describing the rules for success.** ✨
- Formule has born, an executable Domain-Specific Language (DSL).
- PoC, a dramatic simplification: thousands of lines of Java test code replaced by just **few dozen lines of readable DSL**.
- Not only found many issues, but also provided the **first and most complete documentation** of the subsystem's *(High Voltage Controller Wake-up Sequence)* behavior.
- Not just shorter code; a **paradigm shift** to empower **domain experts.**

For the first time, analysts were able to read, write, and modify test scenarios themselves, without needing a programmer.



Verifying Complex Space Systems

The complexity of space systems is continuously increasing, driven by demands for greater autonomy and sophisticated mission capabilities.

Verifying these systems presents significant challenges.

Communication Clarity

Ensuring clear communication between diverse teams (Requirements, Software, V&V, Root Cause Analysis).

Requirements Traceability

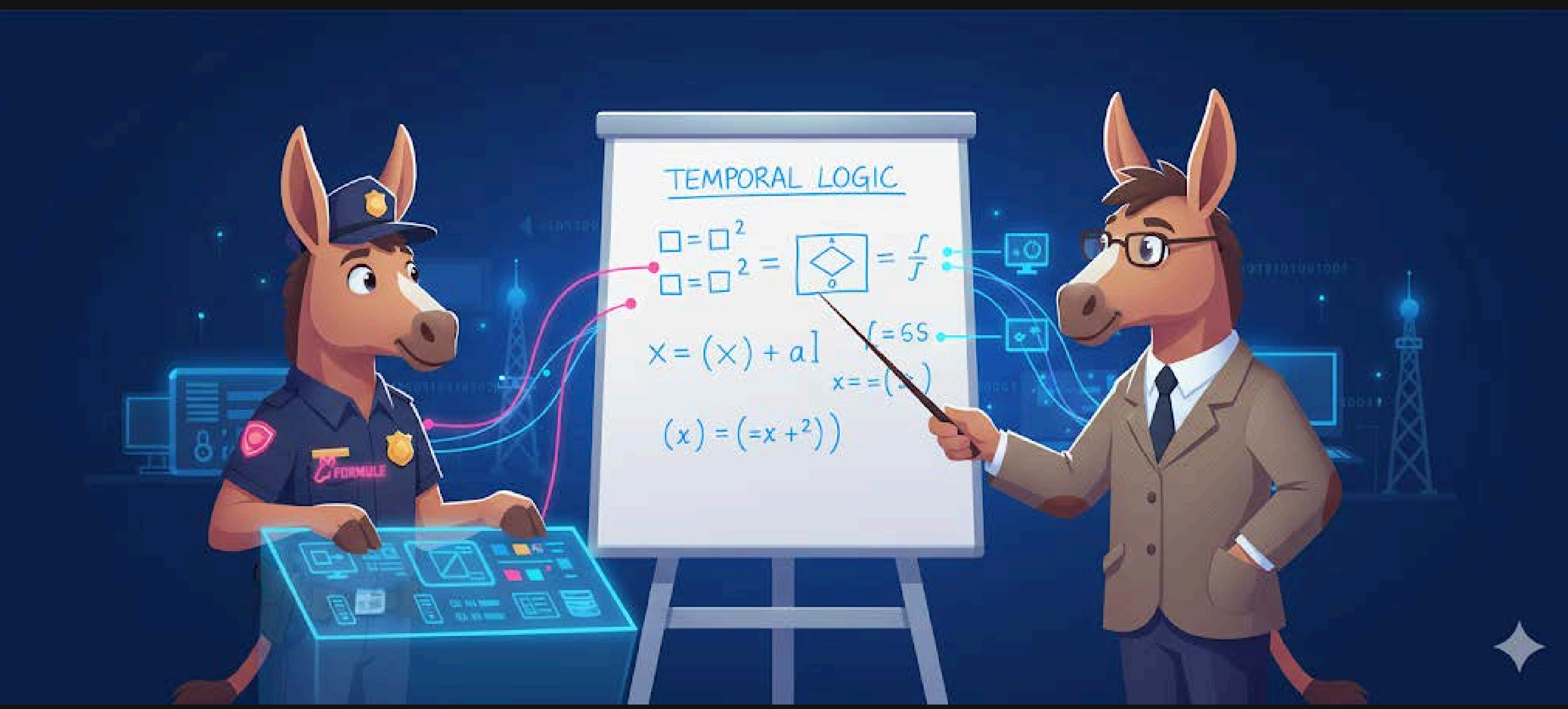
Managing traceability across the application lifecycle and complex mission phases.

Interaction Validation

Validating hardware-software and component-to-component interactions under critical operational constraints.

Methodological Limitations

Overcoming limitations of traditional verification methods.


TEMPORAL LOGIC
FORMULE

The science behind Formule

Formule is a practical application of **Temporal Logic**, a formal system for reasoning about properties of systems over time. It simplifies complex logical concepts into an intuitive, event-driven syntax.

The DSL directly maps to concepts found in formal languages like **Linear Temporal Logic (LTL)**:

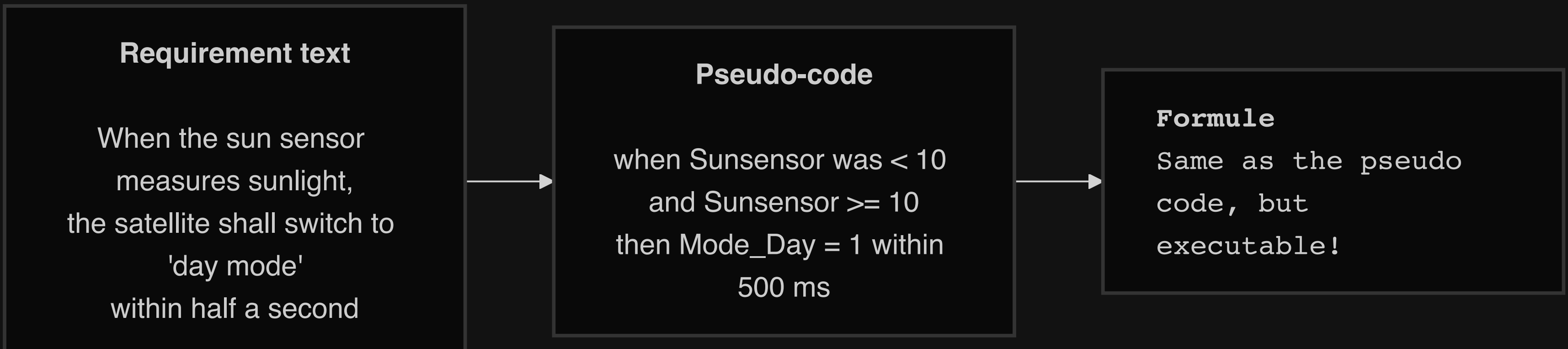
- `G(<condition>)` in LTL becomes `ASSERT <condition>`
- `G(trigger→F(post_condition))` in LTL turns into `WHEN <trigger> THEN <post-condition>`
- Real-time constraints from *Metric Temporal Logic*, e.g.: `WITHIN <time>`

In essence, our DSL provides the power of *formal verification* without the *steep* learning curve of writing raw LTL formulas, making it ideal for engineering requirements.

From text to test

Real-time validation of satellite bus telemetry against operational limits

Let's check this requirement from the YAMCS* quickstart demo:



* *YAMCS /jæmz/ is an open source mission control software developed by **Space Applications Services**. This example demonstrates verification of real-time telemetry data from YAMCS quickstart demo*

Systems work as expected.
www.formule.dev

Turn it into a Formule!

Requirement: *When the sun sensor measures sunlight, the satellite shall switch to 'day mode' within half a second.*

```
1  [requirement id: "REQ-001"]
2  when myproject → Sunsensor was < 10 and
3       myproject → Sunsensor ⩾ 10
4    then myproject → Mode_Day = 1 within 500 ms
```

- **REQ-001**: When the signal of the sun sensor changes from a value less than 10
- **...**: to a value greater than or equal to 10
- **...**: then 'mode day' is active within 500 ms

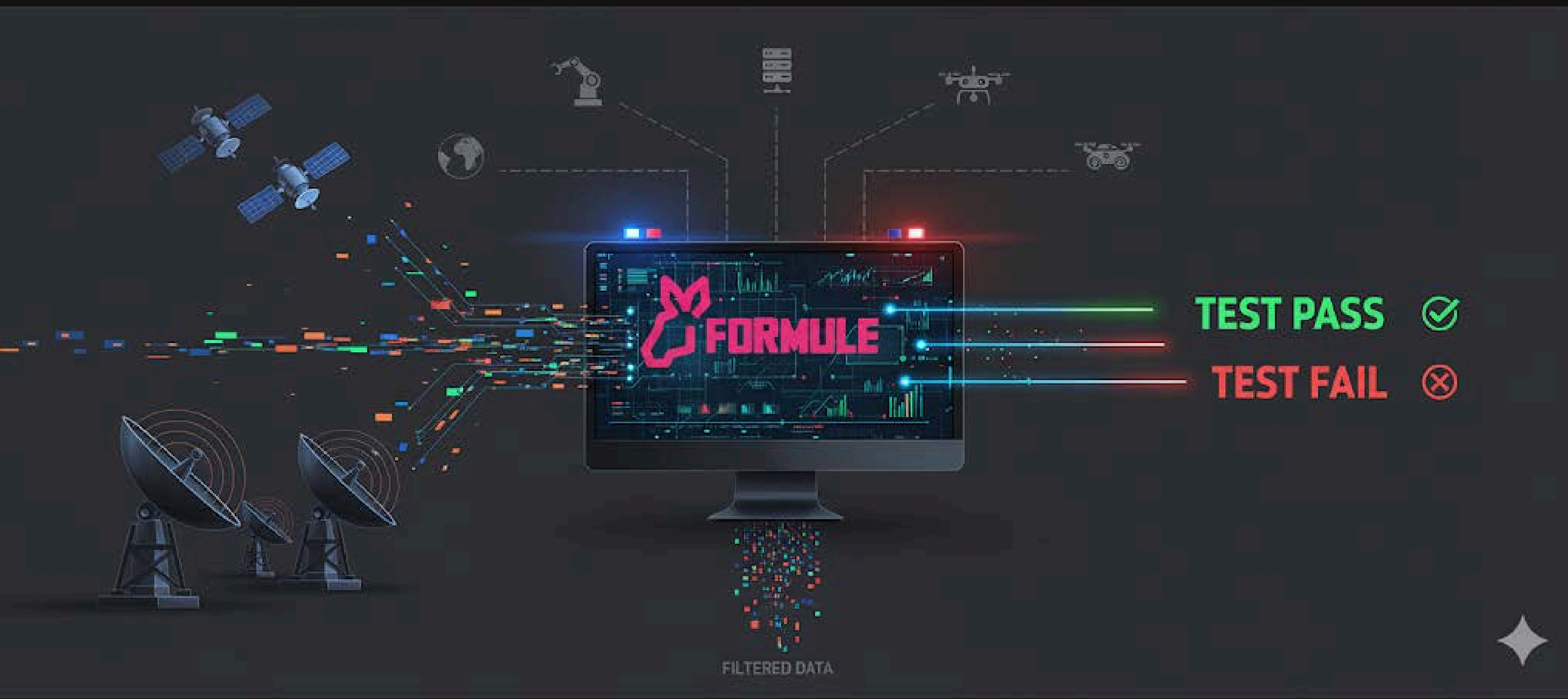
Notice how close it is to our *pseudo-code*!

One more thing ...

```
1  event day_mode_alert
2  output signal day_mode_error is int
3
4  when myproject → Sunsensor was < 10 and myproject → Sunsensor ≥ 10
5    then myproject → Mode_Day = 1 within 500 ms
6    on fail emit day_mode_alert
7    on fail set day_mode_error to 1
8    on fail collect myproject → Sunsensor, myproject → Mode_Day from -5s to +2s
```

- Use normal Formule *requirements* like before
- Define what to do when the verification check *passes* or *fails*
  - Emit events
  - Send output signals
  - Capture relevant signals in a defined time frame
- Use `` `C` ``, `` `Python` `` or `` `Lua` `` bindings to handle collected data

Systems work as expected.
www.formule.dev


FORMULE
TEST PASS
TEST FAIL
FILTERED DATA

Meet FORMULE

- A DSL designed to empower domain experts.
- An event-driven data analytics engine.
- Directly interacts with system data like telemetry and command (TC) logs.
- Supports real-time analysis.
- Run on a PC, server, cloud, or embedded device.
- Integrate into existing pipelines.
- Stream filtered data to other systems.

Architecture of the Formule Ecosystem

Factory
IoT devices
Airplane
Train
Car
Satellite
Control Center
Cloud
Services
Kubernetes
Database

MQTT
logs
Event bus

in-house tools
Python script
Standard analytic tools
Mercury
Log Management

Requirements as Formule scripts
Analytics engine
FORMULE

Python script
Alerts
SQL
Time series dataset

Report
Slides
Chart
Dashboards

Data source

Data can be captured from various sources, either real time or recorded into trace files

Dissector

In order to analyze data, it shall be dissected into events, messages and signals

Analysis

Formule scripts declare time-bound requirements on events, messages and signals

Results

Triggering events and checks can be either persisted into time series data or processed with Python scripts

Visualization

Put your results on real time dashboards, channel into health monitoring systems or generate printable reports, PowerPoint slides, etc.

Systems work as expected.
www.formule.dev

How this helps you reducing costs?

Requirements as code

- Domain Experts to maintain their own tests
- Faster turnaround, less dependency on programmers
- Replace requirements text, no need to sync

Lower data volume

- Filter telemetry, reduce data
- Lower bandwidth and storage usage
- Cut telemetry files, focus on relevant data

Stream data to other systems

- Control test benches
- Trigger external loggers
- Update dashboards
- Send alerts

Like a query language

Analysts can write Formule scripts to catch typical root causes of issues, narrowing down the problem, eliminating repetitive work in root cause analysis.

What about that big elephant in the room?

Options for using LLMs

~~1. Transform requirements to executable code~~

```
No, we don't do that.
Tried with limited success back in 2023. Nowadays it is possible,
but requires deep review of generated code.
```

~~2. Interpret Formule scripts and requirements using an LLM and use RAG to collect data~~

```
No, we don't do that, either.
We want deterministic results and we want them fast.
```

3. Translate requirements from text to Formule script

```
Since Formule DSL is very close to the language of requirements,
it is possible to translate requirements to Formule using LLMs with
high accuracy and low effort.
```

4. Make Formule output available to LLMs and let the user ask questions about it

```
We can stream Formule output (Test results, incoming / generated signals, events) into a database.
Then we might use RAG to make this data available for LLMs.
```

From requirements to Formule using LLMs

- Input: Formule language specification and the YAMCS example
- Prompt: Requirements text (see comments below)
- Output: Formule script

```
1   // when the system wakes up, the battery management system switches to ready mode within half second
2   when SYS_state turns from SYS_STATE_INIT to SYS_STATE_RUN
3     then BMS_modus is BMS_MODE_READY within 500 ms
4
5   // when the battery management is ready, the output stack voltage is between normal limits within 50 ms
6   when BMS_modus turns BMS_MODE_READY
7     then U_STACK ⩾ U_STACK_NORMAL_MIN and U_STACK ⩽ U_STACK_NORMAL_MAX within 50 ms
8
9   // BMS stack voltage can't go higher than 26V
10  assert U_STACK ⩽ U_STACK_MAX
11
12  // when stack voltage is lower or higher than normal while BMS is running,
13  // BMS goes to ERROR mode within 50 ms
14  when BMS_modus is BMS_MODE_READY and (U_STACK < U_STACK_NORMAL_MIN or U_STACK > U_STACK_NORMAL_MAX)
15    then BMS_modus is BMS_MODE_ERROR within 50 ms
```

Conclusion

Formal, declarative language
Human-readable. No software engineering skills are needed. Executable. Automated verification of requirements.

Improved traceability
Reduce costs by removing the need to manually sync requirements and tests. Use Formule for both requirements and test specifications.

Stable and deterministic
No AI, no LLMs, no uncertainty in the core. Pure verification. Each execution on the same dataset yields the same results.

AI in the loop
Use LLMs to prepare requirements or postprocess Formule output.

Ready for on-board execution
Embedded runtime with minimal dependencies and a small footprint. Update Formule scripts without changing the system software.

Reduce data volume
Capture and transmit only relevant diagnostic data. Cut telemetry files so analysts can focus on relevant data.

Ready to Transform Your System Specifications?

Get Started · Documentation · Contact Us

Requirements to Verification - Systems work as expected.