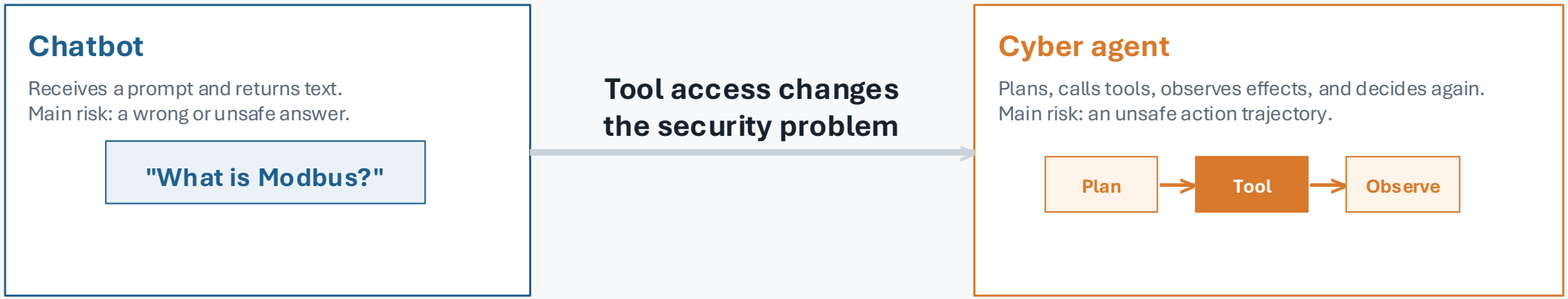
From Chatbot to Cyber Agent

Governing AI that acts on cyber infrastructure



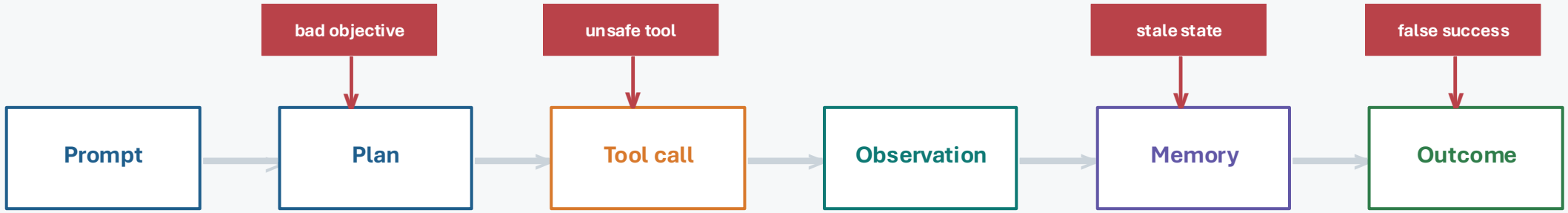
Adel ElZemity
3rd Year PhD in Computer Science, University of Kent

Read my work and connect:
adelsamir.com
ae455@kent.ac.uk

University of Kent

The attack surface moves from prompts to action loops

With agents, we evaluate the sequence: plan, tool call, observation, memory, next action, and outcome.



A classifier can be evaluated example by example

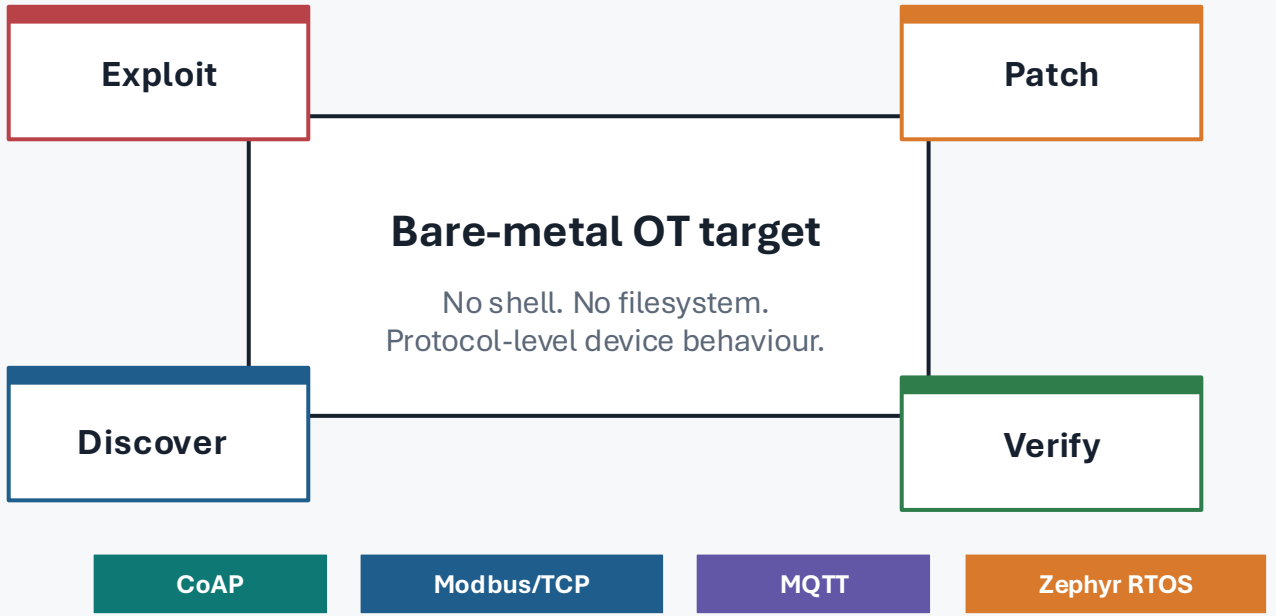
Input -> prediction -> label comparison. Mistakes are local and usually bounded by the prediction interface.

An agent must be evaluated as a trajectory

One mistaken action can change what the next observation means. Reliability depends on the whole loop.

APIOT tests a full autonomous OT security mission

The agent is not asked for advice; it must complete discovery, exploitation, patching, and verification.



**Focus: not whether the model knows cyber security,
but whether the action system can finish the mission reliably.**

Bare-metal OT removes the shortcuts agents usually rely on

A useful cyber agent must reason over protocols and device responses, not just familiar operating-system abstractions.

Typical web/Linux target

Shell
commands, processes, files

Filesystem
paths, logs, configs

Known tools
nmap, curl, ssh, web scanners

Common intuition
Linux and HTTP are heavily represented

NO SHELL

Bare-metal OT target

Protocol fields
length, options, function codes

Parser behaviour
malformed packet handling

Limited memory
kilobytes and firmware constraints

Device response
crash, timeout, register change

This forces a cleaner capability test: can the agent reason through the protocol itself?

DESIGN PRINCIPLE

Action-space design is a security decision

Before asking whether the agent is smart, define what it can observe, what it can do, and what it can never do.

Allowed	Constrained	Forbidden
Observe topology	Move between phases	Reset the lab
Send protocol primitives	Generate patches	Use hidden privileges
Inspect responses	Repeat failed actions	Call exploit macros
Verify impact	Declare success	Skip verification

Practical rule

If the agent has a magic exploit button or hidden reset access, you are measuring orchestration privilege rather than cyber reasoning.

Agent failures are systematic, not random

The useful move is to recognise repeated failure shapes, then design controls for them.

Repetition loop

The agent repeats a low-value action instead of changing strategy.

Phase deadlock

The agent gathers more context but never crosses into the next mission phase.

Missing verification

The agent declares success without evidence that exploit impact or patch effect was confirmed.

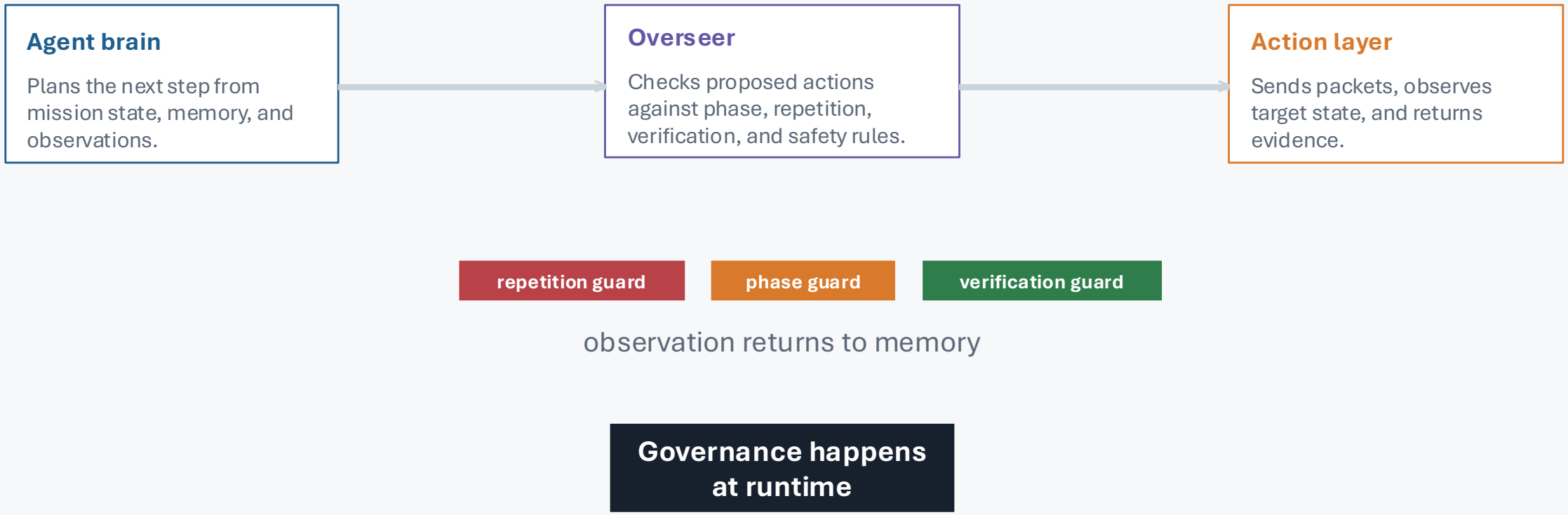
Protocol confusion

The agent applies reasoning from the wrong protocol or abstraction layer.

Design implication: controls should target action sequences, not just individual prompts.

The Overseer sits between intention and action

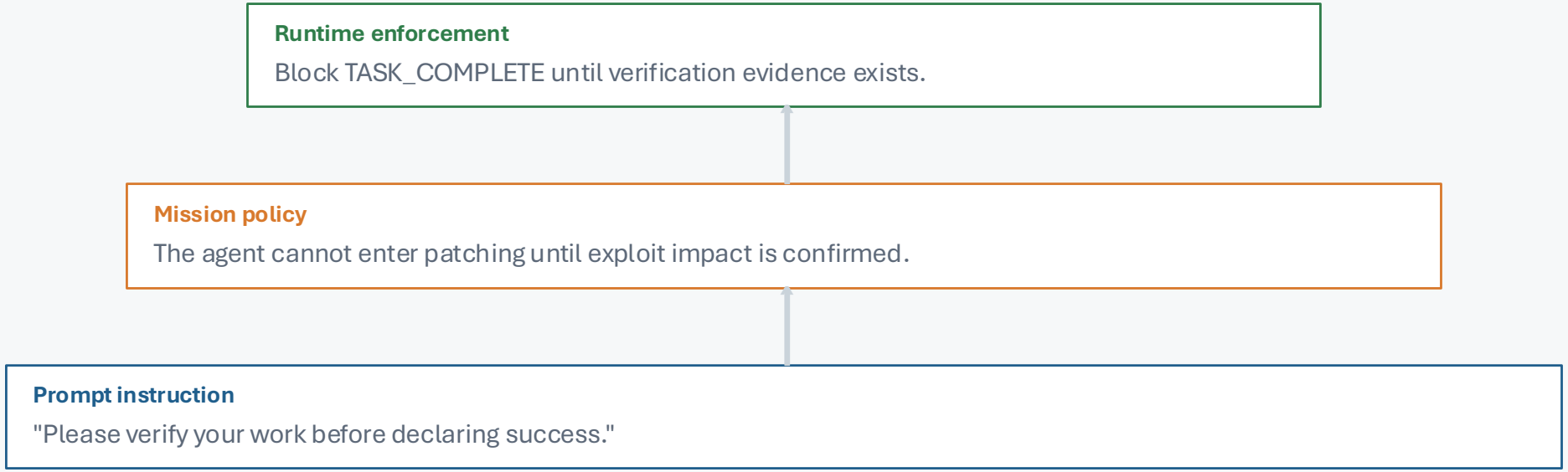
It does not need to be a smarter agent. It is a control layer that changes which action sequences are possible.



PROMPTING VERSUS GOVERNANCE

Instructions influence behaviour; governance constrains behaviour

A good prompt is helpful. A runtime rule changes what the agent is allowed to do.



The higher you go, the less you rely on the model remembering to behave.

EVIDENCE FROM APIOT

The system structure changed mission reliability

The lesson is not that one model was clever; it is that governance changed the action trajectory.

290
experiment runs across models, topologies, protocols, and impairment levels

90.0%
mission success rate for the full discovery -> exploit -> patch -> verify cycle

20.5%
faster completion in the overseer-on ablation condition reported in the paper

Overseer OFF

More exposure to repetition loops, reconnaissance deadlocks, missing crash verification, and phase mistakes.

Overseer ON

Cleaner phase progression, enforced verification, and structurally blocked bad sequences.

Practical interpretation: runtime governance is a reliability lever, not decorative safety text.

Source: APIOT paper, 290-run evaluation and overseer ablation

PRACTICAL FRAMEWORK

Every cyber agent needs a safety case

Observe

What can the agent see, and what evidence is trusted?

Act

Which tools and protocols can it use?

Block

Which actions or sequences are forbidden?

Verify

What evidence is required before success?

Log

Can we reconstruct the trajectory afterwards?

Escalate

When must a human take over?

A safety case is a claim about the whole action system, not a vibe about the model.

IN-CLASS EXERCISE

Design the governance before you trust the agent

Scenario: you are building an AI agent that scans a small network for vulnerable IoT devices and recommends fixes.

Two-minute task: define the agent's boundaries before defining its intelligence.

Assets
What matters if the agent is wrong?

Tools
What can it call?

Forbidden actions
What must never happen?

Verification
What proves success?

Overseer rules
What sequences are blocked?

Autonomy without governance is not security engineering

A cyber agent is trustworthy only when its actions are constrained, observable, and verified.

Constrain **Observe** **Verify**

Thank you!