

# The "Blue Team" Cyber-Defense Orchestrator

AI Planning & Reasoning Project - A.Y. 2025/2026

PRESENTED BY



Arianna Viola

ID: 1868485



Rossella Milici

ID: 2208814



Gianmarco Corsi

ID: 2212369

# Motivation & Goals

Context and Goal of the Blue Team Agent

## ⚠ THE CONTEXT: MODERN THREATS

Corporate networks face constant automated threats like **Worms** and **Ransomware**. Manual intervention is often too slow to prevent their spread.

## 🛡 THE SOLUTION: "BLUE TEAM" AGENT

- 1 **Monitoring:** Real-time health analysis.
- 2 **Planning (PDDL):** Generation of remediation strategy.
- 3 **Reacting (IndiGolog):** Autonomous response to intrusions.

## ◎ MAIN OBJECTIVE

Maintain the CIA Triad:

**Confidentiality:** Protect sensitive data on Database Servers.

**Integrity:** Ensure nodes remain 'Clean' and free of malware.

**Availability:** Minimize service downtime during containment and remediation.



# Domain Taxonomy

IT Environment Modeling: Entities and Dynamics



## NETWORK NODES

### Workstations

HIGH RISK

Low-priority user devices but high risk of initial infection.

### DB Servers

CRITICAL

Central hubs containing sensitive data. Their integrity is critical.

### Web Servers

Public interfaces. They depend on DB Servers to operate correctly.



## ADVERSARIES

### Worm

Self-replicating malware that spreads rapidly and linearly across connected nodes.

### Ransomware

Encrypts data, making it inaccessible. Requires expensive backups or decryption keys.

# Agent Actions

## DEFENSE ACTIONS

### 1. **scan(Node)**:

- Identifies the infection status of a node

### 2. **isolate\_node(Node)**:

- Prevents malware spread but reduces service availability
- Degrades services of dependent nodes (ADL conditional effect)

### 3. **patch(Node)**:

- Removes malware and fixes vulnerabilities

### 4. **restore\_backup(Node)**:

- Recovers encrypted data from backups

### 5. **reconnect(Node)**:

- Brings isolated node back online

## SUBNET MANAGEMENT ACTIONS

### 6. **restart\_router(Subnet)**:

- Restarts network router for a subnet
- Effect: Brings subnet back online

### 7. **verify\_connection(Subnet)**:

- Confirms subnet connectivity after restart
- Always possible (no preconditions)

### 8. **wait\_step**:

- Advances the simulation by one time step
- Used in monitoring loops between scans

# Environment Dynamics & Constraints

## NUMERIC CONSTRAINTS (RESOURCES)

To simulate real-world limitations, we introduce numeric variables:

**Network Bandwidth:** Limited throughput capacity per time-step. Heavy operations like Restore require available bandwidth to be executed.

**Max Bandwidth Limit:** Fixed threshold that defines the network's maximum operation capacity.

**Downtime Costs:** Each node has a priority weight and isolating a node increases downtime cost by its weight  
• High-value nodes (CoreServer: 100) vs. workstations (10)

## OPTIMIZATION CHALLENGE

The core complexity lies in the ***Security vs. Availability trade-off:***

**Isolating** threats prevents spread but halts business operations.

**Keeping services running** risks further infection.

**Goal:** Minimize downtime costs while maintaining integrity.

# PDDL Formalization

## State Predicates and Cost Functions

### KEY PREDICATES

STRIPS

ADL

*;; Network Topology & State*

```
(connected ?n1 ?n2)
(infected ?n ?m)
(isolated ?n)
(clean ?n)
```

*;; ADL Features (Conditional Effects)*

```
(depends-on ?n1 ?n2)
(service-degraded ?n)
```

### NUMERIC FUNCTIONS

NUMERIC

*;; Resource Constraints*

```
(bandwidth-used) // total bandwidth consumed by actions
(max-bandwidth)
;; Optimization Metrics
(downtime-cost) // accumulated cost of service interruption
```

*;; Static Properties*

```
(priority-weight ?node)
```

*// asset value(es. DB=100, PC=10)*

**ADL Note:** We use depends-on to model fault propagation (e.g. if the DB is isolated, the WebServer is service-degraded).

**Global Optimization:** The planner optimizes exclusively for (downtime-cost). Bandwidth is tracked as a resource consumption but is not part of the minimization metric in this instance.

# Heuristics Strategy

Comparison of Fast Downward Approaches



## Blind Search

BASELINE

Uninformed search (Breadth-First).  
Explores the state space without any  
heuristic guidance.

- ✓ No computational overhead  $h(n)$
- ✓ Useful for measuring complexity
- ✗ Exponential node expansion

REFERENCE



## $h_{\text{add}}$

ADDITIVE

"Satisfying" heuristic. Adds the costs of  
the subgoals, ignoring negative  
interactions.

- ✓ High search speed
- ✓ Strong guidance towards the goal
- ✗ Not admissible

MAX SPEED



## $h_{\text{max}}$

ADMISSIBLE

Conservative heuristic. Considers only  
the maximum cost among the parallel  
subgoals.

- ✓ Guarantees Optimality
- ✓ Admissible (always underestimation)
- ✗ Less informative than  $h_{\text{add}}$

OPTIMAL PLAN

# Problem 1 - Worm Outbreak

Linear Scenario (STRIPS)

## TOPOLOGY & STATE



**⚠ THREAT ACTIVE: NODE A**

## CONTEXT

The network has a simple linear topology. Node A has been infected with a worm. Without intervention, the malware will sequentially propagate to Node B and then to Node C. The agent must act quickly to contain the threat and clean up the system.

## PDDL GOAL

```
(:goal (and
  (clean nodeA)
  (clean nodeC) ; Ensure spread stopped
  (not (isolated nodeA)) ; Restore Service
))
```

## EXPECTED PLAN



**Logic:** Isolation is a necessary precondition for the **patch** action, preventing spread during remediation.

# Problem 1 - Experimental Results

"Worm Outbreak" - Linear Topology

## Blind Search

BASELINE



<input checked="" type="checkbox"/> Status	Solved (Opt)
Plan Cost	4
Expanded	6
Evaluated	6
Time (s) (avg on 10 runs)	<b>0.45256</b>
Initial h	1

## $h_{\text{add}}$

ADDITIVE



<input checked="" type="checkbox"/> Status	Solved (Opt)
Plan Cost	4
Expanded	5
Evaluated	6
Time (s) (avg on 10 runs)	<b>0.48216</b>
Initial h	3

## $h_{\text{max}}$

ADMISSIBLE



<input checked="" type="checkbox"/> Status	Solved (Opt)
Plan Cost	4
Expanded	5
Evaluated	6
Time (s) (avg on 10 runs)	<b>0.39170</b>
Initial h	2

## OPTIMAL PLAN

Scan(A) → Isolate(A) → Patch(A) → Reconnect(A)

## INSIGHT

The  $h_{\text{add}}$  and  $h_{\text{max}}$  heuristics are more efficient than blind search, expanding 1 fewer node and leading immediately to the goal.

# Problem 2 - The Critical Hub (ADL)

Dependency Management and Conditional Effects

## STAR TOPOLOGY



## ADL SCENARIO & CHALLENGE

A **Central Database** serves two Web Servers. If the DB is isolated for maintenance or defense, all dependent nodes automatically become **service-degraded**.  
**Challenge:** The agent must understand that isolating the DB has an immediate collateral "cost" on the entire dependent network.

## </> PDDL: CONDITIONAL EFFECTS

```
(:action isolate
  :parameters (?n - node)
  :effect (and
    (isolated ?n) (not(connected ?n ?n)))
    ;Conditional Effect
    (forall (?dep - node)
      (when (depends-on ?dep ?n)
        (service-degraded ?dep)))))
```

### ADL FEATURE

**Forall Logic:** This construct checks all network nodes in a single step. If a dependency exists (`depends-on`), the state of the dependent node is altered.

## END GOAL

- ✓ Restore full operation (no degraded nodes).
- ✓ Ensure that the DB is clean and reconnected.

The planner must understand that reconnecting DB \*automatically\* restores dependent services via the ADL forall effect

# Problem 2 - Experimental Results

Critical Hub (ADL Scenario)

	
<b>Blind</b>	
UNINFORMED	
Status	<b>Solved</b>
Plan Cost	<b>4</b>
Expanded	<b>6</b>
Evaluated	<b>6</b>
Time (avg on 10 runs)	<b>0.40232</b>
Initial h	<b>1</b>

	
<b>h_add</b>	
ADDITIVE	
Status	<b>Solved</b>
Plan Cost	<b>4</b>
Expanded	<b>5</b>
Evaluated	<b>6</b>
Time (avg on 10 runs)	<b>0.41900</b>
Initial h	<b>3</b>

	
<b>h_max</b>	
ADMISSIBLE	
Status	<b>Solved</b>
Plan Cost	<b>4</b>
Expanded	<b>5</b>
Evaluated	<b>6</b>
Time (avg on 10 runs)	<b>0.40324</b>
Initial h	<b>2</b>

## Results Analysis

The planner correctly interpreted the ADL constraints: it inserted the reconnect(db) action as a mandatory final step. Without reconnection, the goal would not have been met due to conditional effects. H\_hadd provided the estimate closest to the actual cost (3 vs. 4)

PLAN SEQUENCE FOUND  
scan → isolate → patch → reconnect

# Problem 3 - Ransomware Crisis

Numeric Planning

## COST ANALYSIS AND COMPLEXITY



## COMPLEX SCENARIO

The system is facing a ransomware infection. Unlike worms, data is encrypted. Restoring from backups is safe but requires high bandwidth availability (20 BW), which may exceed network capacity if not managed.

PLANNER TARGET

**ENHSP-20 (numeric  
fluents and metrics)**

OBJECTIVE

**Minimize Overall Cost**

## OBJECTIVE FUNCTION (METRIC)

```
(:metric minimize (: (downtime-cost)) )
```

The planner focuses on minimizing downtime-cost. Bandwidth usage is tracked but does not influence the optimization score directly."

## NUMERICAL CONSTRAINTS

### ACTION COSTS (BANDWIDTH)

action: Patch	check <= MAX_BANDWIDTH	5 BW
---------------	------------------------	------

action: Restore	check <= MAX_BANDWIDTH	20 BW
-----------------	------------------------	-------

### PRIORITY TO KEEP THE COST LOW



# Problem 3: Optimization Logic

Balancing Resource Costs VS Asset Priority

## RESOURCE CONSTRAINTS (BANDWIDTH)

**Constraint Logic:** Bandwidth is modeled as a **Capacity Check**.

**Patch (5 BW):** Feasible if the capacity is enough

**Restore (20 BW):** Feasible only if sufficient capacity exists.  
Note: The planner filters out actions that exceed the (max-bandwidth) threshold.

## ASSET PRIORITY (DOWNTIME PENALTY)

- Defined in *instance3-ransomware-problem.pddl*:
- **CoreServer**: Weight 100 (Critical).
- **Workstations**: Weight 10 (Low Priority).
- The objective function minimize (downtime-cost) drives the decision. High-priority nodes (CoreServer=100) force the planner to find the quickest valid repair plan available within bandwidth limits

# Problem 3 - Experimental Results

"Ransomware Crisis"

## Blind Search

BASELINE-> STATUS: SOLVED



Cost

**110**

Plan Length

**8**

Expanded

**79**

Evaluated

**152**

Time (s) avg on 10 runs

**0.175**

Initial h

**0**

## h\_add

ADDITIVE-> STATUS: SOLVED



Cost

**110**

Plan Length

**8**

Expanded

**18**

Evaluated

**60**

Time (s) avg on 10 runs

**0.208**

Initial h

**210**

## h\_max

ADMISSIBLE -> STATUS: SOLVED



Cost

**110**

Plan Length

**9**

Expanded

**21**

Evaluated

**69**

Time (s) avg on 10 runs

**0.209**

Initial h

**100**

## OPTIMAL PLAN

Scan(Core)-> Isolate(Core)-> Patch(Core)-> Scan(WS1)->Isolate(WS1)  
->Restore(Core)->Patch(WS1)->Reconnect(Core)

## INSIGHT

All heuristics found the minimal cost (110), proving the planners correctly prioritized the CoreServer (Weight 100). h\_add was extremely efficient, expanding only 18 nodes compared to 79 for Blind search, demonstrating strong guidance in the numeric state space.

# IndiGolog - Intelligent Agent

Control Architecture & OODA Loop



## Active Sensing

The agent actively queries the simulation status to update its knowledge base.

```
execute(scan(N), Result)
```

## Exogenous Events

The environment is dynamic. Events such as **alert\_intrusion**, **service\_crash** and **network\_reset** occur asynchronously and require immediate handling

## Prioritized Concurrency

Use constructs to handle high-priority interruptions, suspending routine tasks for emergencies.

TECH STACK

SWI-Prolog

IndiGolog

# IndiGolog Domain Model

Topology, Fluents, and Causal Laws (domain.pl)

## NETWORK TOPOLOGY

-  **db\_server** Central hub (critical)
-  **web\_server\_1** Depends on db\_server
-  **web\_server\_2** Depends on db\_server
-  **Workstation\_a** Endpoint device

## SUBNETS & DEPENDENCIES

```
subnet(subnet_alpha). subnet(subnet_beta).
```

## PRIMITIVE FLUENTS

```
infected(N), clean(N), isolated(N)  
subnet_online(S), service_degraded(N)  
% Initial state  
initially(clean(N), true) :- node(N).  
initially(infected(N), false) :- node(N).
```

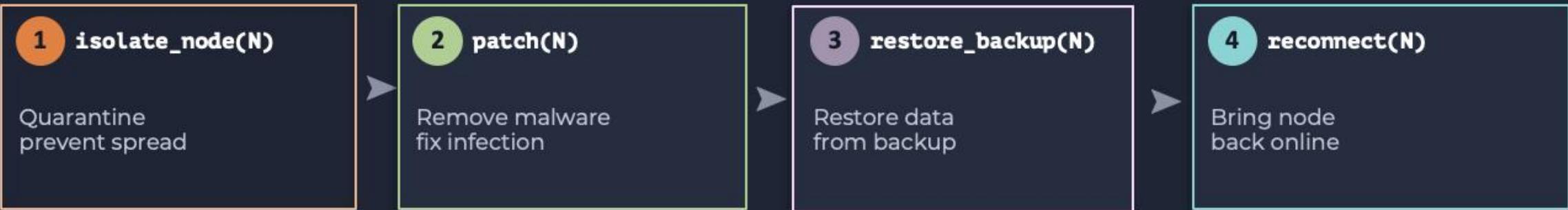
## CAUSAL LAWS

```
% ADL: isolating DB degrades dependents  
causes_val(isolate_node(DB),  
          service_degraded(WS), true, depends_on(WS,DB)).  
  
% Reconnecting restores services  
causes_val(reconnect(DB),  
  
% Exogenous effects service_degraded(WS), false, depends_on(WS,DB)).  
causes_val(alert_intrusion(N), infected(N), true, true).  
causes_val(service_crash(S), subnet_online(S), false, true).
```

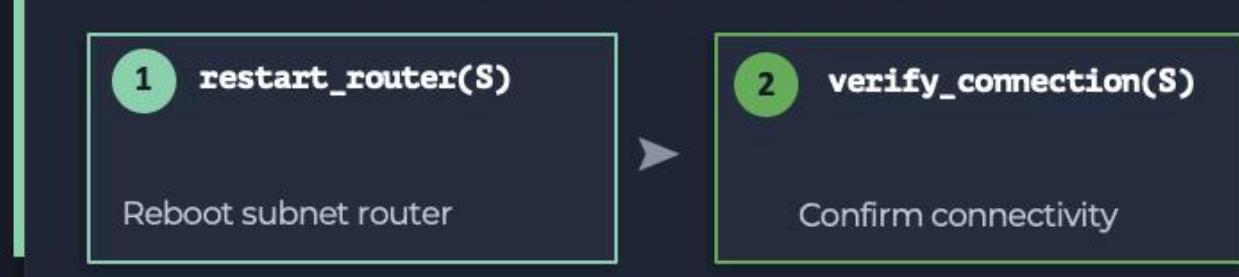
# Recovery Procedures

Building blocks used by all 3 controllers (agent\_logic.pl)

## NODE RECOVERY: recover\_node\_procedure(N)



## SUBNET RECOVERY: recover\_subnet\_procedure(S)



# Active Monitoring: monitor\_network

Scanning loop used by Passive and Reactive-Poll controllers

## PROLOG SOURCE

```
proc(monitor_network,  
    [  
        scan(db_server),  
        wait_step,  
        scan(web_server_1),  
        wait_step,  
        scan(web_server_2),  
        wait_step  
    ]  
).
```

## SENSING MECHANISM

Each `scan(N)` is a sensing action. The engine calls `execute(scan(N), Result)` which returns **true/false** for the fluent `infected(N)`.

## ROLE IN EACH CONTROLLER

**Passive:** Full cycle, THEN checks fluents.

**Reactive:** NOT used — waits for exogenous events.

**Reactive-Poll:** Checks fluents AFTER each scan.

# Controller 1: Passive

Scan ALL → Check ALL → React → Loop

## </> CODE STRUCTURE

```
proc(control(passive),  
while(true, [  
monitor_network, % Phase 1: scan all  
if(infected(db_server), % Phase 2: react  
recover_node_procedure(db_server), no_op),  
if(infected(web_server_1),  
recover_node_procedure(web_server_1), no_op),  
if(infected(web_server_2),  
recover_node_procedure(web_server_2), no_op),  
if(infected(workstation_a),  
recover_node_procedure(workstation_a), no_op),  
recover_all_subnets % Phase 3: subnets  
])  
).
```

## \* LOGIC BREAKDOWN

### Timeline Example

t=0 scan(db\_server) → clean  
t=1 scan(web\_server\_1) → clean  
⚡ INFECTION db\_server!  
t=2 scan(web\_server\_2) → clean ← continue  
t=3 infected(db\_server)? → TRUE!  
t=3 ↗ isolate → patch → restore\_backup →  
reconnect ↘  
t=7 ↳ RECOVERY COMPLETED  
—————  
t=8 ↴ loop  
...

### Limitations

Waits until the scan loop is finished

# Controller 2: Reactive

Event-driven — prioritized\_interrupts

## CODE STRUCTURE

```
proc(control(reactive), [prioritized_interrupts([  
  
    % Priority 1: infected node → recovery  
    interrupt(n, infected(n),  
              recover_node_procedure(n)),  
  
    % Priority 2: subnet down → recovery subnet  
    interrupt(neg(subnet_online(subnet_alpha))),  
              recover_subnet_procedure(subnet_alpha)),  
    interrupt(neg(subnet_online(subnet_beta))),  
              recover_subnet_procedure(subnet_beta)),  
  
    % Priority 3: wait  
    interrupt(true, ?(wait_exog_action))  
])]).
```

## LOGIC BREAKDOWN

### Timeline Example

t=0 → idle (wait\_exog\_action)  
⚡ INFECTION db\_server!  
t=1 Priority 1: infected(db\_server)? → TRUE!  
t=1 ↴ isolate → patch → restore\_backup →  
reconnect ↴  
t=5 ↴ RECOVERY COMPLETED

---

t=5 re-eval interrupt → everything ok → idle

### Limitations

Depends only on prioritized\_interrupts. We  
don't have any active monitoring.

### Advantages

No latency, the interrupt starts immediately

# Controller 3: Reactive Pool

Scan one → Check all → React → Next node → Repeat

## </> CODE STRUCTURE

```
proc(control(reactive_poll),  
     while(true,  
     [  
      scan(db_server),  
      recover_any_infected,  
      recover_all_subnets,  
      wait_step,  
      scan(web_server_1),  
      recover_any_infected,  
      recover_all_subnets,  
      wait_step,  
      scan(web_server_2),  
      recover_any_infected,  
      recover_all_subnets,  
      wait_step  
    ]  
  )  
).
```

## \* LOGIC BREAKDOWN

### Timeline Example

t=0 scan(db\_server) → clean  
t=1 check all → tutto ok  
⚡ INFECTION db\_server!  
t=2 scan(web\_server\_1) → clean  
t=3 check all: infected(db\_server)? → TRUE!  
t=3 ↴ isolate → patch → restore\_backup →  
reconnect ↴  
t=7 ↴ RECOVERY COMPLETED  

---

  
t=8 scan(web\_server\_2) → clean → check → ok  
t=9 ↵ loop

### Limitations

Overhead, latency of 2 steps

### Advantages

Doesn't require prioritized\_interrupts.

# Reasoning Task 1: Legality Check

PROJECTION QUERY

PROLOG

SITCALC

EXECUTION OUTPUT

```
% Precondition (domain.pl)
poss(restore_backup(N), and(isolated(N),
clean(N))) :- node(N).
```

```
% Initial state
initially(clean(N), true) :- node(N).
initially(isolated(N), false) :- node(N).
initially(infected(N), false) :- node(N).
...
```

==> TASK 1: LEGALITY ==>

Testing preconditions of restore\_backup

1a. restore\_backup(db\_server) without  
isolation... BLOCKED (correct: not  
isolated)

1b. restore\_backup(db\_server) when  
infected... BLOCKED (correct: not clean)

1c. restore\_backup(db\_server) when  
isolated+clean... ALLOWED (correct)  
...

# Reasoning Task 2: Temporal Projection

Prediction of Future State with Situation Calculus

## PROJECTION QUERY

PROLOG SITCALC

```
% Derived fluent (domain.pl)
service_available(Node, History) :-  
node(Node),  
holds(neg(isolated(Node)), History),  
holds(neg(service_degraded(Node)), History).  
% Test (reasoning_tests.pl)  
ActionList = [isolate_node(db_server)],  
holds(service_degraded(web_server_1), ActionList)  
% → TRUE (ADL conditional effect)  
holds(isolated(db_server), ActionList)  
% → TRUE  
\+ service_available(web_server_1, ActionList)  
% → FALSE (correct: dependency breaks service)
```

Logic: The test verifies that `service_degraded(web_server_1)` becomes true after `isolate_node(db_server)` thanks to the conditional effect in ``causes_val``

## EXECUTION OUTPUT

CONSOLE

```
After isolate_node(db_server):  
Query: service_available(web_server_1)?  
  
service_degraded(web_server_1)? TRUE  
(correct: ADL conditional effect)  
  
isolated(db_server)? TRUE  
  
service_available(web_server_1)? FALSE  
(correct: dependency breaks service)
```

# Reasoning Task 3: Planning

</> HIGH-LEVEL PROGRAM

PROLOG

INDIGOLOG

```
% Procedure (agent_logic.pl)
proc(recover_subnet_procedure(S),
[
  restart_router(S),
  verify_connection(S)
]
).

% Test (reasoning_tests.pl)
InitialHistory = [service_crash(subnet_alpha)],
findpath(recover_subnet_procedure(subnet_alpha),
InitialHistory, Path)
% → FOUND
% Plan: [restart_router(subnet_alpha),
verify_connection(subnet_alpha)]
...
```

> EXECUTION RESULT

SUCCESS

```
==> TASK 3: PLANNING ==
Scenario: subnet_alpha is offline
Goal: Find recovery plan
```

>> OUTPUT:  
> Plan Found: [restart\_router(subnet\_alpha),  
verify\_connection(subnet\_alpha)]

# Full Execution Trace

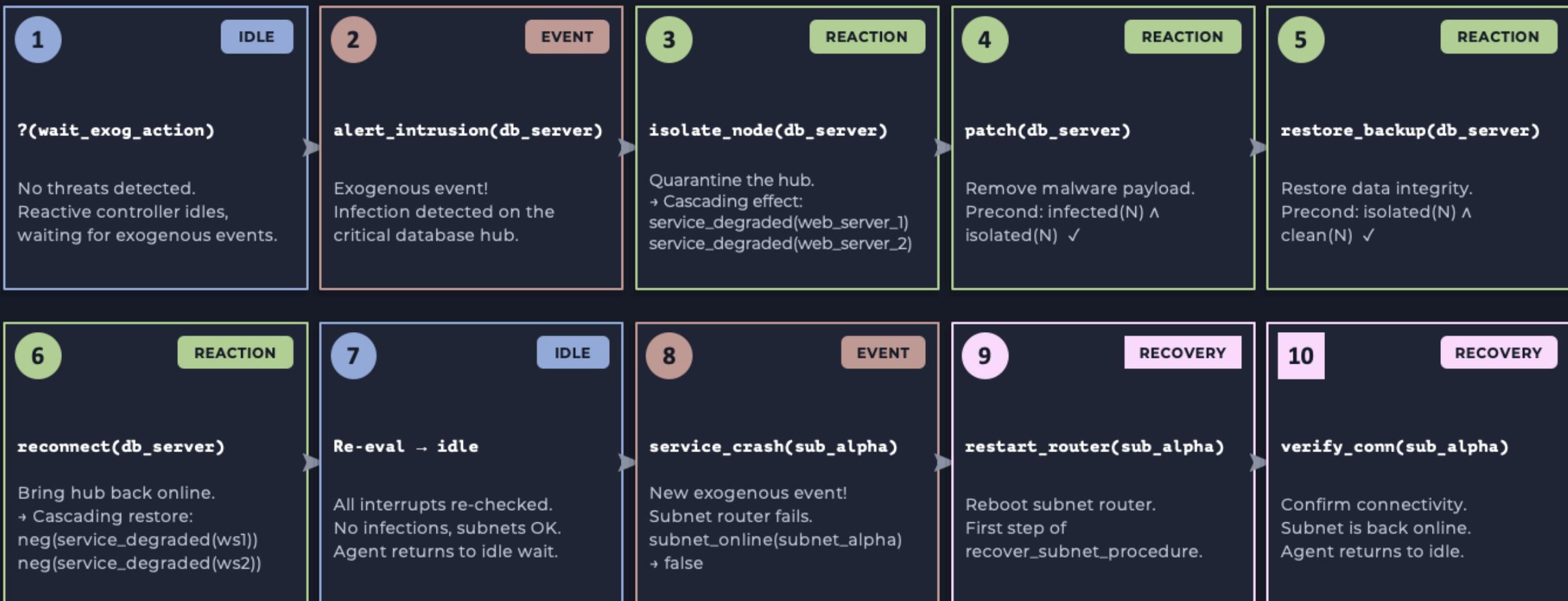
Reactive Controller Demonstration (IndiGolog)

Routine

Event

Recovery

Subnet



**Key:** 4-step node recovery (`isolate` → `patch` → `restore_backup` → `reconnect`) | ADL cascade on `isolate` & `reconnect` | Zero-latency: reactive fires immediately on event

# Reasoning Task 3: Planning

## Project Summary and Next Steps

### ACHIEVEMENTS

#### ✓ Robust Modeling

ADL conditional effects and numeric constraints across PDDL and IndiGolog domains.

#### ✓ Heuristic Optimization

Demonstrated h\_add/h\_max efficiency across 3 problem instances.

#### ✓ 3 Control Strategies

Passive, Reactive, Reactive-Poll with different latency/complexity trade-offs.

#### ✓ Complete OODA Loop

Working IndiGolog agent with sensing, interrupts, and dynamic plan execution.

### FUTURE WORK

#### ⚡ Real-World Integration

Connect with scanning tools (e.g., NMAP) to dynamically populate the KB.

#### ⚡ Probabilistic Scenarios

Stochastic Games to model action failures and intelligent adversaries.

#### ⚡ Enterprise Scalability

Testing on large-scale topologies with hundreds of heterogeneous nodes.

#### ⚡ Multi-Agent Coordination

Multiple IndiGolog agents across different network segments.

## Thanks for the attention