Certified Synthesis of High-Assurance Protocols and Monitors

Arthur Amorim, Max Taylor, Paul Gazzillo, and Lance Joneckis.

One Verified Model, Three Certified Artifacts

Cyber-Physical Systems Are Now Targets for Physical Damage via Cyber Attacks

Stuxnet

5 OCTOBER 1986 No 8,461 Price 50p

THE SUNDAY TIMES

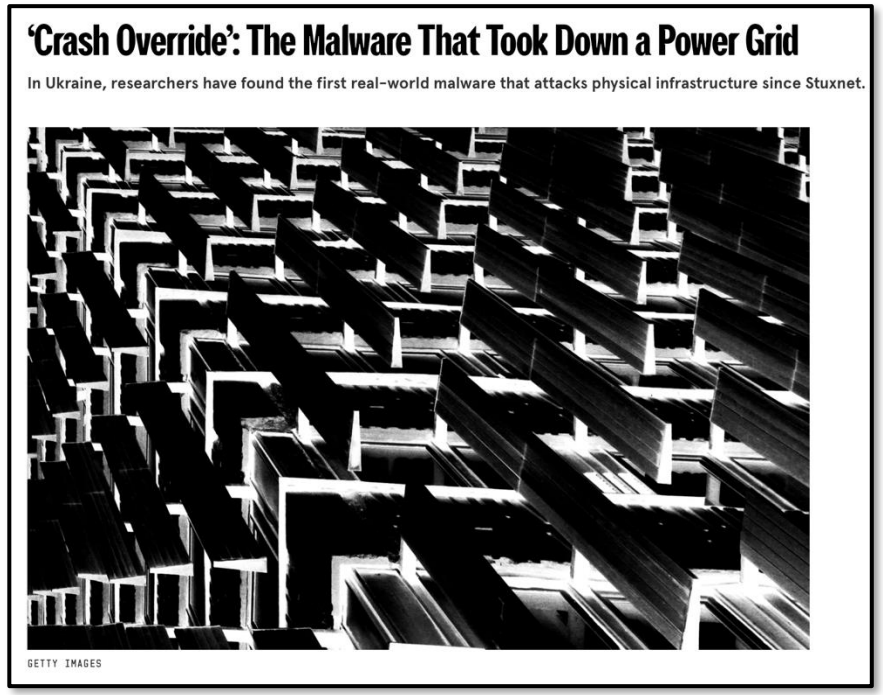
Revealed: the secrets of Israel's nuclear arsenal

INSIGHT

Pictured for the first time in detail: Israel's nuclear bomb plant is built deep beneath the Negev desert and has been secretly operating for more than 20 years

Inside Dimona: full story pages 2 & 3

Crash Override



'Crash Override': The Malware That Took Down a Power Grid

In Ukraine, researchers have found the first real-world malware that attacks physical infrastructure since Stuxnet.

GETTY IMAGES

German Steel Mill

TECH + ENGINEERING

Cyber Attack on German Steel Mill Leads to 'Massive' Real World Damage

A steel mill in Germany lost control of its blast furnace. Hackers had infiltrated the mill's control system, according to the German government's office for information security.

BY R.A. BECKER THURSDAY, JANUARY 8, 2015 NOVA NEXT

Idaho National Laboratory

Stealthy Attacks Use Only Valid Commands in the Wrong Order

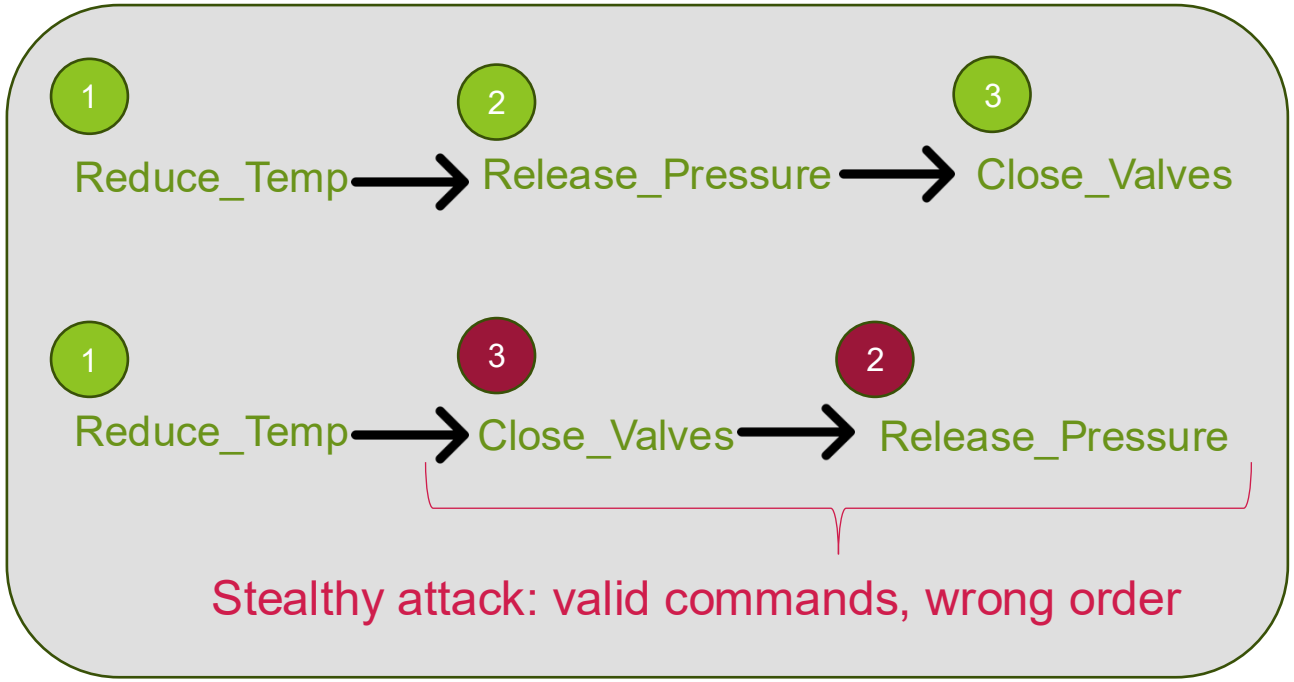
TECH + ENGINEERING

Cyber Attack on German Steel Mill Leads to 'Massive' Real World Damage

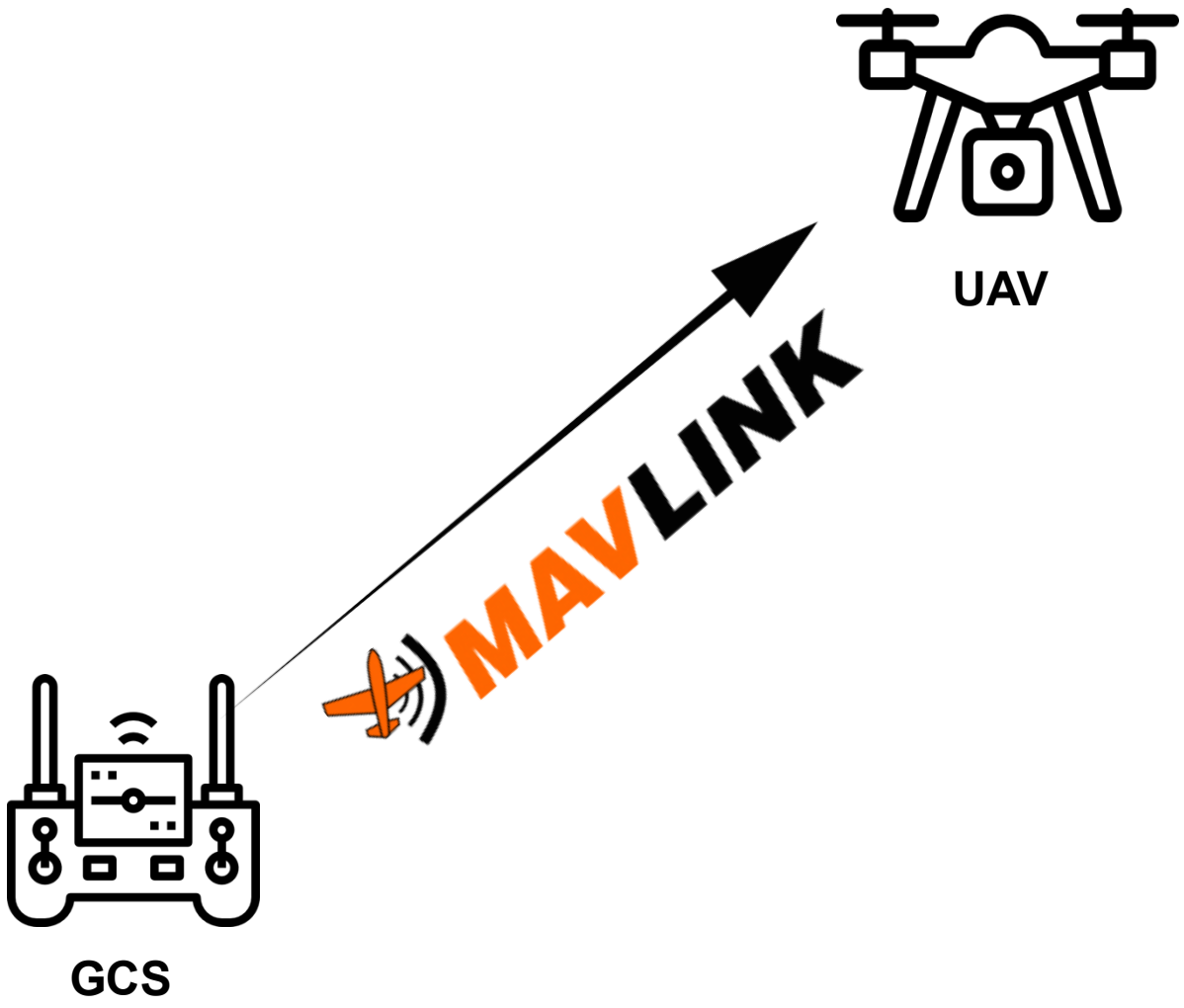
A steel mill in Germany lost control of its blast furnace. Hackers had infiltrated the mill's control system, according to the German government's office for information security.

BY R.A. BECKER THURSDAY, JANUARY 8, 2015 NOVA NEXT

Blasting Furnace Shutdown Sequence:



Widely Used Protocols Are Susceptible to Stealthy Attacks



RVFUZZER: Finding Input Validation Bugs in Robotic Vehicles Through Control-Guided Testing

Taegyu Kim†, Chung Hwan Kim*, Junghwan Rhee*, Fan Fei†, Zhan Tu†, Gregory Walkup†
Xiangyu Zhang†, Xinyan Deng†, Dongyan Xu†
†*Purdue University, {tgkim, feif, tu17, gwalkup, xyzhang, xdeng, dxu}@purdue.edu*
**NEC Laboratories America, {chungkim, rhee}@nec-labs.com*

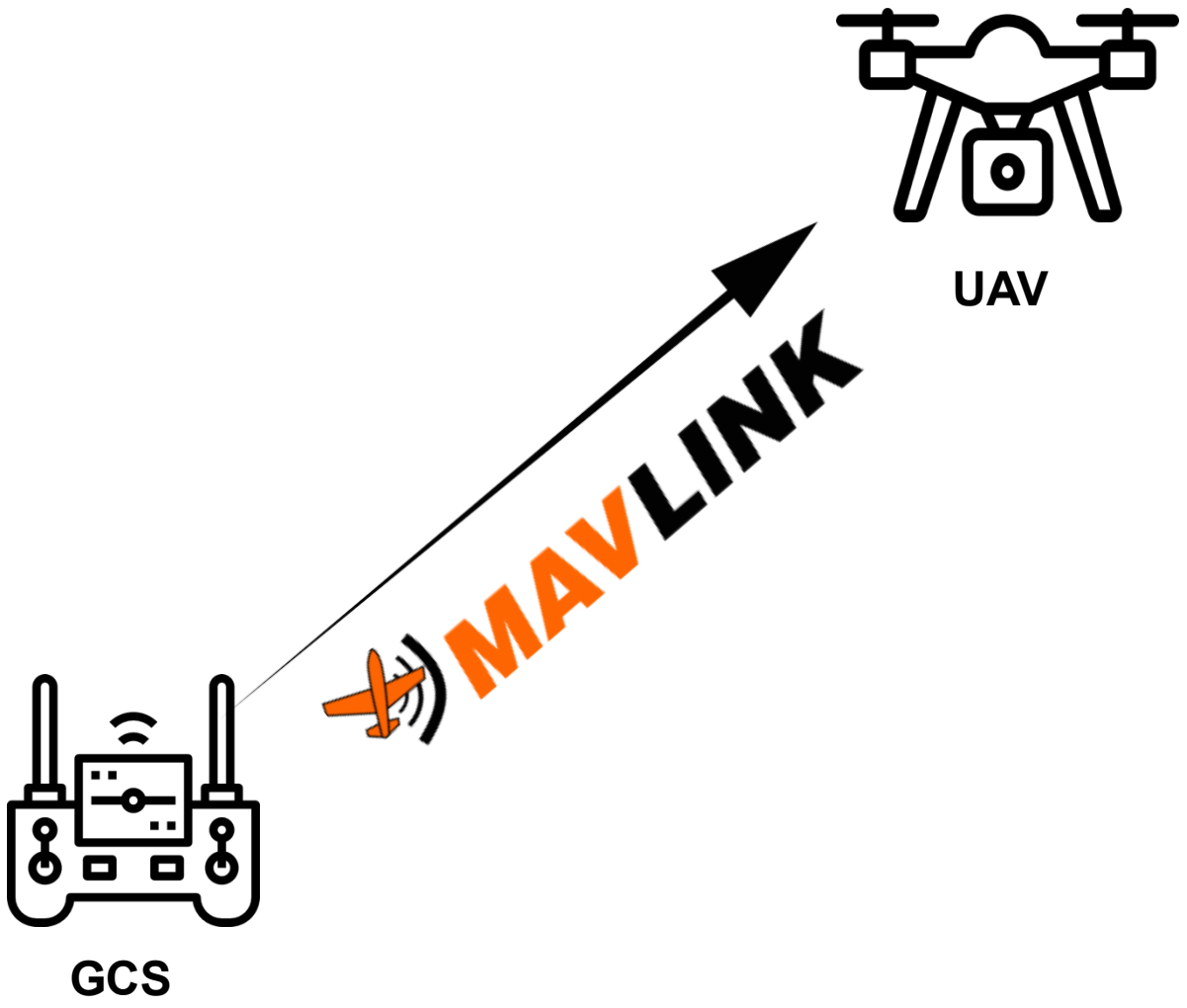
Found 89 input validation bugs

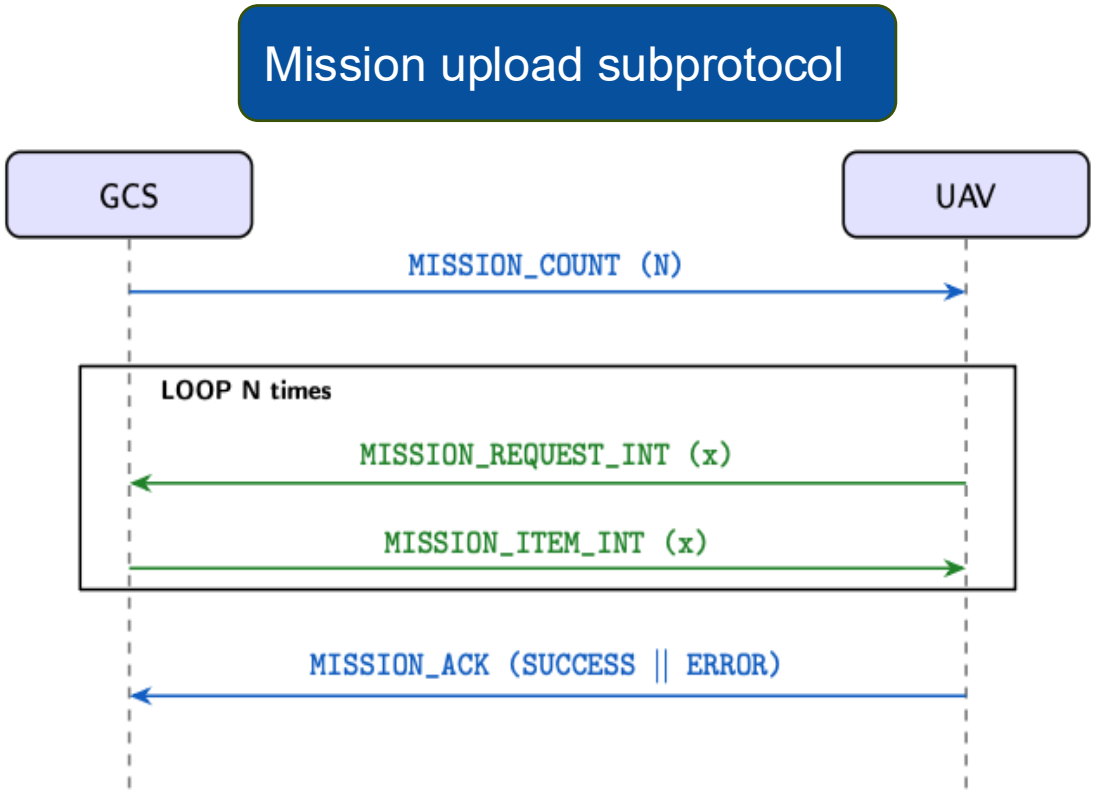
PGFUZZ: Policy-Guided Fuzzing for Robotic Vehicles

Hyungsub Kim, Muslum Ozgur Ozmen, Antonio Bianchi, Z. Berkay Celik, and Dongyan Xu
Purdue University
{kim2956, mozmen, antoniob, zcelik, dxu}@purdue.edu

Found 156 policy violation bugs

Widely Used Protocols Are Susceptible to Stealthy Attacks

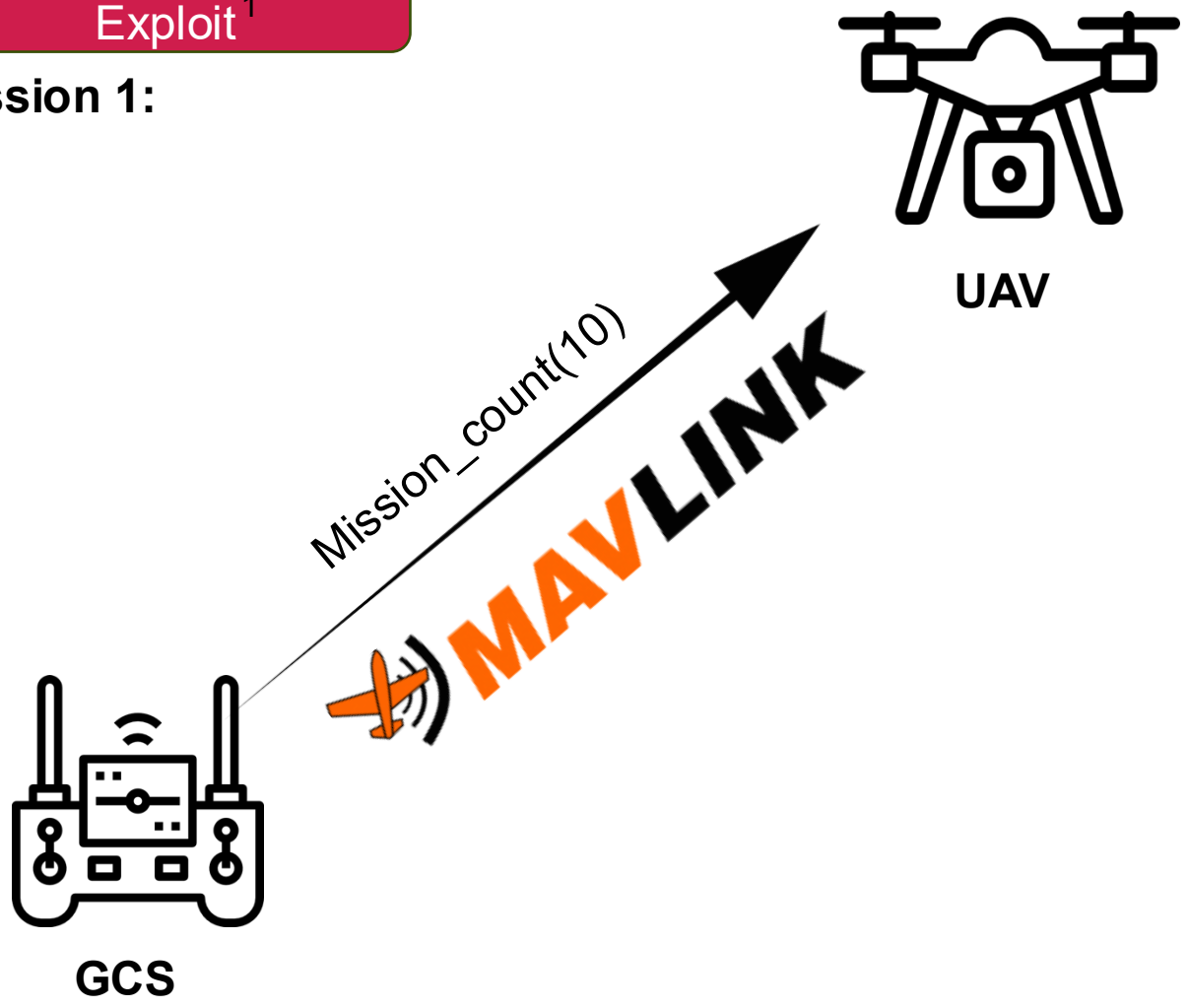




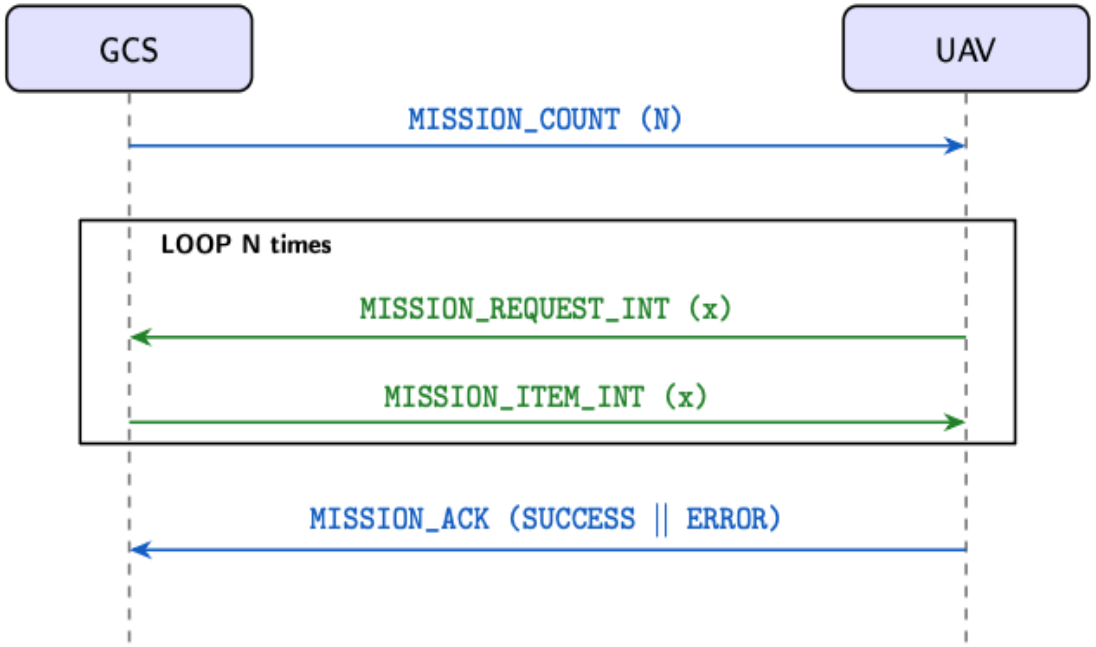
Widely Used Protocols Are Susceptible to Stealthy Attacks

Resource Usage Exploit[1]

Mission 1:



Mission upload subprotocol

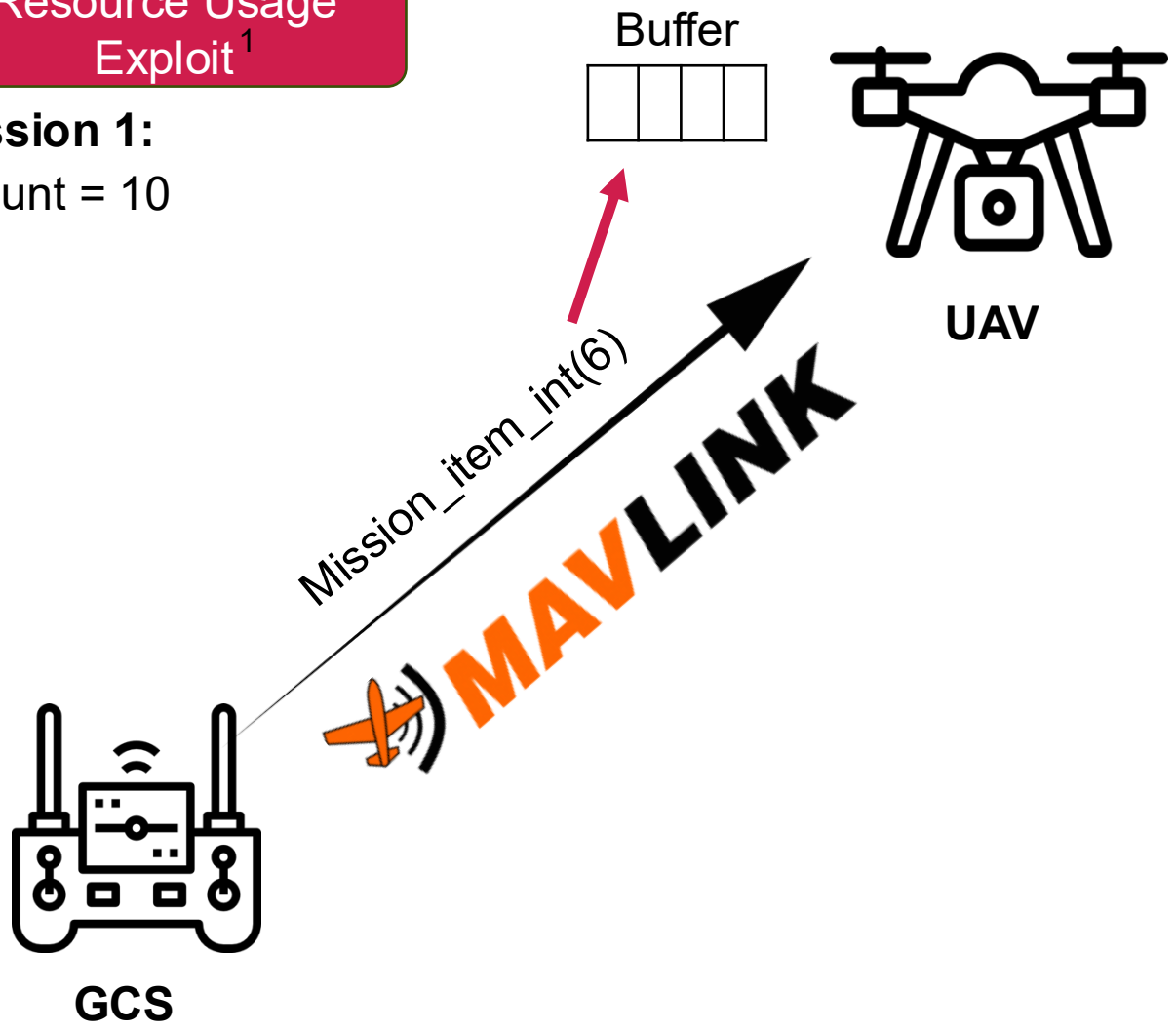


1- https://github.com/ArduPilot/MissionPlanner/issues/1248

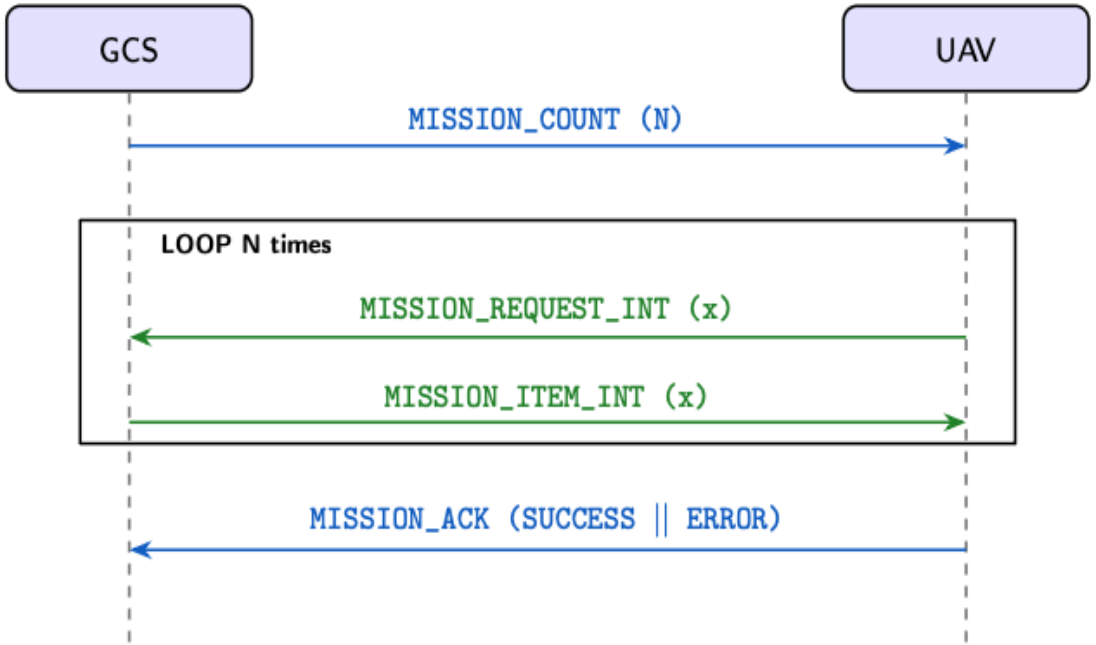
Widely Used Protocols Are Susceptible to Stealthy Attacks

Resource Usage Exploit[1]

Mission 1:
Count = 10



Mission upload subprotocol



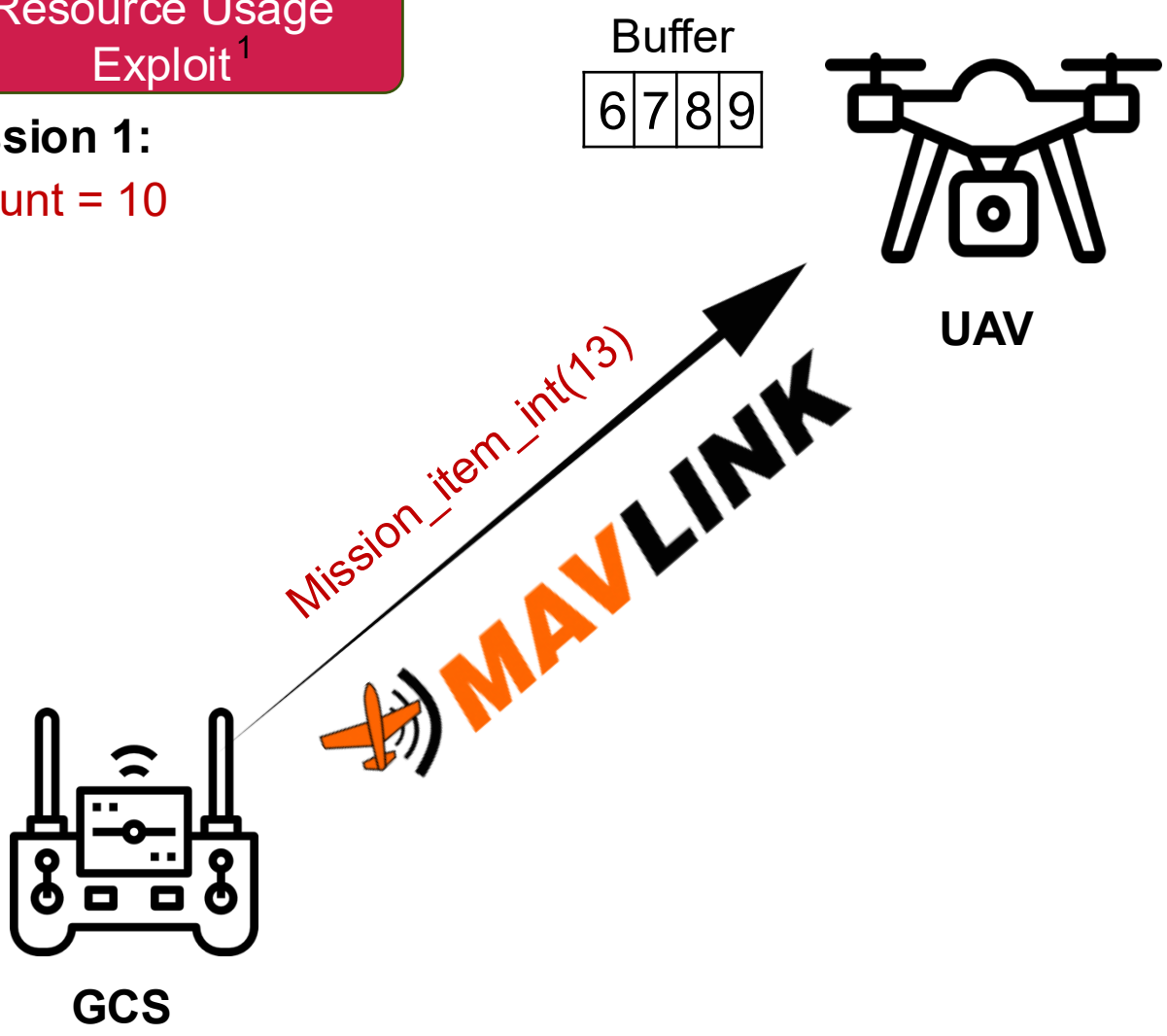
1- https://github.com/ArduPilot/MissionPlanner/issues/1248

Widely Used Protocols Are Susceptible to Stealthy Attacks

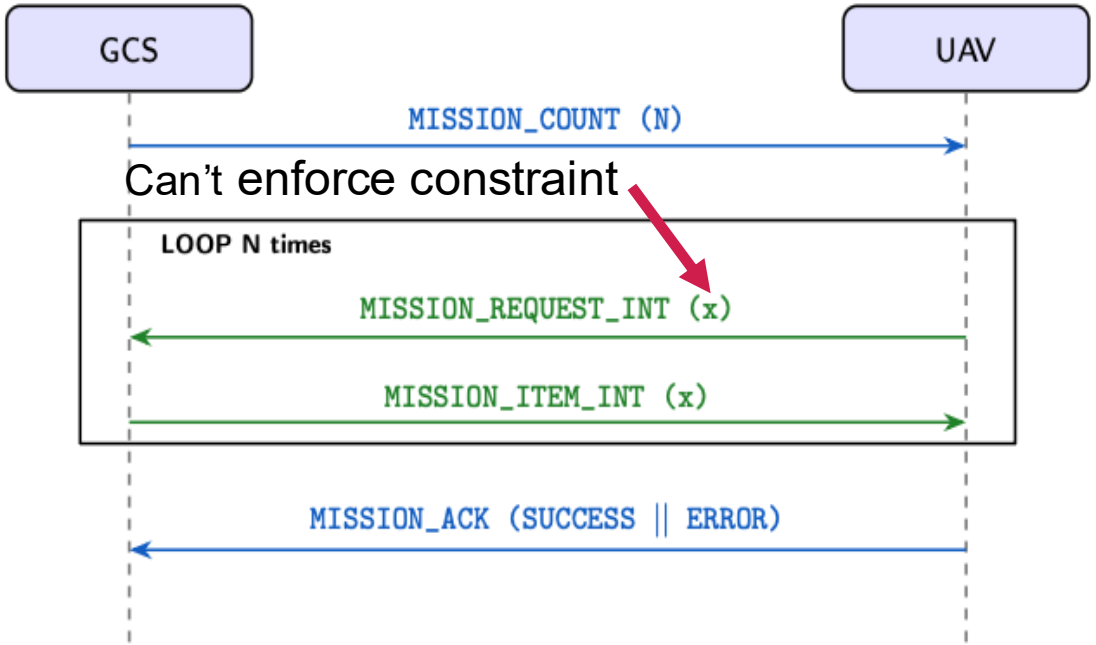
Resource Usage Exploit[1]

Mission 1:

Count = 10



Mission upload subprotocol



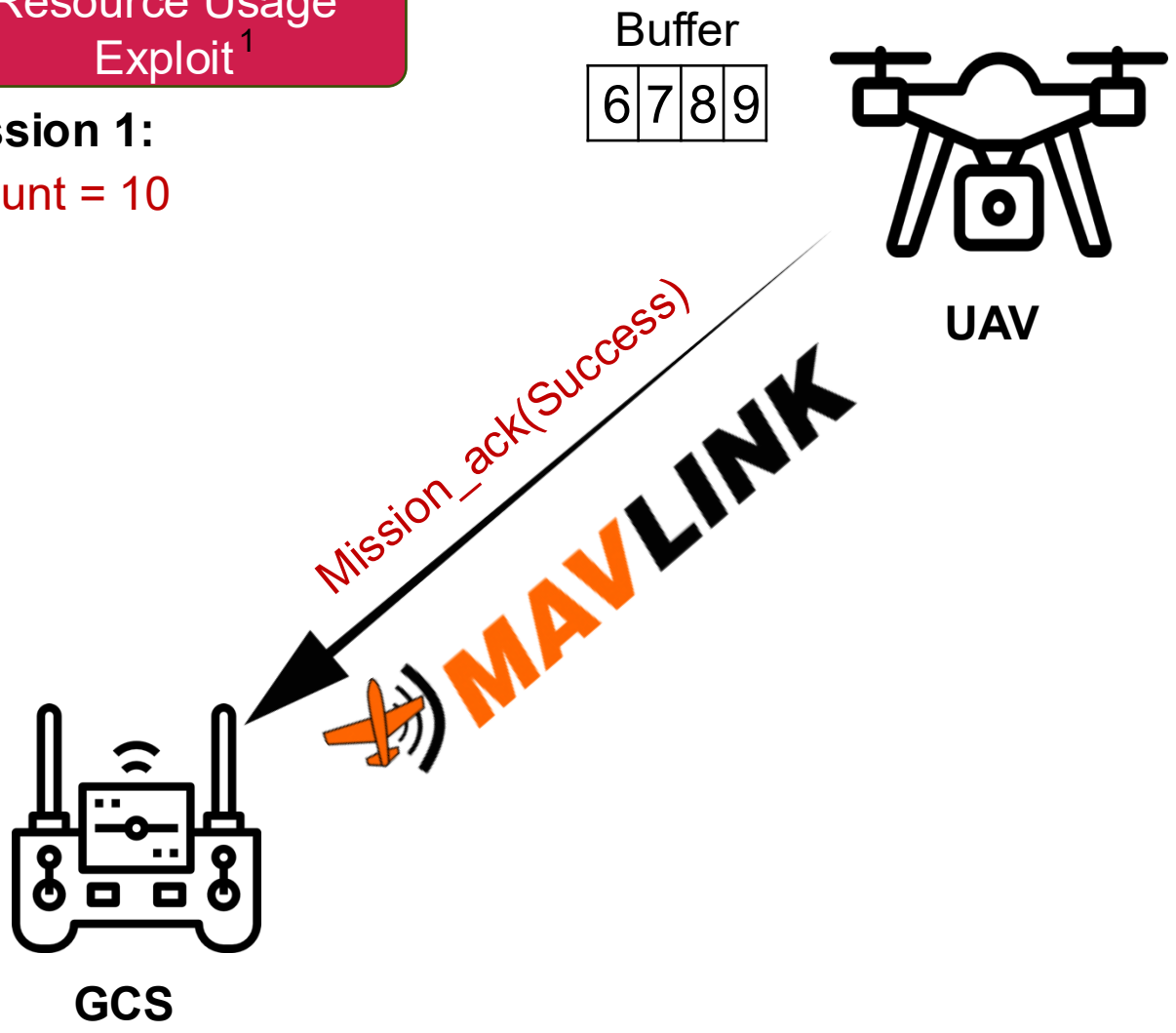
1- https://github.com/ArduPilot/MissionPlanner/issues/1248

Widely Used Protocols Are Susceptible to Stealthy Attacks

Resource Usage Exploit[1]

Mission 1:

Count = 10



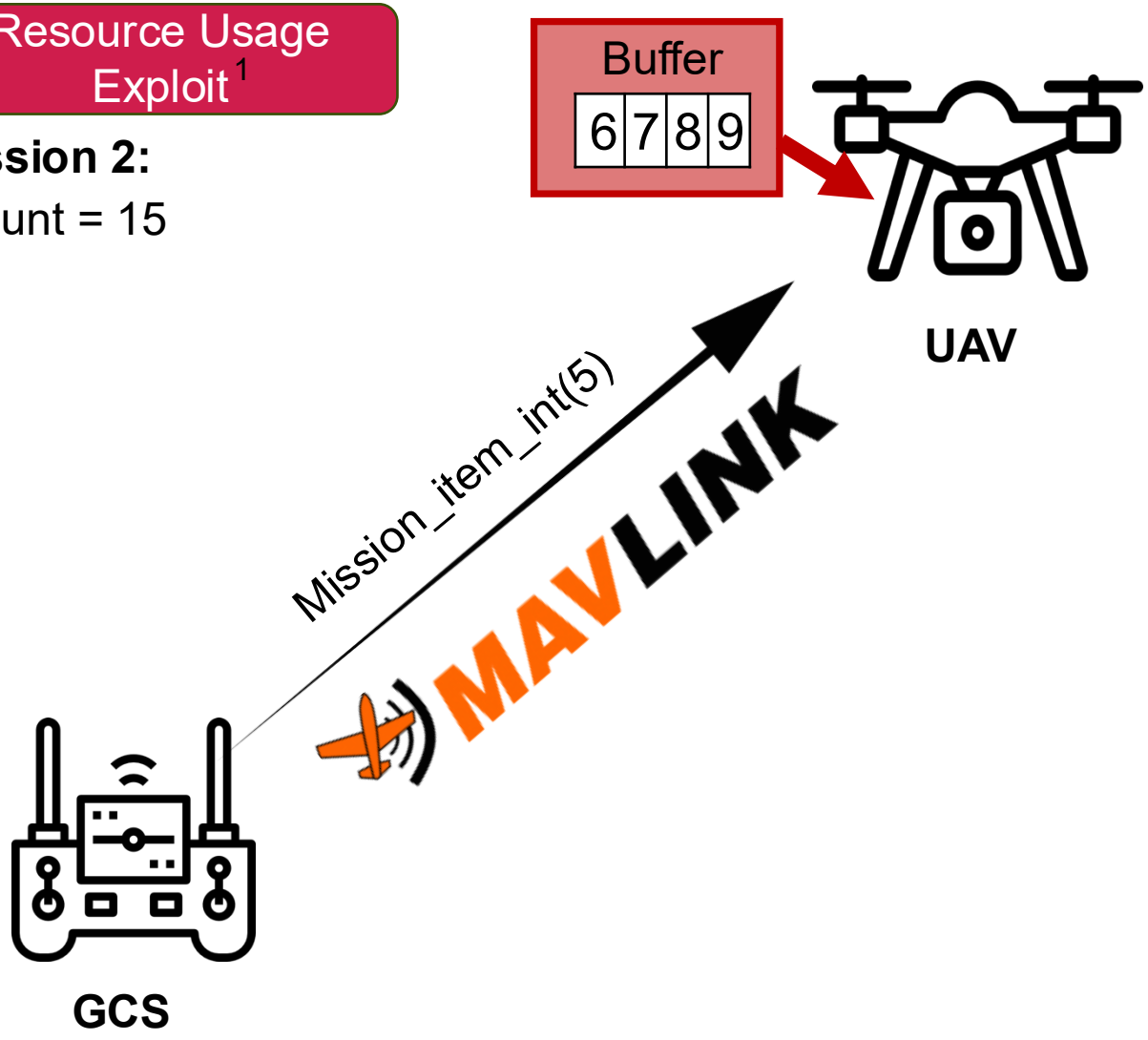
1- https://github.com/ArduPilot/MissionPlanner/issues/1248

Widely Used Protocols Are Susceptible to Stealthy Attacks

Resource Usage Exploit[1]

Mission 2:

Count = 15



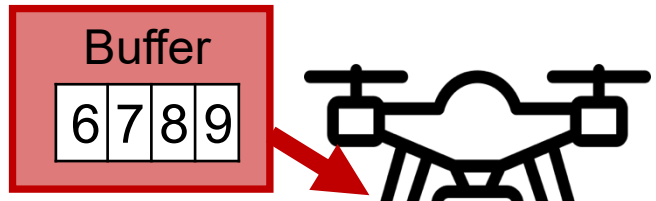
1- https://github.com/ArduPilot/MissionPlanner/issues/1248

Widely Used Protocols Are Susceptible to Stealthy Attacks

Resource Usage Exploit[1]

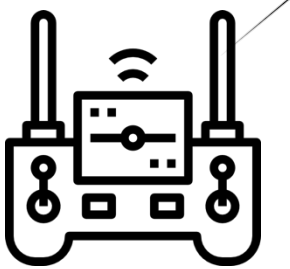
Mission 2:

Count = 15



UAV

UAV starts behaving as mission 1 in the middle of mission 2



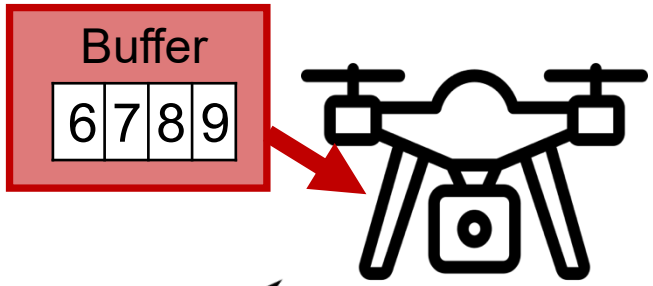
GCS

1- https://github.com/ArduPilot/MissionPlanner/issues/1248

Existing Defenses Miss the Logical Layer

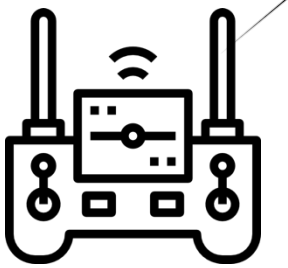
Resource Usage Exploit[1]

Mission 2:
Count = 15



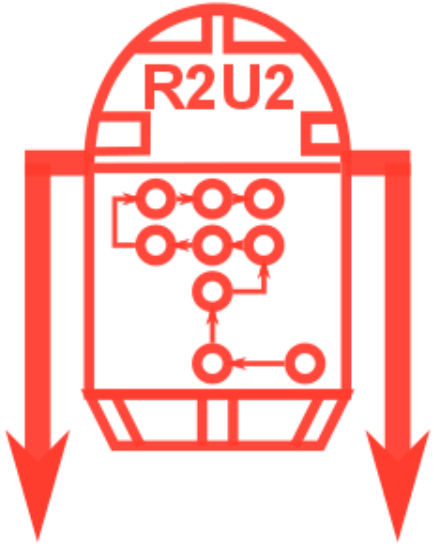
UAV

UAV starts behaving as mission 1 in the middle of mission 2



GCS

Problem: A stealthy attack are invisible to approaches that focus on catching bad behavior at runtime



Intrusion detection systems: No anomalous behavior or virus signature

Physics based anomaly detection: no anomalous physical deviation.

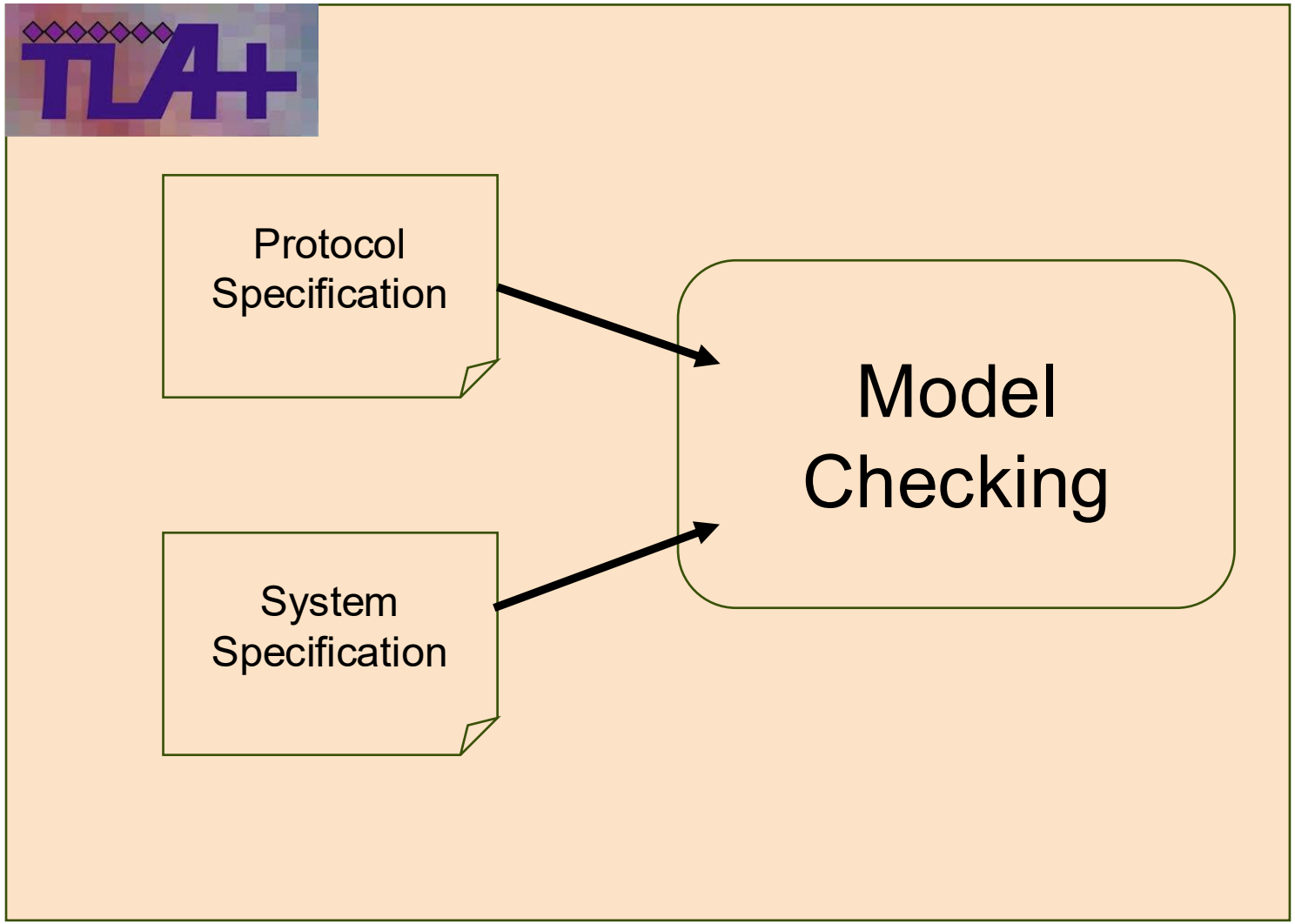
1- https://github.com/ArduPilot/MissionPlanner/issues/1248

Limitations of Formal Tools

Formal methods prevents unsafe sequences from being represented

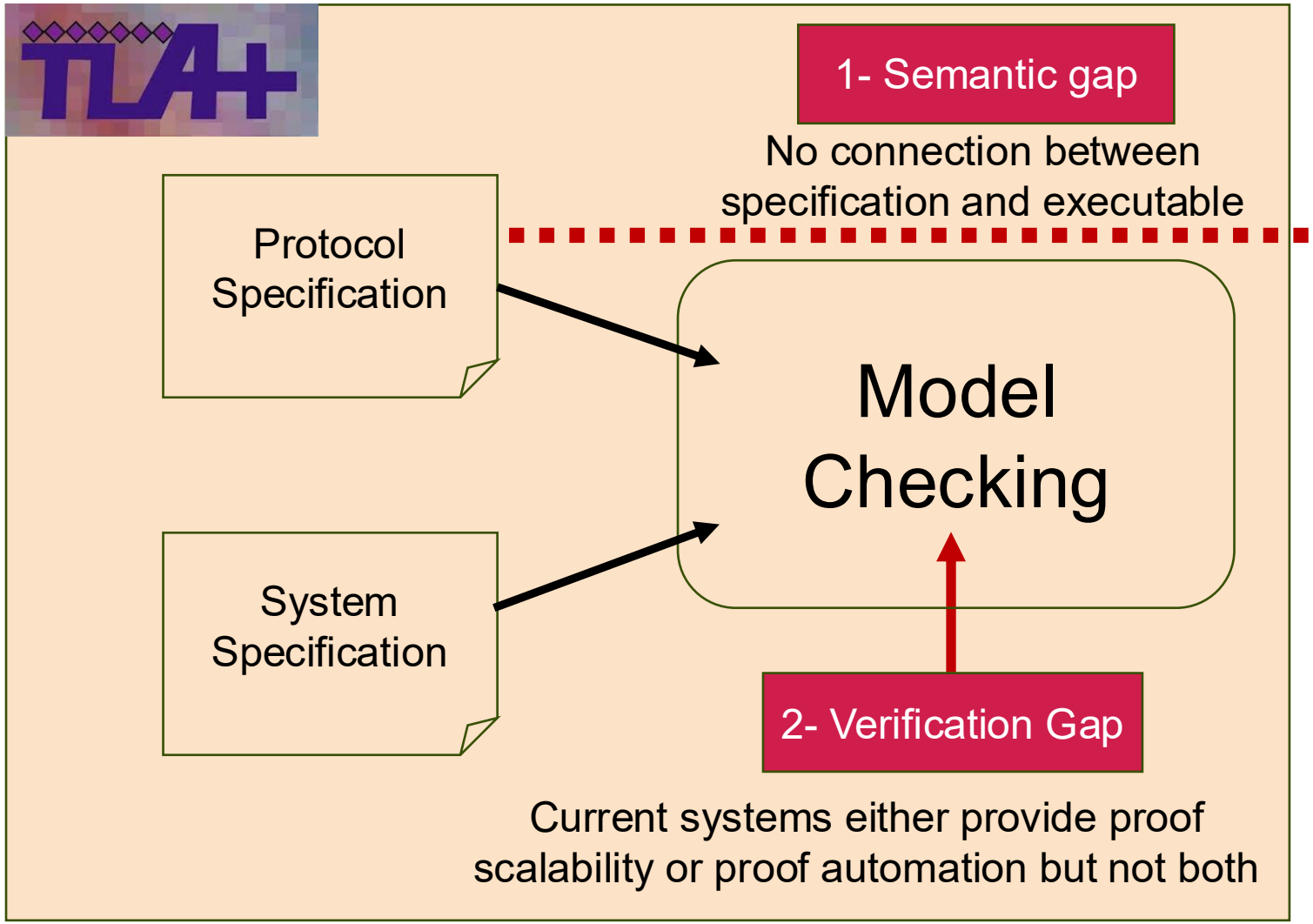
Limitations of Formal Tools

Formal methods prevents unsafe sequences from being represented



Limitations of Formal Tools

Formal methods prevents unsafe sequences from being represented

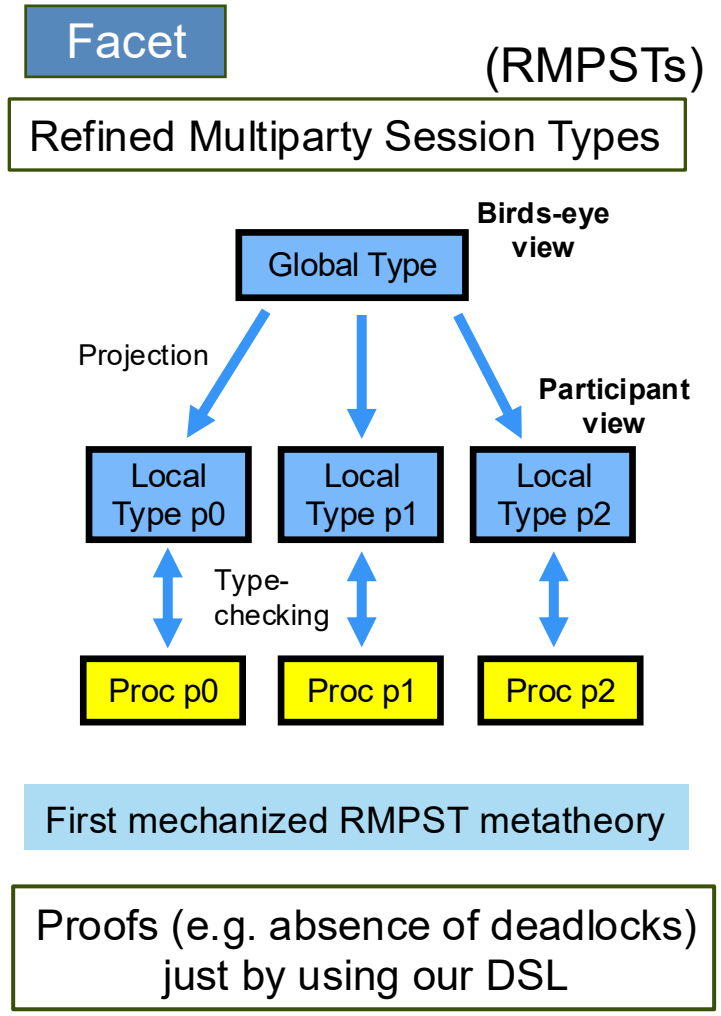


MAVLINK
MICRO AIR VEHICLE COMMUNICATION PROTOCOL

INL
Idaho National Laboratory

Contributions

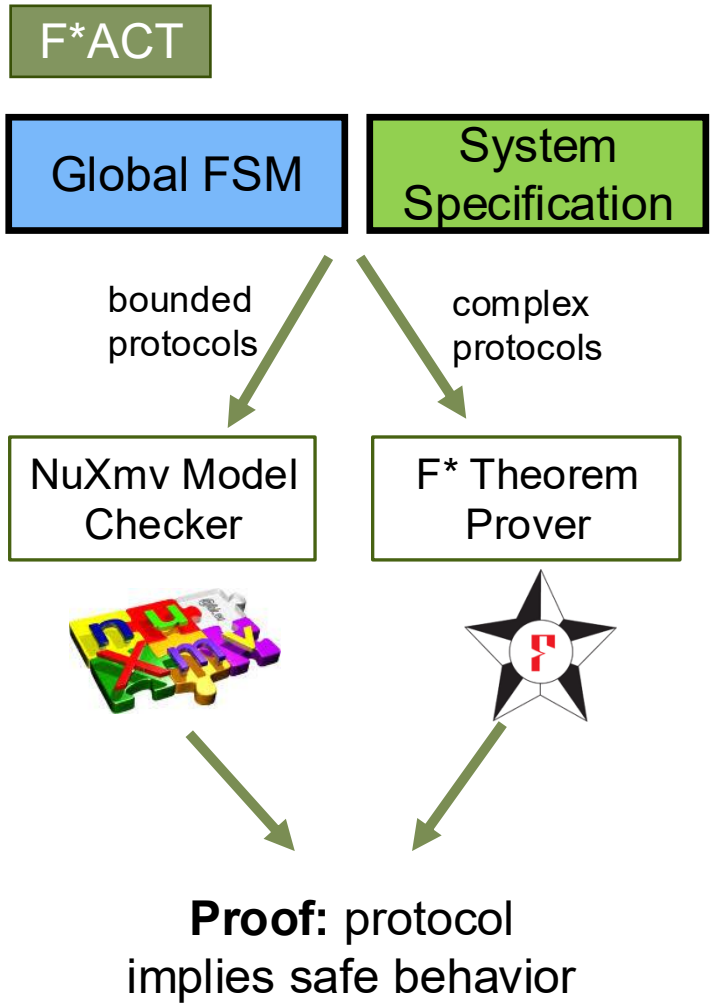
1. End-to-end formally verified pipeline from protocol specification to executable code



Contributions

1. End-to-end formally verified pipeline from protocol specification to executable code

2. Dual verification path providing either automatic model checking or theorem-prover pathways.



Contributions

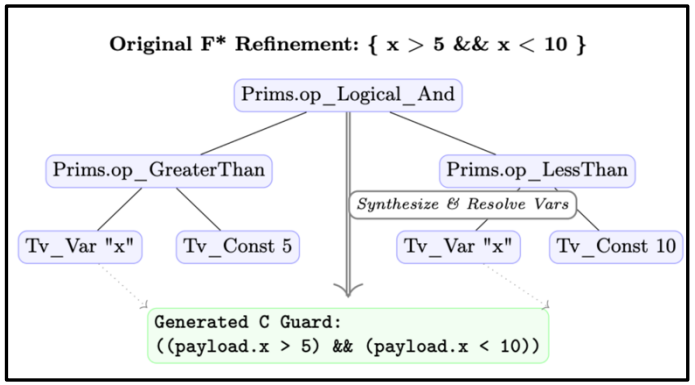
1. End-to-end formally verified pipeline from protocol specification to executable code

2. Dual verification path providing either automatic model checking or theorem-prover pathways.

3. Runtime monitors that retrofit legacy protocols with the same level of assurance.

Platum

retrofits existing protocols



Memory overhead: 8.39Mb
Latency overhead: 13.33µs

Contributions

1. End-to-end formally verified pipeline from protocol specification to executable code

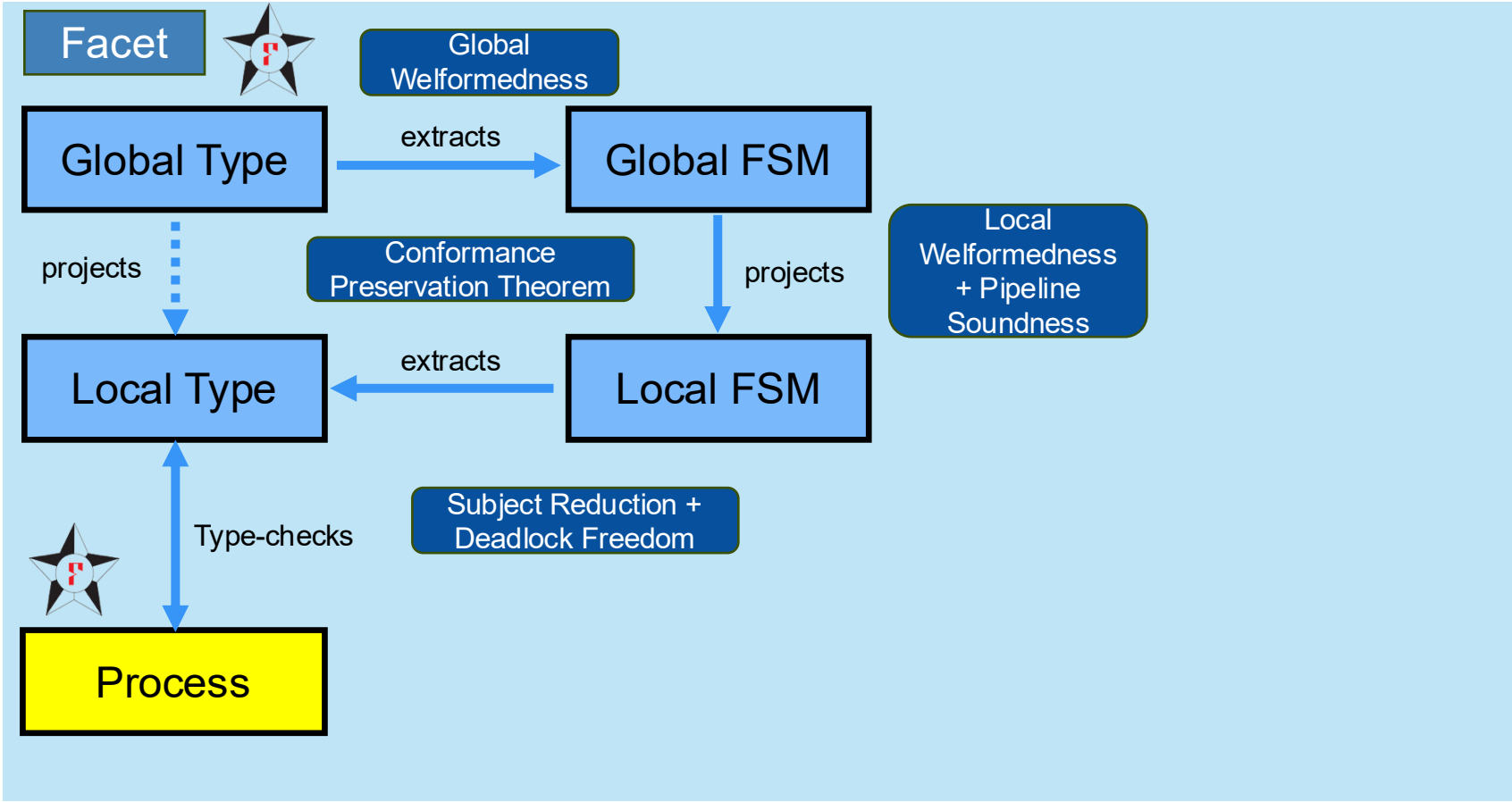
2. Dual verification path providing either automatic model checking or theorem-prover pathways.

3. Runtime monitors that retrofit legacy protocols with the same level of assurance.

4. Single certified semantic model shared across all generated deployment artifacts

Facet: Mechanized RMPST metatheory in F*

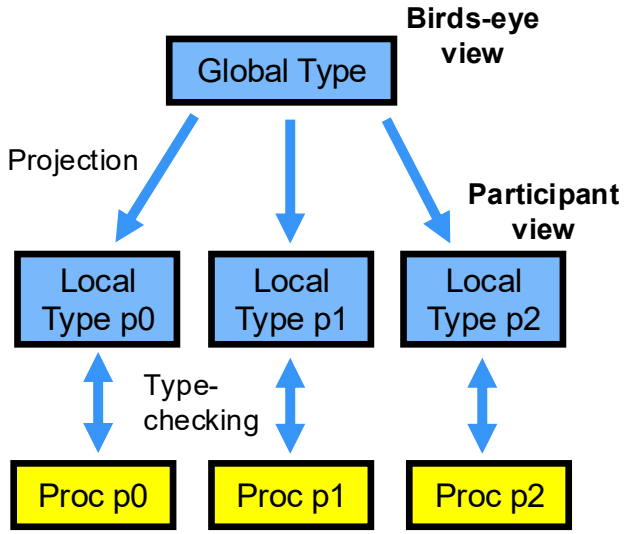
1- Semantic Gap



Facet

(RMPSTs)

Refined Multiparty Session Types

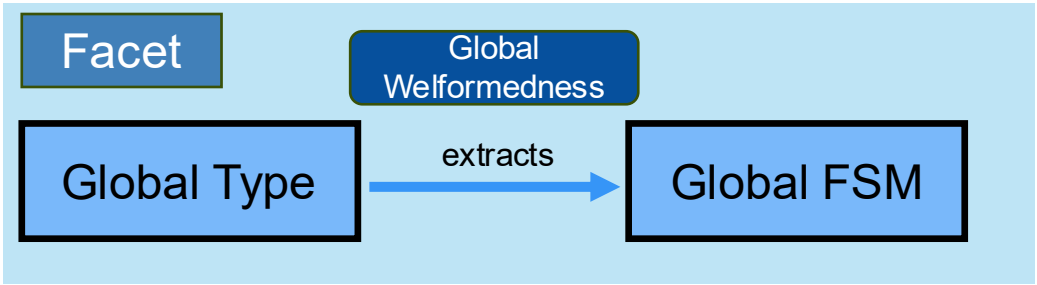


First mechanized RMPST metatheory

Proofs (e.g. absence of deadlocks) just by using our DSL

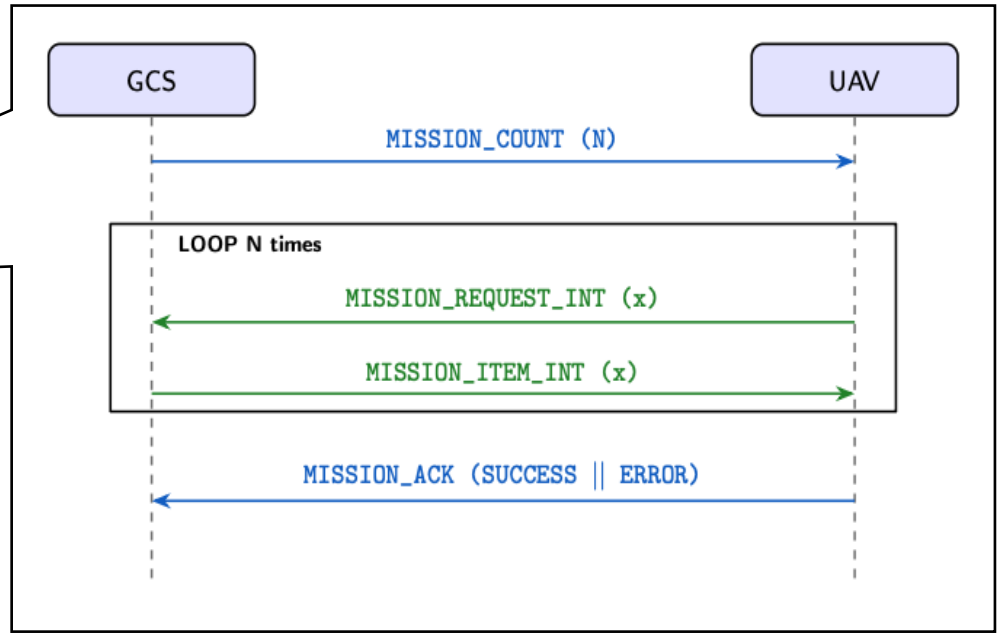
1- Amorim et al. Facet: Mechanized Theory of Refined Multiparty Protocols via Computational Reflection. Under Review

Defining an RMPST for the Mission Upload Protocol.

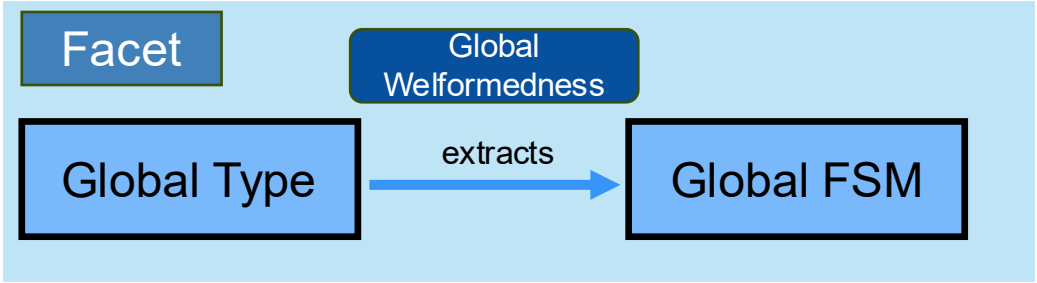


RMPST

$$
\begin{aligned}
&\mathbf{GCS} \to \mathbf{UAV}: \texttt{MISSION\_COUNT}(n\{1 \leq n < \text{LIMIT}\}) \\
&\mu.T(curr\{0 \leq curr < n\})(curr = 0) \\
&\quad \mathbf{UAV} \to \mathbf{GCS}: \\
&\qquad \texttt{MISSION\_REQUEST\_INT}(req\{req = curr < n\}) \\
&\qquad \texttt{MISSION\_ITEM\_INT}(item\{item = curr < n\}) \\
&\qquad T(curr + 1) \\
&\quad \oplus \texttt{MISSION\_ACK}(ack\{\text{type} = \text{ERROR} \vee curr = n\}).\mathbf{End}
\end{aligned}
$$



Defining an RMPST for the Mission Upload Protocol.



RMPST

$$\mathbf{GCS} \to \mathbf{UAV}: \texttt{MISSION COUNT}(n\{1 \leq n < \text{LIMIT}\})$$
$$\mu.T(curr\{0 \leq curr < n\})(curr = 0) \quad (1)$$
$$\mathbf{UAV} \to \mathbf{GCS}:$$
$$\texttt{MISSION REQUEST INT}(req\{req = curr < n\})$$
$$\texttt{MISSION ITEM INT}(item\{item = curr < n\}) \quad (2)$$
$$T(curr + 1) \quad (3)$$
$$\oplus\, \texttt{MISSION ACK}(ack\{\text{type} = \text{ERROR} \vee curr = n\}).\mathbf{End}$$

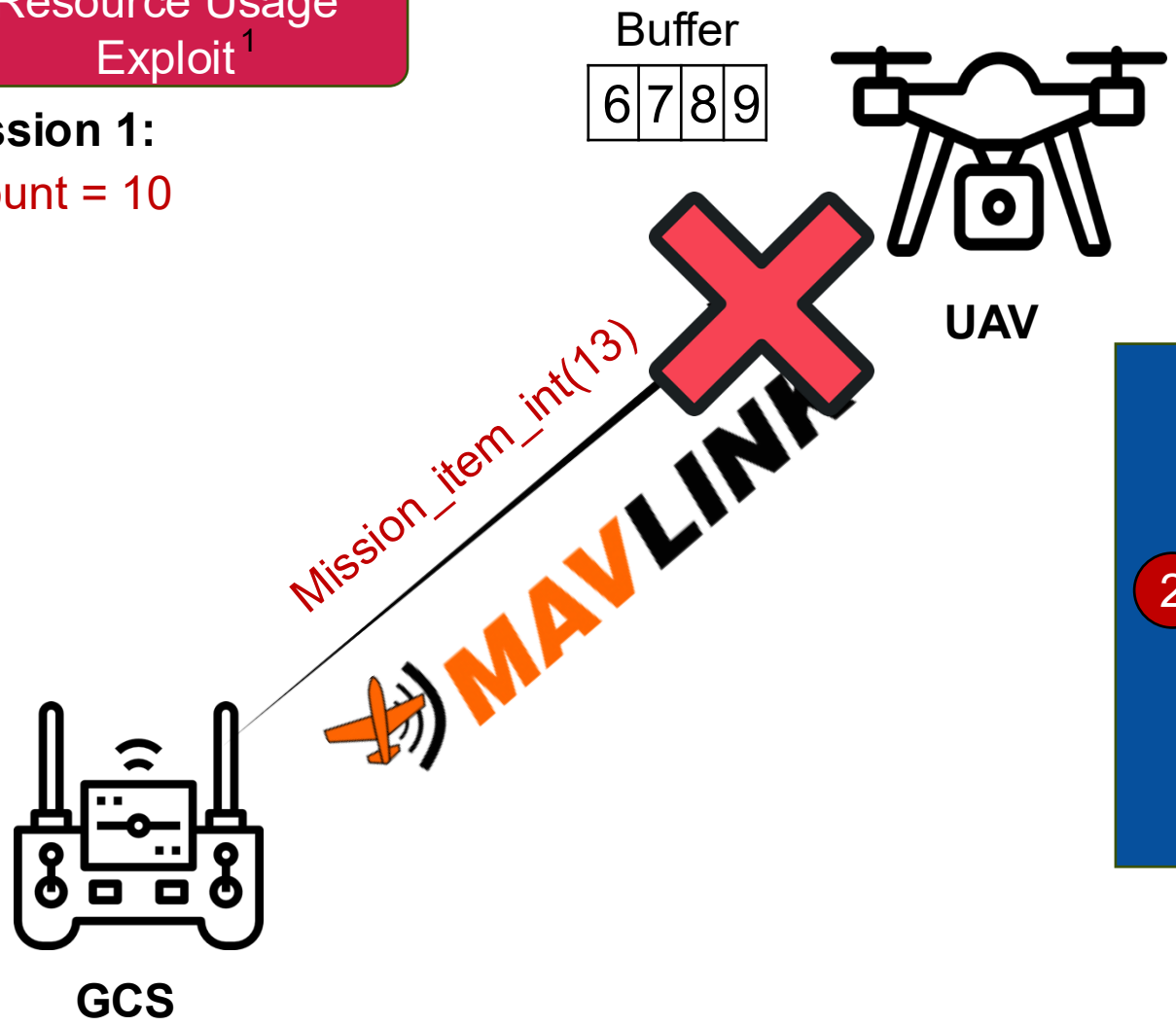
1. Create a recursive variable *curr* that must be less than or equal to n
2. Ensure messages are uploaded in the correct order and not skipped
3. The protocol must only complete with *curr* = *n* or with an error message.

Defining an RMPST for the Mission Upload Protocol.

Resource Usage Exploit[1]

Mission 1:

Count = 10



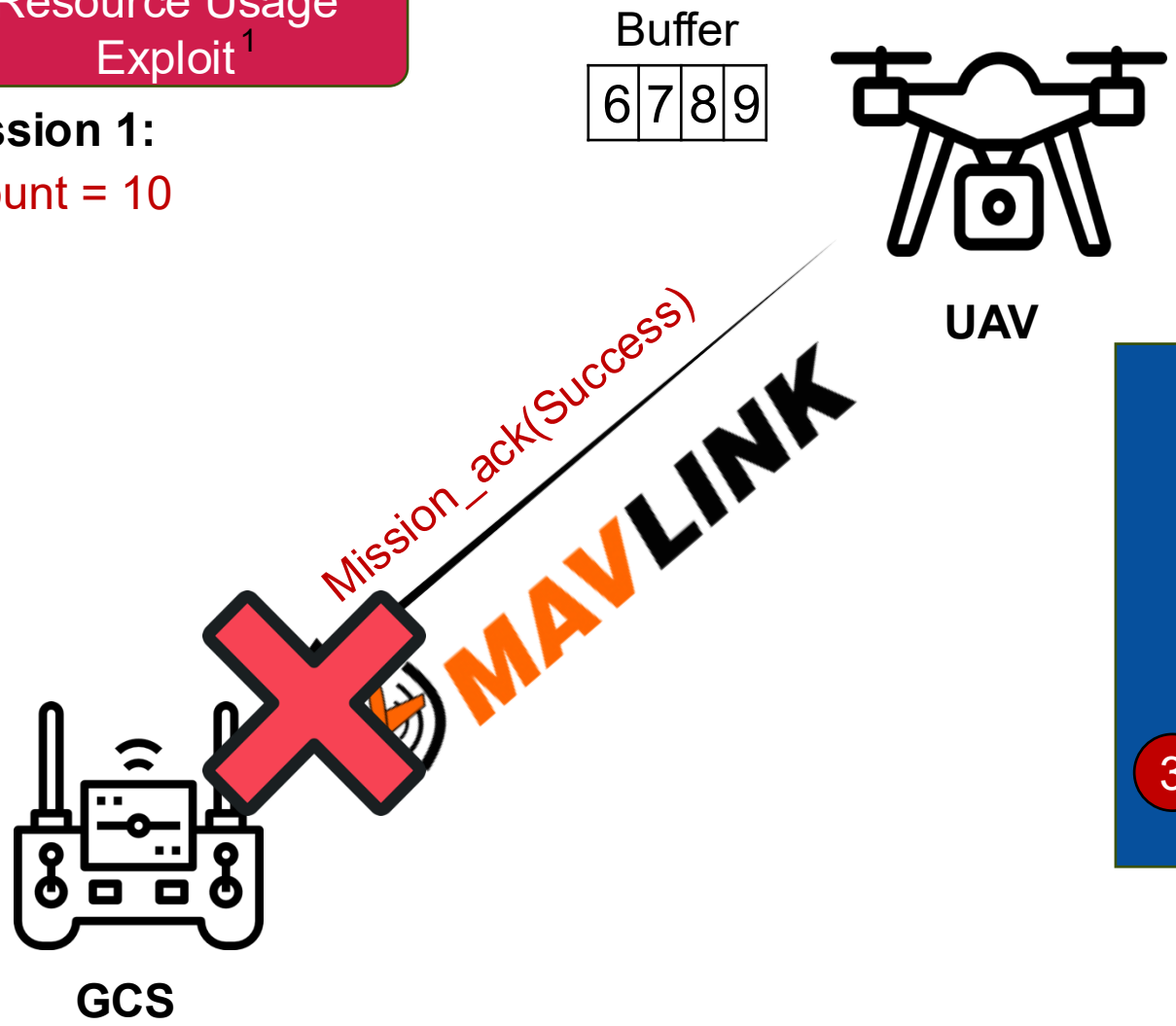
2 Ensure messages are uploaded in the correct order and not skipped

Defining an RMPST for the Mission Upload Protocol.

Resource Usage Exploit[1]

Mission 1:

Count = 10

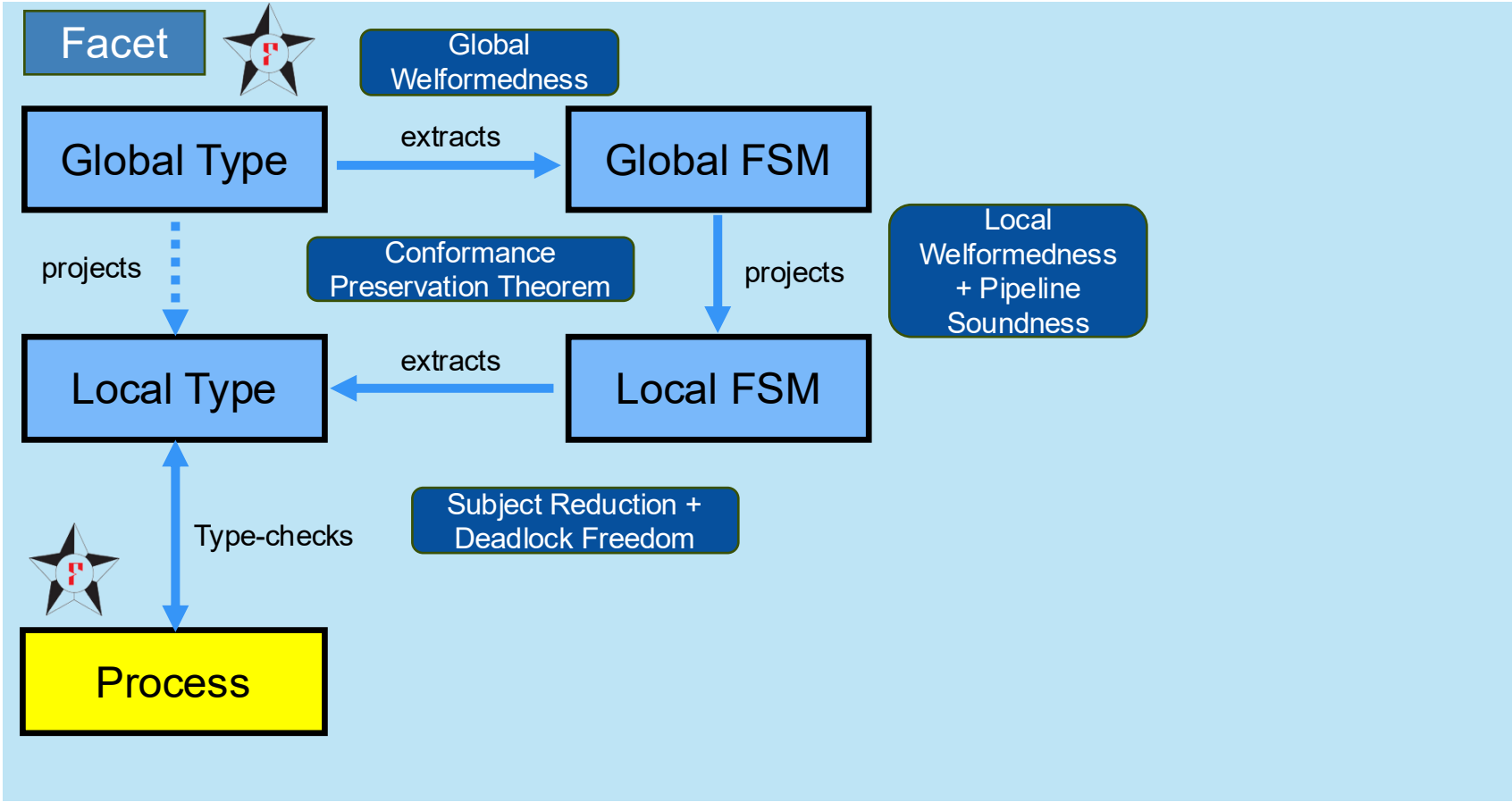


3 The protocol must only complete with *curr* = *n* or with an error message.

Idaho National Laboratory

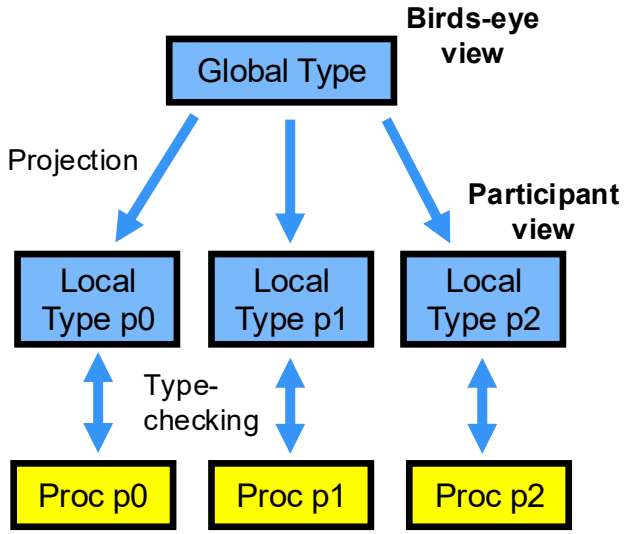
Facet: Mechanized RMPST metatheory in F*

1- Semantic Gap



Facet (RMPSTs)

Refined Multiparty Session Types



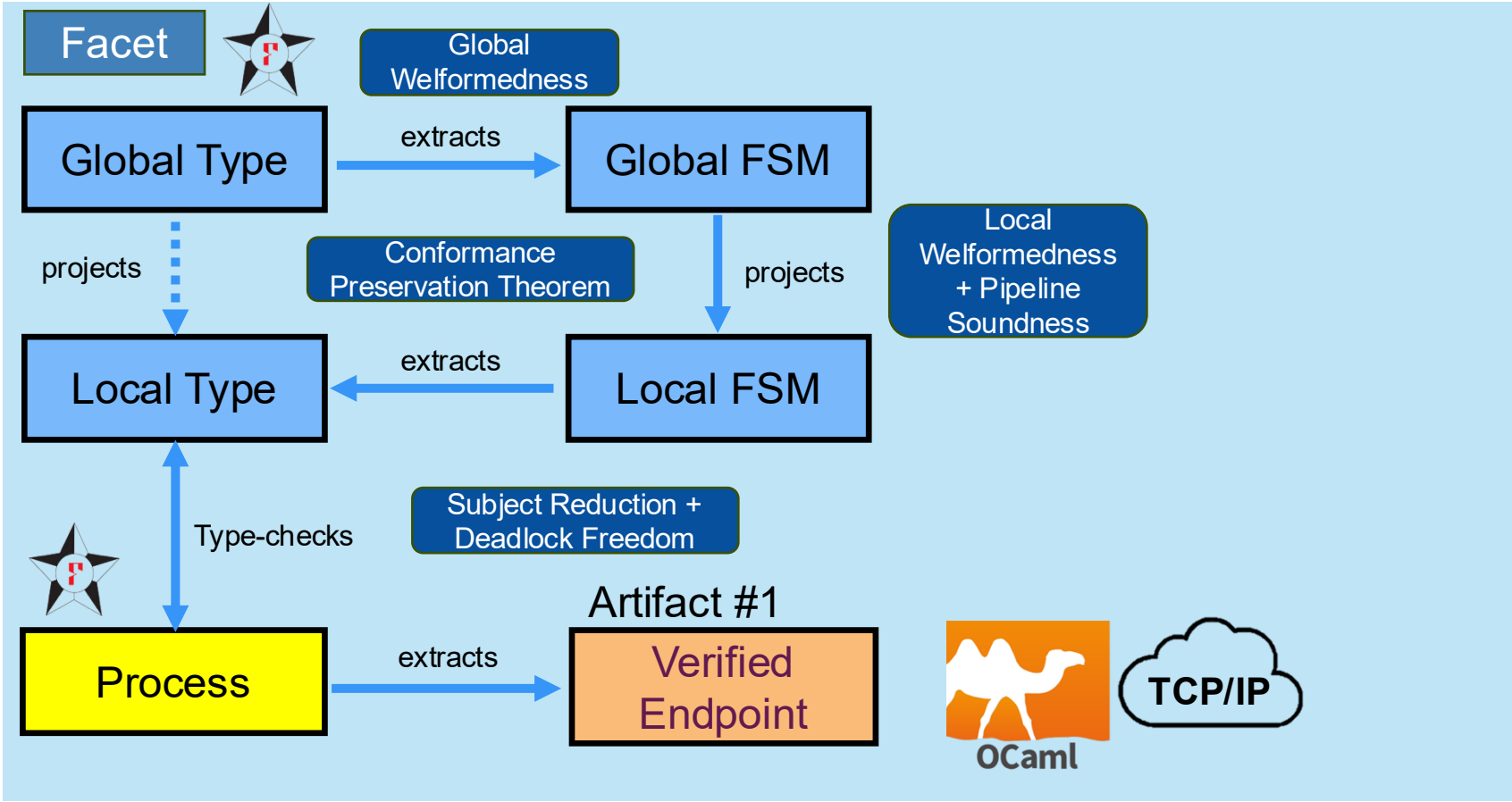
First mechanized RMPST metatheory

Proofs (e.g. absence of deadlocks) just by using our DSL

1- Amorim et al. Facet: Mechanized Theory of Refined Multiparty Protocols via Computational Reflection. Under Review

Facet: Mechanized RMPST metatheory in F*

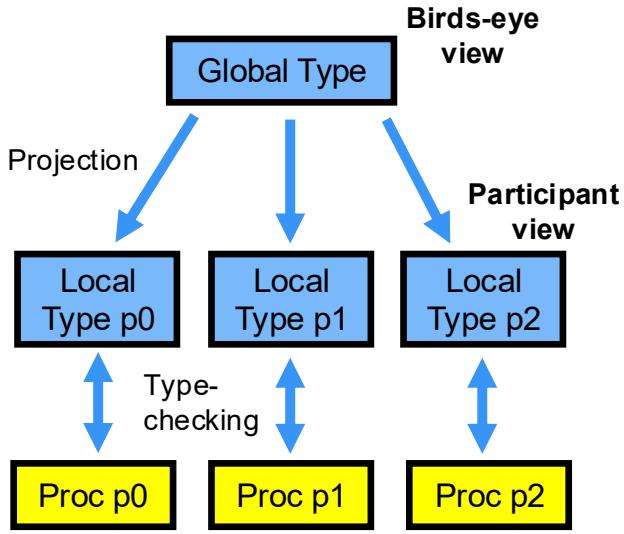
1- Semantic Gap



Facet

(RMPSTs)

Refined Multiparty Session Types



First mechanized RMPST metatheory

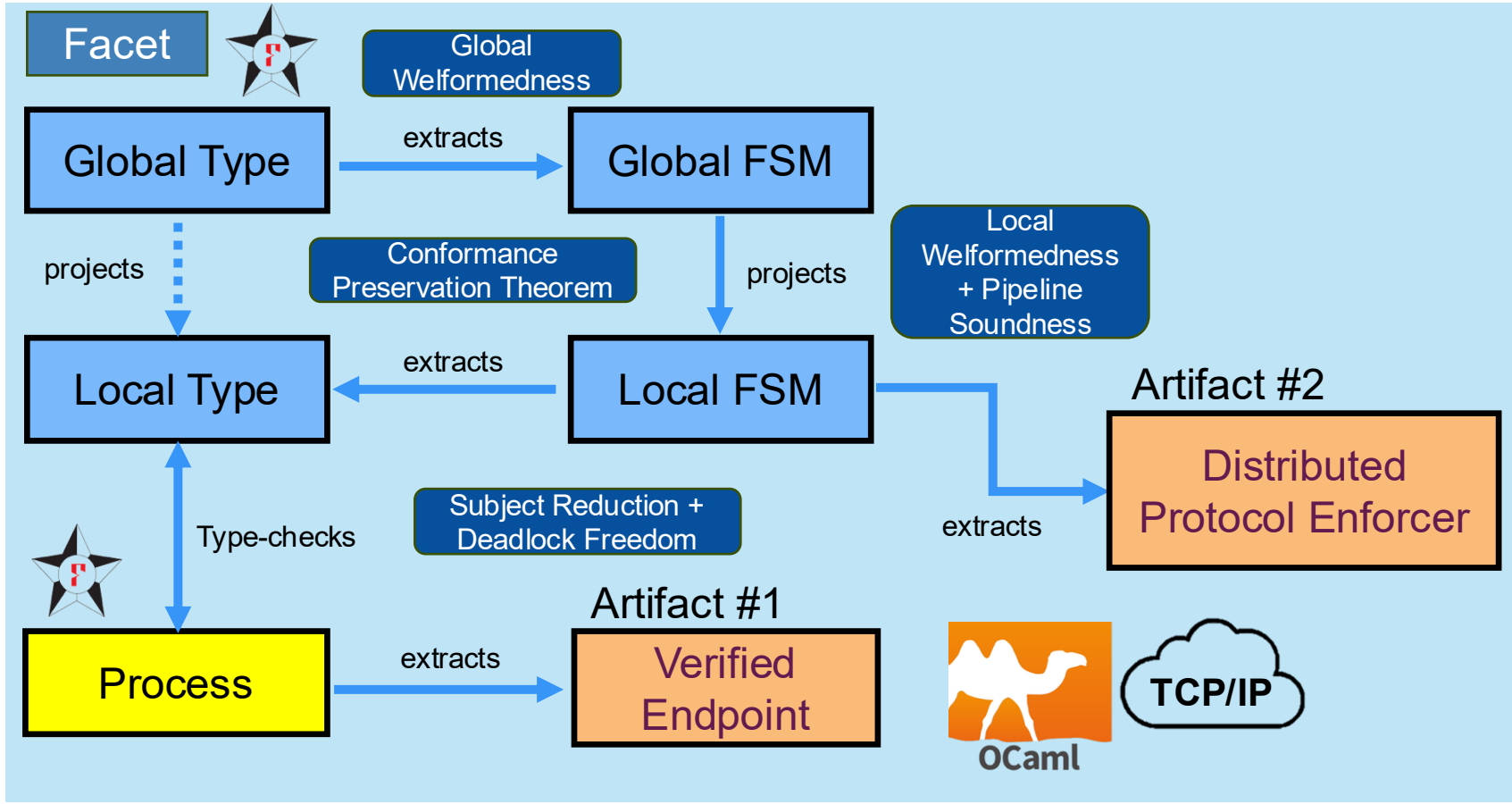
Proofs (e.g. absence of deadlocks) just by using our DSL

1- Amorim et al. Facet: Mechanized Theory of Refined Multiparty Protocols via Computational Reflection. Under Review

Idaho National Laboratory

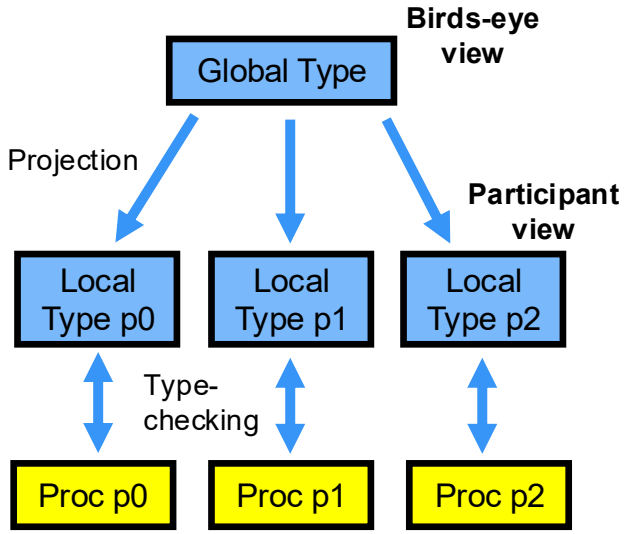
Facet: Mechanized RMPST metatheory in F*

1- Semantic Gap



Facet (RMPSTs)

Refined Multiparty Session Types



First mechanized RMPST metatheory

Proofs (e.g. absence of deadlocks) just by using our DSL

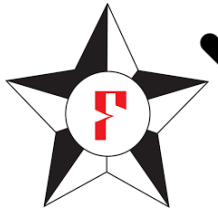
1- Amorim et al. Facet: Mechanized Theory of Refined Multiparty Protocols via Computational Reflection. Under Review

Facet Closes the Semantic Gap With Mechanized Metatheory

Artifact #1

Verified Endpoint

Projection

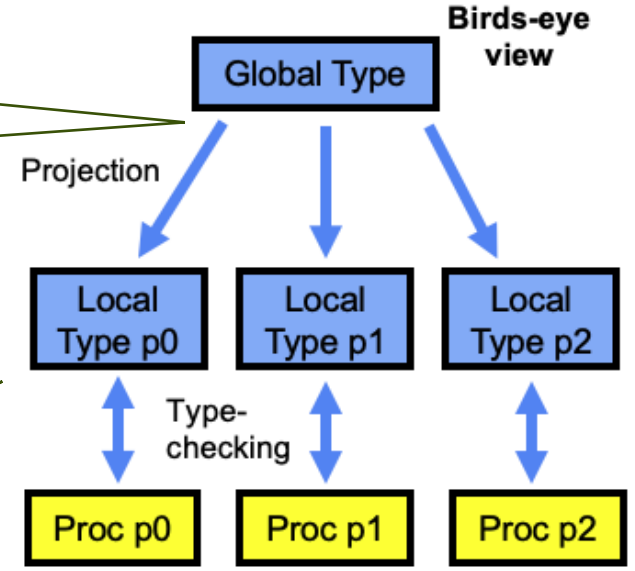


$$\begin{array}{l}\textbf{Server} \rightarrow \textbf{Auth} : \texttt{PIN}\ (expected\_code : \mathbb{Z}\ \{expected\_code \geq 0 \wedge expected\_code \leq 999999\}) \\ \mu.\mathbf{T}(count : \mathbb{Z}\{0 < count \wedge count \leq 5\})\ 1 \\ \quad \textbf{Auth} \rightarrow \textbf{User} : \texttt{RequestAuth}\ (\_ : \texttt{unit}) \\ \quad \textbf{User} \rightarrow \textbf{Auth} : \texttt{SubmitCode}\ (submitted : \mathbb{Z}\ \{submitted \geq 0 \wedge submitted \leq 999999\}) \\ \quad \textbf{Auth} \rightarrow \textbf{User} : \begin{cases} \texttt{Success}\ (\_ : \texttt{unit}\ \{submitted = expected\_code\}).\mathbf{End} \\ \texttt{Failed}\ (\_ : \texttt{unit}\ \{submitted \neq expected\_code \wedge count < 5\}).\ \mathbf{T}(count + 1) \\ \texttt{Locked}\ (\_ : \texttt{unit}\ \{submitted \neq expected\_code \wedge count = 5\}).\mathbf{End} \end{cases}\end{array}$$

$$\begin{array}{l}\mu.\ \mathbf{L}(count : \mathbb{Z}\{0 < count \leq 5\})\ : \\ \quad ?\ \textbf{Auth}\ :\ \texttt{RequestAuth}\ (\_ : \texttt{unit}); \\ \quad !\ \textbf{Auth}\ :\ \texttt{SubmitCode}\ (submitted : \mathbb{Z}\ \{submitted \geq 0 \wedge submitted \leq 999999\}); \\ \quad \&\ \textbf{Auth}\ : \{\ \texttt{Success}\ (\_ : \texttt{unit}).\mathbf{End} \mid \texttt{Failed}\ (\_ : \texttt{unit}).\ \mathbf{L}(count + 1)\ \mid \texttt{Locked}\ (\_ : \texttt{unit}).\mathbf{End}\}\end{array}$$

Messages are **statically** verified against refinements

```
let authenticator_p1 : process =
  Recv 0 [
    ("PIN" ==> fun (expected:int{expected > 0 && expected <= 999999}) ->
      Loop 1 (fun (count:int{count > 0 && count <= 5}) ->
        Send 2 [
          always_send "RequestAuth" () (
            Recv 2 [
              ("SubmitCode" ==> fun (code:int{code >= 0 && code <= 999999}) ->
                Send 2 [
                  send "Success"
                    (fun () -> code = expected)
                    ()
                    Done;
```



Built-in Proofs

1- Absence of Deadlocks
2- Guarded Recursion
3- Session Fidelity
4- Label Uniqueness

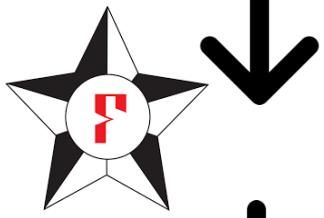
Idaho National Laboratory

Facet Closes the Semantic Gap With Mechanized Metatheory

Artifact #2

Distributed Protocol Enforcer

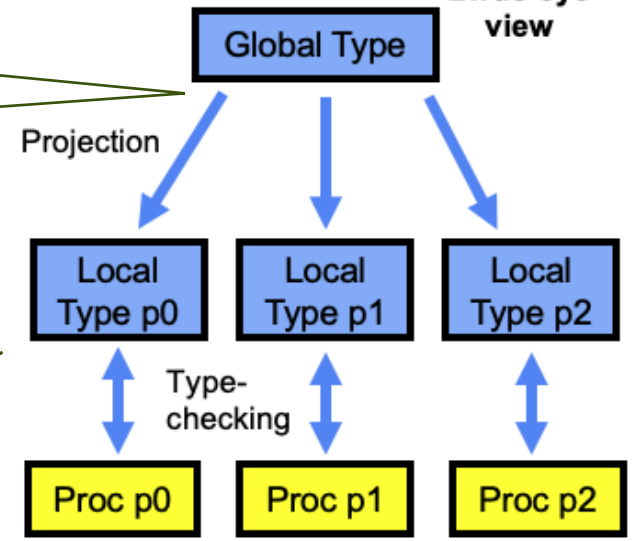
Projection



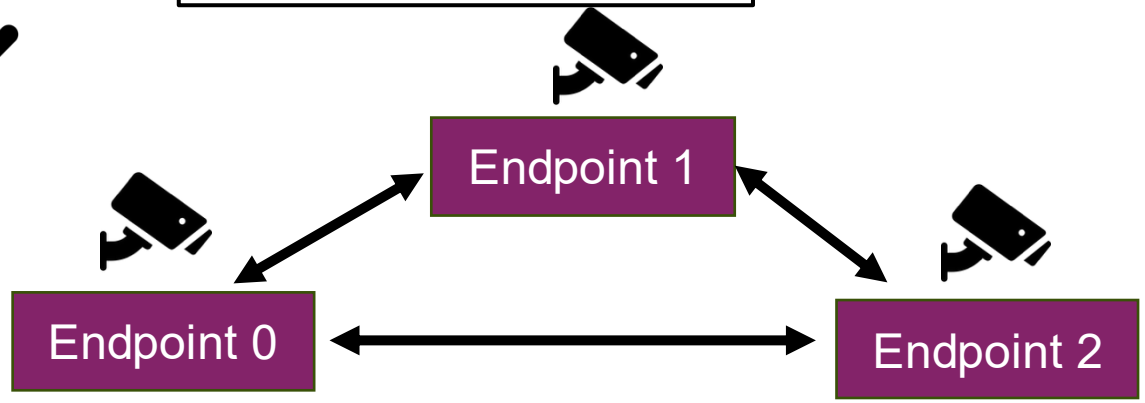
$$\begin{aligned}
&\textbf{Server} \rightarrow \textbf{Auth} : \texttt{PIN}\ (expected_code : \mathbb{Z}\ \{expected_code \geq 0 \wedge expected_code \leq 999999\}) \\
&\mu.\mathbf{T}(count : \mathbb{Z}\{0 < count \wedge count \leq 5\})\ 1 \\
&\quad \textbf{Auth} \rightarrow \textbf{User} : \texttt{RequestAuth}\ (_ : \texttt{unit}) \\
&\quad \textbf{User} \rightarrow \textbf{Auth} : \texttt{SubmitCode}\ (submitted : \mathbb{Z}\ \{submitted \geq 0 \wedge submitted \leq 999999\}) \\
&\quad \textbf{Auth} \rightarrow \textbf{User} : \begin{cases} \texttt{Success}\ (_ : \texttt{unit}\ \{submitted = expected_code\}).\mathbf{End} \\ \texttt{Failed}\ (_ : \texttt{unit}\ \{submitted \neq expected_code \wedge count < 5\}).\ \mathbf{T}(count + 1) \\ \texttt{Locked}\ (_ : \texttt{unit}\ \{submitted \neq expected_code \wedge count = 5\}).\mathbf{End} \end{cases}
\end{aligned}$$

$$\begin{aligned}
&\mu.\ \mathbf{L}(count : \mathbb{Z}\{0 < count \leq 5\})\ : \\
&\quad ?\ \textbf{Auth}\ :\ \texttt{RequestAuth}\ (_ : \texttt{unit}); \\
&\quad !\ \textbf{Auth}\ :\ \texttt{SubmitCode}\ (submitted : \mathbb{Z}\ \{submitted \geq 0 \wedge submitted \leq 999999\}); \\
&\quad \&\ \textbf{Auth}\ : \{\ \texttt{Success}\ (_ : \texttt{unit}).\mathbf{End} \mid \texttt{Failed}\ (_ : \texttt{unit}).\ \mathbf{L}(count + 1)\ \mid \texttt{Locked}\ (_ : \texttt{unit}).\mathbf{End}\}
\end{aligned}$$

Birds-eye view



Messages are **dynamically** verified against refinements



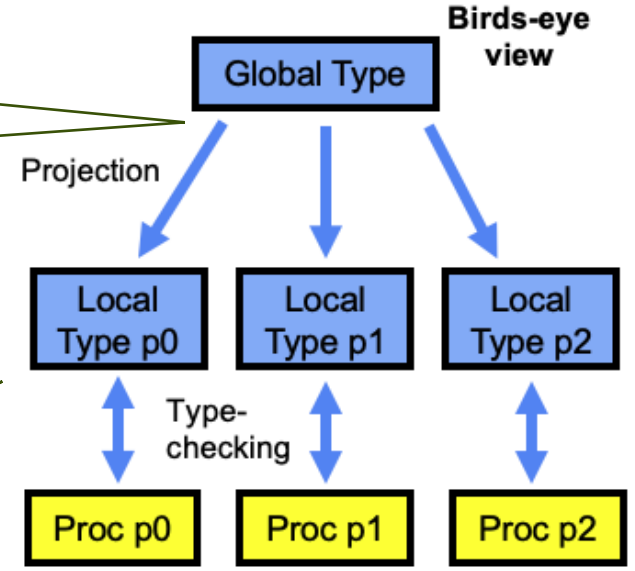
Built-in Proofs

1- Absence of Deadlocks
2- Guarded Recursion
3- Session Fidelity
4- Label Uniqueness

Facet Closes the Semantic Gap With Mechanized Metatheory

$$\begin{aligned}
&\textbf{Server} \rightarrow \textbf{Auth} : \texttt{PIN}\ (\mathit{expected_code} : \mathbb{Z}\ \{\mathit{expected_code} \geq 0 \wedge \mathit{expected_code} \leq 999999\}) \\
&\mu.\mathbf{T}(\mathit{count} : \mathbb{Z}\{0 < \mathit{count} \wedge \mathit{count} \leq 5\})\ 1 \\
&\quad \textbf{Auth} \rightarrow \textbf{User} : \texttt{RequestAuth}\ (_ : \texttt{unit}) \\
&\quad \textbf{User} \rightarrow \textbf{Auth} : \texttt{SubmitCode}\ (\mathit{submitted} : \mathbb{Z}\ \{\mathit{submitted} \geq 0 \wedge \mathit{submitted} \leq 999999\}) \\
&\quad \textbf{Auth} \rightarrow \textbf{User} : \begin{cases} \texttt{Success}\ (_ : \texttt{unit}\ \{\mathit{submitted} = \mathit{expected_code}\}).\mathbf{End} \\ \texttt{Failed}\ (_ : \texttt{unit}\ \{\mathit{submitted} \neq \mathit{expected_code} \wedge \mathit{count} < 5\}).\ \mathbf{T}(\mathit{count}+1) \\ \texttt{Locked}\ (_ : \texttt{unit}\ \{\mathit{submitted} \neq \mathit{expected_code} \wedge \mathit{count} = 5\}).\mathbf{End} \end{cases}
\end{aligned}$$

$$\begin{aligned}
&\mu.\ \mathbf{L}(\mathit{count} : \mathbb{Z}\{0 < \mathit{count} \leq 5\})\ : \\
&\quad ?\ \textbf{Auth}\ :\ \texttt{RequestAuth}\ (_ : \texttt{unit}); \\
&\quad !\ \textbf{Auth}\ :\ \texttt{SubmitCode}\ (\mathit{submitted} : \mathbb{Z}\ \{\mathit{submitted} \geq 0 \wedge \mathit{submitted} \leq 999999\}); \\
&\quad \&\ \textbf{Auth}\ : \{\ \texttt{Success}\ (_ : \texttt{unit}).\mathbf{End} \mid \texttt{Failed}\ (_ : \texttt{unit}).\ \mathbf{L}(\mathit{count}+1) \mid \texttt{Locked}\ (_ : \texttt{unit}).\mathbf{End}\}
\end{aligned}$$



Artifact #1

Verified Endpoint

Messages are **statically** verified against refinements

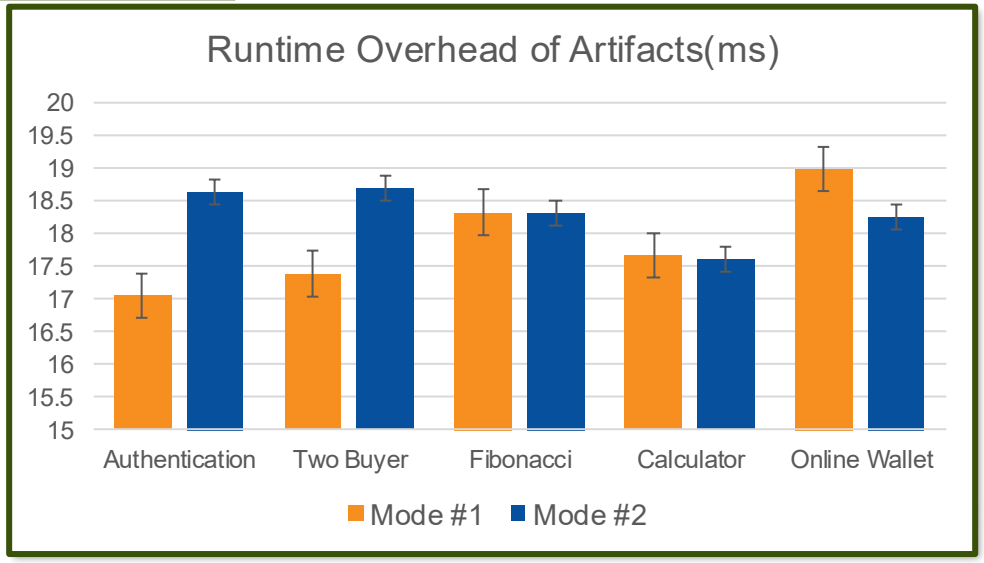
TCP/IP

OCaml

Artifact #2

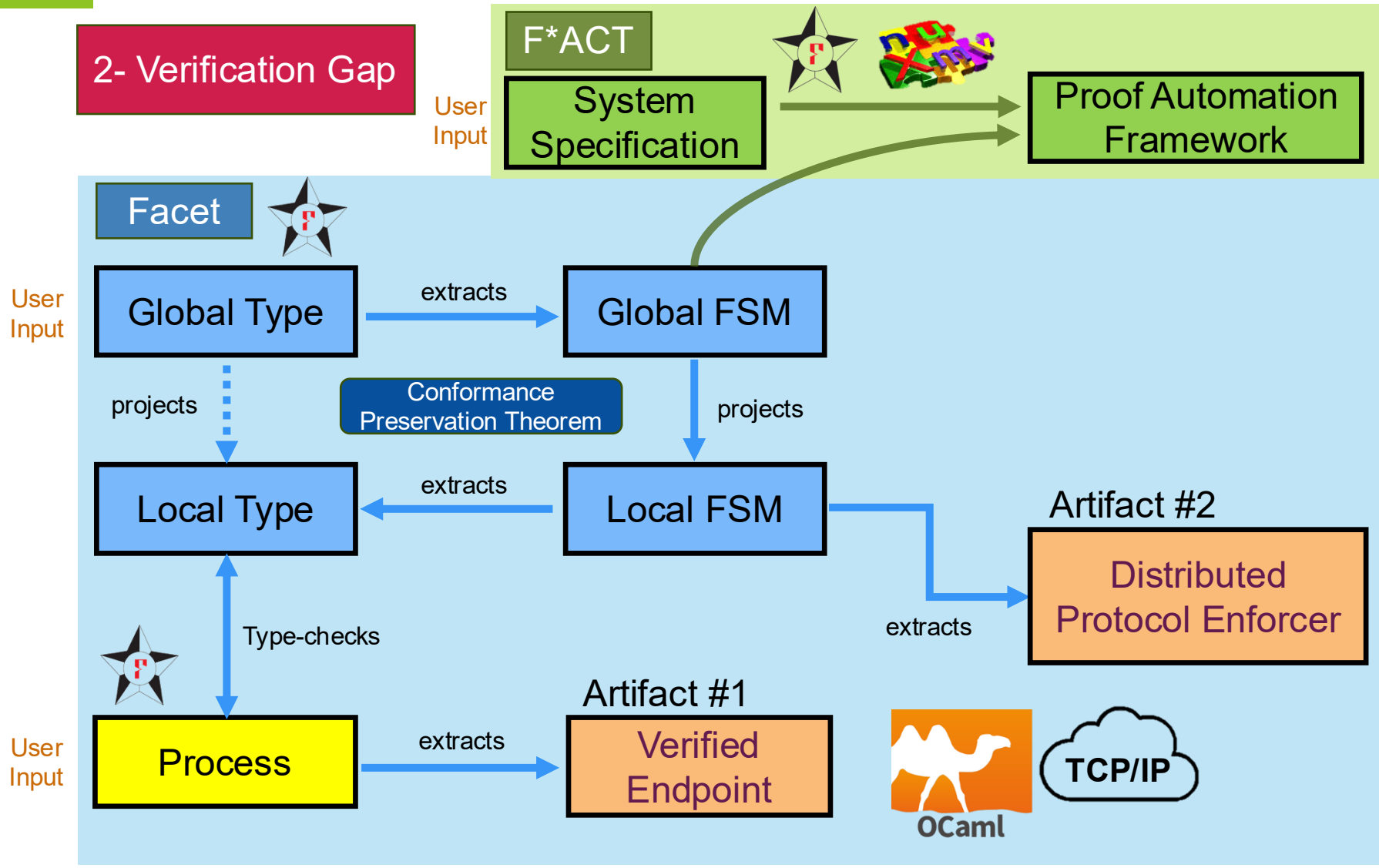
Distributed Protocol Enforcer

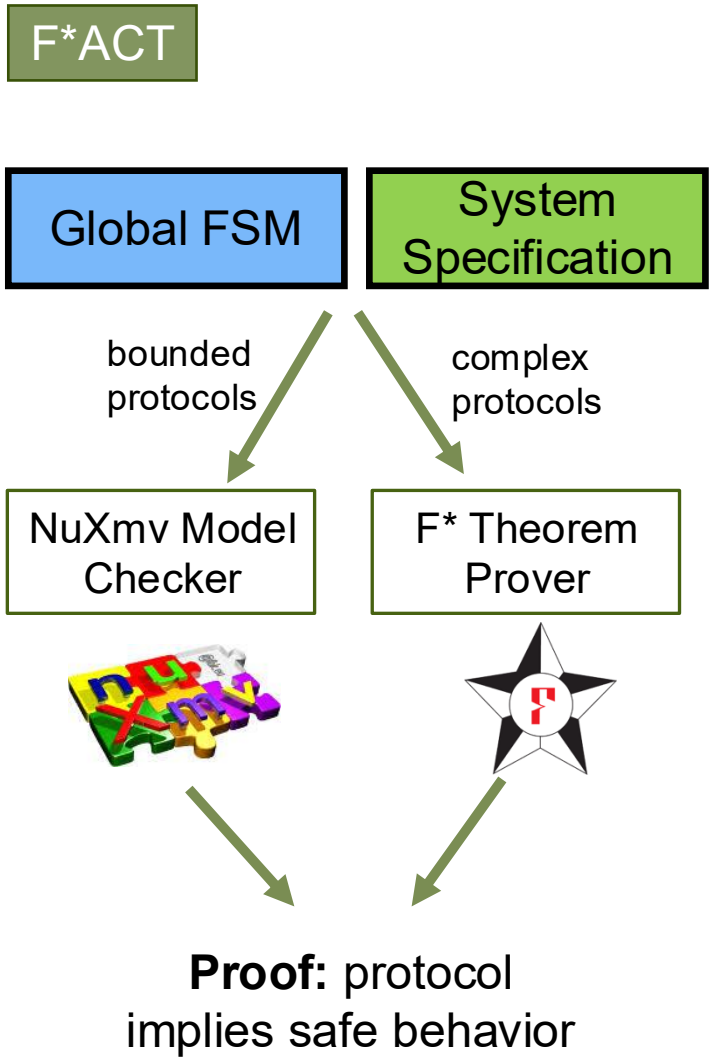
Messages are **dynamically** verified against refinements



Idaho National Laboratory

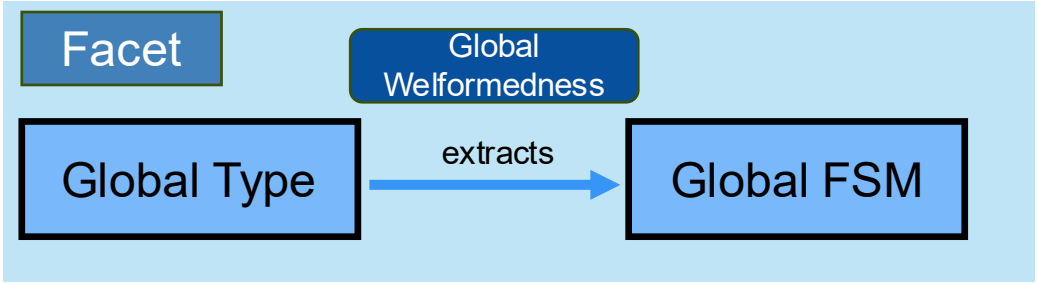
F*ACT: Automated and Scalable Proof Methods.





2- Amorim et al. F*ACT: Formally Assured Communicating Types. In Preparation

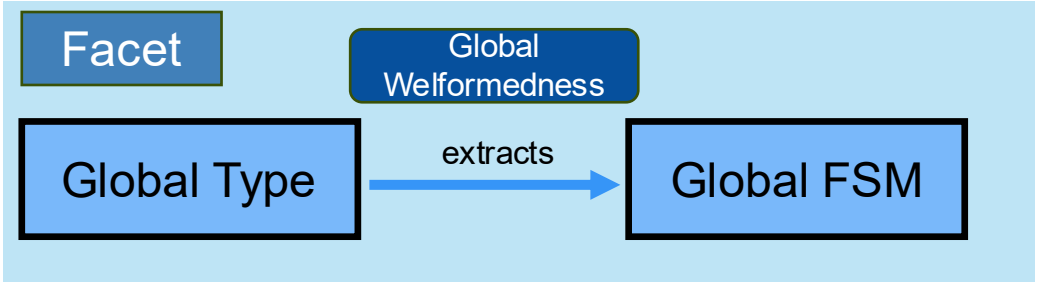
The Engine That Makes This Tractable: Shallow-to-Deep Embedding Extraction



RMPST

$$\begin{aligned}
&\mathbf{GCS} \to \mathbf{UAV}\colon \texttt{MISSION COUNT}(n\{1 \le n < \text{LIMIT}\}) \\
&\mu.T(curr\{0 \le curr < n\})(curr = 0) \\
&\quad \mathbf{UAV} \to \mathbf{GCS}\colon \\
&\qquad \texttt{MISSION REQUEST INT}(req\{req = curr < n\}) \\
&\qquad \texttt{MISSION ITEM INT}(item\{item = curr < n\}) \\
&\qquad T(curr + 1) \\
&\quad \oplus\, \texttt{MISSION ACK}(ack\{\text{type} = \text{ERROR} \vee curr = n\}).\mathbf{End}
\end{aligned}$$

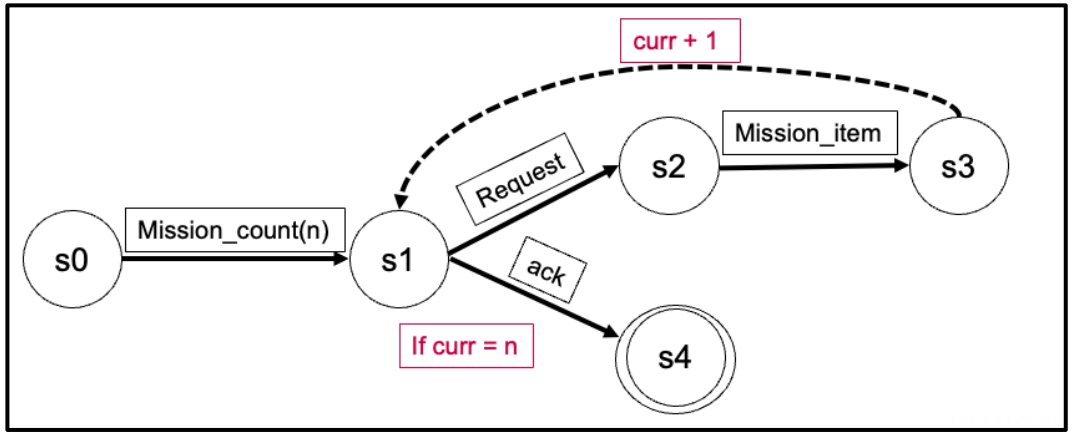
The Engine That Makes This Tractable: Shallow-to-Deep Embedding Extraction



RMPST

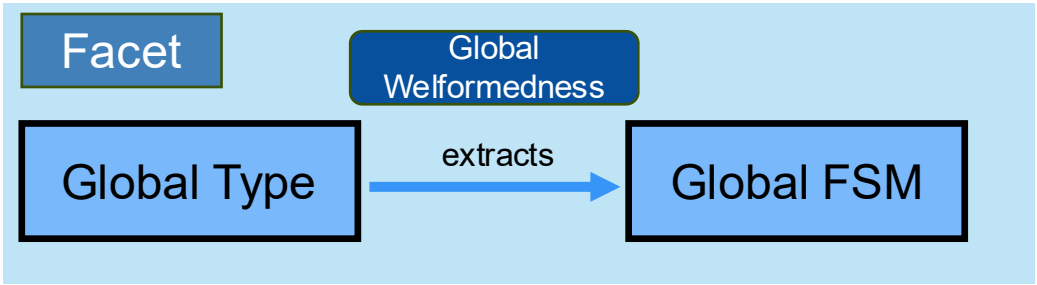
$$\begin{aligned}
&\mathbf{GCS} \to \mathbf{UAV}: \texttt{MISSION COUNT}(n\{1 \le n < \text{LIMIT}\}) \\
&\mu.T(curr\{0 \le curr < n\})(curr = 0) \\
&\quad \mathbf{UAV} \to \mathbf{GCS}: \\
&\qquad \texttt{MISSION REQUEST INT}(req\{req = curr < n\}) \\
&\qquad \texttt{MISSION ITEM INT}(item\{item = curr < n\}) \\
&\qquad T(curr + 1) \\
&\quad \oplus \texttt{MISSION ACK}(ack\{\text{type} = \text{ERROR} \vee curr = n\}).\mathbf{End}
\end{aligned}$$

Proof Internal



Idaho National Laboratory

The Engine That Makes This Tractable: Shallow-to-Deep Embedding Extraction

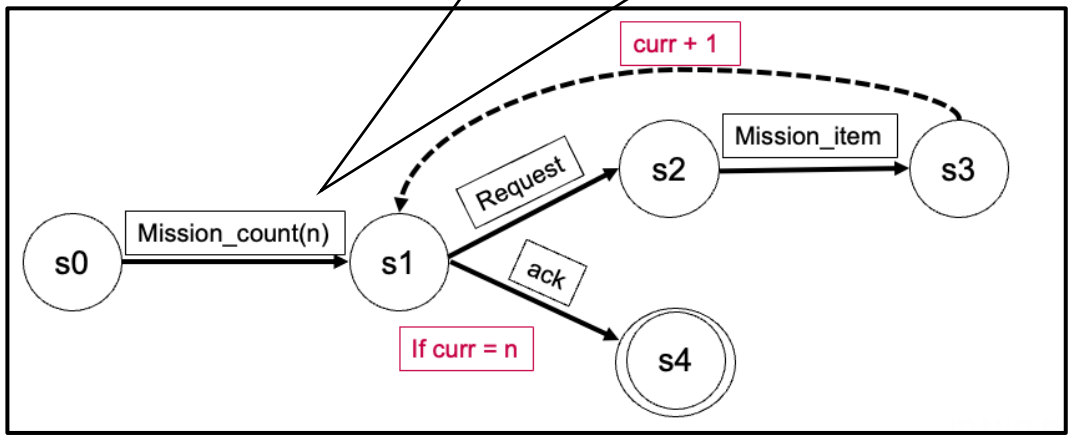



```
Mkprotocol_transition "MISSION_COUNT" 0 1
[
        Mkvar "mc_target_system" KindInt;
        Mkvar "mc_target_component" KindInt;
        Mkvar "mc_count" KindInt;
        Mkvar "mc_mission_type" KindInt;
        Mkvar "mc_opaque_id" KindInt
      ]
      (GAnd (GGe (TVar "mc_count") (TIntLit
1)) (GLt (TVar "mc_count") (TIntLit 65535)))
```

RMPST

$$\begin{aligned}
&\mathbf{GCS} \to \mathbf{UAV}: \texttt{MISSION_COUNT}(n\{1 \le n < \texttt{LIMIT}\}) \\
&\mu.T(curr\{0 \le curr < n\})(curr = 0) \\
&\quad \mathbf{UAV} \to \mathbf{GCS}: \\
&\qquad \texttt{MISSION_REQUEST_INT}(req\{req = curr < n\}) \\
&\qquad \texttt{MISSION_ITEM_INT}(item\{item = curr < n\}) \\
&\qquad T(curr + 1) \\
&\quad \oplus \texttt{MISSION_ACK}(ack\{\text{type} = \text{ERROR} \vee curr = n\}).\mathbf{End}
\end{aligned}$$

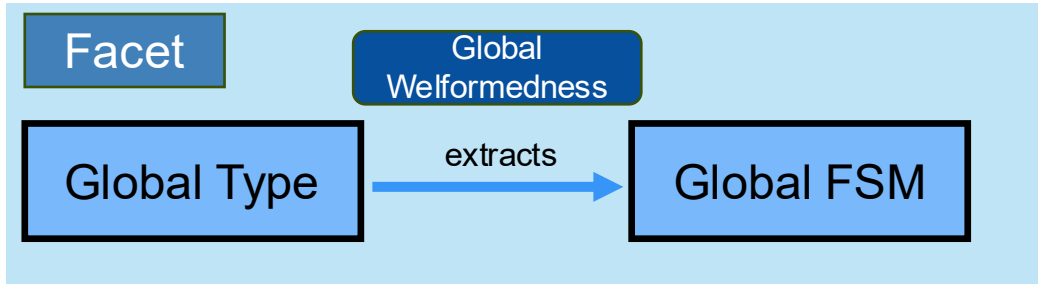
Proof Internal



Our DSL:

$$(0 --> 1)\ [Option\ \text{"MISSION\_COUNT"}\ (\ \lambda\ (cnt : mission\_count\ \{\ cnt\ >=\ 1\ \&\&\ cnt\ <\ mission\_limit\})$$

The Engine That Makes This Tractable: Shallow-to-Deep Embedding Extraction



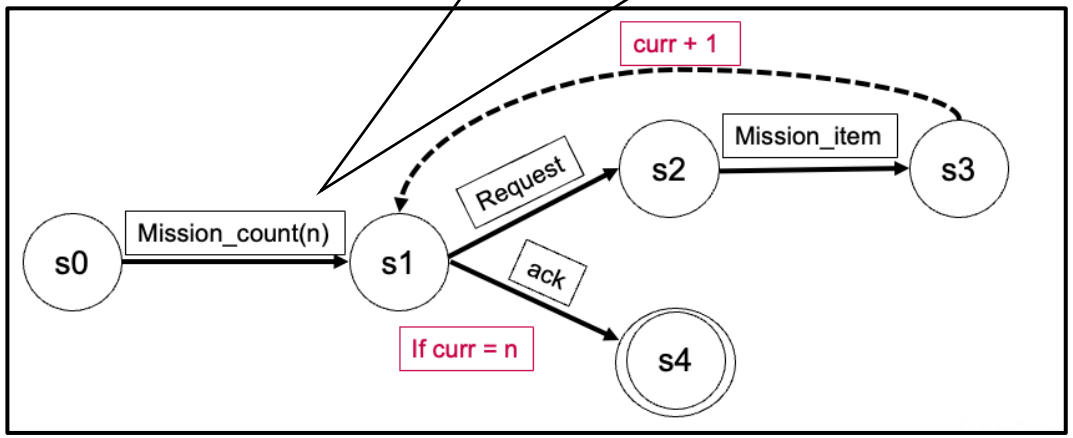

```
Mkprotocol_transition "MISSION_COUNT" 0 1
[
        Mkvar "mc_target_system" KindInt;
        Mkvar "mc_target_component" KindInt;
        Mkvar "mc_count" KindInt;
        Mkvar "mc_mission_type" KindInt;
        Mkvar "mc_opaque_id" KindInt
      ]
      (GAnd (GGe (TVar "mc_count") (TIntLit
1)) (GLt (TVar "mc_count") (TIntLit 65535)))
```

RMPST

$$\begin{aligned}
&\mathbf{GCS} \to \mathbf{UAV}: \texttt{MISSION_COUNT}(n\{1 \le n < \texttt{LIMIT}\}) \\
&\mu.T(curr\{0 \le curr < n\})(curr = 0) \\
&\quad \mathbf{UAV} \to \mathbf{GCS}: \\
&\qquad \texttt{MISSION_REQUEST_INT}(req\{req = curr < n\}) \\
&\qquad \texttt{MISSION_ITEM_INT}(item\{item = curr < n\}) \\
&\qquad T(curr + 1) \\
&\quad \oplus \texttt{MISSION_ACK}(ack\{\text{type} = \text{ERROR} \vee curr = n\}).\mathbf{End}
\end{aligned}$$

Meta-F*

Proof Internal

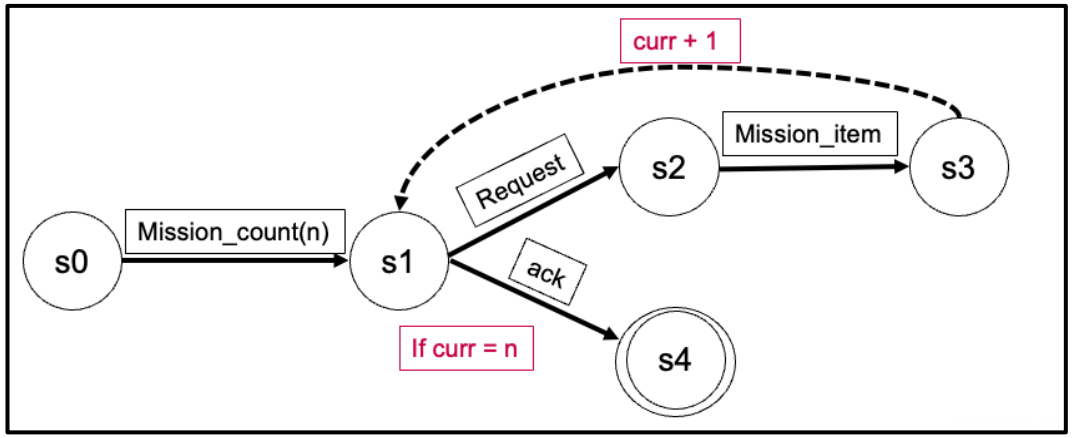


Our DSL:

$$(0 \texttt{--->} 1)\ [Option\ \text{"MISSION\_COUNT"}\ (\ \lambda\ (cnt : mission\_count\ \{\ cnt\ >=\ 1\ \&\&\ cnt\ <\ mission\_limit\})$$

F*ACT Proves Application-Specific Safety Without State Explosion

Global FSM



System Specification

Safety Properties + System Semantics

- ”The GCS must send all missions in order without skips”
- “ An ack(success) message can only be sent when *curr* = *n*”

Operational Semantics

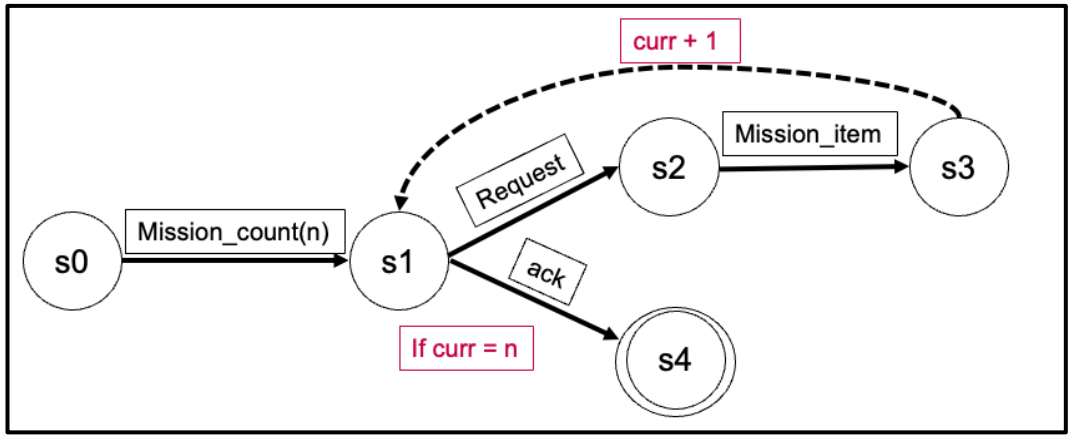
$$State + Command \rightarrow State \mid WRONG$$

Valid State

Idaho National Laboratory

F*ACT Proves Application-Specific Safety Without State Explosion

Global FSM



System Specification

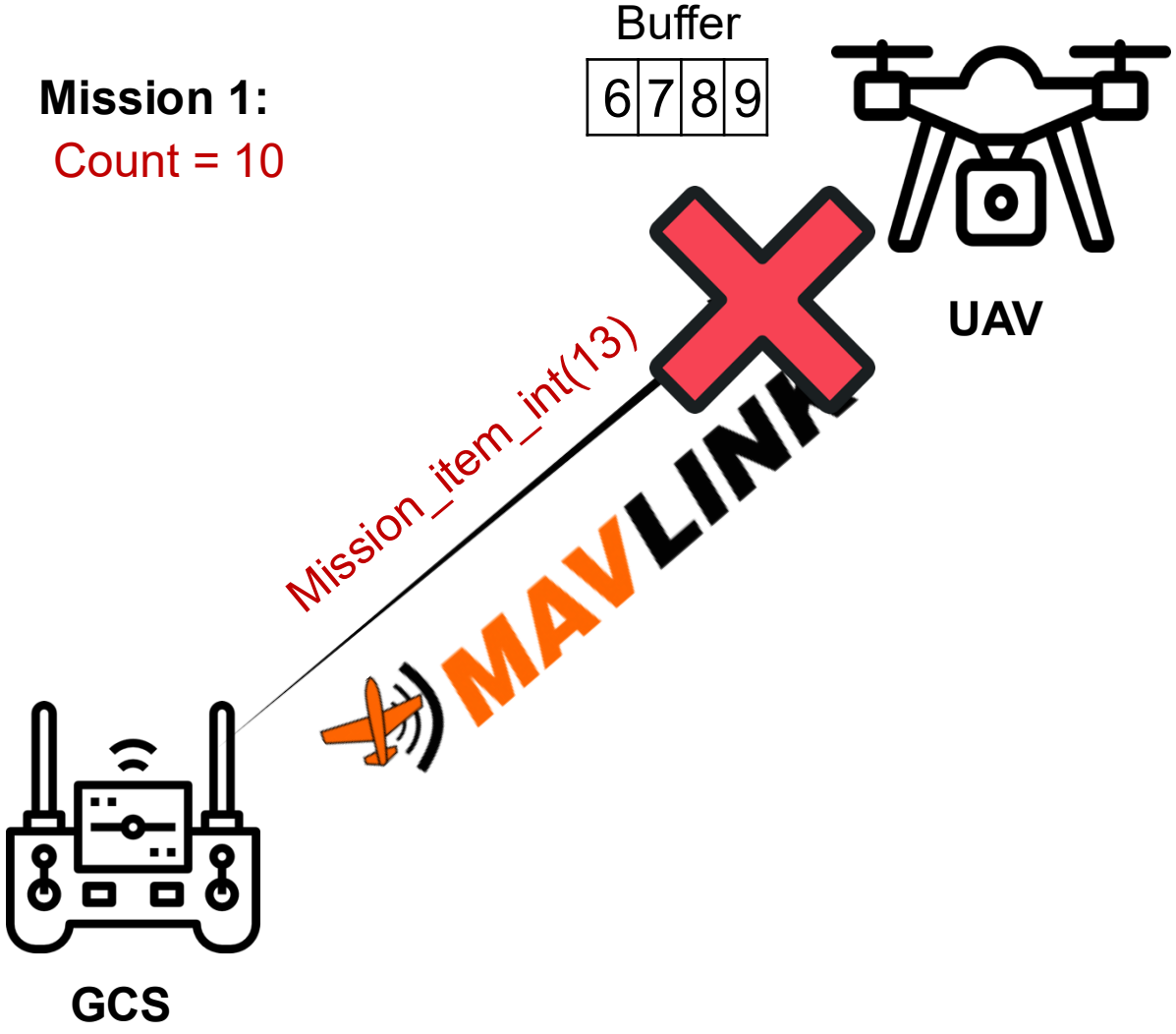
Safety Properties + System Semantics

- ”The GCS must send all missions in order without skips”
- “ An ack(success) message can only be sent when *curr* = *n*”

Operational Semantics

$$(n = 10; \mathrm{m} = 13) \ + \ \text{Succeed} \ \rightarrow \ \boldsymbol{WRONG}$$

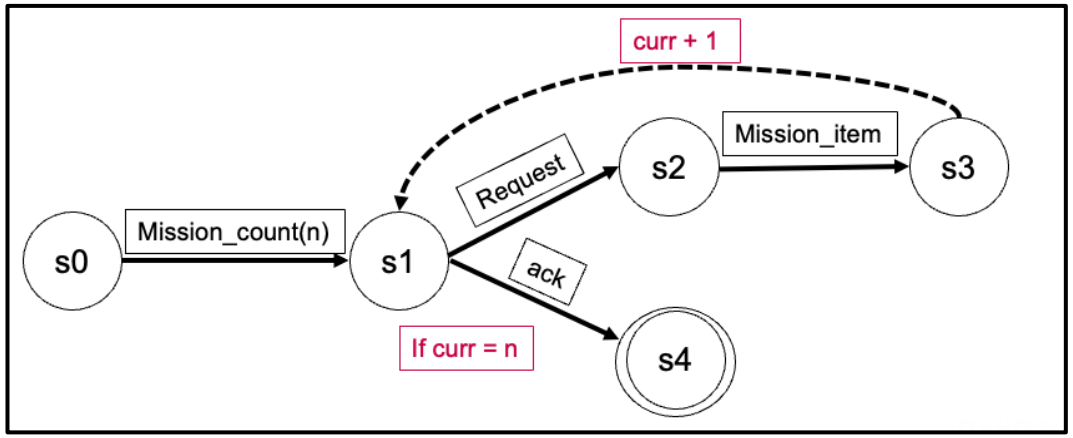
Mission 1:
Count = 10



Safety Condition: following a protocol must never lead the system to a WRONG state

F*ACT Proves Application-Specific Safety Without State Explosion

Global FSM



Model Checking

1- Variables

2- Initial values

3- States (variable updates)

4- Safety properties

System Specification

Safety Properties + System Semantics

- ”The GCS must send all missions in order without skips”
- “ An ack(success) message can only be sent when *curr* = *n*”

Operational Semantics

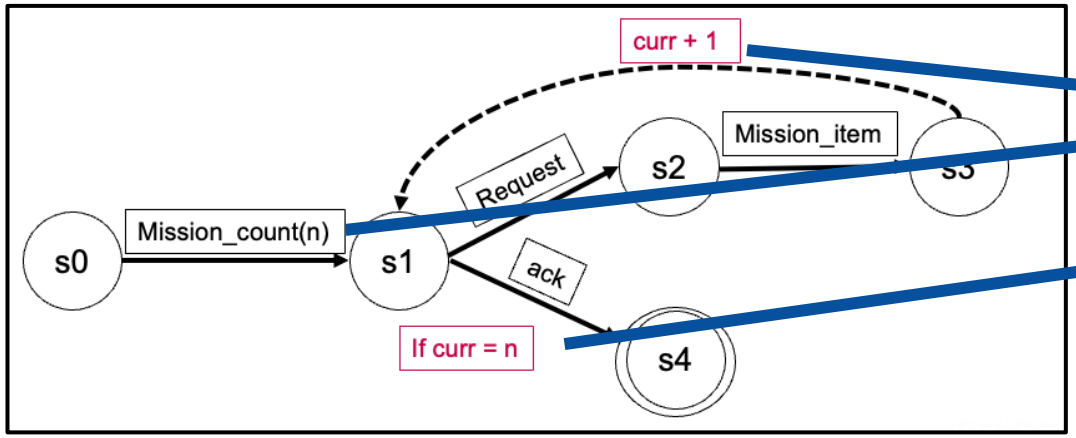
$$(n = 10; \mathrm{m} = 13) \ + \ \text{Succeed} \ \rightarrow \ \boldsymbol{WRONG}$$

Safety Condition: following a protocol must never lead the system to a WRONG state

F*ACT Proves Application-Specific Safety Without State Explosion

Model Checking

Global FSM



1- Variables

2- Initial values

3- States (variable updates)

4- Safety properties

System Specification

Safety Properties + System Semantics

- ”The GCS must send all missions in order without skips”
- “ An ack(success) message can only be sent when *curr* = *n*”

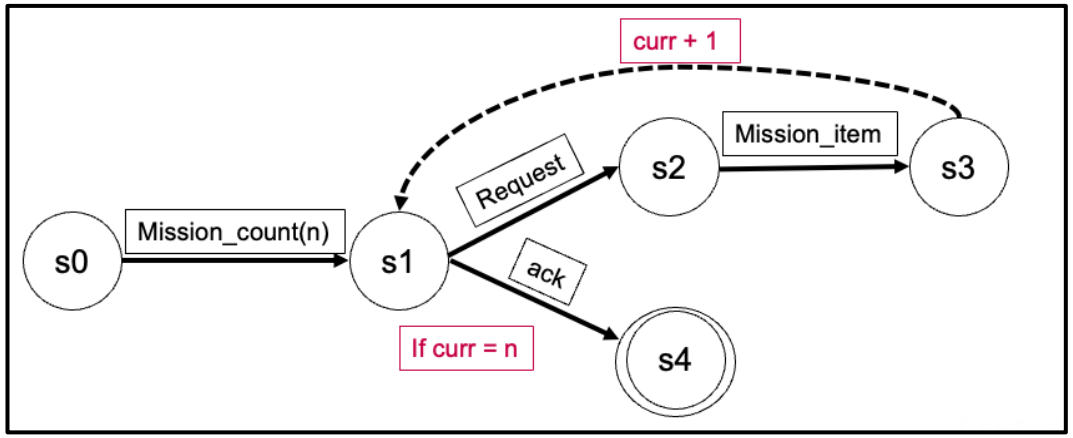
Operational Semantics

$$(n = 10; \mathrm{m} = 13) \ + \ \text{Succeed} \ \rightarrow \ \boldsymbol{WRONG}$$

Safety Condition: following a protocol must never lead the system to a WRONG state

F*ACT Proves Application-Specific Safety Without State Explosion

Global FSM



System Specification

Safety Properties + System Semantics

- ”The GCS must send all missions in order without skips”
- “ An ack(success) message can only be sent when *curr* = *n*”

Operational Semantics

$$(n = 10; \mathrm{m} = 13) \ + \ \text{Succeed} \ \rightarrow \ \boldsymbol{WRONG}$$

Meta-F*

Model Checking

1- Variables

2- Initial values

3- States (variable updates)

4- Safety properties

Theorem proving

Safety Condition: following a protocol must never lead the system to a WRONG state

Platum: Verified Runtime Monitors for Legacy Protocols.

3- Legacy Lock-in

User Input

F*ACT

System Specification

Proof Automation Framework

Facet

Global Welformedness

User Input

Global Type

extracts

Global FSM

projects

Conformance Preservation Theorem

projects

Local Type

extracts

Local FSM

Type-checks

extracts

User Input

Process

extracts

Verified Endpoint

OCaml

TCP/IP

Platum

synthesizes

Artifact #3

Centralized Runtime Monitor

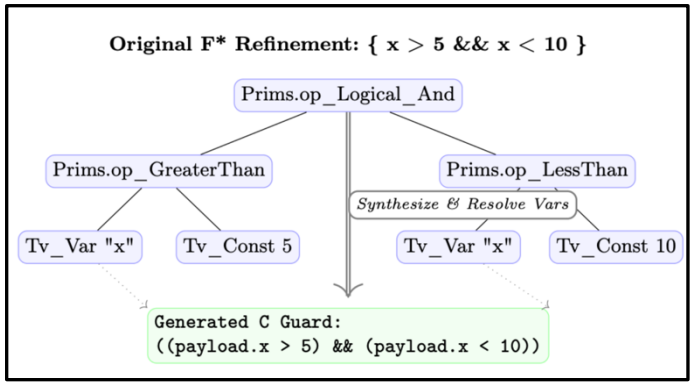
Artifact #2

Distributed Protocol Enforcer

Artifact #1

Platum

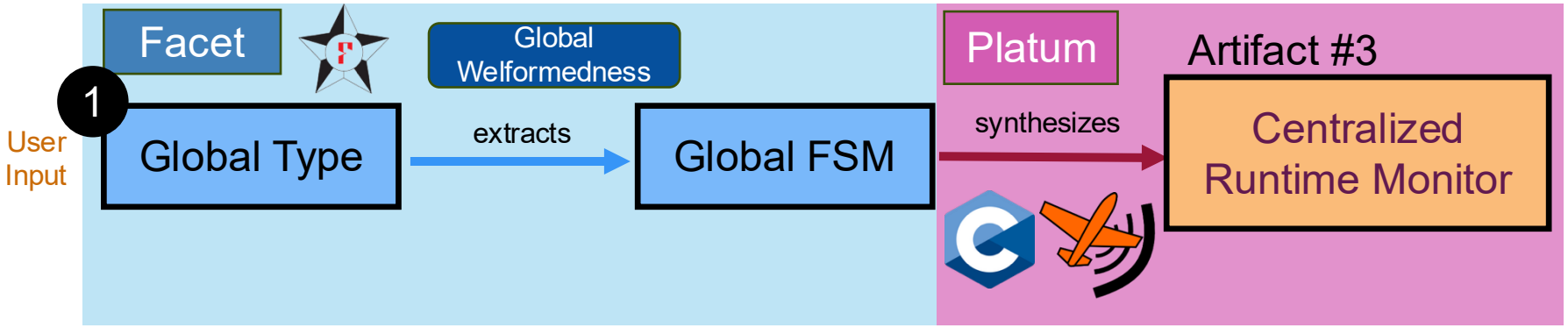
retrofits existing protocols



Memory overhead: 8.39Mb
Latency overhead: 13.33µs

3- Amorim et al. From High-Level Types to Low-Level Monitors: Synthesizing Verified Runtime Checkers for MAVLink To appear in NFM'26

Platum Retrofits Legacy Protocols Without Replacing Them



MAVLink spec

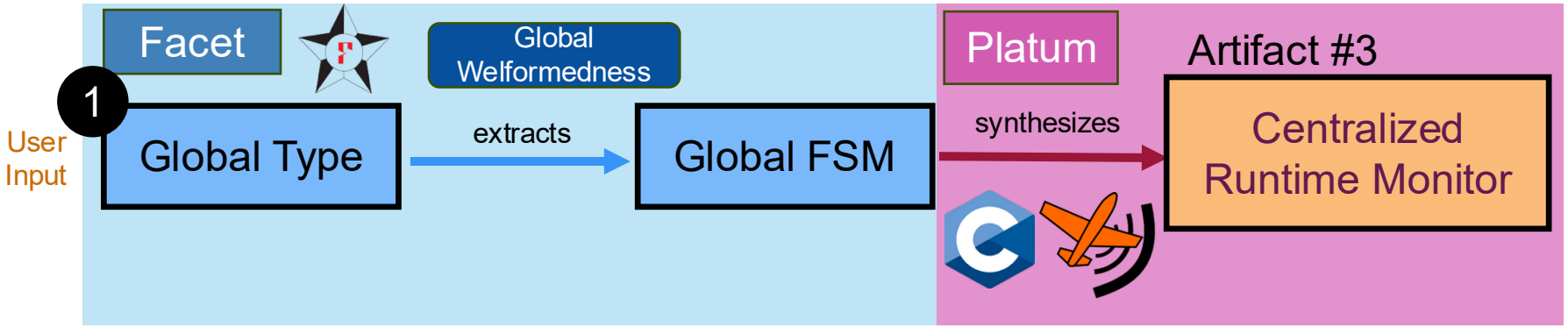
Common.xml

```
<message
name="MISSION_COUNT">
<field type="uint16_t"
name="count">
</field>
</message>"
```

???

Idaho National Laboratory

Platum Retrofits Legacy Protocols Without Replacing Them



MAVLink spec

Common.xml

```
<message
name="MISSION_COUNT">
<field type="uint16_t"
name="count">
</field>
</message>"
```

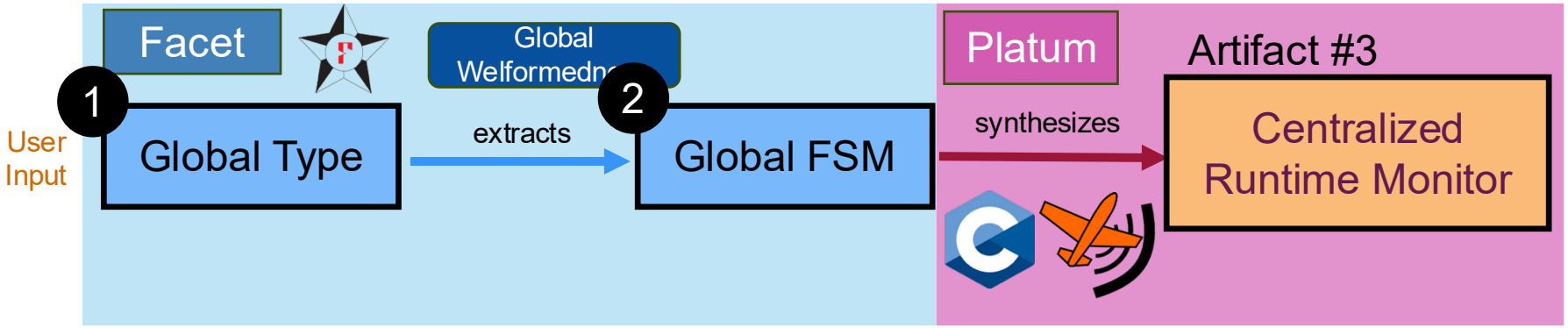
Type extraction

8kLoc

```
type mission_count = {
count : uint16_t;
}
```

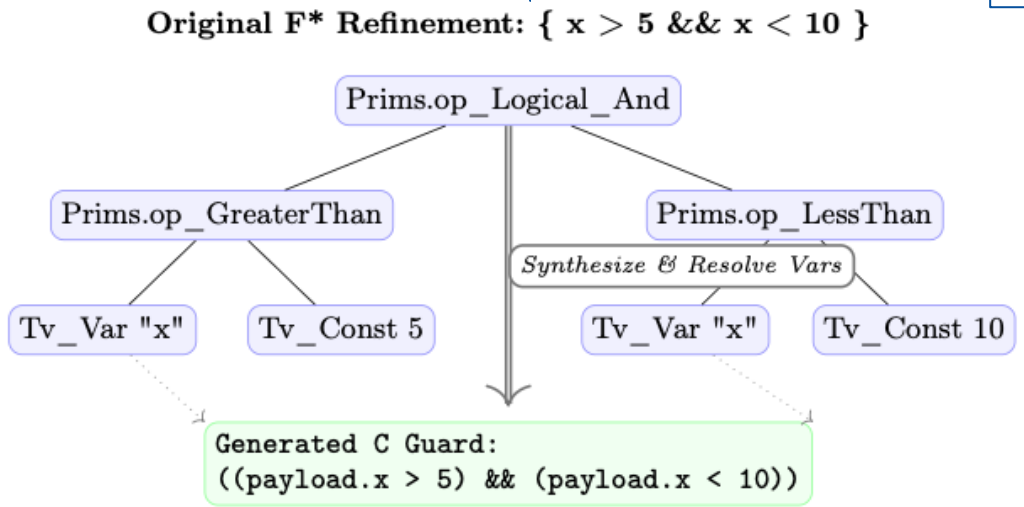
Idaho National Laboratory

Platum Retrofits Legacy Protocols Without Replacing Them



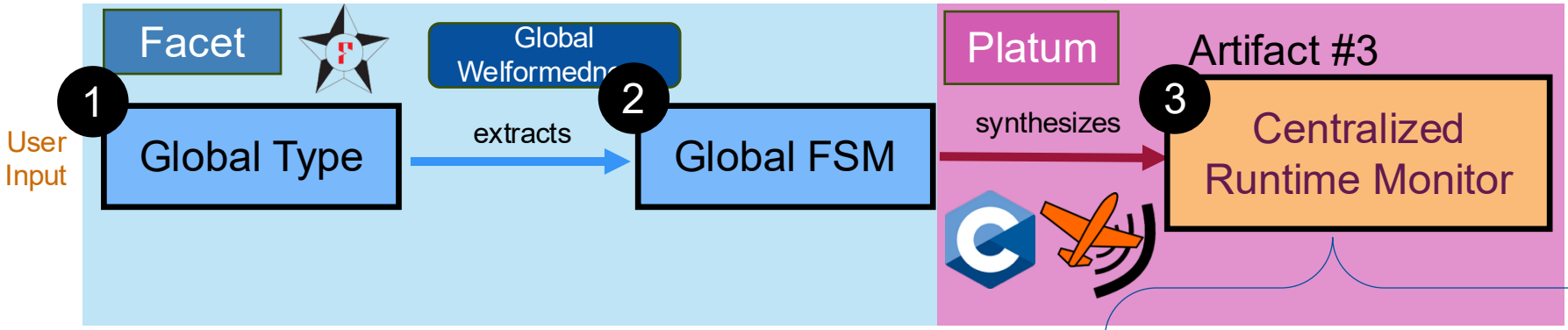
$$(cnt : mission\_count\{\, cnt >= 1 \,\&\&\, cnt < mission\_limit\})$$

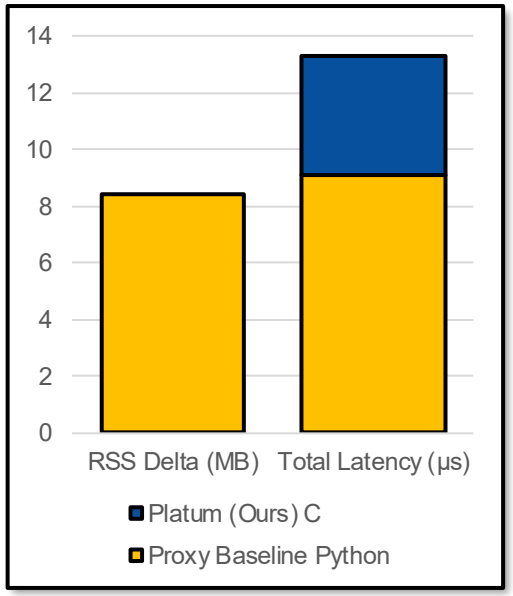
Type Erasure



Idaho National Laboratory

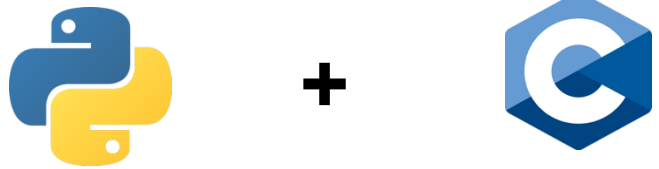
Platum Retrofits Legacy Protocols Without Replacing Them





Memory Overhead: 8.39 Mb

Latency Overhead: 13.3 µs

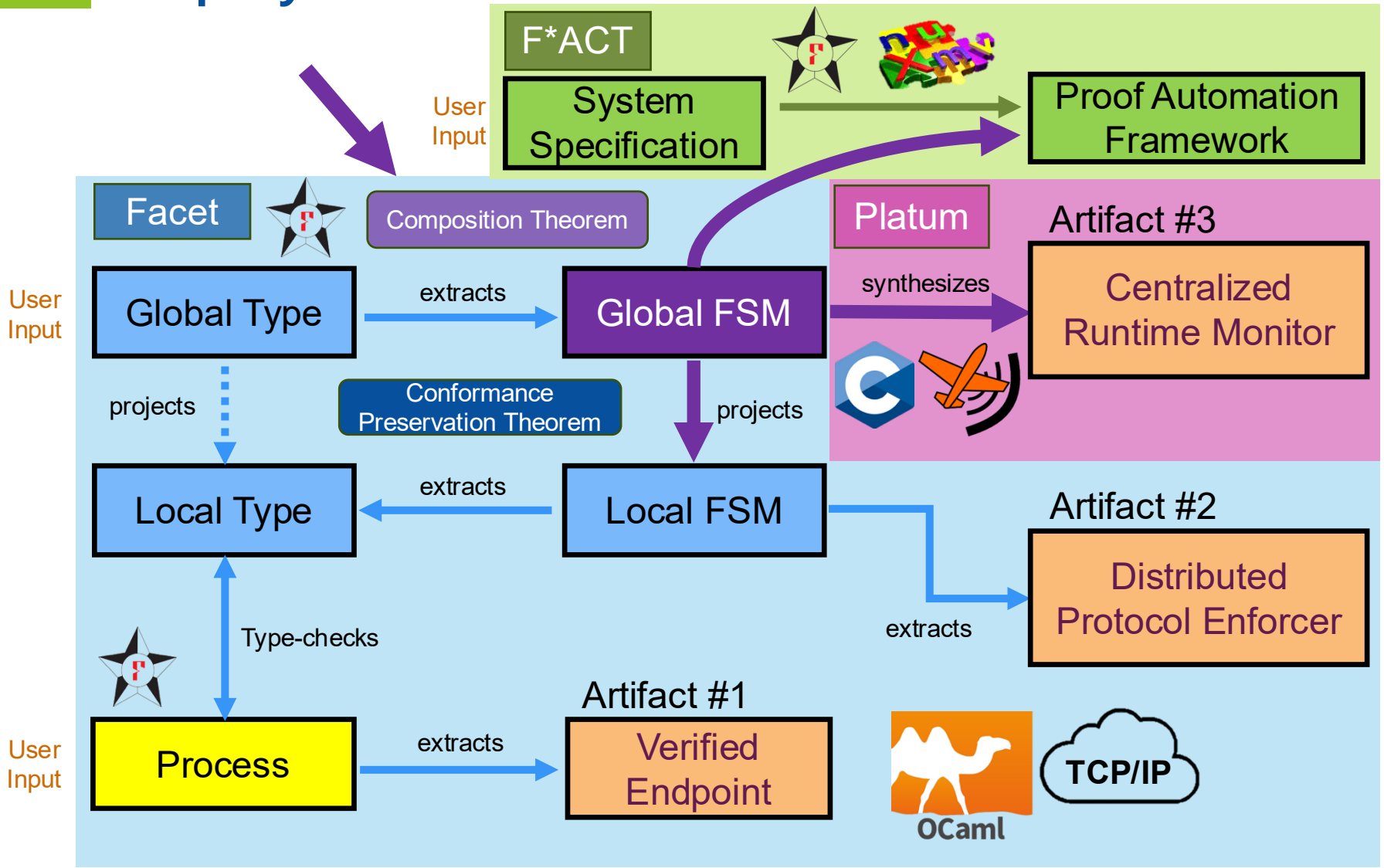


Proxy (PyMAVLink)
- Serialization

+

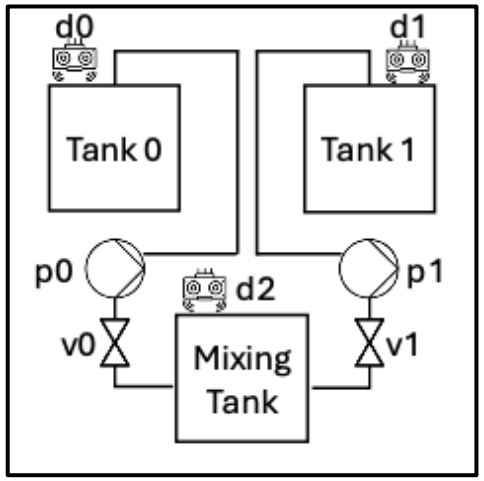
Runtime Monitor
- Flat FSM

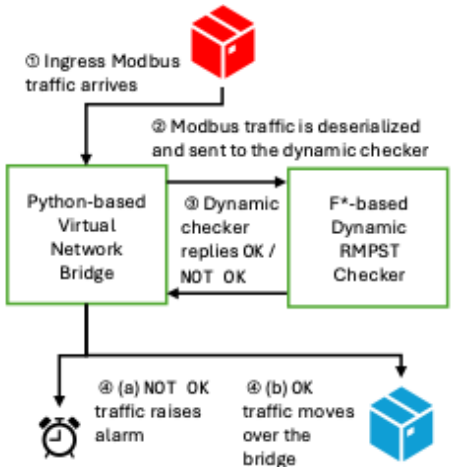
A Single Certified Semantic Model Shared Across All Generated Deployment Artifacts

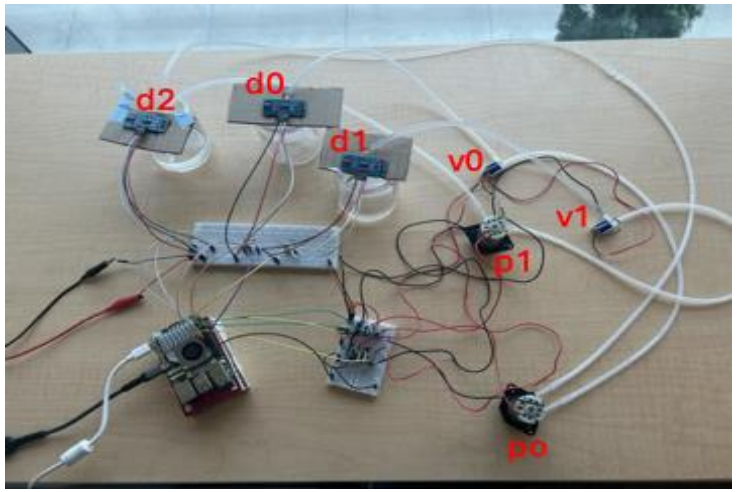


Previous Deployments

Modbus-based industrial control system [4]

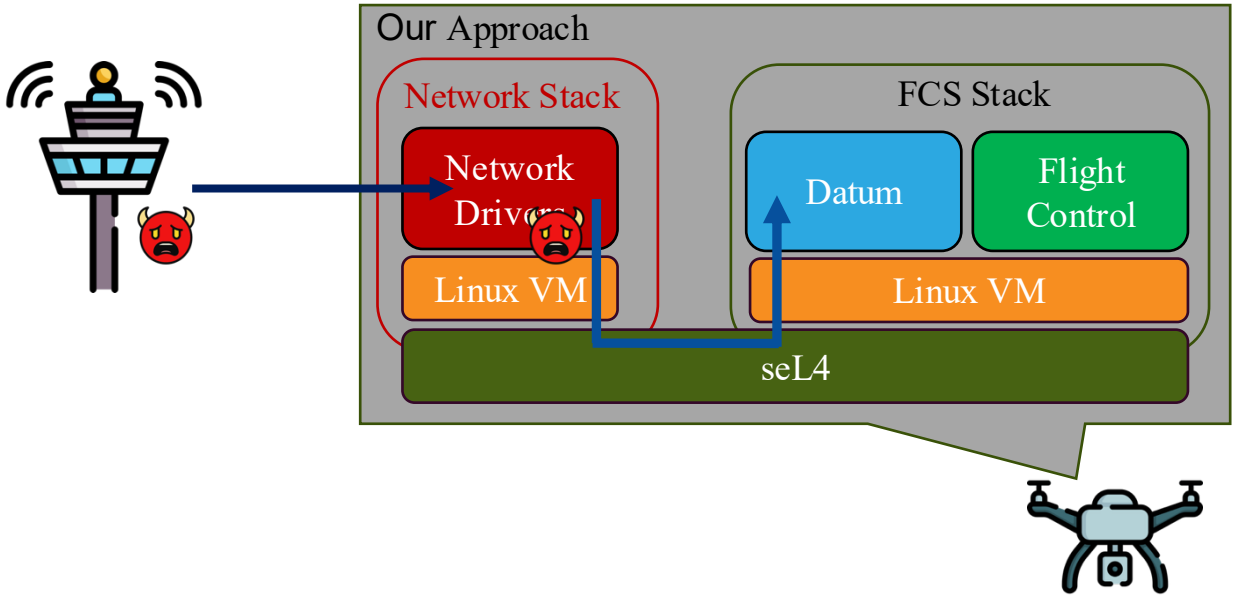






4- Max Taylor and Arthur Amorim. Securing Modbus-Based Industrial Control Systems with Refined Multiparty Session Types. VERDI'25

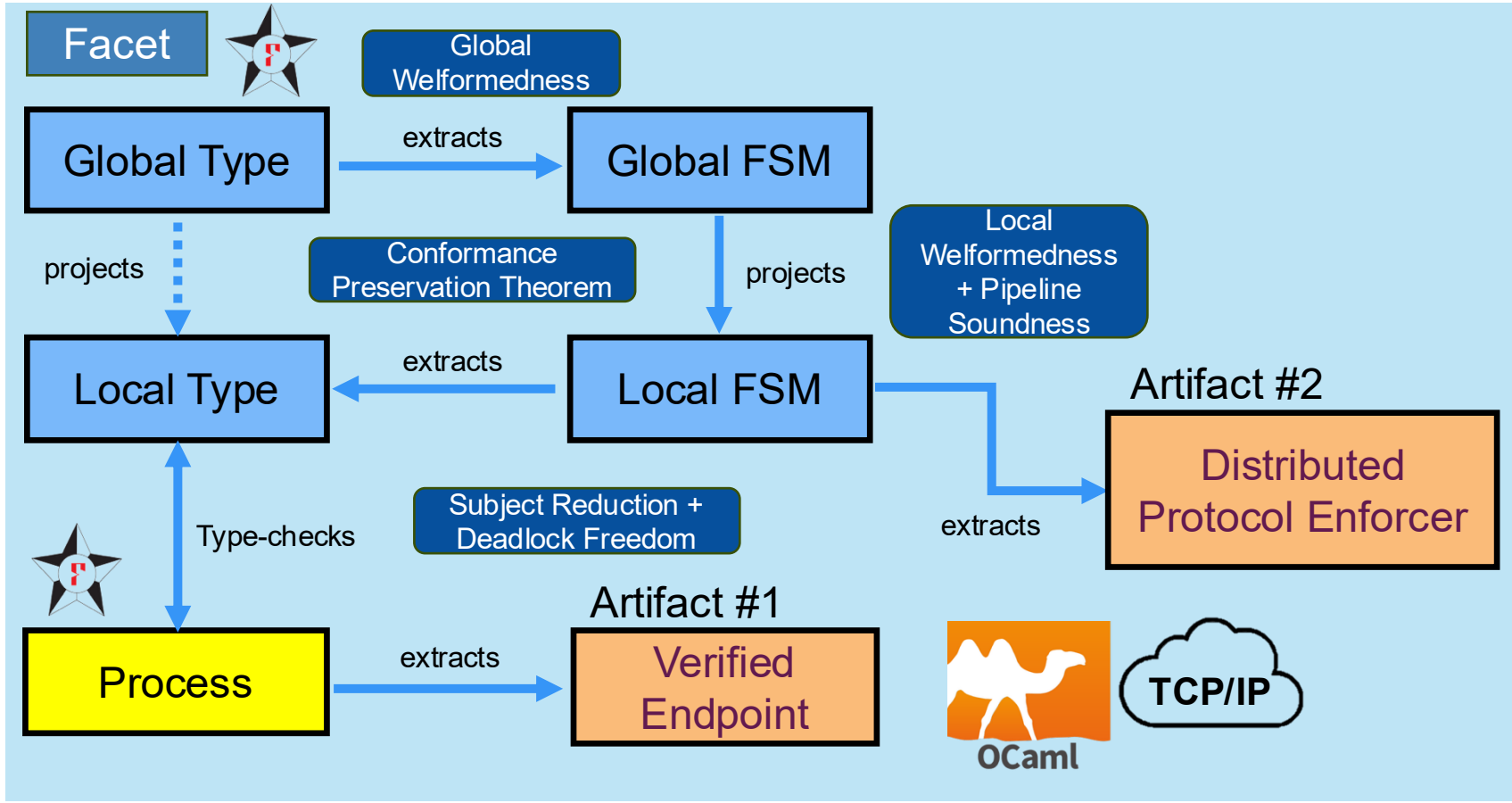
Integrated Datum(predecessor) with Sel4 to protect against stronger threat model.[5]



5- Arthur Amorim et al. UAV Resilience Against Stealthy Attacks. ICUAS'25

One Verified Model, Three Certified Artifacts

1- Semantic Gap

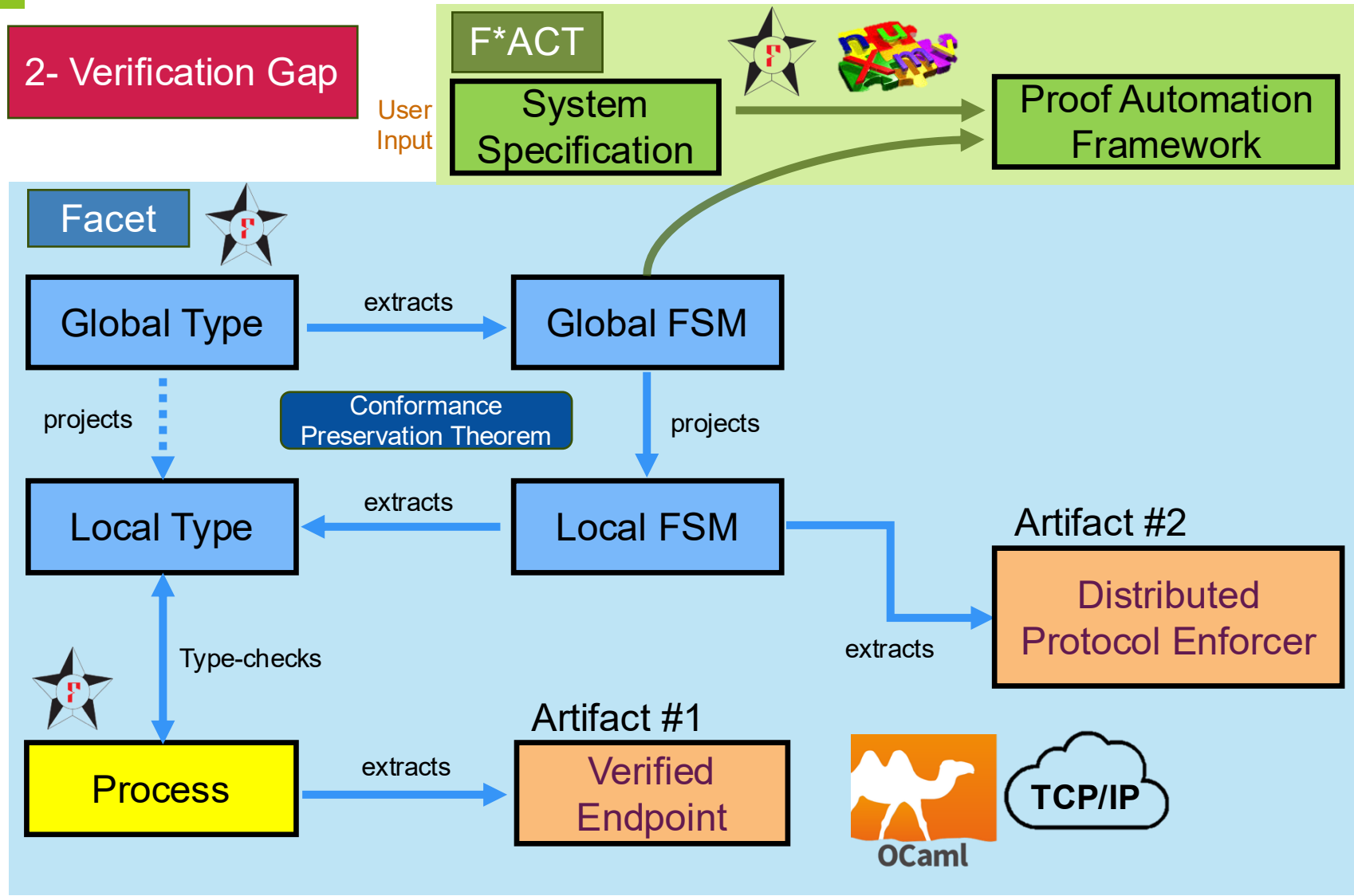


1- Semantic Gap

Facet: first mechanized RMPST metatheory; the proof and the runtime enforcer are the same object

1- Amorim et al. Facet: Mechanized Theory of Refined Multiparty Protocols via Computational Reflection. Under Review

One Verified Model, Three Certified Artifacts



1- Semantic Gap

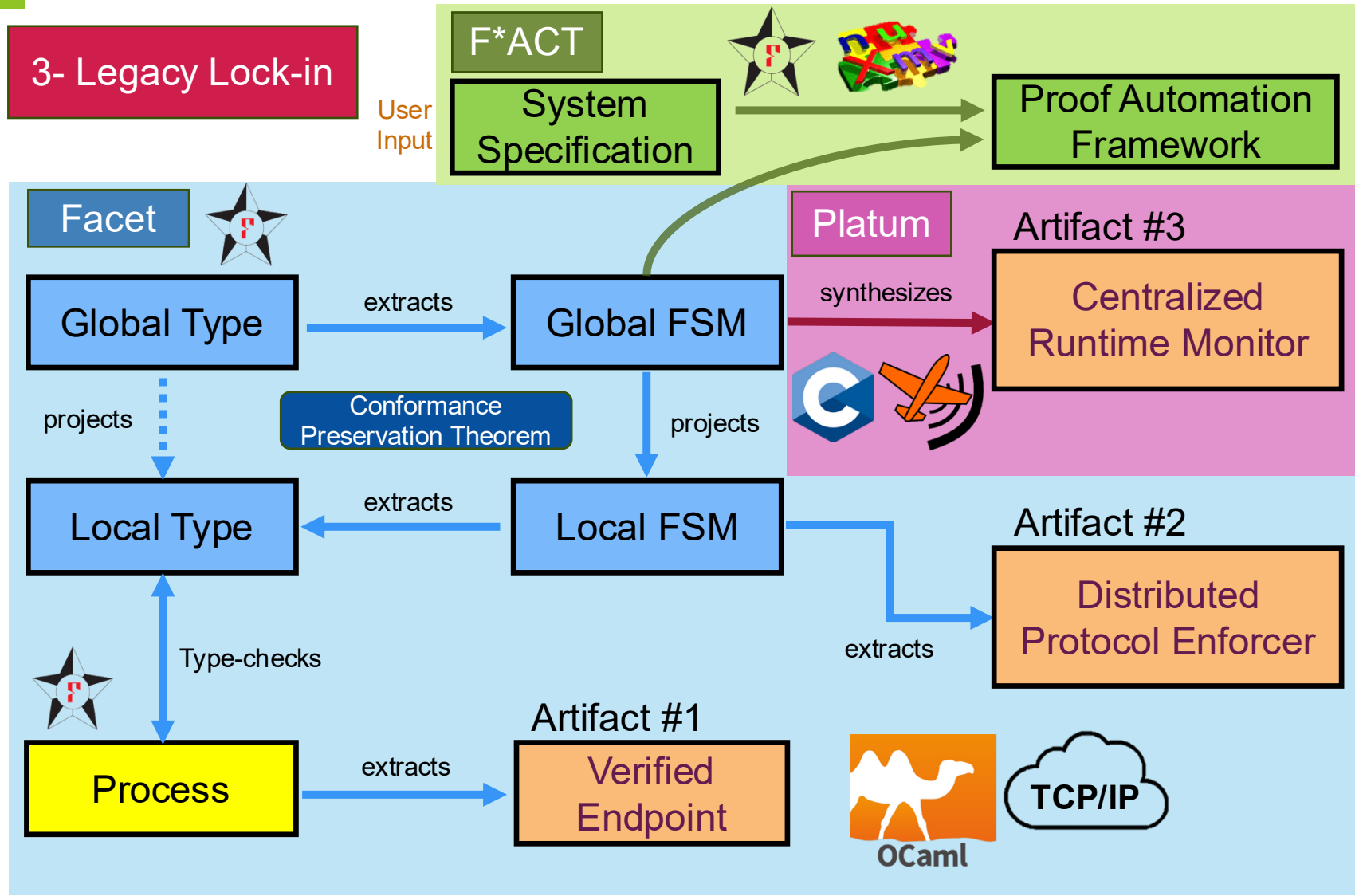
Facet: first mechanized RMPST metatheory; the proof and the runtime enforcer are the same object

2- Proof Celling

F*ACT: combines automated model checking with tactic-based certificate generation for complex protocols

2- Amorim et al. F*ACT: Formally Assured Communicating Types. In Preparation

One Verified Model, Three Certified Artifacts



1- Semantic Gap

Facet: first mechanized RMPST metatheory; the proof and the runtime enforcer are the same object

2- Proof Celling

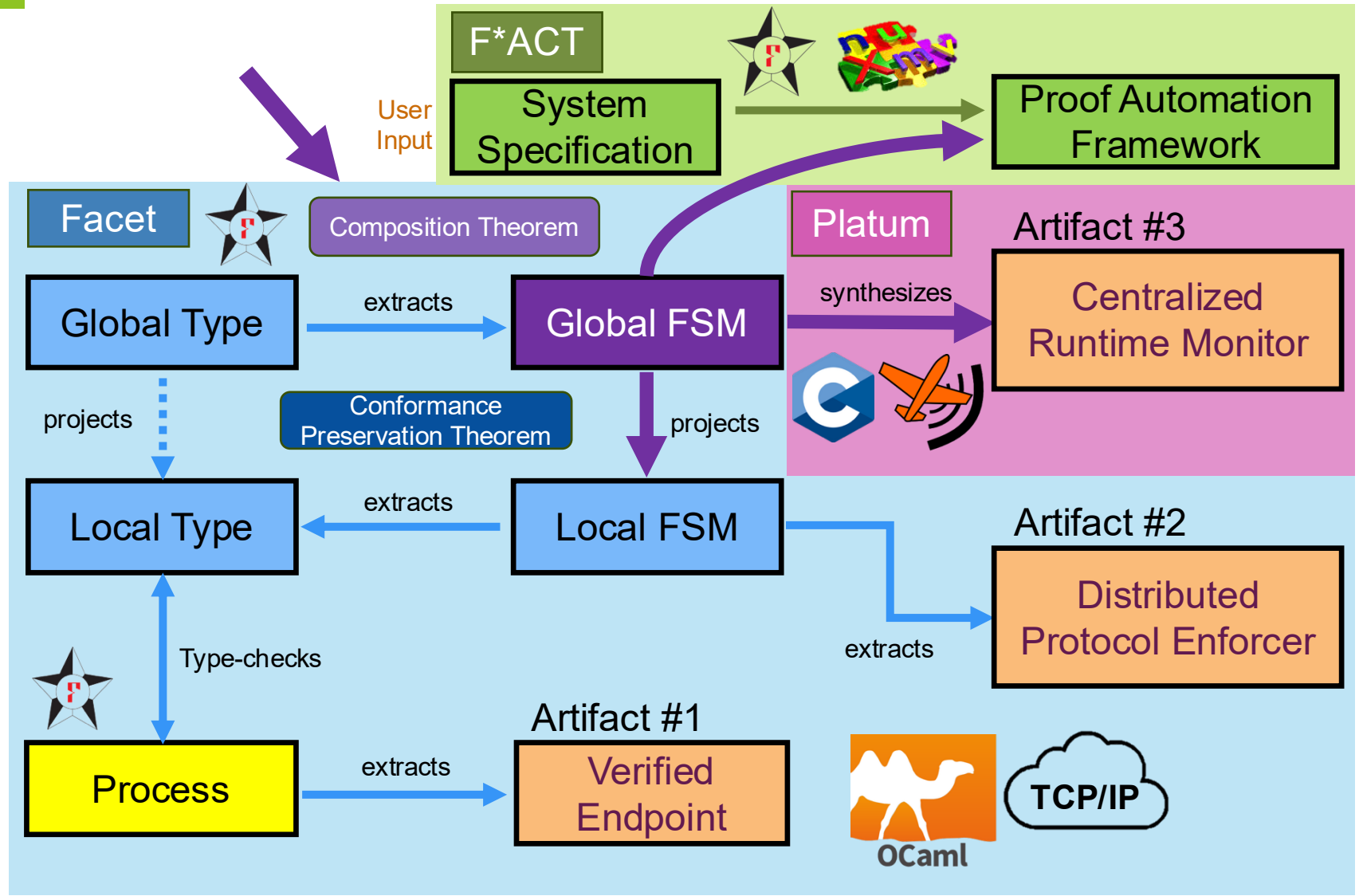
F*ACT: combines automated model checking with tactic-based certificate generation for complex protocols

3- Legacy Lock-in

Platum: synthesizes verified C monitors that retrofit any existing protocol without touching the endpoints

3- Amorim et al. From High-Level Types to Low-Level Monitors: Synthesizing Verified Runtime Checkers for MAVLink To appear in NFM'26

One Verified Model, Three Certified Artifacts



1- Semantic Gap

Facet: first mechanized RMPST metatheory; the proof and the runtime enforcer are the same object

2- Proof Celling

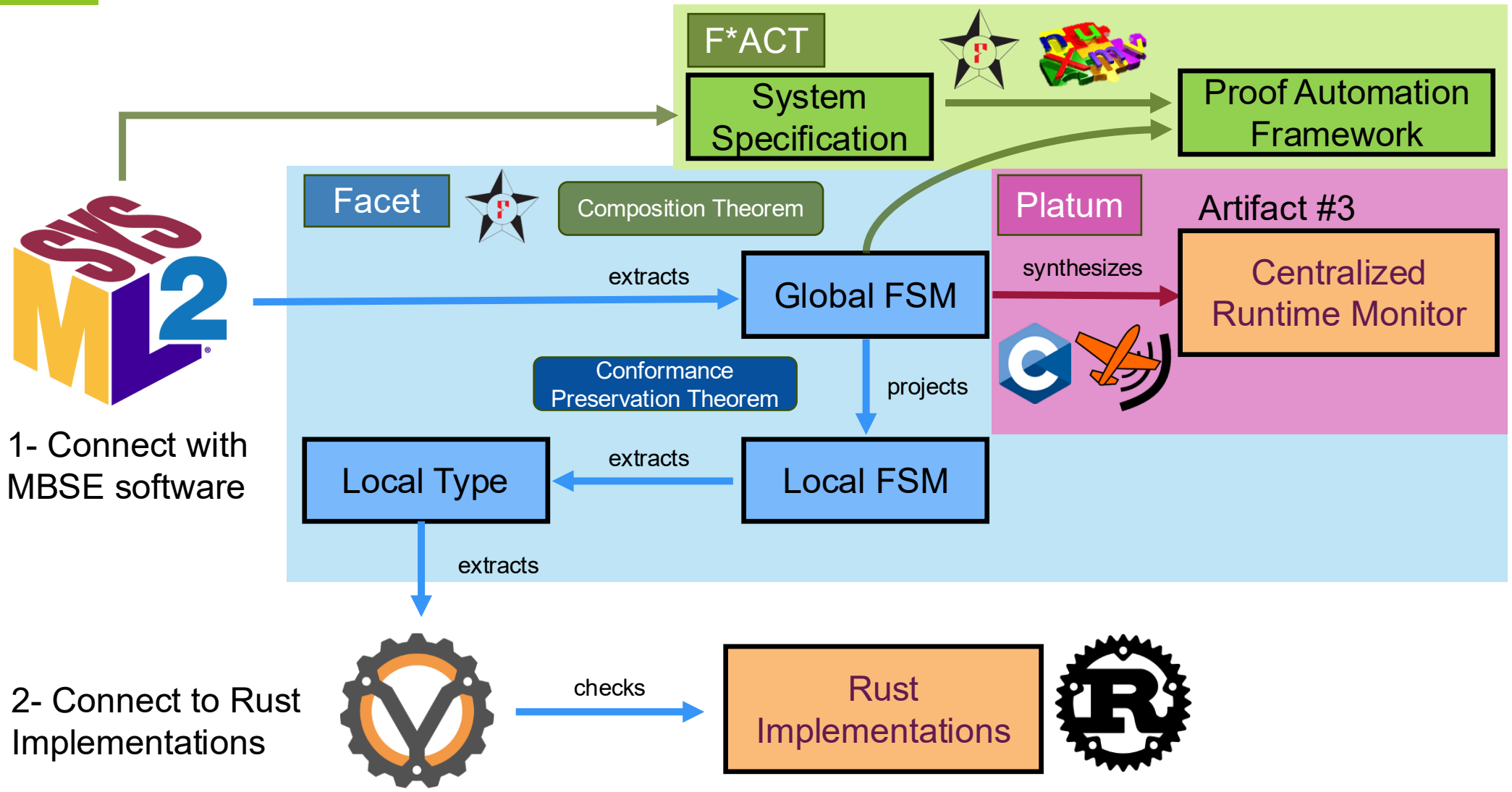
F*ACT: combines automated model checking with tactic-based certificate generation for complex protocols

3- Legacy Lock-in

Platum: synthesizes verified C monitors that retrofit any existing protocol without touching the endpoints

2- Amorim et al. F*ACT: Formally Assured Communicating Types. In Preparation

Future Directions



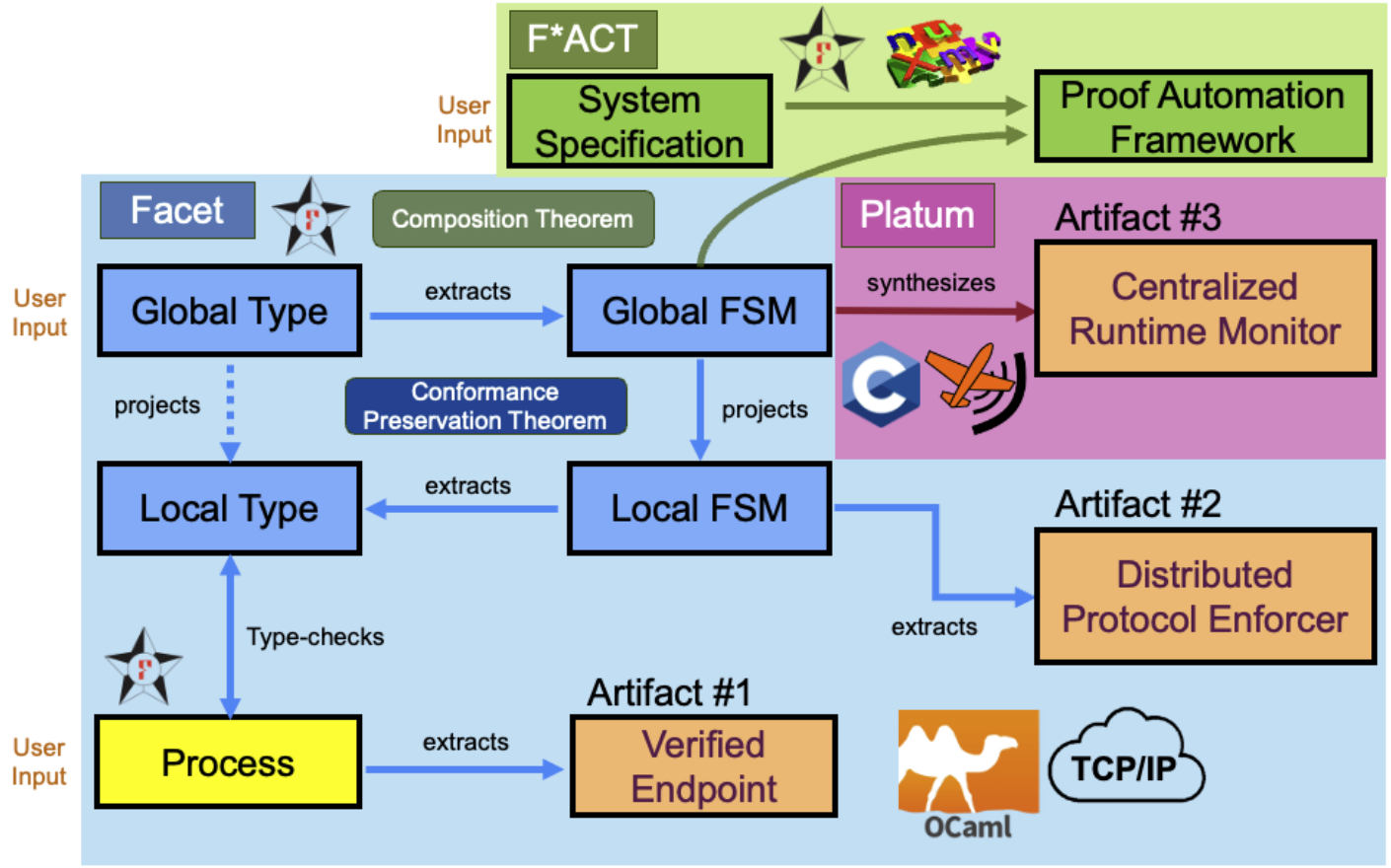
1- Connect with MBSE software

2- Connect to Rust Implementations

One Verified Model, Three Certified Artifacts

OPEN TO WORK

Arthur Amorim
arthur.amorim@ucf.edu
arthur.amorim@inl.gov
Website



1- Semantic Gap

Facet: first mechanized RMPST metatheory; the proof and the runtime enforcer are the same object

2- Verification gap

F*ACT: combines automated model checking with tactic-based certificate generation for complex protocols

3- Legacy Lock-in

Platum: synthesizes verified C monitors that retrofit any existing protocol without touching the endpoints

Why F*?

- Z3 automatically discharges proof obligations in its decision procedures
- Makes it practical to use operational semantics

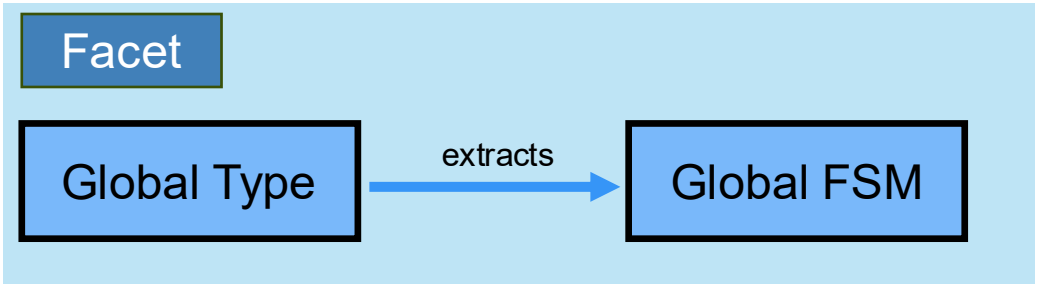
- Native, first-class refinements {x : T | φ(x)}
- RMPSTs map directly to F* refinement types

$$\begin{aligned}
&\textbf{Server} \to \textbf{Auth} : \texttt{PIN}\,(expected_code : \mathbb{Z}\,\{expected_code \geq 0 \wedge expected_code \leq 999999\}) \\
&\mu.\mathbf{T}(count : \mathbb{Z}\{0 < count \wedge count \leq 5\})\ 1 \\
&\quad \textbf{Auth} \to \textbf{User} : \texttt{RequestAuth}\,(_ : \texttt{unit}) \\
&\quad \textbf{User} \to \textbf{Auth} : \texttt{SubmitCode}\,(submitted : \mathbb{Z}\,\{submitted \geq 0 \wedge submitted \leq 999999\}) \\
&\quad \textbf{Auth} \to \textbf{User} : \begin{cases} \texttt{Success}\,(_ : \texttt{unit}\,\{submitted = expected_code\}).\mathbf{End} \\ \texttt{Failed}\,(_ : \texttt{unit}\,\{submitted \neq expected_code \wedge count < 5\}).\ \mathbf{T}(count + 1) \\ \texttt{Locked}\,(_ : \texttt{unit}\,\{submitted \neq expected_code \wedge count = 5\}).\mathbf{End} \end{cases}
\end{aligned}$$

- Principled extraction pipeline to high-performance languages
- Project Everest: Verified TLS 1.3 in production

Idaho National Laboratory

The Engine That Makes This Tractable: Shallow-to-Deep Embedding Extraction



Shallow embedding:

- Easy to write (uses host language)
- hard to prove (must reason about all of F*)

```
(0 --> 1) [ Option ”Counter" (fun (count : int
{count > 0 &&  count<= 5})
```

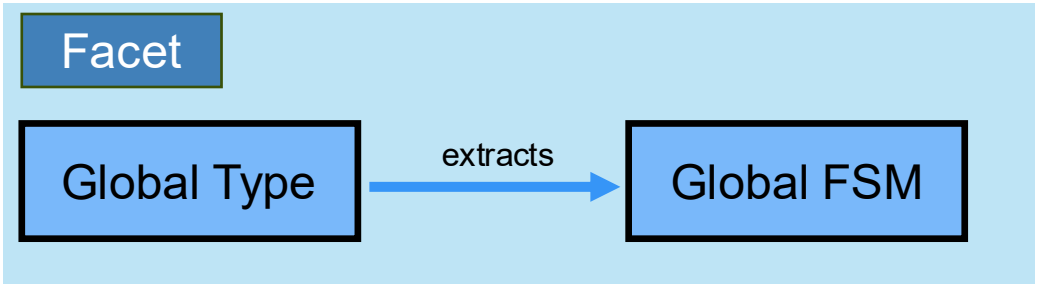
VS

Deep embedding:

-Hard to write (explicit syntax trees),
-Easy to prove (reason only about DSL)

```
Mkmu_info 1
    [Mkvar "count" KindInt]
    (GAnd (GGt (TVar "count") (TIntLit 0))
(GLe (TVar "count") (TIntLit 5)))
```

The Engine That Makes This Tractable: Shallow-to-Deep Embedding Extraction



Users write :

Meta-F*

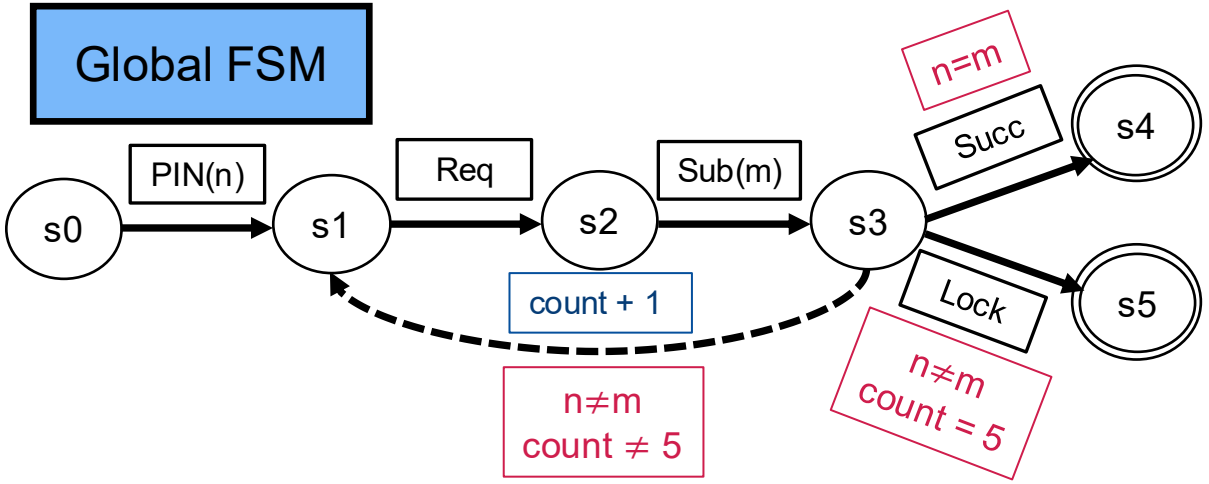
Users Prove:

```
(0 --> 1) [ Option ”Counter" (fun (count : int
{count > 0 &&  count<= 5})
```

```
Mkmu_info 1
    [Mkvar "count" KindInt]
    (GAnd (GGt (TVar "count") (TIntLit 0))
(GLe (TVar "count") (TIntLit 5)))
```

F*ACT Proves Application-Specific Safety Without State Explosion

Global FSM



$$\textbf{Server} \rightarrow \textbf{Auth} : \texttt{PIN}\ (expected\_code : \mathbb{Z}\ \{expected\_code \geq 0 \wedge expected\_code \leq 999999\})$$

$$\mu.\mathbf{T}(count : \mathbb{Z}\{0 < count \wedge count \leq 5\})\ 1$$

$$\textbf{Auth} \rightarrow \textbf{User} : \texttt{RequestAuth}\ (\_ : \texttt{unit})$$

$$\textbf{User} \rightarrow \textbf{Auth} : \texttt{SubmitCode}\ (submitted : \mathbb{Z}\ \{submitted \geq 0 \wedge submitted \leq 999999\})$$

$$\textbf{Auth} \rightarrow \textbf{User} : \begin{cases} \texttt{Success}\ (\_ : \texttt{unit}\ \{submitted = expected\_code\}).\mathbf{End} \\ \texttt{Failed}\ (\_ : \texttt{unit}\ \{submitted \neq expected\_code \wedge count < 5\}).\ \mathbf{T}(count + 1) \\ \texttt{Locked}\ (\_ : \texttt{unit}\ \{submitted \neq expected\_code \wedge count = 5\}).\mathbf{End} \end{cases}$$

System Specification

Safety Properties + System Semantics

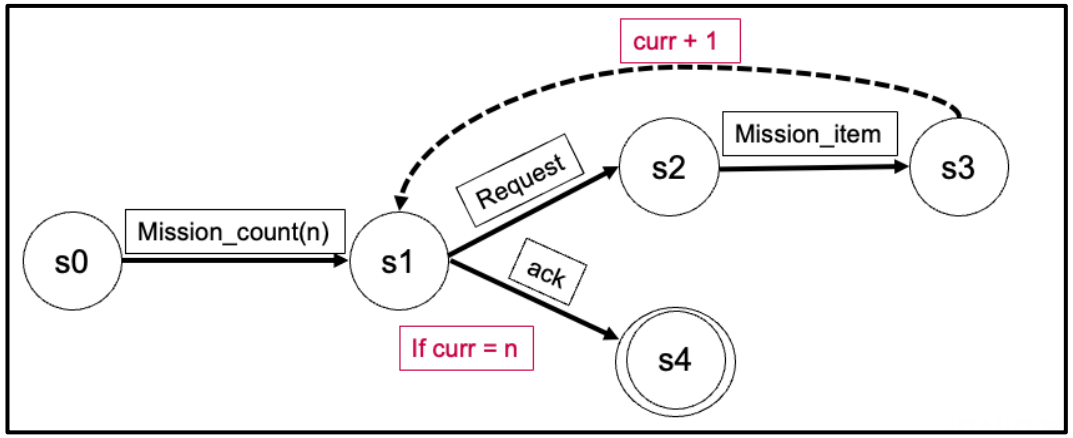
- ”The user must input the exact pin in order to succeed”
- “ After 5 tries with wrong input, the account must be locked”

Operational Semantics

$$\boldsymbol{State} + \boldsymbol{Command} \rightarrow \boldsymbol{State}$$

F*ACT Proves Application-Specific Safety Without State Explosion

Global FSM



System Specification

Safety Properties + System Semantics

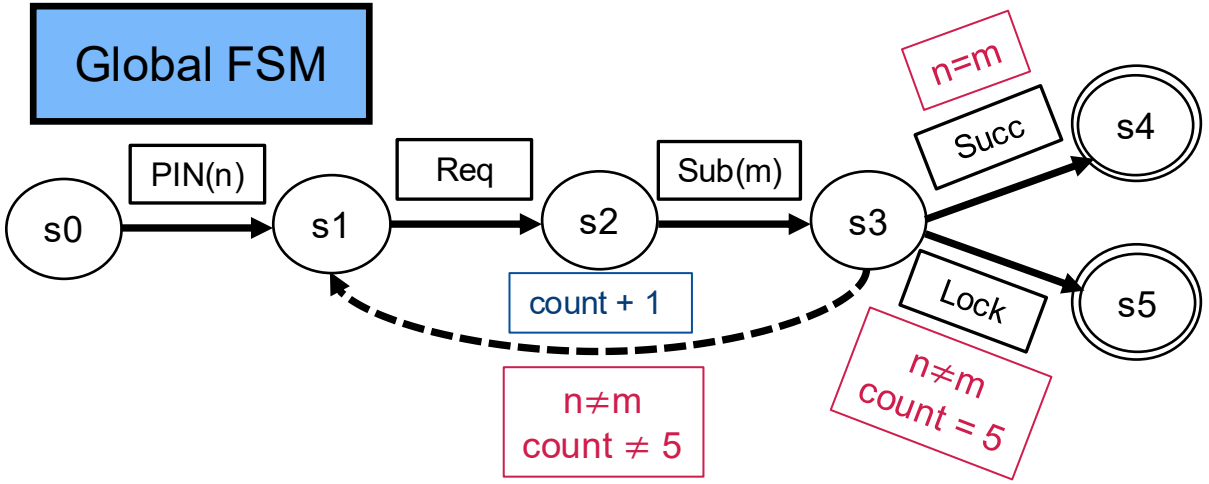
- ”The GCS must send all mission in order without skips”
- “ An ack(success) message can only be sent when *curr* = *n*”

Operational Semantics

$$(\mathrm{n} = 2; \mathrm{m} = 3) \; + \; \text{Succeed} \; \rightarrow \; \boldsymbol{WRONG}$$

Safety Condition: following a protocol must never lead the system to a WRONG state

F*ACT Proves Application-Specific Safety Without State Explosion



System Specification

- "The user must input the exact pin in order to succeed"
- " After 5 tries with wrong input, the account must be locked"

Operational Semantics

$$State + Command \rightarrow State$$

$$(n = 2; m = 3) + Succeed \rightarrow WRONG$$

Model Checking

- Automated
- Counterexample generation
- State explosion

Use it when possible

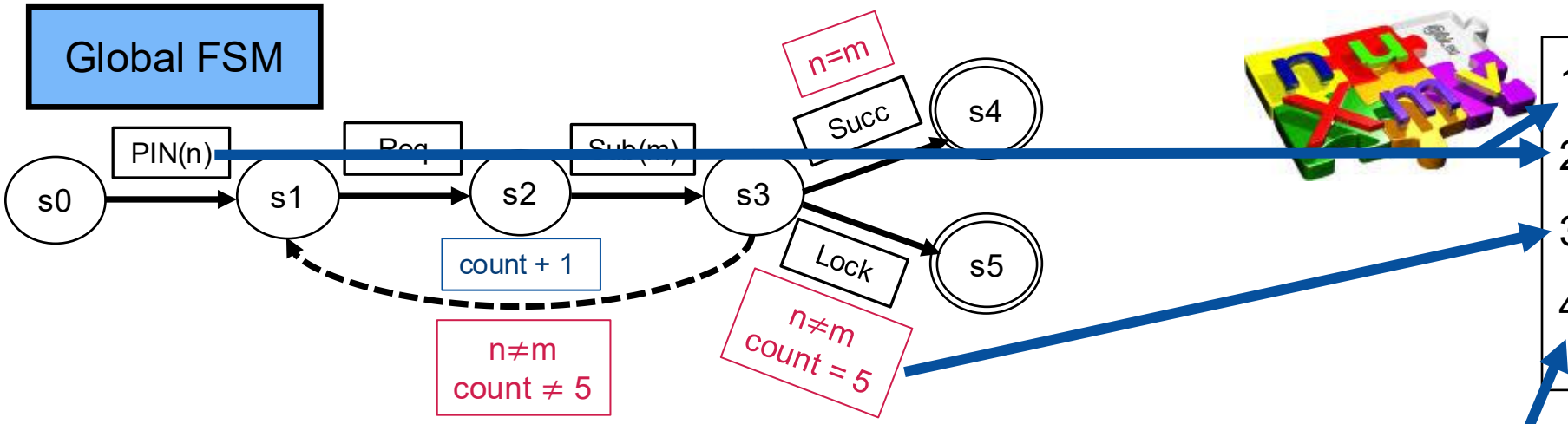
Theorem proving

- Proof scalability
- Handles infinite state protocols
- Complexity (interactive proofs)

Use it when model checking fails

Safety Condition: following a protocol must never lead the system to a WRONG state

F*ACT Proves Application-Specific Safety Without State Explosion



Model Checking

1- Variables
2- Initial values
3- States (variable updates)
4- Safety properties

Theorem proving

- Proof scalability
- Handles infinite state protocols

- Complexity (interactive proofs)

Use it when model checking fails

System Specification

- ”The user must input the exact pin in order to succeed”
- “ After 5 tries with wrong input, the account must be locked”

Operational Semantics

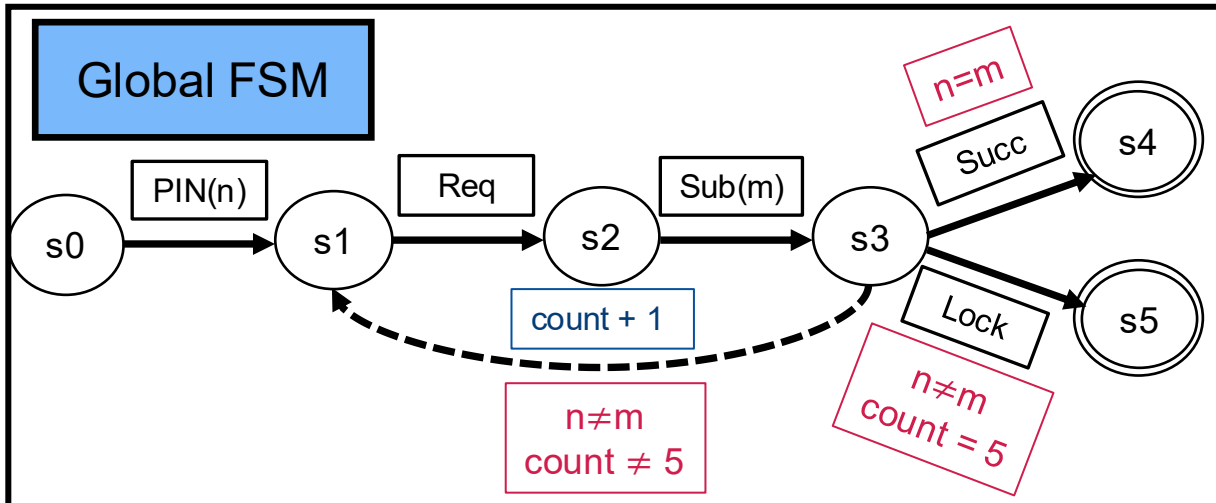
$$\boldsymbol{State} + \boldsymbol{Command} \rightarrow \boldsymbol{State}$$

$$(\mathrm{n} = 2; \mathrm{m} = 3) + \mathrm{Succeed} \rightarrow \boldsymbol{WRONG}$$

Safety Condition: following a protocol must never lead the system to a WRONG state

F*ACT Proves Application-Specific Safety Without State Explosion

Global FSM



System Specification

- "The user must input the exact pin in order to succeed"
- " After 5 tries with wrong input, the account must be locked"

Operational Semantics

$$State + Command \rightarrow State$$

$$(n = 2; m = 3) + Succeed \rightarrow WRONG$$

Meta-F*

Model Checking

1- Variables

2- Initial values

3- States (variable updates)

4- Safety properties

Theorem proving

- Proof scalability
- Handles infinite state protocols

- Complexity (interactive proofs)

Use it when model checking fails

Safety Condition: following a protocol must never lead the system to a WRONG state