IEEE S&P LangSec 2025
Work In Progress
May 15th, 2025

From Legacy to Verified Parsers with AI

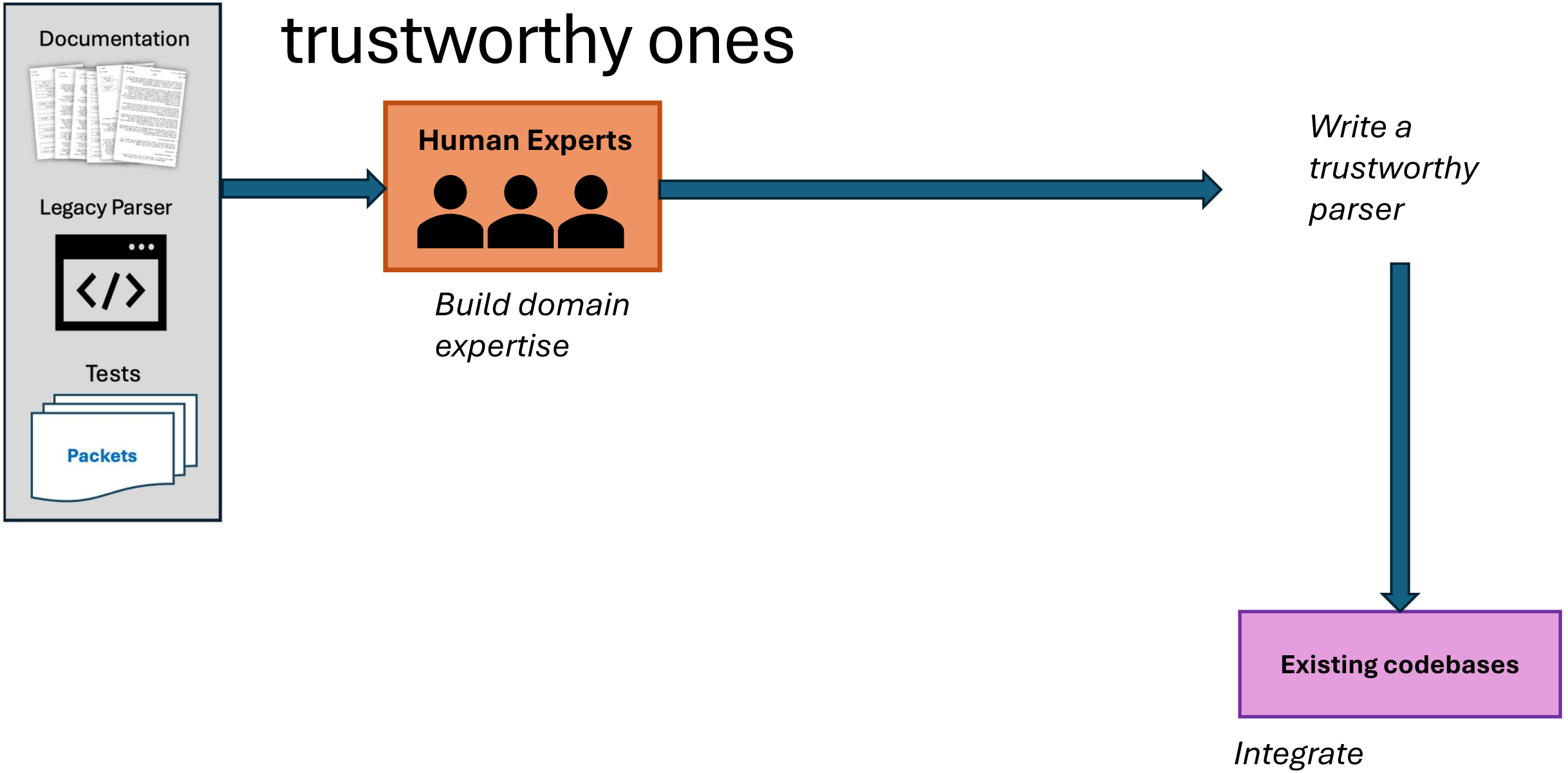
Tahina Ramananandro
RiSE – Microsoft Research

In collaboration with:
Mingwei Zheng (Purdue University)

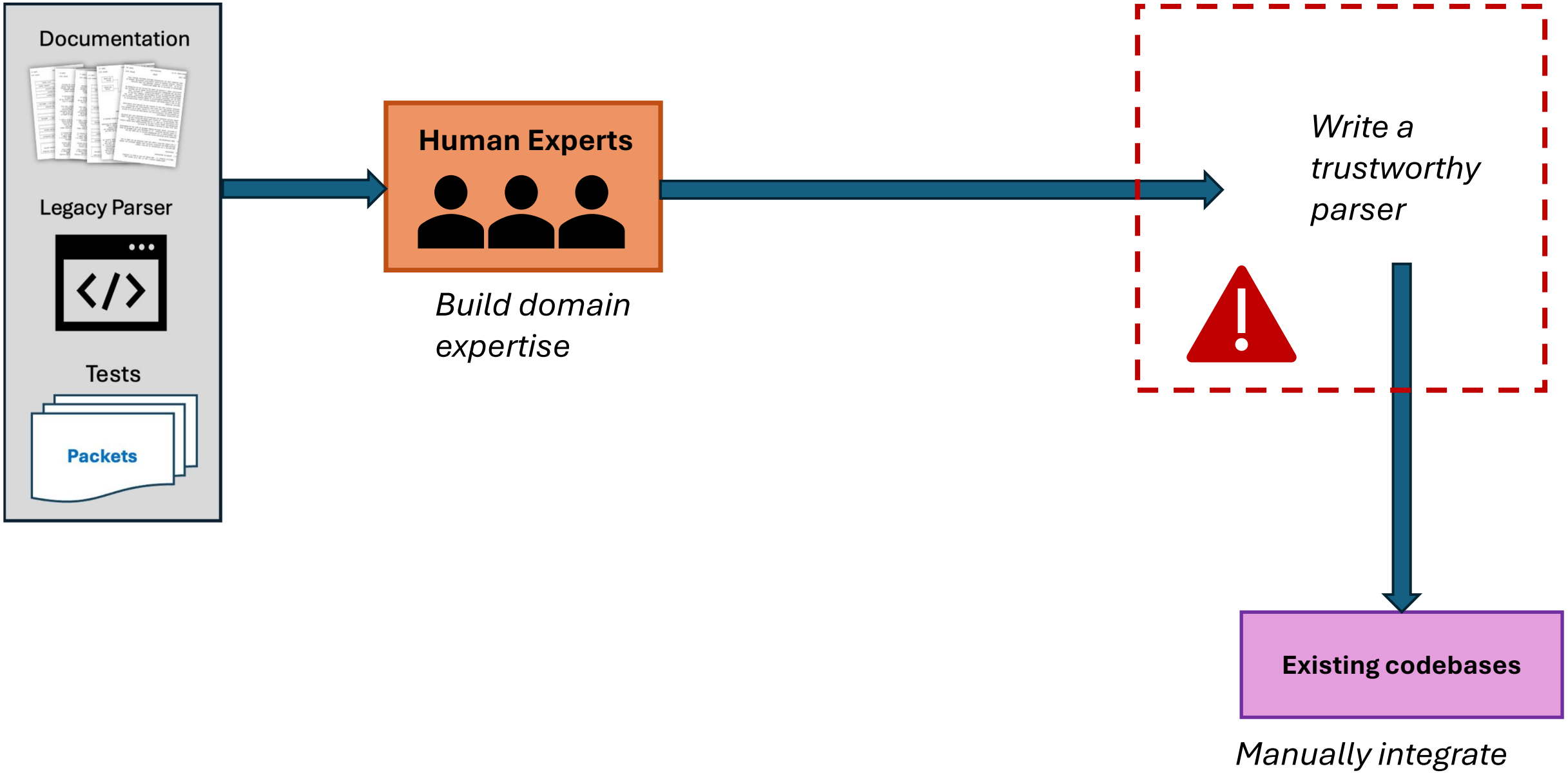
Sarah Fakhoury, Nikhil Swamy, Shuvendu Lahiri, Markus Kuppe
(RiSE – Microsoft Research)

Microsoft internal product teams

Our Goal: replace existing parsers with trustworthy ones

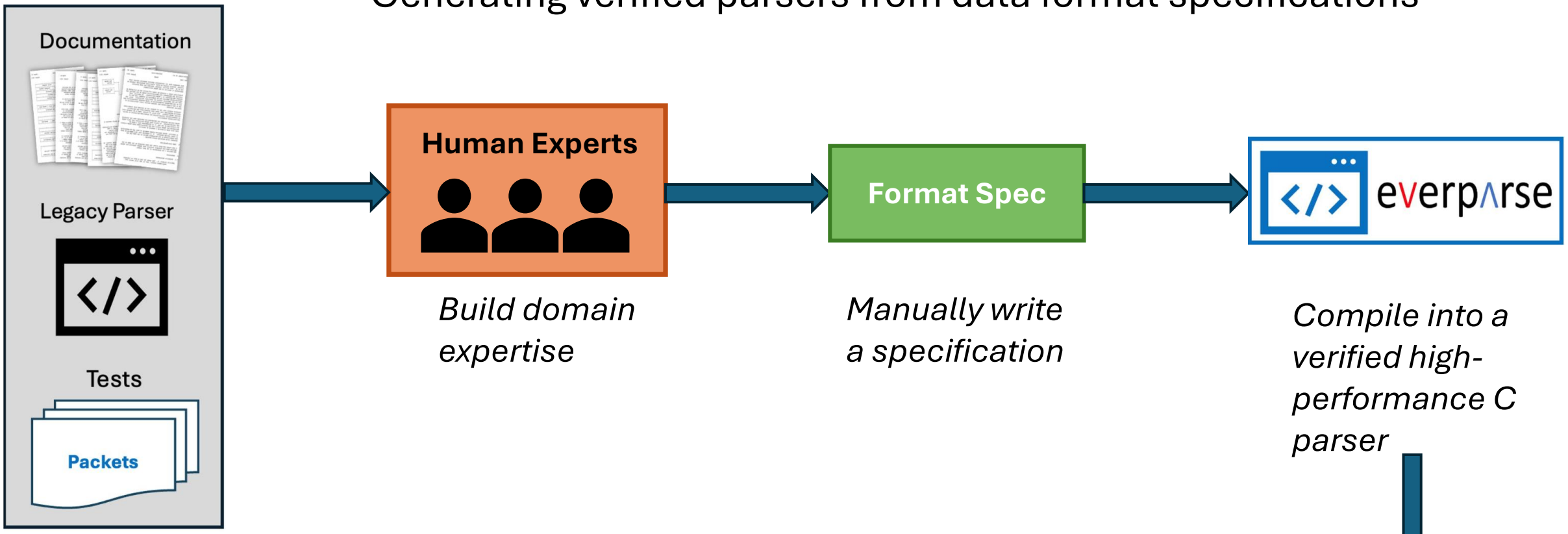


Challenge 1: Get a trustworthy parser



Layer 1: EverParse

Generating verified parsers from data format specifications



Build domain expertise

Manually write a specification

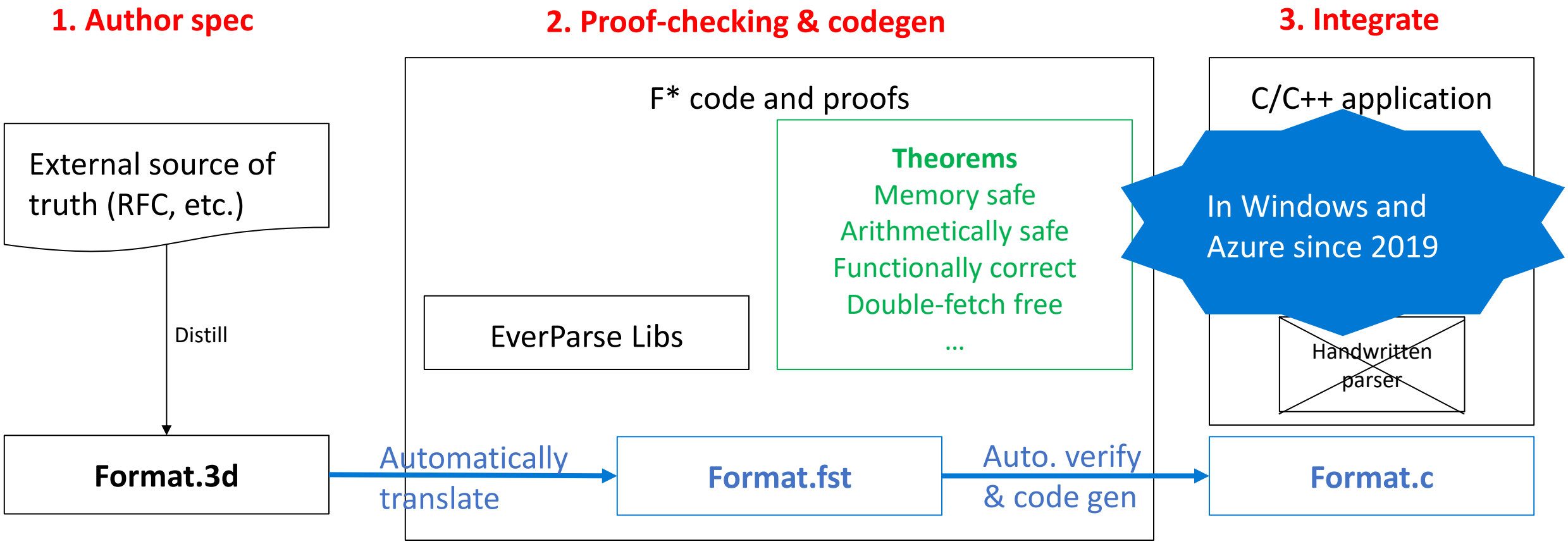
Compile into a verified high-performance C parser

Manually integrate

T. Ramananandro et al. *EverParse: Verified Secure Zero-Copy Parsers for Authenticated Message Formats.* USENIX Security 2019

N. Swamy et al. *Hardening Attack Surfaces with Formally Proven Binary Format Parsers.* PLDI 2022

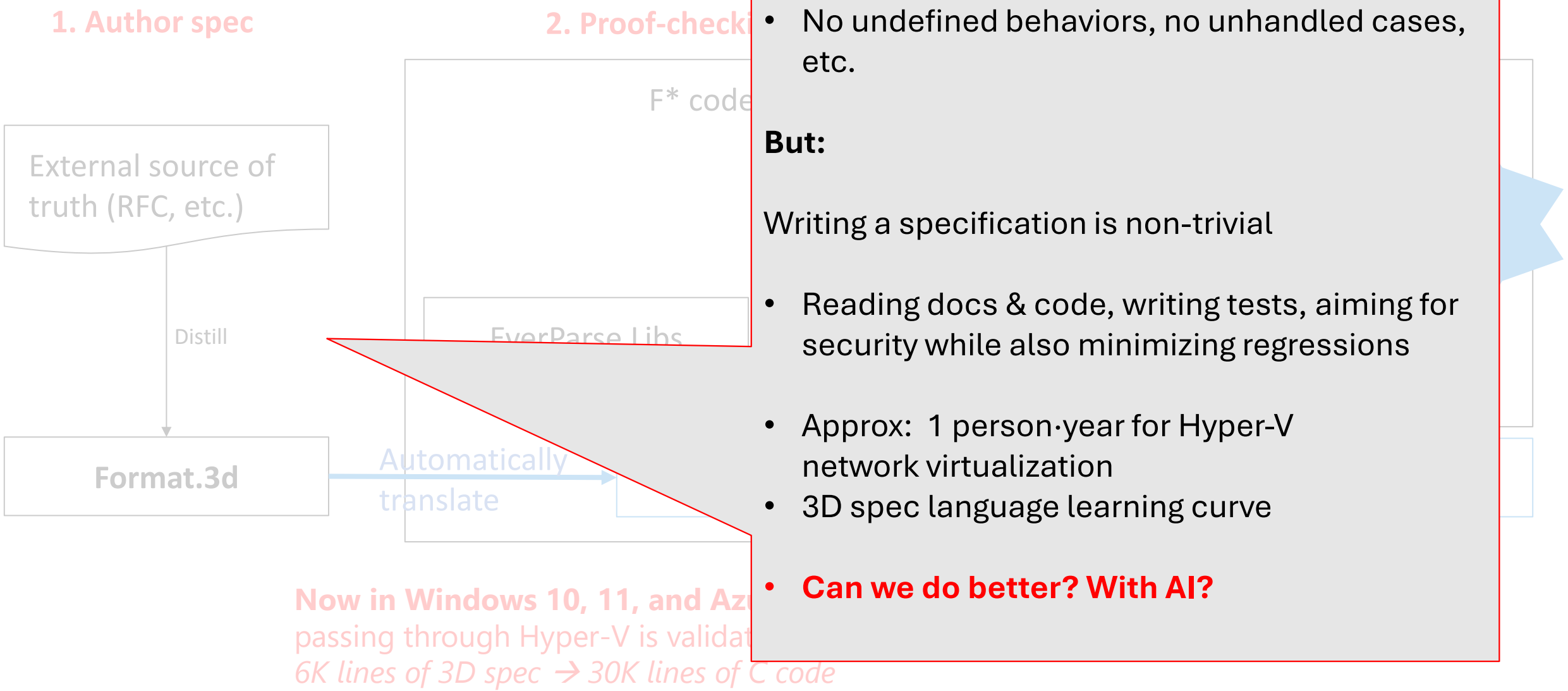
Layer 1: EverParse



Now in Windows 10, 11, and Azure Cloud: Every network packet passing through Hyper-V is validated by EverParse formally verified code

6K lines of 3D spec → 30K lines of C code

Layer 1: EverParse



With a formal spec language, fewer possibilities to make mistakes in spec

- No undefined behaviors, no unhandled cases, etc.

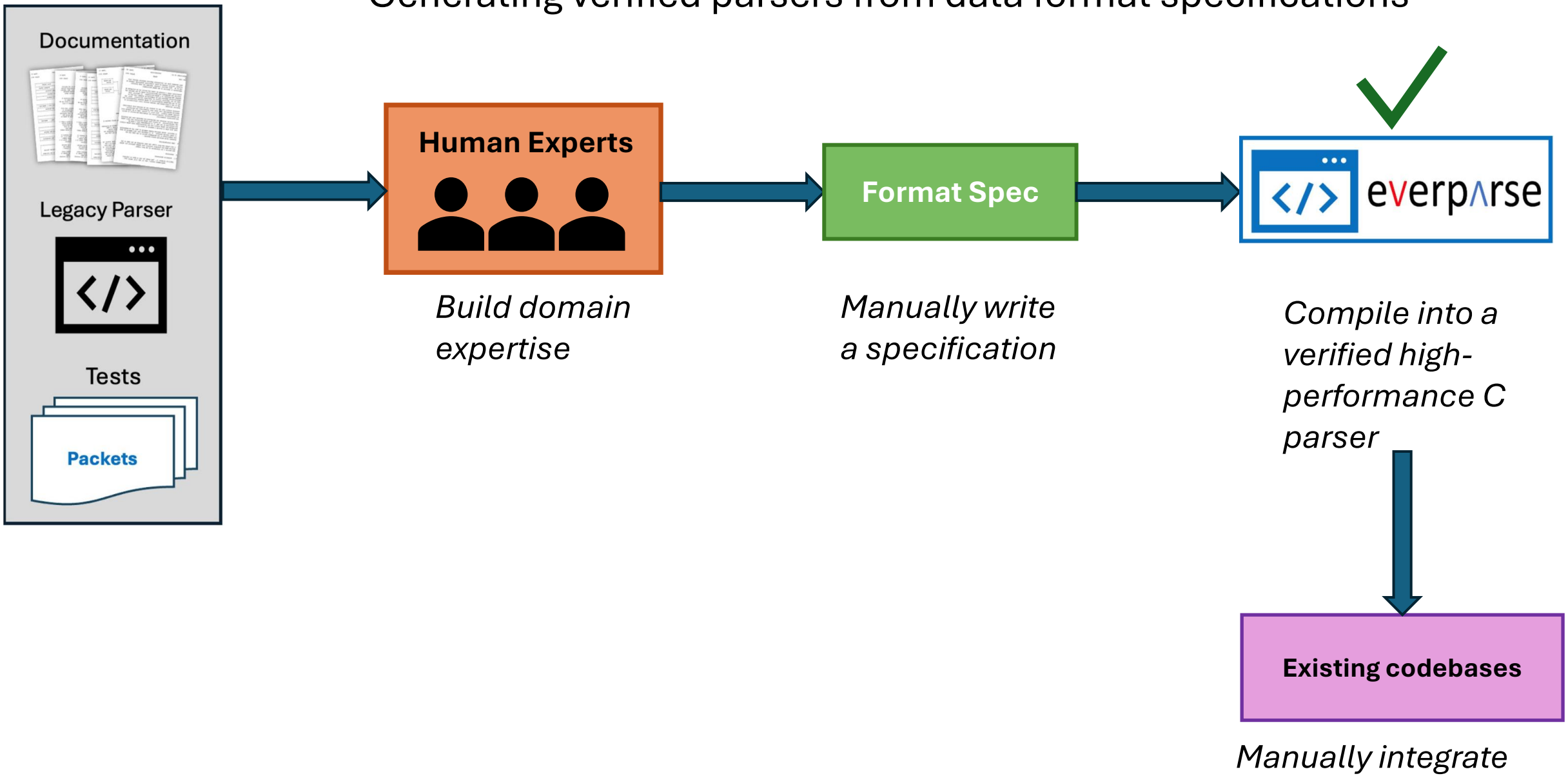
But:

Writing a specification is non-trivial

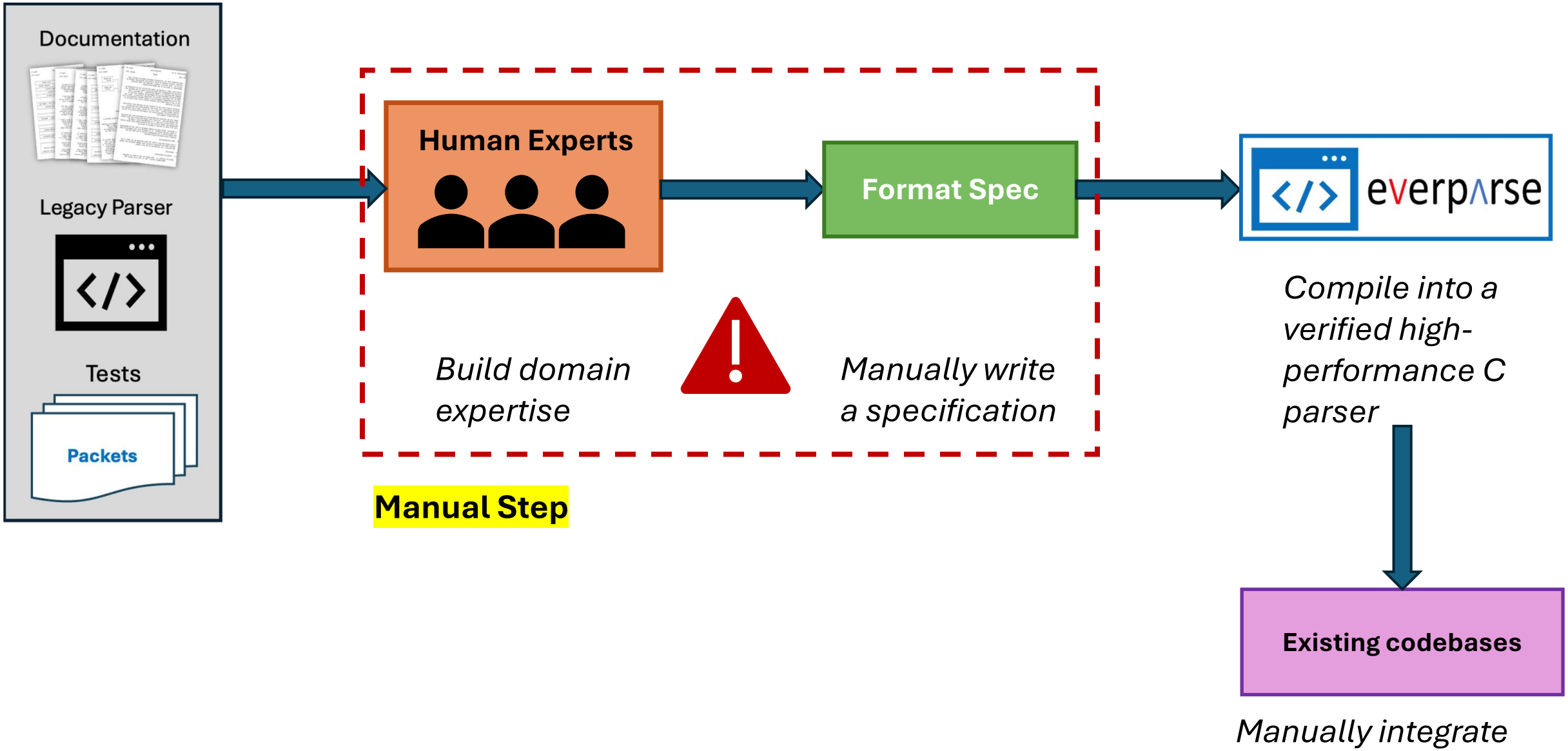
- Reading docs & code, writing tests, aiming for security while also minimizing regressions
- Approx: 1 person·year for Hyper-V network virtualization
- 3D spec language learning curve
- **Can we do better? With AI?**

Layer 1: EverParse

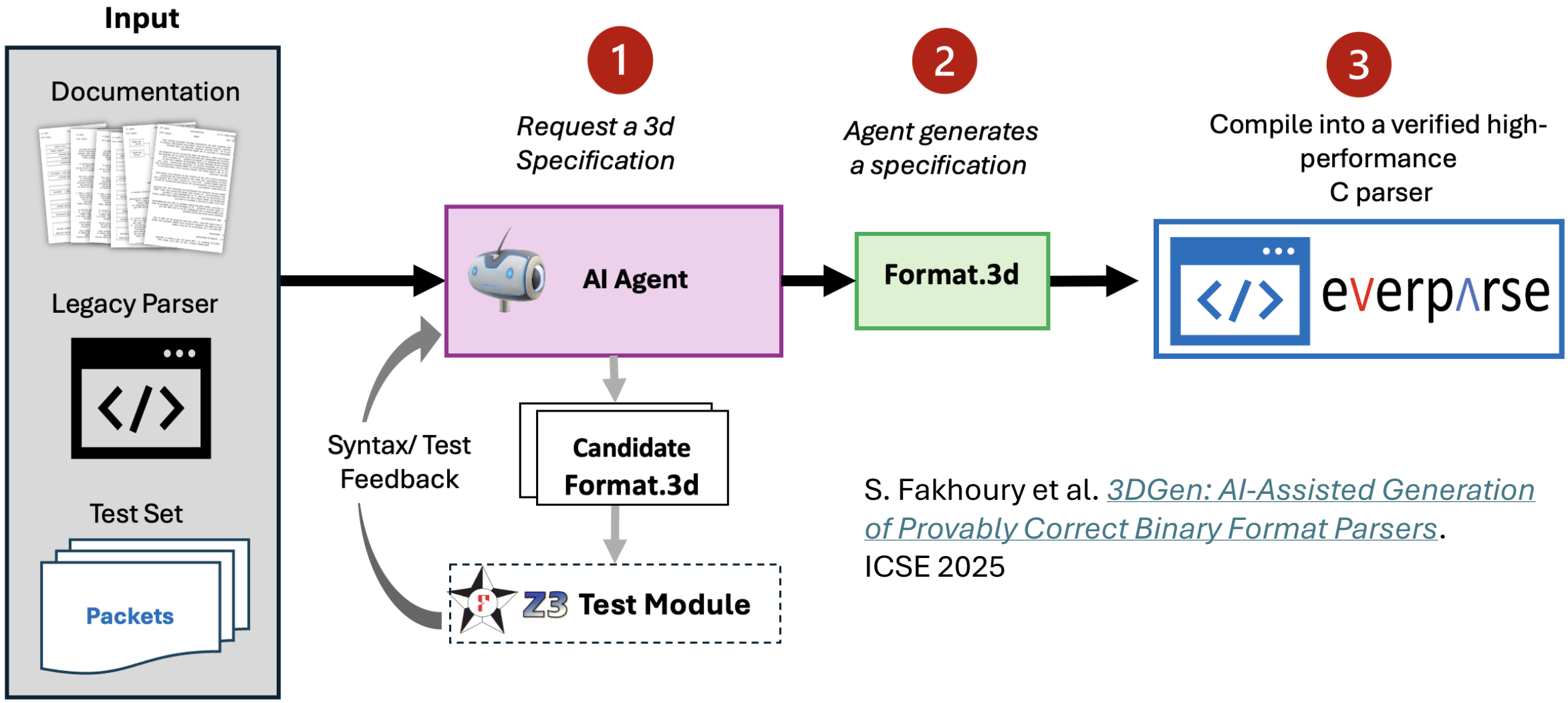
Generating verified parsers from data format specifications



Challenge 2: Automating spec writing



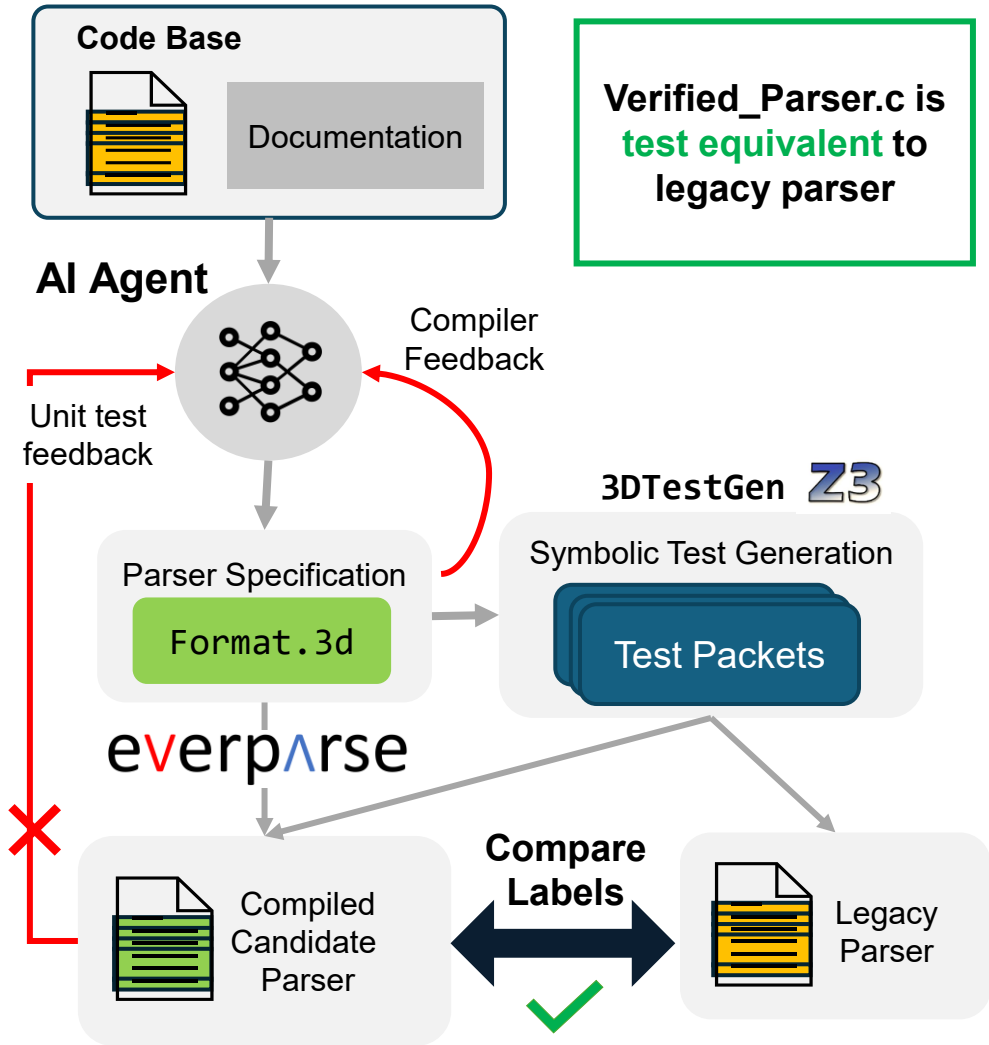
Layer 2: 3DGen
AI-Assisted Generation of Data Format Specifications



S. Fakhoury et al. *3DGen: AI-Assisted Generation of Provably Correct Binary Format Parsers*.
ICSE 2025

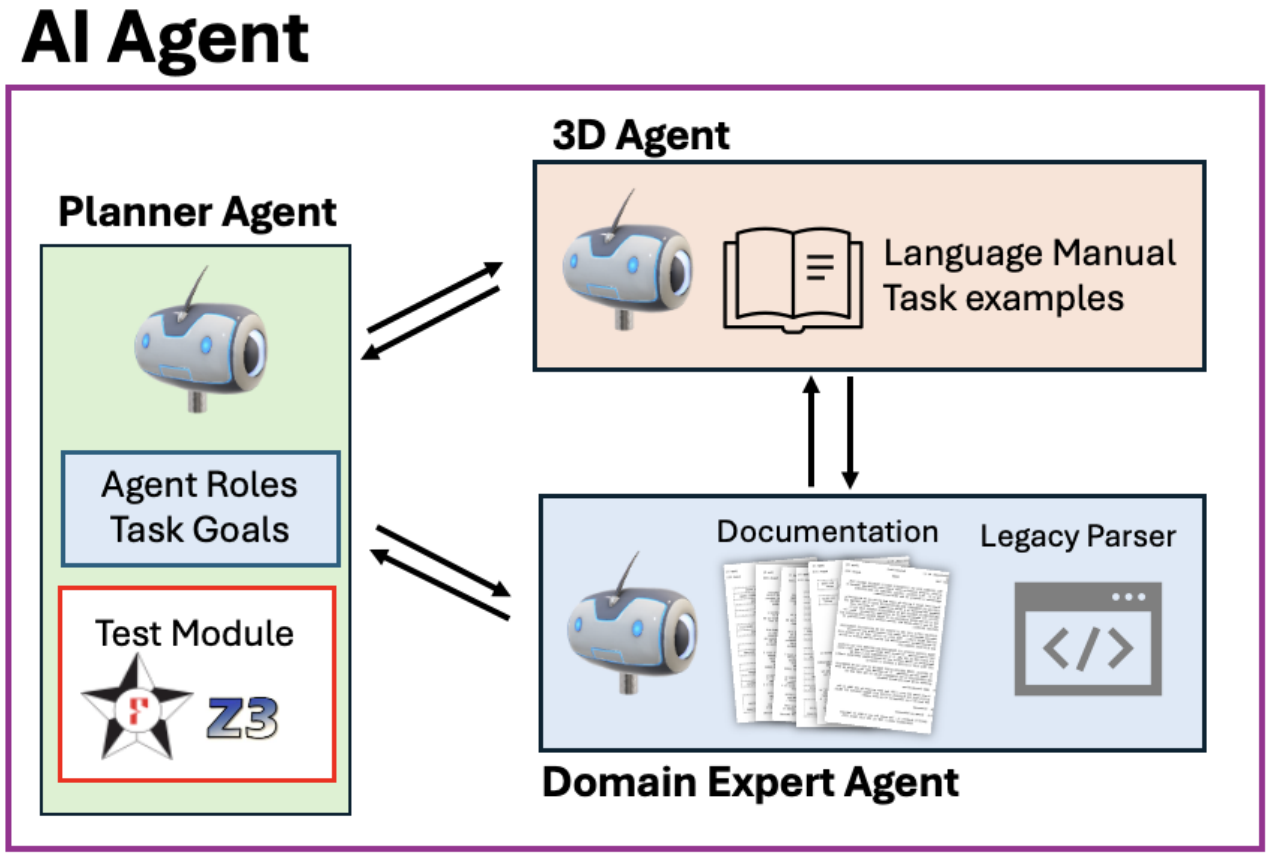
3DGen Workflow

- ✓ Workflow constrains what AI Agents can produce
  - ✓ Provides memory safety guarantees for free with 3D
- ✓ Output backed by symbolic tools
  - Compilers, test generators, proof checkers
- ✓ Tools provide useful + concrete feedback, both for humans & Agents



Agent Implementation

- Three Agent personas collaborating:
  - **Planner:** dictates roles, orchestrates conversation
  - **Domain Expert Agent**: Extracts constraints from NL or Code, provides feedback about generated specification
  - **3D Agent:** Extracts 3D specifications
- Implemented with AutoGen [3]
  - Composable
    Retrieval Augmented (RAG) agents
- No fine-tuning, easy migration to GPT-X!
  - Gpt-4-32k model



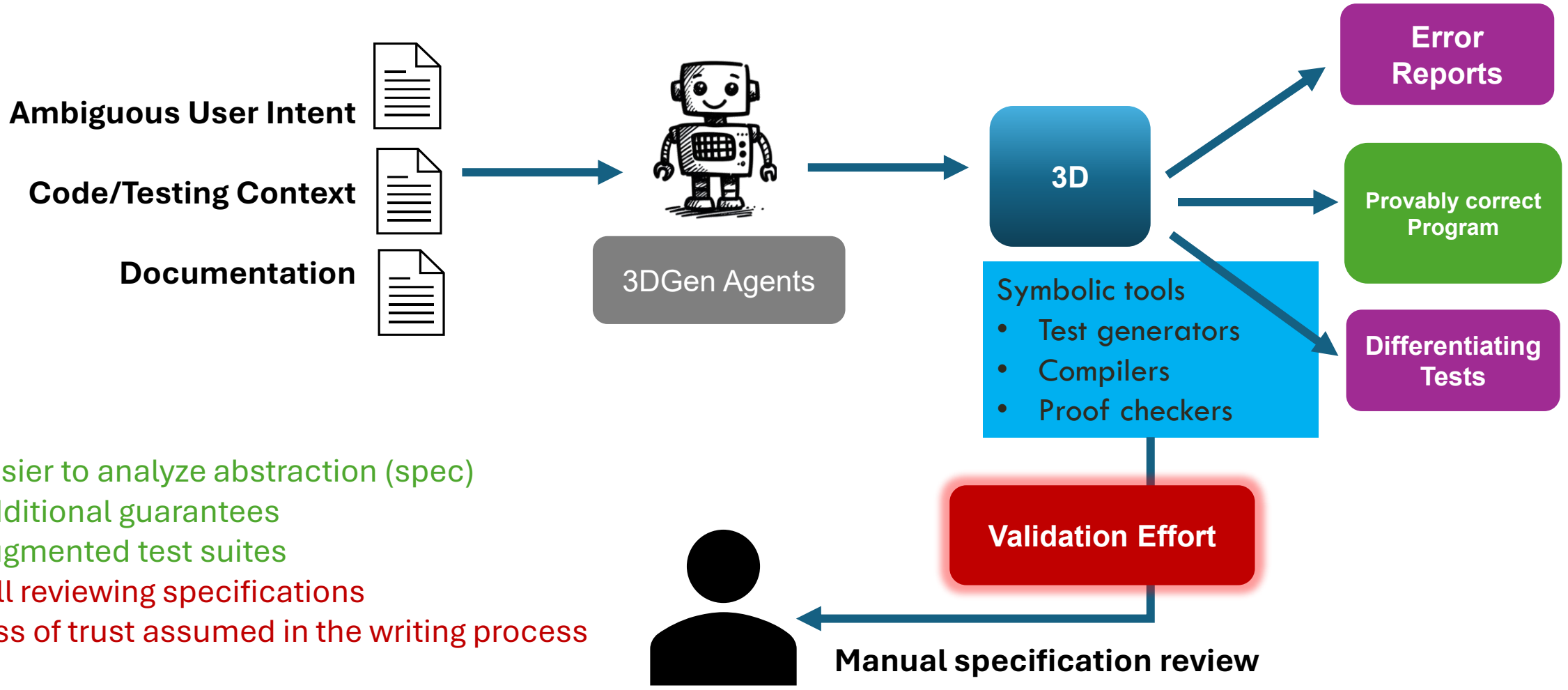
[3] Wu, Qingyun, et al. "AutoGen: Enabling next-gen LLM applications via multi-agent conversation framework." *arXiv preprint arXiv:2308.08155* (2023)

3DGen Results on 20 Network Protocol RFCs

#	Protocol	RFC (Version)	Length (Pages)	Description
1	UDP*	768	3	User Datagram Protocol
2	ICMPv4 *	792	21	Internet Control Message Protocol
3	VXLAN*	7348	22	Virtual eXtensible Local Area Network
4	IPV6*	2460	39 (24)	Internet Protocol version 6
5	IPV4*	791	45 (12)	Internet Protocol version 4
6	TCP*	793	85 (10)	Transmission Control Protocol
7	Ethernet*	7348	22	Ethernet II Frames in VXLAN
8	GRE	2784	9	Generic Routing Encapsulation
9	IGMPv2	2236	24	Internet Group Managment Protocol
10	DHCP	2131	45 (4)	Dynamic Host Configuration Protocol
11	DCCP	4340	129 (14)	Datagram Congestion Control Protocol
12	ARP	826	10	Address Resolution Protocol
13	NTP	5905	110 (4)	Network Time Protocol
14	NBNS	1002	84 (6)	NetBIOS Name Service
15	NSH	8300	40 (8)	Network Service Header
16	TFTP	1350	11	Trivial File Transfer Protocol
17	RTP	3550	104 (3)	Transport Protocol for Real-Time Applications
18	PPP	1661	52 (11)	Point-to-Point Protocol
19	TPKT	2126	25	ISO Transport Service on top of TCP
20	OSPF	5340	94 (13)	Internet Official Protocol Standards

Applying formal methods tools at scale

Constraining AI Output and Lowering Barrier of Use



+ Easier to analyze abstraction (spec)
+ Additional guarantees
+ Augmented test suites
- Still reviewing specifications
- Loss of trust assumed in the writing process

Applying formal methods tools at scale

Constraining AI Output and Lowering Barrier of Use

Reviewing Specifications

- Ground truth cannot be inferred from an imperfect/incomplete system or documentation
  - Users must remain in the loop to review specification correctness

- Specifications of many forms, with varying levels of expertise needed:
  - DSL abstraction (3D Format spec)
  - I/O Tests, Pre/post conditions, Loop invariants

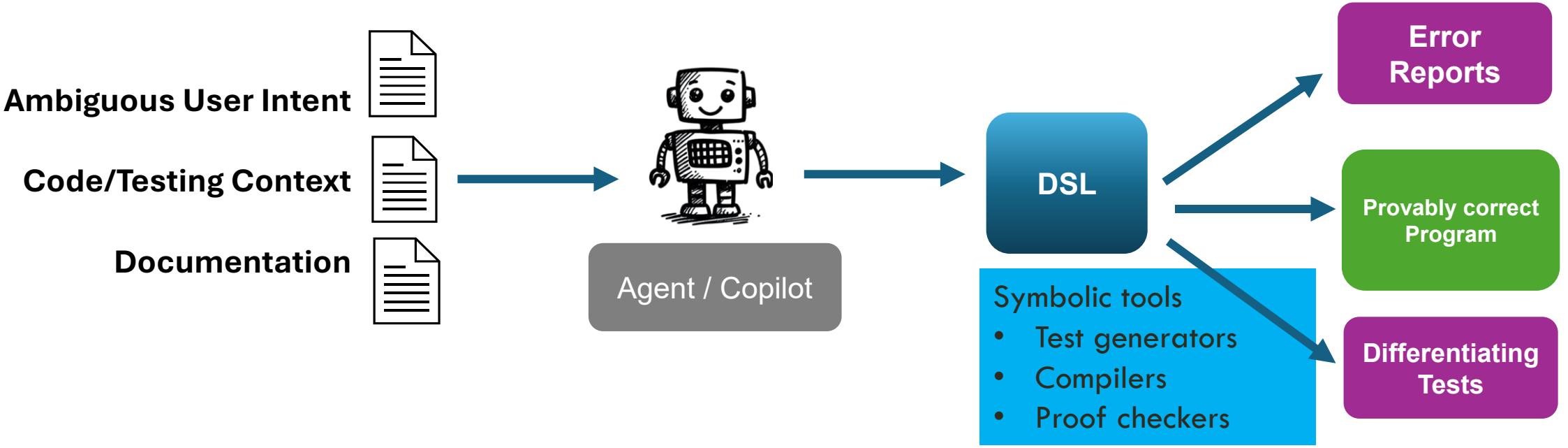
- How do we surface specifications in easy-to-review forms?
  - What part of the spec is a (differential) binary input test exercising?
  - Do we augment specification with NL explanations ?

General workflow

- How do we ensure systematic coverage of NL specification documents?
- Integrating DSL use in multi-agent systems: how do we facilitate interop of DSLs?

Existing PL/FM tools can solve (part of) the trust problem

- Restrict code generation space to provide domain-specific guarantees
- AI agents can be easily taught to code in restrictive DSLs that provide users with: safety & correctness, optimizations, etc.
- Reduce validation burden on the user compared to unconstrained AI use alone



Generalization:

Effectively Analyzable Symbolic Languages

An **all-of-the-above** methodology for trustworthy AI programming

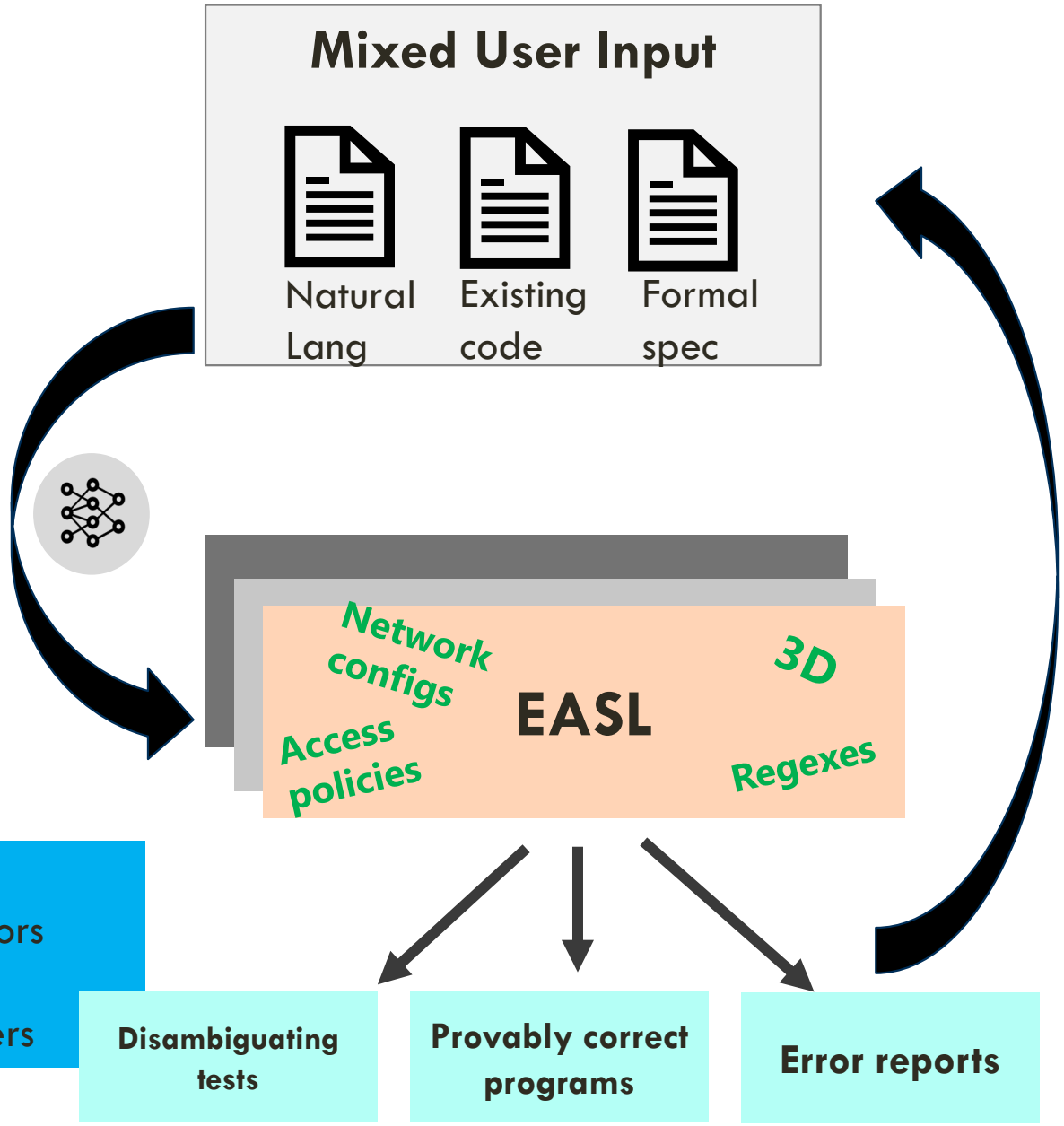
- AI + Testing + Verification

EASLs succinctly capture complex user intent and computational behaviors

AIs target EASLs from mixed user input

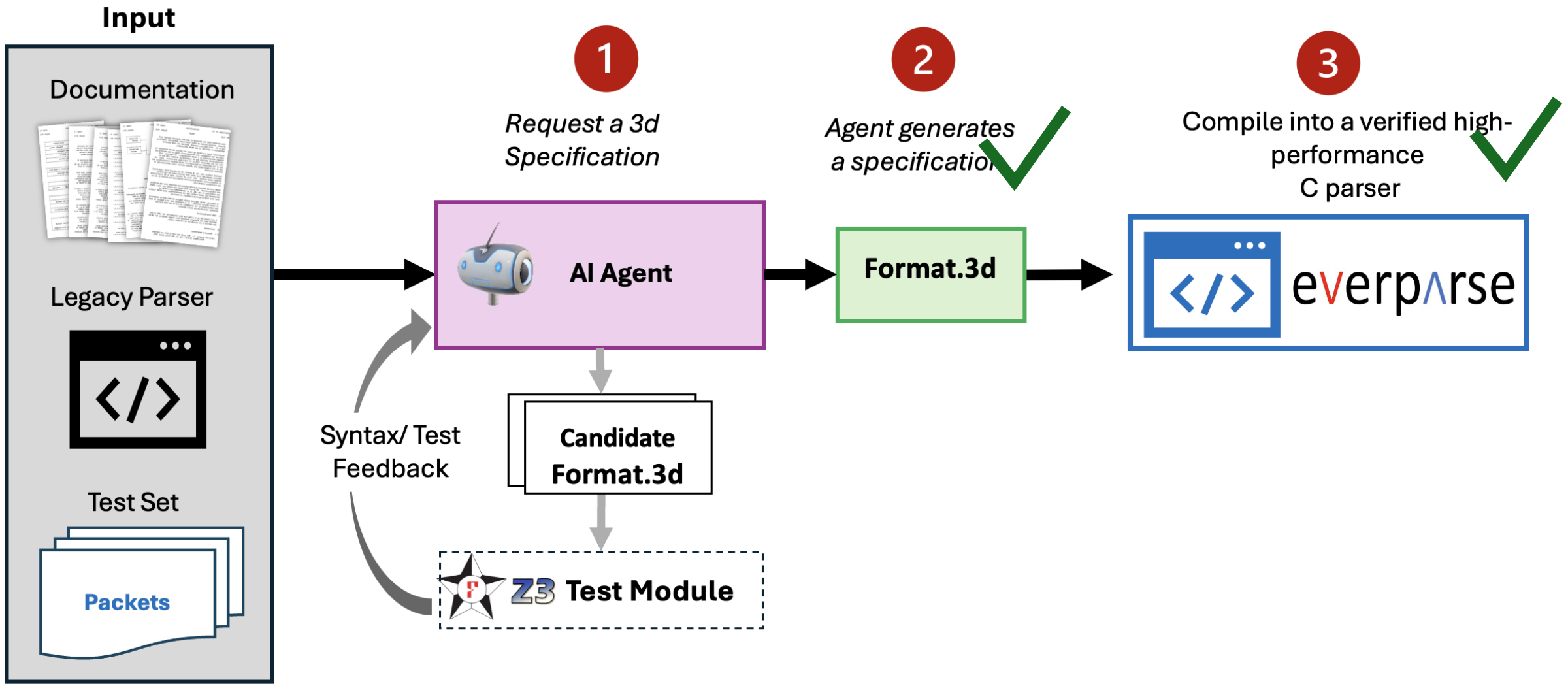
Symbolic tools check EASL code

- provide feedback
- emit provably correct code

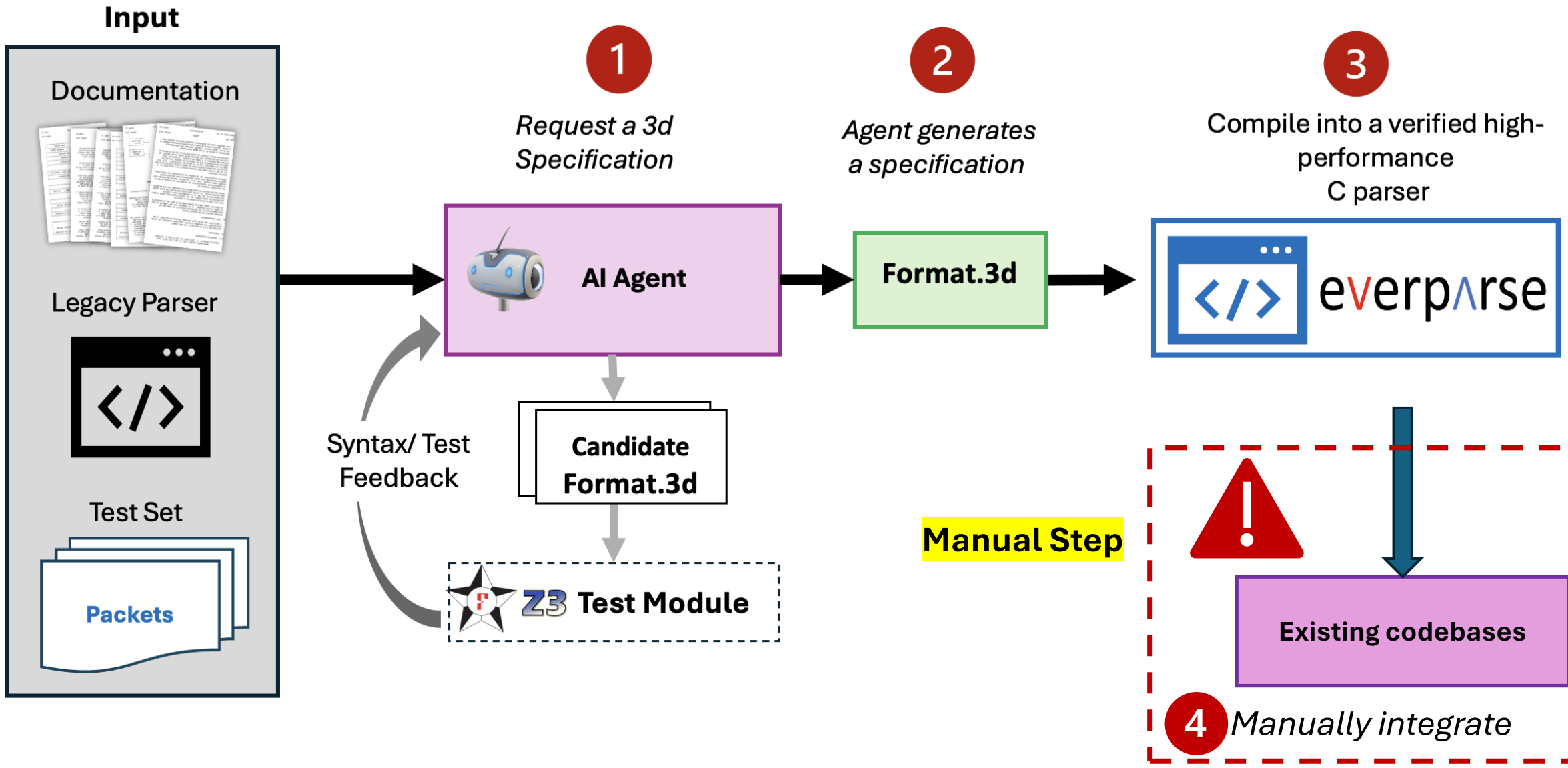


Layer 2: 3DGen

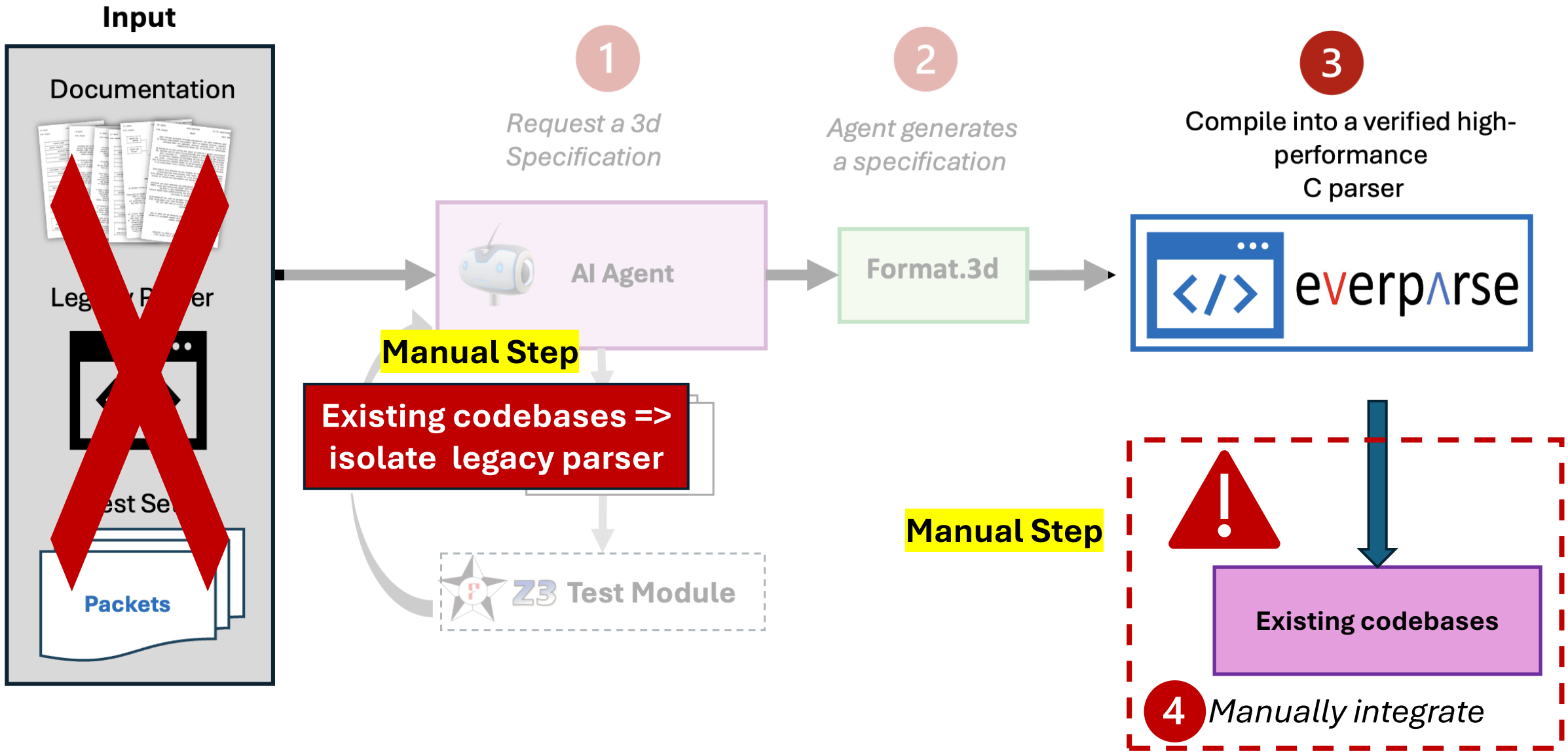
AI-Assisted Generation of Data Format Specifications



Challenge 3: Automating integration

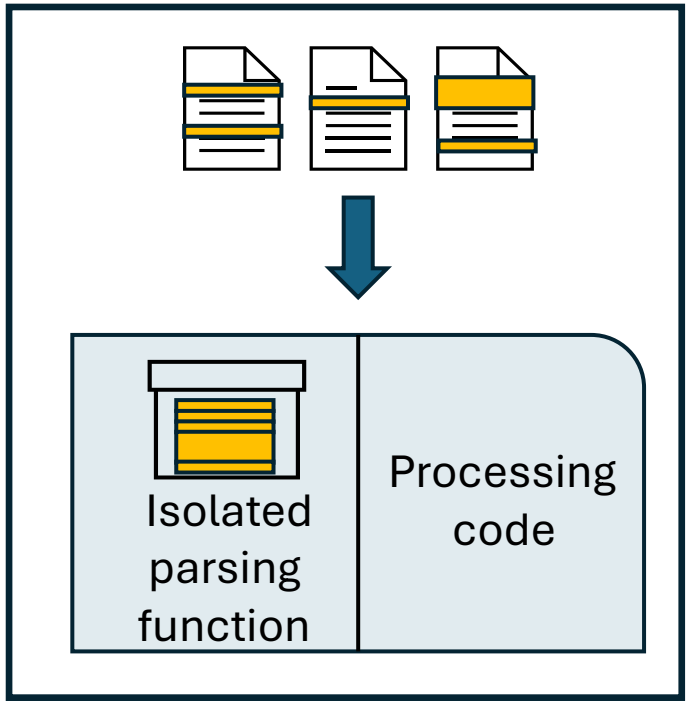


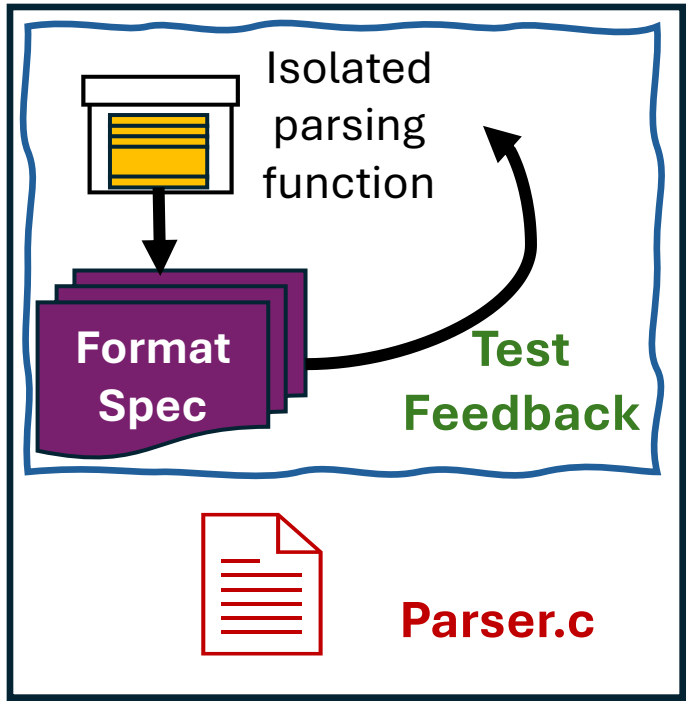
Challenge 3: Automating integration

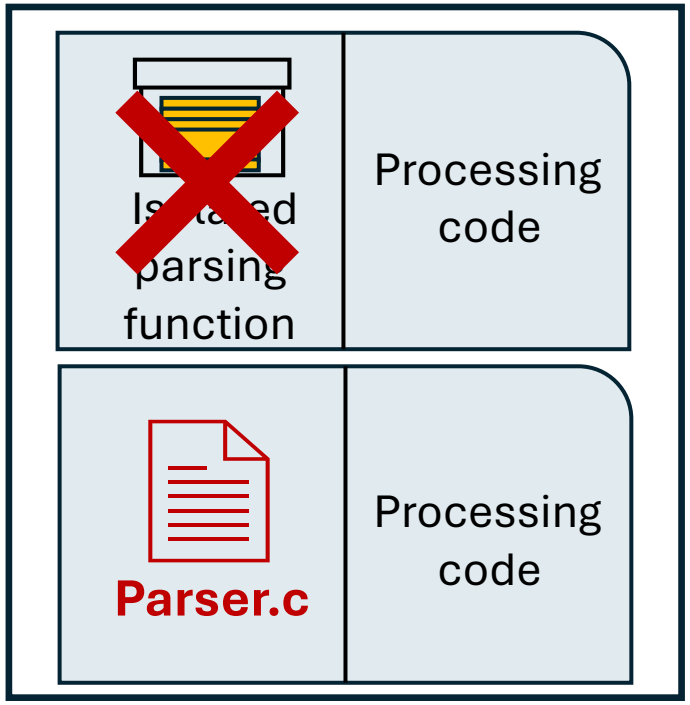


Layer 3: AutoParse

Automatic Refactoring and Integration (*Work in progress*)



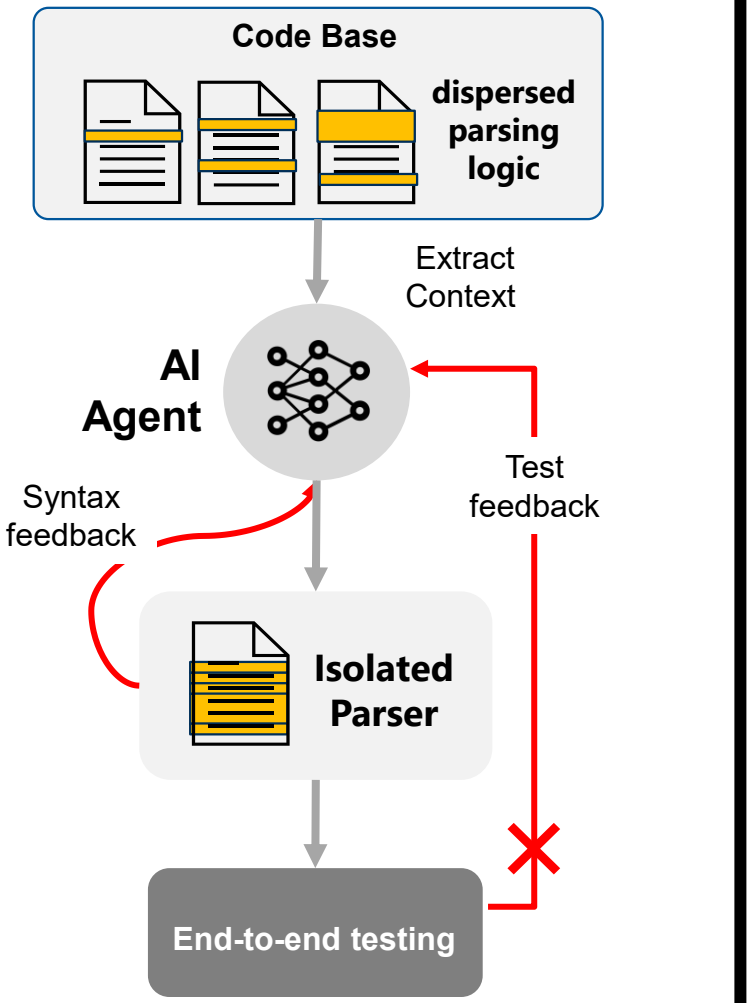




Step 1: Refactor => Create Isolated parsing function

Step 2: Spec Inference and Testing

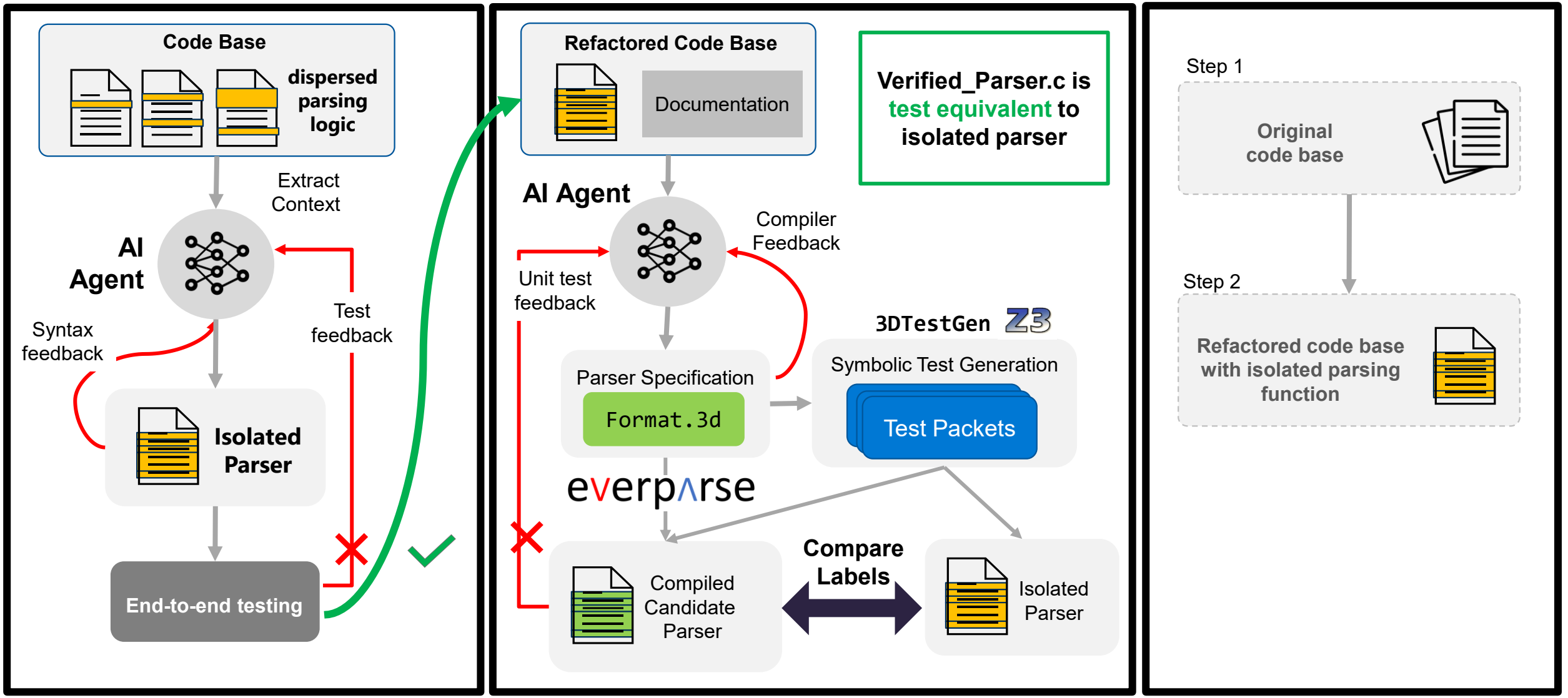
Step 3: Replace Isolated Parser with Verified Code



Step 1

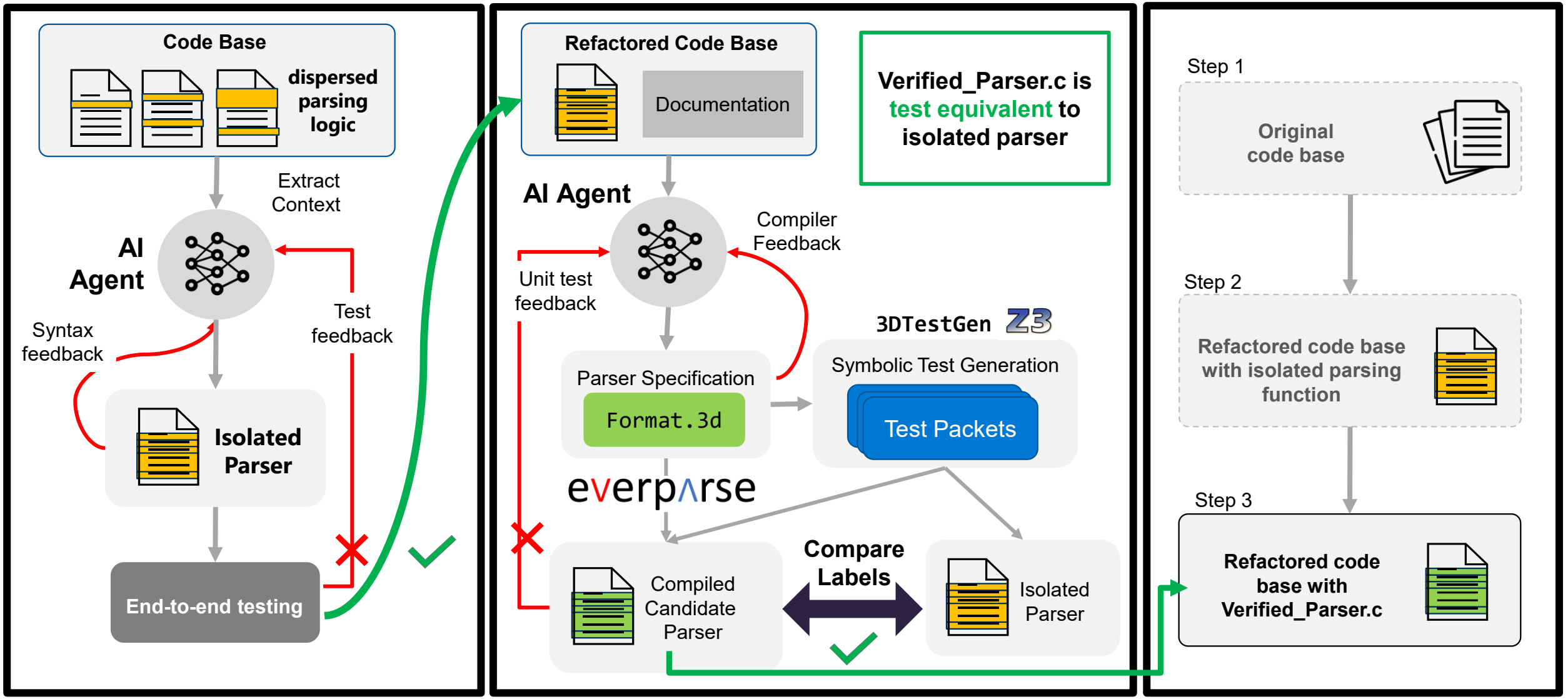
Original code base

Step 1: Refactor to create isolated parsing function



Step 1: Refactor to create isolated parsing function

Step 2: Specification inference and testing



Step 1: Refactor to create isolated parsing function

Step 2: Specification inference and testing

Step 3: Replace isolated parser with verified code

Secure Parsing at MSR RiSE

- **3D, Hyper-V** [PLDI 2022, MSR Blog 2021]
- **3DGen** [ICSE 2025]
- **AutoParse**
- Support for Architecture-Dependent Pointer-Rich Data Structures
- TLS 1.3 [USENIX Security 2019]
- QUIC [S&P 2021]
- ASN.1 DER [CPP 2023]
- CBOR, CDDL, COSE [2025]
- Beyond EverParse: RE#, efficient regex parsing with derivatives [POPL 2025]
- …

For more info: taramana@microsoft.com