



Agentic AI Foundation

AGNTCon + MCPCon North America

Where the **agentic stack** is being built.

Anatomy of a Claude Code Plugin, and Lessons Learned on Creating One

Charlie Lin

Manifest: .claude-
plugin/marketplace.json

```
{
  "$schema": "https://anthropic.com/claude-
code/marketplace.schema.json",
  "name": "rust-ext-review-toolkit",
  "version": "0.2.0",
  "description": "...",
  "owner": {...},
  "plugins": [
    {
      "name": "rust-ext-review-toolkit",
      "description": "....",
      "source": "./plugins/rust-ext-review-toolkit",
      "version": "0.2.0",
      "category": "code-quality",
      "keywords": [...]
    }
  ]
}
```

●
claude marketplace add clin1234/... (/marketplace add ...),
claude plugin install clin1234/... (/plugin install ...)



Agentic AI Foundation

AGNTCon + MCPCon

North America

Where the **agentic stack** is being built.

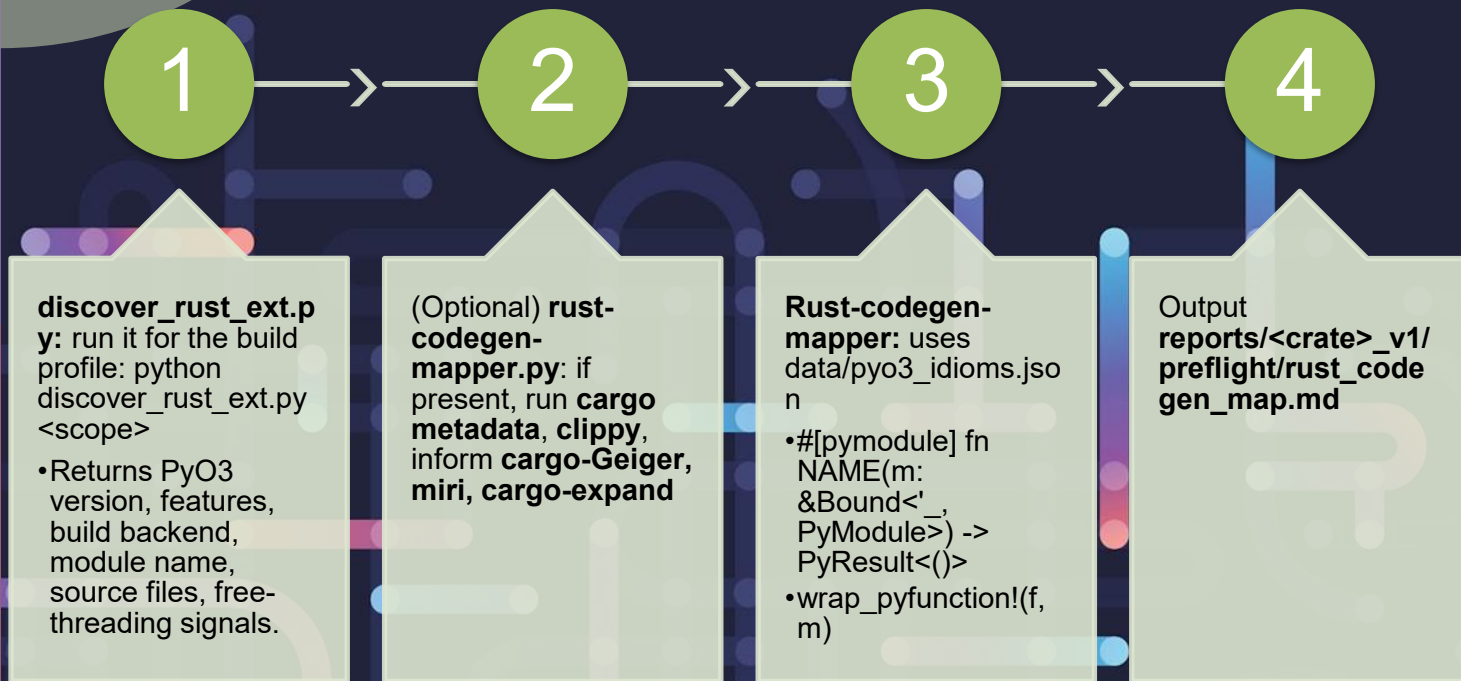
```
$plugin_root=\ngithub.com/clin1234/rust-ext-review-\ntoolkit/plugins/rust-ext-review-toolkit
```

Commands

(in <plugin_root>/commands)

Full command	Purpose	Result
/rust-ext-review-toolkit:explore	Comprehensive or selective review	Consolidated findings and action plan
/rust-ext-review-toolkit:health	Summarize review dimensions	Scores, grades, and recommended next command
/rust-ext-review-toolkit:hotspots	Identify functions needing attention	Ranked list of risky functions
/rust-ext-review-toolkit:migrate	Assess migration work	Phased migration checklist

Preflight Steps That every Command Runs (in `<plugin_root>/scripts`)





Agentic AI Foundation

AGNTCon + MCPCon North America

Where the **agentic stack** is being built.

**/explore: full or
selective review**

Agents (Background dispatchers)

Agents are in
<plugin_root>/agents: MD files
with YAML metadata, tailored
prompt, conditions, classification
roles, and structured output

Agent Example (Parity Checker)

```
---
name: parity-checker
description: Use this agent to find behavioral differences between a Rust/PyO3 extension and a pure-Python fallback
implementation of the same functionality, when a crate ships both.\n\n<example>\nUser: My package has a Rust extension and a
pure-Python fallback. Do they behave the same?\nAgent: I will identify both implementations from the discovery output, then
compare validation, error messages, type coercion, and edge-case behavior to find Python-visible parity gaps.\n</example>
model: opus
color: green
---
```

You are an expert in cross-implementation behavioral parity, specializing in packages that ship **both** a Rust/PyO3 extension and a pure-Python fallback of the same API. Your goal is to find places where the two implementations behave differently in a way Python code can observe.

This agent is **qualitative** — there is no script. You work with Read and Grep, guided by the discovery output.

When This Agent Applies

It applies only to a crate with a **dual implementation**: a compiled Rust extension and a pure-Python fallback for the same public API. This is a common pattern — ``cryptography``, ``pydantic-core``-adjacent packages, ``tokenizers``. Signals:

- ``discover_rust_ext.py`` reports ``has_python_source: true`` (a ``python/`` directory exists)
- a Python module that does ``try: from ._rust import X`` / ``except ImportError: from ._py import X``, or a ``_PURE_PYTHON`` / ``USE_RUST`` flag
- a test suite parametrized over "rust" and "python" backends

If there is no second implementation, **report that and stop** — there is nothing to compare.

Agent Example (Parity Checker)

Analysis Approach

1. ****Locate both implementations.**** From discovery, find the Rust ``#[pyfunction]``/``#[pymethods]`` entry points and the Python fallback module. Build a map of the shared public API — the functions/classes/methods that exist in both.
2. ****Pair the functions.**** For each public name, find the Rust definition and the Python definition.
3. ****Compare, behavior by behavior.**** For each pair, read both and check each Key-Concept dimension. Pay special attention to:
 - input validation that exists in one and not the other
 - the exception `*type*` raised for the same bad input
 - how each handles the edge cases listed above
4. ****Check the dispatch logic.**** Read the ``try/except ImportError`` (or feature-flag) that selects the implementation — confirm the fallback is actually reachable and the two are wired to the same public name.
5. ****Use the test suite as evidence.**** If tests run against both backends, gaps are constrained; if tests only ever exercise the Rust path, the fallback is effectively untested — say so.

Agent Example (Parity Checker)

Classification Rules

- ****FIX****: A validation gap (one path accepts input the other rejects) — especially when the rejected input is unsafe; a different exception **type** for the same error, where user code commonly catches by type.
- ****CONSIDER****: A different exception message; an edge case handled differently with no security impact; a coercion difference (e.g. ``list`` vs ``tuple``).
- ****POLICY****: A deliberate, documented difference (e.g. the fallback is slower but the package documents identical behavior — verify the doc claim).
- ****ACCEPTABLE****: A difference that is not Python-visible (internal representation, performance only).

Output Format

For each parity gap:

...

Parity Gap: [SHORT TITLE]

- ****API****: ``package.function_name``
- ****Rust****: ``src/lib.rs:120``
- ****Python****: ``package/fallback.py:45``
- ****Classification****: `FIX | CONSIDER | POLICY | ACCEPTABLE`
- ****Dimension****: `validation | exception | coercion | edge-case | precision | ...`

****Rust behavior****: [What the extension does for the specific input]

****Python behavior****: [What the fallback does for the same input]

****Observable difference****: [What Python code sees; an input that distinguishes them]

****Security relevance****: [If a validation gap — what guarantee is lost on the looser path]

...

Possible Agents

Group	Agents	Focus
2A	unsafe-block-auditor, pyresult-propagation-checker	Safety-critical (highest value)
2B	gil-discipline-checker, panic-safety-checker	GIL and panic safety
2C	pyclass-traits-checker, pyclass-protocol-checker, module-init-checker	PyO3-specific correctness
2D	lifetime-handle-checker, pyo3-version-compat-scanner	Compatibility and migration
2E	rust-complexity-analyzer	Code quality
2F	parity-checker	Cross-implementation behavior — only when discovery step's <code>has_python_source</code> is true
2G	git-history-analyzer	Churn + similar-bug detection — runs last, uses all prior findings

Output Report Template

PyO3 Extension Review — [crate]

Scope: [what was analyzed] ## Agents: [list]

Executive Summary

[3–5 sentences: overall health, most critical findings, key recommendations]

Crate Profile

- Module: [name] ([N] .rs files)
 - PyO3: [version] | Features: [list] | Python floor: [N]
 - Build backend: [maturin / setuptools-rust / cargo] | Free-threading target: [yes/no]

Key Metrics

Dimension	Status	FIX	CONSIDER	Top Finding
Unsafe Soundness	G/Y/R	N	N	[1-line]
PyResult Propagation	G/Y/R	N	N	[1-line]
GIL Discipline	G/Y/R	N	N	[1-line]
Panic Safety	G/Y/R	N	N	[1-line]
Pyclass Traits	G/Y/R	N	N	[1-line]
Pyclass Protocol	G/Y/R	N	N	[1-line]
Module Init	G/Y/R	N	N	[1-line]
Lifetime Handles	G/Y/R	N	N	[1-line]
PyO3 Version Compat	G/Y/R	N	N	[1-line]
Complexity	G/Y/R	-	N	[1-line]
Rust/Python Parity	G/Y/R	N	N	[1-line]

G = no FIX findings | Y = 1–3 FIX | R = 4+ FIX

Findings by Priority

Number ALL findings sequentially across sections (FIX first, then CONSIDER, then POLICY) so "Finding 12" is referenceable.

Must Fix (FIX) — N

#	Finding	File:Line	Agents
---	-----	-----	-----

Should Consider (CONSIDER) — M

#	Finding	File:Line
---	-----	-----

Tensions

[Where agents disagree or a trade-off exists]

Policy Decisions (POLICY) — P

#	Finding
---	-----

Strengths

[Correct patterns the crate gets right]

Recommended Action Plan

Immediate (FIX) → Short-term (CONSIDER) → Longer-term (POLICY), referencing finding numbers.

Disclaimer



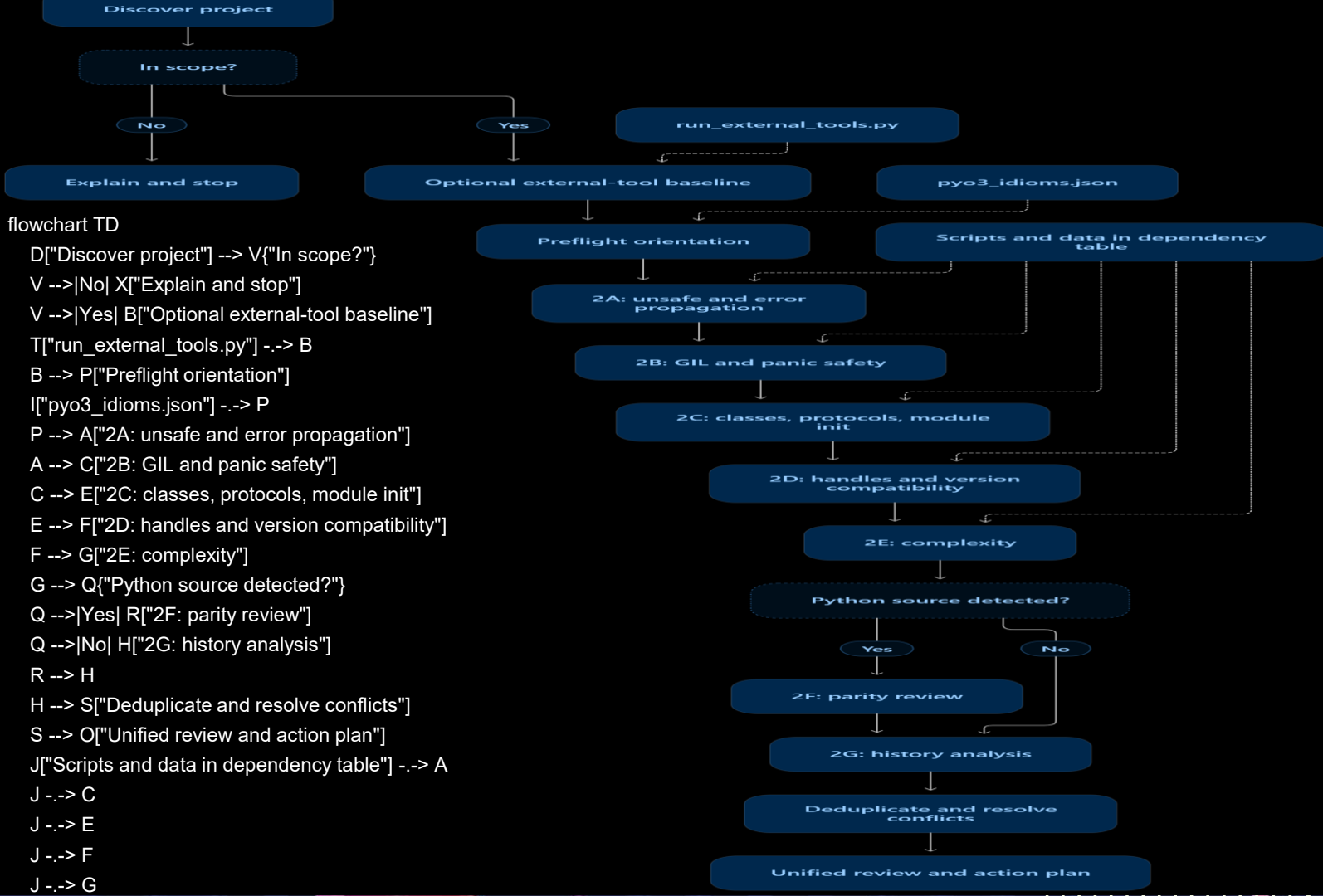
“A large unsolicited findings report is a burden, not a gift: ask the maintainer first, do not auto-file or auto-PR, and disclose security findings privately. The synthesis below is an internal artifact; sharing it is a separate human decision.”



Example report:

<https://gist.github.com/clin1234/abec57c6caee33942c76f5eb986c6dba#file-pydantic-core-md>

Flowchart





Agentic AI Foundation

AGNTCon + MCPCon

North America

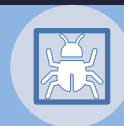
Where the **agentic stack** is being built.

**/health: score each
review dimension**

TLDR version of /explore



Result of each agent is scored starting at 10:



-3 per FIX finding (a real bug / unsound code / a non-compiling API).



-1 per CONSIDER finding, capped at -4.



No change for POLICY / ACCEPTABLE



Floor the score at 1.



Map to a letter: **9-10 = A**, **7-8 = B**, **5-6 = C**, **3-4 = D**, **1-2 = F**.

Output Template

Health Dashboard — [crate]

PyO3 [version] | features: [...] | Python floor: [N] | free-threading target: [yes/no]

Dimension	Score	Grade	FIX	CONSIDER	Worst Finding
-----	-----	-----	-----	-----	-----
Unsafe Soundness	7/10	B	1	0	[1-line]
...					

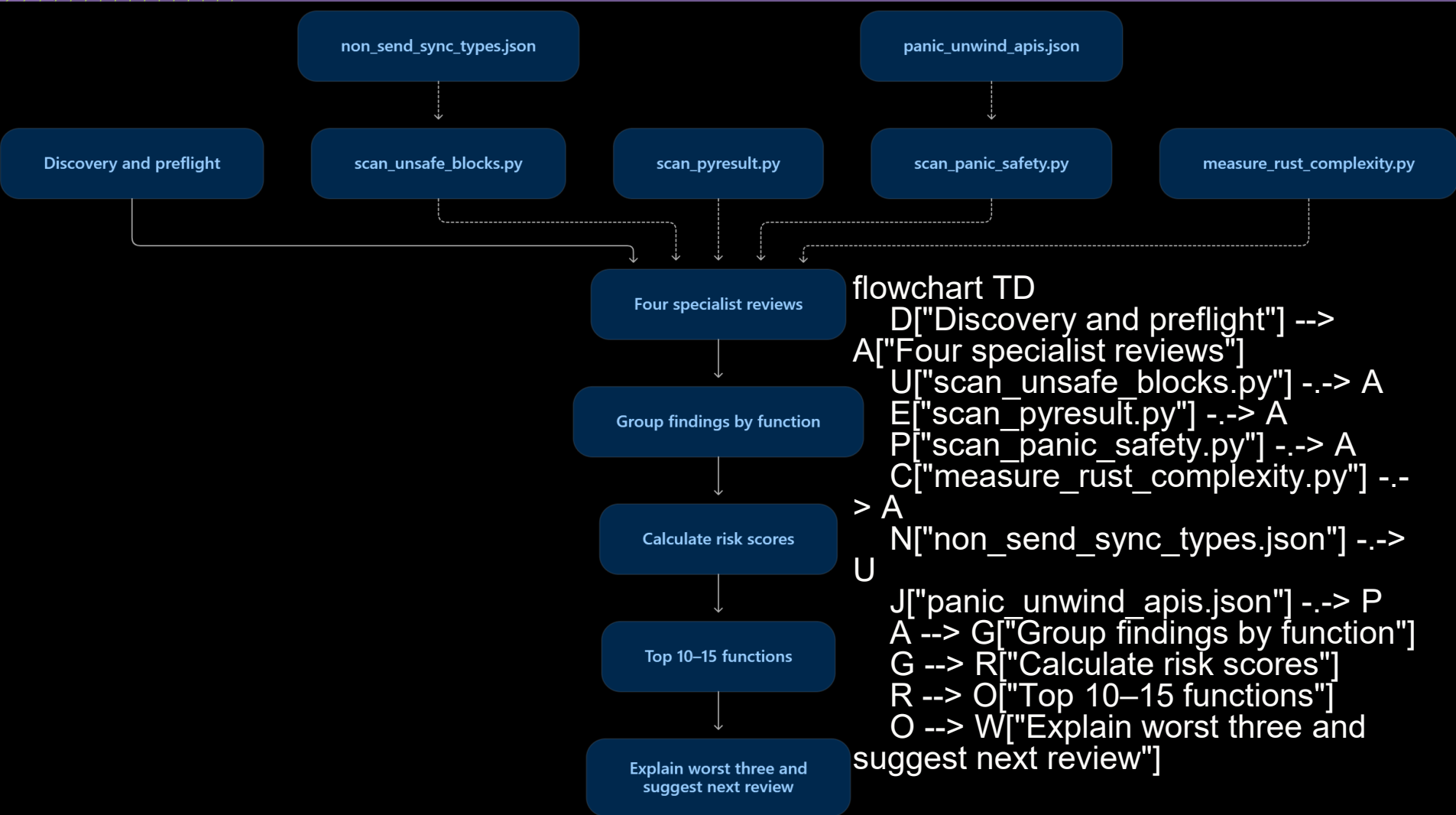
Overall: [average]/10 — [grade]

Headline

[2–3 sentences: the single most important thing to fix, and the crate's overall state.]

Recommended next step

[Either `"/explore . <aspects>"` for the weakest dimensions, or `"/migrate"` if version-compat scored low, or `"/hotspots"` if complexity + unsafe both scored low.]





Agentic AI Foundation

AGNTCon + MCPCon North America

Where the **agentic stack** is being built.

**/migrate: assemble a
migration checklist**

Adopting Newer PyO3



lifetime-handle-checker — legacy &PyAny GIL-refs, Py::as_ref, non-idiomatic handles



pyo3-version-compat-scanner — removed / deprecated / renamed APIs at the crate's version



pyclass-traits-checker — Send + Sync bound gaps, missing frozen for free-threading



module-init-checker — #[pymodule] shape, process-global state

Output Template

Rust Extension Migration Report — [crate]

Current State

- PyO3 version: [version] (or "unresolved — path/workspace dependency")
- Active features: [list]
- Build backend: [maturin / setuptools-rust / cargo]
- Python floor: [3.N or none]
- Free-threading target: [yes/no — and the signals]

Migration Checklist

Phase 1: Bound API migration (PyO3 0.21+)

- [] Replace N `&PyAny`/`&PyDict`/...` GIL references with `&Bound<py, T>`` (lifetime-handle-checker ``legacy_ref_type``)
- [] Replace N `Py::as_ref(py)`` calls with `.bind(py)`` (``legacy_as_ref``)
- [] Convert N `Py<T>`` parameters that are only `.bind()``-ed to `&Bound<py, T>`` (``py_param_should_be_bound``)

Phase 2: Renamed / deprecated APIs

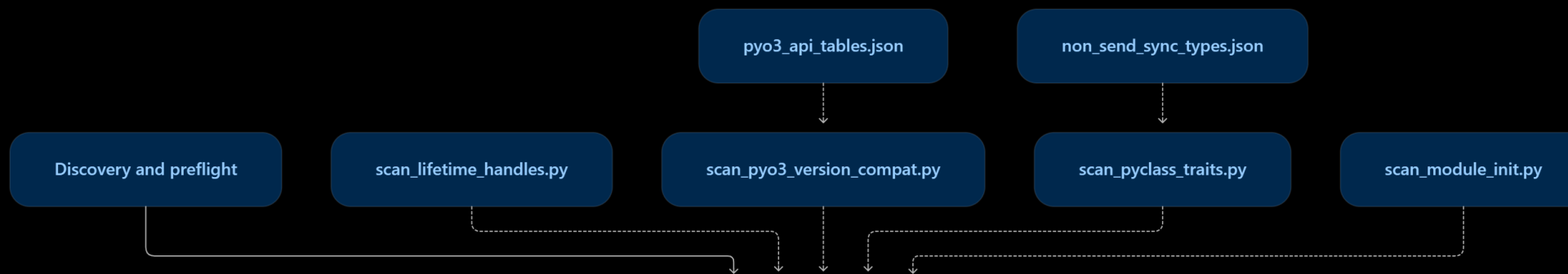
- [] N renamed APIs to migrate (``with_gil`→`attach``, ``downcast`→`cast``, ...) (version-compatible ``renamed_api``)
- [] N deprecated APIs to replace (``deprecated_api``)

Phase 3: Removed APIs (blocking on the target version)

- [] N removed APIs that will not compile (version-compatible ``removed_api``)
- [] Review the ``Send + Sync`` requirement: N ``#[pyclass]`` types with non-Send/Sync fields (pyclass-traits ``non_send_field``)

Phase 4: Free-threading readiness (optional)

- [] N ``#[pyclass]`` types with mutable shared state lack ``frozen`` (pyclass-traits ``missing_frozen_ft``)
- [] Add ``frozen`` + interior locking / atomics to the candidates above
- [] Declare the module's stance: ``#[pymodule(gil_used = true)]`` until the crate is free-threading-audited, or rely on the 0.28+ ``gil_used = false`` default once it is
- [] Build and test against a free-threaded interpreter
- [] For a full GIL-discipline audit, run ``/explore <scope> gil``



flowchart TD

D["Discovery and preflight"] --> A["Four migration reviews"]

L["scan_lifetime_handles.py"] -.-> A

V["scan_pyo3_version_compat.py"] -.-> A

T["scan_pyclass_traits.py"] -.-> A

M["scan_module_init.py"] -.-> A

J["pyo3_api_tables.json"] -.-> V

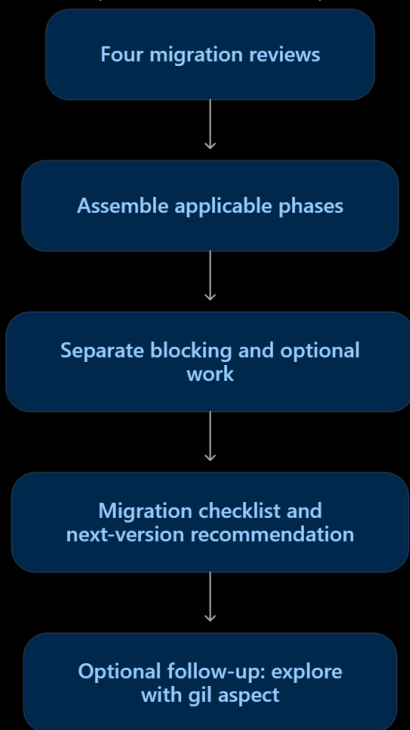
N["non_send_sync_types.json"] -.-> T

A --> C["Assemble applicable phases"]

C --> B["Separate blocking and optional work"]

B --> O["Migration checklist and next-version recommendation"]

O --> G["Optional follow-up: explore with gil aspect"]





Agentic AI Foundation

AGNTCon + MCPCon North America

Where the **agentic stack** is being built.

**/hotspots: rank functions
by combined risk**

Most Prone Code Paths

- **unsafe-block-auditor** — every unsafe block is a soundness obligation
- **pyresult-propagation-checker** — panics across FFI and swallowed exceptions
- **panic-safety-checker** — panics reachable from `#[pymethods]` / `extern "C"`
- **rust-complexity-analyzer**

For each vulnerable function:

- **+3** per FIX finding, **+1** per CONSIDER finding
- **+2** if the rust-complexity-analyzer flagged it a `complex_function`
- **×1.5** if the function carries both unsafe findings *and* a complexity flag

Output Template

Hotspots — [crate]

Top 10-15 Functions to Review

Rank	Function	File:Line	Risk	unsafe	panic	pyresult	complex
1	`MyType::from_raw`	src/lib.rs:120	14	2	FIX	1	- yes
2	...						

The Worst Three

For each of the top three, 2–3 sentences: what makes it dangerous, and the one change that would most reduce its risk.

Where to go next

[Point at `/explore <scope> <aspects>` for the files containing the top hotspots.]

LESSONS?!?!?!?

- Crafting and augmenting agents, skills, and commands mandates combining choice methods of extracting metadata and code snippets with outputting results in a standardized uniform report
- Natural language in MD is (almost) fine
- Vendor neutrality: adopt Open Agent Skills spec (port to other coding agents?)
- More eyeballs... (especially in Claude Marketplace)
- Abi3t exists...

Here it is!

<https://github.com/clin1234/rust-ext-review-toolkit>

gofundme



Scan to donate to Charlie's fundraiser
"Support My Open-Source Conference
Journey"

