

§ 14 PoC 9 — Full↔Light 自動振り分けルータ実装(健全性誤ルート 0/20)

項目	値
対象 spec	spec_section_14_draft.md v0.1, § 14.9 PoC 計画 PoC 9(tier 自動選択)
実施日	2026-06-04
環境	WSL2 Ubuntu 24.04 / CPython 3.12.3 / RustPython 0.5.0 / Rust 1.93.1
目標	PoC 6(Full=CPython 静的 Isolate)と PoC 7/8(Light=RustPython/WASM)の二層に対し、スクリプト+ポリシーから tier を自動選択するルータを実装。PoC 7/8 の乖離マップを根拠に 静的シグナル解析 + プラットフォーム/性能ポリシー で振り分け、20 サンプルで 健全性(誤ルート 0) を検証
結果	☑ 誤ルート 0/20 (乖離するコードを Light に流さない)。18→Light / 2→Full(#04 必然 + #05 保守的)。ポリシー(target=wasm 強制 Light+caveat / exact-exceptions opt-out / perf=fast で hot-loop→Full)すべて動作。 健全性 > 最適性 を実装で担保

0. 結論

§ 14 の二層(Full=CPython 静的 Isolate: 20/20 bit-exact・高速・x86_64 縛り / Light=RustPython→WASM: 19/20・数倍遅・arch 自由)に対し、**どちらで実行すべきかを自動判定するルータ**を実装した。判定根拠は PoC 7/8 で実測した唯一の乖離クラス(例外メッセージ文面)+ 性能 + プラットフォーム制約:

- ・静的 AST 解析で 3 シグナルを抽出: **EXC_TEXT_DEP** (捕捉例外の文字列を検査)/ **UNSUPPORTED** (RustPython 非対応モジュール)/ **HOT_LOOP** (大ループ・ネスト)。
- ・ポリシー: 既定は Light(軽量・可搬)。Full-必要シグナルが立った時のみ Full へ**エスカレート**。プラットフォームが wasm/aarch64 なら Full はビルド不能(PoC 4/5)ゆえ Light 強制(fidelity リスクは caveat で表面化)。
- ・健全性最優先: 「Light に流して誤答する」より「念のため Full(遅いが正しい)」を選ぶ。20 サンプルで **誤ルート 0**。

これで § 14 の二層は「人手で選ぶ」から「ルータが安全に選ぶ」へ。正しさを犠牲にせず、可能な限り軽量の **Light** を使う自動化が成立した。

1. ルータ設計(`tier_router.py`)

```
route(source, target, exact_exceptions, perf) -> Decision{tier, reason, caveats, signals}
```

1.1 静的シグナル(AST visitor)

シグナル	検出内容(根拠)
EXC_TEXT_DEP	<code>except ... as e:</code> で束縛した名 <code>e</code> に対する <code>str(e) / repr(e) / e.args / f"{e}"</code> 。PoC 7/8 で見つかった唯一の CPython/RustPython 乖離(例外文面)を捕捉
UNSUPPORTED	<code>import / from</code> が非対応モジュール(<code>ctypes / multiprocessing / asyncio</code> 等、保守的 <code>denylist</code>)に触れる
HOT_LOOP	リテラル <code>range(N)</code> ($N \geq 100,000$)またはネストループ。Light の $\sim 3\text{-}6\times$ 遅延が効く計算カーネルのヒュースティック

1.2 判定(既定 Light、Full へエスカレート)

```
if target in {wasm, aarch64}:          # Full はビルド不能(PoC 4/5)
    -> LIGHT (forced) + caveats(EXC_TEXT_DEP / UNSUPPORTED があれば fidelity 警告)
elif UNSUPPORTED:                      -> FULL (互換性)
elif EXC_TEXT_DEP and exact_exceptions: -> FULL (例外文面の完全一致)
elif HOT_LOOP and perf == "fast":      -> FULL (性能、Light の数倍遅を回避)
else:                                  -> LIGHT (可搬・十分な忠実度)
```

--run 時は選択 tier の実行系へ委譲: Full= `$FULL_BIN` (PoC 6 の `iso_source`)既定 `python3 /`
Light= `$LIGHT_BIN (rustpython / wasm)`。

2. 健全性評価(20 サンプル)

各サンプルのルータ判定を、実測 bit-exact マップ(CPython vs RustPython, PoC 7)と突き合わせ。誤ルート = Light に振ったが実は乖離する(=誤答)。

sample	router	bitexact	verdict	reason
01_metaclass_registry	light	yes	ok	no Full-requiring signal
02_init_subclass_registry	light	yes	ok	...
03_type_dynamic_class	light	yes	ok	...
04_abc_abstractmethod (EXC_TEXT_DEP)	full	NO	ok	exact exception text
05_slots_repr (EXC_TEXT_DEP)	full	yes	ok	exact exception text
06_property_validation	light	yes	ok	...

```
07..20 (descriptor/getattr/setattr/callable/functools/
        singledispatch/wraps/namedtuple/dataclass/enum/
        defaultdict/generator/contextmanager/eval-exec)
                                light   yes         ok         no Full-requiring signal
```

```
=== soundness: routed Light=18 Full=2 MISROUTES=0 / 20 ===
```

- ・誤ルート 0/20 —— 健全性の核心。乖離する #04 を Light に流していない。
- ・Full 2 件 = #04(必然)+ #05(保守的)。

2.1 #05 は「安全な過剰 Full」 —— 健全性 > 最適性の実例

#05_slots_repr も `print("slots guard:", "z" in str(e))` と例外文面を検査するため EXC_TEXT_DEP が立ち Full へ。実際には RustPython の AttributeError 文面も “z” を含むのでたまたま bit-exact だった。だがルータは静的解析時に「文面が偶然一致する」ことを証明できない。よって安全側に倒して Full を選ぶ —— これは正しい設計判断:

- ・誤って Light(false-Light)= 誤答(高コスト) ≧ 念のため Full(false-Full)= 遅いだけ(低コスト)。
- ・達成可能な Light 上限は 19(bit-exact な全件)、ルータの実績は 18 —— 保守的取りこぼし 1(#05)、危険な取りこぼし 0。
- ・例外文面に依存しないと運用者が判断すれば `--no-exact-exceptions` で #05 も Light に回せる (§ 2.2)。

3. ポリシー動作デモ

```
-- #04 (EXC_TEXT_DEP), native, exact --
signals: exc_text_dep=True hot_loop=False ; "str(e) on caught exception"
TIER=full reason=exact exception text required (EXC_TEXT_DEP)

-- #04, target=wasm (Full ビルド不能) --
TIER=light reason=target=wasm: Full tier unavailable, Light forced
caveat: exception-text dependence: message wording differs on RustPython <- 正直に警告

-- #04, --no-exact-exceptions (文面差を許容) --
TIER=light reason=no Full-requiring signal; Light is sufficient

-- hot-loop (range(2,000,000)), perf=balanced --
signals: hot_loop=True ; "range(2000000) >= 100000"
TIER=light (hot loop present but perf!=fast; Light accepted)

-- hot-loop, perf=fast --
TIER=full reason=hot loop + perf=fast (avoid Light ~3-6x slowdown)
```

- ・target=wasm: Full が出せない環境では Light を強制しつつ、#04 のような fidelity リスクを caveat で表面化(隠さない)。
- ・exact_exceptions: 既定 True(厳密)。運用者が文面差を許容するなら opt-out で Light 化。
- ・perf: fast で hot-loop を Full へ(PoC 7/8 の ~3-6× 遅延回避)、balanced は Light 許容。

4. 重要な観察

- 健全性を実装で保証(誤ルート 0): 唯一の乖離クラス(例外文面)を AST で確実に捕捉し、該当コードを必ず Full へ。20 サンプルで誤答ルート 0。
- 健全性 > 最適性のトレードオフが #05 に出た: 静的に偶然一致を証明できない以上、例外文面依存は一律 Full。false-Full(遅いだけ)を許容し false-Light(誤答)を排除する、という安全側設計が実例で機能。
- ポリシーで運用者の意図を吸収: プラットフォーム制約(wasm→Light 強制)・厳密性(exact_exceptions)・性能(perf=fast で hot-loop→Full)を独立ノブ化。同じスクリプトでも文脈で最適 tier が変わる。
- PoC 4~8 の知見が全部判定根拠に: WASI で Full 不能(PoC 4/5)/ Light の例外文面差(PoC 7/8)/ Light の ~3-6× 遅(PoC 7/8)—— 実測 finding がそのままルーティング規則の根拠。
- 拡張点が明確: UNSUPPORTED denylist は今後 RustPython 非互換が見つかるたびに追記(現状 20 サンプルでは発火せず)。HOT_LOOP は静的ヒューリスティックゆえ、実行時プロファイル併用で精度向上余地。

5. 配布物

```
engine/poc9_router/
├─ tier_router.py      AST シグナル解析 + ポリシー判定 + --run 実行委譲(ライブラリ兼 CLI)
└─ run_router_eval.sh  20 sample 健全性評価(誤ルート計数)+ ポリシーdemo
```

再現:

```
cd engine/poc9_router
bash run_router_eval.sh
# 単体: python3 tier_router.py <script> [--target wasm|aarch64] [--perf fast] [--no-exact-exceptions] [--explain] [--run]
# 実行委譲例: FULL_BIN=/tmp/iso_source LIGHT_BIN=rustpython python3 tier_router.py app.py --run
```

6. § 14.9 PoC ロードマップ 進捗

Phase	内容	状態
☑PoC 0-3	最小実証 / Any / 動的構文 / 巷の実コード 20 + 速度	完了
☑PoC 4-6	償却・根本原因・静的解消・ソース正規ビルド(Full tier 確立)	完了
☑PoC 7-8	Light tier 忠実度 19/20・WASM クロスコンパイル実機実証	完了

Phase	内容	状態
☒PoC 9	Full↔Light 自動振り分けルータ (静的シグナル+ポリシー、誤ルート 0/20)	完了(本書)
PoC 10	実行時プロファイル連携 (HOT_LOOP 動的判定)/ ルータの 実行委譲を本番配線 (FULL_BIN=iso_source)/ UNSUPPORTED denylist を RustPython 差分網羅で拡充	次

PoC 9 完了。Full↔Light 自動振り分けルータを実装 —— 捕捉例外の文字列検査(EXC_TEXT_DEP)/ 非対応モジュール(UNSUPPORTED)/ 大ループ(HOT_LOOP)を AST で静的抽出し、プラットフォーム(wasm/aarch64 は Full 不能ゆえ Light 強制)・厳密性(exact_exceptions)・性能(perf)ポリシーで tier を選択。20 サンプルで誤ルート 0(乖離する #04 を Light に流さない)、Light 18/Full 2。#05 は例外文面依存ゆえ保守的に Full へ(偶然 bit-exact だが静的に証明不能 → 安全側)= 健全性 > 最適性の実装。target=wasm では Light 強制しつつ fidelity を caveat で表面化、perf=fast では hot-loop を Full へ。PoC 4~8 の実測 finding がそのままルーティング規則の根拠となり、§ 14 二層が「安全な自動選択」として閉じた。