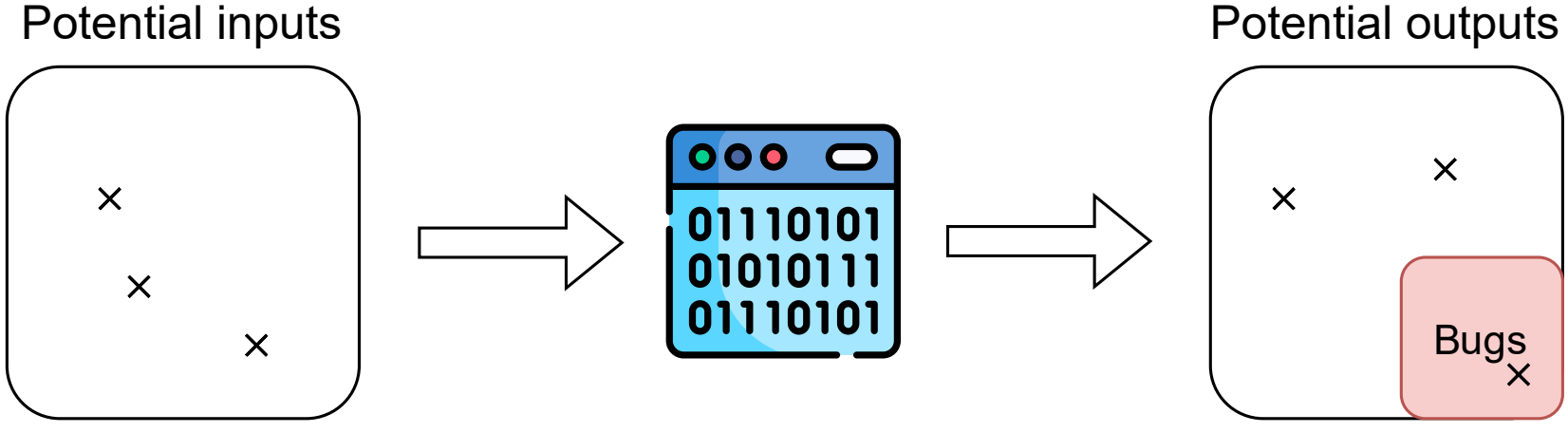
tauri-fuzzer

BUGS BAD

HOW DO WE REMOVE BUGS IN PROGRAMS?

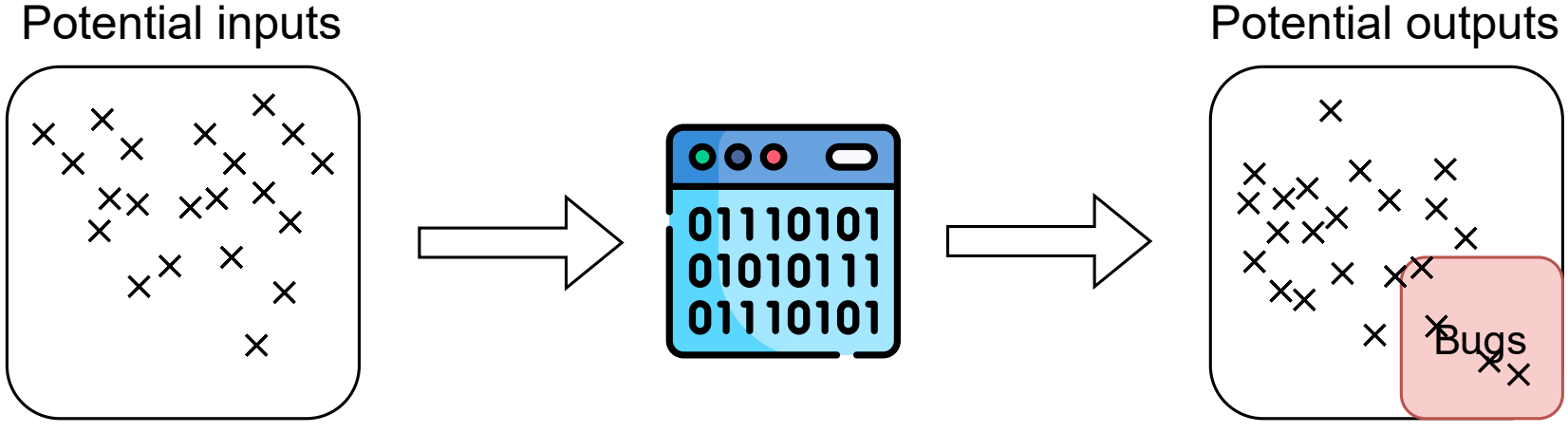
- Writing tests
- Fuzzing
- Formal Verification

TESTING



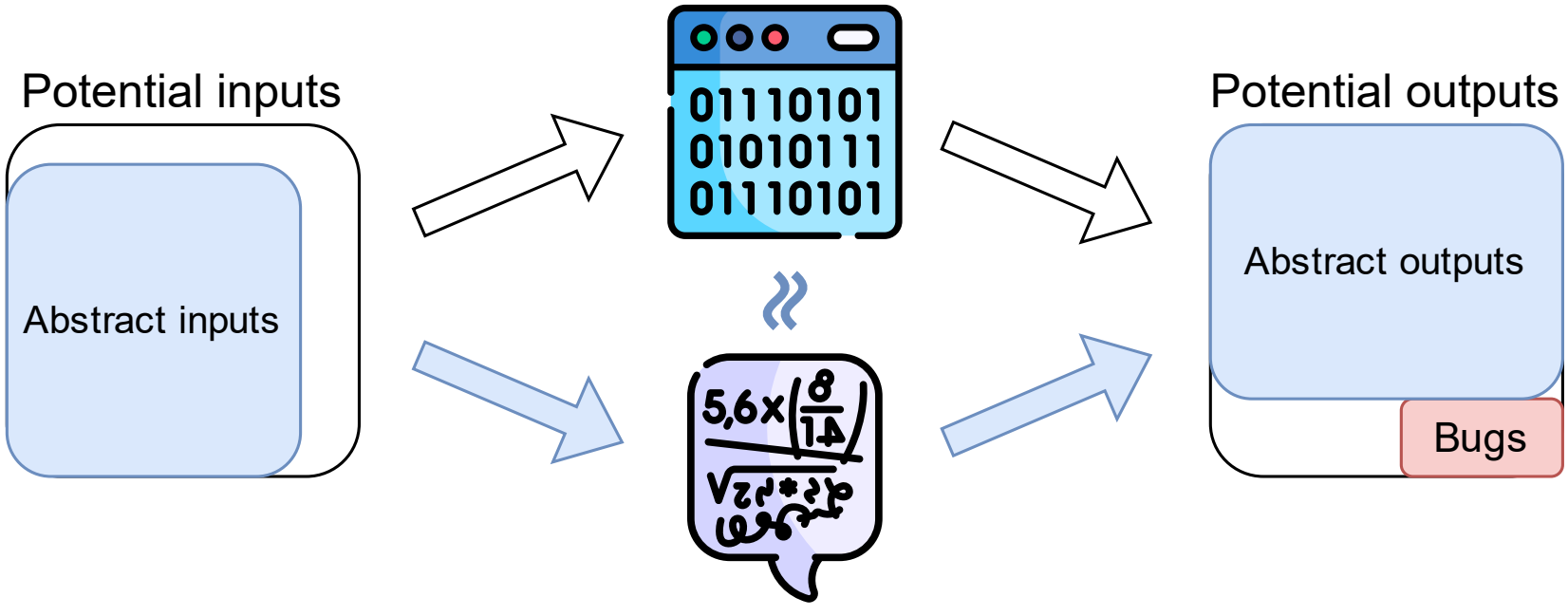
- Manual process
- Setup: easy
- Usage: every software

FUZZING



- Automatic testing
- Setup: moderate
- Usage: important libraries and critical software

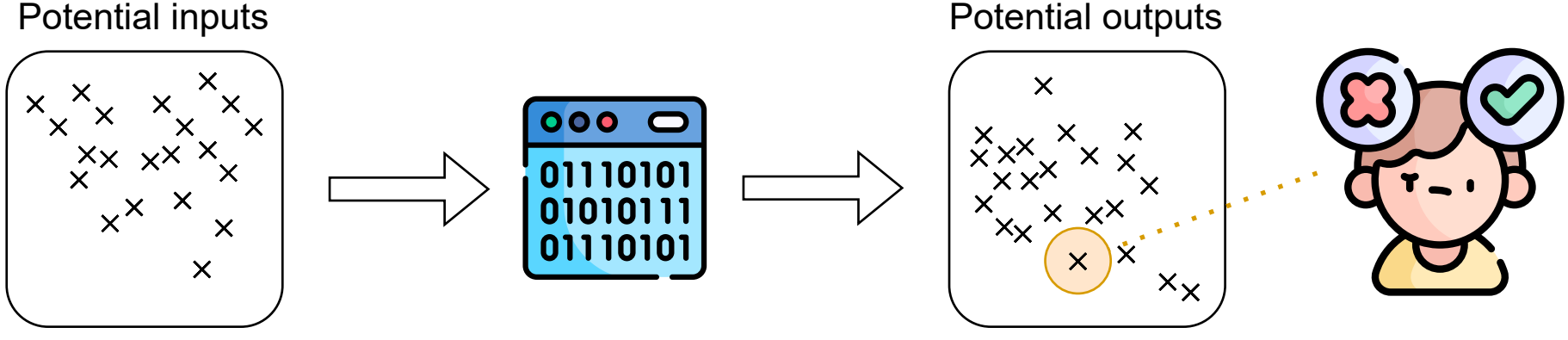
FORMAL VERIFICATION



- Setup: hard
- Expert knowledge and expensive
- Usage: critical software

CURRENT STATE OF FUZZERS

FUZZER CHALLENGE: CHECKING AN EXECUTION



HOW DO WE DETECT IF AN EXECUTION FAILED?

- Write a **checker** specialized to the fuzzed program
- Use a generic **checker** (*crashes*, *memory corruption*)

FUZZING IN THE APPLICATION WORLD

FUZZING IS RARELY USED FOR APPLICATION DEVELOPMENT

- Writing a dedicated checker is costly
- Crashes are less critical
- Memory corruption is rare

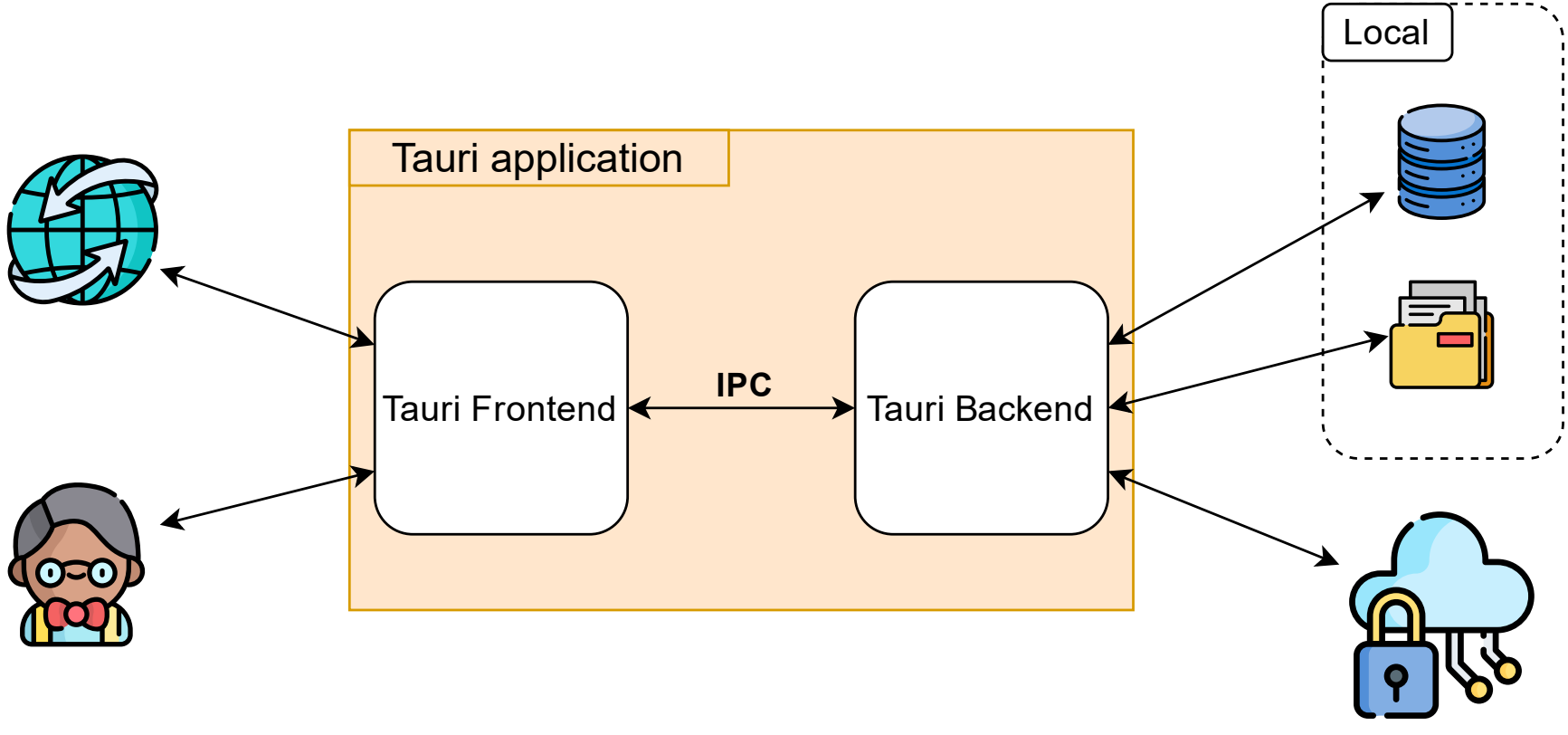
GOAL OF THE `tauri-fuzzer`

Bridging the gap between application devs and fuzzers

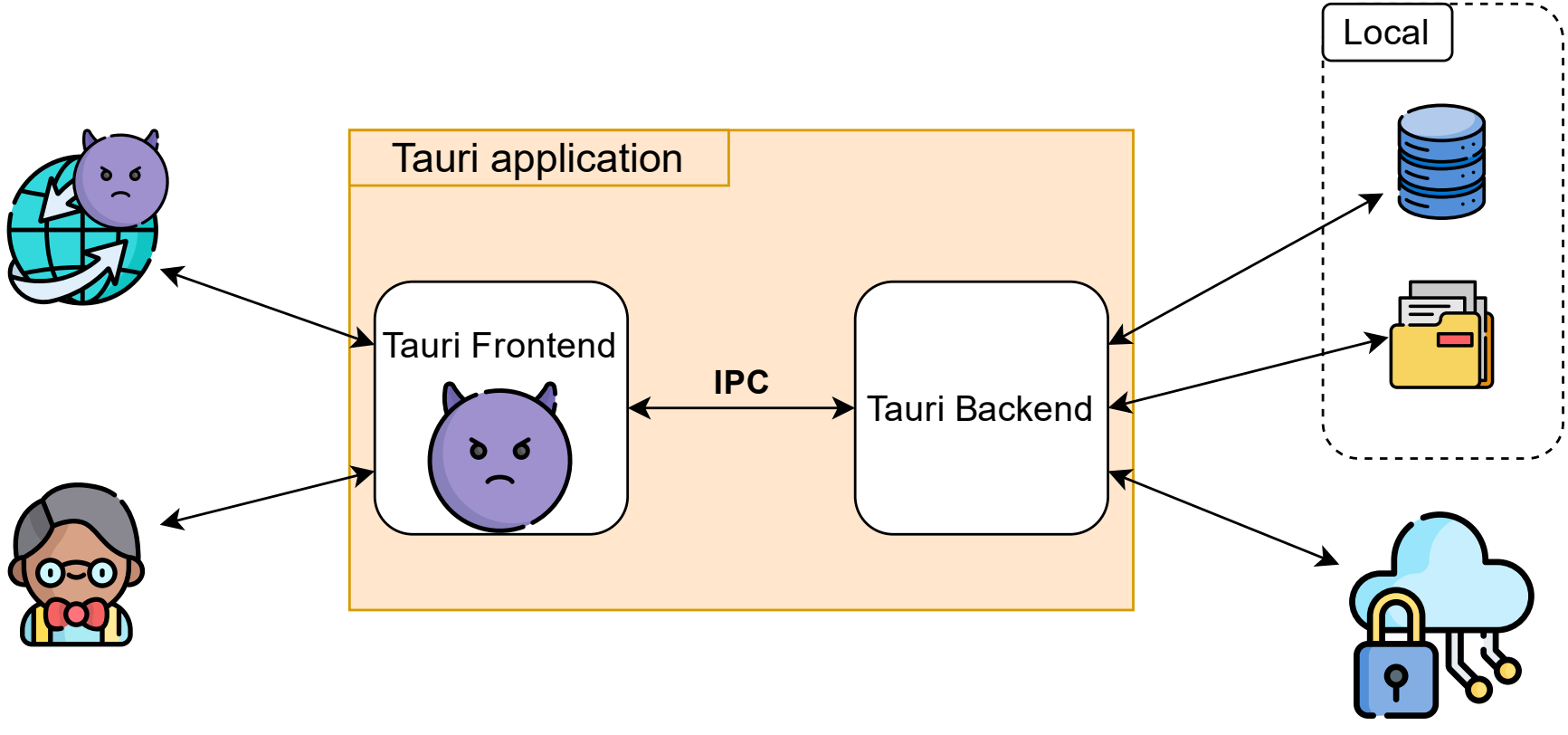
- Provide a **security policies engine** as generic checker
- Make fuzzing a Tauri app as easy as possible

TAURI APP SECURITY MODEL

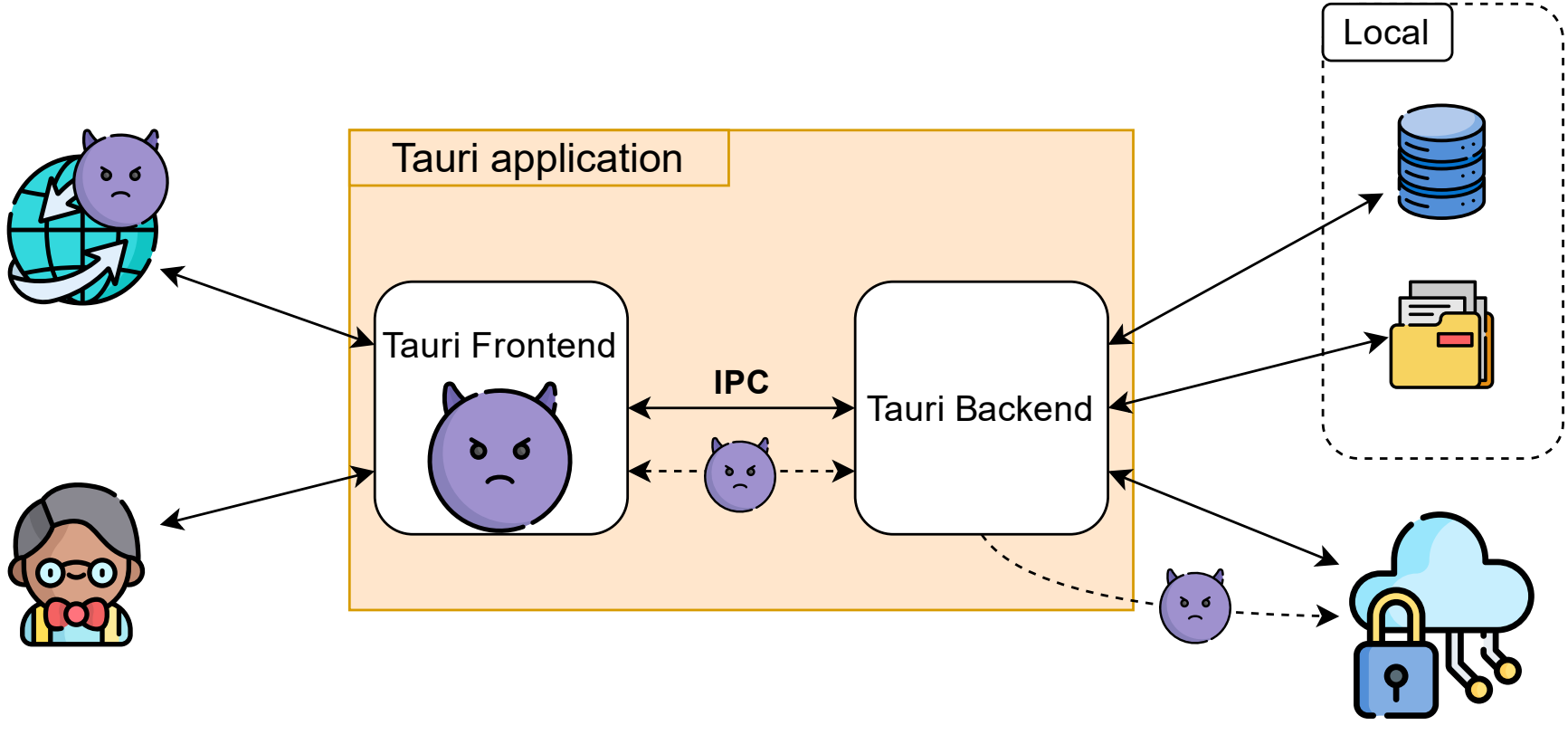
TAURI APP



SECURITY MODEL OF A TAURI APP

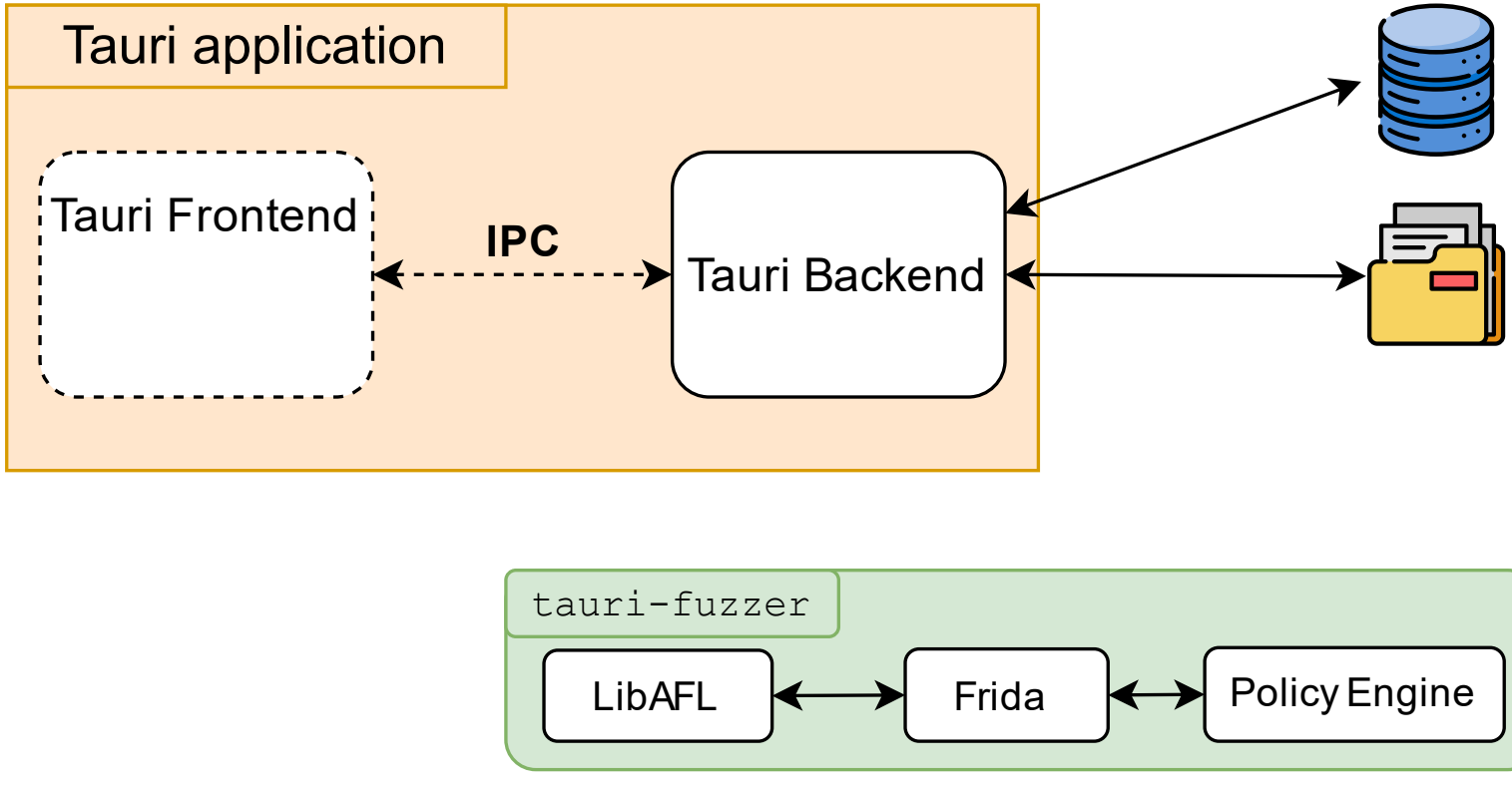


THREAT MODEL OF A TAURI APP

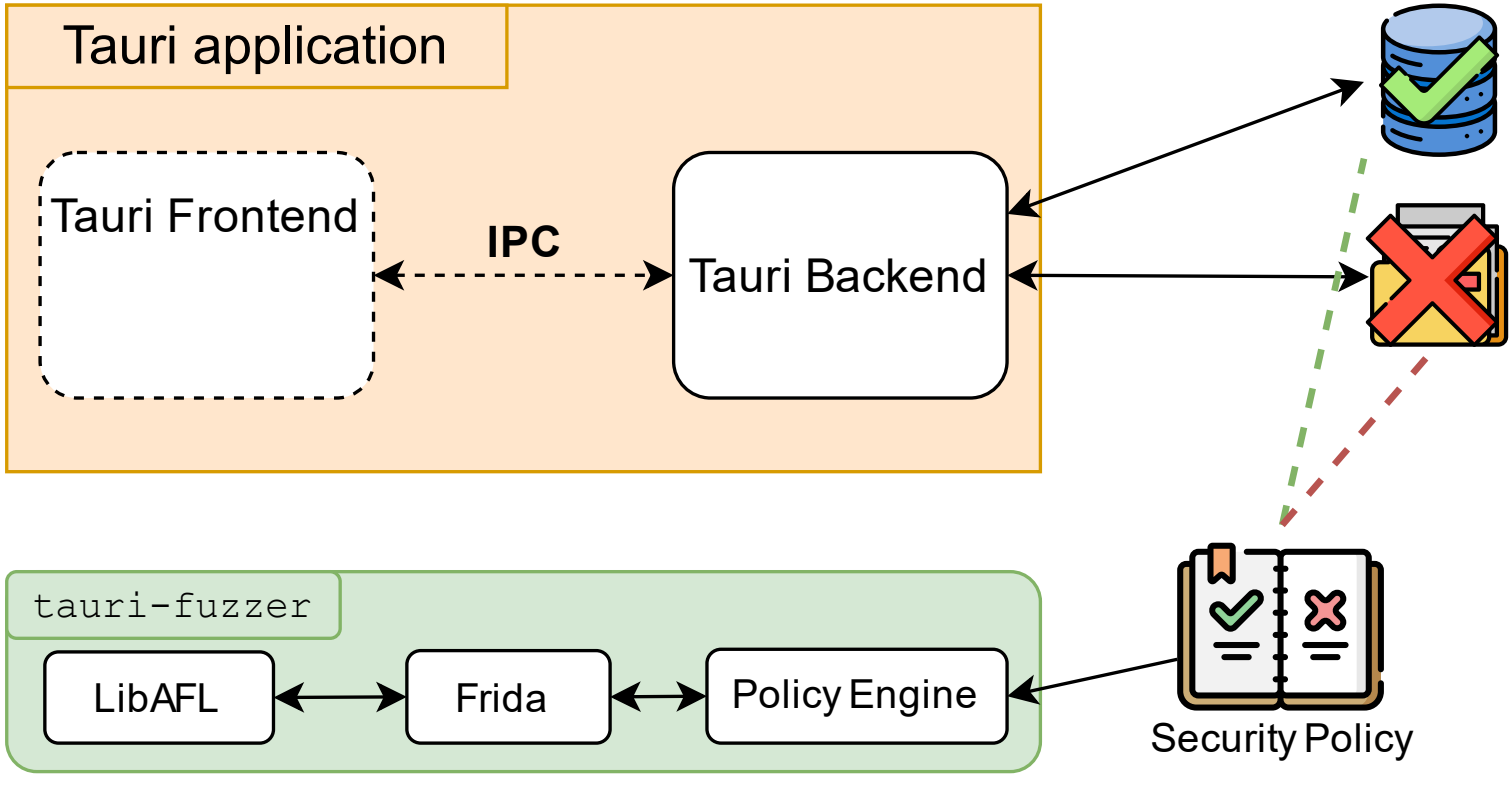


FUZZING SECURITY POLICIES WITH `tauri-fuzzer`

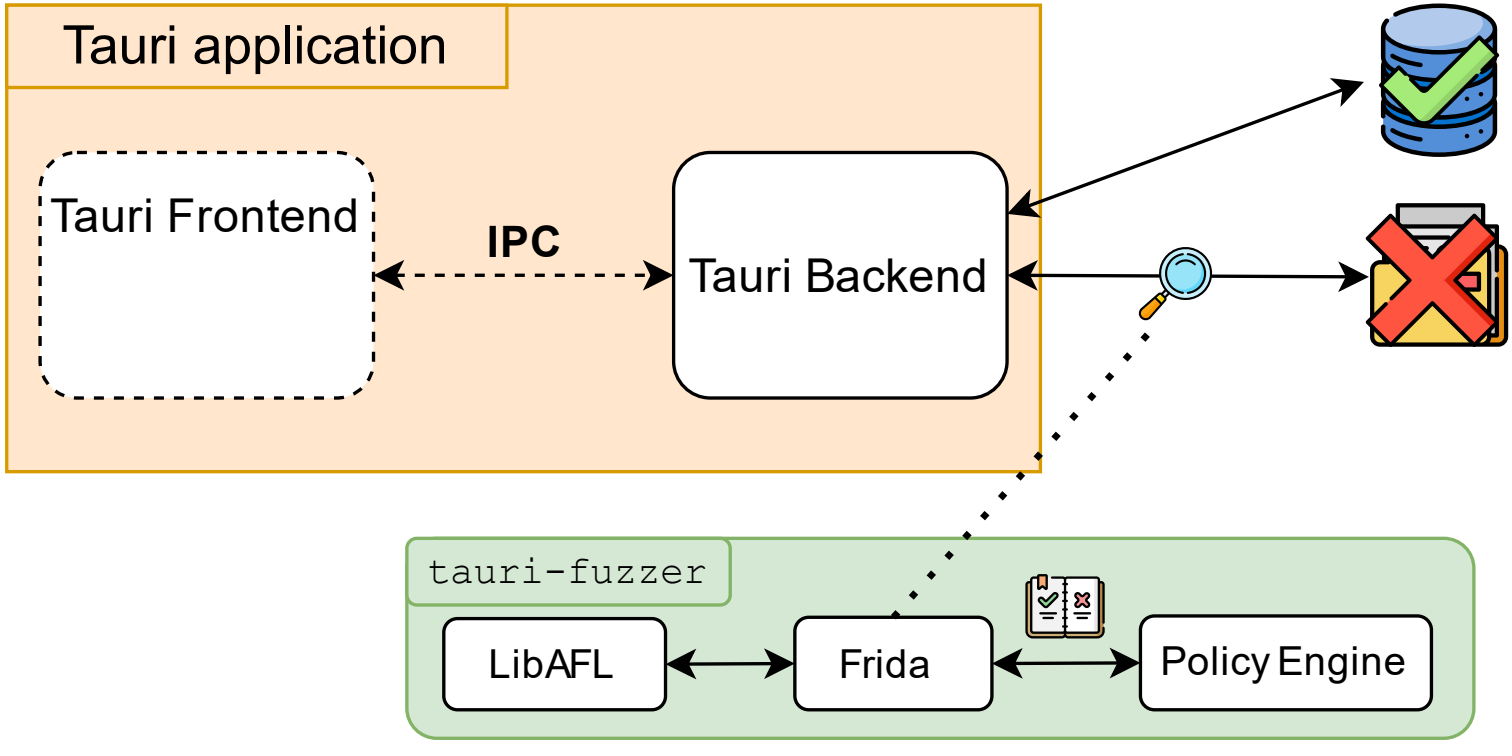
tauri-fuzzer



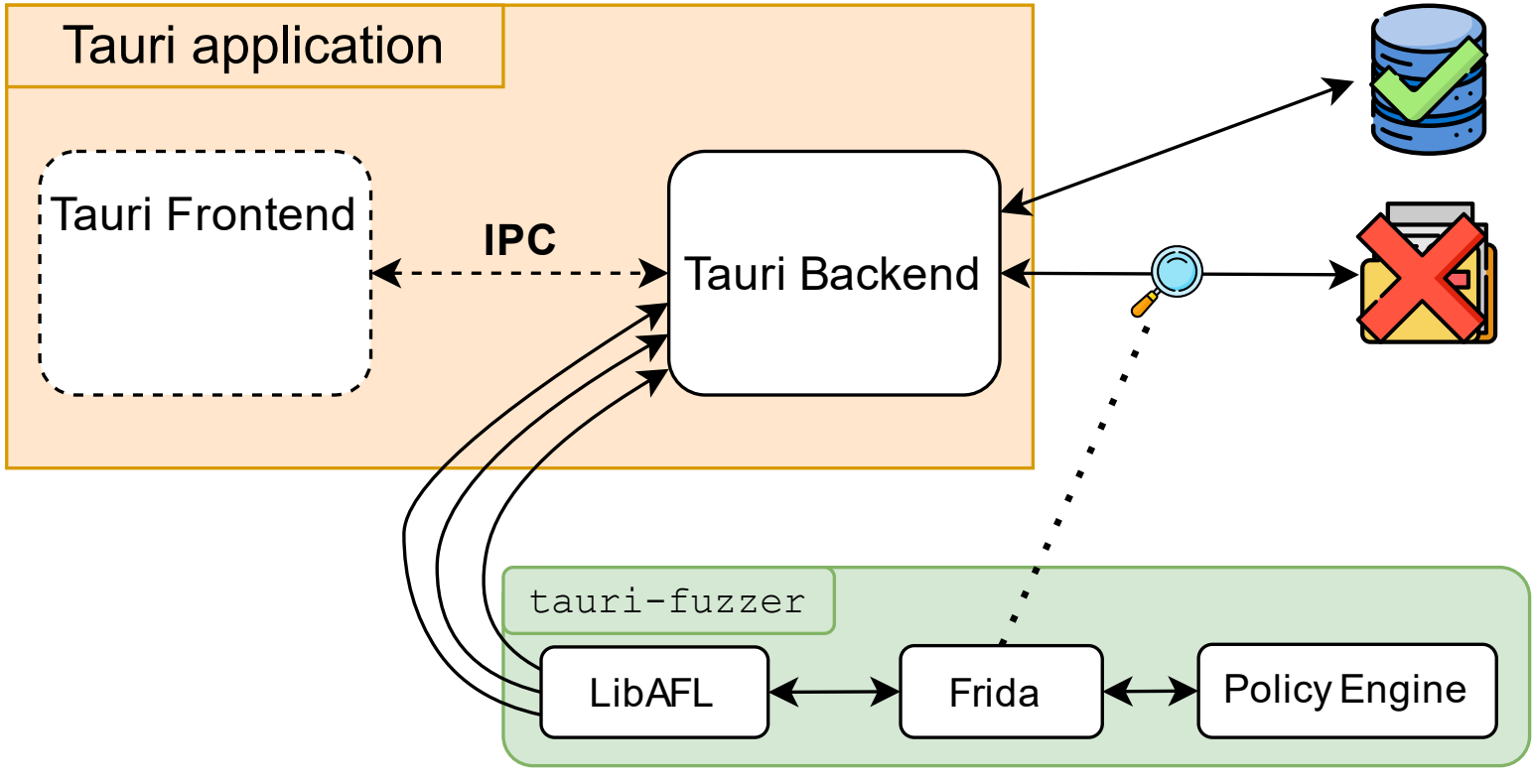
1. DEFINE A SECURITY POLICY



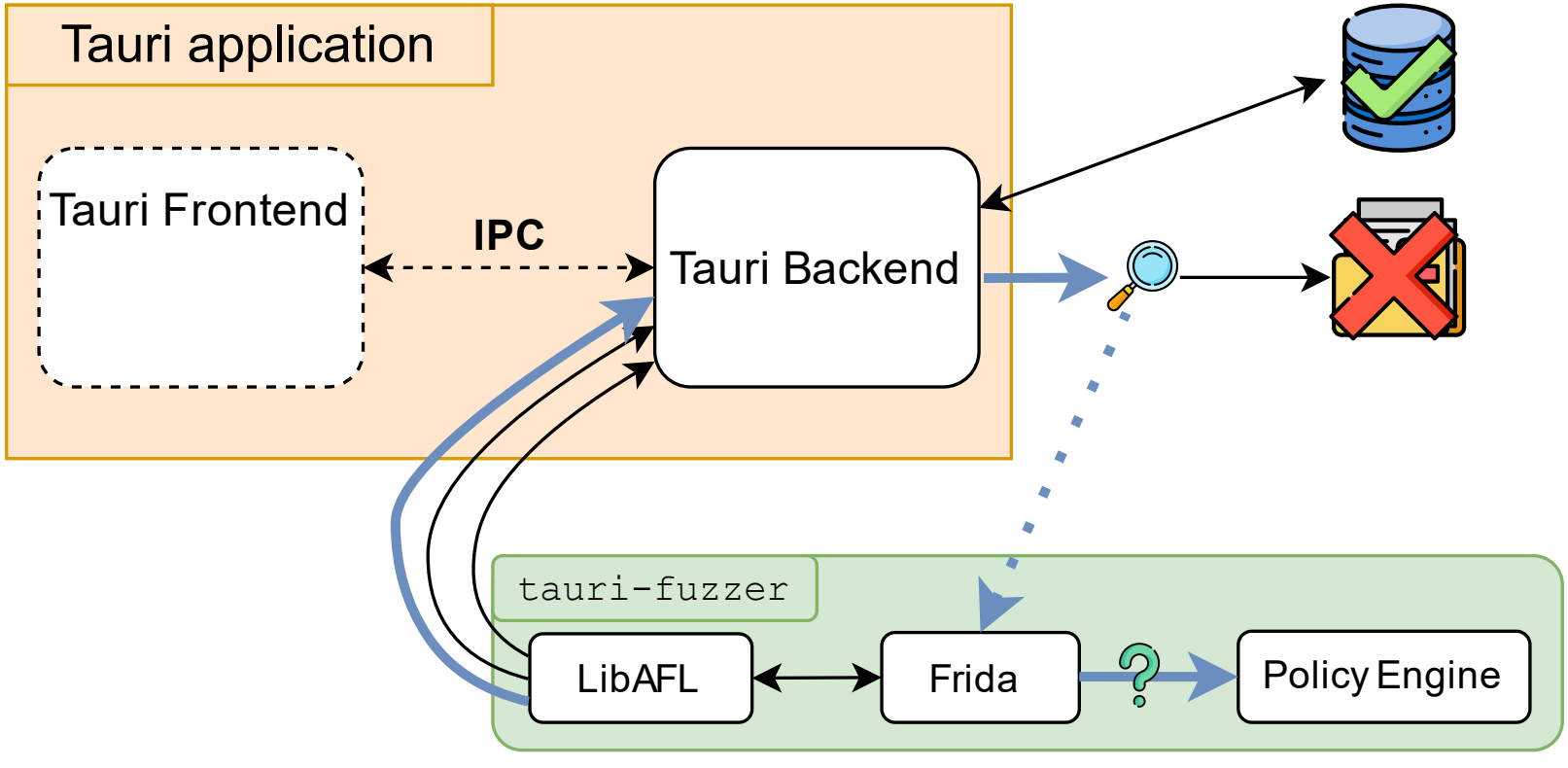
2. FRIDA INSTALL RESOURCE MONITORS



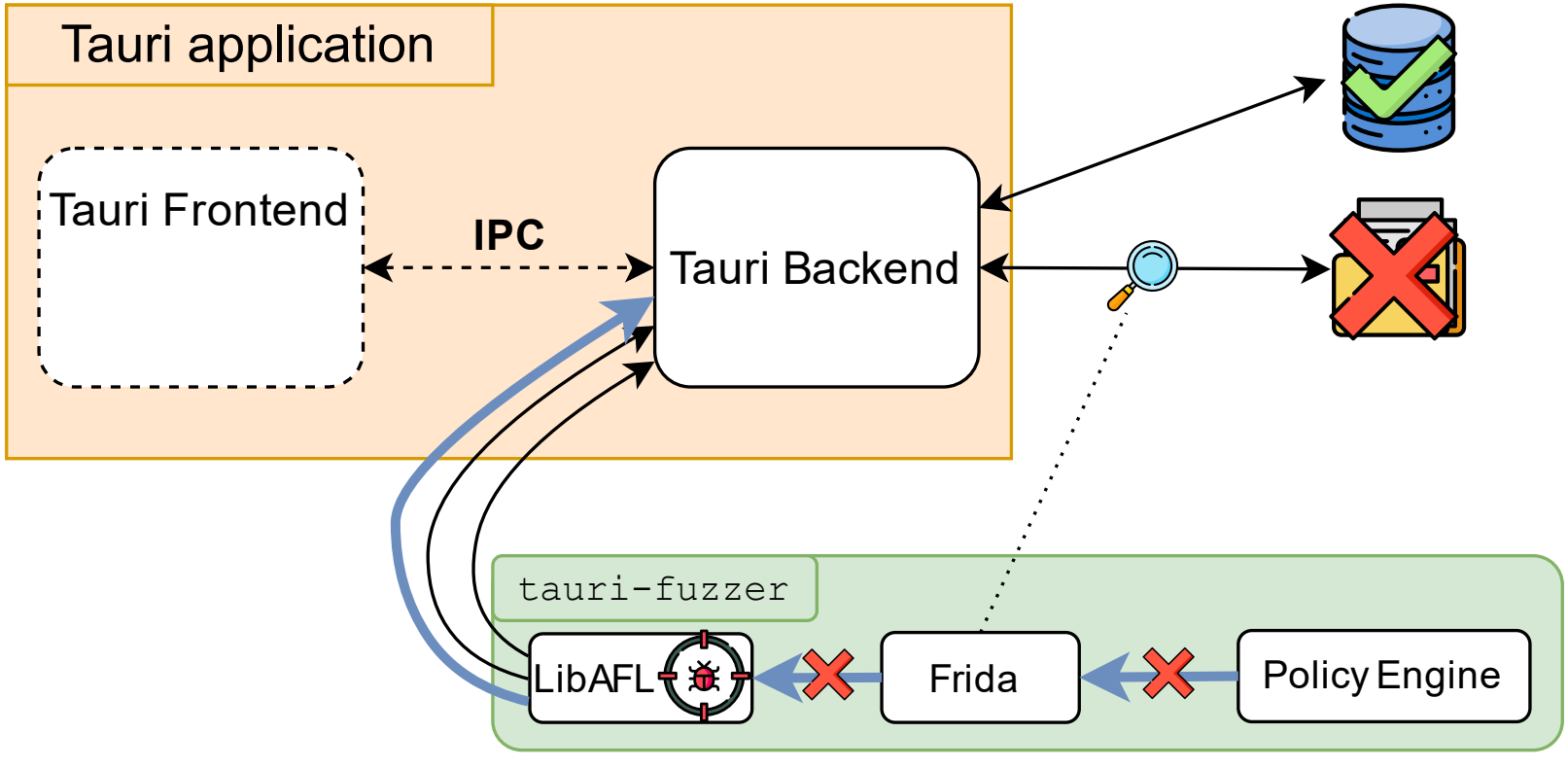
3. LIBAFL FUZZ THE TAURI APP BACKEND



4. ILLEGAL ACCESS TO THE RESOURCES ARE INTERCEPTED



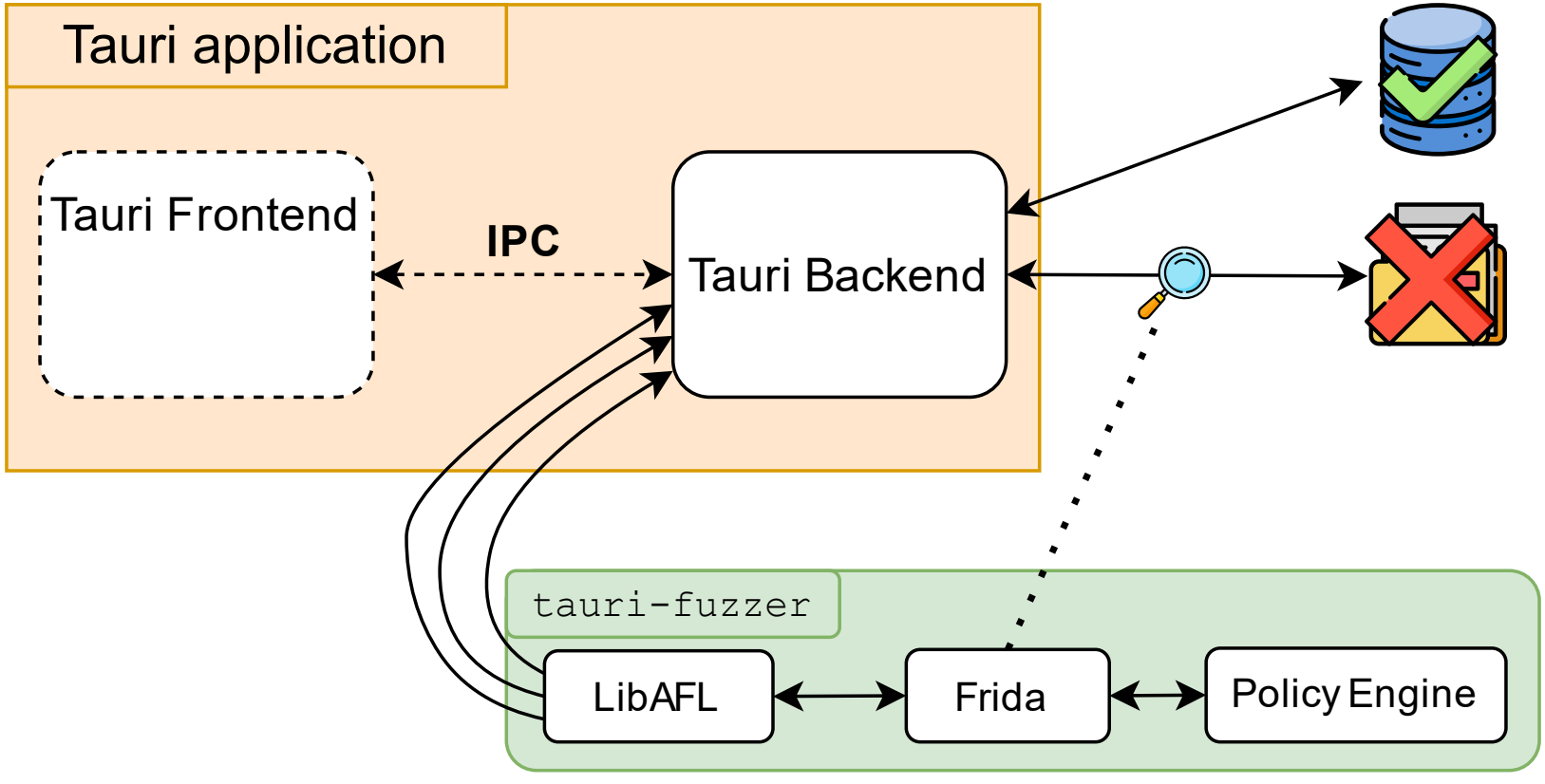
5. VULNERABILITY IS FOUND AND RECORDED



tauri-fuzzer COMPONENTS

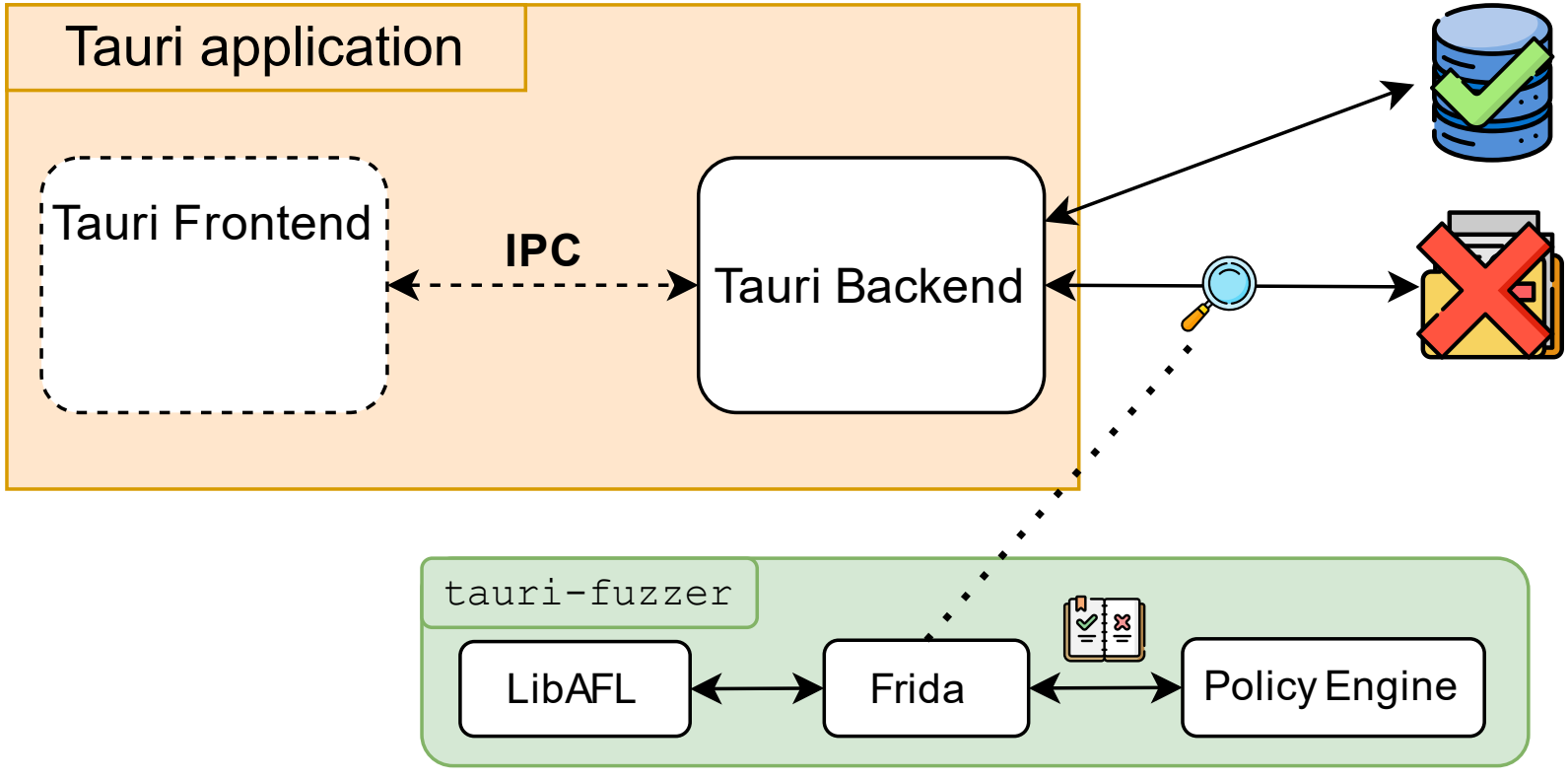
LIBAFL 

- State of the art fuzzer framework with active research
- Generate inputs, record solutions, manage multi-cores...


Tauri application
Tauri Frontend
IPC
Tauri Backend
tauri-fuzzer
LibAFL
Frida
Policy Engine

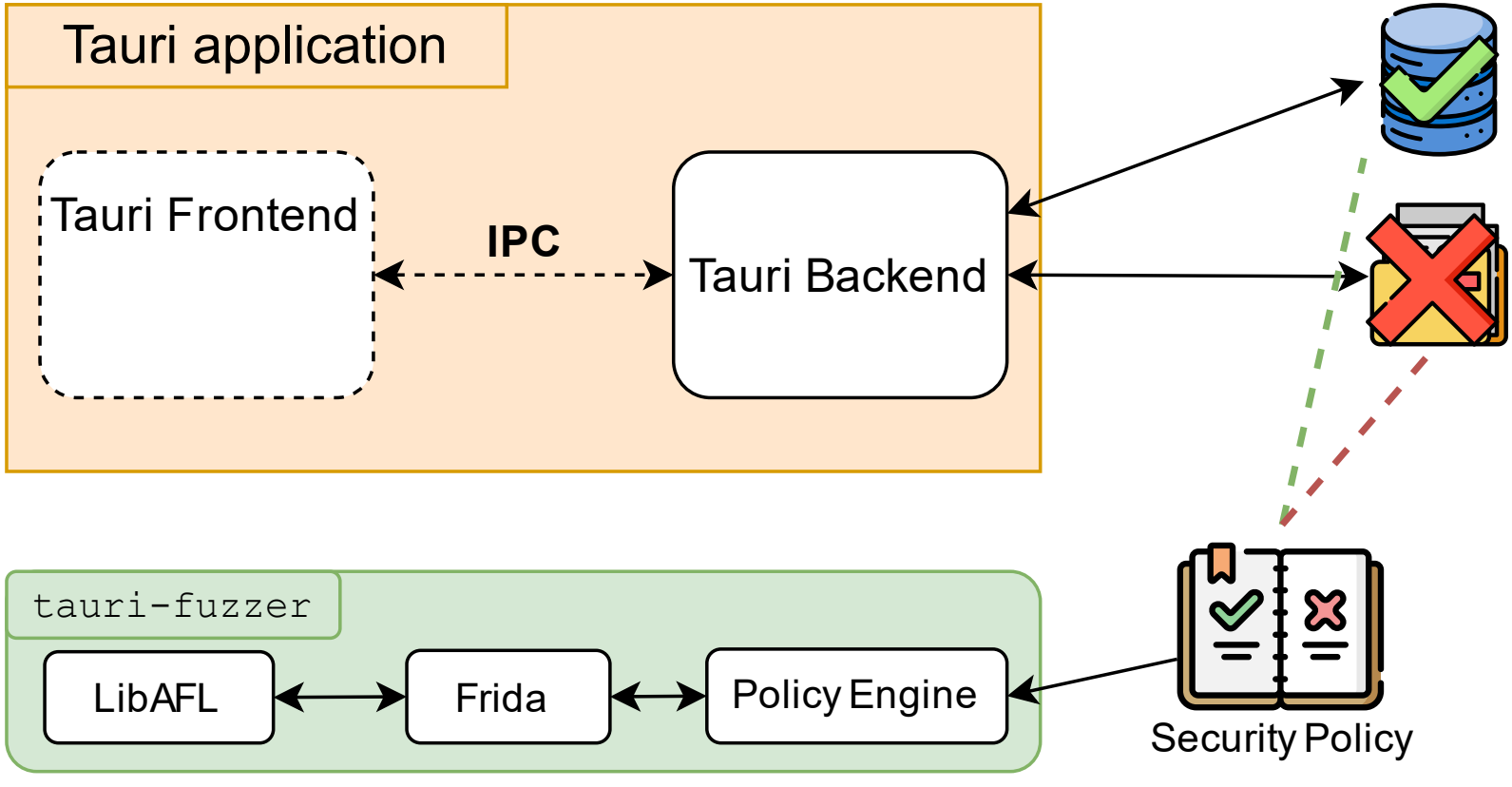
FRIDA

- Dynamic binary rewriter
- Monitor the code accessing the resources
- Portable to all major platforms


Tauri application
Tauri Frontend
IPC
Tauri Backend
tauri-fuzzer
LibAFL
Frida
Policy Engine

POLICY ENGINE

- API to easily write a security policy (*TODO*)
- Check at runtime if security policy is enforced


Tauri application
Tauri Frontend
IPC
Tauri Backend
tauri-fuzzer
LibAFL
Frida
Policy Engine
Security Policy

FUZZING A TAURI APP MADE AS EASY AS POSSIBLE

SETUP FUZZING WITH ONE COMMAND (*TODO*)

1. Create a *fuzz* directory in Tauri app project
2. Parse the Tauri project and look for Tauri commands

```
#[tauri::command]
fn foo_command(...) {}
```

3. Generate fuzzer scripts to fuzz these Tauri commands

DEFAULT SECURITY POLICIES

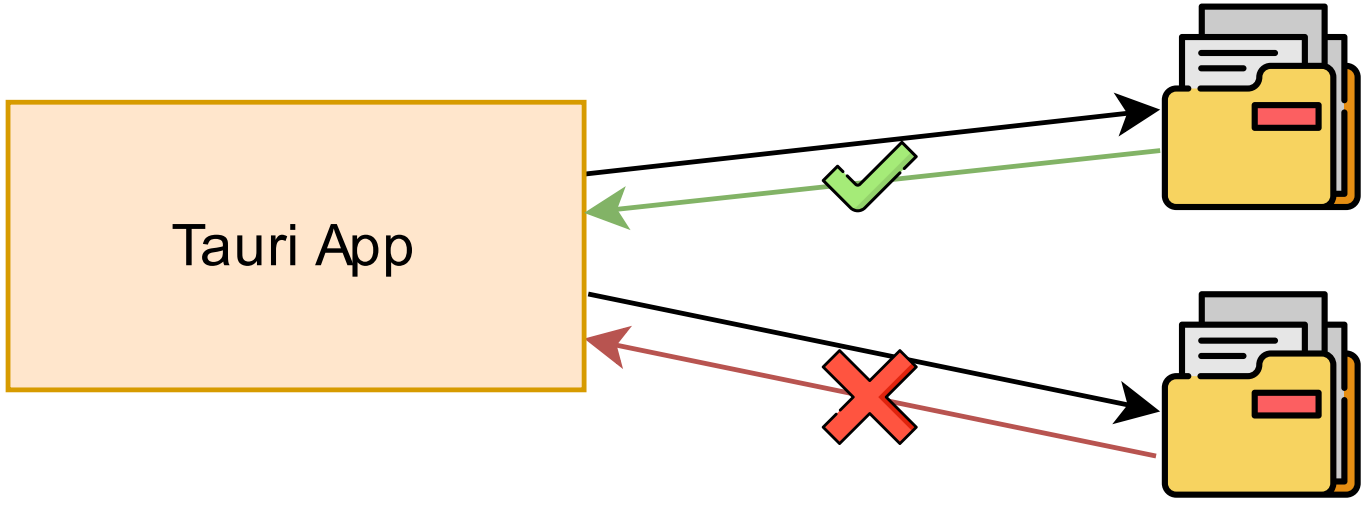
- Security policies can be cumbersome to write

- 2 available default security policies (TODO)

  1. Access to resources can't return an error
  2. Derive a security policy from the Tauri allowlist configuration

DEFAULT SECURITY POLICIES

Access to resources can't return an error



Efficient for input validation vulnerability

DEFAULT SECURITY POLICIES

Derive a security policy from the Tauri allowlist configuration

tauri.conf.json

```
"tauri": {
  "allowlist": {
    "fs": {
      "readFile": true,
      "writeFile": false,
    },
  },
  ...
```

Automatically generated security policy specific to the Tauri app

CHECK OUT THE WORK

- Repository: https://github.com/crabnebula-dev/tauri-fuzzer
- Documentation
  - mdbook: https://github.com/crabnebula-dev/tauri-fuzzer/tree/main/docs
  - Outline (soon)

WHAT SHOULD YOU REMEMBER FROM THIS PRESENTATION?

Problem — *Fuzzing is not used enough for app development*

Goal — *Facilitate fuzzing for app development*

How? — *Provide a security policy checker suited to app fuzzing*

Make fuzzing Tauri app as easy as possible