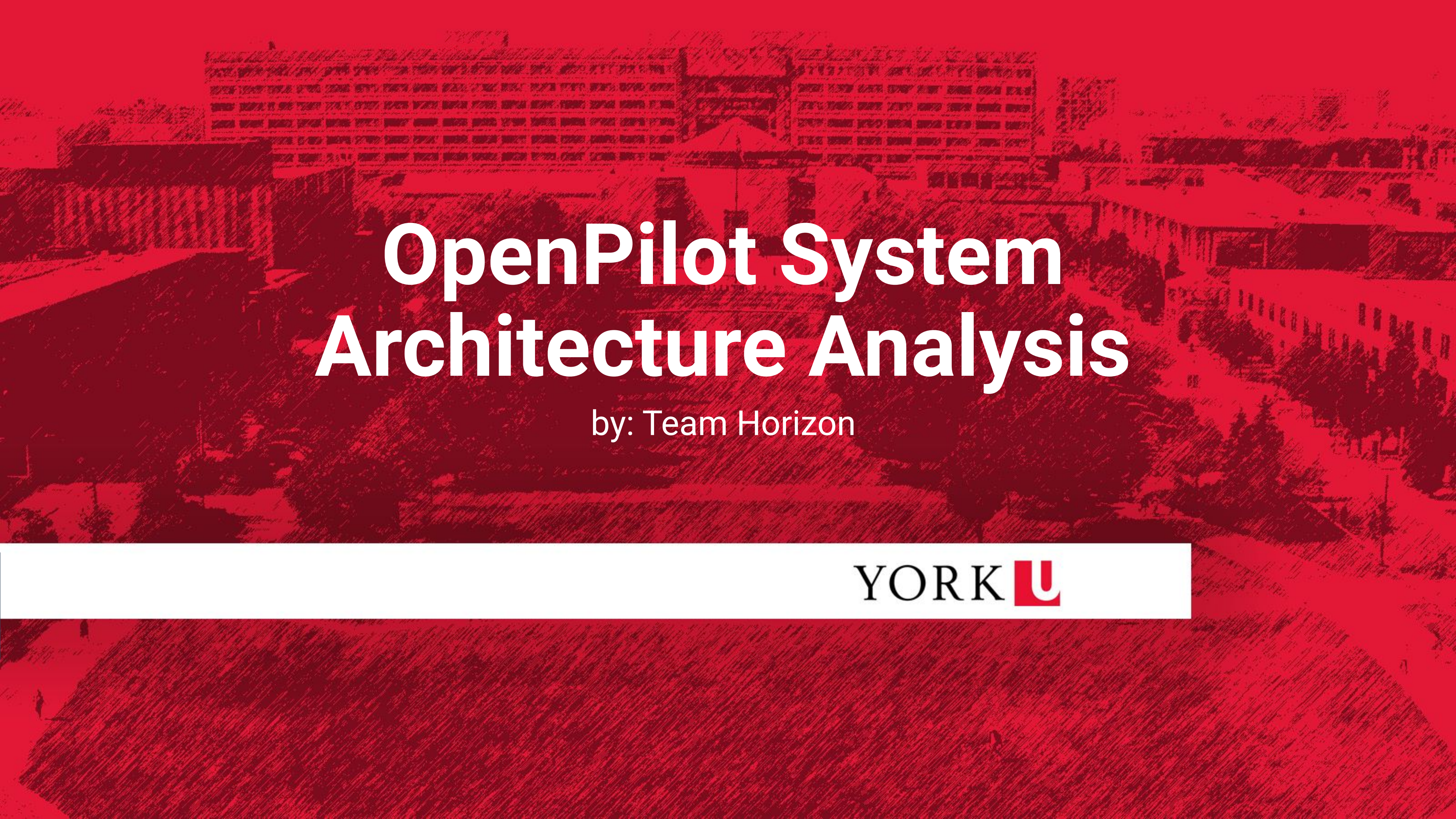


OpenPilot System Architecture Analysis

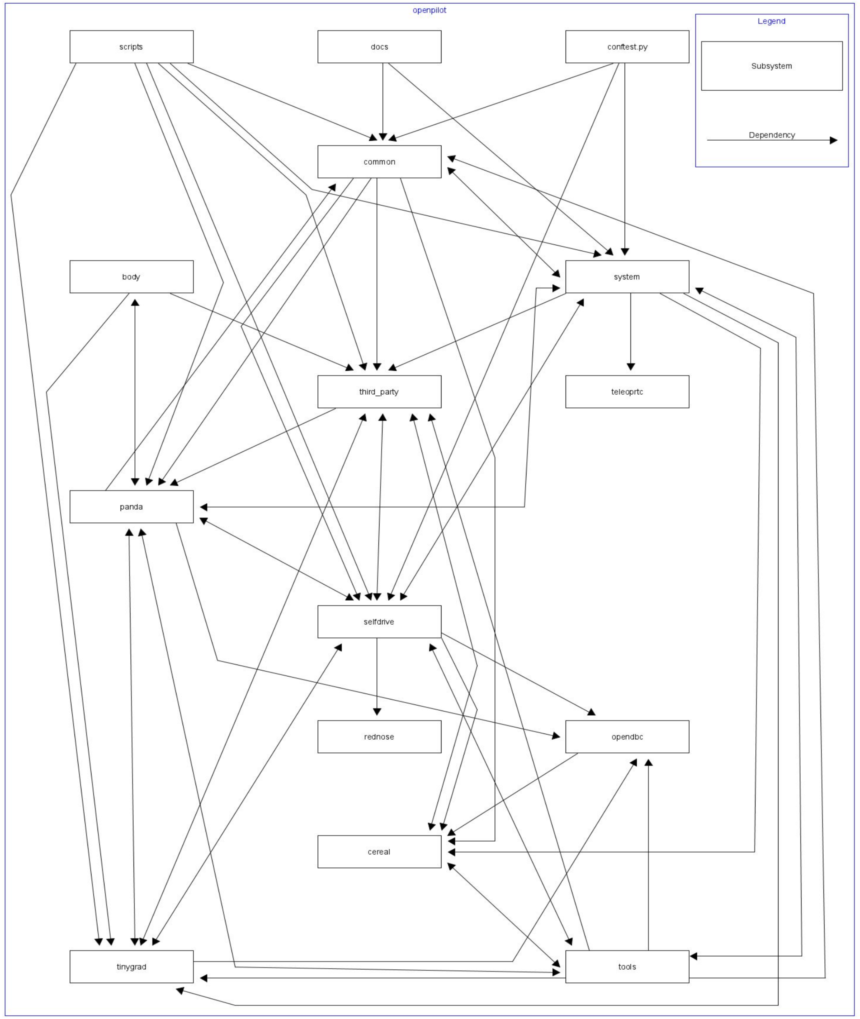
by: Team Horizon

YORK U

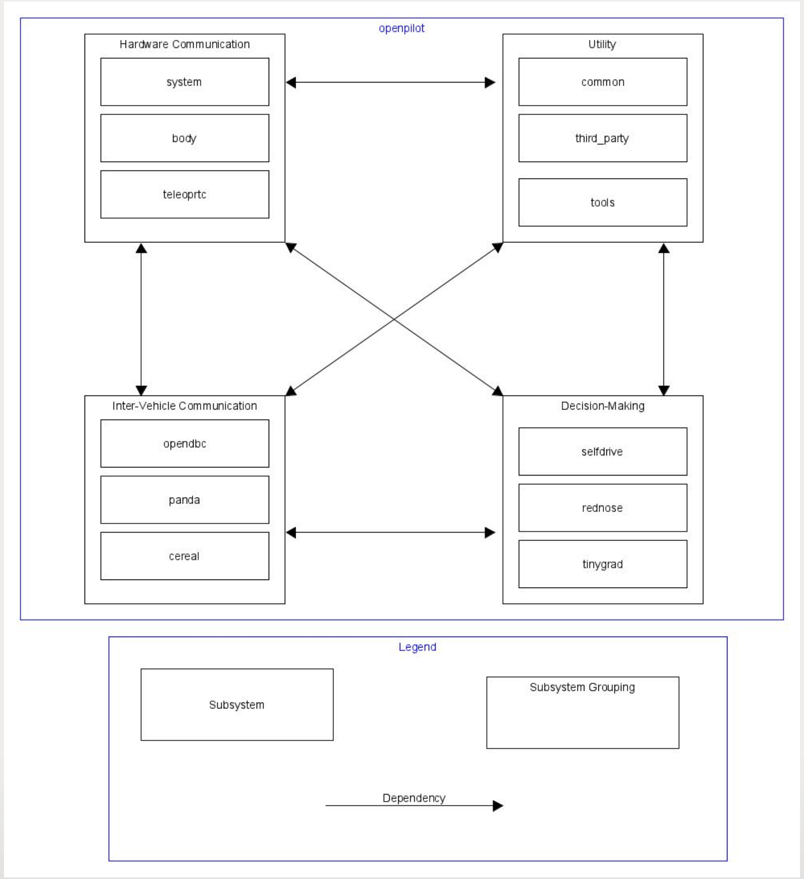
Agenda

YORK U

Concrete Architecture



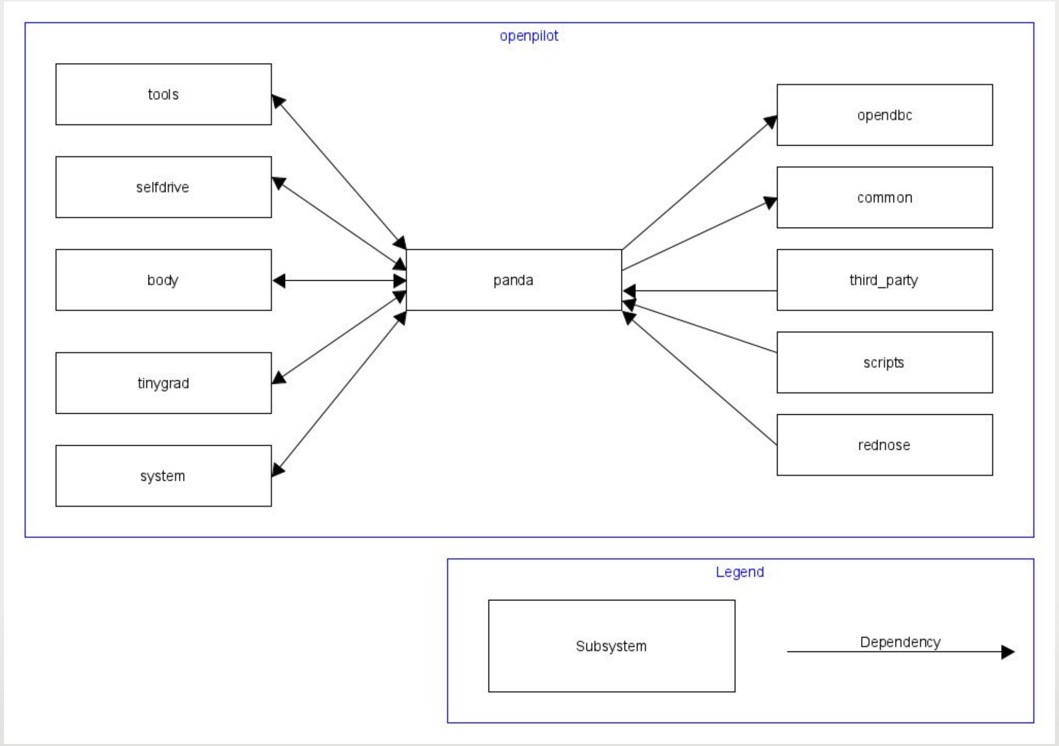
Concrete Architecture



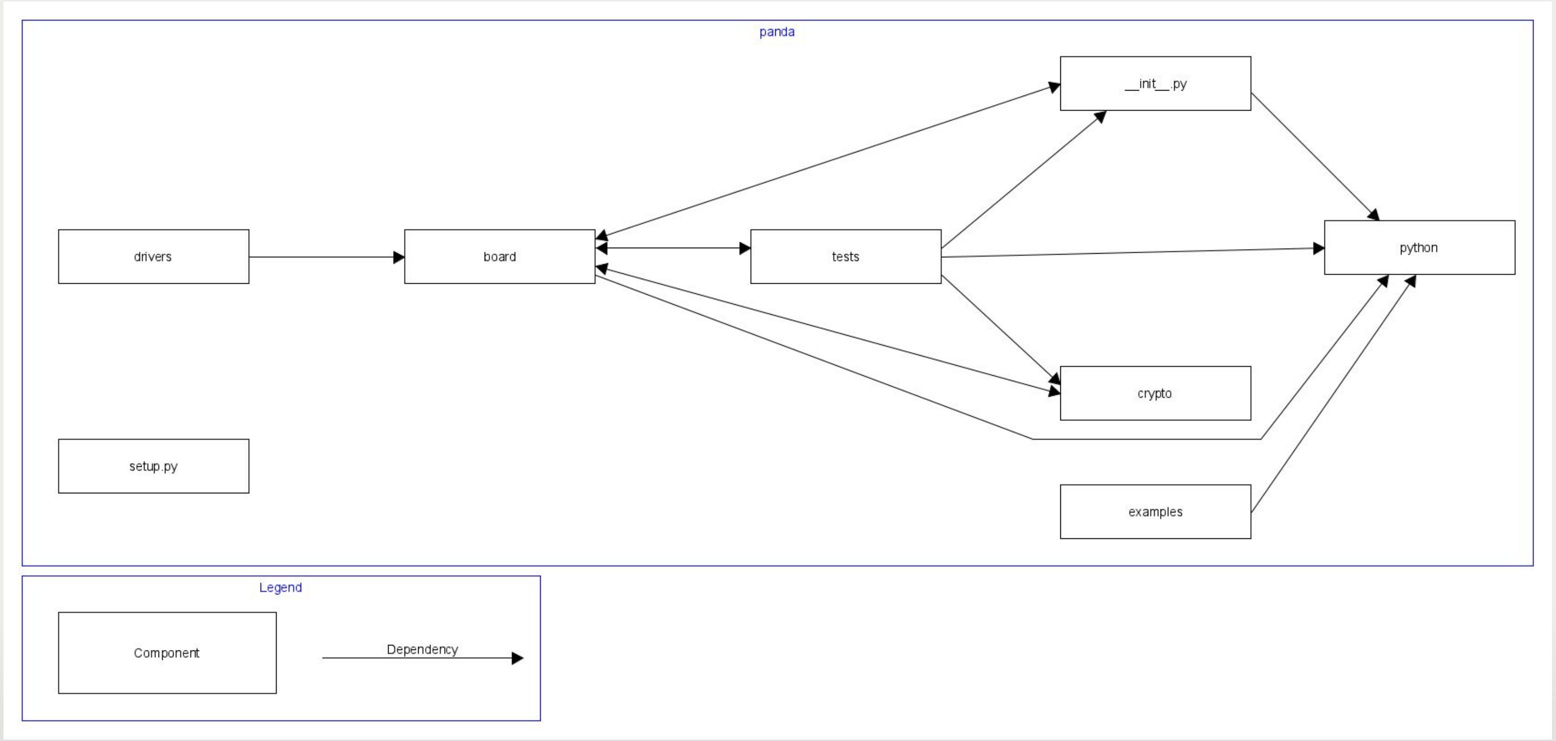
Subsystem Selection: panda

- The Panda subsystem is a key component of openpilot, facilitating communication between various hardware components and the openpilot software system
- Panda acts as a bridge between the vehicle's Controller Area Network (CAN) bus and the openpilot software system
- Panda is a small electrical device that connects to the vehicle via USB, enabling communication with sensors and actuators for autonomous driving
- Panda seamlessly integrates with openpilot software, providing an interface for accessing vehicle data and sending commands to actuators.
- Additional features:
    - monitor the health and status of different hardware components
    - enforces safety measures to ensure a secure driving environment.

Subsystem Architecture



Subsystem Architecture



Subsystem Design Patterns

- No noteworthy design patterns
- Primarily Looked into 5 design patterns:
  - Adapter
    - Potential interface adaptation for different Panda devices or cars
  - Observer
    - Potential use in updating when new sensor data is received
  - Iterator
    - Potential custom collection use.
  - Template
    - Potential structuring of algorithms as skeletons with specific operations defined in subclasses
  - State
    - Potential structuring of state machines with objects instead of conditionals

Subsystem Design Patterns

C Code

- Not an object-oriented programming language
  - Use of OODPs is less likely due to lack of support for their implementation
- Could not find any clear uses of OODPs
  - Ex. States are represented as unsigned integers and managed with switch statements
    - No use of State OODP

Python Code

- Minimal use of inheritance
  - Custom exceptions
  - Abstract handle classes for communicating with Panda device
    - extended to make handle classes for CAN, USB, and SPI communication
  - Panda and PandaDFU classes for interfacing with Panda device
    - Extended by PandaJungle and PandaJungleDFU classes for interfacing with Panda Jungle debugging device
  - No involvement in implementing design patterns

External Interfaces

- The primary interface in this ecosystem is the Controller Area Network, or CAN, along with CAN Flexible Data messages.
- OpenDBC files act as a translation layer, allowing OpenPilot to interpret and interact with the vehicle's diverse CAN network."
- Cereal communication protocol is employed to maintain structured data exchange for driver assistance features
- SHA and RSA files are transmitted by Panda to hash and secure its messages, preventing unauthorised access to vehicle controls

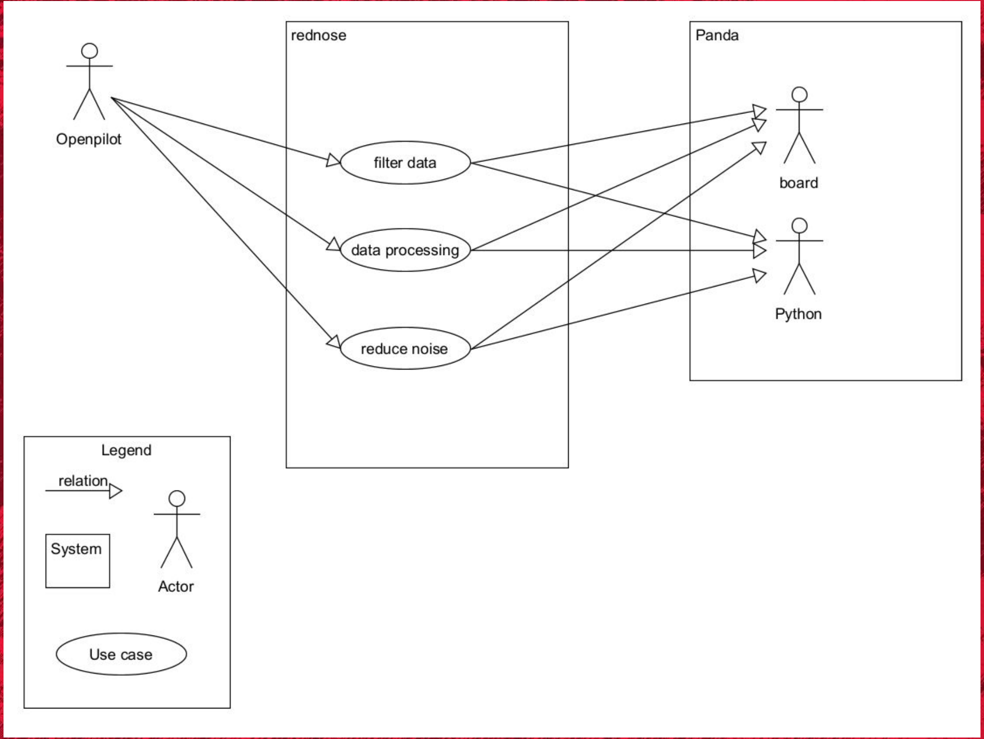
Concurrency

- Implicit Invocation Style

- Some concurrency
    - Encryption
    - Serial Reading
    - Information Sending

Use Cases

- Board: translating sensor data
- Tinygrad, selfdrive, and rednose: decision-making
- Boardd, cereal, and openDBC: decoding and for vehicle control



Lessons Learned

- Understand - Making use of Understand Software to view and gain insight into open-source system structure.

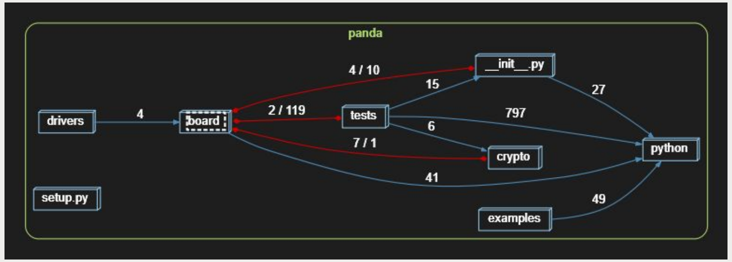
- Subsystem Grouping - Grouping components further than just looking at directory hierarchy.
    - Group into categories that relate to associated behaviour:
        - ➔ Decision-Making
        - ➔ Inter-Vehicle Communication
        - ➔ Hardware Components
        - ➔ Utility

Lessons Learned

- Deep-Dive into Panda
  - Investigation into specific components and their relationships allow for a further understanding of each components purpose.
- Revise previous idea about openpilot structure, including architecture styles and design patterns.
  - Implicit Invocation Architecture Style between components, allowing for loose coupling.
  - Lack of obvious design patterns (and scarce use of inheritance)
- Specific Choices Made
  - Hashing, encryption, internal IDs, prioritization.

Conclusion

- Openpilot's architectures are key to its functionality
- The Panda subsystem is crucial for interfacing with vehicle controls
- Design patterns play a significant role in system robustness and scalability, didn't find any
- Concurrency and real-time processing are cleverly managed



Live Demo

Thank you for your attention!

Any questions/ suggestions/ concerns?