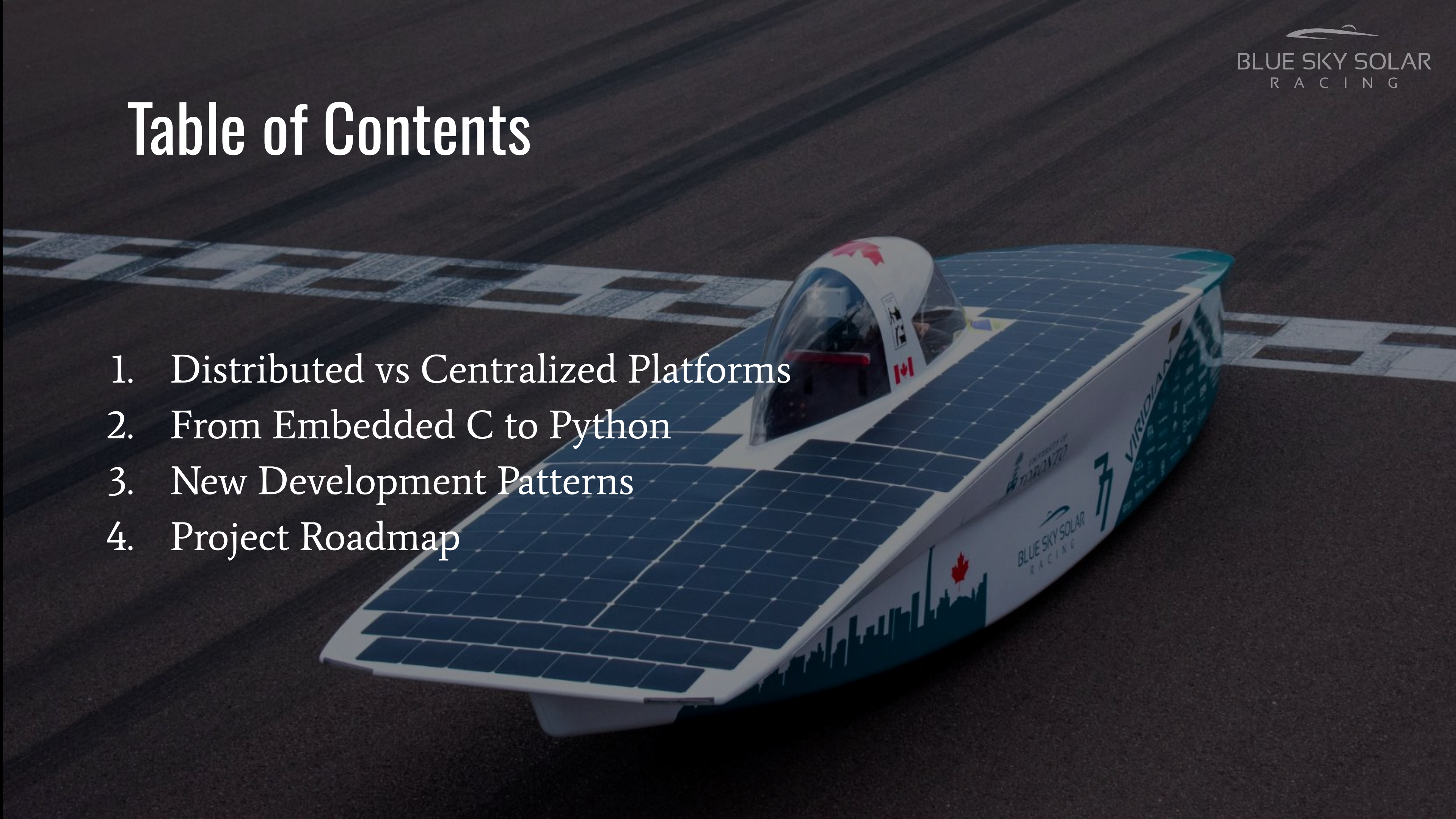
BLUE SKY SOLAR
RACING

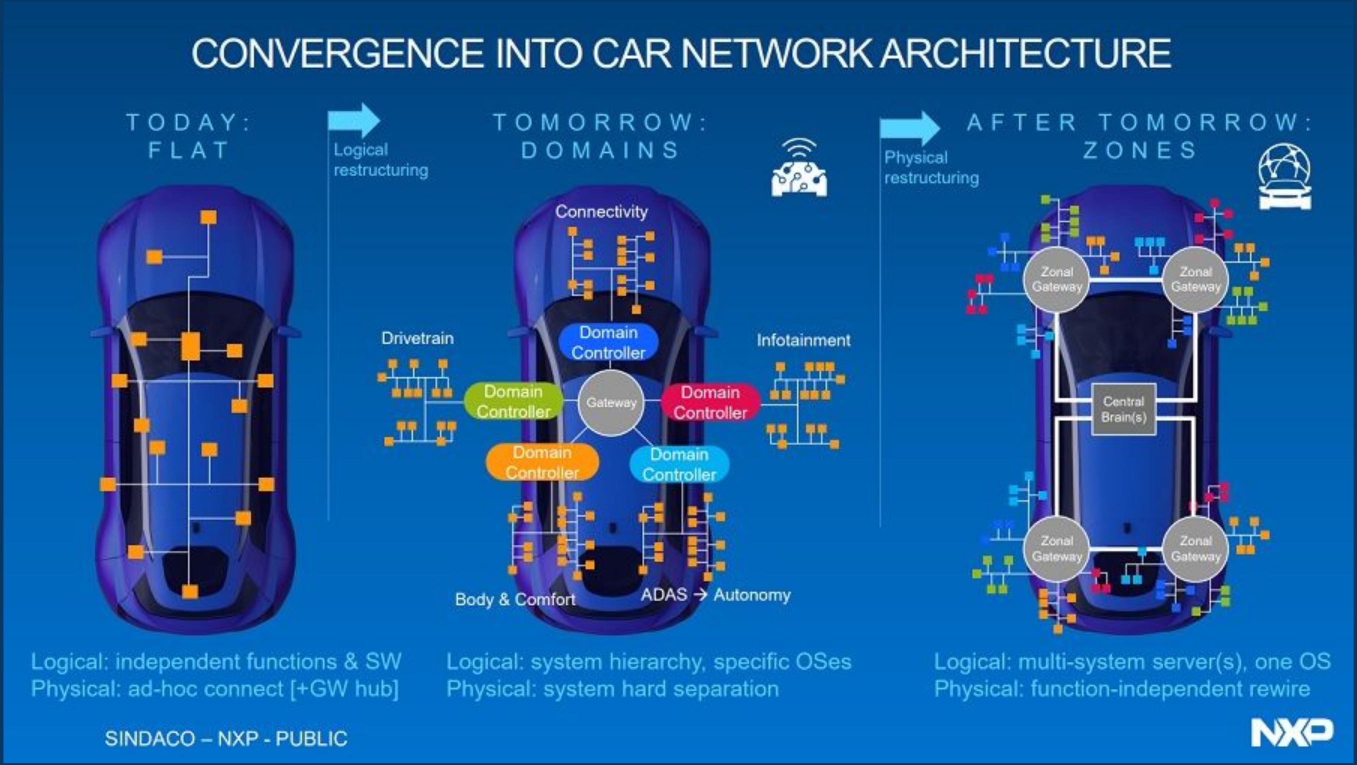
2023 Advances in Blue Sky Solar Racing Software

Table of Contents



1. Distributed vs Centralized Platforms
2. From Embedded C to Python
3. New Development Patterns
4. Project Roadmap

Decentralized vs Centralized Platforms



CONVERGENCE INTO CAR NETWORK ARCHITECTURE

TODAY: FLAT

Logical restructuring

TOMORROW: DOMAINS

Physical restructuring

AFTER TOMORROW: ZONES

Logical: independent functions & SW
Physical: ad-hoc connect [+GW hub]

Logical: system hierarchy, specific OSes
Physical: system hard separation

Logical: multi-system server(s), one OS
Physical: function-independent rewire

SINDACO – NXP - PUBLIC

NXP

Examples of Centralized Platforms

Nvidia: THOR

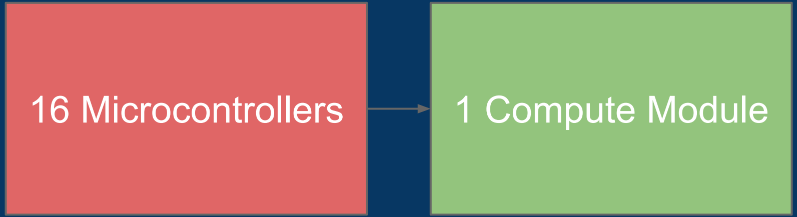
STELLANTIS
STLA Brain
Powered by
Snapdragon
digital chassis
Snapdragon Digital Chassis is a product of Qualcomm Technologies, Inc. and/or its subsidiaries.

Qualcomm: Digital Chassis

BLUE SKY SOLAR
RACING

Why Blue Sky needs a Centralized Platform

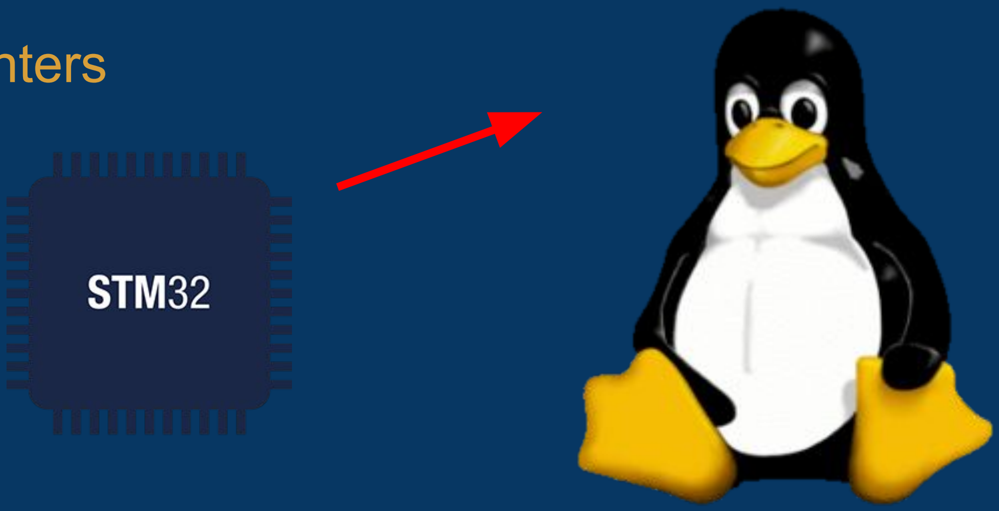
- **75%* less** programmable chips
- **70%* less** wiring
- **50%* less** power consumption
- **More** software flexibility
    - Remote update
    - Local strategy simulation
    - Resolve state inconsistency



* Estimates

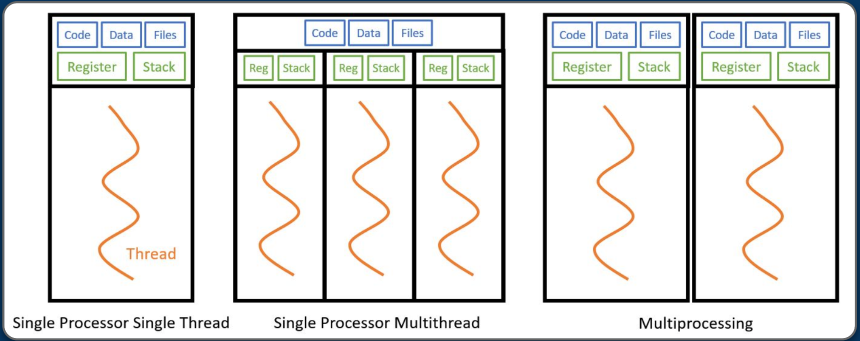
From STM to a Linux Compute Module

- Flexible & Powerful
  - Versatile choices of programming languages
  - Unlimited package and tool support
- Widely used
  - From phones to data centers
- Well Documented
- STM32 MCs reaching EOL



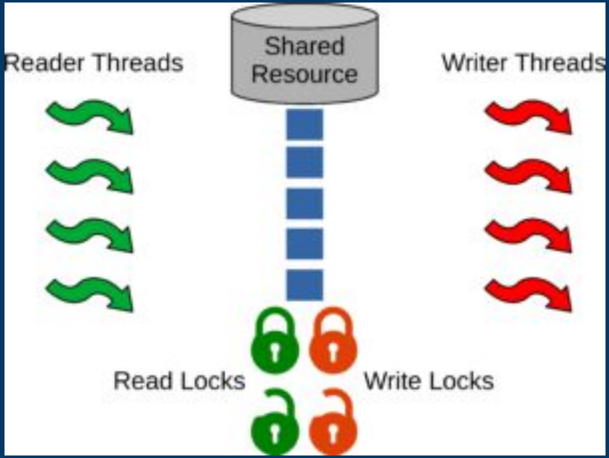
From Microcontrollers to Threads/Processes (Part 1)

- Control car’s peripherals
- Microcontrollers (old)
  - Must develop drivers for each
  - On failure, restarts MC
    - Slow
    - State takes time to sync
      - Dangerous
- Multithreading/multiprocessing (new)
  - Single consolidated program
  - On crash, respawns thread/process
    - Fast
    - State always synced



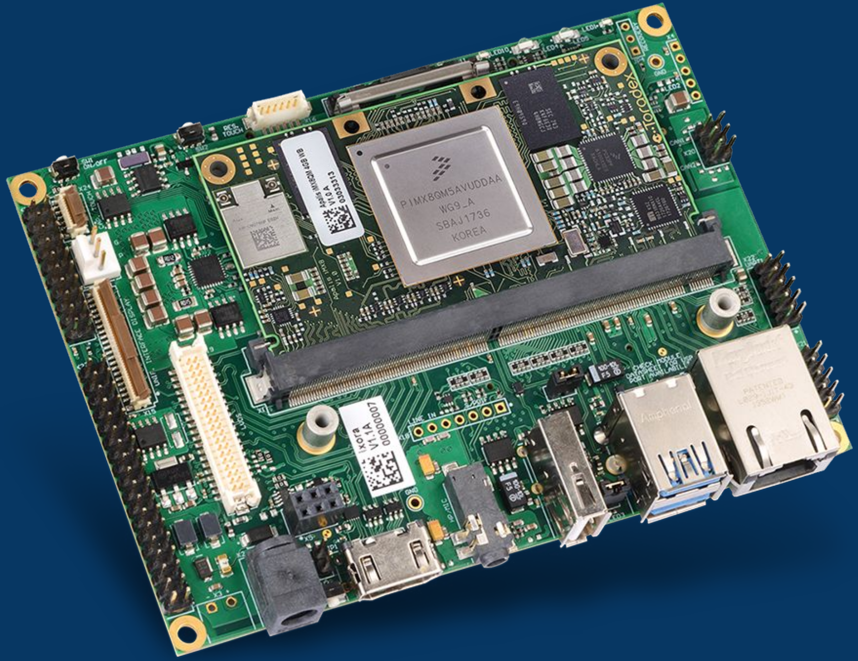
From Microcontrollers to Threads/Processes (Part 2)

- Car’s state must be in sync
- Microcontrollers (old)
    - Data transmission
        - Slow
        - Daisy chain or…
        - UARTHUB (network switch)
- Multithreading/multiprocessing (new)
    - Shared memory
        - Fast
        - Less prone to failure
        - Reader*s*-writer lock
            - ALWAYS synchronized



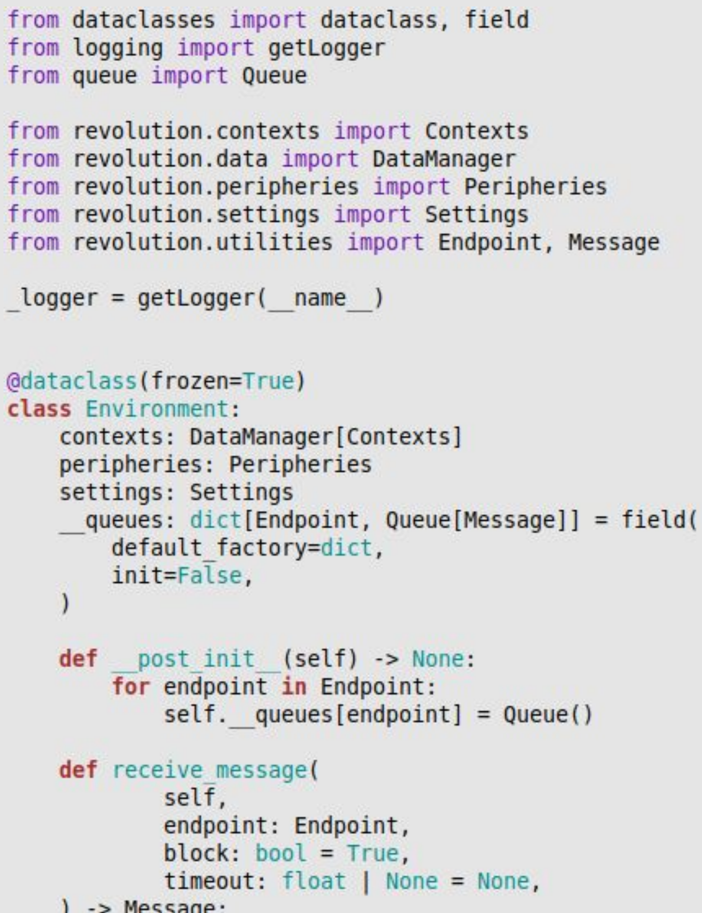
Virtual Hardware

- Simulate H/W access
  - Mocking of…
    - GPIO
    - SPI
    - PWM
    - Serial…
- Unit tests
- Integration tests



From Embedded C to Python

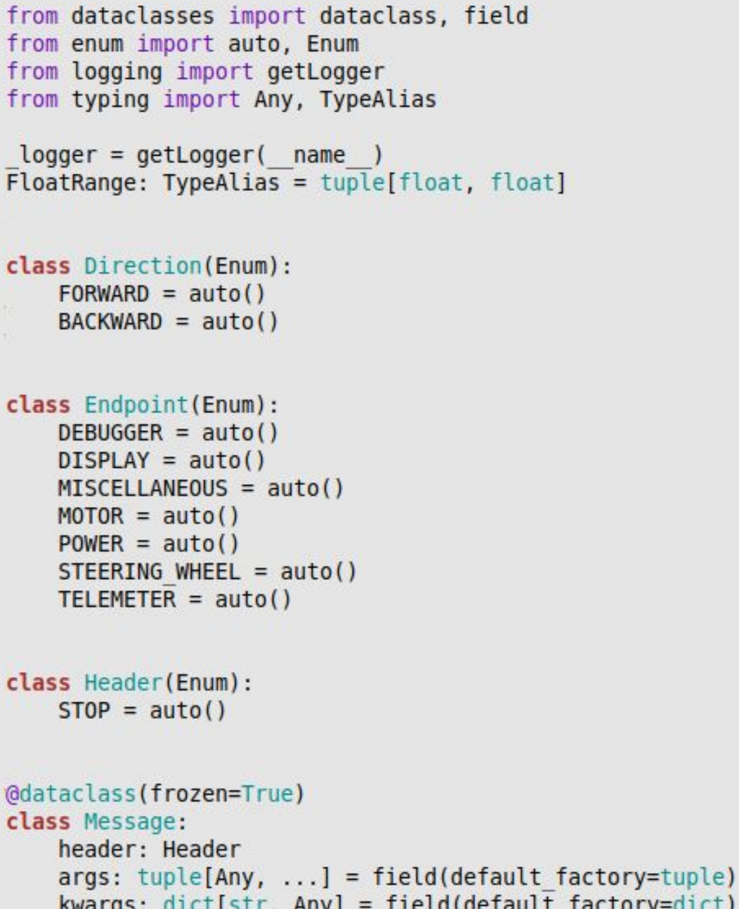
- Faster Development
- Enhanced validation
- Increased Safety & Security
- Future work
- Limitations



```python
from dataclasses import dataclass, field
from logging import getLogger
from queue import Queue

from revolution.contexts import Contexts
from revolution.data import DataManager
from revolution.peripheries import Peripheries
from revolution.settings import Settings
from revolution.utilities import Endpoint, Message

_logger = getLogger(__name__)


@dataclass(frozen=True)
class Environment:
    contexts: DataManager[Contexts]
    peripheries: Peripheries
    settings: Settings
    __queues: dict[Endpoint, Queue[Message]] = field(
        default_factory=dict,
        init=False,
    )

    def __post_init__(self) -> None:
        for endpoint in Endpoint:
            self.__queues[endpoint] = Queue()

    def receive_message(
            self,
            endpoint: Endpoint,
            block: bool = True,
            timeout: float | None = None,
    ) -> Message:
```

Faster Development

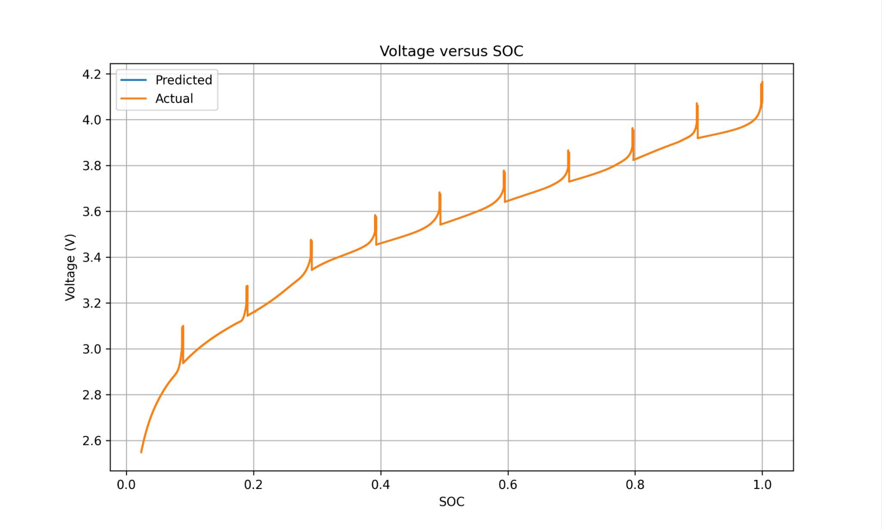
- Productivity
  - More developers
  - Easy to write
  - Easy to read
  - Python >>> C standard library
- Concise code
  - “It’s a beautiful thing, the destruction of [lines].”
- Rich ecosystem
  - No “reinventing the wheel”



```
from dataclasses import dataclass, field
from enum import auto, Enum
from logging import getLogger
from typing import Any, TypeAlias

_logger = getLogger(__name__)
FloatRange: TypeAlias = tuple[float, float]


class Direction(Enum):
    FORWARD = auto()
    BACKWARD = auto()


class Endpoint(Enum):
    DEBUGGER = auto()
    DISPLAY = auto()
    MISCELLANEOUS = auto()
    MOTOR = auto()
    POWER = auto()
    STEERING_WHEEL = auto()
    TELEMETER = auto()


class Header(Enum):
    STOP = auto()


@dataclass(frozen=True)
class Message:
    header: Header
    args: tuple[Any, ...] = field(default_factory=tuple)
    kwargs: dict[str, Any] = field(default_factory=dict)
```

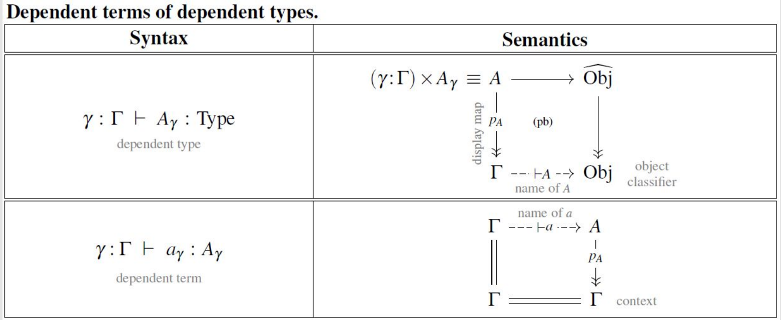
Case Study: SOC Estimation

- Advances in Gen 12…
  - From 719 lines to 117 lines
    - **83% less** lines
  - From 29698 chars to 3515 chars
    - **88.2% less** chars
- Use filterpy
  - Library specialized for filtering
  - Less bug
  - Enhanced accuracy
  - No need to implement EKF algo



Enhanced Validation (Part 1)

- Gen 11 codebase is hard to understand
  - Lots of copy & paste programming
  - Lack of consistent style
  - Beyond salvation…
- Static type checking
  - Python supports type annotations
    - Backed by SOTA type/set theory
    - Mypy static type checker
      - Strict flag
        - Stricter and safer than C
  - C’s type system is stuck in the 70s

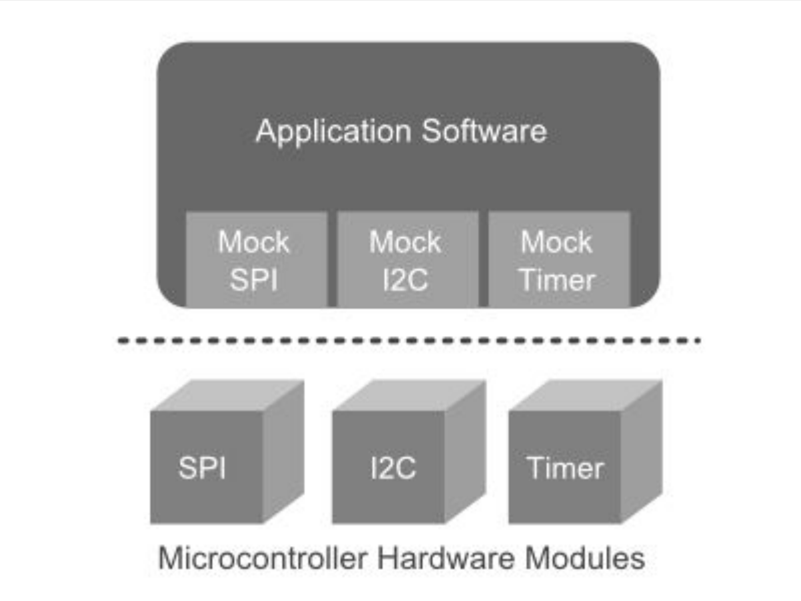


Dependent terms of dependent types.

Syntax	Semantics
$\gamma : \Gamma \;\vdash\; A_\gamma : \mathrm{Type}$ dependent type	$\begin{tikzcd} (\gamma:\Gamma)\times A_\gamma \equiv A \arrow[r] \arrow[d, two heads, "p_A", "\text{display map}"'] \arrow[dr, phantom, "\text{(pb)}"] & \widehat{\mathrm{Obj}} \arrow[d, two heads] \\ \Gamma \arrow[r, dashed, "\vdash A", "\text{name of } A"'] & \mathrm{Obj} & \text{object classifier} \end{tikzcd}$
$\gamma : \Gamma \;\vdash\; a_\gamma : A_\gamma$ dependent term	$\begin{tikzcd} \Gamma \arrow[r, dashed, "\vdash a" description, "\text{name of } a"] \arrow[d, equal] & A \arrow[d, two heads, "p_A"] \\ \Gamma \arrow[r, equal] & \Gamma & \text{context} \end{tikzcd}$

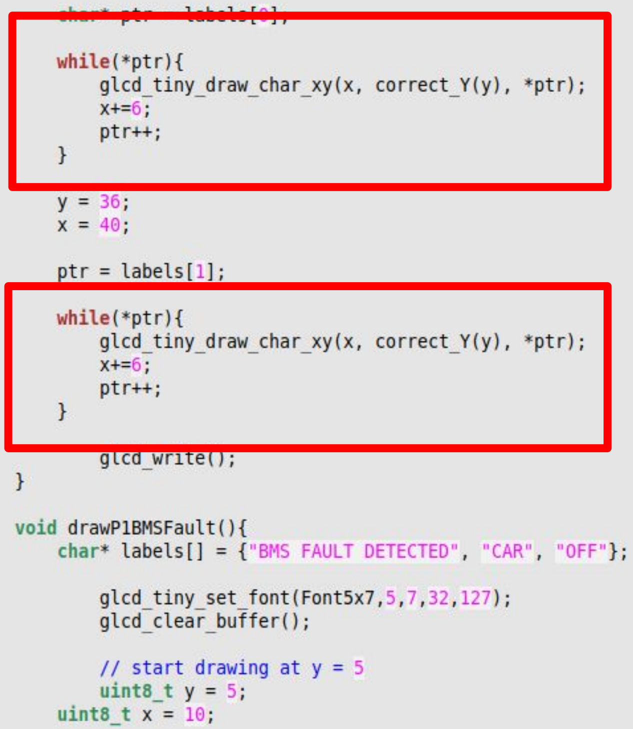
Enhanced Validation (Part 2)

- Gen 12 follows PEP 8
  - Easy to read and proofread
- Unit tests & integration tests
  - Gen 11 did not have any
  - Mocking hardware access



Case Study: Display driver

- Gen 11 is full of copy & paste programming
  - Dozens of examples
- Gen 12 endorses taut, lean modular code

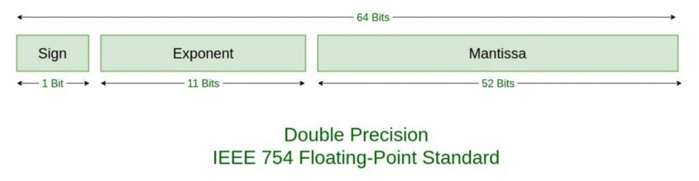


```
    char* ptr = labels[0];

    while(*ptr){
        glcd_tiny_draw_char_xy(x, correct_Y(y), *ptr);
        x+=6;
        ptr++;
    }

    y = 36;
    x = 40;

    ptr = labels[1];

    while(*ptr){
        glcd_tiny_draw_char_xy(x, correct_Y(y), *ptr);
        x+=6;
        ptr++;
    }
        glcd_write();
}

void drawP1BMSFault(){
    char* labels[] = {"BMS FAULT DETECTED", "CAR", "OFF"};

        glcd_tiny_set_font(Font5x7,5,7,32,127);
        glcd_clear_buffer();

        // start drawing at y = 5
        uint8_t y = 5;
    uint8_t x = 10;
```

BLUE SKY SOLAR RACING

State synchronization

- Gen 11
  - Propagated via UARTHUB
    - Blue sky transmission control protocol
  - Risk of uncontrolled message generation
  - Various “float” data sent as “int” -> loss of detail, can be fatal
- Gen 12
  - Shared memory
  - Readers-writer lock
    - Multiple simultaneous reads allowed
    - Multiple simultaneous writes prohibited
    - Simultaneous read and write prohibited



Safety and Security

- C is dangerous and unsafe
    - As per the well-known NSA memo
    - Relies on programmers to not make mistakes
- Python is safer
    - Worry less about memory

Future Work

- Potential for experimentation with…
  - AI workload
    - Gen 13?
  - A lightweight API server
    - Debugging
    - Monitoring
    - Hotfixes

BLUE SKY SOLAR RACING

Limitations

- Speed
  - Python is 10-100 times slower than C
    - We don’t have any heavy computation
    - Efficiently written Python code closes the gap
    - Leverage C interop of Python
      - Offload numerical calculations to numpy, etc.
      - Low-level memory access
    - Revolution is mostly IO bound
- Global Interpreter Lock (GIL)
  - Use multiprocessing in possible places to overcome

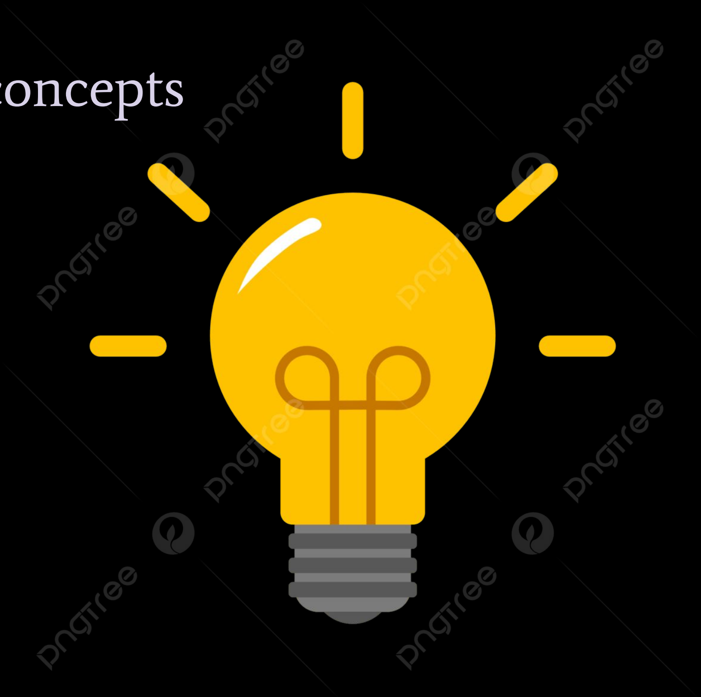
New Development Patterns

- From a single repository to multiple repositories
  - Separation of distinct ideas
  - Project repo “ownership” by individual members
  - Lean and taut repository

BLUE SKY SOLAR
RACING

Separation of Distinct Ideas

- Specialization of isolatable works
- Enrichment and maturity of individual concepts
- Enhanced validation
  - Relevant unit tests



BLUE SKY SOLAR
RACING

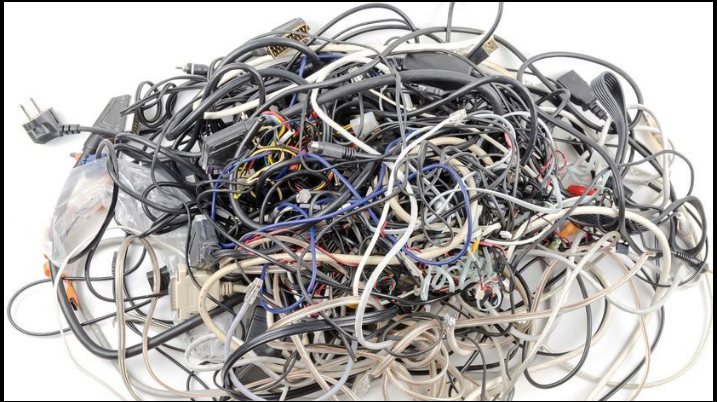
Project Ownership by Members

- Increased sense of duty for their work
- Specialization into different parts of the vehicle
- Development in their own pace

Lean and Taut Repositories

- Acceptable to develop in the main branch
  - Less PRs
  - Less merge conflicts
- Faster development
- Versatile release cycle
  - Individual components no longer in “lockstep”

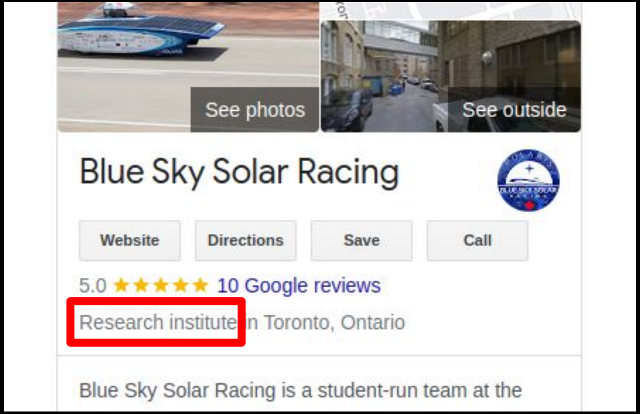
Case Study: Borealis



- 24 branches
  - Many stale or inactive
  - Endless merge conflicts
  - Most out of date
- Giant repository of distant peripherals
  - Most are isolatable
  - Entire team forced to develop on a single repo
  - Accidents can be devastating

Usages by 3rd Parties

- Each task/idea now a separate component
- Matures into libraries
  - In use outside of Blue Sky
- Any bugs or issues can be reported by others
  - Benefits Blue Sky
- Contribution to the open source space
  - Blue Sky is a publicly funded research organization
- Various top teams already do this



Blue Sky Solar Racing

5.0 ★★★★★ 10 Google reviews

Research institute in Toronto, Ontario

Blue Sky Solar Racing is a student-run team at the

BLUE SKY SOLAR
RACING

Project Roadmap

- Much progress was made in early 2023
- Repositories
  - Drivers
    - ADC78H89
    - MCP4161
    - etc...
  - BattLib
  - Revolution

Repositories: Drivers

- Finished (incl. unit tests, type annotations, documentations)
  - ADC78H89
  - MCP4161
  - SN74HCS137
- In progress
  - INA229 (eta 2-4 weeks)
  - MCP23S17 (eta 2-4 weeks)
  - NHD-C12864A1Z-FSW-FBW-HTT (eta 4-8 weeks)

Repository: BattLib

- Finished
    - Battery EKF Algorithm Implementation
    - Unit tests
    - Integration tests
- In progress
    - First version release (eta 2-4 weeks)
    - Type stub generation (eta 2-4 weeks)
    - Documentation (eta 4-8 weeks)

Repository: Revolution (Part 1)

- Finished (Including unit tests w/ mocked HW, type annotations)
    - Shared context
    - Revolution “Architecture”
    - Motor (formerly MCMB)
    - Miscellaneous (formerly BBMB, DCMB, etc.)
    - Steering wheel (formerly SWB, SPB)
    - Telemetry

BLUE SKY SOLAR
RACING

Repository: Revolution (Part 2)

- In progress
  - Display (formerly DCMB)
  - Array and Battery relay (formerly BBMB)
  - Safe state (deactivated on Gen 11)
  - Cruise control
  - BMS integration (consult Jeff)
  - Integration/real-life tests
    - Many components already READY!
    - BFM not yet ready (consult Rishabh)
      - Projected to be ready by early-mid 2024

BLUE SKY SOLAR
RACING

Long-term Goals

- Explore usages of C++
- Generalization of Python IC drivers
  - There are patterns
- Alternate SOC estimation…
  - Advancements in SOTA SOC estimation methods

Blue Sky is a not just a design team

BLUE SKY SOLAR RACING

Blue Sky is a Revolution

Visit us at:
https://github.com/blueskysolarracing/revolution

UNIVERSITY OF
TORONTO
CHALLENGER CLASS
BLUE SKY SOLAR
RACING
77
VIRIDIAN
araucu

Questions?