BLUE SKY SOLAR RACING

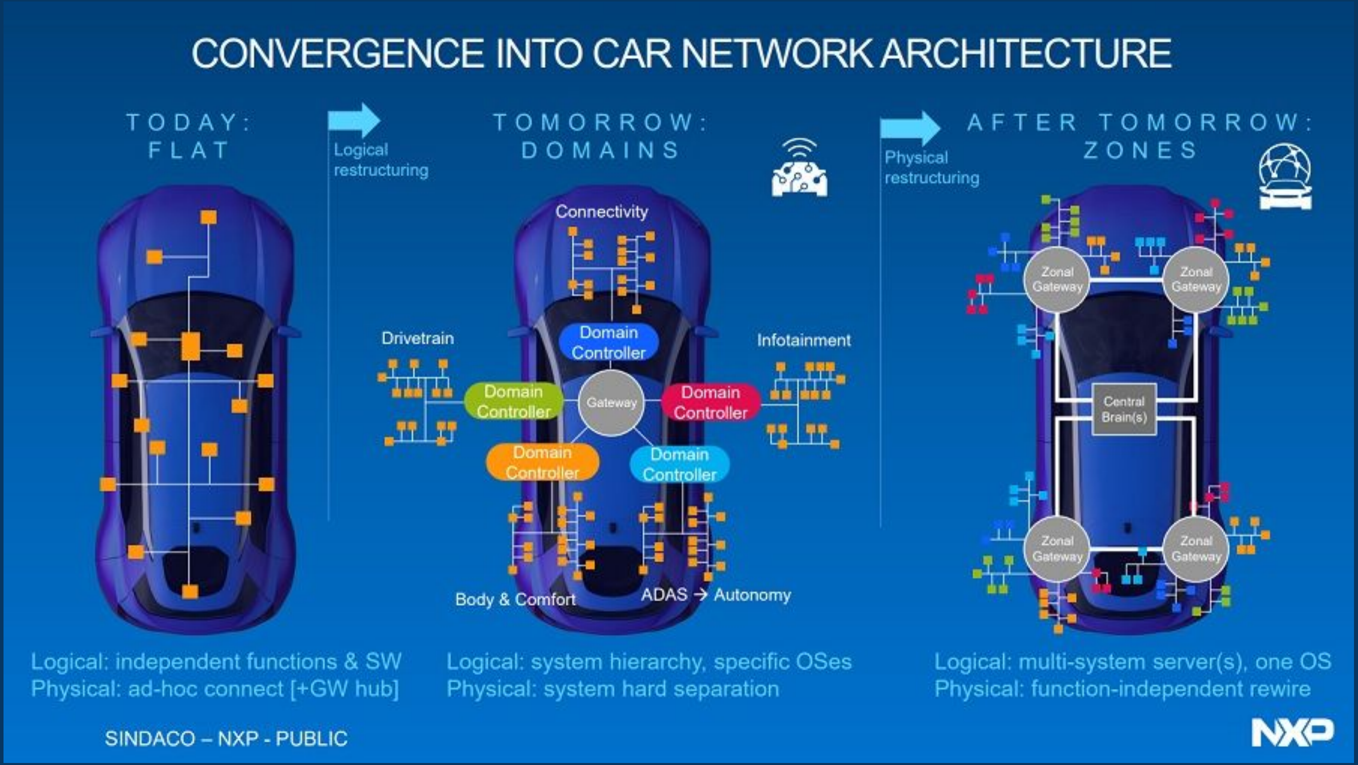
Blue Sky Elec

Towards the Next Generation Vehicle Compute Platform

Contents

BLUE SKY SOLAR
RACING

Decentralized vs Centralized Platform


CONVERGENCE INTO CAR NETWORK ARCHITECTURE
TODAY: FLAT
Logical restructuring
TOMORROW: DOMAINS
Physical restructuring
AFTER TOMORROW: ZONES
Connectivity
Drivetrain
Infotainment
Domain Controller
Gateway
Body & Comfort
ADAS → Autonomy
Zonal Gateway
Central Brain(s)
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

Centralized Platform Examples

ANNOUNCING NVIDIA DRIVE THOR
2000 TFLOPs FP8
77B Transistors
Inference
Transformer Engine
Multi-Domain
Computing Isolation
NVLink-C2C
Scalability

Nvidia: THOR

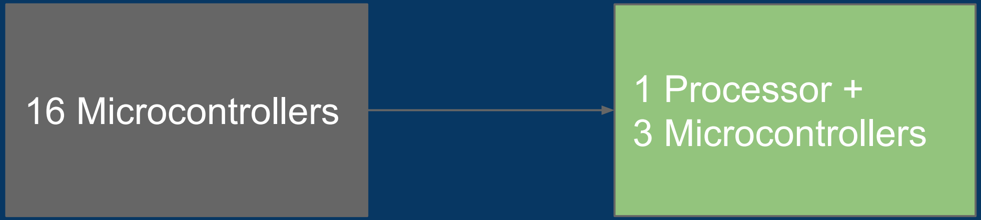
STELLANTIS
STLA Brain
Powered by
Snapdragon
digital chassis
Snapdragon Digital Chassis is a product of Qualcomm Technologies, Inc. and/or its subsidiaries.

Qualcomm: Digital Chassis

BLUE SKY SOLAR
RACING

Why Blue Sky needs a Centralized Platform

- 75 percent reduction in programmable chips



- 70 percent reduction in wiring
- 50+ percent reduction in power consumption
- Infinite increase in software flexibility (remote software update, local strategy simulation)
- Remove state inconsistency introduced by a distributed system

BLUE SKY SOLAR
RACING

Challenges of a Centralized Platform

Lack of Modularity

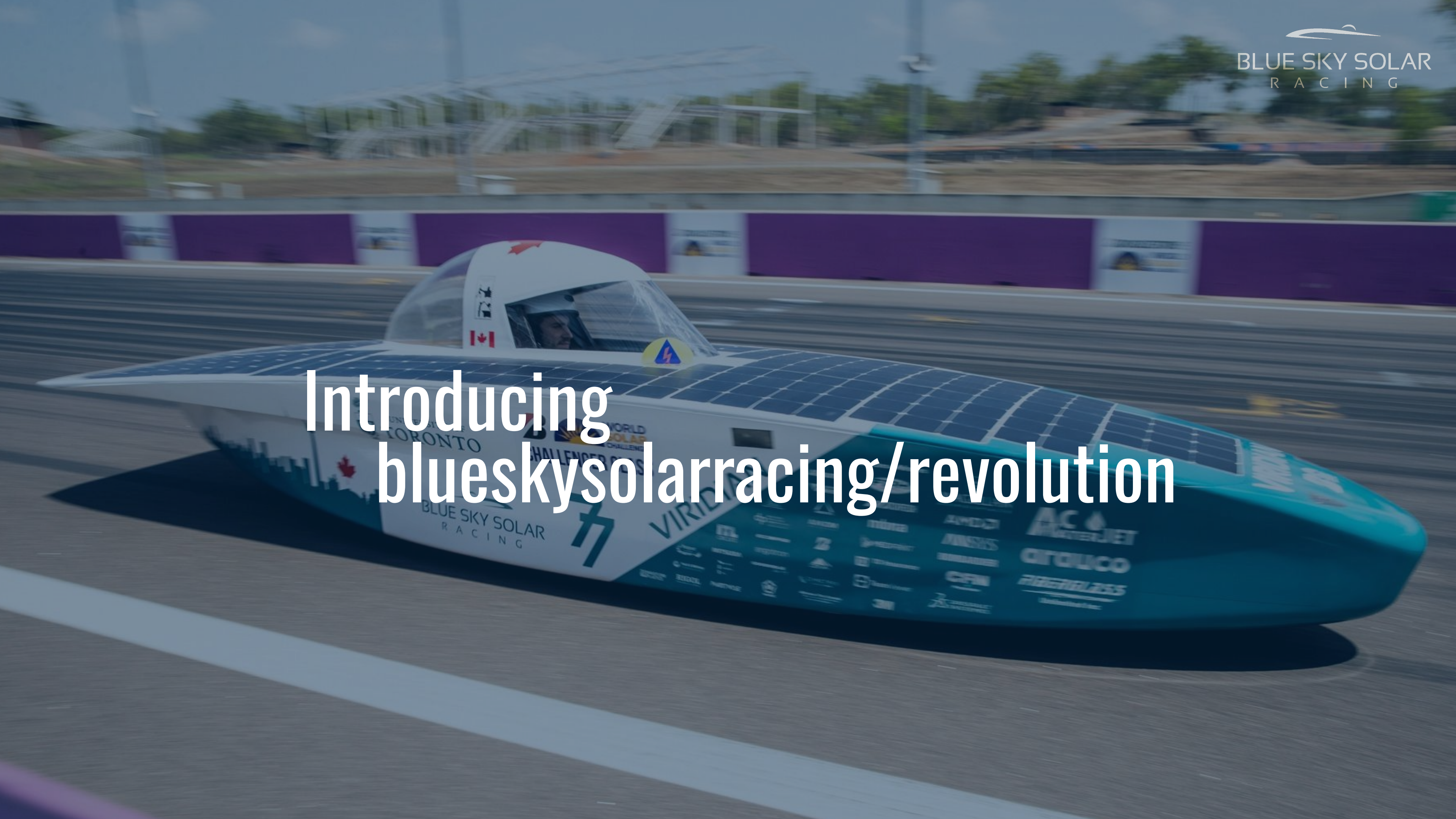
Single Point of Failure

Program Complexity

BLUE SKY SOLAR RACING

Should Blue Sky make this leap?

Today, we have the momentum



BLUE SKY SOLAR RACING

Introducing blueskysolarracing/revolution

A New Linux Computing Platform

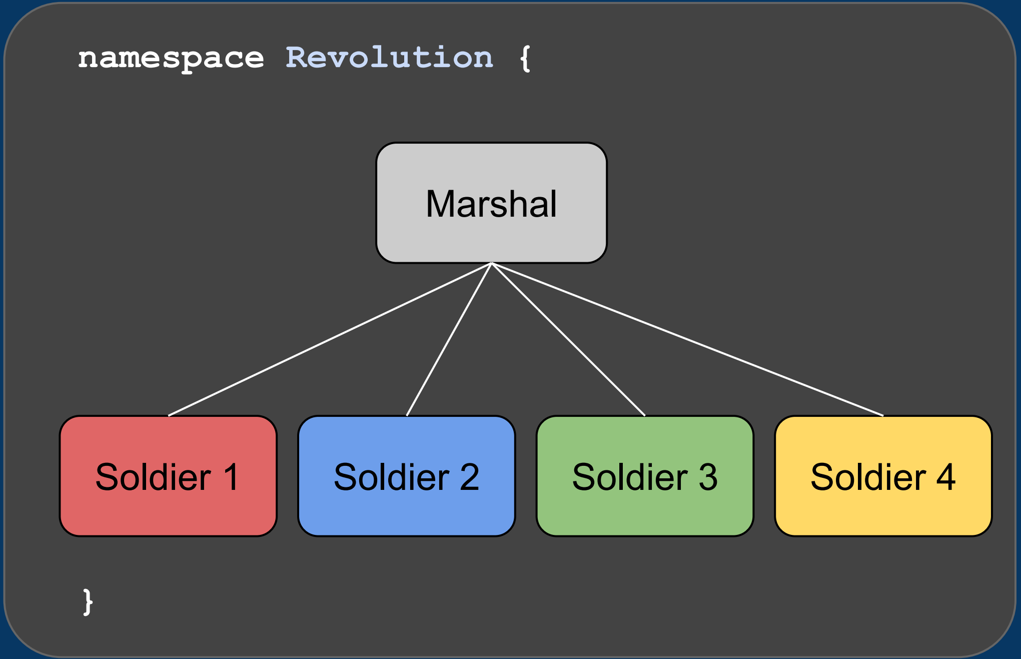
- Remote access
- Powerful
- Well-documented
- Great support

From C to modern C++

- Fast
- C++
  - Low & high level
  - Abstraction/separation
  - STL
  - C compatibility
  - More features

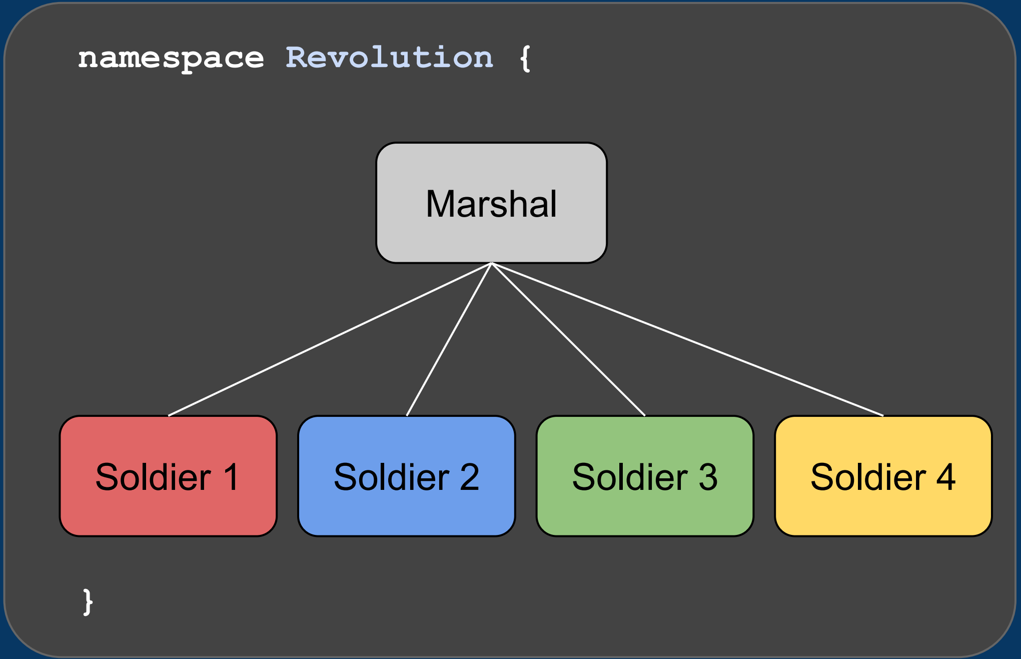
Marshal-Soldier Architecture

- Applications
  - Marshal
    - Syncer
  - Soldiers
    - Display Driver
    - Power Sensor
    - Motor Controller
    - Telemeter
    - And so on…



Car State Synchronization

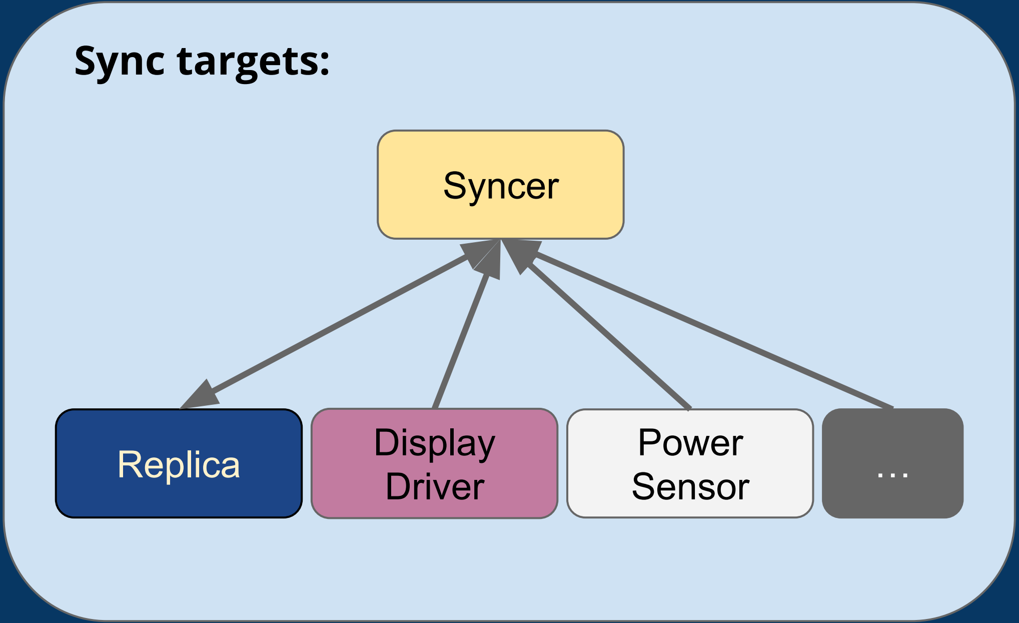
- Everything on the "same page"
- Writes:
  - Soldiers request marshal
  - Marshal propagates
- Reads:
  - Internal cache
    - Efficiency
    - Marshal may crash



BLUE SKY SOLAR
RACING

Crash Recovery

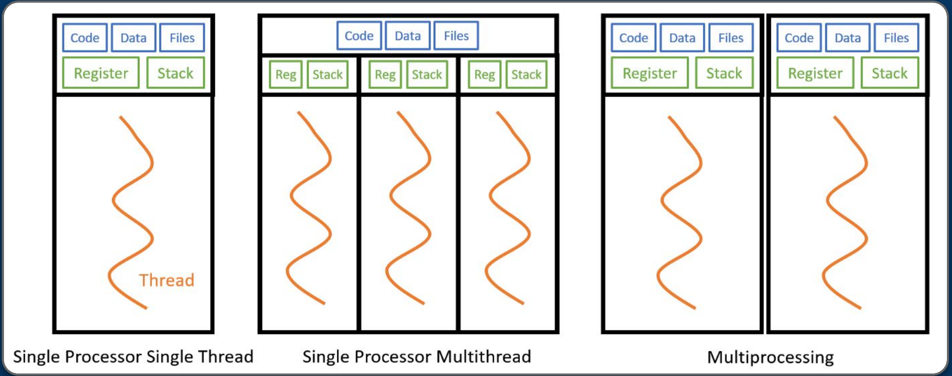
- Recover quickly
- Systemd monitors crash
- Data recovery via sync



BLUE SKY SOLAR
RACING

Why Multiprocess?

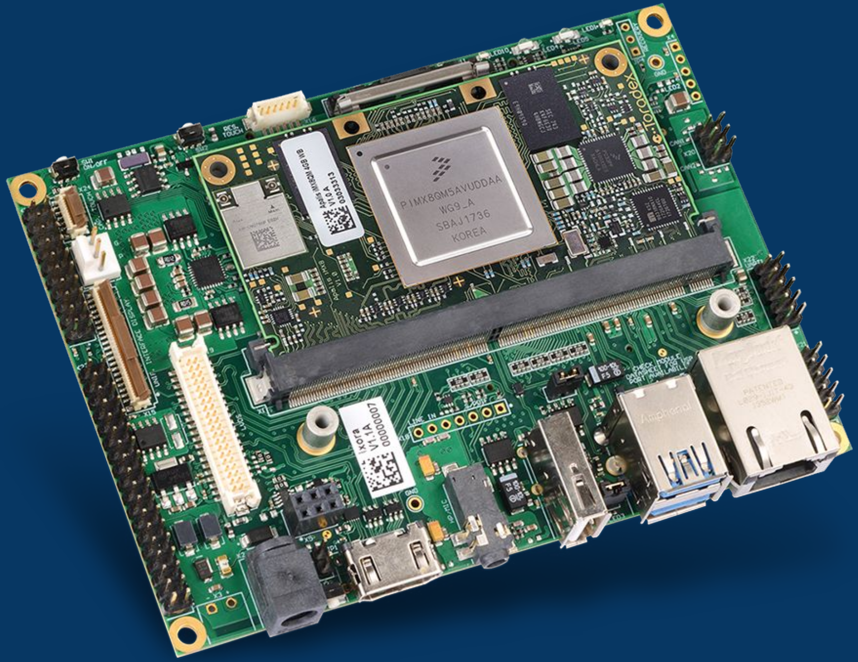
- Multithreading
  - (More) efficient
  - On crash:
    - Complete loss of states
    - Complete loss of control



- Multiprocessing
  - Better logs
  - On crash:
    - Can replicate states from other processes
    - Partial loss of control

Virtual H/W Interface

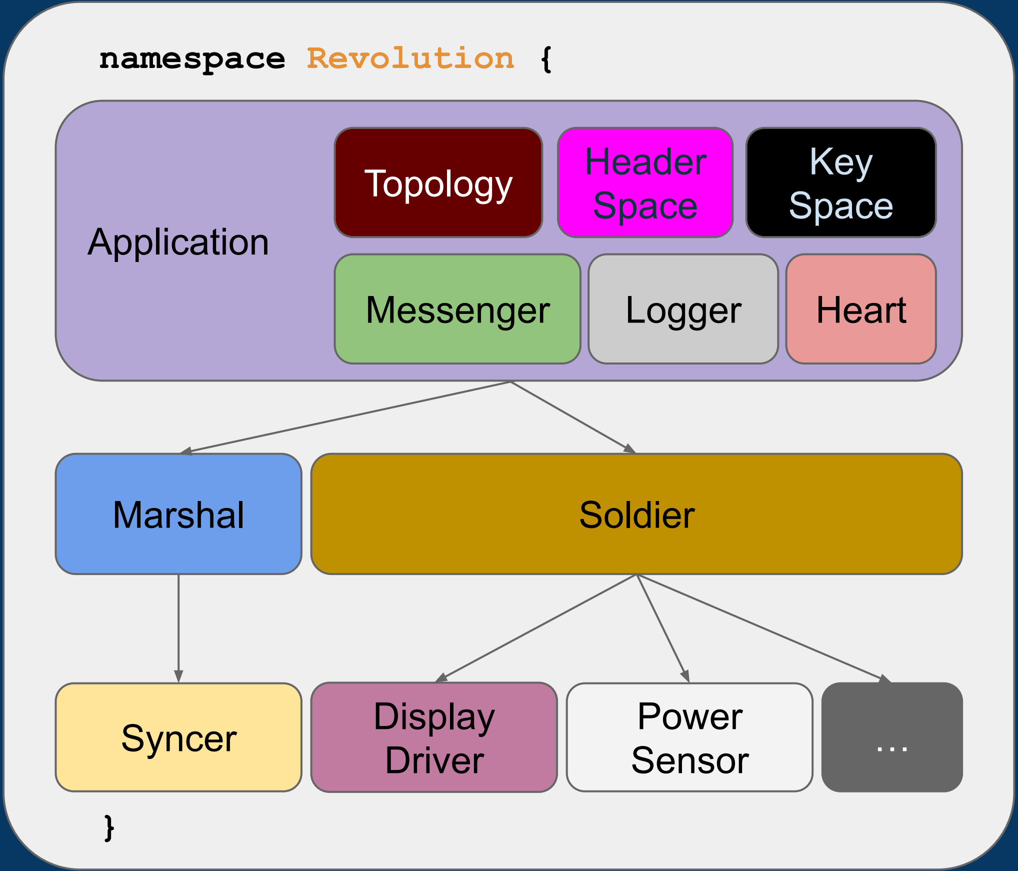
- Simulate H/W access
- Unit tests



BLUE SKY SOLAR RACING

Class Inheritance Map

- Utilities
  - Messenger
  - Logger
  - Heart
  - Client
- Configuration
  - Topology
  - Header space
  - Key space
- Applications
  - Marshal
    - Syncer
  - Soldier
    - Display Driver
    - Power Sensor
    - etc…

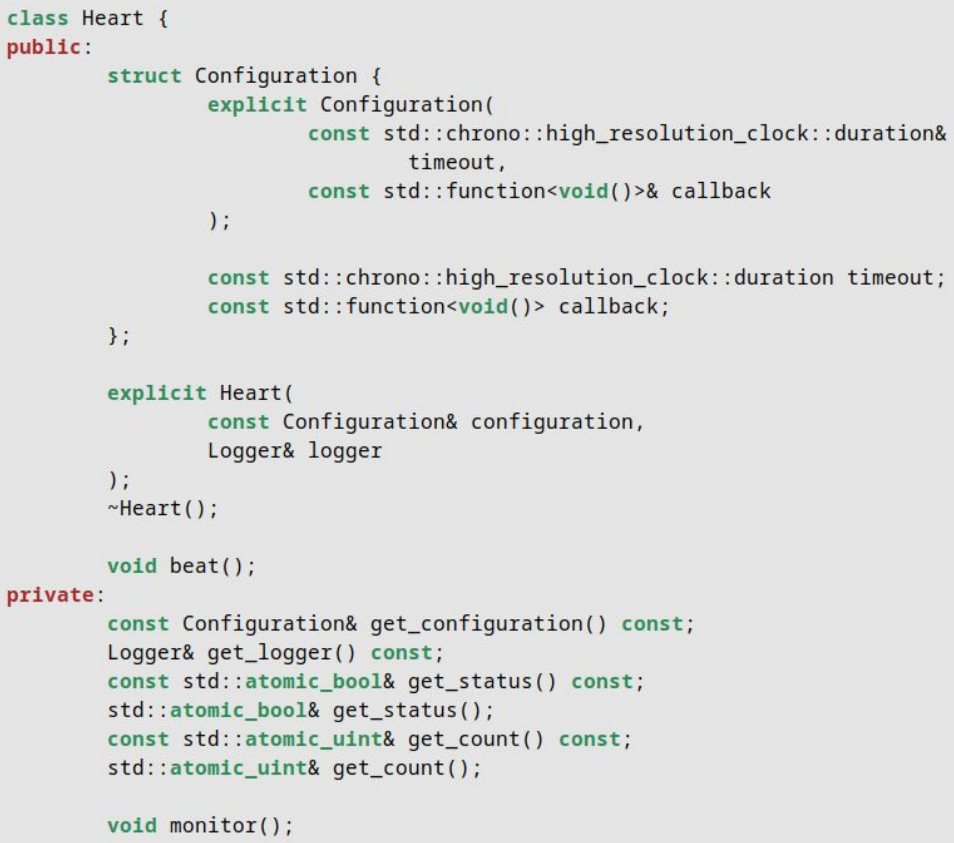


Utilities – Messenger and Logger

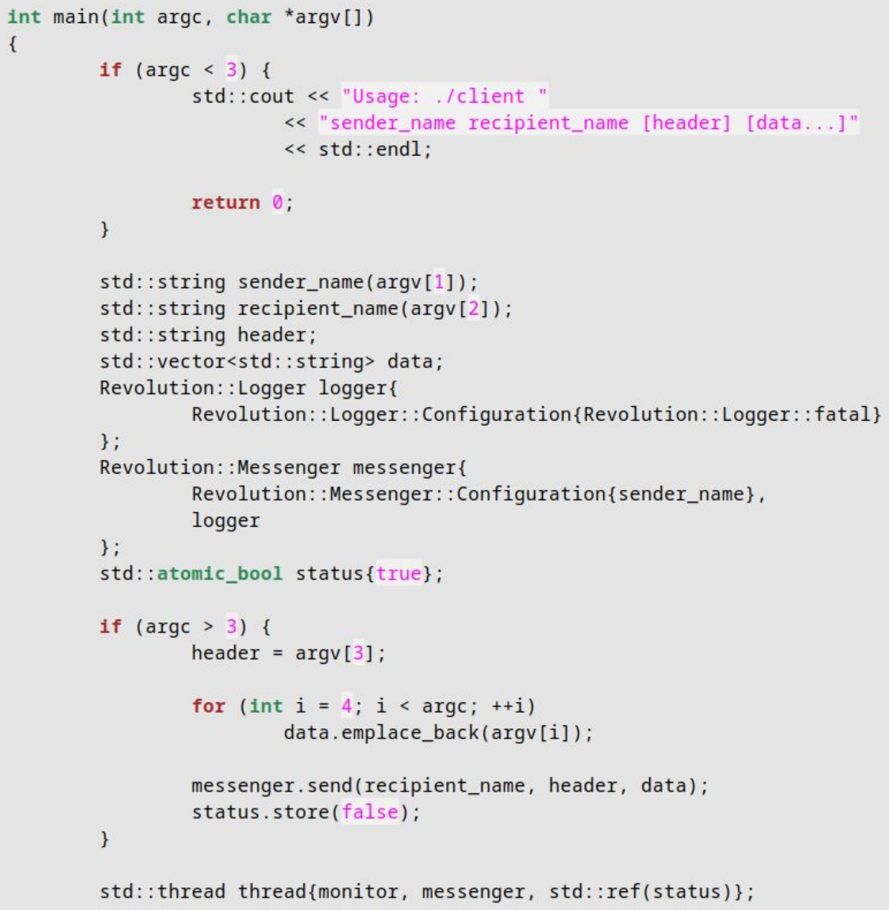
```cpp
class Messenger {
public:
        struct Message {
                static Message deserialize(
                        const std::string& raw_message
                );

                explicit Message(
                        const std::string& sender_name,
                        const std::string& header,
                        const std::vector<std::string>& data = {}
                );

                std::string serialize() const;
                std::string to_string() const;

                const std::string sender_name;
                const std::string header;
                const std::vector<std::string> data;
        };

        struct Configuration {
                explicit Configuration(
                        const std::string& name,
                        const std::string& full_name_prefix = "/",
                        const unsigned int& priority = 0,
                        const int& oflags = O_RDWR | O_CREAT,
                        const mode_t& mode = 0644
                );

                const std::string name;
                const std::string full_name_prefix;
                const unsigned int priority;
                const int oflags;
                const mode_t mode;
```

```cpp
class Logger : public std::ostream {
public:
        struct Severity {
                explicit Severity(
                        const std::string& name,
                        const unsigned int& level
                );

                const std::string name;
                const unsigned int level;
        };

        struct Configuration {
                explicit Configuration(
                        const Severity& severity,
                        const std::string& filename = "",
                        const std::ofstream::openmode& open_mode
                                = std::ofstream::app
                );

                const Severity severity;
                const std::string filename;
                const std::ofstream::openmode open_mode;
        };

        static const Severity trace;
        static const Severity debug;
        static const Severity info;
        static const Severity warning;
        static const Severity error;
        static const Severity fatal;

        explicit Logger(
                const Configuration& configuration
        );
```

Utilities – Heart and Client



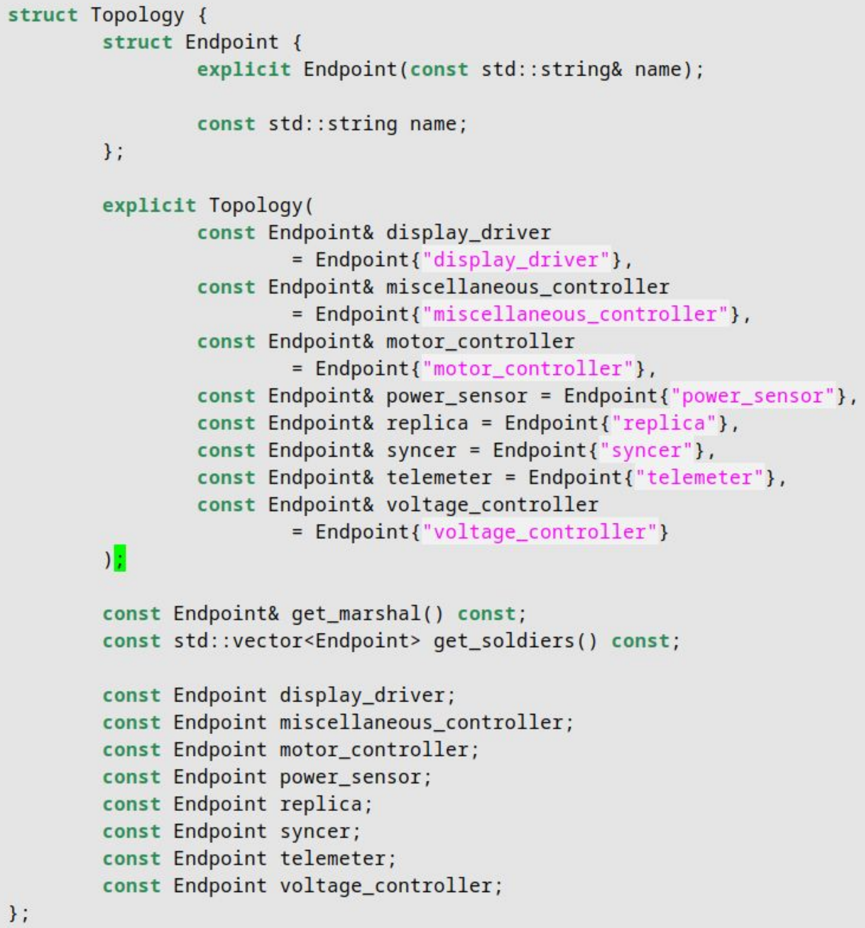
```cpp
class Heart {
public:
        struct Configuration {
                explicit Configuration(
                        const std::chrono::high_resolution_clock::duration&
                                timeout,
                        const std::function<void()>& callback
                );

                const std::chrono::high_resolution_clock::duration timeout;
                const std::function<void()> callback;
        };

        explicit Heart(
                const Configuration& configuration,
                Logger& logger
        );
        ~Heart();

        void beat();
private:
        const Configuration& get_configuration() const;
        Logger& get_logger() const;
        const std::atomic_bool& get_status() const;
        std::atomic_bool& get_status();
        const std::atomic_uint& get_count() const;
        std::atomic_uint& get_count();

        void monitor();
```



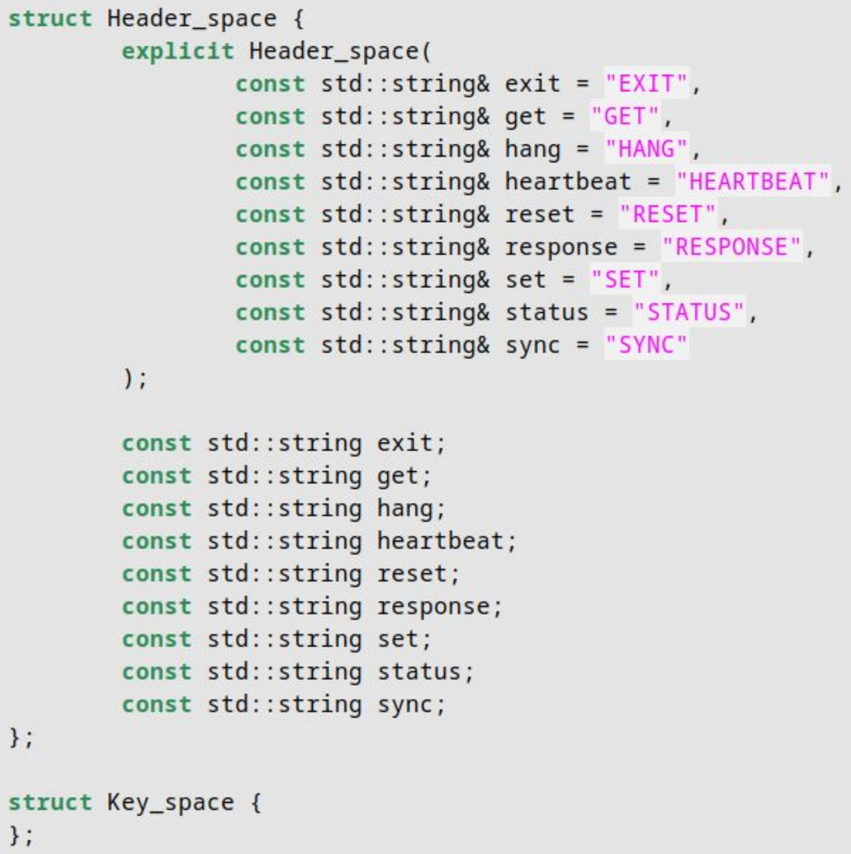
```cpp
int main(int argc, char *argv[])
{
        if (argc < 3) {
                std::cout << "Usage: ./client "
                        << "sender_name recipient_name [header] [data...]"
                        << std::endl;

                return 0;
        }

        std::string sender_name(argv[1]);
        std::string recipient_name(argv[2]);
        std::string header;
        std::vector<std::string> data;
        Revolution::Logger logger{
                Revolution::Logger::Configuration{Revolution::Logger::fatal}
        };
        Revolution::Messenger messenger{
                Revolution::Messenger::Configuration{sender_name},
                logger
        };
        std::atomic_bool status{true};

        if (argc > 3) {
                header = argv[3];

                for (int i = 4; i < argc; ++i)
                        data.emplace_back(argv[i]);

                messenger.send(recipient_name, header, data);
                status.store(false);
        }

        std::thread thread{monitor, messenger, std::ref(status)};
```

Configuration – Topology, Header/Key Space

BLUE SKY SOLAR
RACING



```
struct Topology {
        struct Endpoint {
                explicit Endpoint(const std::string& name);

                const std::string name;
        };

        explicit Topology(
                const Endpoint& display_driver
                        = Endpoint{"display_driver"},
                const Endpoint& miscellaneous_controller
                        = Endpoint{"miscellaneous_controller"},
                const Endpoint& motor_controller
                        = Endpoint{"motor_controller"},
                const Endpoint& power_sensor = Endpoint{"power_sensor"},
                const Endpoint& replica = Endpoint{"replica"},
                const Endpoint& syncer = Endpoint{"syncer"},
                const Endpoint& telemeter = Endpoint{"telemeter"},
                const Endpoint& voltage_controller
                        = Endpoint{"voltage_controller"}
        );

        const Endpoint& get_marshal() const;
        const std::vector<Endpoint> get_soldiers() const;

        const Endpoint display_driver;
        const Endpoint miscellaneous_controller;
        const Endpoint motor_controller;
        const Endpoint power_sensor;
        const Endpoint replica;
        const Endpoint syncer;
        const Endpoint telemeter;
        const Endpoint voltage_controller;
};
```



```
struct Header_space {
        explicit Header_space(
                const std::string& exit = "EXIT",
                const std::string& get = "GET",
                const std::string& hang = "HANG",
                const std::string& heartbeat = "HEARTBEAT",
                const std::string& reset = "RESET",
                const std::string& response = "RESPONSE",
                const std::string& set = "SET",
                const std::string& status = "STATUS",
                const std::string& sync = "SYNC"
        );

        const std::string exit;
        const std::string get;
        const std::string hang;
        const std::string heartbeat;
        const std::string reset;
        const std::string response;
        const std::string set;
        const std::string status;
        const std::string sync;
};

struct Key_space {
};
```

Applications – Marshal and Soldier

BLUE SKY SOLAR
RACING

```
class Application {
public:
    using Handler = std::function<void(const Messenger::Message&)>;
    using Handlers = std::unordered_map<std::string, Handler>;
    using States = std::unordered_map<std::string, std::string>;

    explicit Application(
        const Topology& topology,
        const Header_space& header_space,
        const Key_space& key_space,
        Logger& logger,
        const Messenger& messenger,
        Heart& heart
    );

    virtual void run();
protected:
    const Topology& get_topology() const;
    const Header_space& get_header_space() const;
    const Key_space& get_key_space() const;
    Logger& get_logger() const;
    const Messenger& get_messenger() const;
    Heart& get_heart() const;
    const bool& get_status() const;
    void set_status(const bool& status);
    const States& get_states() const;
    std::vector<std::string> get_state_data() const;
    void set_handler(
        const std::string& name,
        const Handler& handler
    );
    virtual const Topology::Endpoint& get_endpoint() const = 0;

    void handle_exit(const Messenger::Message& message);
    void handle_hang(const Messenger::Message& message) const;
```

```
class Marshal : public Application {
public:
    explicit Marshal(
        const Topology& topology,
        const Header_space& header_space,
        const Key_space& key_space,
        Logger& logger,
        const Messenger& messenger,
        Heart& heart
    );

    void run() override;
protected:
    void handle_write(const Messenger::Message& message) override;
};
```

```
class Soldier : public Application {
public:
    explicit Soldier(
        const Topology& topology,
        const Header_space& header_space,
        const Key_space& key_space,
        Logger& logger,
        const Messenger& messenger,
        Heart& heart
    );

    void run() override;
protected:
    void handle_write(const Messenger::Message& message) override;
};
```

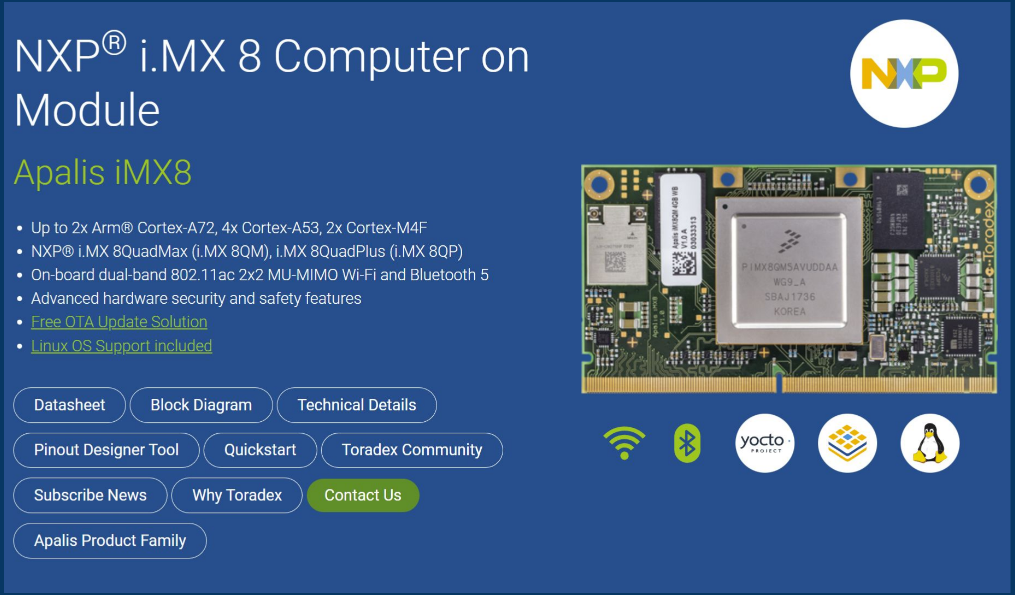
BLUE SKY SOLAR
RACING

Demo

- Setup and startup
- State propagation
- Crash recovery

Going forward as a team

- Obtain board
- Implement HW interface
  - GPIO, SPI, I2C
  - Complete by end of year
- Implement soldiers
  - Display
  - Motor
  - Battery
  - Telemetry
  - Driver
- Complete software by April



NXP® i.MX 8 Computer on Module

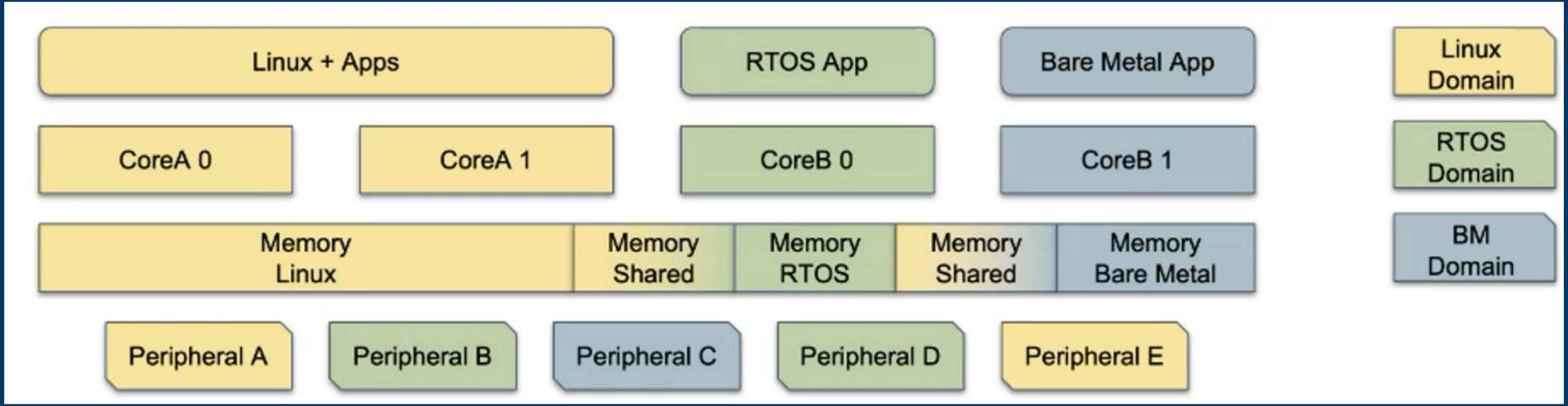
Apalis iMX8

- Up to 2x Arm® Cortex-A72, 4x Cortex-A53, 2x Cortex-M4F
- NXP® i.MX 8QuadMax (i.MX 8QM), i.MX 8QuadPlus (i.MX 8QP)
- On-board dual-band 802.11ac 2x2 MU-MIMO Wi-Fi and Bluetooth 5
- Advanced hardware security and safety features
- Free OTA Update Solution
- Linux OS Support included

BLUE SKY SOLAR
RACING

Our End Goal

- Successful Implementation of Heterogeneous Compute
  - Fulfill Real-Time requirements, while having powerful processing capabilities



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