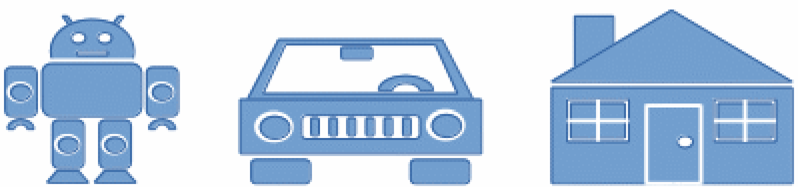
Quest-V: A Secure and Predictable System for IoT and Beyond

Richard West
richwest@cs.bu.edu

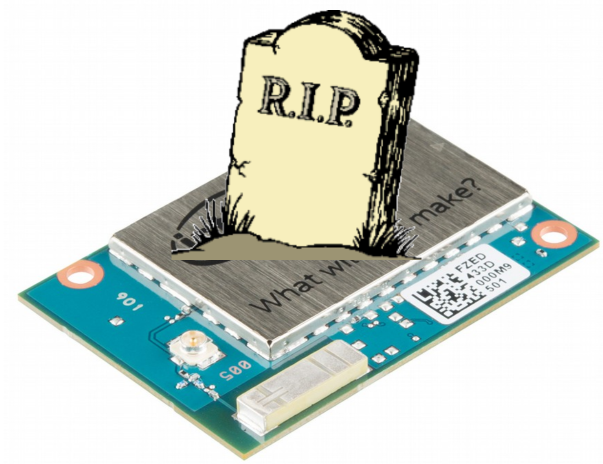


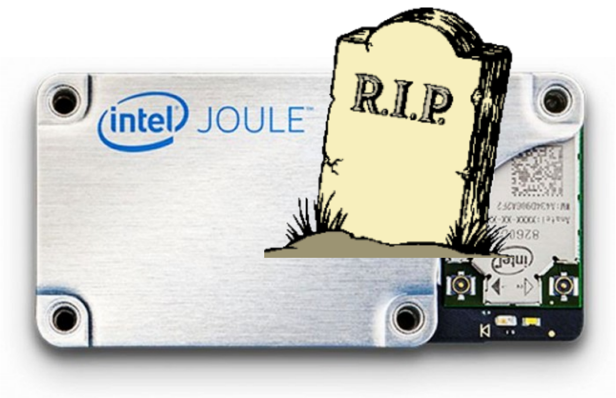
Computer Science

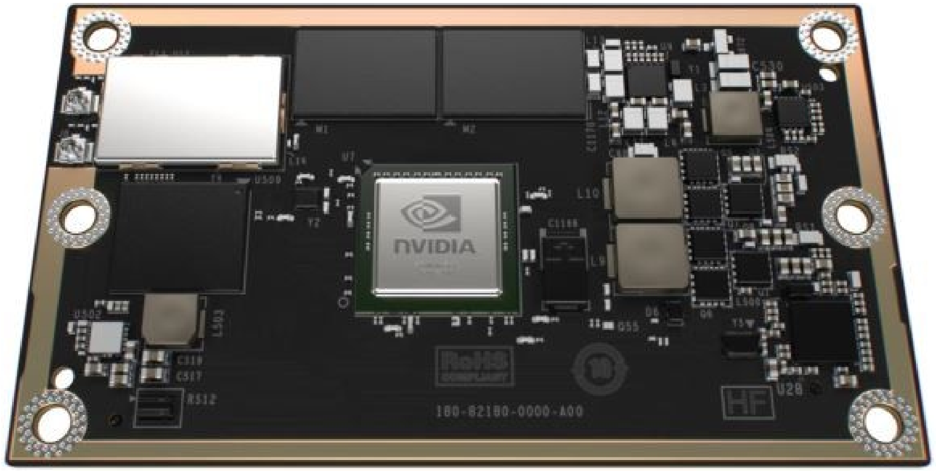
Talk Outline

- Background on embedded single board computers (SBCs)
- Quest(-V) OS for x86 SBCs
- Work status
  - Lessons learned
  - Wish list
  - Impact, papers, etc
- Case study: Quest(-V) for web-connected 3D printers
- Current & future work
- Final words

Emerging Multicore SBCs



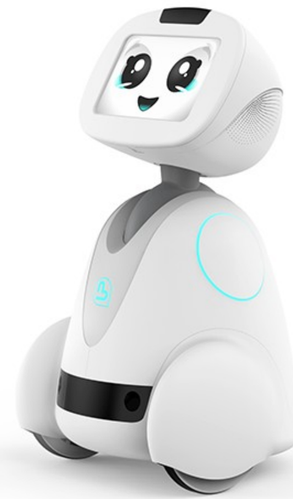


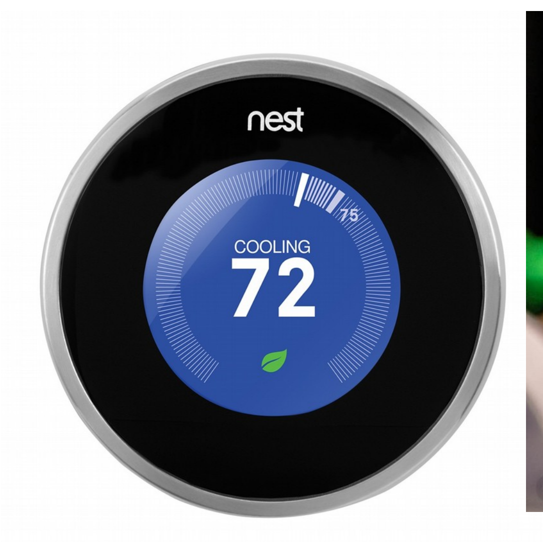


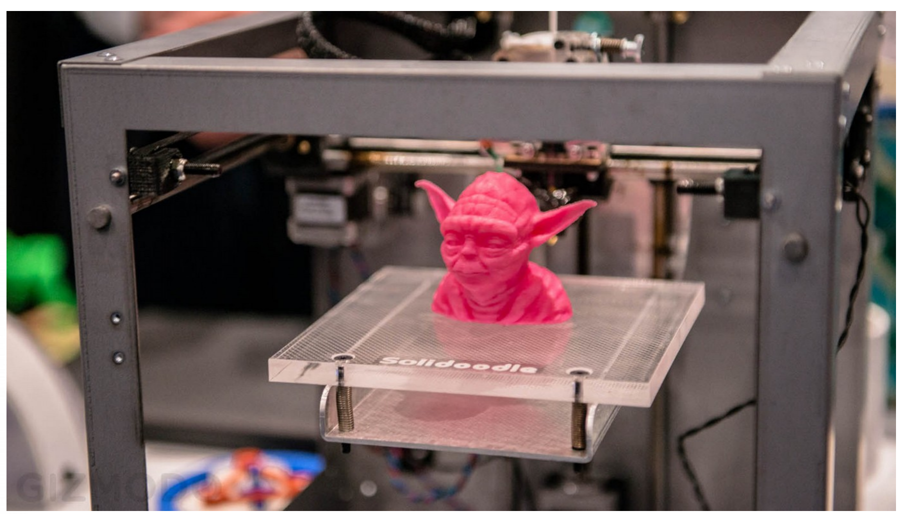
Intel vs ARM SBCs

- ARM: Raspberry Pi, Nvidia Jetson TX1/TX2 & many others
  - Pi 3 Model B: 4 Cortex-A53 cores @ 1.2GHz, 1GB RAM, Broadcom GPU
  - Nvidia Tegra Xavier (automotive AI): 8 ARM64 CPU & 512 CUDA GPU cores
- x86: Joule, Aero, Up Squared, etc
  - Up Squared: 4 CPU cores (Apollo Lake Atom/Celeron/Pentium), Gen 9 iGPU
  - Intel Go (automotive): Xeon/Atom CPUs, Arria 10 FPGA hardware accelerators
- x86 largely standardized according to PC specs
  - BIOS/UEFI, ACPI, PCIe
  - Makes OS development less fragmented for different targets
- Less standardization amongst ARM SoC vendors
  - Bootloader (e.g., U-boot) loads device trees for board-specific configurations
  - ACPI not common in ARM embedded systems

Potential for Smart Devices







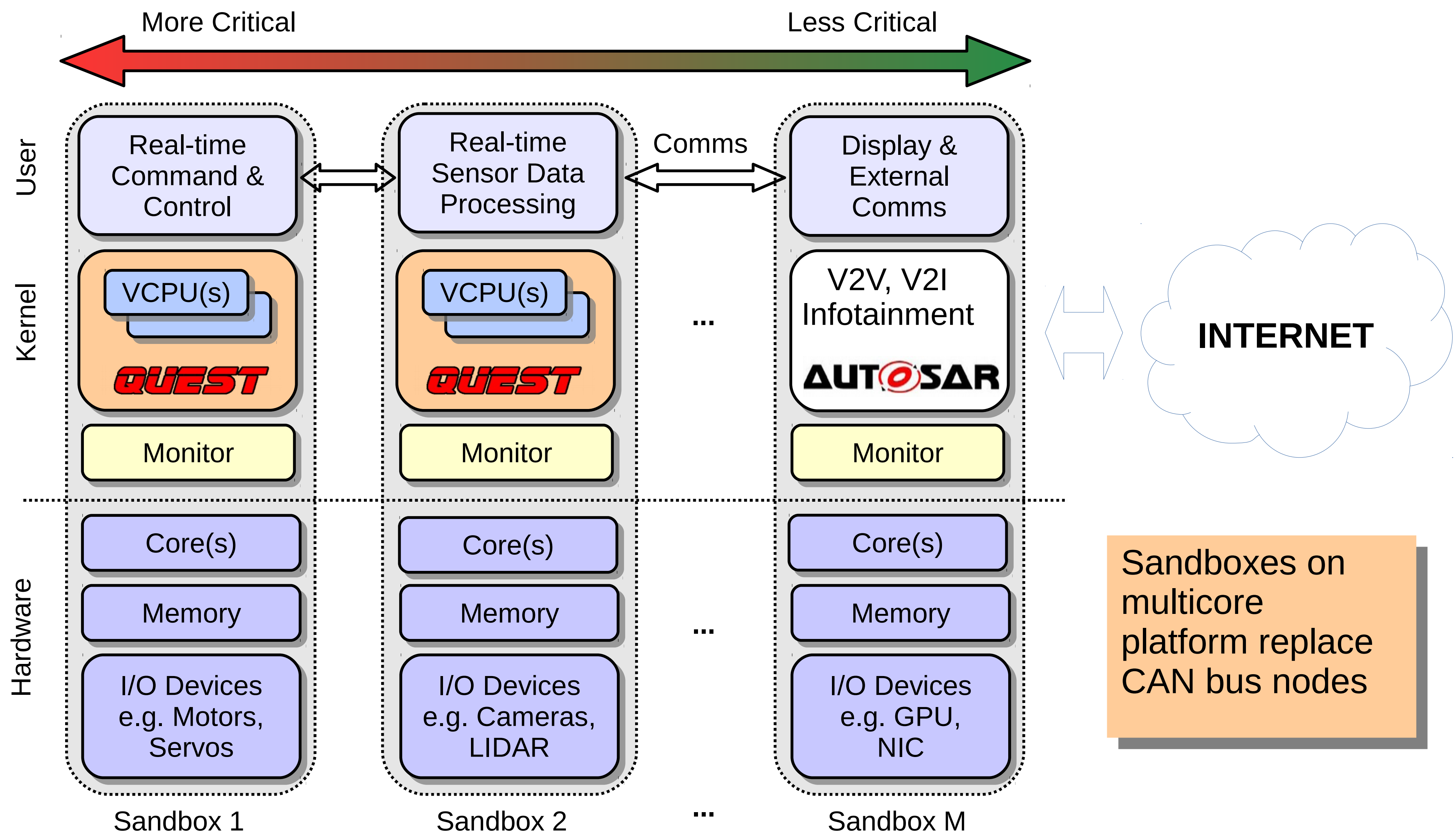
Need New Systems

- (Embedded) OSes that are:

  - Timing Predictable
  - Safe
  - Fault Tolerant
  - Secure
  - Multicore
  - Mixed-criticality-aware

  - Enter **Quest-V**

Example Quest-V Automotive System



Sandboxes on multicore platform replace CAN bus nodes

Work Status

- **AIM: Insights from Implementing Quest(-V) on Intel SBCs**
- [Done] Quest running on Galileo, Edison, MinnowMax, Joule & Aero
- [Near Completion] Quest-V running on Joule & Aero
  - [In progress] Quest-V Linux (works on Aero)
  - [In progress] Drivers for BMM150 (Compass) + BMI160 (IMU) + GPIOs
- [Version 1 complete] Qduino API
  - Includes support for multiple cores – **QduinoMC**
  - Tested on 3D printer & now working on UAVs
- [In progress] Work with automotive partner (Drako Motors)

Lessons Learned

- Intel SBCs for “smart” devices
  - Multiple cores (good for multi-tasking) ✓
  - VT-x capabilities for security/isolation/fault tolerance ✓
  - GPIOs for interfacing sensors + actuators ✓
  - PWMs for motor & servo control ✓
  - Serial interfaces for device communication ✓
  - Shared caches + memory bus affects temporal isolation (not good for real-time!) ✗
    - ARINC 653 requires space-time isolation b/w cores

Wish List 1/2

- Temporal isolation b/w cores
  - Support for cache + bus isolation (way partitioning, page coloring, TDMA bus management?)
  - Cache-allocation technology (CAT/CMT) available on Xeons but not (yet?) Intel SBCs
- Integrated GPU support
  - Joule, Aero, Skull Canyon are a good start
  - Needed for vision+AI+deep learning tasks
  - Edge devices where remote (cloud) processing is impractical

Wish List 2/2

- Simplified VT-x support
  - Basic memory partitioning b/w sandboxes
- Tagged memory regions for confidentiality + integrity on secure information flows between sandboxes
- H/W-assisted port-based I/O interposition
  - To prevent sandbox discovery/access to unauthorized devices

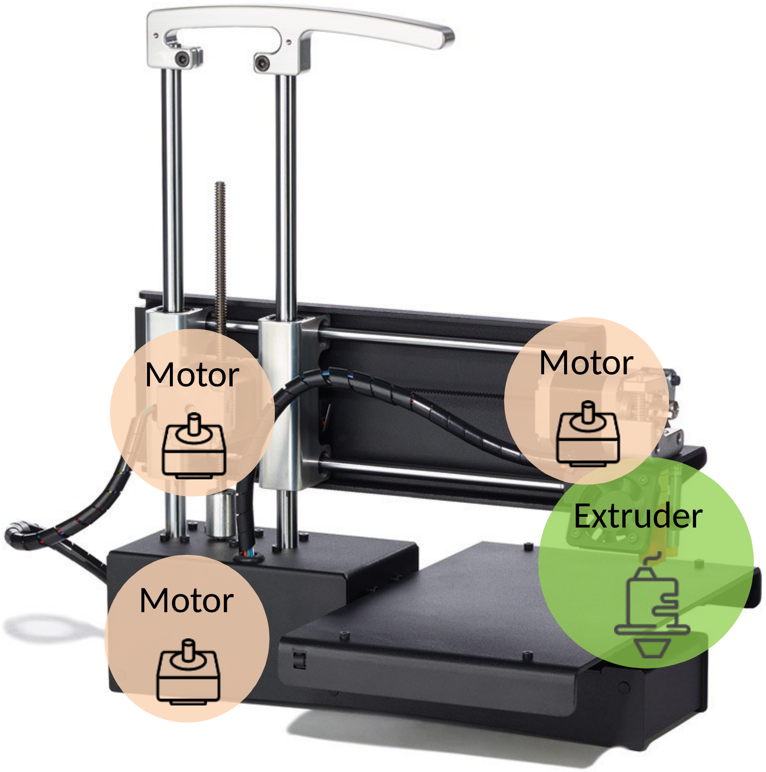
Impact

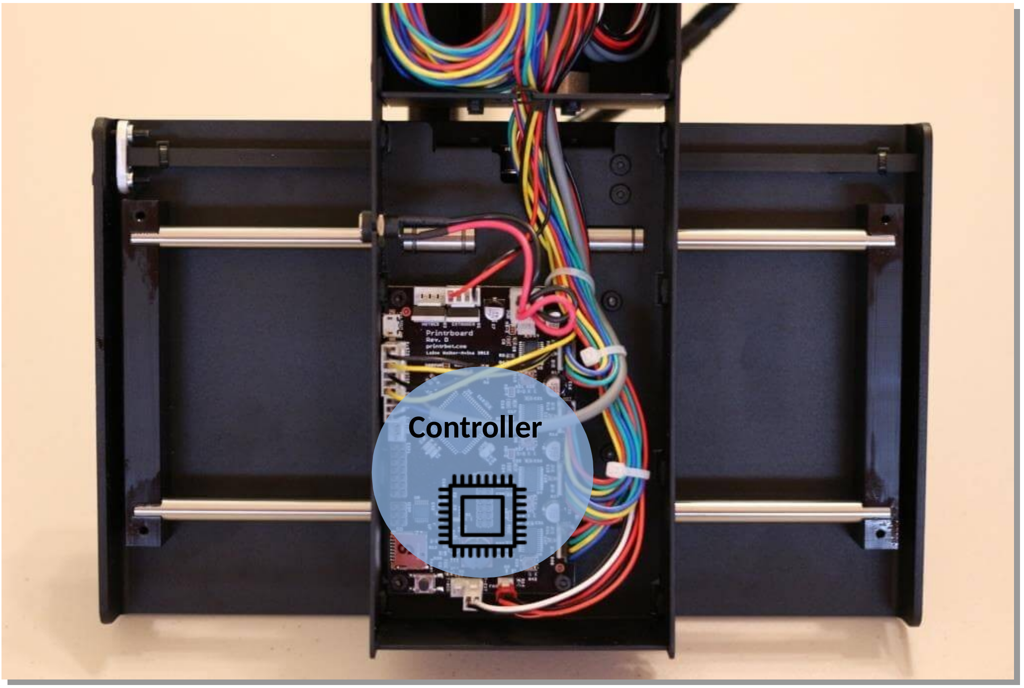
- IEEE RTAS 2017 (QduinoMC)
  - Outstanding paper, best student paper
- ACM TOCS Journal 2016 (Quest-V)
- IEEE RTSS 2016 (MARACAS)
- IEEE RTSS 2015 (Qduino)
- ECRTS 2016 (Quest Mixed-Criticality Scheduling)
- Quest-V is being adopted by Drako Motors as part of DriveOS
- Mercury Systems shortlisted Quest-V as the only academic system to meet their first phase requirements for a separation kernel
- Quest-V well known in real-time research community

Case Study: Quest(-V) Web-Connected 3D Printer

- Remote Job Submission
- Local Slicing
- Correctness Verification

Printrbot Simple Metal



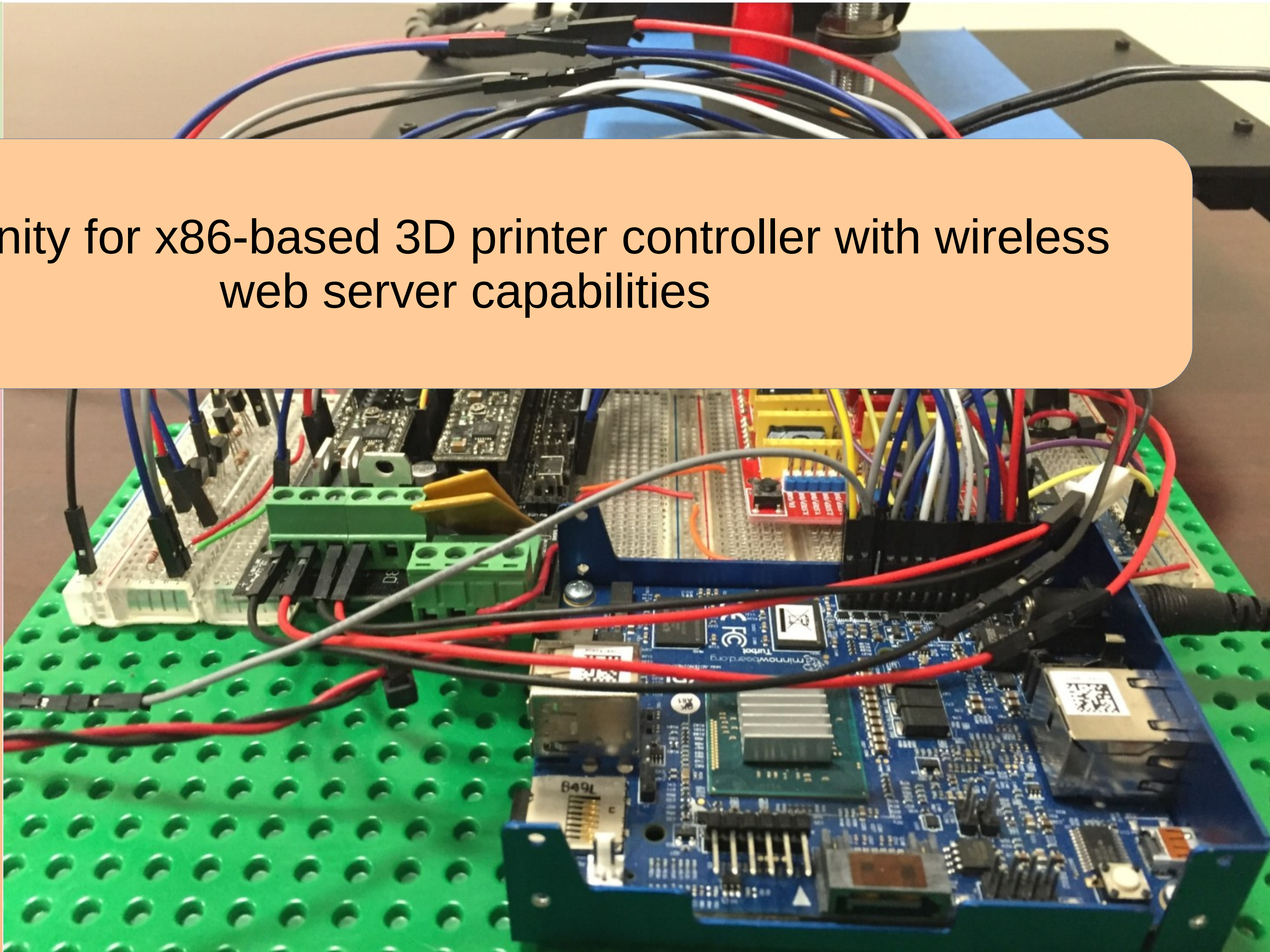


Web Server

Microprocessor	Atmel AVR, 8 bit, 20 MHz
SRAM	8 KB
I/O	UART, SPI, I2C, PWM, GPIO

Custom Controller

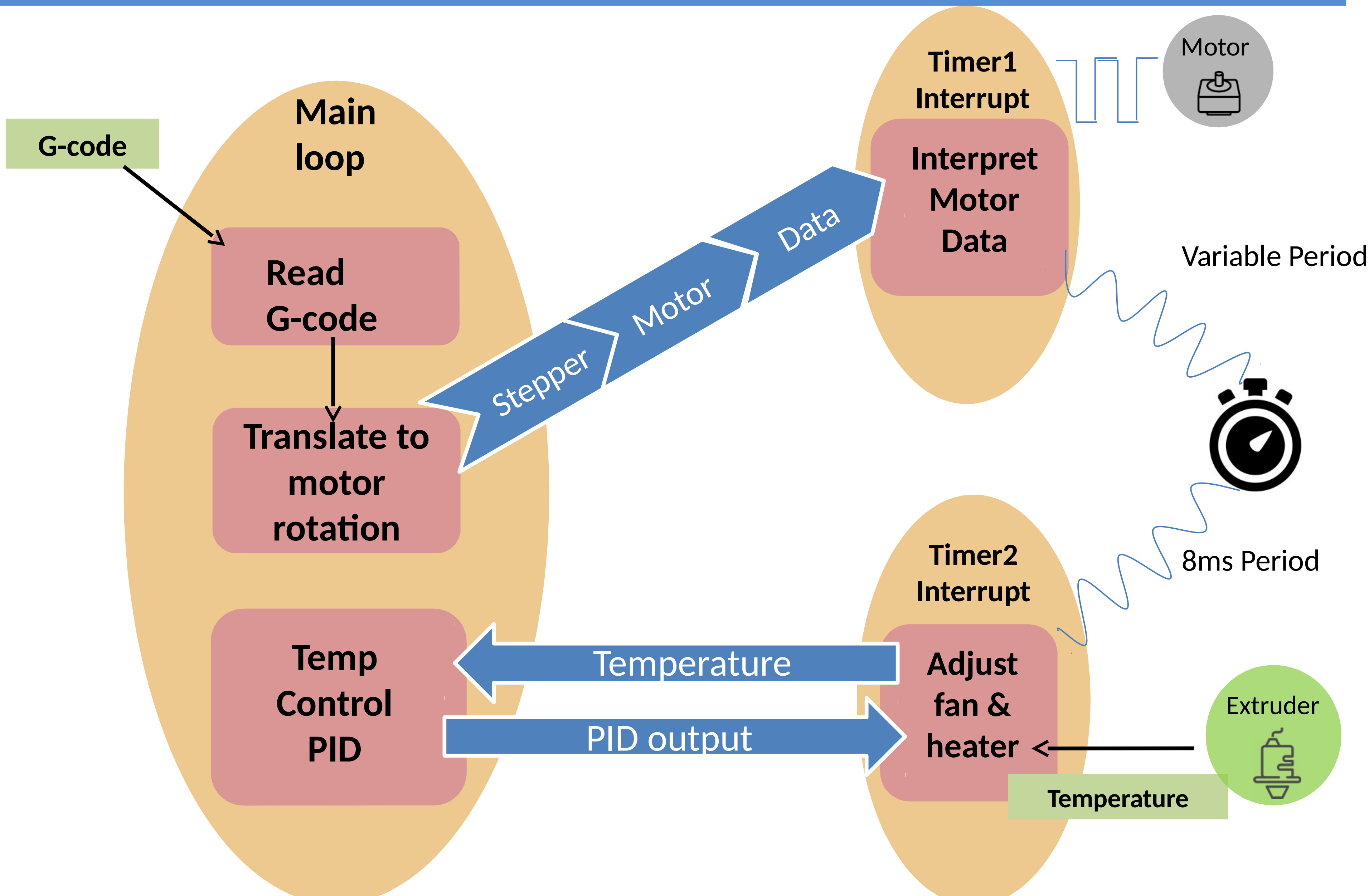
Opportunity for x86-based 3D printer controller with wireless web server capabilities



Companion Analog Circuits

MinnowMAX

Marlin Firmware

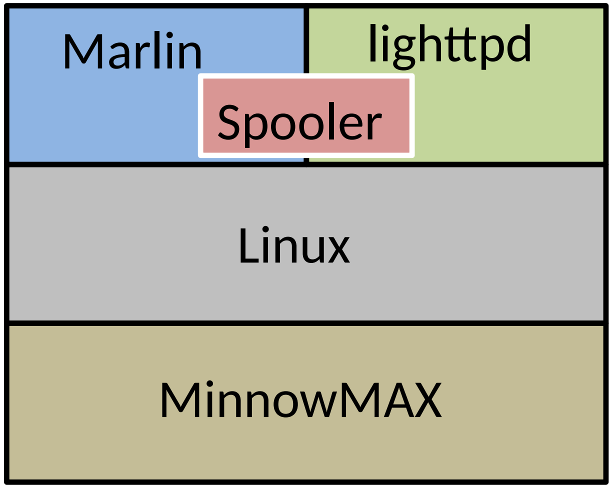


Marlin on Linux

Original Marlin	Linux Port
Main loop + interrupts handlers	Multiple threads
Timer interrupts	Periodic nanosleep
AVR I/O instructions	Intel MRAA IoT library
	lighttpd + spooler

Jitter of the extruder, when submitting relatively large files

Is this bad? Why?

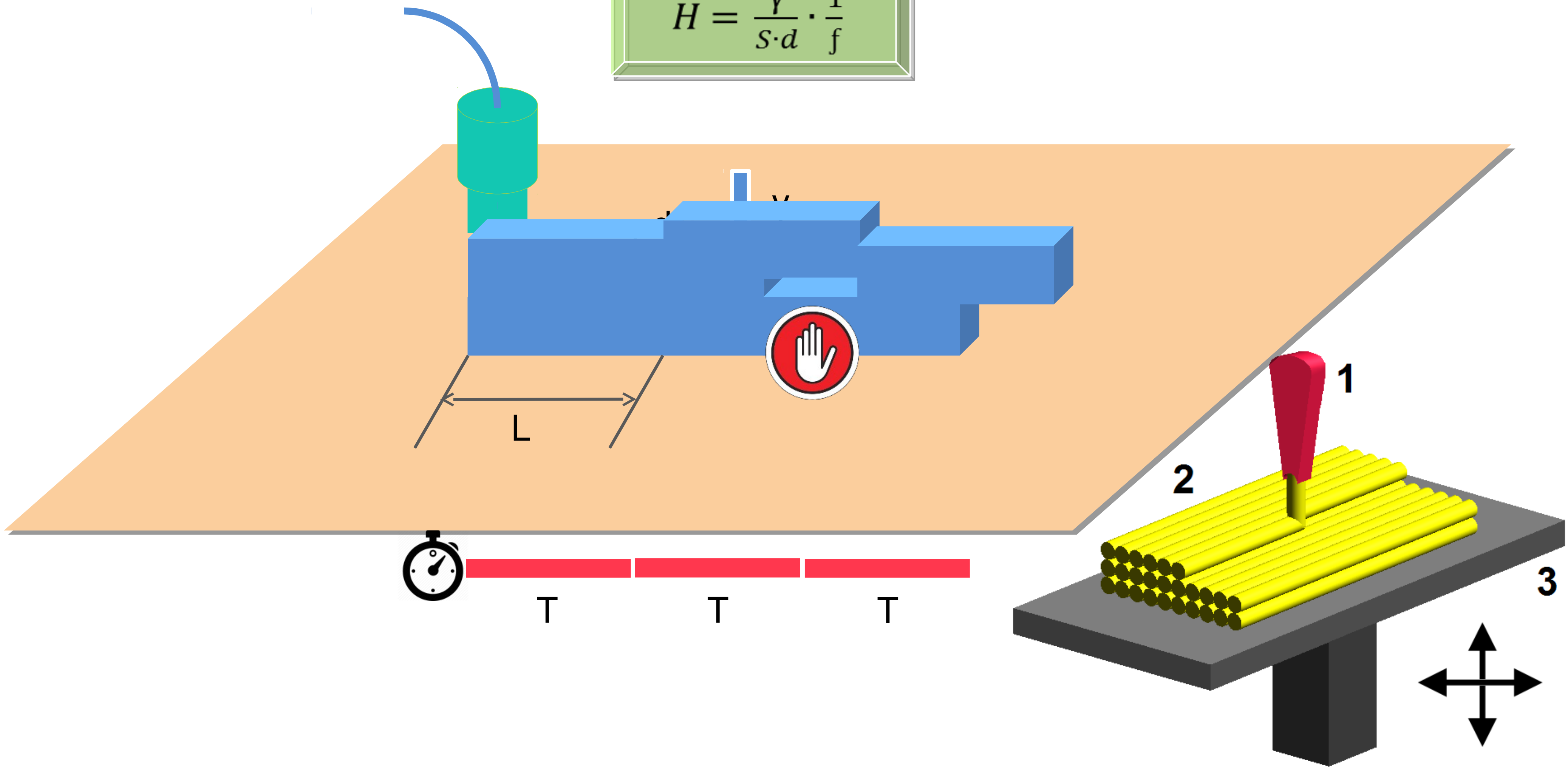


The Timing problem

$$Volume = \gamma \cdot T = L \cdot H \cdot d$$
$$= f \cdot T \cdot S \cdot H \cdot d$$

f -- pulse frequency
S – linear displacement per pulse

$$H = \frac{\gamma}{S \cdot d} \cdot \frac{1}{f}$$



10kHz Pulse Train

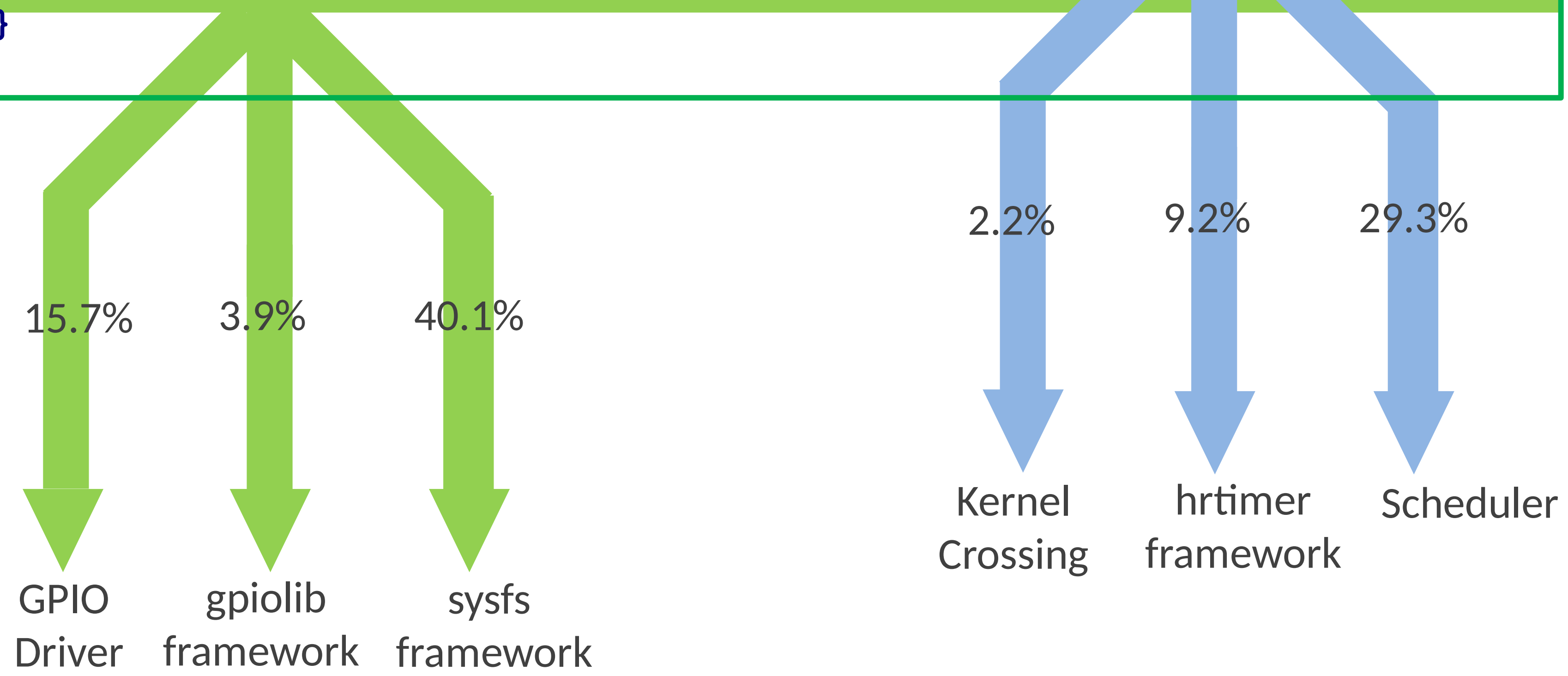
```
struct timespec period = {.tv_sec = 0, .tv_nsec = 100000}; while
(1) {
  nanosleep(&period, NULL);      /* sleep for 100 us */
  mraa_gpio_write(GPIO6, HIGH); /* write 1 to gpio6 */
  mraa_gpio_write(GPIO6, LOW);  /* write 0 to gpio6 */
}
```

	Frequency	Period
Theoretical	10 kHz	100000 ns
Linux	7.91 kHz	100000 ns + 26422 ns
Original PrintrBoard	9.96 kHz	100000 ns + 401 ns

Unstable {

10kHz Pulse Train (Linux)

```
struct timespec period = {.tv_sec = 0, .tv_nsec = 100000}; while
(1) {
  nanosleep(&period, NULL);       /* sleep for 100 us */
  mraa_gpio_write(GPIO6, HIGH); /* write 1 to gpio6 */
  mraa_gpio_write(GPIO6, LOW);  /* write 0 to gpio6 */
}
```



Lack of API with low and predictable overheads

QduinoMC

Goals	Design
Easy to use	Simple APIs based on Arduino
Easy to port existing Arduino programs	
Leverage multiple cores	Multithread loops
	Pinning loops to cores
	Interrupt routing
Allow QoS specification	Loop budget and period
Low I/O access overhead	User-level I/O access

loop (loopID, budget, period, [coreID])

noInterrupts (device, coreID)
noTimer (coreID)

interruptsVcpu (device, budget, period, [coreID])

digitalWrite () / digitalRead ()

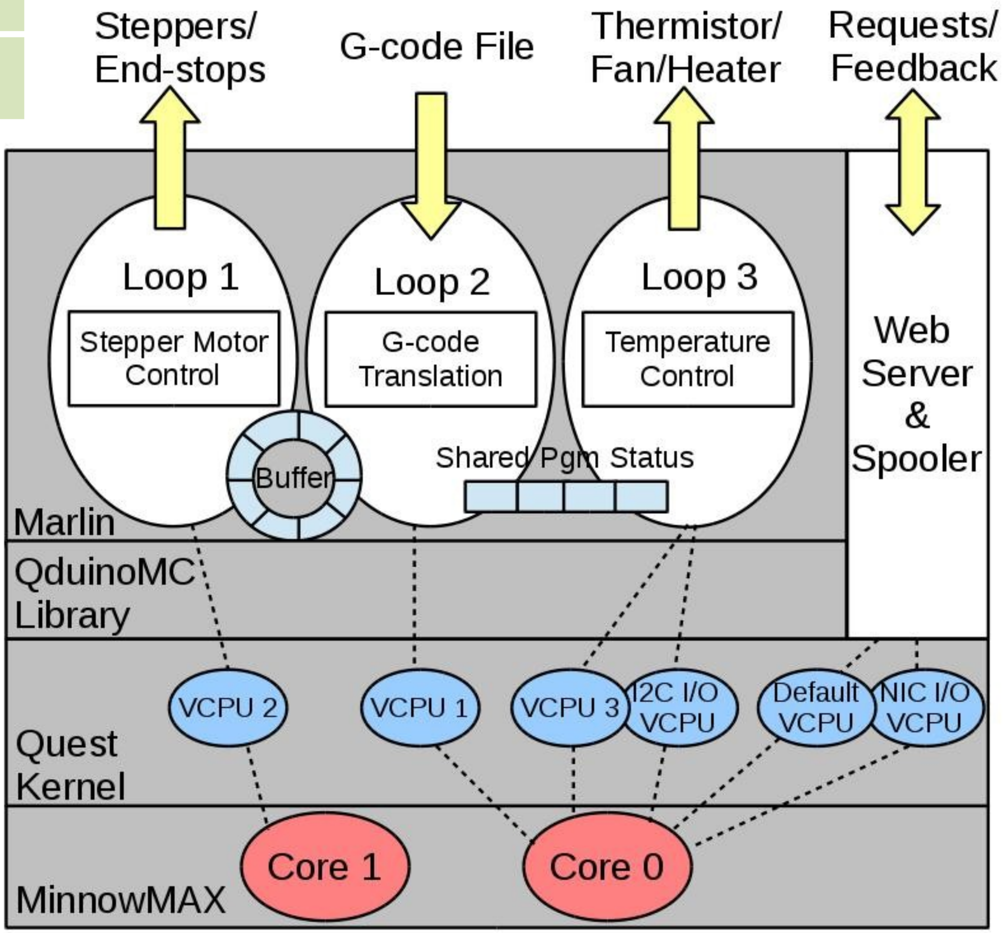
Marlin on QduinoMC

loop (1, 10, 100, 1), loop (2, 30, 100, 0), loop (3, 1, 80, 0)

interruptsVCPU (I2C, 10ms, 100ms), interruptsVCPU (NIC, 10ms, 100ms)

noTimer (1), noInterrupts (ALL, 1)

Added Web server / Spooler



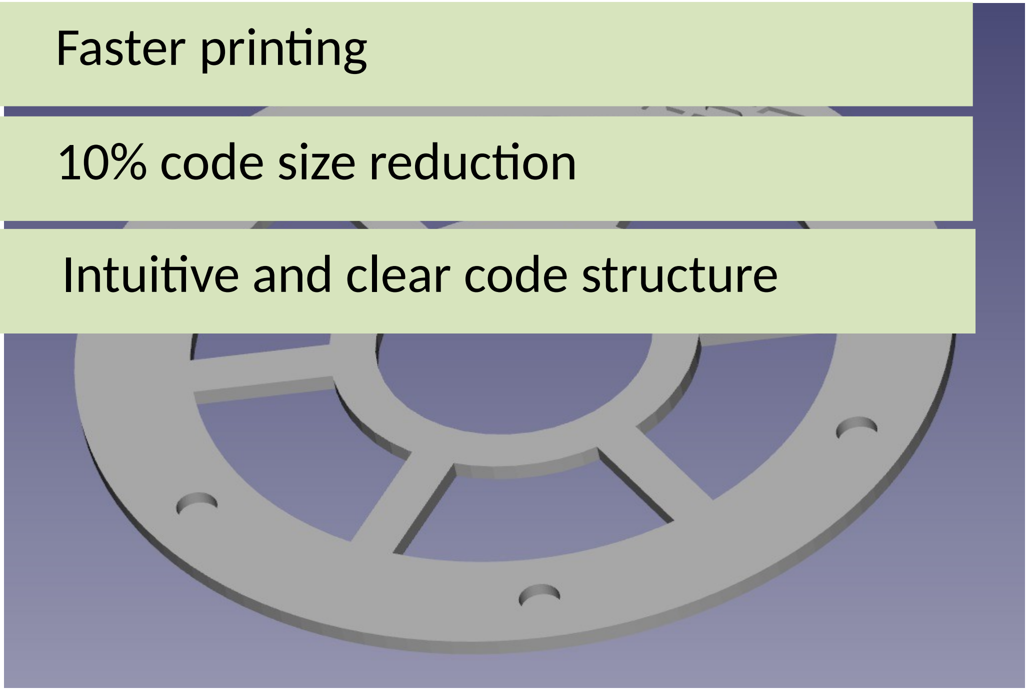
10kHz Pulse Train...Again

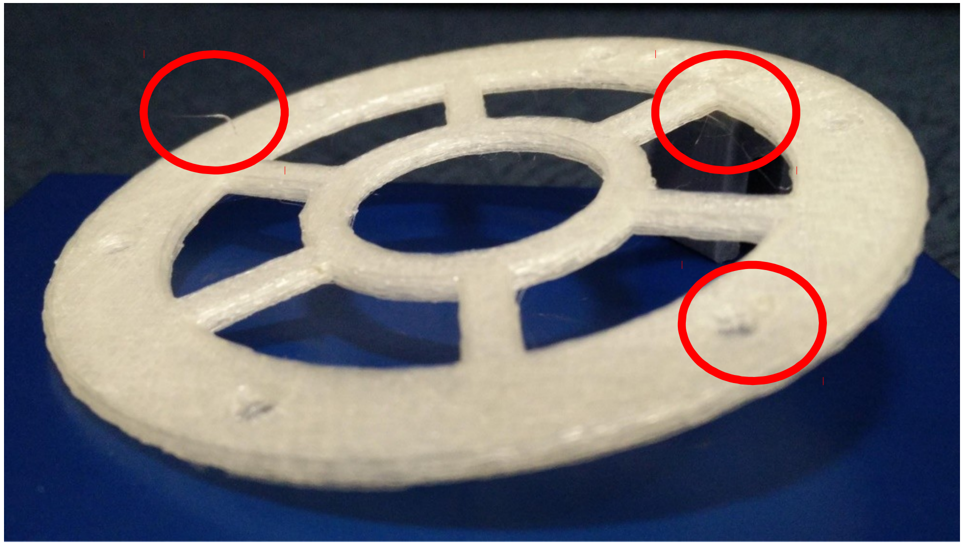
```
void setup ( ) {
 pinMode(GPIO6, OUTPUT);
 noInterrupts(ALL, 1); noTimer(1);
}

void loop (1, 100, 100, 1) {
 delayBusyNanoseconds(100000);
 digitalWrite(GPIO6, 1); digitalWrite(GPIO6, 0);
}
```

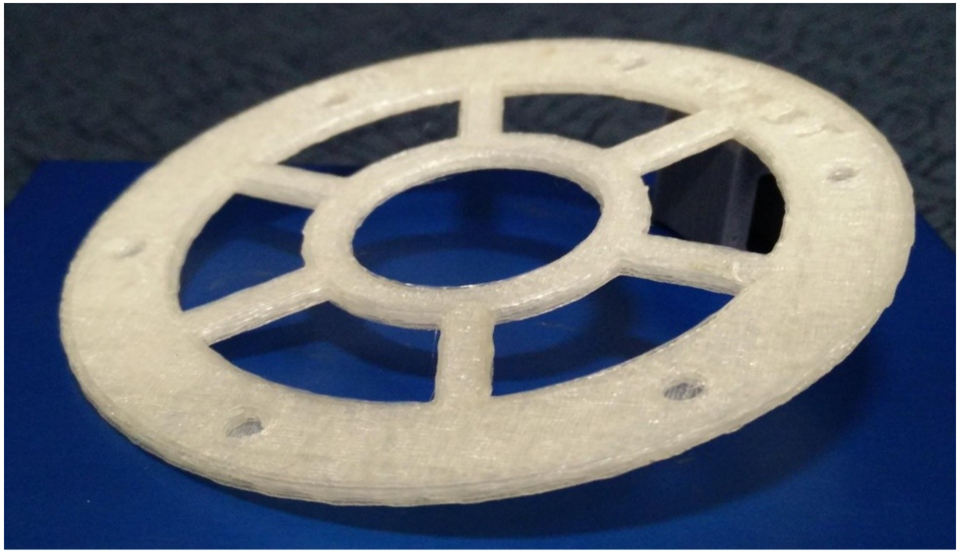
		Frequency	Period
	Theoretical	10 kHz	100000 ns
Stable {	QduinoMC	9.569 kHz	100000 ns + 4504 ns
	Linux	7.91 kHz	100000 ns + 26422 ns
	Original PrintrBoard	9.96 kHz	100000 ns + 401 ns

Test Object

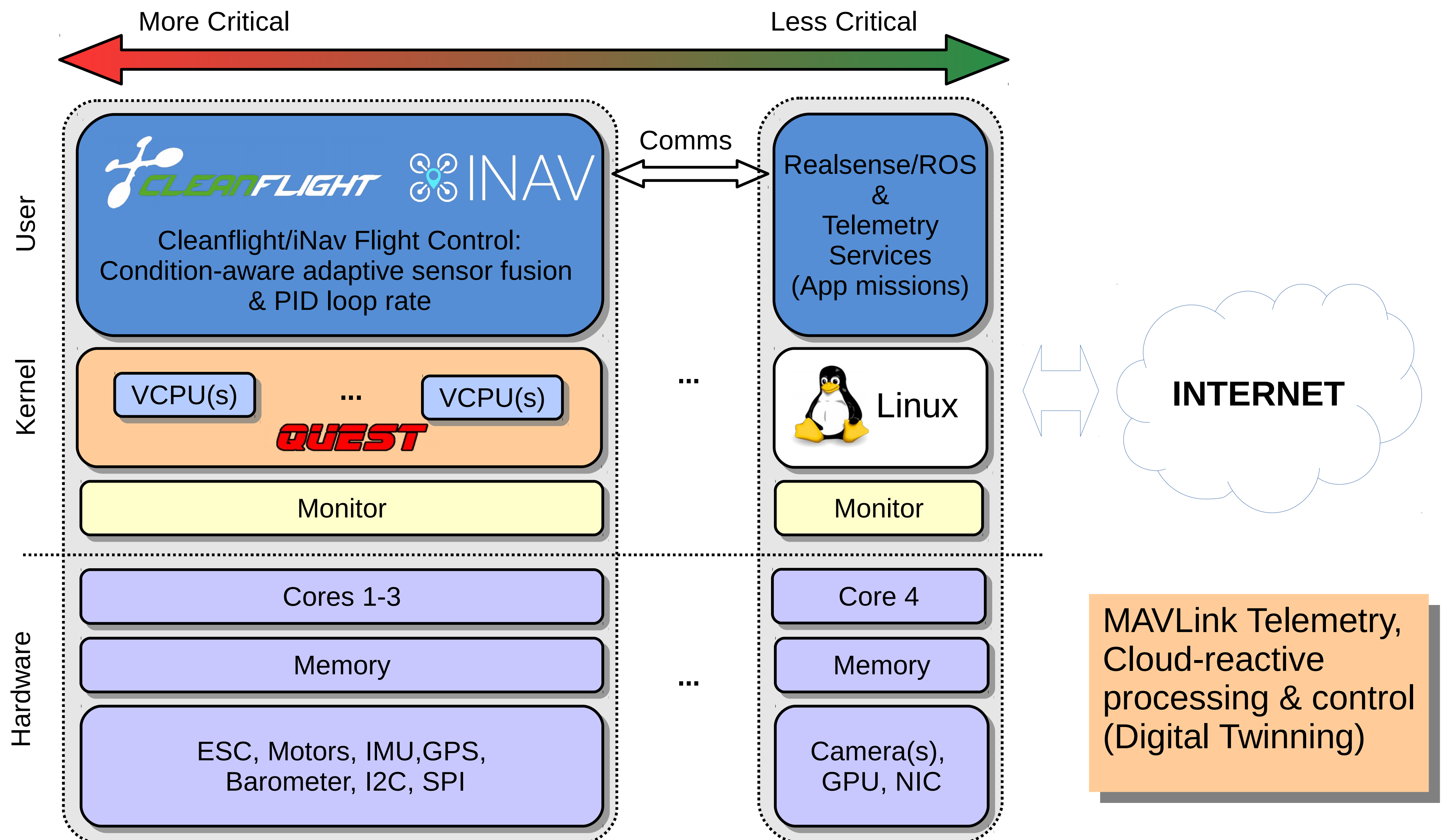
- Higher quality
- Faster printing
- 10% code size reduction
- Intuitive and clear code structure



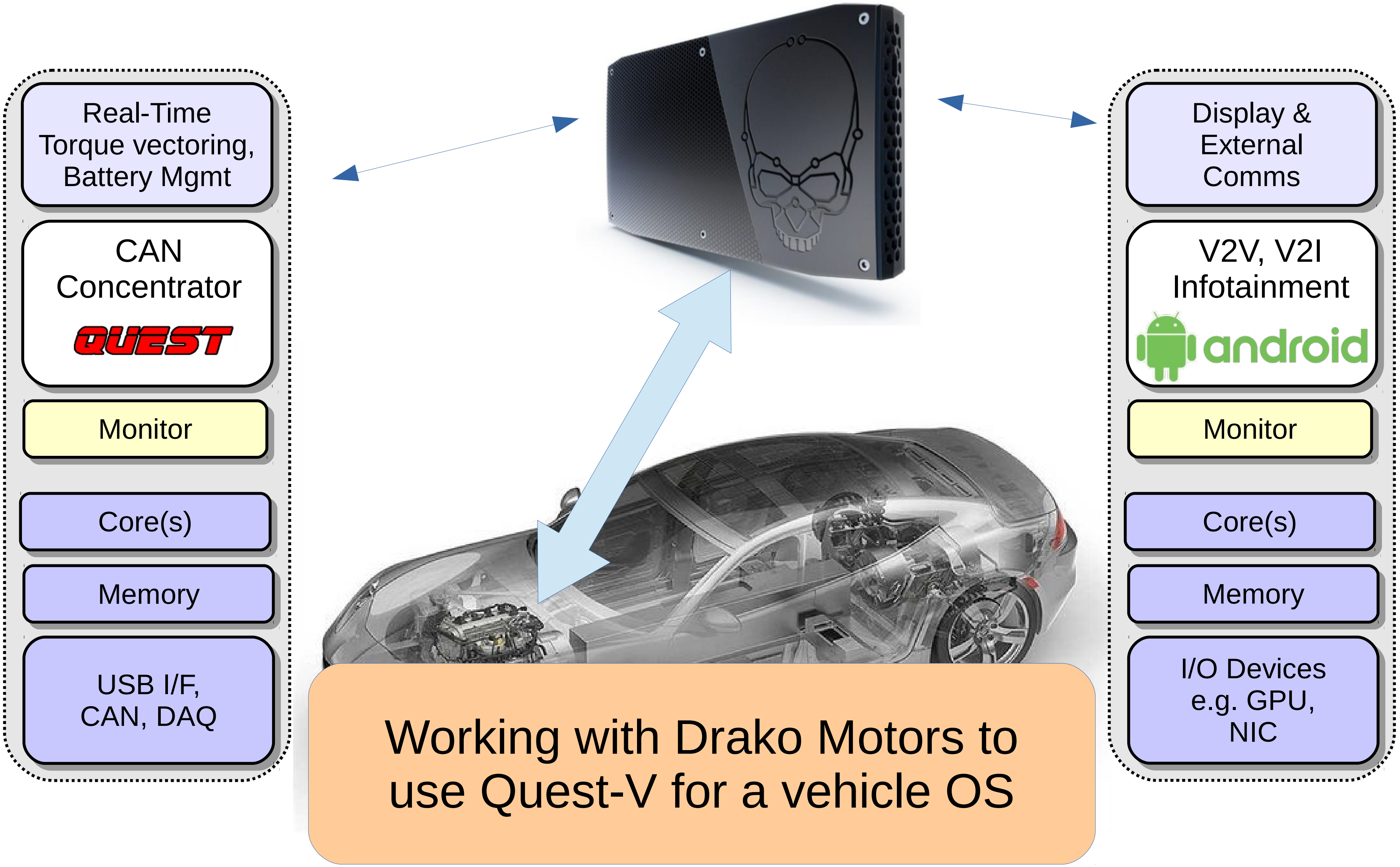




Quest-V DroneOS for Intel Aero



Quest-V DriveOS for Skull Canyon



Final Words

- Drako Motors want to build DriveOS for cars
- Would like an x86 reference architecture for embedded systems
  - PC with iGPU, GPIOs, I2C, SPI, UARTs, ADCs, CAN, etc
  - There is less standardization with ARM
    - ACPI not common in embedded ARM
    - Need device tree configurations setup and loaded by bootloader
- Processing needs of next-gen smart devices requires more capable processors than current embedded CPUs