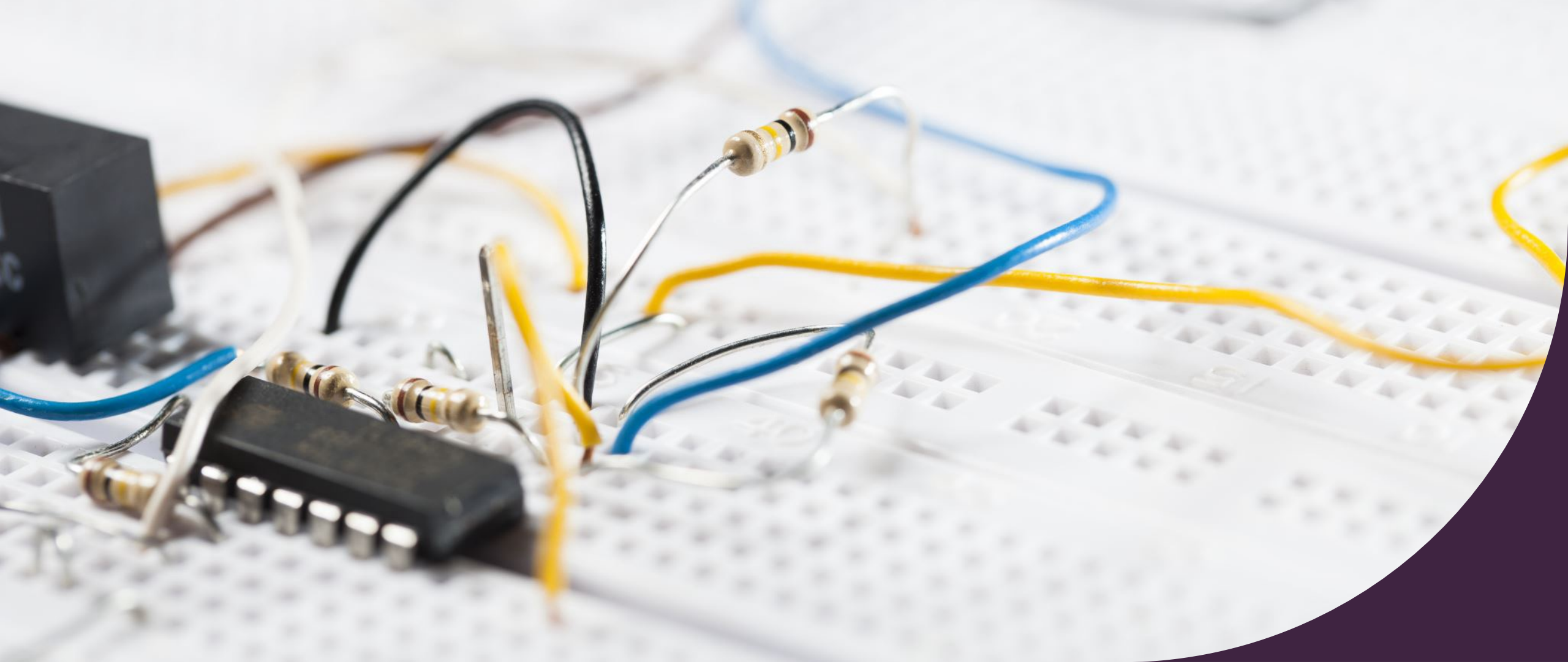


Embedded Systems Part II

Anupam Sobti

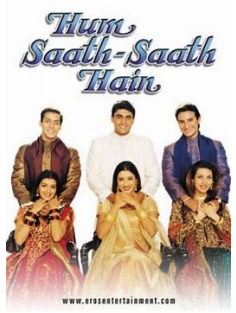
References

- Playlist ESP32 + ZephyrOS
- Playlist Intro to RTOS
- Why RTOS? – YouTube
- RTOSlecture28.ppt (mit.edu)
- 16.070-Week12-RTOS1.ppt (mit.edu)
- Introduction to Embedded Linux Part 1 - Buildroot | Digi-Key Electronics - YouTube

Remember, Remember

Virtualization

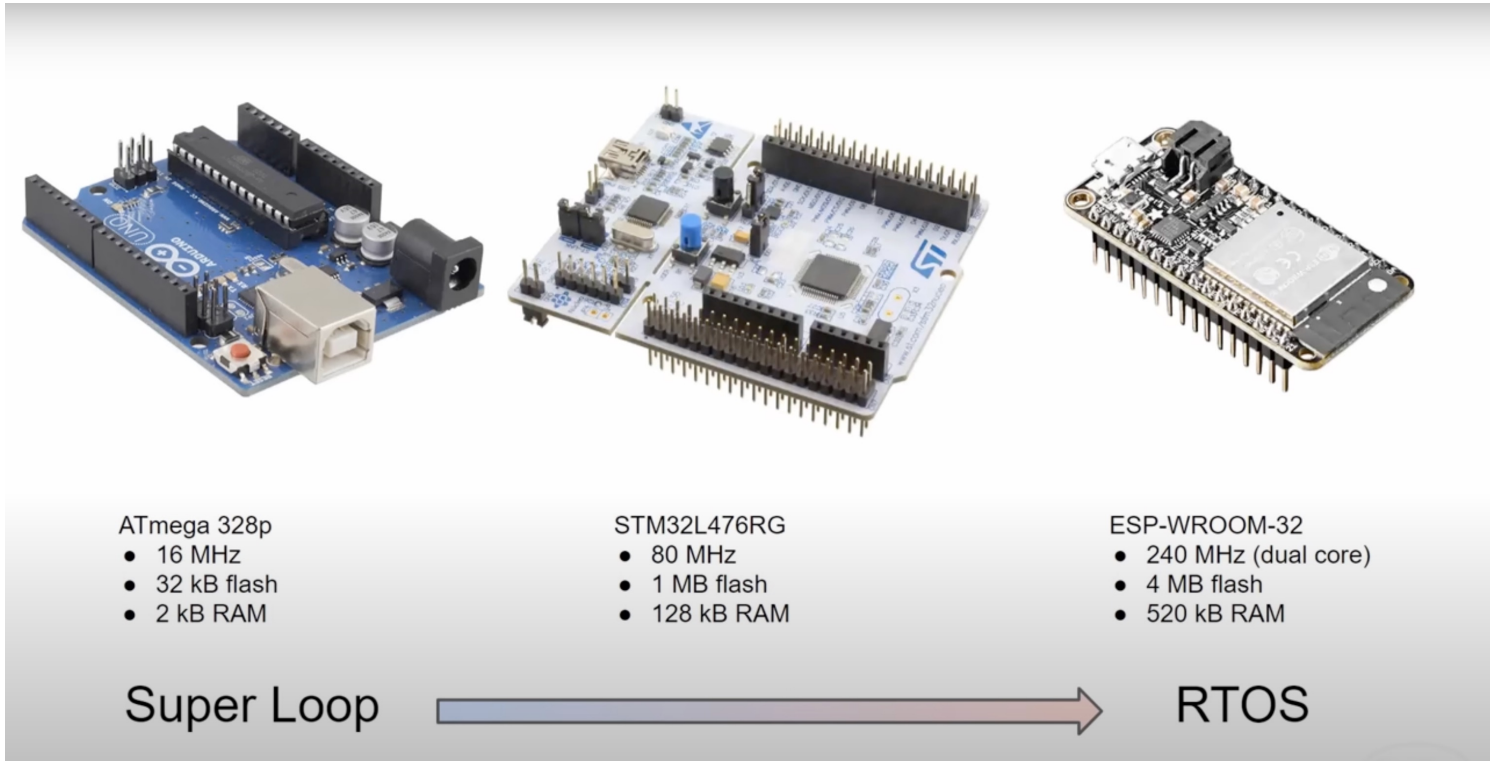
Concurrency



Persistence

From the book operating systems: three-easy-pieces

But when do you need them?



Increasing power comes with increasing complexity

Handling complexity

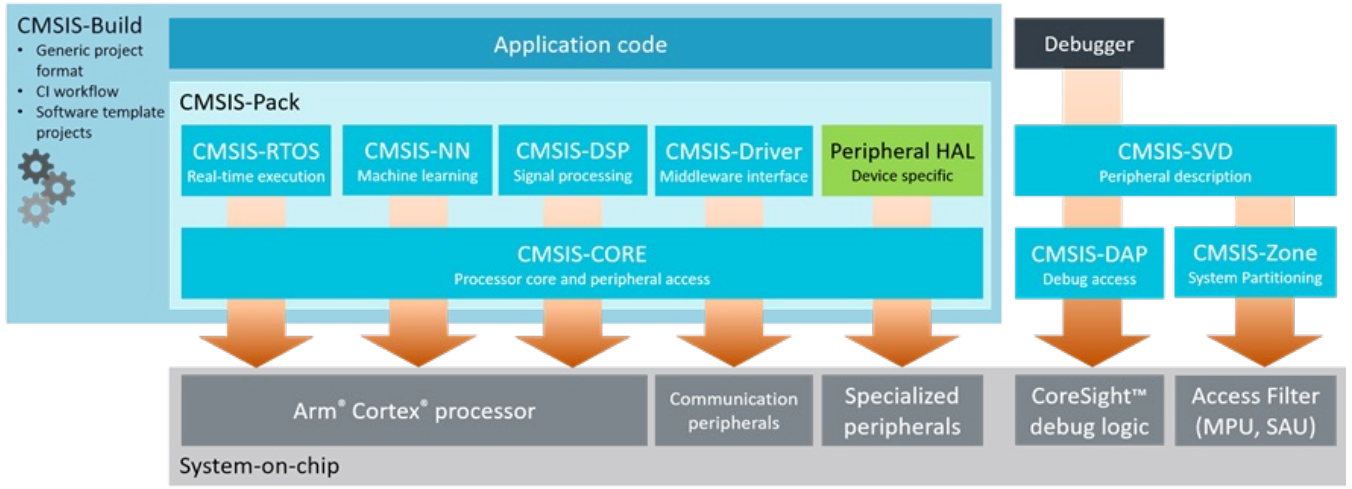
- Use libraries
  - The compilation is still done together.
  - Control flow for all the threads needs to be understood by the programmer.
- Use a process and memory abstraction
  - RTOS
- Build on a common set of tools and environment with various services for commonly used peripherals, process, thread and memory management, inter-process communication, etc.
  - General purpose OS – primarily for human interaction

Libraries

The Arduino environment can be extended through the use of libraries, just like most programming platforms. Libraries provide extra functionality for use in sketches, e.g. working with hardware or manipulating data. To use a library in a sketch, select it from **Sketch > Import Library**.
A number of libraries come installed with the IDE, but you can also download or create your own. See these instructions for details on installing libraries. There is also a tutorial on writing your own libraries. See the API Style Guide for information on making a good Arduino-style API for your library.

- Communication (1360)
- Data Processing (369)
- Data Storage (162)
- Device Control (1097)
- Display (540)
- Other (521)
- Sensors (1281)
- Signal Input/Output (493)
- Timing (239)
- Uncategorized (239)

Libraries - Arduino Reference



Common Microcontroller Software Interface Standard (CMSIS)
CMSIS enables consistent device support and simple software interfaces to the processor and its peripherals, simplifying software reuse, reducing the learning curve for microcontroller developers, and reducing the time to market for new devices.

Supports 5000+ devices!

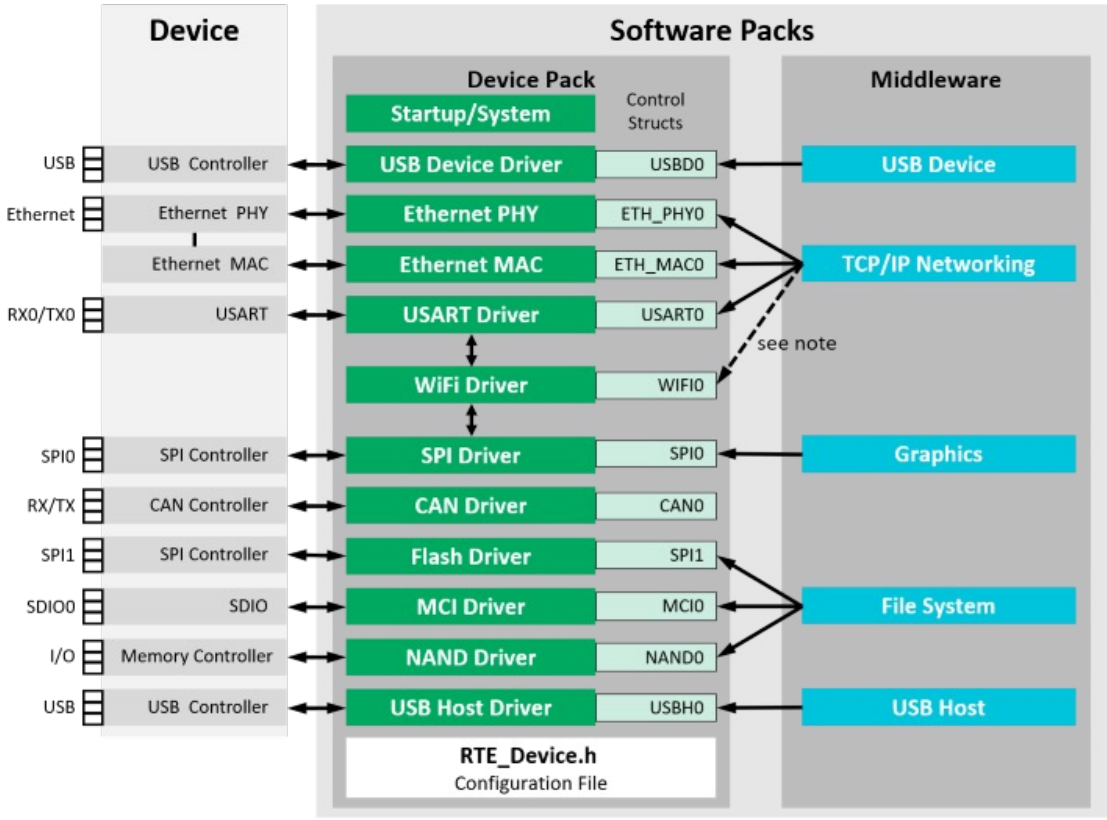
CMSIS – Arm®

CMSIS Libraries

CMSIS Resources

Everything you need to know to make the right decision for your project. Includes technical documentation, industry insights, and where to go for expert advice.

- CMSIS-RTOS is an API that enables consistent software layers with middleware and library components
- CMSIS-DSP library is a rich collection of DSP functions that Arm has optimized for the various Cortex-M processor cores
- CMSIS-Driver interfaces are available for many microcontroller families
- CMSIS-Pack defines the structure of a software pack containing software components
- CMSIS-SVD files enable detailed views of device peripherals with current register state
- CMSIS-DAP is a standardized interface to the Cortex Debug Access Port (DAP)
- CMSIS-NN is a collection of efficient neural network kernels



Operating System

- Manage resources
  - Who is using what and for how long
  - Who's been waiting?
- Concurrent Execution
  - Who will run and for how long?
  - What should be the priority of the task
- Multi-user security
- Graphics support
- Networking support
- Peripherals communication

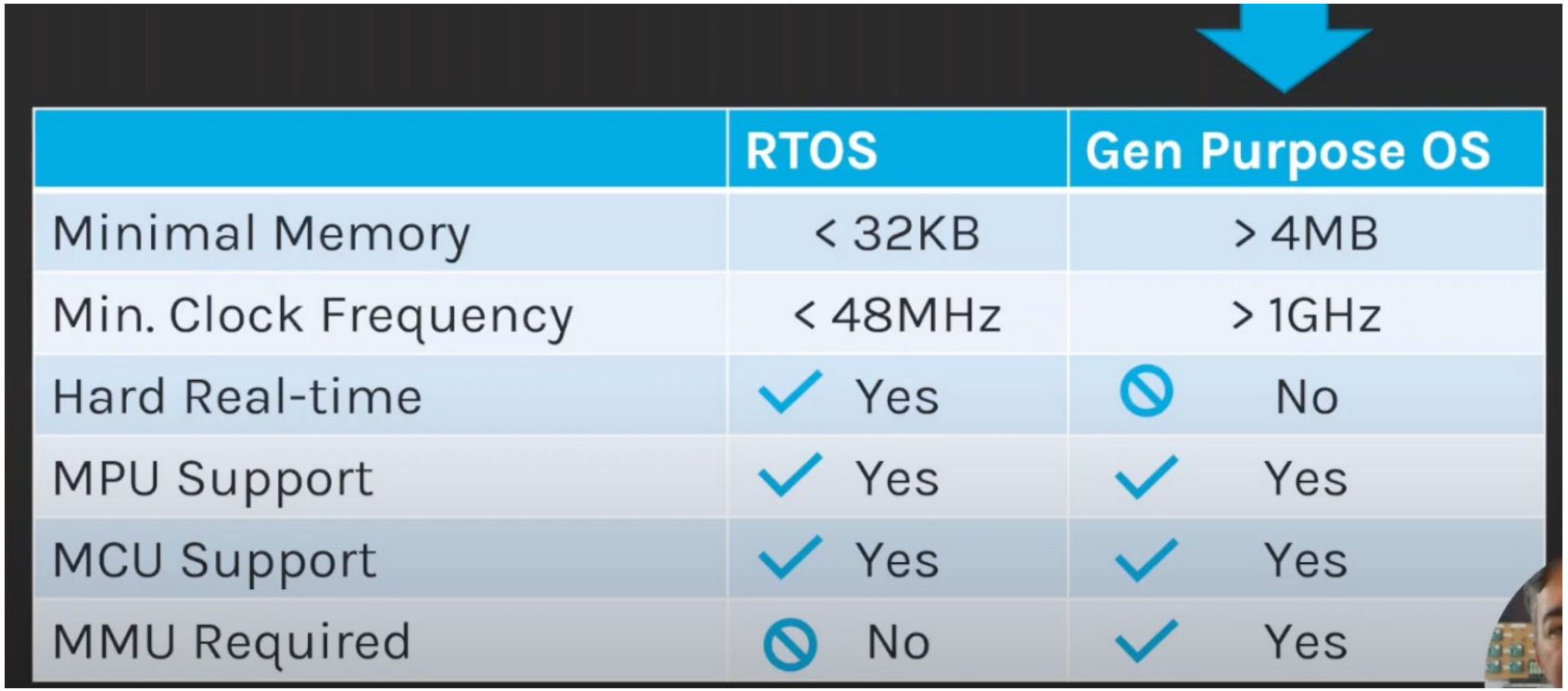
Real-time Operating System

- Stripped off operating system with essentials
  - User shell, file/disk access may be unnecessary
  - You get control on the resources
    - No background processes
    - Bounded no of tasks
  - You get control on the timing
    - Manipulation of task priorities
    - Choice of scheduling options

Scheduler – Dispatcher – Intertask Communication

16.070-Week12-RTOS1.ppt (mit.edu)

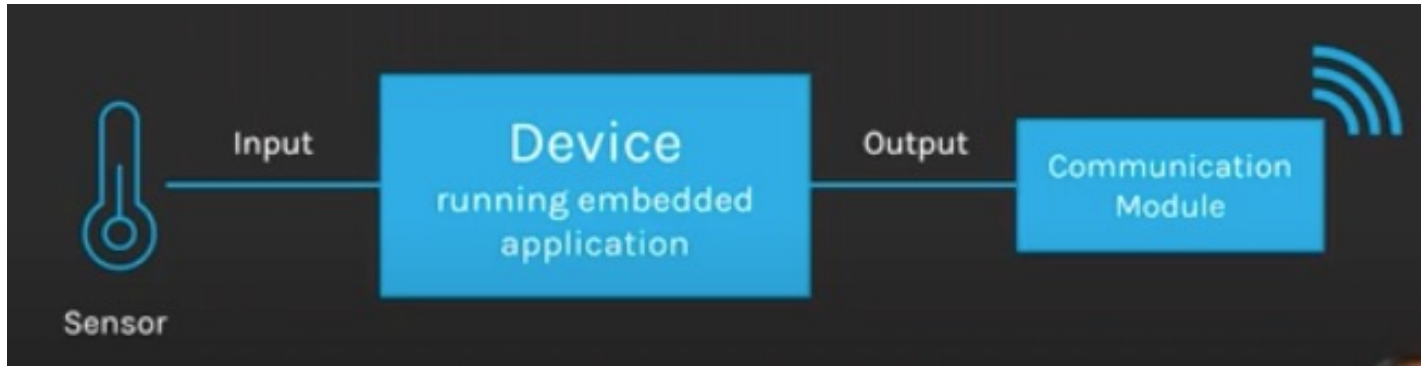
Resource Requirements

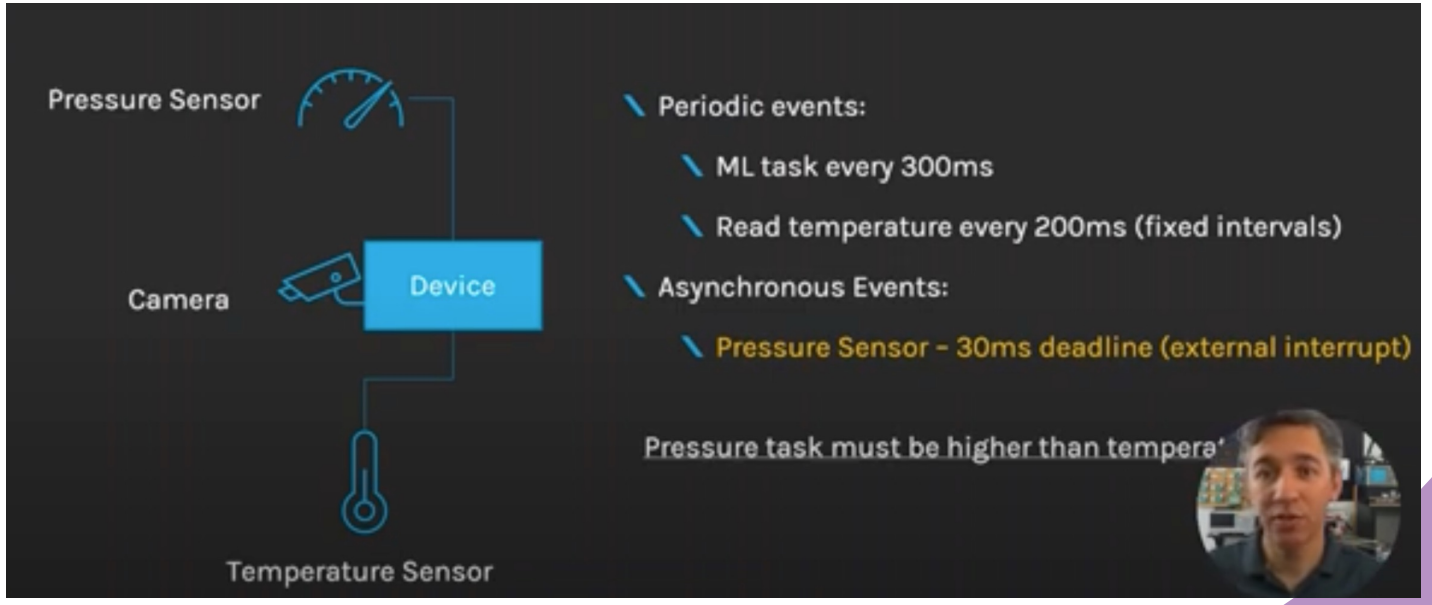


	RTOS	Gen Purpose OS
Minimal Memory	< 32KB	> 4MB
Min. Clock Frequency	< 48MHz	> 1GHz
Hard Real-time	✓ Yes	⊘ No
MPU Support	✓ Yes	✓ Yes
MCU Support	✓ Yes	✓ Yes
MMU Required	⊘ No	✓ Yes

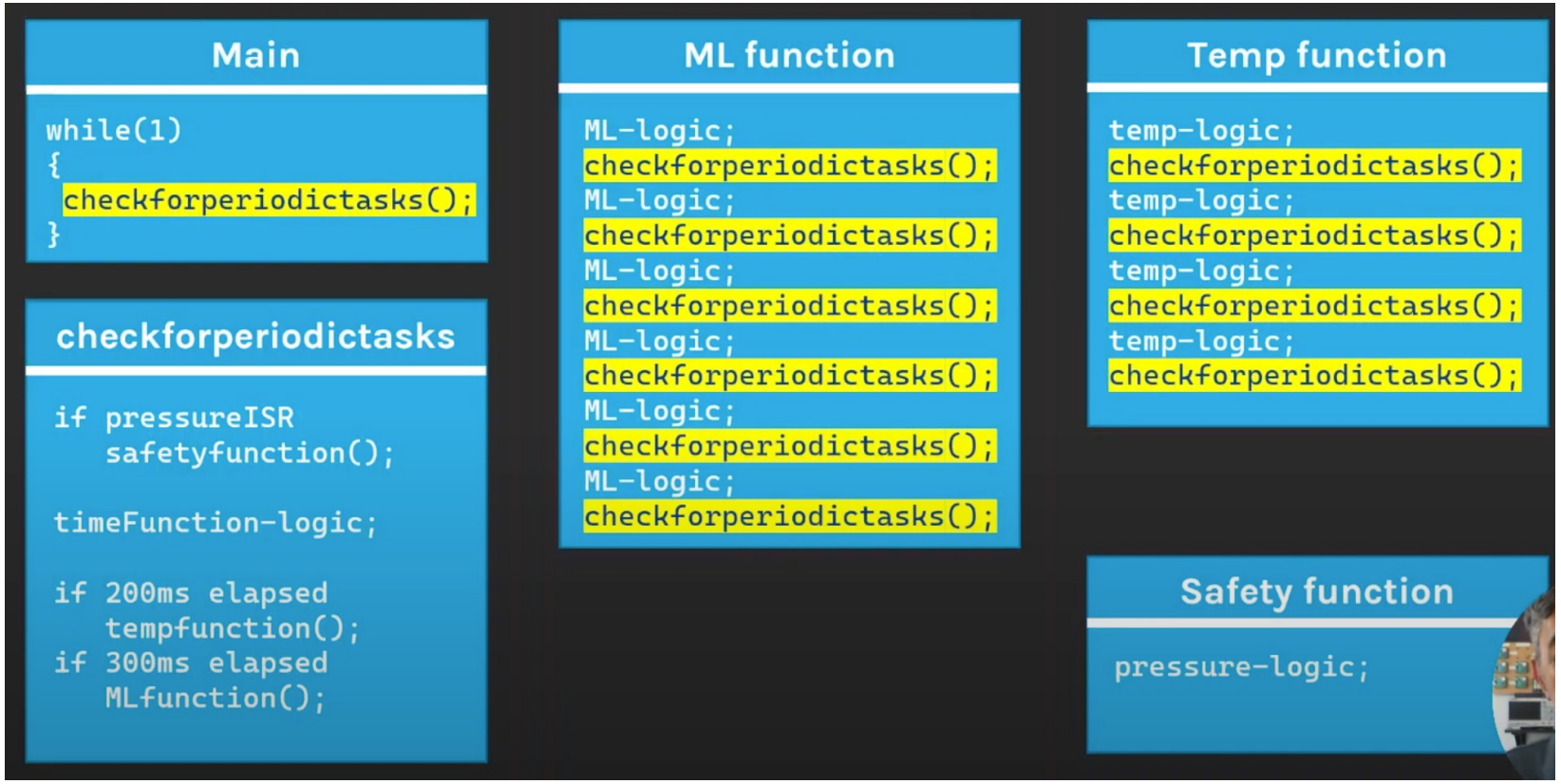
Typical Embedded Application

Imagine
- Pressure in a chamber that might explode
- Tilt in a rocket that's landing!
- Wind in a flight that's ascending!





No RTOS


Main
while(1)
{
checkforperiodictasks();
}
checkforperiodictasks
if pressureISR
safetyfunction();
timeFunction-logic;
if 200ms elapsed
tempfunction();
if 300ms elapsed
MLfunction();
ML function
ML-logic;
checkforperiodictasks();
ML-logic;
checkforperiodictasks();
ML-logic;
checkforperiodictasks();
ML-logic;
checkforperiodictasks();
ML-logic;
checkforperiodictasks();
ML-logic;
checkforperiodictasks();
Temp function
temp-logic;
checkforperiodictasks();
temp-logic;
checkforperiodictasks();
temp-logic;
checkforperiodictasks();
temp-logic;
checkforperiodictasks();
Safety function
pressure-logic;

With RTOS

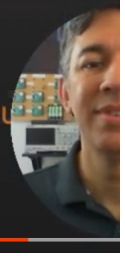
```c
int main(…)
{
  pthread_create(&thread_0, NULL, machine_learning_entry, NULL);
  pthread_create(&thread_1, NULL, temperature_entry, NULL);
  pthread_create(&thread_2, NULL, safety_entry, NULL);
}
void* machine_learning_entry(void* argument)
{
  while (1)
  {
    wait_for_300ms_interrupt();
    perform_machine_learning();
  }
}
void* temperature_entry(void* argument)
{
  while (1)
  {
    Wait_for_200ms_interrupt();
    read_temperature();
  }
}
void* safety_entry(void* argument)
{
  while (1)
  {
    wait_for_pressure_sensor_interrupt();
    open_relief_valve();
  }
}
```

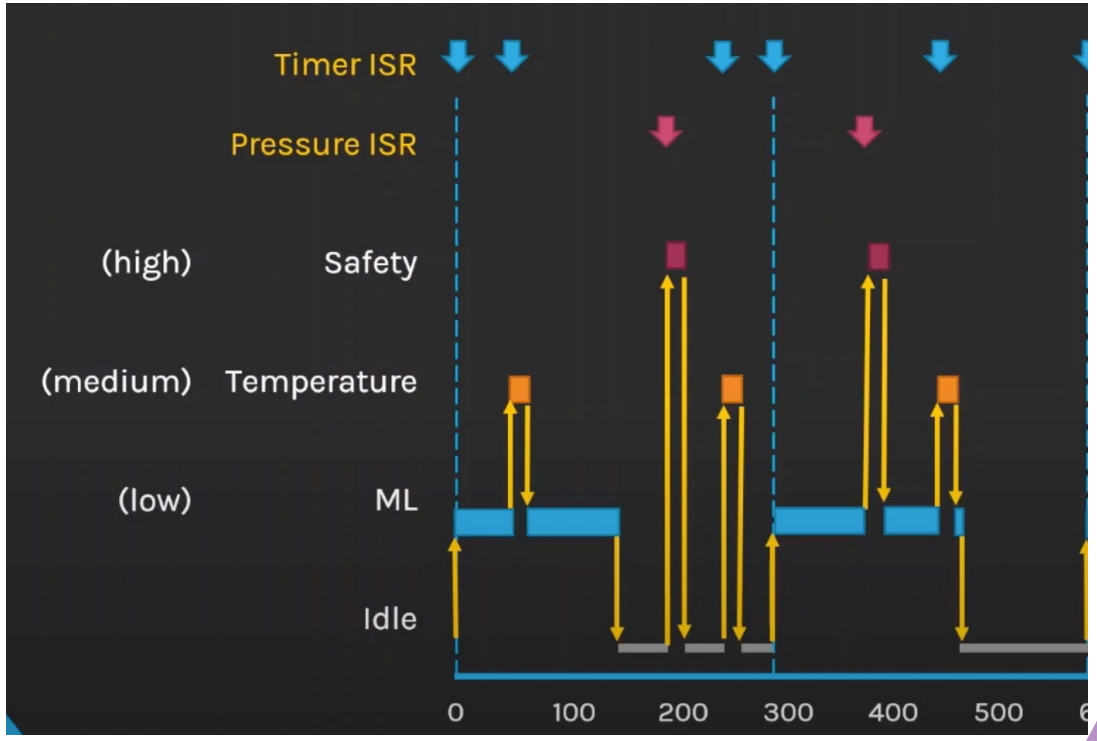
Enhance \ Simp

Thread 1: ML

Thread 2: Temperature

Thread 3: Pressu





Which RTOS?

VxWorks Operating System

freeRTOS

Owned and
maintained by AWS

An initiative by
The Linux Foundation

Azure RTOS system components

Every Azure RTOS component is fully connected, easy to use—and helps developers get to market faster.

Azure RTOS ThreadX
A high-performance real-time operating system

Azure RTOS NetX Duo
A TCP/IP IPv4/IPv6 embedded network stack that includes cloud connectivity and IPsec and TLS/DTLS security protocols

Azure RTOS FileX
An embedded FAT file system that offers optional fault tolerant features

Azure RTOS GUIX Studio and GUIX
A complete design environment and run-time to create and maintain 2D graphical user interfaces

Azure RTOS USBX
A USB stack that provides host, device, and on-the-go support

Azure RTOS LevelX
A framework provides NAND and NOR flash wear leveling facilities to embedded applications

Google and Facebook Select Zephyr RTOS for Next Generation Products

September 30, 2020

How to use the RTOS? (FreeRTOS)

- Create new tasks (example 1)

```
int main( void )
{
    /* Create one of the two tasks. */
    xTaskCreate(    vTask1,     /* Pointer to the function that implements the task. */
                    "Task 1",   /* Text name for the task.  This is to facilitate debugging only. */
                    1000,       /* Stack depth - most small microcontrollers will use much less stack
                    NULL,       /* We are not using the task parameter. */
                    1,          /* This task will run at priority 1. */
                    NULL );     /* We are not using the task handle. */

    /* Create the other task in exactly the same way. */
    xTaskCreate( vTask2, "Task 2", 1000, NULL, 1, NULL );

    /* Start the scheduler to start the tasks executing. */
    vTaskStartScheduler();

    /* The following line should never be reached because vTaskStartScheduler()
    will only return if there was not enough FreeRTOS heap memory available to
    create the Idle and (if configured) Timer tasks.  Heap management, and
    techniques for trapping heap exhaustion, are described in the book text. */
    for( ;; );
    return 0;
}
```

FreeRTOS Documentation and Book (just download examples)

Don’t hog the CPU (at least in high priority tasks) – ex3

```c
int main( void )
{
    /* Create the first task at priority 1... */
    xTaskCreate( vTaskFunction, "Task 1", 1000, (void*)pcTextForTask1, 1, NULL );

    /* ... and the second task at priority 2.  The priority is the second to
    last parameter. */
    xTaskCreate( vTaskFunction, "Task 2", 1000, (void*)pcTextForTask2, 2, NULL );

    /* Start the scheduler to start the tasks executing. */
    vTaskStartScheduler();

    /* The following line should never be reached because vTaskStartScheduler()
    will only return if there was not enough FreeRTOS heap memory available to
    create the Idle and (if configured) Timer tasks.  Heap management, and
    techniques for trapping heap exhaustion, are described in the book text. */
    for( ;; );
    return 0;
}
```

```c
void vTaskFunction( void *pvParameters )
{
char *pcTaskName;
volatile uint32_t ul;

    /* The string to print out is passed in via the parameter.  Cast this to a
    character pointer. */
    pcTaskName = ( char * ) pvParameters;

    /* As per most tasks, this task is implemented in an infinite loop. */
    for( ;; )
    {
        /* Print out the name of this task. */
        vPrintString( pcTaskName );

        /* Delay for a period. */
        for( ul = 0; ul < mainDELAY_LOOP_COUNT; ul++ )
        {
            /* This loop is just a very crude delay implementation.  There is
            nothing to do in here.  Later exercises will replace this crude
            loop with a proper delay/sleep function. */
        }
    }
}
```

Instead use vTaskDelay() that tells the OS (ex4) that the process will resume after x seconds

```c
vTaskDelay( pdMS_TO_TICKS( 1000 ) );
```

Or if you're worried about variable execution time (ex5)

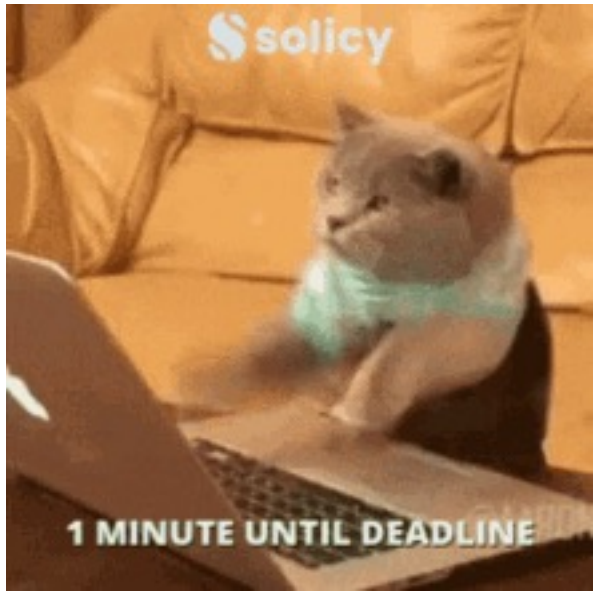
```
void vTaskFunction( void *pvParameters )
{
char *pcTaskName;
TickType_t xLastWakeTime;
const TickType_t xDelay250ms = pdMS_TO_TICKS( 250UL );

    /* The string to print out is passed in via the parameter.  Cast this to a
    character pointer. */
    pcTaskName = ( char * ) pvParameters;

    /* The xLastWakeTime variable needs to be initialized with the current tick
    count.  Note that this is the only time we access this variable.  From this
    point on xLastWakeTime is managed automatically by the vTaskDelayUntil()
    API function. */
    xLastWakeTime = xTaskGetTickCount();

    /* As per most tasks, this task is implemented in an infinite loop. */
    for( ;; )
    {
        /* Print out the name of this task. */
        vPrintString( pcTaskName );

        /* We want this task to execute exactly every 250 milliseconds.  As per
        the vTaskDelay() function, time is measured in ticks, and the
        pdMS_TO_TICKS() macro is used to convert this to milliseconds.
        xLastWakeTime is automatically updated within vTaskDelayUntil() so does not
        have to be updated by this task code. */
        vTaskDelayUntil( &xLastWakeTime, xDelay250ms );
    }
}
```

Khali baithe kya karein (What shall we do while sitting idle?)

```
void vTaskFunction( void *pvParameters )
{
char *pcTaskName;
const TickType_t xDelay250ms = pdMS_TO_TICKS( 250UL );

    /* The string to print out is passed in via the parameter.  Cast this to a
    character pointer. */
    pcTaskName = ( char * ) pvParameters;

    /* As per most tasks, this task is implemented in an infinite loop. */
    for( ;; )
    {
        /* Print out the name of this task AND the number of times ulIdleCycleCount
        has been incremented. */
        vPrintStringAndNumber( pcTaskName, ulIdleCycleCount );

        /* Delay for a period.  This time we use a call to vTaskDelay() which
        puts the task into the Blocked state until the delay period has expired.
        The delay period is specified in 'ticks'. */
        vTaskDelay( xDelay250ms );
    }
}
/*-----------------------------------------------------------*/

/* Idle hook functions MUST be called vApplicationIdleHook(), take no parameters,
and return void. */
void vApplicationIdleHook( void )
{
    /* This hook function does nothing but increment a counter. */
    ulIdleCycleCount++;
}
```

Modify priorities on the go – ex9

```
void vTask1( void *pvParameters )
{
UBaseType_t uxPriority;

    /* This task will always run before Task2 as it has the higher priority.
    Neither Task1 nor Task2 ever block so both will always be in either the
    Running or the Ready state.

    Query the priority at which this task is running - passing in NULL means
    "return our own priority". */
    uxPriority = uxTaskPriorityGet( NULL );

    for( ;; )
    {
        /* Print out the name of this task. */
        vPrintString( "Task1 is running\r\n" );

        /* Setting the Task2 priority above the Task1 priority will cause
        Task2 to immediately start running (as then Task2 will have the higher
        priority of the    two created tasks). */
        vPrintString( "About to raise the Task2 priority\r\n" );
        vTaskPrioritySet( xTask2Handle, ( uxPriority + 1 ) );

        /* Task1 will only run when it has a priority higher than Task2.
        Therefore, for this task to reach this point Task2 must already have
        executed and set its priority back down to 0. */
    }
}
```



Producer Consumer (ex10)

```c
int main( void )
{
    /* The queue is created to hold a maximum of 5 long values. */
    xQueue = xQueueCreate( 5, sizeof( int32_t ) );

    if( xQueue != NULL )
    {
        /* Create two instances of the task that will write to the queue.  The
        parameter is used to pass the value that the task should write to the queue,
        so one task will continuously write 100 to the queue while the other task
        will continuously write 200 to the queue.  Both tasks are created at
        priority 1. */
        xTaskCreate( vSenderTask, "Sender1", 1000, ( void * ) 100, 1, NULL );
        xTaskCreate( vSenderTask, "Sender2", 1000, ( void * ) 200, 1, NULL );

        /* Create the task that will read from the queue.  The task is created with
        priority 2, so above the priority of the sender tasks. */
        xTaskCreate( vReceiverTask, "Receiver", 1000, NULL, 2, NULL );

        /* Start the scheduler so the created tasks start executing. */
        vTaskStartScheduler();
    }
    else
    {
        /* The queue could not be created. */
    }

    /* The following line should never be reached because vTaskStartScheduler()
    will only return if there was not enough FreeRTOS heap memory available to
    create the Idle and (if configured) Timer tasks.  Heap management, and
    techniques for trapping heap exhaustion, are described in the book text. */
    for( ;; );
    return 0;
}
```

You can also send arbitrary data types using struct

Set timers to do tasks (ex13)

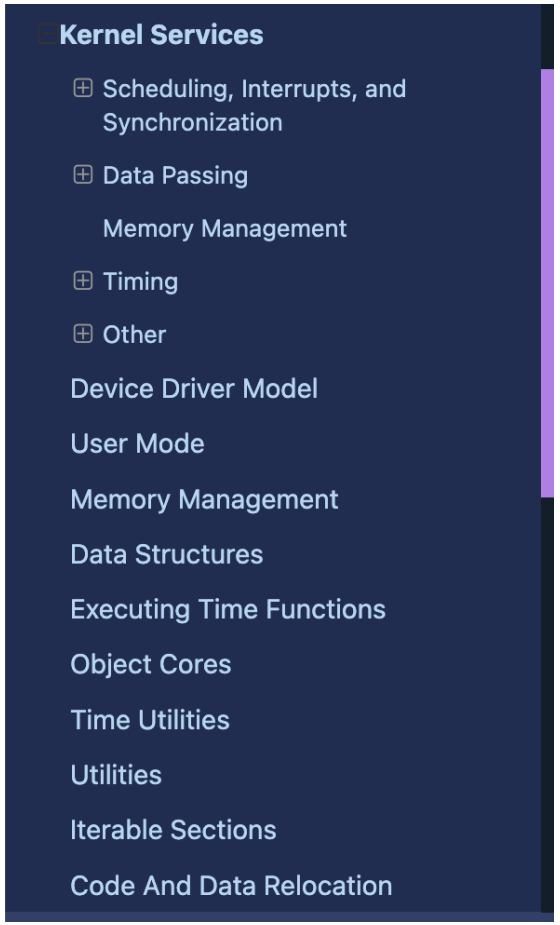
```
/* Create the one shot software timer, storing the handle to the created
software timer in xOneShotTimer. */
xOneShotTimer = xTimerCreate( "OneShot",                    /* Text name for the software timer - not used by F
                              mainONE_SHOT_TIMER_PERIOD,    /* The software timer's period in ticks. */
                              pdFALSE,                      /* Setting uxAutoRealod to pdFALSE creates a one-sh
                              0,                            /* This example does not use the timer id. */
                              prvOneShotTimerCallback );    /* The callback function to be used by the software

/* Create the auto-reload software timer, storing the handle to the created
software timer in xAutoReloadTimer. */
xAutoReloadTimer = xTimerCreate( "AutoReload",                 /* Text name for the software timer - not used
                                 mainAUTO_RELOAD_TIMER_PERIOD, /* The software timer's period in ticks. */
                                 pdTRUE,                       /* Set uxAutoRealod to pdTRUE to create an auto
                                 0,                            /* This example does not use the timer id. */
                                 prvAutoReloadTimerCallback ); /* The callback function to be used by the soft
```

One time or recurring

Similarly, other task execution models

- Semaphores
- Interrupt Handlers
- Condition Variables
- Events

And in other operating systems (Zephyr)



Scheduling, Interrupts, and Synchronization

These pages cover basic kernel services related to thread scheduling and synchronization.

- Threads
- Scheduling
- CPU Idling
- System Threads
- Workqueue Threads
- Operation without Threads
- Interrupts
- Polling API
- Semaphores
- Mutexes
- Condition Variables
- Events
- Symmetric Multiprocessing

Use your favorite RTOS

- Focus on execution model for your applications

  Are there multiple tasks that are interrelated?

  Which resources might they be using?

  Do you need locks for resources?

  Should there be multiple priorities?

  Can you put the system to sleep?

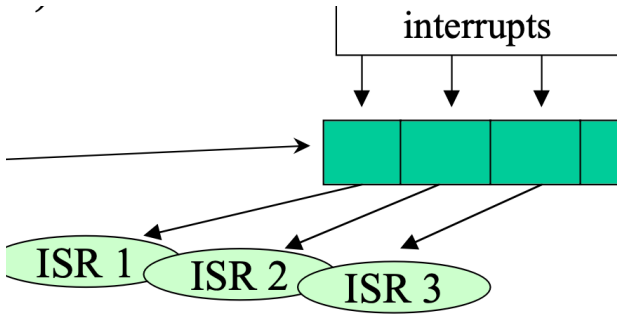
  Can you use interrupts?

The simplest

- Polled Loops
- Interrupt driven

```
while (TRUE)
    {
    if IFF_data_here==TRUE then
            {
            process_data();
            IFF_data_here=FALSE;
            }
    }
```

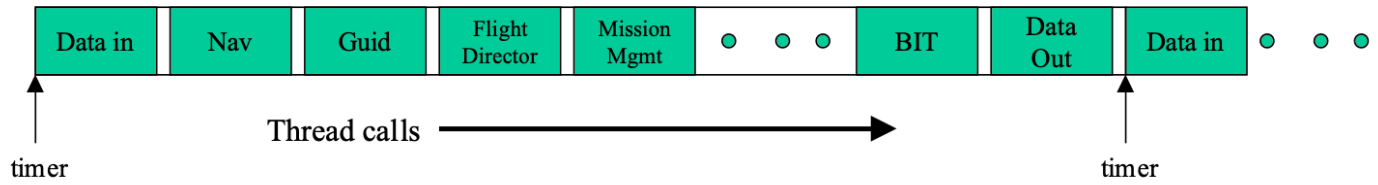
Polled Loop



Interrupt Driven

Designing multi-tasking systems

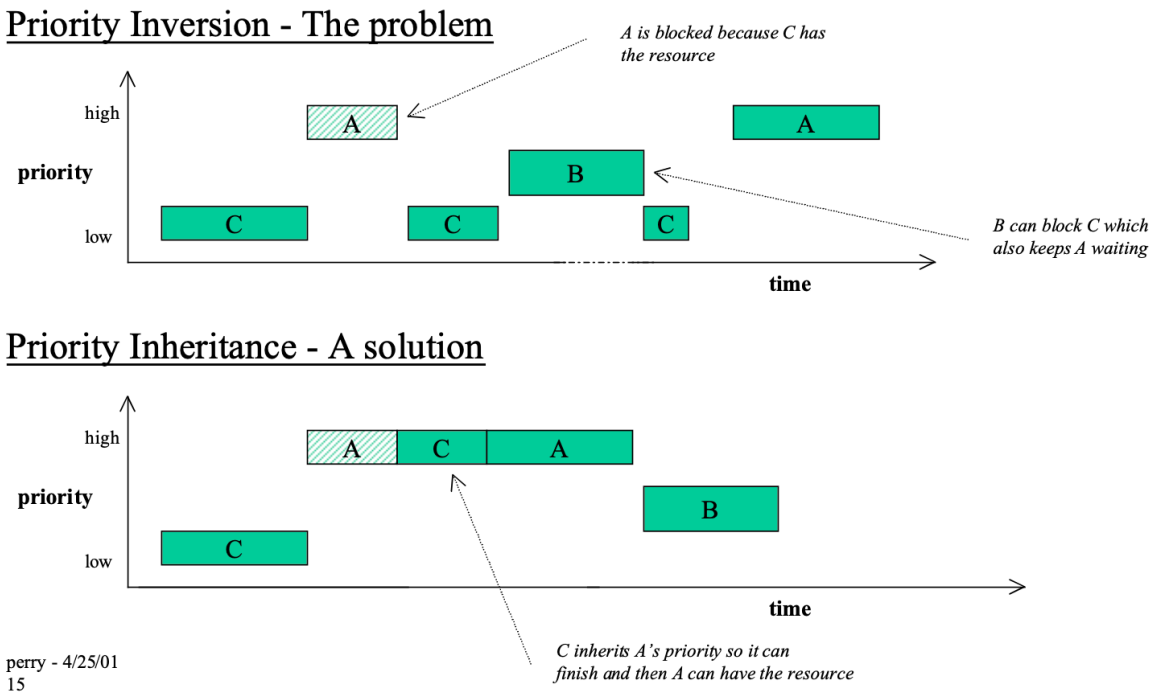
- Cyclic executive systems
- Round Robin Systems
- Preemptive priority systems



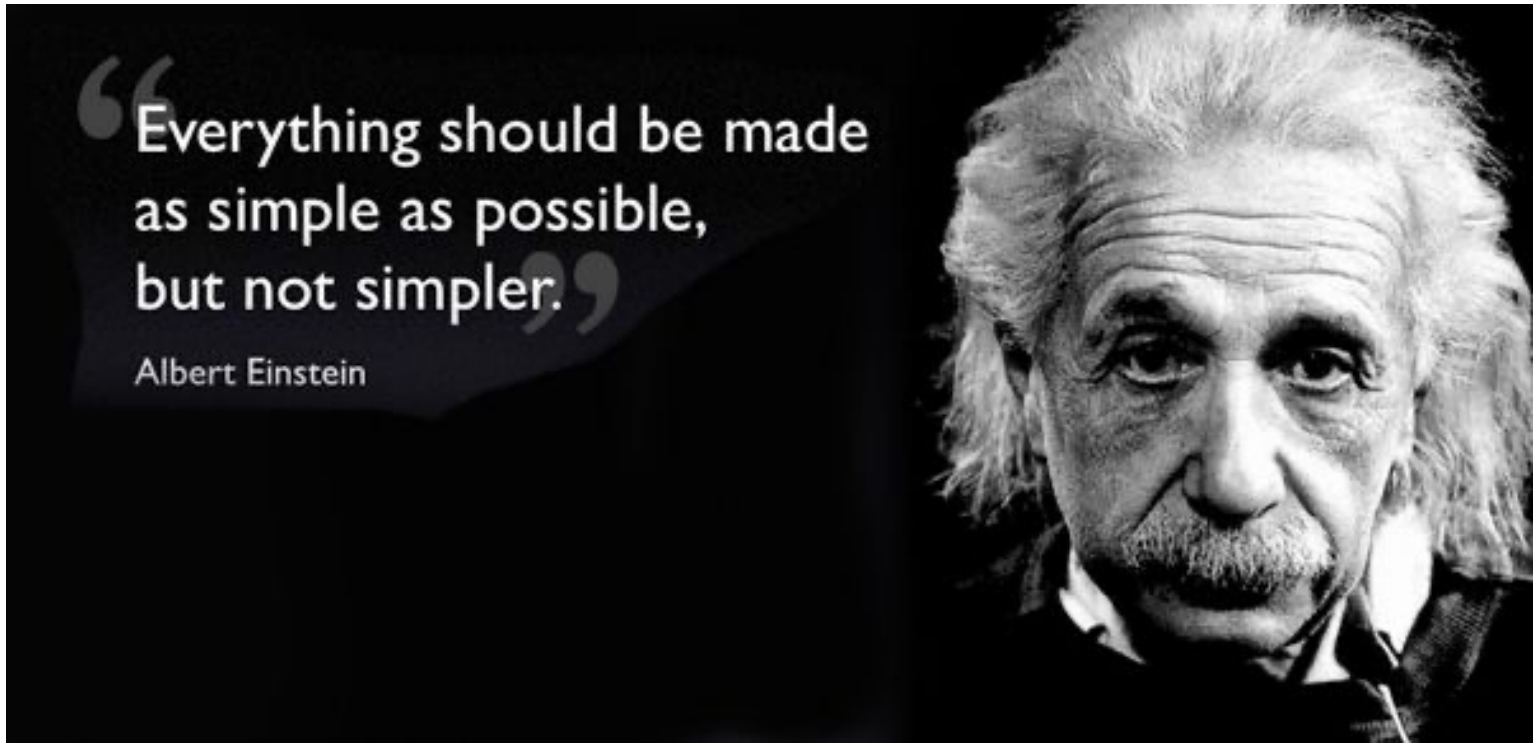
Managing priority

- User driven - static
- Dynamic management – priority inversion

Priority Inversion



Choose your approach wisely!



Memory managment

- Static – allocate memory at compile time to be sure of memory usages
- Dynamic

  Optimal usage at the risk of runtime errors
- Heaps:
- heap_1 - the very simplest, does not permit memory to be freed.
- heap_2 - permits memory to be freed, but does not coalescence adjacent free blocks.
- heap_3 - simply wraps the standard malloc() and free() for thread safety.
- heap_4 - coalescences adjacent free blocks to avoid fragmentation. Includes absolute address placement option.
- heap_5 - as per heap_4, with the ability to span the heap across multiple non-adjacent memory areas.

That’s all great but where are my devices? [zephyr, linux, etal]

- Board Support Packages (BSPs)

  Device Drivers (UART, GPIO, Timers, Storage, Comm – ETH, Wifi, BT)

  Interrupt Handlers, Memory Management, Configuration for peripherals,

  Debug Support, Power Management

- Device Trees (also a part of the BSP)

Things to check

- How does the chip address pins/functionalities? [device driver]

  e.g., PORTB

- How are things placed on the board? [device trees]

  e.g., in case of LED, is it active high or low?

esp32_datasheet_en.pdf (espressif.com)

esp32_technical_reference_manual_en.pdf (espressif.com)

esp32-wroom-32_datasheet_en.pdf (espressif.com)

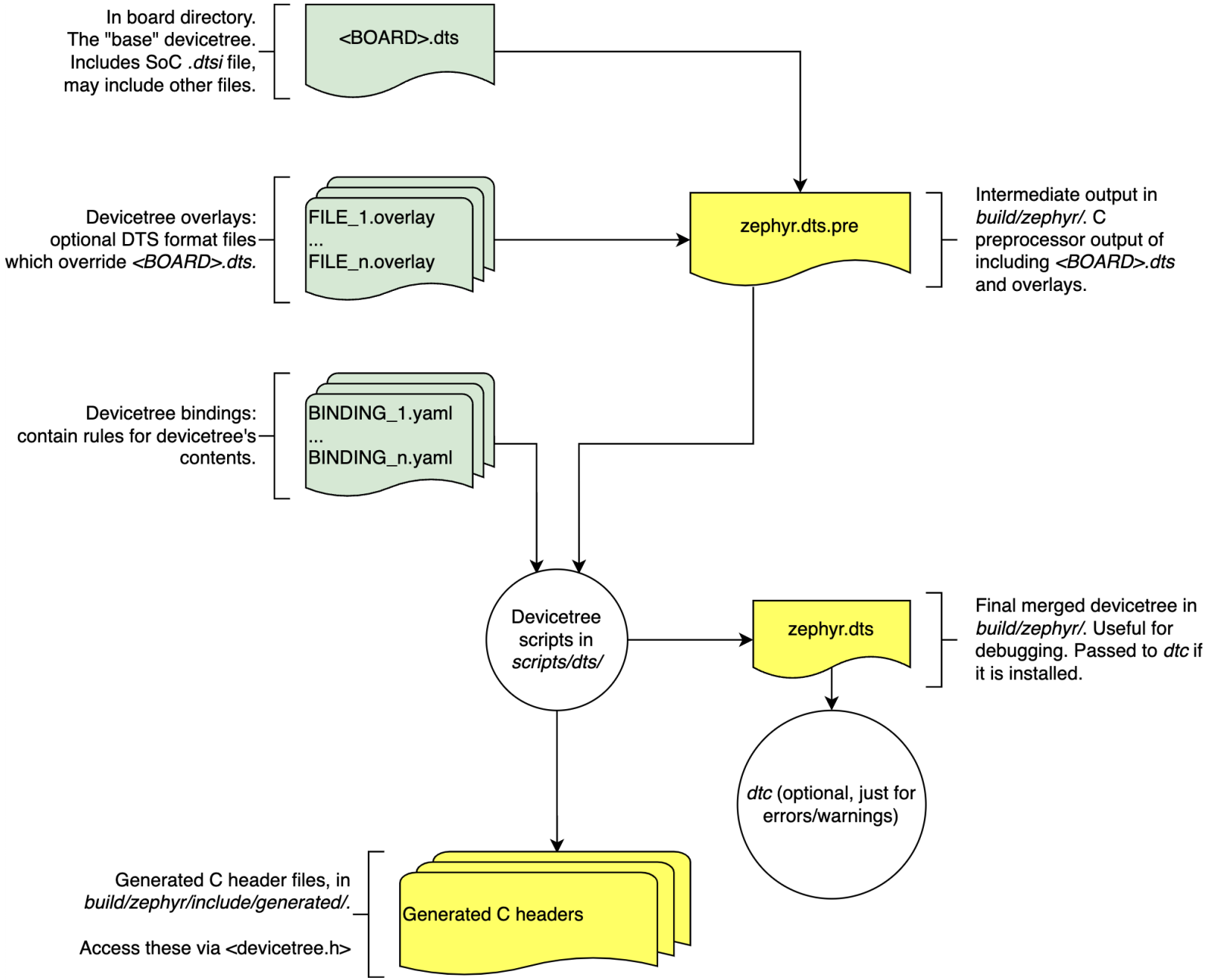
uPesy ESP32 Wroom DevKit

From application code to board specific binaries

- Drivers (code walkthrough)
  `zephyrproject/zephyr/drivers/gpio/gpio_esp32.c`
- Device trees, includes and overlays
  `zephyrproject/zephyr/dts/xtensa/espressif/esp32/esp32_common.dtsi`
  `zephyrproject/zephyr/dts/xtensa/espressif/esp32/esp32_wroom_32ue_n16.dtsi`

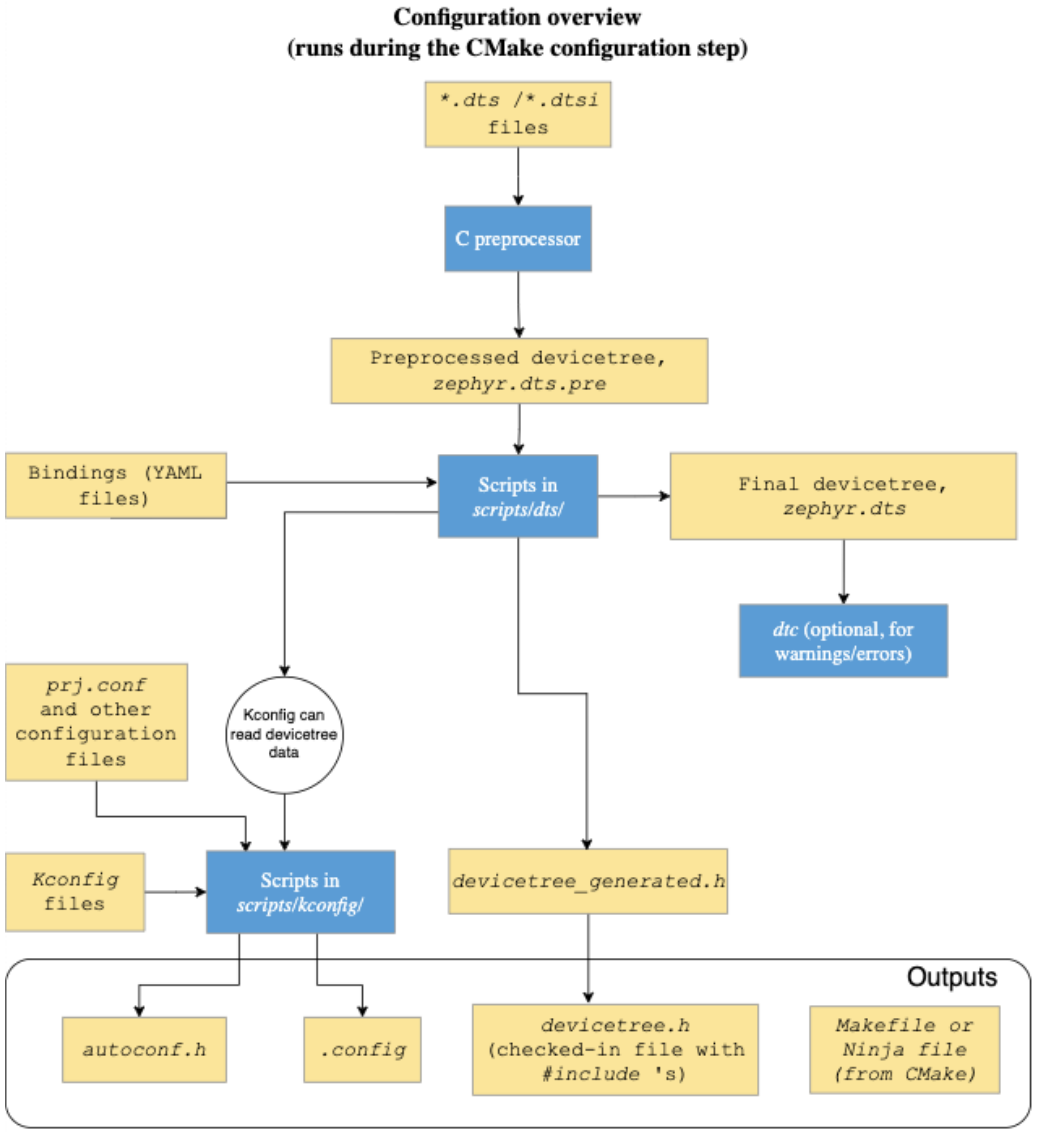
  `For your project/PCB: esp32.overlay`
- Device tree bindings to board (code walkthrough)
  `zephyr/dts/bindings/led/gpio-leds.yaml`

  `zephyrproject/zephyr/dts/bindings/display/solomon,ssd1306fb-common.yaml`



[Input and output files – Zephyr Project Documentation]

Build System



Access devices from application code (C)

```
/dts-v1/;

/ {

        aliases {
                sensor-controller = &i2c1;
        };

        soc {
                i2c1: i2c@40002000 {
                        compatible = "vnd,soc-i2c";
                        label = "I2C_1";
                        reg = <0x40002000 0x1000>;
                        status = "okay";
                        clock-frequency = < 100000 >;
                };
        };
};
```

Here are a few ways to get node identifiers for the `i2c@40002000` node:

- `DT_PATH(soc, i2c_40002000)`
- `DT_NODELABEL(i2c1)`
- `DT_ALIAS(sensor_controller)`
- `DT_INST(x, vnd_soc_i2c)` for some unknown number `x`. See the `DT_INST()` documentation for details.

Devicetree access from C/C++ – Zephyr Project Documentation

More samples

- Samples and Demos – Zephyr Project Documentation

Beyond RTOS (GHz processors)

Bare-metal programming

- Little or no software overhead
- Low power requirement
- High control of hardware
- Single-purpose or simple applications, hardware-dependent
- Strict timing (e.g. motor control)

Real-time Operating System (RTOS)

- Scheduler overhead
- More powerful microcontroller required
- High control of hardware
- Multithreading, some common libraries
- Multiple tasks: networking, user interface, etc.

Embedded General Purpose Operating System (GPOS)

- Large overhead (scheduler, memory management, background tasks, etc.)
- Microprocessor usually required (and often external RAM+NVM)
- Low direct control of hardware (files or abstraction layers)
- Multiple threads and processes, many common libraries (portable application code)
- Multiple complex tasks: networking, filesystem, graphical interface, etc.

Buildroot
Making Embedded Linux Easy

- Easy to use
- Limited configuration settings
- Requires full image rebuild

OpenWrt
WIRELESS FREEDOM

- Focused on networking gear (e.g. routers)
- Package-based updates

yocto
PROJECT

- Steep learning curve
- Customizable
- Active community

Assume a SD card bootloader to load the operating system from

How to use it?

- Non-trivial PCB design

  Helpful compute modules available

Specification

Form factor:	55 mm × 40 mm
Processor:	Broadcom BCM2711 quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
Memory:	1GB, 2GB, 4GB or 8GB LPDDR4 (depending on variant)
Connectivity:	• Optional wireless LAN, 2.4GHz and 5.0GHz IEEE 802.11b/g/n/ac wireless, Bluetooth 5.0, BLE with onboard and external antenna options • Onboard Gigabit Ethernet PHY supporting IEEE1588 • 1 × USB 2.0 interface • PCIe Gen 2 x1 interface • 28 GPIO signals • SD card interface for SD card or external eMMC (for use only with Compute Module 4 variants without eMMC)
Video:	• Dual HDMI interface (up to 4Kp60 supported) • 2-lane MIPI DSI display interface • 2-lane MIPI CSI camera interface • 4-lane MIPI DSI display interface • 4-lane MIPI CSI camera interface
Multimedia:	H.265 (4Kp60 decode); H.264 (1080p60 decode, 1080p30 encode); OpenGL ES 3.0 graphics
Input power:	5V DC
Operating temperature:	-20°C to +85°C
Production lifetime:	Raspberry Pi Compute Module 4 will remain in production until at least January 2028
Compliance:	For a full list of local and regional product approvals, please visit pip.raspberrypi.com

Or use them together

- Often vendors provide high speed microprocessors and microcontrollers on the same chip/board

  Speed and efficiency combo

  Controllers run RTOS while processors run OS

  ex1, flight stabilization on a flight controller runs on an RTOS while the AI tasks for autonomy run on a general purpose OS

  ex2, in-car systems -> ECUs run RTOS while infotainment systems might run general purpose OS

Product Examples

- Products Running Zephyr - Zephyr Project

Product	CPU	Linux Version
Amazon Echo	Intel Atom x5-Z8350	Amazon Linux
Samsung Smart TV	ARM Cortex-A9	Tizen
Sony PlayStation 4	AMD Jaguar	FreeBSD

Product	CPU	Linux Version
Roku Streaming Stick	ARM Cortex-A53	Roku OS
TiVo Bolt	Broadcom BCM7445	TiVo OS
Western Digital My Cloud	Marvell Armada 370	BusyBox
Nest Learning Thermostat	ARM Cortex-A9	Nest OS
Tesla Model S	NVIDIA Tegra 3	Tesla OS
LG Smart Refrigerator	ARM Cortex-A7	WebOS
Google Chromecast	Marvell Armada 1500 Mini	Google Cast OS
Asus Tinker Board	Rockchip RK3288	TinkerOS
Nintendo Switch	NVIDIA Tegra X1	Nintendo Switch OS
Philips Smart TV	ARM Cortex-A9	Android TV
Sonos One	ARM Cortex-A53	Sonos OS
Fitbit Versa	ARM Cortex-M4	Fitbit OS
HP LaserJet Enterprise	ARM Cortex-A9	HP FutureSmart
Garmin Forerunner 945	ARM Cortex-M4	Garmin OS
DJI Phantom 4 Pro	Intel Atom	A customized version of Linux
Bose SoundTouch 30	ARM Cortex-A9	Bose OS
Canon EOS R5	DIGIC X	Canon OS

Open Source Projects

- Open Source Autopilot for Drones - PX4 Autopilot
- System Overview – OpenEnergyMonitor 0.0.1 documentation

TinyML

Give magical abilities to your systems

Resources

- MIT 6.5940 Fall 2023 TinyML and Efficient Deep Learning Computing

  EfficientML.ai Lecture, Fall 2023, MIT 6.5940 – YouTube

- microsoft/EdgeML: This repository provides code for machine learning algorithms for edge devices developed at Microsoft Research India. (github.com)

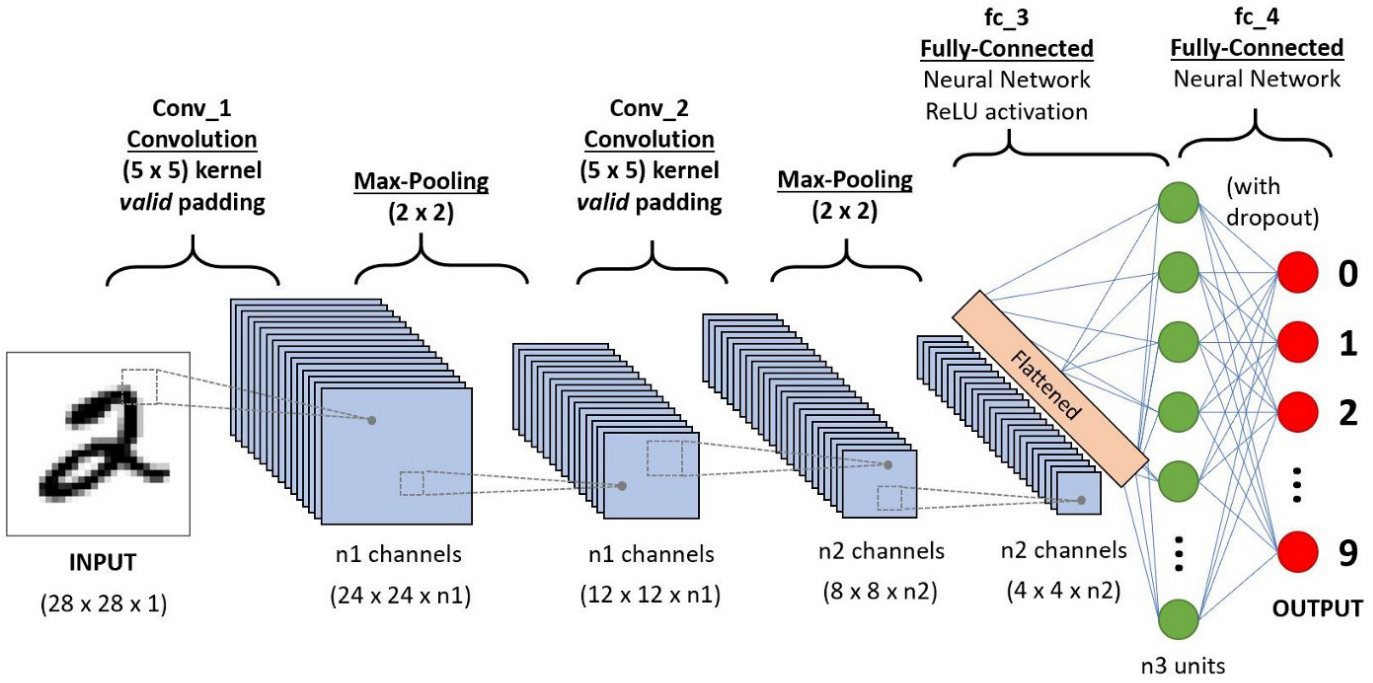
Topics

- Pruning
- Quantization, Distillation
- Wake word, Visual wake word

Smaller devices:

- Bonsai
- ProtoNN
- Shallow RNNs

Basics of CNN

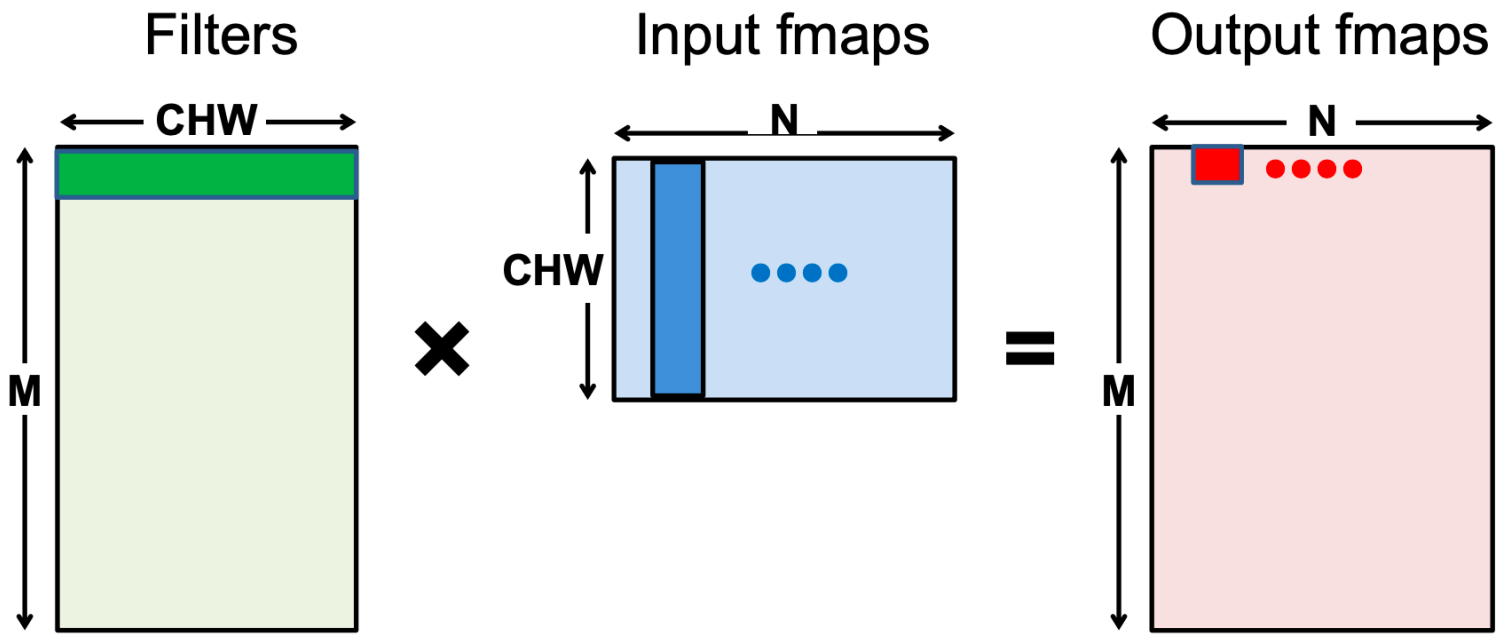


What is being processed?

- Let's take a deeper look at DNN computation
- Wide variety of devices support DNN computations today

Tutorial on Hardware Accelerators for Deep Neural Networks (mit.edu)

Matrices everywhere



- After flattening, having a batch size of N turns the matrix-vector operation into a matrix-matrix multiply

Conv Layer Computation

Naïve 7-layer for-loop implementation:

```
for (n=0; n<N; n++) {
    for (m=0; m<M; m++) {
        for (x=0; x<F; x++) {
            for (y=0; y<E; y++) {
                O[n][m][x][y] = B[m];
                for (i=0; i<R; i++) {
                    for (j=0; j<S; j++) {
                        for (k=0; k<C; k++) {
                            O[n][m][x][y] += I[n][k][Ux+i][Uy+j] × W[m][k][i][j];
                        }
                    }
                }

                O[n][m][x][y] = Activation(O[n][m][x][y]);
            }
        }
    }
}
```

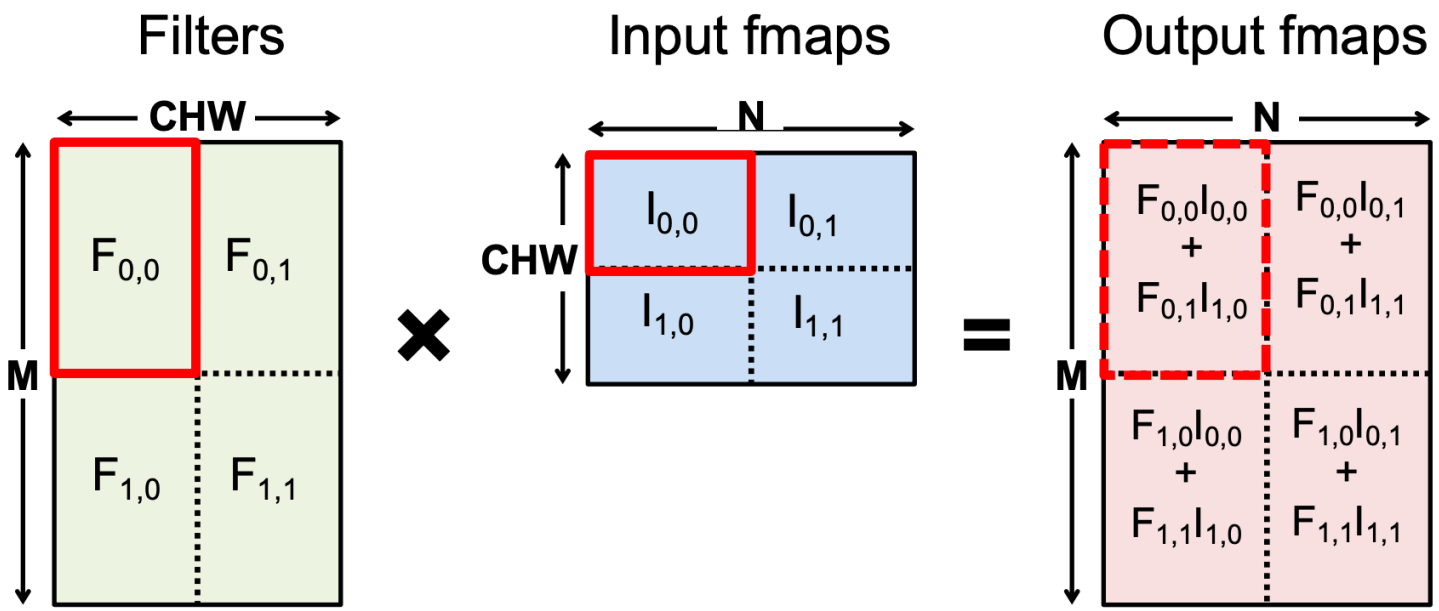
for each output fmap value

convolve a window and apply activation

Key ways to save energy

- Minimizing **data movement** is the key to achieving high **energy efficiency** for DNN accelerators
- Dataflow taxonomy:
  - **Output Stationary**: minimize movement of psums
  - **Weight Stationary**: minimize movement of weights
  - **Input Stationary**: minimize movement of inputs

Tiled implementation

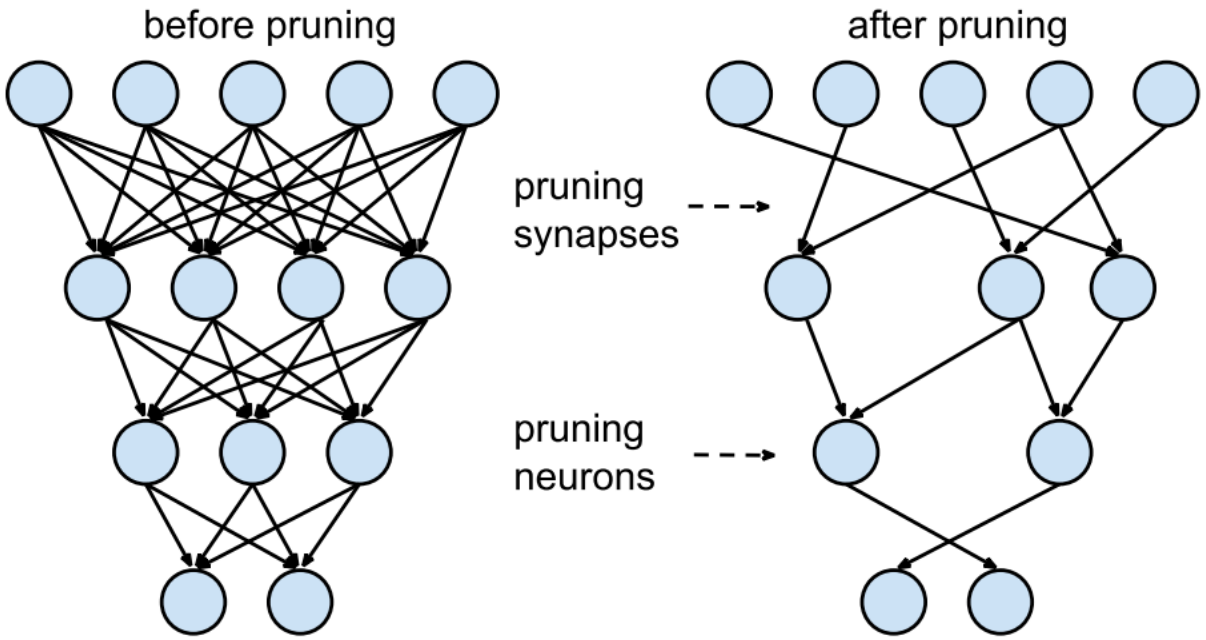


Matrix multiply tiled to fit in cache
and computation ordered to maximize reuse of data in cache

Quantization

24 bits per pixel

Pruning


before pruning
after pruning
pruning
synapses
pruning
neurons

Perception with CPUs

- General purpose processing – building for desktop/mobile/brov

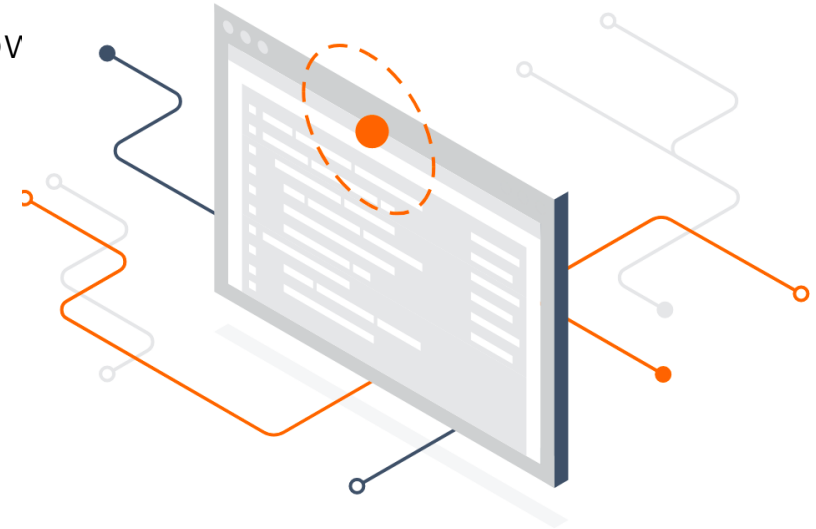
  Javascript based frameworks

- Software libraries for optimization

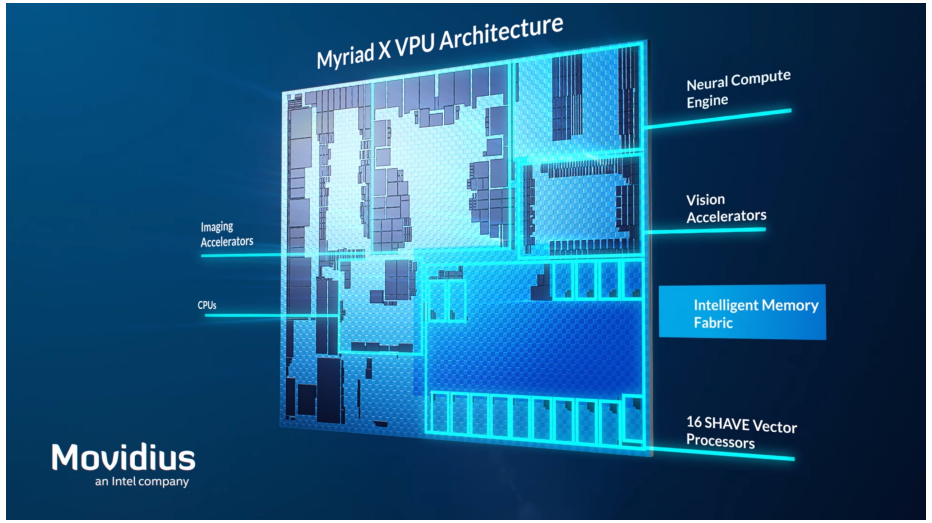
  OpenBLAS

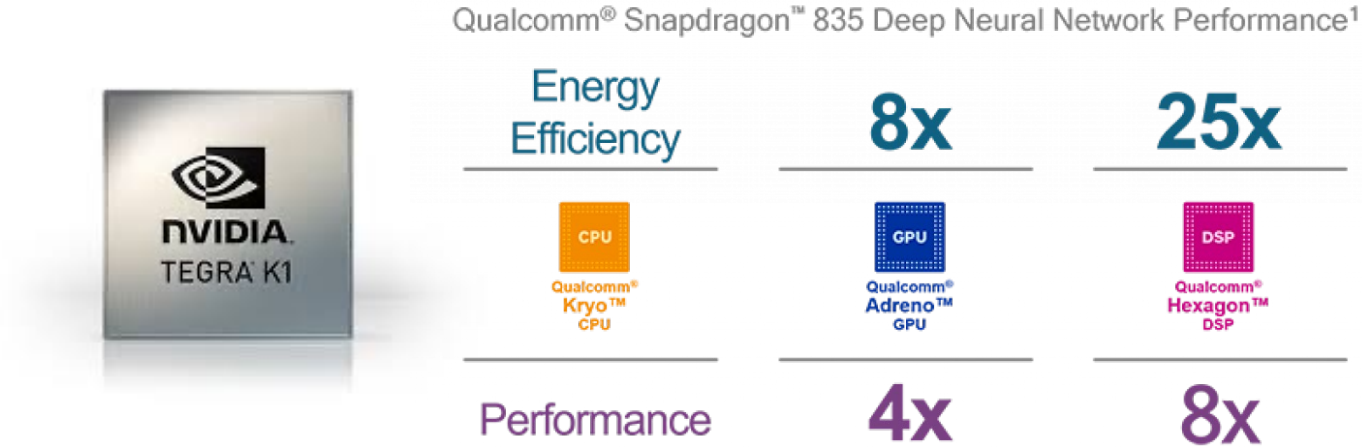
  Intel oneDNN (MKL)

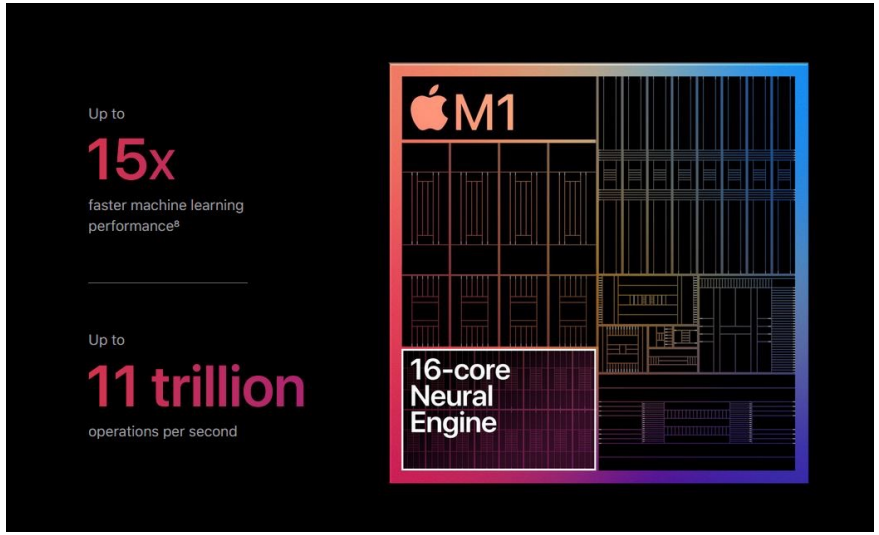
  LAPACK



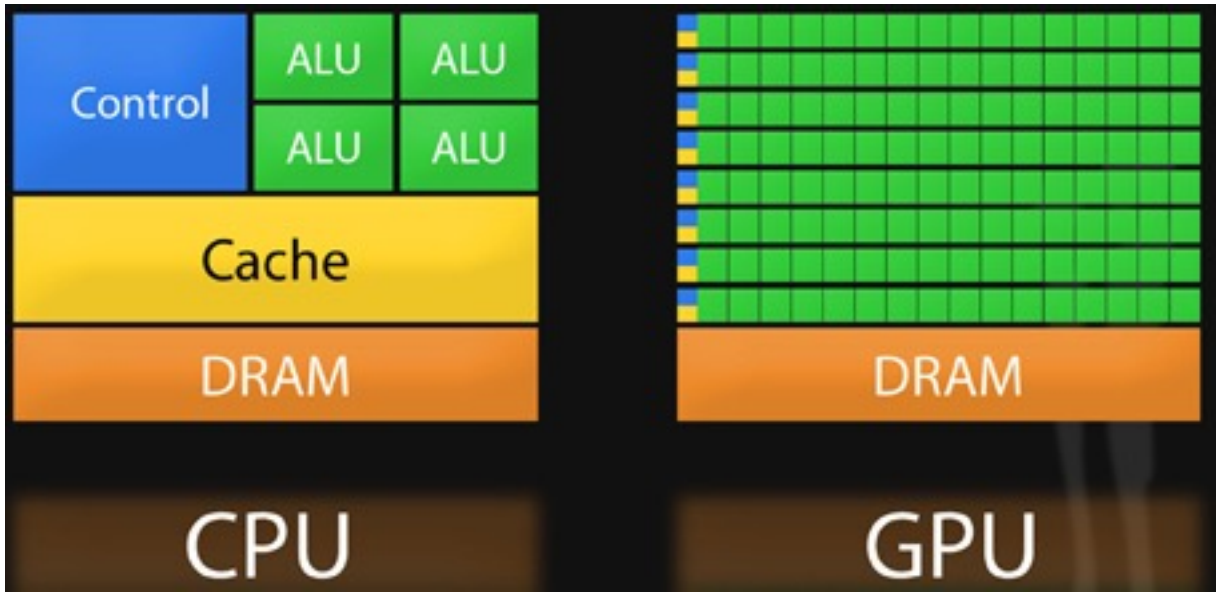
Perception: CPU + Accelerator



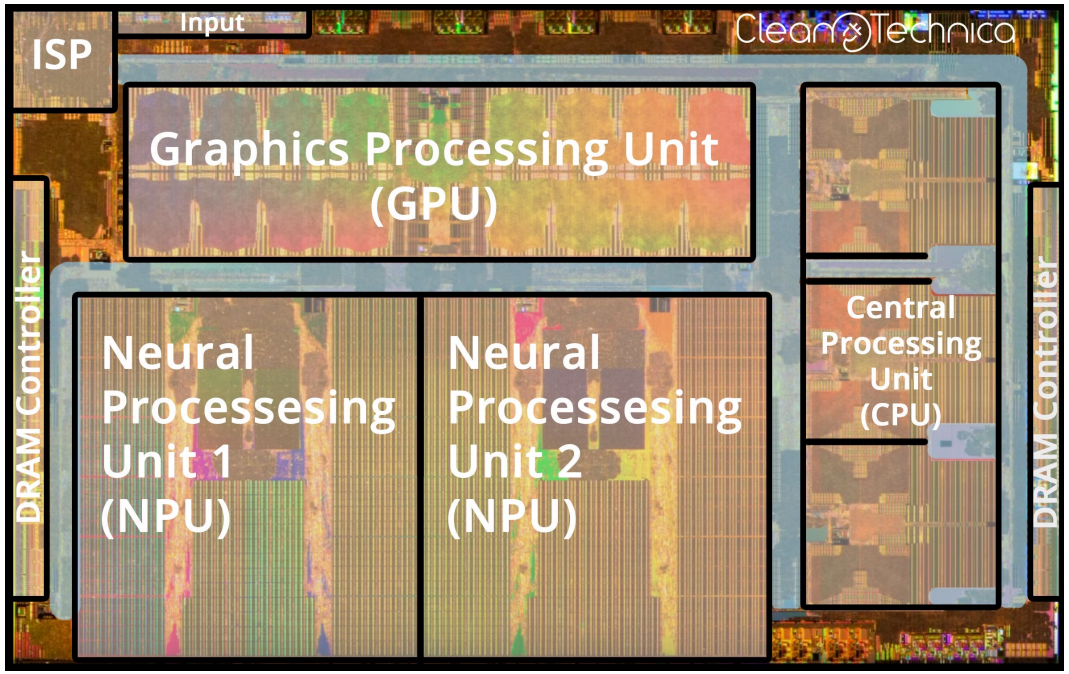




Perception with GPUs

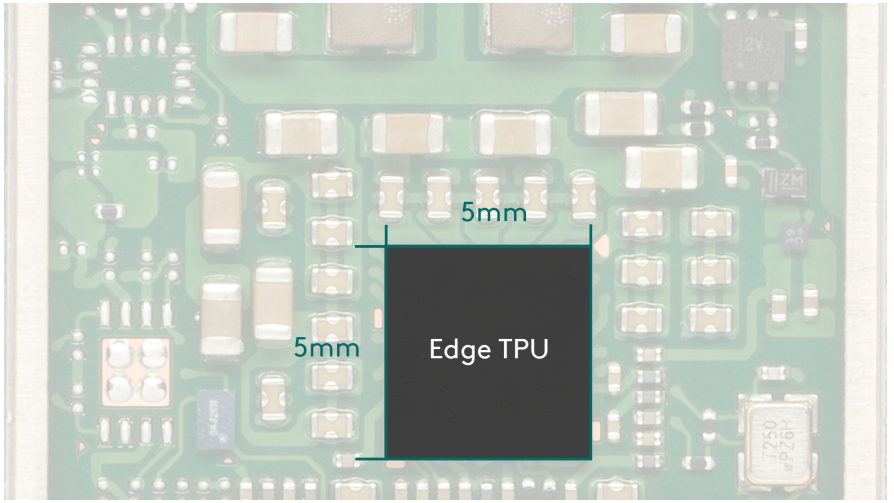


Dedicated Inference Chips

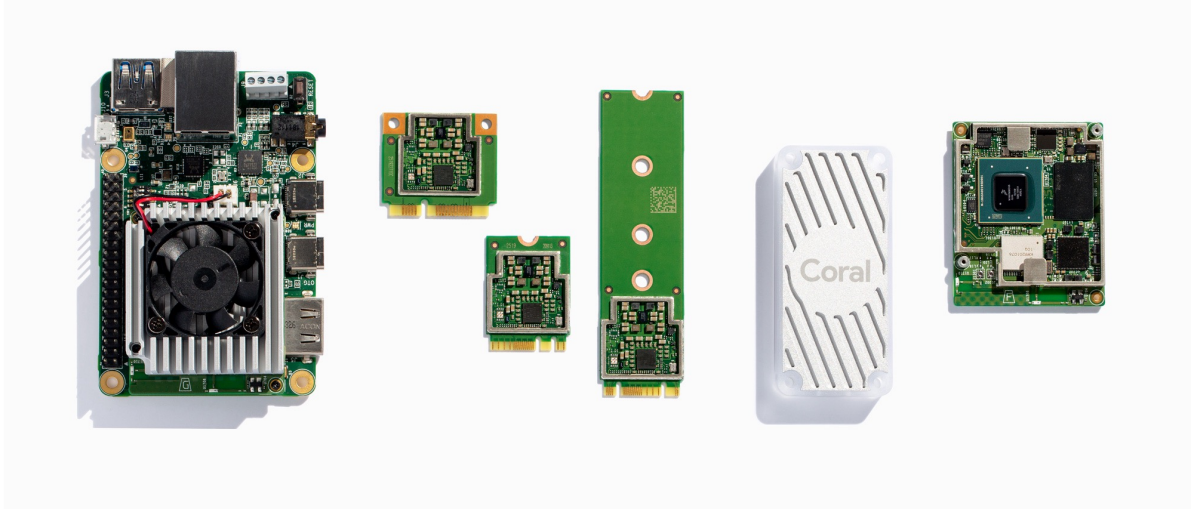


Tesla FSD Chip

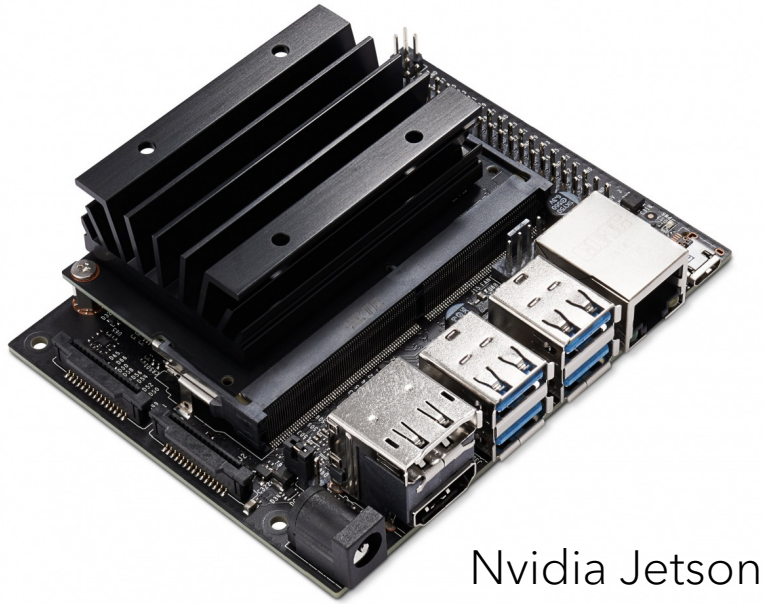
PULP



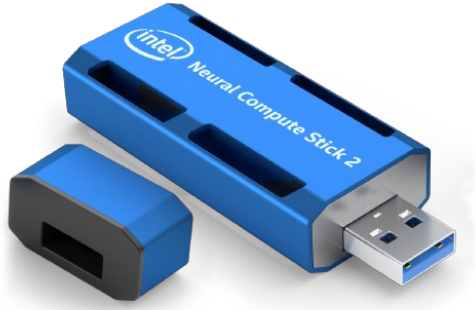
Boards/USB Accelerators



Google Coral

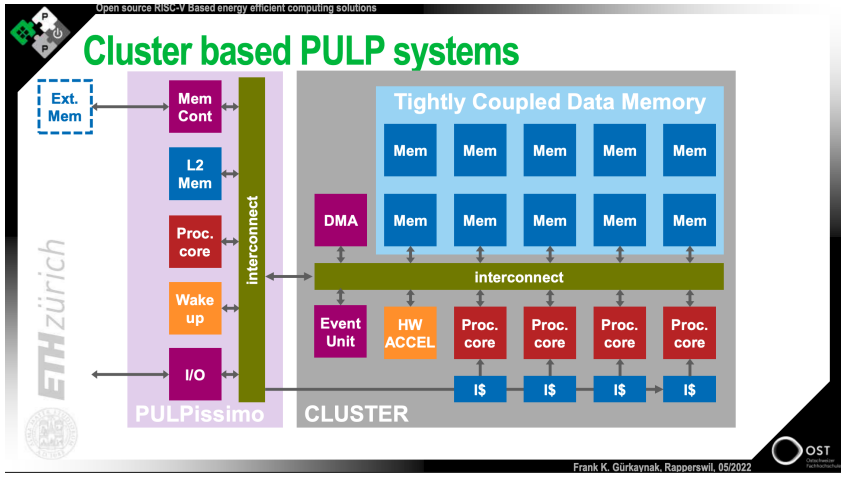


Nvidia Jetson



Intel Movidius

PULP Platform: Parallel Ultra-Low Power



27g, sub-10 W drone capable of autonomous navigation

pulp-platform/pulp-dronet: A deep learning-powered visual navigation engine to enables autonomous navigation of pocket-size quadrotor - running on PULP (github.com)

kgf_rapperswil_2022.pdf (pulp-platform.org)

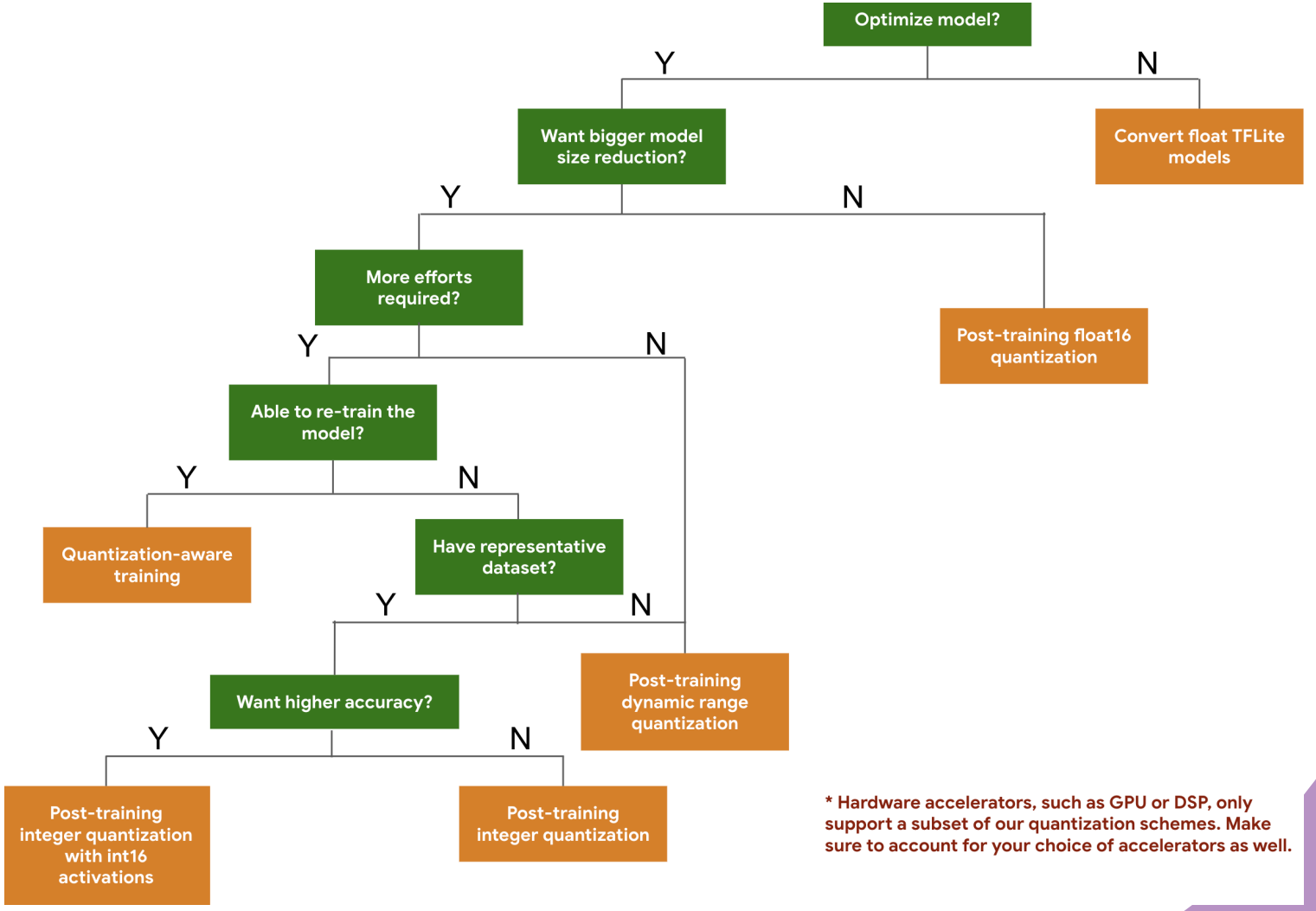
Pruning

- [Pruning – Part 1] (fully)
- [Pruning – Part 2] (Pg 10-35)

Quantization

- Quantization (pg 23-52)

Quantization



Model optimization | TensorFlow Lite

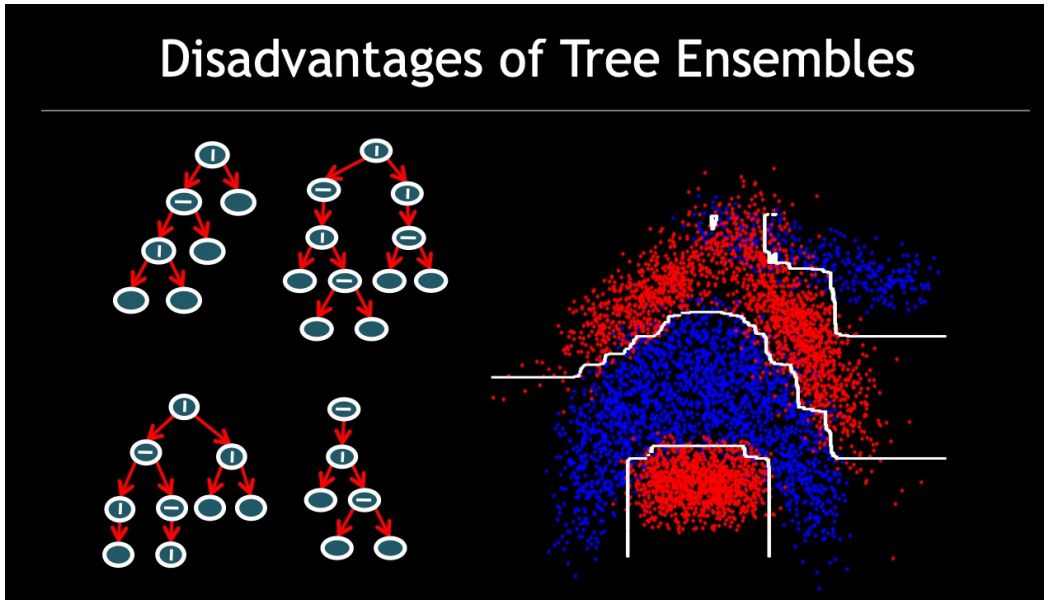
Distillation

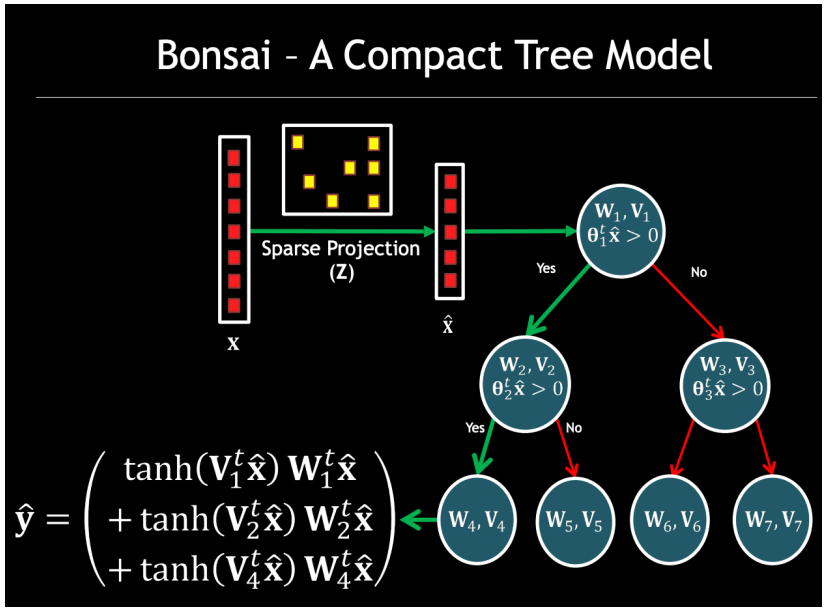
- Distillation (pg 1-36)

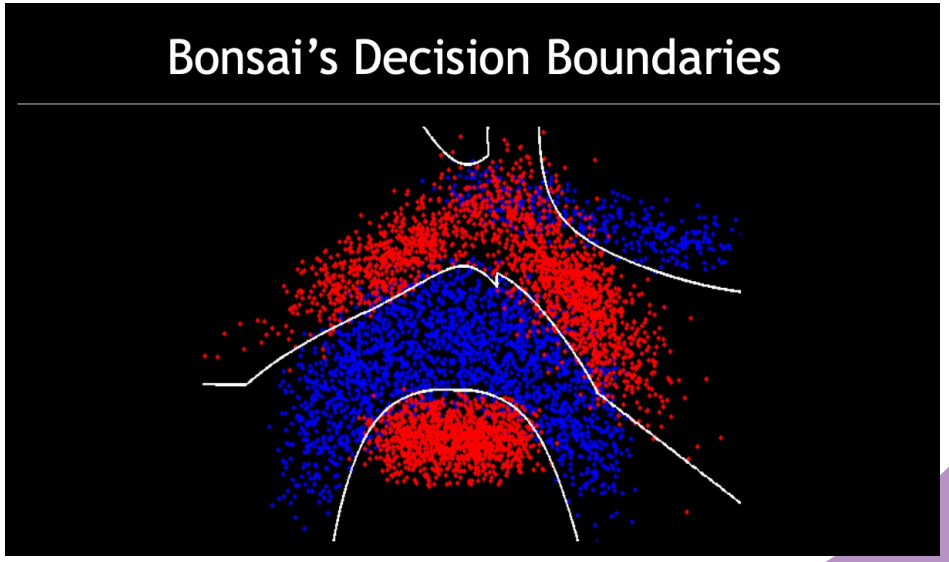
MCUNet

- [Slides]

Bonsai

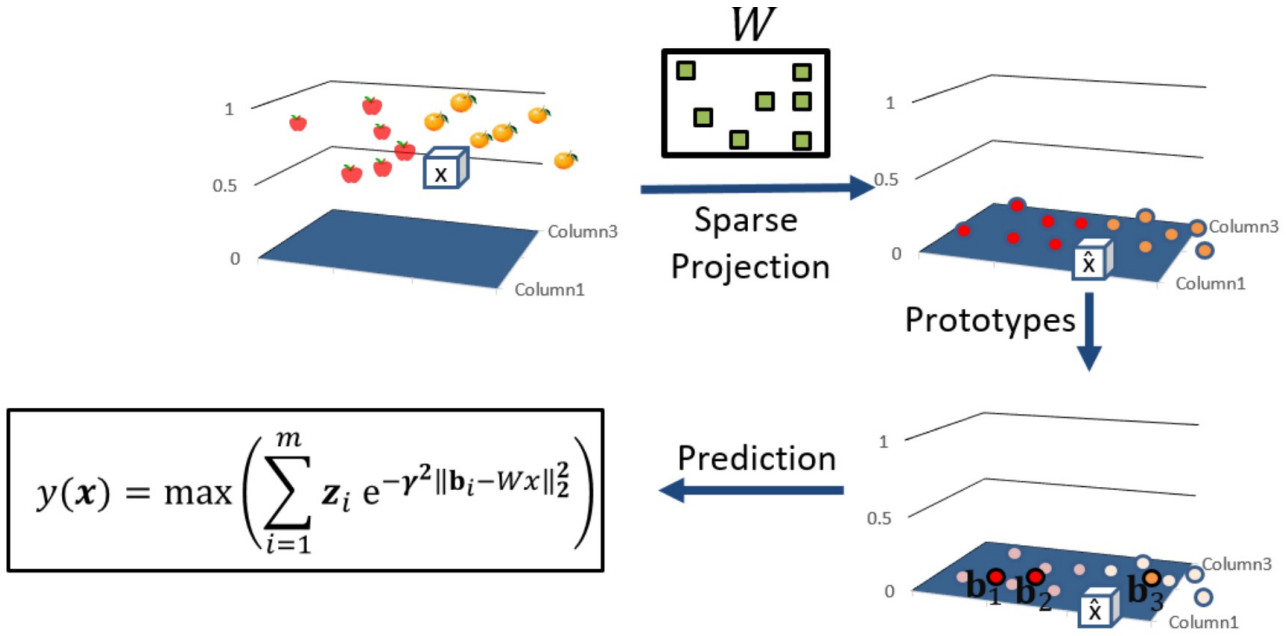

Disadvantages of Tree Ensembles


Bonsai - A Compact Tree Model
Sparse Projection
(Z)
x
x̂
W1, V1
θ1t x̂ > 0
Yes
No
W2, V2
θ2t x̂ > 0
W3, V3
θ3t x̂ > 0
Yes
No
W4, V4
W5, V5
W6, V6
W7, V7
ŷ = (tanh(V1t x̂) W1t x̂ + tanh(V2t x̂) W2t x̂ + tanh(V4t x̂) W4t x̂)


Bonsai's Decision Boundaries

ProtoNN

- https://github.com/Microsoft/EdgeML/wiki/files/ProtoNNICMLSlides.pdf



$$y(\boldsymbol{x}) = \max\left(\sum_{i=1}^{m} \boldsymbol{z}_i \, \mathrm{e}^{-\gamma^2 \|\mathbf{b}_i - W x\|_2^2}\right)$$

GesturePod

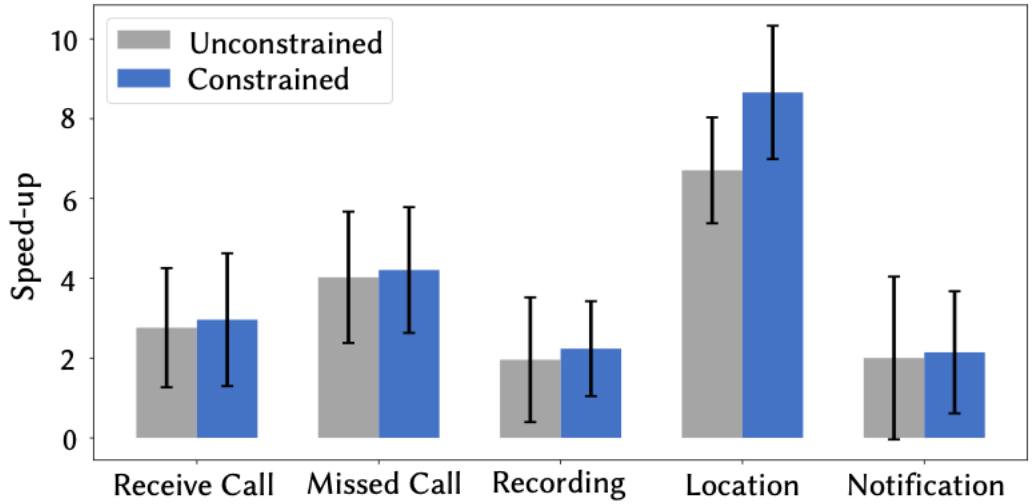
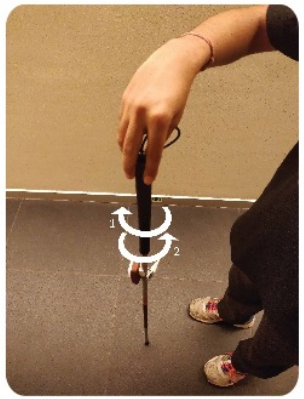


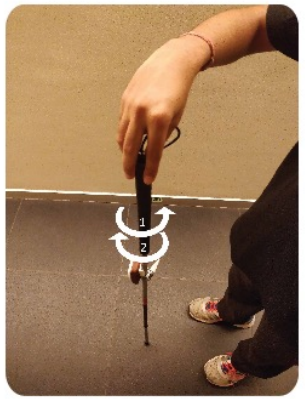
Figure 7: Speedup in task completion times using I-Cane over using smartphone alone.

(a) Constrained environment

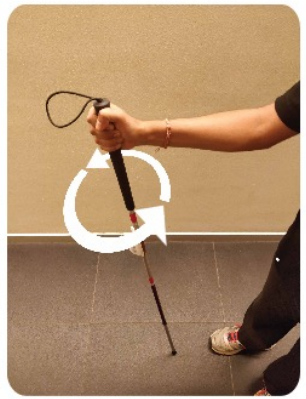
(b) Double Tap



(c) Right Twist



(d) Left Twist



(e) Twirl

(f) Double Swipe

EOC – End of Class

- Poll