RTOS Jump Start

Real Time Operating Systems
Focused on FreeRTOS
- Feb 2017

Matthew Eshleman
covemountainsoftware.com

Background - Matthew Eshleman

- 20+ years of embedded software engineering, software architecture, and project planning
    - MSEE Georgia Tech
    - Jumped into the RTOS world around 1998 - haven’t looked back
- Learn more: http://covemountainsoftware.com/consulting

Agenda

- What is an RTOS?
- What is available?
- Pros and Cons
- RTOS Fundamentals (examples using FreeRTOS running on the ESP32)
- Details on Demo, including 50,000 foot view of MQTT
- Demo!

What is a Real Time Operating System (RTOS) ?

- Framework of “Tasks” (threads).
  - Mechanisms enabling communication between threads
  - Mechanisms to protect data from standard thread data corruption issues
- Scheduler
  - Threads, with priorities
- Maybe “drivers”
- “Real Time” - critical point.

“Real Time”

- “Real Time” does not mean “fast”
- Real Time means deterministic
  - A real time operating system guarantees the worst case latency/delays regarding RTOS controlled behavior
  - Careful: your code can break that contract. Example: the code running with the RTOS disables interrupts for an extended time… delaying all RTOS controlled behavior

What is available?

- Linux - using “real time” extensions (maybe.. sort of?)
- µC/OS, ThreadX, VxWORKS, FreeRTOS, Integrity, QNX, Nucleus, RIOT, NuttX… and many more.
- Free and open source? Certified? Royalty Free? There is likely an RTOS available to meet your requirements.
- Our focus today is FreeRTOS

Decisions… decisions…

- Selecting an RTOS: Key Points to consider during decision making
  - Licensing
  - Size (RAM, etc)
    - Note, every OS object (semaphore, queues, etc) created will increase RAM usage. Some RTOS's balance their RAM usage differently. FreeRTOS uses "more RAM" per object than a few others, for example.
  - Interrupt handling
  - "Task Context Switch Time"
  - Cool algorithms: "Priority Inheritance", "Preemptive", "Cooperative"
  - Tools, environment (Debugging, Tracing, etc)

Using an RTOS: Pros / Cons

- Con
  - RAM/ROM Size and CPU overhead versus true bare metal while(1)
  - Multi-threaded software can be error prone and difficult to find/fix common thread bugs
    - End user report: “Every now and then it….”
      - Deadlock, priority inversion, thread corruption… oh my!
- Pros
  - Framework of common behaviors
  - Facilities to ensure interrupts/threads play nice with each other
  - Battery life and power usage
    - The RTOS may/can help power down portions of the system when IDLE

RTOS and FreeRTOS Fundamentals

Image: http://maxpixel.freegreatpicture.com/Books-Library-Knowledge-University-Education-1517100

Scheduler

- Makes the decision regarding which task is currently executing
  - Might have a “scheduling policy” setting to help make this decision. For example, Linux supports multiple scheduler policies, such as round-robin, FIFO, and more.
  - Nearly all RTOS options support a priority based preemptive scheduler, where the highest priority thread runs first or preempts a lower priority thread/task.
- Manages the “context switch” process:
  - Saving to RAM the previous thread’s register context and loading the CPU registers with the new thread’s context.
  - i.e. overhead. The price we pay for threads and an overall “OS” like environment.
- Key point: a thread may be executing code when a higher priority thread “preempts” the thread. Much like when any firmware code is stopped by a hardware interrupt service routine.
  - This is when typical thread data corruption issues may happen

Image: http://opensourceforu.com/2010/12/getting-started-with-rtlinux/

Threads / Tasks

- Independent code whose execution is managed by the RTOS scheduler within an independent “context” which includes a dedicated stack.

- Typically might be a function that looks like this:

```
void Task(void *)
{
    while (1)
    {
        vOSBlockCallHere();

        /* do stuff here */
    }
}
```

Play nice!

FreeRTOS - Create a Task (from FreeRTOS)

```
BaseType_t xTaskCreate(TaskFunction_t pvTaskCode,
                       const char * const pcName,
                       unsigned short usStackDepth,
                       void *pvParameters,
                       UBaseType_t uxPriority,
                       TaskHandle_t *pxCreatedTask
                      );
```

- **pvTaskCode - Pointer to the task entry function.**
    - Tasks are normally implemented as an infinite loop, and must never attempt to return or exit from their implementing function. Tasks can however delete themselves.
- pcName - A name for the task. Primarily for debugging.
- **usStackDepth - The number of words (not bytes!) to allocate for use as the task's stack.**
- pvParameters - A value that will passed into the created task as the task's parameter.
- **uxPriority - The priority at which the created task will execute.**
- pxCreatedTask - Used to pass a handle to the created task out of the xTaskCreate() function. pxCreatedTask is optional and can be set to NULL

FreeRTOS Task Usage Examples

- Simple polling loop (Demo code, see AccelReader.cpp):

```
void PollingTask(void *) {

   while (1) {
      vTaskDelay(33_milliseconds);

      /* do polling stuff here */
      }
   }
```

- Waiting on queued message (Demo code, see ColorMappedDataVisualizer.hpp):

```
void PendingOnQueuedMessageTask(void *) {

   while (1) {
      Msg receivedMsg;
      if (xQueueReceive(mQueue, &(receivedMsg), (TickType_t ) portMAX_DELAY)) {

         /* act upon queued message here */
      }
   }
}
```

Semaphore

- A semaphore is a RTOS object providing a thread safe OS controlled signaling mechanism.
- Typically used to signal to waiting threads that a resource is now available.
- Underlying concept is a count, often times referred to as “tokens”.
- Binary semaphore only supports ‘0’ or ‘1’ tokens (count)
- Might be used as a “lock,” but I recommend a mutex when available.
- Examples:
  - Represent how many free buffers remain in a pool
  - Signal when an ISR is complete

Image: https://publicdomainreview.org/collections/flag-semaphore-gif/

Semaphore - Create

```
SemaphoreHandle_t xSemaphoreCreateBinary( void );
```

```
SemaphoreHandle_t xSemaphoreCreateCounting( UBaseType_t uxMaxCount,
                                            UBaseType_t uxInitialCount);
```

Notice when creating the “Counting” semaphore, you can set the Max Count and the Initial Count.

Semaphore - Usage Example

```
void TaskWaitingOnISRSignal(void *) {

    while (1) {
        xSemaphoreTake(m_IsrSignalSema, portMAX_DELAY);

        /* Code reacting to the signal */
    }
}
```

Signals to the waiting task

```
void MyHardwareIsrFunc() {
    if(someIsrLogic) {
      xSemaphoreGiveFromISR(m_IsrSignalSema, &xHigherPriorityTaskWoken);
    }
}
```

FreeRTOS requires use of special APIs when called from an ISR

Mutex - Mutual Exclusion

- A RTOS object providing a true locking mechanism, often times with optional OS enabled algorithms to help prevent “priority inversion” induced failures.
- Lock —> <use resource> —> Unlock.
- Types: Non-recursive, recursive.
- Non-recursive: single lock. If the same thread tries to lock again, then it will hang.
- Recursive: The same thread may lock the same mutex multiple times without blocking. Must unlock the mutex the exact same number of times
  - When in doubt, I tend to use recursive.
  - Note: FreeRTOS - can’t be used in an ISR. Which is fine, should never try to lock anything in an ISR anyhow.

Mutex - Create

```
SemaphoreHandle_t xSemaphoreCreateMutex( void );

SemaphoreHandle_t xSemaphoreCreateRecursiveMutex( void );
```

Notice that FreeRTOS enables their mutex feature within their semaphore APIs.

Mutex - Lock/Unlock

Unlock:

```
BaseType_t xSemaphoreGive( SemaphoreHandle_t xSemaphore );

BaseType_t xSemaphoreGiveRecursive( SemaphoreHandle_t xMutex );
```

Note

Lock:

```
BaseType_t xSemaphoreTake( SemaphoreHandle_t xSemaphore,
                           TickType_t xTicksToWait );

BaseType_t xSemaphoreTakeRecursive( SemaphoreHandle_t xMutex,
                                    TickType_t xTicksToWait );
```

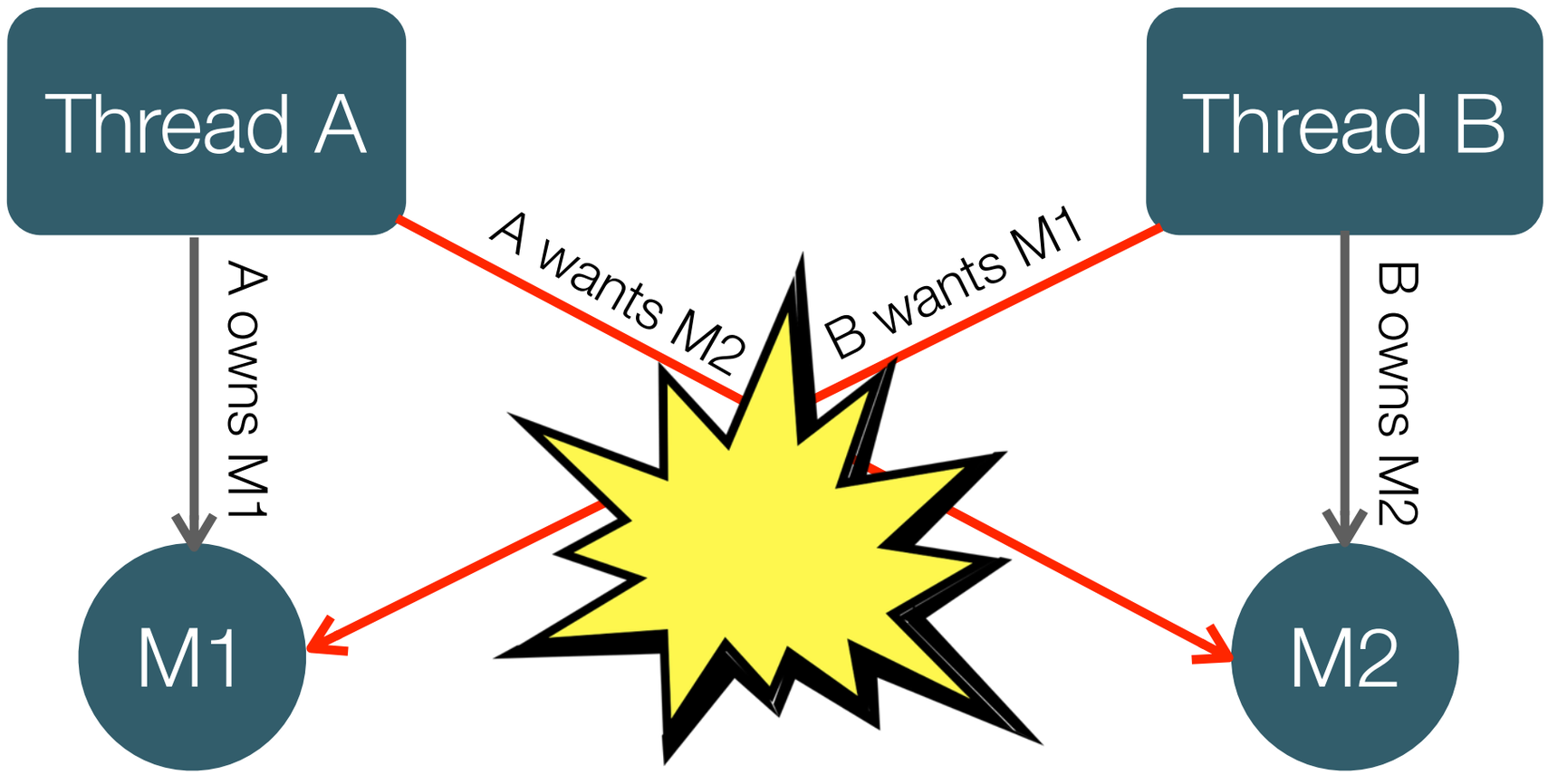
Note

Reviewing FreeRTOS's heavy use of macros, we actually find a Queue being used!

It would have been nice if FreeRTOS had used the same lock/unlock APIs for a normal vs a recursive mutex.

Deadlock

- When two or more threads compete for resources, usually in different orders, resulting in all threads waiting on a resource other threads already locked/own.



Priority Inversion, a rover, and the planet Mars

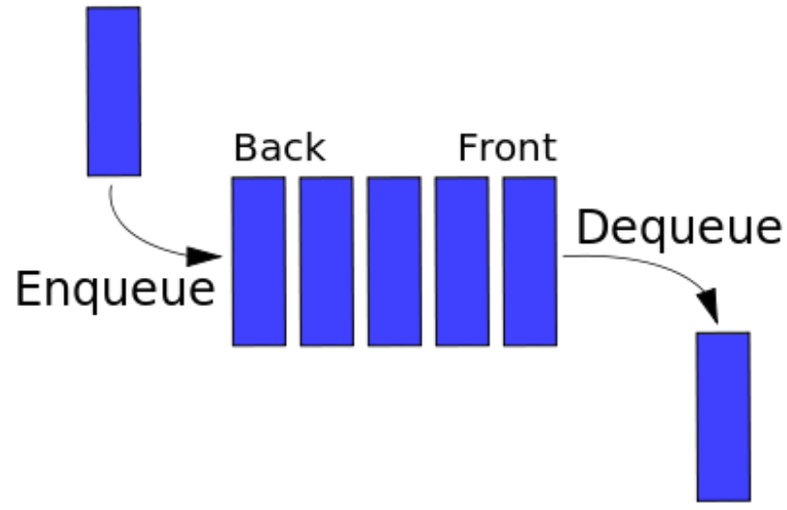
- Priority Inversion - Low priority thread A owns a resource that high priority thread C is waiting upon. Thread B, with priority higher than A but less than C, starts running, and now C is waiting on B and A!
    - Results in Thread C missing deadlines… or causing a system reset, as it did with the Mars Pathfinder.

To fix priority inversion… we add priority inheritance

- Priority Inheritance
  - If the RTOS detects priority inversion, the RTOS will temporarily raise the priority of the low priority thread to the same priority of the waiting thread.
  - This is how NASA/JPL fixed the Pathfinder reset issue: they sent a patch that updated the mutex in question to use the RTOS's priority inheritance algorithm
- From FreeRTOS:
  - "Priority inheritance does not cure priority inversion! It just minimizes its effect in some situations. Hard real time applications should be designed such that priority inversion does not happen in the first place."
  - FreeRTOS implements priority inheritance by default, for some RTOS's it is optional or a parameter that must be set when creating the mutex (as was the case with the Mar's Pathfinder).

Queues

- Nearly all RTOS vendors provide a ‘queue’, which is a thread safe mechanism to send a message to waiting threads



- Typically “First In, First Out”

- FreeRTOS API to create:

```
QueueHandle_t xQueueCreate( UBaseType_t uxQueueLength,
                            UBaseType_t uxItemSize );
```

Image: https://en.wikipedia.org/wiki/Queue_(abstract_data_type)#/media/File:Data_Queue.svg

Queues continued…

- Pro
  - “Events” are not lost (unless queue is full)
  - Order of events is maintained by the queue
  - Send data with an event
- Cons
  - Extra memory usage compared to a simple semaphore signal
    - Usage: overhead + MessageSize*NumberOfMsgs
  - How deep should the queue be? (Non-trivial issue in safety oriented systems)
  - How to handle when the queue is full? Drop message? Retry? Assert?
- Demo code usage example: ColorMappedDataVisualizer.hpp, class internal queue usage supporting 5 different commands/message types.

Timers

- An RTOS object that generates an event at certain time intervals or perhaps at a certain absolute time
- FreeRTOS supports interval based timers
- Important Note: FreeRTOS implements timers in an internal task. Timers are controlled (internally) via a message queue to that task. Timer callbacks must be aware of this fact and behave accordingly.
    - i.e. Timer callbacks must not themselves block or they may block other timers
- Demo code: PatternGenerator class. A FreeRTOS timer drives the pattern generator output.
    - FreeRTOS APIs are self-explanatory: http://www.freertos.org/FreeRTOS-Software-Timer-API-Functions.html

Interrupts

- Last… but not least… RTOS and Interrupts
- Some RTOS systems are hands on with interrupts (more overhead with flexibility). Some, like FreeRTOS, are more light weight (more restrictions or special APIs to consider)
- Exact details depend upon the microcontroller
- When reviewing the FreeRTOS APIs, take careful note of which objects may be accessed from an ISR:
  - Examples:
    - xQueueSendFromISR(…)
    - xSemaphoreGiveFromISR(…)
    - xTimerStartFromISR(…)
  - All of above should be paired with a call to: portYIELD_FROM_ISR(…)
- As always with your Interrupt Service Routing code: **Get in. Get out… fast.**

The Demo

Image: https://www.adafruit.com/products/1461

The Demo

- Elements/Technology in the demo
  - ESP32 (3 boards, networked)
  - FreeRTOS
  - Wifi/MQTT
  - Accelerometer (1 unit)
  - LED Light Strips (All 3 units)

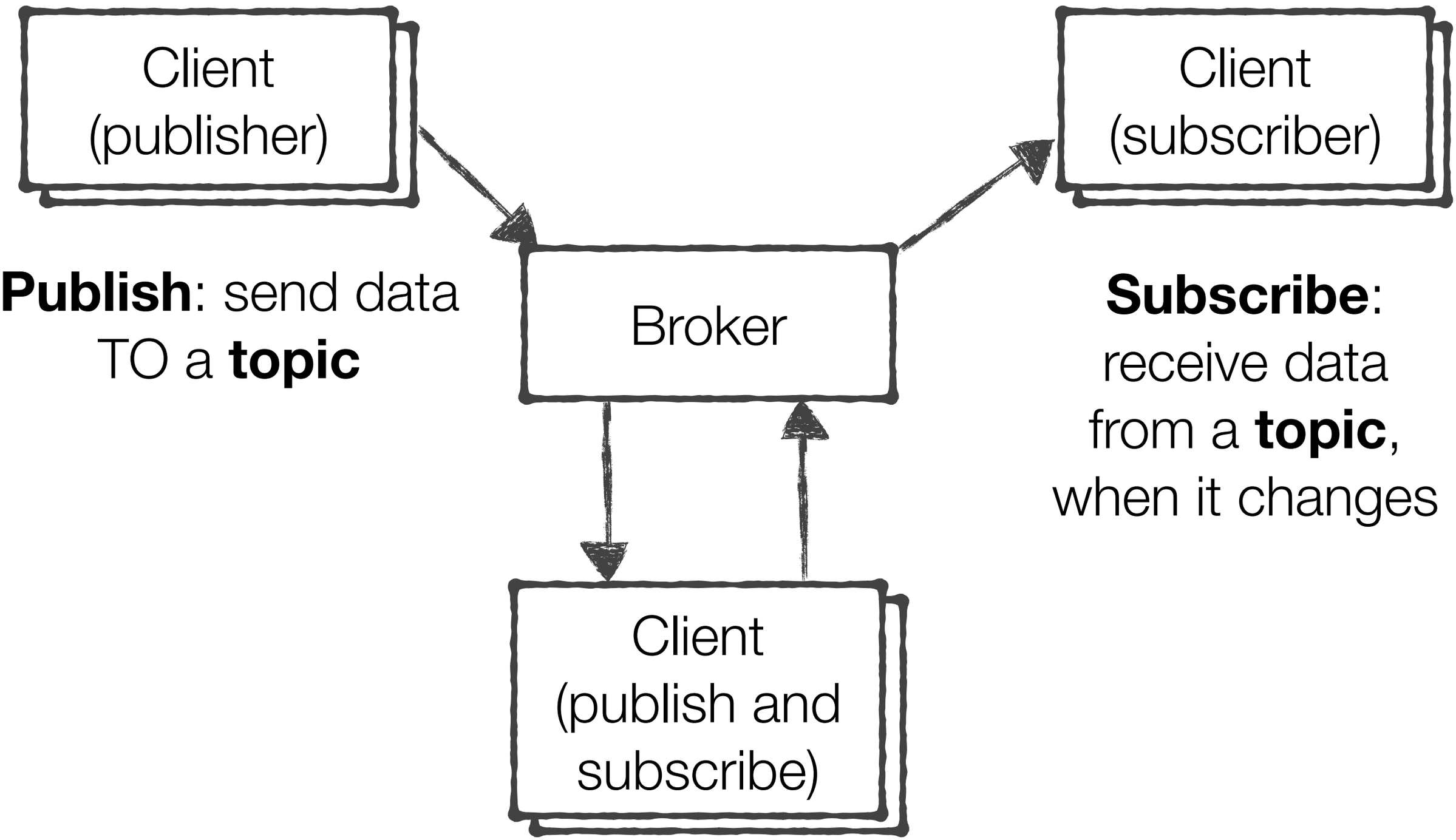
ESP32

- From Espressif, creator of the ESP8266
- Supports Wifi and Bluetooth
- Dual Core! (this is what caught my attention)
- Various internal peripherals, pin mapping, etc:
    - GPIO, UART, SPI, I2C, IR Tx/Rx (RMT), Flash, SD, ADC, DAC, PWM, more
    - random number generator, hardware encryption support… more!
- Relatively open “ESP-IDF” SDK development: https://github.com/espressif/esp-idf
    - Lots of ongoing effort and changes… still somewhat raw
    - First time I used twitter to solve an issue: https://twitter.com/onelittlebat/status/814609491622625281
- Today’s Demo is using I2C for accelerometer. RMT for Led Light Strips, GPIO for a crude CPU meter, and of course Wifi for MQTT

MQTT

- Created 1999
- What is MQTT? (from: http://mqtt.org/faq)
  - “MQTT stands for MQ Telemetry Transport. It is a publish/subscribe, extremely simple and lightweight messaging protocol, designed for constrained devices and low-bandwidth, high-latency or unreliable networks.” - Great for M2M and the IoT
- AWS, others also using MQTT for their IoT services

MQTT - Brokers, clients, topics

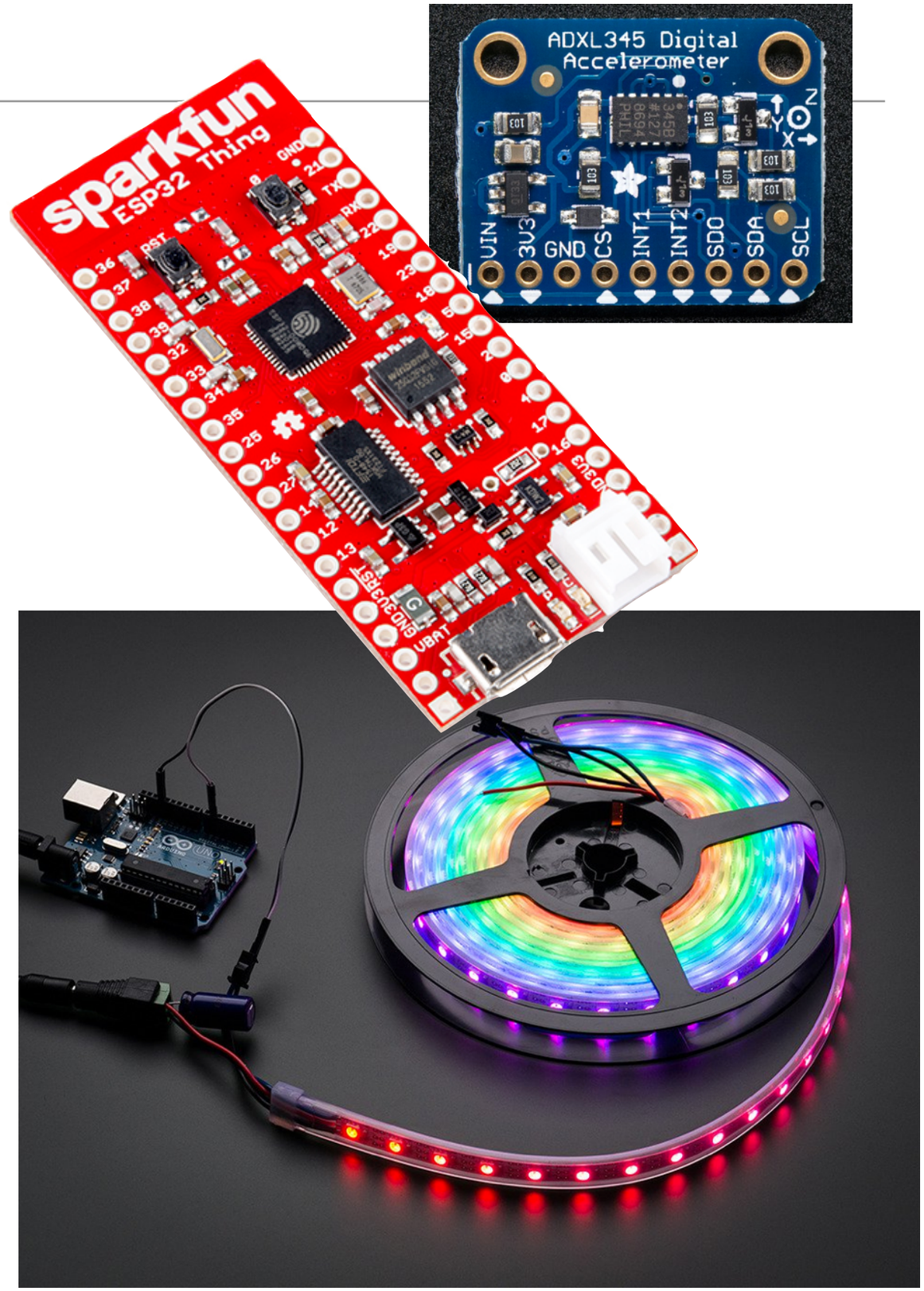


Publish: send data TO a **topic**

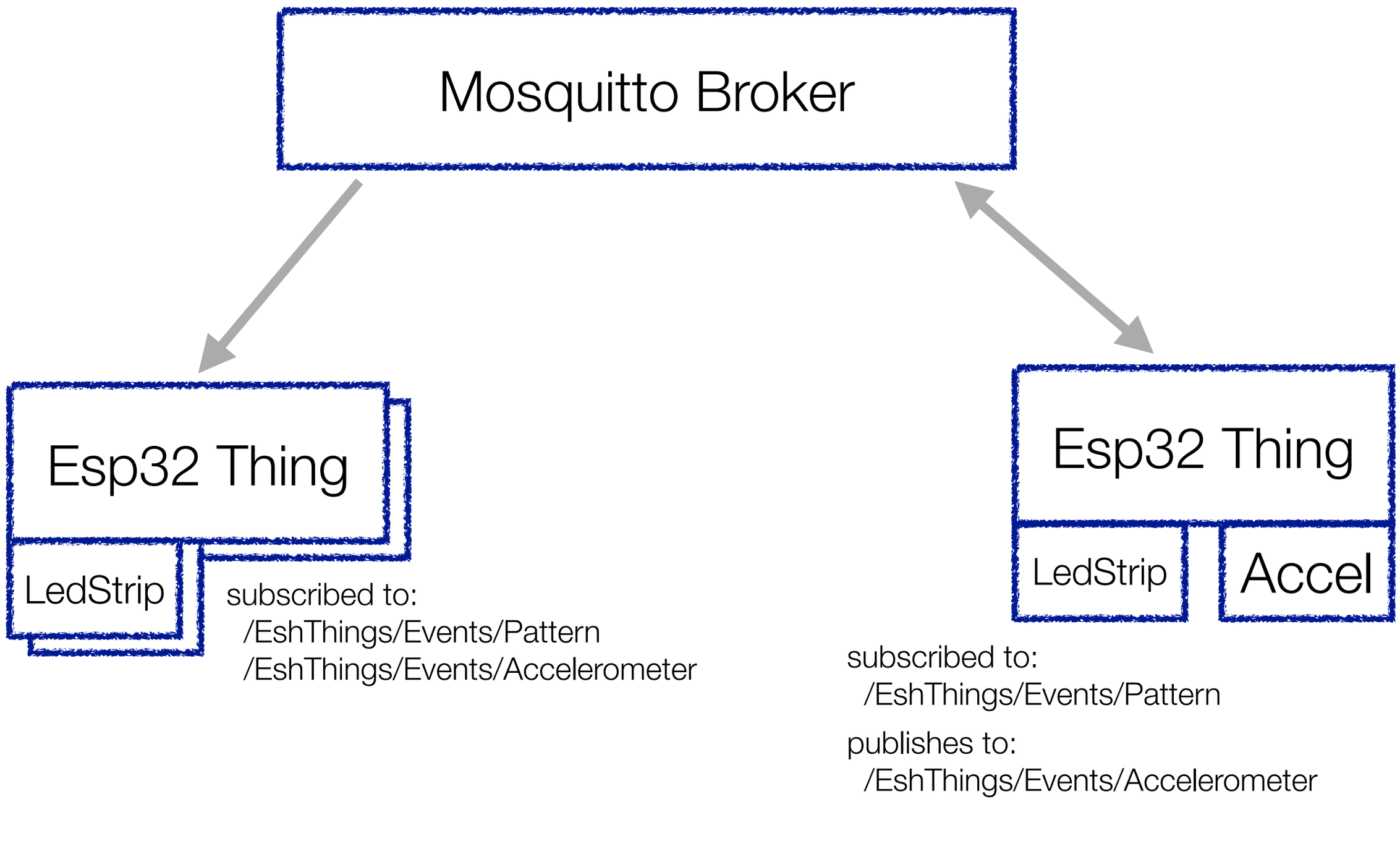
Subscribe: receive data from a **topic**, when it changes

Demo Setup

- ESP32 Thing
  - https://www.sparkfun.com/products/13907
- Adafruit ADXL345 Accelerometer
  - https://www.adafruit.com/products/1231
- Adafruit NeoPixel Digital RGB LED Strip
  - https://www.adafruit.com/products/1461
- Some cheap USB power converters, 2 Amps each
  - http://www.mpja.com/5VDC-2A-Plug-Supply-Dual-USB/productinfo/32736+PS
- Mosquitto MQTT Broker running on laptop
  - https://mosquitto.org/
- o-scope to monitor CPU usage via GPIO



Demo System Diagram



Demo…

Thank you!

Any questions?
matthew@covemountainsoftware.com

- Resources:
  - http://www.freertos.org/
  - http://esp32.com/
  - http://espressif.com/products/hardware/esp32/resources
  - https://www.sparkfun.com/products/13907
  - https://www.adafruit.com/products/1461
  - https://www.adafruit.com/products/1231
  - http://research.microsoft.com/en-us/um/people/mbj/Mars_Pathfinder/Mars_Pathfinder.html
  - http://research.microsoft.com/en-us/um/people/mbj/mars_pathfinder/Authoritative_Account.html
  - https://mosquitto.org/
  - https://github.com/espressif/esp-idf
  - https://github.com/Lucas-Bruder/ESP32_LED_STRIP
  - https://github.com/imxieyi/esp32-i2c-adxl345
  - https://github.com/tuanpmt/esp32-mqtt
  - http://www.andrewnoske.com/wiki/Code_-_heatmaps_and_color_gradients
  - https://covemountainsoftware.com/2016/12/27/brother-can-you-spare-a-gpio/