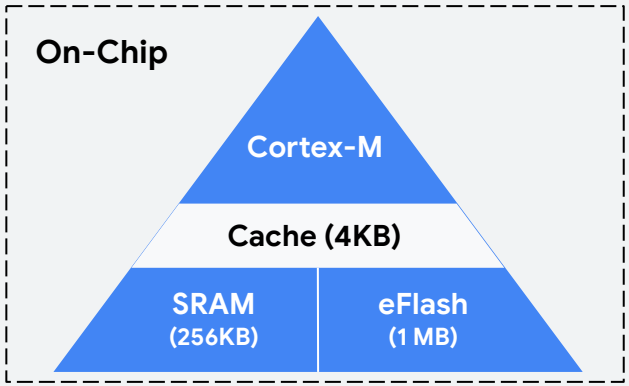
TFLite Micro: Memory Allocation

The Tensor Arena

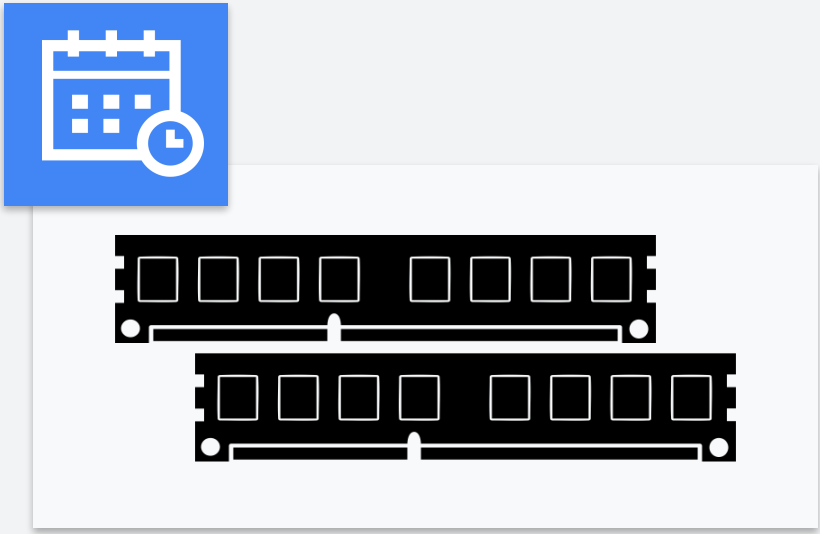
Why Care About **Memory**?

- Embedded systems typically have **only hundreds or tens of kilobytes** of RAM
- **Easy to hit memory limits** when building an end-to-end application
- So any framework that integrates with embedded products **must offer control over how memory usage**



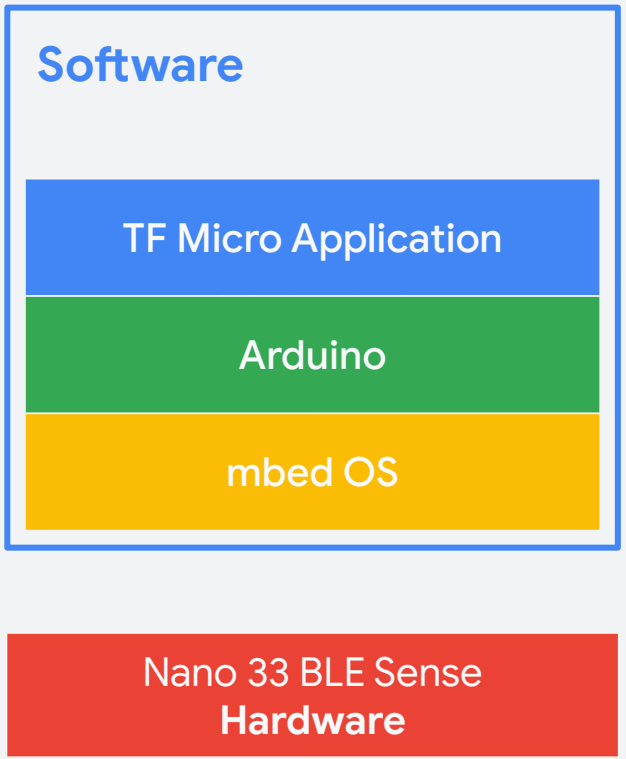
Long-Running Applications

- Products are **expected to run for months** or even years, which poses challenges for memory allocation
- Need to guarantee that memory allocation will not end up **fragmented** → **contiguous memory cannot be allocated** even if there's enough memory overall



Lack of **OS Support**

- In embedded systems, the standard C and C++ memory APIs (**`malloc`** and **`new`**) **rely on operating system support**
- Many devices have **no OS**, or have very **limited functionality**



How **TFL Micro** solves these challenges

1. Ask developers to **supply a contiguous area of memory** to the interpreter, and in return the framework avoids any other memory allocations

```
constexpr int kTensorArenaSize = 2000;
uint8_t tensor_arena[kTensorArenaSize];

...

static tflite::MicroInterpreter static_interpreter(model, resolver,
  tensor_arena, kTensorArenaSize, error_reporting);
```

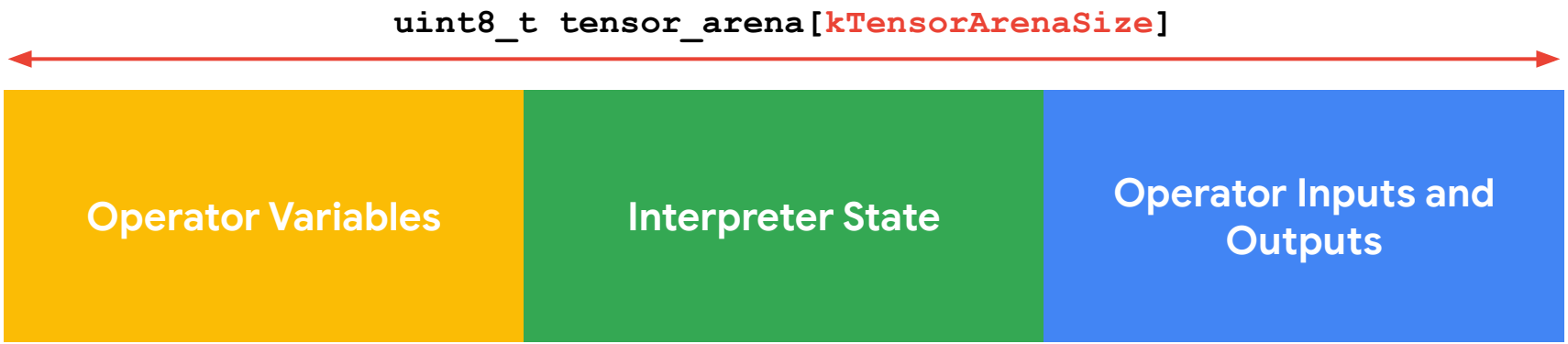
How **TFL Micro** solves these challenges

1. Ask developers to **supply a contiguous area of memory** to the interpreter, and in return the framework avoids any other memory allocations

2. Framework **guarantees that it won't allocate from this "arena" after initialization**, so long-running applications won't fail due to fragmentation

How **TFL Micro** solves these challenges

1. Ask developers to **supply a contiguous area of memory** to the interpreter, and in return the framework avoids any other memory allocations
2. Framework **guarantees that it won't allocate from this "arena" after initialization**, so long-running applications won't fail due to fragmentation
3. Ensures clear budget for the memory used by ML, and that the **framework has no dependency on OS facilities needed by malloc or new**


uint8_t tensor_arena[kTensorArenaSize]
Operator Variables
Interpreter State
Operator Inputs and Outputs

Arena **size**?

- **Depends on what ops are in the model** (and the parameters of those operations)
- Size of operator inputs and outputs is platform independent, **but different devices** can have **different operator implementations**
- → **hard to forecast *exact* size** of arena needed

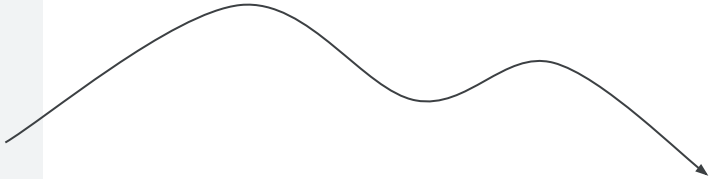
```
constexpr int kTensorArenaSize = 2000;
uint8_t tensor_arena[kTensorArenaSize];

...

static tflite::MicroInterpreter static_interpreter(model,
   resolver, tensor_arena, kTensorArenaSize, error_reporting);
```

Solution

- **Create as large an arena as you can** and run your program on-device
- Use the arena_used_bytes() function to get the actual size used.
- **Resize the arena to that length** and rebuild
- Best to **do this on your deployment platform**, since different op implementations may need varying scratch buffer sizes



```
constexpr int kTensorArenaSize = 6000;
uint8_t tensor_arena[kTensorArenaSize];

...

static tflite::MicroInterpreter static_interpreter(model,
   resolver, tensor_arena, kTensorArenaSize, error_reporting);
```

** Call `MicroInterpreter::arena_used_bytes()` to get the actual memory size used.*