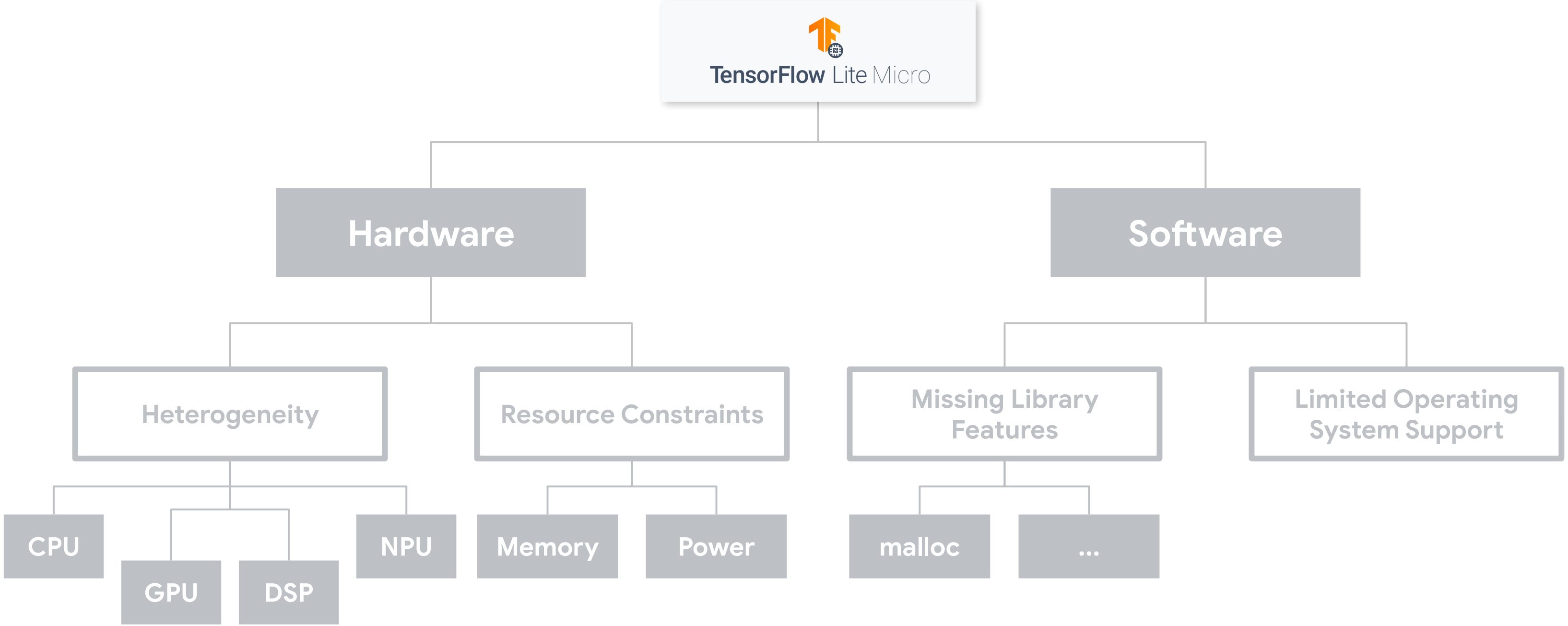
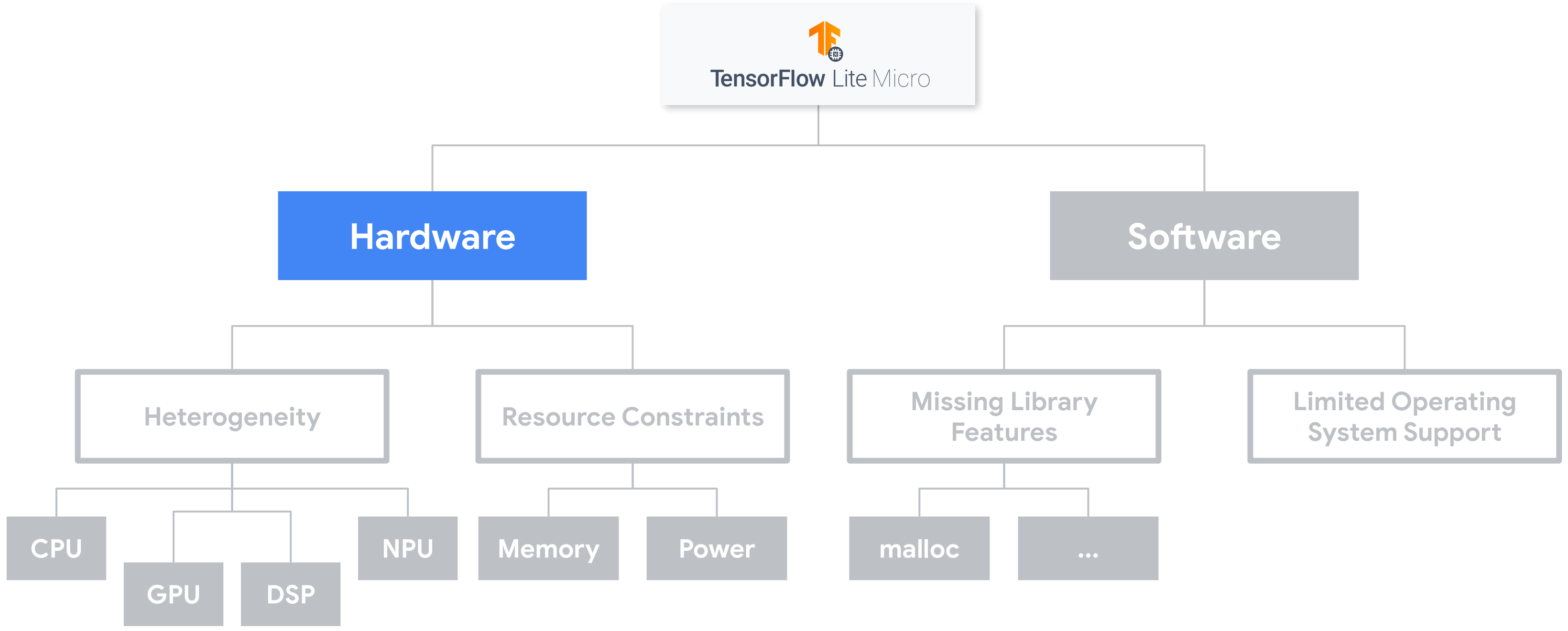
TensorFlow Lite Micro

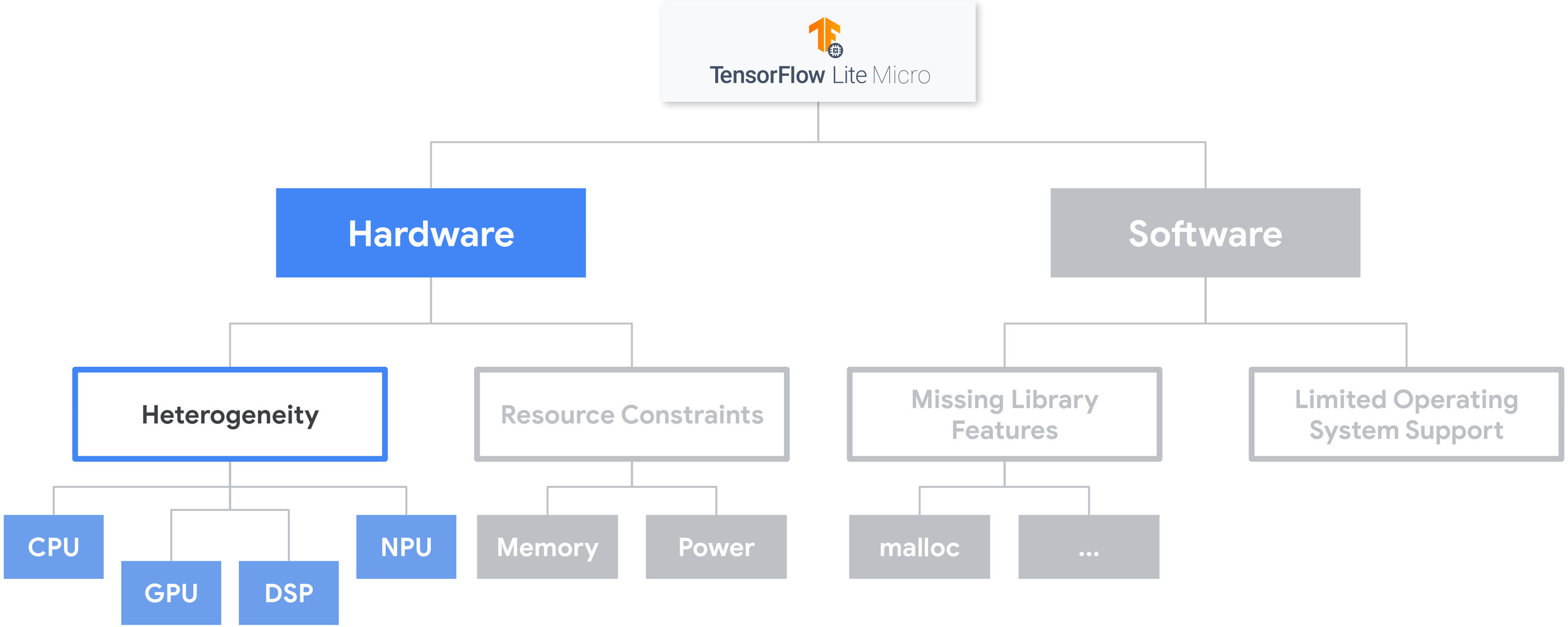
Embedded Machine Learning on TinyML Systems

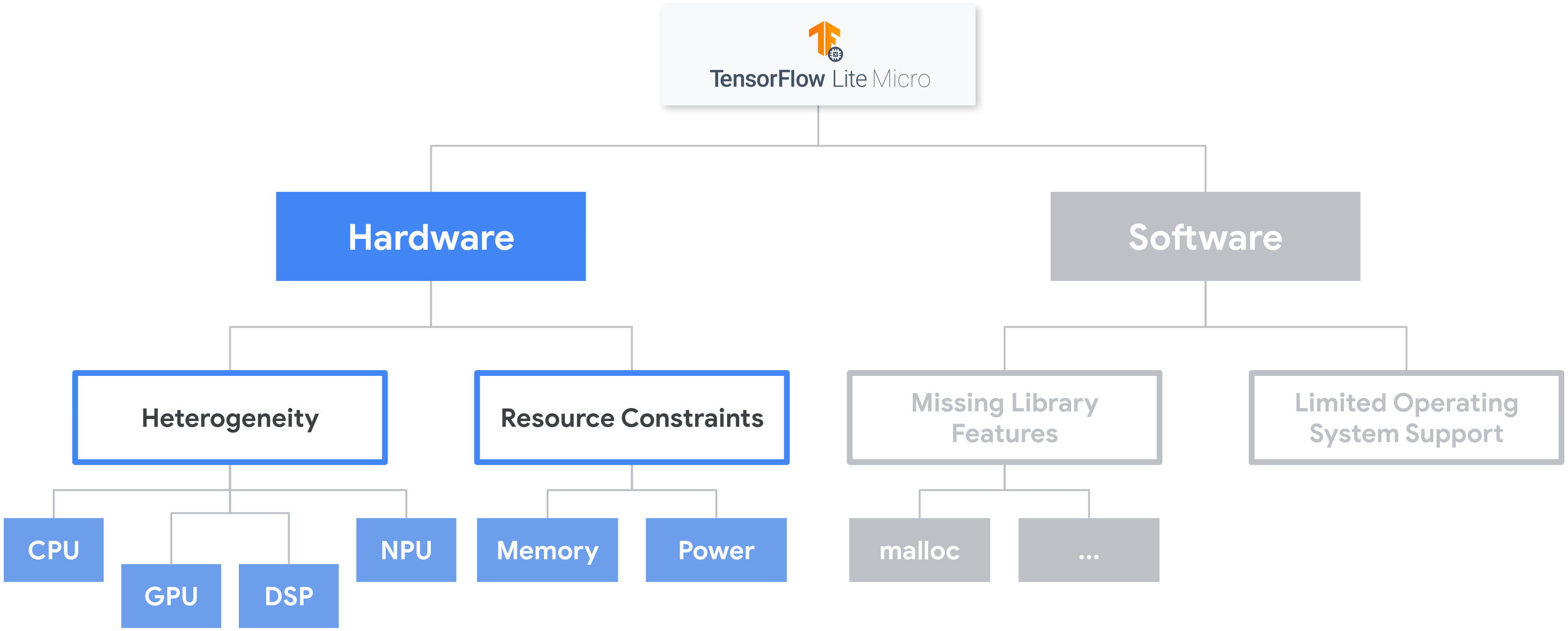
Robert David, Jared Duke, Advait Jain, **Vijay Janapa-Reddi**,
Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj,
Shlomi Regev, Rocky Rhodes, Tiezhen Wang, Pete Warden.

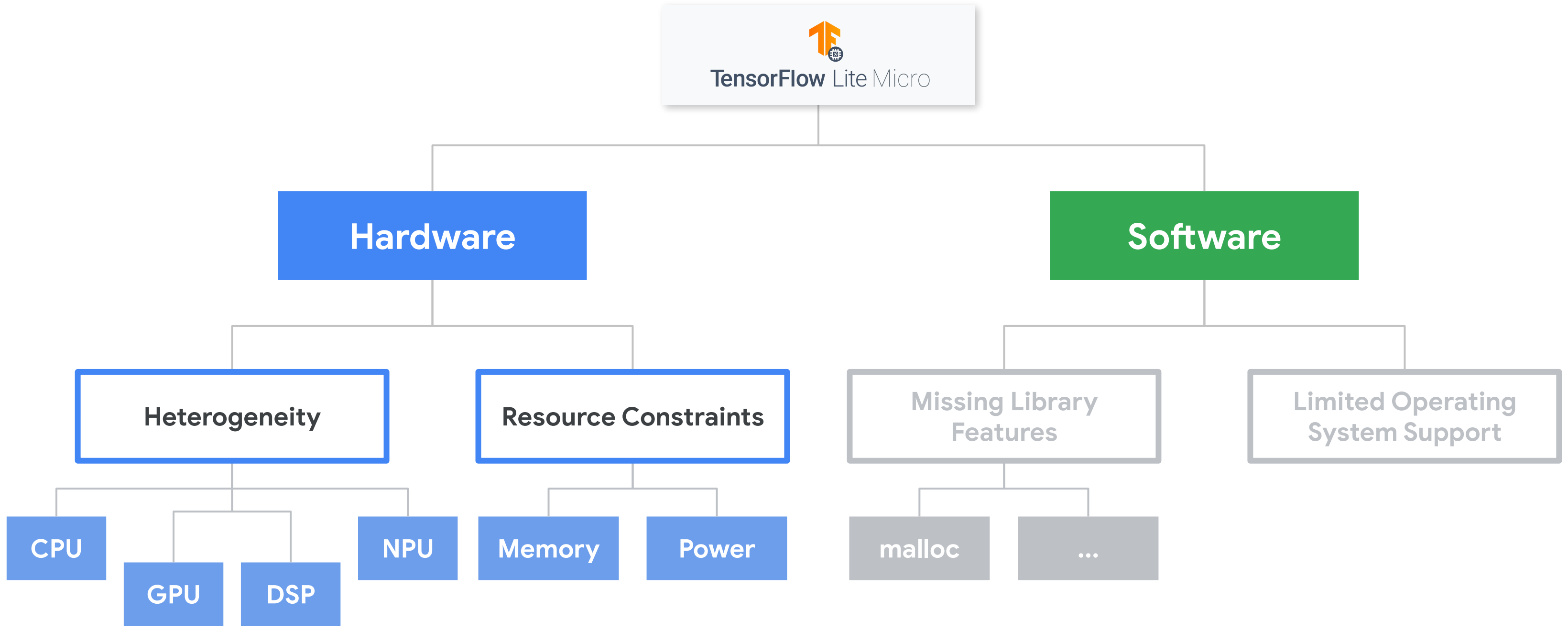
TensorFlow Lite Micro

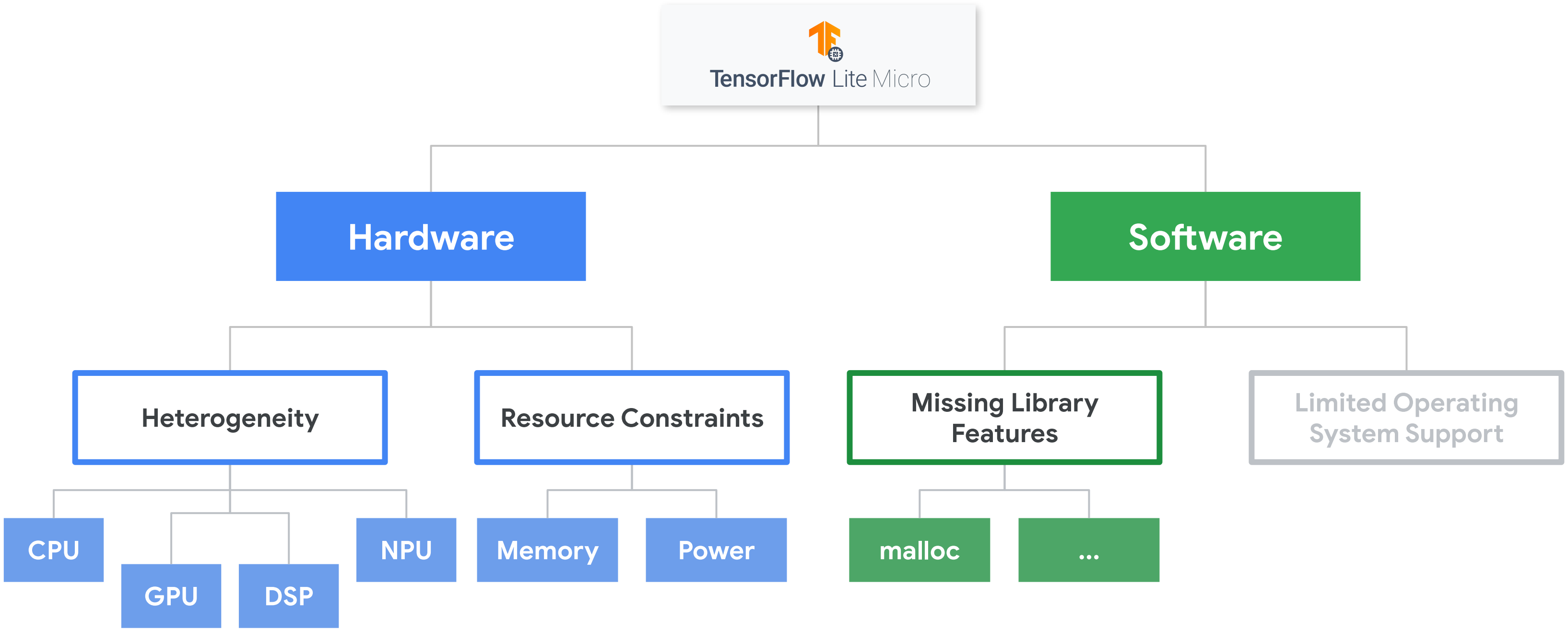

TensorFlow Lite Micro
Hardware
Software
Heterogeneity
Resource Constraints
Missing Library Features
Limited Operating System Support
CPU
GPU
DSP
NPU
Memory
Power
malloc
...

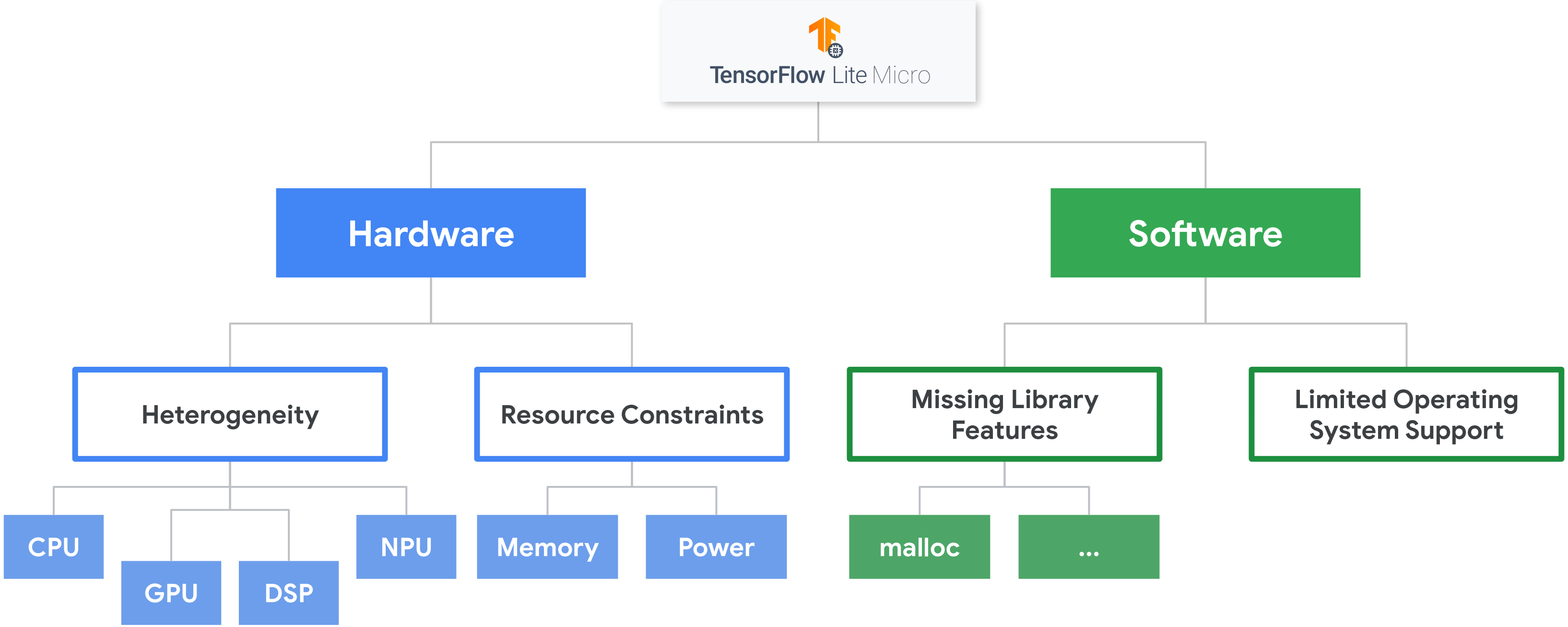

TensorFlow Lite Micro
Hardware
Software
Heterogeneity
Resource Constraints
Missing Library Features
Limited Operating System Support
CPU
GPU
DSP
NPU
Memory
Power
malloc
...


TensorFlow Lite Micro
Hardware
Software
Heterogeneity
Resource Constraints
Missing Library Features
Limited Operating System Support
CPU
GPU
DSP
NPU
Memory
Power
malloc
...


TensorFlow Lite Micro
Hardware
Software
Heterogeneity
Resource Constraints
Missing Library Features
Limited Operating System Support
CPU
GPU
DSP
NPU
Memory
Power
malloc
...


TensorFlow Lite Micro
Hardware
Software
Heterogeneity
Resource Constraints
Missing Library Features
Limited Operating System Support
CPU
GPU
DSP
NPU
Memory
Power
malloc
...


TensorFlow Lite Micro
Hardware
Software
Heterogeneity
Resource Constraints
Missing Library Features
Limited Operating System Support
CPU
GPU
DSP
NPU
Memory
Power
malloc
...


TensorFlow Lite Micro
Hardware
Software
Heterogeneity
Resource Constraints
Missing Library Features
Limited Operating System Support
CPU
GPU
DSP
NPU
Memory
Power
malloc
...

TensorFlow Lite Micro

...

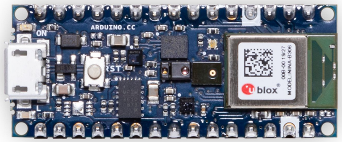
Arduino
BLE Sense 33

Himax
WE-I Plus EVB

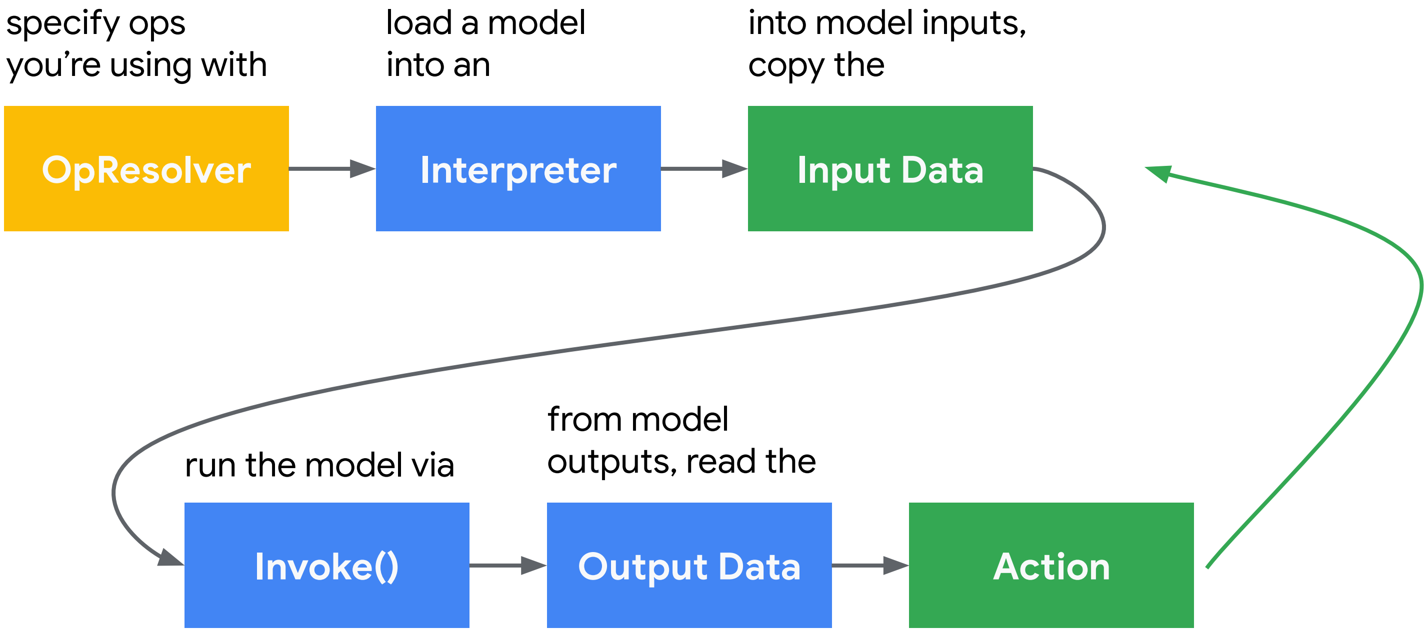
SparkFun
Edge 2

Espressif
EYE

...



How do you use **TFL Micro**?

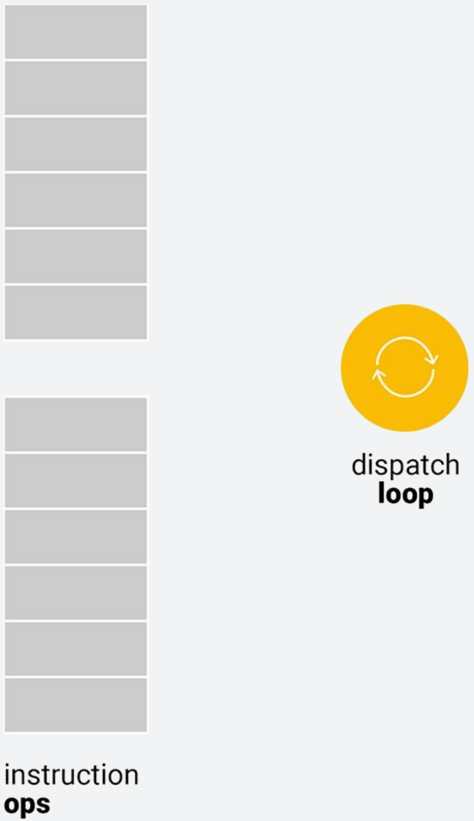


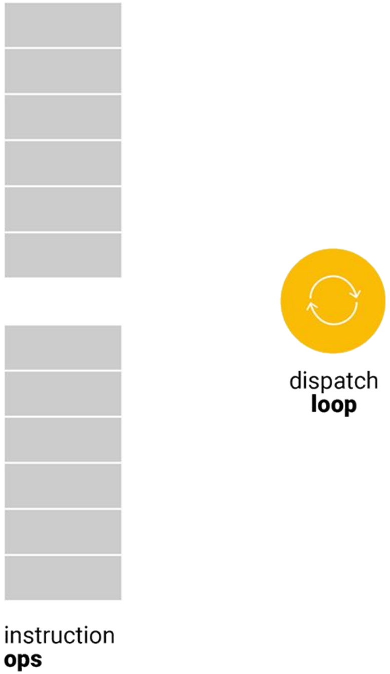
TFLite Micro: Interpreter

TFLite Micro Design

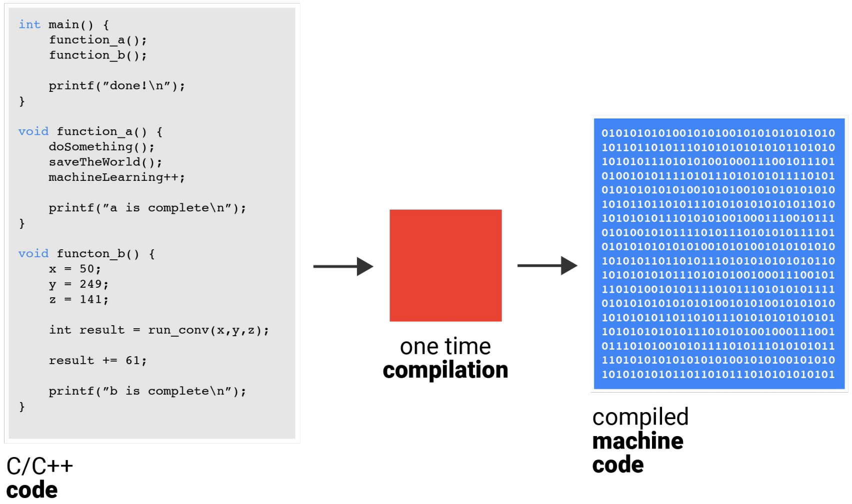
- TFLite Micro uses an **interpreter** design

- Store the model as data and loop through its ops at **runtime**

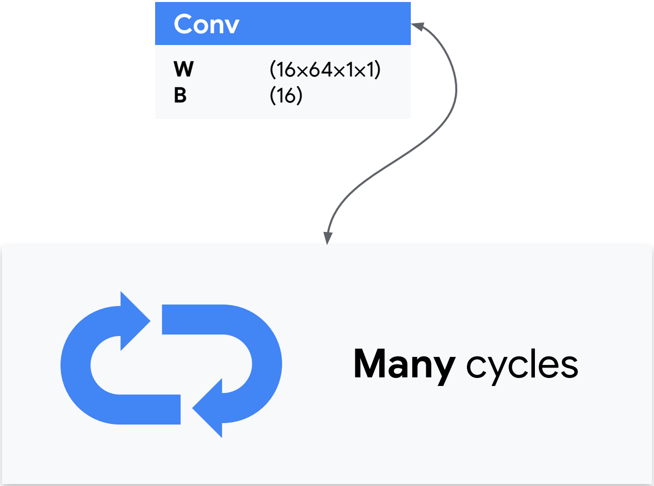




Interpreter
(generally **slower** than compiled code)

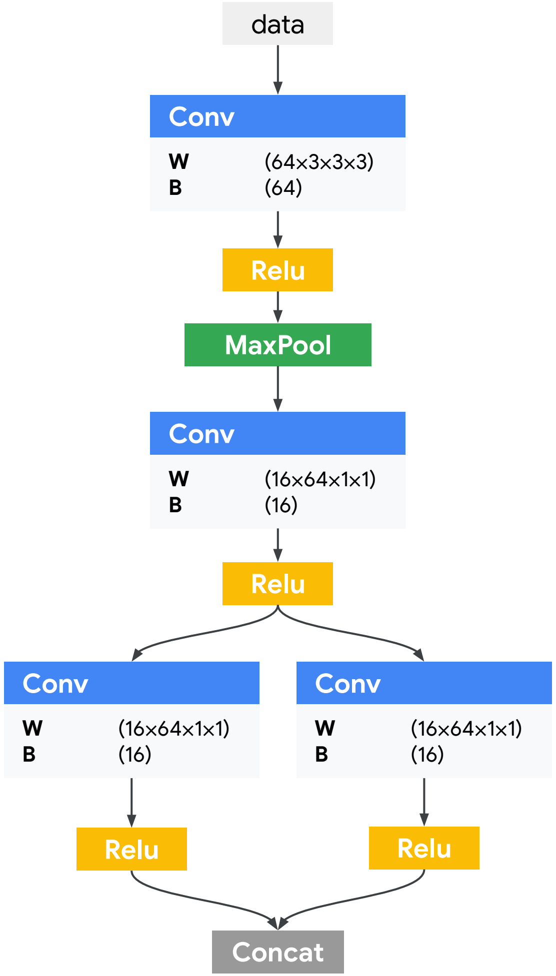


Compiler
(generally **faster** than interpreted code)



ML is **Different**

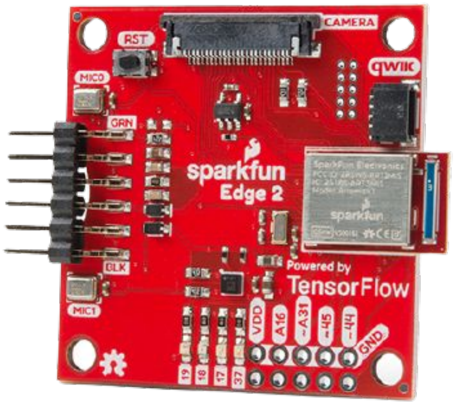
- Each layer like a **`Conv`** or **`softmax`** can take tens of thousands or even millions of cycles to complete execution



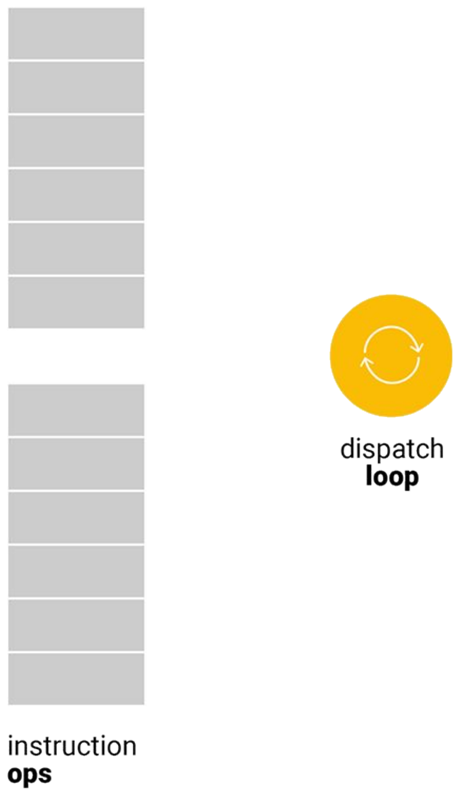
ML is **Different**

- Parsing overhead is **relatively small** for the TFMicro interpreter when we consider the **overall network graph**

Model	Total Cycles	Calculation Cycles	Interpreter Overhead
Visual Wake Words (Ref)	18,990.8K	18,987.1K	**< 0.1%**
Google Hotword (Ref)	36.4K	34.9K	**4.1%**

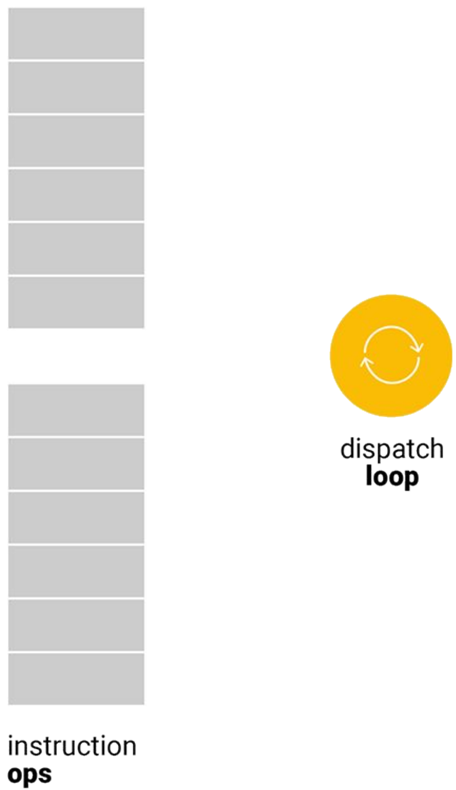


Sparkfun Edge 2
(Apollo 3 **Cortex-M4**)



Interpreter
Advantages

- Change the model **without recompiling** the code



Interpreter **Advantages**

- Change the model **without recompiling** the code
- **Same operator code** can be used across multiple **different models** in the system

Arduino
BLE Sense 33

Himax
WE-I Plus EVB

Espressif
EYE

SparkFun
Edge 2

Interpreter **Advantages**

- Same **portable** model serialization format can be used **across a lots of systems.**

TFLite Micro
Interpreter Execution

```
if (op_type == CONV2D) {
  Convolution2d(conv_size, input, output, weights);
} else if (op_type == FULLY_CONNECTED) {
  FullyConnected(input, output, weights)

}
```

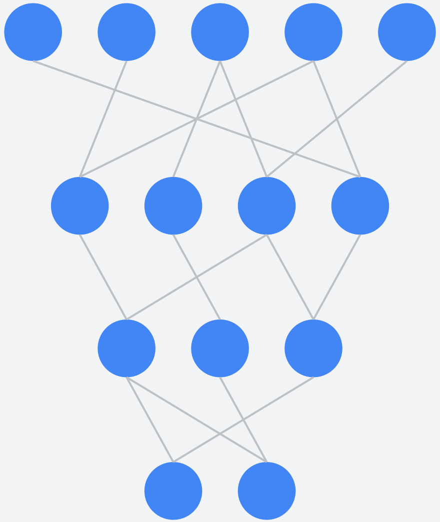
TFLite Micro: Model Format

The FlatBuffer File Format

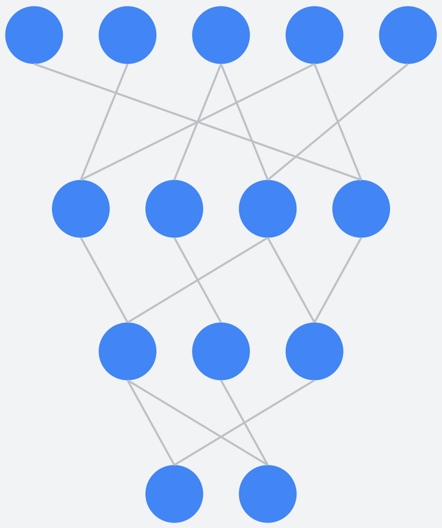
```
// Map the model into a usable data structure.  This doesn't involve any
// copying or parsing, it's a very lightweight operation.

    model = tflite::GetModel(g_model);
    if (model->version() != TFLIGHT_SCHEMA_VERSION) {
        TF_LITE_REPORT_ERROR(error_reporter,
                    "Model provided is schema version %d not equal "
                    "to supported version %d.",
                    model->versison(), TFLITE_SCHEMA_VERSION);

        return;
    }
```

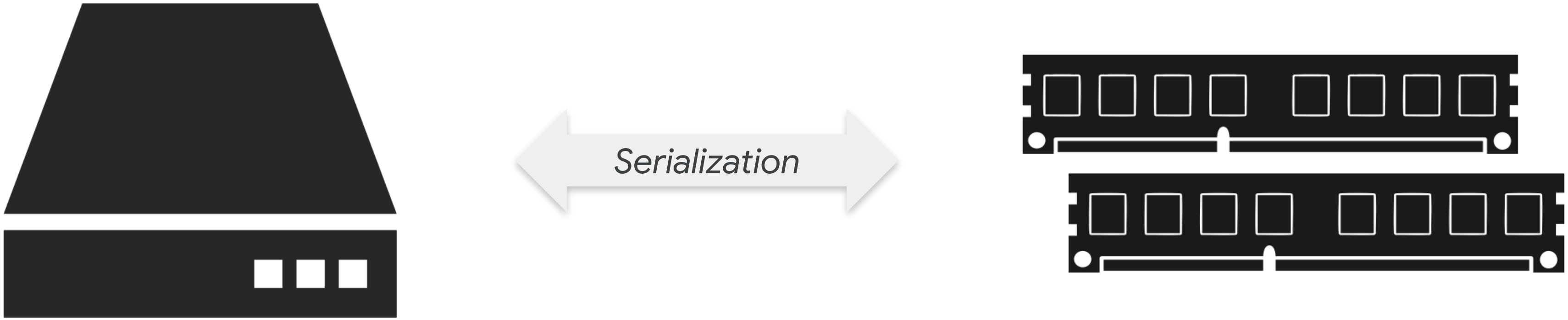
```
// Map the model into a usable data structure.  This doesn't involve any
// copying or parsing, it's a very lightweight operation.

    model = tflite::GetModel(g_model);
    if (model->version() != TFLIGHT_SCHEMA_VERSION) {
        TF_LITE_REPORT_ERROR(error_reporter,
                             "Model provided is schema version %d not equal "
                             "to supported version %d.",
                             model->versison(), TFLITE_SCHEMA_VERSION);

        return;
    }
```

```
// Map the model into a usable data structure.  This doesn't involve
any
// copying or parsing, it's a very lightweight operation.

model = tflite::GetModel(g_model);
if (model->version() != TFLIGHT_SCHEMA_VERSION) {
    TF_LITE_REPORT_ERROR(error_reporter,
                        "Model provided is schema version %d not equal
"
                        "to supported version %d.",
                        model->versison(), TFLITE_SCHEMA_VERSION);

    return;
}
```



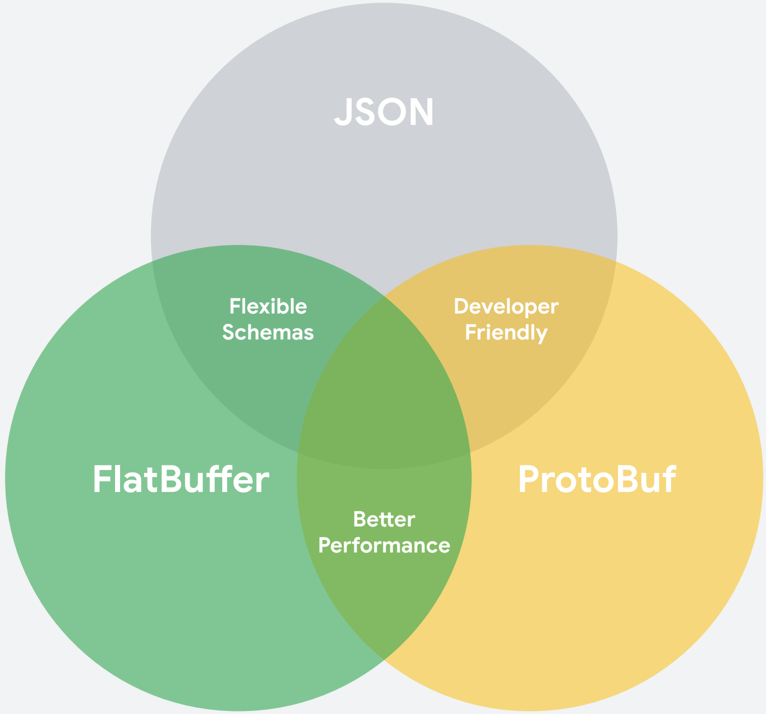
How is `g_model` stored?

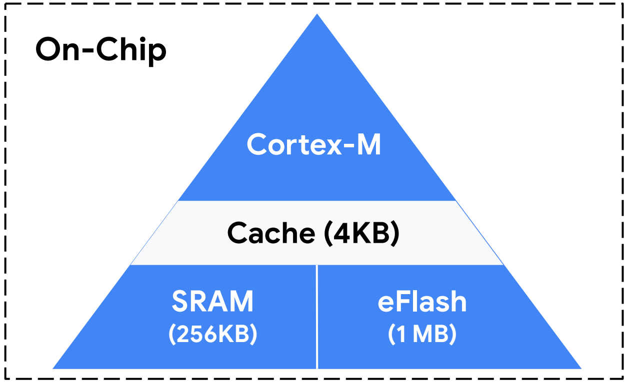


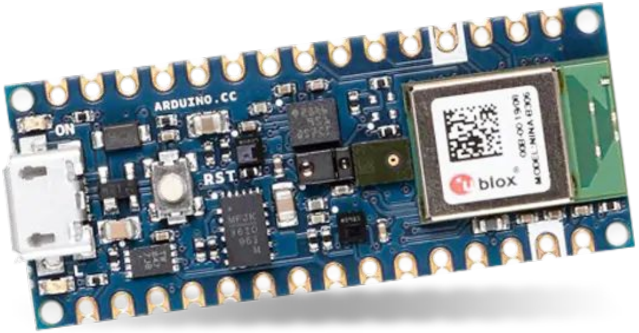

Serialization

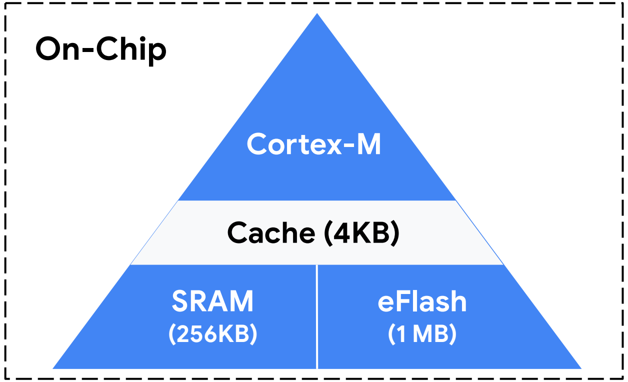
Serialization Libraries

- JSON
- ProtoBuf
- FlatBuffer



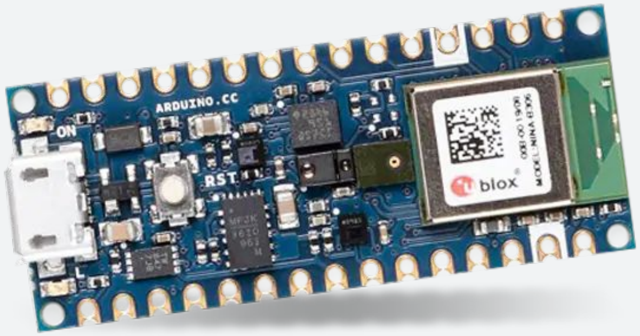

On-Chip
Cortex-M
Cache (4KB)
SRAM
(256KB)
eFlash
(1 MB)


ARDUINO.CC
ON
RST
blox



Hardware & Software Limitations

- Limited **OS support**
- Limited **compute**
- Limited **memory**



What is `g_model`?

- **Array of bytes**, and acts as the equivalent of a file on disk

- Holds **all** of the **information about the model**, its **operators**, their **connections**, and the trained **weights**

```
// Automatically created from a TensorFlow Lite flatbuffer using the command:
// xxd -i model.tflite > model.cc

// This is a standard TensorFlow Lite model file that has been converted into a
// C data array, so it can be easily compiled into a binary for devices that
// don't have a file system.

// See train/README.md for a full description of the creation process.

#include "model.h"

// Keep model aligned to 8 bytes to guarantee aligned 64-bit accesses.
alignas(8) const unsigned char g_model[] = {
    0x1c, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x00, 0x00, 0x12, 0x00,
    0x1c, 0x00, 0x04, 0x00, 0x08, 0x00, 0x0c, 0x00, 0x10, 0x00, 0x14, 0x00,
    0x00, 0x00, 0x18, 0x00, 0x12, 0x00, 0x00, 0x00, 0x03, 0x00, 0x00, 0x00,
    0x60, 0x09, 0x00, 0x00, 0xa8, 0x02, 0x00, 0x00, 0x90, 0x02, 0x00, 0x00,
    0x3c, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00,
    0x0c, 0x00, 0x00, 0x00, 0x08, 0x00, 0x0c, 0x00, 0x04, 0x00, 0x08, 0x00,
    0x08, 0x00, 0x00, 0x00, 0x08, 0x00, 0x00, 0x00, 0x0b, 0x00, 0x00, 0x00,
    0x13, 0x00, 0x00, 0x00, 0x6d, 0x69, 0x6e, 0x5f, 0x72, 0x75, 0x6e, 0x74,
    0x69, 0x6d, 0x65, 0x5f, 0x76, 0x65, 0x72, 0x73, 0x69, 0x6f, 0x6e, 0x00,
    0x0c, 0x00, 0x00, 0x00, 0x48, 0x02, 0x00, 0x00, 0x34, 0x02, 0x00, 0x00,
    0x0c, 0x02, 0x00, 0x00, 0xfc, 0x00, 0x00, 0x00, 0xac, 0x00, 0x00, 0x00,
    0x8c, 0x00, 0x00, 0x00, 0x3c, 0x00, 0x00, 0x00, 0x34, 0x00, 0x00, 0x00,
    0x2c, 0x00, 0x00, 0x00, 0x24, 0x00, 0x00, 0x00, 0x1c, 0x00, 0x00, 0x00,
    0x04, 0x00, 0x00, 0x00, 0xfe, 0xfd, 0xff, 0xff, 0x04, 0x00, 0x00, 0x00,
    0x05, 0x00, 0x00, 0x00, 0x31, 0x2e, 0x35, 0x2e, 0x30, 0x00, 0x00, 0x00,
    0x7c, 0xfd, 0xff, 0xff, 0x80, 0xfd, 0xff, 0xff, 0x84, 0xfd, 0xff, 0xff,
    ...
    0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x06, 0x00, 0x06, 0x00, 0x05, 0x00,
    0x06, 0x00, 0x00, 0x00, 0x00, 0x72, 0x0a, 0x00, 0x0c, 0x00, 0x07, 0x00,
    0x00, 0x00, 0x08, 0x00, 0x0a, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x09,
    0x04, 0x00, 0x00, 0x00};
const int g_model_len = 2512;
```

FlatBuffers

- Does **not require copies** to be made before using the data inside the model

FlatBuffers

- Does **not require copies** to be made before using the data inside the model
- The **format** is formally specified as a **schema file**

FlatBuffers

- Does **not require copies** to be made before using the data inside the model
- The **format** is formally specified as a **schema file**
- Schema file is used to automatically **generate code** to access the information in the **model byte array**

`g_model` FlatBuffer **Format**

Metadata (version, quantization ranges, etc)

Name	*Args*	*Input*	*Output*	*Weights*
Conv2D	3x3	0	1	2
FC	-	1	3	4
Softmax	-	3	5	-

Weight Buffers

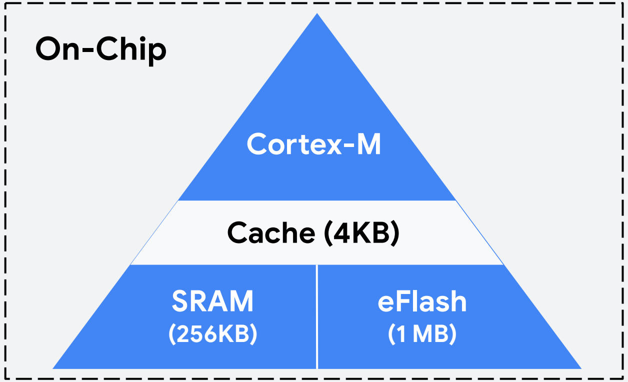
Index	*Type*	*Values*
2	Float	0.01, 7.45, 9.23, ...
4	Int8	34, 19, 243, ...
...	...	...

TFLite Micro: Memory Allocation

The Tensor Arena

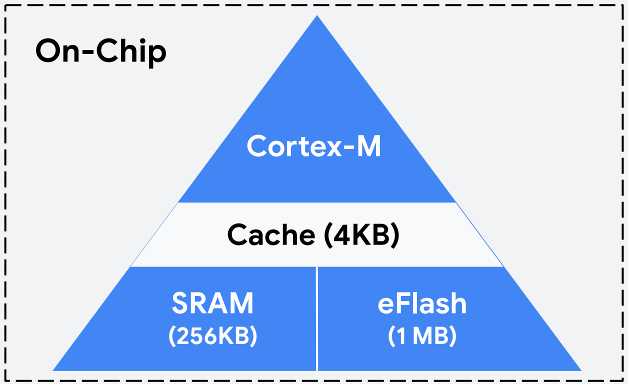
Why Care About **Memory**?

- Embedded systems typically have **only hundreds or tens of kilobytes** of RAM



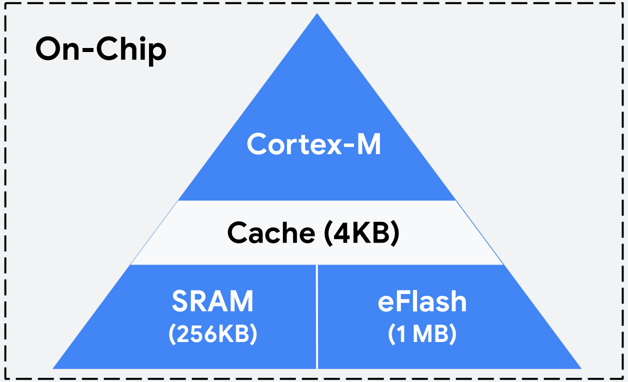
Why Care About **Memory**?

- Embedded systems typically have **only hundreds or tens of kilobytes** of RAM
- **Easy to hit memory limits** when building an end-to-end application



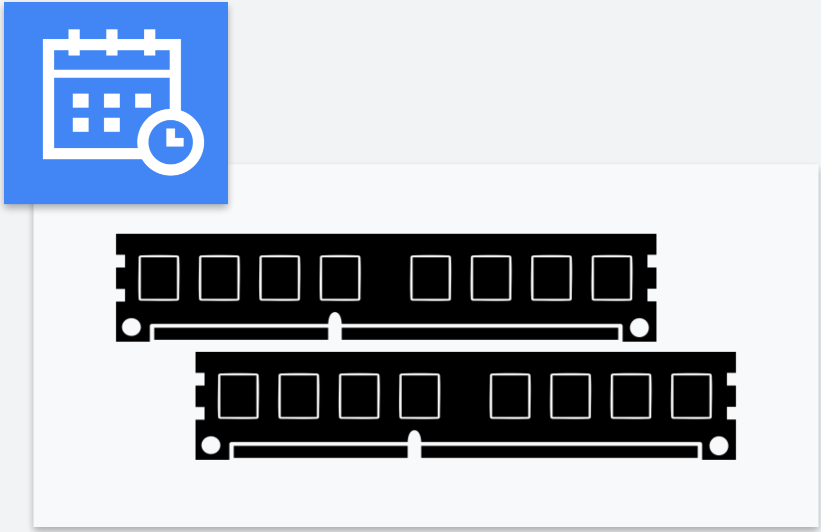
Why Care About **Memory**?

- 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**



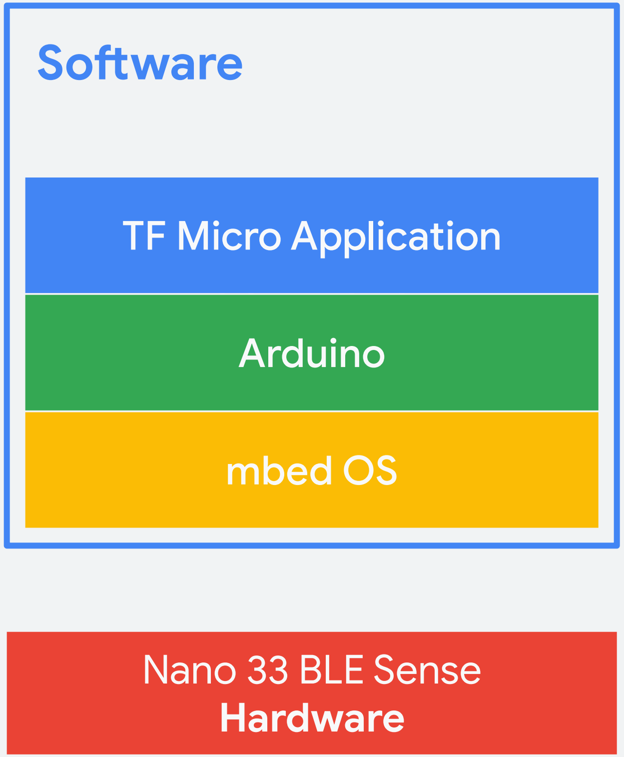
Long-Running Applications

- 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

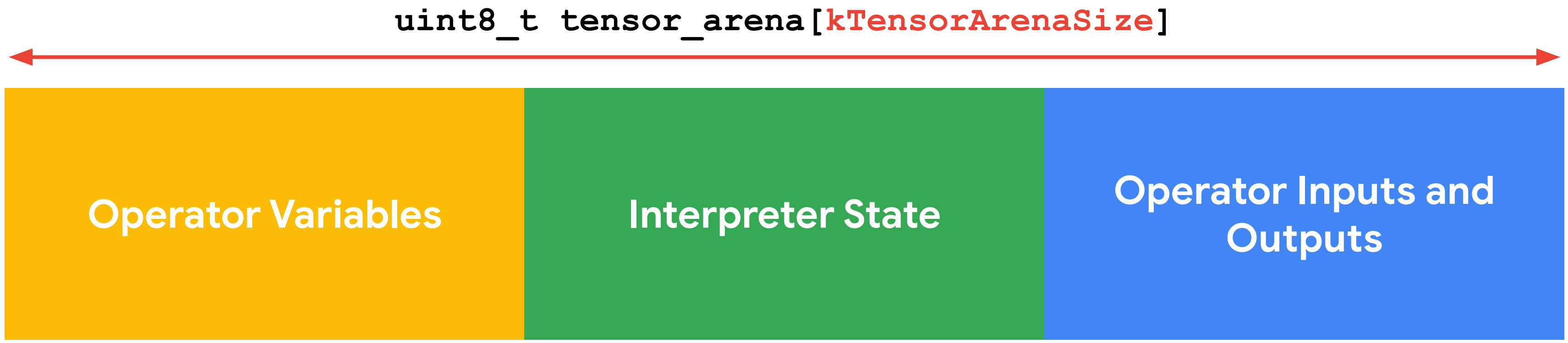
2. Framework **guarantees that it won't allocate from this "arena" after initialization**, so long-running applications won't fail due to fragmentation

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)

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

...

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

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**

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

...

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

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);
```

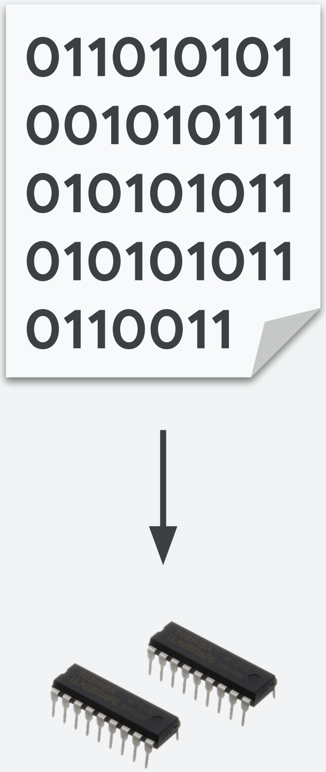
* *Call* `MicroInterpreter::arena_used_bytes()` *to get the actual memory size used.*

TFLite Micro: NN Operations

The OpsResolver

Why Care About **Binary Size?**

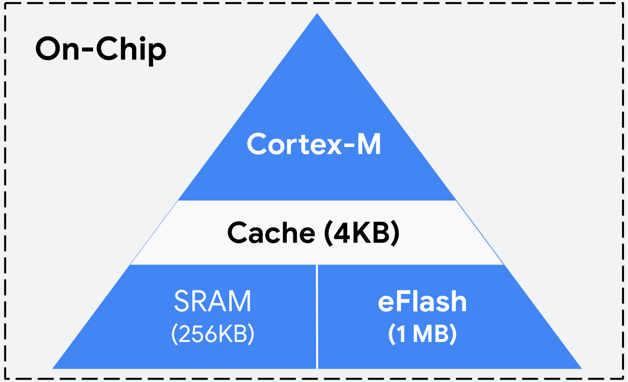
- **Executable code** used by a framework takes up space in Flash



Why Care About **Binary Size?**

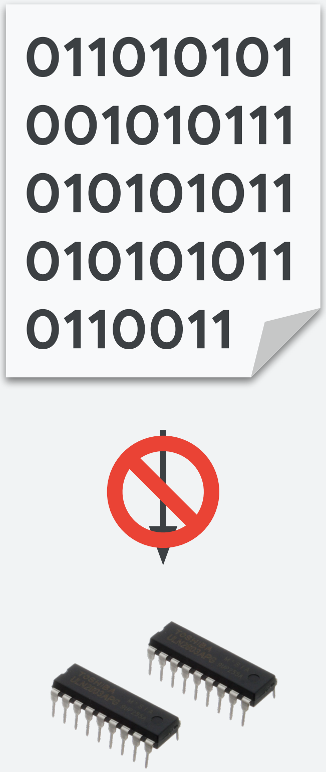
- **Executable code** used by a framework takes up space in Flash

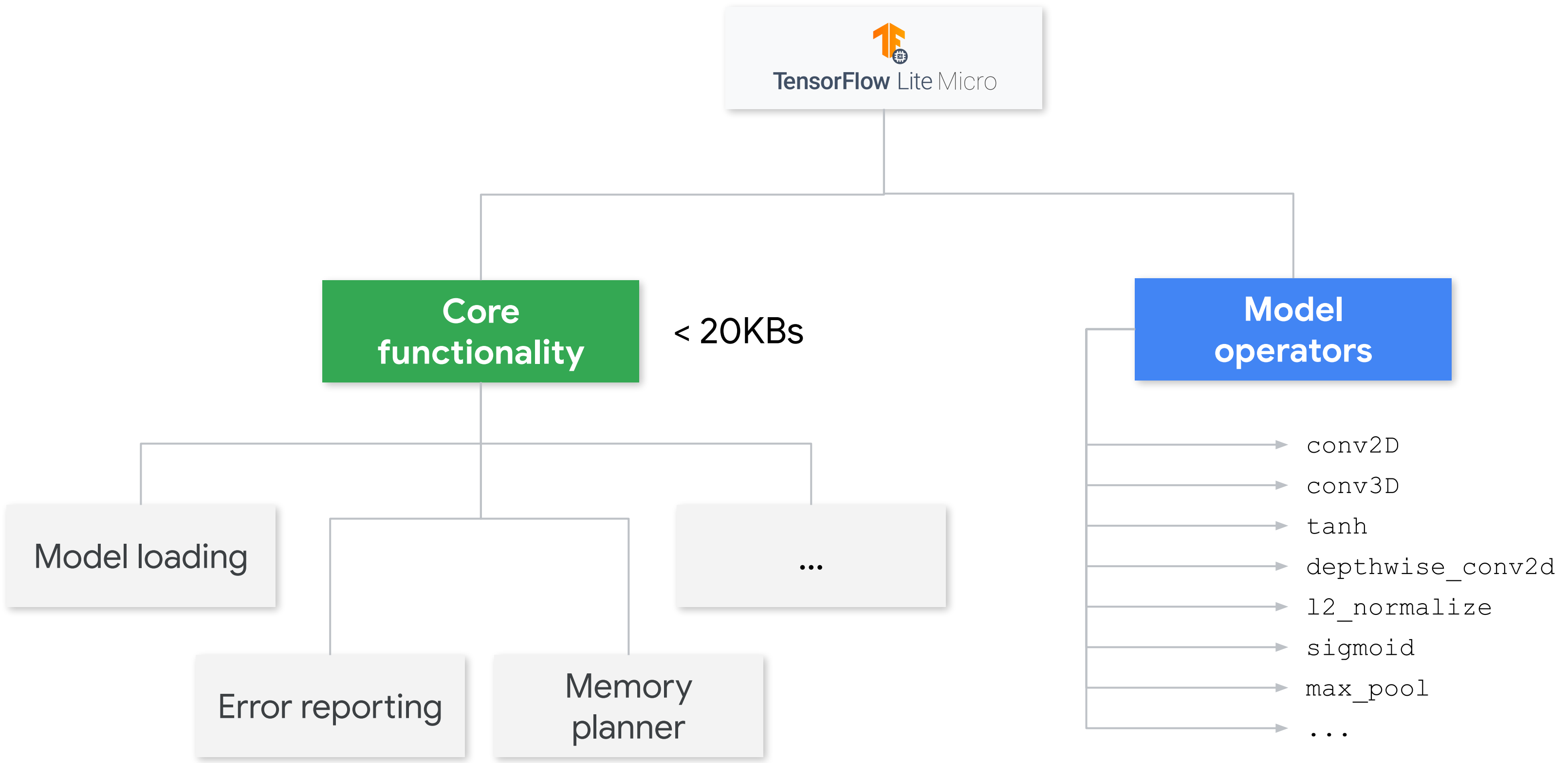
- **Flash is a limited resource** on embedded devices and often just tens of kilobytes available



Why Care About **Binary Size?**

- **Executable code** used by a framework takes up space in Flash
- **Flash is a limited resource** on embedded devices and often just tens of kilobytes available
- If compiled **code is too large**, it **won't be usable** by applications.




TensorFlow Lite Micro
Core functionality
< 20KBs
Model operators
Model loading
...
Error reporting
Memory planner
conv2D
conv3D
tanh
depthwise_conv2d
l2_normalize
sigmoid
max_pool
...

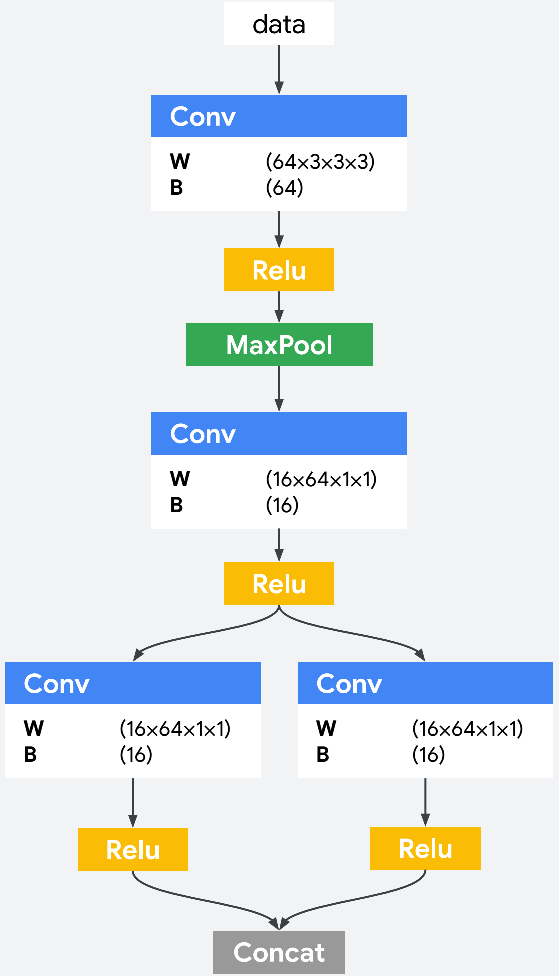
Optimizing Operator Usage in TFL Micro

- There are **many operators in TensorFlow** (~1400 and growing)

Optimizing Operator Usage in TFL Micro

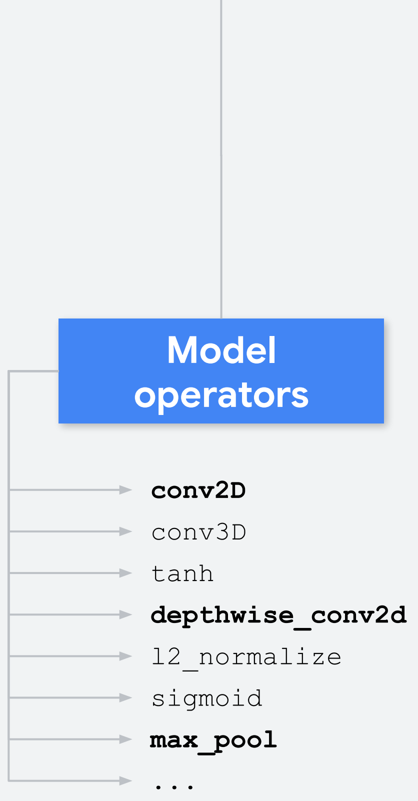
- There are **many operators in TensorFlow** (~1400 and growing)

- **Not all operators are used** or even needed to perform inference



Optimizing Operator Usage in TFL Micro

- There are **many operators in TensorFlow** (~1400 and growing)
- **Not all operators are used** or even needed to perform inference
- Bring in or **load only those that are important** to conserve memory usage

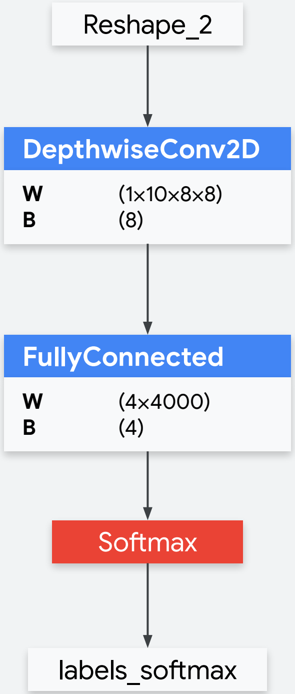


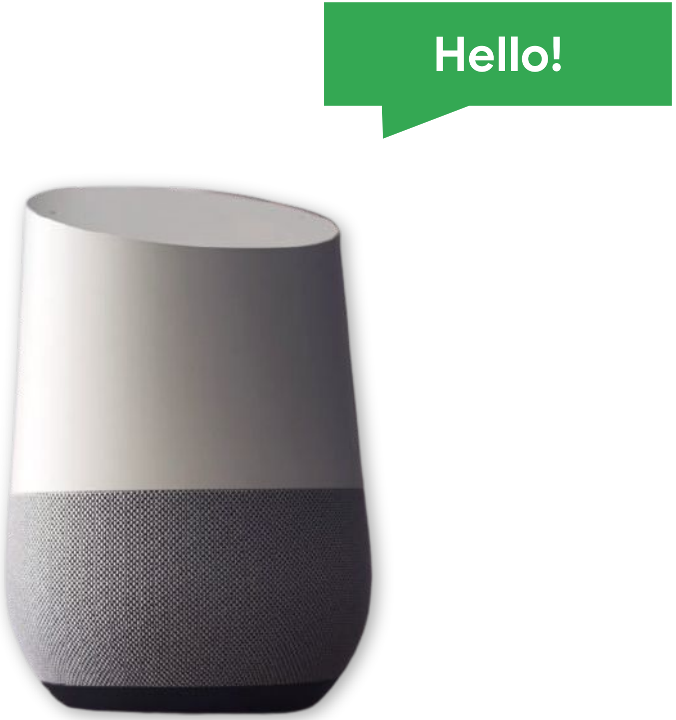
How to **Reduce the Size** Taken by Ops?

Allow developers to specify which ops they want to be included in the binary

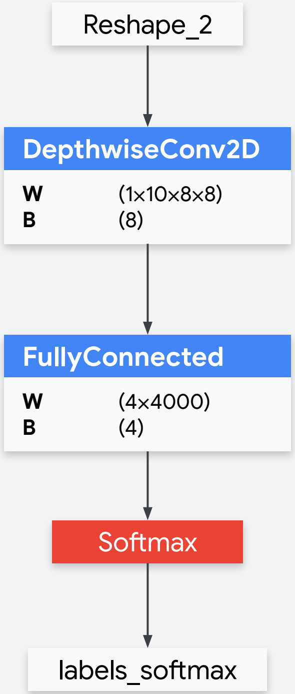
```
tflite::MicroMutableOpResolver<4>
op_resolver(error_reporter);
if (op_resolver.AddDepthwiseConv2D() != kTfLiteOk) {
    return;
}
```

TinyConv Keyword Spotting Model



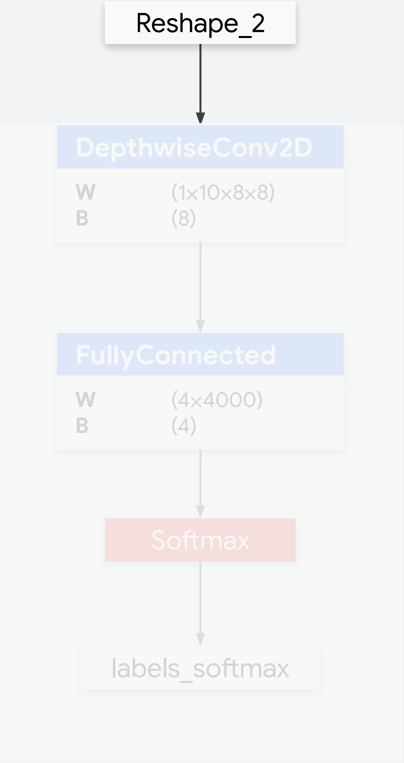


TinyConv Keyword Spotting Model



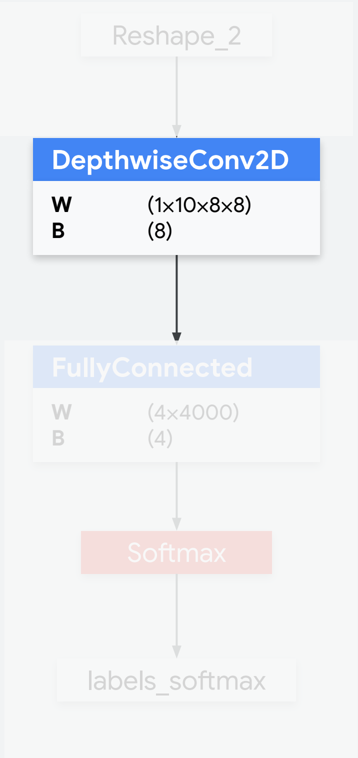

```
static tflite::MicroMutableOpResolver<4> micro_op_resolver(error_reporter);

if (micro_op_resolver.AddDepthwiseConv2D() != kTfLiteOk) {
  return;
}
if (micro_op_resolver.AddFullyConnected() != kTfLiteOk) {
  return;
}
if (micro_op_resolver.AddSoftmax() != kTfLiteOk) {
  return;
}
if (micro_op_resolver.AddReshape() != kTfLiteOk) {
  return;
}
```

TinyConv Keyword Spotting Model



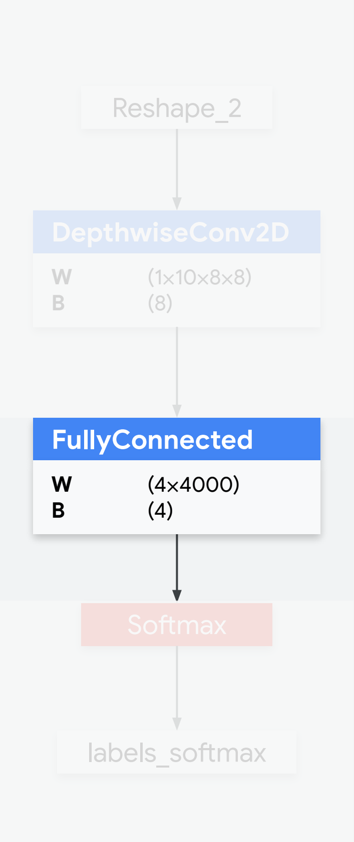

```
static tflite::MicroMutableOpResolver<4> micro_op_resolver(error_reporter);

if (micro_op_resolver.AddDepthwiseConv2D() != kTfLiteOk) {
  return;
}
if (micro_op_resolver.AddFullyConnected() != kTfLiteOk) {
  return;
}
if (micro_op_resolver.AddSoftmax() != kTfLiteOk) {
  return;
}
if (micro_op_resolver.AddReshape() != kTfLiteOk) {
  return;
}
```

TinyConv Keyword Spotting Model



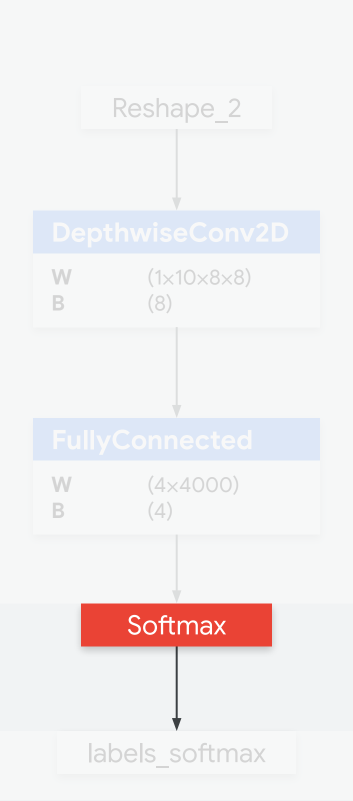

```
static tflite::MicroMutableOpResolver<4> micro_op_resolver(error_reporter);

if (micro_op_resolver.AddDepthwiseConv2D() != kTfLiteOk) {
  return;
}
if (micro_op_resolver.AddFullyConnected() != kTfLiteOk) {
  return;
}
if (micro_op_resolver.AddSoftmax() != kTfLiteOk) {
  return;
}
if (micro_op_resolver.AddReshape() != kTfLiteOk) {
  return;
}
```

TinyConv Keyword Spotting Model




```
static tflite::MicroMutableOpResolver<4> micro_op_resolver(error_reporter);

if (micro_op_resolver.AddDepthwiseConv2D() != kTfLiteOk) {
  return;
}
if (micro_op_resolver.AddFullyConnected() != kTfLiteOk) {
  return;
}
if (micro_op_resolver.AddSoftmax() != kTfLiteOk) {
  return;
}
if (micro_op_resolver.AddReshape() != kTfLiteOk) {
  return;
}
```

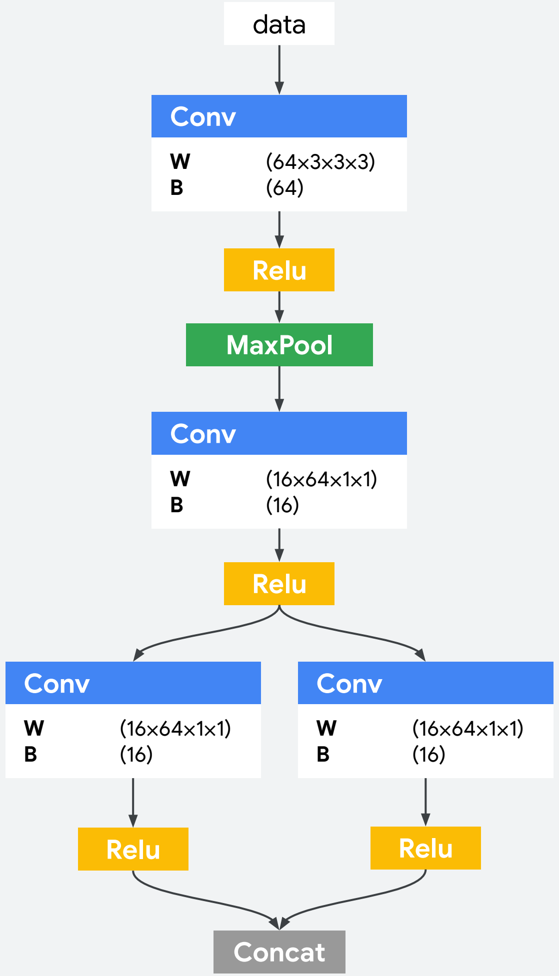
TinyConv Keyword Spotting Model




```
static tflite::MicroMutableOpResolver<4> micro_op_resolver(error_reporter);

if (micro_op_resolver.AddDepthwiseConv2D() != kTfLiteOk) {
  return;
}
if (micro_op_resolver.AddFullyConnected() != kTfLiteOk) {
  return;
}
if (micro_op_resolver.AddSoftmax() != kTfLiteOk) {
  return;
}
if (micro_op_resolver.AddReshape() != kTfLiteOk) {
  return;
}
```

Which Ops to Include?

https://netron.app

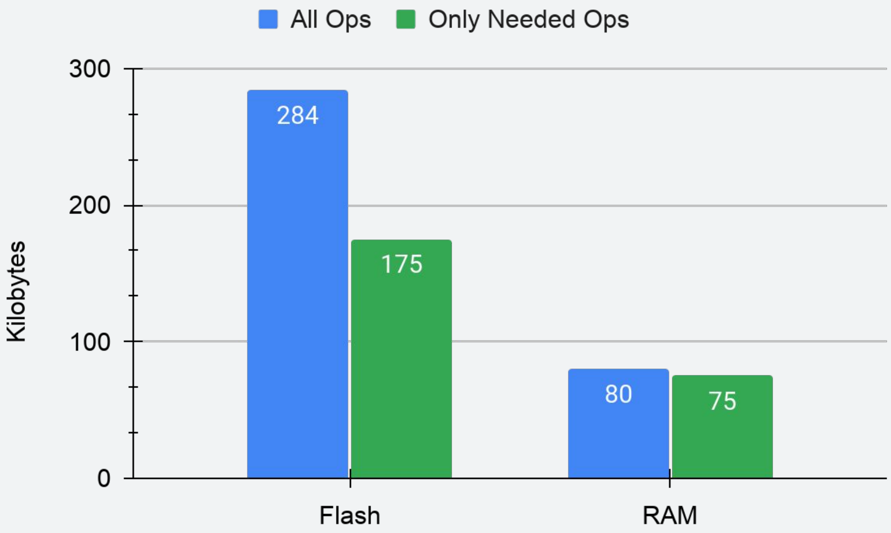


If memory is not an issue, you can choose to simply include all operators, both used and unused, at the expense of increased memory consumption

```
static tflite::AllOpsResolver resolver;

// Build an interpreter to run the model with.
static tflite::MicroInterpreter static_interpreter(
    model, resolver, tensor_arena, kTensorArenaSize, error_reporter);
interpreter = &static_interpreter;
```

Memory **Improvements**

- Selective op registration **reduces memory consumption by 30%**
- **Memory reduction varies by model**, depending on the operators used by the model



In Summary, what is TensorFlow Lite Micro?

TensorFlow Lite Micro

Compatible with the TensorFlow training environment.

Built to fit on **embedded systems**:

- Very **small binary footprint**
- **No** dynamic memory allocation
- **No** dependencies on complex parts of the standard C/C++ libraries
- **No** operating system dependencies, **can run on bare metal**
- Designed to be **portable** across a wide variety of systems

Thank You!