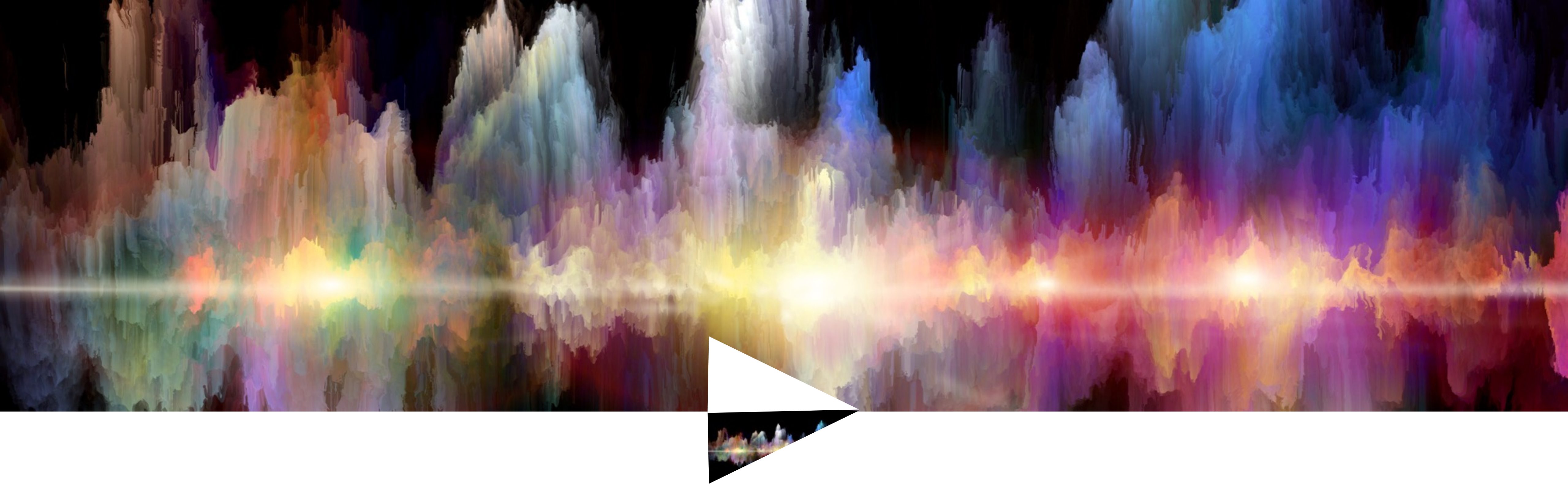


MLIR-based offline memory planning and other graph-level optimisations for xcore.ai

Deepak Panickal   Laszlo Kindrat*(Modular)   Scott Roset

XMOS

*Work done while at XMOS

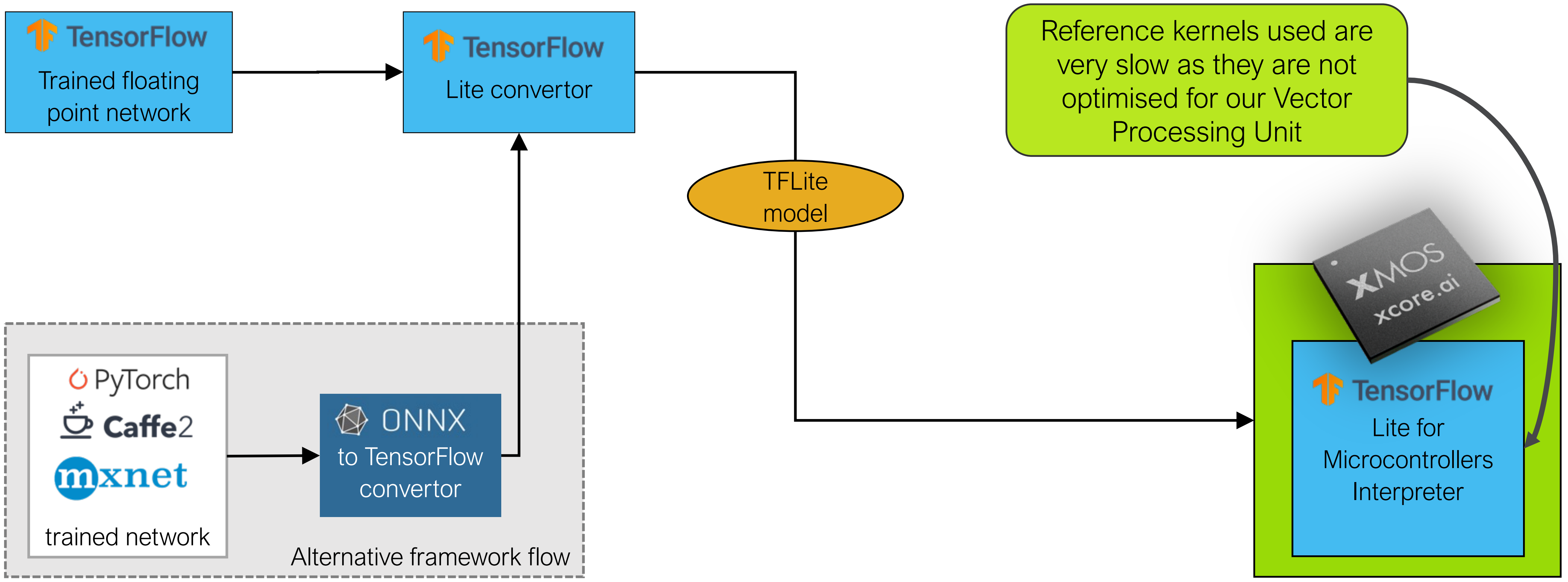
BACKGROUND

xcore.ai is a high performance, low latency microcontroller, with 16 logical cores split between two multithreaded processor ‘tiles’, each with 512KB of SRAM

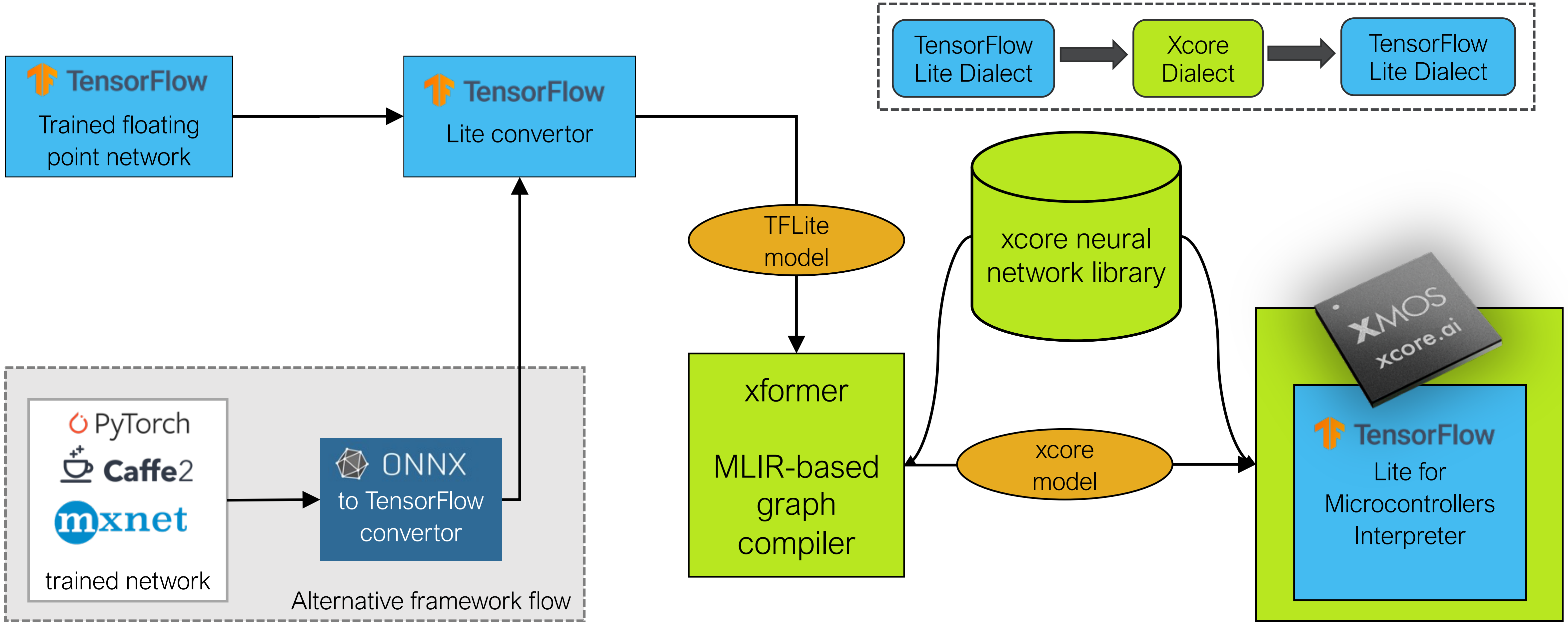
We have developed an MLIR-based graph compiler(xformer) to optimise TFLite models to deploy on xcore.ai

I will outline our workflow and focus on three MLIR passes implemented as part of memory usage optimisations

UNOPTIMISED WORKFLOW


TensorFlow
Trained floating point network
TensorFlow
Lite convertor
TFLite model
Reference kernels used are very slow as they are not optimised for our Vector Processing Unit
XMOS xcore.ai
TensorFlow
Lite for Microcontrollers Interpreter
PyTorch
Caffe2
mxnet
trained network
ONNX
to TensorFlow convertor
Alternative framework flow

OPTIMISED WORKFLOW WITH GRAPH COMPILER



INTERPRETER-LESS WORKFLOW

Why interpreter-less?

Code size is critical - only include the code for operators used in the model

Remove unnecessary runtime overhead such as interpreter setup and such code

tflite-micro-compiler

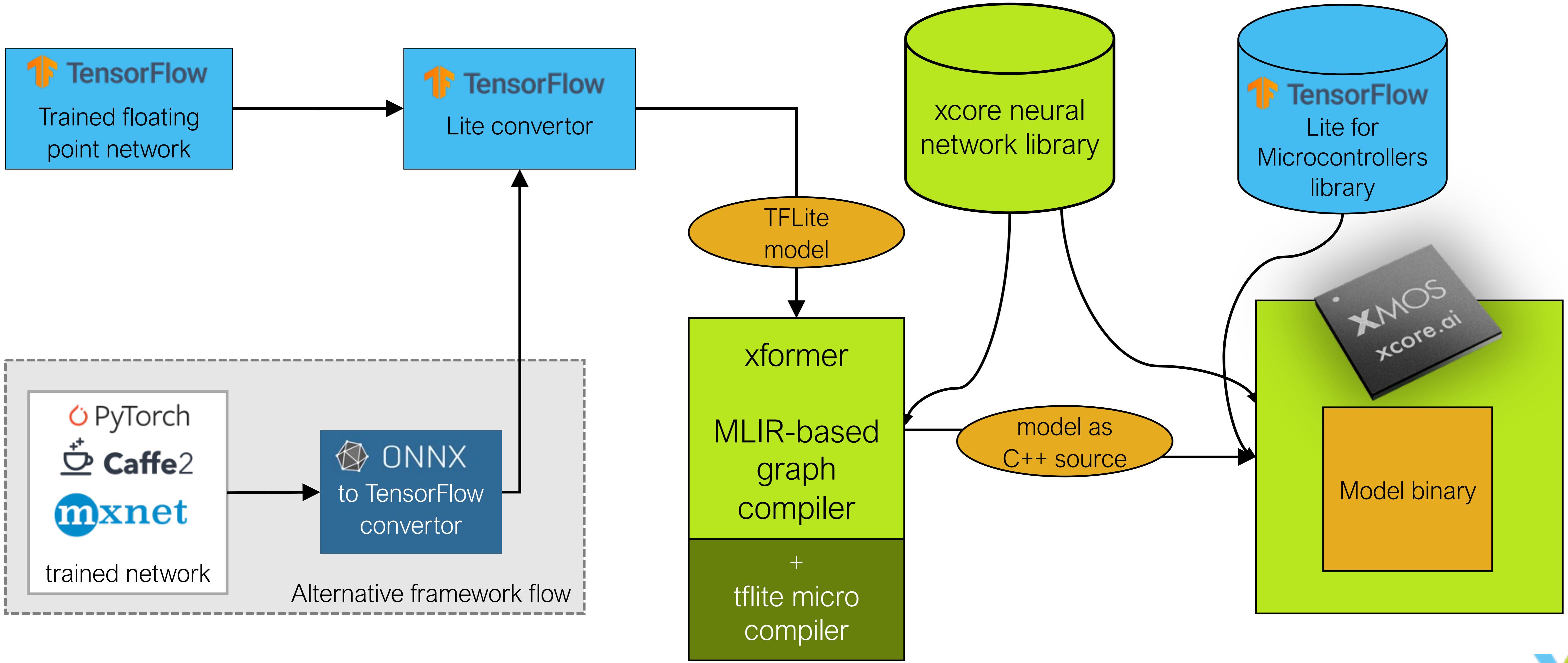
A tiny open-source project

Runs tflite-micro interpreter and logs runtime info

Generates a C++ file for the model

https://github.com/cpetig/tflite_micro_compiler

INTERPRETER-LESS WORKFLOW



AN EXAMPLE – MOBILENETV2 (160X160X3, ALPHA = 1.0)

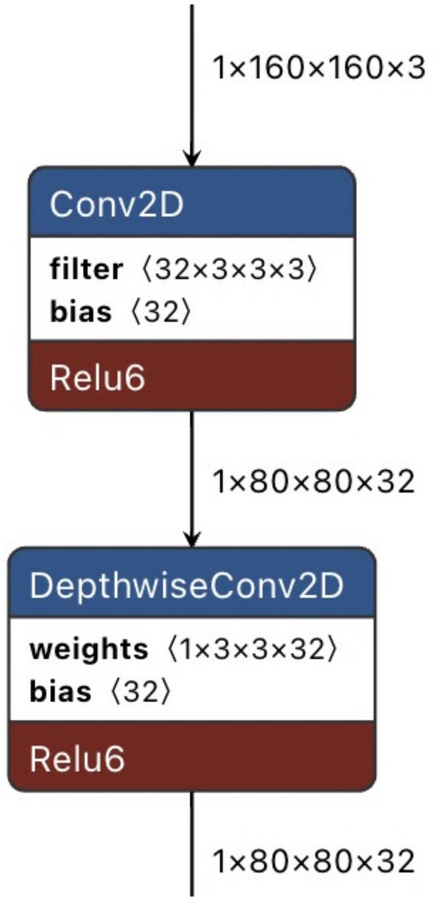
	Unoptimised	Initial optimisations	Arena planning	Operation splitting	Offloading to flash
RAM used (KB)	1385				
Binary size (KB)	2590				

AN EXAMPLE – MOBILENETV2 (160X160X3, ALPHA = 1.0)

	Unoptimised	Initial optimisations	Arena planning	Operation splitting	Offloading to flash
RAM used (KB)	1385	1258			
Binary size (KB)	2590	2312			

MEMORY PLAN ANALYSIS PASS

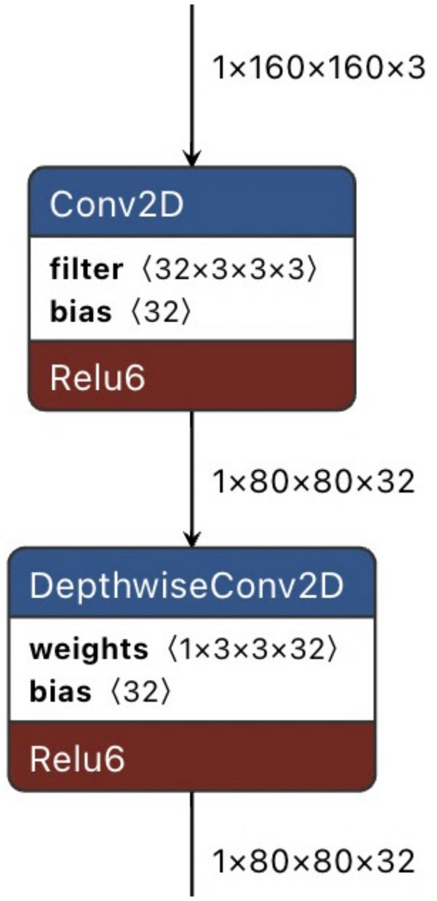
- Plans the tensor arena and allocates offsets for tensors



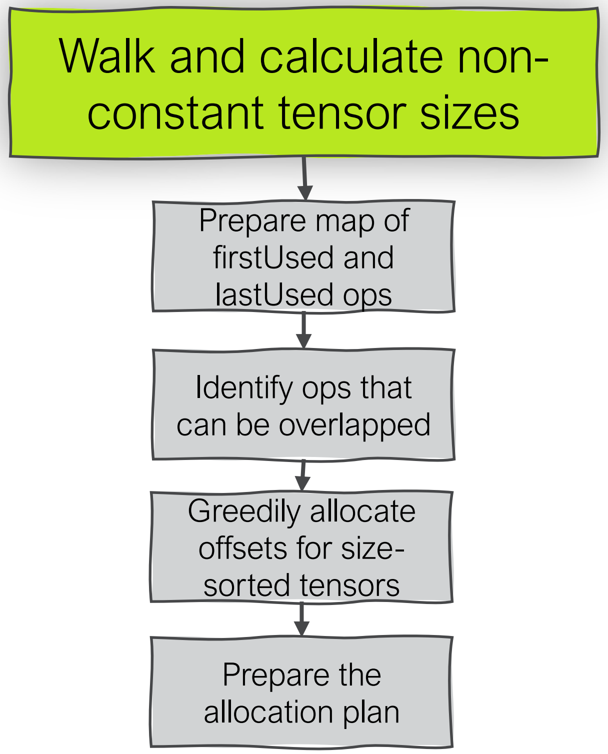
MEMORY PLAN ANALYSIS PASS

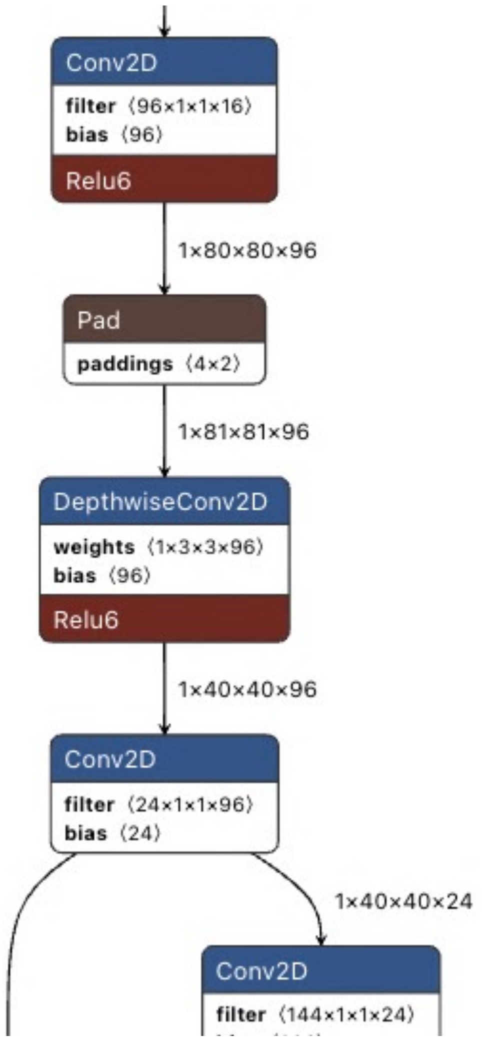
-
  ```
  %2 = "tfl.conv_2d"(%arg0, %0, %1) {dilation_h_factor = 1 : i32,
  dilation_w_factor = 1 : i32, fused_activation_function = "RELU6", padding =
  "SAME", stride_h = 2 : i32, stride_w = 2 : i32} :
  (tensor<?x160x160x3x!quant.uniform<i8:f32, 0.0039215688593685627:-128>>,
  tensor<32x3x3x3x!quant.uniform<i8<-127:127>:f32:0, 0.0052513480186462402>>,
  tensor<32x!quant.uniform<i32:f32:0, 2.0593523004208691E-5>>) ->
  tensor<?x80x80x32x!quant.uniform<i8:f32, 0.023529412224888802:-128>>
  ```

-
  ```
  %5 = "tfl.depthwise_conv_2d"(%2, %3, %4) {depth_multiplier = 1 : i32,
  dilation_h_factor = 1 : i32, dilation_w_factor = 1 : i32,
  fused_activation_function = "RELU6", padding = "SAME", stride_h = 1 : i32,
  stride_w = 1 : i32} : (tensor<?x80x80x32x!quant.uniform<i8:f32,
  0.023529412224888802:-128>>, tensor<1x3x3x32x!quant.uniform<i8<-
  127:127>:f32:3, 0.021009480580687523}>>, tensor<32x!quant.uniform<i32:f32:0,
  4.9434072570875287E-4>>) -> tensor<?x80x80x32x!quant.uniform<i8:f32,
  0.023529412224888802:-128>>
  ```

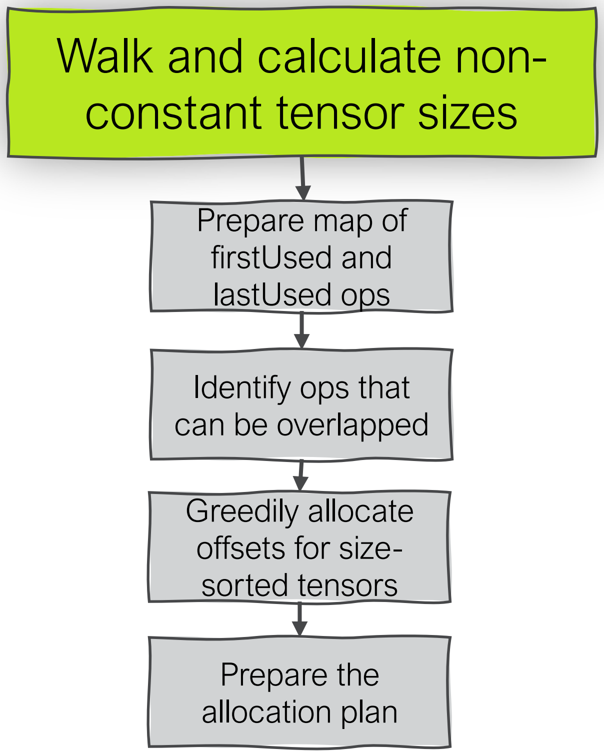


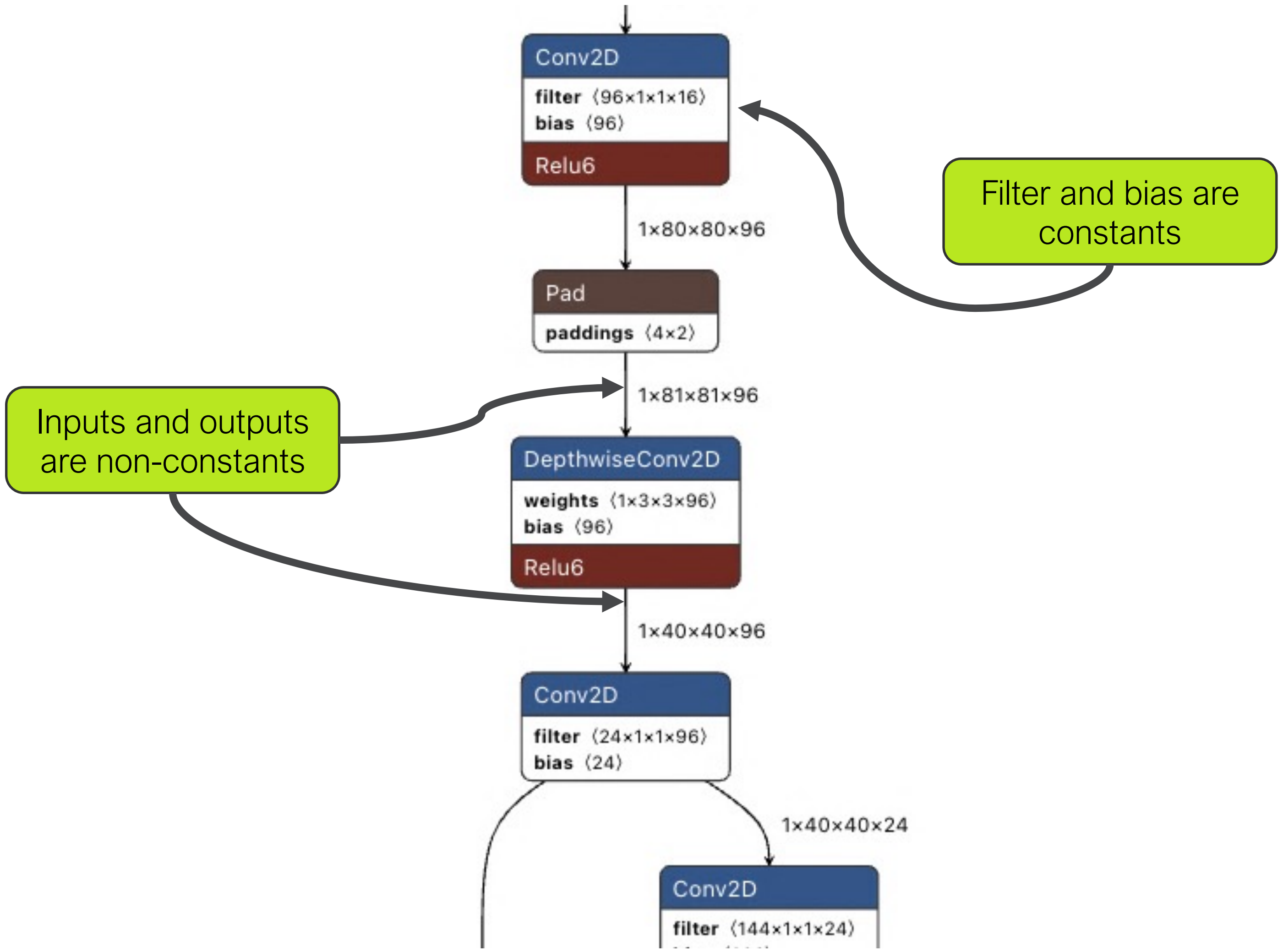
MEMORY PLAN ANALYSIS PASS


Walk and calculate non-constant tensor sizes
Prepare map of firstUsed and lastUsed ops
Identify ops that can be overlapped
Greedily allocate offsets for size-sorted tensors
Prepare the allocation plan

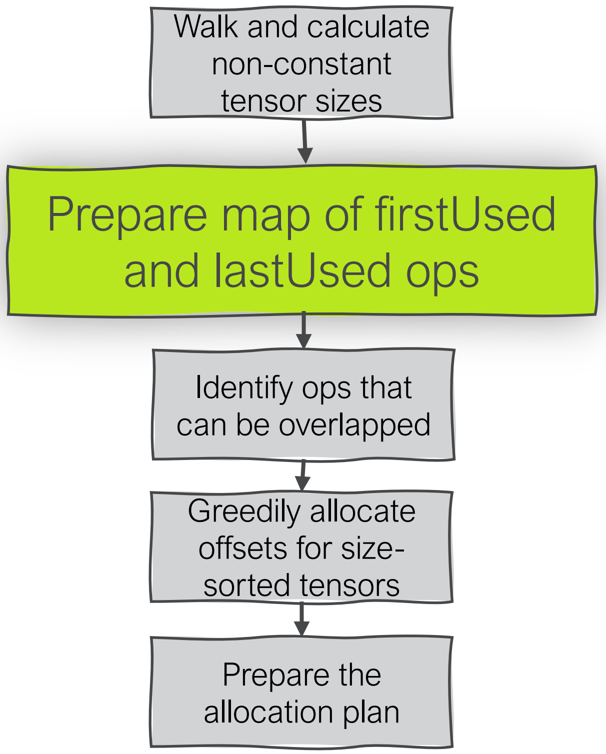


MEMORY PLAN ANALYSIS PASS


Walk and calculate non-constant tensor sizes
Prepare map of firstUsed and lastUsed ops
Identify ops that can be overlapped
Greedily allocate offsets for size-sorted tensors
Prepare the allocation plan

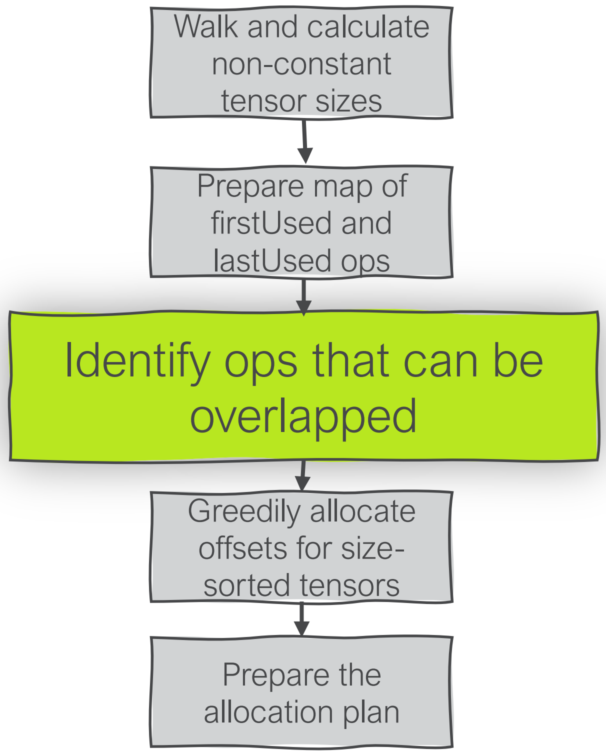


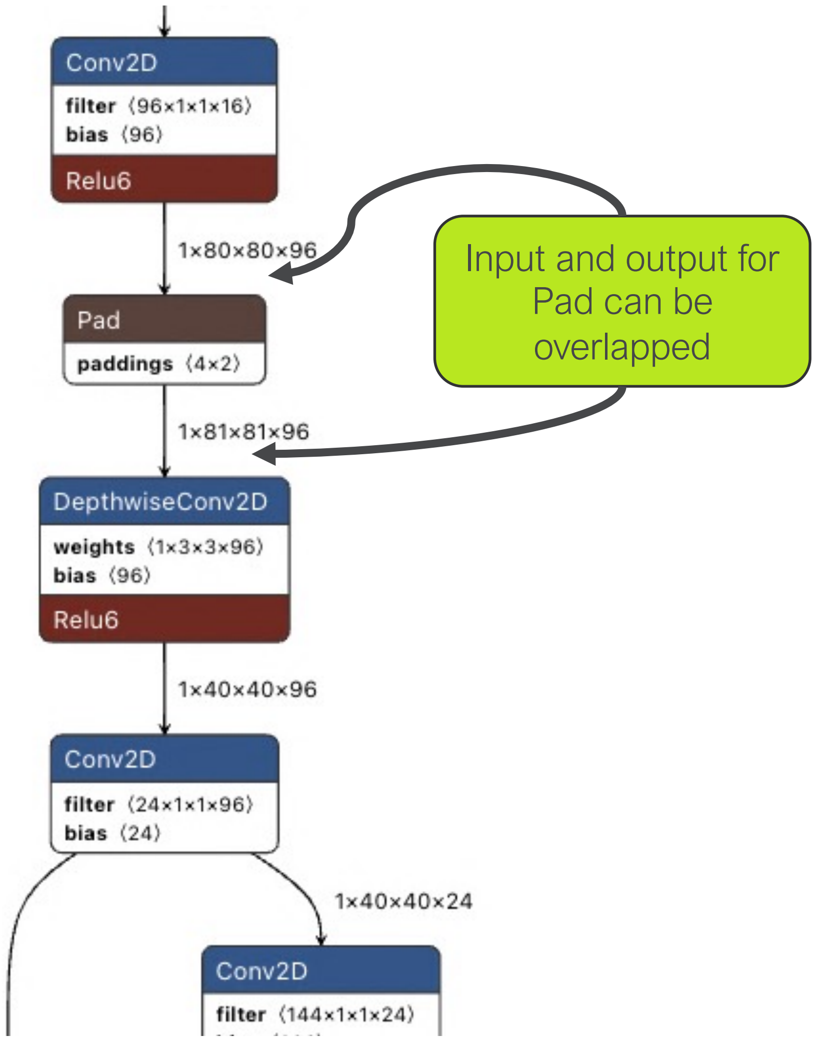
MEMORY PLAN ANALYSIS PASS


Walk and calculate non-constant tensor sizes
Prepare map of firstUsed and lastUsed ops
Identify ops that can be overlapped
Greedily allocate offsets for size-sorted tensors
Prepare the allocation plan

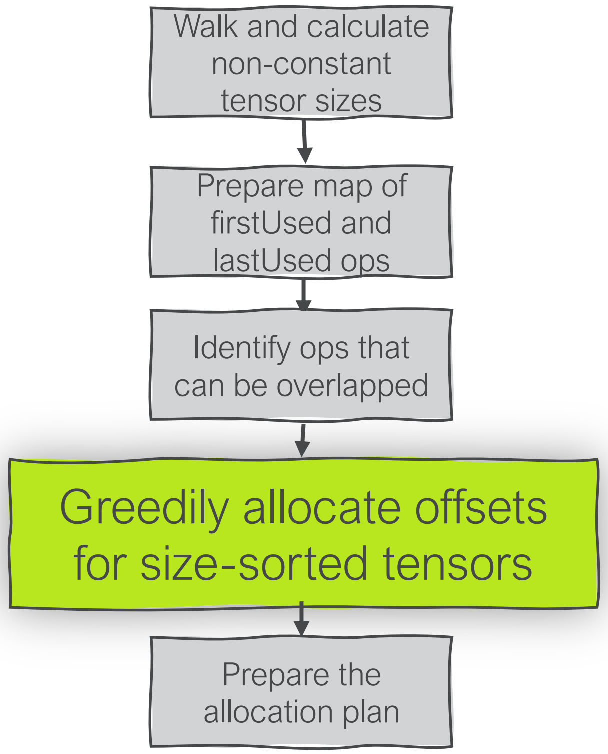
- Use Liveness Analysis pass in MLIR
- This is used to identify simultaneously alive tensors
- The largest simultaneously alive tensors defines the peak memory usage for the graph

MEMORY PLAN ANALYSIS PASS


Walk and calculate non-constant tensor sizes
Prepare map of firstUsed and lastUsed ops
Identify ops that can be overlapped
Greedily allocate offsets for size-sorted tensors
Prepare the allocation plan

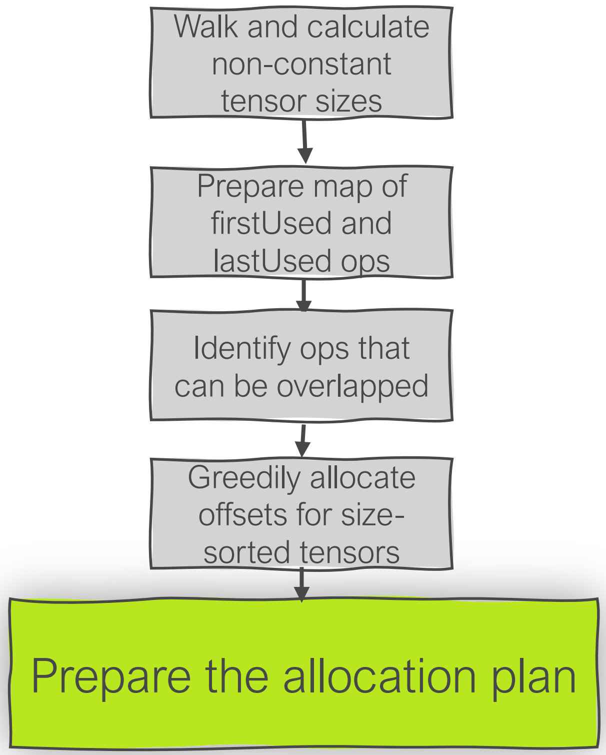


MEMORY PLAN ANALYSIS PASS


Walk and calculate non-constant tensor sizes
Prepare map of firstUsed and lastUsed ops
Identify ops that can be overlapped
Greedily allocate offsets for size-sorted tensors
Prepare the allocation plan

- The allocation algorithm is similar to the one used in Tensorflow Lite for Microcontrollers for arena planning
- Doing it in MLIR gives us much more control
- We want to minimise the total memory used while avoiding fragmentation

MEMORY PLAN ANALYSIS PASS


Walk and calculate non-constant tensor sizes
Prepare map of firstUsed and lastUsed ops
Identify ops that can be overlapped
Greedily allocate offsets for size-sorted tensors
Prepare the allocation plan

- The allocation plan is an array of offsets, one for every tensor in the model, written to flatbuffer metadata
- -1 is used for constant tensors
- 0,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1, -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1 204800,-1,-1,0,-1,-1,204800,-1,-1,629856, 629856,827136,629856,800256,629856,-1,-1

AN EXAMPLE – MOBILENETV2 (160X160X3, ALPHA = 1.0)

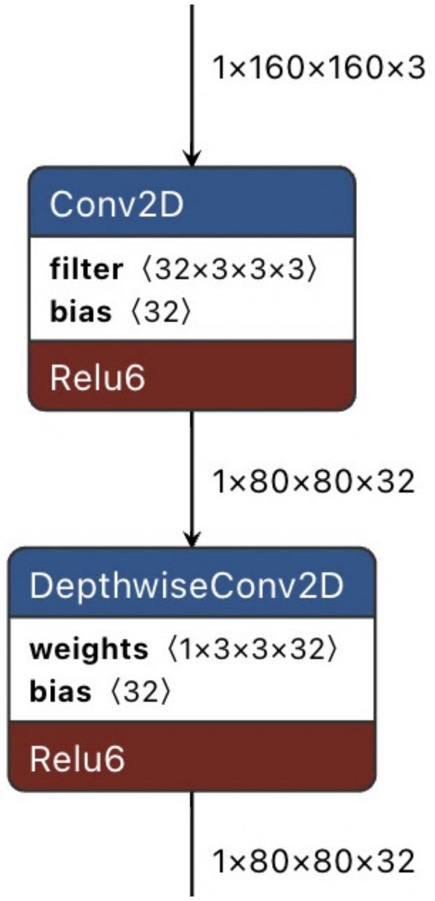
	Unoptimised	Initial optimisations	Arena planning	Operation splitting	Offloading to flash
RAM used (KB)	1385	1258			
Binary size (KB)	2590	2312			

AN EXAMPLE – MOBILENETV2 (160X160X3, ALPHA = 1.0)

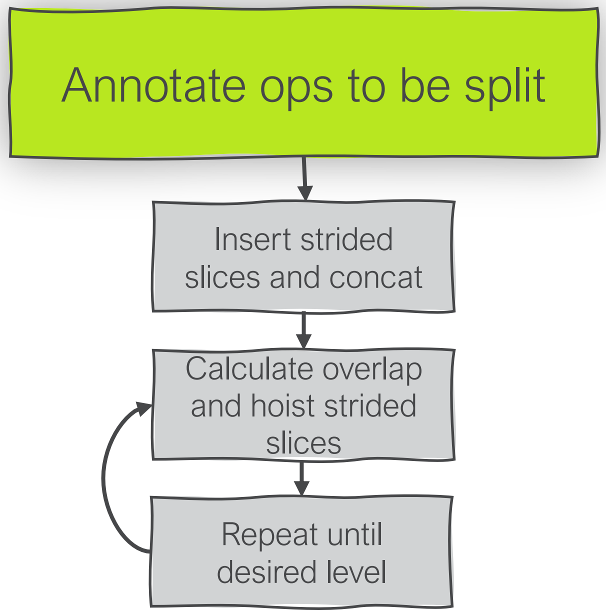
	Unoptimised	Initial optimisations	Arena planning	Operation splitting	Offloading to flash
RAM used (KB)	1385	1258	798		
Binary size (KB)	2590	2312	2312		

OPERATION SPLIT PASS

- Splits operations into multiple ops to reduce peak memory usage

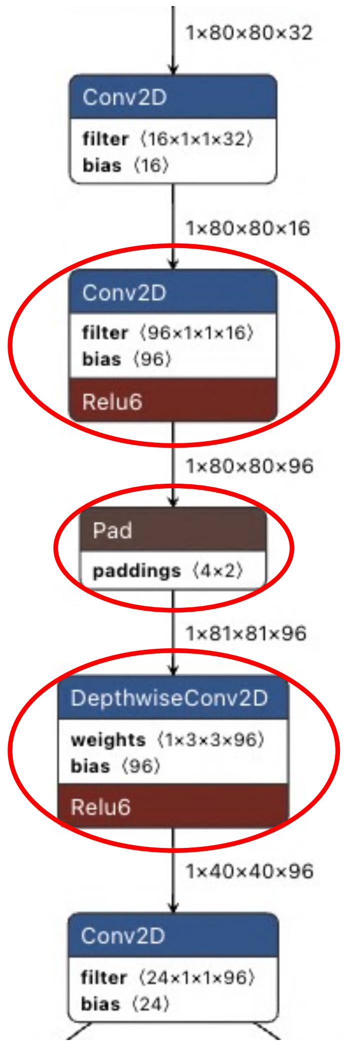


OPERATION SPLIT PASS

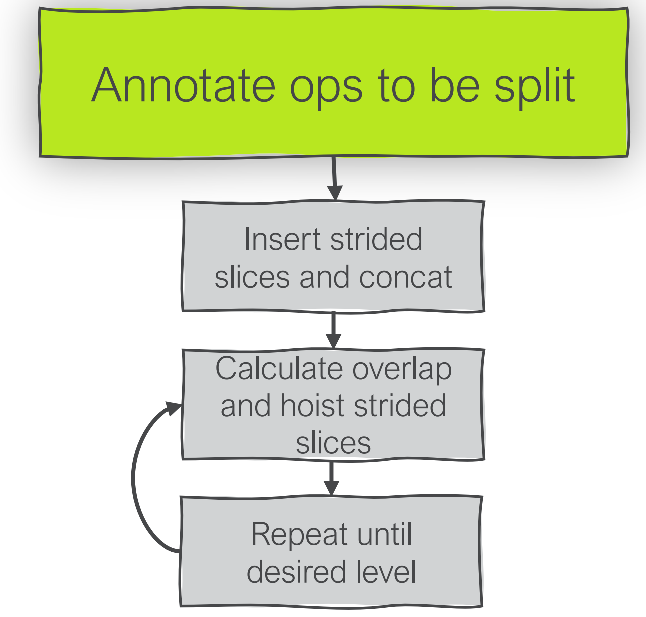


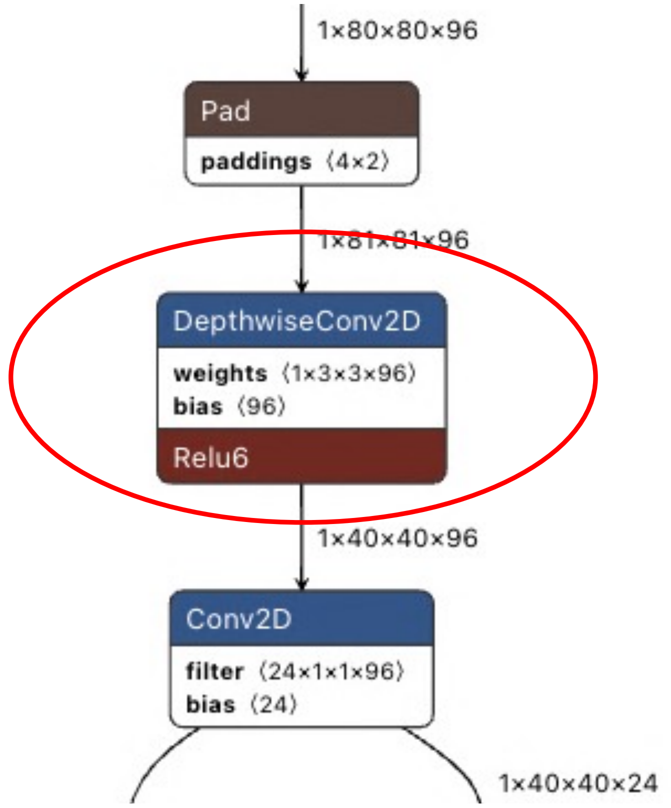
Annotate ops to be split
Insert strided slices and concat
Calculate overlap and hoist strided slices
Repeat until desired level

- Annotation can be specified by command-line options
- We also have partially working auto-annotation based on Liveness Analysis pass and peak memory usage

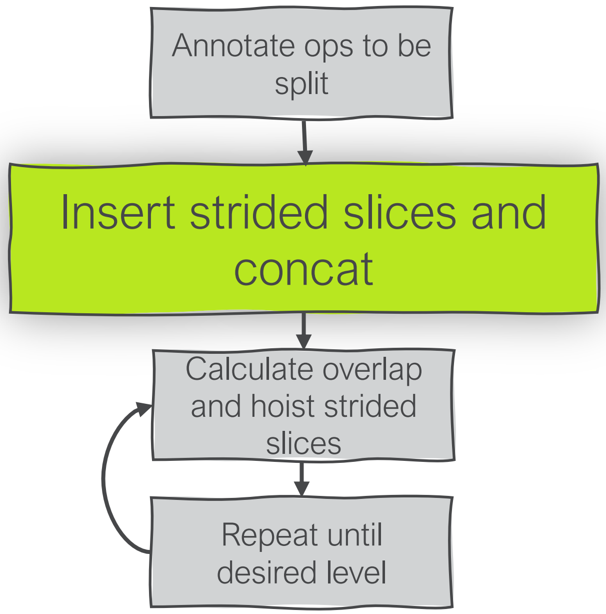


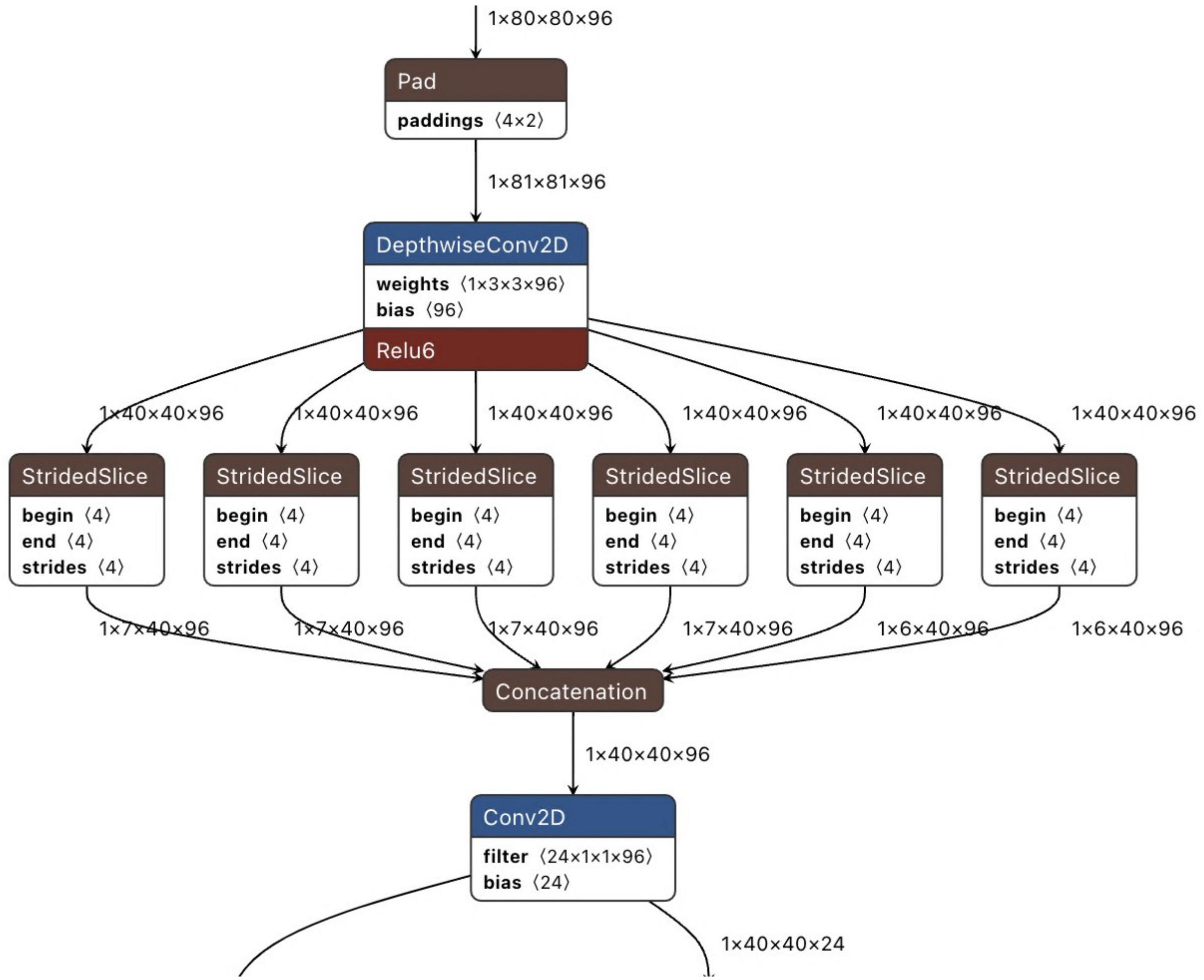
OPERATION SPLIT PASS


Annotate ops to be split
Insert strided slices and concat
Calculate overlap and hoist strided slices
Repeat until desired level

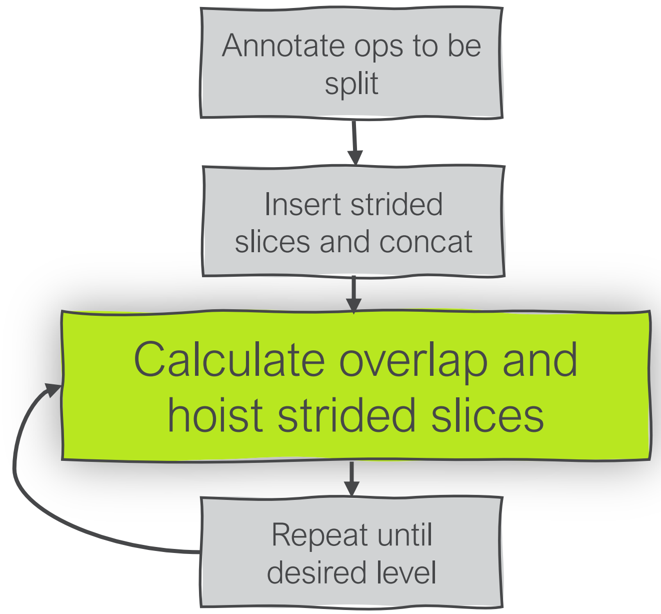


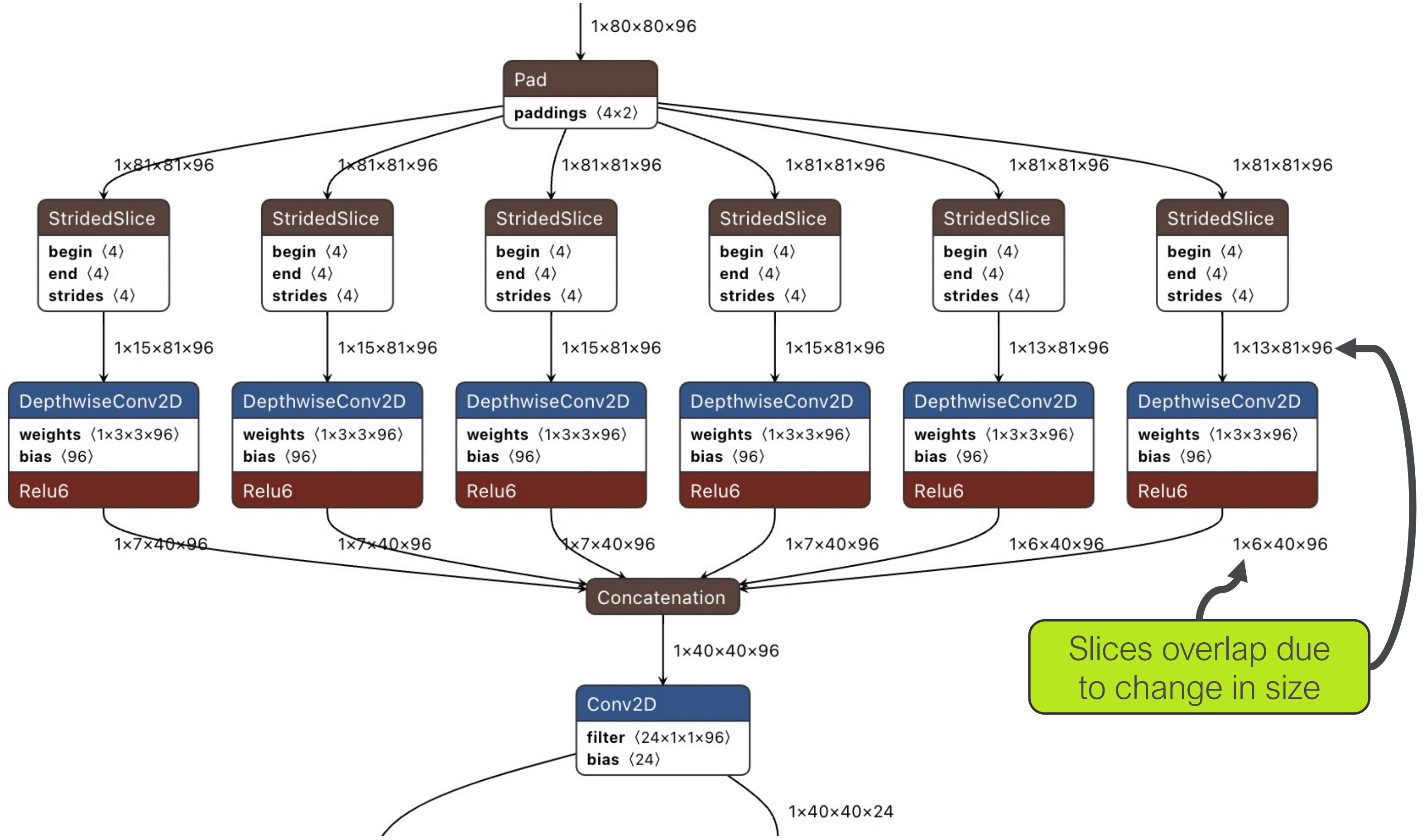
OPERATION SPLIT PASS


Annotate ops to be split
Insert strided slices and concat
Calculate overlap and hoist strided slices
Repeat until desired level

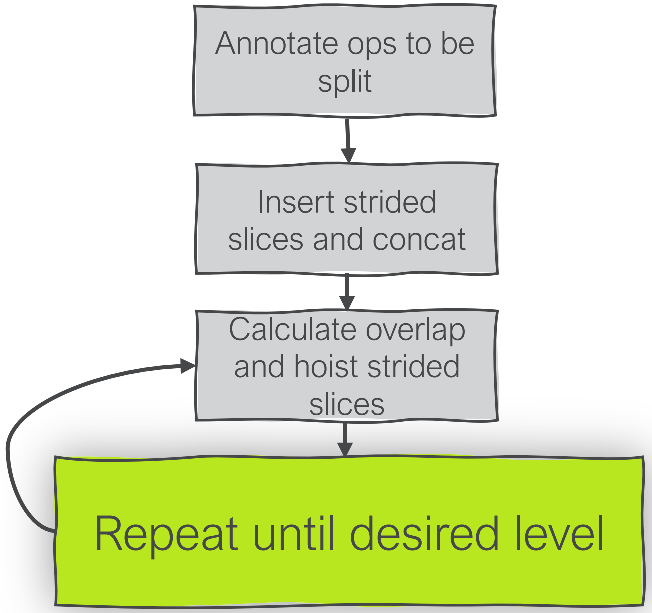


OPERATION SPLIT PASS

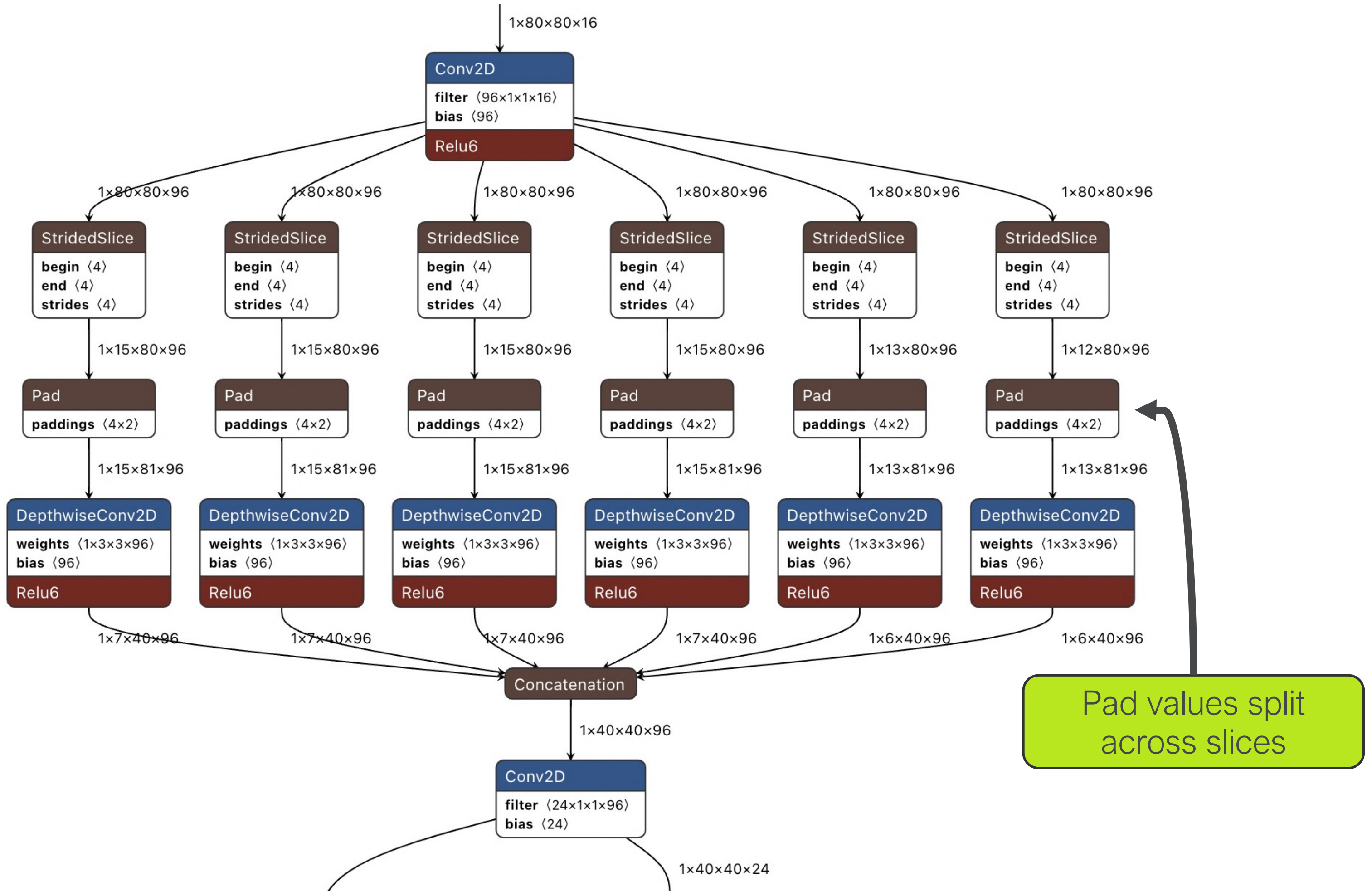

Annotate ops to be split
Insert strided slices and concat
Calculate overlap and hoist strided slices
Repeat until desired level



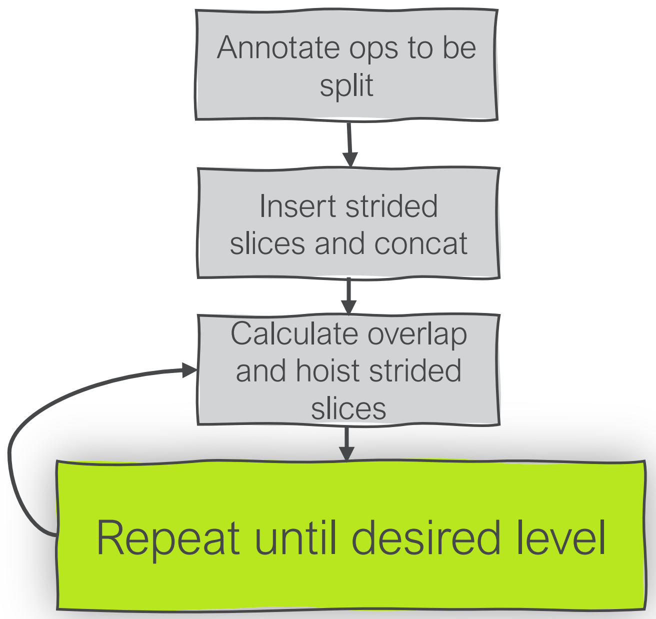
OPERATION SPLIT PASS



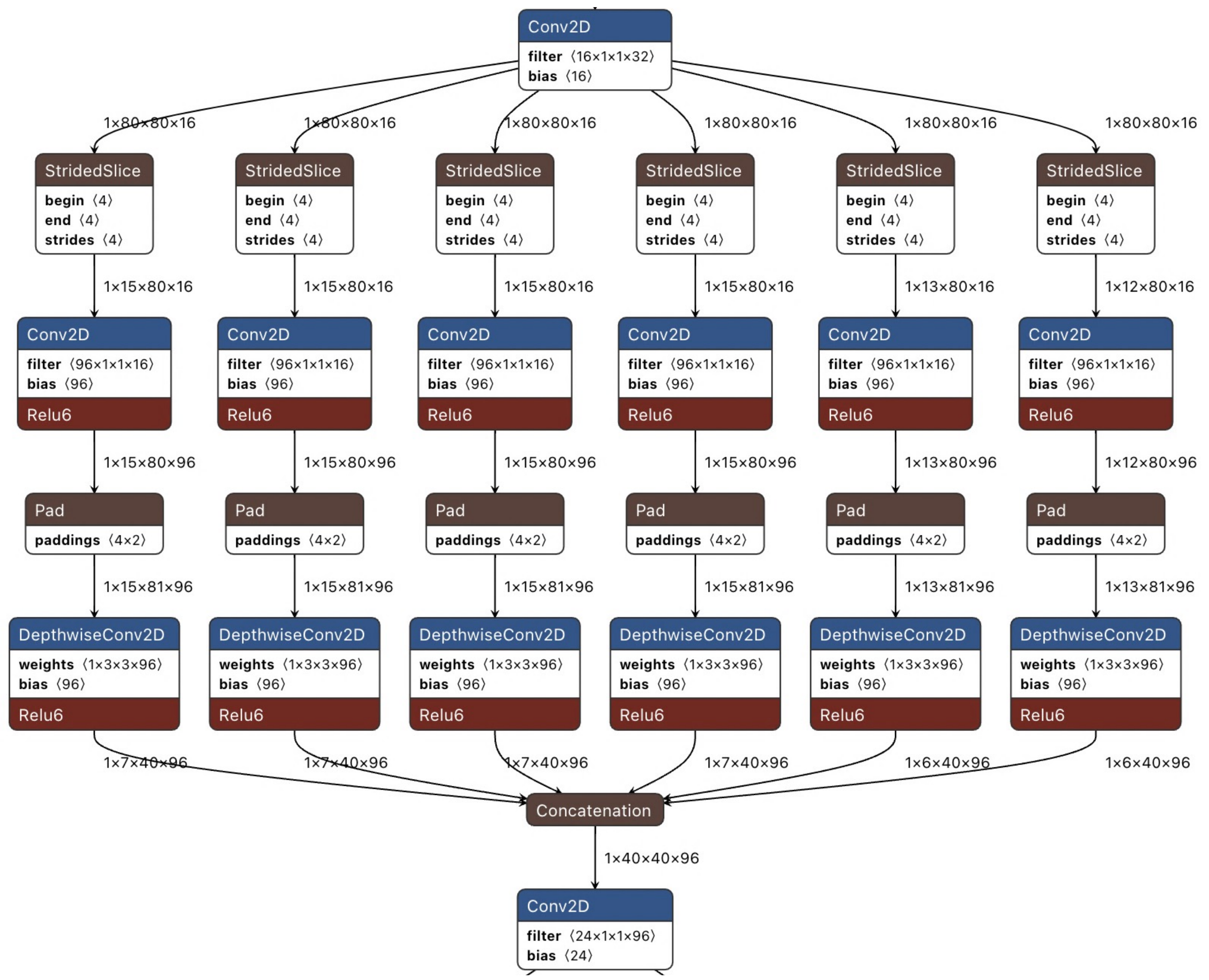
Annotate ops to be split
Insert strided slices and concat
Calculate overlap and hoist strided slices
Repeat until desired level



OPERATION SPLIT PASS



Annotate ops to be split
Insert strided slices and concat
Calculate overlap and hoist strided slices
Repeat until desired level



AN EXAMPLE – MOBILENETV2 (160X160X3, ALPHA = 1.0)

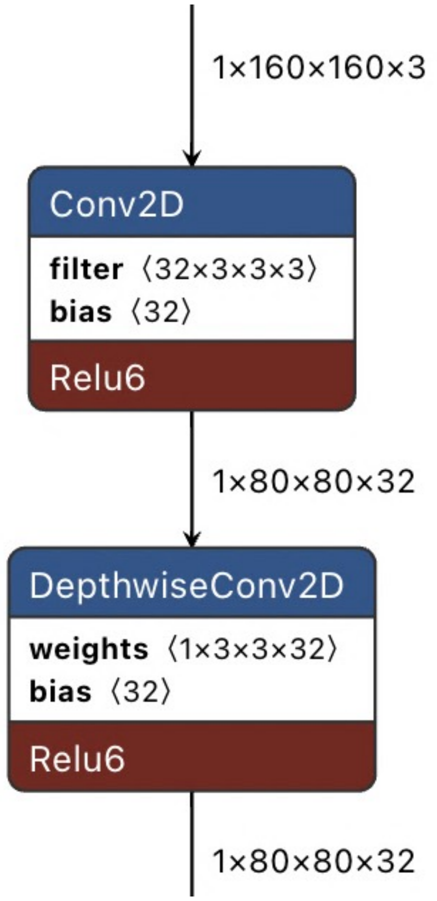
	Unoptimised	Initial optimisations	Arena planning	Operation splitting	Offloading to flash
RAM used (KB)	1385	1258	798		
Binary size (KB)	2590	2312	2312		

AN EXAMPLE – MOBILENETV2 (160X160X3, ALPHA = 1.0)

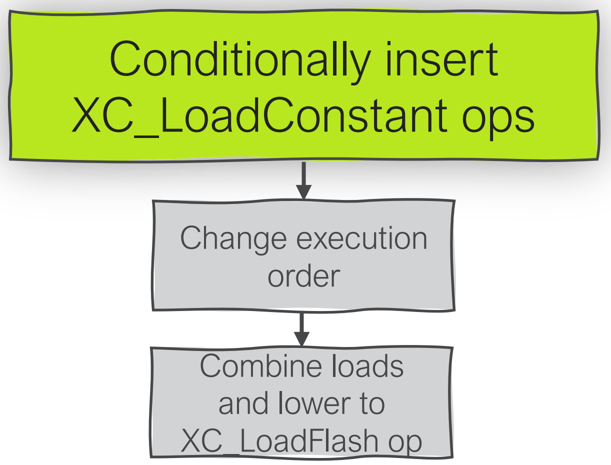
	Unoptimised	Initial optimisations	Arena planning	Operation splitting	Offloading to flash
RAM used (KB)	1385	1258	798	363	
Binary size (KB)	2590	2312	2312	2337	

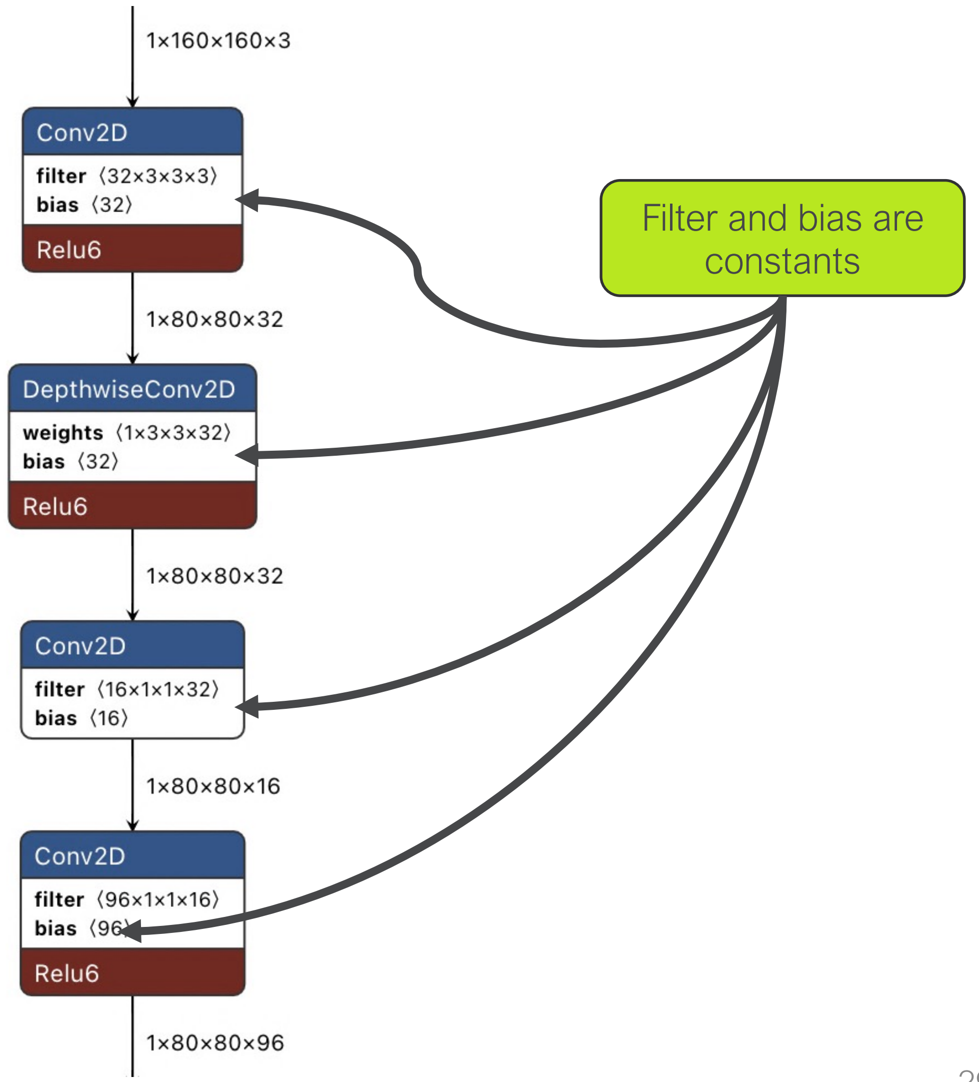
FLASH IMAGE PASS

- Offload weights in the model to a flash image which can then be streamed in at runtime



FLASH IMAGE PASS


Conditionally insert XC_LoadConstant ops
Change execution order
Combine loads and lower to XC_LoadFlash op

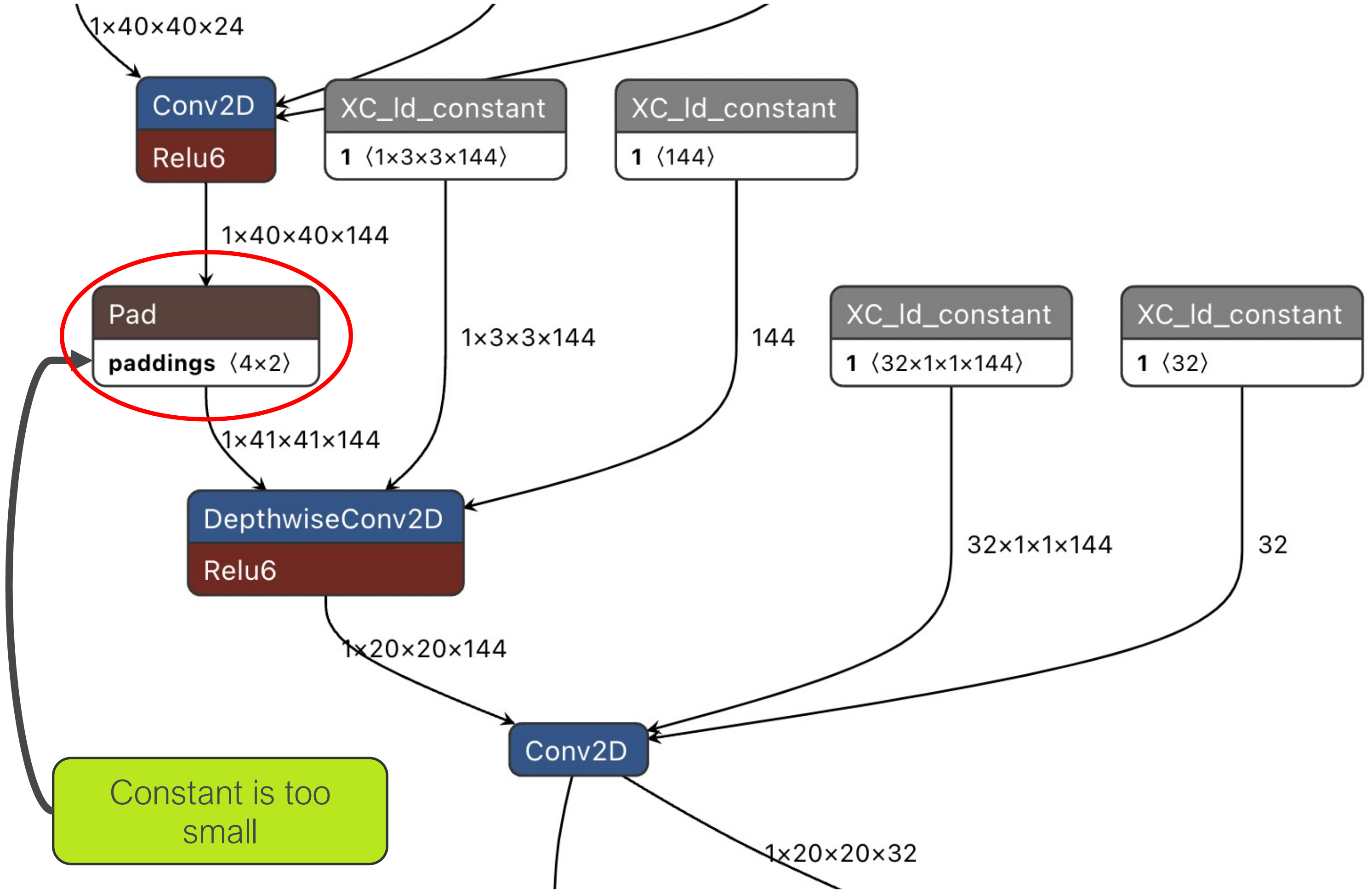


FLASH IMAGE PASS

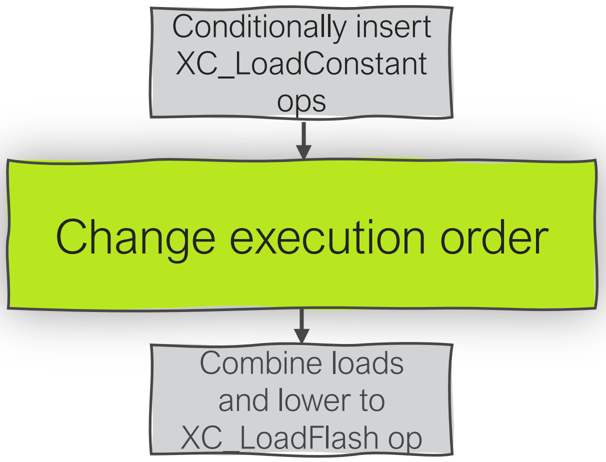
Conditionally insert XC_LoadConstant ops

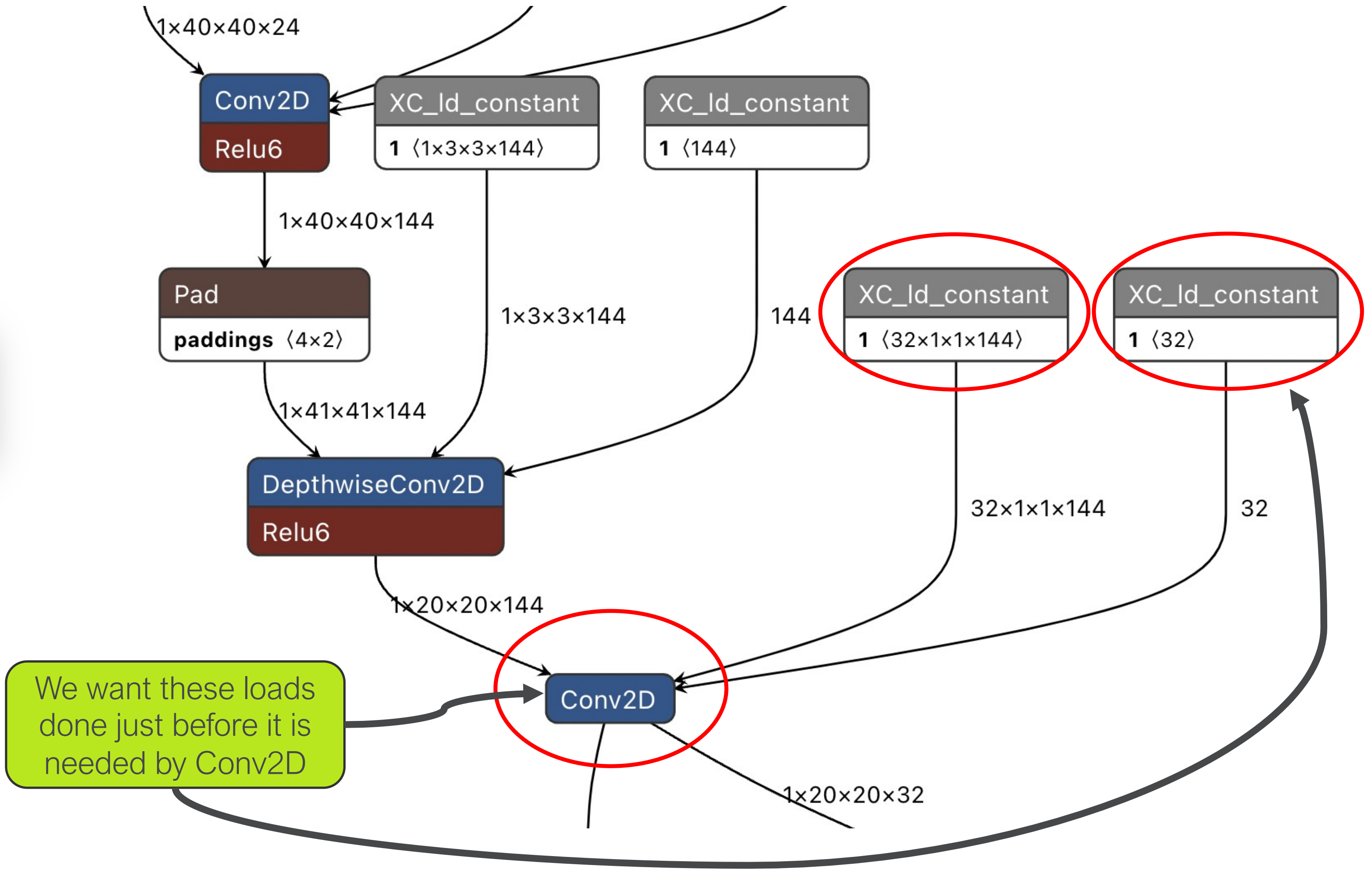
Change execution order

Combine loads and lower to XC_LoadFlash op

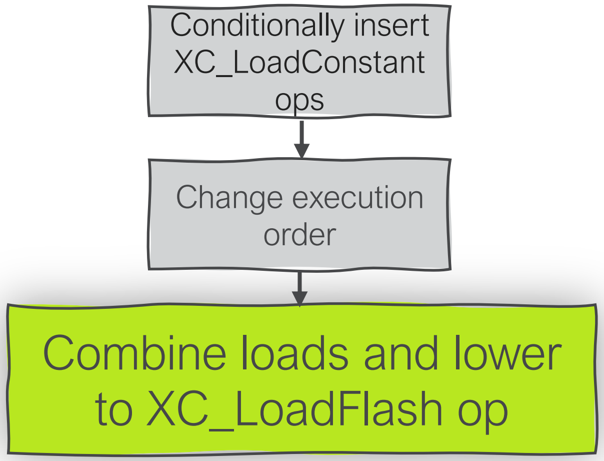


FLASH IMAGE PASS

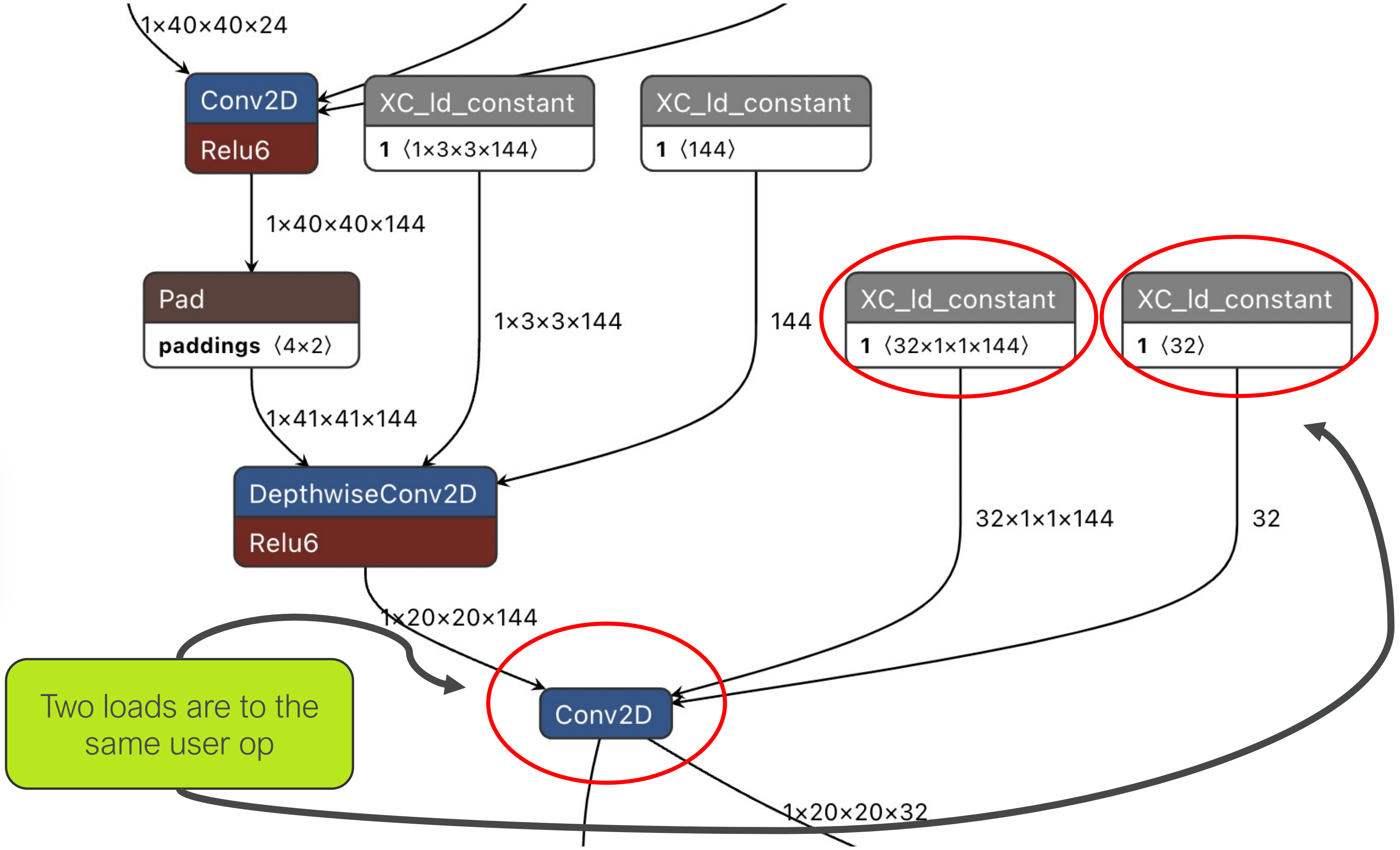

Conditionally insert XC_LoadConstant ops
Change execution order
Combine loads and lower to XC_LoadFlash op



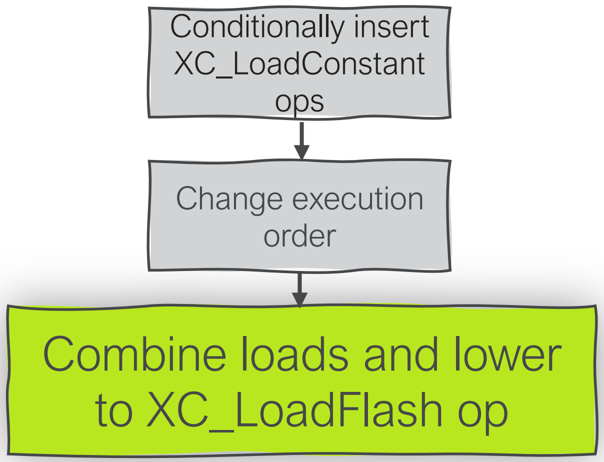
FLASH IMAGE PASS

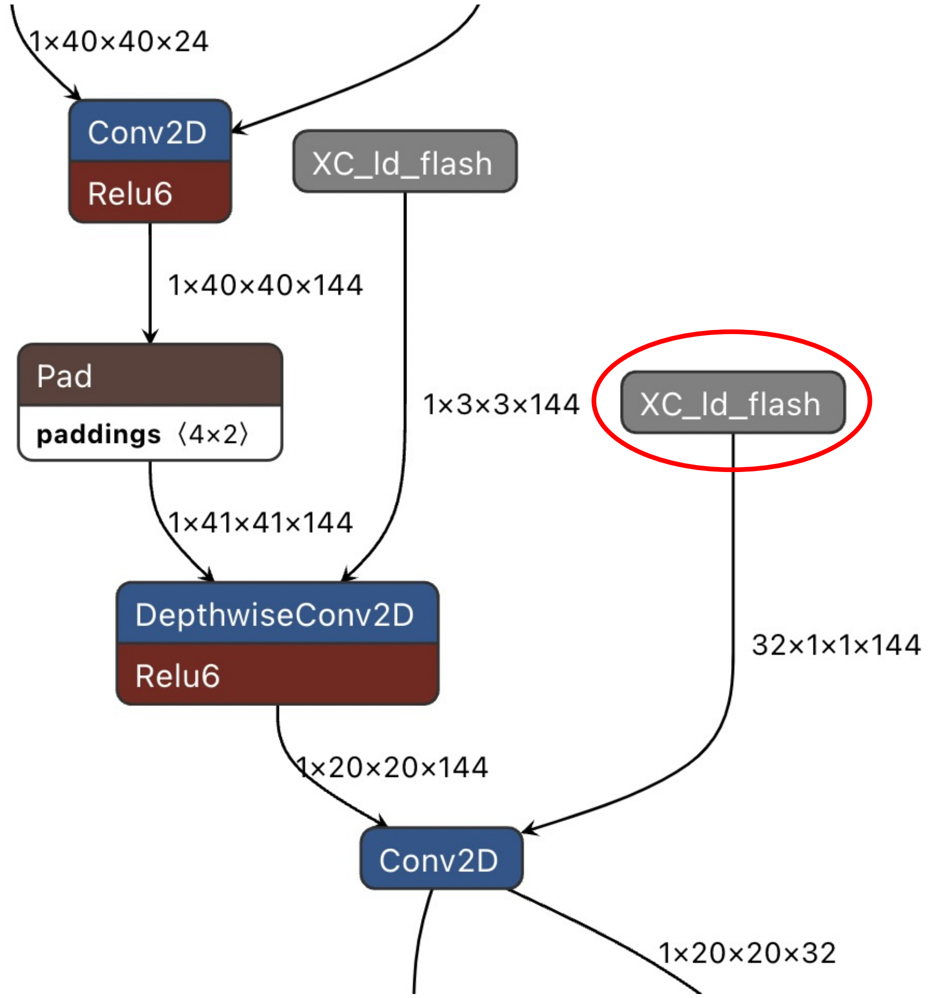


Conditionally insert XC_LoadConstant ops
Change execution order
Combine loads and lower to XC_LoadFlash op



FLASH IMAGE PASS


Conditionally insert XC_LoadConstant ops
Change execution order
Combine loads and lower to XC_LoadFlash op



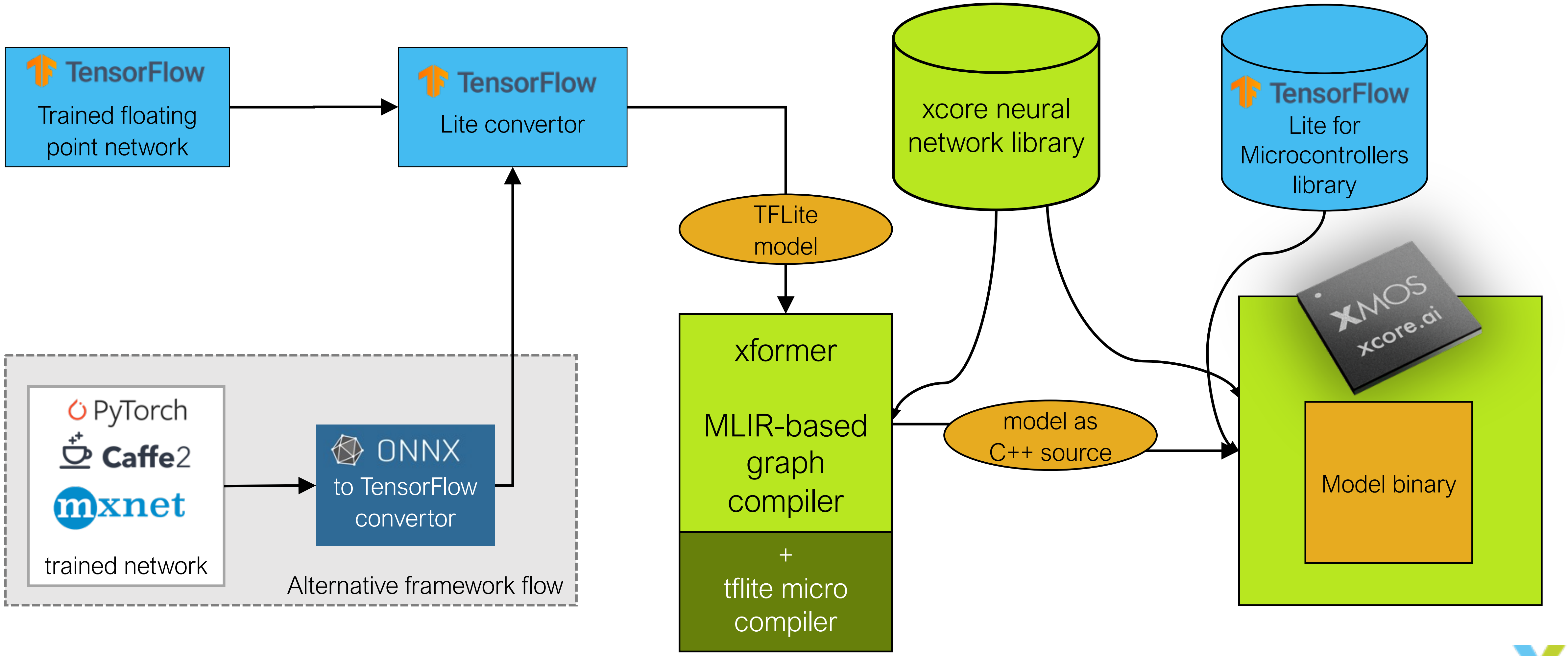
AN EXAMPLE – MOBILENETV2 (160X160X3, ALPHA = 1.0)

	Unoptimised	Initial optimisations	Arena planning	Operation splitting	Offloading to flash
RAM used (KB)	1385	1258	798	363	
Binary size (KB)	2590	2312	2312	2337	

AN EXAMPLE – MOBILENETV2 (160X160X3, ALPHA = 1.0)

	Unoptimised	Initial optimisations	Arena planning	Operation splitting	Offloading to flash
RAM used (KB)	1385	1258	798	363	384
Binary size (KB)	2590	2312	2312	2337	488

CHALLENGES AND FUTURE PLANS



CHALLENGES AND FUTURE PLANS

Challenges

Identify prior art to reuse

Find what is the “correct” way, what idioms to use

Future plans

Adapt memory plan analysis to add page in/out ops for handling larger models

Better execution order

EXPERIENCE WITH MLIR

The MLIR framework made it easy for us to quickly add optimisations and productise our AI tools

We reuse a lot of code from TensorFlow and the MLIR project

Being able to work at the right level of abstraction is intuitive

- Thank you!

- deepakpanickal@xmos.com

- All code is available publicly at
- https://github.com/xmos/ai_tools
- https://github.com/xmos/lib_tflite_micro