µIR – An intermediate representation for transforming and optimizing the microarchitecture of application accelerators

Amirali Sharifian[1], Reza Hojabr[1], Navid Rahimi[1],
Sihao Liu[2], Apala Guha[1]
Tony Nowatzki[2] and Arrvindh Shriraman[1]

https://github.com/sfu-arch/uir

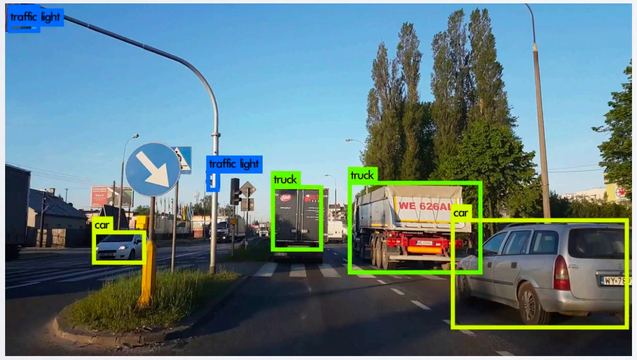
Simon Fraser University[1], UCLA[2]

SFU

UCLA Computer Science

The Accelerator Flowchart

Study the Application

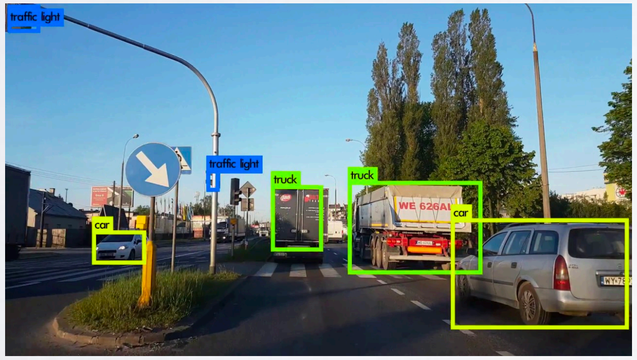


Object Detection Model

The Accelerator Flowchart

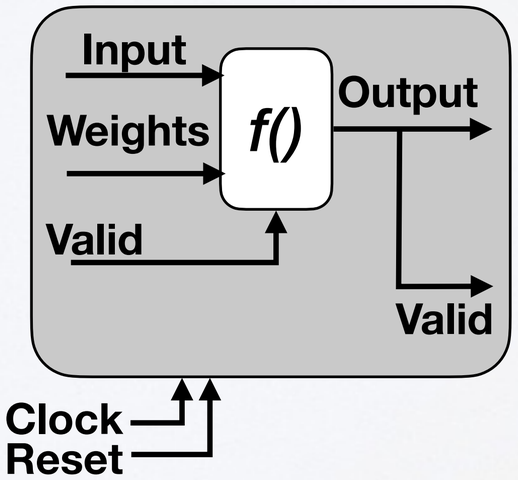
Study the Application

Design Hardware



Object Detection Model

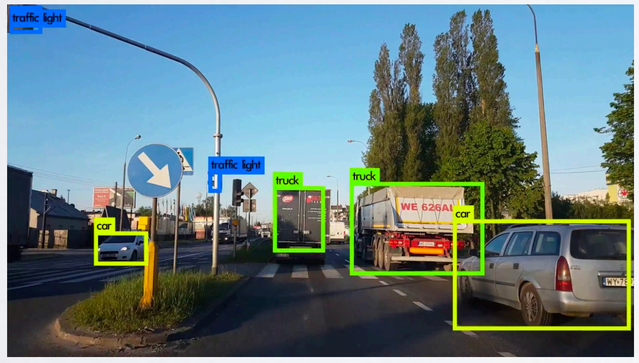
RTL Design



The Accelerator Flowchart

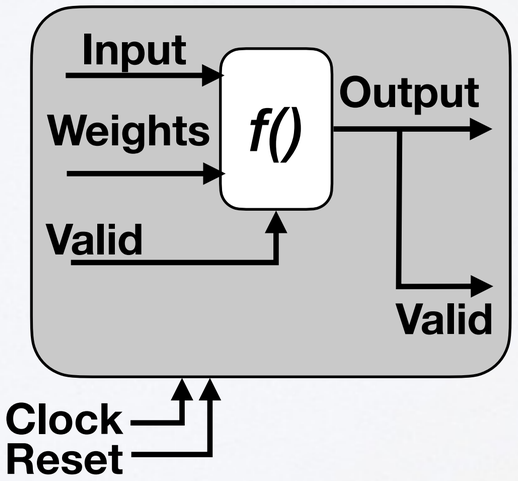
Study the Application → Design Hardware → New Software

Compiler



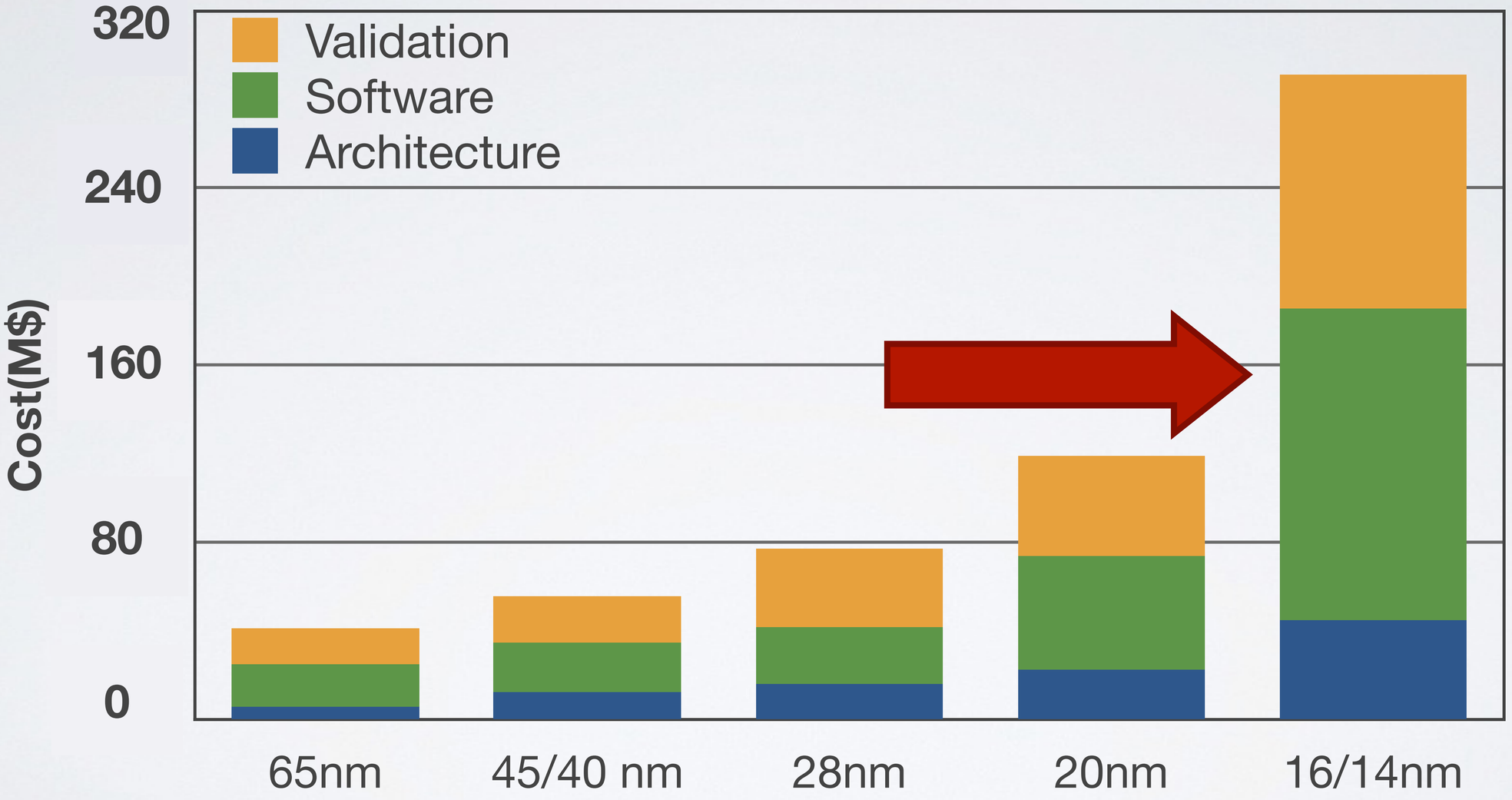
Object Detection Model

RTL Design



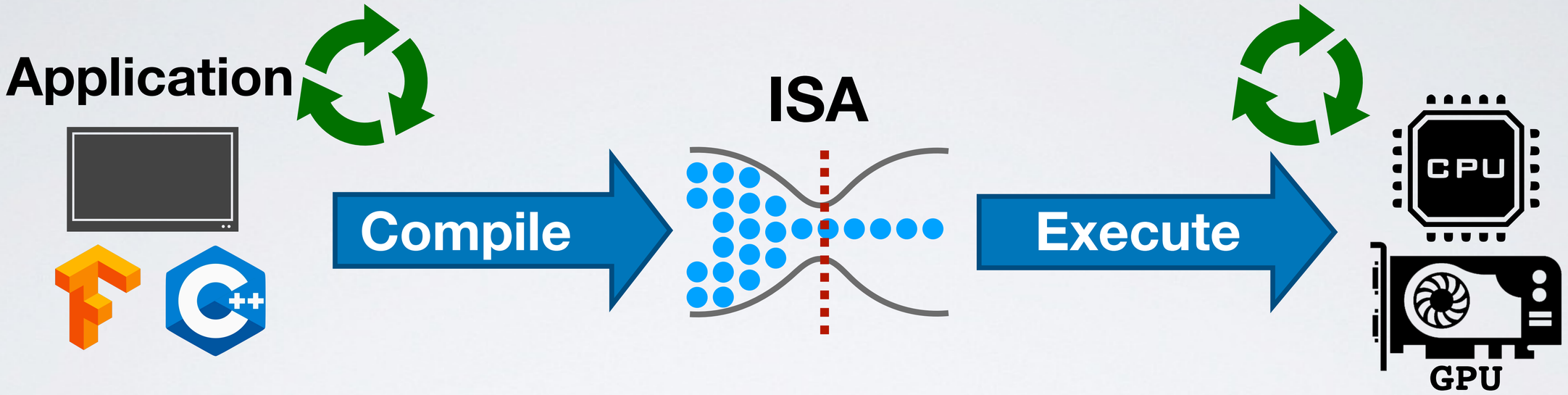

```
40
41  # A map of names to methods that help
42  MODEL_BUILD_UTIL_MAP = {
43      'get_configs_from_pipeline_file':
44          config_util.get_configs_from_
45      'create_pipeline_proto_from_confi
46          config_util.create_pipeline_p
```

Problems With Design Flowchart



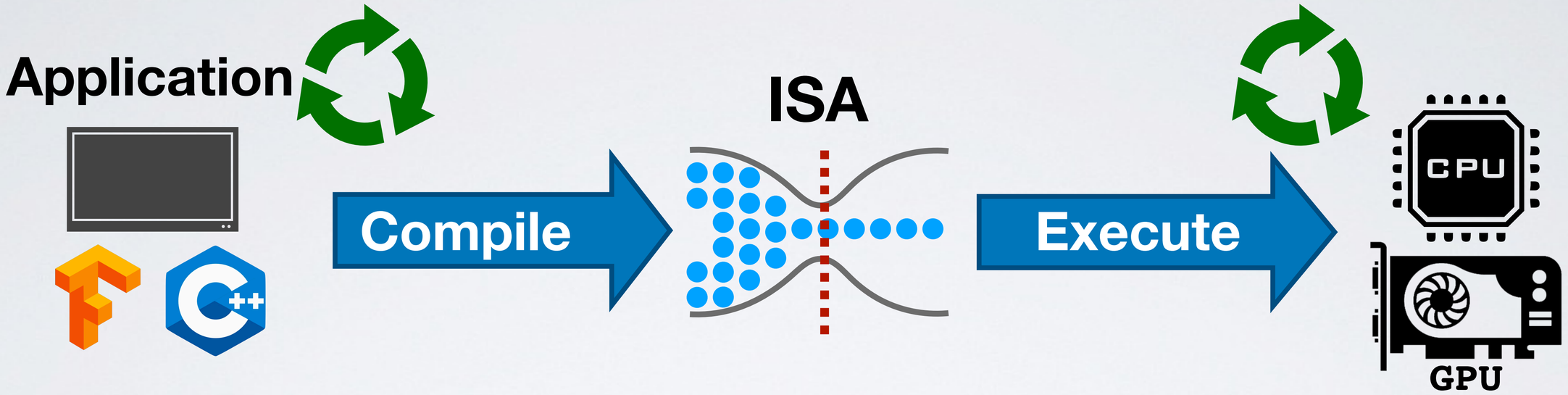
Feature Dimension (Transistor Count) — Source: IBS

ISA-based Flowchart



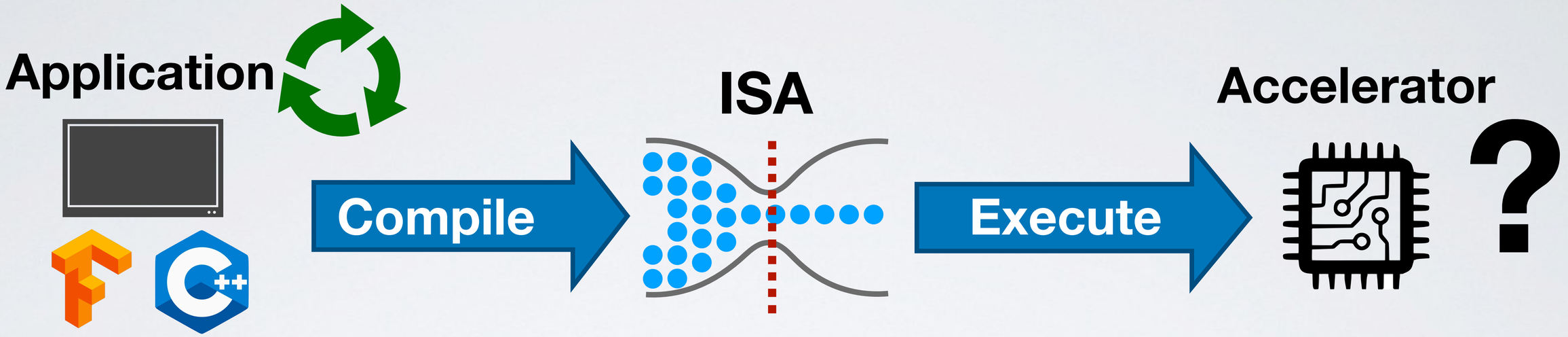
✓ Isolate Application from Architecture

ISA-based Flowchart



✓ Isolate Application from Architecture

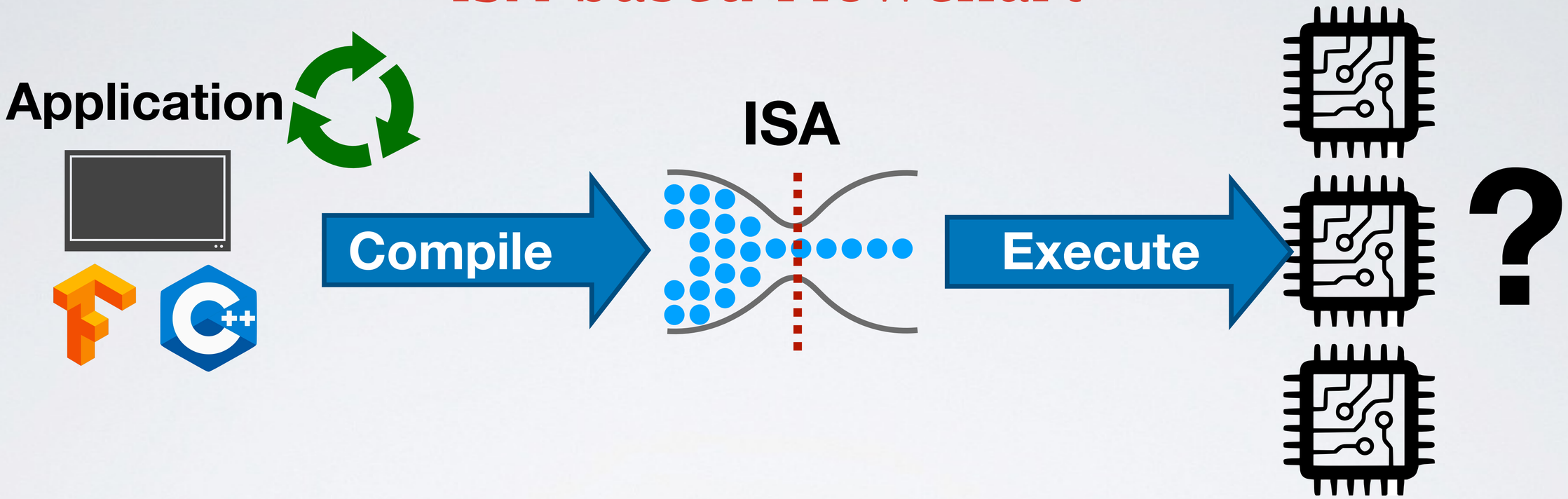
ISA-based Flowchart



✓ Isolate Application from Architecture

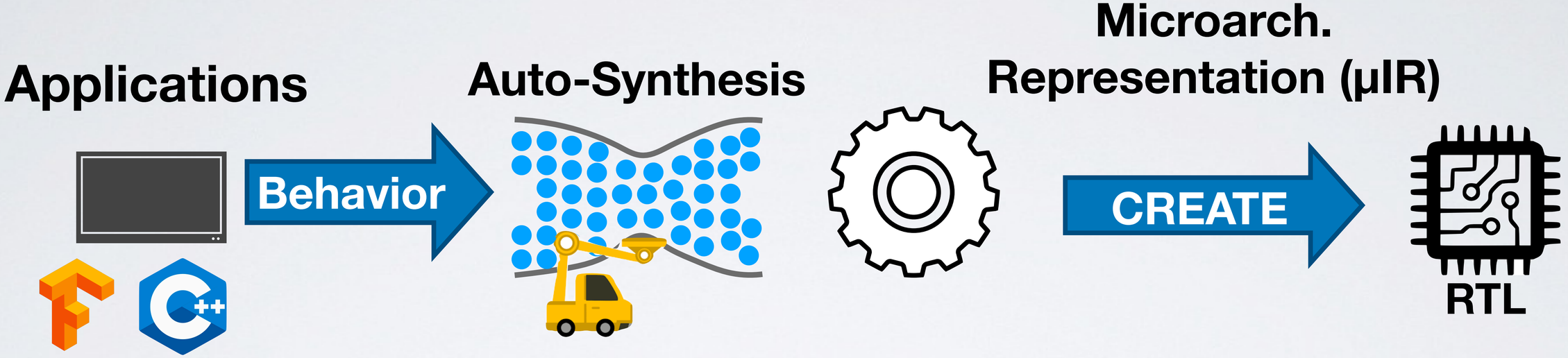
▸ Not expressive enough to create hardware

ISA-based Flowchart



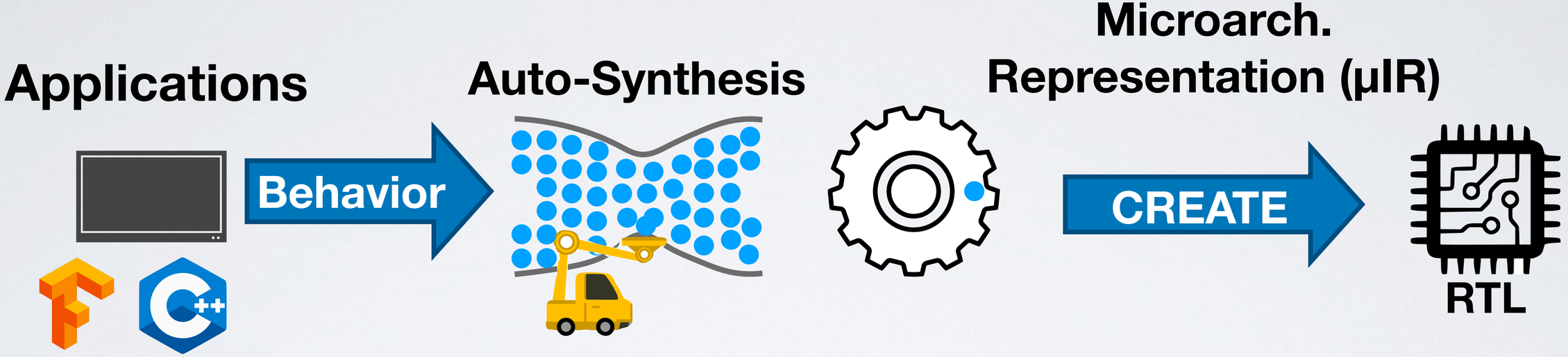
- ✓ Isolate Application from Architecture
- Not expressive enough to create hardware
- Not precise enough to explore hardware

µIR — A New Accelerator Flowchart



✓ End-to-End flow — Existing software for behavior/functionality

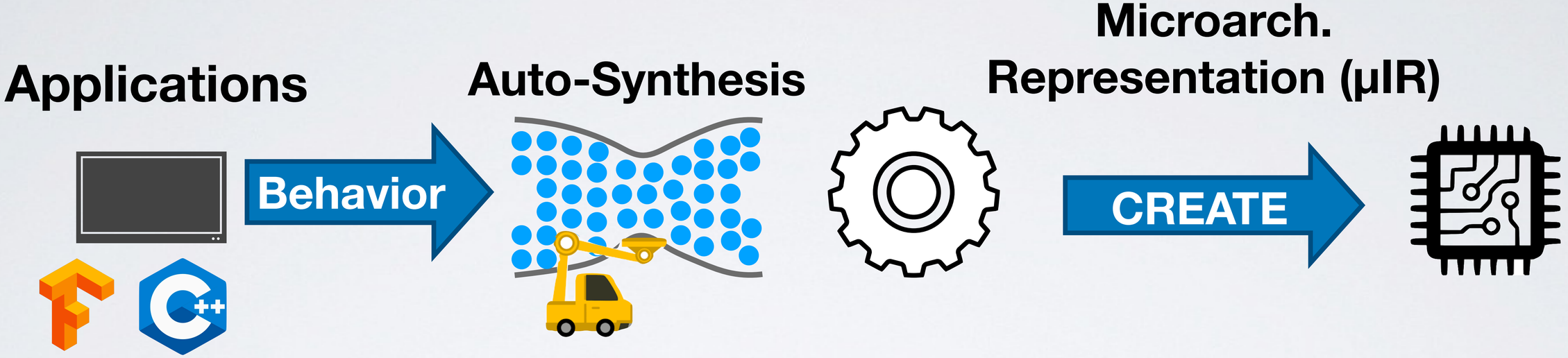
µIR — A New Accelerator Flowchart



✓ End-to-End flow — Existing software for behavior/functionality

✓ Reduce effort — Compiler to extract behaviour

µIR — A New Accelerator Flowchart

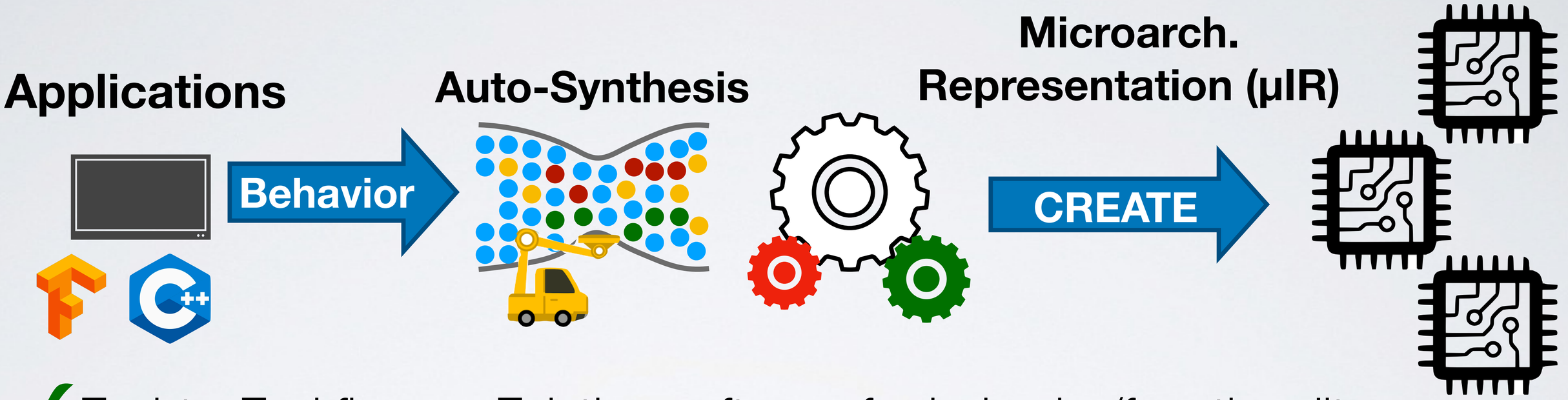


✓ End-to-End flow — Existing software for behavior/functionality

✓ Reduce effort — Compiler to extract behaviour

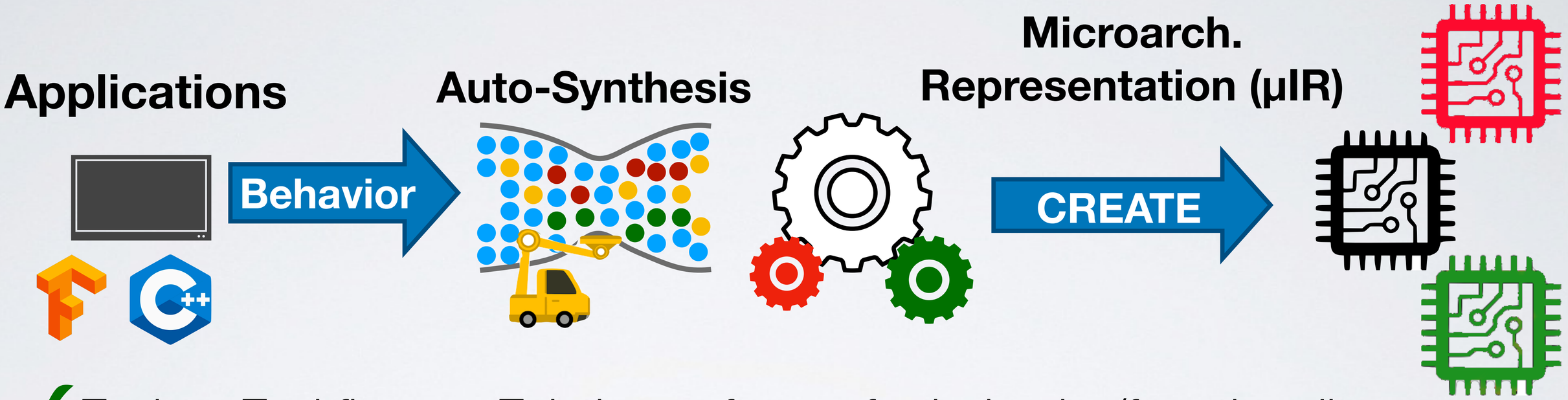
✓ Design exploration — New model for exploring architectures

µIR — A New Accelerator Flowchart

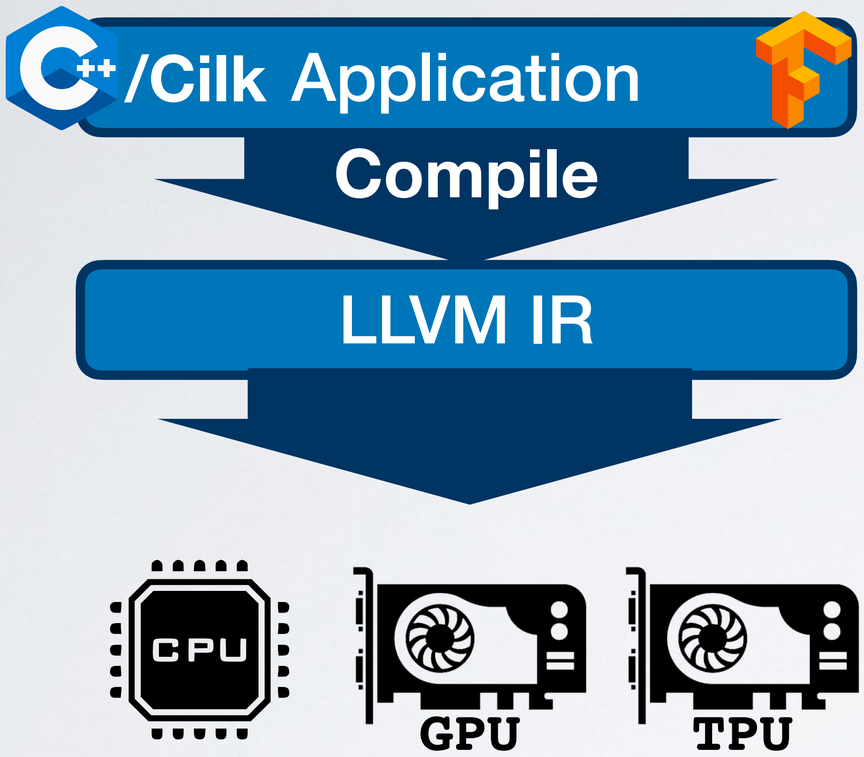


- ✓ End-to-End flow — Existing software for behavior/functionality
- ✓ Reduce effort — Compiler to extract behaviour
- ✓ Design exploration — New model for exploring architectures
- ✓ Extensibility — Extensible to capture domain information

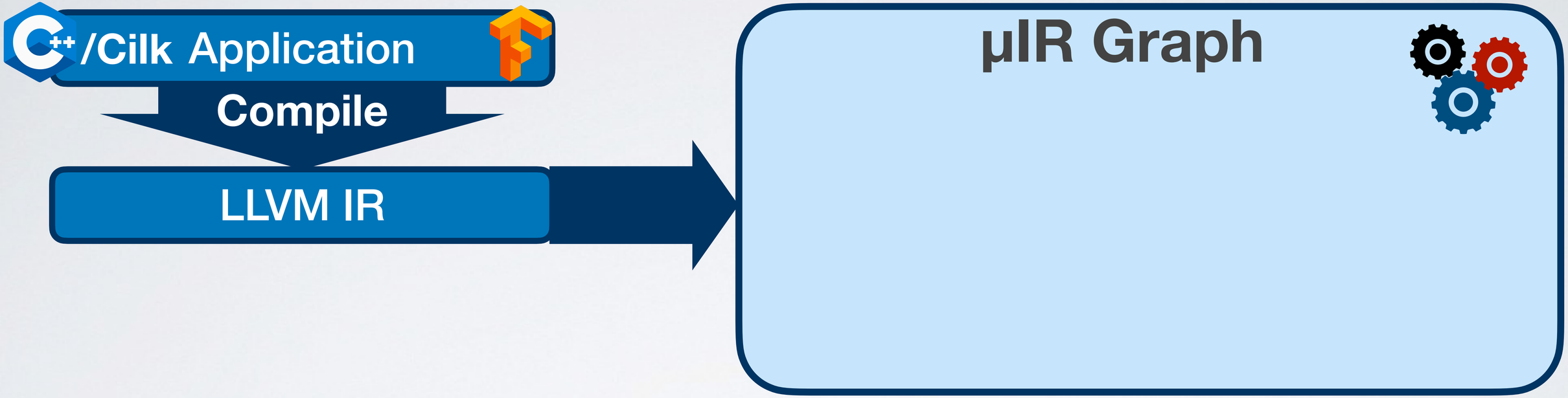
µIR — A New Accelerator Flowchart



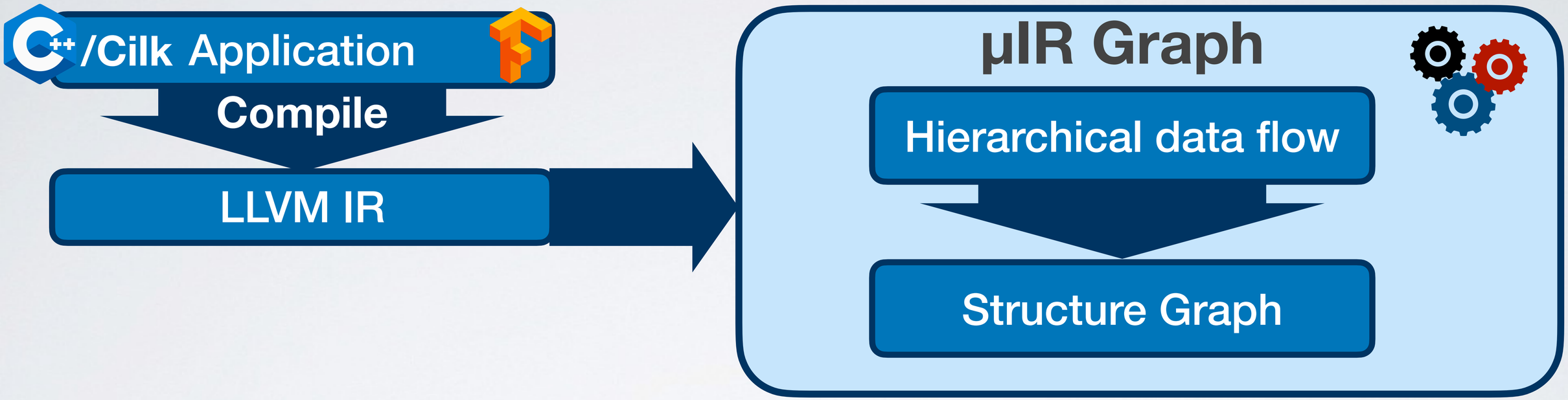
- ✓ End-to-End flow — Existing software for behavior/functionality
- ✓ Reduce effort — Compiler to extract behaviour
- ✓ Design exploration — New model for exploring architectures
- ✓ Extensibility — Extensible to capture domain information


/Cilk Application
Compile
LLVM IR
CPU
GPU
TPU

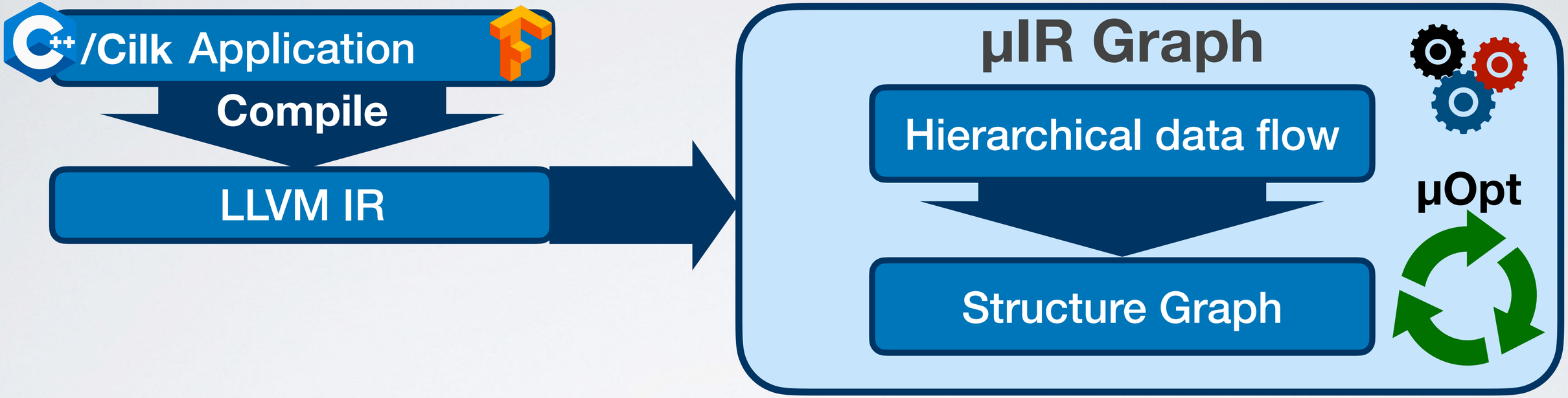
µIR A New Accelerator Flowchart



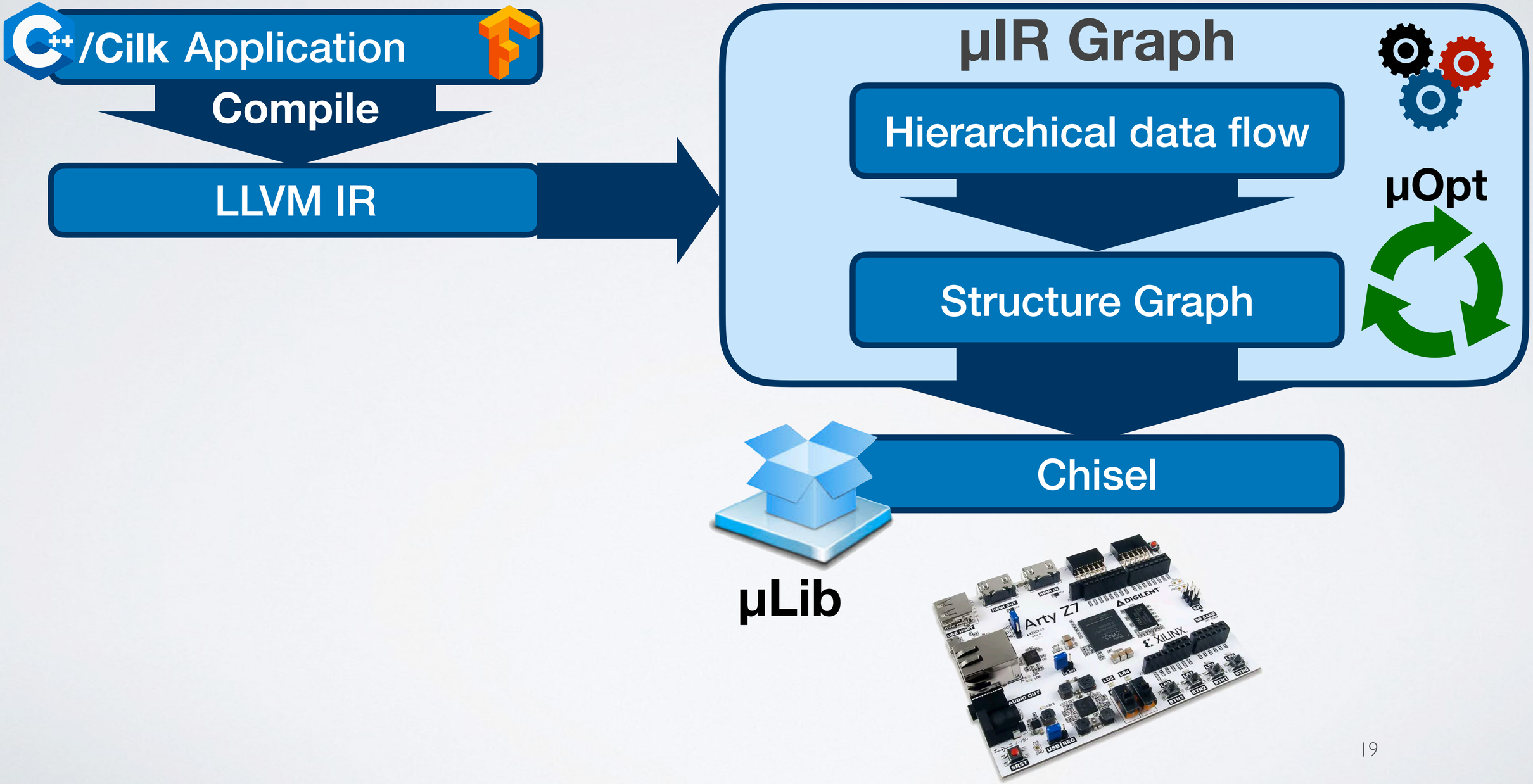
µIR A New Accelerator Flowchart



µIR A New Accelerator Flowchart

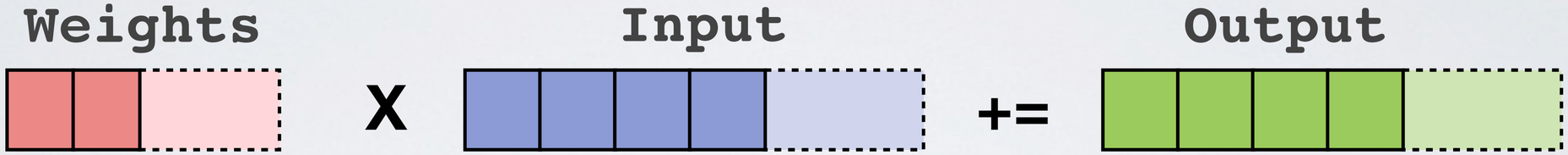


µIR A New Accelerator Flowchart



- Motivation
- **µIR behaviour graph**
- µIR structural graph
- Evaluation
- Summary

μIR Behavioural Graph (1/3)



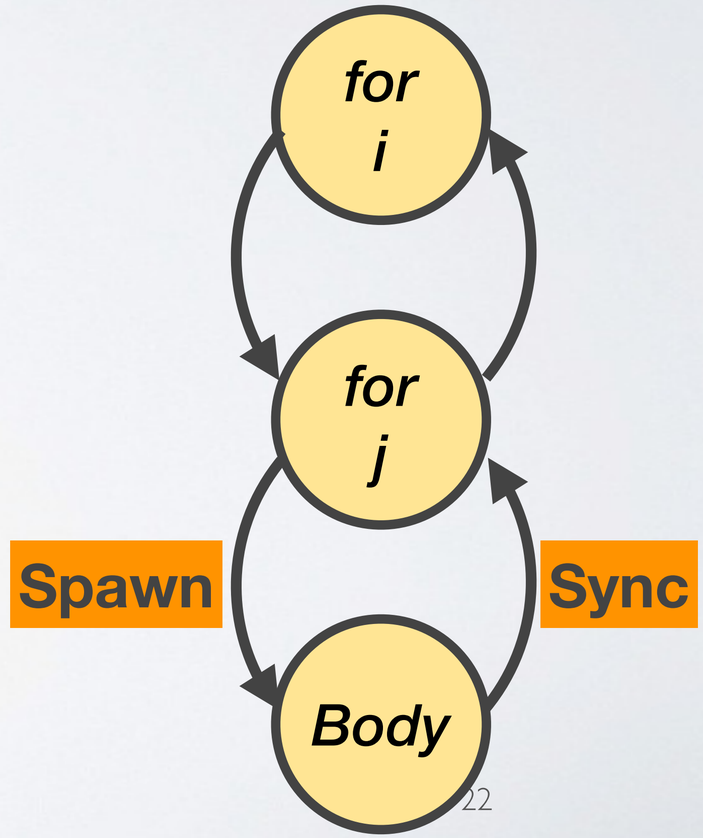

```
parallel_for(i=0; I<(M-W); i++)
 parallel_for(j=0; j<W; j++)
    output[i] += input[i+j] * weight[j];
```

$$Conv = \sum_{i=0}^{N-W} \sum_{j=0}^{W} W_ij * IN_ij$$

µIR Behavioural Graph (2/3)

```
parallel_for(i=0; I<(M-W); i++)
 parallel_for(j=0; j<W; j++)
    output[i] += input[i+j] * weight[j];
```

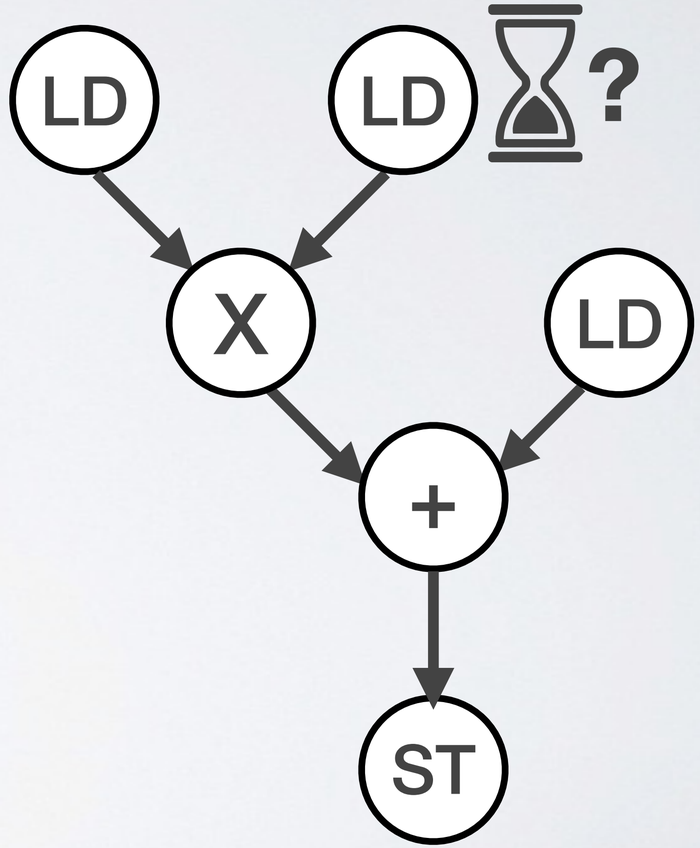
- **Hierarchical and heterogeneous task graph**
  - Decompose the input program to task blocks.
  - Graph of task blocks
    - Implement arbitrary heterogenous parallel and serial



µIR Behavioural Graph (3/3)

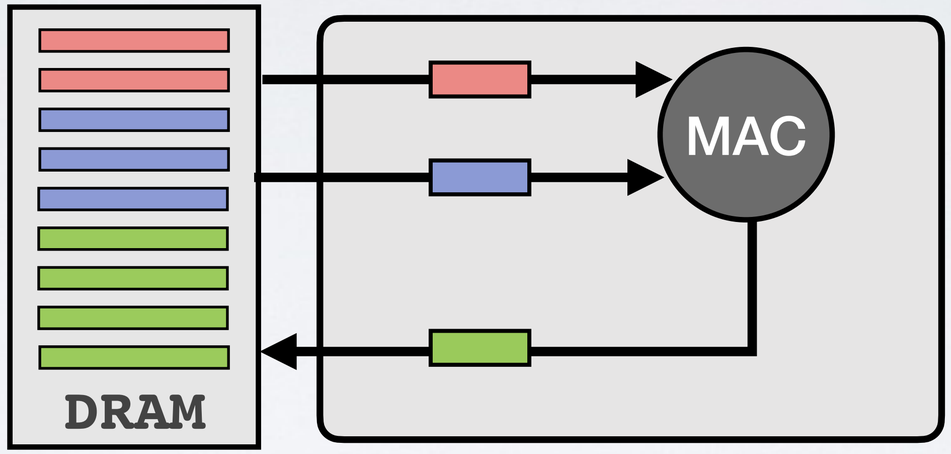
```
output[i] += input[i+j] * weight[j];
```

- **µIR Dataflow graph for each task block**
  - Pipelined (single and multi-cycle)
  - Typed (e.g., FP precision)
  - Non-deterministic cycle ops (e.g., Mem ops)
  - Permits local transformations because tasks are asynchronous

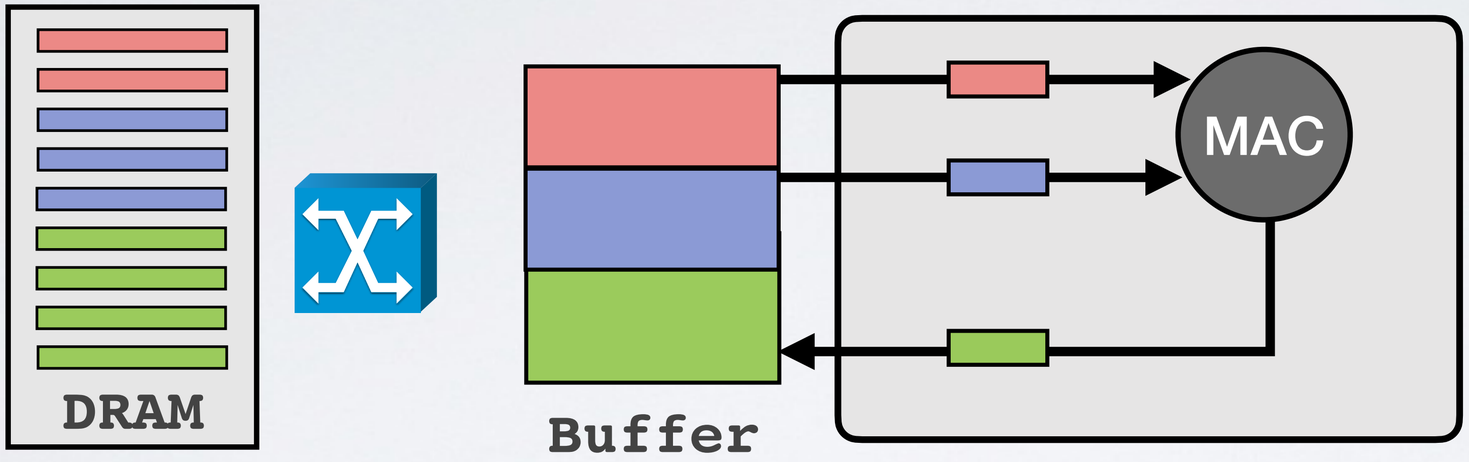


- Motivation
- µIR behaviour graph
- **µIR structural graph**
- Evaluation
- Summary

µIR Structural Graph : Stage 0 — Unoptimized

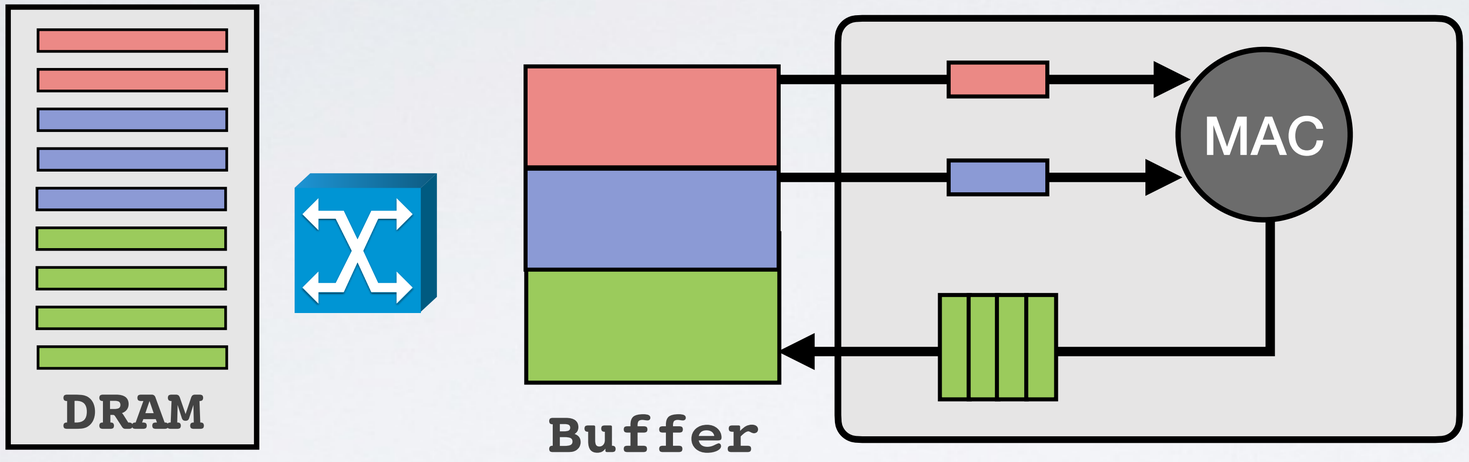


µIR Structural Graph : Stage 1 — Locality



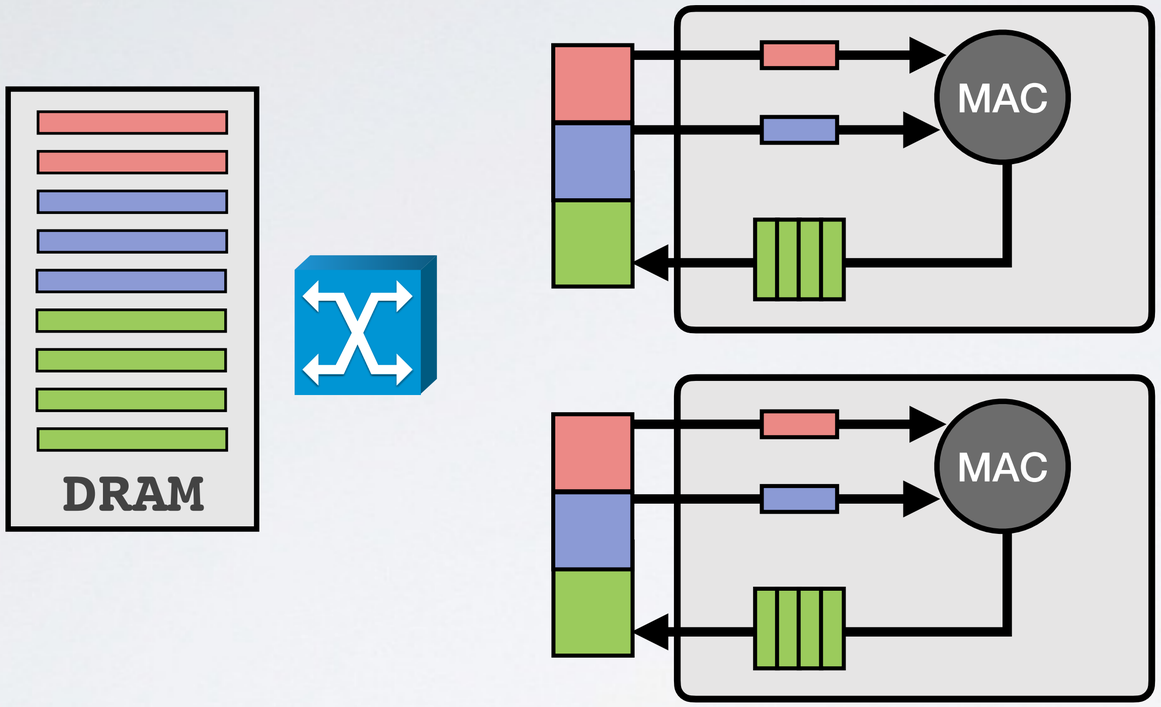
- Define custom memory hierarchy — Buffer, FIFO

µIR Structural Graph : Stage 1 — Locality



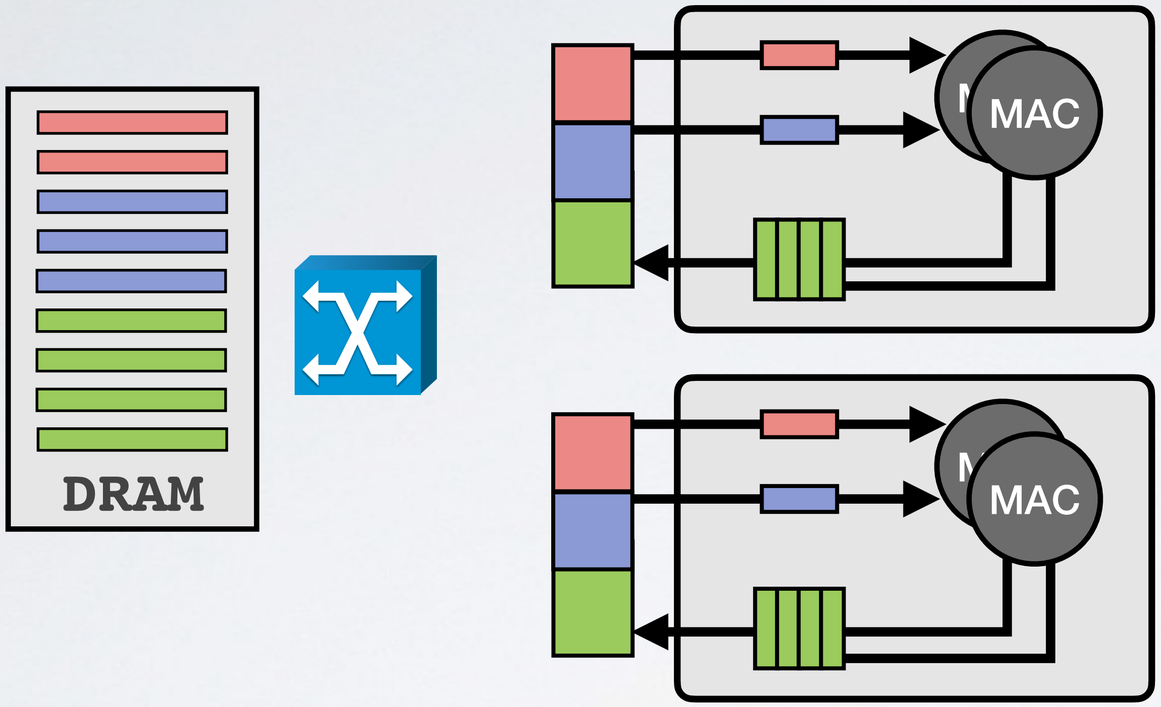
- Define custom memory hierarchy — Buffer, FIFO

µIR Structural Graph : Stage 2 — Tiling



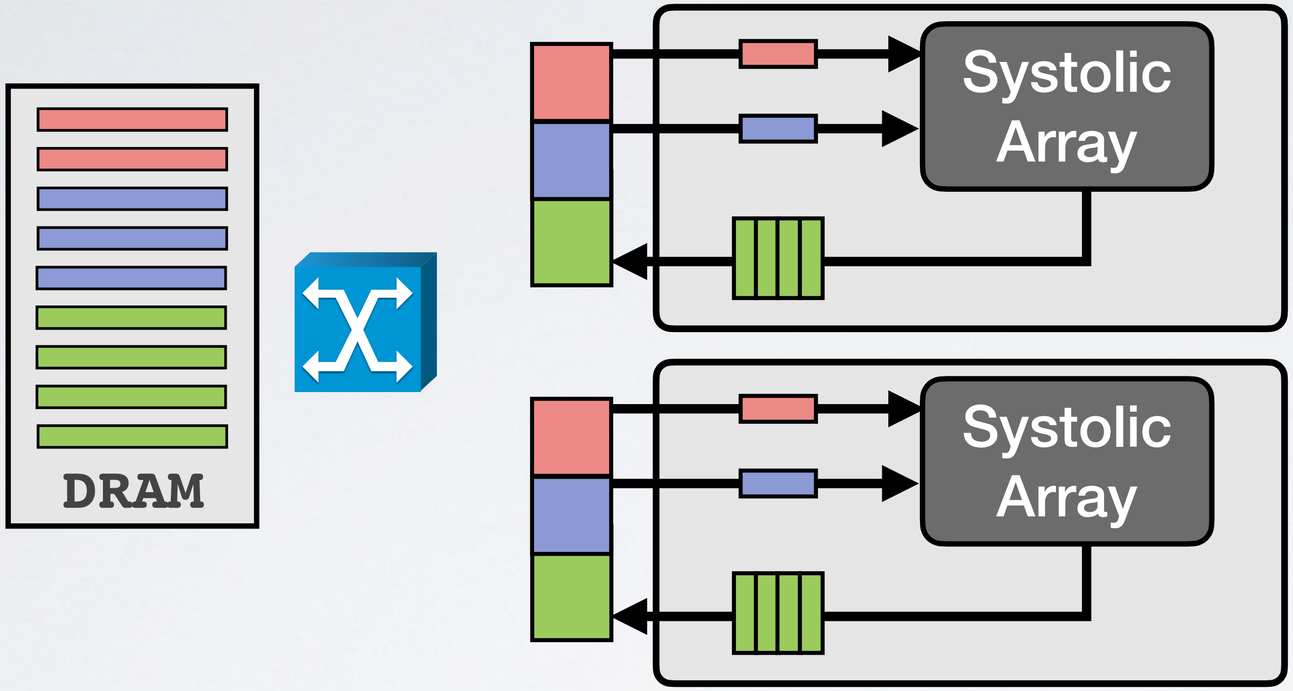
- Define custom memory hierarchy — Buffer, FIFO
- Tiling asynchronous task blocks

μIR Structural Graph : Stage 3 — Pipelining



- Define custom memory hierarchy — Buffer, FIFO
- Tiling asynchronous task blocks
- Start pipelining the operands

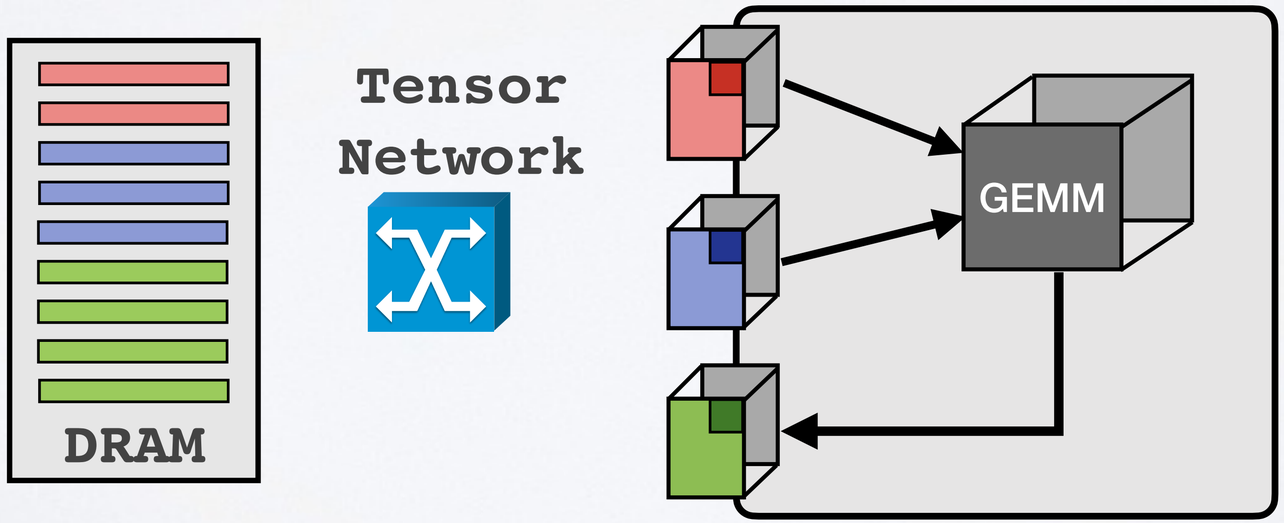
μIR Structural Graph : Stage 3 — Pipelining



- Define custom memory hierarchy — Buffer, FIFO
- Tiling asynchronous task blocks
- Start pipelining the operands
- Define custom operator

μIR Structural Graph : Stage 4 Higer-Order Ops

- Extensible IR
  - Introduce new ops and new types
- Existing stages/transformations have to work
- μIR components are generic: Dataflow nodes, buffers and memory network



- Motivation
- μIR behaviour graph
- μIR structural graph
- **Evaluation**
- Summary

µIR Targets

- **FPGA**
  - Arria 10 Soc
  - Scripted process

FPGAs FPGA

https://github.com/sfu-arch/uir-fpga

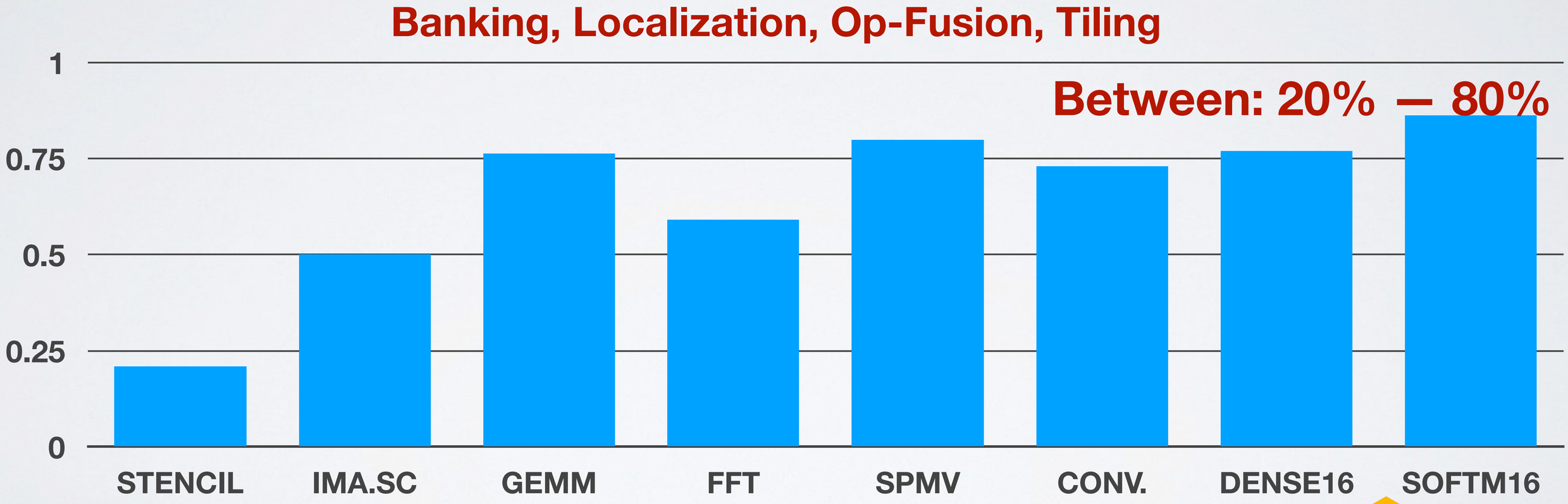
- **Cycle accurate simulation (e.g., gem5)**
  - C++ driver and a Python binding for ease of use
  - Cache-based memory interface

Simulator

https://github.com/sfu-arch/uir-sim

µIR Iterative Optimizations — Stacking

- Normalized to the baseline accelerator



µIR Productivity

	#Changes µIR	#Changes Firrtl	Ratio
Saxpy	9	73	**9.3**
Stencil	12	142	**12.4**
Image Sca.	10	84	**8.4**

Available Now!

https://github.com/sfu-arch/uir

https://github.com/sfu-arch/uir-sim

https://github.com/sfu-arch/uir-fpga

https://github.com/sfu-arch/uir-docker

Thanks to Open-Source community!