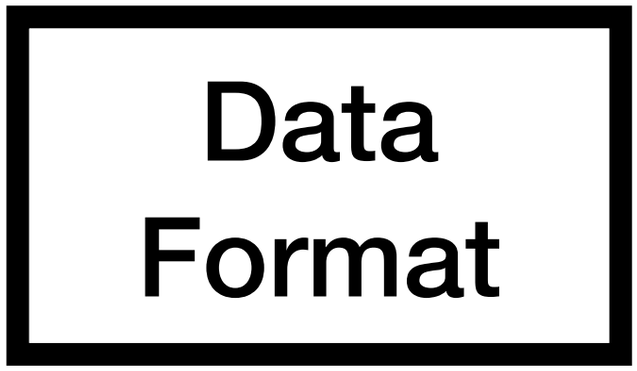
CGRA Meta-Compilation

Ross Daly, Jack Melchert

Notes on flow chart notation



**Configuration Files,
Source Code,
IRs, etc…**

Note: Can be a serialized file, or in memory

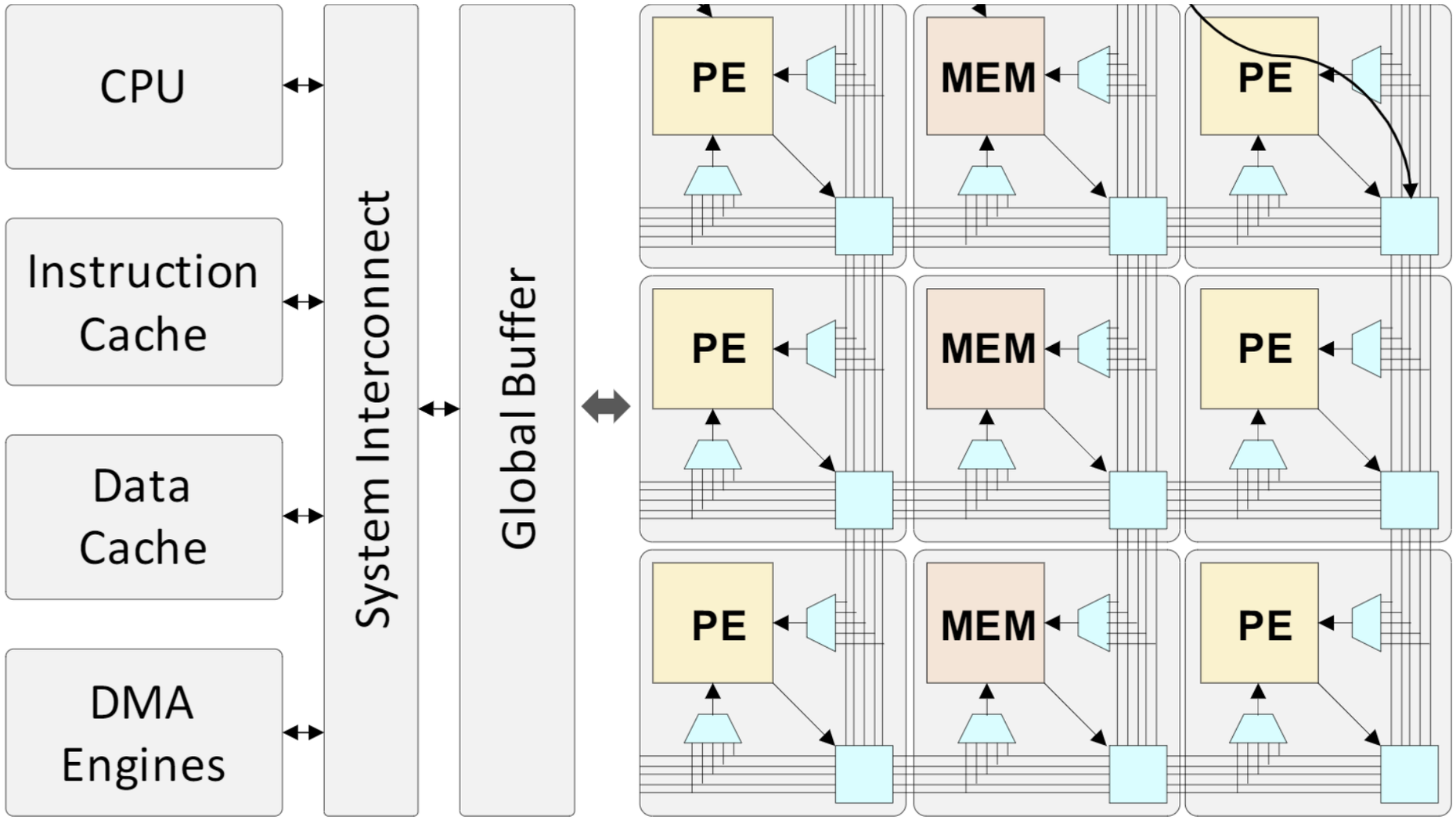
**Parser,
Code generator,
Mapper, PnR,
linker, etc…**

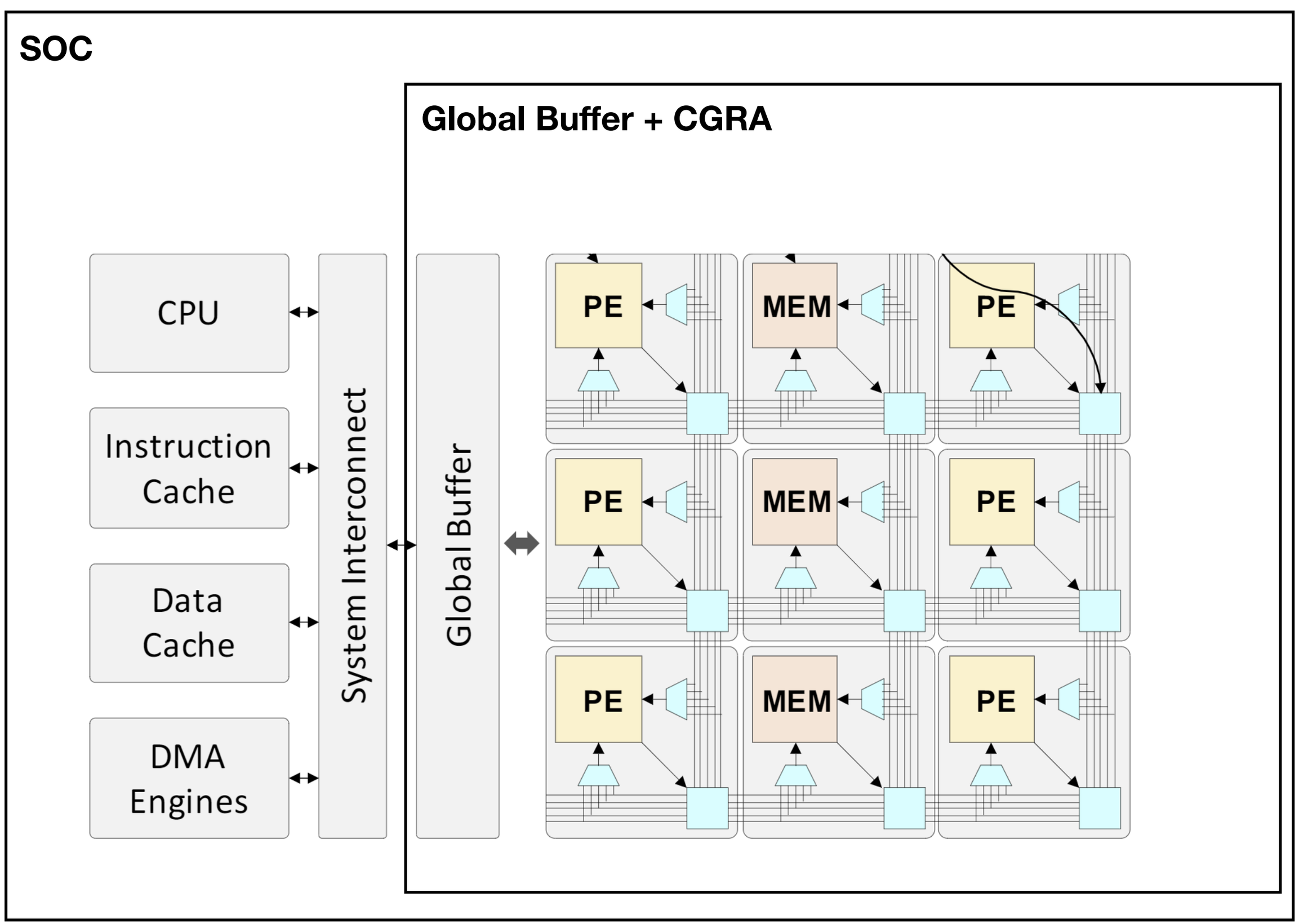
Note: Produces either a data format OR another processing step

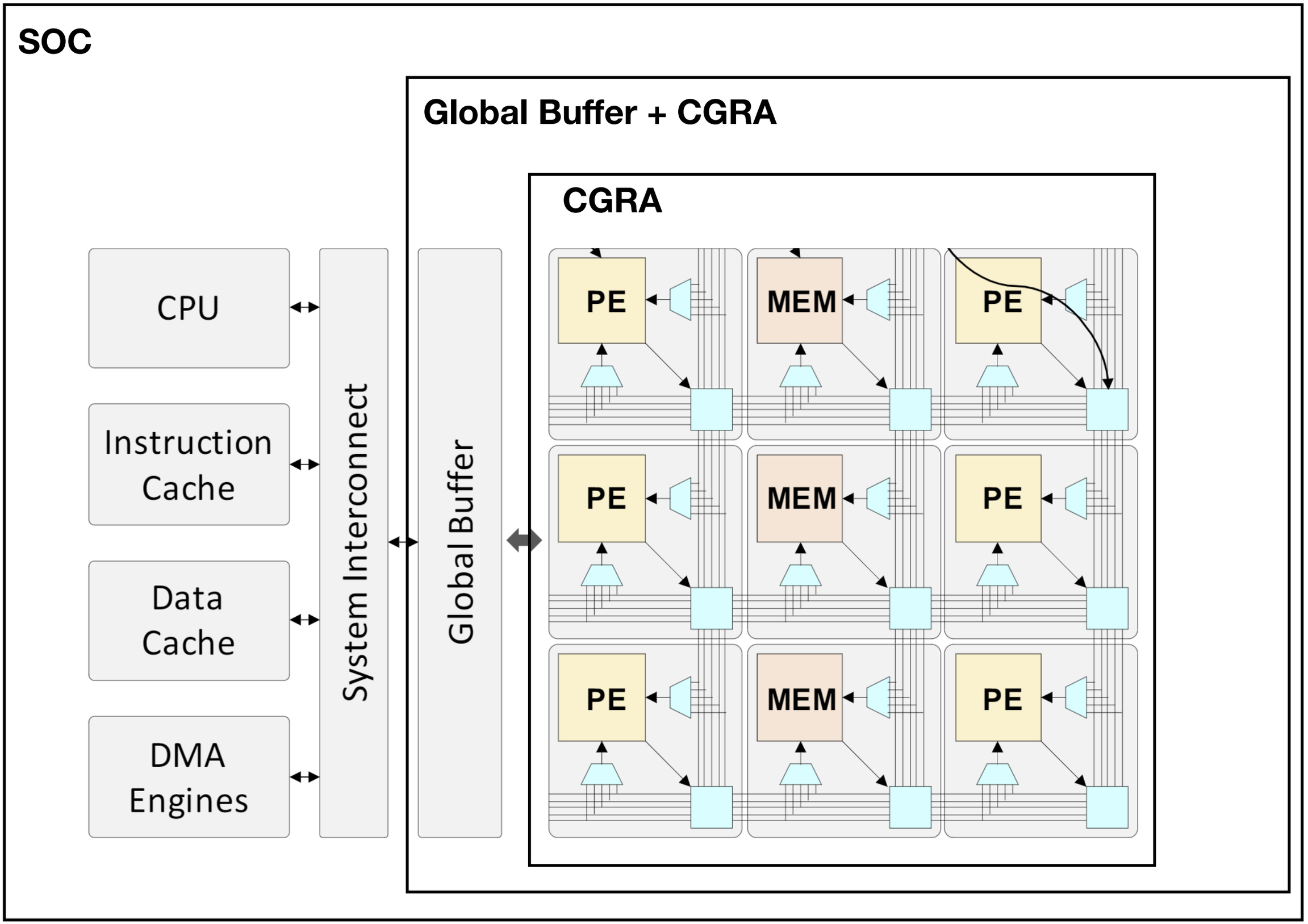
Goal: Accelerate class(es) of programs

- Current: ML, vision, image processing
- Future: Sparse
- Use the Halide DSL to represent these programs.
- We do NOT know what the best hardware accelerator is.
  - But we have a good sense of the general structure.
  - Island-style CGRAs with a multi-level memory hierarchy
- We are designing a “Meta” CGRA SOC

SOC


CPU
Instruction Cache
Data Cache
DMA Engines
System Interconnect
Global Buffer
PE
MEM
PE
PE
MEM
PE
PE
MEM
PE


SOC
Global Buffer + CGRA
CPU
Instruction Cache
Data Cache
DMA Engines
System Interconnect
Global Buffer
PE
MEM
PE
PE
MEM
PE
PE
MEM
PE

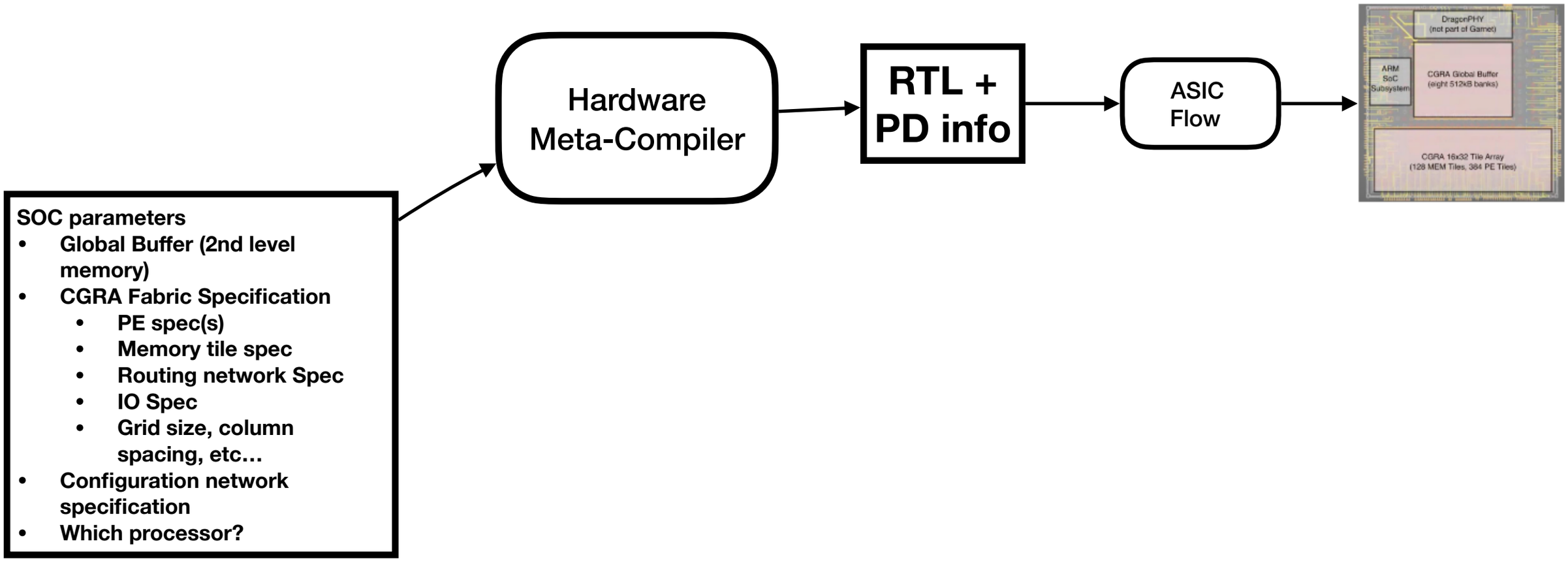

SOC
Global Buffer + CGRA
CGRA
CPU
Instruction Cache
Data Cache
DMA Engines
System Interconnect
Global Buffer
PE
MEM
PE
PE
MEM
PE
PE
MEM
PE

Meta CGRA SOC

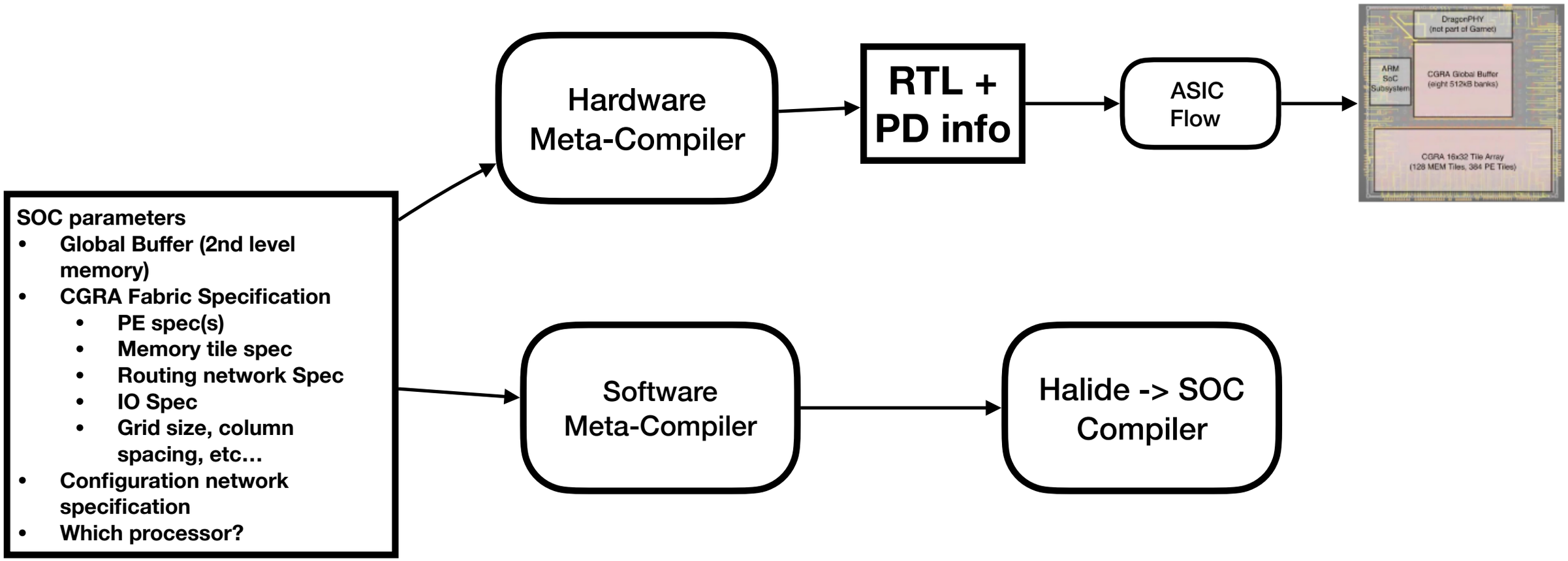
SOC parameters

- CGRA Fabric Specification
  - PE spec(s)
    - Operations?
    - number/size of inputs/outputs?
    - Internal routing?
  - Memory tile spec
    - Size? Bandwidth? Address generation capabilities?
  - Routing network Spec
    - Number of lines? connectivity?
  - IO Spec
  - Grid size, column spacing, etc…
- Global Buffer (2nd level memory)
  - Number of banks
  - Number of controllers/address generators
  - Total capacity, etc…
- Configuration network specification (MMIO addressing, etc…)
- ARM vs RISCV? What processor? number of DMAs? Caches?

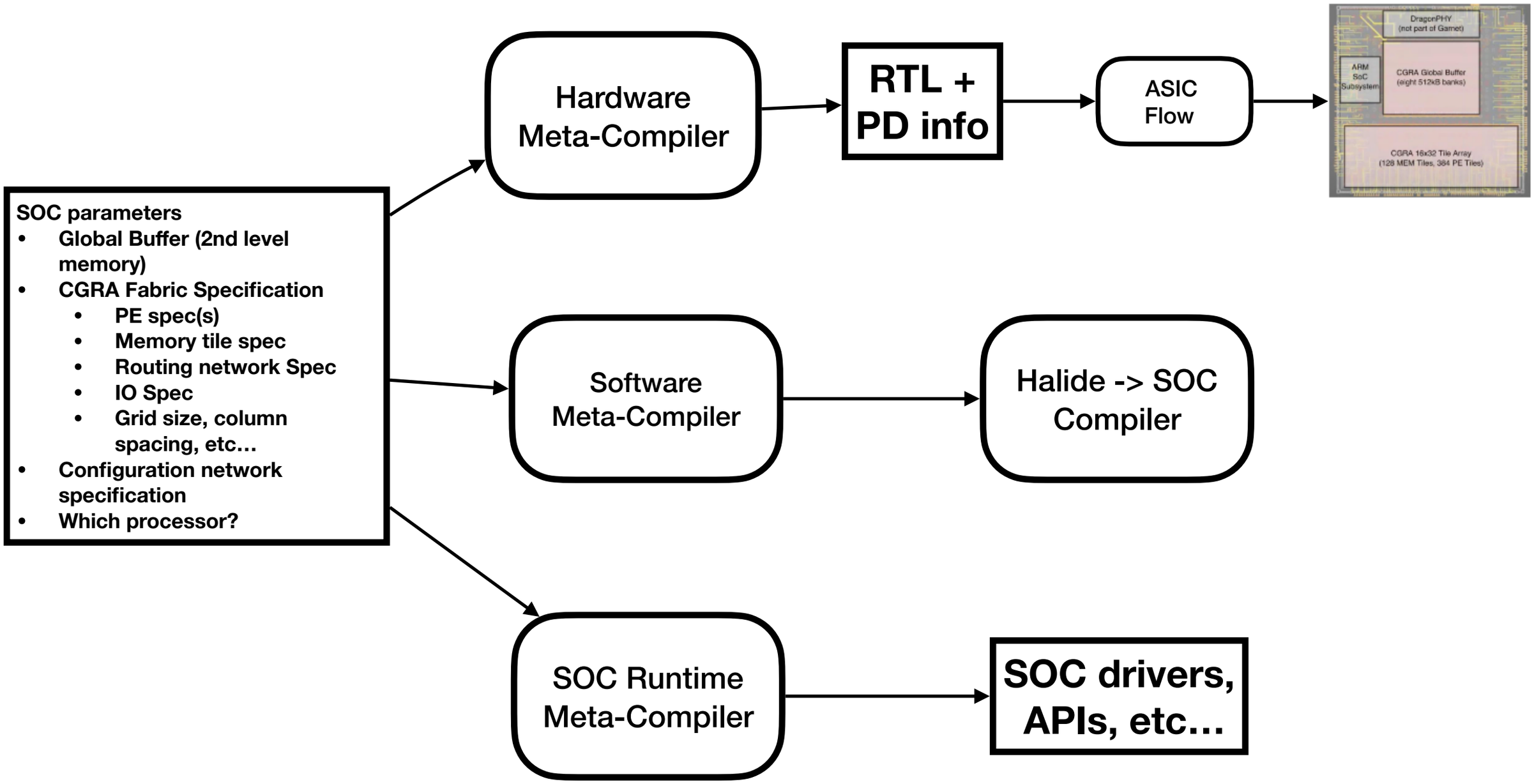
Meta Compiler(s)



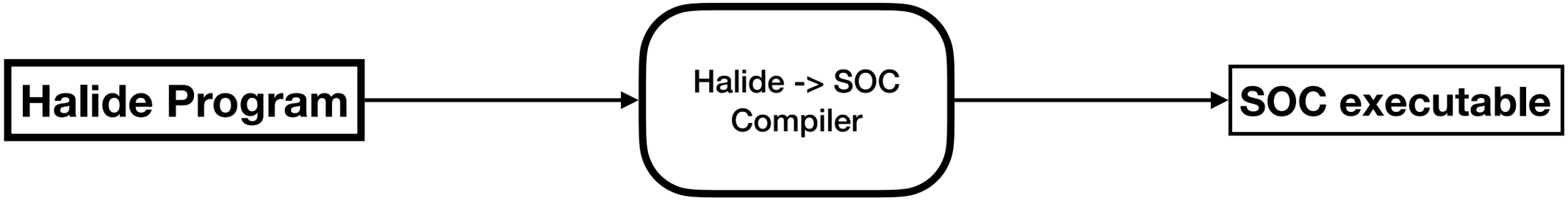
Meta Compiler(s)



Meta Compiler(s)



Halide -> SOC Compiler



Halide Example: 3x3 convolution in hardware

```
// Algorithm
RDom win(0, 3, 0, 3);
conv(x, y)  = 0.0;
conv(x, y) += input(x + win.x, y + win.y) *
              weights(win.x, win.y);
output(x, y) = conv(x, y);

// Schedule
output.tile(x, y, xo, yo, xi, yi, 64, 64);
output.hw_accelerate(xo, xi);

input.stream_to_accelerator();
input.store_at(output, xo)
     .compute_at(output, xo)

conv.update()
    .unroll(win.x)
    .unroll(win.y);
```

Defines a 3x3 convolution using input and weights.

Creates an accelerator from input to output operating on 64x64 image tiles.

Specifies that input should be stored in a memory.

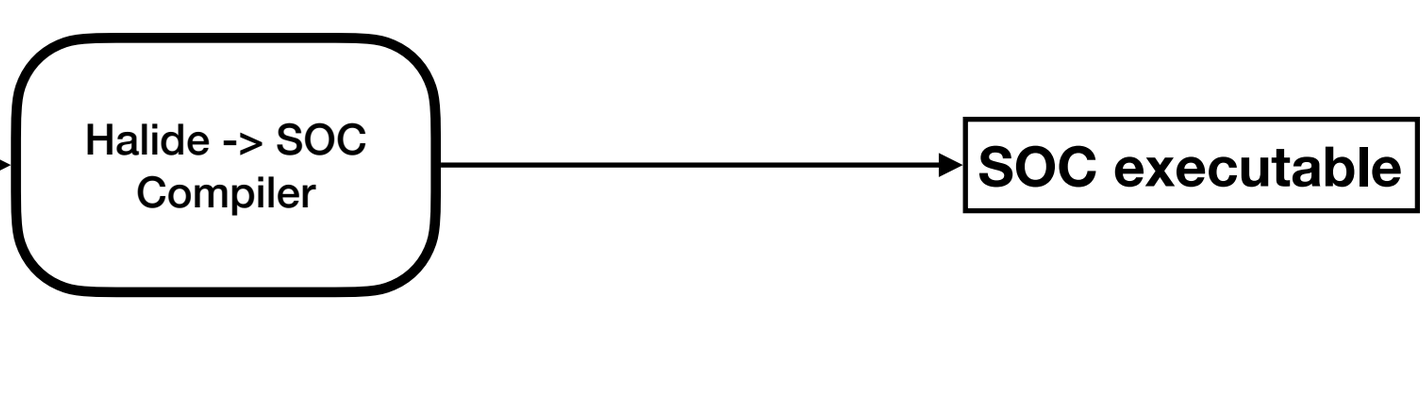
Unrolls the implicit RDom for loops; thus 9 multipliers are created.

Halide -> SOC Compiler

```
brighten(x, y) = input(x, y) * 2;
output(x, y) = brighten(x, y);

output.tile(x,y,xo,yo,xi,yi,64,64)
      .hw_accelerate(xo, xi);

input.stream_to_accelerator();
input.store_at(output, xo)
     .compute_at(output, xo);
```



Halide -> SOC Compiler

```
brighten(x, y) = input(x, y) * 2;
output(x, y) = brighten(x, y);

output.tile(x,y,xo,yo,xi,yi,64,64)
      .hw_accelerate(xo, xi);

input.stream_to_accelerator();
input.store_at(output, xo)
     .compute_at(output, xo);
```

Halide -> SOC Compiler

SOC executable

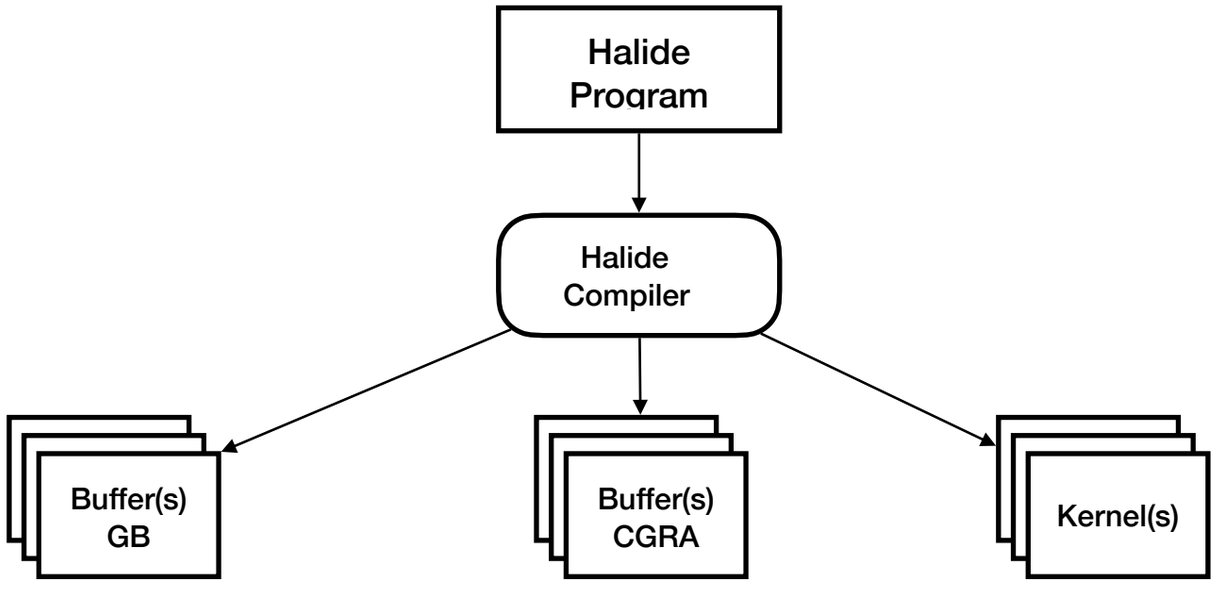
- SOC configuration
  - Processor executable
    - DMA config(s)
    - Clks/Interrupt config
    - Configuration drivers
    - main()
  - CGRA+GB Bitstream(s)
    - Global Buffer config
    - CGRA configuration
      - Mem tile(s) config
      - PE tile(s) config
      - IO tile(s) config
      - Routing network config

Halide Program

Halide -> SOC Compiler

Global Buffer Bitstream

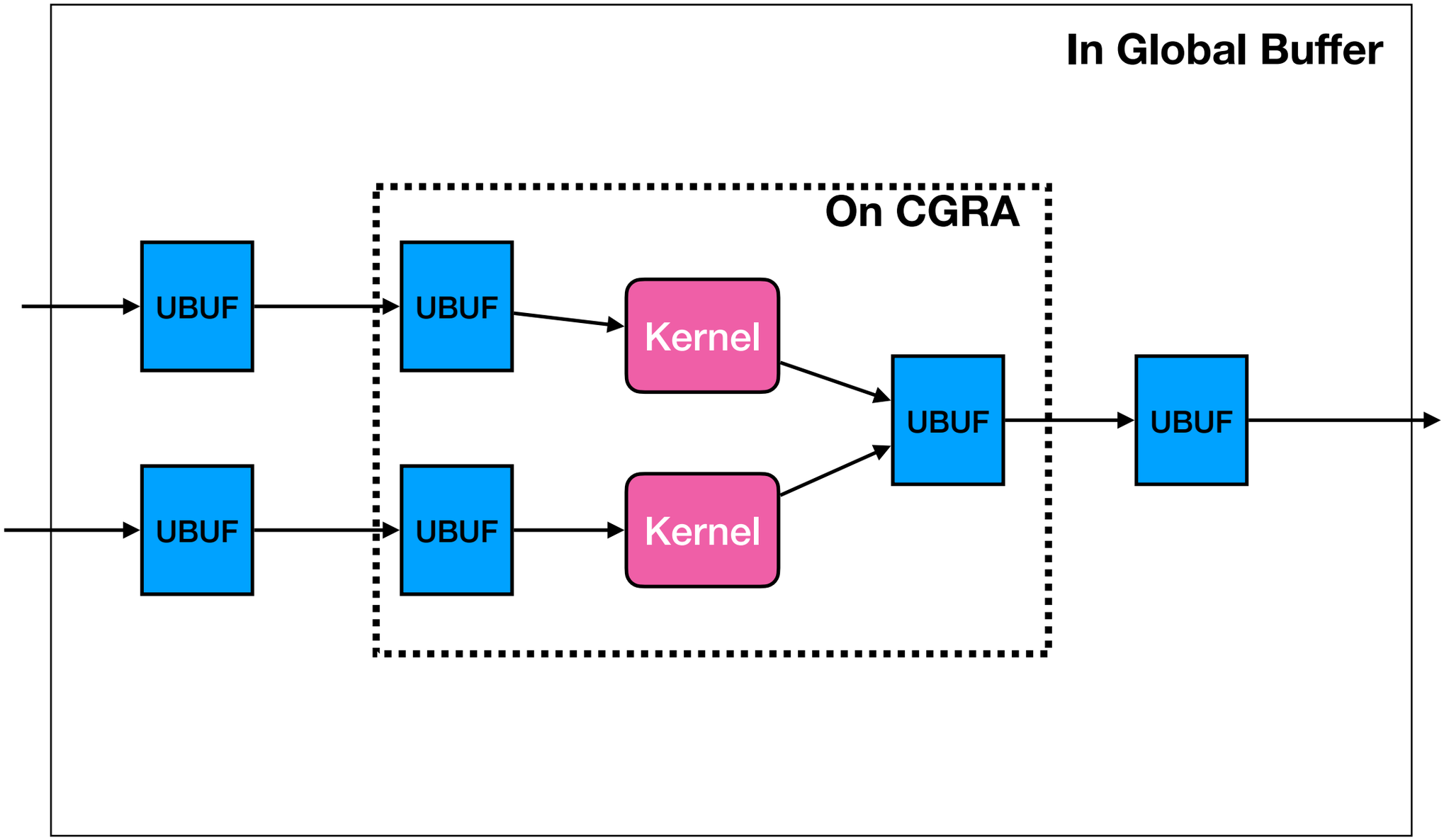
CGRA Bitstream



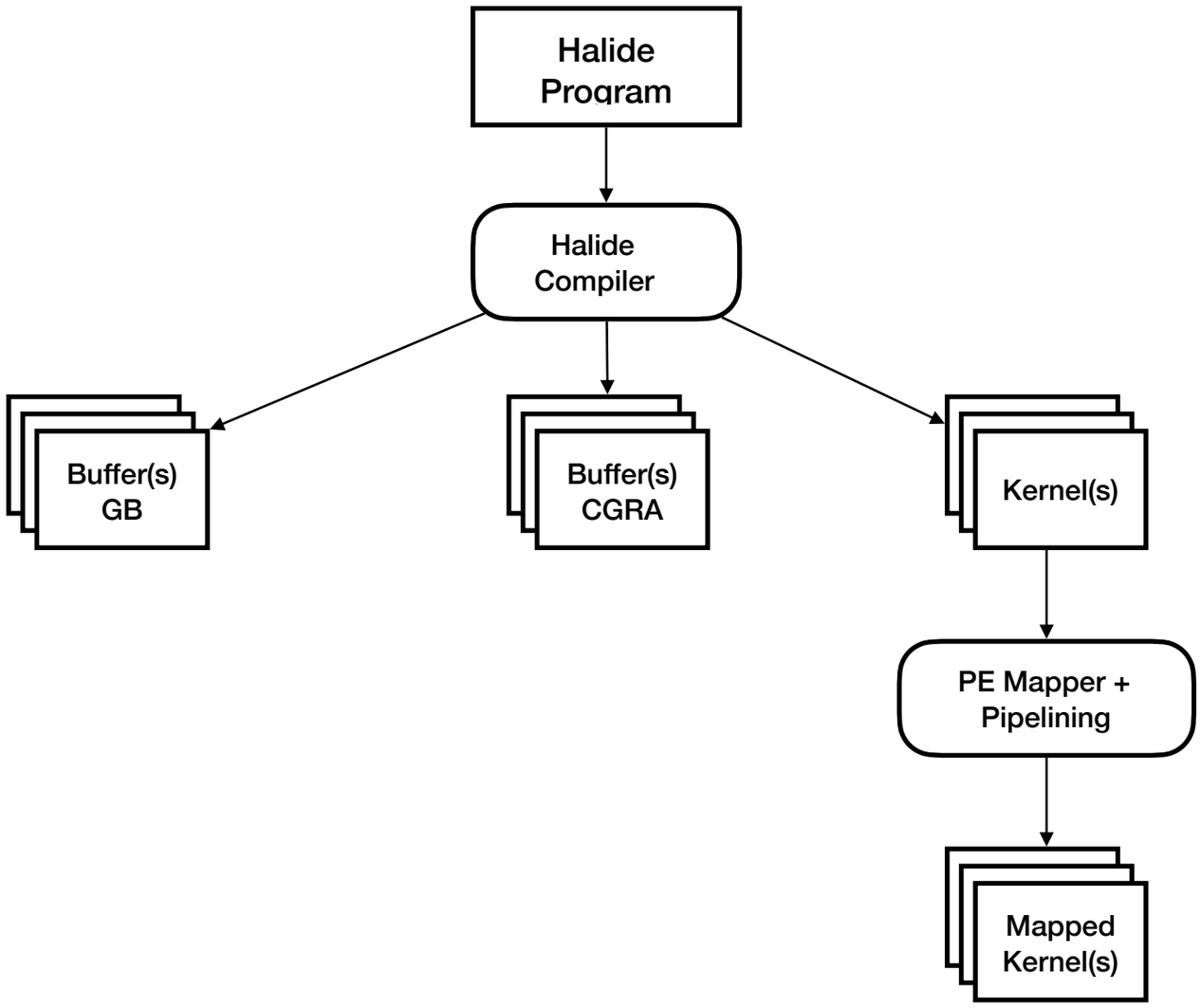
Compile to IR:
Unified Buffers +
Computation Kernels

Global Buffer
Bitstream

CGRA
Bitstream

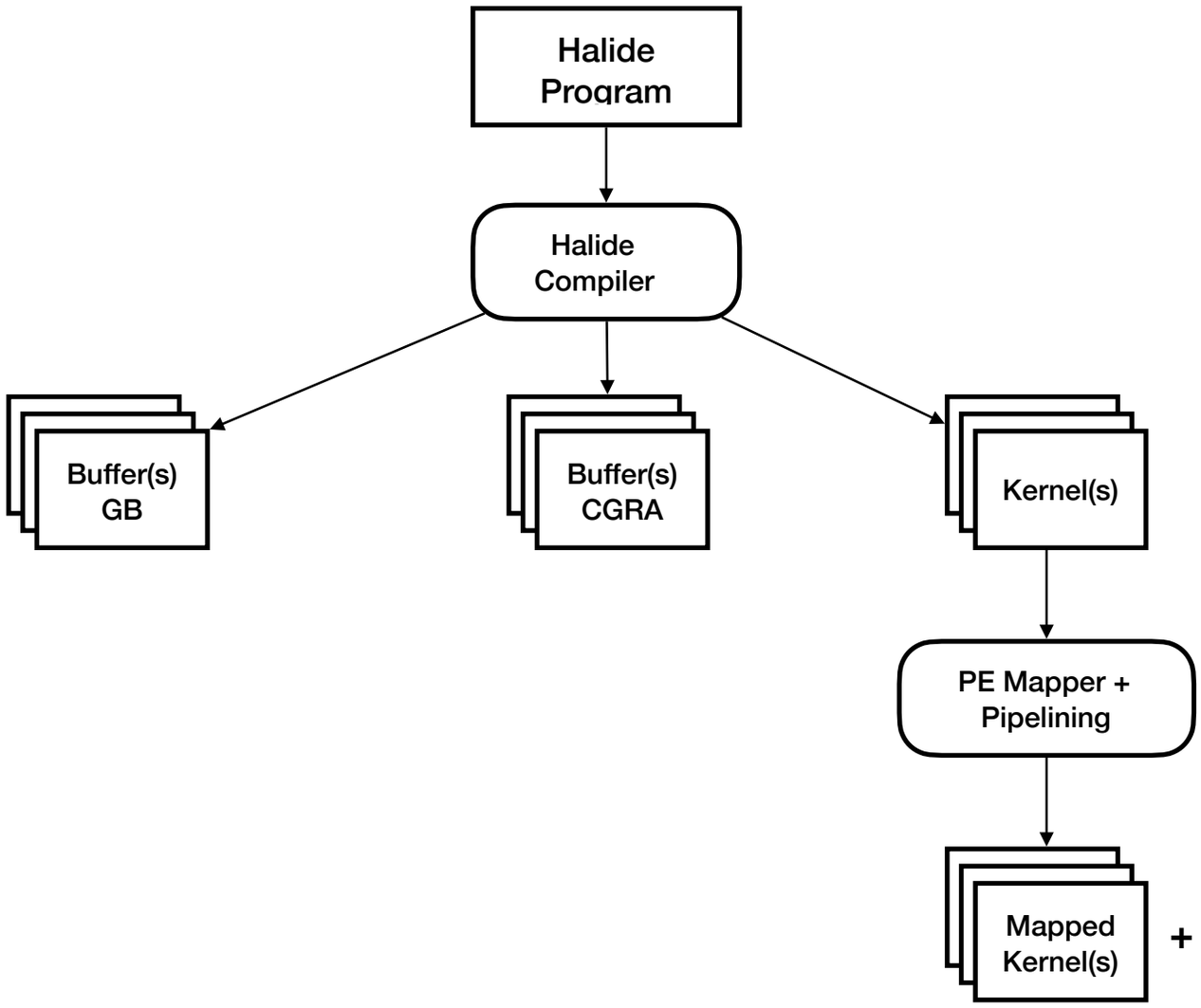

In Global Buffer
On CGRA
UBUF
UBUF
Kernel
UBUF
UBUF
UBUF
UBUF
Kernel

“Relaxed-timing” Mapping/Instruction Selection



Global Buffer Bitstream

CGRA Bitstream



+ latency info

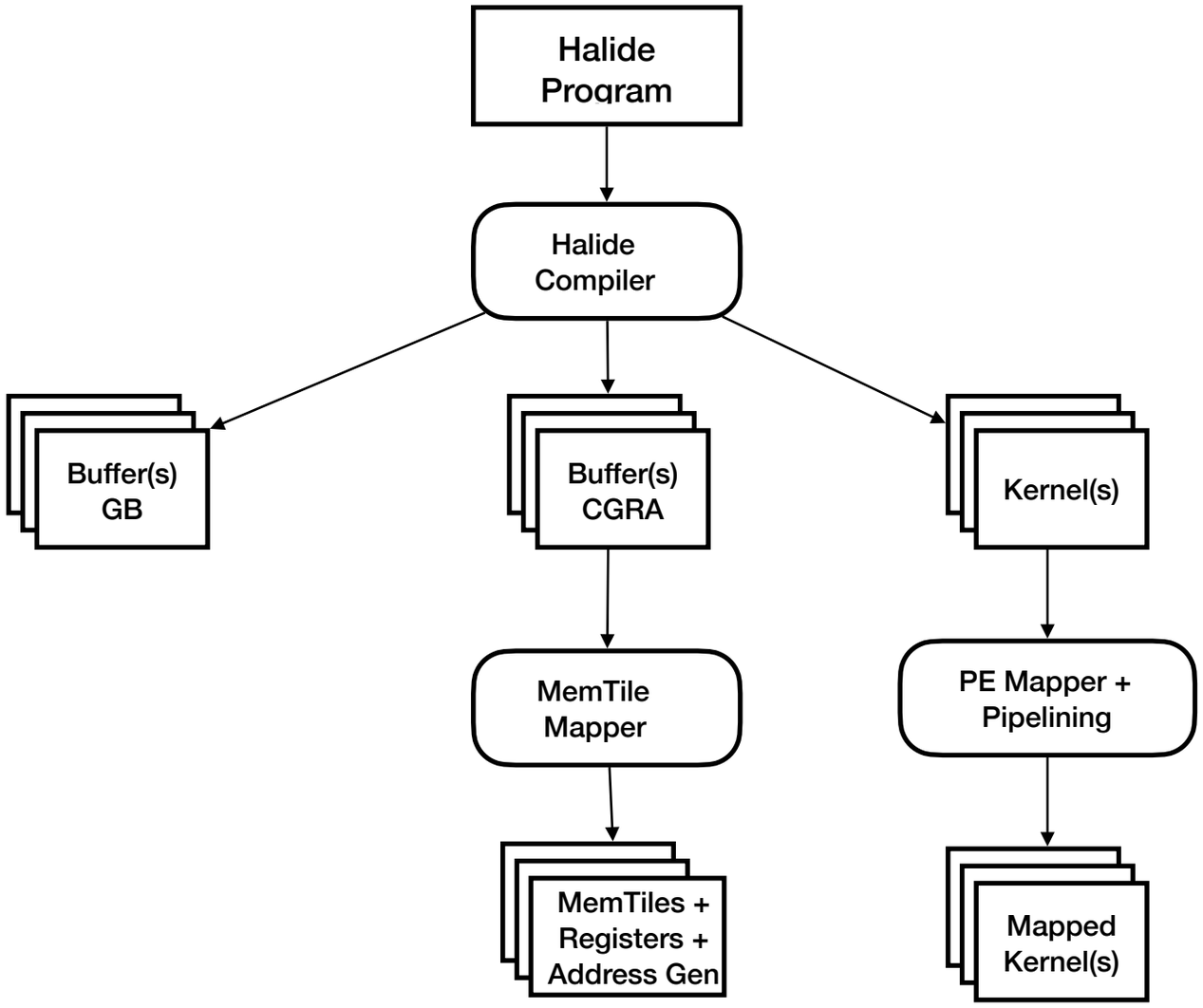
“Relaxed-timing” Mapping/Instruction Selection

Note: We can pipeline these kernels
And sometimes need to!

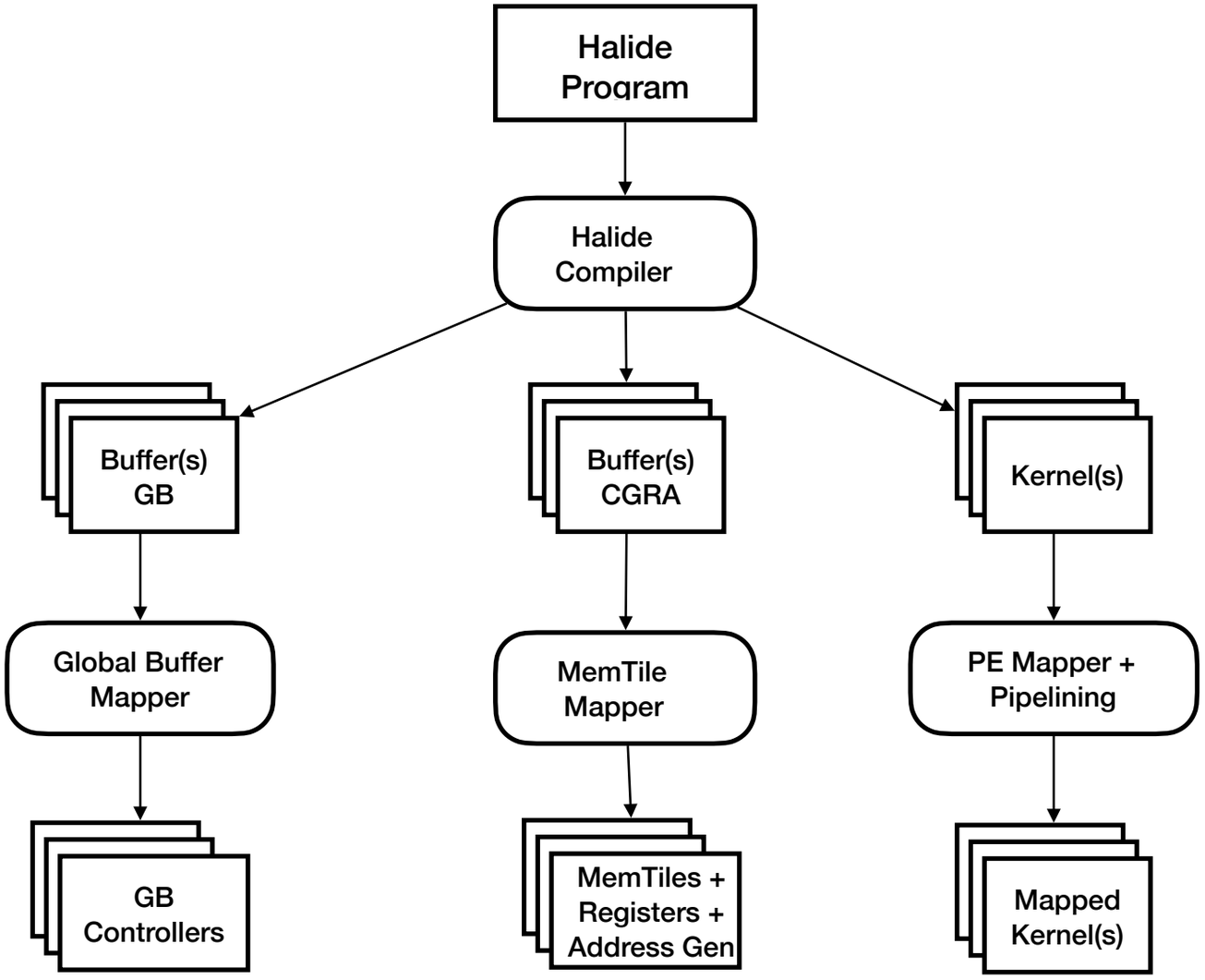
Global Buffer
Bitstream

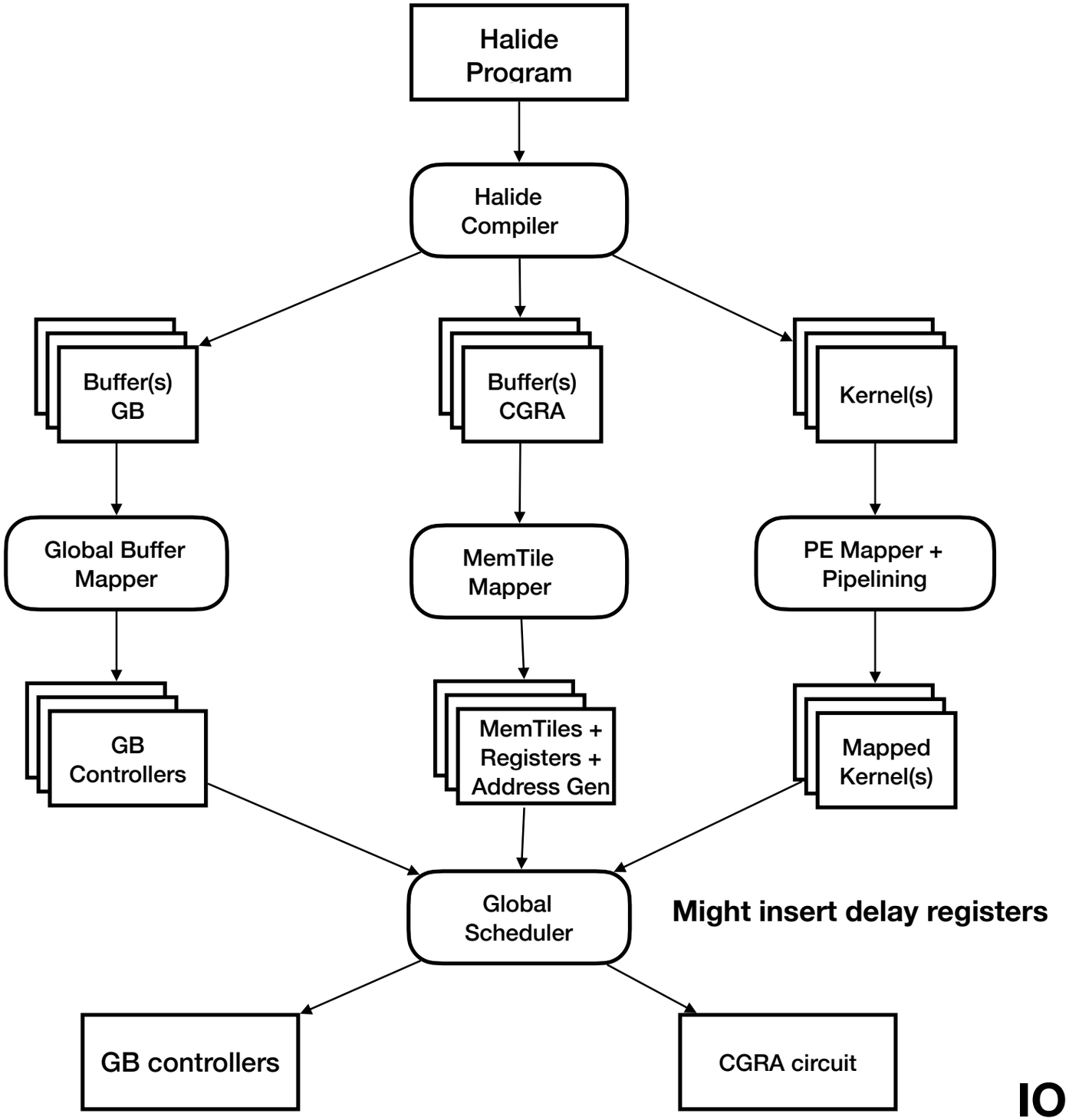
CGRA
Bitstream

“Relaxed-timing” Mapping/Instruction Selection



“Relaxed-timing” Mapping/Instruction Selection





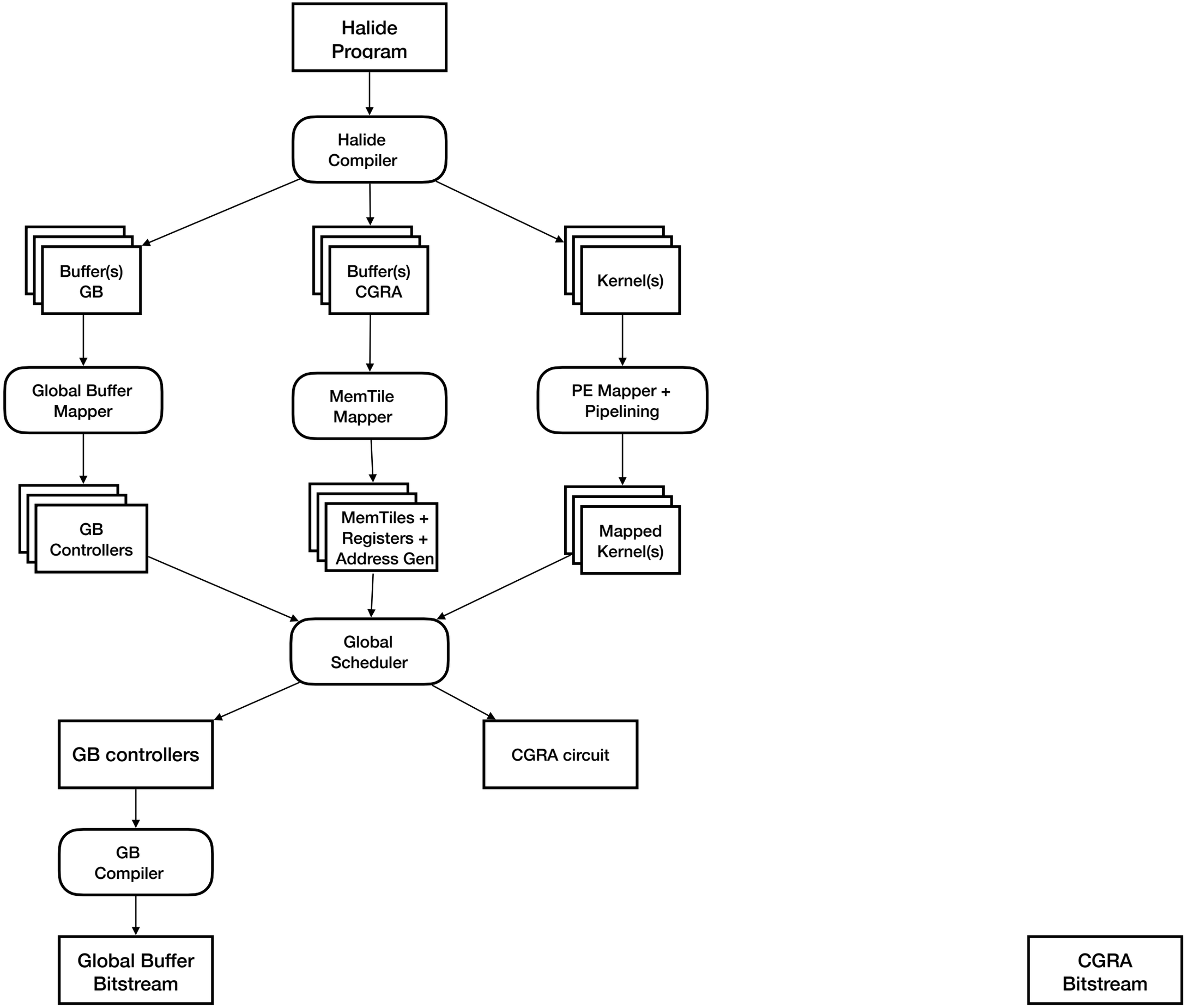
Might insert delay registers

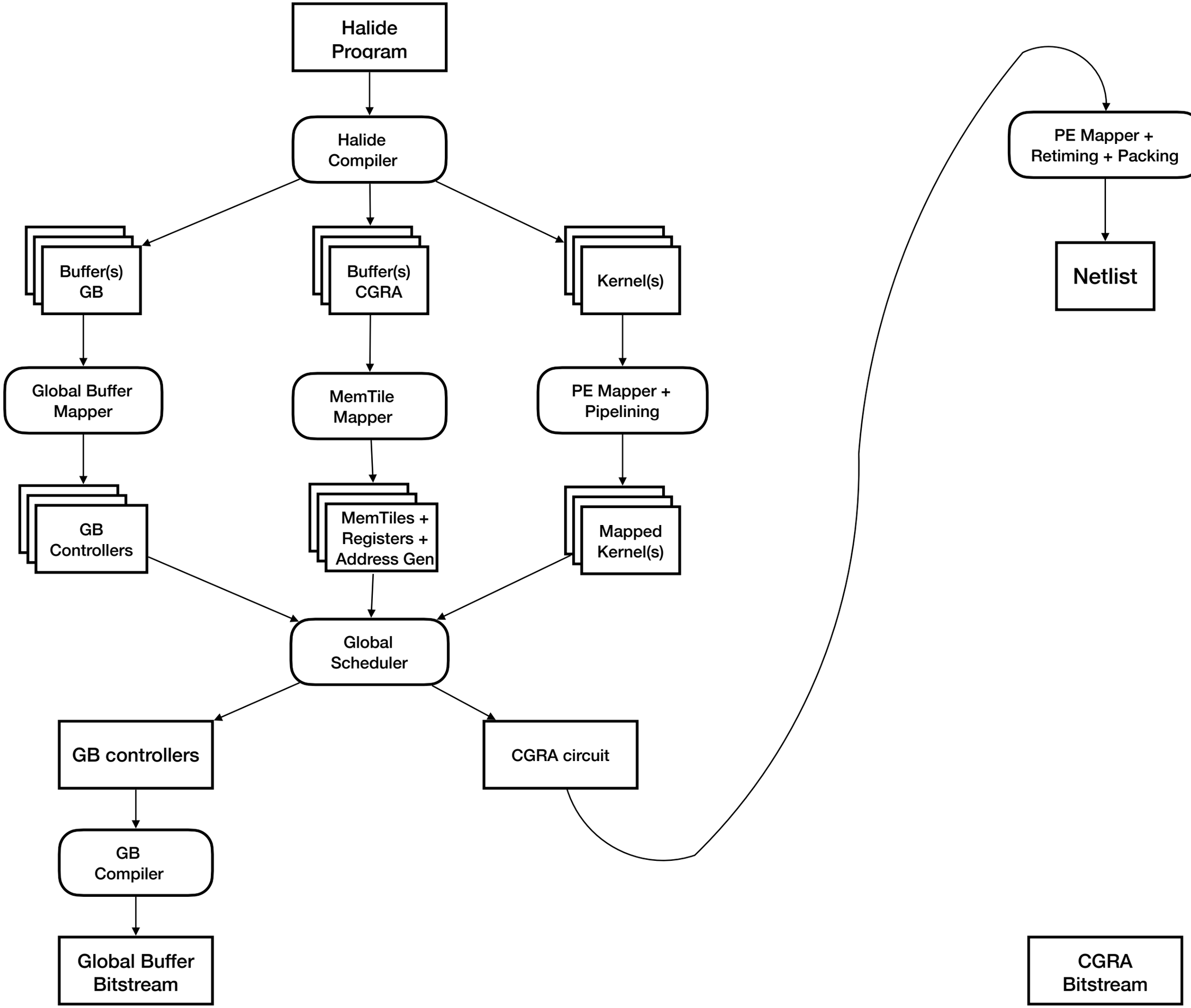
Schedule everything!

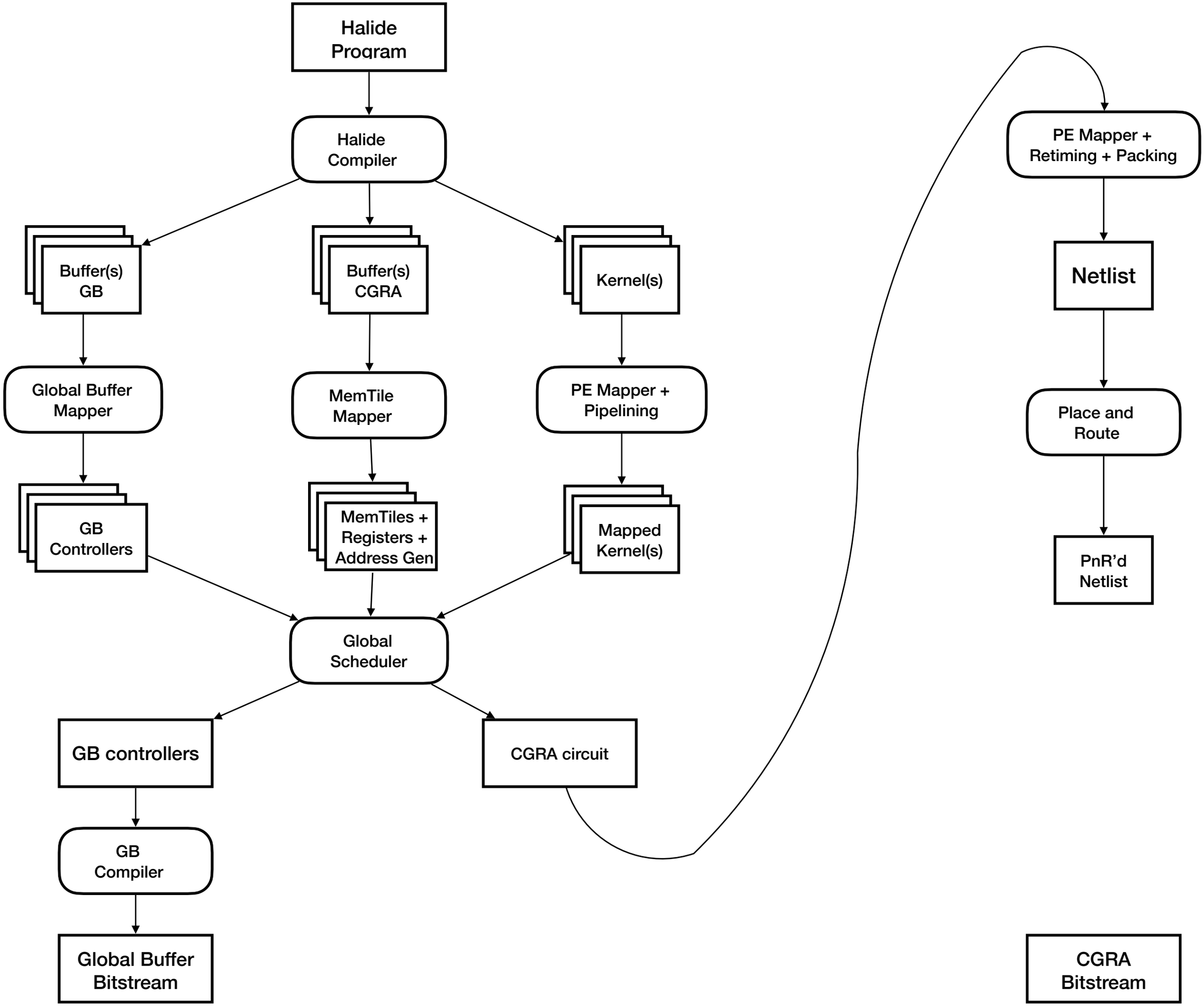
IO Timing cannot be modified after scheduling

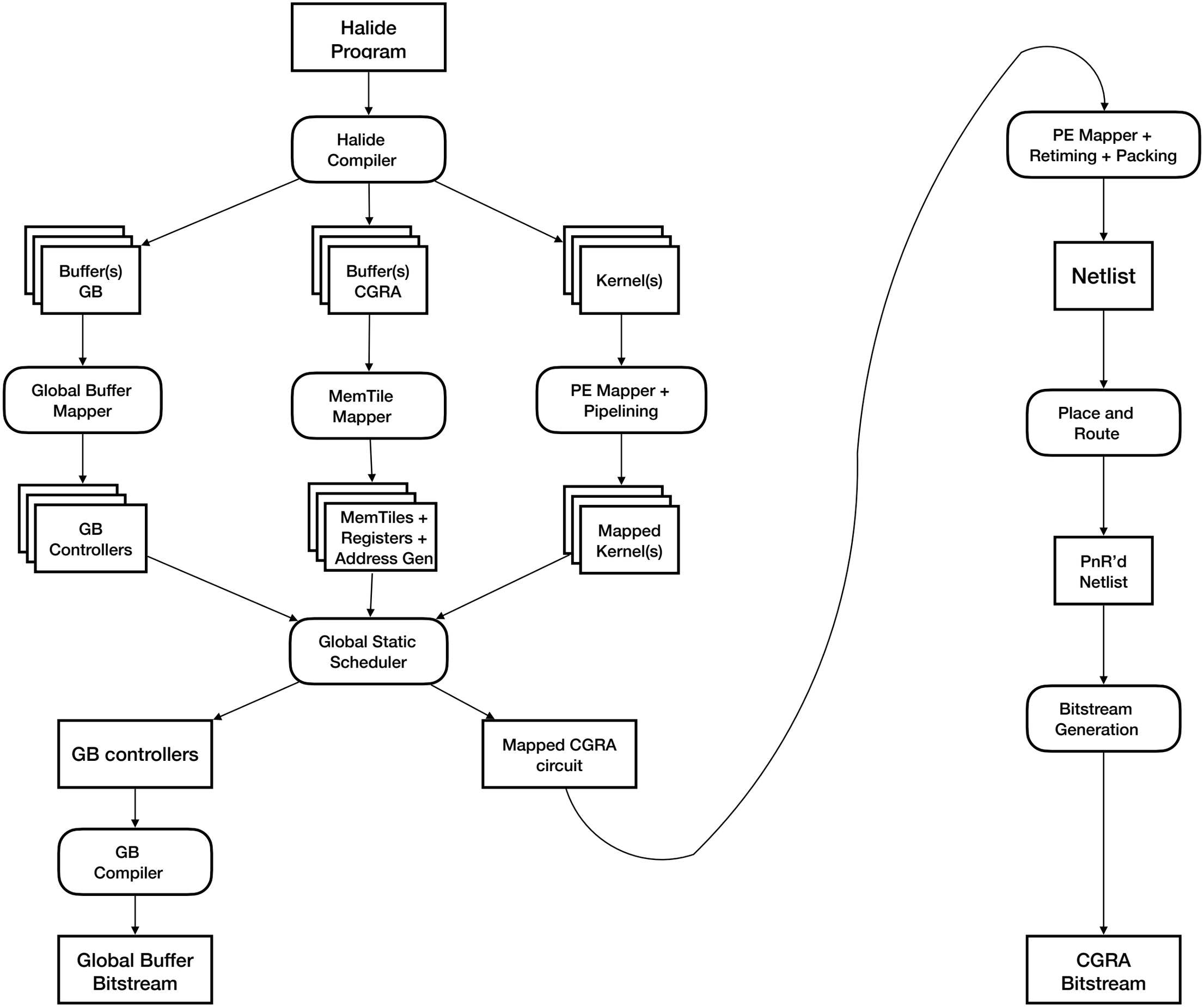
Global Buffer Bitstream

CGRA Bitstream


Halide Program
Halide Compiler
Buffer(s) GB
Buffer(s) CGRA
Kernel(s)
Global Buffer Mapper
MemTile Mapper
PE Mapper + Pipelining
GB Controllers
MemTiles + Registers + Address Gen
Mapped Kernel(s)
Global Scheduler
GB controllers
CGRA circuit
GB Compiler
Global Buffer Bitstream
CGRA Bitstream


Halide Program
Halide Compiler
Buffer(s) GB
Buffer(s) CGRA
Kernel(s)
Global Buffer Mapper
MemTile Mapper
PE Mapper + Pipelining
GB Controllers
MemTiles + Registers + Address Gen
Mapped Kernel(s)
Global Scheduler
GB controllers
CGRA circuit
GB Compiler
Global Buffer Bitstream
PE Mapper + Retiming + Packing
Netlist
CGRA Bitstream


Halide Program
Halide Compiler
Buffer(s) GB
Buffer(s) CGRA
Kernel(s)
Global Buffer Mapper
MemTile Mapper
PE Mapper + Pipelining
GB Controllers
MemTiles + Registers + Address Gen
Mapped Kernel(s)
Global Scheduler
GB controllers
CGRA circuit
GB Compiler
Global Buffer Bitstream
PE Mapper + Retiming + Packing
Netlist
Place and Route
PnR'd Netlist
CGRA Bitstream


Halide Program
Halide Compiler
Buffer(s) GB
Buffer(s) CGRA
Kernel(s)
Global Buffer Mapper
MemTile Mapper
PE Mapper + Pipelining
GB Controllers
MemTiles + Registers + Address Gen
Mapped Kernel(s)
Global Static Scheduler
GB controllers
Mapped CGRA circuit
GB Compiler
Global Buffer Bitstream
PE Mapper + Retiming + Packing
Netlist
Place and Route
PnR'd Netlist
Bitstream Generation
CGRA Bitstream

Current State of our Meta Compilation

Meta CGRA SOC

SOC parameters

- CGRA Fabric Specification
  - PE spec(s)
    - Operations?
    - number/size of inputs/outputs?
    - Internal routing?
  - Memory tile spec
    - Size? Bandwidth? Address generation capabilities?
  - Routing network Spec
    - Number of lines? connectivity?
  - IO Spec
  - Grid size, column spacing, etc…
- Global Buffer (2nd level memory)
  - Number of banks
  - Number of controllers/address generators
  - Total capacity, etc…
- Configuration network specification (MMIO addressing, etc…)
- ARM vs RISCV? What processor? number of DMAs? Caches?

Meta-CGRA

SOC parameters

- CGRA Fabric Specification
  - PE spec(s) = **PEak** program
  - Memory tile spec = **Lake** program
  - Routing network Spec = **Canal** program
  - IO/grid size, etc… = **Gemstone** program
- Global Buffer = hard-coded (future **Lake**?)
- Configuration network specification = hard-coded
- Processor/DMAs/Cache = hard-coded

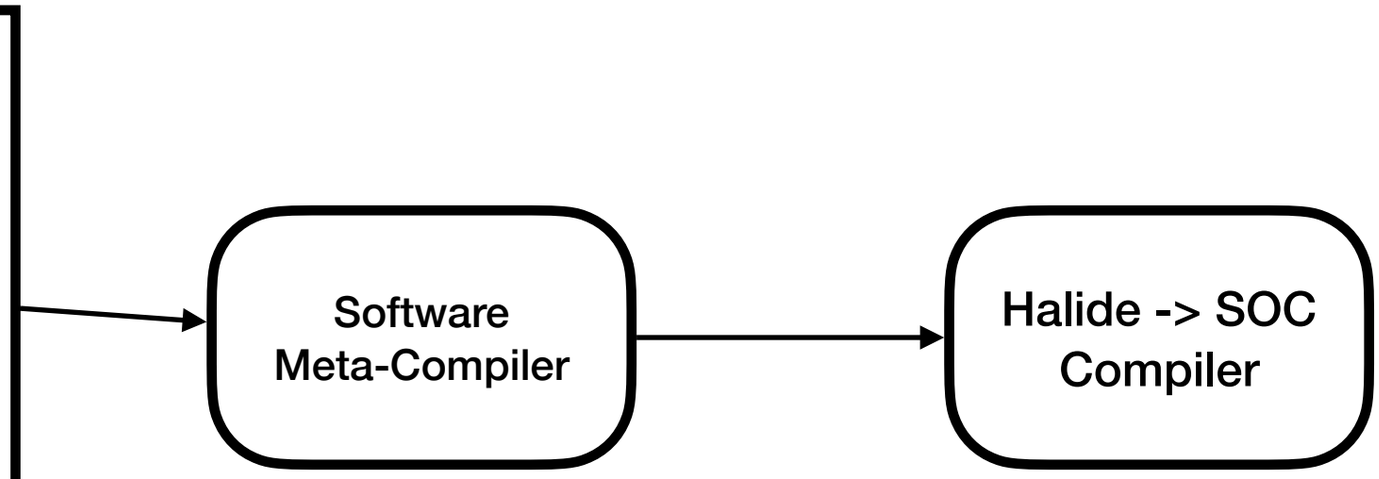
Amber parameters

SOC parameters
- CGRA Fabric Specification
  - PE spec(s) = **Lassen**
  - Memory tile spec = **Tahoe**
  - Routing network Spec = **(no name)**
  - IO/grid size, etc… = **Amber**
- Global Buffer = hard-coded
- Configuration network specification = hard-coded
- Processor/DMAs/Cache = hard-coded

Meta Compiler(s)

SOC parameters

- **Global Buffer (2nd level memory)**
- **CGRA Fabric Specification**
  - **PE spec(s)**
  - **Memory tile spec**
  - **Routing network Spec**
  - **IO Spec**
  - **Grid size, column spacing, etc…**
- **Configuration network specification**
- **Which processor?**



Currently Being Used

Spec

Meta compiled

Hard coded

parameter independent

Halide Program

Halide Compiler

Buffer(s) CGRA

Kernel(s)

Hard Coded CGRAMapper

Hard Coded CGRAMapper

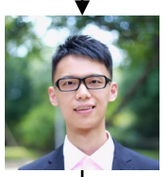
MemTiles

Mapped Kernel(s)

Scheduler

GB controllers

Mapped CGRA circuit



Global Buffer Bitstream

Hard coded MiniMapper

Netlist

Canal Program

Archipelago

PnR'd Netlist

Bitstream Generation

CGRA Bitstream

Currently Being Used

Spec

Meta compiled

Hard coded

parameter independent

Halide Program

Halide Compiler

Buffer(s) CGRA

Kernel(s)

Clockwork

Hard Coded CGRAMapper

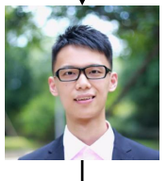
Hard Coded CGRAMapper

MemTiles

Mapped Kernel(s)

Scheduler

GB controllers



Global Buffer Bitstream

Mapped CGRA circuit

Garnet

Hard coded MiniMapper

Netlist

Canal Program

Archipelago

PnR'd Netlist

Bitstream Generation

CGRA Bitstream

Improved Flow!

Spec

Meta compiled

Hard coded

parameter independent

Halide Program

Halide Compiler

Buffer(s) CGRA

Kernel(s)

Peak Program

Clockwork

Hard Coded CGRAMapper

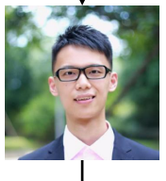
MetaMapper

MemTiles

Mapped Kernel(s)

Scheduler

GB controllers



Global Buffer Bitstream

Mapped CGRA circuit

Garnet

MetaMapper

Missing some features

Netlist

Canal Program

Archipelago

PnR'd Netlist

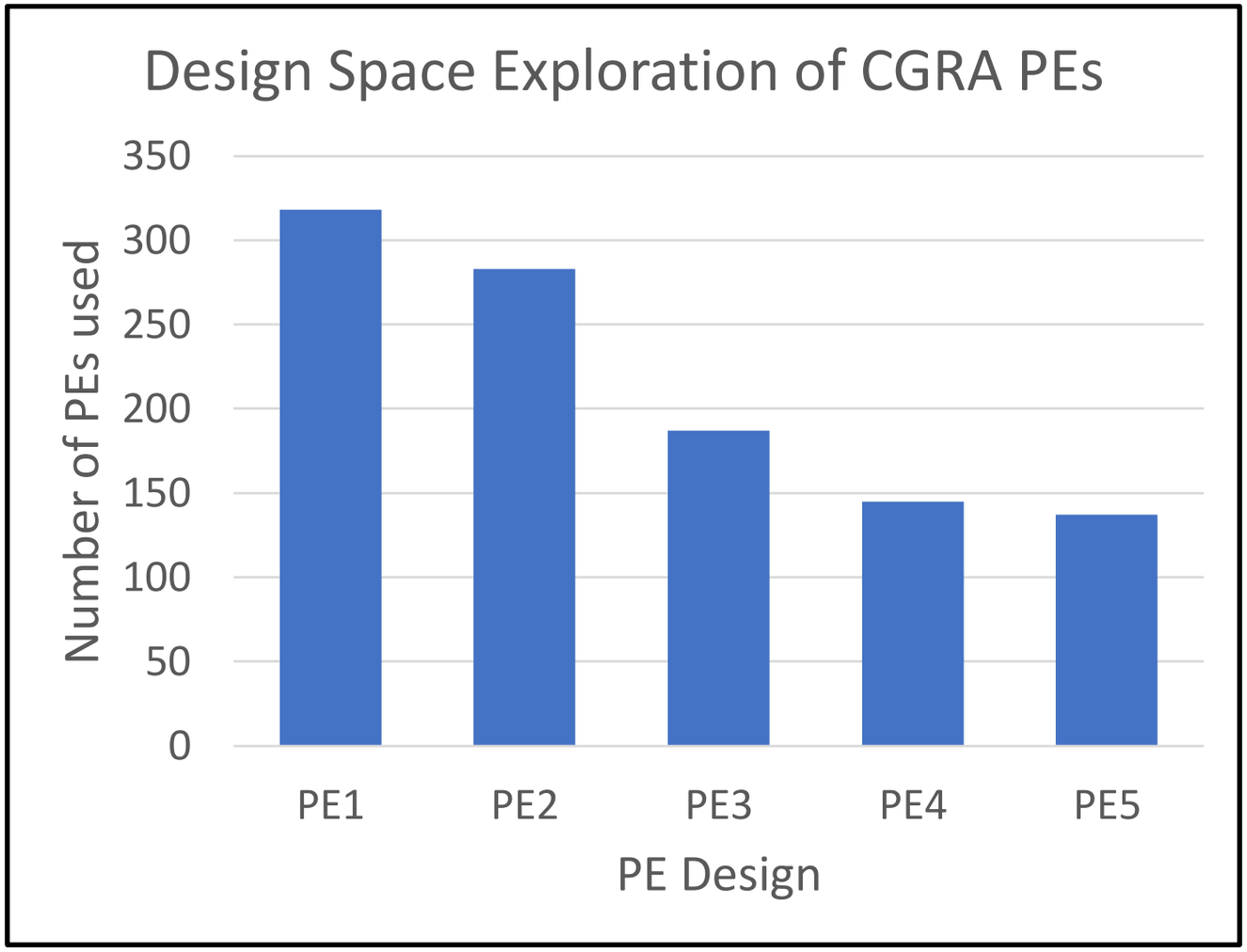
Bitstream Generation

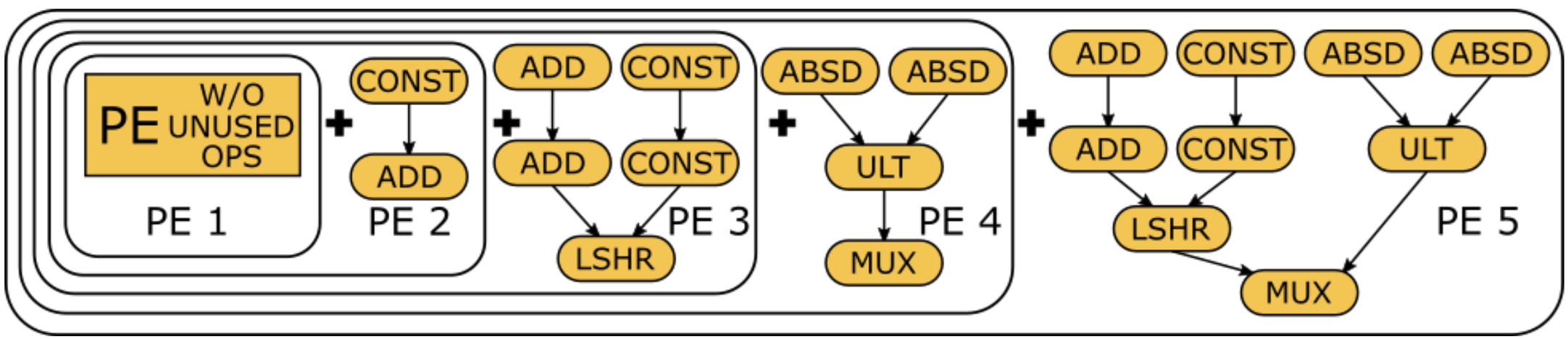
...ish

CGRA Bitstream

PE DSE with MetaMapper

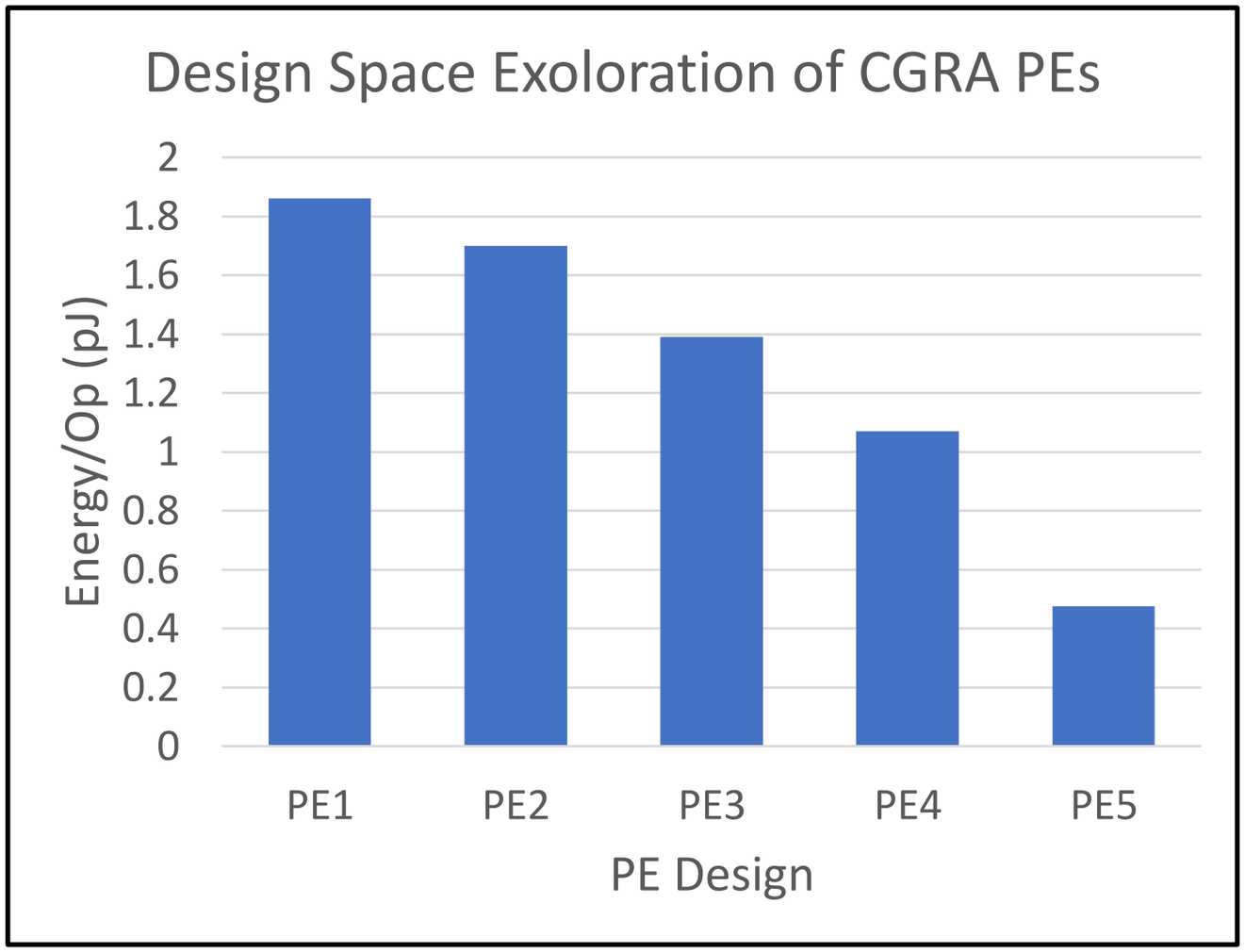
- We used MetaMapper to evaluate PE designs
- Specialized PEs to camera pipeline application
- Evaluated the number of PEs used to accelerate application

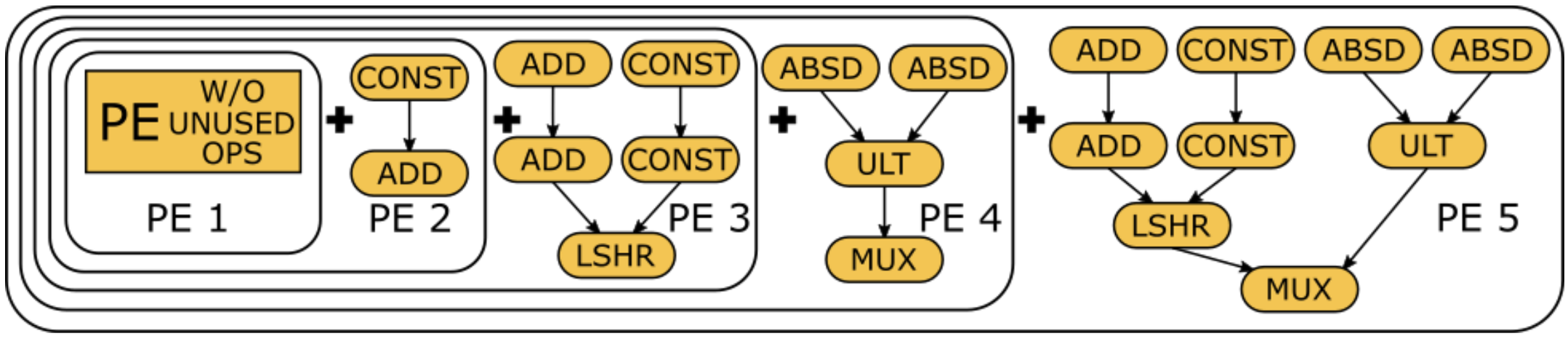




PE DSE with MetaMapper

- Mapped kernels were scheduled with clockwork
- Verified that the mapped application is correct
- CoreIR output can be compiled to Verilog





Thank you