Computer Architecture Review (Control Flow Architecture)

Instructor: Siting Liu

School of Information Science and Technology (SIST), ShanghaiTech University

Shanghai Institute of Semiconductor Innovation (SISI)

https://faculty.sist.shanghaitech.edu.cn/liust/EE219/EE219.html

Computer Architecture - EE219

faculty.sist.shanghaitech.edu.cn

立志成才报国裕民

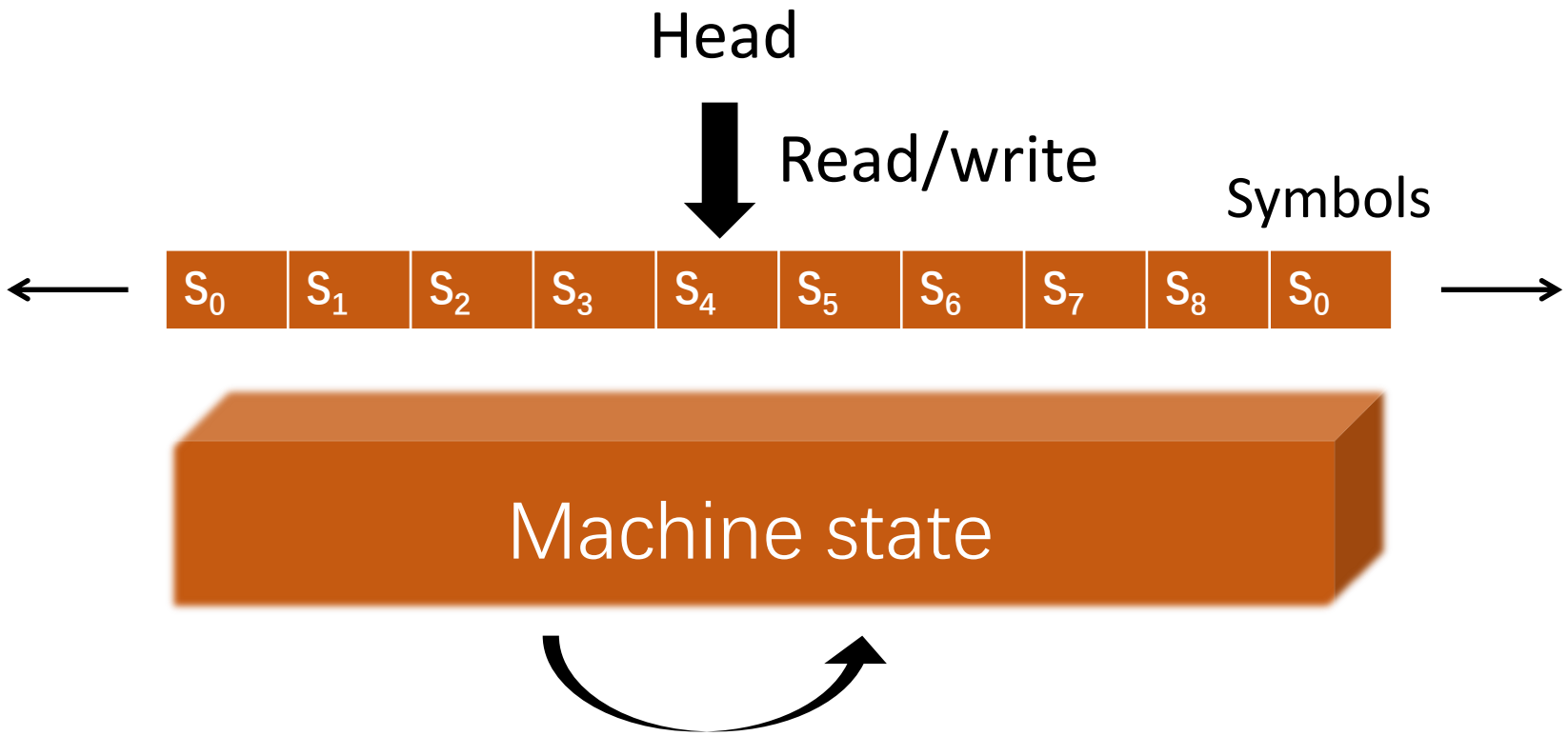
Objective

- Understand the design philosophy of conventional processors, e.g., CPU;
- Connect the algorithm to physical constraints of silicon;
- Integrate the accelerators we just designed into a computing system;
  - Make it a complete computer system, narrowed for NN workload;
  - Need software stack to make the design user-friendly;
  - Requires ISA as a hardware-software bridge;
- Study the techniques that can improve AI-chip design;



立志成才报国裕民

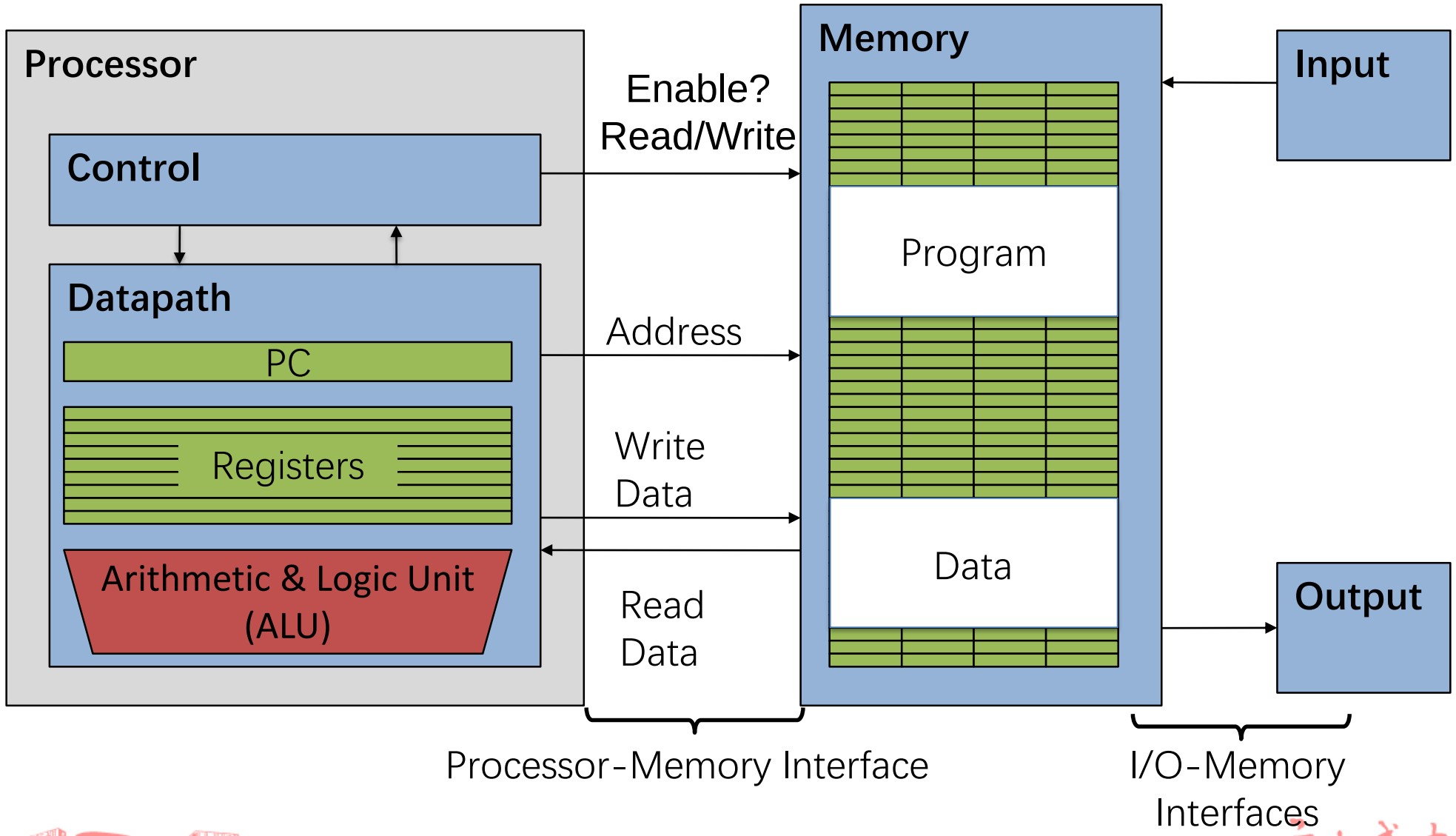
Control Flow Architecture – Turing Machine



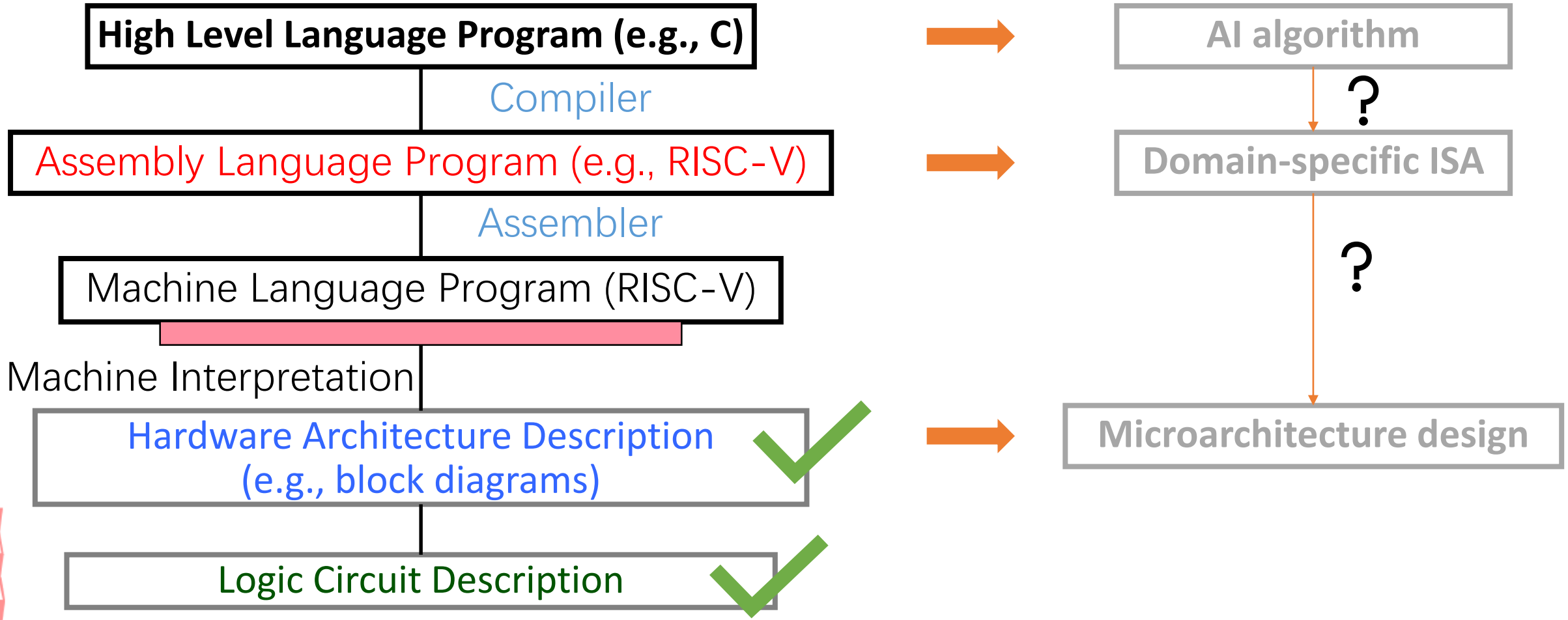
Update state according to the symbol being read and the current state

立志成才报国裕民

Von Neumann Architecture



How does CPU work to Run a Program?



立志成才报国裕民

A Simple CPU Design

- A digital circuit that understands and executes instructions
- Requires ISA as the design manual, it decides
  - How memory is managed, such as addressing, load/store;
  - How to encode/decode an instruction, i.e., decide the instruction format;
  - Registers, program counter and the other flags;



立志成才报国裕民

RISC-V for Example

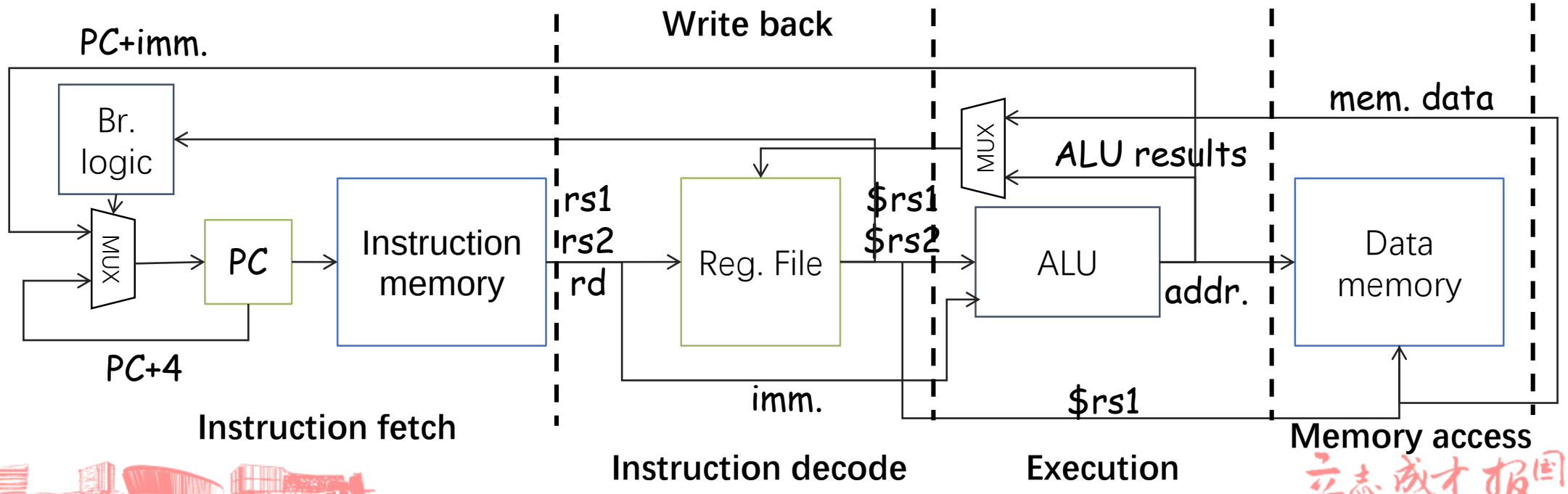
- Memory is byte-addressable; load-store architecture;
- Fixed-instruction length, with each field represents certain information;
  - R-type for variable arithmetic and logic operations;
  - I-type for variable-constant arithmetic and logic operations and **Load** operations;
  - S-type for **Store** and **branch** operations;
  - (U)J-type for jump operations;
- 32 general-purpose registers (reg. file, logical), indexed by x0-x31 with x0=0
- Use program counter to indicate the address of the instruction being executed.
- … …



立志成才报国裕民

A Simple CPU that Executes RISC-V Instructions

- To implement 4 instructions
  - add rd, rs1, rs2; lw rd, imm(rs1); sw rs0, imm(rs1); beq rs0, rs1, label
- Basic components for (1) instruction fetch; (2) instruction decode and operand fetch; (3) execution; (4) memory access (if present); (5) write back (if present); (6) should also update PC for next instruction fetch.



Evaluation of Performance – Iron Law

- Performance

$$\text{Time} = \frac{\#\ \text{Instruction}}{\text{Program}} \times \frac{\text{Instruction}}{\text{Cycle}} \times \text{Period}.$$

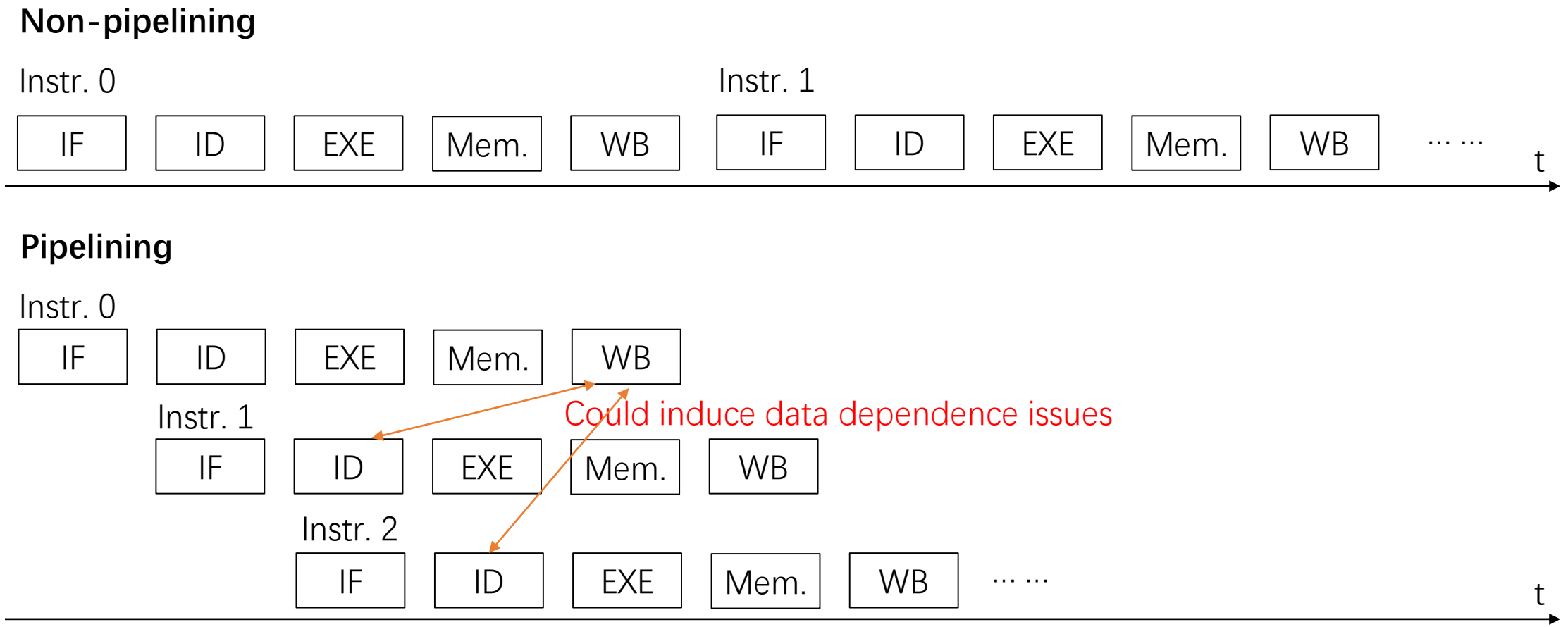
- In computer architecture, **instruction-level parallelism**, **data-level parallelism** and thread-level parallelism are employed to improve performance;



立志成才报国裕民

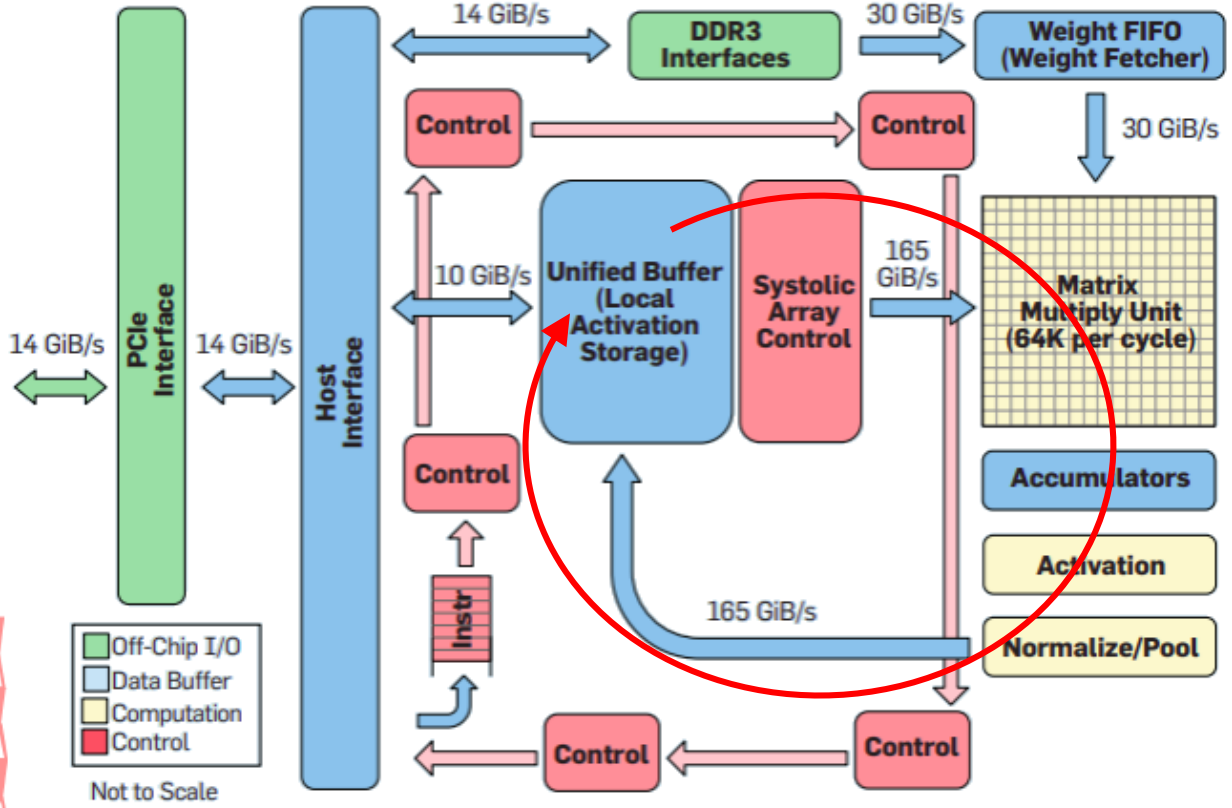
Instruction-level Parallelism (ILP)

- ILP can be achieved with pipelining and multi-issue



Pipelining in NN Accelerators (TPU)

- Early NN accelerators that follow the NN architecture to design the hardware (dataflow-based accelerator)

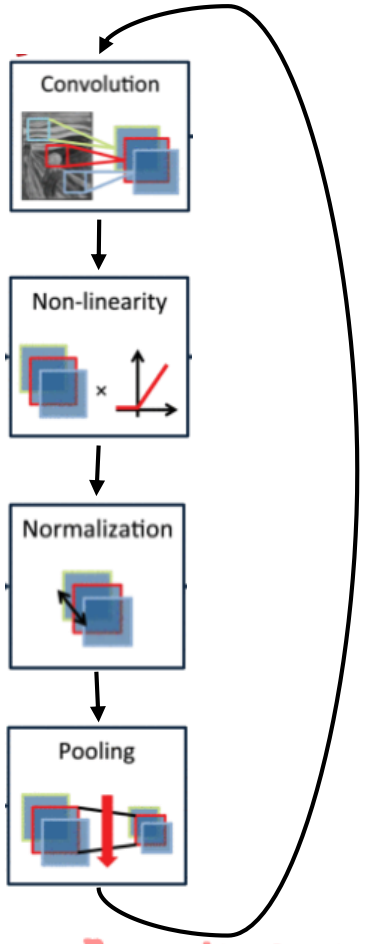


Tile N+1 loading

Tile N MMU

Tile N-1 accumulating

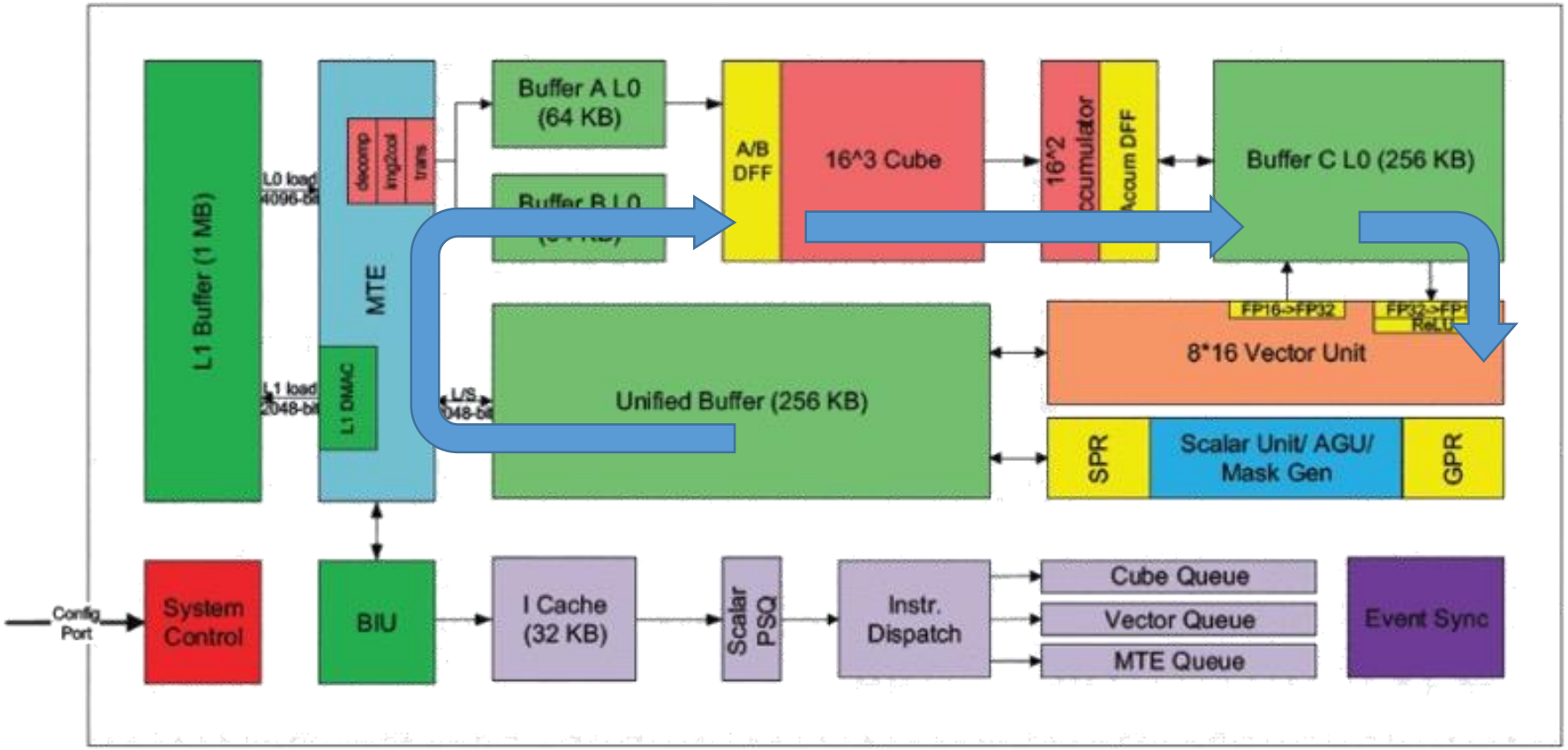
Tile N-2 activation



立志成才报国裕民

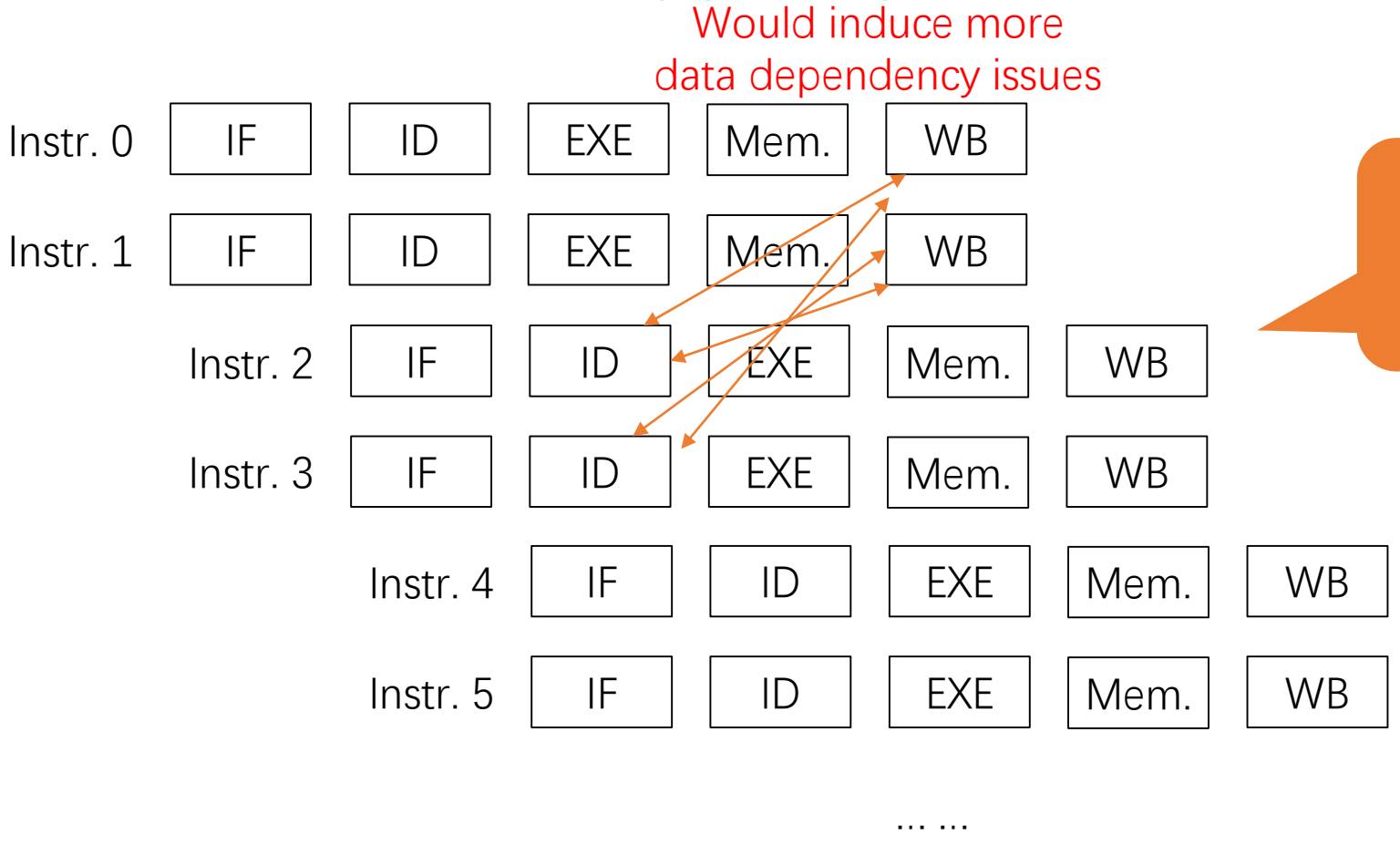
Pipelining in NN Accelerators (Ascend)

- Early NN accelerators that follow the NN architecture to design the hardware (dataflow-based accelerator)



Multi-issue

- ILP can be achieved with pipelining and multi-issue

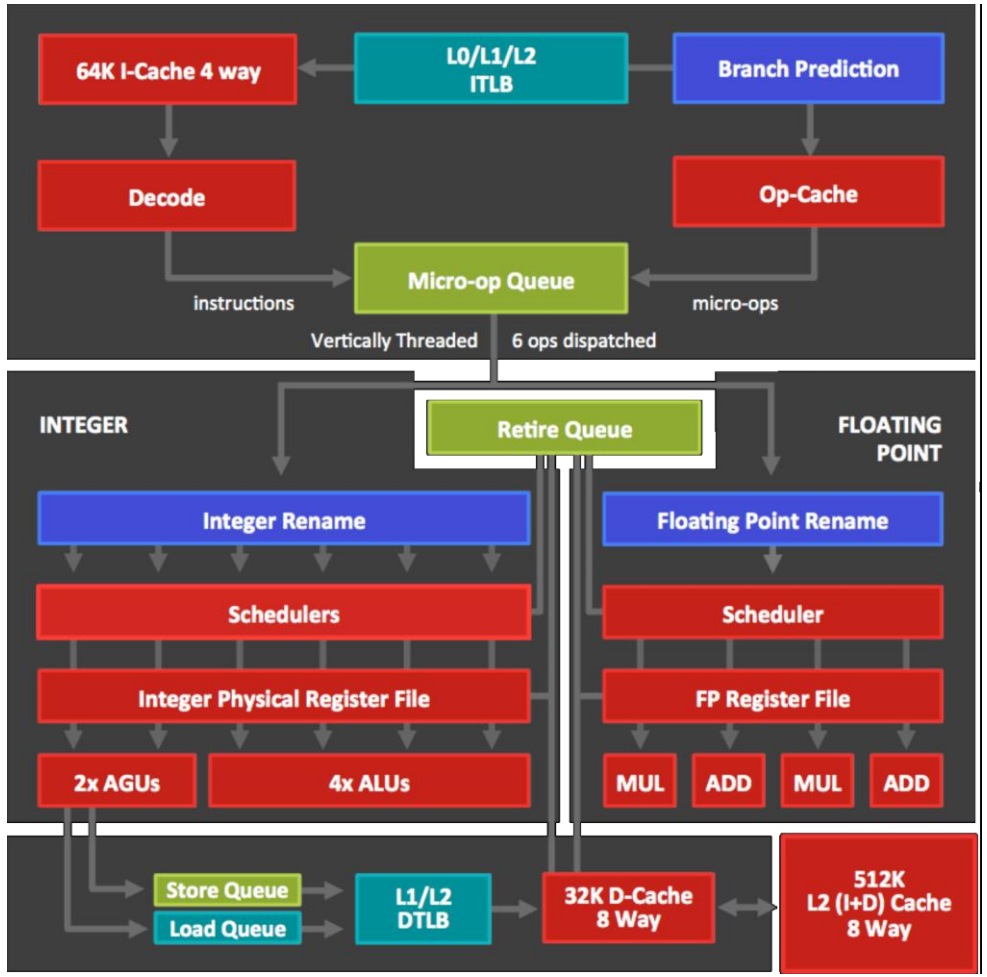


Who is responsible for scheduling the instructions?

立志成才报国裕民

Dynamic vs. Static Multi-issue

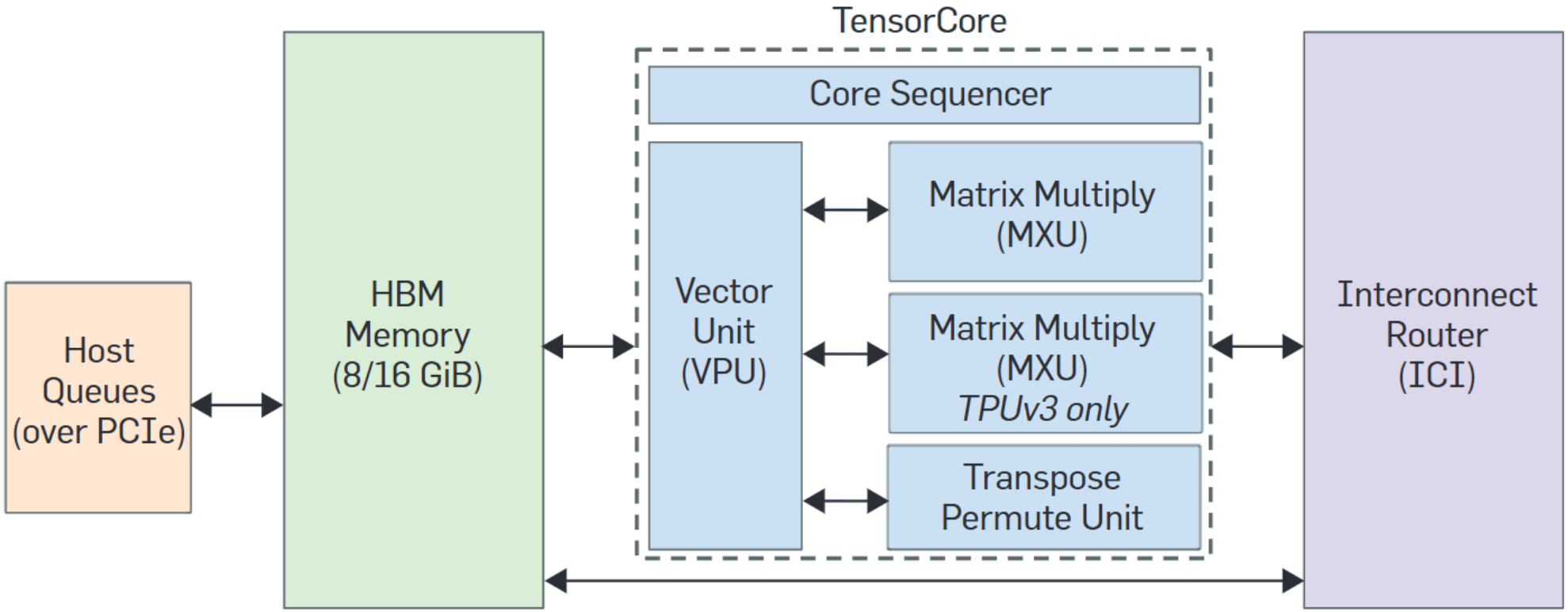
- Dynamic multi-issue
- Rely mainly on hardware to schedule the instructions, less on compiler.
- Example: superscalar.



Credit to AMD

Static Multi-issue – VLIW

- Rely mainly on compiler to schedule the instructions, less on hardware.
- Example: TPUv2, Groq tensor streaming processing (TSP) and beyond.



TPU Instruction Bundle

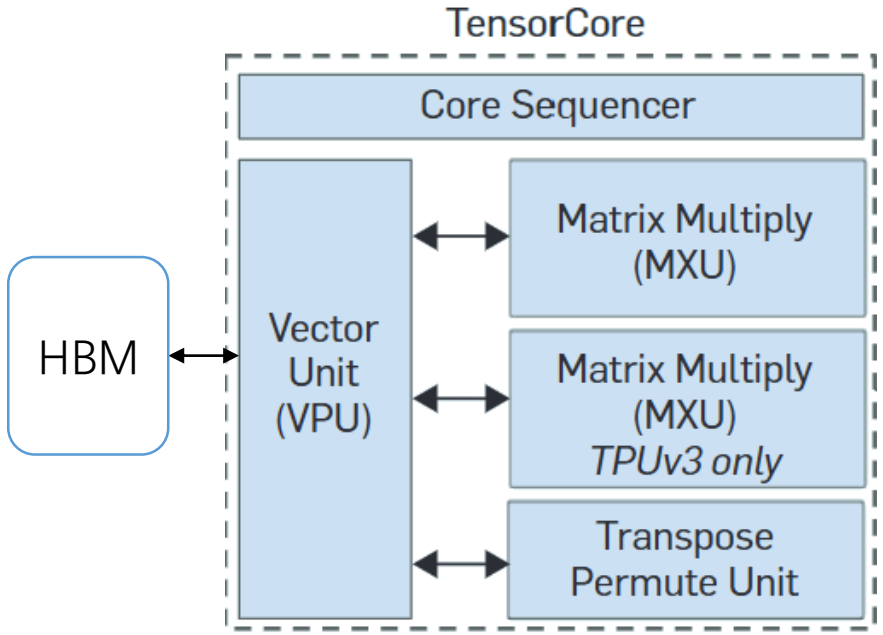
- Use standard VLIW compilation techniques including loop unrolling and instruction scheduling (when and where)

Scalar 0	Scalar 1	Vector ALU 0	Vector ALU 1	Vector LD/ST	Vector LD/ST	Matrix push	Matrix pop	Misc. (delay/Sync./Wait/DMA)	Imms.

- Loop unrolling

Load vector A[0]
Load vector B[0]
Add C[0], A[0], B[0]
Store vector C[0]

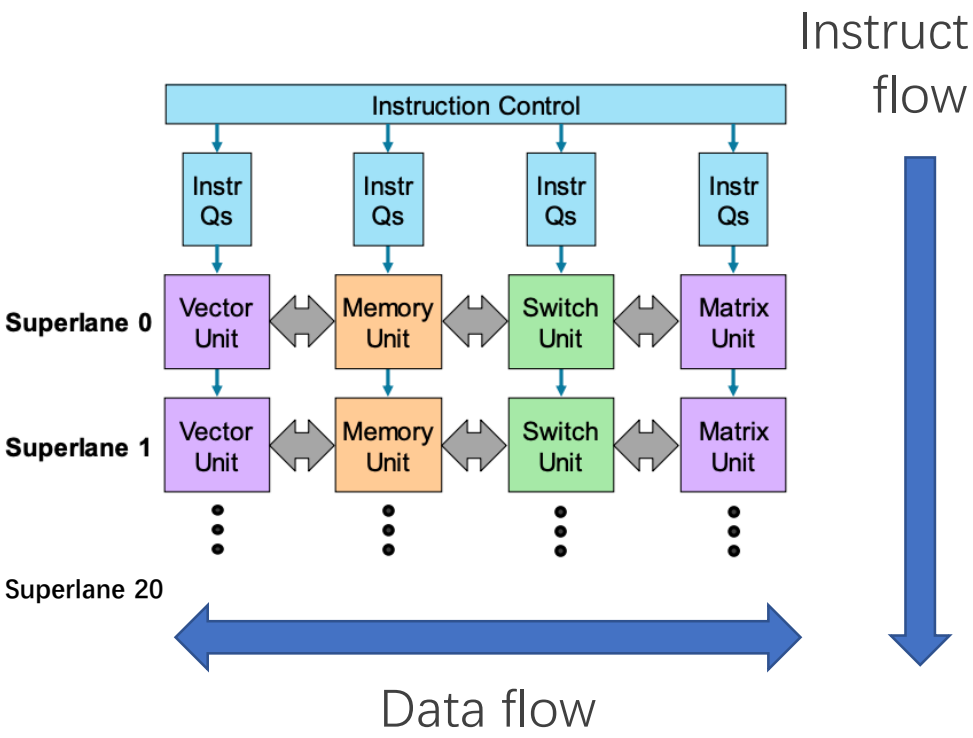
Load vector A[0]
Load vector B[0]
Add C[0], A[0], B[0]
Store vector C[0]
Load vector A[1]
Load vector B[1]
Add C[1], A[1], B[1]
Store vector C[1]



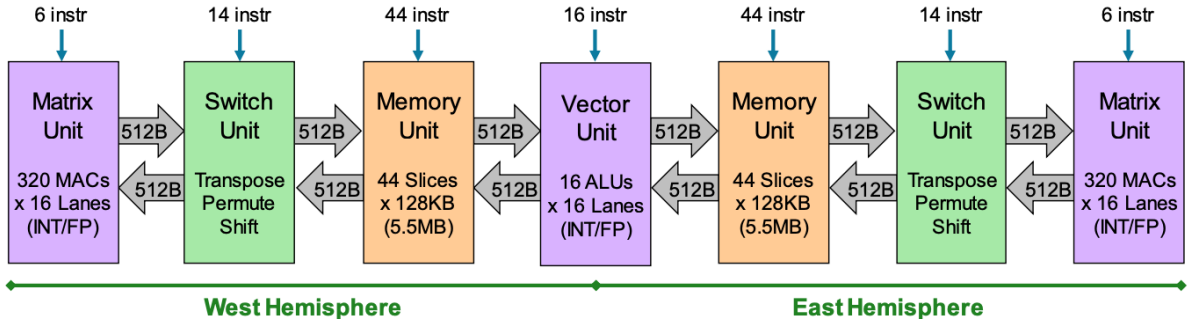
立志成才报国裕民

Groq TSP & LPU

- Groq tensor streaming processor (TSP), start-up by former TPU member
- VLIW, SIMD & systolic-like



144-wide VLIW architecture per superlane



- 16 lanes as a superlane (minimum granularity of computation)
- Memory instruction semantics have both an address and a dataflow direction
- SXM's permuter and lane shifters to move data elements vertically
- Unified stream of 512B per superlane both east- and westward

ISA Design

- Delayed **NOP** instruction for precise relative timing, inserted by compiler implicitly;
- **Sync** and **Notify** instructions to implement global barrier synchronization; **Notify** is broadcasted to all the other instruction queues parked on a **Sync** instruction; only required once after the chip resets (for each program);
- Memory access through **read** and **write** (direct addressing), and **gather** or **scatter** (stream-indirect addressing); bank conflict is avoided by compiler;
- **add_sat**, **add_mod**, **mul_sat**, **mul_mod** etc. for different vector arithmetic; allows multiple ALU operations without committing the intermediate results to memory;
- **Shift** up/down by select; reshape by **permute** and **transpose** in switch module;



立志成才报国裕民

On the Compiler Side

- Maximize functional slice utilization and minimize latency;
- Chain results from one slice (e.g. matrix unit) to the input of another slice (e.g. vector unit), eliminating **`read`** and **`write`** operations to store the intermediate results to memory slice;
- Expose low-level memory component (SRAM banks) for fine-grain memory management; each memory slice contains two banks, pseudo-dual-ported SRAM;
- Interleave memory banks to avoid pipeline fill/drain bubbles between layers; and distribute tensor across memory slices to avoid memory slice contention.



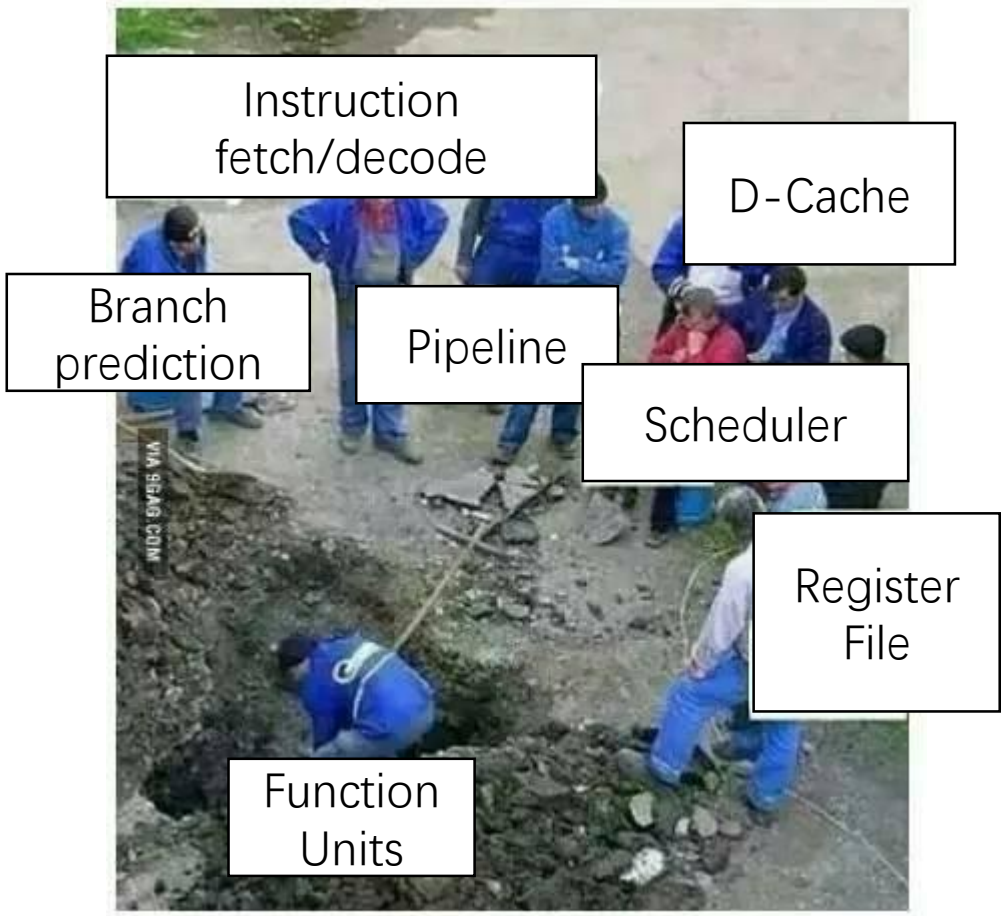
立志成才报国裕民

Source of Inefficiency in CPUs

Table 3. Datapath energy breakdown for our base implementation in mJ/frame. IF is instruction fetch/decode (including the I-cache). D-$ is the D-cache. Pip is the pipeline registers, buses, and clocking. Ctl is random control. RF is the register file. FU is the functional elements. Data estimates from processor simulations.

	IF	D-$	Pip	Ctl	RF	FU	Total
IME	410	218	257	113	113	68	1179
FME	286	196	205	90	90	54	921
Intra	54	20	29	13	13	8	137
CABAC	12	2	8	4	4	2	32
Total	762	436	499	220	220	132	2269

VLIW partially reduce this inefficiency.



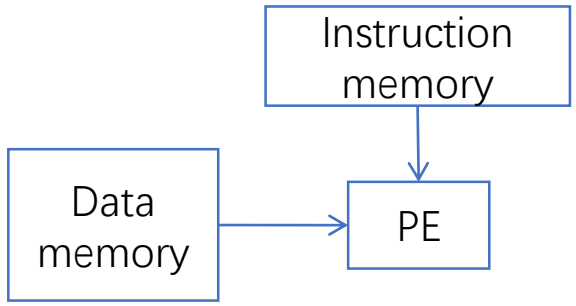
http://www.biaoqingb.com/gaoxiao/100640.html

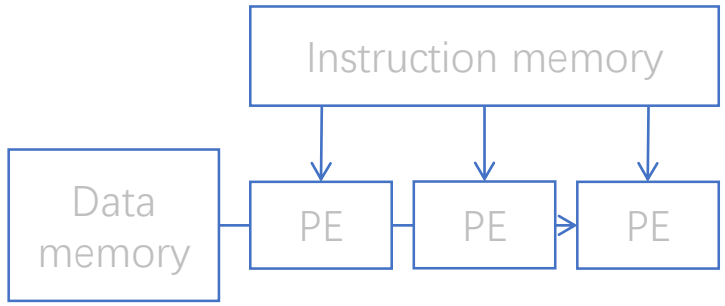
Understanding Sources of Inefficiency in General-Purpose Chips. ISCA'2010.

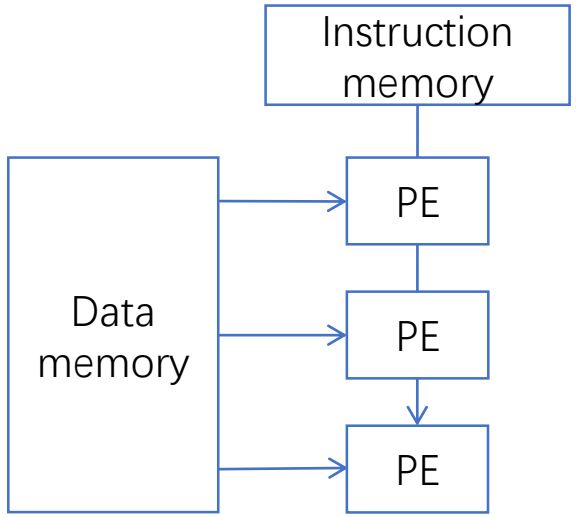
立志成才报国裕民

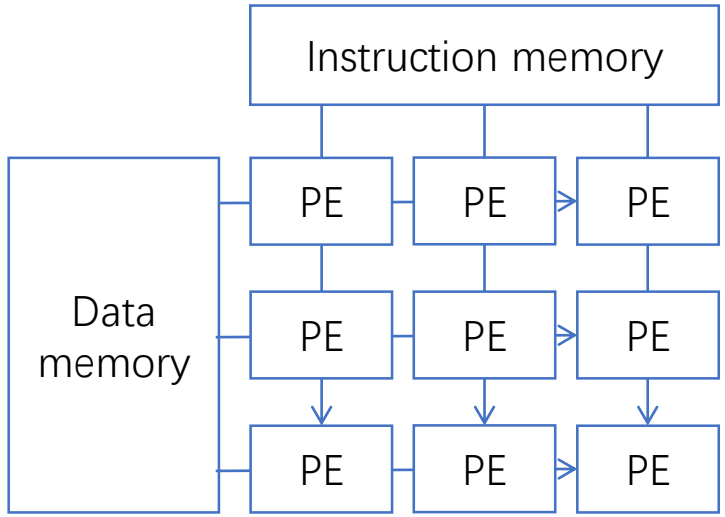
More Parallelism

- Flynn's Taxonomy
- SISD (scalar)
- **SIMD (data-level parallelism)**
  - **Requires that no dependency exists between data!**
  - Vector architectures
  - Multimedia extensions
  - GPU (SIMT)
- MISD
- MIMD









立志成才报国裕民

SIMD – Vector Extension in RISC-V

- ISA extension
  - Vector registers
  - Additional parameters



Vector Extension – Other Important Parameters

- Vector Length Agnostic (VLA)
- SEW: standard element width, the bit width of each element, can be {8, 16, 32, 64};
- LMUL: vector length multiplier, maximum number of vector registers used for one operation, can be {1/8, ¼, ½, 1, 2, 4, 8};
- VL: vector length, number of effective vector elements, $VL \leqslant VLMAX = VLEN * LMUL/SEW$
- Use **vsetvli** to automatically setup VL.



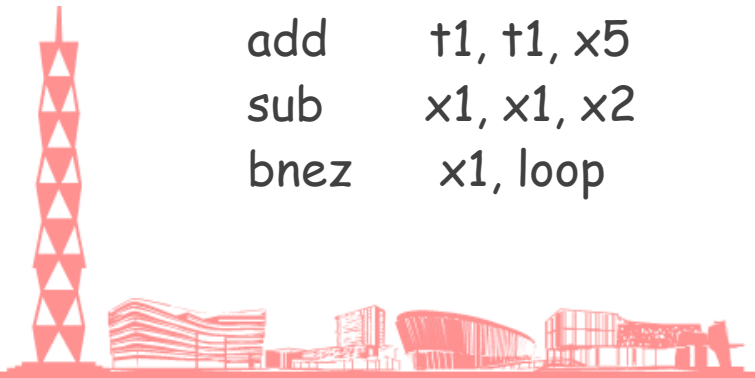
立志成才报国裕民

Vector Extension – Example

$$B[i] = B[i] + A, \qquad 0 \le i < N$$

- Assume *A* locates at (**x4**), *B* locates at (**t1**), total vector length *N* = **$x1**

```
flw      ft0, 0(x4)          # load A
loop:
   vsetvli  x2, x1, e32, m1, ta, ma # e32, m1, SEW=32, LMUL=1; set VL, x2; ma ta mask/tail agnostic
                               # x2/VL = min(VLMAX, $x1), hardware knows what to do
   vfmv.v.f  v1, ft0           # broadcast A to vector register v1
   vle32.v   v0, (t1)          # load B[i : i+VL-1]
   vfadd.vv  v2, v0, v1        # v2[i] = v0[i] + v1[i]
   vse32.v   v2, (t1)          # store the update B[i : i+VL-1]
   slli      x5, x2, 2         # calculate the storage consumed, VL * 4
   add       t1, t1, x5        # update the vector pointer
   sub       x1, x1, x2        # update the remaining number of elements
   bnez      x1, loop          # continue with the remaining elements
```



立志成才报国裕民

Vector Extension vs. Scalar Machine

$$B[i] = B[i] + A, \qquad 0 \le i < N$$

- Assume *A* locates at (**x4**), *B* locates at (**t1**), total vector length *N* = **$x1**

```
flw      ft0, 0(x4)
loop:
   vsetvli  x2, x1, e32, m1, ta, ma

   vfmv.v.f  v1, ft0
   vle32.v   v0, (t1)
   vfadd.vv  v2, v0, v1
   vse32.v   v2, (t1)
   slli      x5, x2, 2
   add       t1, t1, x5
   sub       x1, x1, x2
   bnez      x1, loop
```

Scalar assembly

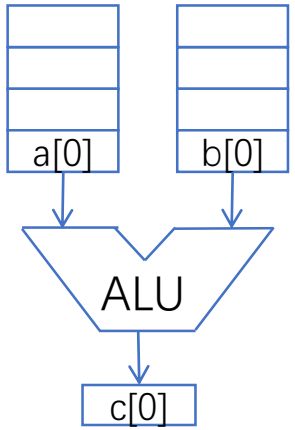
```
flw      ft0, 0(x4)      # Load scalar A
li       x2, 0           # i = 0
loop:
   bge      x2, x1, done    # If i >= N, exit
   flw      ft1, 0(t1)      # Load B[i]
   fadd.s   ft1, ft1, ft0   # B[i] + A
   fsw      ft1, 0(t1)      # Store result to B[i]
   addi     t1, t1, 4       # Advance pointer by 4 bytes
   addi     x2, x2, 1       # i++
   j        loop
```

立志成才报国裕民

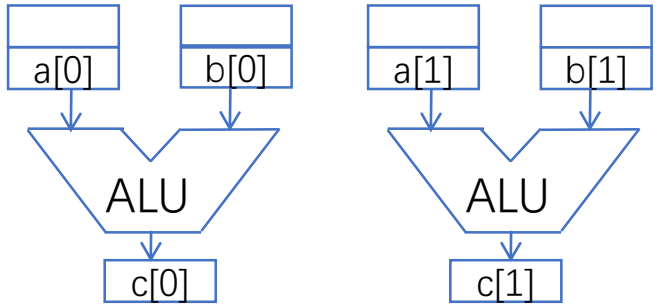
Vector ALU

- Deeper/wider pipeline/ALU

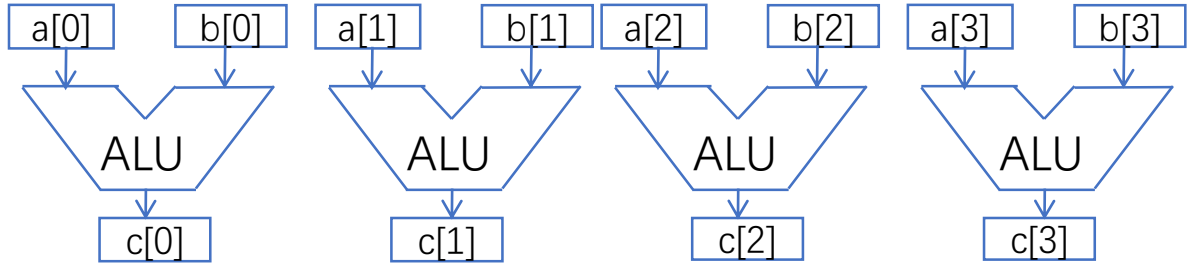
Serial, pipelined



Mixed, multi-lane



Fully parallel

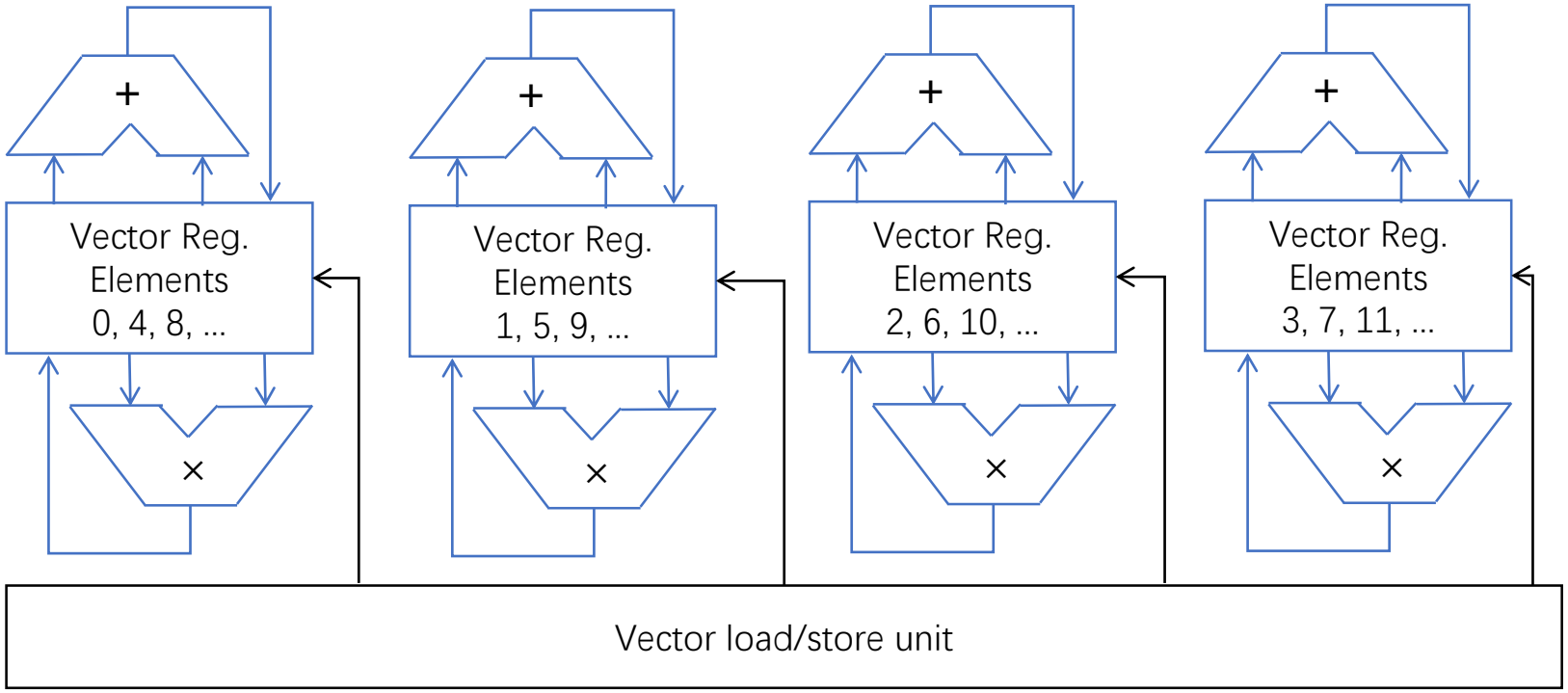


Serial	Mixed	Parallel
Latency		
Bandwidth		
Hardware		

立志成才报国裕民

Vector ALU

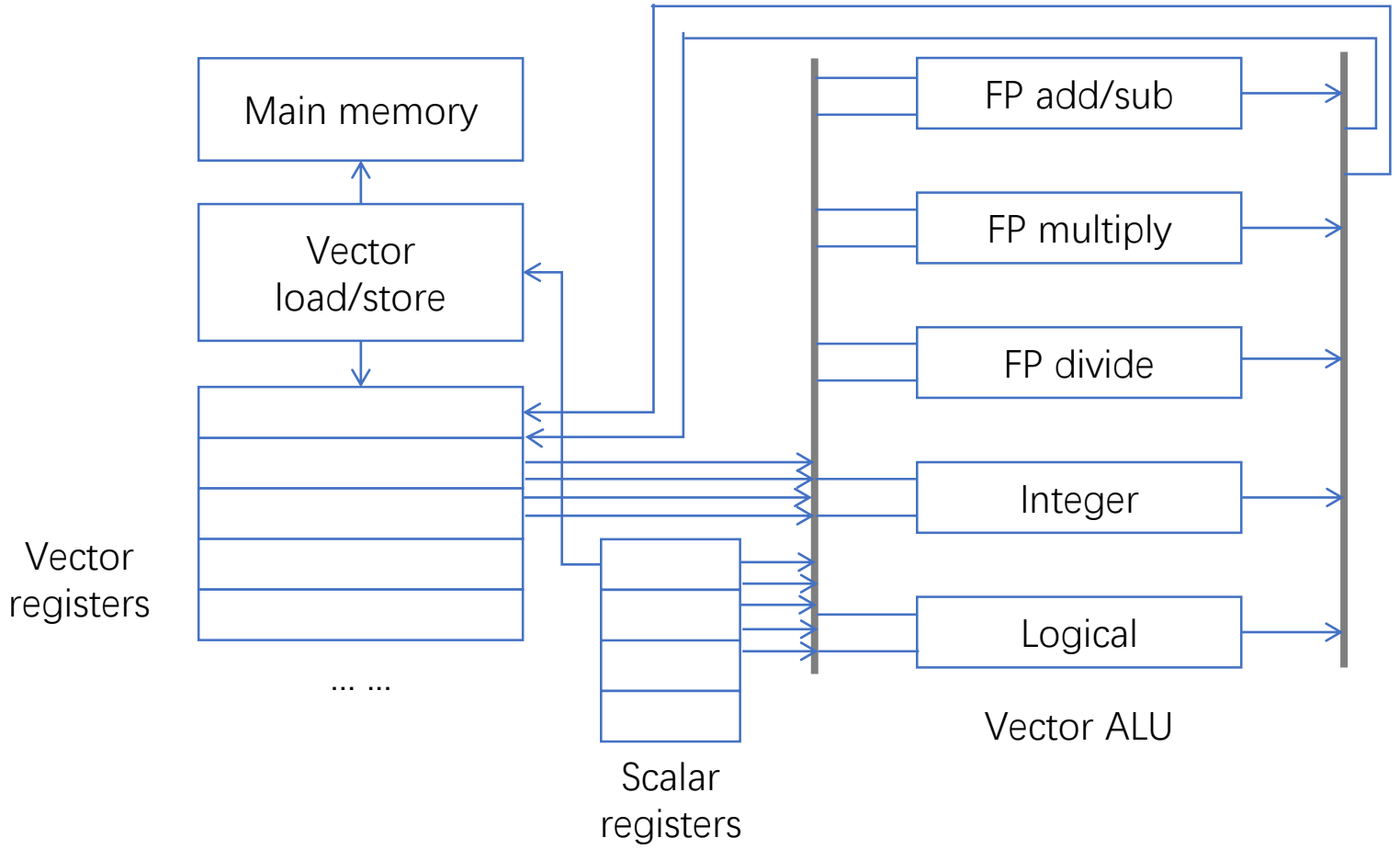
- Can also allocate vector registers to different ALUs




```
# Integer multiply-add
# overwrite addend
vmacc.vv vd, vs1, vs2, vm
# vd[i] += (vs1[i] * vs2[i])
vmacc.vx vd, rs1, vs2, vm
# vd[i] += (x[rs1] * vs2[i])
```

立志成才报国裕民

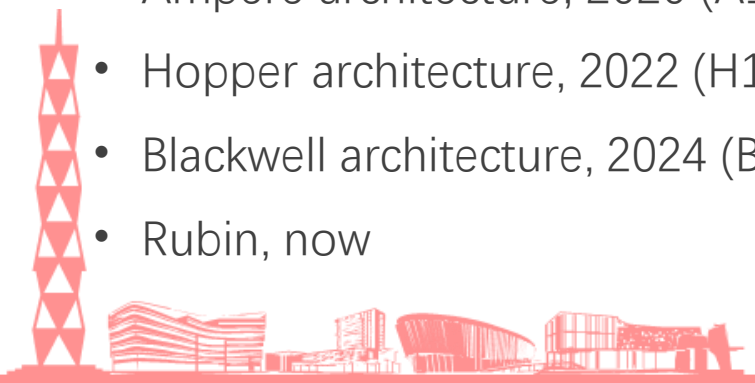
A Possible Vector Processor Implementation



立志成才报国裕民

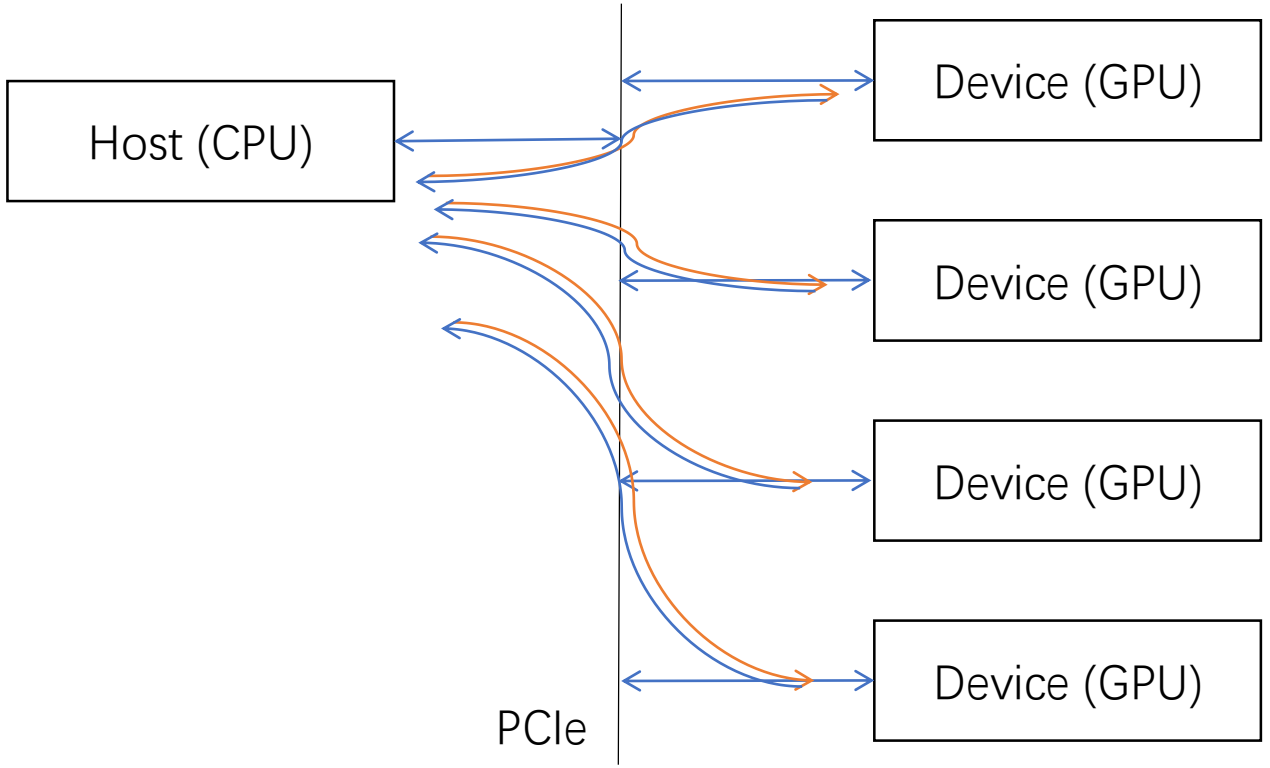
GPU Architecture – History

- 1993, Jen-Hsun Huang co-founded Nvidia;
- NV1 (3D game, 2MB VRAM, ST microelectronics 1995), NV2, NV3 (AKA, RIVA 128/128ZX, polygon-based redering, 2D/3D, 1997 4/8MB), NV4, NV5, …
- First graphic card to call “GPU”, GeForce 256, 1999, IPO at 12$/share
- First programmable GPU GeForce3, 2001
- SLI (Scalable Link Interface), enable mutli-GPU connection, 2004
- CUDA, 2006
- Fermi architecture, 2009
- Kepler architecture, 2012 (AlexNet appears the same year)
- Ampere architecture, 2020 (A100)
- Hopper architecture, 2022 (H100)
- Blackwell architecture, 2024 (B200/GB200)
- Rubin, now



立志成才报国裕民

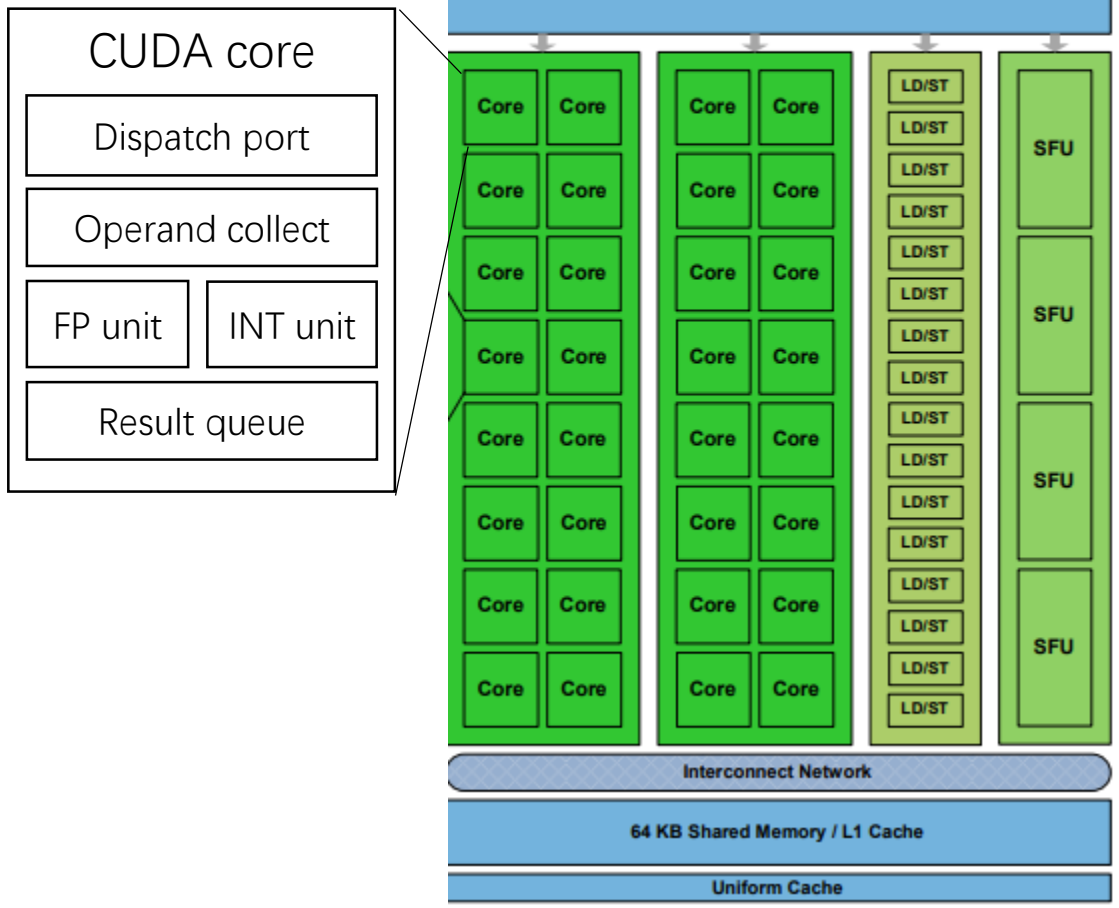
GPU Work Flow



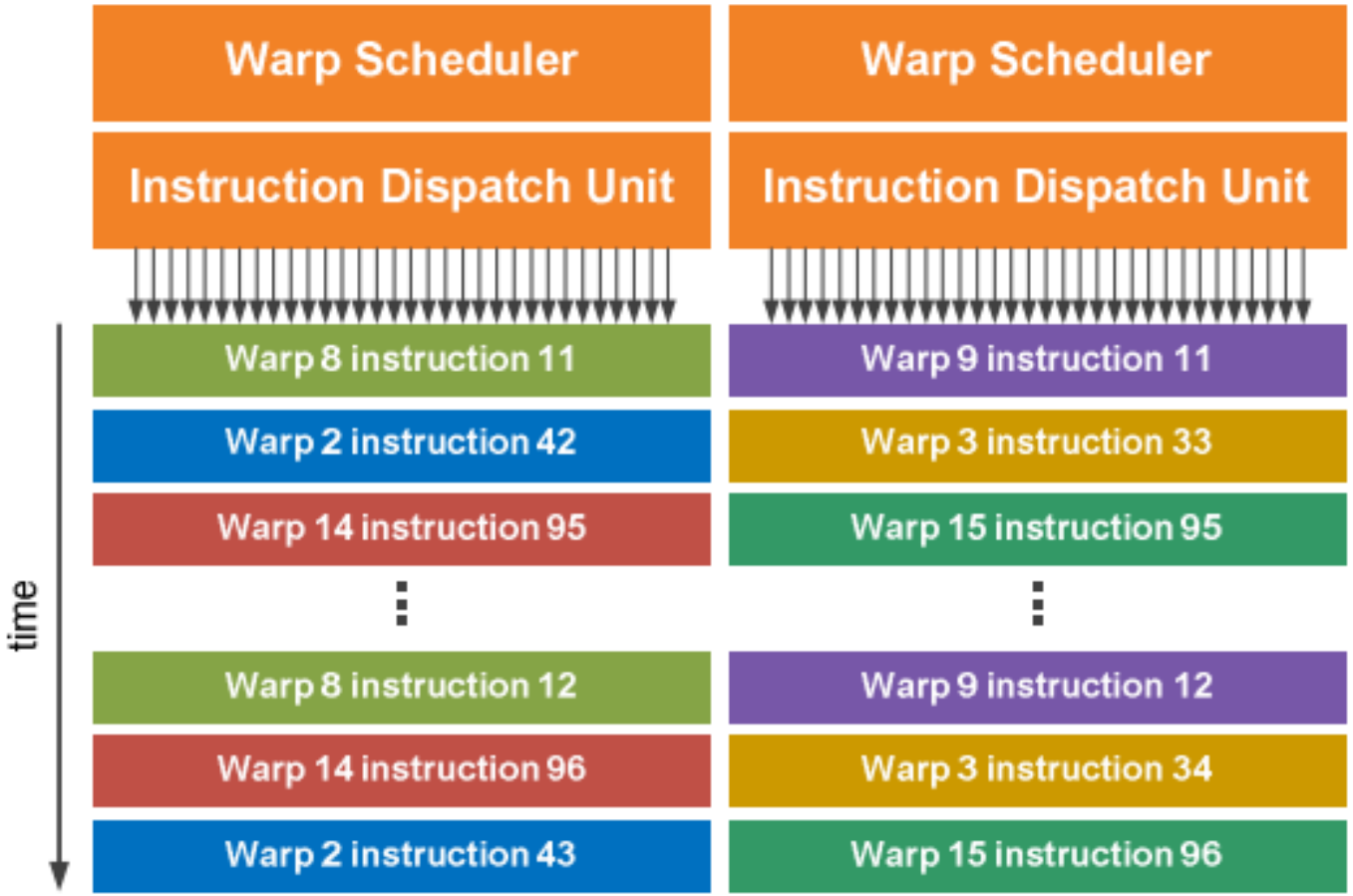
立志成才报国裕民

Fermi Architecture

- The SM schedules threads in groups of 32 parallel threads called warps (thread of vector instructions)
  - 2 warp schedulers
  - 2 instruction dispatch units
  - Dispatch 1 instruction for each warp of
    - 16 cores
    - 16 load/store units
    - 4 SFUs
  - Fermi's scheduler does not need to check for dependency
    - 2 INT instructions/2 FP instructions
    - 1 mixed INT/FPU/load/store/SFU instruction
    - Double instruction does not support dual dispatch



Warp Scheduler



Because threads of SIMD instructions are independent, the scheduler may select a different SIMD Thread each time.

CUDA (Compute Unified Device Architecture)

- Enable NVIDIA GPUs to execute programs written in C, C++, Fortran, OpenCL, etc.
- CUDA's hierarchy of threads

CUDA program

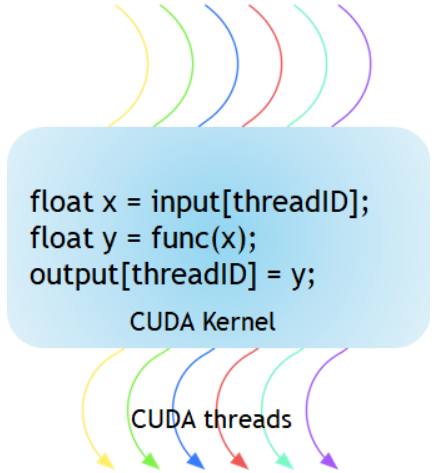
Kernels

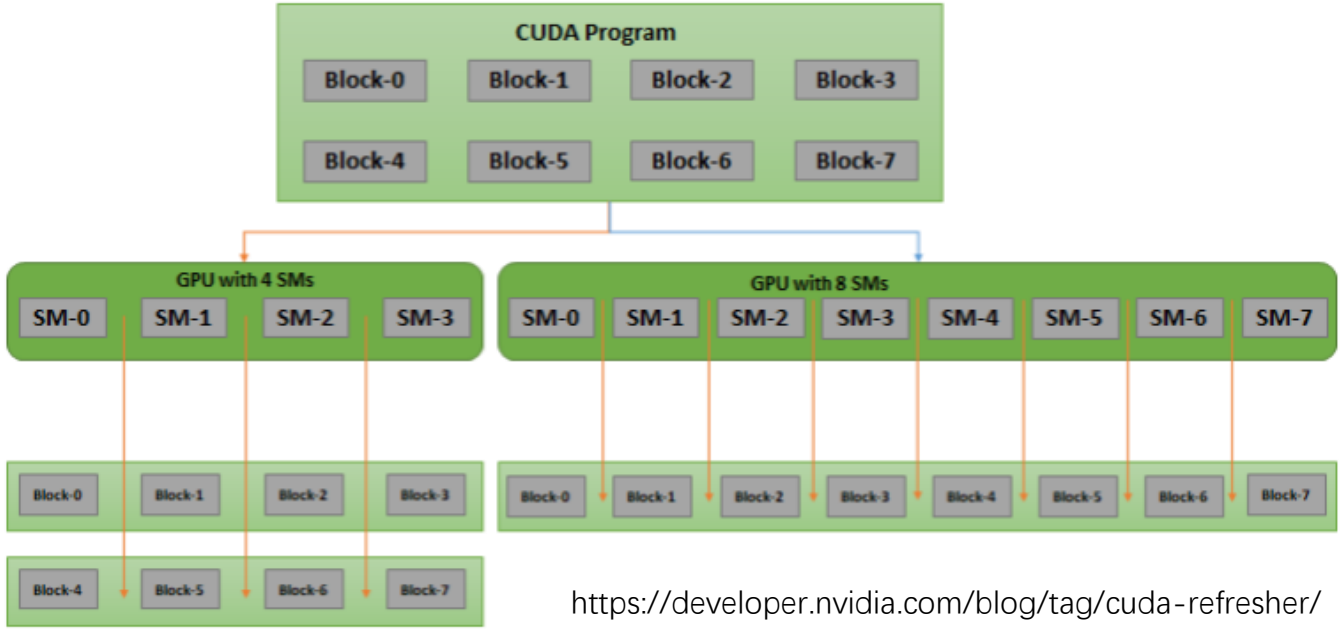
Grids

Blocks

Threads

```
/** CUDA kernel device code - CUDA Sample Codes
 * Computes the vector multi. of A and B into C. The three vectors have the same number of elements asnumElements.
 */
__global__ void vectorMulti( float *A, float *B, float *C, int numElements) {
  int i = blockDim.x * blockIdx.x + threadIdx.x;
  if (i < numElements) {
    C[i] = A[i] * B[i];
  }
}
```



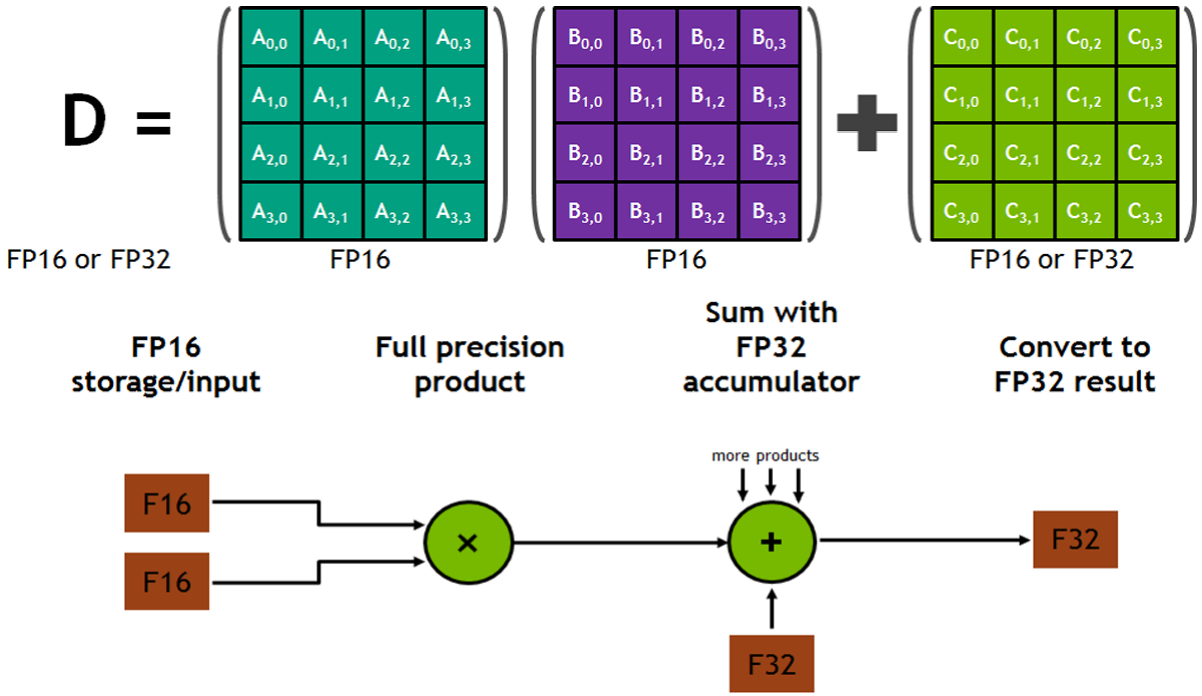


https://developer.nvidia.com/blog/tag/cuda-refresher/

立志成才报国裕民

Recent NVIDIA GPUs

- Volta architecture with tensor core



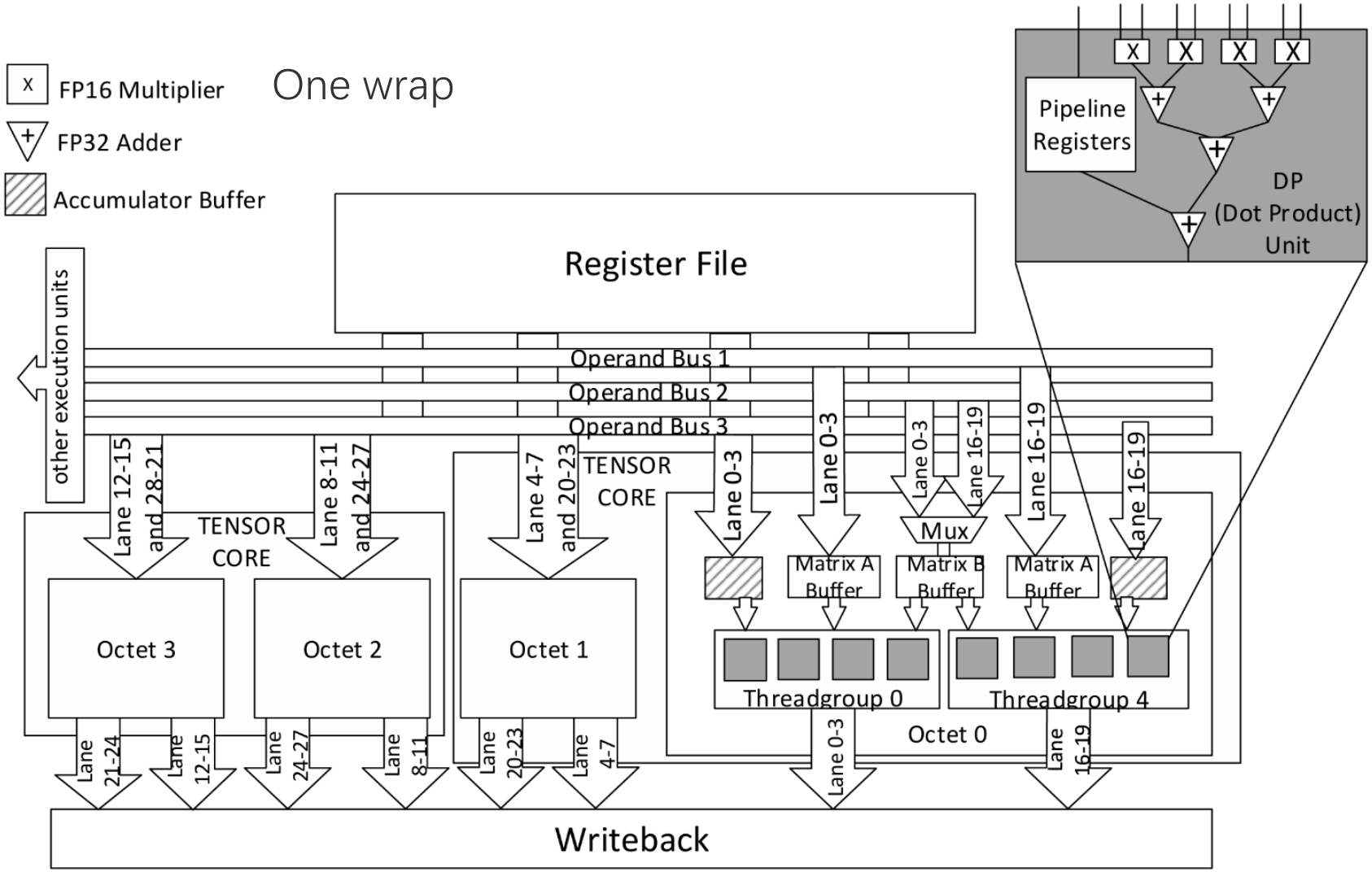


- **wmma** API for tensor core programming, e.g.

```
wmma::load_matrix_sync()
wmma::mma_sync()
wmma::store_matrix_sync()
```

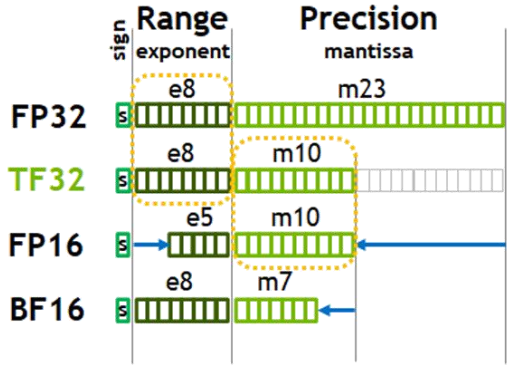

立志成才报国裕民

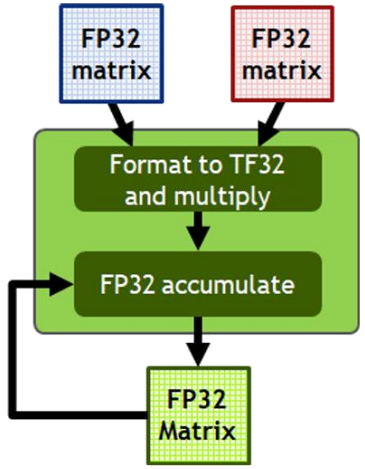
Demystifying Tensor Core



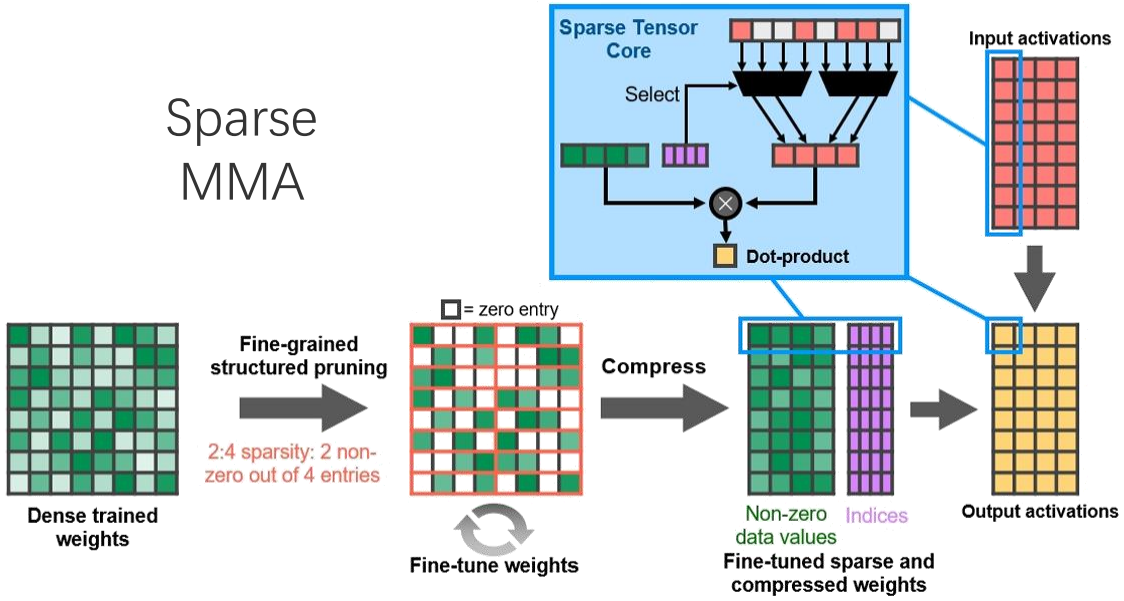
M. A. Raihan, N. Goli and T. M. Aamodt, "Modeling Deep Learning Accelerator Enabled GPUs," *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Madison, WI, USA, 2019,

NVIDIA A100 Architecture





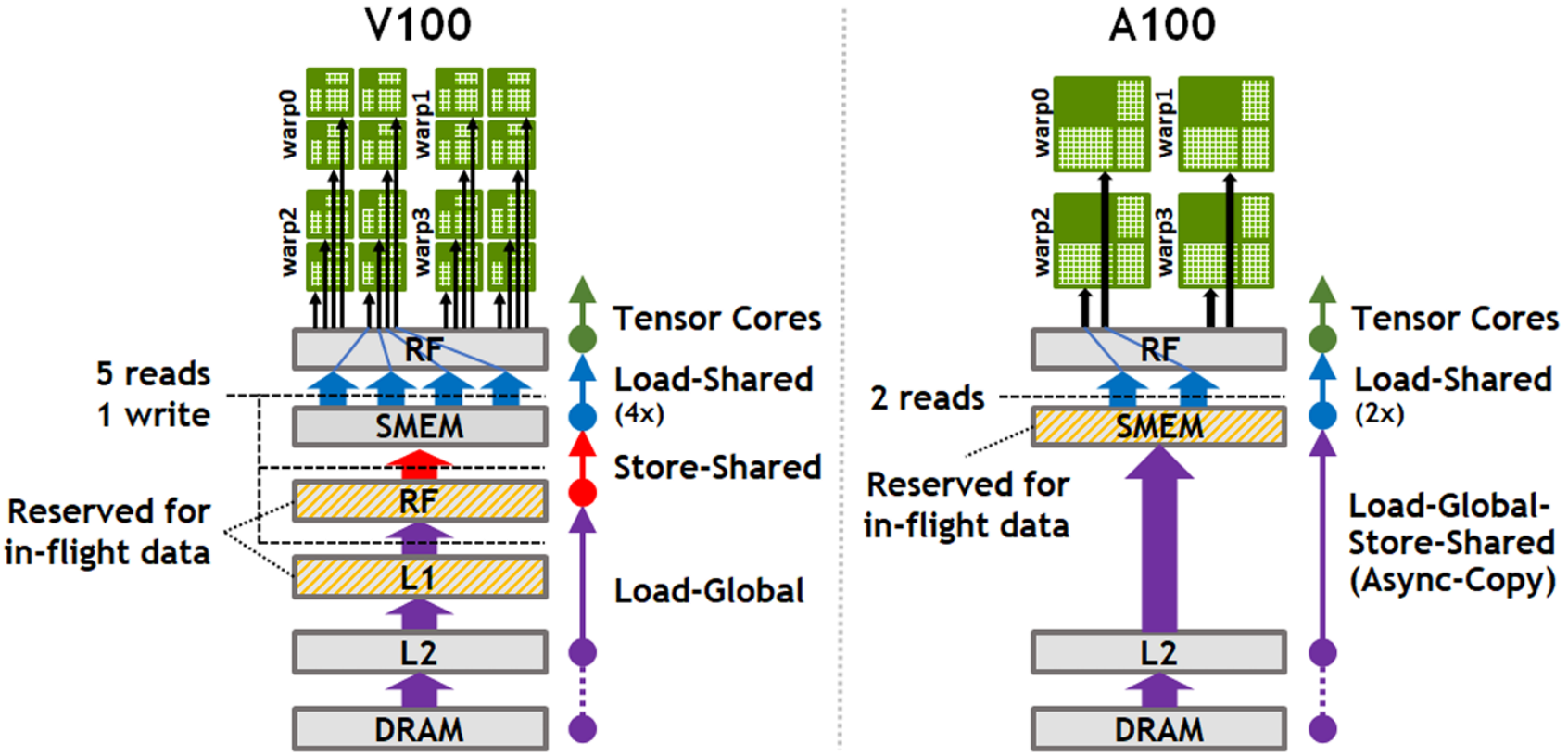
Sparse MMA



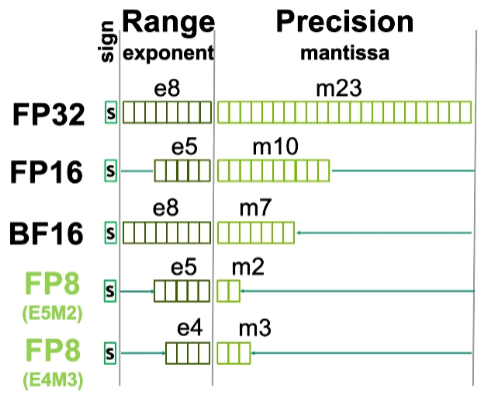


NVIDIA A100 Architecture – More Data Reuse

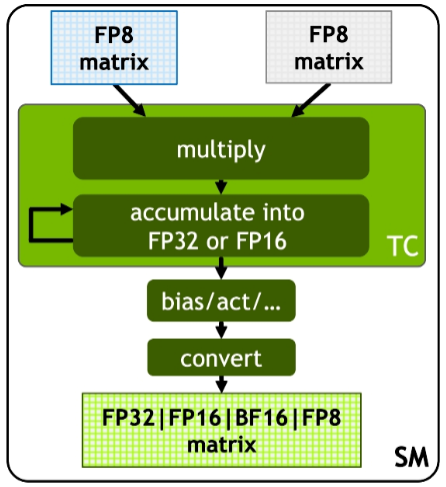
- Enable async. read to the shared memory;
- Data shared across all 32 threads in a warp, compared to 8 threads on Volta's tensor core.



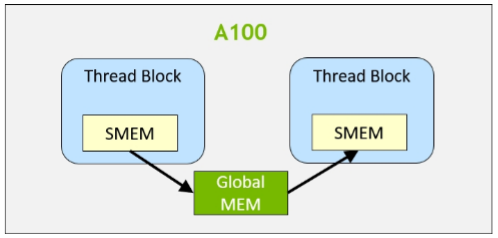
NVIDIA H100 Architecture

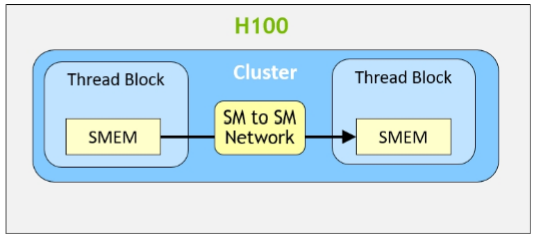


Allocate 1 bit to either range or precision



Support for multiple accumulator and output types





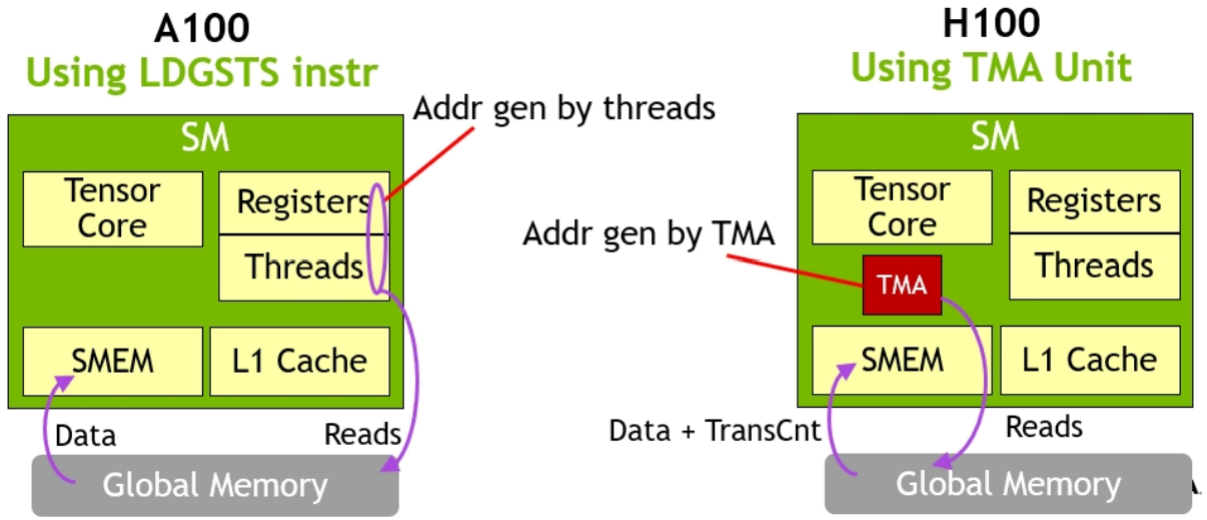
Distributed SMEM

CUDA Programming Model Exposure	A100	New for H100
Barrier.arrive(),	Asynchronous Barrier	**Asynchronous Transaction Barrier**
Barrier.wait()	Waiter spins in SMEM	**Waiter sleeps until all threads arrive**
Memcpy_async()	Direct copy to SMEM	**Asynchronous mem copy unit (called TMA)**

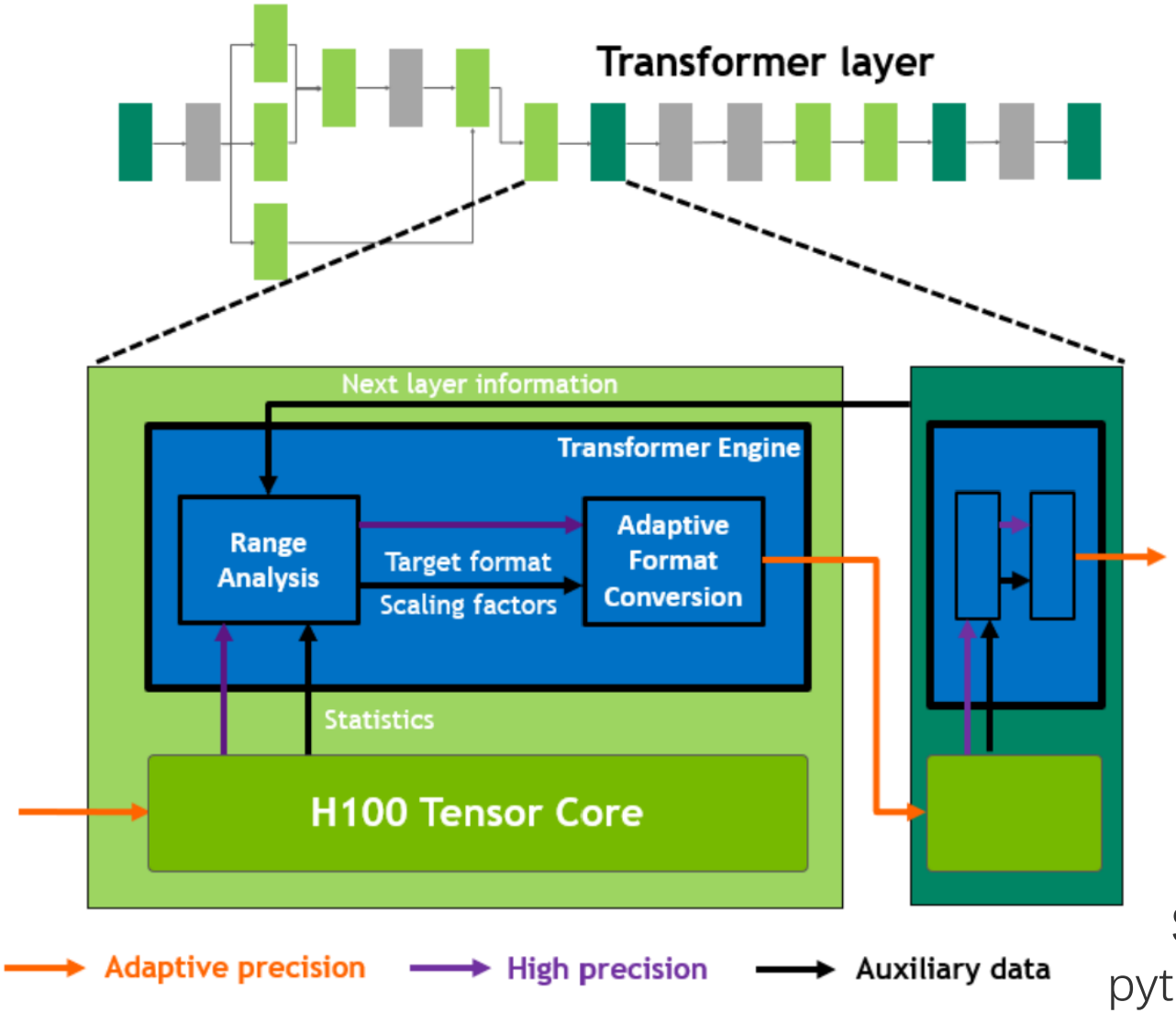


NVIDIA H100 Architecture – TMA

- Tensor memory accelerator



NVIDIA H100 Architecture – Transformer Engine



Supported by JAX and pytorch, details not revealed

Summary

- Old wisdom can be applied to modern AI accelerator design
  - ILP (pipeline and multi-issue/VLIW)
  - DLP (SIMD)



立志成才报国裕民