Green Computing Platforms

Step2. CA1001: Vector accelerator and GPU

http://archlab.naist.jp/Lectures/ARCH/ca1001/ca1001e.pdf

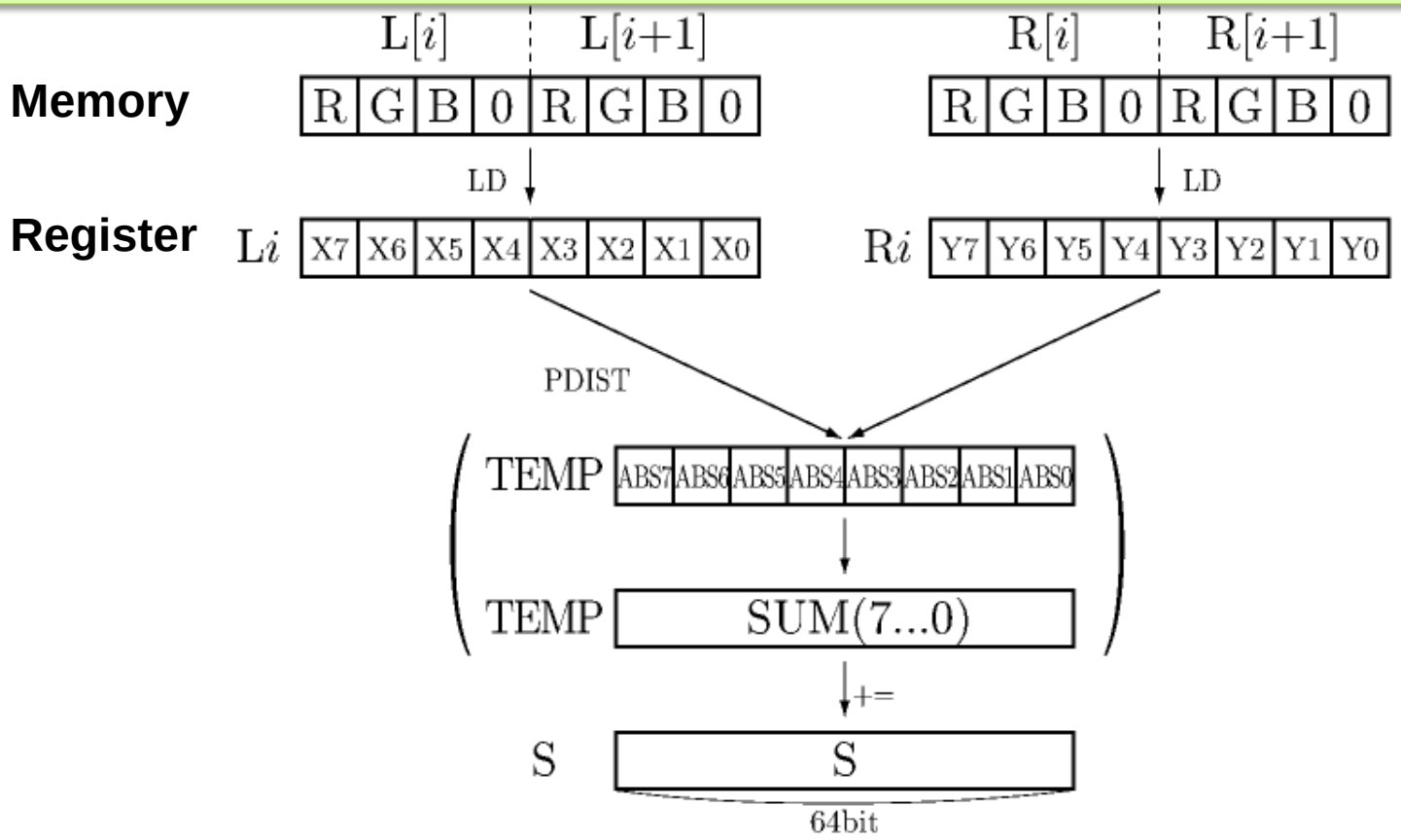
Download the template and submit through UNIPA.

http://archlab.naist.jp/Lectures/ARCH/ca1001/ca1001e.docx

in http://archlab.naist.jp/Lectures

Short Vector

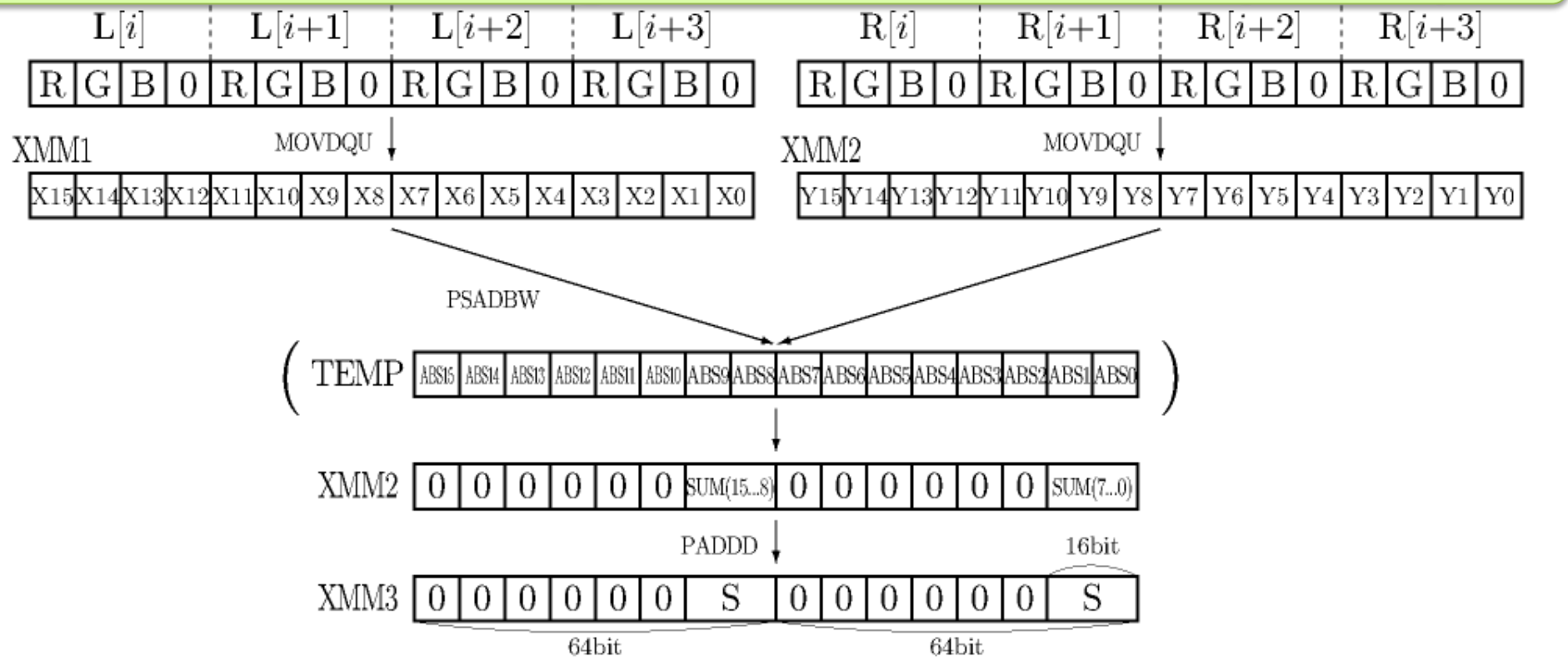
SPARC: Visual Instruction Set (VIS)



- **PDIST: accumulate SAD to 3'rd operand**

```
LD      L[i], L[i+1]  → Li（8バイトロード）
LD      R[i], R[i+1]  → Ri（8バイトロード）
PDIST   Li, Ri, S     → S （8バイトSAD命令）
```

intel: Streaming SIMD Extensions (SSE2)



- **PSADBW: add bytewise SAD (with no accumulation)**

```
MOVDQU L[i],L[i+1],L[i+2],L[i+3] → xmm1 (16B)
MOVDQU R[i],R[i+1],R[i+2],R[i+3] → xmm2 (16B)
PSADBW xmm1, xmm2                → xmm2
PADDD  xmm2, xmm3                → xmm3 (16B)
```

Vector processors

Comparable throughput of memory for full-use of ALU

Working set larger than capacity of cache

How to speed-up programs with no locality

- Cache < Working set < Main memory

  Employ software/hardware prefetching to cache
  Slow but wide memory bus will support cache

- Main memory < Working set

  Employ overlapped data transmission through network
  Very slow then should reduce the amount of transmission

Prefetching with non-blocking cache

1. main memory ⇒ cache … invoke prefetching for next iteration by special insn
2. cache ⇒ register … load instruction
3. register ⇒ register … execution
4. register ⇒ cache … store instruction
5. cache ⇒ main memory … write back by cache-line replacement

- Normal computers have no explicit “keep in cache” instruction.
- Each cache-line is automatically invalidated by LRU algorithm.

Some computers have separated local memory space.

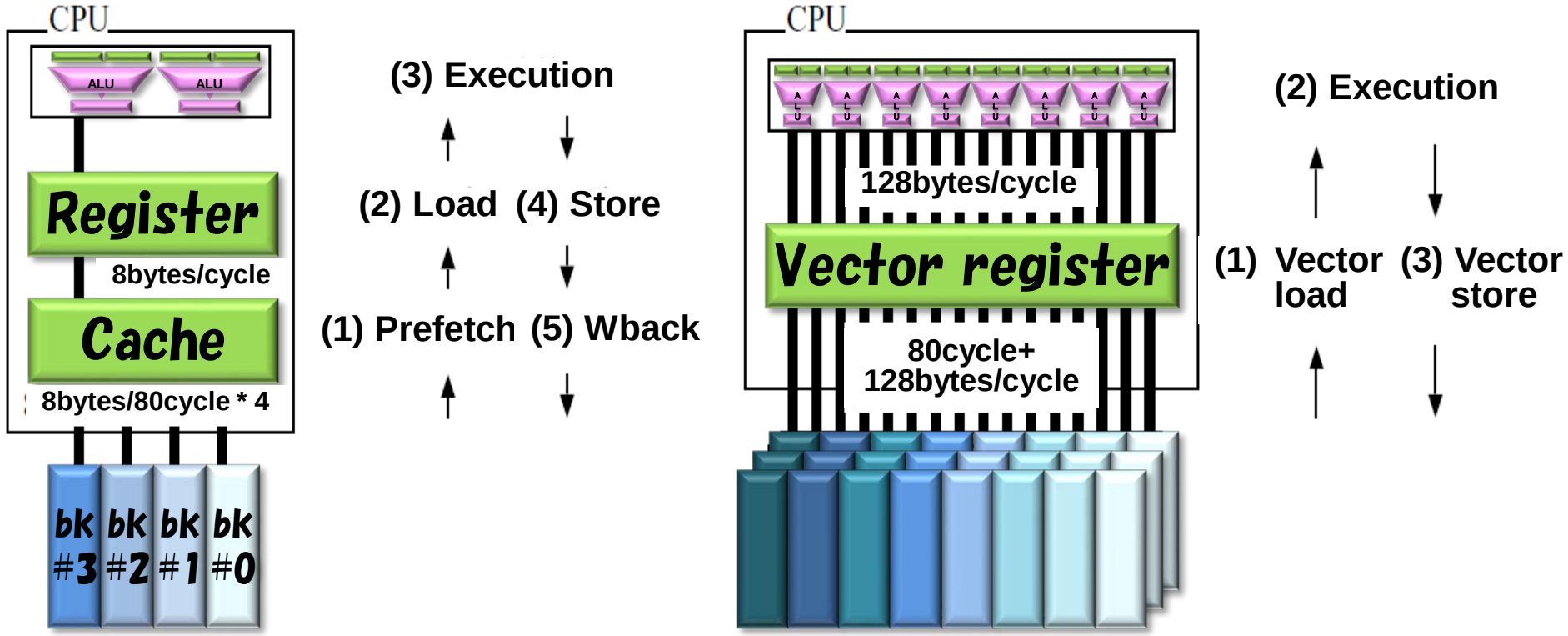
- Software can explicitly invoke DMA from/to main memory.
  But programmer should make elaborate scheduling of local memory.

Vector computers have large register space.

1. main memory ⇒ vector register … vector load instruction
2. register ⇒ register … vector operation
3. vector register ⇒ main memory … vector store instruction

- Large register space provides sufficient data to multiple ALUs.

Multi-banked memory for parallel execution



(a) Popular structure with cache

(b) Vector pipeline

How to overcome problems in superscalars

Obstacles for speed-up in scalar processors

1. Dependency of execution

⇒ Alleviate by large instruction window

2. Branch penalty

⇒ Employ accurate branch prediction

⇒ Replace branch with conditional-exec insn

In vector processors:

1. Program is limited to regular parallel execution

⇒ No irregular dependency

2. No branch instruction

⇒ Only conditional-exec with mask-register

Facilities in vector processors

Explicit registers:

- **General purpose registers … shared by scalar-unit**
- **Vector registers … 64bits x 64 x 256sets**
- **Mask registers … 1bit x 64 x 256sets**

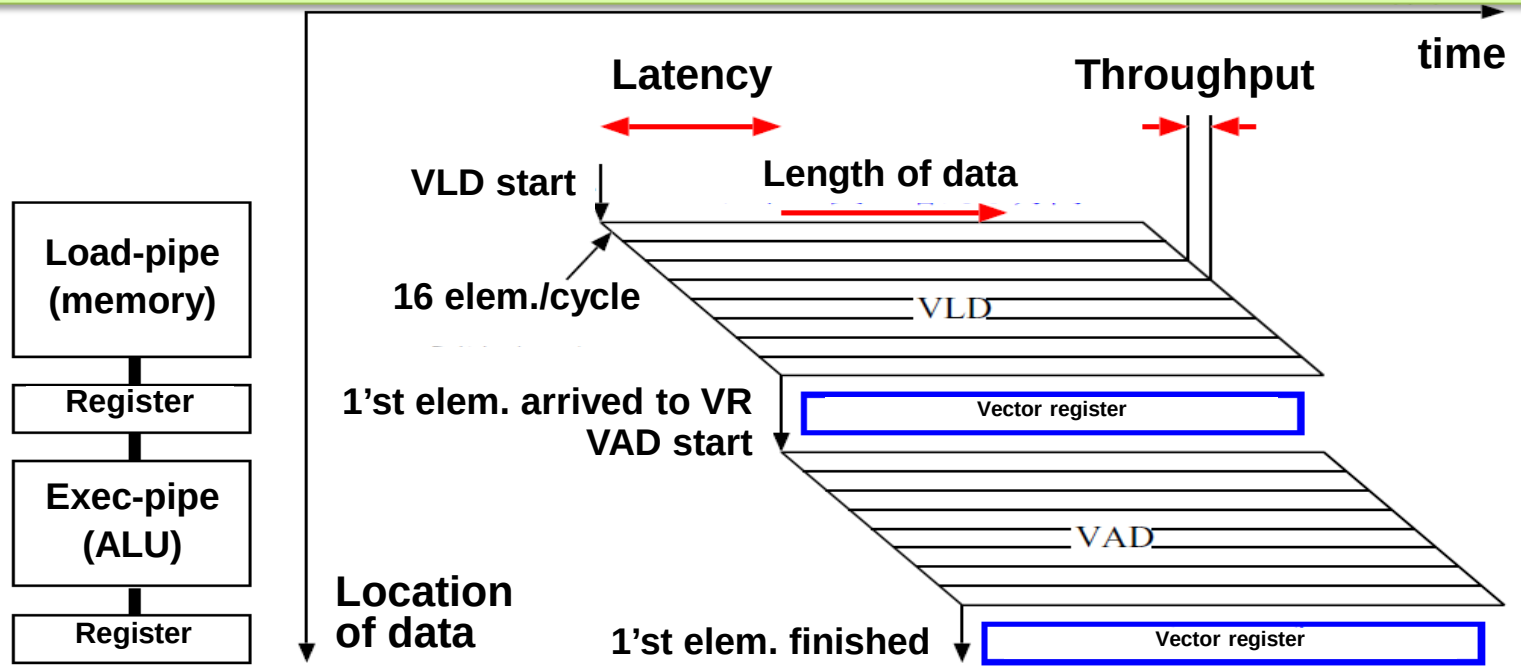
Implicit register

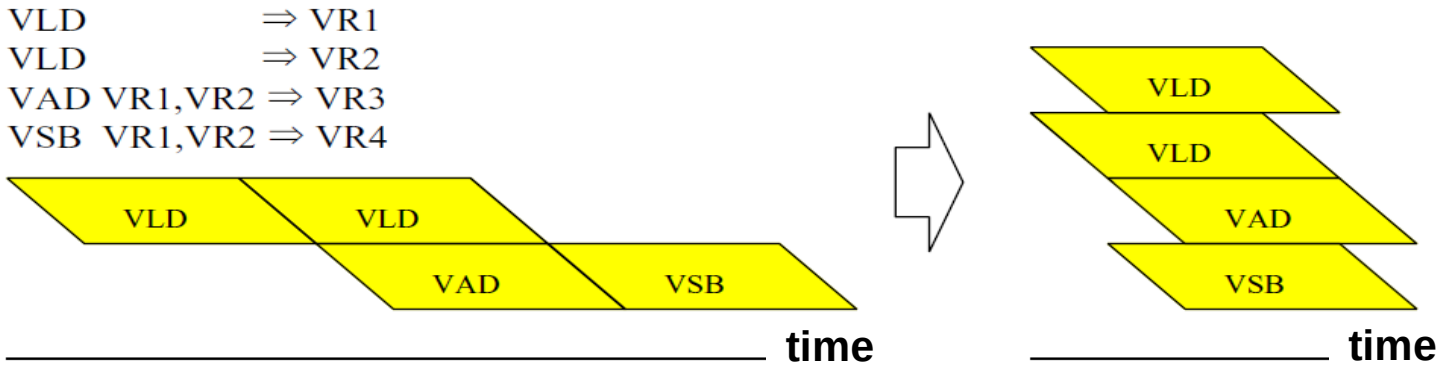
- **Address translation (virtual address ⇒ real address)**

- **Vector length register … number of elements (1～2048)**

2048: 32 vector registers are concatenated to form 64bits x 2048 x 8sets

Chaining of vector operations





(a) Load-pipe*1 and add/sub-pipe*1 (b) Load-pipe*2 and add/sub-pipe*2

Mask registers

Simple loop:

```
for (i=0; i<2048; i++)
  C[i] = A[i] + B[i];
```

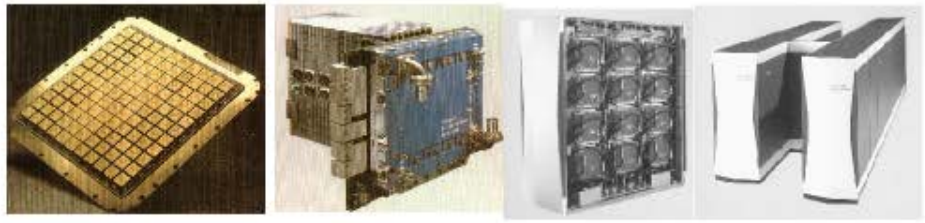
```
VLD ( load )      A    ⇒ VR1 ⋯ load 2048 elements
VLD ( load )      B    ⇒ VR2 ⋯ load 2048 elements
VAD ( add ) VR1,VR2    ⇒ VR3 ⋯ add 2048 elements
VST ( store )   VR3    ⇒ C   ⋯ store 2048 elements
```

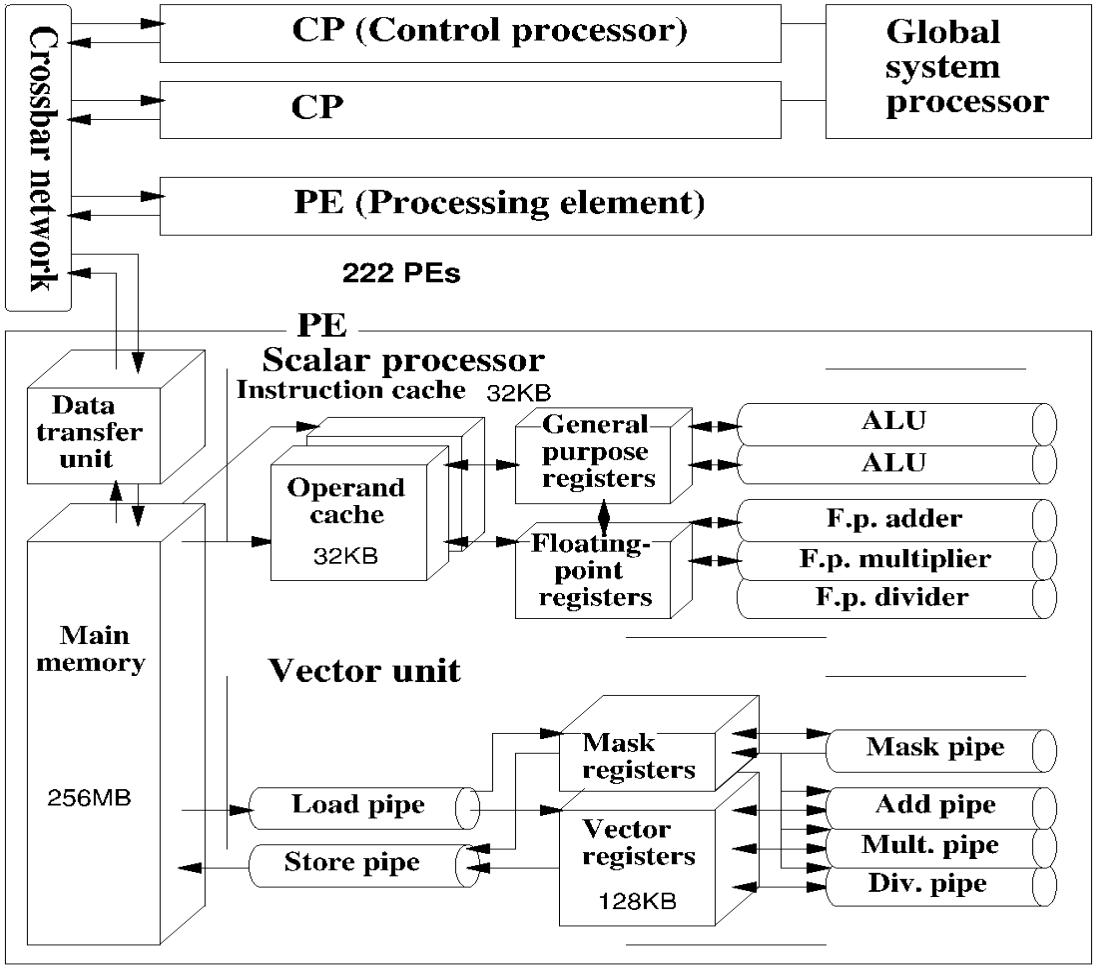
With conditional execution:

```
for (i=0; i<2048; i++) {
  if (A[i] > B[i])
    C[i] = A[i] - B[i];
  else
    C[i] = B[i] - A[i];
}
```

```
VLD ( load )      A          ⇒ VR1 ⋯ load 2048 elements
VLD ( load )      B          ⇒ VR2 ⋯ load 2048 elements
VLE ( cmp )  VR1,VR2         ⇒ MR1 ⋯ compare & set mask-registers
VSB ( sub ) VR1,VR2,MR1      ⇒ VR3 ⋯ conditional subtract with mask
VSB ( sub ) VR2,VR1,~MR1     ⇒ VR3 ⋯ conditional subtract with !mask
VST ( store )    VR3         ⇒ C   ⋯ store 2048 elements
```

A vector super-computer VPP500-5000 by FUJITSU





Vector instructions (load/store)

VLM/VSM R1,R2,MR

- vector load/store to/from mask register
  R2: distance

VLD/VST R1,R2,VR,MR

- vector load/store to/from vector register
  R2: distance

VLC/VSC R1,R2,VR,MR

- vector compressed load/store
  R2: distance

VIL/VIS R,VR1,VR2,MR

- vector indirect load/store
  VR2[i]=mem[R+VR1[i]]

Vector instructions (execution)

VAD/VSB/VML/VDV VR1/R,VR2,VR3,MR

- vector add/subtract/multiply/divide with mask

VSL/VSR/VSA VR1/R,VR2,VR3,MR

- vector shift left/right/arithmetic with mask

VMX/VMN VR1/R,VR2,VR3,MR

- vector select max/min value with mask

VAN/VOR/VXO VR1/R,VR2,VR3,MR

- vector and/or/xor with mask

VAN/VOR/VXO MR1,MR2,MR3

- vector and/or/xor mask with mask

Vector instructions (compare)

VLE/VLT/VGE/VGT/VEQ/VNE VR1/R,VR2,MR1,MR2

- vector compare and set mask register
- mask is set to 1 if true
- mask is set to 0 if false
-

Vector instructions (misc.)

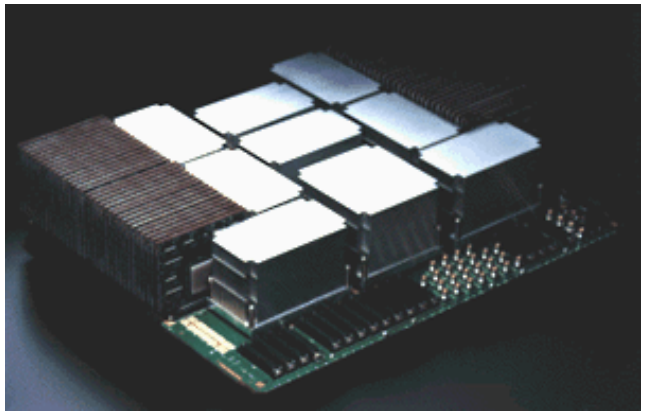
- vector convert integer to/from float/double
- vector search MAX/MIN in vector register
- vector sum of vector register
- vector inner product
- vector shift elements in vector register
- generate pattern in vector register
  - sequential numbers
  - subarray masks

GPU

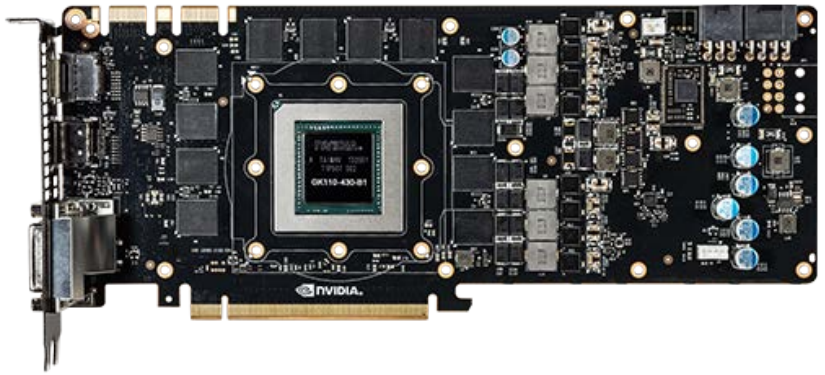
Difference between vector and GPU

- Number of ALUs are the same (32ALUs/thread)
- Vector: hide long latency for main memory by in-order pipelining
- GPU: hide long latency for main memory by out-of-order multi-threading

http://pr.fujitsu.com/jp/news/1999/Apr/20-1tenpu.html

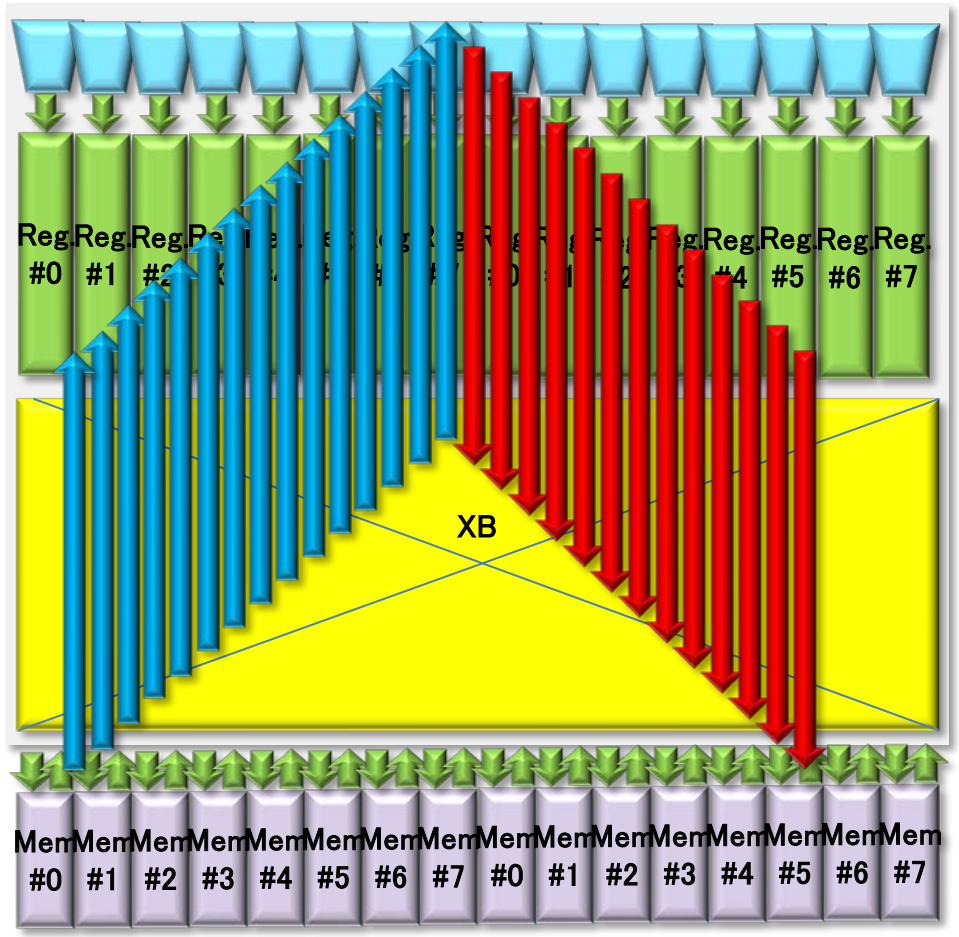


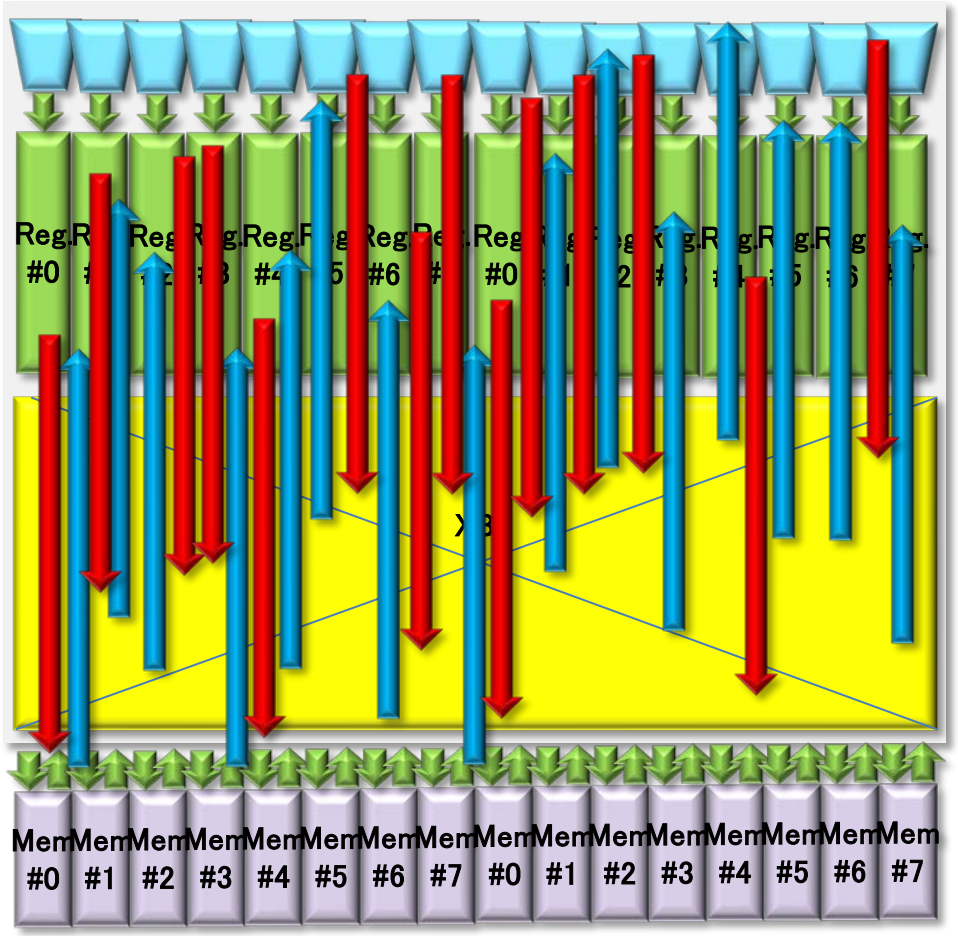
http://www.nvidia.co.jp/object/geforce-gtx-titan-black-jp.html



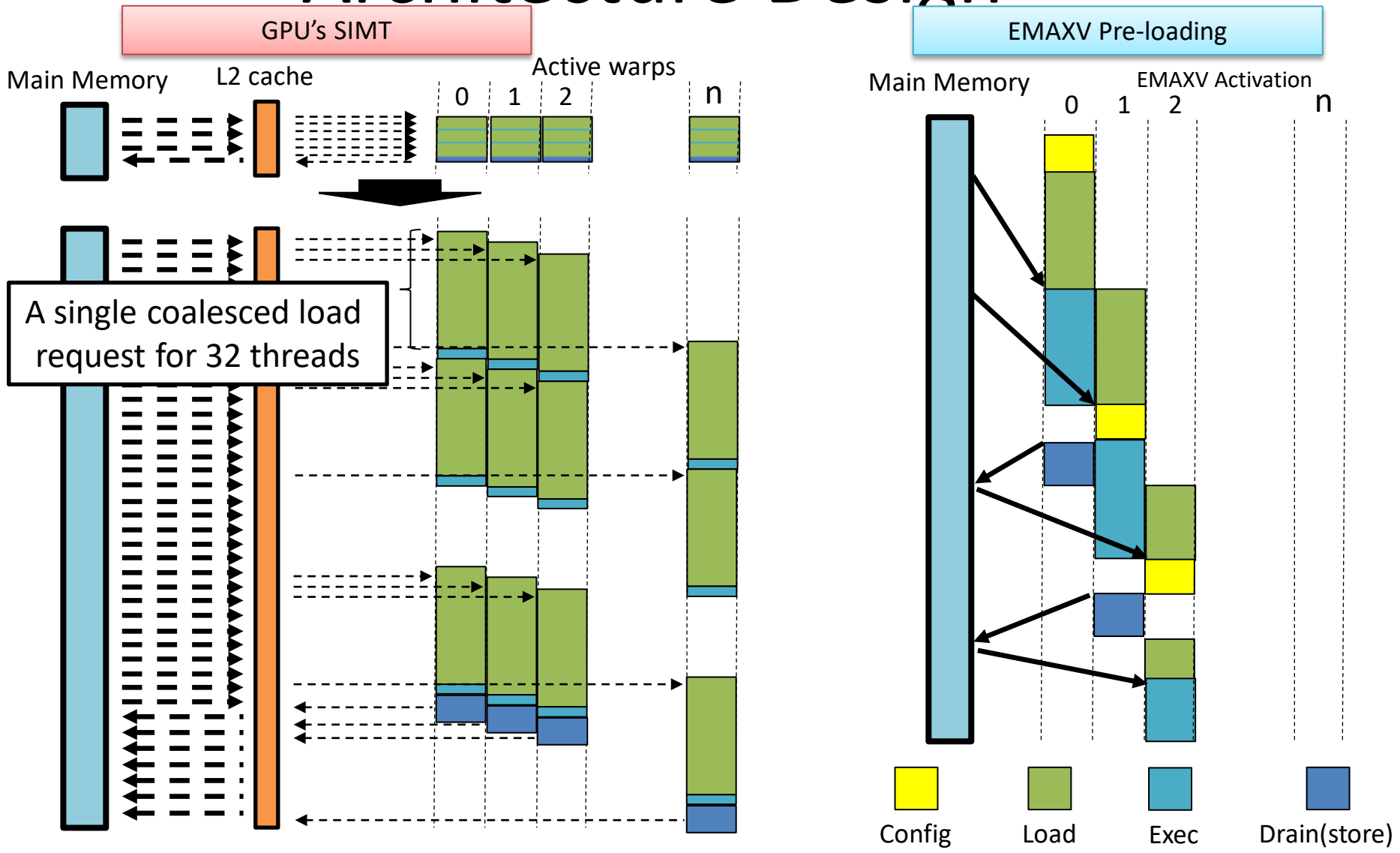
Difference between vector and GPU

- In-order chaining of long load⇒exec⇒store
- Multi-threading of huge number of short requests

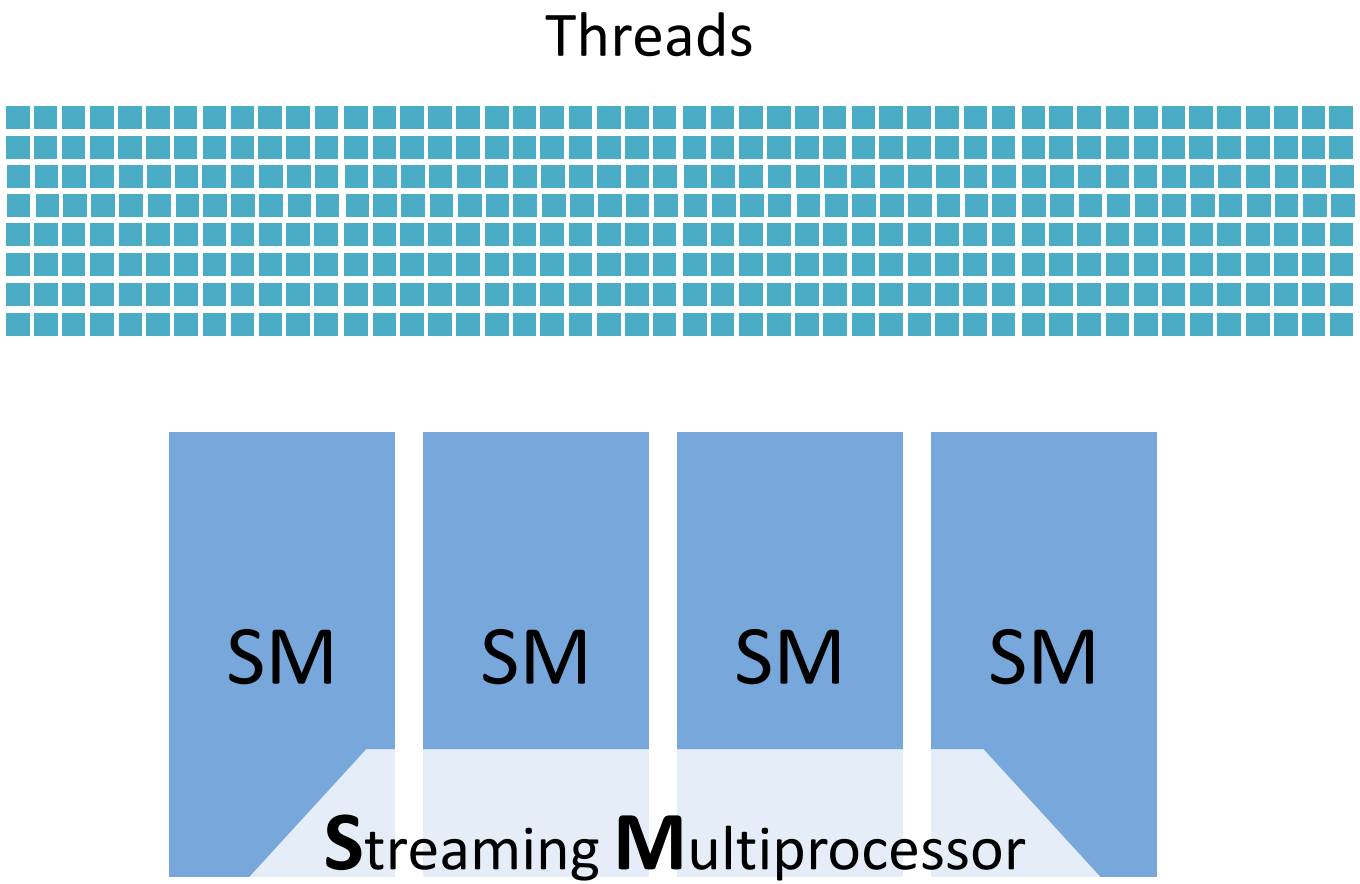




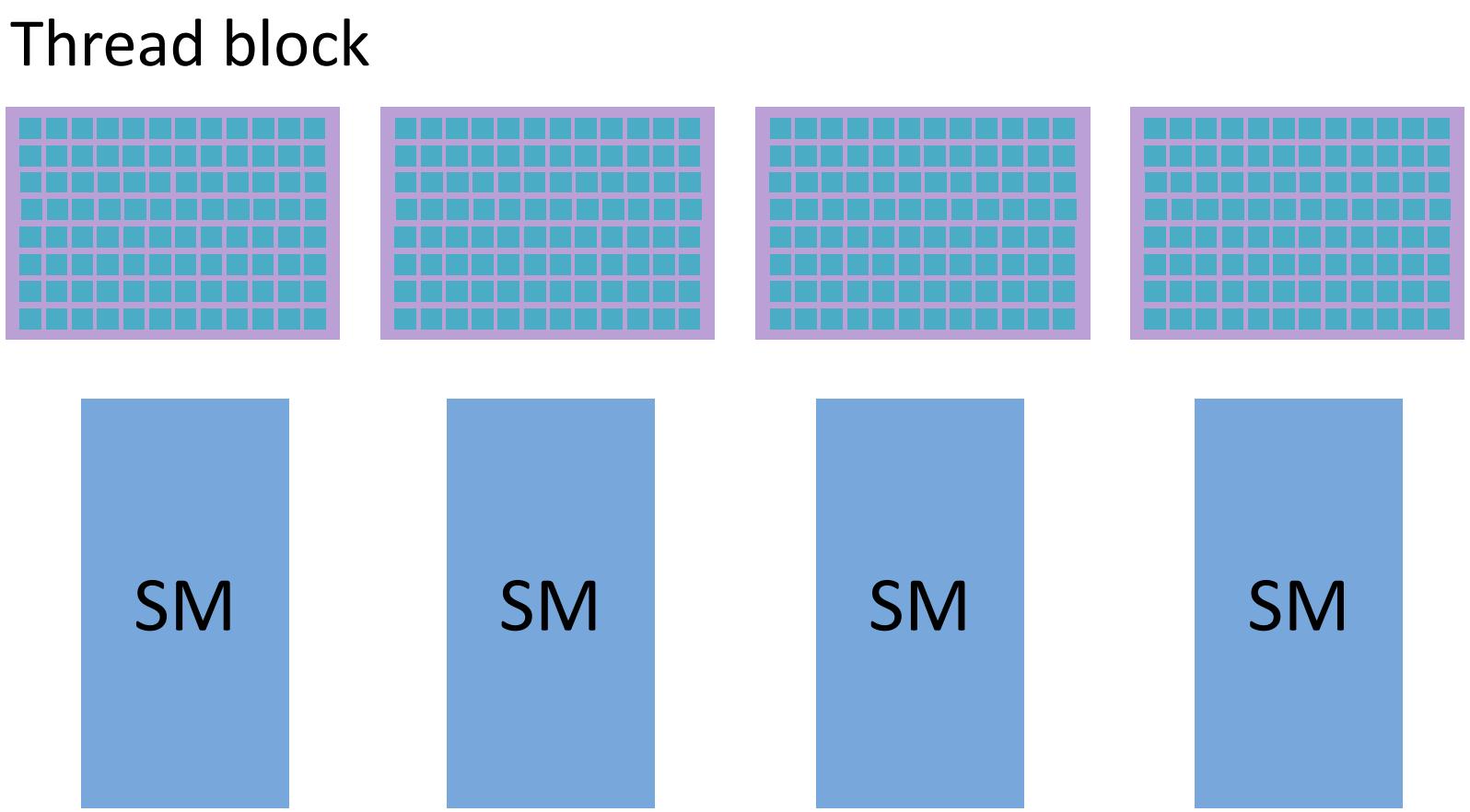
Architecture Design



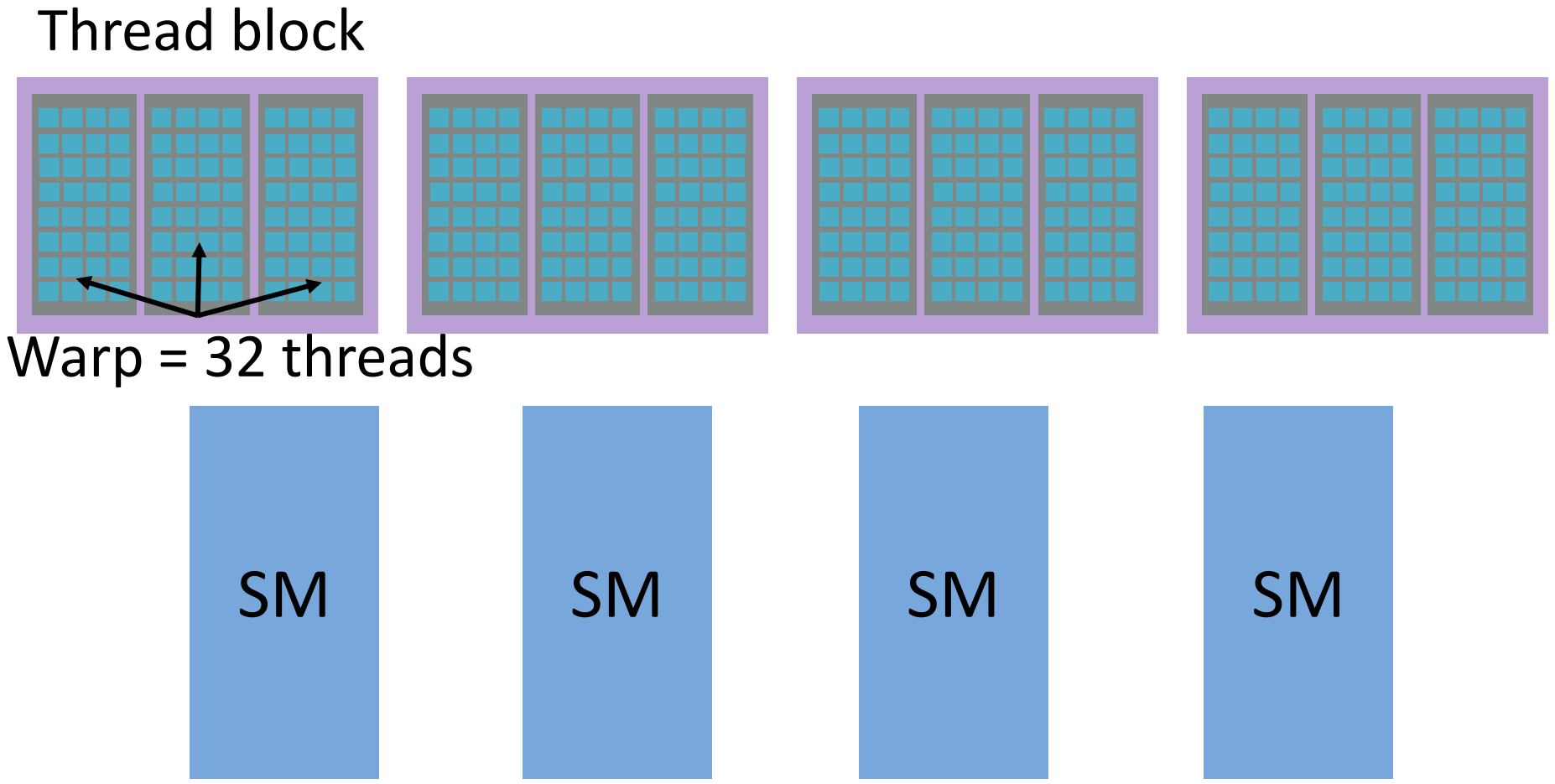
SIMT Architecture: threading



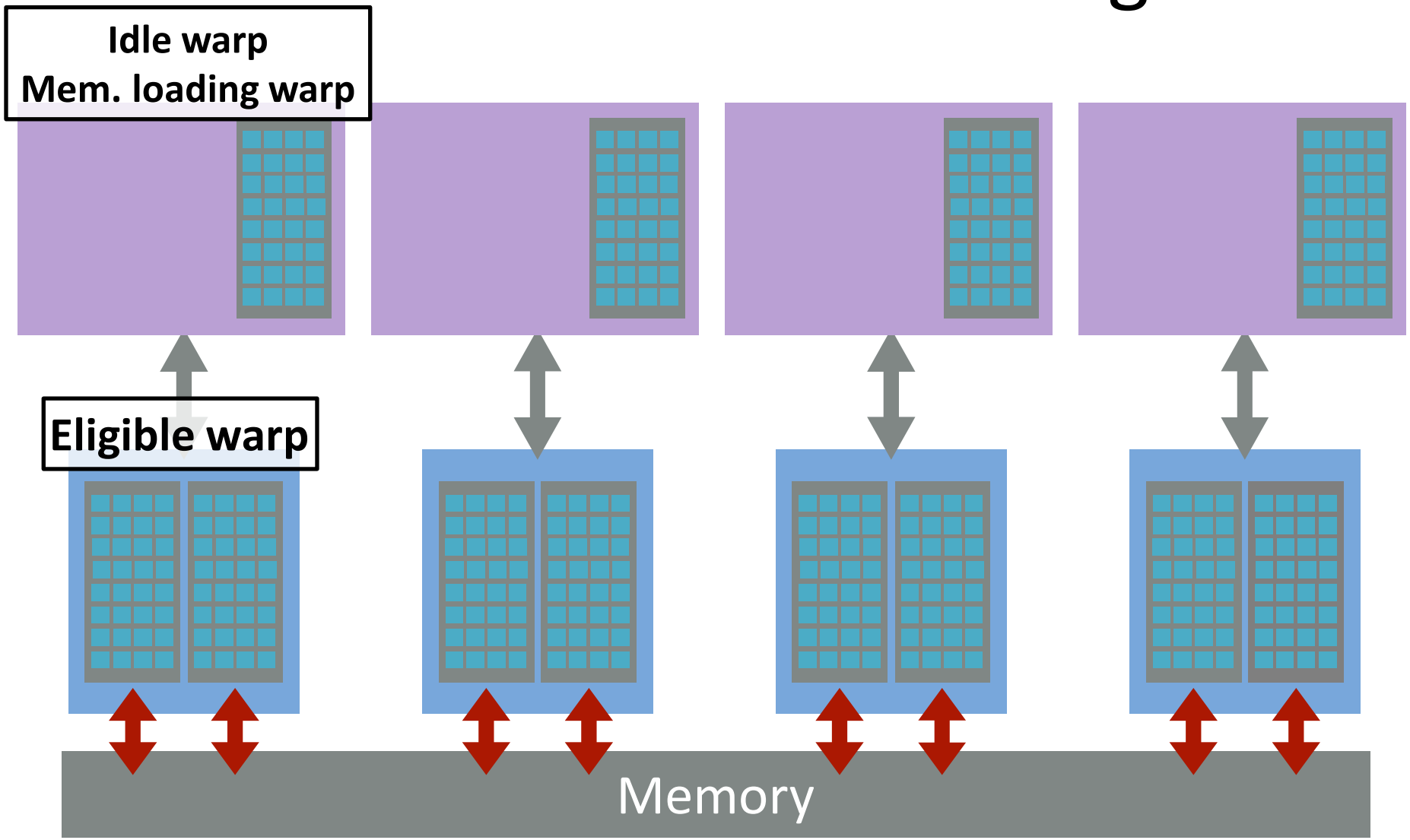
SIMT Architecture: threading



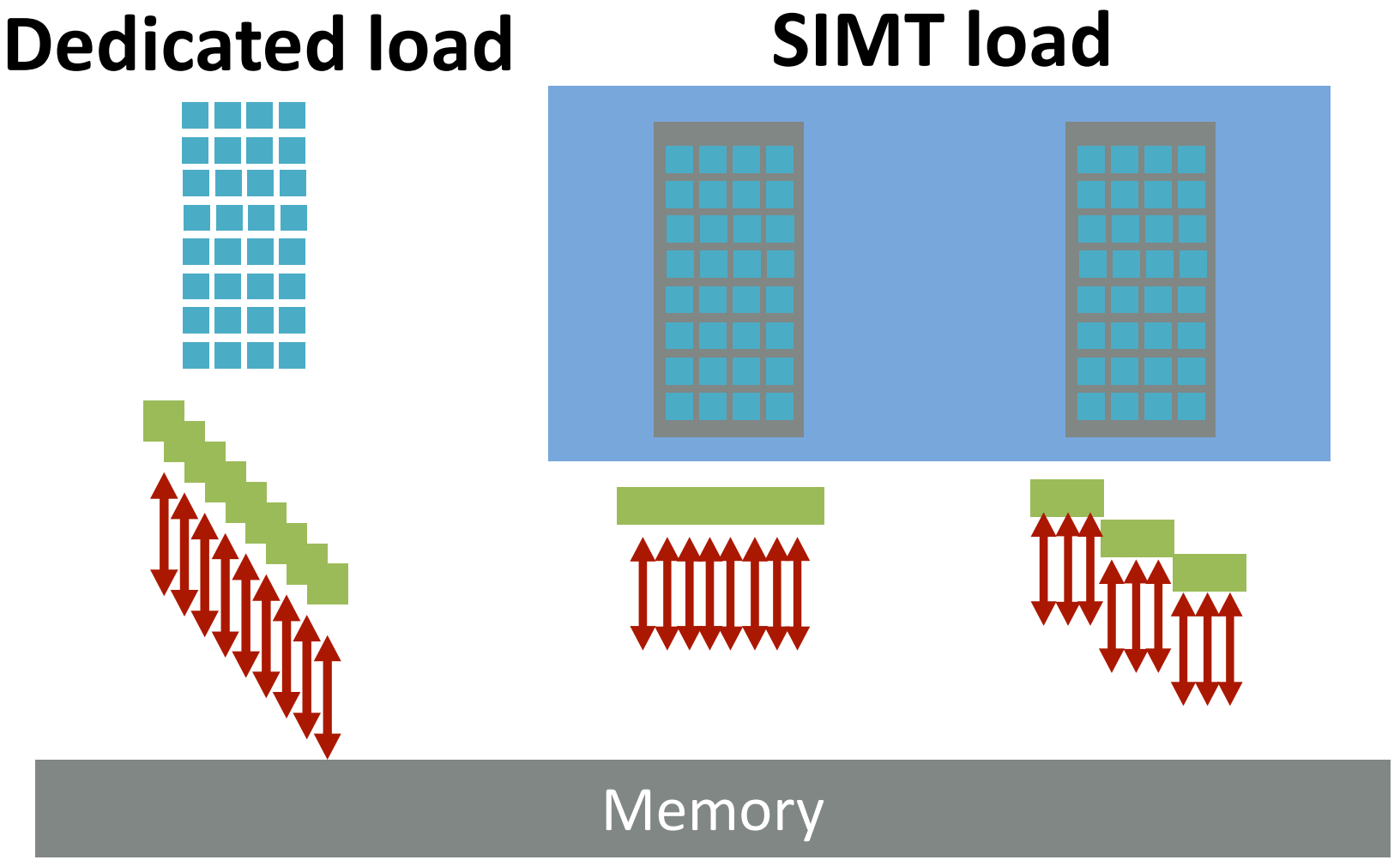
SIMT Architecture: threading



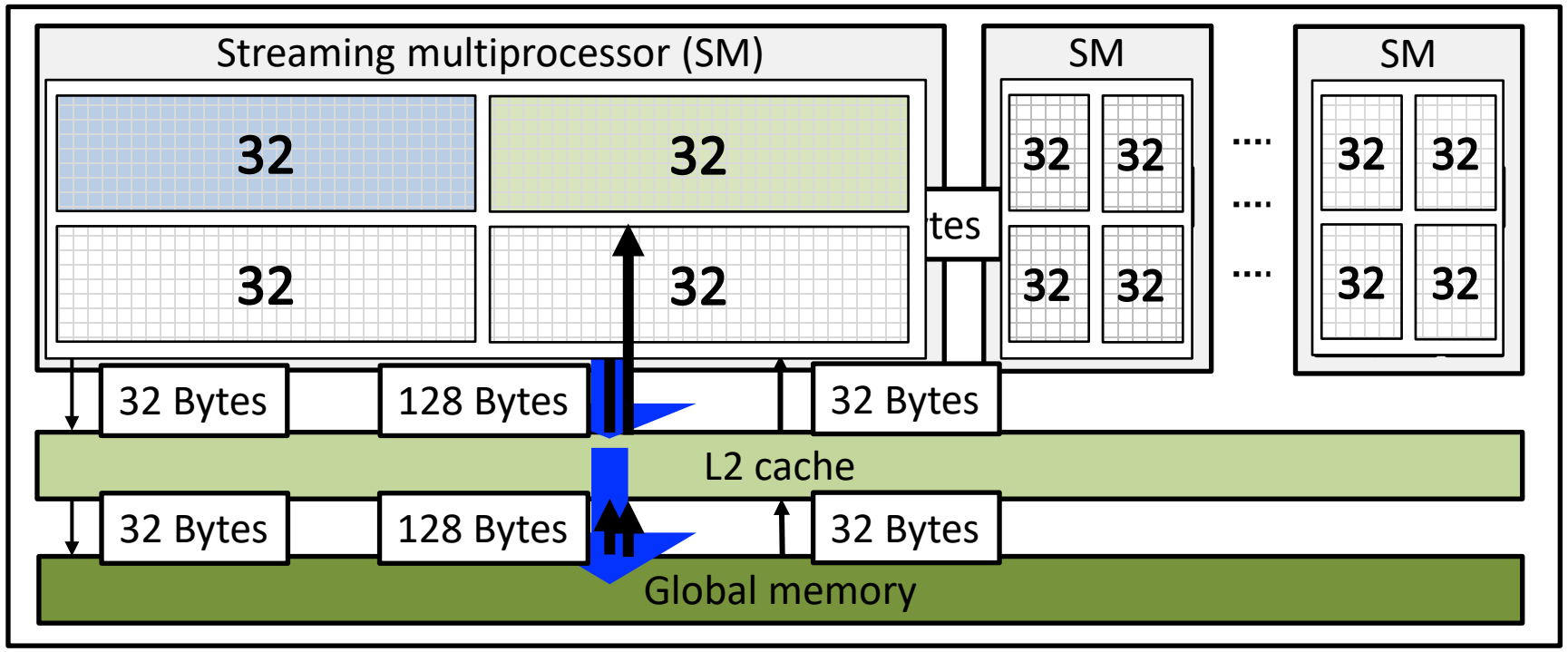
SIMT Architecture: threading

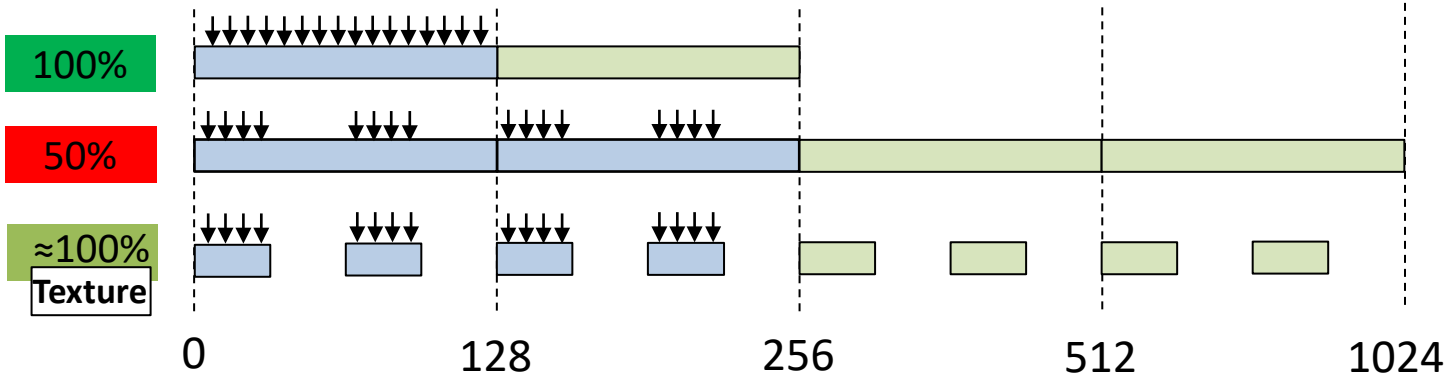


SIMT Architecture: threading



SIMT Architecture: memory system





Evaluation: GPU devices

Devices	GTX670	GTX650	TegraK1	GTX780	GTX980
Stream multiprocessor (SM)	7	4	1	12	16
Core counts	1344	768	192	2304	2048
Core frequency (MHz)	1058	1098	852	902	1253
Compute compatibility	3		3.2	3.5	5.2
32-bit registers per thread	63		255		
Shared memory per SM (KB)	64 (shared with L1)				64
Warp schedulers : cores ratios	4:192				4:128
Active block per SM	16				32
L2 cache size (KB)	512	384	128	1536	2048
Memory bus (bit)	256	192	64	384	256
Memory frequency (MHz)	3004	3004	924	3004	3505

Kepler

Maxwell

That’s all for today