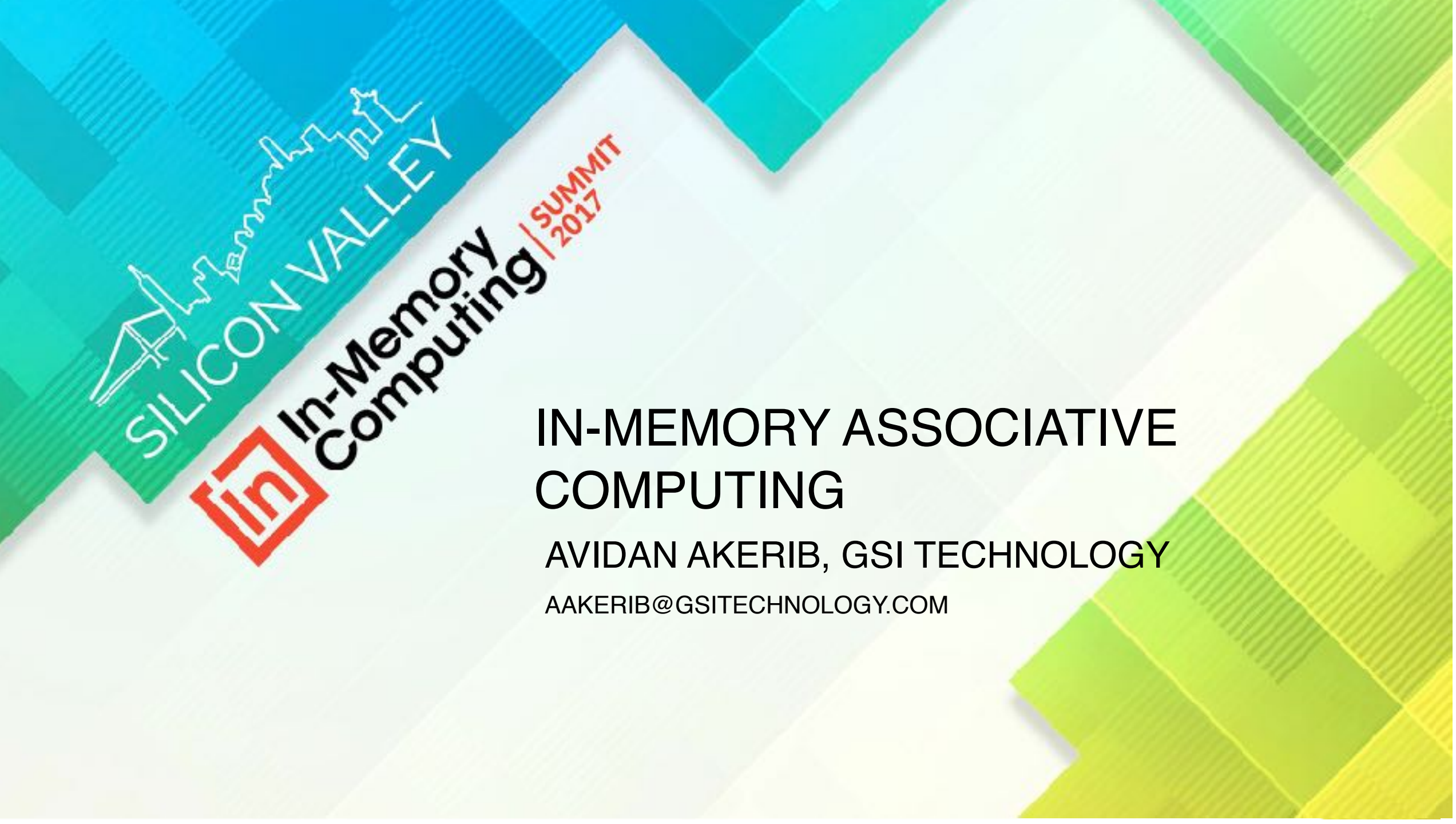


IN-MEMORY ASSOCIATIVE COMPUTING

AVIDAN AKERIB, GSI TECHNOLOGY

AAKERIB@GSITECHNOLOGY.COM

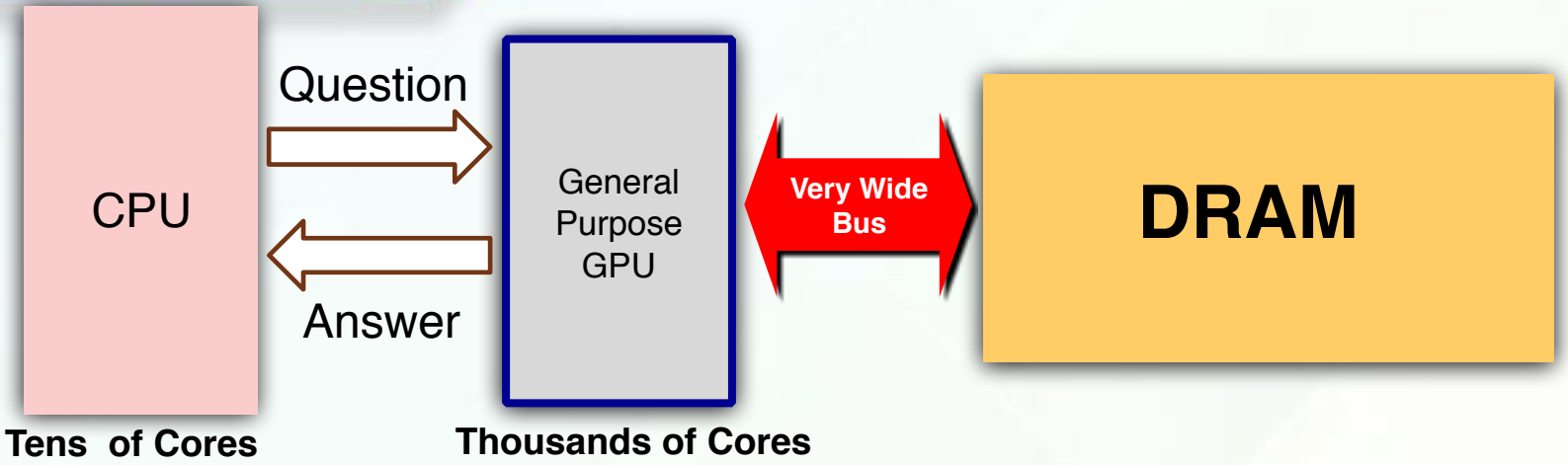
AGENDA

- The AI computational challenge
- Introduction to associative computing
- Examples
- An NLP use case
- What's next?

THE CHALLENGE IN AI COMPUTING

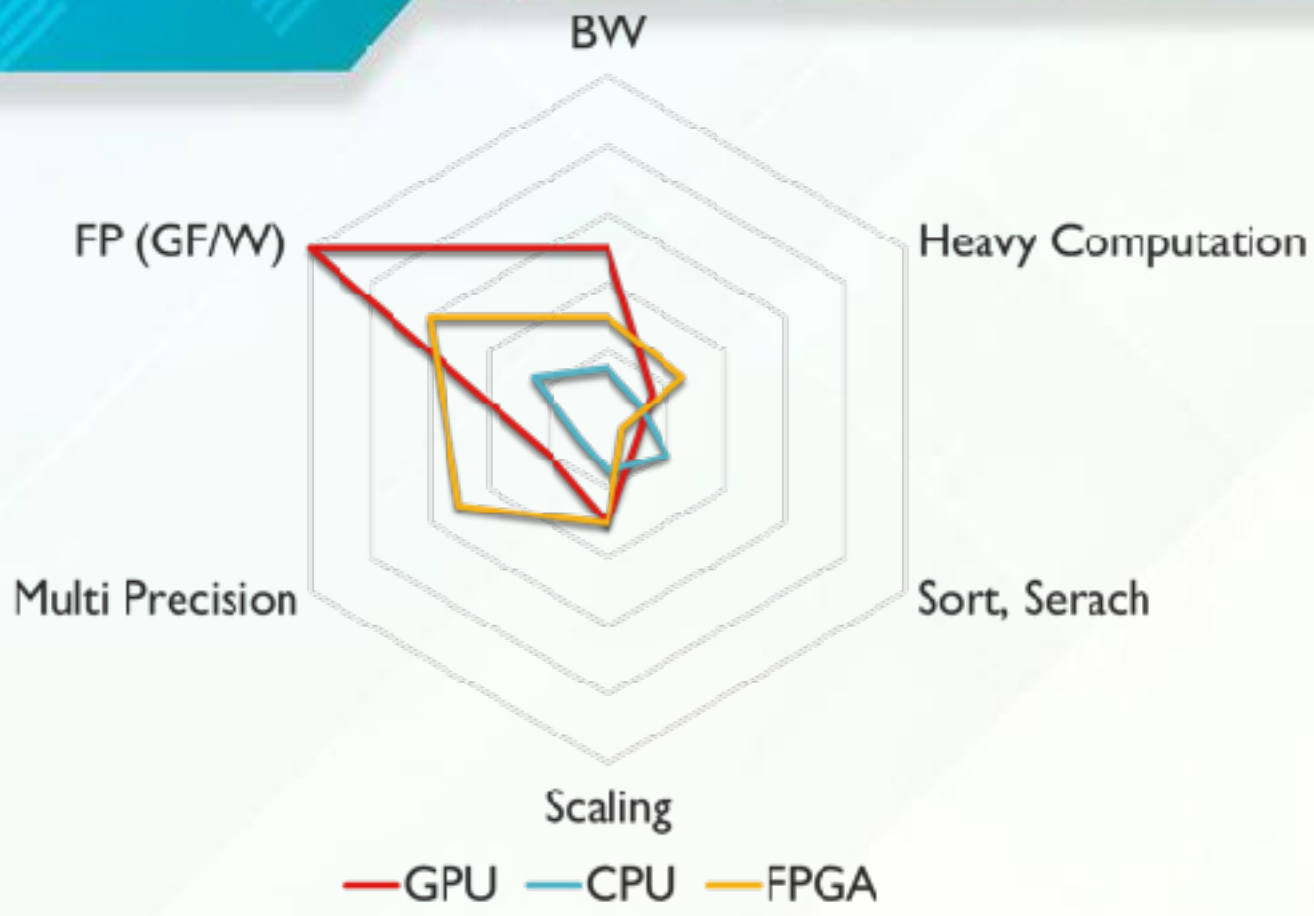
AI Requirement	Use Case Example
32 bit FP	Neural network learning
Multi precision	Neural network inference, data mining, etc.
Scaling	Data center
Sort-search	Top-K, recommendation, speech, classify image/video
Heavy computation	Non linearity, Softmax, exponent , normalize
Bandwidth	Required for speed and power

CURRENT SOLUTION

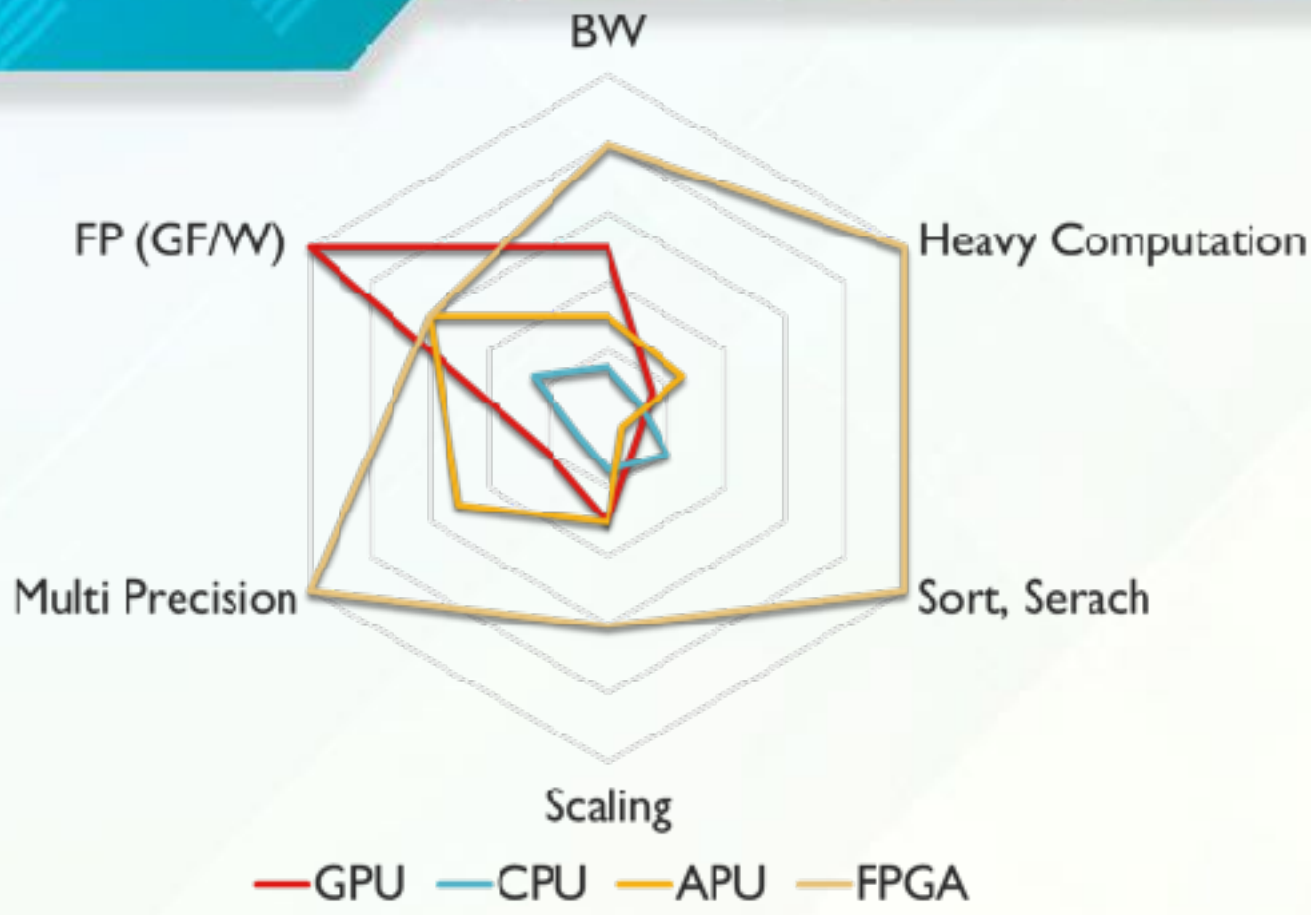


- Bottleneck when register file data needs to be replaced on a regular basis
  - ➢ Limits performance
  - ➢ Increases power consumption
- Does not scale with the search, sort, and rank requirements of applications like recommender systems, NLP, speech recognition, and data mining that requires functions like Top-K and Softmax.

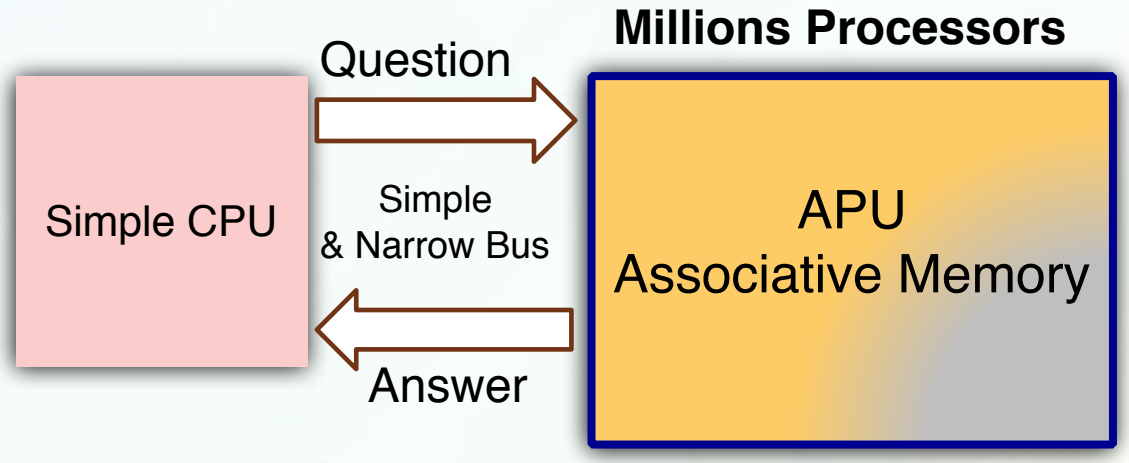
GPU VS CPU VS FPGA



GPU VS CPU VS FPGA VS APU



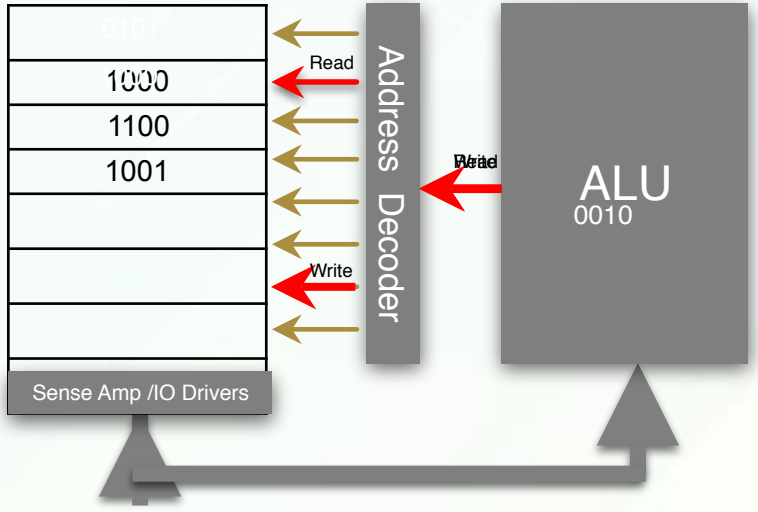
GSI'S SOLUTION
APU—ASSOCIATIVE PROCESSING UNIT



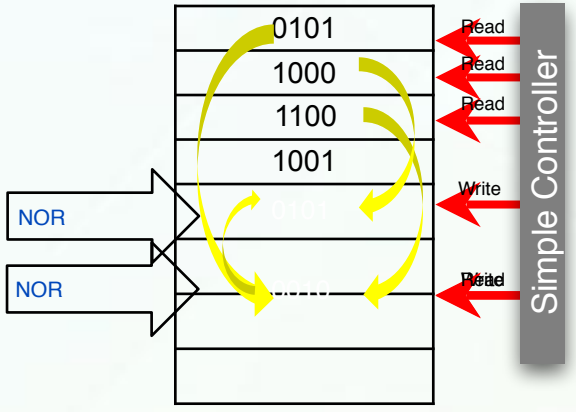
- Computes in-place directly in the memory array—removes the I/O bottleneck
- Significantly increases performances
- Reduces power

IN-MEMORY COMPUTING CONCEPT

THE COMPUTING MODEL FOR THE PAST 80 YEARS

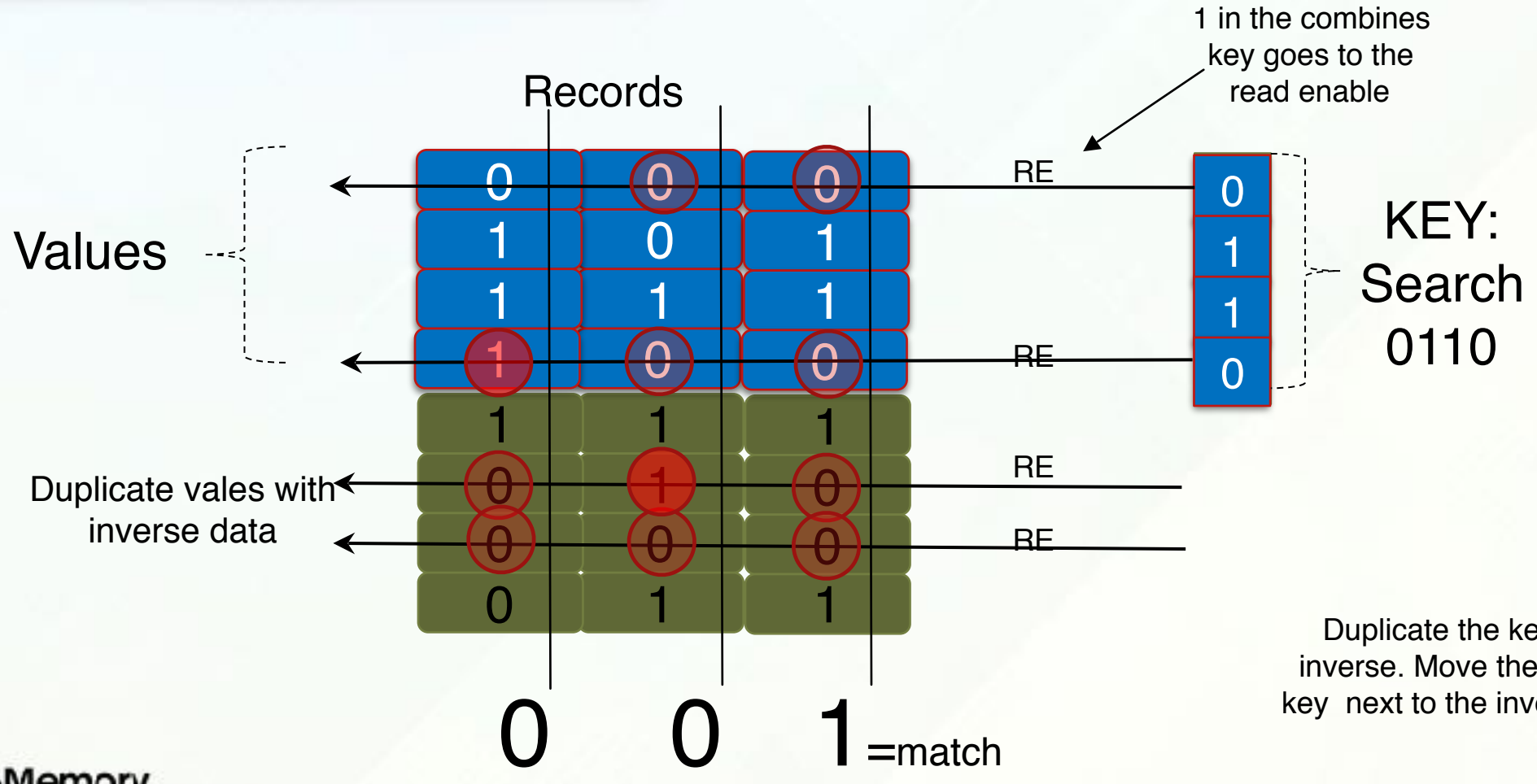


THE CHANGE—IN-MEMORY COMPUTING



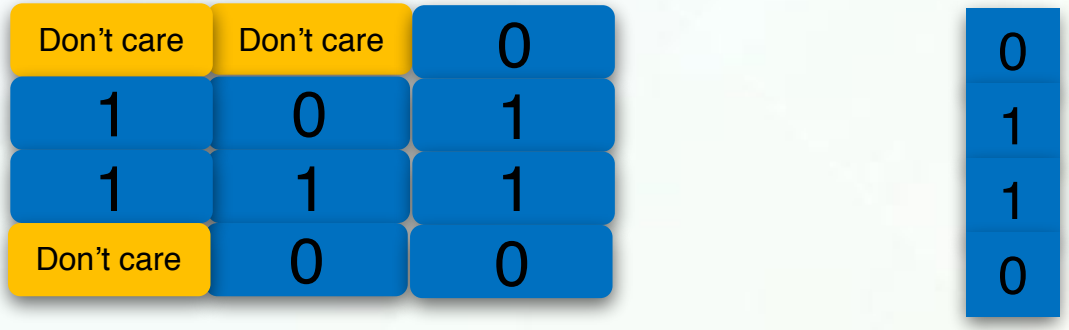
- Patented in-memory logic using only Read/Write operations
- Any logic/arithmetic function can be generated internally

CAM/ ASSOCIATIVE SEARCH

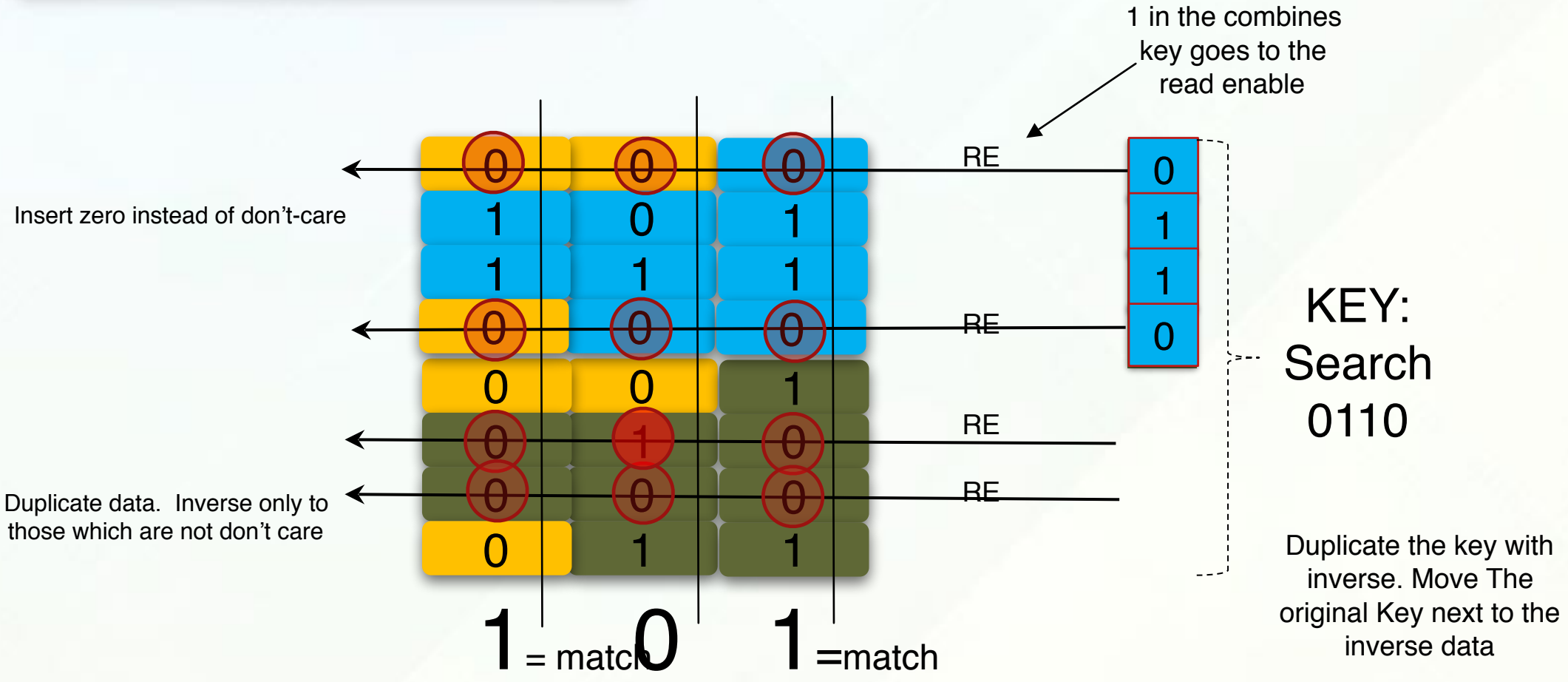


Duplicate the key with inverse. Move the original key next to the inverse data

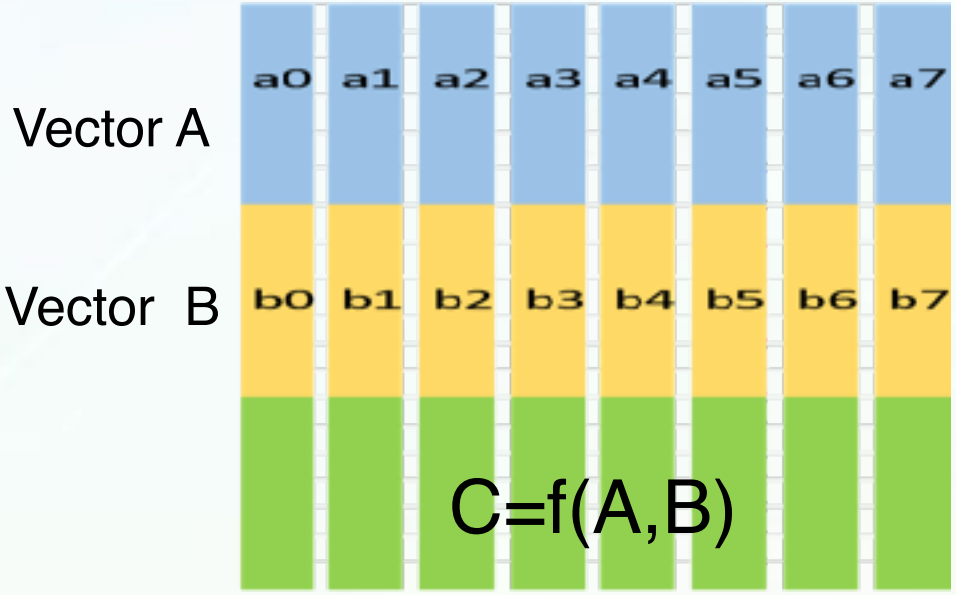
TCAM SEARCH WITH STANDARD MEMORY CELLS



TCAM SEARCH WITH STANDARD MEMORY CELLS



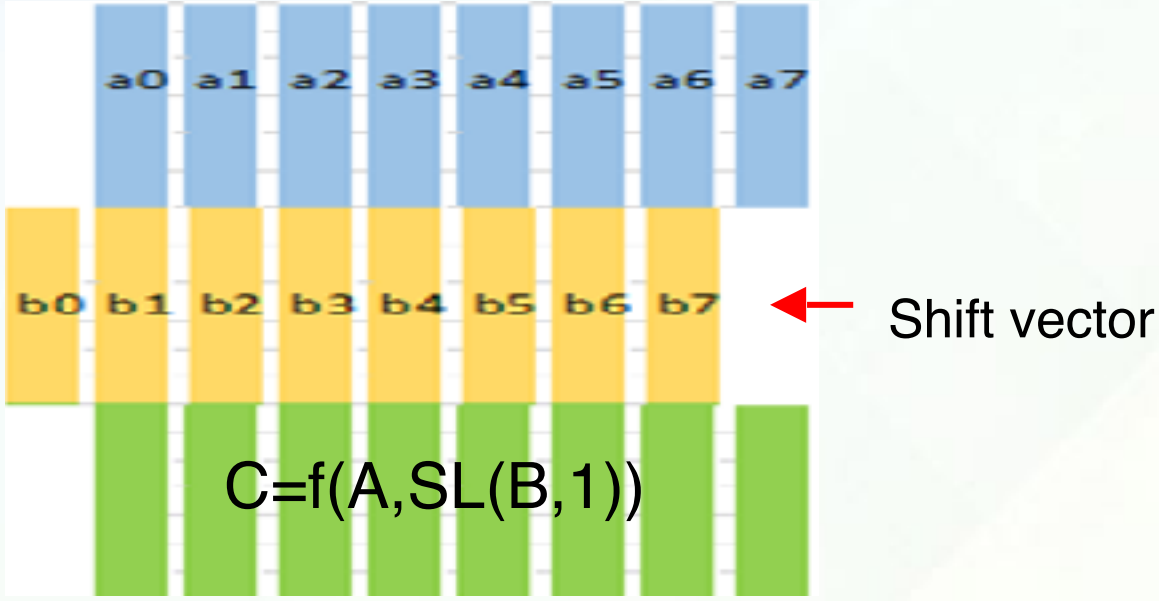
COMPUTING IN THE BIT LINES



Each bit line becomes a processor and storage

millions of bit lines = millions of processors

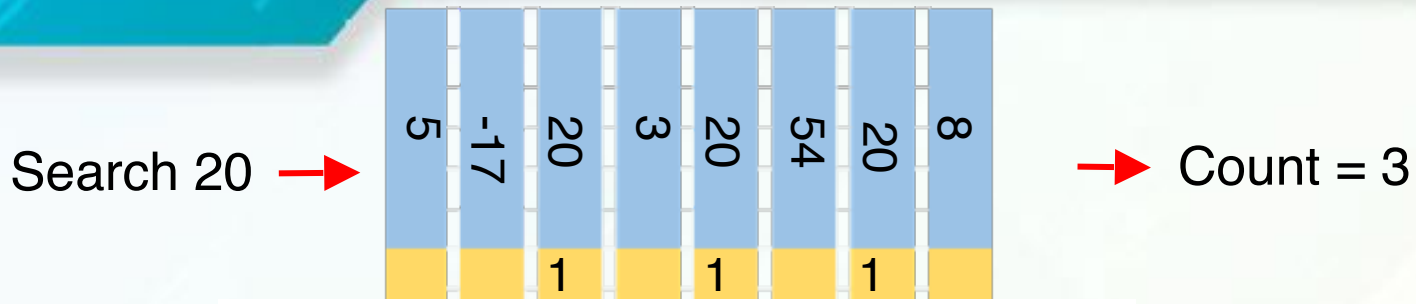
NEIGHBORHOOD COMPUTING



Parallel shift of bit lines @ 1 cycle sections

Enables neighborhood operations such as convolutions

SEARCH & COUNT

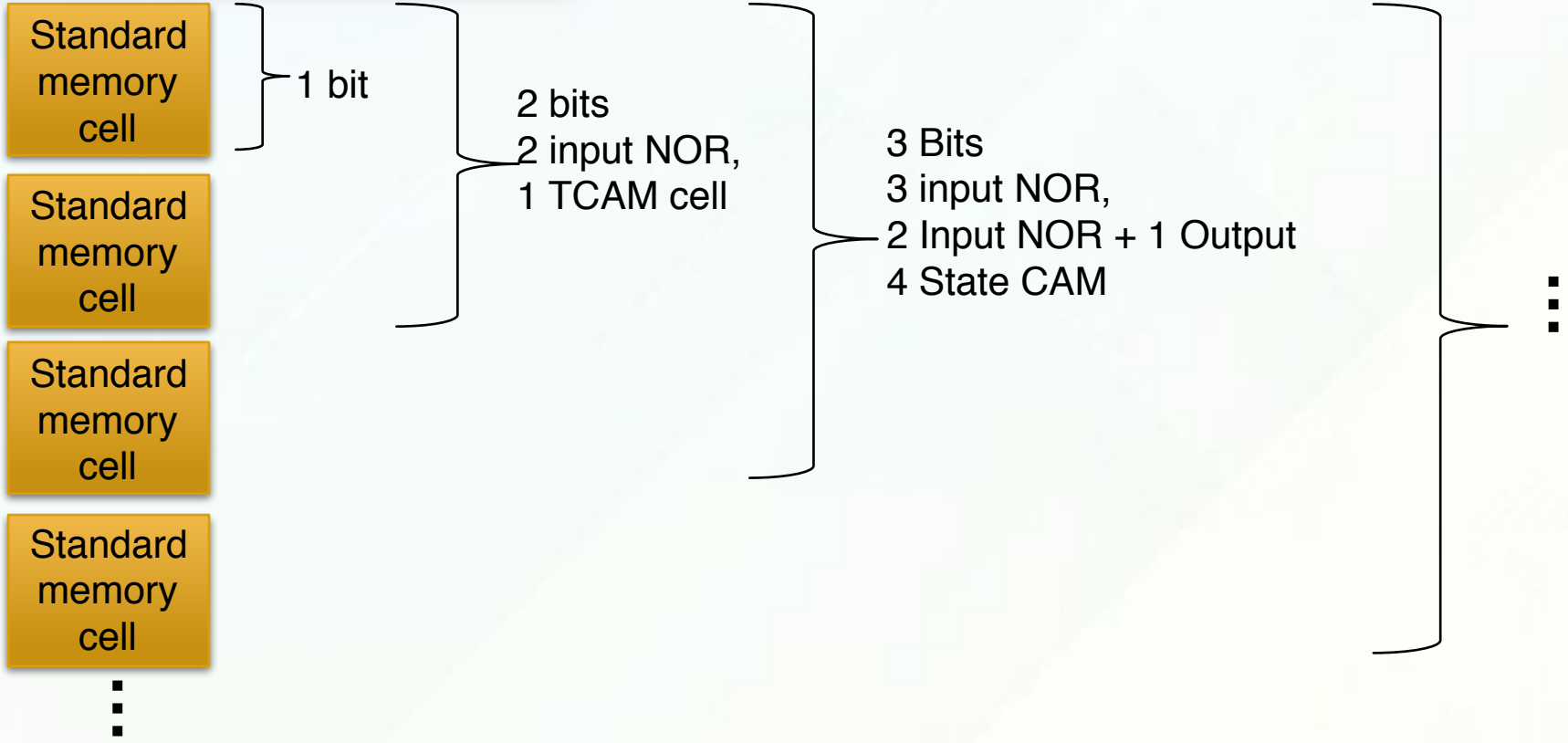


- Search (binary or ternary) all bit lines in 1 cycle
  - 128 M bit lines => 128 Peta search/sec
- Key applications for search and count for predictive analytics:
  - Recommender systems
  - K-nearest neighbors (using cosine similarity search)
  - Random forest
  - Image histogram
  - Regular expression

DATABASE SEARCH AND UPDATE

- Content-based search, record can be placed anywhere
  - Update, modify, insert, delete is immediate
- Exact Match
  - CAM/TCAM
- Similarity Match
- In-Place Aggregate

TRADITIONAL STORAGE CAN DO MUCH MORE

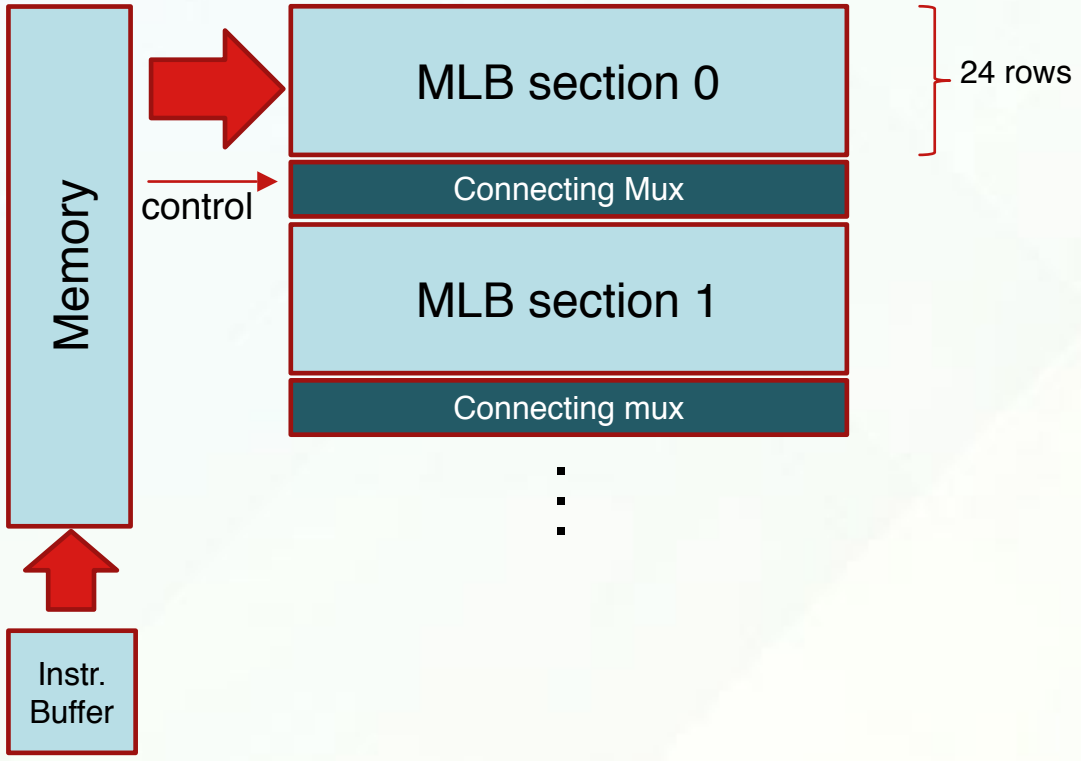


CPU/GPGPU VS APU

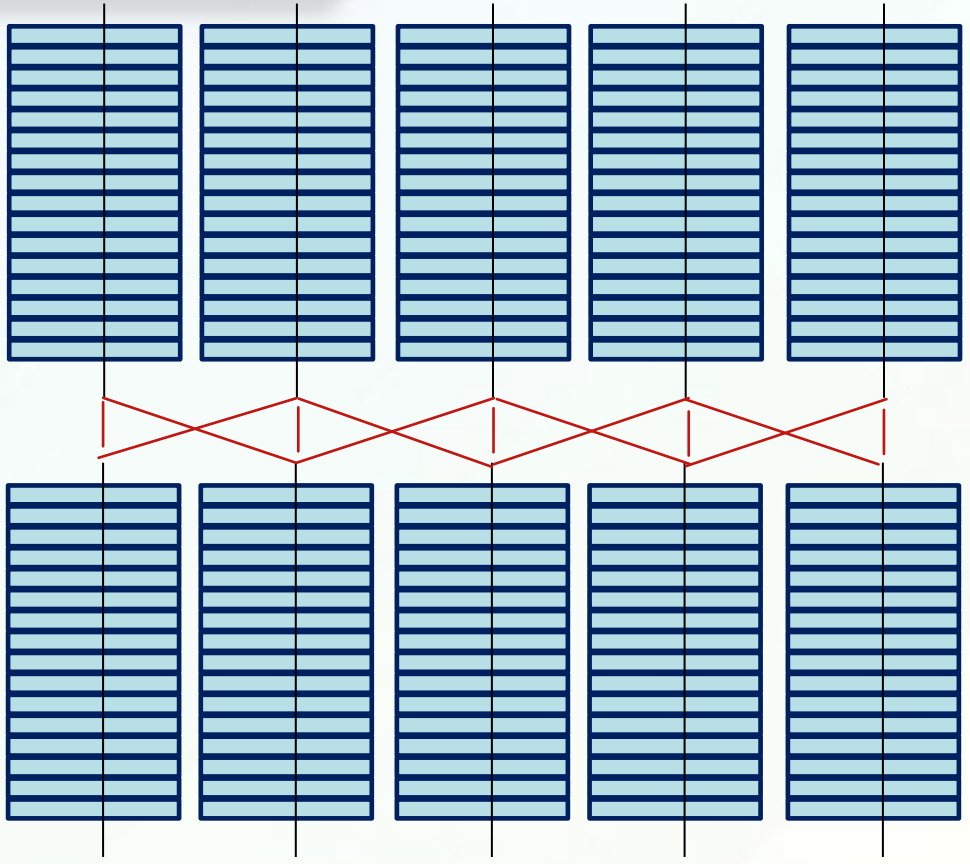
CPU/GPGPU (Current Solution)	In-Place Computing (APU)
Send an address to memory	Search by content
Fetch the data from memory and send it to the processor	Mark in place
Compute serially per core (thousands of cores at most)	Compute in place on millions of processors (the memory itself becomes millions of processors
Write the data back to memory, further wasting IO resources	No need to write data back—the result is already in the memory
Send data to each location that needs it	If needed, distribute or broadcast at once

ARCHITECTURE

SECTION COMPUTING TO IMPROVE PERFORMANCE



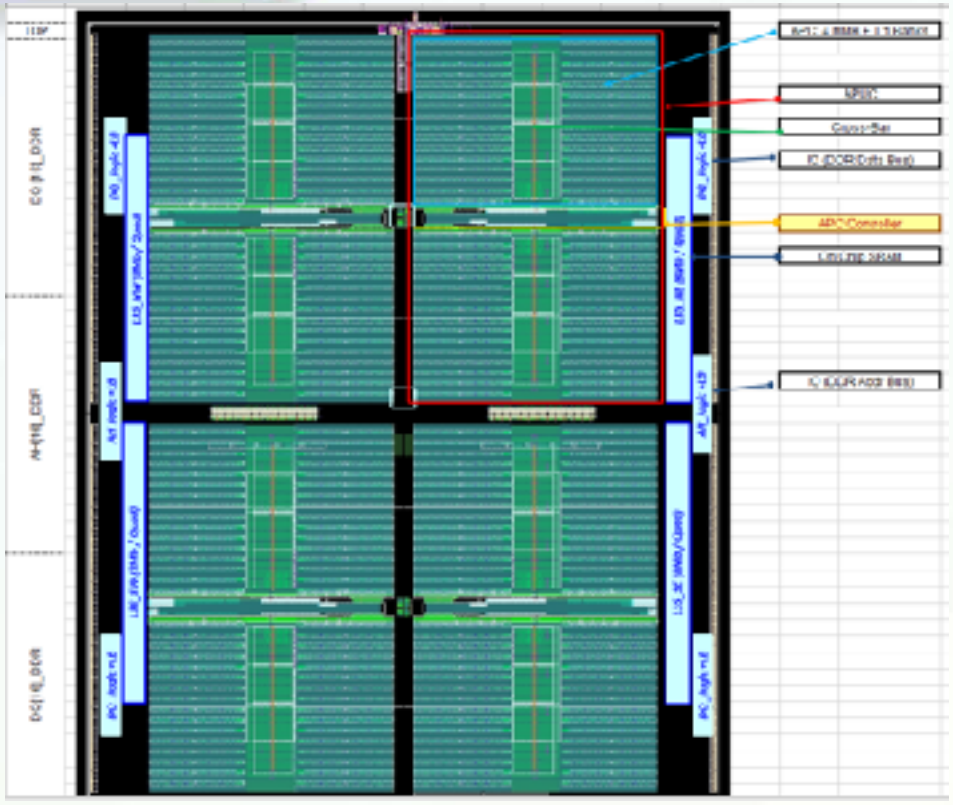
COMMUNICATION BETWEEN SECTIONS



Shift between sections enable neighborhood operations (filters , CNN etc.)

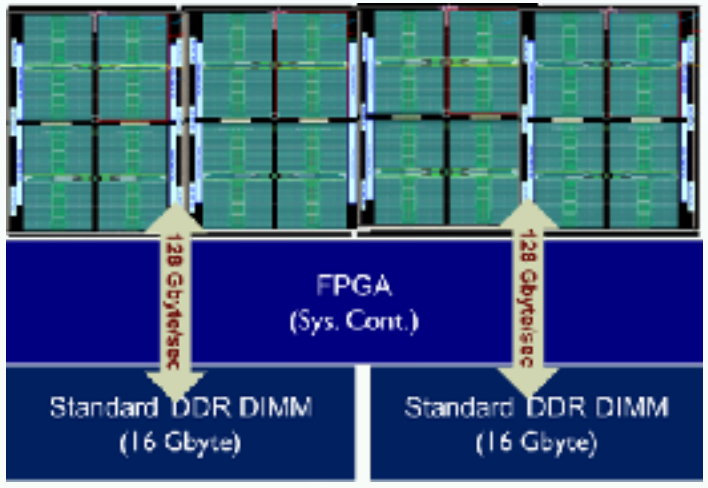
Store, compute, search and move data anywhere

APU CHIP LAYOUT



2M bit processors or 128K vector processors runs at 1G Hz
with up to 2 Peta OPS peak performance

EVALUATION BOARD PERFORMANCE



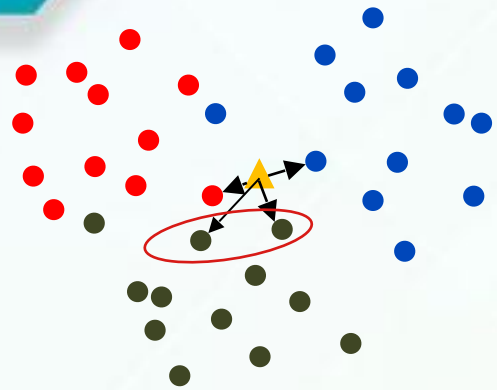
- Precision :
  - Unlimited : from 1 bit to 160 bits or more.
  - 6.4 TOPS (FP) – 8 Peta OPS for one bit computing or 16 bit exact search
- Similarity Search, Top-k , min, max , Softmax,
  - O(1) complexity in µs, any size of K compared to ms with current solutions
- In-memory IO
  - 2 Petabit/sec > 100X GPGPU/CPU/FPGA
- Sparse matrix multiplication
  - > 100X GPGPU/CPU/FPGA

APU SERVER

- 64 APU chips , 256-512GByte DDR,
- From 100TFLOPS Up to 128 Peta OPS with peak performance 128TOPS/W
- O(1) Top-K, min, max,
- 32 Peta bits/sec internal IO
- < 1K Watts
- > 1000X GPGPs on average
- Linearly scalable
- Currently 28nm process and scalable to 7nm or less
  - Well suited to advanced memory technology such as non volatile ReRAM and more

EXAMPLE APPLICATIONS

K-NEAREST NEIGHBORS (K-NN)



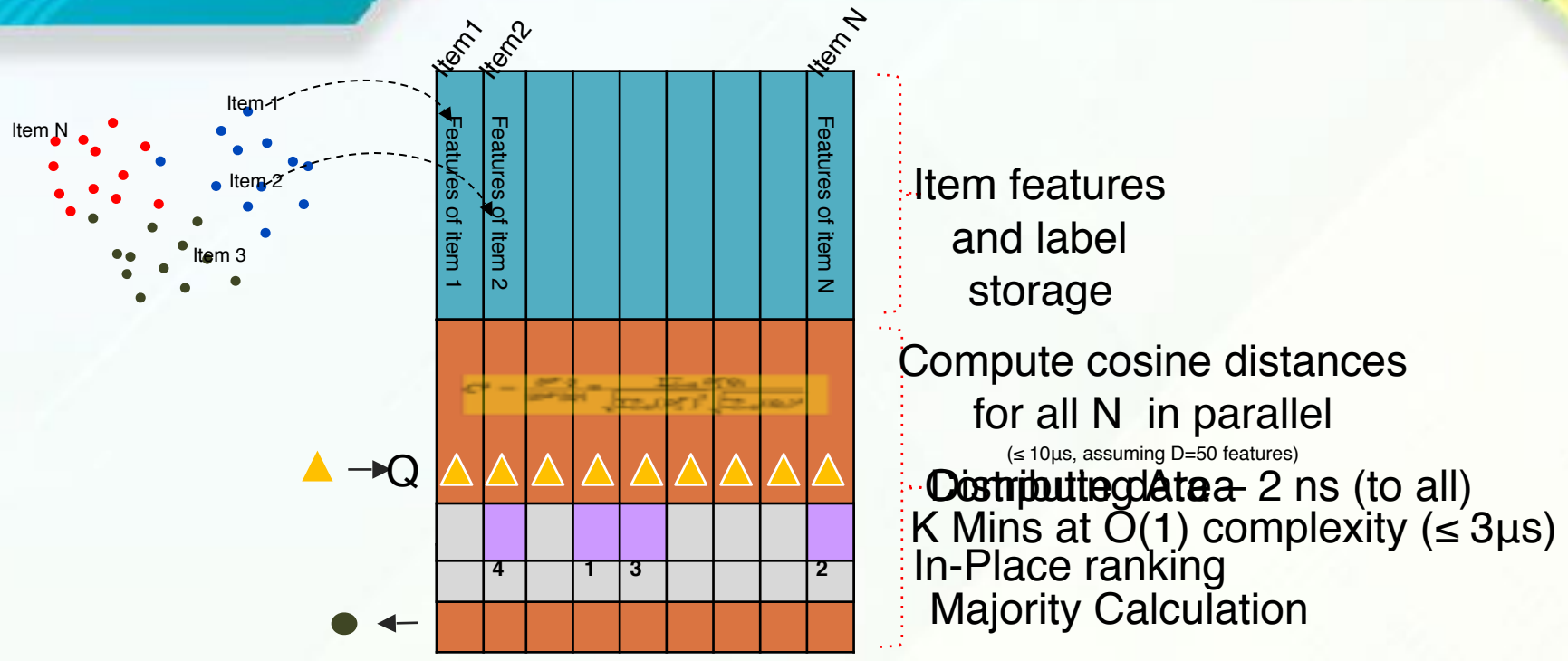
Simple example: N = 36, 3 groups, 2 dimensions (D = 2) for X and Y

K = 4

Group **Green** selected as the majority

For actual applications: N = Billions, D = Tens, K = Tens of thousands

K-NN USE CASE IN AN APU



With the data base in an APU, computation for all N items done in

≤ 0.05 ms, independent of K

LARGE DATABASE EXAMPLE USING APU SERVERS

- Number of items: billions
- Features per item: tens to hundreds
- Latency: ≤ 1 msec
- Throughput: Scales to 1M similarity searches/sec
- k-NN: Top 100,000 nearest neighbors

EXAMPLE K-NN FOR RECOGNITION

Image

Text

Convolution Layer

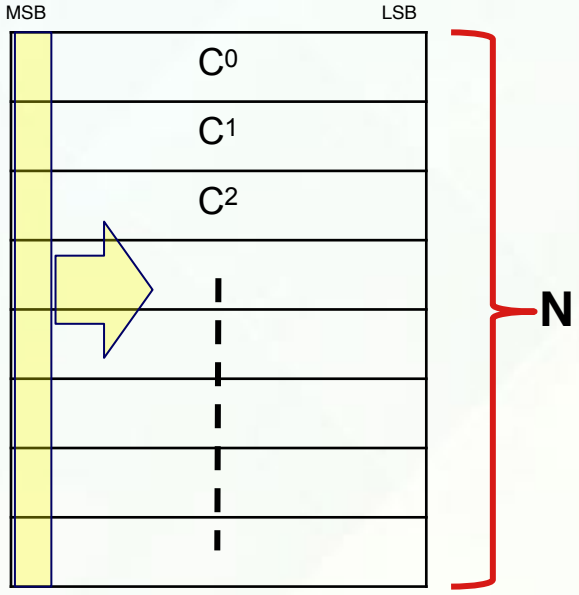
Feature Extractor (Neural Network)

BOW, Word Embedding

K-NN Classifier (Associative Memory)

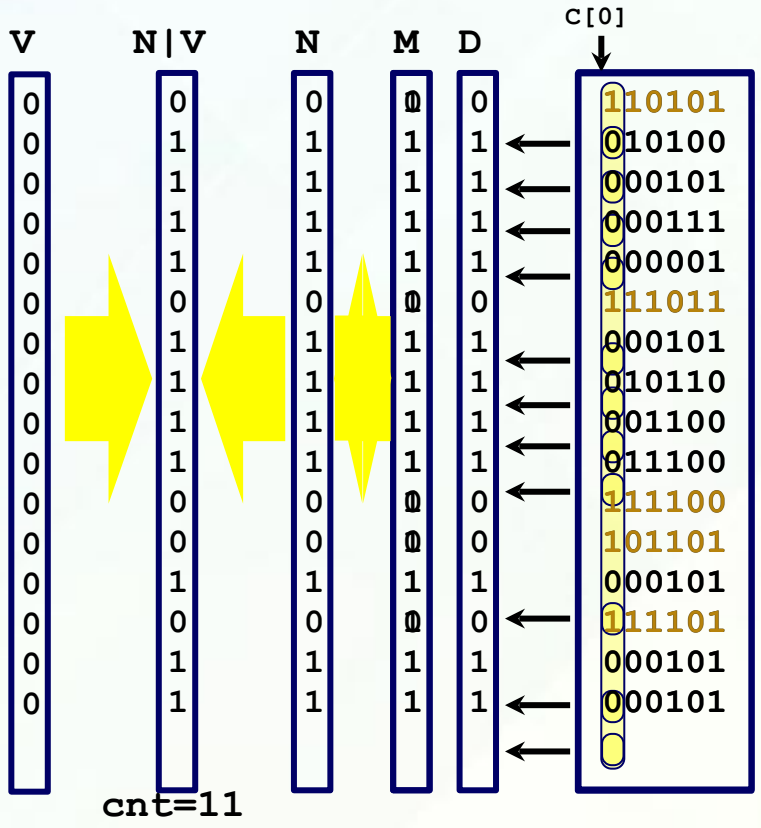
K-MINS: O(1) ALGORITHM

```
KMINS(int K, vector C){
  M := 1, V := 0;
  FOR b = msb to b = lsb:
    D := not(C[b]);
    N := M & D;
    cnt = COUNT(NIV)
    IF cnt > K:
      M := N;
    ELIF cnt < K:
      V := NIV;
    ELSE: // cnt == K
      V := NIV;
      EXIT;
    ENDIF
  ENDFOR
}
```



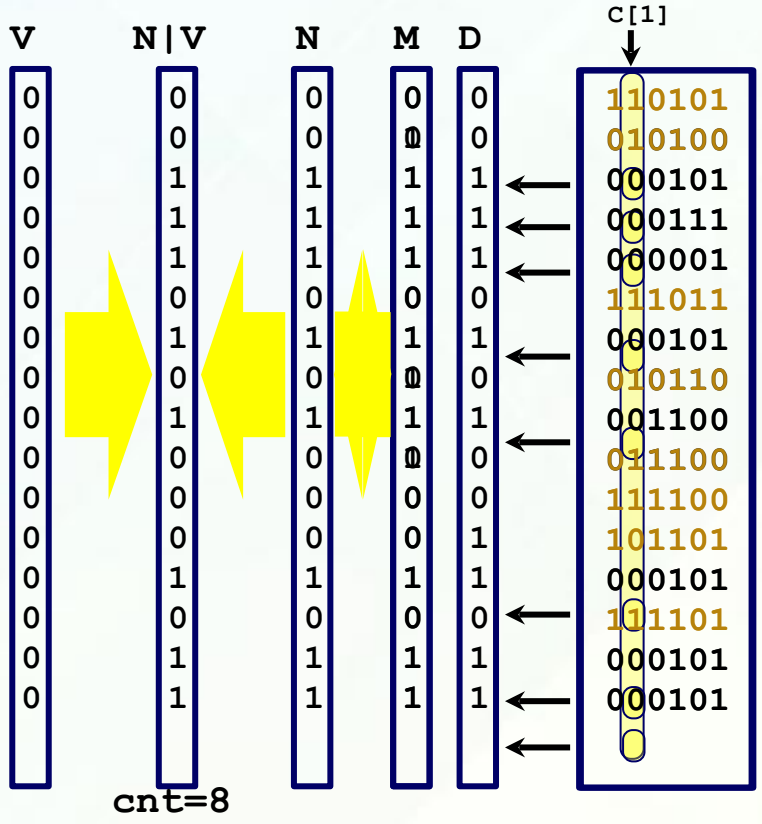
K-MINS: THE ALGORITHM

```
KMINS(int K, vector C){
  M := 1, V := 0;
  FOR b = msb to b = lsb:
    D := not(C[b]);
    N := M & D;
    cnt = COUNT(NIV)
    IF cnt > K:
      M := N;
    ELIF cnt < K:
      V := NIV;
    ELSE: // cnt == K
      V := NIV;
      EXIT;
    ENDIF
  ENDFOR
}
```



K-MINS: THE ALGORITHM

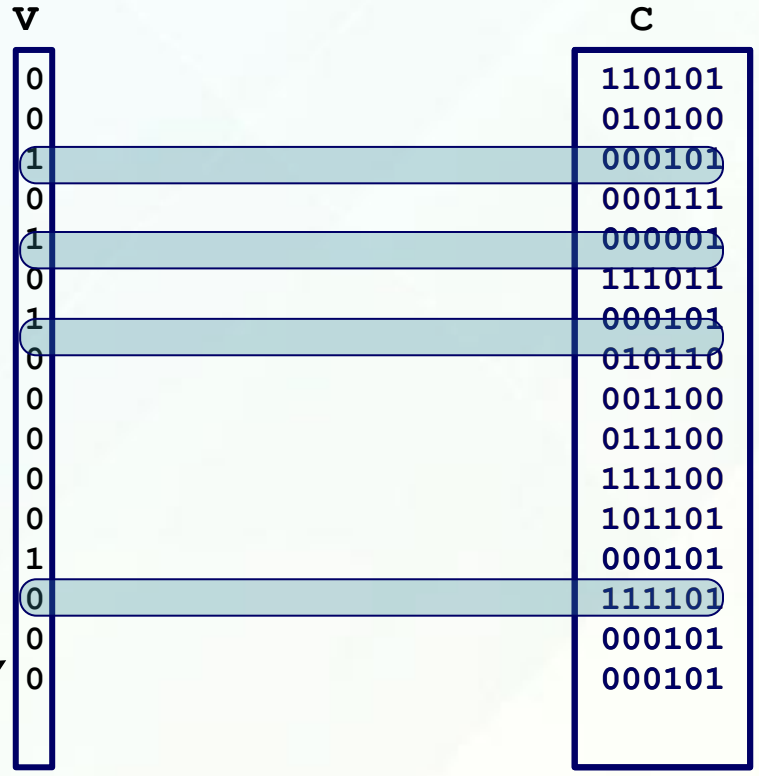
```
KMINS(int K, vector C){
  M := 1, V := 0;
  FOR b = msb to b = lsb:
    D := not(C[b]);
    N := M & D;
    cnt = COUNT(NIV)
    IF cnt > K:
      M := N;
    ELIF cnt < K:
      V := NIV;
    ELSE: // cnt == K
      V := NIV;
      EXIT;
    ENDIF
  ENDFOR
}
```



K-MINS: THE ALGORITHM

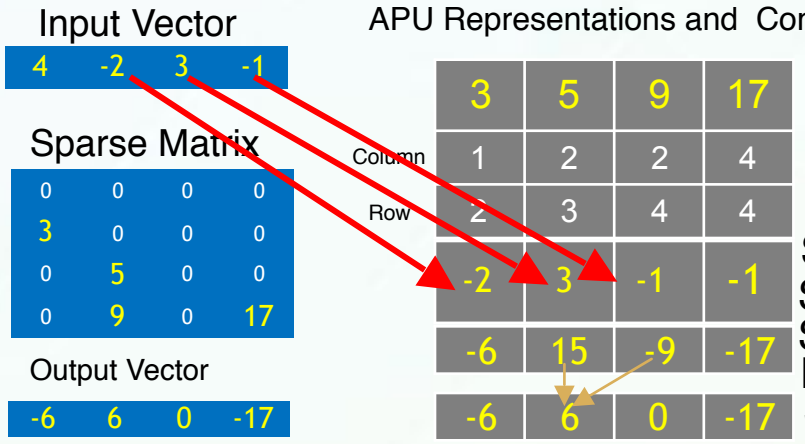
```
KMINS(int K, vector C){
  M := 1, V := 0;
  FOR b = msb to b = lsb:
    D := not(C[b]);
    N := M & D;
    cnt = COUNT(NIV)
    IF cnt > K:
      M := N;
    ELIF cnt < K:
      V := NIV;
    ELSE: // cnt == K
      V := NIV;
      EXIT;
    ENDIF
  ENDFOR
}
```

final output



O(1) Complexity

DENSE (1XN) VECTOR BY SPARSE NXM MATRIX



Search all columns for row = 2 :Distribute -2 : 2 Cy
Search all columns for row = 3 :Distribute 3 : 2 Cy
Search all columns for row = 4 :Distribute -1 : 2 Cy
Multiply in Parallel : 100 Cy
Shift and Add all belonging to same column

Complexity including IO : O (N +logβ)
where β is the number of nonzero elements in the sparse matrix
N << M in general for recommender systems

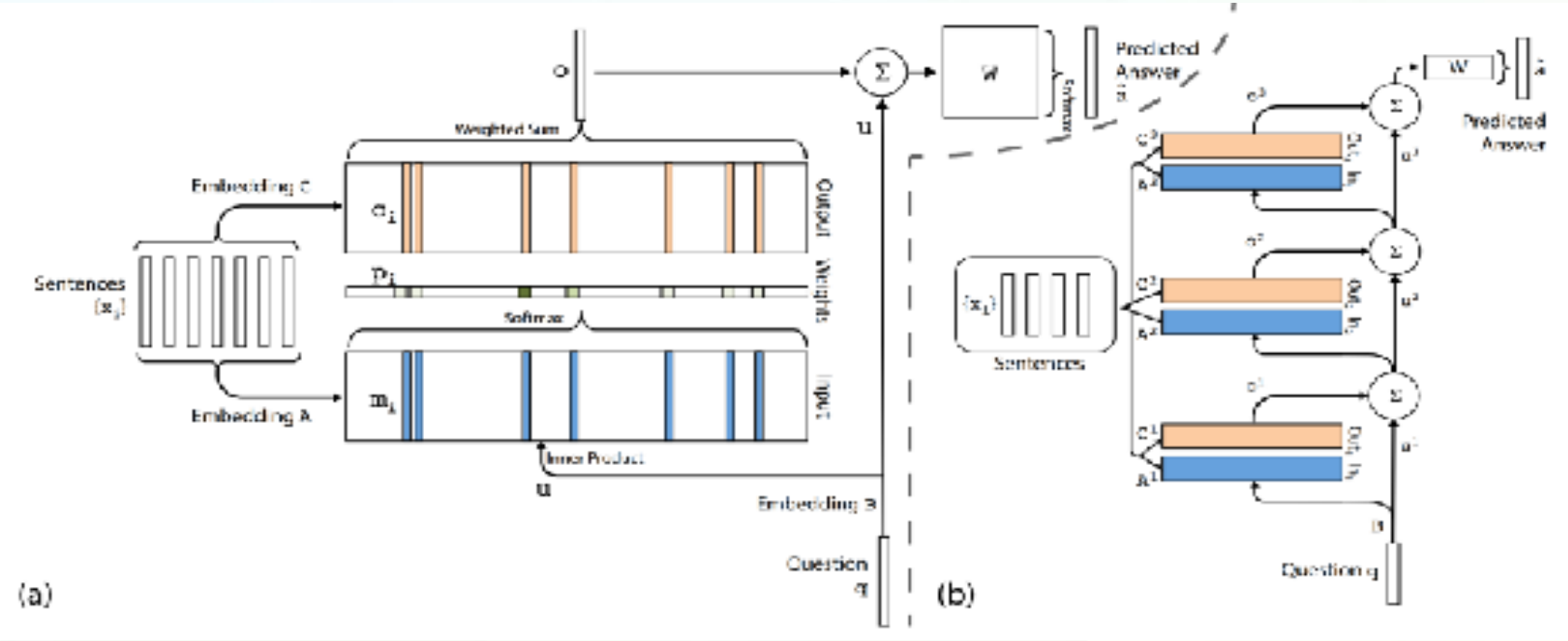
SPARSE MATRIX MULTIPLICATION PERFORMANCE ANALYSIS

- G3 circuit matrix
  - 1.5M X 1.5M sparse matrix
  - Roughly 8M nonzero elements
- 20–100 GFLOPS with GPGPU solution
- APU solution provides 64 TFLOPS using the same amount of power as the GPGPU solution above
- > 500x improvement with APU solution

ASSOCIATIVE MEMORY FOR NATURAL LANGUAGE PROCESSING (NLP)

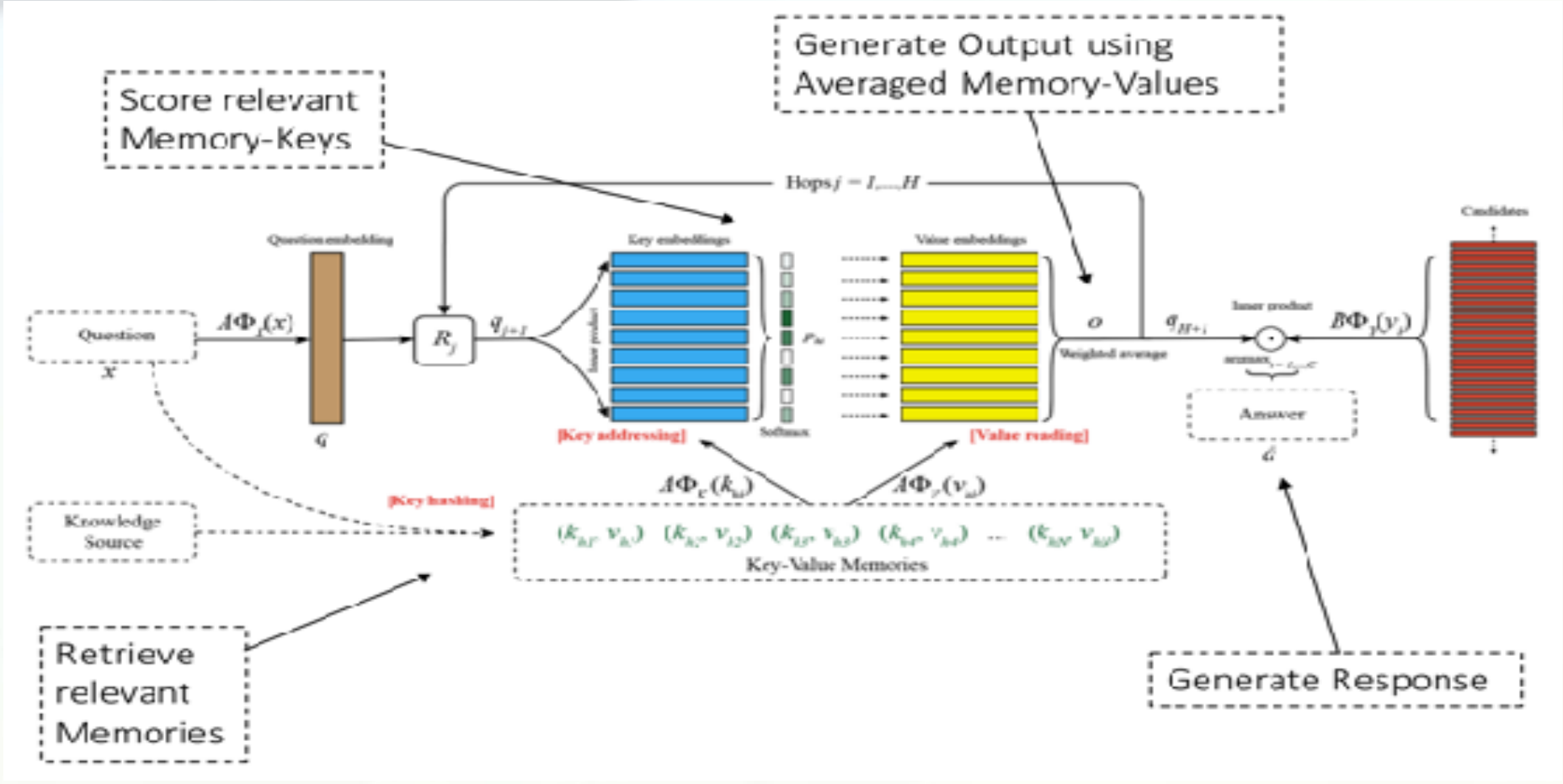
- Q&A, dialog, language translation, speech recognition' etc.
- Requires learning things from the past – needs memory
- More memory, more accuracy
  - i.e. "Dan put the book in his car, ….. Long story here…. Mike took Dan's car … Long story here …. He drove to SF"
  - Q : Where is the book now?
  - A: Car, SF

END-TO-END MEMORY NETWORKS

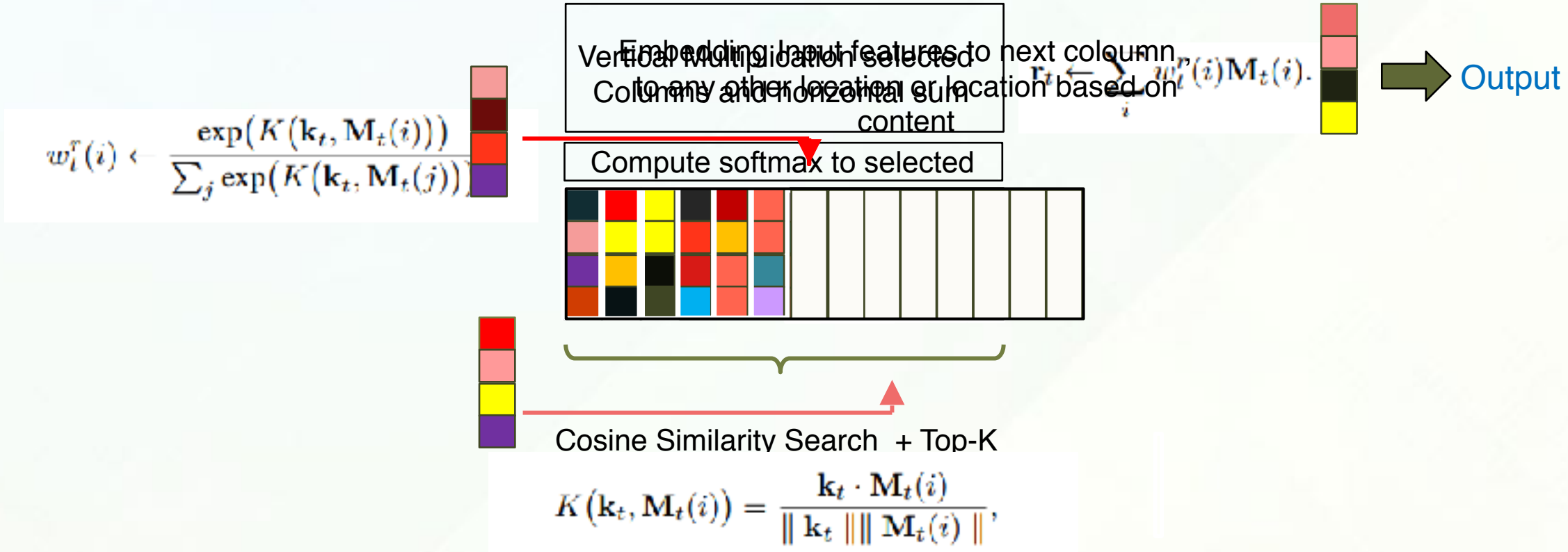


End-To-End Memory Networks, (Weston et. al., NIPS 2015). (a): Single hope, (b) 3 hops

Q&A : END TO END NETWORK



REQUIREMENTS FOR AUGMENTED MEMORY

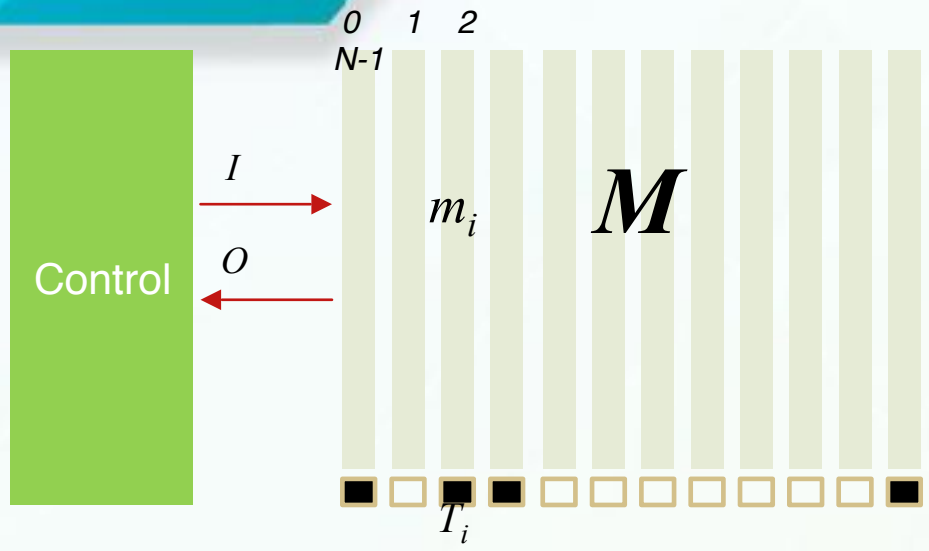


$$w_t^r(i) \leftarrow \frac{\exp\left(K\left(\mathbf{k}_t, \mathbf{M}_t(i)\right)\right)}{\sum_j \exp\left(K\left(\mathbf{k}_t, \mathbf{M}_t(j)\right)\right)}$$

$$\mathbf{r}_t \leftarrow \sum_i w_t^r(i)\mathbf{M}_t(i).$$

$$K\left(\mathbf{k}_t, \mathbf{M}_t(i)\right) = \frac{\mathbf{k}_t \cdot \mathbf{M}_t(i)}{\| \mathbf{k}_t \| \| \mathbf{M}_t(i) \|},$$

APU MEMORY FOR NLP



Broadcast input I to selected columns

Compute any function at selected columns

Generate output

Generate tags for selection

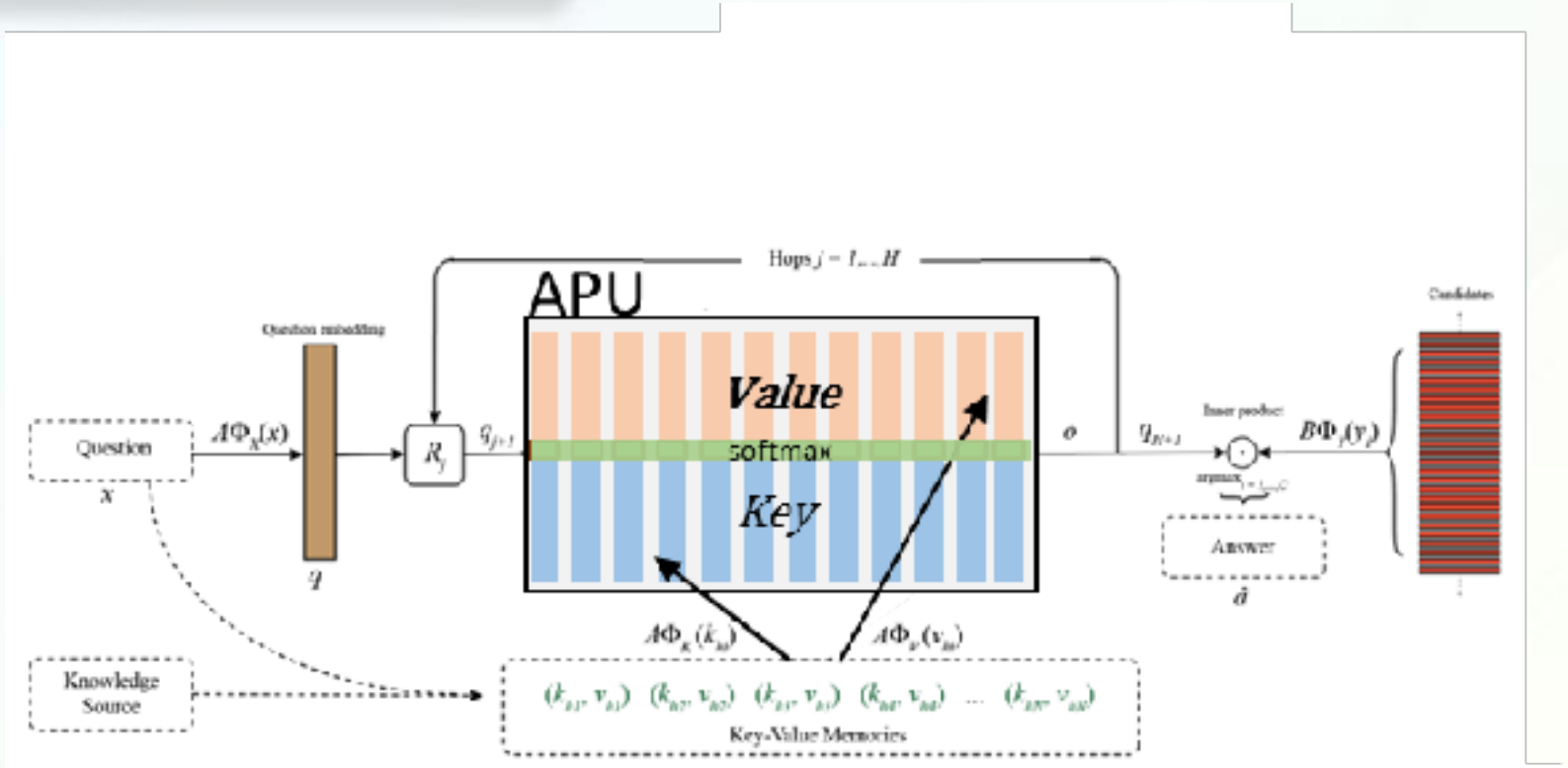
$$m_i = T_i I + \bar{T} m_i$$

$$m_i = T_i f(m_{i\pm s}, M) + \bar{T}_i m_i$$

$$O = z(M)$$

$$T_i = 1 \text{ if } g(m_{i\pm s}, M) \text{ TRUE}; 0 \text{ otherwise}$$

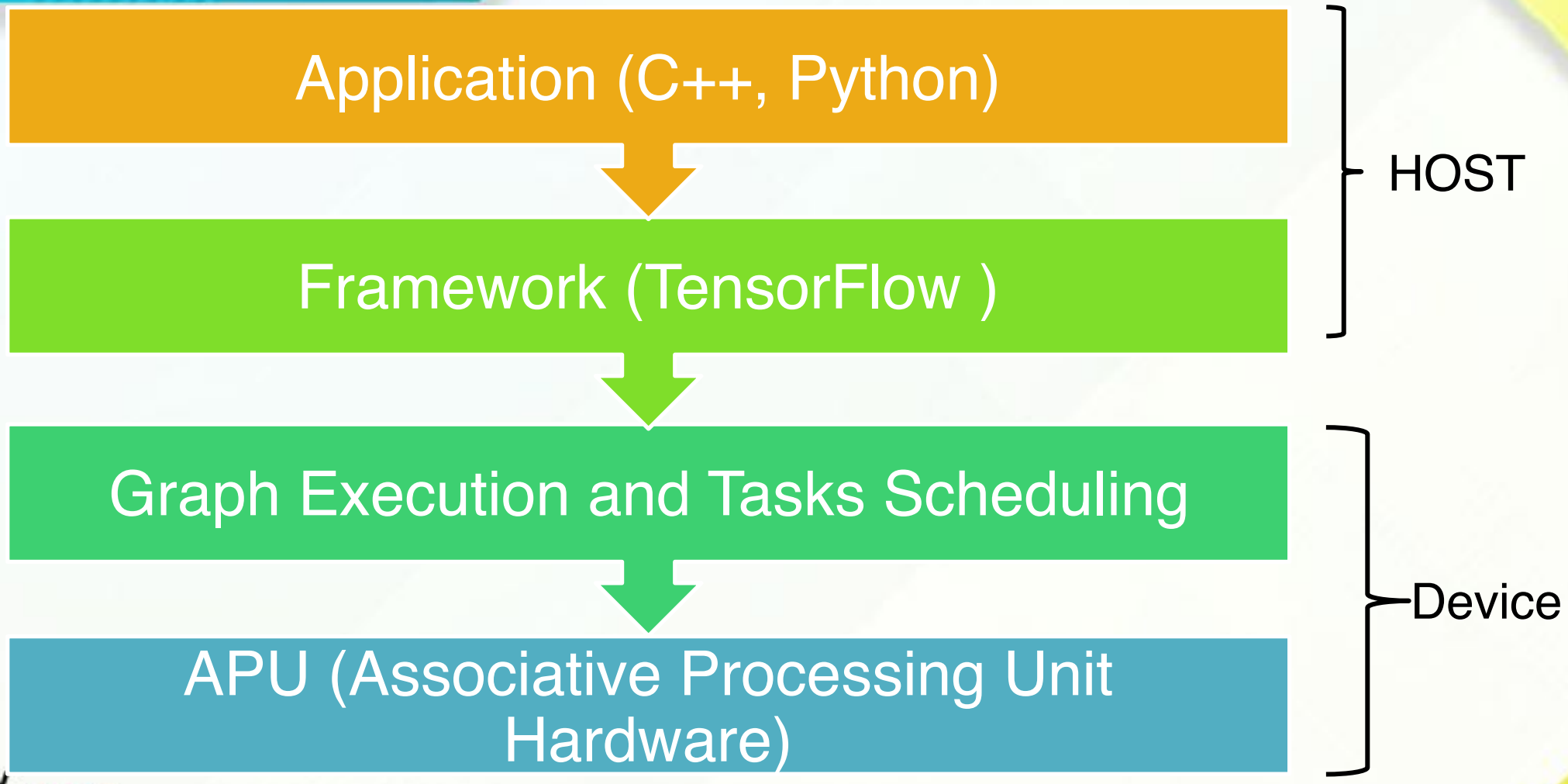
GSI SOLUTION FOR END TO END



Constant time of 3 μsec per iteration, any memory size.

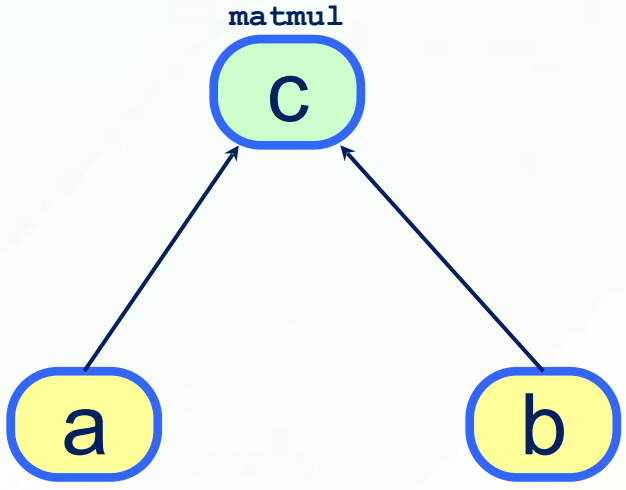
PROGRAMING MODEL

PROGRAMMING MODEL



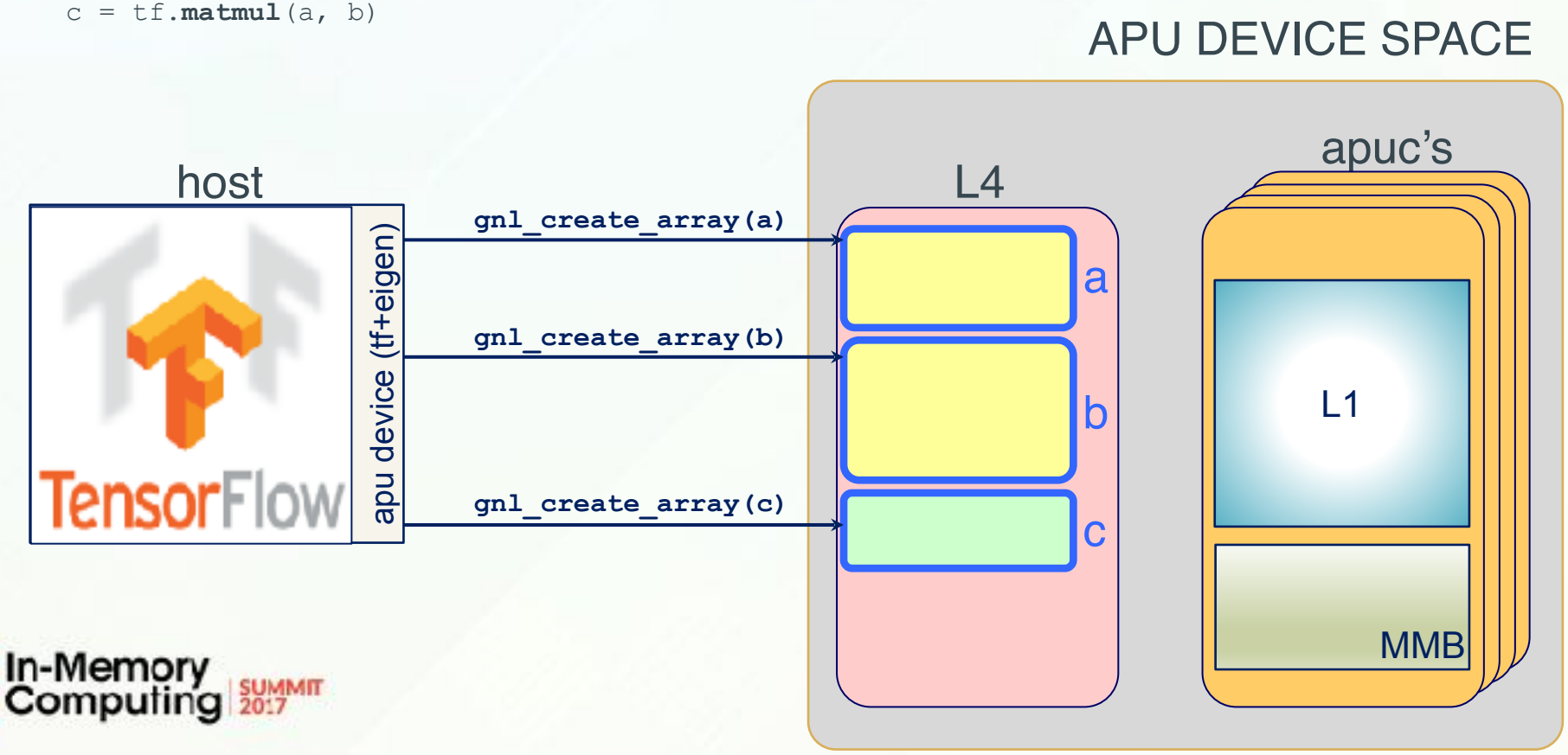
A TF EXAMPLE: MATMUL

```
a = tf.placeholder(tf.int32, shape=[3,4])

b = tf.placeholder(tf.int32, shape=[4,6])

c = tf.matmul(a, b) # shape = [3,6]
```



A TF EXAMPLE: MATMUL GRAPH PREPARATION

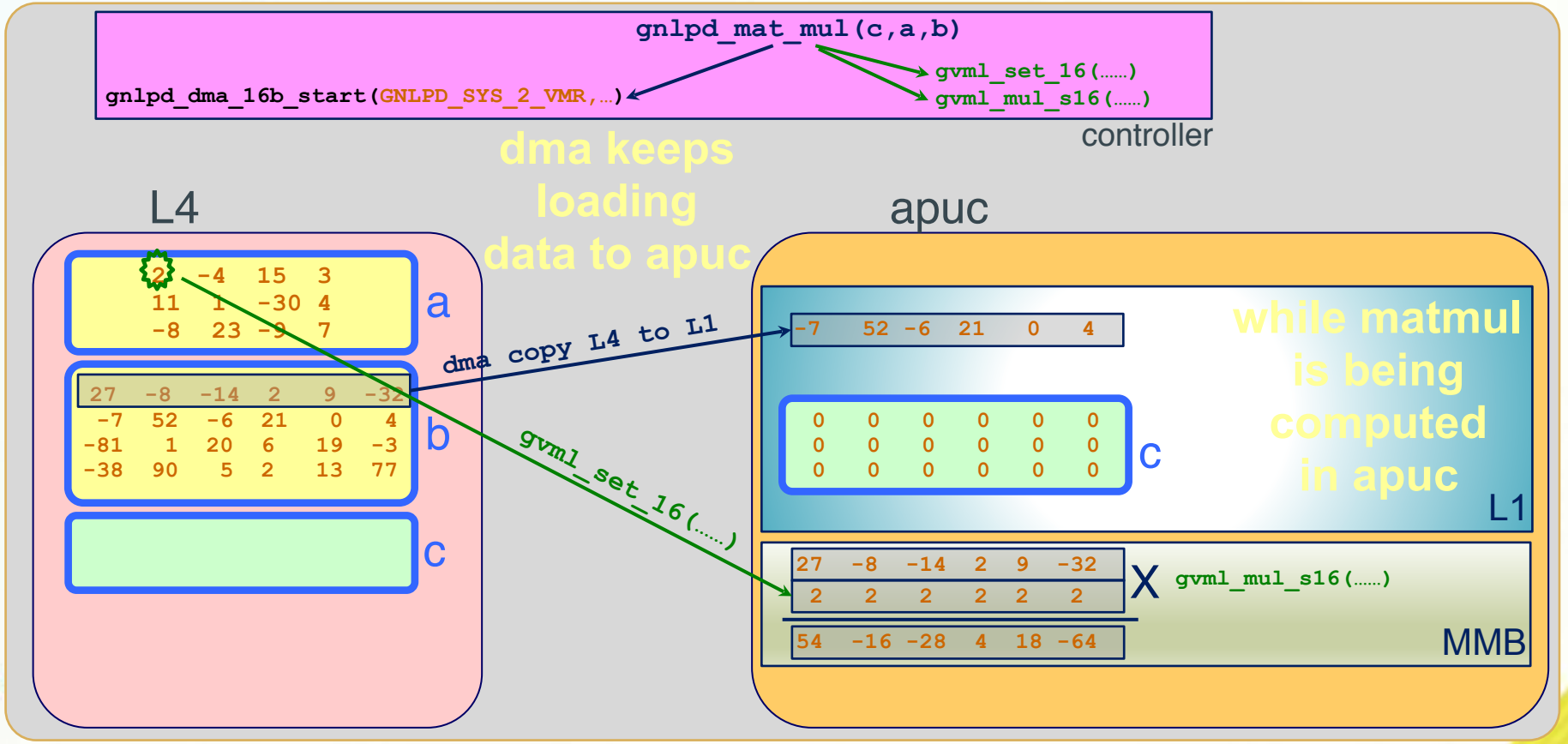
```
a = tf.placeholder(tf.int32, shape=[3, 4])

b = tf.placeholder(tf.int32, shape=[4, 6])

c = tf.matmul(a, b)
```



A TF EXAMPLE: MATMUL GVML_SET, GVML_MUL

```
with tf.Session() as sess:
    result = sess.run(c, feed_dict= {a: [[2  -4  15  3]  b: [[27  -8  -14  2  9  -32]
                                         [11  1  -30 4]       [-7  52  -6  21  0  4]
                                         [-8  23 -9  7]],     [-81  1  20  6  19  -3]
                                                              [-38 90  5   2  13  77]]})
```

APU DEVICE SPACE



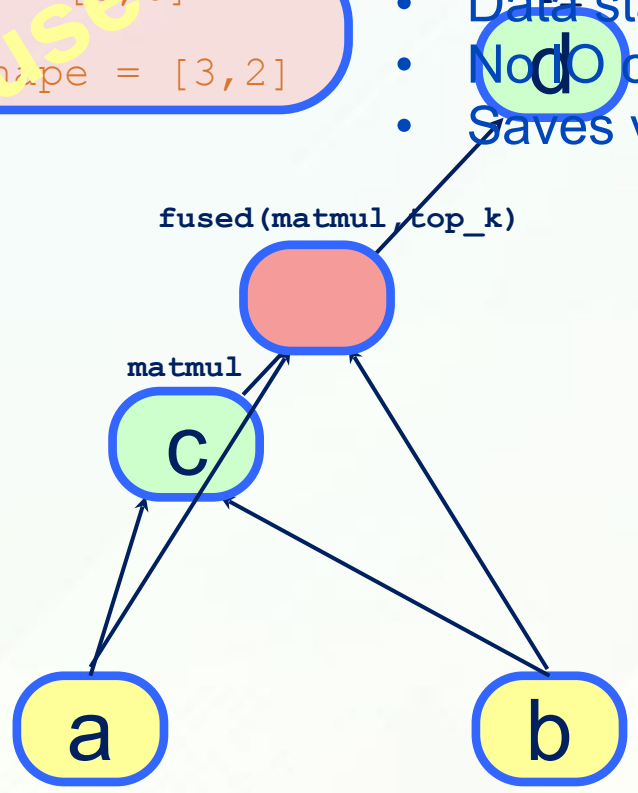
In-Memory Computing SUMMIT 2017

TENSORFLOW ENHANCEMENT: FUSED OPERATIONS

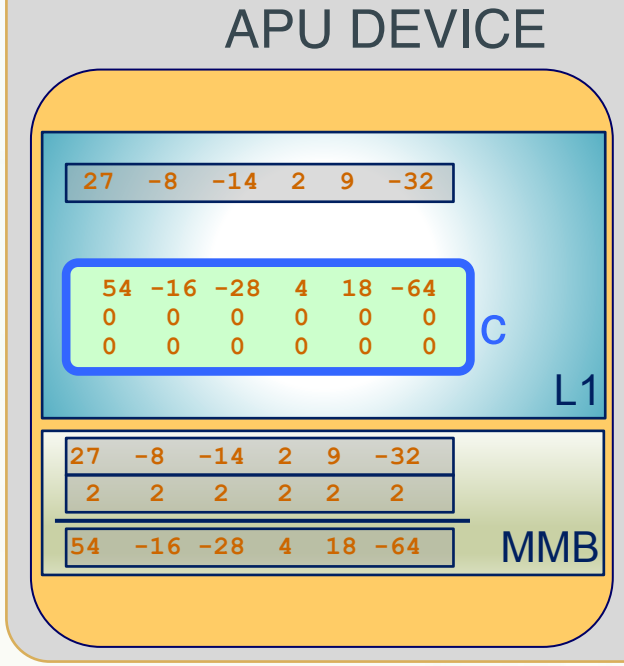
```
a = tf.placeholder(tf.int32, shape=[3,4])
b = tf.placeholder(tf.int32, shape=[4,6])
c = tf.matmul(a, b) # shape = [3,6]
d = tf.nn.top_k(c, k=2) # shape = [3,2]
```

fused

- The two operations are computed inside the apuc
- Data stays in L1
- No IO operations between them
- Saves valuable data transfer time and power



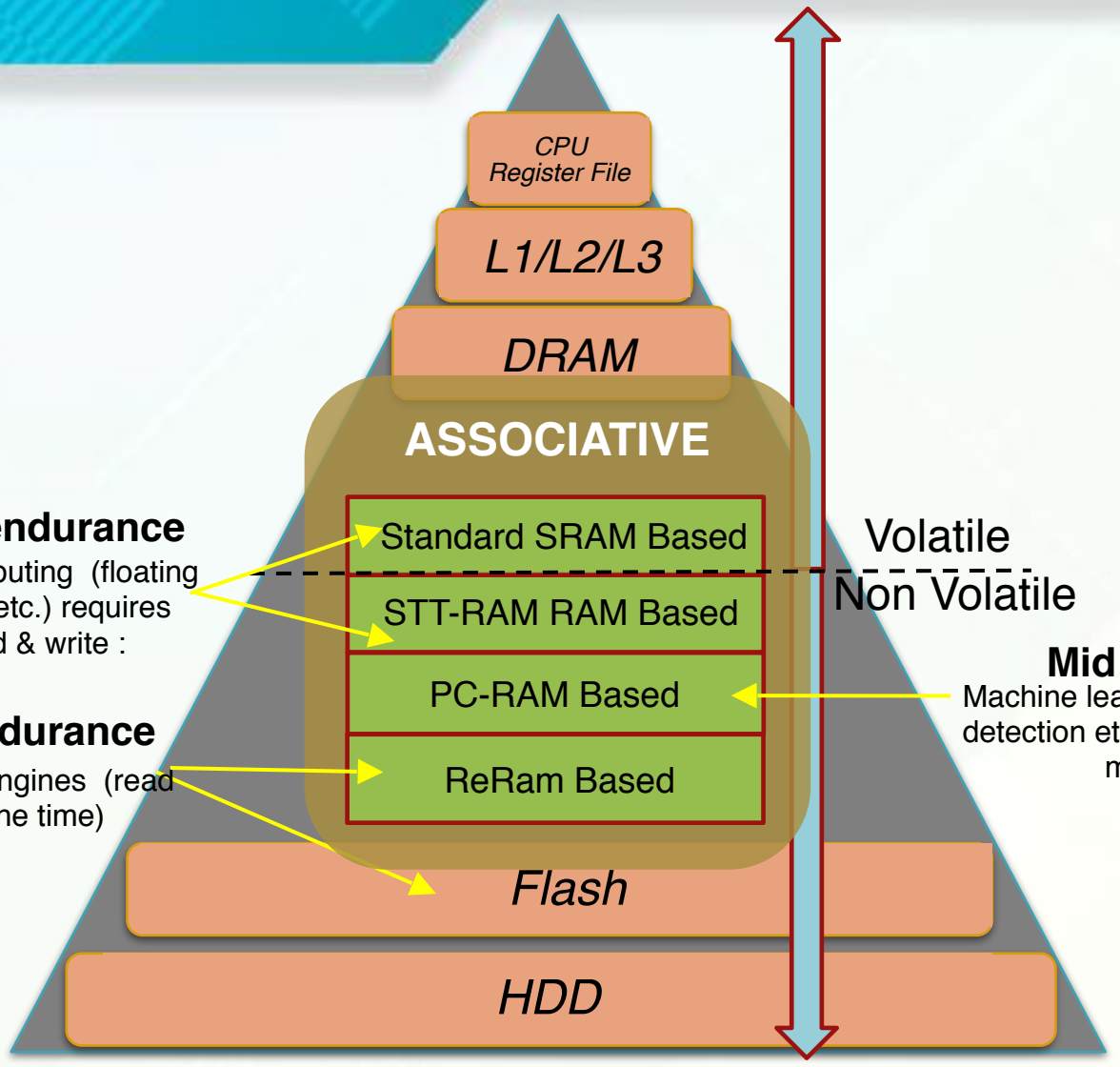
A TF EXAMPLE: MATMUL CODE EXAMPLE




```
APL_FRAG add_u16(RN_REG x, RN_REG y, RN_REG t_xory, RN_REG t_ci0)
{
    SM_0XFFFF:              RL = SB[x];   // RL[0-15] = x
    SM_0XFFFF:              RL |= SB[y]; // RL[0-15] = x|y
    {
        SM_0XFFFF:  SB[t_xory] = RL;  // t_xory[1-15] = x|y
        SM_0XFFFF:  RL = SB[x , y];   // RL[0-15] = x&y
    }
    // Add init state:
    //        0:            RL = co[0]
    //        1..15:        RL = x&y
    {
        (SM_0X1111 << 1):  SB[t_ci0] = NRL;         // t_ci0[5,9,13] = x&y
        (SM_0X1111 << 1):  RL |= SB[t_xory] & NRL; // RL[1] = Cout[1] = x&y | ci(x|y)
                                                        // 5,9,13: RL = Cout0[5,9,13] = x&y | ci(x|y)
        (SM_0X1111 << 4):  RL |= SB[t_xory];        // RL[4,8,12] = Cout1[4,8,12] = x&y | 1&(x|y)
    }
    {
        (SM_0X1111 << 2):  SB[t_ci0] = NRL;         // Propagate Cin0
        (SM_0X1111 << 2):  RL |= SB[t_xory] & NRL; // Propagate Cout0
        (SM_0X1111 << 5):  RL |= SB[t_xory] & NRL; // Propagate Cout1
    }
    {
        (SM_0X1111 << 3):  SB[t_ci0] = NRL;         // Propagate Cin0
        (SM_0X1111 << 3):  RL |= SB[t_xory] & NRL; // Propagate Cout0
        (SM_0X1111 << 6):  RL |= SB[t_xory] & NRL; // Propagate Cout1
        (SM_0X0001 << 15): GL = RL;
    }
    {
        (SM_0X1111 << 4):  SB[t_ci0] = NRL;         // t_ci0[8,12,16] = Cout0[7, 11, 15]
        SM_0X0001:         SB[t_ci0] = GL;                  // t_ci0[8,12,16] = Cout0[7, 11, 15]
        (SM_0X1111 << 7):  RL |= SB[t_xory] & NRL; // Propagate Cout1
        (SM_0X0001 << 15): GL = RL;
    }
. . .
```

FUTURE APPROACH – NON VOLATILE CONCEPT

SOLUTIONS FOR FUTURE DATA CENTERS



High endurance
Full computing (floating points etc.) requires read & write :

Low endurance
Data search engines (read most of the time)

Volatile
Non Volatile

Mid endurance
Machine learning, malware detection detection etc., : Much more read and much less write

THANK YOU