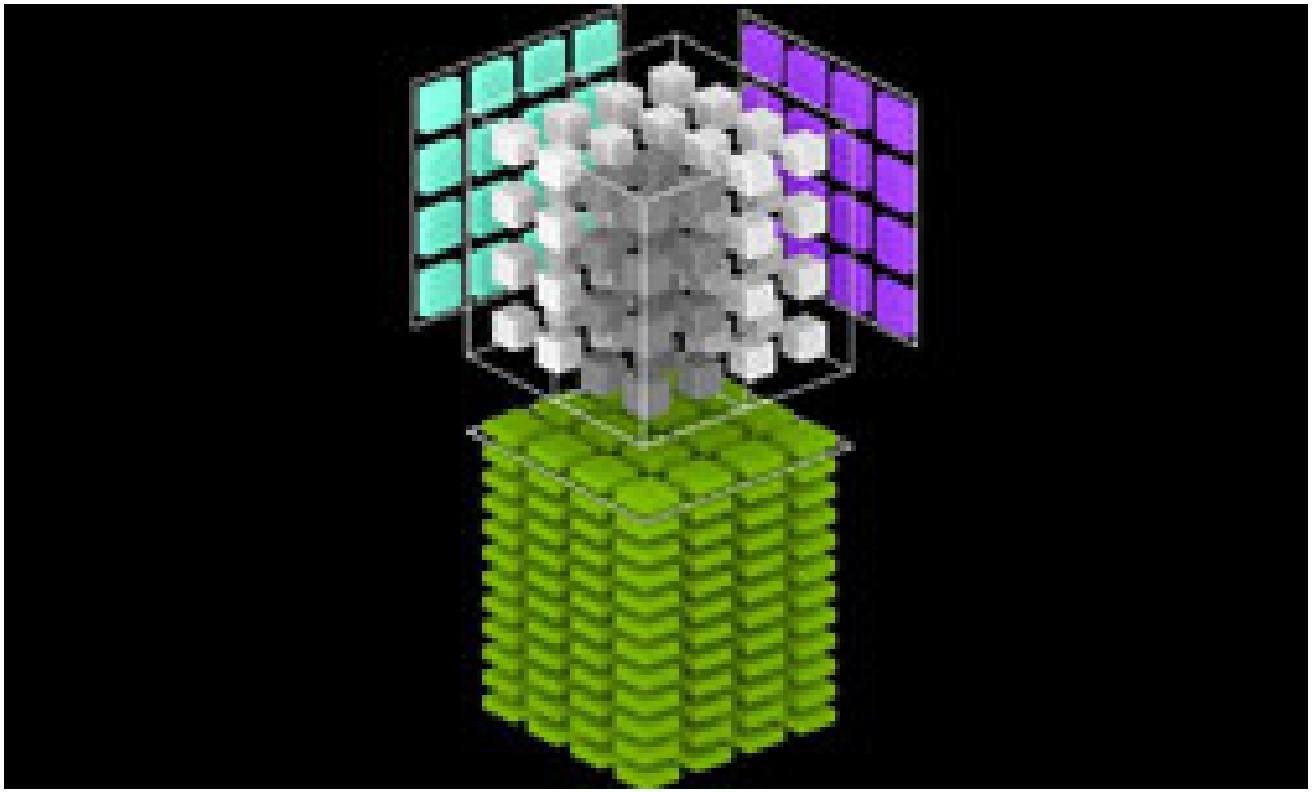
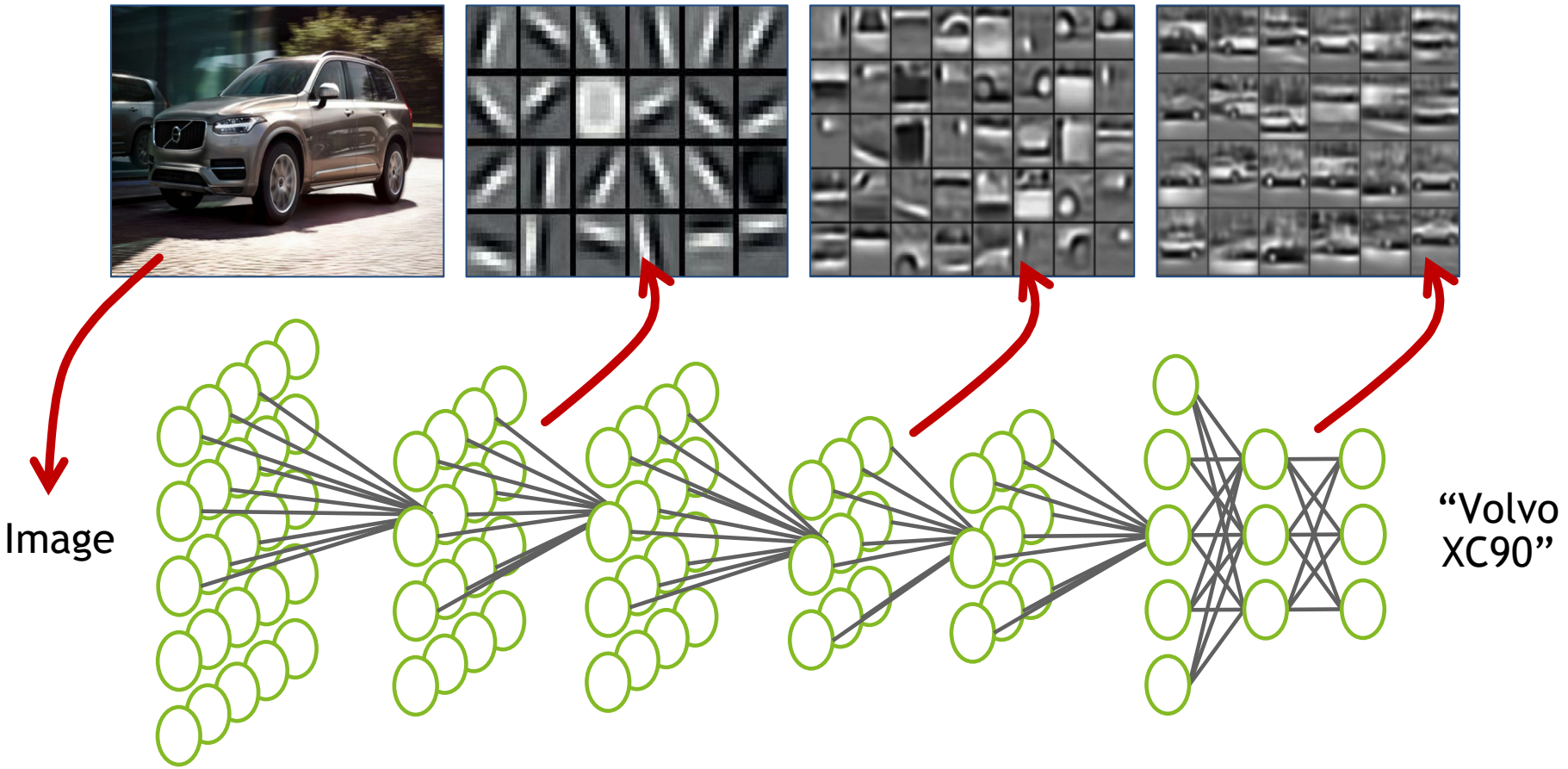
Architectures for Compute-in-Memory



What's a Deep Network?

Google DeepMind AlphaGo

What's a Deep Network?



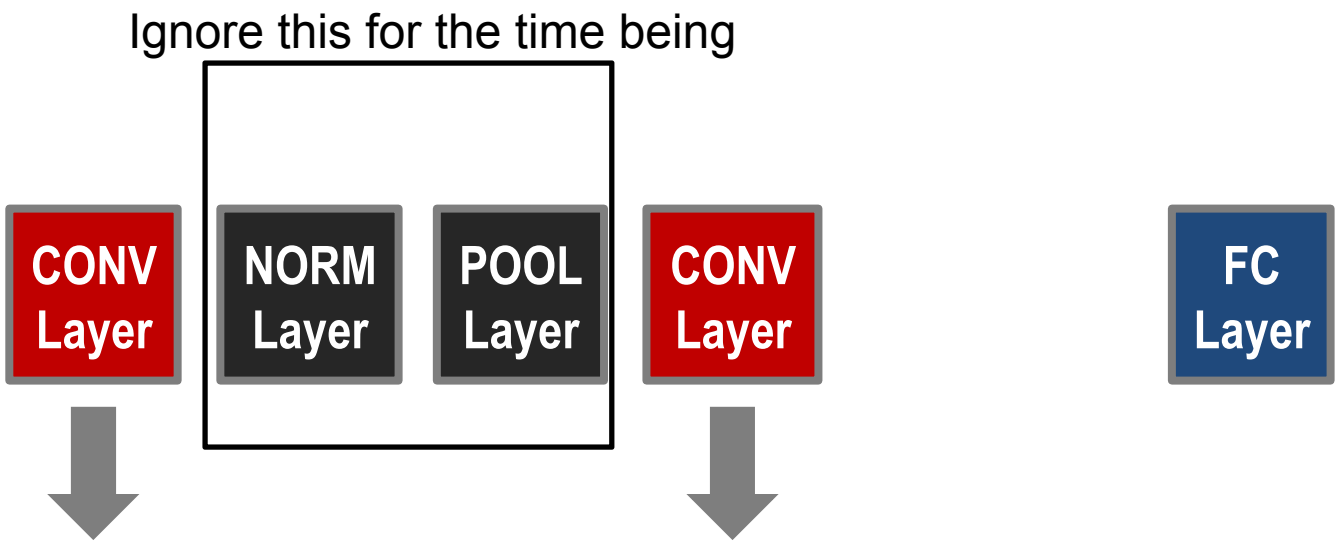
Why Should We Care?

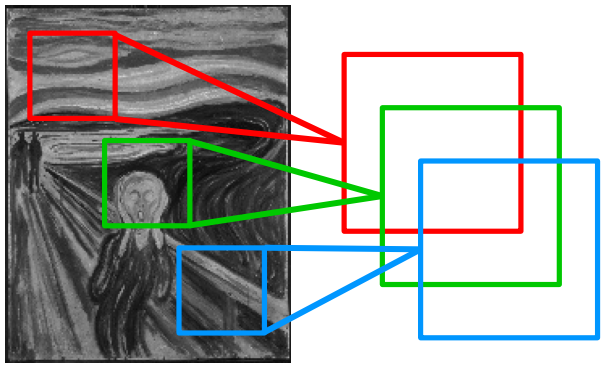
From EE Times – September 27, 2016

”Today the job of training machine learning models is limited by compute, if we had faster processors we’d run bigger models…in practice we train on a reasonable subset of data that can finish in a matter of months. We could use improvements of several orders of magnitude – 100x or greater.”

– Greg Diamos, Senior Researcher, SVAIL, Baidu

A Very Brief Introduction to Neural Nets





Convolutions account for more than 90% of overall computation, dominating **runtime** and **energy consumption**

What is a Convolution?

Naïve 7-layer for-loop implementation:

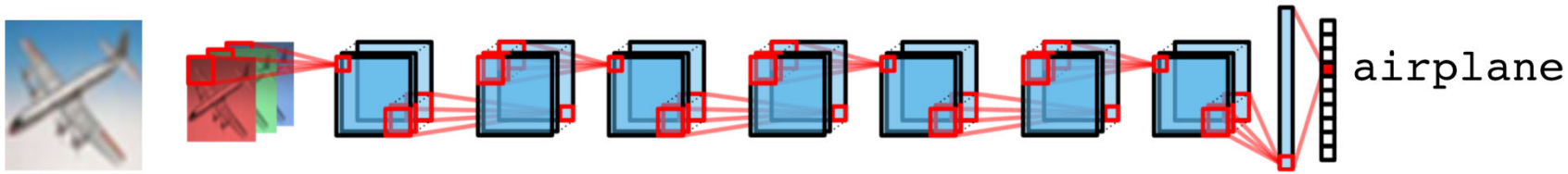
```
for (n=0; n<N; n++) {
    for (m=0; m<M; m++) {
        for (x=0; x<F; x++) {
            for (y=0; y<E; y++) {
                O[n][m][x][y] = B[m];
                for (i=0; i<R; i++) {
                    for (j=0; j<S; j++) {
                        for (k=0; k<C; k++) {
                            O[n][m][x][y] += I[n][k][Ux+i][Uy+j] × W[m][k][i][j];
                        }
                    }
                }

                O[n][m][x][y] = Activation(O[n][m][x][y]);
            }
        }
    }
}
```

for each output fmap value

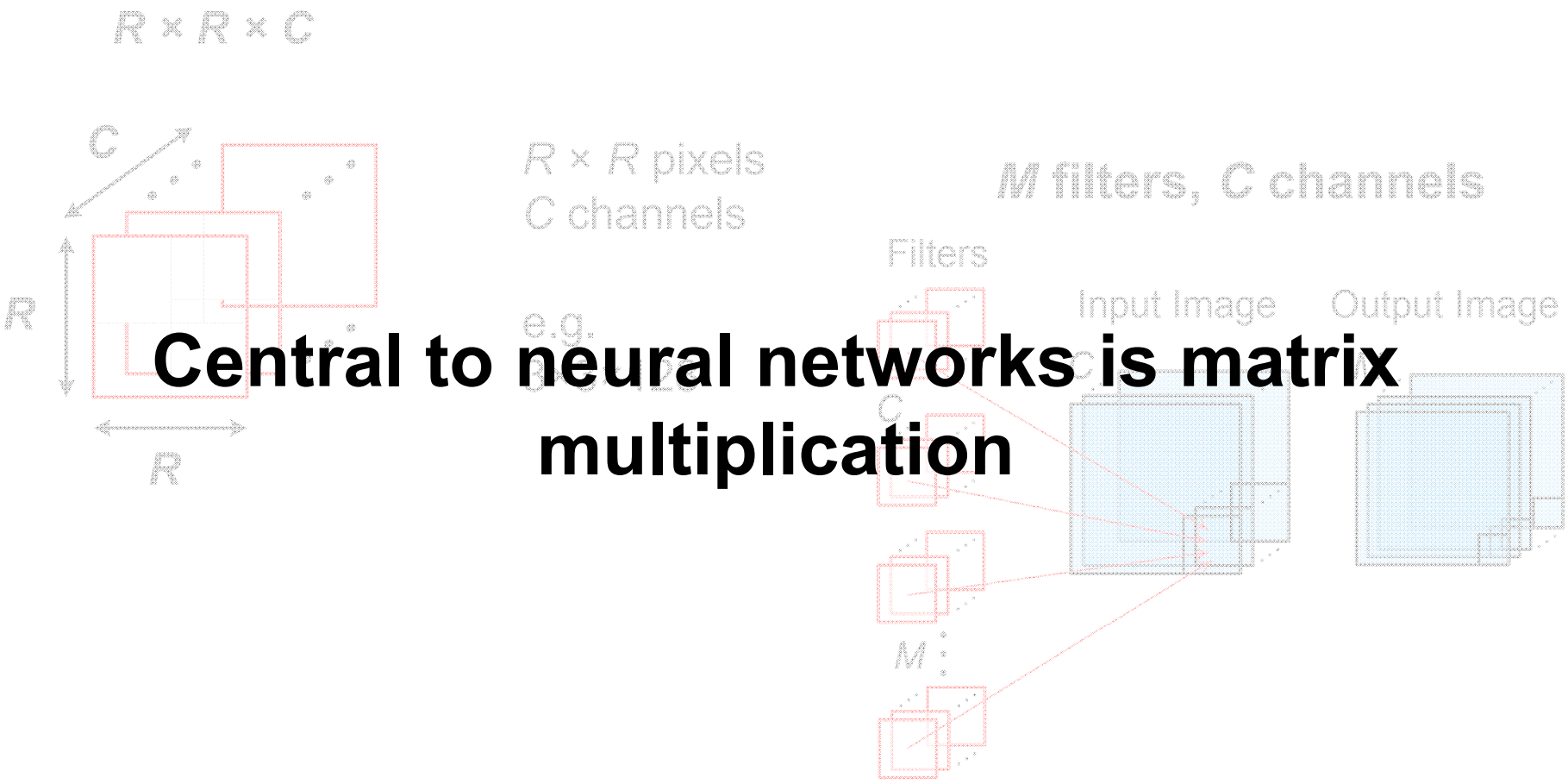
convolve a window and apply activation

What are our computations?



“deep” = multiple stages

multi-channel, multi-filter convolution



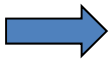
Central to neural networks is matrix multiplication

Matrix Multiplication

$\mathbf{C} = \mathbf{AB}$

$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$

```
for i=1 to n
    for j=1 to n
        for k=1 to n
        c_ij^k = c_ij^k-1 + a_ik b_kj
        end k
    end j
end i
```



$a(i, j, k) = a_{ik}^{\ j}$

$b(i, j, k) = b_{kj}^{\ i}$

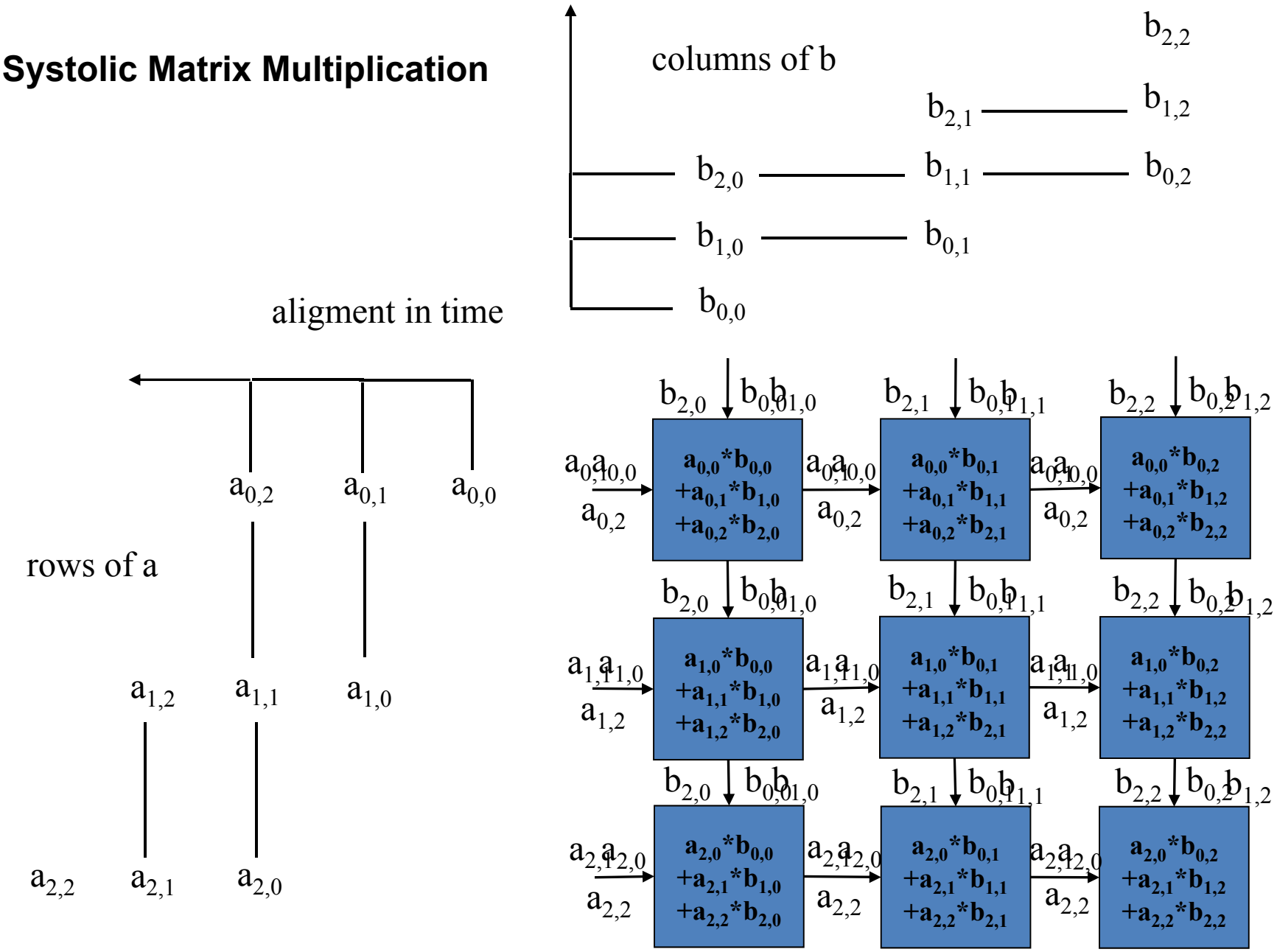
$c(i, j, k) = c_{ij}^{\ k}$

```
for i=1 to n
    for j=1 to n
        for k=1 to n
        a( i, j, k ) = a( i, j-1, k )
        b( i, j, k ) = b( i-1, j, k )
        c( i, j, k ) = c( i, j, k-1 ) + a( i, j, k ) b( i, j, k )
        end k
    end j
end i
```

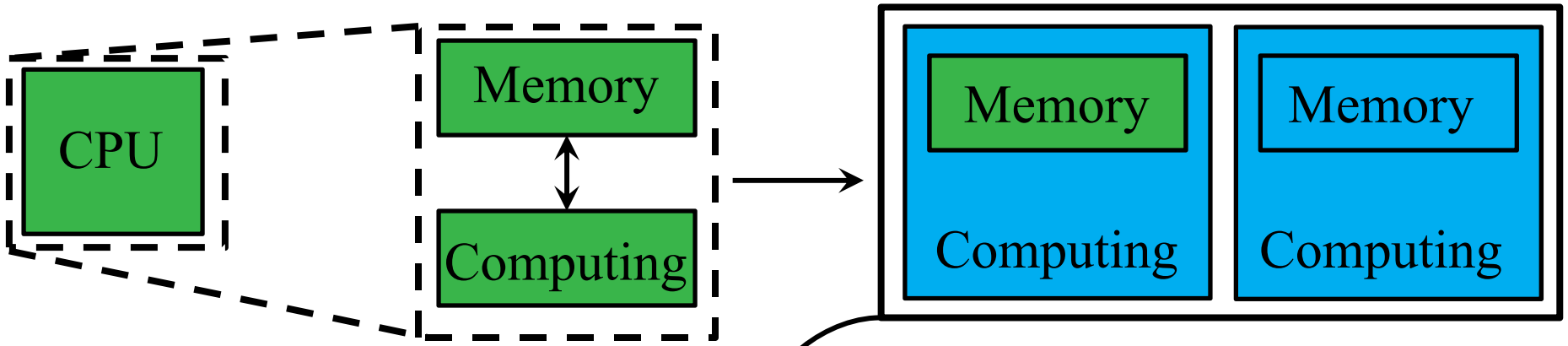
Systolic Matrix Multiplication

- **Processors are arranged in a 2-D grid.**
- **Each processor accumulates one element of the product.**
- **The elements of the matrices to be multiplied are "pumped through" the array.**

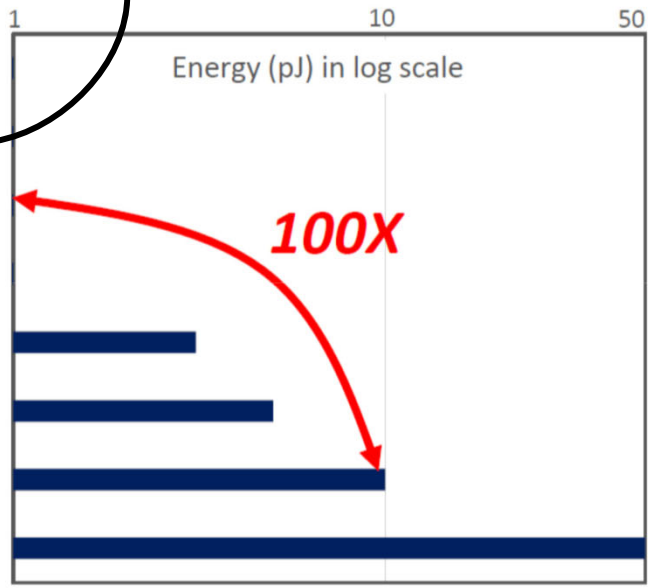
Systolic Matrix Multiplication



Processing in Memory

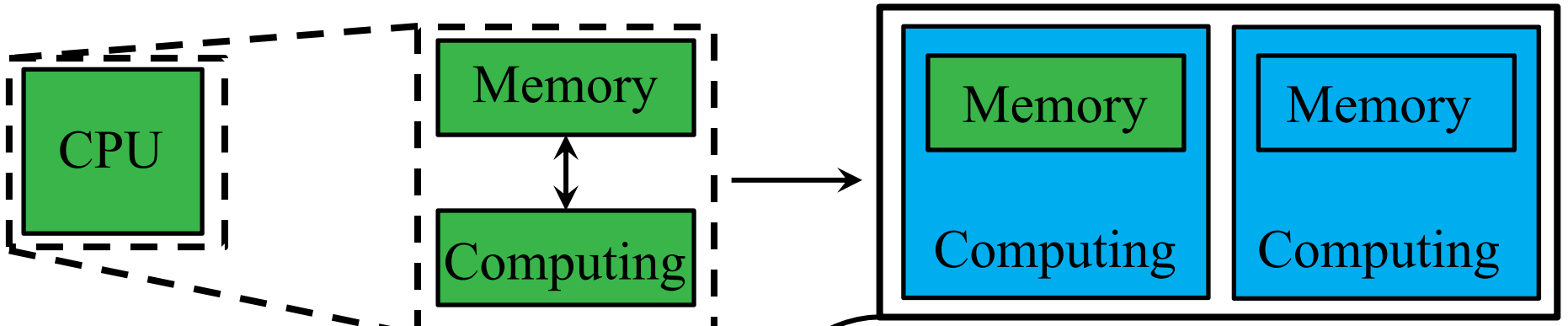


Operation	Energy (pJ)
Integer ADD (8b)	0.03
Integer ADD (16b)	0.05
Integer ADD (32b)	0.1
Integer MULT (8b)	0.2
Integer MULT (32b)	3.1
8KB SRAM Read (32b)	5
32KB SRAM Read (32b)	10
1MB SRAM Read (32b)	50



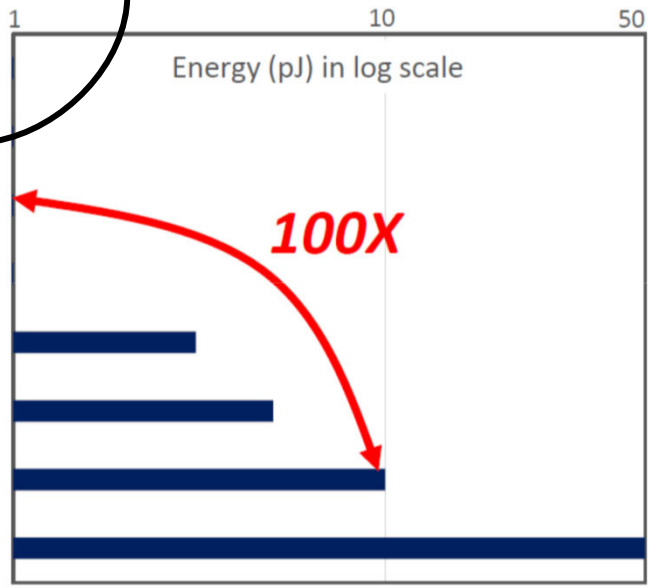
"Computing's Energy Problem (and what we can do about it)", M. Horowitz, ISSCC 2014

Processing in Memory



Digital/Analog processing in memory

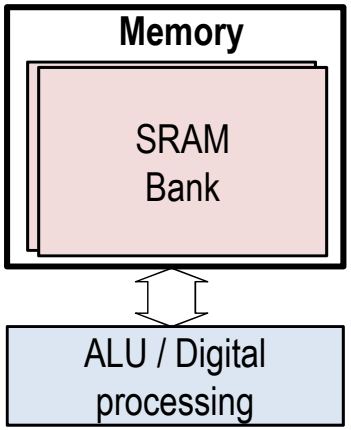
Operation	Energy (pJ)
Integer ADD (8b)	0.03
Integer ADD (16b)	0.05
Integer ADD (32b)	0.1
Integer MULT (8b)	0.2
Integer MULT (32b)	3.1
8KB SRAM Read (32b)	5
32KB SRAM Read (32b)	10
1MB SRAM Read (32b)	50



Processing in memory can bring the combined energy of memory access and computation down to 50 fJ/Op

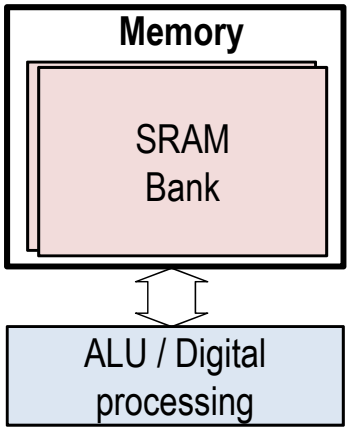
"Computing's Energy Problem (and what we can do about it)", M. Horowitz, ISSCC 2014

digital



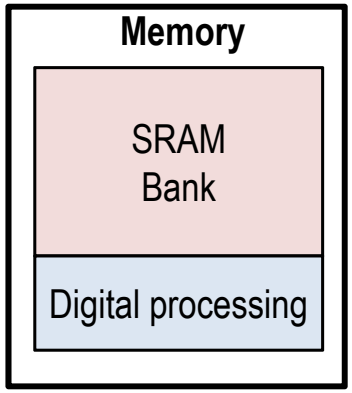
- data access energy and latency dominating
- data reuse and data compression

digital



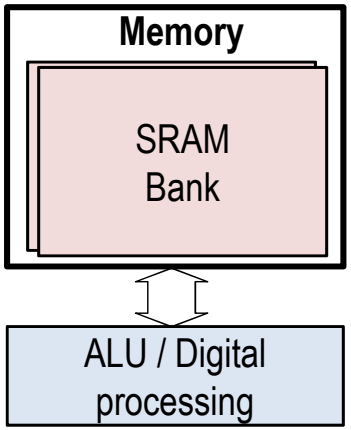
- data access energy and latency dominating
- data reuse and data compression

near memory



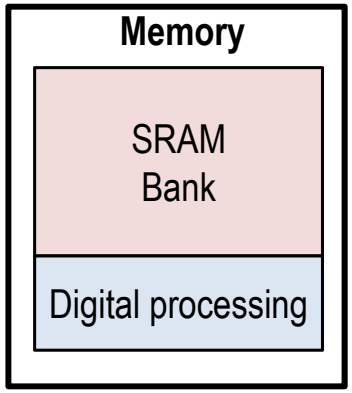
- computation still digital
- eliminates data transfer costs
- memory read energy dominates

digital



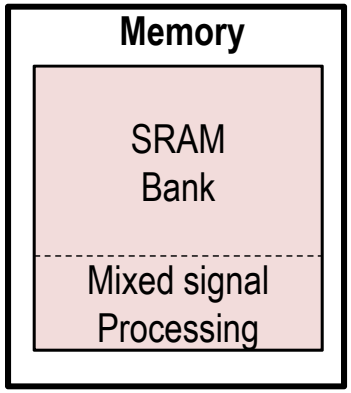
- data access energy and latency dominating
- data reuse and data compression

near memory



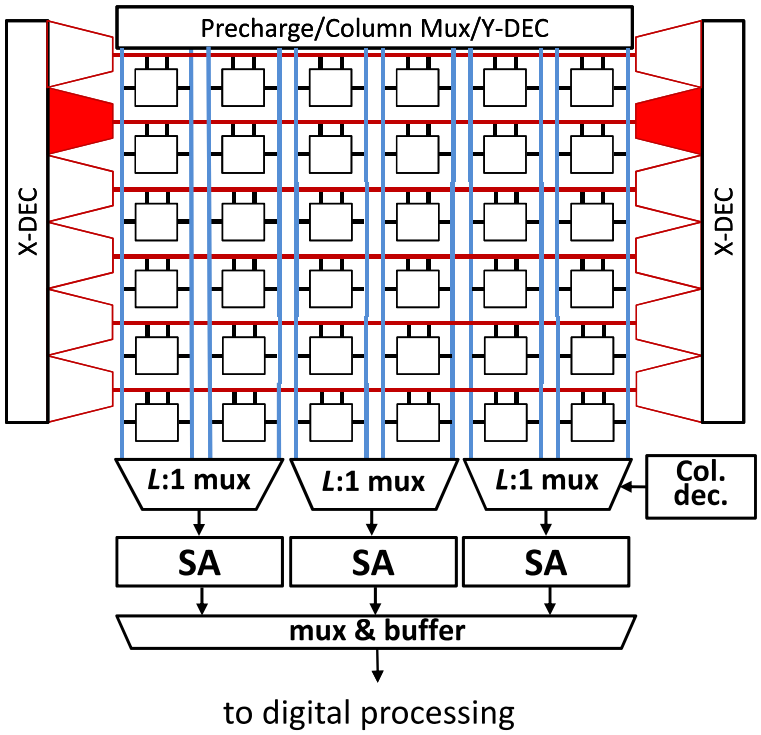
- computation still digital
- eliminates data transfer costs
- memory read energy dominates

deep in-memory



- memory access and computation combined
- mixed signal computation
- significant energy & latency reduction

Standard Memory Design

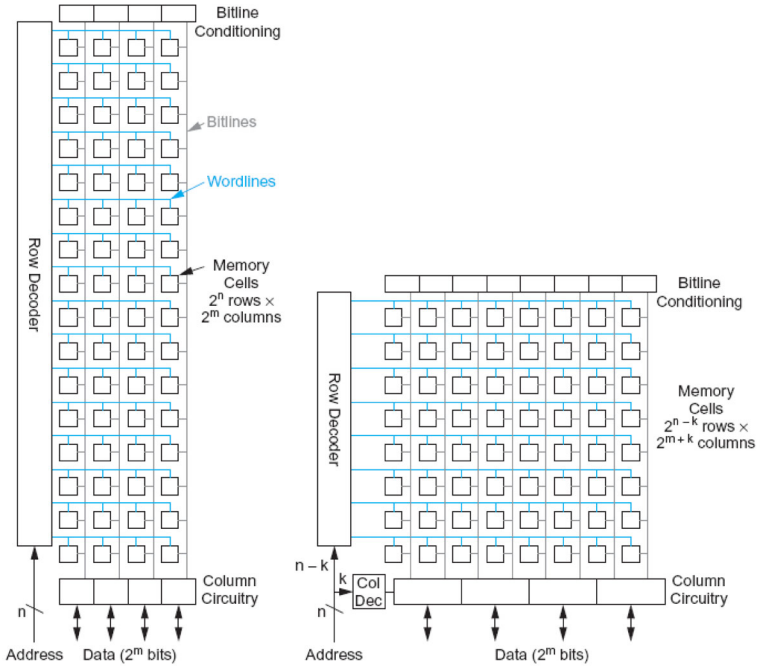


- **energy** and **latency** costs of data access >> those of computation
- single row read per memory access
- column mux before SA
- N_{col}/L bits per access

Standard Memory Design

2^n *words* of 2^m *bits* each

If n >> m, fold by 2^k into fewer *rows* of more *columns*



Good regularity – easy to design

Very high density if good cells are used

Cell size accounts for most of array size

- Reduce cell size at expense of complexity

6T SRAM Cell

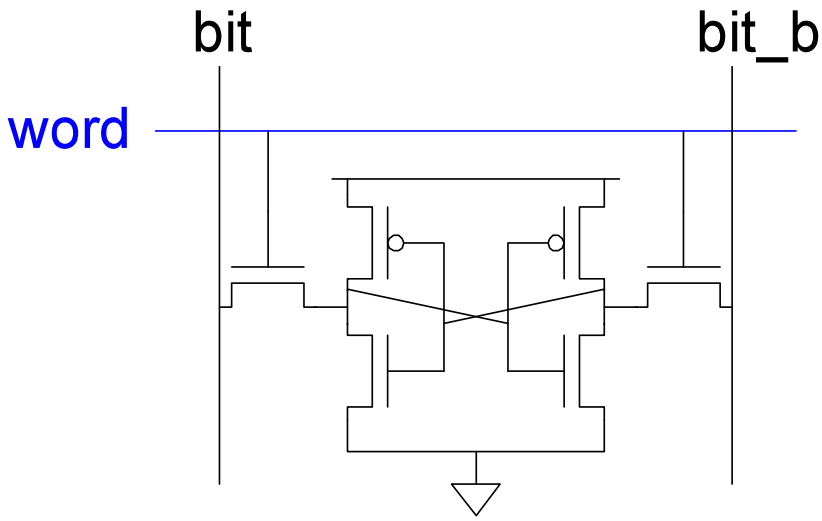
- Used in most commercial chips
- Data stored in cross-coupled inverters

Read:

- Precharge bit, bit_b
- Raise wordline

Write:

- Drive data onto bit, bit_b
- Raise wordline



Precharge both bitlines high

Then turn on wordline

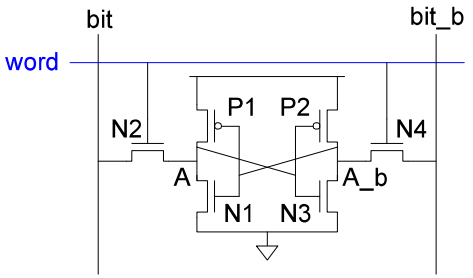
One of the two bitlines will be pulled down by the cell

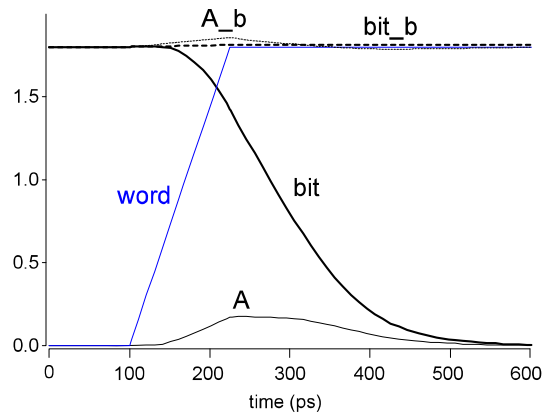
Ex: A = 0, A_b = 1

- bit discharges, bit_b stays high
- But A bumps up slightly

Read stability

- A must not flip
- N1 >> N2





Drive one bitline high, the other low

Then turn on wordline

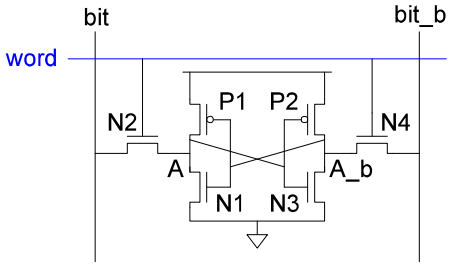
Bitlines overpower cell with new value

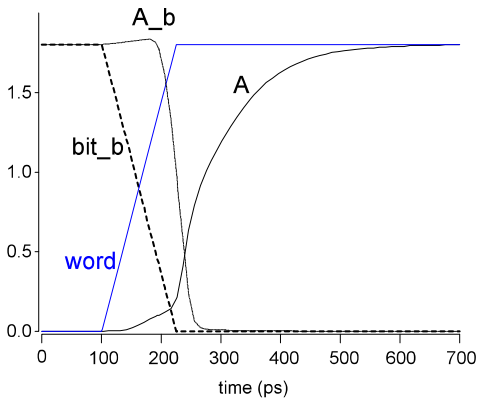
Ex: A = 0, A_b = 1, bit = 1, bit_b = 0

- Force A_b low, then A rises high

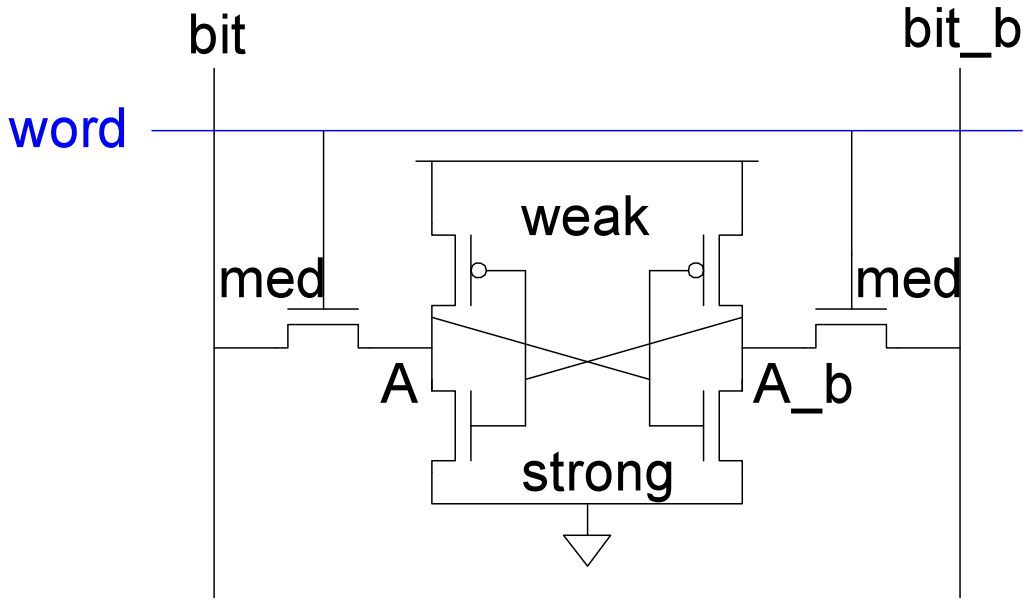
Writability

- Must overpower feedback inverter
- N2 >> P1



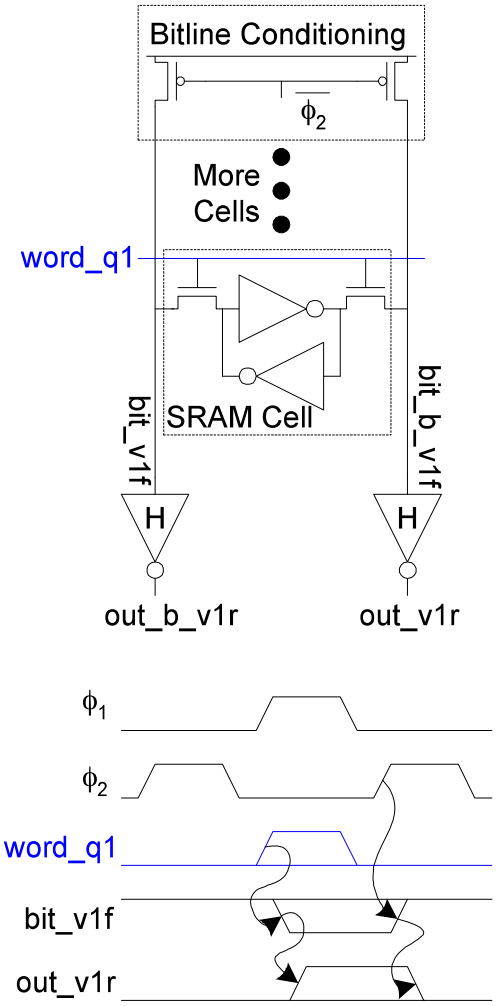


- High bitlines must not overpower inverters during reads
- But low bitlines must write new value into cell

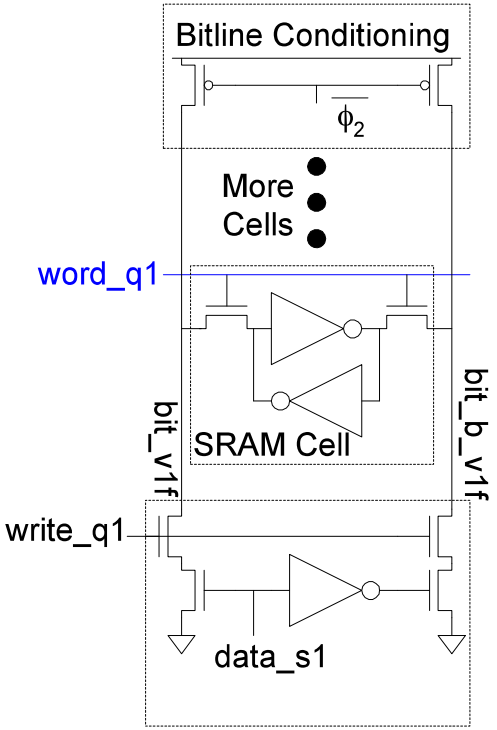


Standard Memory Design

Read

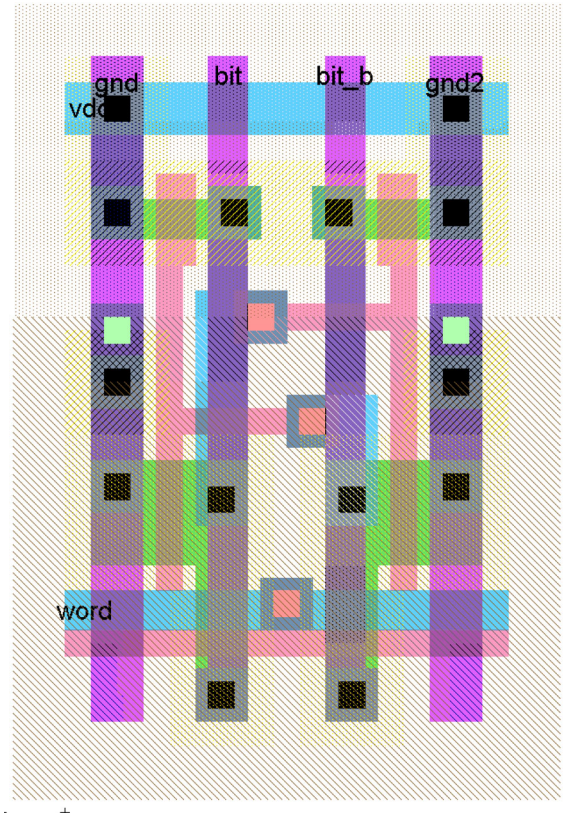


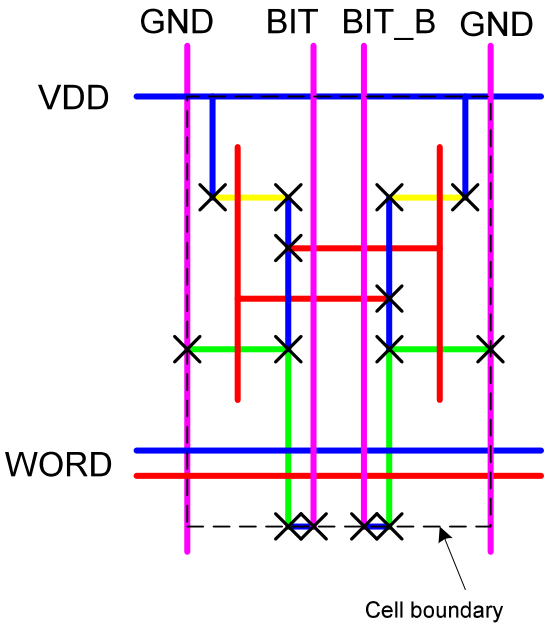
Write

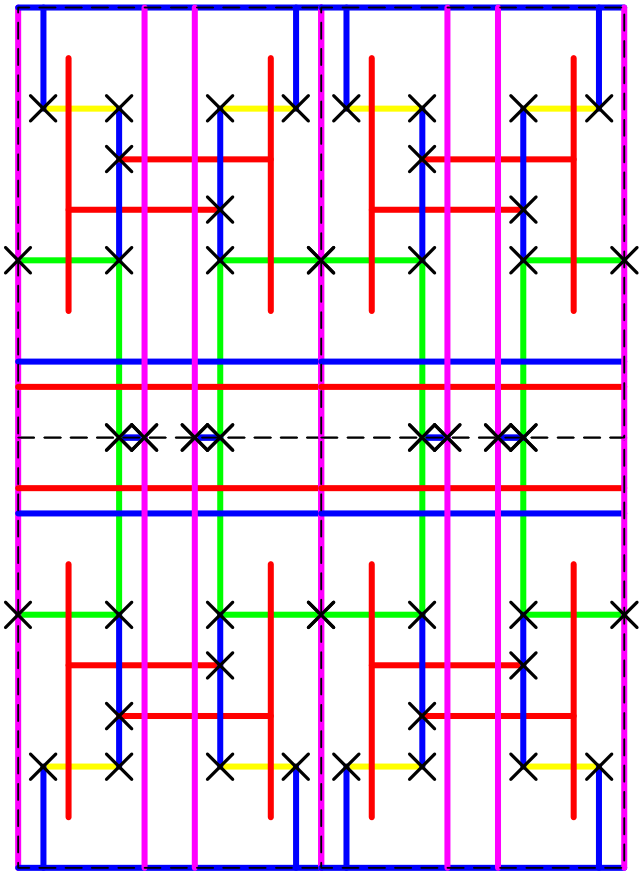


Standard Memory Design

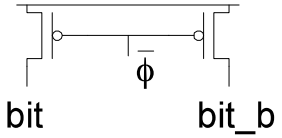
- Cell size is critical: 26 x 45 λ (even smaller in industry)
- Tile cells sharing V_{DD}, GND, bitline contacts



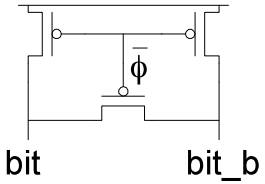




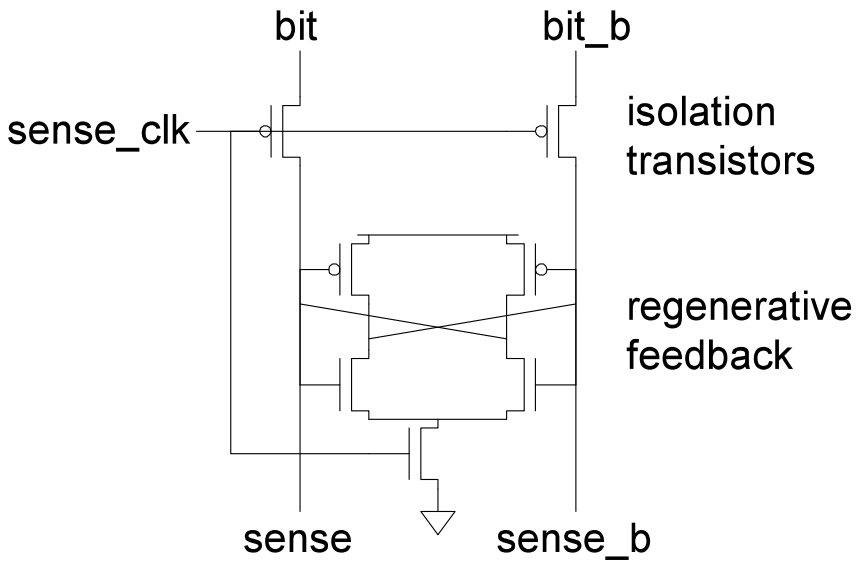
- Precharge bitlines high before reads



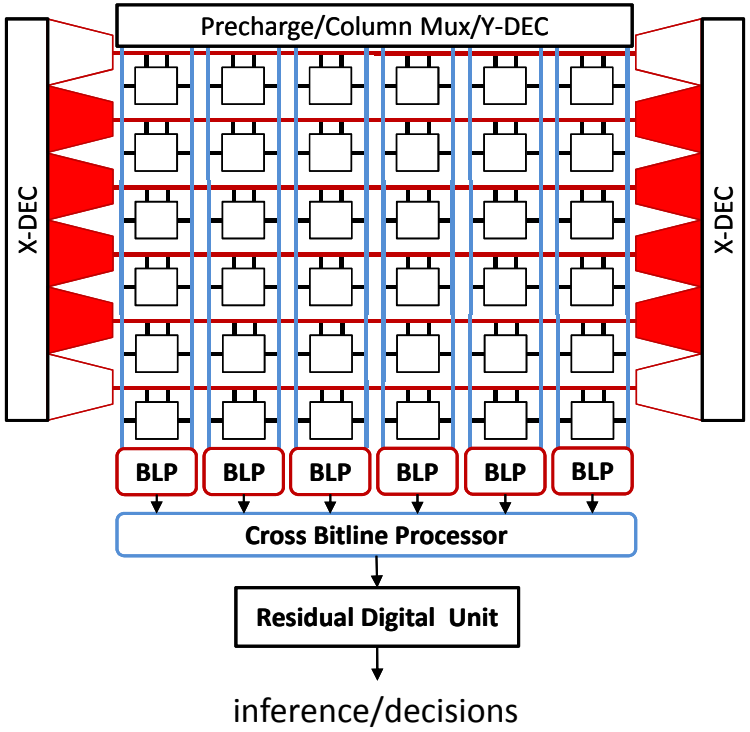
- Equalize bitlines to minimize voltage difference when using sense amplifiers



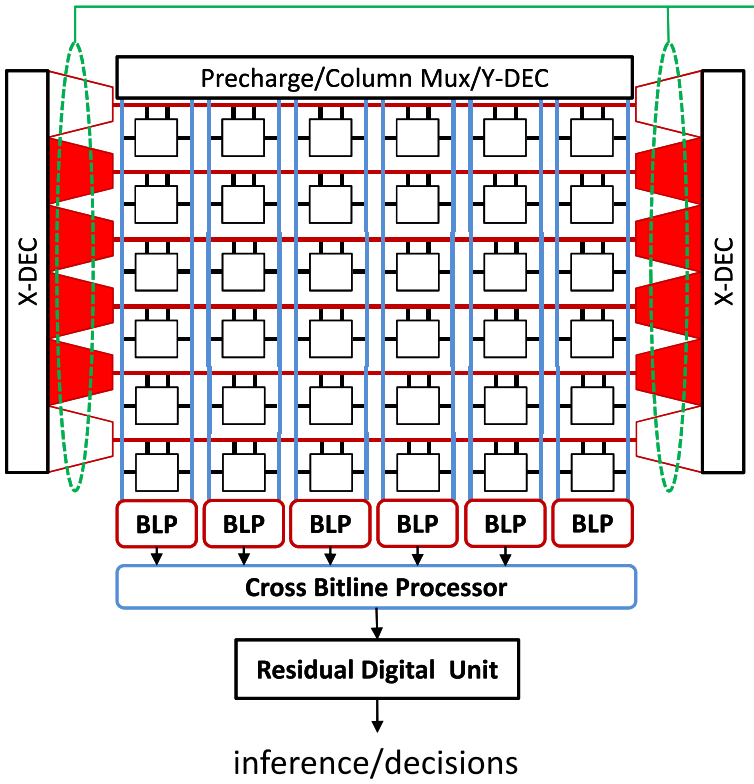
- Clocked sense amp saves power
- Requires sense_clk after enough bitline swing
- Isolation transistors cut off large bitline capacitance



Compute-in-Memory Design



Compute-in-Memory Design



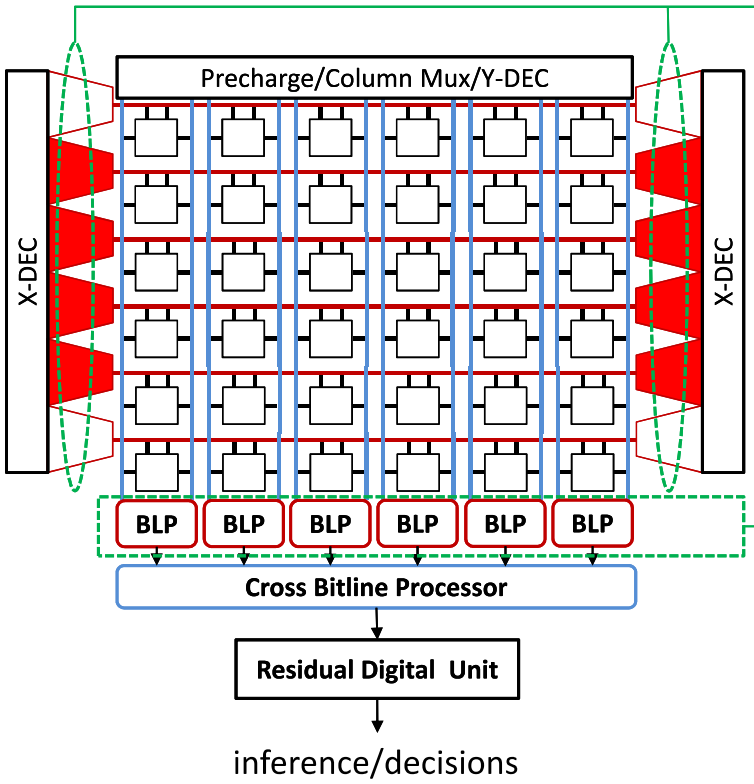
multi-row functional READ
(reads multiple-bits/row/precharge)

analog, mixed-signal
low-SNR processing

①

Compute-in-Memory Design

p y ( )

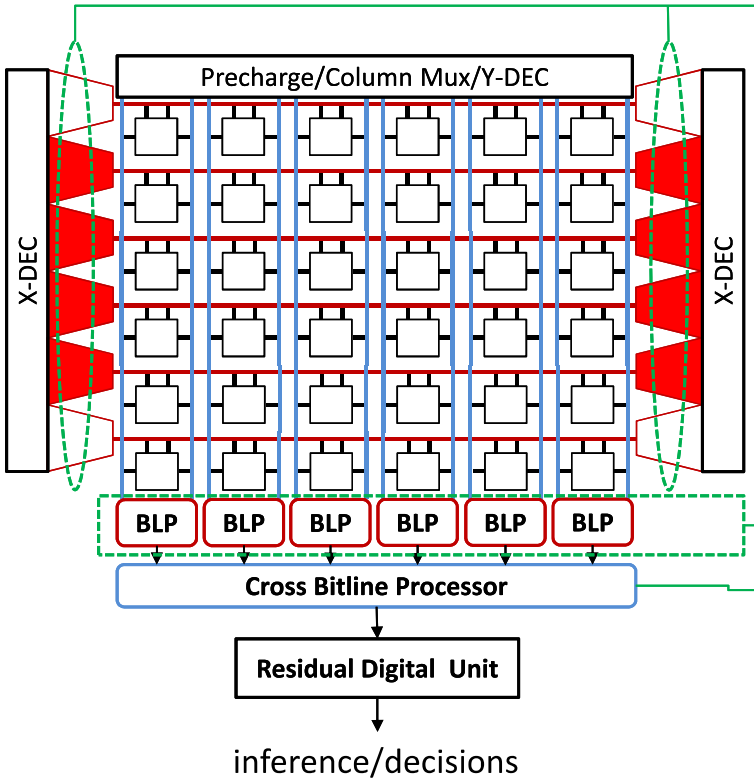


multi-row functional READ
(reads multiple-bits/row/precharge) ①

analog, mixed-signal
low-SNR processing

bitline processing (BLP)
(SIMD analog processing) ②

Compute-in-Memory Design



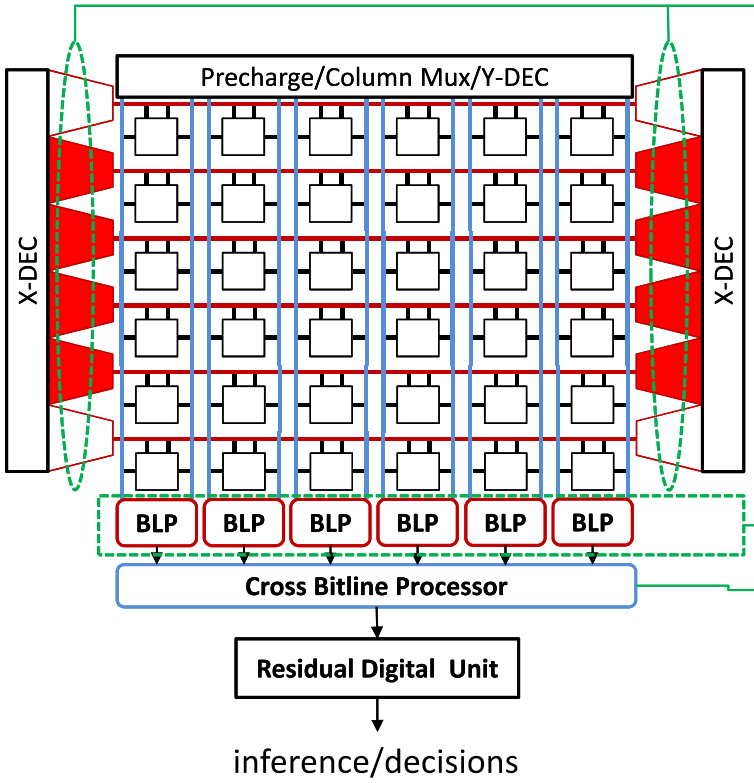
multi-row functional READ
(reads multiple-bits/row/precharge) ①

analog, mixed-signal
low-SNR processing

bitline processing (BLP)
(SIMD analog processing) ②

cross bitline processing (CBLP)
(analog averaging enhances SNR) ③

Compute-in-Memory Design



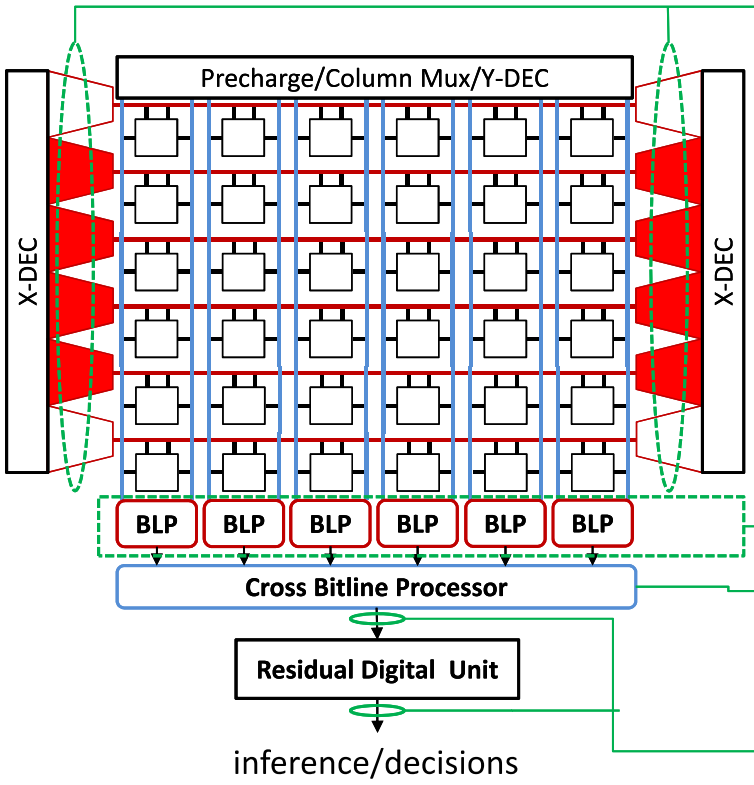
multi-row functional READ
(reads multiple-bits/row/precharge) ①

analog, mixed-signal
low-SNR processing

bitline processing (BLP)
(SIMD analog processing) ②

cross bitline processing (CBLP)
(analog averaging enhances SNR) ③

Compute-in-Memory Design



multi-row functional READ
(reads multiple-bits/row/precharge) ①

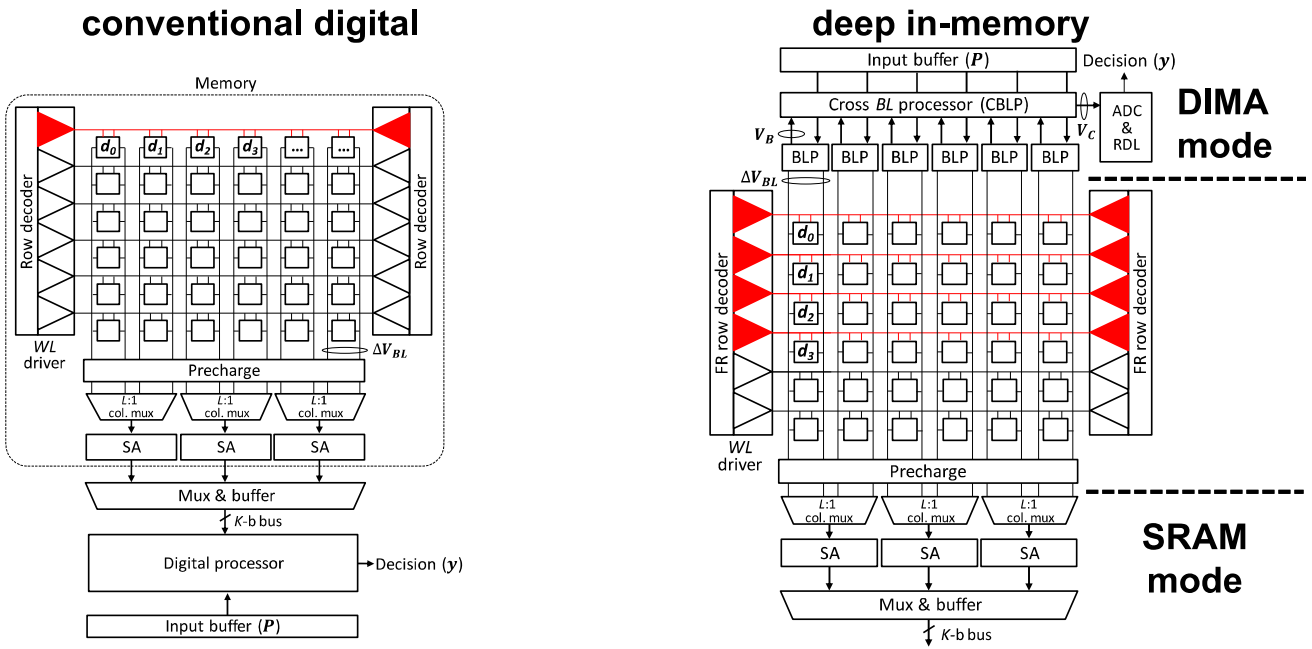
analog, mixed-signal low-SNR processing

bitline processing (BLP)
(SIMD analog processing) ②

cross bitline processing (CBLP)
(analog averaging enhances SNR) ③

low complexity, low (decision) rate digital output ④

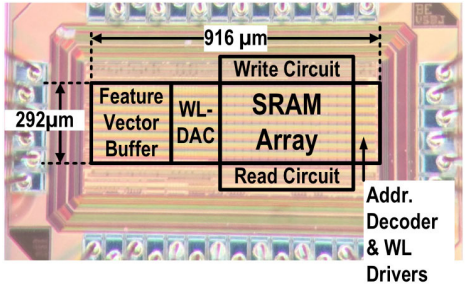
Compute-in-Memory Benefits?



attribute	conventional	deep in-memory	benefits
words read/access	$N_{col}/(LB_W)$	N_{col}	reduces # of memory accesses
BL swing/bit (mV)	$250-300$	$20-150$	reduces memory access energy
# of rows/access	1	B	enhances throughput

Active Area of Research!

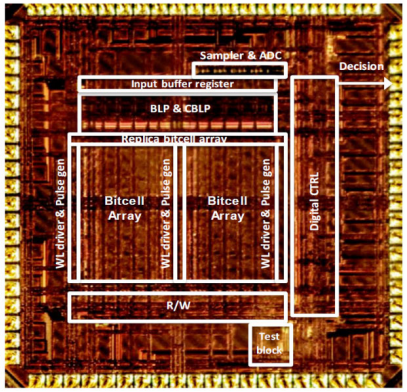
Adaboost



Adaboost;
MNIST;
energy savings= 13X;
EDP reduction= 175X;

[JSSC 2017, Princeton]

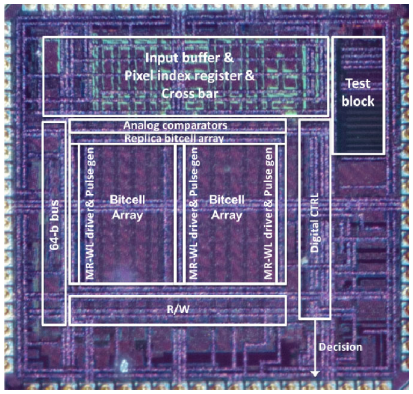
multi-functional



SVM, TM, k-NN, MF;
MIT-CBCL, MNIST, ..;
energy savings = 10X;
EDP reduction = 50X;

[JSSC 2018, UIUC]

random forest IC



RF with 64 trees;
KUL traffic sign;
energy savings = 3X
EDP reduction = 7X

[ESSCIRC 2017, UIUC]