

6.5930/1

Hardware Architectures for Deep Learning

# **Kernel Computation - Impact of Memory Hierarchy**

February 21, 2024

Joel Emer and Vivienne Sze

Massachusetts Institute of Technology  
Electrical Engineering & Computer Science



# Goals of Today's Lecture

---

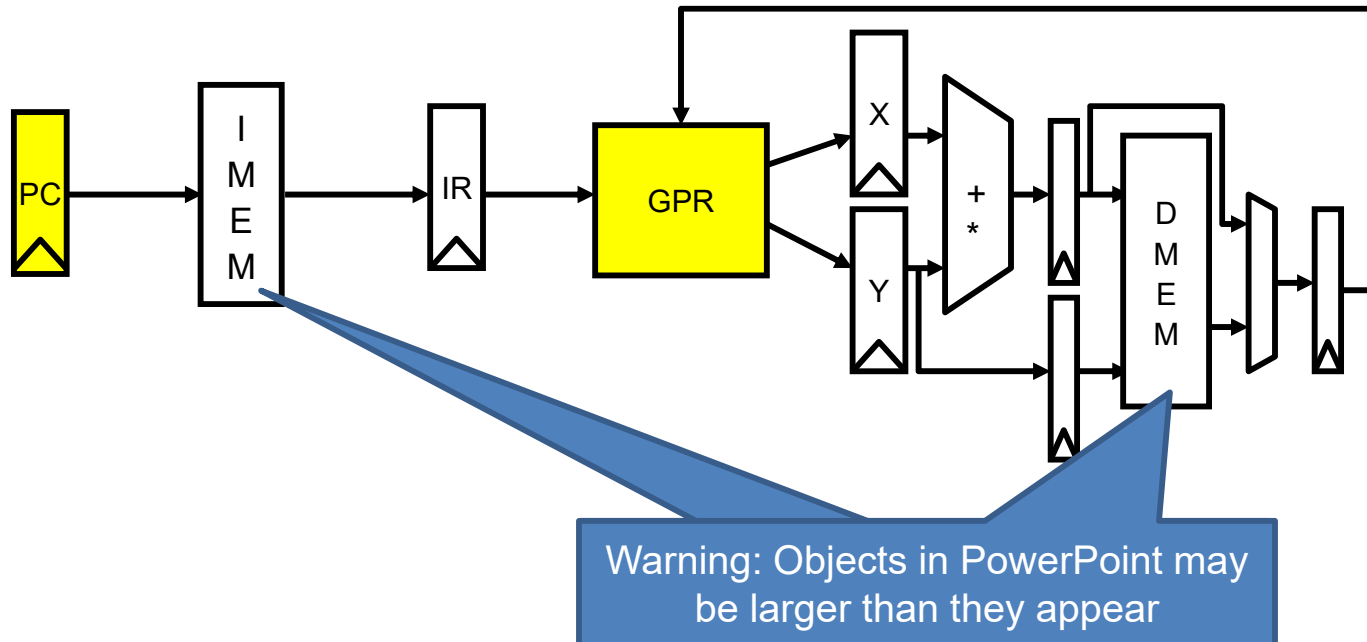
- Understand impact of memory hierarchy
  - Overview of caches
  - Structuring algorithms to work well in caches using tiling
  - Storage technologies

# Readings for this Week

---

- Efficient Processing of Deep Neural Networks
  - Chapter 4 of <https://doi.org/10.1007/978-3-031-01766-7>

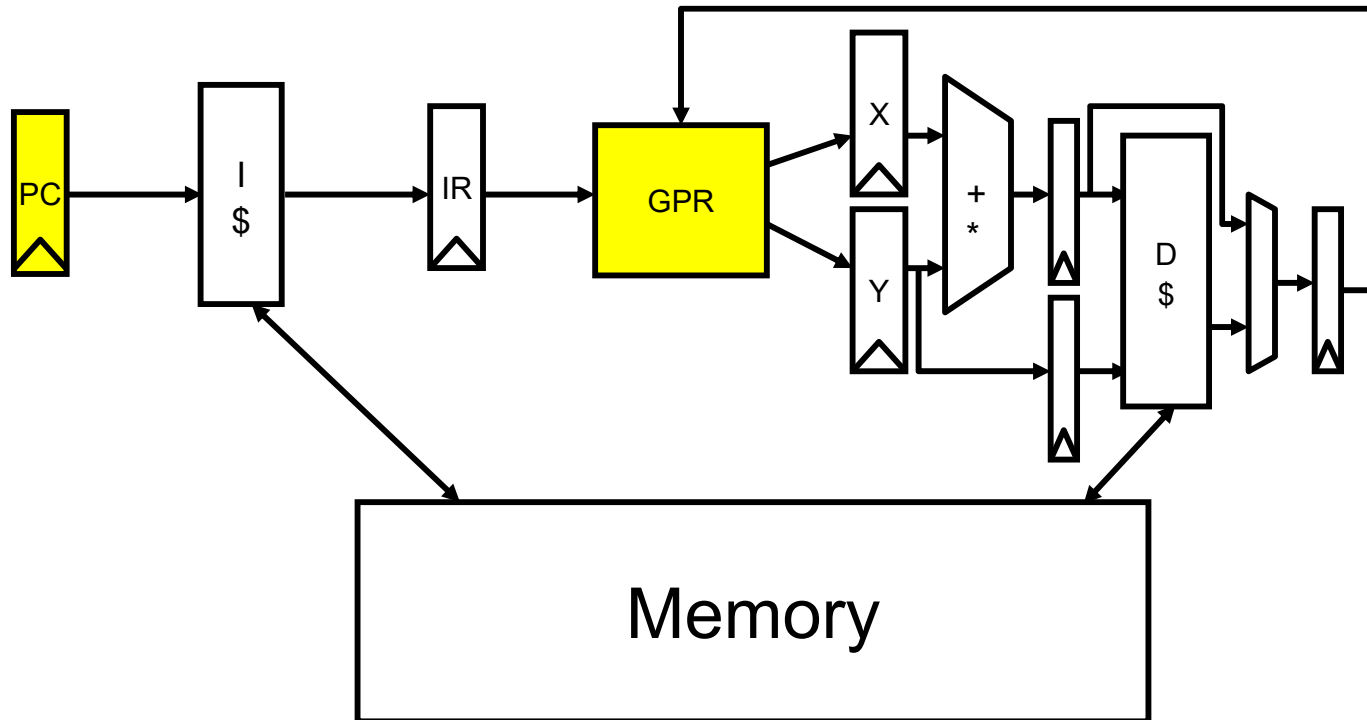
# Simple Pipelined $\mu$ Architecture



What are consequences of putting large memory (e.g., megabytes) directly in pipeline?

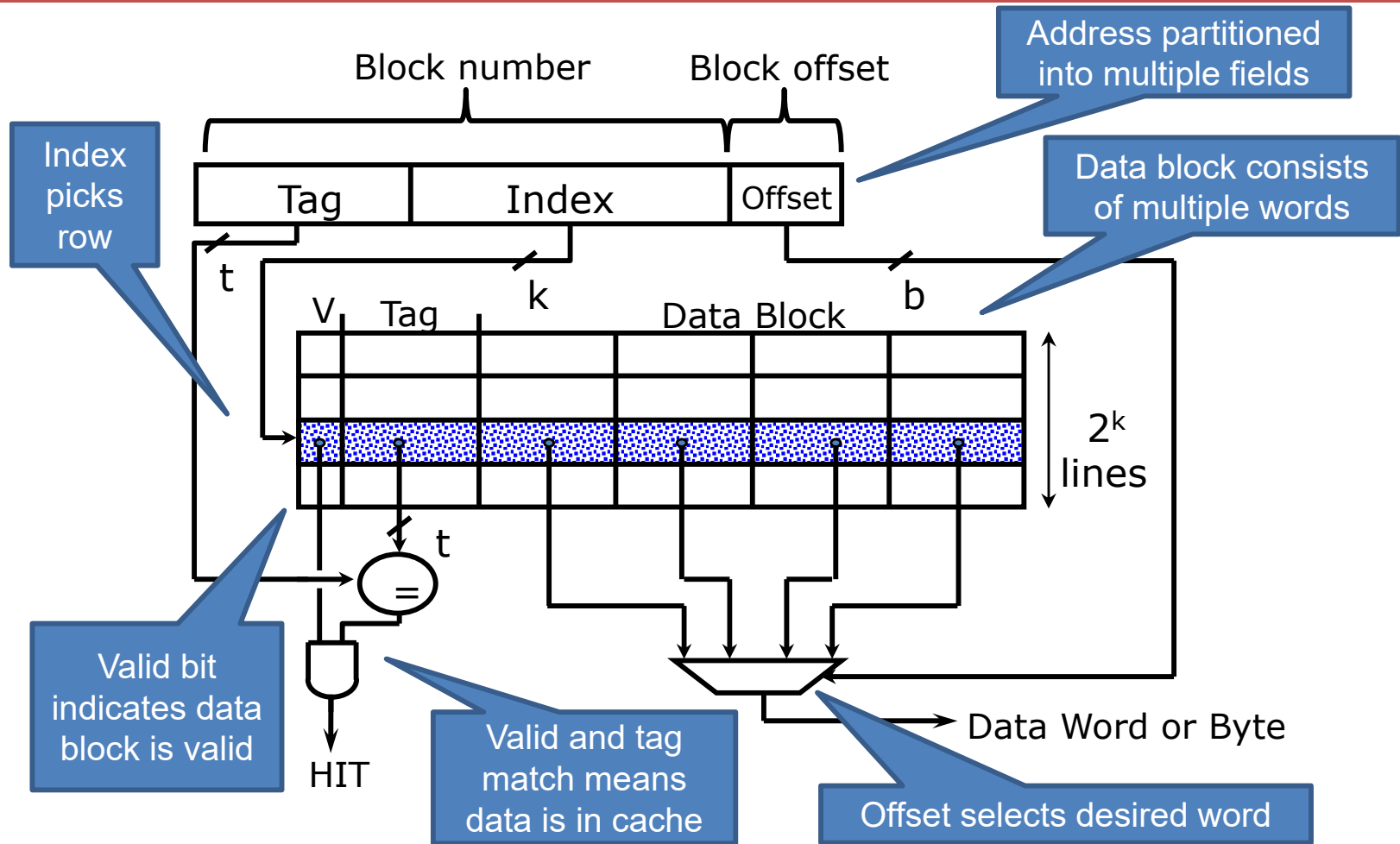
Long latency => dependency stalls  
Large energy consumption

# Pipelined $\mu$ Architecture with Caches



Instruction cache (I\$) and data cache (D\$) hold memory data for reuse in small energy efficient buffer

# Direct Mapped Cache



# Cache Operation

Look at data address, search cache tags to find match.  
Then if...

Found in cache  
a.k.a. HIT

Not in cache  
a.k.a. MISS

Return copy  
of data from  
cache

Read block of data from  
Main Memory

Wait ...

Return data to processor  
and update cache

Fill

Metric: Hit Rate =  $\#Hits / (\#Hits + \#Misses)$



# Treatment of Writes

---

- Cache hit:
  - *write through*: write both cache & memory
    - generally higher traffic but simplifies cache in processor pipeline
  - *write back*: write cache only  
(memory is written only when the entry is evicted)
    - a dirty bit per block can further reduce the traffic
- Cache miss:
  - *no write allocate*: only write to main memory
  - *write allocate (aka fetch on write)*: fetch into cache
- Common combinations:
  - write through and no write allocate
  - write back with write allocate



# Cache Locality

---

Caches **implicitly** try to optimize data movement by trying to exploit two common properties of memory references:

- *Spatial Locality*: If a location is referenced it is likely that locations near it will be referenced in the near future.
  - Exploited by having block size larger than a word, which also amortizes fill overheads by getting more bytes with one access
- *Temporal Locality*: If a location is referenced it is likely to be referenced again in the near future.
  - Exploited by holding blocks for future access

# Fully Connected (FC) Computation

Predefined constant

```
int i[CHW];    # Input activations
int f[M*CHW]; # Filter Weights
int o[M];     # Output activations
```

```
CHWm = -CHW
```

```
for m in [0, M):
```

```
    o[m] = 0
```

```
    CHWm += CHW
```

```
    for chw in [0, CHW):
```

```
        o[m] += i[chw] * f[CHWm + chw]
```

M iterations

C\*H\*W iterations

M\*C\*H\*W loads of each weight and input activation

# Impact of spatial locality

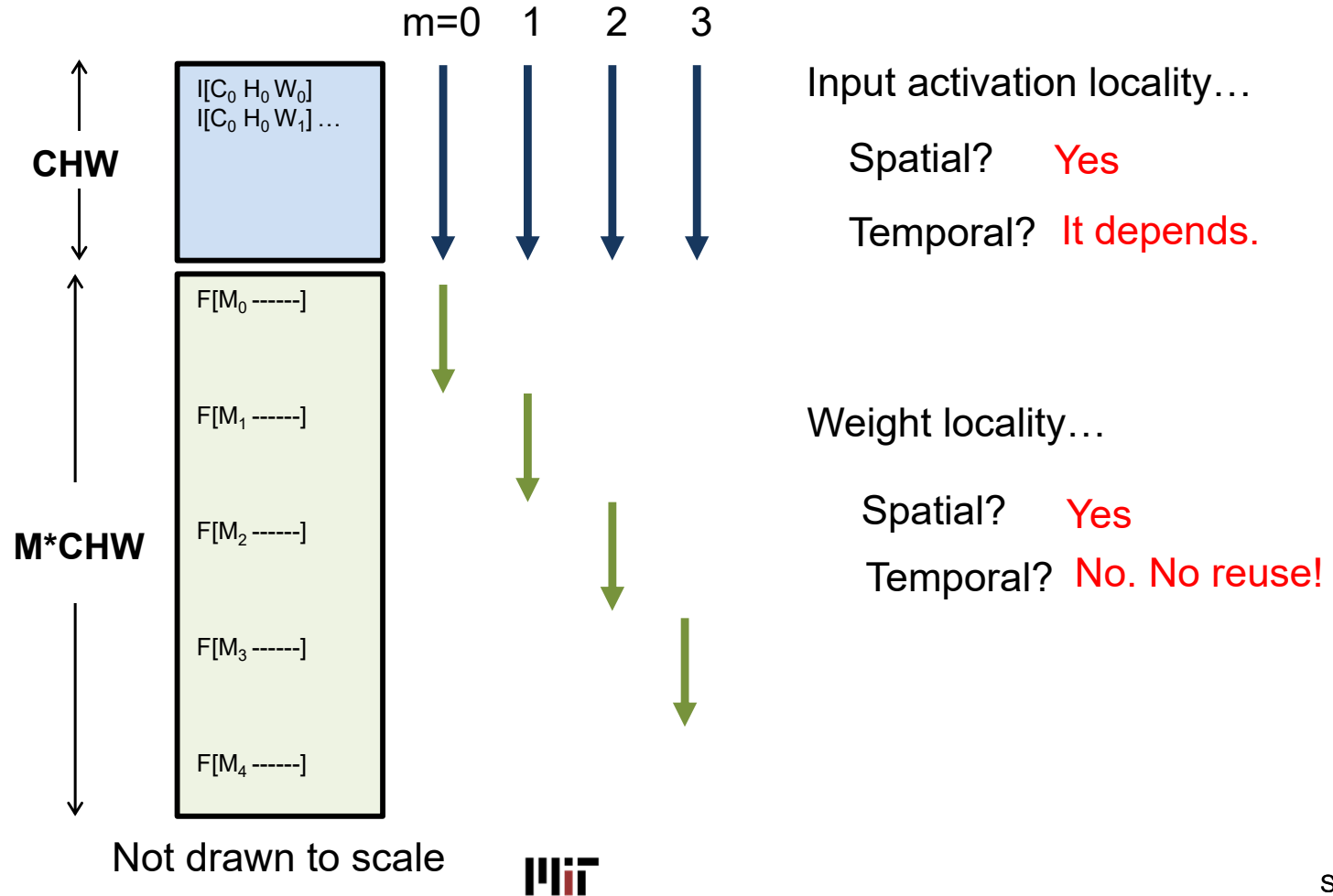
---

- Typical in-pipeline cache size
  - 64K bytes => 16K FP32 words
  - 64 byte blocks => 16 FP32 words/block

Hit rate of long sequential reference streams due to spatial locality?

15/16 => ~94%

# FC – Data Reference Pattern



# FC – Data Reference Pattern

Tensor: `f_MCHW[M, CHW]`

Rank: CHW

	0	1	2	3	4	5	6	7
Rank: M	9	4	7	2	4	5	4	3
1	2	7	3	9	8	4	3	3
2	2	5	8	7	9	2	5	7
3	7	2	2	1	7	6	2	3

Tensor: `i_CHW[CHW]`

Rank: CHW

0	1	2	3	4	5	6	7
3	4	7	5	1	6	2	5

Tensor: `unknown[M]`

Rank: M

0	1	2	3
0	0	0	0

# Amount of temporal locality

---

- Typical in-pipeline cache size
  - 64K bytes => 16K FP32 words
  - 64 byte blocks => 16 FP32 words/block
- Typical layer size:
  - H, W = 256   C = 128

Size of input activations?

$256 \times 256 \times 128 \times 4 \Rightarrow 32\text{MB}$

What does this imply for  
Input activation temporary locality?

No temporal locality  
since  $32\text{MB} > 64\text{K bytes}$

# Computational Intensity

---

$$\text{Computational Intensity} = \frac{MACS}{Data\ Words}$$

$$O_m = I_{chw} \times F_{m,chw}$$

Number MACS:  $M*CHW = M*C*H*W$

# Computational Intensity – Ideal FC

$$\text{Computational Intensity} = \frac{\text{MACS}}{\text{Data Words}}$$

$$O_m = I_{chw} \times F_{m,chw}$$

Number MACS:	$M \times C \times H \times W$	Filter weight accesses:	$M \times CHW = M \times C \times H \times W$
		Input activation accesses:	$CHW = C \times H \times W$
		Output activation accesses:	$M$

$$\text{Computational Intensity} = \frac{M \times C \times H \times W}{M \times C \times H \times W + C \times H \times W + M} = \frac{1}{1 + \frac{1}{M} + \frac{1}{C \times H \times W}} \sim 1$$

If in one cycle a processor can deliver one word from memory and perform one MAC how well will the machine perform running this code?



# Computational Intensity – Naïve FC

$$\text{Computational Intensity} = \frac{\text{MACS}}{\text{Data Words}}$$

```

CHWm = -CHW;
for m in [0, M):
  o[m] = 0;
  CHWm += CHW
  for chw in [0, CHW):
    o[m] += i[chw] * f[CHWm + chw]
  
```

Number MACS:	$M \times C \times H \times W$	Filter weight accesses	$M \times C \times H \times W$
		Input activation accesses	$M \times C \times H \times W$
		Output activation accesses	$M$

$$\text{Computational Intensity} = \frac{M \times C \times H \times W}{M \times C \times H \times W + M \times C \times H \times W + M} = \frac{1}{2 + \frac{1}{C \times H \times W}} \sim \frac{1}{2}$$

## Einsum for strip mined FC

---

$$O_m = I_{chw} \times F_{m,chw}$$

$$I_{\underbrace{chw/T}_{chw1}, \underbrace{chw\%T}_{chw0}} = I_{chw} \quad F_{m, \underbrace{chw/T}_{chw1}, \underbrace{chw\%T}_{chw0}} = F_{m,chw}$$

$$O_m = I_{chw1,chw0} \times F_{m,chw1,chw0}$$

# Fully Connected – Strip Mined

```
for m in [0, M):
  for chw in [0, C*H*W):
    o[m] += i[chw] * f[CHW*m + chw]
```

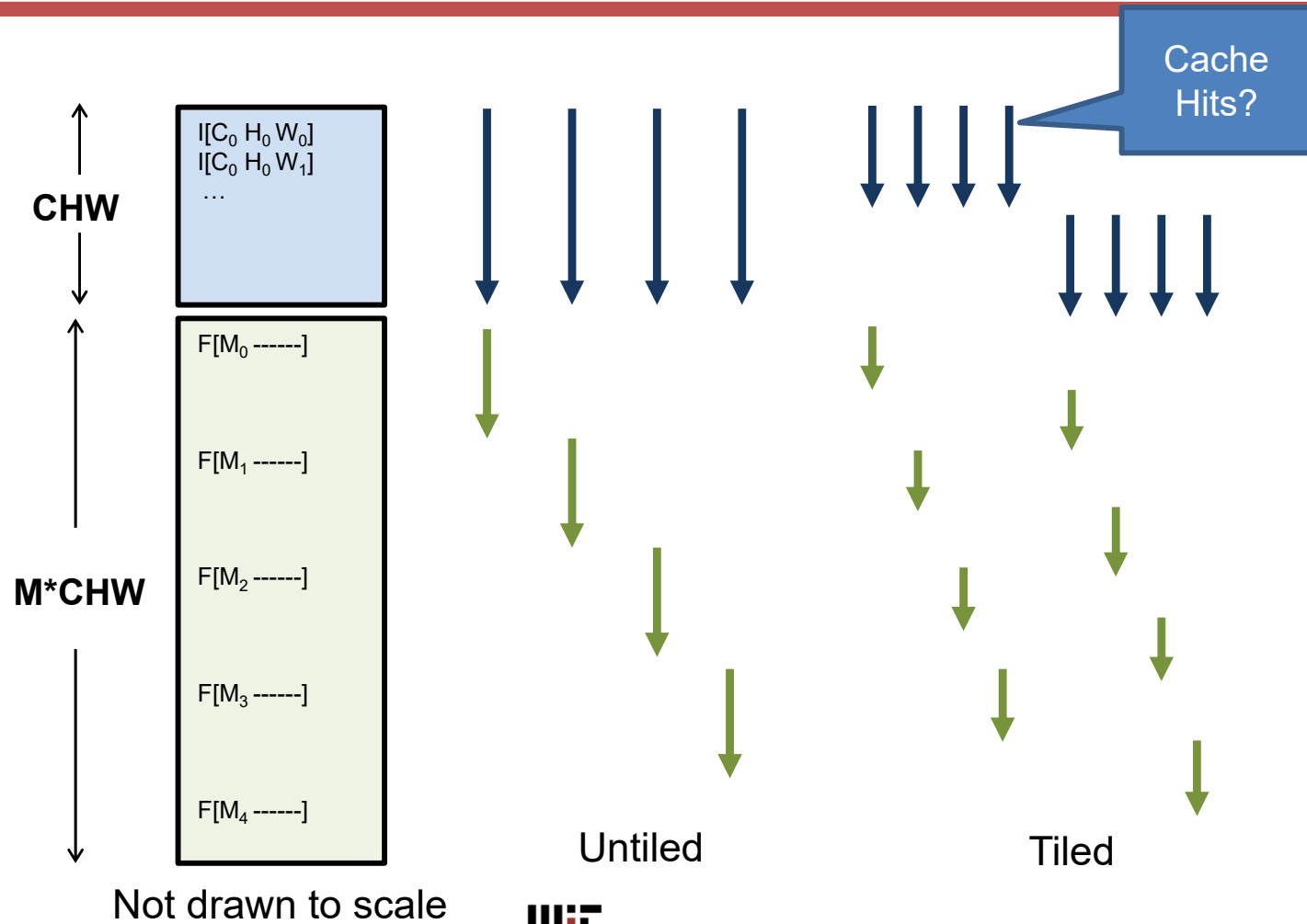
```
// Level 1
for chw1 in [0, CHW1):
  for m in [0, M):
    // Level 0
    for chw0 in [0, CHW0):
      chw = CHW0*chw1+chw0
      o[m] += i[chw] * f[CHW*m + chw]
```

CHW1\*CHW0 =  
C\*H\*W

Inner loop working  
set = CHW0

Just considering input activations, **Less than cache size**  
what value should CHW0 be?

# FC - Strip Mined Data Reference Pattern



# Computational Intensity – Strip Mined

```
// Level 1
for chw1 in [0, CHW1):
  for m in [0, M):
    // Level 0
    for chw0 in [0, CHW0):
      chw = CHW0*chw1+chw0
      o[m] += i[chw] * f[CHW*m + chw]
```

Number MACS:	$M \times C \times H \times W$	Filter weight accesses:	$M \times C \times H \times W$
		Input activation accesses:	$C \times H \times W$
		Output activation accesses:	$M$

$$\text{Computational Intensity} = \frac{M \times C \times H \times W}{M \times C \times H \times W + C \times H \times W + M} = \frac{1}{1 + \frac{1}{M} + \frac{1}{C \times H \times W}} \sim 1$$

# Matrix-Vector Multiply – Strip Mined

Tensor: f\_MCHW[M, CHW]

Rank: CHW

	0	1	2	3	4	5	6	7	
Rank: M	0	9	4	7	2	4	5	4	3
1	2	7	3	9	8	4	3	3	
2	2	5	8	7	9	2	5	7	
3	7	2	2	1	7	6	2	3	

Tensor: i\_CHW[CHW]

Rank: CHW

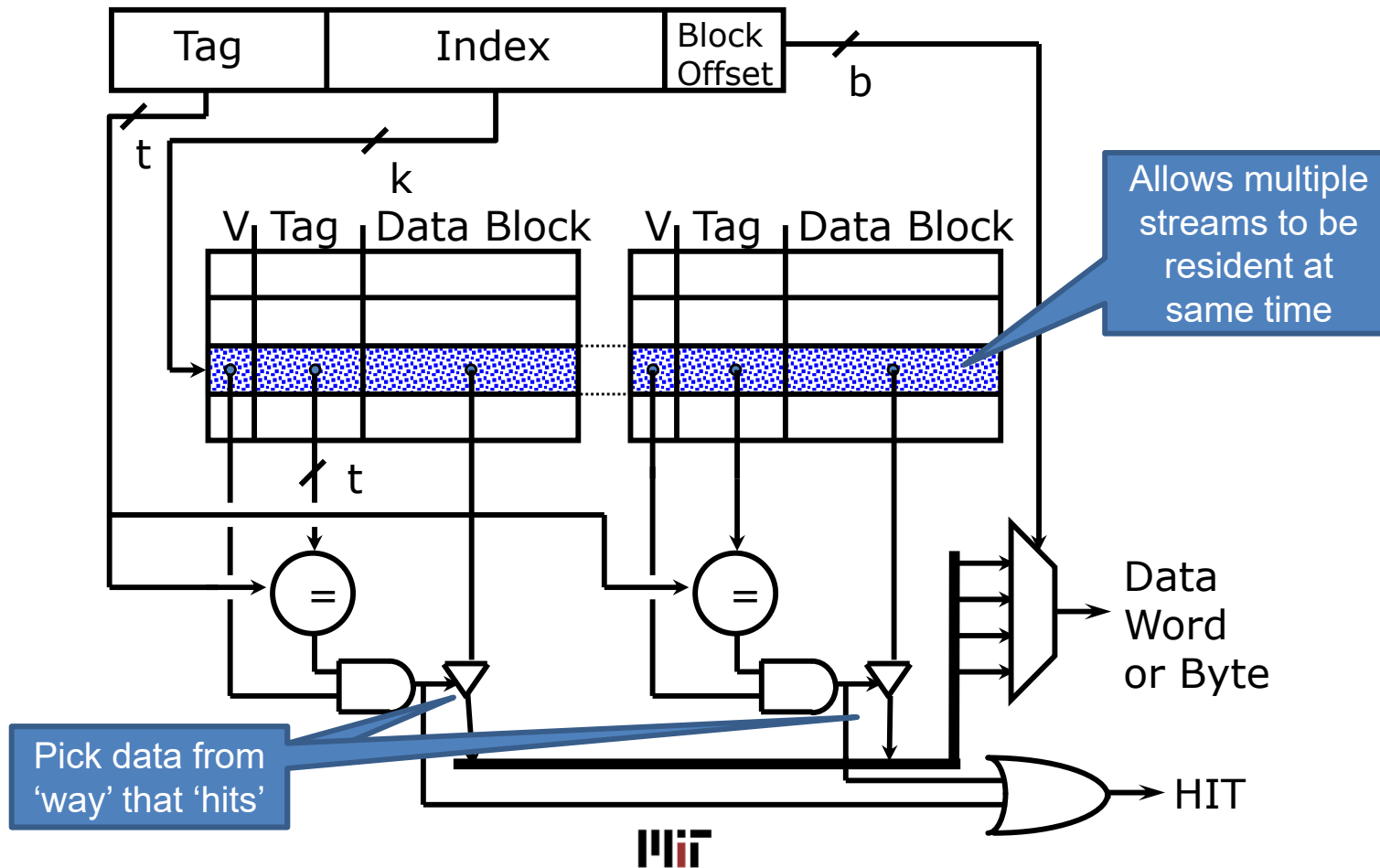
0	1	2	3	4	5	6	7
3	4	7	5	1	6	2	5

Tensor: unknown[M]

Rank: M

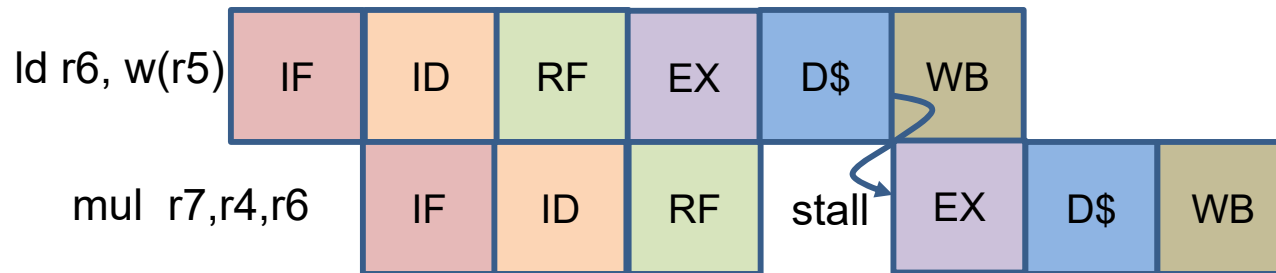
0	1	2	3
0	0	0	0

# Associative Cache

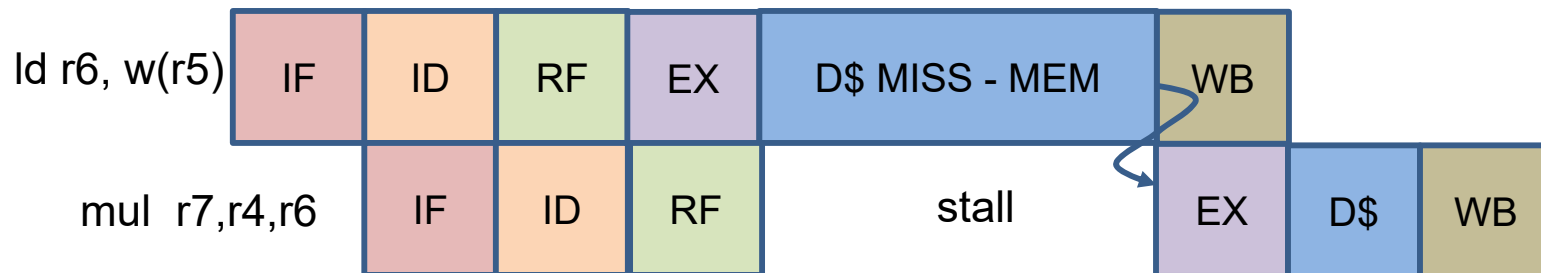


# Cache Miss Pipeline Diagram

HIT



MISS



Time (cycles)





# Avoiding Cache Miss Stalls

---

- Reorganize code so loads are far ahead of use
  - Requires huge amount of unrolling
  - Consumes lots of registers
- Add 'prefetch' instructions that just load cache
  - Consumes instruction issue slots
- Add hardware that automatically loads cache

# Hardware Data Prefetching

---

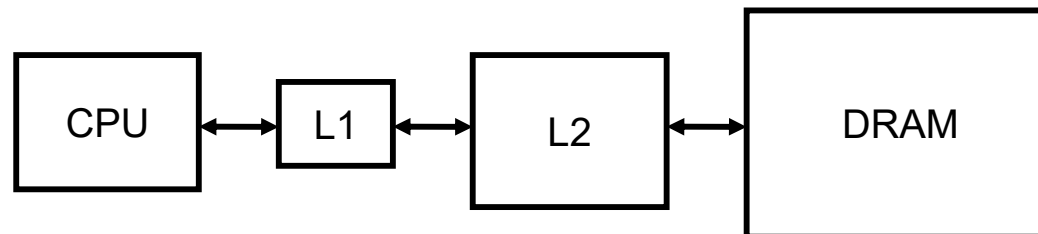
- Prefetch-on-miss:
  - Prefetch  $b + 1$  upon miss on  $b$
- One Block Lookahead (OBL) scheme
  - Initiate prefetch for block  $b + 1$  when block  $b$  is accessed
  - Can extend to  $N$  block lookahead
- Strided prefetch
  - If observe sequence of accesses to block  $b$ ,  $b+N$ ,  $b+2N$ , then prefetch  $b+3N$  etc.

Example: IBM Power 5 [2003] supports eight independent streams of strided prefetch per processor, prefetching 12 lines ahead of current access

# Multi-level Caches

---

- A memory cannot be large and fast
- Add level of cache to reduce miss penalty
  - Each level can have longer latency than level above
  - So, increase sizes of cache at each level



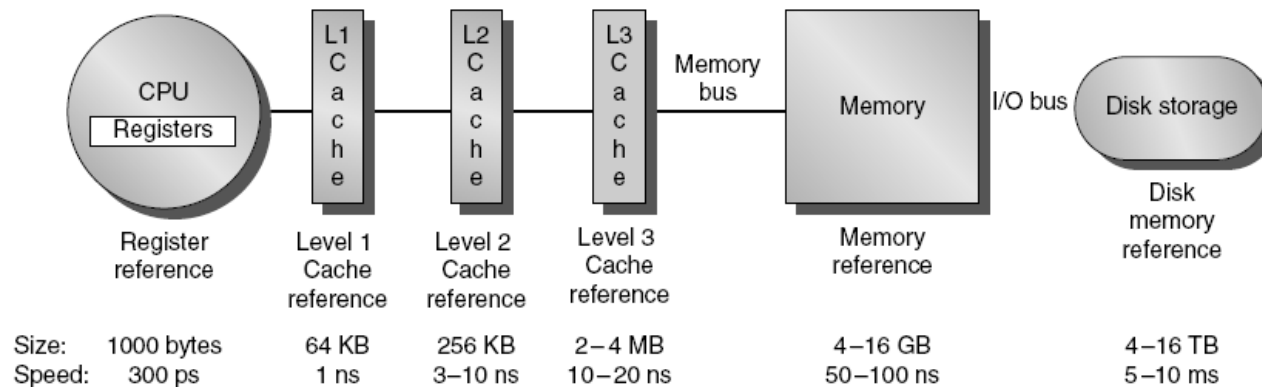
Metrics:

Local miss rate = misses in cache / accesses to cache

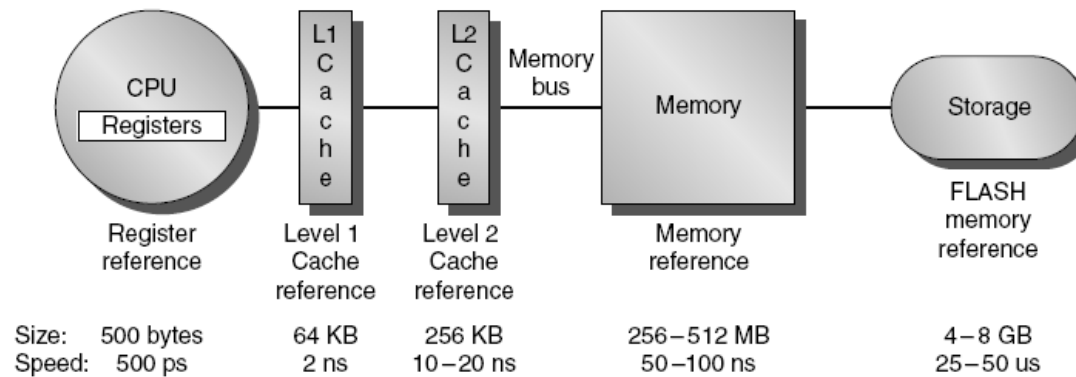
Global miss rate = misses in cache / CPU memory accesses

Misses per instruction = misses in cache / number of instructions

# Contemporary CPU Cache Hierarchy

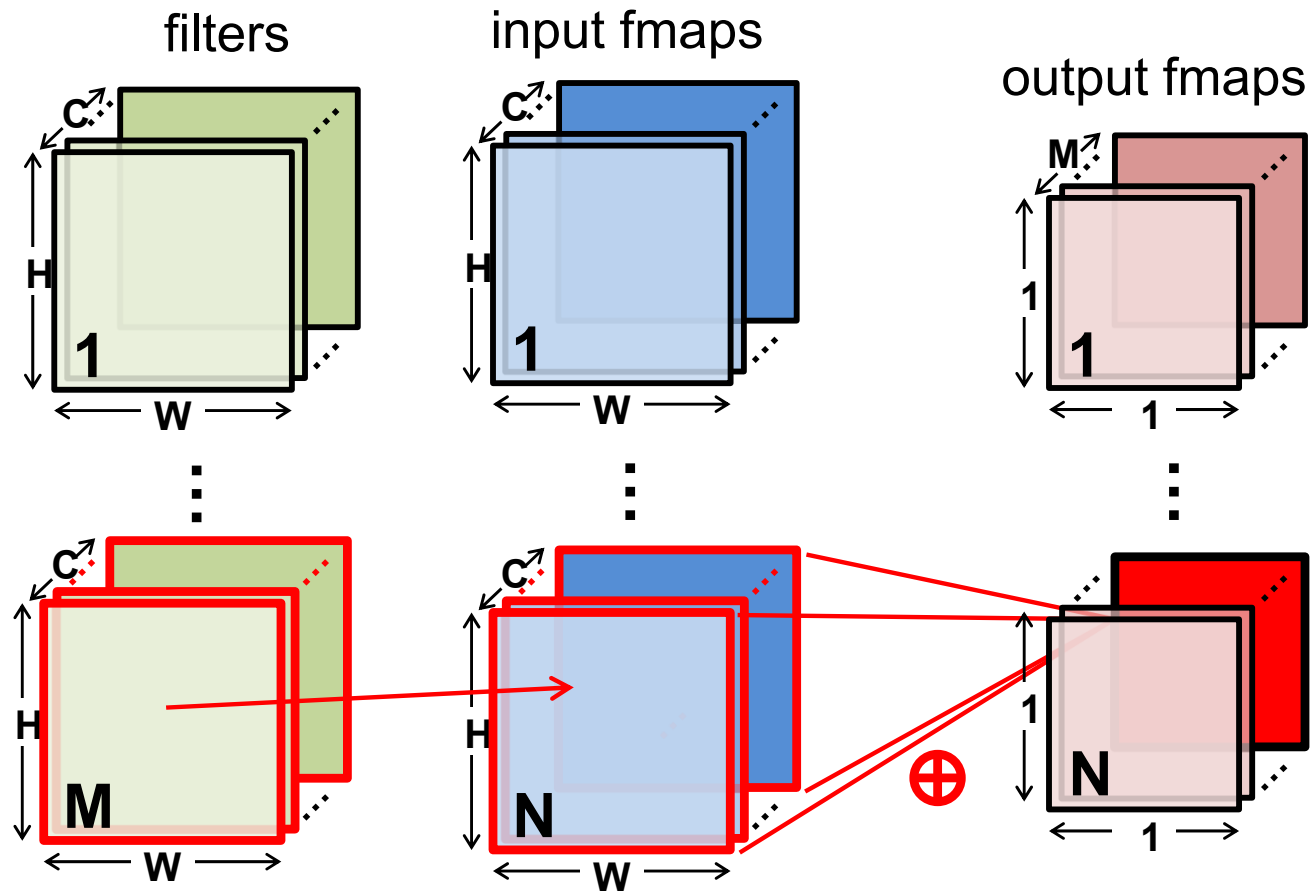


(a) Memory hierarchy for server

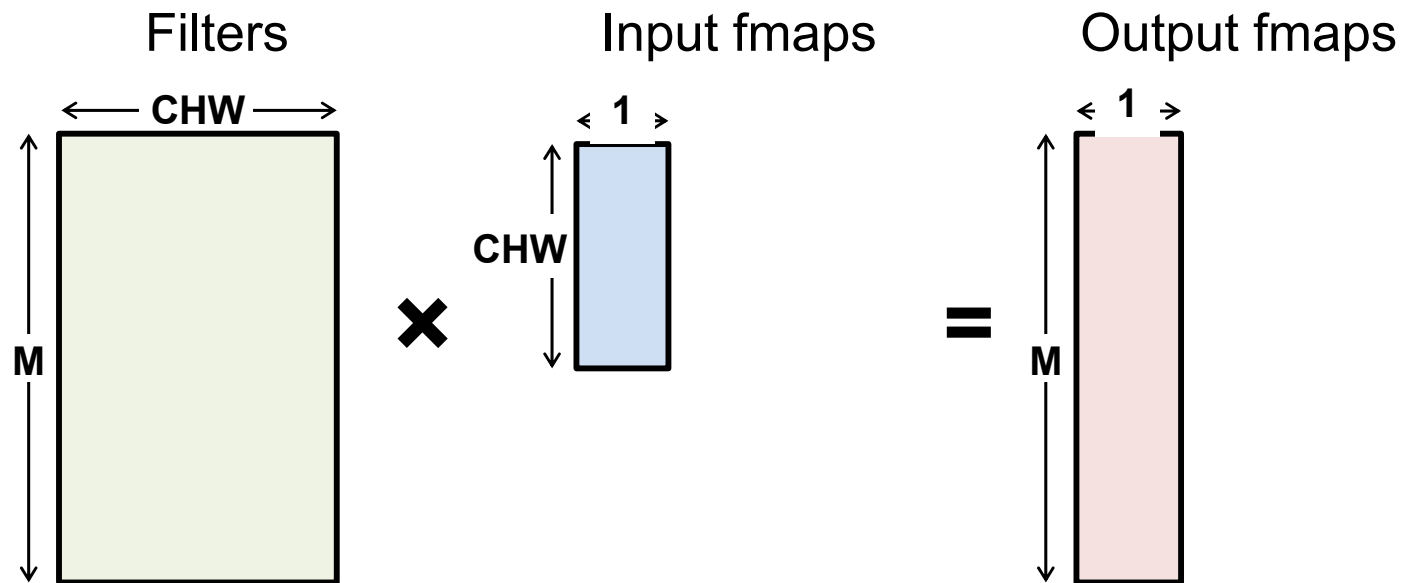


(b) Memory hierarchy for a personal mobile device

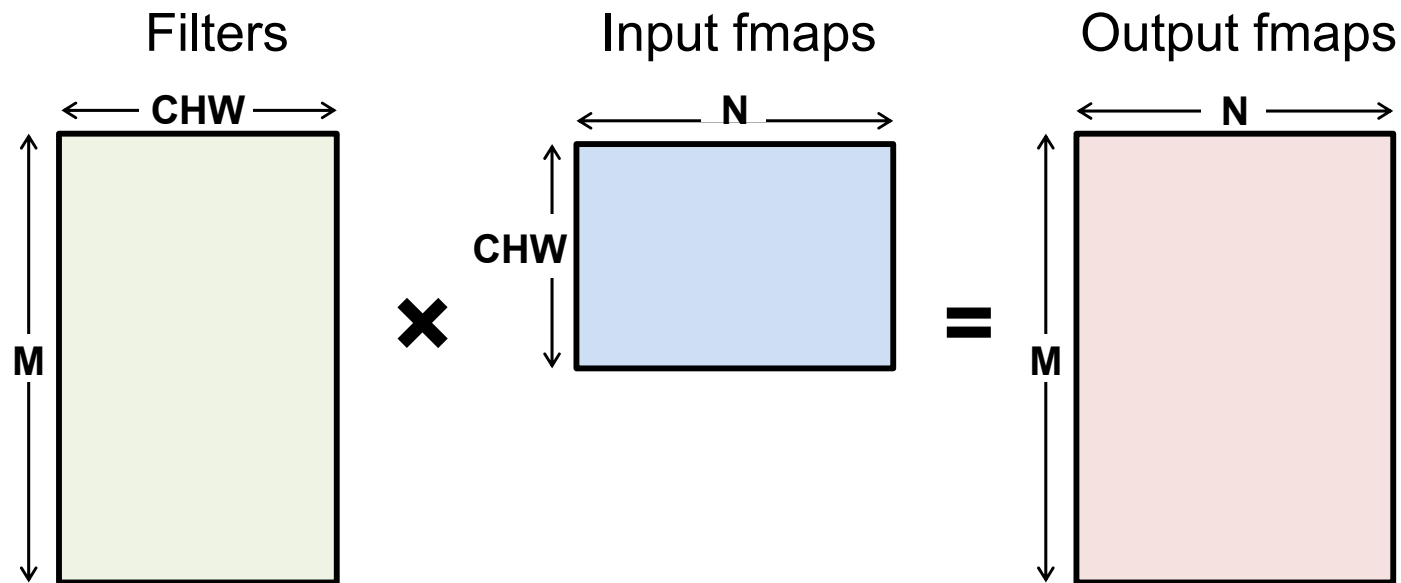
# FC Layer – Multichannel



# Fully-Connected (FC) Layer



# Fully-Connected (FC) Layer

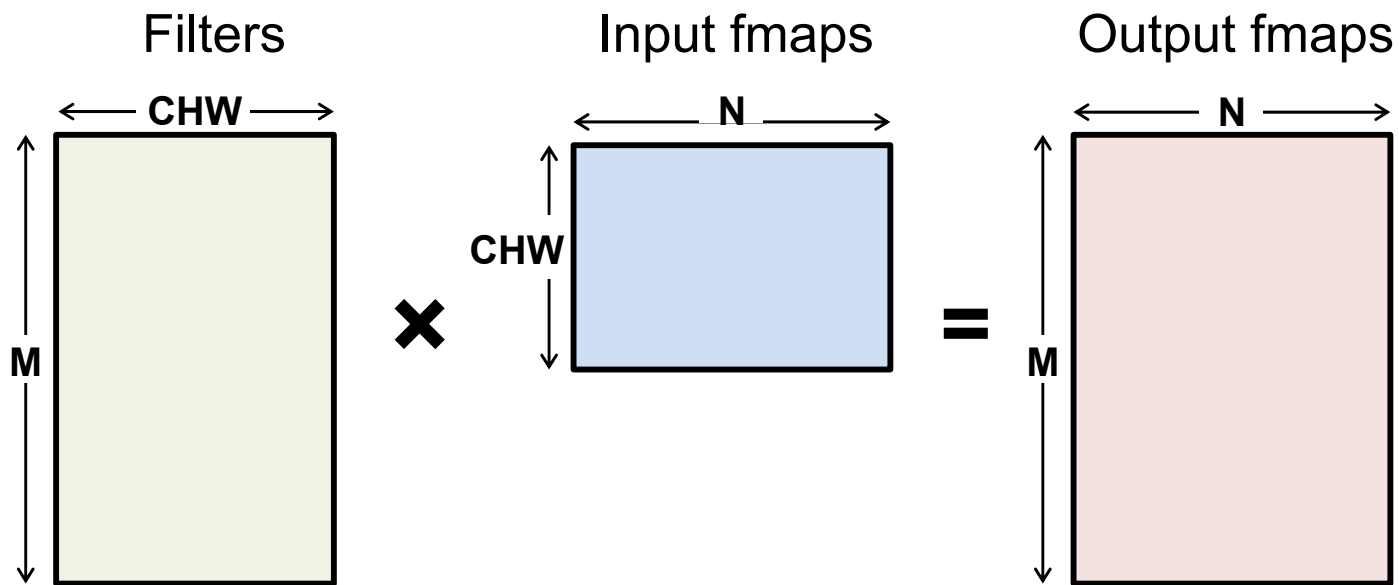


- After flattening, having a batch size of  $N$  turns the matrix-vector operation into a matrix-matrix multiply

# FC Einsum Notation

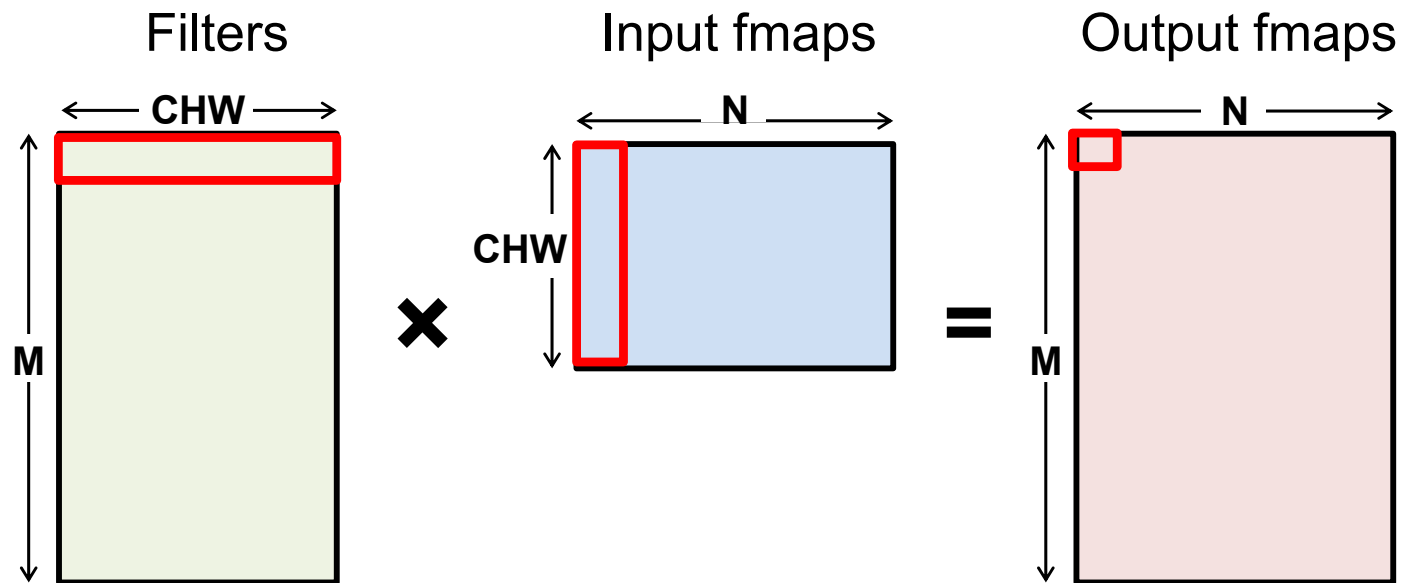
---

$$O_{n,m} = F_{m,chw} \times I_{n,chw}$$



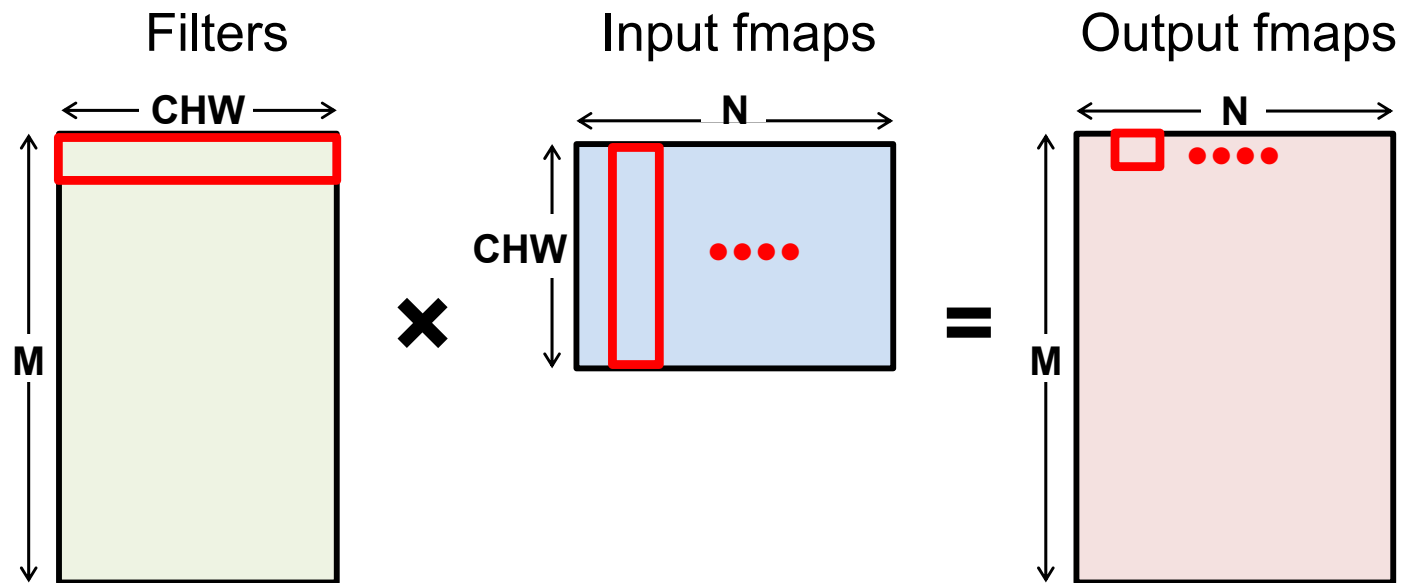


# Fully-Connected (FC) Layer



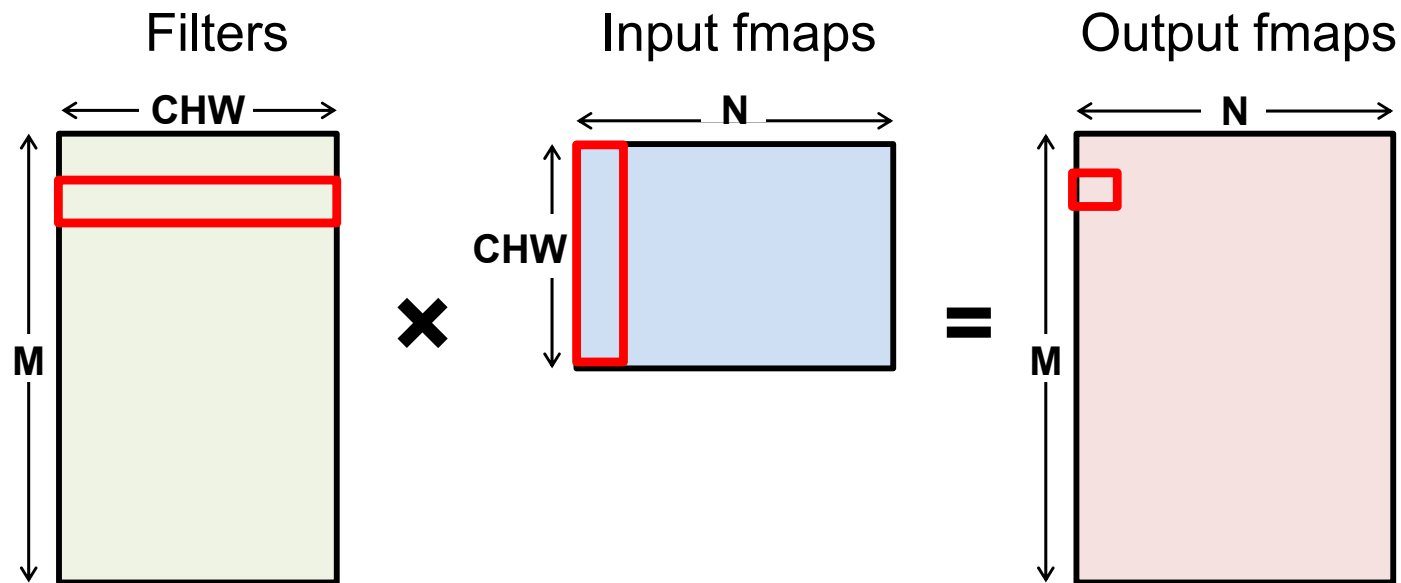
- After flattening, having a batch size of  $N$  turns the matrix-vector operation into a matrix-matrix multiply

# Fully-Connected (FC) Layer



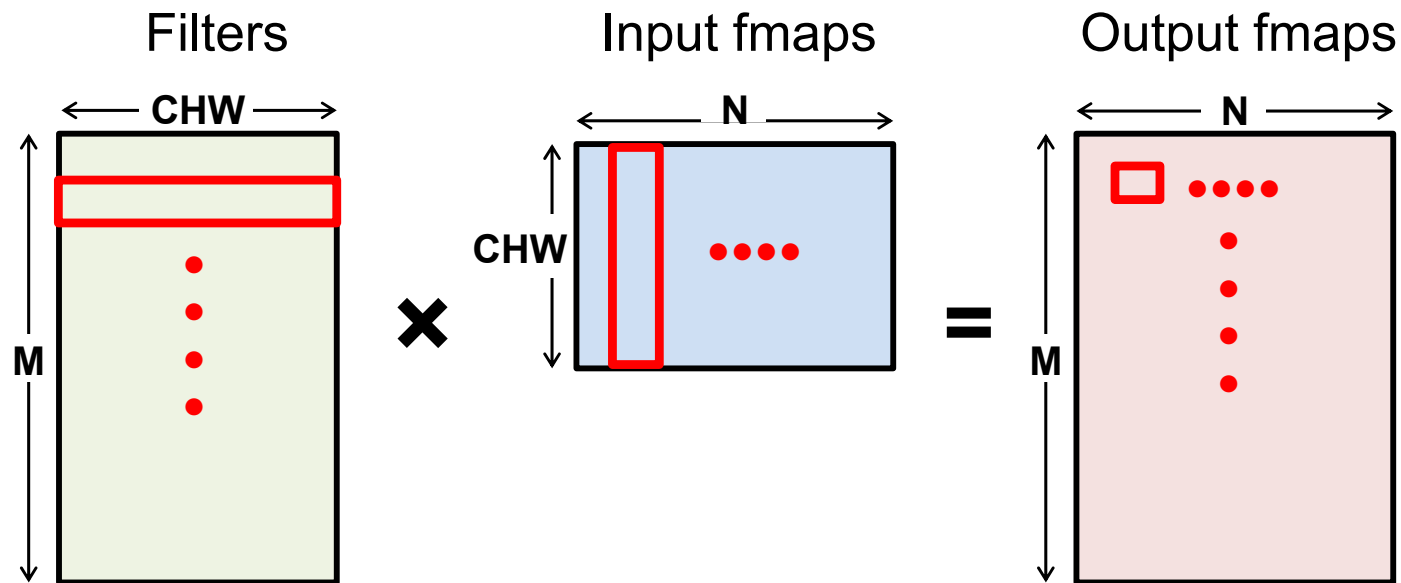
- After flattening, having a batch size of  $N$  turns the matrix-vector operation into a matrix-matrix multiply

# Fully-Connected (FC) Layer



- After flattening, having a batch size of  $N$  turns the matrix-vector operation into a matrix-matrix multiply

# Fully-Connected (FC) Layer



- After flattening, having a batch size of  $N$  turns the matrix-vector operation into a matrix-matrix multiply

How much temporal locality for naïve implementation? **None**

# Matrix-Matrix Multiply

Tensor: f\_MCHW[M, CHW]

Rank: CHW

	0	1	2	3	4	5	6	7	
Rank: M	0	9	4	7	2	4	5	4	3
	1	2	7	3	9	8	4	3	3
	2	2	5	8	7	9	2	5	7
	3	7	2	2	1	7	6	2	3

Tensor: i\_NCHW[N, CHW]

Rank: CHW

	0	1	2	3	4	5	6	7	
Rank: N	0	6	7	5	9	7	1	5	1
	1	4	2	5	4	8	9	3	8
	2	9	2	9	7	7	7	8	9
	3	8	9	2	4	1	3	3	9

Tensor: unknown[M, N]

Rank: N

	0	1	2	3
Rank: M	0	0	0	0
	1	0	0	0
	2	0	0	0
	3	0	0	0

# Matrix-Matrix Multiply Tiled

Tensor:  $f\_MCHW[M, CHW]$

Rank: CHW

		0	1	2	3	4	5	6	7
Rank: M	0	9	4	7	2	4	5	4	3
	1	2	7	3	9	8	4	3	3
	2	2	5	8	7	9	2	5	7
	3	7	2	2	1	7	6	2	3

Tensor:  $i\_NCHW[N, CHW]$

Rank: CHW

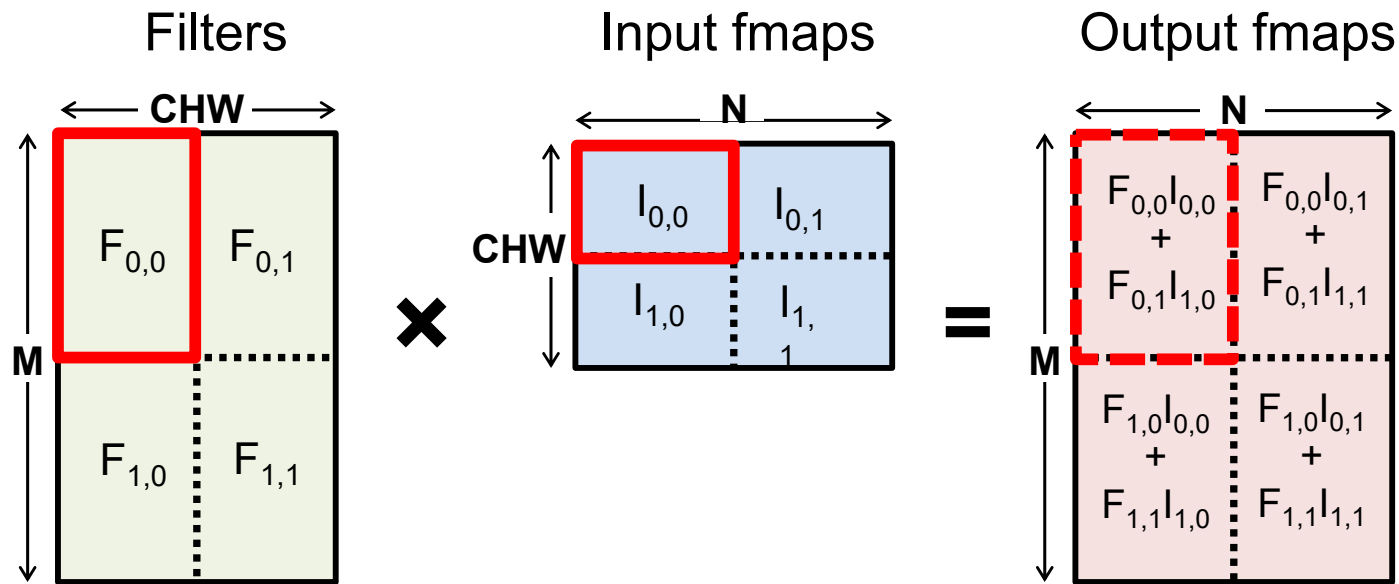
		0	1	2	3	4	5	6	7
Rank: N	0	6	7	5	9	7	1	5	1
	1	4	2	5	4	8	9	3	8
	2	9	2	9	7	7	7	8	9
	3	8	9	2	4	1	3	3	9

Tensor:  $unknown[M, N]$

Rank: N

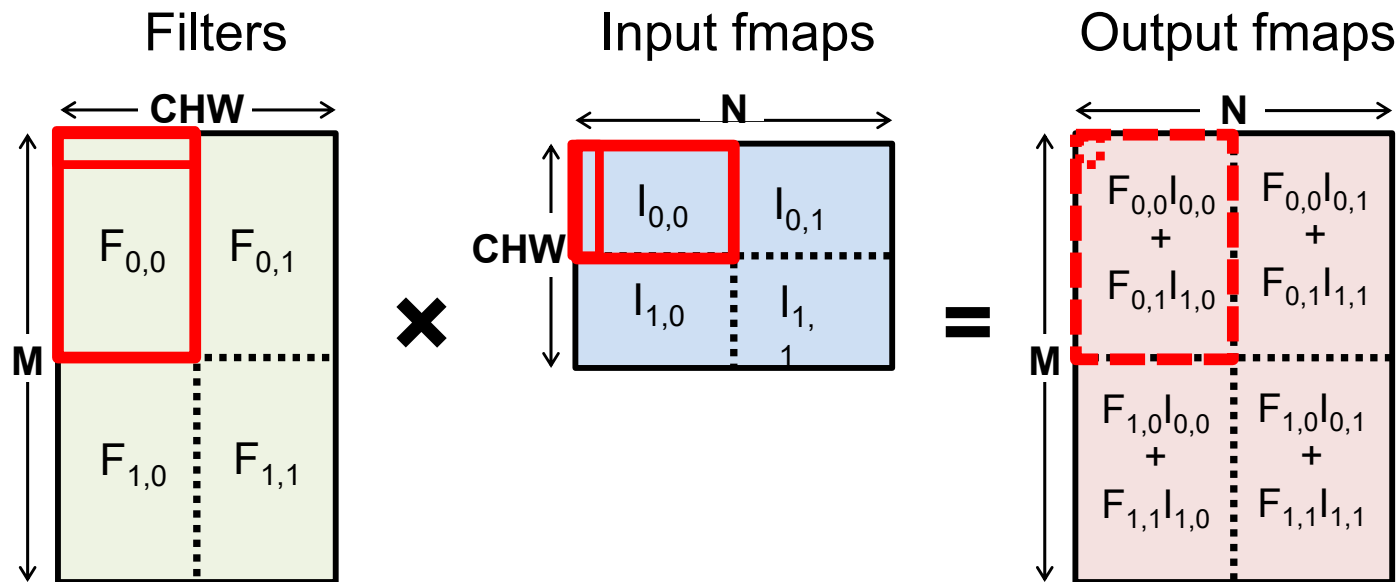
		0	1	2	3
Rank: M	0	0	0	0	0
	1	0	0	0	0
	2	0	0	0	0
	3	0	0	0	0

# Tiled Fully-Connected (FC) Layer



Matrix multiply tiled to fit in cache  
and computation ordered to maximize reuse of data in cache

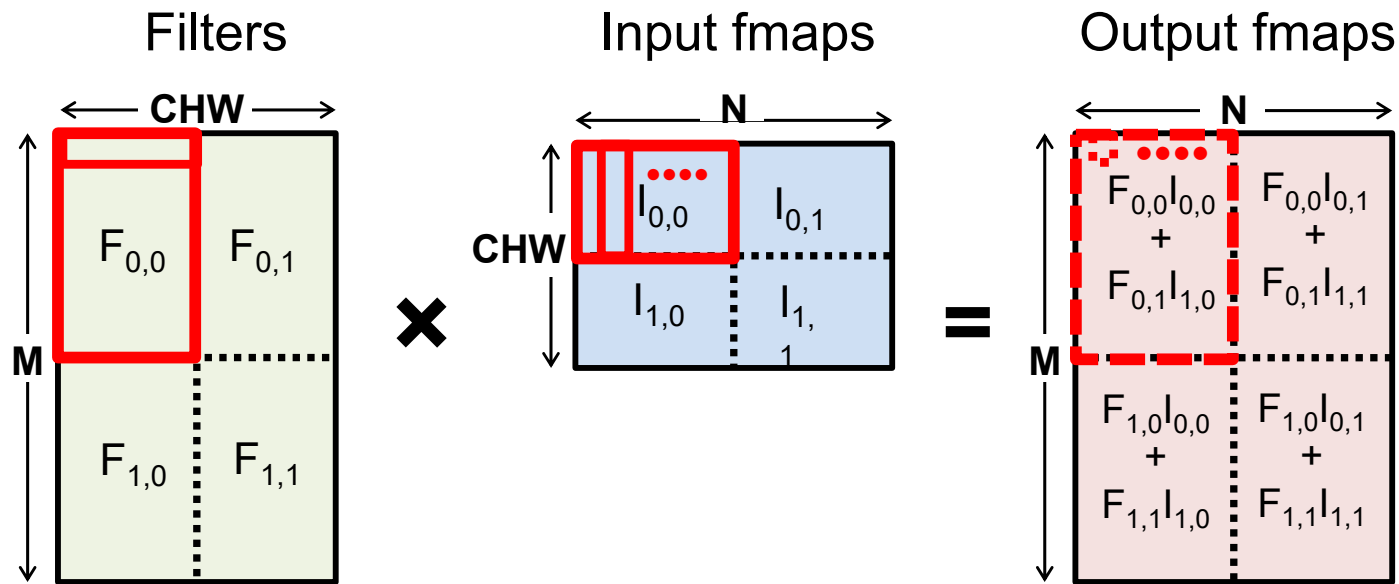
# Tiled Fully-Connected (FC) Layer



Matrix multiply tiled to fit in cache  
and computation ordered to maximize reuse of data in cache



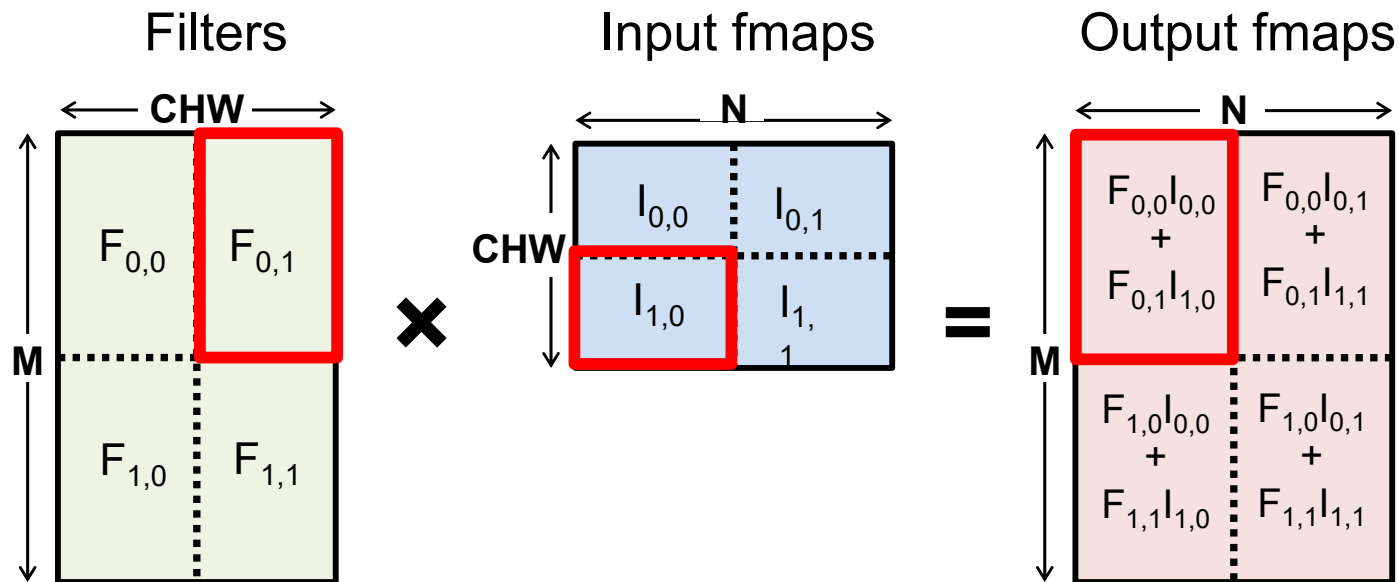
# Tiled Fully-Connected (FC) Layer



Matrix multiply tiled to fit in cache  
and computation ordered to maximize reuse of data in cache

\*Dotted line means partial result

# Tiled Fully-Connected (FC) Layer



Matrix multiply tiled to fit in cache  
and computation ordered to maximize reuse of data in cache

# Einsum for tiled FC

---

$$O_m = I_{n,chw} \times F_{m,chw}$$

$$I_{n,chw} \rightarrow I_{n1,chw1,n0,chw1}$$

$$F_{m,chw} \rightarrow F_{m1,chw1,m0,chw0}$$

$$O_{m1,m0} = I_{n1,chw1,n0,chw0} \times F_{m1,chw1,m0,chw0}$$

# Fully-Connected (FC) Layer

---

- Implementation: **Matrix Multiplication (GEMM)**
  - **CPU:** OpenBLAS, Intel MKL, etc
  - **GPU:** cuBLAS, cuDNN, etc
- Library will note shape of the matrix multiply and select implementation optimized for that shape.
- Optimization usually involves proper tiling to storage hierarchy

# Tradeoffs in Memories

# Overview of Memories

---

Memory consist of arrays of cells that hold a value.

- Types of Memories/Storage
  - Latches/Flip Flops (Registers)
  - SRAM (Register File, Caches)
  - DRAM (Main Memory)
  - Flash (Storage)

# Elements of Memory Operation

---

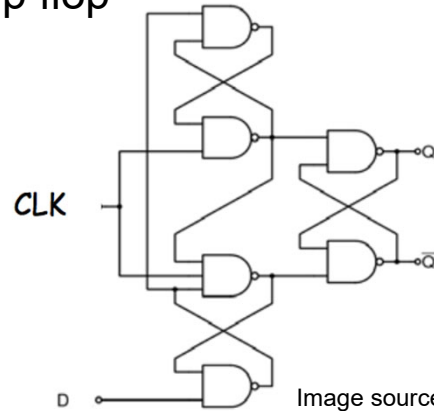
Implementations vary based on:

- How a memory cell holds a value?
  - How is a value obtained from a memory cell?
  - How is a value set in a memory cell?
  - How is array constructed out of individual cells?
- 
- Results in tradeoffs between cost, density, speed, energy and power consumption

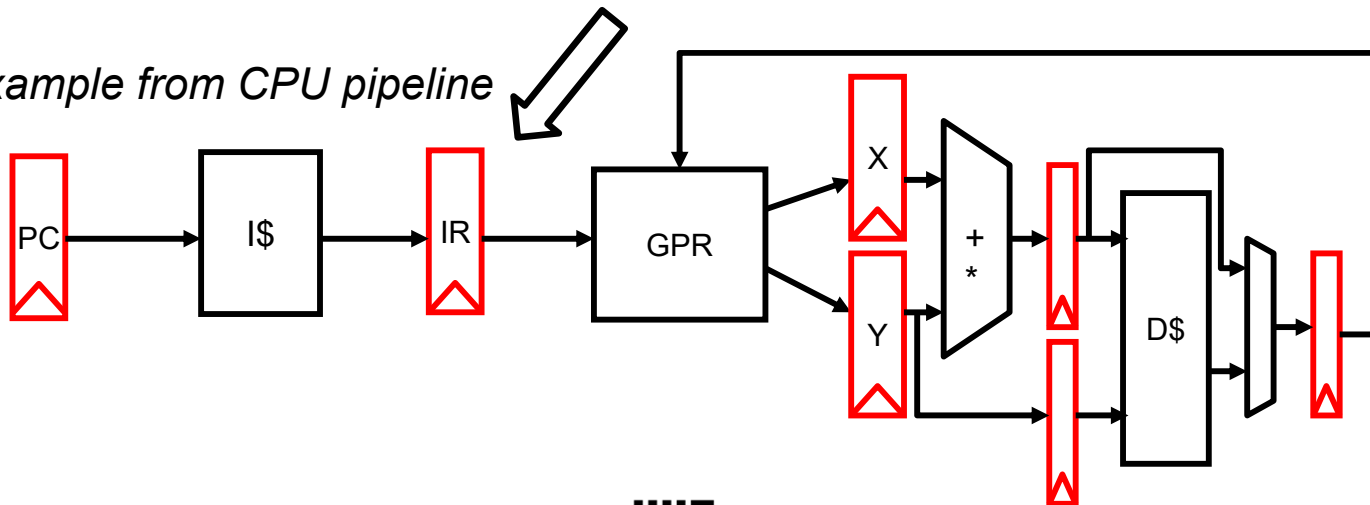
# Latches/Flip Flops

- Fast and low latency
- Located with logic

D-flip flop



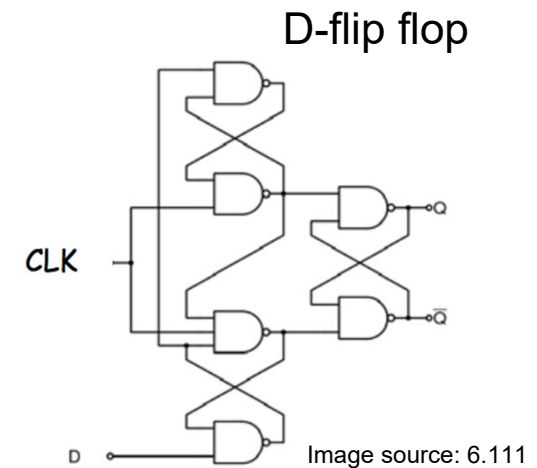
*Example from CPU pipeline*



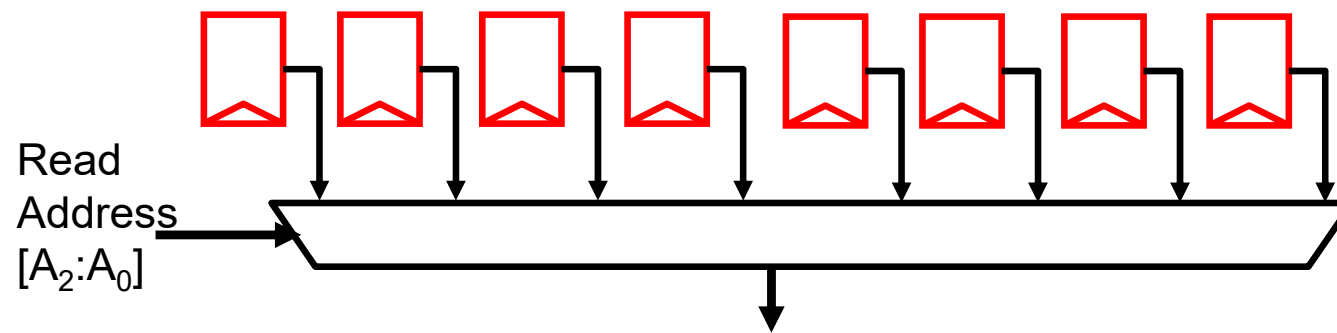


# Latches/Flip Flops (< 0.5 kB)

- Fast and low latency
- Located with logic
- Not very dense
  - 10+ transistors per bit
  - Usually use for arrays smaller than 0.5kB

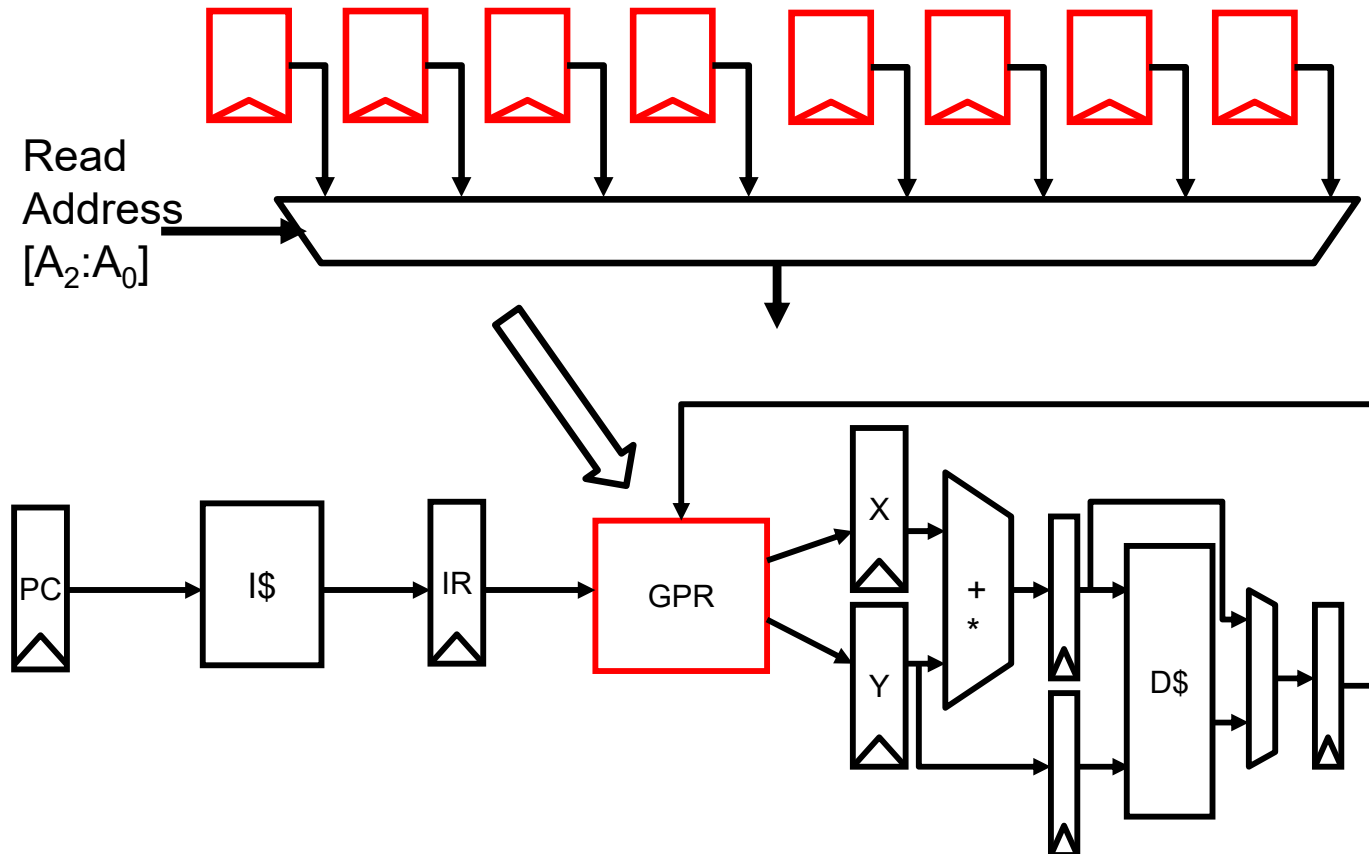


*Array of Flip flops*



# Latches/Flip Flops (< 0.5 kB)

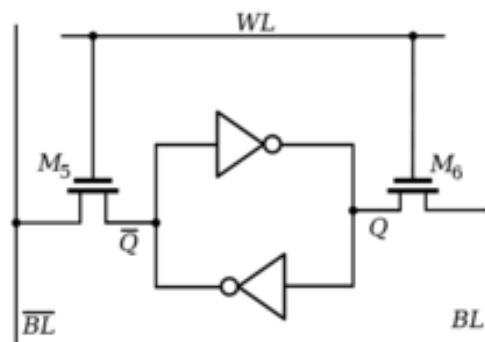
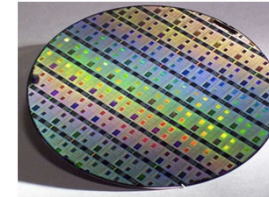
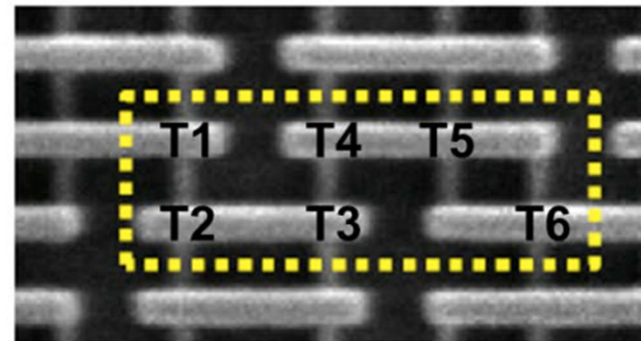
*Array of Flip flops*



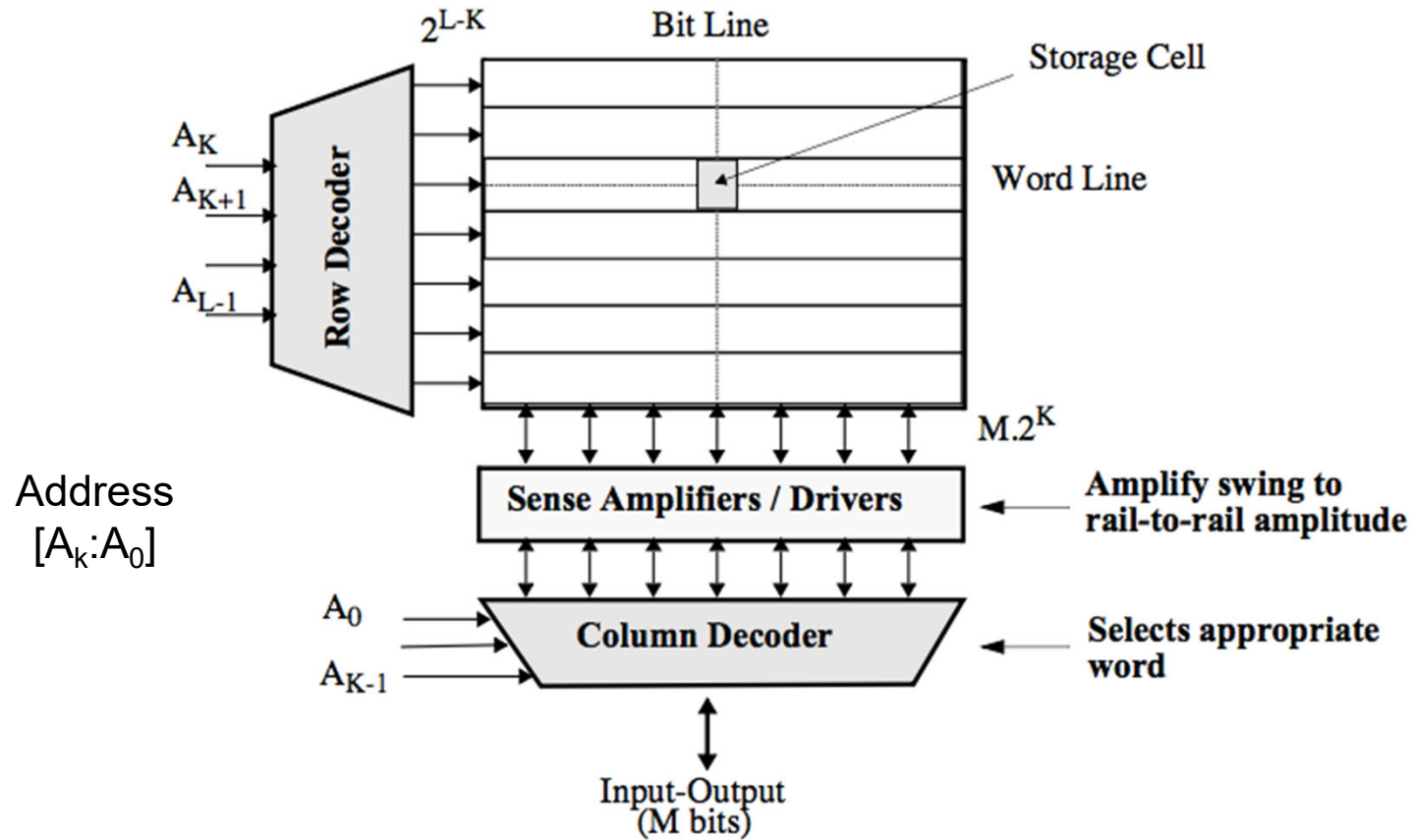
# SRAM

- Higher density than register
  - Usually, 6 transistors per bit-cell
- Less robust and slower than latches/flip-flop

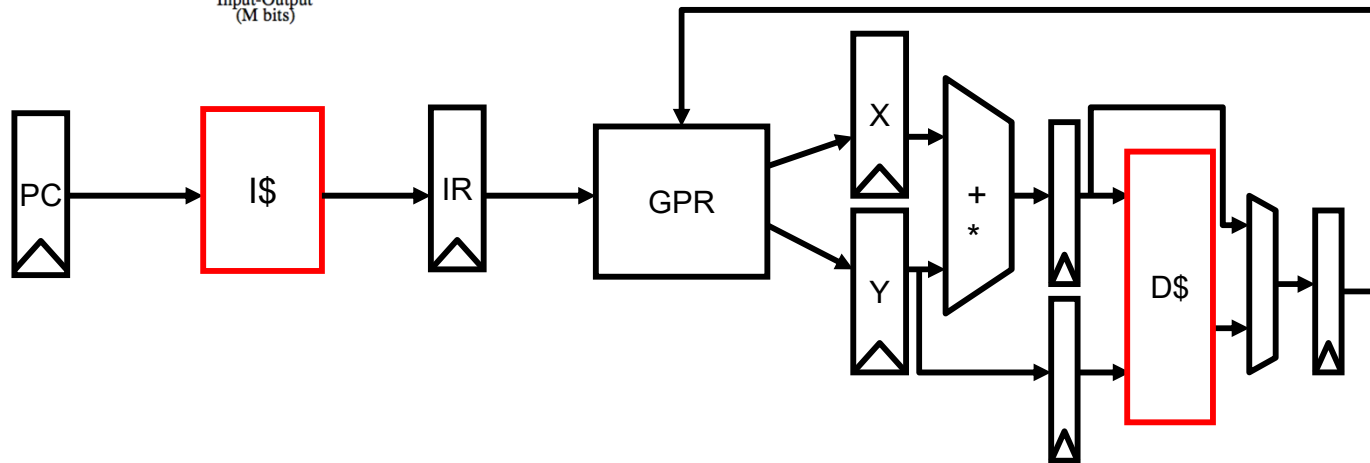
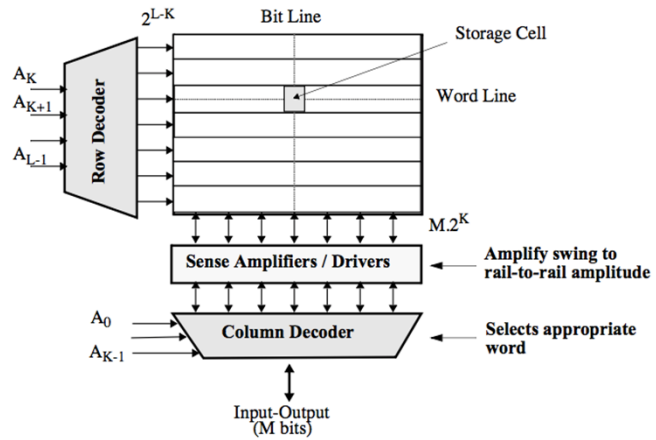
IC wafer

Bit cell size  $0.75\mu\text{m}^2$  in 14nm

# SRAM (kB – MB)

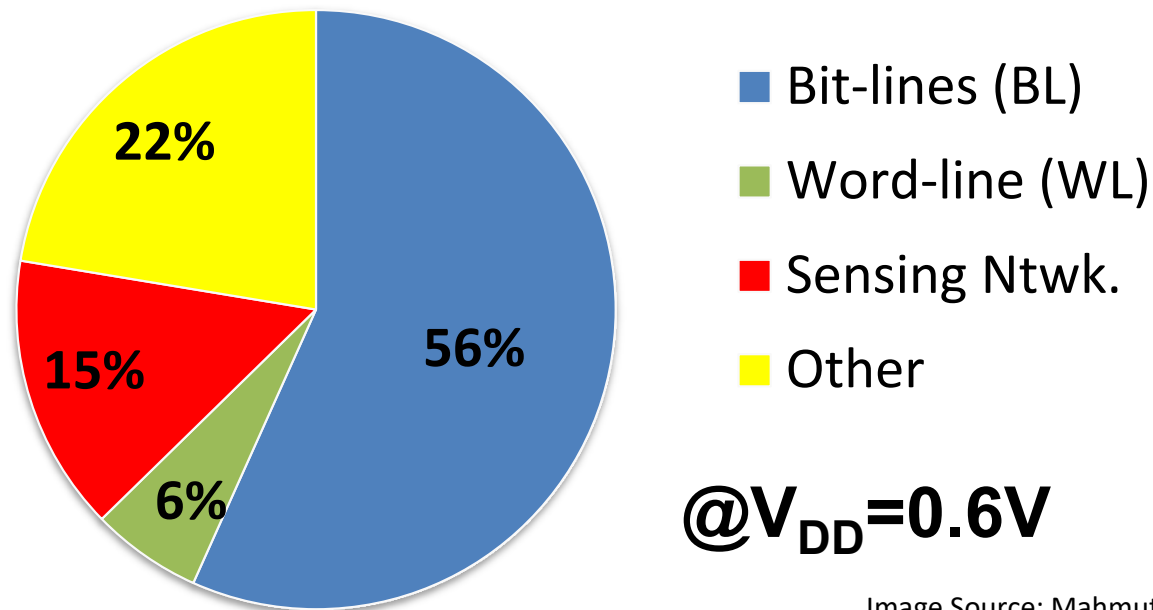


# SRAM



# SRAM Power Dominated by Bit Line

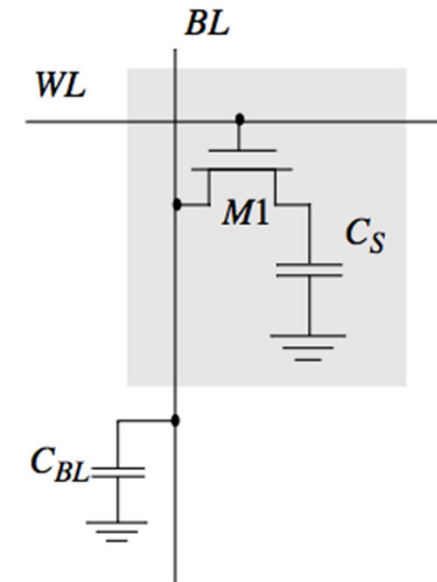
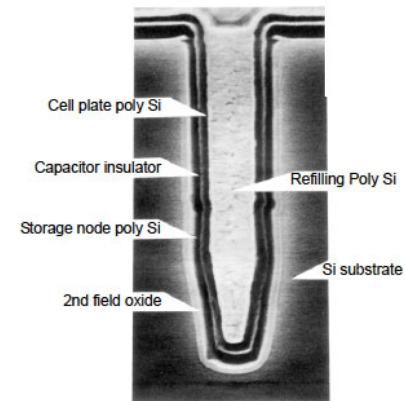
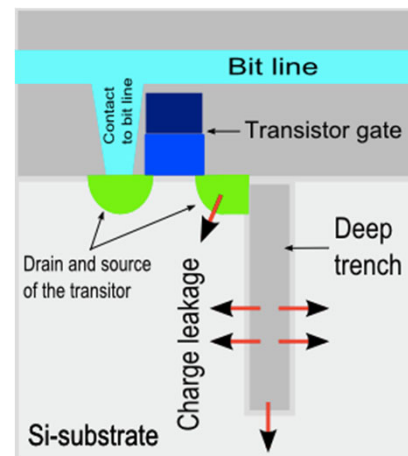
## Measured SRAM Power Breakdown



Larger array  $\rightarrow$  Longer bit-lines  
 $\rightarrow$  Higher capacitance  $\rightarrow$  Higher power

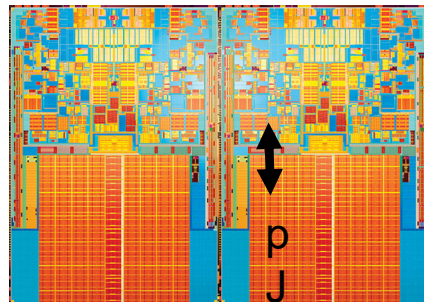
# DRAM

- Higher density than SRAM
  - 1 transistor per bit-cell
  - Needs periodic refresh
- Special device process

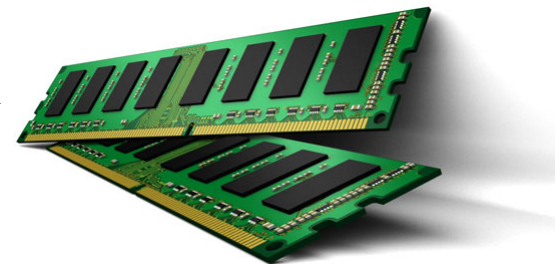


# DRAM (GB)

- Higher density than SRAM
  - 1 transistor per bit-cell
  - Needs periodic refresh
- Special device process
  - Usually off-chip (except eDRAM – which is pricey!)
  - Off-chip interconnect has much higher capacitance



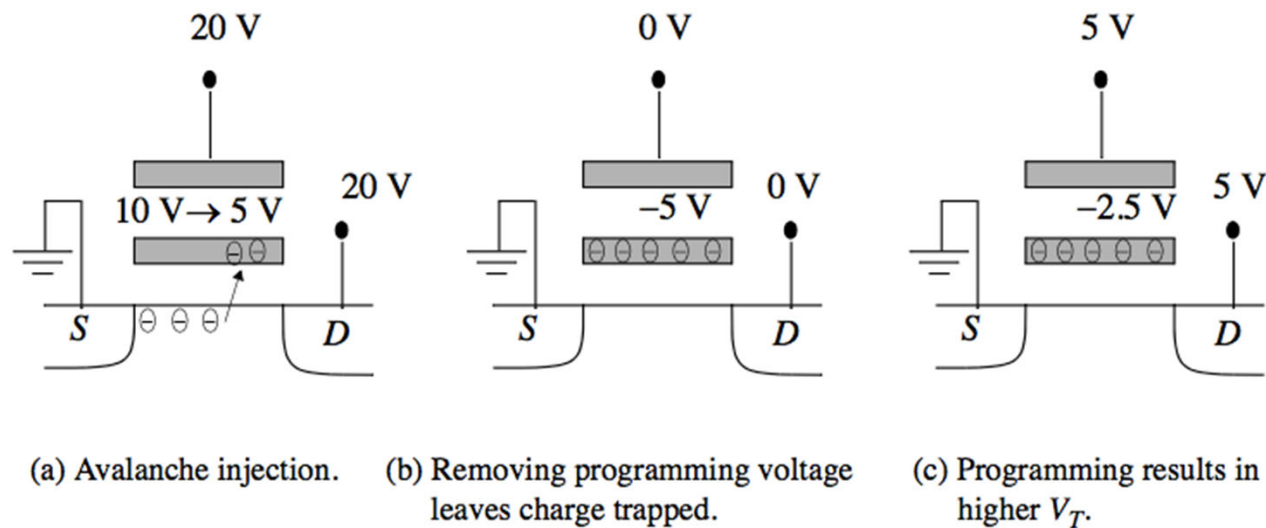
← nJ →



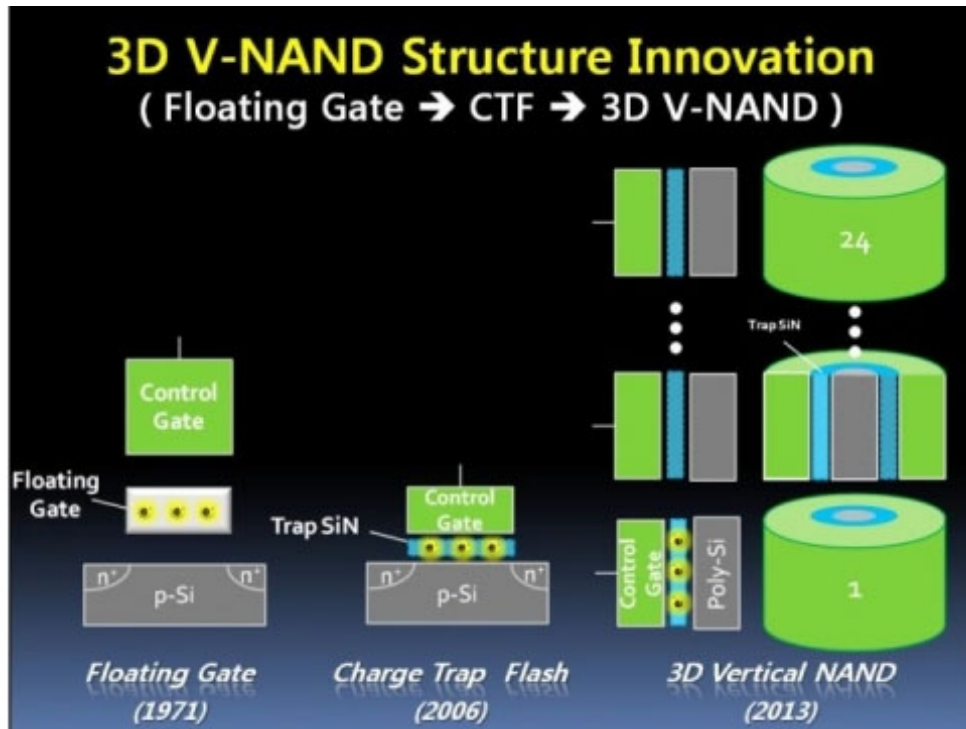


# Flash (100GB to TB)

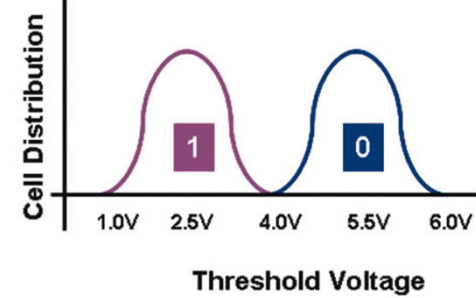
- More dense than DRAM
- Non-volatile
  - Needs high powered write (change  $V_{TH}$  of transistor)



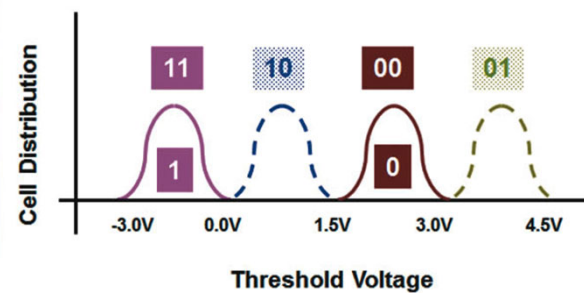
# Flash Memory



Single Level Cell (SLC)



Multi-levels cell (MLC)



Single Level Cell (SLC)

Multi-levels cell (MLC)

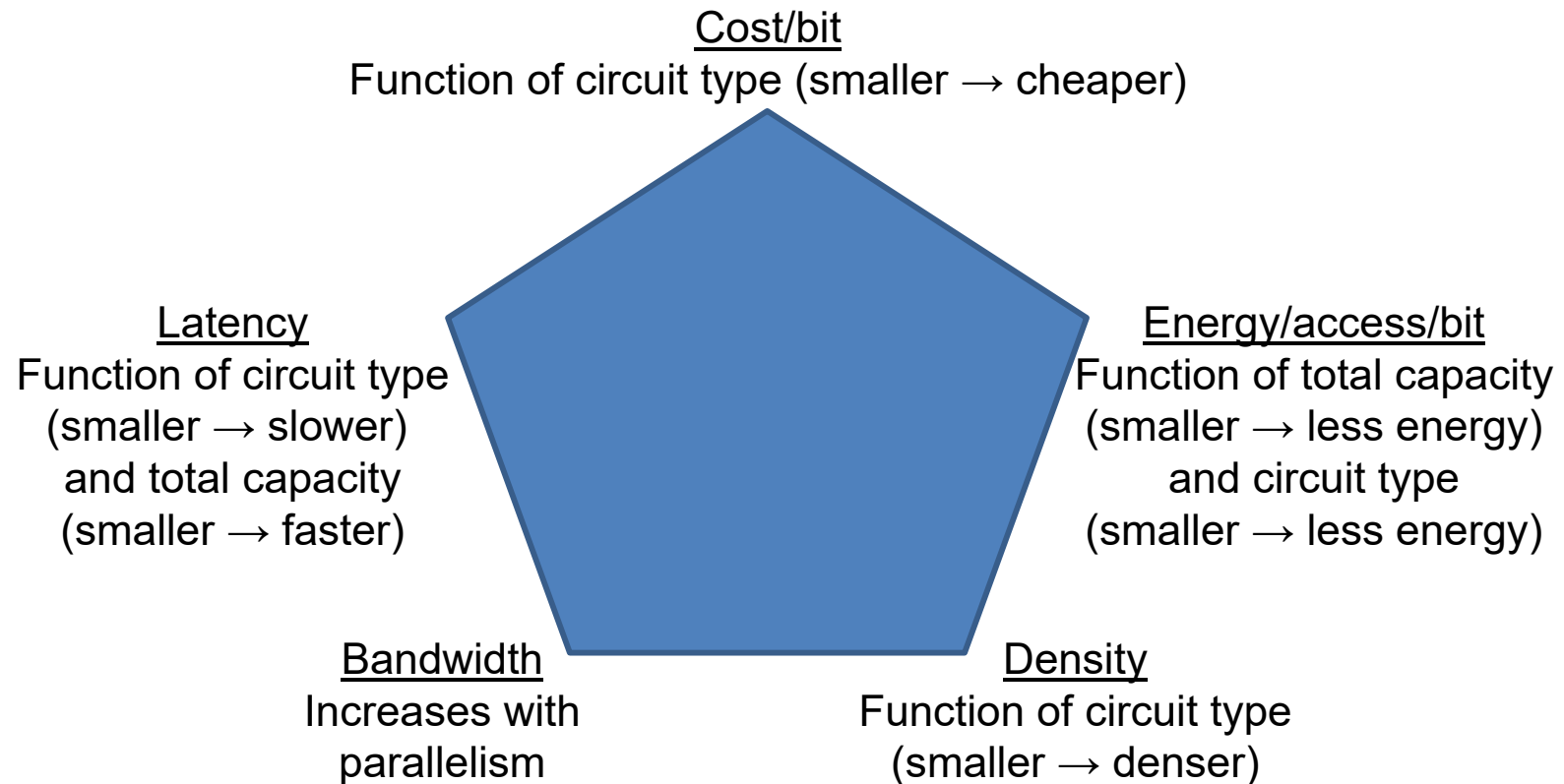
48 layer, Ternary level cell (TLC)

Aug 2015

256 Gb per die (for SSD)



# Memory Tradeoffs



Most attributes tend to improve with technology scaling, lower voltage and sometimes smaller capacitors



# Summary

---

- Reduce main memory access with caches
  - Main memory (i.e., DRAM) is slow and has high energy consumption
  - Exploits spatial and temporal locality
- Tiling to reduce cache misses
  - Possible since processing order does not affect result (MACs are commutative)
  - Add levels to loop nest to improve temporal locality
  - Size of tile depends on cache size and cache associativity
- Tradeoffs in storage technology
  - Various tradeoffs in cost, speed, energy, capacity...
  - Different technologies appropriate at different spots in the design

## Next Lecture: Vectorization

Thank you!