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

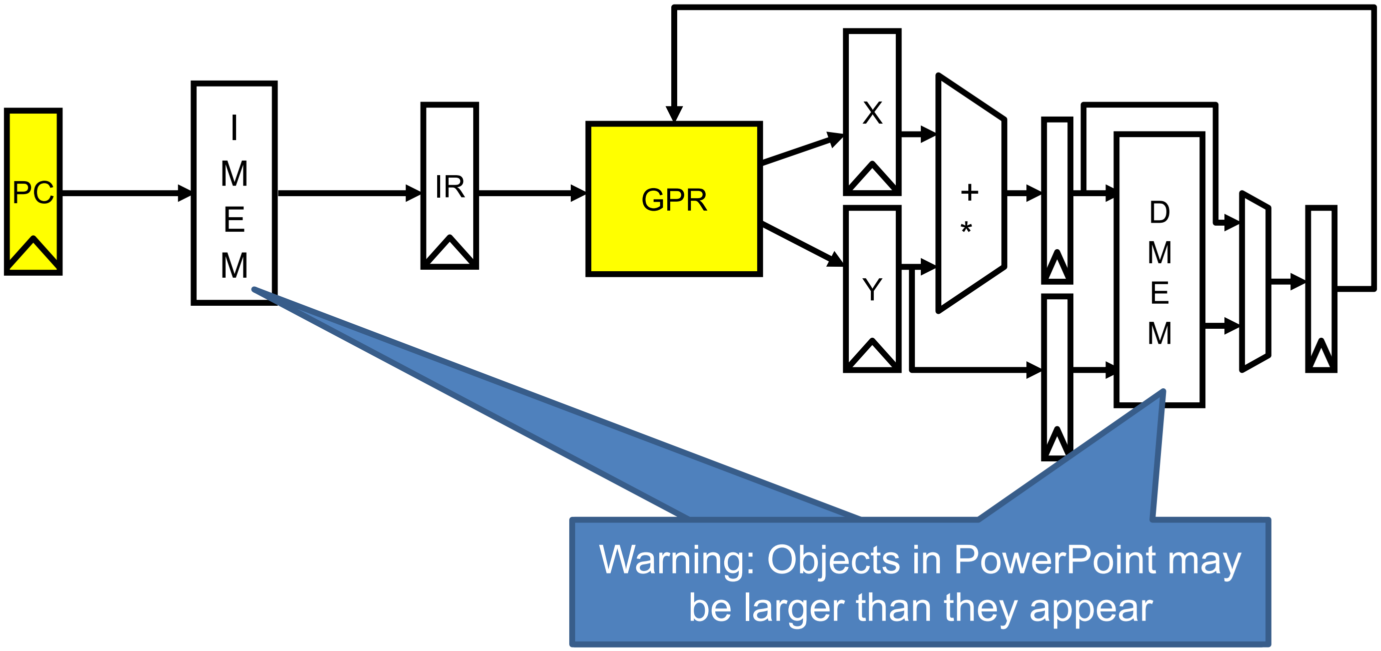
- Understand impact of memory hierarchy
  - Overview of caches
  - Structuring algorithms to work well in caches using tiling
  - Storage technologies

Readings for this Week

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

.

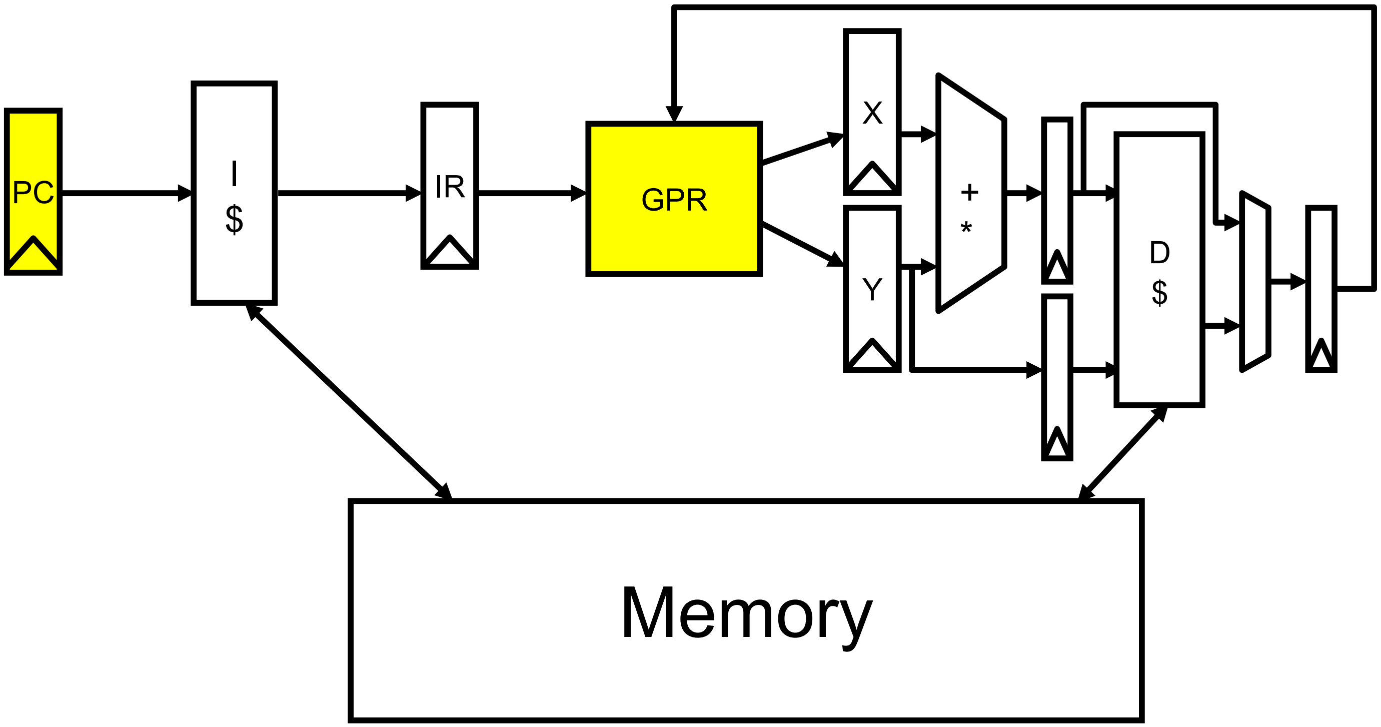
Simple Pipelined µArchitecture



Warning: Objects in PowerPoint may be larger than they appear

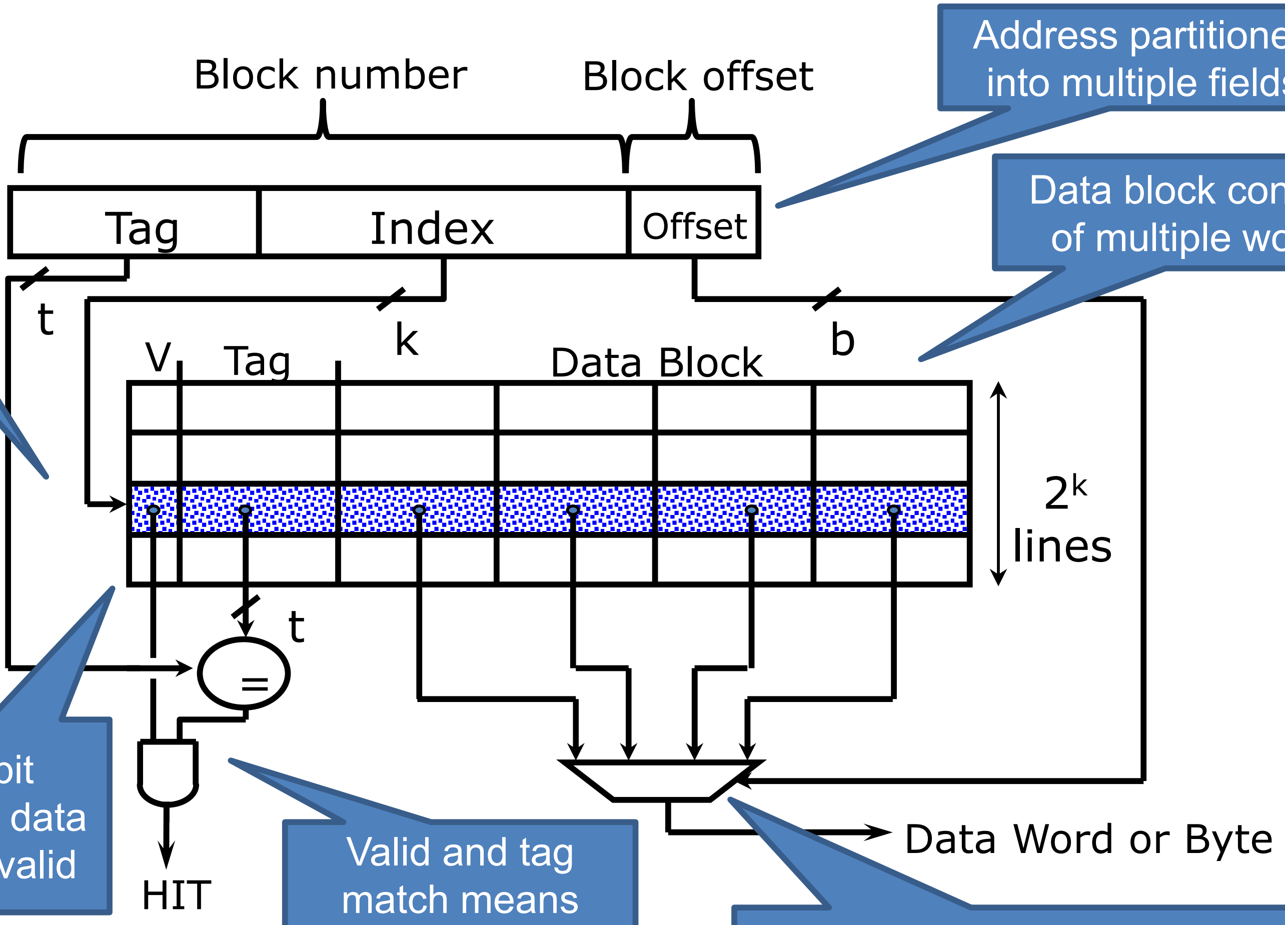
What are consequences of putting large memory (e.g., megabytes) directly in pipeline?

Pipelined µArchitecture with Caches



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

Direct Mapped Cache



Address partitioned into multiple fields

Index picks row

Data block consists of multiple words

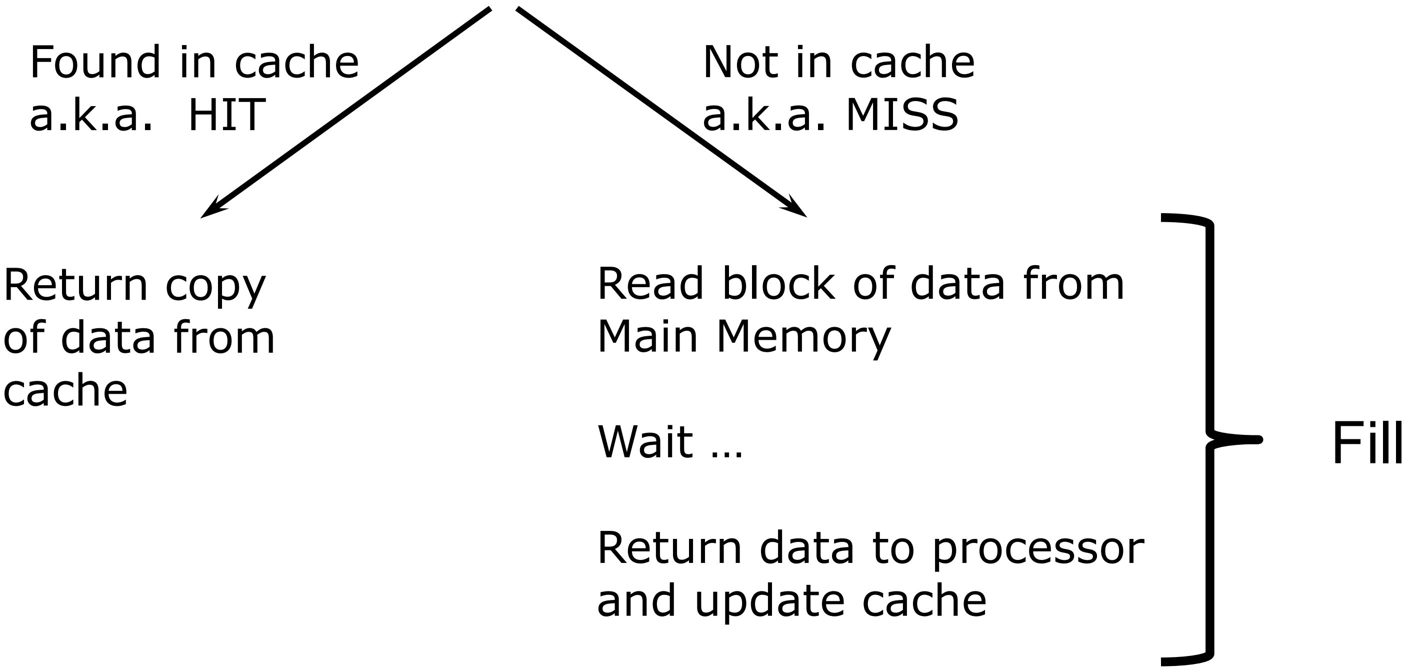
Valid bit indicates data block is valid

Valid and tag match means data is in cache

Offset selects desired word

Cache Operation

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



Found in cache
a.k.a. HIT

Return copy of data from cache

Not in cache
a.k.a. MISS

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

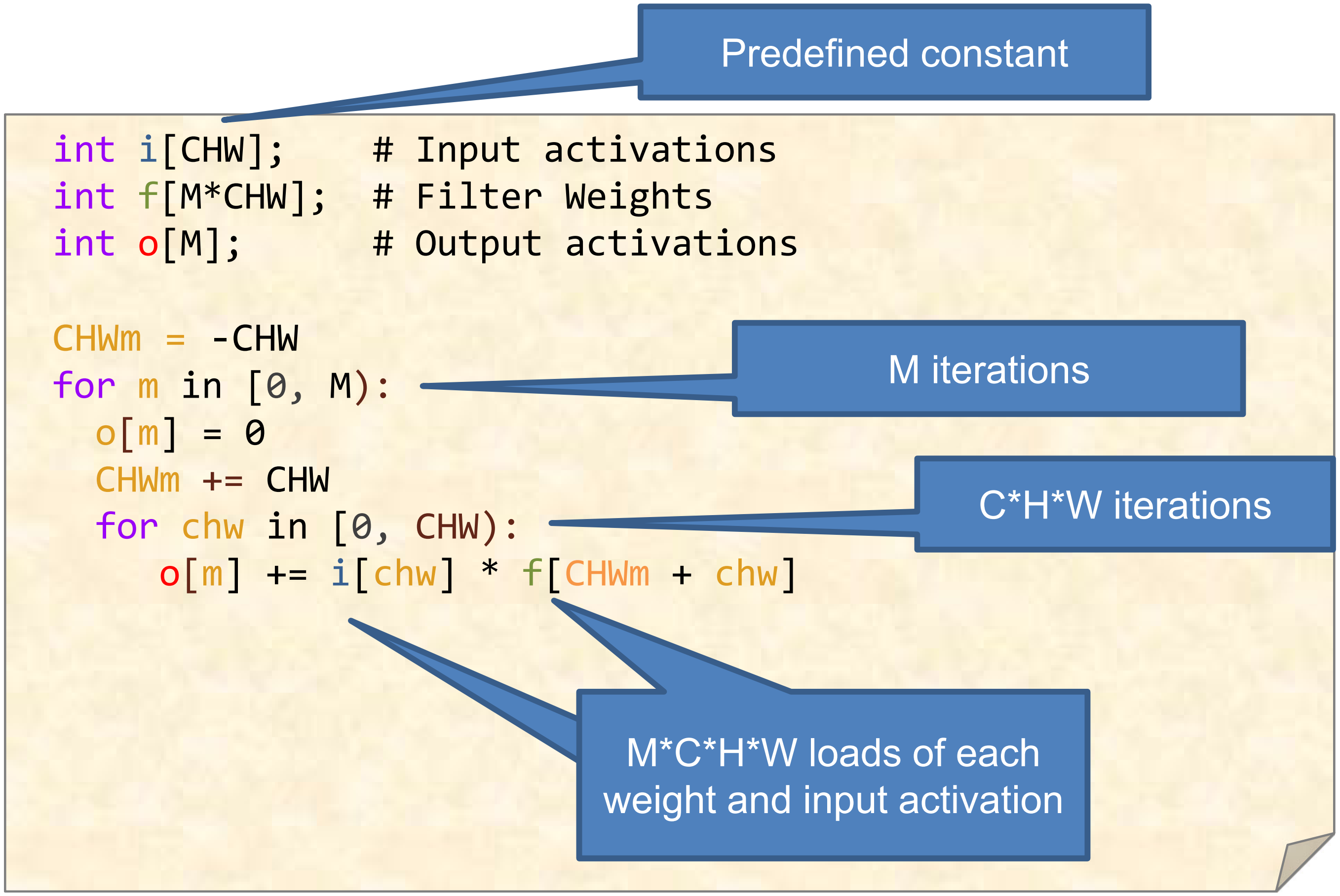
- Cache hit:
  - *write through:* write both cache & memory
    - generally higher traffic but simplifies cache in processor pipeline
  - *write back:* write cache only
    (memory is written only when the entry is evicted)
    - a dirty bit per block can further reduce the traffic
- Cache miss:
  - *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



```
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]
```

Predefined constant

M iterations

C*H*W iterations

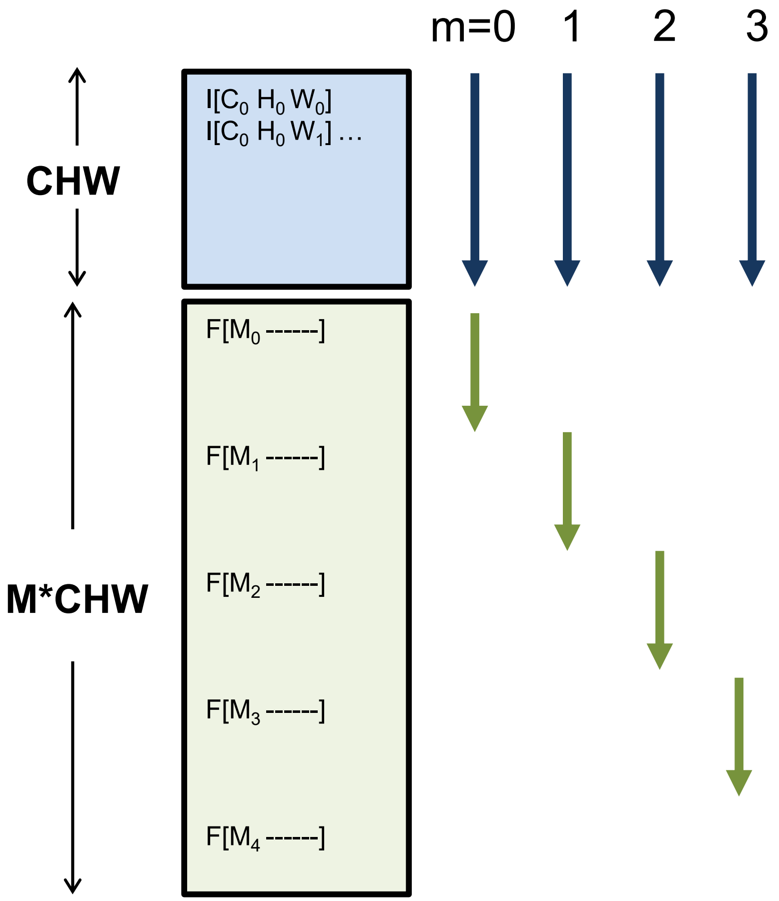
M*C*H*W loads of each weight and input activation

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?

FC – Data Reference Pattern



m=0 1 2 3

CHW

$I[C_0\ H_0\ W_0]$
$I[C_0\ H_0\ W_1]$ …

M*CHW

$F[M_0$ ------]

$F[M_1$ ------]

$F[M_2$ ------]

$F[M_3$ ------]

$F[M_4$ ------]

Not drawn to scale

Input activation locality…

Spatial?

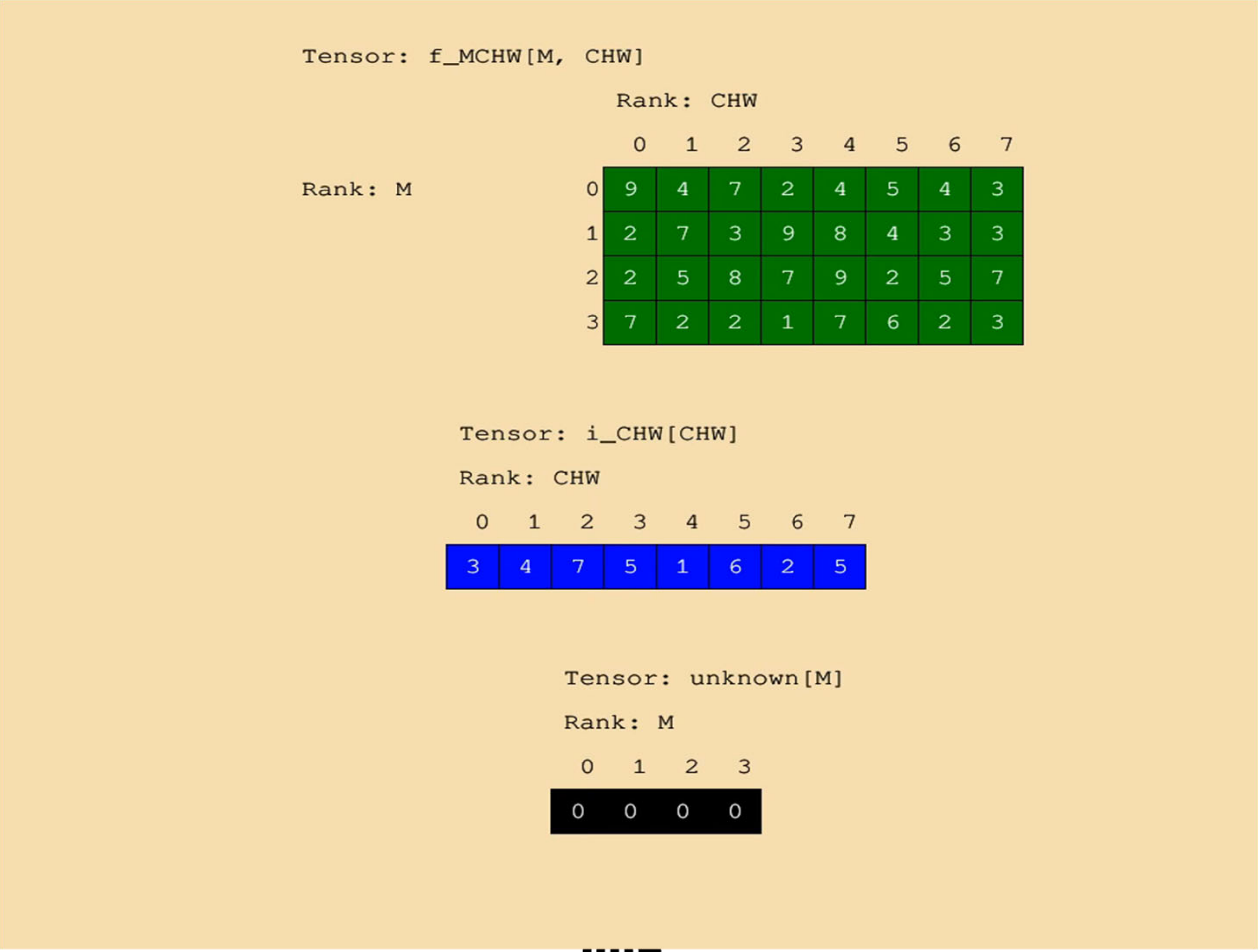
Temporal?

Weight locality…

Spatial?

Temporal?

FC – Data Reference Pattern



Tensor: f_MCHW[M, CHW]

Rank: CHW

Rank: M

	0	1	2	3	4	5	6	7
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

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?

What does this imply for
Input activation temporary locality?

Computational Intensity

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

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

Number MACS:

Computational Intensity – Ideal FC

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

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

Number MACS: M*C*H*W

Filter weight accesses:

Input activation accesses:

Output activation accesses:

Computational Intensity =

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

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

```
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*C*H*W

Filter weight accesses

Input activation accesses

Output activation accesses

Computational Intensity =

Einsum for strip mined FC

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

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

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

Fully Connected – Strip Mined

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

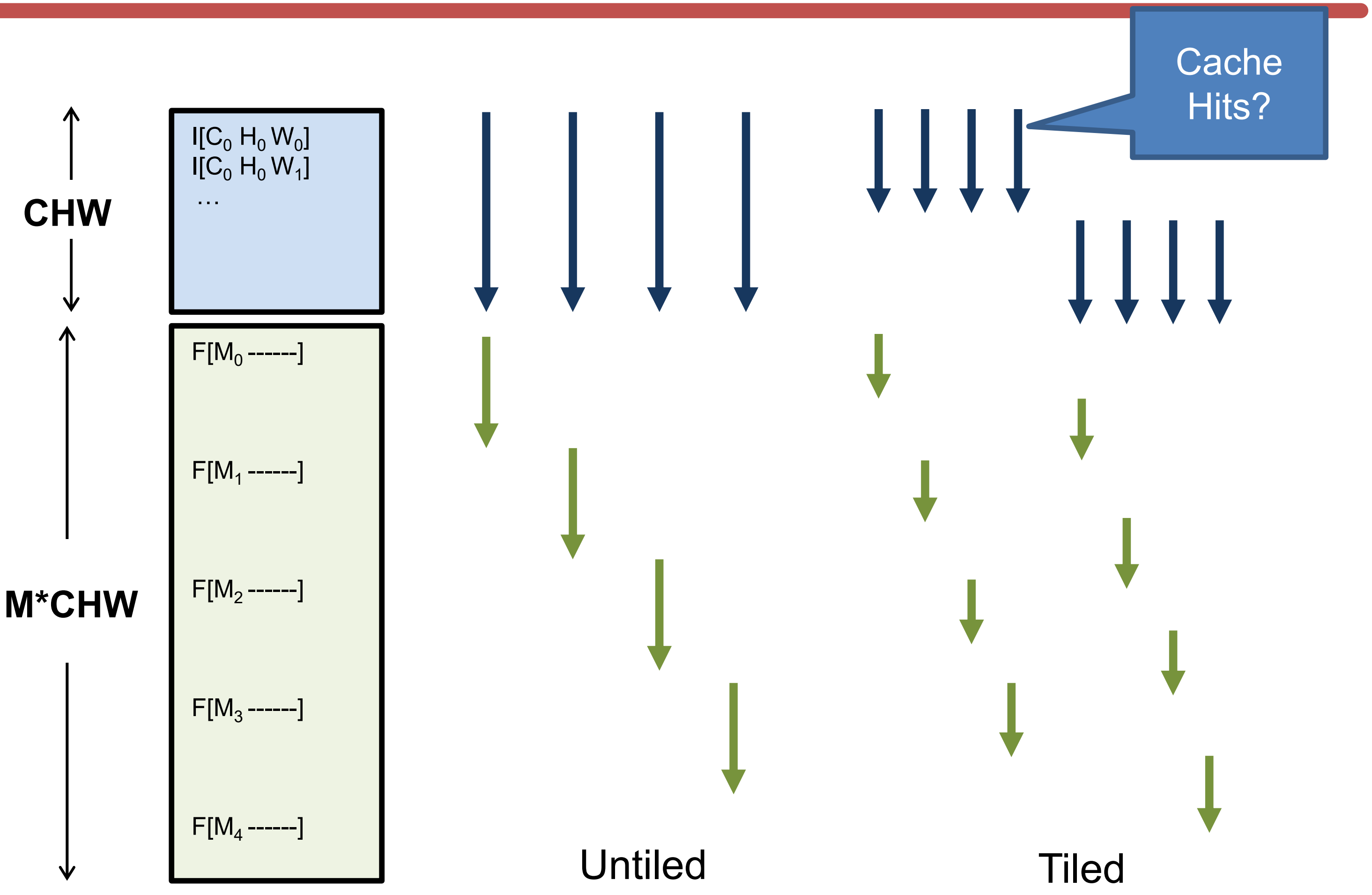
```
// Level 1
for chw1 in [0, CHW1):
  for m in [0, M):
// Level 0
    for chw0 in [0, CHW0):
      chw = CHW0*chw1+chw0
      o[m] += i[chw] * f[CHW*m + chw]
```

CHW1*CHW0 = C*H*W

Inner loop working set = CHW0

Just considering input activations, what value should CHW0 be?

FC - Strip Mined Data Reference Pattern



CHW

$I[C_0\ H_0\ W_0]$
$I[C_0\ H_0\ W_1]$
…

M*CHW

$F[M_0$ ------]

$F[M_1$ ------]

$F[M_2$ ------]

$F[M_3$ ------]

$F[M_4$ ------]

Untiled

Tiled

Cache Hits?

Not drawn to scale

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*C*H*W

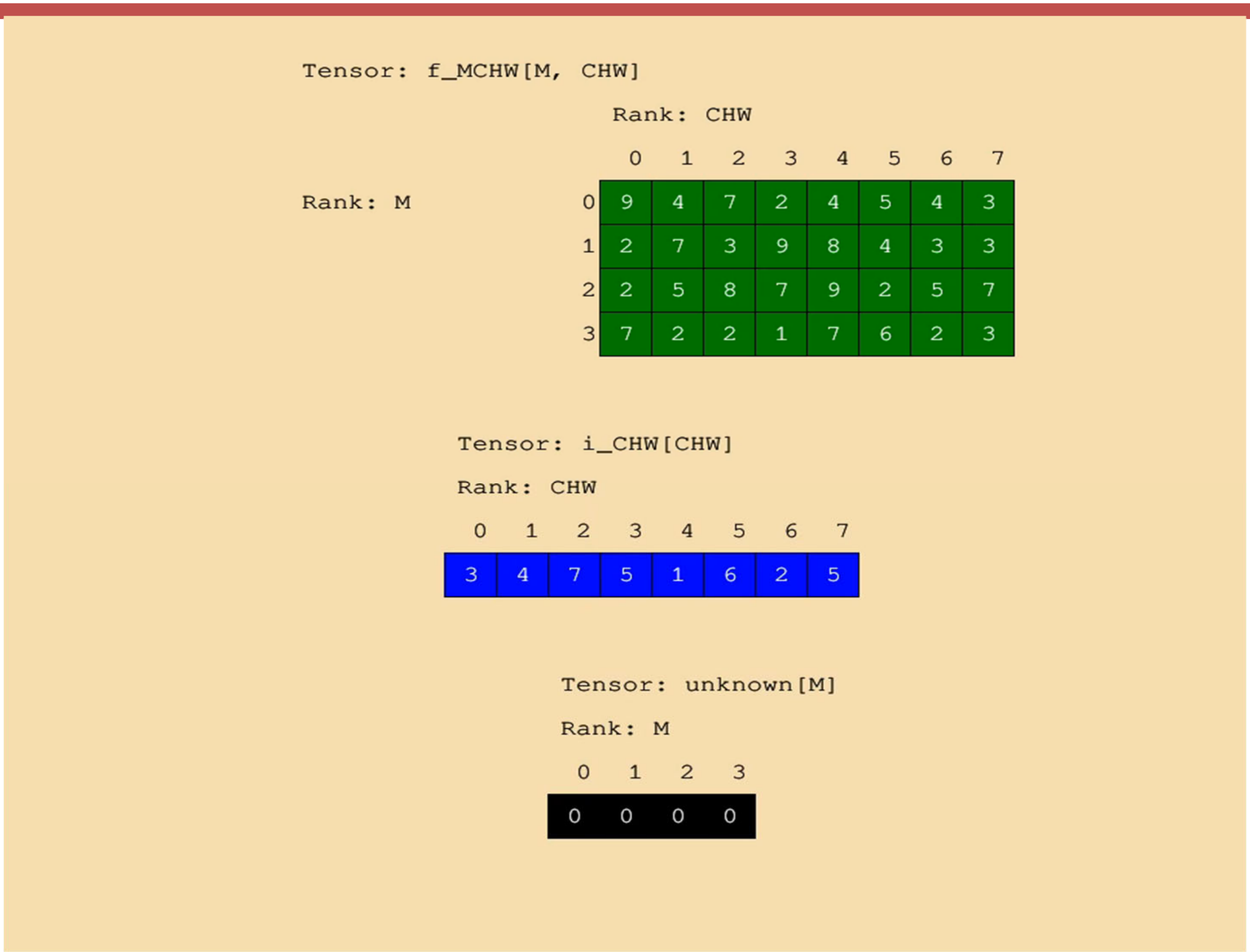
Filter weight accesses:

Input activation accesses:

Output activation accesses

Computational Intensity =

Matrix-Vector Multiply – Strip Mined



Tensor: f_MCHW[M, CHW]

Rank: CHW

Rank: M

	0	1	2	3	4	5	6	7
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

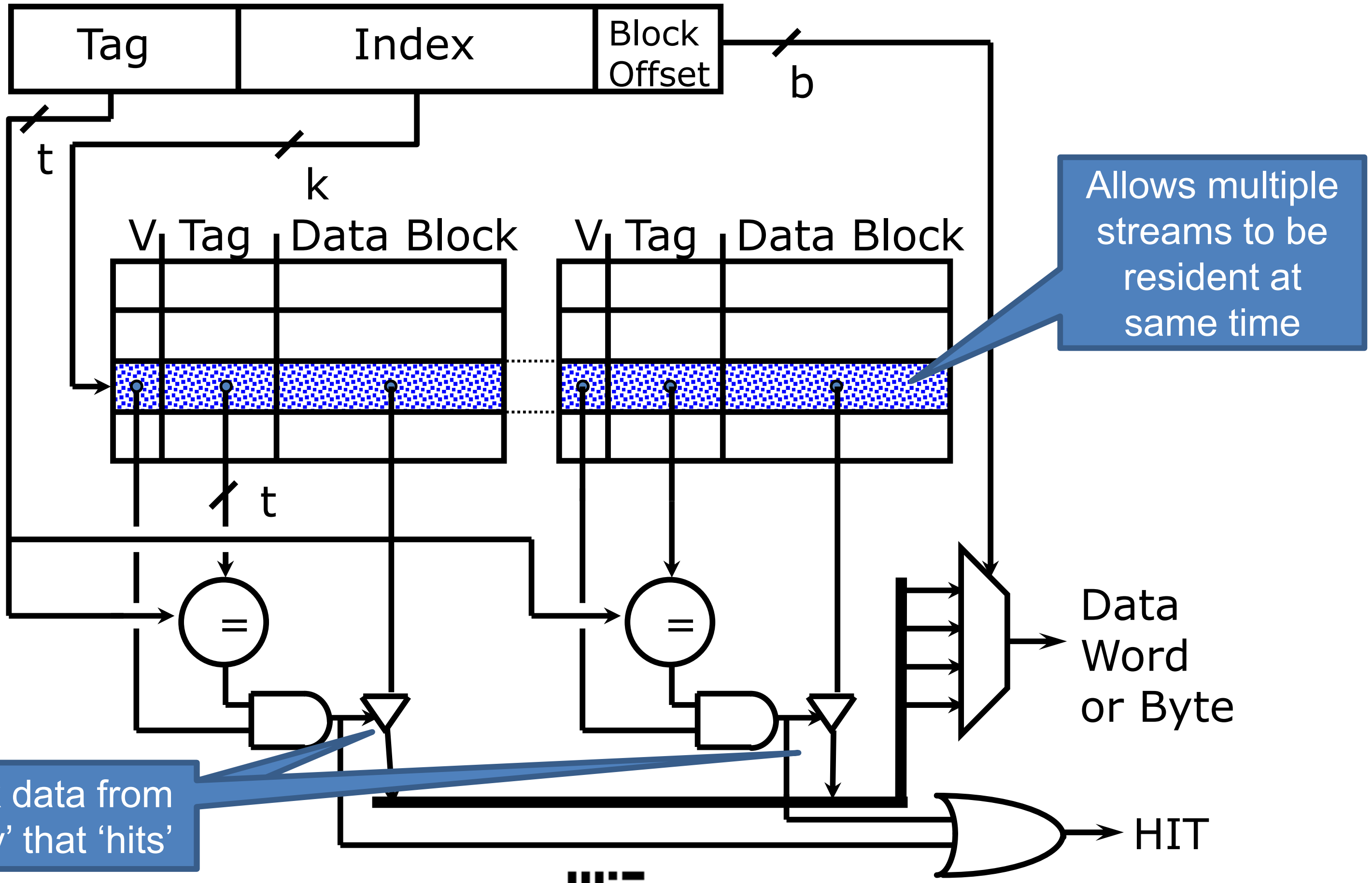
0	1	2	3	4	5	6	7
3	4	7	5	1	6	2	5

Tensor: unknown[M]

Rank: M

0	1	2	3
0	0	0	0

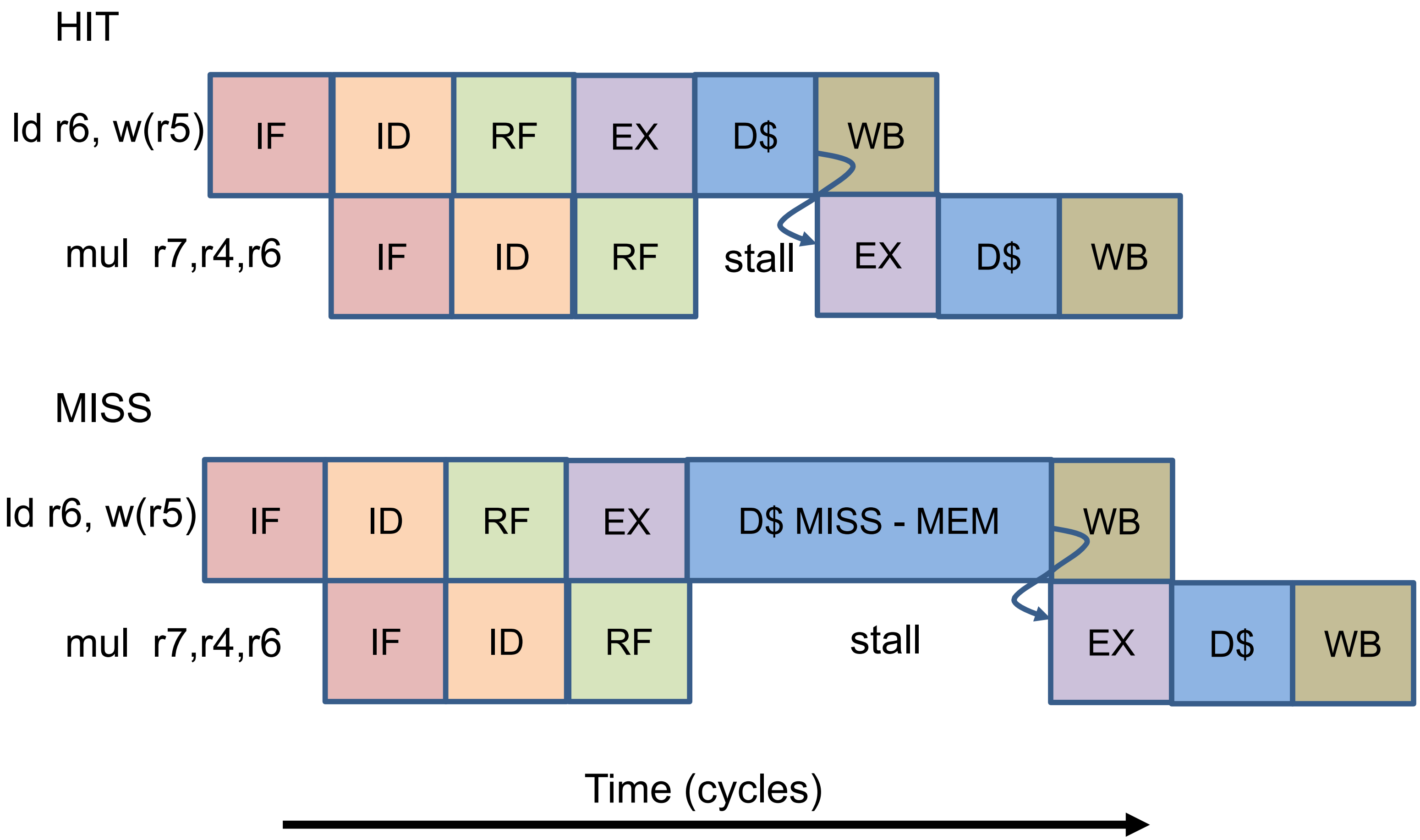
Associative Cache



Allows multiple streams to be resident at same time

Pick data from 'way' that 'hits'

Cache Miss Pipeline Diagram



HIT

ld r6, w(r5)	IF	ID	RF	EX	D$	WB		
mul r7,r4,r6		IF	ID	RF	stall	EX	D$	WB

MISS

ld r6, w(r5)	IF	ID	RF	EX	D$ MISS - MEM			WB		
mul r7,r4,r6		IF	ID	RF	stall			EX	D$	WB

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

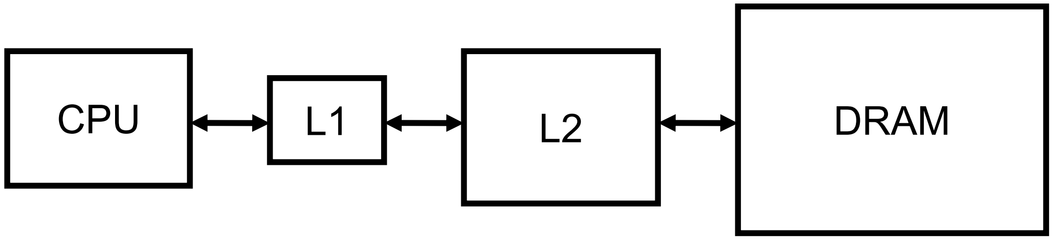
Hardware Data Prefetching

- 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



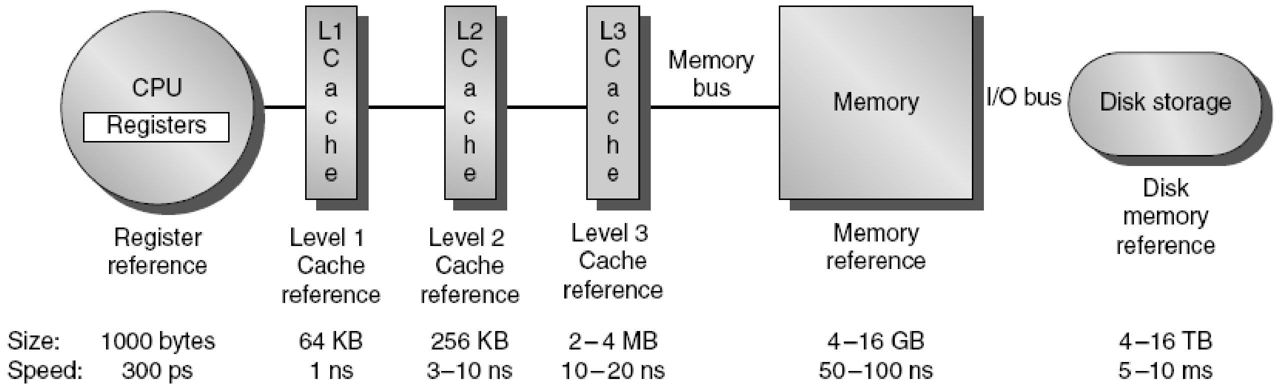
Metrics:

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



CPU

Registers

L1 Cache

L2 Cache

L3 Cache

Memory bus

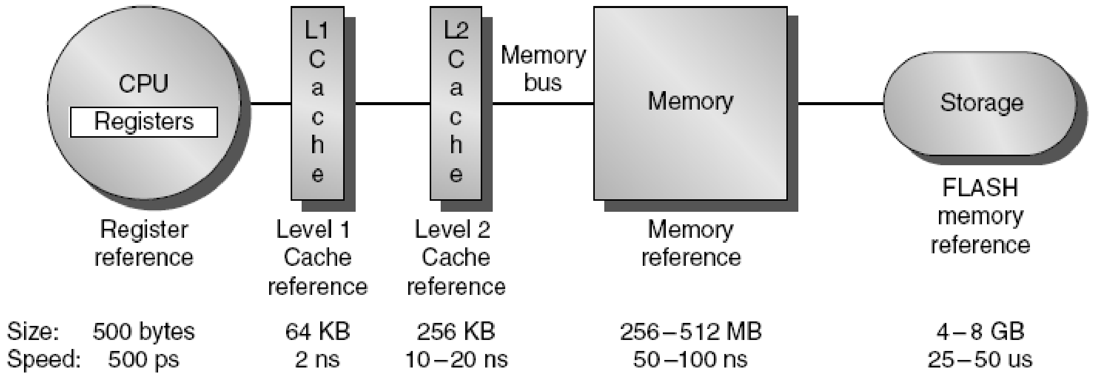
Memory

I/O bus

Disk storage

	Register reference	Level 1 Cache reference	Level 2 Cache reference	Level 3 Cache reference	Memory reference	Disk memory reference
Size:	1000 bytes	64 KB	256 KB	2–4 MB	4–16 GB	4–16 TB
Speed:	300 ps	1 ns	3–10 ns	10–20 ns	50–100 ns	5–10 ms

(a) Memory hierarchy for server



CPU

Registers

L1 Cache

L2 Cache

Memory bus

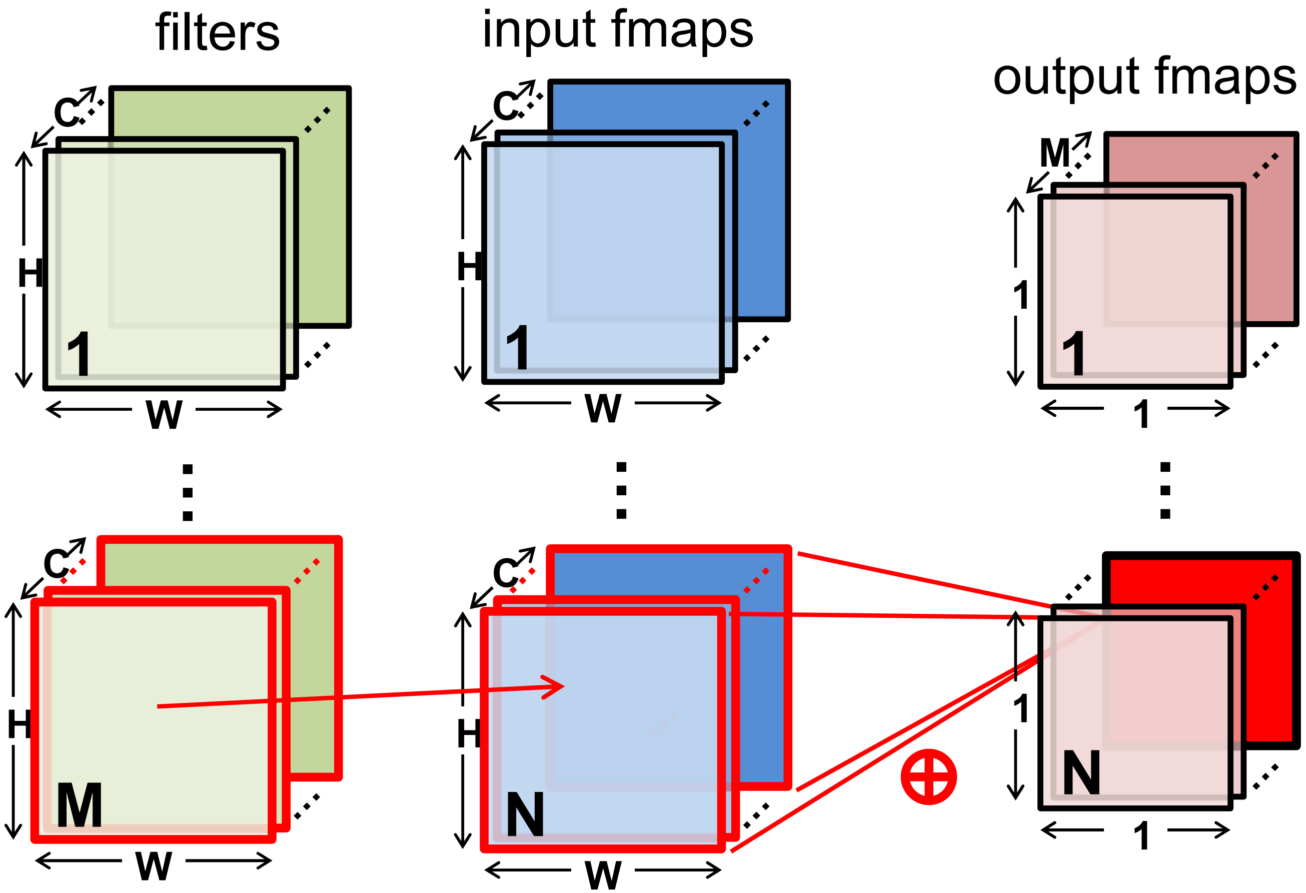
Memory

Storage

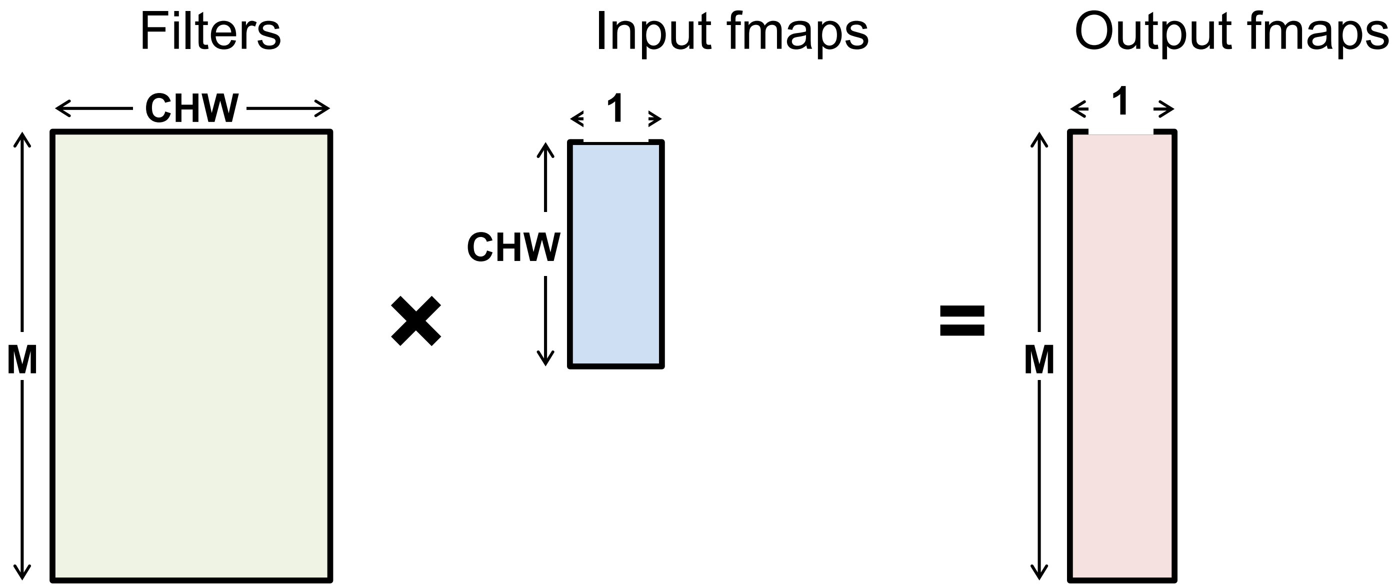
	Register reference	Level 1 Cache reference	Level 2 Cache reference	Memory reference	FLASH memory reference
Size:	500 bytes	64 KB	256 KB	256–512 MB	4–8 GB
Speed:	500 ps	2 ns	10–20 ns	50–100 ns	25–50 us

(b) Memory hierarchy for a personal mobile device

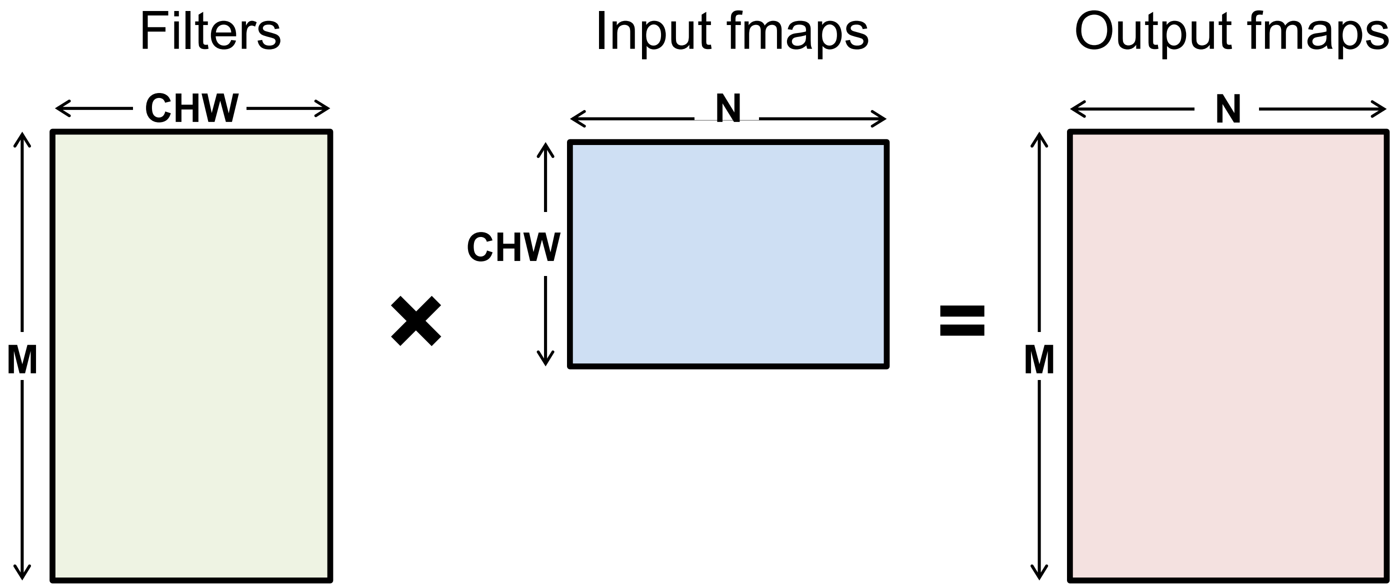
FC Layer – Multichannel



Fully-Connected (FC) Layer



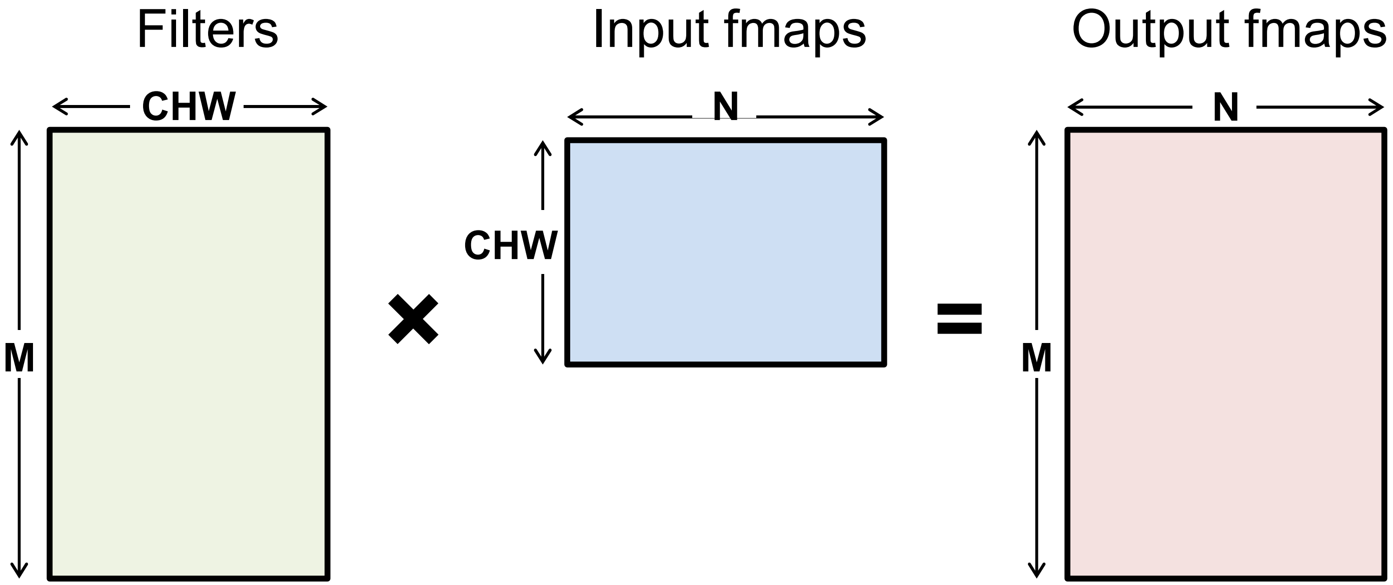
Fully-Connected (FC) Layer



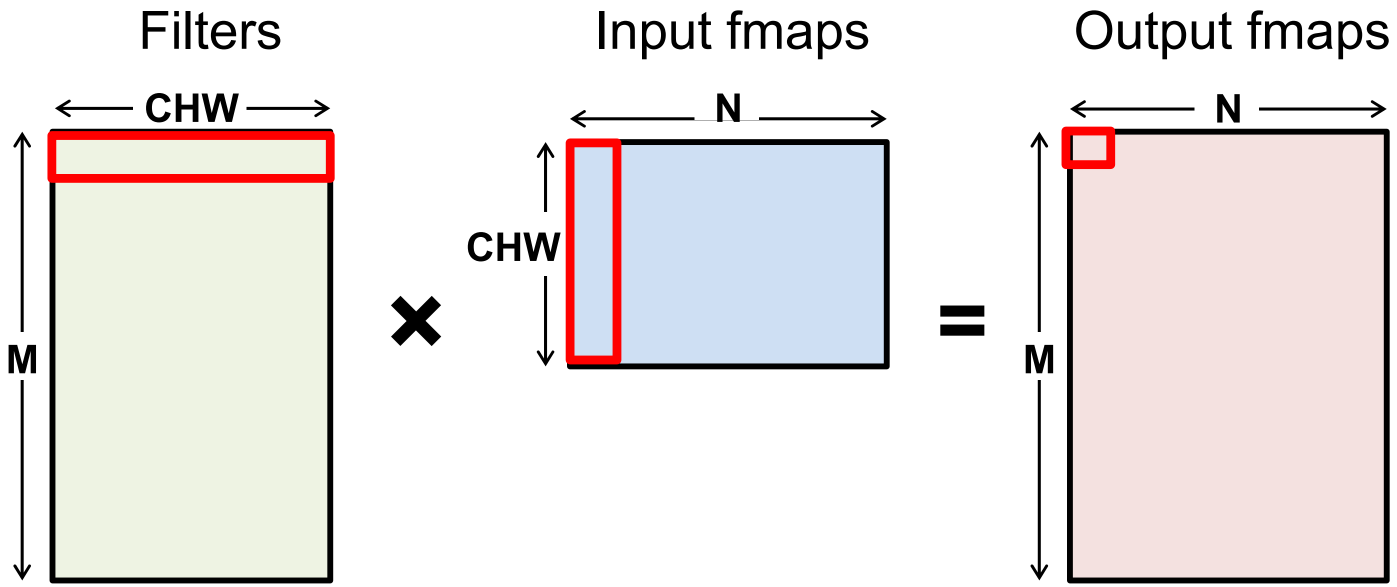
- After flattening, having a batch size of N turns the matrix-vector operation into a matrix-matrix multiply

FC Einsum Notation

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

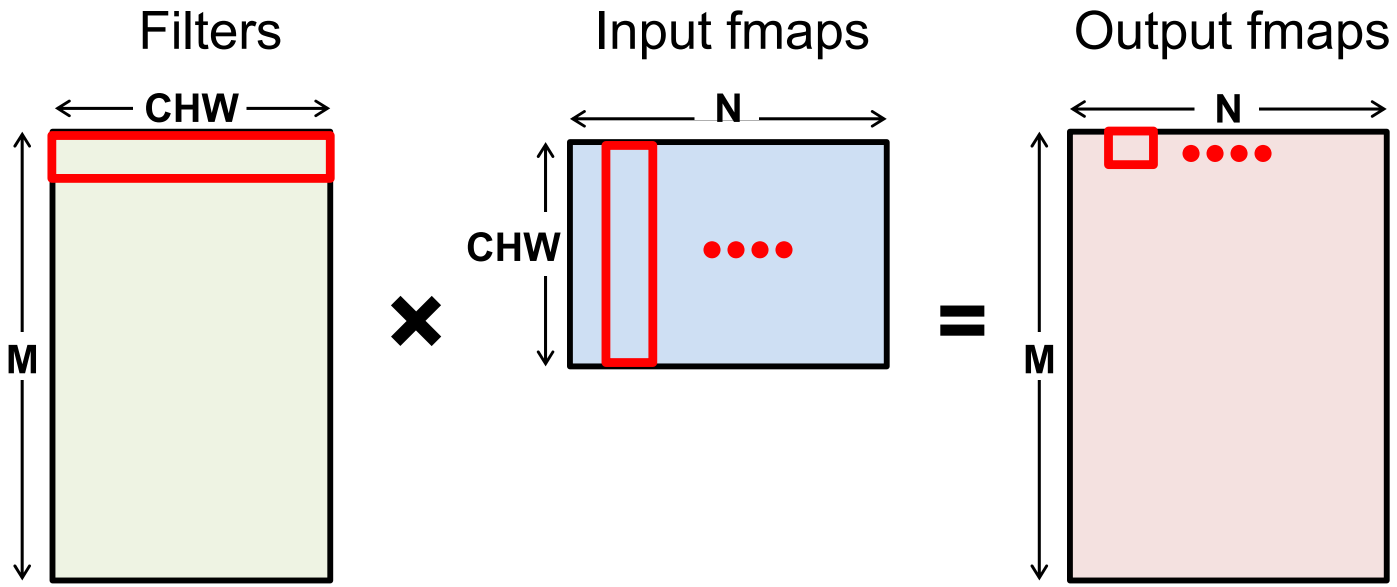


Fully-Connected (FC) Layer



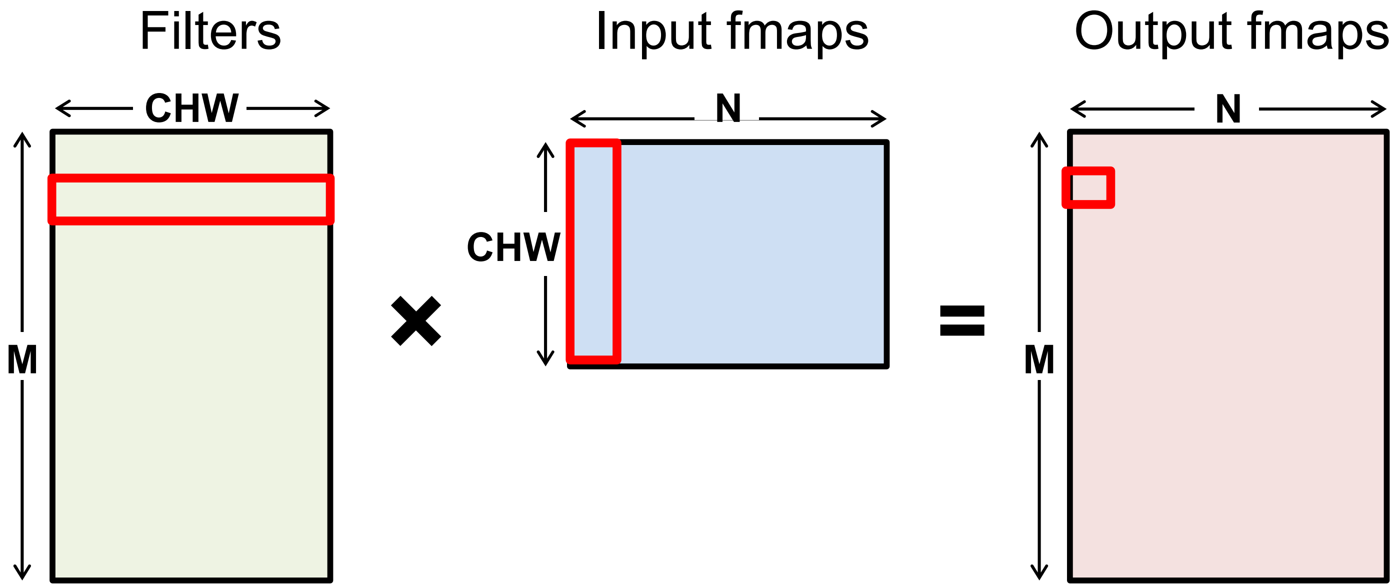
- After flattening, having a batch size of N turns the matrix-vector operation into a matrix-matrix multiply

Fully-Connected (FC) Layer



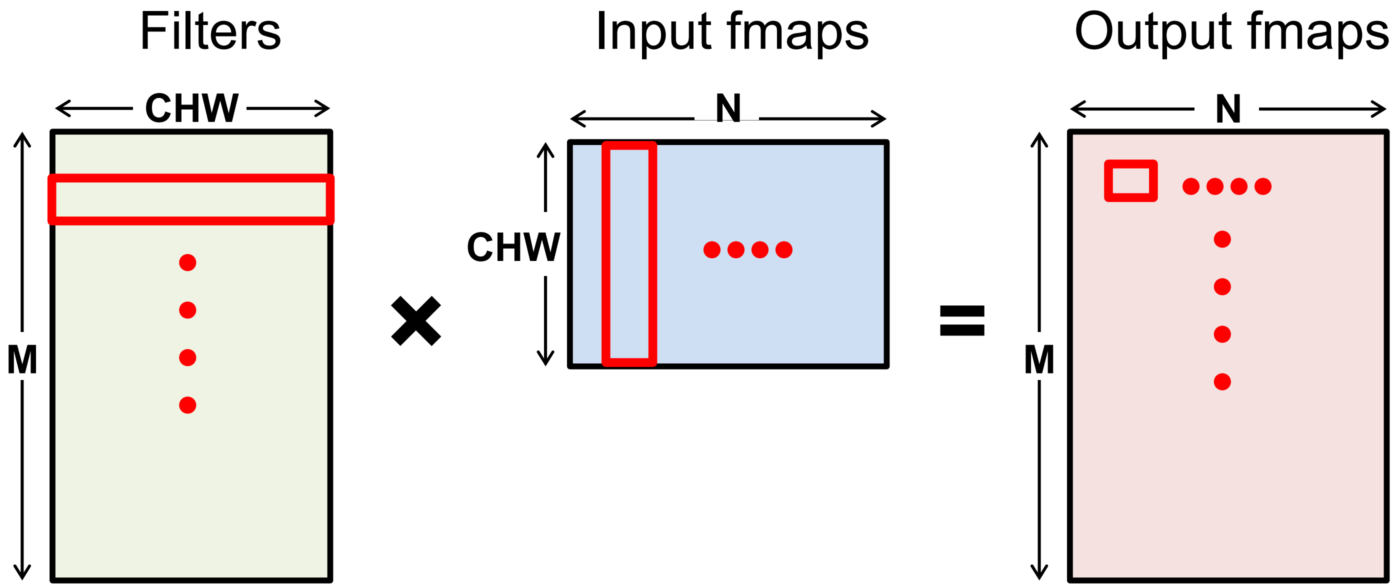
- After flattening, having a batch size of N turns the matrix-vector operation into a matrix-matrix multiply

Fully-Connected (FC) Layer



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

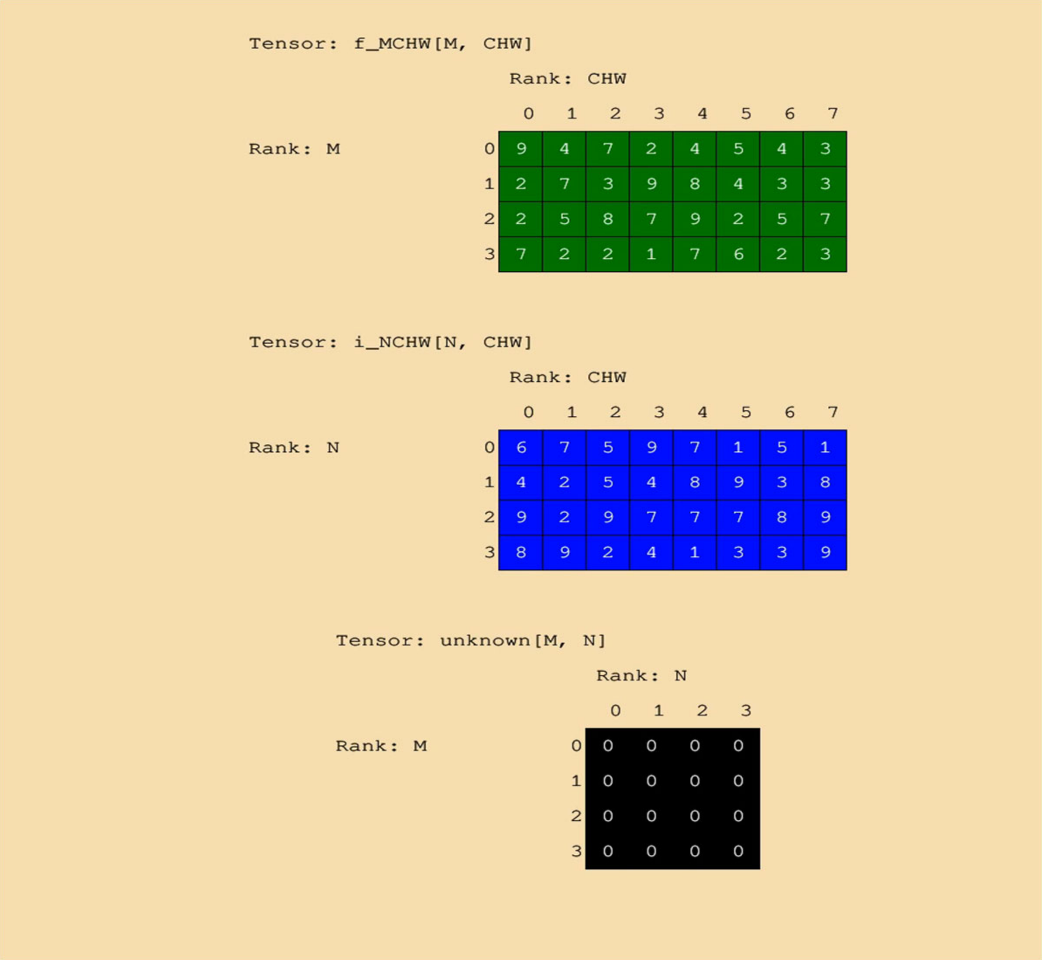
Fully-Connected (FC) Layer



- 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

Rank: M

	0	1	2	3	4	5	6	7
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

Rank: N

	0	1	2	3	4	5	6	7
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

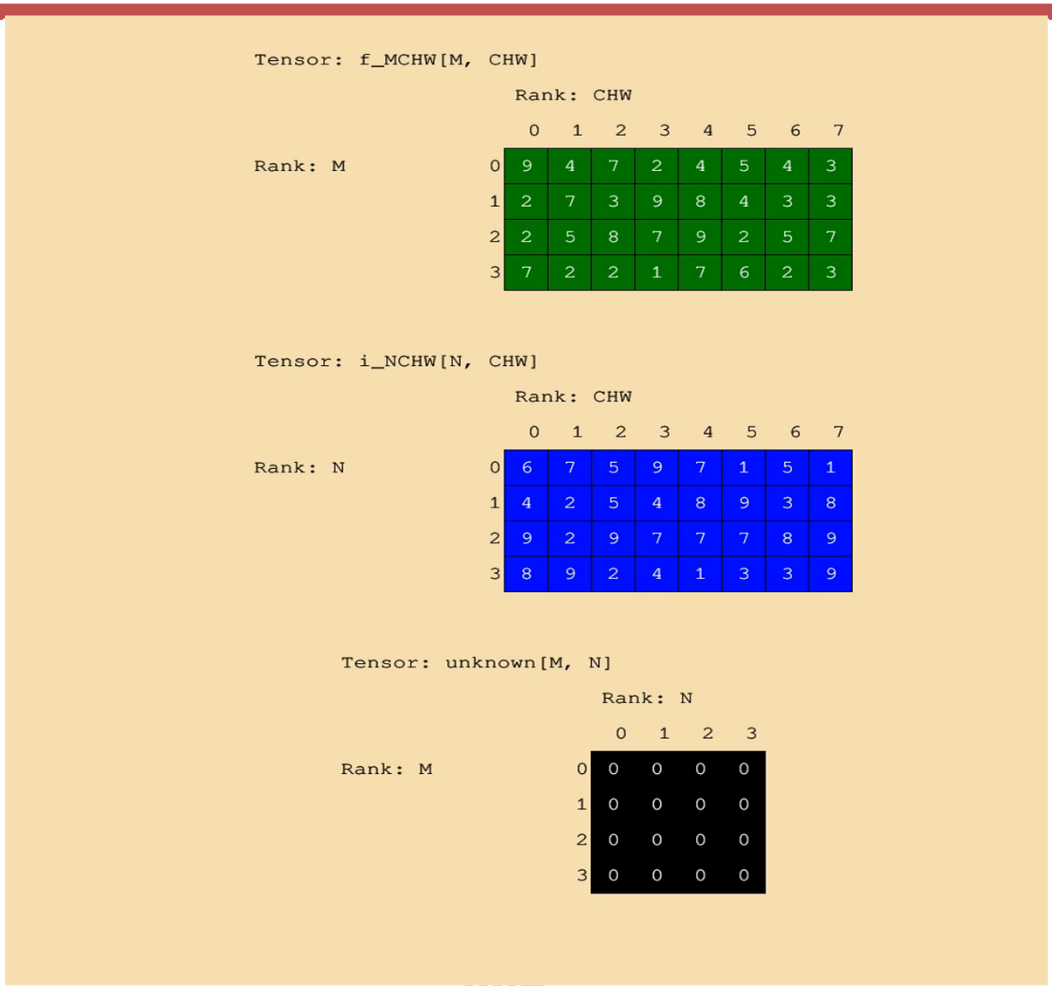
Tensor: unknown[M, N]

Rank: N

Rank: M

	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0

Matrix-Matrix Multiply Tiled



Tensor: f_MCHW[M, CHW]

Rank: CHW

Rank: M

	0	1	2	3	4	5	6	7
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

Rank: N

	0	1	2	3	4	5	6	7
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

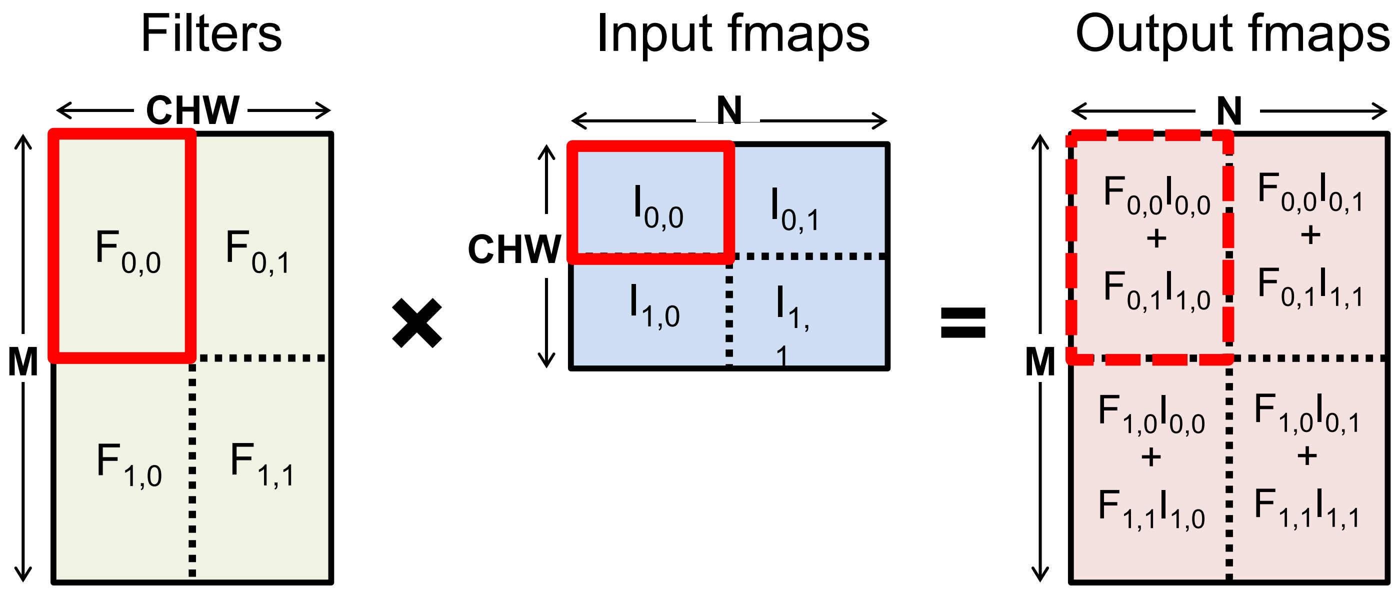
Tensor: unknown[M, N]

Rank: N

Rank: M

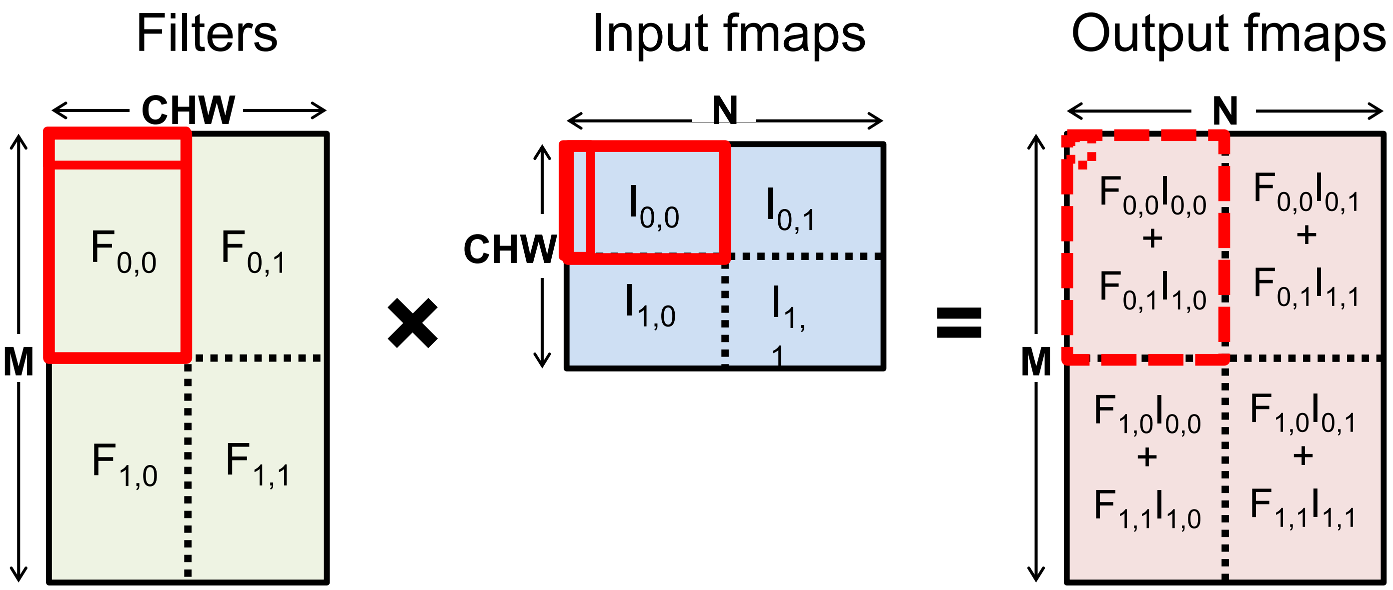
	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0

Tiled Fully-Connected (FC) Layer


Filters
CHW
M
$F_{0,0}$
$F_{0,1}$
$F_{1,0}$
$F_{1,1}$
×
Input fmaps
N
CHW
$I_{0,0}$
$I_{0,1}$
$I_{1,0}$
$I_{1,1}$
=
Output fmaps
N
M
$F_{0,0}I_{0,0} + F_{0,1}I_{1,0}$
$F_{0,0}I_{0,1} + F_{0,1}I_{1,1}$
$F_{1,0}I_{0,0} + F_{1,1}I_{1,0}$
$F_{1,0}I_{0,1} + F_{1,1}I_{1,1}$

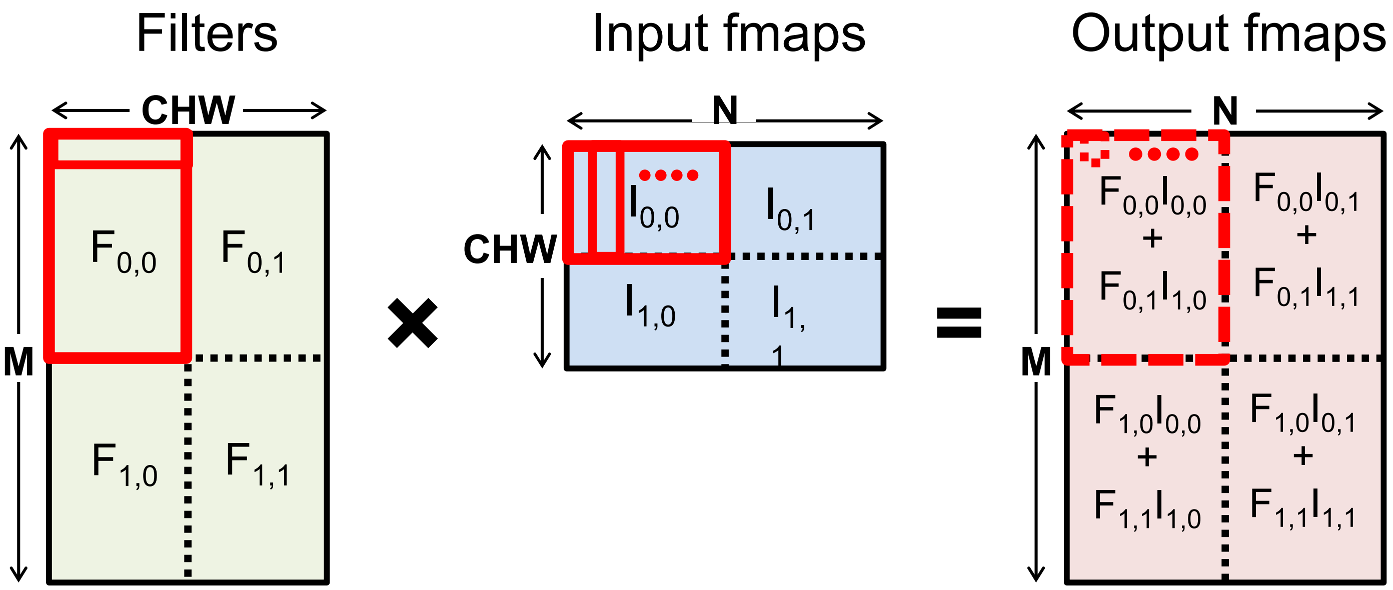
Matrix multiply tiled to fit in cache
and computation ordered to maximize reuse of data in cache

Tiled Fully-Connected (FC) Layer



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

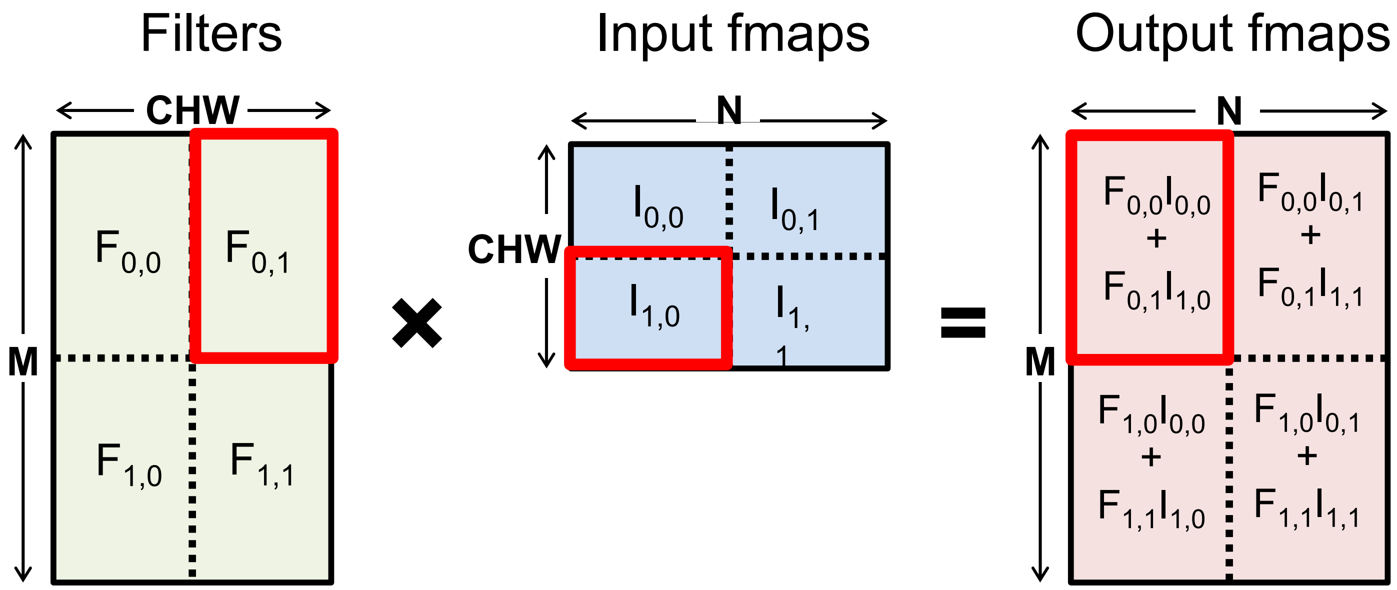
Tiled Fully-Connected (FC) Layer


Filters
CHW
M
$F_{0,0}$
$F_{0,1}$
$F_{1,0}$
$F_{1,1}$
×
Input fmaps
N
CHW
$I_{0,0}$
$I_{0,1}$
$I_{1,0}$
$I_{1,1}$
=
Output fmaps
N
M
$F_{0,0}I_{0,0} + F_{0,1}I_{1,0}$
$F_{0,0}I_{0,1} + F_{0,1}I_{1,1}$
$F_{1,0}I_{0,0} + F_{1,1}I_{1,0}$
$F_{1,0}I_{0,1} + F_{1,1}I_{1,1}$

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


Filters
CHW
M
$F_{0,0}$
$F_{0,1}$
$F_{1,0}$
$F_{1,1}$
×
Input fmaps
N
CHW
$I_{0,0}$
$I_{0,1}$
$I_{1,0}$
$I_{1,1}$
=
Output fmaps
N
M
$F_{0,0}I_{0,0} + F_{0,1}I_{1,0}$
$F_{0,0}I_{0,1} + F_{0,1}I_{1,1}$
$F_{1,0}I_{0,0} + F_{1,1}I_{1,0}$
$F_{1,0}I_{0,1} + F_{1,1}I_{1,1}$

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

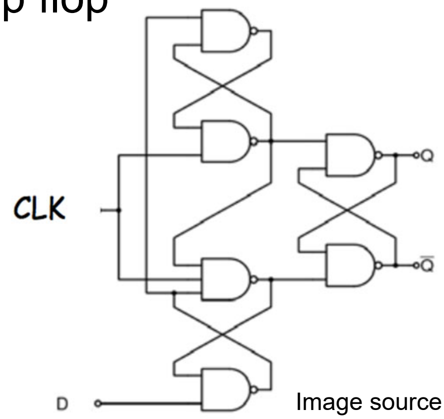
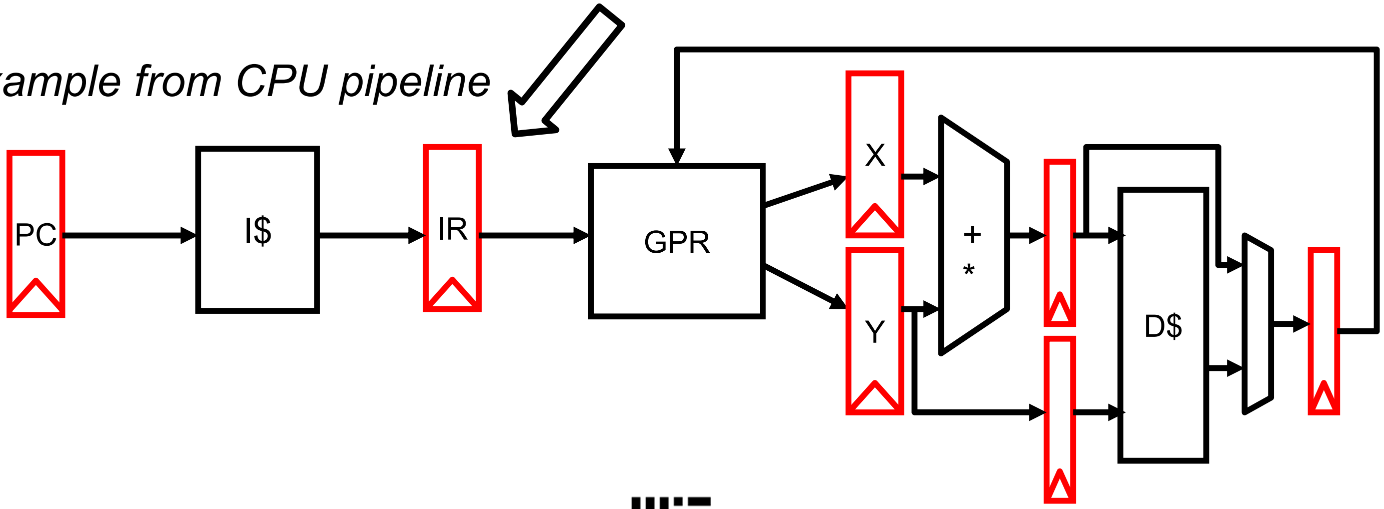


Image source: 6.111

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

D-flip flop

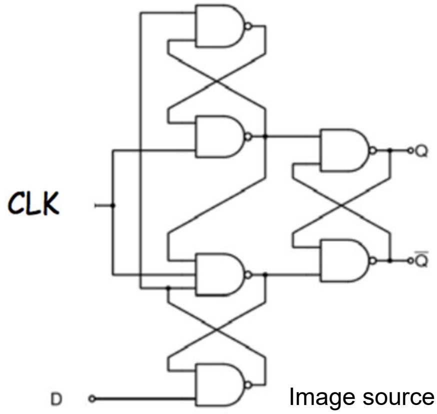
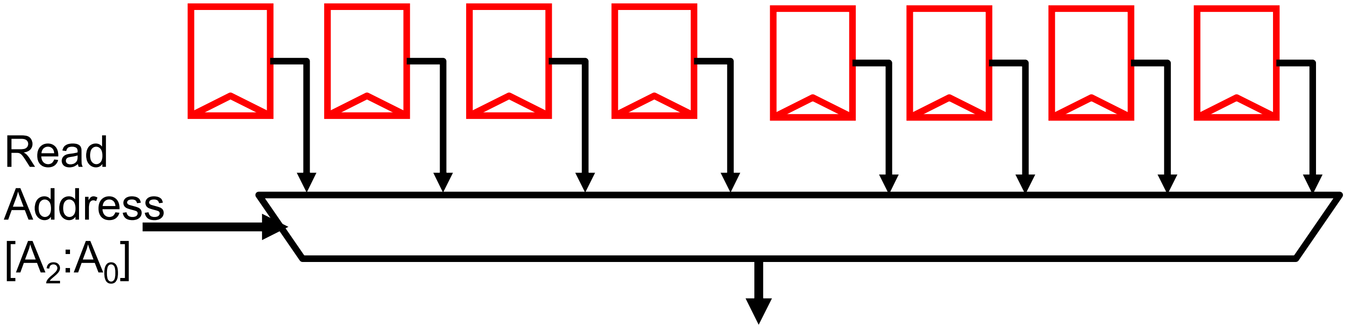


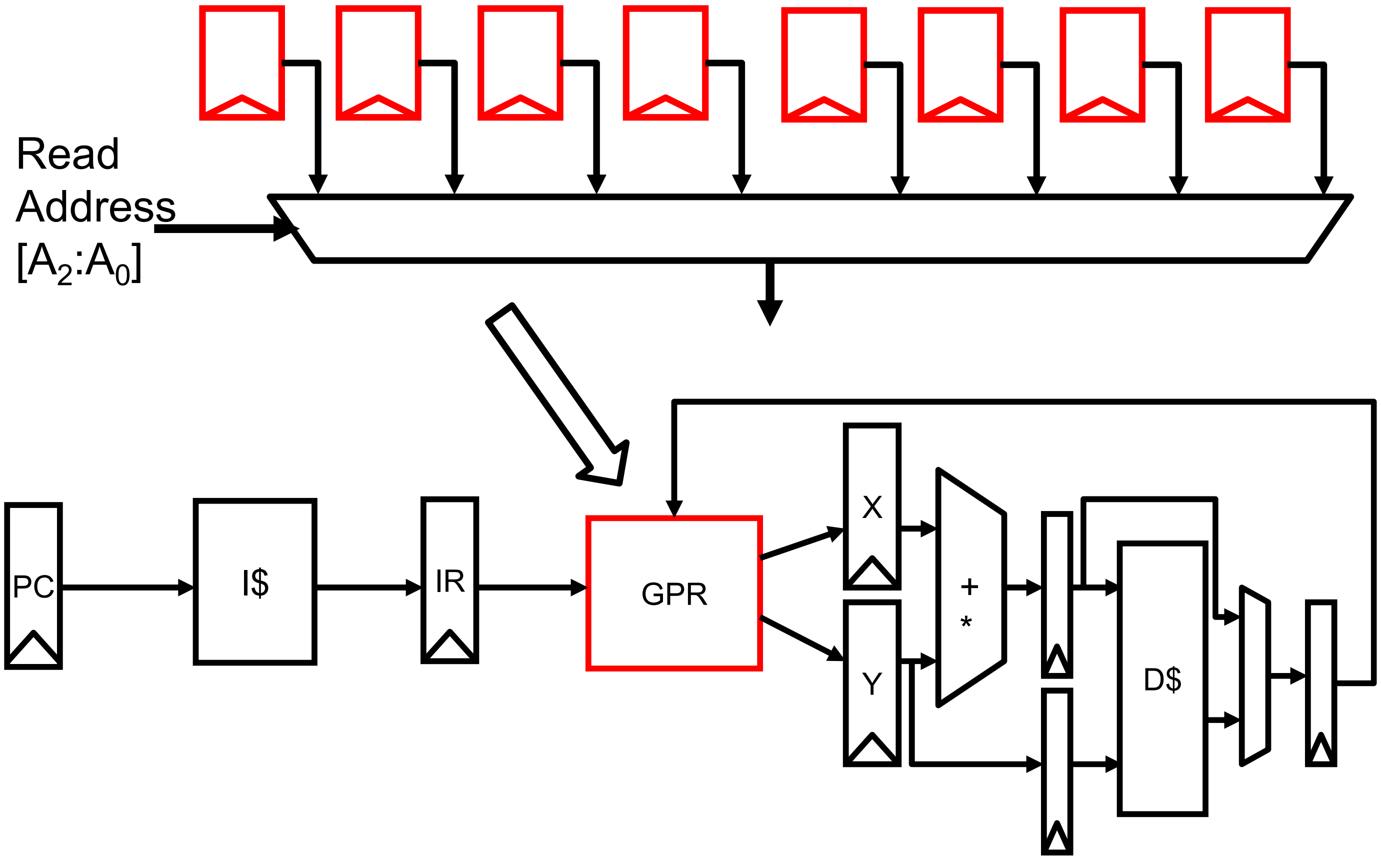
Image source: 6.111

Array of Flip flops



Latches/Flip Flops (< 0.5 kB)

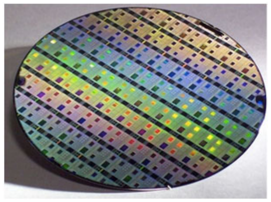
Array of Flip flops



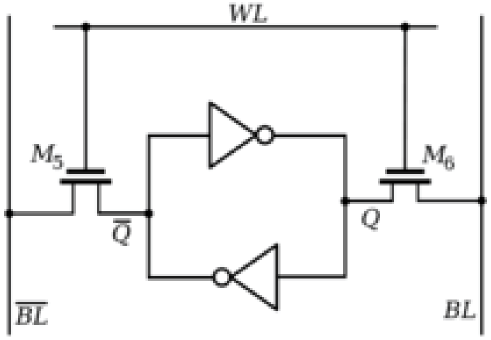
Read Address $[A_2:A_0]$

SRAM

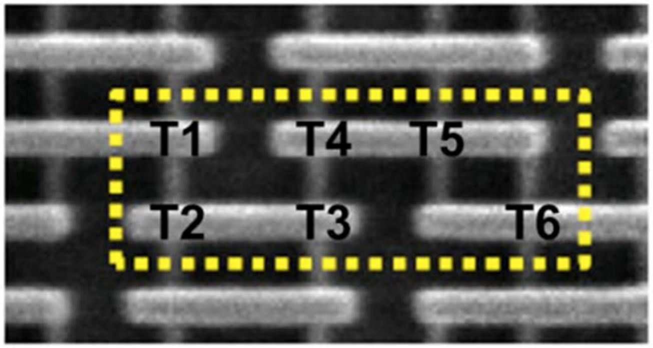
IC wafer



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

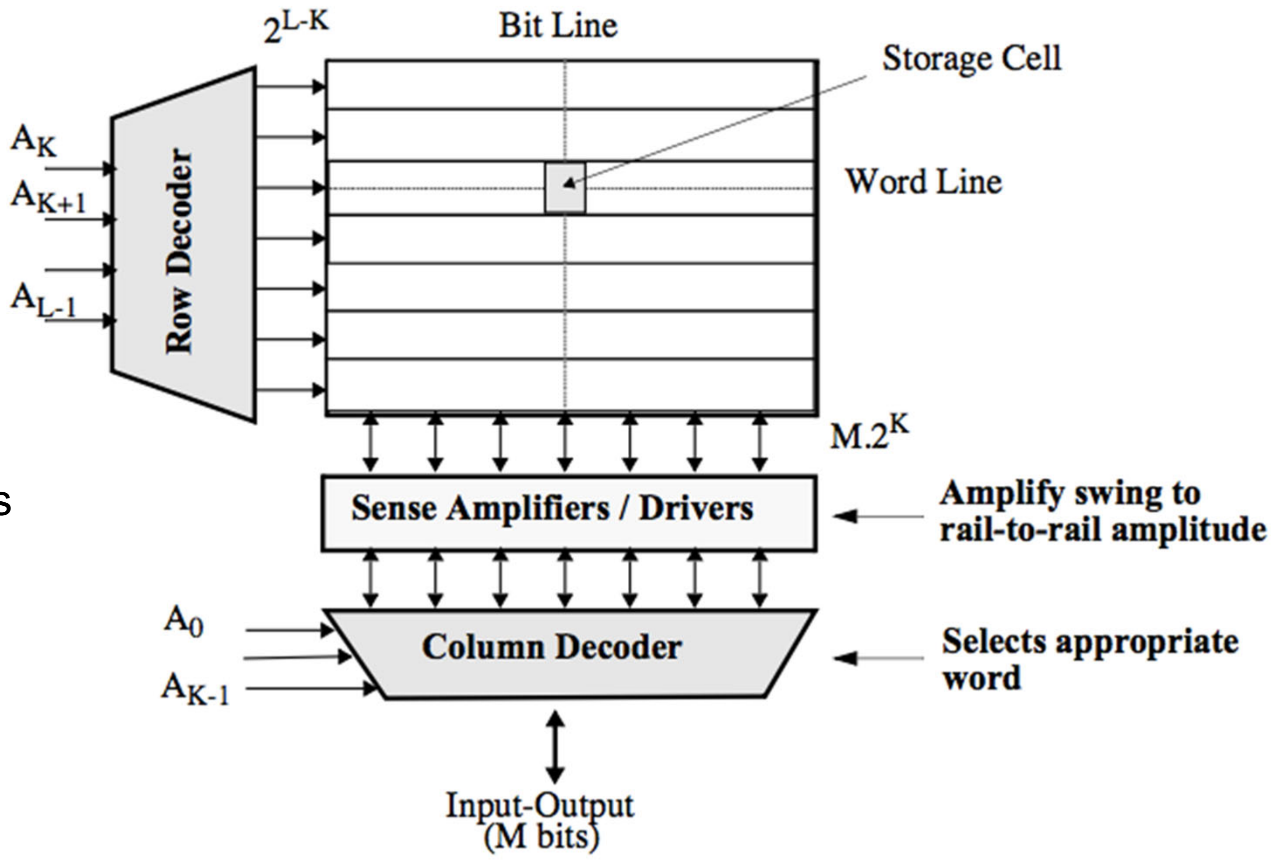


Bit cell size 0.75um^2 in 14nm

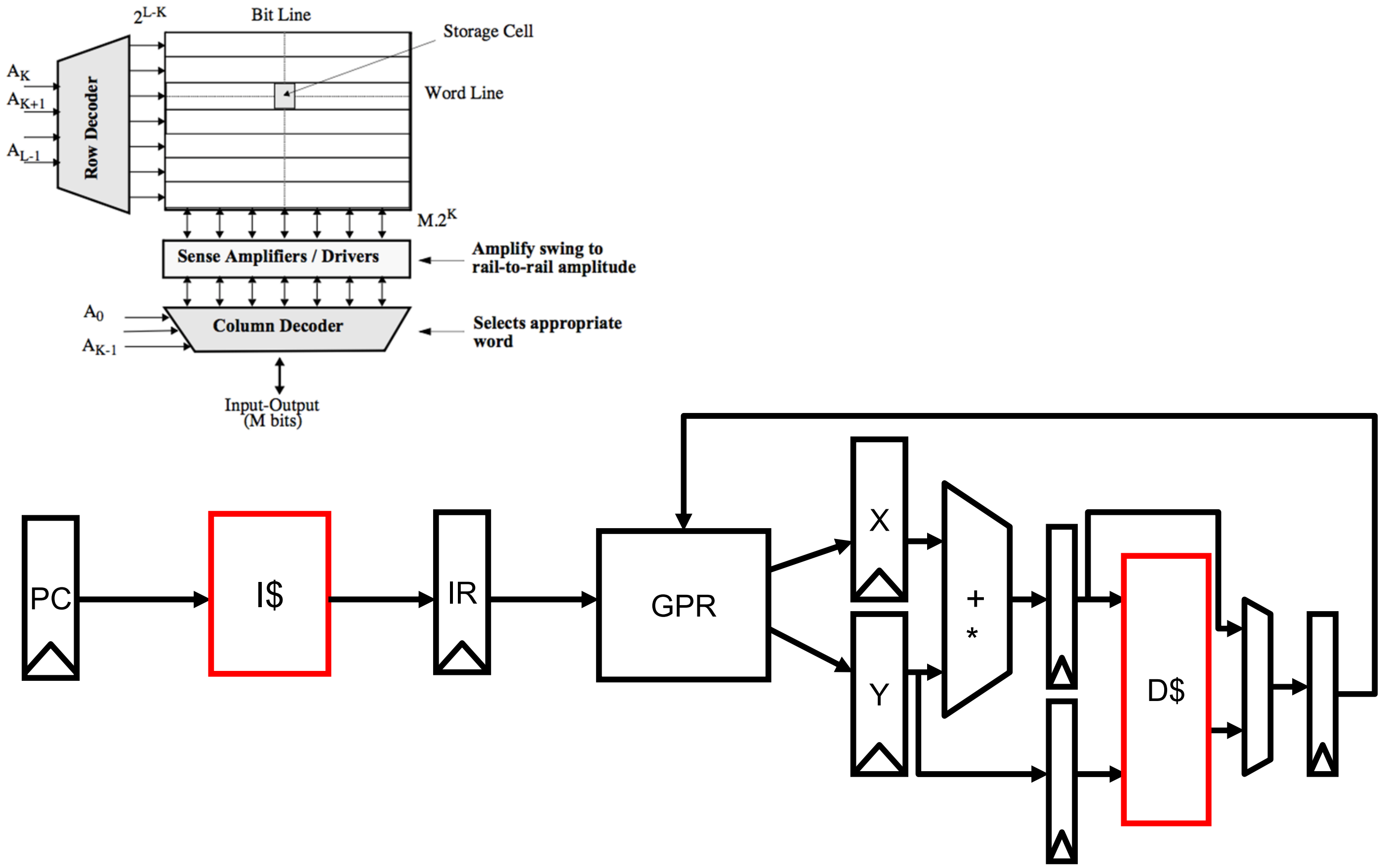


SRAM (kB – MB)

Address
$[A_k:A_0]$

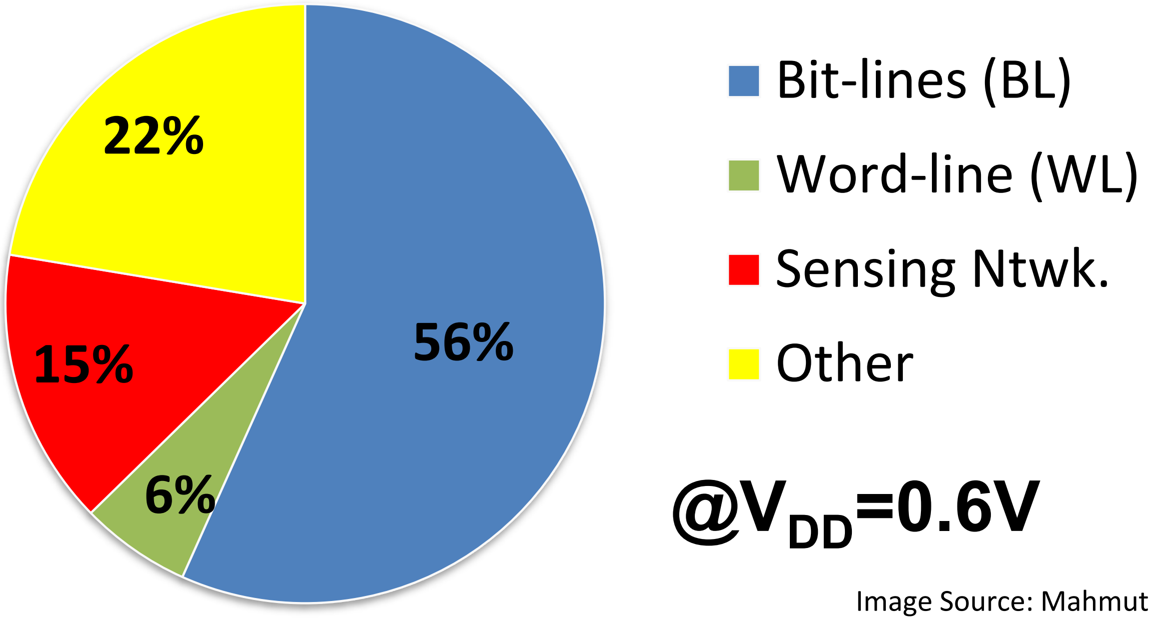


SRAM


2^{L-K}
Bit Line
Storage Cell
Word Line
A_K
A_{K+1}
A_{L-1}
Row Decoder
M.2^K
Sense Amplifiers / Drivers
Amplify swing to rail-to-rail amplitude
A_0
A_{K-1}
Column Decoder
Selects appropriate word
Input-Output (M bits)
PC
I$
IR
GPR
X
Y
+
*
D$

SRAM Power Dominated by Bit Line

Measured SRAM Power Breakdown



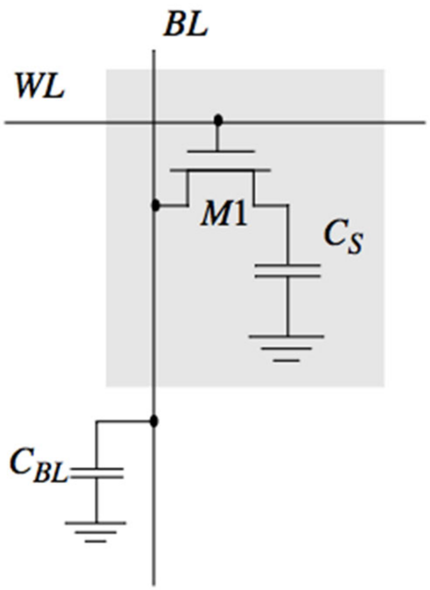
@V_{DD}=0.6V

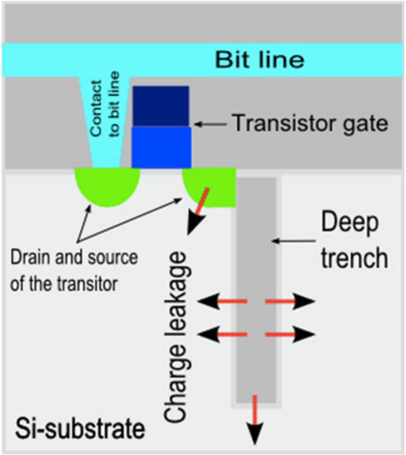
Image Source: Mahmut Sinangil

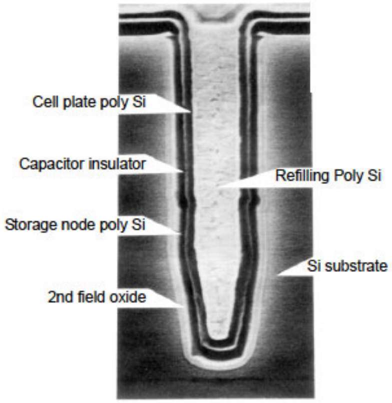
Larger array → Longer bit-lines
→ Higher capacitance → Higher power

DRAM

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

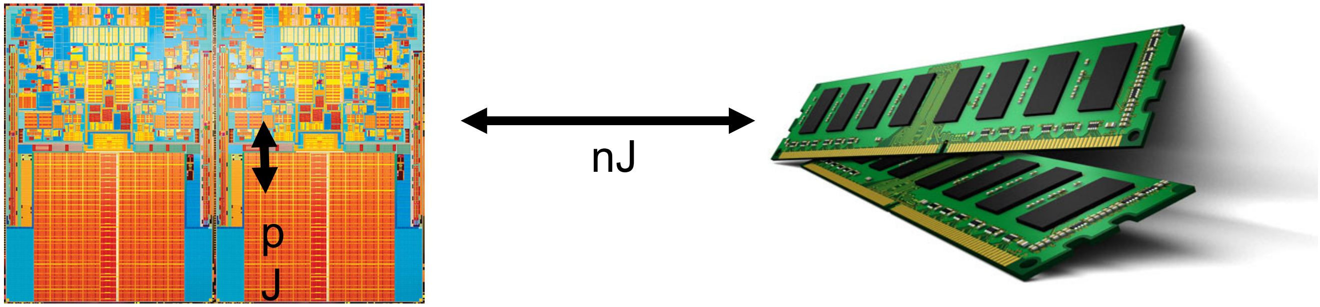






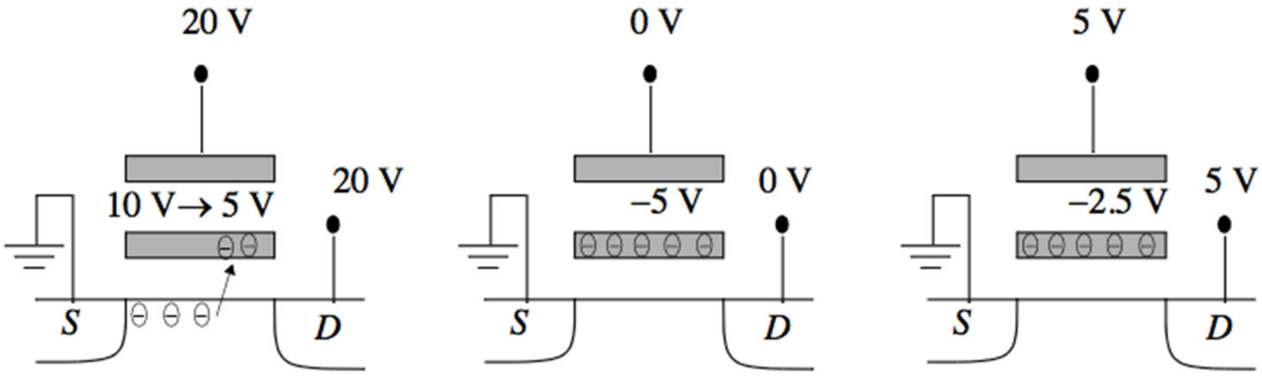
DRAM (GB)

- 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



Flash (100GB to TB)

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

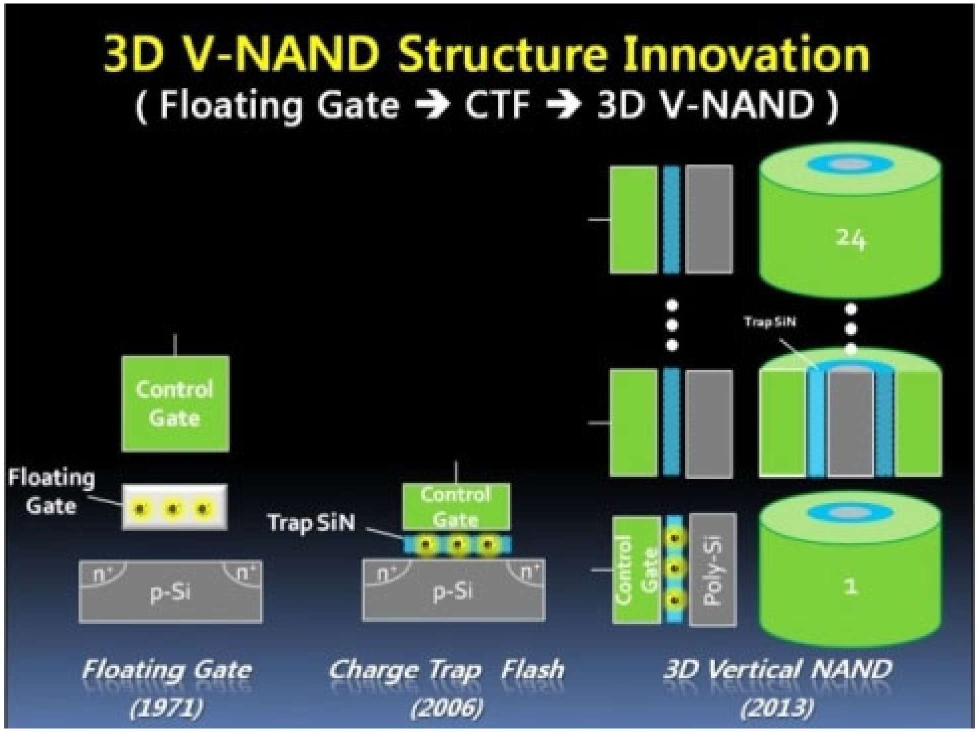


(a) Avalanche injection.

(b) Removing programming voltage leaves charge trapped.

(c) Programming results in higher V_T.

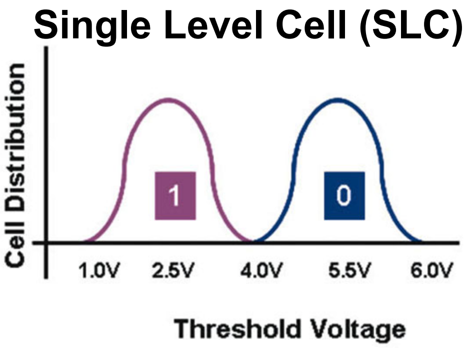
Flash Memory

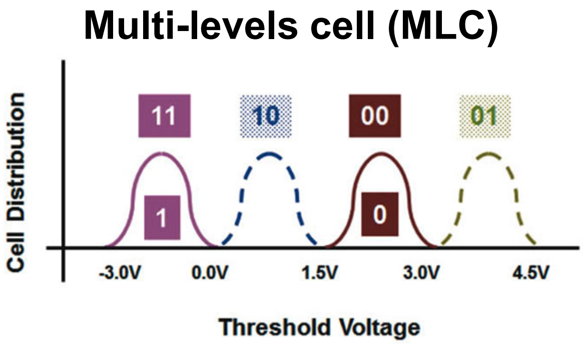


Single Level Cell (SLC)

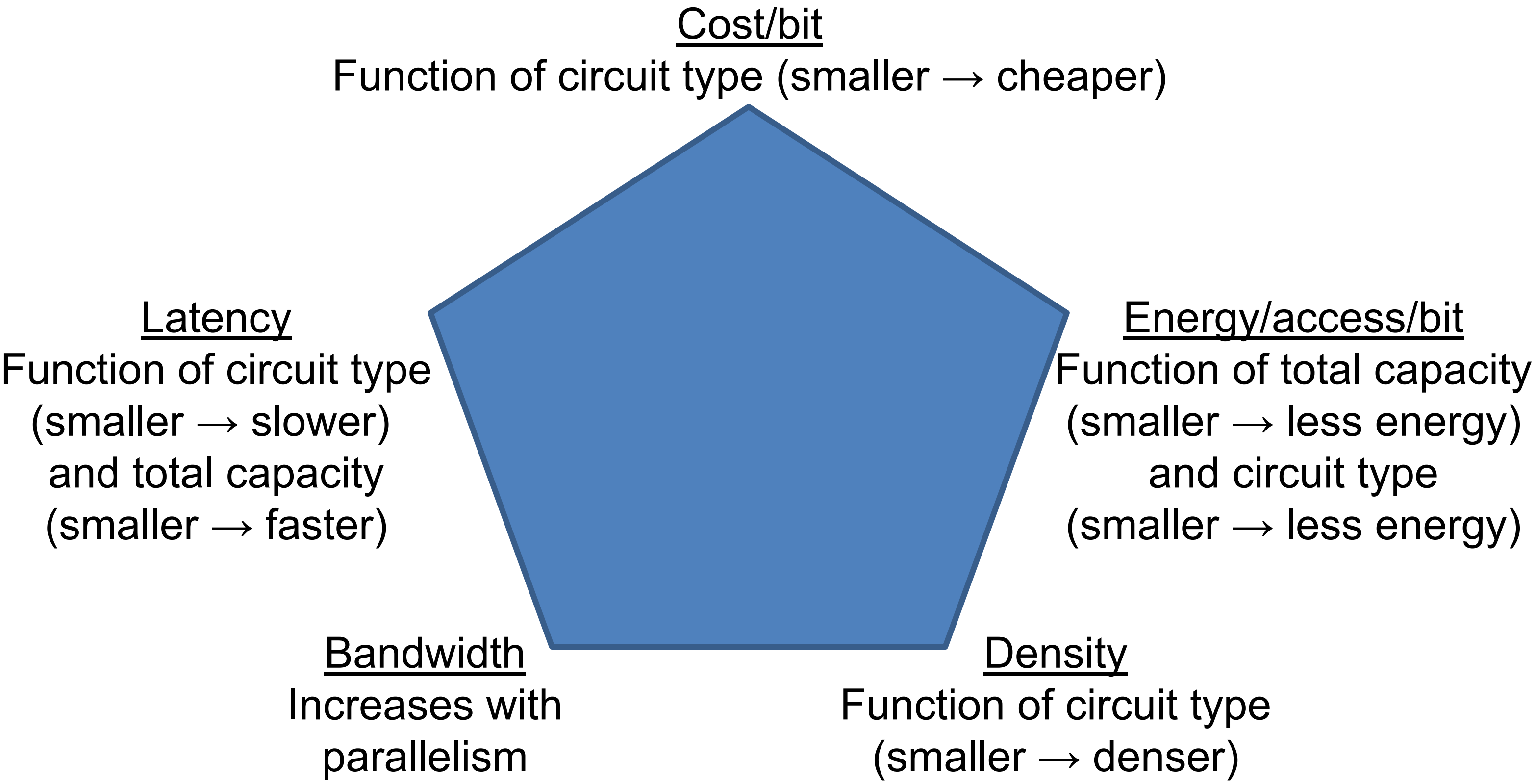
Multi-levels cell (MLC)

48 layer, Ternary level cell (TLC)
Aug 2015
256 Gb per die (for SSD)





Memory Tradeoffs



Cost/bit
Function of circuit type (smaller → cheaper)

Latency
Function of circuit type
(smaller → slower)
and total capacity
(smaller → faster)

Energy/access/bit
Function of total capacity
(smaller → less energy)
and circuit type
(smaller → less energy)

Bandwidth
Increases with
parallelism

Density
Function of circuit type
(smaller → denser)

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!