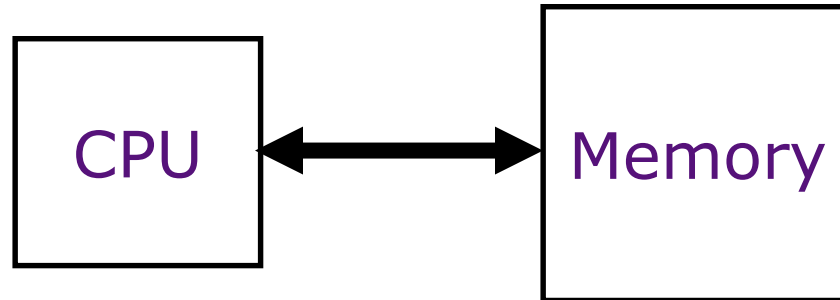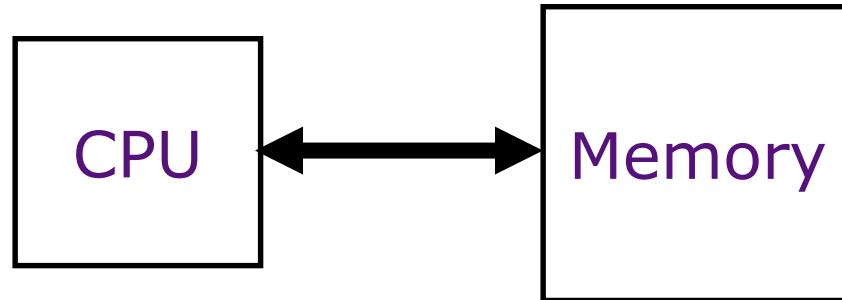# Cache Organization

Daniel Sanchez
Computer Science and Artificial Intelligence Laboratory
M.I.T.

CPU-Memory Bottleneck

CPU-Memory Bottleneck

Performance of high-speed computers is usually limited by memory *bandwidth* & *latency*

- Latency (time for a single access)
    Memory access time >> Processor cycle time

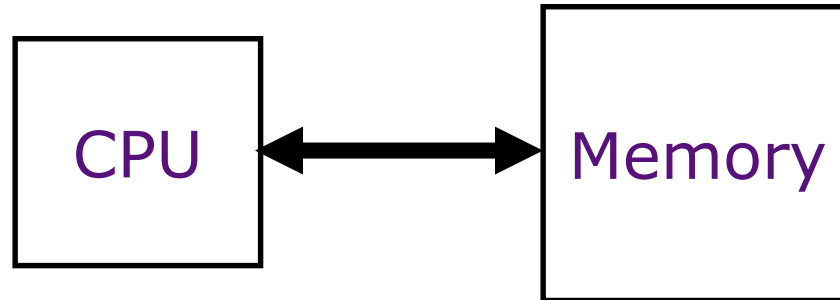- Bandwidth (number of accesses per unit time)

CPU-Memory Bottleneck

Performance of high-speed computers is usually limited by memory *bandwidth* & *latency*

- Latency (time for a single access)
    Memory access time >> Processor cycle time

- Bandwidth (number of accesses per unit time)
    if fraction m of instructions access memory,
        $\Rightarrow 1+m$ memory references / instruction
        $\Rightarrow$ CPI = 1 requires $1+m$ memory refs / cycle

Memory Technology

- Early machines used a variety of memory technologies
  - Manchester Mark I used CRT Memory Storage
  - EDVAC used a mercury delay line

- Core memory was first large scale reliable main memory
  - Invented by Forrester in late 40s at MIT for Whirlwind project
  - Bits stored as magnetization polarity on small ferrite cores threaded onto 2 dimensional grid of wires

Memory Technology

- Early machines used a variety of memory technologies
  - Manchester Mark I used CRT Memory Storage
  - EDVAC used a mercury delay line

- Core memory was first large scale reliable main memory
  - Invented by Forrester in late 40s at MIT for Whirlwind project
  - Bits stored as magnetization polarity on small ferrite cores threaded onto 2 dimensional grid of wires

- First commercial DRAM was Intel 1103
  - 1Kbit of storage on single chip
  - charge on a capacitor used to hold value

- Semiconductor memory quickly replaced core in 1970s
 - Intel formed to exploit market for semiconductor memory

Memory Technology

- Early machines used a variety of memory technologies
  - Manchester Mark I used CRT Memory Storage
  - EDVAC used a mercury delay line

- Core memory was first large scale reliable main memory
  - Invented by Forrester in late 40s at MIT for Whirlwind project
  - Bits stored as magnetization polarity on small ferrite cores threaded onto 2 dimensional grid of wires

- First commercial DRAM was Intel 1103
  - 1Kbit of storage on single chip
  - charge on a capacitor used to hold value

- Semiconductor memory quickly replaced core in 1970s
  - Intel formed to exploit market for semiconductor memory

- Flash memory
  - Slower, but denser than DRAM. Also non-volatile, but with wearout issues

- Phase change memory (PCM, 3D XPoint)
  - Slightly slower, but much denser than DRAM and non-volatile

DRAM Architecture

Word Line
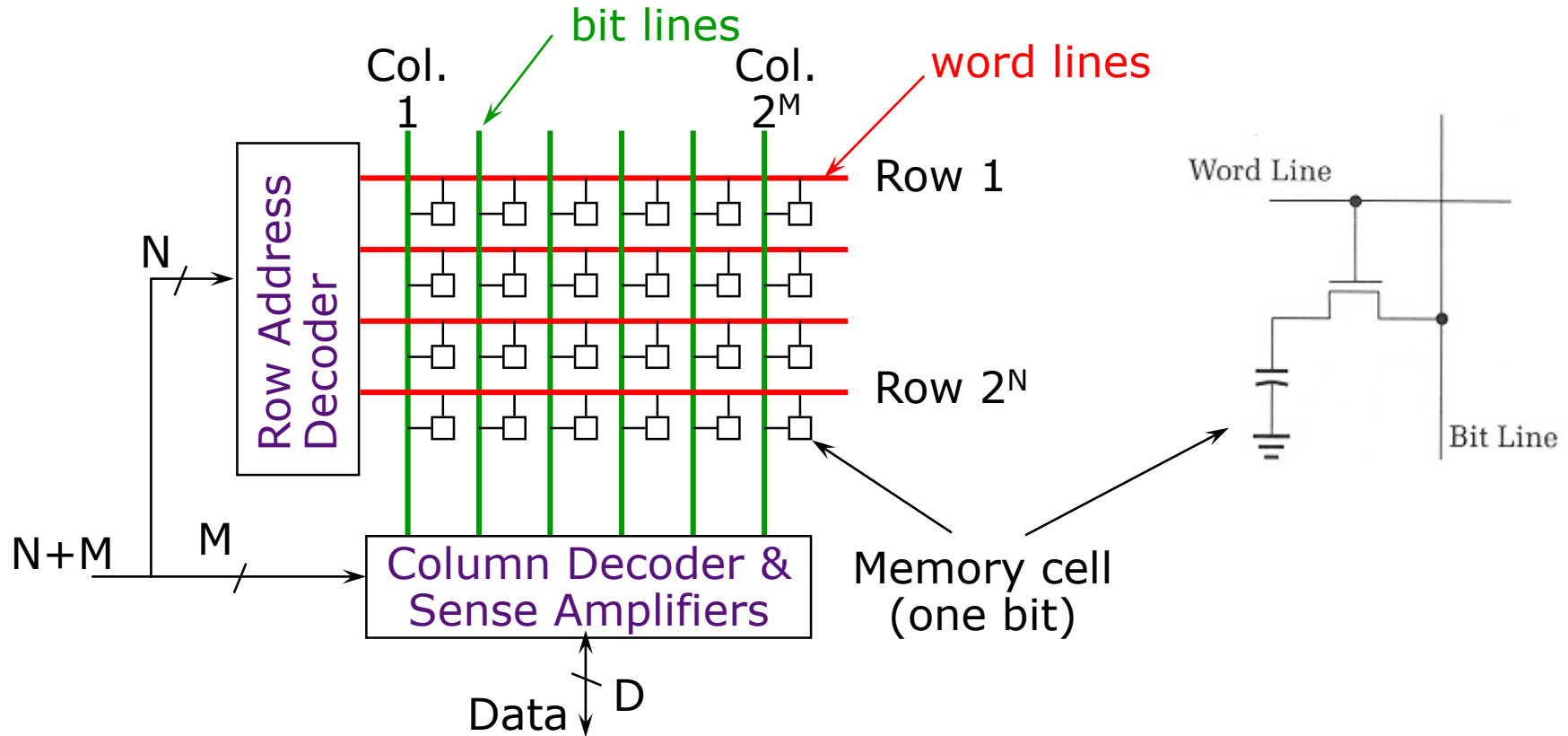
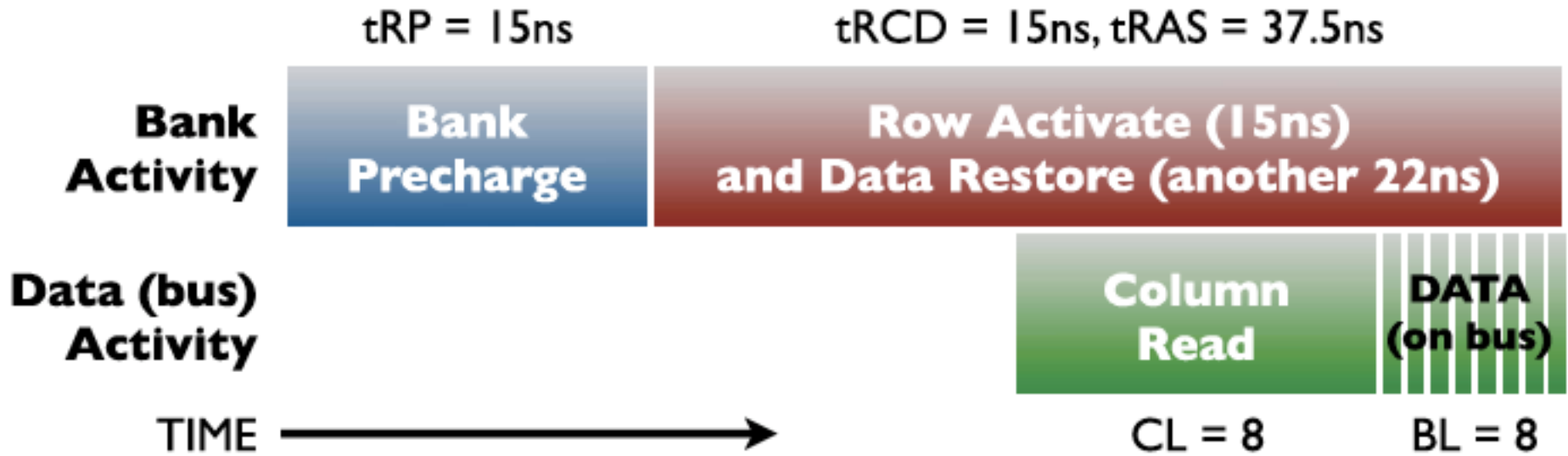Bit Line

Memory cell
(one bit)
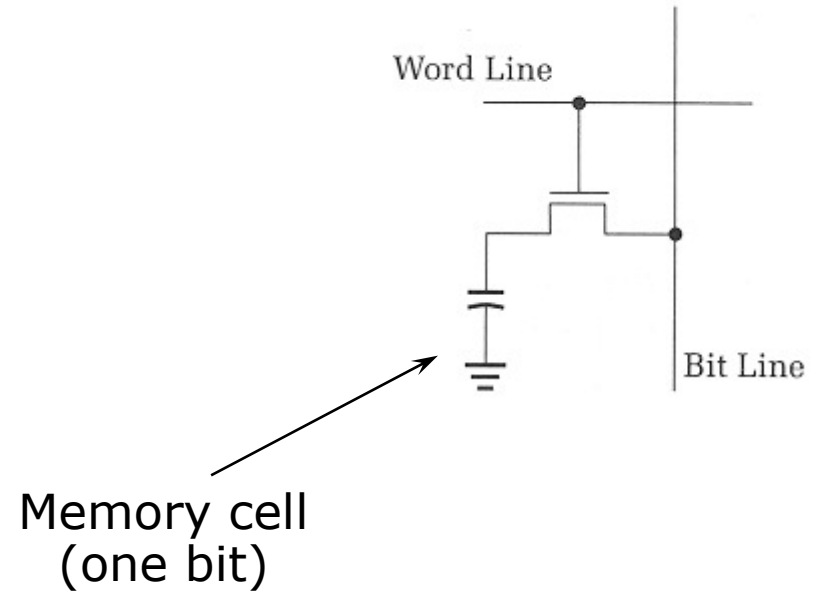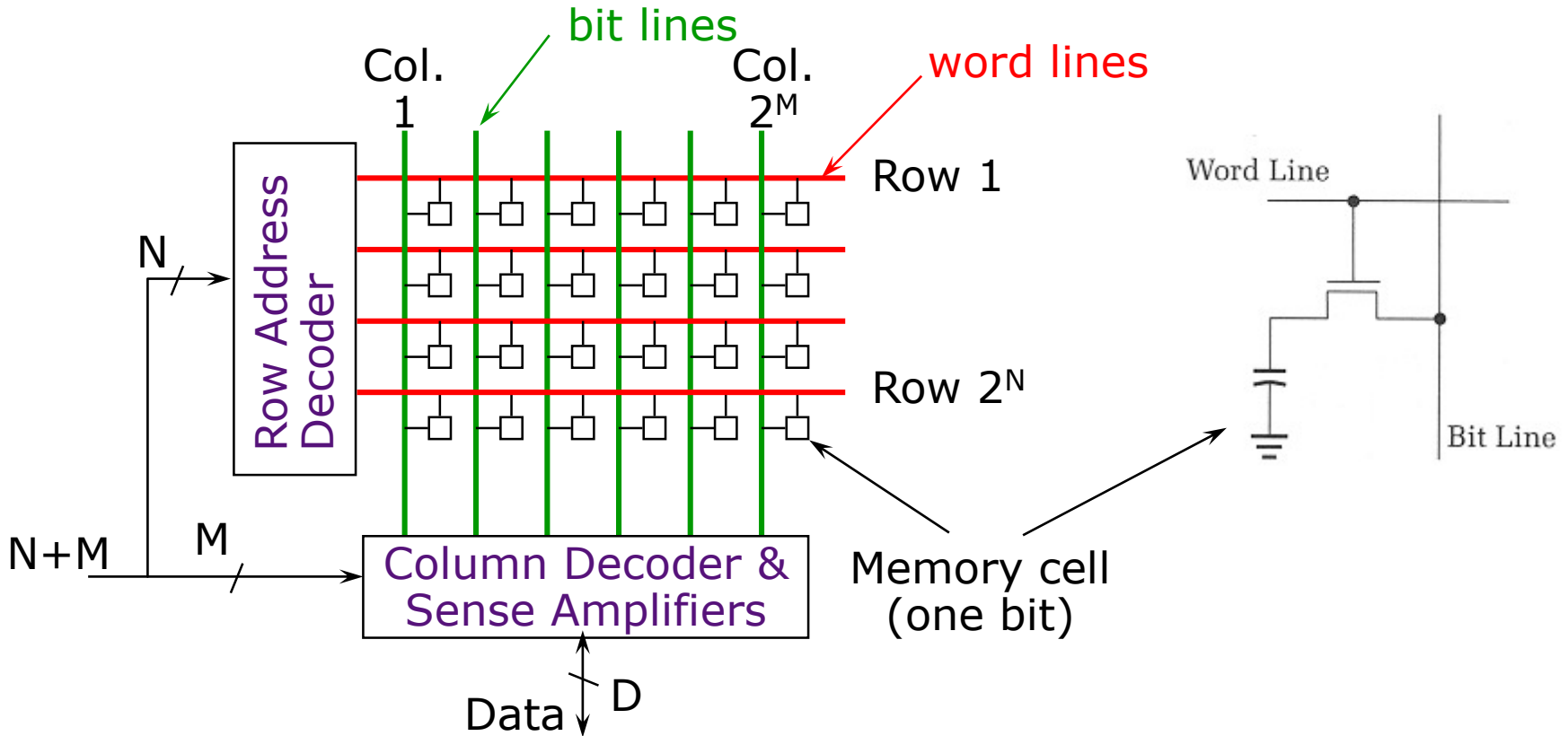
DRAM Architecture

- Bits stored in 2-dimensional arrays on chip

DRAM Architecture

- Bits stored in 2-dimensional arrays on chip
- Modern chips have around 8 logical banks on each chip
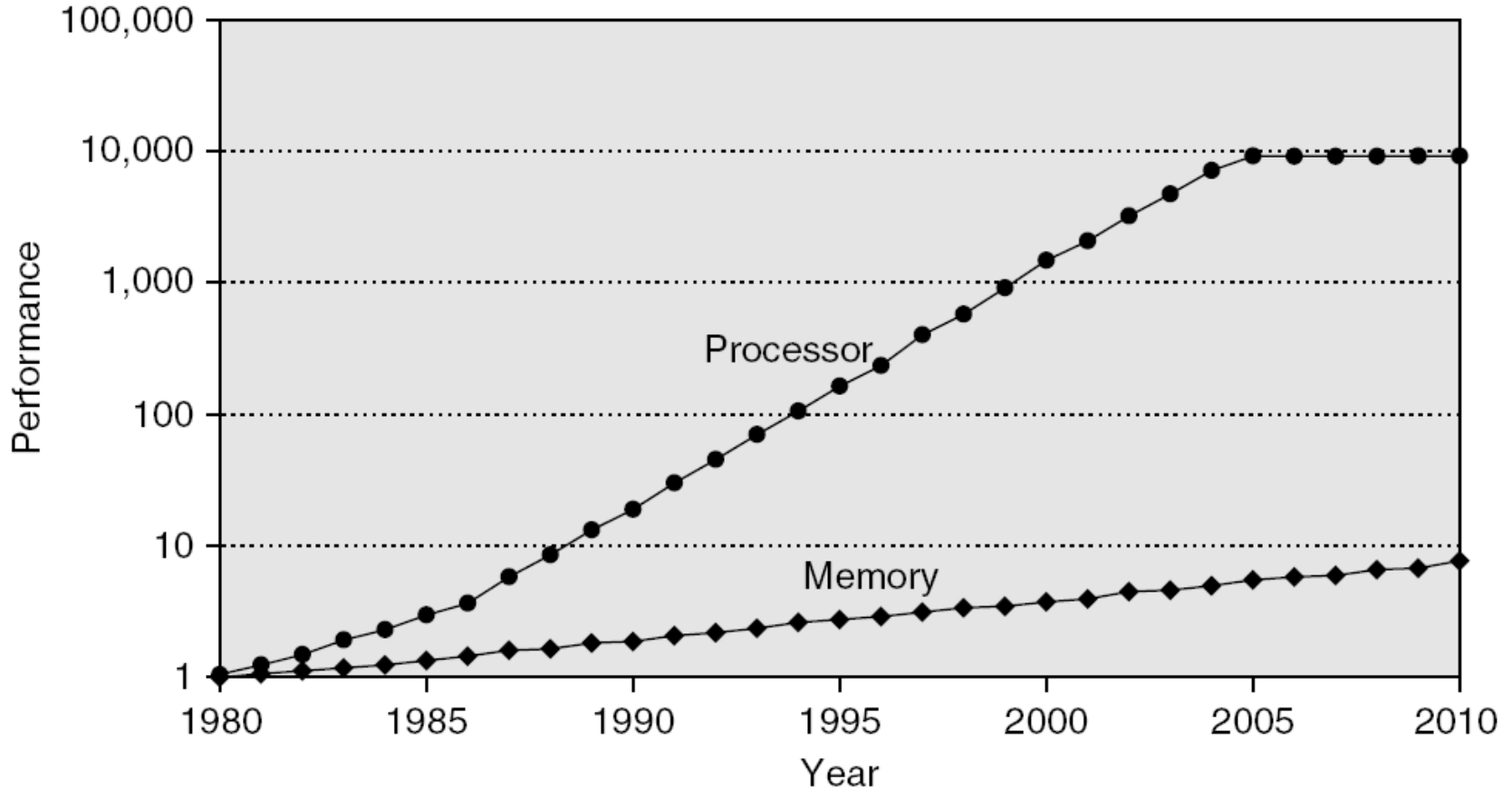 - Each logical bank physically implemented as many smaller arrays

DRAM timing

tRP = 15ns   tRCD = 15ns, tRAS = 37.5ns

Bank Activity   Bank Precharge   Row Activate (15ns) and Data Restore (another 22ns)

Data (bus) Activity   Column Read   DATA (on bus)
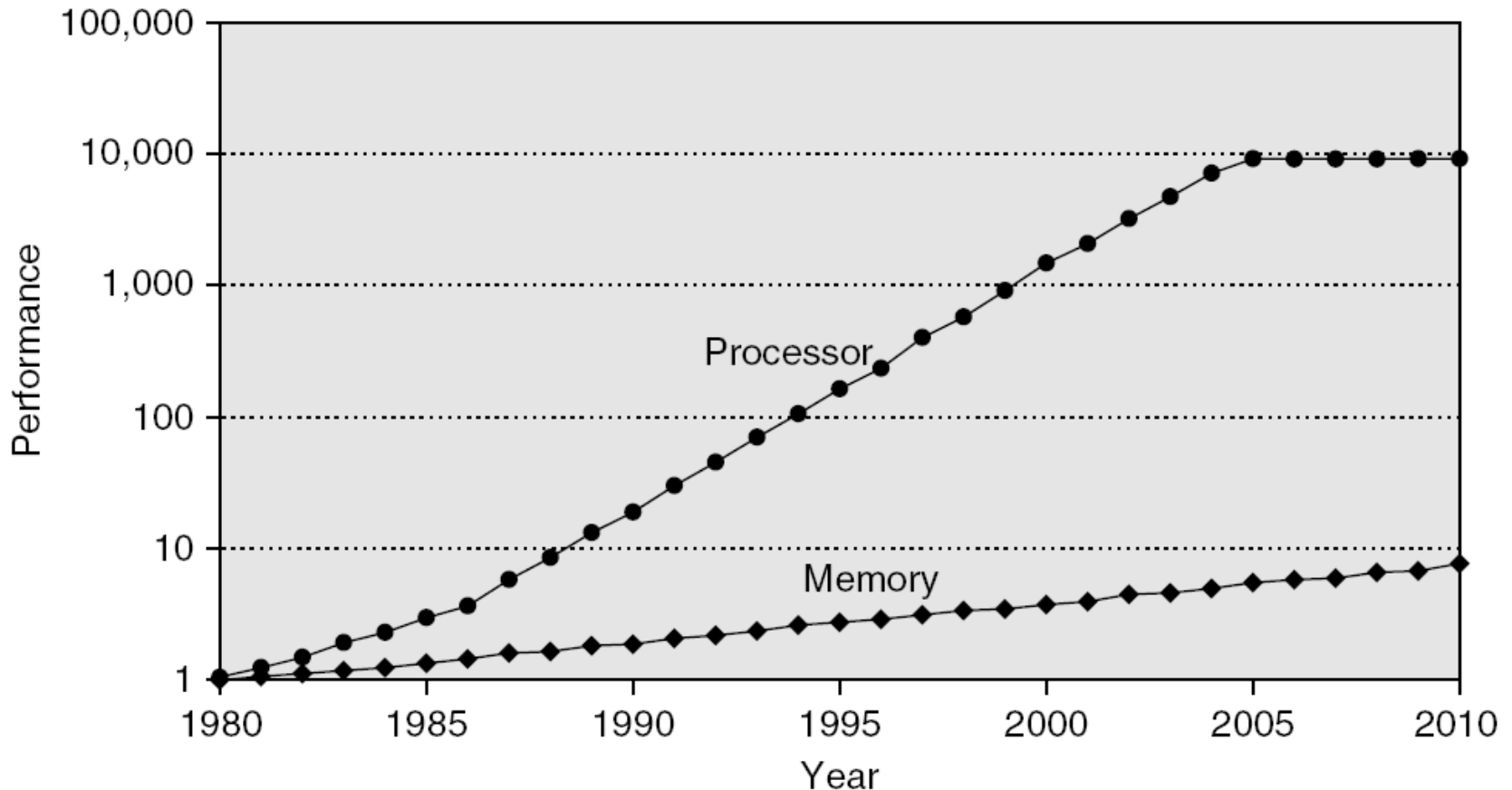
TIME   CL = 8   BL = 8

DRAM Spec:
      CL, tRCD, tRP, tRAS, e.g., 9-9-9-24
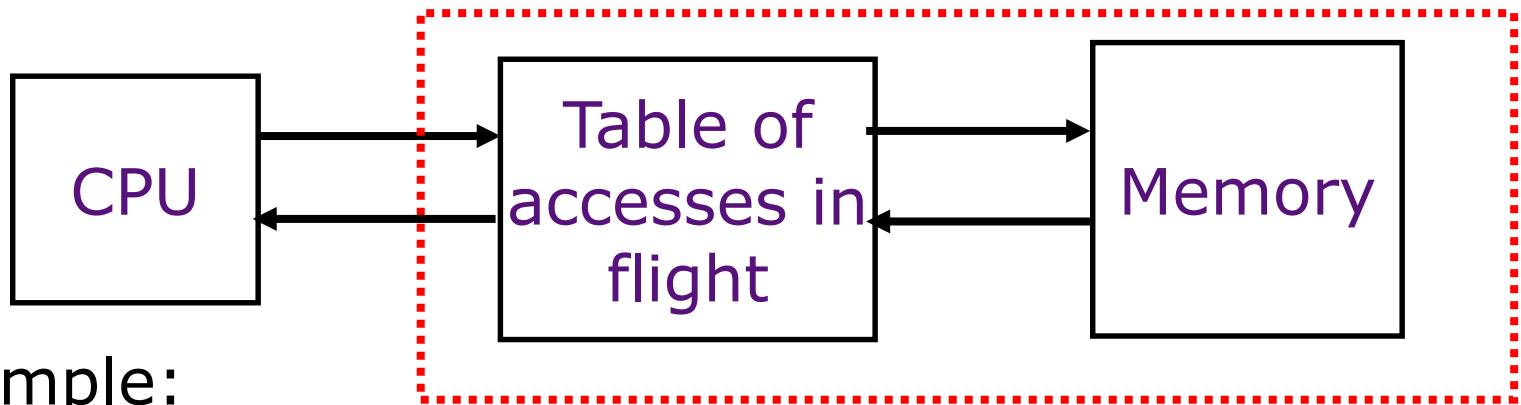
Processor-DRAM Gap (latency)

Processor-DRAM Gap (latency)

Four-issue 2GHz superscalar accessing 100ns DRAM could execute 800 instructions during time for one memory access!

Little's Law

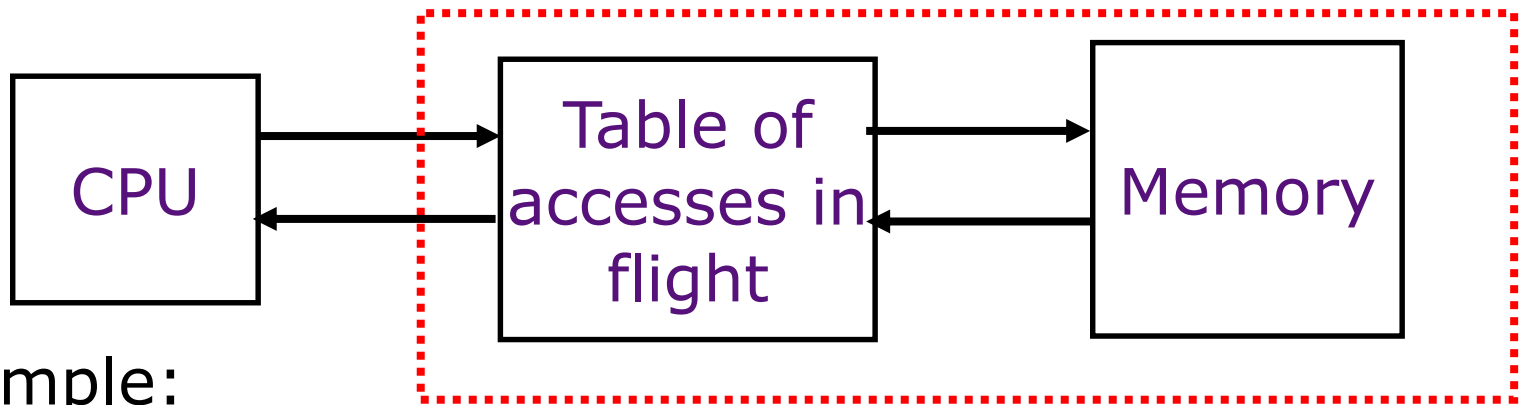*Throughput (T) = Number in Flight (N) / Latency (L)*

Example:

> *--- Assume infinite-bandwidth memory*
> *--- 100 cycles / memory reference*
> *--- 1 + 0.2 memory references / instruction*

Little's Law

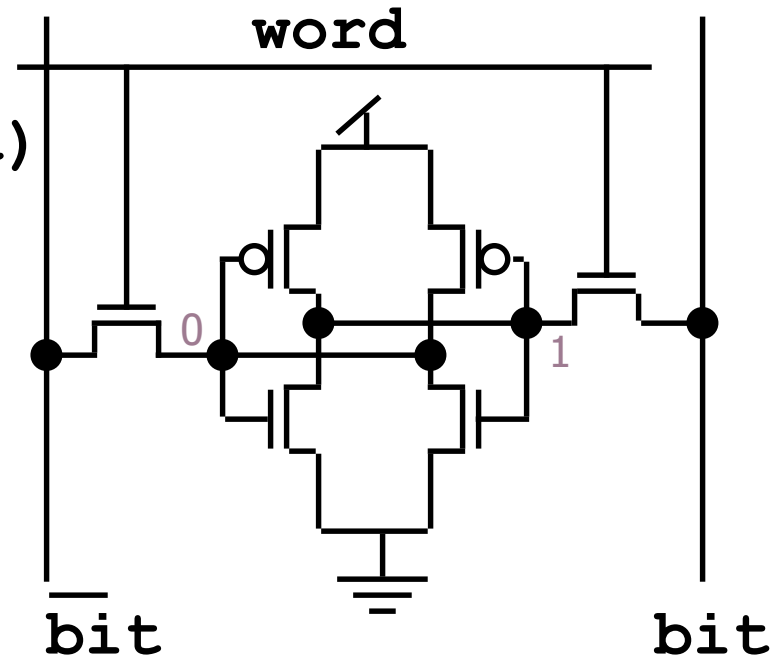*Throughput (T) = Number in Flight (N) / Latency (L)*

Example:

--- Assume infinite-bandwidth memory
--- 100 cycles / memory reference
--- 1 + 0.2 memory references / instruction

$\Rightarrow$ *Table size = 1.2 * 100 = 120 entries*

120 independent memory operations in flight!

Basic Static RAM Cell
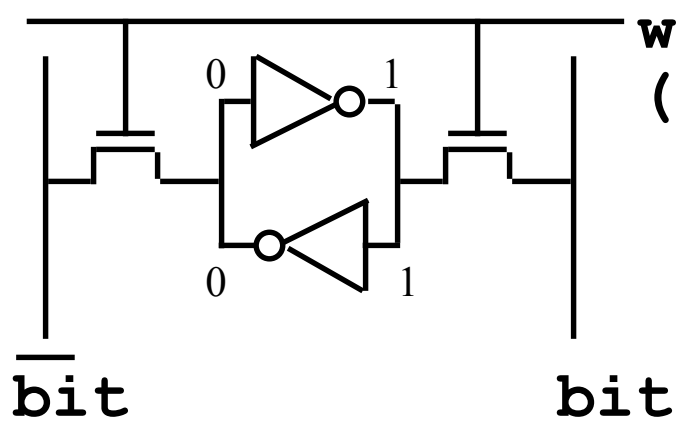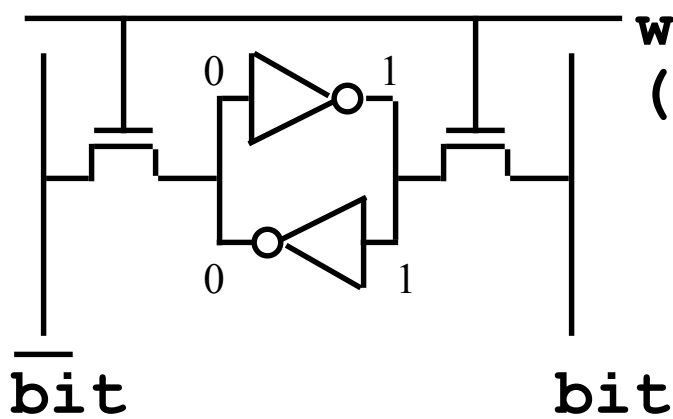
6-Transistor SRAM Cell

word
(row select)

bit

bit

word

bit

bit

Basic Static RAM Cell

6-Transistor SRAM Cell

word
(row select)

$\overline{\textbf{bit}}$ **bit**

- Write:
  1. Drive bit lines (bit=1, $\overline{\text{bit}}$=0)
  2. Select word line

word

$\overline{\textbf{bit}}$ **bit**

Basic Static RAM Cell
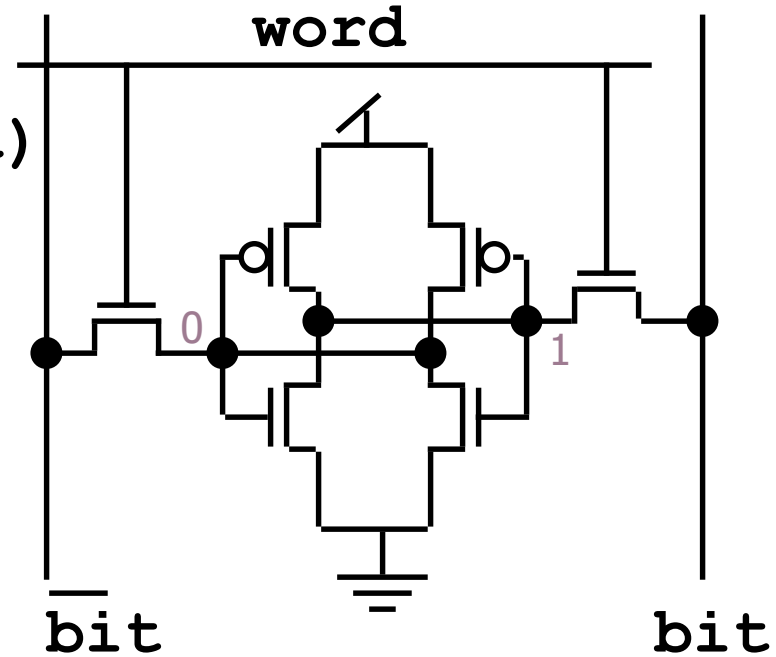
6-Transistor SRAM Cell

word
(row select)
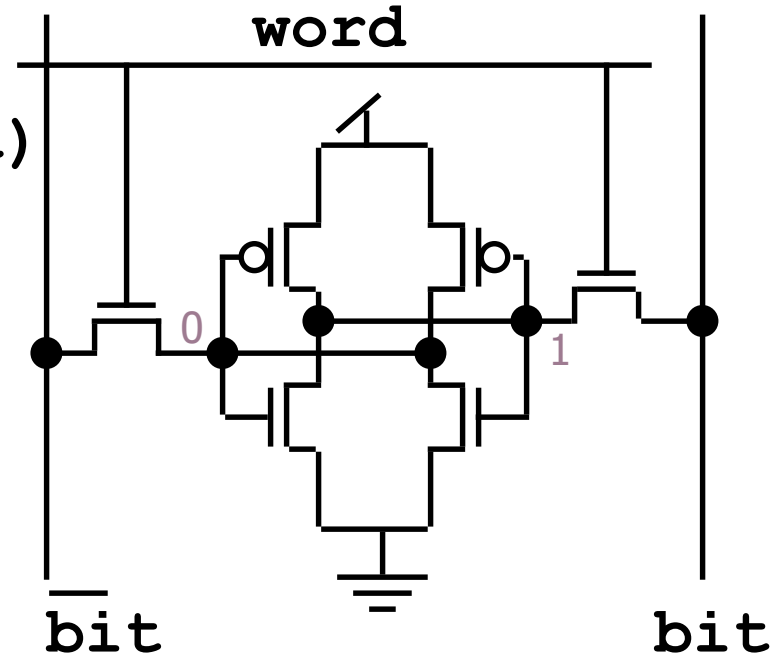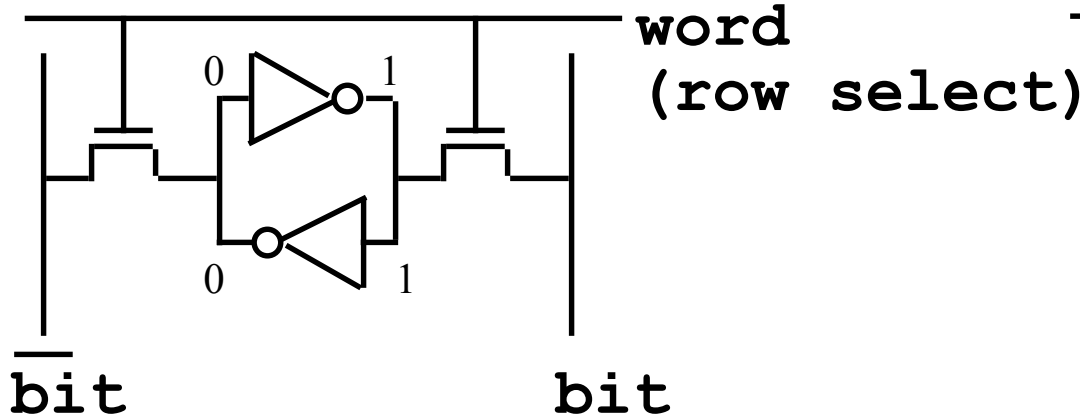
bit   **bit**

- Write:
    1. Drive bit lines (bit=1, bit=0)
    2. Select word line
- Read:
    1. Precharge bit and bit to Vdd
    2. Select word line
    3. Cell pulls one bit line low
    4. Column sense amp detects difference between bit & bit

Multilevel Memory

Strategy: <u>Reduce</u> average latency using small, fast memories called caches.

Caches are a mechanism to reduce memory latency based on the empirical observation that the patterns of memory references made by a processor are often highly predictable:

Multilevel Memory

Strategy: <u>Reduce</u> average latency using small, fast memories called caches.

Caches are a mechanism to reduce memory latency based on the empirical observation that the patterns of memory references made by a processor are often highly predictable:

		PC
	…	*96*
Loop:	*add r2, r1, r1*	*100*
	subi r3, r3, #1	*104*
	bnez r3, loop	*108*
	…	*112*

Typical Memory Reference Patterns

Typical Memory Reference Patterns

Typical Memory Reference Patterns

Typical Memory Reference Patterns

Address

n loop iterations

Instruction fetches

subroutine call

Stack accesses

Time

Typical Memory Reference Patterns

Address

n loop iterations

Instruction fetches

subroutine call

Stack accesses

argument access

Time

Typical Memory Reference Patterns

Typical Memory Reference Patterns

Address

n loop iterations

Instruction fetches

subroutine call

Stack accesses

subroutine return

argument access

Data accesses

Time

Typical Memory Reference Patterns

Address

n loop iterations

Instruction fetches

subroutine call

subroutine return

Stack accesses

argument access

Data accesses

Time

Typical Memory Reference Patterns

Address

n loop iterations

Instruction fetches

Stack accesses

subroutine call

subroutine return

argument access

Data accesses

vector access

Time

Typical Memory Reference Patterns

Address

n loop iterations

Instruction fetches

subroutine call

subroutine return

Stack accesses

argument access

vector access

Data accesses

scalar accesses

Time

Common Predictable Patterns

Two predictable properties of memory references:

- *Temporal Locality:* If a location is referenced, it is likely to be referenced again in the near future

- *Spatial Locality:* If a location is referenced, it is likely that locations near it will be referenced in the near future

Memory Hierarchy

holds frequently used data
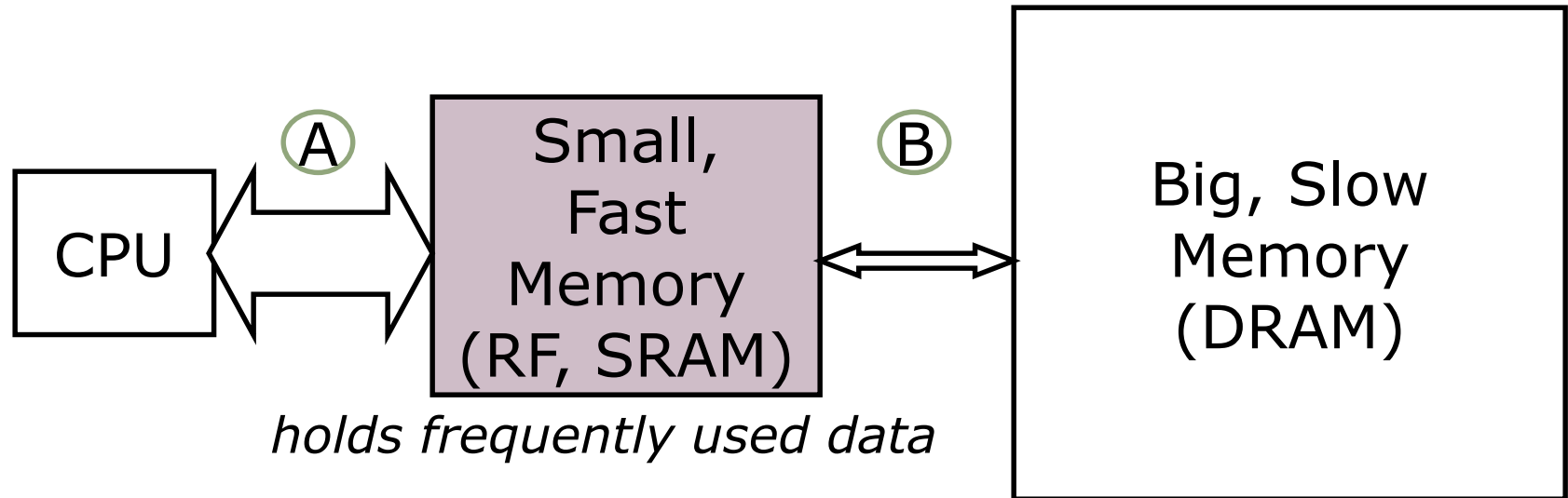
- *size*:   Register << SRAM << DRAM   *why?*
- *latency:*   Register << SRAM << DRAM   *why?*
- *bandwidth:*   on-chip >> off-chip   *why?*

Memory Hierarchy
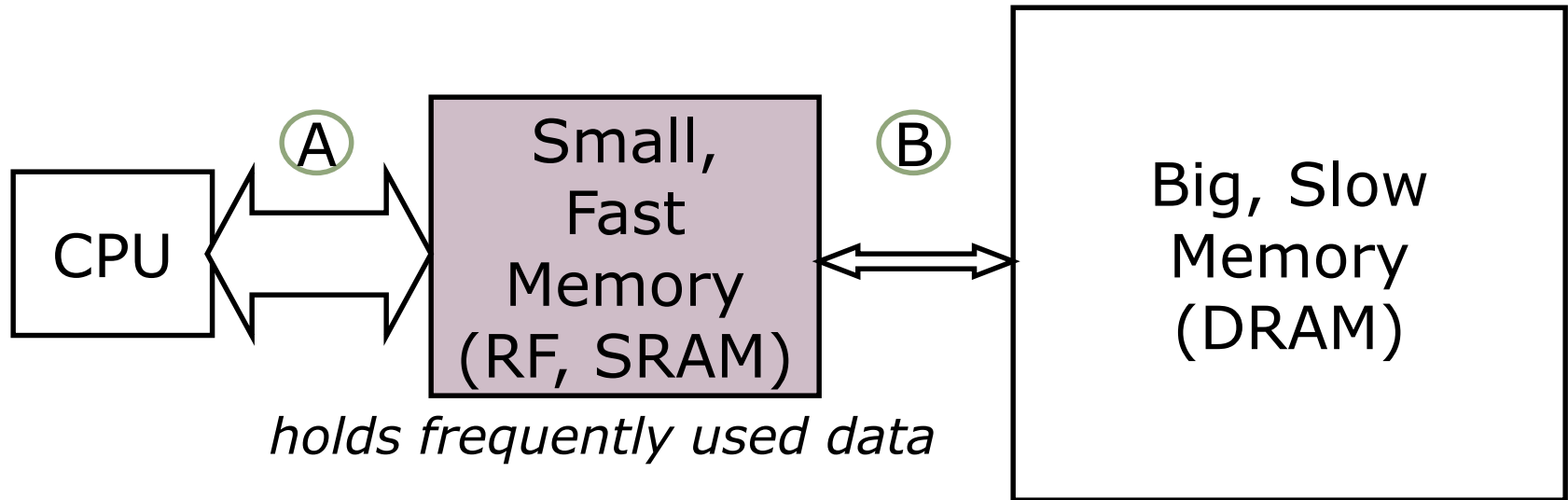
- *size*: Register << SRAM << DRAM   *why?*
- *latency:* Register << SRAM << DRAM   *why?*
- *bandwidth:* on-chip >> off-chip   *why?*

On a data access:
   hit (data ∈ fast memory)   ⇒ low latency access
   miss (data ∉ fast memory) ⇒ long latency access *(DRAM)*

Management of Memory Hierarchy

- ## Small/fast storage, e.g., registers
  - Address usually specified in instruction
  - Generally implemented directly as a register file
    - but hardware might do things behind software's back, e.g., stack management, register renaming

- ## Large/slower storage, e.g., memory
  - Address usually computed from values in register
  - Generally implemented as a cache hierarchy
    - hardware decides what is kept in fast memory
    - but software may provide "hints", e.g., don't cache or prefetch

Inside a Cache

Inside a Cache

Processor → Address → CACHE → Address → Main Memory

Processor ← Data ← CACHE ← Data ← Main Memory

copy of main memory location 100

copy of main memory location 101

Address Tag	Data Block					
100	Data Byte	Data Byte			-------	
304	Data Byte				-------	
6848						
416						

Line

Data Block
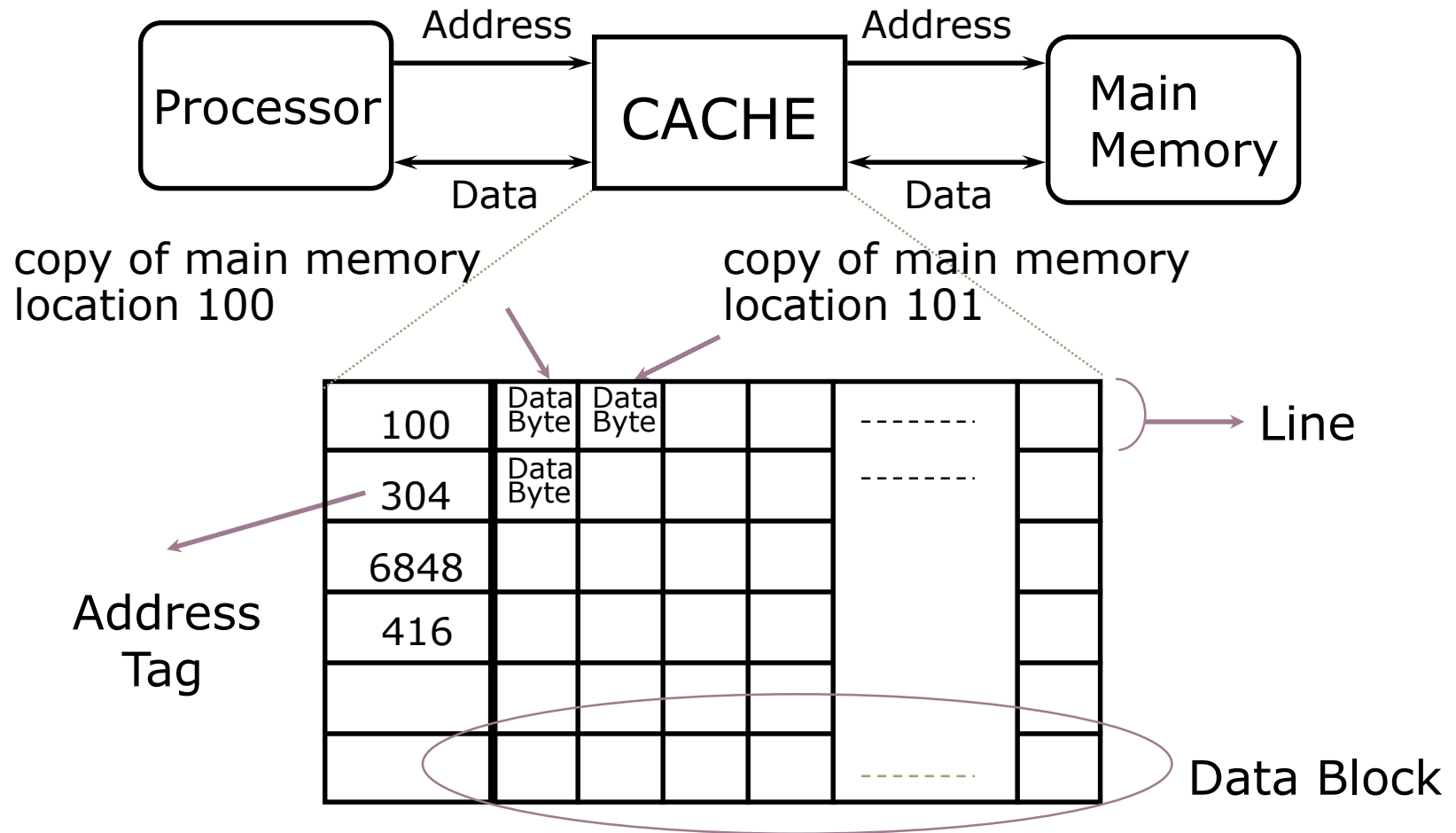
Q: How many bits needed in tag? _____

Inside a Cache
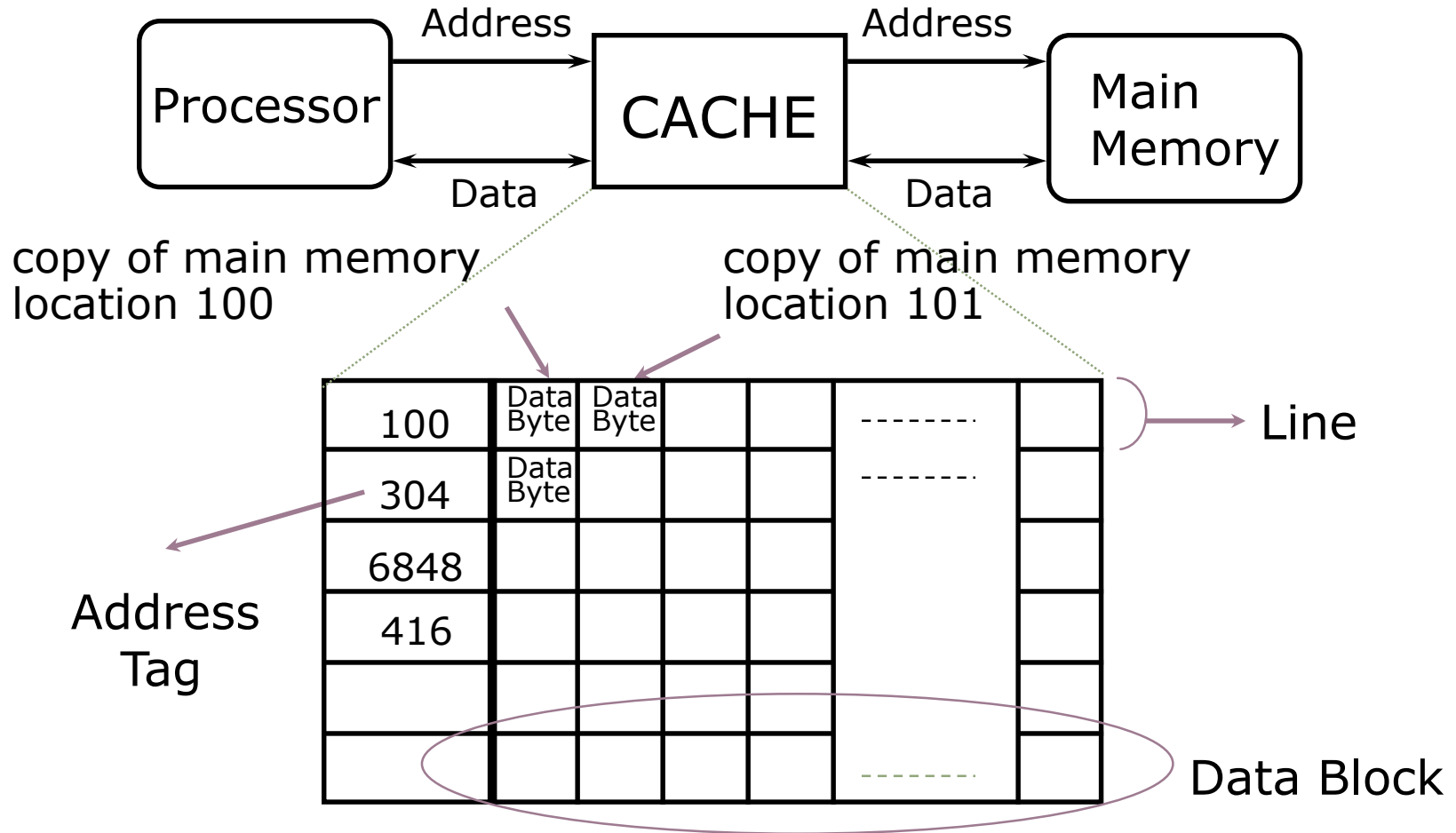
Q: How many bits needed in tag? <u>Enough to uniquely identify block</u>

Cache Algorithm (Read)

Look at Processor Address, search cache tags to find match. Then either

Cache Algorithm (Read)

Look at Processor Address, search cache tags to find match. Then either

Found in cache
a.k.a. HIT
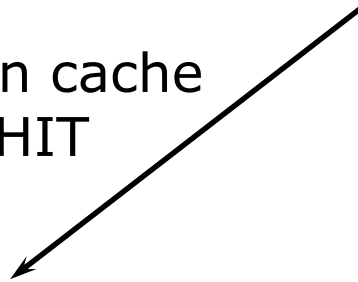
Return copy
of data from
cache

Cache Algorithm (Read)

Look at Processor Address, search cache tags to find match. Then either

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

Cache Algorithm (Read)

Look at Processor Address, search cache tags to find match. Then either

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

Which line do we replace?

Direct-Mapped Cache
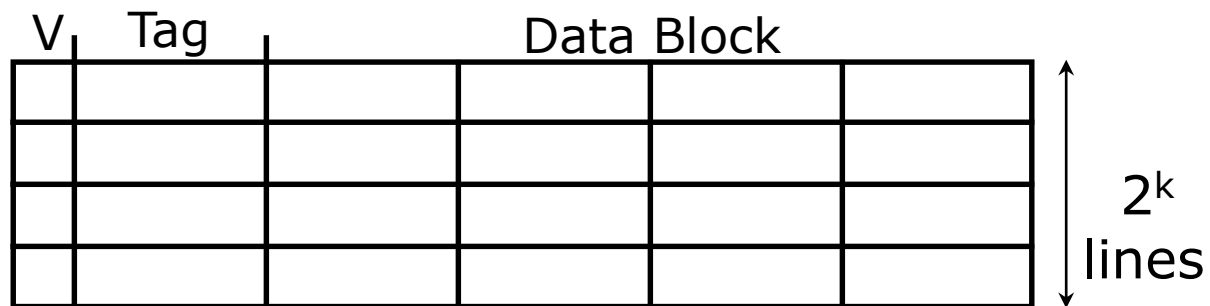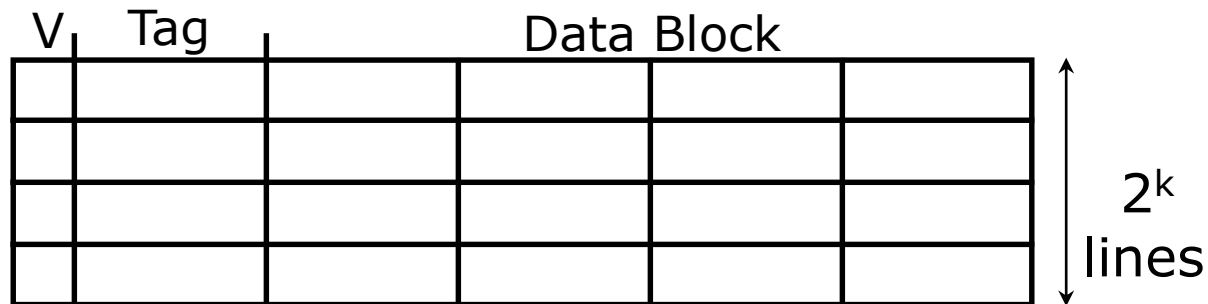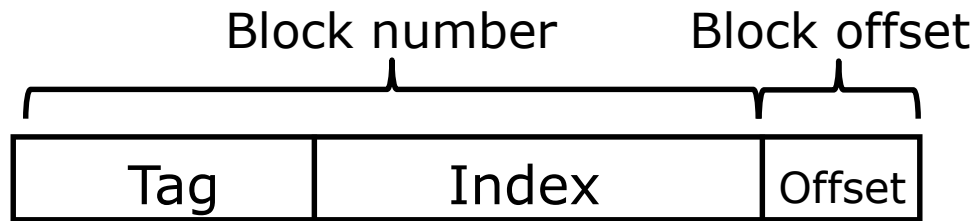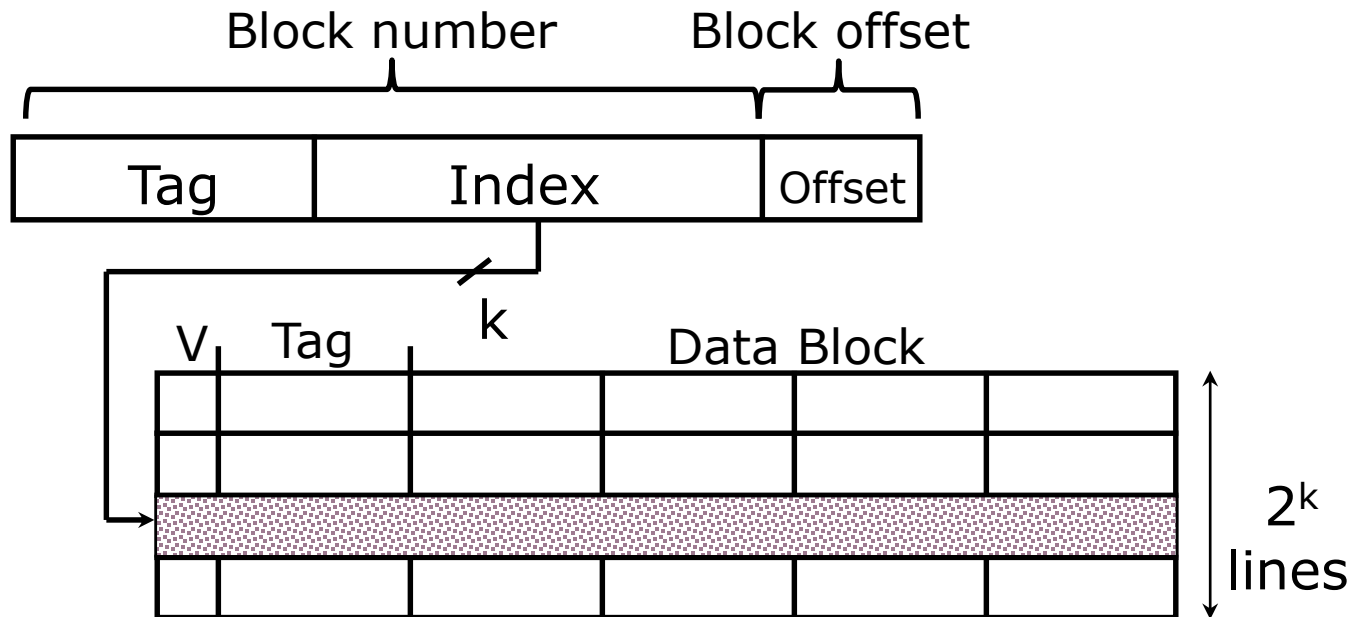
	V	Tag		Data Block		

2^k lines

Direct-Mapped Cache

Block number   Block offset

Tag	Index	Offset

V	Tag		Data Block		

2^k lines

Direct-Mapped Cache

Block number   Block offset

Tag	Index	Offset

k

V   Tag   Data Block

2^k lines

Direct-Mapped Cache

Direct-Mapped Cache

Direct-Mapped Cache

Block number   Block offset

Tag	Index	Offset

t

V   Tag   k   Data Block   b

2^k lines

t

HIT

Data Word or Byte
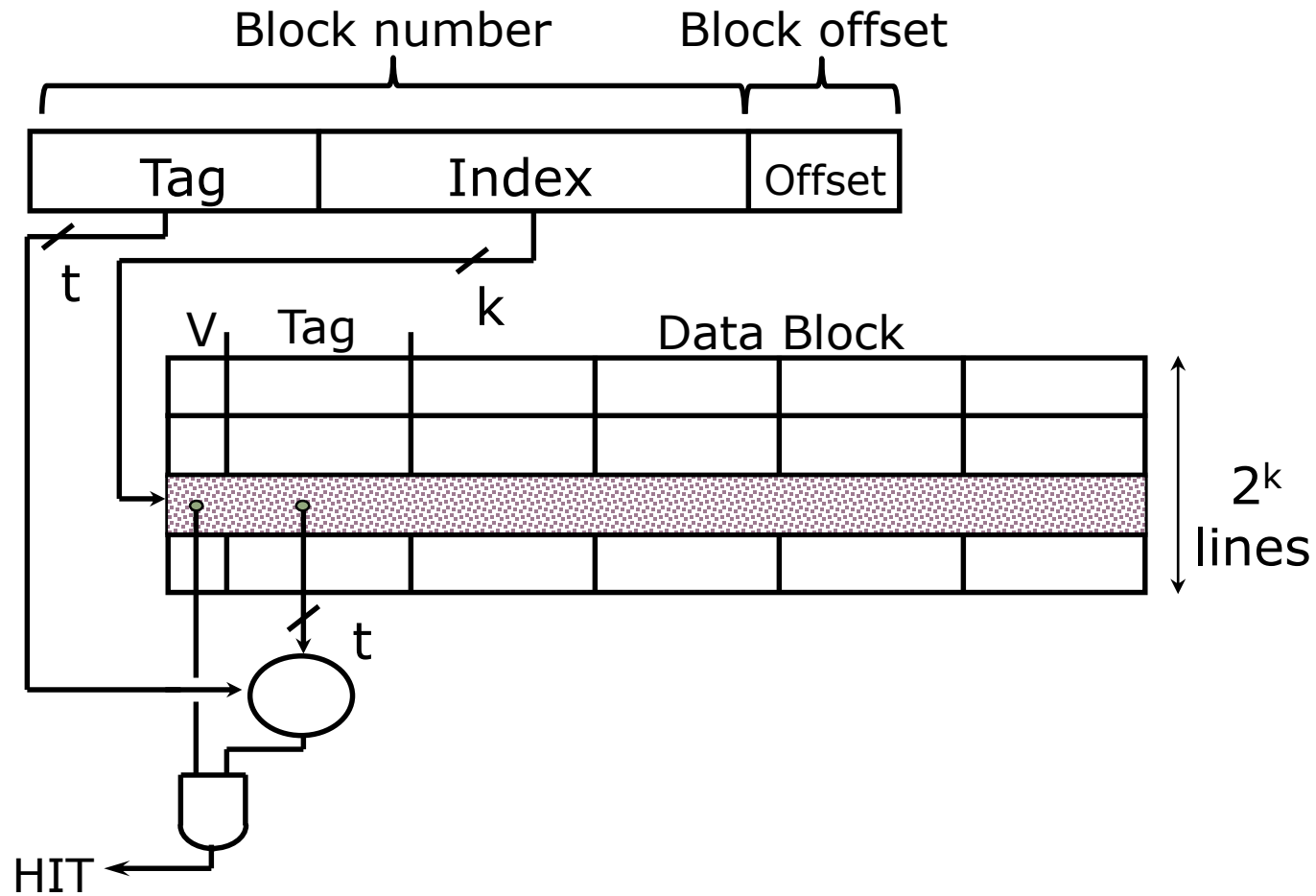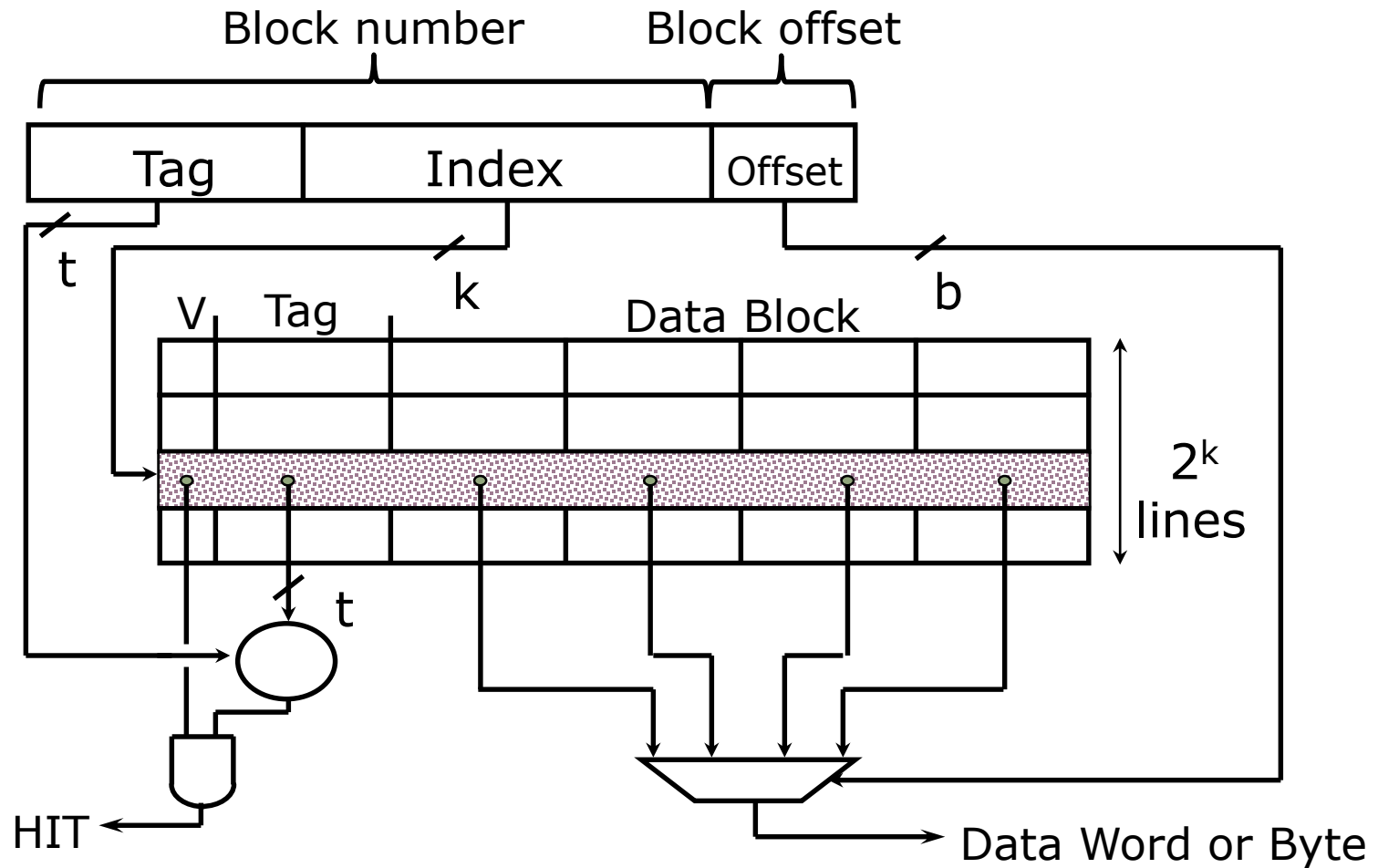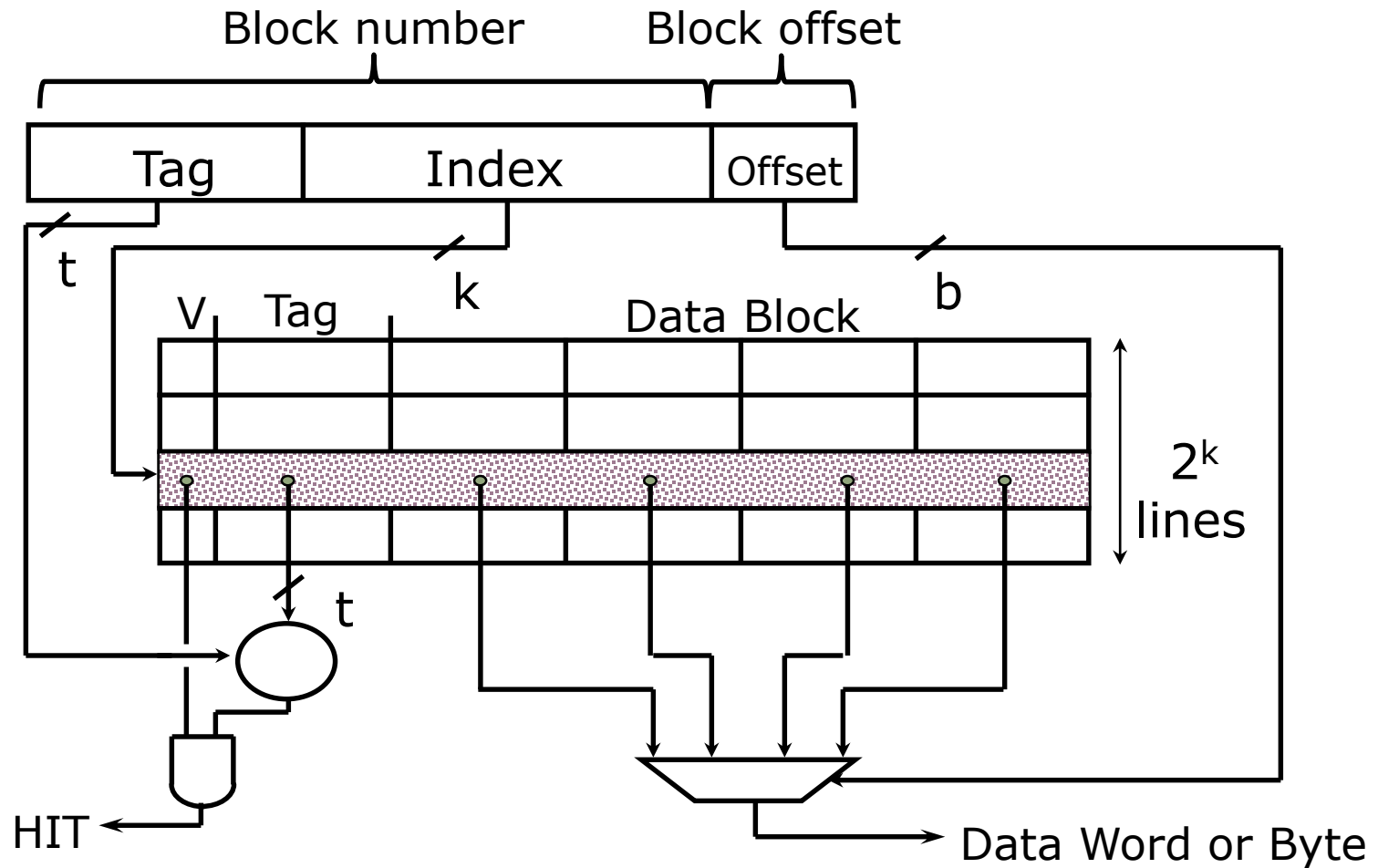
Q: What is a bad reference pattern? _____

Direct-Mapped Cache

Block number   Block offset

Tag   Index   Offset

t   k   b

V   Tag   Data Block

2^k lines

t

HIT

Data Word or Byte
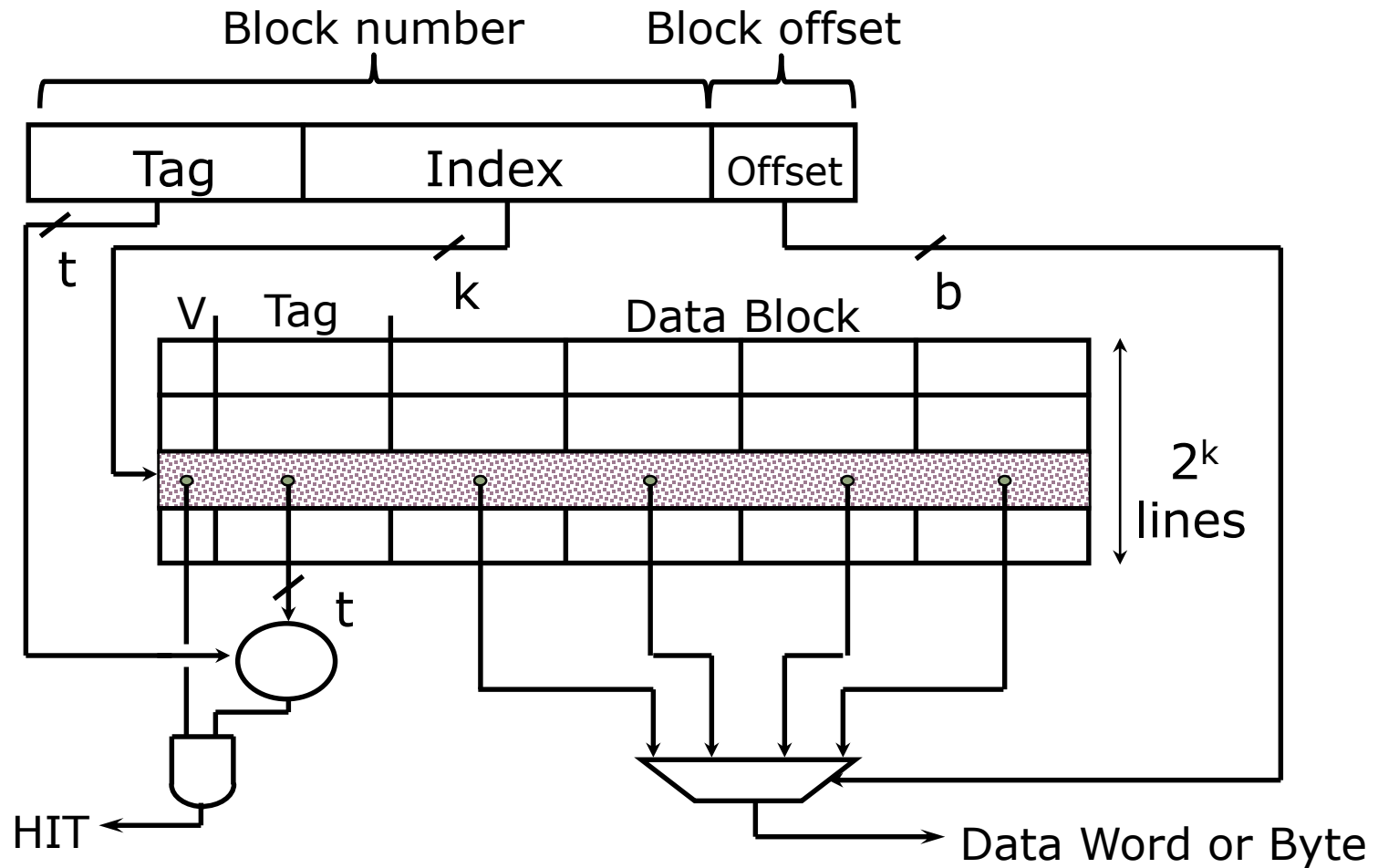
Q: What is a bad reference pattern?   *Strided at size of cache*

Direct Map Address Selection
higher-order vs. lower-order address bits

Index	Tag	Offset

k   t   b

V   Tag   Data Block

2^k lines

t

=

HIT

Data Word or Byte
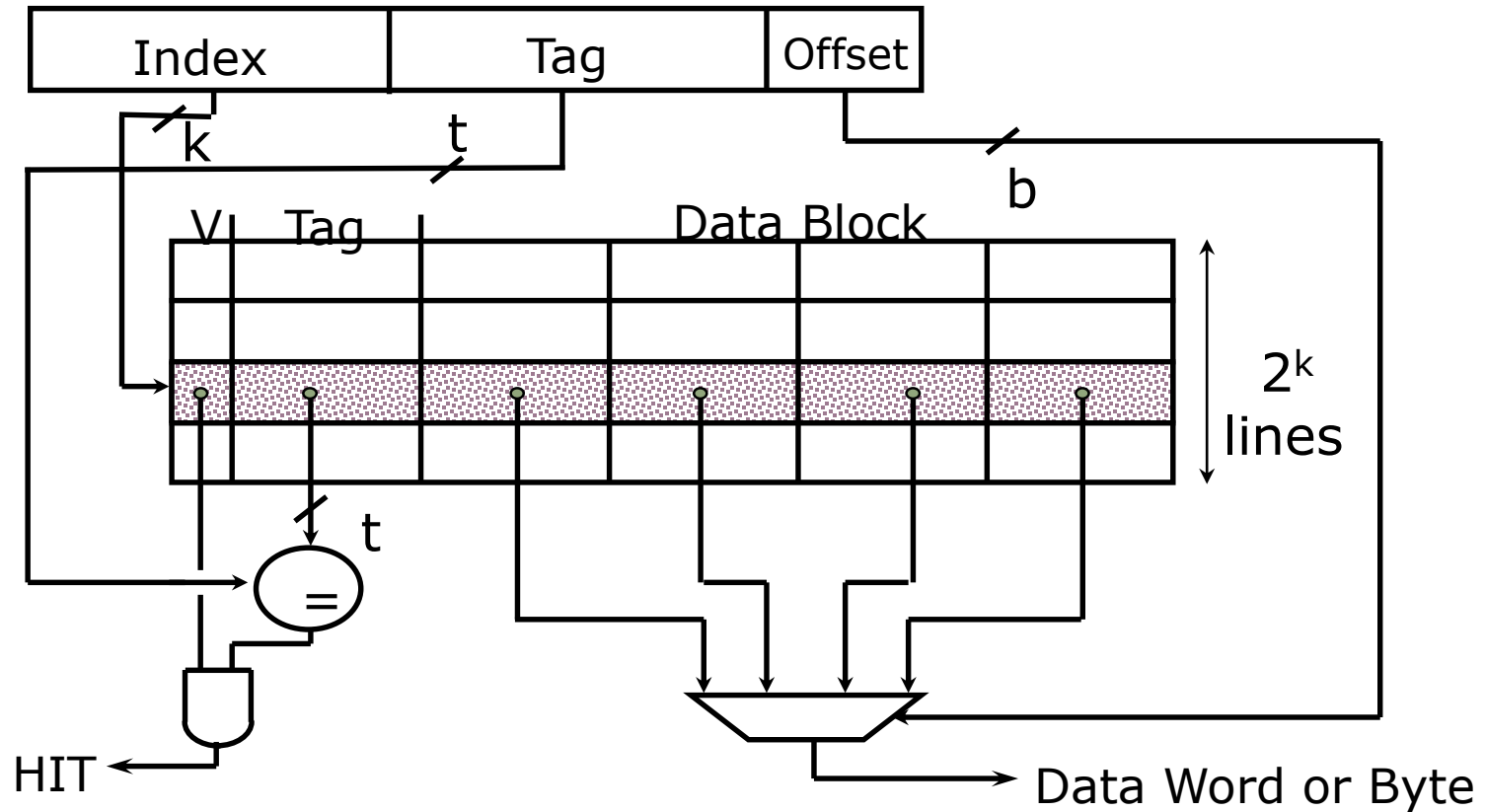
Q: Why might this be undesirable? _____

Direct Map Address Selection
higher-order vs. lower-order address bits
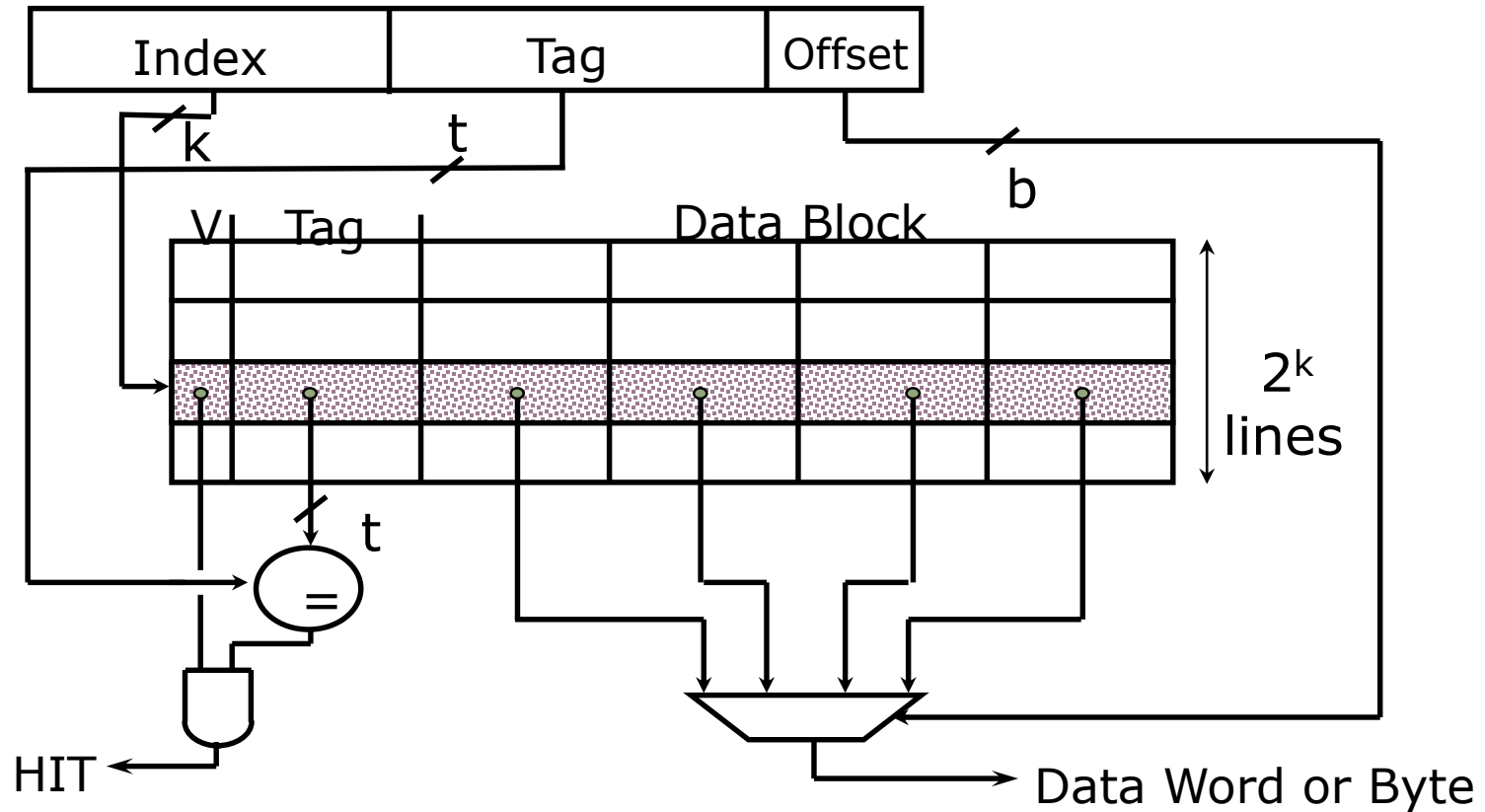
Q: Why might this be undesirable? Spatially local blocks conflict

Hashed Address Mapping
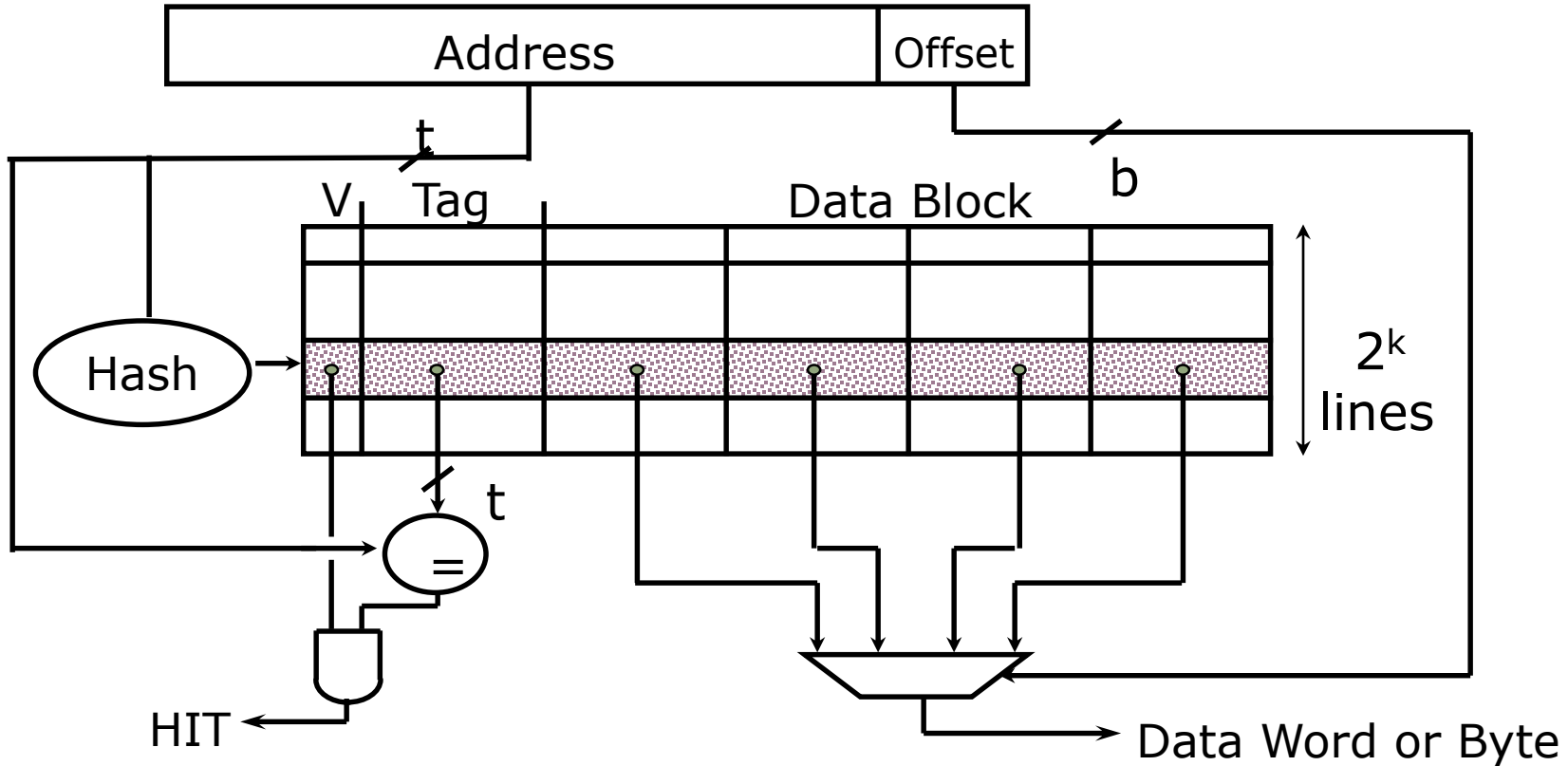
Q: What are the tradeoffs of hashing?

Hashed Address Mapping
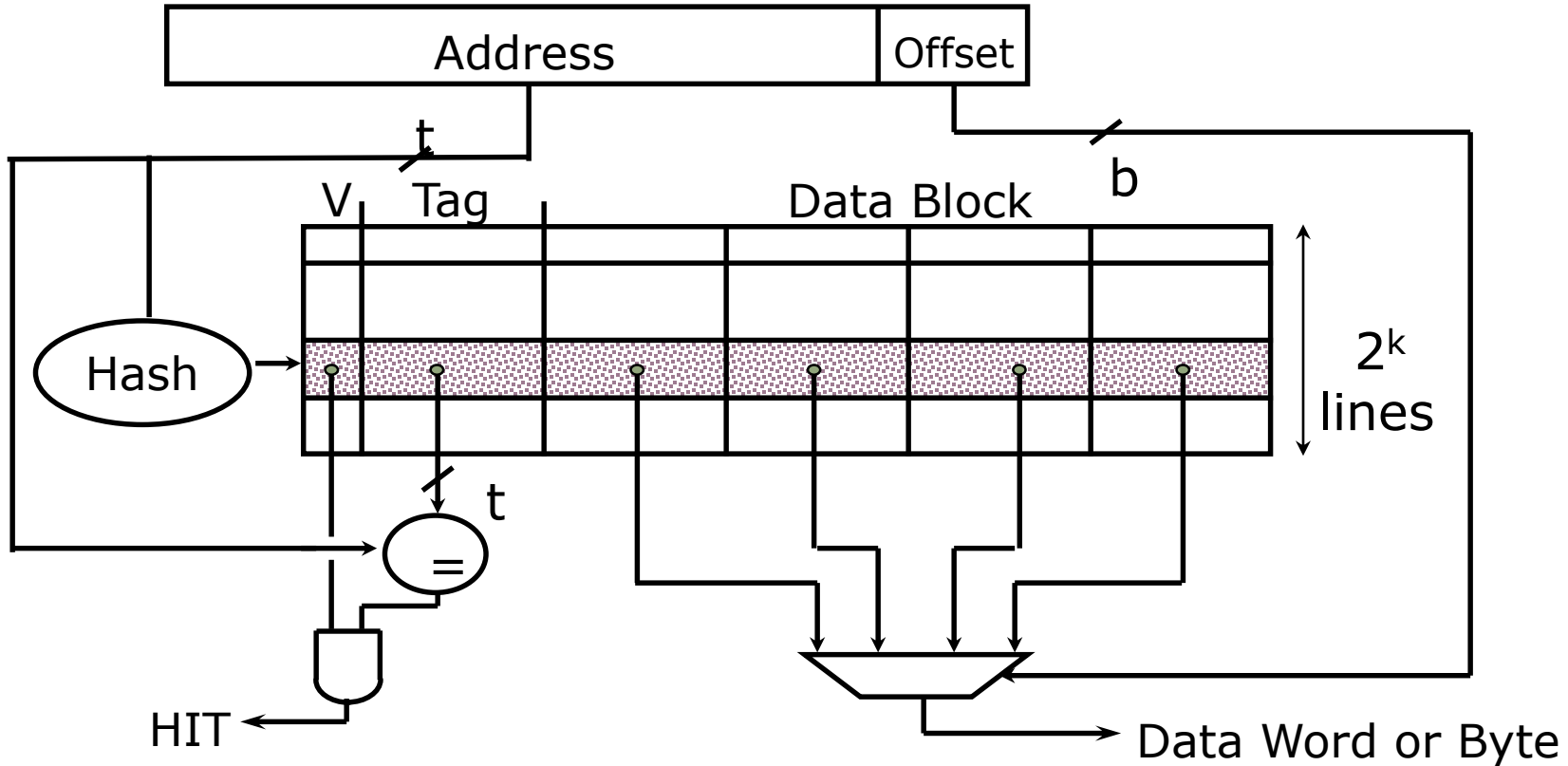
Q: What are the tradeoffs of hashing?

Good: Regular strides don't conflict
Bad: Hash adds latency
Tag is larger

2-Way Set-Associative Cache

Tag	Index	Block Offset

V	Tag	Data Block	V	Tag	Data Block
●	●	●			●

2-Way Set-Associative Cache

Tag	Index	Block Offset

k

V	Tag	Data Block

V	Tag	Data Block
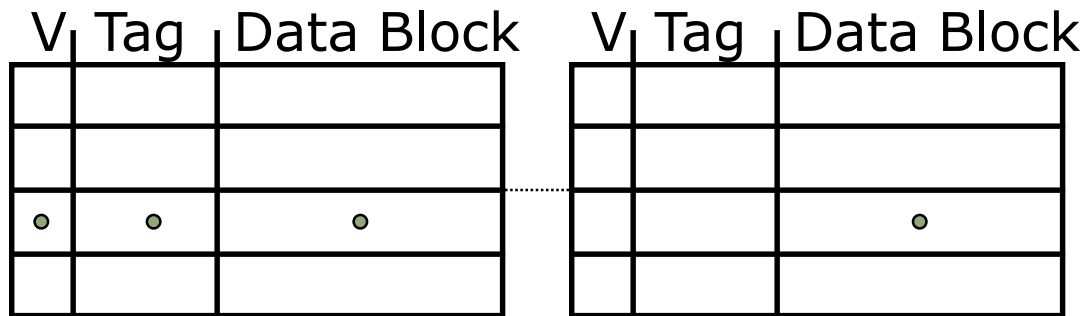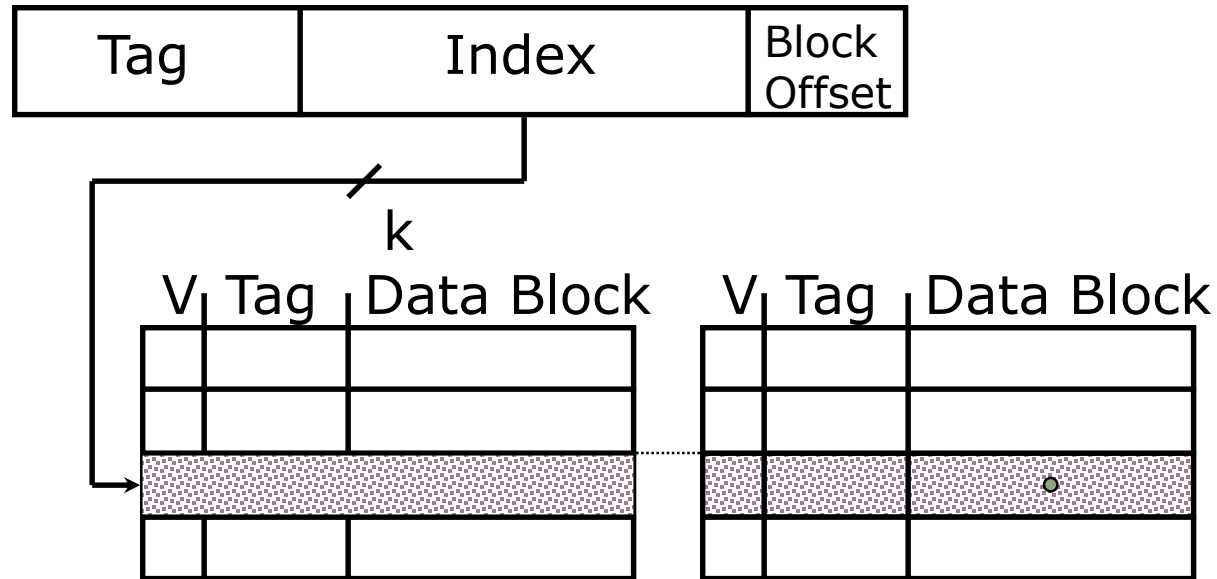
2-Way Set-Associative Cache
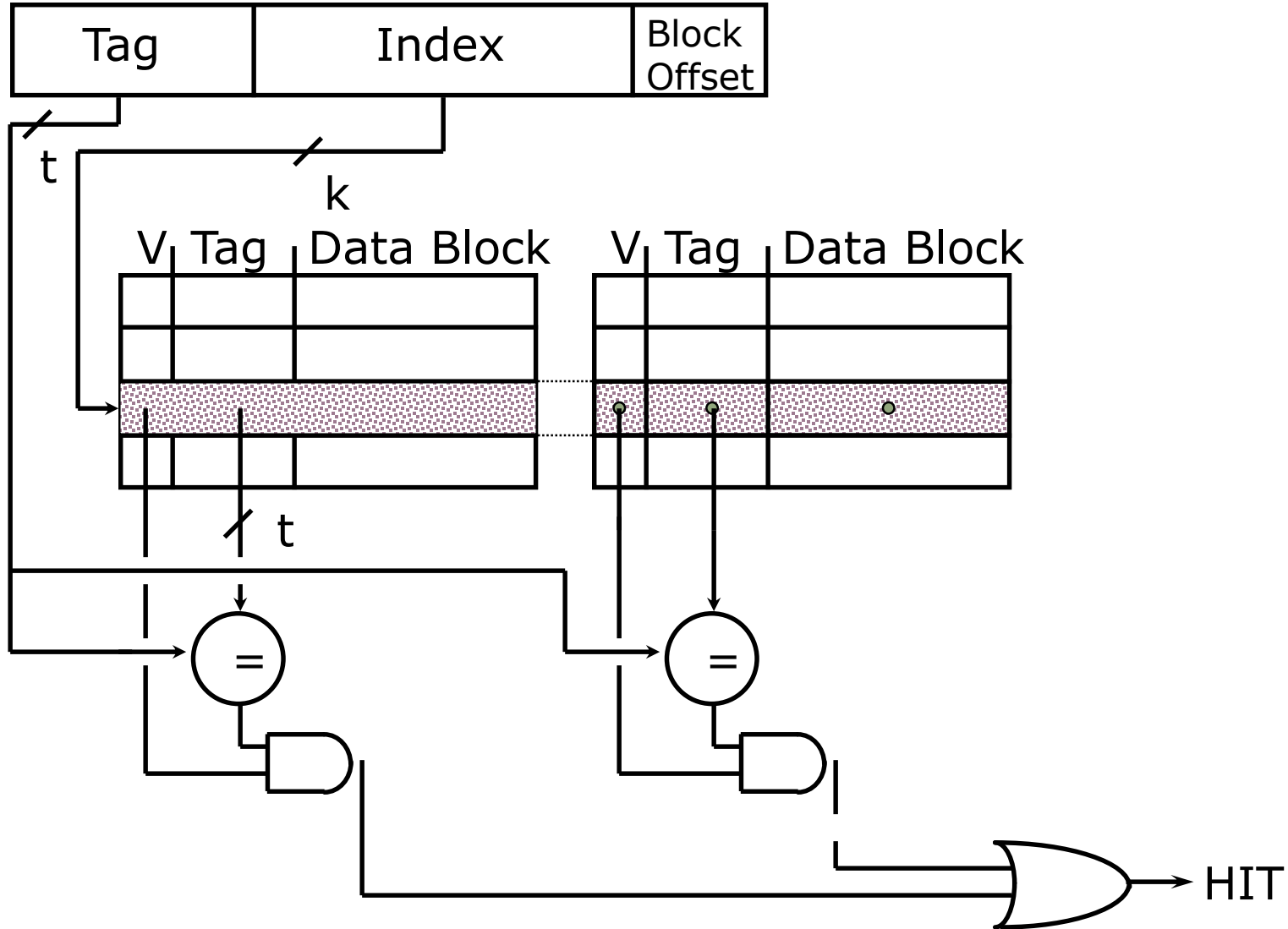
2-Way Set-Associative Cache
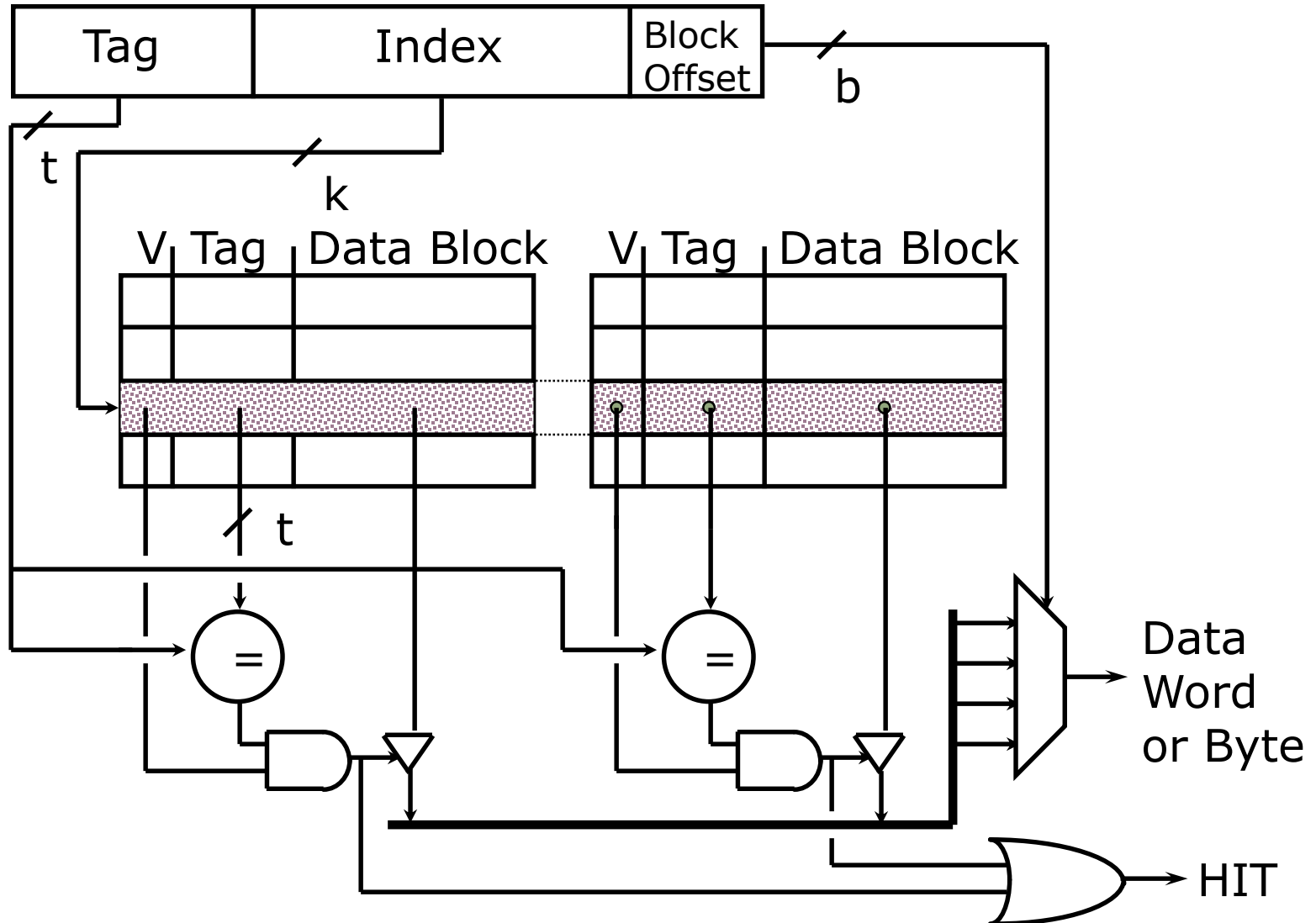
Fully Associative Cache

Placement Policy

Block Number

1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

Memory

Set Number   0 1 2 3 4 5 6 7

Cache

Direct
Mapped
only into
block 4
(12 mod 8)

block 12
can be placed

Placement Policy

Block Number
```
                          1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```

Memory

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

Cache

Direct
Mapped
only into
block 4
(12 mod 8)
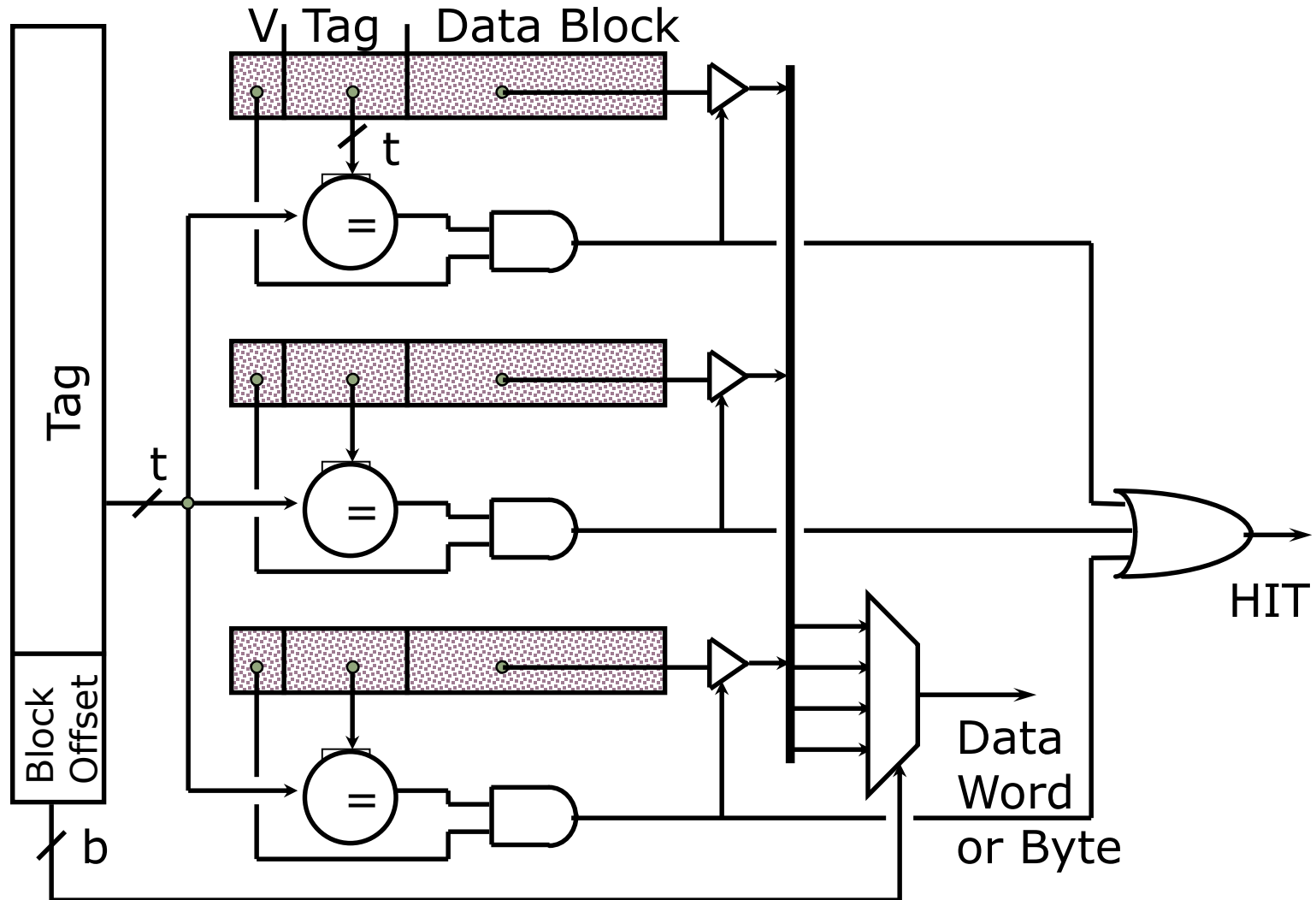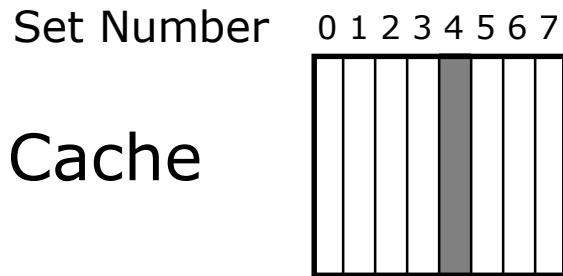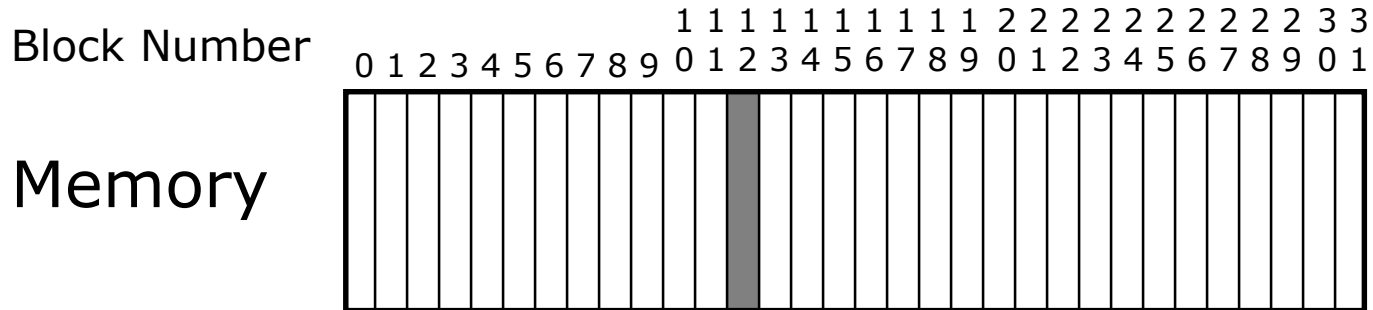
(2-way) Set
Associative
anywhere in
set 0
(12 mod 4)

block 12
can be placed

Placement Policy

Block Number

1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

Memory

Set Number   0 1 2 3 4 5 6 7   0   1   2   3
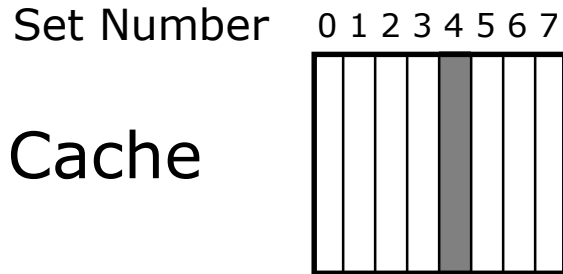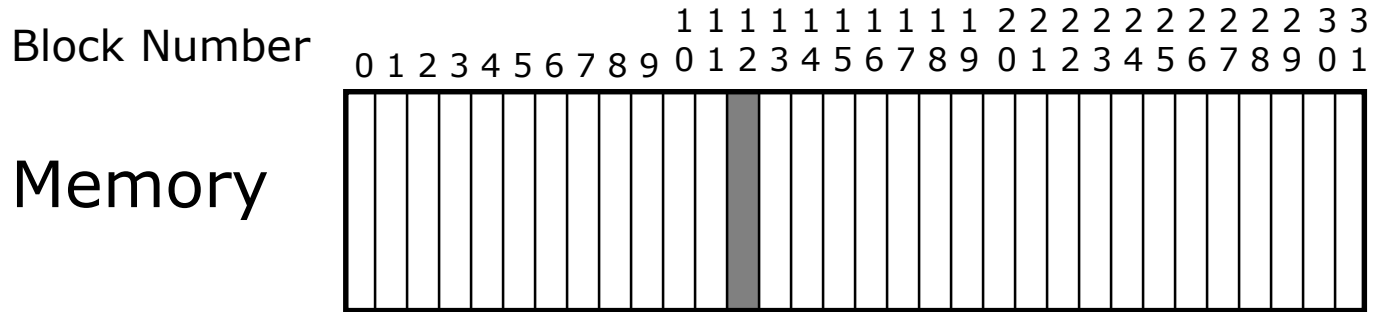
Cache

Direct
Mapped
only into
block 4
(12 mod 8)
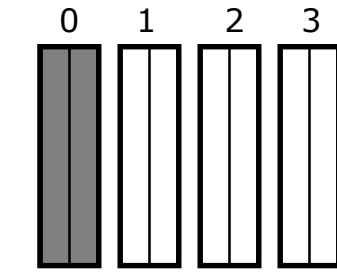
(2-way) Set
Associative
anywhere in
set 0
(12 mod 4)
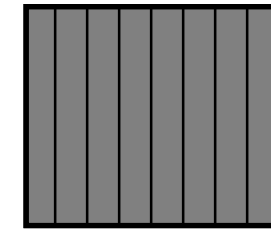
Fully
Associative
anywhere

block 12
can be placed

Improving Cache Performance

Average memory access time =
  Hit time + Miss rate x Miss penalty

To improve performance:
- reduce the hit time
- reduce the miss rate (e.g., larger, better policy)
- reduce the miss penalty (e.g., L2 cache)

What is the simplest design strategy?

Improving Cache Performance

Average memory access time =
          Hit time + Miss rate x Miss penalty

To improve performance:
- reduce the hit time
- reduce the miss rate (e.g., larger, better policy)
- reduce the miss penalty (e.g., L2 cache)

What is the simplest design strategy?

Biggest cache that doesn't increase hit time past 1-2 cycles (approx. 16-64KB in modern technology)

[design issues more complex with out-of-order superscalar processors]

Causes for Cache Misses

- *Compulsory:*
  First reference to a block *a.k.a.* cold start misses
  - misses that would occur even with infinite cache

- *Capacity:*
  cache is too small to hold all data the program needs
  - misses that would occur even under perfect
    placement & replacement policy

- *Conflict:*
  misses from collisions due to block-placement strategy
  - misses that would not occur with full associativity

Effect of Cache Parameters on Performance

	Larger capacity cache	**Higher associativity cache**	**Larger block size cache ***
Compulsory misses			
Capacity misses			
Conflict misses			
Hit latency			
Miss latency			

* Assume substantial spatial locality

Effect of Cache Parameters on Performance

	Larger capacity cache	**Higher associativity cache**	**Larger block size cache ***
Compulsory misses	=		
Capacity misses			
Conflict misses			
Hit latency			
Miss latency			

* Assume substantial spatial locality

Effect of Cache Parameters on Performance

	Larger capacity cache	Higher associativity cache	Larger block size cache *
Compulsory misses	**=**		
Capacity misses	**↓**		
Conflict misses			
Hit latency			
Miss latency			

* Assume substantial spatial locality

Effect of Cache Parameters on Performance

	Larger capacity cache	Higher associativity cache	Larger block size cache *
Compulsory misses	=		
Capacity misses	⬇		
Conflict misses	⬇		
Hit latency			
Miss latency			

* Assume substantial spatial locality

Effect of Cache Parameters on Performance

	Larger capacity cache	Higher associativity cache	Larger block size cache *
Compulsory misses	=		
Capacity misses	↓		
Conflict misses	↓		
Hit latency	↑		
Miss latency			

* Assume substantial spatial locality

Effect of Cache Parameters on Performance

	Larger capacity cache	**Higher associativity cache**	**Larger block size cache ***
Compulsory misses	=		
Capacity misses	⬇		
Conflict misses	⬇		
Hit latency	⬆		
Miss latency	=		

* Assume substantial spatial locality

Effect of Cache Parameters on Performance

	Larger capacity cache	**Higher associativity cache**	**Larger block size cache ***
Compulsory misses	=	=	
Capacity misses	↓		
Conflict misses	↓		
Hit latency	↑		
Miss latency	=		

* Assume substantial spatial locality

Effect of Cache Parameters on Performance

	Larger capacity cache	**Higher associativity cache**	**Larger block size cache ***
Compulsory misses	=	=	
Capacity misses	↓	=	
Conflict misses	↓		
Hit latency	↑		
Miss latency	=		

* Assume substantial spatial locality

Effect of Cache Parameters on Performance

	Larger capacity cache	Higher associativity cache	Larger block size cache *
Compulsory misses	=	=	
Capacity misses	↓	=	
Conflict misses	↓	↓	
Hit latency	↑		
Miss latency	=		

* Assume substantial spatial locality

Effect of Cache Parameters on Performance

	Larger capacity cache	Higher associativity cache	Larger block size cache *
Compulsory misses	=	=	
Capacity misses	⬇	=	
Conflict misses	⬇	⬇	
Hit latency	⬆	⬆	
Miss latency	=		

* Assume substantial spatial locality

Effect of Cache Parameters on Performance

	Larger capacity cache	**Higher associativity cache**	**Larger block size cache ***
Compulsory misses	=	=	
Capacity misses	↓	=	
Conflict misses	↓	↓	
Hit latency	↑	↑	
Miss latency	=	=	

* Assume substantial spatial locality

Effect of Cache Parameters on Performance

	Larger capacity cache	Higher associativity cache	Larger block size cache *
Compulsory misses	=	=	⬇
Capacity misses	⬇	=	
Conflict misses	⬇	⬇	
Hit latency	⬆	⬆	
Miss latency	=	=	

* Assume substantial spatial locality

Effect of Cache Parameters on Performance

	Larger capacity cache	Higher associativity cache	Larger block size cache *
Compulsory misses	=	=	↓
Capacity misses	↓	=	↓
Conflict misses	↓	↓	
Hit latency	↑	↑	
Miss latency	=	=	

* Assume substantial spatial locality

Effect of Cache Parameters on Performance

	Larger capacity cache	**Higher associativity cache**	**Larger block size cache ***
Compulsory misses	=	=	⬇
Capacity misses	⬇	=	⬇
Conflict misses	⬇	⬇	**?**
Hit latency	⬆	⬆	
Miss latency	=	=	

* Assume substantial spatial locality

Effect of Cache Parameters on Performance

	Larger capacity cache	**Higher associativity cache**	**Larger block size cache ***
Compulsory misses	=	=	↓
Capacity misses	↓	=	↓
Conflict misses	↓	↓	?
Hit latency	↑	↑	=
Miss latency	=	=	

* Assume substantial spatial locality

Effect of Cache Parameters on Performance

	Larger capacity cache	Higher associativity cache	Larger block size cache *
Compulsory misses	=	=	↓
Capacity misses	↓	=	↓
Conflict misses	↓	↓	?
Hit latency	↑	↑	=
Miss latency	=	=	↑↓

* Assume substantial spatial locality

Block-level Optimizations

- ## Tags are too large, i.e., too much overhead
  - Simple solution: Larger blocks, but miss penalty could be large.

Block-level Optimizations

- ## Tags are too large, i.e., too much overhead
  - Simple solution: Larger blocks, but miss penalty could be large.

- ## Sub-block placement (aka sector cache)
  - A valid bit added to units smaller than the full block, called sub-blocks
  - Only read a sub-block on a miss
  - *If a tag matches, is the sub-block in the cache?*

100	*1*	*1*	*1*	*1*
300	*1*	*1*	*0*	*0*
204	*0*	*1*	*0*	*1*

Replacement Policy

Which block from a set should be evicted?

- Random

Replacement Policy

Which block from a set should be evicted?

• Random

Replacement Policy

Which block from a set should be evicted?

- Random

- Least Recently Used (LRU)
  - LRU cache state must be updated on every access
  - true implementation only feasible for small sets (2-way)
  - pseudo-LRU binary tree was often used for 4-8 way

Replacement Policy

Which block from a set should be evicted?

- Random

- Least Recently Used (LRU)
  - LRU cache state must be updated on every access
  - true implementation only feasible for small sets (2-way)
  - pseudo-LRU binary tree was often used for 4-8 way

- First In, First Out (FIFO) a.k.a. Round-Robin
  - used in highly associative caches

Replacement Policy

Which block from a set should be evicted?

• Random

• Least Recently Used (LRU)
  • LRU cache state must be updated on every access
  • true implementation only feasible for small sets (2-way)
  • pseudo-LRU binary tree was often used for 4-8 way

• First In, First Out (FIFO) a.k.a. Round-Robin
  • used in highly associative caches

• Not Least Recently Used (NLRU)
 • FIFO with exception for most recently used block or blocks

Replacement Policy

Which block from a set should be evicted?

- Random

- Least Recently Used (LRU)
    - LRU cache state must be updated on every access
    - true implementation only feasible for small sets (2-way)
    - pseudo-LRU binary tree was often used for 4-8 way

- First In, First Out (FIFO) a.k.a. Round-Robin
    - used in highly associative caches

- Not Least Recently Used (NLRU)
    - FIFO with exception for most recently used block or blocks

- One-bit LRU
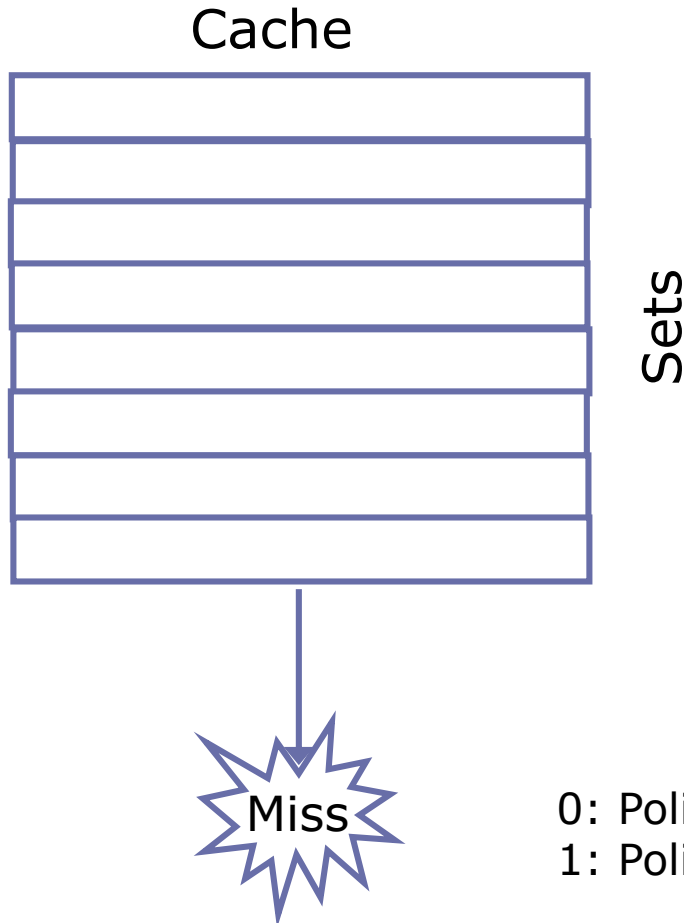   - Each way represented by a bit. Set on use, replace first unused.

Multiple replacement policies
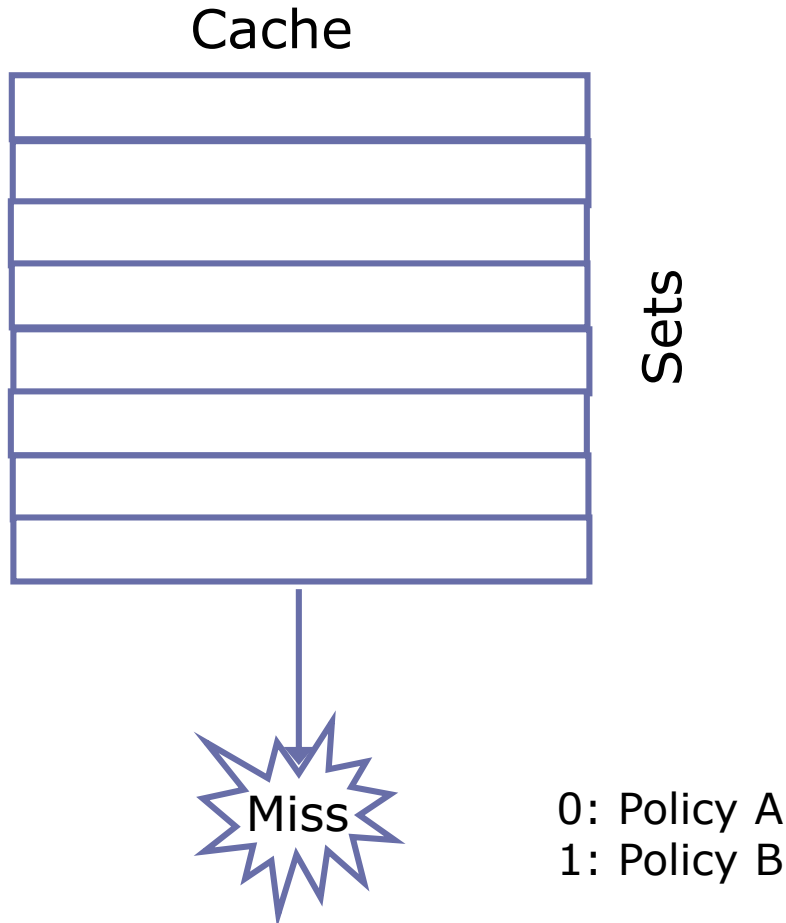
Use the best replacement policy for a program

Cache

Sets

Miss

0: Policy A
1: Policy B

Multiple replacement policies

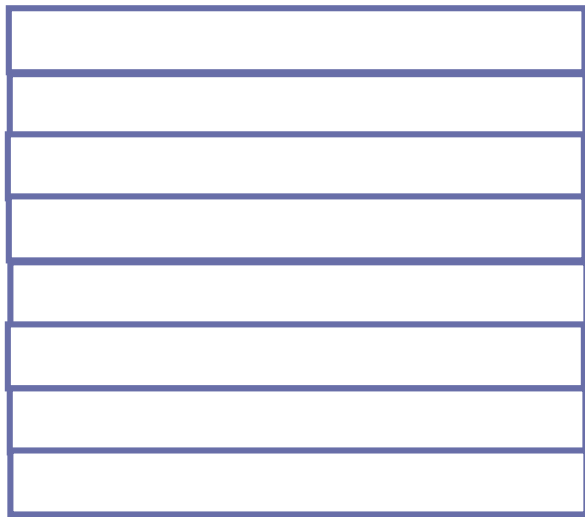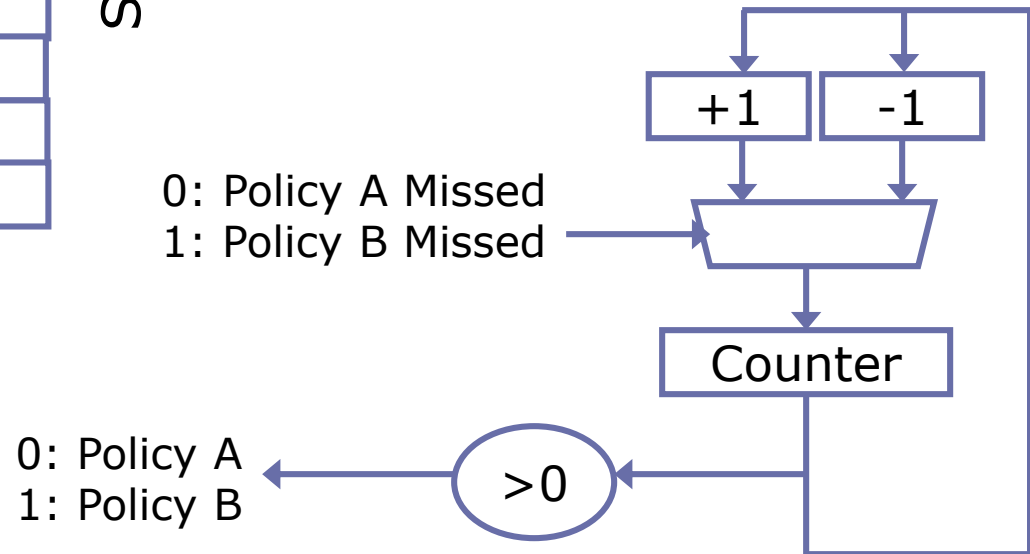Use the best replacement policy for a program

Cache

How do we decide
which policy to use?

Sets

Miss

0: Policy A
1: Policy B

Multiple replacement policies

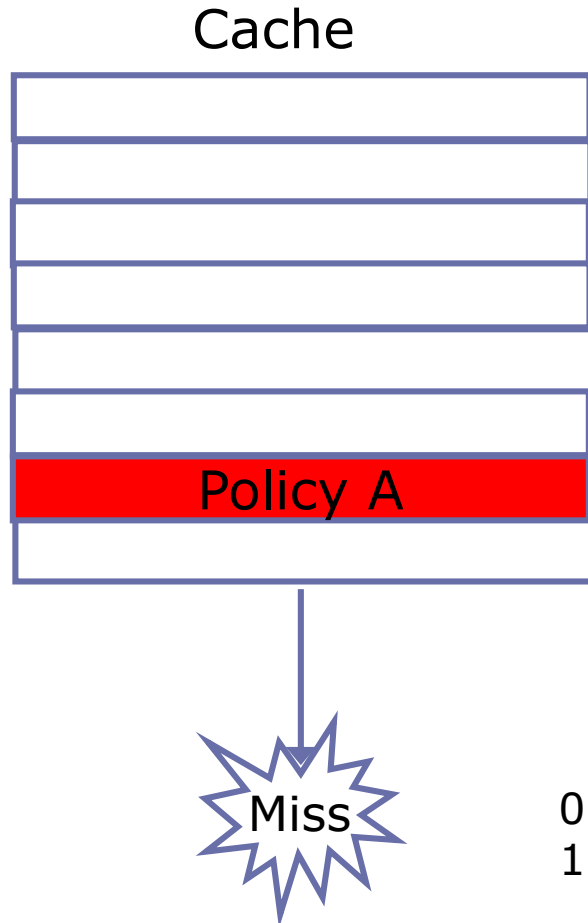Use the best replacement policy for a program

Cache

How do we decide
which policy to use?

Sets

+1   -1

0: Policy A Missed
1: Policy B Missed

Counter

Miss

0: Policy A
1: Policy B

>0

Multiple replacement policies

Use the best replacement policy for a program
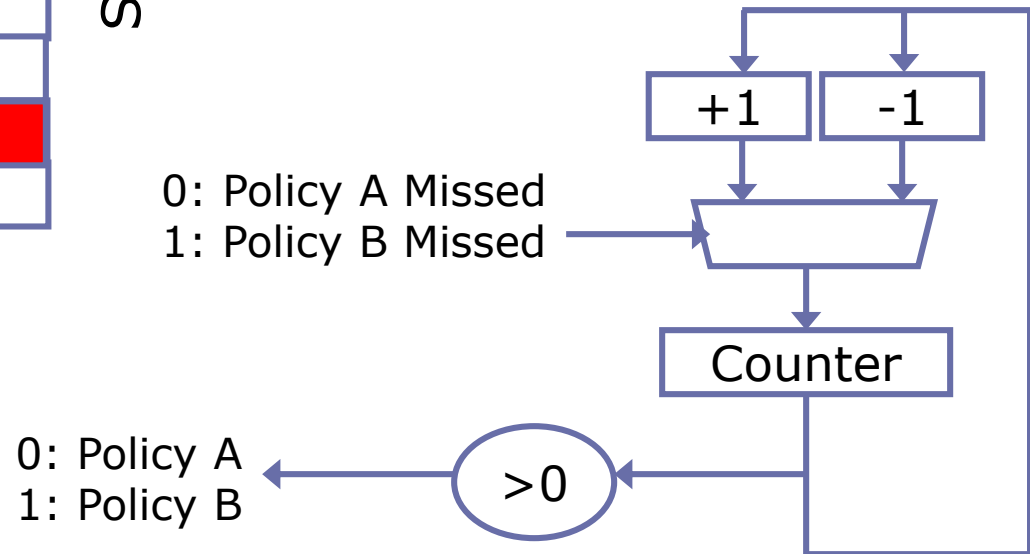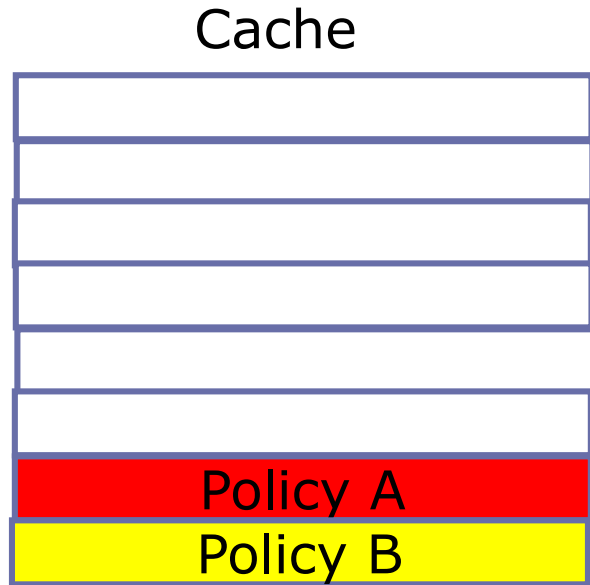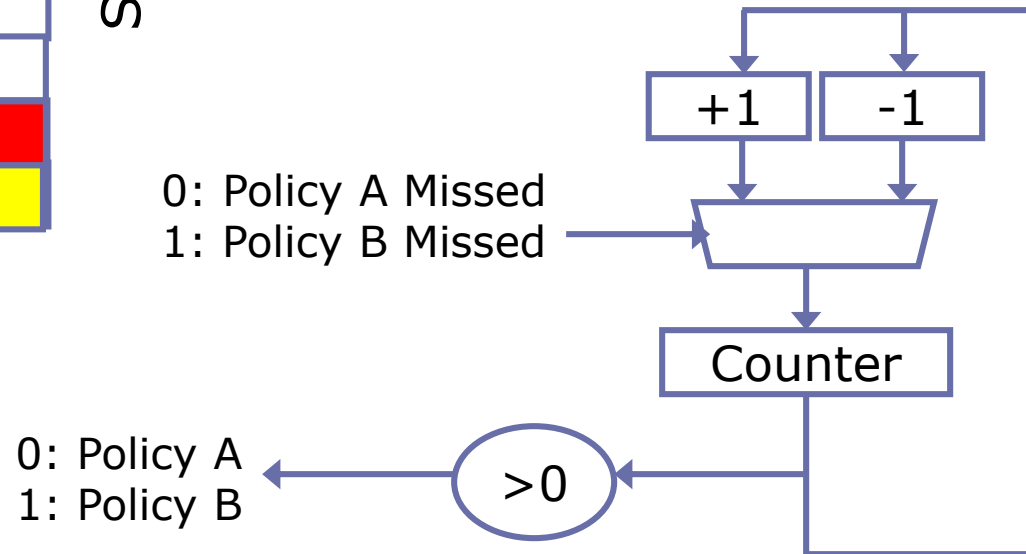
Cache

How do we decide
which policy to use?

Sets

+1   -1

0: Policy A Missed
1: Policy B Missed

Counter

Policy A

Miss

0: Policy A
1: Policy B

>0

Multiple replacement policies

Use the best replacement policy for a program

Cache

Sets

Policy A

Policy B

Miss

How do we decide which policy to use?

+1   -1

0: Policy A Missed
1: Policy B Missed

Counter

0: Policy A
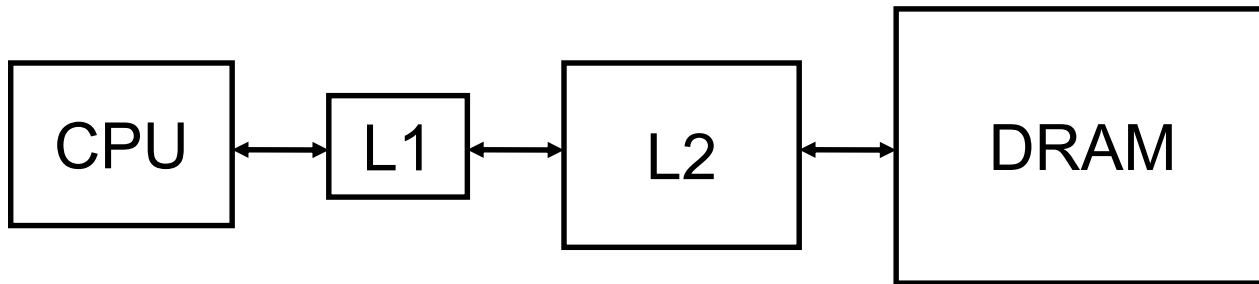1: Policy B

>0

Multilevel 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

```
CPU <-> L1 <-> L2 <-> DRAM
```

Metrics:

Local miss rate = misses in cache/ accesses to cache

Global miss rate = misses in cache / CPU memory accesses

Misses per instruction (MPI) = misses in cache / number of instructions

Inclusion Policy

- Inclusive multilevel cache:
  - Inner cache holds copies of data in outer cache
  - On miss, line inserted in inner and outer cache; replacement in outer cache invalidates line in inner cache
  - External accesses need only check outer cache
  - Commonly used (e.g., Intel CPUs up to Broadwell)

- Non-inclusive multilevel caches:
  - Inner cache may hold data not in outer cache
  - Replacement in outer cache doesn't invalidate line in inner cache
  - Used in Intel Skylake, ARM

- Exclusive multilevel caches:
  - Inner cache and outer cache hold different data
  - Swap lines between inner/outer caches on miss
  - Used in AMD processors

Inclusion Policy

- ## Inclusive multilevel cache:
  - Inner cache holds copies of data in outer cache
  - On miss, line inserted in inner and outer cache; replacement in outer cache invalidates line in inner cache
  - External accesses need only check outer cache
  - Commonly used (e.g., Intel CPUs up to Broadwell)

- ## Non-inclusive multilevel caches:
  - Inner cache may hold data not in outer cache
  - Replacement in outer cache doesn't invalidate line in inner cache
  - Used in Intel Skylake, ARM

- ## Exclusive multilevel caches:
  - Inner cache and outer cache hold different data
  - Swap lines between inner/outer caches on miss
  - Used in AMD processors

Why choose one type or the other?
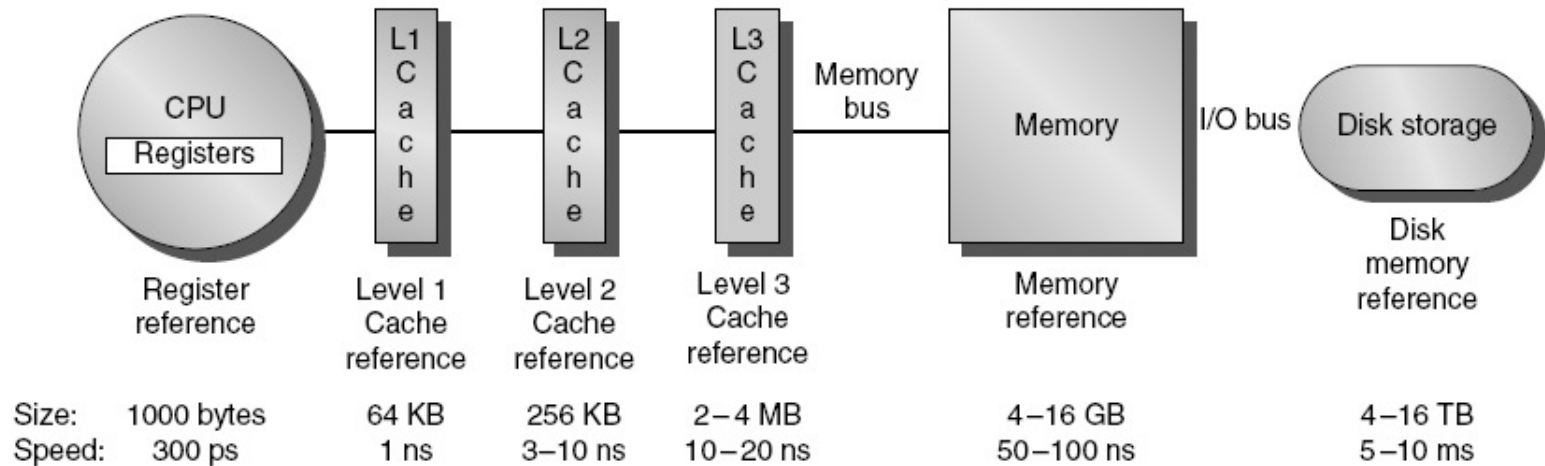
Victim Caches (HP 7200)
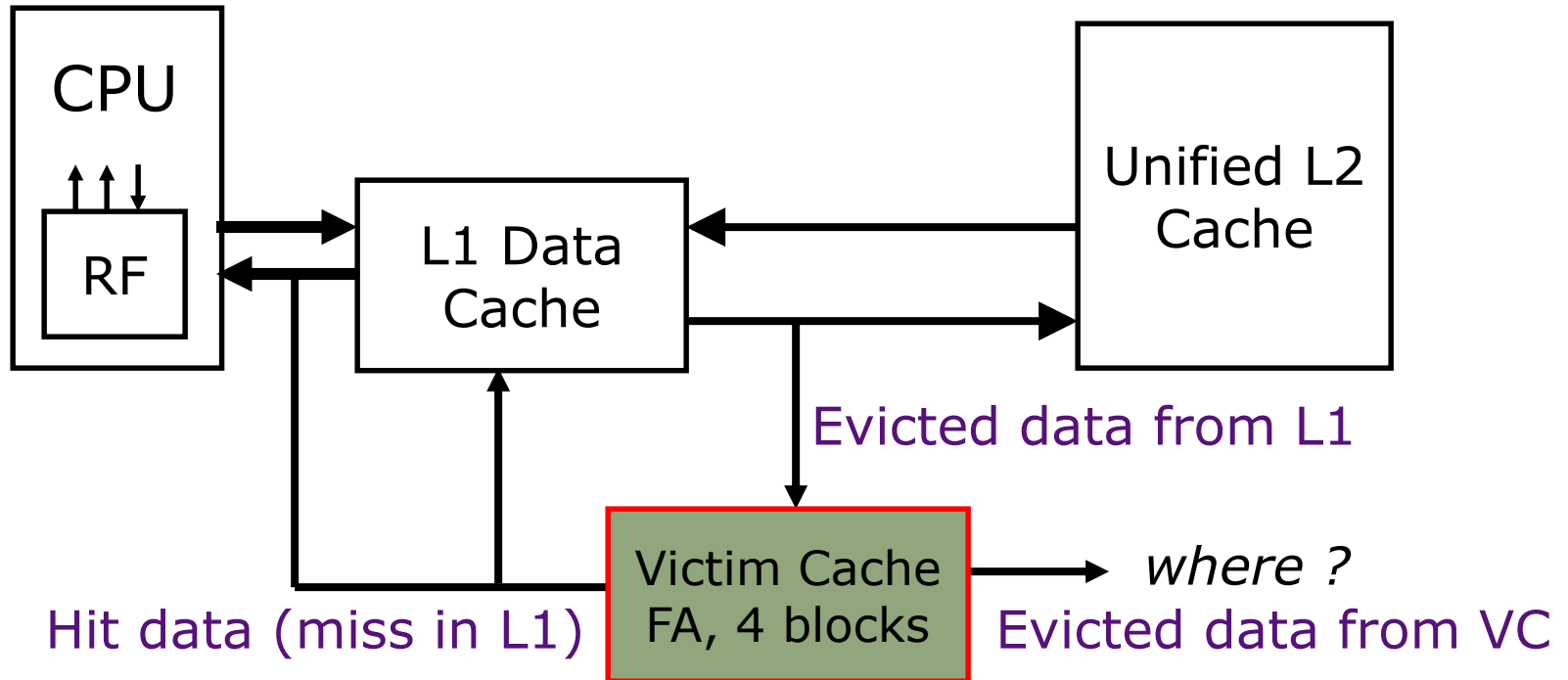
Victim cache is a small associative back up cache, added to a direct mapped cache, which holds recently evicted lines
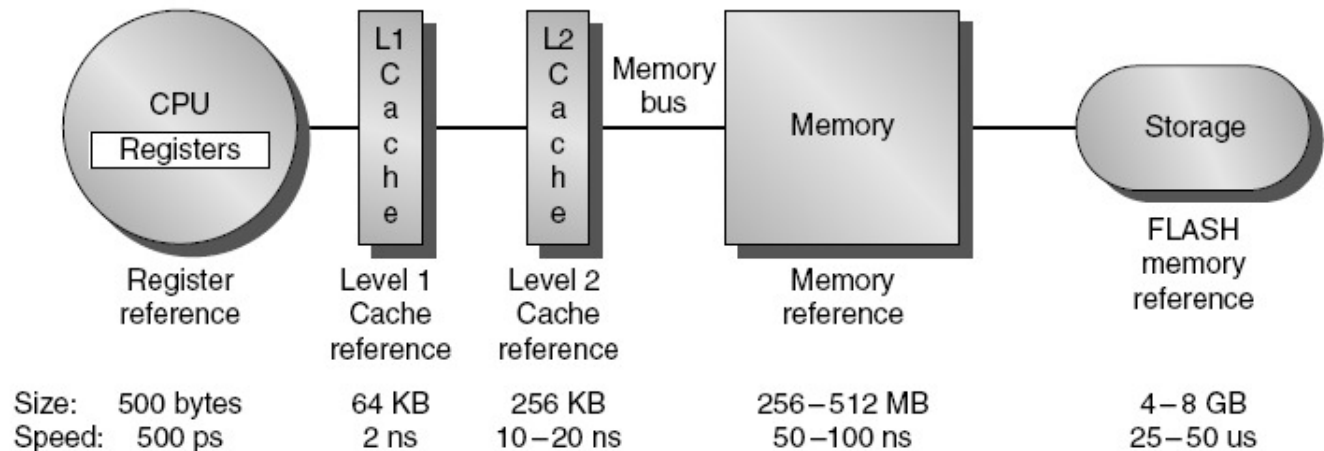• First look up in direct mapped cache
• If miss, look in victim cache
• If hit in victim cache, swap hit line with line now evicted from L1
• If miss in victim cache, L1 victim -> VC, VC victim->?
Fast hit time of direct mapped but with reduced conflict misses
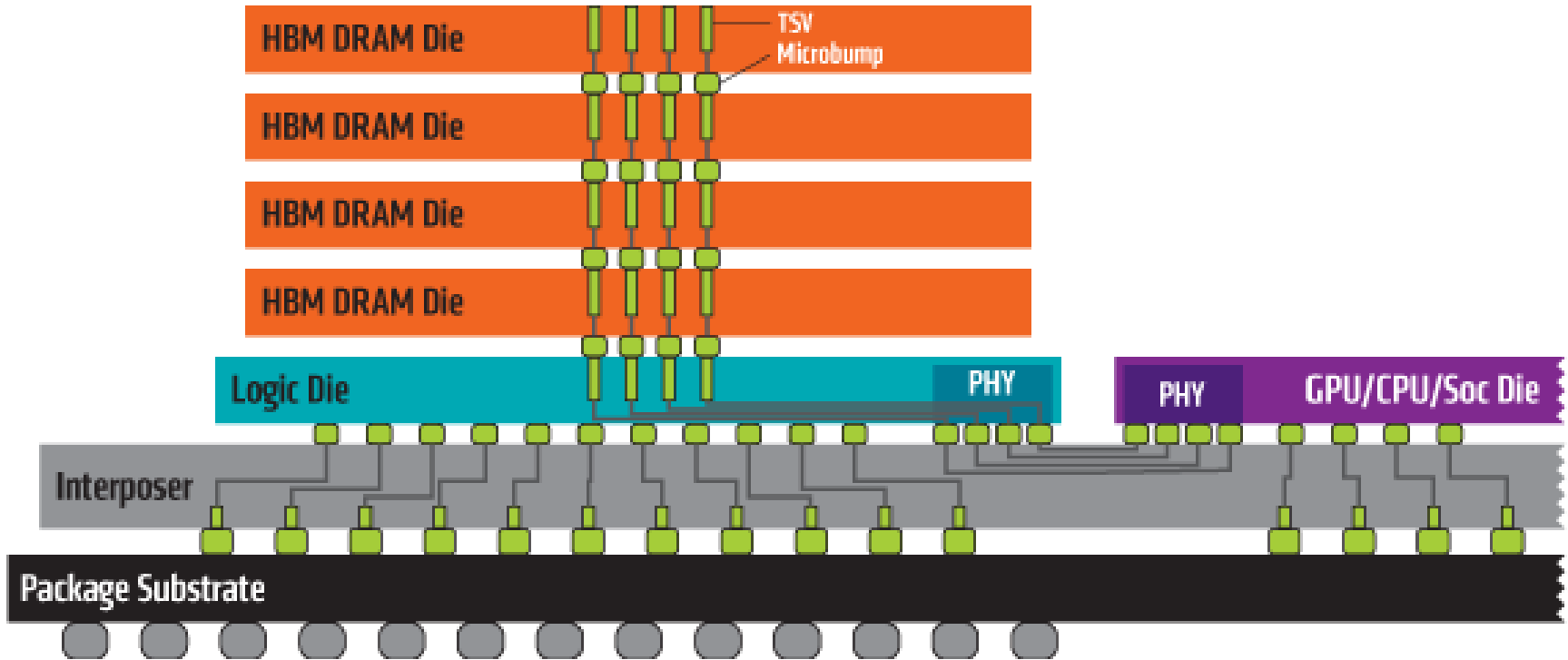
Typical memory hierarchies

| 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

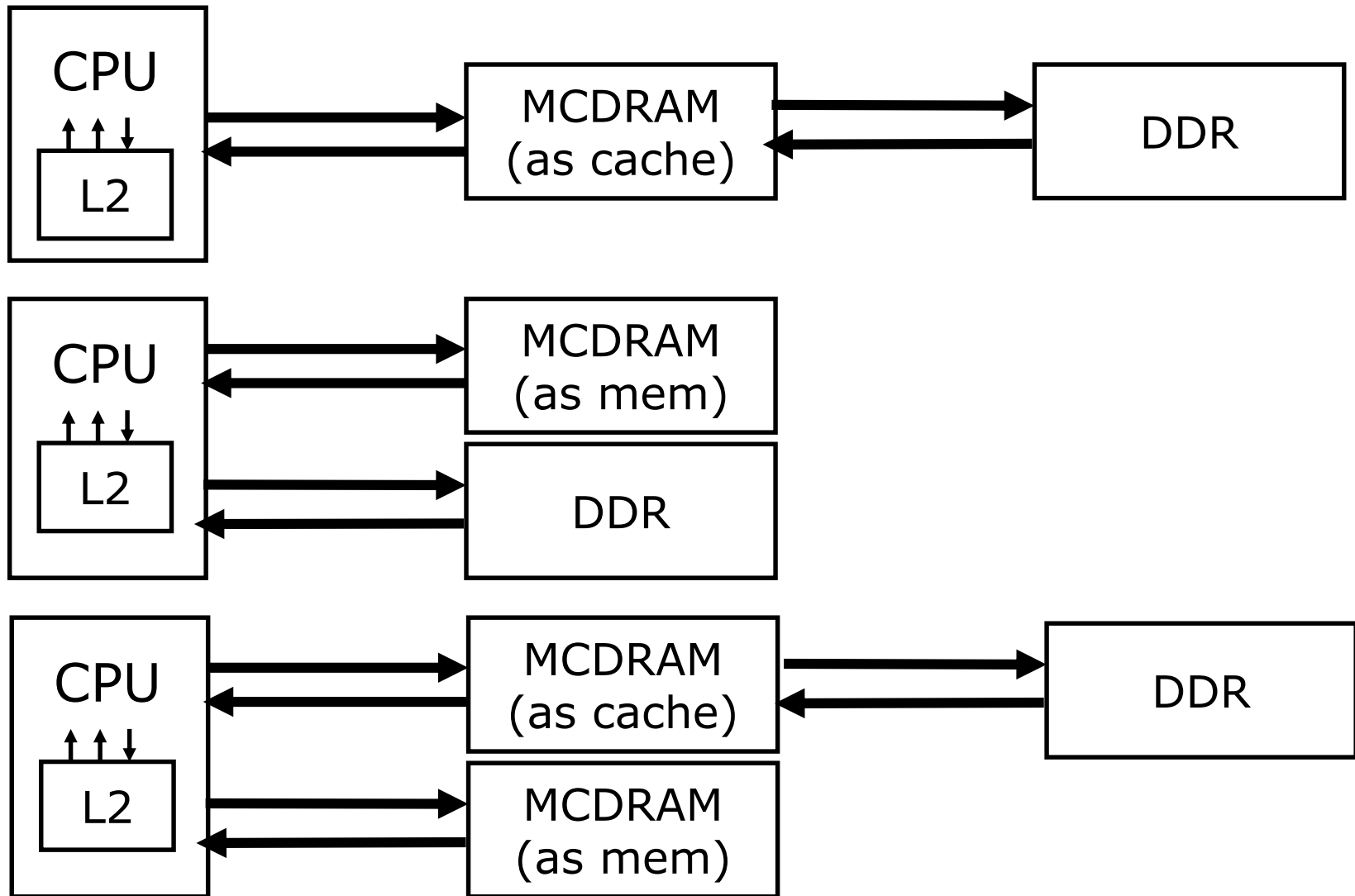| Size: | 500 bytes | 64 KB | 256 KB | 256−512 MB | 4−8 GB |
| Speed: | 500 ps | 2 ns | 10−20 ns | 50−100 ns | 25−50 us |

(b) Memory hierarchy for a personal mobile device

HBM DRAM or MCDRAM

Source: AMD

Mixed technology caching (Intel Knights Landing)

Thank you!

Next lecture:
Virtual memory