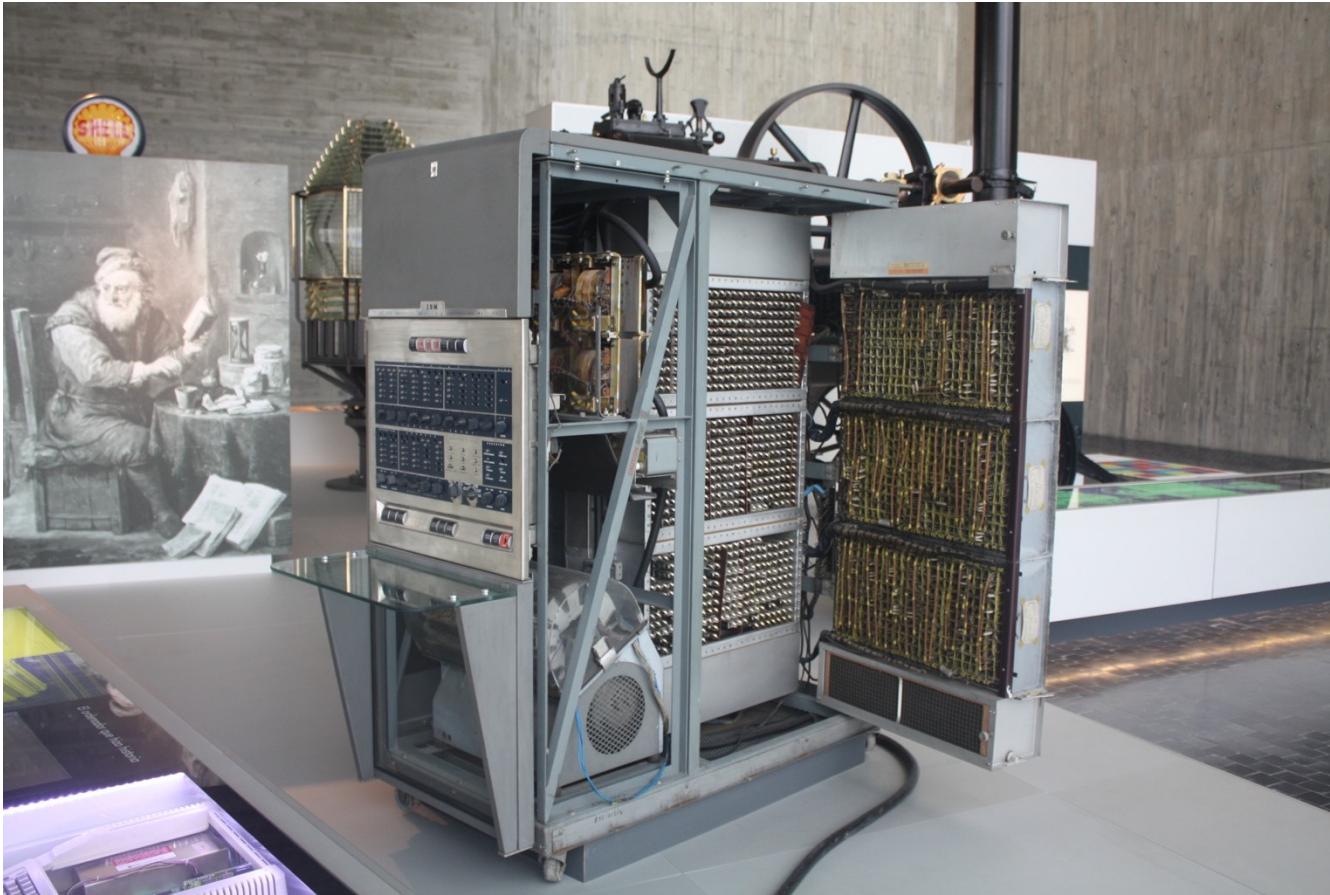# Instruction Set Architecture

*Mengjia Yan*
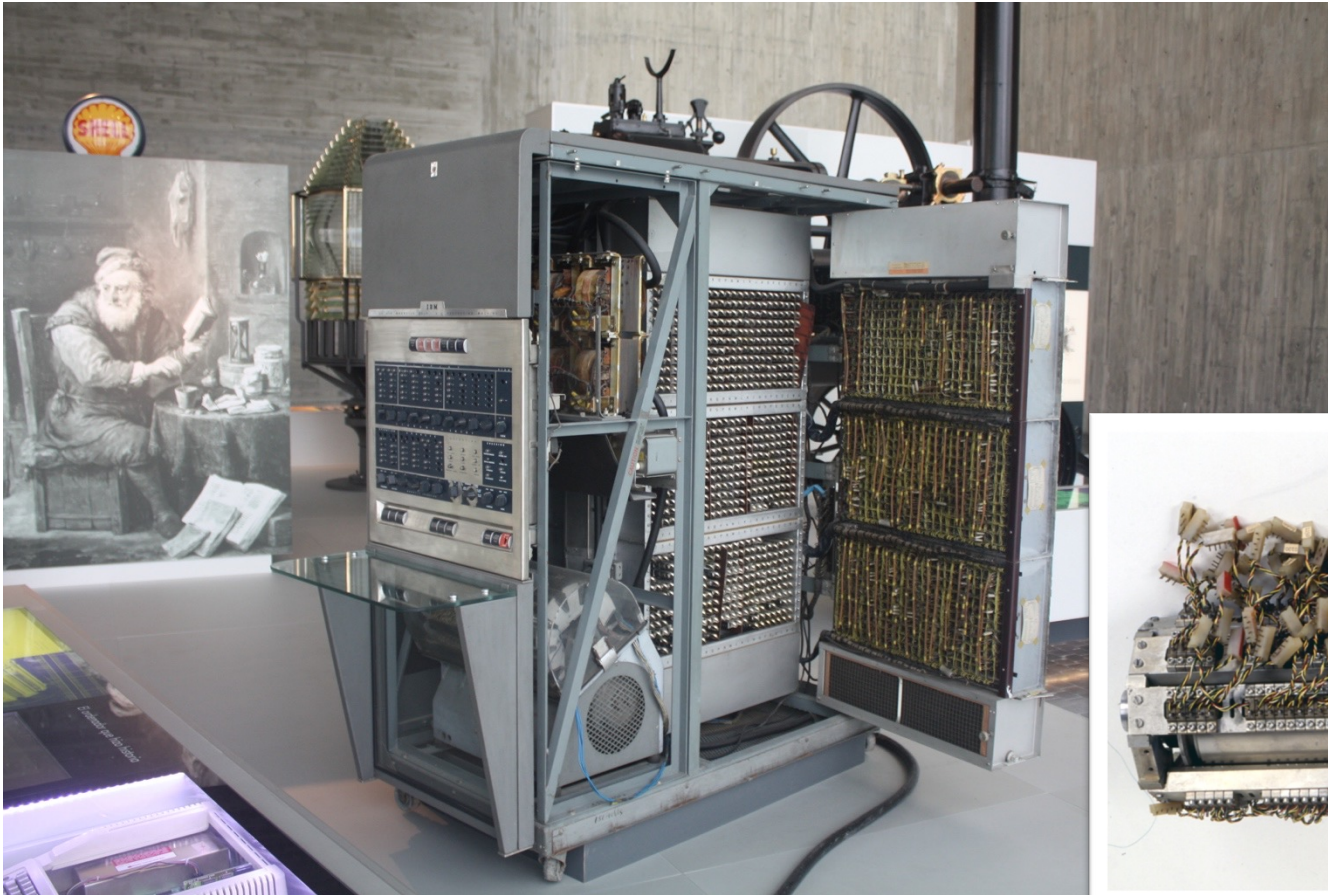Computer Science & Artificial Intelligence Lab
M.I.T.

# The IBM 650

The first mass produced computer in the world.



Side view of an IBM 650 Console Unit. First computer in Spain (1959) now at National Museum of Science and Technology in A Coruña. From Wikipedia.

# The IBM 650

The first mass produced computer in the world.



Memory drum
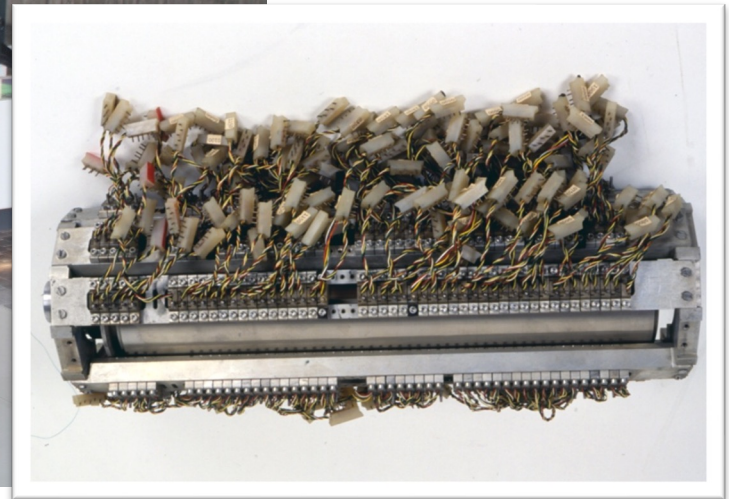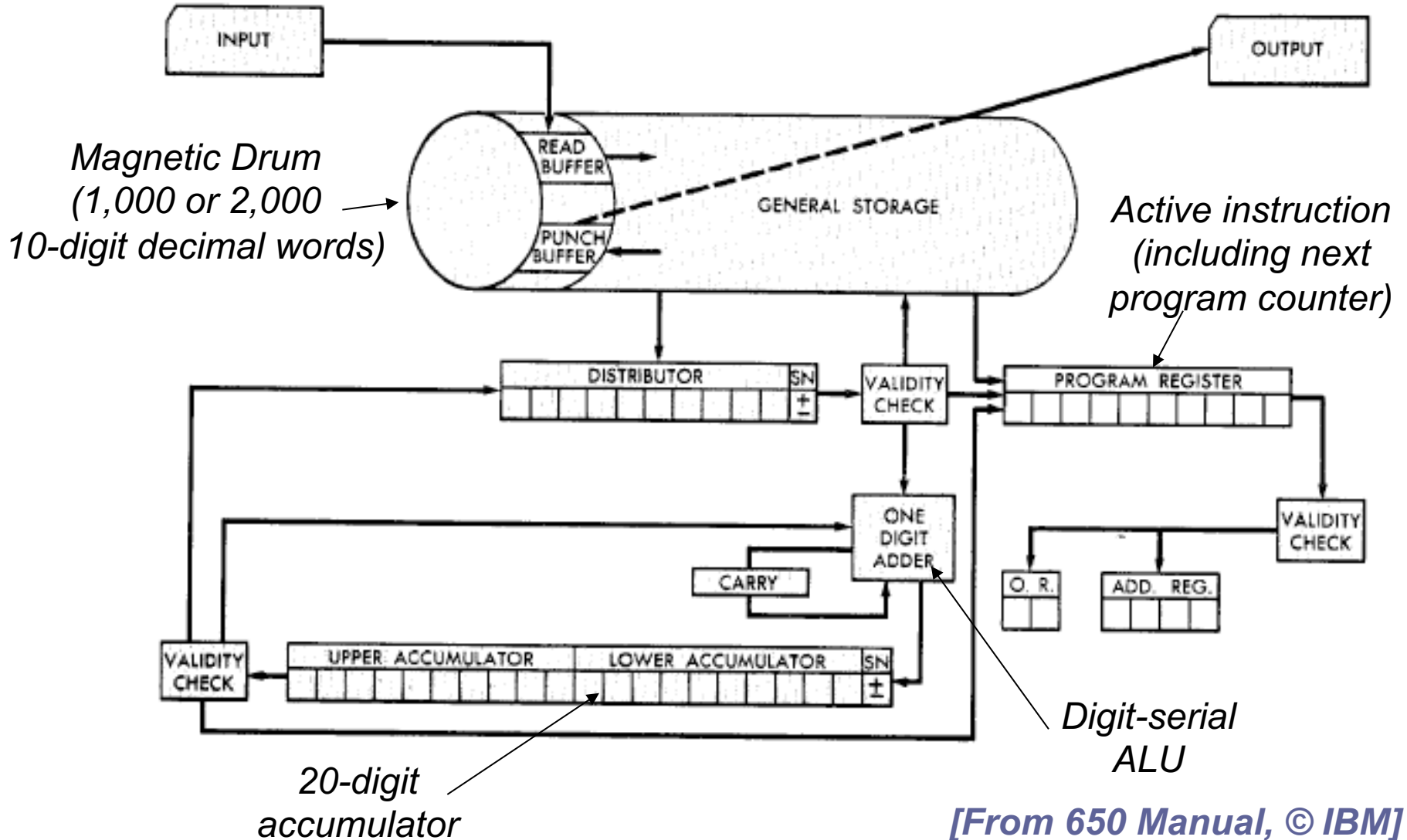
Side view of an IBM 650 Console Unit. First computer in Spain (1959) now at National Museum of Science and Technology in A Coruña. From Wikipedia.

# The IBM 650 (1953-4)



*Magnetic Drum (1,000 or 2,000 10-digit decimal words)*

*Active instruction (including next program counter)*

*Digit-serial ALU*

*20-digit accumulator*

*[From 650 Manual, © IBM]*

# Programmer's view of a machine: IBM 650

A drum machine with 44 instructions

Instruction:     60 1234 1009
    "Load the contents of location 1234 into the *distributor*;
    put it also into the *upper accumulator*; set *lower
    accumulator* to zero; and then go to location 1009 for the
    next instruction."

# Programmer's view of a machine: IBM 650

A drum machine with 44 instructions

Instruction:     60 1234 1009
    "Load the contents of location 1234 into the *distributor*; put it also into the *upper accumulator*; set *lower accumulator* to zero; and then go to location 1009 for the next instruction."

- Programmer's view of the machine was inseparable from the actual hardware implementation

# Programmer's view of a machine: IBM 650

A drum machine with 44 instructions

Instruction:    60 1234 1009
    "Load the contents of location 1234 into the *distributor*; put it also into the *upper accumulator*; set *lower accumulator* to zero; and then go to location 1009 for the next instruction."

- Programmer's view of the machine was inseparable from the actual hardware implementation
- Good programmers optimized the placement of instructions on the drum to reduce latency!

# Compatibility Problem at IBM

By early 60's, *IBM had 4 incompatible lines of computers!*

$$701 \quad \rightarrow \quad 7094$$
$$650 \quad \rightarrow \quad 7074$$
$$702 \quad \rightarrow \quad 7080$$
$$1401 \quad \rightarrow \quad 7010$$

# Compatibility Problem at IBM

By early 60's, *IBM had 4 incompatible lines of computers!*

| | | |
|---|---|---|
| 701 | → | 7094 |
| 650 | → | 7074 |
| 702 | → | 7080 |
| 1401 | → | 7010 |

Each system had its own
- Instruction set
- I/O system and Secondary Storage:
    magnetic tapes, drums and disks
- Assemblers, compilers, libraries,...
- Market niche
    business, scientific, real time, ...

# Compatibility Problem at IBM

By early 60's, *IBM had 4 incompatible lines of computers!*

| | | |
|---|---|---|
| 701 | → | 7094 |
| 650 | → | 7074 |
| 702 | → | 7080 |
| 1401 | → | 7010 |

Each system had its own
- Instruction set
- I/O system and Secondary Storage:
  magnetic tapes, drums and disks
- Assemblers, compilers, libraries,...
- Market niche
  business, scientific, real time, ...

$\Rightarrow$ *IBM 360*

# IBM 360: Design Premises
*Amdahl, Blaauw, and Brooks, 1964*

The design must lend itself to *growth and successor Machines*

# IBM 360: Design Premises
*Amdahl, Blaauw, and Brooks, 1964*

The design must lend itself to *growth and successor Machines*

- General method for connecting I/O devices
- Total performance - answers per month rather than bits per microsecond ⇒ *programming aids*
- Machine must be capable of *supervising itself* without manual intervention
- Built-in *hardware fault checking* and locating aids to reduce down time
- Simple to assemble systems with redundant I/O devices, memories, etc. for *fault tolerance*
- Some problems required floating point words larger than 36 bits

# Processor State and Data Types

*"The information held in the processor at the end of an instruction to provide the processing context for the next instruction."*

# Processor State and Data Types

*"The information held in the processor at the end of an instruction to provide the processing context for the next instruction."*

Program Counter, Accumulator, ...

- The information held in the processor will be interpreted as having data types manipulated by the instructions.

# Processor State and Data Types

*"The information held in the processor at the end of an instruction to provide the processing context for the next instruction."*

Program Counter, Accumulator, ...

- The information held in the processor will be interpreted as having data types manipulated by the instructions.

- If the processing of an instruction can be **interrupted** then the *hardware* must save and restore the state in a transparent manner

# Processor State and Data Types

*"The information held in the processor at the end of an instruction to provide the processing context for the next instruction."*

Program Counter, Accumulator, ...

- The information held in the processor will be interpreted as having data types manipulated by the instructions.

- If the processing of an instruction can be **interrupted** then the *hardware* must save and restore the state in a transparent manner

*Programmer's machine model* is a **contract** between the hardware and software

# Instruction Set

*The control for **changing** the information held in the processor are specified by the instructions available in the instruction set architecture or ISA.*

# Instruction Set

*The control for **changing** the information held in the processor are specified by the instructions available in the instruction set architecture or ISA.*

Some things an ISA must specify:
- *A way to reference registers and memory*
- *The computational operations available*
- *How to control the sequence of instructions*

# Instruction Set

*The control for **changing** the information held in the processor are specified by the instructions available in the instruction set architecture or ISA.*

Some things an ISA must specify:
- *A way to reference registers and memory*
- *The computational operations available*
- *How to control the sequence of instructions*

- *A binary representation for all of the above*

# Instruction Set

*The control for **changing** the information held in the processor are specified by the instructions available in the instruction set architecture or ISA.*

Some things an ISA must specify:
- *A way to reference registers and memory*
- *The computational operations available*
- *How to control the sequence of instructions*

- *A binary representation for all of the above*

*ISA must satisfy the needs of the software:*
*- assembler, compiler, OS, VM*

# IBM 360: *A General-Purpose Register (GPR) Machine*

- Processor State
  - 16 General-Purpose 32-bit Registers
  - 4 Floating Point 64-bit Registers
  - A Program Status Word (PSW)
    - *PC, Condition codes, Control flags*

- Data Formats
  - 8-bit bytes, 16-bit half-words, 32-bit words, 64-bit double-words
  - 24-bit addresses

- A 32-bit machine with 24-bit addresses
  - *No instruction contains a 24-bit address!*

- Precise interrupts

# IBM 360: Initial Implementations (1964)

|  | *Model 30* | *. . .* | *Model 70* |
|---|---|---|---|
| *Memory Capacity* | 8K - 64 KB | | 256K - 512 KB |
| *Memory Cycle* | 2.0μs | ... | 1.0μs |
| *Datapath* | 8-bit | | 64-bit |
| *Circuit Delay* | 30 nsec/level | | 5 nsec/level |
| *Registers* | in Main Store | | in Transistor |
| *Control Store* | Read only 1μsec | | Dedicated circuits |

- Six implementations (Models, 30, 40, 50, 60, 62, 70)
- 50x performance difference across models
- *ISA completely hid the underlying technological differences between various models*

With minor modifications, IBM 360 ISA is still in use

# IBM 360: Fifty-five years later…
# z15 Microprocessor

- 9.2 billion transistors, 12-core design
- Up to 190 cores (2 spare) per system
- 5.2 GHz, 14nm CMOS technology
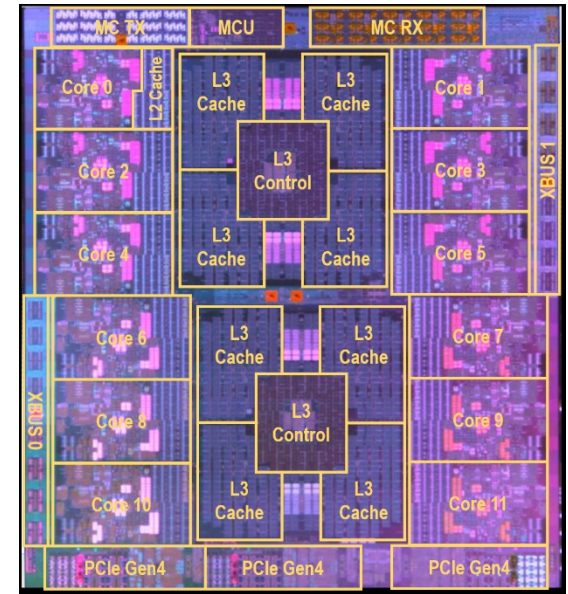


September 2019
Image credit: IBM

# IBM 360: Fifty-five years later…
# z15 Microprocessor

- 9.2 billion transistors, 12-core design
- Up to 190 cores (2 spare) per system
- 5.2 GHz, 14nm CMOS technology

- 64-bit virtual addressing
  - Original 360 was 24-bit; 370 was a 31-bit extension
- Superscalar, out-of-order
  - 12-wide issue
  - Up to 180 instructions in flight
- 16K-entry Branch Target Buffer
  - Very large buffer to support commercial workloads
- Four Levels of caches
  - 128KB L1 I-cache, 128KB L1 D-cache
  - 4MB L2 cache per core
  - 256MB shared on-chip L3 cache
  - 960MB shared off-chip L4 cache
- Up to 40TB of main memory per system
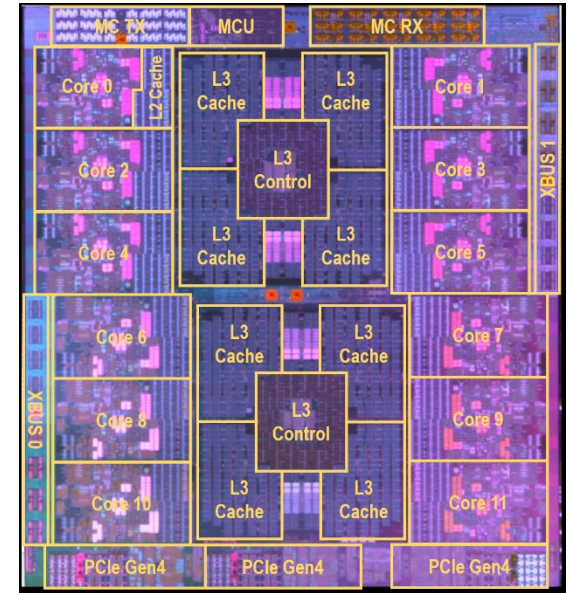


September 2019
Image credit: IBM

# Summary: Instruction Set Architecture (ISA) versus Implementation

- ISA is the hardware/software interface

- Many possible implementations of one ISA

# Summary: Instruction Set Architecture (ISA) versus Implementation

- ## ISA is the hardware/software interface
  - Examples: *MIPS, RISC-V, Alpha, x86, IBM 360, VAX, ARM, JVM*

- ## Many possible implementations of one ISA

# Summary: Instruction Set Architecture (ISA) versus Implementation

- ## ISA is the hardware/software interface
  - Examples: *MIPS, RISC-V, Alpha, x86, IBM 360, VAX, ARM, JVM*

- ## Many possible implementations of one ISA
  - 360 implementations: model 30 (c. 1964), z15 (c. 2019)
  - x86 implementations: *8086 (c. 1978), 80186, 286, 386, 486, Pentium, Pentium Pro, Pentium-4, Core i7, AMD Athlon, AMD Opteron, Transmeta Crusoe, SoftPC*
  - MIPS implementations: *R2000, R4000, R10000, ...*
  - JVM: *HotSpot, PicoJava, ARM Jazelle, ...*

# Summary: Instruction Set Architecture (ISA) versus Implementation

- ## ISA is the hardware/software interface
  - Examples: *MIPS, RISC-V, Alpha, x86, IBM 360, VAX, ARM, JVM*

- ## Many possible implementations of one ISA
  - 360 implementations: model 30 (c. 1964), z15 (c. 2019)
  - x86 implementations: *8086 (c. 1978), 80186, 286, 386, 486, Pentium, Pentium Pro, Pentium-4, Core i7,  AMD Athlon, AMD Opteron, Transmeta Crusoe, SoftPC*
  - MIPS implementations: *R2000, R4000, R10000, ...*
  - JVM: *HotSpot, PicoJava, ARM Jazelle, ...*

*What potential drawbacks with ISAs?*

# Memory and Caches

*Mengjia Yan*
Computer Science and Artificial Intelligence Laboratory
M.I.T.
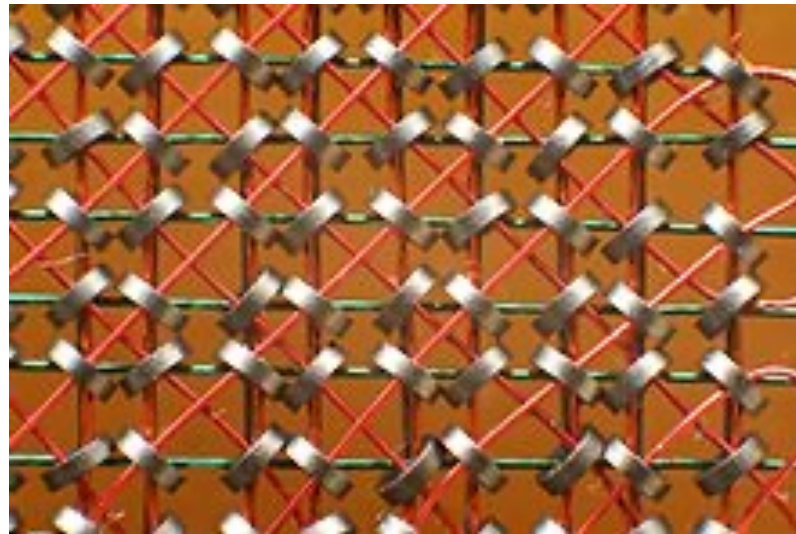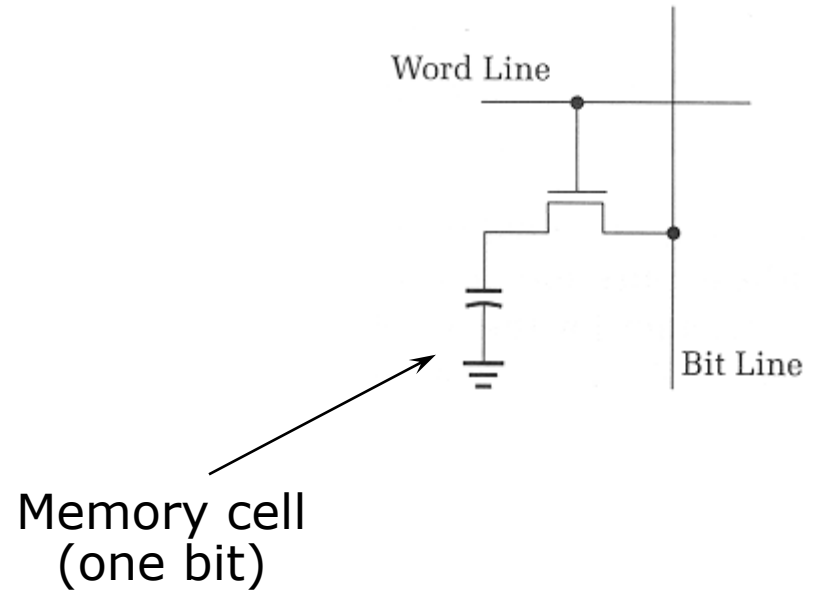
# Memory Technology

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



store acoustic signals in mercury

# Memory Technology

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





store acoustic signals in mercury

# 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
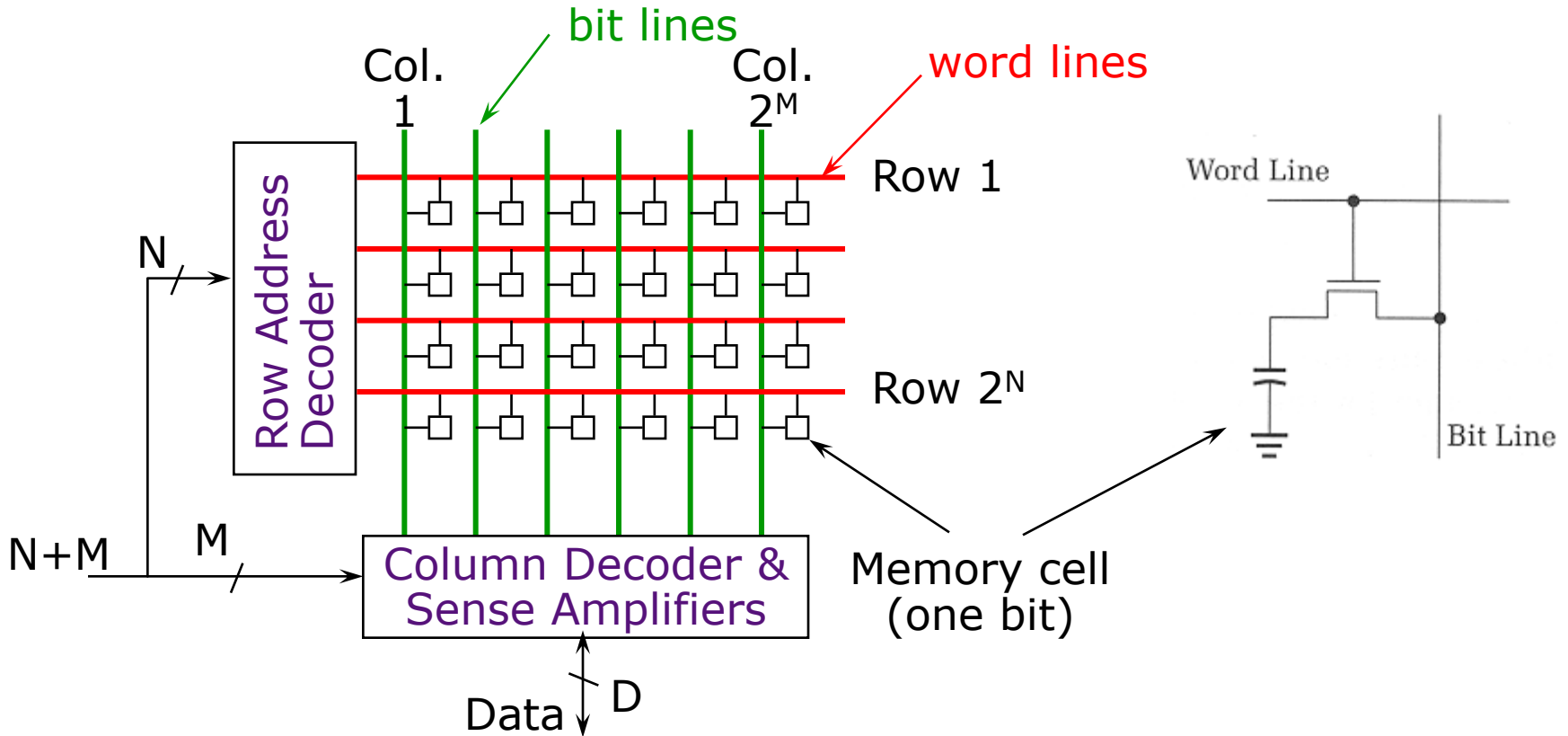
From Wikipedia

# 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

From Wikipedia

# 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



bit lines

word lines

Col. 1

Col. $2^M$

Row 1

Row $2^N$

Row Address Decoder

N

N+M

M

Column Decoder & Sense Amplifiers

Data $\downarrow$ D

Memory cell (one bit)

Word Line

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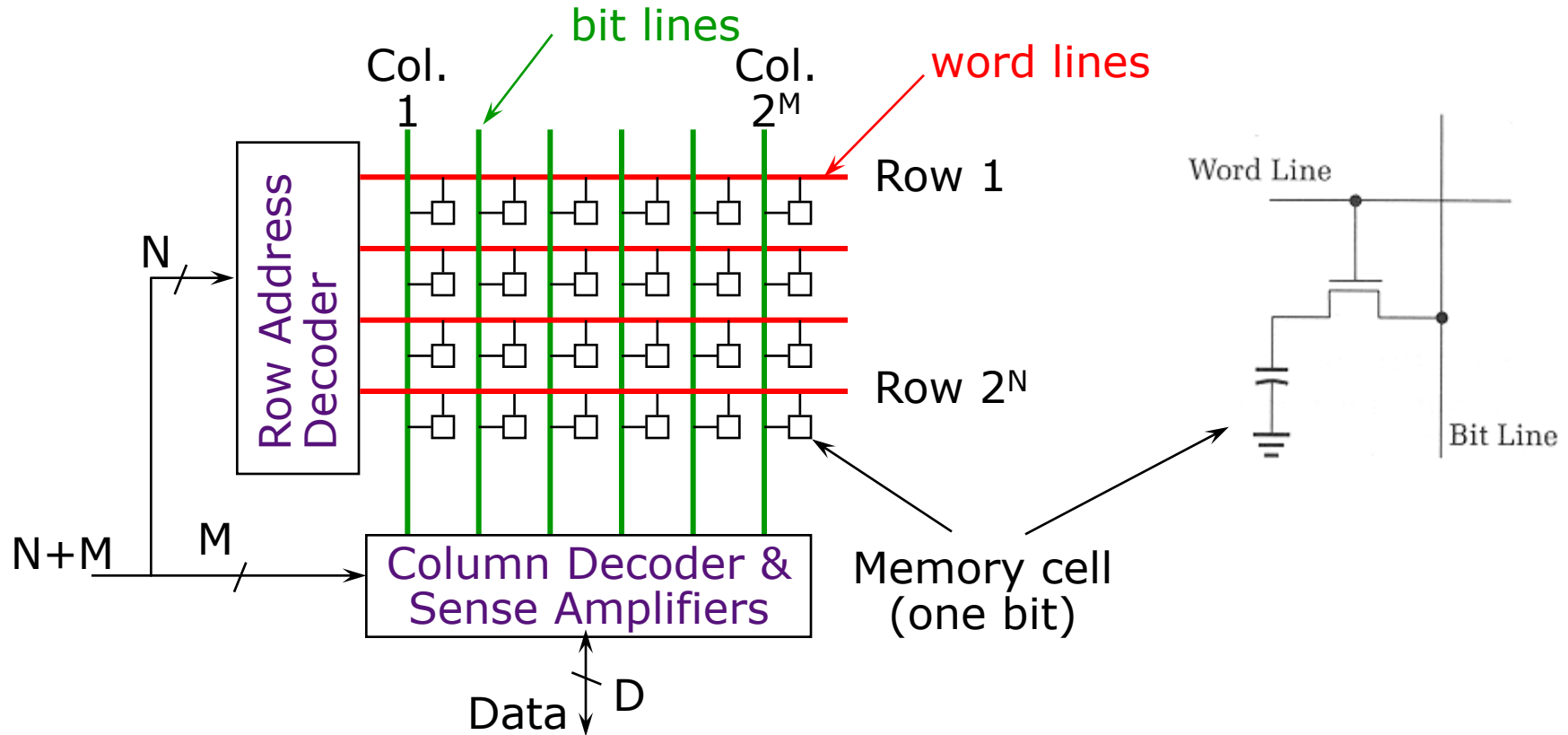
- Bits stored in 2-dimensional arrays on chip
- Modern chips have around 8 logical banks on each chip
  - Each logical bank physically implemented as many smaller arrays

# Processor-DRAM Gap (latency)
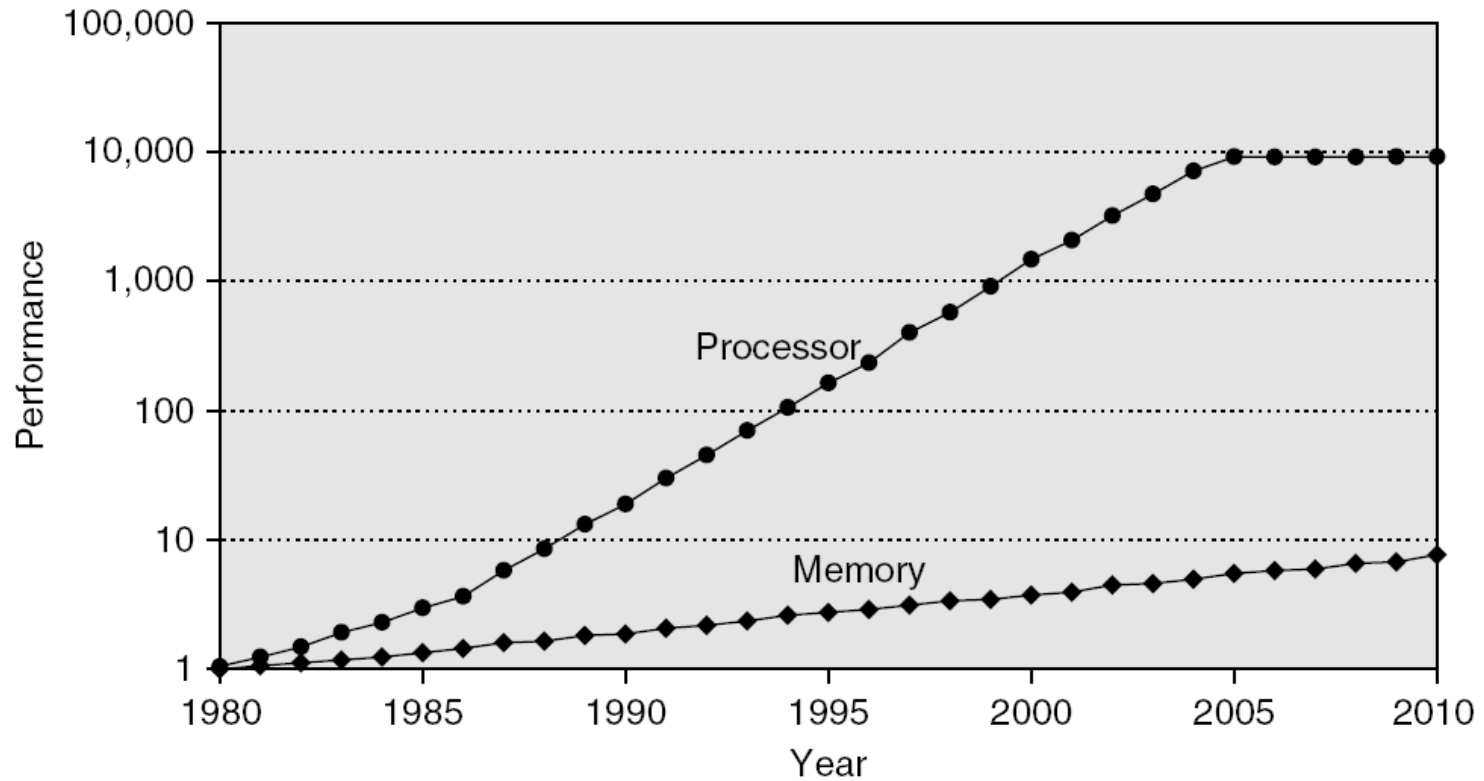
# Processor-DRAM Gap (latency)
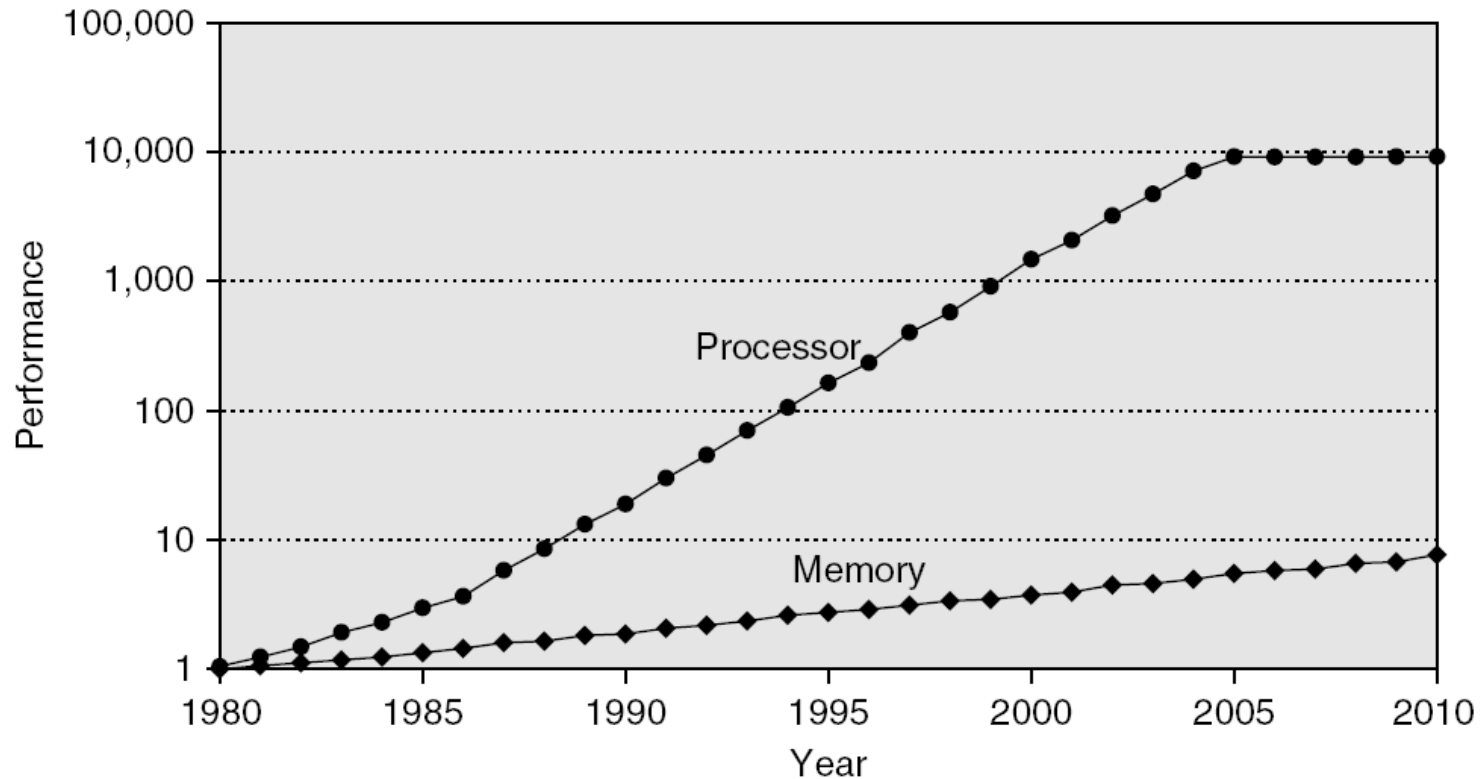


Four-issue 2GHz superscalar accessing 100ns DRAM could execute 800 instructions during time for one memory access!

# Processor-DRAM Gap (latency)
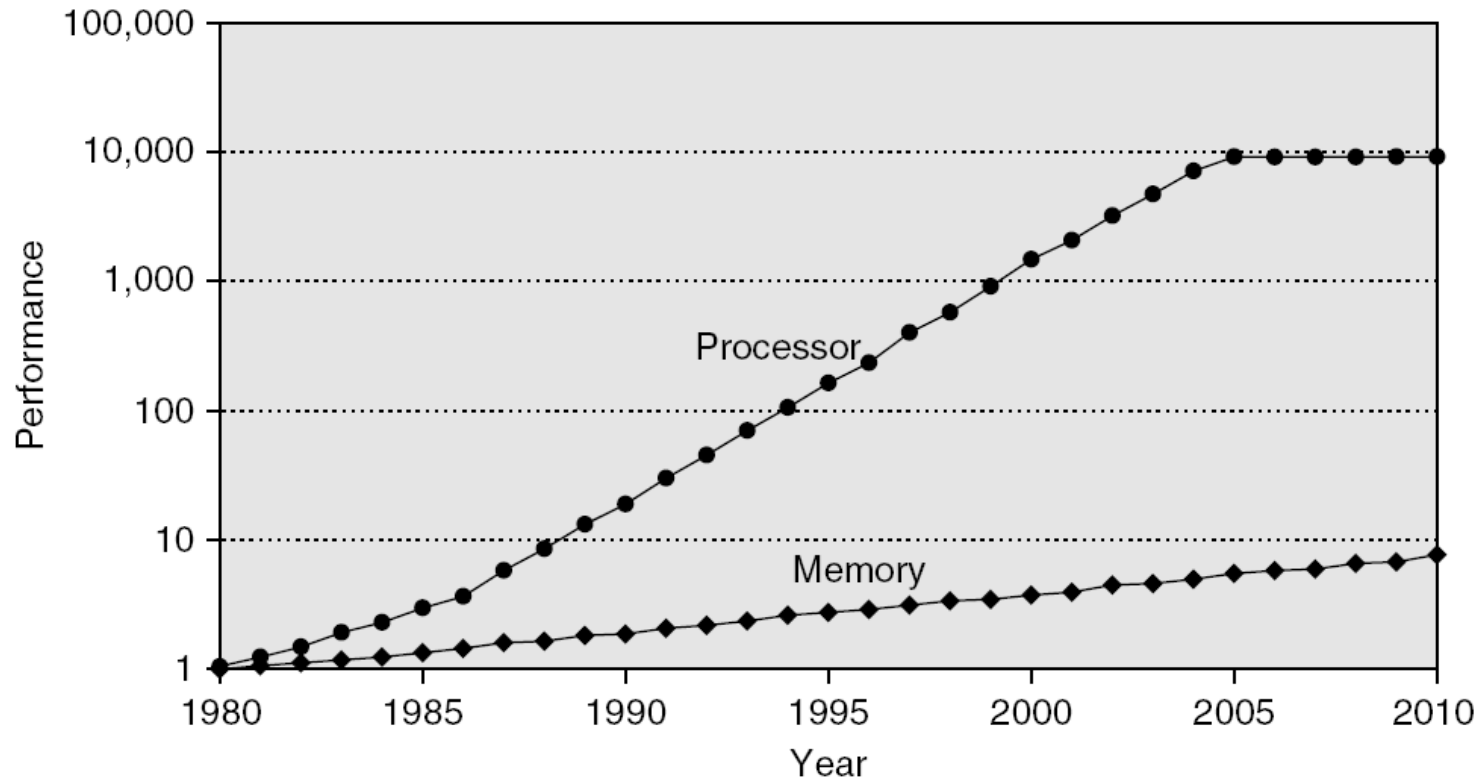


Four-issue 2GHz superscalar accessing 100ns DRAM could execute 800 instructions during time for one memory access!

How to bridge this gap?

# Little's Law

*Throughput (T) = Number in Flight (N) / Latency (L)*
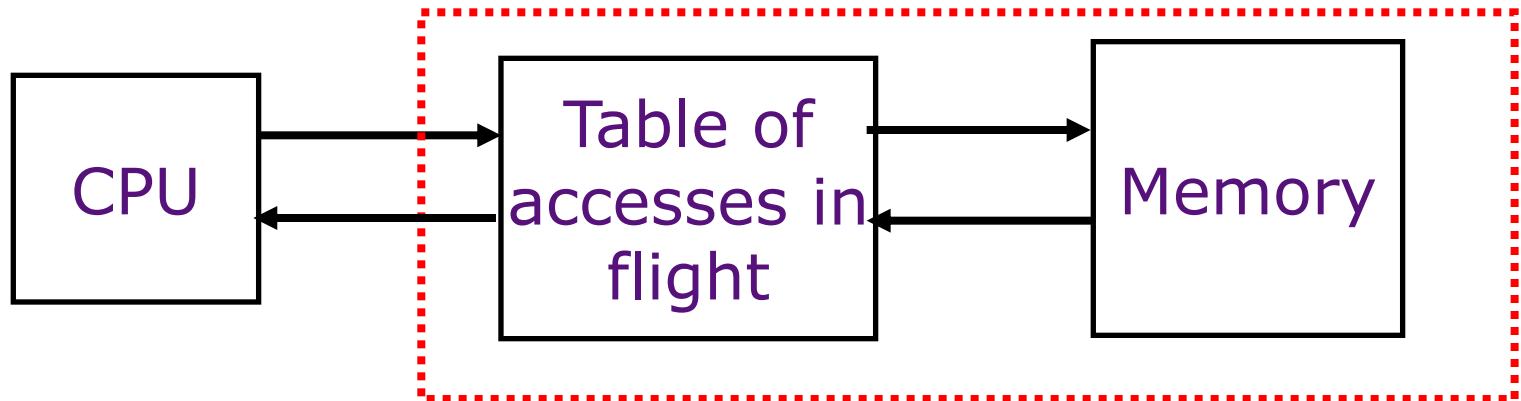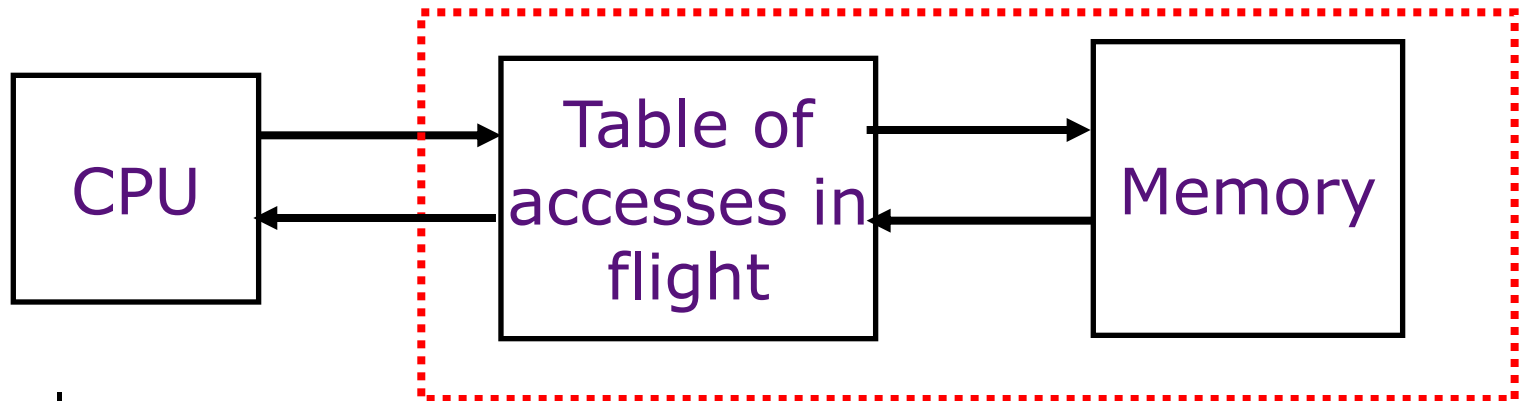
# Little's Law

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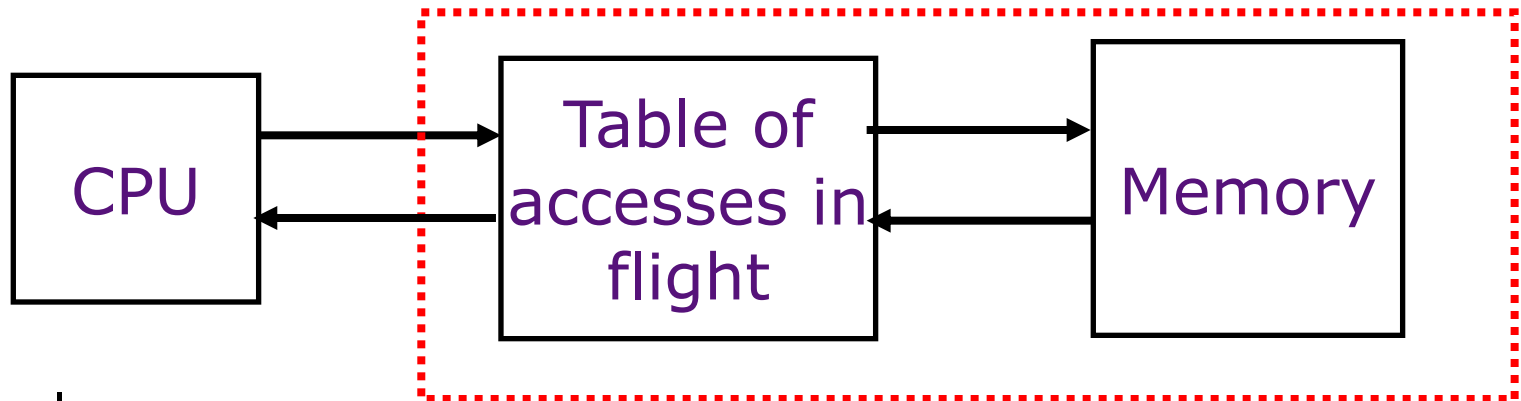


Example:

*--- Assume infinite-bandwidth memory*

*--- 100 cycles / memory reference*

*--- 1 + 0.2 memory references / instruction*

How many inflight accesses do we need to hold to keep the CPU busy?

# 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*

How many inflight accesses do we need to hold to keep the CPU busy?

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

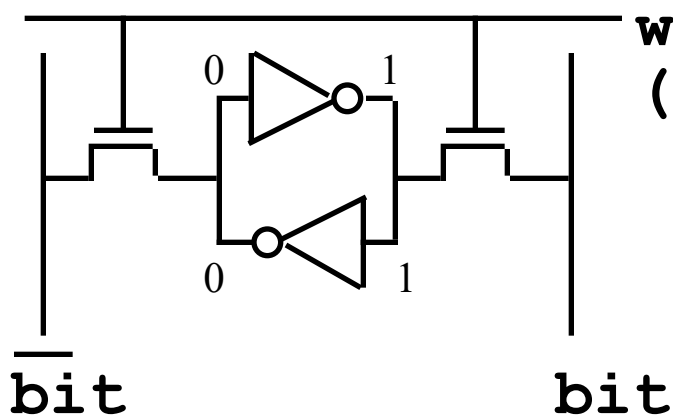### 120 independent memory operations in flight!

# Basic Static RAM Cell

**6-Transistor SRAM Cell**

word
(row select)

$\overline{\text{bit}}$     bit
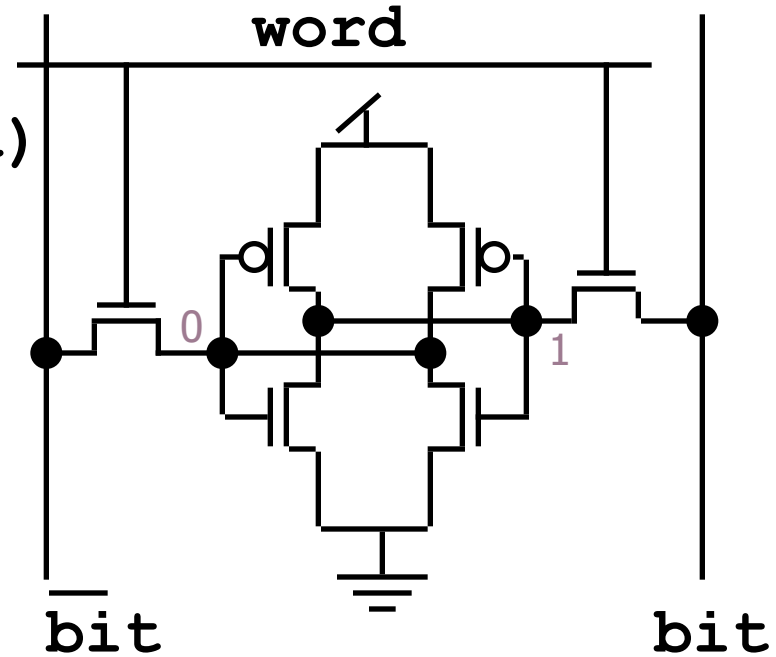
word

$\overline{\text{bit}}$     bit
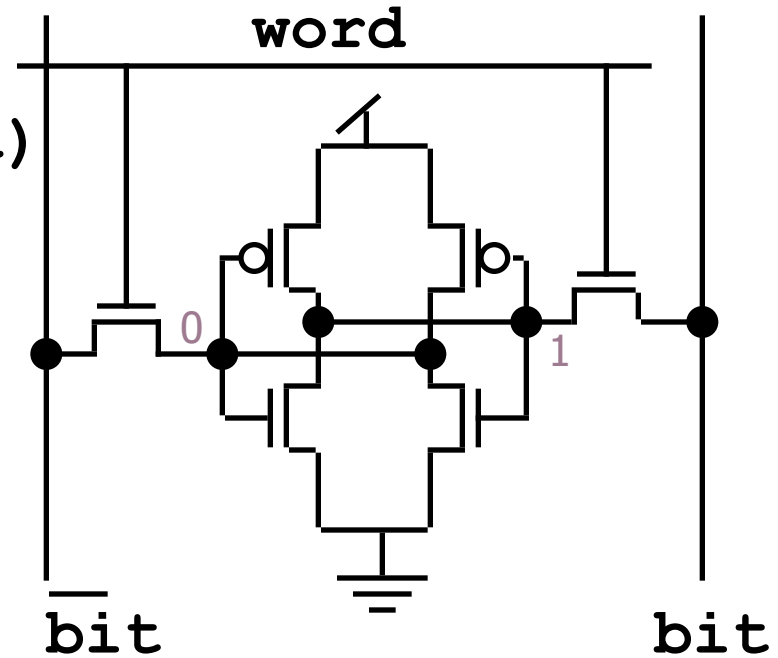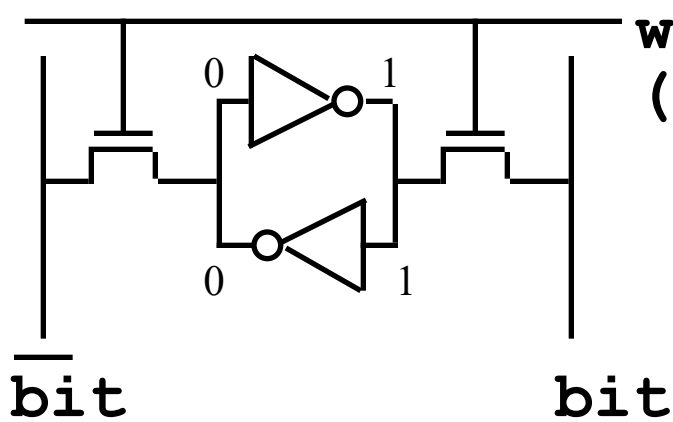
# Basic Static RAM Cell

**6-Transistor SRAM Cell**



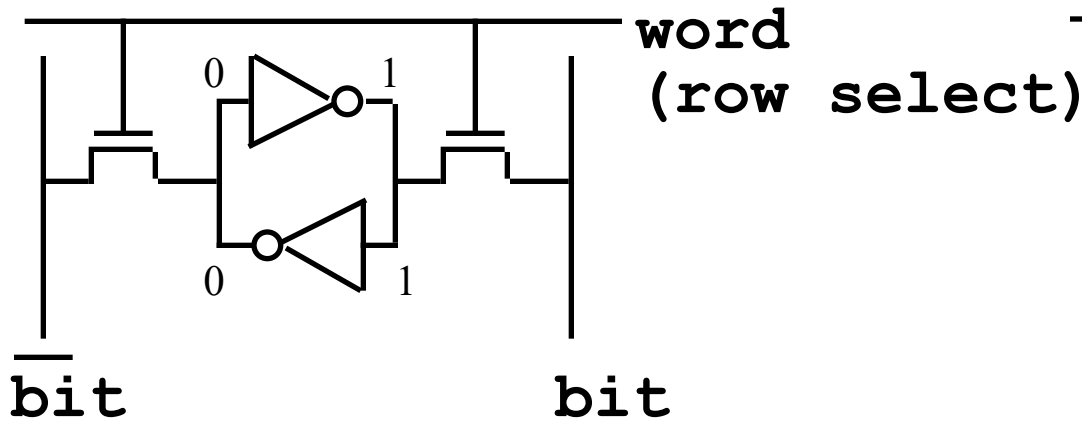- Write:
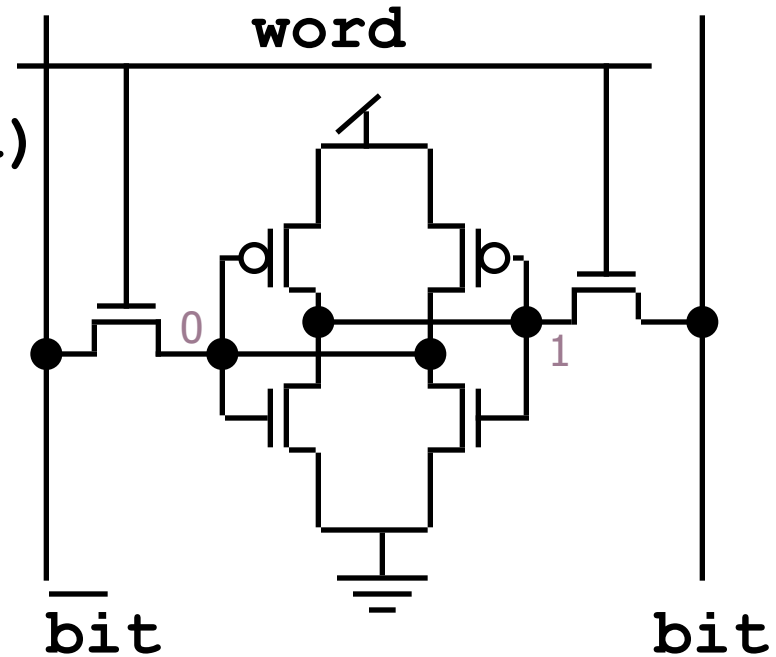    1. Drive bit lines (bit=1, $\overline{\text{bit}}$=0)
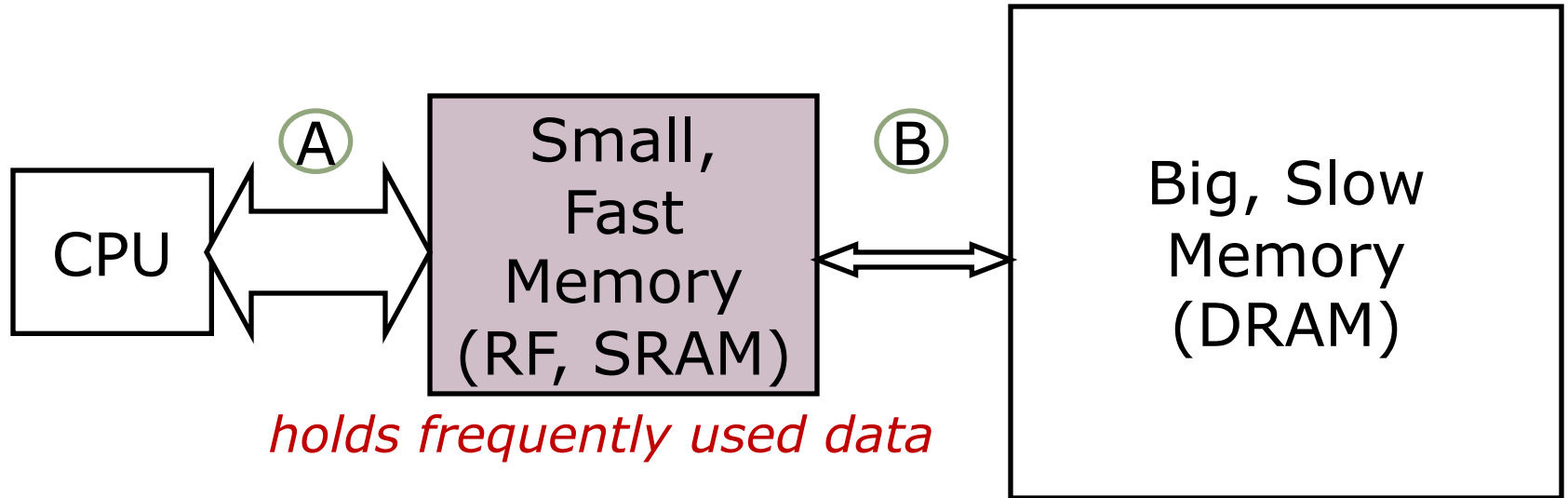    2. Select word line

# Basic Static RAM Cell

**6-Transistor SRAM Cell**



**word (row select)**

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

**word**

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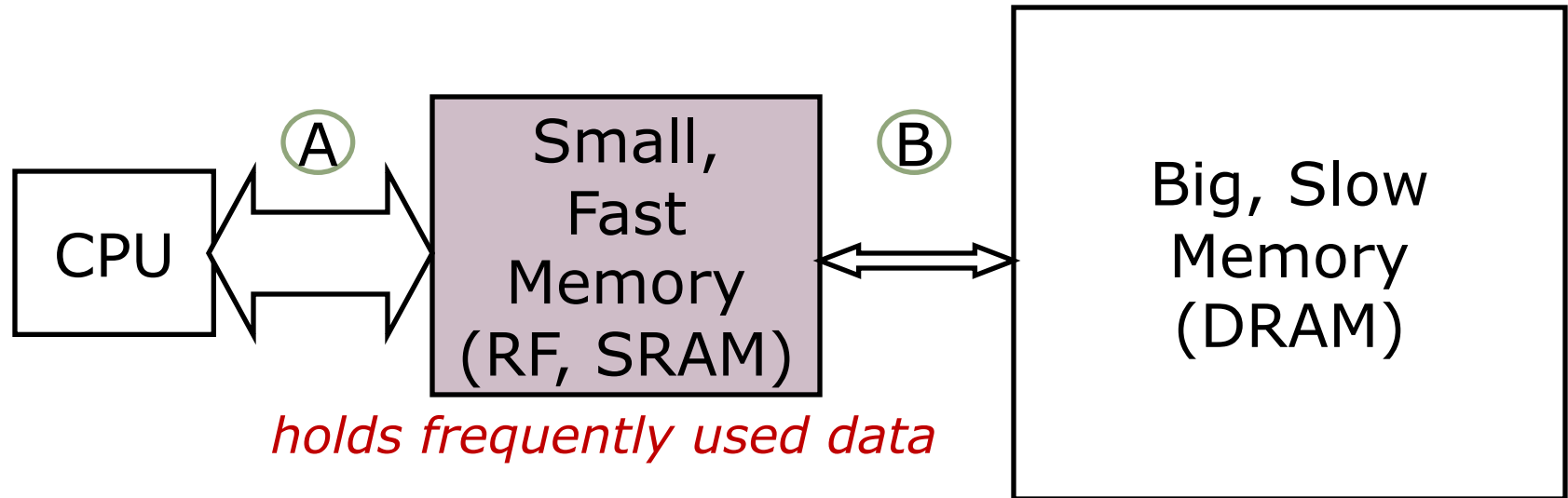
- Write:
    1. Drive bit lines (bit=1, $\overline{bit}$=0)
    2. Select word line

- Read:
    1. Precharge bit and $\overline{bit}$ to Vdd
    2. Select word line
    3. Cell pulls one bit line low
    4. Column sense amp detects difference between bit & $\overline{bit}$

# The Concept of Caching



holds frequently used data

# The Concept of Caching



On a data access:

 data ∈ fast memory        ⇒ low latency access
 data ∉ fast memory        ⇒ long latency access *(DRAM)*

# The Concept of Caching
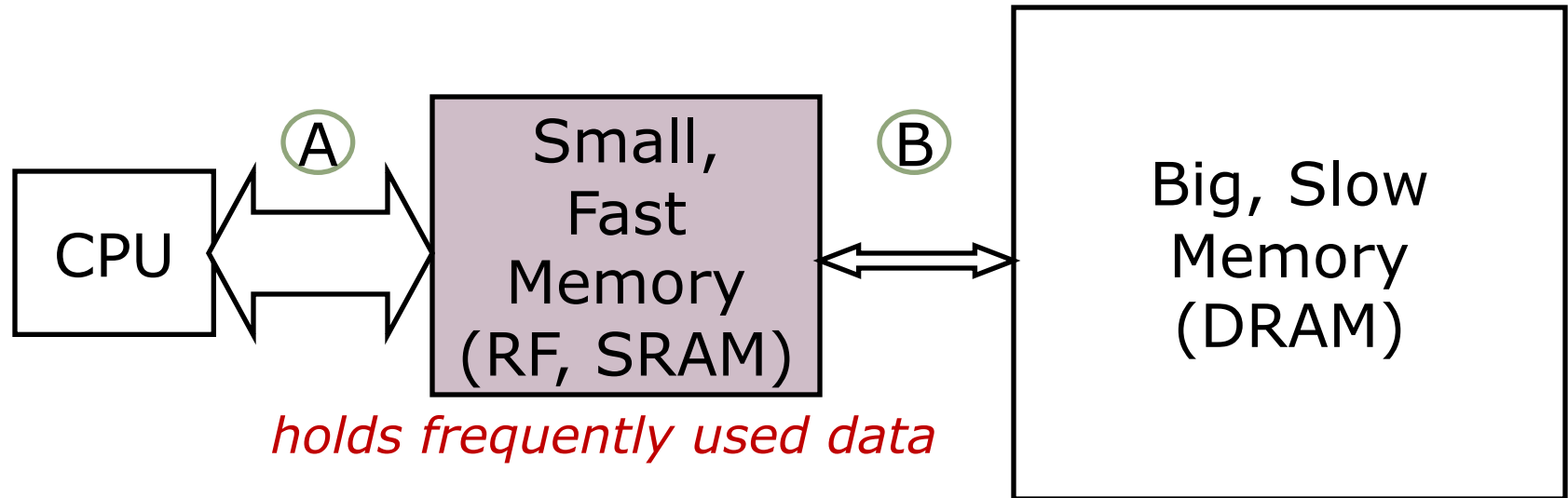


**On a data access:**

   data ∈ fast memory          ⇒ low latency access
   data ∉ fast memory          ⇒ long latency access *(DRAM)*

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

# 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:
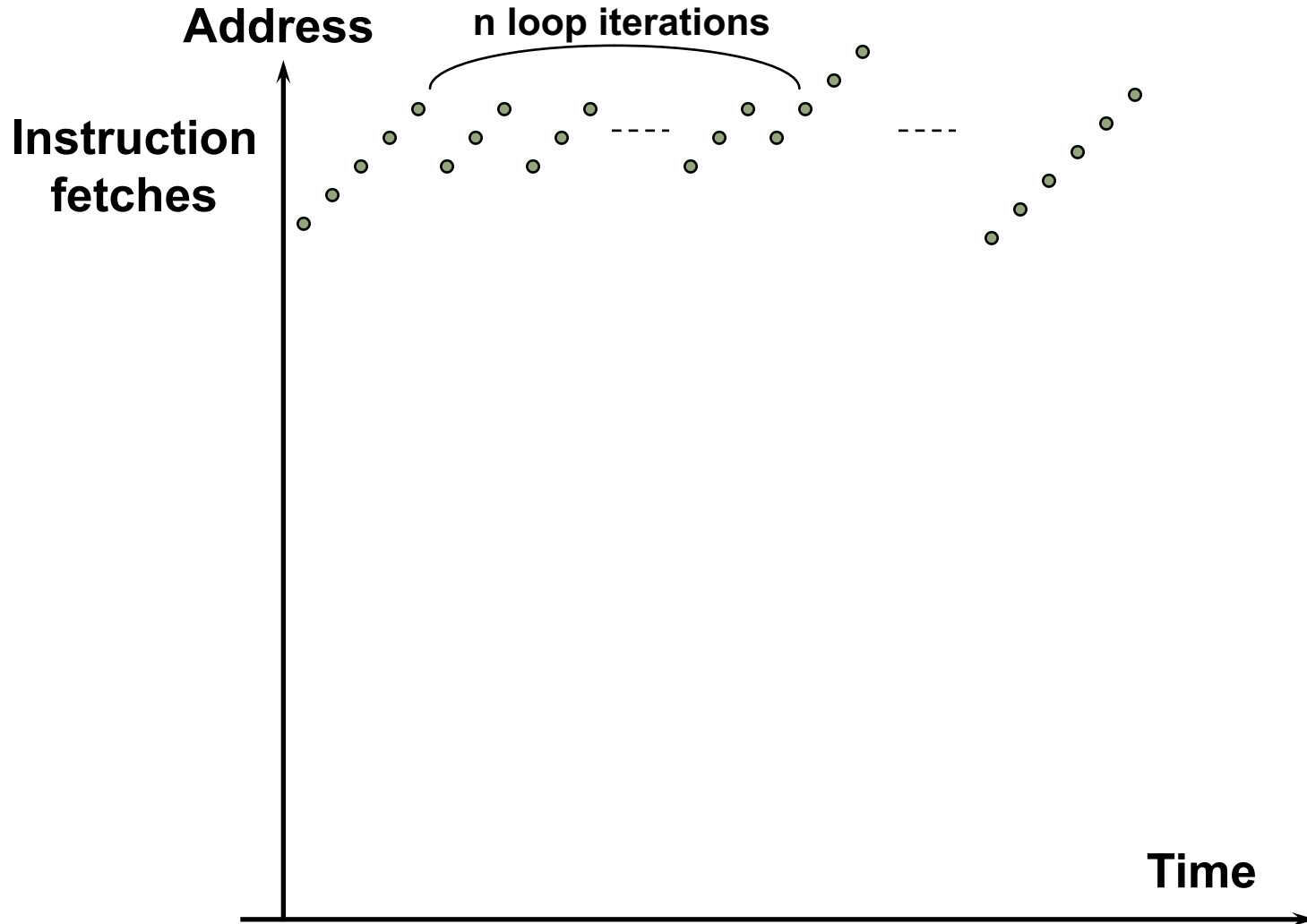
# Multilevel Memory

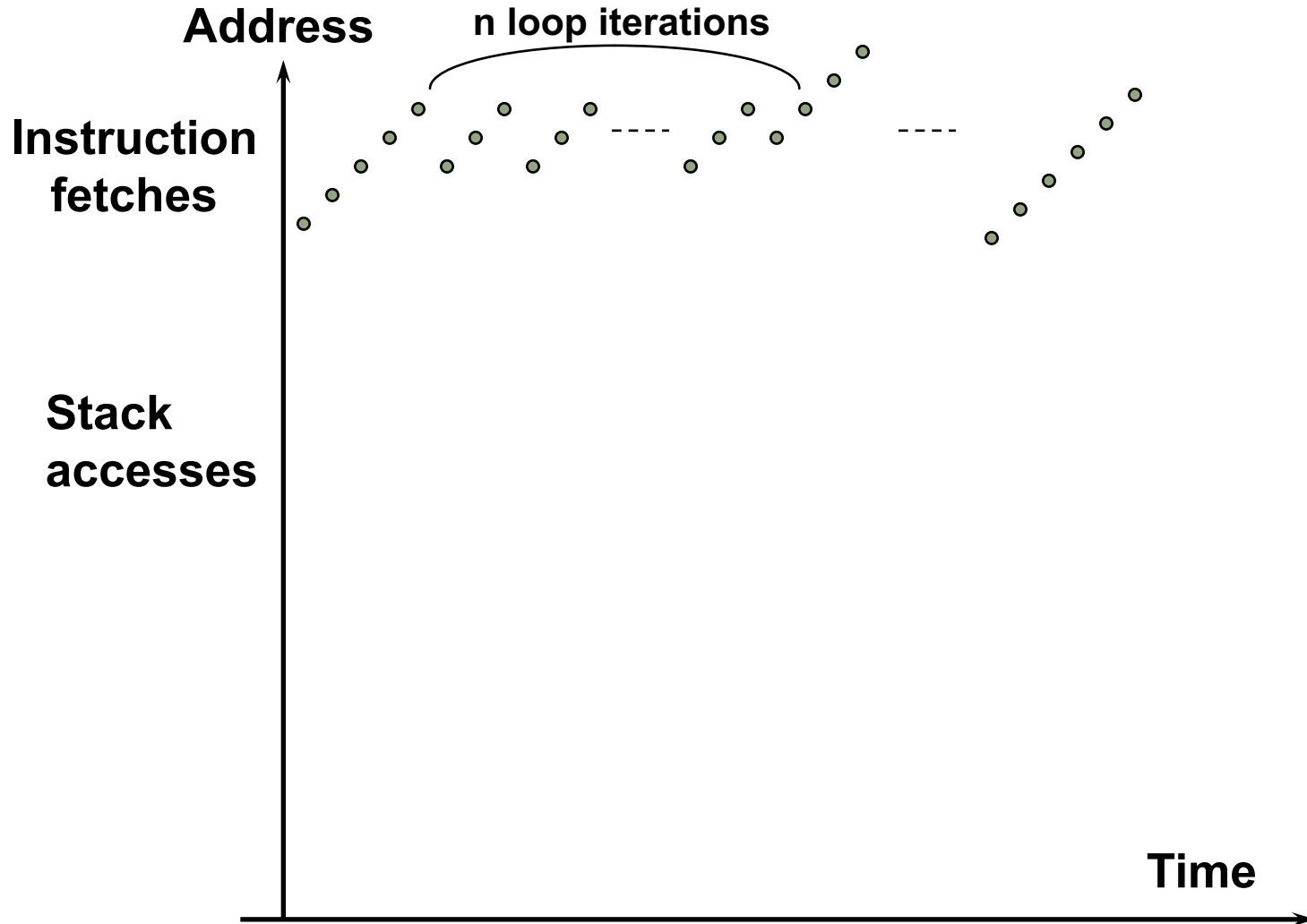Strategy: <u>Reduce</u> average latency using small, fast memories called caches.

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   x2, x1, x1      100
        addi  x3, x3, -1      104
        bne   x3, x0, Loop    108
        …                     112
```
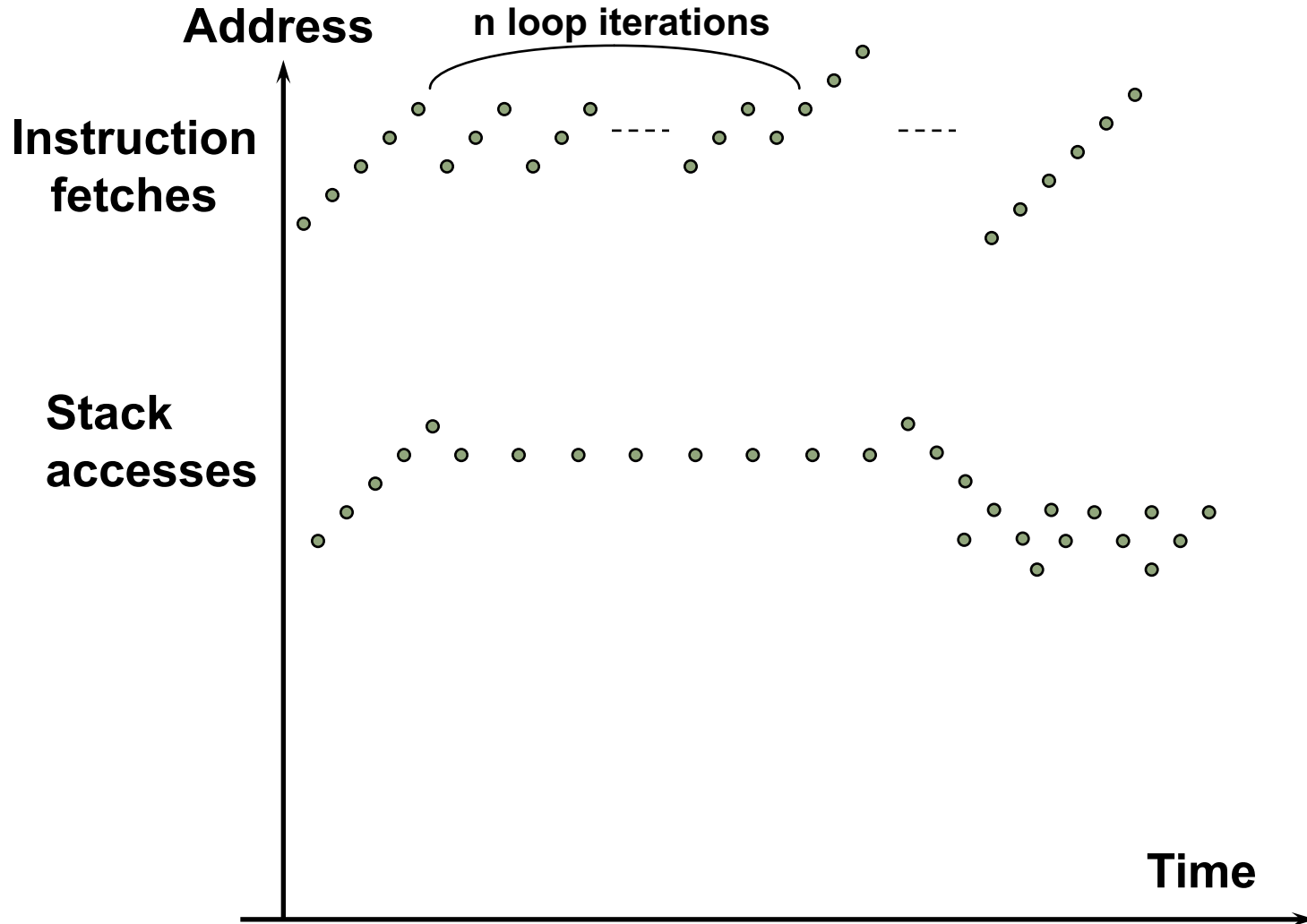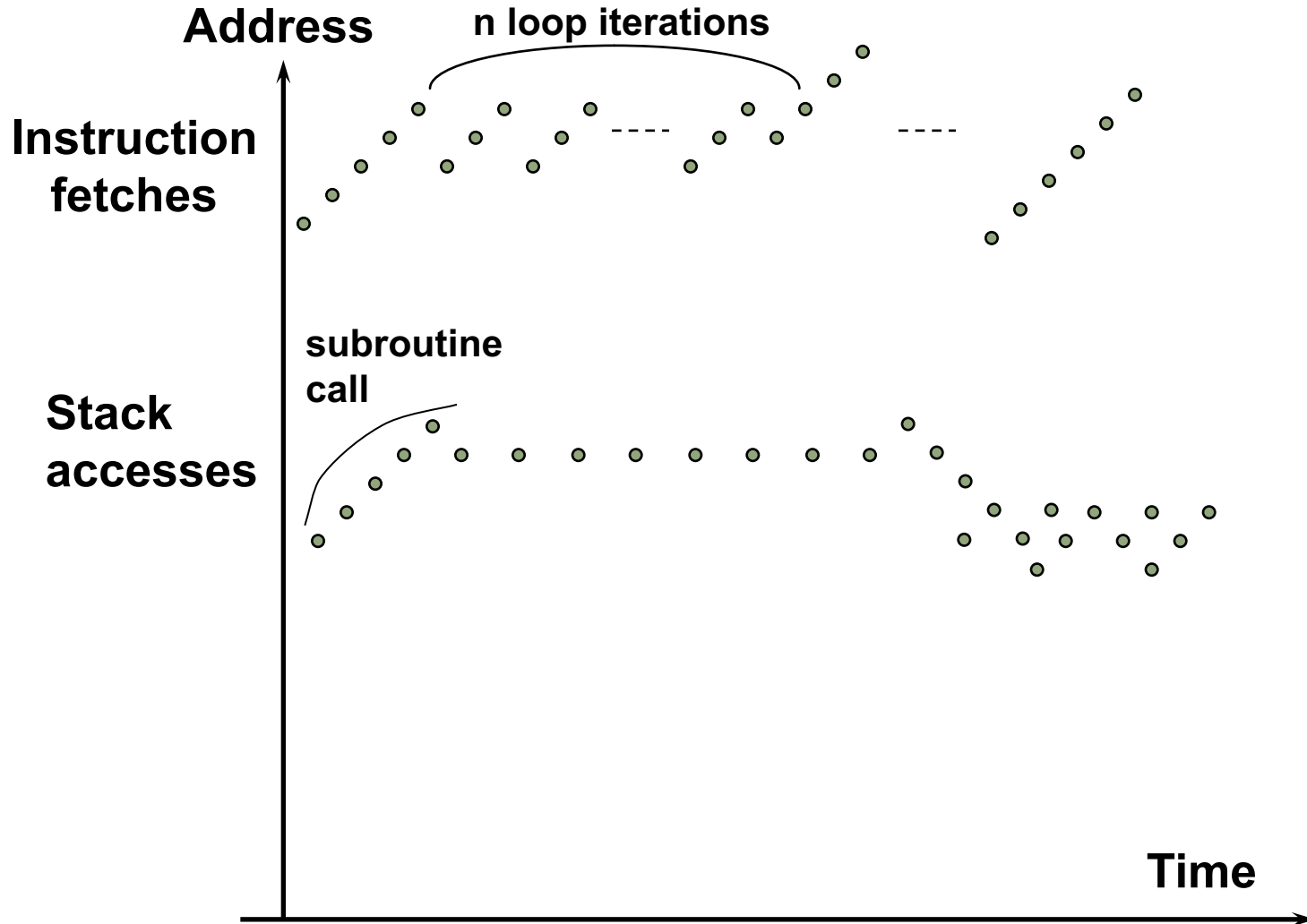
# Typical Memory Reference Patterns



Address

n loop iterations

Instruction
fetches

Time

# Typical Memory Reference Patterns



Address

n loop iterations

Instruction fetches

Stack accesses

Time

# Typical Memory Reference Patterns



**Address**

**n loop iterations**

**Instruction fetches**

**Stack accesses**

**Time**

# Typical Memory Reference Patterns



**Address**

**n loop iterations**

**Instruction fetches**

**subroutine call**

**Stack accesses**

**Time**

# Typical Memory Reference Patterns

# Typical Memory Reference Patterns



**Address**

**n loop iterations**

**Instruction fetches**

**Stack accesses**

**subroutine call**

**subroutine return**

**argument access**

**Time**

# Typical Memory Reference Patterns



**Address**

n loop iterations

**Instruction fetches**

subroutine call

**Stack accesses**

argument access

subroutine return

**Data accesses**

**Time**

# 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

# Typical Memory Reference Patterns



**Address**

n loop iterations

**Instruction fetches**

**Stack accesses**

subroutine call

subroutine return

argument access

**Data accesses**

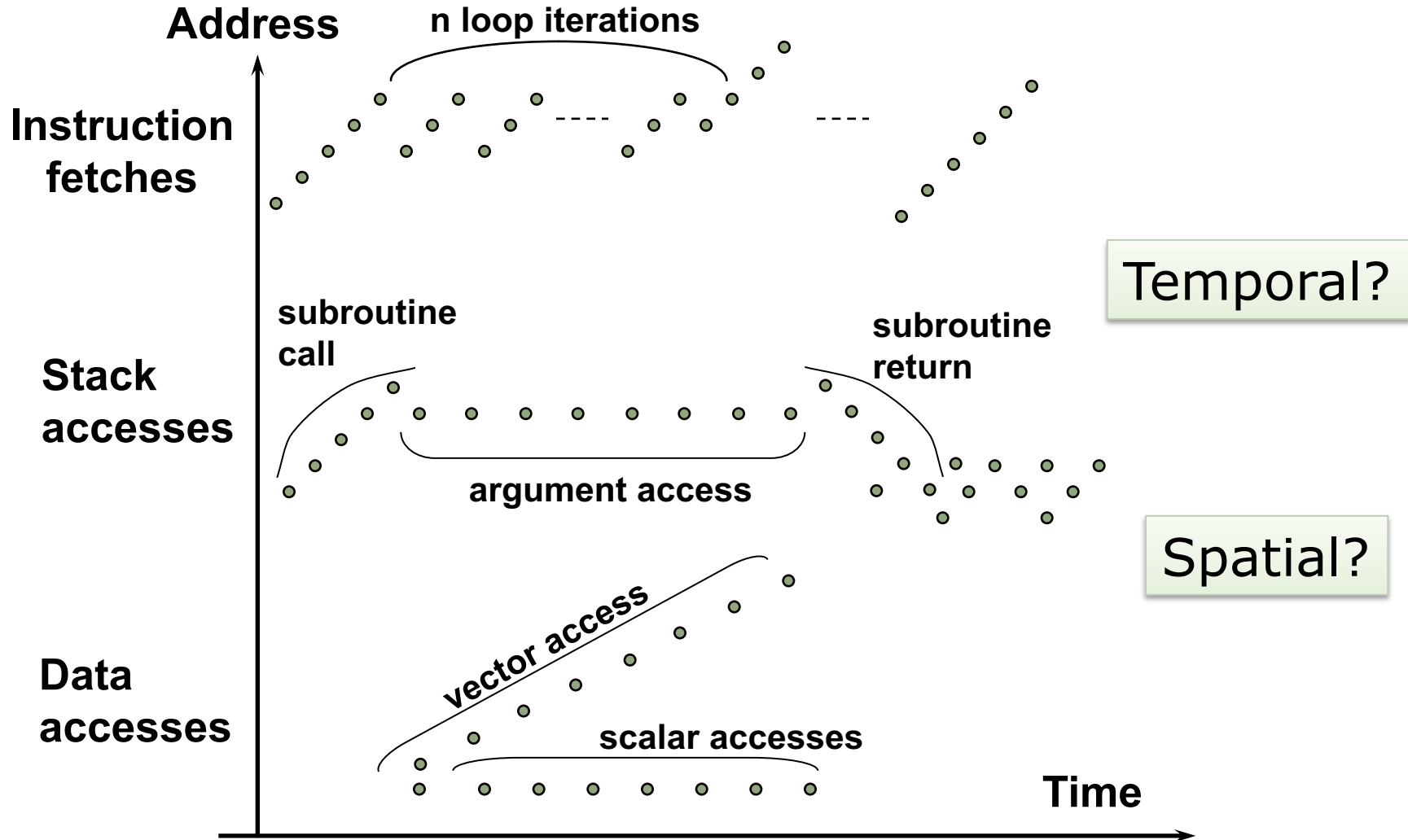vector access

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

# Temporal vs. Spatial Reuse?

# Data Orchestration Techniques

Two approaches to controlling data movement in the memory hierarchy:

- *Explicit:* Manually at the direction of the programmer using instructions

- *Implicit:* Automatically by the hardware in response to a request by an instruction, but transparent to the programmer.

# Data Orchestration Techniques

Two approaches to controlling data movement in the memory hierarchy:

- *Explicit:* Manually at the direction of the programmer using instructions

  Registers

- *Implicit:* Automatically by the hardware in response to a request by an instruction, but transparent to the programmer.

# Data Orchestration Techniques

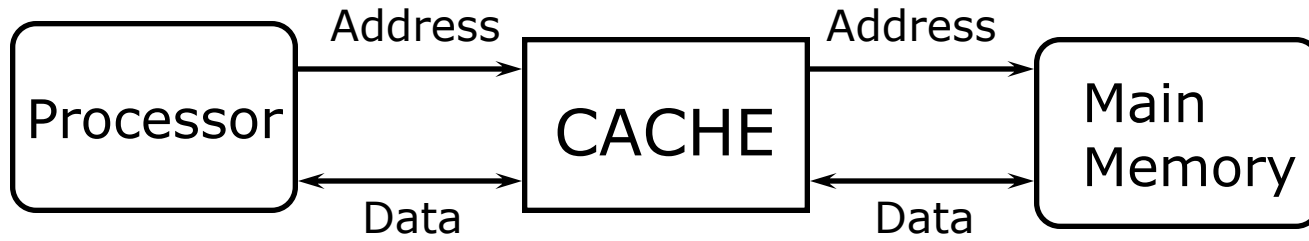Two approaches to controlling data movement in the memory hierarchy:

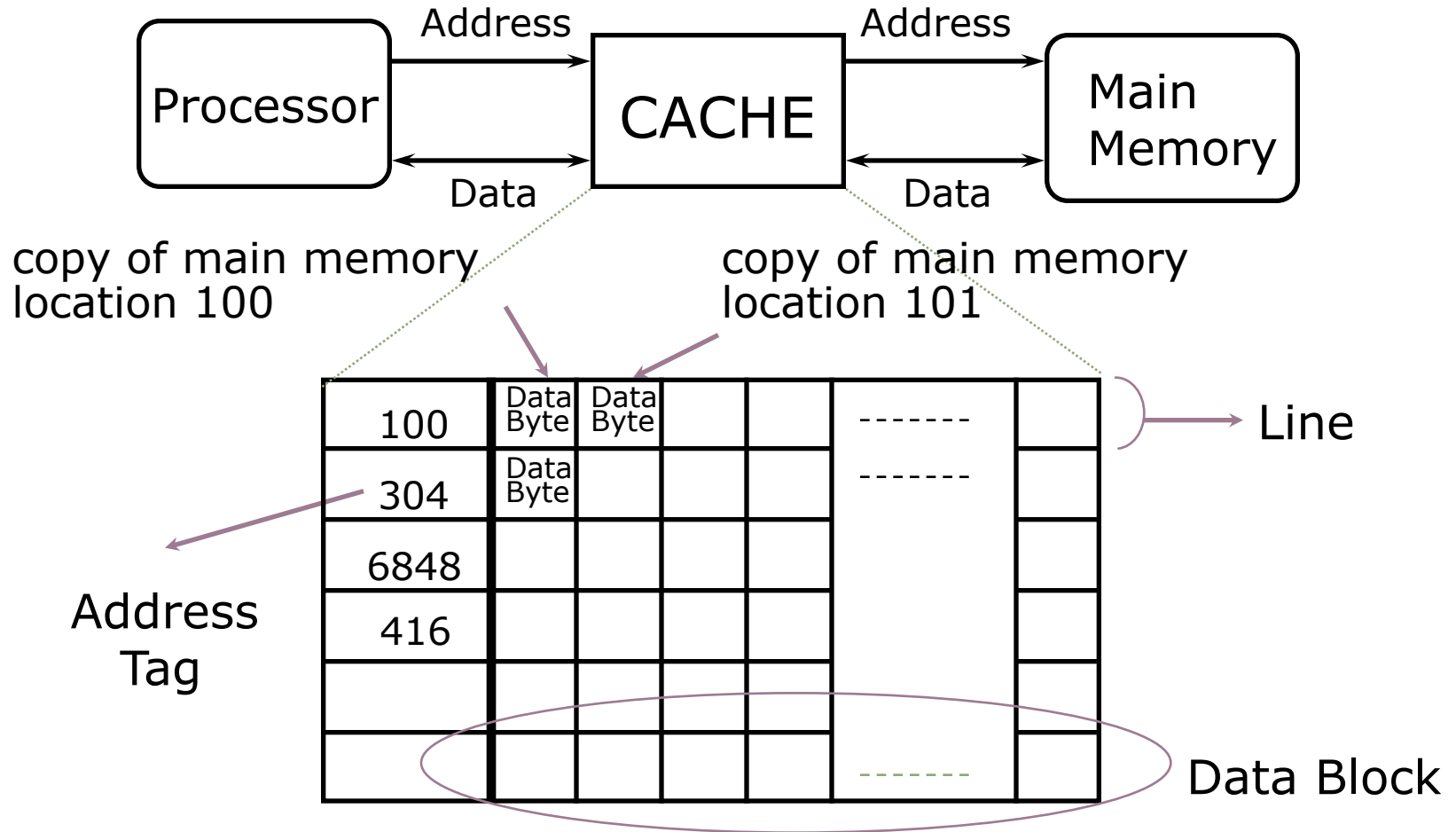– *Explicit:* Manually at the direction of the programmer using instructions

Registers

– *Implicit:* Automatically by the hardware in response to a request by an instruction, but transparent to the programmer.
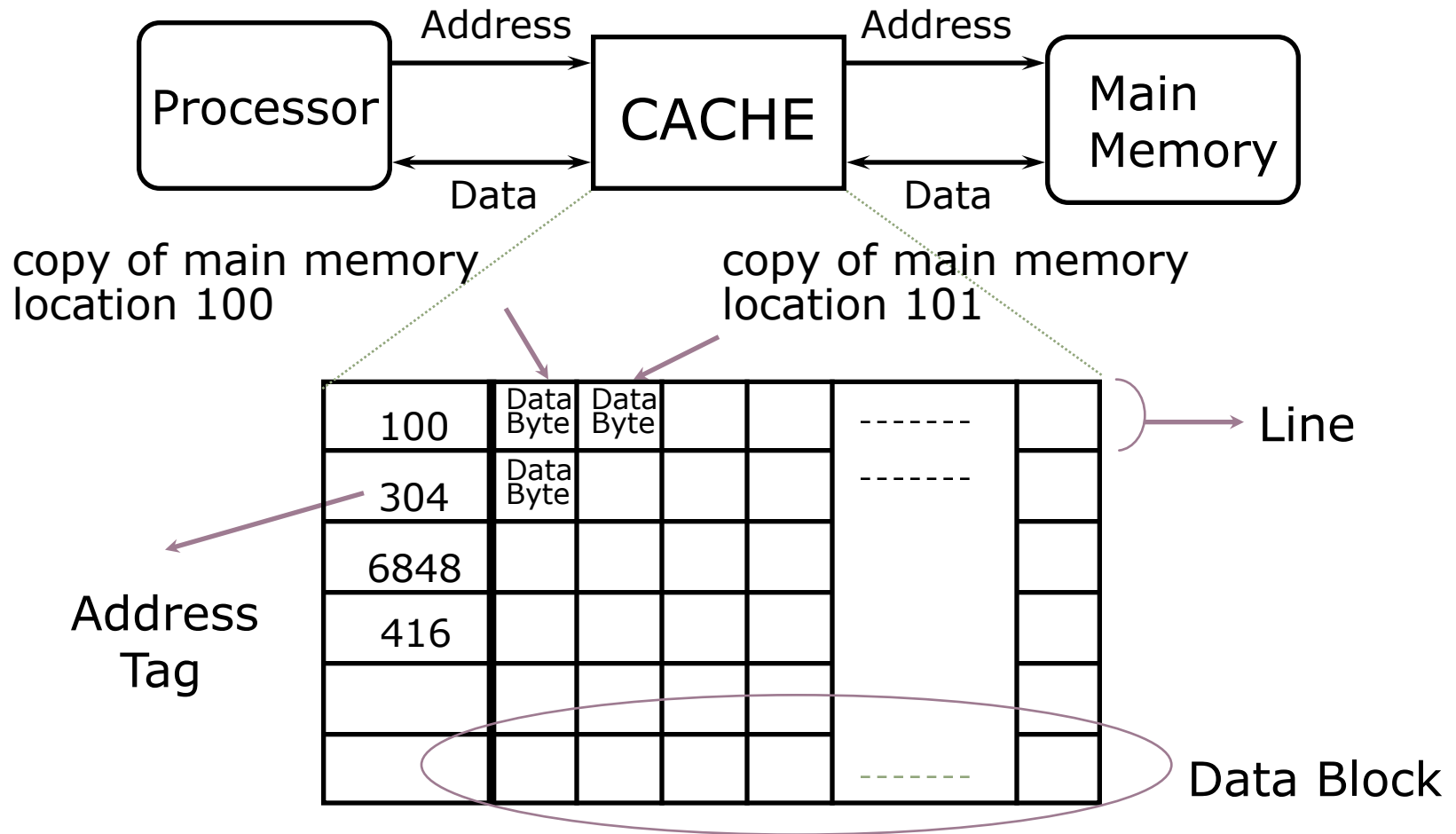
Cache hierarchy

# Inside a Cache
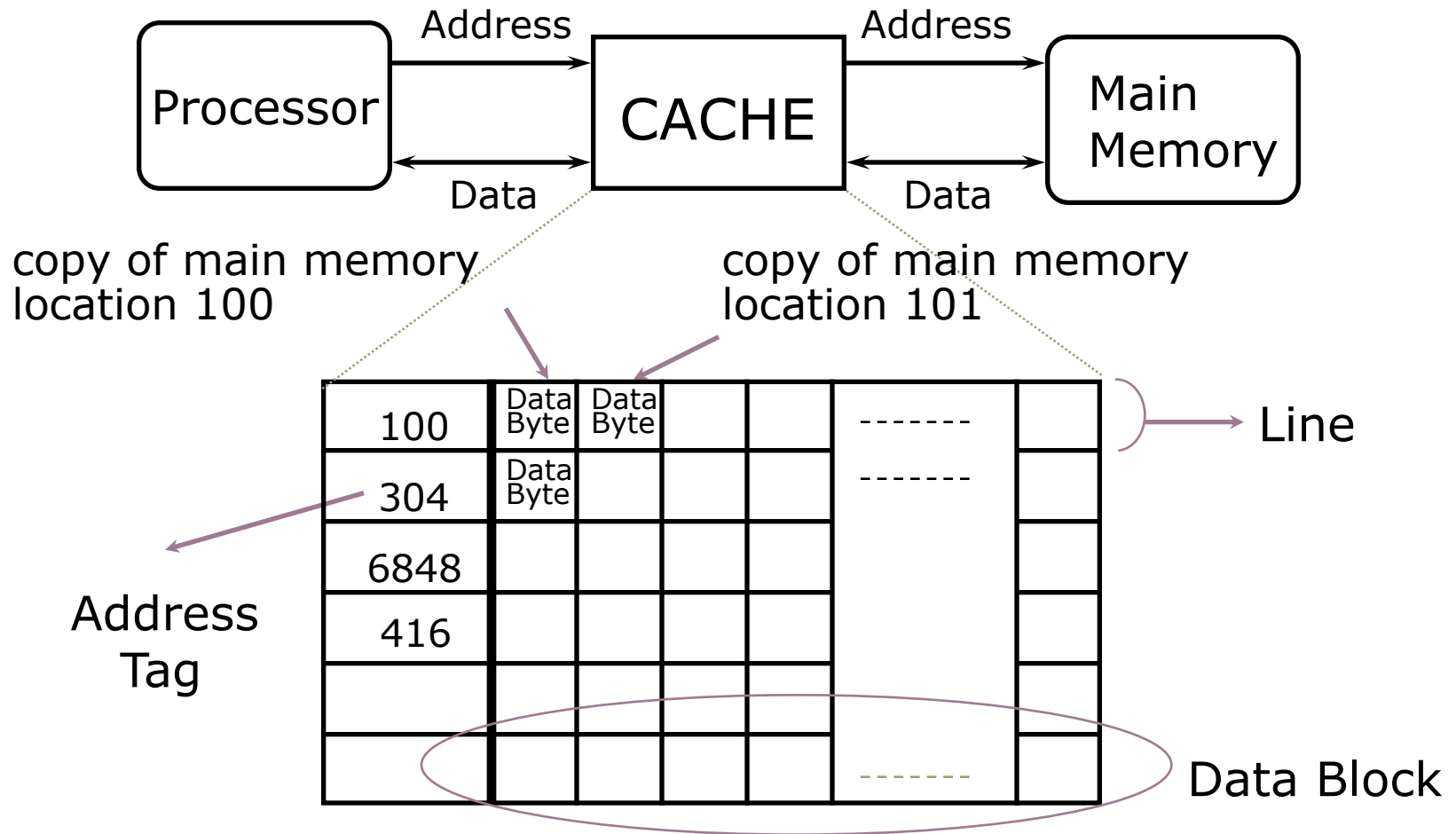
# Inside a Cache

# Inside a Cache



*Q: How many bits needed in tag?* _____

# Inside a Cache



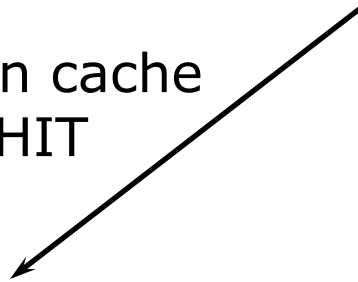Q: *How many bits needed in tag?* <u>Enough to uniquely identify block</u>

# 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

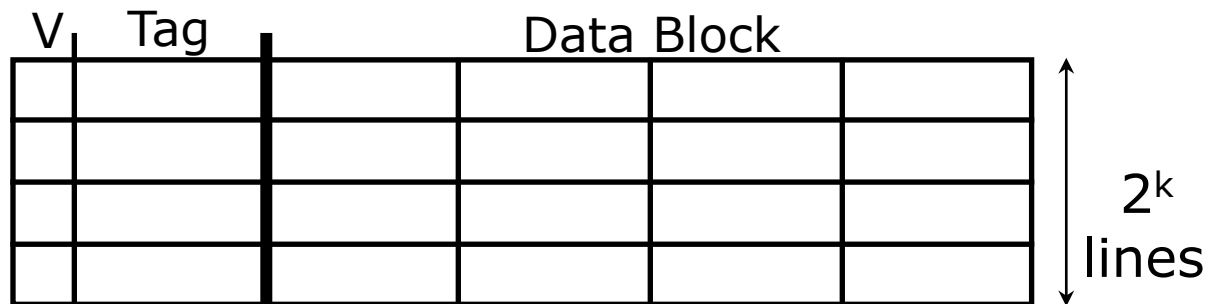Not in cache
a.k.a. MISS

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
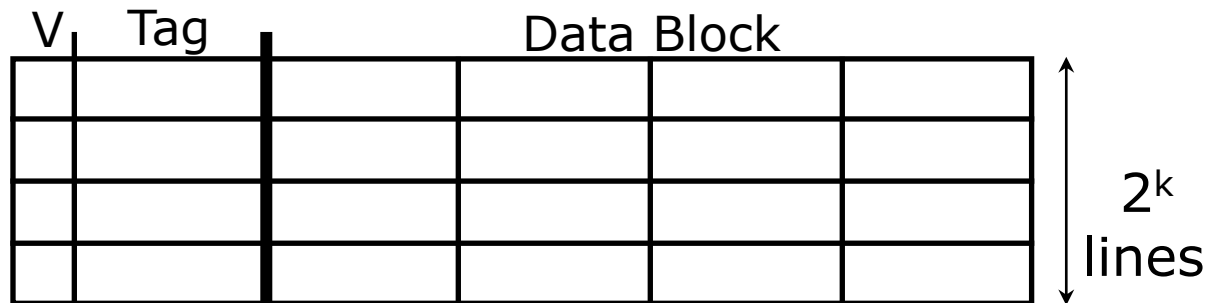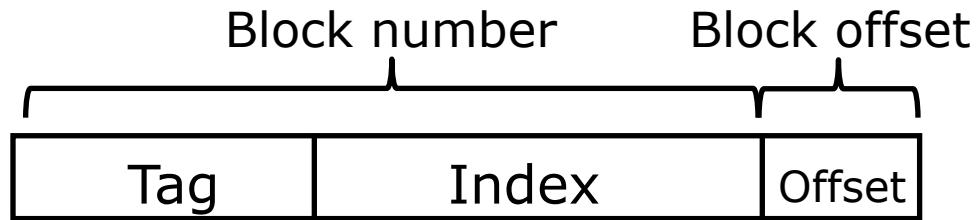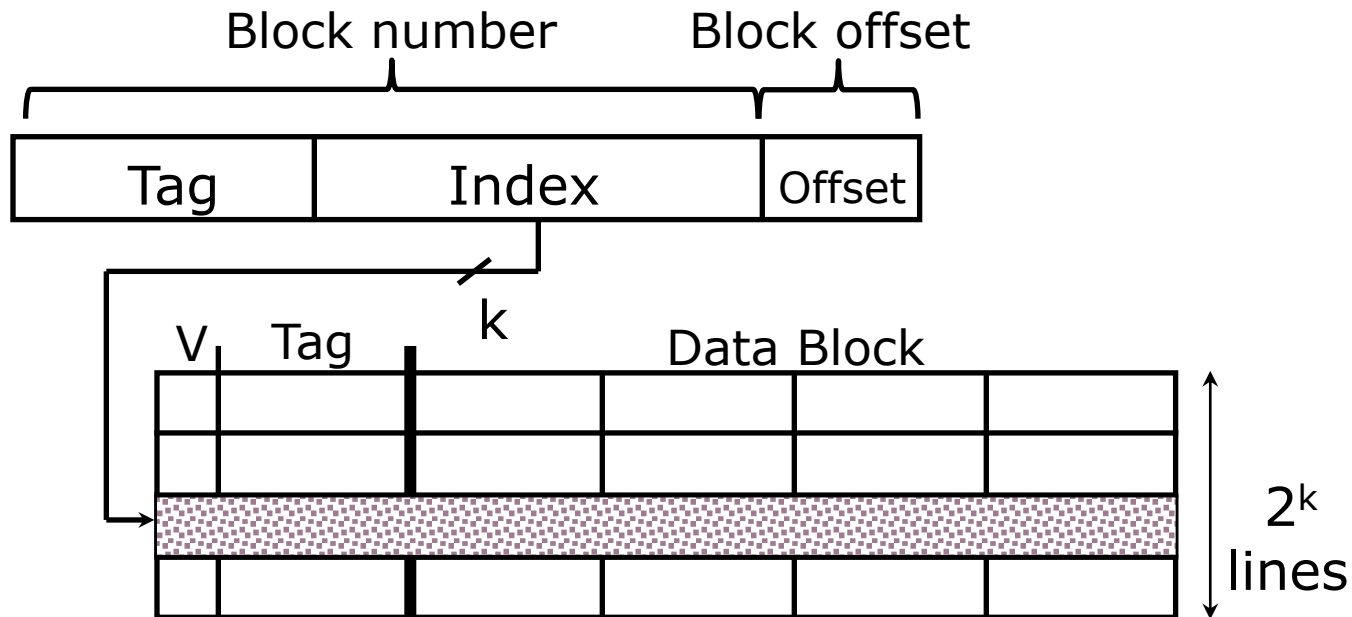
| V | Tag | | Data Block | | | |
|---|---|---|---|---|---|---|
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |

$2^k$ lines

# Direct-Mapped Cache

Block number      Block offset

| Tag | Index | Offset |
|---|---|---|

V   Tag         Data Block

| | | | | | |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

$2^k$ lines

# Direct-Mapped Cache

# Direct-Mapped Cache

Block number          Block offset

| Tag | Index | Offset |
|-----|-------|--------|

t

k

V   Tag          Data Block

$2^k$ lines

t

HIT

# 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
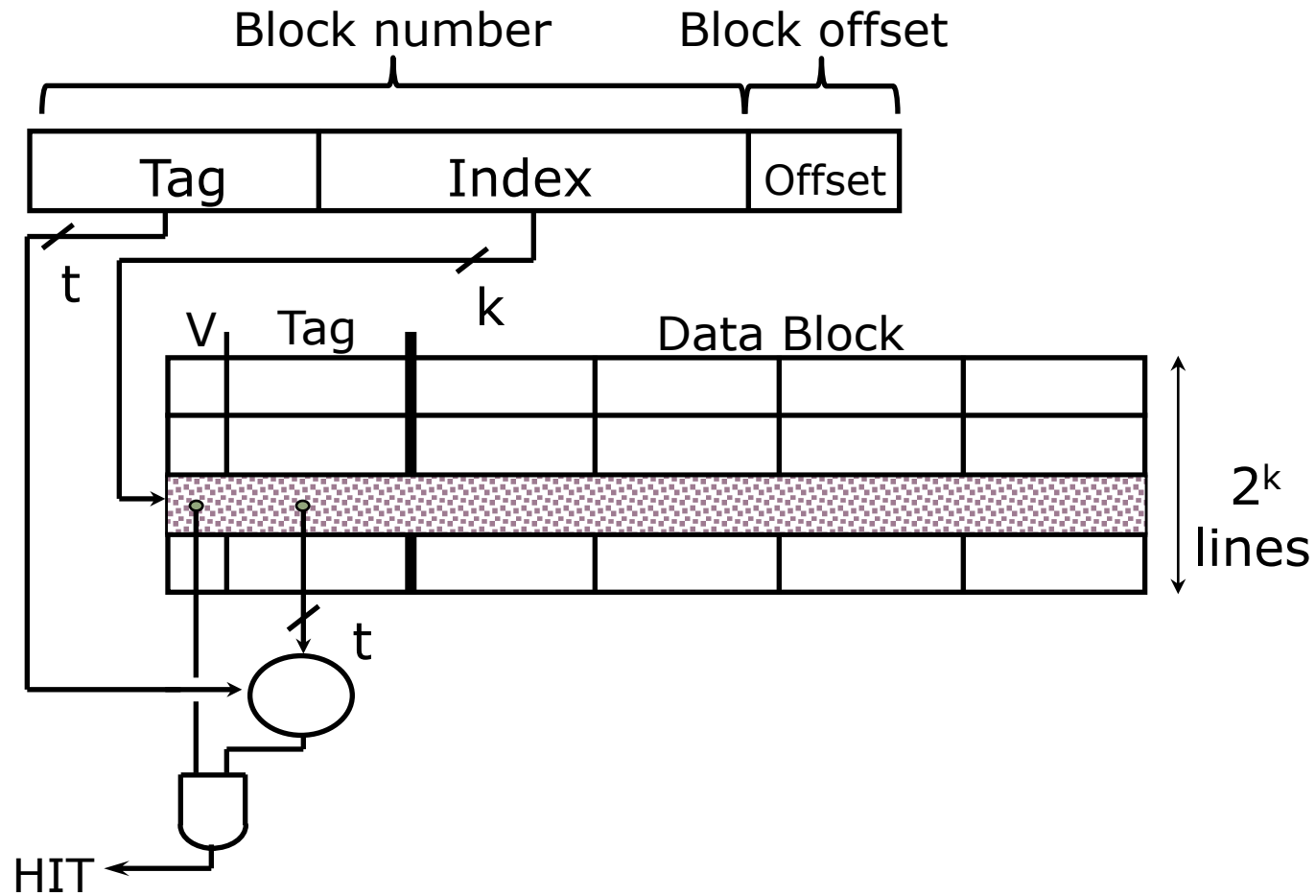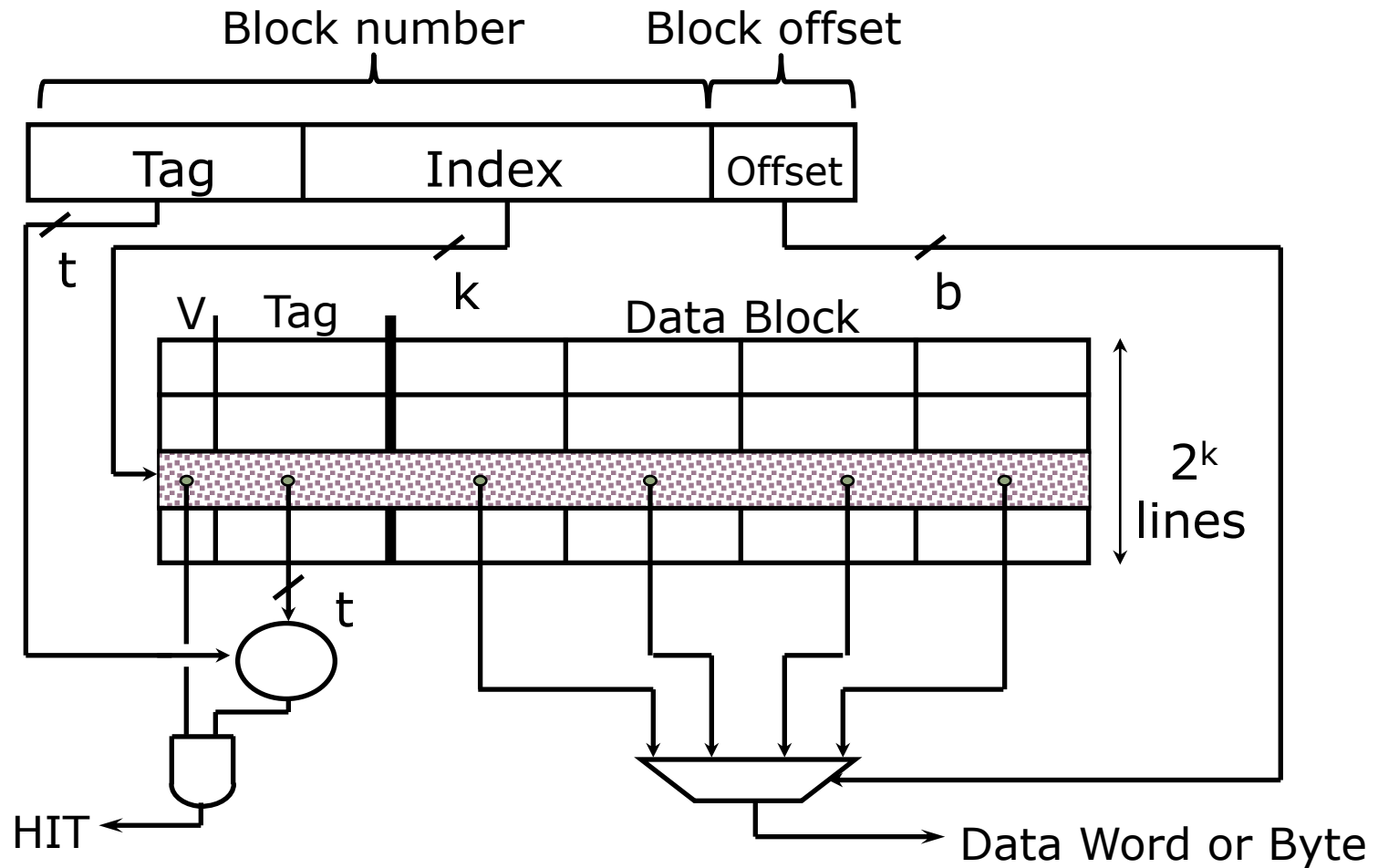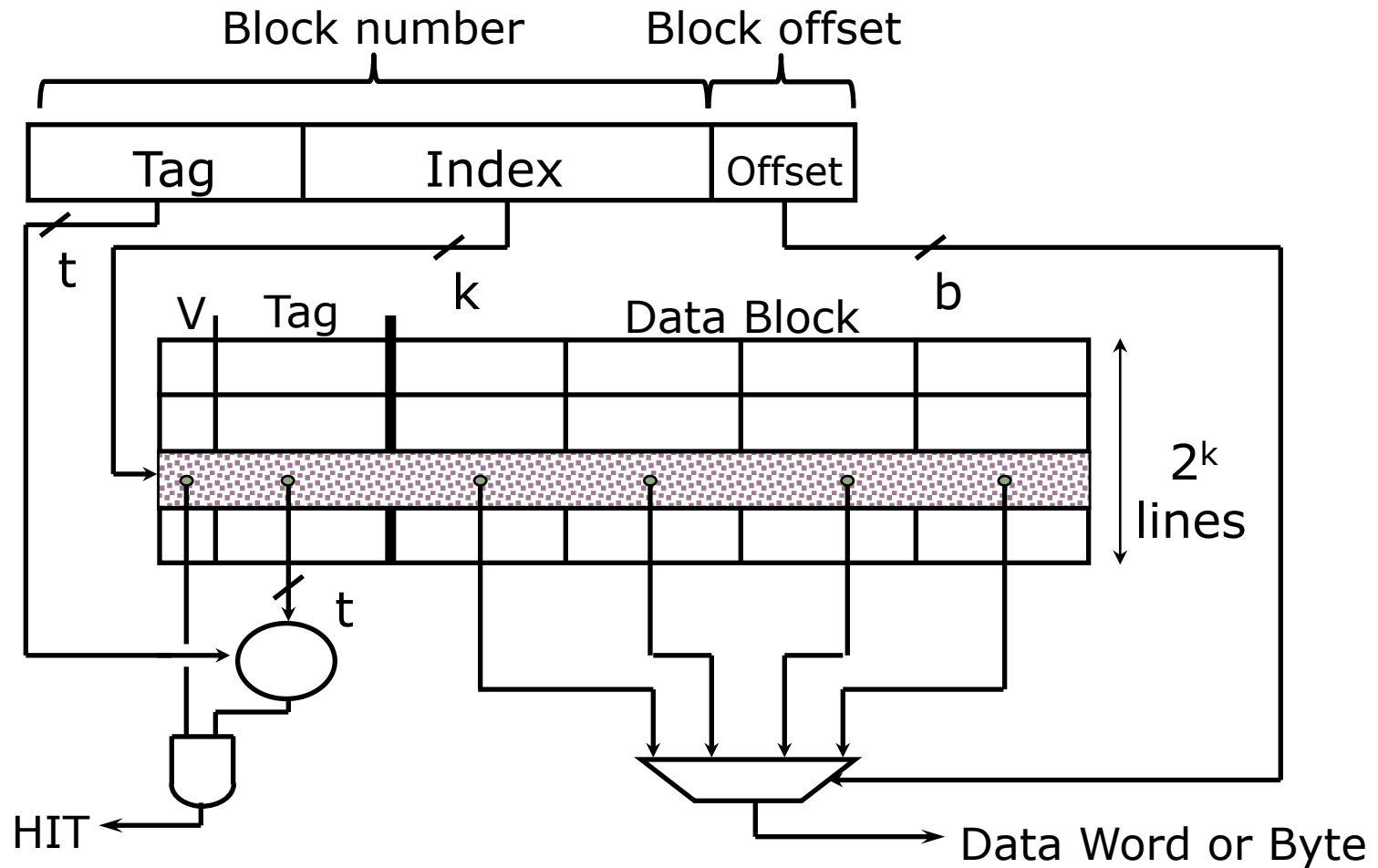
*Q: What is a bad reference pattern?*

# 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
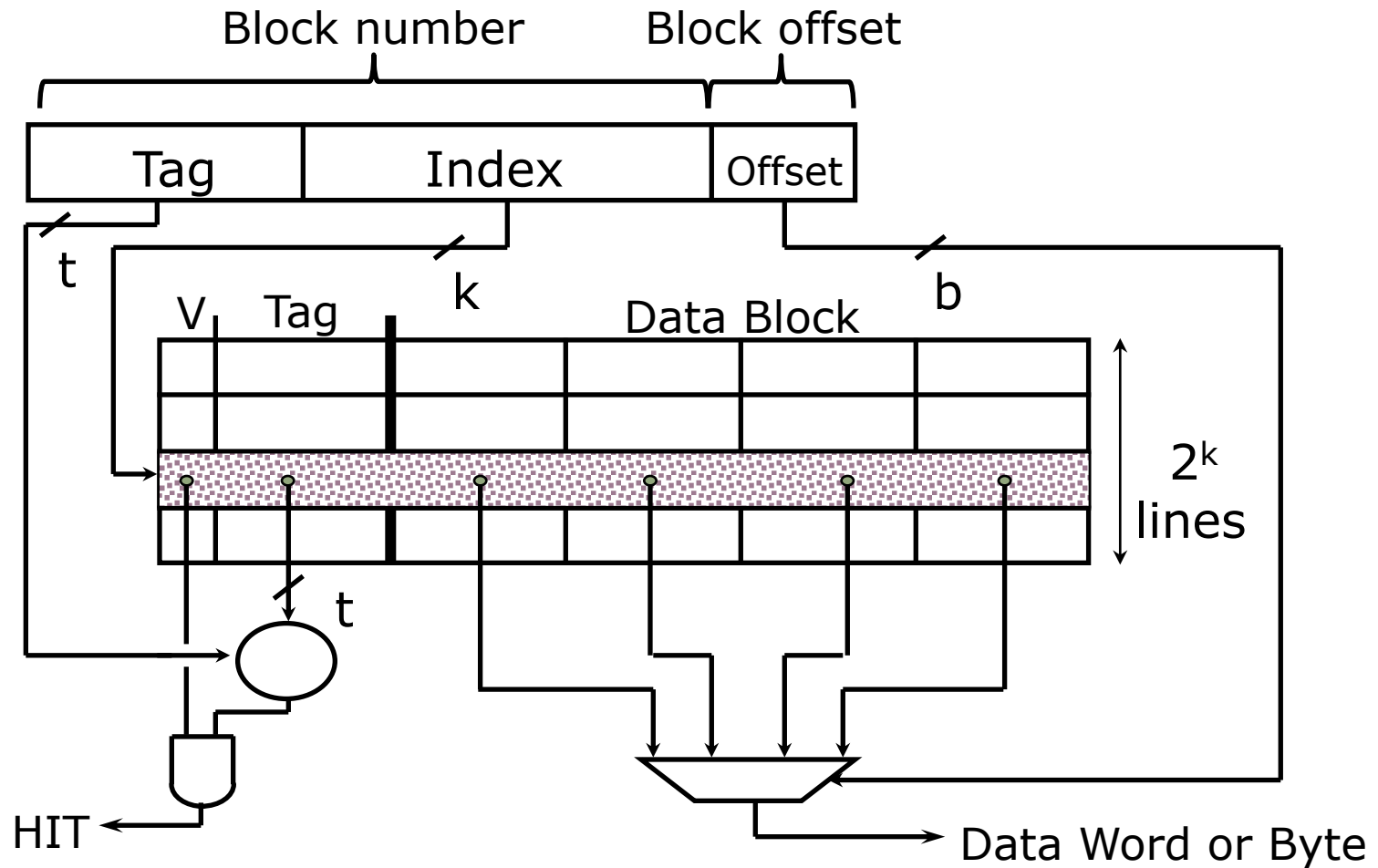
*Q: What is a bad reference pattern?*   *Strided at size of cache*

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



| Index | Tag | Offset |
|-------|-----|--------|

$k$

$t$

$b$

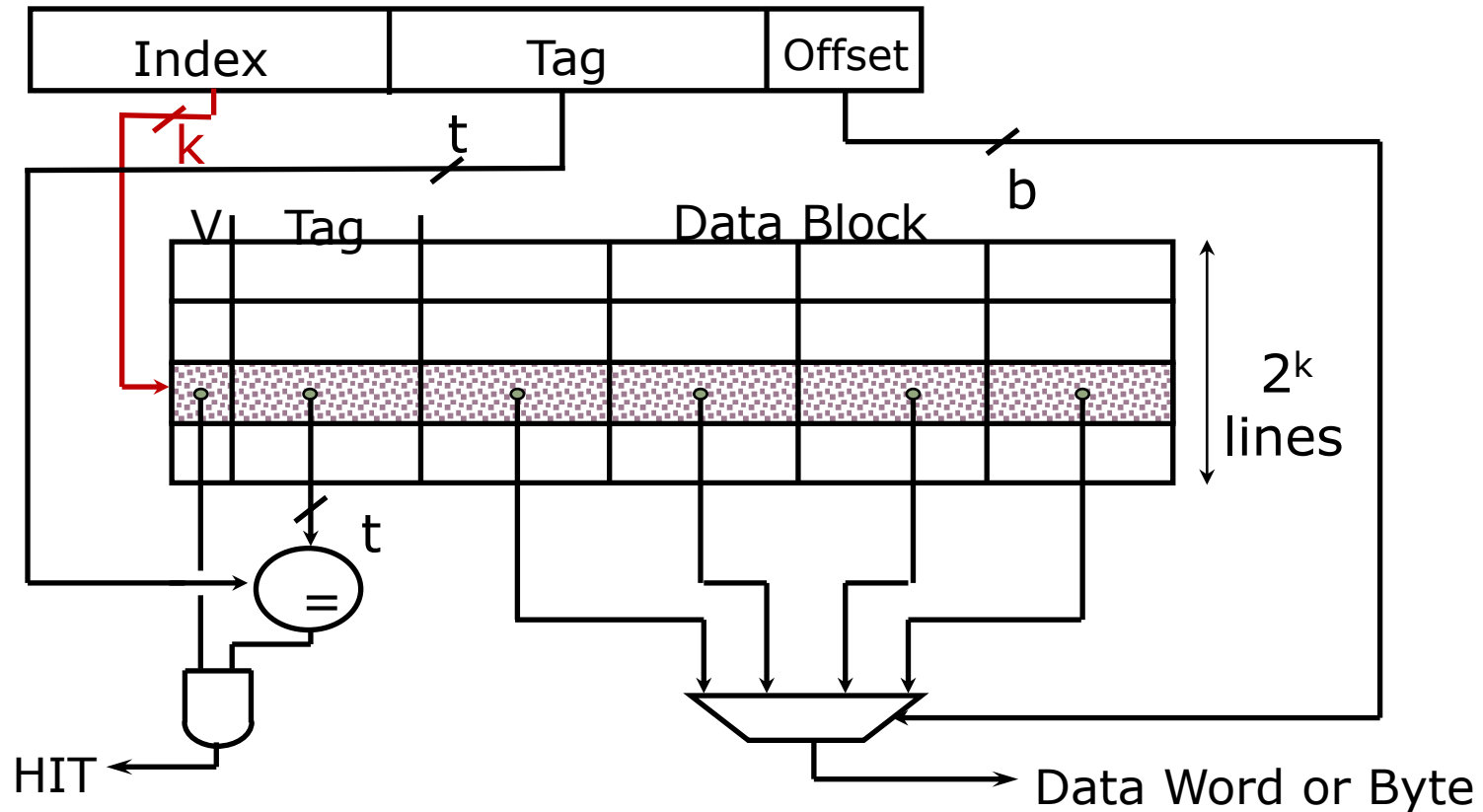V   Tag                    Data Block

$2^k$ lines

$t$

=

HIT

Data Word or Byte

*Q: Why might this be undesirable?* _____

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



| Index | Tag | Offset |
|---|---|---|

$k$

$t$

$b$

V | Tag | Data Block

$2^k$ lines

$t$

=

HIT

Data Word or Byte
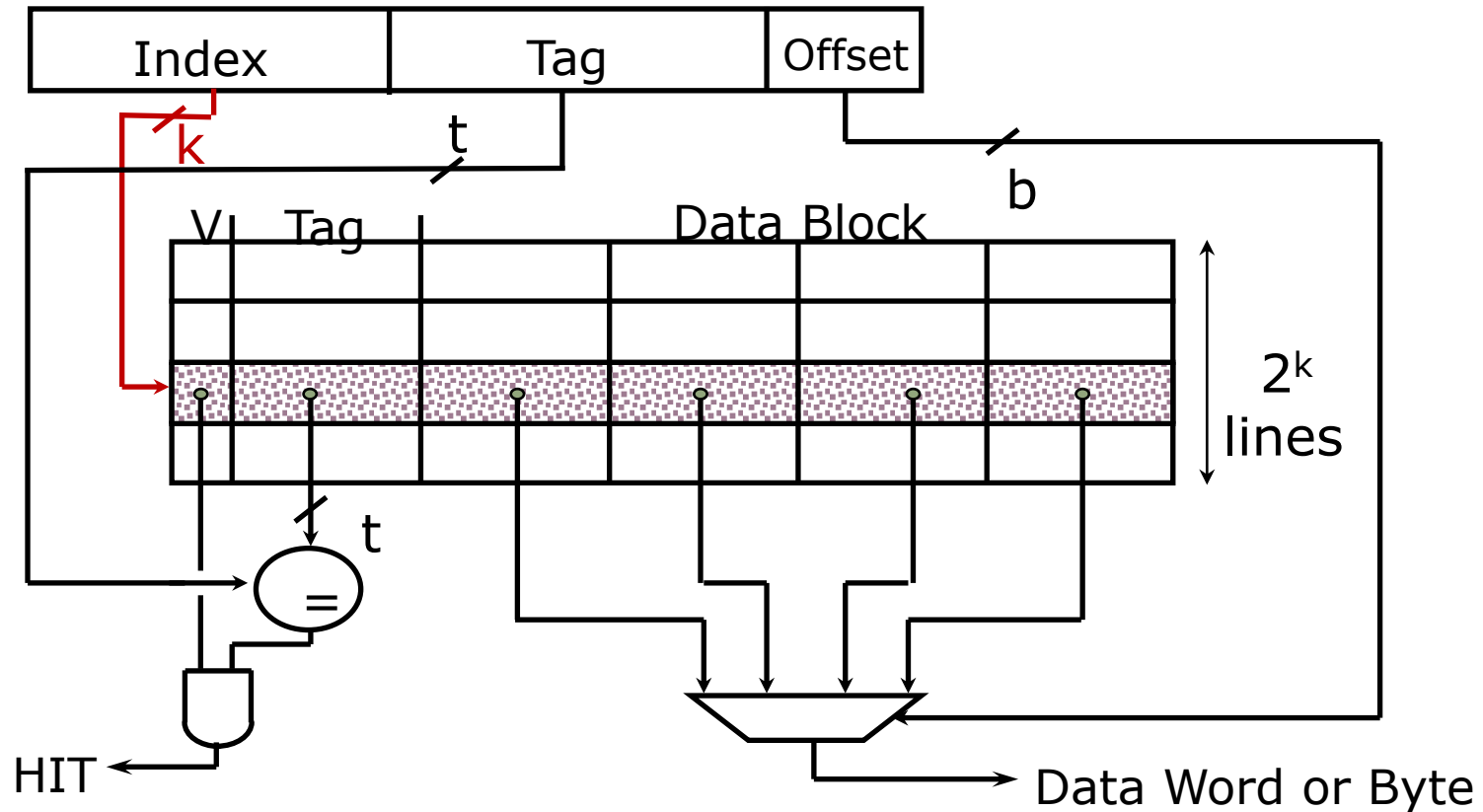
*Q: Why might this be undesirable?* <span style="color:red">Spatially local blocks conflict</span>

# Hashed Address Mapping
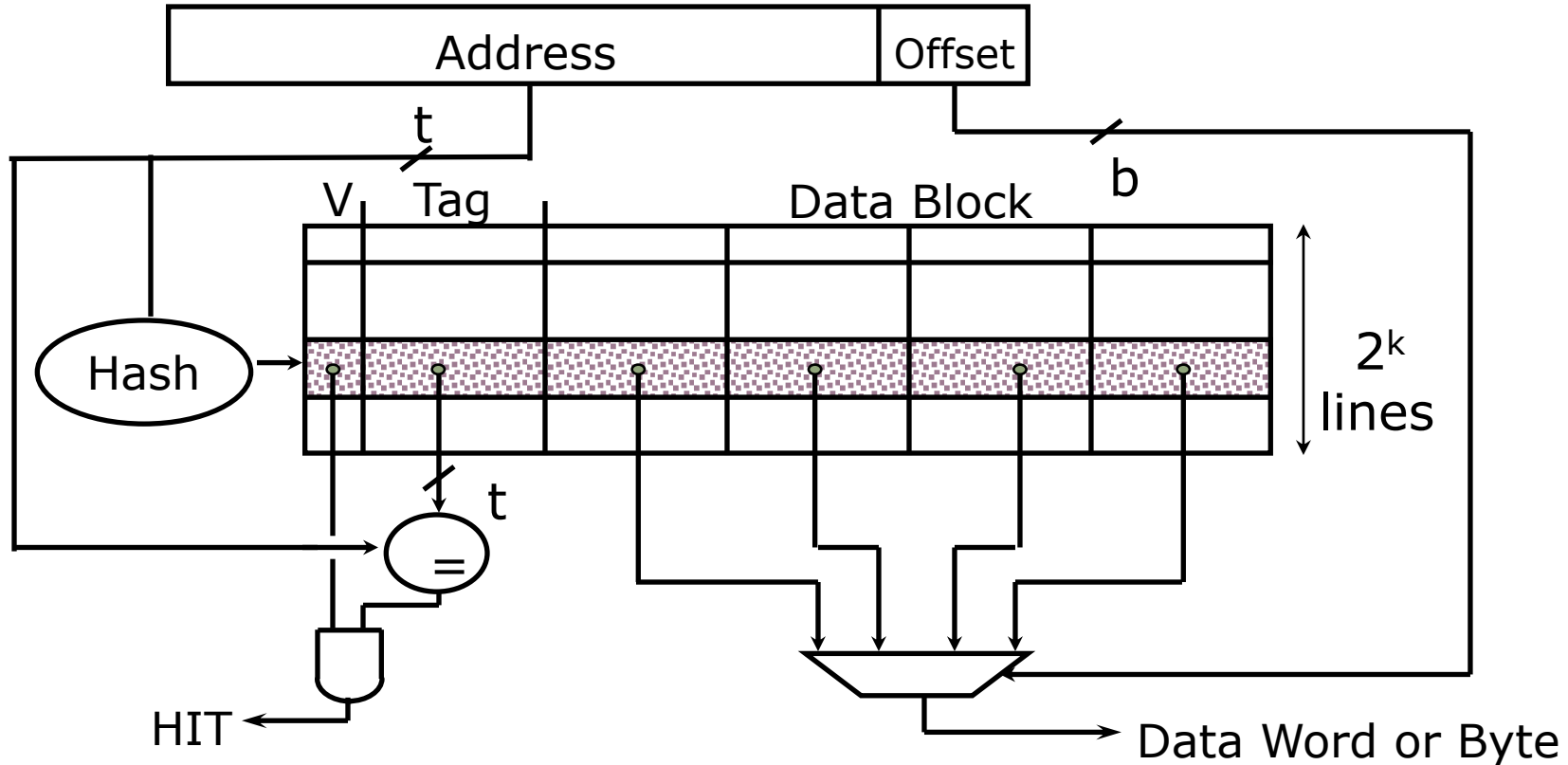


*Q: What are the tradeoffs of hashing?*

# Hashed Address Mapping
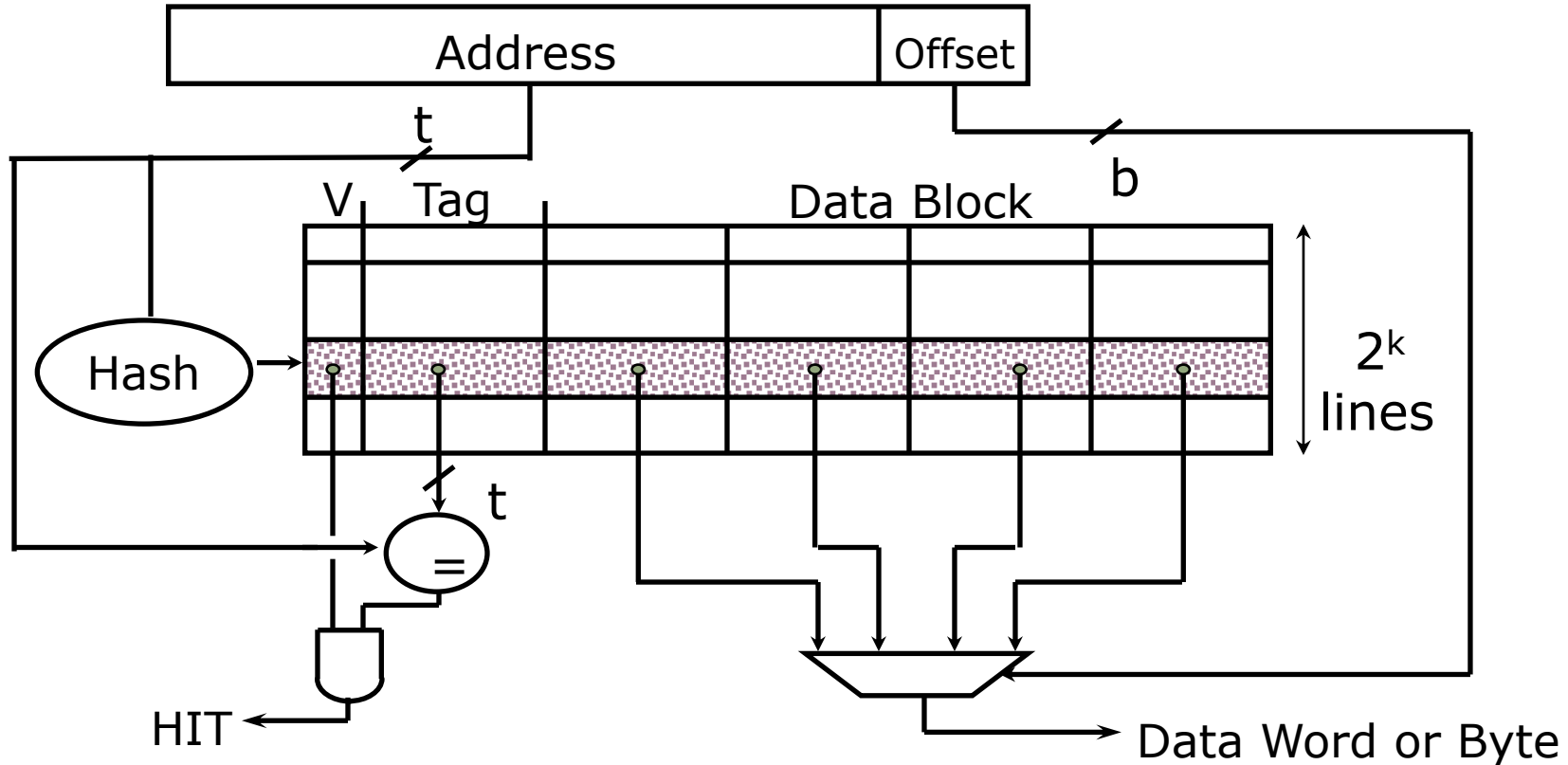


*Q: What are the tradeoffs of hashing?*

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

# 2-Way Set-Associative Cache

| Tag | Index | Block Offset |
|-----|-------|--------------|
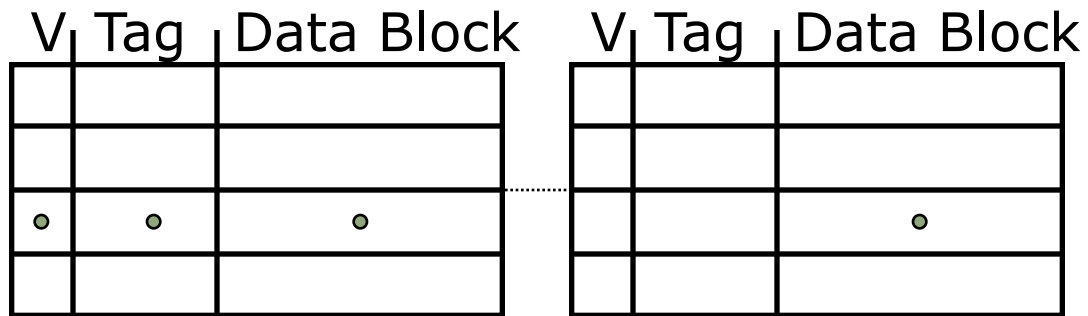
# 2-Way Set-Associative Cache
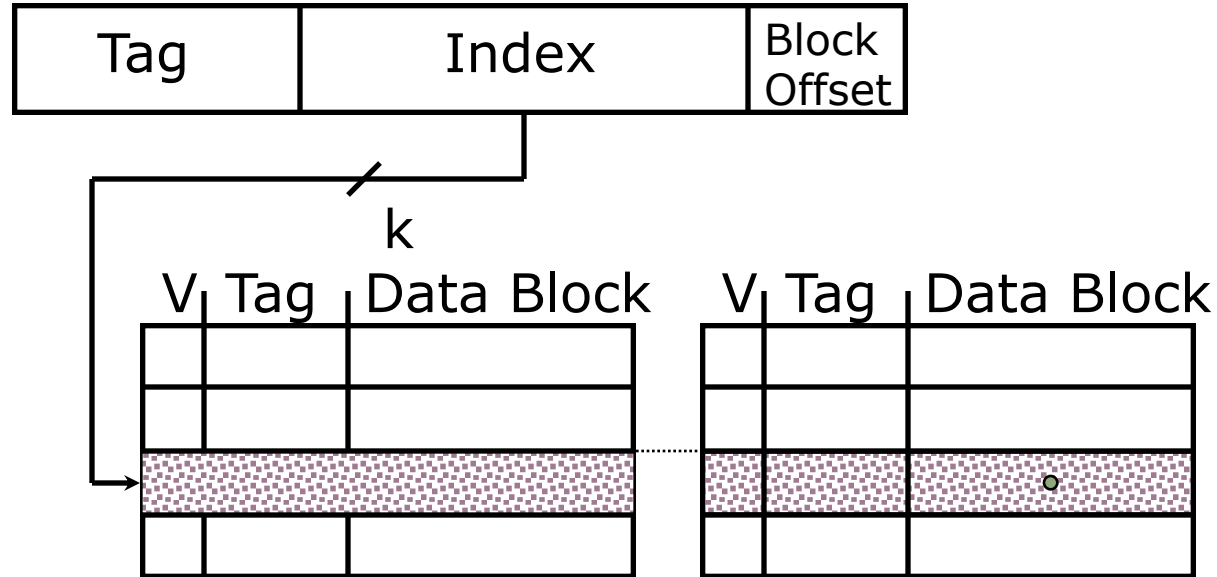
# 2-Way Set-Associative Cache
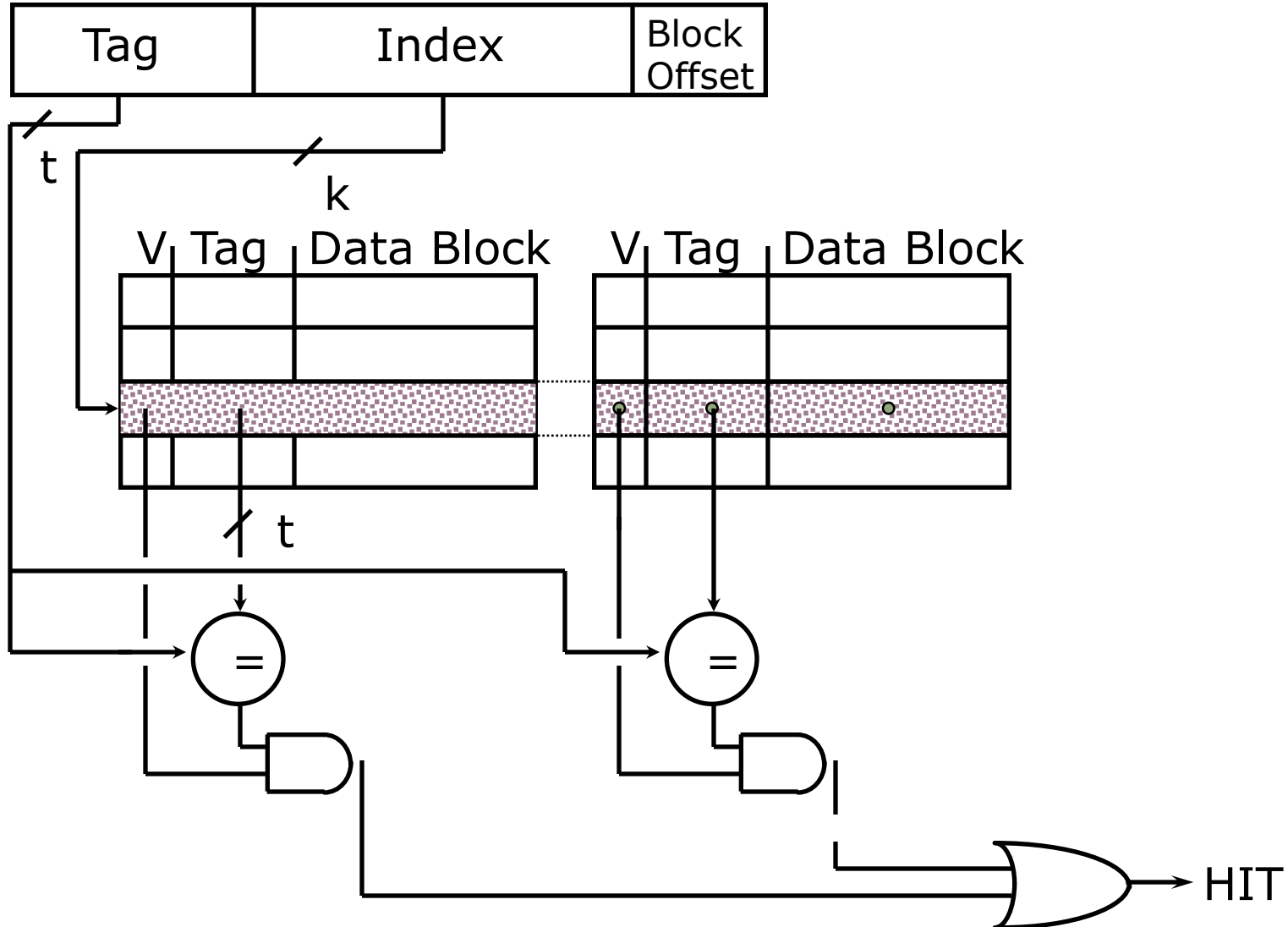
# 2-Way Set-Associative Cache

# Fully Associative Cache



V  Tag    Data Block

t

Tag

t

Block Offset

b

HIT

Data Word
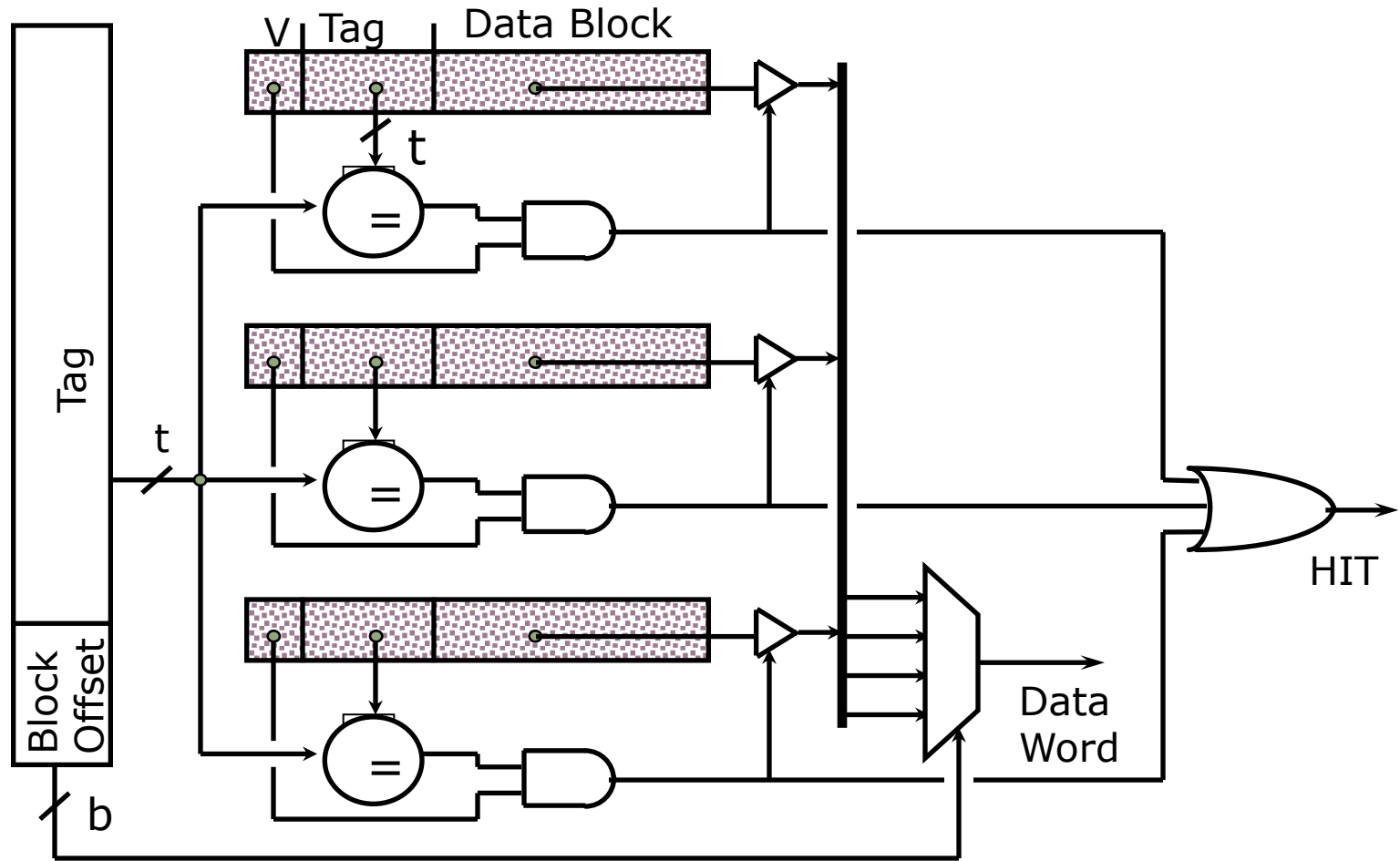
*Q: Where are the index bits?* _____

# Fully Associative Cache
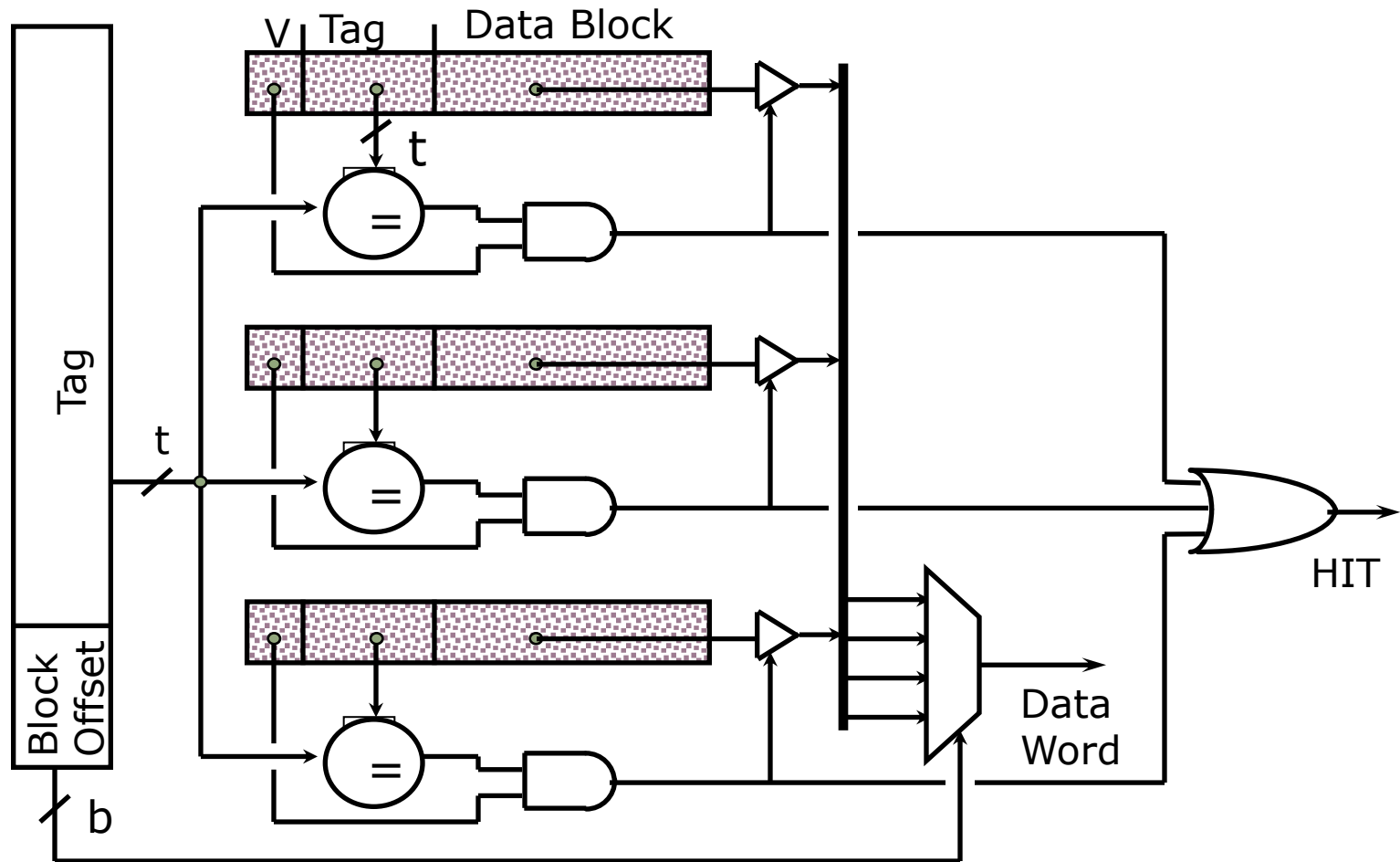


*Q: Where are the index bits?* _____ Not needed _____

# 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)
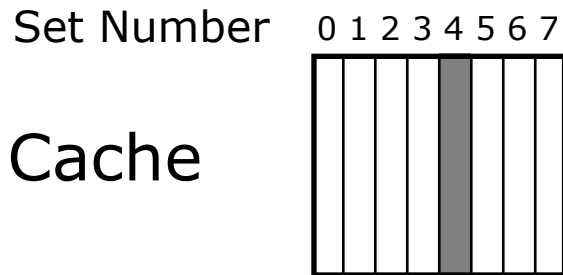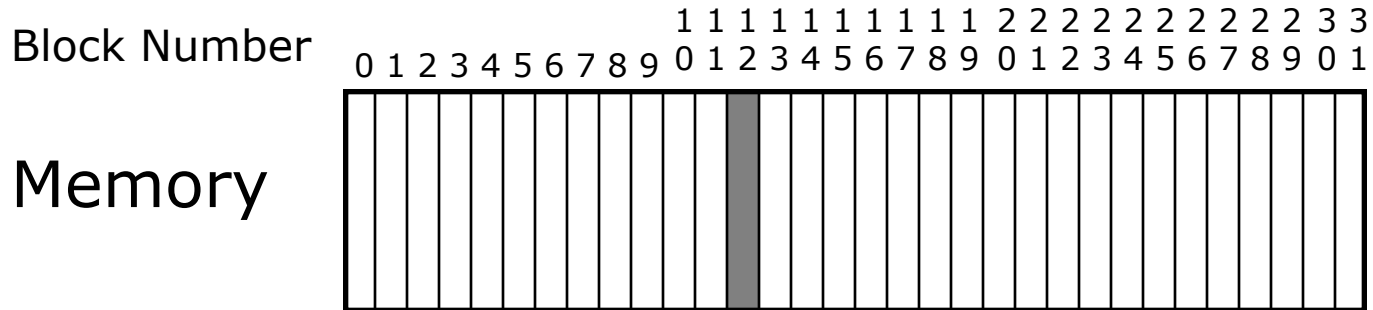
(2-way) Set
Associative
anywhere in
set 0
(12 mod 4)

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
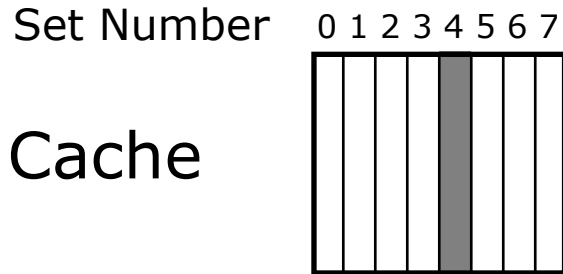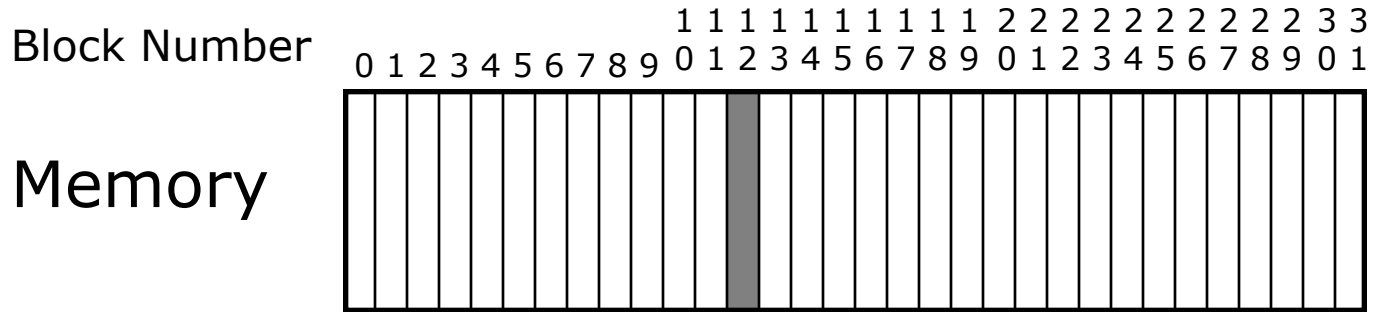
## Cache

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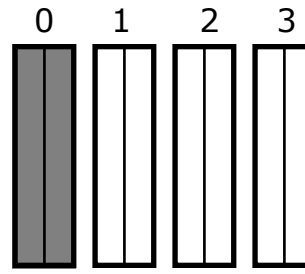
(2-way) Set Associative
anywhere in set 0
*(12 mod 4)*
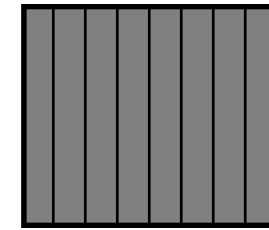
Fully Associative
anywhere

block 12 can be placed

# Cache Performance

Average memory access time (AMAT) =
Hit time + Miss rate x Miss penalty

# Cache Performance

Average memory access time (AMAT) =
               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?*

# Cache Performance

Average memory access time (AMAT) =
        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 fully-associative placement & perfect 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

# Backup Slides

# Processor Performance

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} * \frac{\text{Cycles}}{\text{Instruction}} * \frac{\text{Time}}{\text{Cycle}}$$

- Instructions per program depends on source code, compiler technology and ISA

- Cycles per instructions (CPI) depends upon the ISA and the microarchitecture

- Time per cycle depends upon the microarchitecture and the base technology

# Processor Performance

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} * \frac{\text{Cycles}}{\text{Instruction}} * \frac{\text{Time}}{\text{Cycle}}$$

– Instructions per program depends on source code, compiler technology and ISA

– Cycles per instructions (CPI) depends upon the ISA and the microarchitecture

– Time per cycle depends upon the microarchitecture and the base technology

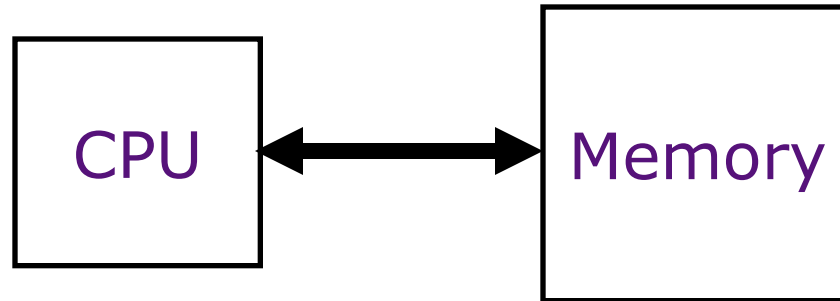| Microarchitecture | CPI | cycle time |
|---|---|---|
| Microcoded | >1 | short |
| Single-cycle unpipelined | 1 | long |
| Pipelined | 1 | short |

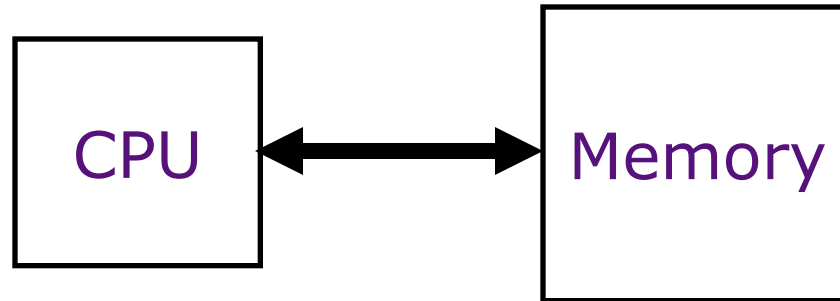# CPU-Memory Metrics
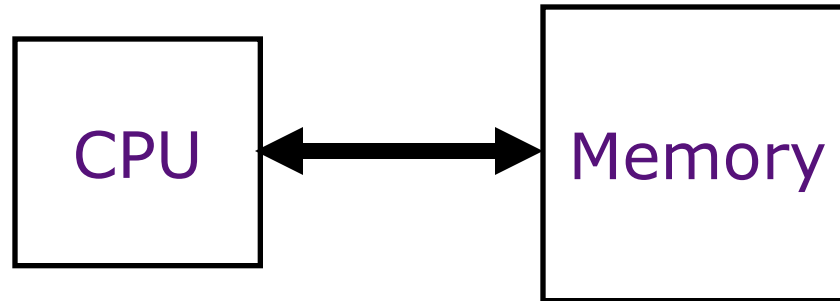
# CPU-Memory Metrics
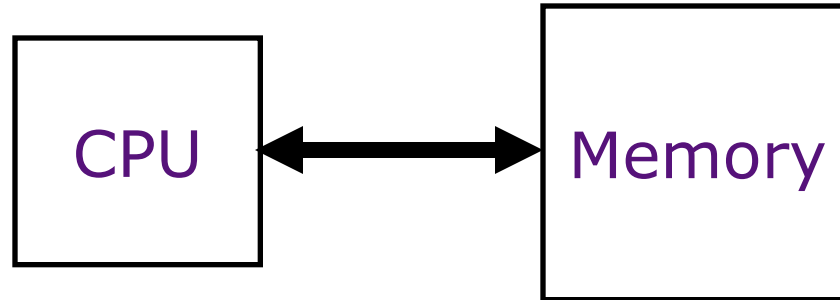


- Latency (time for a single access)
  Memory access time >> Processor cycle time

# CPU-Memory Metrics



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

- Bandwidth (number of accesses per unit time)
  - If memory latency is t cycles, and we have k banks,
  - The memory bandwidth is k/t requests per cycle

# CPU-Memory Metrics



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

- Bandwidth (number of accesses per unit time)
  If memory latency is t cycles, and we have k banks,
  The memory bandwidth is k/t requests per cycle

- Energy (nJ per access)

# Management of Memory Hierarchy

- ## Small/fast storage, e.g., registers
  - Address usually specified directly in instruction
  - Generally implemented using **explicit** data orchestration
    - e.g., 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 using **implicit** data orchestration
    - e.g., as a cache hierarchy where hardware decides what is kept in fast memory
    - but software may provide "hints", e.g., don't cache or prefetch

*Thank you!*

*Next lecture:*
*Caches + Virtual memory*