Instruction Set Architecture

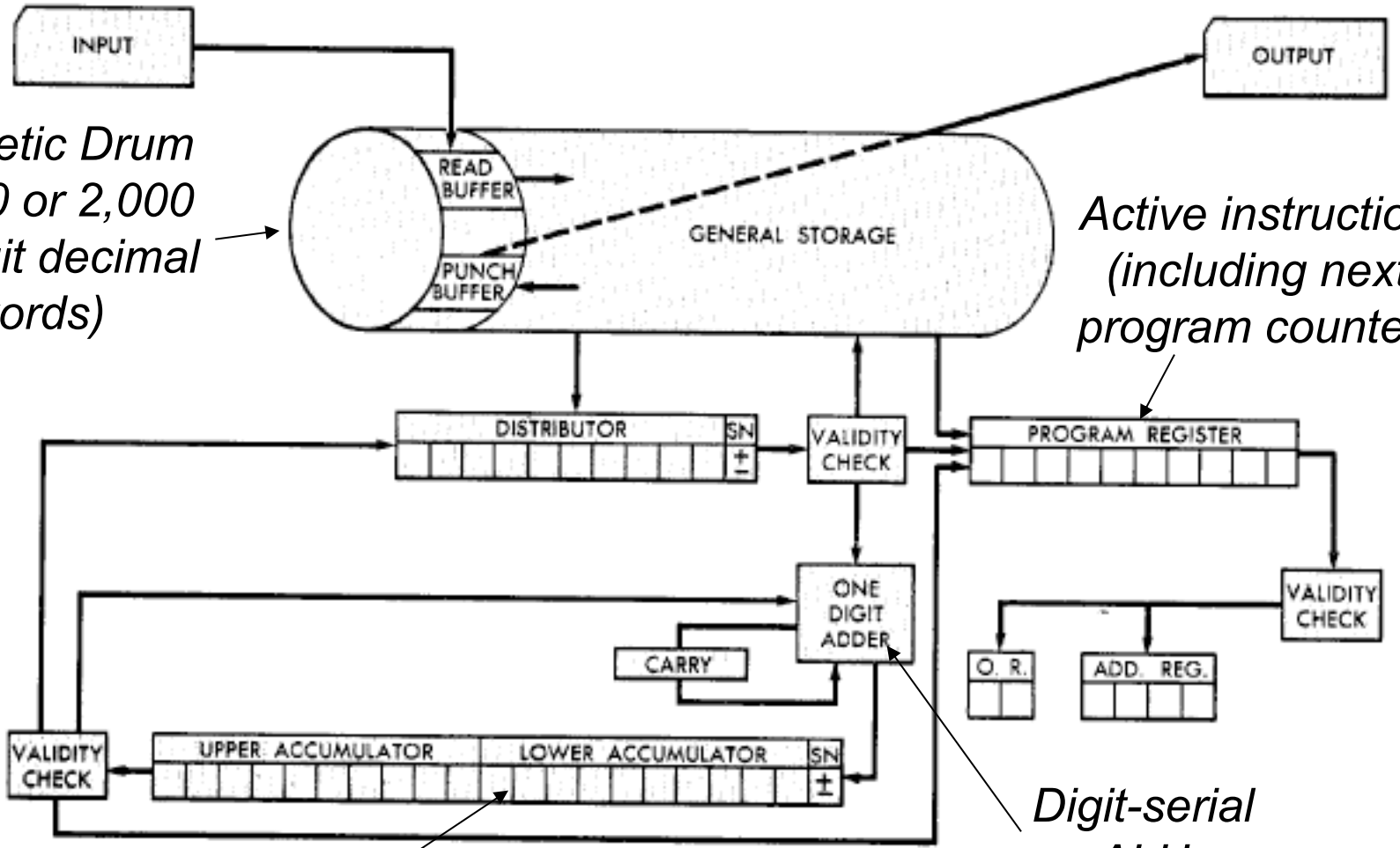
Mengjia Yan
Computer Science & Artificial Intelligence Lab
M.I.T.

Quiz Date

- Quiz 1: Oct 14 (in tutorial)
- Quiz 2: Nov 16 (in class)
- Quiz 3: Dec 14 (in class)

- Lab release and due dates are on syllabus

The IBM 650 (1953-4)



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

Active instruction (including next program counter)

20-digit accumulator

Digit-serial ALU

[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	→	7094
650	→	7074
702	→	7080
1401	→	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, ...

⇒ *IBM 360*

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*

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

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

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!*

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

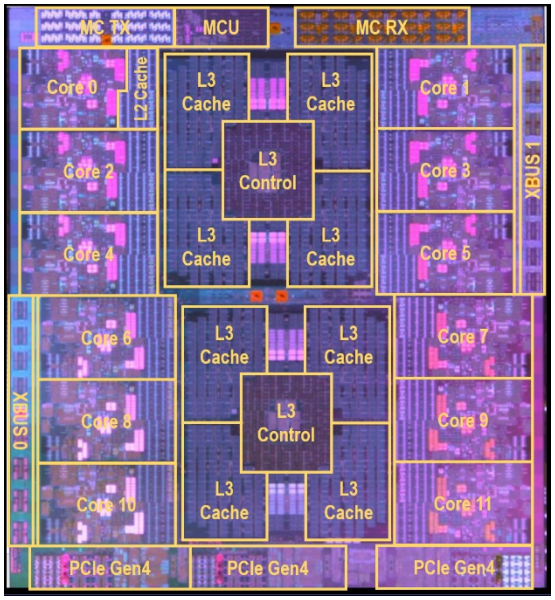
- Six implementations (Models, 30, 40, 50, 60, 62, 70)
- 50x performance difference across models
- ***ISA completely hid the underlying technological differences between various models***

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

- 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
  - Defines set of programmer visible state
  - Defines data types
  - Defines instruction semantics (operations, sequencing)
  - Defines instruction format (bit encoding)
  - Examples: *MIPS, RISC-V, Alpha, x86, IBM 360, VAX, ARM, JVM*

Summary: Instruction Set Architecture (ISA) versus Implementation

- ISA is the hardware/software interface
  - Defines set of programmer visible state
  - Defines data types
  - Defines instruction semantics (operations, sequencing)
  - Defines instruction format (bit encoding)
  - 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, ...*

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

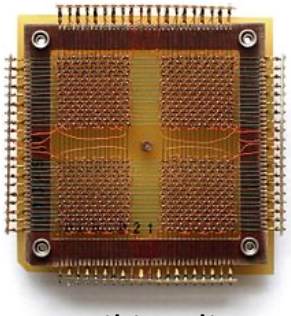
Microarchitecture	CPI	cycle time
Microcoded	>1	short
Single-cycle unpipelined	1	long
Pipelined	1	short

Memory and Caches

Mengjia Yan
Computer Science and Artificial Intelligence Laboratory
M.I.T.

Memory Technology

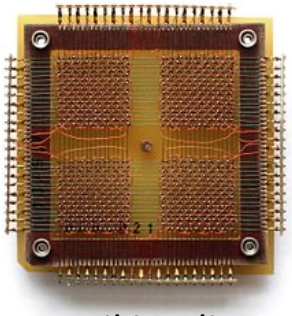
- Early machines used a variety of memory technologies
  - Manchester Mark I used CRT Memory Storage
  - EDVAC used a mercury delay line



Wikipedia

Memory Technology

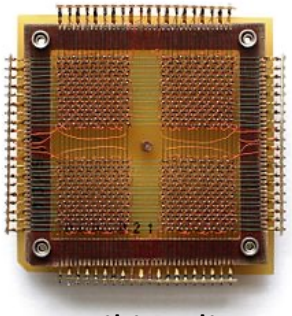
- Early machines used a variety of memory technologies
  - Manchester Mark I used CRT Memory Storage
  - EDVAC used a mercury delay line
- Core memory was first large scale reliable main memory
  - Invented by Forrester in late 40s at MIT for Whirlwind project
  - Bits stored as magnetization polarity on small ferrite cores threaded onto 2 dimensional grid of wires



Wikipedia

Memory Technology

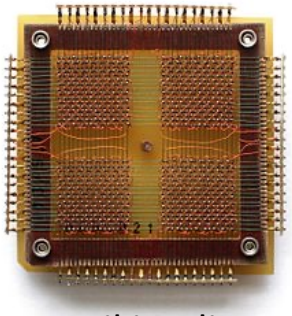
- Early machines used a variety of memory technologies
  - Manchester Mark I used CRT Memory Storage
  - EDVAC used a mercury delay line
- Core memory was first large scale reliable main memory
  - Invented by Forrester in late 40s at MIT for Whirlwind project
  - Bits stored as magnetization polarity 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



Wikipedia

Memory Technology

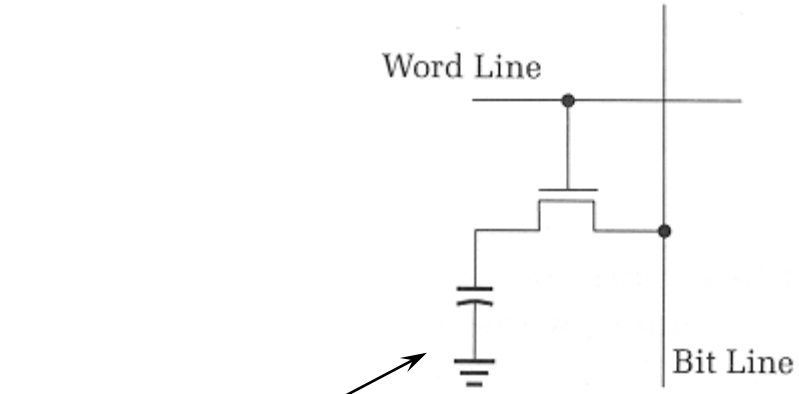
- Early machines used a variety of memory technologies
  - Manchester Mark I used CRT Memory Storage
  - EDVAC used a mercury delay line
- Core memory was first large scale reliable main memory
  - Invented by Forrester in late 40s at MIT for Whirlwind project
  - Bits stored as magnetization polarity 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



Wikipedia

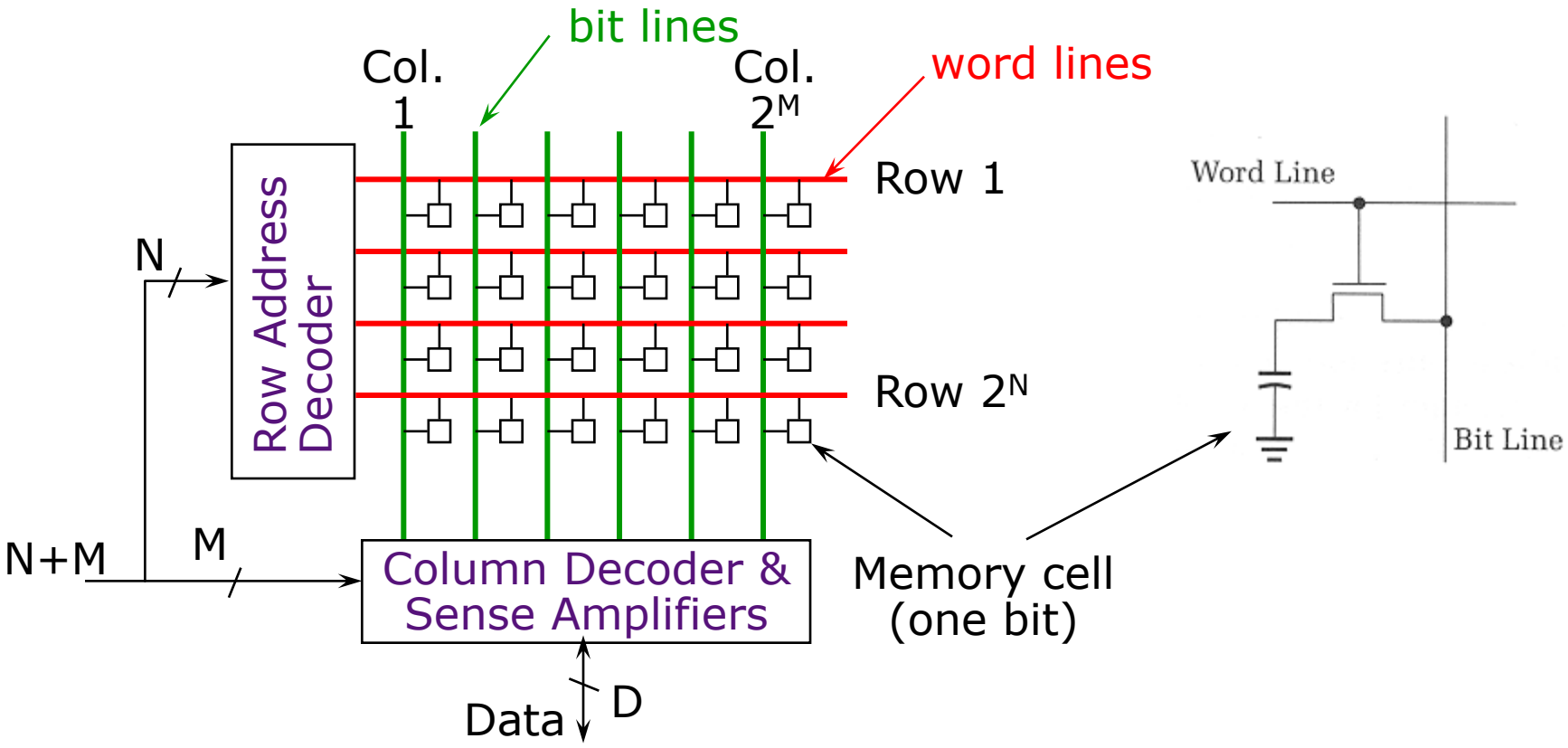
- Flash memory
  - Slower, but denser than DRAM. Also non-volatile, but with wearout issues
- Phase change memory (PCM, 3D XPoint)
  - Slightly slower, but much denser than DRAM and non-volatile

DRAM Architecture



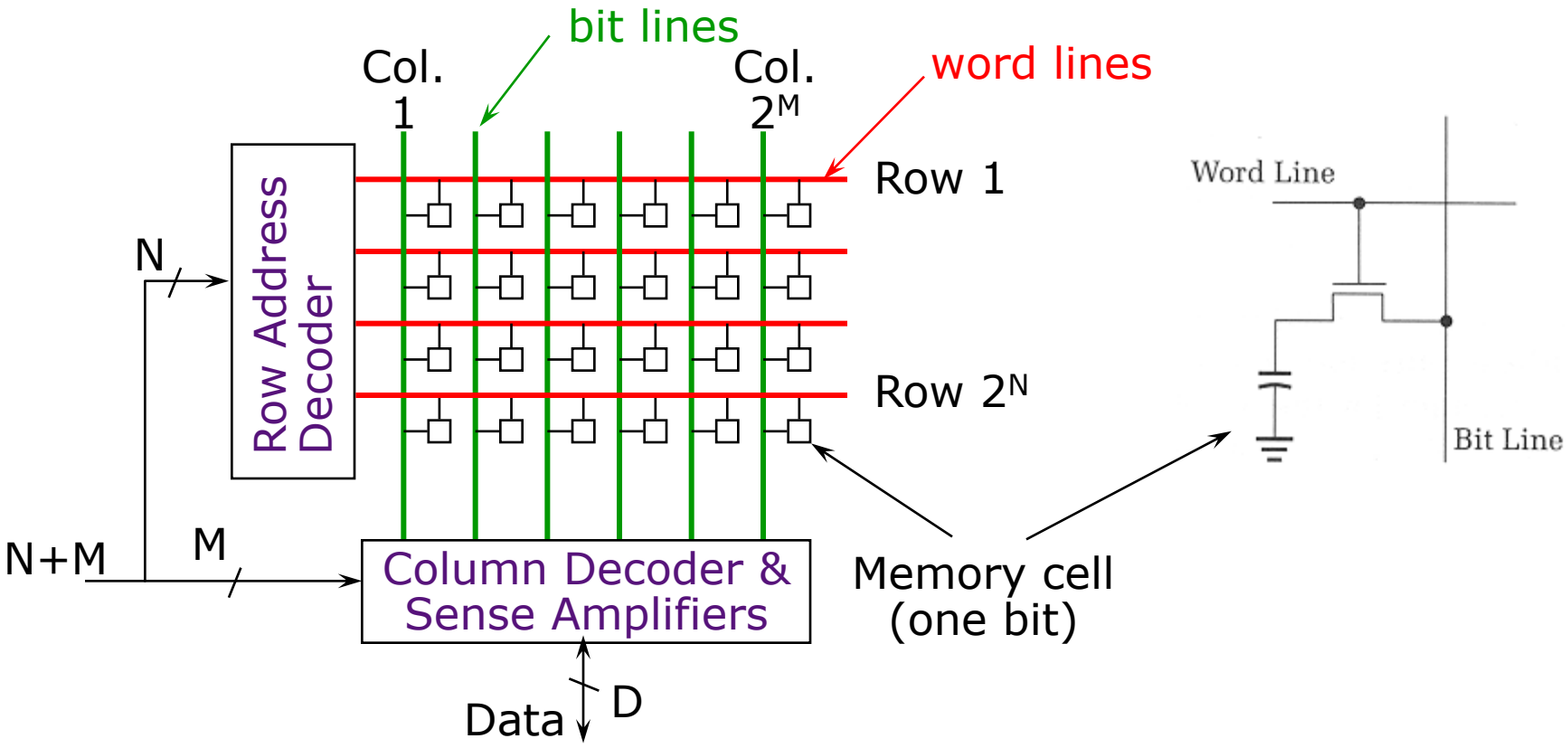
Memory cell
(one bit)

DRAM Architecture



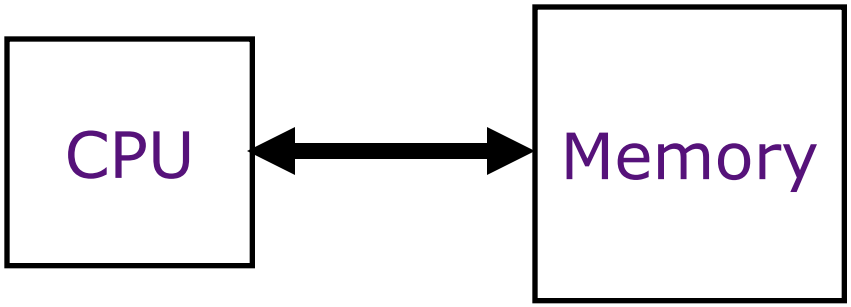
- Bits stored in 2-dimensional arrays on chip

DRAM Architecture



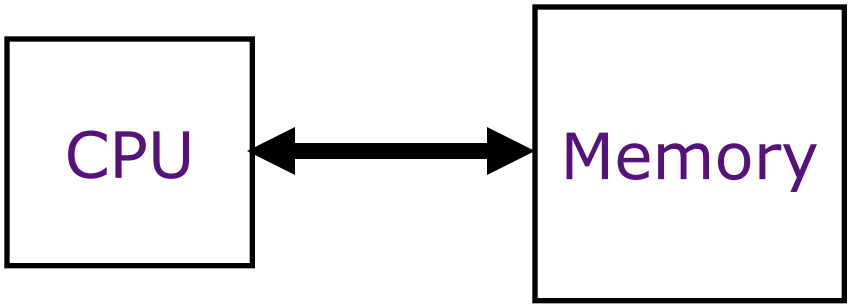
- Bits stored in 2-dimensional arrays on chip
- Modern chips have around 8 logical banks on each chip
  - Each logical bank physically implemented as many smaller arrays

CPU-Memory Metrics



- 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

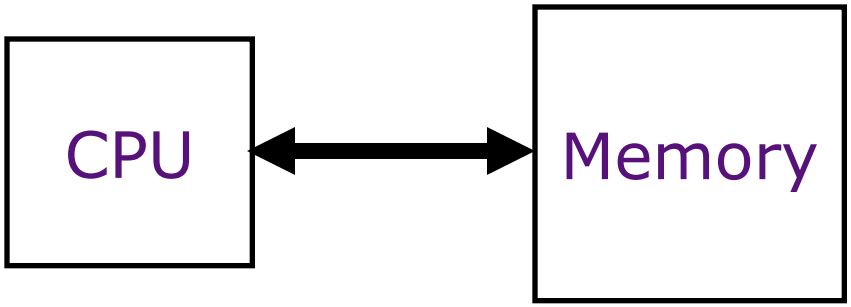
CPU-Memory Metrics



- 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

CPU-Memory Metrics

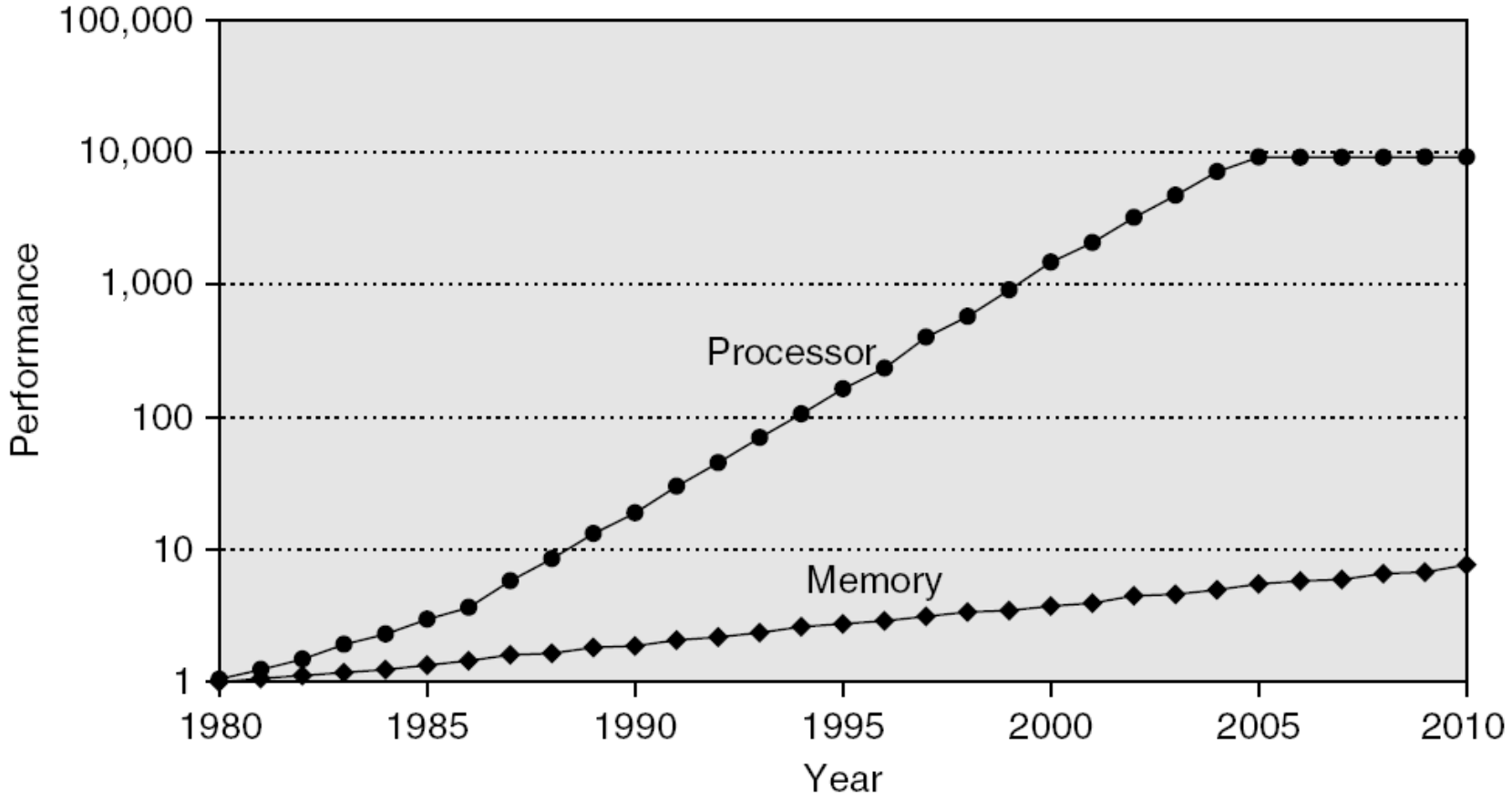


- Latency (time for a single access)
  Memory access time >> Processor cycle time

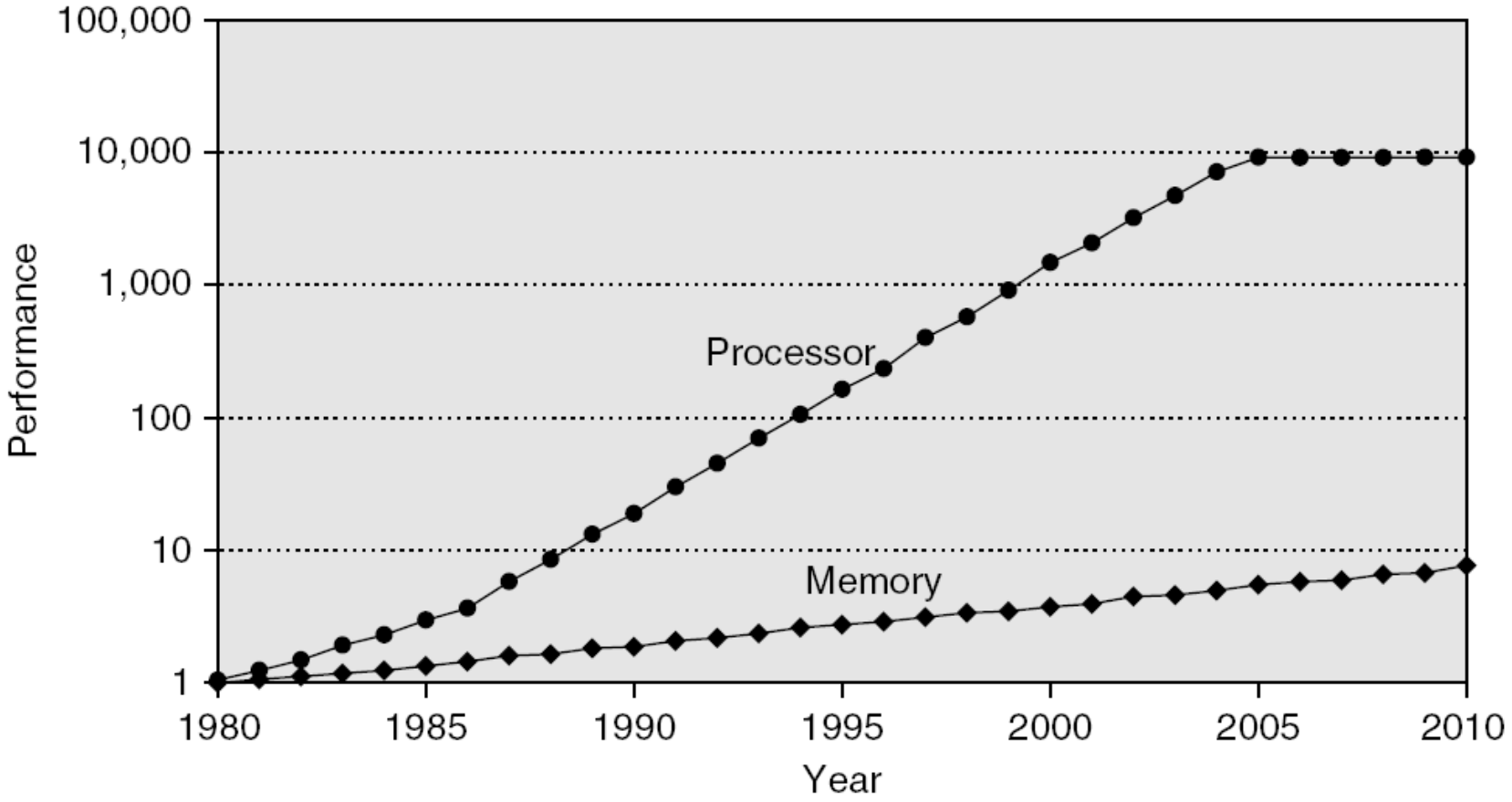
- Bandwidth (number of accesses per unit time)
  if fraction *m* of instructions access memory,
  $\Rightarrow$1+*m* memory references / instruction
  $\Rightarrow$ CPI = 1 requires 1+*m* memory refs / cycle

- Energy (nJ per access)

Processor-DRAM Gap (latency)



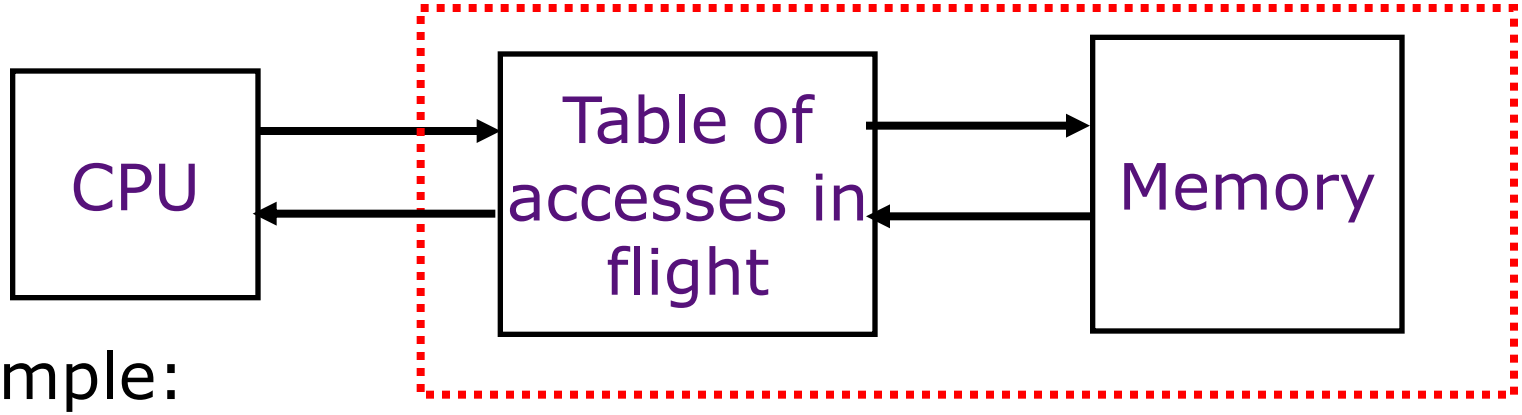
Processor-DRAM Gap (latency)



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)

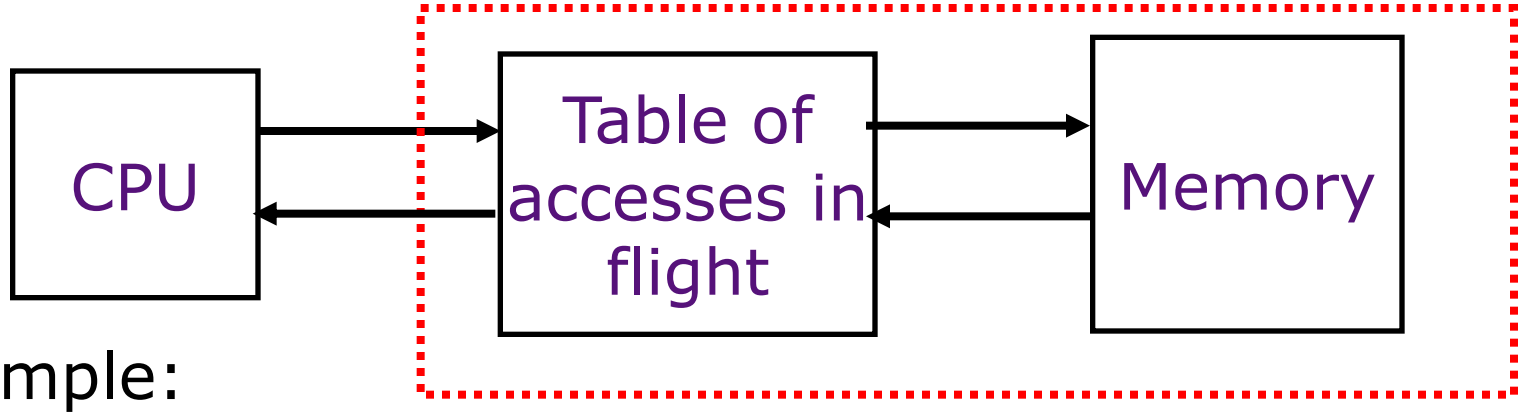


Example:

--- 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)



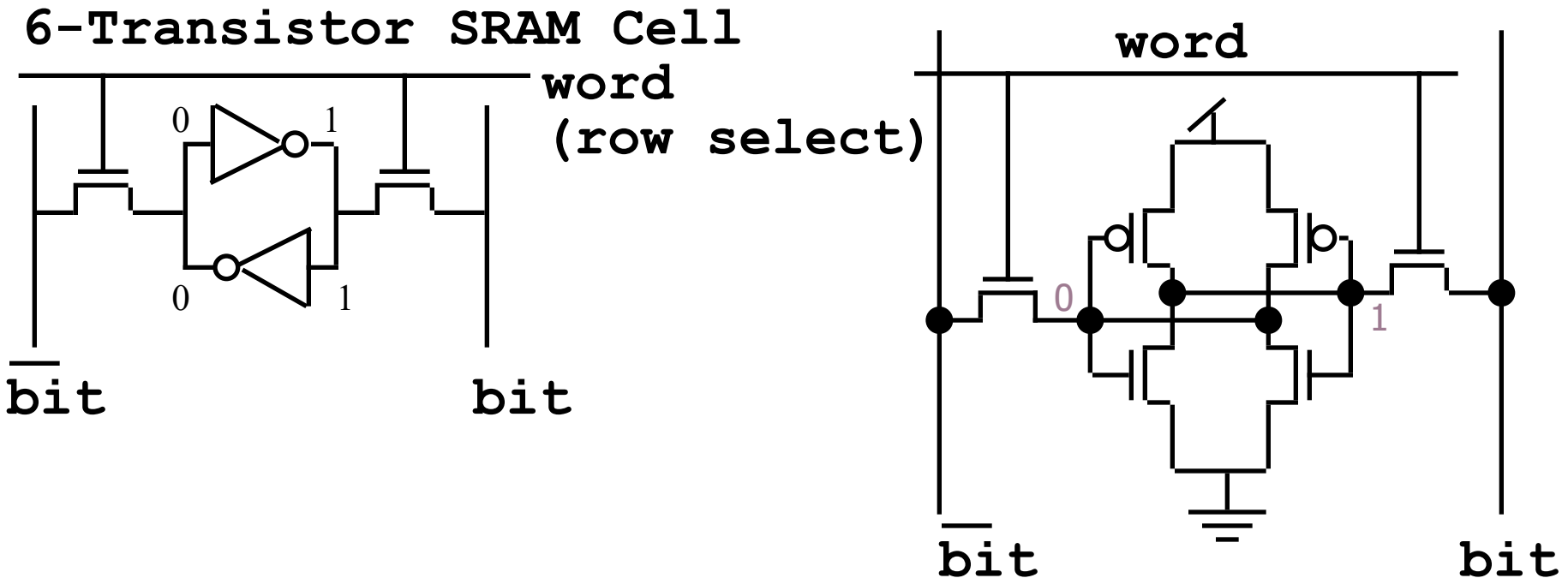
Example:

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

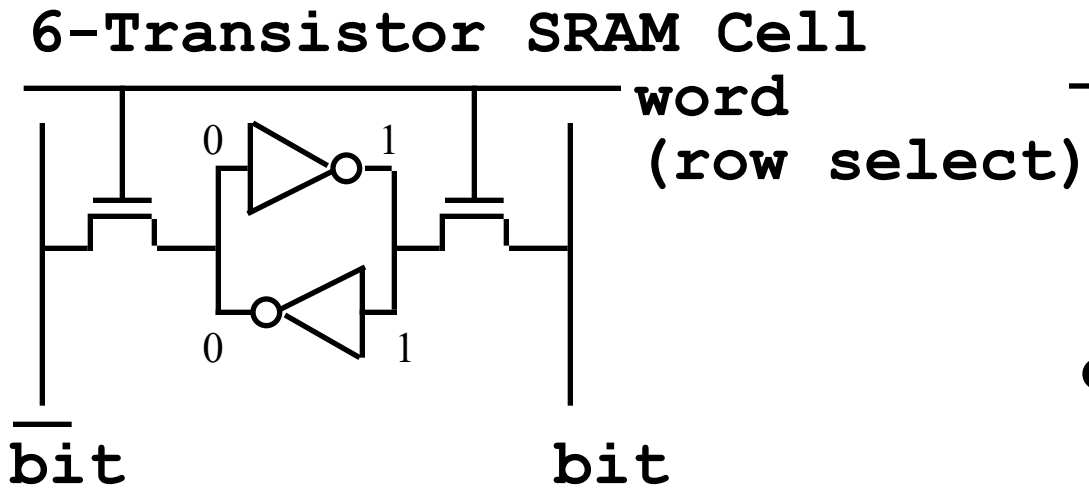
*⇒ Table size = 1.2 * 100 = 120 entries*

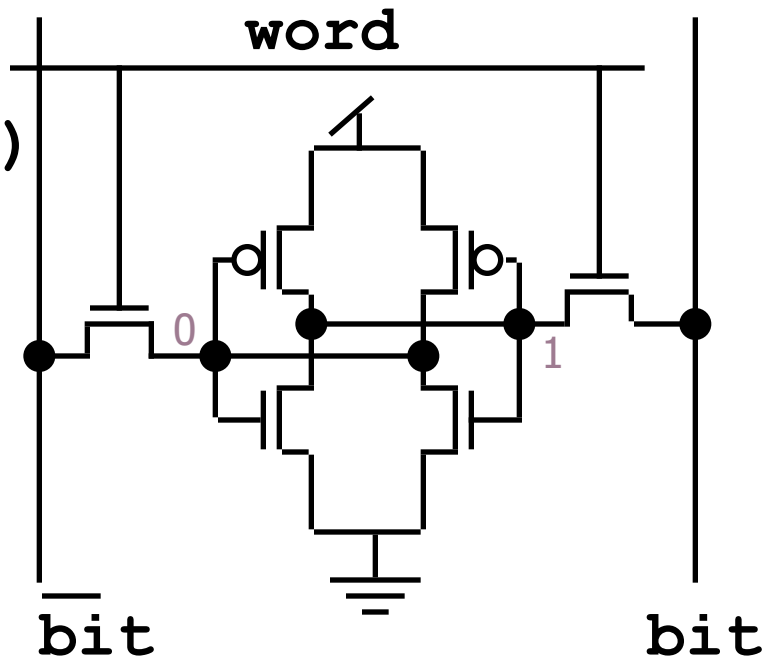
120 independent memory operations in flight!

Basic Static RAM Cell



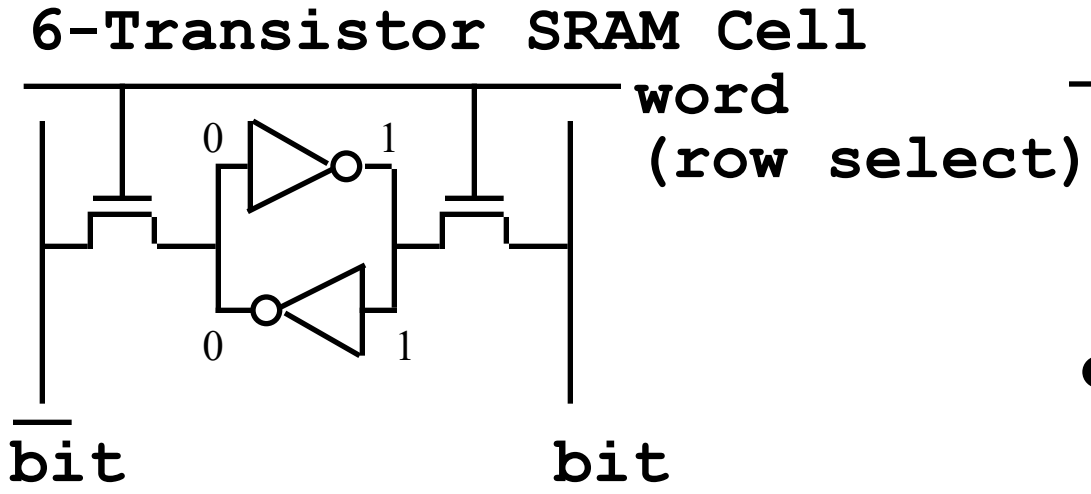
Basic Static RAM Cell

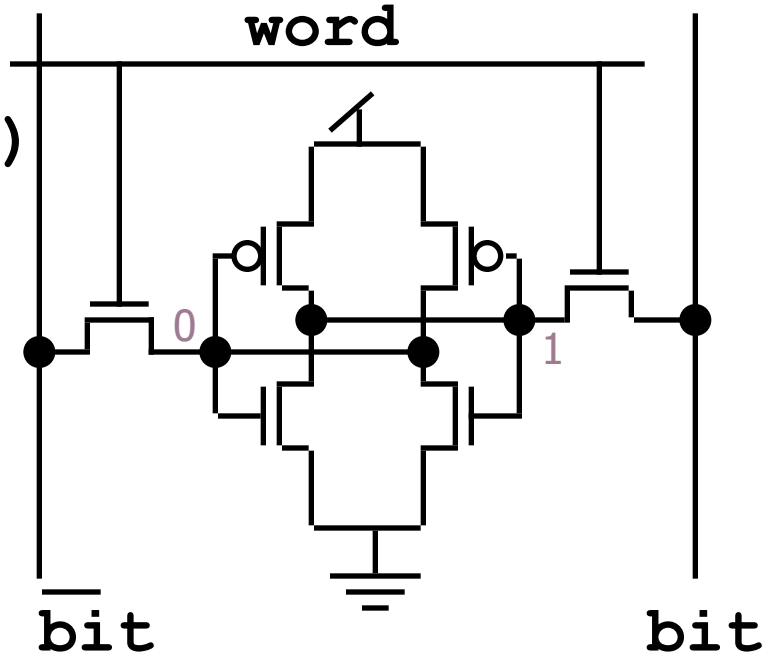




- Write:
  1. Drive bit lines (bit=1, $\overline{\text{bit}}$=0)
  2. Select word line

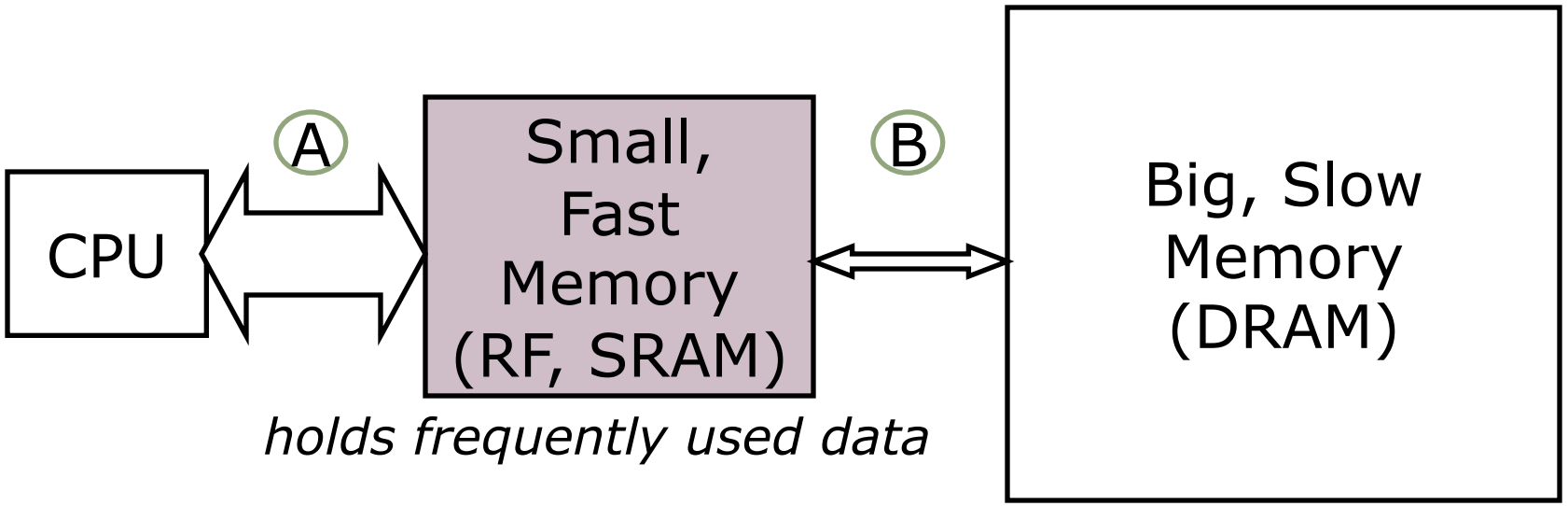
Basic Static RAM Cell



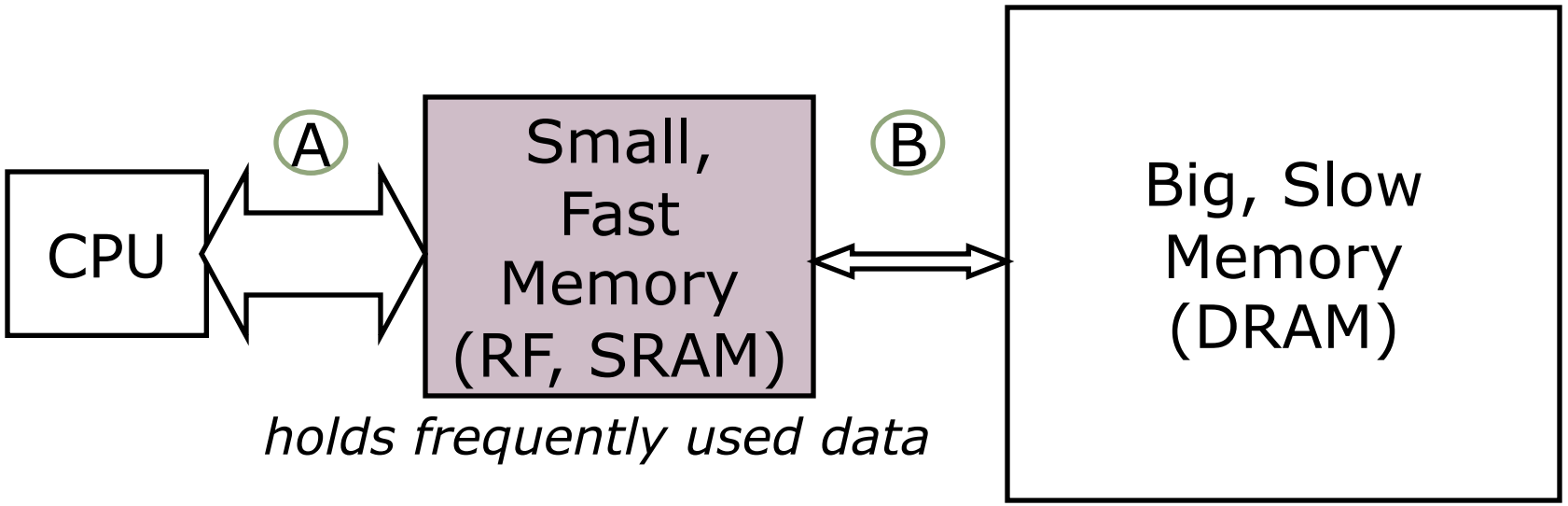


- Write:
  1. Drive bit lines (bit=1, $\overline{\text{bit}}$=0)
  2. Select word line
- Read:
  1. Precharge bit and $\overline{\text{bit}}$ to Vdd
  2. Select word line
  3. Cell pulls one bit line low
  4. Column sense amp detects difference between bit & $\overline{\text{bit}}$

Memory Hierarchy

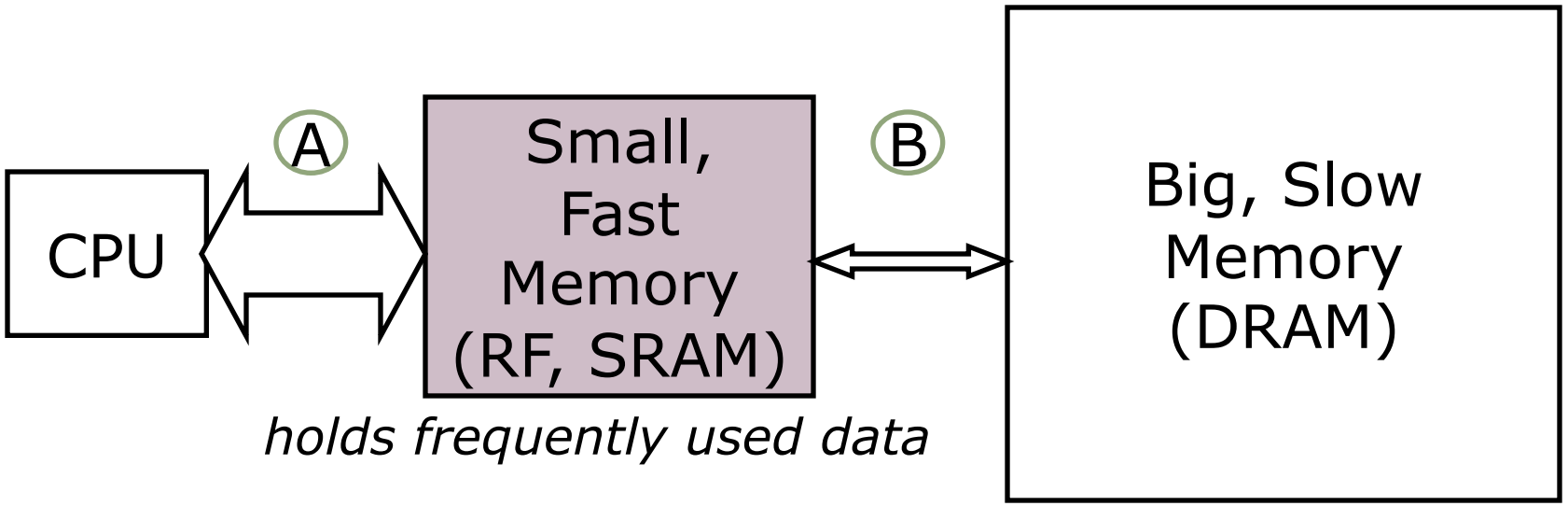


Memory Hierarchy



- *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:

data ∈ fast memory ⇒ low latency access

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

Multilevel Memory

Strategy: Reduce 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:

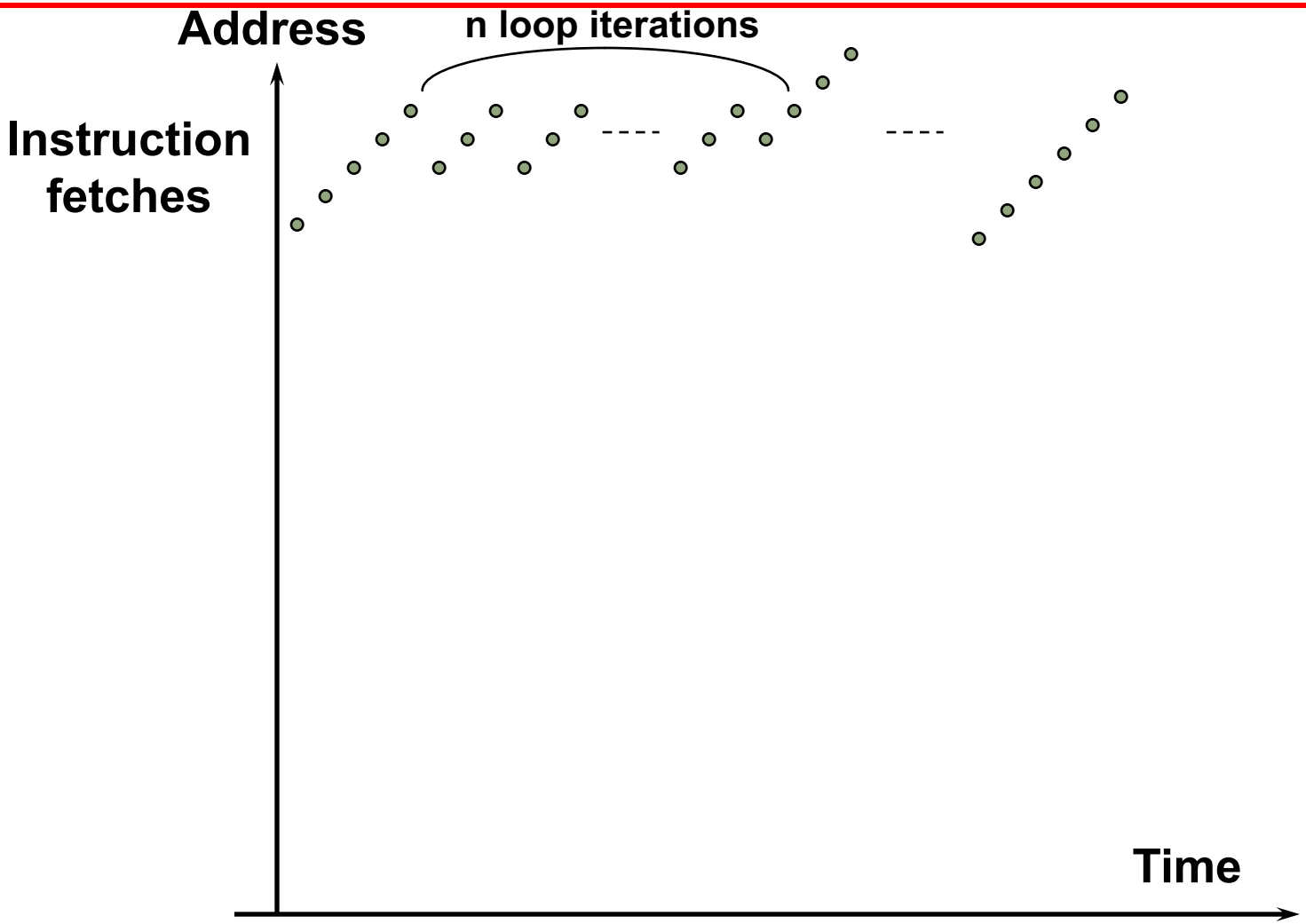
Multilevel Memory

Strategy: Reduce average latency using small, fast memories called caches.

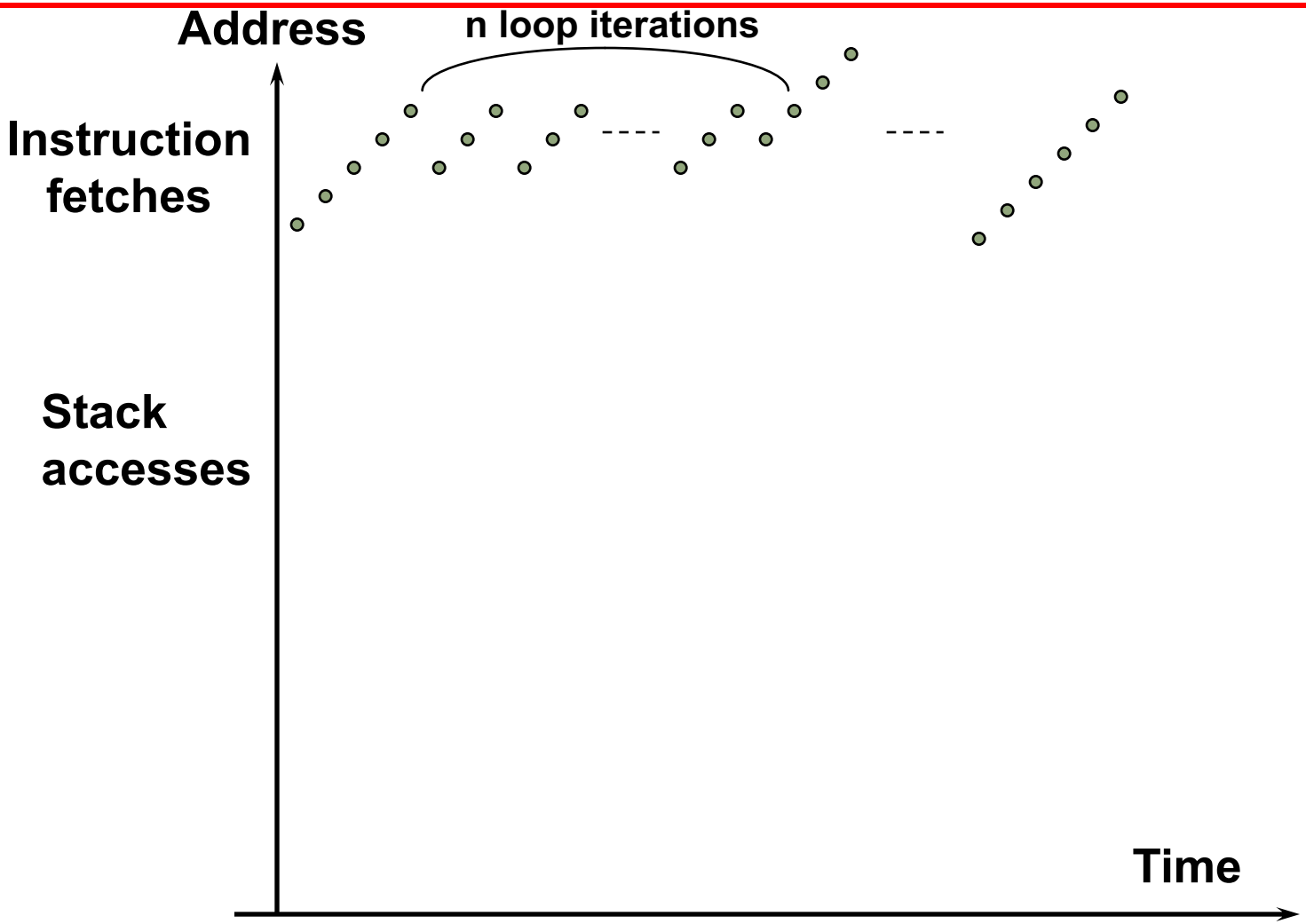
Caches are a mechanism to reduce memory latency based on the **empirical** observation that the patterns of memory references made by a processor are often highly predictable:

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

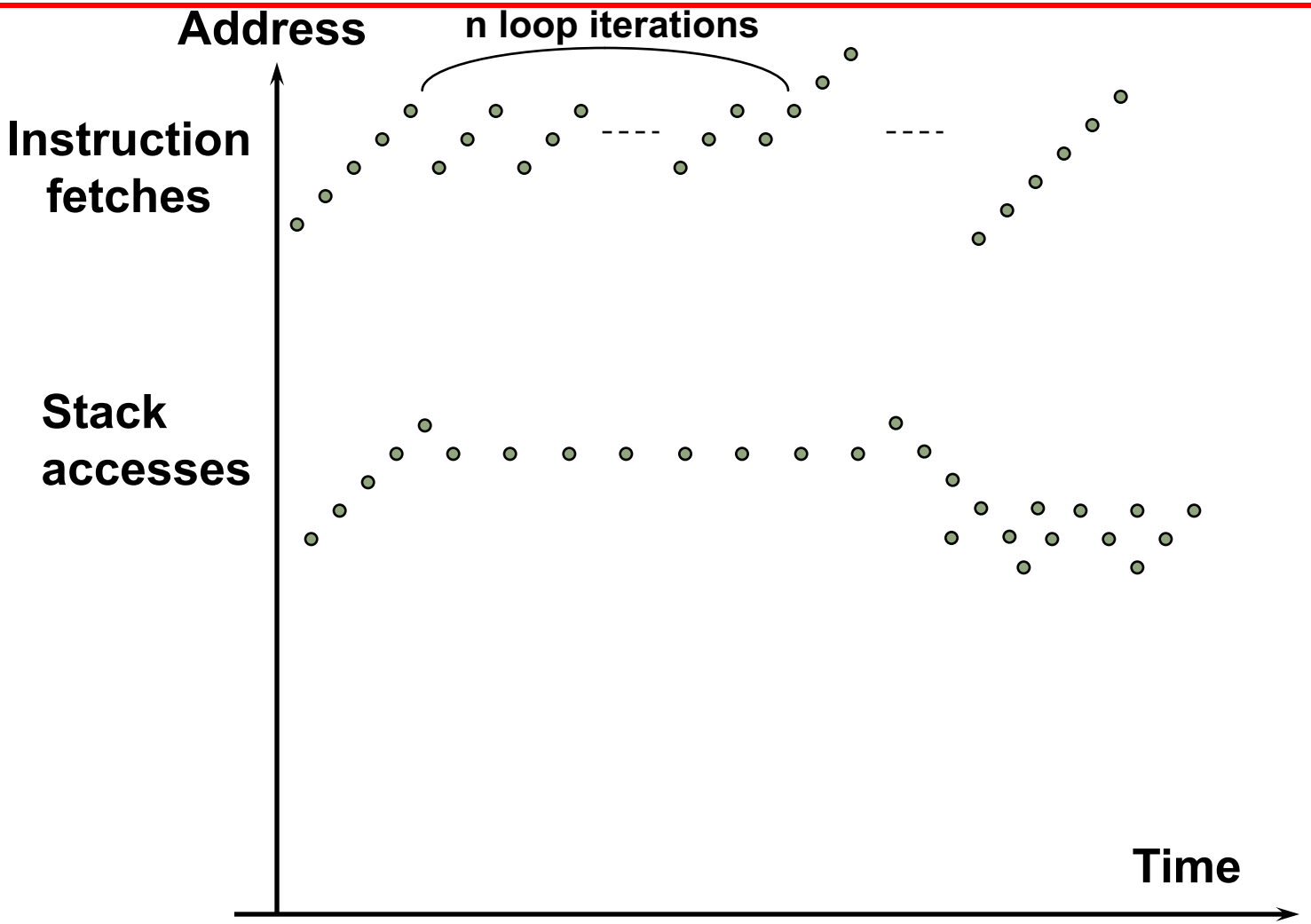
Typical Memory Reference Patterns



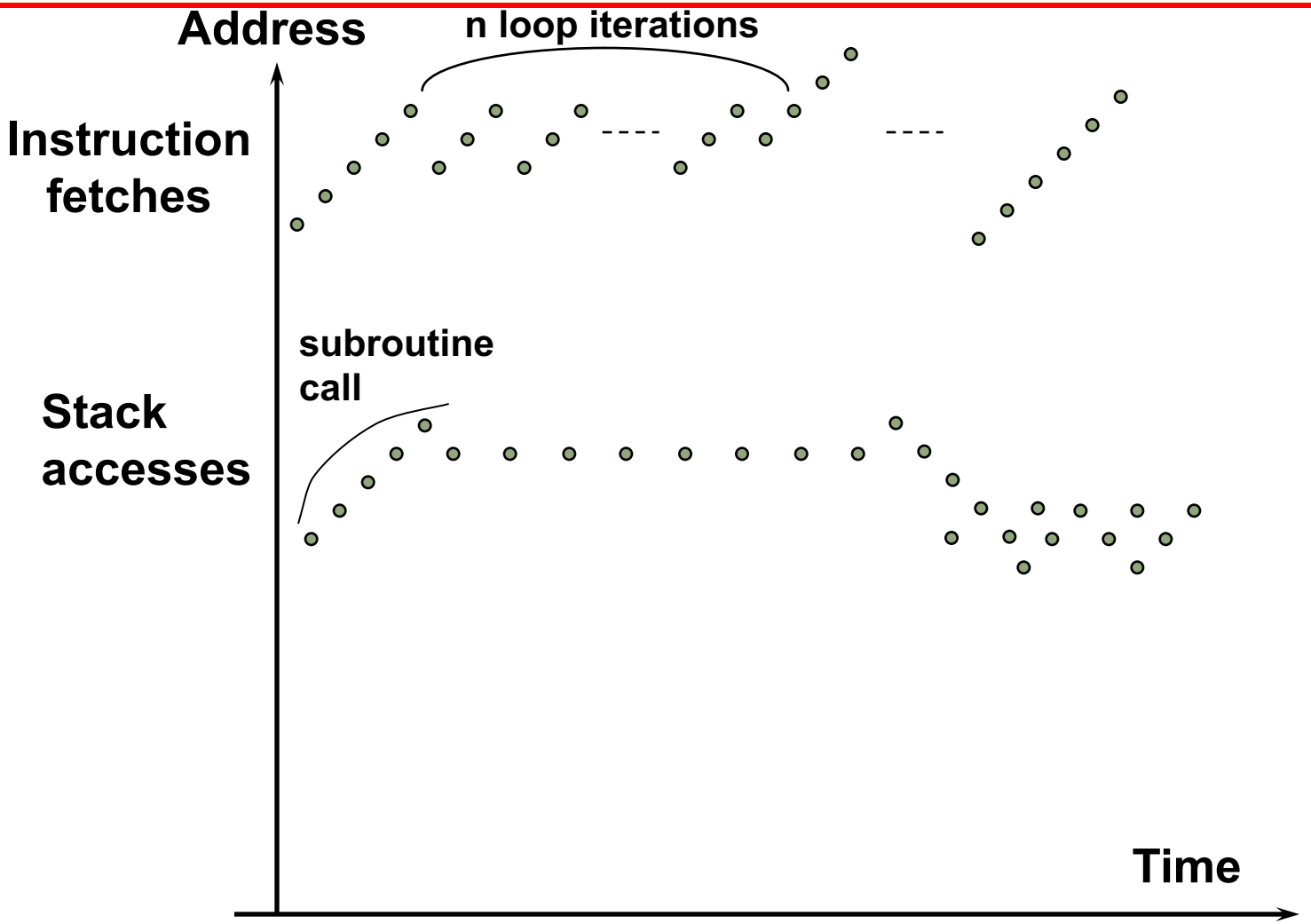
Typical Memory Reference Patterns



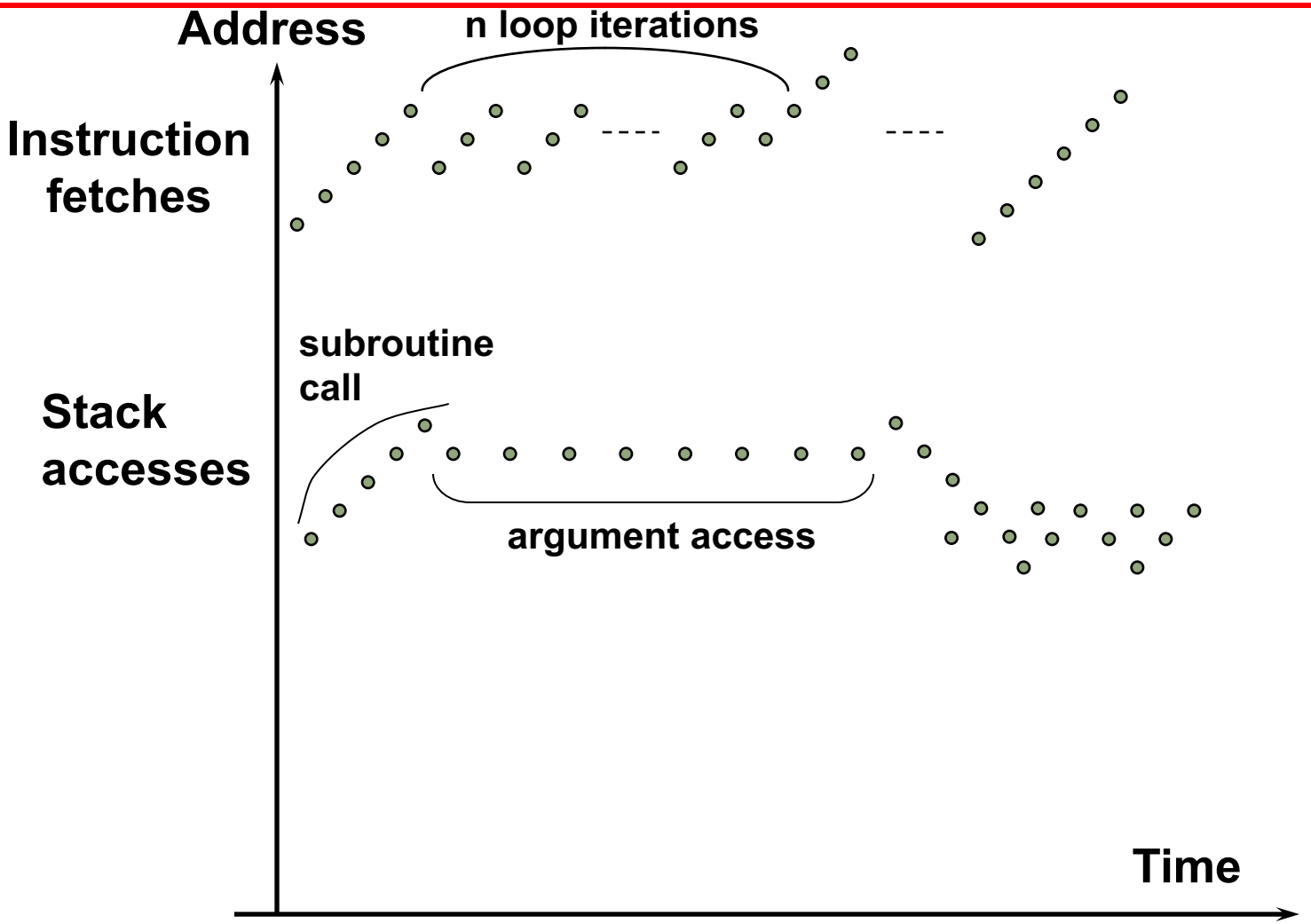
Typical Memory Reference Patterns



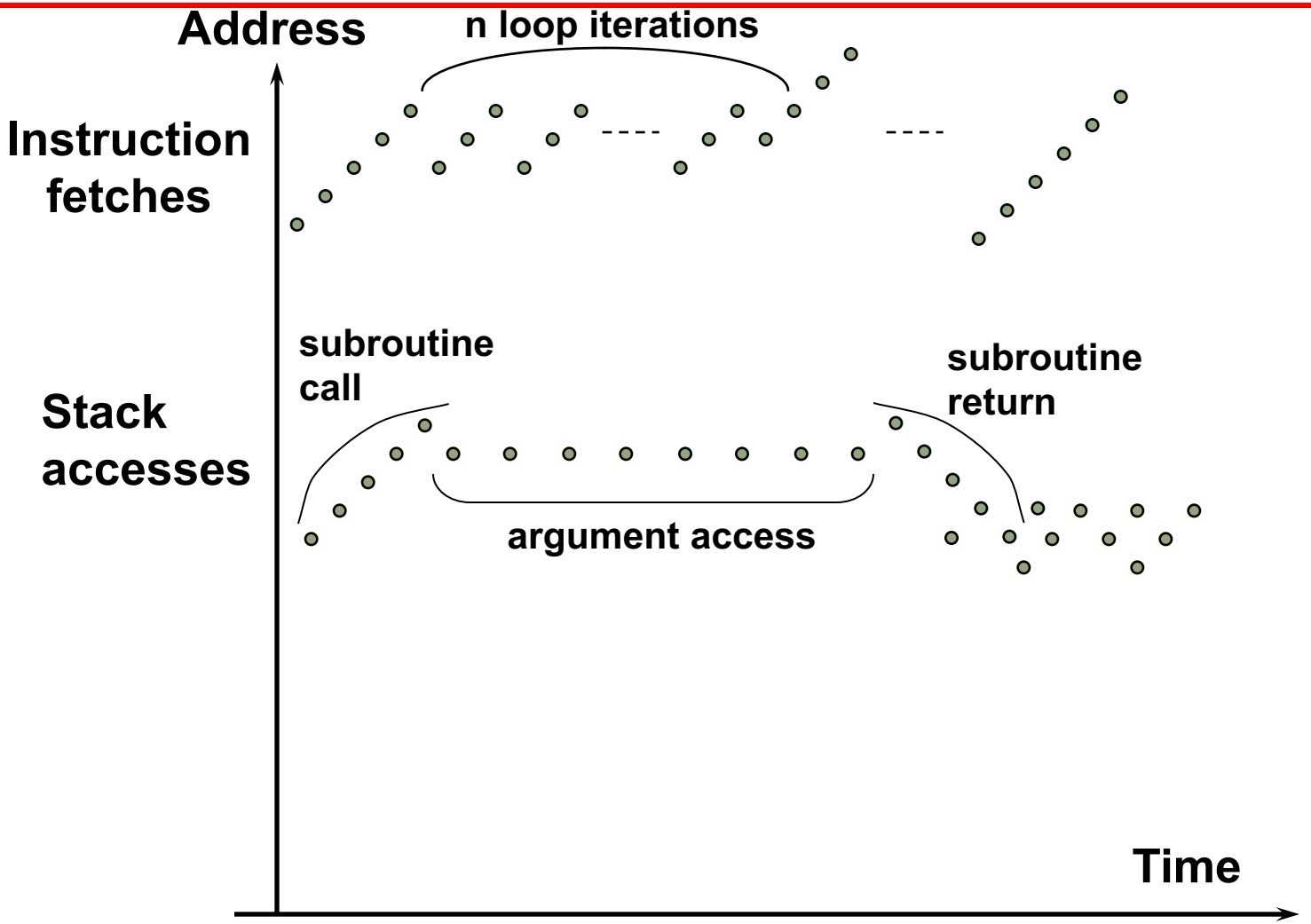
Typical Memory Reference Patterns



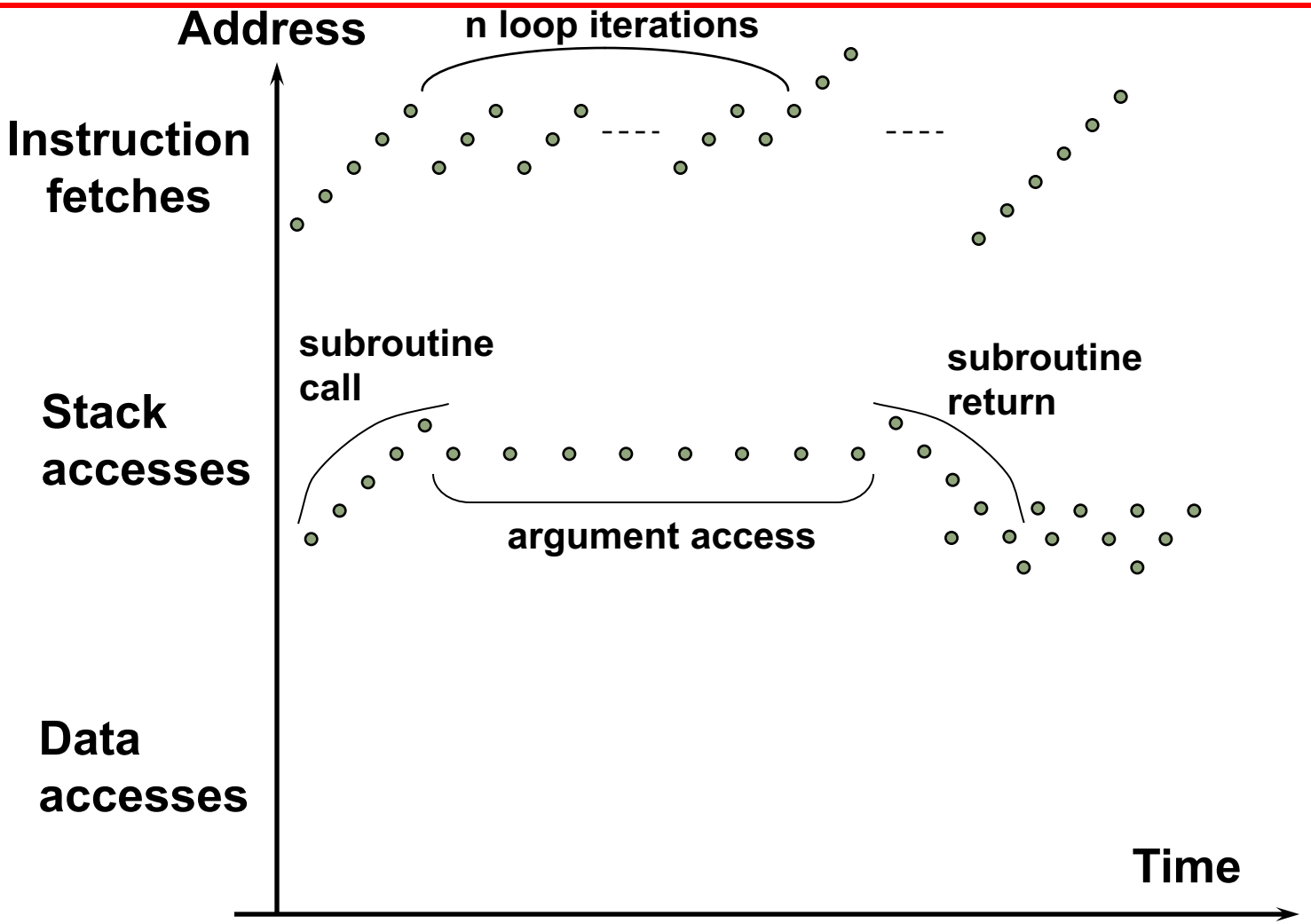
Typical Memory Reference Patterns



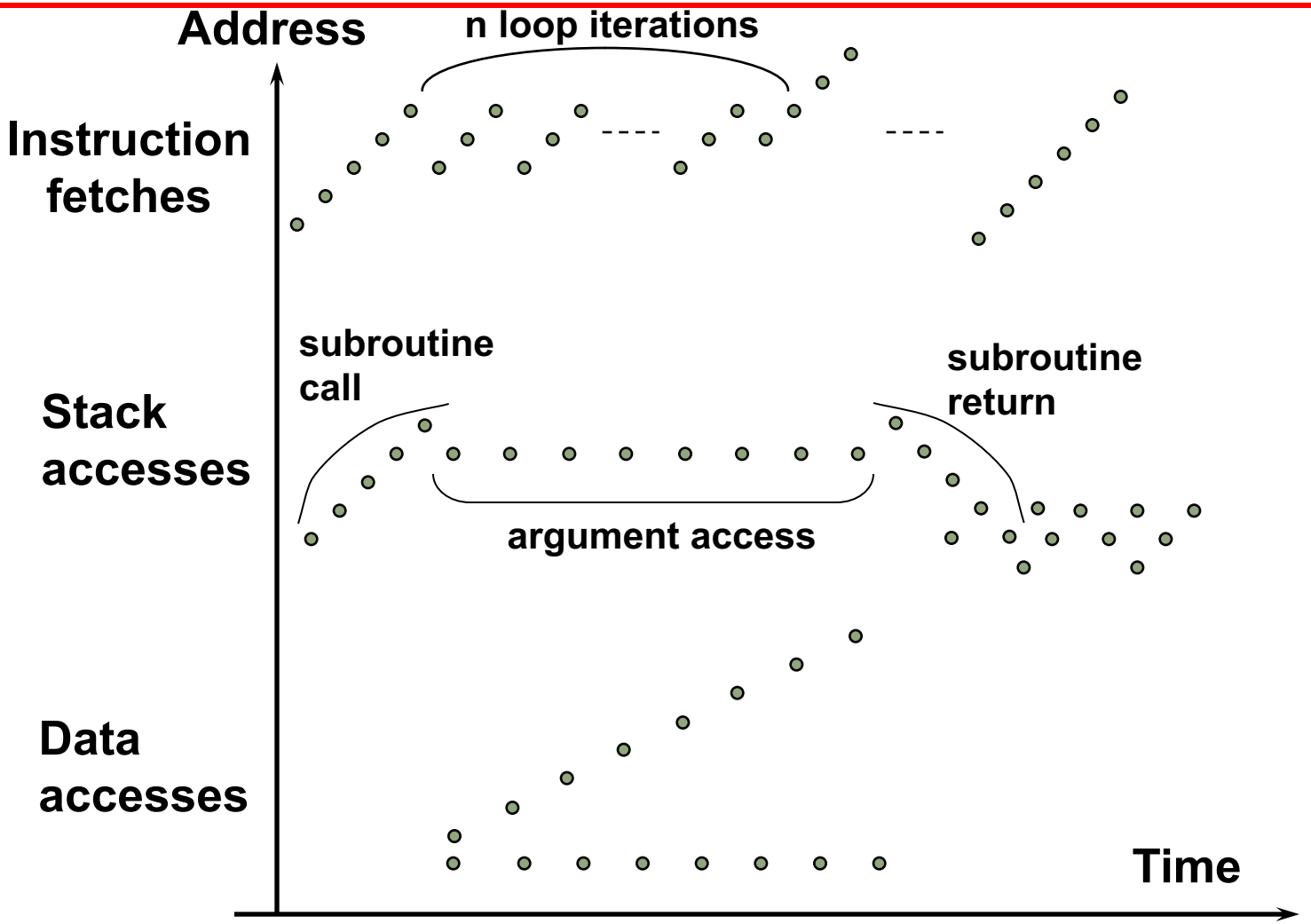
Typical Memory Reference Patterns



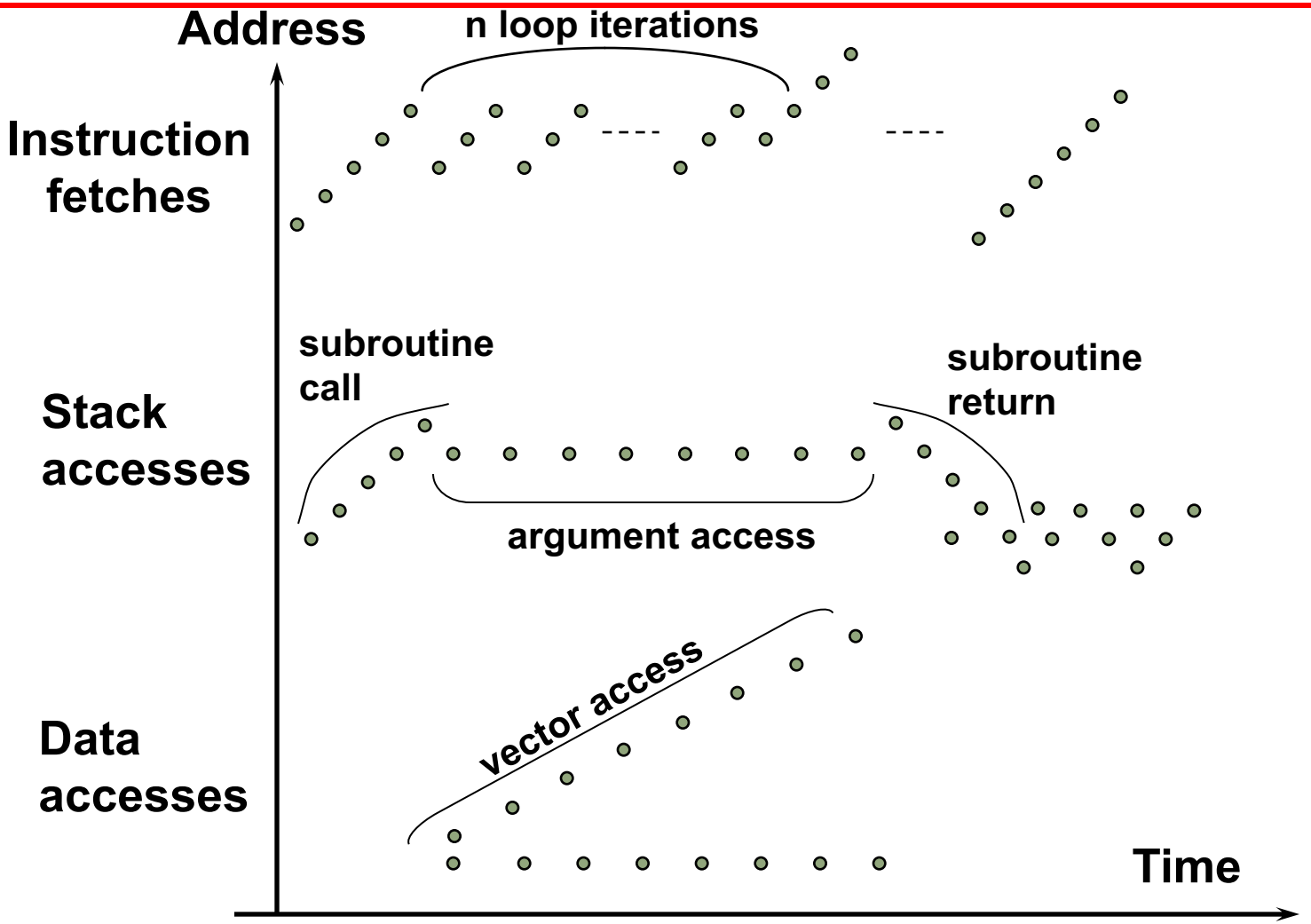
Typical Memory Reference Patterns



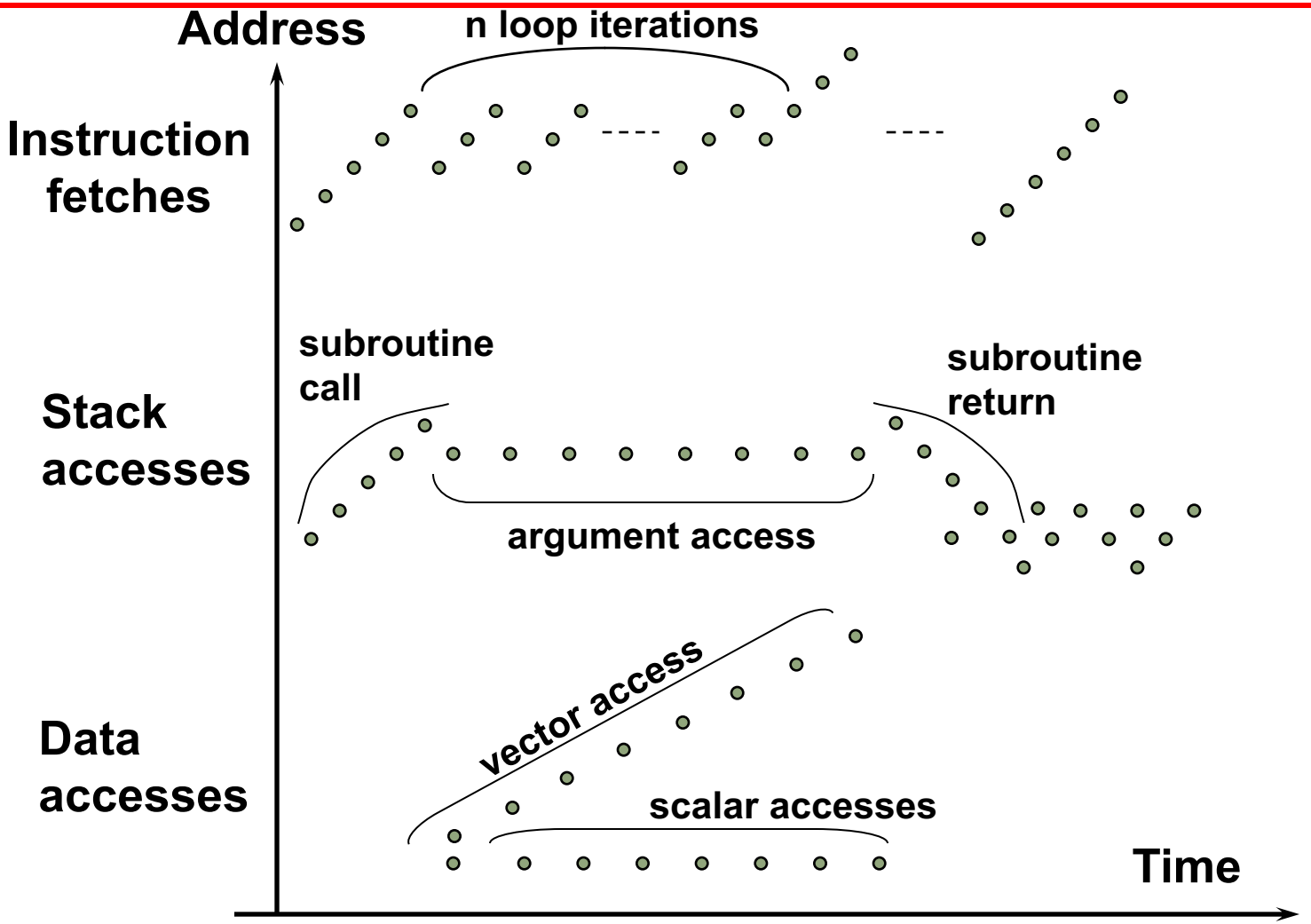
Typical Memory Reference Patterns



Typical Memory Reference Patterns



Typical Memory Reference Patterns



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

Data Orchestration Techniques

Two approaches to controlling data movement in the memory hierarchy:

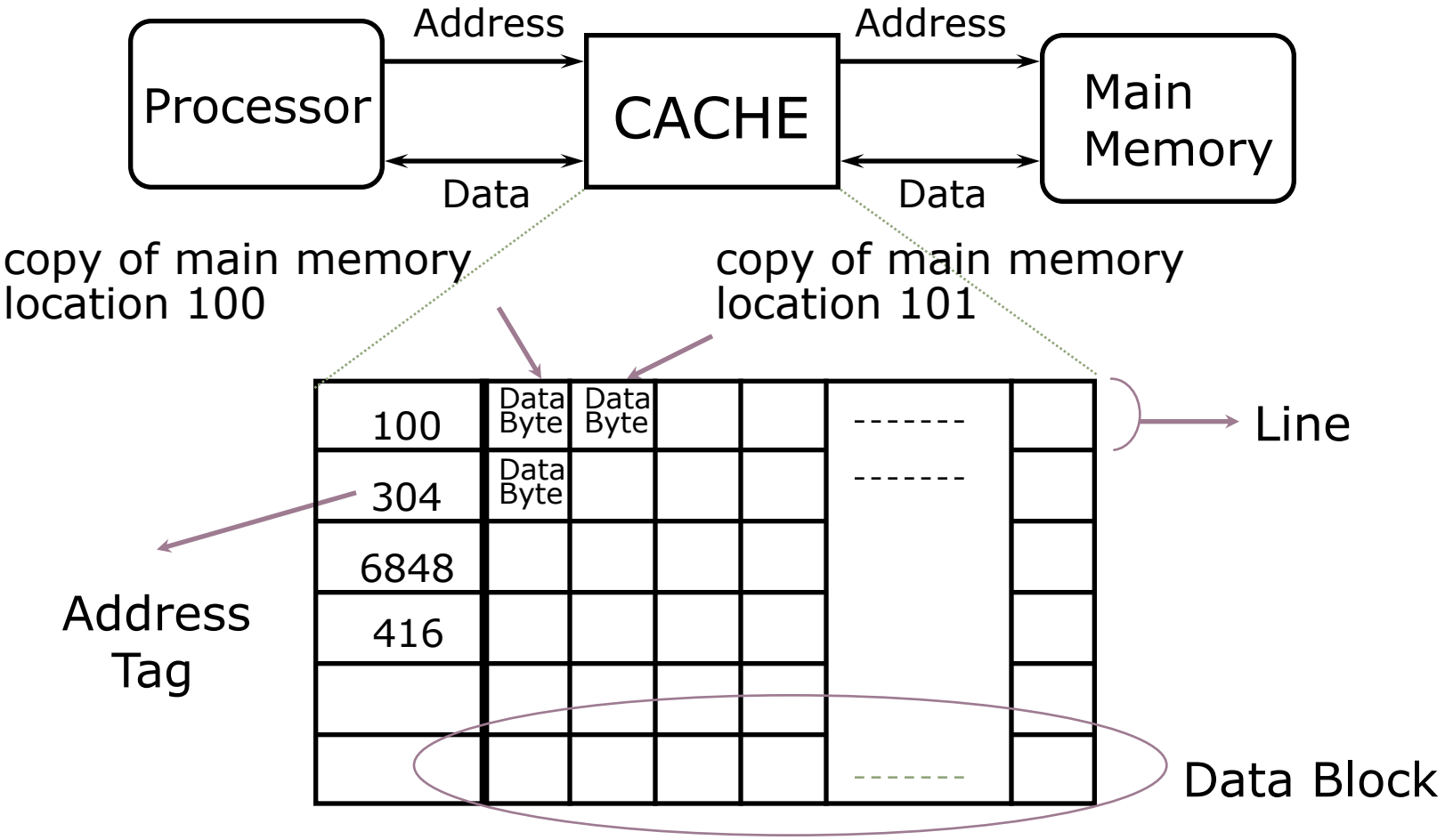
- *Explicit:* Manually at the direction of the programmer using instructions
- *Implicit:* Automatically by the hardware in response to a request by an instruction, but transparent to the programmer.

Management of Memory Hierarchy

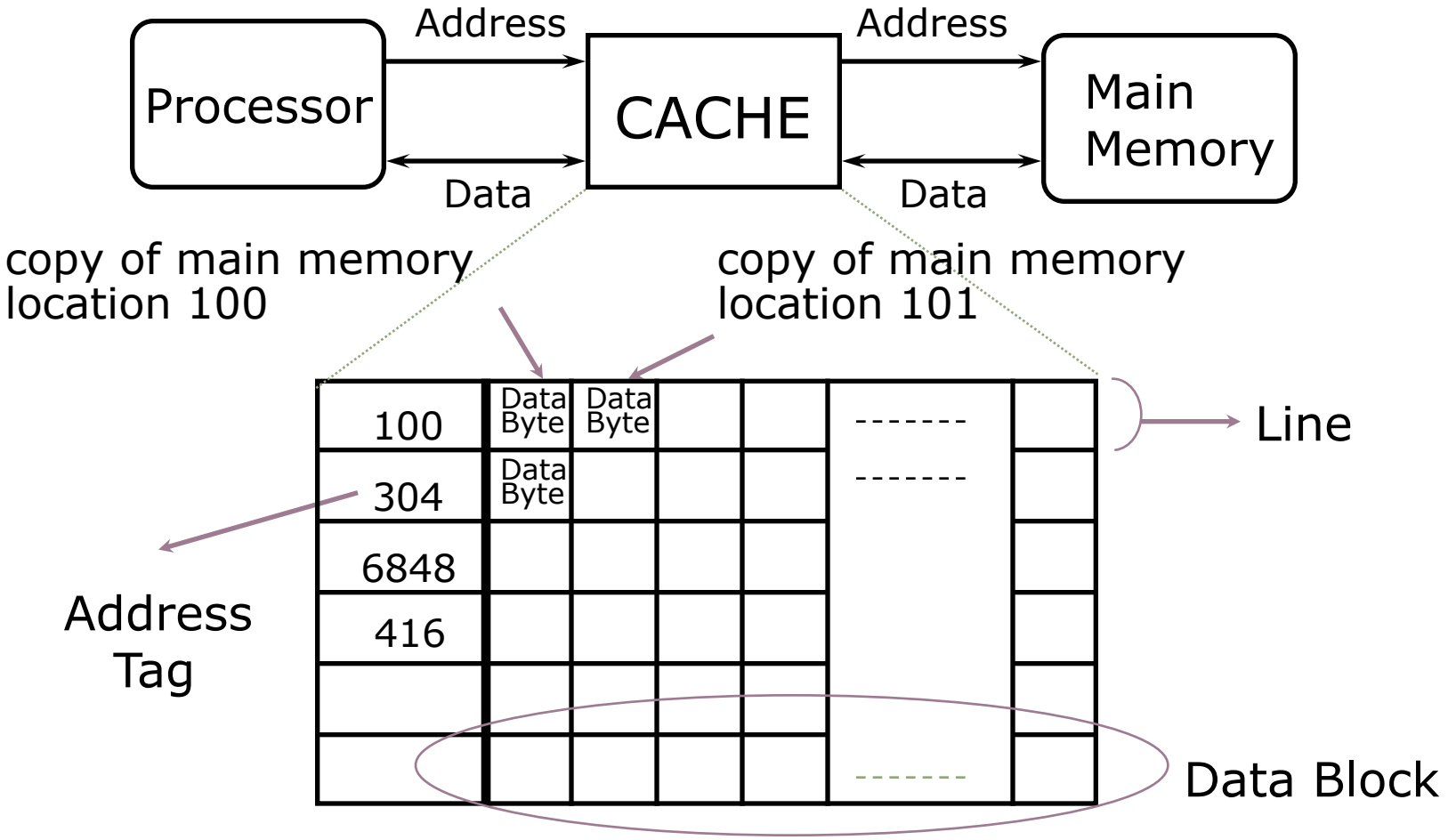
- Small/fast storage, e.g., registers
  - Address usually specified directly in instruction
  - Generally implemented using **explicit** data orchestration
    - e.g., directly as a register file
    - but hardware might do things behind software's back, e.g., stack management, register renaming

- 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

Inside a Cache

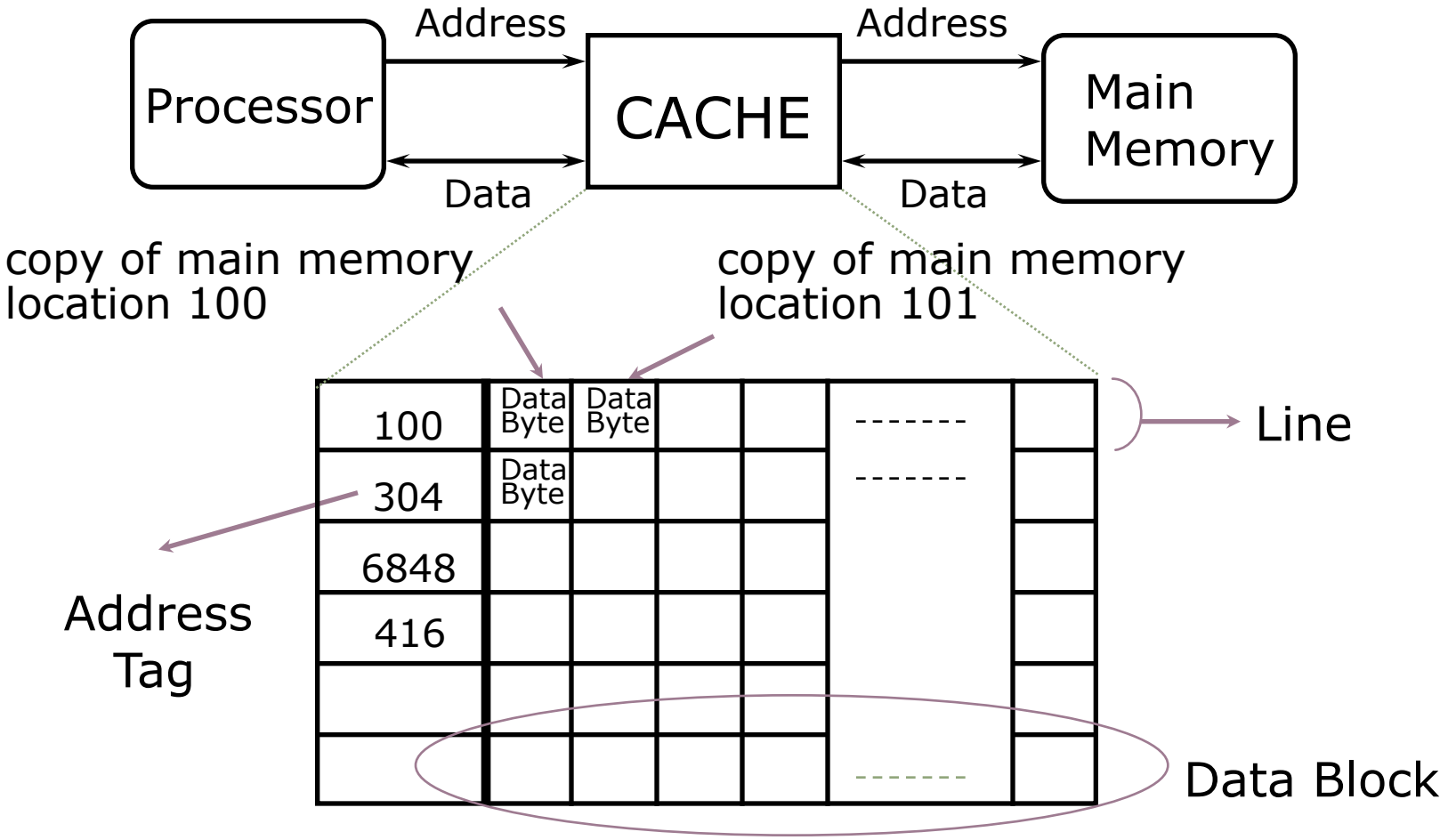


Inside a Cache



Q: How many bits needed in tag? _____

Inside a Cache



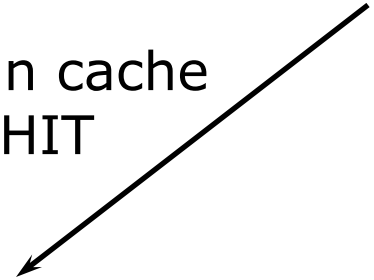
Q: How many bits needed in tag? Enough to uniquely identify block

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

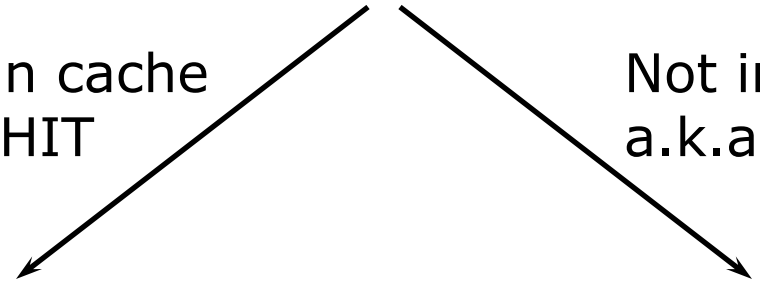


Found in cache
a.k.a. HIT

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

Return copy of data from cache

Not in cache
a.k.a. MISS

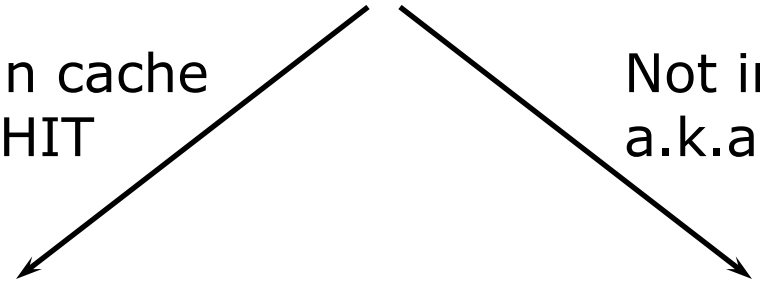
Read block of data from Main Memory

Wait …

Return data to processor and update cache

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

Not in cache
a.k.a. MISS

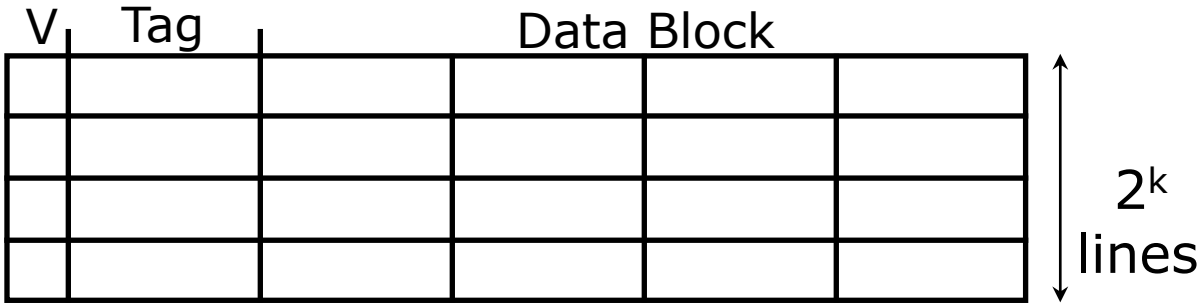
Read block of data from Main Memory

Wait …

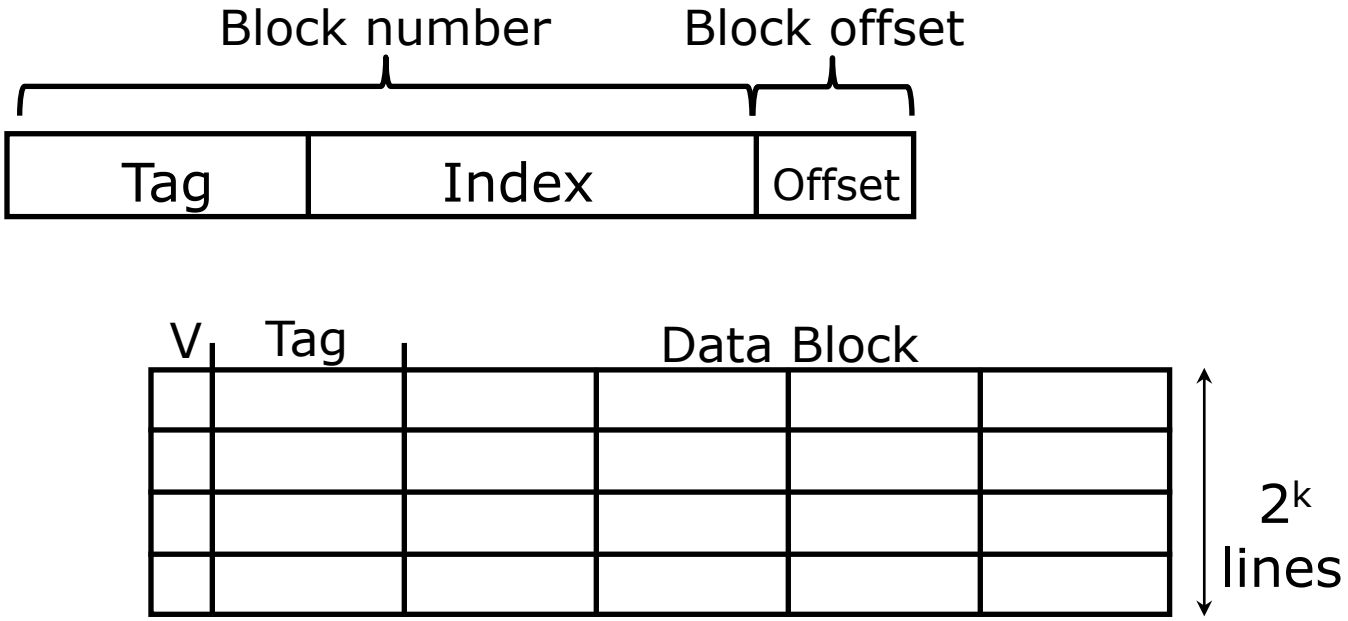
Return data to processor and update cache

Which line do we replace?

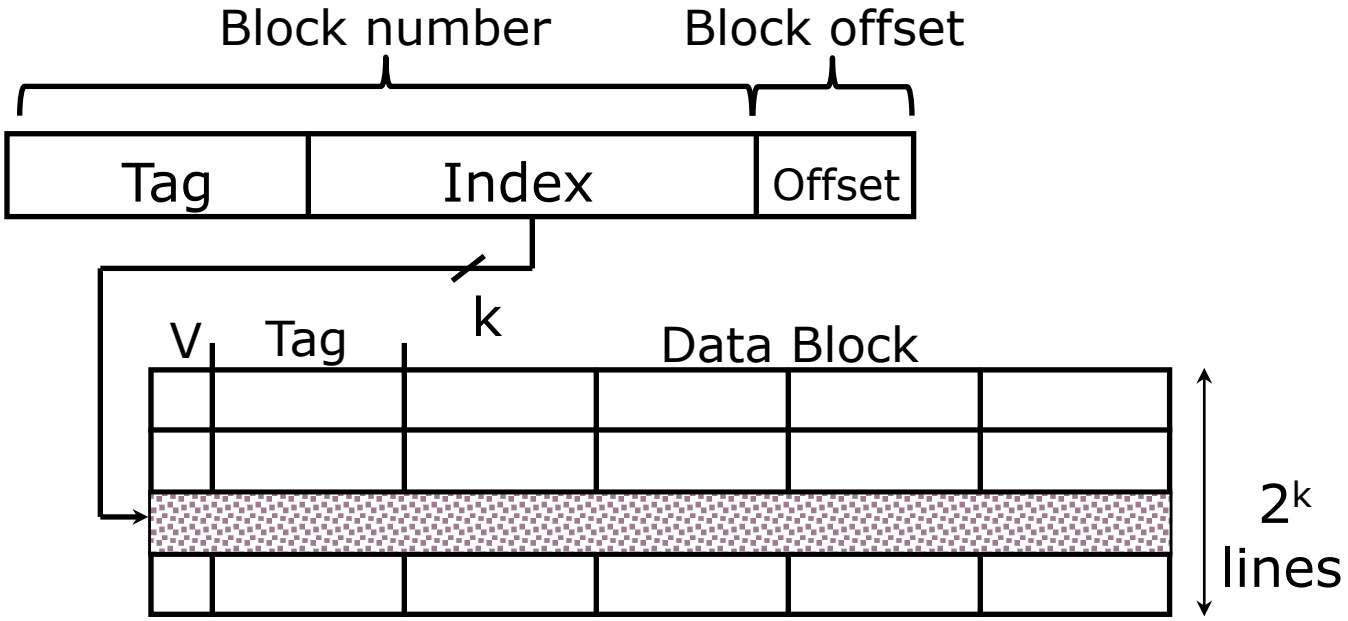
Direct-Mapped Cache



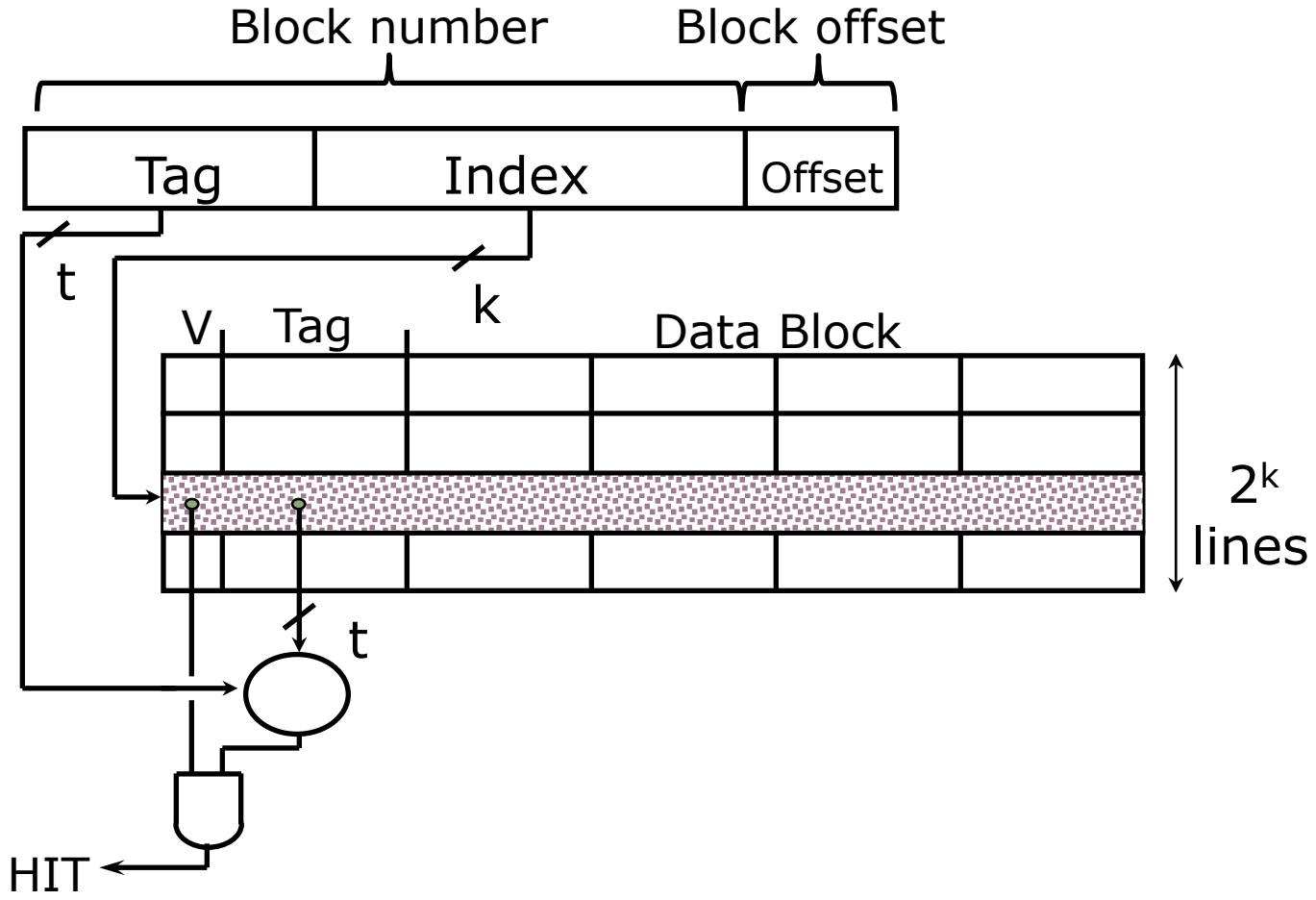
Direct-Mapped Cache



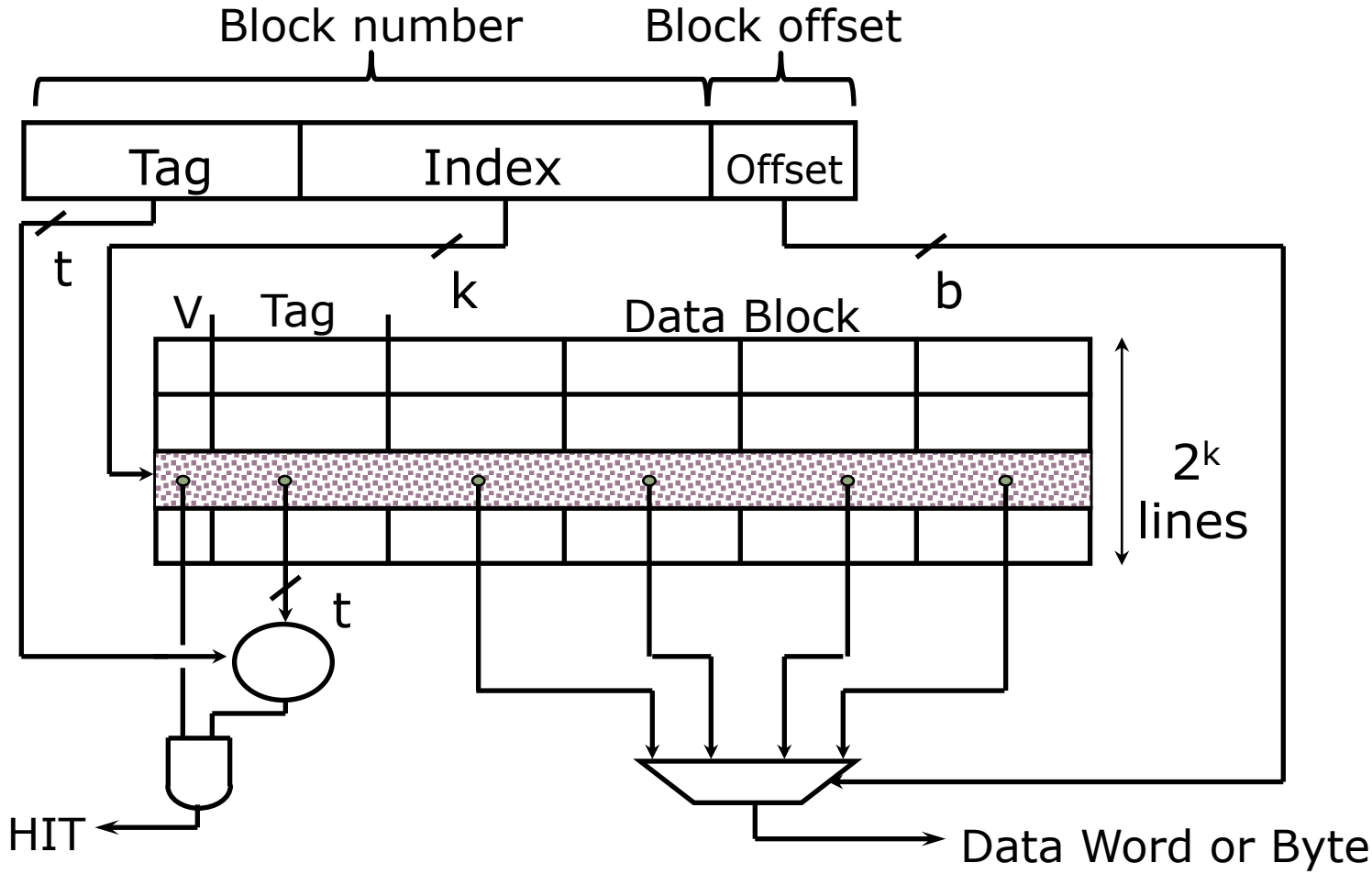
Direct-Mapped Cache



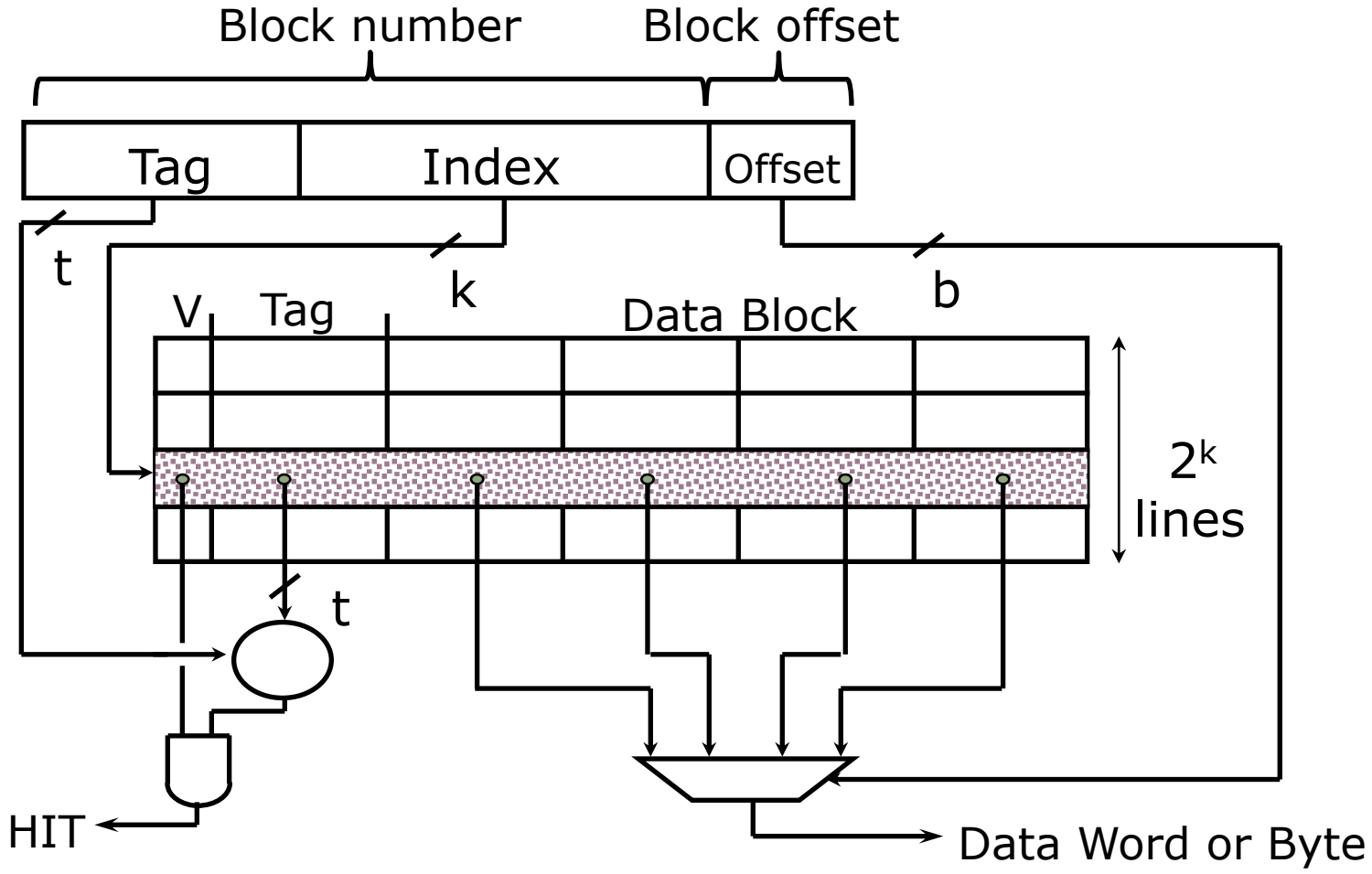
Direct-Mapped Cache



Direct-Mapped Cache

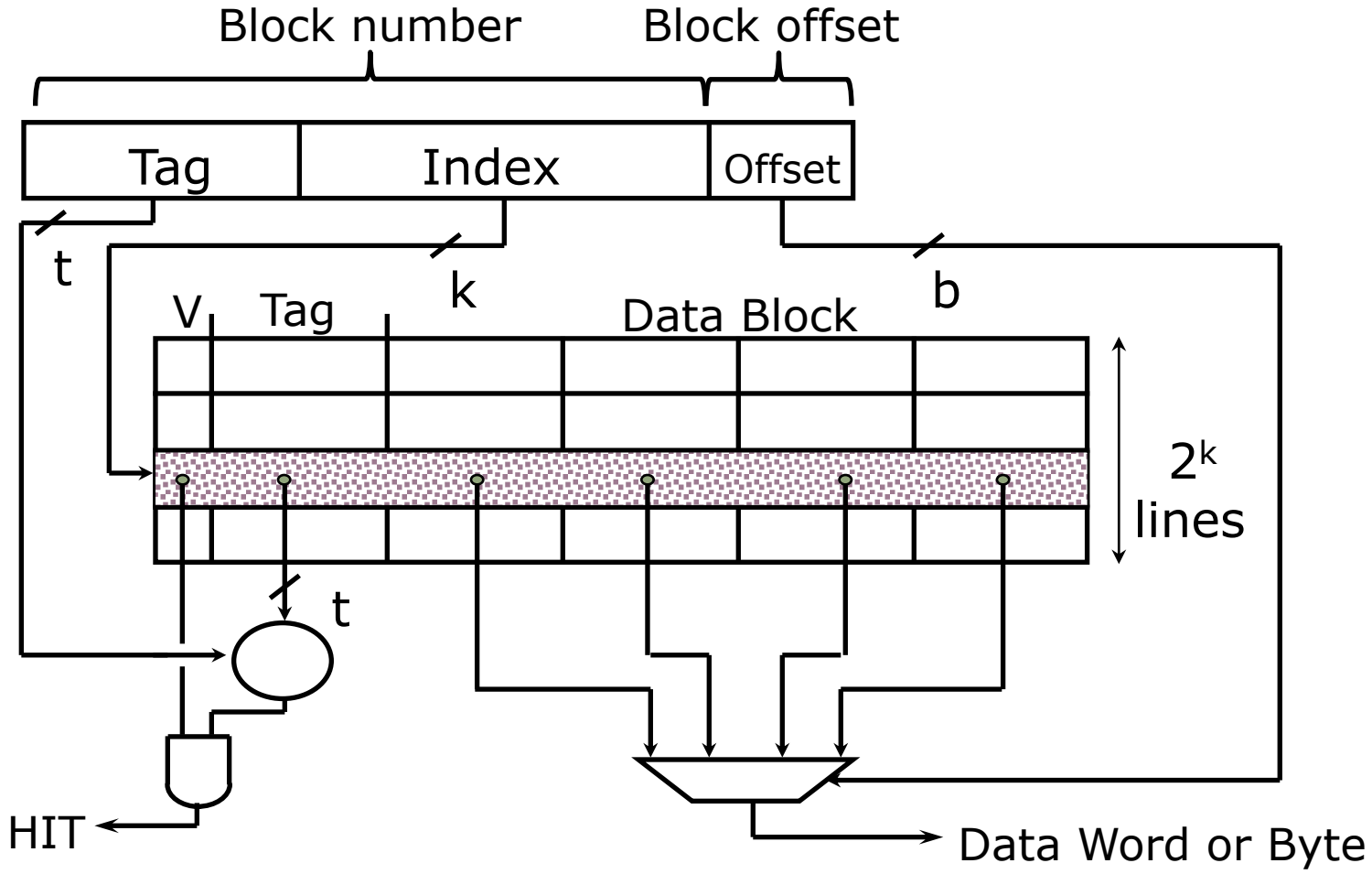


Direct-Mapped Cache



Q: What is a bad reference pattern?

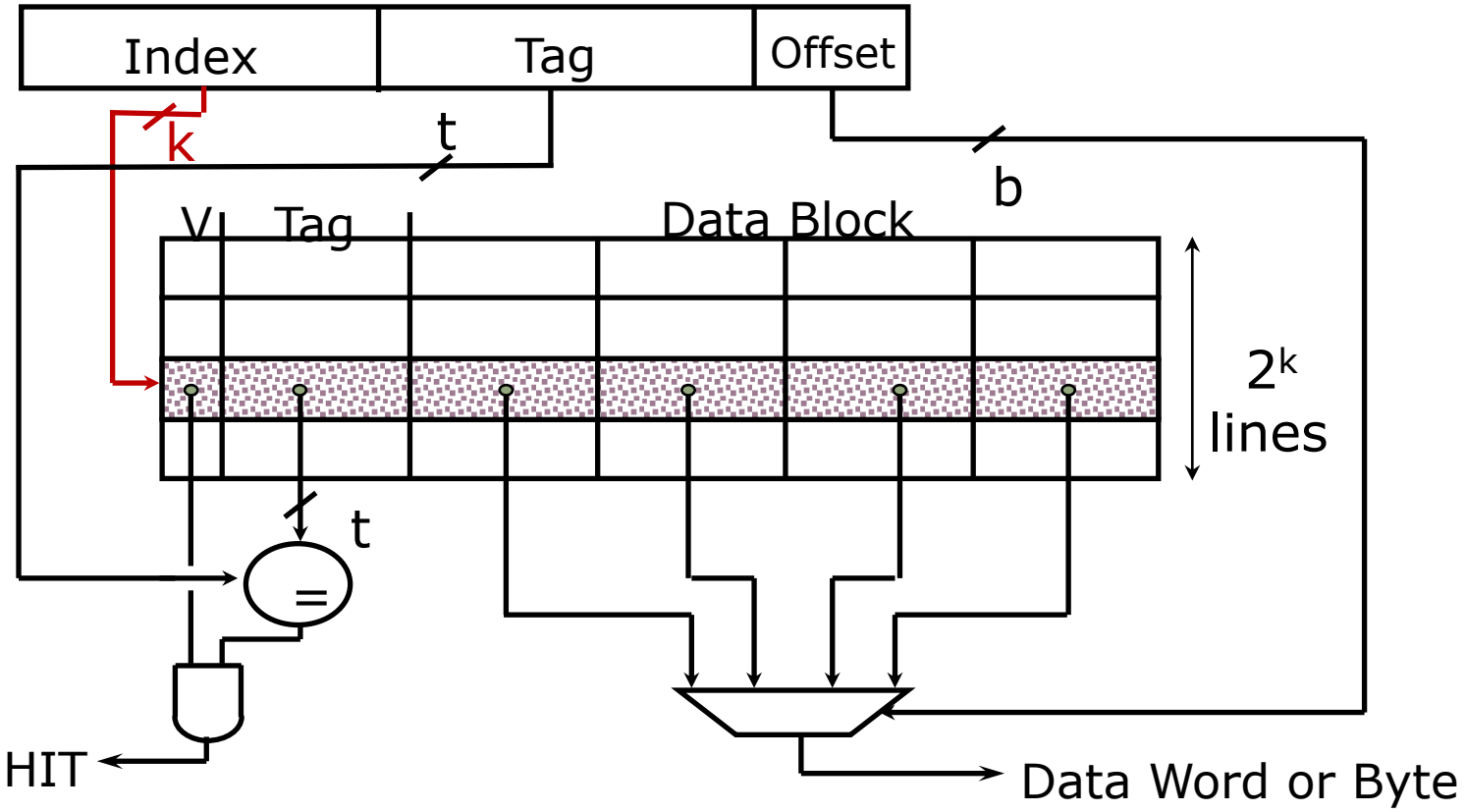
Direct-Mapped Cache



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

Direct Map Address Selection

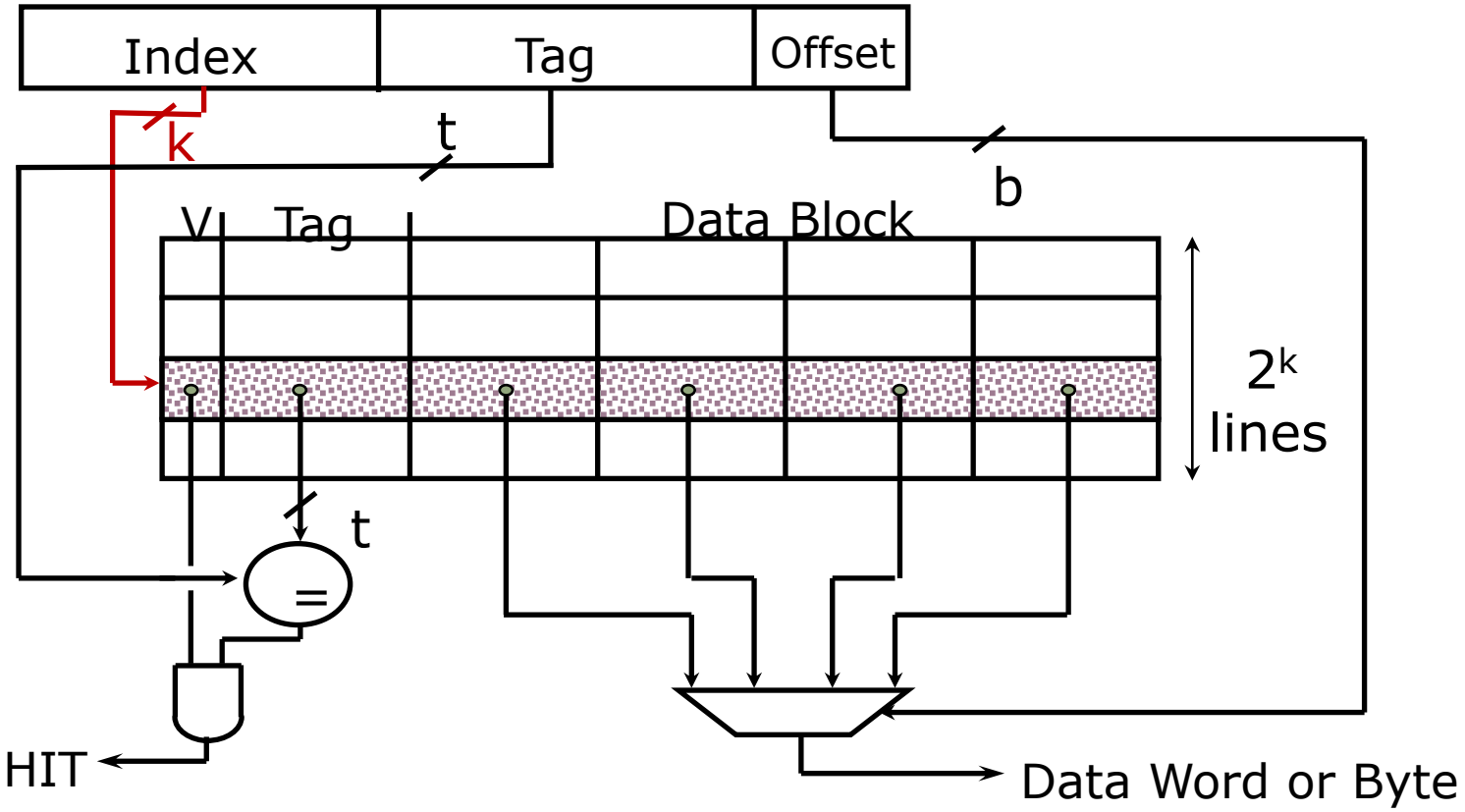
higher-order vs. lower-order address bits



Q: Why might this be undesirable? _____

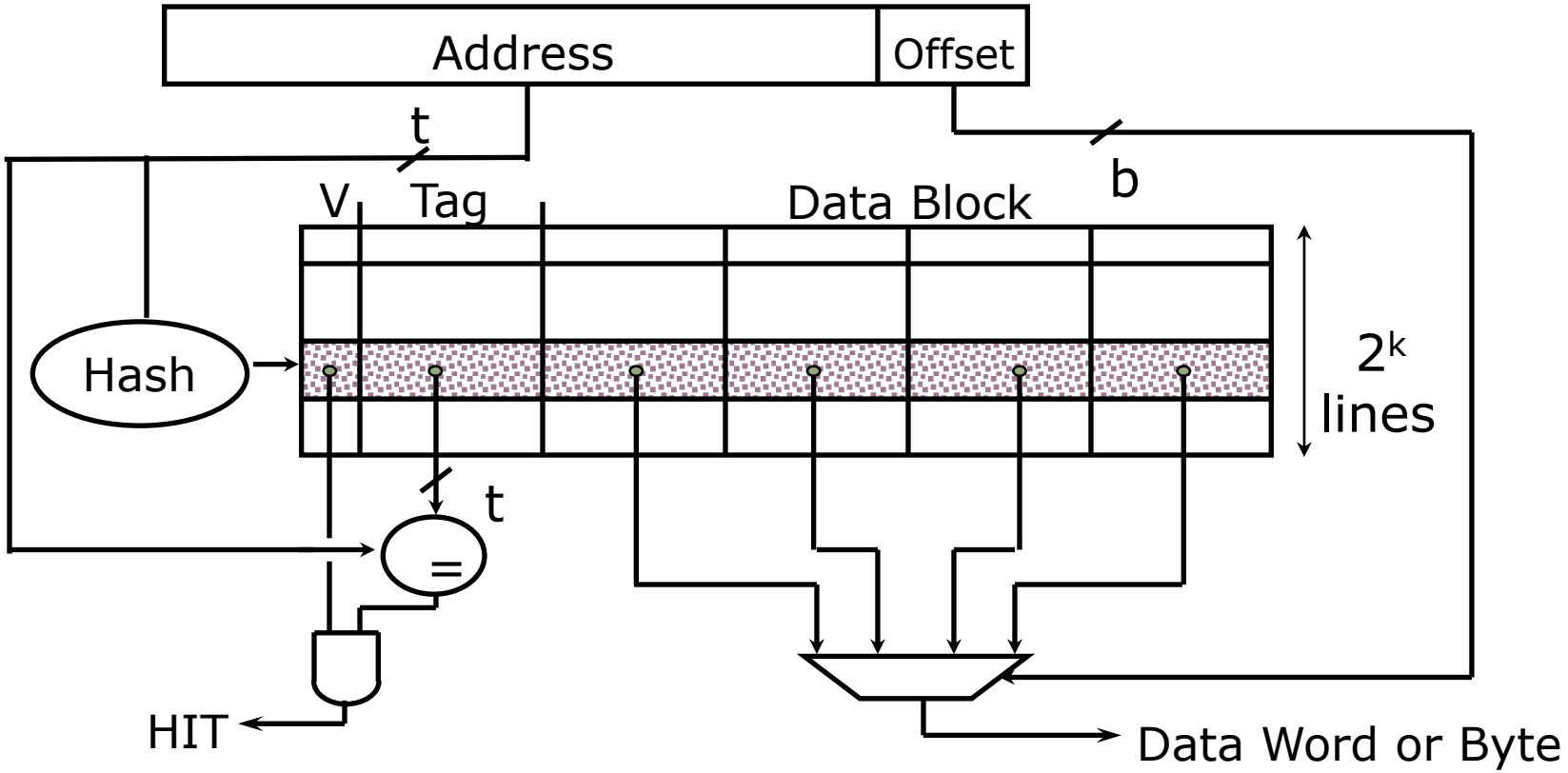
Direct Map Address Selection

higher-order vs. lower-order address bits



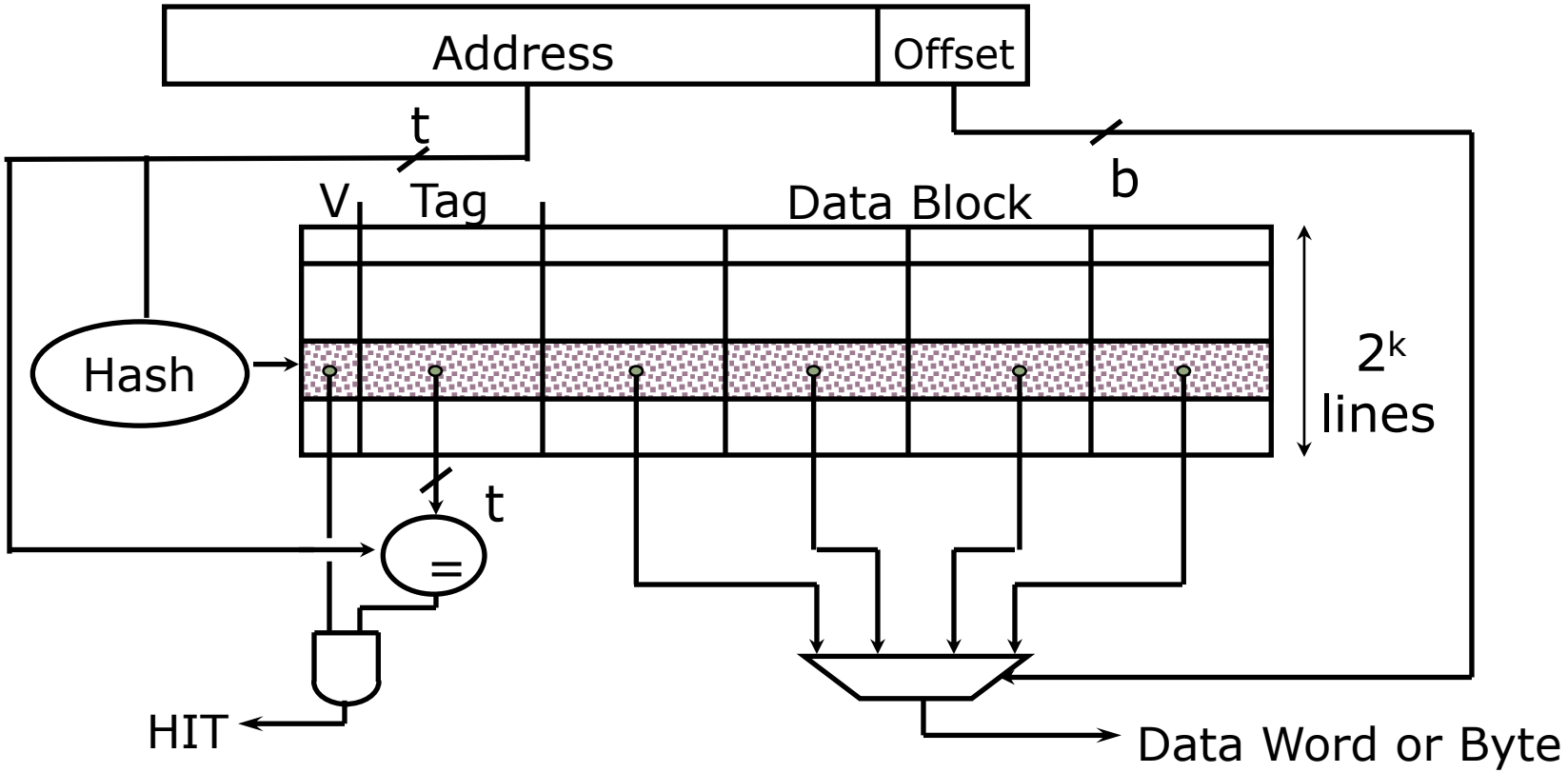
Q: Why might this be undesirable? Spatially local blocks conflict

Hashed Address Mapping



Q: What are the tradeoffs of hashing?

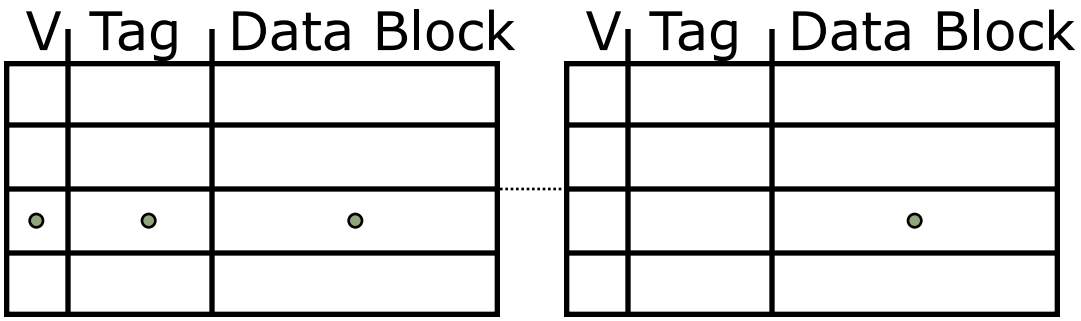
Hashed Address Mapping



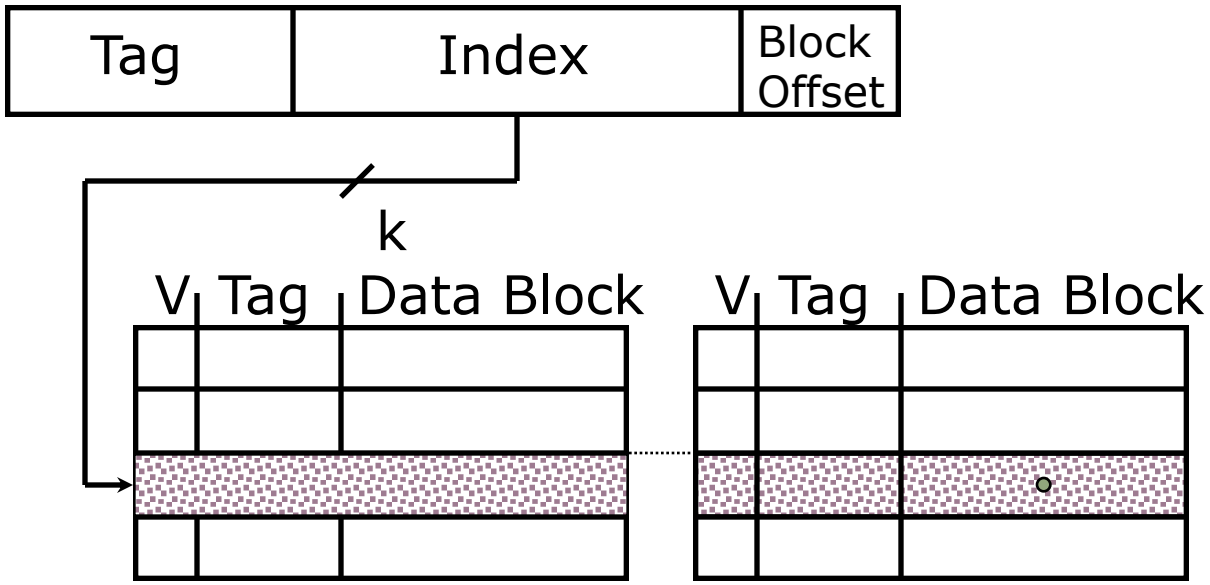
Q: What are the tradeoffs of hashing?

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

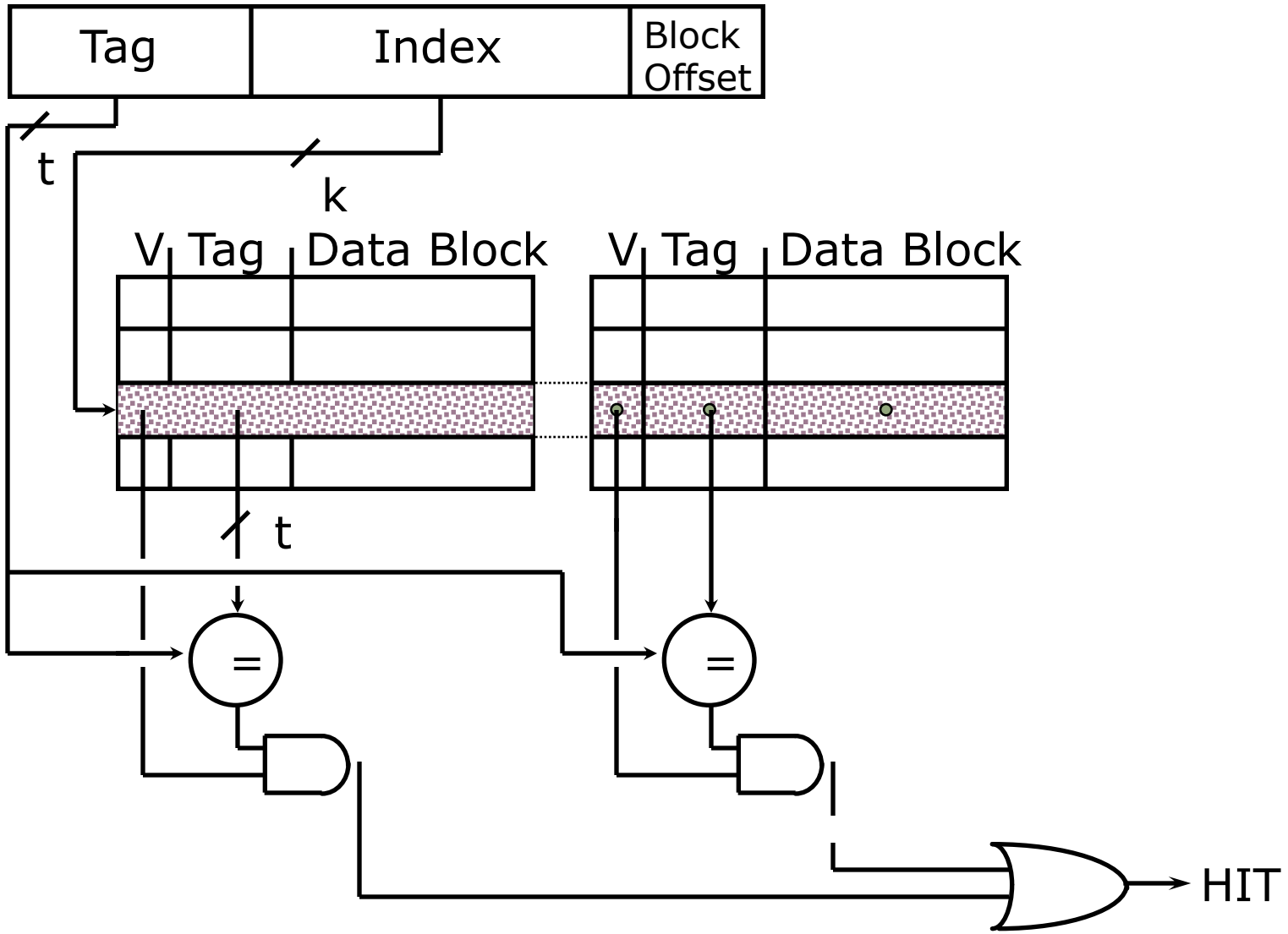
2-Way Set-Associative Cache



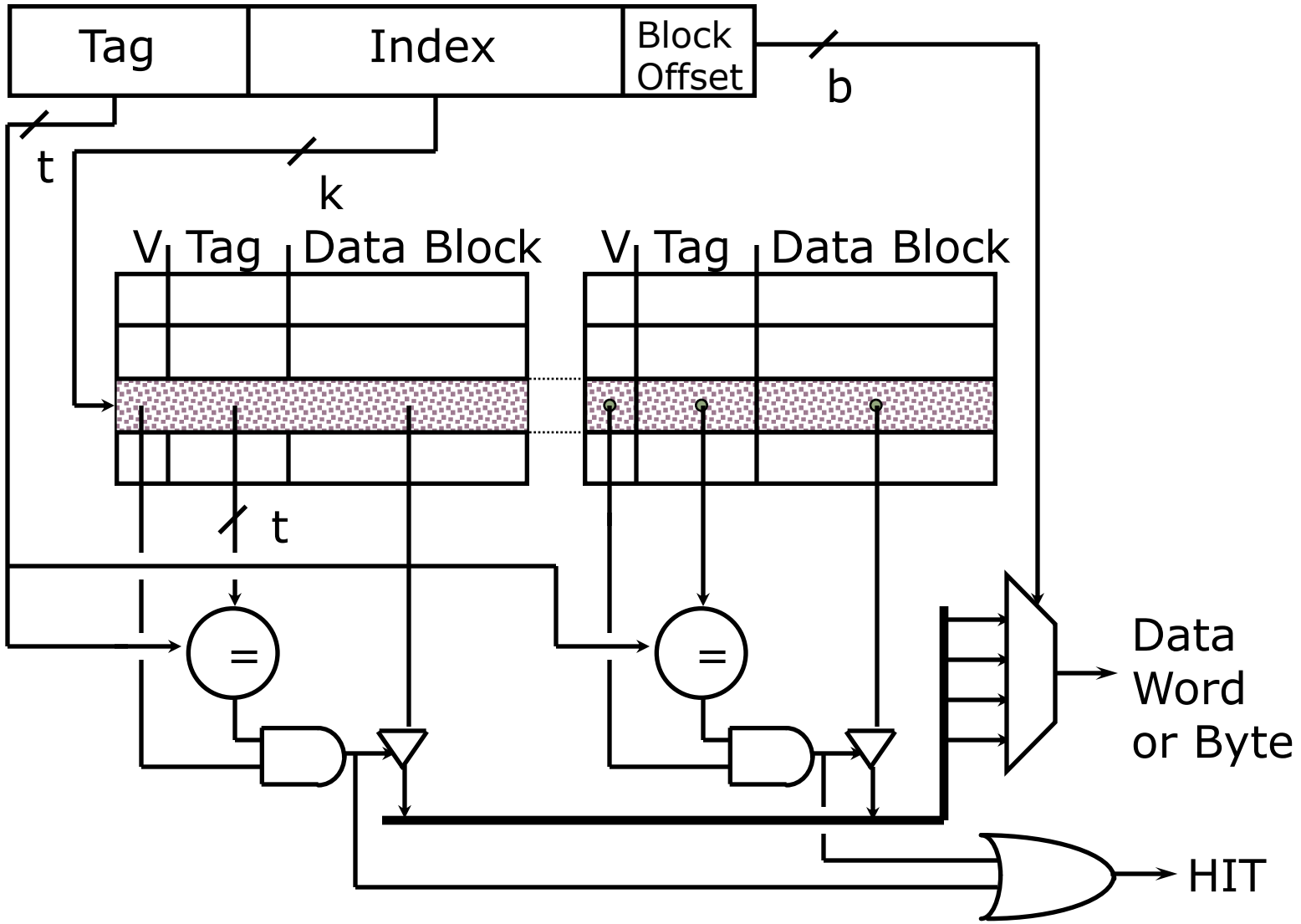
2-Way Set-Associative Cache



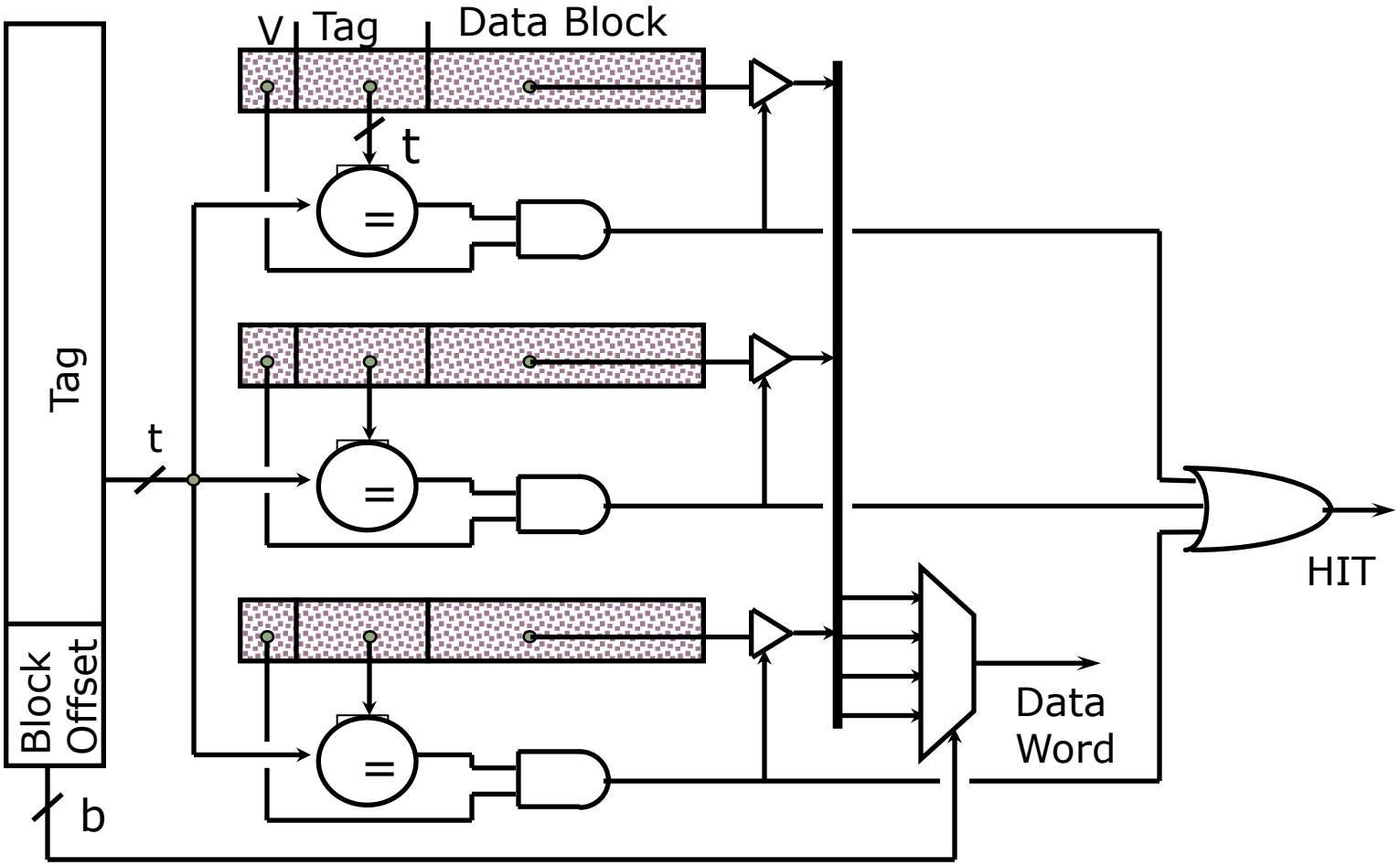
2-Way Set-Associative Cache



2-Way Set-Associative Cache

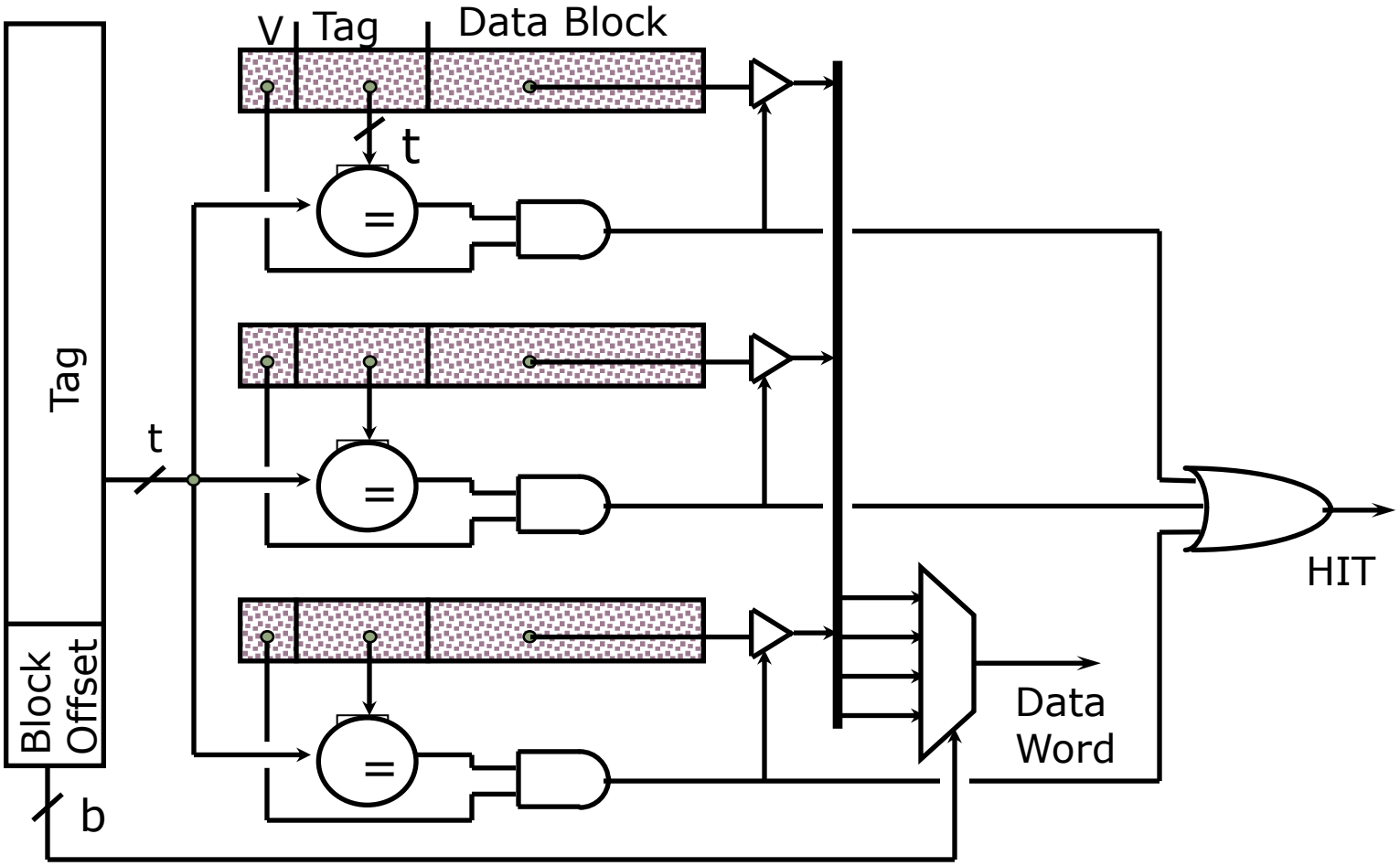


Fully Associative Cache



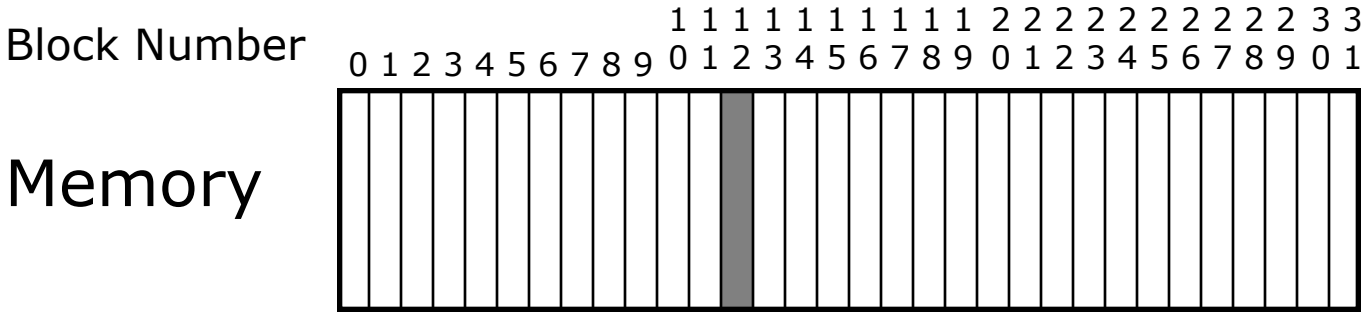
Q: Where are the index bits? _____

Fully Associative Cache



Q: Where are the index bits? Not needed

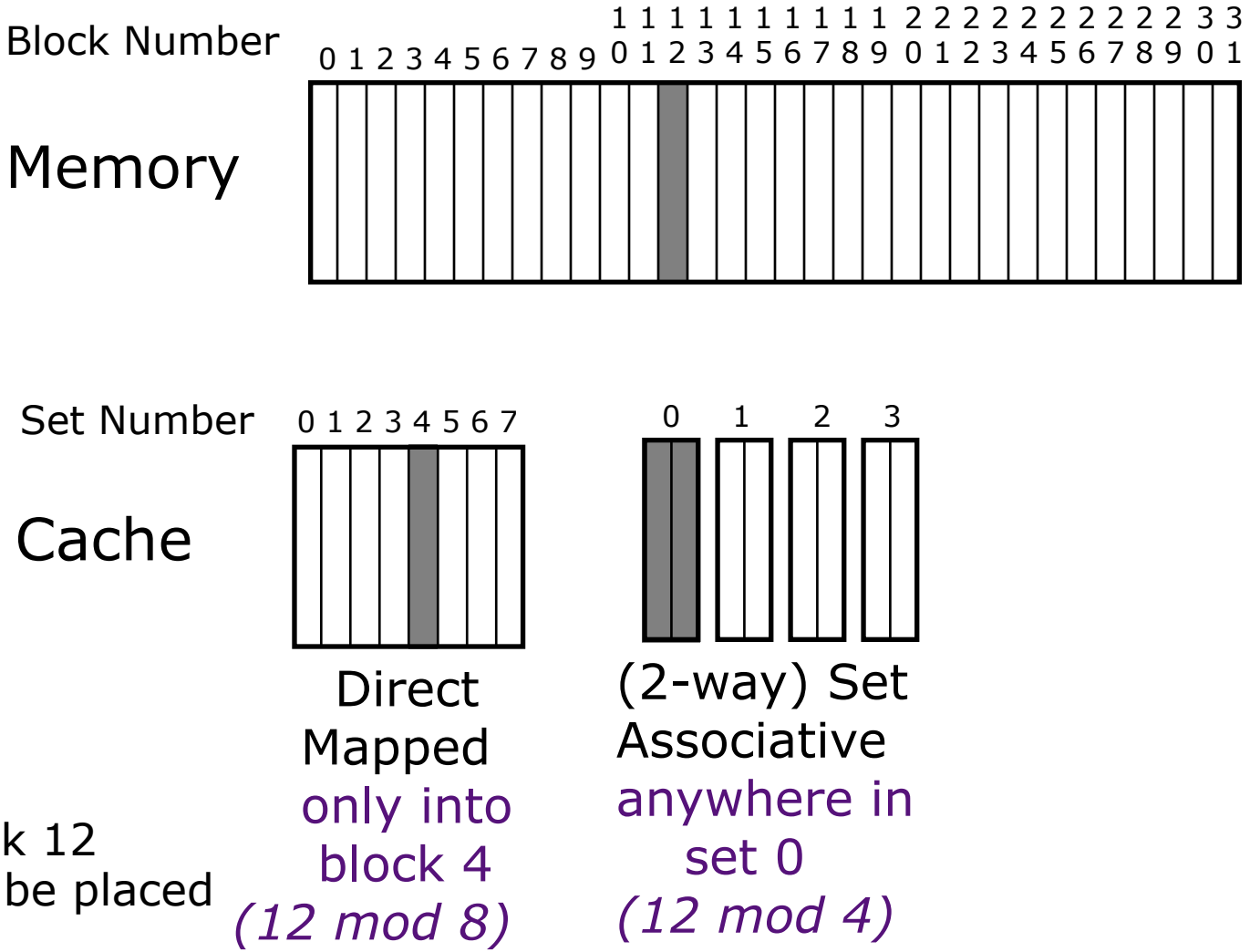
Placement Policy



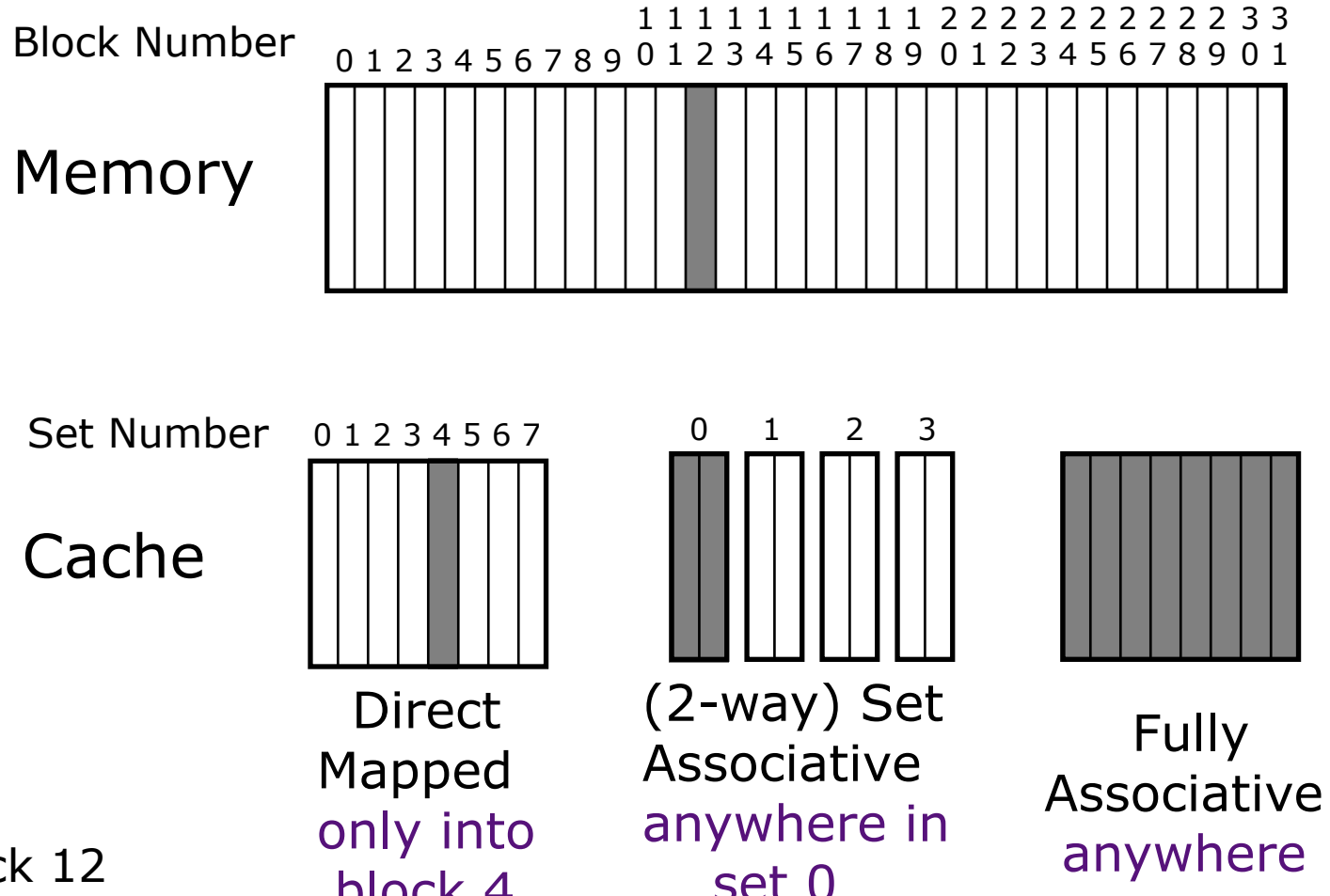
Direct Mapped only into block 4 *(12 mod 8)*

block 12 can be placed

Placement Policy



Placement Policy



Improving Cache Performance

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

Improving 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?

Improving 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 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
300
204

1		***1***		***1***		***1***	
1		***1***		***0***		***0***	
0		***1***		***0***		***1***	

Thank you!

Next lecture:
Virtual memory