
Memory

Memory types

Composition

Hierarchy

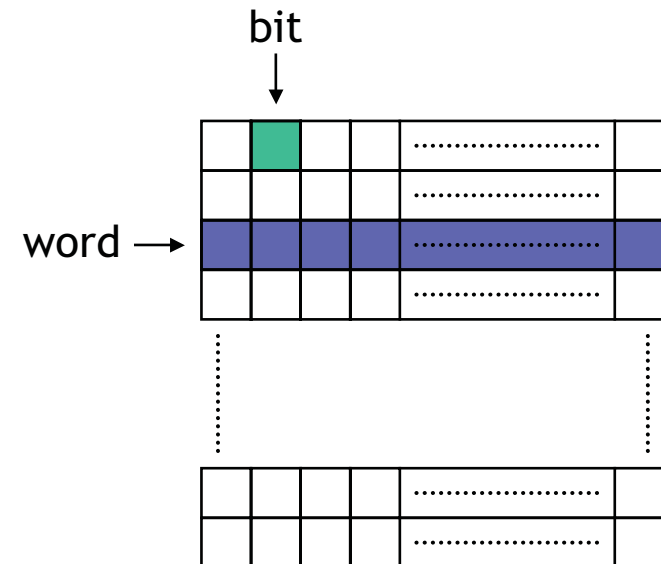
Caches

Introduction

- Embedded system's functionalities can be classified into processing, storage and communication
- Processing: Transformation of data
 - ▶ Processors
- Storage: Retention of data
 - ▶ Memories
- Communication: Transfer of data
 - ▶ Buses

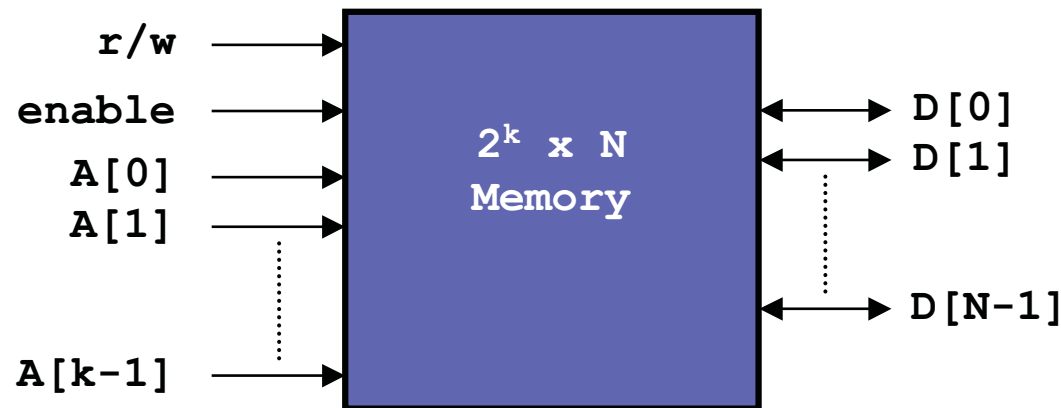
Basic concepts

- Stores large number of bits
- Memories are organized in words
- An $M \times N$ memory
 - ▶ $M=2^k$ words
 - k address input signals
 - ▶ N bits per word
- E.g.: 4096 x 8 memory:
 - ▶ $4,096 \times 8 = 32,768$ bits
 - ▶ $4,096 = 2^{12}$
 - 12 address input signals
 - ▶ 8 input/output data signals



Basic concepts

- Memory access signals
 - ▶ **r/w** Selects read or write
 - ▶ **enable** Read or write only when asserted
- Memory address/data
 - ▶ **A[0..k-1]** Address
 - ▶ **D[0..N-1]** Data



Memory classification

- Traditional ROM/RAM distinctions
 - ▶ **ROM**: Read only, bits stored without power
 - ▶ **RAM**: Read and write, lose stored bits without power
- Traditional distinctions blurred
 - ▶ Advanced ROMs can be written to
 - **PROM, EPROM, E²PROM, FLASH**
 - ▶ Advanced RAMs can hold bits without power
 - **NVRAM**
- Writing
 - ▶ Manner and speed a memory can be written
- Storage
 - ▶ Ability to hold stored bits once written

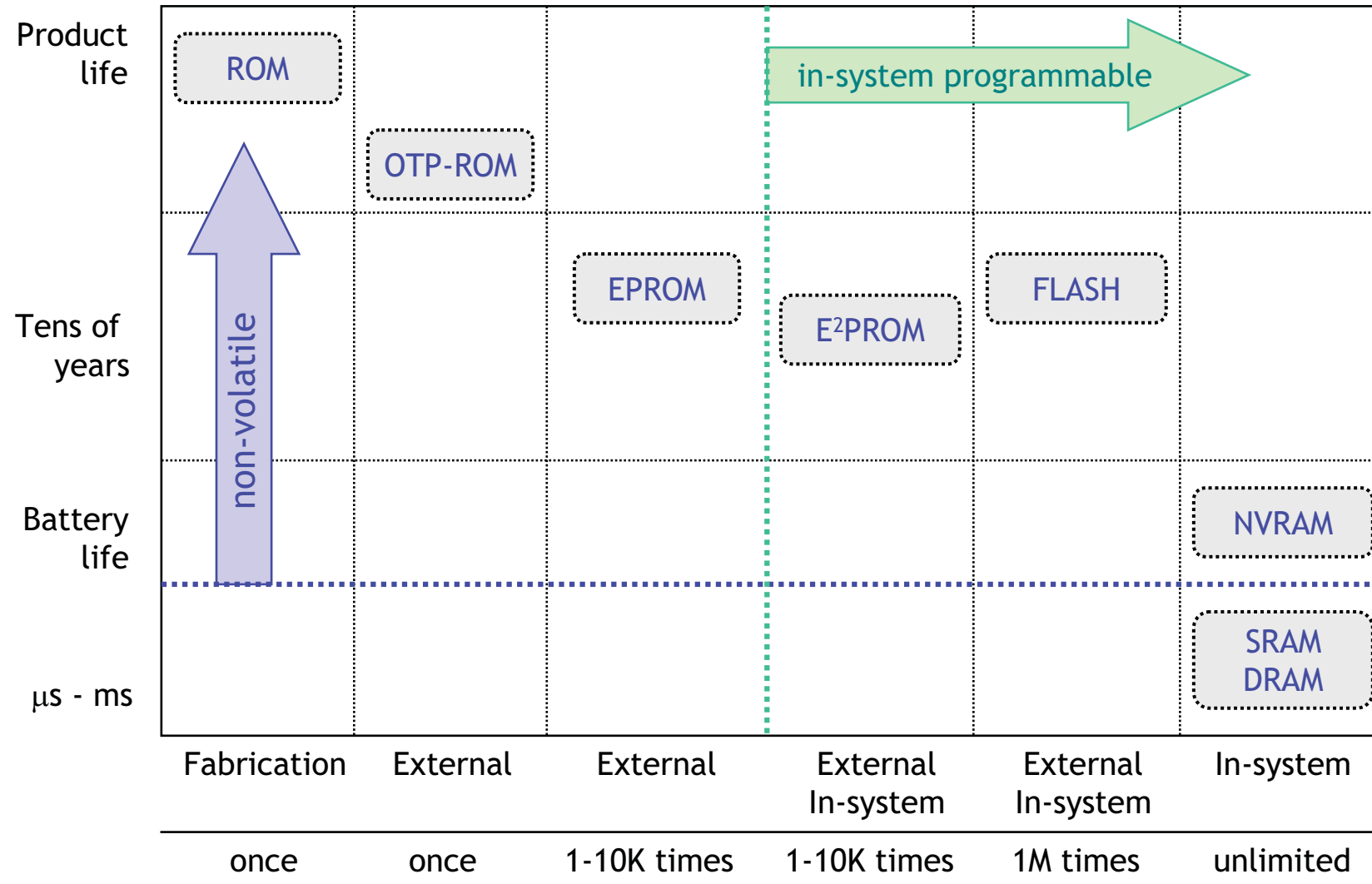
Memory classification: Writing

- High end
 - ▶ Processor writes to memory simply and quickly (RAM)
- Middle range
 - ▶ Processor writes to memory, but slower (FLASH, E²PROM)
- Lower range
 - ▶ A programmer writes to memory (EPROM, OTP ROM)
- Low end
 - ▶ Bits stored only during fabrication (ROM)

Memory classification: Storage

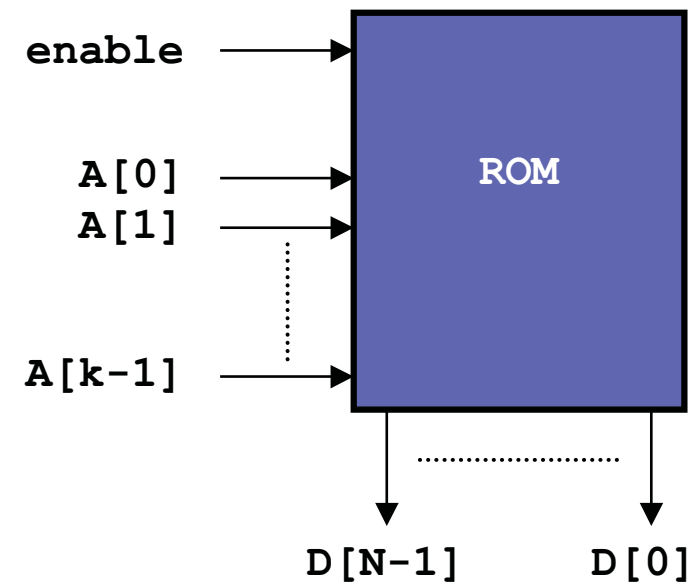
- High end
 - ▶ Essentially never loses bits (ROM)
- Middle range
 - ▶ Hold bits months or years after memory's power source turned off (NVRAM)
- Lower range
 - ▶ Hold bits as long as power supplied to memory (SRAM)
- Low end
 - ▶ Begins losing bits immediately after written (DRAM)

Memory classification

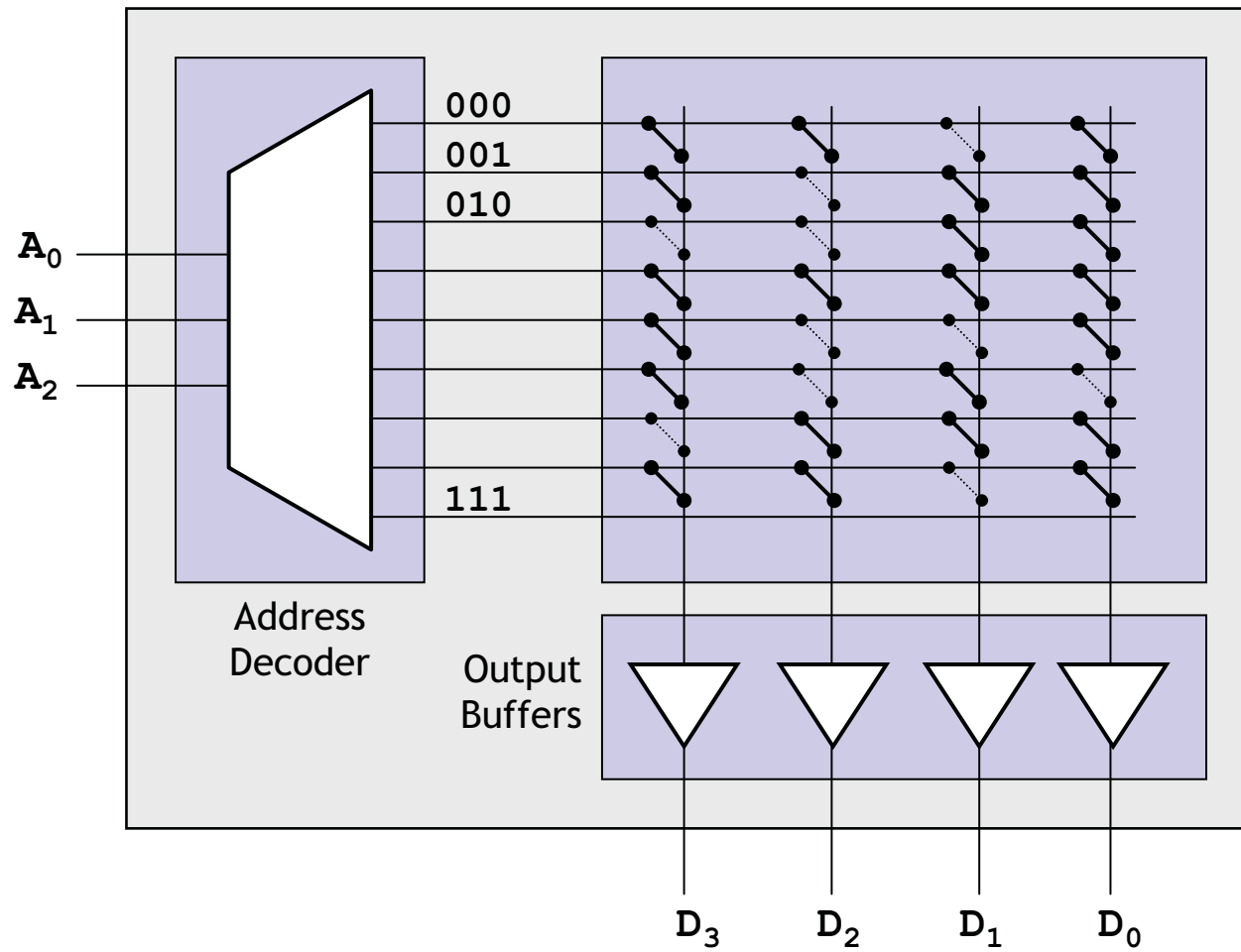


ROM

- Nonvolatile memory
 - ▶ Can be read from
 - ▶ Can not be written to
- Written, i.e. “programmed”
 - ▶ Before inserting to embedded system
- Uses
 - ▶ Store programs for GPP
 - ▶ Store constant data
 - ▶ Combinational circuits



ROM



OTP ROM

- Bits “programmed” by user after manufacture
 - ▶ User provides file of desired contents of ROM
 - File is input to machine called "ROM programmer"
 - ▶ Each programmable connection is a fuse
 - ROM programmer blows fuses where necessary
- Very slow writing
 - ▶ Typically written only once
- Very high storage permanence
 - ▶ Bits don't change unless re-programmer

EPROM

- Programmable component is a MOS transistor
 - ▶ Transistor has “floating” gate
 - ▶ Write
 - Large positive voltage at gate
 - ▶ Erase
 - Shining UV rays on surface of floating-gate
- Better writing
 - ▶ Can be erased and reprogrammed thousands of times
- Reduced storage permanence
 - ▶ Program lasts about 10 years
 - ▶ Susceptible to radiation and electric noise

E²PROM

- Programmed and erased electronically
 - ▶ Typically by using higher than normal voltage
 - ▶ Can program and erase individual words
- Better writing
 - ▶ In-system programmable
 - Needs a circuit to provide higher than normal voltage
 - ▶ Writes very slow due to erasing and programming
 - Special pin (busy) indicates when E²PROM still writing
 - ▶ Can be erased and programmed tens of thousands of times
- Similar storage permanence to EPROM
 - ▶ About 10 years

FLASH

- Extension of EEPROM
 - ▶ Same floating gate principle
 - ▶ Same write ability and storage permanence
- Fast erase
 - ▶ Large blocks of memory erased at once
 - Blocks typically several thousand bytes large
- Writes to single words may be slower
 - ▶ Read entire block
 - ▶ Update word
 - ▶ Write entire block again
- Very used in embedded systems

RAM

- Typically volatile memory
 - ▶ Bits are not held without power supply
- Read and written easily
 - ▶ In system
- Internal structure more complex than ROM
 - ▶ A word consists of cells, each cell storing 1 bit
 - ▶ Address lines enable a word
 - Connection in rows
 - ▶ Input/output data lines connected to each cell
 - Connection in columns
 - ▶ R/W control signal connected to every cell

SRAM vs. DRAM

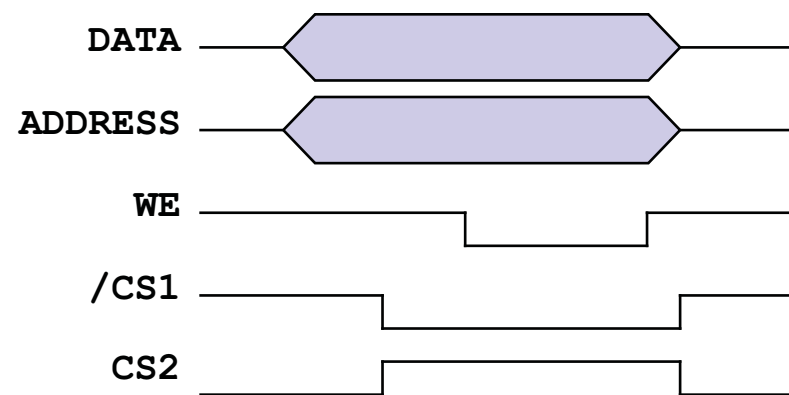
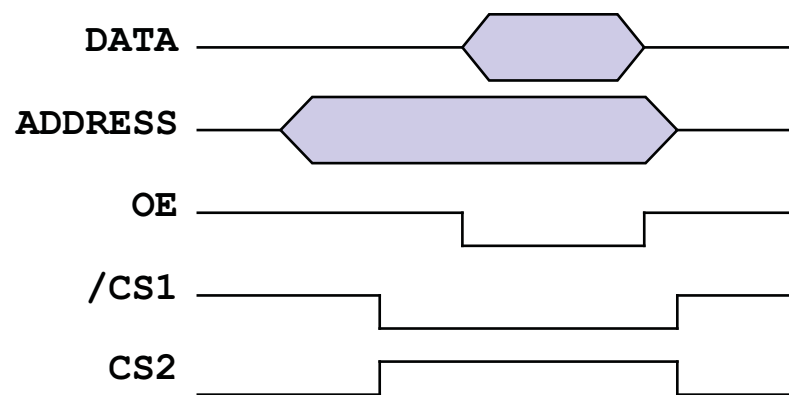
- SRAM: Static RAM
 - ▶ Memory cell uses flip-flop to store bit
 - Requires 6 transistors
 - ▶ Holds data as long as power supplied
- DRAM: Dynamic RAM
 - ▶ Memory cell uses MOS transistor and parasitic capacitor to store bit
 - More compact than SRAM
 - ▶ Refresh required due to capacitor leak
 - Word's cells refreshed when read
 - Typical refresh rate 15.625 μ s
 - ▶ Slower to access than SRAM

RAM variations

- PSRAM: Pseudo-static RAM
 - ▶ DRAM with built-in memory refresh controller
 - ▶ Popular low-cost high-density alternative to SRAM
- NVRAM: Nonvolatile RAM
 - ▶ Holds data after external power removed
 - ▶ Battery-backed RAM
 - SRAM with own permanently connected battery
 - ▶ SRAM with E²PROM or FLASH
 - Stores complete RAM contents on E²PROM or FLASH before power turned off

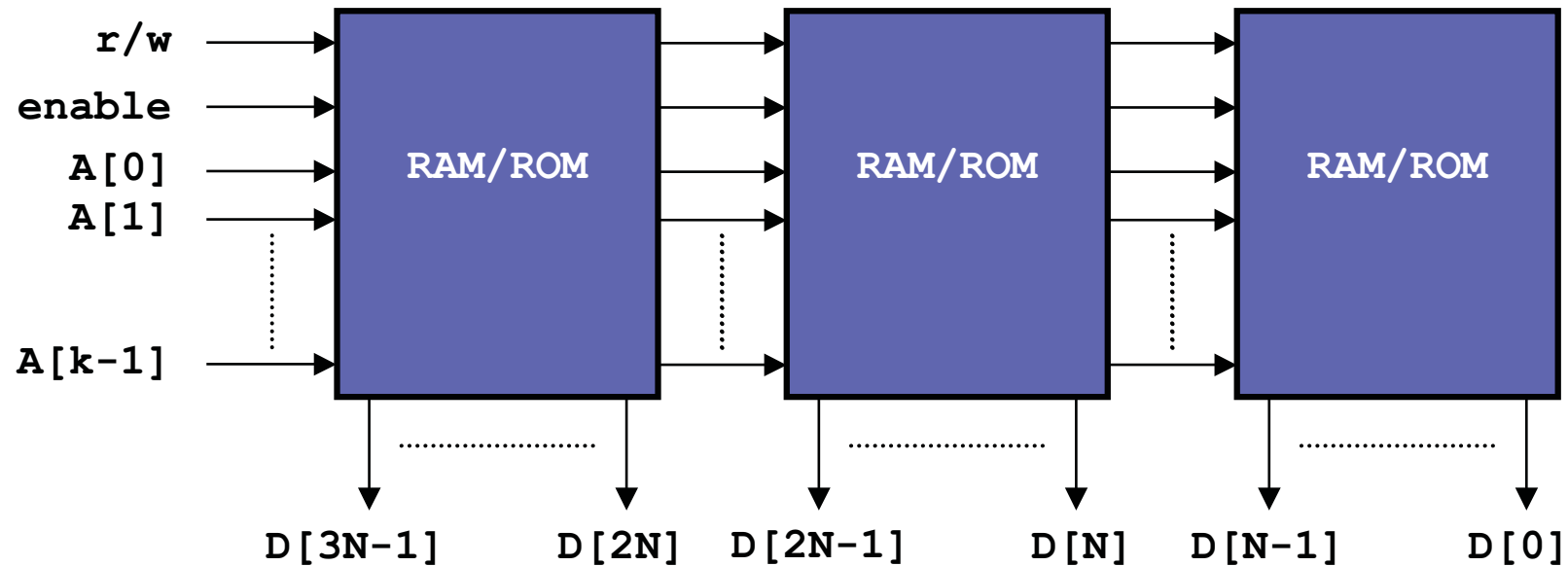
Example: HM6264 RAM/ROM

- Low-cost low-capacity memory devices
 - ▶ Used in 8-bit microcontroller-based embedded systems
- First two numeric digits indicate device type
 - ▶ RAM: 62 ROM: 27
- Subsequent digits indicate capacity in Kbits



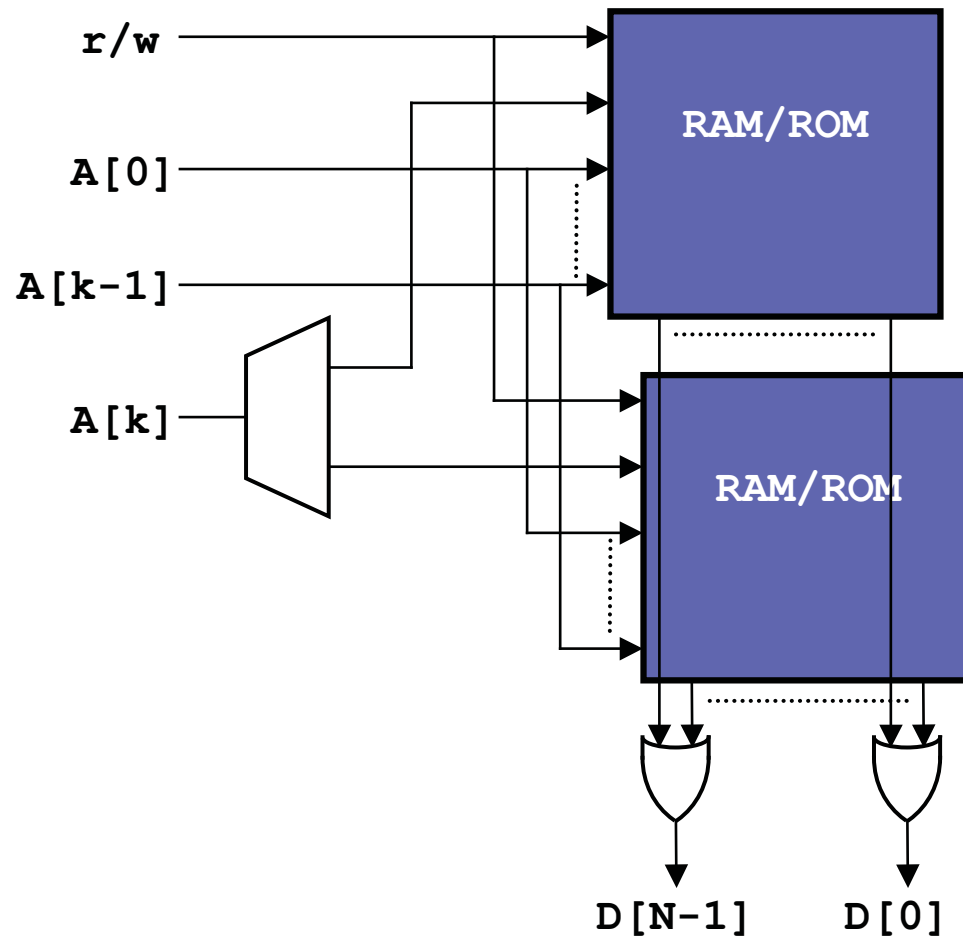
Composing memory

- Wider words (more bits per word)



Composing memory

- Wider addressing space (more words)



Il principio di località dei programmi

- Località spaziale
 - ▶ se l'istruzione di indirizzo i entra in esecuzione, con probabilità ≈ 1 anche l'istruzione di indirizzo $i + di$ entrerà in esecuzione (di è un intero piccolo)
 - Motivazione
 - ▶ di solito le istruzioni sono eseguite in sequenza
 - ▶ i salti sono relativamente rari o comunque spesso sono *polarizzati* verso un ramo
-

Il principio di località dei programmi

- Località temporale

- ▶ se un'istruzione entra in esecuzione al tempo t , con probabilità ≈ 1 la stessa istruzione sarà rieseguita al tempo $t + dt$ (dove dt è piccolo rispetto a t)

- Motivazione

- ▶ spesso le istruzioni rieseguite fanno parte di un ciclo, la cui presenza è giustificata solo se esso viene reiterato molte volte
 - ▶ se un'istruzione appartenente a un ciclo entra in esecuzione, è molto probabile che, entro il tempo di un'iterazione del ciclo, essa venga rieseguita
 - ▶ i cicli brevi generalmente sono molto più numerosi di quelli lunghi
-

Il principio di località dei programmi

- Località spaziale e temporale sono indipendenti
 - Se valgono entrambi, sono riassunti nel principio di località **spazio-temporale**
 - ▶ se l'istruzione di indirizzo i entra in esecuzione al tempo t , con probabilità ≈ 1 l'istruzione di indirizzo $i + di$ entrerà in esecuzione al tempo $t + dt$, dove di e dt sono piccoli rispetto a i e t , rispettivamente
 - La località spazio-temporale è statisticamente ben verificata dalla maggior parte dei programmi
 - ▶ la maggioranza delle istruzioni appartiene a cicli interni, con corpo sequenziale, brevi, iterati numerose volte e operanti su dati contigui
 - ▶ legge empirica: 90% del tempo è speso sul 10% codice
-

Interpretazione del principio di località

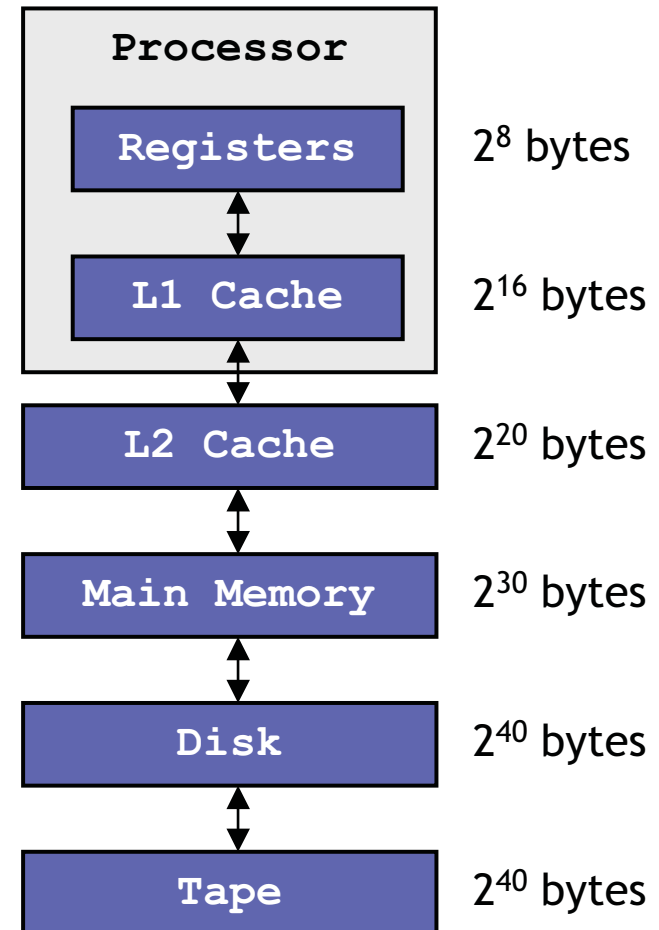
- Le istruzioni del programma si possono raggruppare in “blocchi” di istruzioni consecutive
 - ▶ se un’istruzione (qualsiasi) di un blocco entra in esecuzione, allora l’intero blocco di istruzioni verrà eseguito
 - ▶ se un blocco (qualsiasi) entra in esecuzione, allora entro breve tempo lo stesso blocco verrà rieseguito
 - I blocchi sono per esempio i “corpi” dei cicli più interni al programma
 - Anche i dati (oltre alle istruzioni) possono soddisfare al principio di località spazio-temporale
-

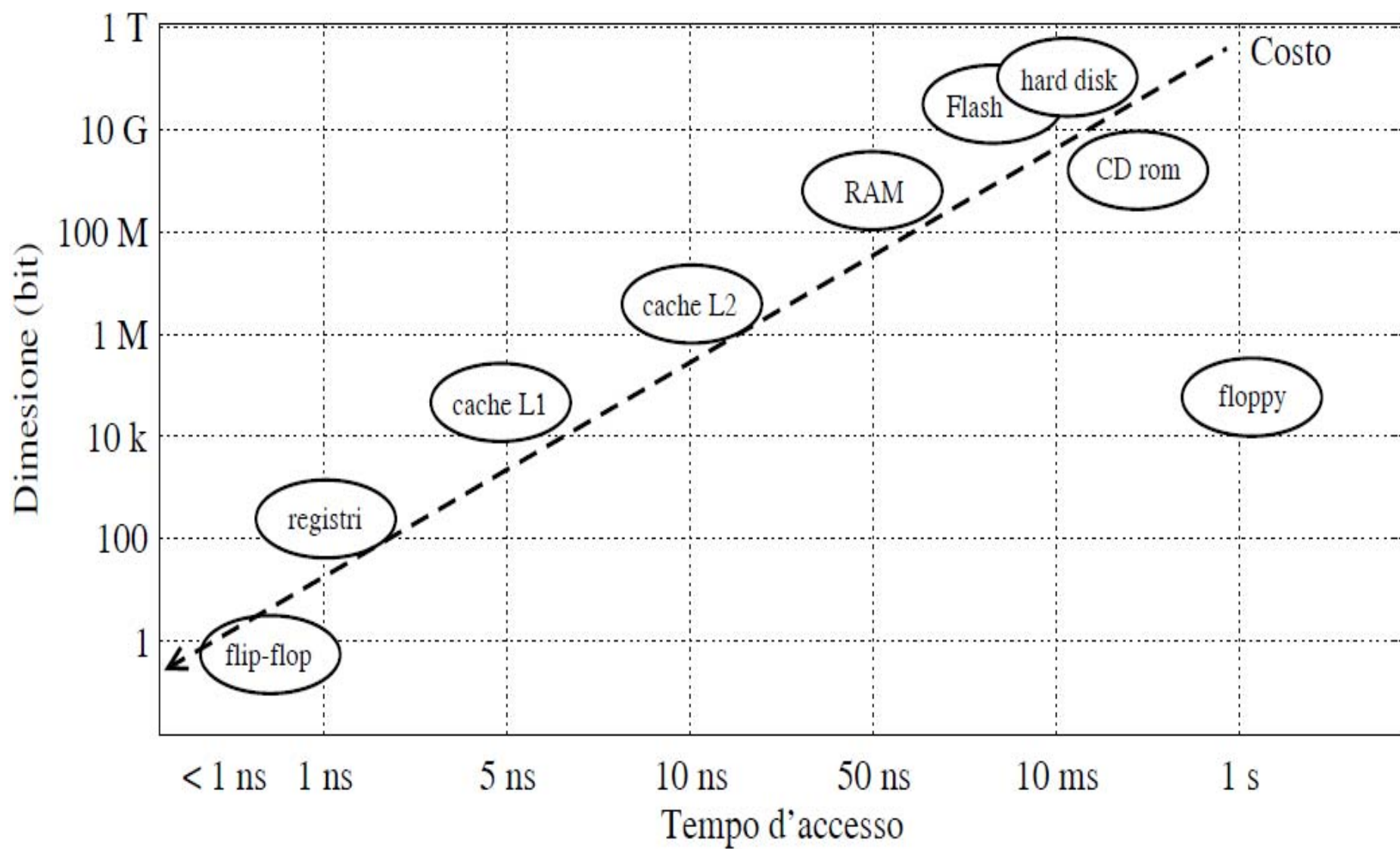
La gerarchia di memoria

- La memoria viene organizzata in livelli caratterizzati da velocità, dimensioni e costi diversi
 - I blocchi possono essere trasferiti da un livello inferiore a uno superiore
 - Cerco di tenere i *blocchi* di informazione usati più di frequente vicino alla CPU, per ottimizzare i tempi
 - Il dimensionamento del sistema e le politiche di gestione derivano da analisi statistico/quantitative delle applicazioni
 - L'obiettivo è fornire la sensazione di una memoria con la velocità del primo livello e la capacità del (dei) successivo(i)
-

Memory hierarchy

- Registers & L1 Cache
 - ▶ Small, very expensive, very fast
- L2 Cache
 - ▶ Larger, expensive, very fast
- Main memory
 - ▶ Large, inexpensive, slower
- Disk
 - ▶ Very large, inexpensive, slow
- Tape
 - ▶ Largest, inexpensive, sequential access





Cache memory

- Usually designed with SRAM
 - ▶ Faster but more expensive than DRAM
- Usually on same chip as processor
 - ▶ Space limited
 - ▶ Faster access (1 cycle vs. several cycles)
- Cache operation:
 - ▶ Request for main memory access (read/write)
 - ▶ First, check cache for copy
 - Cache hit: copy is in cache, quick access
 - Cache miss: copy not in cache, read address and possibly its neighbors into cache

Cache mapping

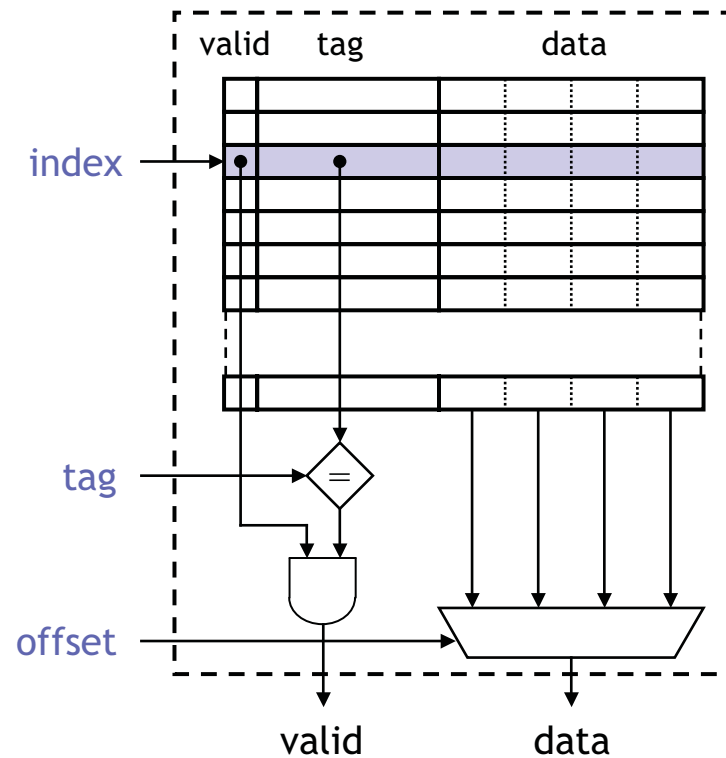
- Far fewer number of available cache addresses
- Cache mapping
 - ▶ Assigns main memory address to cache addresses
 - ▶ Determine hit or miss
- Three basic techniques:
 - ▶ Direct mapping
 - ▶ Fully associative mapping
 - ▶ Set-associative mapping
- Caches partitioned into blocks (lines) of adjacent memory addresses
 - ▶ Usually 4 or 8 addresses per line

Direct mapping

- Main memory address divided into 3 fields
 - ▶ Index
 - Cache address
 - Number of bits determined by cache size
 - ▶ Tag
 - Compared with tag stored in cache at the address indicated by the index
 - If tags match, check valid bit
 - ▶ Offset
 - Used to find particular word in cache line
- Valid bit
 - ▶ Indicates whether data has been loaded from memory

Direct mapping

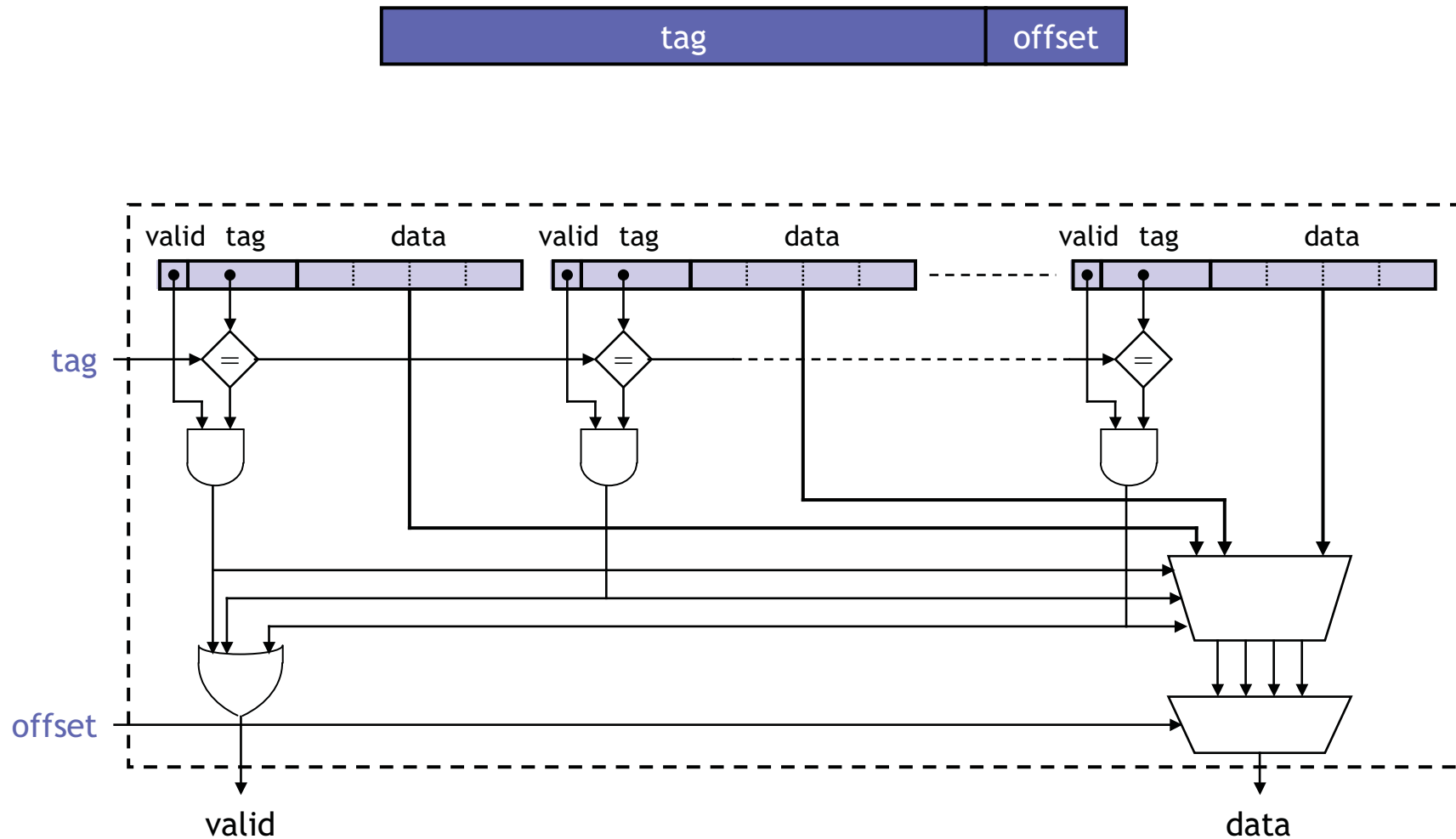
tag	index	offset
-----	-------	--------



Fully associative mapping

- Main memory address divided into 2 fields
 - ▶ Tag
 - Compared with all tags stored in cache simultaneously
 - If tags match, check valid bit
 - If valid bit is 1, the cache line is found
 - ▶ Offset
 - Used to find particular word in cache line
- Valid bit
 - ▶ Indicates whether data has been loaded from memory

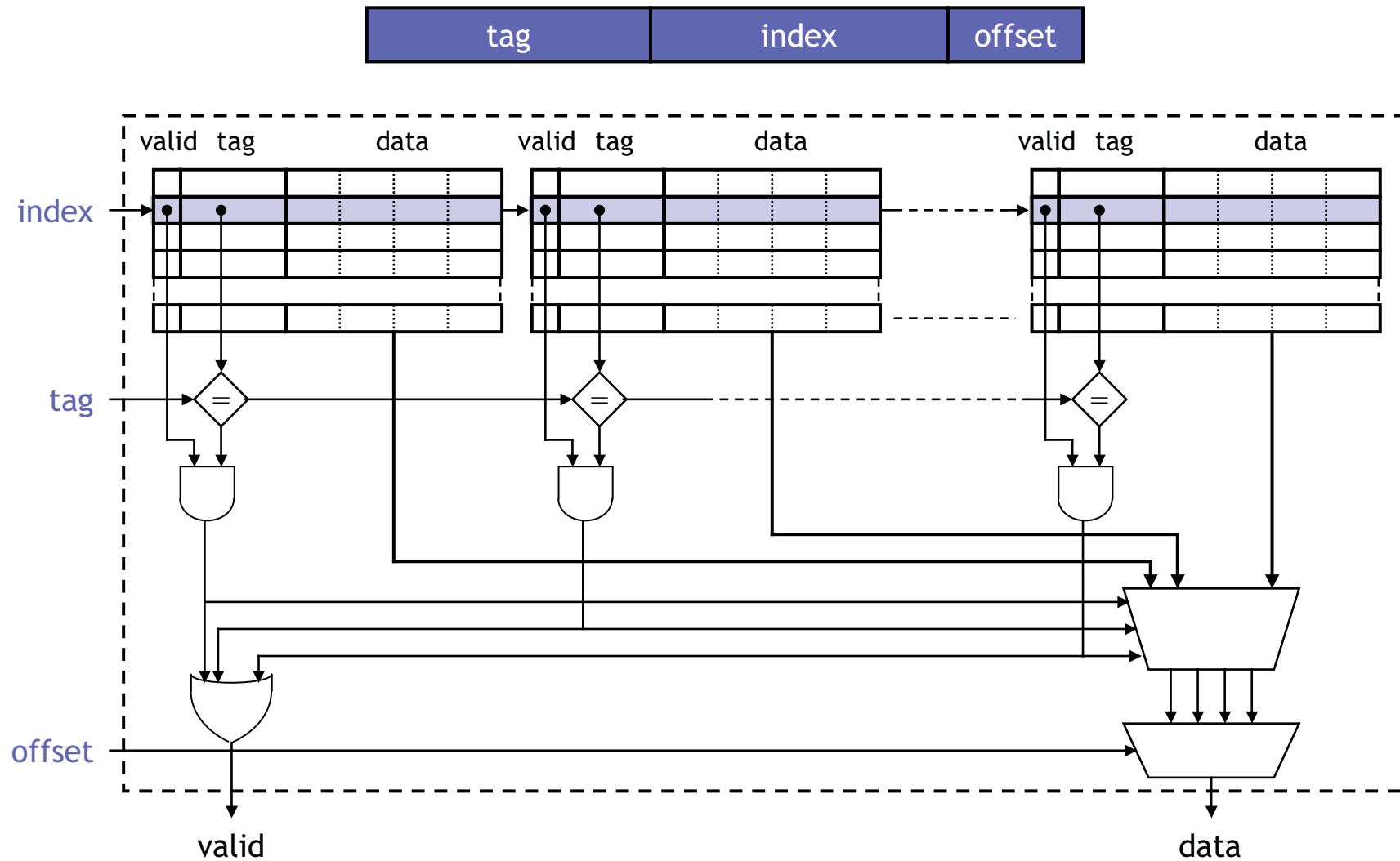
Fully associative mapping



Set-associative mapping

- Compromise between direct mapping and fully associative mapping
- Main memory address divided into 3 fields
 - ▶ Index, Offset and Tag
 - Same as in direct mapping
- But...
 - ▶ Each cache address contains data and tags of 2 or more memory address locations
 - ▶ Tags of a set are simultaneously compared
 - As in fully associative mapping
- Cache with set size N called N-way set-associative
 - ▶ 2-way, 4-way, 8-way are common

Set associative mapping



Replacement policy

- Technique for choosing which block to replace
 - ▶ When fully associative cache is full
 - ▶ When set-associative cache's line is full
 - ▶ Direct mapped cache has no choice
- Random
 - ▶ Replace block chosen at random
- LRU: Least-Recently Used
 - ▶ Replace block not accessed for longest time
- FIFO: First-In-First-Out
 - ▶ Push block onto queue when accessed
 - ▶ Choose block to replace by popping queue

Write techniques

- When writing data to cache, main memory must also be updated (i.e. written)
- Write-through
 - ▶ Memory written whenever cache is written
 - Easiest to implement
 - Processor must wait for slower main memory write
 - Potential unnecessary writes
- Write-back
 - ▶ Memory only written when “dirty” block replaced
 - An extra “dirty” bit for each block must be set when a cache block written to
 - Reduces number of slow main memory writes

Cache impact on performance

- Most important parameters in terms of performance:
 - ▶ Total size of cache
 - Total number of data bytes cache can hold
 - Tag, Valid and other house-keeping bits not included
 - ▶ Degree of associativity
 - ▶ Data block size
- Larger caches achieve lower miss rates but higher access cost

Cache impact on performance

- 2 Kbyte cache:

- ▶ Miss rate = 15%,
- ▶ Hit cost = 2 cycles
- ▶ Miss cost = 20 cycles
- ▶ Average cost of memory access:
 $[(1-0.15) \times 2] + (0.15 \times 20) = 4.7 \text{ cycles}$

$$t_{avg} = (1-h)*M + h*C$$

- 4 Kbyte cache:

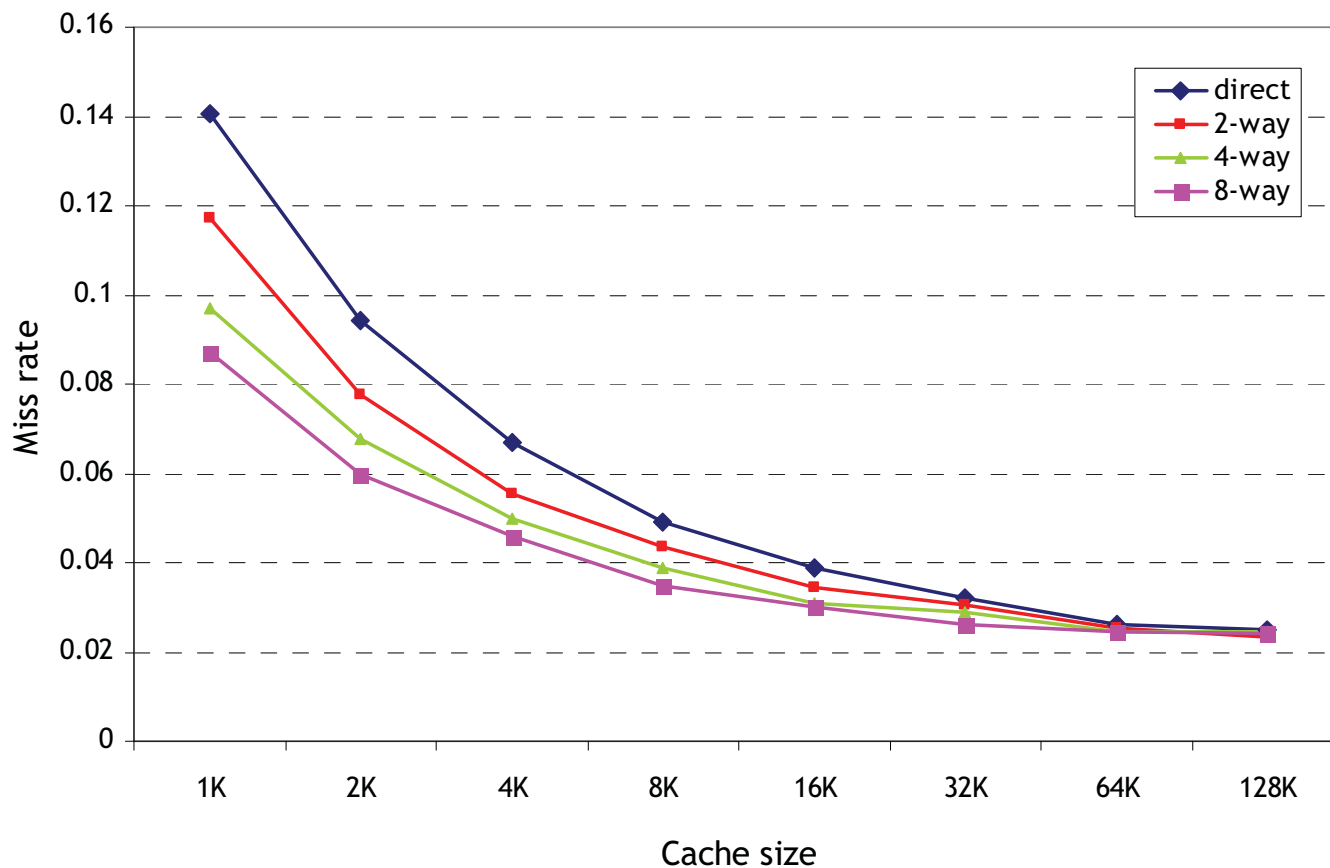
- ▶ Miss rate = 6.5%
- ▶ Hit cost = 3 cycles
- ▶ Miss cost = 20 cycles
- ▶ Average cost of memory access:
 $[(1-0.065) \times 3] + (0.065 \times 20) = 4.105 \text{ cycles}$

Cache impact on performance

- 8 Kbyte cache
 - ▶ Miss rate = 5.4%
 - ▶ Hit cost = 4 cycles
 - ▶ Miss cost = 20 cycles
 - ▶ Average cost of memory access:
$$[(1-0.054) \times 4] + (0.054 \times 20) = 4.864 \text{ cycles}$$
- Conclusion:
 - ▶ 4 Kbyte: best compromise
 - ▶ 2 Kbyte and 8 Kbyte are slower on average

Cache performance trade-offs

- Improving cache hit rate without increasing size
 - ▶ Increase line size or set-associativity



Caratteristica	AMD Opteron	ARM7D	Intel Xscale
CPU	CISC, 64 bit	RISC 32 bit	RISC 32 bit
Applicazione	high-end desktop	GPS, PDA, giochi	embedded
Organizzazione L1	separata	unificata	separata
Dimensione L1	64 KB, 64 KB	2 KB	32 KB, 32 KB
Associatività L1	2 vie	4 vie	32 vie
Sostituzione L1	LRU	LRU	round-robin
Scrittura L1	write-back	write-back	configurabile
Organizzazione L2	unificata	—	—
Dimensione L2	1 MB	—	—
Associatività L2	16 vie	—	—
Sostituzione L2	LRU approssimata	—	—
Scrittura L2	write-back	—	—