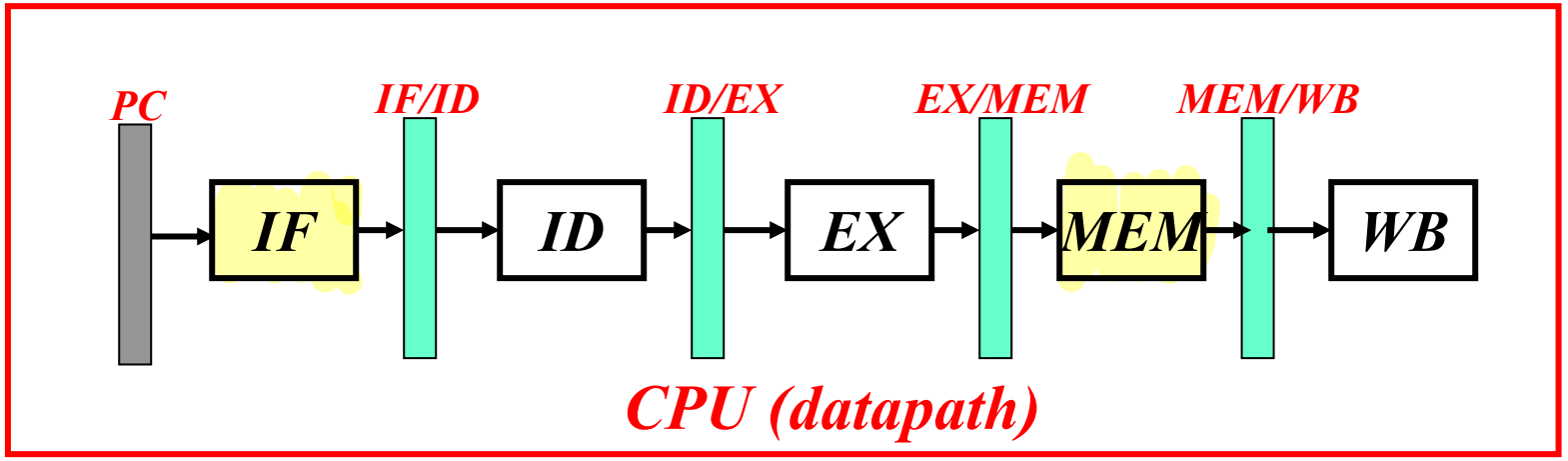
# Gerarchia di Memoria e Cache

Andrea Bartolini – a.bartolini@unibo.it

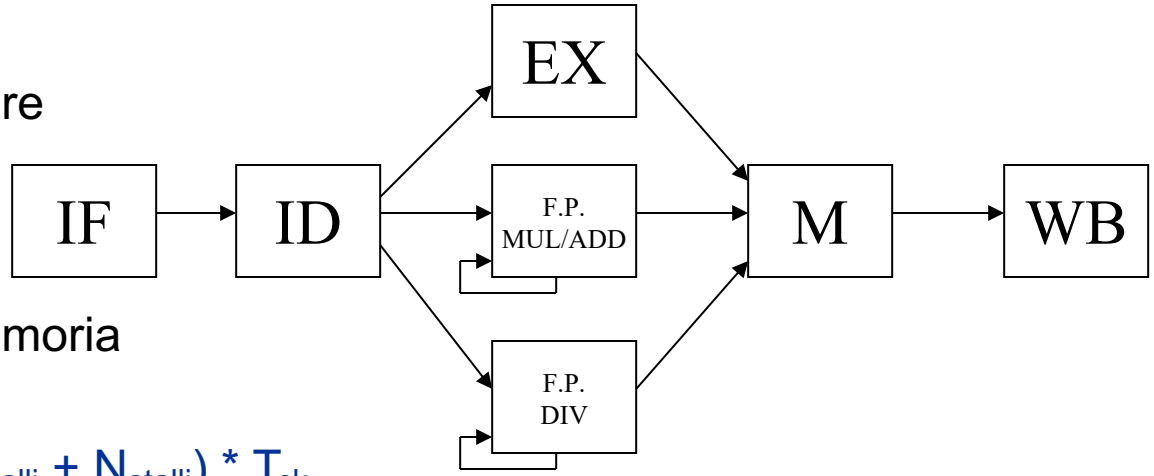
Derivate da slides Onur Mutlu, Tulio Salmon Cinotti

# Dov'è la memoria?

Abbiamo finora supposto che per accedere in memoria ci vuole solo un ciclo



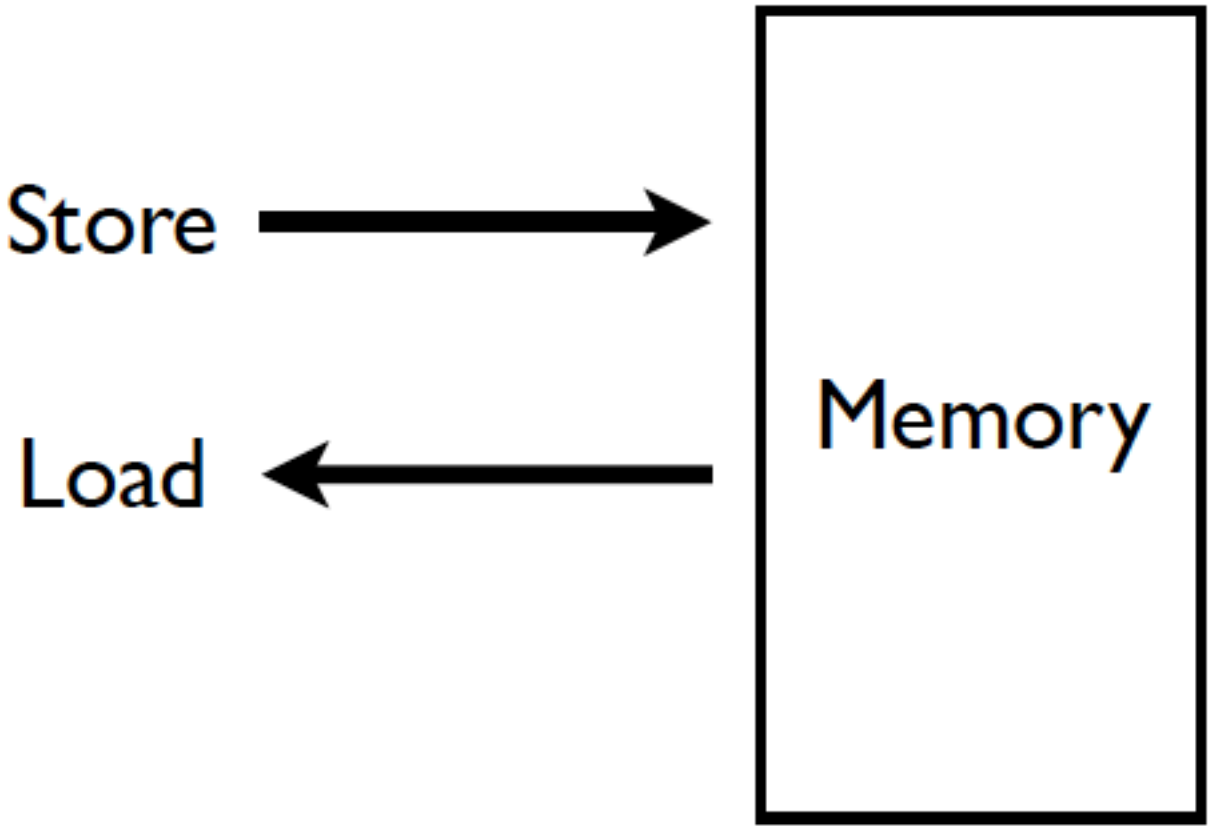

IF: legge ad ogni ciclo 4Bytes contenti l'istruzione da eseguire all'indirizzo PC




MEM: scrive e legge dalla memoria dati.

$CPU_{time} = (N_{istruzioni} * CPI_{senzastalli} + N_{stalli}) * T_{ck}$

# Memoria (Vista del programmatore)



# Ideal Memory

- Tempo di accesso nullo (latenza)
- Capacità infinita
- Costo nullo
- Larghezza di banda infinita (per supportare più accessi in parallelo)

LATENZA (ns): tempo che intercorre da quando metto un indirizzo/dato all'interno della memoria a quando quel dato è all'interno della pipeline.

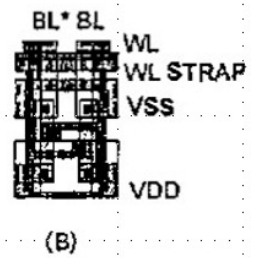
Da quando faccio richiesta alla memoria a quando viene servita.

BANDA: Quantità di byte che riesco a trasferire da e verso la memoria.

(dyga byte/s)

# Come immagazziniamo bits?

- Flip-Flops (o Latches) — Richiamo circuiti digitali
  - Accesso parallelo, molto veloce
  - Molto costosi (un bit costa decine di transistor)
- Static RAM – SRAM
  - Relativamente veloce, solo una parola di dati alla volta
  - Costoso (un bit costa 6 transistor) →




- Dynamic RAM - DRAM
  - Più lento, una parola di dati alla volta, la lettura distrugge il contenuto (refresh), ha bisogno di un processo di fabbricazione speciale
  - Economico (un bit costa solo un transistor più un condensatore) →


- Altre tecnologie di archiviazione (flash, disco rigido, nastro)
  - Molto più lento, l'accesso richiede molto tempo, non-volatile
  - Molto economico (nessun transistor direttamente coinvolto)

# The Problem

- Bigger is slower
  - SRAM, 512 Bytes, sub-nanosec
  - SRAM,  KByte~MByte, ~nanosec
  - DRAM, Gigabyte, ~50 nanosec
  - Hard Disk, Terabyte, ~10 millisec
- Faster is more expensive (dollars and chip area)
  - SRAM, < 10$ per Megabyte
  - DRAM, < 1$ per Megabyte
  - Hard Disk < 1$ per Gigabyte
  - These sample values (circa ~2011) scale with time
- Other technologies have their place as well
  - Flash memory (mature), PC-RAM, MRAM, RRAM (not mature yet)

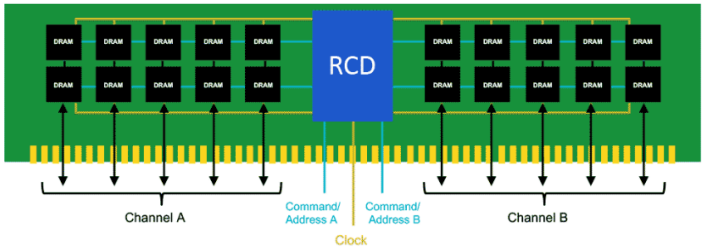
# Problema

- I requisiti della memoria ideale si contraddicono

- Più grande è più lento
  - Grande => più tempo per indirizzamento
- Più veloce è più costoso
  - Memory technology: SRAM vs. DRAM vs. Disk vs. Tape

- Una larghezza di banda più elevata è più costosa
  - Servono più banchi, più porte, maggiore frequenza o una tecnologia più veloce

# DDR4 & DDR5

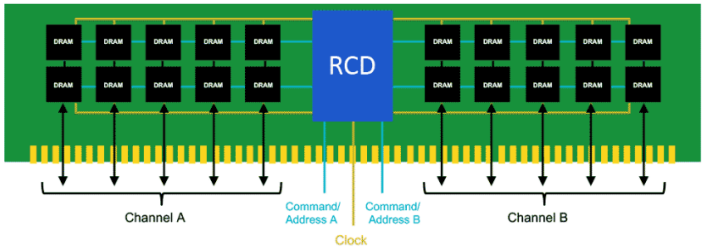



| Features | DDR4 | DDR5 | DDR5 Advantages |
|---|---|---|---|
| Speed | 1.6 to 3.2 GT/s<br>0.8 to 1.6 GHz clock | 4.8 to 8.4 GT/s<br>1.6 to 4.2 GHz clock | Higher bandwidth |
| IO Voltage | 1.2 V | 1.1 V | Lower power |
| Power Management | On motherboard | On DIMM PMIC | Better power efficiency<br>Better scalability |
| Channel Architecture | 72-bit data channel (64 data + 8 ECC)<br>1 channel per DIMM | 40-bit data channel (32 data + 8 ECC)<br>2 channels per DIMM | Higher memory efficiency<br>Lower latency |
| Burst Length | BC4, BL8 | BC8, BL16 | Higher memory efficiency |
| Max. Die Density | 16Gb | 64Gb | Higher capacity DIMMs |
| More Intelligence | SPD ($I^2C$) | SPD Hub & Temperature Sensors ($I^3C$) | Enhanced system management<br>Greater telemetry for thermal management |

Latency 50-70ns

frequenza = frequenza del clock

# DDR4 & DDR5



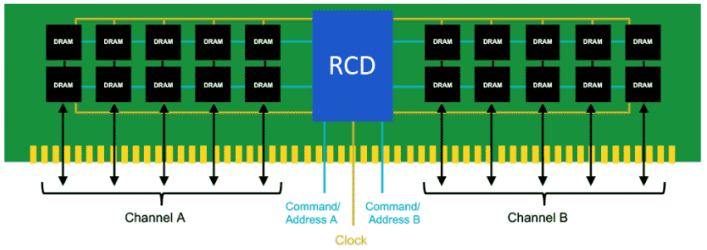

Latency 50-70ns

| Features | DDR4 | DDR5 | DDR5 Advantages |
|---|---|---|---|
| Speed | 1.6 to 3.2 GT/s<br>0.8 to 1.6 GHz clock | 4.8 to 8.4 GT/s<br>1.6 to 4.2 GHz clock | Higher bandwidth |
| IO Voltage | 1.2 V | 1.1 V | Lower power |
| Power Management | On motherboard | On DIMM PMIC | Better power efficiency<br>Better scalability |
| Channel Architecture | 72-bit data channel (64 data + 8 ECC)<br>1 channel per DIMM | 40-bit data channel (32 data + 8 ECC)<br>2 channels per DIMM | Higher memory efficiency<br>Lower latency |
| Burst Length | BC4, BL8 | BC8, BL16 | Higher memory efficiency |
| Max. Die Density | 16Gb | 64Gb | Higher capacity DIMMs |
| More Intelligence | SPD ($I^2C$) | SPD Hub & Temperature Sensors ($I^3C$) | Enhanced system management<br>Greater telemetry for thermal management |

→ 64b data width & 1ch x DIMM → Peak MT/s

DDR4 DIMM : SKU CMW**16G**X4M2D**3200**C18

# DDR4 & DDR5




Latency 50-70ns

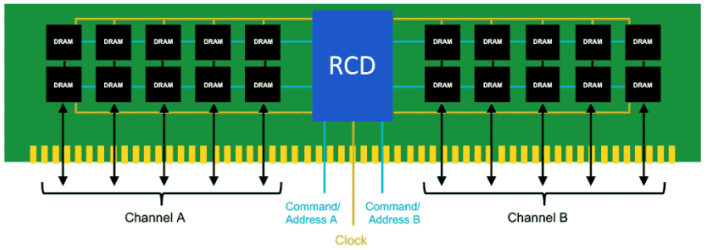
| Features | DDR4 | DDR5 | DDR5 Advantages |
|---|---|---|---|
| Speed | 1.6 to 3.2 GT/s<br>0.8 to 1.6 GHz clock | 4.8 to 8.4 GT/s<br>1.6 to 4.2 GHz clock | Higher bandwidth |
| IO Voltage | 1.2 V | 1.1 V | Lower power |
| Power Management | On motherboard | On DIMM PMIC | Better power efficiency<br>Better scalability |
| Channel Architecture | 72-bit data channel (64 data + 8 ECC)<br>1 channel per DIMM | 40-bit data channel (32 data + 8 ECC)<br>2 channels per DIMM | Higher memory efficiency<br>Lower latency |
| Burst Length | BC4, BL8 | BC8, BL16 | Higher memory efficiency |
| Max. Die Density | 16Gb | 64Gb | Higher capacity DIMMs |
| More Intelligence | SPD ($I^2C$) | SPD Hub & Temperature Sensors ($I^3C$) | Enhanced system management<br>Greater telemetry for thermal management |

→ 64b data width & 1ch x DIMM → Peak MT/s

DDR4 DIMM : SKU CMW**16G**X4M2D**3200**C18

=> Bandwidth = 1ch * 3200 MT/s * 64b/T = 3.2 GT/s * 8B/T = 25.6GB/s

# DDR4 & DDR5




Latency 50-70ns

| Features | DDR4 | DDR5 | DDR5 Advantages |
|---|---|---|---|
| Speed | 1.6 to 3.2 GT/s<br>0.8 to 1.6 GHz clock | 4.8 to 8.4 GT/s<br>1.6 to 4.2 GHz clock | Higher bandwidth |
| IO Voltage | 1.2 V | 1.1 V | Lower power |
| Power Management | On motherboard | On DIMM PMIC | Better power efficiency<br>Better scalability |
| Channel Architecture | 72-bit data channel (64 data + 8 ECC)<br>1 channel per DIMM | 40-bit data channel (32 data + 8 ECC)<br>2 channels per DIMM | Higher memory efficiency<br>Lower latency |
| Burst Length | BC4, BL8 | BC8, BL16 | Higher memory efficiency |
| Max. Die Density | 16Gb | 64Gb | Higher capacity DIMMs |
| More Intelligence | SPD ($I^2C$) | SPD Hub & Temperature Sensors ($I^3C$) | Enhanced system management<br>Greater telemetry for thermal management |

→ 64b data width & 1ch x DIMM → Peak MT/s

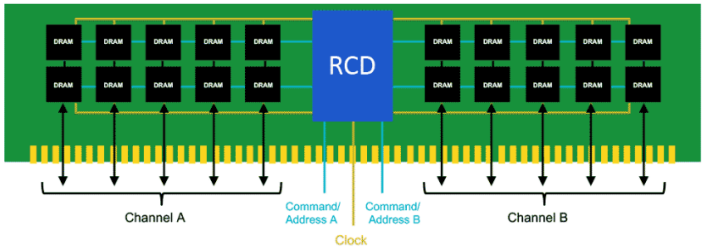
DDR4 DIMM : SKU CMW**16G**X4M2D**3200**C18

=> Bandwidth = 1ch * 3200 MT/s * 64b/T = 3.2 GT/s * 8B/T = 25.6GB/s

→ 32b data width & 2ch x DIMM → Peak MT/s

DDR5 DIMM : SKU CMK**32G**X5M2B**5200**C40

# DDR4 & DDR5




Latency 50-70ns

| Features | DDR4 | DDR5 | DDR5 Advantages |
|---|---|---|---|
| Speed | 1.6 to 3.2 GT/s<br>0.8 to 1.6 GHz clock | 4.8 to 8.4 GT/s<br>1.6 to 4.2 GHz clock | Higher bandwidth |
| IO Voltage | 1.2 V | 1.1 V | Lower power |
| Power Management | On motherboard | On DIMM PMIC | Better power efficiency<br>Better scalability |
| Channel Architecture | 72-bit data channel (64 data + 8 ECC)<br>1 channel per DIMM | 40-bit data channel (32 data + 8 ECC)<br>2 channels per DIMM | Higher memory efficiency<br>Lower latency |
| Burst Length | BC4, BL8 | BC8, BL16 | Higher memory efficiency |
| Max. Die Density | 16Gb | 64Gb | Higher capacity DIMMs |
| More Intelligence | SPD ($I^2C$) | SPD Hub & Temperature Sensors ($I^3C$) | Enhanced system management<br>Greater telemetry for thermal management |

→ 64b data width & 1ch x DIMM → Peak MT/s

DDR4 DIMM : SKU CMW**16G**X4M2D**3200**C18

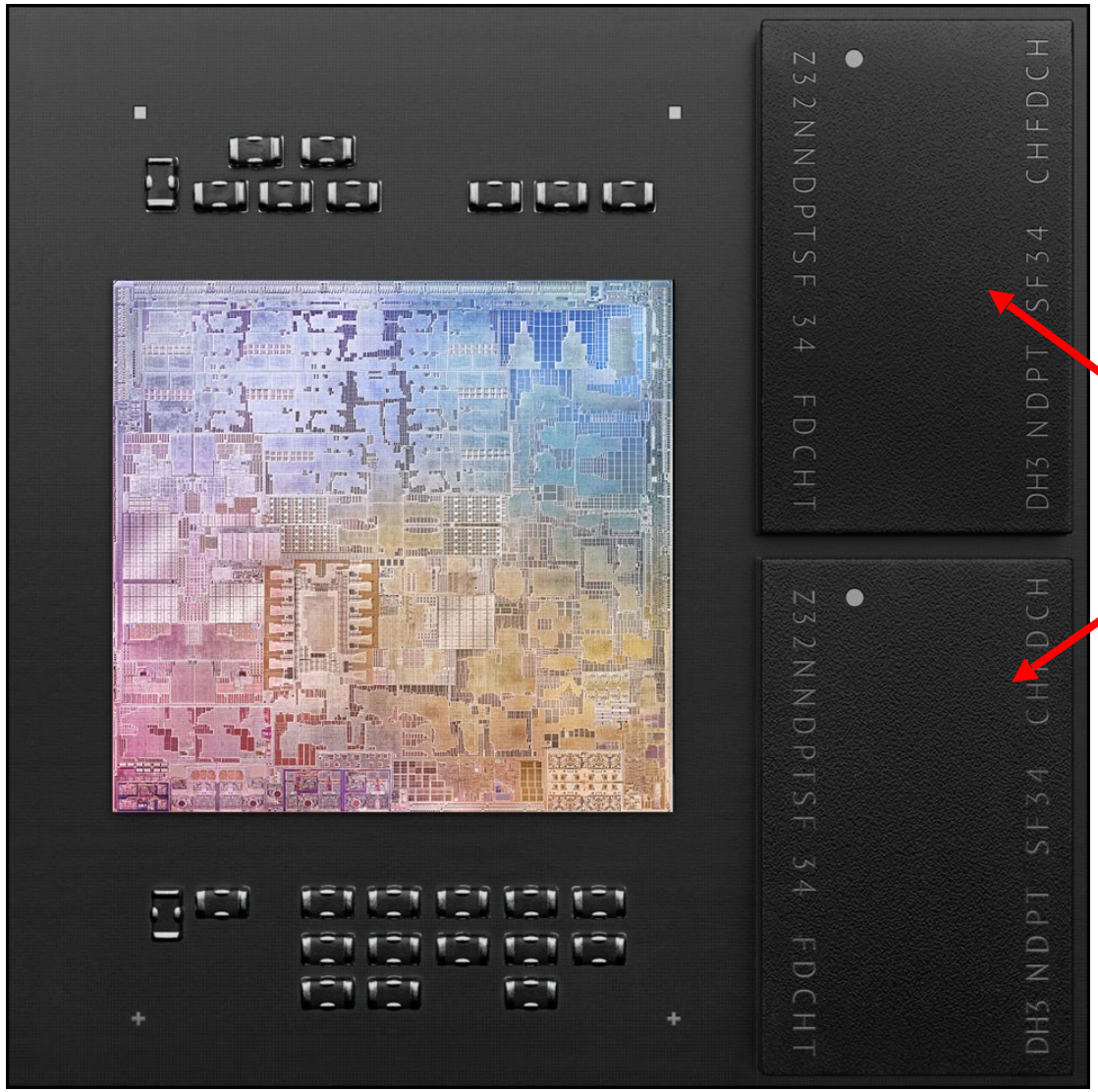
=> Bandwidth = 1ch * 3200 MT/s * 64b/T = 3.2 GT/s * 8B/T = 25.6GB/s

→ 32b data width & 2ch x DIMM → Peak MT/s

DDR5 DIMM : SKU CMK**32G**X5M2B**5200**C40

=> Bandwidth = 2ch * 5200 MT/s * 32b/T = 5.2 GT/s * 8B/T = 41.6GB/s

# DRAM Packaging, Apple M1



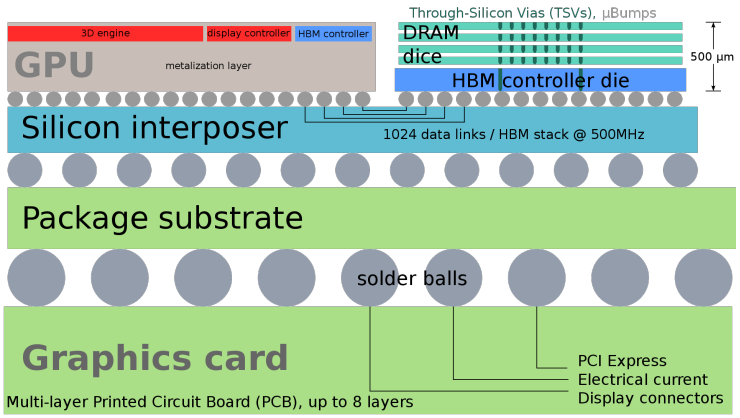

Two DRAM chips on same package as system SoC

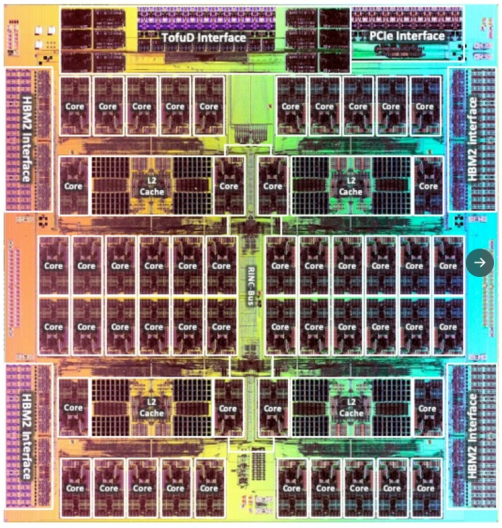
- 128b databus, running at 4.2Gb/s
- 68GB/s bandwidth

# HBM – High Bandwidth Memory

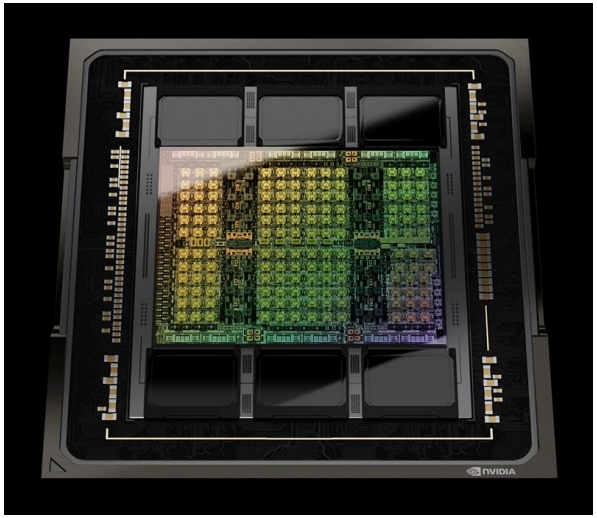
HBM – High Bandwidth Memory

- 4 stacked DDR die each with 2ch x 128bit
- HBM2 up to 3.2GT/s
- HBM3 up to 6.4GT/s




**FUJITSU AA64FX** 4x8GB HBM2



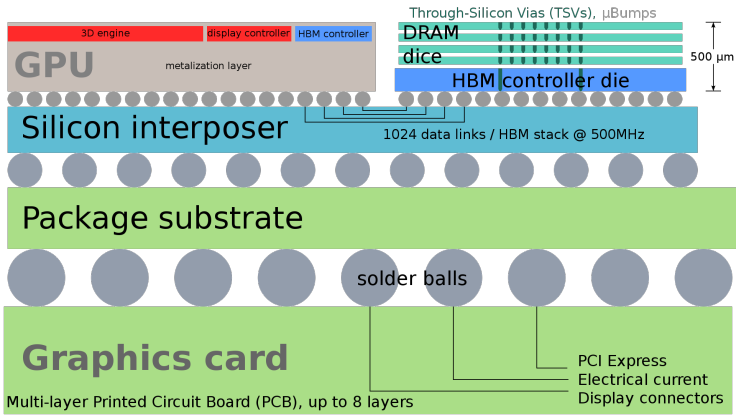
**NVIDIA GH100** 6x12GB HBM3

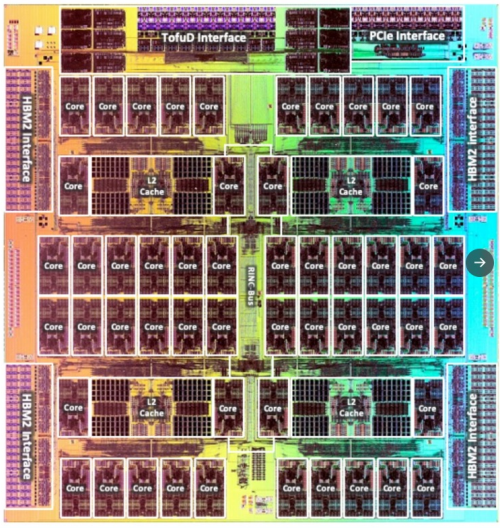
Latency >100ns

# HBM – High Bandwidth Memory

HBM – High Bandwidth Memory

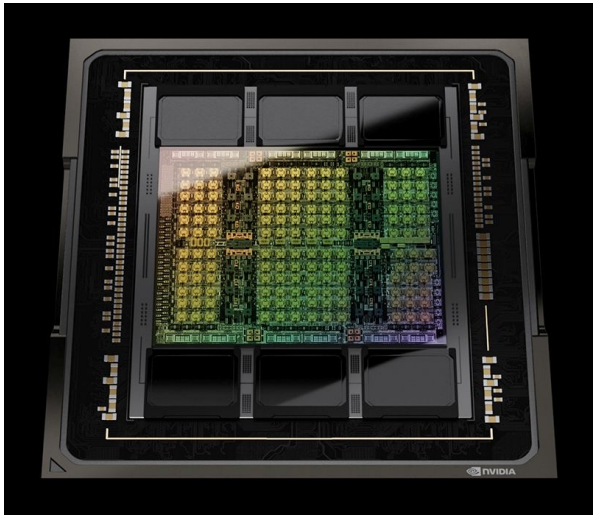
- 4 stacked DDR die each with 2ch x 128bit
- HBM2 up to 3.2GT/s
- HBM3 up to 6.4GT/s




**FUJITSU AA64FX** 4x8GB HBM2

- 1xHBM => 2GT/s x 1024b/T = 256GB/s
- 4xHBM => 4x 256GB/s = 1TB/s



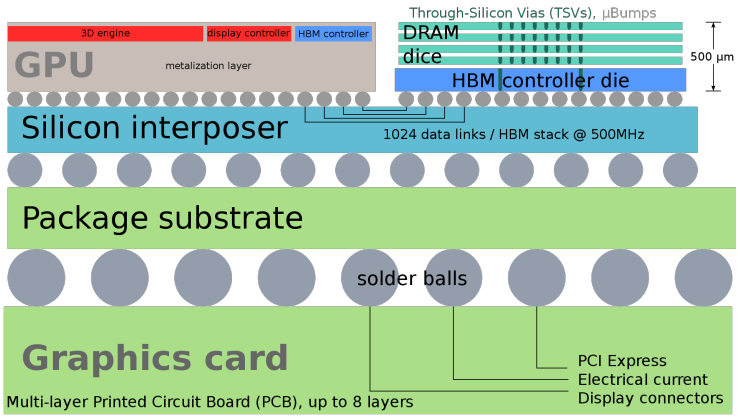
**NVIDIA GH100** 6x12GB HBM3

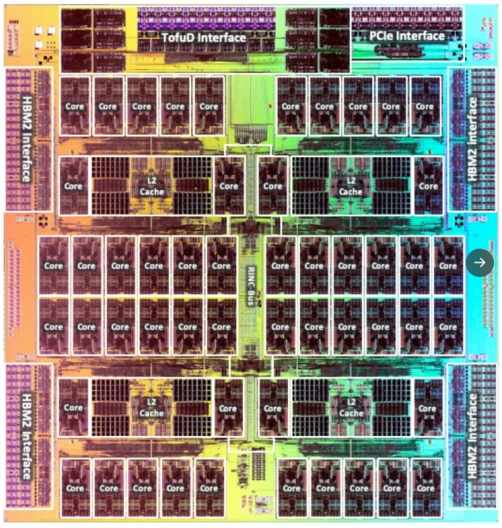
Latency >100ns

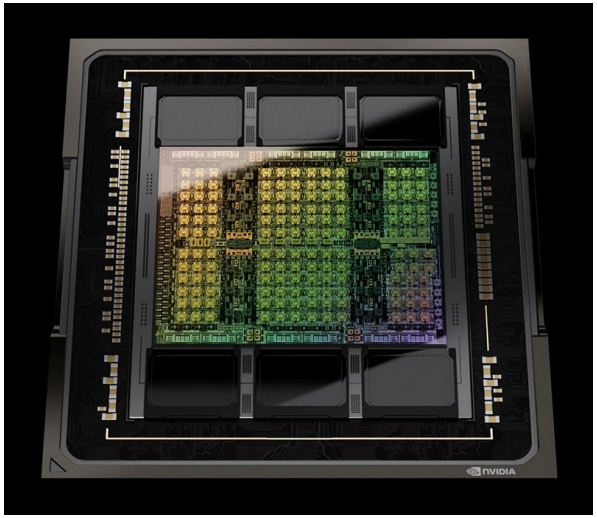
# HBM – High Bandwidth Memory

HBM – High Bandwidth Memory

- 4 stacked DDR die each with 2ch x 128bit
- HBM2 up to 3.2GT/s
- HBM3 up to 6.4GT/s



**FUJITSU AA64FX** 4x8GB HBM2

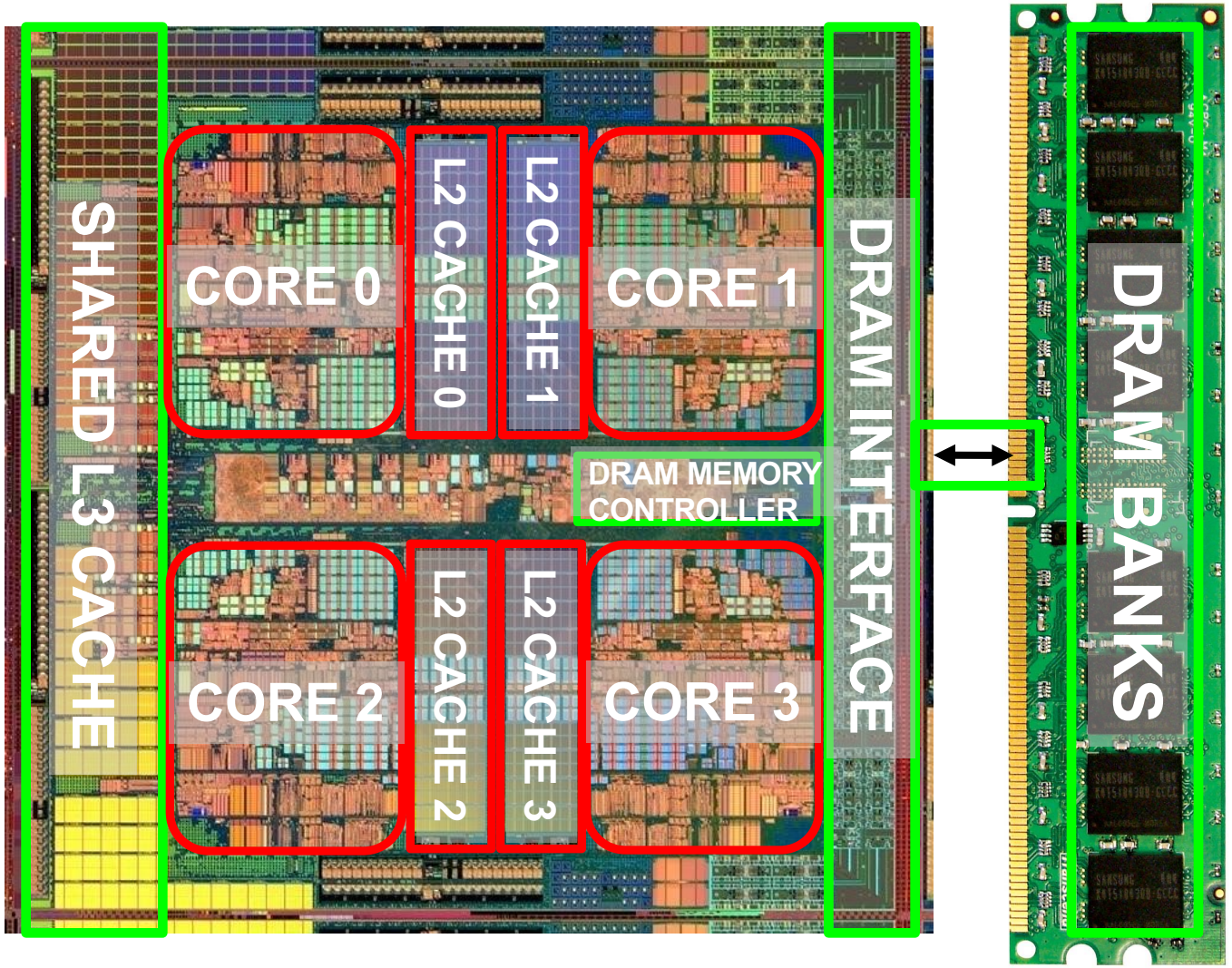
- 1xHBM => 2GT/s x 1024b/T = 256GB/s
- 4xHBM => 4x 256GB/s = 1TB/s

**NVIDIA GH100** 6x12GB HBM3

- 1xHBM => 4GT/s x 1024b/T = 412GB/s
- 6xHBM => 6x 512GB/s = 3TB/s

Latency >100ns

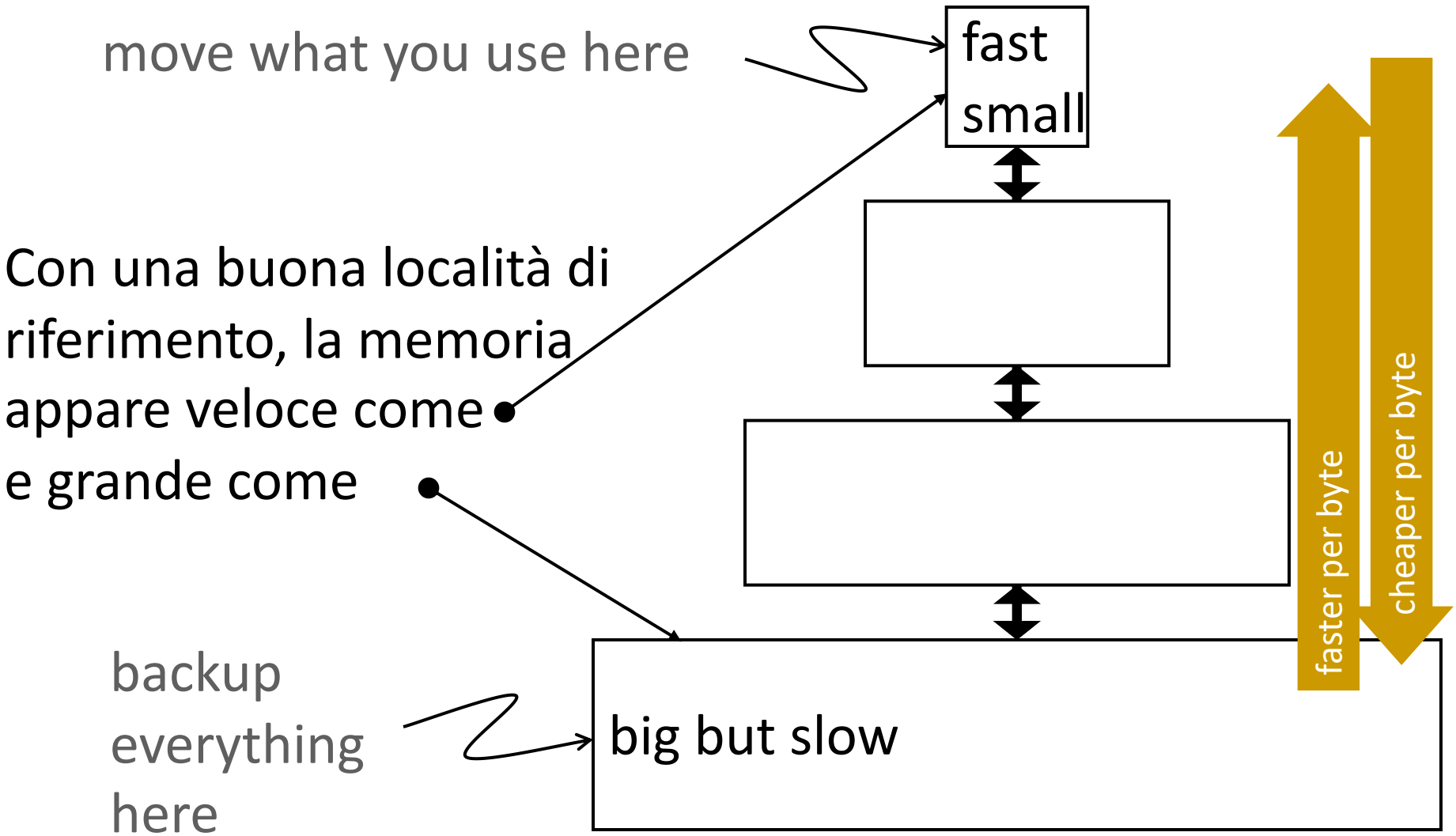
# Memory in a Modern System



# Why Memory Hierarchy?

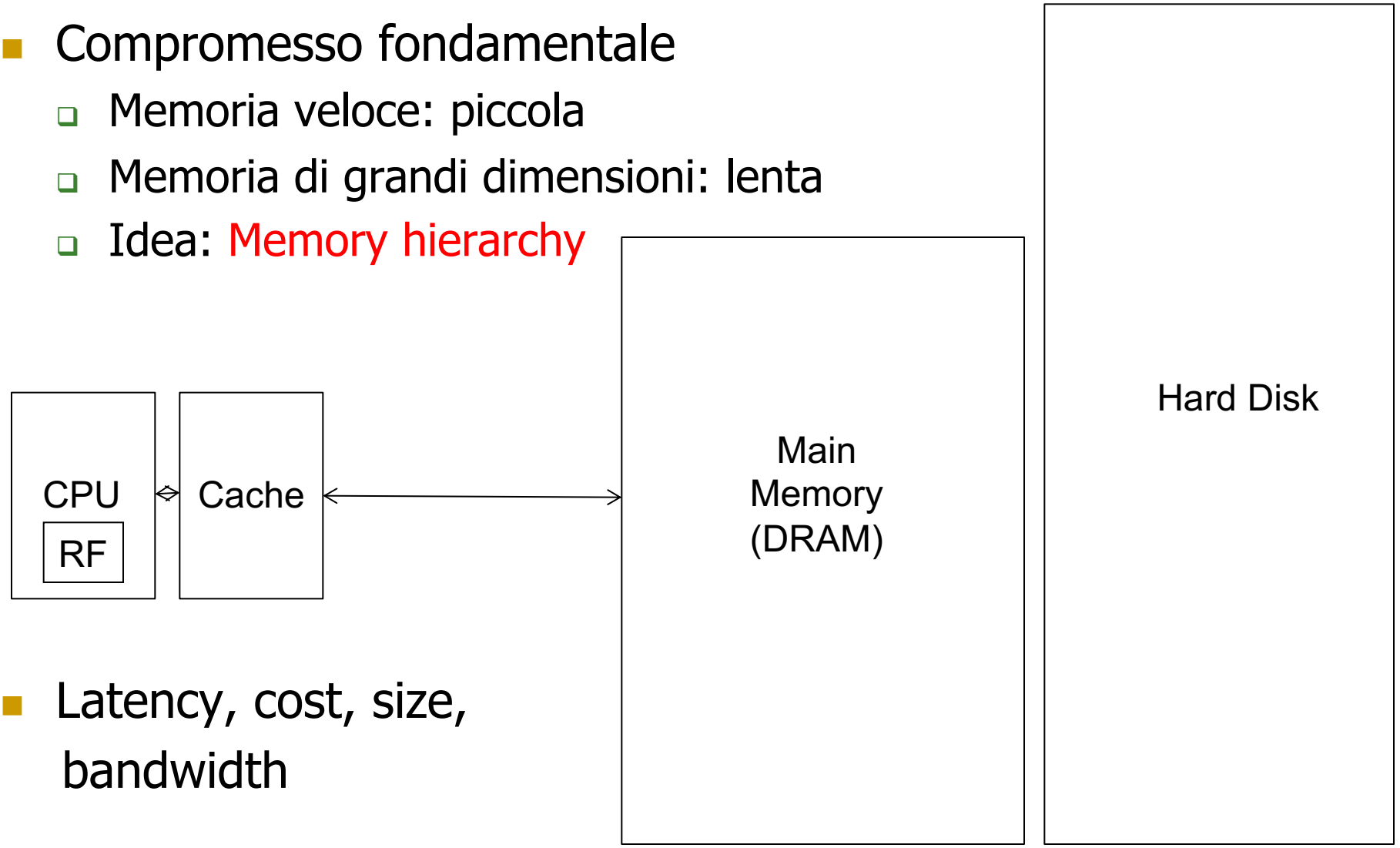
- Vogliamo la memoria sia veloce che grande
- Ma non possiamo ottenere entrambi con un unico livello di memoria
- Idea: Avere più livelli di storage (progressivamente più grandi e più lenti all'allontanarsi dal processore) e che garantiscano che la maggior parte dei dati di cui il processore ha bisogno sia mantenuta nel (nei) livello(i) vicini

# The Memory Hierarchy



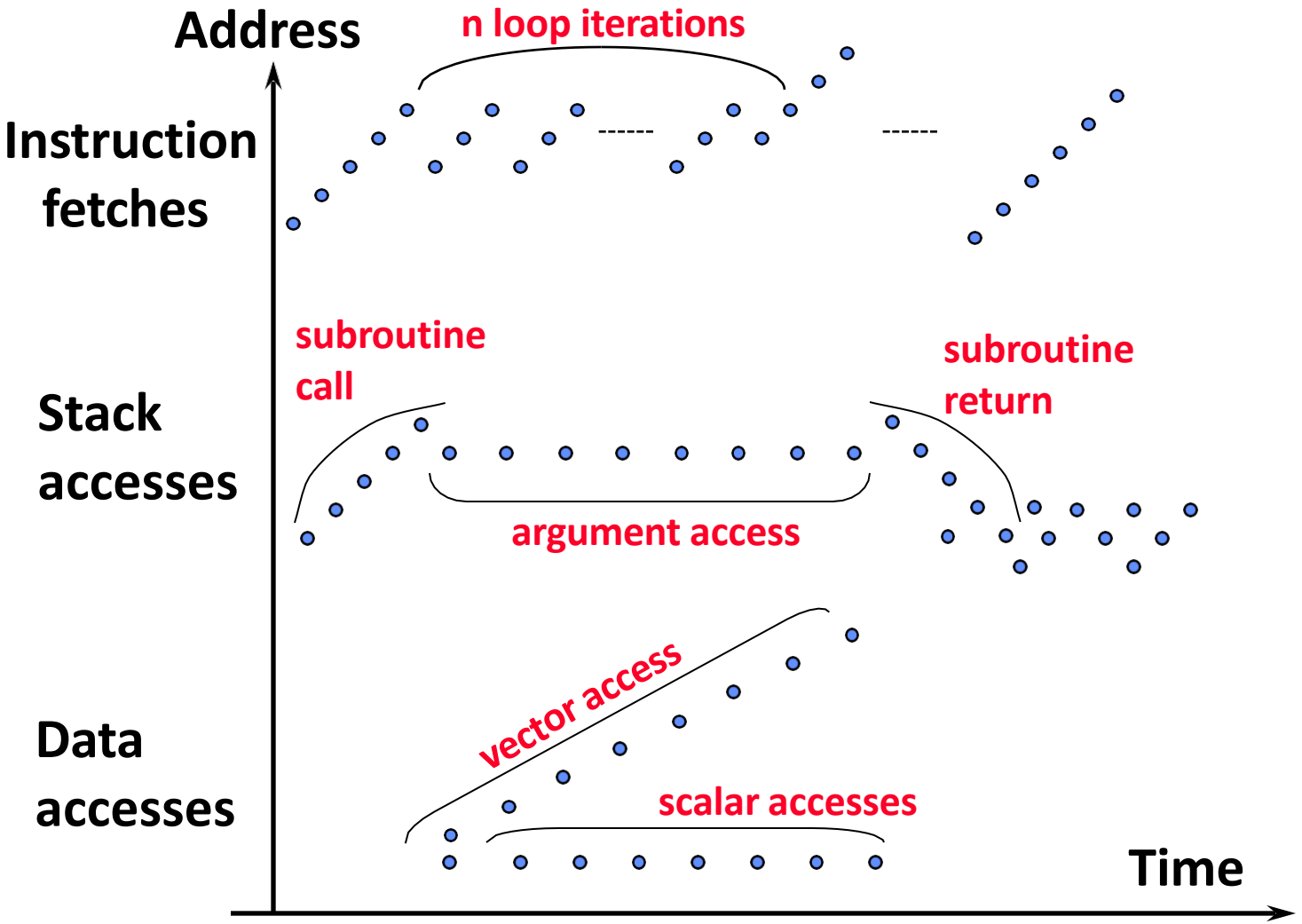
# Memory Hierarchy

- Compromesso fondamentale
  - Memoria veloce: piccola
  - Memoria di grandi dimensioni: lenta
  - Idea: Memory hierarchy




- Latency, cost, size, bandwidth

# Typical Memory Reference Patterns



# Località / Locality

- Il passato recente è un ottimo predittore/stima del prossimo futuro.

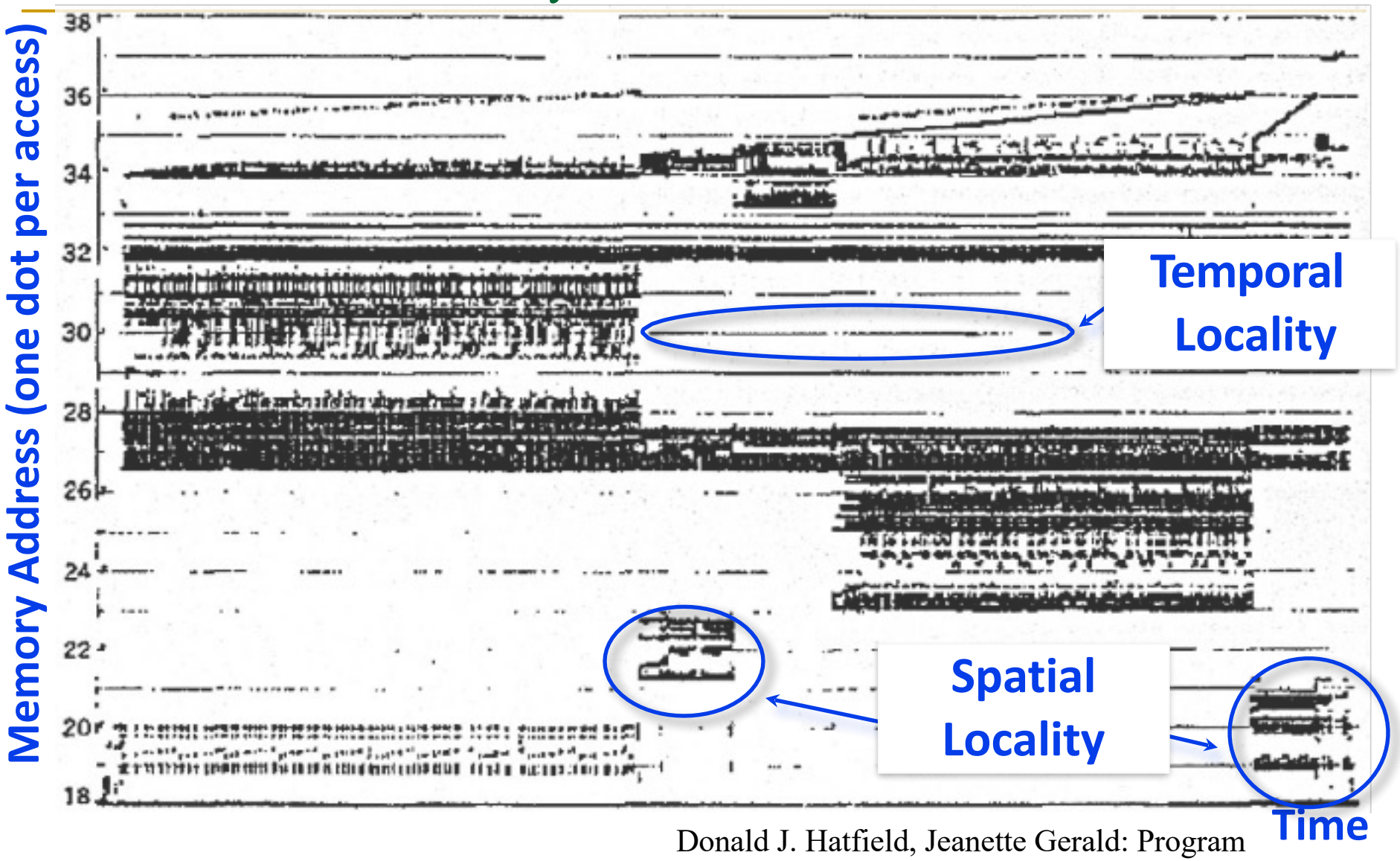
- Località temporale (Temporal Locality): Se hai appena fatto qualcosa, è molto probabile che la rifarai presto
  - dal momento che siete qui oggi, c'è una buona probabilità che sarete qui ancora e ancora regolarmente
  -

- Località Spaziale (Spatial Locality): Se hai fatto qualcosa, è molto probabile che rifarai qualcosa di simile / correlato (nello spazio)
  - ogni volta che ti trovo in questa stanza, probabilmente sei seduto vicino alle stesse persone
  -

# Memory Locality

- Un programma "tipico" ha un sacco di località nei riferimenti di memoria
  - i programmi tipici sono composti da "loop«
- Temporale: Un programma tende ad indirizzare la stessa locazione di memoria molte volte e tutte in un breve intervallo di tempo
- Spaziale: Un programma tende ad indirizzare posizioni di memoria consecutive
  - esempi rilevanti:
    - 1. accessi alla memoria delle istruzioni
    - 2. accesso a campi di strutture di matrici/dati

# Memory Reference Patterns




Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192 (1971)

# Caching Basics: Exploit Temporal Locality

- Idea: Immagazzinare i dati utilizzati di recente in una memoria veloce gestita automaticamente (denominata cache)
- Anticipazione: i dati saranno acceduti nuovamente in un breve intervallo
- Principio di Località Temporale
- I dati a cui si è acceduto di recente saranno nuovamente acceduti nel prossimo futuro
- Concetto introdotto da Maurice Wilkes in:
  - Wilkes, "Slave Memories and Dynamic Storage Allocation," IEEE Trans. On Electronic Computers, 1965.
  - "The use is discussed of a fast core memory of, say 32000 words as a slave to a slower core memory of, say, one million words in such a way that in practical cases the effective access time is nearer that of the fast memory than that of the slow memory."

# Caching Basics: Exploit Spatial Locality

- Idea: Memorizzare i dati presenti agli indirizzi adiacenti a quello a cui si è acceduto di recente nella memoria veloce gestita automaticamente
  - Dividere logicamente la memoria in blocchi di uguale dimensione
  - Recuperare per memorizzare interamente nella cache il blocco a cui sta accedendo
  - Anticipazione: i dati nelle vicinanze saranno acceduti in un breve intervallo
- Principio di Località Spaziale
  - I dati vicini saranno acceduti nel prossimo futuro
  - E.s., accesso sequenziale alle istruzioni, scorrimento array
  - Realizzato già nell' IBM 360/85
    - 16 Kbyte cache with 64 byte blocks
    - Liptay, "Structural aspects of the System/360 Model 85 II: the cache," IBM Systems Journal, 1968.

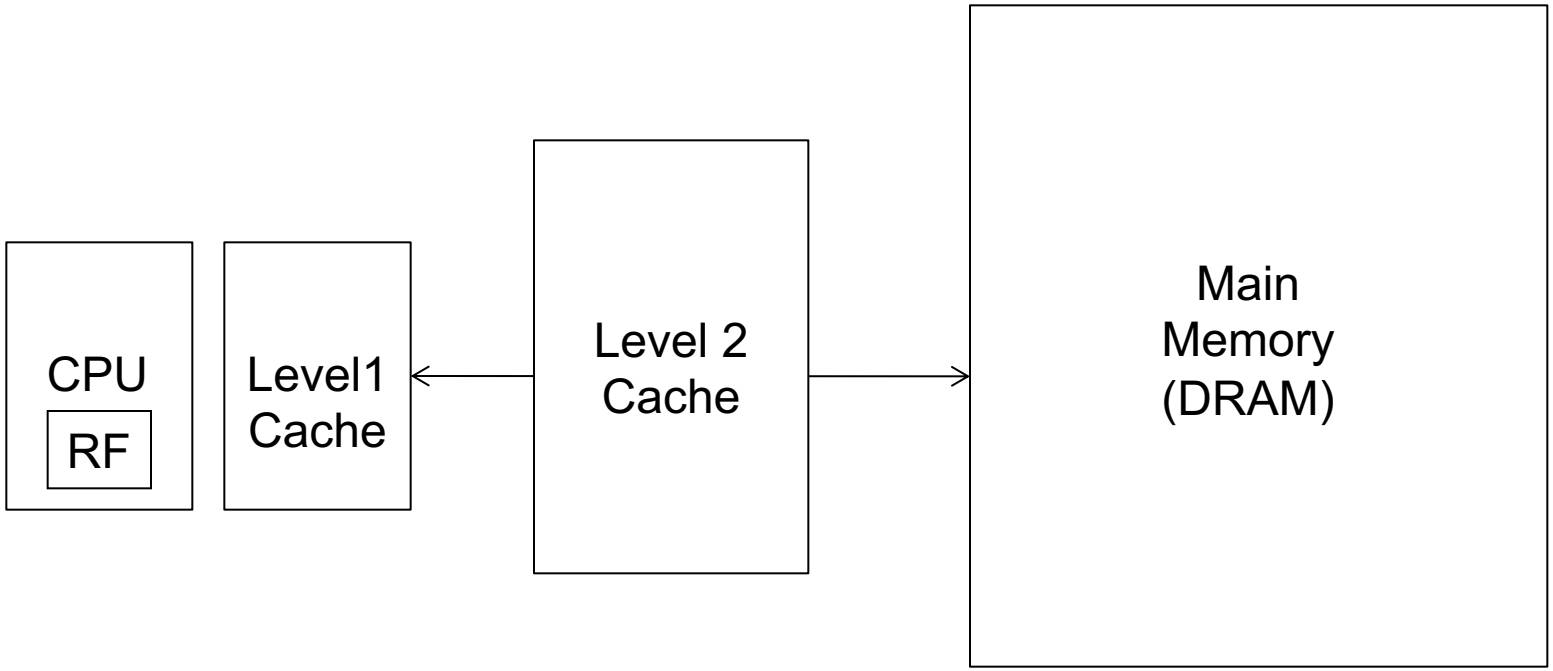
# The Bookshelf Analogy

- Book in your hand
- Desk
- Bookshelf
- Boxes at home
- Boxes in storage

- Recently-used books tend to stay on desk
  - Comp Arch books, books for classes you are currently taking
  - Until the desk gets full
- Adjacent books in the shelf needed around the same time
  - If I have organized/categorized my books well in the shelf

# Caching in a Pipelined Design

- La cache deve essere strettamente integrata nella pipeline
  - Idealmente, con accesso in 1 ciclo in modo che le operazioni dipendenti dalla load non si blocchino
- Pipeline ad alta frequenza -> impossibile rendere grande la cache
  - Ma, vogliamo una cache grande ed una pipeline veloce
- Idea: Cache hierarchy



# A Note on Manual vs. Automatic Management

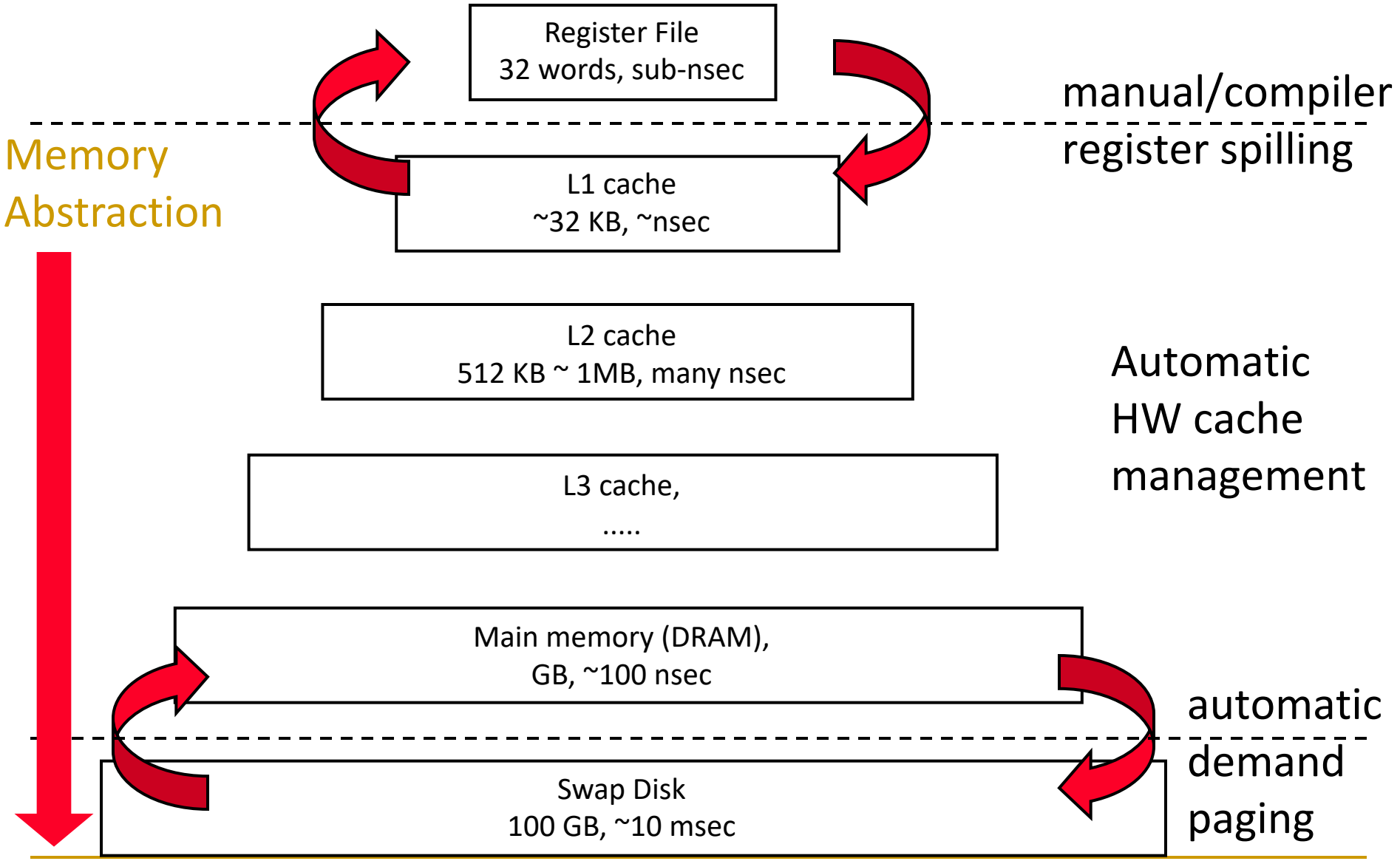
- Manuale: Il programmatore gestisce lo spostamento dei dati tra i livelli

  -- troppo doloroso per i programmatori su programmi complessi

  - usata in domain specific architectures / digital signal processor
  - ancora usata in processori embedded ( si usa un on-chip scratch pad SRAM al posto di una cache) e GPU (chiamata «shared memory»)

- Automatico: L'hardware gestisce lo spostamento dei dati tra i livelli, in modo trasparente al programmatore
- ++ rende più facile la vita del programmatore
  - il programmatore medio non ha bisogno di conoscerne l'esistenza
  - Non c'è bisogno di conoscere quanto è grande la cache e come funziona per scrivere un programma "corretto"! (Se si desidera un programma "veloce"?)
  -

# A Modern Memory Hierarchy



# Hierarchical Latency Analysis

- Per un determinato livello di gerarchia di memoria **i** con un tempo di accesso tecnologico-intrinseco di $t_i$. Il tempo di accesso percepito $T_i$ è più lungo di $t_i$
- Fatta eccezione per la gerarchia più esterna, se si cerca un determinato indirizzo c'è
  - una probabilità (hit-rate $h_i$) di "hit" e access time di $t_i$
  - una probabilità (miss-rate $m_i$) di "miss" e access time di $t_i + T_{i+1}$
  - $h_i + m_i = 1$
- Perciò

$$T_i = h_i \cdot t_i + m_i \cdot (t_i + T_{i+1})$$

$$T_i = t_i + m_i \cdot T_{i+1}$$

$h_i$ e $m_i$ sono definiti per essere la hit-rate e miss-rate dei soli accessi che non hanno mancato il livello $L_{i-1}$

# Hierarchy Design Considerations

- Equazione della latenza ricorsiva

$$T_i = t_i + m_i \cdot T_{i+1}$$

- L'obiettivo: raggiungere il $T_1$ desiderato nel costo permesso
- si vorrebbe $T_i \approx t_i$, due strade percorribili:

- Tenere $m_i$ basso
  - Aumentare la capacità $C_i$ riducendo $m_i$, ma attenzione ad aumentare $t_i$
  - Ridurre $m_i$ con una gestione più intelligente della cache (replacement::anticipate what you don't need, prefetching::anticipate what you will need)

- Tenere $T_{i+1}$ basso
  - Velocizzando gli ultimi livelli, ma attenzione ai costi
  - introducendo come compromesso gerarchie intermedie

# Intel Pentium 4 Example

- 90nm P4, 3.6 GHz
- L1 D-cache
  - $C_1$ = 16K
  - $t_1$ = 4 cyc int / 9 cycle fp
- L2 D-cache
  - $C_2$ =1024 KB
  - $t_2$ = 18 cyc int / 18 cyc fp
- Main memory
  - $t_3$ = ~ 50ns or 180 cyc
- Notice
  - best case latency is not 1
  - worst case access latencies are into 500+ cycles

if $m_1$=0.1, $m_2$=0.1
$T_1$=7.6, $T_2$=36

if $m_1$=0.01, $m_2$=0.01
$T_1$=4.2, $T_2$=19.8

if $m_1$=0.05, $m_2$=0.01
$T_1$=5.00, $T_2$=19.8

if $m_1$=0.01, $m_2$=0.50
$T_1$=5.08, $T_2$=108