PROGRAM STUDI

S1 SISTEM KOMPUTER

UNIVERSITAS DIPONEGORO

HIRARKI DAN CACHE MEMORI

Oky Dwi Nurhayati, ST, MT
email: okydn@undip.ac.id

MEMORY HIERARCHY

Memory Hierarchy (1/4)

- Prosesor
  - menjalankan program
  - sangat cepat waktu eksekusi dalam orde nanoseconds sampai dengan picoseconds
  - perlu mengakses kode dan data program! Dimana program berada?
- Disk
  - HUGE capacity (virtually limitless)
  - VERY slow: runs on order of milliseconds
  - so how do we account for this gap?

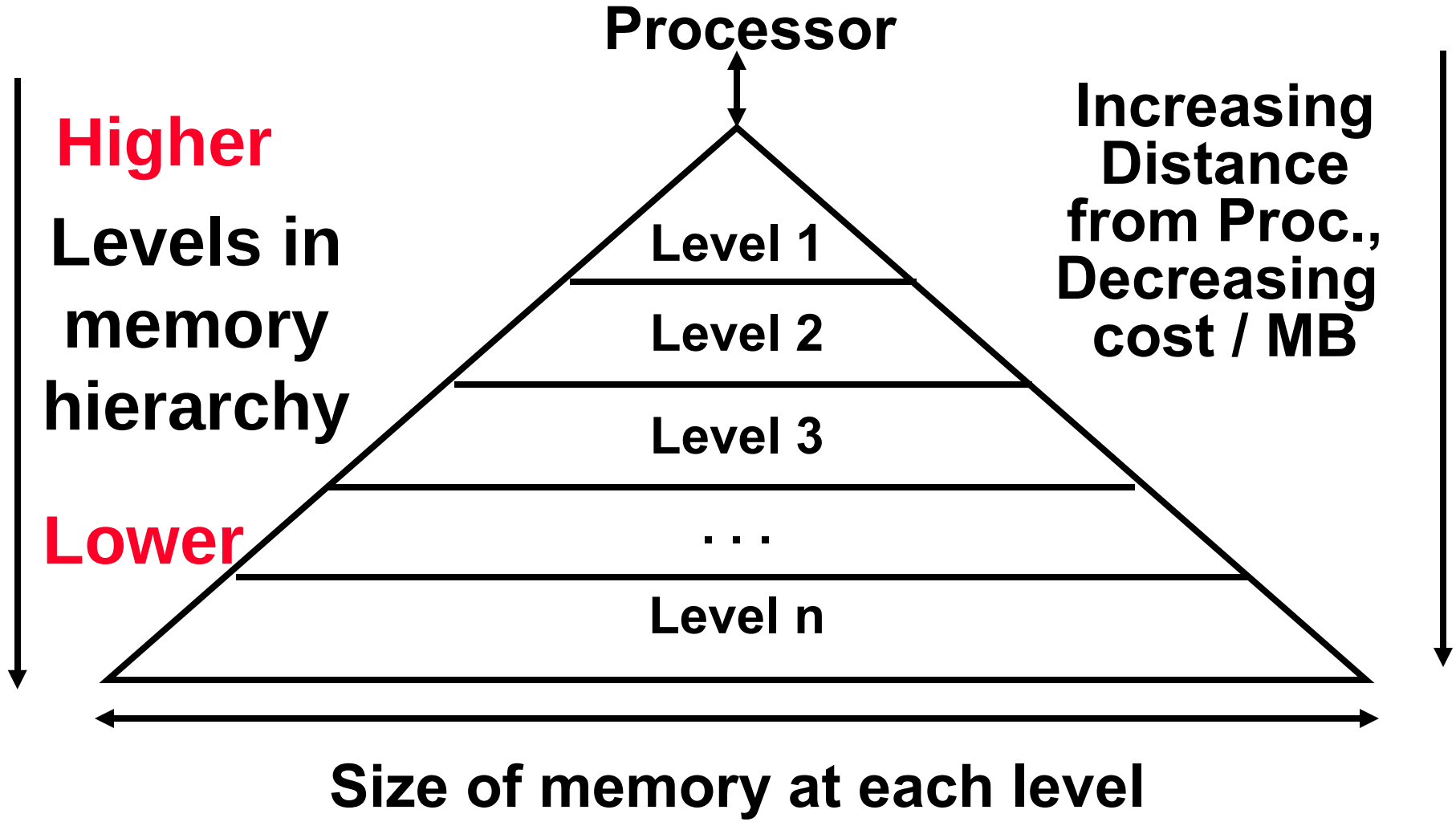
➲ Menggunakan teknologi memori!

Memory Hierarchy (2/4)

- Memory (DRAM)
  - Kapasitas jauh lebih besar dari registers, lebih kecil dari disk (tetap terbatas)
  - Access time ~50-100 nano-detik, jauh lebih cepat dari disk (mili-detik)
  - Mengandung subset data pada disk (basically portions of programs that are currently being run)

- Fakta: memori dengan kapasitas besar (murah!) lambat, sedangkan memori dengan kapasitas kecil (mahal) cepat.
- Solution: bagaimana menyediakan (ilusi) kapasitas besar dan akses cepat!

Memory Hierarchy (3/4)



Memory Hierarchy (4/4)

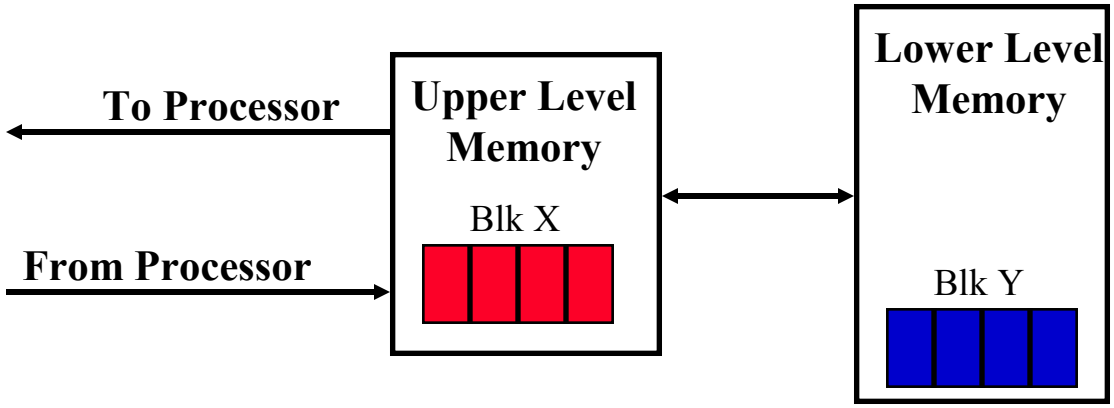
- Pada tingkat yang lebih dekat dengan Prosesor, mempunyai karakteristik:
  - Lebih kecil,
  - Lebih cepat,
  - Subset semua data pada level lebih atas (mis. menyimpan data yang sering digunakan)
  - Efisien dalam pemilihan mana data yang akan disimpan, karena tempat terbatas
- Lowest Level (usually disk) contains all available data

Why hierarchy works

- The Principle of Locality:
  - Program access a relatively small portion of the address space at any instant of time.

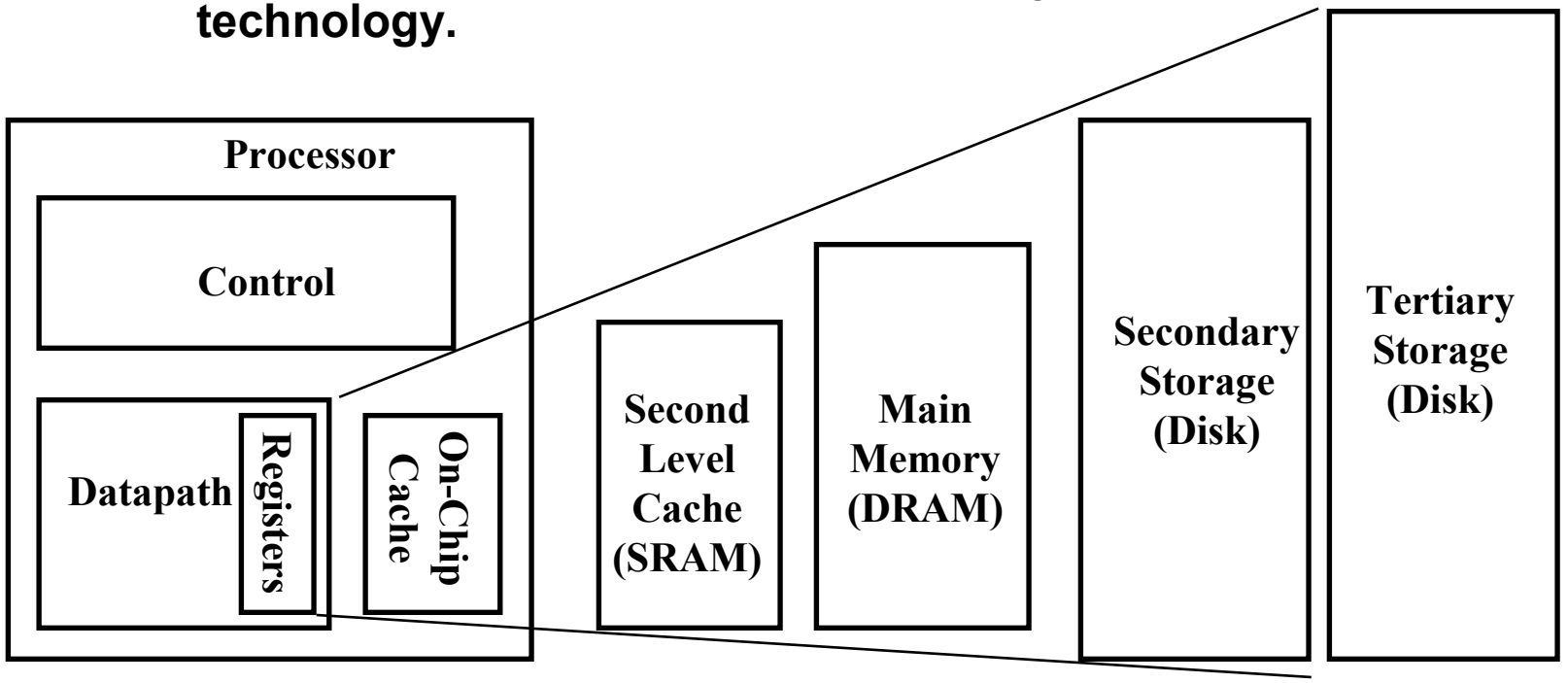
Memory Hierarchy: How Does it Work?

- **Temporal Locality** (Locality in Time):
  - → Keep most recently accessed data items closer to the processor
- **Spatial Locality** (Locality in Space):
  - → Move blocks consists of contiguous words to the upper levels



Modern Computer System

- By taking advantage of the principle of locality:
  - Present the user with as much memory as is available in the cheapest technology.
  - Provide access at the speed offered by the fastest technology.



Speed (ns):	1s	10s	100s	10,000,000s (10s ms)	10,000,000,000s (10s sec)
Size (bytes):	100s	Ks	Ms	Gs	Ts

How is the hierarchy managed?

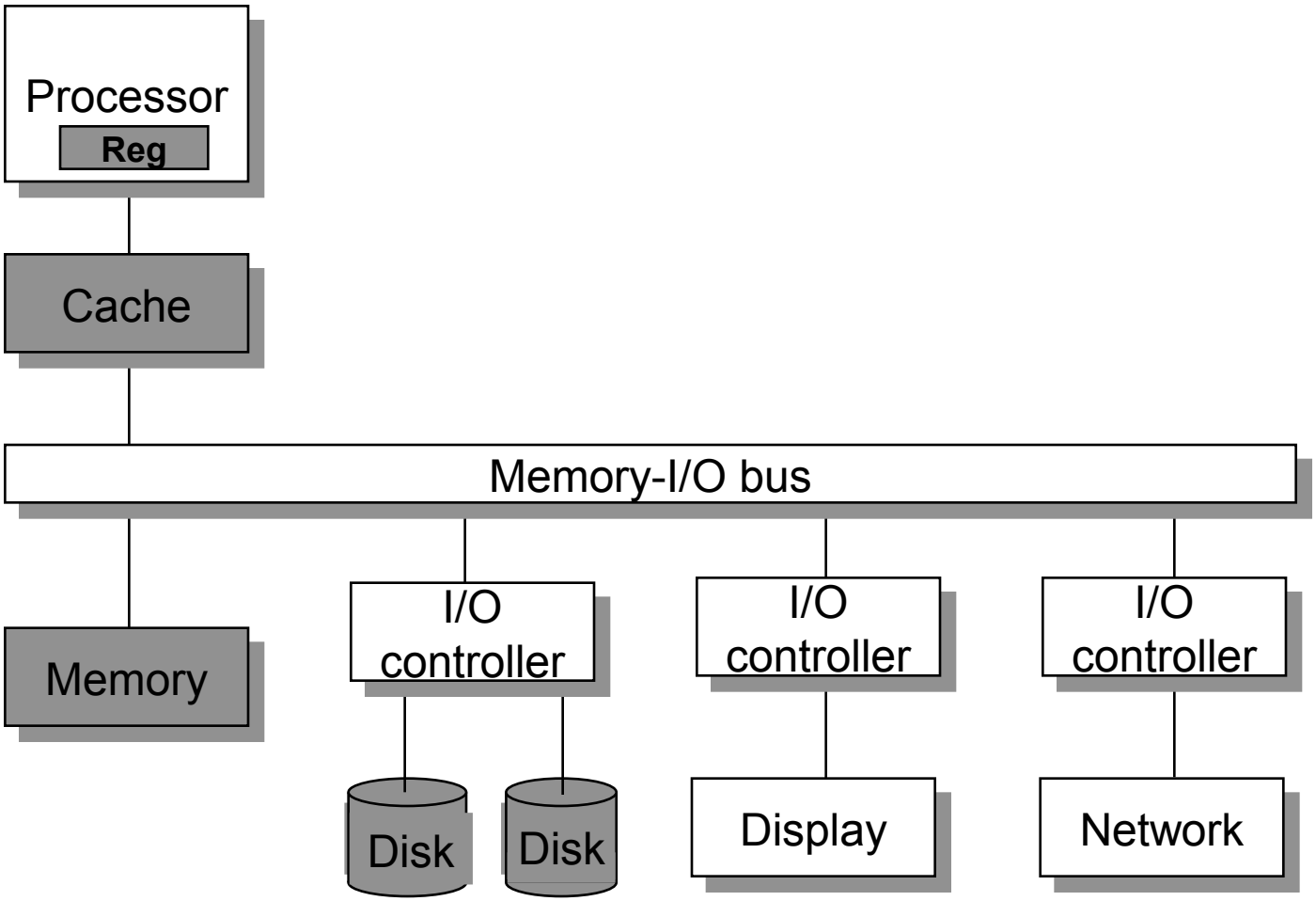
- Registers <-> Memory
  - by compiler (programmer?)
- Cache <-> Memory
  - by the hardware
- Memory <-> Disks
  - by the hardware and operating system (virtual memory)
  - by the programmer (files)

Basis of Memory Hierarchy

- Disk contains everything.
- When Processor needs something, bring it into to all lower levels of memory.
- Cache contains copies of data in memory that are being used.
- Memory contains copies of data on disk that are being used.
- Entire idea is based on Temporal Locality: if we use it now, we'll want to use it again soon

Cache

Organisasi Hierarki Memori: Cache



What is a cache?

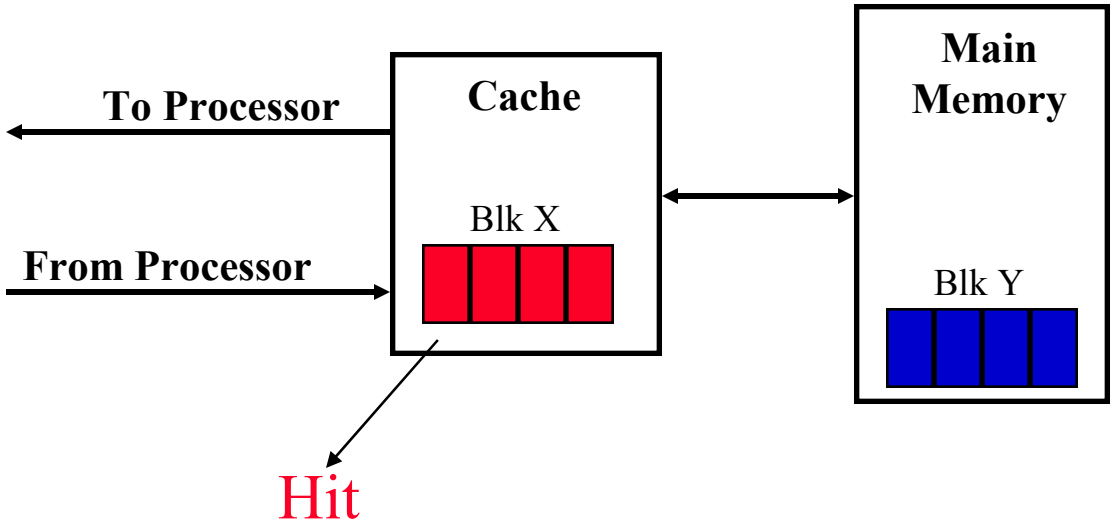
- Kecil, storage cepat untuk meningkatkan "average access time" dibandingkan memori
- Memanfaatkan spatial & temporal locality
- Sebenarnya "cache" terlihat pada hirarkis penyimpanan
  - Registers "a cache" on variables – software managed
  - First-level cache a cache on second-level cache
  - Second-level cache a cache on memory
  - Memory a cache on disk (virtual memory)

Cache Design

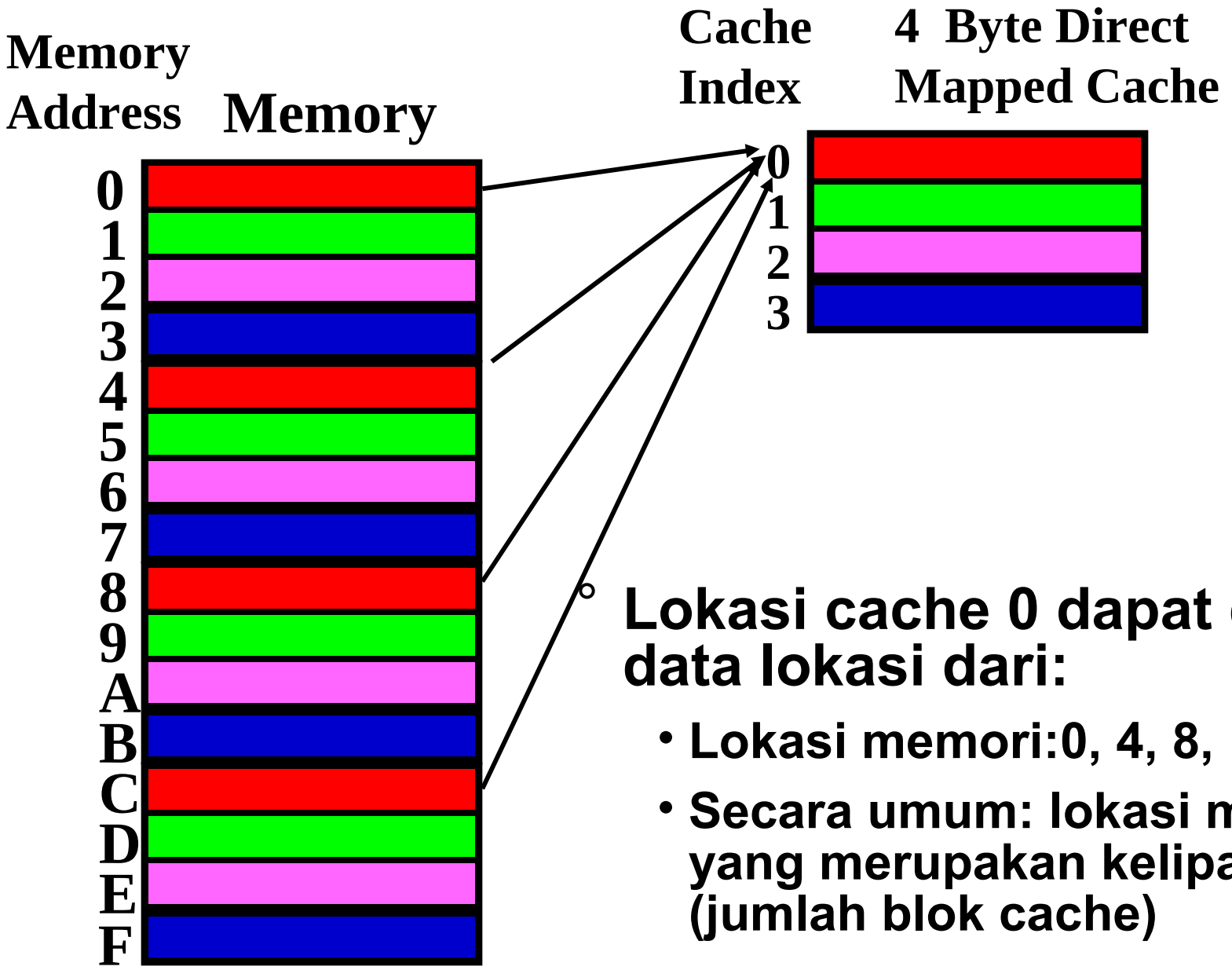
- Bagaimana organisasi cache? (ingat: cache akan menerima alamat memori dari prosesor, tidak ada perubahan pada rancangan prosesor ada/tanpa cache).
- Bagaimana melakukan mapping (relasi) antara alamat memori dengan cache (ingat: prosesor hanya melihat memory byte addressable)
- Bagaimana mengetahui elemen data tsb berada di cache (hit) atau tidak ada (miss) (ingat: cache jauh lebih kecil dari main memory)?

Cache: Blok Memory

- **Transfer data antara cache dan memori dalam satuan blok (kelipatan words)**
- **Mapping (penerjemahan) antara blok di cache dan di main memory (ingat: cache copy dari main memory)**



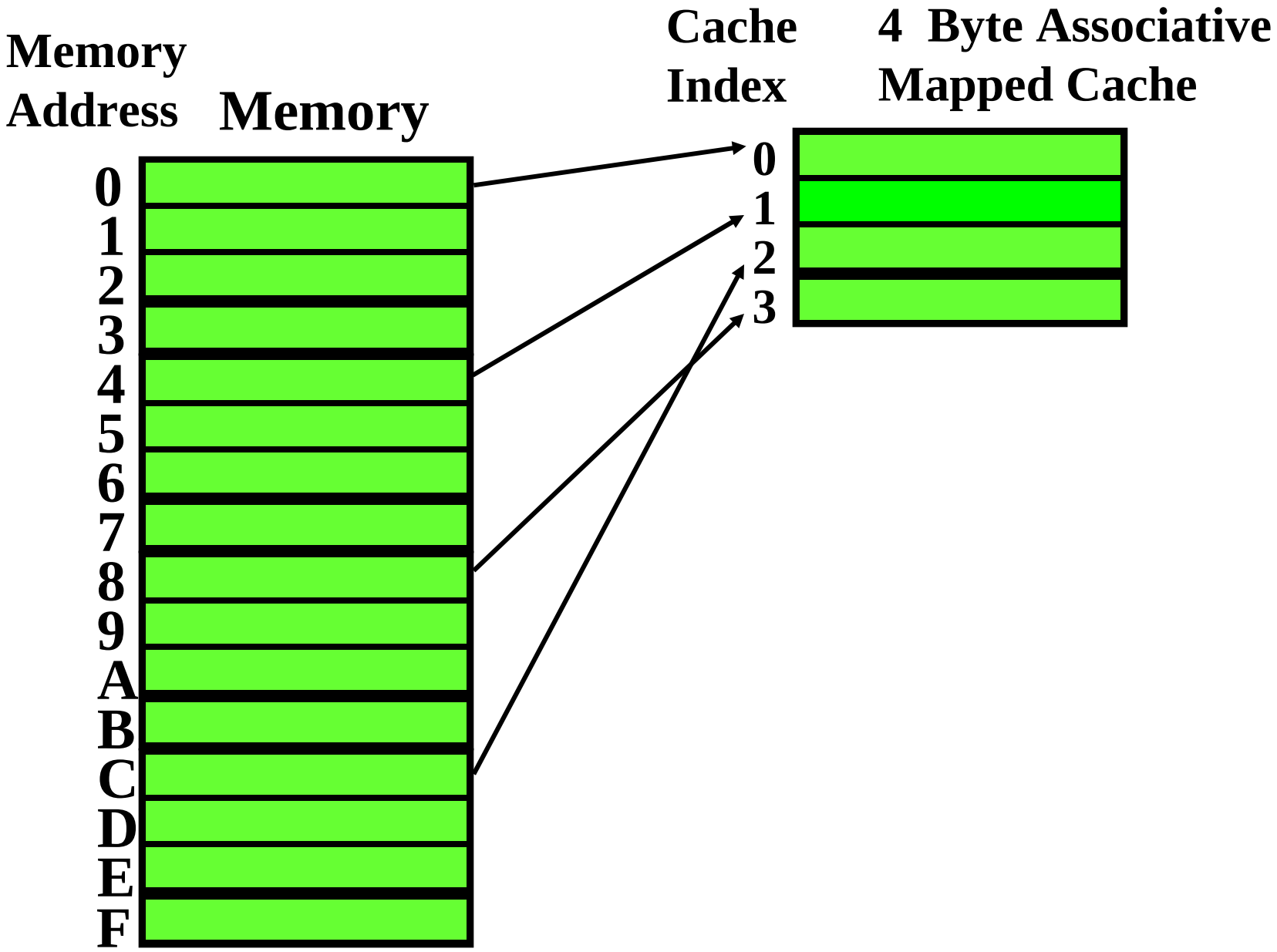
Direct Mapping



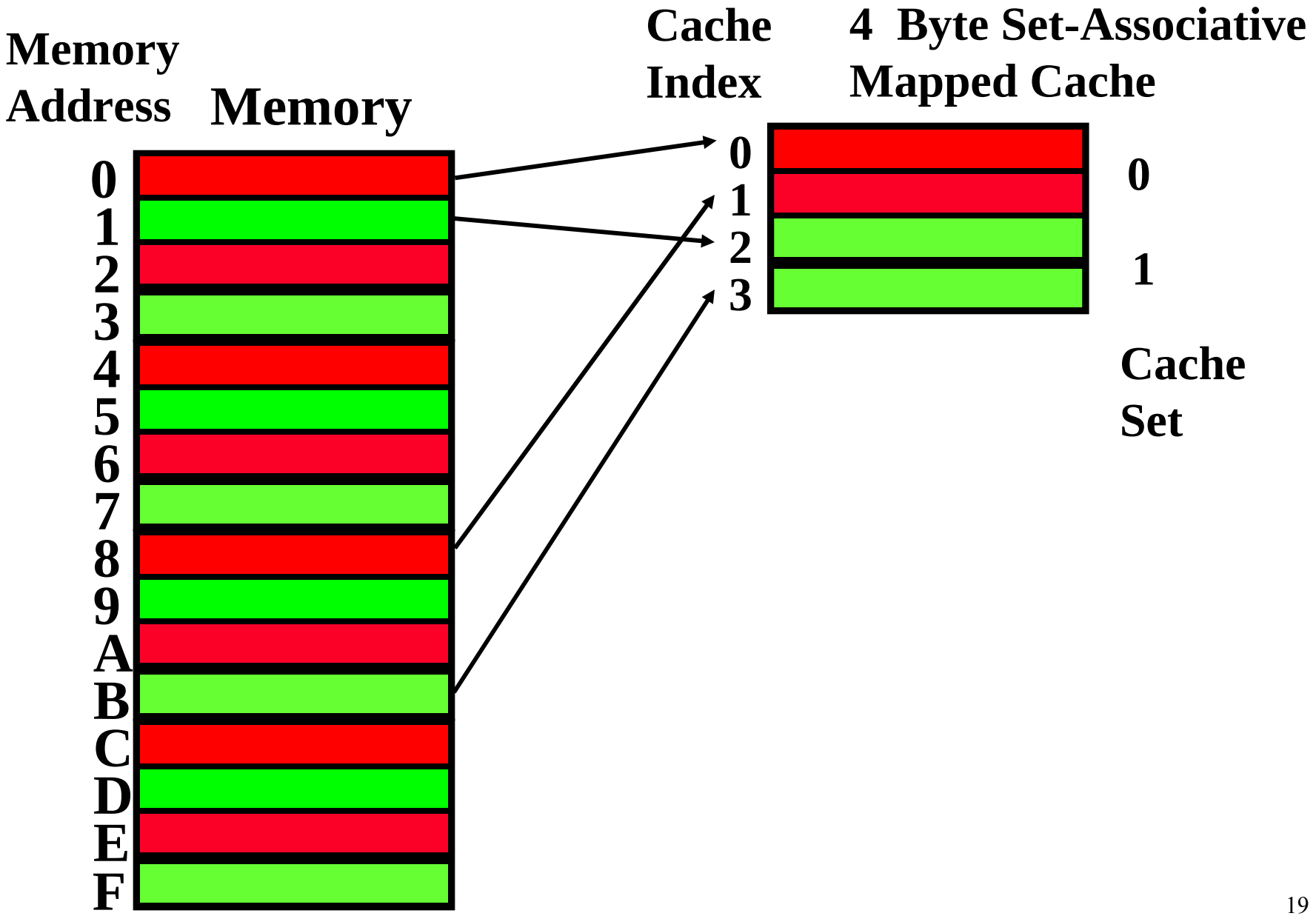
Lokasi cache 0 dapat diisi data lokasi dari:

- **Lokasi memori:0, 4, 8, ...**
- **Secara umum: lokasi memori yang merupakan kelipatan 4 (jumlah blok cache)**

Associative Mapping



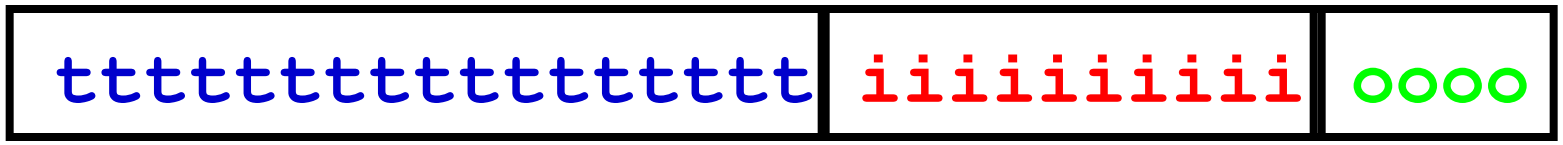
Set-Associative Mapping



Address Mapping

Example: Direct-Mapped

- Masalah: multiple alamat memori “map” ke indeks blok yang sama!
  - Bagaimana kita mengetahui apakah alamat blok yang diinginkan berada di cache?
  - Harus ada identifikasi blok memori dikaitkan dengan alamat
- Bagaimana jika ukuran blok > 1 byte?
- Solusi: bagi alamat dalam bentuk fields



tag	index	byte
to check	to	offset
if have	select	within
correct block	block	block

Cache Address Terminology

- Semua fields untuk penerjemahan dianggap sebagai bilangan positif integer.
- **Index**: indeks pada blok cache, dimana blok (baris, entry) dari cache alamat tersebut harus berada.
- **Offset**: sekali kita telah menemukan blok tsb, manakah byte pada blok tersebut akan diakses.
- **Tag**: sisa bit dari alamat setelah field index dan offset; digunakan untuk membedakan alamat memory yang mappingnya pada blok yang sama dari cache.

Direct-Mapped Cache Example (1/3)

- Misalkan kita mempunya 16KB data dengan skema direct-mapped cache.
  - Setiap blok terdiri dari 4 word
- Tentukan ukuran field: tag, index, dan offset jika menggunakan komputer 32-bit.
- Offset
  - Diperlukan untuk mengambil satu byte dalam blok
  - blok mempunyai: 4 words
    16 bytes
    2^4 bytes
  - diperlukan 4 bit untuk menentukan alamat byte

Direct-Mapped Cache Example (2/3)

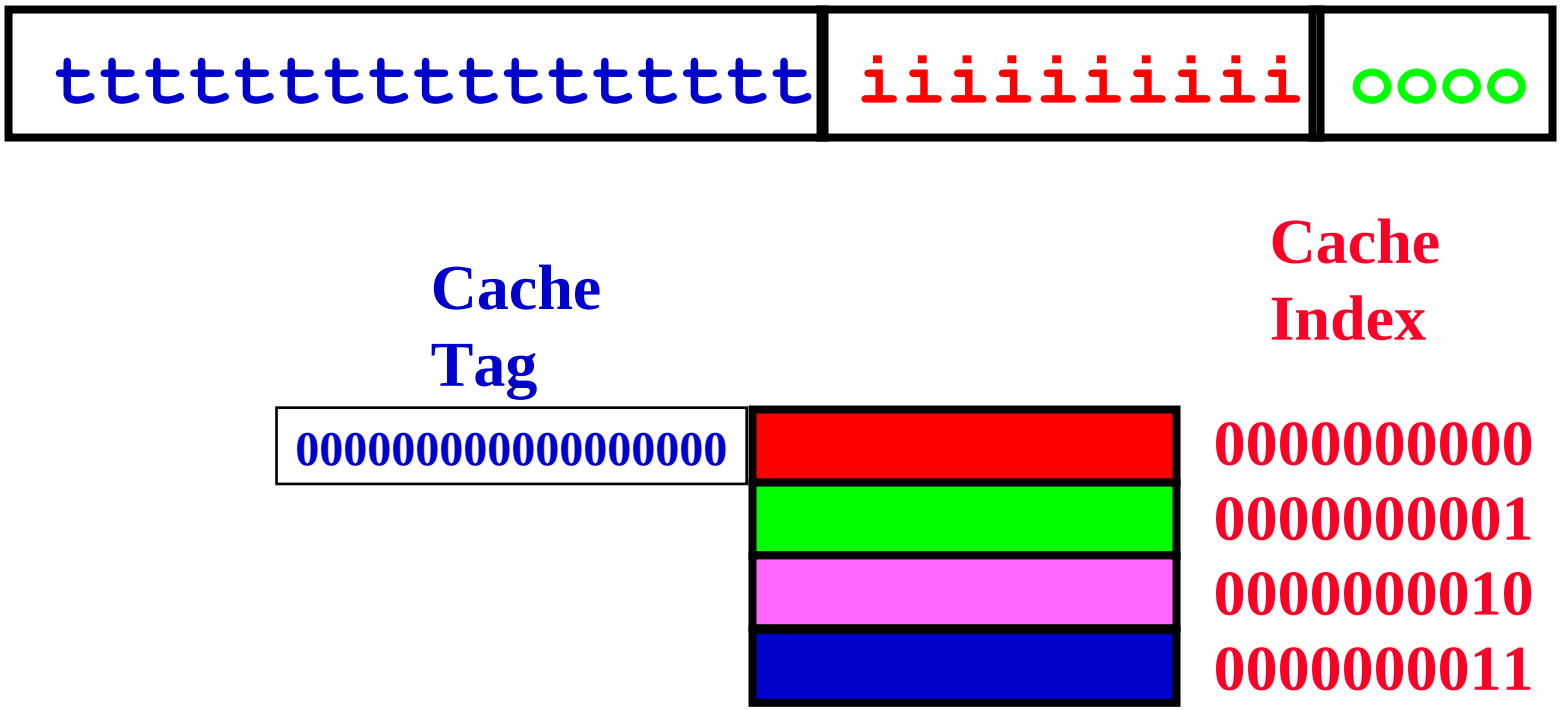
- **Index**: (~index into an "array of blocks")
  - diperlukan untuk menentukan blok yang mana pada cache (**rows** yang mana)
  - cache mempunyai 16 KB = 2^{14} bytes
  - blok mempunyai 2^4 bytes (4 words)
  - # rows/cache = # blocks/cache (sebab setiap rows terdiri dari satu blok)
    = bytes/cache / bytes/row
    = 2^{14} bytes/cache / 2^4 bytes/row
    = 2^{10} rows/cache
  - diperlukan **10 bit** untuk menentukan indeks pada rows dari cache

- **Tag**: 32 – (**4** + **10**) = **18 bit**

Direct-Mapped Cache Example (3/3)

- Contoh direct mapped cache tsb membagi field atas
  - 4 bits offset
  - 10 bit index
  - 18 bit tag
- Struktur cache:
  - Menyimpan tag (unik untuk setiap blok) dikaitkan dengan blok/rows.
  - Akses dilakukan dengan mencari index blok/ rows
  - Bandingkan tag dari alamat dan tag yang disimpan pada row tersebut
  - Jika tag sama => HIT, ambil byte (offset)
  - Jika tidak sama => MISS, ambil blok dari memori.

Akses Memori pada Direct-mapped Cache



Replacement Algorithms

- Berlaku bagi:
  - Associative Mapping
  - Set-Associative Mapping
- Bagaimana menempatkan blok baru jika cache sudah penuh → berpengaruh pada kinerja!
- Beberapa algoritme:
  - LRU: Least Recently Used
  - LFU: Least Frequently Used
  - Random