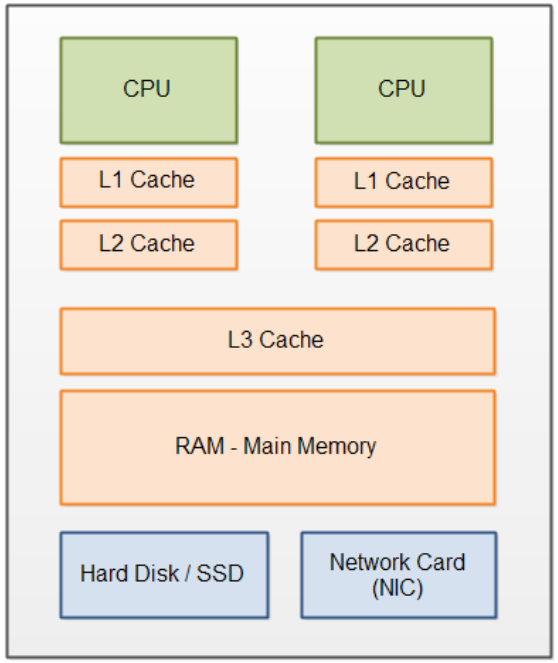
Datové struktury 1

5. přednáška

Pavel Veselý

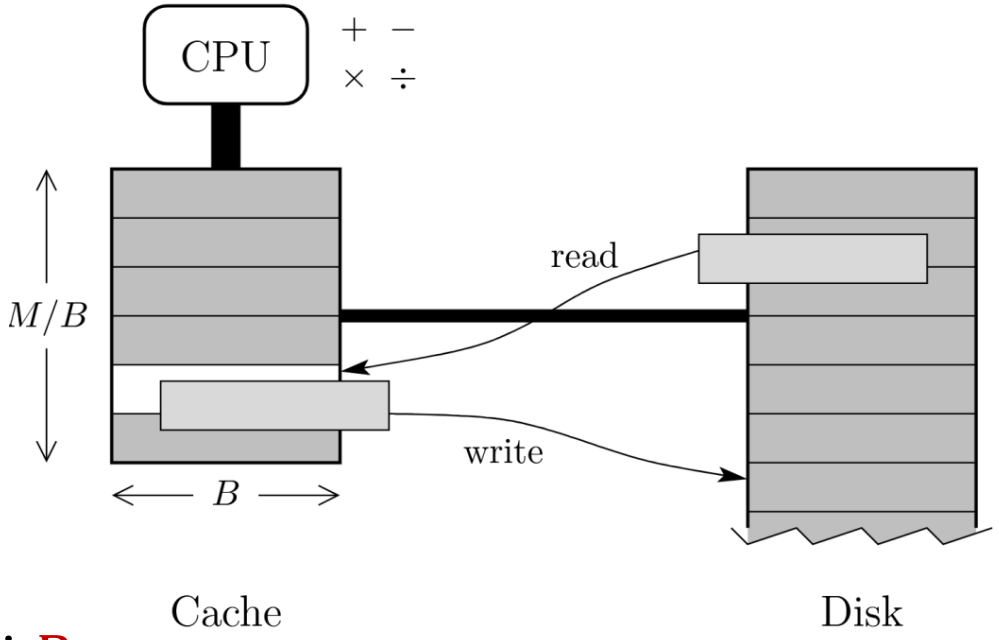
Plán

- cache-oblivious transpozice
- Online algoritmy pro správu cache
  - Aneb: co když neznáte budoucnost?
- Hešování!!!



I/O složitost pro dvě úrovně

- Pomalá paměť: disk, externí paměť
  - Složena z bloků velikosti B
- Rychlá paměť: cache, interní paměť
  - Velikost M, také rozdělena na bloky velikosti B
- I/O složitost: # přesených bloků
  - Příklad sekvenční průchod polem délky N
    - I/O složitost $O(N/B + 1)$

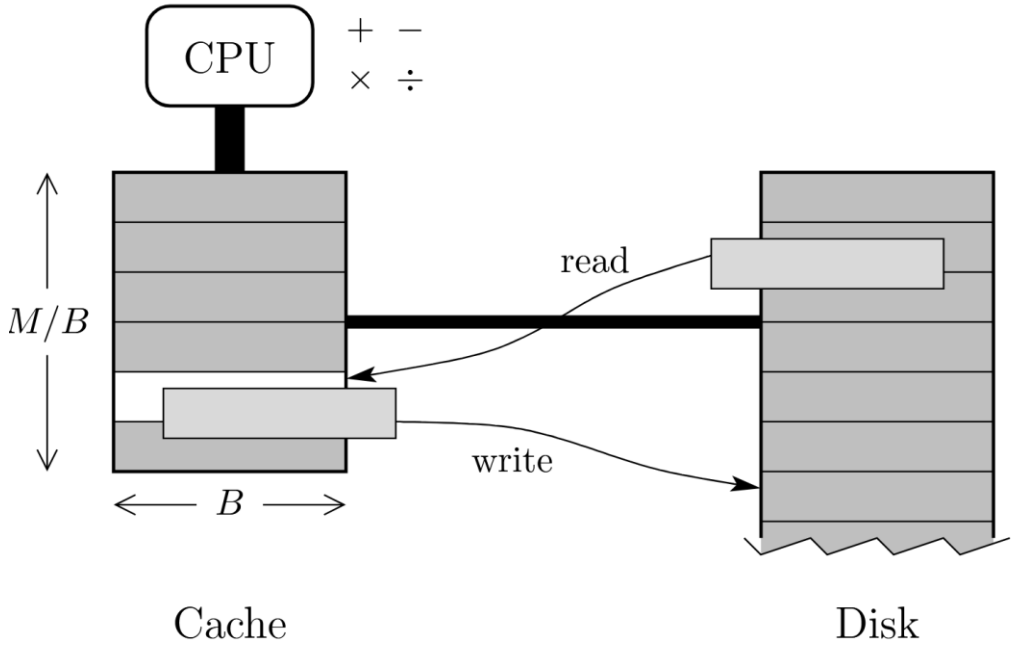


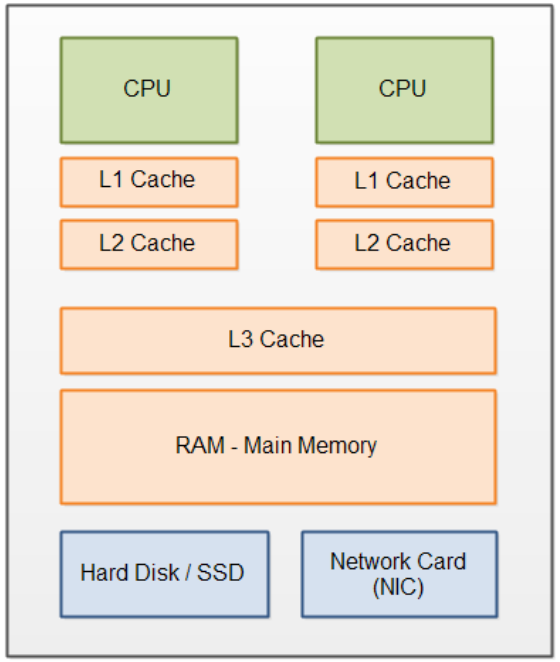
Zdroj obrázku: Demaine: Cache-Oblivious Algorithms and Data Structures

Teoretické modely

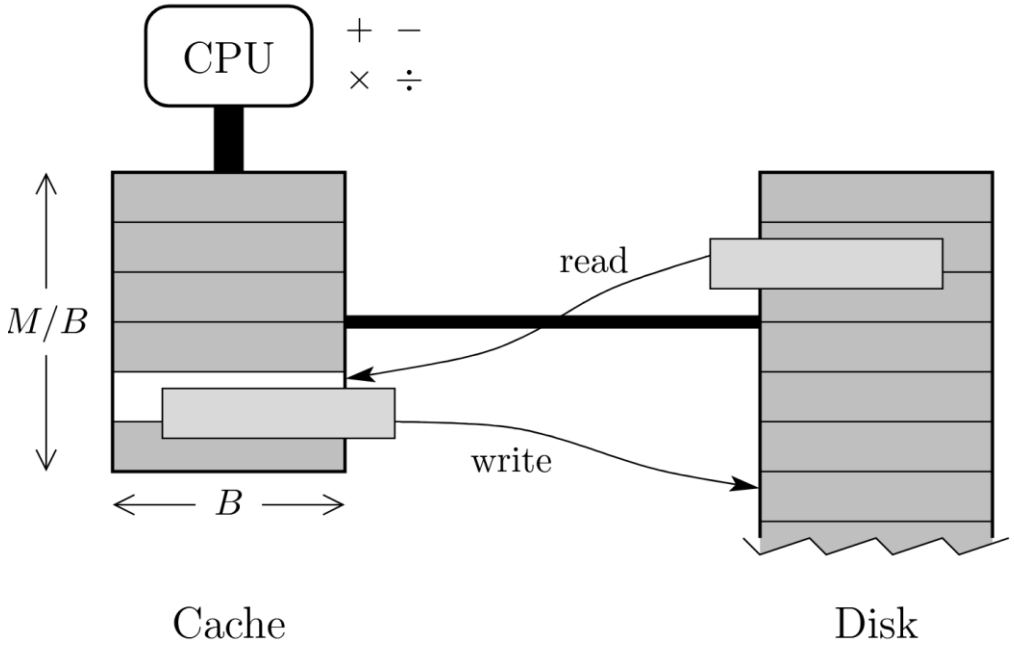
- I/O model
  a **cache-aware** algoritmy a DS
  - Známe parametry M a B
- **Cache-oblivious** algoritmy a DS
  [Frigo, Leiserson, Prokop, Ramachandran, 1999]
  - Nepotřebují znát M a B

    → fungují na různých architekturách
    a efektivně využívají celou paměťovou hierarchii





Předpoklady modelu
(které lze odstranit)

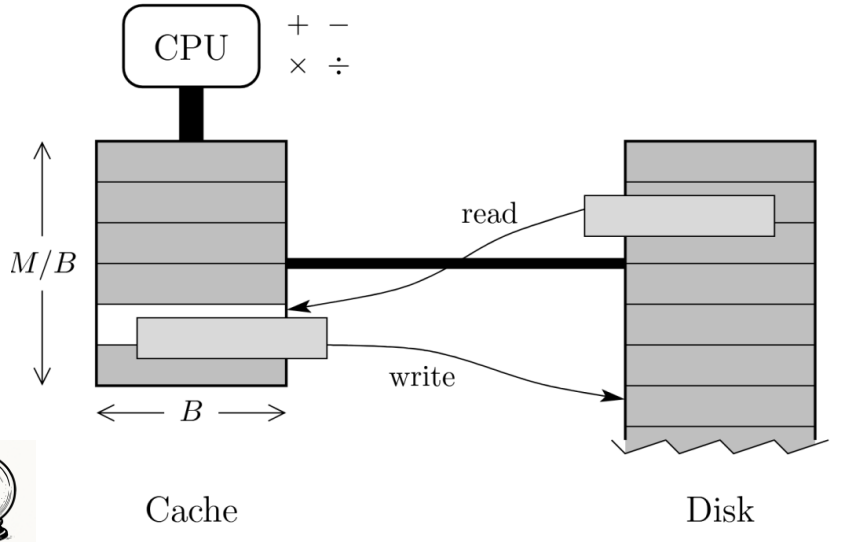


1. Cache řízena algoritmem (optimálně)
   - Algoritmus rozhoduje, která stránka je vyhozena z cache
   - Optimální algoritmus na správu cache:
     - *Longest Forward Distance (LFD)*: vyhodí blok, který bude potřeba nejpozději v budoucnu
   - Odstraníme pomocí *Least Recently Used* (LRU)
2. *Tall cache assumption:* $\frac{M}{B} \geq \Omega(B)$ … tedy # bloků je $\Omega(B)$
   - Není potřeba vždy
3. Plná asociativita: blok může být kdekoliv v cache

Matice $N \times N$

- Uložena po řádcích
  - →Průchod po řádcích má I/O složitost $O(N^2/B + 1)$
  - →Ale průchod po sloupcích $\Omega(N^2)$ (ledaže $M/B > N$)
- Transpozice
  - cache-aware – rozdělení na podmatice $B \times B$
    - transponujeme podmatice a prohazujeme přes diagonálu
    - I/O složitost $O(N^2/B + 1)$
  - cache-oblivious – rozděl a panuj!
    - I/O složitost $O(N^2/B + 1)$

Online správa cache



- Nemáme optimální algoritmus OPT pro správu cache
- Online algoritmus
  - rozhoduje se bez znalosti budoucí části vstupu
  - vs. klasický „offline“ algoritmus – zná celý vstup dopředu 
  - Online rozhodnutí: který blok vyhodit z cache?
- *Least Recently Used* (LRU)
  - odstraní z cache blok, ke kterému bylo přistoupeno nejdále v minulosti
- Cena algoritmus = # výpadků, tedy načtení bloku (cache misses)

Definice: Algoritmus ALG je α-kompetitivní, pokud na libovolném vstupu platí:

$$\text{cena ALG} \leq \alpha \cdot (\text{cena OPT}) + \text{konstanta nezávislá na vstupu}$$

- Kompetitivní poměr ALG = $\inf\{ \alpha \mid \text{ALG je } \alpha\text{-kompetitivní} \}$

Hešování

- Prvkům z univerza $\mathcal{U}$ přiřadíme „náhodná“ čísla
  - Nebo alespoň „náhodně vypadající“

Hešovací funkce: $h: \mathcal{U} \to [m]$

- Příklad:
  - $h(x) = a \cdot x \mod m$, kde a je nesoudělné s m

Aplikace:

- **Hešovací tabulka**
- Vzorkování
- Hledání duplikátů
- Vyhledávání v textu

- **Bloom filtry**
- Datové skeče
  - technika MinHash
  - CountSketch
  - …

Hešovací tabulka

- Reprezentace množiny klíčů (nebo slovníku)
  - Operace `Find, Insert, Delete`
- Tabulka $\approx$ pole přihrádek velikosti m
- Kolize = dva prvky skončí ve stejné přihrádce

Hlavní části hešovací tabulky:

1. Hešovací funkce
2. Řešení kolizí
   - Separované řetězce – spojový seznam (či dynamické pole) v každé buňce
   - Otevřená adresace – v přihrádce max. jeden klíč (přespříště)
   - Kukaččí hešování (přespříště)
3. Dynamizace: udržujeme určité maximální (a minimální) zaplnění
   - Přehešujeme do nové tabulky – podobně jako dynamické pole

Hešovací tabulka se separovanými řetězci

- Prozatím: plně náhodná hešovací funkce
  - každý prvek z $\mathcal{U}$ zahešován uniformně a nezávisle na ostatních
- Find(x): projdeme řetězec v přihrádce $h(x)$
- Insert(x): vložíme na začátek řetězce $h(x)$
  - ale musíme ověřit, jestli už tam není
- Delete(x): projdeme řetězec na $h(x)$
- Vše tedy v $O(\text{délka řetězce na } h(x))$

Hešovací funkce

- Plně náhodná (každý prvek z $\mathcal{U}$ zahešován uniformně a nezávisle na ostatních)
  - Potřebuje prostor $\Omega(\mathcal{U})$ na uložení ☹
- Pevná funkce
  - Lze na ní zaútočit – způsobit velký počet kolizí ☹
- Náhodný výběr z množiny funkcí
  - Např. funkce má několik parametrů volených náhodně
  - Cíle:
    - Malá pravděpodobnost kolizí
    - Dvojice prvků se hešují nezávisle

Příště

- Hešování ☺
  - Hešovací funkce univerzální, *k*-nezávislé, …