Architektura počítačů
Paměťová hierarchie

http://d3s.mff.cuni.cz/teaching/computer_architecture/

Lubomír Bulej

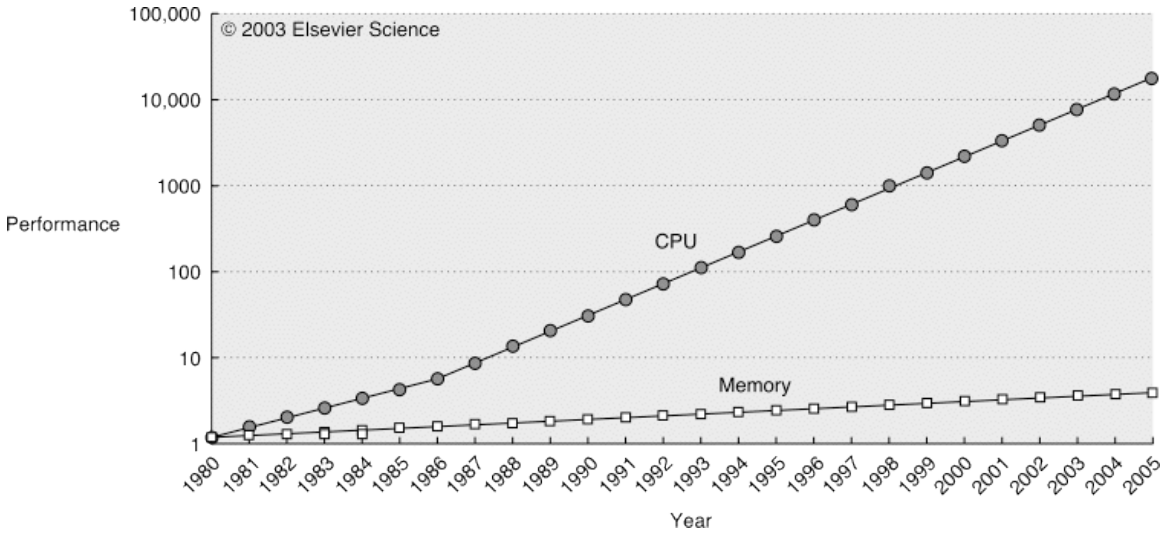
bulej@d3s.mff.cuni.cz

CHARLES UNIVERSITY IN PRAGUE

faculty of mathematics and physics

Paměťová zeď

- **Výkon procesorů omezen výkonem pamětí**
  - Výkon procesorů roste rychleji než výkon pamětí
    - Jednoduché operace trvají desetiny ns, přístup do paměti trvá desítky ns
    - Nedosažitelný cíl – kombinace: Paměť stejně rychlá jako procesor, dostatečná kapacita, rozumná cena



Paměťová zeď (2)

- **Burks, Goldstine, von Neumann: *Preliminary discussion of the logical design of an electronic computing instrument* (1946)**
  - *„Ideally, one would desire an **infinitely large** memory capacity such that any particular word would be **immmediately available** [...] We are forced to recognize the possibility of constructing a **hierarchy of memories**, each of which has a **greater capacity** than the preceding but which is **less quickly** accessible.“*

Paměťová zeď (3)

- **Analogie s knihami a knihovnou**
  - Knihovna
    - Spousta knih, ale přístup k nim je pomalý (cesta do knihovny)
    - Velikost knihovny (nějakou dobu trvá najít správnou knihu)
  - Jak se vyhnout vysoké latenci?
    - Půjčit si nějaké knihy domů
      - Mohou ležet na polici nebo pracovním stole
        - Často používané knihy mohou být po ruce (časová lokalita)
        - Půjčíme si více knih na podobné téma (prostorová lokalita)
        - Odhadneme, co budeme potřebovat příště *(prefetching)*
      - Police a stůl mají omezenou kapacitu

Paměťová zeď (4)

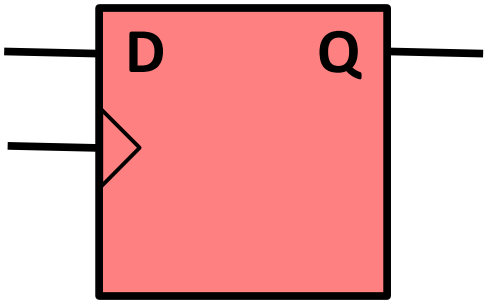
- **Jak překonat paměťovou zeď**
  - Princip lokality přístupu do paměti
    - Vlastnost většiny reálných programů (instrukce i data)
    - Časová lokalita *(temporal locality)*
      - K nedávno použitým datům budeme velmi pravděpodobně přistupovat znovu → Nedávná data uložíme v malé, velmi rychlé paměti
    - Prostorová lokalita *(spatial locality)*
      - S velkou pravděpodobností budeme přistupovat k datům poblíž těch, ke kterým jsme přistupovali nedávno → Přistupovat k datům po větších blocích (zahrnující i okolní data)

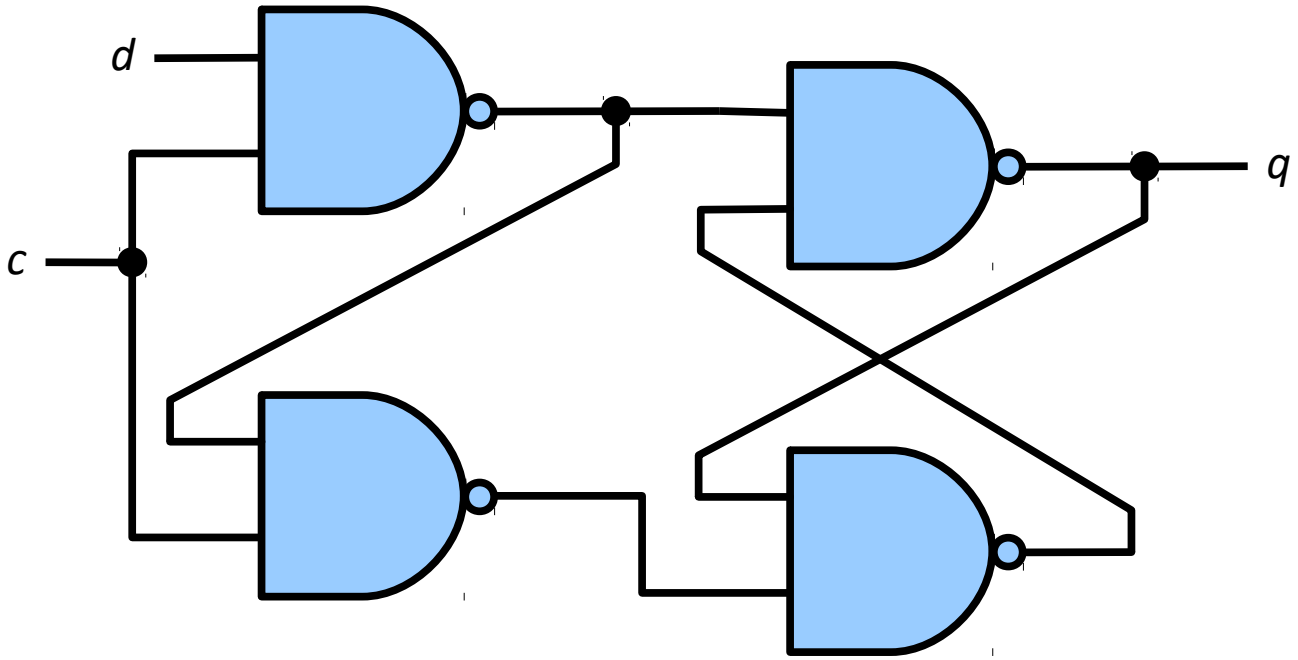
Volatilní paměti

- **Statická RAM**
  - Primární cíl: Rychlost
  - Sekundární cíl: Kapacita
    - 6 tranzistorů na bit, rychlost závisí na ploše (pro malé kapacity latence < 1 ns)
  - Dobře se kombinuje s ostatní procesorovou logikou
  - Obsah není nutné periodicky obnovovat

Statická RAM

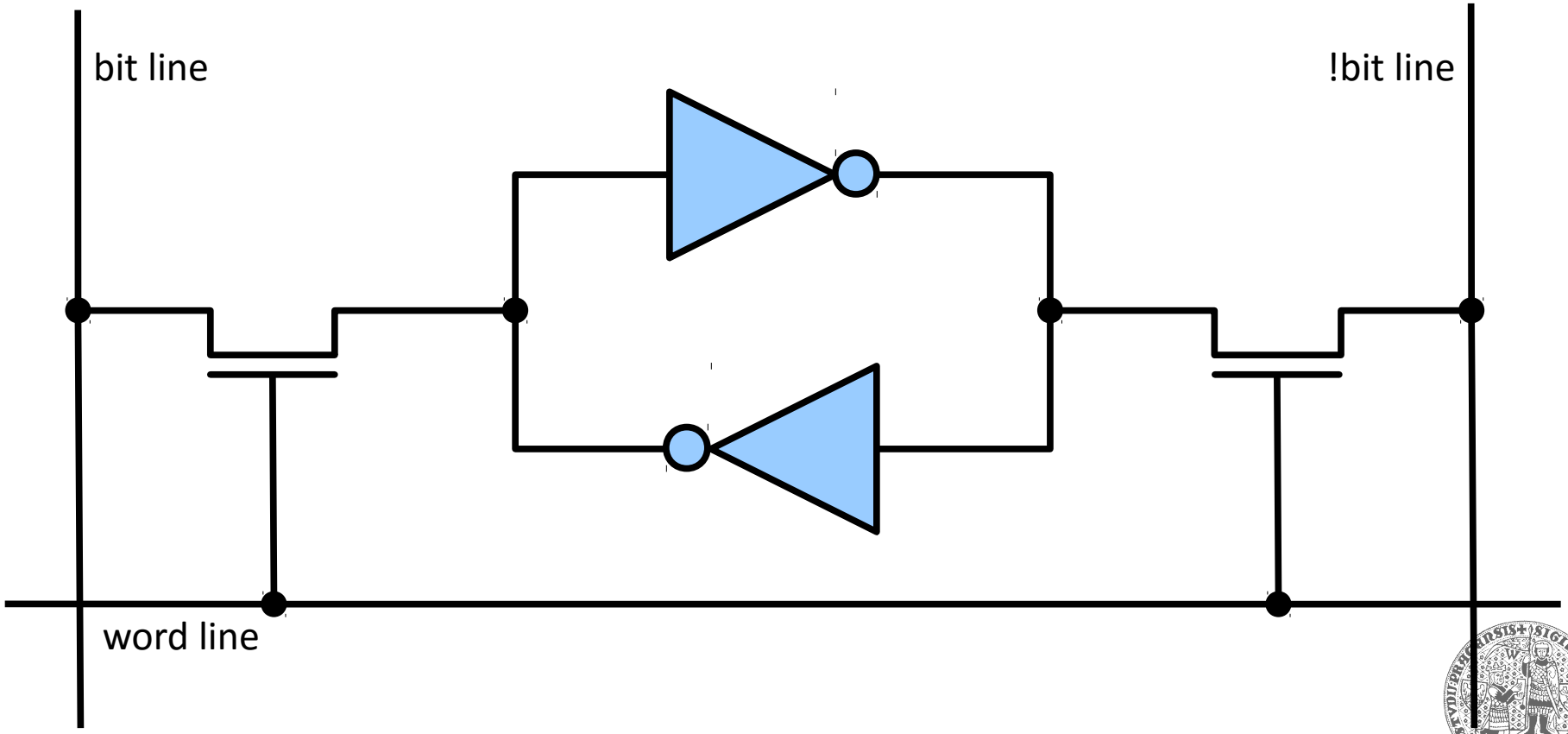
- **Klopný obvod typu D**
  - 1 bit, ~ 4 hradla, ~ 9 tranzistorů





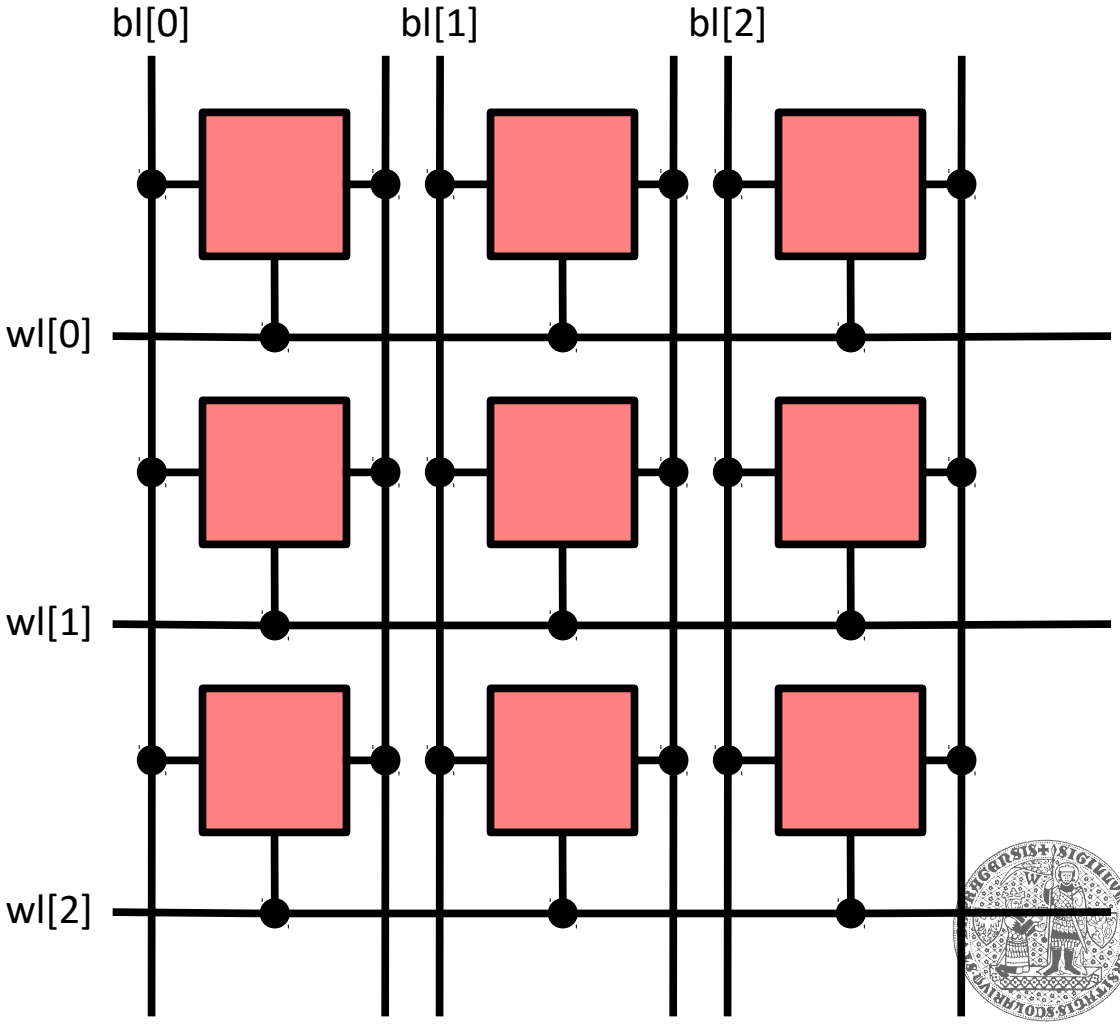
Buňka statické RAM

- **Dvojice invertorů a řídící tranzistory**
  - 6 tranzistorů na 1 buňku

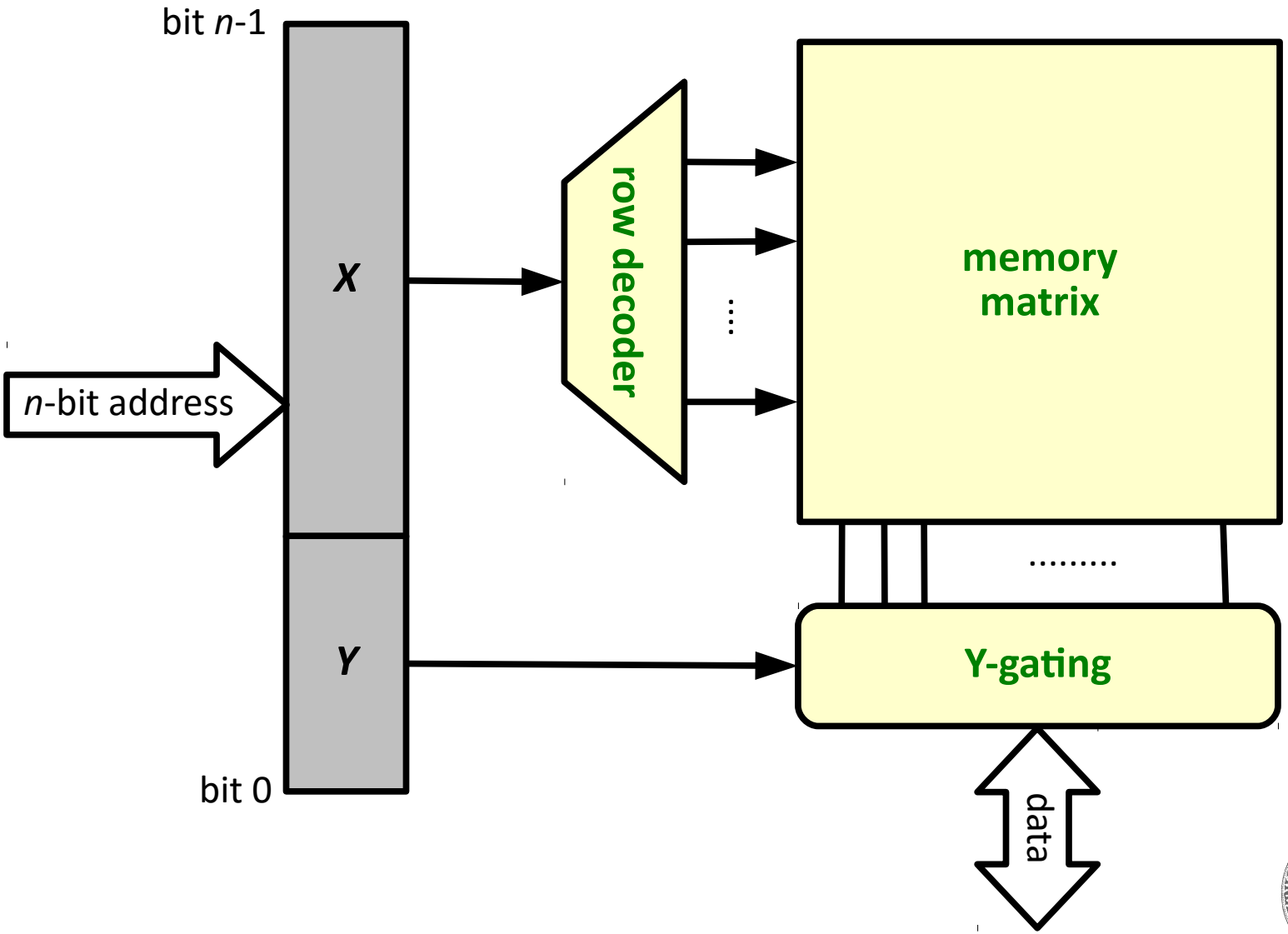


Statická RAM v maticovém uspořádání

- ***M* × *N* bitů: *M* řádků po *N* bitech**
  - Výběr řádku
    - Dekodér 1 z *M*
  - Přístup ve dvou krocích
    1. Výběr řádku *(word lines)*
    2. Čtení sloupců *(bit lines)*



Statická RAM (2)

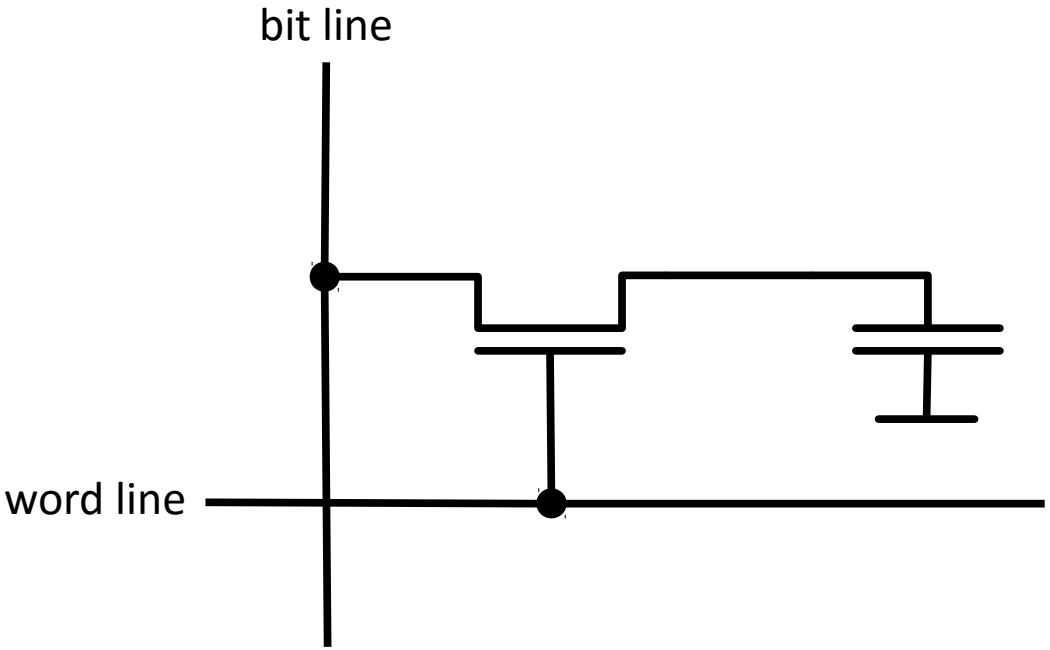


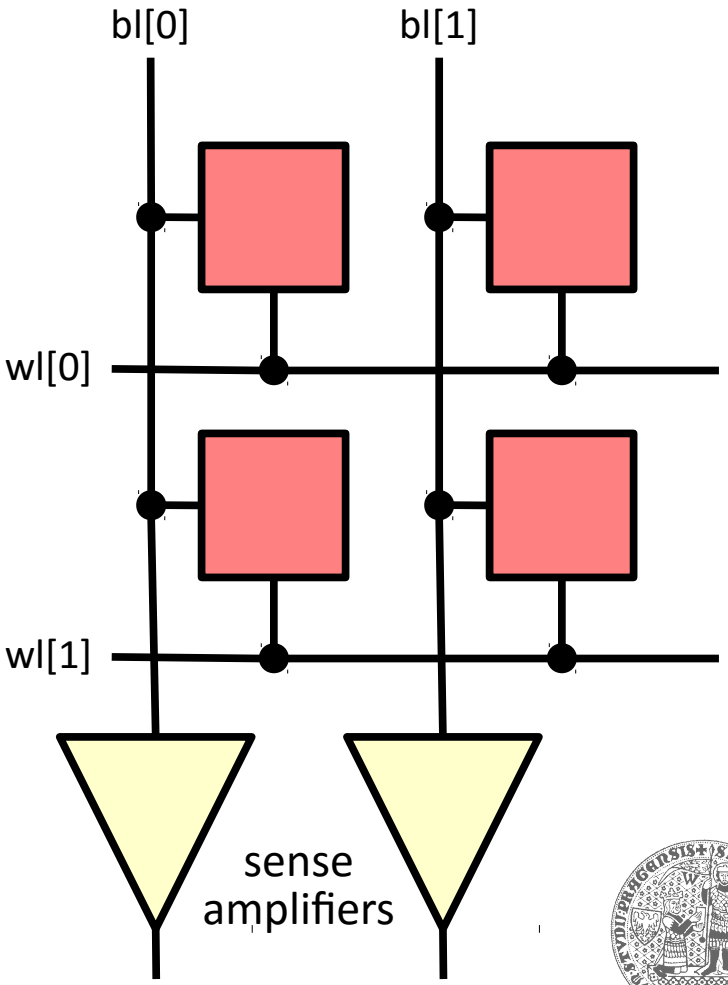
Volatilní paměti (2)

- **Dynamická RAM**
  - Primární cíl: Hustota (cena za bit)
    - 1 tranzistor a 1 kondenzátor na bit
    - Vysoká latence
      - 40 ns uvnitř samotné paměti
      - 100 ns mezi obvody
  - Obsah je nutné periodicky občerstvovat (číst a znovu zapisovat)
  - Nelze snadno kombinovat s logikou procesoru
    - Jiný výrobní postup

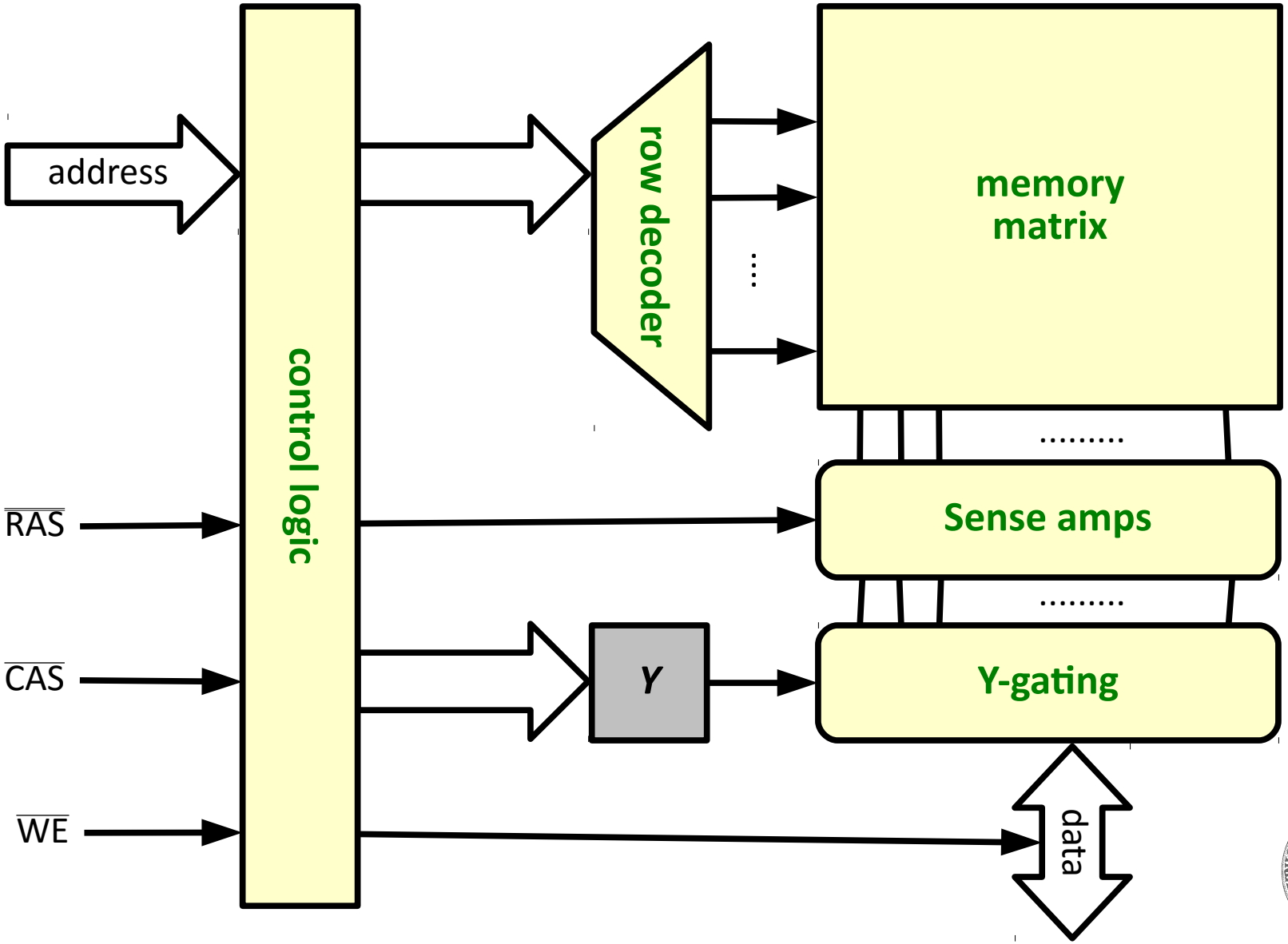
Buňka dynamické RAM

- **Kondenzátor a řídící tranzistor**
  - Informace uložena ve formě elektrického náboje
    - Kondenzátor se samovolně vybíjí/nabíjí v důsledku ztrát a obsahu sousedních buněk
  - Čtení je destruktivní (přečtená hodnota se ihned zapisuje zpět)





Dynamická RAM

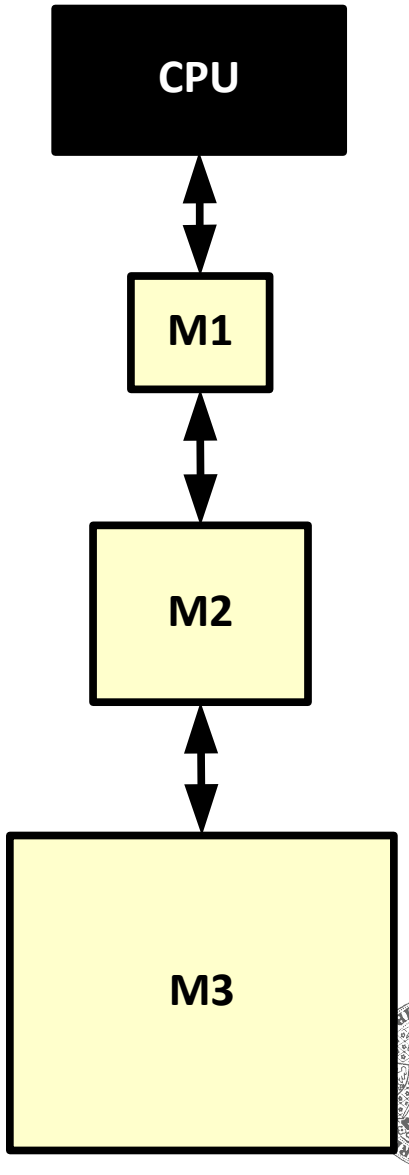


Zvyšování výkonu DRAM

- **Pozorování**
  - Nejdéle trvá čtení řádku DRAM
  - Řádek obsahuje více dat než jen požadované slovo
  - Amortizace ceny čtení řádku
    - Použít více slov z jednoho přečteného řádku
    - Pipelining výstupu dat a výběru nového řádku
      - Přečtený řádek uložen do výstupního registru, zahájení čtení dat z jiného řádku současně s přenosem dat z paměti do procesoru po sběrnici

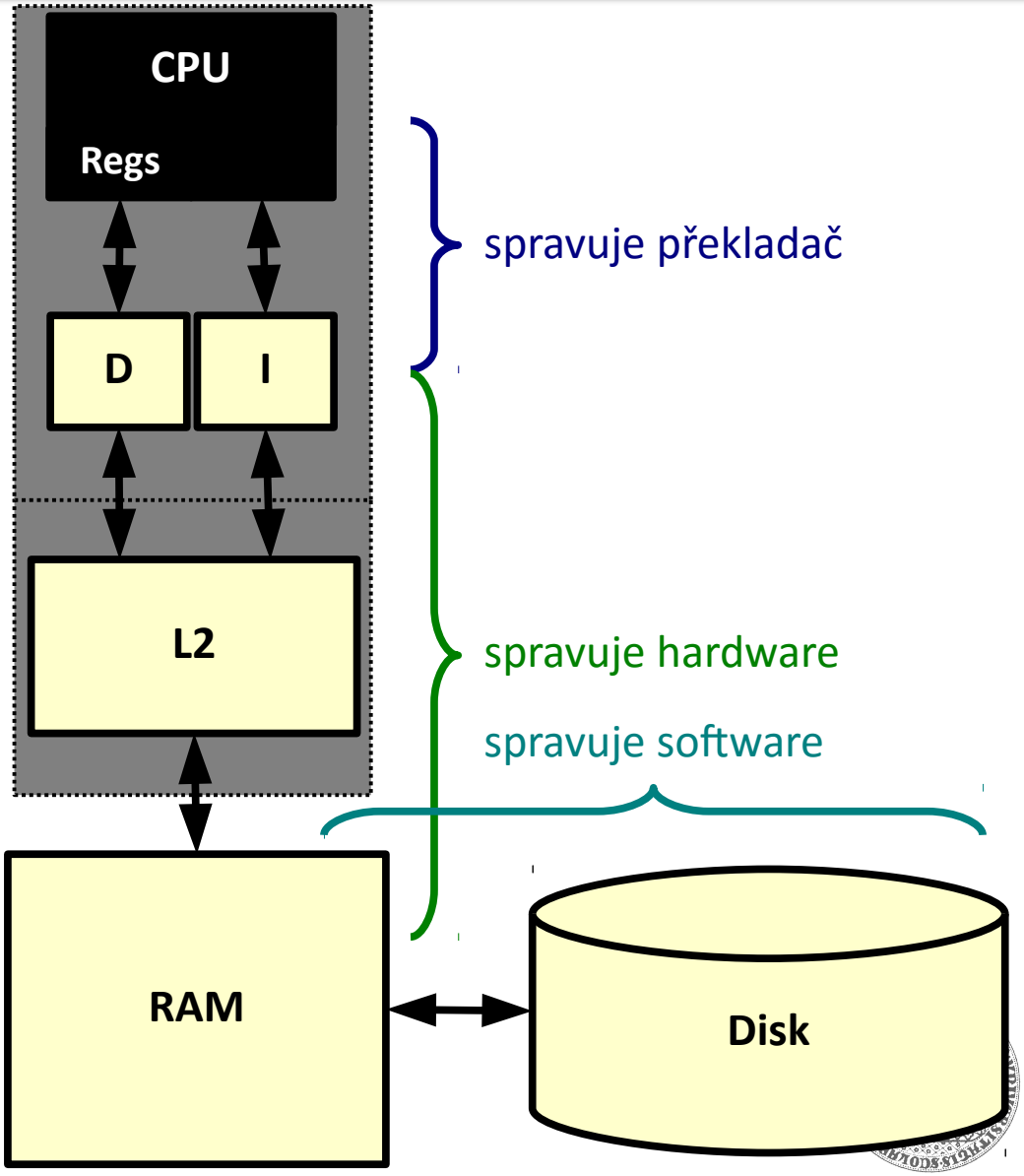
Využití lokality přístupu

- **Hierarchie paměťových komponent**
  - Vyšší vrstvy: Rychlé, malé, drahé
  - Nižší vrstvy: Pomalé, velké, levné
  - Vzájemné propojení sběrnicemi
    - Přidávají latenci, omezují propustnost
  - Nejčastěji používaná data v M1
    - Druhá nejpoužívanější data v M2 atd.
    - Přesun dat mezi vrstvami
  - Optimalizace průměrné doby přístupu
    - $Lat_{avg} = Lat_{hit} + Lat_{miss} \times \%_{miss}$



Hierarchie paměťových komponent

- **M0: Registry**
  - Data pro instrukce
- **M1: Primární cache**
  - Oddělená instrukční a datová
  - SRAM (kB)
- **M2: Sekundární cache**
  - Nejlépe na čipu, určitě v pouzdře
  - SRAM (MB)
- **M3: Operační paměť**
  - SRAM (kB—MB, embedded zařízení)
  - DRAM (GB)
- **M4: Odkládací paměť**
  - Soubory, swap
  - HDD, flash (TB)



Hierarchie paměťových komponent (2)

- **Analogie s knihovnou**
  - Registry ↔ aktuálně otevřená stránka v knize
    - Jen jedna stránka
  - Primární cache ↔ knihy na stole
    - Aktivně využívány, velmi rychlý přístup, malá kapacita
  - Sekundární cache ↔ knihy na polici
    - Aktivně využívány, poměrně rychlý přístup, střední kapacita
  - Operační paměť ↔ knihovna
    - Skoro veškerá data, pomalý přístup, velká kapacita
  - Odkládací paměť ↔ meziknihovní výpůjčka
    - Velmi pomalé, ale také velmi málo časté

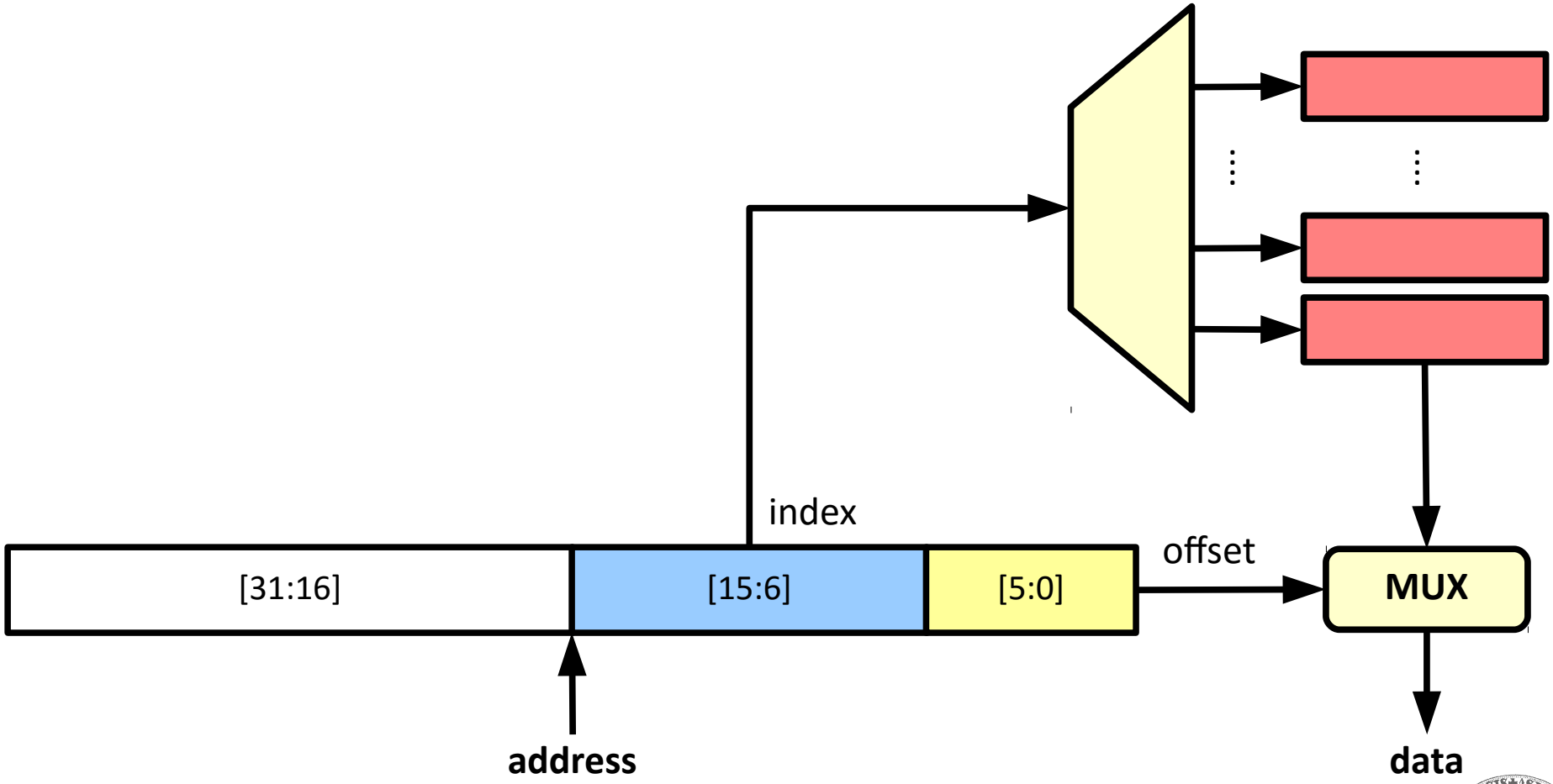
Cache

- **Iluze velké a rychlé paměti**
  - Přesun dat mezi vrstvami cache řídí hardware
    - Automatické nalezení chybějících dat
      - Řadič cache *(cache controller)*
    - SRAM, integrovaná na čipu
    - Software může dávat nápovědy
  - Organizace cache (ABC)
    - Asociativita, velikost bloku, kapacita
    - Klasifikace výpadků cache

Přímé mapování paměti do cache

- **Základní struktura**
  - Pole řádek *(cache lines)*
    - Např. 1024 řádek po 64 B → 64 KB
  - „Hardwarová hashovací tabulka“ podle adresy
    - Např. 32bitové adresy
      - 64bajtové bloky → spodních 6 bitů adresuje bajt v bloku *(offset bits)*
      - 1024 bloků → dalších 10 bitů představuje číslo bloku *(index bits)*
      - V principu lze použít i jiné bity (není příliš časté)

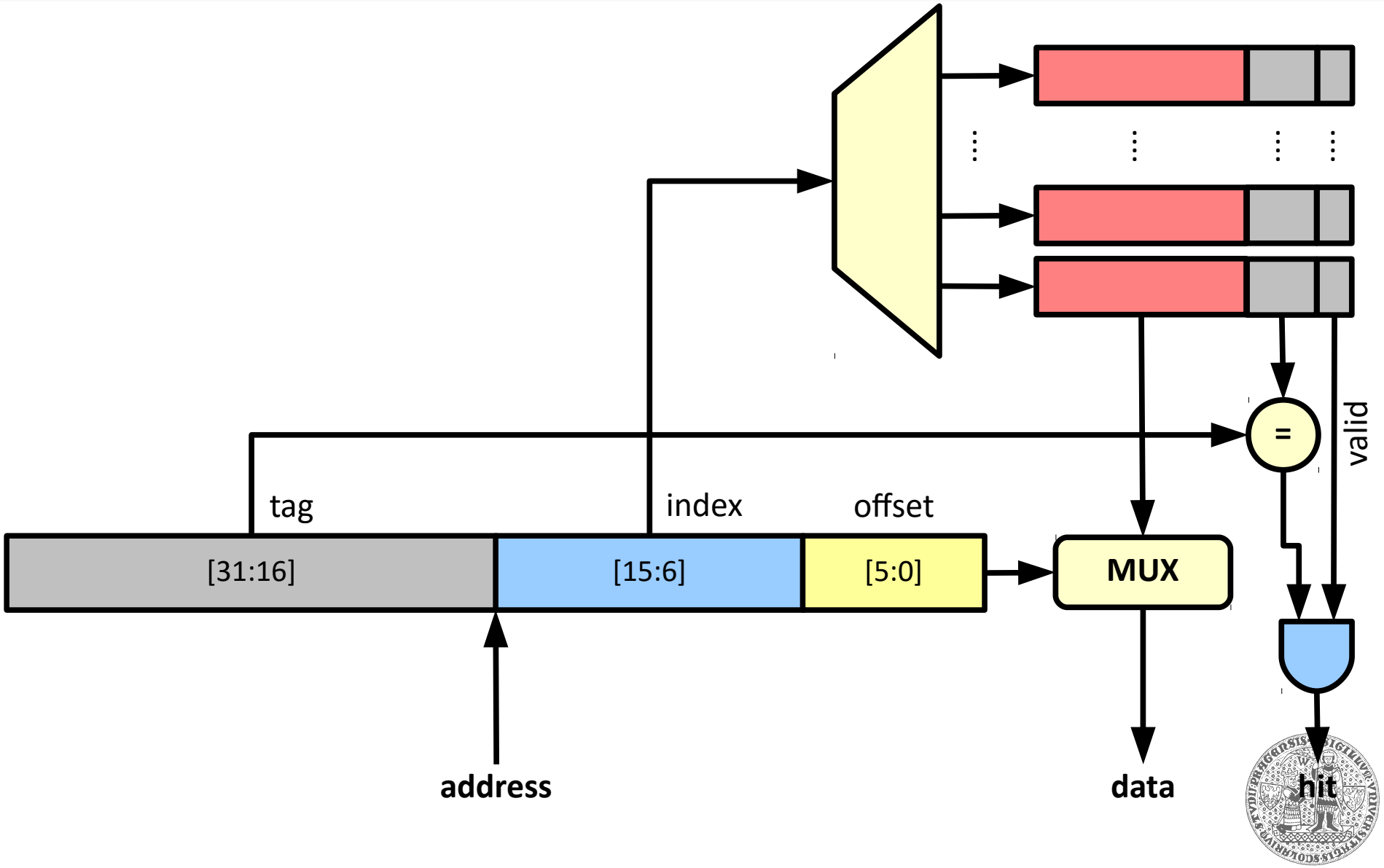
Přímé mapování paměti do cache (2)



Přímé mapování paměti do cache (3)

- **Nalezení správných dat**
  - V každém řádku cache může být jeden z 2^{16} bloků operační paměti
    - Kolize „hashovací funkce“
    - Detekce správných dat
      - Příznak platnosti řádky *(valid bit)*
      - Tag se zbývajícími bity adresy *(tag bits)*
  - **Algoritmus**
    1. Přečteme řádek určený indexem
    2. Pokud je nastaven *valid bit* a tag se shoduje s bity v adrese, jedná se o ***cache hit***
    3. Jinak se jedná o ***cache miss***

Přímé mapování paměti do cache (4)



Režie tagů a valid bitů

- **Pro 64 KB cache s 1024 řádky po 64 B**
  - Pro 32bitové adresy
    - (16 bitů na tag + 1 valid bit) × 1024 řádků ~ 2,1 KB
    - Režie 3,3 %
  - Pro 64bitové adresy
    - (48 bitů na tag + 1 valid bit) × 1024 řádků ~ 6,1 KB
    - Režie 9,6 %

Obsluha výpadku cache

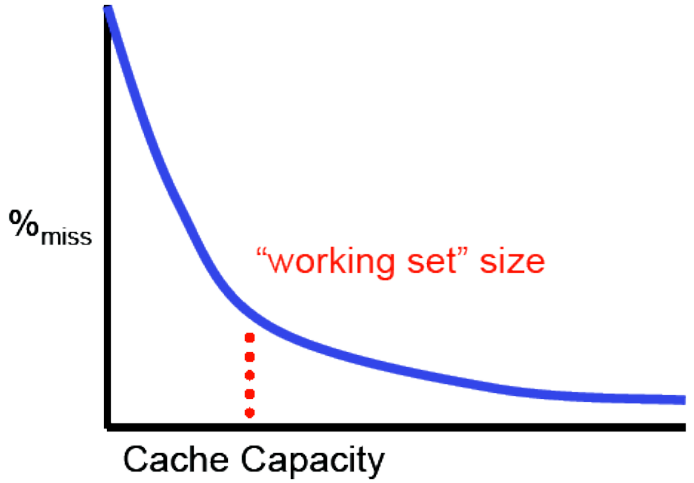
- **Naplnění dat do cache**
  - Řadič cache
    - Sekvenční obvod/stavový automat
    - Vyžádá si data z následující úrovně hierarchie na základě adresy výpadku
    - Zapíše data, tag a valid bit do řádku cache
  - Výpadky cache zpožďují pipeline
    - Analogie datového hazardu
    - Zpožďovací logika je řízena signálem *cache miss*

Výkonnost cache

- **Operace cache**
  - Přístup (čtení/zápis) do cache *(access)*
  - Nalezení požadovaných dat *(hit)*
  - Výpadek cache *(miss)*
  - Načtení dat do cache *(fill)*
- **Charakteristika cache**
  - $\%_{miss}$: Poměr výpadků a všech přístupů *(miss rate)*
  - t_{hit}: Doba přístupu do cache při hitu
  - t_{miss}: Doba potřebná k načtení dat do cache
- **Výkonnostní metrika: Průměrný čas přístupu**
  - $t_{avg} = t_{hit} + t_{miss} \times \%_{miss}$

Snížení miss rate

- **Přímá cesta: zvyšování kapacity**
  - Miss rate klesá monotonně
    - Zákon klesajících výnosů
  - t_{hit} roste s 2. odmocninou kapacity
- **Složitější cesta**
  - Při konstantní kapacitě
    - Změna organizace cache



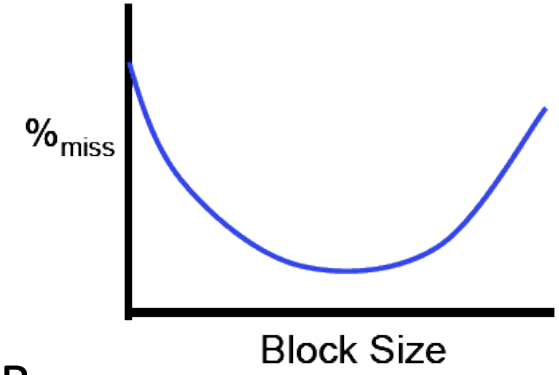
[1]

Organizace cache: Velikost řádku

- **Zvětšení velikosti řádku**
  - Předpoklad: Využití prostorové lokality
  - Změna poměru indexových/offsetových bitů (velikost tagu se nemění)
  - **Důsledky**
    - Snížení miss rate (do určité míry)
    - Nižší režie na tagy
    - Více potenciálně zbytečných přenosů dat
    - Předčasná náhrada užitečných dat

Vliv velikosti řádku na miss rate

- **Načítání okolních dat *(spatial prefetching)***
  - Pro bloky se sousedními adresami mění miss/miss na miss/hit
- **Rušení *(interference)***
  - Pro bloky s nesousedními adresami v sousedních řádcích cache mění hit na miss
    - Znemožňuje současný výskyt v cache
- **Vždy oba efekty**
  - Ze začátku dominuje pozitivní efekt
  - Limitní případ: Cache s jedním řádkem
  - Obvyklá rozumná velikost řádku: 16 – 128 B



[1]

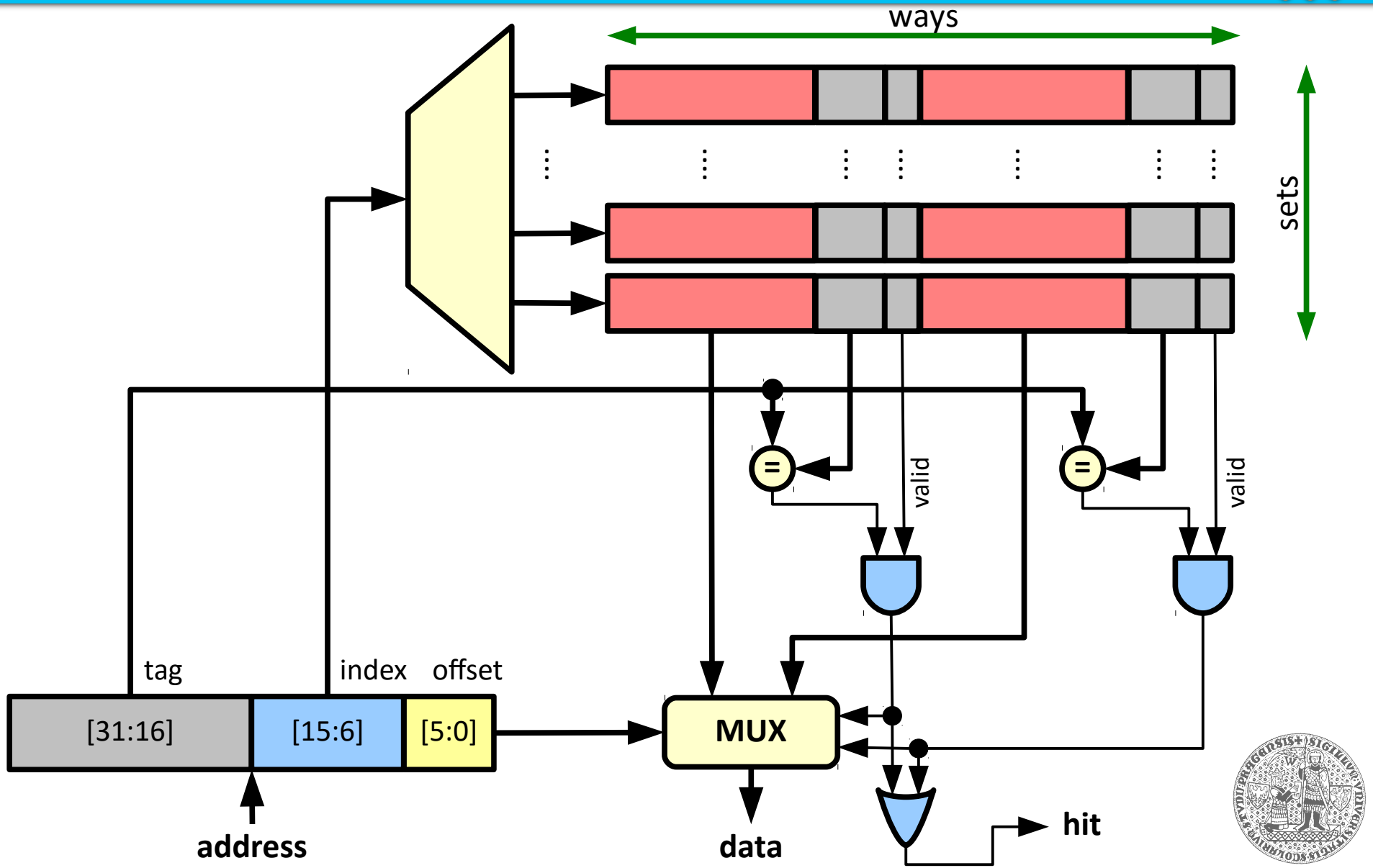
Vliv velikosti řádku na dobu plnění cache

- **V principu**
  - Přečtení větších řádků by vždy mělo trvat déle
- **V praxi**
  - Pro izolované výpadky se t_{miss} nemění
    - *Critical Word First / Early Restart*
      - Z paměti se nejprve čte právě požadované slovo (pro minimalizaci zpoždění pipeline procesoru)
      - Ostatní slova řádku cache se dočítají následně
  - V případě skupin výpadků se t_{miss} zvyšuje
    - Nelze číst/přenášet/doplňovat více řádků současně
    - Hromadění zpoždění
    - Omezená přenosová kapacity mezi pamětí a procesorem

Asociativní mapování paměti do cache

- **Množinová asociativita *(set associativity)***
  - Skupiny řádků = množiny, řádek v množině = cesta
    - Např.: 2-cestná množinově-asociativní cache *(2-way set-associative)*
    - Limitní případy
      - Pouze jedna cesta: Přímo mapovaná cache
      - Pouze jedna množina: Plně asociativní cache
  - **Cíl:** Omezení konfliktů
    - Blok paměti může být ve různých řádcích jedné množiny
  - Prodlužuje t_{hit}
    - Výběr dat z řádků v množině

Asociativní mapování paměti do cache (2)

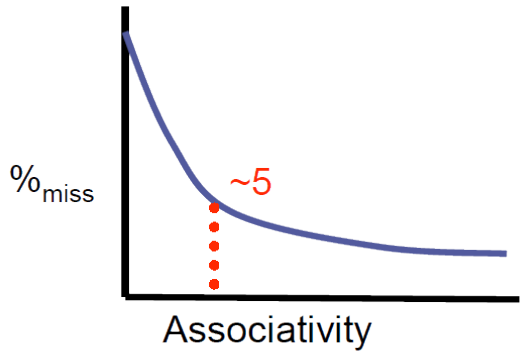


Asociativní mapování paměti do cache (3)

- **Algoritmus**
  1. Pomocí index bitů adresy *(set)* najdeme množinu řádků
  2. Přečteme **současně** všechna data a tagy ze všech řádků *(ways)* v množině
  3. Porovnáme **současně** tagy řádků s tagem z adresy
  - Vliv na tag/index bity
    - Více cest → méně množin
    - Více tag bitů

Vliv asociativity na miss rate

- **Vyšší stupeň asociativity**
  - Snižuje miss rate
    - Zákon klesajících výnosů
  - Prodlužuje t_{hit}



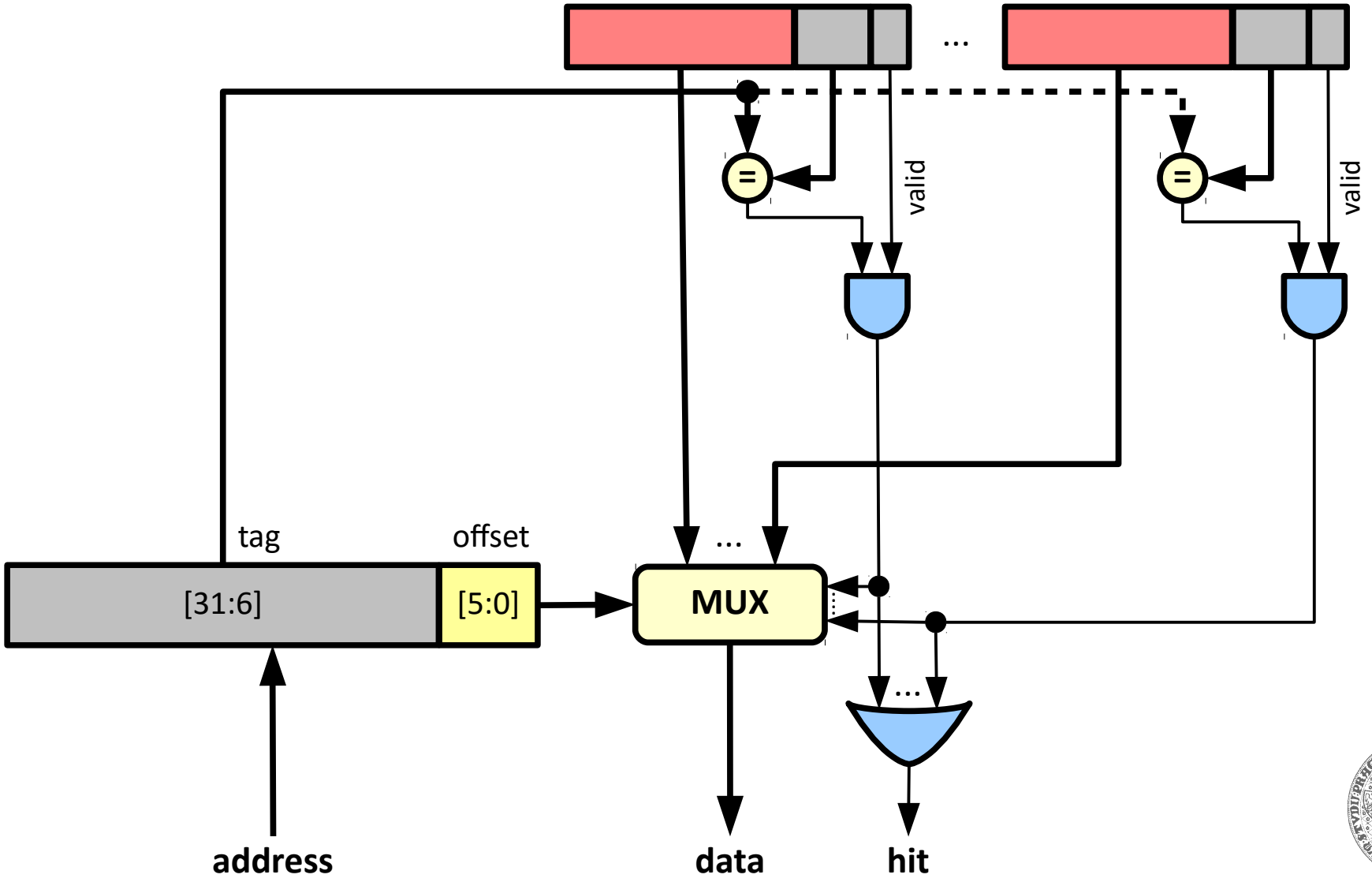
[1]

  - Dává smysl mít *n*-cestnou asociativitu, kde *n* není mocnina dvojky
    - Ničemu to nevadí, i když to není obvyklé
    - Velikost řádku a počet množin by měly být mocninou dvojky
      - Zjednodušuje to indexaci (lze jednoduše použít bity adresy)

Plně asociativní cache

- **Množinově-asociativní mapování s 1 množinou (počet cest roven počtu bloků)**
  - Blok paměti může být v libovolném bloku/řádku cache (cache line)
  - Všechny bity adresy (kromě offset bitů) představují tag
  - Asociativní paměť
    - Paměť adresovaná klíčem
    - Klíč = tag

Plně asociativní cache (2)



3C Model

- **Klasifikace výpadků cache**
  - ***Compulsory (cold) miss***
    - „Tuhle adresu jsem nikdy neviděl“
    - K výpadku by došlo i v nekonečně velké cache
  - ***Capacity miss***
    - Výpadek způsobený příliš malou kapacitou cache
      - Opakovaný přístup k bloku paměti oddělený alespoň *N* přístupy do *N* jiných bloků (kde *N* je počet řádků cache)
    - K výpadku by došlo i v plně asociativní cache
  - ***Conflict miss***
    - Výpadek způsobený příliš malým stupněm asociativity
    - Všechny ostatní výpadky

Miss rate: ABC

- **Důsledky 3C modelu**
  - Pokud nevznikají konflikty, zvýšení asociativity nepomůže
  - Asociativita *(**A**ssociativity)*
    - Snižuje počet konfliktních výpadků
    - Prodlužuje t_{hit}
  - Velikost bloku *(**B**lock size)*
    - Zvyšuje počet konfliktních/kapacitních výpadků (méně řádků)
    - Snižuje počet studených/kapacitních výpadků (prostorová lokalita)
    - Vesměs neovlivňuje t_{hit}
  - Kapacita *(**C**apacity)*
    - Snižuje počet kapacitních výpadků
    - Prodlužuje t_{hit}

Čtení dat z cache

- **Tag a (všechna) data lze číst současně**
  - Pokud tag nesouhlasí, data se nepoužijí (cache miss)
    - Vygeneruje se požadavek na doplnění (fill) z nižší vrstvy
- **Read miss: kam uložit data z nižší vstvy?**
  - Obsah některého řádků nutno nahradit novými daty
    - Původní obsah je „vyhozen“ (evicted)
  - Přímo mapovaná cache
    - Cílový řádek určen jednoznačně indexovými bity adresy
  - (Množinově) asociativní cache
    - Všechny cesty v množině jsou kandidáty, nutno vybrat „oběť“
    - Ideálně: nezahodit data, která budou brzy potřeba
      - Random
      - LRU *(Least Recently Used)*: Ideální vzhledem k časové lokalitě
      - NMRU *(Not Most Recently Used)*: Aproximace LRU

Zápis dat do cache (1)

- **Write hit**
  - Data zapsána do příslušného řádku cache
    - Při zápisu pouze do cache budou data v paměti a v cache nekonzistentní
  - Write through
    - Při každé operaci zápisu data uložena do cache i paměti/nižší vrstvy
    - Problém: operace s pamětí (a tedy instrukce zápisu) trvají příliš dlouho
    - Řešení: data zapsána do cache a **write bufferu**, odkud jsou zapsána do paměti
      - Procesor musí čekat pouze pokud je write buffer plný (Kdy k tomu může dojít?)
      - Při hledání dat v cache je nutné se podívat i do write bufferu
  - Write-back
    - Při operaci zápisu data uložena pouze do cache
      - Vyžaduje „dirty bit“ pro indikaci stavu řádku ve vztahu k paměti/nižší vrstvě
    - Modifikovaný (dirty) řádek zapsán do nižší úrovně až když je nahrazen
    - Zlepšuje výkon v situacích, kdy program generuje zápisy do paměti stejně rychle nebo rychleji, než je paměť schopna obsluhovat
    - Složitější na implementaci

Zápis dat do cache (2)

- **Write miss v případě write-through cache**
  - Je možné zároveň číst tag a zapisovat data
    - Pokud dojde k přepsání špatných dat, správná data jsou ještě v paměti
  - Write allocate
    - Nejprve se do cache doplní data z nižší vrstvy
    - Poté jako write hit: příslušná část řádku se přepíše zapisovanými daty
    - Může zabránit výpadkům při příštím přístupu (lokalita)
      - K tomu nemusí nutně dojit.
    - Vyžaduje dodatečnou přenosovou kapacitu
  - No write allocate
    - Data se zapisují pouze do nižší vrstvy/paměti
    - Eliminuje čtení z nižší vrstvy při write miss
      - Vhodné pro data, která procesorem pouze „prochází“
      - Např. nulování obsahu stránky, zápis bloku dat na disk, odeslání dat po síti...
  - Některé procesory umožňují nastavit strategii zápisu na úrovni jednotlivých stránek

Zápis dat do cache (3)

- **Write miss v případě write-back cache**
  - Není vždy možné zároveň číst tag a zapisovat data
    - Změněný řádek/cesta musí být nejprve zapsán do paměti/nižší úrovně
  - Zápis vyžaduje buď dva kroky...
    - kontrola hit/miss a poté vlastní zápis
  - … nebo použití **store bufferu**
    - kontrola hit/miss současně s „odložením“ dat do bufferu
    - při write hit data zapsána ze store bufferu do cache
  - Zápis modifikovaného řádku (při nahrazení)
    - Data nejprve přesunuta do **write-back bufferu**, později do paměti/nižší vrstvy
    - Při hledání dat v cache opět nutno prohledávat i write-back buffer

Víceúrovňové cache (1)

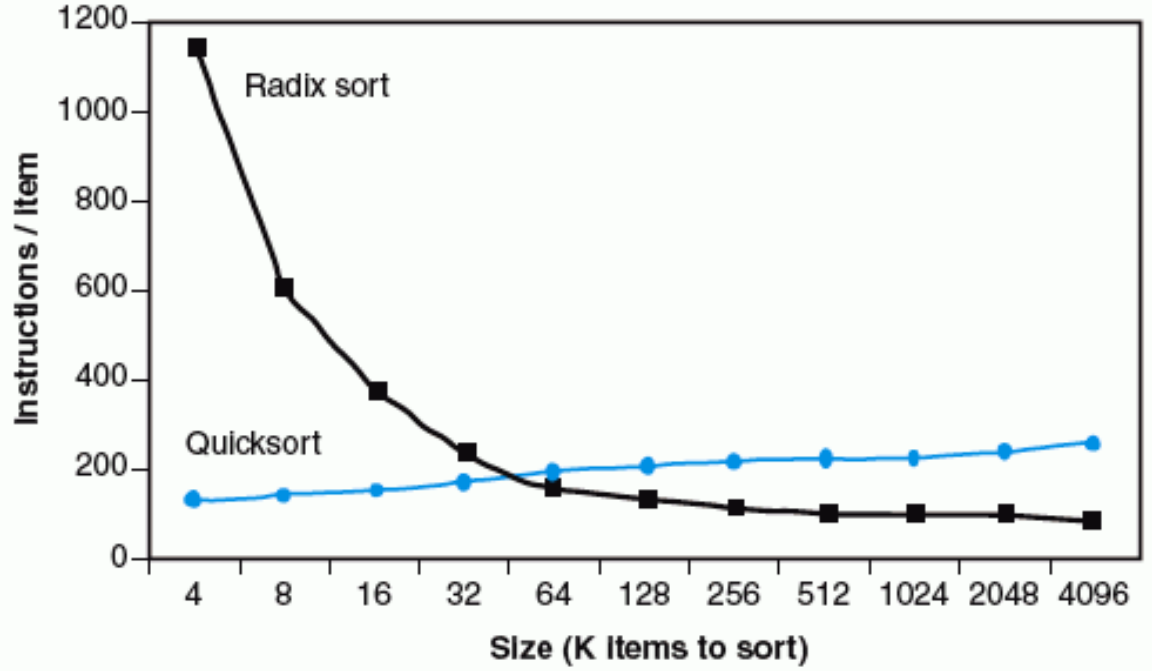
- **Cíl: snížení penalizace při výpadku**
  - 1-úrovňová cache:
    Total CPI = 1.0 + Memory stall cycles per instruction
    - 4 GHz procesor, přístup do paměti 100 ns (400 taktů), 2% výpadků:
      Total CPI = 1.0 + 2% x 400 = 9
  - 2-úrovňová cache:
    Total CPI = 1.0 + Primary stalls per instruction + Secondary stalls per instruction
    - Přístupová doba 5 ns (20 taktů) pro hit/miss, sníží celkový počet výpadků na 0.5%:
      Total CPI = 1.0 + 2% x 20 + 0.5% x 400 = 3.4

Víceúrovňové cache (2)

- **Různé úrovně cache mají různé role**
  - Umožňuje optimalizovat pro jiná (různá) kritéria než u jednoúrovňové cache
- **Primární cache**
  - Mimimalizace hit time
  - Umožňuje zvýšit taktovací frekvenci nebo snížit počet stupňů pipeline
  - Typicky menší kapacita, menší velikost řádků (nižší penalizace za cache miss)
- **Sekundární cache**
  - Mimimalizace miss rate
  - Snižuje penalizaci za přístup do paměti
  - Výrazně vyšší kapacita (přístupová doba není kritická), větší velikost řádků, vyšší stupeň asociativity (důraz na snížení počtu výpadků)

Proč je důležité o cache vědět?

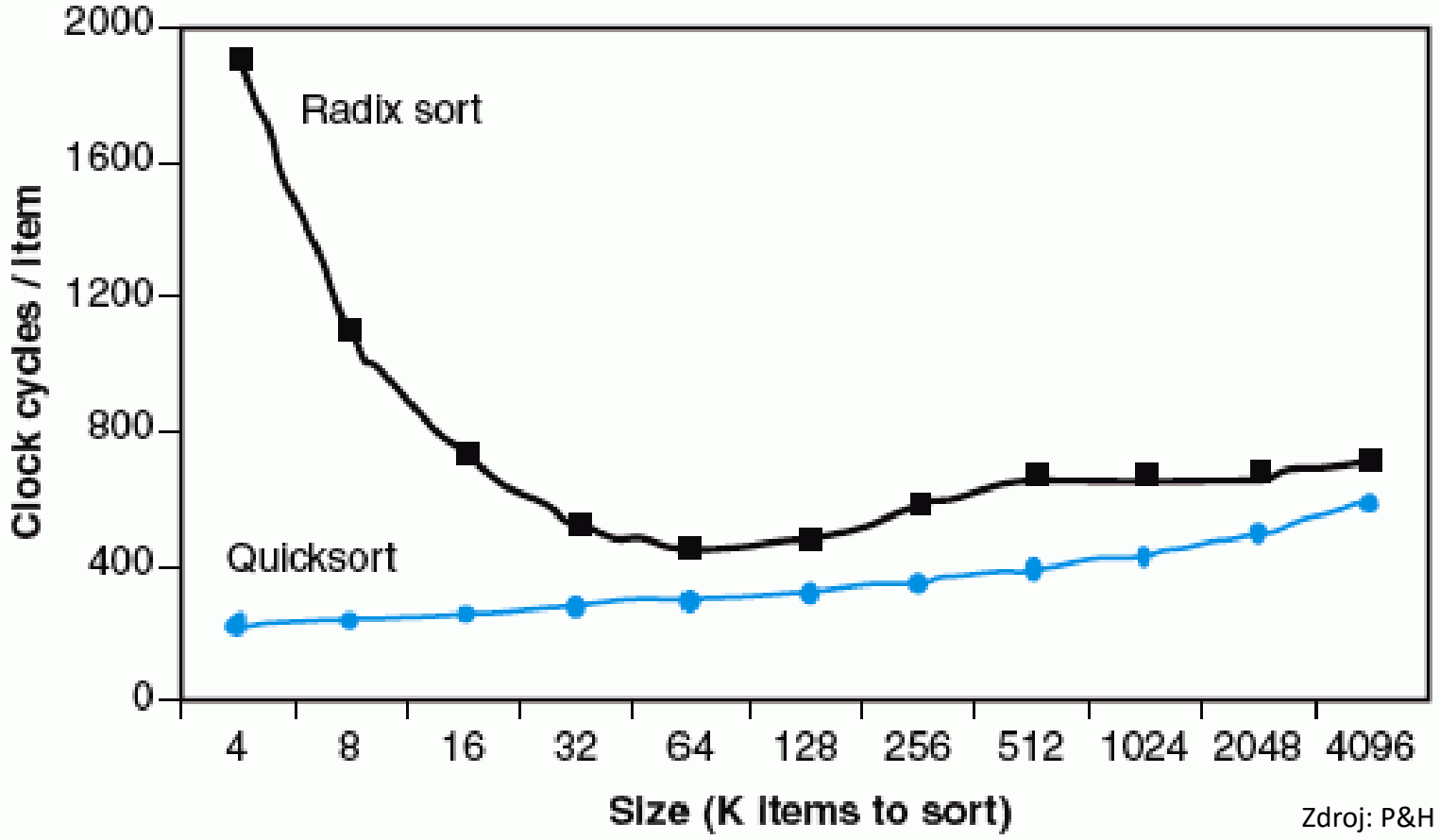
- ***Quick Sort* vs. *Radix Sort***
  - LaMarca, Ladner (1996)
  - **O(*n*×log *n*)** vs. **O(*n*)**
  - Teoreticky „není co řešit“



Zdroj: P&H

Proč je důležité o cache vědět? (2)

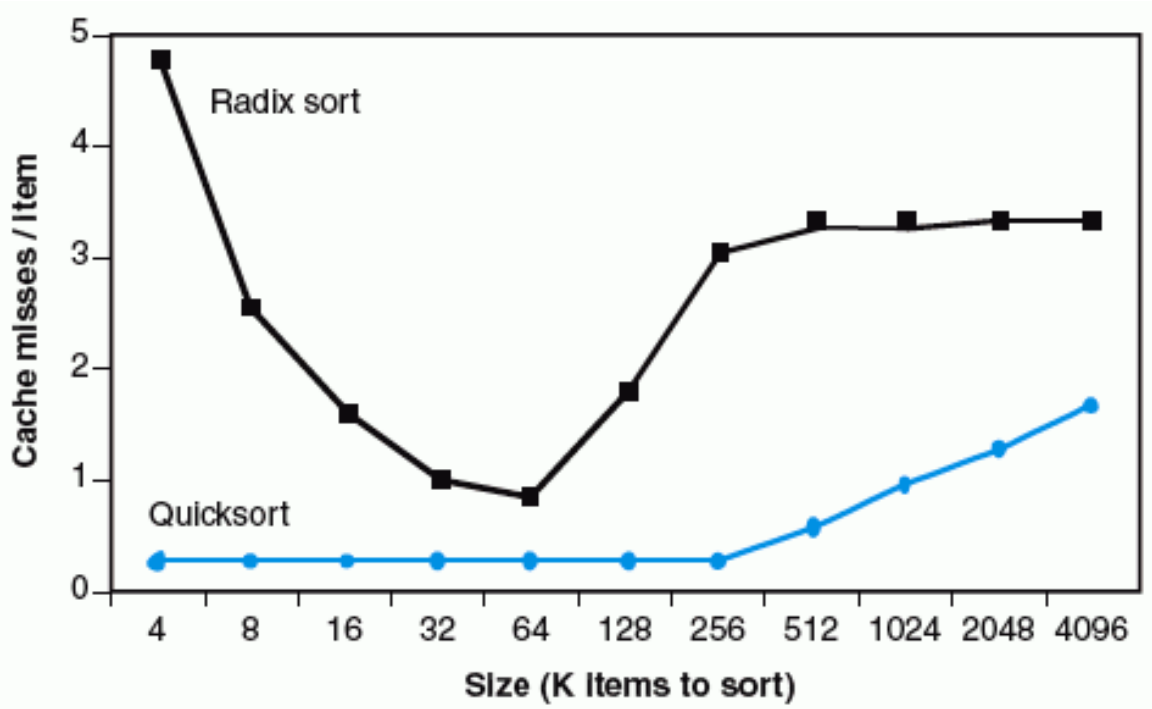
- **Jenže: Quick Sort se pro větší množství dat ukázal rychlejší ...**



Zdroj: P&H

Proč je důležité o cache vědět? (3)

- **Důvod**
  - Způsob přístupu k datům v implementaci algoritmu *Radix Sort* způsoboval příliš mnoho výpadků cache



Zdroj: P&H

Proč je důležité o cache vědět? (4)

- **Řešení**
  - Úprava implementace algoritmu *Radix Sort*, aby pracoval s daty nejprve v rámci bloku paměti, který je již načtený v cache (řádku cache)

Zdroj: P&H

Shrnutí: paměť dominuje výkonu CPU

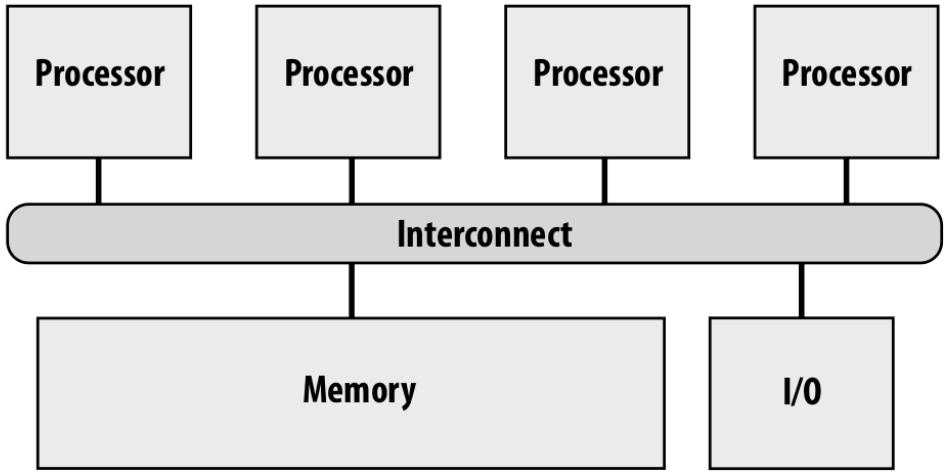
- **Paměťová zeď (*the memory wall*)**
  - výkonnost roste rychleji u procesorů než u paměti
- **Neexistuje ideální paměťová technologie**
  - rychlá, velká, levná – nelze mít vše najednou
- **Lokalita přístupu do paměti**
  - časová + prostorová, vlastnost reálných programů
- **Řešení: hierarchie pamětí**
  - optimalizace průměrné doby přístupu do paměti
  - různé technologie v různých vrstvách
  - mechanizmus pro přesun dat mezi vrstvami

Shrnutí: cache jako iluze ideální paměti

- **1-3 úrovně rychlé paměti mezi CPU a hlavní pamětí**
  - SRAM, kapacita L1 ~ 64KiB, L2/L3 ~ 256KiB-16MiB
  - z pohledu programátora (i CPU) transparentní
    - CPU (datová cesta) požaduje data pouze po cache
    - přesun dat mezi cache a hlavní pamětí zajišťuje HW
  - data uložena v řádcích odpovídajících blokům paměti
    - tag – část adresy, která činí mapování jednoznačné
- **3C model: klasifikace výpadků cache**
  - změna organizace cache s cílem odstranit výpadky
- **ABC: základní parametry cache**
  - asociativita, velikost bloku, kapacita

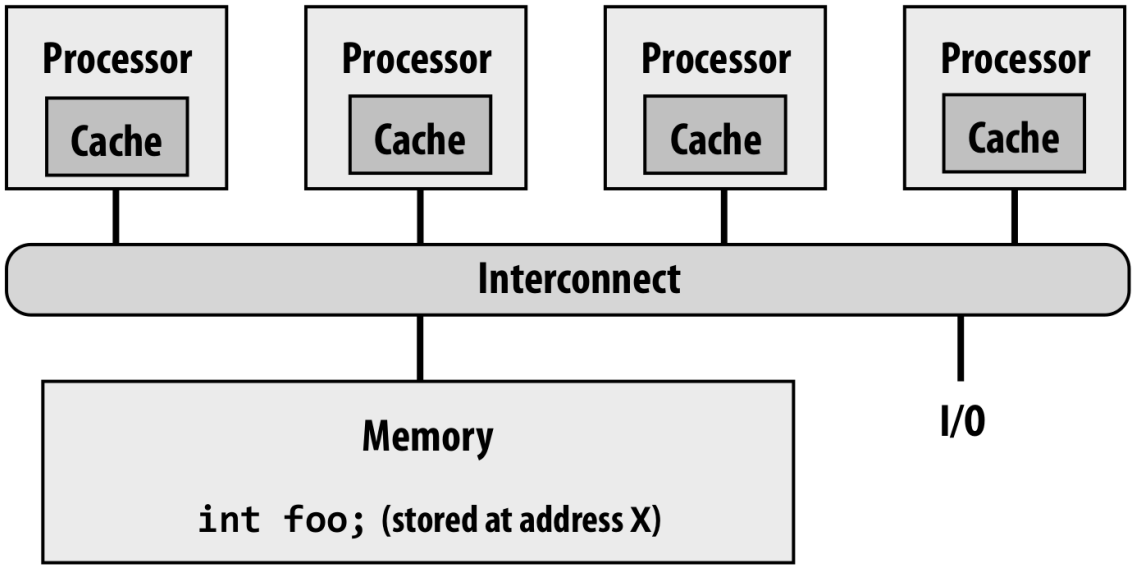
Paralelismus a paměťová hierarchie

- ***Víceprocesorové systémy se sdílenou pamětí***
  - Procesory čtou a zapisují do sdílených proměnných
    - Generují požadavky na čtení/zápis na konkrétní adresy v paměti
- ***Očekávané chování***
  - Čtení z nějaké adresy v paměti vždy vrátí hodnotu, která byla na tuto adresu naposledy zapsána libovolných procesorem



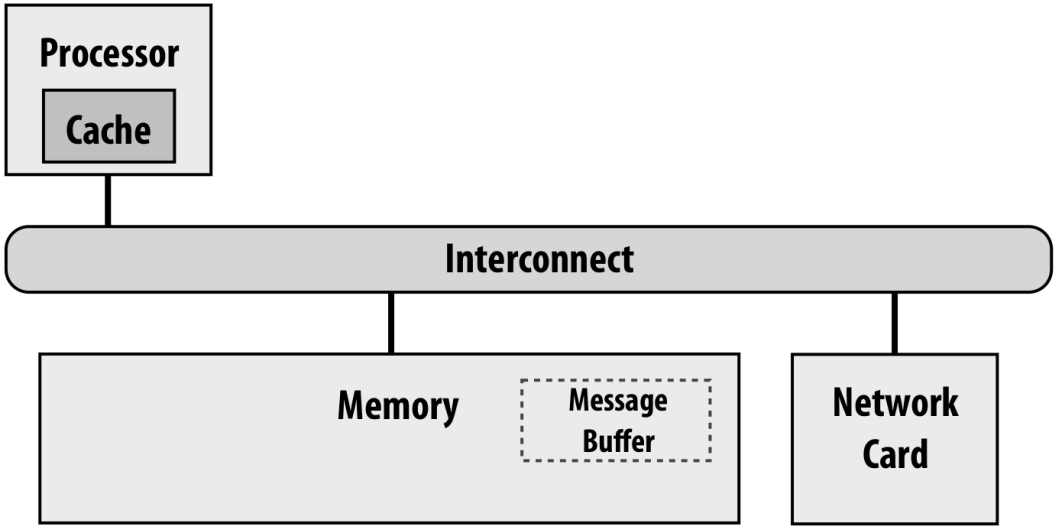
Problém s koherencí dat v cache (1)

- ***Důsledek existence lokálního a globálního stavu***
  - Moderní procesory replikují obsah paměti v lokální cache
  - V důsledku zápisů mohou mít procesory různé hodnoty pro stejnou adresu v paměti



Problém s koherencí dat v cache (2)

- ***Existuje i u jednoprocesorových systémů***
  - DMA přenosy mezi IO zařízeními a pamětí
  - Jak zařízení tak CPU může číst stará data
- ***Typická řešení***
  - Zápisy do paměti označené jako uncached
  - Flush cache po skončení práce nad bufferem



Problém s očekávaným chováním

- ***Očekávané chování***
  - Čtení z nějaké adresy v paměti vždy vrátí hodnotu, která byla na tuto adresu naposledy zapsána **libovolných procesorem**
- ***Co znamená „poslední“ ?***
  - Co když dva procesory zapisují současně?
  - Co když po zápisu procesorem P1 následuje čtení procesorem P2 tak rychle, takže není možné uvědomit ostatní?
- ***V sekvenčním programu je „poslední“ určeno pořadím v programu (nikoliv časem)***
  - Platí i v rámci vlákna v paralelním programu
  - V případě více vláken pro určení pořadí nestačí.

Koherentní paměťový systém

- ***Procesor vidí vlastní zápisy v programovém pořadí***
  - Intuitivní požadavek pro jednoprocesorové systémy.
- ***Zápisy do paměti nakonec uvidí všechny procesory***
  - Definuje koherentní pohled na paměť. Pokud by procesor mohl neustále číst starou hodnotu, paměť by byla nekoherentní.
  - Není určeno kdy přesně se informace o zápisu propaguje.
- ***Zápisy do stejného místa jsou serializované (uspořádané)***
  - Všechny procesory uvidí zápisy do stejného místa ve stejném pořadí.

Uspořádání (serializace) zápisů

- ***Zápisy do stejného místa jsou serializované (uspořádané)***
  - Dva zápisy do stejného místa libovolnými dvěma procesory musí ostatní procesory vidět ve stejném pořadí.
- ***Příklad***
  - P1 zapíše hodnotu **a** do **X**.
    Poté P2 zapíše hodnotu **b** do **X**.
  - Pokud by každý procesor viděl zápisy v jiném pořadí…
    - P1 by nejprve viděl svůj zápis **a** do **X** a až poté cizí zápis **b** do **X**
    - P2 by nejprve viděl svůj zápis **b** do **X** a až poté cizí zápis **a** do **X**
  - V koherentním systému neexistuje globální uspořádání, které by takový výsledek paralelního programu umožnilo.

Koherence vs konzistence

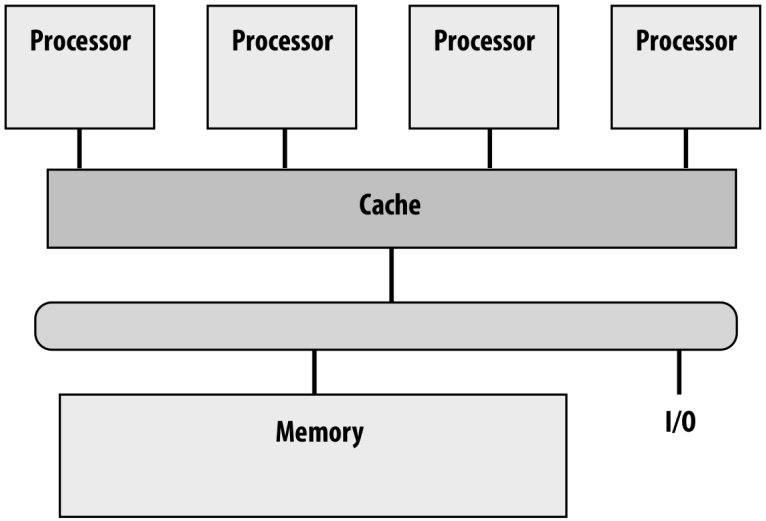
- ***Koherence***
  - Určuje ***jaké hodnoty*** uvidíme při čtení
  - Týká se čtení/zápisu do jednoho místa v paměti
- ***Konzistence***
  - Určuje ***kdy*** budou zápisy viditelné pro čtení
  - Týká se čtení/zápisu do více míst v paměti
- ***Pro naše účely***
  - Pokud procesor zapíše na adresu X a potom na adresu Y, pak libovolný jiný procesor, který vidí výsledek zápisu do Y, uvidí také zápis do X.

Zajištění koherence

- ***Hardwarová řešení***
  - HW zajistí, že čtení nějakého místa v paměti libovolným procesorem vrátí poslední hodnotu zapsanou do tohoto místa
    - Pro vhodnou/smysluplnou definici „posledního“
  - Metadata udržují informace o stavu dat v cache ve vztahu k ostatním
  - Řešení založena na zneplatnění (invalidaci) nebo aktualizaci (update) dat v cache
    - Koherenční protokol: pravidla pro změny stavu (metadat) konkrétního bloku v cache v rámci celého systému

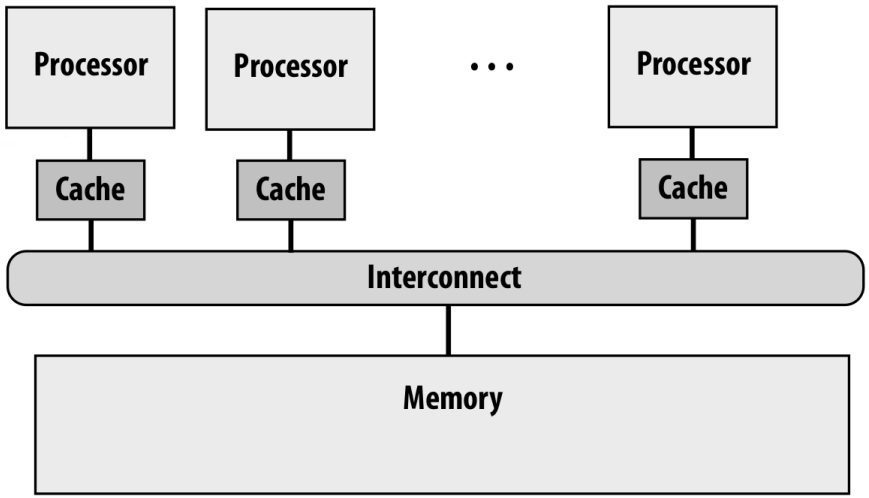
Snadné řešení: sdílená cache

- ***Problémy se škálovatelností***
  - Interference, contention
- ***Potenciální výhody***
  - Jemná granularita sdílení (překryv pracovních množin)
  - Akce jednoho procesoru mohou přednačítat data pro jiné



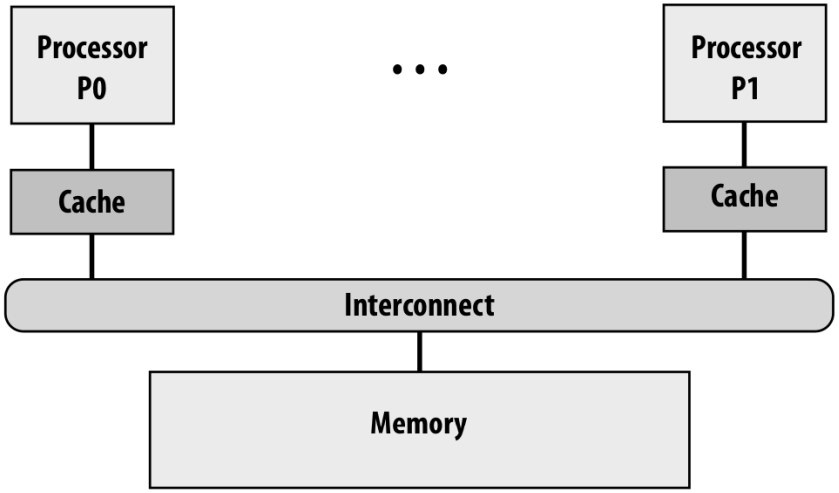
Řešení založená na snoopingu

- ***Procesory (řadiče cache) sdílejí projovovací médium***
  - Všechny události související s koherencí šířeny (broadcast) všem procesorům (řadičům cache) v systému
- ***Řadiče cache monitorují (snoop) paměťové operace***
  - Individuálně reagují tak, aby byla zajištěna koherence dat
  - Musí reagovat jak na události jak ze strany procesoru (datové cesty), tak propojovacího média (aktivita ostatních procesorů)

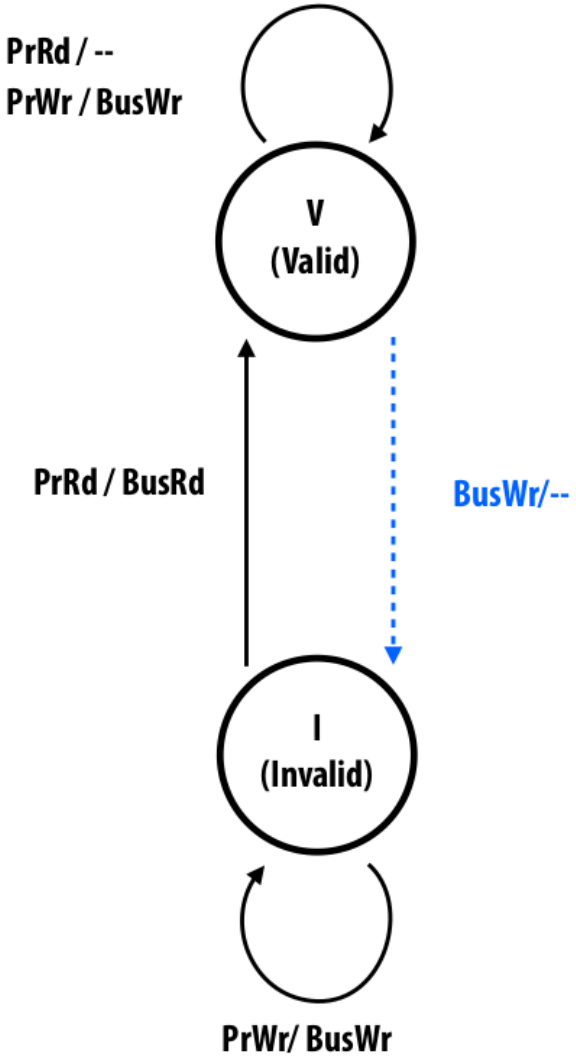


Jednoduchá implementace koherence

- ***Write-through cache***
  - Granularita koherence je cache line
- ***Při zápisu invaliduje cache-line ostatních procesorů***
  - Broadcast na sdíleném spoji
  - Příští čtení stejné cache line na jiném procesoru bude cache miss
  - Procesor načte aktuální hodnotu z paměti



Základní Valid/Invalid (VI) protokol



- *A/B = radič cache vidí akci A, provede akci B*
  - Reakce na aktivitu na sdíleném spoji
  - Reakce na požadavek procesoru
- *Akce protokolu*
  - Processor Read (PrRd)
  - Processor Write (PrWr)
  - Bus Read (BusRd)
  - Bus Write (BusWr)
- *Požadavky na sdílený spoj*
  - Zápisové transakce viditelné pro všechny řadiče cache
  - Zápisové transakce viditelné ve stejném pořadí
- *Zjednodušující předpoklady*
  - WT cache používá strategii write no-allocate
  - Paměťové transakce a transakce na sdíleném spoji jsou atomické
  - Procesor čeká na dokončení paměťové operace předtím než zahájí novou
  - Invalidace proběhne okamžitě, jako součást přijetí invalidační výzvy

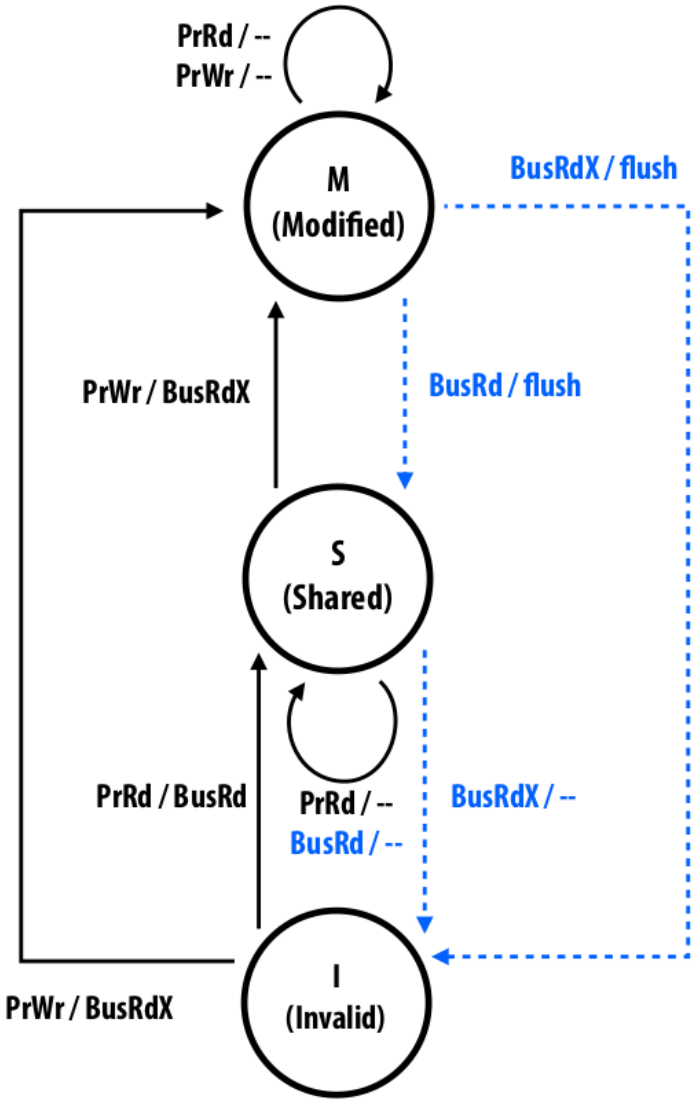
Write-through strategie je neefektivní

- ***Každý zápis propagován do paměti***
  - Vysoké požadavky na přenosovou kapacitu
- ***Write-back cache absorbují většinu zápisů***
  - Výrazně nižší požadavky na přenosovou kapacitu
  - Jak zajistit propagaci zápisů/uspořádání?
  - Vyžaduje sofistikovanější protokol

Invalidační protokol pro write-back cache

- ***Cache line ve výhradním (exclusive) stavu možno modifikovat bez signalizace ostatním***
  - Ostatní cache ji nemají, takže ostatní procesory nemohou data z cache line číst aniž by vygenerovaly požadavek na čtení z paměti
- ***Zápis možný pouze do cache line ve výhradním stavu***
  - Pokud chce procesor zapisovat do cache line, která není ve výhradním stavu, řadič cache nejprve signalizuje výhradní čtení (read-exclusive)
  - Požadavek na výhradní čtení informuje ostatní cache o nadcházejícím zápisu
  - Požadavek na výhradní čtení je nutný i pro cache line, která už v cache je
  - Dirty cache line je nutně výhradní (exclusive)
- ***Pokud řadič cache vidí požadavek na výhradní čtení***
  - Pokud se týká cache line, kterou má u sebe, musí ji invalidovat

Základní invalidační protokol MSI



- ***Hlavní cíle protokolu***
  - Získat výhradní přístup pro zápis
  - Nalezení nejnovější kopie při cache miss
- ***Stavy protokolu***
  - I: neplatná cache line
  - S: clean cache line v jedné nebo více cache
  - M: dirty cache line právě v jedné cache
- ***Akce protokolu***
  - Processor Read (PrRd)
  - Processor Write (PrWr)
  - Bus Read (BusRd)
    - Čtení cache line bez úmyslu ji modifikovat
  - Bus Read Exclusive (BusRdX)
    - Čtení cache line s úmyslem ji modifikovat.
  - Bus Write Back (BusWB)
    - Zápis cache line do paměti

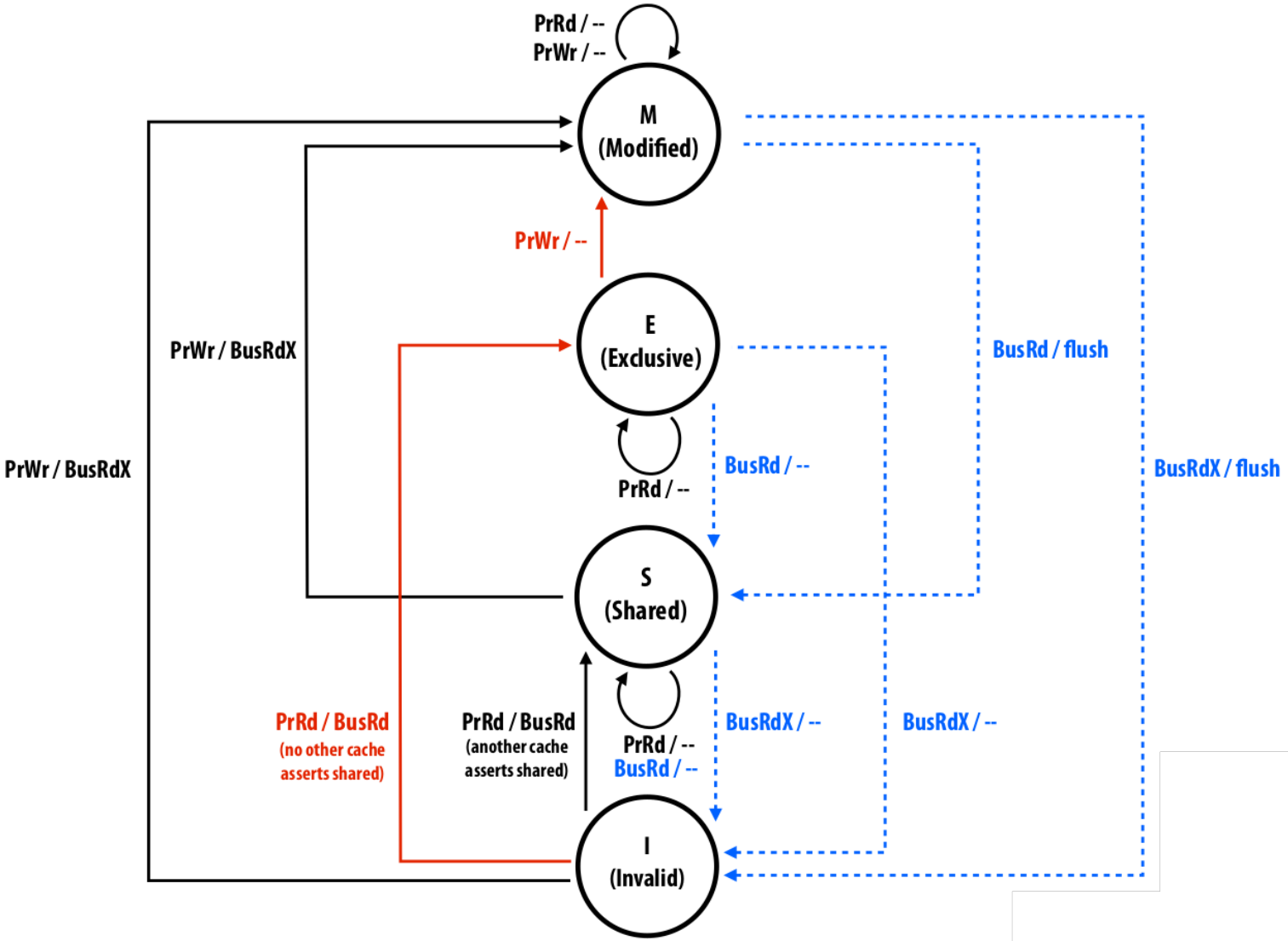
Uspokojí MSI požadavky na koherenci?

- ***Propagace zápisů***
  - V rámci invalidace.
- ***Uspořádání zápisů***
  - Zápisy, které se objeví na sběrnici, jsou uspořádány v pořadí, v jakém se objeví na sběrnici (BusRdX)
  - Čtení, která se objeví na sběrnici, jsou uspořádány v pořadí, v jakém se objeví na sběrnici (BusRd)
  - Zápisy do cache line ve výhradním (M) stavu se na sběrnici neobjeví
    - Posloupnost zápisů do cache line se nachází mezi dvěma transakcemi na sběrnici.
    - Všechny zápisy v posloupnosti provádí stejný procesor P, který je vidí ve správném pořadí.
    - Všechny ostatní procesory uvidí tyto zápisy až po sběrnicové transakci pro danou cache line a všechny zápisy této transakci předchází.
    - Všechny procesory vidí zápisy ve stejném pořadí.

Invalidační protokol MESI (1)

- ***MSI vyžaduje 2 transakce pro běžný případ čtení a následné modifikace dat***
  - 1. transakce: BusRd pro přesun ze stavu I do stavu S.
  - 2. transakce: BusRdX pro přesun ze stavu S to stavu M.
- ***Nutné i v případě, kdy se cache line nikdy nesdílí***
- ***Řešení: nový stav E (exclusive clean)***
  - Cache line není modifikovaná, ale existuje pouze v jedné cache
  - Odděluje exkluzivitu cache line od vlastnictví (cache line není dirty, takže kopie dat v paměti je platnou kopií)
  - Přesun ze stavu E do stavu M nevyžaduje sběrnicovou transakci

Invalidační protokol MESI (2)



Efektivnější (a složitější) protokoly

- ***MOESI (AMD Opteron)***
  - V protokolu MESI přesun ze stavu M do stavu S vyžaduje zápis dat do paměti
  - MOESI přidává stav O (owned, not exclusive) bez zápisu dat (cache line zůstane dirty)
  - Ostatní procesory mohou mít cache line ve stavu S, právě jeden procesor má cache line ve stavu O
  - Data v paměti nejsou aktuální, takže cache obsahující cache line ve stavu O musí obsluhovat cache missy ostatních procesorů
- ***MESIF (Intel)***
  - Jako MESI, ale jedna cache drží sdílenou cache line ve stavu F (forward) místo ve stavu S
  - Cache obsahující cache line ve stavu F obsluhuje cache miss
  - Cache, která cache line četla naposled, ji načte ve stavu F
    - Stav F migruje do poslední cache, která cache line načetla po read missu
    - Předpokládá se, že tato cache tuto cache line hned nezahodí (a stav F se neztratí)
  - Zjednodušuje rozhodování o tom, která cache má obsloužit cache miss

Důsledky implementace koherence

- ***Každá cache musí poslouchat a reagovat na koherenční události šířené po sdíleném spoji***
  - Nutno duplikovat tagy cache aby vyhledání tagu neinterferovalo s load/store požadavky procesoru
- ***Vyšší zatížení sdíleného spoje***
  - Může být významné/limitující pro velký počet jader
- ***GPU koherenci neimplementují vůbec nebo v omezené formě***
  - Příliš vysoká režie, malé uplatnění u grafických aplikací

Důsledky pro programátora

- ***Co je špatně na následujícím kódu?***

```
// allocate per-thread accumulators
int counters [NUM_THREADS];
```

- ***Lepší verze***

```
// allocate per-thread accumulators
struct PerThreadState {
    int counter;
    char padding [64 – sizeof (int)];
}

PerThreadState counters [NUM_THREADS];
```

Falešné sdílení (false sharing)

- ***Dvě vlákna zapisují do různých proměnných ve stejné cache line***
  - Cache line přeskakuje mezi cache zapisujících procesorů
  - Koherenční protokol způsobuje velké množství komunikace přestože spolu vlákna vůbec nekomunikuji
  - Veškerá komunikace je pouze nežádoucí produkt falešného sdílení
- ***Může výrazně ovlivnit výkonnost programu na architekturách implementujících koherenci***
  - Naprostá většina běžných CPU
  - Bez ohledu na programovací jazyk

Reference

[1] Roth A., Martin M.: *CIS 371 – Computer Organization and Design*, University of Pennsylvania, Dept. Of Computer and Information Science, 2009