Principy počítačů a operačních systémů

Hierarchie paměti a cache

Zimní semestr 2011/2012

Poděkování

Při přípravě této prezentace jsem převzal a přeložil velké množství materiálu z prezentace

- Roth, A., Martin, M. **CIS 371 – Computer Organization an Design**. University of Pennsylvania, Dept. of Computer and Information Science, Spring 2009.

Motivace

Výkon procesoru je omezen rychlostí paměti

- operace "add" trvá 0.33ns na 3GHz procesoru
- latence dnešních pamětí je více než 33ns
- naivní implementace: přístup do paměti (čtení/zápis) může být až 100x pomalejší než ostatní operace

Nedosažitelný cíl

- paměť pracující stejně rychle jako procesor
- kapacita vždy odpovídající potřebám programu
- rozumná cena

Nelze mít všechno současně!

Volatilní paměti

Statické

- primární cíl: rychlost, sekundární cíl: kapacita
  - 6 tranzistorů na bit, rychlost závisí na ploše, pro malé kapacity latence < 1ns
- dobře se kombinuje s ostatní procesorovou logikou
- obsah není nutné periodicky obnovovat

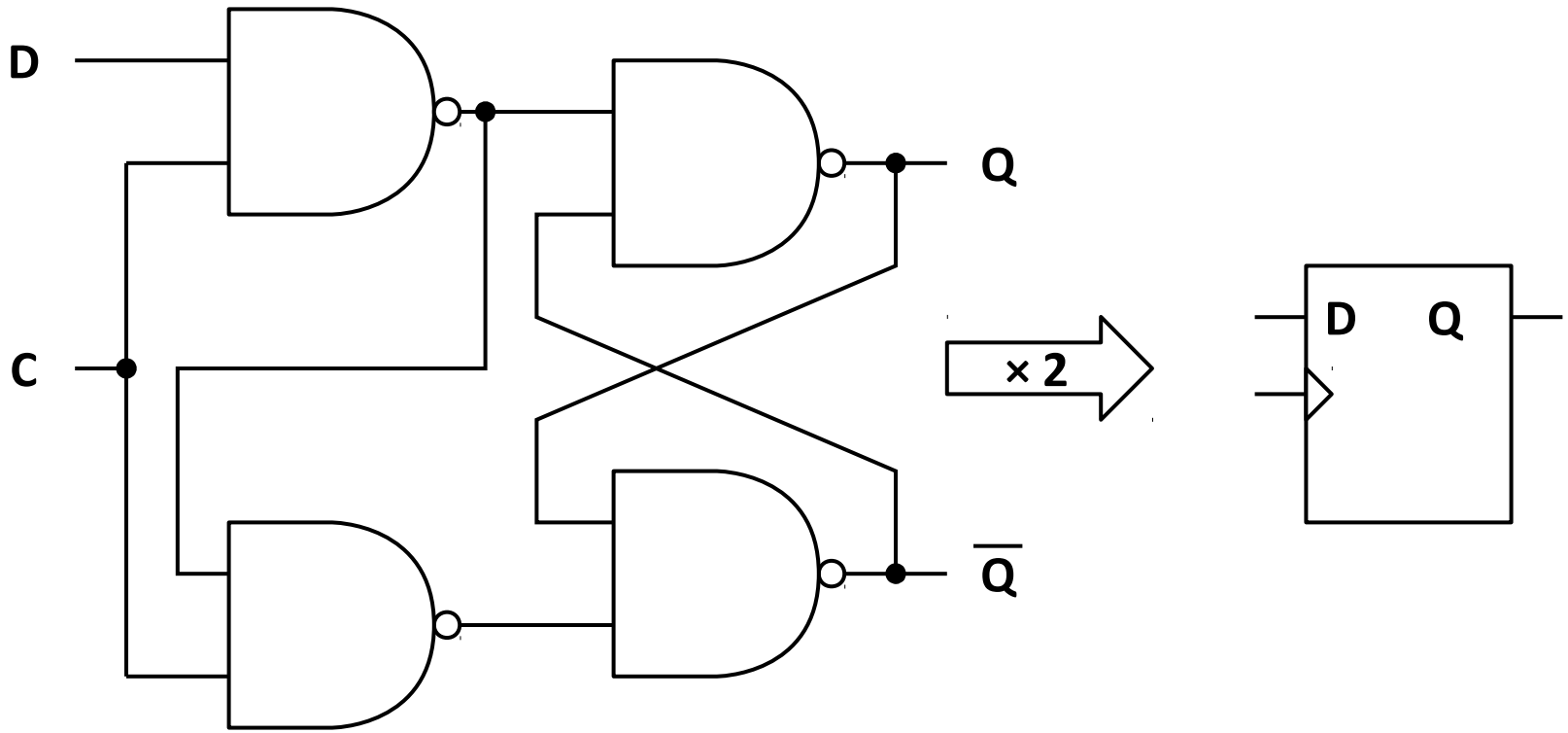
Dynamické

- primární cíl: hustota (cena za bit)
  - 1 tranzistor + 1 kondenzátor na bit, vysoká latence (> 40ns uvnitř samotné paměti, 100ns mezi obvody)
- obsah je nutné periodicky obnovovat (znovu zapisovat)
- nelze dobře kombinovat s logikou (jiný výrobní proces)

Statická paměť: klopné obvody

1 bit ~ klopný obvod typu D

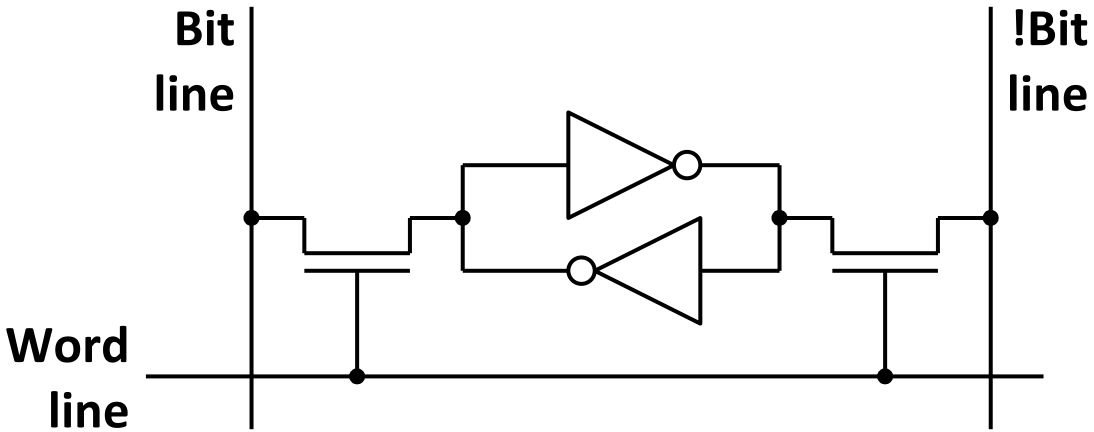
- ~ 9 hradel ~ 18 tranzistorů



Statická paměť: buňka paměti SRAM

1 bit ~ dvojice invertorů + řídící tranzitory

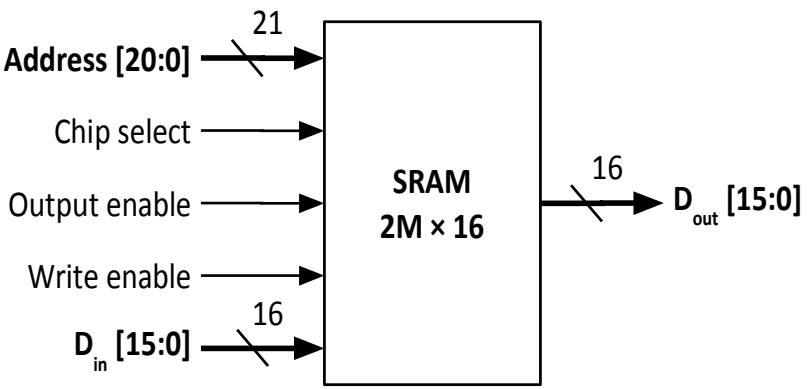
- invertor ~ 2 tranzistory
- 6 tranzistorů na 1 buňku

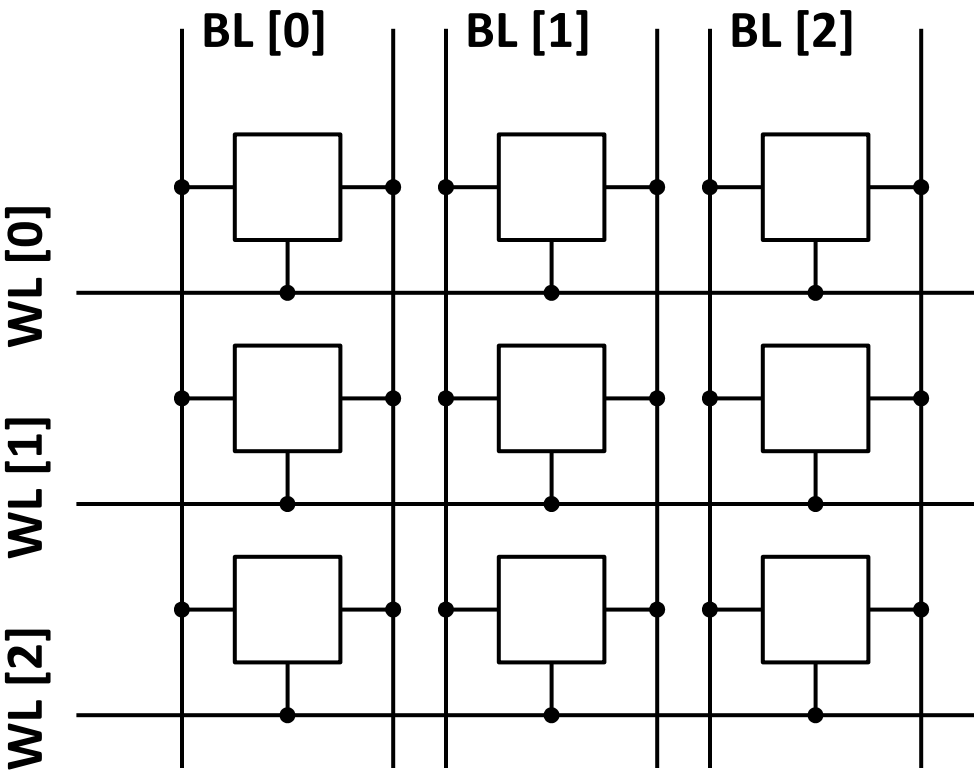


Statická paměť: SRAM v maticovém uspořádání

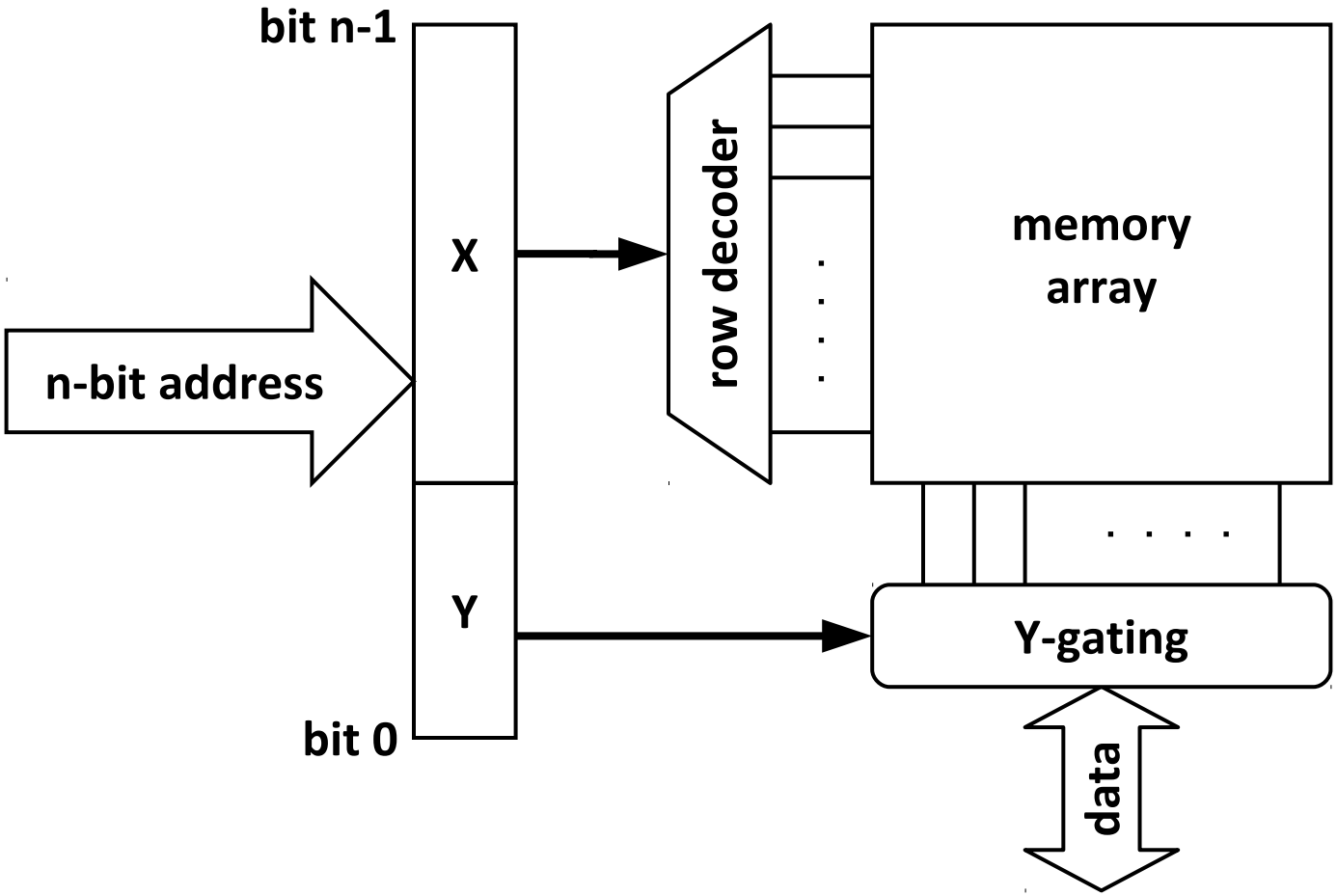
m × **n** bitů ~ **m** řádků po **n** bitech

- vysoká hustota integrace
  - výběr řádku ~ dekodér 1 z N
- přístup do paměti ve dvou krocích
  1. výběr řádku (*word lines*)
  2. čtení sloupců (*bit lines*)





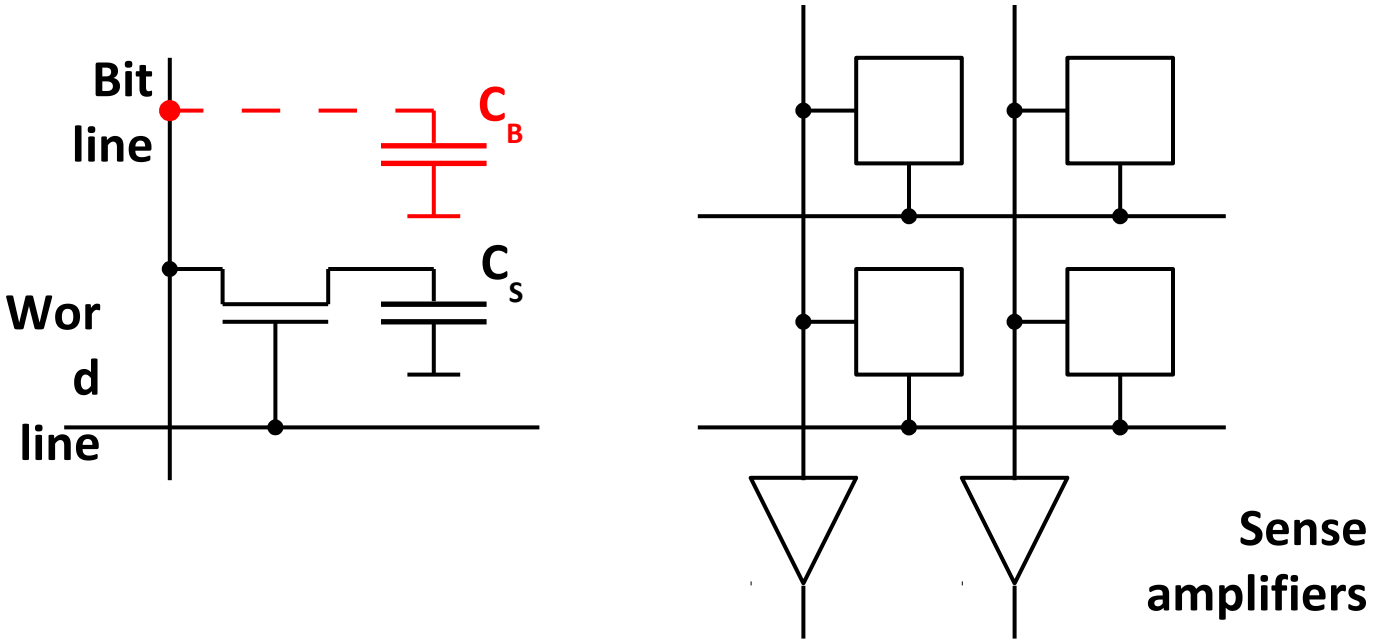
Vnitřní organizace paměti



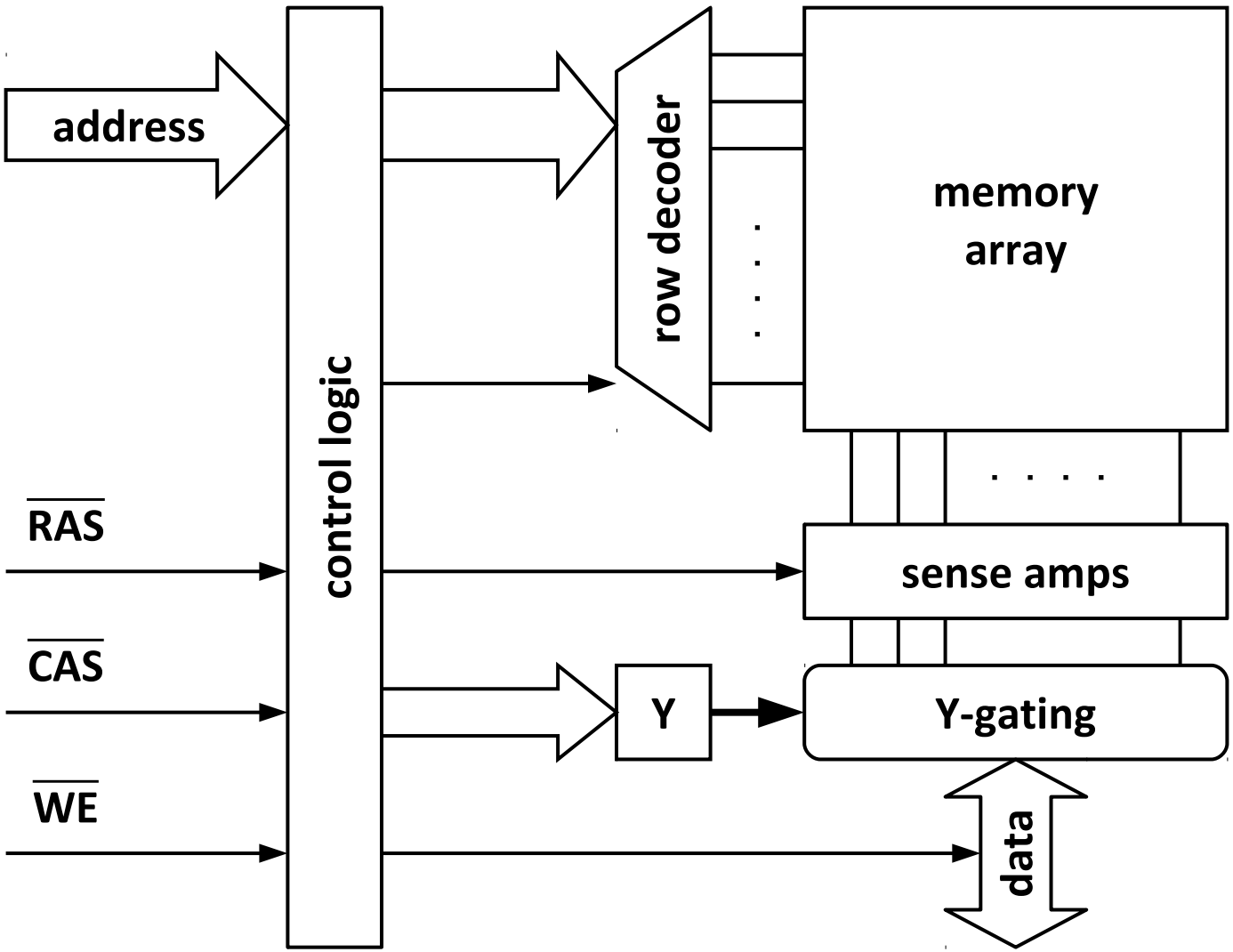
Dynamická paměť: buňka paměti DRAM

1 bit ~ kondenzátor + řídící tranzistor

- informaci nutno obnovovat
  - informace se časem ztrácí
  - destruktivní čtení



Typická organizace DRAM



Zvyšování výkonu DRAM

Pozorování

- nejdéle trvá čtení řádku
- řádek obsahuje více než jen požadované slovo

Amortizace ceny čtení řádku

- použít více slov z jednoho přečteného řádku
  - data z přečteného řádku jsou uložena v registru ⇒ je možné přečíst několik "sloupců" za sebou
- pipelining výstupu dat a výběru nového řádku
  - přečtený řádek zkopírován do (dalšího) výstupního registru, což umožní zahájit čtení dat z jiného řádku současně s přenosem dat z paměti (výstupního registru)

Permanentní paměti

Magnetický záznam

- pevný disk, páska, disketa
  - vysoká latence
  - vesměs sekvenční přístup, mechanické zpoždění

Optický a magneto-optický záznam

- CD/DVD/BD ROM/RW
- MiniDisc

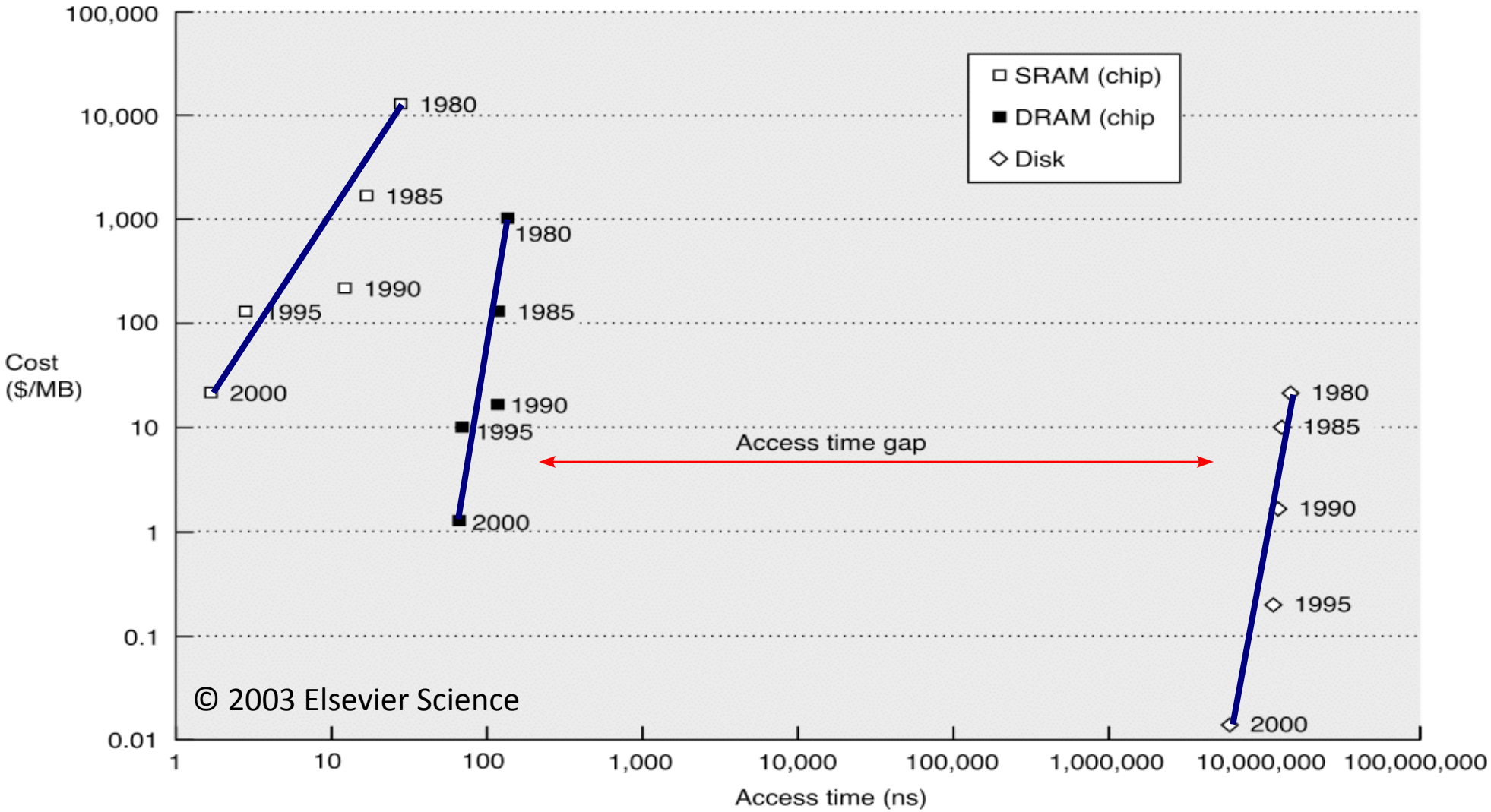
Paměť typu Flash

- elektrický náboj zachycený v polovodiči
- absence mechanického zpoždění ⇒ rychlé čtení
  - zápis (po velkých blocích) více problematický

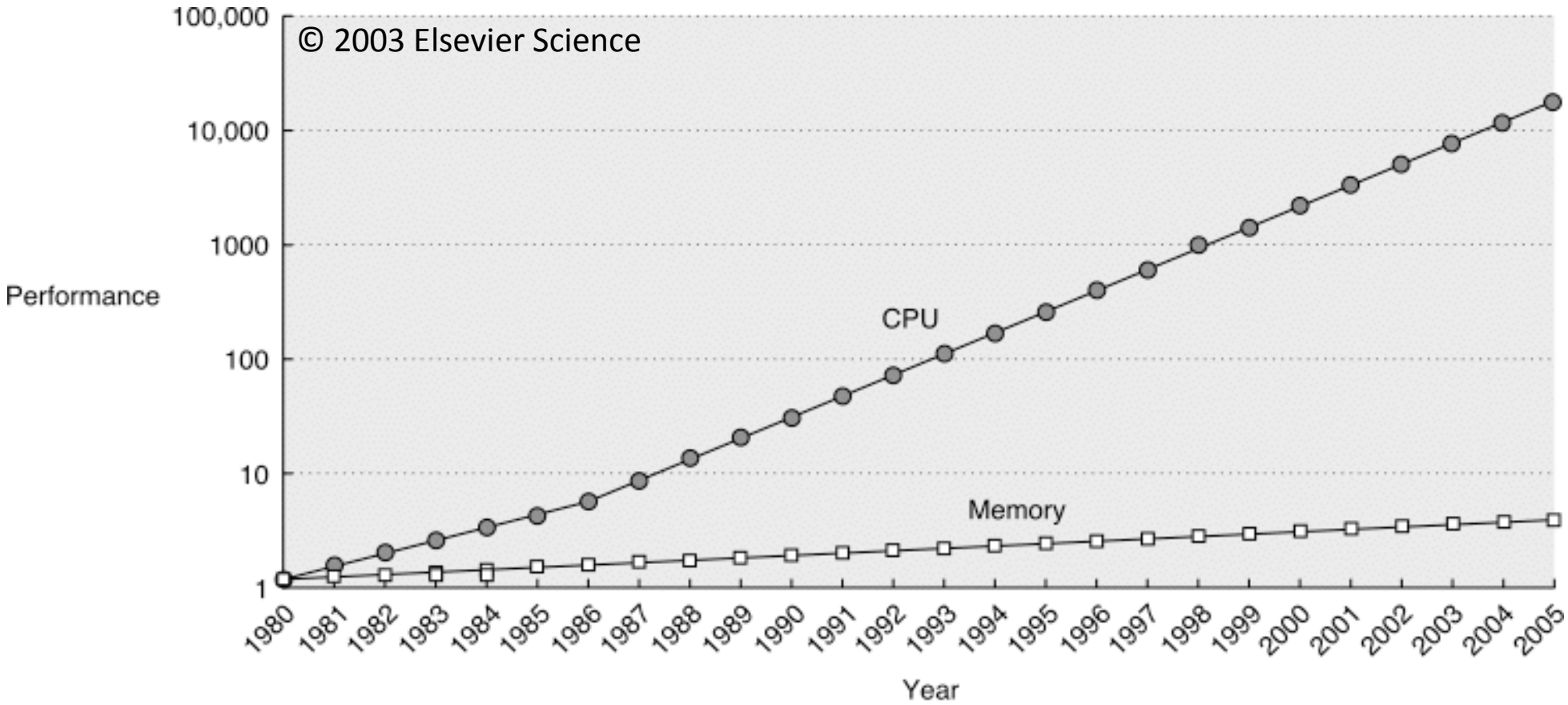
Co se dá koupit za stejnou cenu?

	SRAM	DRAM	Flash	Disk
Kapacita	8 MiB	4 GiB 500x levnější	64 GB 16x levnější	1.5 TB 25x levnější
Latence	<1 - 2 ns (na čipu)	~ 50 ns > 100x pomalejší	~ 75 µs 1500x pomalejší	10 ms 133x pomalejší
Propustnost	300 GB/s (registry)	~ 25 GB/s 12x pomalejší	250 MB/s 100x pomalejší	100 MB/s 2.5x pomalejší

Vývoj paměťových technologií



Paměťová bariéra (*the memory wall*)



Výkon procesoru dominován výkonem pamětí

- 35-55% nárůst rychlosti vs. 7% snížení latence
- výkon procesorů roste rychleji než výkon pamětí

Jak překonat paměťovou bariéru?

Lokalita přístupu do paměti

- vlastnost reálných programů (až na výjimky)
- funguje pro instrukce i pro data

Časová (temporal) lokalita

- k nedávno použitým datům bude s velkou pravděpodobností přistupováno znovu
- reaktivní využití: nedávno použitá data si schováme v malé, velmi rychlé paměti

Prostorová (spatial) lokalita

- s velkou pravděpodobností se bude přistupovat k datům poblíž těch, se kterými se právě pracuje
- proaktivní využití: přistupovat k datům po větších blocích, které zahrnují i okolní data

To se vědělo už na úplném počátku...

Ideally, one would desire an infinitely large memory capacity such that any particular word would be immediately 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.

Burks, Goldstine, von Neumann

"Preliminary discussion of the logical design of an electronic computing instrument"

IAS memo, 1946

Analogie s knihovnou

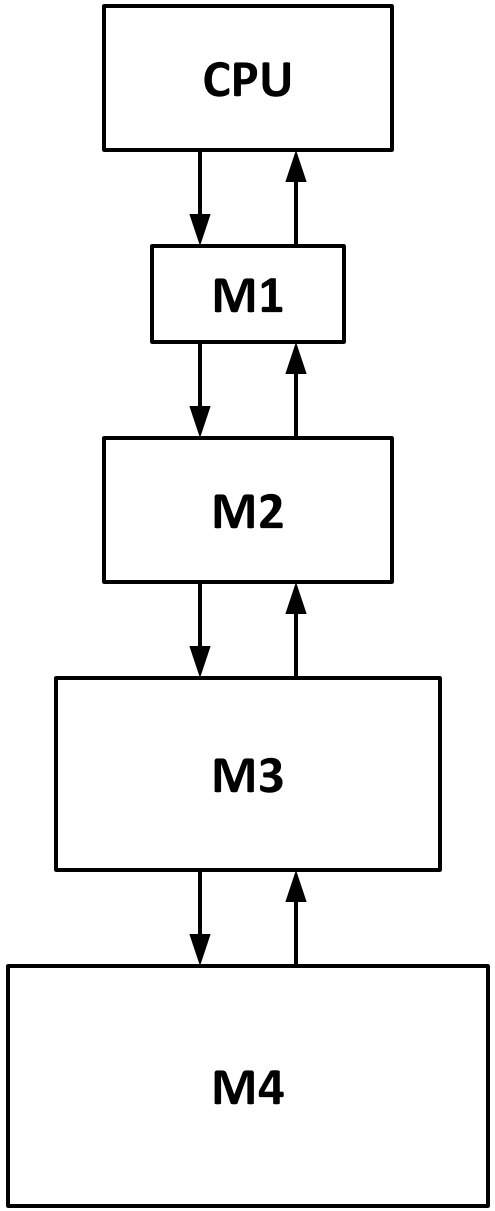
Spousta knih, ale přístup k nim je pomalý...

- vzdálenost (nějakou dobu trvá se tam dostat)
- velikost (nějakou dobu trvá najít knihu)

Jak se vyhnout vysoké latenci?

- půjčit si nějaké knihy domů
  - doma mohou ležet na pracovním stole, nebo na polici
- jenže stůl/police mají omezenou kapacitu
  - často používané knihy musí být po ruce (časová lokalita)
  - půjčíme si více knih na podobné téma (prostorová lokalita)
  - odhadmeme co budeme potřebovat příště (prefetching)

Využití lokality přístupu: hierarchie pamětí



Hierarchie pamětových komponent

- vyšší vrstvy: rychlé ↔ malé ↔ drahé
- nižší vrstvy: pomalé ↔ velké ↔ levné

Vzájemně propojené "sběrnicemi"

- přidávají latenci, omezují propustnost

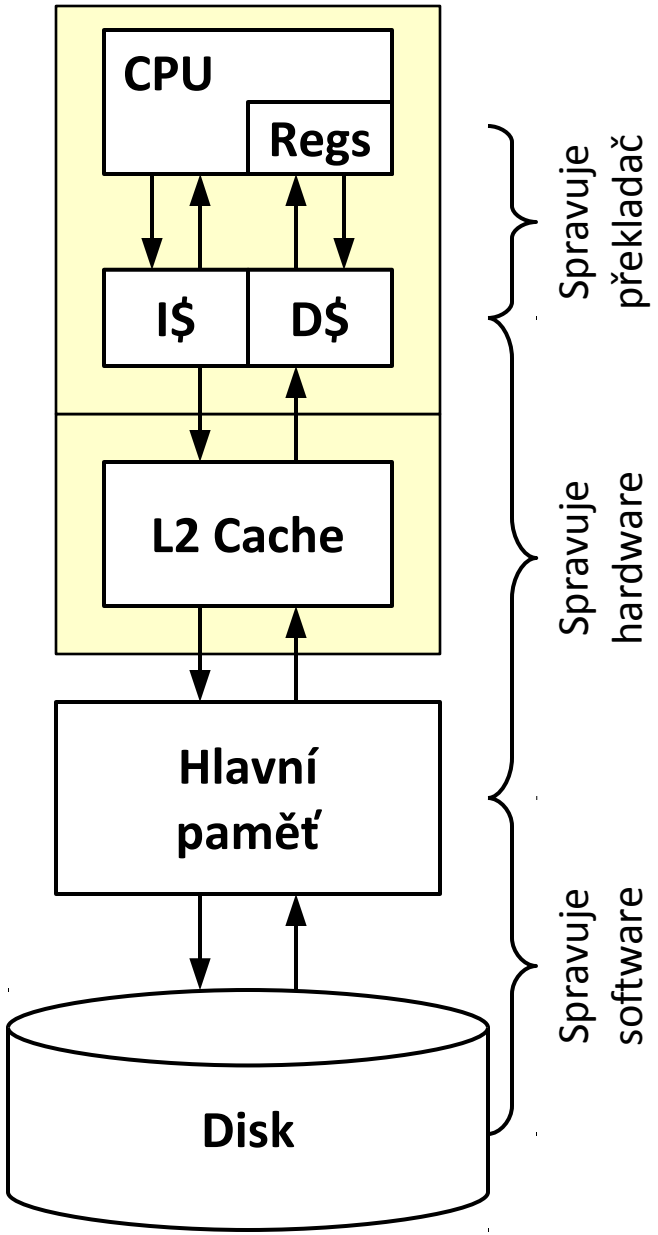
Nejčastěji používaná data v M1

- M1 + druhá nejpoužívanější data v M2
  - M2 + třetí ....
- přesun dat mezi vrstvami hierarchie

Optimalizace průměrné doby přístupu

$$Lat_{avg} = Lat_{hit} + \%_{miss} \times Lat_{miss}$$

Využití lokality přístupu: hierarchie pamětí



M0: registry

M1: primární cache

- oddělená, instrukční a datová
- každá 8-64 KiB

M2: sekundární cache

- nejlépe na čipu, určitě v pouzdře
- SRAM, 512 KiB až 16 MiB

M3: hlavní paměť

- DRAM, 1-8 GiB (servery 128 GiB)

M4: odkládací paměť

- soubory, swap
- magnetické disky, flash paměti

Zpět k analogii s knihovnou

Registry ⇔ knihy na stole

- aktivně využívány, na stůl se jich moc nevejde

Cache ⇔ police na knihy

- střední kapacita, poměrně rychlý přístup

Hlavní paměť ⇔ knihovna

- velká kapacita, je v ní skoro vše, pomalý přístup

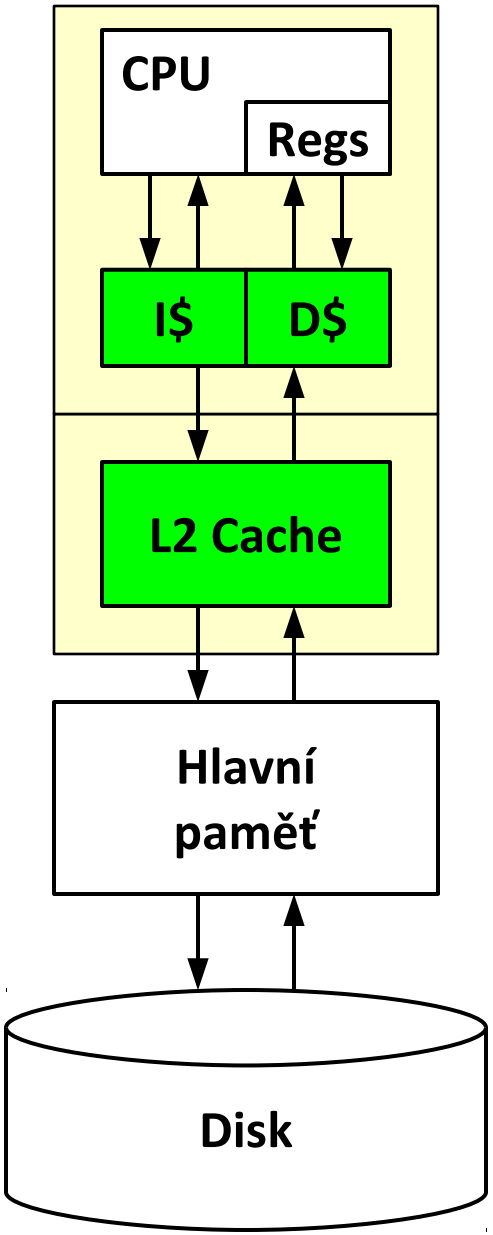
Odkládací paměť ⇔ meziknihovní výpůjčka

- velmi pomalé, ale (doufejme) velmi neobvyklé

Cache

Iluze velké a rychlé paměti

Cache = iluze velké a rychlé paměti



Přesun dat mezi vrstvami řídí HW

- automaticky obstarává chybějící data
  - řadič cache (*cache controller*)
- SRAM, integrována na čipu
  - srovnej: hlavní paměť DRAM, mimo čip

Organizace cache (ABC)

- asociativita, velikost bloku, kapacita
- klasifikace výpadků cache

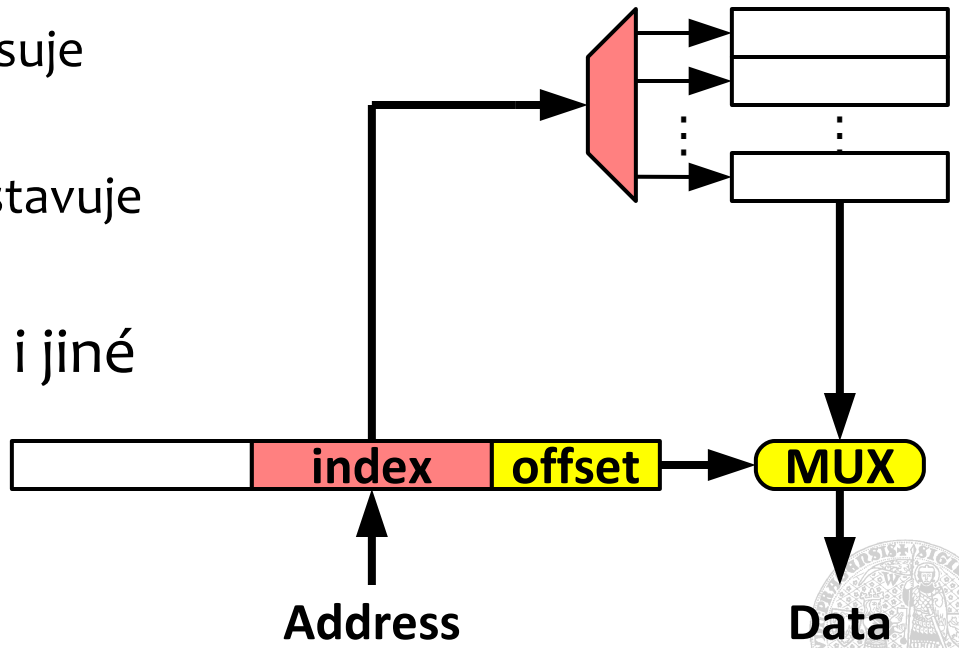
Přímé mapování paměti do cache

Základní struktura cache

- pole řádek (*cache lines*), např. 1024 řádek po 64B ⇒ 64 KiB
- "hardwarová hašovací tabulka"

Jak najít řádek odpovídající bloku paměti?

- dekódovat část adresy... ale kterou?
- předpokládejme 32b adresy
  - 64B bloky ⇒ spodních 6 bitů adresuje bajt v bloku (*offset bits*)
  - 1024 bloků ⇒ dalších 10 bitů představuje číslo bloku (*index bits*)
- jako indexové by se daly použít i jiné bity, ale tyhle fungují nejlépe



Přímé mapování: jak najít správná data?

V řádku cache může být **jeden z 2^{16} bloků paměti**

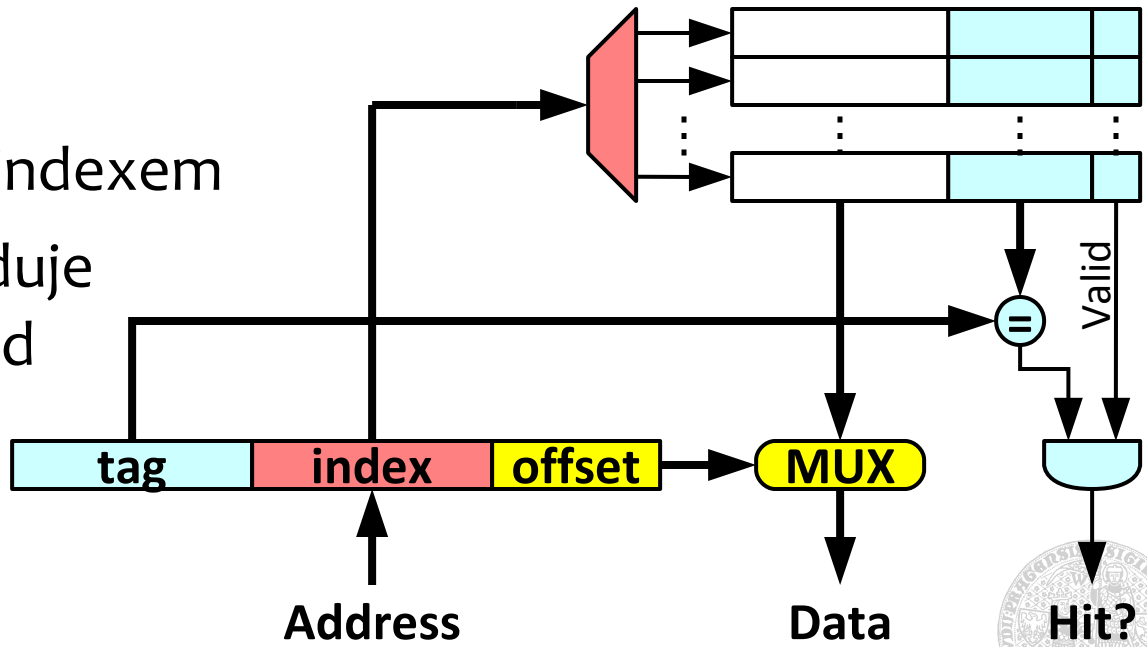
- adresy bloků mají stejné indexové bity ⇒ stejný řádek cache

Jak poznáme, že v řádku vůbec nějaká data jsou?

- k řádku připojíme **tag** a příznak platnosti (***valid bit***)
- porovnáme tag se zbývajícími bity adresy (***tag bits***)
  - indexové bity adresy není nutné porovnávat

Vyhledávací algoritmus

1. přečteme řádek určený indexem
2. pokud se tag řádku shoduje s tagem v adrese a pokud je nastaven *valid bit*, jedná se o ***cache hit***
3. jinak ***cache miss***



Jaká je režie tagů a valid bitů?

Do "64 KiB cache" se vejde 64 KiB **dat**

- místo pro uložení tagů a valid bitů představuje režii

Režie pro 64 KiB cache s 1024 řádky po 64 B

- 64 B řádky → 6 bitů na offset
- 1024 řádků → 10 bitů na index
- 32 bitů adresy → 16 bitů na tag
- (16 tag bitů + 1 valid bit) × 1024 řádků ≅ 2.2 KiB ≅ 3.5%

Co když budou adresy 64 bitové?

- velikost tagu se zvýší na 48 bitů ~ 9.8%

Obsluha výpadku cache (*cache miss*)

Co když data nenajdeme v cache?

- jak se tam vůbec dostanou?

Řadič cache (*cache controller*)

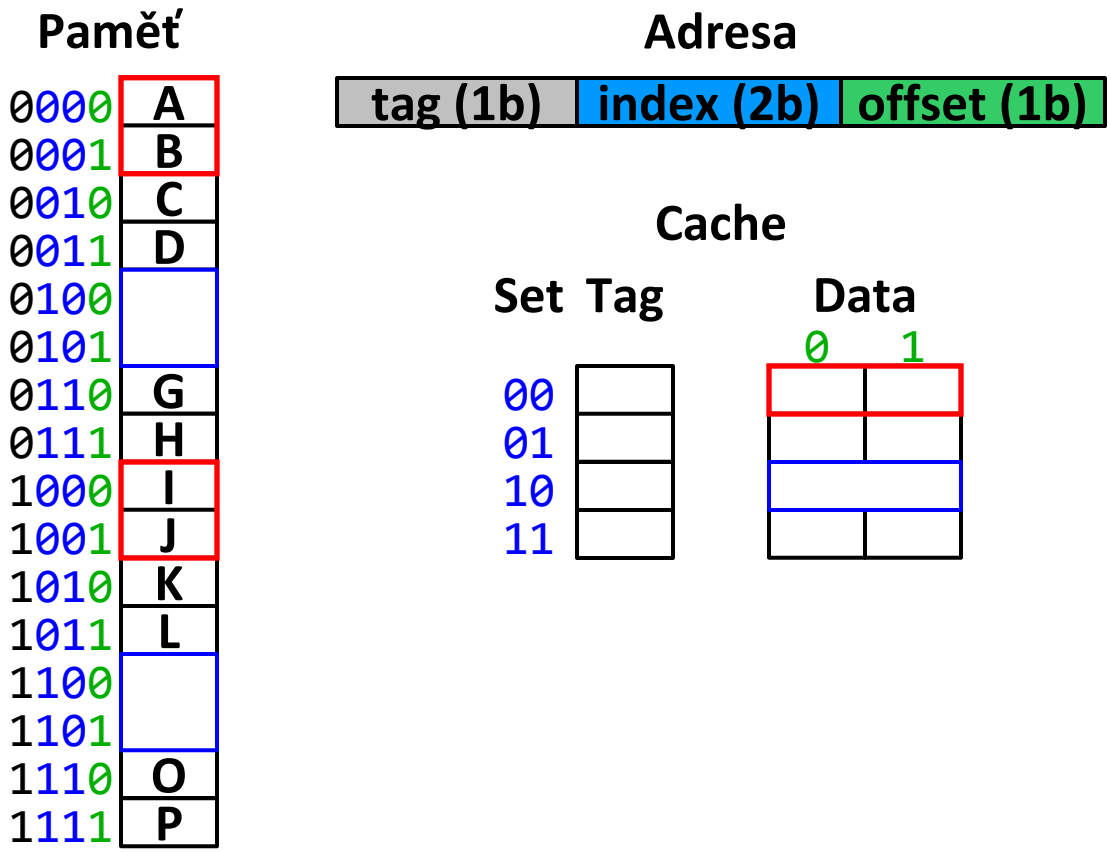
- sekvenční obvod – automat
- zapamatuje si adresu výpadku
- vyžádá si data z následující úrovně hierarchie
- počká na data
- zapíše data + tag + valid bit do řádku

Výpadky cache zpožďují pipeline (jako hazardy)

- zpožďovací logika řízena signálem "cache miss"
- při doplnění dat je signál deaktivován

Příklad: přímé mapování paměti do cache

4-bitová adresa, kapacita 8B, 2B bloky



Příklad: cache miss při čtení adresy 1110

4-bitová adresa, kapacita 8B, 2B bloky

Paměť

0000	A
0001	B
0010	C
0011	D
0100	E
0101	F
0110	G
0111	H
1000	I
1001	J
1010	K
1011	L
1100	M
1101	N
1110	O
1111	P

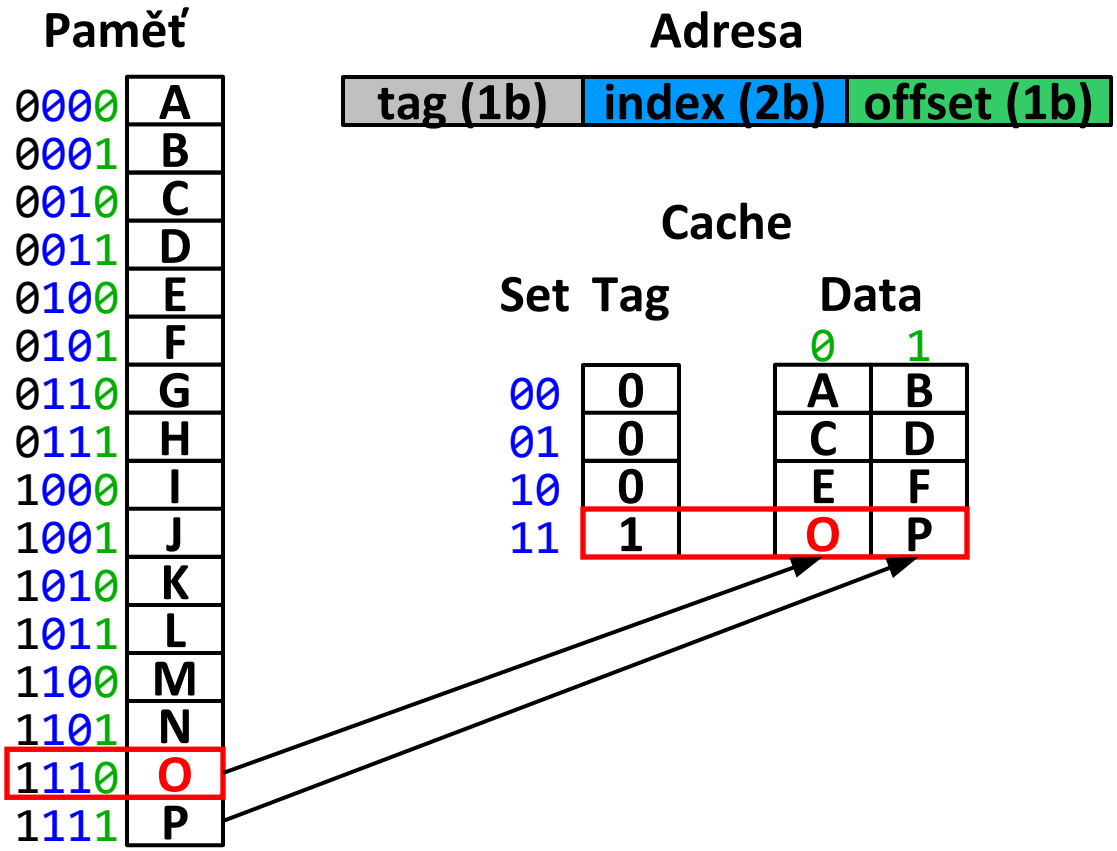
Adresa

Cache

Set	Tag	Data 0	Data 1
00	0	A	B
01	0	C	D
10	0	E	F
11	0	G	H

Příklad: plnění řádku blokem z adresy 1110

4-bitová adresa, kapacita 8B, 2B bloky



Výkonnost cache

Operace cache

- přístup (čtení nebo zápis) do cache (*access*)
- požadovaná data nalezena (*hit*)
- požadovaná data nenalezena, výpadek cache (*miss*)
- načtení dat do cache (*fill*)

Charakteristika cache

- $\%_{miss}$ ... #výpadků / #přístupů (*miss-rate*)
- t_{hit} ... doba přístupu do cache při cache hit
- t_{miss} ... čas potřebný k načtení dat do cache

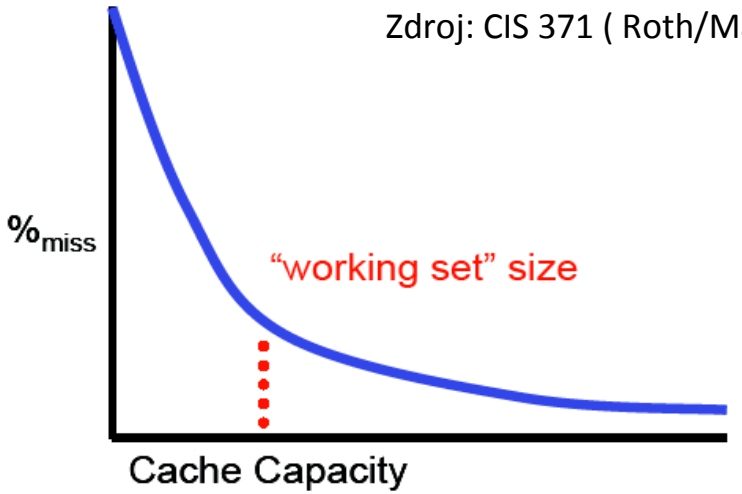
Výkonnostní metrika: průměrný čas přístupu

$$t_{avg} = t_{hit} + \%_{miss} \times t_{miss}$$

Kapacita vs. výkonnost cache

Jak snížit $\%_{miss}$?

- nejjednodušší cesta: zvýšit kapacitu
- miss-rate klesá monotonně
  - zákon klesajících výnosů
- t_{hit} roste s 2. odmocninou kapacity



Zdroj: CIS 371 (Roth/Martin)

Při konstantní kapacitě...

- pro změnu $\%_{miss}$ nutno změnit organizaci cache

Organizace cache: velikost řádku

Zvětšení velikosti řádku

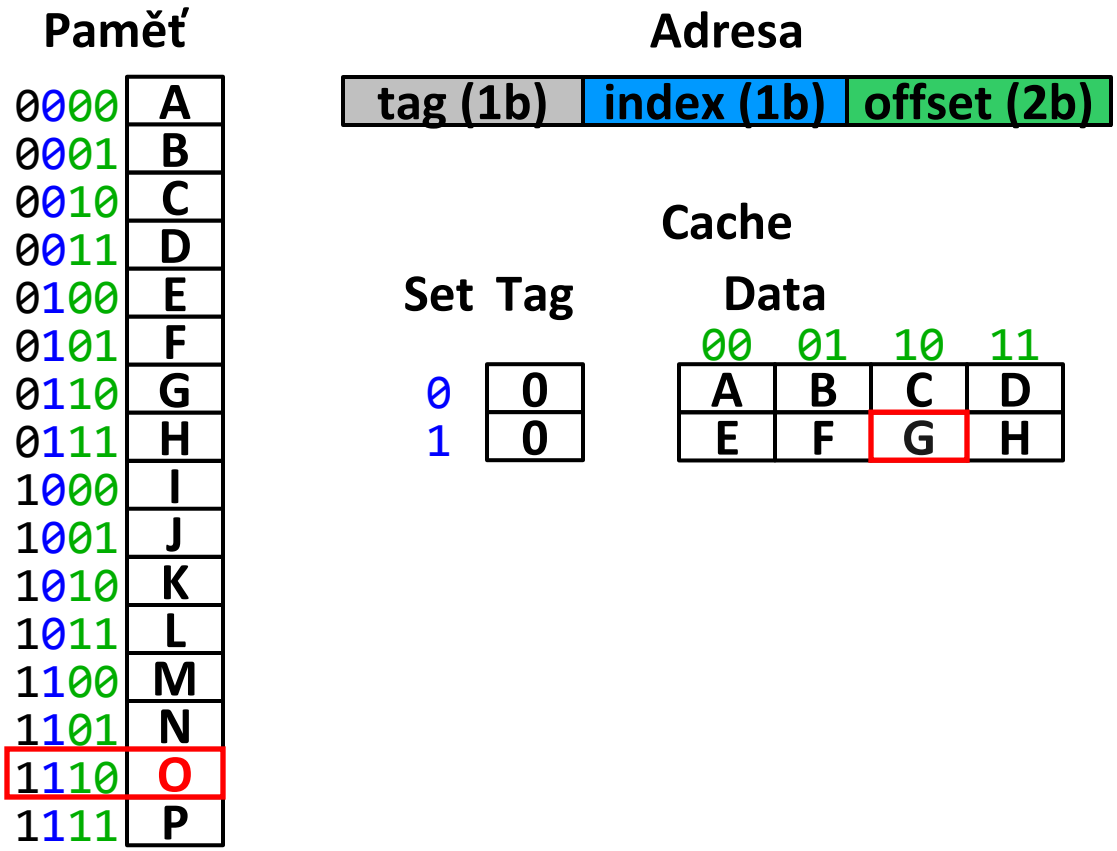
- důraz na využití prostorové lokality
- změna poměru indexových/offsetových bitů
- velikost tagu se nemění

Důsledky

- snížení $\%_{miss}$ (do určité míry)
- nižší režie na tagy (proč?)
- více potenciálně zbytečných přenosů dat
- předčasná náhrada užitečných dat

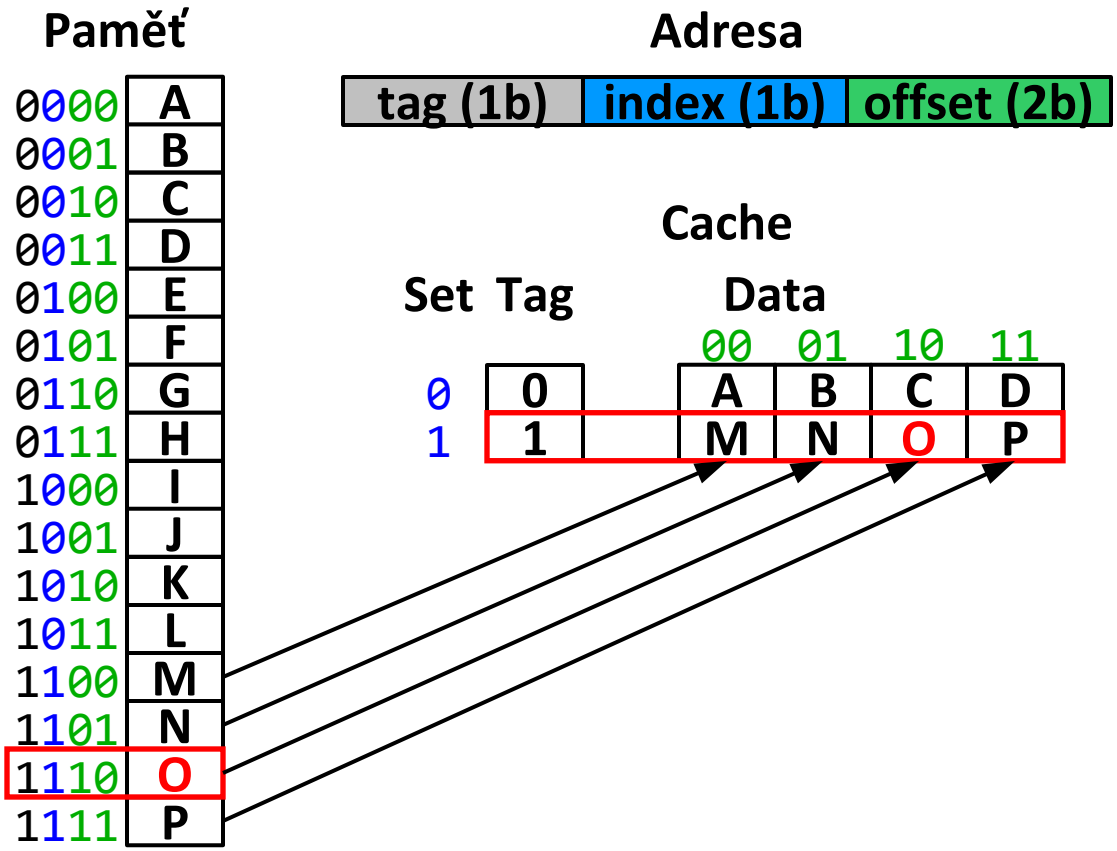
Příklad: cache miss při čtení adresy 1110

4-bitová adresa, kapacita 8B, 4B bloky



Příklad: plnění řádku blokem z adresy 1110

4-bitová adresa, kapacita 8B, 4B bloky



Vliv velikosti řádku cache na miss-rate

Načítání okolních dat (*spatial prefetching*)

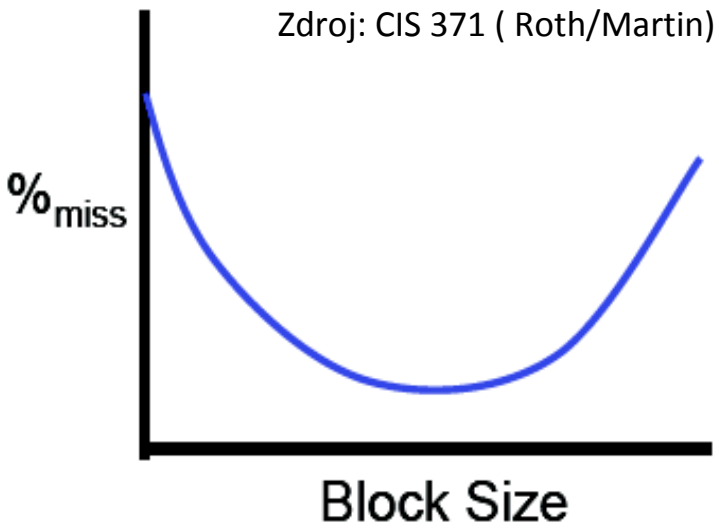
- mění **miss/miss** na **miss/hit** pro bloky se sousedními adresami

Rušení (interference)

- mění **hit** na **miss** pro bloky s nesousedními adresami v sousedních řádcích
  - zmemožnuje současný výskyt v cache
  - limitně cache s jedním řádkem

Vždy se uplatňují oba efekty

- ze začátku dominuje pozitivní vliv načítání okolních dat
  - závisí na velikosti cache
- rozumná velikost řádku je 16-128B
  - závisí na programu



Vliv velikosti řádku na dobu plnění cache

Prodlužuje se při zvětšování řádku t_{miss}?

- přečtení větších řádků by mělo trvat déle...
  - to je pravda, ale...

V případě izolovaných výpadků se t_{miss} nemění

- Critical Word First / Early Restart
- z paměti se nejprve čte právě požadované slovo
  - jakmile dorazí, pipeline může pokračovat
- ostatní slova řádku cache se dočítají později

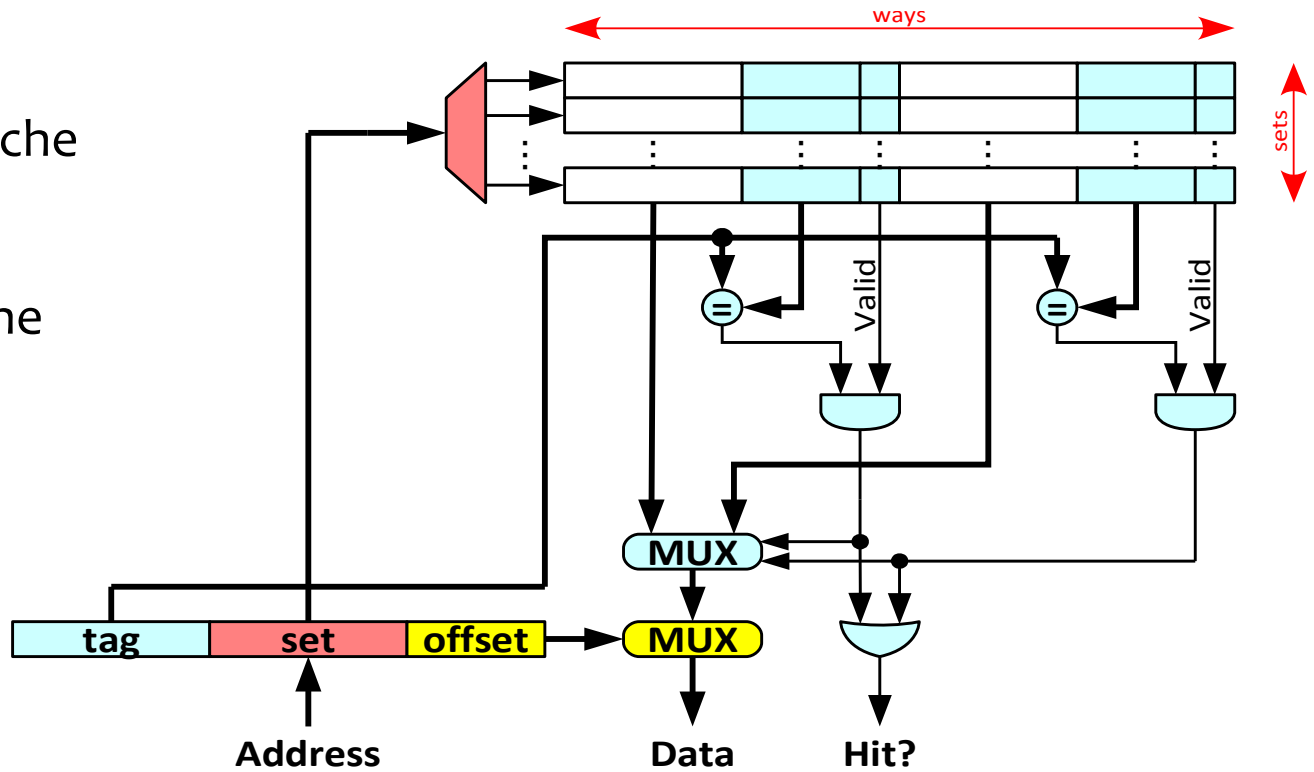
V případě skupin výpadků se t_{miss} zvyšuje

- najednou nelze číst/přenášet/doplňovat více řádků
  - zpoždění se začnou hromadit
- vesměs problém přenosové kapacity mezi pamětí a CPU

Množinově-asociativní mapování paměti do cache

Množinová asociativita (*set associativity*)

- skupiny řádků = *množiny*, řádek v množině = *cesta*
  - příklad: 2-cestná množinově-asociativní cache (*2-way set-associative*)
- pouze 1 cesta
  - přímo mapovaná cache
- pouze 1 množina
  - plně asociativní cache



Omezuje konflikty

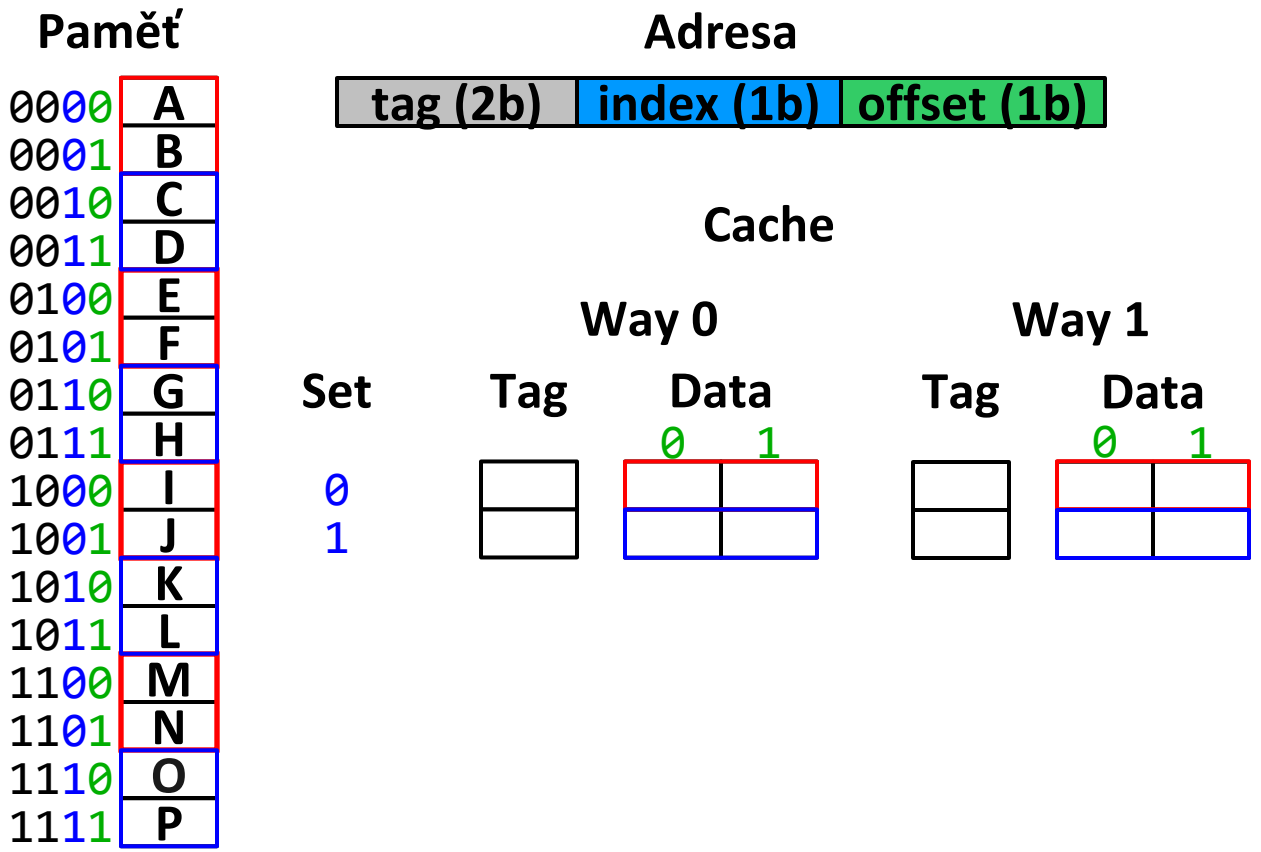
- blok může být ve více řádcích

Prodlužuje t_{hit}

- výběr dat z řádků v množině

Příklad: mapování paměti do 2-cestné cache

4-bitová adresa, kapacita 8B, 2B bloky, 2 cesty



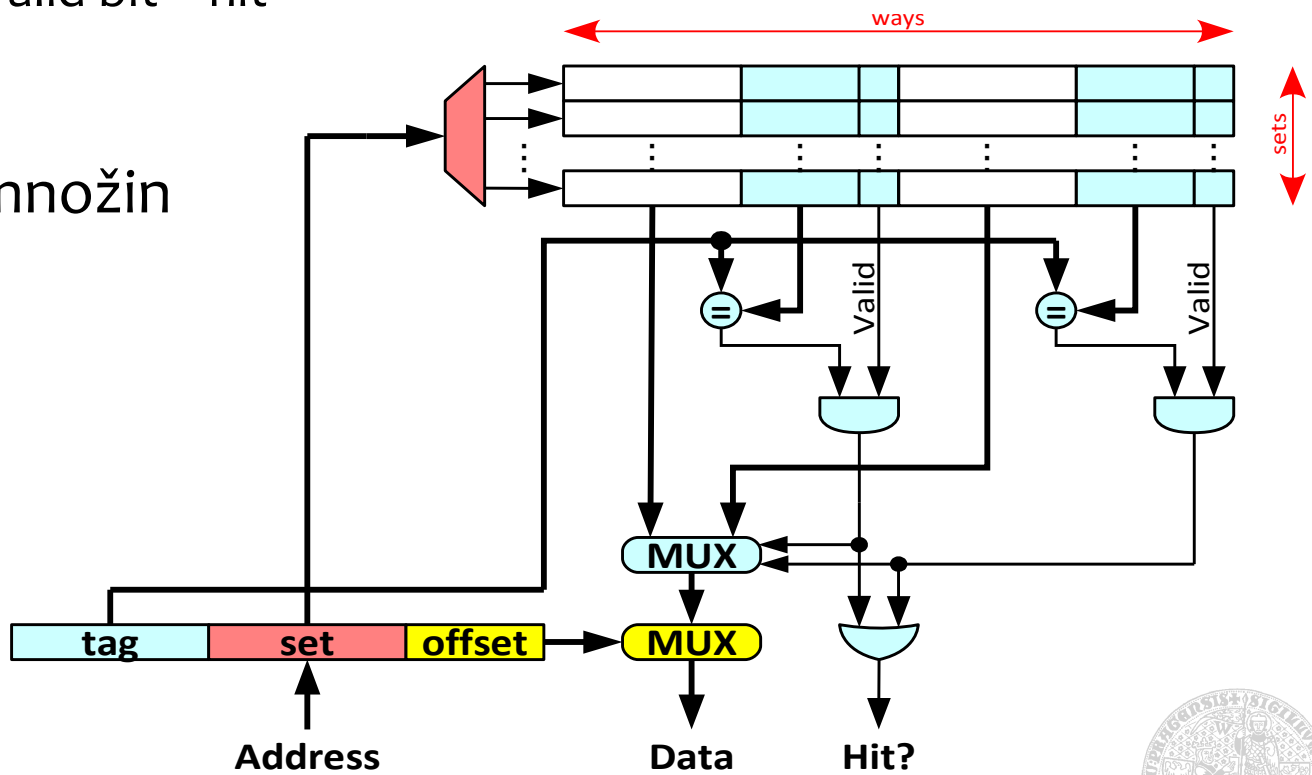
Množinově-asociativní mapování paměti do cache

Vyhledávací algoritmus

1. pomocí index bitů adresy (set) najdeme množinu řádků
2. přečteme data + tagy ze všech řádků v množině současně
3. současně porovnáme tagy řádků s tagem z adresy
   - libovolná shoda + valid bit = hit

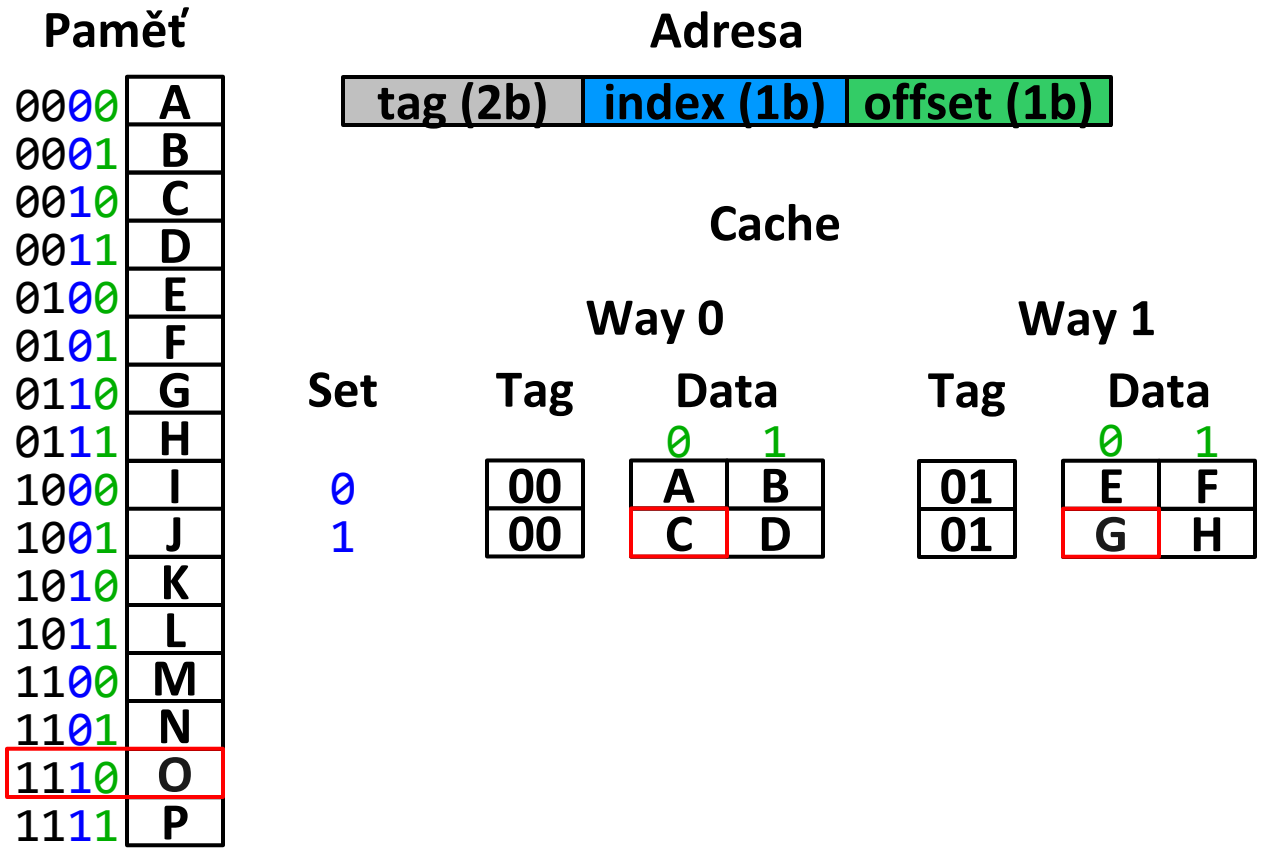
Vliv na tag/index bity

- více cest ⇒ méně množin
- více tag bitů



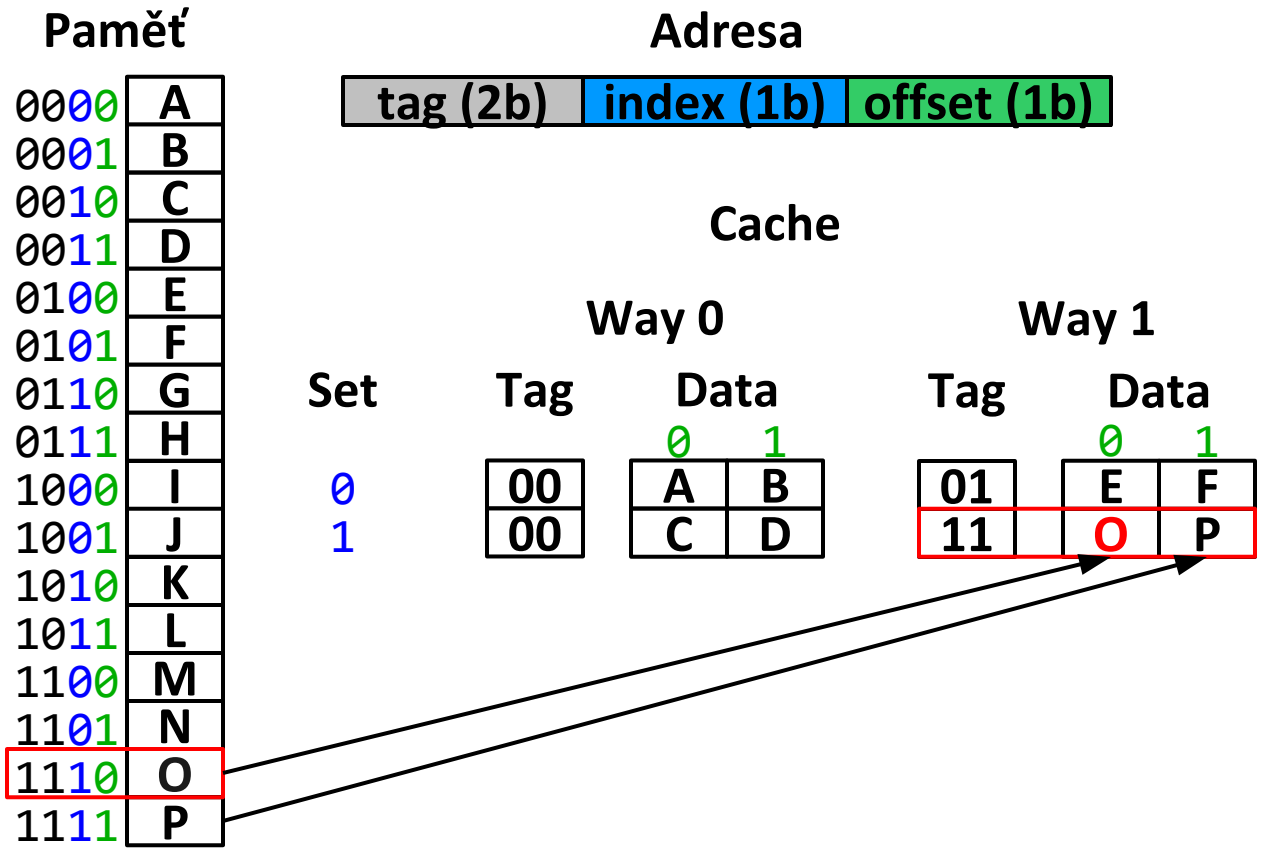
Příklad: cache miss při čtení adresy 1110

4-bitová adresa, kapacita 8B, 2B bloky, 2 cesty



Příklad: plnění řádku blokem z adresy 1110

4-bitová adresa, kapacita 8B, 2B bloky, 2 cesty



Náhrada řádků v cache

Je potřeba načíst nový řádek, ale není volné místo

- obsah některého řádku nutno nahradit obsahem nového řádku (tj. původní obsah "vyhodit")
  - ale kterého?

Přímé mapování

- určeno jednoznačně

(Množinově) asociativní mapování

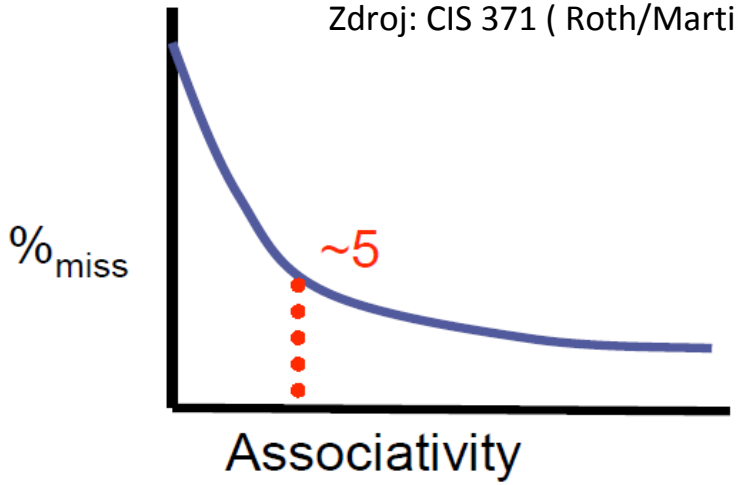
- strategie výběru oběti (později)
  - random, FIFO
  - LRU (*Least Recently Used*) - respektuje časovou lokalitu
  - NMRU (*Not Most Recently Used*) – aproximace LRU

Vliv asociativity cache na miss-rate

Vyšší stupeň asociativity...

- snižuje $\%_{miss}$
  - zákon klesajících výnosů
- prodlužuje t_{hit}
  - čím vyšší stupeň, tím pomalejší

Zdroj: CIS 371 (Roth/Martin)



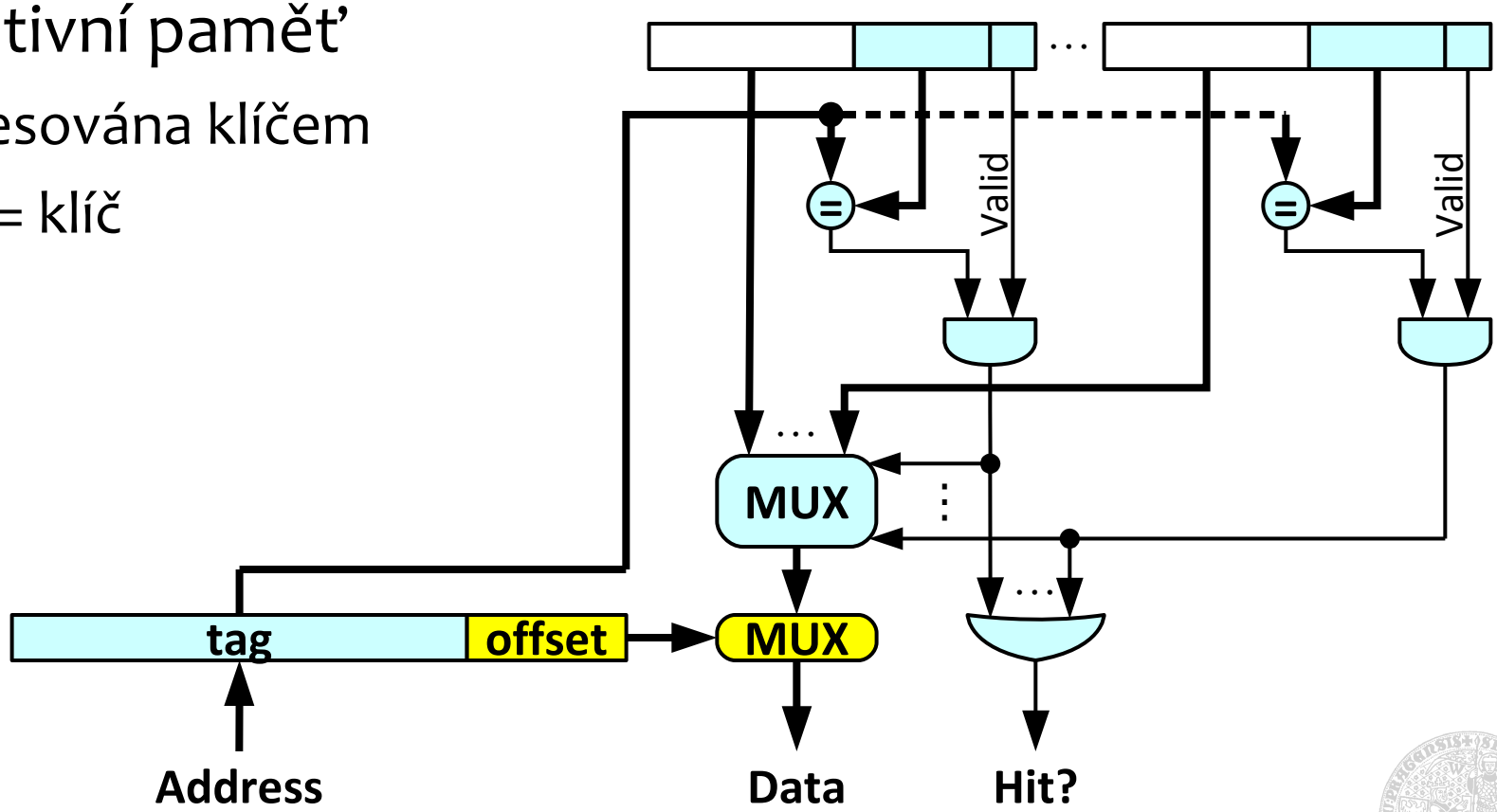
Je možné mít 3-cestně asociativní cache? Klidně...

- pouze velikost řádku a počet množin by měly být mocninou 2
  - zjednodušuje indexaci (prostě se oddělí bity adresy)

Plně asociativní mapování paměti do cache

Množinově-asociativní s 1 množinou

- blok paměti může být v libovolném bloku cache
- všechny bity adresy (až na offset bity) představují tag
- asociativní paměť
  - adresována klíčem
  - tag = klíč



Klasifikace výpadků cache: 3C Model

Compulsory (cold) miss

- "tuhle adresu jsem nikdy neviděl"
- **k výpadku by došlo i v nekonečně velké cache**

Capacity miss

- způsobený příliš malou kapacitou cache
- **k výpadku by došlo i v plně asociativní cache**
- jak se pozná? opakovaný přístup do bloku paměti oddělený alespoň N přístupy do jiných bloků (N je počet řádků v cache)

Conflict miss

- způsobený příliš malým stupněm asociativity
- jak se pozná? **všechny ostatní výpadky**

Miss-rate: ABC

K čemu je 3C model?

- abychom věděli, jak odstranit výpadky
- pokud nevznikají konflikty, zvýšení asociativity nepomůže

A: asociativita (*Associativity*)

- snižuje počet konfliktních výpadků
- prodlužuje $\mathbf{t}_{\mathbf{hit}}$

B: velikost bloku (*Block size*)

- zvyšuje počet konfliktních/kapacitních výpadků (méně řádků)
- snižuje počet studených/kapacitních výpadků (prostorová lok.)
- vesměs neovlivňuje $\mathbf{t}_{\mathbf{hit}}$

C: kapacita (*Capacity*)

- snižuje počet kapacitních výpadků
- prodlužuje $\mathbf{t}_{\mathbf{hit}}$

Jak je to se zápisem dat do cache?

Zatím jsme řešili pouze čtení

- čtení instrukcí, čtení dat
- tag a data řádku je možné číst současně
  - pokud nesedí tag, data neplatí (nevadí, pipeline stall)

Jak je to se zápisem?

- zápis dat do paměti, instrukcí se to netýká
- lze číst tag a zapisovat data současně?
  - pokud nesedí tag, dojde k poškození dat
  - asociativní cache: do které "cesty" se psalo?

Zapisy probíhají ve dvou fázích (zřetězených)

- vyhledání řádku se shodným tagem
- zápis dat do příslušené "cesty" cache

Kdy se propagují zápisy do nižší úrovně hierarchie?

Write-Through: okamžitě

- pokud jsou data v cache, aktualizují se data v řádku
- změna se okamžitě propaguje do nižší úrovně
- vyžaduje více přenosové kapacity (např. opakované modifikace)

Write-Back: při uvolnění řádku

- řádek vyžaduje "dirty" bit indikující změnu dat
  - při uvolnění "clean" řádku se nic neděje
  - při uvolnění "dirty" řádku je nutné ho zapsat do paměti
- write back buffer (WBB): zápis mimo kritickou cestu
  1. vyšli "fill" požadavek na přečtení nových dat do řádku
  2. během čekání zapiš "dirty" řádek do bufferu
  3. jakmile dorazí data, ulož je do cache
  4. zapiš obsah bufferu do nižší úrovně
- vyžaduje méně přenosové kapacity

Jak se řeší výpadek při zápisu?

Kdy (jestli vůbec) může nastat?

- při zápisu do paměti cache neobsahuje řádek s daty bloku paměti, do kterého se zapisuje

Write-allocate

- řádek se nejprve doplní z nižší vrstvy a až poté se do něj zapíše
- snižuje počet výpadků při čtení (lokalita)
- vyžaduje dodatečnou přenosovou kapacitu
- běžně používané (obzvláště u write-back cache)

Write-non-allocate

- pouze zápis do nižší úrovně, řádek se do vyšší vrstvy nenačítá
- v podstatě opak write-allocate

Shrnutí

Výkon paměti dominuje výkon procesoru

Paměťová bariéra (*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

Cache: iluze rychlé a velké 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

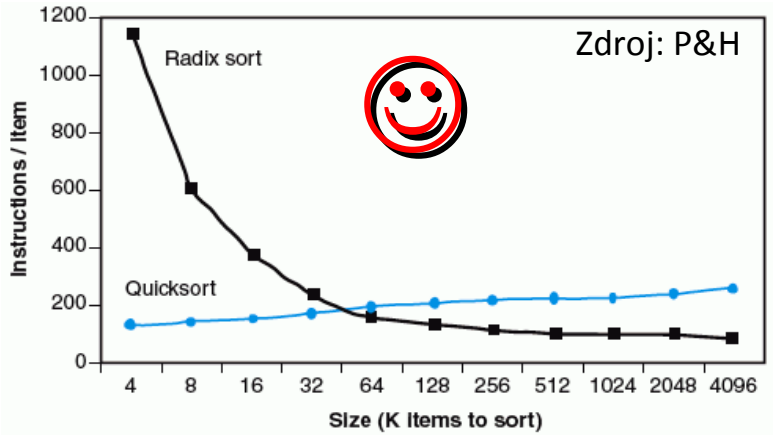
ABC: základní parametry cache

- asociativita, velikost bloku, kapacita

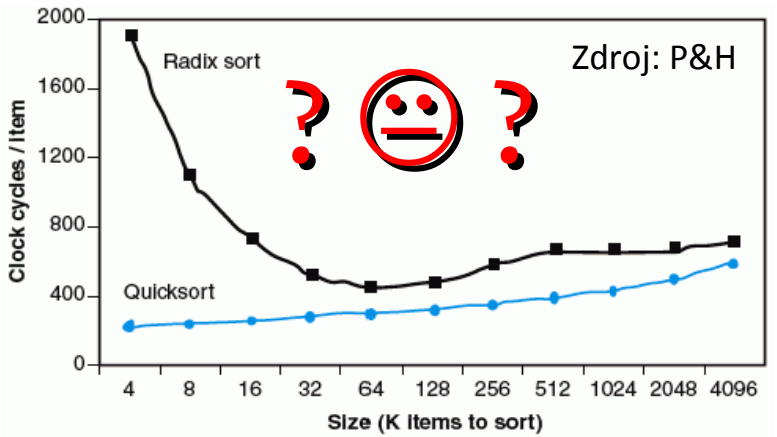
Proč je důležité o cache vědět? (1)

Příklad: LaMarca, Ladner (1996)

- Quick Sort vs. Radix Sort
  - ***O(n log n)*** vs. ***O(n)***
  - teoreticky "není co řešit"



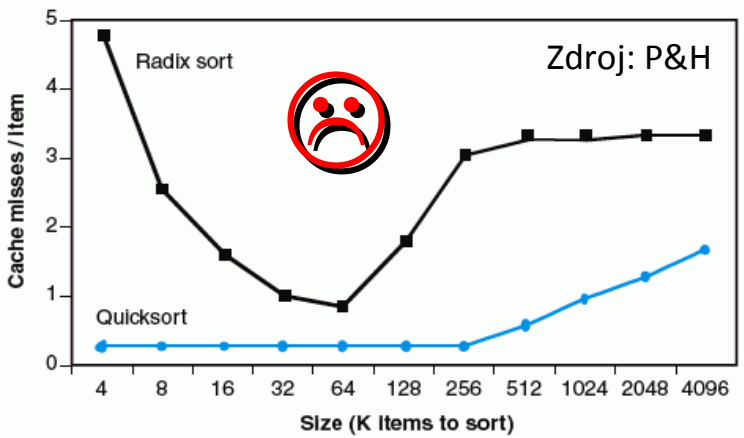
- ***Jenže... Quick Sort se pro větší množství dat ukázal rychlejší...***



Proč je důležité o cache vědět? (2)

Quick Sort vs. Radix Sort

- způsob přístupu k datům v implementaci algoritmu Radix Sort způsoboval příliš mnoho výpadků cache



Řešení: návrh algoritmů s ohledem na cache

- úprava algoritmu Radix Sort tak, aby pracoval s daty nejprve v rámci bloku paměti, který je v cache (cache line)