

Úvod do organizace počítače

Hierarchie paměťového
systému

Fyzická a virtuální paměť,
cache

Přehled pamětí

- Uložení dat a instrukcí
- Model - lineární pole 32-bitových slov
- MIPS: 32-bitová adresa => 2^{32} slov
- Doba odezvy je pro každé slovo stejná
- Paměť lze číst/zapsat v 1 cyklu 😊
- Typy pamětí
 - Fyzická: *Instalována v počítači*
 - Virtuální: Disková paměť, která se chová jako hlavní paměť
 - Cache: Rychlá paměť pro dočasné umístění dat nebo instrukcí

Instrukce pro práci s pamětí

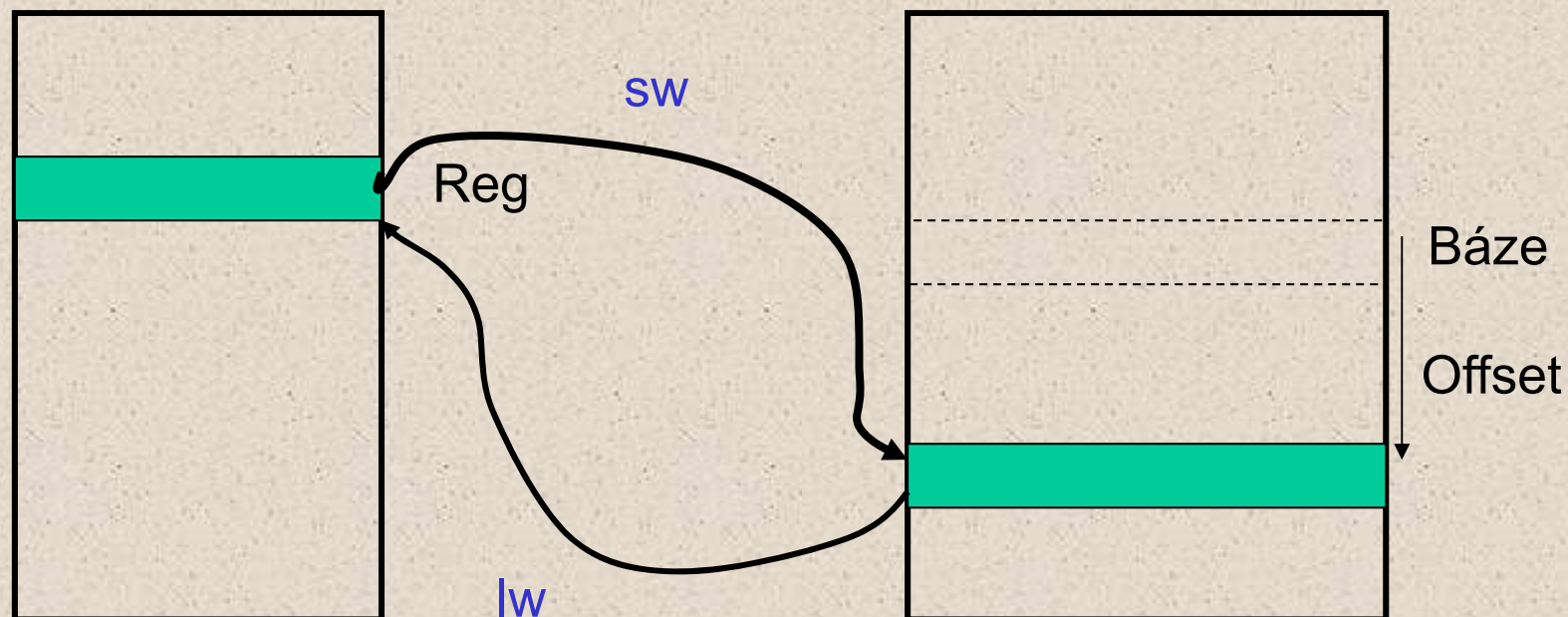
- lw reg, offset(base)
- sw reg, offset(base)

Load => přenos paměť do registru

Store => přenos registr do paměti

Registrový soubor

Paměť



Úvod do hierarchie

- Předpokládejme, že jste navštívili knihovnu a hledáte materiály o historii počítačů.
 - Procházíte mezi policemi a vybíráte knihy, které se týkají vašeho zájmu. Pokládáte je na pracovní stůl.
 - K policím jste šli proto, abyste si přinesli to, co budete brzy potřebovat.
 - A také proto, abyste mohli chvíli nerušeně pracovat, aniž byste se museli znovu zvednout a navštívit police s knihami.

Lokalita

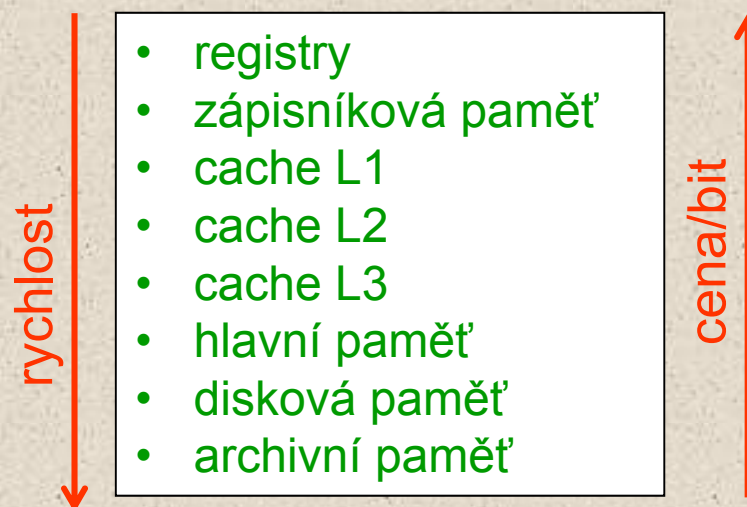
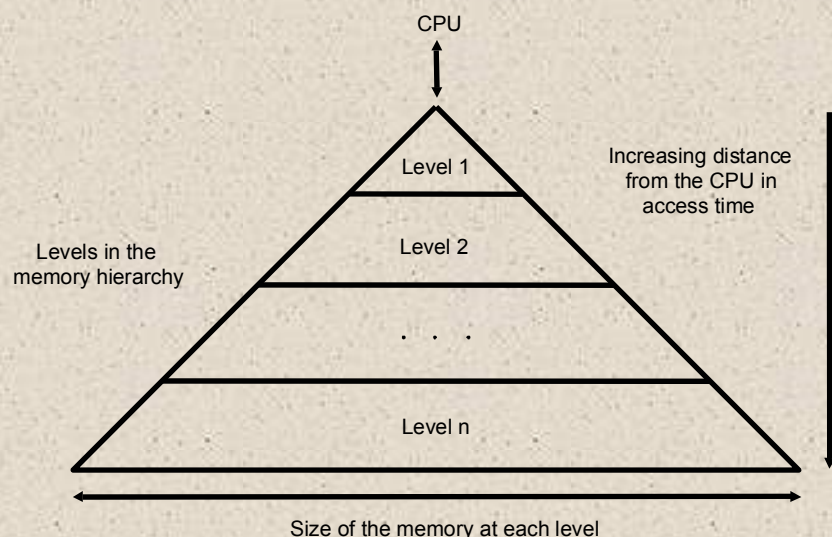
- Tento příklad znázorňuje *princip lokality* – programy přistupují v daném krátkém časovém intervalu k relativně malé části adresního prostoru (stejně jako vy jste přinesli jen malou část knih).
- Existují dva typy lokality ...
 - *Časová lokalita* – je-li položka referencována, má tendenci být referencována v krátkém časovém úseku opět. (knihu, kterou jste právě odložili na stůl pravděpodobně zase brzy vezmete do ruky).
 - *Prostorová lokalita* – Je-li položka referencována, budou patrně referencovány i s ní sousedící položky (naleznete-li knihu, je pravděpodobné, že vás mohou zajímat i knihy, ležící v jejím těsném okolí).

Technologie pamětí

- Programátoři vyžadují od nepaměti obrovské kapacity velmi rychlých pamětí.
- V současné době existuje velká řada nejrůznějších typů pamětí, různé rychlosti, ceny a výkonu. Všechny ale mají jednu tendenci. Čím jsou rychlejší, tím mají vyšší cenu a větší spotřebu.
- A tak namísto pořízení rozsáhlé rychlé paměti za vysokou cenu byl uveden koncept paměťové hierarchie, ve kterém počítače používají paměti různých kapacit a typů.

Technologie pamětí

- V současné době se ke stavbě hierarchického paměťového systému používají hlavně tři typy pamětí...
 - Hlavní paměť používá technologii DRAM (dynamická RAM).
 - Úrovně blíže k CPU (L1/L2/L3 cache) jsou statické RAM (SRAM). Jsou mnohem dražší, ale také mnohem rychlejší.
 - Největší kapacitu, ale také nejmenší rychlost mají paměti, používané na nejnižší úrovni. (např. harddisky). Tato technologie je velmi levná, ale doba přístupu je dlouhá.



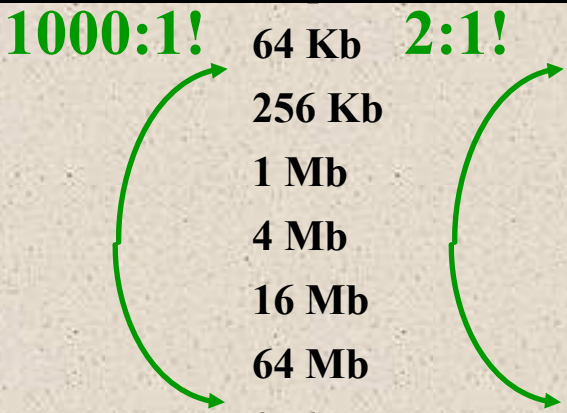
Technologie pamětí

Technologie paměti	Typická doba přístupu	\$ za GB v roce 2004
SRAM	0.5–5 ns	\$4000–\$10,000
DRAM	50–70 ns	\$100–\$200
Magnetický disk	5,000,000–20,000,000 ns	\$0.50–\$2

Trendy technologie

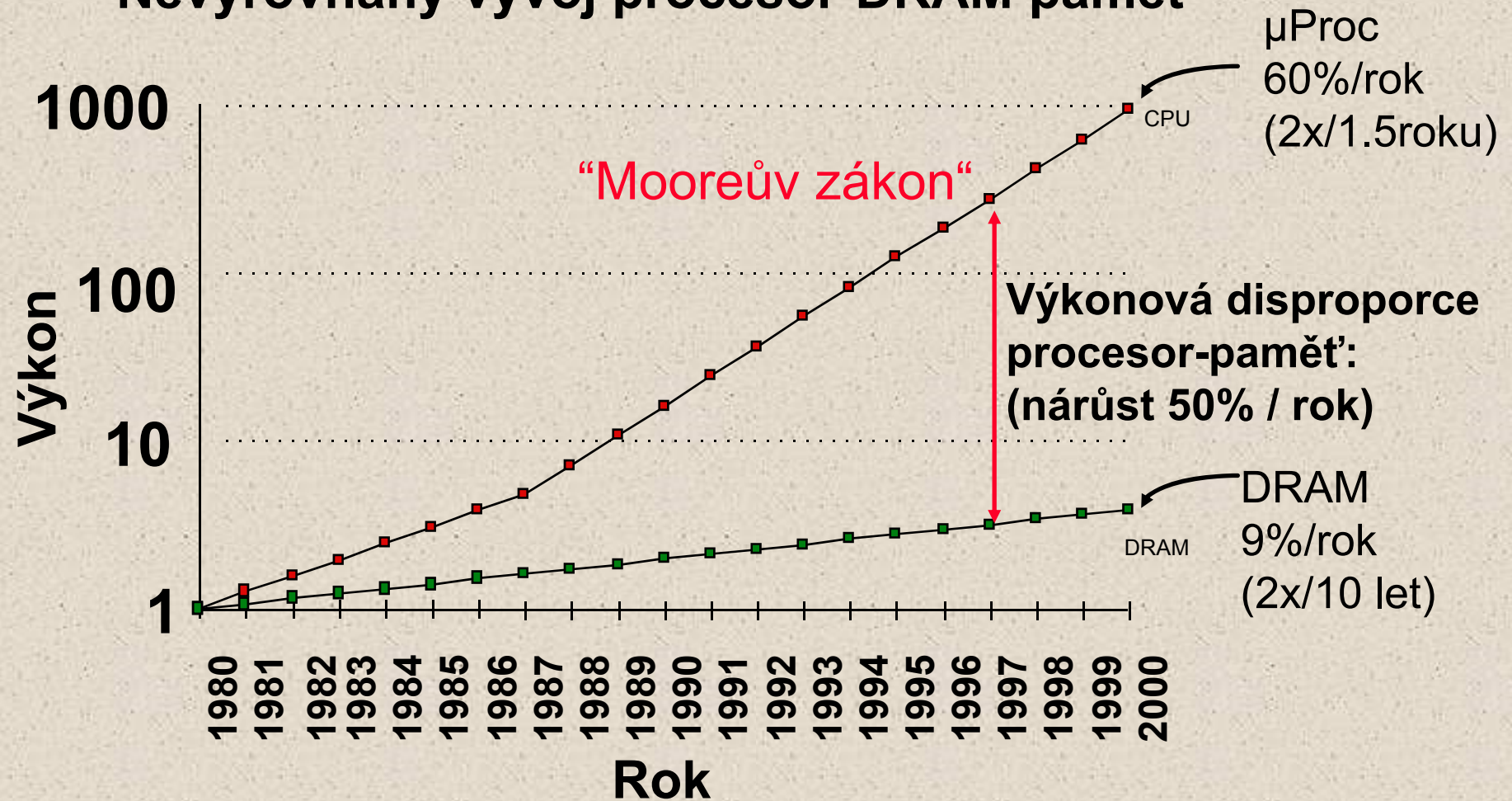
	Kapacita	Rychlost (latence)
Logika:	2x za 3 roky	2x za 3 roky
DRAM:	4x za 3 roky	2x za 10 let
Disk:	4x za 3 roky	2x za 10 let

DRAM		
Rok	Kapacita	Doba cyklu
1980	64 Kb	250 ns
1983	256 Kb	220 ns
1986	1 Mb	190 ns
1989	4 Mb	165 ns
1992	16 Mb	145 ns
1995	64 Mb	120 ns
1998	256 Mb	100 ns
2001	1 Gb	80 ns



Jak je to s pamětí?

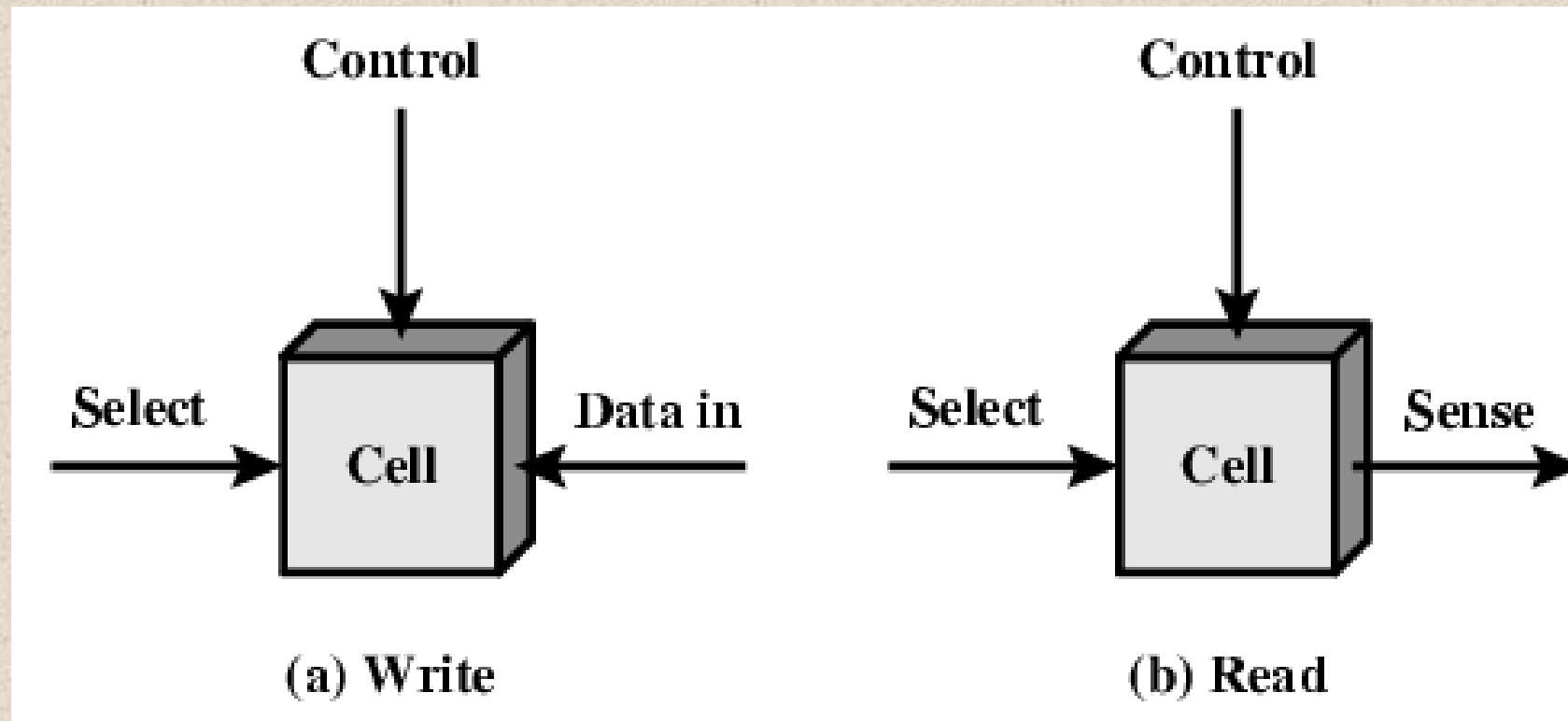
Nevyrovnaný vývoj procesor-DRAM paměť



Paměť

- SRAM:
 - Informace je uchována pomocí dvojice invertujících hradel
 - Velmi rychlé, ale větší nároky než DRAM (4 až 6 tranzistorů)
- DRAM:
 - Informace je uchována jako náboj na kondenzátoru (musí se zotavovat)
 - Velmi malá buňka, ale pomalejší než SRAM (poměr 1: 5 až 1:10)

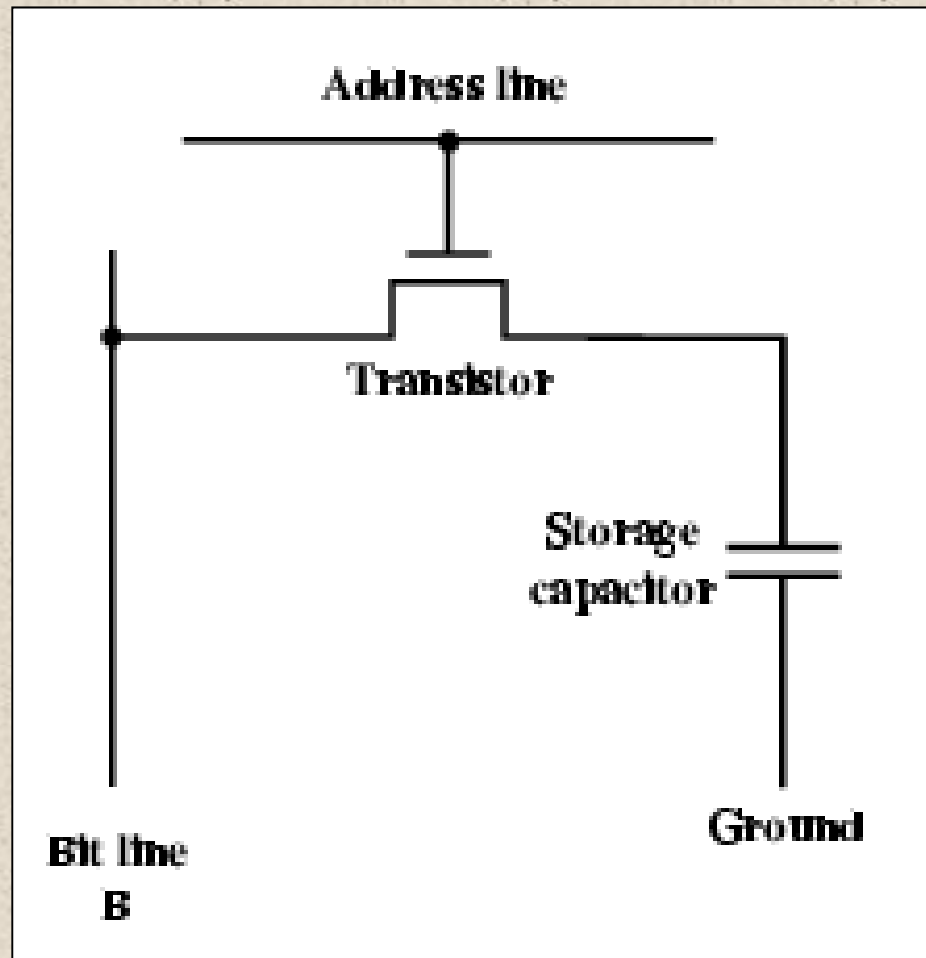
Operace buňky paměti



Dynamická RAM

- Bity jsou uloženy ve formě náboje na kapacitě
- Náboj se „vytrácí“ vlivem svodových proudů
- Je třeba zotavovat i když je paměť napájena
- Jednoduchá konstrukce
- Malý rozměr „bitu“
- Levnější než statická
- Jsou třeba zotavovací obvody
- Pomalejší
- Hlavní paměť
- V podstatě analogová záležitost
 - Úroveň náboje určuje hodnotu

Struktura dynamické RAM



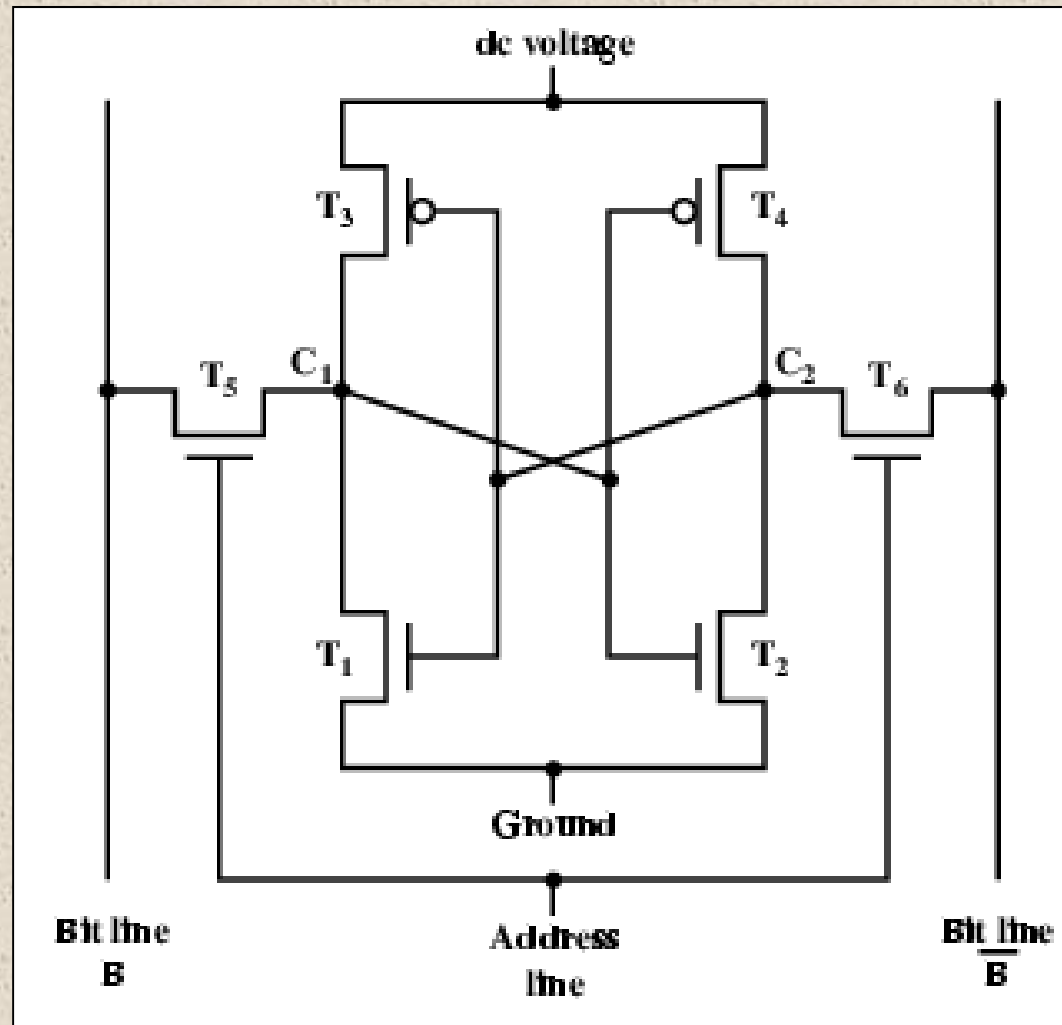
Operace DRAM

- Adresní linky jsou aktivní při zápisu nebo čtení
 - Tranzistorový spínač sepnut (protéká proud)
- Zápis
 - Napětí bitové linky
 - Vysoká úroveň pro 1 nízká pro 0
 - Pak je aktivována adresní linka
 - Přenos náboje do kondenzátoru
- Čtení
 - Vybere se adresní linka
 - tranzistorový „spínač“ se sepne
 - Náboj kondenzátoru se přivede pomocí bitové linky na čtecí zesilovače
 - Porovnává se s referenční úrovní aby se určila 1 nebo 0
 - Náboj kondenzátoru je třeba obnovit

Statická RAM

- Bity jsou uloženy jako stav klopného obvodu
- Svodové proudy nemají vliv
- Není třeba zotavovat
- Složitější konstrukce buňky
- Větší rozměr buňky
- Vyšší cena
- Nejsou třeba zotavovací obvody
- Rychlejší
- Cache
- Čistě digitální princip
 - Využívají se klopné obvody (flip-flops)

Struktura buňky statické RAM



Operace statické RAM

- Uspořádání tranzistorů poskytuje stabilní logický stav
- Stav 1
 - C_1 vysoká úroveň, C_2 nízká úroveň
 - T_1 T_4 nevedou, T_2 T_3 vedou
- Stav 0
 - C_2 vysoká úroveň, C_1 nízká úroveň
 - T_2 T_3 nevedou, T_1 T_4 vedou
- Adresní linka a tranzistory T_5 T_6 řídí spínání
 - Zápis – „dopraví“ hodnotu do B & komplement do \overline{B}
 - Čtení – hodnota se objeví na lince B

SRAM kontra DRAM

- Obě ztrácí informaci po vypnutí napájení
 - Pro uchování informace je třeba zachovat napájení
- Dynamická buňka
 - jednoduchá konstrukce, menší
 - Vyšší hustota
 - Levnější
 - Třeba zotavovat
 - Používá se pro stavbu větších paměťových celků
- Statická buňka
 - Rychlejší
 - Cache

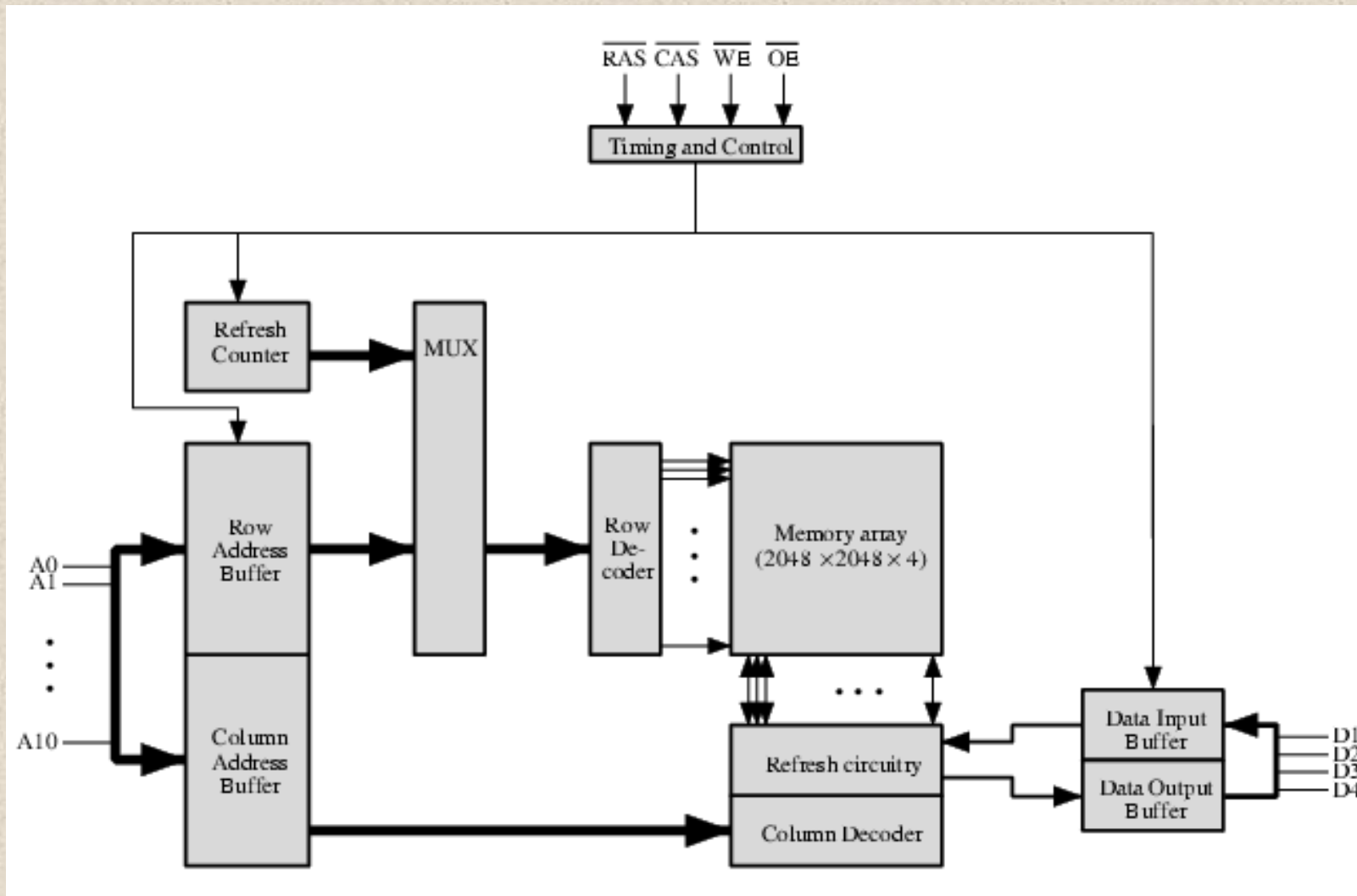
Detaily organizace

- 16 Mbitový čip může být organizován jako 1M 16 bitových slov
- Organizace „bit per čip“ využívá 16 Mbitové čipy, s 1 bitem 1 v každém slově
- 16Mbit čip lze také organizovat jako pole 2048 x 2048 x 4 bitů
 - Redukuje se počet adresních pinů
 - Multiplexují se adresy řádek a sloupečků
 - 11 pinů pro adresy ($2^{11}=2048$)
 - Přidáním jednoho pinu adresy se adresovací schopnost zvýší 4 krát

Proces zotavení

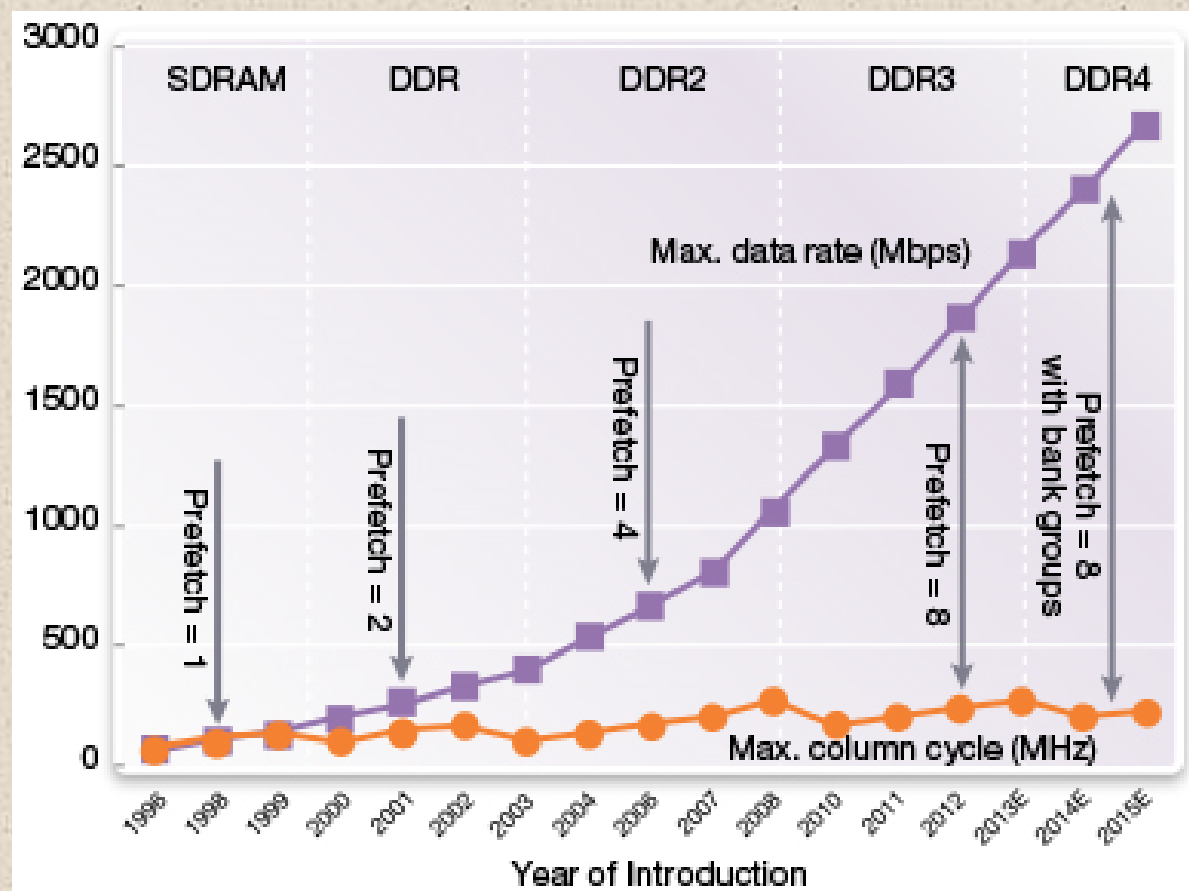
- Zotavovací obvody umístěny na čipu
- Čip je během zotavení „nedostupný“
- Probíhá po řádcích
- Čtení a zpětný zápis
- Zabírá čas
- Snižuje celkový výkon

Typická 16 Mb DRAM (4M x 4)

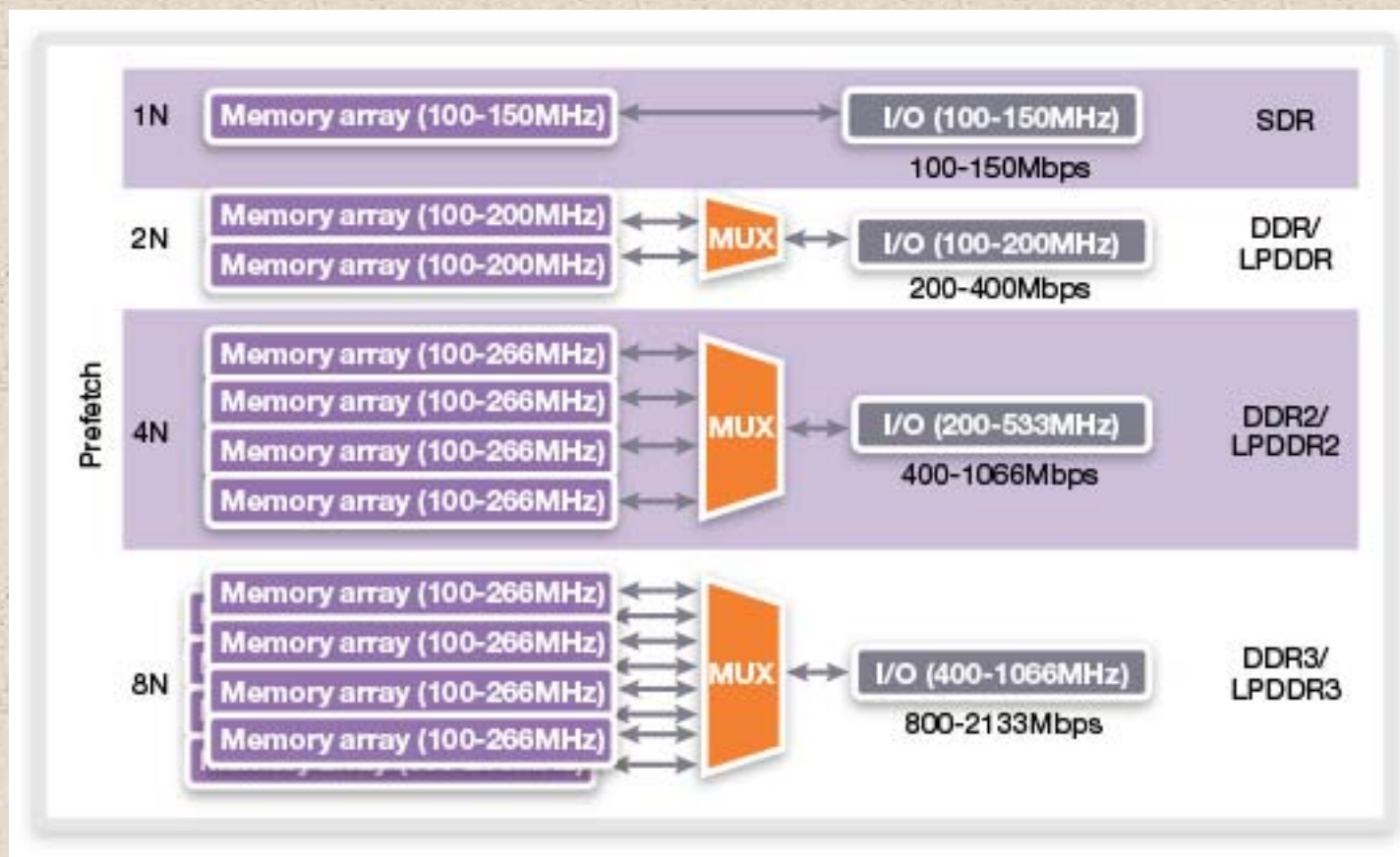


Zvýšení výkonu SDRAM vlivem „Prefetch“

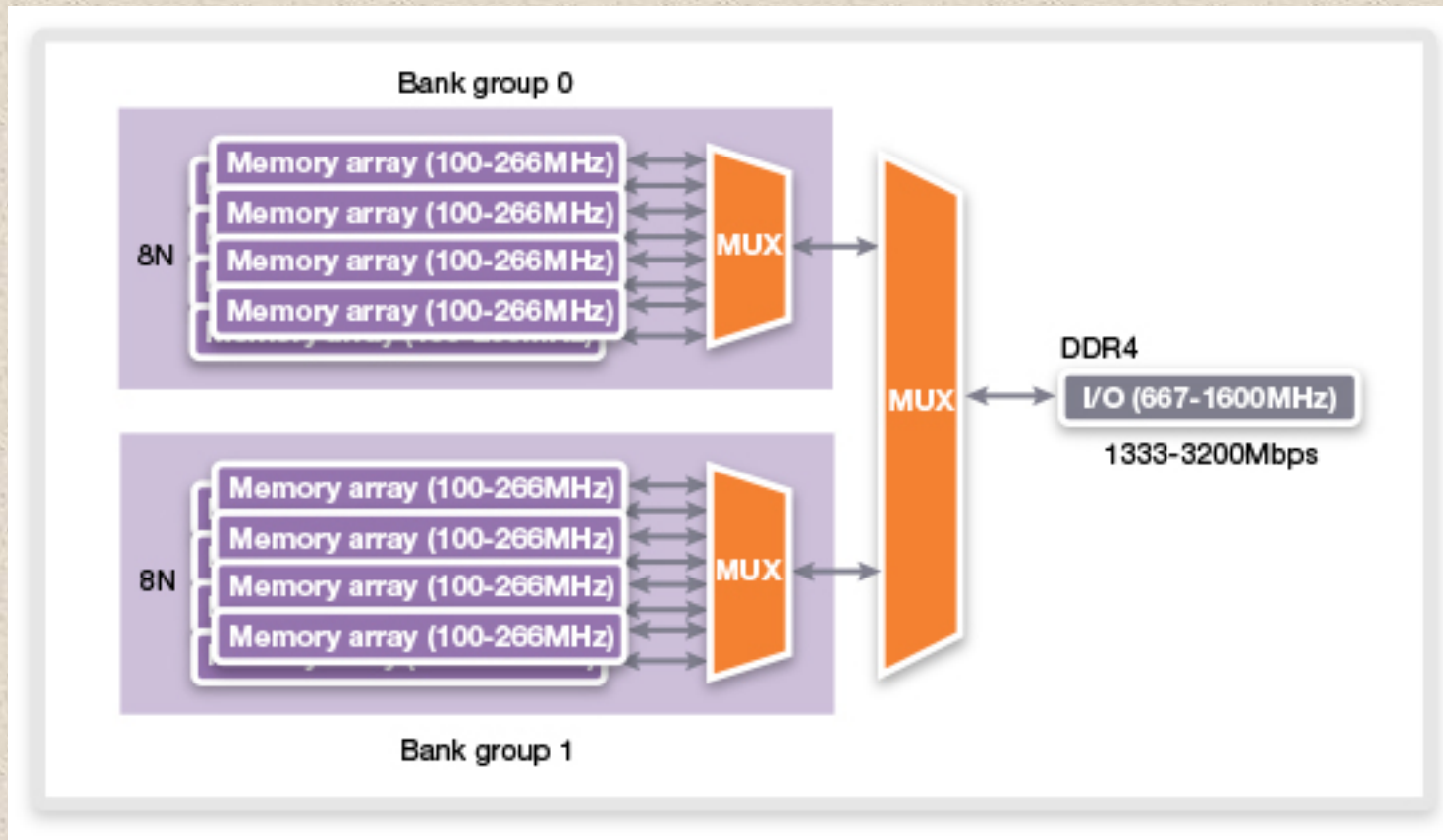
Další zvyšování výkonu je kryto hlavně paralelní činností více kanálů.



Historický vývoj DRAM „Prefetch“



Osminásobný Prefetch u DDR4



Zdroje:

JEDEC Standard: DDR4 SDRAM (JESD79-4). JEDEC Solid State Technology Association. September 2012.

Hierarchie paměťového systému

1. Registry

- Rychlá interní paměť v procesoru
- *Rychlost* = 1 cykl hodin CPU
- *Perzistence* = několik cyklů
- *Kapacita* ~ 0.1K až 2K bytů

2. Cache

- Rychlá paměť interní *nebo* externí (k procesoru)
- *Rychlost* = Několik cyklů hodin CPU
- *Perzistence* = desítky až stovky cyklů *pipeline*
- *Kapacita* ~ 0.5MB až 2MB

3. Hlavní paměť

- Hlavní paměť obvykle externí vzhledem k procesoru, < 16GB
- *Rychlost* = 1-10 hodin CPU
- *Perzistence* = milisekundy až dny

4. Disková paměť

- **Velmi pomalá – doba přístupu = 1 až 15 milisekund**
- Použití pro odkládání dat (*instrukcí*) z hlavní paměti

Proč hierarchie?

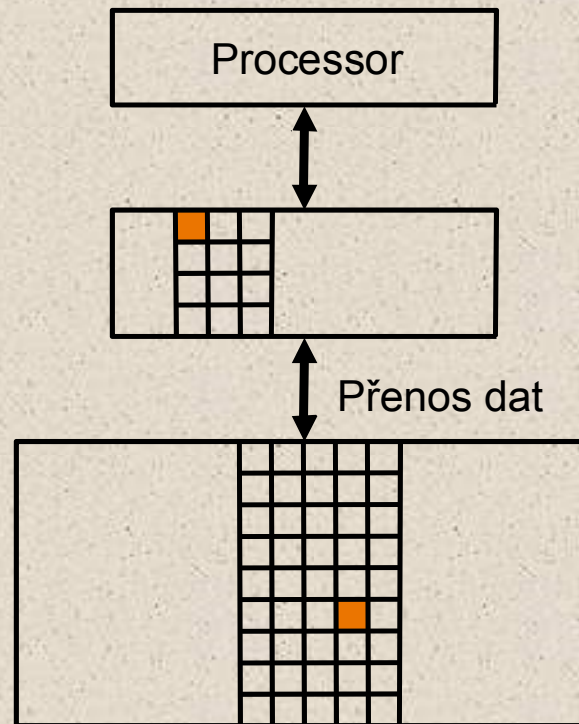
- Vzhledem k rozdílu v ceně a rychlosti je výhodné vybudovat hierarchii úrovní s rychlejšími paměťmi blíže k procesoru a s levnějšími paměťmi na nižších úrovních.
- Cílem tohoto uspořádání je prezentovat uživateli/programátorovi paměťový systém s kapacitou odpovídající levné technologii, ale s rychlostí, která odpovídá rychlým a drahým paměťovým prvkům.

Organizace hierarchie

- U víceúrovňové organizace je každá úroveň organizována jako podmnožina libovolné nižší úrovně. Všechna data obsahuje nejnižší úroveň.
- Data se kopírují mezi sousedními úrovněmi. Přenášejí se po *blocích*. Blok - minimální jednotka informace, která se může nacházet v každé úrovni hierarchie.
- Jsou-li data požadovaná procesorem přítomná v nejvyšší úrovni cache, nastane tzv. *cache hit*. Nejsou-li nalezena, musí být získána z nižší úrovně. Tomu se říká *cache miss*.

Organizace hierarchie

- Mezi sousedními úrovněmi dochází k výměně informace (přenosu bloků u cache, popř. stránek u VM).



Cache paměť - pojmy

- **Blok:** Skupina slov **Set:** Skupina bloků
- **Hit** 😊
 - *Je-li hledán blok B v cache a je nalezen*
 - Hit Time: čas potřebný k nalezení bloku B
 - Hit Rate: četnost přístupů, kdy B je nalezen v cache
- **Miss** ☹️
 - *Je-li hledán blok B v cache a není nalezen*
 - Miss Rate: četnost přístupů, kdy B je hledán v cache a není nalezen
- **Příčiny:**
 - Špatná strategie výměny bloků
 - Špatná lokalita prováděných programů

Příklad: Cache Miss

- Předpokládejme dvouúrovňový systém s cache. Obsahuje hlavní paměť a cache (která je rychlejší a má mnohem menší kapacitu).
- Předpokládejme, procesor vyžaduje data z paměťového místa X_n . Tato data právě nejsou v cache => vzniká miss a data musíme čerpat z hlavní paměti (a kopírovat je do cache).

X 4
X 1
X n - 2
X n - 1
X 2
X 3

a. Před referencí X_n

X 4
X 1
X n - 2
X n - 1
X 2
X n
X 3

b. Po referenci X_n

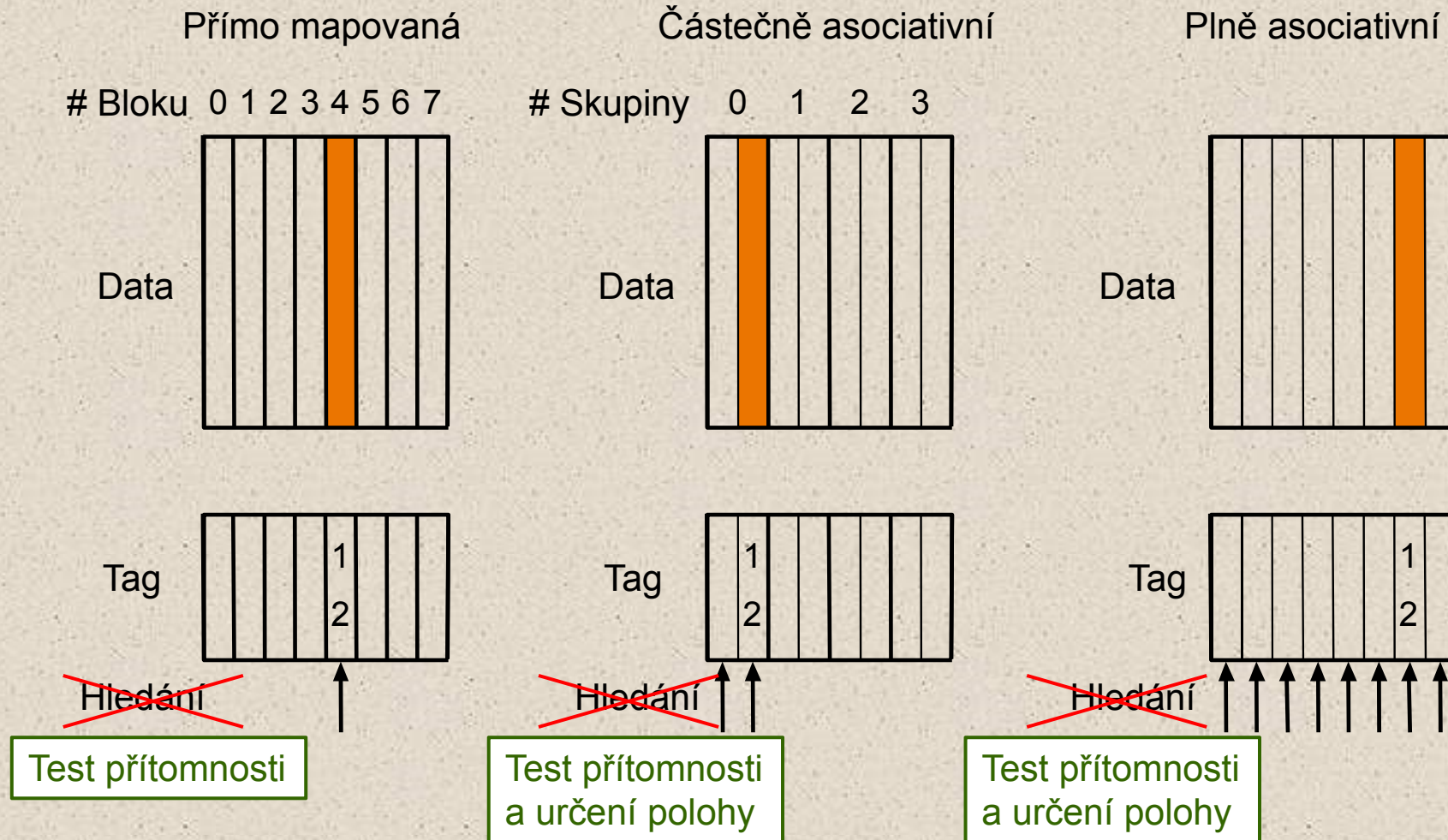
Příklad trochu podrobněji

- Jestliže rozebereme předchozí příklad trochu podrobněji, vznikají dvě otázky...
 - Jak se dozvíme, že data jsou v cache?
 - Jestliže ano, pak kde?
- Tyto otázky můžeme vyřešit současně, jestliže každému slovu v paměti přidělíme slovo v cache, jehož poloha je odvozena od adresy slova v hlavní paměti.
 - Tato organizace se nazývá *přímo mapovaná cache*.
 - Je to jednoduché mapování, u kterého je každému místu v paměti přiřazeno právě jedno místo v cache.
 - Transformační předpis je jednoduchý ...
(Adresa bloku) modulo (# bloků v cache)

Cache paměť

- Princip *lokality programů*
 - *lw* a *sw* během daného krátkého časového úseku přistupují k velmi malé části paměti
- Cache *obsahuje* kopie úseků paměti => dočasná paměť
 - 3 **mapovací schémata**: Dán paměťový blok B
 - **Přímo mapovaná**: cache *jedna k jedné* -> paměťová mapa (B může být obsažen jen v jediném bloku cache)
 - **Částečně asociativní (vícecestná)**: jeden z n bloků cache obsahuje B
(n ... malé číslo, obvykle $n=2, 4, 8$)
 - **Plně asociativní**: Kterýkoli blok cache může obsahovat B
- Různá mapovací schémata pro různé způsoby přístupu na data

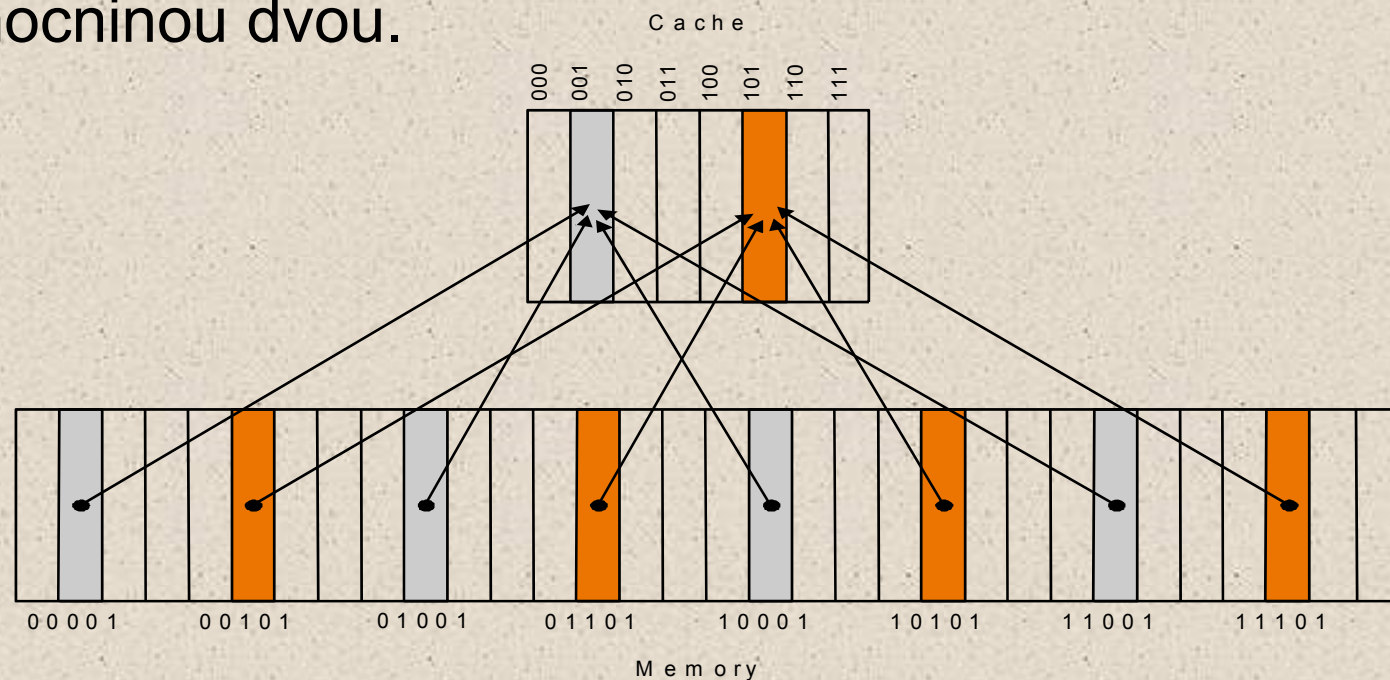
Tři strategie organizace cache



Přímo mapovaná cache

Tato organizace je velmi jednoduchá, protože operaci modulo lze provést jednoduše uplatněním dolních $\log_2(\text{velikost cache v blocích})$ bitů adresy.

- cache se adresuje dolní částí adresního pole.
- to je pravda jen tehdy, je-li počet položek v cache mocninou dvou.



Význam pole „tag“

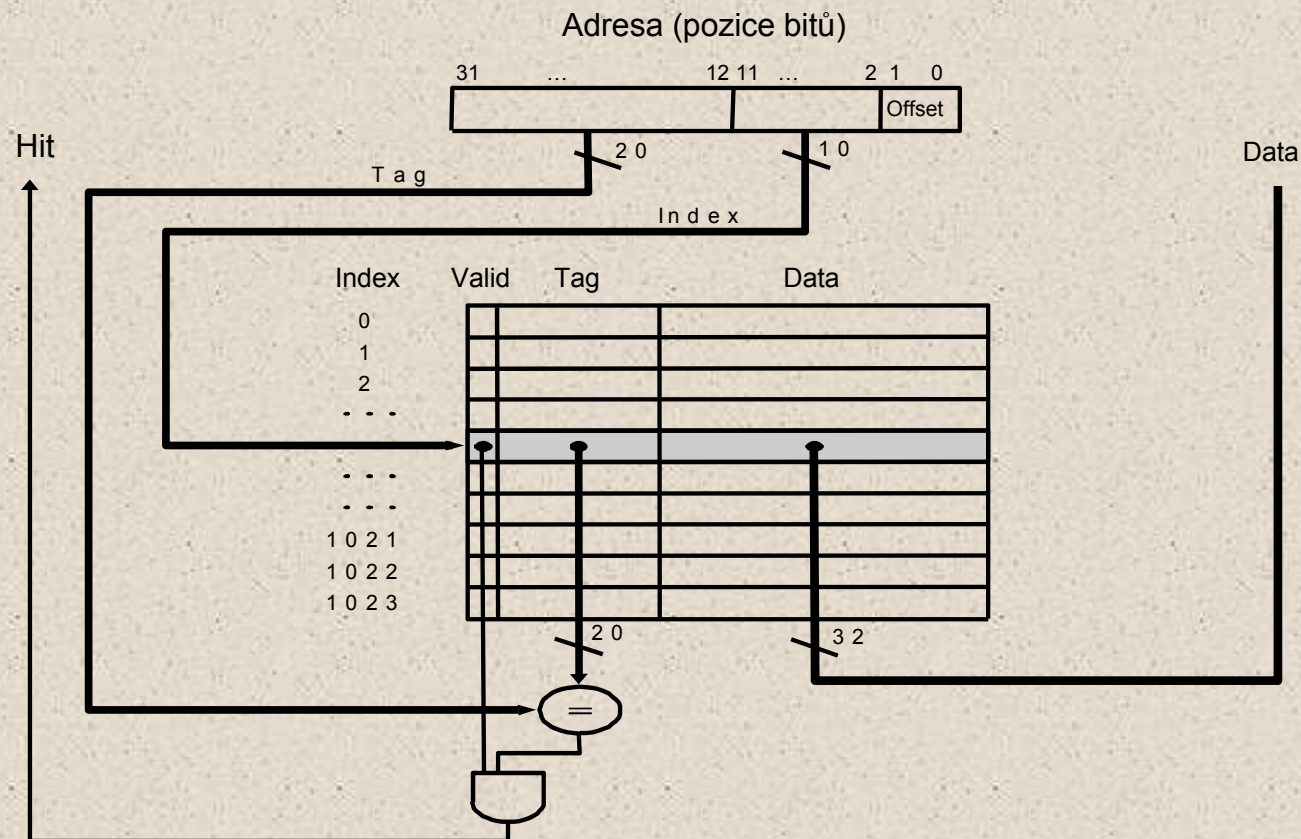
- To znamená, že každý blok hlavní paměti se mapuje na jeden jediný blok cache, ale na jeden vstupní bod cache se mapuje větší množství bloků hlavní paměti.
 - Jak můžeme určit, že právě požadovaný blok je umístěný v cache?
- Zajistíme to doplněním souboru *tagů* do cache. Ke každému bloku v cache jednoduše uložíme ještě jeho horní část adresy. Výběr bloku provedeme polem *index* a zkontrolujeme, zda horní část adresy požadovaného bloku je totožná s polem *tag*, uloženým v cache.

Nutnost použití příznaku platnosti

- Vedle pole *tag* je třeba zaznamenat, zda blok v cache obsahuje platnou informaci.
 - Při startu procesoru bývá cache naplněna náhodnými daty.
 - I po chvíli činnosti, nemusí být obsah všech položek v cache platný.
- Tento problém lze jednoduše odstranit přidáním bitu platnosti (valid bit) ke každému bloku. Není –li tento bit nastaven, nemůže být blok „nalezen“ (nenastane **hit**).

Příklad přímo mapované cache

- Příklad přímo mapované cache paměti s 1024 položkami, každá o šířce 1 slovo. Nižší bity jsou použity pro výběr řádky cache.



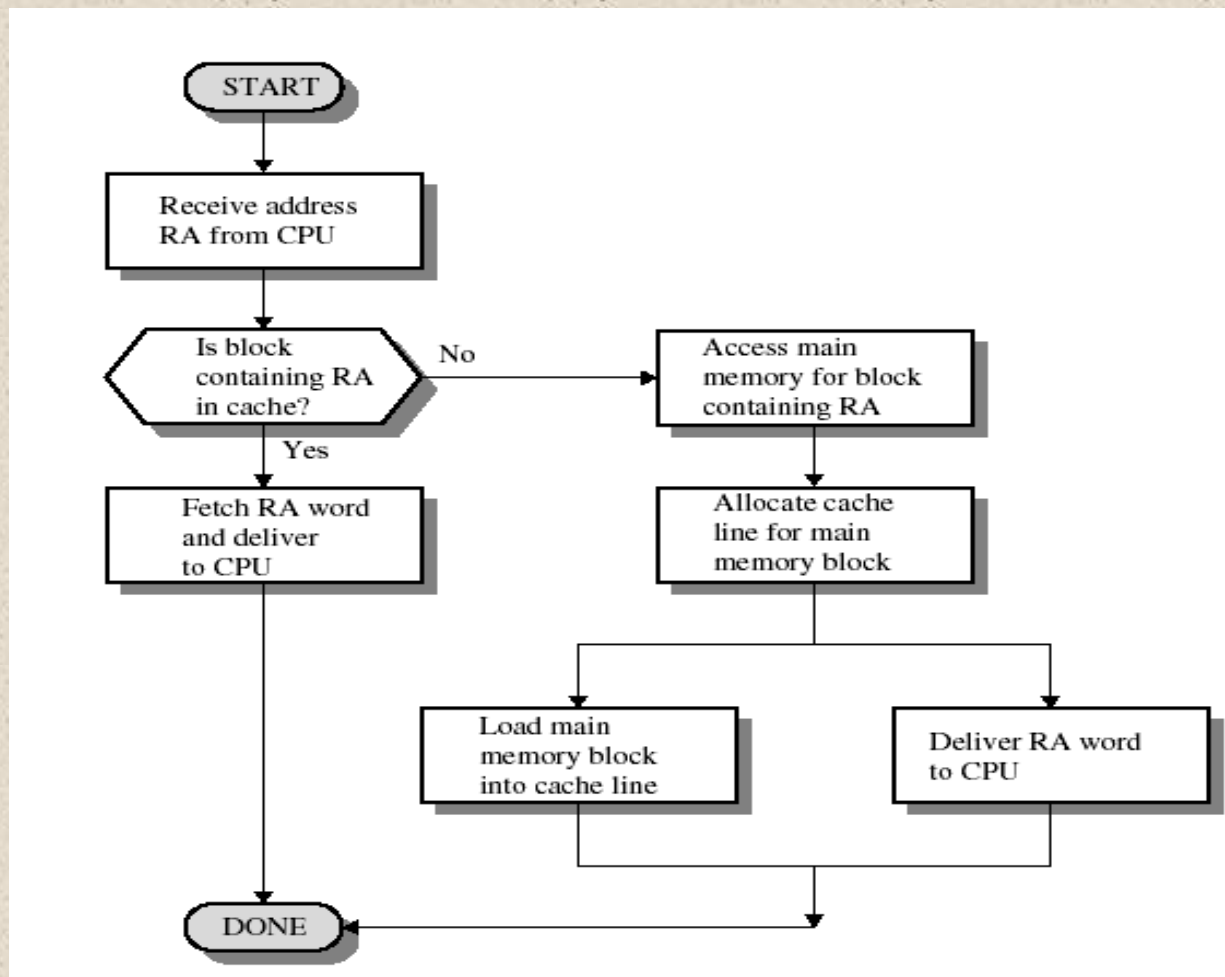
Čtení z cache

- Co se stane, když procesor vyžaduje data z paměti (operace čtení)?
 - Jedná-li se o „cache hit“, neděje se nic zvláštního, řídicí jednotka pouze vyšetřuje příchod signálu „hit“.
 - Jedná-li se o „cache miss“, musí se CPU s tímto problémem zabývat.

Obvyklé řešení:

CPU se pozastavuje a čeká, dokud nejsou data přenesena z nižší úrovně (z paměti a nebo z cache nižší úrovně).

Cache – operace čtení



Cache miss x zastavení pipeline

- Je-li CPU zastavena, každý registr si musí uchovat svoji hodnotu. Je to jednodušší, než pozastavit pipeline, protože se zastavuje vše a nemusí pokračovat provádění některých instrukcí, jako je tomu v případě pozastavení samotné pipeline.
- Řízení situace „cache miss“, přesun dat z nižší úrovně do cache obstarává vyhrazený řadič.
- Čteme-li z datové paměti, jednoduše zastavíme celý procesor, dokud nejsou data k dispozici.

Instrukční výpadek (miss)

- Uvažujme případ, že nastane miss v instrukční cache. Jaké úkoly bude vyhrazený řadič plnit?
 - Zašle adresu chybějící instrukce (aktuální PC – 4) do paměti.
 - Vyžádá operaci čtení z hlavní paměti a čekání na dokončení čtení.
 - Zapíše položku do cache (uloží datový blok, horní část adresy do pole tag a nastaví příznak platnosti).
 - Restartuje provedení instrukce v prvním kroku. Procesor znovu přečte instrukci (tentokrát ji v cache nalezne).

Příklad: Přístup do paměti s cache

- Cache obsahuje 8 bloků. Při startu je prázdná. Budeme sledovat její činnost ...

1) Adresa 10110
vstup 110 - miss

Index	Valid	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Příklad: Přístup do paměti s cache

- Cache obsahuje 8 bloků. Při startu je prázdná. Budeme sledovat její činnost ...

- 1) Adresa 10110
vstup 110 - miss
- 2) Adresa 11010
vstup 010 - miss

Index	Valid	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10	Memory[10110]
111	N		

Příklad: Přístup do paměti s cache

- Cache obsahuje 8 bloků. Při startu je prázdná. Budeme sledovat její činnost ...

- 1) Adresa 10110
vstup 110 - miss
- 2) Adresa 11010
vstup 010 - miss
- 3) Adresa 10110
vstup 110 - hit

Index	Valid	Tag	Data
000	N		
001	N		
010	Y	11	Memory[11010]
011	N		
100	N		
101	N		
110	Y	10	Memory[10110]
111	N		

Příklad: Přístup do paměti s cache

- Cache obsahuje 8 bloků. Při startu je prázdná. Budeme sledovat její činnost ...

- 1) Adresa 10110
vstup 110 - miss
- 2) Adresa 11010
vstup 010 - miss
- 3) Adresa 10110
vstup 110 - hit
- 4) Adresa 10010
vstup 010 - miss

Index	Valid	Tag	Data
000	N		
001	N		
010	Y	11	Memory[11010]
011	N		
100	N		
101	N		
110	Y	10	Memory[10110]
111	N		

Příklad: Přístup do paměti s cache

- Cache obsahuje 8 bloků. Při startu je prázdná. Budeme sledovat její činnost ...

- 1) Adresa 10110
vstup 110 - miss
- 2) Adresa 11010
vstup 010 - miss
- 3) Adresa 10110
vstup 110 - hit
- 4) Adresa 10010
vstup 010 - miss

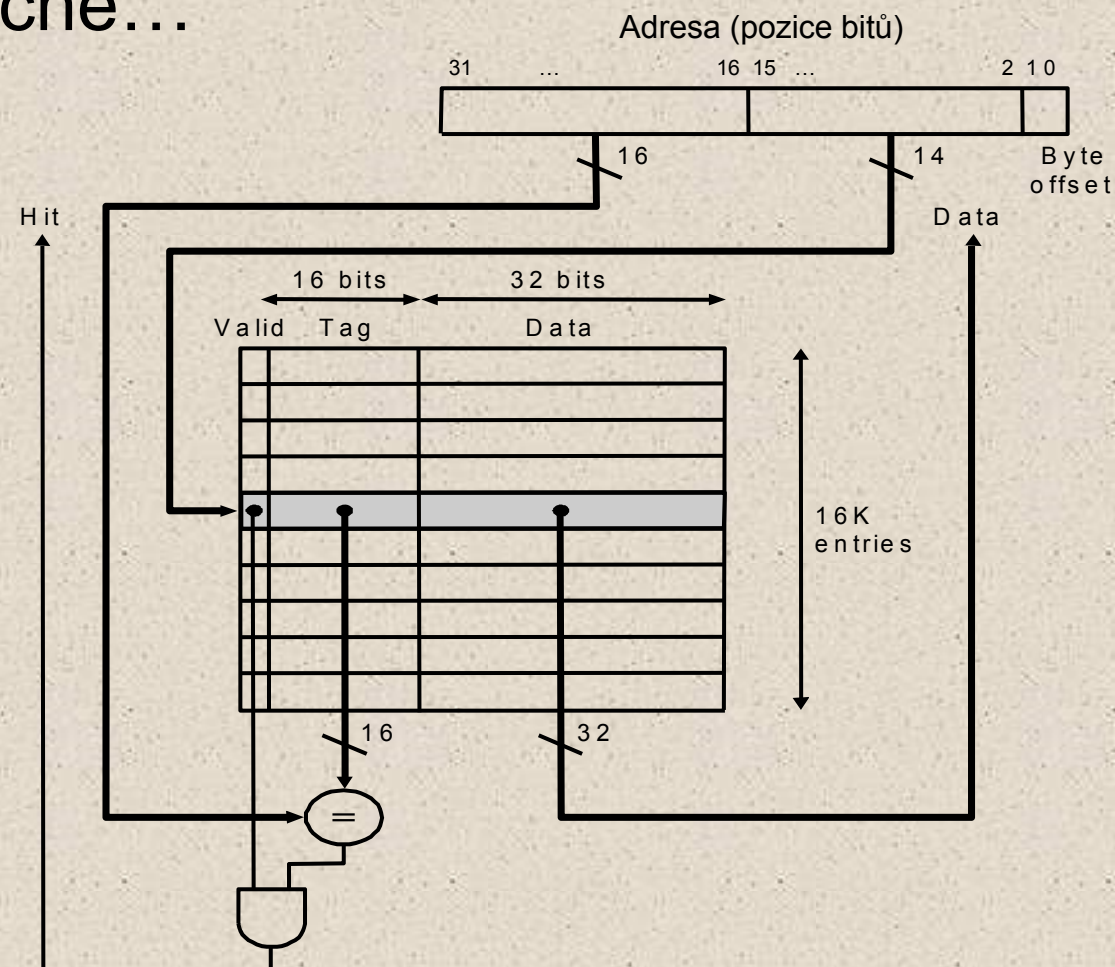
Index	Valid	Tag	Data
000	N		
001	N		
010	Y	10	Memory[10010]
011	N		
100	N		
101	N		
110	Y	10	Memory[10110]
111	N		

Reálné cache

- Jako reálný příklad uvedeme DEC3100, pracovní stanice s procesorem MIPS R2000.
- Protože čtení instrukce a dat probíhá ve stejném cyklu, používají se oddělené cache paměti pro instrukce a data - I a D cache (každá 64KB).
- Požadavek na čtení je jednoduchý...
 - Zaslání adresy příslušné cache paměti. To se děje buď z PC (I) nebo z ALU (D).
 - Hlásí-li cache miss, zasílá adresu do hlavní paměti. Jakmile paměť informaci přečte, je uložena do cache a zpracování pokračuje.

Cache DEC3100

- Jak pro data tak i pro instrukce je vyhrazena jedna taková cache...



Záписy do cache

- Z předchozího vyplývá, že čtení z paměti, je-li přítomna cache, je jednoduché. Co se ale bude dít, má-li proběhnout zápis?
- Předpokládejme, že zápis proběhne jen do datové cache (hlavní paměť se nezmění). Po tomto zápisu se data v paměti a v cache přestanou shodovat, jsou *nekonzistentní*.
- Nejjednodušší metodou, jak řešit tento problém, je zapsat data do obou ve stejnou dobu. Tato strategie se nazývá **metoda přímého zápisu** (*write-through*).

Metoda přímého zápisu

- Pokud je adresovaná položka v cache (write hit), má smysl aktualizovat kopii v cache a současně zapsat do „originálu“ v hlavní paměti..
- Jak by se měl systém chovat v případě, že se zapisovaná položka v cache nenachází?
 - Nejjednodušším řešením je zapsat data do řádky v cache a současně aktualizovat tag a příznak platnosti (valid bit).
 - Dokonce není třeba zkoumat, zda data jsou nebo nejsou v cache. Zkrátka data zapíšeme do příslušného řádku.

Ztráta výkonu?

- Uvedený přístup byl implementován u DEC3100.
- Metoda přímého zápisu je velmi jednoduchá – data se jednoduše zapisují do cache. Snadno se implementuje.
- Vychází z pozorování, že u běžných programů představuje operace zápisu jen 10 – 30% ze všech operací s pamětí (čtení, zápis).
- Z hlediska výkonu nedává ale právě nejlepší výsledky.
 - Kdykoliv se procesor provádí zápis, aktualizuje se cache a **současně** také hlavní paměť. To trvá dlouho vzhledem k relativně pomalé hlavní paměti (dlouhá doba zápisového cyklu).

Zápisový buffer

- Redukci výkonu, způsobenou zápisy, lze omezit použitím speciální vyrovnávací paměti – **zápisového bufferu** (*write buffer*).
 - Zápisový buffer obsahuje data, která čekají na zápis do hlavní paměti.
 - Zápis procesoru probíhá tak, že se zapíše do cache a do zápisového bufferu a pak procesor může pokračovat ve výpočtu.
 - Po ukončení zápisu do paměti se zápisový buffer opět uvolní.
 - Je-li zápisový buffer ještě plný a procesor opět požaduje zápis, musí být pozastaven, dokud se buffer neuvolní.

Selhání zápisového bufferu?

- Uvedená strategie zlepšuje výkon, ale může selhat.
- Jestliže procesor generuje zápisy rychleji než se mohou provádět zápisy do paměti, zápisový buffer příliš nepomůže.
- I když je četnost zápisů nižší, přesto může docházet k zastavování, pokud se budou zápisy shlukovat. Lze částečně kompenzovat větší hloubkou zápisového bufferu než 1.
- DEC3100 má hloubku zápisového bufferu 4.

Strategie nepřímého zápisu - Write-Back

- Alternativní přístup je interpretovat všechny zápisové operace jen jako zápisy do cache. Hlavní paměť se aktualizuje až při výměně bloků.
- Tato metoda se nazývá **metoda nepřímého zápisu** (*write-back*).
- Metoda vede na podstatně složitější implementaci, může ale přinést značné urychlení zápisových operací.

Strategie zápisu bloků

1) Write-Through:

- Zápis dat (a) do cache *a současně* (b) do bloku v hlavní paměti
- **Výhoda:** Případ „Miss“ je jednodušší & levnější, protože není třeba zapisovat blok zpět do nižší úrovně
- **Výhoda:** Snazší implementace, je třeba pouze zápisový buffer

2) Write-Back:

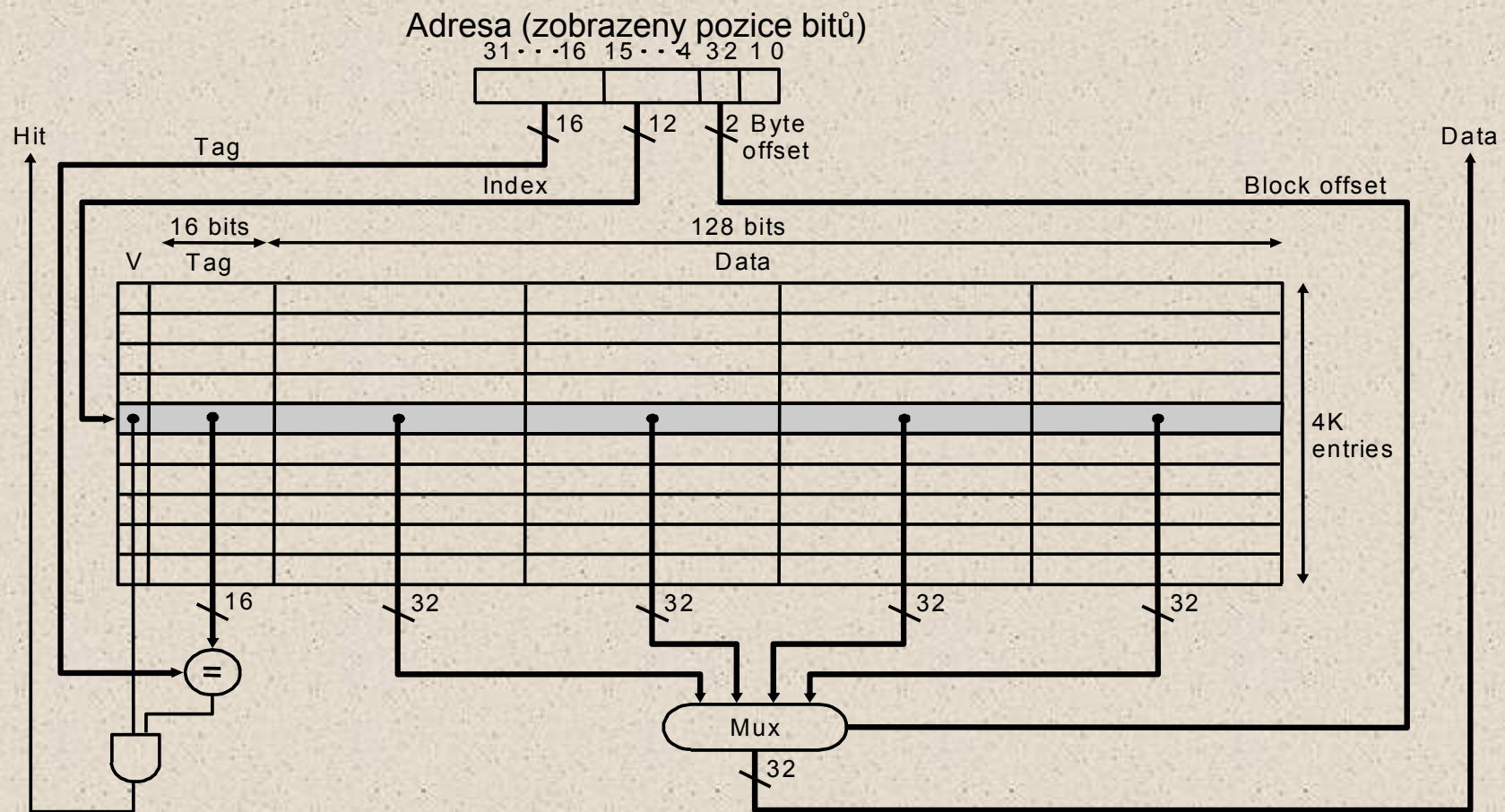
- Zápis dat *pouze* do bloku cache. Zápis do paměti pouze při výměně bloků
- **Výhoda :** Zápisy omezeny pouze rychlostí zápisu cache
- **Výhoda :** Podporovány zápisy více slov najednou, efektivní je pouze zápis celého bloku do hlavní paměti najednou

Prostorová lokalita

- Dosud jsme ignorovali prostorovou lokalitu – pozorování, že data, sousedící s právě referencovanými daty mají větší šanci přístup v krátké době.
- Tento princip může být zohledněn tak, že vytvoříme bloky, které mají větší velikost než pouhé jedno slovo.
- Nastane-li výpadek, přečteme kromě požadovaného ještě další slova, ležící „vedle“ (ve stejném bloku). Je vysoce pravděpodobné, že budou v zápětí požadována.

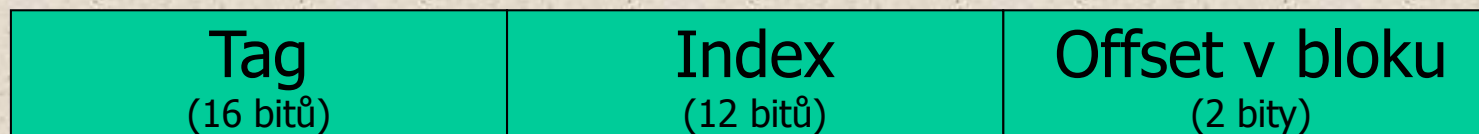
Cache s délkou bloku větší než jedno slovo

- Struktura cache, která využívá prostorové lokality...



Sémantika cache - bloky s větším počtem slov

- Předpokládejme, že blok obsahuje 4 slova.
- Dva nejnižší bity použijeme k výběru slova v bloku.
- Dalších 12 bitů vybírá blok (přímo mapovaná cache).
- Horních 16 bitů se používá jako tag.
- Poznámka: Používáme jeden tag pro čtyři slova v paměti...
 - Zlepšuje efektivitu a ukládá se menší množství pomocné informace (tag), vztažené na jedno slovo.



Vliv prostorové lokality

```
void copyij(int src[2048][2048],
            int dst[2048][2048])
{
    int i,j;
    for (i = 0; i < 2048; i++)
        for (j = 0; j < 2048; j++)
            dst[i][j] = src[i][j];
}
```

5.2 ms

```
void copyji(int src[2048][2048],
            int dst[2048][2048])
{
    int i,j;
    for (j = 0; j < 2048; j++)
        for (i = 0; i < 2048; i++)
            dst[i][j] = src[i][j];
}
```

162 ms

2.8 GHz Intel Core i7

- Hierarchická organizace paměti
- Výkon závisí na typu přístupu (access patterns)
 - Včetně způsobu pohybu ve vícerozměrném poli

Výpadek v cache s víceslovnými bloky

- Výpadky při čtení jsou stejně jednoduché jako u jednoslovných bloků – jednoduše načteme data z paměti.
- Výpadky při zápisu jsou složitější.
 - Protože každý blok obsahuje více než jedno slovo, nemůžeme jednoduše zapsat tag a datové slovo – ostatní slova nepřísluší stejnému bloku (čtveřici slov).
 - Pro **write-through** cache, nejsou-li tagy shodné, můžeme načíst celý blok z paměti. Po načtení bloku můžeme zapsat slovo, které způsobilo výpadek do bloku a do paměti.
 - Použitím této strategie způsobuje výpadek při zápisu čtení z paměti (na rozdíl od cache s jedním slovem v bloku).

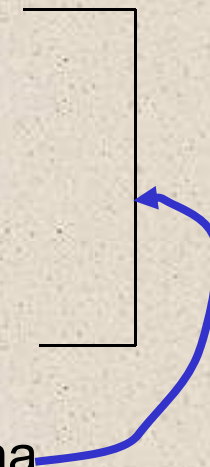
Cena za více slov v bloku

- Umístíme-li v bloku větší počet slov, četnost výpadků klesá.
- Naopak narůstá cena za načtení nového bloku, protože načítáme více slov.
- Zvětšujeme-li dále velikost bloku, převáží ztráta načítáním velkých bloků a účinnost cache se (podstatně) sníží.

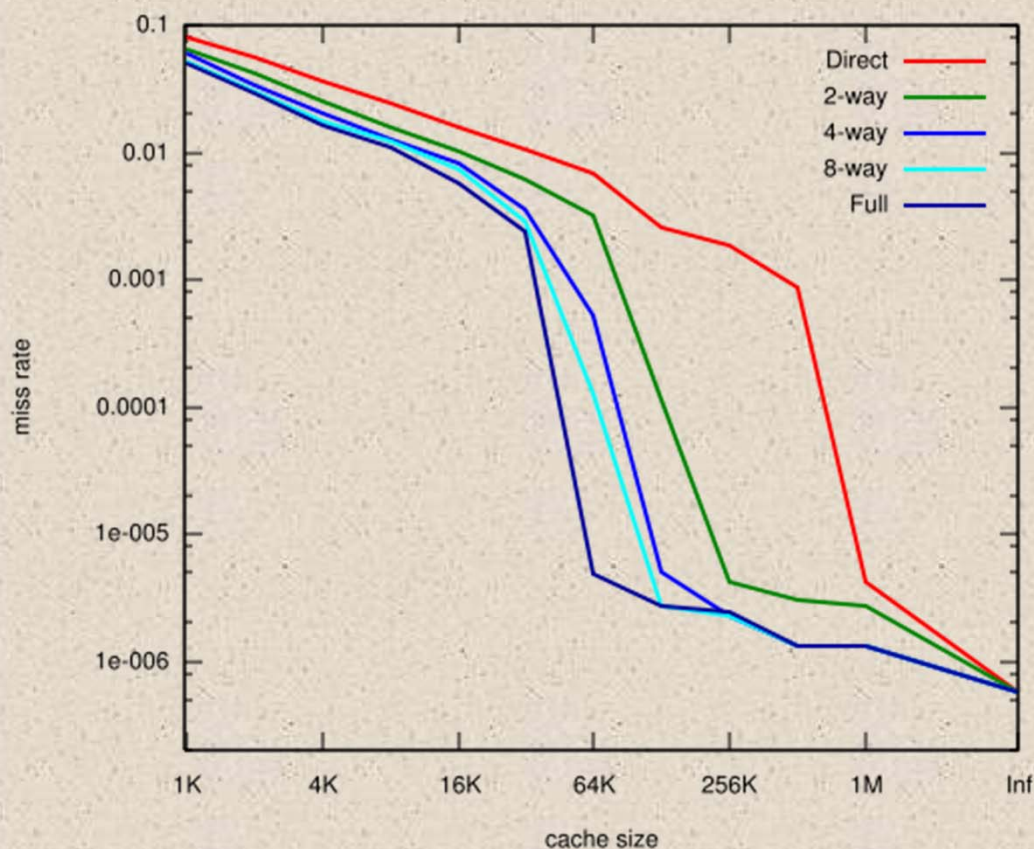
Případ – data v cache nenalezena

Příklad: „Miss“ v instrukční cache (instrukce nebyla v cache nalezena)

1. Do paměti poslána originální hodnota PC (současný PC – 4)
2. Hlavní paměť provede **Read**, čekáme na dokončení přístupu
3. Zápis položky do cache (**Write**) :
 - Data z paměti se zapíše do datového pole cache
 - Zápis horních bitů adresy do příslušného pole Tag
 - Nastavení příznakového bitu **Valid Bit ON**
4. Restart provedení instrukce ve stupni pipeline IF – instrukce je znovu načtena, nyní již bude v cache nalezena



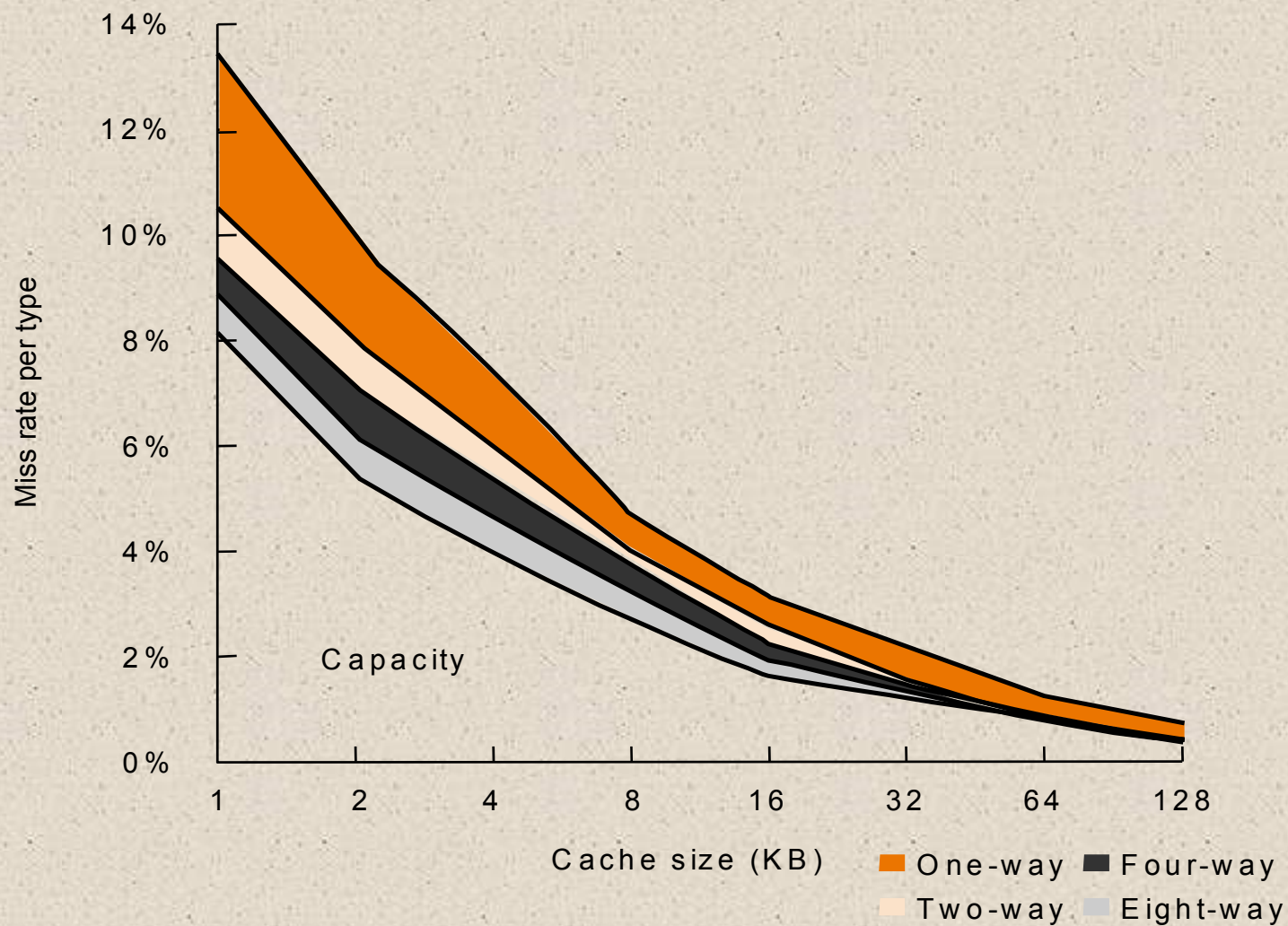
Četnost „výpadků“ závisí na organizaci



Typy výpadků:

- **compulsory miss**
(např. počáteční plnění)
- **capacity miss**
(z důvodu malé kapacity cache)
- **conflict miss**
(příčinou je konflikt)

Četnost výpadků a jejich příčiny



Typy výpadků

- Co způsobuje výpadek bloku? Existují tři příčiny...
 - **Povinné výpadky (Compulsory Misses)** – při prvním přístupu k bloku v cache se blok v cache nenachází.
 - **Kapacitní výpadky (Capacity Misses)** – jestliže cache neobsáhne všechny bloky potřebné k běhu programu. Nastávají, když jsou bloky odklizeny do nižší úrovně a při potřebě opět načítány.
 - **Výpadky kvůli konfliktu (Conflict Misses)** – nastávají u přímo mapované a nebo částečně asociativní cache, kdy různé bloky obsazují stejnou pozici v cache.

Povinné výpadky (Compulsory Misses)

- Generují se v okamžiku prvního přístupu na blok. Nejlépe se redukují zvětšením velikosti bloku.
 - Redukuje se počet referencí k „novým“ blokům pro každý program.
 - Celý program pak obsazuje menší počet bloků.
- Zvětšování má ale za důsledek vyšší cenu za výměnu bloku (cena za miss). Proto je třeba volit kompromis.

Kapacitní výpadky (Capacity Misses)

- Výpadky z důvodů kapacitních lze potlačit zvětšováním kapacity cache.
- Zvětšování kapacity s sebou ale zároveň přináší nárůst přístupové doby, což by mohlo vést k poklesu výkonu.

Výpadky kvůli konfliktům (Conflict Misses)

- Tento typ výpadků lze redukovat zvýšením asociativity cache.
- Výpadky tohoto typu vznikají tehdy, obsazuje-li nový blok místo jiného, i když je v cache ještě volné místo. Zvětšováním počtu možných pozic pro daný blok se tyto výpadky redukuje.
- Opět je třeba volit kompromis, protože zvyšování míry asociativity může způsobovat určitou časovou ztrátu.

Měření výkonu cache

- Abychom porozuměli těmto závislostem, je třeba pochopit princip měření.
- Protože čas CPU je rozdělen na hodinové takty věnované výpočtu a hodinové takty strávené čekáním na paměť, vystačíme s jednoduchým vzorcem (který předpokládá, že úspěšné přístupu jsou součástí běžných prováděcích cyklů) ...

$$\text{CPU time} = (\text{CPU execution cycles} + \text{Memory-stall cycles}) * \text{Clock cycle time}$$

Stall cycle ...“prostoj“

Měření výkonu cache

- Učinili jsme mnoho předpokladů (hlavně ten, že všechna pozastavení pipeline jsou způsobena výpadkem cache). V moderních procesorech vyžaduje přesná predikce výkonu komplexní simulaci procesoru a celého paměťového systému.
- Víme, že přístupy do paměti jsou jednotlivá čtení a zápisy...

$$\text{Memory-stall cycles} = \text{Read-stall cycles} + \text{Write-stall cycles}$$

Zpoždění při čtení a zápisu

- Zpoždění při čtení lze snadno určit..

$$\text{read-stall cycles} = \text{reads/program} *$$

$$\text{read miss rate} * \text{read miss penalty}$$

- Zpoždění vlivem zápisů se určuje obtížněji.
Předpokládáme-li strategii write-through, existují dvě příčiny. Výpadek při zápisu (když blok musí být před zápisem načten) a zpoždění dané činností zápisových bufferů (je-li zápisový buffer zaplněn)...

$$\text{write-stall cycles} = (\text{writes/program} *$$

$$\text{write miss rate} * \text{write miss penalty})$$

$$+ \text{write buffer stalls}$$

Zpoždění při zápisu

- Zpoždění při zápisech vlivem zápisových bufferů závisí na jejich časování i na jejich frekvenci. Původní jednoduchá rovnice pak ztrácí na přesnosti. Naštěstí dostatečně dlouhé zápisové buffery a dostatečně rychlá paměť podstatně snižují počet zápisových zpoždění a lze je proto ignorovat (kdyby tomu tak nebylo, znamenalo by to špatný návrh paměťového systému).
- Navíc strategie write-back může přinášet další zpoždění způsobená nutností zápisu bloku zpět do hlavní paměti při výměně.

Zobecněná rovnice výkonu

- Rovnici můžeme zjednodušit, jsou-li „pokuty“ za výpadek při čtení i zápisu stejné (což obvykle platí; doba potřebná k načtení bloku z nižší úrovně)...

$$\text{memory-stall cycles} = \text{memory accesses/program} * \text{miss rate} * \text{miss penalty}$$

- Lze také psát ...

$$\text{memory-stall cycles} = \text{instructions/program} * \text{misses/instruction} * \text{miss penalty}$$

Příklad: Výpočet výkonu cache

- Předpokládejme, že procesor má instrukční cache s četností výpadků 2% a datovou cache s četností výpadků 4%.
- Procesor má $CPI = 2.0$ bez zpoždění vlivem výpadků paměti a „pokuta“ za výpadek je 40 cyklů za každý.
- Instrukce Load a Store tvoří 36% z celého instrukčního toku. Žádné jiné instrukce přístup do paměti nevyžadují.
- Kolikrát pomalejší je tento systém než ten, u kterého by nevznikaly žádné výpadky cache?

Příklad (řešení): Výpočet výkonu cache

- Počet ztracených cyklů při výpadcích při čtení instrukcí je roven:

$$I * 2\% * 40 \text{ cyklů} = 0.80I$$

- Počet cyklů při výpadcích kvůli datům je roven

$$I * 36\% * 4\% * 40 \text{ cyklů} = 0.56I$$

- Proto je celkový počet cyklů při výpadcích roven 1.36I, což je více než jeden cykl na provedenou instrukci.

– Parametr CPI díky „pokutám“ vzroste na 3.36.

- Poměr výpočetních dob CPU je roven...

$$\frac{CPUtime \text{ _with _stalls}}{CPUtime \text{ _without _stalls}} = \frac{I \times CPI_{stall} \times ClockCycle}{I \times CPI_{perfect} \times ClockCycle} = \frac{CPI_{stall}}{CPI_{perfect}} = \frac{3.36}{2} = 1.68$$

Řešení

Proto cache bez výpadků je 1.68 krát rychlejší než reálná cache.

Vztah výkonu procesoru a paměti

- Zachováme-li vlastnosti paměťového systému a procesor zrychlíme, relativní ztráta se zvýší.
 - Klesne-li CPI, zvýší se vliv „pokuty“ za výpadky.
 - Doba cyklu procesoru se zlepšuje rychleji než paměť (!historie). „Pokuta za výpadek se měří v počtu cyklů CPU na jeden výpadek (miss).
 - Jestliže paměti dvou systémů mají stejné přístupové doby, stroj, jehož CPU má rychlejší hodiny, vykazuje také větší „pokutu“ za výpadek.

Příklad: Vztah výkonu procesoru a paměti

- Předpokládejme počítač z předchozího příkladu. Procesor bude pracovat na dvojnásobné frekvenci, „pokuta“ za výpadek cache zůstane stejná.
- Jak rychlý bude celý počítač, předpokládáme-li stejnou četnost výpadků?
- Poznámka:
Jestliže neuvažujeme vliv cache a paměti, nový stroj bude mít dvojnásobnou rychlost než předchozí.

Příklad: Vztah výkonu procesoru a paměti

- Nová „pokuta“ za výpadek je 80 hodinových taktů.
- Počet cyklů výpadku na instrukci ...
 $(2\% * 80) + 36\% * (4\% * 80) = 2.75$
- Proto nový počítač má CPI = 4.75 v porovnání s CPI = 3.36 pomalejšího stroje.
- Použitím podobného vzorce jako v předchozím případě lze vypočítat relativní výkon...

$$\frac{\text{Výkon(rychlé_hodiny)}}{\text{Výkon(pomalé_hodiny)}} = \frac{I \times \text{CPI}_{\text{fast}} \times \text{ClockCycle}}{I \times \text{CPI}_{\text{slow}} \times \frac{\text{ClockCycle}}{2}} = \frac{3.36}{4.75 \times \frac{1}{2}} = 1.41$$

- Počítač s dvojnásobnou hodinovou frekvencí je 1.41 krát rychlejší. Kdyby neexistovaly výpadky cache, byla by jeho rychlost dvojnásobná!

Cache s flexibilnějším umístováním bloků

- Dosud jsme uvažovali *přímo mapované cache* - každý blok lze umístit do jediného místa v cache.
- To je jedna krajní varianta ze všech rozličných strategií pro umístování bloků.
- Opačným extrémem je *plně-asociativní* cache – u tohoto typu mechanismu cache, blok z hlavní paměti může být umístěn do libovolného místa v cache.
 - Jestliže umožníme umístění bloku do libovolného místa v cache, budeme jej také muset umět v libovolném místě nalézt. Prohledání celé cache.

Cena plně asociativní cache

- Aby takový mechanismus byl prakticky použitelný, musí hledání bloku ve všech místech probíhat paralelně.
- Každý vstupní bod cache (místo pro blok) obsahuje odpovídající komparátor – tyto komparátory podstatně zvyšují cenu hardware této organizace cache. Uvedená strategie je vhodná pro cache s malým počtem vstupních bodů.

Částečně asociativní cache

- Praktičtější řešení představují *částečně asociativní cache*, které představují kompromisní řešení mezi přímo mapovanou a plně asociativní organizací.
- U tohoto typu cache existuje pevný počet míst (2 nebo více) kde může být daný blok umístěn. Existuje-li pro každý blok n možných pozic, nazývá se taková organizace *n -cestná částečně asociativní cache*.
 - n -cestná částečně asociativní cache se skládá z velkého počtu skupin bloků, z nichž každá obsahuje n bloků.
 - Každý blok v paměti se mapuje na určitou skupinu bloků v cache; může být uložen do libovolné pozice v této korespondující skupině bloků.

Zpět k vícecestné částečně asociativní cache

- Na každou strategii pro umísťování bloků lze nahlížet jako na variaci vícecestné paměti...
 - Přímo mapovaná cache je vlastně jednocestná částečně asociativní cache; každý vstupní bod uchovává jeden blok (skupina bloků o velikosti 1).
 - Plně asociativní cache s m vstupními body je vlastně m -cestná „částečně“ asociativní cache; má jen jednu skupinu bloků a v ní se může kdekoliv nacházet libovolný blok z hlavní paměti.

Různé organizace cache

- Cache s 8-bloky můžeme organizovat...

**One-way set associative
(direct mapped)**

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set associative (fully associative)

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

Vlastnosti různých organizací cache

- Nahlédneme na některé reprezentativní statistiky dvou programů, abychom posoudili, jak zvýšená míra asociativity zlepšuje situaci (organizace podobná DECStation 3100 cache s velikostí bloku 4 slova...

Program	Associativity	Instruction Miss Rate	Data Miss Rate	Eff. Combined Miss Rate
gcc	1	2.0%	1.7%	1.9%
gcc	2	1.6%	1.4%	1.5%
gcc	4	1.6%	1.4%	1.5%
spice	1	0.3%	0.6%	0.4%
spice	2	0.3%	0.6%	0.4%
spice	4	0.3%	0.6%	0.4%

- Zvýší-li se asociativita z 1 na 2, gcc výpadky se redukuje o ~20%. Spice již tak nízkou četnost má, tedy uvedená změna příliš nepomůže. Změna asociativity ze 2 na 4 již nic nevylepší.

Nalezení bloku u částečně asociativní cache

- Podobně jako u přímo mapované cache, každý blok ve vícecestné cache obsahuje tag, který určuje adresu bloku.
- Každá adresa do paměti je rozdělena na tři části...

Tag	Index	Block Offset
-----	-------	--------------

- Hodnota indexu slouží k výběru skupiny, do které blok patří. Tag se komparuje se všemi tagy bloků ve skupině, aby se zjistilo, zda daný vstupní blok obsahuje hledaný blok.
- Rychlost je podstatná – všechny tagy ve vybrané skupině se komparují **paralelně !!!**

Částečně asociativní mapování

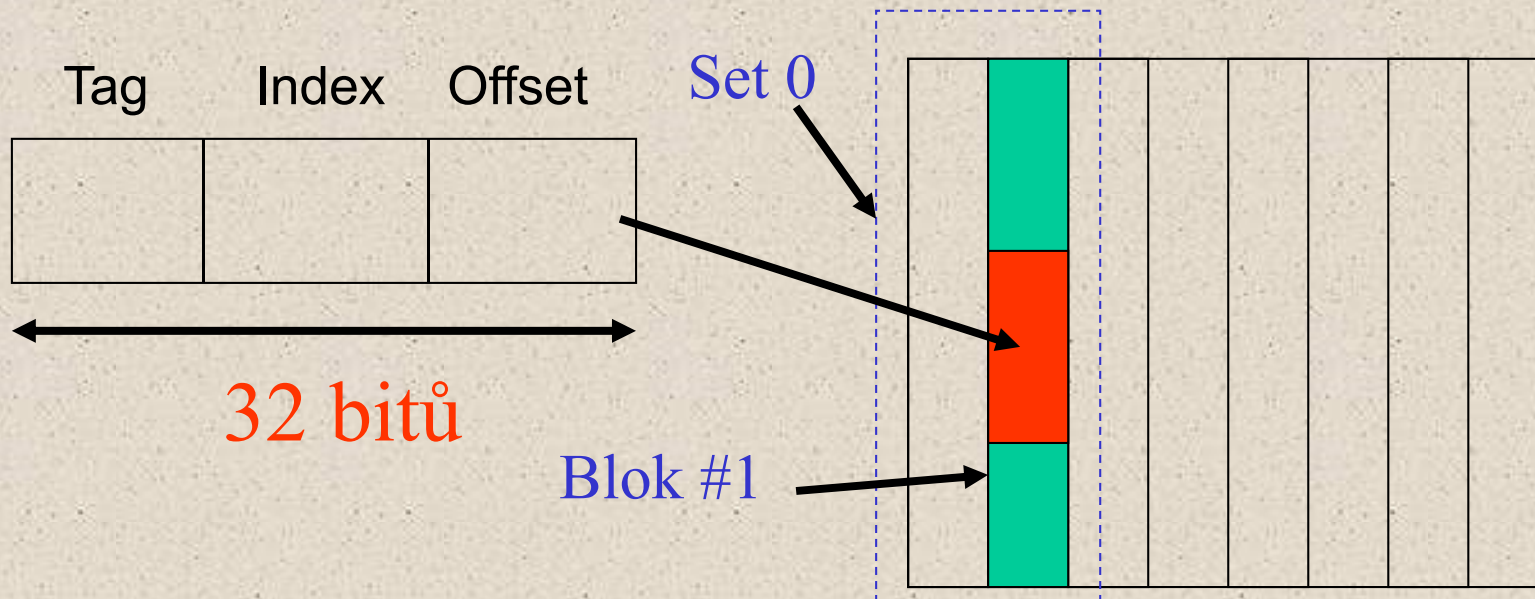
- **Zobecňuje všechna mapovací schémata cache**
 - Předpokládejme, že cache obsahuje N bloků
 - 1- cestná částečně asociativní cache: Přímé mapování
 - M -cestná částečně asociativní cache: Je-li $M = N$, pak se jedná o plně asociativní cache
- **Výhoda**
 - Zvyšuje úspěšnost – klesá „miss rate“ (více míst, kde lze nalézt B)
- **Nevýhoda**
 - Narůstá „hit time“ (více míst, kde je třeba hledat B)
 - Složitější hardware

Činnost částečně asociativní cache

Složky adresy cache:

- *Index i*: Vybírá množinu S_i
- *Tag*: Použit pro výběr hledaného bloku, porovnáním n bloků ve vybrané množině S
- *Blok Offset*: Adresa cílové položky dat uvnitř bloku

Dvoucestná
cache



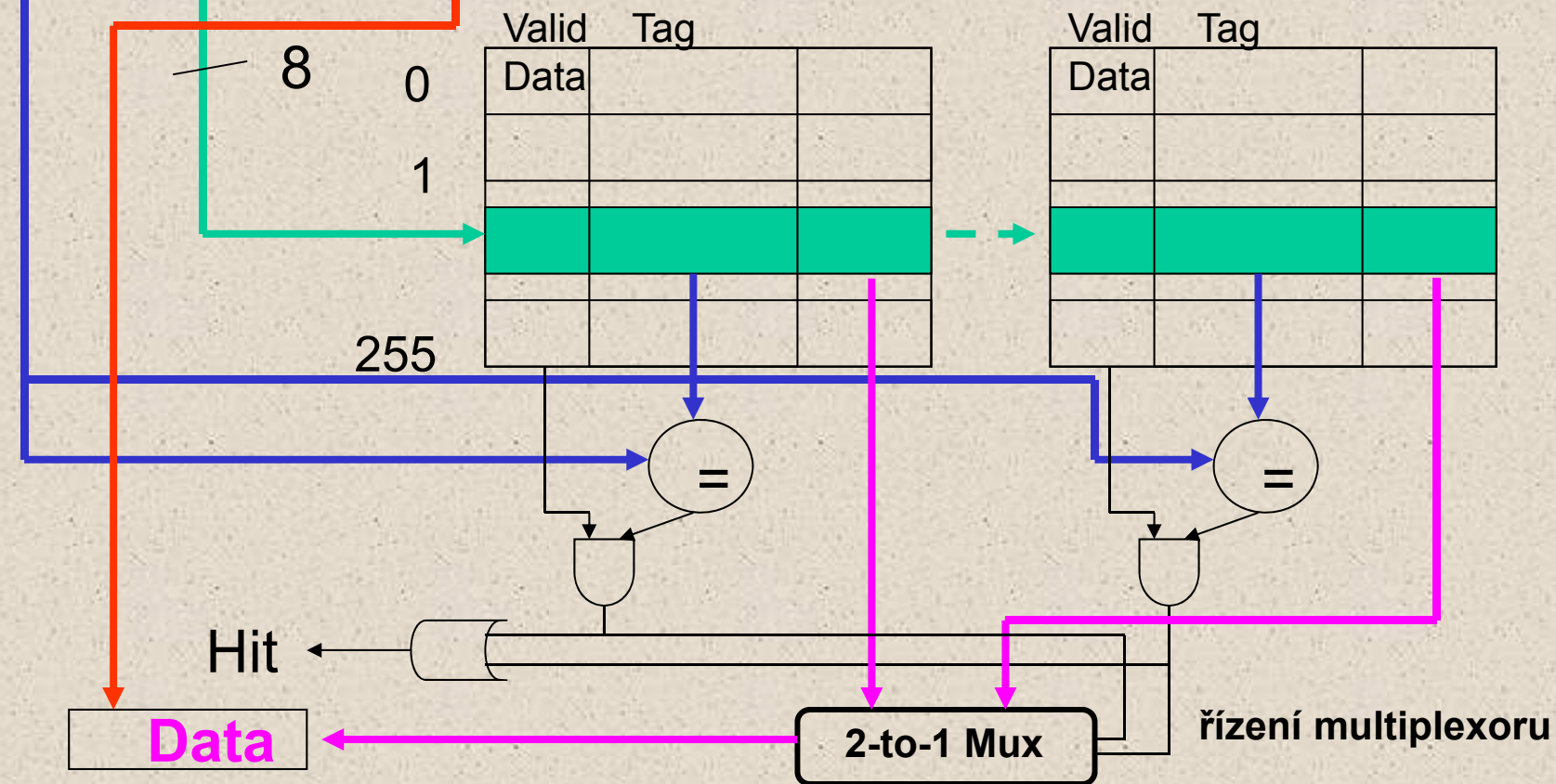
Příklad: Hardware 2-cestné cache

Adresa pro cache



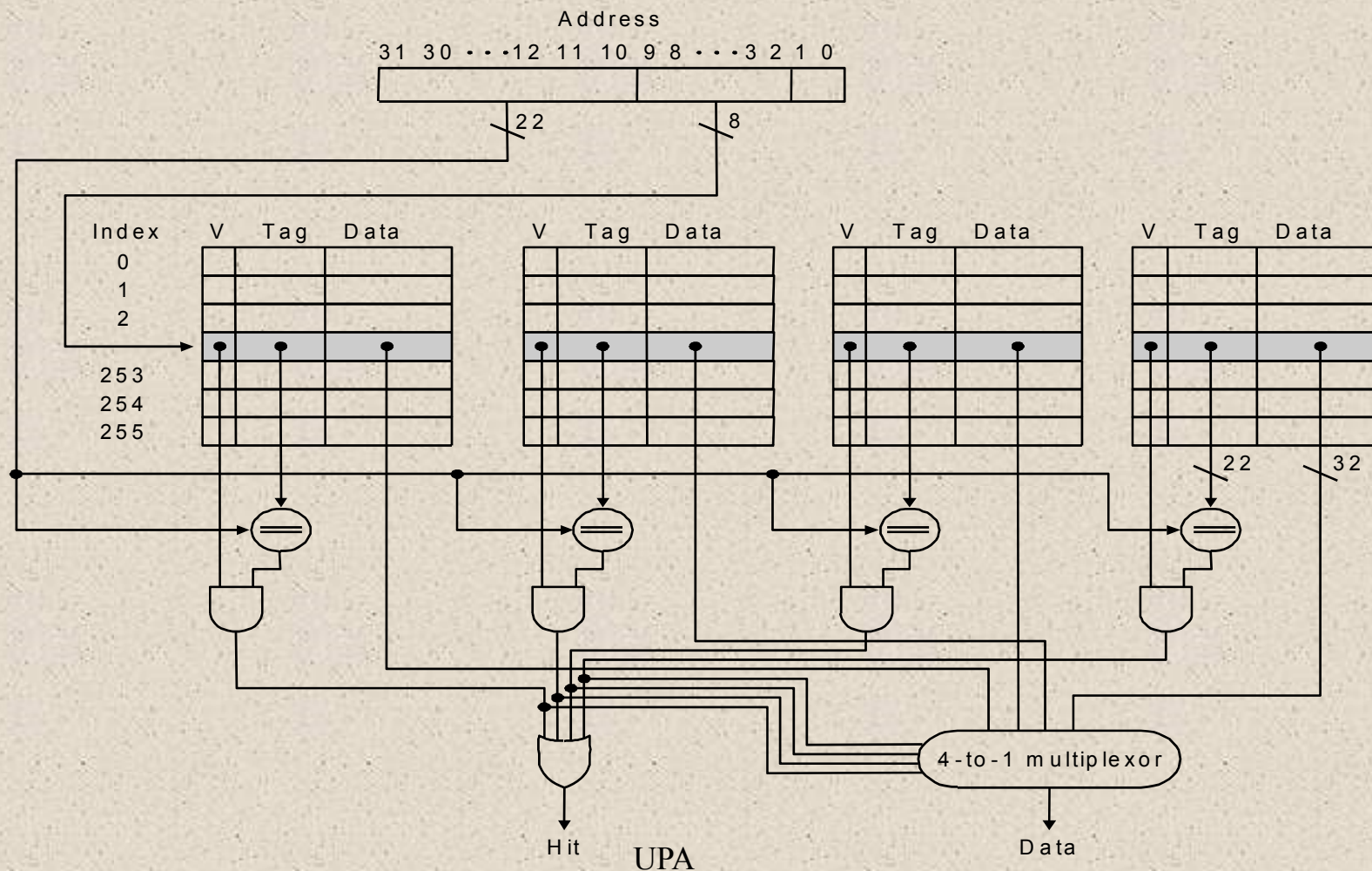
N-cestná cache:

N multiplexorů a hradel, komparátory



Nalezení bloku u 4-cestné cache

- Příklad (4-cestná částečně asociativní cache)...



Návrh strategie cache

- Rostoucí asociativita cache vyžaduje více hardware. Jak se má návrhář vypořádat s volbou strategie umísťování bloků v cache?
- Je třeba zvážit cenu výpadku oproti ceně za vyšší míru asociativity u jednotlivých organizací (přímo-mapovaná, n-cestná a plně-asociativní).
 - Dvě různé metriky: čas a cena
 - Ke které se obrátíte? Rozpočet cache?

Strategie výměny bloků

- Nastane-li výpadek u přímo mapované cache, požadovaný blok může být zapsán do jediného místa a blok, který toto místo zaujímá musí být vyměněn.
- U asociativní cache je na výběr, kam bude požadovaný blok zapsán.
 - U plně asociativní cache jsou všechny bloky přítomné v cache kandidáty na výměnu.
 - U n-cestné částečně asociativní cache, všechny bloky korespondující skupiny jsou kandidáty na výměnu.

Algoritmus LRU

- Existuje celá řada strategií jak vybrat blok vhodný pro výměnu (z kandidátů), které lze použít v případě výpadku.
- Velmi často používaným algoritmem je *least recently used (LRU)*.
- Nahrazován je blok, který nejdéle nebyl použit (využívá myšlenku časové lokality).
- LRU strategie se implementuje tak, že sledujeme relativní použití oproti ostatním členům skupiny.

Implementace LRU

- U dvoucestné částečně asociativní cache je sledování jednoduché.
 - Ke každému vstupnímu bodu se přidá jeden bit. Kdykoliv je blok referencován, je příslušný bit daného bloku nastaven a odpovídající bit druhého bloku nulován.
 - Nastane-li výpadek a musí dojít k výměně bloků, dojde k ní u bloku, jehož bit je vynulovaný. Bit nového bloku je pak nastaven.
- Situace je poněkud složitější, obsahuje-li skupina více bloků ($n = 4, 8$).
- Zmíníme se o jiných strategiích výměny. (Jedná se o poměrně složitou problematiku, kterou se nebudeme zabývat do hloubky).

Víceúrovňové cache

- Všechny moderní počítače používají cache paměti.
- Většina novějších počítačů používá víceúrovňové cache paměti. Dnešní systémy mívají obvykle dvě úrovně cache (L1 a L2).
- Výkonné systémy dokonce 3 úrovně (přidána L3) – Pentium 4 Extreme Edition má 2MB L3 cache.
- Podívejme se na funkci takového víceúrovňového systému.

Hierarchie cache

- V takovém systému se do L2 cache vykoná přístup, jestliže nastane výpadek v L1 cache.
- Do L3 cache vykoná přístup, jestliže nastane výpadek v L2 cache.
- Teprve v případě, že dojde k výpadku na všech úrovních cache, vykoná se přístup k nejpomalejší technologii - k hlavní paměti.
- V této hierarchii L1 cache je nejmenší a nejrychlejší ze všech pamětí. L2 cache je větší, ale pomalejší. L3 cache je ještě větší a ještě pomalejší. Hlavní paměť je největší a nejpomalejší ze všech.

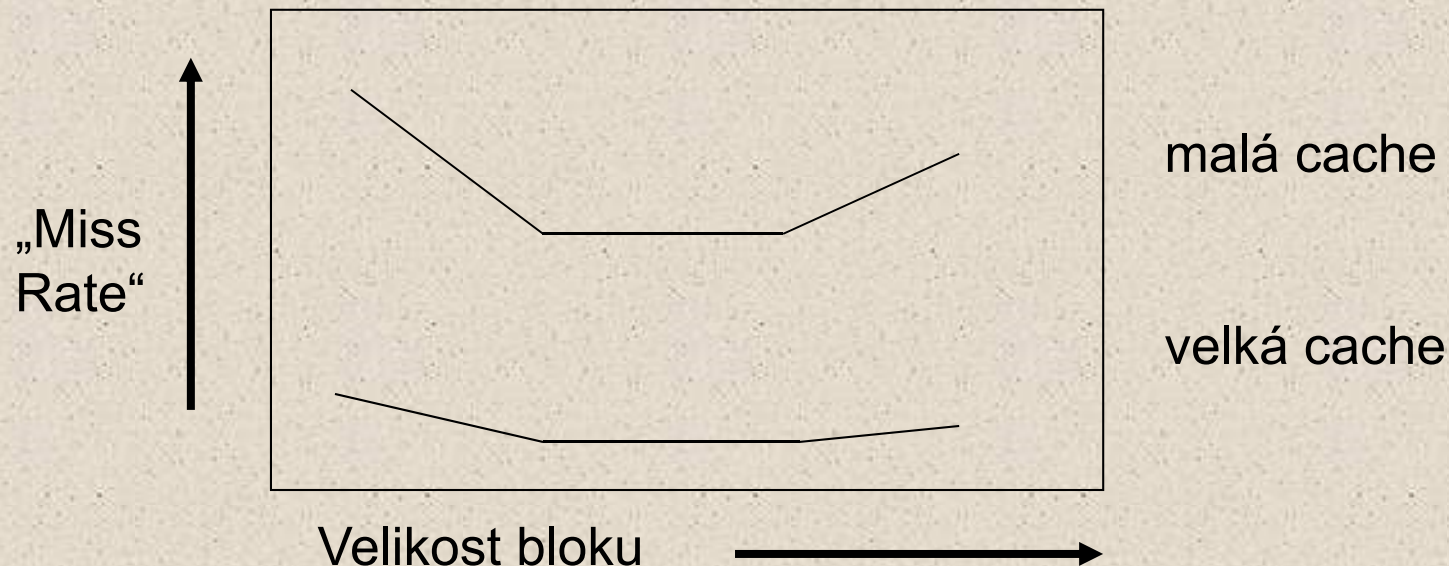
Uvažujeme-li „Hit Time“

- Dosud jsme opomíjeli „hit time“, množství času, které cache potřebuje ke zjištění, že se jedná o hit.
- Tato doba není nulová, protože cache musí porovnat svůj obsah s požadavkem, zjistit, zda je požadované slovo přítomno v cache. Z tohoto důvodu **hit time** narůstá, zvětšuje-li se kapacita cache.
- V určitém bodě tento nárůst převládne nad vlivem zlepšující se četnosti výpadků cache. (větší neznamená vždy lepší!).

Cena za „Cache Miss“

Příklad: „Miss“ v instrukční cache (instrukce v cache nenalezena)

- **Pozorování:** Větší bloky se načítají pomaleji
- **Operace:** Větší bloky využívají lépe *prostorovou lokalitu*
- **Co vybrat?** **Řešení:** Cache co možná největší,
to omezí vliv velikosti bloku



Návrh víceúrovňové cache

- U víceúrovňových cache může být L1 cache malá, aby se minimalizoval parametr „hit time“, zatímco L2 cache může být velká tak, aby byla příznivá četnost výpadků (!nízká). Tím se maskuje poměrně dlouhá přístupová doba do hlavní paměti.
 - Cena za miss v L1 cache se drasticky redukuje přítomností L2 cache, což dovoluje implementovat L1 poměrně malou (! ale rychlou !) s větší četností výpadků.
 - Doba přístupu L2 cache není tolik rozhodující, protože ovlivňuje „pokutu“ za miss v L1 víc, než přímo L1 „hit time“ nebo hodinovou frekvenci CPU.

Příklad: Pentium 4 Extreme Edition

- Pentium 4 Extreme Edition má tříúrovňovou cache...
 - 8KB L1 cache – Jedná se o 4-cestnou částečně asociativní cache s přenosovou rychlostí 23295 MB/sec. při čtení
 - 512KB L2 cache – Jde o 8-cestnou částečně asociativní cache s přenosovou rychlostí 12920 MB/sec. při čtení.
 - 2MB L3 cache – Jde o 8-cestnou částečně asociativní cache s přenosovou rychlostí 6522 MB/sec. při čtení.

Příklad: Pentium 4 Extreme Edition

Review of Cache Architecture

Memory Hierarchy

~Latency ~Size

1's nS 10's K

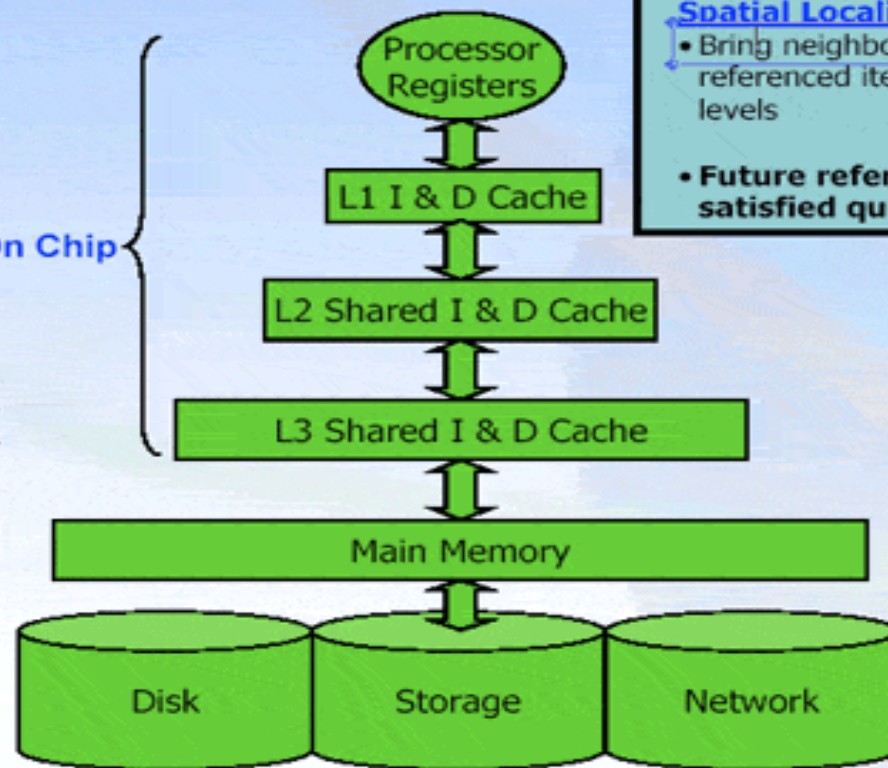
10's nS 100's K

10's nS 1000's K

100's nS 1000's M

1's mS 100's G

On Chip



Temporal Locality

- Keep recently referenced items at higher levels

Spatial Locality

- Bring neighbors of recently referenced items to higher levels

- Future references satisfied quickly

