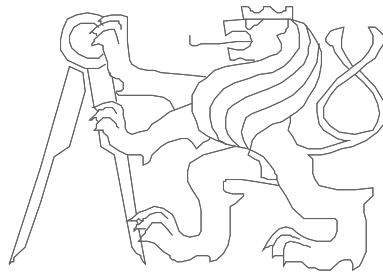


Architektury počítačů

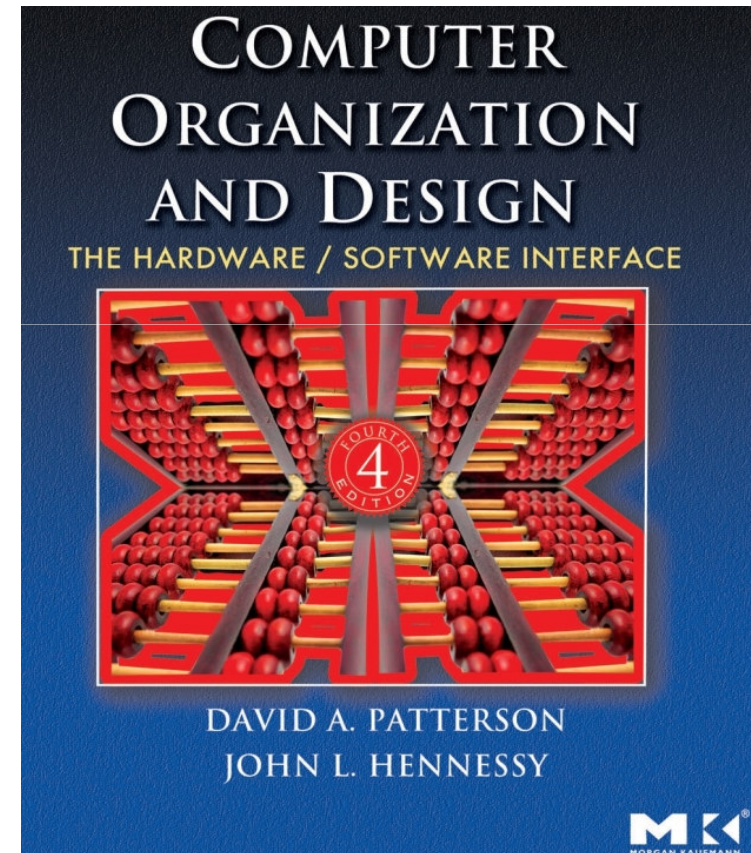
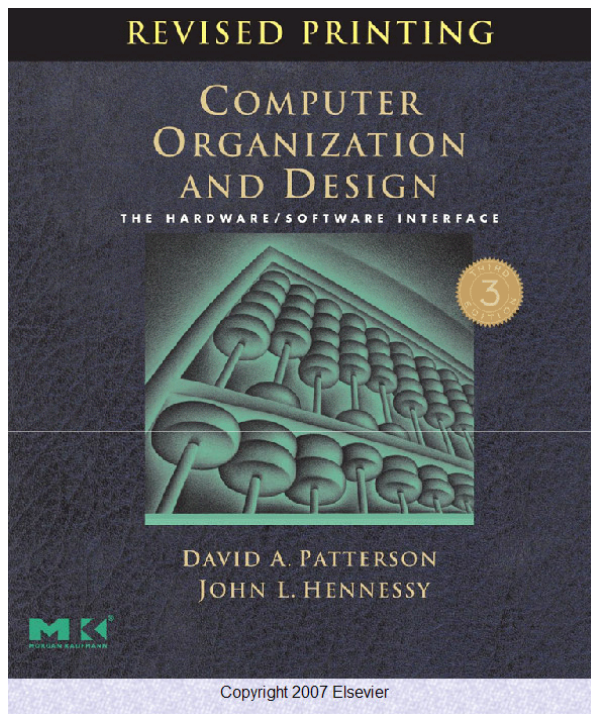
Paměť – část první
- úvod, realizace pamětí, cache

Hlavní inspirace a zdroj: Patterson



České vysoké učení technické, Fakulta elektrotechnická

Literatura k předmětu



Motivace pro přednášku z pohledu programátora?

Otázka 1.: Vykonávají programy to samé?

Otázka 2.: Který program je rychlejší (pokud některý)?

A:

```
int matrix[M][N];
int i,j,sum=0;
...
for(i=0;i<M;i++)
  for(j=0;j<N;j++)
    sum+=matrix[i][j];
```

B:

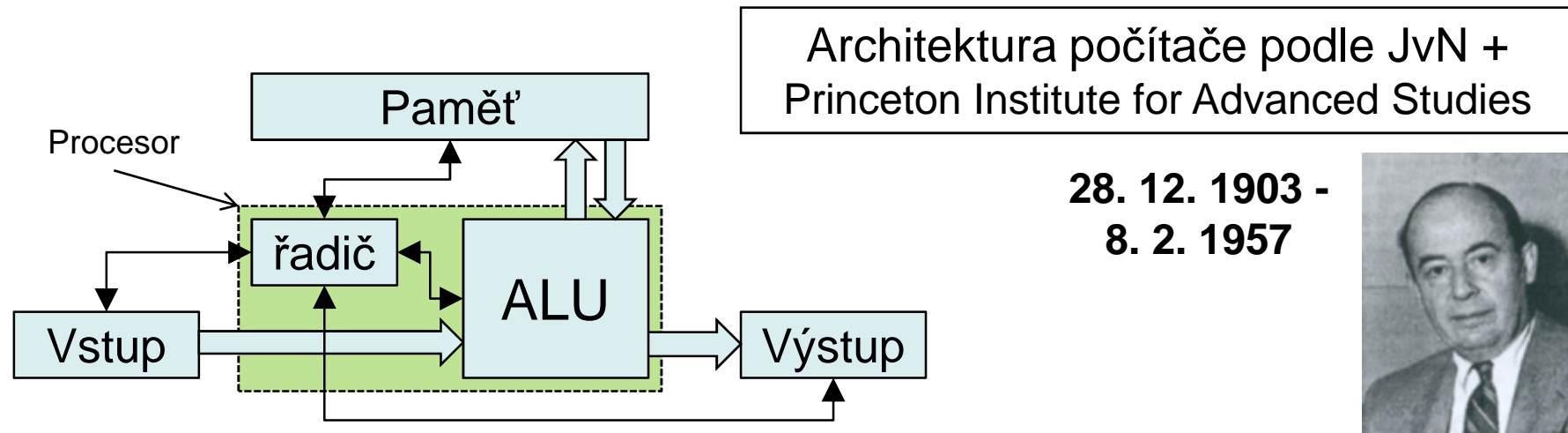
```
int matrix[M][N];
int i,j,sum=0;
...
for(j=0;j<N;j++)
  for(i=0;i<M;i++)
    sum+=matrix[i][j];
```

Lze doporučit výhodnější způsob procházení matice?

Osnova přednášky

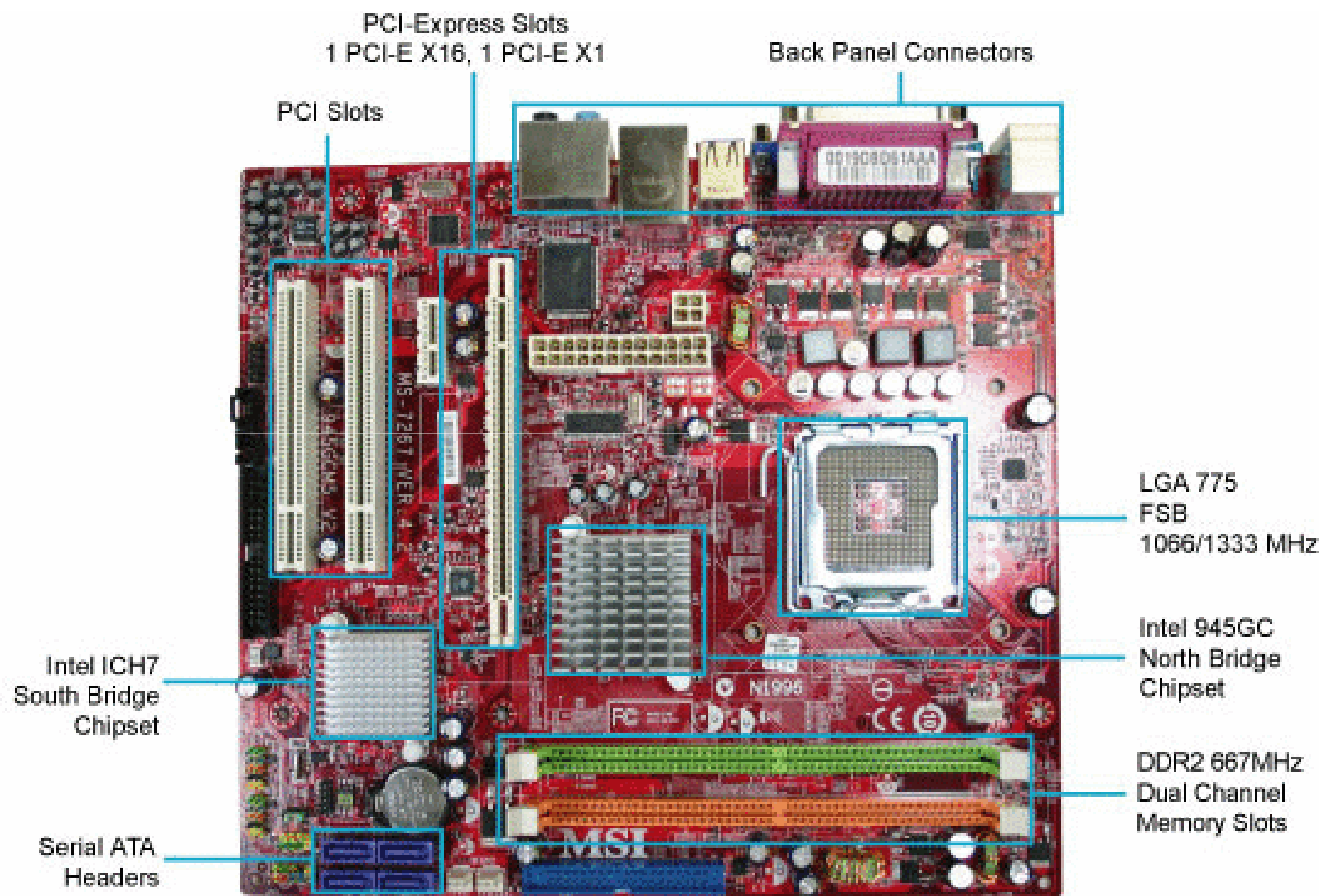
- Paměťová terminologie
- Hierarchie pamětí
- SP (skrytá paměť), resp. cache
 - Více realistické SP
 - Víceúrovňové SP
- Virtuální paměť
- Problémy hierarchických pamětí
- Realizace pamětí - paměťové čipy
- Jiné principy vedlejších pamětí

John von Neumann, maďarský fyzik



- 5 funkčních jednotek – řídicí jednotka (řadič), aritmeticko-logická jednotka, paměť, vstupní zařízení, výstupní zařízení
- Nezávislost struktury počítače na zpracovávaných problémech. Musí se zavést program a musí se uložit do paměti. Ten řídí činnost počítače.
- **Programy a výsledky (data) se ukládají do téže paměti.** Ta je rozdělena na stejně velké části (buňky), které jsou průběžně očíslované – adresa.
- Po sobě jdoucí instrukce se ukládají do po sobě jdoucích buněk.
- Existují instrukce aritmetické, logické, přenosu, skokové a ostatní.

Architektura počítače

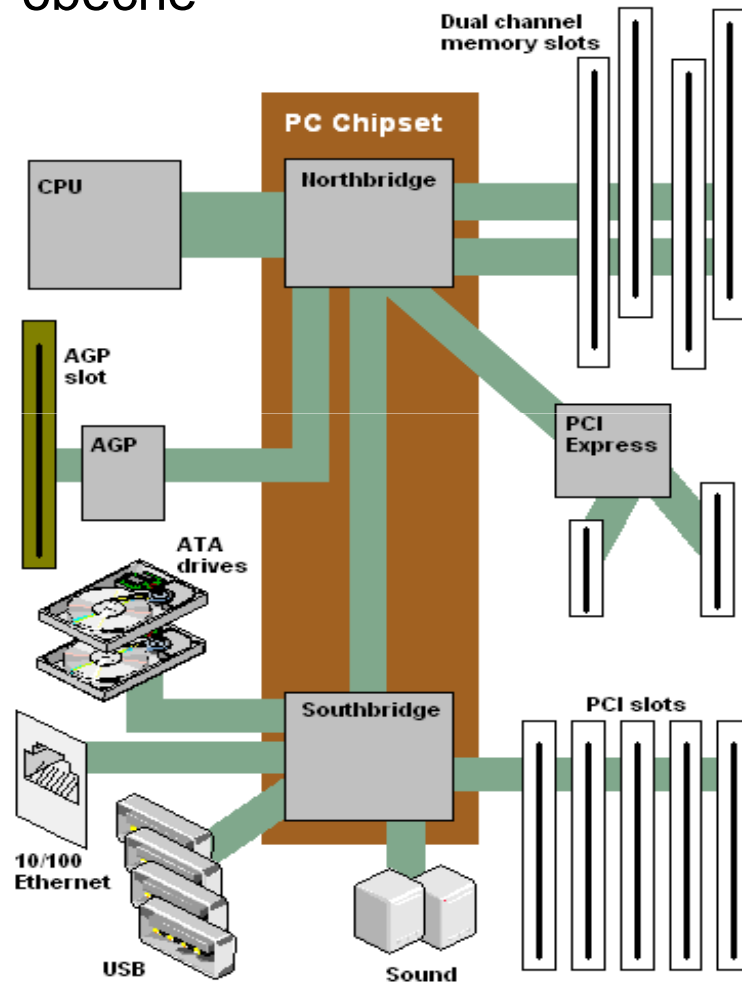


Architektura počítače

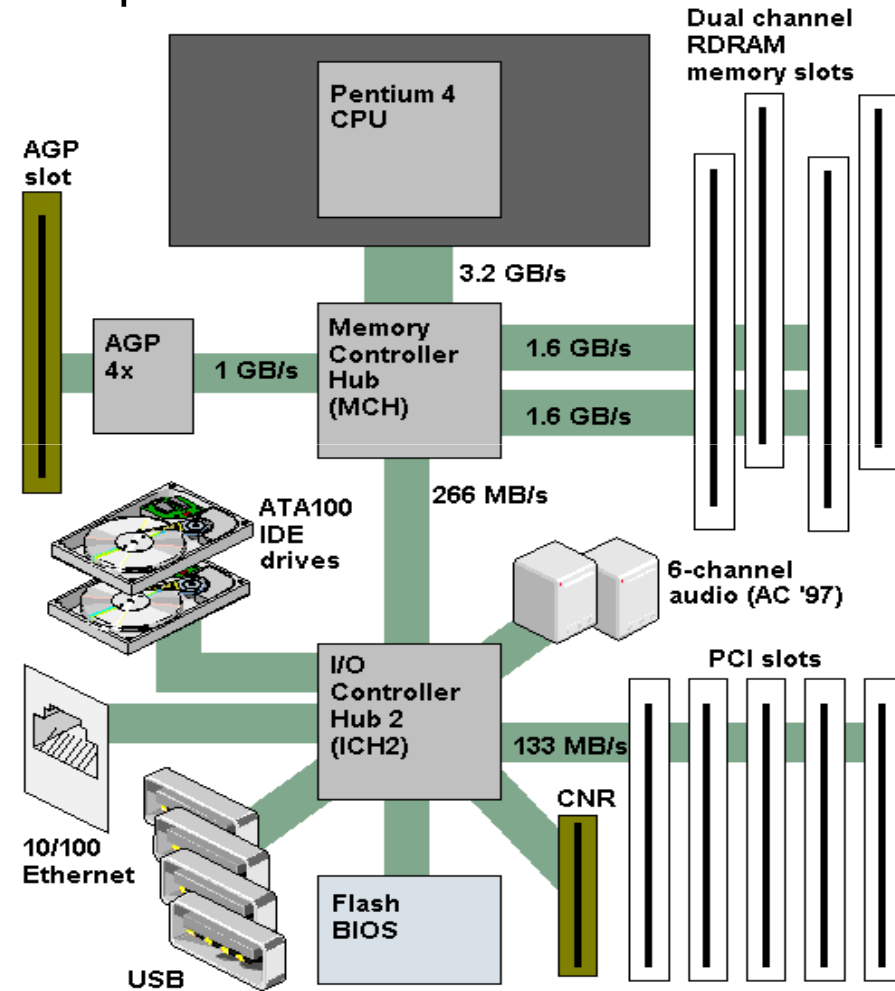
From Computer Desktop Encyclopedia
© 2005 The Computer Language Co., Inc.

From Computer Desktop Encyclopedia
© 2001 The Computer Language Co., Inc.

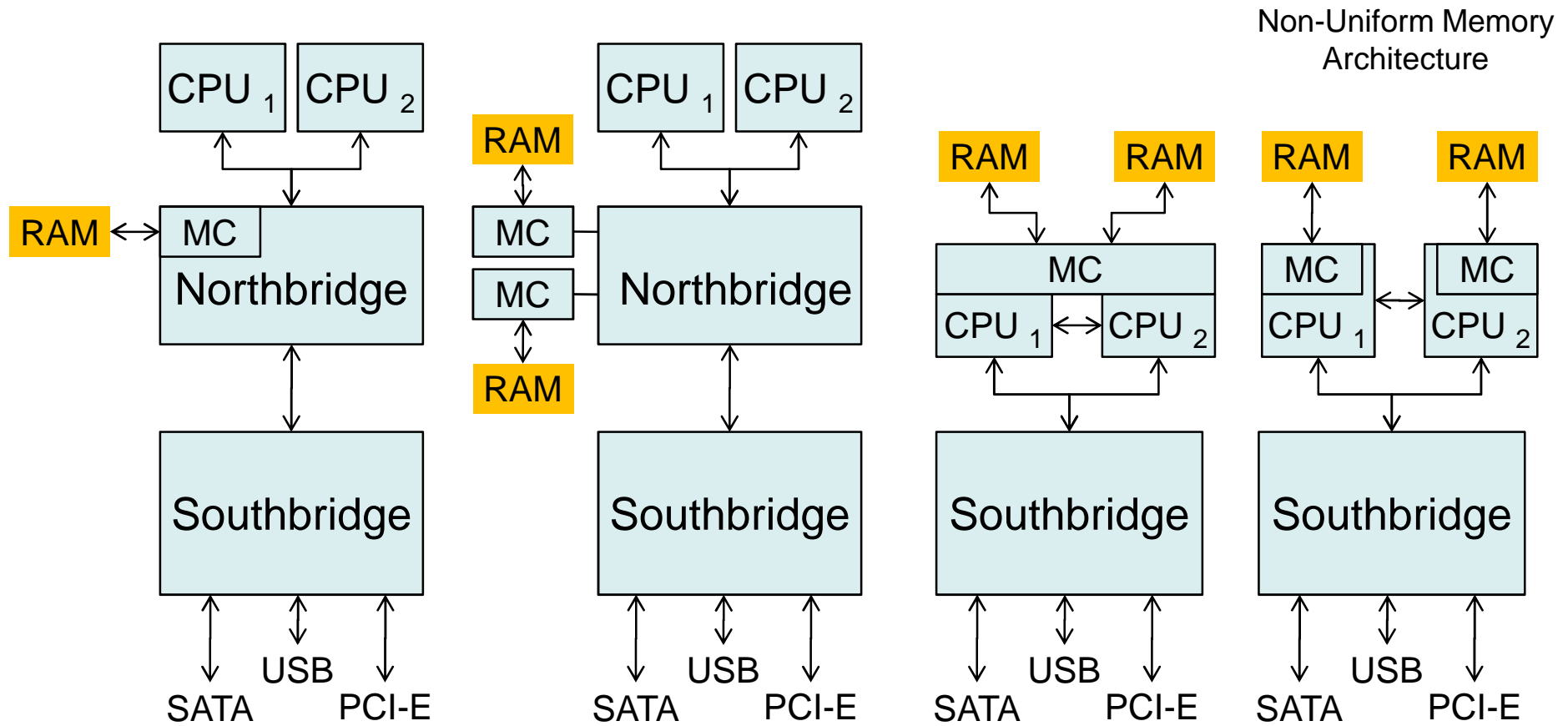
obecně



příklad

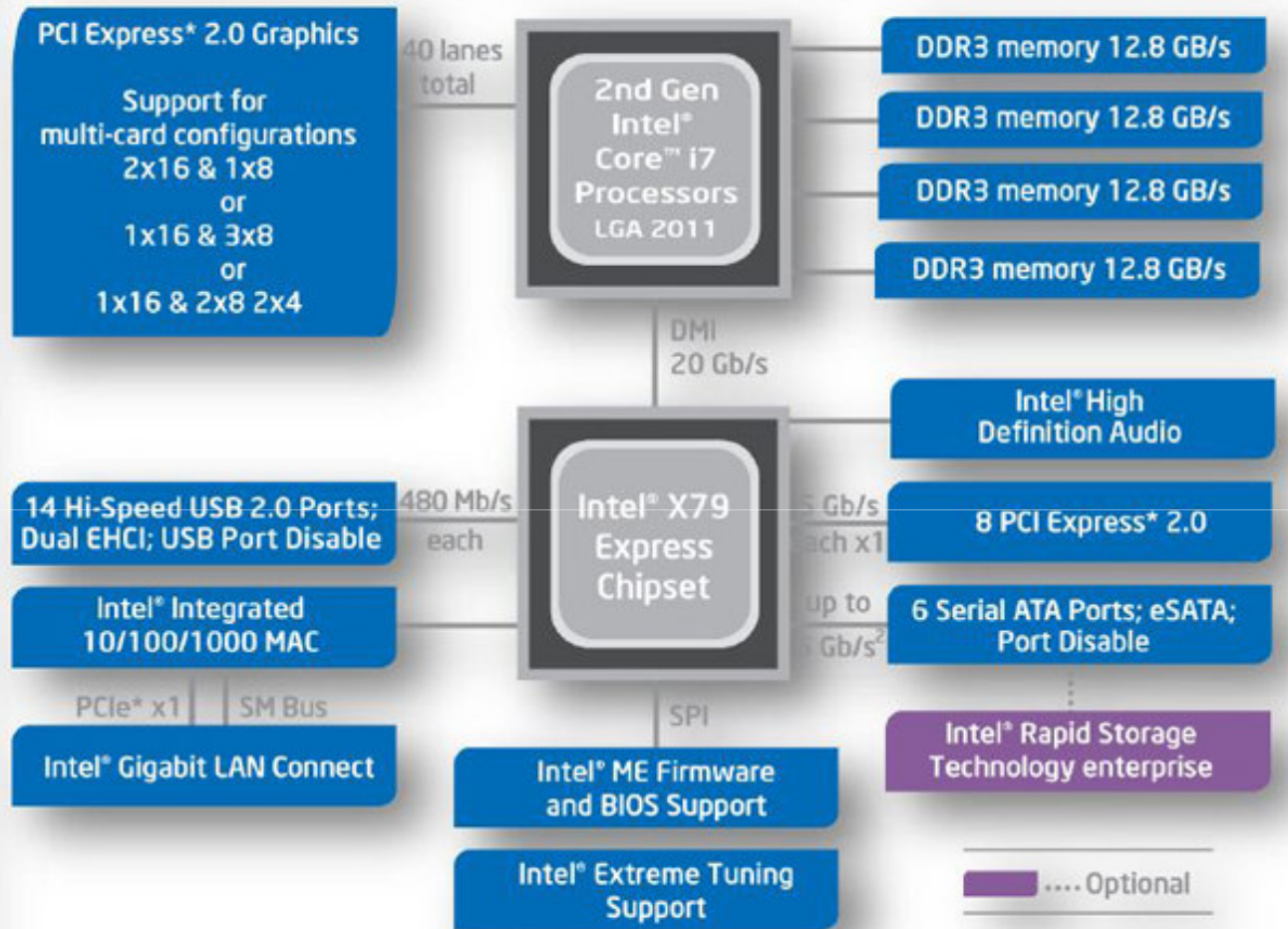


Architektura počítače



MC - Memory controller – obsahuje obvody pro zajištění operace čtení a zápisu z/do paměti. Také se stará o udržení obsahu paměti – refreshing každých několik desítek ms

Konkrétněji...



¹Theoretical maximum bandwidth
²All SATA ports capable of 3 Gb/s. 2 ports capable of 6 Gb/s.

Intel X79 Express Chipset Block Diagram

Detaily v přednáškách č.5 a 6

Terminologie kolem paměti

- Adresa, pojem snad není třeba vysvětlovat.
- Hodnota, vlastní informace. Paměťová buňka však může obsahovat i další informaci (třeba o platnosti hodnoty, apod.).
- **Parametry paměti:**
 - Vybavovací doba paměti, kritický parametr. Délka časového intervalu mezi objevením se požadavku a okamžikem, kdy jsou data k dispozici.
 - Doba přístupu, zastaralý parametr; vybavovací doba + obnovení obsahu po destruktivním čtení.
 - Propustnost, výkonový parametr. Schopnost zpracovat uvedené množství za jednotku času.
 - Latence = zpoždění, podobně jako vybavovací doba.

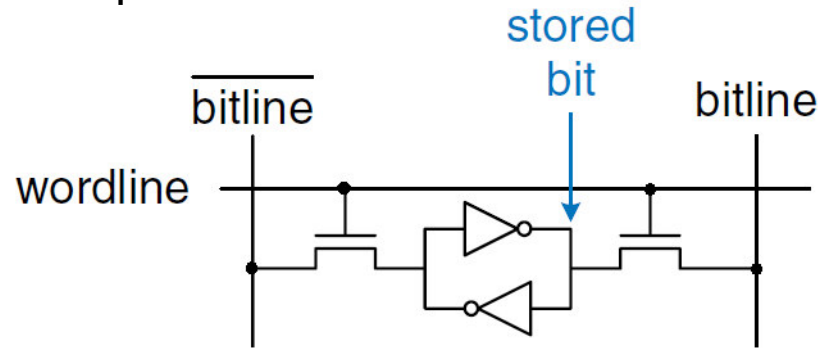
Terminologie kolem pamětí

- Typy pamětí RWM (RAM), ROM, FLASH,
- Provedení RAM pamětí:
SRAM (statická), **DRAM** (dynamická).
- RAM = *Random Access Memory* – paměť s **libovolným** přístupem

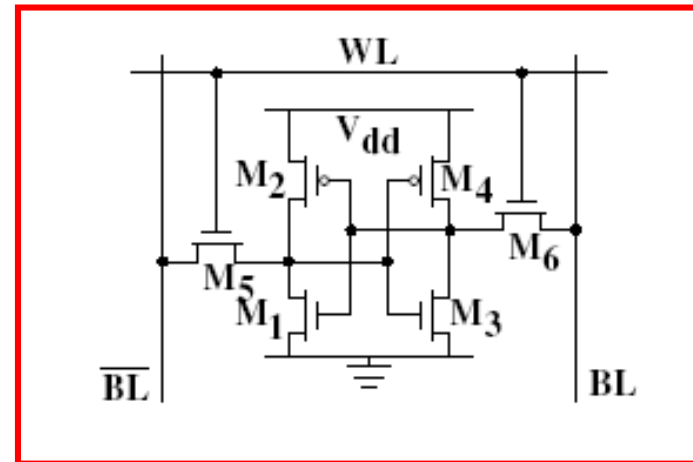
typ paměti	počet tranzistorů	plocha na 1 bit	dostupnost dat	latence
SRAM	cca 6	$< 0,1 \mu\text{m}^2$	vždy	$< 1\text{ns} - 5\text{ns}$
DRAM	1	$< 0,001 \mu\text{m}^2$	potřebuje refresh	desítky ns

Typický čip a buňka SRAM

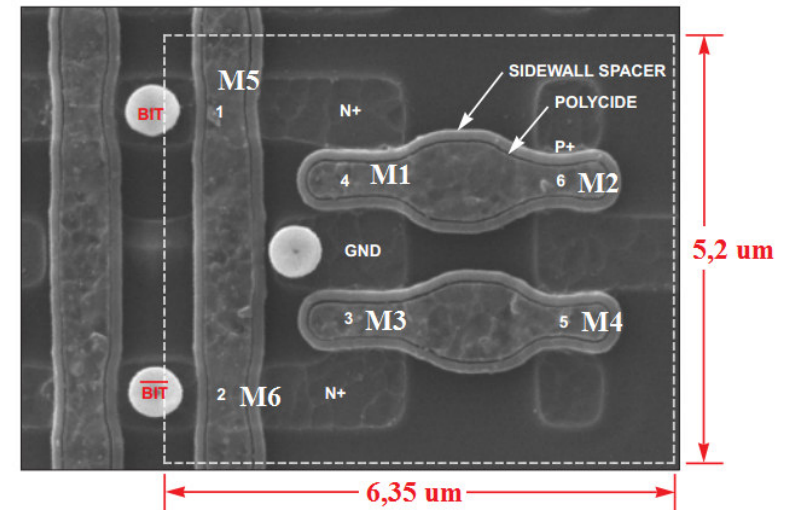
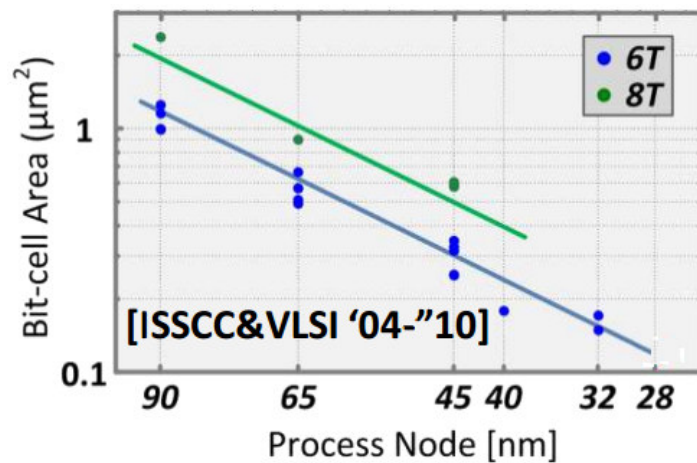
Princip:



SRAM paměťová buňka
technologie CMOS

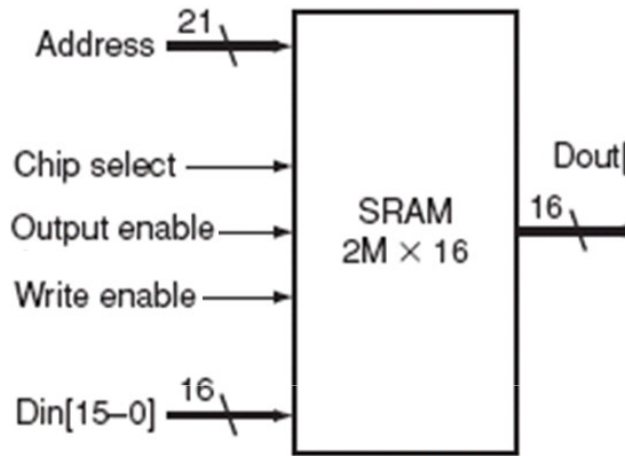


Plocha paměťové buňky:

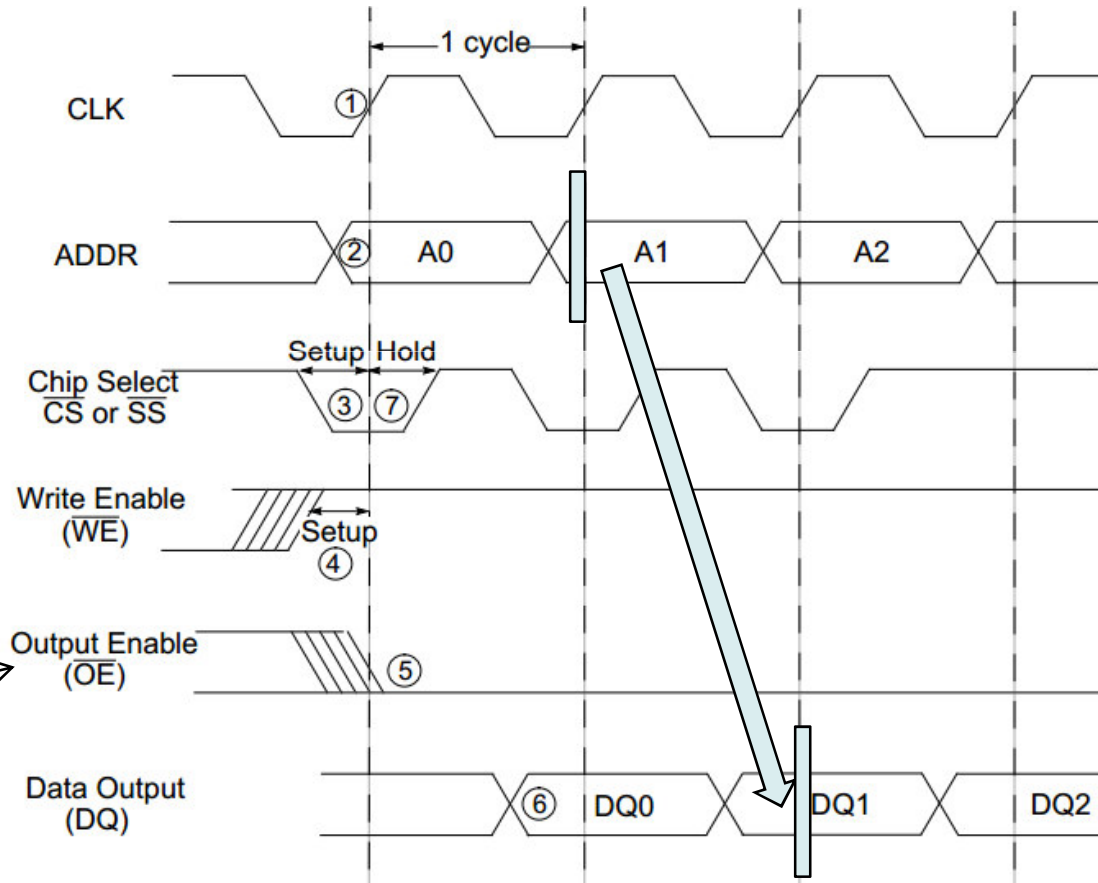


Typický čip a buňka SRAM

Typický SRAM čip



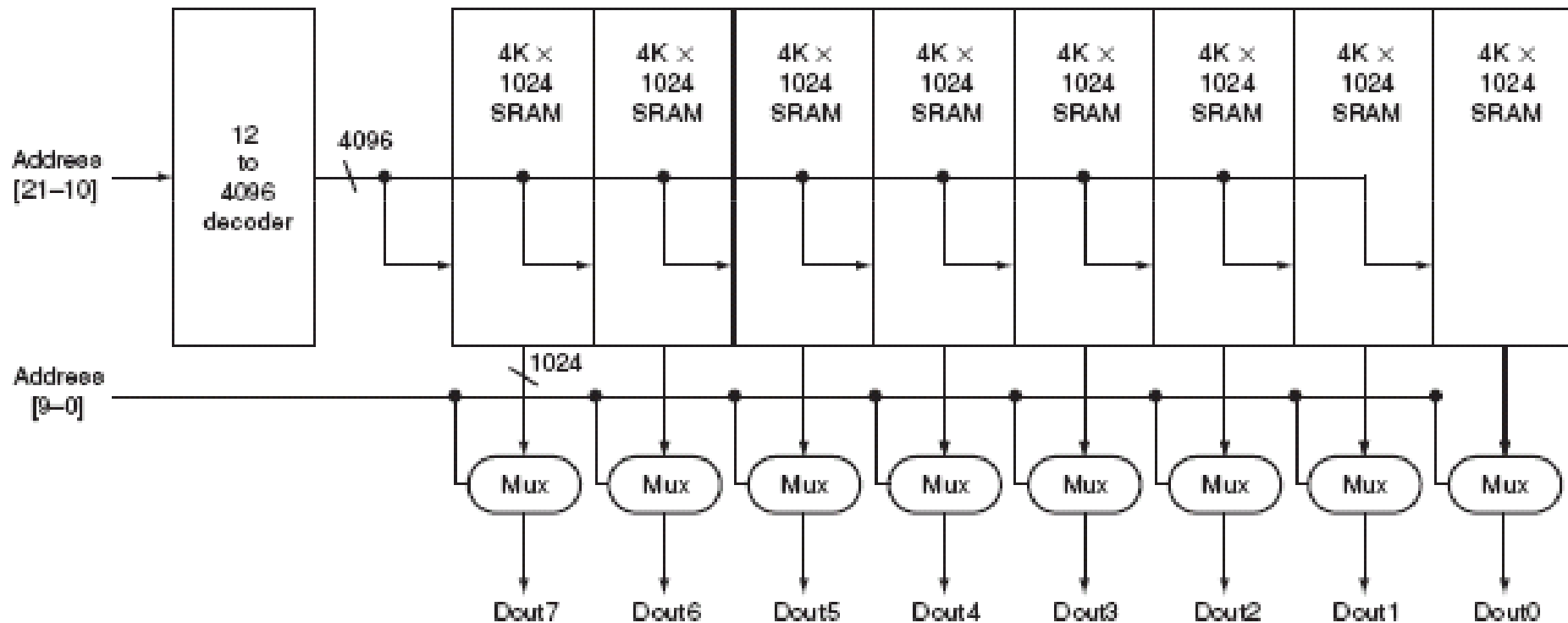
Příklad čtení – typicky synchronní :



OE signál je asynchronní

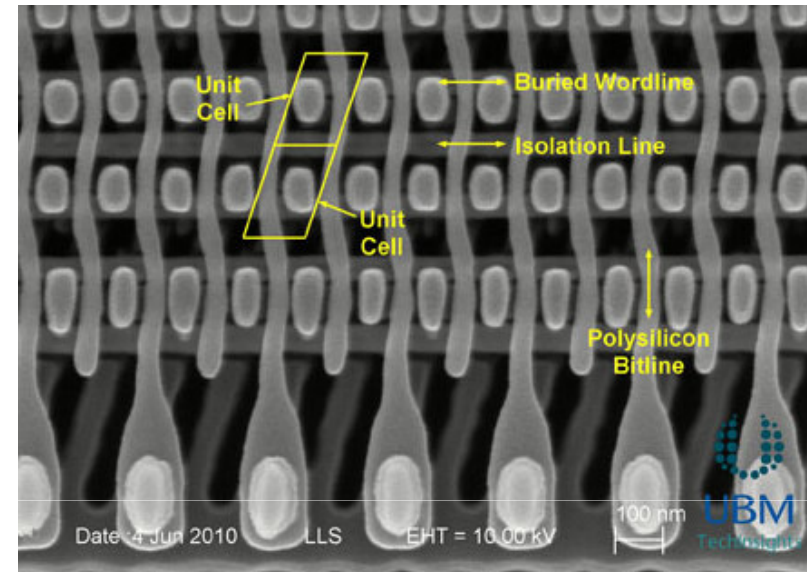
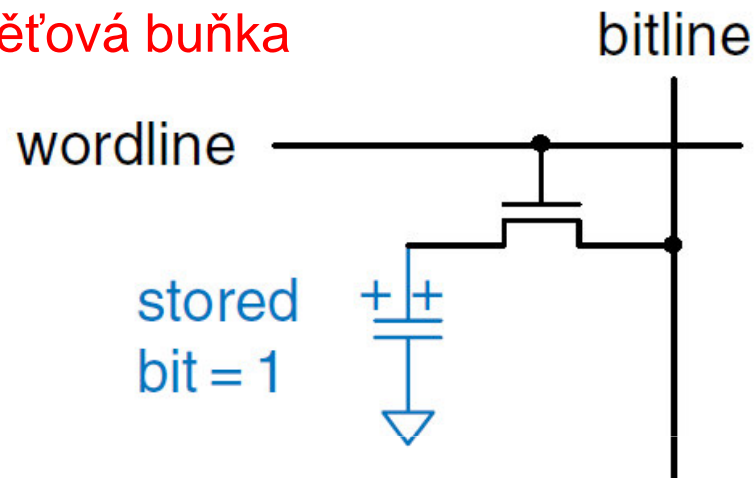
Typický čip a buňka SRAM

Větší paměť?



Detail paměťové buňky dynamické paměti

Jednotranzistorová dynamická paměťová buňka

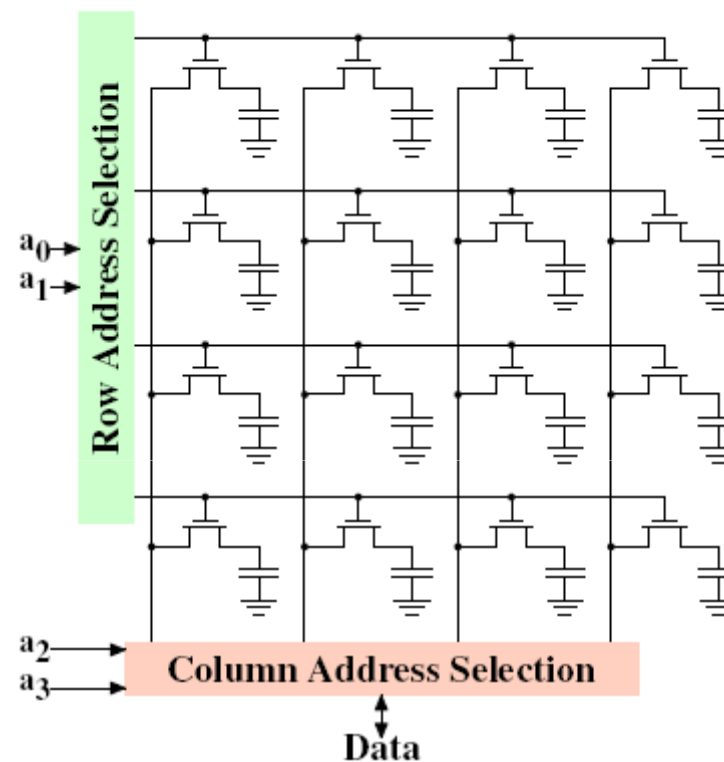
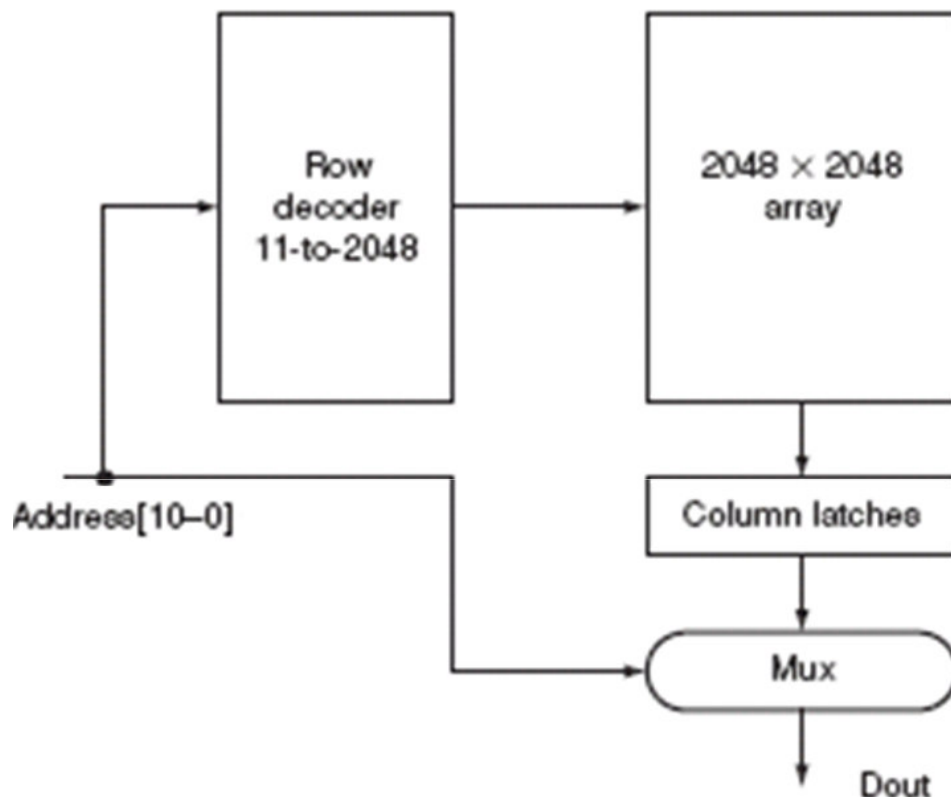


nMOS tranzistor představuje přepínač, který připojí (nebo ne) kondenzátor na vodič „bitline“. Připojení je řízeno vodičem „wordline“.

Proces čtení kondenzátor vybíjí. Proto musí být poté obnoven.

Občerstvování pamětí (refresh) – náboj se z kondenzátoru samovolně ztrácí. Nezbytná pracovní fáze dynamické paměti. Negativně ovlivňuje (prodlužuje) průměrnou vybavovací dobu.

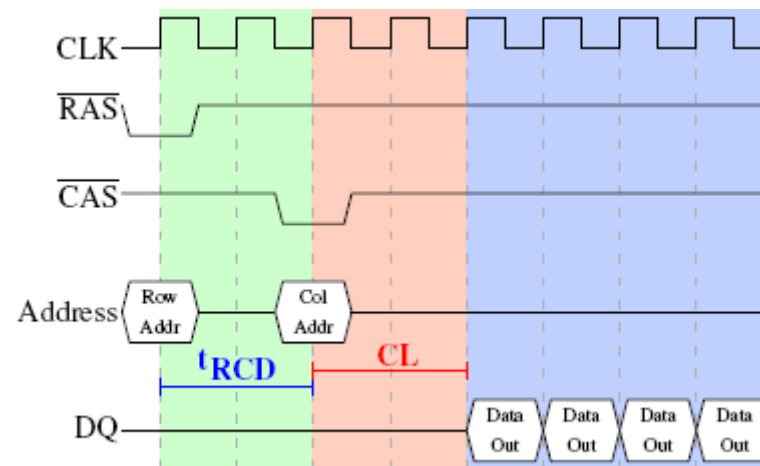
Vnitřní organizace čipu DRAM paměti



4M × 1 DRAM čip je uvnitř realizován
jako pole 2048 × 2048 1b paměťových buněk

Vývoj DRAM paměťových čipů v čase

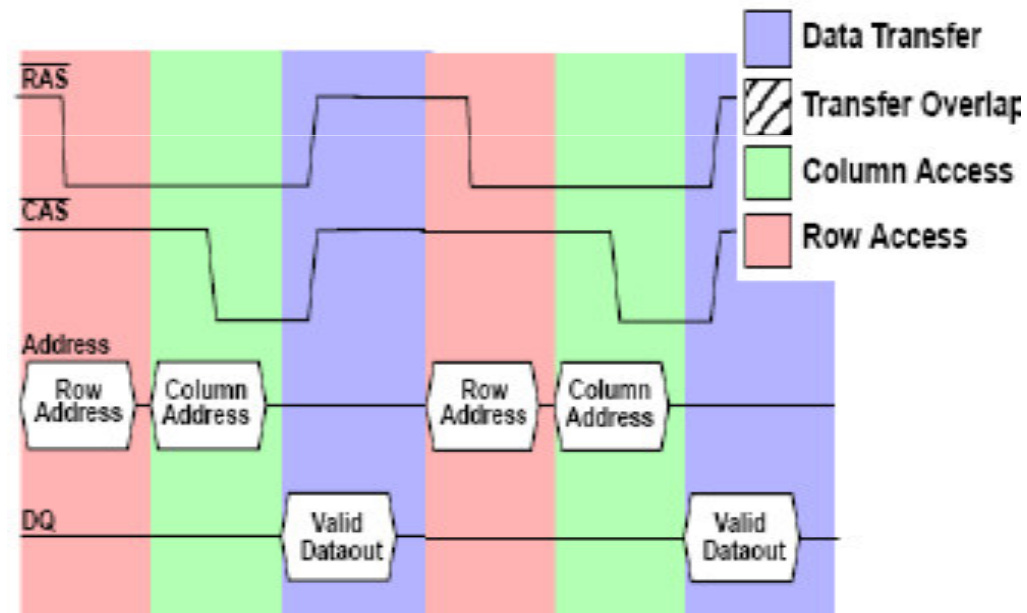
Rok	Kapacita	Cena[\$]/GB	Doba přístupu [ns]
1980	64 Kb	1 500 000	250
1983	256 Kb	500 000	185
1985	1 Mb	200 000	135
1989	4 Mb	50 000	110
1992	16 Mb	15 000	90
1996	64 Mb	10 000	60
1998	128 Mb	4 000	60
2000	256 Mb	1 000	55
2004	512 Mb	250	50
2007	1 Gb	50	40



RAS – Row Address Strobe,
CAS – Column Address Strobe

Klasická DRAM – asynchronní rozhraní

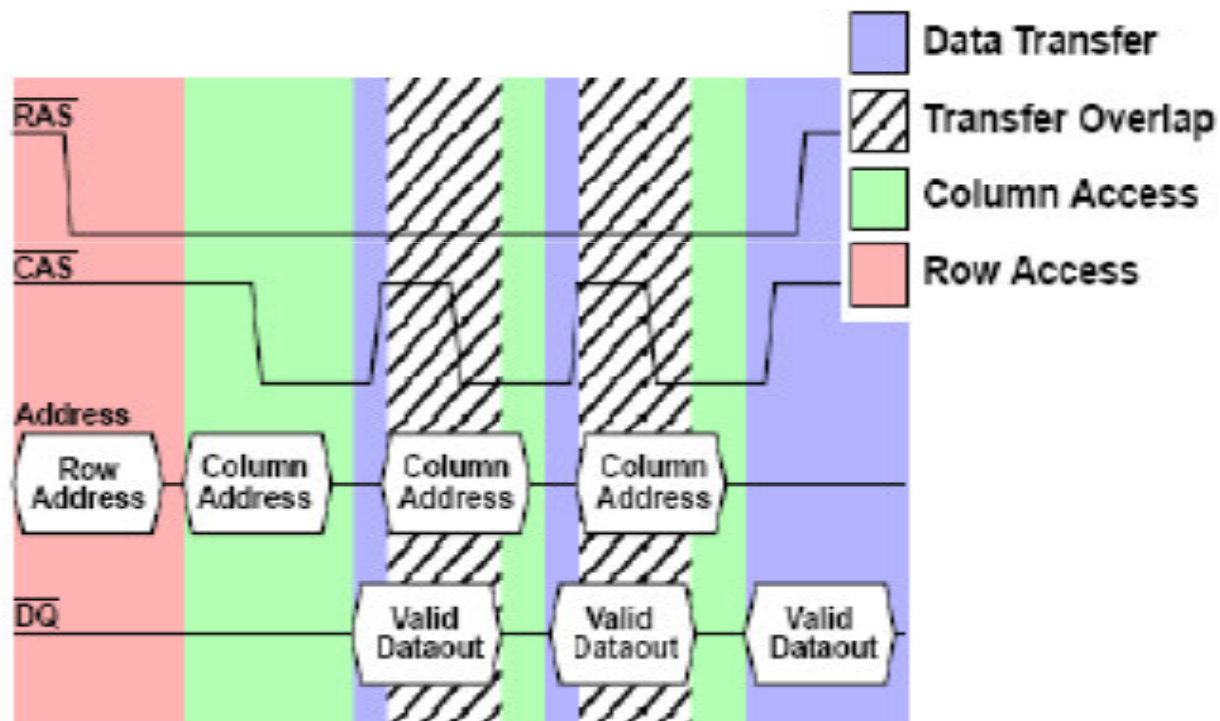
- Důvod rozdělení adresy na 2 části byl dán malým počtem pinů původních DRAM pouzder.
- Toto rozdělení se dodnes zachovává, ačkoli pouzdro už není problém. Uneslo by více vývodů...



RAS – Row Address Strobe,
CAS – Column Address Strobe

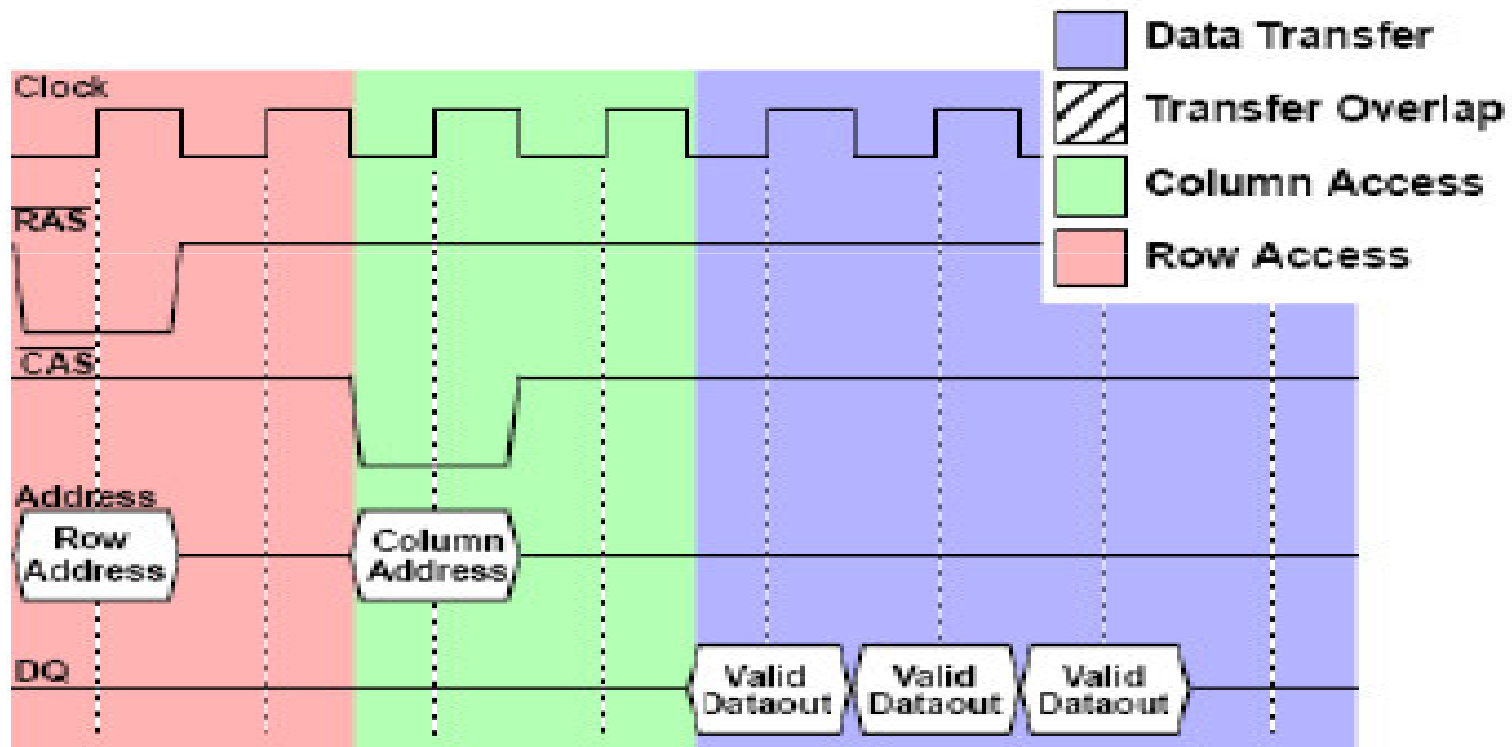
EDO DRAM – cca 1995

- EDO DRAM má registr na výstupu, což umožní překrýt následující CAS s čtením předchozích dat.



SDRAM – konec 90.let – synchronní DRAM

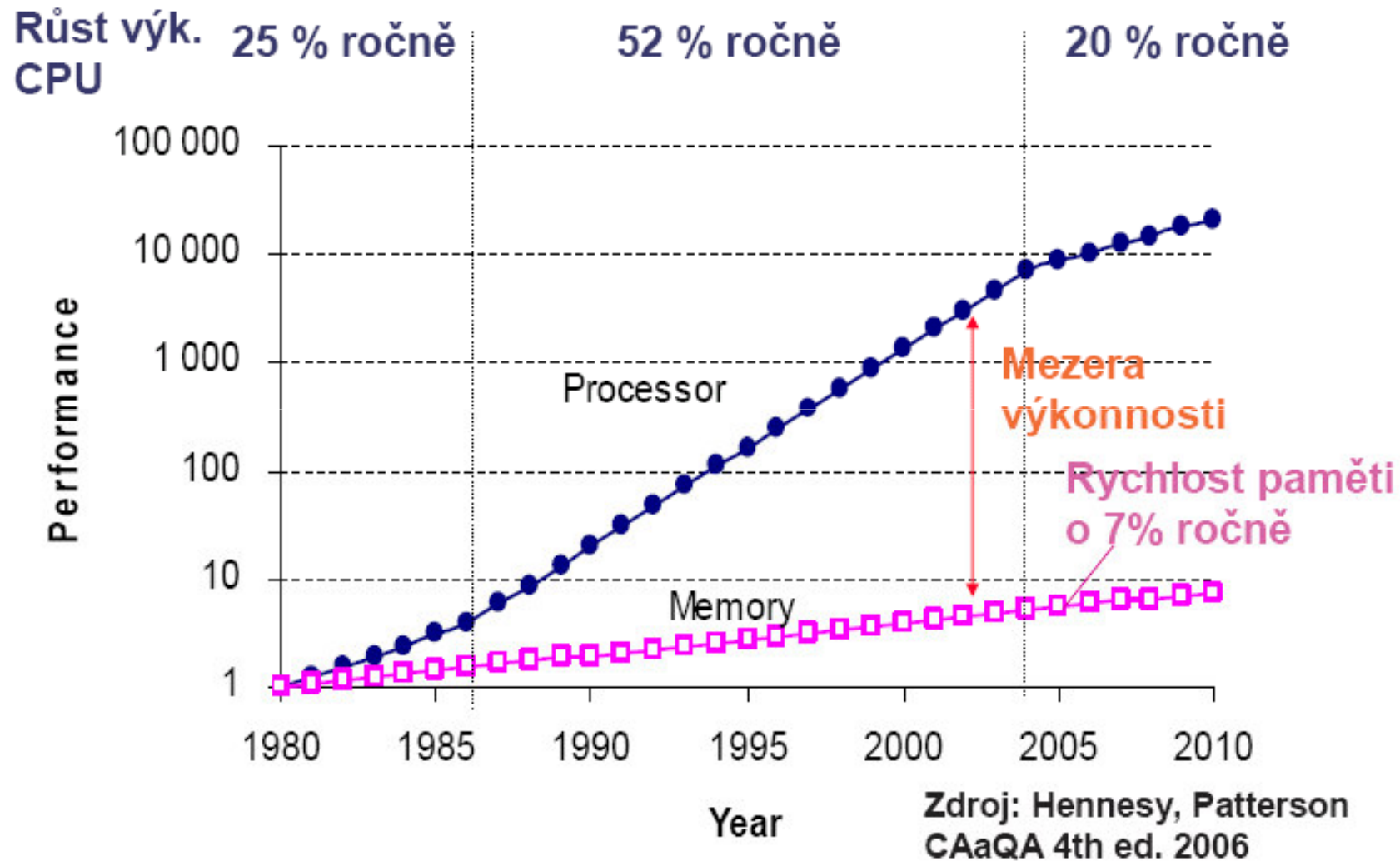
- SDRAM čip obsahuje čítač, který umožňuje nastavit délku souvislého (burst) čtení dat.



SDRAM – paměť současnosti

- **SDRAM** – frekvence až 100 MHz, 2.5V.
- **DDR SDRAM** – použití obou hran CLK při přenosu dat, 2.5V.
- **DDR2 SDRAM** – snížení spotřeby použitím napětí 1.8V, frekvence až 400 MHz.
- **DDR3 SDRAM** – snížení spotřeby použitím napětí 1.5V, frekvence až 800 MHz.
- **DDR4 SDRAM** ...
- Dále ještě existují paměti a moduly **RAMBUS**, které ovšem mají zcela odlišné rozhraní i způsob použití.
- **Všechny tyto inovace vylepšují propustnost čtení dat, nikoli latenci čtení první položky.**

Disproporce ve výkonu proc x pam, Moorův zákon



Bubble sort – již znáte z cvičení

```
int pole[5]={5,3,4,1,2};
int main()
{
    int N = 5,i,j,tmp;
    for(i=0; i<N; i++)
        for(j=0; j<N-1-i; j++)
            if(pole[j+1]<pole[j])
            {
                tmp = pole[j+1];
                pole[j+1] = pole[j];
                pole[j] = tmp;
            }
    return 0;
}
```

Jakou
využitelnou
vlastnost mají
naše programy?

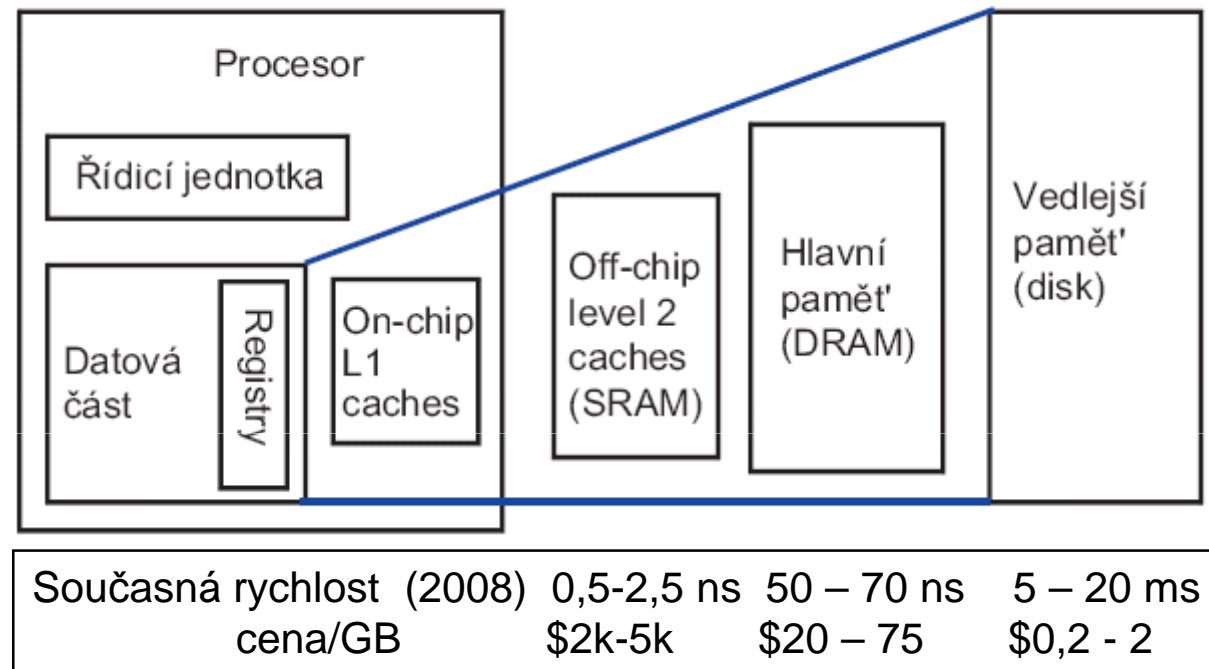
Paměťová hierarchie – základní principy

- Programy/procesy přistupují v daném okamžiku **jen k malé části** svého adresového prostoru
- **Časová lokalita**
 - Položky, ke kterým se přistupovalo nedávno, budou zapotřebí brzy znovu.
 - Příklad: programová smyčka, proměnné instrukcí.
- **Prostorová lokalita**
 - Položky poblíž právě používaným budou brzy zapotřebí také.
 - Příklad: sekvenční přístup ke kódu (paměť programu), datová pole (paměť dat).

Co z uvedeného plyne?

- Je výhodné uspořádat paměťový prostor hierarchicky – paměťová hierarchie.
- Všechny potřebné informace uchovávejte v sekundární paměti.
- Položky nedávno používané a blízké zkopírujte do (menší) paměti implementované DRAM.
 - Operační paměť.
- Položky ještě častěji používané (i ty jim blízké) zkopírujte do menší a rychlejší SRAM.
 - Skrytá paměť.

Paměťová hierarchie



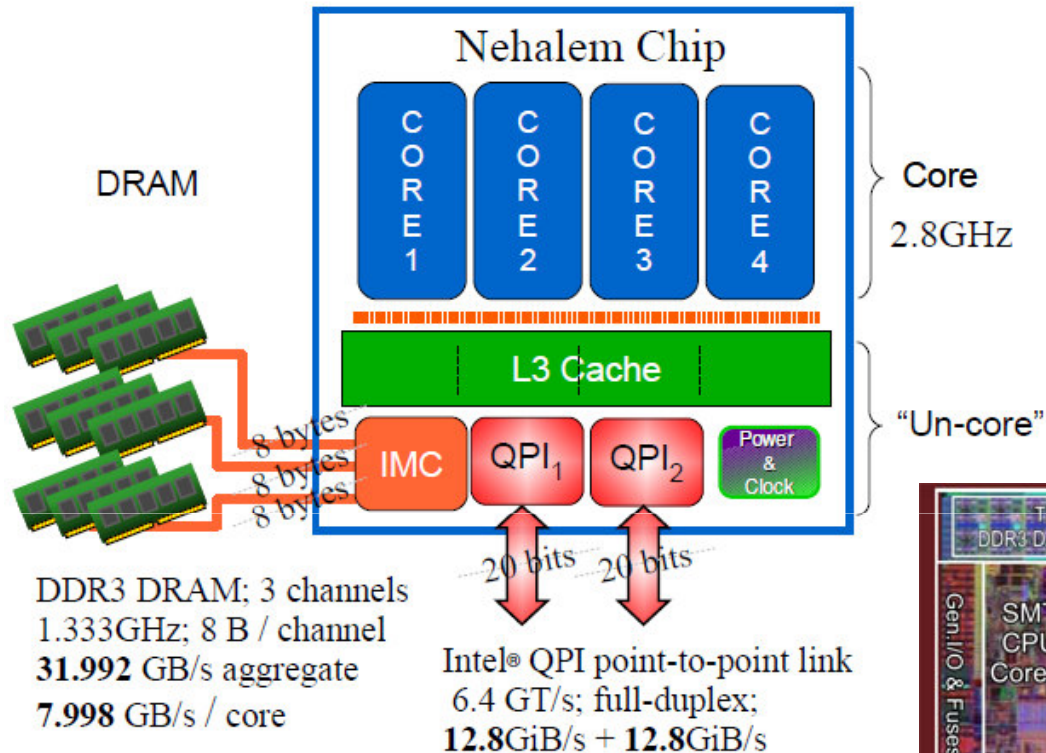
- To se jedna a tatáž informace může objevit na více místech hierarchické paměti? Ano.

Jak a kde pak ale hledanou informaci najdeme?

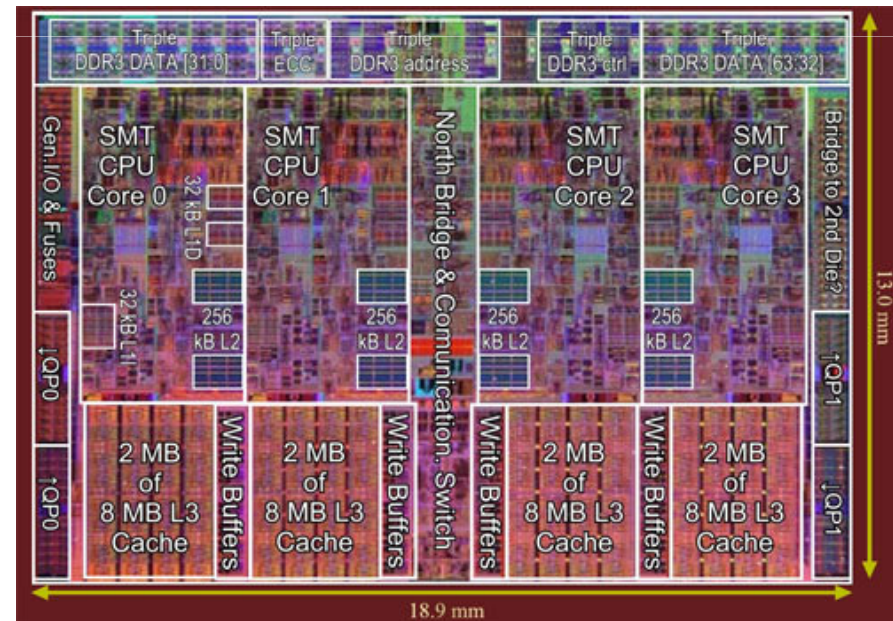
- Podle adresy a případně dalších informací (např. o platnosti).
- Hledat začneme v paměti nejvyšší hierarchické úrovni (nejblíže procesoru).
- Požadavky:
 - Paměťová konzistence.
- Prostředky:
 - Virtualizace adresy,
 - Mechanizmy uvolňování místa a migrace informace mezi paměťovými úrovněmi.
 - Hit, miss.

Příklad procesoru včetně cache

Harvardská architektura - Intel Nehalem



- IMC: integrated memory controller with 3 DDR3 memory channels,
- QPI: Quick-Path Interconnect ports
- Podívejte se na velikosti jednotlivých cache!!!



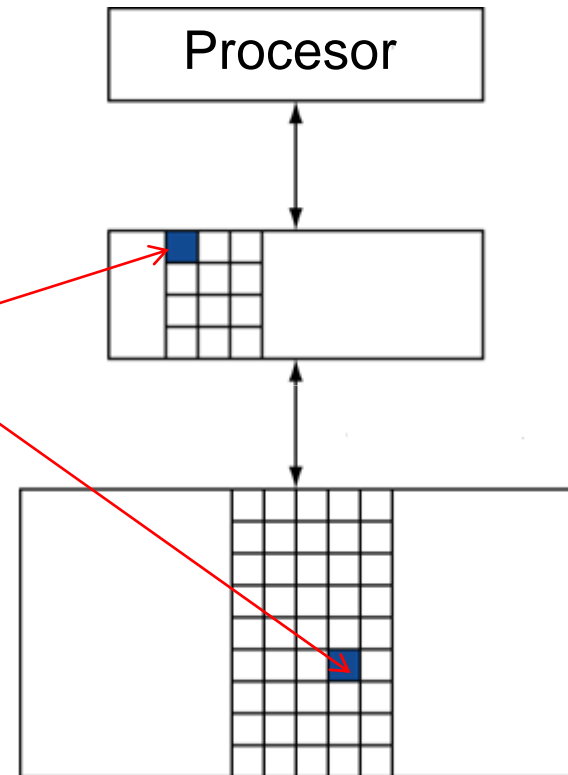
Řešení disproporce rychlosti? Skrytá paměť (SP) – cache.

Cache, česky keš

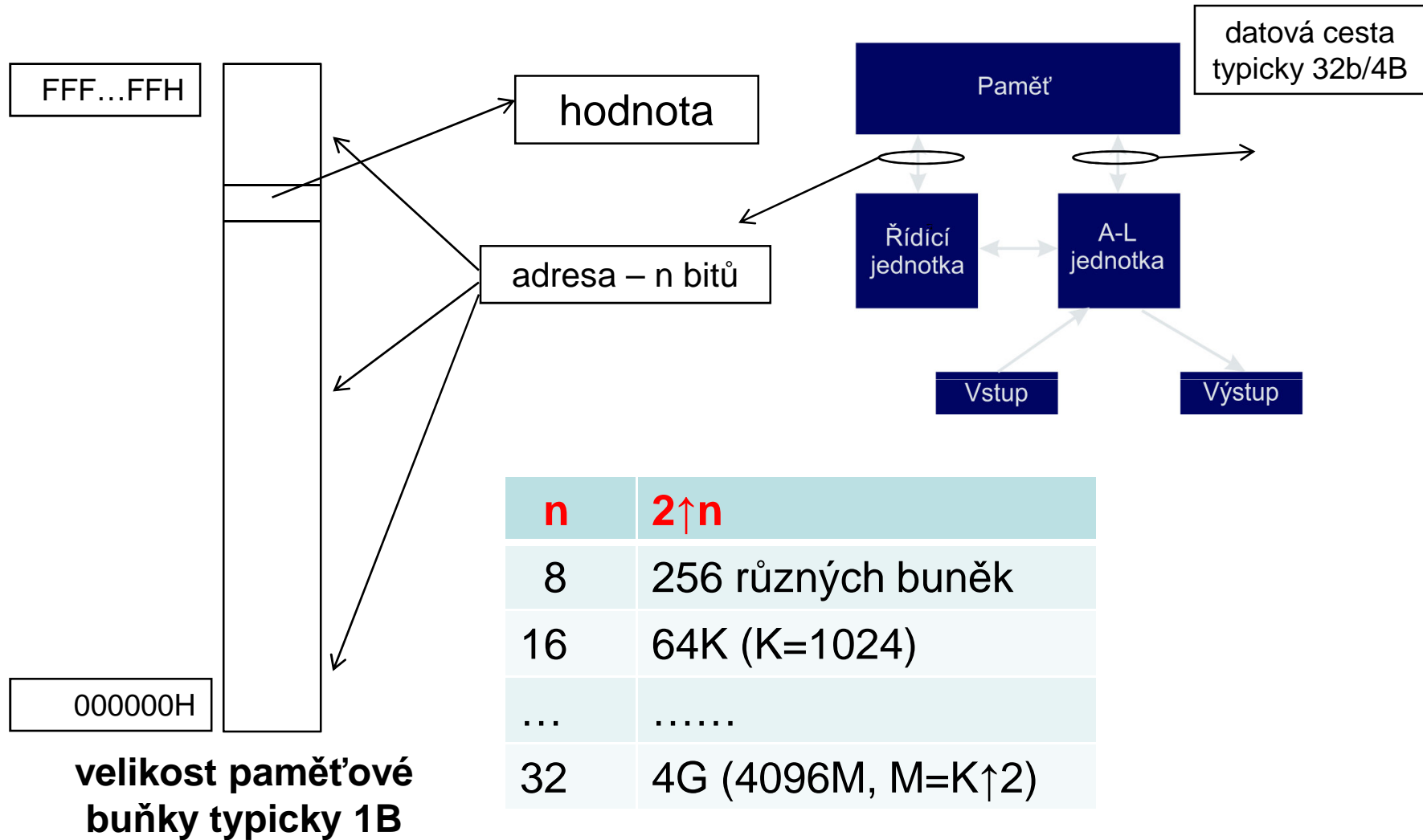
- Nebo-li **skrytá paměť**, SP,
- je označení pro vyrovnávací paměť používanou ve výpočetní technice.
- Zařazujeme ji mezi dva subsystémy s různou rychlostí. Vyrovnává se jí rychlost přístupu k informacím.
- Účelem skryté paměti je urychlit přístup k často používaným datům na „pomalých“ médiích jejich překopírováním na média rychlá.

Terminologie kolem skryté paměti

- **Cache hit** pojmenování situace, kdy požadovaná hodnota ve skryté paměti (cache) **je**.
- **Cache miss**, opak. **Není** tam.
- **Cache line** nebo **Cache block** – základní kopírovatelná jednotka mezi hierarchickými úrovněmi.
- V praxi se velikost Cache Line pohybuje od 8B do 1KB, typicky 64B.



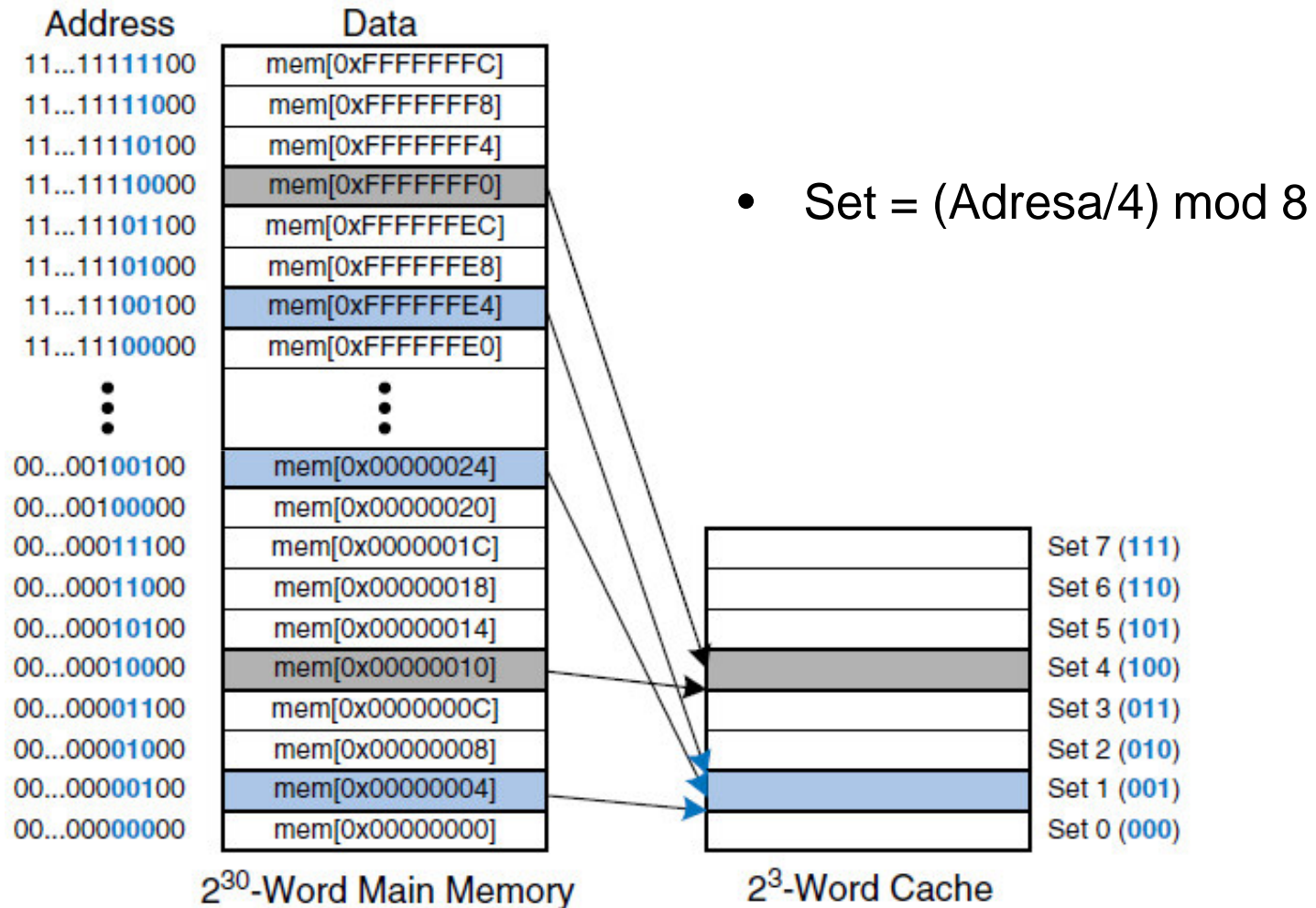
Co je to adresní prostor? Adresa/hodnota



Příklad

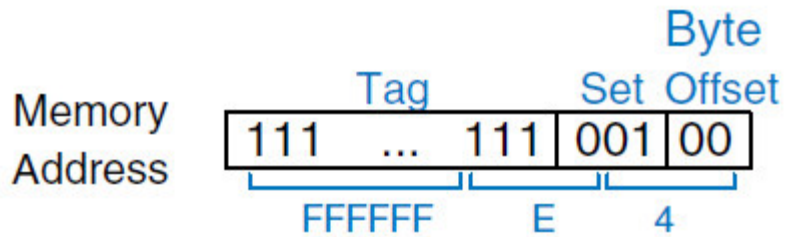
- Mějme cache o velikosti 8-mi bloků. Kam se do ní umístí data z adresy 0xF0000014?
 - Plně asociativní,
 - Přímě mapované, nebo
 - S omezeným stupněm asociativity $N=2$ (2-cestná, 2-way cache).

Přímo mapovaná cache



Přímo mapovaná cache

přímo mapovaná cache:
one block in each set



Capacity – C

Number of sets – S

Block size – b

Number of blocks – B

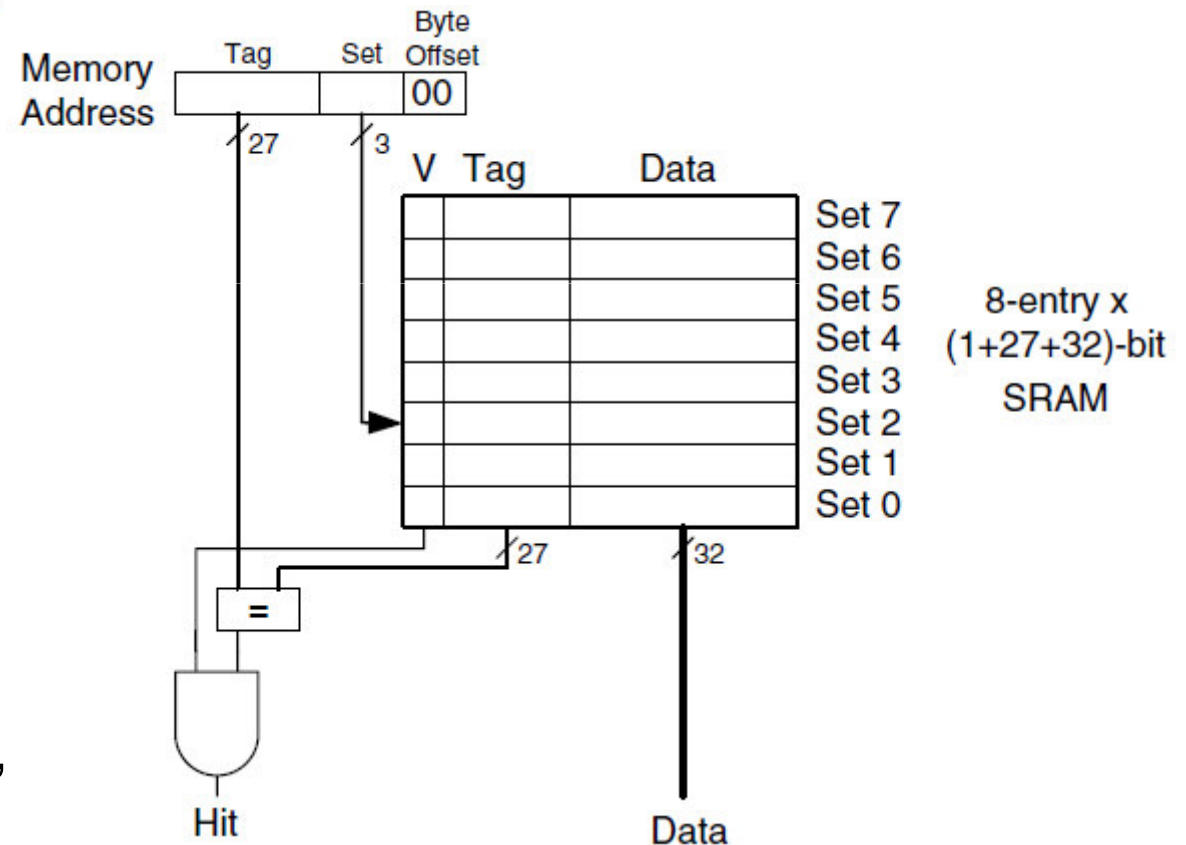
Degree of associativity – N

C = 8 (8 words),

S = B = 8,

b = 1 (one word in the block),

N = 1



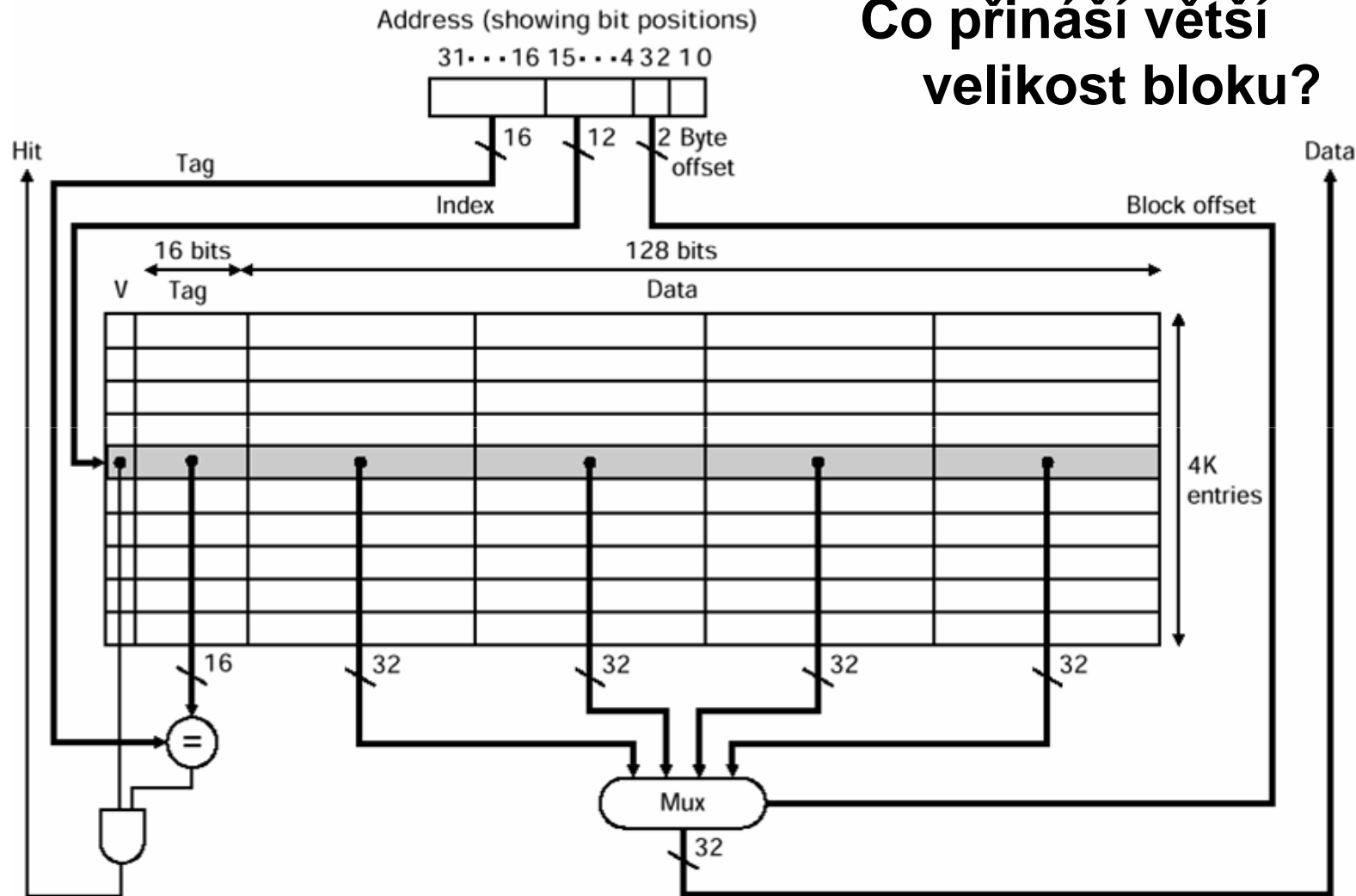
Realističtější formát řádky cache

- **Tag** je index odpovídajícího bloku v operační paměti (v podstatě se jedná o hodnotu ukazatele/adresy dělenou délkou bloku).
- **Data** pole obsahující vlastní hodnoty na příslušné/ných adrese/ách.
- **Validity bit** – bit platnosti. Indikuje, zda je obsah pole Data vůbec platný.
- **Dirty bit** – rozšiřující pole v obsahu paměti. Indikuje, že v cache (cache) je **jiná hodnota**, než v paměti hlavní.

V	Další bity, např. D	Tag	Data
---	---------------------	-----	------

Přímo mapovaná skrytá paměť – velikost bloku 4 slova

Co přináší větší velikost bloku?



Cache s omezeným stupněm asociativity N=2

Co přináší zvětšení stupně asociativity?

Capacity – C

Number of sets – S

Block size – b

Number of blocks – B

Degree of associativity – N

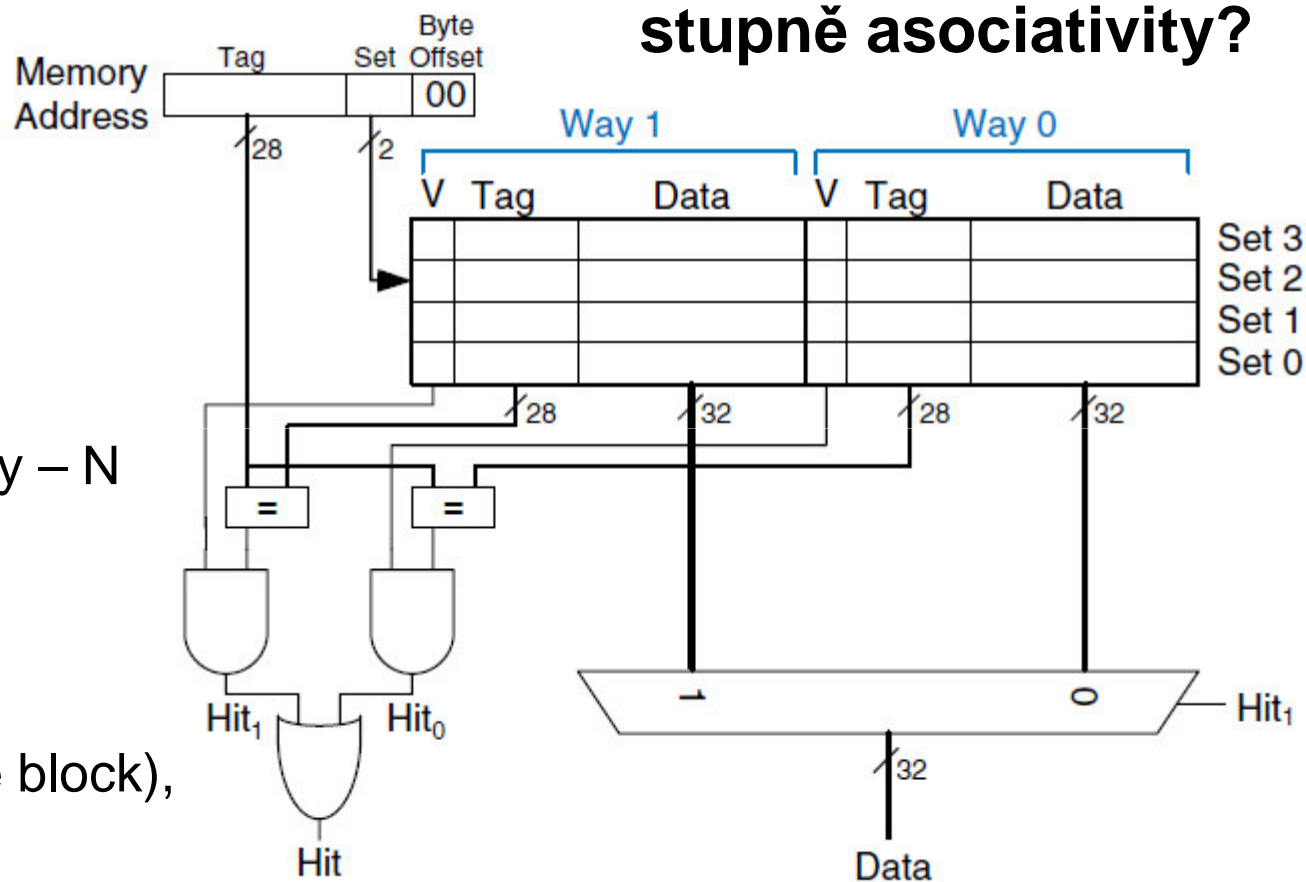
C = 8 (8 words),

S = 4,

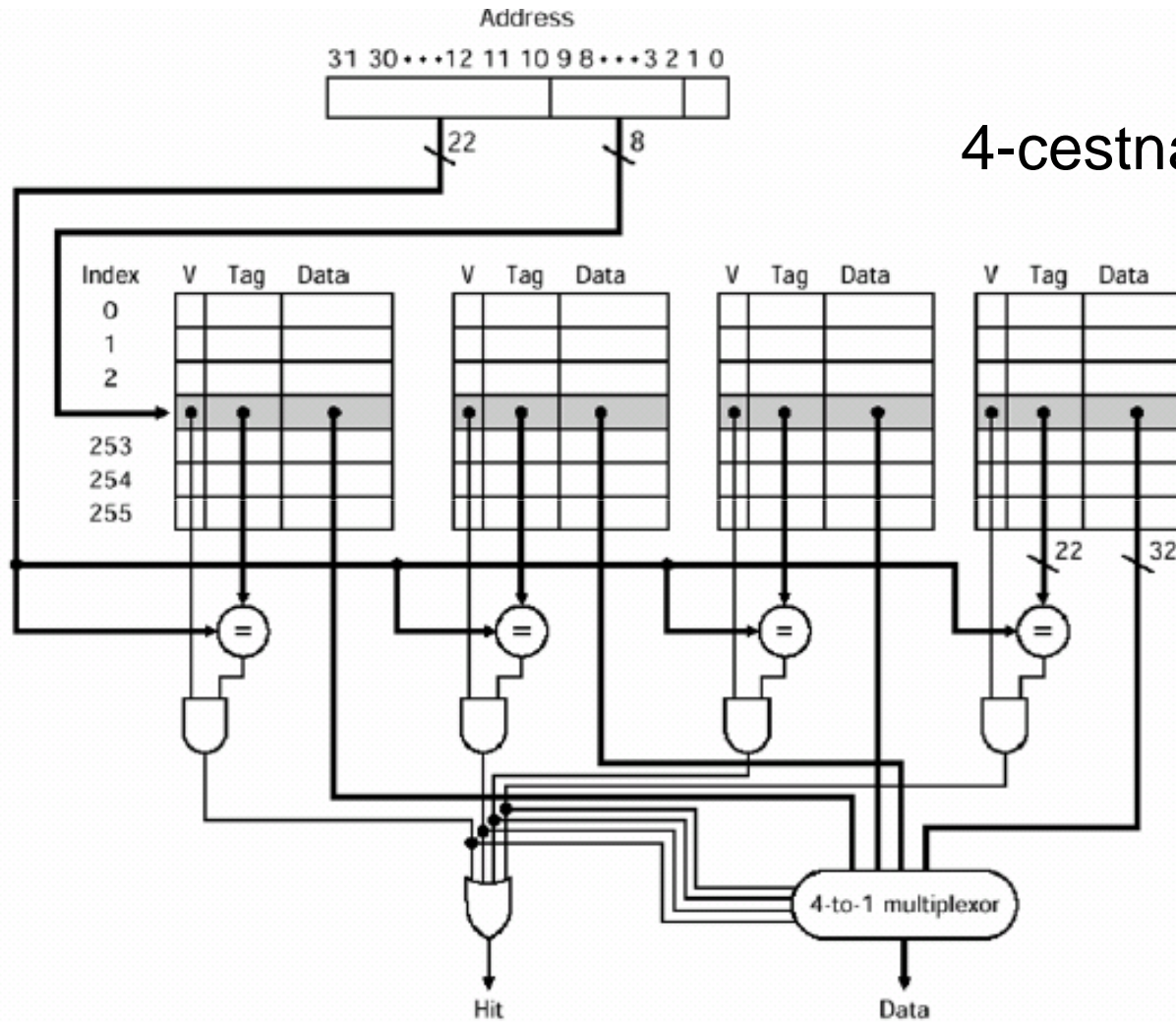
b = 1 (one word in the block),

B = 8

N = 2

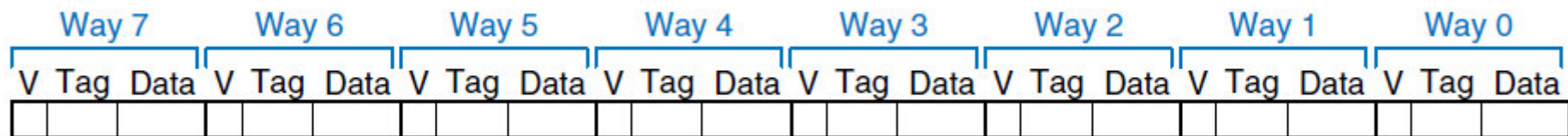


Cache s omezeným stupněm asociativity N=4



Plně asociativní cache

- Plně asociativní cache obsahuje jenom jeden set, stupeň asociativity je roven počtu bloků ($N=B$). Adresa paměti se může mapovat kamkoliv.
- ...je jiné pojmenování pro B-cestně asociativní cache s jedním setem
- ... má pro danou kapacitu má nejméně konfliktů, ale potřebuje nejvíce HW prostředků (komparátory) – roste plocha čipu
- ...je vhodná pro relativně malé cache



A fully associative cache has only $S=1$ set.

Diskuze k plně asociativní cache

- Šířka pole Tag odpovídá šířce adresy,
- Každý řádek cache obsahuje tolik jednobitových kompara-torů, kolik je šířka adresy,
- Počet řádků cache určuje její kapacitu,
- Cache musí mít strategii uvolňování obsahu (migrace dat mezi hierarchickými úrovněmi) v případě vyčerpání její kapacity.

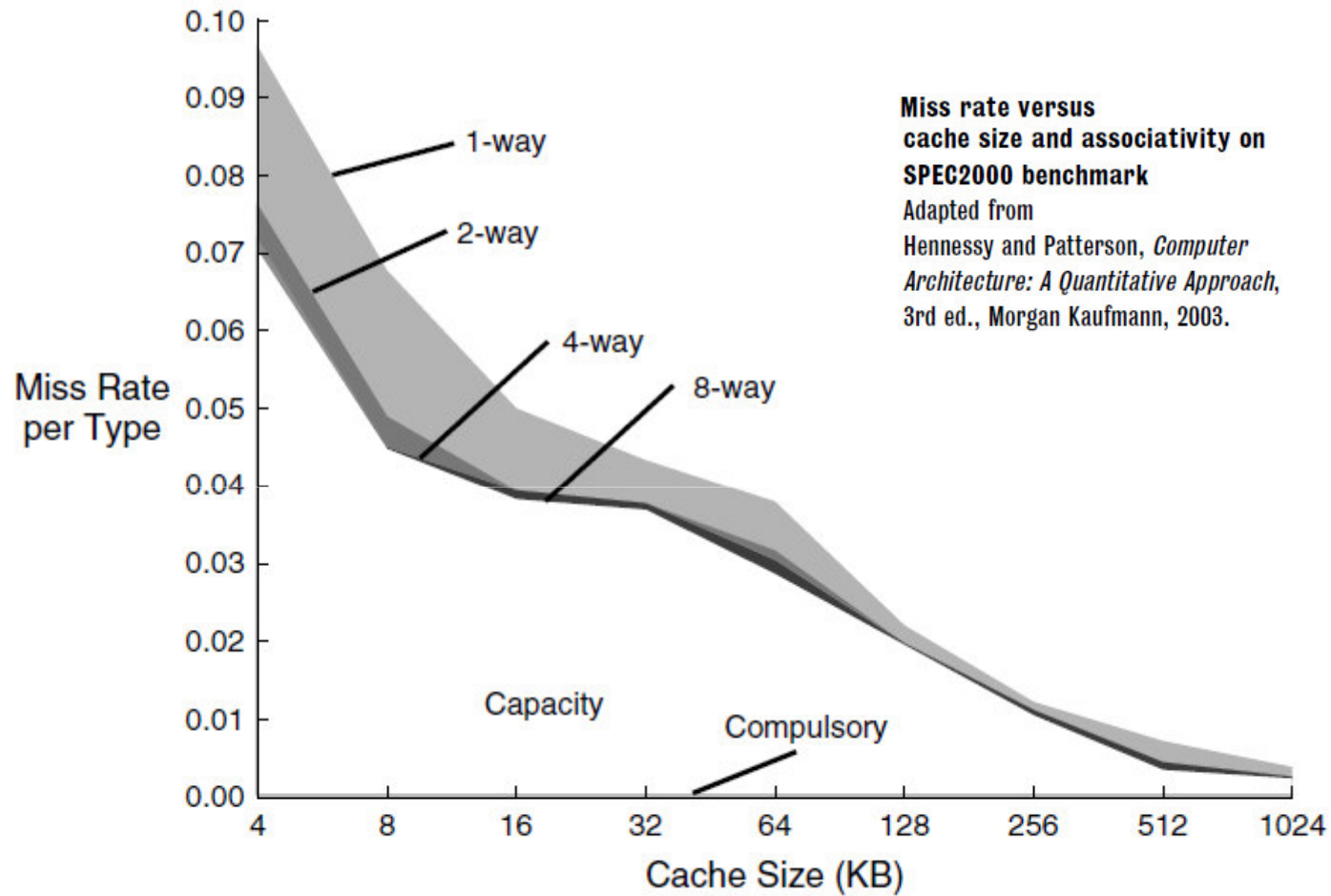
- Takováto cache je ale velmi drahá.
- Proto existují a používají se cache
 - Přímo mapované,
 - S omezeným stupněm asociativity.

Terminologie kolem skryté paměti II.

- **Hit Rate** - podíl počtu paměťových přístupů, které byly úspěšné (nalezla své údaje) při přístupu do té-které úrovně paměťové hierarchie.
- **Miss Rate** – podobně pro neúspěšný přístup.
- **Miss Penalty** – čas potřebný pro načtení bloku (údajů) z paměti nižší hierarchické úrovně.
- **Average Memory Access Time (AMAT)**
$$\text{AMAT} = \text{HitTime} + \text{MissRate} \times \text{MissPenalty}$$

MissPenalty – může být vypočtena rekurentně

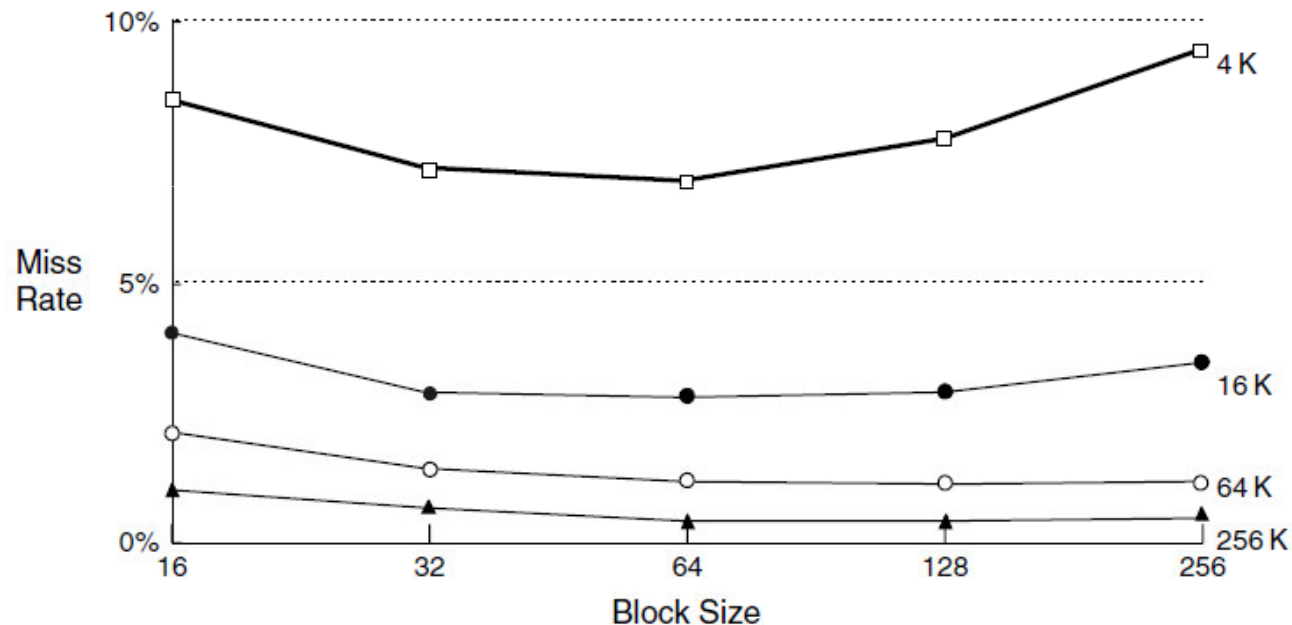
Porovnání



- Pamatujte: 1. miss rate není vlastností cache!
2. miss rate není vlastností programu!

Co přináší prostorová lokalita?

Miss rate můžeme redukovat zvýšením velikosti bloku – co znamená využití principu prostorové lokality. Na druhou stranu, zvětšování velikosti bloku při dané velikosti cache rovněž znamená snižování počtu setů – to se projeví nárůstem konfliktů (nárůstem miss rate)...



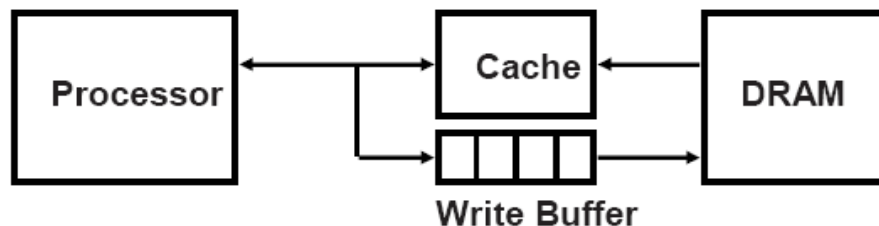
Miss rate versus block size and cache size on SPEC92 benchmark Adapted from Hennessy and Patterson, *Computer Architecture: A Quantitative Approach*, 3rd ed., Morgan Kaufmann, 2003.

Řešení situace **Cache Miss**, data v cache nejsou

- Data se nejprve musí z hlavní paměti přečíst. Jenže: co když je cache plná?
- Strategie uvolňování bloků/řádek cache
- **Náhodná** (Random) – vybere se libovolný blok. Snadné, ale hloupé.
- **LRU** (Least Recently Used) musíme znát informace o posledním použití tohoto bloku (jedná se o celé číslo).
- **LFU** (Least Frequently Used), ke každému bloku si pamatujeme informace o tom, jak často byl blok požadován.
- **ARC** (Adaptive Replacement Cache), ve které se vhodným způsobem kombinuje strategie LRU a LFU.
- **Write-back**. Zároveň musíme obsah uvolňovaných řádek cache do hlavní paměti zapsat (D bity označené řádky). Zajištěno automaticky.

Řešení situace **Zápis dat** procesorem do paměti

- Na cestě je i cache!
- **Konzistence dat** – samozřejmý požadavek na shodu obsahu stejných adres na různých médiích.
- **Write through** – současně se zápisem do cache se data zapíše do zápisové fronty a pak asynchronně do paměti.
- **Write back** – data se do cache zapíše s poznámkou Dirty (D bit Inf pole). Ke skutečnému zápisu dat do hlavní paměti dojde až v okamžiku případného rušení příslušného řádku cache, kdy hrozí jejich ztráta.
- **Dirty bit** – rozšiřující pole v obsahu paměti. Indikuje, že v cache (cache) je **jiná hodnota**, než v paměti hlavní.



V	Další bity, např. D	Tag	Data
---	---------------------	-----	------

Trend - Víceúrovňové SP

- Primární SP je bezprostředně připojena k procesoru
 - Rychlá, malá. Nejdůležitější: minimální Hit Time
- L2 SP ošetřuje výpadky primární SP
 - Větší, pomalejší, ale stále rychlejší než hlavní paměť.
Nejdůležitější: low Miss Rate
- Hlavní paměť ošetřuje výpadky L2
- Současné nejvýkonnější systémy mají i L3

	Typicky pro L1	Typicky pro L2
Počet bloků	250-2000	15 000-250 000
KB	16-64	2 000-3 000
Velikost bloku v B	16-64	64-128
Miss penalty (v hod)	10-25	100-1 000
Miss rates	2-5%	0,1-2%

Pochopili jste tuto přednášku?

- Pokud ano, tak již si uvědomujete, že využití 2 principů (principy časové a prostorové lokality) může vést k významnému urychlení Vašeho programu, a to efektivním využitím cache...!!!
- Existují HW a SW (kompilátorem) techniky, které na základě těchto principů optimalizují práci z cache. HW techniky z pohledu programátora ovlivnit nemůžete. U kompilátoru můžete nastavit stupeň optimalizace...
(Rozbor HW technik je mimo rozsah APO, patří do A4M36PAP.)
- Nicméně, i sebelepší kompilátor pouze kompiluje co napsal programátor. Výběr algoritmů, uložení datových struktur v paměti a manipulace s nimi – to vše je určeno programátorem. Proto stále je v rukou programátora „nejvíc“ práce a od něj do značné míry závisí jak bude program „rychlý“.

Pochopili jste tuto přednášku?

- Instrukční cache – pokročilé
 - Vhodným uspořádáním kódu, příp. přeuspořádáním funkcí v paměti
 - Profilace
- Datová cache – snadné
 - Vhodným uspořádáním dat – data, která plánujeme používat sekvenčně, řadit sekvenčně v paměti, apod.
 - Sloučení polí nebo souvisejících datových struktur
 - Práce po blocích dat – co nejdřív používat již použité
 - iterace ve vnořených cyklech – viz úvodní příklad – s cílem procházet paměť sekvenčně a ne po skocích
 - sloučení dvou smyček do jedné – Loop fusion
 - atd.

Pochopili jste tuto přednášku?

- Prostorová lokalita – konflikty v cache:

```
/* Před optimalizací */
```

```
int values[SIZE];
```

```
int keys[SIZE];
```

```
int scores[SIZE];
```

```
/* Po optimalizaci */
```

```
struct item{
```

```
    int value;
```

```
    int key;
```

```
    int score;
```

```
};
```

```
struct item records[SIZE];
```

Předpokládejme 2-cestně
asociativní cache...

```
for(i=0; i<SIZE; i++)  
    for(j=0; j<SIZE; j++)  
        ...
```

Pochopili jste tuto přednášku?

- Časová lokalita:

```
/* Před optimalizací */
```

```
for (i = 0; i < SIZE; i++)  
    for (j = 0; j < SIZE; j++)  
        a[i][j] = b[i][j] * c[i][j];  
for (i = 0; i < SIZE; i++)  
    for (j = 0; j < SIZE; j++)  
        d[i][j] = a[i][j] - c[i][j];
```

```
/* Po optimalizaci */
```

```
for (i = 0; i < SIZE; i++)  
    for (j = 0; j < SIZE; j++)  
        { a[i][j] = b[i][j] * c[i][j];  
          d[i][j] = a[i][j] - c[i][j]; }
```

Nejedná se jenom o úsporu instrukcí, ale také efektivněji používáme cache...

Pochopili jste tuto přednášku?

- Dalším příkladem je násobení matic

```
for(i=0; i < N; i++)  
  for(j=0; j < N; j++) {  
    tmp = 0;  
    for (k=0; k < N; k++)  
      tmp += y[i][k]*z[k][j];  
    x[i][j] = tmp;  
  }
```

Pomůže nám nějak když
prohodíme tyto dva řádky?
Bude program ekvivalentní?

(Viz úvodní příklad...)

Pochopili jste tuto přednášku?

- Dalším příkladem je násobení matic

Lépe je však použít tzv. blokové násobení.

Idea: Rozdělme výpočet na submatice $B \times B$, které se vejdou do cache.. => eliminace „capacity misses“

```
for (jj = 0; jj < N; jj = jj+B)
  for (kk = 0; kk < N; kk = kk+B)
    for (i = 0; i < N; i++)
      for (j = jj; j < min(jj+B-1,N); j++) {
        tmp = 0;
        for (k = kk; k < min(kk+B-1,N); k++)
          tmp += y[i][k]*z[k][j];
        x[i][j] = x[i][j] + tmp;
      }
```

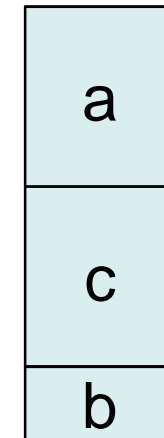
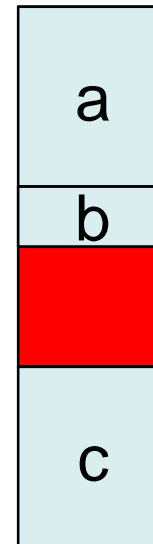
Ke čtení: <http://suif.stanford.edu/papers/lam-aspl91.pdf>

Pochopili jste tuto přednášku?

- Neplýtvejme pamětí – použijeme minimální množství paměti
- Spatřujete rozdíl v těchto deklaracích?

- **/* Před optimalizací */**

```
int a=0;  
char b='a';  
int c=1;
```



- **/* Po optimalizaci */**

```
int a=0;  
int c=1;  
char b='a';
```

Pochopili jste tuto přednášku?

- **Data, ke kterým přistupujete ve stejnou dobu (krátce za sebou) uložte vedle sebe (seskupte).**
- **Data, ke kterým přistupujete často uložte vedle sebe (seskupte).**
- Někdy se je potřeba rovněž zamyslet nad zarovnáním dat v paměti – buď přímo v assembleru nebo v jazyce C – zkontrolujte si jestli Váš kompilátor zarovnává double na 8-byte hranici, pokud ne:
 - alokujte si kolik potřebujete + 4B (nebo i víc – dle zarovnání)
 - pomocí ANDu získajte zarovnanou adresu pro svá data, příklad:

```
double a[5];  
double *p, *newp;  
p = (double*)malloc ((sizeof(double)*5)+4);  
newp = (p+4) & (-7);
```


Pochopili jste tuto přednášku?

- Hledání prvočísel - Eratostenovo sito:

/*Před optimalizací*/

```
boolean array[max];
for(i=2;i<max;i++) {
    array = 1;
}
for(i=2;i<max;i++)
    if(array[i])
        for(j=2;j<max;j+=i)
            array[j] = 0; /*přenos z paměti do cache a zápis 0*/
```

Přenos nastává pouze při
cache miss

Pochopili jste tuto přednášku?

- Hledání prvočísel - Eratostenovo sito:

```
/*Po optimalizaci*/
```

```
boolean array[max];
```

```
for(i=2;i<max;i++) {
```

```
    array = 1;
```

```
}
```

```
for(i=2;i<max;i++)
```

```
    if(array[i])
```

```
        for(j=2;j<max;j+=i)
```

```
            if(array[j]!=0) /*přenos z paměti do cache a čtení*/
```

```
                array[j] = 0; /*zápis 0 pouze někdy*/
```

- Redukujte neúčinné zápisy (redukce zápisů do paměti – dirty cache lines musejí být vždy zapsány před odstraněním z cache)