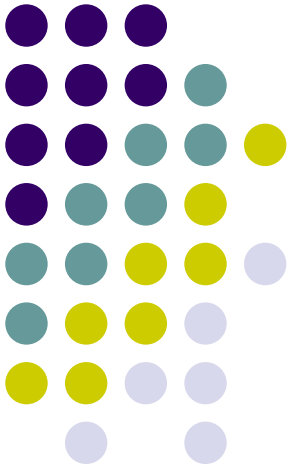
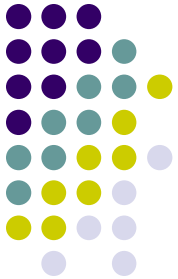
Cache paměti

Hardware



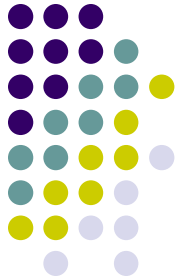
Cache paměti



- **SRAM – statická paměť** – nejrychlejší ze všech technologií.
- Kdyby hlavní operační paměť počítače byla SRAM tak by moderní počítače měli o dost vyšší výkon a nepotřebovali bychom žádné cache paměti
- Bohužel to není možné, protože cena rychlých statických pamětí SRAM je extrémně vysoká (až 10 milionů kč/GB)
- Tyto paměti neumíme vyrábět ve velkých kapacitách a levně, proto tento typ paměti nemůže být použit jako velká operační paměť. Taková paměť by pak tvořila 99 procent ceny počítače

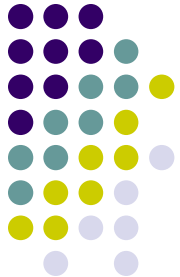
- Jako operační paměť počítače používáme DRAM
- **DRAM – dynamická paměť** – ve srovnání se všemi ostatními technologiemi pamětí ani levná ani drahá, ani rychlá ani pomalá… takový kompromis

Cache



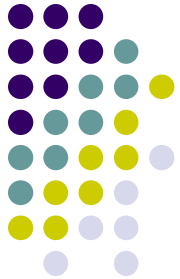
- Rychlosti procesoru se počínaje typem i80386 dále zvyšují mnohem rychleji než rychlost pamětí
- Procesor by uměl vykonat mnoho instrukcí pro čtení nebo zápis do paměti, ale paměť nestíhá a trvá jí příliš dlouho než vybaví nebo zapíše data
- Zavádění CACHE pamětí řeší nesoulad mezi vysokou rychlostí procesoru a pomalou paměti
- **Vyrovnávací paměť - cache** je velmi rychlá paměť malé kapacity, sloužící ke zrychlení přístupu k nejčastěji používaným datům
- Paměti Cache se zásadně vyrábějí jako **statické** (SRAM)
- Paměť statická je **rychlejší** a **nevyžaduje refresh**, ale jedna paměťová buňka má větší rozměry a tím pádem paměť má nižší kapacitu než paměti dynamické a je velmi drahá

Cache



- Účelem cache je urychlit přístup k **často používaným datům** v pomalé velké operační paměti jejich překopírováním do rychlejší cache paměti
- Cache paměť je mnohem rychlejší než velká operační paměť
- Do malé cache se ale nemůže vejít celý obsah velké operační paměti a je v ní tedy uložen pouhý zlomek dat
- Nejlépe by v cache měli být uložena data, ke kterým se přistupuje nejčastěji – musí tedy existovat mechanismus výběru adres, které budou zkopírovány v cache (jak dopředu poznám, co budu potřebovat často?)
- Dále musí také existovat nějaký mechanismus, který při čtení umožní procesoru **okamžitě** zjistit, zda adresa, ke které se chce přistupovat je dostupná v cache či nikoliv
- Mikroprocesor musí být schopen ihned zjistit, jestli data z adresy, se kterou chce pracovat, jsou zkopírovaná v rychlé cache – kdyby to nezjistil ihned a musel je v cache nějakou dobu hledat, tak je to celé k ničemu a nic se tím neurychlí

Cache L1, L2, L3



- **Externí cache** paměť je samostatný chip, který je umístěn mezi pomalejší operační pamětí a rychlým procesorem
- První **externí** cache paměti se objevují právě u počítačů s procesorem 80386

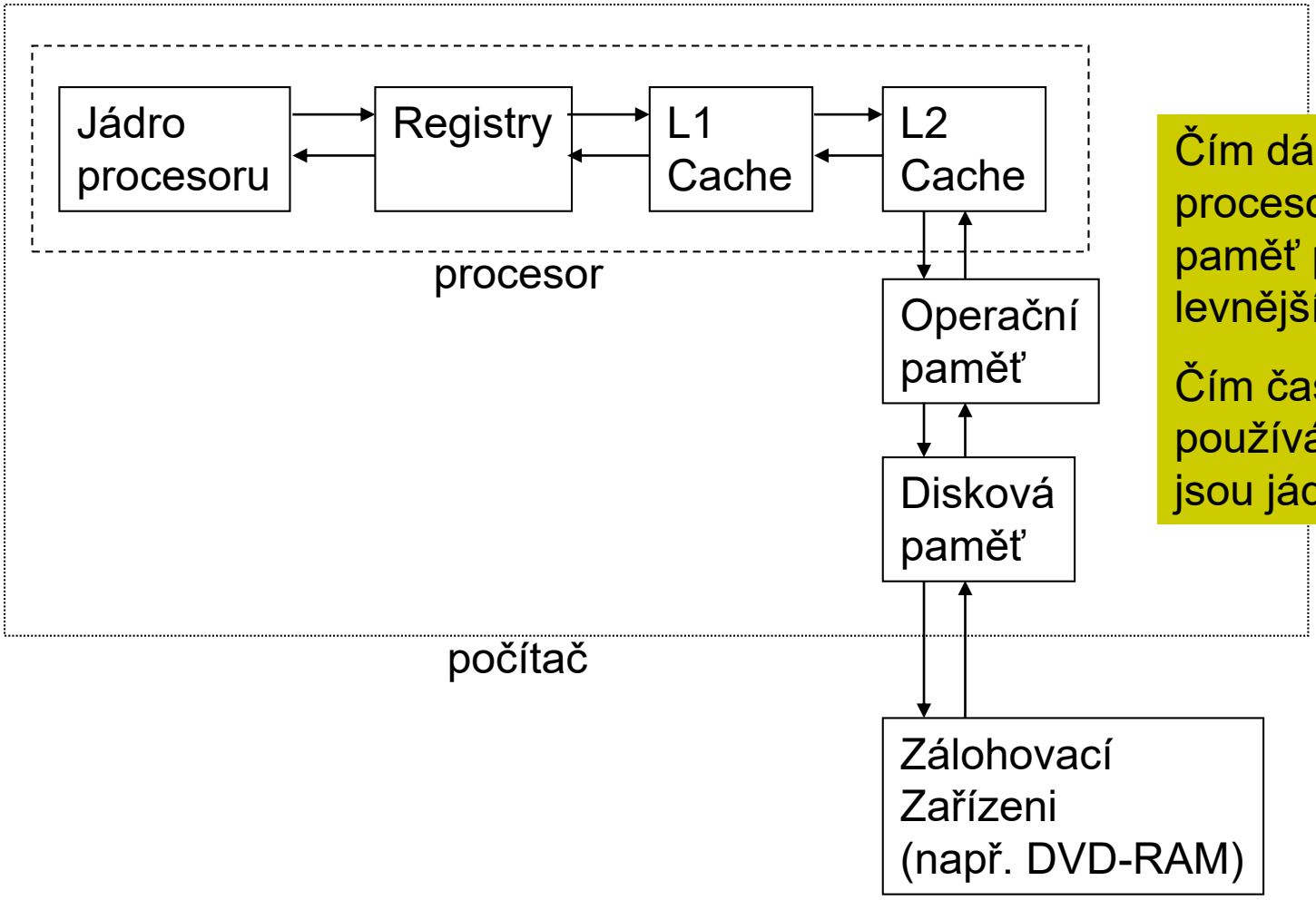
- Později dochází k integraci cache a mikroprocesoru v jediném pouzdře (**Interní cache** paměť se objevuje poprvé u procesoru 80486)

- Vyrovnávací paměť modernějších procesorů může být **dvojstupňová**.
- **L1 (Level 1 cache)** je velmi rychlá, ale extrémně drahá, protože je vyrobena nejlepší možnou technologií. Proto je také její kapacita maličká – pár kB. Jsou v ní uložena jen ta opravdu nejnejnejčastěji používaná data.

- **L2 (Second level cache)** je podstatně levnější, protože je vyrobena méně nákladnou technologií (stále jde o SRAM, ale pomalejší SRAM – ne všechny statické paměti jsou stejně rychlé). Z tohoto důvodu je také pomalejší, ale její kapacita může být několikrát větší, než kapacita L1 cache.

- Nejmodernější vícejádrové mikroprocesory používají třístupňovou cache – každé jádro má svou superrychlou ale velmi malou **L1 cache** a pomalejší (ale stále velmi rychlou v porovnání s DRAM pamětí) **L2 cache** s větší kapacitou (je levnější, tak může být větší)
- Všechna jádra vícejádrového mikroprocesoru pak sdílí dohromady velikou (několik MB) **L3 cache** (pomalejší než L1 a L2, ale levnější, takže ještě větší než L2)

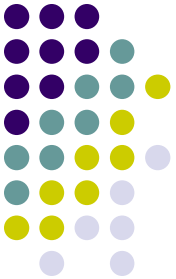
Hierarchie paměti



Čím dále od jádra procesoru, tím je paměť pomalejší, ale levnější (cena/kapacita)

Čím častěji jsou data používána, tím blíže jsou jádru procesoru

Cache – uvolňování místa

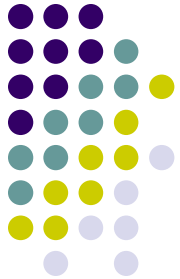


- Obsah paměti cache je mnohem menší než obsah operační paměti a tudíž se do ní nemůže vejít vše
- Cache se rychle zaplní nejčastěji používanými daty
- Tato data ale nezůstanou v cache navždy – občas je třeba nahradit je jinými častěji používanými daty
- Do cache se neustále ukládají nová, právě teď často používaná data, která nahrazují starší již nepoužívaná
- Pokud dojde k zaplnění cache paměti a je potřeba do ní uložit nějaká nová data, pro které tu už ale není místo, je nutné, aby některý ze starších záznamů cache paměť opustil a uvolnil tak místo
- *Představte si to, jako kdyby ve vašem mobilním telefonu bylo možné uložit v telefonním seznamu kontakty jen na 10 osob. Takový seznam se rychle zaplní, takže pokud jste právě poznali nějakou novou osobu, se kterou si budete teď často telefonovat, musíte smazat nějaké staré telefonní číslo, na které už moc často nevoláte, abyste si na uvolněné místo mohli uložit nový kontakt.*

Cache - uvolňování místa

- Jak vybrat „oběť“, která se z cache vyhodí, aby se v ní uvolnilo místo pro nová data?
- Tento problém řeší některá ze strategií náhrady:
  - **LRU** (Least Recently Used) používá se algoritmus, který vyřadí **nejdéle nepoužívaný** blok.
  - Pro každý záznam je potřeba mít uložené „časové razítko“, kdy byl naposledy využit.
  - Tuto strategii dnes používá naprostá většina cache pamětí.
  - **MFU** (Most Frequently Used) - ponechávají se často používané položky a položka **nejméně používaná** se smaže (dobré uvědomit si rozdíl mezi nejméně používaný a nejdéle nepoužívaný).
  - Pro každý záznam je potřeba mít počítadlo, kolikrát již byl použit.
  - **RANDOM** - náhodný výběr oběti – primitivní, ale funguje překvapivě dobře
  - **FIFO** - odebraná je ta položka, která je ve FIFO nejdéle (Cache pak vlastně funguje jako „průtoková fronta“)

Zápis do cache



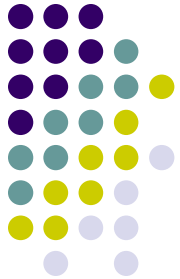
- Čtení z paměti cache je bezproblémové, ale se zápisem do cache vznikají problémy…
- Cachovaná data jsou vlastně uložena na dvou místech – originál v paměti a kopie pro rychlejší přístup v cache
- Nově zapsaný nebo změněný cachovaný bajt by měl být zapsán jak v paměti cache tak v operační paměti?

- **Write-Through:**
- (zápis skrz cache, přímý zápis) je jstarší a pomalejší způsob.
- Data ukládaná do cache zapisuje současně i do pomalé operační paměti

Write-Back:

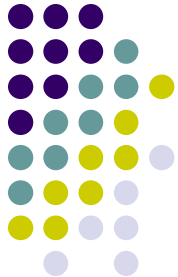
- (opožděný zápis) data jsou zapisována do operační paměti až ve chvíli, kdy je to třeba, a nikoliv okamžitě při jejich změně.
- Data jsou zapsána pouze do cache a teprve při odstranění z cache (např. LRU strategií) jsou provedené změny zapsány do operační paměti. Než se data dostanou do operační paměti, mohou několikrát změnit svoji hodnotu – při tom se tedy mění pouze hodnota cache a stav původního bajtu v operační paměti je neaktuální (to nás netrápí, aktuální se hledají nejdříve v cache). Tento způsob práce cache paměti vykazuje oproti předešlému způsobu vyšší výkon.

Princip fungování cache



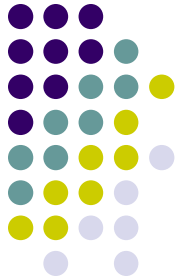
- Cache paměti bývají organizovány jako tzv. **asociativní paměti** (vyhledává se podle obsahu a ne podle adresy)
- K datům v paměti cache se nepřistupuje přes **adresu**, jak jsme zvyklý u běžné paměti
- Asociativní paměti jsou tvořeny tabulkou, která obsahuje vždy sloupec, v němž jsou umístěny tzv. **klíče**, podle kterých se v asociativní paměti vyhledává
- Dále jsou v tabulce umístěna **data**, která paměť uchovává, a také další informace nutné k zajištění správné funkce paměti
- Těmi „dalšími informacemi“ mohou být např. tyto
  - informace o platnosti (neplatnosti) uložených dat
  - informace pro realizaci LRU algoritmu (čas posledního využití dat)
  - informace protokolu MESI (Modified Exclusive Shared Invalid), který zajišťuje synchronizaci dat v cache pamětech v případě, že cache pamětí je v počítači více
- Každý záznam v paměti cache se tedy skládá z **klíče**, **vlastních dat** a dalších několika **řídících bitů**

Princip fungování cache



- Při přístupu do cache je nutné zadat **adresu**, kterou požadujeme nalézt – tato adresa buď celá nebo několik jejich bitů bude fungovat jako tzv. **vstupní klíč**
- Tento **vstupní klíč** se následně **porovnává** s klíči v cache paměti
- Pokud je v paměti nalezena položka, jejíž **klíč** se shoduje se **vstupním klíčem**, jsou v paměti cache přítomna požadovaná data
- Lze si to představit zhruba tak, že v paměti dojde k vyhledání dat na základě **shody klíčů**, přičemž tím klíčem je část nebo celá adresa, na které mají data ležet v operační paměti
- Dochází tedy k hledání, zda v paměti cache je přítomen obsah požadované adresy (zda je tato adresa cachovaná)
- Toto hledání není sekvenční, ale paralelní, proběhne okamžitě a naráz v celé cache – každý „řádek“ paměti obsahuje **komparátory** pro okamžité porovnání **klíče**

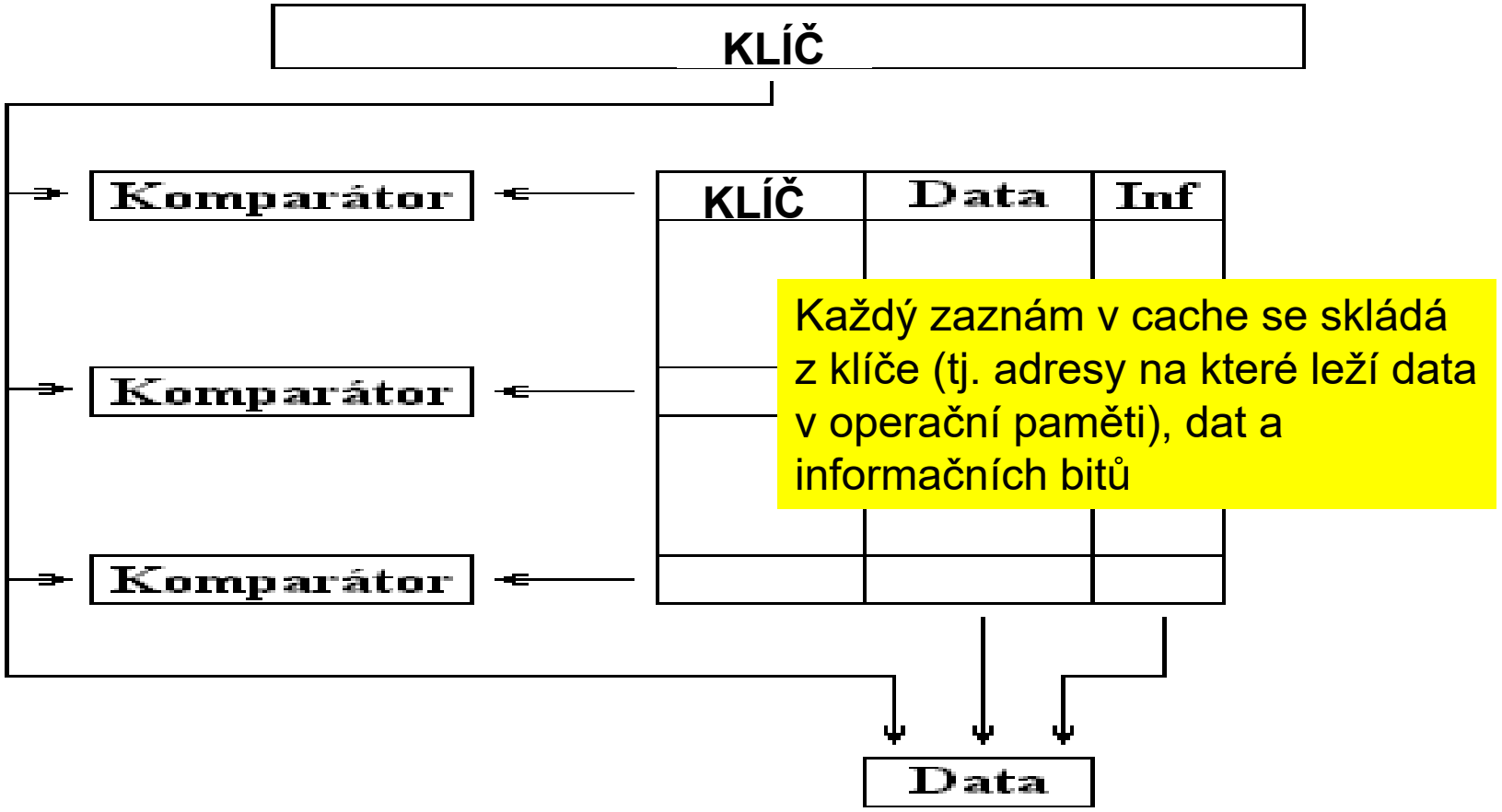
Plně asociativní cache



- U **plně asociativní** cache paměti je **celá** adresa, ze které se budou číst data (popř. na kterou se budou data zapisovat), uložena jako **klíč**
- Jak se tedy hledá v cache?
- Hledaná adresa = **vstupní klíč**
- Tento **vstupní klíč** je přiveden na vstup **všech komparátorů** společně s klíčem v daném řádku tabulky
- Pokud některý z klíčů v tabulce **je shodný** se zadaným klíčem na vstupu, ohlásí odpovídající komparátor na daném řádku shodu a znamená to, že požadovaná informace **je v cache paměti přítomna** a je možné ji rychle použít
- Pokud všechny komparátory signalizují **neshodu**, je to známka toho, že požadovaná informace **v cache paměti není** a je nutné ji číst z pomalé operační paměti

Plně asociativní cache

Požadovaná adresa = vstupní klíč



Každý zaznám v cache se skládá z klíče (tj. adresy na které leží data v operační paměti), dat a informačních bitů

Plně asociativní cache

- Příklad obsahu primitivní cache s kapacitou 8B

Klíč	Data	platnost
12345h	A7h	1
2A4D1h	FFh	1
00000h	00h	0
FF2C5h	14h	1
145ADh	BCh	1
75683h	11h	1
A1122h	22h	1
71243h	5Ch	1

Tento záznam v paměti cache znamená, že je v ní z hlavní operační paměti zkopírován bajt z adresy 145ADh jehož hodnota je BCh