CC protokol je distribuovaný algoritmus implementovaný souborem kooperujících řadičů cache.

Pozn. TLB není koherentní

5. Koherence pamětí cache na systémech se sdílenou pamětí, protokol MSI.

Na systémech s více procesory/jádry, které sdílí paměť, nastává problém s aktualitou dat. Každé CPU má typicky svoji cache (v případě více jader má typicky vlastní L1 a L2 cache, sdílenou L3), do které si načítá hodnoty z hlavní paměti. Je potřeba zajistit, aby na jedné adrese při čtení se našla vždy stejná data, ať jsou tato data v jakékoliv cache nebo v hlavní paměti (tzv. koherentní kopie).

Koherence pamětí cache znamená, že pro danou adresu existuje jediná (koherentní) verze sdílených dat v jedné, několika nebo i ve všech pamětech cache. Na kopii v paměti nezáleží, ta může být zastaralá (neplatná). **Koherence** se tedy zabývá jednoznačností zápisů na 1 adresu. Oproti tomu **paměťová konzistence** udává, že paměťové operace nad více adresami musí být prováděny ve stejném pořadí a toto pořadí musí vidět všechna jádra stejně.

Protokoly pro zajištění koherence využívají následující mechanismy:

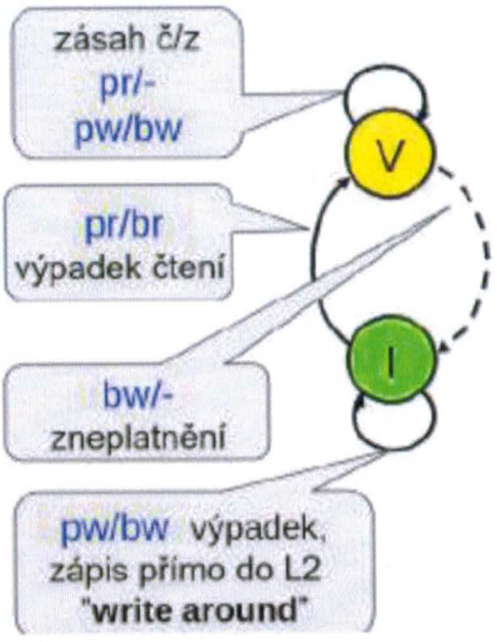
- **monitorování komunikace:** typicky na sběrnici – pokud nějaké jádro chce zapsat, musí to oznámit na sběrnici, na které ostatní poslouchají a mohou invalidovat přepsanou hodnotu. Sběrnice je úzkým hrdlem.
- **distribuované adresáře**: U hlavní paměti jsou tzv. adresáře, které uchovávají informaci, ve kterých cache jsou uložené které cache line. V případě zápisu pak adresář informuje vlastníky kopií přepisované hodnoty. Výhodou je, že umožňuje více paralelních zápisů (odstranění úzkého hrdla). Nevýhodou je, že to přidává paměťový overhead. Typicky se využívá na NUMA systémech.

Při práci s pamětí rozlišujeme několik "módů":

- Způsob aktualizace paměti:
  - write-through – okamžitý zápis z cache do hlavní paměti při změně
  - **write-back** – zápis do hlavní paměti, až když by hodnota měla být odstraněna z cache (opožděný zápis)
- Způsob zajištění koherence:
  - **write-invalidate** – při zápisu se data v cache zneplatní
  - write-update – při zápisu se v cache data aktualizují

Jednoduchým způsobem zajištění koherence na systémech se sběrnicí je právě monitorování s využitím okamžitého zápisu do paměti. V případě zápisu jádro vystaví na sběrnici signál bw (bus write) nebo brx (bus read exclusive), na základě kterého ostatní jádra invalidují zapisovanou hodnotu. Jedná se o tzv. protokol valid-invalid:

Koherence se týká jednoznačnosti zápisu na 1 adresu
Konzistence se týká pořadí přístupů na různé adresy



Aktualizace	Koherence
„write-through,	write-invalidate"
zapisuje slovo,	zneplatní blok

Legenda:
(vstup/výstup) řadiče cache
- pr, pw = proc. read, write
- br, bw = bus read, write
- blok v cache označen jedním bitem "in/valid"

použití: CC mezi privátními cache L1 a sdílenou L2 ve vícejádrových procesorech.
Nevýhoda: velké množství výpadků

Protokol MSI

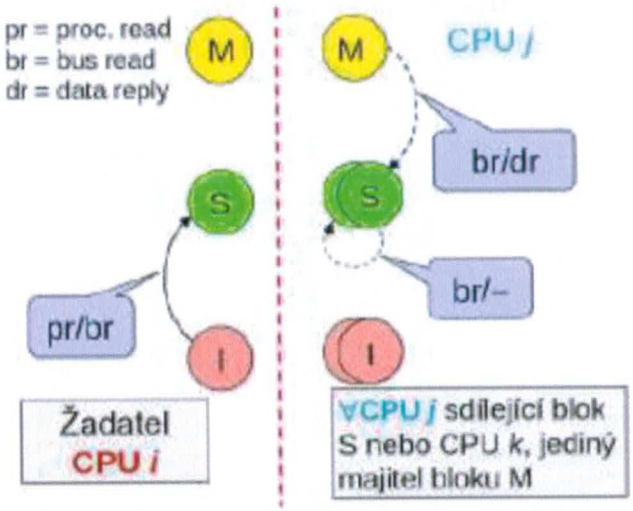
Protokol valid-invalid vede na velké množství výpadků a byl by výrazně neefektivní. Z tohoto důvodu zavádíme další stavy, které nám umožní udržet špinavou kopii v L1 cache bez okamžitého zápisu. Základem je třístavový protokol MSI:

- Modified – jen 1 cache má platnou špinavou kopii
- Shared – 1 nebo více pamětí cache má kopii shodnou s RAM (čistá kopie)
- Invalid – v této cache je neplatná/stará kopie

Blok S lze jen číst. Pokud bychom jej chtěli zapsat, přechází blok S do stavu M a ostatní kopie musí být zneplatněny. V následujících diagramech je vždy vlevo žadatel, vpravo ostatní bloky. Při žádání o zápis/čtení může nastat několik situací.

Výpadek při čtení (R-)

Jádro chce přečíst hodnotu, kterou nemá v cache nebo ji má cache, ale s příznakem I (invalid). Vystaví na sběrnici signál bus read, na jehož základě výstaví buď jiné jádro, nebo hlavní paměť hodnotu. Pokud je v jiném jádru hodnota označená jako Modified, je hodnota na sběrnici vystavena tímto jádrem a zároveň přechází do stavu Shared a je zapsána do paměti. Po vystavení hodnoty žadatel přechází ze stavu Invalid do stavu Shared.



pr → CacheCtrl *i* → R- :
1. žádost o sběrnici;
2. když přidělena: br;
3. když *victim* dirty: wb;
4. když data přijata: I → S;

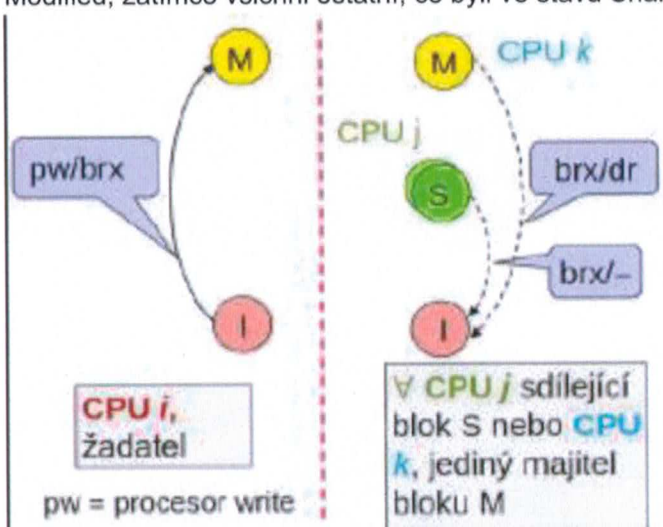
CacheCtrl *j*:
1. br: když stav = M: dr M;
2. M → S;
3. dirty bit → sběrnice

MemCtrl:
br: když dirty bit = 0: dr S;

Závěr: data dá na sběrnici paměť (S) nebo majitel (M). Pokud byl blok ve stavu (M), putují data i do paměti.

Výpadek při zápisu (W-)

Jádro chce zapsat hodnotu, kterou nemá v cache, nebo ji má, ale s příznakem Invalid. V tomto případě žadatel dá na sběrnici bus read exclusive. Data mu vystaví buď hlavní paměť, nebo vlastník` (pokud je v jiné cache line hodnota ve stavu Modified). Žadatel přechází do stavu Modified, zatímco všichni ostatní, co byli ve stavu Shared/Modified přechází do stavu Invalid:



pw → CacheCtrl *i* → W- :
1. žádost o sběrnici;
2. když přidělena: brx (bus read exclusive) na sběrnici;
3. když victim dirty: wb;
4. když data přijata: I → M;

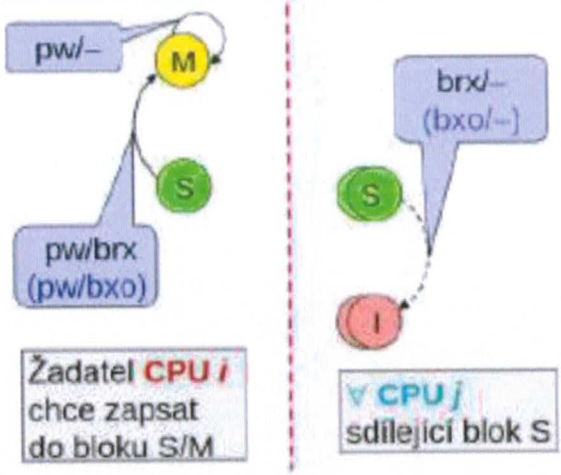
CacheCtrl *j*, *k*:
1. brx: když stav = M: dr M;
2. změna stavu S, nebo M → I;

MemCtrl:
brx: když dirty bit = 0:
dr S, S → I; kopie neplatná

Závěr: před zápisem se musí data zneplatnit u předchozího vlastníka (M) nebo všech vlastníků (S). Data dá na sběrnici paměť (S) nebo vlastník (M).

Zásah při zápisu (W+)

Jádro chce zapisovat hodnotu, kterou již v cache má (ve stavu Shared nebo Modified). Všechna ostatní jádra na základě signálu bus read exclusive přechází do stavu Invalid. Pokud na sběrnici máme dostupný pouze signal brx (a nemáme signal bus exclusive owner == bxo), pak na základě signálu brx sdílená paměť vystaví hodnotu na sběrnici, přestože ji nepotřebujeme.



pw → CacheCtrl *i* → W+:
1. když stav = M: **exit**;
2. žádost o sběrnici;
3. když přidělena: brx („bus read exclusive");
4. stav S → M;

CacheCtrl *j*:
brx: stav S → I

MemCtrl:
brx: když dirty bit = 0: dr S;
(zbytečné, žadatel má kopii S.)
Alternativa:
bxo: „bus exclusive owner" žádný přenos!

Závěr: před zápisem se musí data zneplatnit u všech vlastníků (S). Příkaz bxo místo brx vyžádá jen invalidaci bloku (S) u ostatních cache a blokuje načítání z paměti.

Příklad

	Akce	Sběrnice	Stav C1	Stav C2	Stav C3	Data dodá	Data přijme
0	-	-	M	I	I	-	-
1	P3 čte	br	S	I	S	C1	C3, SM
2	P3 píše	brx	I	I	M	SM	C3
3	P1 píše	brx	M	I	I	C3	C1
4	P2 čte	br	S	S	I	C1	C2, SM
5	P3 píše	brx	I	I	M	SM	C3

Vysvětlení:

1. P3 čte, jediný vlastníkem je C1, vystaví tedy data. Zároveň proběhne write-back do hlavní paměti
2. P3 píše, vystaví brx a tím se znevalidní kopie v C1. Pokud uvažujeme pouze signál brx, pak hlavní paměť na něj zareaguje a vystaví data. Alternativně může být využit i signál bxo, pak by nikdo data nepřijímal/nedodával
3. P1 píše, C3 předá svoji kopii C1. **POZOR: neprobíhá zápis do hlavní paměti**

4. P2 čte, jediná kopie je v C1, která ji dodá, a zároveň proběhne write-back do paměti
5. P3 píše, shared kopie se znevalidní. Jsou dvě kopie v systému, namísto rozhodování, jestli hodnotu vystaví C1 nebo C2, je hodnota vystavena z hlavní paměti.

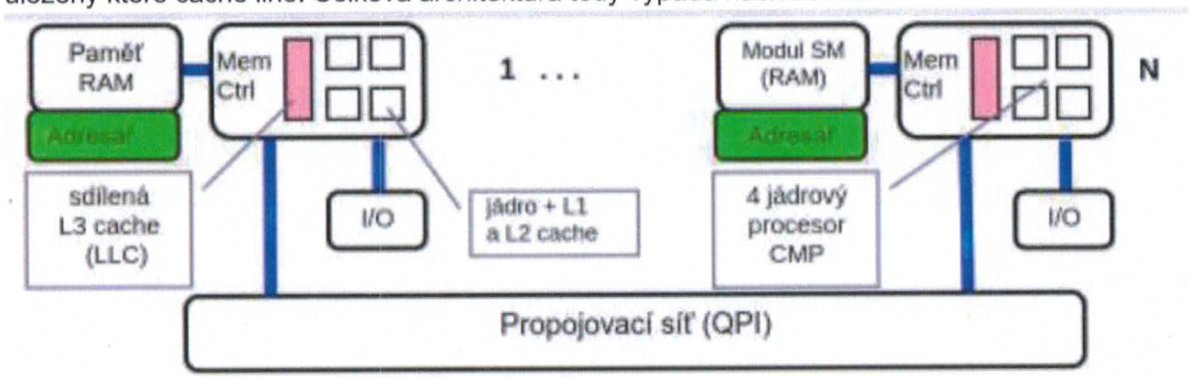
Rozšíření protokolu MSI

MSI je poměrně jednoduchý protokol, který provádí některé neefektivní operace. Například i v případě, že je v systému jediná kopie ve stavu S, v případě žádosti o čtení jiným jádrem dodá hodnotu hlavní paměť, což je pomalé. Z tohoto důvodu se zavedlo rozšíření MESI. Stav E značí, že existuje čistá kopie pouze v jediné cache. Dále pak byla zavedena rozšíření MOESI a MESIF, která poskytují další vylepšení.

AMD Intel

Koherence na NUMA systémech (distribuované adresáře)

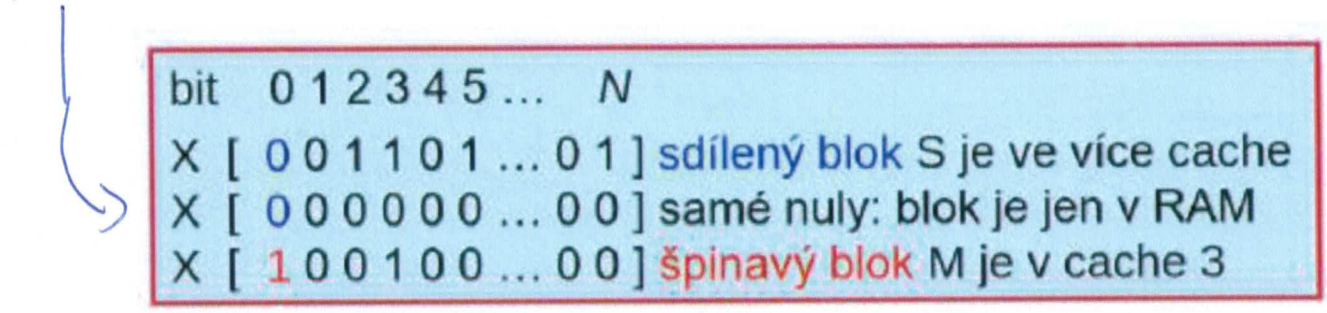
V systémech NUMA není jedna sdílená sběrnice – jednotlivé procesory jsou propojeny nějakou propojovací sítí (např. do kruhové topologie). Koherenci cache tedy nelze řešit naslouchánim na sběrnici, podobně jako tomu bylo u UMA systémů (např. v protokolu MSI). Proto se využívají distribuované adresáře (ty jsou uloženy v paměti), ve kterých je uloženo, v jakých cache jsou uloženy které cache line. Celková architektura tedy vypadá následovně:



Pokud tedy procesor chce pracovat se sdílenou pamětí, udělá žádost na svůj domovský adresář (označujeme Home) a ten se podívá do svého obsahu, kde jsou příslušné cache line uložené a kontaktuje jen ty relevantní cache, aby invalidovaly data. V rámci adresáře se nachází řadič distribuovaného adresáře (DirCtl), který řadí požadavky. Struktura adresáře je taková, že pro každou cache line (64B) máme jeden řádek v adresáři. První bit je validity bit značící čistost/špinavost. Zbývajících N bitů je bitová maska, která pro každou z N cache udává, zda se v ní daná cache line vyskytuje, nebo ne:

↑
adresář určuje pořadí záznamů, v případě sběrnice určuje pořadí arbitr sběrnice

pokračování MSI



Problémem zde je škálovatelnost. Pokud bychom měli například 256 cache, potřebujeme 257 bitů pro každou cache line, přičemž v cache line je 64B dat, tzn. režie je 257 / (64 * 8) = 50%. Proto se typicky omezuje např. počet sdílených kopií (tím omezíme délku bitové masky)
- nebo vázané adresáře

Pokud v textu najdete chybu, nebudete něčemu rozumět nebo budete mít dojem, že by bylo vhodné něco doplnit, kontaktujte na discordu uživatele Fifinas.

Problém Falešného sdílení

- různá data (prvky vektoru) modifikována společně dvěma vlákny by neměla být v jednom bloku cache
- při zápisem jedním jádrem bude kopie bloku v druhém jádru znehodnocena
- např. vektorový zápis, kdy každá položka je pro jiné jádro, ale jsou ve stejné cache line :/
  => řešením je zvýšení dimenze pole (1 prvek na celý blok, padding)