Paralelní architektury - úvod

Úvod do paralelních architektur

Příklady paralelních architektur

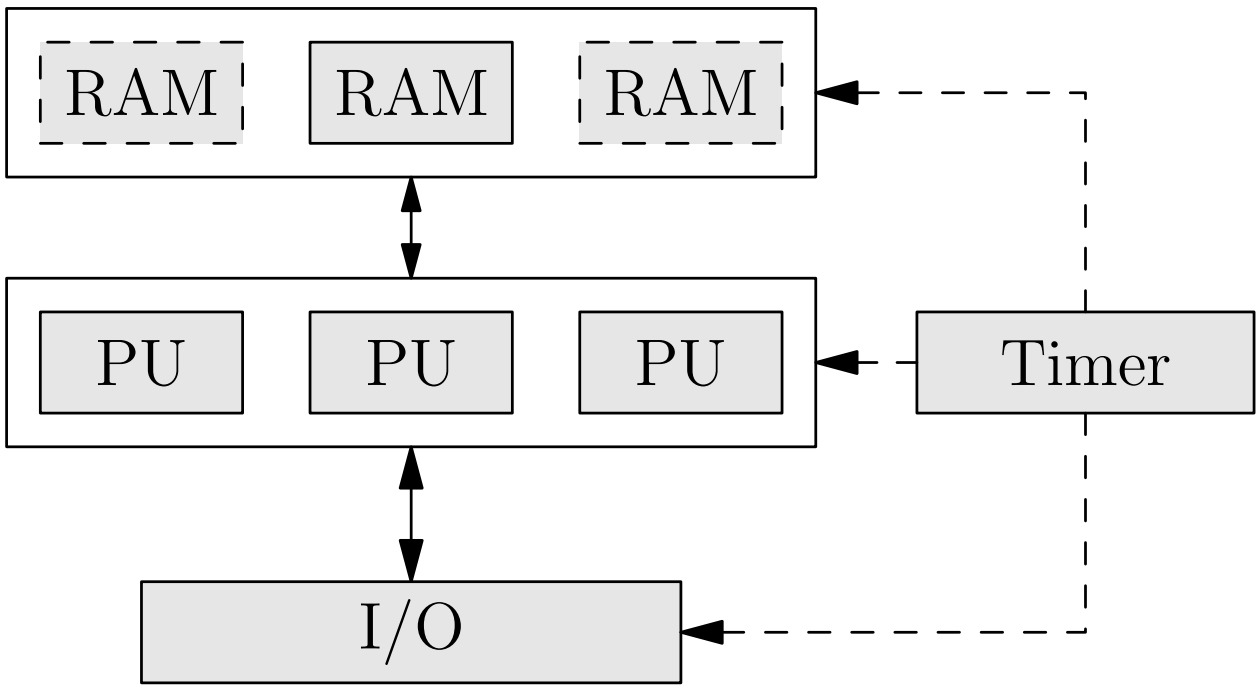
- Processor arrays
- Multiprocesory
  - Multiprocesory se sdílenou pamětí
  - Multiprocesory s distribuovanou pamětí
- Multipočítače
  - Nesymetrické multipočítače
  - Symetrické multipočítače
- Grid
- GPU

Dělení paralelních architektur

- Flynnova taxonomie
- Komunikační modely paralelních architektur

Paralelní architektura

Definice: **Paralelní architektura je taková, která obsahuje více jednotek pro zpracování dat (PU).**



Paralelní architektura II.

Příklad: Najděte nejlepší způsob, jak v pěti lidech setřídit balíček zamíchaných karet.

- nejprve je nutné karty rozdělit mezi zúčastněné.

Paralelní architektura II.

Příklad: Najděte nejlepší způsob, jak v pěti lidech setřídit balíček zamíchaných karet.

- nejprve je nutné karty rozdělit mezi zúčastněné.
- každý si třídí svůj balíček

Paralelní architektura II.

Příklad: Najděte nejlepší způsob, jak v pěti lidech setřídit balíček zamíchaných karet.

- nejprve je nutné karty rozdělit mezi zúčastněné.
- každý si třídí svůj balíček
- setříděné balíčky se složí do jednoho

Paralelní architektura II.

Příklad: Najděte nejlepší způsob, jak v pěti lidech setřídit balíček zamíchaných karet.

- nejprve je nutné karty rozdělit mezi zúčastněné.
- každý si třídí svůj balíček
- setříděné balíčky se složí do jednoho

Paralelní architektura II.

Příklad: Najděte nejlepší způsob, jak v pěti lidech setřídit balíček zamíchaných karet.

- nejprve je nutné karty rozdělit mezi zúčastněné.
- každý si třídí svůj balíček
- setříděné balíčky se složí do jednoho

Narozdíl od případu, kdy karty třídí jeden člověk, se zde nutně objevuje **komunikace**.

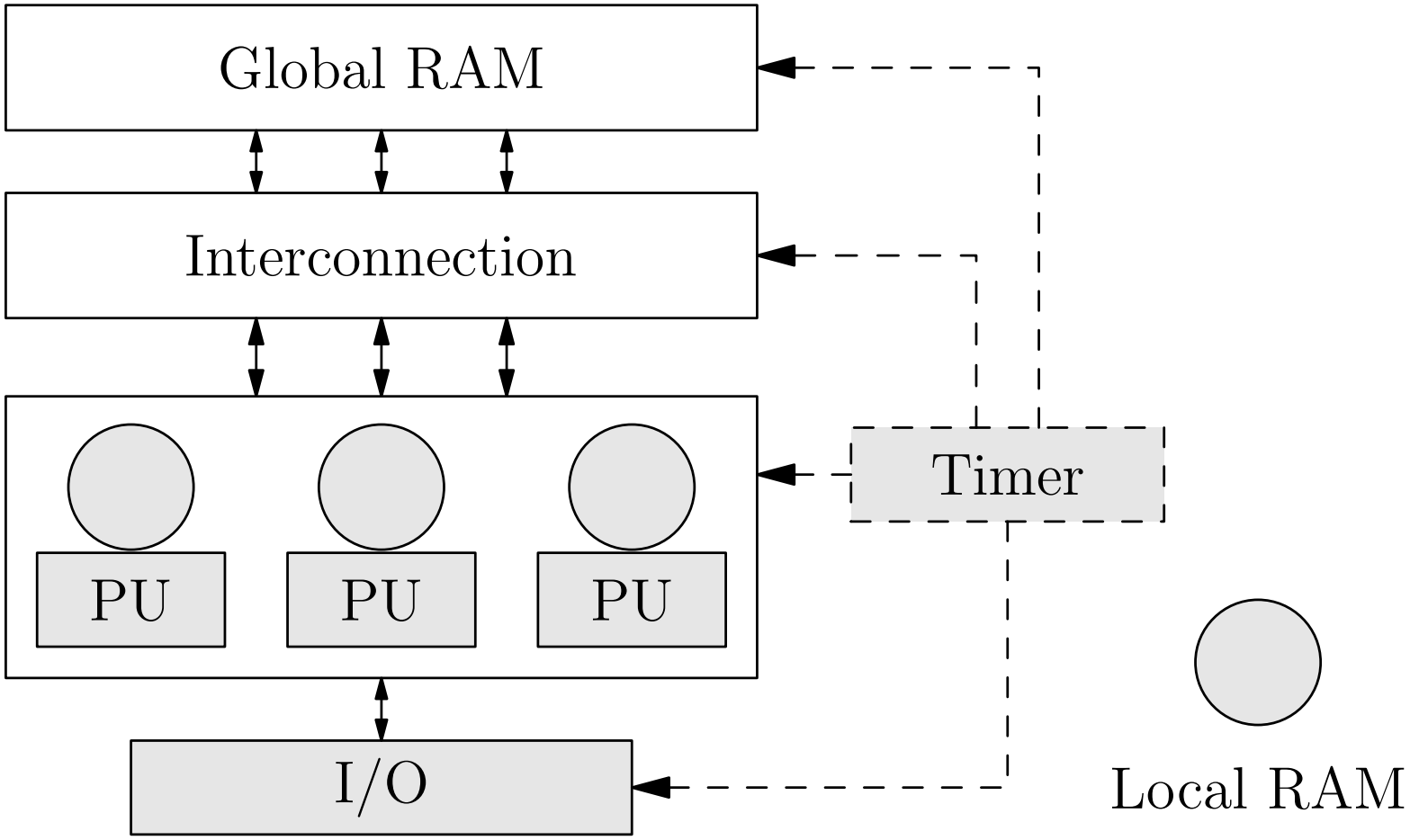
Paralelní architektura II.

Příklad: Najděte nejlepší způsob, jak v pěti lidech setřídit balíček zamíchaných karet.

- nejprve je nutné karty rozdělit mezi zúčastněné.
- každý si třídí svůj balíček
- setříděné balíčky se složí do jednoho

Narozdíl od případu, kdy karty třídí jeden člověk, se zde nutně objevuje **komunikace**. Ta bývá největším problémem paralelních algoritmů a architektur.

Paralelní architektura II.



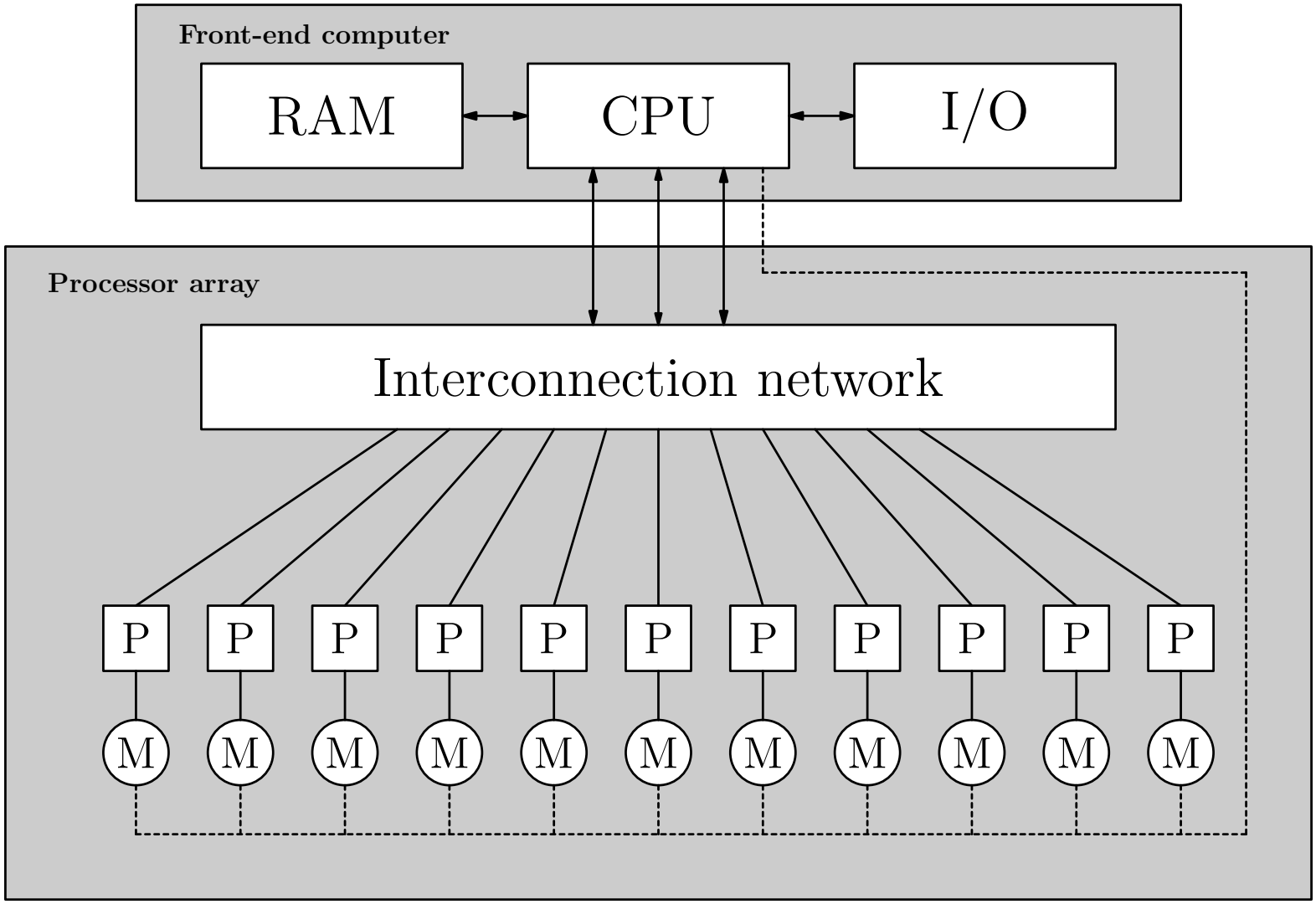
Paralelní architektura II.

- **Globální paměť** - zprostředkovává komunikaci mezi výpočetními jednotkami
- **Lokální paměť** - výpočetní jednotky ji používají ve chvíli, kdy provádějí výpočty nezávisle na ostatních
  - lokální paměť je přítomna ve všech paralelních architekturách
  - přístup do globální paměti je vždy pomalý
  - lokální paměť může být:
    - operační paměť výpočetního uzlu v případu klastru nebo gridu
    - cache paměť v případě více procesorových systémů
    - registr v případě vektorových architektur (MMX, GPU)
    - ...
- **Časovač** (Timer)
  - stejně jako u sekvenčních systémů i zda má význam synchronizace
  - některé architektury (klastr,grid) jsou synchronizovány pouze přístupem do globální paměti, časovač zde pak chybí

Vektorové počítače - vector computers

- mají podporu i pro vektorovou aritmetiku
  - např. umožňují v jednom kroku sečíst dva vektory
- patří sem jedny z prvních superpočítačů
  - Cray-1 - 1976 - http://en.wikipedia.org/wiki/Cray-1
  - Cyber-205
  - SSE rozšíření architektury x86

Procesorová pole - processor arrays I.



Procesorová pole - processor arrays II.

Tato architektura se skládá z:

- plnohodnotného počítače - front-end computer
- ... a několika jednoduchých, synchronizovaných výpočetních jednotek vybavených vlastní malou lokální pamětí.

Procesorová pole - processor arrays II.

Tato architektura se skládá z:

- plnohodnotného počítače - front-end computer
- ... a několika jednoduchých, synchronizovaných výpočetních jednotek vybavených vlastní malou lokální pamětí.

- front-end zpracovává kód uložený v operační paměti - RAM
- narazí-li na možnost paralelního výpočtu, pak
  - distribuuje potřebná data do lokalních pamětí výpočetních jednotek
    - CPU má přímý přístup do lokálních pamětí
  - výpočetním jednotkám pošle instrukce, které mají provést

Procesorová pole - processor arrays II.

Tato architektura se skládá z:

- plnohodnotného počítače - front-end computer
- ... a několika jednoduchých, synchronizovaných výpočetních jednotek vybavených vlastní malou lokální pamětí.

- front-end zpracovává kód uložený v operační paměti - RAM
- narazí-li na možnost paralelního výpočtu, pak
  - distribuuje potřebná data do lokalních pamětí výpočetních jednotek
    - CPU má přímý přístup do lokálních pamětí
  - výpočetním jednotkám pošle instrukce, které mají provést

Důvodem konstrukce procesorových polí byla výrazně nižší cena jednoduchých výpočetních jednotek oproti plnohodnotnému CPU.

Procesorová pole - processor arrays III.

Zpracování podmínek

- všechny jednotky provádějí ten samý kód
  - není možné, aby jedna polovina prováděla jiný výpočet, než druhá polovina
- je ale možné některé jednotky dočasně z výpočtu vyřadit - tzv. **maskování**
- díky tomu lze provádět kód s podmínkami, jejichž vyhodnocení je pro každou jednotku různé
- nejprve je zpracována jedna větev podmíněného výpočtu a jednotky, u nichž tato větev nemá být prováděna, jsou odstaveny
- potom se situace obrátí
- ve skutečnosti tak výpočet každé jednotky zabere talik času jako zpracování obou větví
- pokud ale všechny jednotky provádějí jen jednu větev, není nutné čekat na zpracování té druhé
- proto je vhodné, aby prováděný kód byl co nejvíce "homogenní"

Procesorová pole - processor arrays IV.

Paralelní redukce a procesorového pole

- systém je vybaven mechanismem pro slučování mezivýsledků
  - jde o tzv. **redukci**
  - např. při počítaní skalárního součinu každá jednotka provede vynásobení jedné složky vektoru
  - mechanismus pro redukci pak provede sečtení napočítaných hodnot
  - podporováno bývá více operací - sčítání, násobení, MAX, MIN, AND, OR atd.

Procesorová pole - processor arrays V.

Některé nevýhody procesorových polí

- tato architektura je vhodná jen pro některé typy úloh
  - numerická matematika, zpracování signálu, zpracování obrazových dat, vizualizace
- p.p. nejsou vhodná pro kód obsahující mnoho podmínek
- většinou jsou to jednouživatelské systémy
  - neumí rozdělit výpočetní jednotky mezi více uživatelů
- jsou špatně škálovatelná
  - vyžadují velmi rychlou komunikaci (IN) mezi CPU a výpočetními jednotkami, to je drahé
  - s malým počtem výpočetních jednotek by se "nezaplatily" náklady na rychlou IN
  - velký počet výpočetních jenotek by IN zahltil
- šlo o velmi specializovaná zařízení, proto byla velmi drahá
- s rozvojem VLSI klesly ceny plnohodnotných CPU a tedy i důvody pro p.p.

Procesorová pole - processor arrays V.

Některé nevýhody procesorových polí

- tato architektura je vhodná jen pro některé typy úloh
  - numerická matematika, zpracování signálu, zpracování obrazových dat, vizualizace
- p.p. nejsou vhodná pro kód obsahující mnoho podmínek
- většinou jsou to jednouživatelské systémy
  - neumí rozdělit výpočetní jednotky mezi více uživatelů
- jsou špatně škálovatelná
  - vyžadují velmi rychlou komunikaci (IN) mezi CPU a výpočetními jednotkami, to je drahé
  - s malým počtem výpočetních jednotek by se "nezaplatily" náklady na rychlou IN
  - velký počet výpočetních jenotek by IN zahltil
- šlo o velmi specializovaná zařízení, proto byla velmi drahá
- s rozvojem VLSI klesly ceny plnohodnotných CPU a tedy i důvody pro p.p.

Tyto důvody vedly k úpadku těchto architektur a jejich nahrazení multiprocesory.

Multiprocesorové architektury I.

Multiprocesor se skládá z

- několika plnohodnotných procesorů
- sdílené paměti
  - stejné adresy u dvou různých CPU ukazují na stejné místo v paměti

Multiprocesorové architektury I.

Multiprocesor se skládá z

- několika plnohodnotných procesorů
- sdílené paměti
  - stejné adresy u dvou různých CPU ukazují na stejné místo v paměti

Výhody oproti procesorovým polím

- lze je vyrábět z běžných a tedy i levných CPU
- mohou být sdíleny více uživately
- neztrácejí efektivitu při zpracování kódu s podmínkami

Multiprocesorové architektury I.

Multiprocesor se skládá z

- několika plnohodnotných procesorů
- sdílené paměti
  - stejné adresy u dvou různých CPU ukazují na stejné místo v paměti

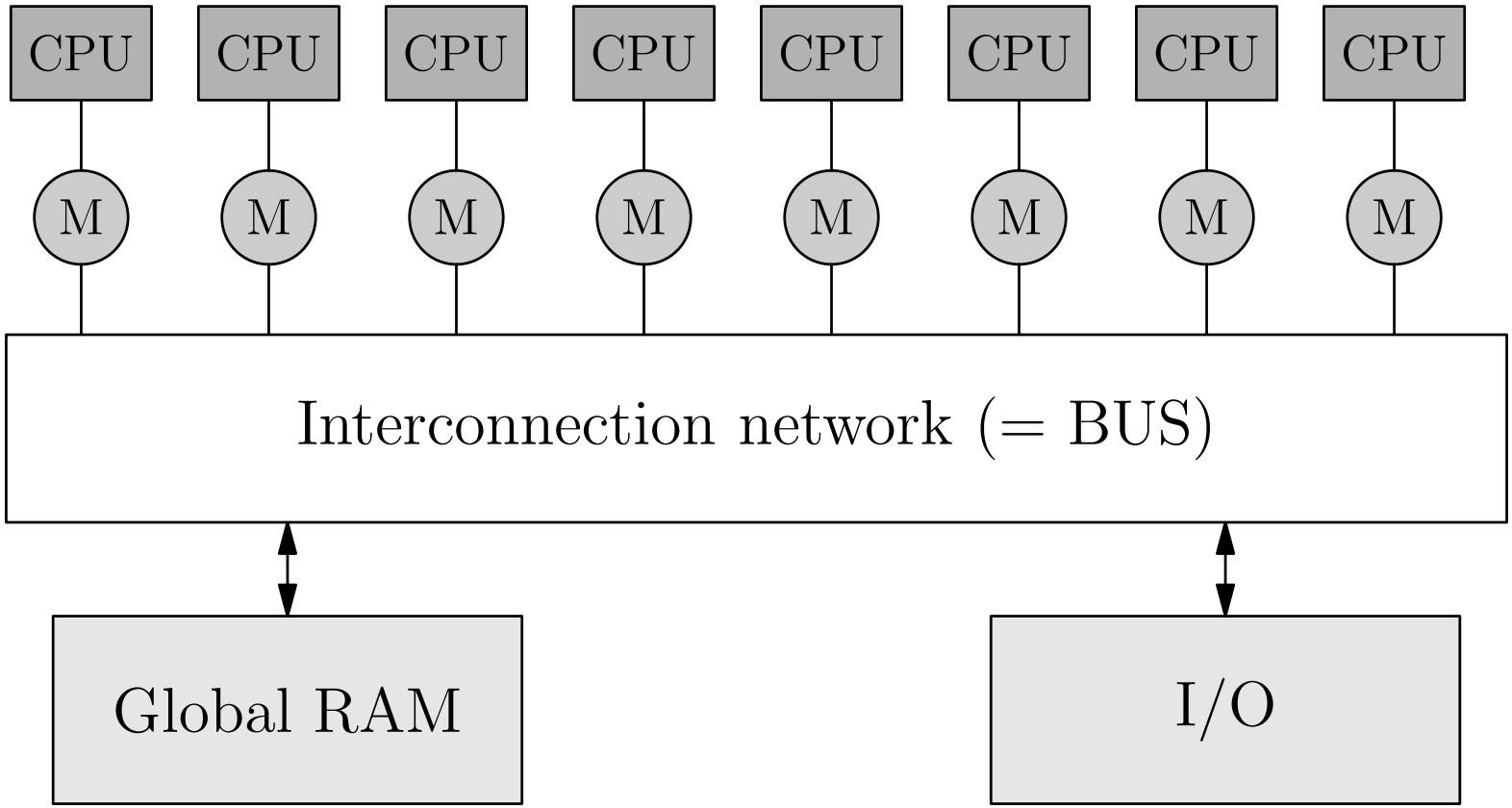
Výhody oproti procesorovým polím

- lze je vyrábět z běžných a tedy i levných CPU
- mohou být sdíleny více uživately
- neztrácejí efektivitu při zpracování kódu s podmínkami

Dělí se na

- systémy se sdílenou pamětí
- systémy s distribuovanou pamětí

Multiprocesorové systémy se sdílenou pamětí I.



Multiprocesorové systémy se sdílenou pamětí II.

- tyto systémy jaou odvozeny z jednoprocesorového systému pouhým přidáním dalších CPU propojených **sběrnicí** (BUS)
- všechny procesory jsou rovnocenné
  - odtud název - **symmetric multiprocessor - SMP**
- přístup do globální paměti je vždy stejně rychlý
  - odtud název - **uniform memory acces multiprocessor - UMA**

Multiprocesorové systémy se sdílenou pamětí II.

- tyto systémy jaou odvozeny z jednoprocesorového systému pouhým přidáním dalších CPU propojených **sběrnicí** (BUS)
- všechny procesory jsou rovnocenné
  - odtud název - **symmetric multiprocessor - SMP**
- přístup do globální paměti je vždy stejně rychlý
  - odtud název - **uniform memory acces multiprocessor - UMA**

Příkladem SMP jsou dnes běžné vícejádrové PC.

Multiprocesorové systémy se sdílenou pamětí III.

Přístup více CPU do paměti

- rozlišujeme dva typy proměnných
  - **soukromé** (private) - jsou přístupné jen jednomu procesoru
  - **sdílené** (shared) - může k nim přistupovat více procesorů

Multiprocesorové systémy se sdílenou pamětí III.

Přístup více CPU do paměti

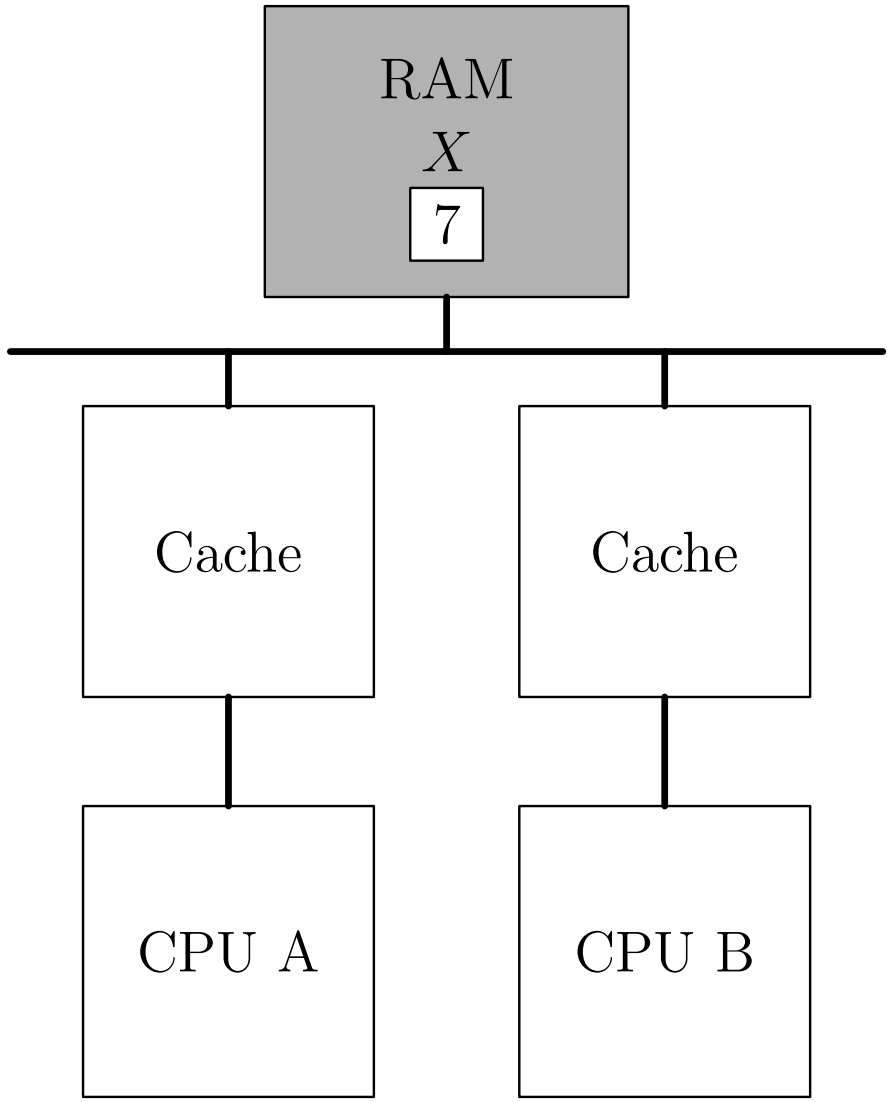
- rozlišujeme dva typy proměnných
  - **soukromé** (private) - jsou přístupné jen jednomu procesoru
  - **sdílené** (shared) - může k nim přistupovat více procesorů

Ošetření sdílených proměnných

- multiprocesor se sdílenou pamětí neumožňuje, aby současně přistupovalo více procesorů na stejné místo v paměti
  - pokud se tak stane výsledek je nepředvídatelný
  - většinou je nutné se této situaci vyhnout dobře napaným kódem
- jelikož jsou všechny přístupy do paměti cachovány, je nutné ošetřit správnou koherenci globálních paměti a lokálními

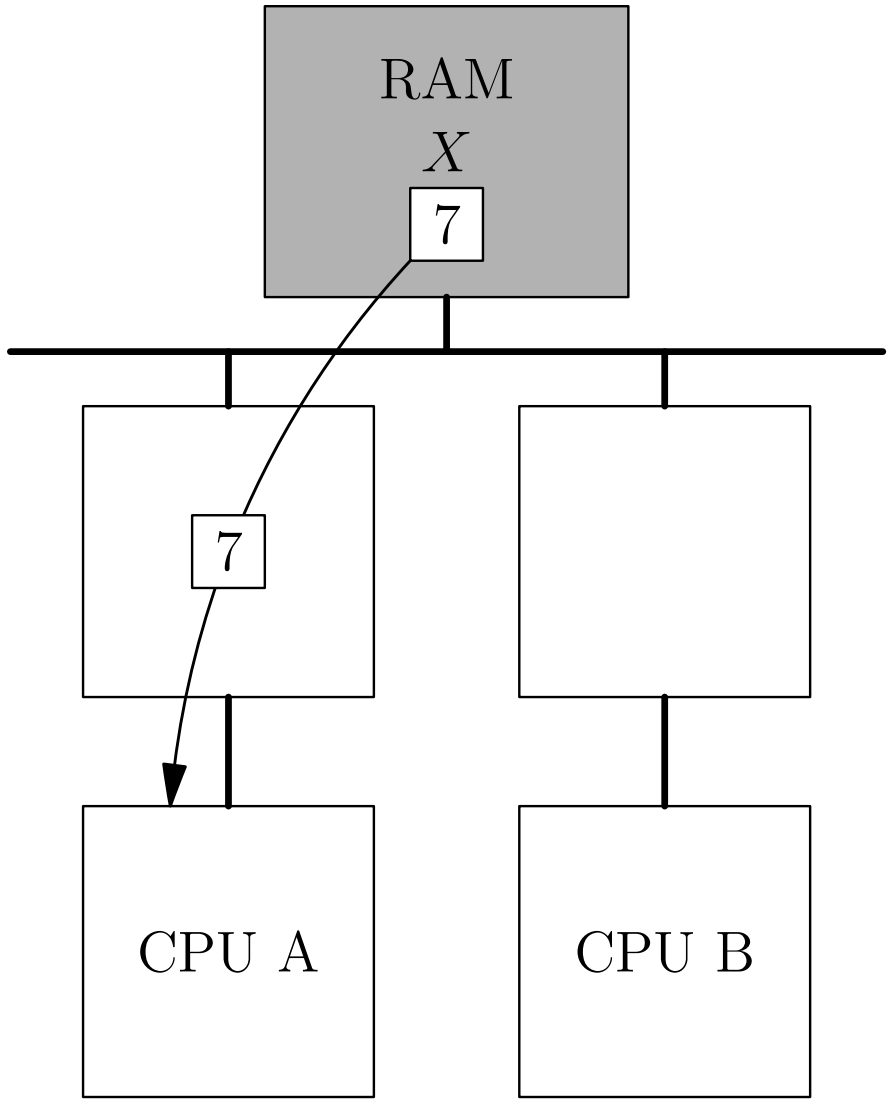
Multiprocesorové systémy se sdílenou pamětí III.

Cache coherence problem



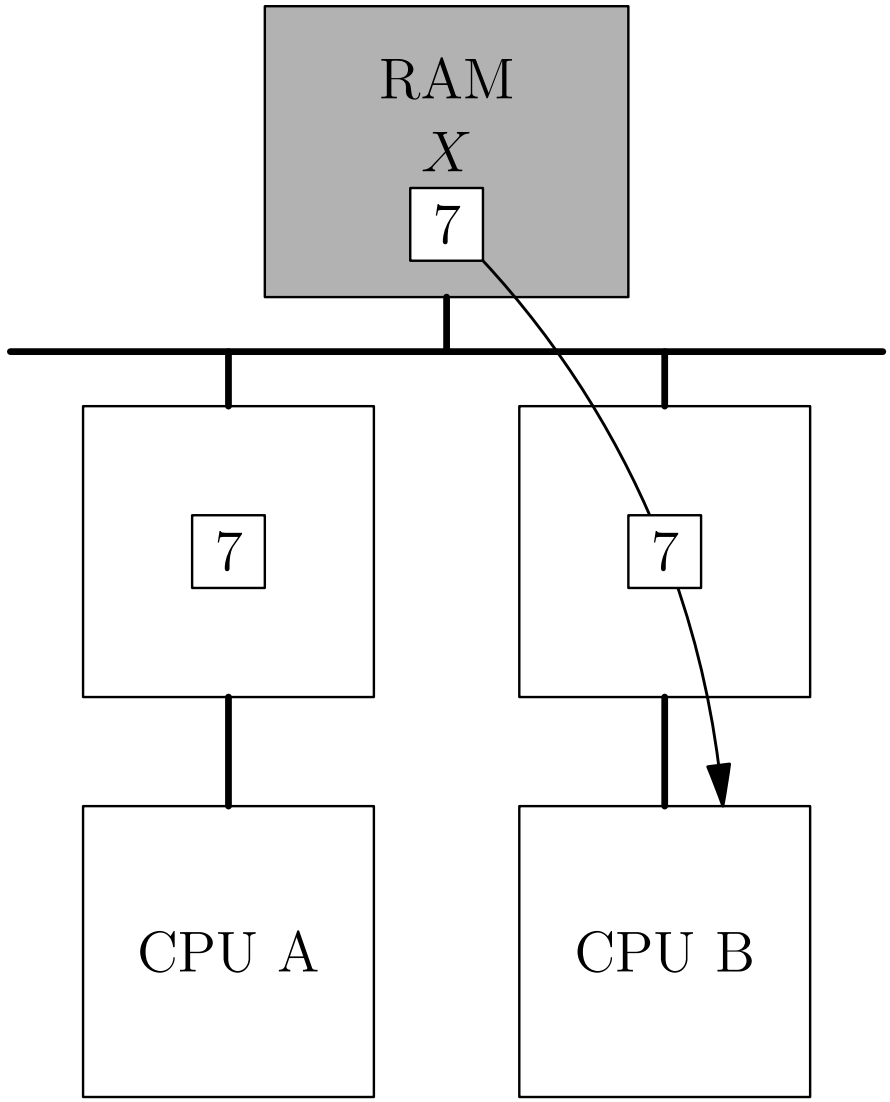
Multiprocesorové systémy se sdílenou pamětí IV.

CPU A načítá proměnnou *X*



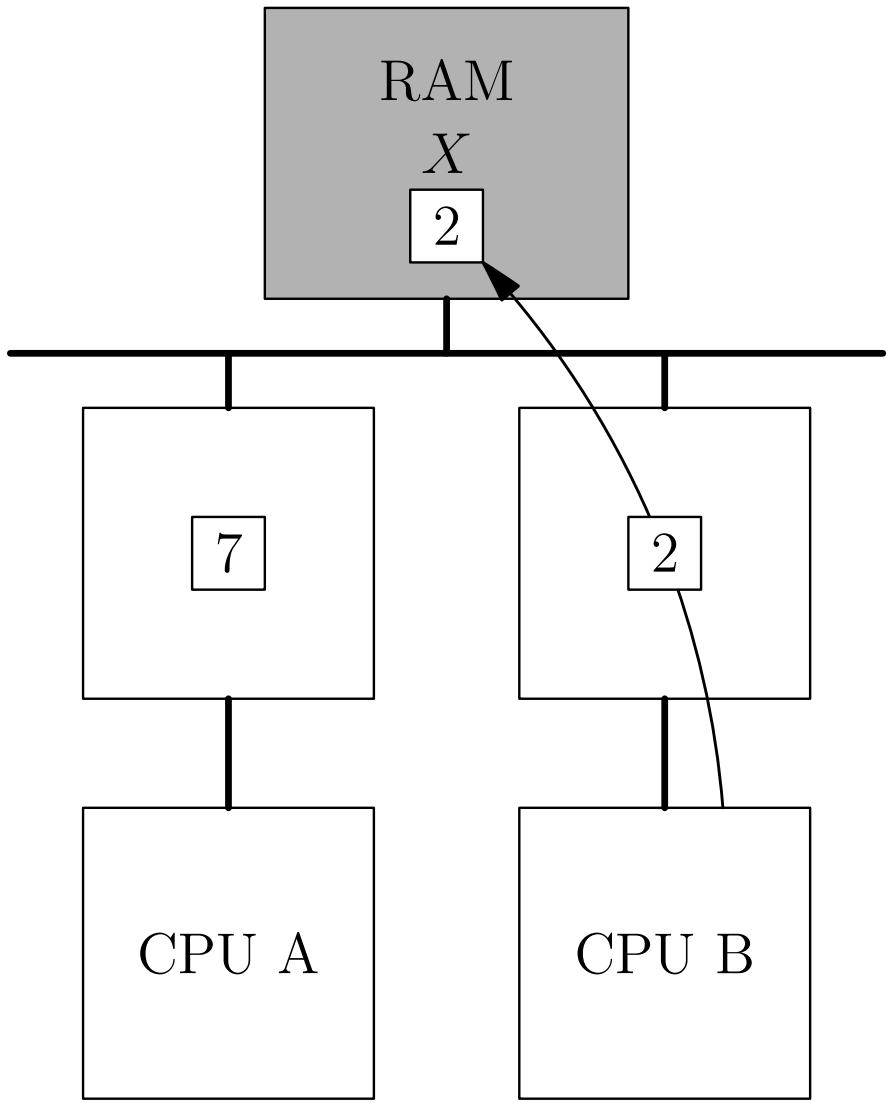
Multiprocesorové systémy se sdílenou pamětí IV.

CPU B načítá proměnnou *X*



Multiprocesorové systémy se sdílenou pamětí IV.

CPU B zapisuje 2 do X, což se neprojevuje v cache procesoru A



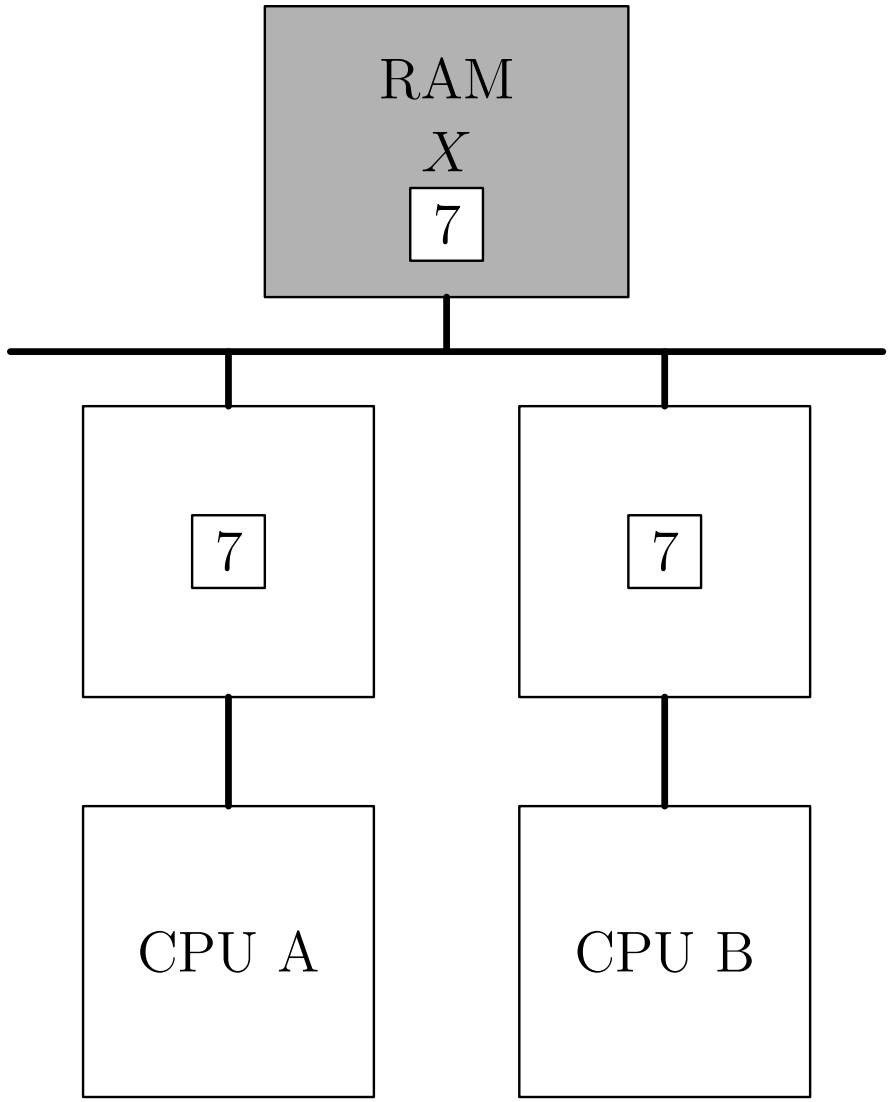
Multiprocesorové systémy se sdílenou pamětí V.

Cache coherence problem je řešen hardwarově. Existují dva způsoby řešení:

- **update protocol**
- **invalidate protocol**

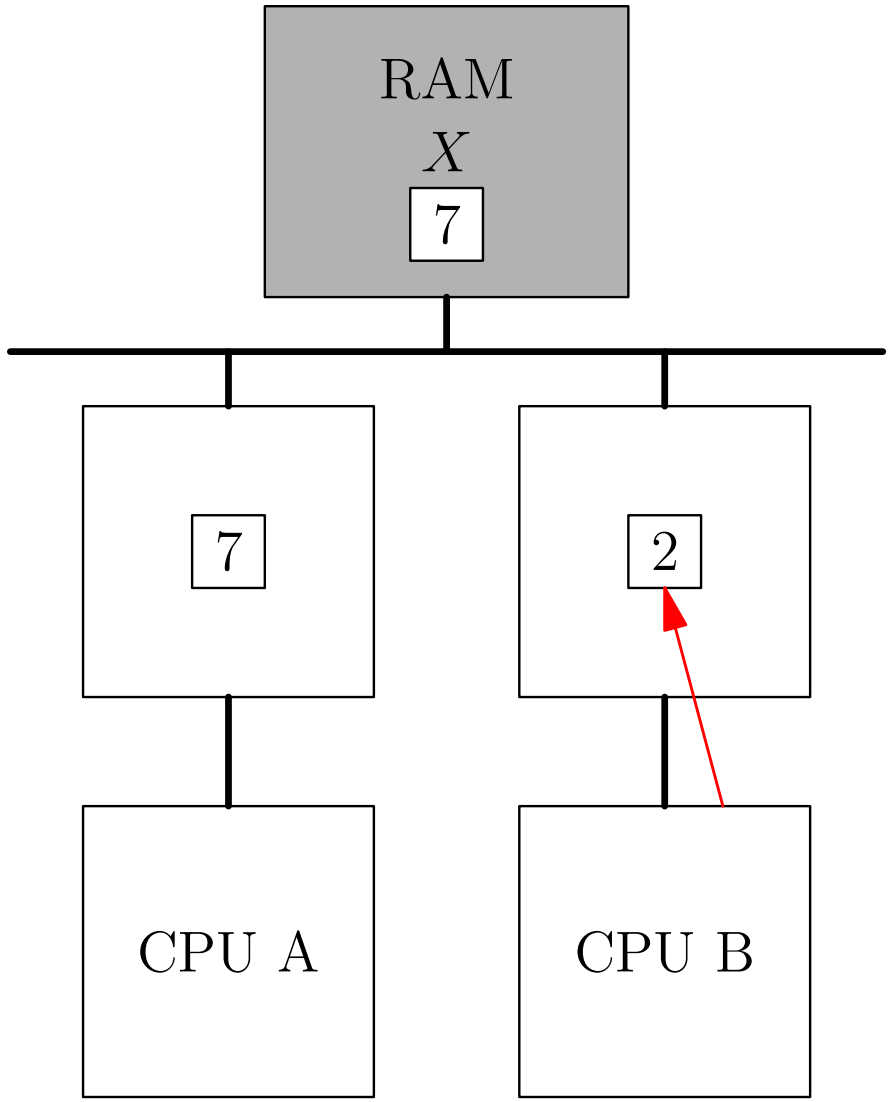
Multiprocesorové systémy se sdílenou pamětí VI.

Update protocol - X je sdílená proměnná



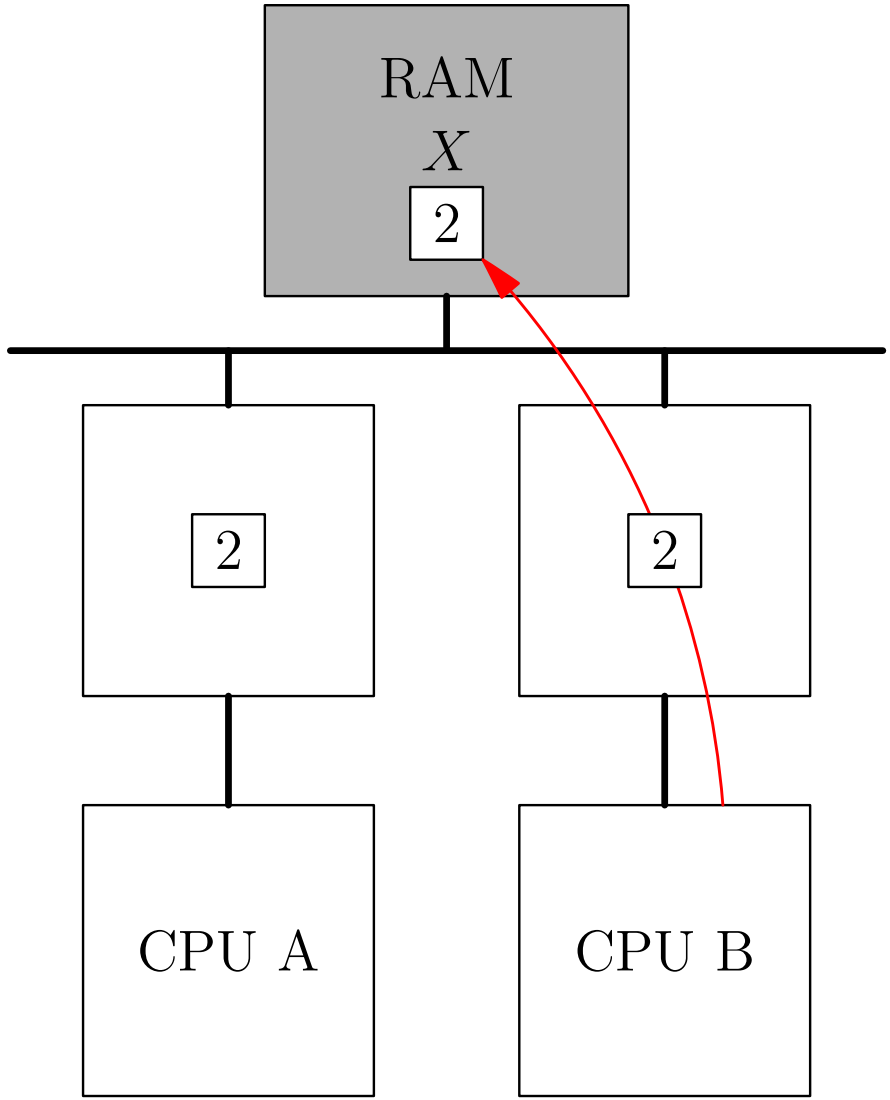
Multiprocesorové systémy se sdílenou pamětí VI.

Procesor *B* zapisuje 2 do *X* ve své cache, ...



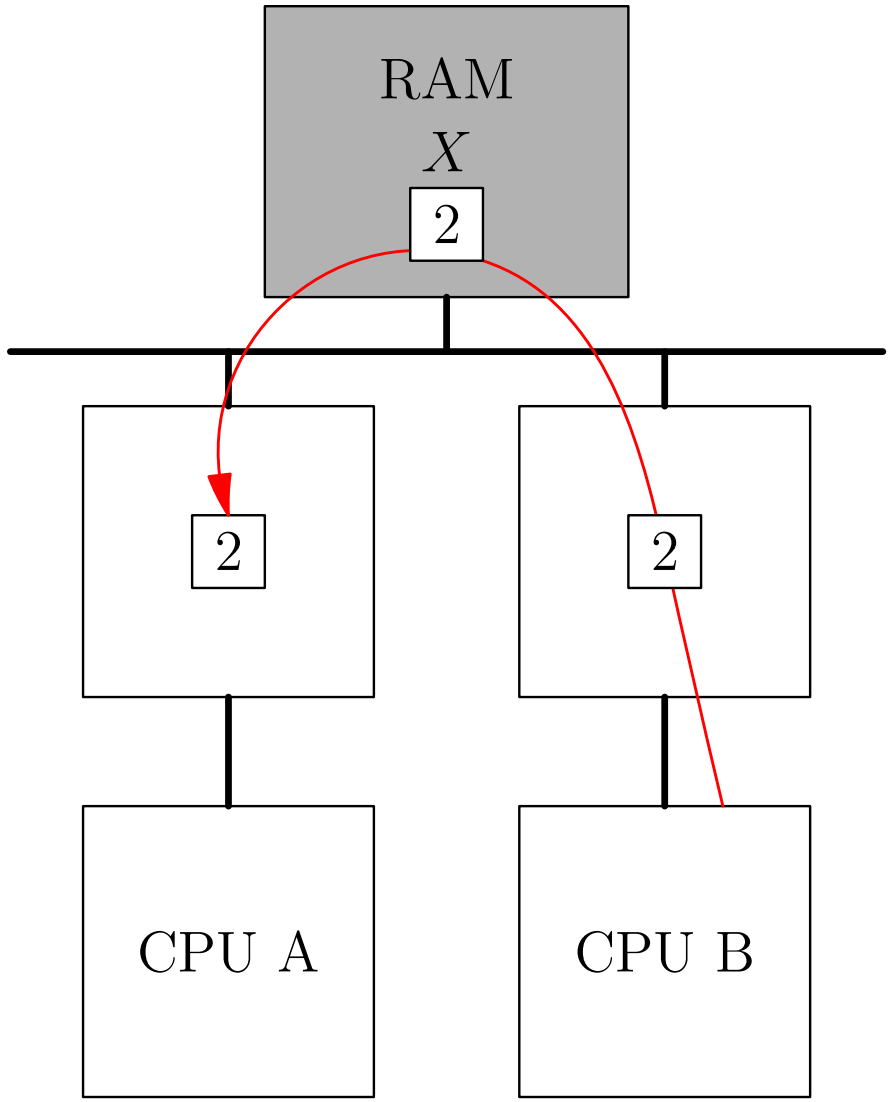
Multiprocesorové systémy se sdílenou pamětí VI.

... současně mění hodnotu X i v RAM ...



Multiprocesorové systémy se sdílenou pamětí VI.

... a v cache procesoru *A*.



Multiprocesorové systémy se sdílenou pamětí VII.

Nevýhody *update* protokolu:

- pokud procesor *A* načte proměnnou *X* jen jednou na začátku, a potom s ní pracuje pouze procesor *B*, zbytečně pokaždé posílá novou hodnotu

Multiprocesorové systémy se sdílenou pamětí VII.

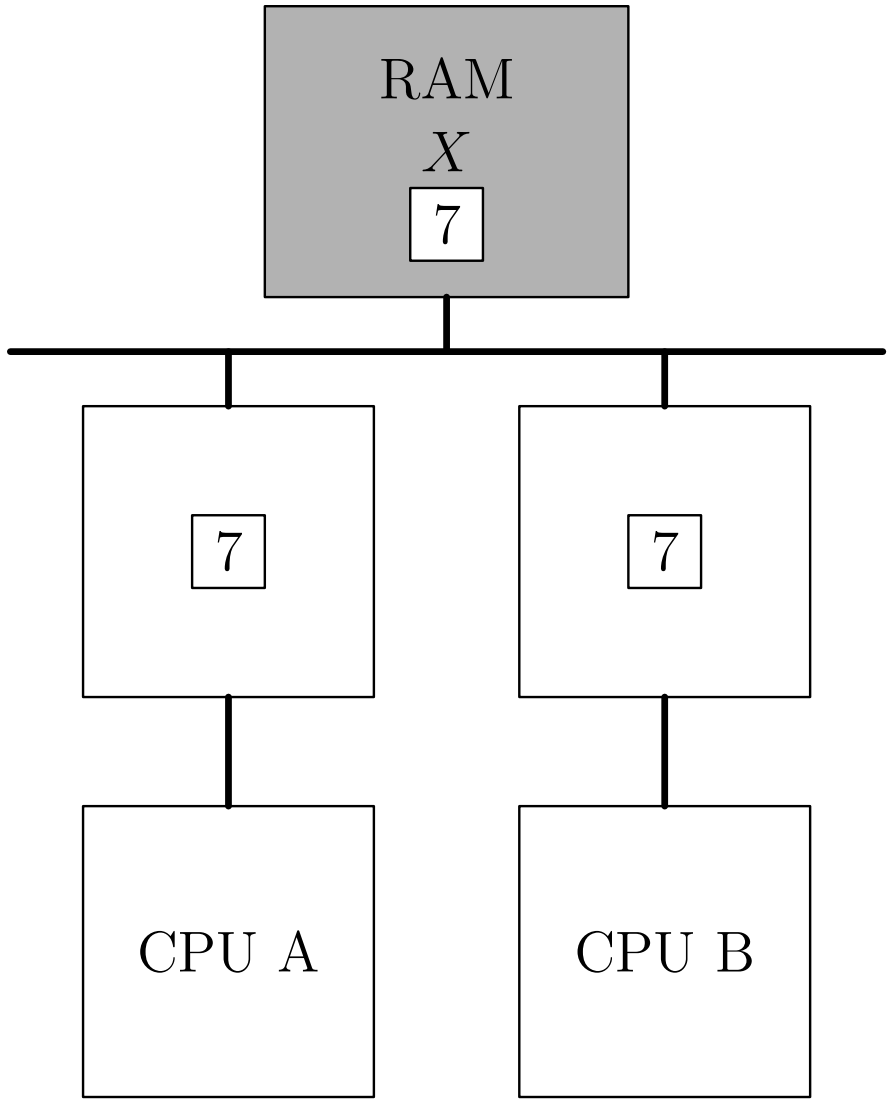
Nevýhody *update* protokolu:

- pokud procesor *A* načte proměnnou *X* jen jednou na začátku, a potom s ní pracuje pouze procesor *B*, zbytečně pokaždé posílá novou hodnotu

V současnosti se častěji používá *invalidate* protokol.

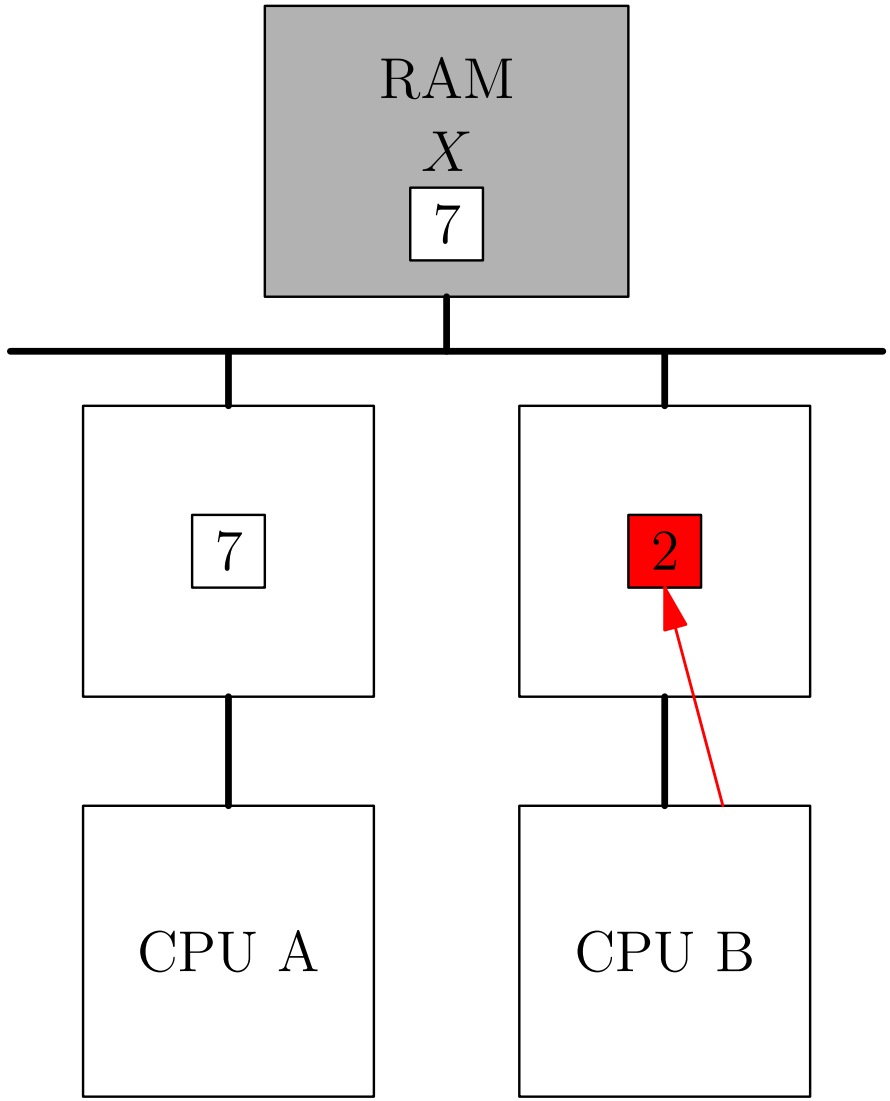
Multiprocesorové systémy se sdílenou pamětí VIII.

Invalidate protocol - X je sdílená proměnná



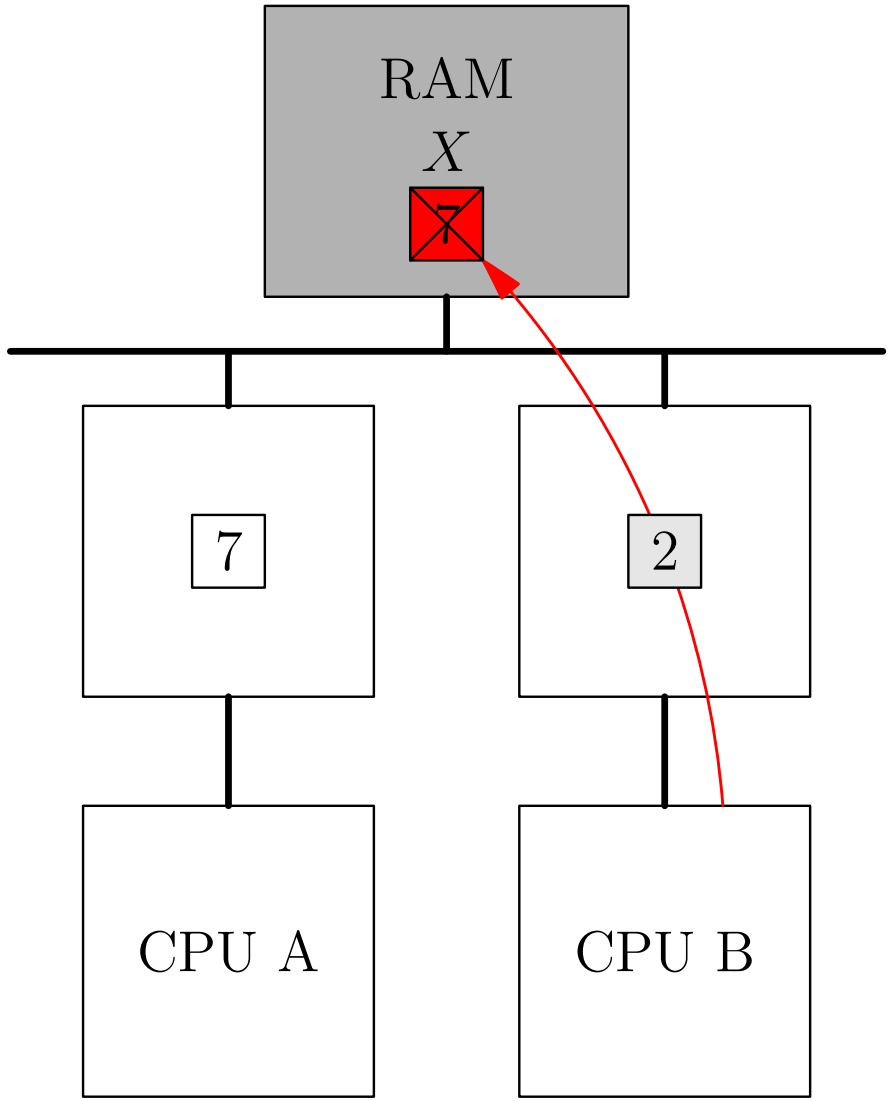
Multiprocesorové systémy se sdílenou pamětí VIII.

Procesor *B* zapisuje 2 do *X* ve své cache a označuje *X* jako *DIRTY*, ...



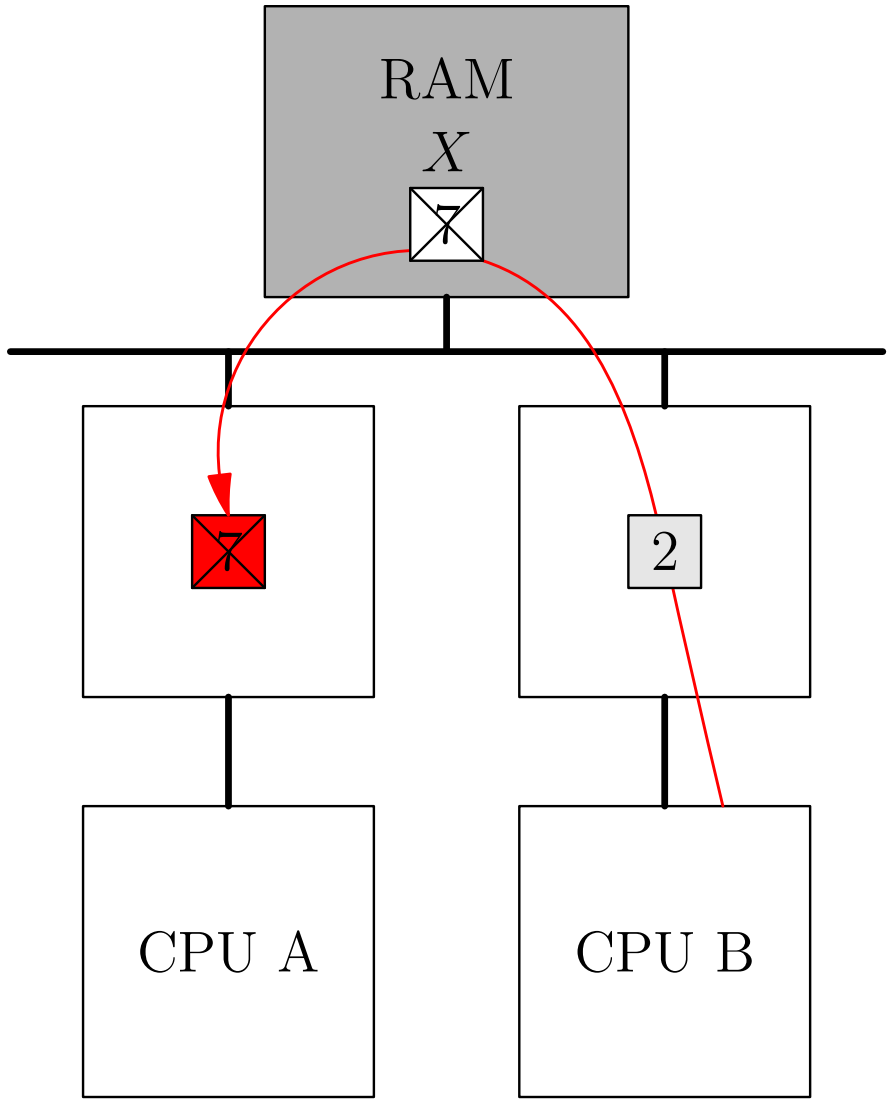
Multiprocesorové systémy se sdílenou pamětí VIII.

... současně označuje hodnotu *X* v RAM za neplatnou - *INVALID* ...



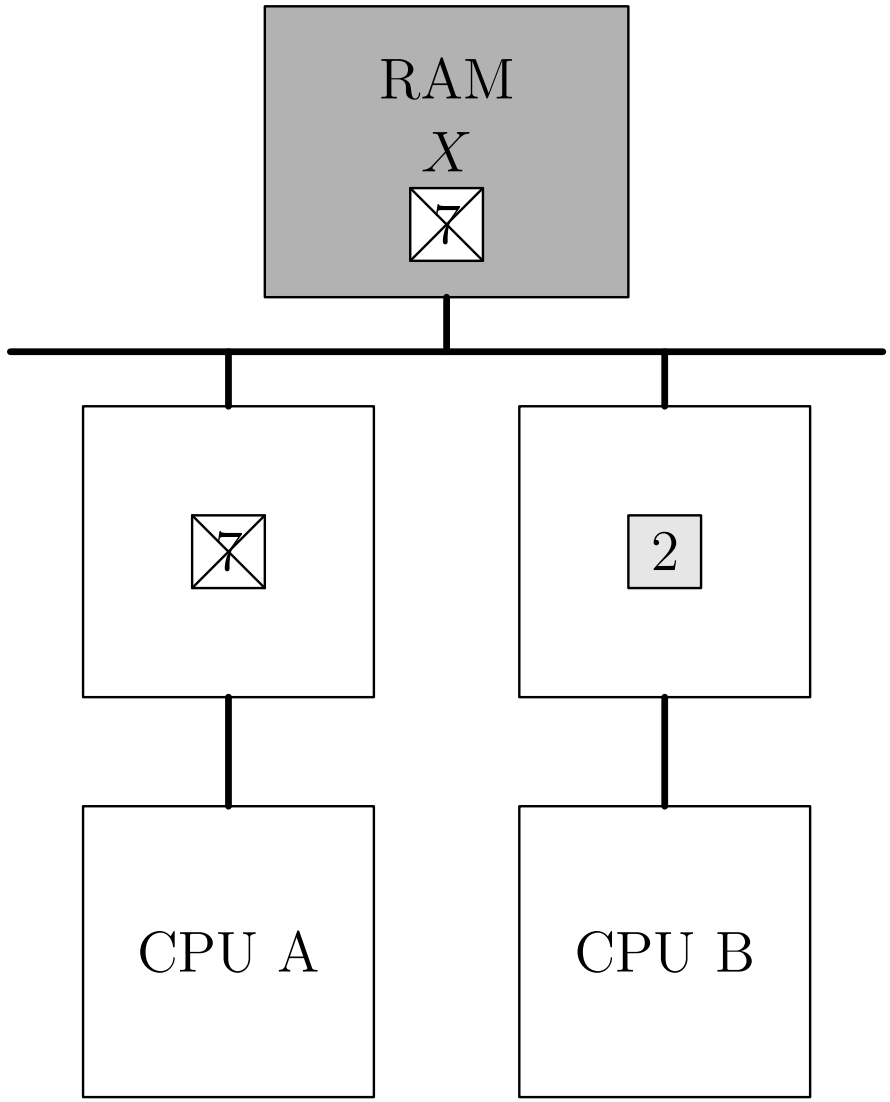
Multiprocesorové systémy se sdílenou pamětí VIII.

... a stejně tak označí i hodnotu *X* v cache procesoru *B*.



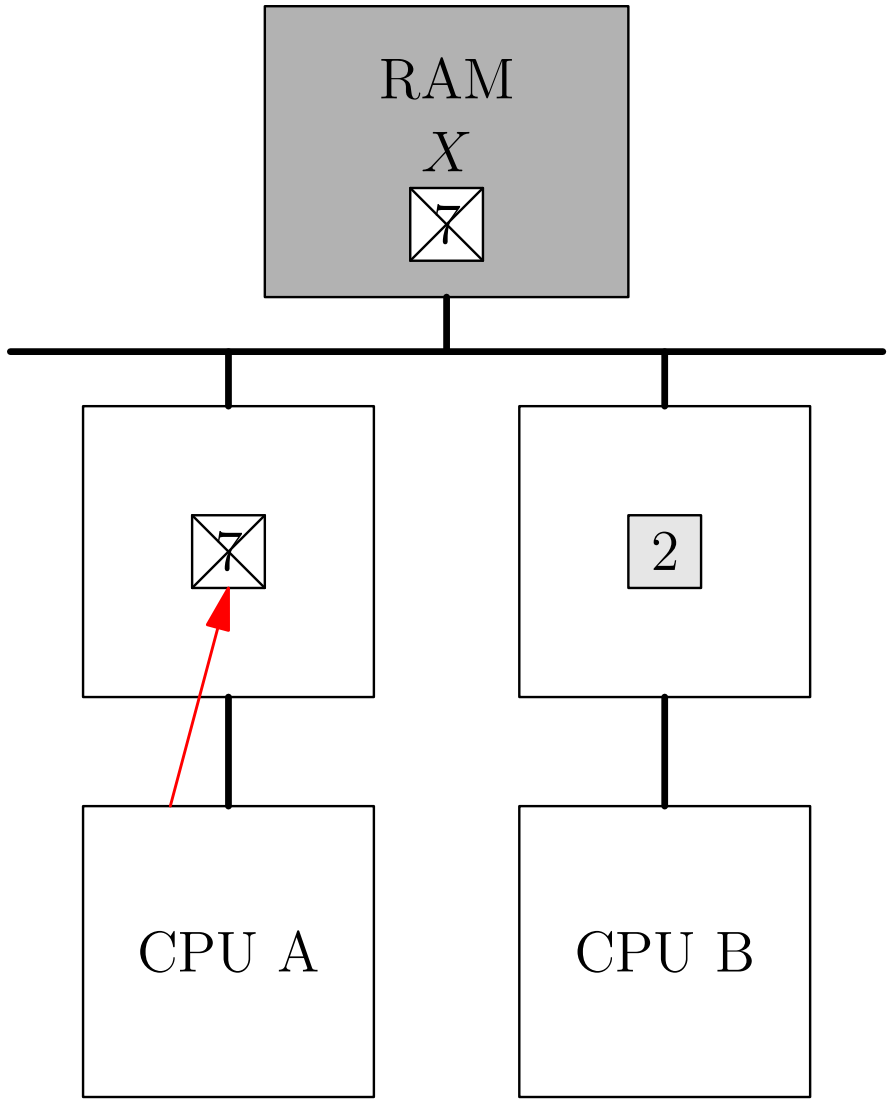
Multiprocesorové systémy se sdílenou pamětí VIII.

Nakonec je *X* *DIRTY* v cache CPU *B* a *INVALID* v RAM a cache CPU *A*.



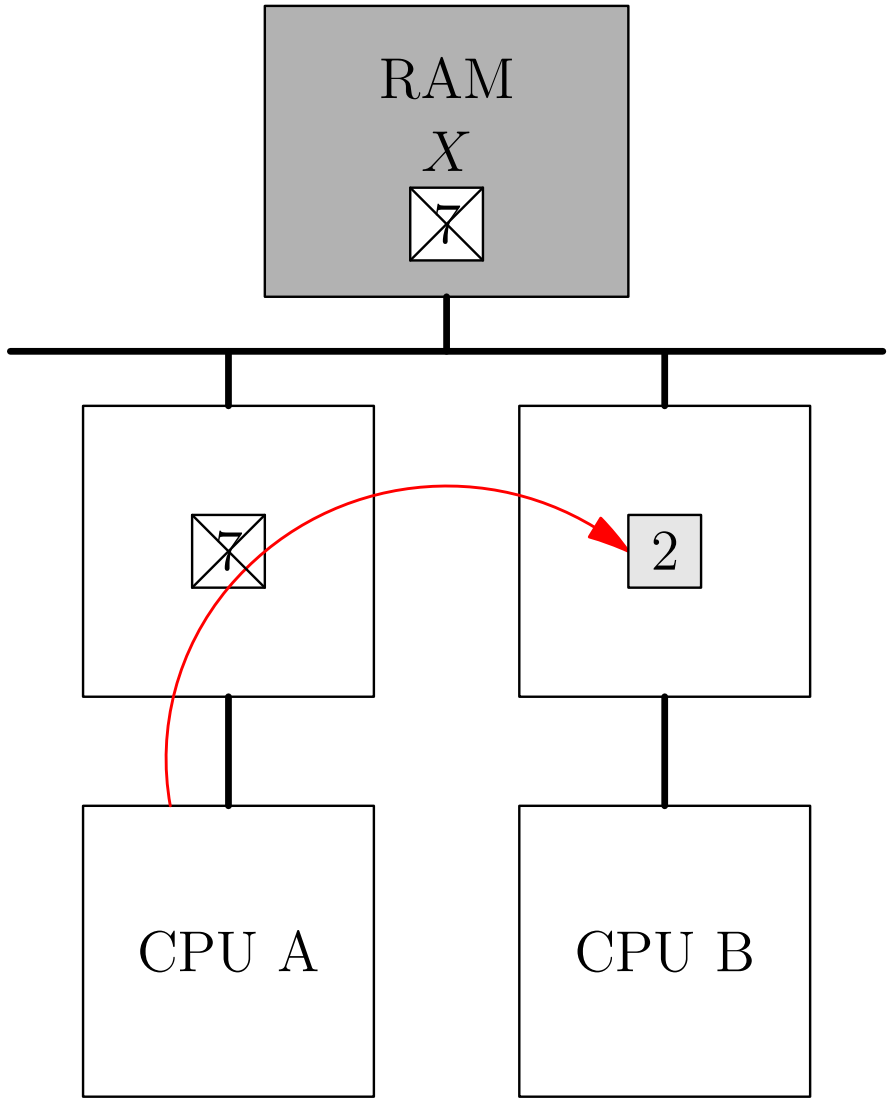
Multiprocesorové systémy se sdílenou pamětí VIII.

CPU *A* načítá *X* ze své cache a vidí ji označenou jako *INVALID*.



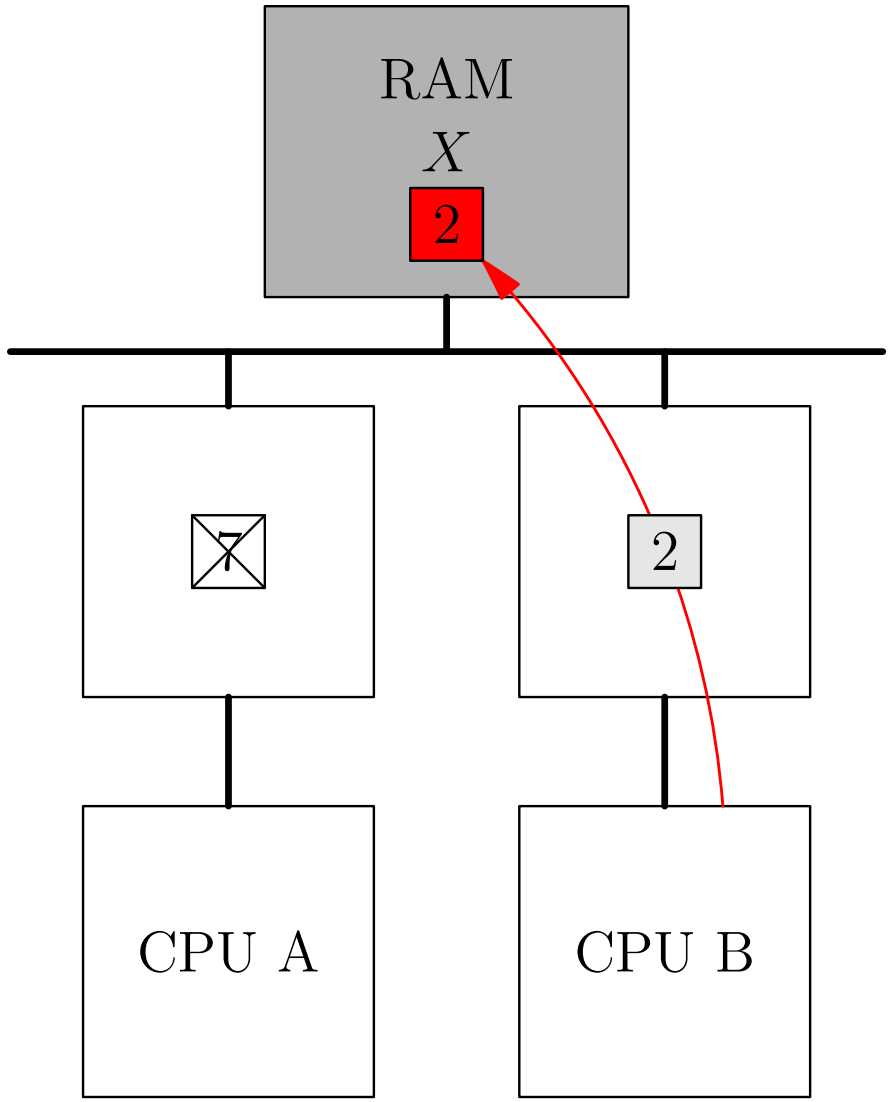
Multiprocesorové systémy se sdílenou pamětí VIII.

CPU *A* se tedy dotazuje CPU *B*, které má *X* označenou jako *DIRTY*.



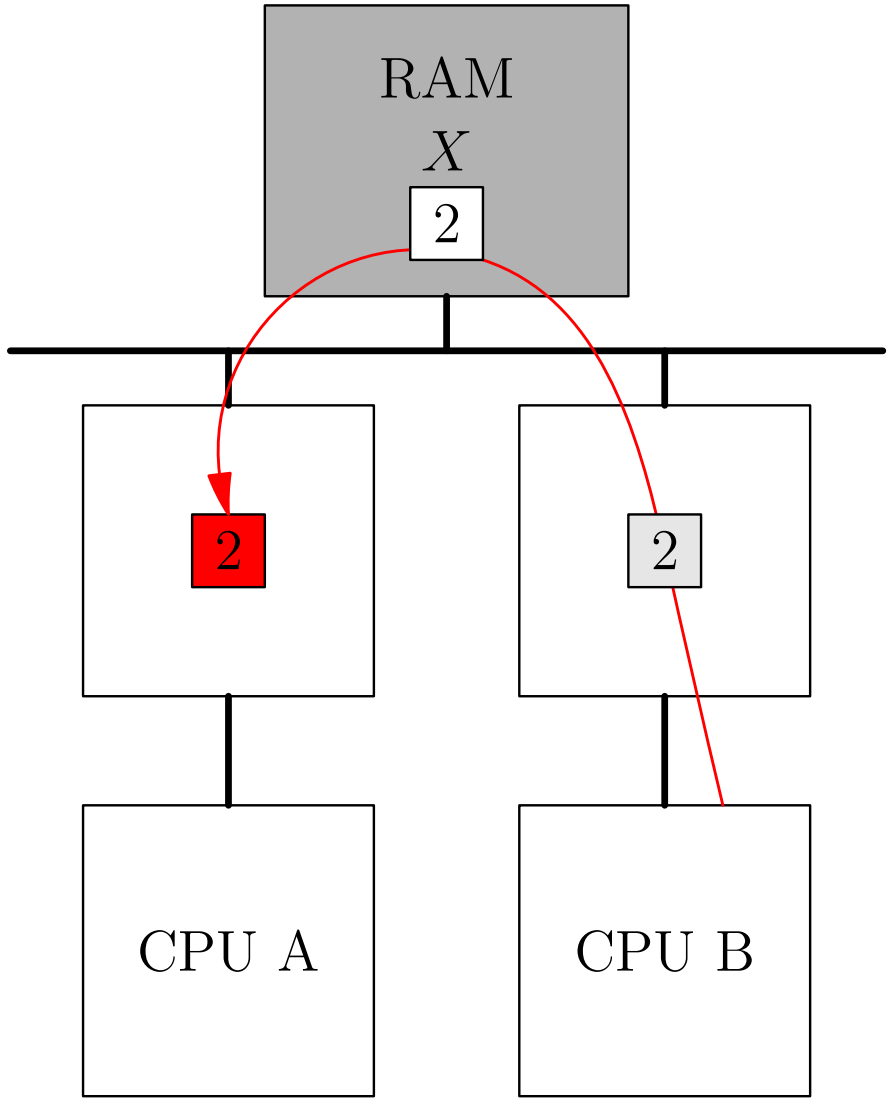
Multiprocesorové systémy se sdílenou pamětí VIII.

CPU *B* kopíruje hodnotu *X* do RAM ...



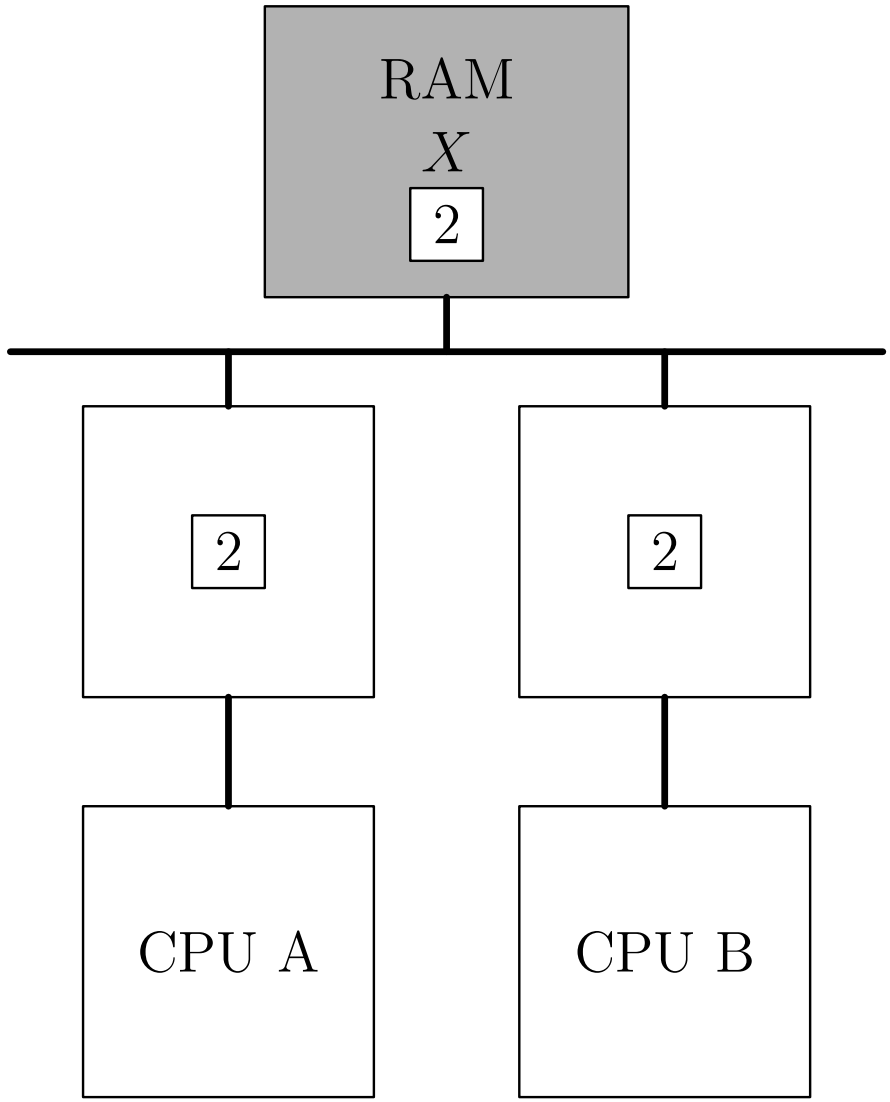
Multiprocesorové systémy se sdílenou pamětí VIII.

... a do cache CPU *A*.



Multiprocesorové systémy se sdílenou pamětí VIII.

Proměnná *X* je nakonec všude označena jako sdílená - *SHARED*.



Multiprocesorové systémy se sdílenou pamětí IX.

Nevýhody invalidate protokolu - tzv. **false sharing**:

- protokoly update/invalidate se ve skutečnosti vždy vztahují na celou *cache line*
- dva procesory mohou měnit dvě ruzné proměnné uložené ve stejné **cache line** (přitom každý jednu a tu samou),
  - např. dvě vlákna ukládají mezivýsledky do sdíleného pole
- systém to nepozná a stejně se pokaždé provádí *update*
- režie spojená s *invalidate* protokolem je tu zbytečná
- *update* protokol je v takové situaci lepší

Multiprocesorové systémy se sdílenou pamětí X.

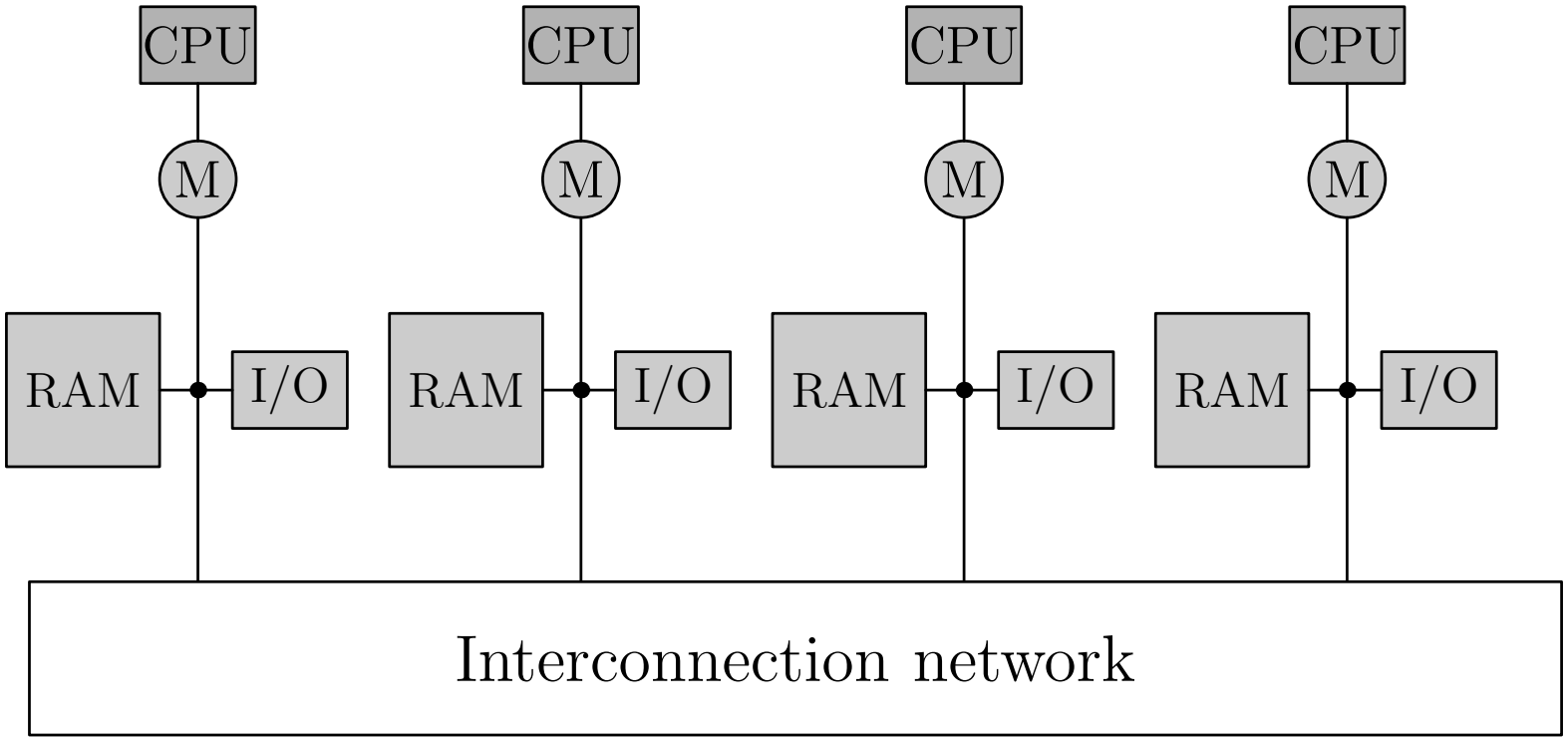
Snoopy cache system

- každý procesor odposlouchává všechnu komunikaci tj. i ostatních CPU
- podle toho pak nastavuje stavy *SHARE, INVALID* a *DIRTY* u sdílených proměnných

Multiprocesorové systémy s distribuovanou pamětí I.

- uzkým hrdlem multiprocesorů se sdílenou pamětí je datová komunikace
- s rostoucím počtem procesorů není paměťový systém schopen obsluhovat všechny požadavky
- problem *cache coherence* se také stává komplikovanější
- může se stát, že přidáváním dalších procesorů výkon multiprocesoru dokonce sníží
- počet CPU u multiprocesorů se sdílenou pamětí je omezen na několik desítek ≈ 64
- řešením je zavedení multiprocesorů s distribuovanou pamětí
  - příklad - Cray T3D

Multiprocesorové systémy s distribuovanou pamětí II.



Multiprocesorové systémy s distribuovanou pamětí III.

- každý procesor má rychlý přístup do své paměti, ale velice pomalý přístup do cizí paměti
- předpokládá se, že běžný kód splňuje *spatial locality*
- většina přístupů do paměti tedy bude probíhat na úrovni lokální paměti daného CPU
- což snižuje zátěž na IN - *interconnection*
- a to umožňuje vytvářet systémy s mnohem větším počtem CPU (řádově tisíce)
- soubor všech lokálních pamětí jednotlivých CPU dohromady tvoří jeden logický *adresový prostor*
- protože rychlost přístupu k jednotlivým adresám se liší, mluvíme o **NUMA** architektuře
  - **NUMA** = non-uniform memory acces
  - rozdíl může být až stonásobný

Multiprocesorové systémy s distribuovanou pamětí IV.

- problém *cache coherence* nemůže být řešen pomocí "čmuchacího" protokolu
  - multiprocesory s distribuovanou pamětí nepoužívájí sběrnici
  - ani s jiném komunikačním zařízením by to s tak velkým počtem procesorů nebylo realizovatelné
- místo toho se používá tzv. adresář → **directory-based protocol**

Multiprocesorové systémy s distribuovanou pamětí V.

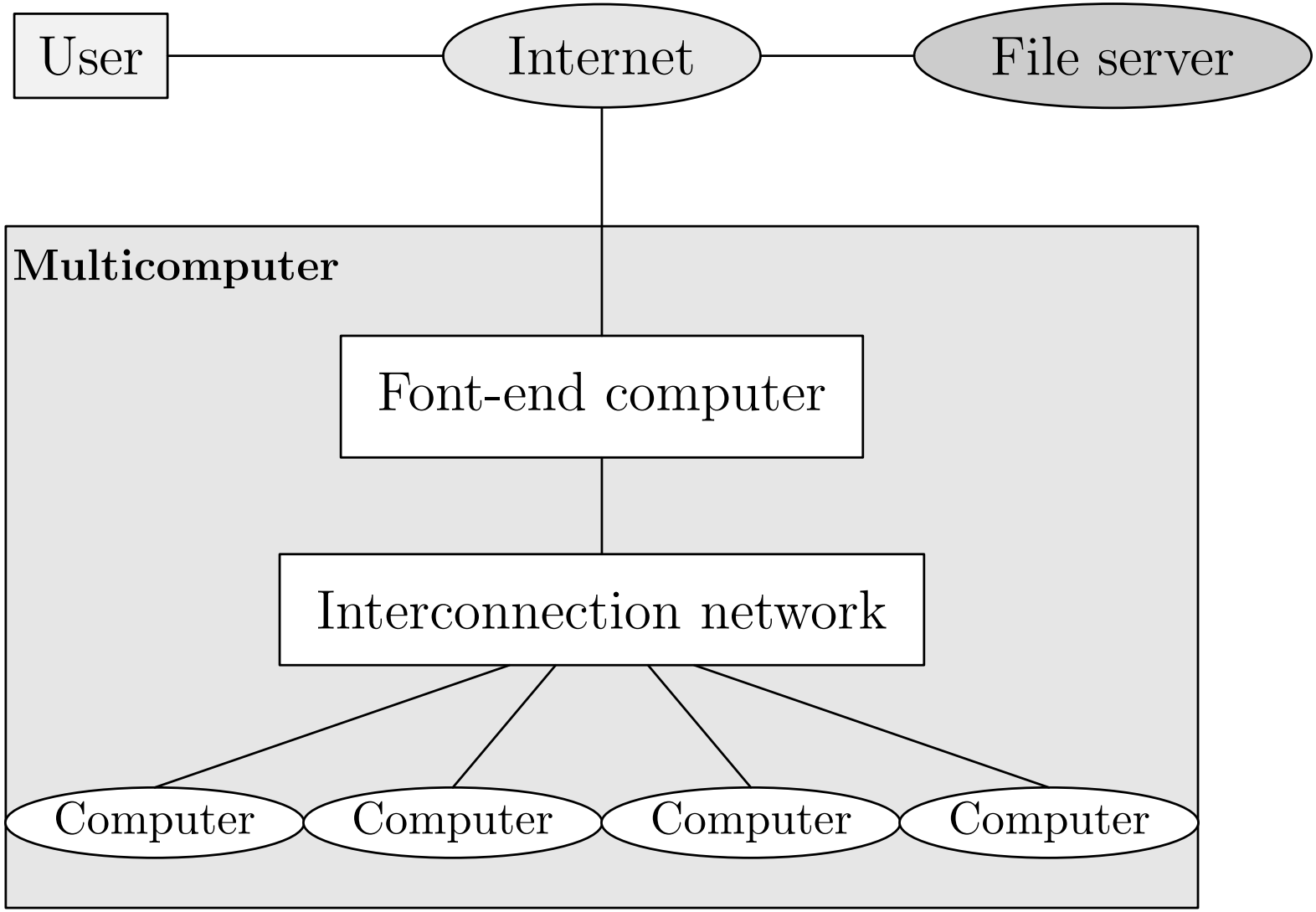
Directory based systems

- jde o systémy doplněné tabulkou popisující stavy jednotlivých bloků paměti
- ty mohou být opět *shared, invalid* a *dirty*
  - *SHARED* - proměnná je sdílena více CPU, ale všichni ji používají jen ke čtení - její hodnota v globální virtuální paměti je korektní
  - *INVALID* - jeden z procesorů provedl zápis - hodnota proměnné v globální virtuální paměti není správná
  - *DIRTY* - procesor, který má proměnnou označenou za *dirty*, je ten, který k ní má momentálně exklusivní přístup
- tento adresář může být velký a zásadně zpoalovat přístup do paměti
- někdy se provádí jeho rozdělení mezi všechny procesory, čímž se zátěž distribuuje na celý systém

Multipočítače

- *multicomputer* je podobný multiprocesoru se sdílenou pamětí, ale nemá společný adresový prostor
- každý procesor má přístup je do své vlastní paměti
- stejná adresa u dvou různých procesorů odpovídám různým paměťovým buňkám
- procesory spolu komunikují pouze posíláním zpráv přes komunikační síť, která je propojuje
- dělí se na
  - **nesymetrické**
  - **symetrické**

Nesymetrické multipočítače I.



Nesymetrické multipočítače II.

- první multipočítače byly asymetrické
- měly jeden počítač (*front-end computer*) pro komunikaci s uživateli a s I/O zařízeními
- dále obsahovaly řadu počítačů určených pouze pro výpočty - *back-end*
- tyto výpočetní počítače jsou vybaveny velmi malým a jednoduchým operačním systémem
  - jsou bez multitaskingu, virtuální paměti, I/O rozhraní
- příklady
  - Intel iPSC - 128 uzlů s operačním systémem NX
  - nCUBE/ten - Intel 80286 (font), 1024 uzlů s operačním systémem VERTEX

Nesymetrické multipočítače III.

Výhody nesymetrických multipočítačů:

- jednoduché výpočetní uzly jsou levnější, než plnohodnotné počítače
- jednoduchý operační systém výpočetních uzlů umožňuje optimálnější využití hardwaru
  - multitasking může výrazně zpomalovat

Nesymetrické multipočítače III.

Nevýhody nesymetrických multipočítačů:

- pokud selže přístupový počítač (*front-end*), je celý multipočítač nepřístupný
- škálovatelnost je omezena výkonem komunikační sítě a přístupového počítače
  - multipočítače jsou víceuživatelské
  - *front-end* je používán k ladění, kompilování, I/O operacím
  - při větším počtu uživatelů to může vést k přetížení
  - lze to částečně řešit přiráním dalšího *front-endu* - to ale není příliš elegantní
- primitvní operační systémy na výpočetních uzlech znesnadňují ladění kódu
  - *back-end* nedokáže jednoduše vypisovat ladicí zprávy
  - je nutné poslat po síti zprávu *front-endu*, který jí pak vypíše
- je nutné psát zvláštní kód pro *front-end* a *back-end*

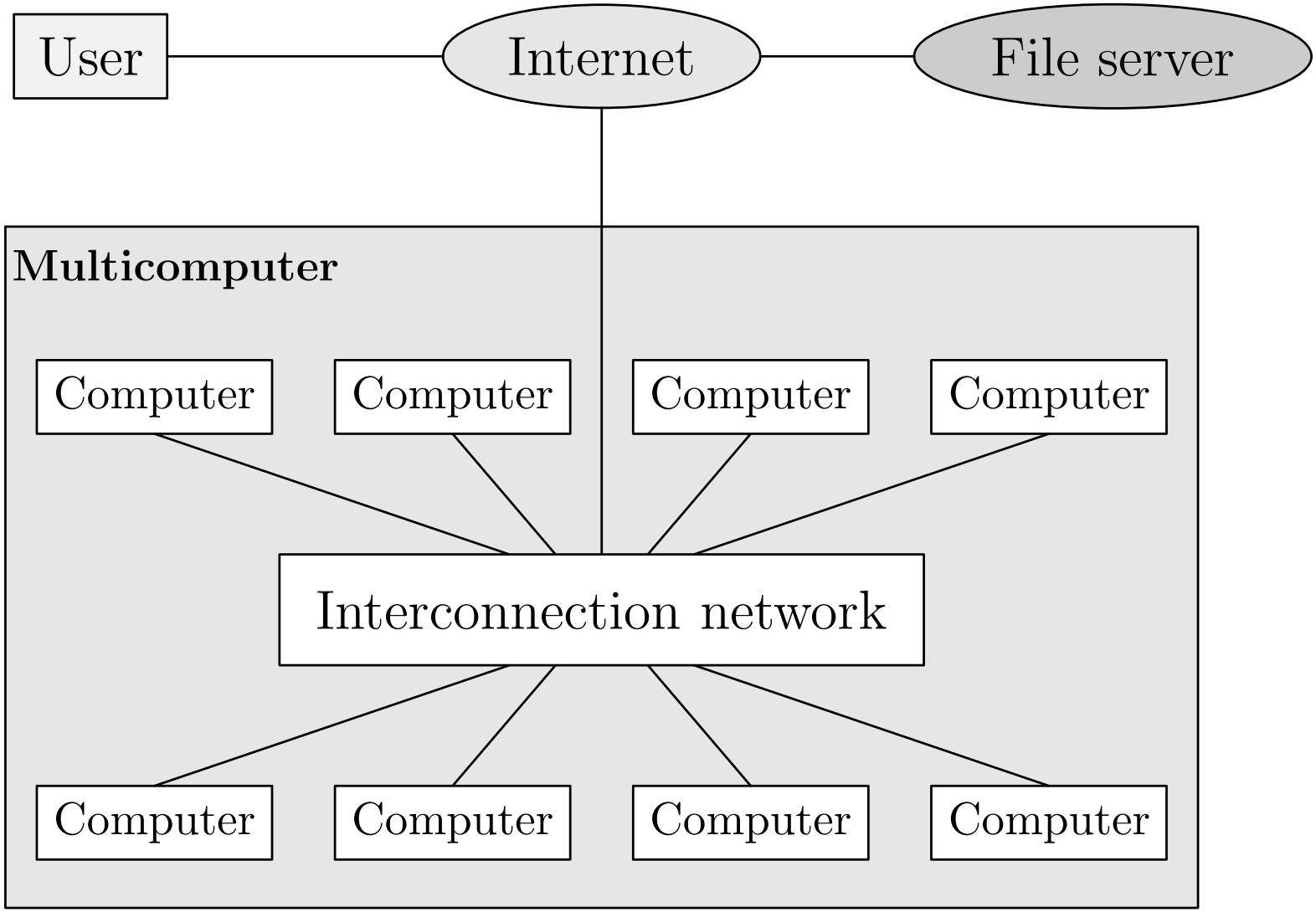
Nesymetrické multipočítače III.

Nevýhody nesymetrických multipočítačů:

- pokud selže přístupový počítač (*front-end*), je celý multipočítač nepřístupný
- škálovatelnost je omezena výkonem komunikační sítě a přístupového počítače
  - multipočítače jsou víceuživatelské
  - *front-end* je používán k ladění, kompilování, I/O operacím
  - při větším počtu uživatelů to může vést k přetížení
  - lze to částečně řešit přiráním dalšího *front-endu* - to ale není příliš elegantní
- primitvní operační systémy na výpočetních uzlech znesnadňují ladění kódu
  - *back-end* nedokáže jednoduše vypisovat ladicí zprávy
  - je nutné poslat po síti zprávu *front-endu*, který jí pak vypíše
- je nutné psát zvláštní kód pro *front-end* a *back-end*

Zejména poslední dva body vedly k přechodu k symetrickým multipočítačům.

Symetrické multipočítače I.



Symetrické multipočítače II.

- u symetrického multipočítače jsou všechny počítače rovnocenné
- na každém počítače běží stejný, plně funkční operační systém
- uživatel se může přihlásit na libovolný uzel
- programátor nemusí rozlišovat mezi kódem pro přístupový a výpočetní uzel
  - má-li některý počítač provést odlišný kód, řeší se to `if-then-else` konstrukcí

Symetrické multipočítače III.

Nevýhody symetrických multipočítačů

- nevytváří dojem jednotného paralelního počítače, ale spíše shluku propojených počítačů
  - lze k nim přistupovat přes více IP adres
- pokud každý počítač slouží jako *front-end* tj. k ladění a kompilování, je těžké vybalancovat zatížení jednotlivých uzlů
- uzly, které by měly sloužit jen k výpočtům, jsou zatěžovány obsluhou uživatelů

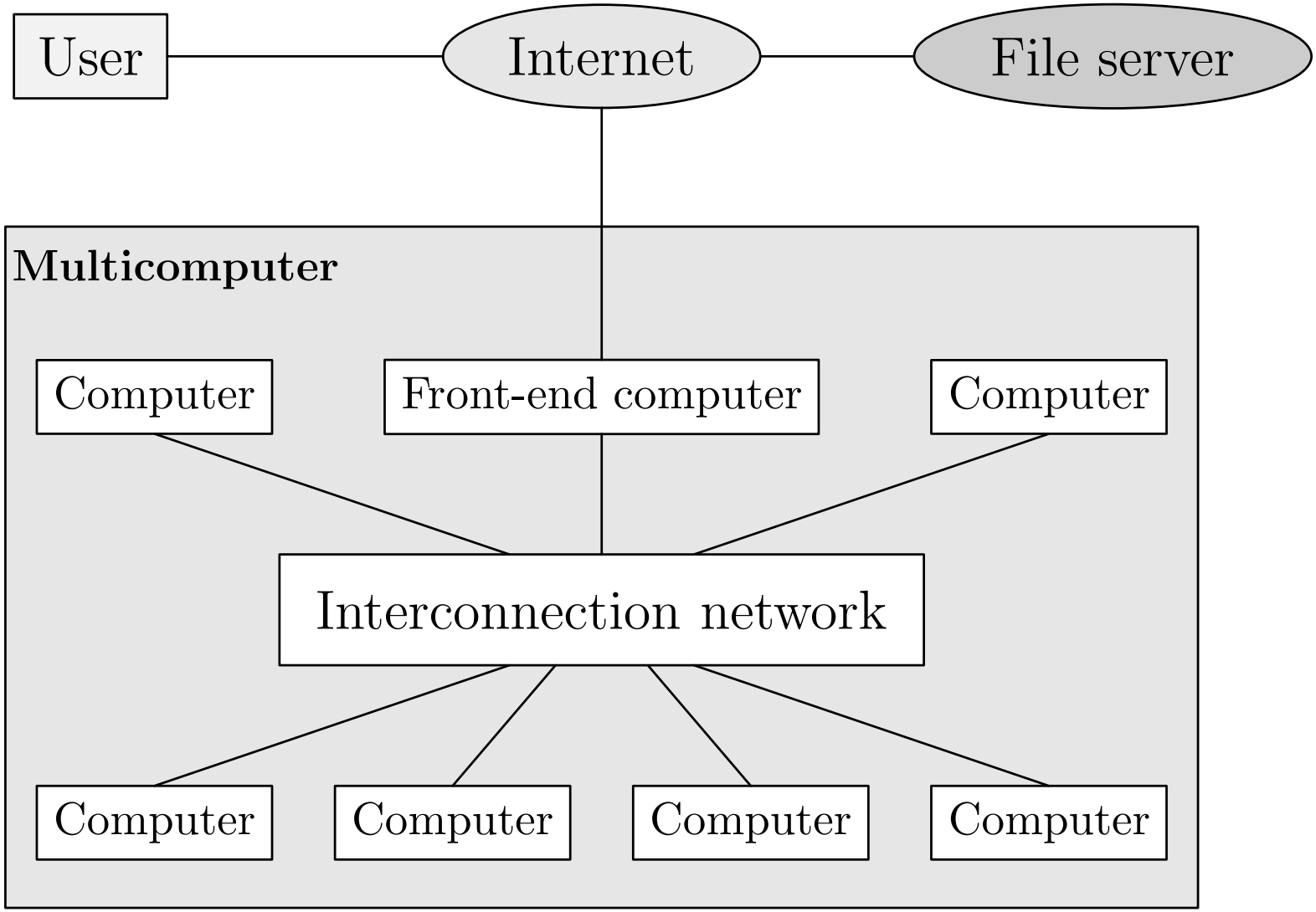
Symetrické multipočítače III.

Nevýhody symetrických multipočítačů

- nevytváří dojem jednotného paralelního počítače, ale spíše shluku propojených počítačů
  - lze k nim přistupovat přes více IP adres
- pokud každý počítač slouží jako *front-end* tj. k ladění a kompilování, je těžké vybalancovat zatížení jednotlivých uzlů
- uzly, které by měly sloužit jen k výpočtům, jsou zatěžovány obsluhou uživatelů

Za účelem odstranění těchto problémů se symetrický multipočítač doplní přístupovým uzlem - *front-endem*.

Symetrické multipočítače IV.



Symetrické multipočítače IV.

Příkladem multipočítačů jsou zejména klastry (*cluster*).
Jaké jsou rozdíly mezi klastrem a počítačovou sítí?

- uzly klastru nemají připojené displeje
- na všech uzlech by měl běžet zcela identický operační systém
- je kladen větší dúraz na kvalitu síťového spojení
  - gigabitový Ethernet (1Gb/sec, 100μsec), Myrinet (20Gb/sec, 7μsec)

Symetrické multipočítače V.

Dalším příkladem může být tzv. **massive parallel processing**. Jde o klastr se speciální vysoce rychlostní komunikační sítí.

Model: IBM BCX/5120 (**CINECA** Italy)
Architecture: eServer e326 Cluster Opteron
Processor Type: Opteron Dual Core 2.6 GHz
Number of Processors: 2560
Nodes: 1280 (4 core per node)
RAM: 10240 GB
Internal Network: Infiniband (5Gb/s)
Disk Space: 100 TB
Operating System: Red Hat RHEL4
Peak Performance: 26.6 TFlop/s
Available compilers: Fortran F90, C, C++
Parallel libraries: MPI, OpenMP
In November 2006 IBM BCX/5120 has been ranked 44th in TOP500.

Grid

- podobá se klastru obsahujícím tisíce uzlů
- jednoltivé uzly jsou většinou běžná PC
- grid je mnohem více heterogení
  - obsahuje uzly s různým výkonem, architekturou i operačním systémem
- nevyžaduje, aby všechny uzly byly neustále dostupné

GPU I.

GPU = graphical processing unit

- jde o akcelerátory výpočtů nutných pro vizualizaci 3D dat
- využívá se faktu, že provádění operací je na sobě nezávislé
  - transformovat 1,000,000 polygonů
  - nanést textury
  - vypočítat osvětlení
  - ...
- na podmínky se naráží prakticky jen při ořezavání zakrytých ploch
- to umožňuje efektivní využití téměř libovolného počtu výpočetních jednotek
  - mohou jich být tisíce
  - dnes je 100 - 800 (nVidia, ATI)
- pipeline se nemusí nikdy vyprazdňovat z důvodu neúspěšného spekulativního provádění kódu, může být tedy mnohem delší

GPU II.

GPU - pokračování

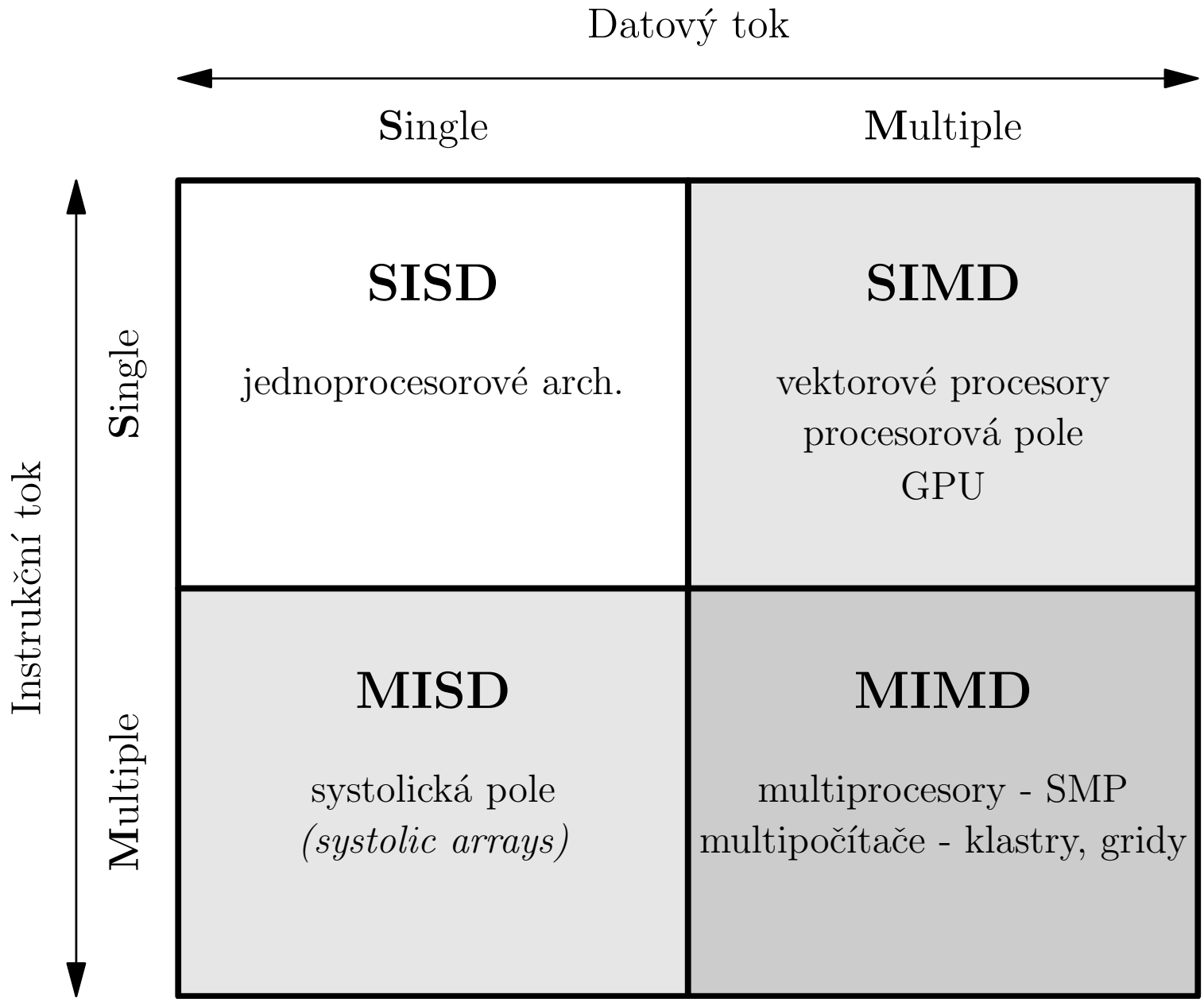
- do paměti se přistupuje vždy sekvenčně
  - jde hlavně o načítání velkých textur
- paměťový subsytém je k tomu uzpůsoben

Dělení paralelních architektur

Existuje mnoho způsobů, jak klasifikovat paralelní architektury. Nejznámější jsou:

- Flynnova taxonomie - 1966
  - rozděluje paralelní architektury v závislosti na toku instrukcí a dat
- rozdělení podle způsobu komunikace
  - architektury se sdílenou a distribuovanou pamětí

Flynnova taxonomie I.



Flynnova taxonomie II.

SISD architektury

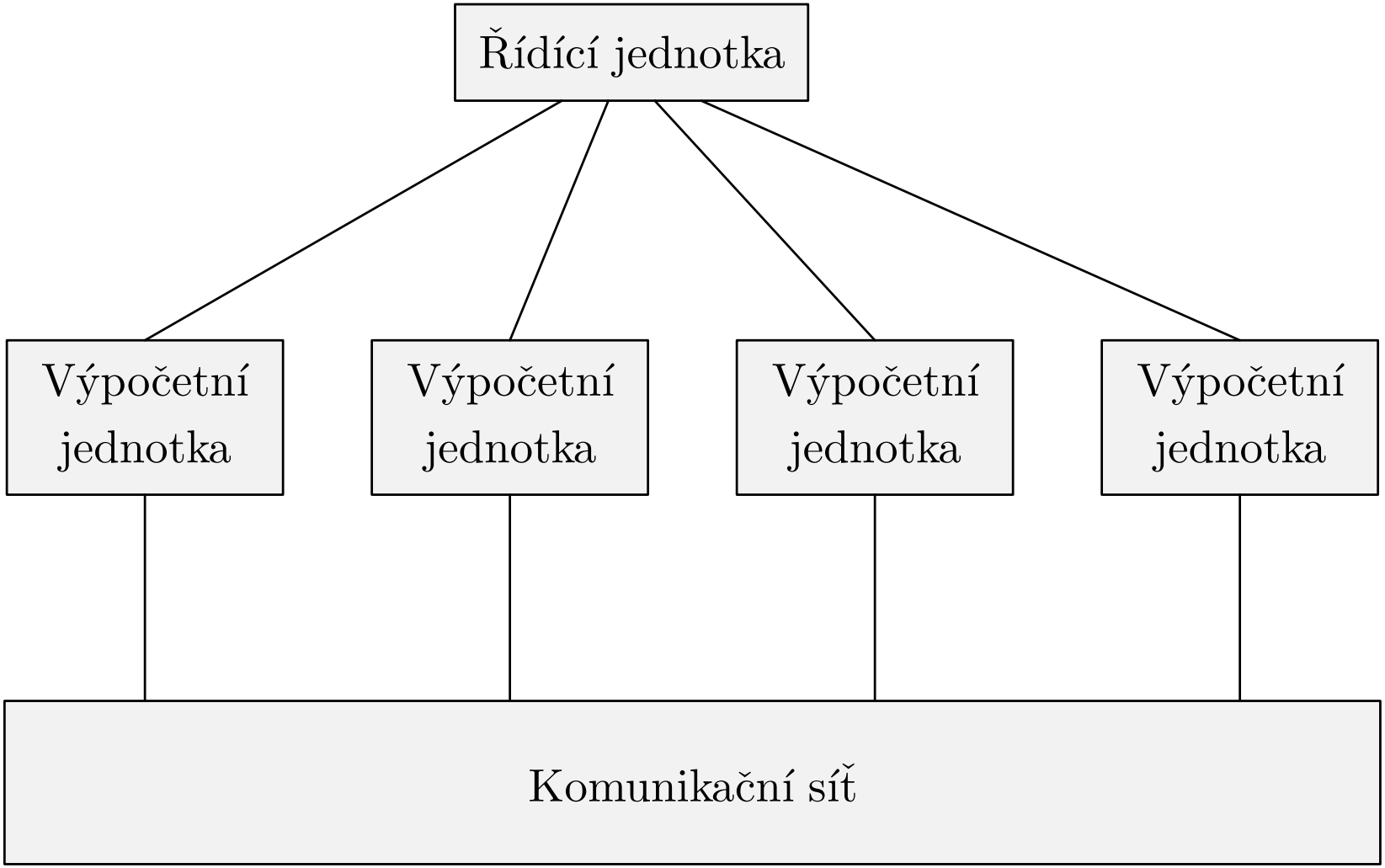
- na jednu instrukci připadá jeden jednoduchý datový typ
- jde typicky o jednoprocesorové architektury
- ty sice mohou zpracovávat více instrukcí a dat současně...
  - pipelining, superskalární zpracování, ...
- ... to je ale považováno spíše za paralelní **zpracování** sekvenčního kódu než za **provedení** paralelního kódu
- navíc - čistě sekvenční architektury dnes prakticky neexistují

Flynnova taxonomie - SIMD I.

SIMD architektury

- na jednu instrukci připadá více dat
  - např. instrukce sečíst dva vektory
- tyto architektury mají jednu řídící jednotku a několik jednotek výpočetních
- výpočetní jednotky v daný okamžik provádí stejnou instrukci, každá ale s různými daty
- jde hlavně o vektorové procesory ...
  - MMX, SSE, 3DNow! rozšíření procesorů architektury x086
  - GPU
    - moderní GPU umožňují rozdělení výpočetních jednotek do více skupin, které pak mohou zpracovávat odlišné úlohy
    - např. rozdělení na *vertex* a *pixel shadery*
- ... nebo procesorová pole

Flynnova taxonomie - SIMD II.

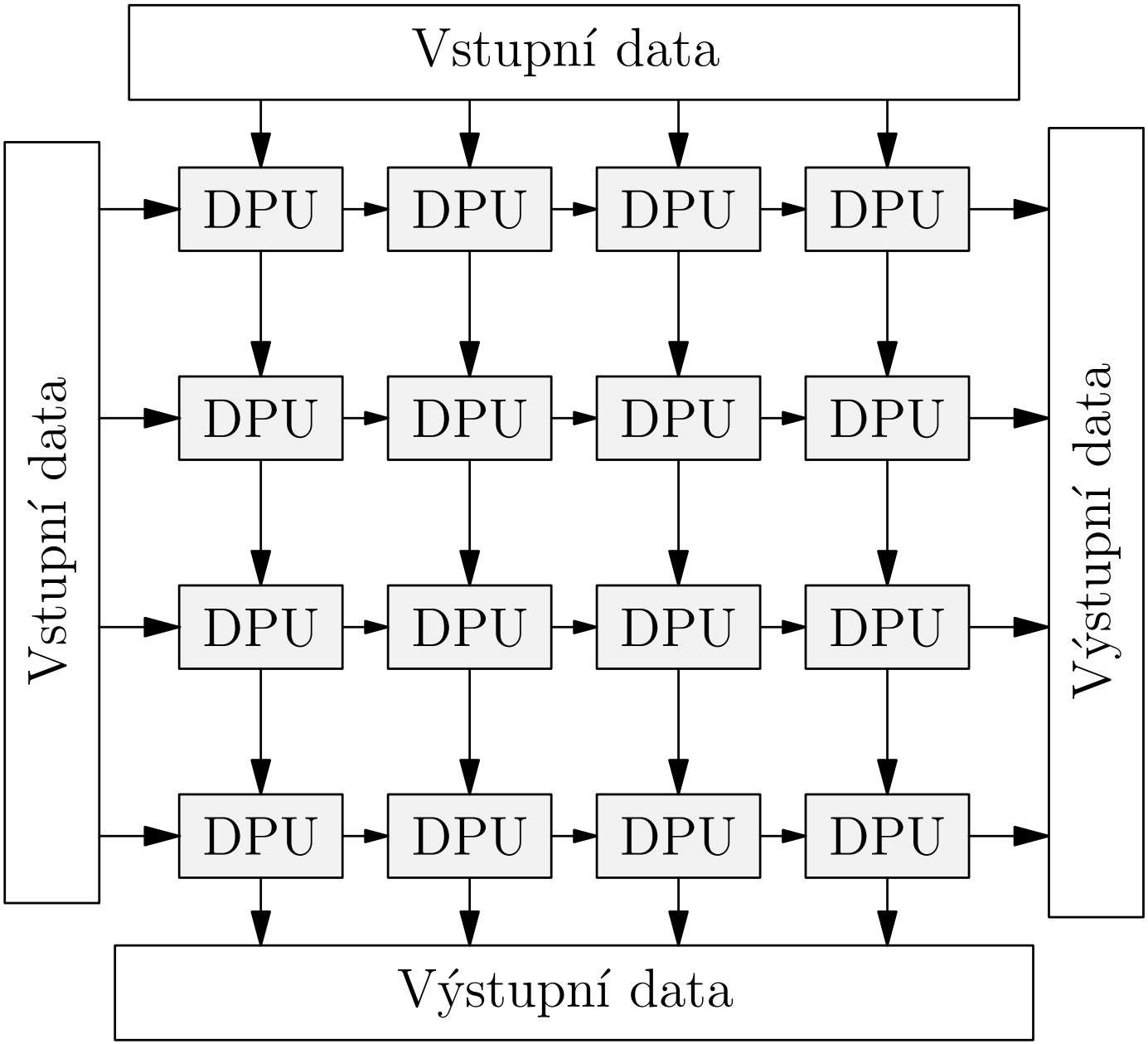


Flynnova taxonomie - MISD I.

MISD

- jedny data jsou postupně zpracovány více instrukcemi
- typickým zástupcem jsou systolická pole
  - název pochází od slova *systola* = srdeční kontrakce pumpující krev
- systolická pole jsou velmi speciální architektury
- příklady použití
  - některé třídící algoritmy
  - Hornerovo schéma pro vyčíslení polynomu
  - násobení matic - Cannonův algoritmus

Flynnova taxonomie - MISD II.

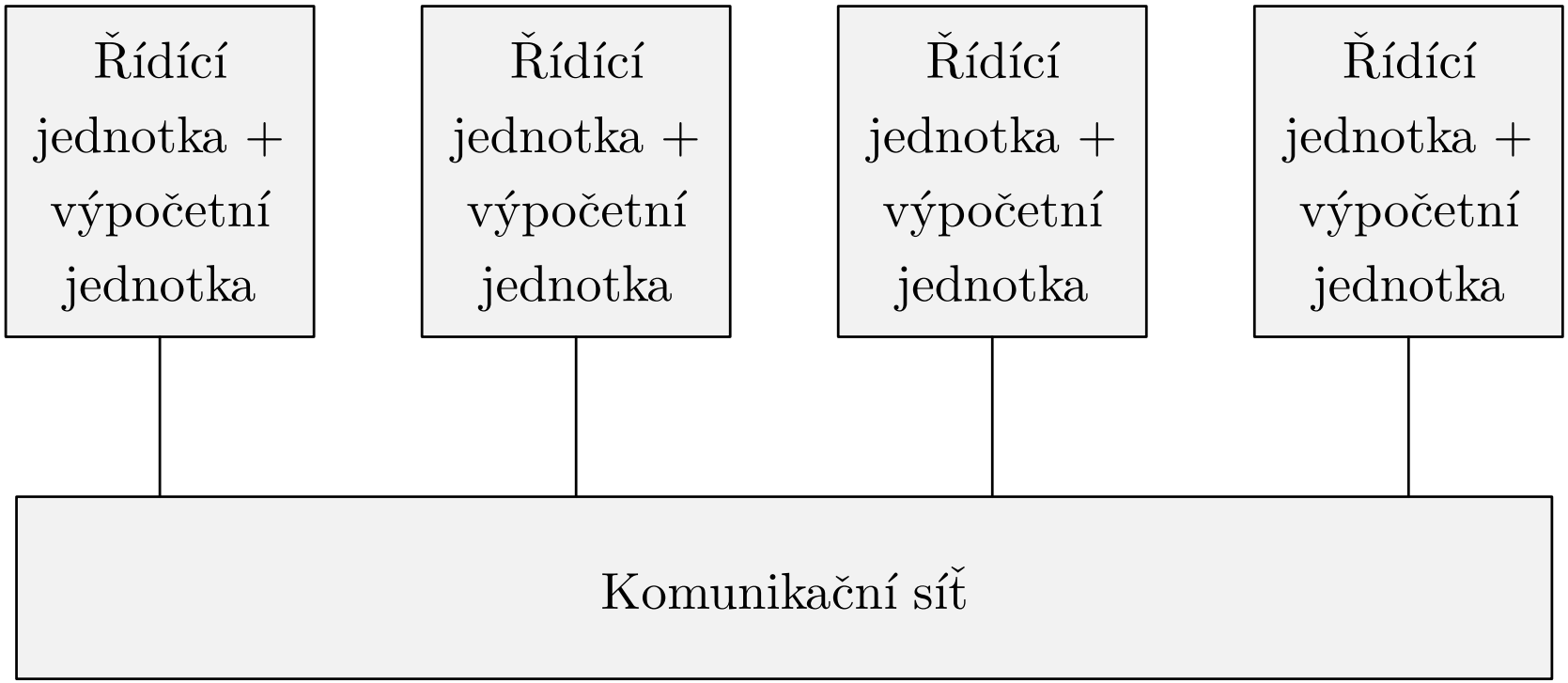


Flynnova taxonomie - MIMD I.

MIMD

- jde o systémy, které jsou schopné současně provádět různé instrukce a různými daty
- typickým příkladem jsou multiprocesory a multipočítače

Flynnova taxonomie - MIMD II.



Flynnova taxonomie - MIMD III.

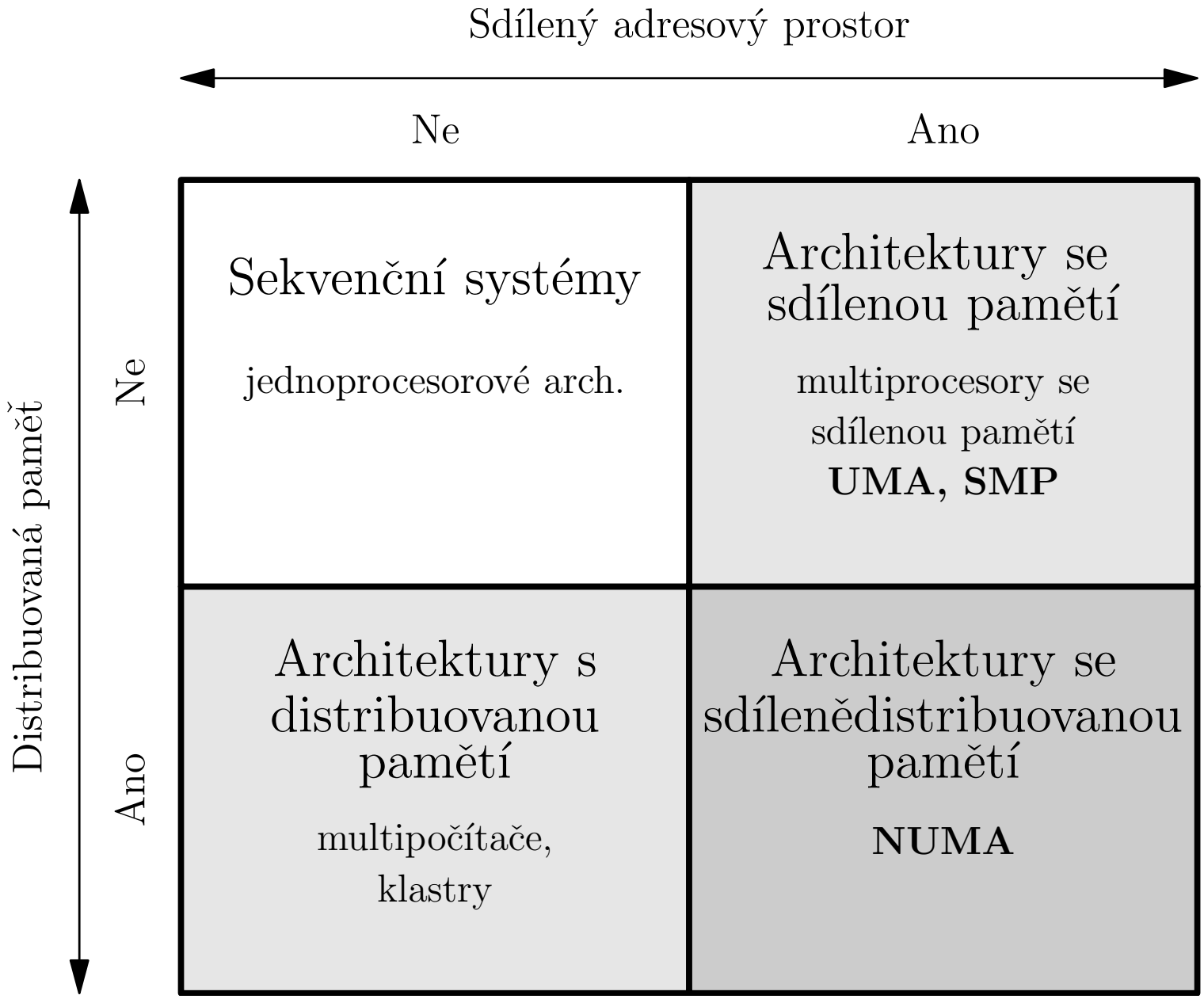
- v praxi nepíšeme zvláštní kód pro každý procesor
  - všude běží stejný program, ale podle ID procesoru se zpracovávají různé větve
- mluvíme pak spíše o **SPMD** architektuře
  - SPMD = single program multiple data
- pratkicky všechny významné paralelní architektury dnes spadají do MIMD kategorie
  - to je velká nevýhoda Flynnovy taxonomie

Komunikační modely I.

Podle způsobu komunikace dělíme paralelní architektury na:

- systémy se sdíleným adresovým prostorem
- systémy s distribuovanou pamětí
- systémy se sdíleně distribuovanou

Komunikační modely II.



Architektury se sdílenou pamětí

- obsahují fyzicky sdílenou paměť, do které mají všechny procesory (výpočetní jednotky) stejně rychlý přístup
  - jde o UMA architektury
- stejná adresa na různých procesorech odkazuje na stejnou fyzickou paměťovou buňku
- sdílená paměť je tu prostředkem komunikace
- kromě sdílené paměti mohou být jednotlivé procesory vlastní lokální paměť - *cache*
  - ta bývá mnohem rychlejší ...
  - ... ale není přístupná ostatním procesorům, nejde tedy o NUMA architekturu
- typickým příkladem je SMP - symetrický multiprocesing
- sdílená paměť se stává úzkým hrdlem celého systému, proto se tyto architektury omezují na maximálně 100 procesorů
- pro vývoj se často používá standard OpenMP

Architektury s distribuovanou pamětí

- nemají společnou paměť ani virtuální adresový prostor
- komunikují spolu pomocí posílání zpráv přes komunikační síť
- to je náročnější z pohledu programátora
- odstranění společné paměti umožňuje vytvářet systémy s tisíci procesory
- pro vývoj se často používá standard MPI

Architektury se sdílenědistribuovanou pamětí

- jde o architektury s distribuovanou pamětí, které mají podporu pro sdílený virtuální adresový prostor
- jde o NUMA architektury
- podpora pro virtuální adresový prostor bývá zabudováná již na úrovni hardware

Sdílený adresový prostor vs. posílání zpráv

- programování založené na posílání zpráv je náročnější
- posílání zpráv lze snadno a efektivně emulovat na systémech se sdílenou pamětí
  - programy napsané pomocí standardu MPI dobře běží i na SMP systémech
- neplatí to naopak

Přehled I.

- paralelní počítače se konstruují od poloviny šedesátých let
- vysoká cena integrovaných obvodů a tedy i řídících jednotek vedla ke stavění procesorvých polí, která obsahovala velký počet zjednodušených výpočetních jednotek
- s rozvojem VLSI obvodů klesala cena procesorů, což vedlo k rostoucí oblibě víceprocesorových systémů - 1980
- ty byly výhodnější pro zpracování kódu s podmínkami a mohlo na nich pracovat více uživatelů
- u systému se 100 a více procesory se naráží na potíže s paměťovým systémem
  - problém *cache coherence* a rovnocenné propojení procesorů s paměťovými moduly je náročné

Přehled II.

- tento problém řeší architektury s distribuovanou pamětí
- s rostoucím výkonem běžných PC a síťových komponent (Ethernet) spolu s klesajícími cenami těchto zařízení došlo v devadesátých letech k velkému příklonu k distribuovaným systémům
- po roce 2000 se ukazuje, že je obtížné nadále zvyšovat výkonu jednoprocesorových systémů zvyšováním frekvence CPU
  - ještě asi v roce 2002 IBM odhadovalo, že v roce 2010 budou mainframy používat 10 GHz procesory
- hledají se cesty, jak zvýšit výkon jednoho PC

Přehled III.

- jedním řešením jsou vícejádrové procesory = **SMP** ($\rightarrow$1980)
  - se současnou architekturou údajně Intel/AMD vystačí do maximálně 16 jader
- druhým řešením je GPGPU = **vektorový počítač** ($\rightarrow$1965)
- trend současnosti je CUDA a propojování více GPU dohromady (SLI)
- současné síťové prvky nejsou dostatečně rychlé pro stavění klastrů z GPU
- to lze očekávat zřejmě s nástupem - 10/40/100 Gb Ethernetu

Přehled IV.

- vedle tohoto směru se rozvíjí technologie gridu
  - dnes jsou snahy využít grid i k numerickému počítání (asynchronní iterativní maticové řešiče)
- z *grid computingu* vzniká tzv. *cloud computing* (2007) apod.