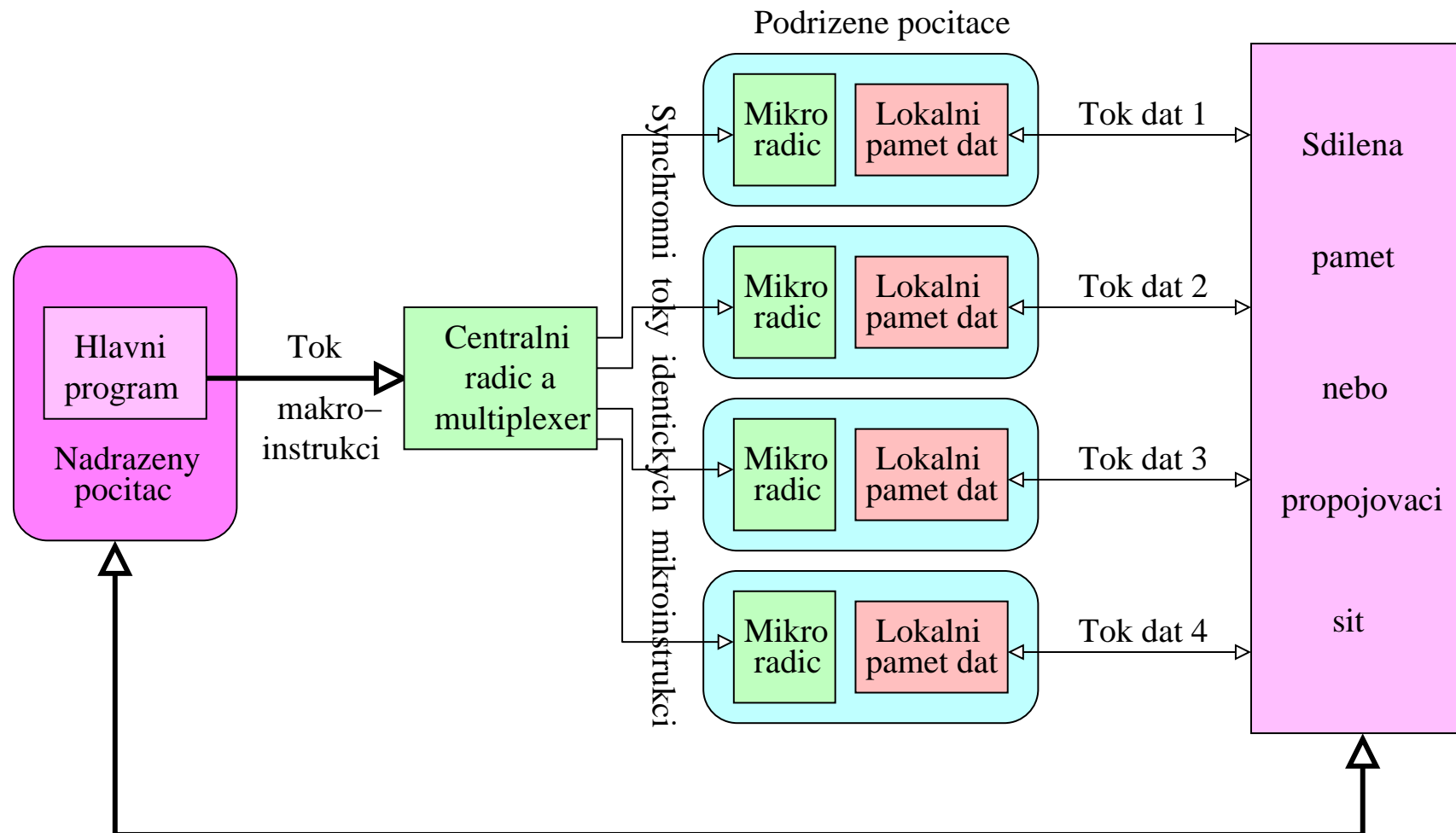


## Přednáška #3: Paralelní architektury a modely

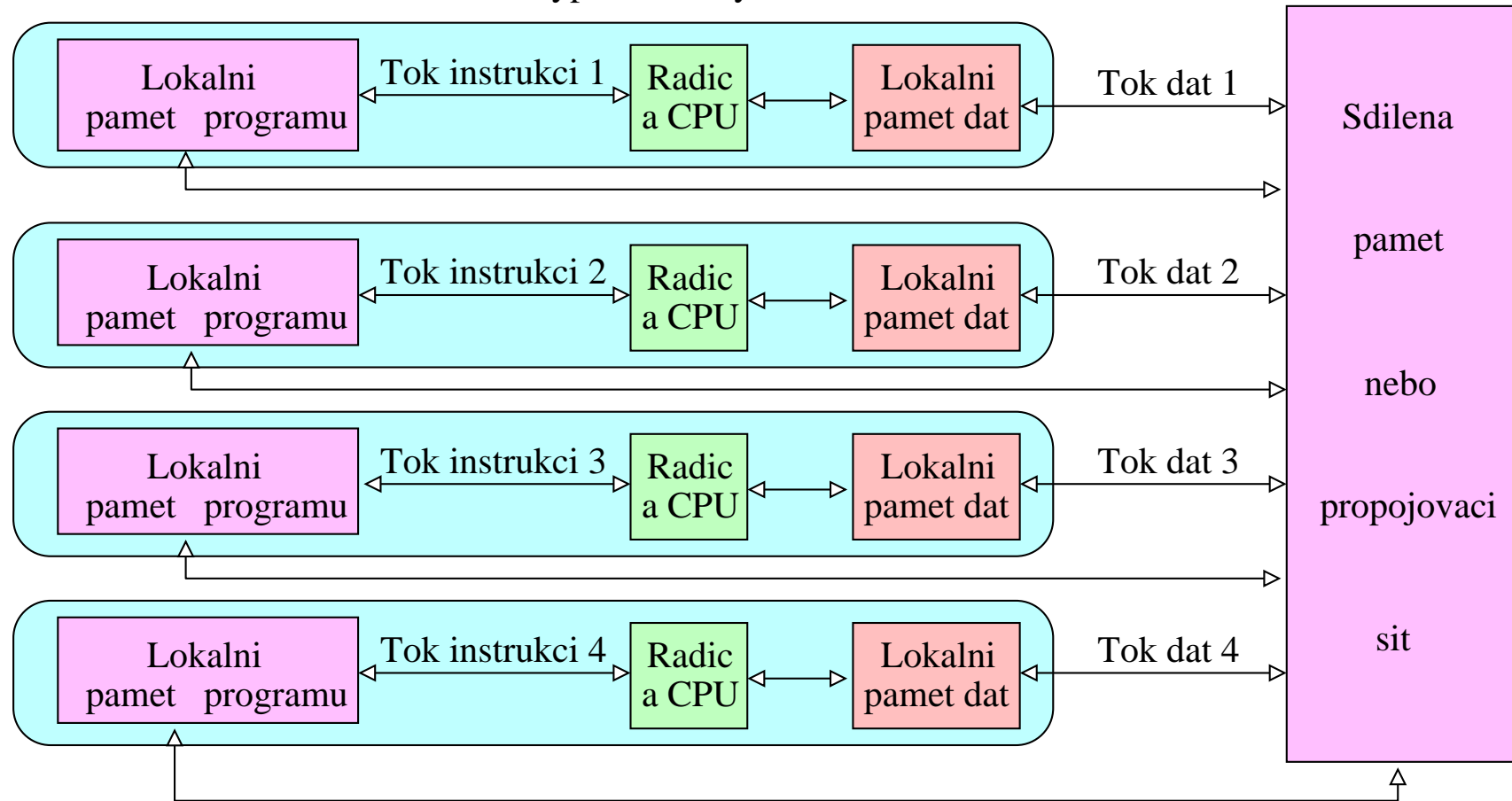
### Taxonomie paralelních architektur

#### Taxonomie z hlediska toků instrukcí a dat

##### SIMD



Vypocetni uzly



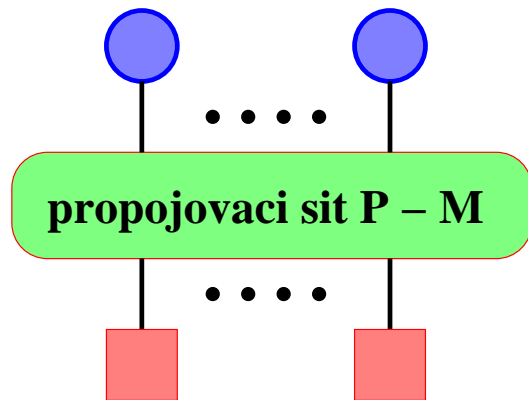
- rozdílná výpočetní síla uzlů
- **implicitní** (SIMD) vs. **explicitní** (MIMD) synchronizace (bariérová)
- hybrid MIMD a SIMD = **SMIMD** = **synchronizovaný** MIMD (např. TMC CM-5)
- **SPMD** (Single Program Multiple Data): programování nad **paralelními datovými strukturami**
  - ⇒ **datově paralelní jazyky**, např. HPF, OpenMP.
  - Konstrukce pro **paralelní pole** a jejich **distribuci** mezi procesory
- SIMD a MIMD se liší v paralelním provádění podmíněných příkazů typu if a case:
 

```
if podmínka then instr-posloup-A else instr-posloup-B
```
- SIMD a MIMD jsou **asymptoticky výpočetně ekvivalentní**:
  - MIMD může simulovat stejně velký SIMD: krokování
  - SIMD může simulovat stejně velký MIMD:
    - \* synchronní interpretace lokálních dat v SIMD pamětech jako MIMD programů
    - \* každý SIMD procesor má virtuální čítač ukazující do jeho MIMD programu
    - \* centrální řadič rozesílá cyklicky mikrokódy pro všechny MIMD instrukce
    - \* simulace 1 paralelního kroku MIMD trvá ( $\#$  MIMD instrukcí)-krát déle

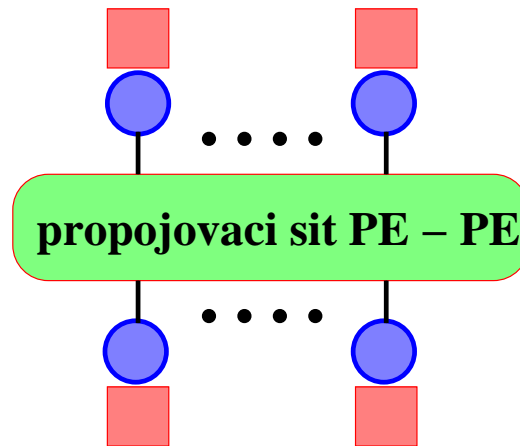
● = procesor (P)

■ = pamet (M)

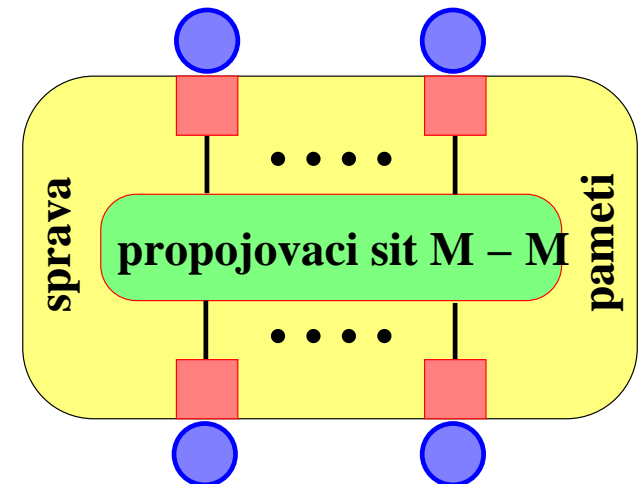
■● = vypocetni uzel (PE)



(1)



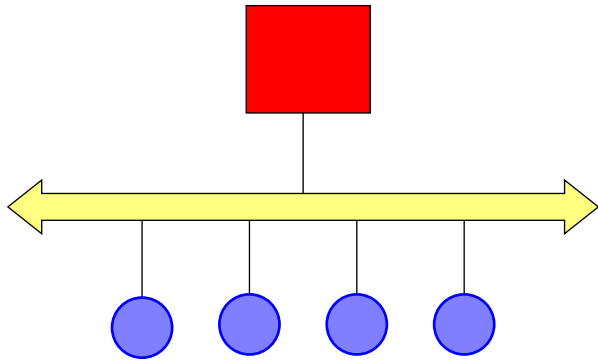
(2)



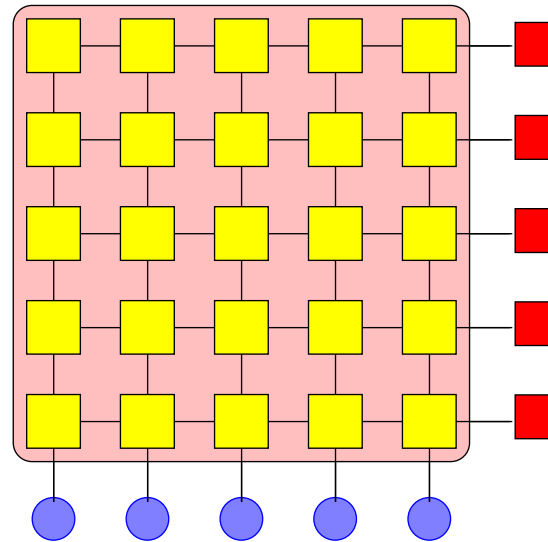
(3)

1. multiprocesorové systémy se **sdílenou pamětí**, symetrické multiprocesory (SMP, UMA)
  - HW/SW komunikace = **Read/Write**
2. multiprocesorové systémy s **distribuovanou pamětí** (NUMA)
  - HW/SW komunikace = **Send/Receive**
3. multiprocesorové systémy s **virtuálně sdílenou** (distribuovaně sdílenou) pamětí (CC-NUMA (Cache-Coherent))
  - HW komunikace = **Send/Receive**, SW komunikace = **Read/Write**

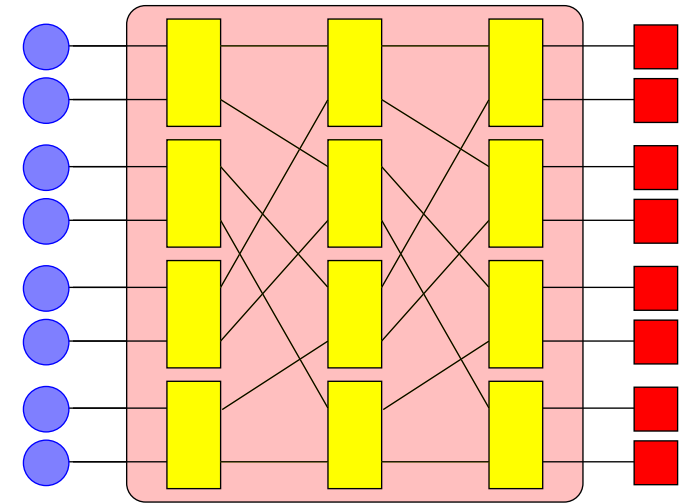
## Multiprocesorové systémy se sdílenou pamětí



(a)

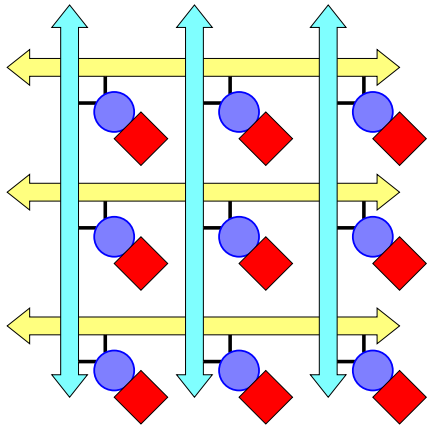


(b)

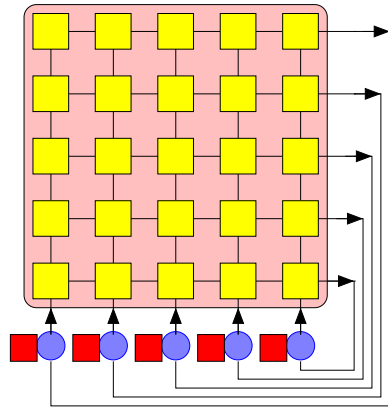


(c)

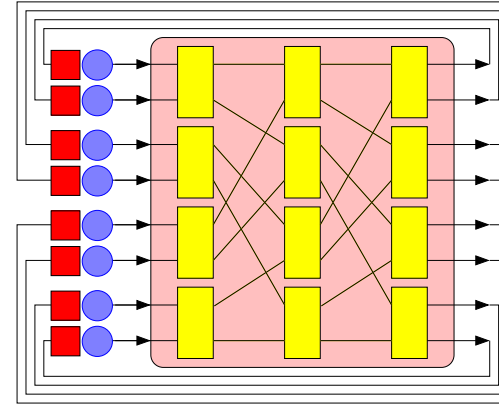
- (a) multiprocesorová sběrnice
- (b) křížový přepínač
- (c) nepřímá propojovací síť (MIN)



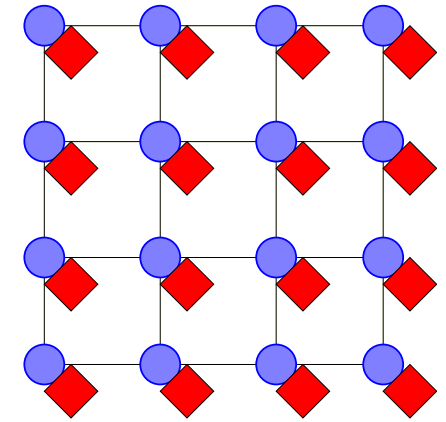
(a)



(b)



(c)



(d)

- (a) 2-D mřížka multiprocesorových sběrnic
- (b) křížový přepínač
- (c) vícestupňová nepřímá propojovací síť (MIN)
- (d) přímá propojovací síť (příští přednáška)

- Výpočetní jednotka s uživatelsky definovaným programem.
- Vstupní páska pro čtení a výstupní páska pro zápis.
- (Neomezený) počet paměťových buněk.
- Instrukční sada zahrnuje instrukce pro přesuny dat mezi paměťovými buňkami, srovnání, podmíněné větvení a aritmetické operace.
- Výpočet začne první instrukcí a končí po provedení instrukce HALT.
- Všechny instrukce trvají **jednotkový** (**konstantní**) čas bez ohledu na délku operandů.
- **Časová složitost algoritmu** = # provedených instrukcí.
- **Prostorová složitost algoritmu** = # použitých paměťových buněk.

- (Neomezený) # procesorů RAM označených  $P_1, P_2, P_3, \dots$  (bez pásek).
- (Neomezený) # sdílených paměťových buněk  $M[1], M[2], M[3], \dots$
- Každý  $P_i$  má vlastní lokální paměť (registry) a zná svůj index  $i$ .
- Každý  $P_i$  může přistupovat do kterékoli buňky sdílené paměti v konstantním čase (řešení případných konfliktů, viz dále).
- Vstup PRAM algoritmu:  $n$  položek v (obvykle prvních)  $n$  buňkách sdílené paměti.
- Výstup PRAM algoritmu:  $n'$  položek v  $n'$  buňkách sdílené paměti.
- PRAM procesory provádějí synchronně 3 typy instrukcí:
  1. Čtení buňky sdílené paměti (READ, krátce R).
  2. Lokální operace (LOCAL, krátce L).
  3. Zápis do buňky sdílené paměti (WRITE, krátce W).
- Jediný způsob komunikace procesorů = READ/WRITE buněk sdílené paměti.



- $P_1$  má speciální **aktivační registr**  $R_A$ , obsahující bit Active/Idle pro každý  $P_i$
- Výpočet trvá, dokud se  $P_1$  nezastaví (  $\implies$  ostatní  $P_i$  skončily).
- PRAM výpočty jsou specifikovány 2 parametry  $(n, p)$ :  $p = \#$  aktivních procesorů,  $n$  je prostorová složitost =  $\#$  použitých buněk sdílené paměti (vstupní, výstupní a pomocná sdílená data).
- **Paralelní časová složitost** = čas výpočtu na  $P_1$ .
  - **Jednotkový model**: **READ**, **WRITE**, **LOCAL** trvají čas 1.
  - **Globální model**: **READ**, **WRITE** trvají čas  $d > 1$  a **LOCAL** trvá čas 1.

### Význam PRAM modelu

- Je **silný**: **jakýkoli**  $P$  může provést **READ/WRITE** **jakékoli** buňky sdíl. paměti v konst. čase.
- Je **jednoduchý a intuitivní**: zanedbává jakoukoli režii na komunikaci či synchronizaci, což zjednodušuje analýzu složitosti a správnosti PRAM algoritmů. Tudiž:
- Je **užitečný**: je idealizací existujících (a běžných) par. počítačů se sdílenou pamětí.
- Slouží jako **zkušební model**: jestliže nějaký problém nemá rozumné/efektivní řešení na PRAM, nemá rozumné/efektivní řešení na **jakémkoli jiném** paralelním stroji.

**Exclusive Read Exclusive Write (EREW) PRAM:** Žádným dvěma procesorům není dovoleno READ nebo WRITE do téže buňky sdílené paměti najednou.

**Concurrent Read Exclusive Write (CREW) PRAM:** Současná čtení téže buňky paměti jsou dovolena, ale v 1 okamžiku se pouze 1 procesor smí pokusit zapsat do dané buňky.

**Concurrent Read Concurrent Write (CRCW) PRAM:** Jsou dovoleny jak současná čtení tak současné zápisy téže paměťové buňky.

- **Prioritní (Priority) CRCW:** Procesorům jsou přiděleny pevné priority. Zápis je povolen procesoru s **nejvyšší** prioritou.
- **Náhodný (Arbitrary) CRCW:** Ukončit zápis je povoleno **náhodně** vybranému procesoru. Algoritmus nesmí činit žádné předpoklady o tom, který procesor byl vybrán.
- **Shodný (Common) CRCW:** Všem žádajícím procesorům je povoleno dokončit zápis právě tehdy, když všechny zapisované **hodnoty jsou stejné**. Každý algoritmus pro tento model musí zajistit splnění této podmínky. V opačném případě není algoritmus správný a stav počítače není definován.

$p$ -procesorový PRAM,  $p < n$ , sdílené pole  $n$  neseřazených položek,  $P_1$  hledá hodnotu  $h$ .

1. **EREW-PRAM**( $n + p + 1, p$ ) **algoritmus**:

- (a)  $P_1$  distribuuje hodnotu  $h$  procesorům  $P_2, \dots, P_p$  pomocí **binárního distribučního stromu**:  $\langle \text{RW} \rangle^{\lceil \log p \rceil}$  ( $\Theta(\log p)$  kroků).
- (b) Paralelní lokální hledání:  $\langle \text{RL} \rangle^{\lceil \frac{n}{p} \rceil}$  ( $O(n/p)$  kroků).
- (c) Všechny procesory nastaví hodnotu svého příznaku **Nalezeno** a provedou **paralelní redukci**:  $\langle \text{WR} \rangle^{\lceil \log p \rceil}$  ( $\Theta(\log p)$  kroků).

$$T(n, p) = O(\log p + n/p).$$

2. **CREW PRAM**( $n + p + 1, p$ ) **algoritmus**: Podobné, pouze  $P_1, \dots, P_p$  si přečtou  $h$  najednou v  $O(1)$  čase.

Nicméně, redukce na konci trvá stejně  $\Theta(\log p)$  kroků, proto

$$T(n, p) = O(\log p + n/p).$$

3. **Shodný-CRCW PRAM**( $n + 2, p$ ) **algoritmus**: Redukce trvá nyní také  $O(1)$  čas.

Všechny procesory s **Nalezeno**=1 zapíšou 1 do sdílené buňky přidružené procesoru  $P_1$ .

$$T(n, p) = O(n/p).$$

**Definice 1.** PRAM podmodel  $A$  je *výpočetně silnější* než podmodel  $B$ , psáno  $A \geq B$ , jestliže jakýkoliv algoritmus napsaný pro  $B$  poběží *beze změny* na  $A$  s tímtež paralelním časem, předpokládáme-li stejné architektonické parametry. ♣

**Lemma 2.**  $\text{PRIORITY} \geq \text{ARBITRARY} \geq \text{COMMON} \geq \text{CREW} \geq \text{EREW}$ .

**Důkaz.** Plyne z definic. ♣

**Definice 3.** *Nechť  $K$  je problém se vstupní množinou o velikosti  $n$ . Předpokládejme, že  $K$  lze řešit  $PRAM(n, p)$  algoritmem  $A$  v čase  $T(n, p)$ . Pak*

1.  $A$  je **časově efektivní**, jestliže

(a)  $T(n, p) = O(\log^{O(1)} n) (= POLYLOG(n))$  a

(b)  $C(n, p) = O(SU(n) \log^{O(1)} n)$ .

2.  $A$  je **cenově optimální a časově efektivní**, jestliže

(a)  $T(n, p) = O(\log^{O(1)} n)$  a

(b)  $C(n, p) = pT(n, p) = O(SU(n))$ .

3.  $A$  je **plně paralelní**, jestliže

(a)  $T(n, p) = O(1)$  a

(b)  $C(n, p) = O(SU(n))$  (což je ekvivalentní tomu, že  $p = O(SU(n))$ ).

1. Plně paralelní Shodný-CRCW PRAM( $n + 1, n$ ) algoritmus pro výpočet  $OR(x_1, \dots, x_n)$ ,  $p = n$ .

**Algorithm** OR( $n + 1, n$ )

---

**In:** boolean  $X[1, \dots, n]$ ; **Out:** boolean  $y$ ;

---

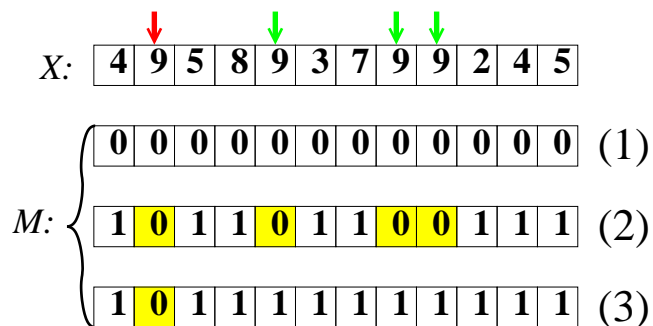
(1)  $P_1$  writes 0 into  $y$ ;(2) **for all**  $P_i, i = 1, \dots, n$ , **do\_in\_parallel**    **if** ( $X[i] = 1$ ) **then** write 1 into  $y$ ;

2. Shodný-CRCW PRAM( $2n + 2, n^2$ ) algoritmus pro výpočet nejlevějšího maxima  $\max(x_1, \dots, x_n)$ ,  
 $p = n^2$  procesorů v mřížce indexovaných  $P_{i,j}$ ,  $i, j = 1, \dots, n$ , s časem  $T(n, p) = O(1)$ .

### Algorithm MAX( $2n + 2, n^2$ )

**In:** integer  $X[1, \dots, n]$ ; **Out:** (integer, integer) max; **Aux:** boolean  $M[1, \dots, n]$ ;

- (1) **for all**  $P_{i,1}$ ,  $i = 1, \dots, n$ , **do\_in\_parallel** write 0 into  $M[i]$ ;
- (2) **for all**  $P_{i,j}$ ,  $i, j = 1, \dots, n$ , **do\_in\_parallel**  
     **if** ( $X[i] < X[j]$ ) **then** write 1 into  $M[i]$ ;  
     (\* Zeros in  $M$  survived only at positions of maxima of  $X$  \*)
- (3) **for all**  $P_{i,j}$ ,  $i, j = 1, \dots, n$ , **do\_in\_parallel**  
     **if** ( $M[i] = 0$  **and**  $i < j$ ) **then** write 1 into  $M[j]$ ;  
     (\* Only the zero at the left-most position in  $M$  can survive \*)
- (4) **for all**  $P_{i,1}$ ,  $i = 1, \dots, n$ , **do\_in\_parallel**  
     **if** ( $M[i] = 0$ ) **then** write  $(i, X[i])$  into max;



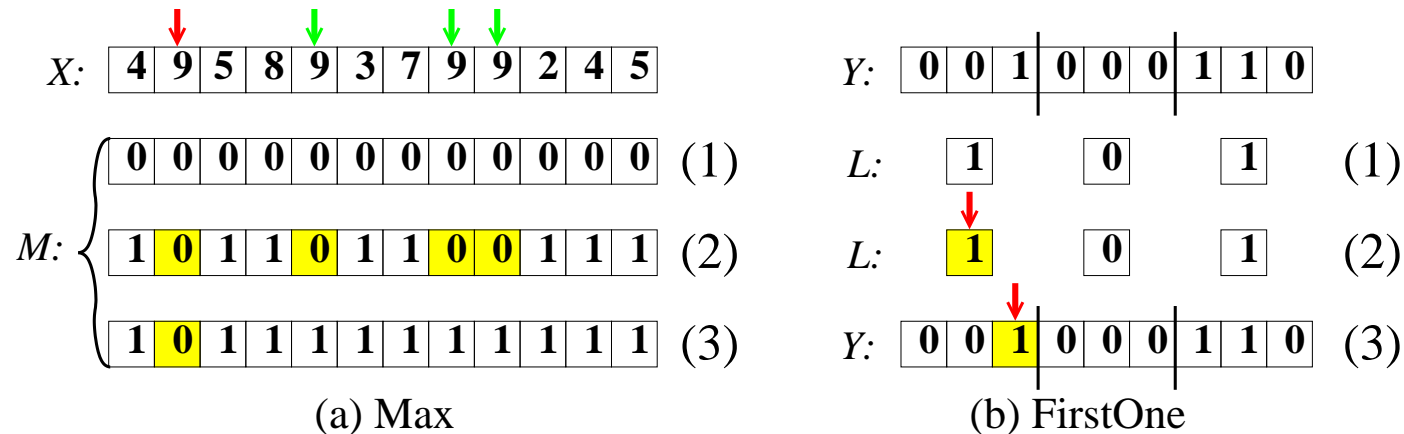
3. Plně paralelní Shodný-CRCW PRAM( $n + \sqrt{n} + 1, n$ ) algoritmus pro výpočet indexu prvního výskytu hodnoty 1 v binárním poli  $X$ ,  $p = n$ .

**Algorithm** FIRSTONE( $n + \sqrt{n} + 1, n$ )

**In:** boolean  $Y[1, \dots, n]$ ; **Out:** integer  $l$  = the index of the first value 1 in  $X$ ;

**Aux:** boolean  $L[1, \dots, \sqrt{n}] := [0, \dots, 0]$ ;

- (1) split the array  $Y$  into  $\sqrt{n}$  segments  $Y_i$  of size  $\sqrt{n}$ ;
- (2) **for all**  $Y_i$ ,  $i = 1, \dots, \sqrt{n}$ , **do\_in\_parallel**  
     apply alg. OR( $\sqrt{n} + 1, \sqrt{n}$ ) to  $Y_i$  and write 1 into  $L[i]$  if  $1 \in Y_i$ ;
- (3) **for all**  $P_i$ ,  $i = 1, \dots, n$ , **do\_in\_parallel**  
     apply alg. MAX( $2\sqrt{n} + 2, n$ ) to  $L$  and compute index  $k$  of the leftmost 1 in  $L$  ;
- (4) **for all**  $P_i$ ,  $i = 1, \dots, n$ , **do\_in\_parallel**  
     apply alg. MAX( $2\sqrt{n} + 2, n$ ) to  $Y_k$  and find index  $k$  of the leftmost 1 in  $Y_k$  ;





## Hrubší simulace zachovávající paralelní cenu

**Lemma 4.** *Nechť  $A$  je  $PRAM(n, p)$  algoritmus s časem  $t = T(n, p)$  a nechť  $p' < p$ . Pak  $A$  lze simulovat pomocí  $PRAM(n, p')$  algoritmu na PRAM téhož typu v čase  $T(n, p') = O(tp/p')$ .*

**Důkaz.**

1. Rozdělme  $p$  simulovaných procesorů do  $p'$  skupin o velikostech  $p/p'$ .
2. Přiřadíme každému z  $p'$  simulujících procesorů jednu tuto skupinu.
3. Každý simulující procesor simuluje 1 krok své skupiny procesorů:
  - (a) provedením nejdříve všech jejích operací READ a lokálních výpočtů,
  - (b) po té provedením jejích operací WRITE.



## Důsledky

**Důsledek 5.** *Každý  $PRAM(n, p)$  algoritmus s cenou  $C(n, p)$  lze provést sekvenčně v čase  $t = O(C(n, p))$ .*



**Důsledek 6.** *Pokud jsme vyvinuli  $PRAM(n, p)$  algoritmus s cenou  $C(n, p) = o(SU(n))$ , pak jsme automaticky vyvinuli nový nejlepší sekvenční algoritmus.*



**Brentův plánovací princip**

**Důsledek 7.** *Nechť  $A$  je PRAM( $n, p$ ) algoritmus s prací  $w = W(n, p)$  a časem  $t = T(n, p)$  a nechť  $p' < p$ . Pak  $A$  lze simulovat pomocí PRAM( $n, p'$ ) algoritmu na PRAM téhož typu v čase*

$$t' = T(n, p') \leq w/p' + t.$$

**Důkaz.** Simulace je podobná předchozí, ale dynamická.

- Podle definice paralelní práce,  $w = \sum_{i=1}^t p_i$ ,  
kde  $p_i = \#$  aktivních procesorů v kroku  $i$  algoritmu  $A$ .
- Každý z  $p'$  procesorů bude simulovat krok  $i$  v nejvýše  $t'_i = \lceil p_i/p' \rceil \leq p_i/p' + 1$  krocích.
- Celkový čas simulace je nejvýše

$$t' = \sum_{i=1}^t t'_i \leq \sum_{i=1}^t (p_i/p' + 1) = w/p' + t.$$

Protože simulující procesory provedou přesně a pouze operace  $A$ , práce zůstává stejná:  $w' = w$ . Cena  $c'$  simulace je

$$c' = p't' \leq w + p't.$$

Pokud  $p' = O(\min_j p_j)$ , pak  $c' = O(w)$ .



**Věta 8.** *Nechť  $A$  je  $PRAM(n, p)$  algoritmus s časem  $t$  a nechť  $m < n$ . Pak  $A$  lze simulovat  $PRAM(m, \max(p, m))$  algoritmem na PRAM téhož typu v čase  $t' = O(tn/m)$ .*

*Předpokládáme, že simulující procesory mají lokální paměti dostatečně velké pro uložení sdílených dat.*

**Důkaz.**

1. Rozdělme  $n$  simulovaných sdílených paměťových buněk do  $m$  souvislých úseků  $S_i$  o velikosti  $n/m$ .
2. Každý simulující procesor  $P'_i$ ,  $1 \leq i \leq p$ , bude simulovat procesor  $P_i$  původního PRAM.
3. Každý simulující procesor  $P'_i$ ,  $1 \leq i \leq m$ , uloží počáteční obsah  $S_i$  do své lokální paměti a bude používat  $M'[i]$  jako pomocnou paměťovou buňku pro simulování přístupů k buňkám  $S_i$ .
4. Simulace jedné původní operace READ:  
Každý  $P'_i$ ,  $i = 1, \dots, \max(p, m)$  opakuje pro  $k = 1, \dots, n/m$ :
  - (a) Proved' WRITE hodnoty  $k$ -té buňky  $S_i$  do  $M'[i]$ ,  $i = 1, \dots, m$ ,
  - (b) Proved' READ hodnoty, kterou by simulovaný procesor  $P_i$ ,  $i = 1, \dots, p$ , četl v tomto simulovaném podkroku.
5. Lokální krok procesoru  $P_i$ ,  $i = 1, \dots, p$ , je simulován procesorem  $P'_i$  v 1 kroku.
6. Simulace jedné operace WRITE je analogická simulaci operace READ.

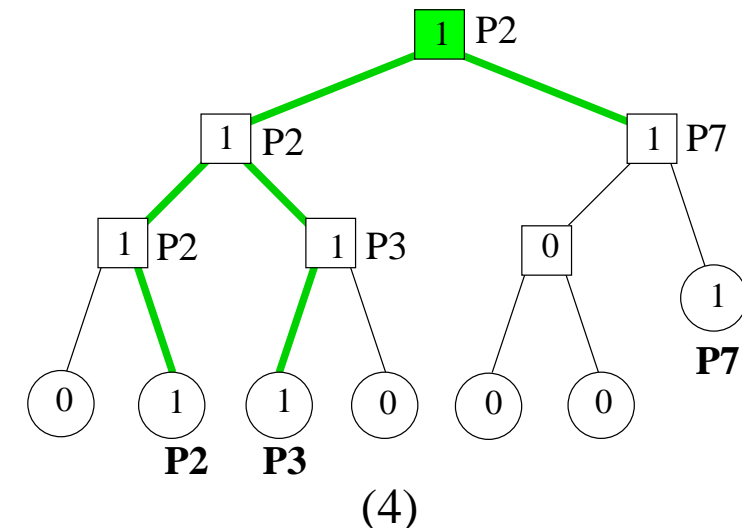
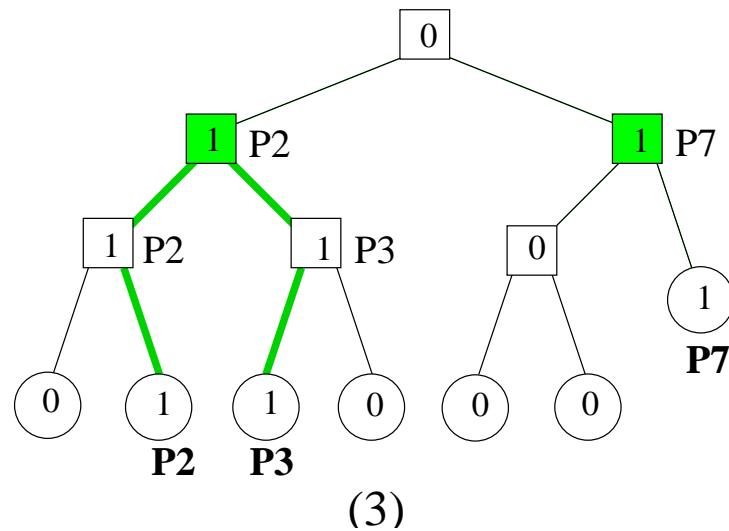
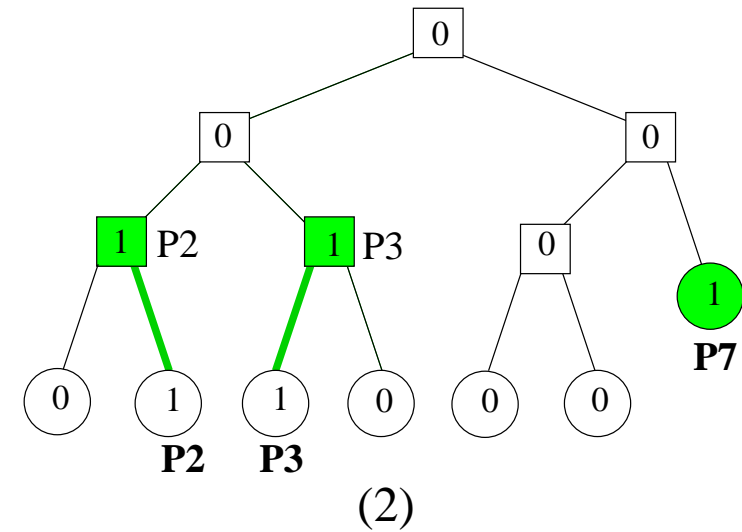
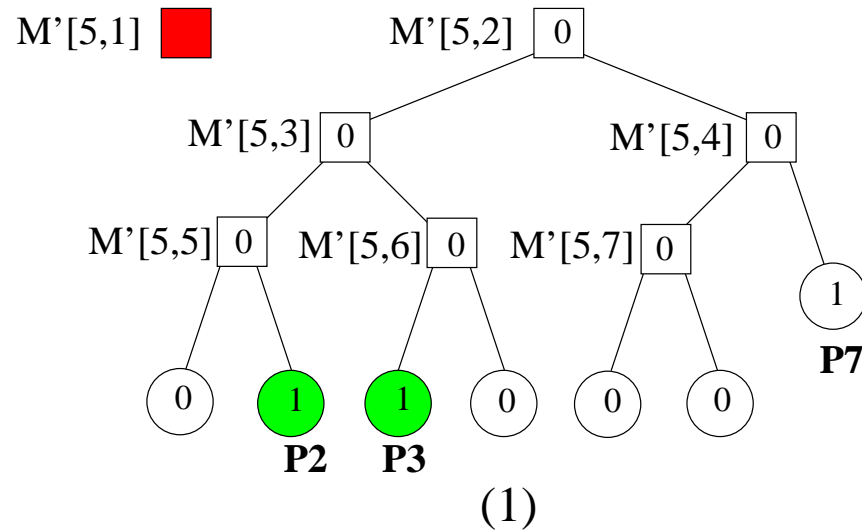


**Věta 9.** *Uvažujme Prioritní-CRCW s prioritním systémem založeným jednoduše na indexování: procesory s nižším indexem mají vyšší prioritu. Jeden krok Prioritní-CRCW-PRAM( $n, p$ ) algoritmu lze simulovat pomocí EREW-PRAM( $np, p$ ) algoritmu v  $O(\log p)$  krocích.*

**Důkaz.** (Konstruktivní.)

- Každý procesor  $P_k$  v Prioritní-CRCW je simulován EREW procesorem  $P'_k$ .
- Každá buňka sdílené paměti  $M[i]$ ,  $i = 1, \dots, m$ , v Prioritní-CRCW je simulována **polem**  $p$  buněk sdílené paměti  $M'[i, k]$ ,  $k = 1, \dots, p$ , na EREW.
- $M'[i, 1]$  hraje roli  $M[i]$ .
- $M'[i, 2], \dots, M'[i, p]$  jsou **pomocné buňky** potřebné pro ošetření konfliktů.
- Na počátku jsou **prázdné** a jsou organizované jako **vnitřní uzly úplného binárního stromu**  $T_i$  s  $p$  listy.
- Výška každého stromu  $T_i$  je  $\lceil \log p \rceil$ .

3 kroky simulace pro  $p = 7$ , kdy procesory  $P_2, P_3$  a  $P_7$  chtějí zapsat do  $M[5]$ .



**Simulace kroku Prioritní-WRITE:** Každý EREW procesor musí zjistit, zda je procesorem s nejmenším indexem v rámci skupiny procesorů, žádajících zápis do téže buňky. A pokud ano, pak se musí stát **vítězem** skupiny a provést operaci WRITE. Ostatní procesory ve skupině skončí neúspěchem. Postup je následující:

1. Pokud  $P_k$  chce zapisovat do  $M[i]$ , procesor  $P'_k$  se stane **aktivní** a stane se  $k$ -tým listem  $T_i$ . Ví, zda je pravým či levým potomkem svého rodiče.
2. Každý aktivní **levý** procesor uloží své ID do rodičovské buňky ve svém stromě, označí ji jako **obsazenou** a zůstane aktivní.
3. Každý aktivní **pravý** procesor zkontroluje svou rodičovskou buňku. Je-li prázdná, uloží do ní své ID a zůstane aktivní. Je-li obsazená, přepne se do stavu nečinný (prohrál).
4. Toto se opakuje  $\log p$  krát na dalších hladinách stromu.
5. Procesor, kterému se podařilo postoupit až do kořene  $T_i$ , se stává **vítězem**, který může zapsat do  $M[i]$ . Procesory, které používaly  $T_i$ , musí pak projít strom  $T_i$  dolů v opačném pořadí a nastavit příslušné buňky pole  $M'[i, 2], \dots, M'[i, p]$  na stav prázdný.

**Simulace kroku Prioritní-READ:** je podobná.

1. Paralelně se provedou stejné průchody stromy  $T_i$  nahoru, aby se určili vítězové ve skupinách.
2. Vítězové přečtou hodnoty z buněk  $M'[* , 1]$ .
3. Během vyčišťovacího zpětného průchodu dolů je načtená hodnota distribuována do procesorů, které v části (a) prohrály.



**Lemma 10.** *Jeden krok Prioritní-CRCW-PRAM( $n, p$ ) algoritmu lze simulovat pomocí EREW-PRAM( $n + 3p, p$ ) algoritmu v  $O(\log p)$  krocích.*

**Důkaz.** (Konstruktivní.)

1. Každý procesor  $P_k$  v Prioritní-CRCW je simulován EREW procesorem  $P'_k$ .
2. Každá buňka  $M[i]$  v Prioritní-CRCW je simulována EREW buňkou  $M[i]$ .
3. EREW používá **pomocné** pole  $A[1, \dots, p][1, 2, 3]$ .
4. Chce-li  $P_k$  přístup do  $M[i]$ , pak procesor  $P'_k$  zapíše dvojici  $(i, k)$  do  $A[k][1, 2]$ .  
Pokud  $P_k$  nechce přístup nikam, pak procesor  $P'_k$  zapíše  $(0, k)$  do  $A[k][1, 2]$ .
5. Všech  $p$  procesorů setřídí pole  $A$  v lexikografickém pořadí použitím paralelního třídění v  $O(\log p)$  krocích.
6. Každý  $P'_k$  zapíše do  $A[k][3]$  příznak  $f$ :
 
$$f = \begin{cases} 0, & \text{je-li } A[k][1] = 0 \\ & \text{nebo } A[k][1] = A[k-1][1], \\ 1, & \text{jinak.} \end{cases}$$
7. Další kroky se liší podle toho, zda simulujeme WRITE nebo READ.



## ■ Simulace WRITE:

1. Každý  $P'_k$  přečte trojici  $(i, j, f)$  z buňky  $A[k]$  a zapíše ji do  $A[j]$ .
2. Každý  $P'_k$  přečte trojici  $(i, k, f)$  z buňky  $A[k]$  a jestliže  $f = 1$ , pak zapíše do  $M[i]$ .

## ■ Simulace READ:

1. Každý  $P'_k$  přečte trojici  $(i, j, f)$  z buňky  $A[k]$ .
2. Pokud  $f = 1$ , přečte hodnotu  $v_i$  z  $M[i]$  a přepíše s ní  $A[k][3]$ .
3. V nejvýše  $\log p$  krocích je pak tato 3. složka **rozkopírovaná** do následujících buněk pole  $A$ , dokud se nenarazí na **konec** pole  $A$  nebo na položku s **jinou první** složkou.

```

for all  $k := 1, \dots, p$  do_in_parallel
  for  $l := 0, \dots, \log p - 1$  do_sequentially
    {  $P'_k$  READs its cell  $A[k] = (i, j, s)$ ;
      if  $((s <> 0) \text{ and } (k + 2^l \leq p))$ 
        then {  $P'_k$  READs cell  $A[k + 2^l] = (i', j', s')$ ;
          if  $((s' = 0) \text{ and } (i = i'))$ 
            then  $P'_k$  WRITEs  $s = v_i$  into  $A[k + 2^l][3]$ .
        }
    }
```

4. Každý  $P'_k$  přečte trojici  $(i, j, v_i)$  z buňky  $A[k]$  a zapíše do  $A[j]$ .
5. Každý  $P'_k$ , který žádal o READ, si přečte hodnotu  $v_i$  z trojice  $(i, k, v_i)$  v buňce  $A[k]$ .

Předpokládejme, že  $p = 7$ ,  $m = 4$ , a

$P_1 \rightarrow M[2]$ ,  $P_2 \rightarrow M[4]$ ,  $P_3 \rightarrow M[2]$ ,  $P_4 \rightarrow M[1]$ ,  $P_5 \rightarrow M[4]$ ,  $P_6 \rightarrow M[2]$ ,

a  $P_7$  nechce přístup do žádné buňky.

Pole  $A$  v prvních třech krocích simulace:

(2, 1, )	(4, 2, )	(2, 3, )	(1, 4, )	(4, 5, )	(2, 6, )	(0, 7, )
(0, 7, )	(1, 4, )	(2, 1, )	(2, 3, )	(2, 6, )	(4, 2, )	(4, 5, )
(0, 7, 0)	(1, 4, 1)	(2, 1, 1)	(2, 3, 0)	(2, 6, 0)	(4, 2, 1)	(4, 5, 0)

Pole  $A$  při simulaci operace WRITE:

(2, 1, 1)	(4, 2, 1)	(2, 3, 0)	(1, 4, 1)	(4, 5, 0)	(2, 6, 0)	(0, 7, 0)
-----------	-----------	-----------	-----------	-----------	-----------	-----------

Pole  $A$  při simulaci operace READ:

(0, 7, 0)	(1, 4, $v_1$ )	(2, 1, $v_2$ )	(2, 3, 0)	(2, 6, 0)	(4, 2, $v_4$ )	(4, 5, 0)
(0, 7, 0)	(1, 4, $v_1$ )	(2, 1, $v_2$ )	(2, 3, $v_2$ )	(2, 6, 0)	(4, 2, $v_4$ )	(4, 5, $v_4$ )
(0, 7, 0)	(1, 4, $v_1$ )	(2, 1, $v_2$ )	(2, 3, $v_2$ )	(2, 6, $v_2$ )	(4, 2, $v_4$ )	(4, 5, $v_4$ )
(2, 1, $v_2$ )	(4, 2, $v_4$ )	(2, 3, $v_2$ )	(1, 4, $v_1$ )	(4, 5, $v_4$ )	(2, 6, $v_2$ )	(0, 7, 0)

### Důsledek

Každý PRAM algoritmus s polylogaritickým časem je vzhledem k PRAM modelům **robustní**. Tedy, má-li polylogaritický čas na jednom modelu PRAM, bude mít polylogaritický čas na kterémkoli jiném.

- Procesory pracují **asynchronně**, neexistují centrální hodiny.
- Operace globální čtení, globální zápis, lokální – jako PRAM.
- Je nutná **explicitní** synchronizace: **bariérová synchronizace**.
- Doba přístupu do sdílení paměti **není jednotková**.
- APRAM výpočet = posloupnost **globálních fází**, ve kterých procesory pracují asynchronně, oddělených bariérovou synchronizací.
- Dva procesory nemohou přistupovat do téže buňky sdílené paměti v téže globální fázi, pokud jeden z nich zapisuje.

lokální operace	1
globální READ nebo WRITE	$d$
$k$ po sobě jdoucích globálních READ nebo WRITE	$d + k - 1$
barierová synchronizace	$B(p)$

- $d$  a  $B$  jsou neklesajícími funkcemi  $p$ .
- Předpokládáme, že  $2 \leq d \leq B(p) \leq p$ , ale v praxi je  $B(p) = O(d \log p)$  nebo  $B(p) = O(d \log p / \log d)$ .
- Po sobě jdoucí globální R/Ws jsou zřetězeny (např. implementace pomocí sběrnice s rozdělenými transakcemi).

1. **Centrální čítač**, inicializovaný na 0 a na příchozí fázi, procesy přistupují ve vzájemném vyloučení.

- (a) Proces dorazí k bariéře, zkontroluje, zda je v příchozí fázi a inkrementuje čítač.
- (b) Je-li čítač  $< p$ , deaktivuje se.
- (c) Jinak nastaví bariéru do odchozí fáze a aktivuje ostatní procesy.
- (d) Poslední aktivovaný proces nastaví bariéru do příchozí fáze.
- (e)  $B(p) = \Theta(dp)$

2. **binární redukční strom**. Každý proces

- (a) dorazí k bariéře a zkontroluje, zda je v příchozí fázi,
- (b) čeká, až skončí redukce v jeho podstromu,
- (c) po jejím skončení pošle signál rodiči,
- (d) kořen stromu počká na redukci z obou podstromů a přepne do odchozí fáze,
- (e) procesy se aktivují ve zpětném pořadí.
- (f)  $B(p) = \Theta(d \log p)$

## Triviální simulace, která nezachovává operační složitost

**Lemma 11.** *Jeden PRAM cyklus  $\langle \text{RLW} \rangle$  lze na APRAM simulovat v čase  $2B(p) + 2d + 1 = O(B(p))$ .*

**Důkaz.** Vlož synchronizační bariéru za každou globální READ a za každou globální WRITE.



## Operace zachovávající operační složitost

**Věta 12.** *EREW PRAM algoritmus mající čas  $t$  při použití  $p$  procesorů lze simulovat na APRAM v čase  $O(B(p)t)$  s  $p/B(p)$  procesory.*

**Důkaz.** Bude odvozeno na prosemináři!!!



- Cenově optimální simulace cenově optimálního PRAM algoritmu.
- Paralelní redukce na PRAM:  $C(n, p) = \Theta(n)$  a  $T(n, p) = \Theta(\log n)$  pro  $p = \Theta(n / \log n)$ .
- Paralelní redukce na APRAM: optimální  $\#$  procesorů je

$$p' = \Theta \left( \frac{n}{B(\frac{n}{\log n}) \log n} \right).$$

**Nejrychlejší paralelní redukce na APRAM**

- $n$  čísel v paměti APRAM počítače s  $n$  procesory lze sečíst v čase

$$T(n, n) = O(B(n) \log_d n).$$

- $n$  čísel v paměti APRAM počítače lze sečíst cenově-optimálně v čase

$$T(n, p') = O(B(n) \log_d n).$$

**Důkaz.** Bude odvozeno na prosemináři!!!