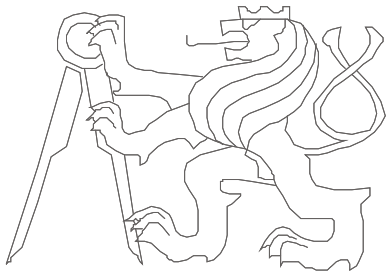
# Pokročilé architektury počítačů

Úvod – část I.

Paralelizmus na úrovni instrukcí, vláken, programů a dat; Časový a prostorový paralelizmus; Datová a řídicí závislost; Bernsteinovy podmínky paralelizmu;



České vysoké učení technické, Fakulta elektrotechnická

# Paralelizmus na úrovni instrukcí

- Paralelizmus na nejnižší úrovni – bit-level parallelism (šířka slova; součet dvou 64-bitových čísel v 32-bit mikroproc., zběrnice,..)
- Paralelizmus na úrovni instrukcí – Instruction level parallelism
  - Zřetězení (pipelining) – časový paralelizmus (sekvenční instr. proud)
  - Superskalární vykonávání (v širším smyslu) – prostorový paralelizmus

## Zřetězení:

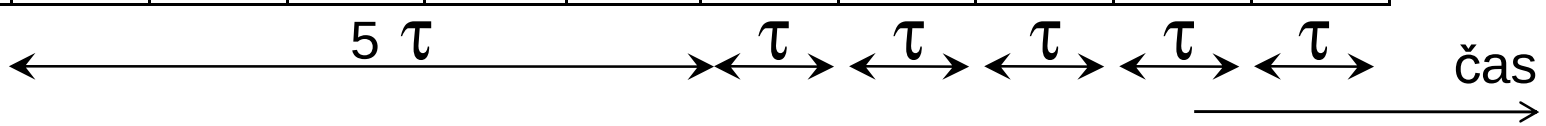
- Předpokládejme, že vykonání instrukce můžeme rozdělit do 5 stupňů:


IF – Instruction Fetch, ID – Instr. decode (and Operand Fetch),
MEM – Memory Access, EX – Execute, WB – Write Back

a dále $\tau = \max \{ \tau_i \}_{i=1}^{k}$, kde $\tau_i$ je čas šíření (*propagation delay) v i*-tém stupni.

# Paralelizmus na úrovni instrukcí - zřetězení

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| IF | I1 | I2 | I3 | I4 | I5 | I6 | I7 | I8 | I9 | I10 |
| ID | | I1 | I2 | I3 | I4 | I5 | I6 | I7 | I8 | I9 |
| EX | | | I1 | I2 | I3 | I4 | I5 | I6 | I7 | I8 |
| MEM | | | | I1 | I2 | I3 | I4 | I5 | I6 | I7 |
| ST | | | | | I1 | I2 | I3 | I4 | I5 | I6 |
| | | | | | | | | | | |
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |




- Čas vykonání *n* instrukcí *k*-stupňové pipeline:

$$T_k = k.\tau + (n-1)\,\tau$$

Předpoklad: ideálně vyvážená pipeline

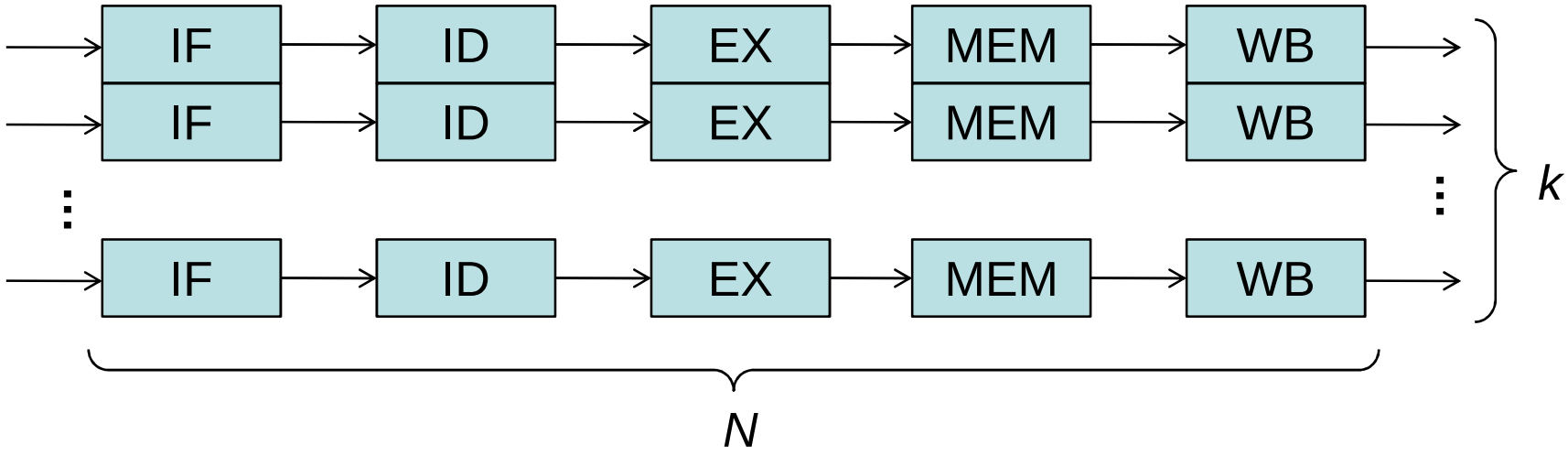
- Zrychlení: $S_k = \frac{T_1}{T_k} = \frac{nk\tau}{k\tau + (n-1)\tau}$ $\lim_{n\to\infty} S_k = k$

# Paralelizmus na úrovni instrukcí - zřetězení

- Neredukuje čas vykonání individuální instrukce, právě naopak..
- Hazardy:
  Strukturální (řešeny duplikací),
  datové (důsledek datových závislostí) a
  řídicí (instrukce měnící PC)...
- Hazardy způsobují (mohou způsobovat) pozastavení vykonávání (stall) nebo vyprázdnění pipeline
- Pozn. : Hlubší pipeline (více stupňů) znamená méně hradel v každém stupni a tím pádem možnost zvýšit pracovní frekvenci procesoru.., avšak více stupňů znamená i vyšší režii (nutnost lépe řadit instrukce do pipeline)

# Paralelizmus na úrovni instrukcí – zřetězení + superskalárnost

Zřetězené superskalární vykonávání:




$$S_{k,N} = \frac{T_1}{T_{k,N}} = \frac{nNk\tau}{k\tau + (n-1)\tau} \qquad \lim_{n\to\infty} S_{k,N} = Nk$$

- Sekvenční instrukční proud
- Datové závislosti jsou dynamicky zjišťovány hardwarem (vs. softwarem při kompilaci -> staticky: WLIV)

# Paralelizmus na úrovni instrukcí

Podporné techniky ILP pro zvýšení stupně paralelizmu:

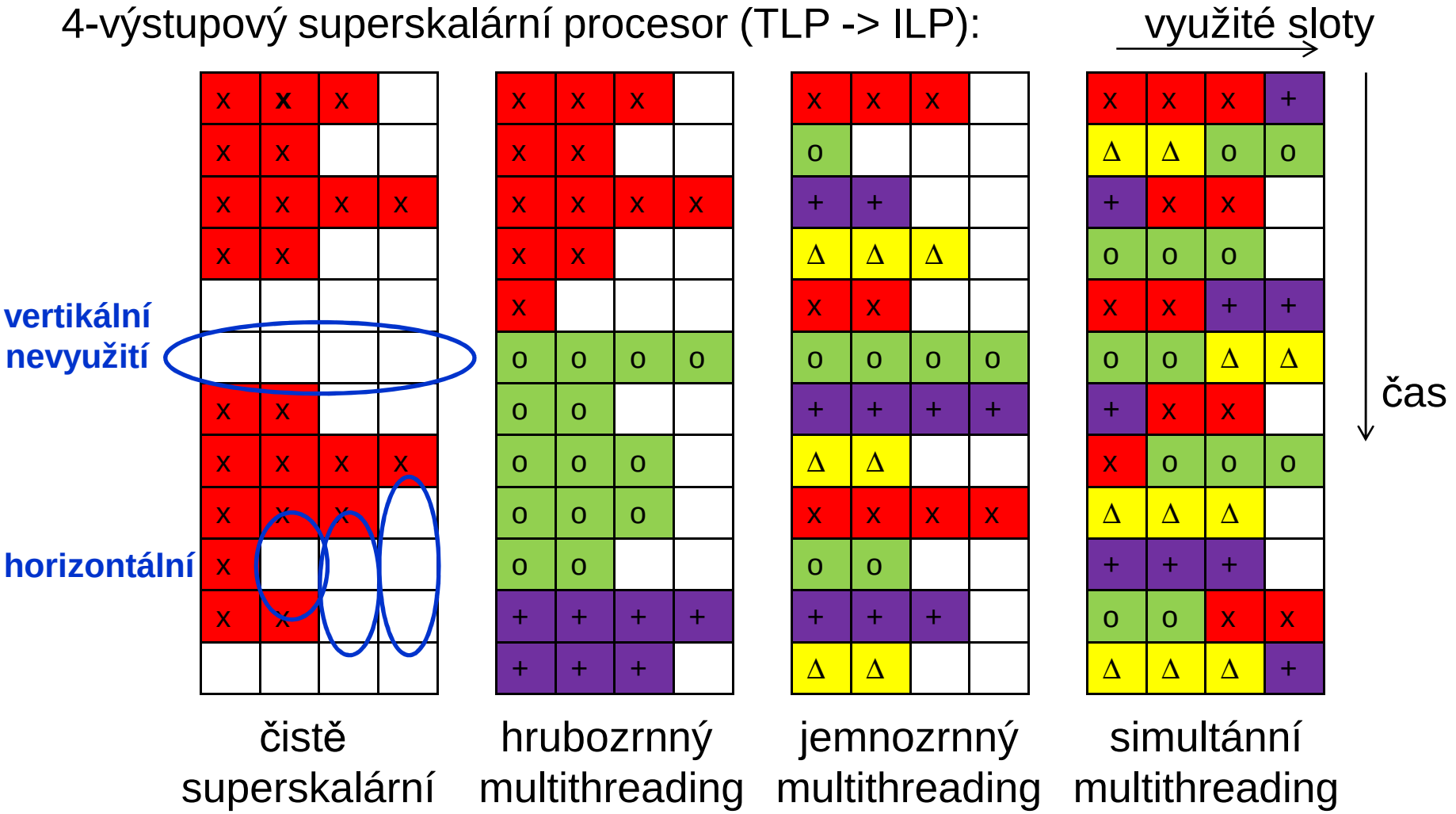
- Předání výsledků uvnitř pipeline (forwarding)
- Vykonávání mimo pořadí (out-of-order execution)
- Přejmenovávání registrů (register renaming)
- Spekulativní vykonávání (speculative execution)
- Předpovídání větvení (branch prediction)
- VLIW (Very Long Instruction Word) a EPIC – MIMD na nejnižší úrovni
- Detailněji v přednáškách č. 05 až 08…

# Paralelizmus na úrovni vláken

- Multithreading – vícero vláken sdílí funkční jednotky procesoru (jádra) – snaha využít nevyužité v porovnání s multiprocessingem
- Zvyšuje propustnost celého systému, ne individuálního vlákna
- Procesor (jádro) musí uchovávat stav každého vlákna – přepínaní kontextu (kopie pracovních registrů - RF, GPR, PC, …)
- Podpora virtuální paměti
- Schopnost přepínání vláken rychle vs. přepínání procesů
- Multithreading:
  - temporální (nebo také prokládaný – interleaved)
    - jemnozrnný
    - hrubozrnný
  - simultánní (nebo také hyperthreading – Intel) – vždy jemnozrnný

# Paralelizmus na úrovni vláken

4-výstupový superskalární procesor (TLP -> ILP):



# Paralelizmus na úrovni vláken

- Čistě superskalární – limitován nedostatkem ILP, dlouhé přestoje při výběru instrukcí z paměti (instruction L2 cache miss)
- Hrubozrnný MT – dlouhé přestoje odstraněny přepínáním vláken; prázdné cykly (start-up perioda) a nevyužití všech prostředků zůstává;
- Jemnozrnný MT – přepínaní v každém cyklu; nevyužití všech prostředků zůstává; pozastavená vlákna jsou ignorována;
- Simultánní MT – přepínaní v každém cyklu; souběžné vykonávání více než jednoho vlákna; využití všech prostředků je dáno požadavky vláken

# Paralelizmus na úrovni úloh

- TLP (Task-level parallelism) také též funkční nebo řídicí paralelizmus – softwarový pohled
- multiprocessing jako podpora TLP – softwarový (multitasking) i hardwarový pohled (symetrický / asymetrický; těsně / volně vázaný,..) - HW prostředky jsou pro vlákna rozděleny, avšak SMT možný (procesor Intel Core i7-980X: 6 jader, 2 vlákna/jádro simultánně)
- TLP: SPMD program:

```
if CPU_ID == 0
    then do task "A"
else if CPU_ID == 1
    then do task "B"
end if
```

Každý procesor (jádro) rozpozná své ID a vykoná pouze svou část programu

# Paralelizmus na úrovni úloh

- TLP  MPMD program: rozdělení programu na moduly (vzájemně komunikující!) – náročné vědecké aplikace, ale také klient-server aplikace;
- Klíčové prvky:
  - komunikace mezi uzly (z hardwarové stránky) nebo procesy či vlákny (ze softwarové stránky)
  - vzájemná synchronizace
- Podle míry komunikace se HPC programy spouštějí na:
  - těsne vázaných procesorech (MPP)
  - volně vázaných procesorech (Cluster, Grid)
- Běh nezávislých programů

# Paralelizmus na úrovni dat

- identické operace nad množinou dat (SIMD)
- rozdělení dat jednotlivým uzlům (procesům):

```
for  i  from lower_limit to upper_limit
     do  a[i] = b[i] + c[i]
```

} Každý procesor (jádro) má jiný dolní a horní limit = pracuje s jinými daty

- paralelizmus – explicitně vyjádřen (OpenMP), implicitní (kompilátory)
- podpora programovacími jazyky

# Závislosti v programech

- Podmínkou paralelního vykonání více programových segmentů – nezávislost od ostatních segmentů
- Vyjádření závislostních relací – teorie grafů
  - Uzly – příkazy (segmenty)
  - Hrany (vždy orientované) – relace mezi uzly

Analýza grafu – zjištění existence paralelizmu

Tři typy závislostí:

- Datová – indikuje uspořádací vztahy mezi příkazy
- Zdrojová – zdroje daného systému (konflikty sdílených zdrojů – registre, paměť, ALU, FPU, procesory…)
- Řídicí – pořadí vykonávání příkazů se nedá určit před spuštěním programu (podmíněné skoky, iterace)

# Datová závislost

- Datová:
  - Toková závislost (true dependency)
  - Toková antizávislost (anti-dependency)
  - Výstupní závislost (output dependency)
  - Vstupně-výstupní závislost
  - Neznáma závislost

Na úrovni instrukcí při realizaci pipeline

Při návrhu paralelního programu

- Toková závislost (Read-after-Write: RAW) $S1 \rightarrow S2$:
  S2 je tokově závislý od S1 pokud $\exists$ vykonávací cesta z S1 do S2 a nejméně jeden výstup S1 se přivádí do S2. Zápis: $O(S1) \cap I(S2)$ , S1-> S2
- Toková antizávislost (Write-After-Read: WAR) $S1 \dashv\rightarrow S2$:
  $I(S1) \cap O(S2)$ , S1-> S2
- Výstupní závislost (Write-after-Write: WAW) $S1 \ominus\rightarrow S2$:
  $O(S1) \cap O(S2)$, S1->S2 (vytvářejí tu samou výstupní proměnnou)

# Datová závislost

- Vstupně-výstupní závislost $S1 \xrightarrow{V/V} S2$
  když oba V/V příkazy (read, write) se odkazují ne tentýž soubor (ne proměnnou)

- Neznámá závislost – závislostní relace se nedá určit
  - Index proměnné je sám indexován
  - Proměnná se objevuje více než jednou s indexy, které mají rozdílné koeficienty smyčkové proměnné
  - Index v smyčkové proměnné je nelineární
  - Apod.

# Datová závislost

P1: C = D*E

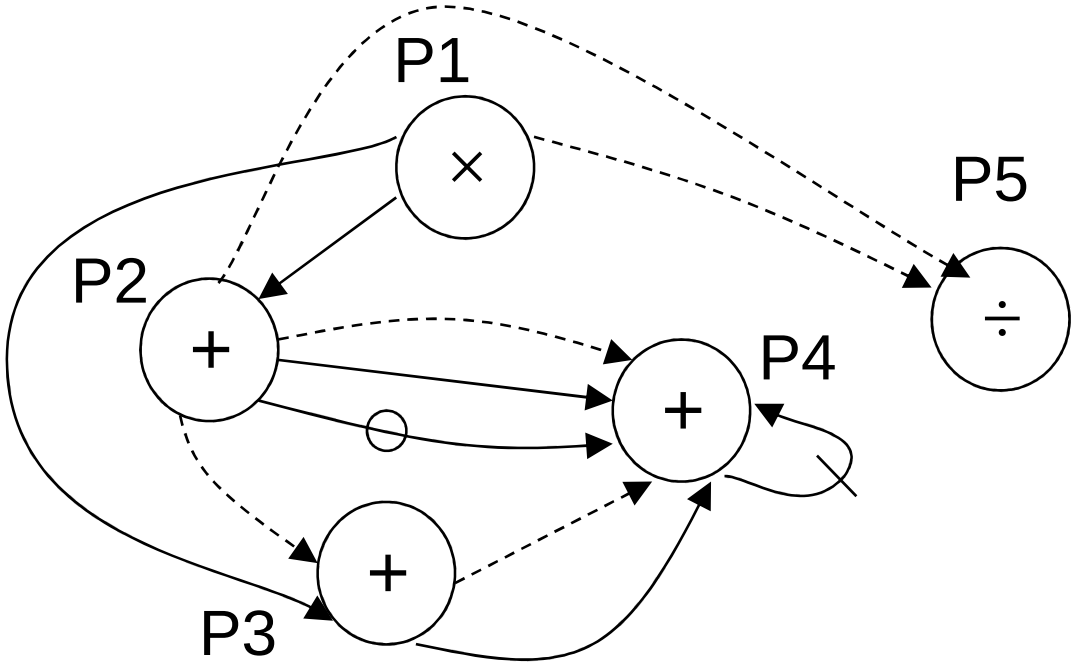
P2: M = G+C

P3: A = B+C

P4: M = A+M

P5: F = G/E

Případně je možné k šipkám místo značek psát: WAW, RAW, WAR, I/O




Plné čáry – datová závislost, Přerušované – zdrojová

# Datová závislost

P1: C = D*E

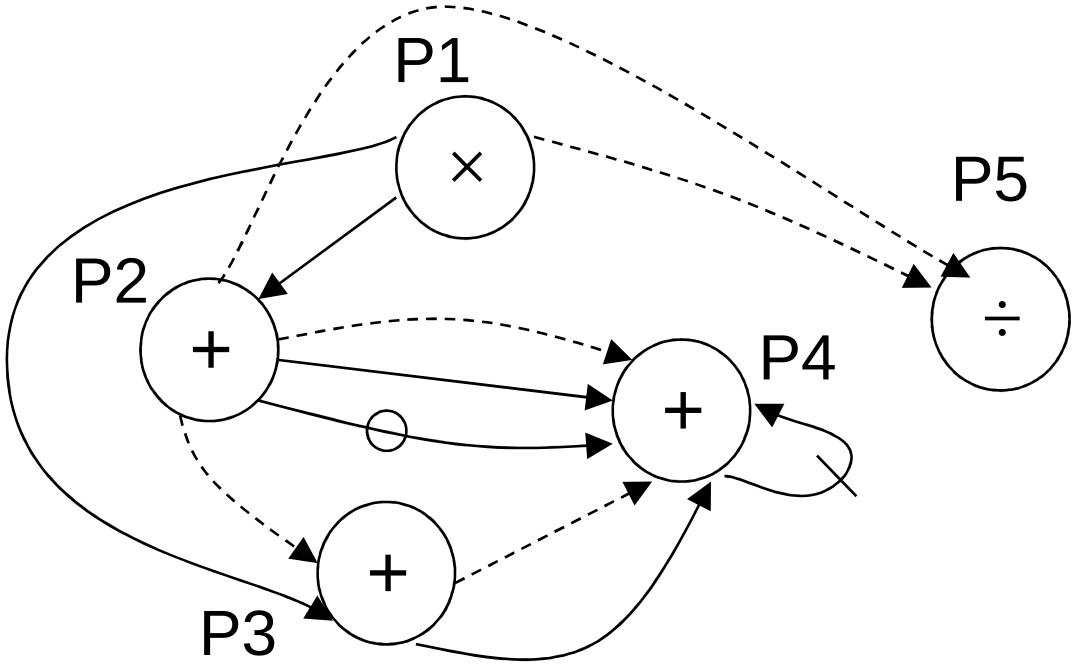
P2: M = G+C

P3: A = B+C

P4: M = A+M

P5: F = G/E

Případně je možné k šipkám místo značek psát: WAW, RAW, WAR, I/O




Pamatujte, že tohle nemusí být jen jedna operace.. Dívejme se obecněji..!

Plné čáry – datová závislost, Přerušované – zdrojová

# Bersteinovy podmínky paralelizmu

- Určují kdy se mohou vykonat dva procesy paralelně v smyslu prostorového paralelizmu
  (proces – softwarová entita zodpovídající abstrakci programového fragmentu na různých úrovních zpracování)
- I – vstupní množina procesu (∀ proměnné potřebné k vykonání procesu)
- O – výstupní množina procesu (generované procesem)
- Procesy Pi a Pj můžeme vykonat paralelně (Pi || Pj) pokud:
  **[I(Pi) ∩ O(Pj)] ∪ [O(Pi) ∩ I(Pj)] ∪ [O(Pi) ∩ O(Pj)] = Ø**
- P1 || P2 || … || Pk  tehdy a jen tehdy, pokud Pi || Pj pro ∀i ≠ j
- Komutativnost platí (Pi || Pj  = Pj || Pi)
- Tranzitivnost neplatí (Pi || Pj  ∧  Pj || Pk  neimplikuje Pi || Pk)
- Asociativnost platí ([Pi || Pj] || Pk = Pi || [Pj ||Pk])

# Bersteinovy podmínky paralelizmu

| Prog. fragment | Všechny dvojice | Všechny trojice |
|---|---|---|
| P1: C = D*E | P1 \|\| P4, P1 \|\| P5 | P1 \|\| P4 \|\| P5 |
| P2: M = G+C | P2 \|\| P3, P2 \|\| P5 | P2 \|\| P3 \|\| P5 |
| P3: A = B+C | P3 \|\| P5 | x |
| P4: M = A+M | P4 \|\| P5 | x |
| P5: F = G/E | x | x |

Bersteinovy podmínky jsou nutnými podmínkami paralelizmu, ne však postačujícími…
Před vykonáním Pj musejí být všechny předchozí závislé (i nepřímo) Pi (i<j) vykonány!

Pokud Pi || Pj $\Rightarrow$ současně, nebo v libovolném pořadí

| Nemůžeme vykonat postupnost: | Můžeme však: | Nebo: | Nebo: |
|---|---|---|---|
| 1. P1 \|\| P4 \|\| P5 | 1. P1 | 1. P1 | 1. P1 \|\| P5 |
| 2. P2 \|\| P3 | 2. P2 \|\| P3 | 2. P2 \|\| P3 \|\| P5 | 2. P2 \|\| P3 |
| | 3. P4 \|\| P5 | 3. P4 | 3. P4 |

# Bersteinovy podmínky paralelizmu

Prog. fragment

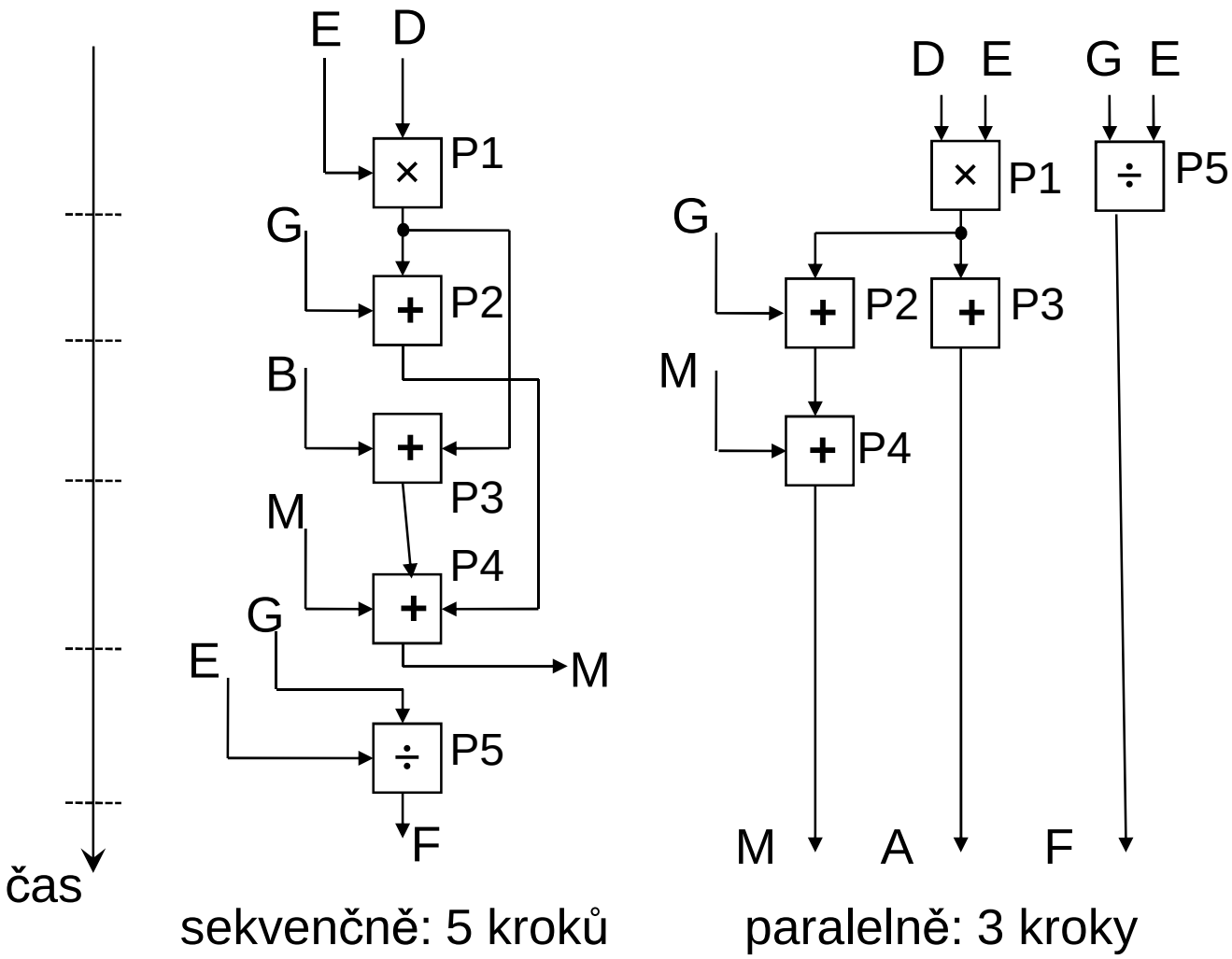
P1: C = D*E

P2: M = G+C

P3: A = B+C

P4: M = A+M

P5: F = G/E



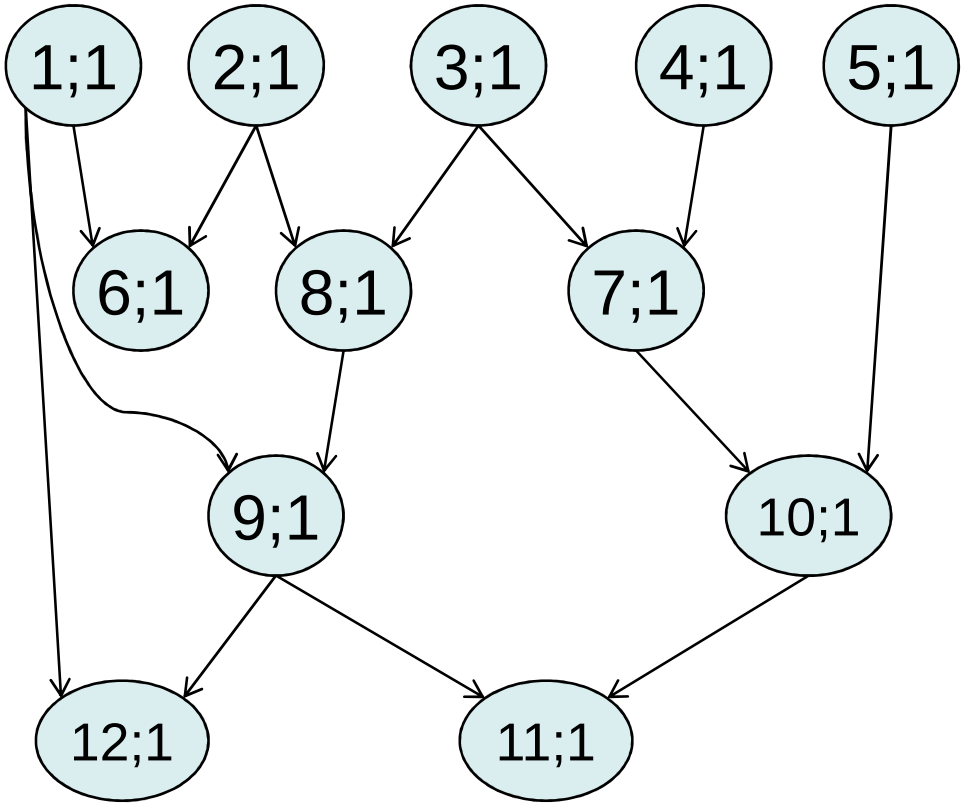
# Úvodní příklad

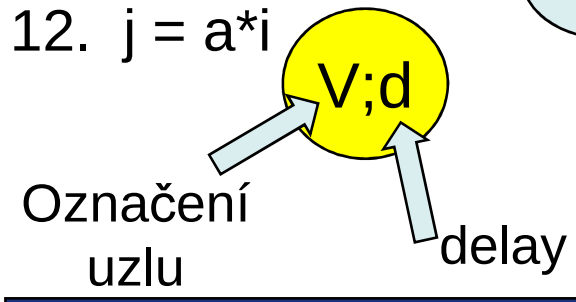
1. a = 1
2. b = 2
3. c = 3
4. d = 4
5. e = 5
6. f = a*b
7. g = c*d
8. h = b–c
9. i = a+h
10. b = g+e
11. c =b*i
12. j = a*i

Realizujme program na dvoj-procesorovém systému obsahujícím dvoj-výstupní procesory schopné vydávat jednu instrukci přístupu do paměti a jednu aritmetickou operaci za jeden cyklus. Latence komunikace mezi procesory nechť je L=2 cykly. Komunikace je neblokující.

# Úvodní příklad

1. a = 1
2. b = 2
3. c = 3
4. d = 4
5. e = 5
6. f = a*b
7. g = c*d
8. h = b–c
9. i = a+h
10. b = g+e
11. c =b*i
12. j = a*i

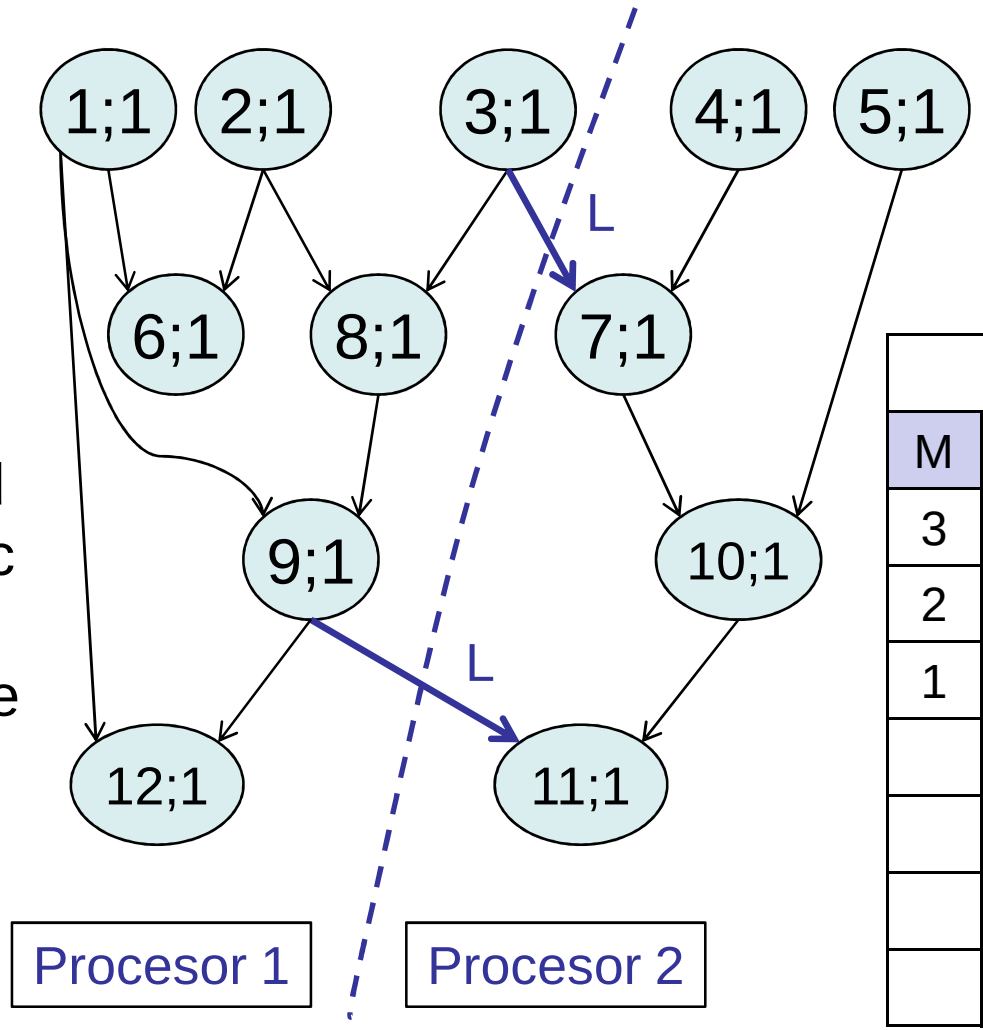



Váha uzlu je mírou množství práce přirazené tomuto uzlu. Nejjednodušší mírou je počet instrukcí (příp. doba vykonání uzlu - počet cyklů).

# Úvodní příklad

1. a = 1
2. b = 2
3. c = 3
4. d = 4
5. e = 5
6. f = a*b
7. g = c*d
8. h = b–c
9. i = a+h
10. b = g+e
11. c =b*i
12. j = a*i

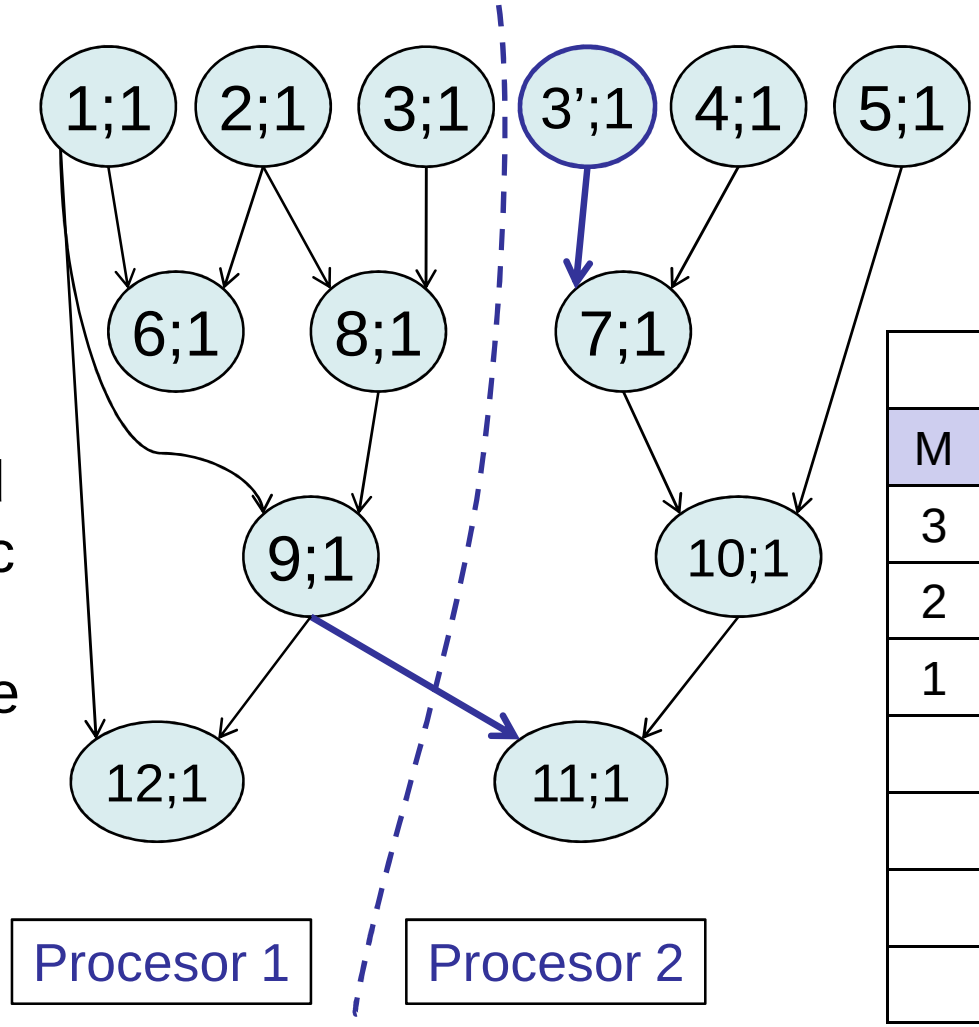



| P1 | | | |
|---|---|---|---|
| M | C | S | R |
| 3 | | | |
| 2 | | 3 | |
| 1 | 8 | 3 | |
| | 9 | | |
| | 12 | 9 | |
| | 6 | 9 | |
| | | | |

| P2 | | | |
|---|---|---|---|
| M | C | S | R |
| 4 | | | |
| 5 | | | 3 |
| | | | 3 |
| | 7 | | |
| | 10 | | 9 |
| | | | 9 |
| | 11 | | |

# Úvodní příklad

1. a = 1
2. b = 2
3. c = 3
4. d = 4
5. e = 5
6. f = a*b
7. g = c*d
8. h = b–c
9. i = a+h
10. b = g+e
11. c =b*i
12. j = a*i



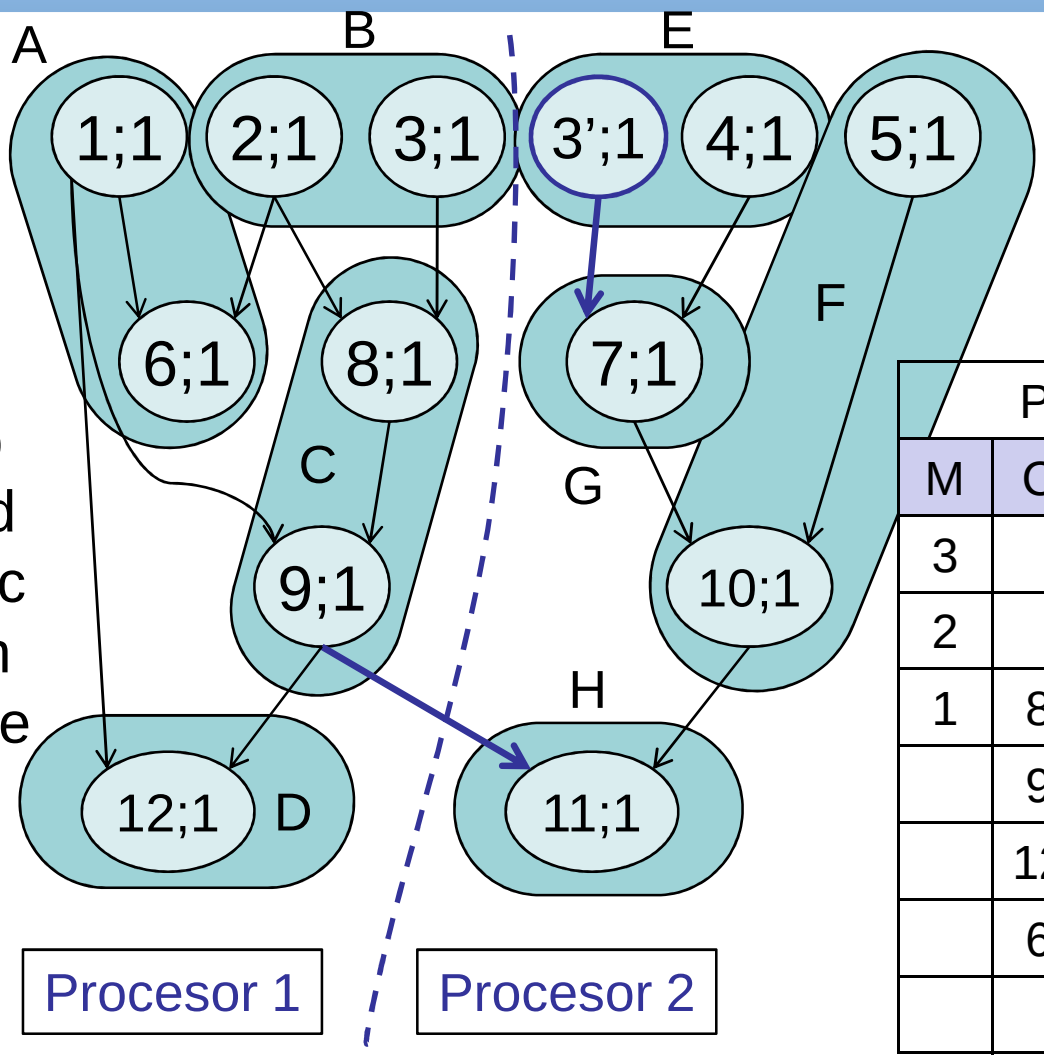

Duplikace uzlů může přinést značné zrychlení..

| P1 | | | |
|---|---|---|---|
| M | C | S | R |
| 3 | | | |
| 2 | | | |
| 1 | 8 | | |
| | 9 | | |
| | 12 | 9 | |
| | 6 | 9 | |
| | | | |

| P2 | | | |
|---|---|---|---|
| M | C | S | R |
| 4 | | | |
| 3 | | | |
| 5 | 7 | | |
| | 10 | | |
| | | | 9 |
| | | | 9 |
| | 11 | | |

# Úvodní příklad

1. a = 1
2. b = 2
3. c = 3
4. d = 4
5. e = 5
6. f = a*b
7. g = c*d
8. h = b–c
9. i = a+h
10. b = g+e
11. c =b*i
12. j = a*i

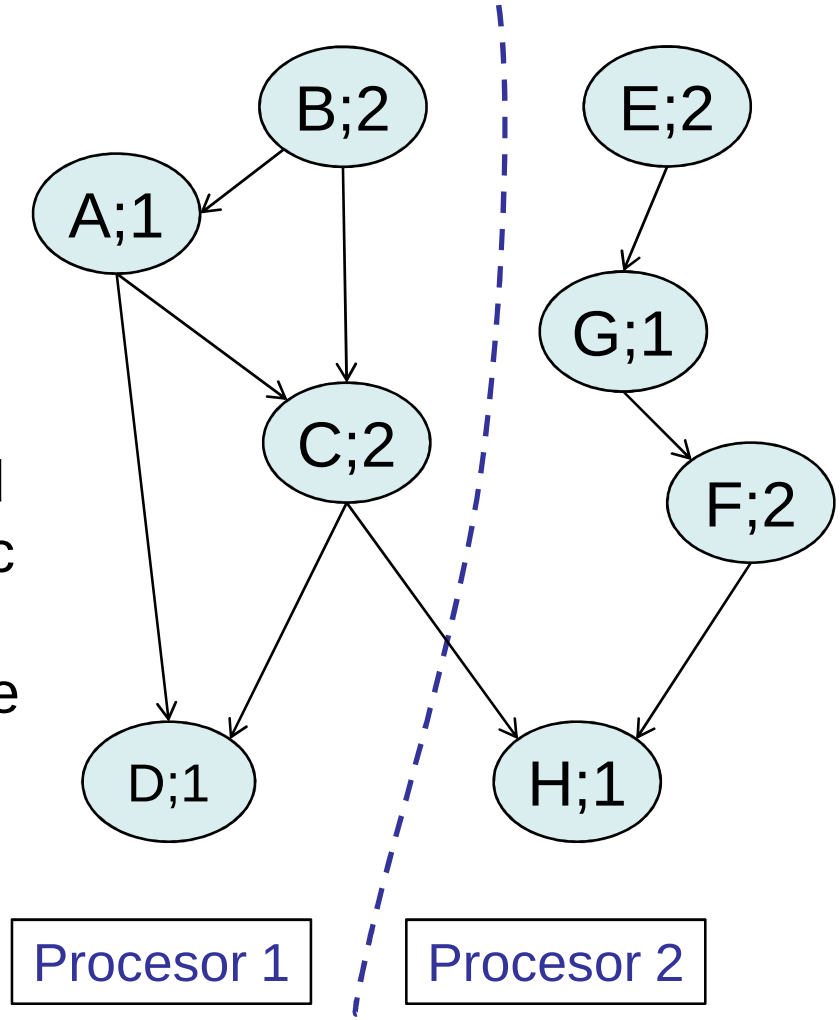



| P1 | | | |
|---|---|---|---|
| M | C | S | R |
| 3 | | | |
| 2 | | | |
| 1 | 8 | | |
| | 9 | | |
| | 12 | 9 | |
| | 6 | 9 | |
| | | | |

| P2 | | | |
|---|---|---|---|
| M | C | S | R |
| 4 | | | |
| 3 | | | |
| 5 | 7 | | |
| | 10 | | |
| | | | 9 |
| | | | 9 |
| | 11 | | |

# Úvodní příklad

1. a = 1
2. b = 2
3. c = 3
4. d = 4
5. e = 5
6. f = a*b
7. g = c*d
8. h = b–c
9. i = a+h
10. b = g+e
11. c =b*i
12. j = a*i



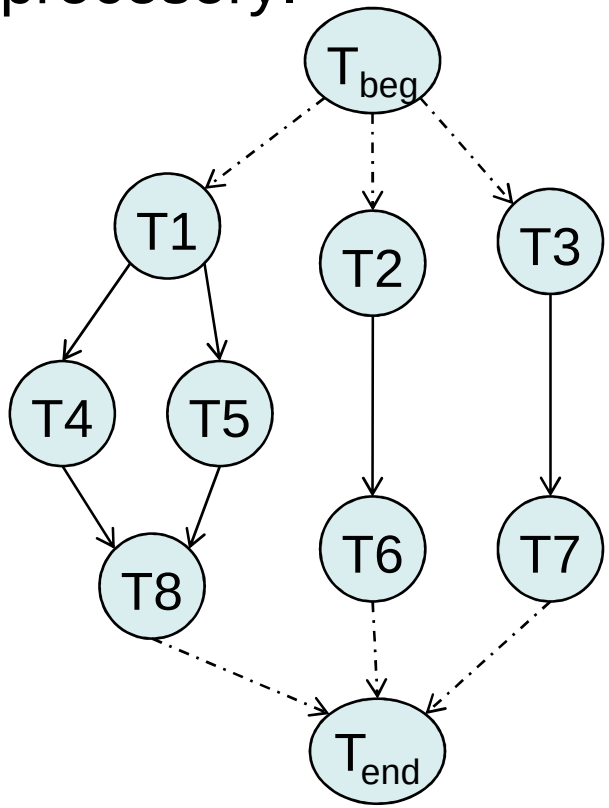

Zrnové balení může přinést značné zjednodušení rozvrhování při zachování zrychlení..

| P1 | | |
|---|---|---|
| C | S | R |
| B | | |
| B | | |
| A | | |
| C | | |
| C | | |
| D | C | |
| | C | |
| | | |

| P2 | | |
|---|---|---|
| C | S | R |
| E | | |
| E | | |
| G | | |
| F | | |
| F | | |
| | | C |
| | | C |
| H | | |

# Úvodní příklad č.2

Mějme tři ekvivalentní procesory.



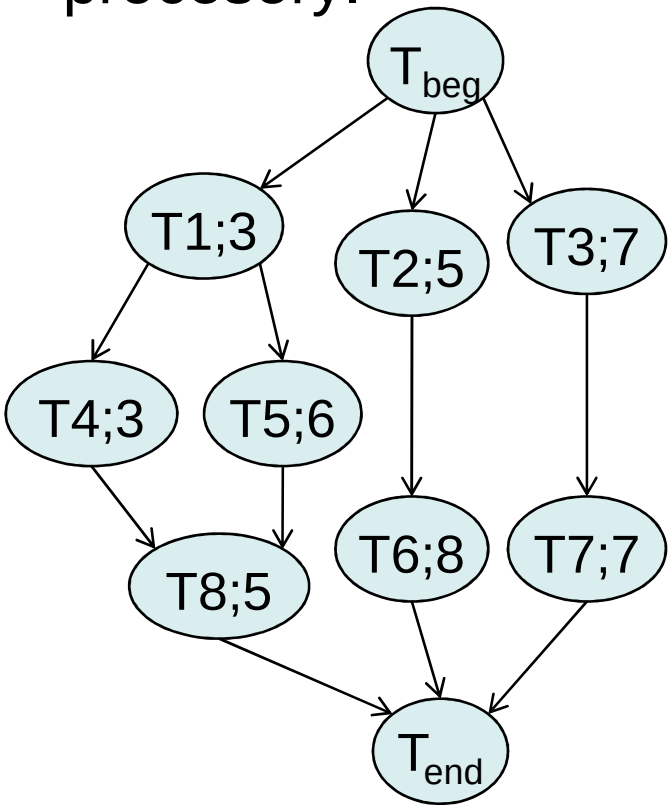

| T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 |
|---|---|---|---|---|---|---|---|
| 3 | 5 | 7 | 3 | 6 | 8 | 7 | 5 |

| T1,T4 | T1,T5 | T2,T6 | T3,T7 | T4,T8 | T5,T8 |
|---|---|---|---|---|---|
| 2 | 6 | 2 | 5 | 3 | 1 |

Jak rozdělíme jednotlivé úkoly těmto třem procesorům ???

# Úvodní příklad č.2

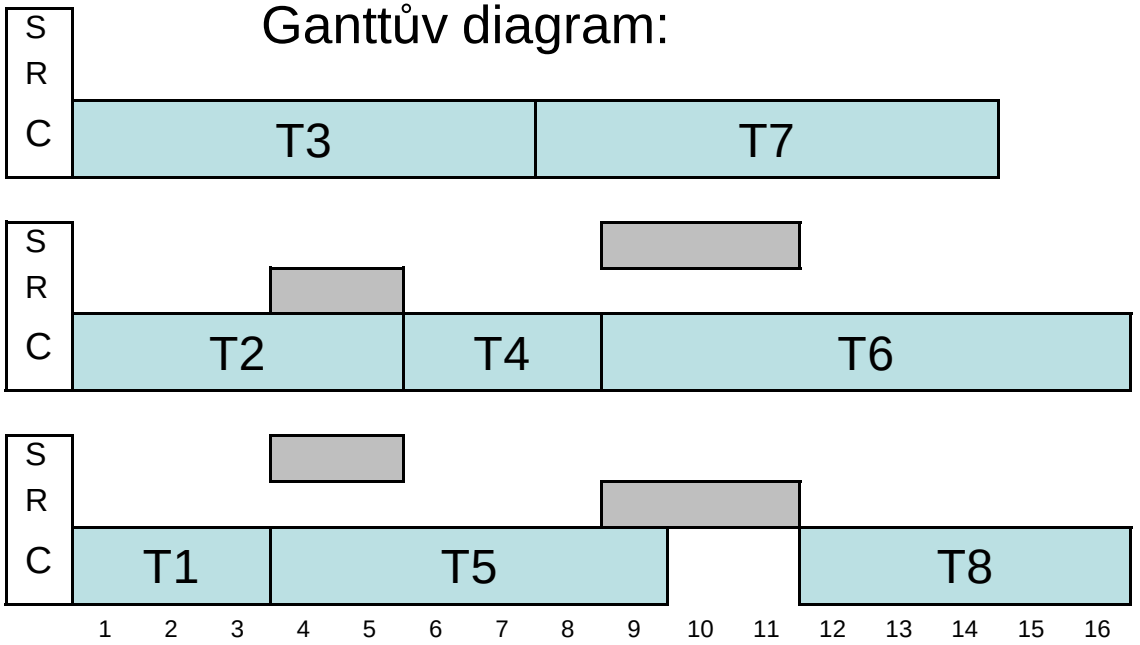
Mějme tři ekvivalentní procesory.




| T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 |
|---|---|---|---|---|---|---|---|
| 3 | 5 | 7 | 3 | 6 | 8 | 7 | 5 |

| T1,T4 | T1,T5 | T2,T6 | T3,T7 | T4,T8 | T5,T8 |
|---|---|---|---|---|---|
| 2 | 6 | 2 | 5 | 3 | 1 |

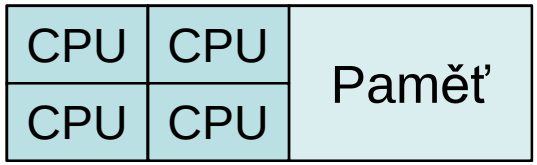
Ganttův diagram:



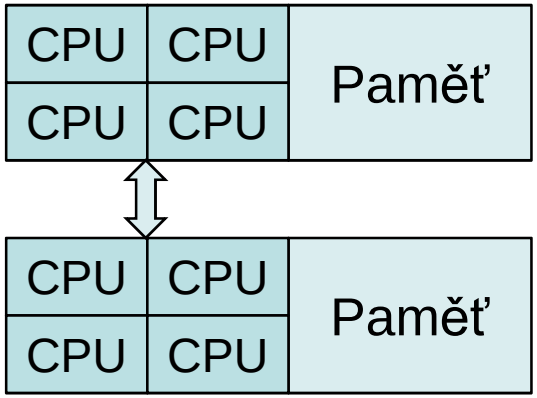

S = 44 / 16 = 2,75

# Paměťové architektury paralelních počítačů

- **Systémy se sdílenou pamětí (SMS)** – schopnost přistoupit do celé paměti všem procesorům (globální adresní prostor), sdílení paměťových zdrojů, komunikace paměť-CPU geometricky narůstá při zvyšování počtu CPU, taktéž zachování paměťové koherence…
  - **UMA** (Uniform Memory Access) – shodný čas přístupu do paměti, SMP (Symetric Multiprocessor), CC-UMA (Cache Coherent UMA)
  - **NUMA** (Non-Uniform) – různý čas přístupu; spojením několika SMP – kdy jeden SMP uzel může přímo přistupovat do paměti jiného; když zachováva Cache Coherency – CC-NUMA




UMA

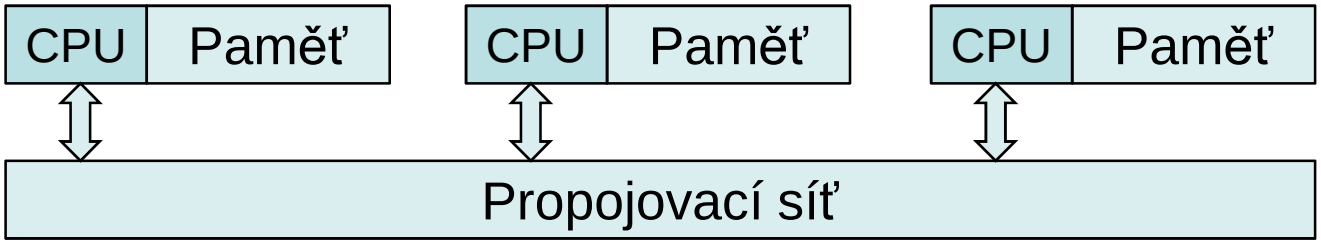



NUMA, RMA (Remote Memory Access)

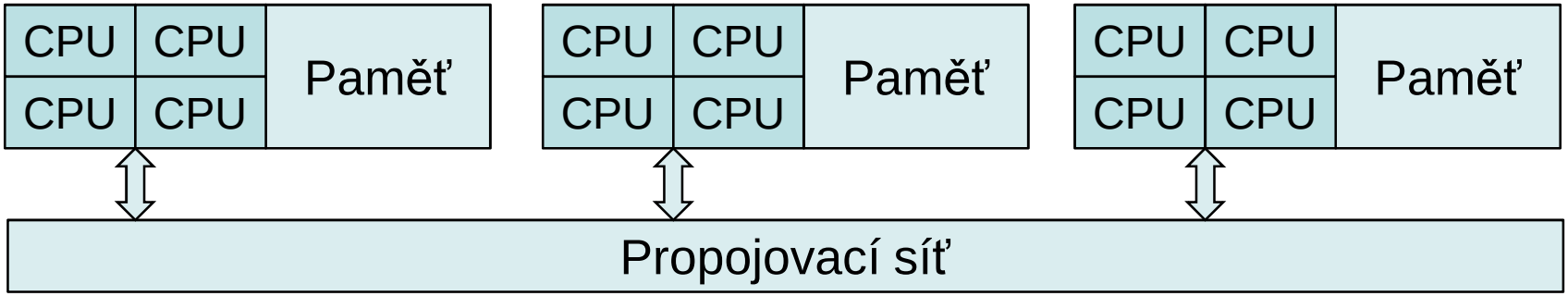
DSM (Distributed Shared Memory) – **DGAS** (D. Global Address Space)

# Paměťové architektury paralelních počítačů

- **Systémy s distribuovanou pamětí (DMS)** – vlastní lokální paměťový prostor, vlastní fyzická paměť; komunikace a synchronizace zajištěna na úrovni programátora; škálovatelnost se zvětšujícím se počtem CPU; NORMA (No RMA)




- **Hybridní** (distribuovaná + sdílená)



# Programovací modely

Abstrakce nad hardwarem a paměťovou architekturou;

Není nutně svázáno s konkrétní architekturou..

- **Sdílená paměť** – úlohy sdílejí společný adresní prostor, asynchronní čtení a zápis; zámky, semafory,..; není potřebná explicitní komunikace při vyměňování dat; Kde jsou uložena data, se kterými procesor pracuje?
- **Vlákna** – Posix Threads (Pthreads) – velmi explicitní paralelizmus – program musí být navržen „paralelně“; OpenMP – paralelizmus vyjádřen direktivy.

# Programovací modely

- **Posílání správ** – výměna dat posíláním a přijímáním zpráv; také pro SMS nejen DMS;
  Jaká musí být maximální latence komunikace, aby se nedegradoval výkon?
- **Datově paralelní** – zaměřujeme se na paralelní vykonávaní operací nad datovými množinami; vhodné jak pro SMS tak DMS; podpora jak v jazycích (HPF – High Performance Fortran) tak direktivách kompilátoru (OpenMP),
- **Hybridní** – kombinace předchozích obyčejně vedoucí na SPMD (Single Program Multiple Data), pro náročné aplikace MPMD.

# Návrh paralelního programu - rozvrhování

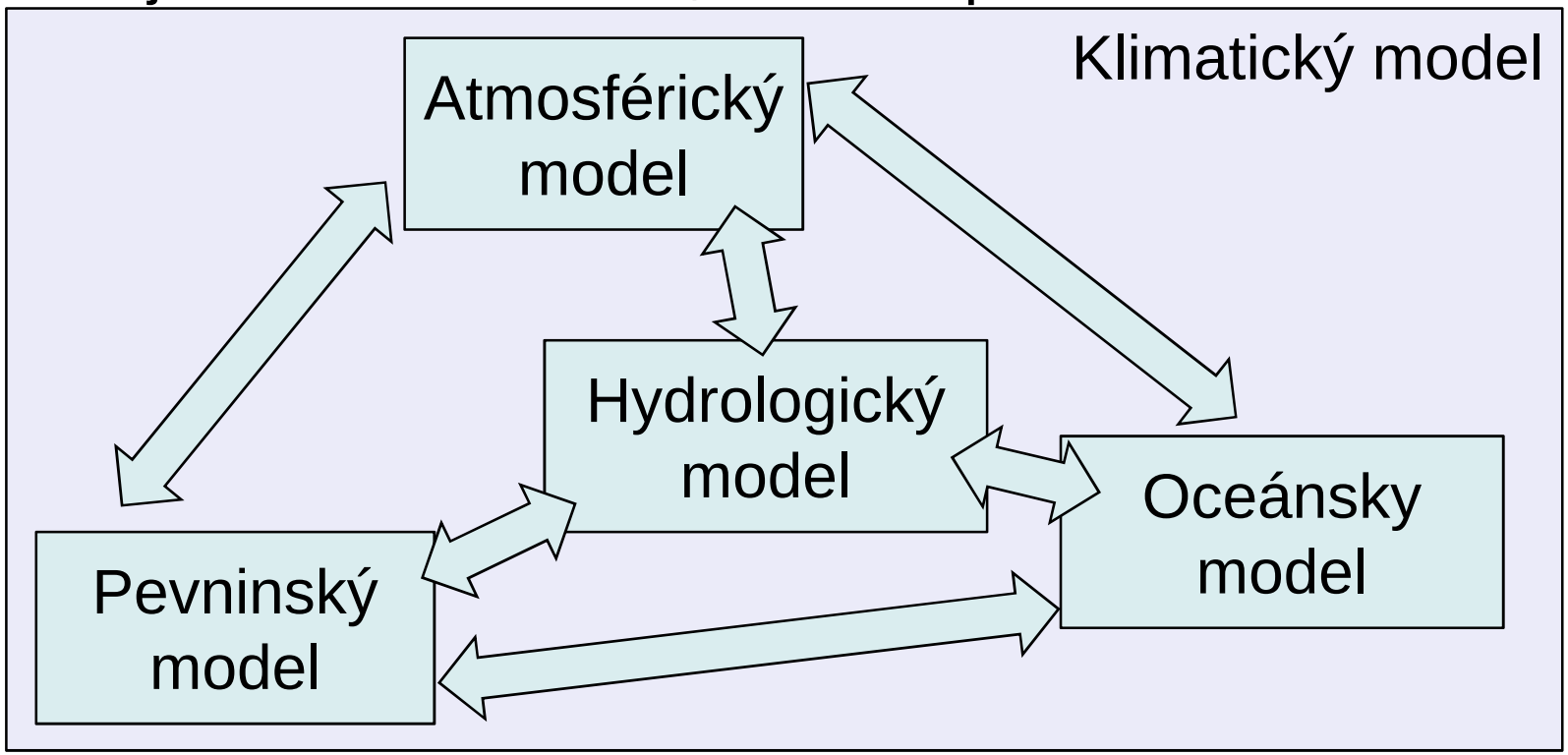
## Rozvrhování (Scheduling)


Zdroj: http://shmu.sk

# Návrh paralelního programu - rozvrhování

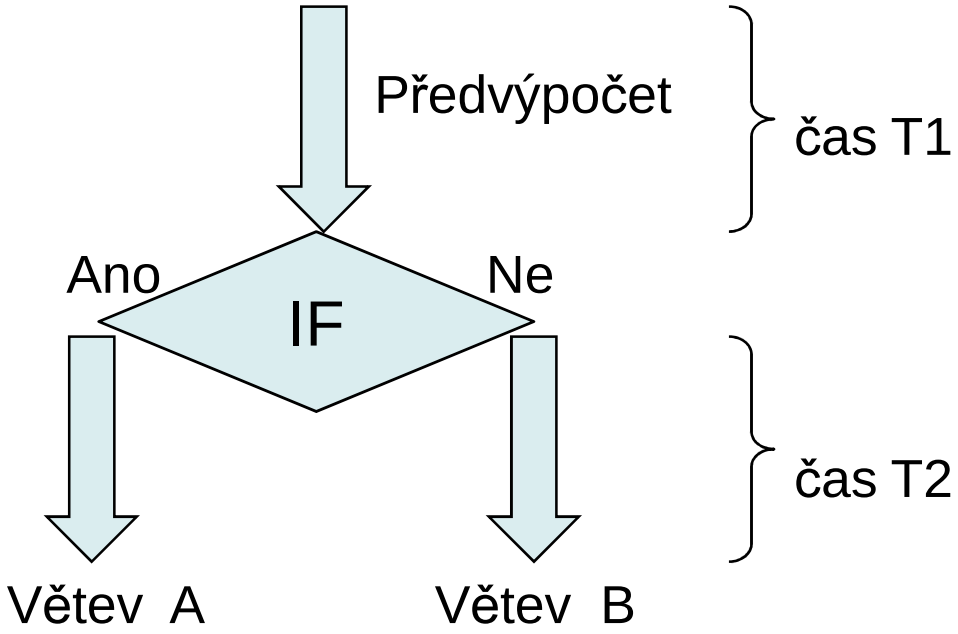
## Rozvrhování (Scheduling)

- Pohled shora (funkční dekompozice): Cílem je rozdělit program do množiny paralelně proveditelných úloh s ohledem na vzájemnou komunikaci; možno aplikovat rekurentně



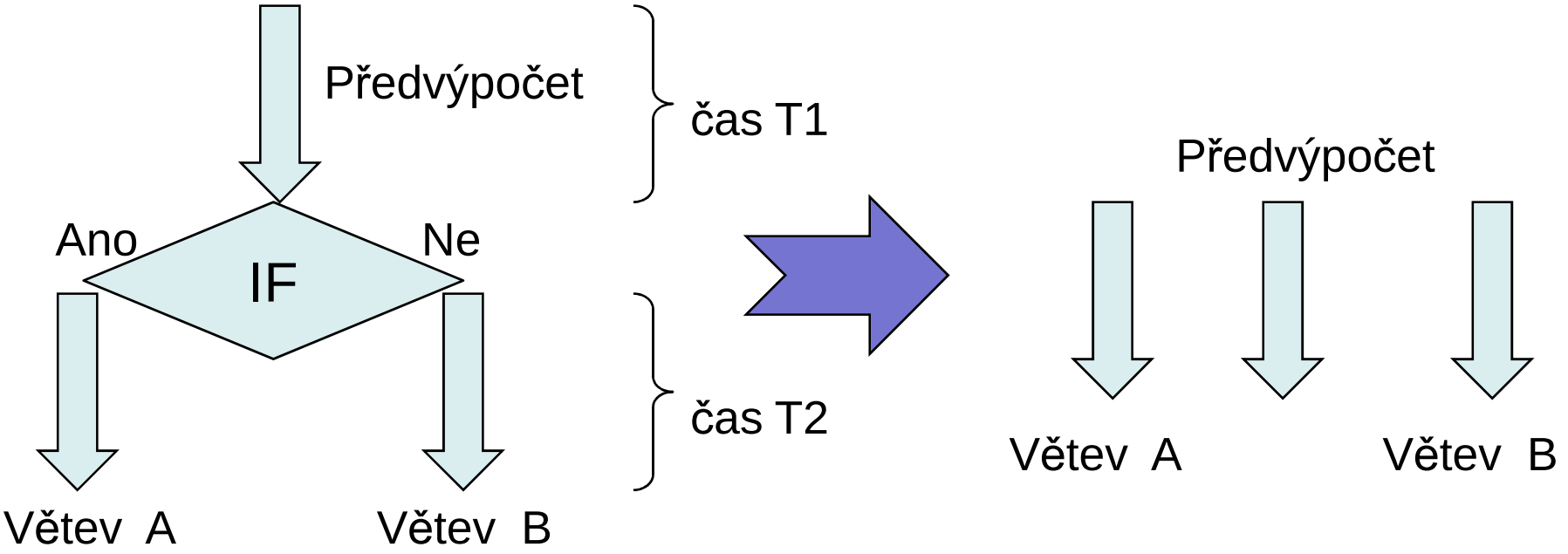
## Rozvrhování (Scheduling)

- Spekulativní dekompozice



## Rozvrhování (Scheduling)

- Spekulativní dekompozice



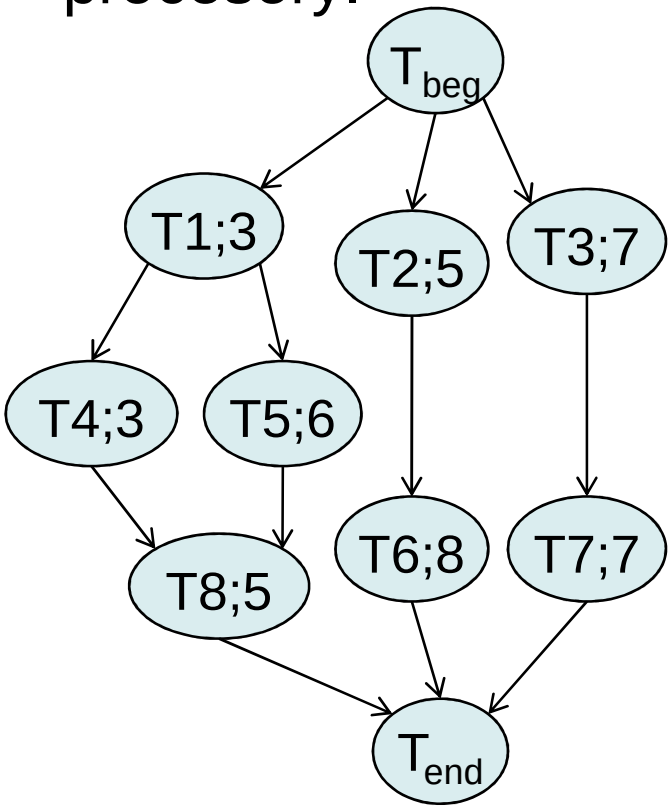
# Návrh paralelního programu - rozvrhování

## Rozvrhování (Scheduling)

- Pohled zdola: Cílem je seskupit sekvenčně prováděné instrukce, příkazy, programové fragmenty bez návaznosti na jiné (jedna linie toku instrukcí) – zrnové balení na nejnižší úrovni, a pak příp. pokračovat podle určité strategie v zrnovém balení s ohledem na komunikaci (viz úvodní příklady).
- Cílem mít co největší soudržnost a co nejmenší spřaženost.
- Kompilátor vs. programátor.
- Zohlednění paměťové architektury.
- Homogenní vs. heterogenní počítačový systém.
- Scheduling také jako algoritmus přidělování zdrojů systému (množství úloh a málo CPU..).

# Úvodní příklad č.2 – rozvrhování / mapování / vyvažování zátěže

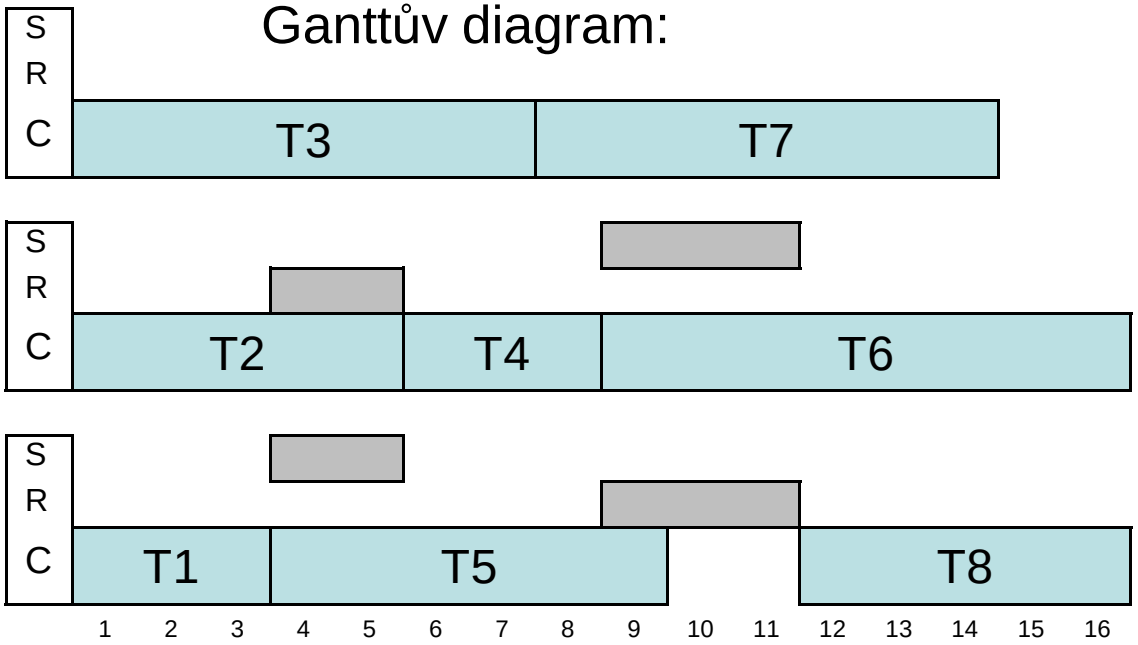
Mějme tři ekvivalentní procesory.




| T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 |
|---|---|---|---|---|---|---|---|
| 3 | 5 | 7 | 3 | 6 | 8 | 7 | 5 |

| T1,T4 | T1,T5 | T2,T6 | T3,T7 | T4,T8 | T5,T8 |
|---|---|---|---|---|---|
| 2 | 6 | 2 | 5 | 3 | 1 |

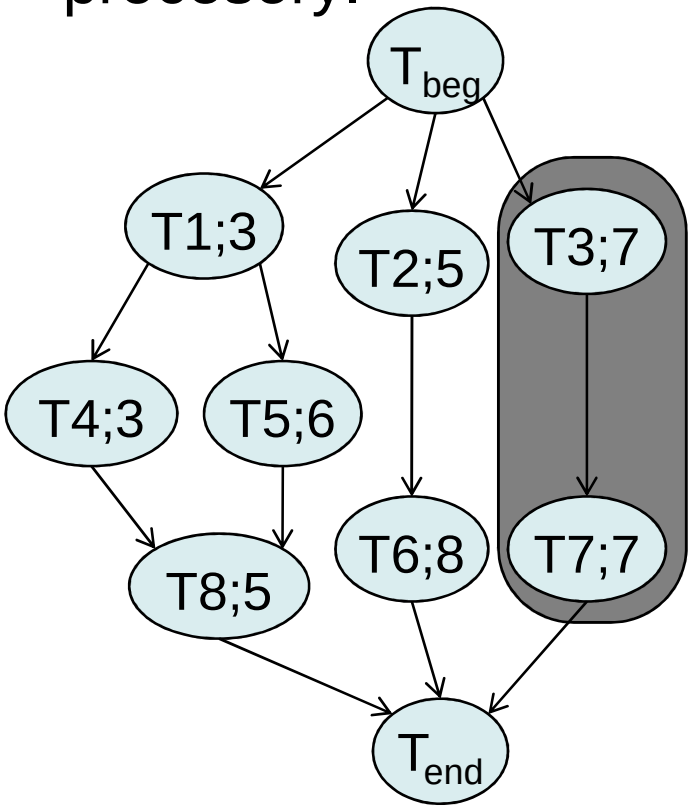
Ganttův diagram:




S = 44 sec / 16 sec = 2,75

# Úvodní příklad č.2 – rozvrhování / mapování / vyvažování zátěže

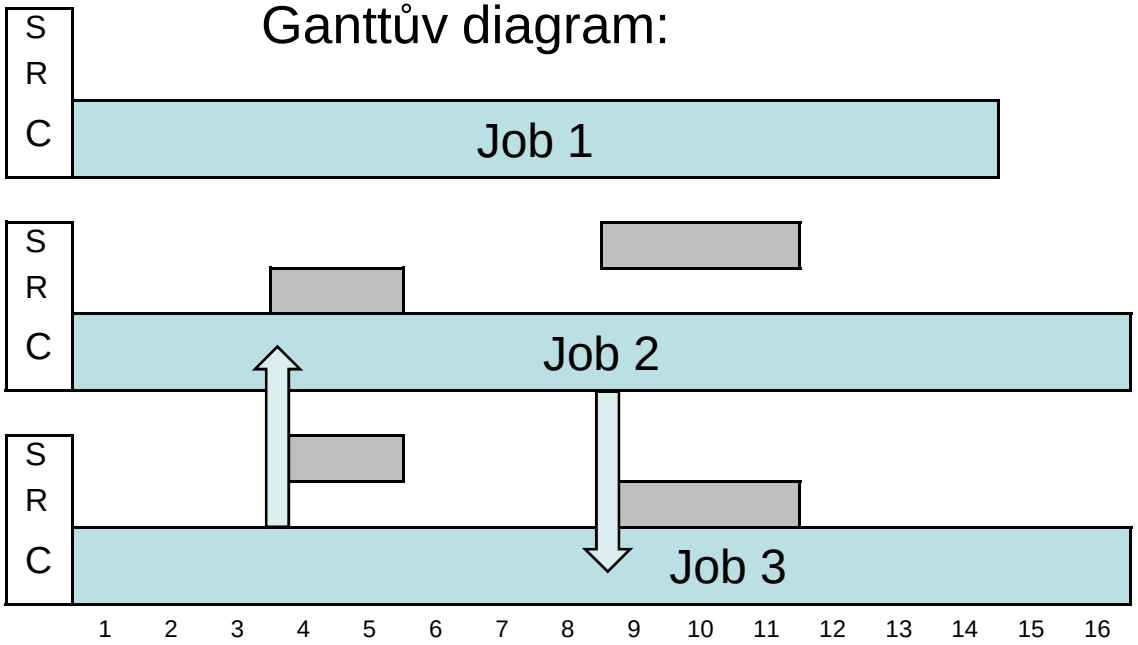
Mějme tři ekvivalentní procesory.




| T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 |
|---|---|---|---|---|---|---|---|
| 3 | 5 | 7 | 3 | 6 | 8 | 7 | 5 |

| T1,T4 | T1,T5 | T2,T6 | T3,T7 | T4,T8 | T5,T8 |
|---|---|---|---|---|---|
| 2 | 6 | 2 | 5 | 3 | 1 |

Ganttův diagram:

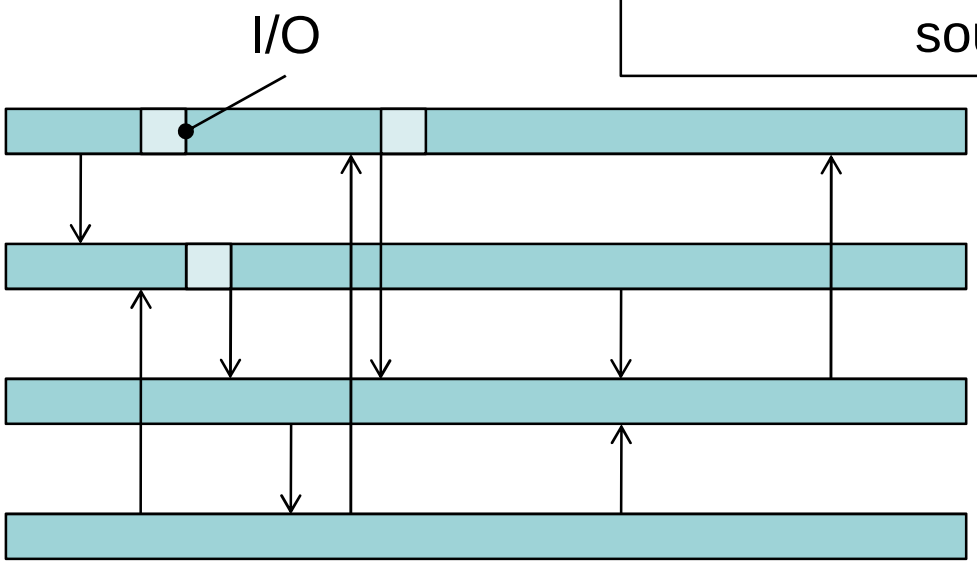



S = 44 sec / 16 sec = 2,75

# Rozvrhování

- Scheduling jako algoritmus přidělování zdrojů systému – rozhodování, která úloha na kterém CPU a kdy
  - First-come-first-serve (čekání na ostatní způsobuje prostoje),
  - Gang scheduling (problémem jsou I/O a blokující komunikace),
  - Paired gang scheduling.

Počet CPU < počet úloh
=> nemohou všechny běžet současně



# Návrh paralelního programu - rozčleňování

## **Rozčleňování (Partitioning)** – Doménová dekompozice

**Ostření**

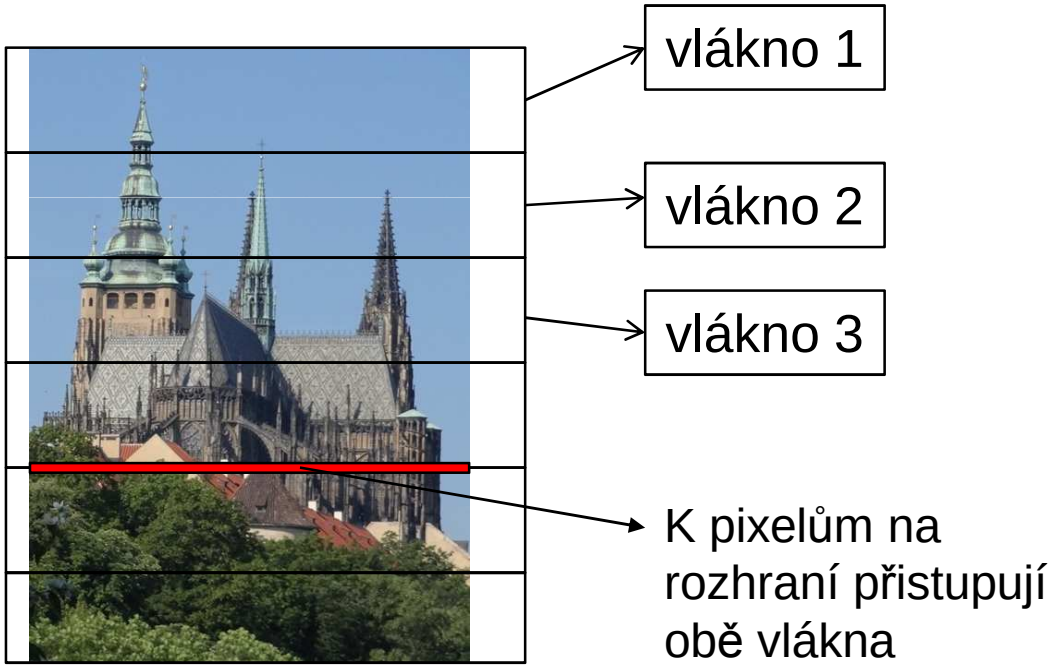
**Jak na to?**

# Návrh paralelního programu - rozčleňování

**Rozčleňování (Partitioning)** – Doménová dekompozice

Jak se obraz zaostřuje? → Konvoluce

Jak paralelně?

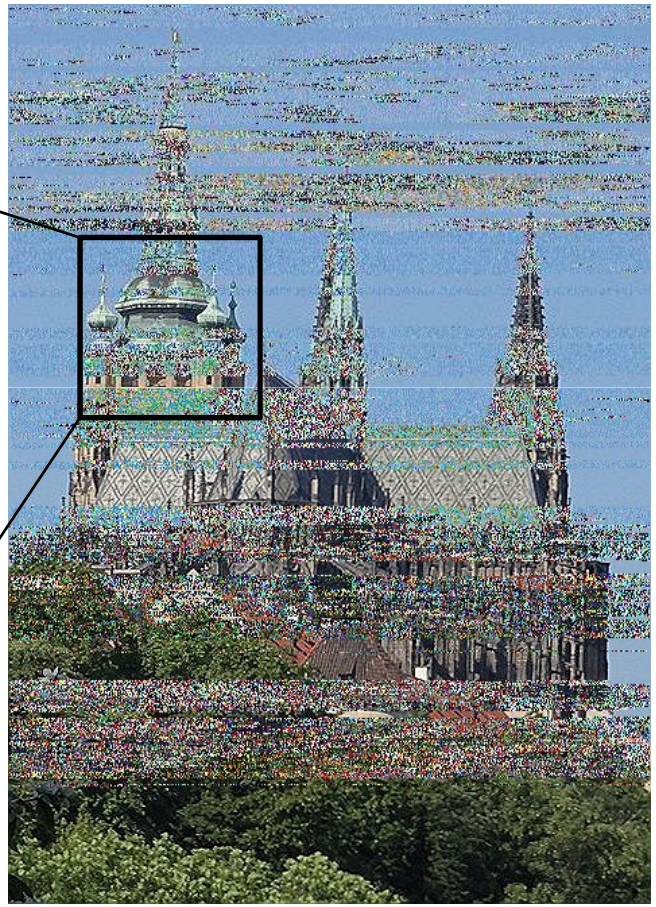



**A co konflikty při přístupu do paměti?**

# Návrh paralelního programu - rozčleňování

**Rozčleňování (Partitioning)** – Doménová dekompozice

Může to dopadnout i takto:

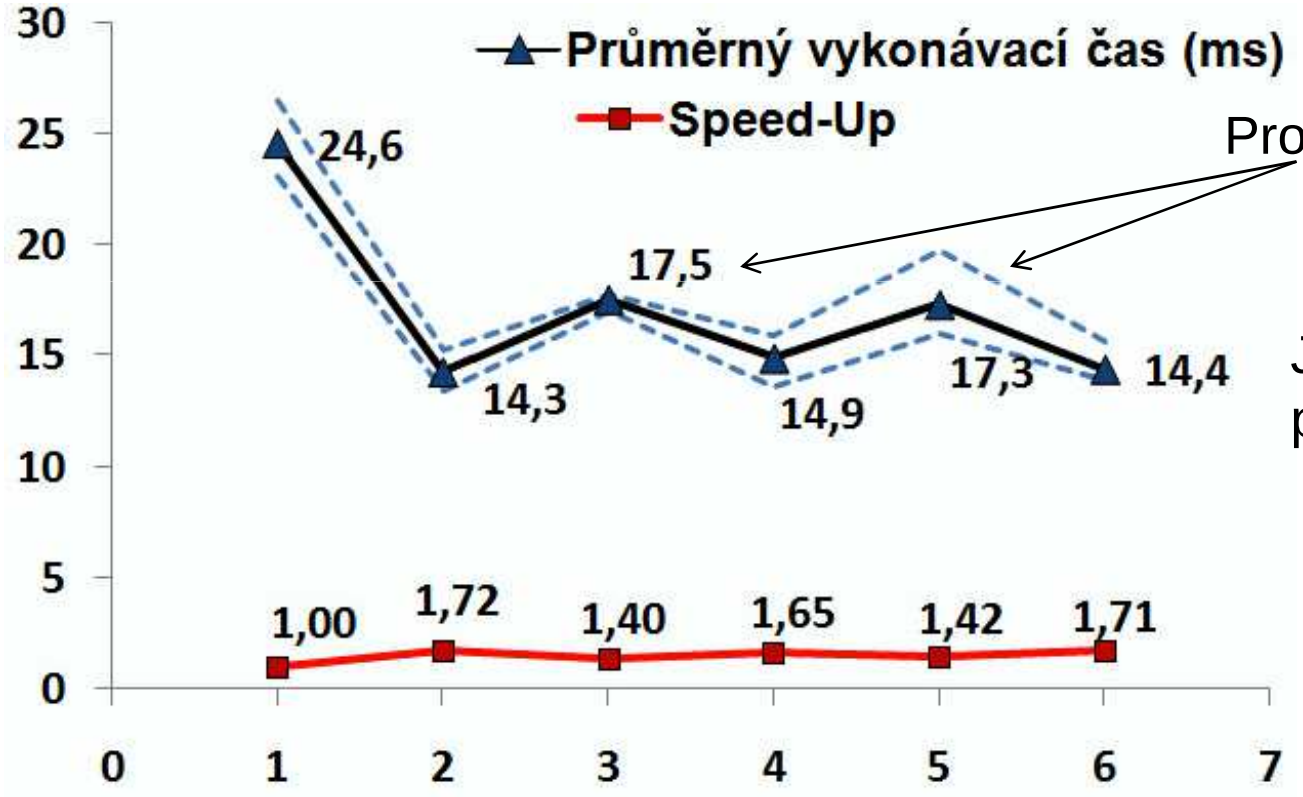


Co je špatně?

# Návrh paralelního programu - rozčleňování

**Rozčleňování (Partitioning)** – Doménová dekompozice

Jak dopadl chrám sv.Víta na dvou-jádrovém CPU?




Proč narůstá?

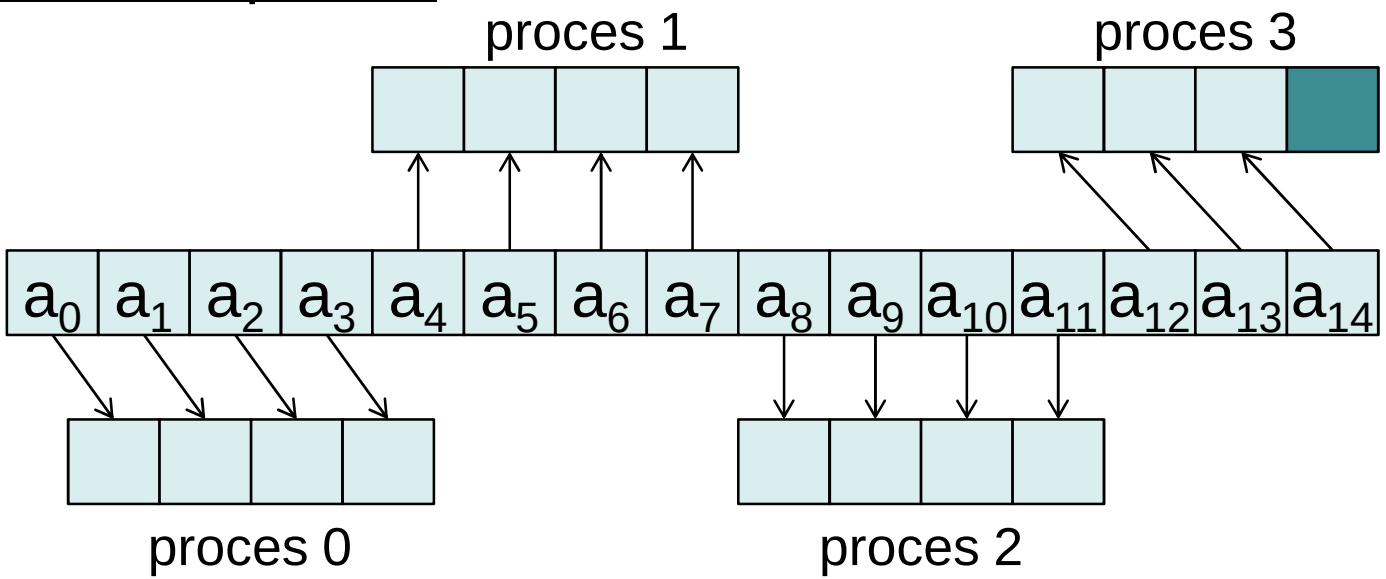
Jak by graf pokračoval?

# Návrh paralelního programu - rozčleňování

**Rozčleňování (Partitioning)** – Doménová dekompozice

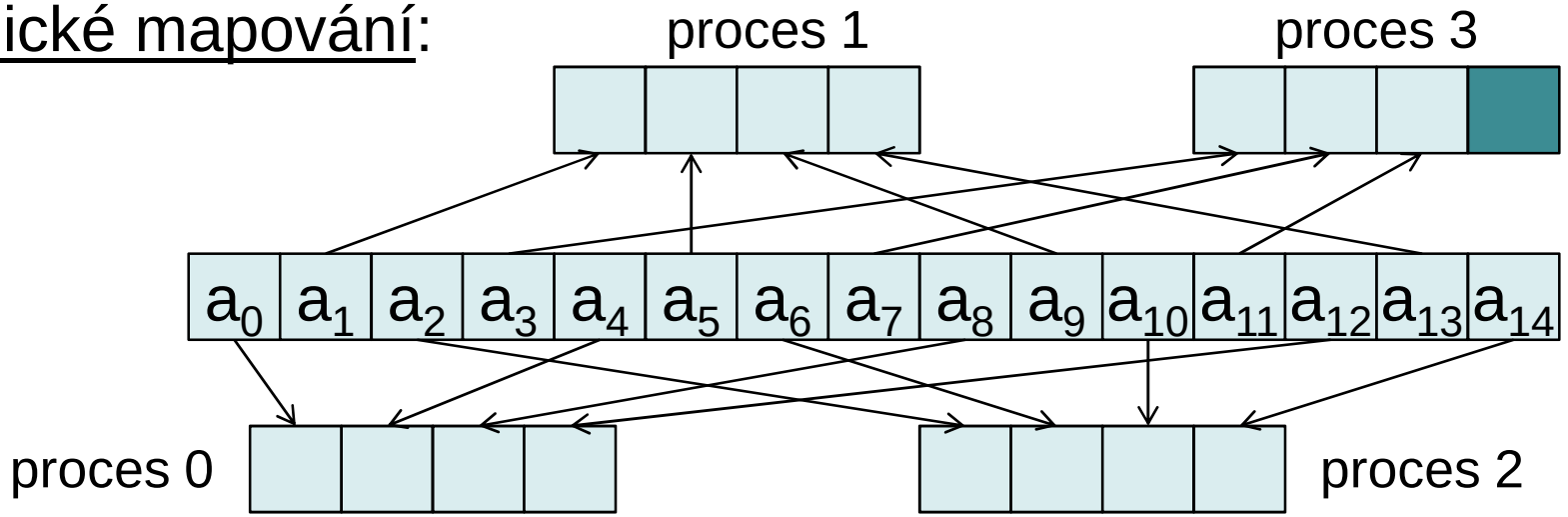
- Množina dat je rozdělena jednotlivým procesům
- $A = (a_0, a_1, \ldots, a_{n-1})$ n prvků
  $P = (q_0, q_1, \ldots, q_{p-1})$ p procesů

Blokové mapování:

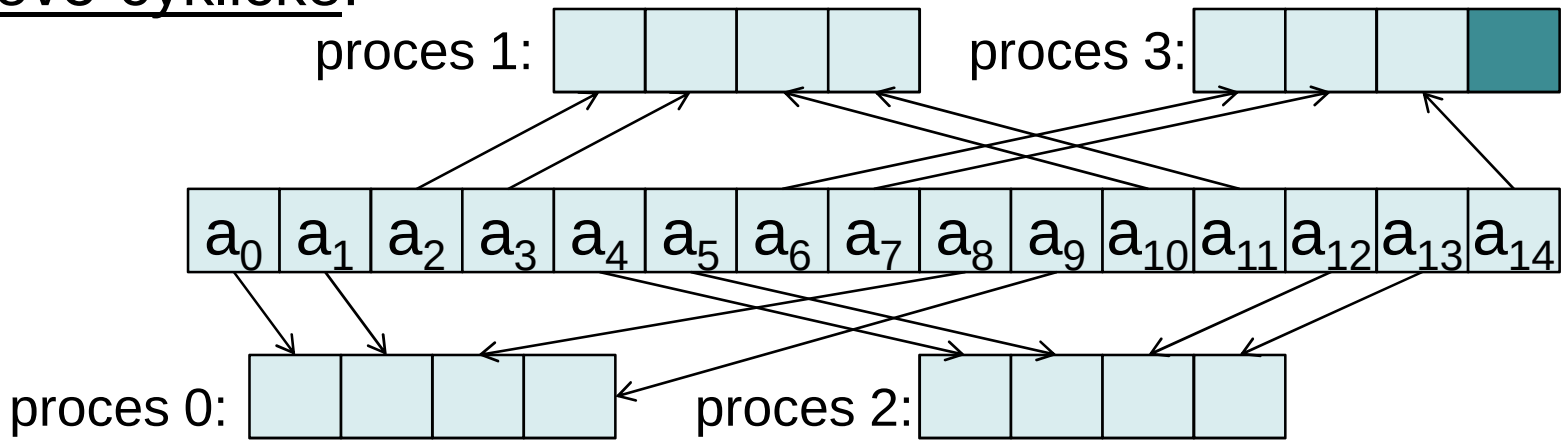


# Návrh paralelního programu - rozčleňování

- Cyklické mapování:




- Blokově-cyklické:



# Návrh paralelního programu - rozčleňování

- Který způsob je nejvýhodnější? Závisí od charakteru řešené úlohy.. (může ovlivnit přesnost nebo dobu výpočtu)

$$\sum_{i=1}^{N} \frac{1}{i^2} = \sum_{i=N}^{1} \frac{1}{i^2} = \sum_{i=1}^{N} \frac{1}{(N-i+1)^2}$$

$$\sum_{i=1}^{10^{10}} \frac{1}{i^2} \approx 1.64493405783018 65,$$

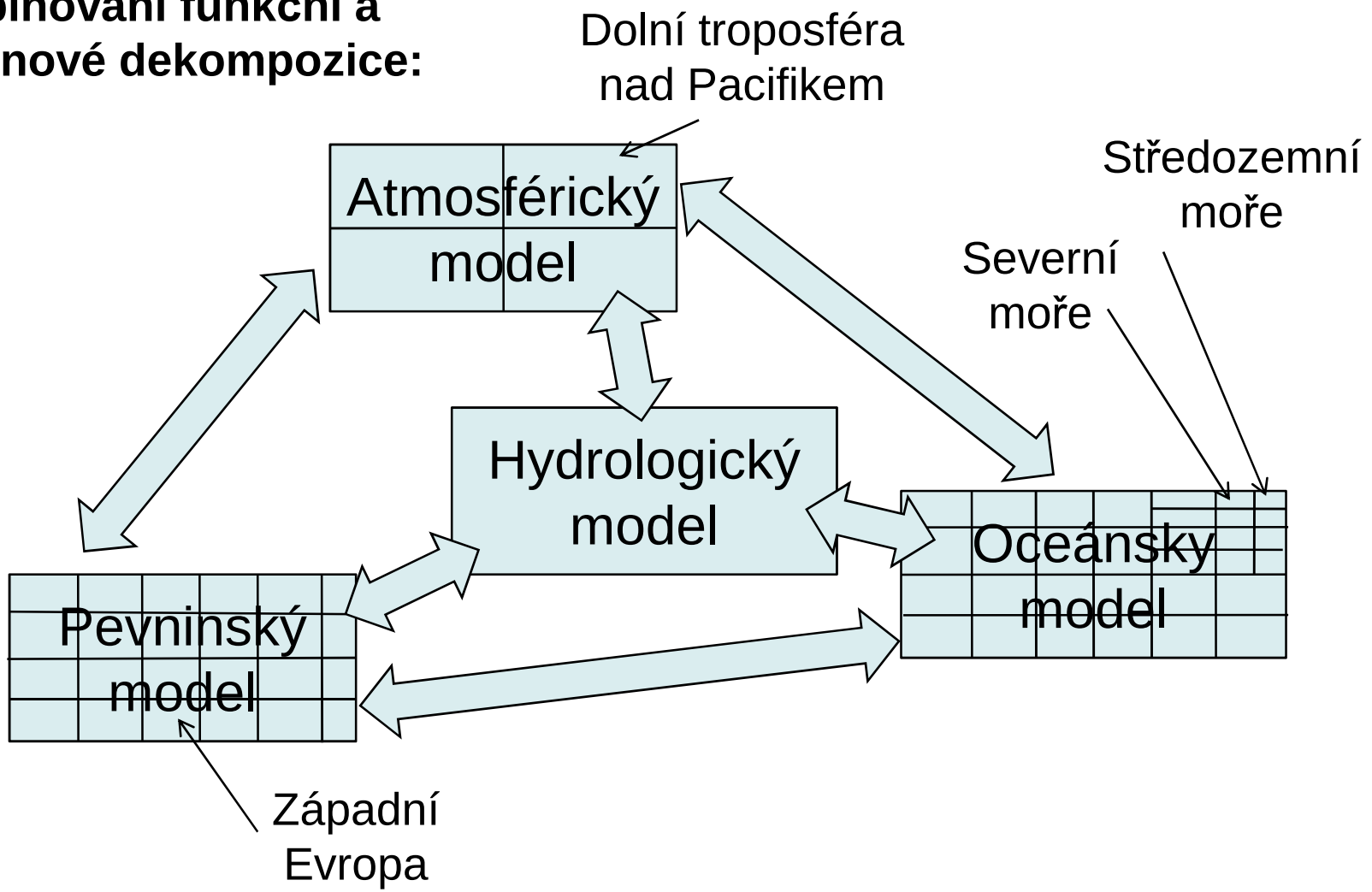
$$\sum_{i=10^{10}}^{1} \frac{1}{i^2} \approx 1.64493406674822 64$$

V závislosti na zvoleném mapování bude výsledek někde mezi.. (typ double)

Doba výpočtu - Výpočet nad některými skupinami elementů může vyžadovat větší počet iterací pro dosažení konvergence…

# Návrh paralelního programu - rozčleňování

**Kombinování funkční a doménové dekompozice:**

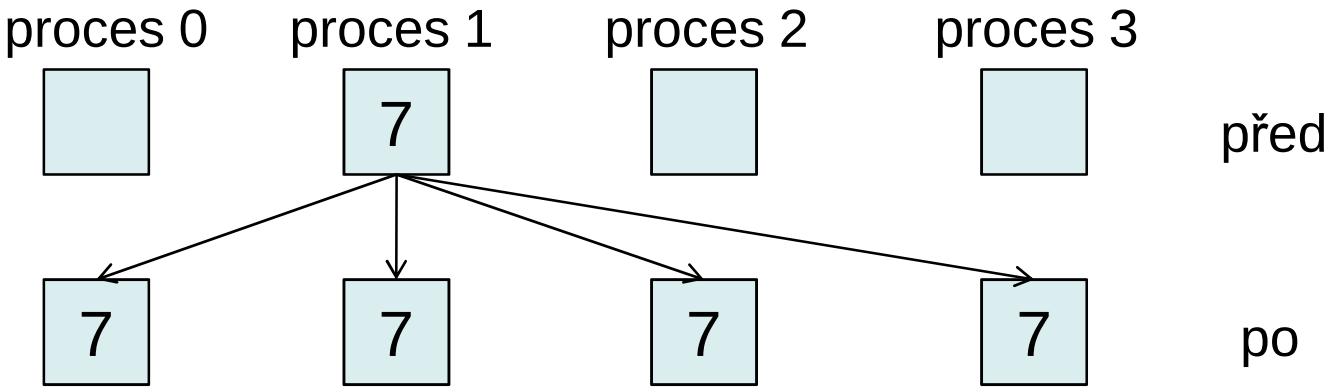


# Návrh paralelního programu – komunikace

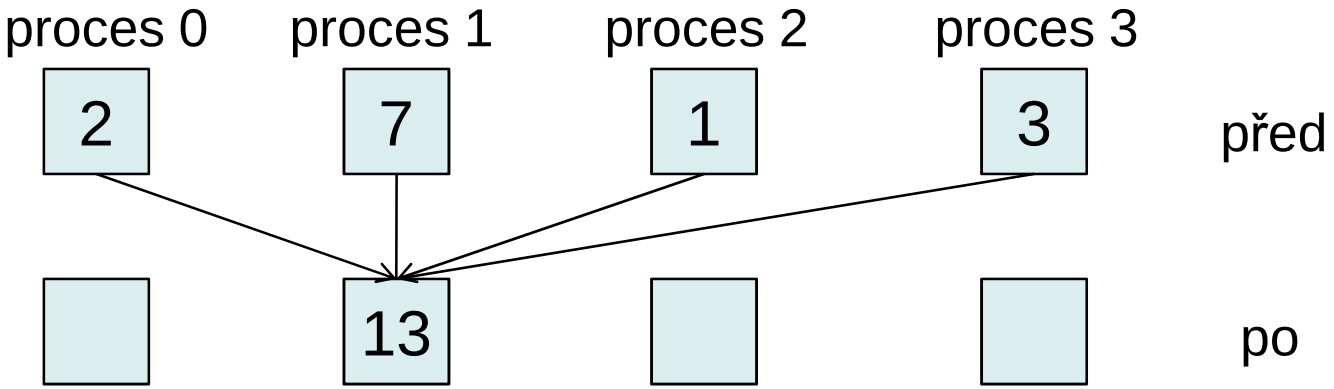
- Přímá komunikace mezi procesy (vlákny) může být programátorovi skryta (závisí na modelu: sdílená paměť, datově paralelní model, vlákna, posílání správ..).
- Cena komunikace.
- Zpoždění (Latency) a šířka pásma (Bandwidth) – mnoho malých správ – dominance latence.., málo velkých – všímáme si víc šířky pásma..
- Synchronní a asynchronní komunikace.
- Point-to-point (Unicast) a kolektivní komunikace; Kolektivní:
  - Broadcast (one-to-all) – šíření – jeden uzel posílá svá data všem,
  - Multicast (one-to-many),
  - Scatter – rozesílání – různé údaje z jednoho uzlu do všech ostatních,
  - Gather – opak scatter,
  - Reduction – redukce – sesbírání dat ze všech uzlů do jednoho,
  - a další.. (Allreduce, Allgather, AlltoAll) -> Kolektivní kom.: vždy blokující.

# Návrh paralelního programu – komunikace

Broadcast (source = 1):

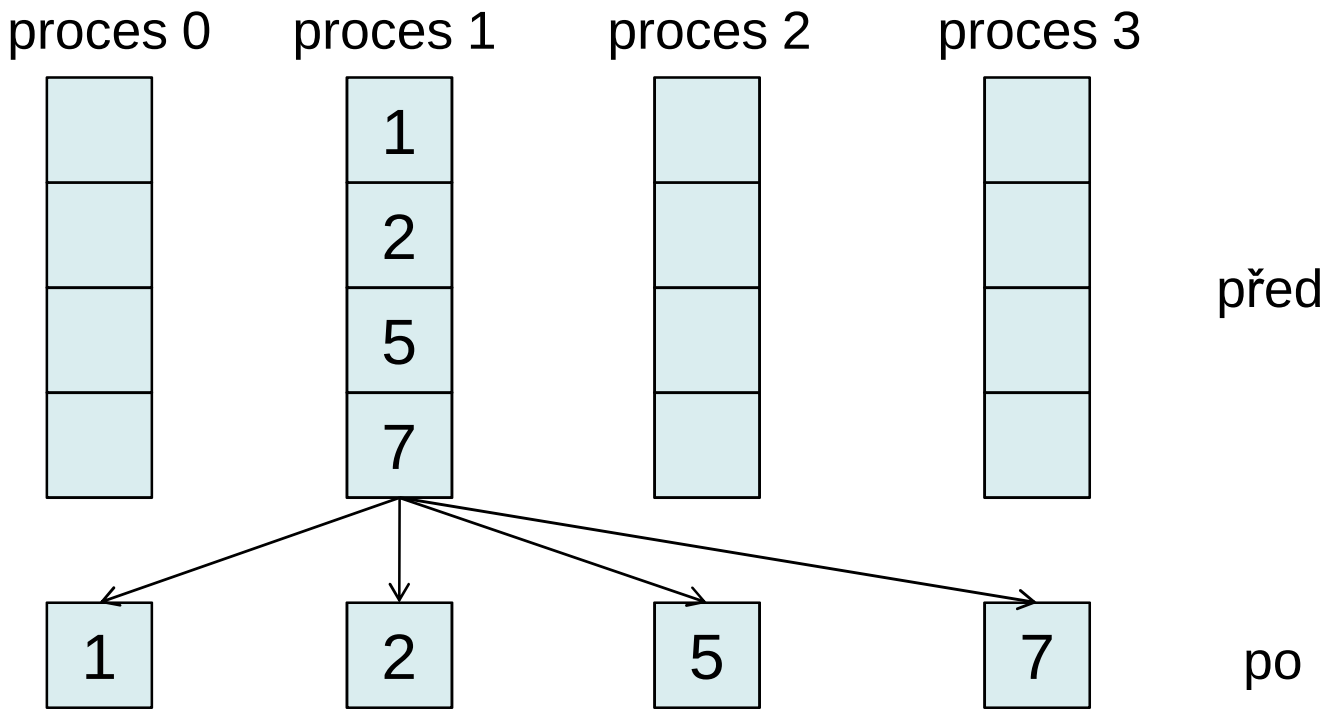



Reduce (destination = 1, operation +):



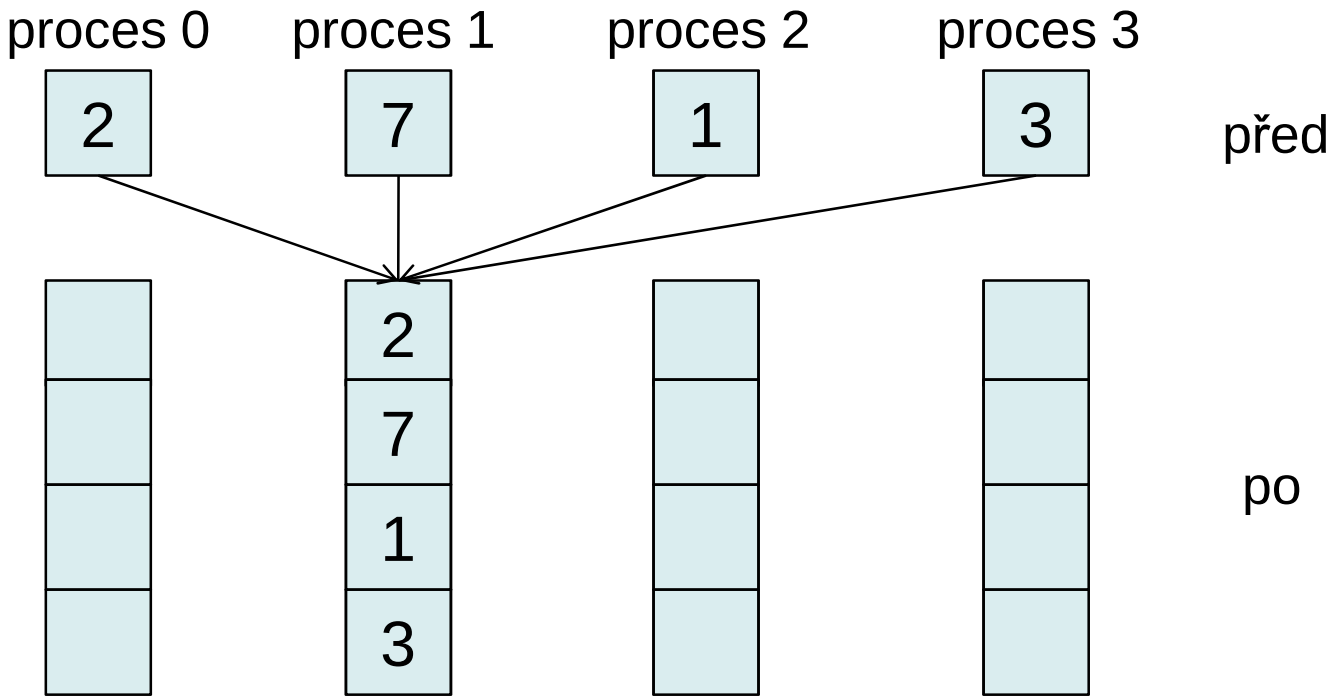
# Návrh paralelního programu – komunikace

Scatter (source = 1):



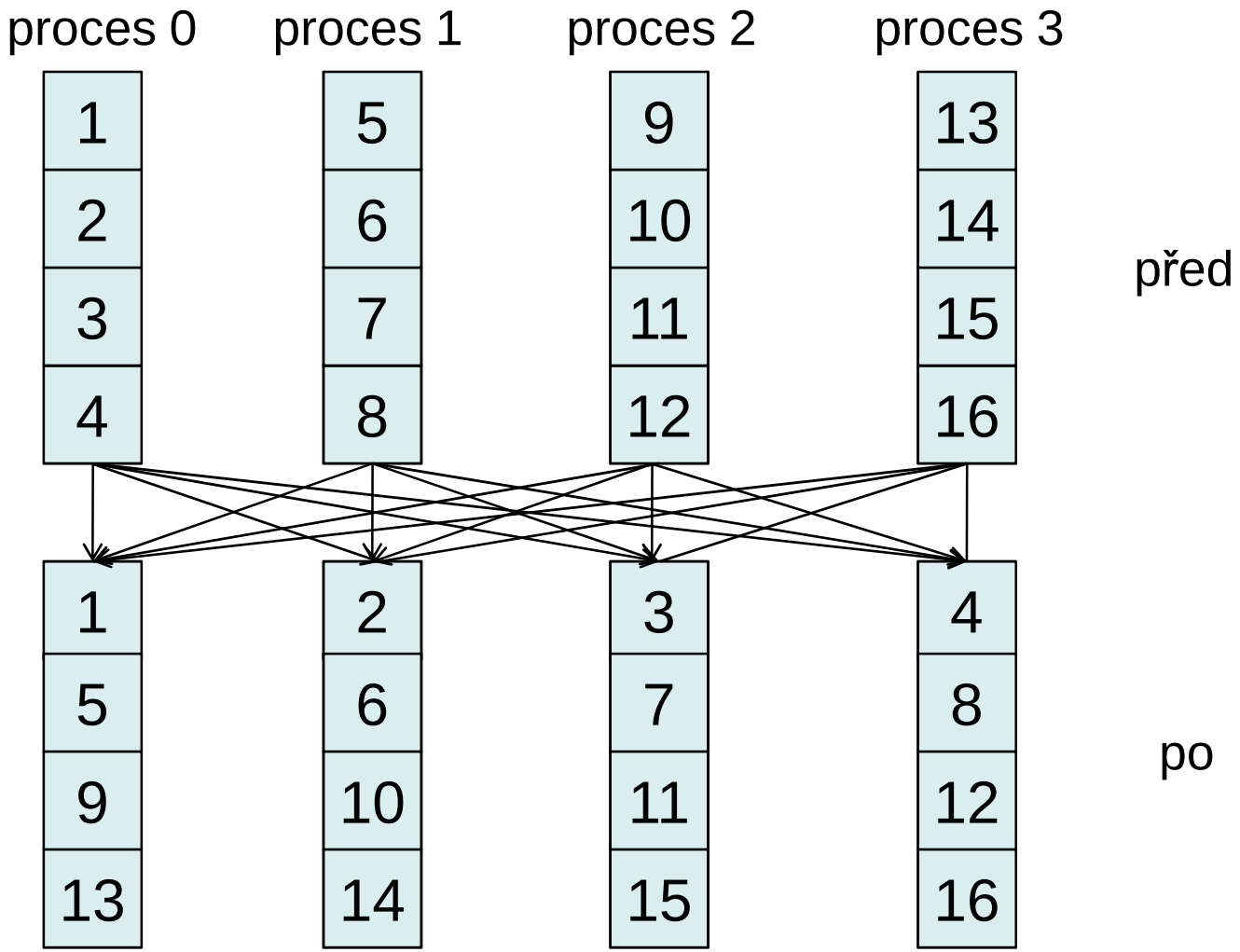
# Návrh paralelního programu – komunikace

Gather (destination = 1)



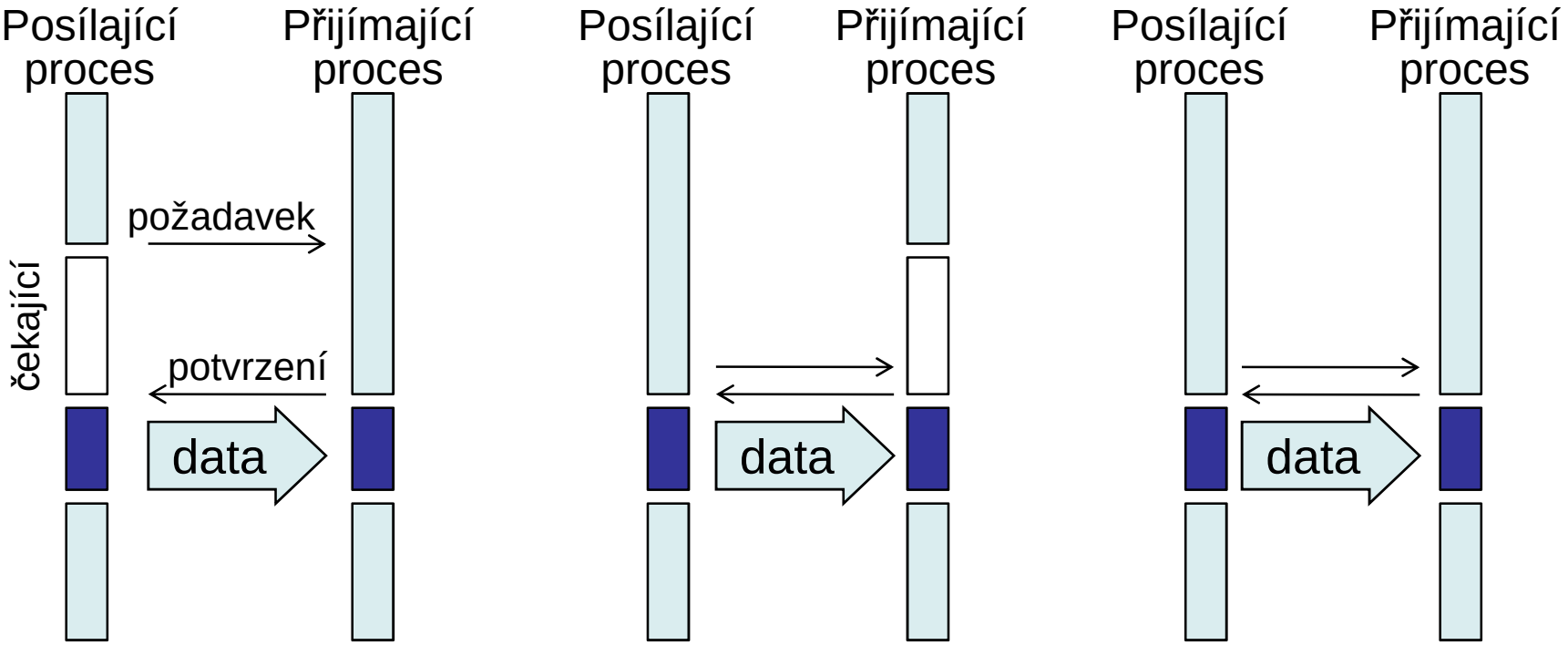
# Návrh paralelního programu – komunikace

All to All:



# Návrh paralelního programu – komunikace

- Blokující komunikace point-to-point <u>bez užití mezipaměti</u>

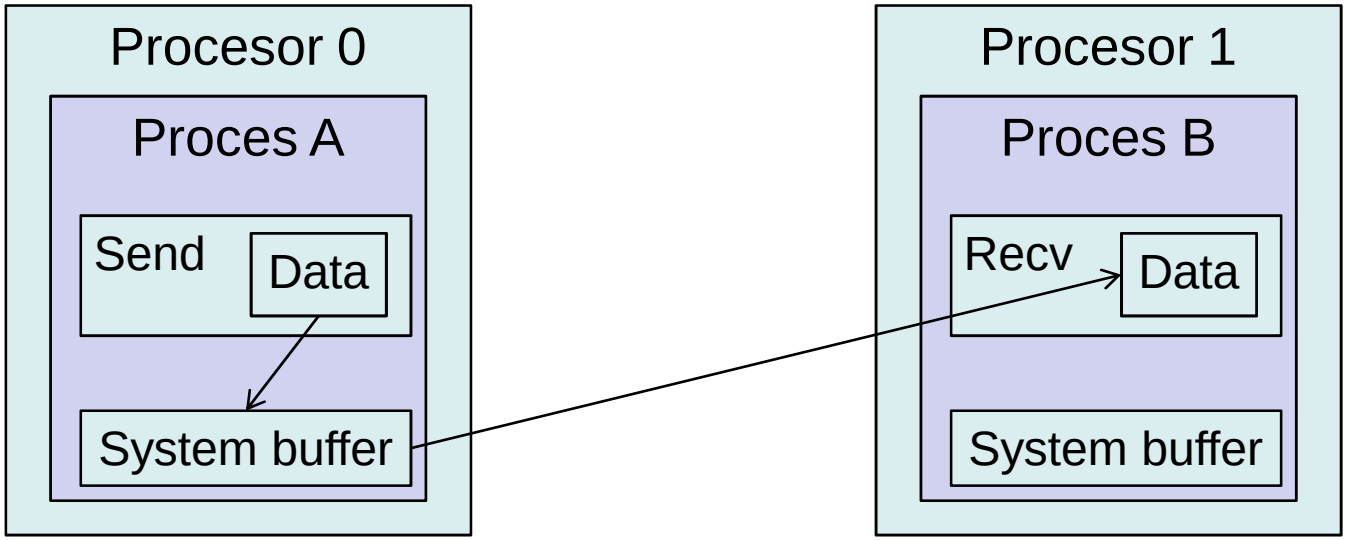


# Návrh paralelního programu – komunikace

- **Dead-lock** při point-to-point blokující komunikaci bez užití mezipaměti.

| P0: | P1: |
|---|---|
| send() | send() |
| recieve() | recieve() |

- Řešením je užití mezipamětí..
  Aplikační mezipaměť (application buffer) a systémová mezipaměť (system buffer) (skrytá programátorovi).



# Návrh paralelního programu – komunikace

**Blokující** komunikace

- Odesílání je ukončeno (návrat z rutiny), až když je aplikační buffer opět možné volně použít (systémový buffer nemusí být použit – v tom případě je odesílání implementováno jako synchronní).
- Synchronní odesílání – vše jak předchozí + dokončeno přijímání.
- Bufferované odesílání – data jsou nakopírována do odesílacího buffru – použití při nedostatku prostoru v systémovém buffru.
- Přijímání – blokuje, dokud data nejsou přijata v aplikačním buffru

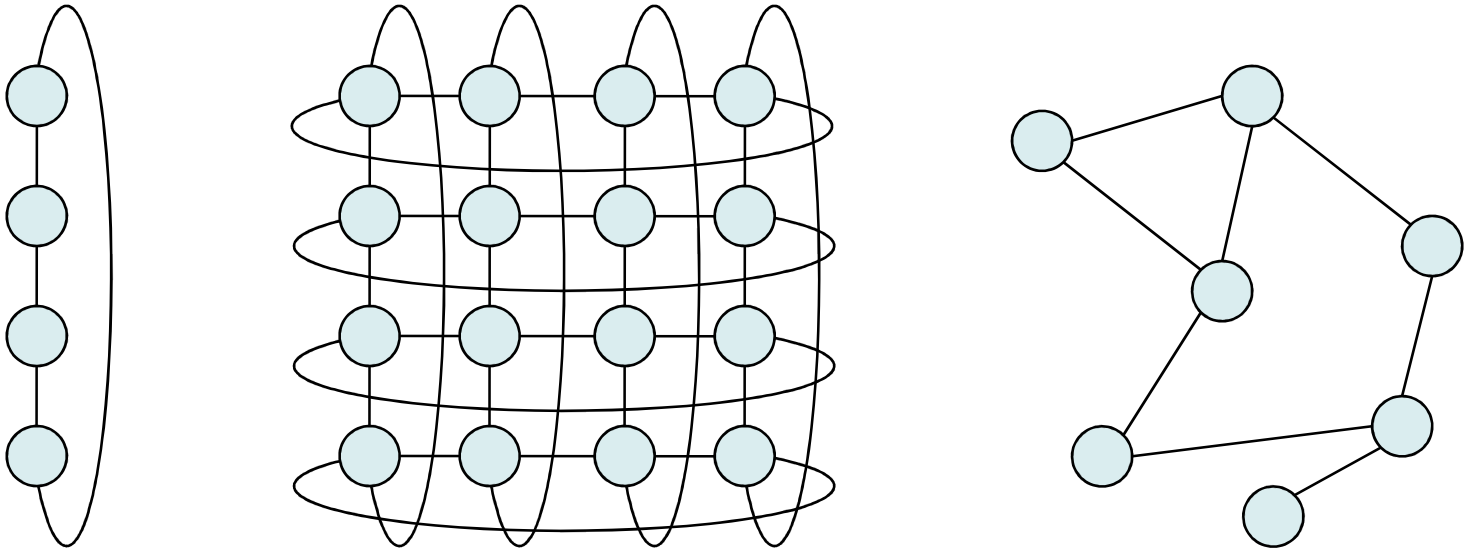
**Neblokující** komunikace

- Odesílaní – program pokračuje bez čekání; aplikační buffer je možné použít až po jeho uvolnění – nutno testovat…!!!
- Synchronní odesílaní – test je úspěšný až po přijetí dat.
- Bufferované odesílání – nutno testovat…
- Přijímání – program pokračuje bez čekání; – nutno testovat…

# Návrh paralelního programu – Virtuální topologie

Virtuální topologie:

- Výhodné při specifických požadavcích na komunikaci.
- Definují sousednost procesů (uzlů) – sousedící procesy mohou navzájem přímo komunikovat.
- Implementace může (závislé na implementaci) optimalizovat proces mapování procesů na fyzické uzly systému…



# Návrh paralelního programu – synchronizace

- Bariéra (Barrier)
  - každá úloha se zastaví na bariéře, pokračuje se když všechny dosáhnou bariéry,
  - nová iterace v datově paralelní smyčce..

- Mutex / Lock / Semaphore
  - konflikty přístupu do společné paměti.

- Operátory synchronní komunikace.

# Úkol k zamyšlení

Je daný program. Třeba využít maximální stupeň paralelizmu mezi 16 instrukcemi, předpokládajíc, že konflikty mezi zdroji a funkčními jednotkami nejsou možné. Všechny instrukce se vykonávají za jeden strojový cyklus. Zanedbávají se všechny ostatní režie.

a) Nakreslit programový graf se 16 uzly za účelem vizualizace vztahů mezi těmito 16 instrukcemi.

b) Použít troj-výstupový superskalární procesor na vykonání tohoto programu za minimální čas. Procesor může za jeden strojový cyklus vydávat jednu instrukci přístupu do paměti (*Load* alebo *Store*, ale ne obě), jednu instrukci *Add/Sub* a jednu instrukci *Mul*.

c) Program realizovat na dvoj-procesorovém systému, přičemž každý procesor je troj-výstupový superskalární procesor.
Program rozčlenit na dvě vyvážené poloviny. Vypracovat optimální rozvržení paralelního vykonání rozděleného programu dvěma procesory v minimálním čase.

# Úkol k zamyšlení

| | | |
|---|---|---|
| 1: | Load R1, A | /R1 ← Mem(A)/ |
| 2: | Load R2, B | /R2 ← Mem(B)/ |
| 3: | Mul R3, R1, R2 | /R3 ← (R1) x (R2)/ |
| 4: | Load R4, D | /R4 ← Mem(D)/ |
| 5: | Mul R5, R1, R4 | /R5 ← (R1) x (R4)/ |
| 6: | Add R6, R3, R5 | /R6 ← (R3) + (R5)/ |
| 7: | Store X, R6 | /Mem(X) ← (R6)/ |
| 8: | Load R7, C | /R7 ← Mem(C)/ |
| 9: | Mul R8, R7, R4 | /R8 ← (R7) x (R4)/ |
| 10: | Load R9, E | /R9 ← Mem(E)/ |
| 11: | Add R10, R8, R9 | /R10 ← (R8) + (R9)/ |
| 12: | Store Y, R10 | /Mem(Y) ← (R10)/ |
| 13: | Add R11, R6, R10 | /R11 ← (R6) + (R10)/ |
| 14: | Store U, R11 | /Mem(U) ← (R11)/ |
| 15: | Sub R12, R6, R10 | /R12 ← (R6) – (R10)/ |
| 16: | Store V, R12 | /Mem(V) ← (R12)/ |