PRL: Paralelní a distribuované algoritmy 1

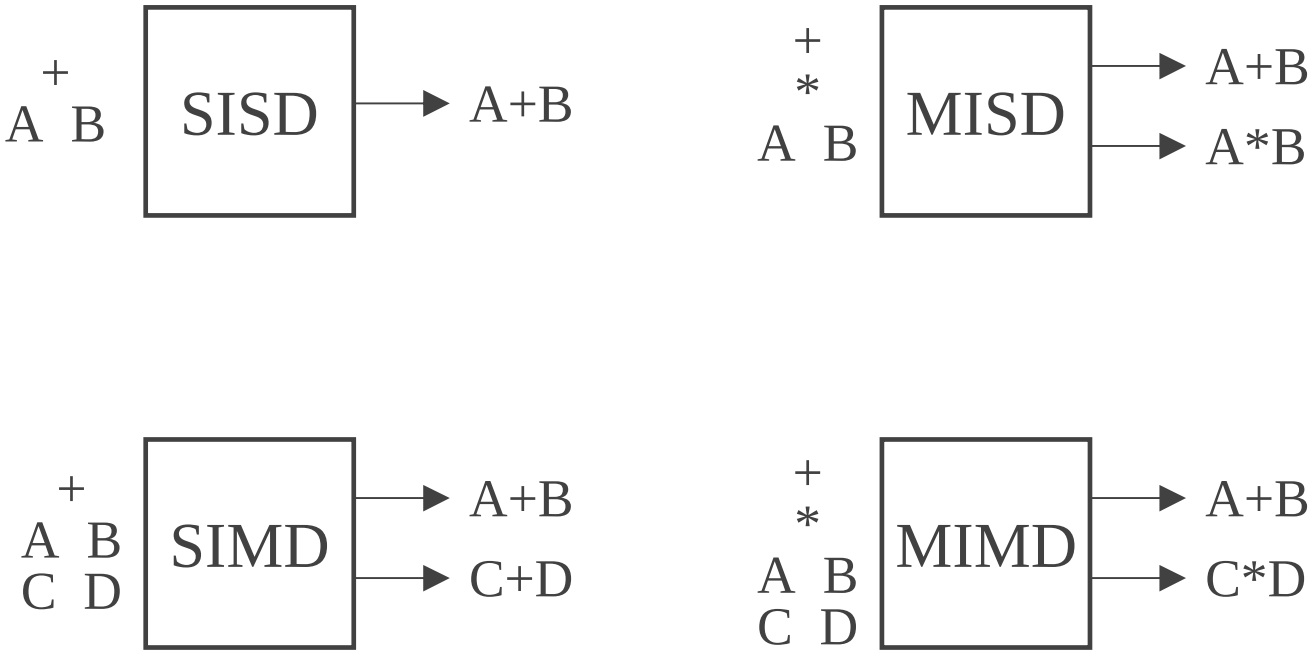
Petr Hanáček

Cor pre 2006/7, pre 2007/8, po 2007/8

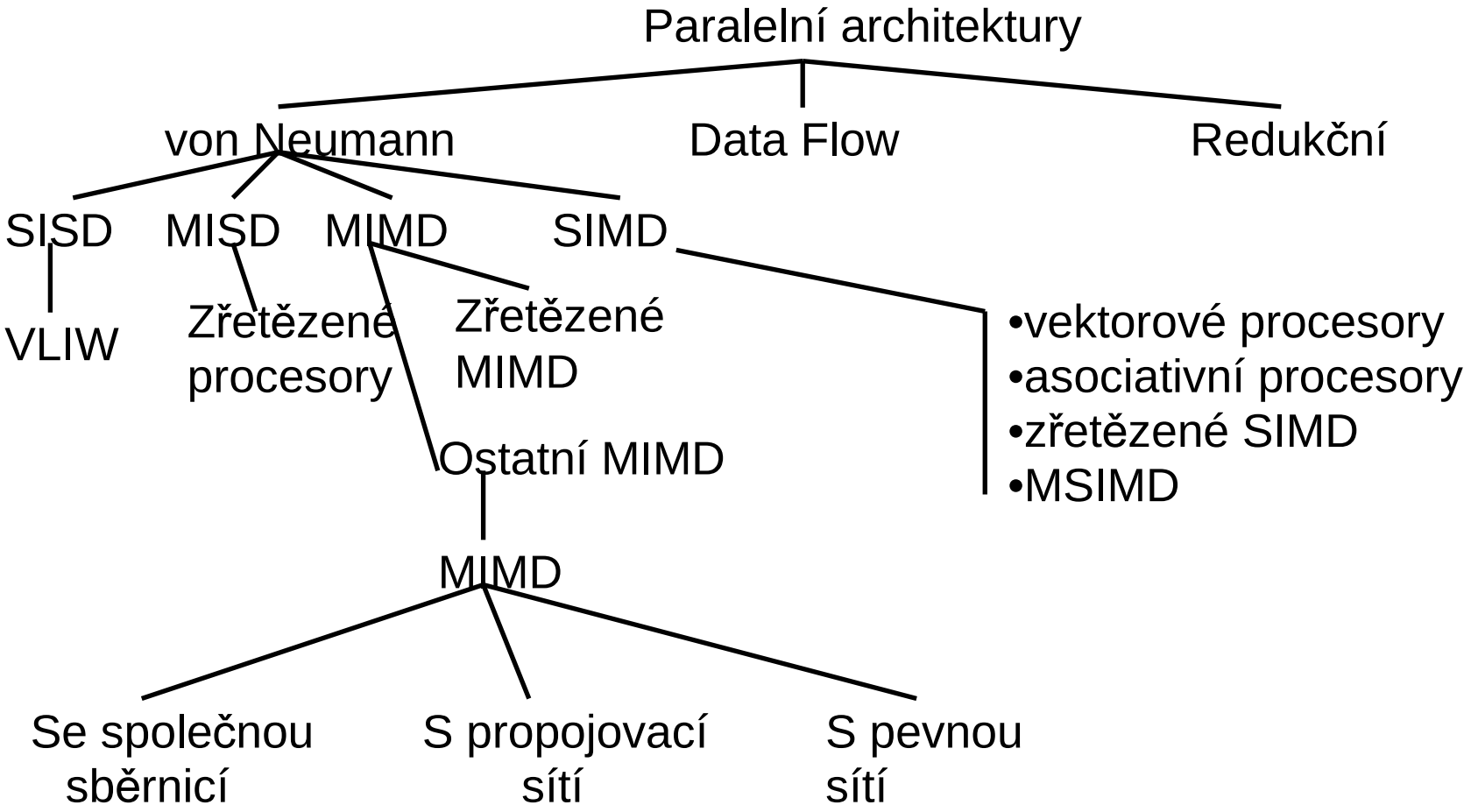
Paralelní zpracování - architektura

- Flynnova klasifikace :
  - SISD - konvenční
  - SIMD - vektorové
  - MISD - řetězové
  - MIMD
    - » multiprocesory (sdílená paměť)
    - » multicomputery (předávání zpráv)
      - Distribuované systémy

Schopnosti těchto kategorií



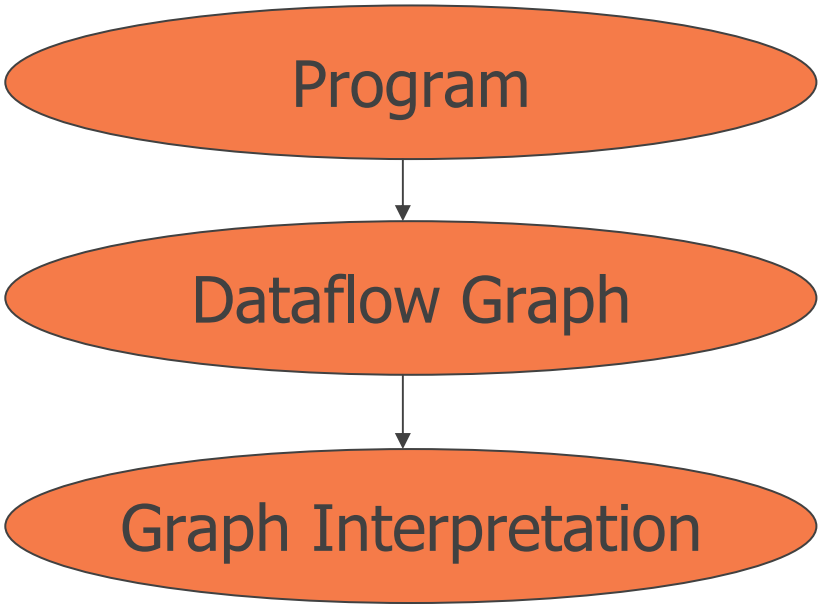
Paralelní architektury



Ne-von Neumannovské architektury

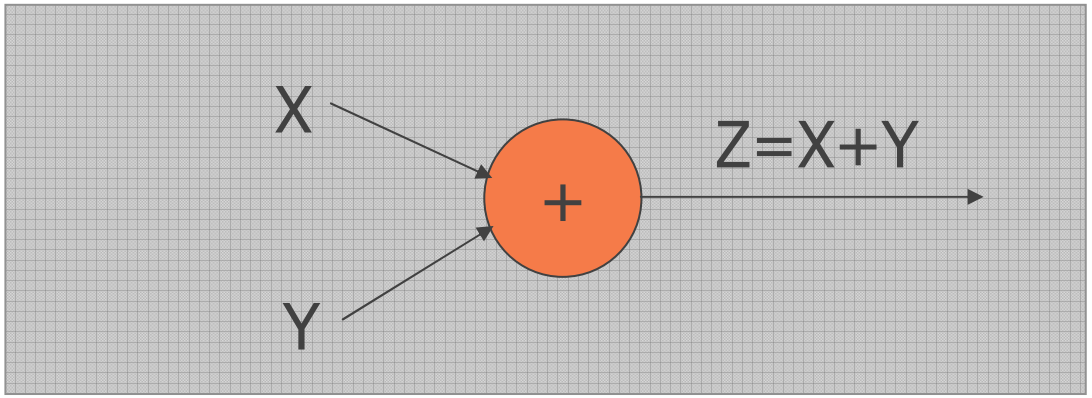
Architektura Dataflow

- není von Neumannovská architektura (nemá program a PC)
- provádí interpretaci grafu toku dat

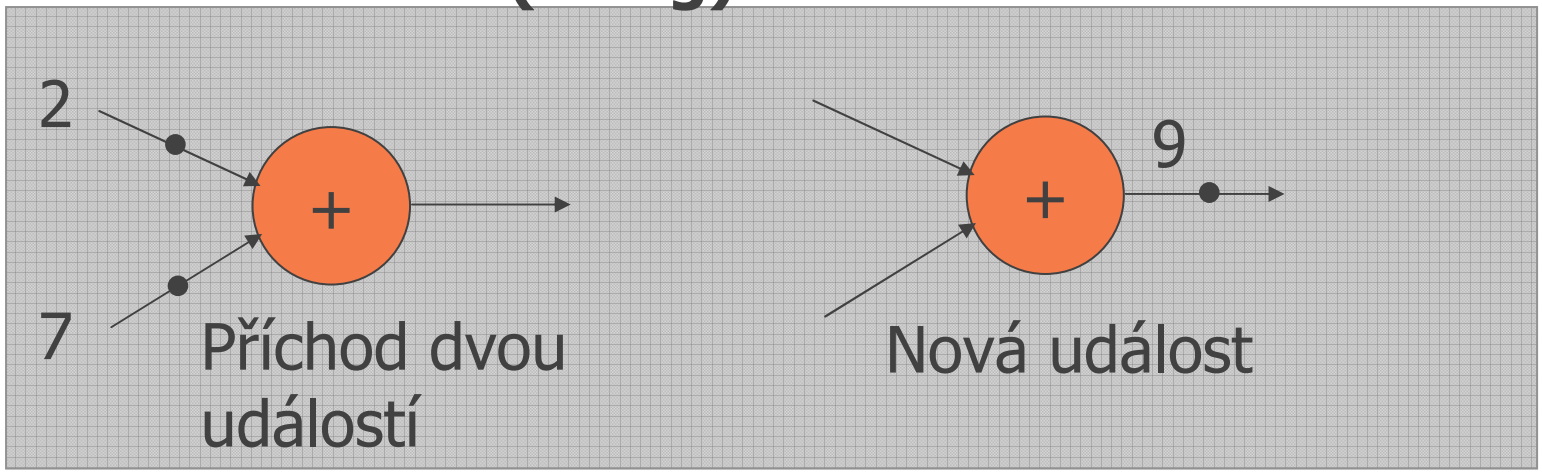


Graf toku řízení

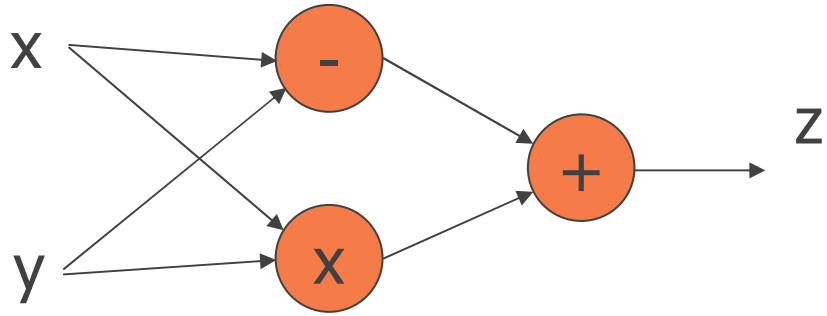
Uzel grafu



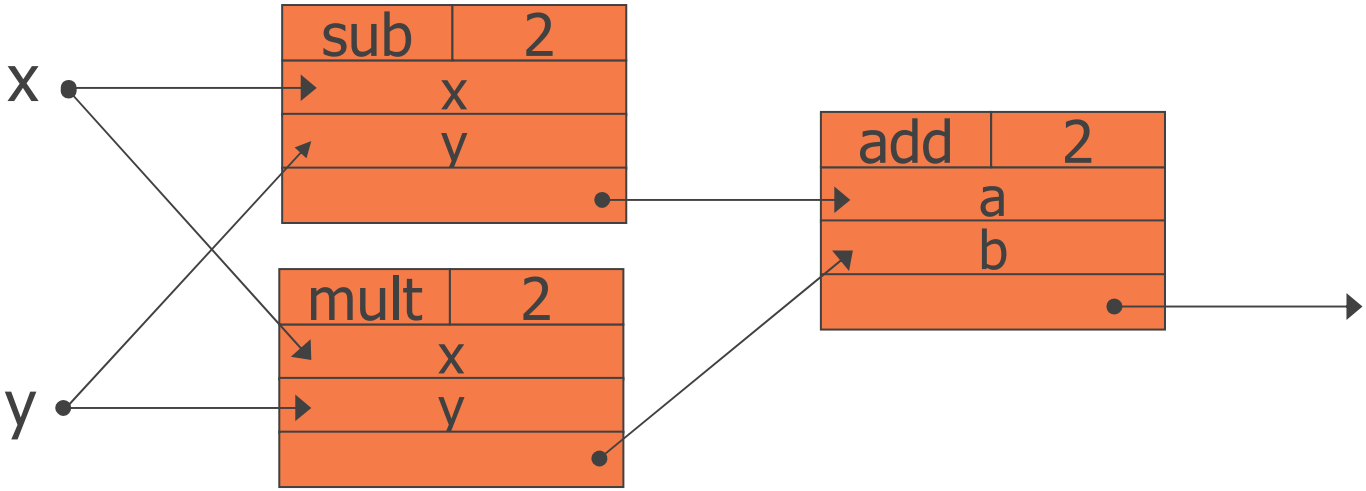
Provedení uzlu (firing)



Příklad



Uložení grafu toku dat v paměti aktivit



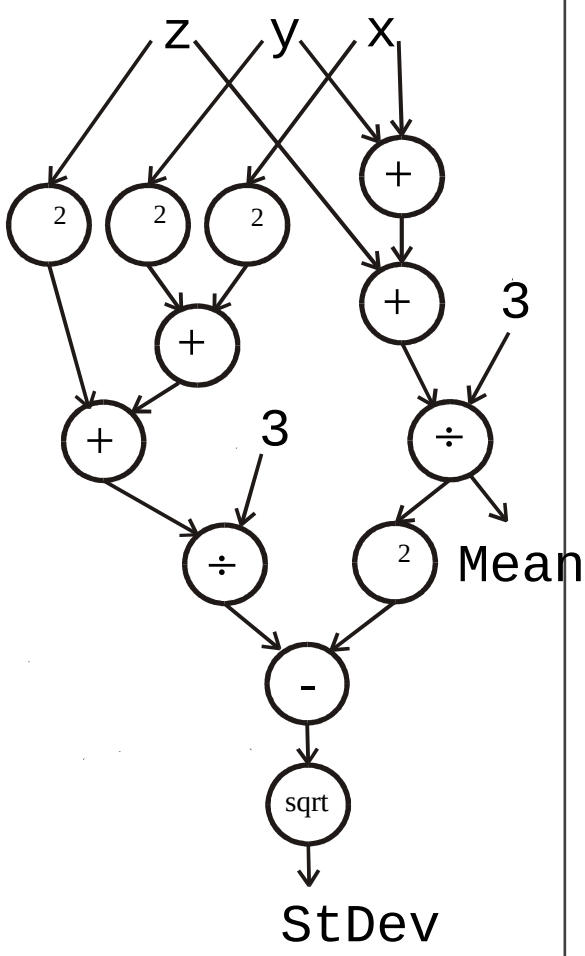
Vhodné jazyky

- Hlavní požadavek
  - The single-assignment rule
  - Proměnná se může vyskytovat na levé straně přiřazení pouze jedenkrát
- Příklady:
  - VAL, Id, LUCID
- Program je převeden do grafu toku dat
  - Orientovaný graf
  - Uzly představují instrukce
  - Hrany představují datové závislosti mezi instrukcemi

Příklad v jazyce VAL

- Funkce Stats: počítá průměr a standardní odchylku tří vstupních hodnot

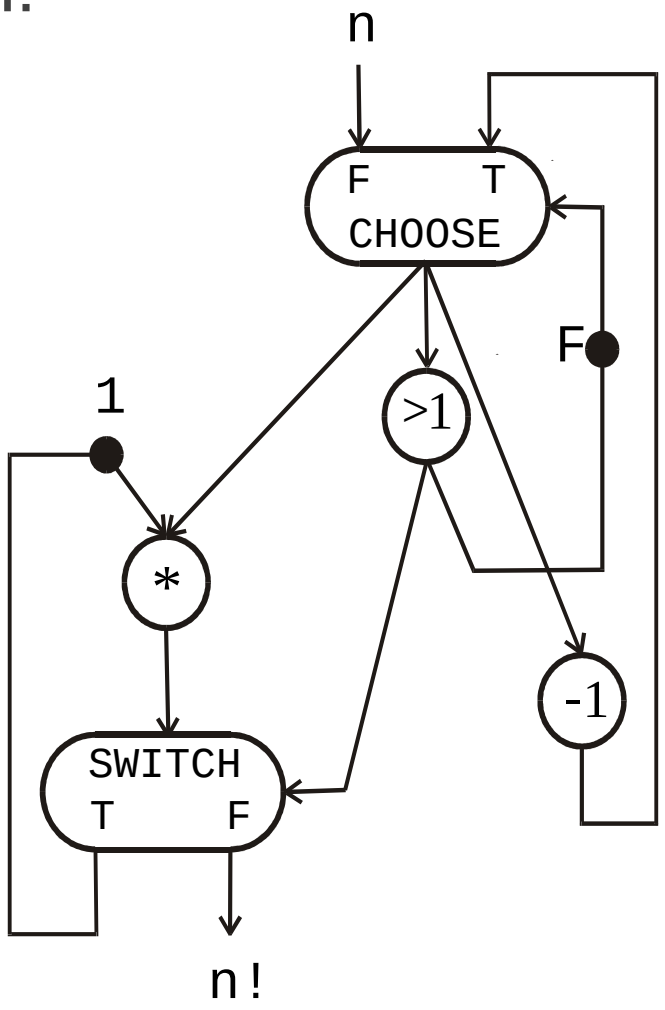
```
function Stats(x,y,z: real returns real,real);
 let
    Mean := (x + y + z)/3;
    StDev := SQRT((x**2 + y**2 + z**2)/3
                                  - Mean**2);
 in
    Mean, StDev
 endlet
endfun
```



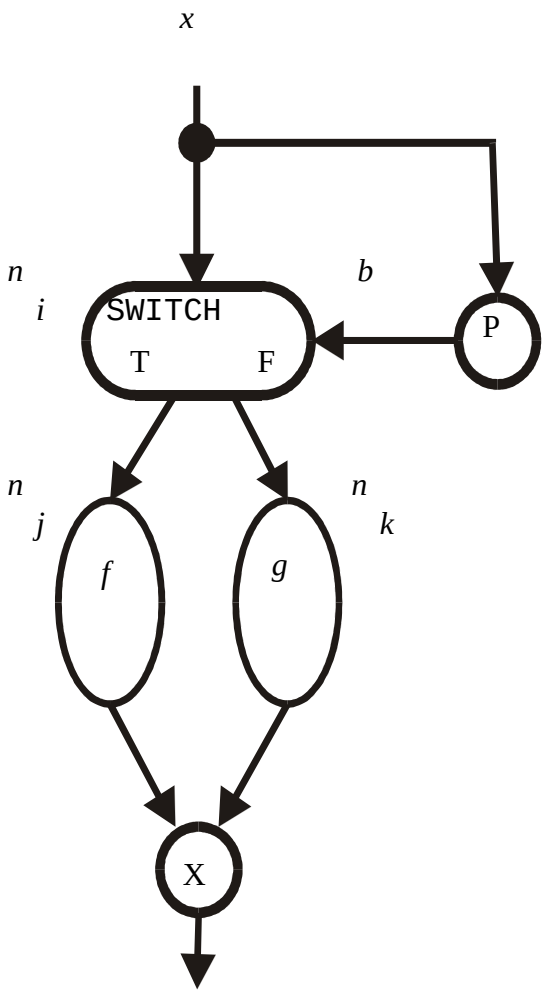
Příklad v jazyce Id

- Program počítá faktoriál n!
  čísla n

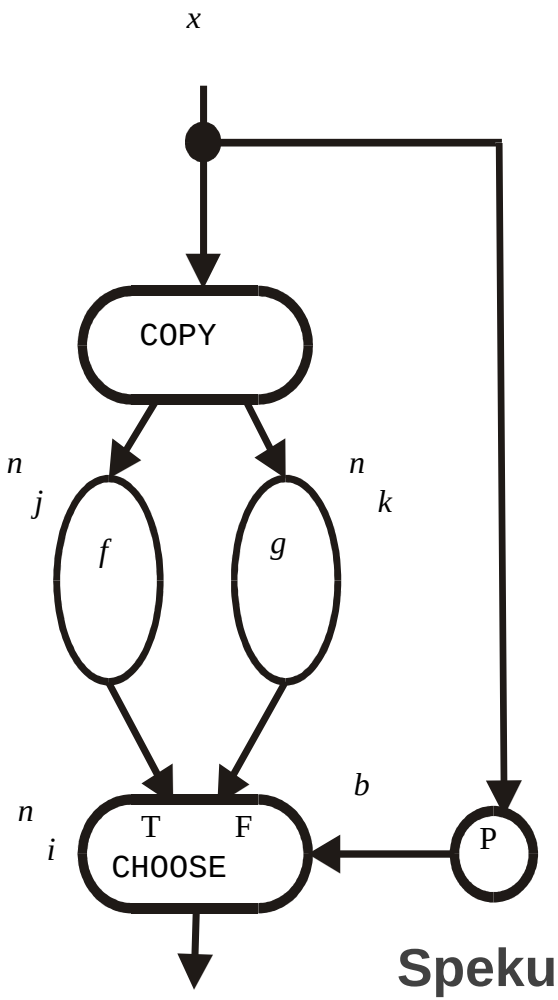
```
( initial j <- n; k <- 1
  while j > 1 do
   new j <- j - 1;
   new k <- k * j;
  return k )
```



Implementace podmíněných výrazů („skoků“)

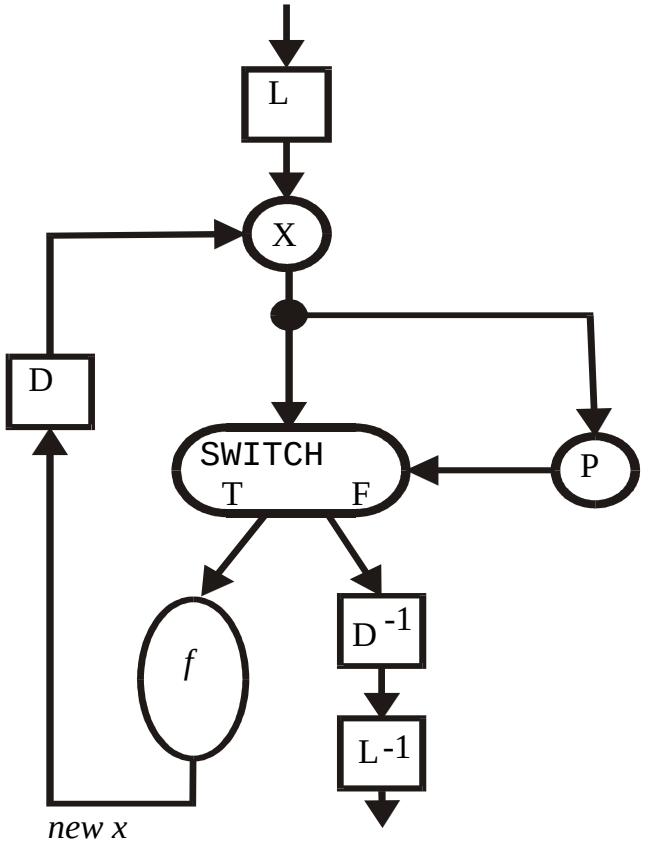


Normální



Spekulativní

Implementace cyklů



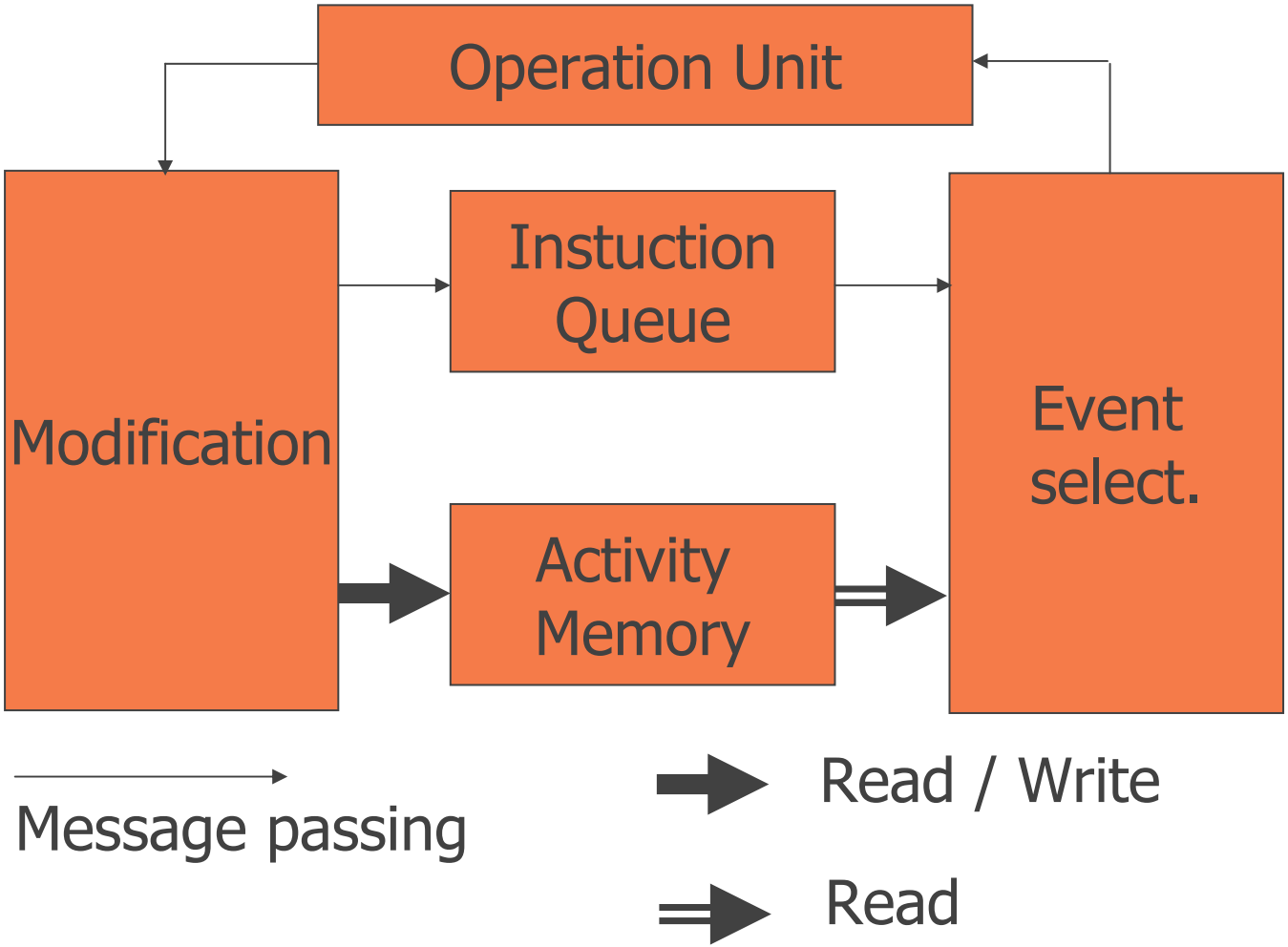
L: inicializace – nový kontext cyklu

D: inkrementace proměnné cyklu

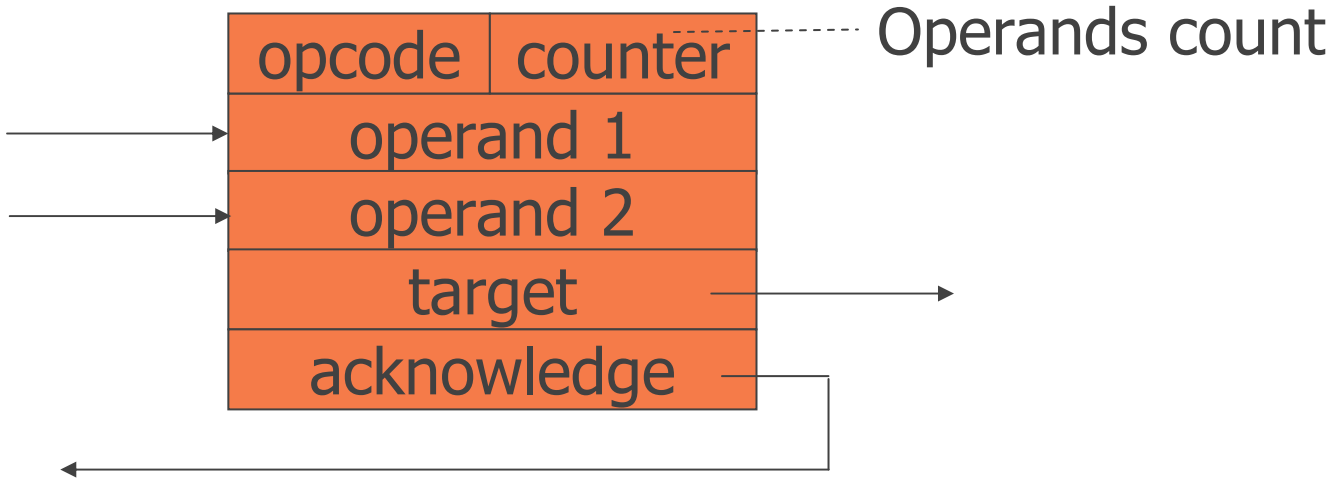
D^{-1}: nastavení proměnné cyklu na 1

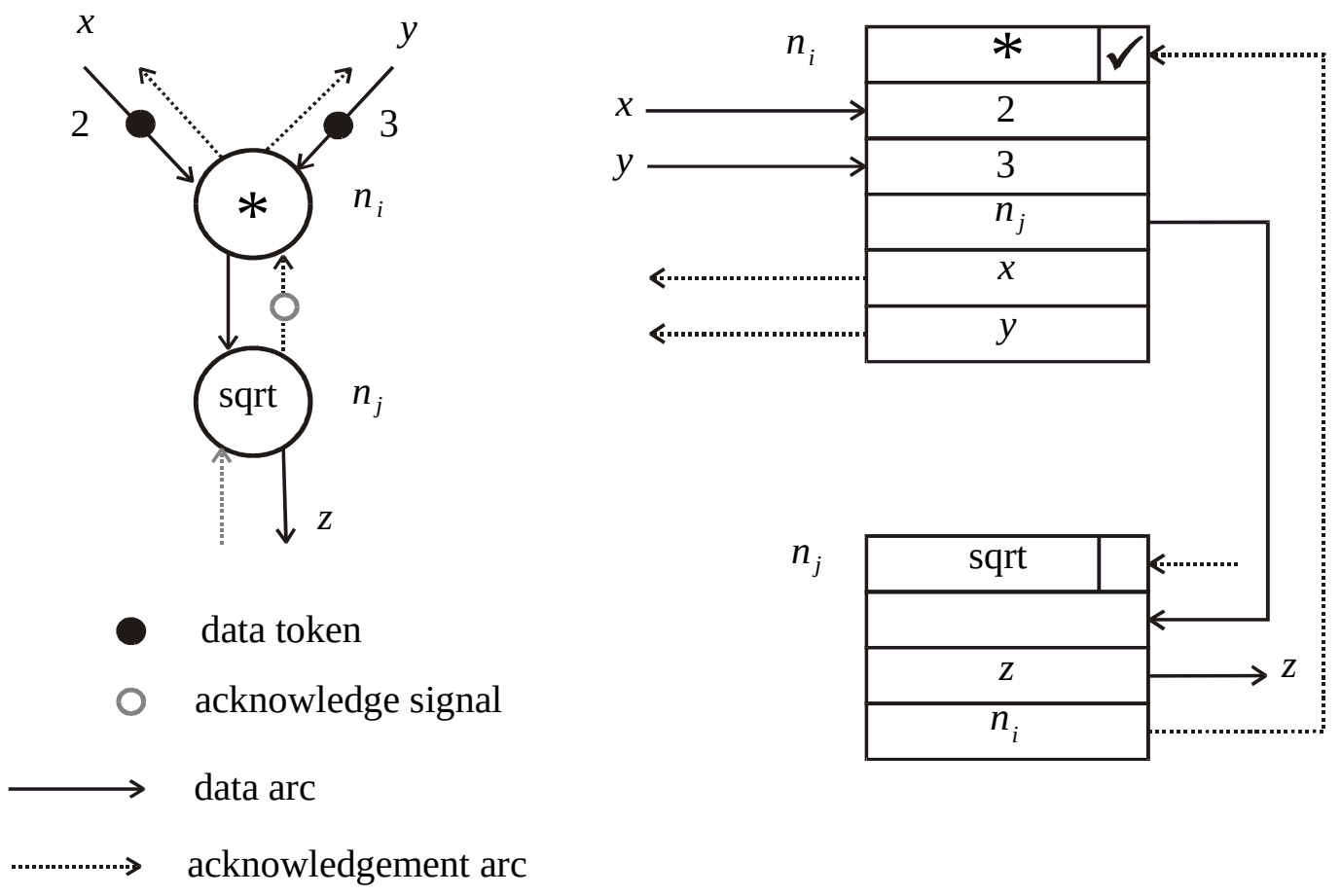
L^{-1}: nastavení původního kontextu

Dataflow procesor



- Request
  - <opcode, operand, target>
- Result
  - <result, target>
- Zdroj paralelismu - větší počet operačních jednotek, které pracují paralelně
- Problém - vyskytnou-li se na jedné hraně grafu dvě události, může dojít ke změně jejich pořadí → potvrzení



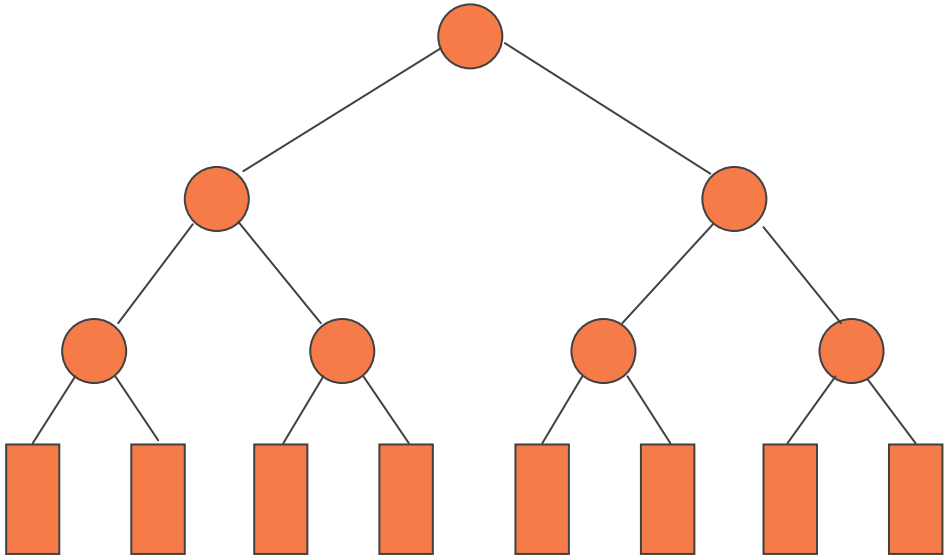

x
y
2
3
*
n_i
sqrt
n_j
z
data token
acknowledge signal
data arc
acknowledgement arc
n_i
*
x
y
2
3
n_j
x
y
n_j
sqrt
z
n_i
z

Redukční počítač

- není von Neumannovská architektura
- redukce - náhrada části výrazu jeho významem
- 2 * 3 → 6
- zpravidla se používá redukce řetězců
- Příklad:
- výraz
  - (x*y)+(x-y)
- Se zapíše
  - +<(*<x y>)(-<x y>)> <…> - sekvence
- Redukce pro x=3 y=2
  - +<(*<3 2>)(-<3 2>)>
  - +< 6 1 >
  - 7

Použitá architektura (stromová)

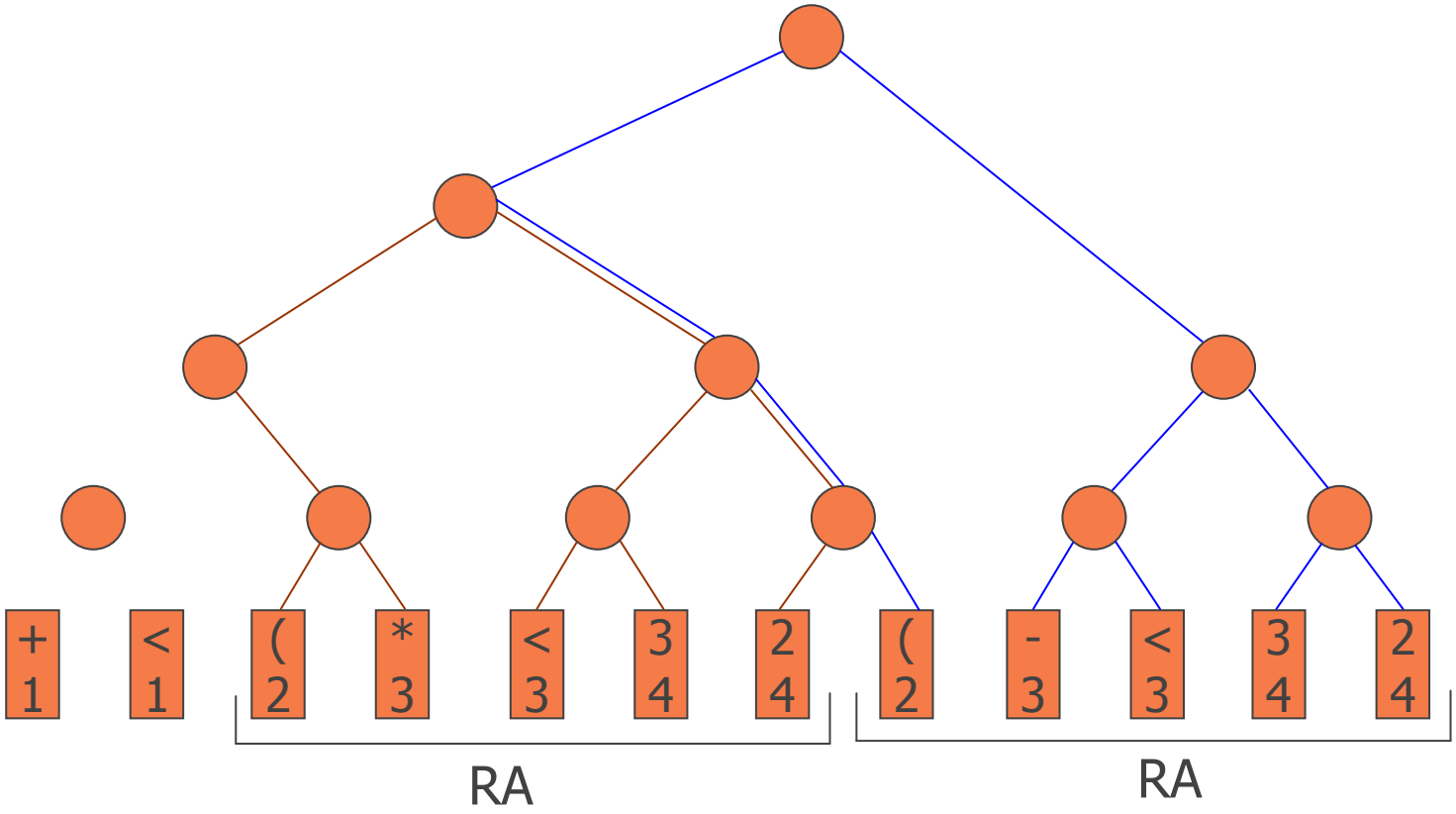
- **T** uzly (procesory + komunikace)
- **L** uzly (paměti)



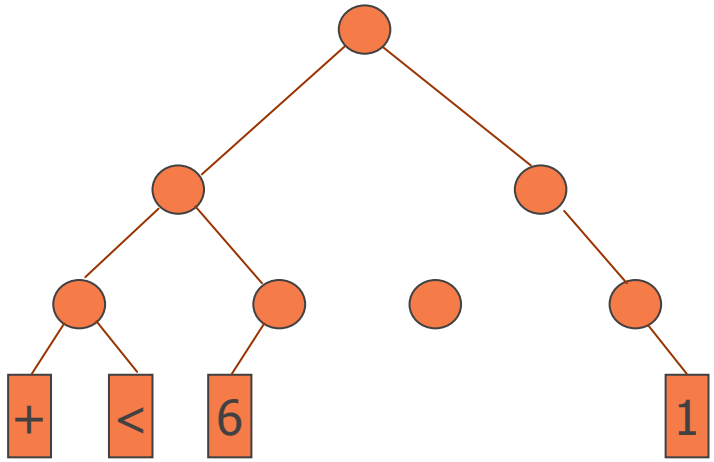
fáze :

- 1. rozdělení L - uzlů na redukční pole
- 2. provedení redukce
- 3. posuv v paměti L-uzlů

Redukce – fáze I



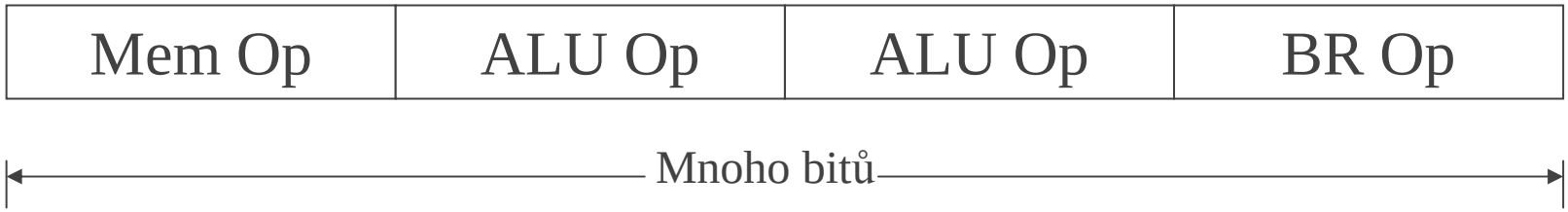
Redukce – fáze II a III

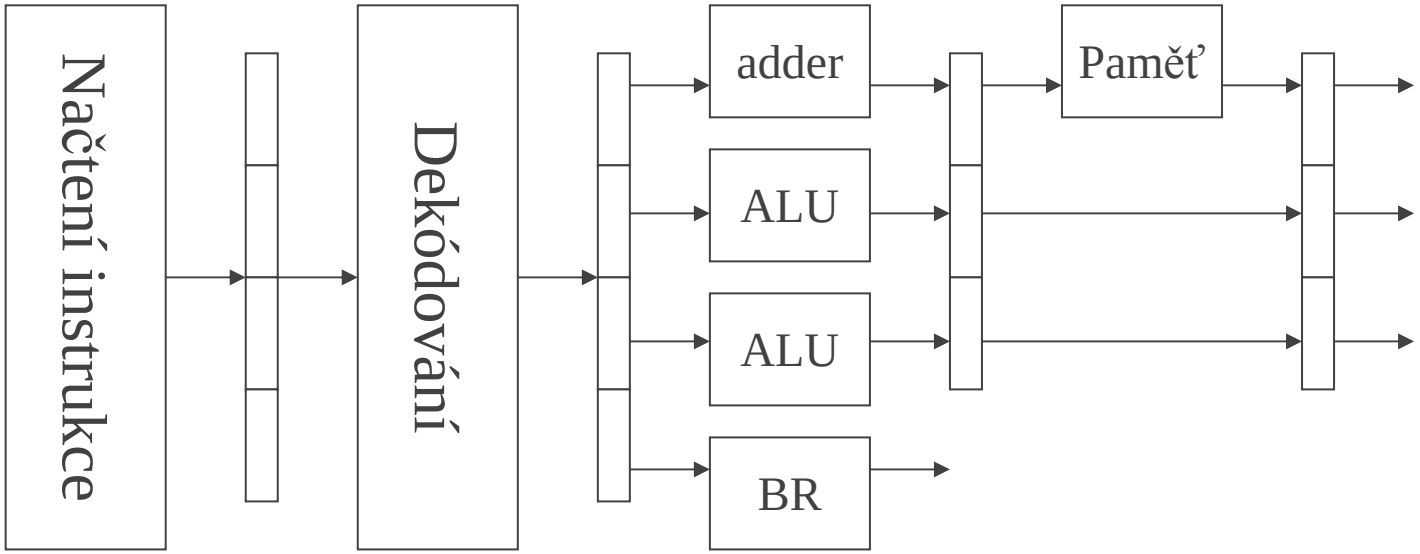


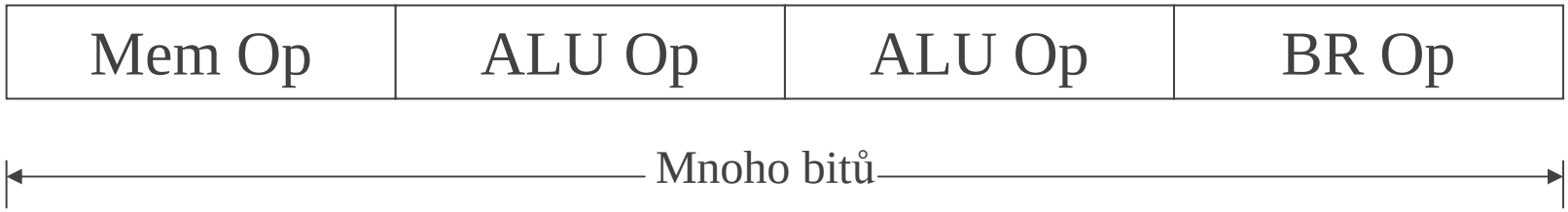
Von Neumannovské architektury

VLIW - Very Long Instruction Word

- jediný tok řízení, který řídí všechny procesory
- každá instrukce - samostatné pole s operačním kódem pro všechny procesory
  - jediný PC, který prochází program
  - instrukce jsou prováděny sekvenčně
  - instrukce také určují, jak probíhá komunikace mezi procesory. Jsou v nich uloženy informace o odesílateli, příjemci a velikosti dat. Tyto informace s ohledem na časování generuje kompilátor

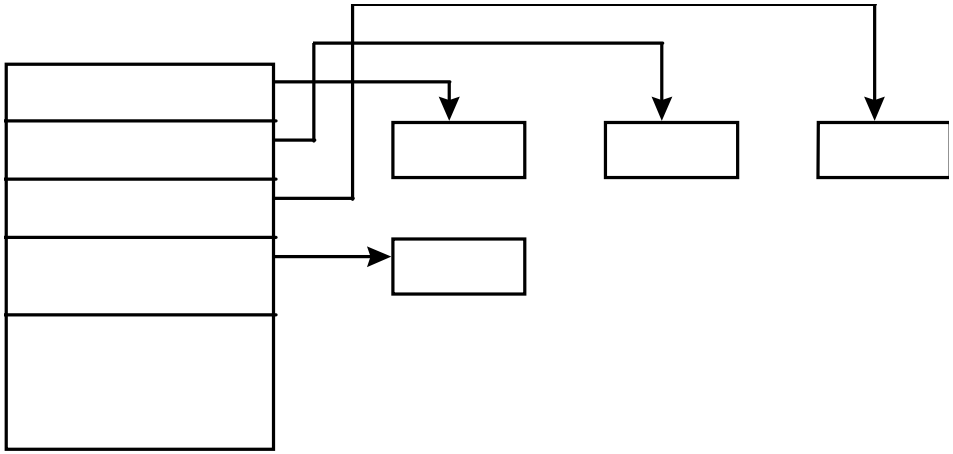



Načtení instrukce
Dekódování
adder
ALU
ALU
BR
Paměť


Mem Op
ALU Op
ALU Op
BR Op
Mnoho bitů

VLIW - Kompilace

- Skládání instrukci VLIW z několika jednoduchých instrukcí



Nejlepší: Pracují všechny procesory

Nejhorší: Pracuje jeden procesor

VLIW – Vlastnosti

- Výhody
  - Jednoduchý hardware
  - Dobře škálovatelné
- Nevýhody
  - Podmíněné skoky (když jedna instrukce provede skok, ostatní instrukce mají problém)
  - Problém toku dat (instrukce zpracovávané v jednom kroku nemohou navzájem používat své výsledky)
    - » Některé závislosti nejsou staticky rozpoznatelné v době překladu (např. aliasing ukazatelů, oddělený překlad, dynamické sestavování) => překladač musí být konzervativní a výsledná výkonnost je nižší než při schopnosti rozpoznat všechny závislosti
    - » Některé latence nejsou odhadnutelné v době překladu (např. latence instrukcí LOAD a STORE kvůli případnému Hit nebo Miss v cache) => některé VLIW používají softwarově řízenou lokální paměť místo datové cache; jinde překladač předpokládá 100% Hit-Rate
  - Velikost programu - synchronizační NOPy jsou u VLIW navíc
  - VLIW nejsou softwarově zpětně kompatibilní jako superskalární procesory (i když existují cesty, jak toho dosáhnout)

Vztah VLIW a superskalárních proc.

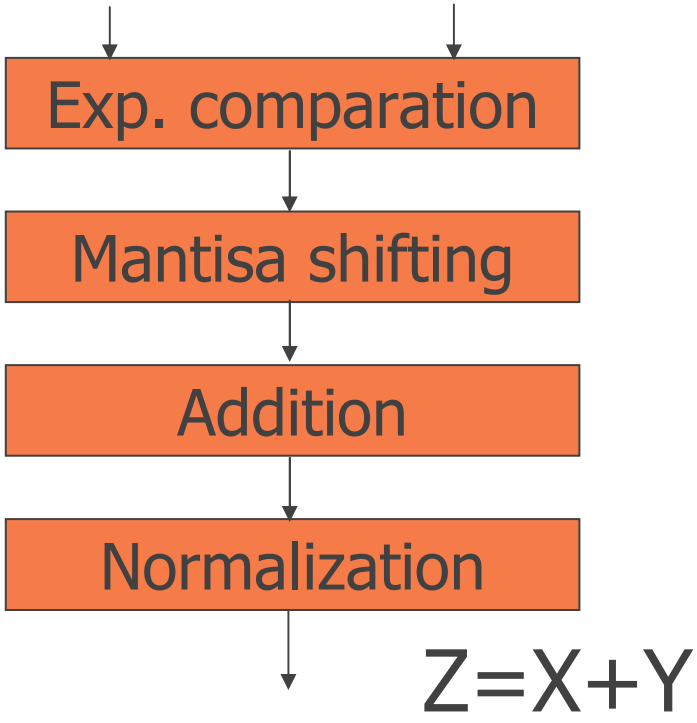
- Statické superskalární procesory (Static, In-Order Superscalar)
  - Zpracovávají více instrukcí paralelně ale v programovém pořadí
  - Typická šířka 3-4 instrukce
  - Paralelní provádění může zahájit pouze limitovaná kombinace typů instrukcí („párovatelné instrukce“)
    - » Velmi závislé na kvalitním překladači
- VLIW procesory
  - Instrukce obsahuje více paralelních operací
  - Konflikty detekuje a řeší překladač (téměř výlučně)
  - Šířka instrukcí typicky 6-10 paralelních operací
- Dynamické superskalární procesory (Dynamic, Out-Of-Order Superscalar)
  - Zpracovává více instrukcí paralelně i mimo programové pořadí
  - Dnes typicky podporuje spekulativní provádění instrukcí za skokem a někdy i spekulativní provádění Load/Store instrukcí

MISD - Zřetězené procesory

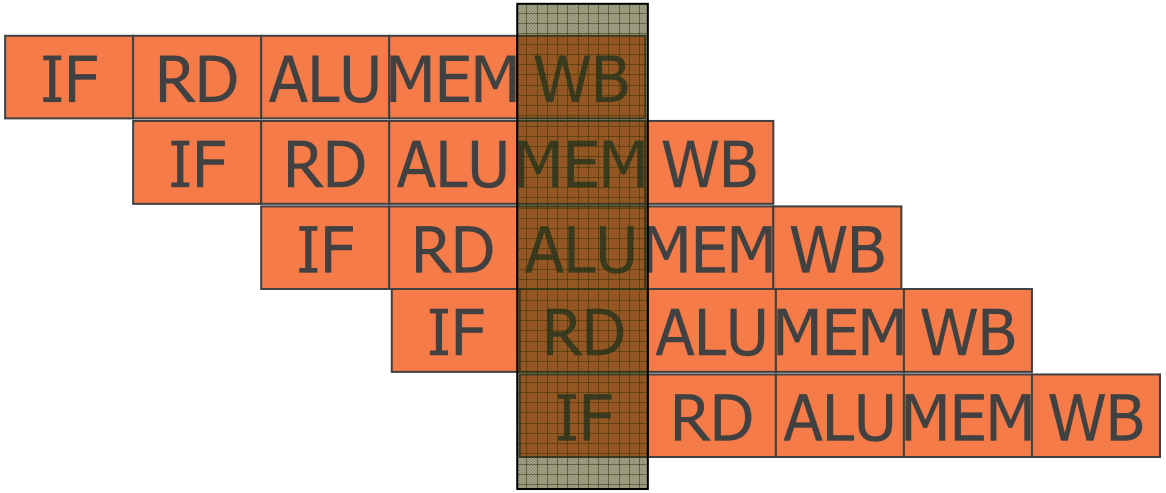
- Lineárně propojené procesory
- Řešení úloh s proudovým charakterem
- Data procházejí postupně jednotlivými procesory

Příklad MISD

- Sčítání reálných čísel
- $X=m_x*2^{ex}$ $Y=m_y*2^{ey}$

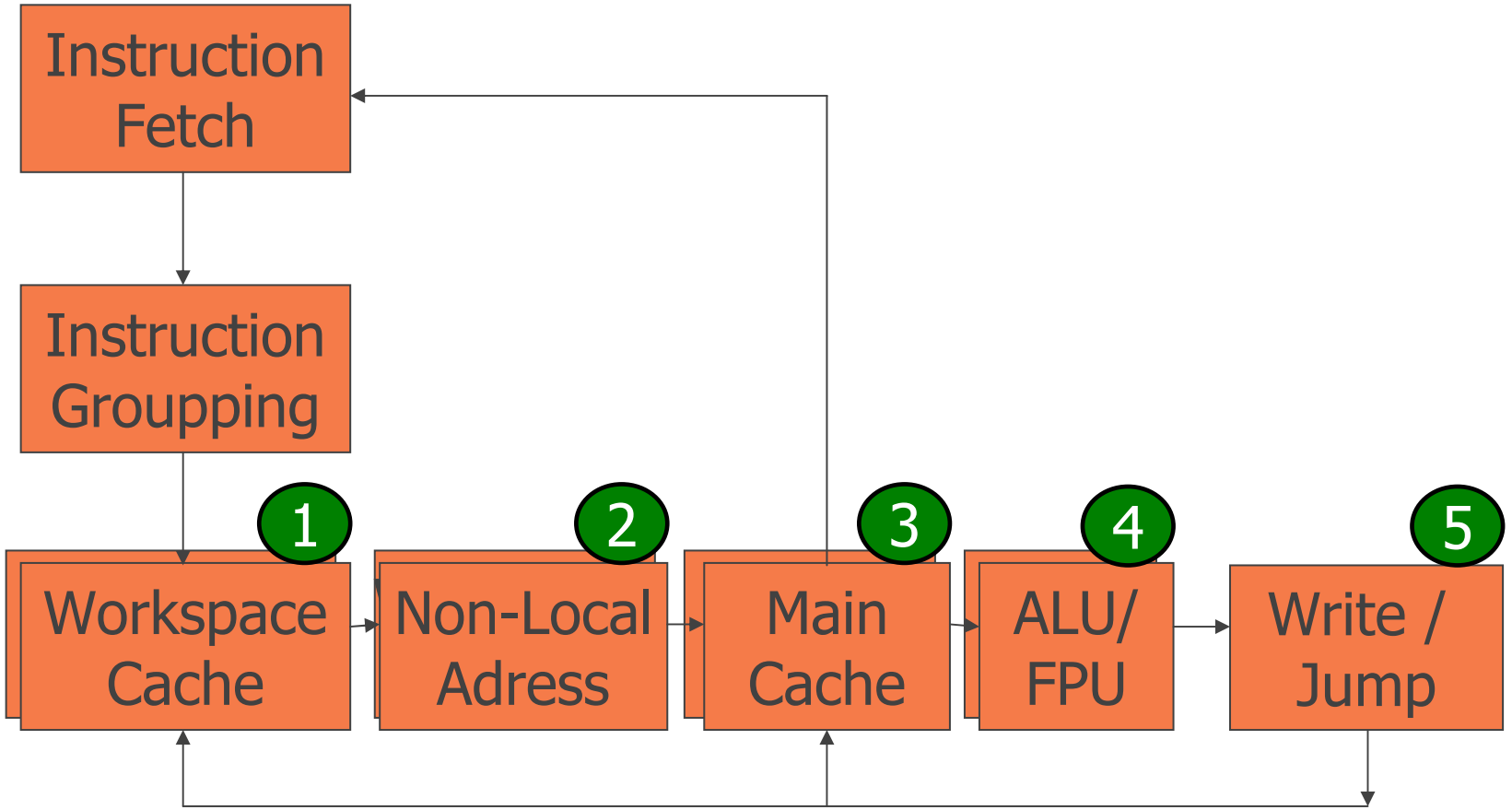


Zřetězené procesory: II



- IF – Instruction Fetch
- RD - Read
- ALU - Compute
- MEM – Access Memory
- WB – Write Back

Zřetězené procesory: III



Zřetězené procesory: III

a[i+1]=b[j+15]+c[k+7]

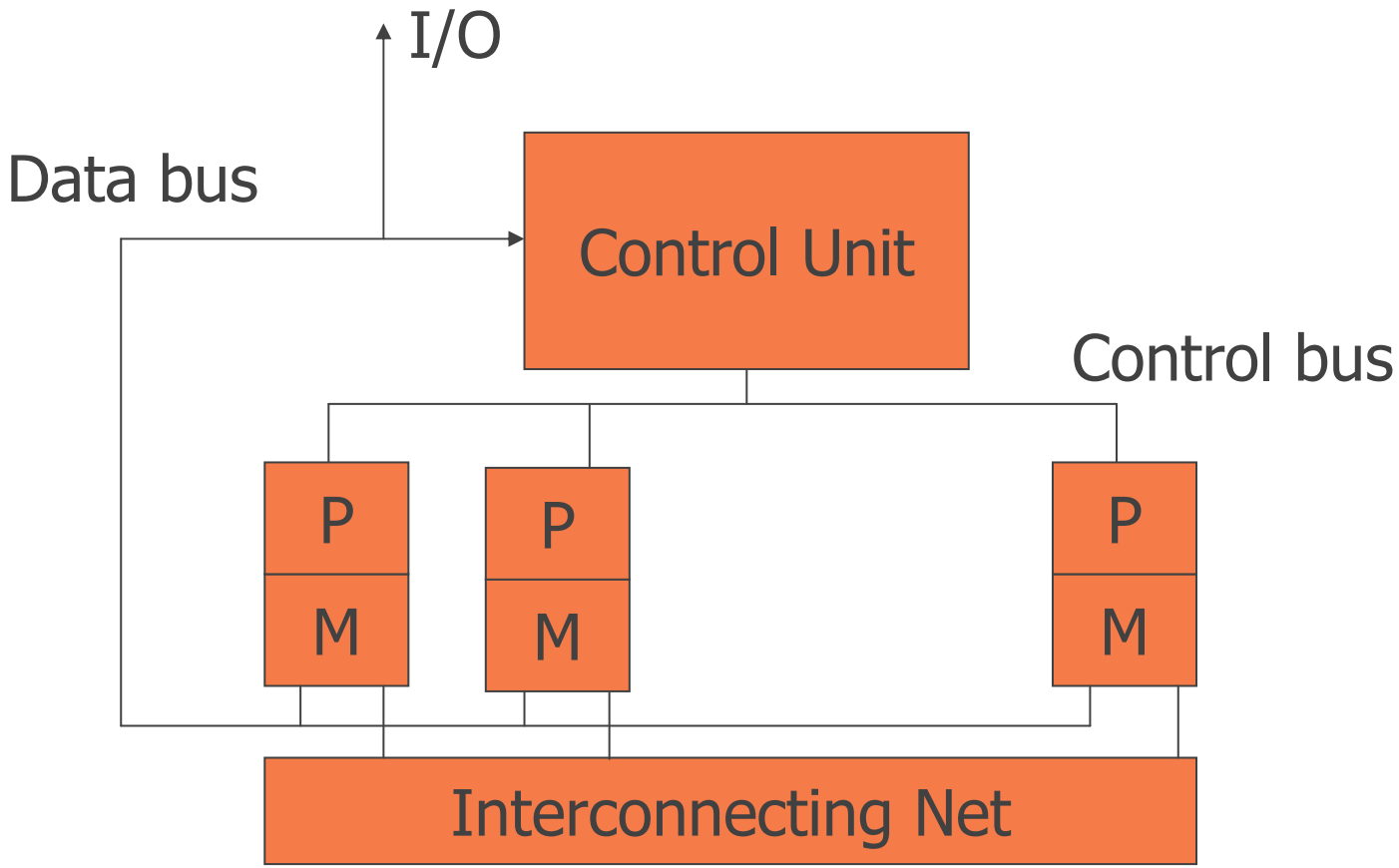
ldl j	1
ldl b	1
wsub	2
ldnl 15	2,3
----------	
ldl k	1
ldl c	1
wsub	2

ldnl 7	2,3
add	4
----------	
ldl i	1
ldl a	1
wsub	2
stnl 1	2,5

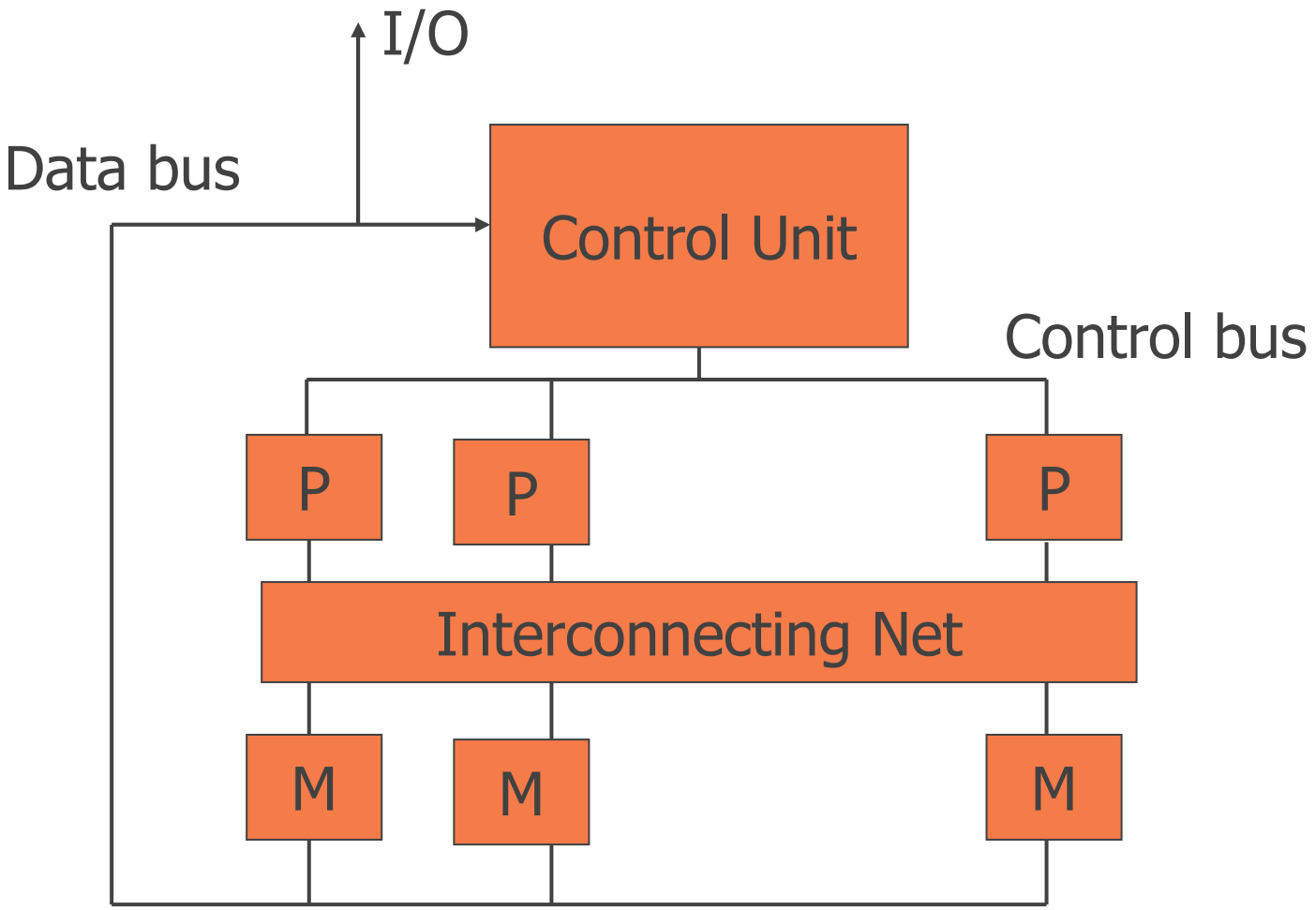
Jiný příklad MISD

- Zjištění, zda číslo Z je prvočíslem
- Máme N procesorů, kde $N = Z^{1/2}$
- Každý procesor se pokouší dělit číslo Z svým dělitelem
- Ve skutečnosti je k dispozici méně procesorů, každý má přiřazenu skupinu dělitelů

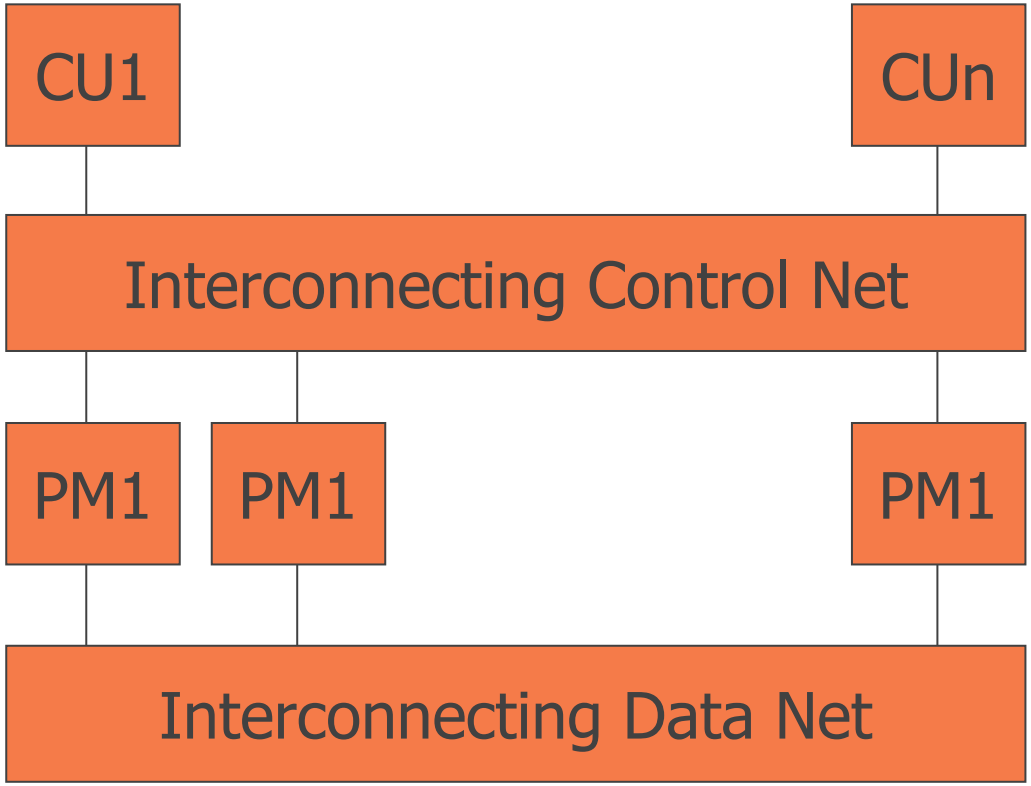
SIMD – Vektorový procesor I



SIMD – Vektorový procesor II



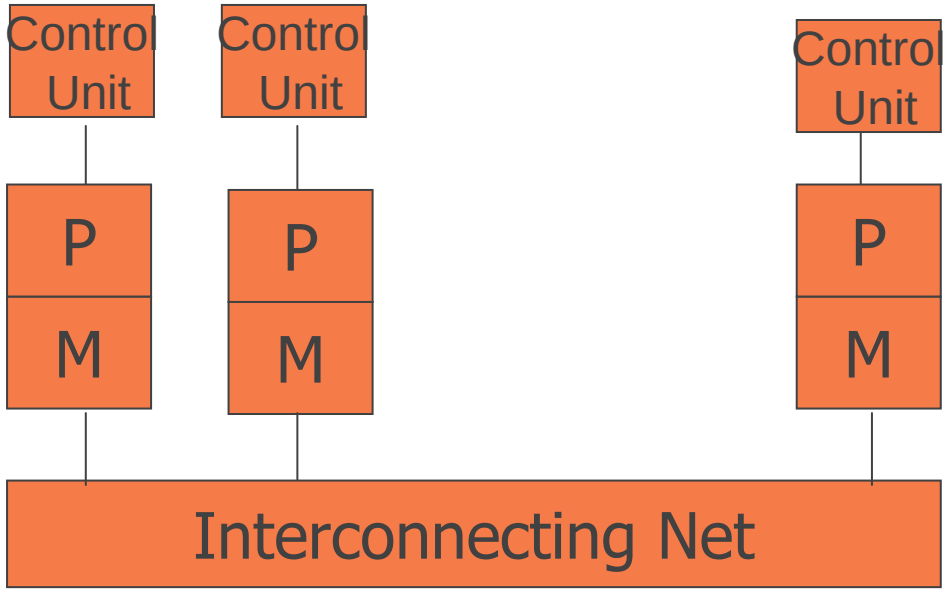
MSIMD – Multiple SIMD



Dělení procesorů na nezávislé skupiny
Dynamické alokování procesorů

SPMD

- SPMD – Same Program, Multiple Data
- všechny procesory provádějí stejný program, ale nezávisle na sobě – bez synchronizace (každý procesor má svůj řadič)



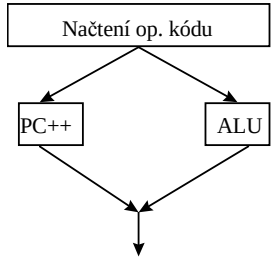
Výhody SIMD

- Jednodušší než MIMD
- Menší nároky na paměť
- Jeden instrukční tok a synchronizace zjednodušuje programy
- Odpadá režie se složitou synchronizací mezi procesy, která je u MIMD
- Rychlejší komunikace mezi procesory než u MIMD
  - Menší latence
  - Menší režie na struktury komunikačních paketů, směrování atd.

Nevýhody SIMD

- Ne všechny problémy jsou datově paralelizovatelné
- Pokles výkonnosti u programů s mnoha podmíněnými skoky
- Nejsou vhodné při malém počtu procesorů (neexistuje něco jako „starter kit“)
- Vyžadují ne úplně běžné procesory

Granularita paralelismu



- uvnitř instrukcí - INTRA INSTRUCTION
  - nejjemnější paralelismus
- mezi instrukcemi INTER INSTRUCTION
  - některé instrukce se provádějíí paralelně, ale není to vidět na úrovni programovacího jazyka
  - zřetězení u RISC
  - více jednotek (T10000)
- mezi příkazy
  - vektorové počítače
  - specializované koprocesory (FPU)
- mezi bloky procesu (vlákna, thready)
  - parbegin
  - thread1;
  - thread2;
  - parend
- mezi procesy
  - zpravidla nemají sdílenou paměť
  - mohou být odděleně kompilovány