Cray 1

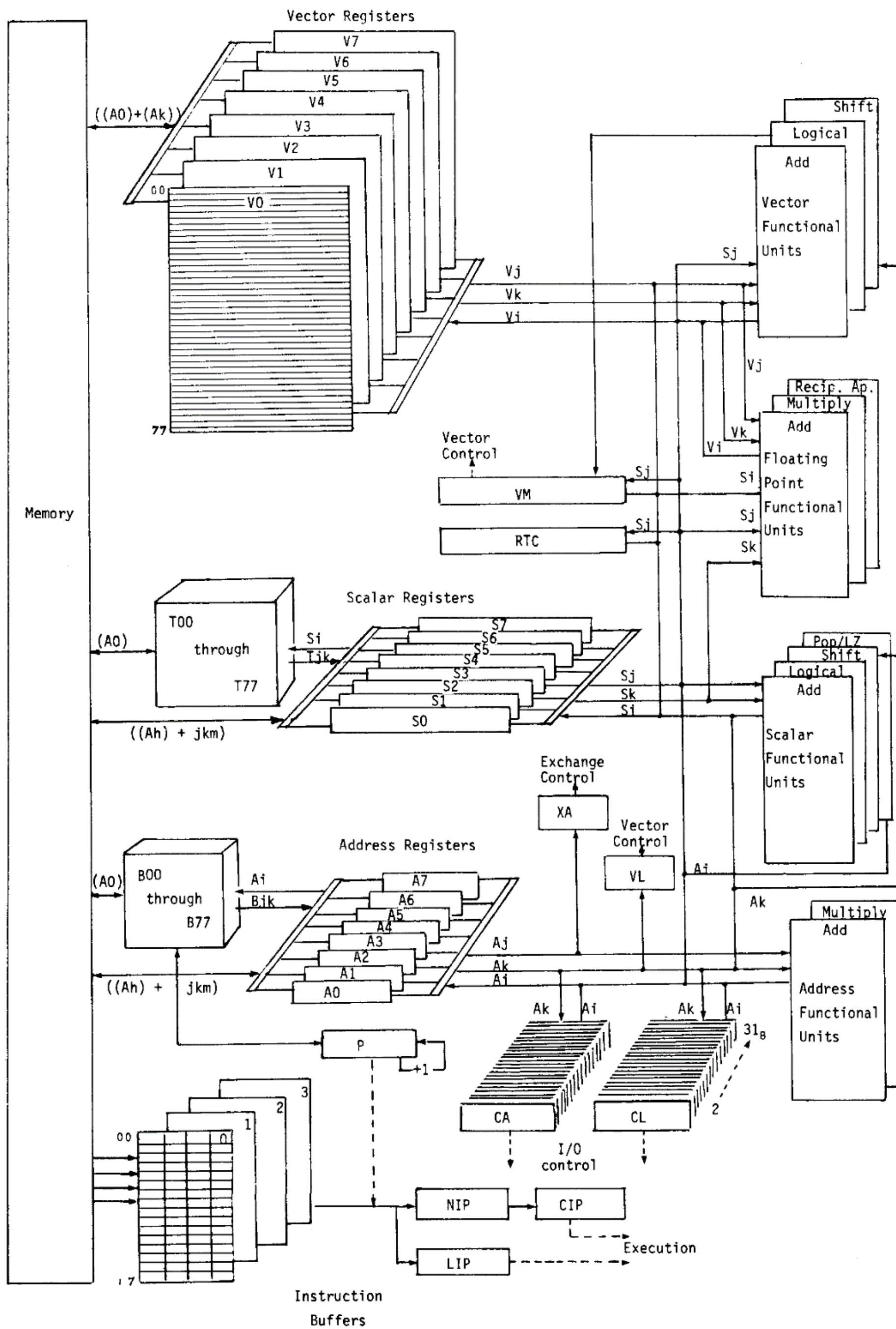
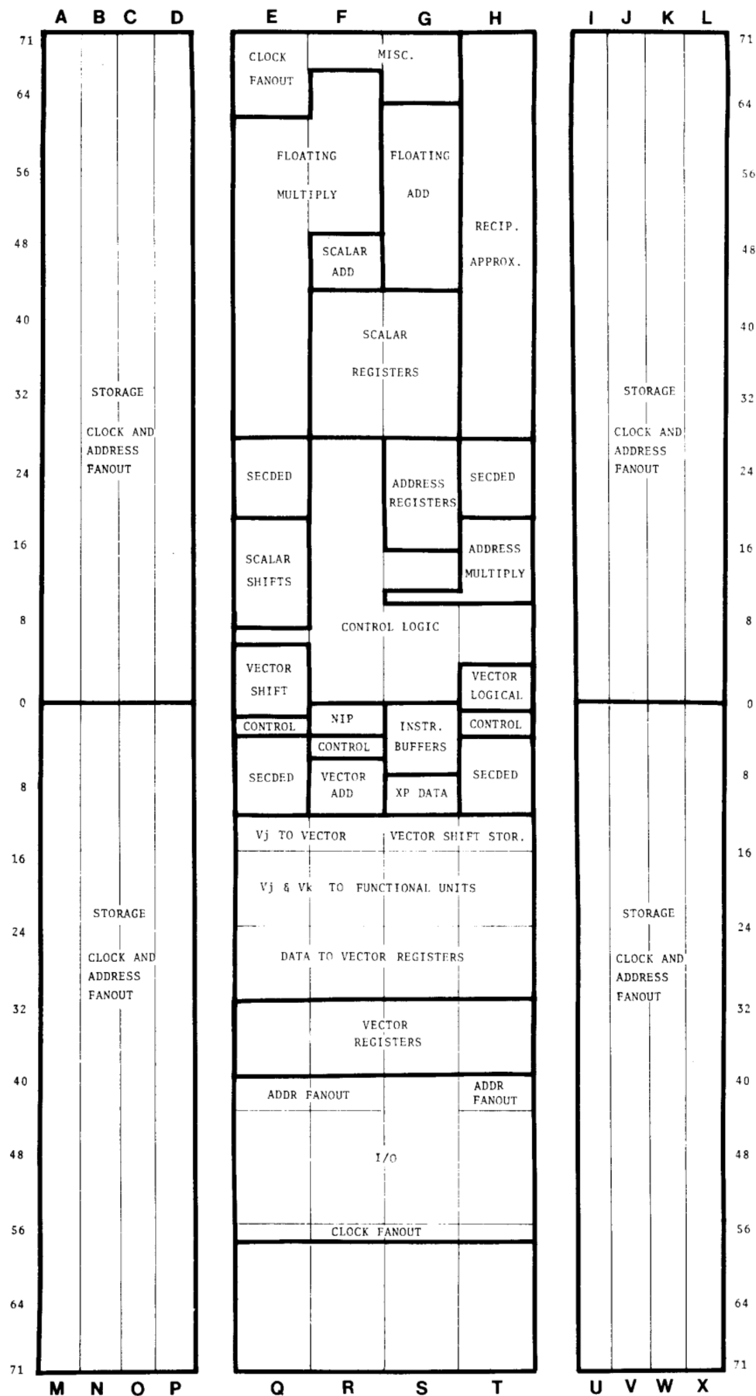
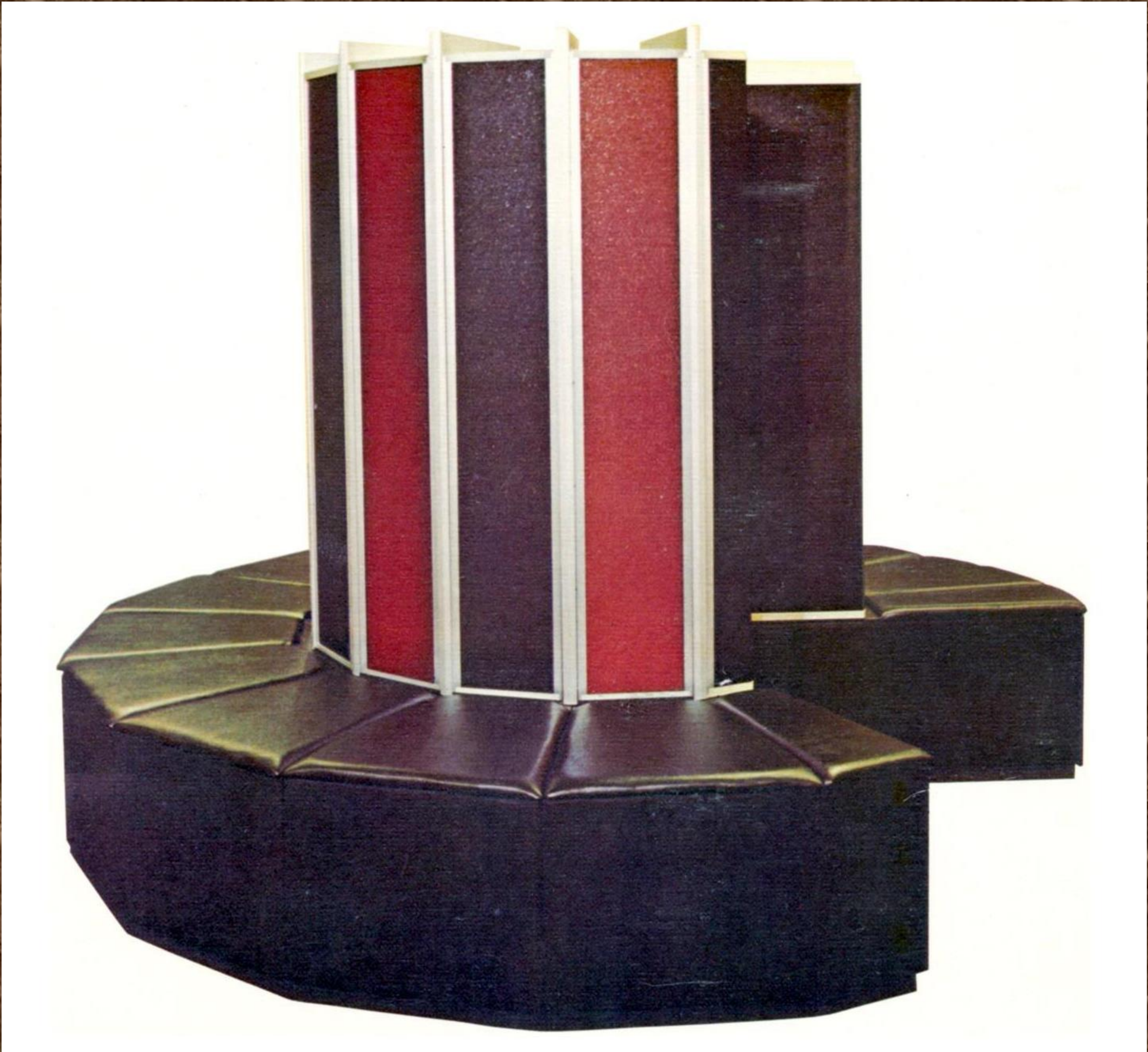


Figure 3-1. Computation section

Cray 1



Cray 1



- Dimensions

  Base - 103½ inches diameter by 19 inches high

  Columns - 56½ inches diameter by 77 inches high including height of base
- 24 chassis
- 1662 modules (16 banks); 113 module types
- Each module contains up to 288 IC packages per module
- Power consumption approximately 115 kw input for maximum memory size
- Freon cooled with Freon/water heat exchange
- Three memory options
- Weight 10,500 lbs (maximum memory size)
- Three basic chip types

  5/4 NAND gates

  Memory chips

  Register chips

Paralelizace

Hardware

- Proč paralelizace?
  - Zrychlení stojí příliš mnoho energie
    - $P = k f^2$
  - Místa je dost (10^8 tranzistorů/chip)
- Jak paralelizovat?
  - ve světě Intel (IA-32)
    - Pipelining (1989: i486) – bez duplikace
    - Superskalarita (1993: Pentium) – duplikace ALU, původní instrukce
    - SIMD (1996: Pentium MMX) – duplikace ALU, nové instrukce
    - Hyperthreading (1998: Xeon) – duplikace registrů
    - Multi-core (2005: Core 2 Duo) – duplikace CPU
  - jinde
    - Vektorové instrukce (1974: Cray) – částečná duplikace

Paralelizace

- **Z pohledu software**
  - Jemnozrnný paralelismus
    - ILP
      - Pipelining
      - Superskalarita
    - SIMD
      - Mírně vektorové instrukce (2-8)
      - Vektorové instrukce (64)
  - Hrubozrnný paralelismus
    - SMP
      - Hyperthreading
      - Multi-core
      - Multi-socket
    - NUMA
      - Multi-core a multi-socket (cc-NUMA)
      - NUMA architektury
    - Cluster

Paralelizace

- **Z pohledu překladače**
  - Jemnozrnný paralelismus
    - ILP
      - Scheduling, software pipelining – změna pořadí instrukcí
    - Vektorizace - automatické použití SIMD instrukcí
      - Překladač vybírá speciální instrukce
      - Překladač určuje pořadí provádění
  - Hrubozrnný paralelismus
    - SMP – Multithreading
      - Strip-Mining - Překladač dělí kód na nezávislé bloky
      - Pořadí provádění je náhodné

Vektorizace

- Paralelní provádění sousedních iterací
  - N=2..64
- Podmínky korektnosti
  - Absence závislostí mezi sousedními iteracemi
  - Předvídatelnost počtu iterací
- Aplikovatelná na cykly s afinním chováním
  - Numerické aplikace (FORTRAN 90)
  - Nepoužitelné při nebezpečí aliasingu (objektové programování)
  - Obvykle vyžaduje přenesení odpovědnosti na programátora (restrict)
- Řada afinních transformací cyklů
  - Loop Reversal
  - Loop Skewing

Loop reversal

- **Výměna vzájemného zanoření cyklů**
  - Příklad: Učebnicový zápis násobení matic

```
for I := 1 to M do
  for J := 1 to N do
    for K := 1 to P do
      C[I,J] := C[I,J]+A[I,K]*B[K,J]
```

- Nelze paralelizovat - následující iterace závisí na předchozí

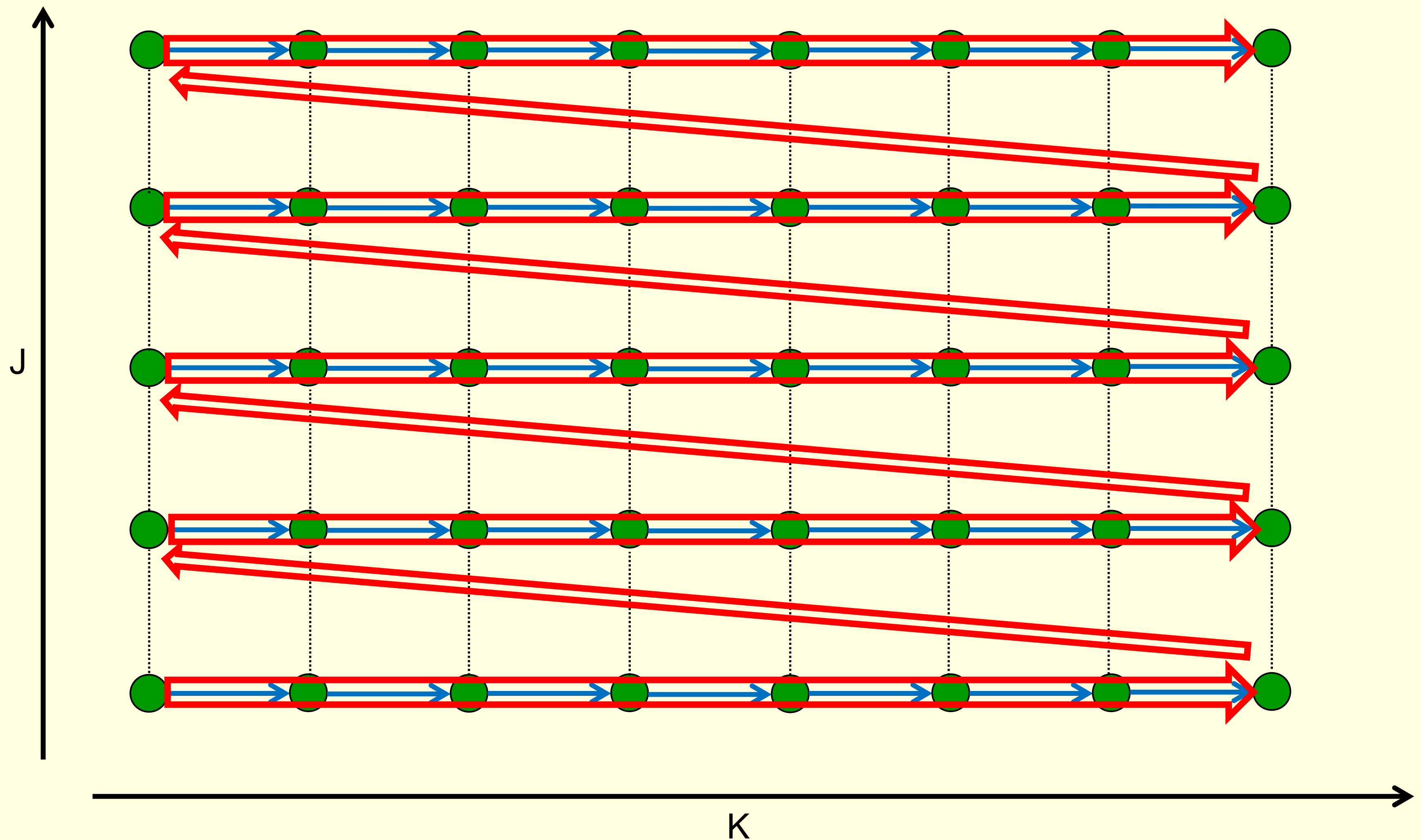
  - Po výměně zanoření cyklů K/J

```
for I := 1 to M do
  for K := 1 to P do
    for J := 1 to N do
      C[I,J] := C[I,J]+A[I,K]*B[K,J]
```

- Překladač si musí být jistý zachováním semantiky
  - Je nutno vyloučit aliasing C=A resp. C=B
- Tato úprava může zhoršit chování vzhledem k cache
  - Závisí na rozměrech P, N
- V nenumerických aplikacích je užitečnost nejistá

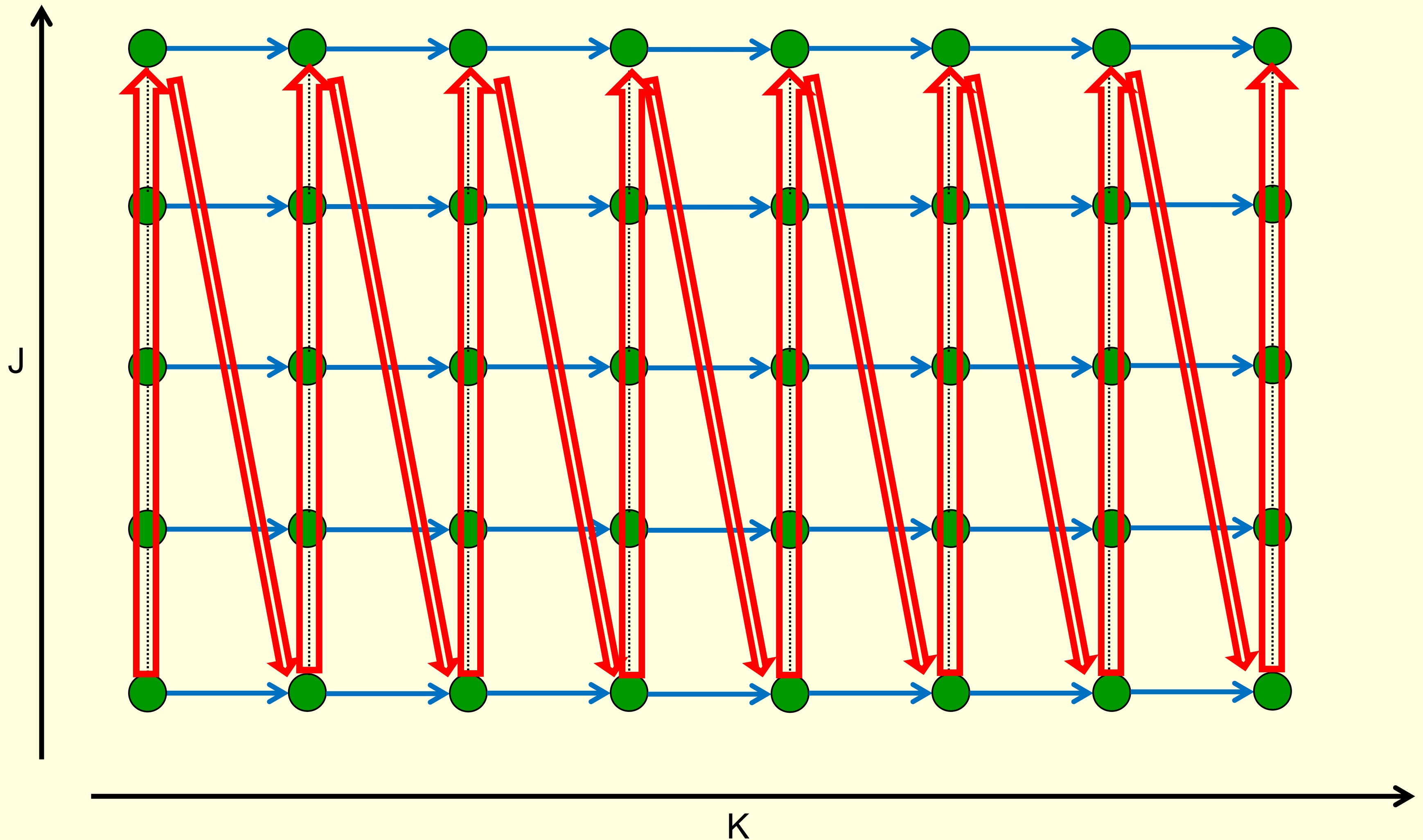
Loop reversal

- Původní průchod
  - Většina sousedů v průchodu je závislá



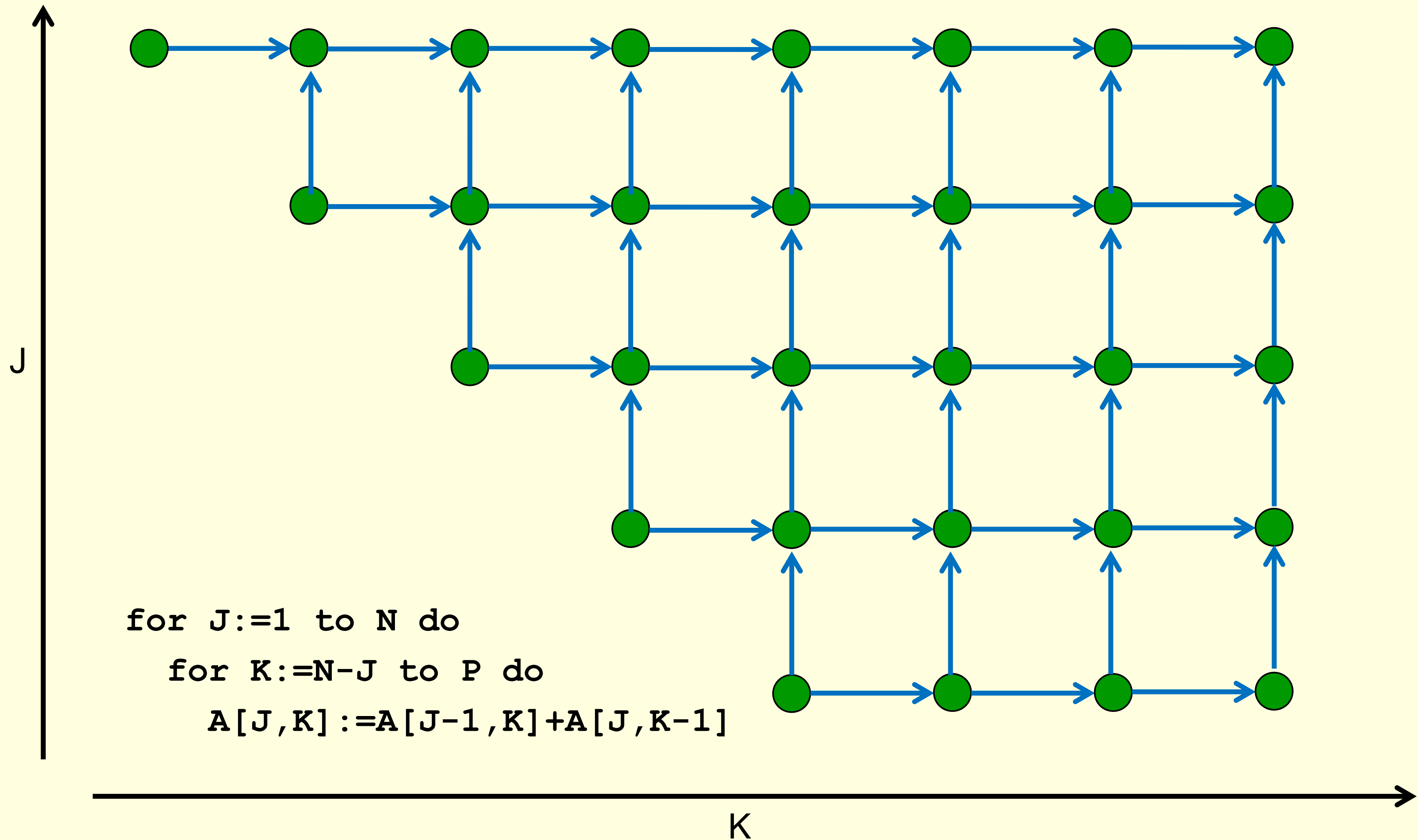
Loop reversal

- Upravený průchod
  - Většina sousedů v průchodu je nezávislá



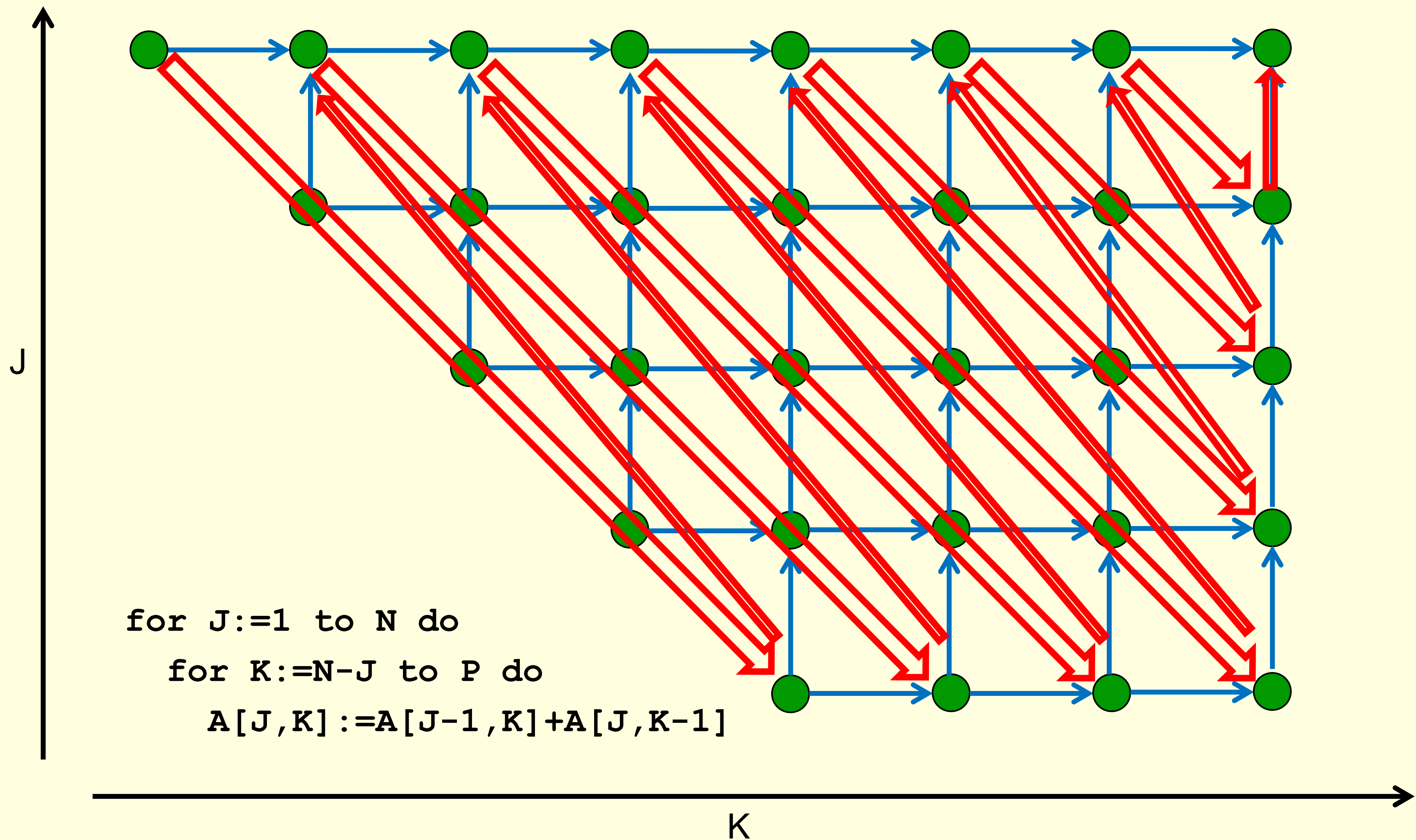
Loop skewing

- Obecnější příklad



Loop skewing

- Obecnější příklad



Loop skewing

Loop skewing

```
for J:=1 to N do
  for K:=N-J to P do
    A[J,K]:=A[J-1,K]+A[J,K-1]
```

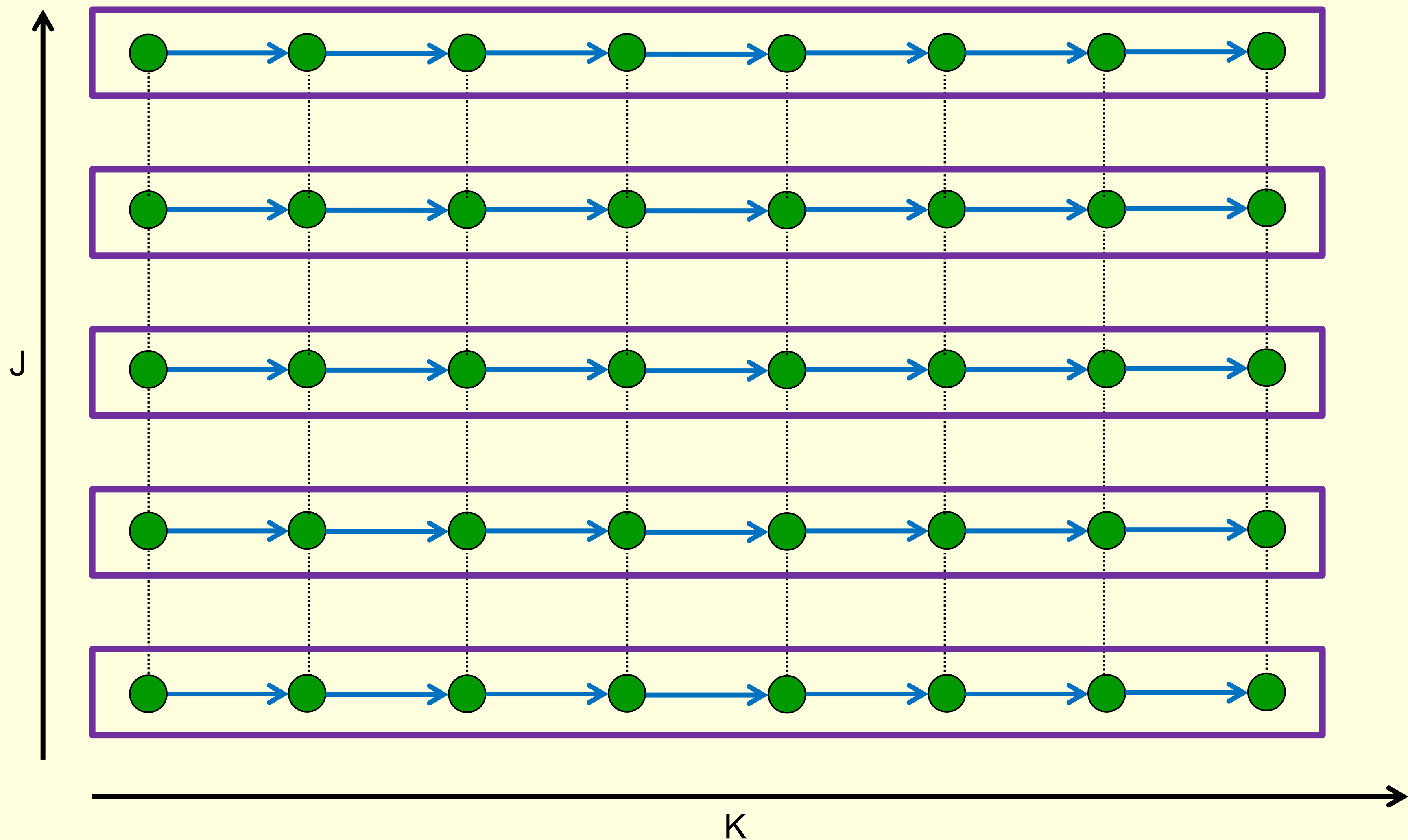
- Prostor iterací = prostor k-tic řídících proměnných
  - Hranice dány lineárními nerovnicemi (s konstantními koeficienty)
- Detekce závislostí mezi iteracemi
  - Přístupy do paměti (indexy) určeny lineárními výrazy (s k. k.)
- Směry (vektory) závislostí popisují zakázané směry iterací
  - Hledá se vektor ležící mimo konvexní obal vektorů závislostí
- Výsledná smyčka pro jemnozrnnou paralelizaci
  - Vnější iterace: Vektor (vektory) v konvexním obalu závislostí
  - Vnitřní iterace: Vektor (vektory) mimo konvexní obal závislostí
- Složitější případy: Prostor iterací se dělí na části řešené zvlášť
  - Zjednodušení mezí cyklů: dělení na části jednoduchých tvarů

Hrubozrnná paralelizace

- Velká běhová režie
  - Vytváření vláken/úloh, plánování
  - Vyžaduje relativně velké na sobě nezávislé bloky
  - Předvídatelně velké bloky: Počet bloků odpovídá počtu CPU
  - Nepředvídatelně velké bloky: Počet bloků větší než počet CPU
    - Bloky se přidělují na CPU dynamicky
- Různé schopnosti běhového prostředí
  - Vlákna bez vzájemné synchronizace
    - Bloky na sobě nesmějí vůbec záviset
  - Pipeline parallelism, task parallelism
    - Plánovač respektuje vzájemné závislosti bloků
    - Závislosti nesmějí být cyklické

Strip mining

- Hledání zcela nezávislých bloků



Hrubozrnná paralelizace

- Bloky k paralelizaci je třeba vybírat v horních patrech iterací
  - Zajištění dostatečné velikosti bloků
- Smyčku je vhodné upravit, aby vnější iterace nebyly závislé
  - Stejné metody jako u jemnozrnné paralelizace, ale opačný cíl
- Loop skewing pro hrubo- i jemnozrnnou paralelizaci
  - Vnější iterace nezávislé (pro hrubozrnnou paralelizaci)
  - Střední iterace závislé
  - Vnitřní iterace nezávislé (pro jemnozrnnou paralelizaci)
  - Problém: Vrstev iterací je obvykle málo
    - Středních iterací musí být dost na pokrytí konvexního obalu závislostí

Hrubozrnná paralelizace

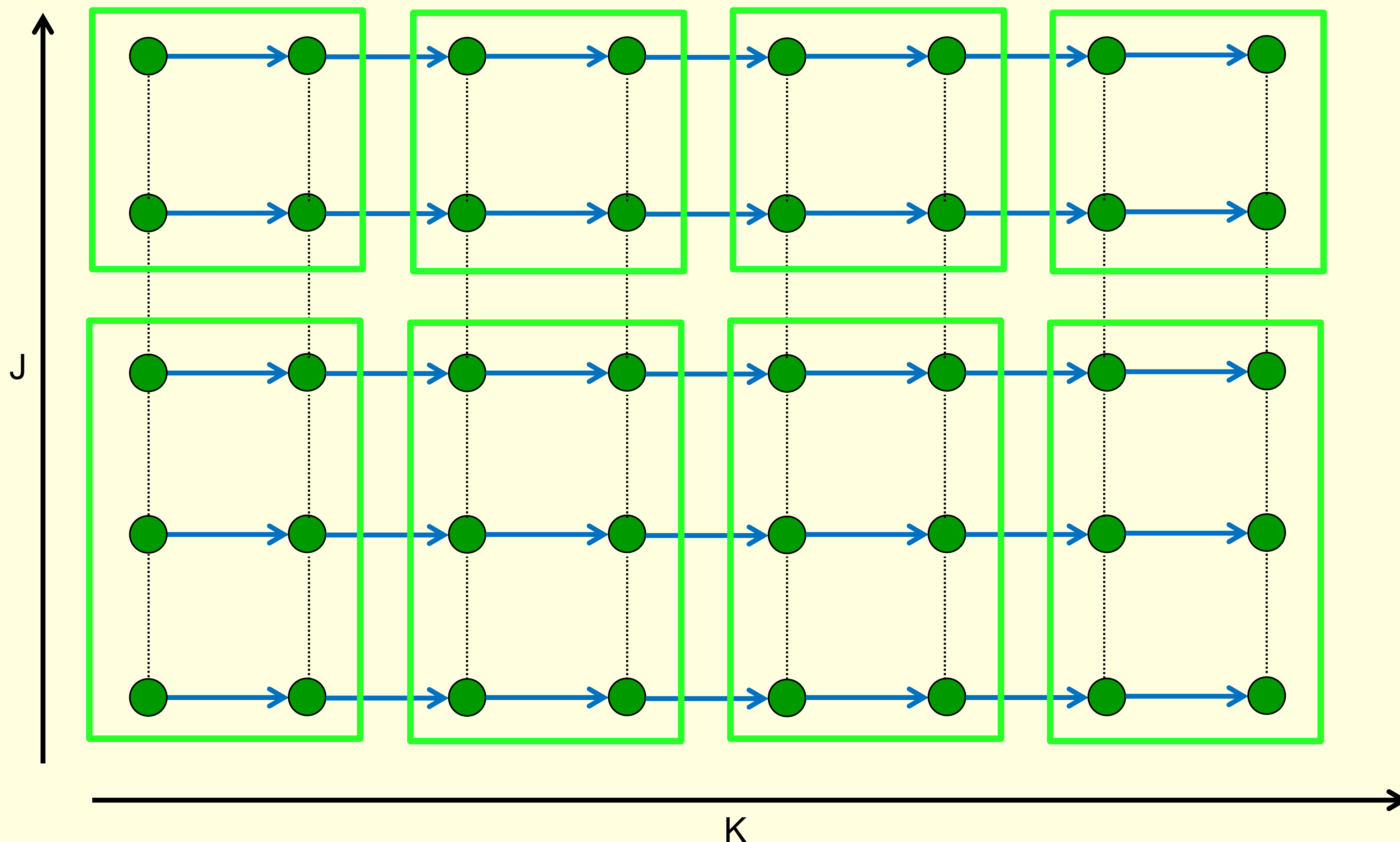
  - Problém: Vrstev iterací je obvykle málo

Řešení: Blocking

- Prostor iterací se rozdělí na menší díly (bloky)
  - Tím vznikají další úrovně iterace
- Iterace nad bloky se uspořádají pro hrubozrnný paralelismus
  - Vnější iterace řeší nezávislé skupiny bloků
    - Strip mining: Vlákna
  - Vnitřní iterace procházejí závislé bloky uvnitř skupin
    - Task parallelism: Tasky
- Iterace uvnitř bloky se upraví pro jemnozrnný paralelismus
  - Vnější iterace jsou závislé
  - Vnitřní iterace jsou bez závislostí
    - SIMD, ILP

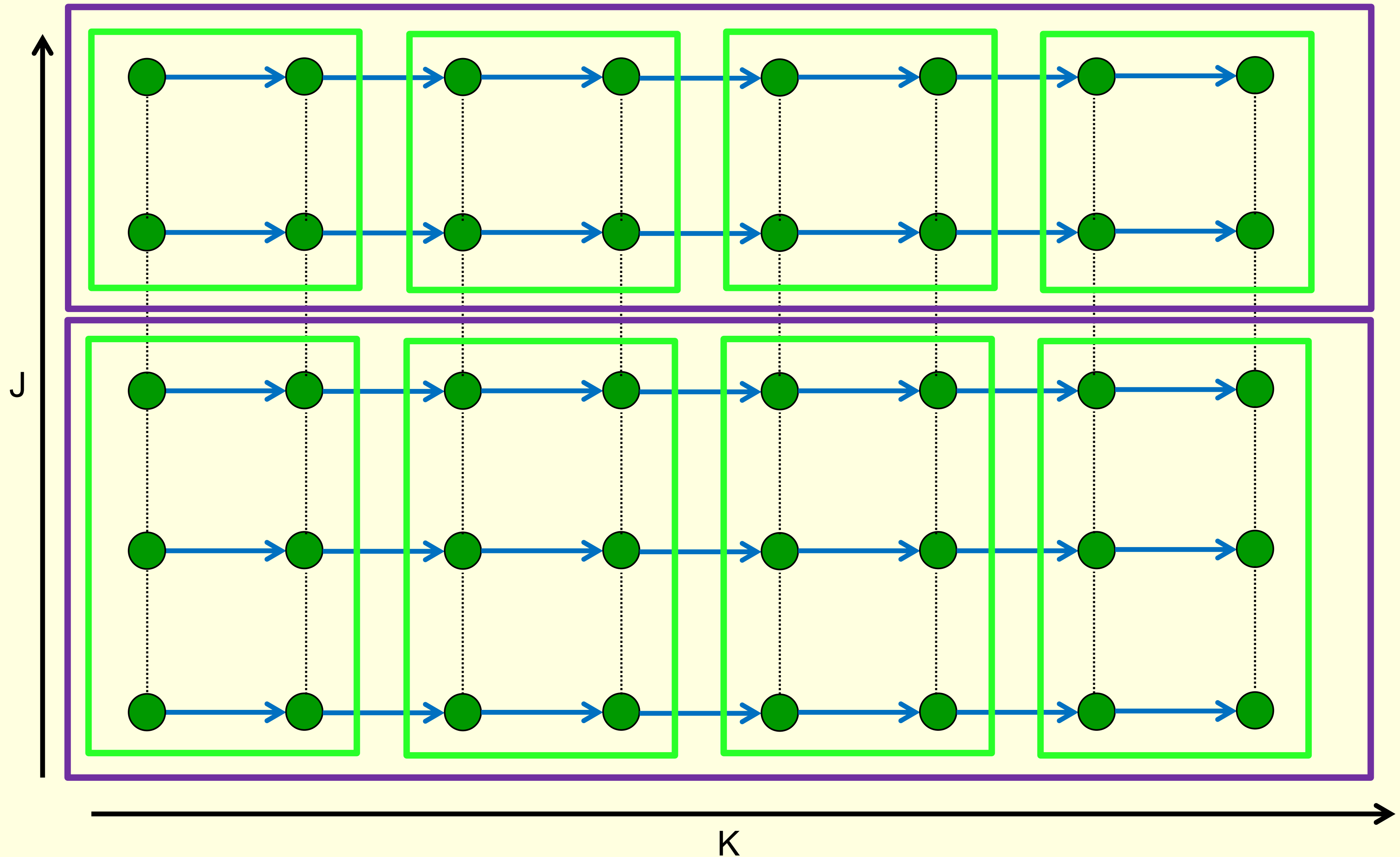
Blocking

- Vytvoření bloků



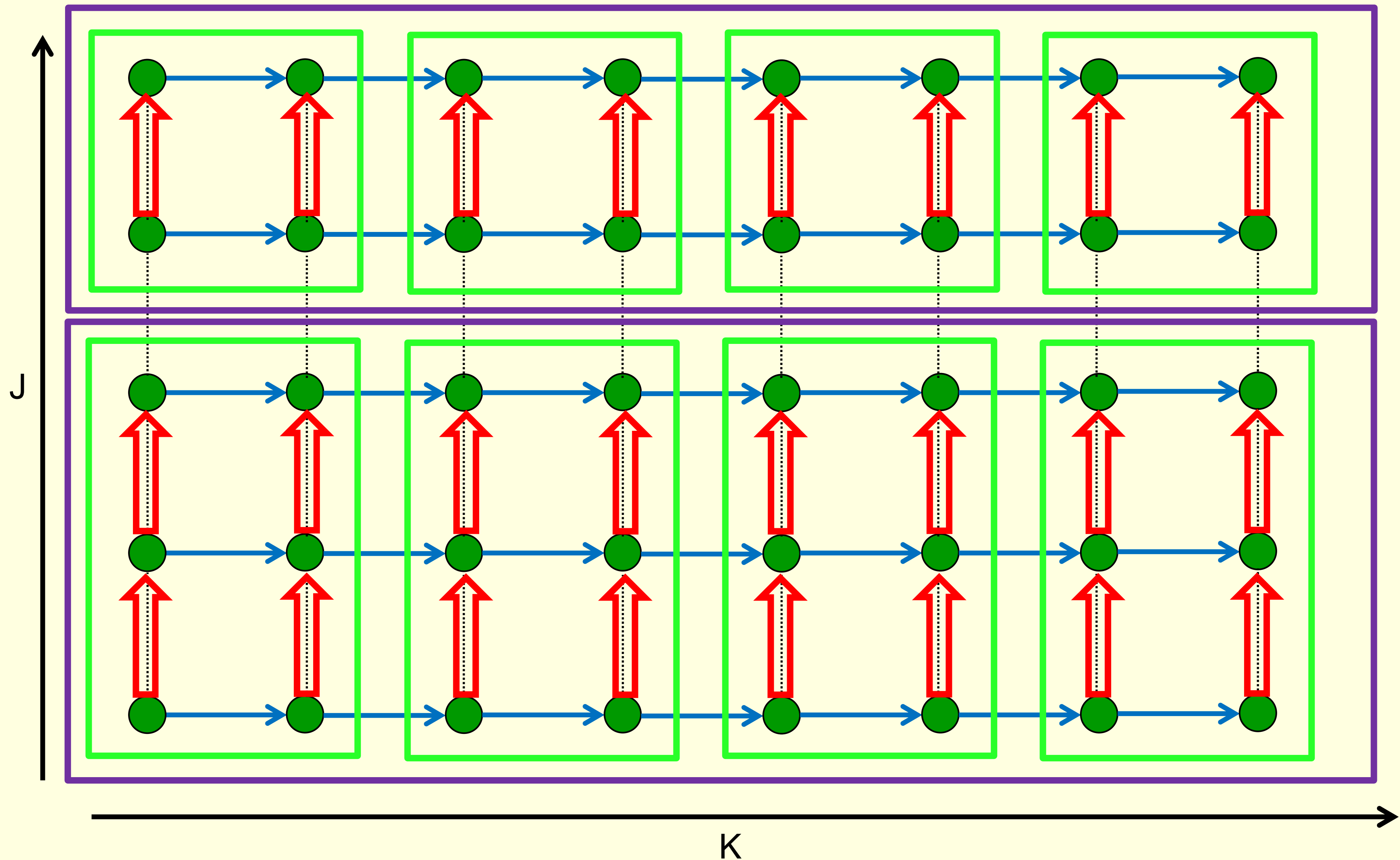
Blocking

- Strip mining nad bloky



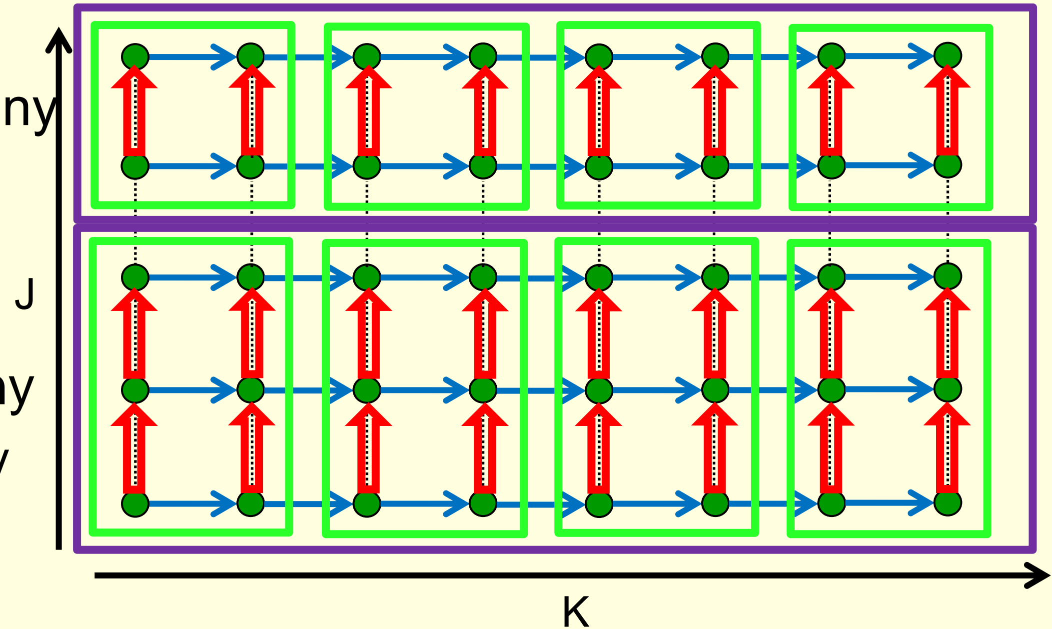
Blocking

- Loop skewing uvnitř bloků



Blocking

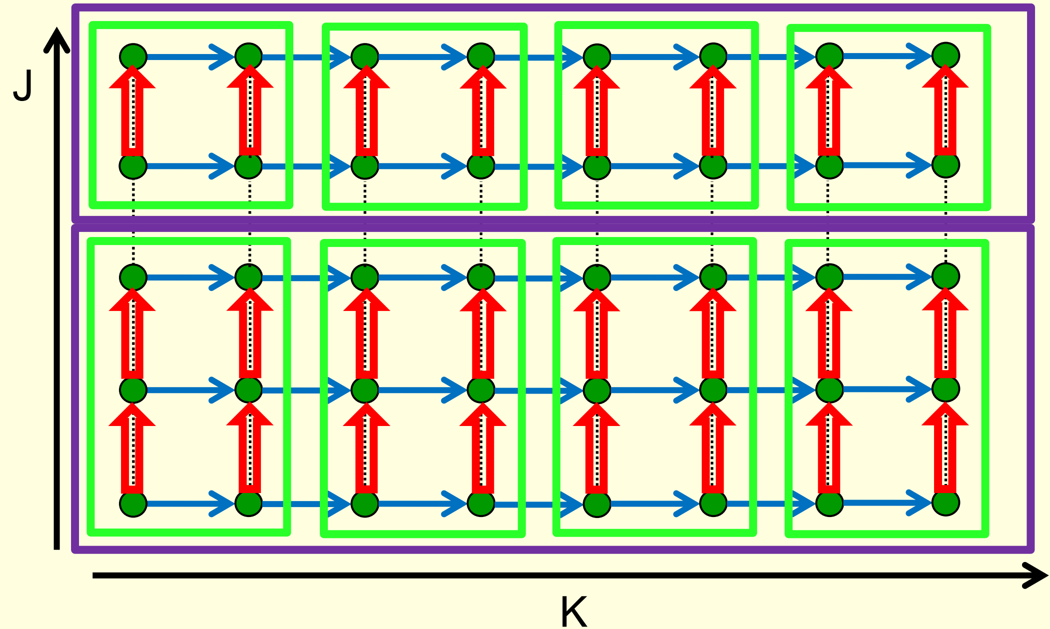
- Blocking
  - Původní iterace rozděleny na 2 patra
    - Vždy ekvivalentní
    - Problémy se zbytky
  - Nové iterace promíchány
    - Pouze některé výměny jsou možné
    - Cílem je nezávislost v okrajových patrech




```
parallel for J1 := 0 to N-1 step SJ do
  for K1 := 0 to P-1 step SK do
    for K2 := 1 to SK do
      parallel for J2 := 1 to SJ do
      begin
        J := J1 * SJ + J2;
        K := K1 * SK + K2;
        C[I,J] := C[I,J]+A[I,K]*B[K,J]
      end
```

Blocking

- Blocking
  - Implementace vlákny



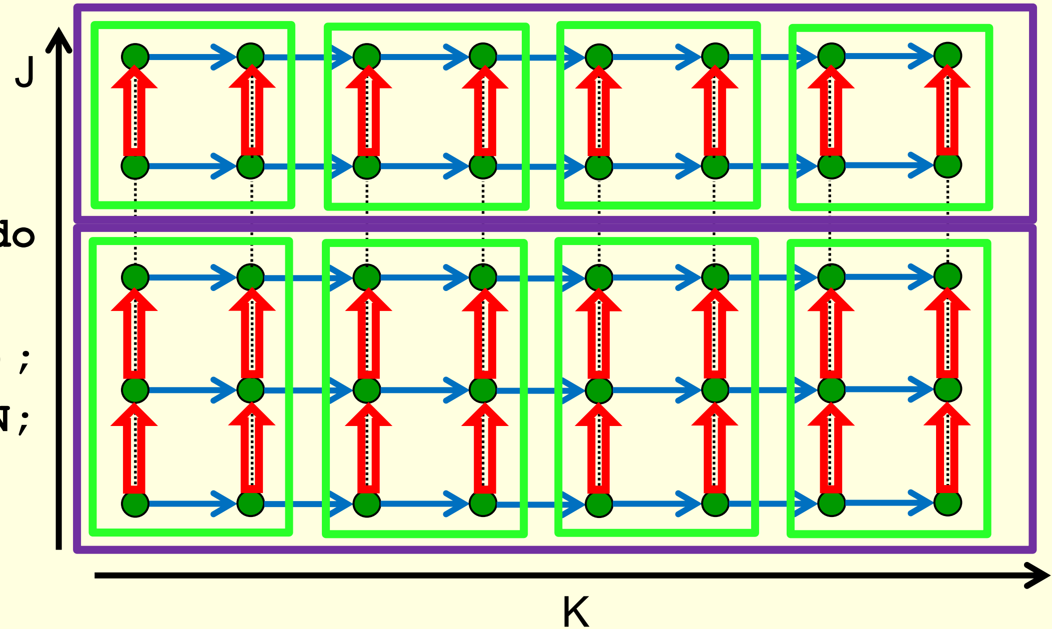

```
for J1 := 0 to N-1 step SJ do
  create_thread( f, J1);
wait();

function f( J1)
begin
  for K1 := 0 to P-1 step SK do
    for K2 := 1 to SK do
      parallel for J2 := 1 to SJ do
      begin
        J := J1 * SJ + J2;
        K := K1 * SK + K2;
        C[I,J] := C[I,J]+A[I,K]*B[K,J]
      end
end
```

Blocking

- Blocking
  - Task parallelism

```
for J1 := 0 to N-1 step SJ do
  for K1 := 0 to P-1 step SK do
  begin
    TN := new_task( f, J1, K1);
    add_dep( TP, TN); TP := TN;
  end;
run();
```




```
function f( J1, K1)
begin
    for K2 := 1 to SK do
      parallel for J2 := 1 to SJ do
      begin
        J := J1 * SJ + J2;
        K := K1 * SK + K2;
        C[I,J] := C[I,J]+A[I,K]*B[K,J]
      end
end
```

Optimalizace pro cache

- Podmínky dobrého využití cache
  - Prostorová lokalita
    - Jednotkou přístupu je cache line (typ. 64 B)
    - Přístupy do sousedních míst mají být blízko, nejlépe najednou
    - Zápisy také způsobují výpadky - antidependence se řeší také
  - Časová lokalita
    - Čím delší je doba mezi přístupy, tím větší je pravděpodobnost výpadku
    - Přístupy k témuž paměťovému místu je výhodné v čase rozmístit nepravidelně
- Blocking většinou vyhovuje i z pohledu cache
  - Obvykle řeší časovou lokalitu
  - Prostorová lokalita závisí na datových strukturách
    - Překladače procedurálních jazyků datové struktury neupravují
    - Mohou však upravit časový průběh přístupů