

➤ Vstup

- ❖ Částečně sekvenční kód
 - Možnost přesně definovat rozsahy platnosti
- ❖ Ekvivalenty strojových instrukcí
 - Speciální zpracování přesunových instrukcí
- ❖ Instrukce pracují s virtuálními registry
- ❖ Optimistický kód
 - Všechny jednoduché proměnné bez aliasu v registrech
- ❖ Spočtené rozsahy platnosti proměnných
 - Množina bodů v mezikódu, kde je proměnná živá
 - Hranice základních bloků
 - Hranice mezi instrukcemi
 - Přesnější informace v místech, kde je proměnná čtena/zapisována

➤ Výstup

- ❖ Přiřazení virtuálních registrů fyzickým

➤ **Optimalizace: Minimalizace přesunových instrukcí**

➤ **Vytvoření matice kolizí**

❖ $C : VAR \times VAR \rightarrow Bool$

- Matice je velká, ale potřebujeme rychlý přístup

❖ Kolize = neprázdný průnik oblastí platnosti

- Zvláštní posouzení okrajových doteků oblastí

➤ **Odhad ceny za spill-kód**

❖ $P : VAR \rightarrow Integer$

- Odhad počtu použití proměnné v průměrném provedení procedury

➤ **Vlastní algoritmus alokace**

❖ Barvení grafu (VAR,C) n barvami

- n je počet fyzických registrů
- NP-úplná úloha, dobré heuristické řešení

➤ **Doplnění spill-kódu**

➤ Barvení grafu (VAR,C) n barvami

- ❖ n je počet fyzických registrů
- ❖ NP-úplná úloha, dobré heuristické řešení

➤ Princip

- ❖ Má-li vrchol méně než n sousedů, lze jej vždy obarvit

➤ Dvě fáze

- ❖ Postupné odebírání vrcholů
 - Ve vhodném pořadí tak, aby později měly šanci na obarvení
 - Odebrané vrcholy se ukládají na zásobník
- ❖ Zpětná rekonstrukce grafu
 - Vrcholy se odebírají ze zásobníku
 - Pokud je lze obarvit, obarví se a přidají do grafu
 - Neobarvitelné nebudou přiděleny do registru

- Vstup: graf (VAR, C) , ceny P , počet n

```
(VAR2, C2) := (VAR, C)
```

```
while not empty(VAR2) do
```

```
  if not exists v:VAR2 st deg(v, C2) < n then
```

```
    select v st P(v) is minimal; // or deg(v, C2) is maximal
```

```
  end if;
```

```
  (VAR2, C2) := (VAR2 - {v}, C2 - edges(v))
```

```
  stack.push(v);
```

```
end while;
```

```
while not empty(stack) do
```

```
  stack.pop(v)
```

```
  M = {u ∈ VAR2; <v, u> ∈ C}
```

```
  if exists i ∈ {1..n} st not i ∈ color(M) then
```

```
    color(v) := i
```

```
    (VAR2, C2) := (VAR2 + {v}, C2 +)
```

```
  else
```

```
    spill(v) := true
```

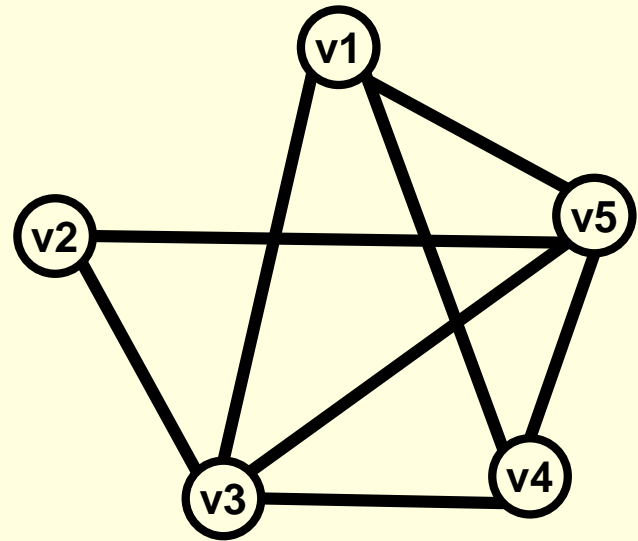
```
  end if
```

```
end while
```

- Výstup: přidělení registrů *color*, nepřidělené registry *spill*

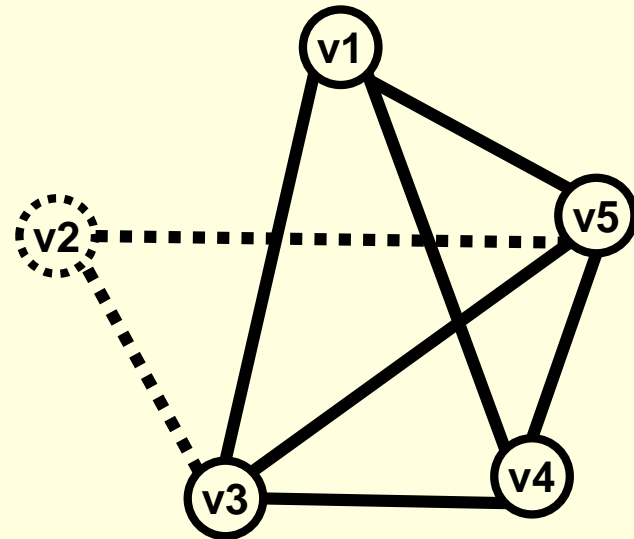
❖ *Example (N=3)*

- $\text{deg } v_2 < N \Rightarrow \text{push } v_2$



❖ Example ($N=3$)

- $\text{deg } v_2 < N \Rightarrow \text{push } v_2$
- All degrees $\geq N \Rightarrow$ select lowest priority node, e.g. v_1

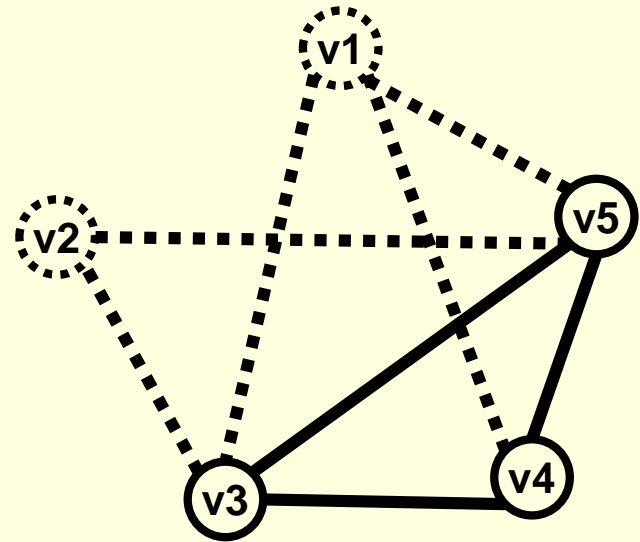


❖ Stack:

- v_2

❖ Example ($N=3$)

- $\text{deg } v_2 < N \Rightarrow \text{push } v_2$
- All degrees $\geq N \Rightarrow$ select lowest priority node, e.g. v_1
- All degrees $< N \Rightarrow$ select any, e.g. v_3

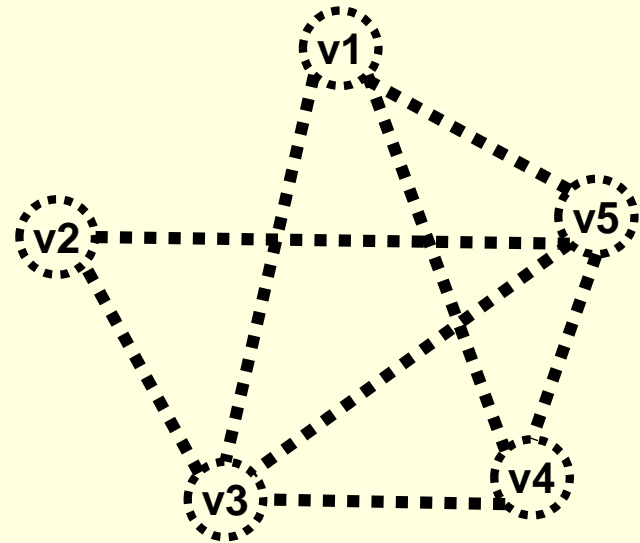


❖ Stack:

- v_3
- v_1
- v_2

❖ Example ($N=3$)

- $\text{deg } v_2 < N \Rightarrow \text{push } v_2$
- All degrees $\geq N \Rightarrow$ select lowest priority node, e.g. v_1
- All degrees $< N \Rightarrow$ select any, e.g. v_3
- Same for the rest, v_4, v_5



❖ Stack:

- v_5
- v_4
- v_3
- v_1
- v_2

❖ Example ($N=3$)

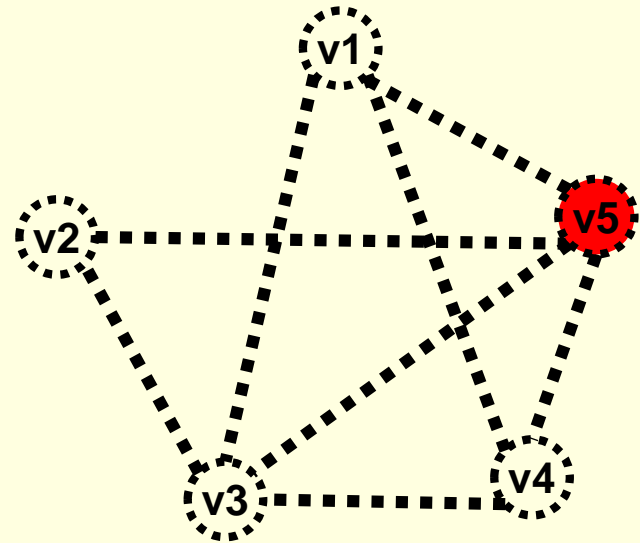
- $\text{deg } v_2 < N \Rightarrow \text{push } v_2$
- All degrees $\geq N \Rightarrow$ select lowest priority node, e.g. v_1
- All degrees $< N \Rightarrow$ select any, e.g. v_3
- Same for the rest, v_4, v_5

❖ Second phase

- Select a color for v_5

❖ Stack:

- v_5
- v_4
- v_3
- v_1
- v_2



❖ Example ($N=3$)

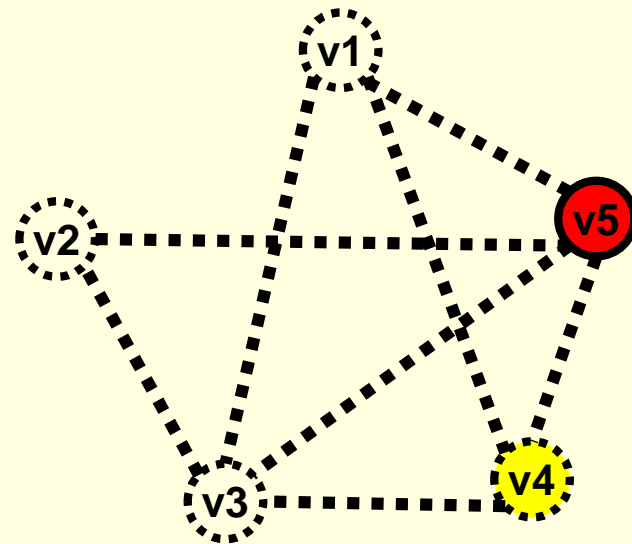
- $\text{deg } v_2 < N \Rightarrow \text{push } v_2$
- All degrees $\geq N \Rightarrow$ select lowest priority node, e.g. v_1
- All degrees $< N \Rightarrow$ select any, e.g. v_3
- Same for the rest, v_4, v_5

❖ Second phase

- Select a color for v_5
- Select a color for v_4

❖ Stack:

- v_4
- v_3
- v_1
- v_2



❖ Example ($N=3$)

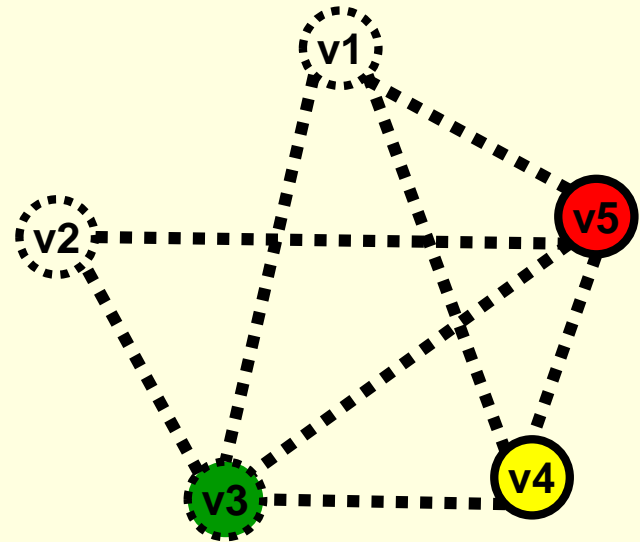
- $\text{deg } v_2 < N \Rightarrow \text{push } v_2$
- All degrees $\geq N \Rightarrow$ select lowest priority node, e.g. v_1
- All degrees $< N \Rightarrow$ select any, e.g. v_3
- Same for the rest, v_4, v_5

❖ Second phase

- Select a color for v_5
- Select a color for v_4
- Select a color for v_3

❖ Stack:

- v_3
- v_1
- v_2



❖ Example ($N=3$)

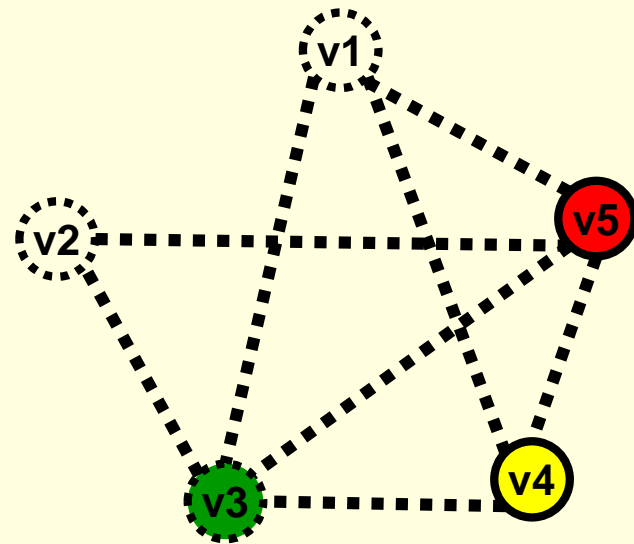
- $\text{deg } v_2 < N \Rightarrow \text{push } v_2$
- All degrees $\geq N \Rightarrow$ select lowest priority node, e.g. v_1
- All degrees $< N \Rightarrow$ select any, e.g. v_3
- Same for the rest, v_4, v_5

❖ Second phase

- Select a color for v_5
- Select a color for v_4
- Select a color for v_3
- No color for $v_1 \rightarrow$ spill

❖ Stack:

- v_1
- v_2

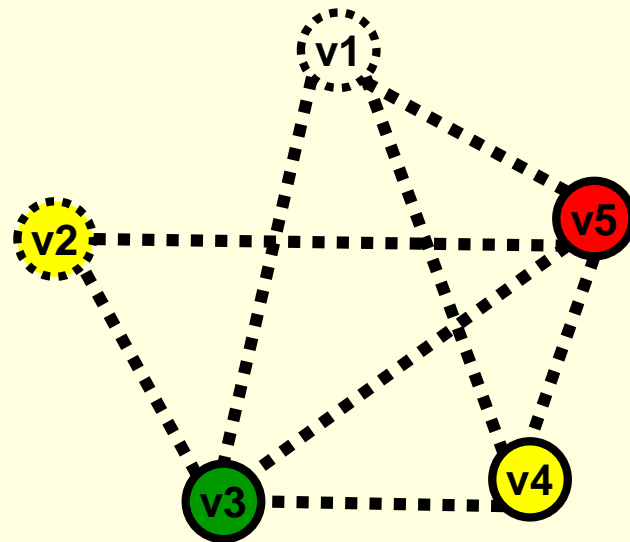


❖ Example ($N=3$)

- $\text{deg } v_2 < N \Rightarrow \text{push } v_2$
- All degrees $\geq N \Rightarrow$ select lowest priority node, e.g. v_1
- All degrees $< N \Rightarrow$ select any, e.g. v_3
- Same for the rest, v_4, v_5

❖ Second phase

- Select a color for v_5
- Select a color for v_4
- Select a color for v_3
- No color for $v_1 \rightarrow$ spill
- Select a color for v_2



❖ Stack:

- v_2

➤ **Optimalizace: Minimalizace přesunových instrukcí**

❖ **Jednoduché řešení: Ztotožnit virtuální registry spojené přesunovou instrukcí**

- Nelze vždy – ztotožňované registry mohou spolu kolidovat
- Duplikace kódu může pomoci

❖ **Složitější řešení: Úprava algoritmu alokace**

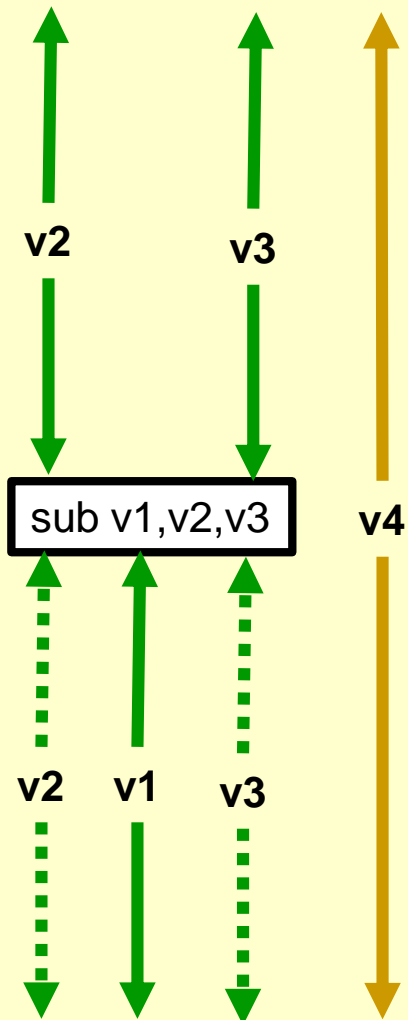
- Určení skupin uzlů svázaných přesunovými instrukcemi
- Při obarvování vybírat zároveň všechny obarvitelné uzly skupiny

➤ Doplnění spill-kódu

- ❖ Alokovat neobarvené uzly na zásobníku
 - Podobný algoritmus barvení grafu s neomezeným n
- ❖ Opatřit každou instrukci spill-kódem
 - Prefix: Přesun neobarvených vstupních operandů do registrů
 - Suffix: Přesun neobarvených výstupních operandů do paměti
- ❖ Problém: kde vzít registry pro tyto operandy
 - Jednoduché řešení: Rezervované nepřidělované registry
 - Lepší řešení: Alokovat pomocné registry v rámci normální alokace
 - Vstupy a výstupy každé instrukce označeny jako uzly grafu
 - Nekolidují s virtuálními registry, na které se vážou
 - Kolidují mezi sebou a s ostatními živými virtuálními registry
 - Opatřeny nejvyšší cenou
- ❖ Problém: spill-kód je často zbytečný
 - Dodatečná optimalizace – eliminace redundantních přesunů

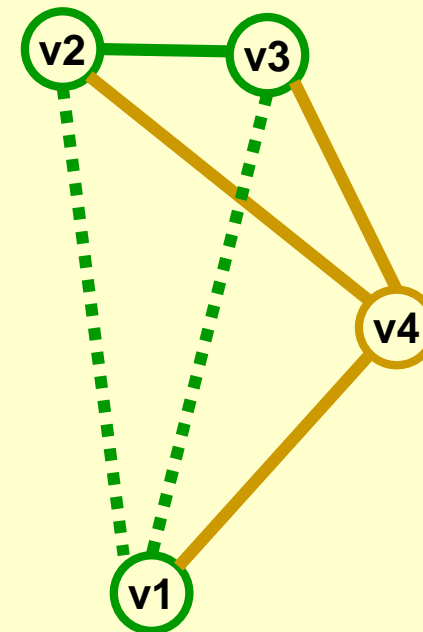
Optimistic approach

Variables v1,v2,v3 in registers



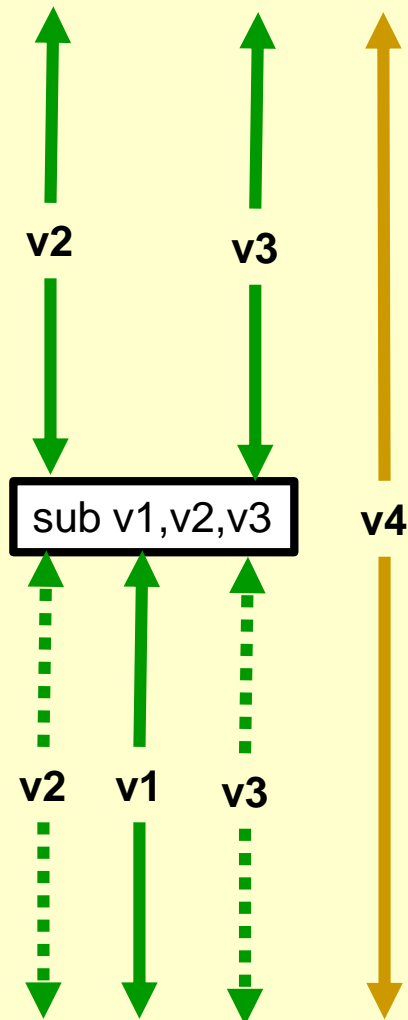
Corresponding collision graph

the collisions v1-v2 and v1-v3 appear if the live range of v2 or v3 extends after the instruction



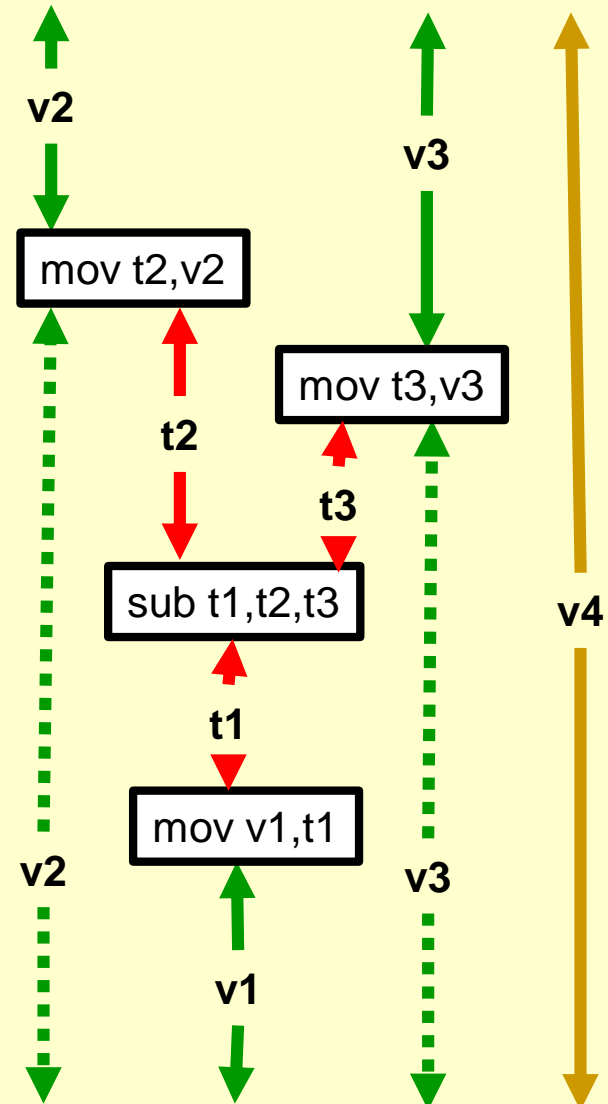
Optimistic approach

Variables v1,v2,v3 in registers



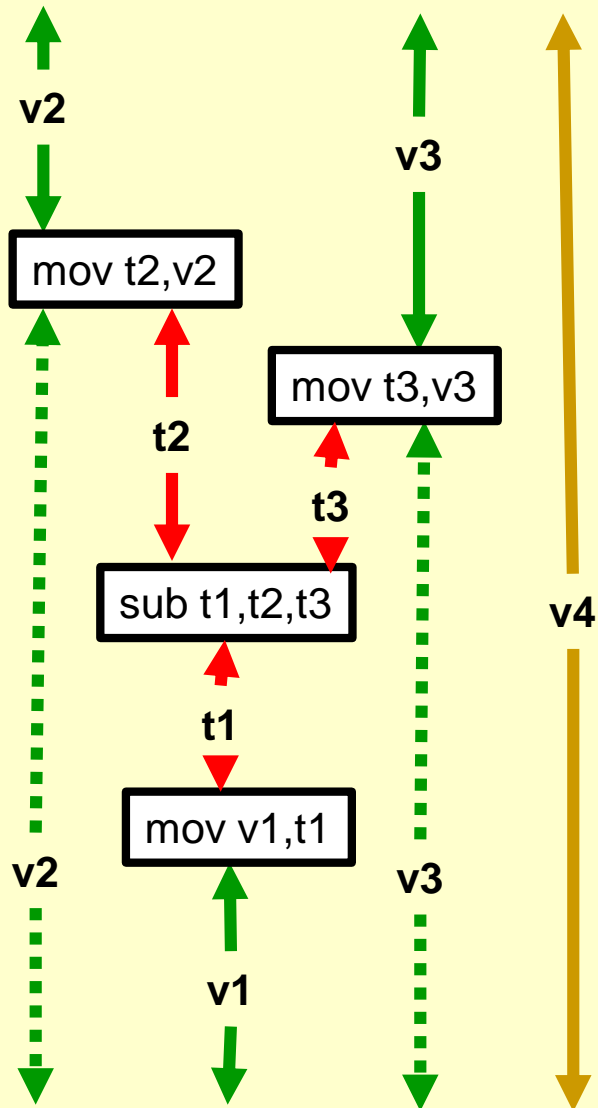
Pessimistic approach

Temporary registers t1,t2,t3 needed



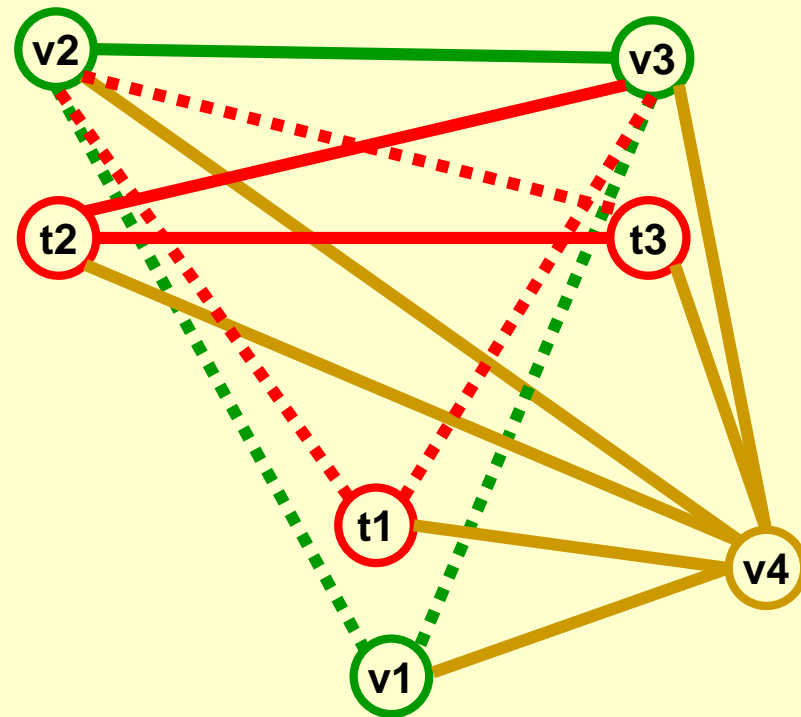
Pessimistic approach

Temporary registers t1,t2,t3 needed



Corresponding collision graph

There is no collision `v2-t2` or `v3-t3` because they two nodes carry the same contents
`t1,t2,t3` have infinite priority in coloring



➤ **Množiny povolených registrů**

- ❖ Pro každý operand každé instrukce je určena množina registrů, v nichž může být tento operand

➤ **Ortogonalní sada instrukcí**

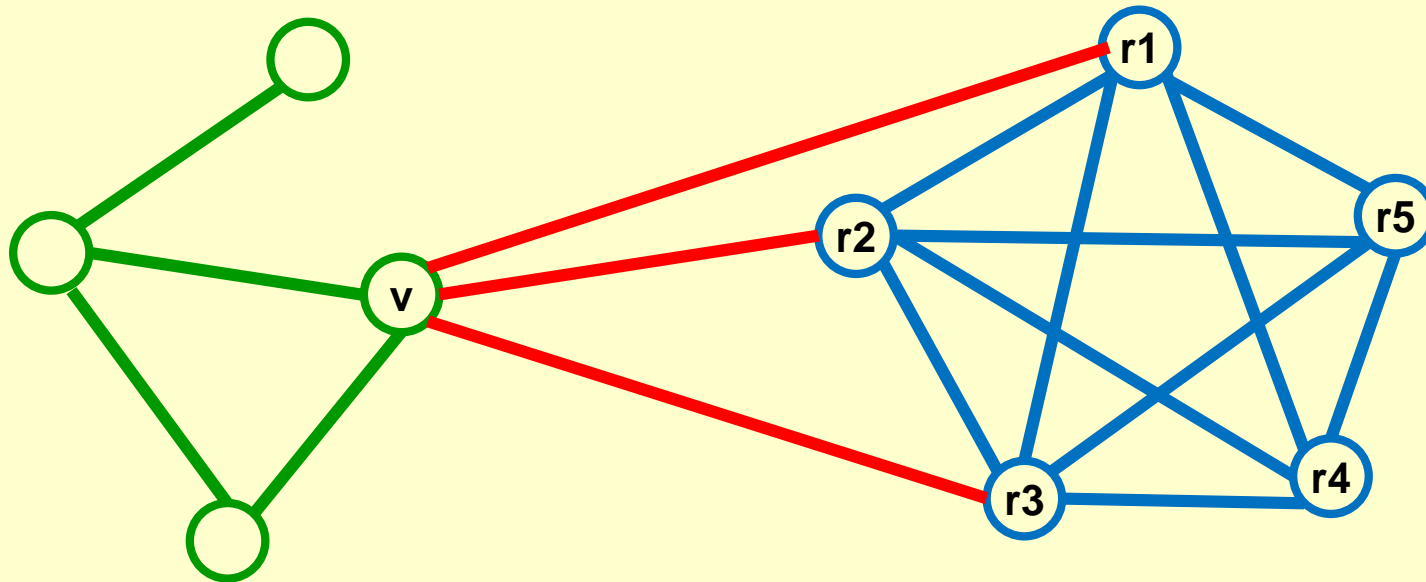
- ❖ Každé dvě množiny povolených registrů jsou buď identické nebo disjunktní
- ❖ Alokace registrů se spouští pro každou množinu zvlášť

➤ **Neortogonalní sada instrukcí**

- ❖ Množiny povolených registrů se simulují v grafu kolizí
 - Přidat úplný podgraf uzlů vysoké priority - reprezentují fyzické registry
 - Spojit hranou všechny virtuální registry s těmi fyzickými, do kterých nesmí být přiděleny
- ❖ Problém: Nemusí existovat obarvení
 - Odložení do paměti, přestože stačí přesun mezi registry

Representing physical register requirements in collision graph

- Add a complete subgraph containing a vertex for each physical register
- For every variable with limited physical register options, add an edge to every node representing a forbidden physical register
- Run the coloring algorithm, with infinite priority of physical register nodes
- Assign the meaning of colors depending on the coloring of the physical register nodes



➤ Další problémy

❖ Subregistry

- Intel IA-32: al-ax-eax-edx:eax
- Řešení:
 - Povýšení na větší registr - neoptimální
 - Různé triky při vytváření grafů kolizí

❖ Registrové volací konvence

- Volání procedury se chová jako instrukce používající několik registrů
- Neortogonální – volací konvence předepisuje, ve kterém registru má být který operand
 - Často vede k neřešitelným kolizím – zbytečný spill-kód
- Řešení: Rozsah platnosti proměnné se rozdělí na oblasti
 - Kolem každé instrukce, používající proměnnou, jedna oblast
 - Oblasti se dotýkají v místě, kde by byla přesunová instrukce, pokud by byla zapotřebí
 - Uzly reprezentující oblasti téže proměnné se přednostně obarvují toutéž barvou