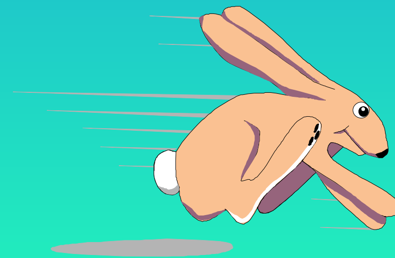
Preference-Directed Graph Coloring

Akira Koseki,
Hideaki Komatsu, Toshio Nakatani

IBM Tokyo Research Laboratory



Coloring-Based Register Allocation

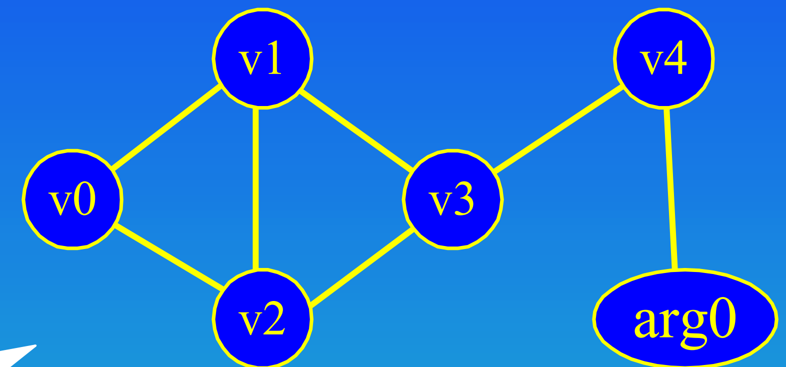
- Map *M variables to N* registers.
- Build an interference graph.
- Find an N-coloring.
  - NP-complete.
  - Chaitin's heuristic: *simplify* by removing *low-degree* nodes.
- Able to pack the variables to N colors with spills suppressed

Coloring-Based Register Allocation Example

v0 v1 v2 v3 v4 arg0

```
i0:      v0 = [arg0]
i1:L1:   v1 = [v0]
i2:      v2 = [v0+4]
i3:      v3 = v0
i4:      v4 = v1 + v2
i5:      arg0 = v3
i6:      call
i7:      v0 = v4+1
i8:      if v0 != 0 goto L1
i9:      ret
```

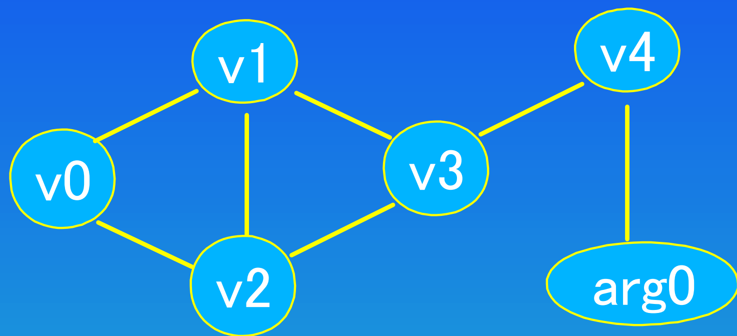
Sample Code



Interference Graph

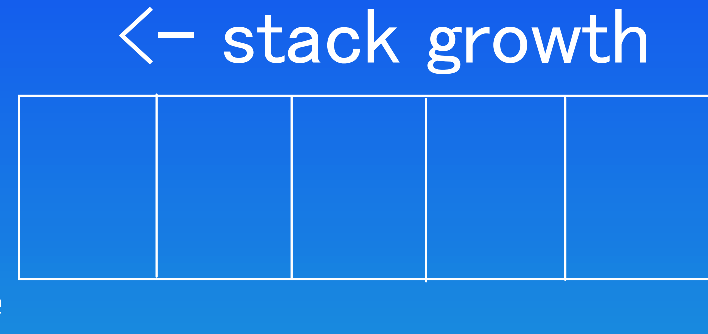
Map to 3 colors (registers)

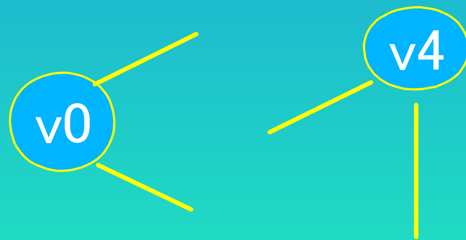
Chaitin-Style Coloring Simplification



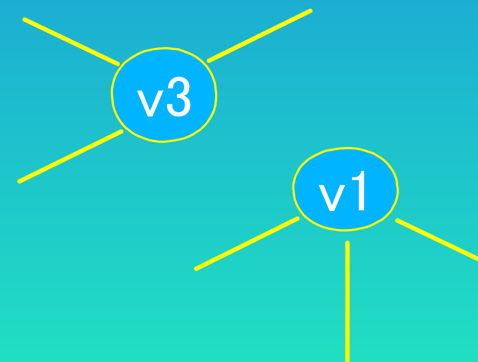
Interference Graph

push low-degree nodes



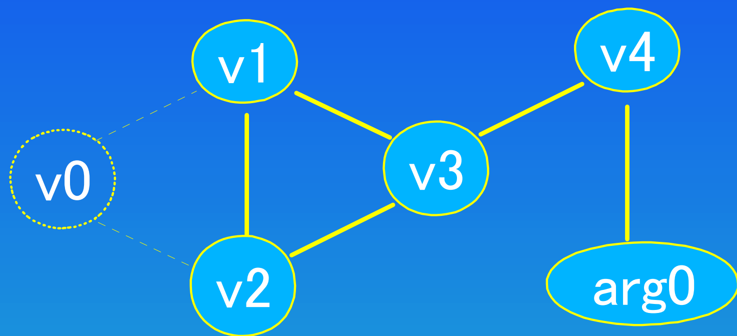


low-degree nodes



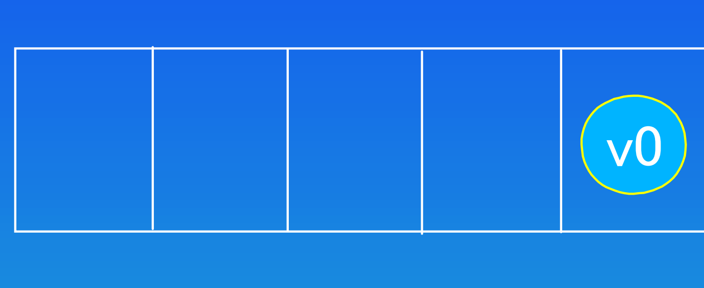
significant-degree nodes

Chaitin-Style Coloring Simplification

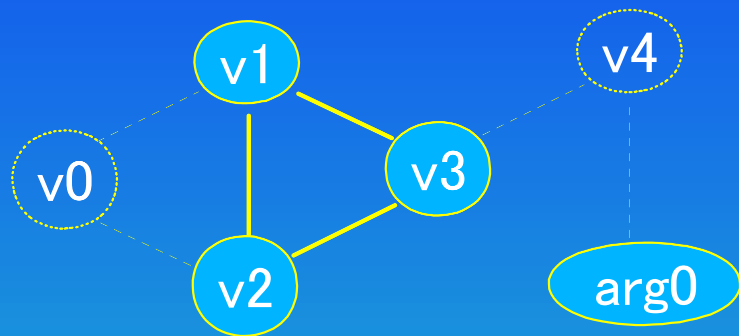


Interference Graph

push low-degree nodes

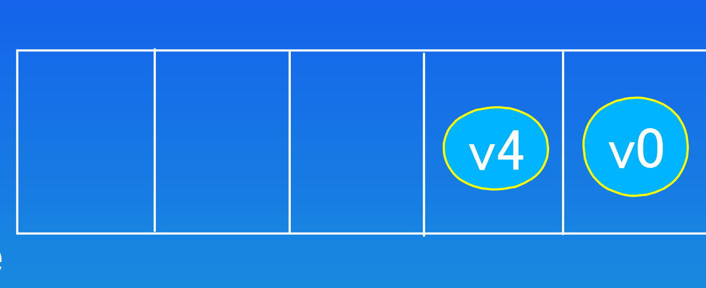


Chaitin-Style Coloring Simplification

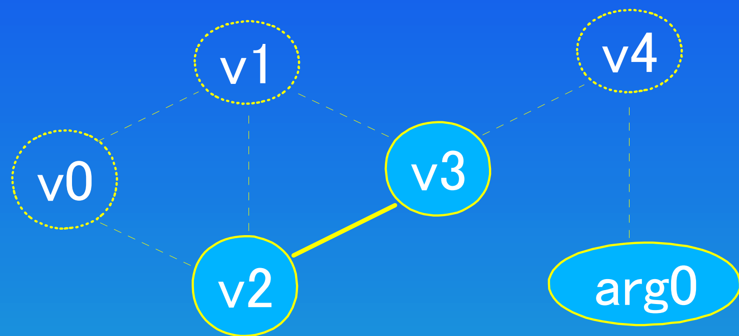


Interference Graph

push low-degree nodes

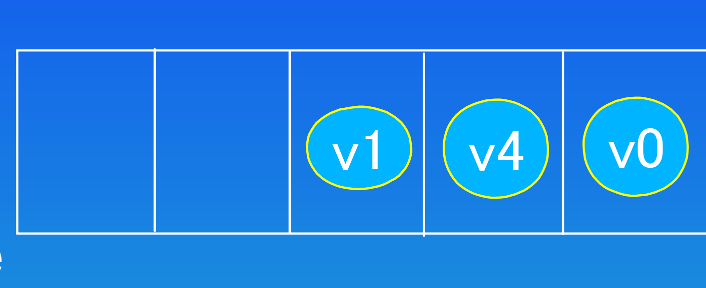


Chaitin-Style Coloring Simplification

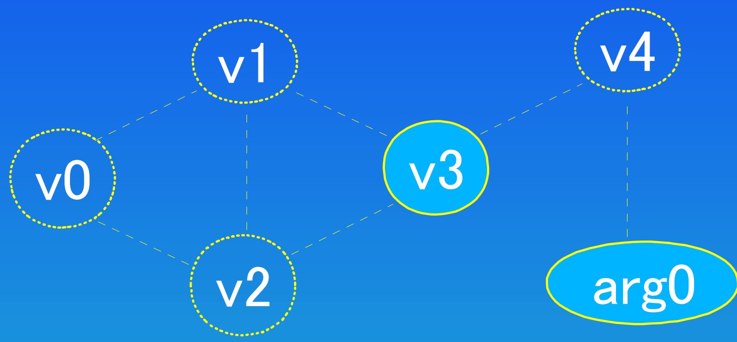


Interference Graph

push low-degree nodes

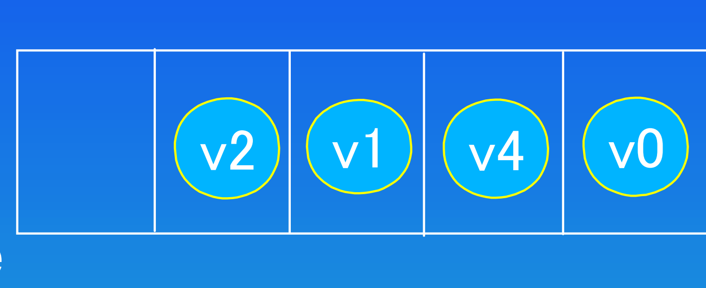


Chaitin-Style Coloring Simplification

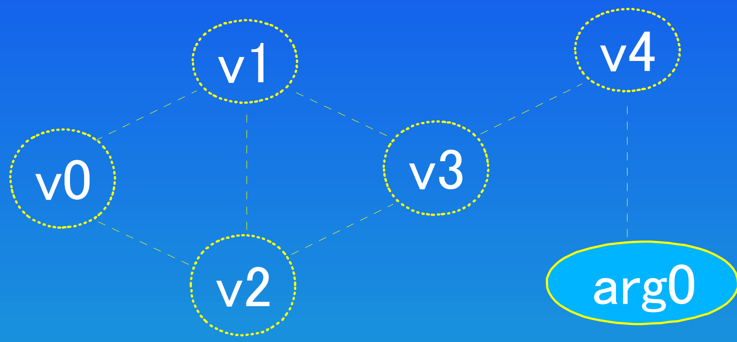


Interference Graph

push low-degree nodes



Chaitin-Style Coloring Simplification



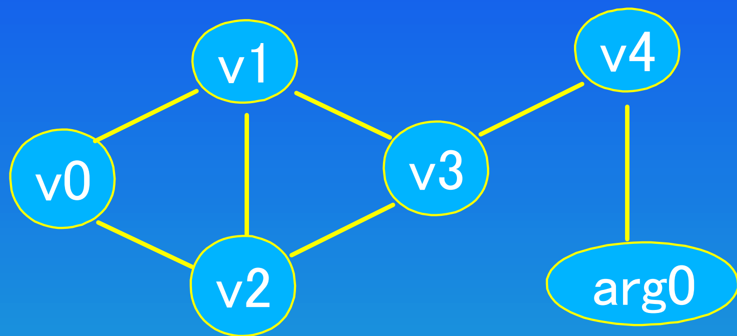
Interference Graph

push low-degree nodes

3-Colorability Established!

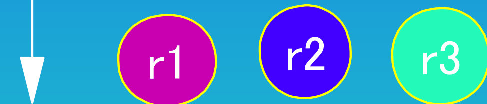
3−Coloring Register Selection Order

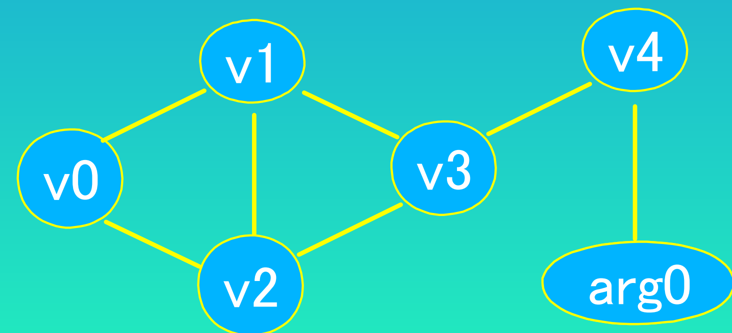
Chaitin-Style Coloring
Register Selection



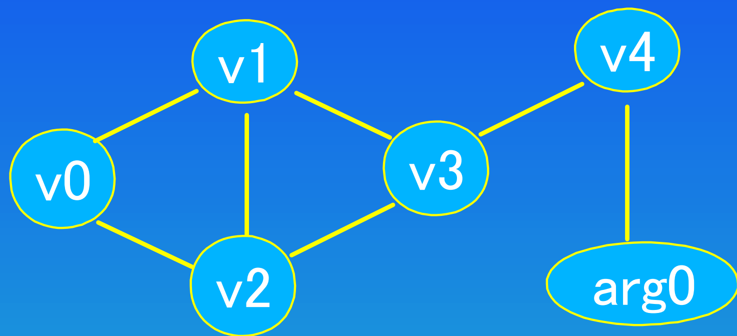
Interference Graph

pop node and
select colors

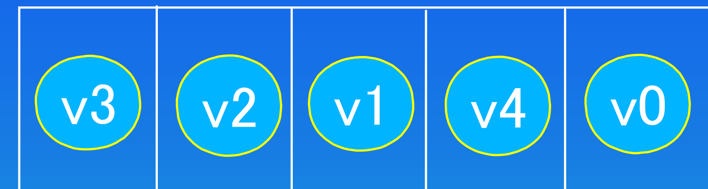




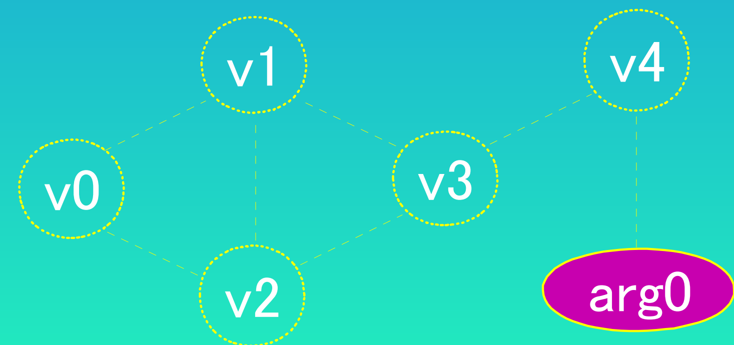
Chaitin-Style Coloring
Register Selection



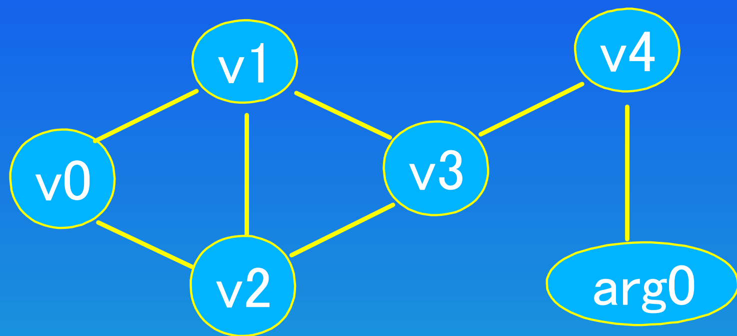
Interference Graph



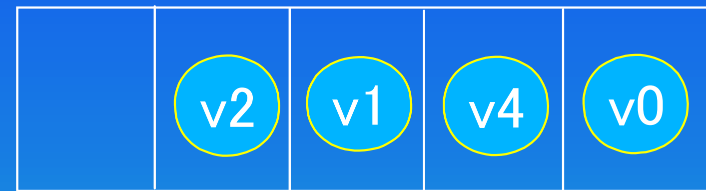
pop node and
select colors



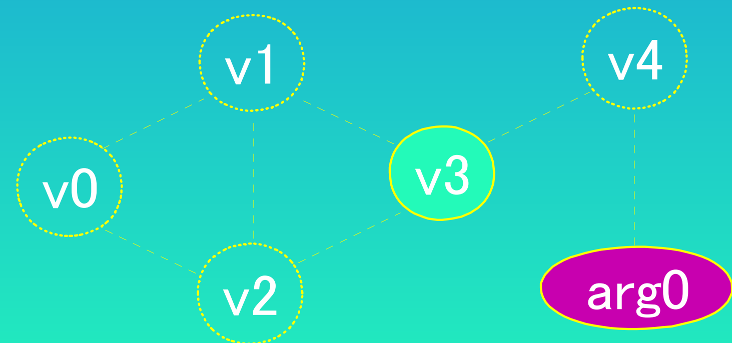
Chaitin-Style Coloring
Register Selection



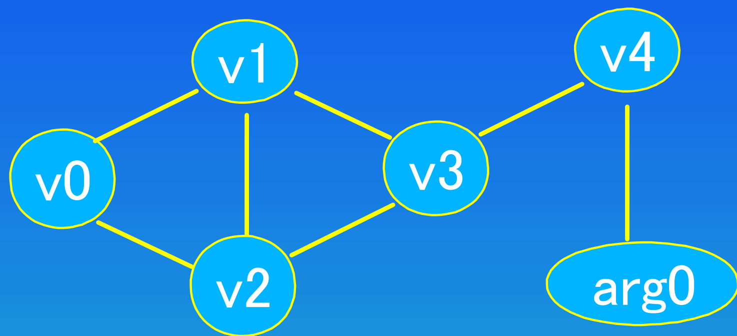
Interference Graph



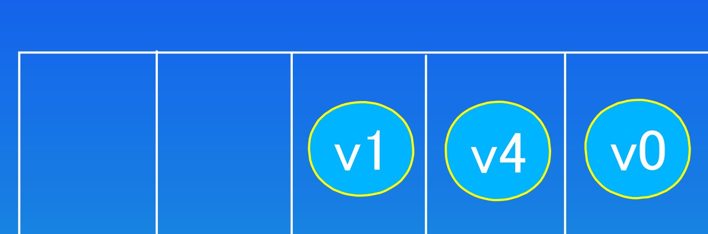
pop node and
select colors



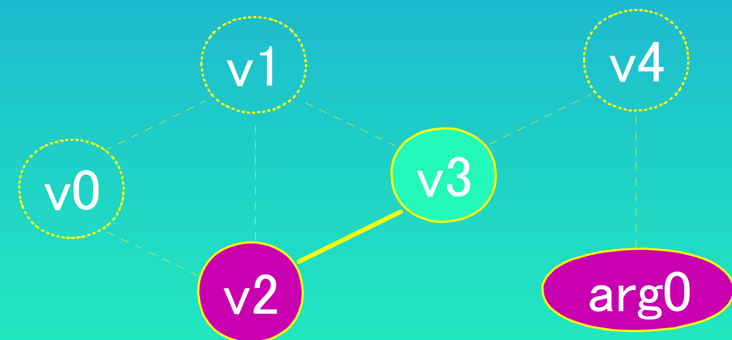
Chaitin-Style Coloring
Register Selection



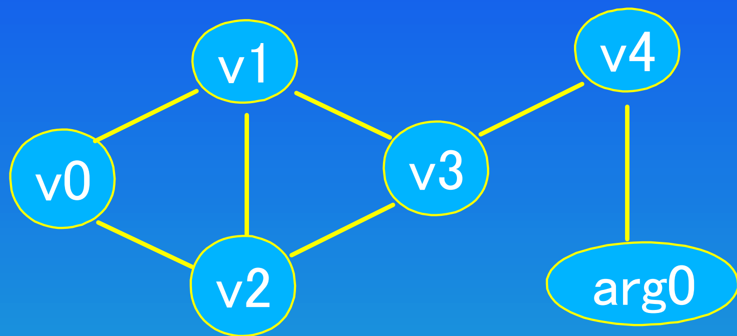
Interference Graph



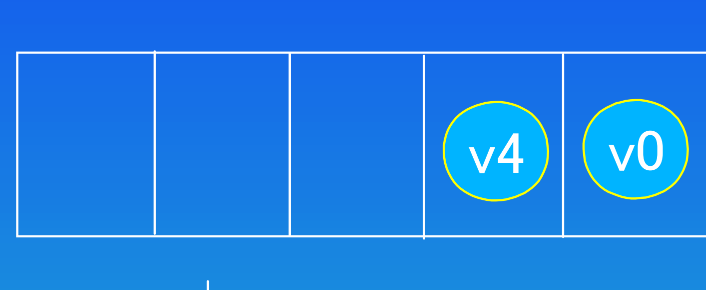
pop node and
select colors



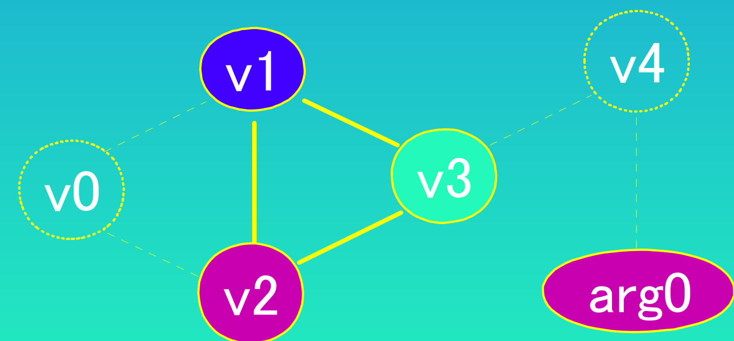
Chaitin-Style Coloring Register Selection



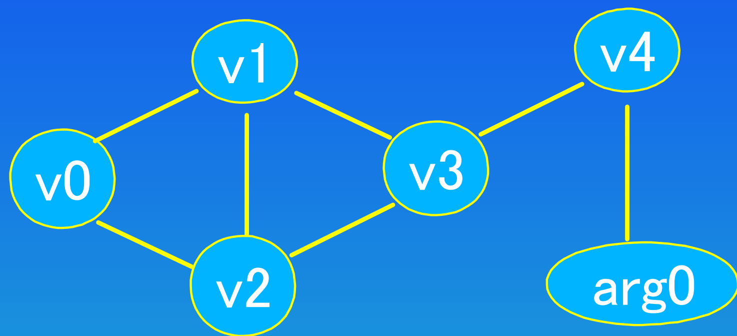
Interference Graph



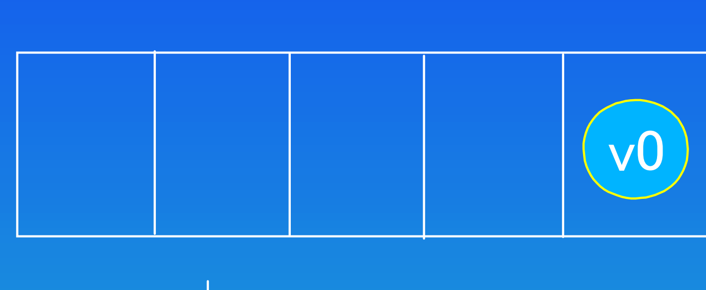
pop node and select colors



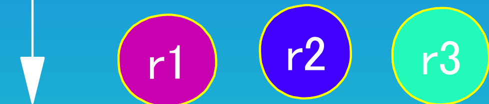
Chaitin-Style Coloring
Register Selection

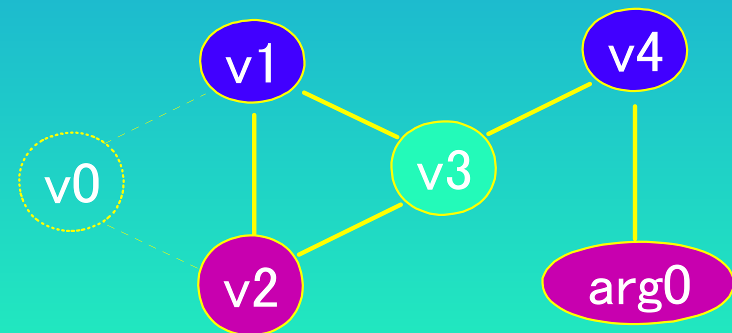


Interference Graph

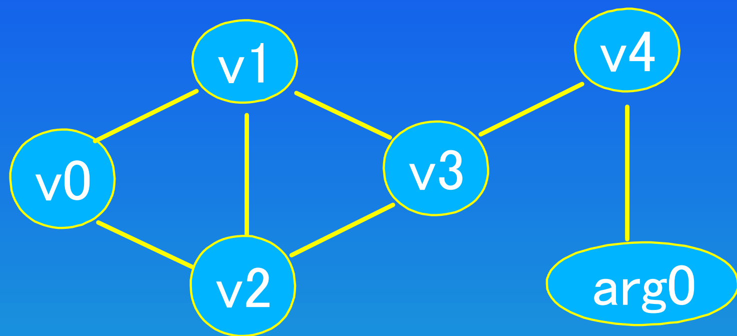


pop node and
select colors

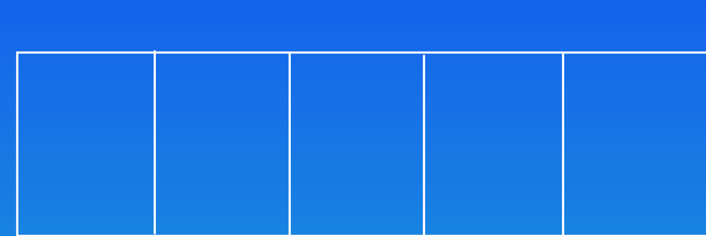




Chaitin-Style Coloring Register Selection

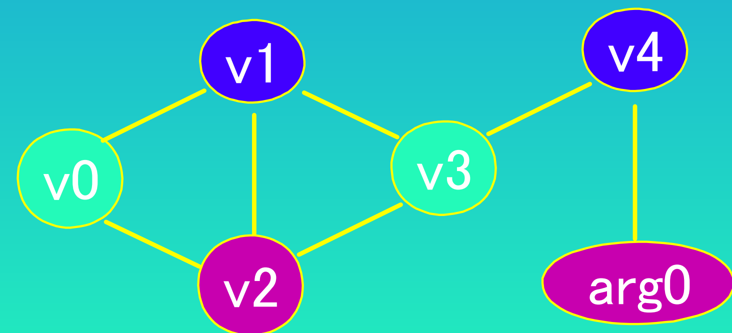


Interference Graph



pop node and select colors

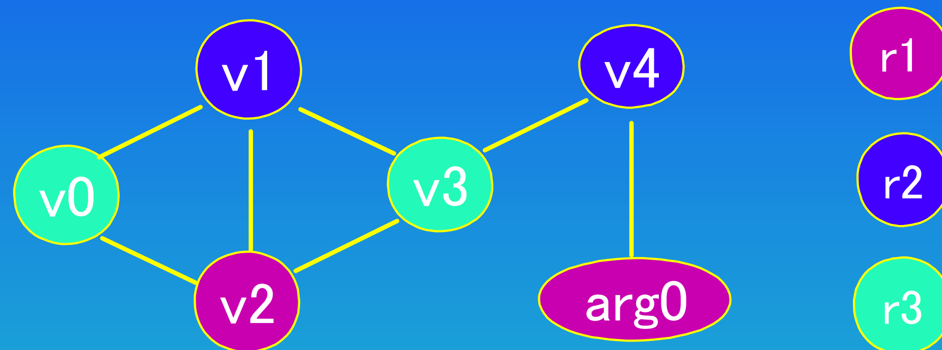
3-Coloring Obtained!



Chaitin-Style Coloring Code Generation

```
        v0 = [arg0]
L1:     v1 = [v0]
        v2 = [v0+4]
        v3 = v0
        v4 = v1 + v2
        arg0 = v3
        call
        v0 = v4+1
        if v0 != 0 goto L1
        ret
```

Coloring Result

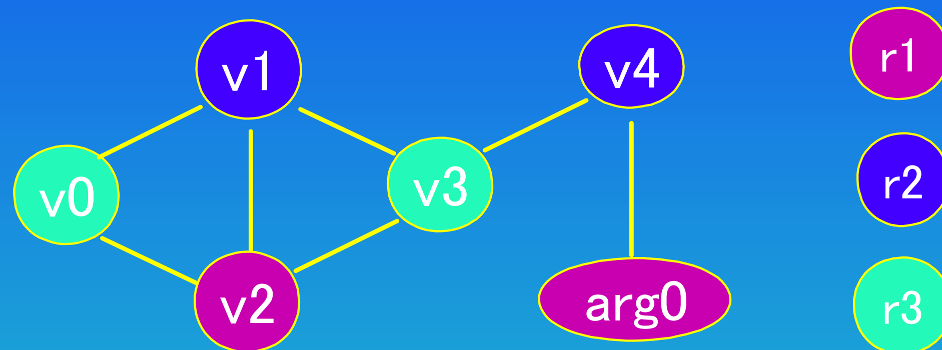


Conventions

- r1: first argument, return value, volatile
- r2: second argument, volatile
- r3: non-volatile

Chaitin-Style Coloring Result

```
      save r3
      r3 = [r1]
L1:   r2 = [r3]
      r1 = [r3+4]
      r2 = r2 + r1
      r1 = r3
      save r2
      call
      restore r2
      r3 = r2+1
      if r3 != 0 goto L1
      restore r3
      ret
```



Conventions:

- r1: first argument, return value, volatile
- r2: second argument, volatile
- r3: non-volatile

Chaitin-Style Coloring Observations

```
      save r3
      r3 = [r1]
L1:   r2 = [r3]
      r1 = [r3+4]
      r2 = r2 + r1
      r1 = r3
      save r2
      call
      restore r2
      r3 = r2+1
      if r3 != 0 goto L1
      restore r3
      ret
```

Chance to use a paired load?

Trivial copy?

Caller-side saving unnecessary?

Motivation

- Registers are not symmetrical for Graph Coloring
  - same registers for trivial copies (coalescing)
  - volatile and non-volatile registers
  - specific registers for:
    - arguments
    - return values
    - operations
  - specific pairs of registers for some instructions

Prior Approaches

- Minimizing trivial copies
  - Register coalescing
    - iterated coalescing [Geoge96]
    - optimistic coalescing [Park98]

- Exploiting volatile and non-volatile registers
  - call-cost directed register allocation [Lueh97]

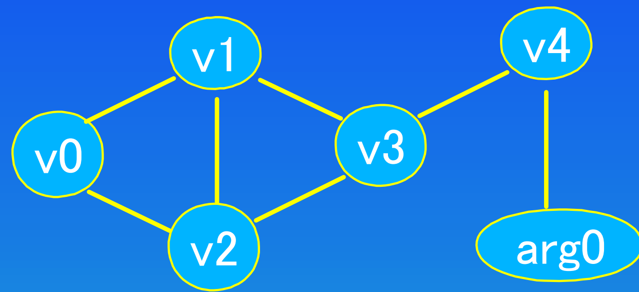
Our Focus

- Seek a register selection order
  - to maximize the chace of satisfying various kinds of register selection requirements
  - still to keep N-colorability
- Integrate the register selection requirements
  - to prioritize those requirements

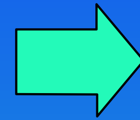
Approach

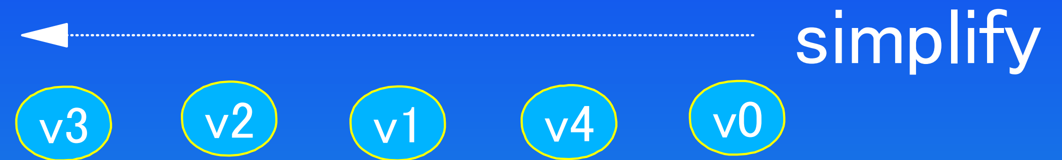
- Build a ***coloring precedence graph***.
  - represent relaxed selection ordering for N-coloring.
- Build a ***register preference graph***.
  - represent all register selection requirements uniformly.
  - with weights based on benefits.

Coloring Precedence Graph



Interference Graph

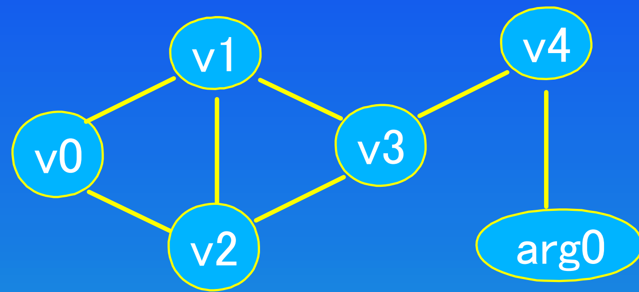




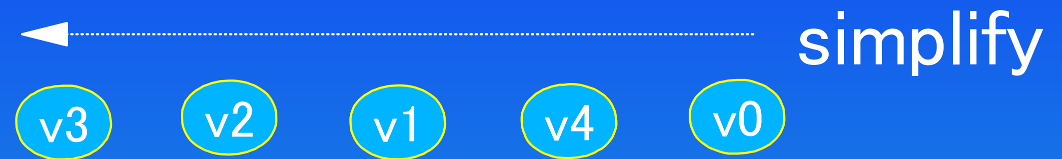
A Simplification Order

to Keep 3-Colorobility

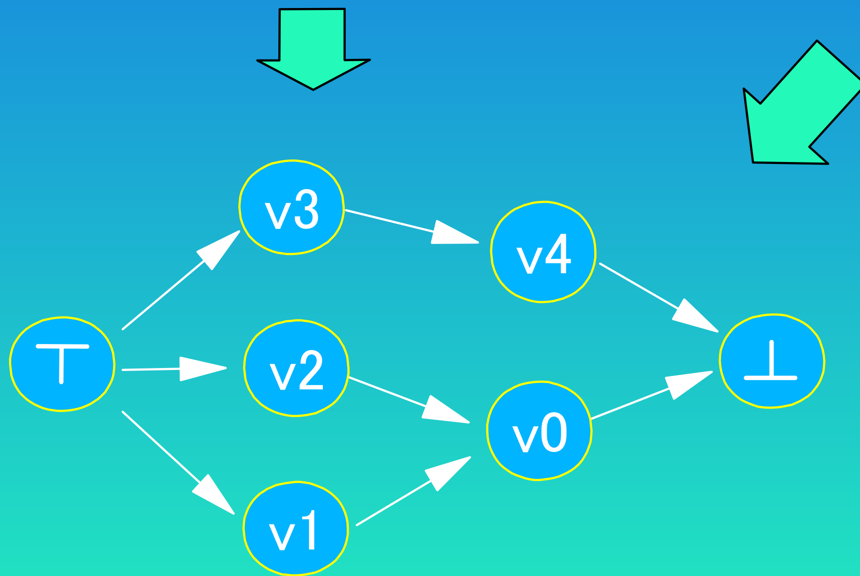
Coloring Precedence Graph



Interference Graph



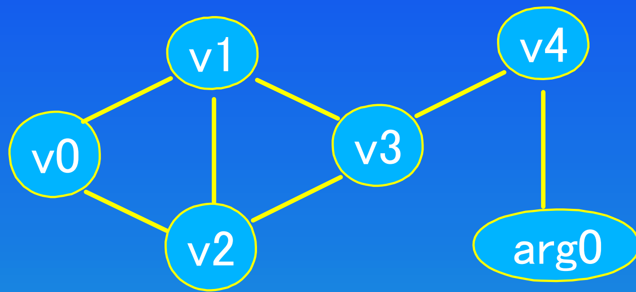
A Simplification Order



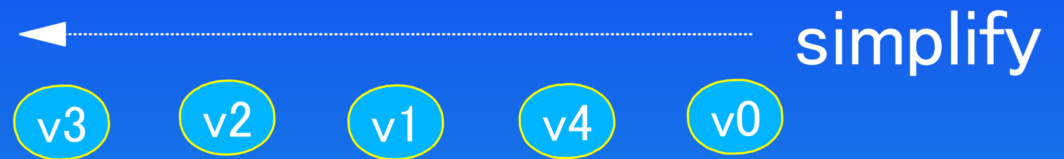
Coloring Precedence Graph

to Keep 3-Colorability

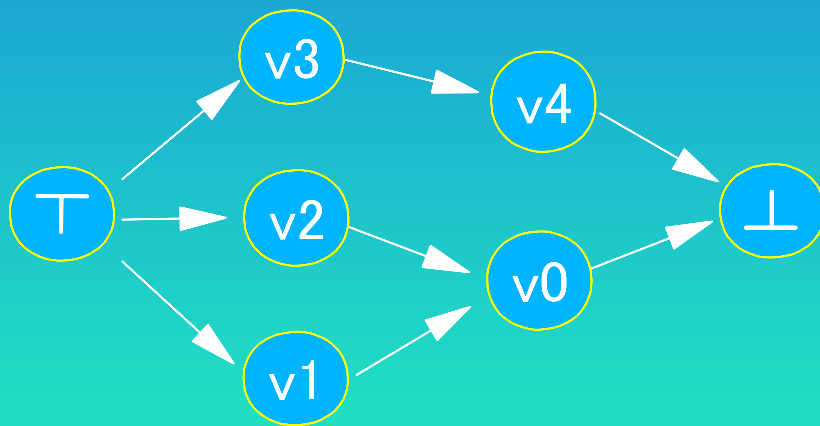
Coloring Precedence Graph



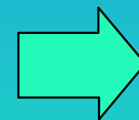
Interference Graph

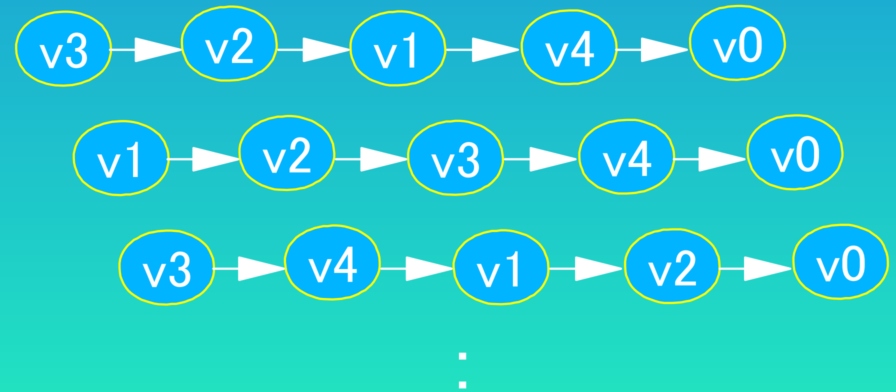


A Simplification Order



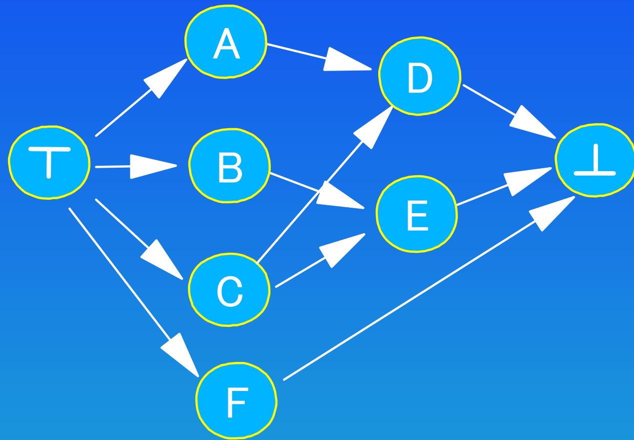
Coloring Precedence Graph
to Keep 3-Colorability





3−colorable orders

Coloring Precedence Graph semantics

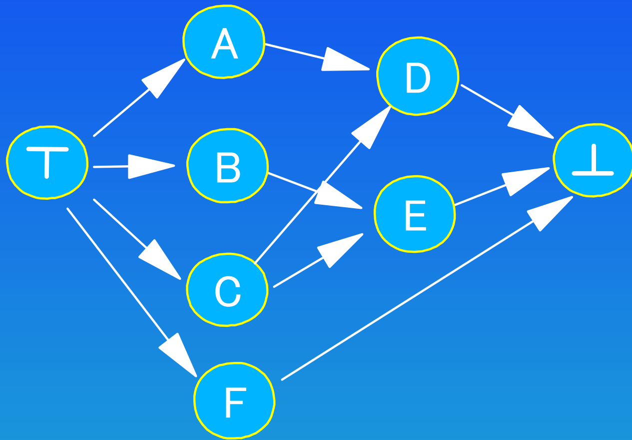


Removing A requires removing B

Nothing requires Removing D

Removing C requires nothing

Coloring Precedence Graph semantics

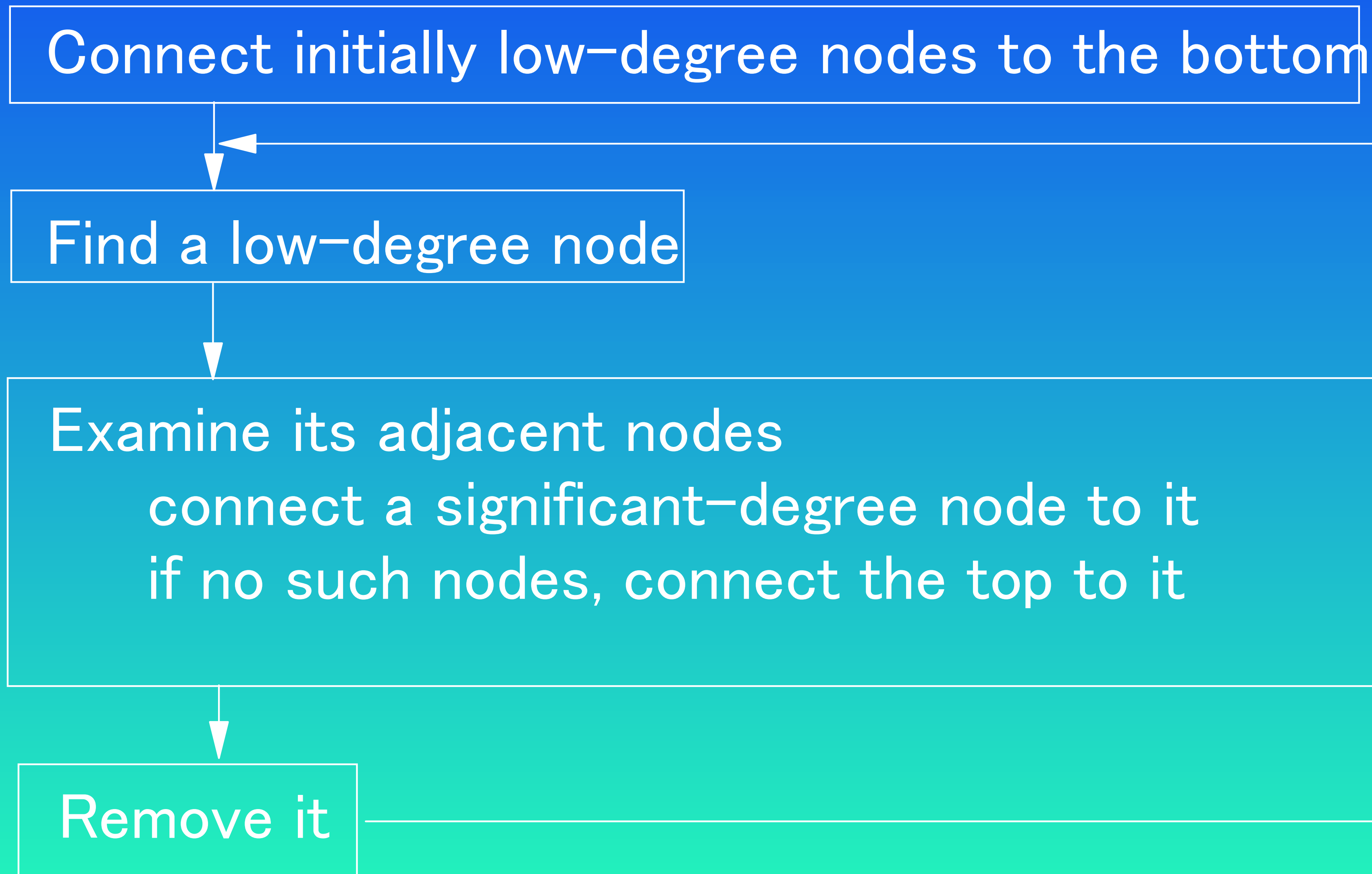


Removing A requires removing B
(A cannot be removed until removing B)

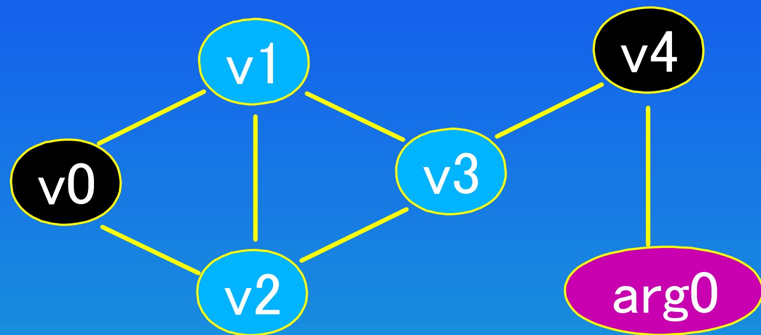
Nothing requires Removing D
(Removing D affects nothing)

Removing C requires nothing
(C is initially removable)

Coloring Precedence Graph generation

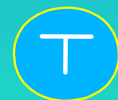


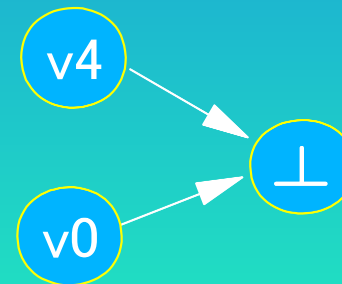
Coloring Precedence Graph generation



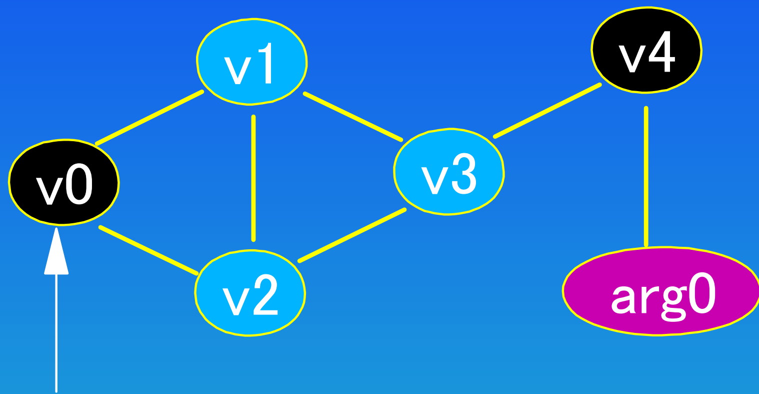
Connect low-degree nodes to the bottom.

low-degree

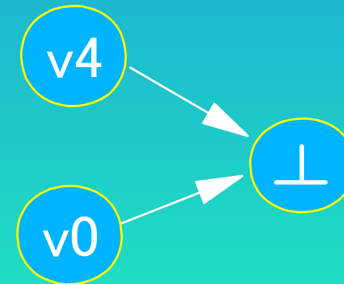




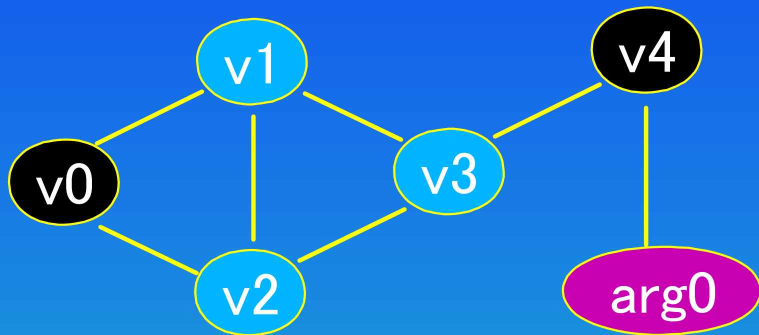
Coloring Precedence Graph generation



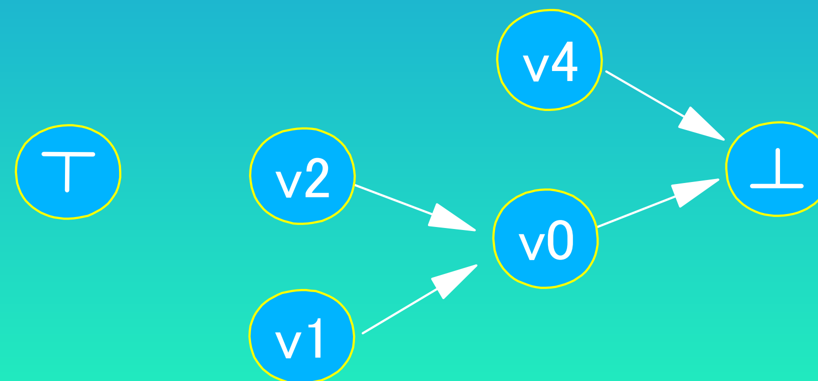
Find a low-degree node.



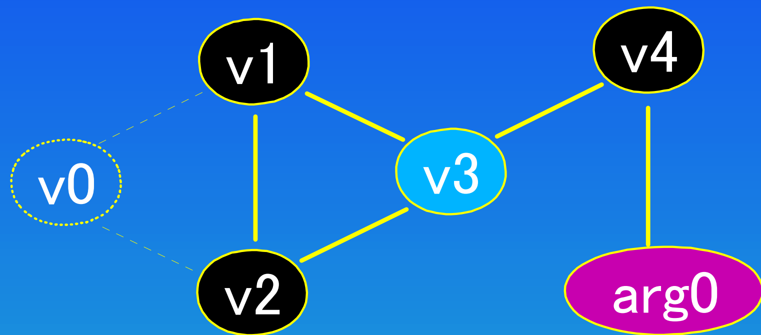
Coloring Precedence Graph generation



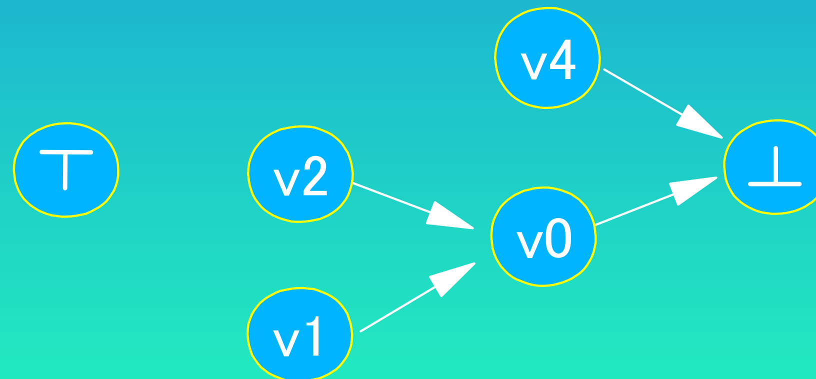
Examine the adjacent nodes



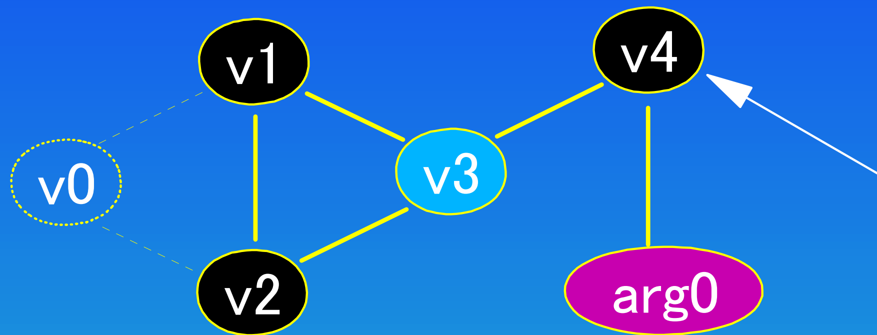
Coloring Precedence Graph generation



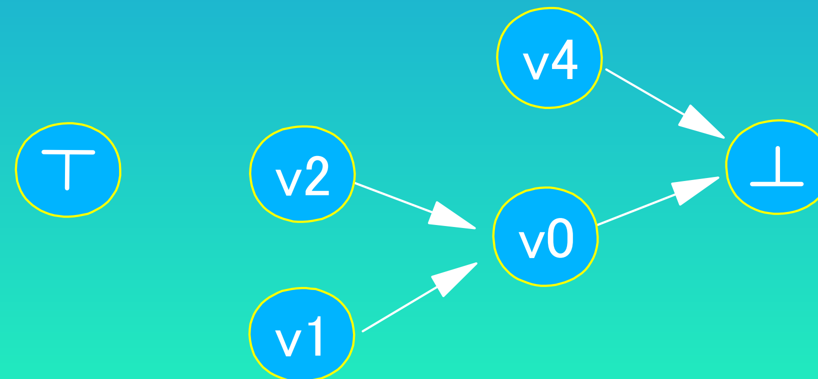
Remove the node.



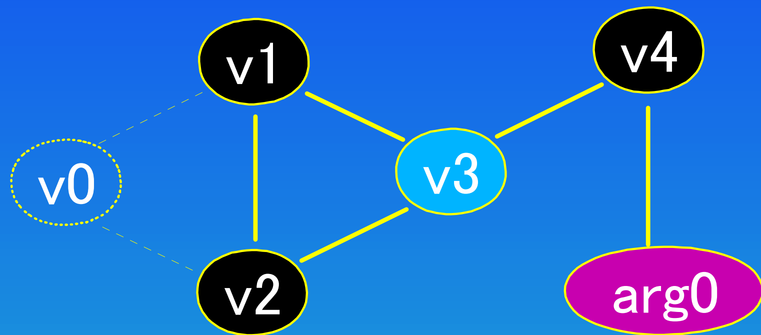
Coloring Precedence Graph generation



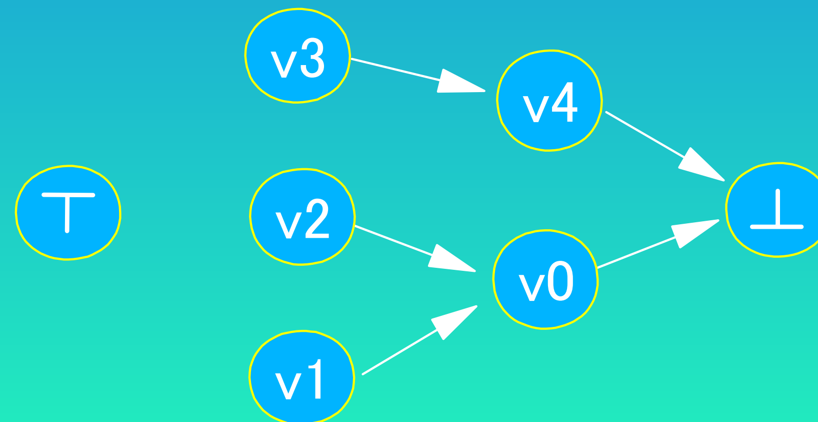
Find a low-degree node.



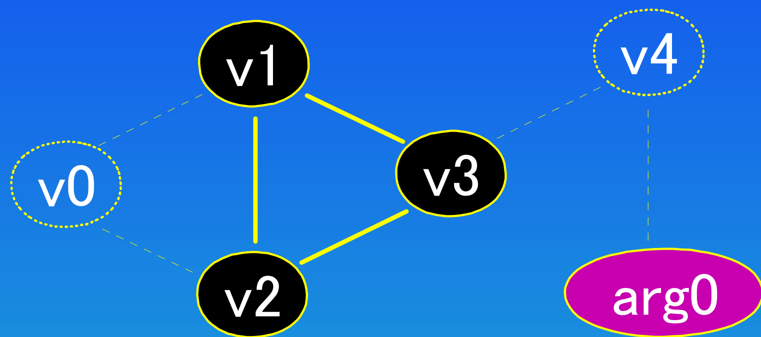
Coloring Precedence Graph generation



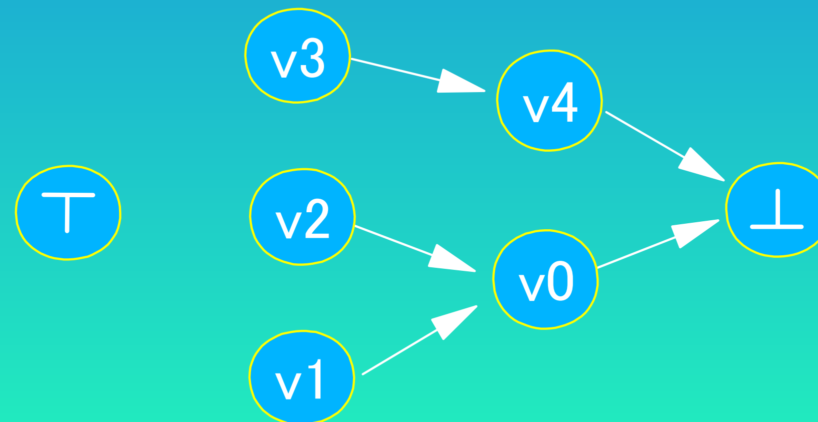
Examine the adjacent nodes



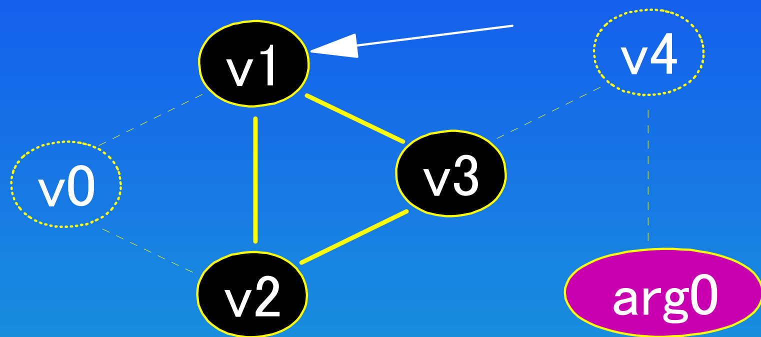
Coloring Precedence Graph generation



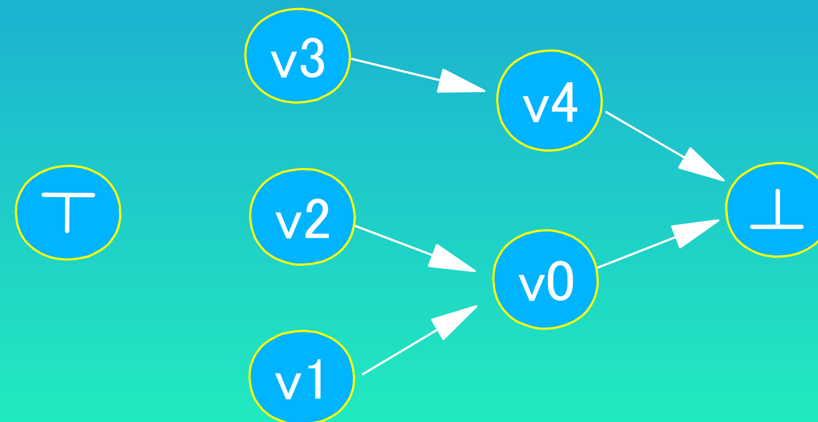
Remove the node.



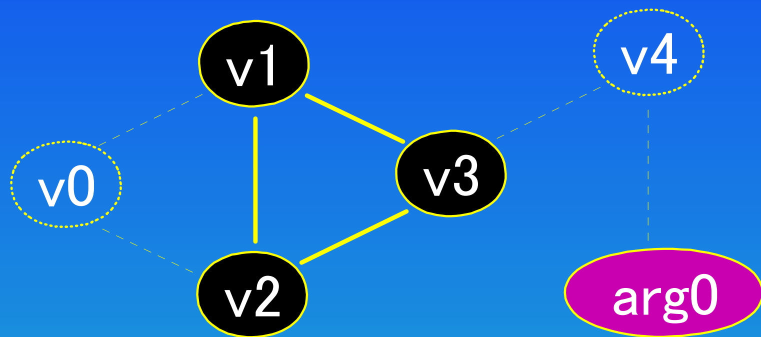
Coloring Precedence Graph generation



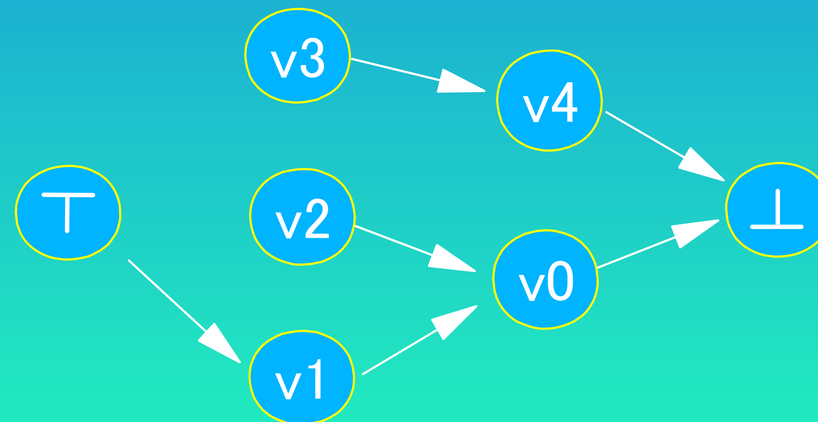
Find a low-degree node.



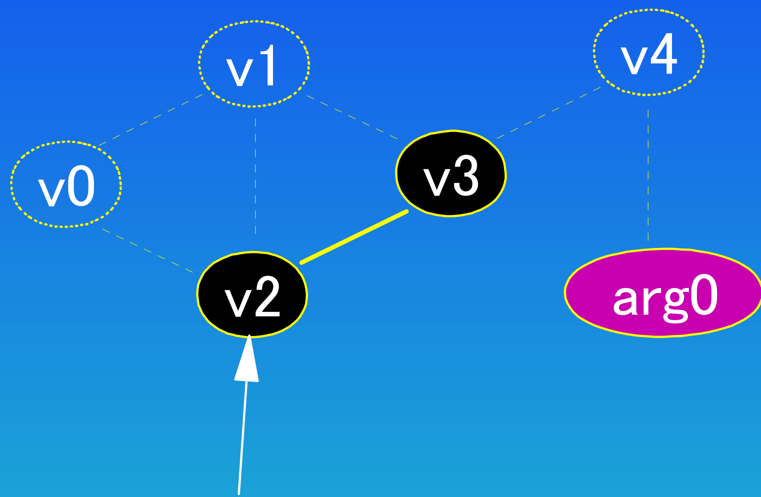
Coloring Precedence Graph generation



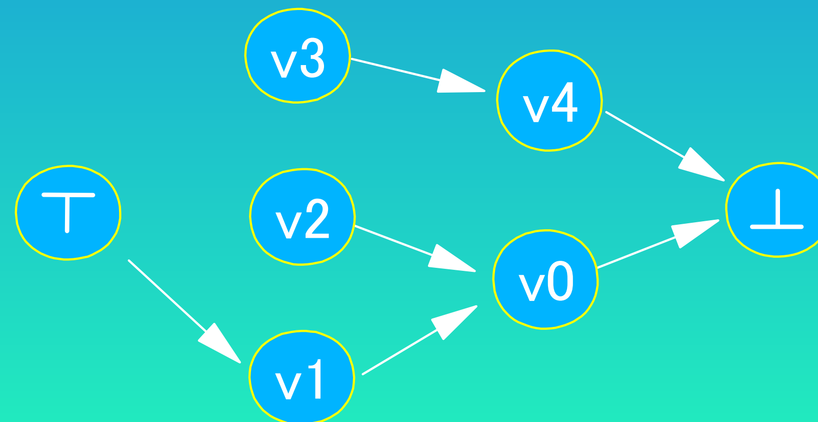
Examine the adjacent nodes



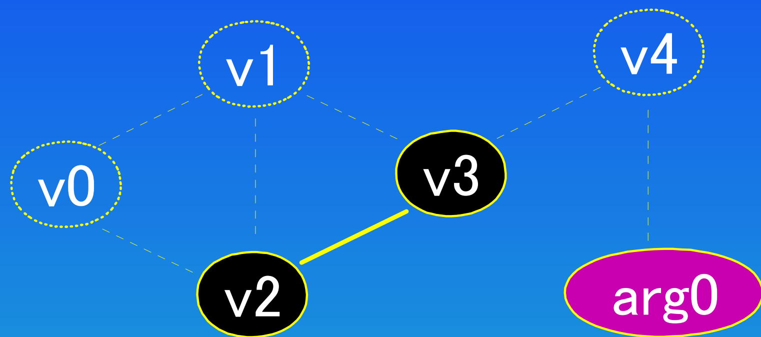
Coloring Precedence Graph generation



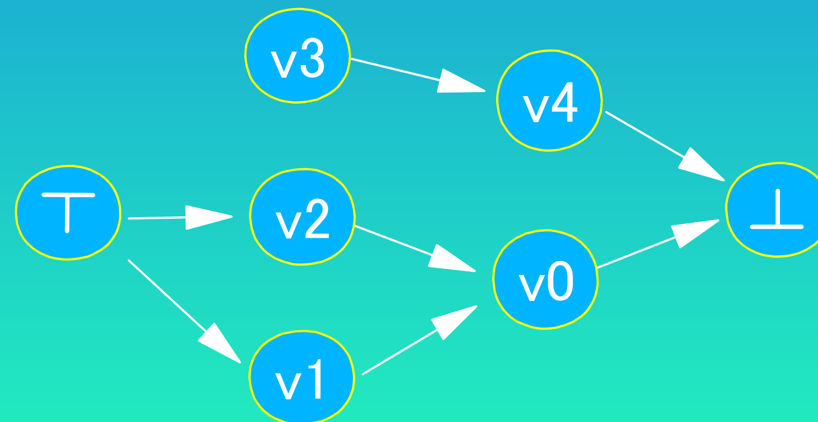
Find a low-degree node.



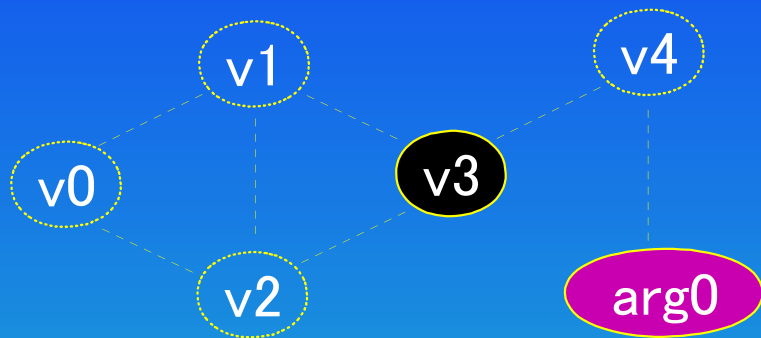
Coloring Precedence Graph generation



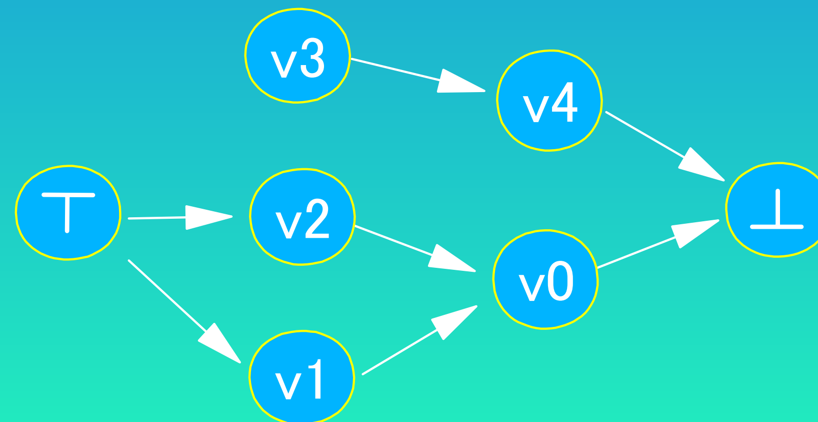
Examine the adjacent nodes



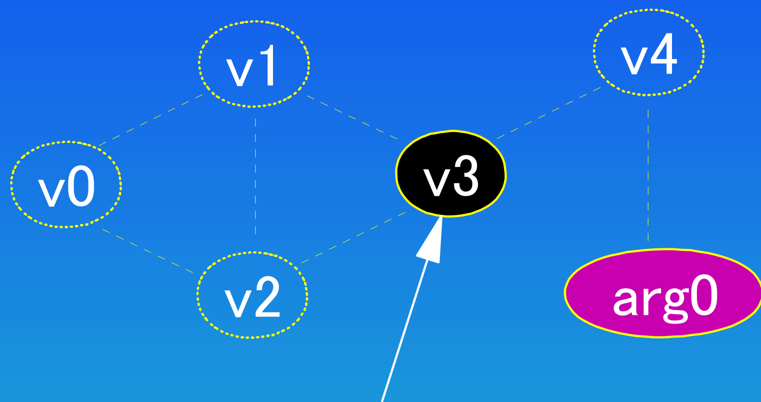
Coloring Precedence Graph generation



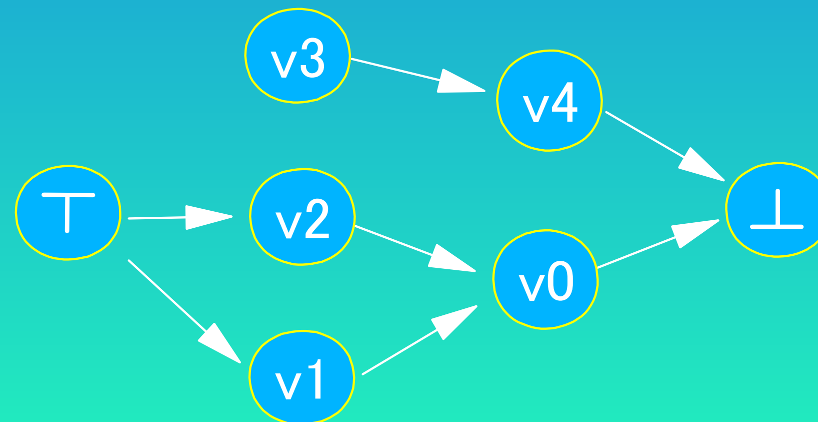
Remove the node.



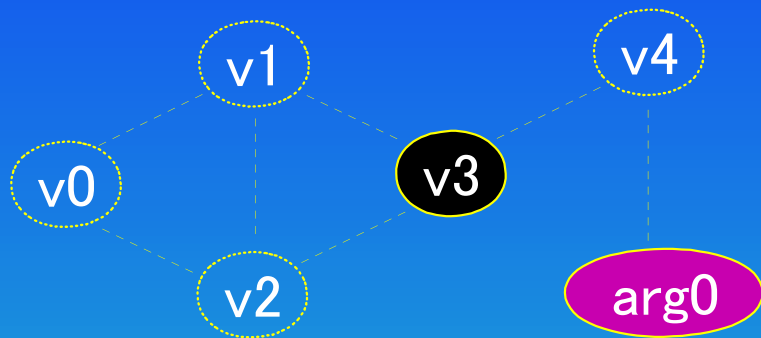
Coloring Precedence Graph generation



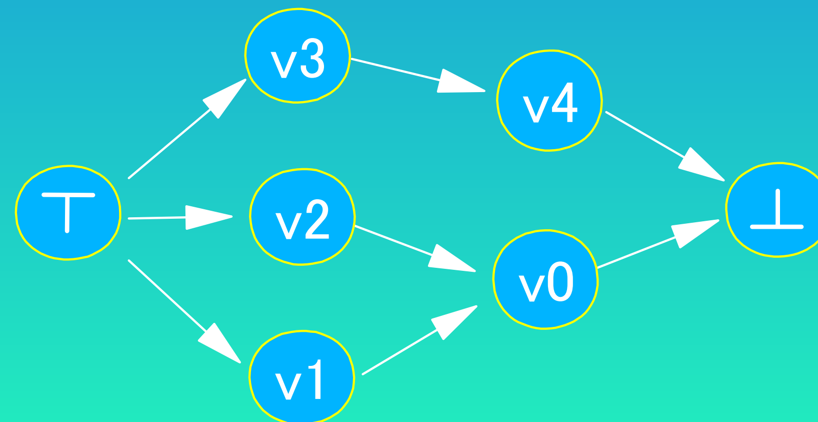
Find a low-degree node.



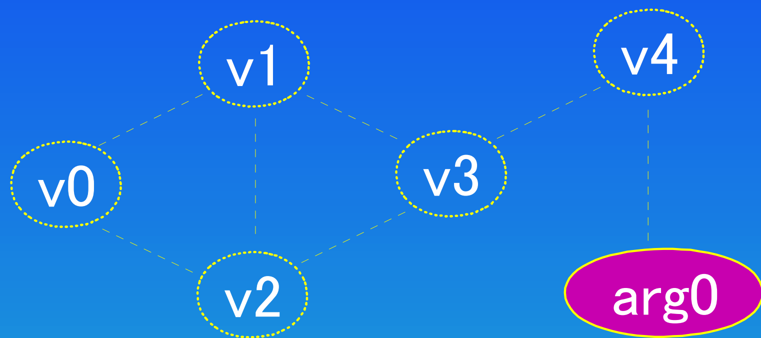
Coloring Precedence Graph generation



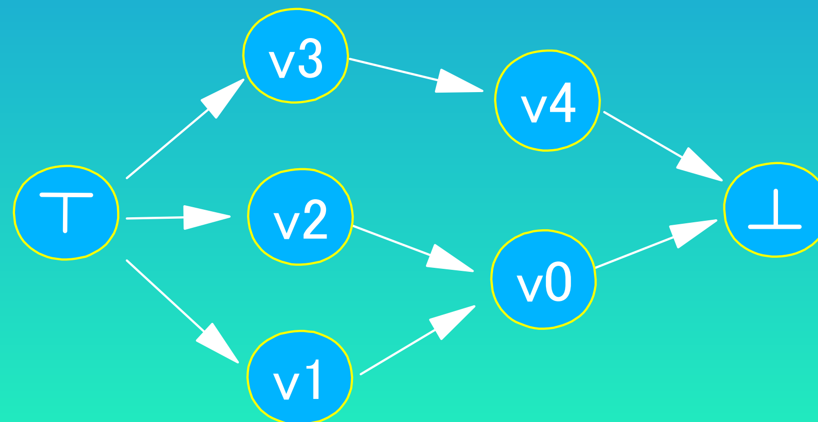
Examine the adjacent nodes



Coloring Precedence Graph generation



Remove the node.

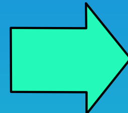


Approach

- Build a *coloring precedence graph*.
  - represent relaxed selection ordering for N-coloring.
- Build a *register preference graph*.
  - represent all register selection requirements uniformly.
  - with weights based on benefits.

Preference Representation

```
      v0 = [arg0]
L1:   v1 = [v0]
      v2 = [v0+4]
      v3 = v0
      v4 = v1 + v2
      arg0 = v3
      call
      v0 = v4+1
      if v0 != 0 goto L1
      ret
```



Map to 3 registers with

Register Preferences:

Conventions:

- v4 prefers a non-volatile reg
- v0,v1,v2,v3 prefer volatile regs

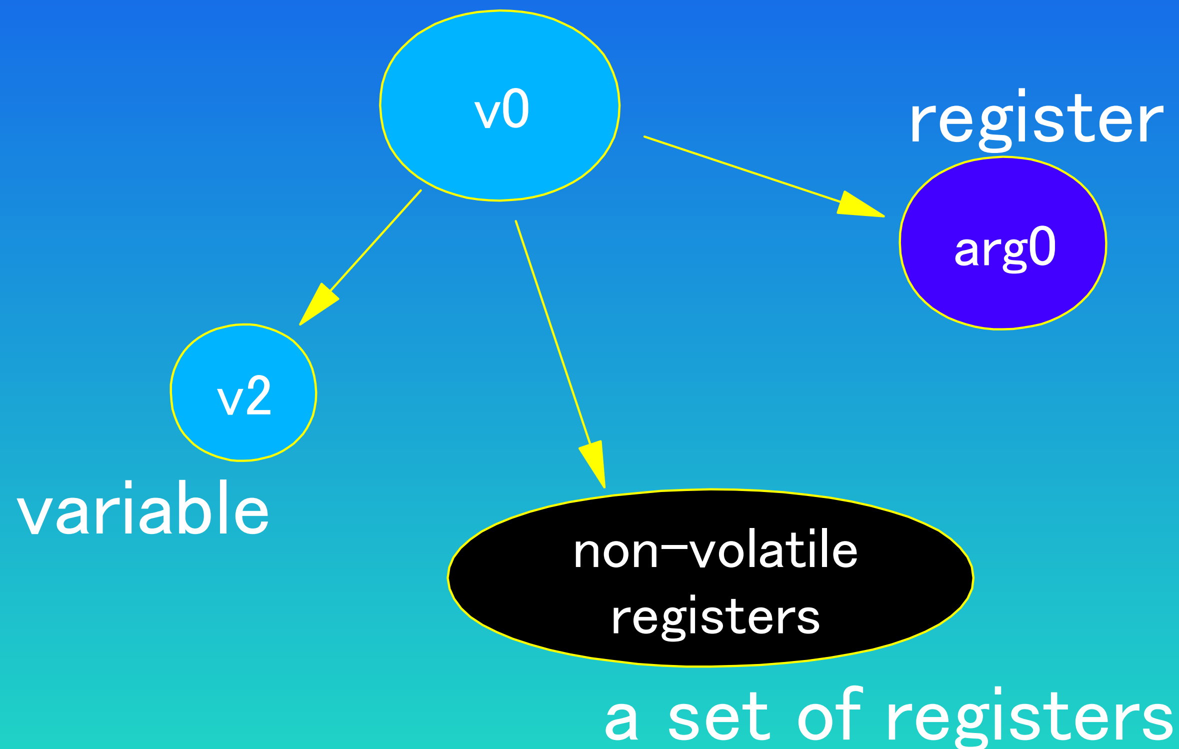
Copy coalescing:

- v3,v0 prefer the same reg
- v3 prefers arg0 (r1)

Operation requirement:

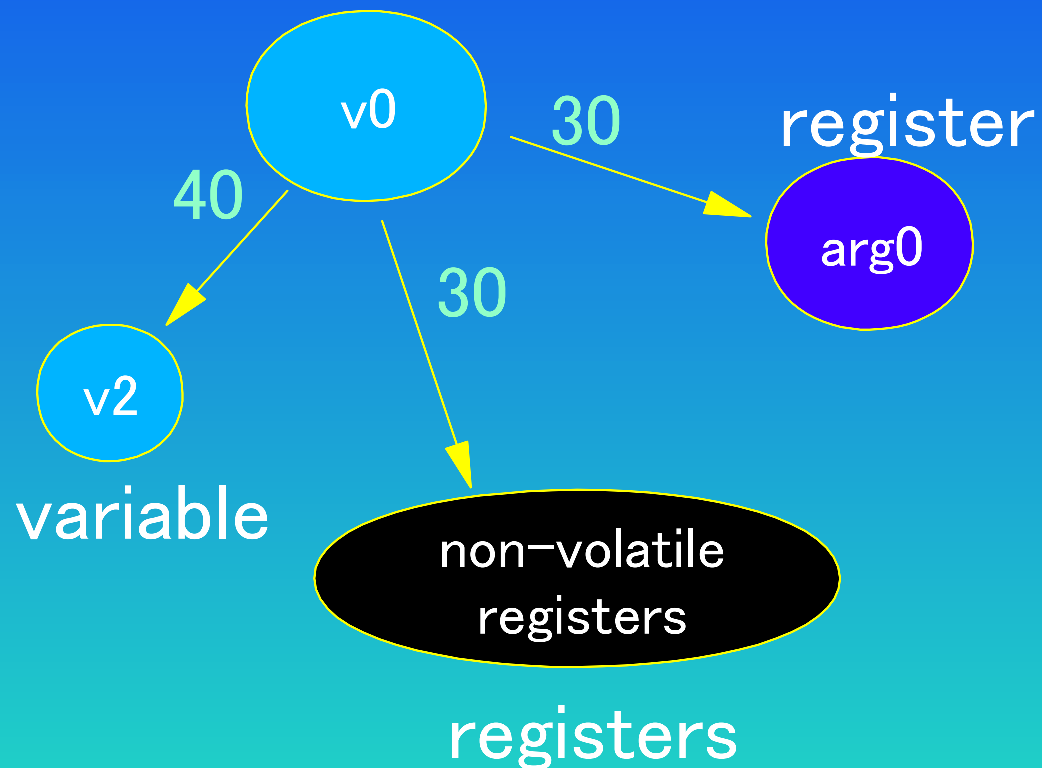
- v1,v2 prefer contiguous regs

Preference Representation graphical notation



Relations Between Nodes

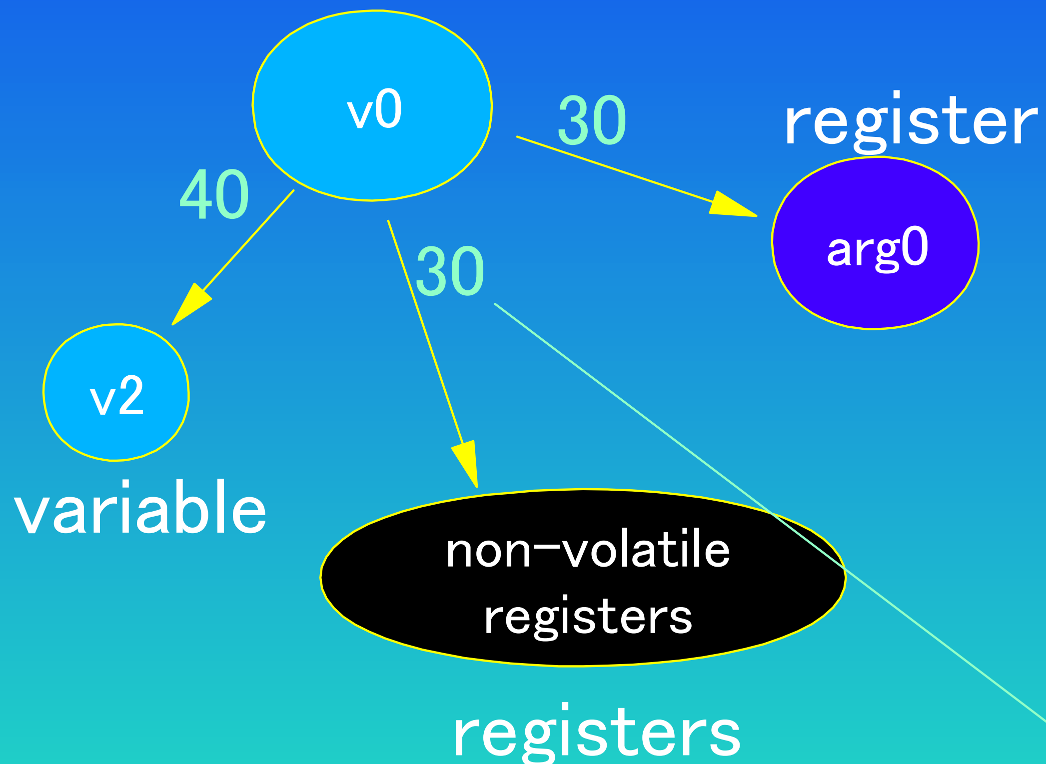
Preference Representation
unified metrics



weights based on benefits in execution cycles

weight = execution cycles saved by honoring the preference * frequency

Preference Representation unified metrics



weights based on benefits in execution cycles

weight = execution cycles saved by honoring the preference * frequency

30 cycles can be saved

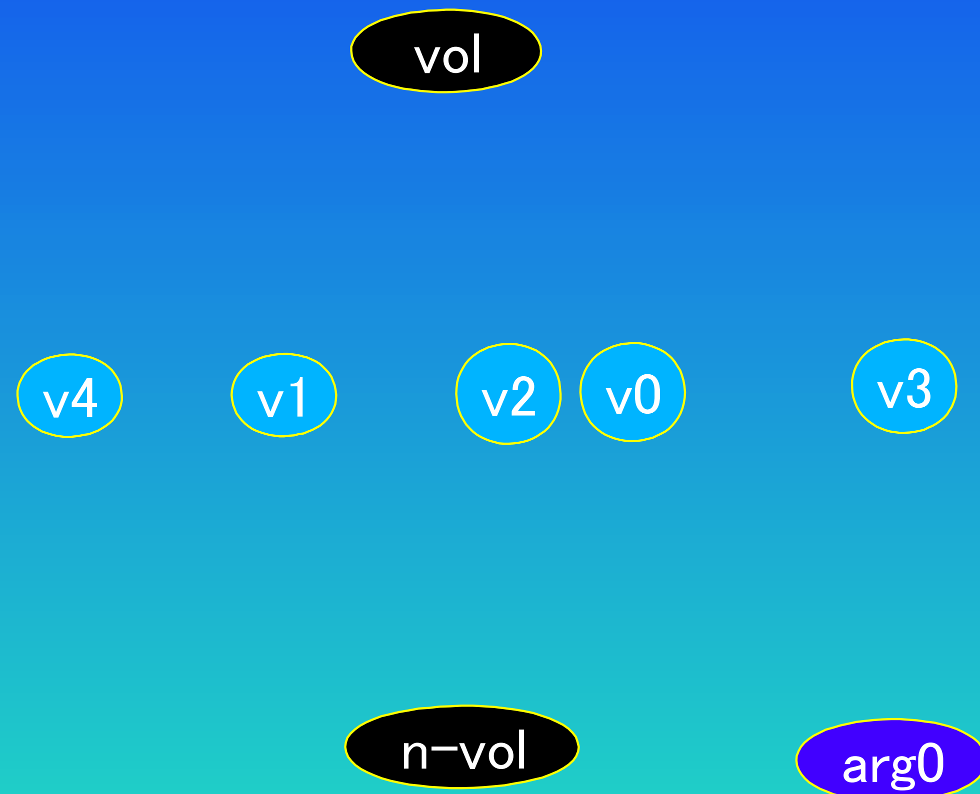
Register Preference Graph

Register Preferences:

Conventions:
- v4 prefers a non-volatile reg
- v0,v1,v2,v3 prefer volatile reg

Copy coalescing:
- v3,v0 prefer the same reg
- v3 prefers arg0 (r1)

Operation requirement:
- v1,v2 prefer contiguous regs



Register Preference Graph

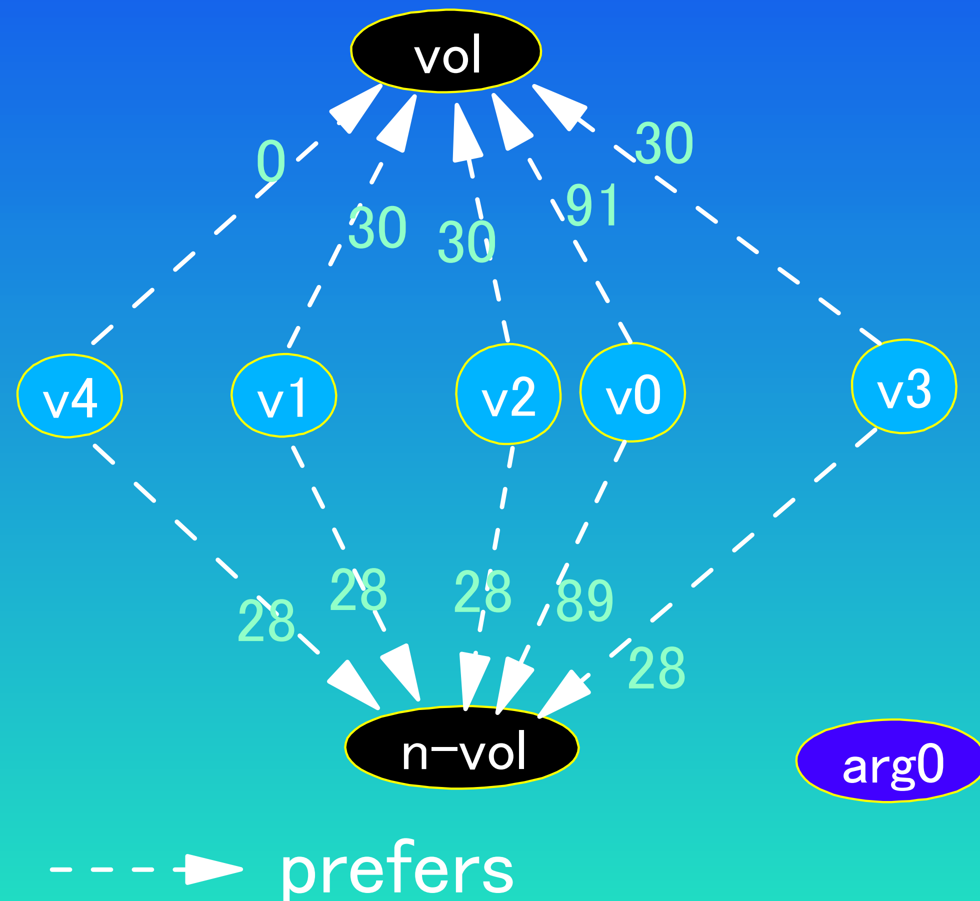
Register Preference Graph

Register Preferences:

Conventions:
- v4 prefers a non-volatile reg
- v0,v1,v2,v3 prefer volatile reg

Copy coalescing:
- v3,v0 prefer the same reg
- v3 prefers arg0 (r1)

Operation requirement:
- v1,v2 prefer contiguous regs



Register Preference Graph

Register Preference Graph

Register Preferences:

Conventions:

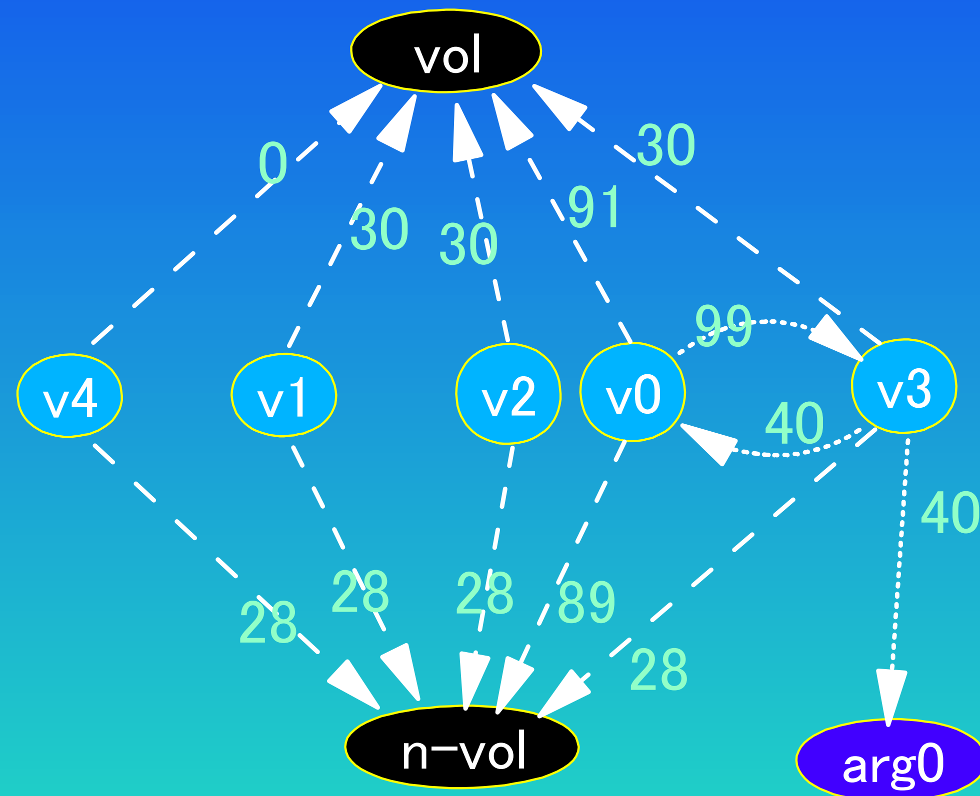
- v4 prefers a non-volatile reg
- v0,v1,v2,v3 prefer volatile reg

Copy coalescing:

- v3,v0 prefer the same reg
- v3 prefers arg0 (r1)

Operation requirement:

- v1,v2 prefer contiguous regs



Register Preference Graph

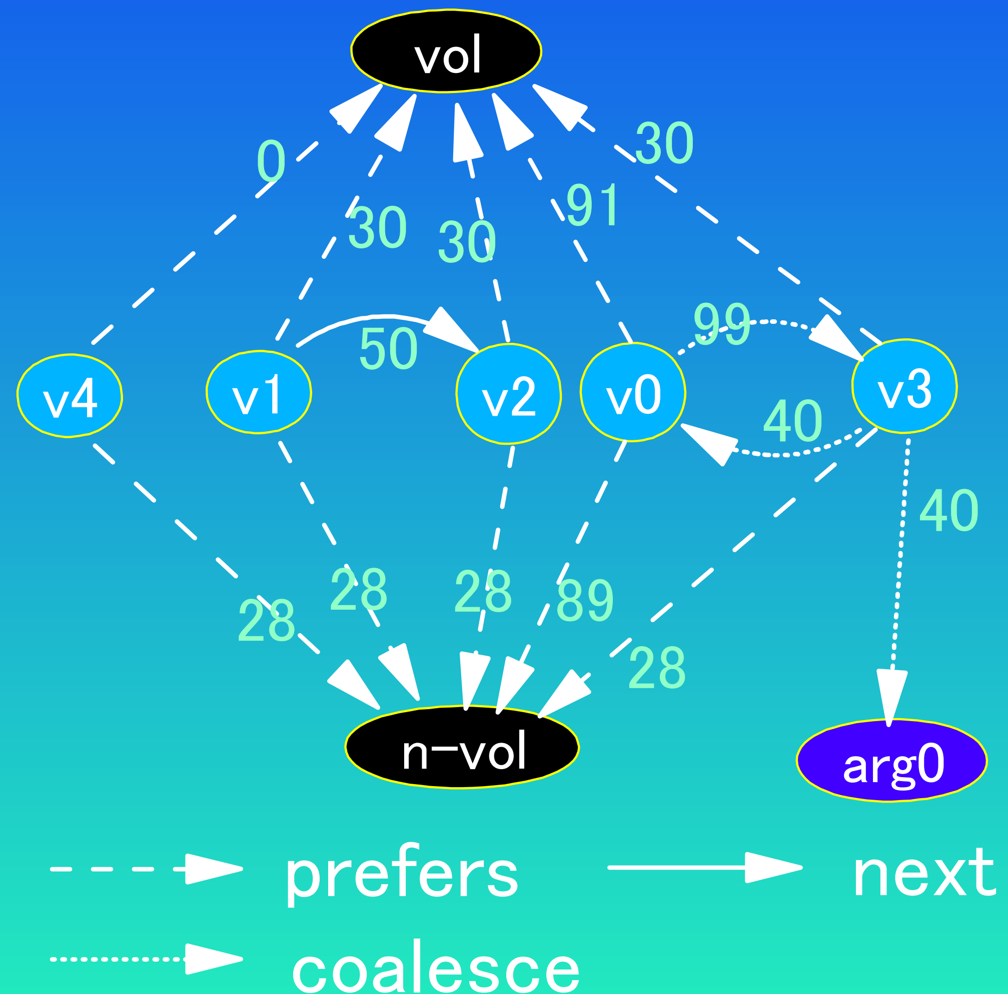
Register Preference Graph

Register Preferences:

Conventions:
- v4 prefers a non-volatile reg
- v0,v1,v2,v3 prefer volatile reg

Copy coalescing:
- v3,v0 prefer the same reg
- v3 prefers arg0 (r1)

Operation requirement:
- v1,v2 prefer contiguous regs



Register Preference Graph

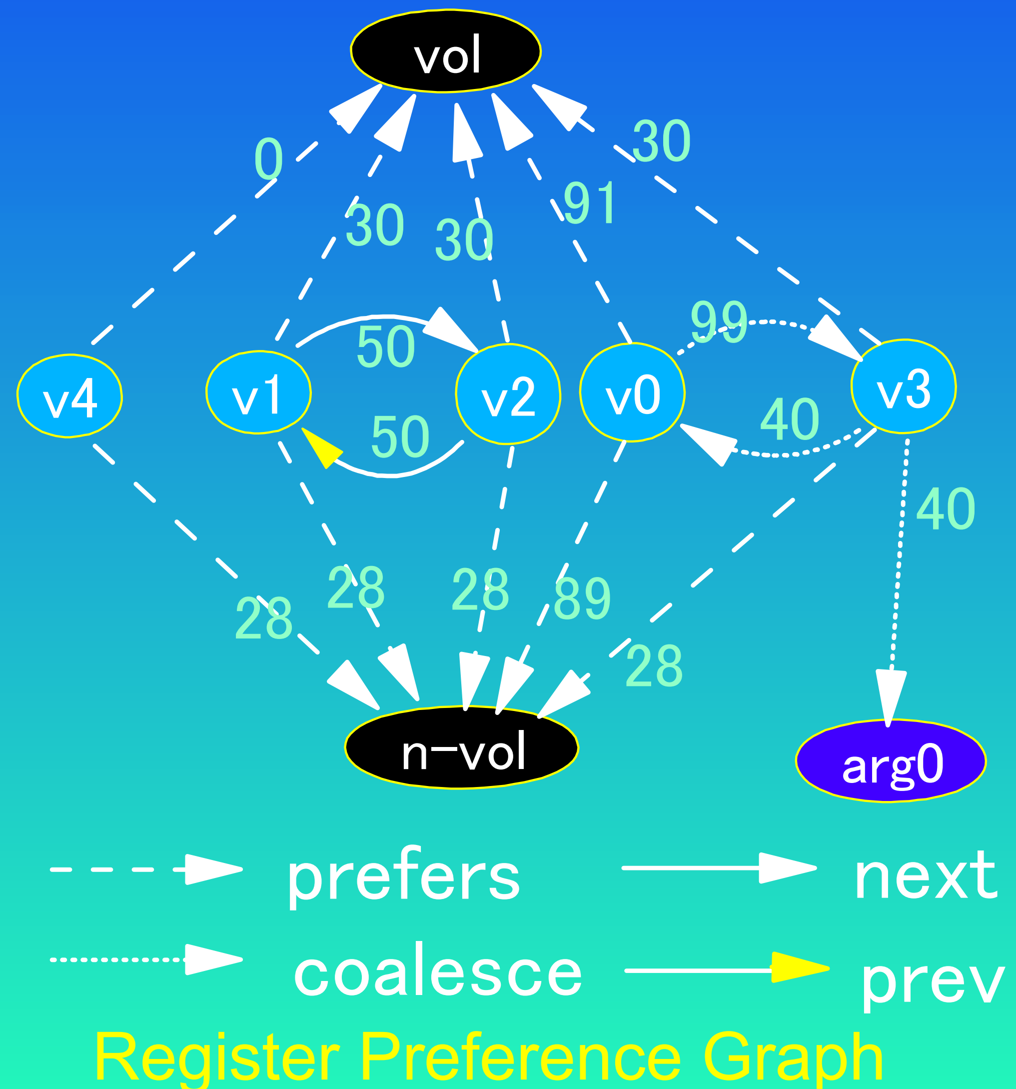
Register Preference Graph

Register Preferences:

Conventions:
- v4 prefers a non-volatile reg
- v0,v1,v2,v3 prefer volatile reg

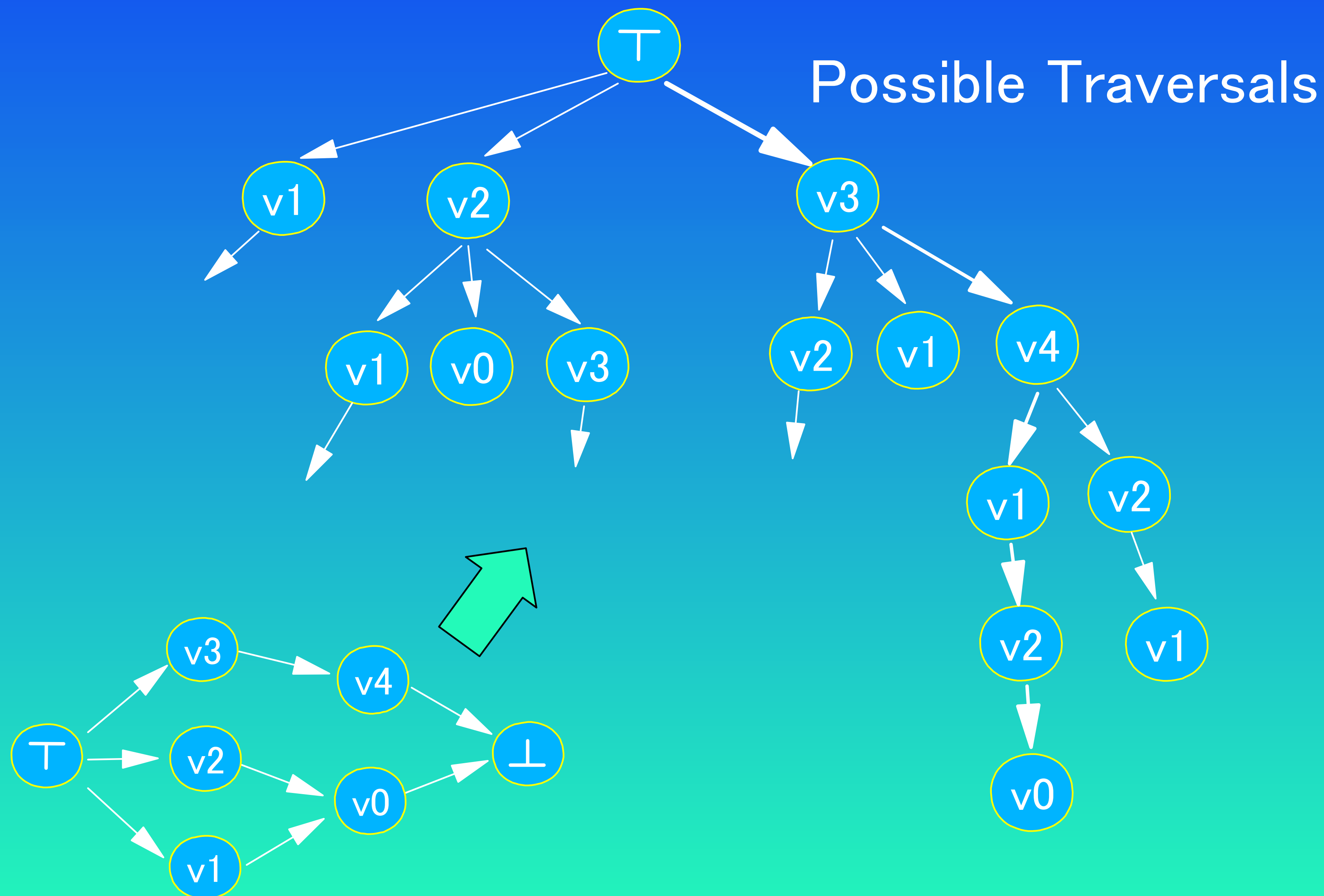
Copy coalescing:
- v3,v0 prefer the same reg
- v3 prefers arg0 (r1)

Operation requirement:
- v1,v2 prefer contiguous regs

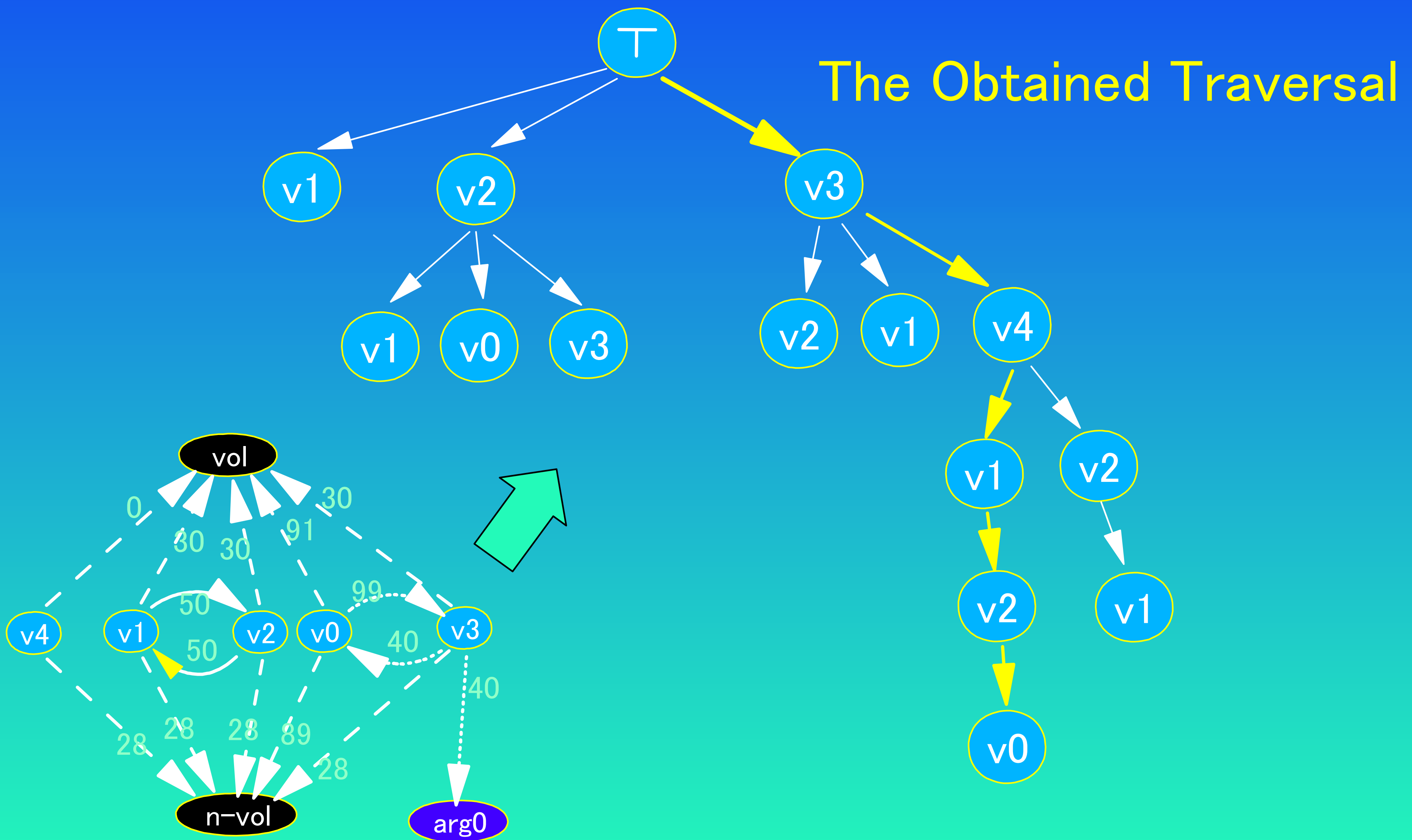


Register Preference Graph

Register Selection Algorithm



Register Selection Algorithm



Preference-Reflected Code

```
        save r3
        r3 = [r1]
L1:     r2 = [r3]
        r1 = [r3+4]
        r2 = r2 + r1
        r1 = r3
        save r2
        call
        restore r2
        r3 = r2+1
        if r3 != 0 goto L1
        restore r3
        ret
```

```
        save r3
        r1 = [r1]
L1:     r2,r3 = [r1]
;;      loads are paired
        r3 = r2 + r3
;;      no need for the copy
;;      no need for saving the value
        call
;;      no need for restoring the value
        r1 = r3 +1
        if r1 != 0 goto L1
        restore r3
        ret
```

Evaluation

Implement our coloring system on Java just-in-time compiler for IBM JDK 1.3 for Intel Itanium

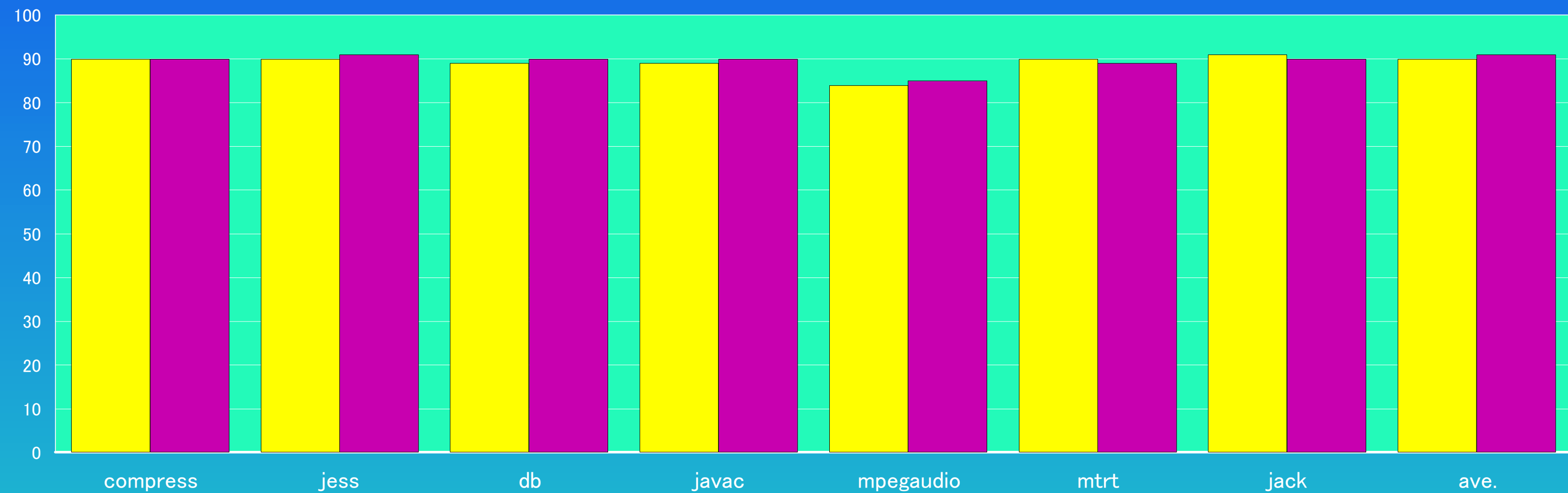
Comparison

- **Latest coalesing method**
- **Our algorithm only doing coalescing**
- **Our full-featured algorithm**

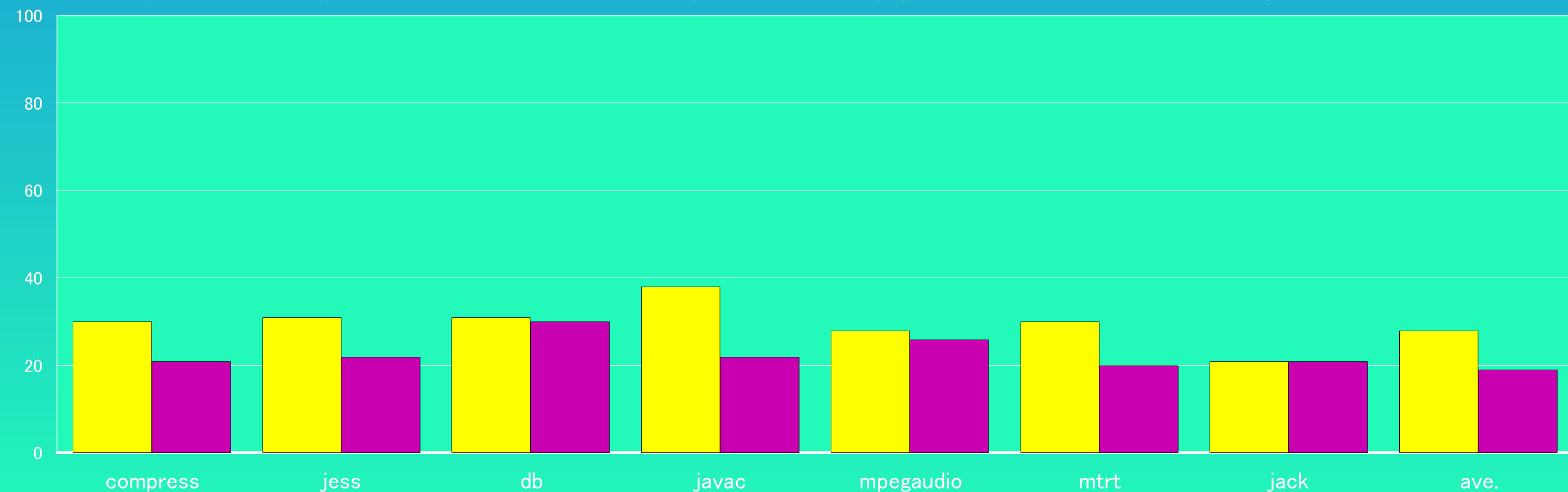
Coalescing Performance

■ optimistic ■ only coalescing (100% = Chaitin’s)

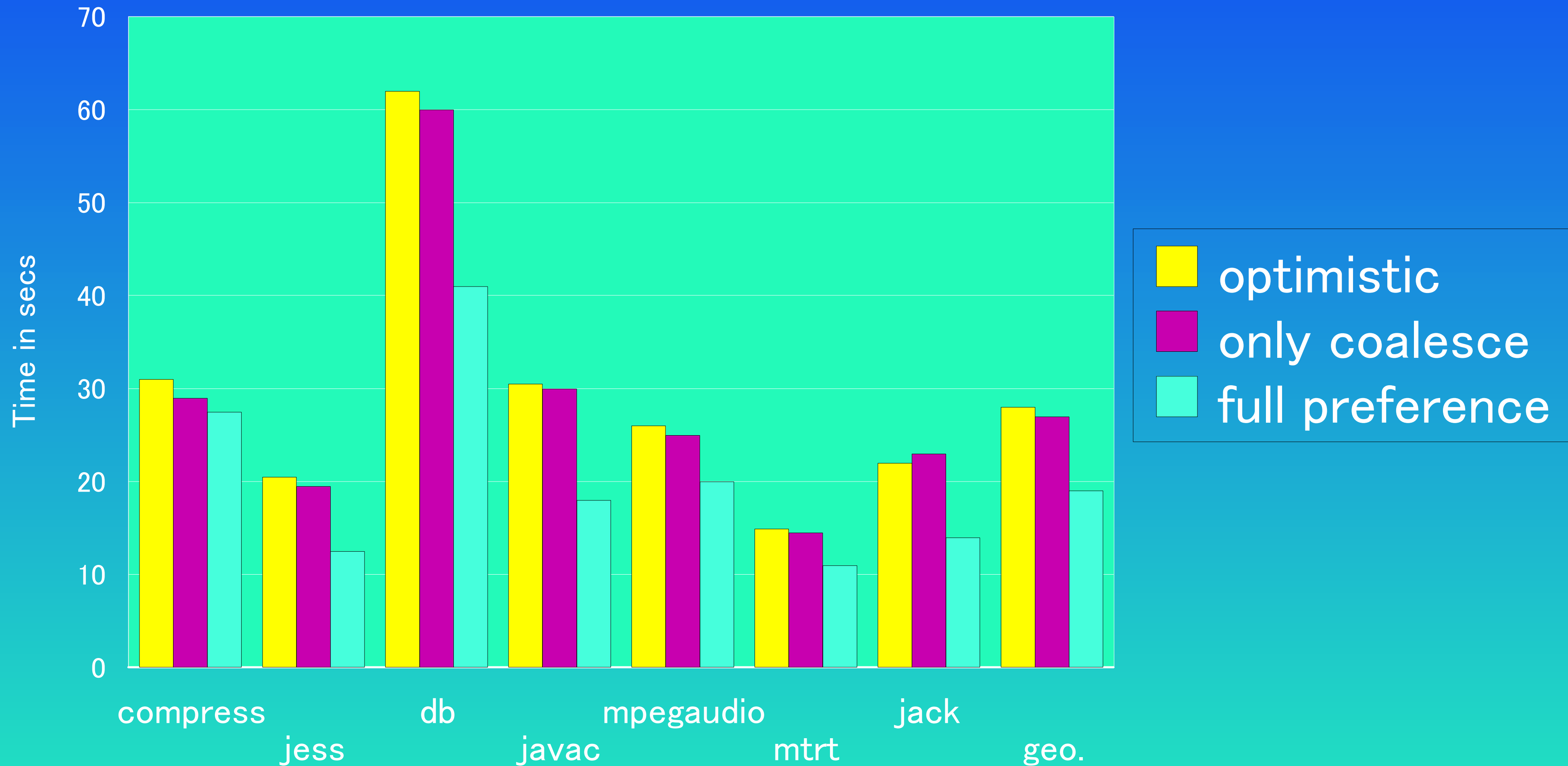
coalesce ratio



spill ratio



Runtime Performance



using 24 registers

Conclusions

- Coloring Precedence Graph gives more chances to use irregular registers.
- Register Preference Graph gives a uniformed metrics to evalute the registers
- New graph coloring system using both graphs shows good performance