Compiler Construction

Advanced graph coloring

Simone Campanoni
simone.campanoni@northwestern.edu

A coloring algorithm

Algorithm:

1. Repeatedly select a node and remove it from the graph, putting it on top of a stack

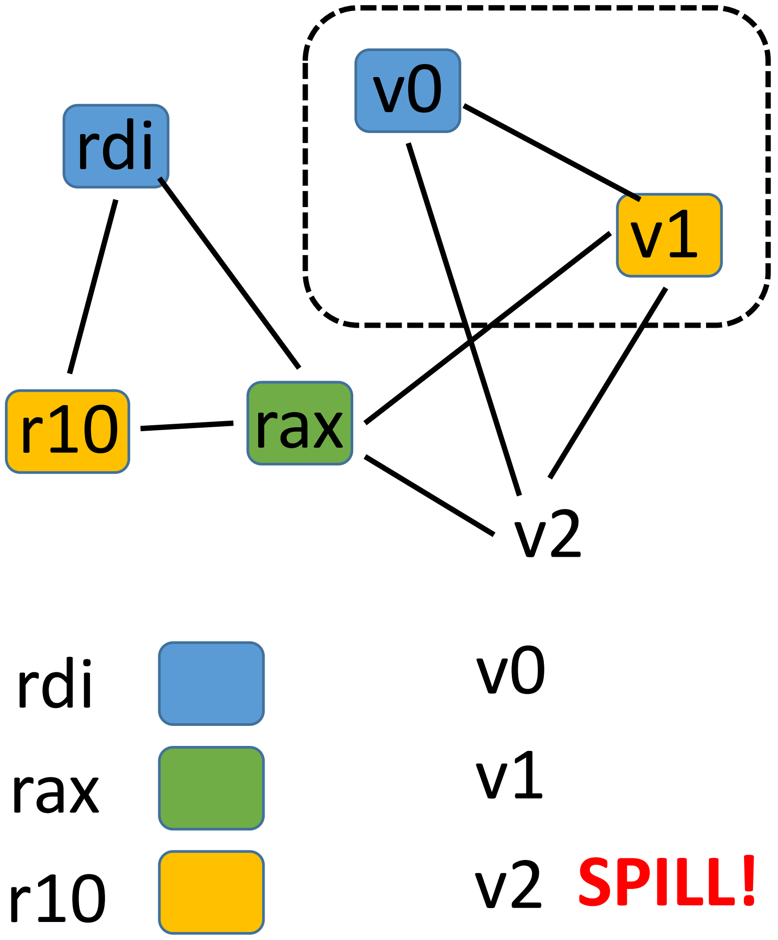
2. When the graph is empty, rebuild it
   - Select a color on each node as it comes back into the graph, making sure no adjacent nodes have the same color
   - If there are not enough colors, the algorithm fails
     - Spilling comes in here
     - Select the nodes (variables) you want to spill

Outline

- Coalescing and freezing
- Advanced register order
- Advanced spilling

Limitation of our basic approach

```
(@myF 1
  %v0 <- rdi
  %v1 <- %v0
  %v2 <- %v0
  rax <- %v0
  rax += %v1
  rax += %v2
  return
)
```

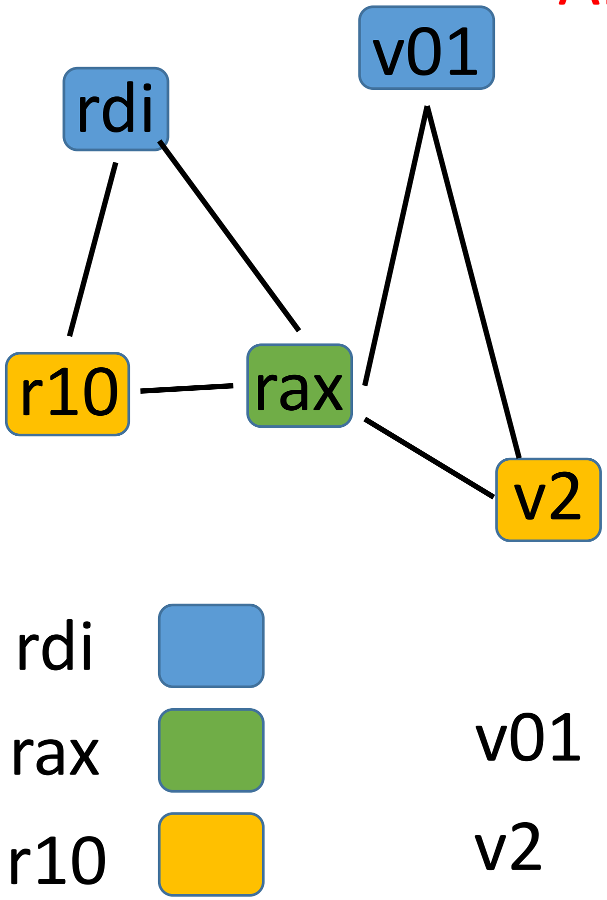



```
(@myF 1 0
  rax <- rdi
  rax += rdi
  rax += rdi
  return
)
```

What is the best L1 code?

Advanced heuristic: coalescing

```
(@myF 1
  %v0 <- rdi
  %v1 <- %v0
  %v2 <- %v0
  rax <- %v0
  rax += %v1
  rax += %v2
  return
)
```

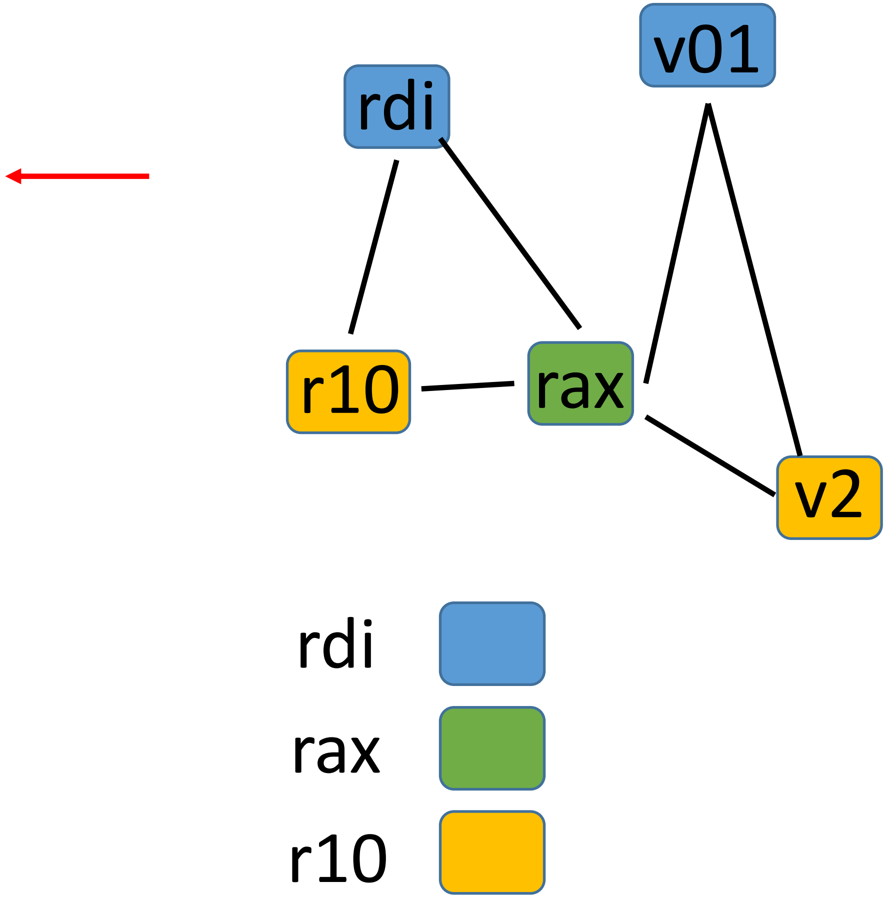


Are they useful?

```
(@myF 1 0
  rdi <- rdi
  rdi <- rdi
  r10 <- rdi
  rax <- rdi
  rax += rdi
  rax += r10
  return
)
```

Advanced heuristic: coalescing

```
(@myF 0
  %v0 <- rdi
  %v1 <- %v0
  %v2 <- %v0
  rax <- %v0
  rax += %v1
  rax += %v2
  return
)
```

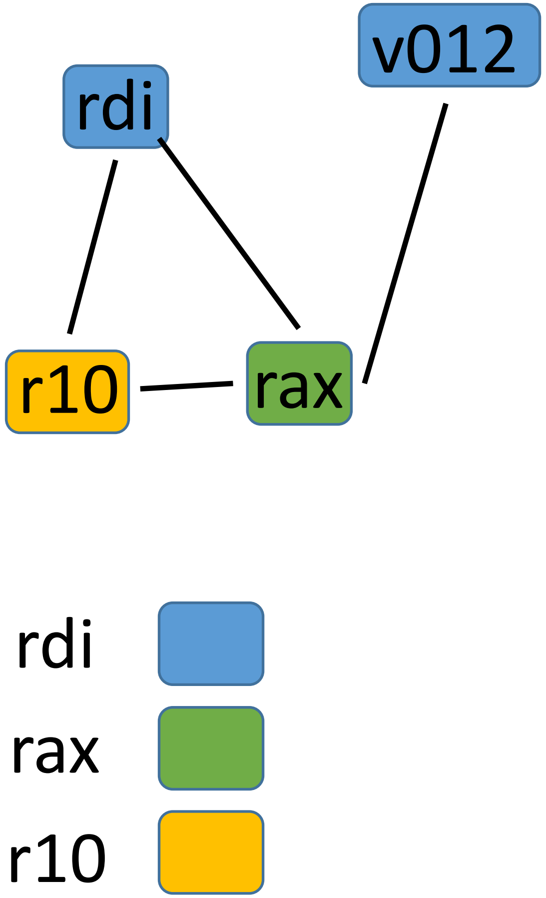



```
(@myF 1 0

  r10 <- rdi
  rax <- rdi
  rax += rdi
  rax += r10
  return
)
```

Advanced heuristic: coalescing

```
(@myF 0
  %v0 <- rdi
  %v1 <- %v0
  %v2 <- %v0
  rax <- %v0
  rax += %v1
  rax += %v2
  return
)
```



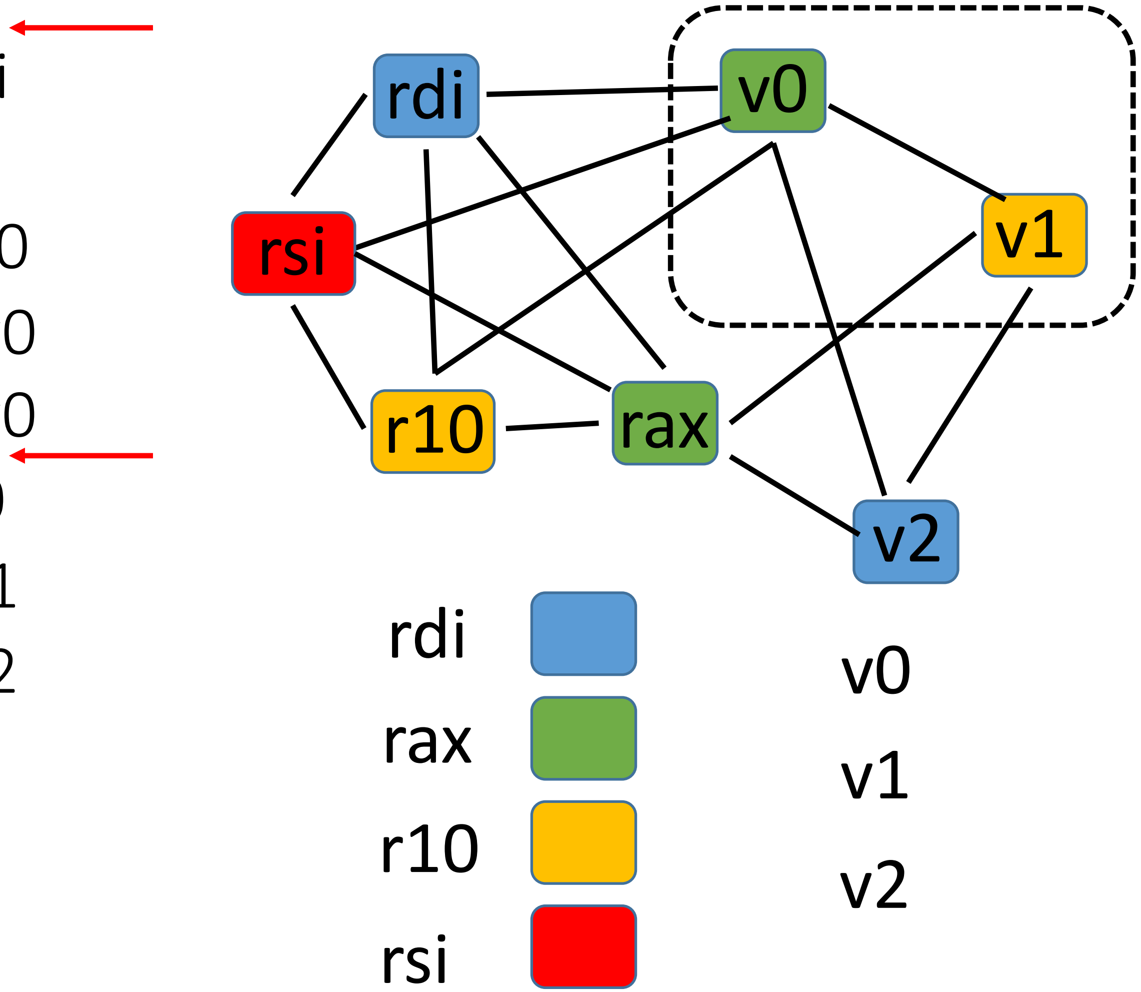

```
(@myF 1 0
  rax <- rdi
  rax += rdi
  rax += rdi
  return
)
```

Coalescing problem

- Coalescing can significantly increase the quality of the code
- Merging N nodes increases the degree of the resulting node
- This might generate a graph that requires more colors
  - More spills!

Coalescing: the potential problem

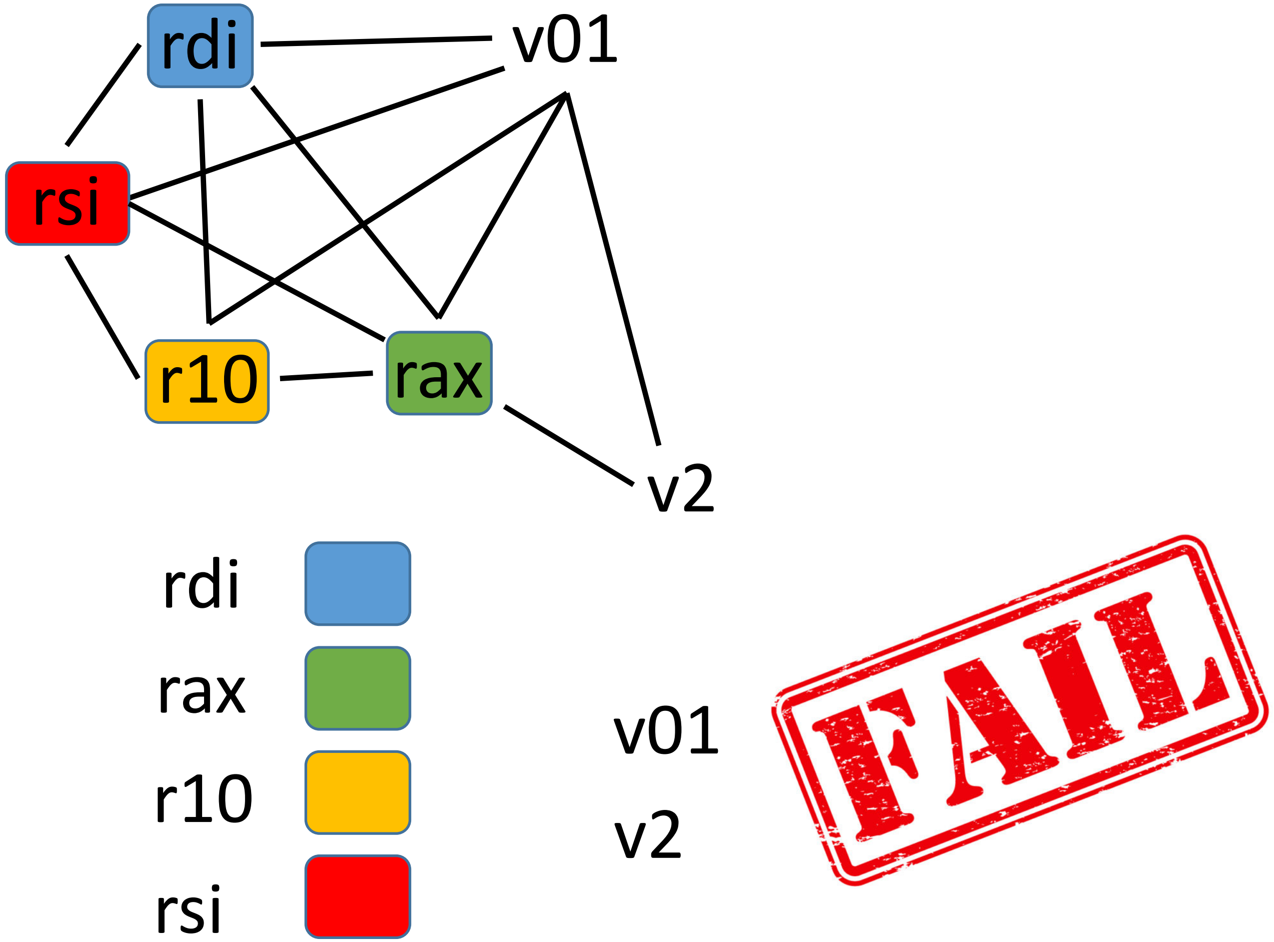
```
(@myF 3
  %v0 <- rdi
  %v0 += rdi
  %v0 += rsi
  %v0 += r10
  %v1 <- %v0
  %v2 <- %v0
  rax <- %v0
  rax += %v1
  rax += %v2
  return
)
```



- Graph coloring without coalescing succeeded!
- Let's try to do coalescing before graph coloring

Coalescing: the potential problem

```
(@myF 3
  %v0 <- rdi
  %v0 += rdi
  %v0 += rsi
  %v0 += r10
  %v1 <- %v0
  %v2 <- %v0
  rax <- %v0
  rax += %v1
  rax += %v2
  return
)
```



Coalescing problem

- Coalescing can significantly increase the quality of the code
- Merging N nodes increases the degree of the resulting node
- This might generate a graph that requires more colors
  - More spills!
- So when should we apply it?
- Two common conservative strategies:

1. Briggs
2. George

Briggs

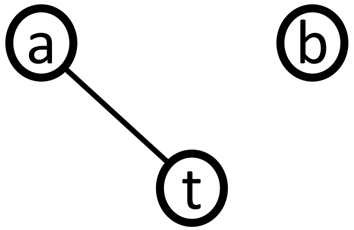
Nodes a and b can be coalesced if the resulting node ab will have fewer than K neighbors of degree >= K

- K = Number of general purpose registers
- This coalescing is guaranteed not to turn a K-colorable graph into a non-K-colorable graph

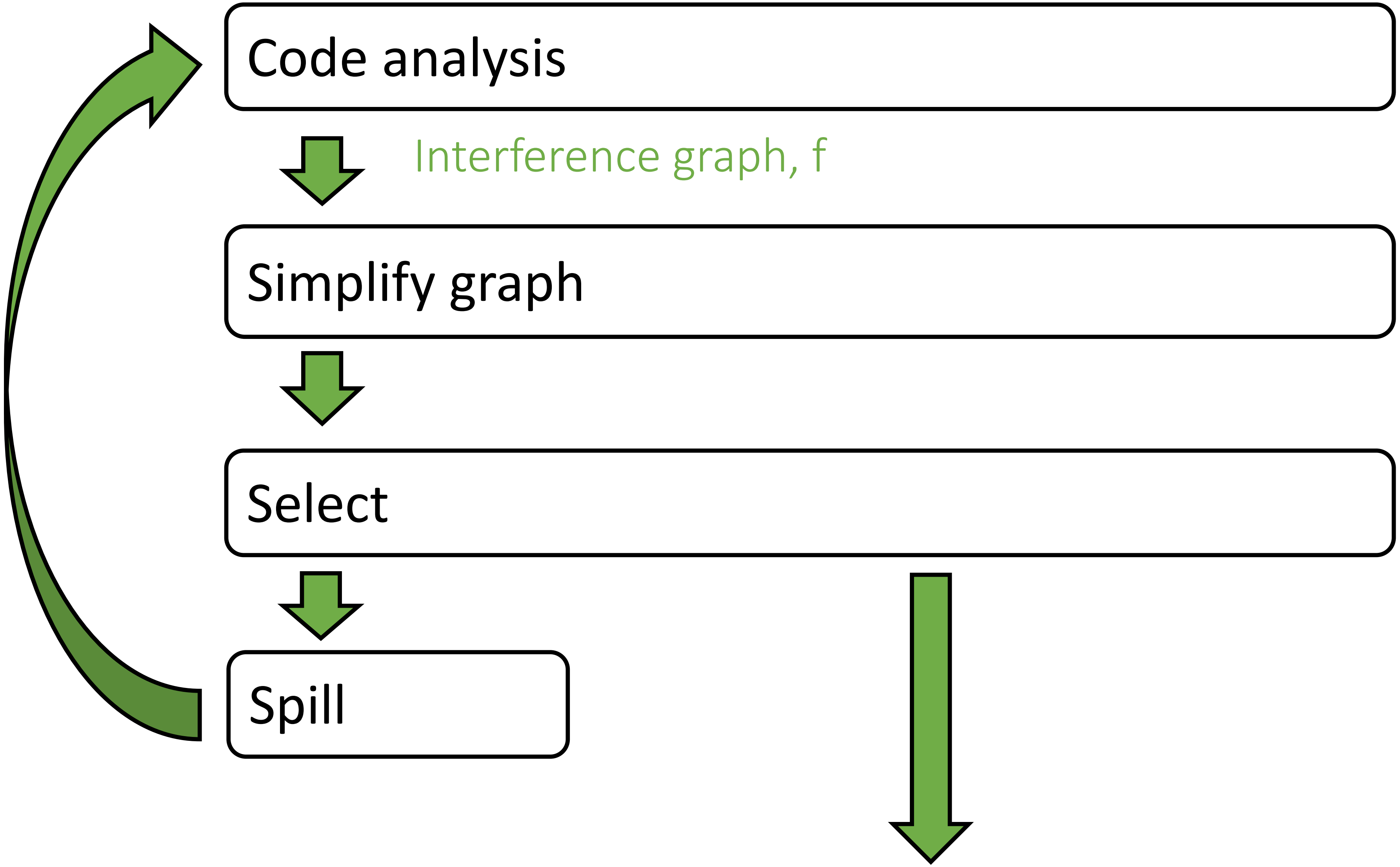
George

Nodes a and b can be coalesced if
for every adjacent node t of a, either

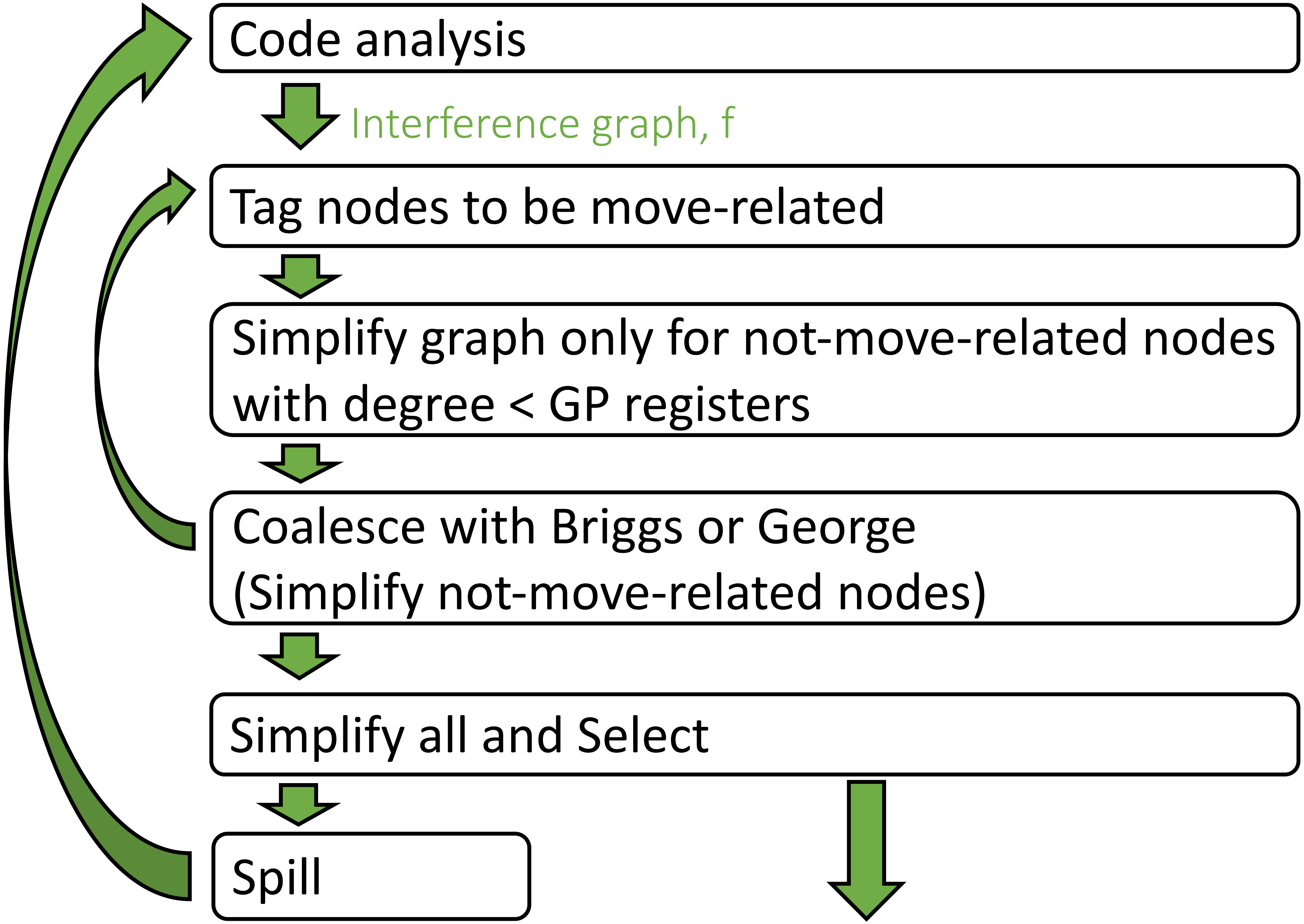
- (t, b) already exists or
- Degree(t) < K



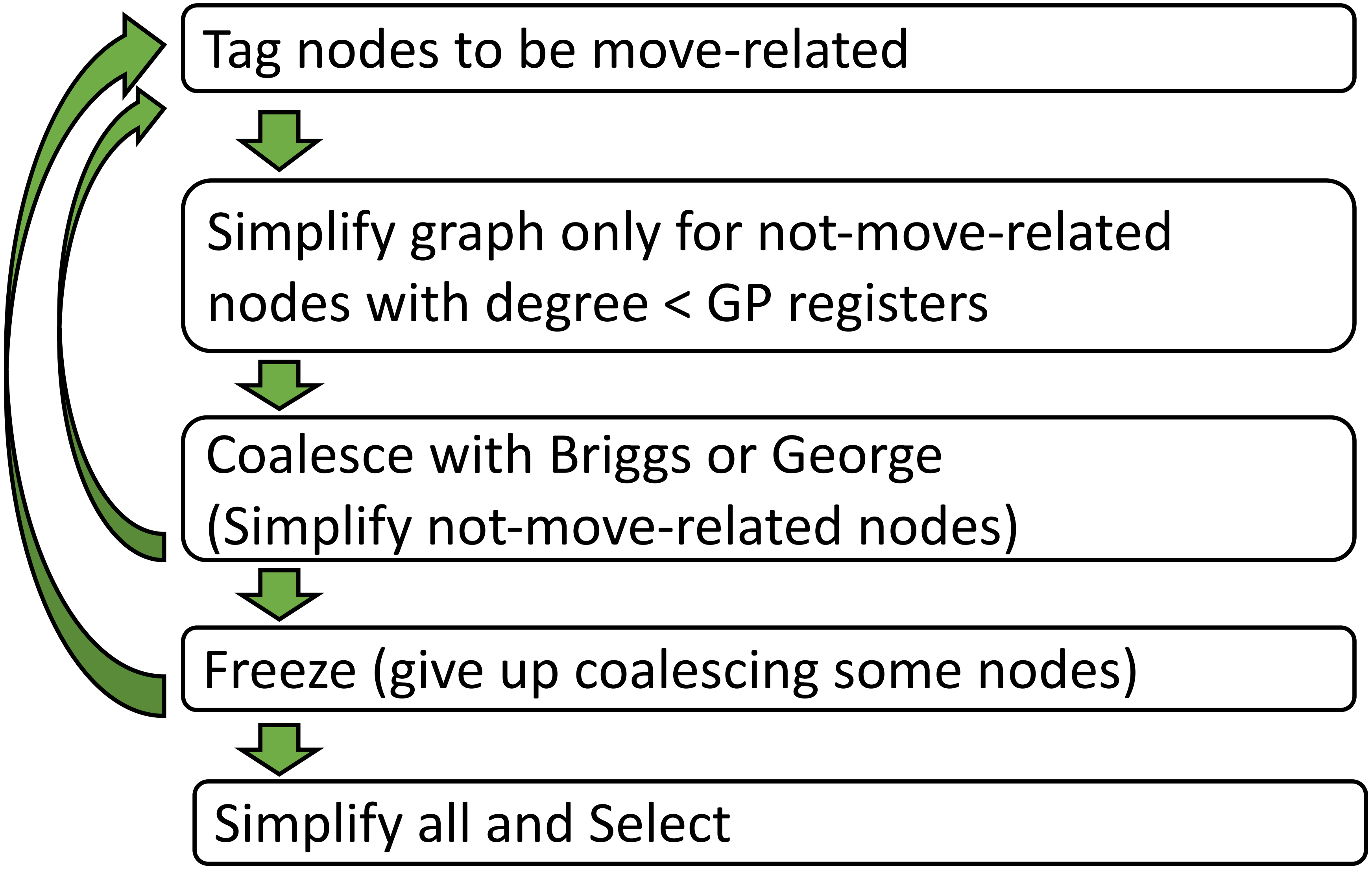
Graph coloring without coalescing



Graph coloring with coalescing



Advanced heuristic: freeze move nodes



Outline

- Coalescing and freezing
- Advanced register order
- Advanced spilling

Example

```
(@myF
  1
  %myV1 <- 1
  %myV2 <- 1
  %myV3 <- 1
  %myV4 <- 1
  %myV5 <- 1
  %myV6 <- 1
  %myV7 <- 1
  mem rdi 0  <- %myV1
  mem rdi 8  <- %myV2
  mem rdi 16 <- %myV3
  mem rdi 24 <- %myV4
  mem rdi 32 <- %myV5
  mem rdi 40 <- %myV6
  mem rdi 48 <- %myV7
  return
)
```

Registers

Arguments	Result	Caller save	Callee save
rdi	rax	r10	r12
rsi		r11	r13
rdx		r8	r14
rcx		r9	r15
r8		rax	rbp
r9		rcx	rbx
		rdi	
		rdx	
		rsi	

Example

```
(@myF
  1
  %myV1 <- 1
  %myV2 <- 1
  %myV3 <- 1
  %myV4 <- 1
  %myV5 <- 1
  %myV6 <- 1
  %myV7 <- 1
  mem rdi 0  <- %myV1
  mem rdi 8  <- %myV2
  mem rdi 16 <- %myV3
  mem rdi 24 <- %myV4
  mem rdi 32 <- %myV5
  mem rdi 40 <- %myV6
  mem rdi 48 <- %myV7
  return
)
```

Caller save
r10
r11
r8
r9
rcx
rdi
rdx
rsi
rax

Will we color this graph without spilling?

Yes

Example 2

```
(@myF
  1
  %myV1 <- 1
  %myV2 <- 1
  %myV3 <- 1
  %myV4 <- 1
  %myV5 <- 1
  %myV6 <- 1
  %myV7 <- 1
  ... // computation that uses %myV* variables
  mem rsp -8 <- :ret
  call :myF2 0
  :ret
  mem rdi 0  <- %myV1
  mem rdi 8  <- %myV2
  mem rdi 16 <- %myV3
  mem rdi 24 <- %myV4
  mem rdi 32 <- %myV5
  mem rdi 40 <- %myV6
  mem rdi 48 <- %myV7
  return
)
```

- Will we color this graph without spilling?
- Which variables will we spill?
- Can we do better?
- What about using callee save registers?
  - Yes, but we need to save them at the beginning of the function and restore them before every return

Example: assuming 2 caller save registers

Approach: advanced graph coloring

```
(@myF
  1

rsi  %myV1 <- 1
r12  %myV2 <- 1

     ... // computation that uses myV* variables

  mem rdi 0  <- %myV1
  mem rdi 8  <- %myV2

  return
)
```

Example: assuming 2 caller save registers

Approach: advanced graph coloring

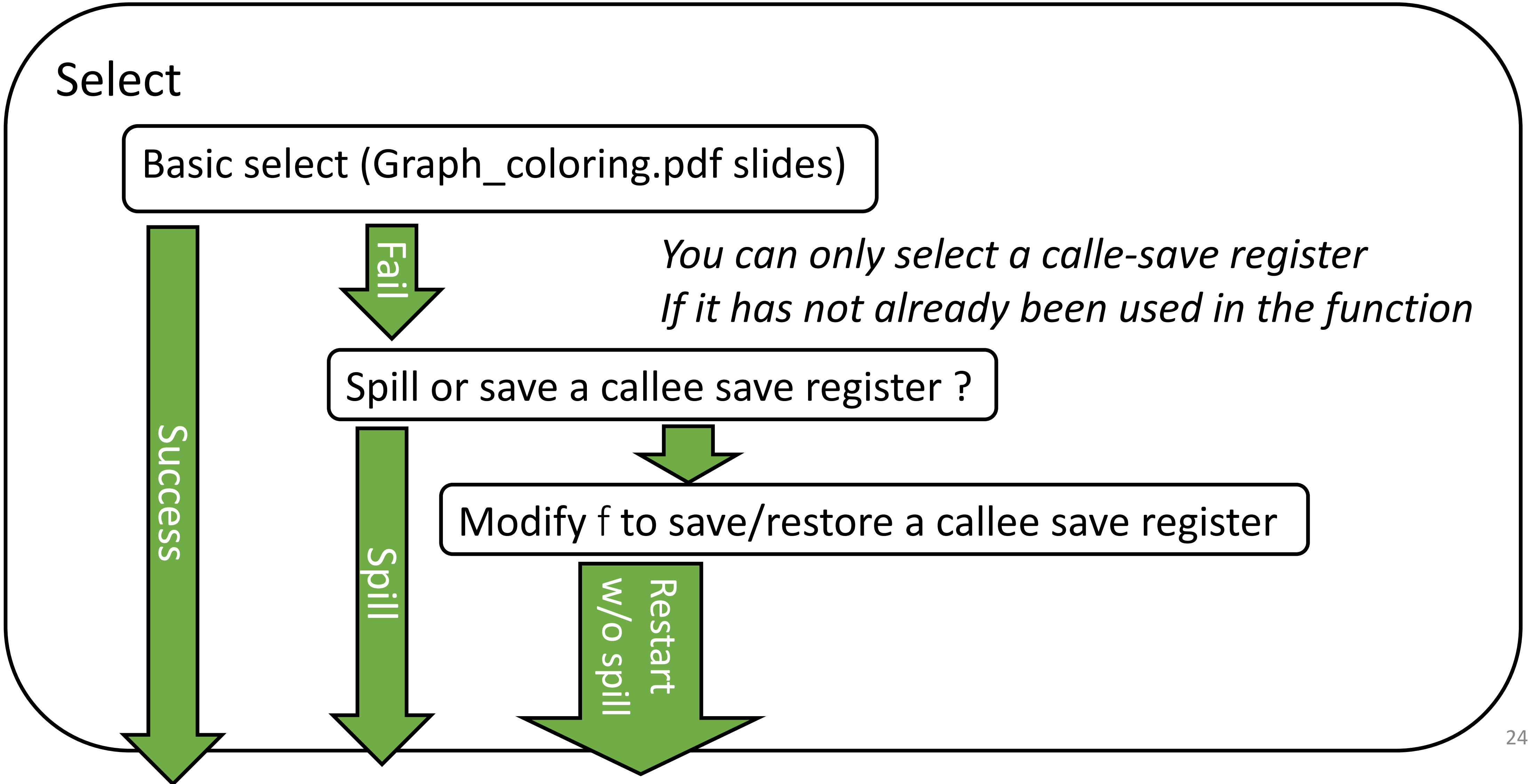
```
(@myF
    1 1
    mem rsp 0 <- r12
rsi %myV1 <- 1
r12 %myV2 <- 1

    … // computation that uses myV* variables

    mem rdi 0  <- %myV1
    mem rdi 8  <- %myV2

    r12 <- mem rsp 0
    return
)
```

Select


Select
Basic select (Graph_coloring.pdf slides)
Success
Fail
You can only select a calle-save register
If it has not already been used in the function
Spill or save a callee save register ?
Spill
Modify f to save/restore a callee save register
Restart
w/o spill

Advanced heuristics: register order

- Until now:
  - Caller-save registers are used first
  - Callee-save registers are used only at the end

- Change the order of registers depending on the code in f
  - E.g., a lot of calls => prefer callee save registers
  - E.g., a few calls => prefer caller save registers

- This heuristic requires extra code analysis to count #calls

Advanced heuristic: node selection

- Idea: variables used the most at run-time should be in registers
- Approach: give priority to nodes (variables) used in loops
- This heuristic requires a code analysis
  usually found in middle-ends: loop identification

Outline

- Coalescing and freezing
- Advanced register order
- Advanced spilling

Advanced heuristic: spilling

- Spill a subset of variables at every iteration
  - E.g., 1 at a time

- After having spilled variables
  - Run the register allocation algorithm for spilled variables
  - This will save space in the stack (lower memory pressure)
  - 1 color = 1 stack location

Always have faith in your ability

Success will come your way eventually

Best of luck!