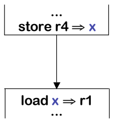
Register Allocation

Global Register Allocation
Webs and Graph Coloring
Node Splitting and Other Transformations

What a Smart Allocator Needs to Do

- Determine ranges for each variable can benefit from using a register (webs)
- Determine which of these ranges overlap (interference)
- Find the benefit of keeping each web in a register (spill cost)
- Decide which webs gets a register (allocation)
- Split webs if needed (spilling and splitting)
- Assign hard registers to webs (assignment)
- Generate code including spills (code gen)

Global Register Allocation

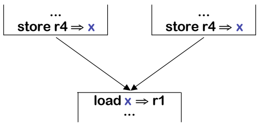


This is an assignment problem, not an allocation problem !

What's harder across multiple blocks?

- Could replace a load with a move
- Good assignment would obviate the move
- Must build a control-flow graph to understand inter-block flow
- Can spend an inordinate amount of time adjusting the allocation

Global Register Allocation



What if one block has x in a register, but the other does not?

A more complex scenario

- Block with multiple predecessors in the control-flow graph
- Must get the "right" values in the "right" registers in each predecessor
- In a loop, a block can be its own predecessors

This adds tremendous complications

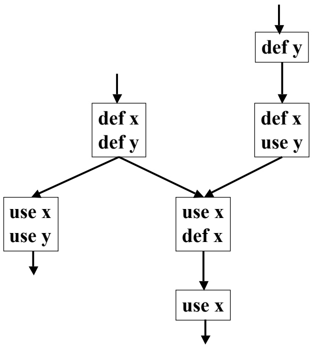
Outline

- What is Register allocation and Its Importance
- Simple Register Allocators
- Webs
- Interference Graphs
- Graph Coloring
- Splitting
- More Optimizations

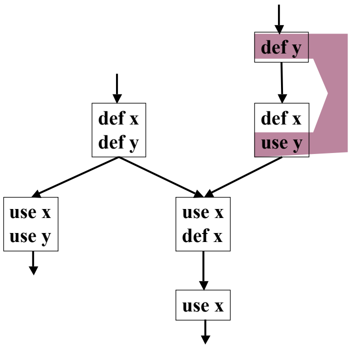
Webs

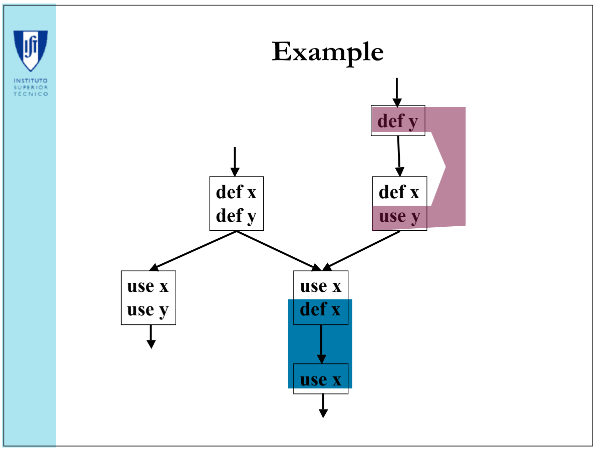
- What needs to Gets Memorized is the Value
- Divide Accesses to a Variable into Multiple Webs
  - All definitions that reaches a use are in the same web
  - All uses that use the value defined are in the same web
  - Divide the Variable into Live Ranges
- Implementation: use DU chains
  - A du-chain connects a definition to all uses reached by the definition
  - A web combines du-chains containing a common use

Example

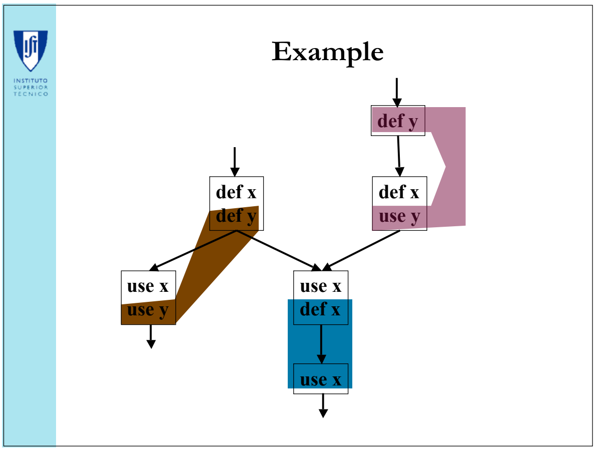


Example

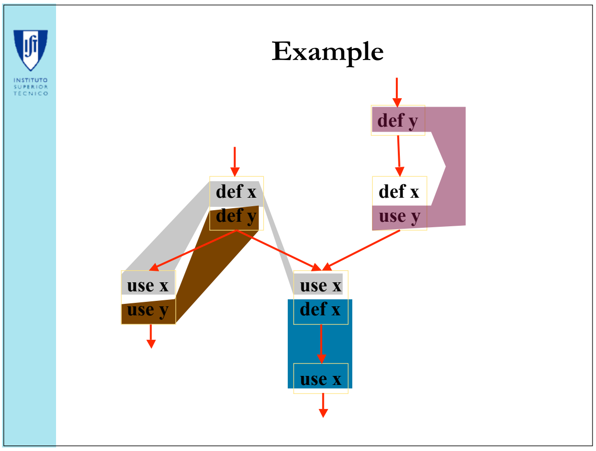




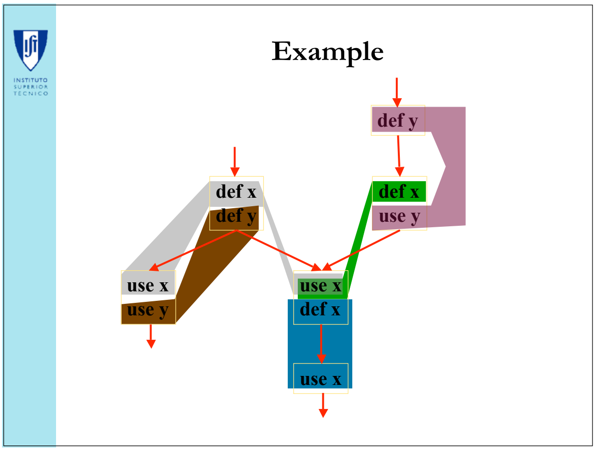
Example



Example

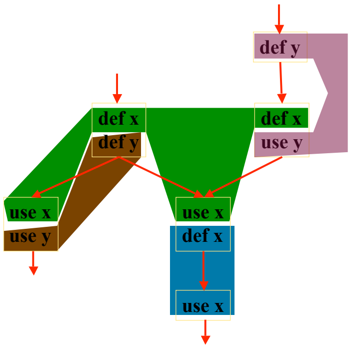


Example

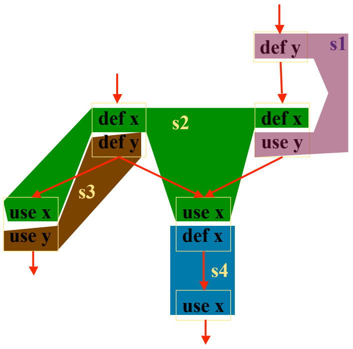


Example

Example



Example

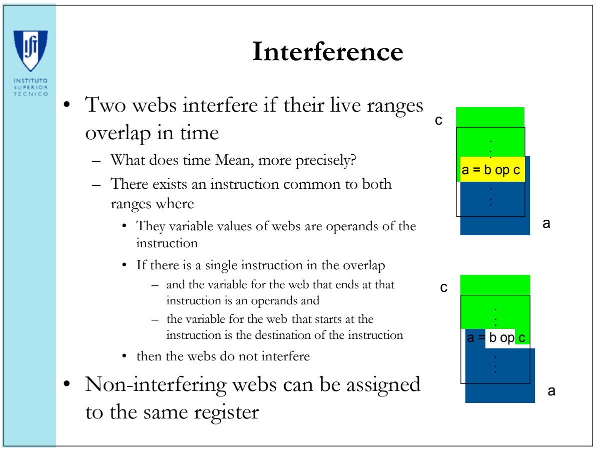


Webs (continued)

- In two Webs of the same Variable:
  - No use in one web will ever use a value defined by the other web
  - Thus, no value need to be carried between webs
  - Each web can be treated independently as values are independent
- Web is used as the unit of Register Allocation
  - If a web is allocated to a register, all the uses and definitions within that web don't need to load and store from memory
  - Solves the issue of cross Basic Block register assignment
  - Different webs may be assigned to different registers or one to register and one to memory

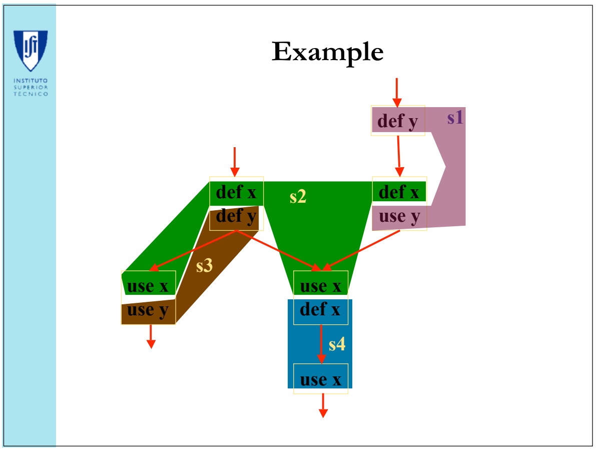
Outline

- What is Register Allocation
- A Simple Register Allocator
- Webs
- Interference Graphs
- Graph Coloring
- Splitting
- More Optimizations

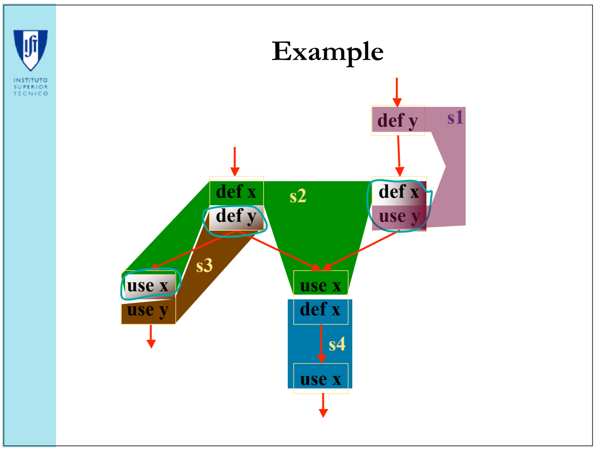


Interference

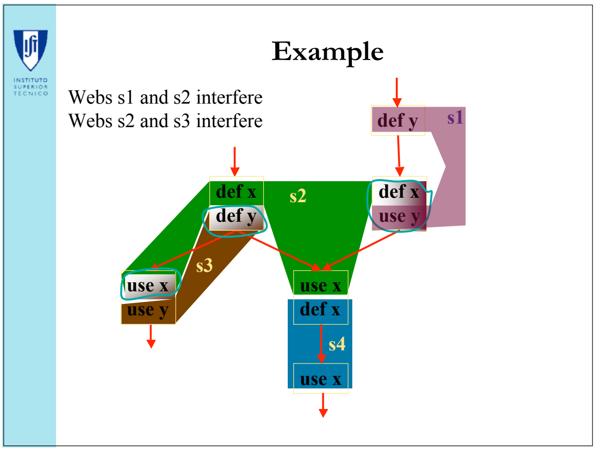
- Two webs interfere if their live ranges overlap in time
  - What does time Mean, more precisely?
  - There exists an instruction common to both ranges where
    - They variable values of webs are operands of the instruction
    - If there is a single instruction in the overlap
      - and the variable for the web that ends at that instruction is an operands and
      - the variable for the web that starts at the instruction is the destination of the instruction
    - then the webs do not interfere
- Non-interfering webs can be assigned to the same register



Example

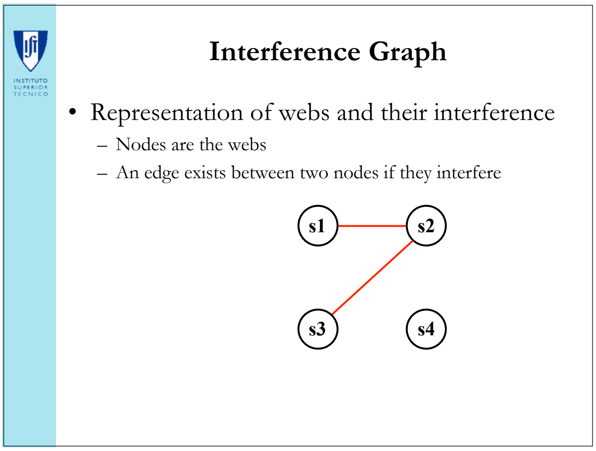


Example



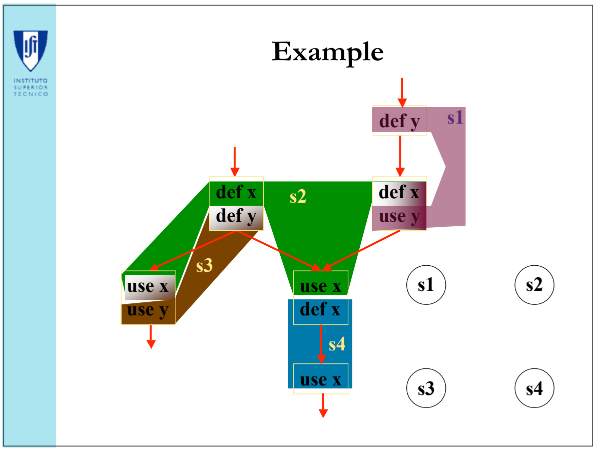
Example

Webs s1 and s2 interfere
Webs s2 and s3 interfere

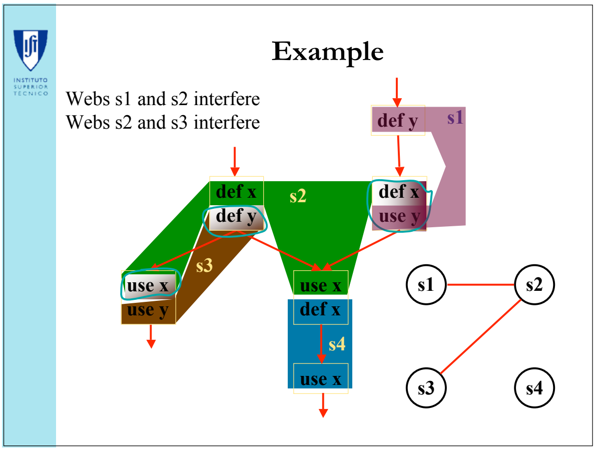


Interference Graph

- Representation of webs and their interference
  - Nodes are the webs
  - An edge exists between two nodes if they interfere

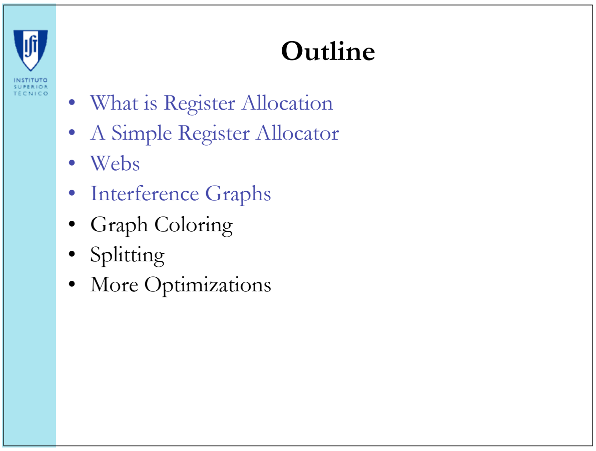


Example



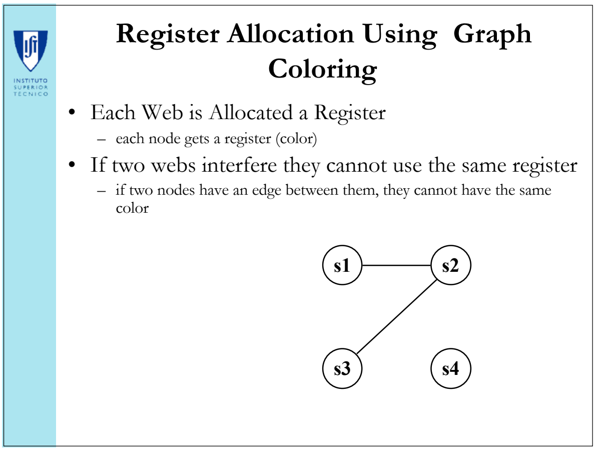
Example

Webs s1 and s2 interfere
Webs s2 and s3 interfere



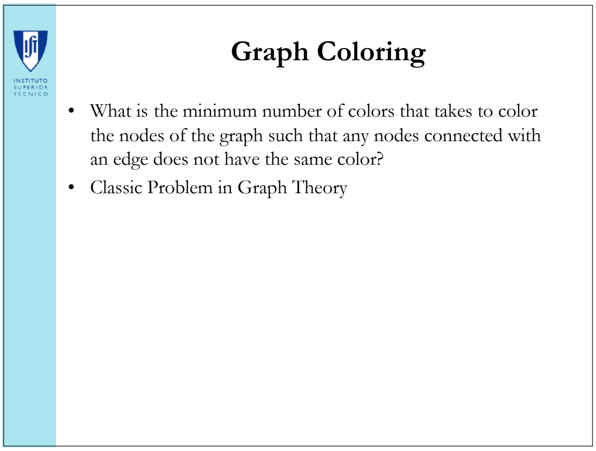
Outline

- What is Register Allocation
- A Simple Register Allocator
- Webs
- Interference Graphs
- Graph Coloring
- Splitting
- More Optimizations



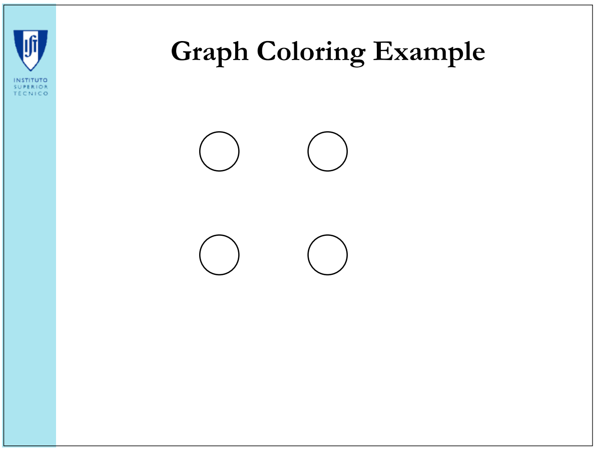
Register Allocation Using Graph Coloring

- Each Web is Allocated a Register
  - each node gets a register (color)
- If two webs interfere they cannot use the same register
  - if two nodes have an edge between them, they cannot have the same color

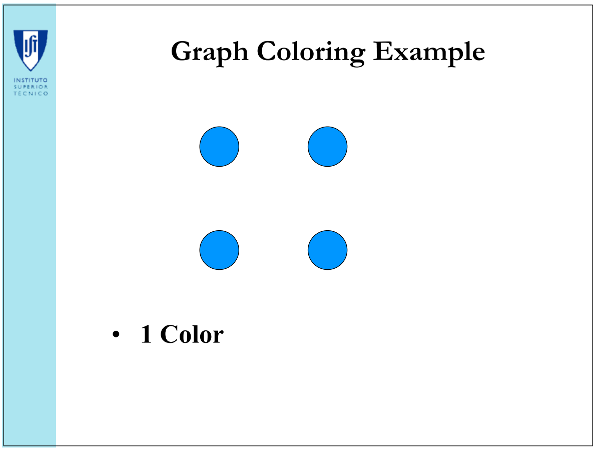


Graph Coloring

- What is the minimum number of colors that takes to color the nodes of the graph such that any nodes connected with an edge does not have the same color?
- Classic Problem in Graph Theory

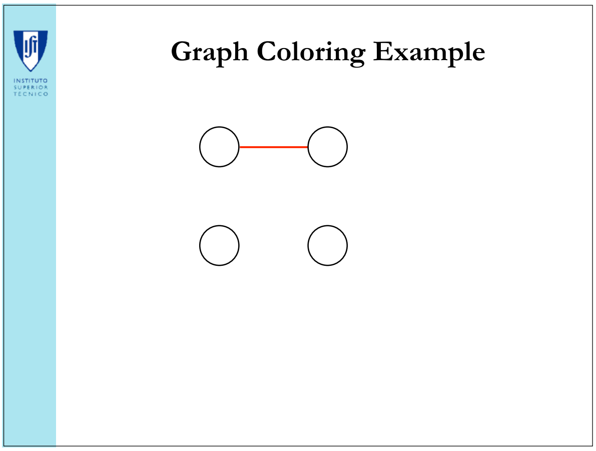


Graph Coloring Example

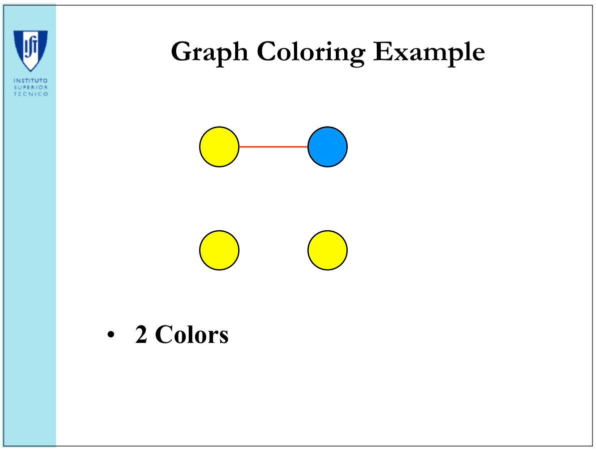


Graph Coloring Example

- **1 Color**

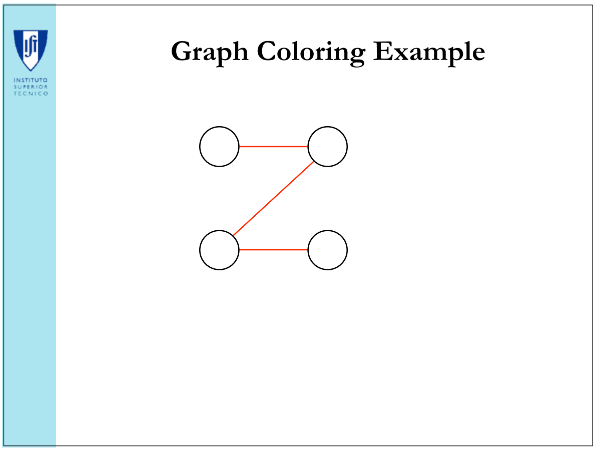


Graph Coloring Example

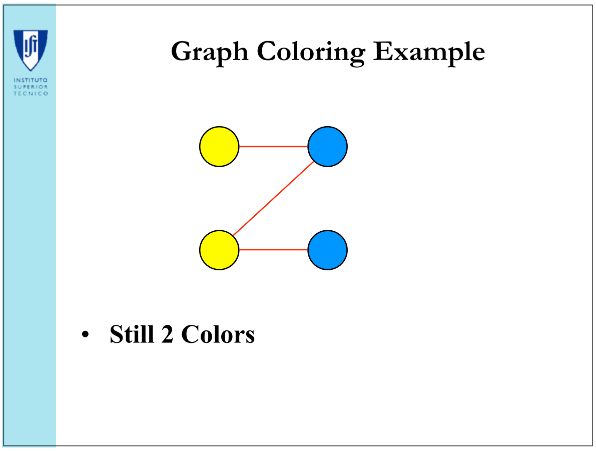


Graph Coloring Example

- **2 Colors**



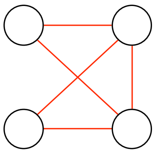
Graph Coloring Example



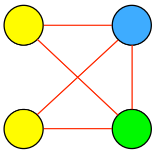
Graph Coloring Example

- **Still 2 Colors**

Graph Coloring Example



Graph Coloring Example



- **3 Colors**

Heuristics for Register Coloring

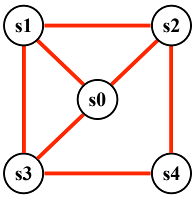
- Coloring a graph with N colors
- If degree < N (degree of a node = # of edges)
  - Node can always be colored
  - After coloring the rest of the nodes, you'll have at least one color left to color the current node
- If degree >= N
  - still may be colorable with N colors
  - exact solution is NP complete

Heuristics for Register Coloring

- Remove nodes that have degree < N
  - push the removed nodes onto a stack
- If all the nodes have degree >= N
  - Find a node to spill (no color for that node)
  - Remove that node
- When empty, start the coloring step
  - pop a node from stack back
  - Assign it a color that is different from its connected nodes (since degree < N, a color should exist)

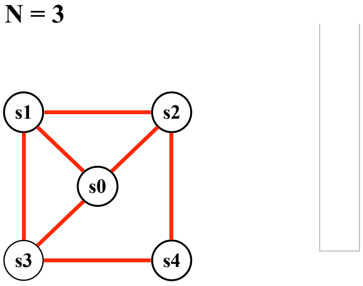
Coloring Example

N = 3



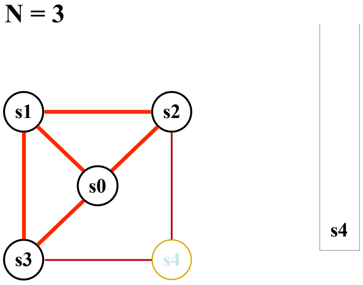
Coloring Example

N = 3



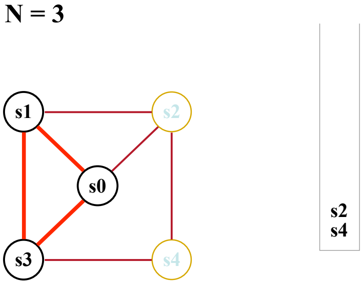
Coloring Example

N = 3



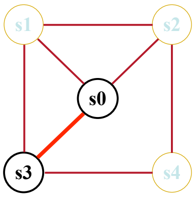
Coloring Example

N = 3



Coloring Example

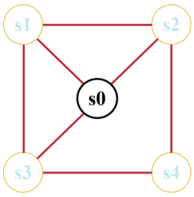
N = 3



s1
s2
s4

Coloring Example

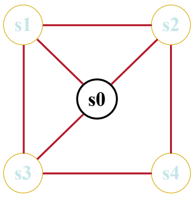
N = 3



s3
s1
s2
s4

Coloring Example

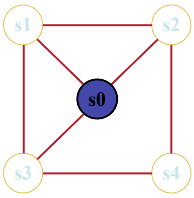
N = 3 🟦 🟩 🟫



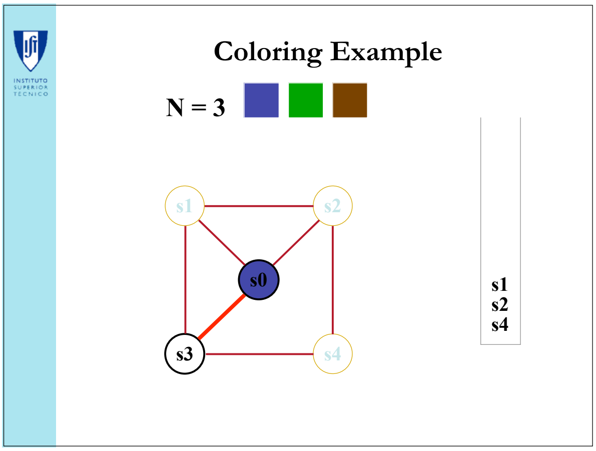
s3
s1
s2
s4

Coloring Example

N = 3 🟦 🟩 🟫



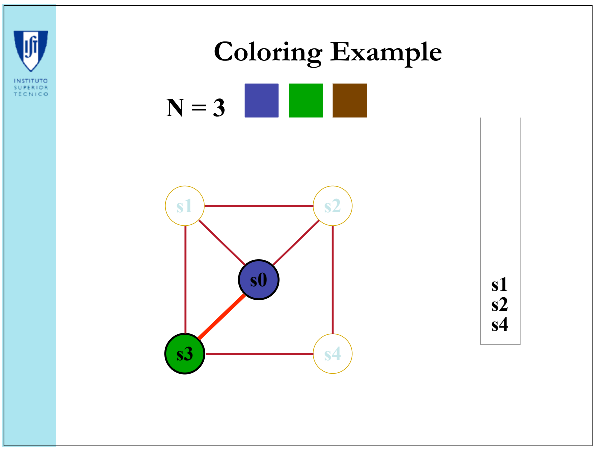
s3
s1
s2
s4



Coloring Example

N = 3 🟦 🟩 🟫

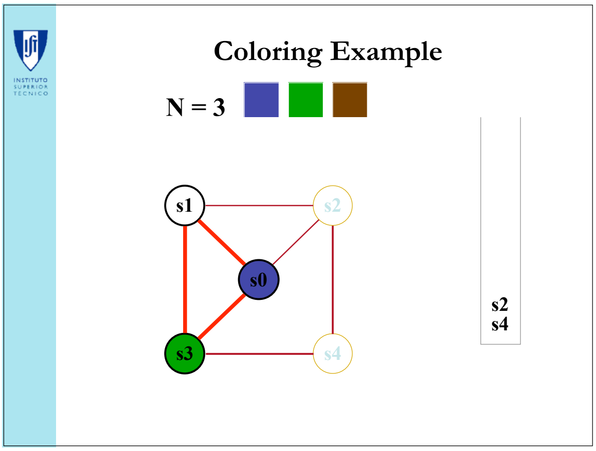
s1
s2
s4



Coloring Example

N = 3 🟦 🟩 🟫

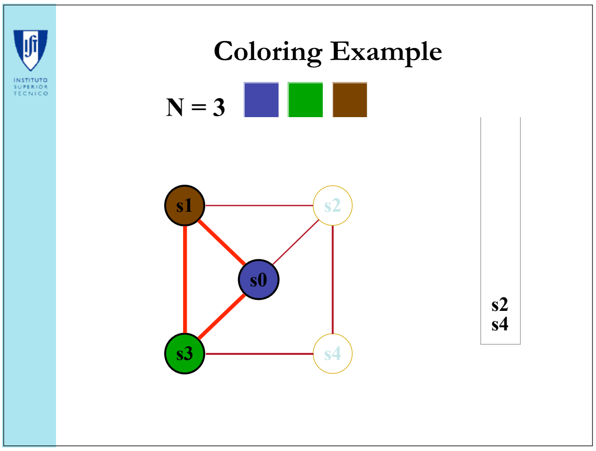
s1
s2
s4



Coloring Example

N = 3 🟦 🟩 🟫

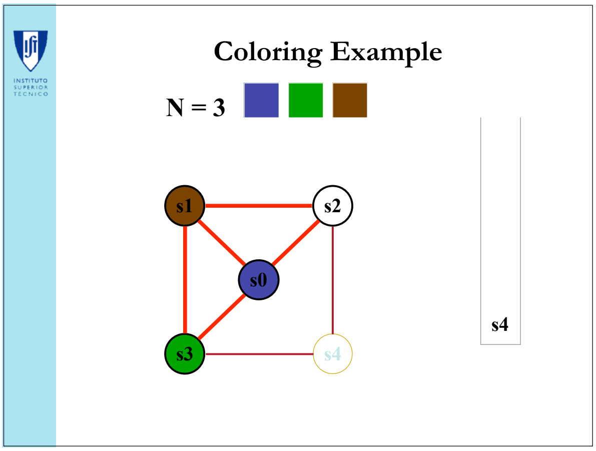
s2
s4



Coloring Example

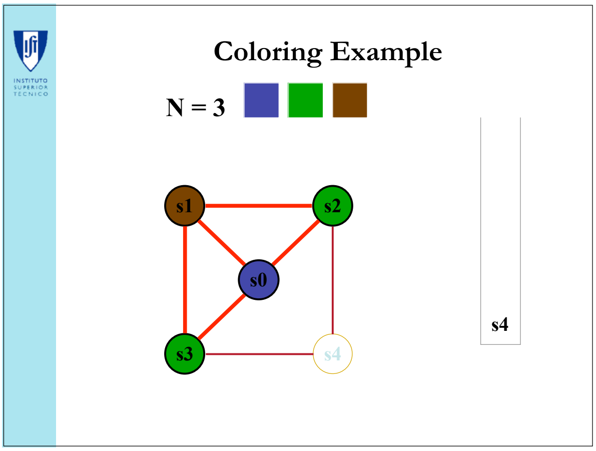
N = 3 🟦 🟩 🟫

s2
s4



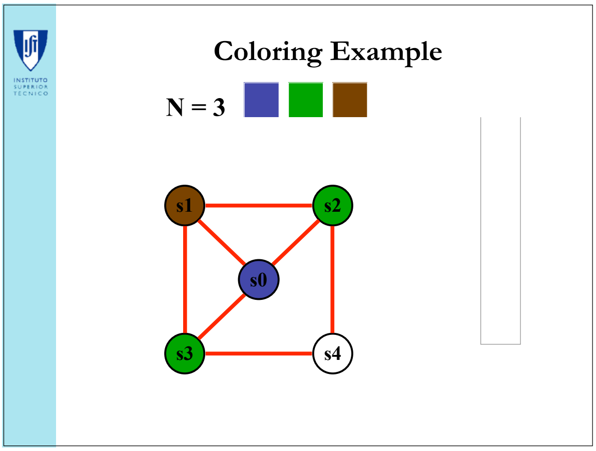
Coloring Example

N = 3 🟦 🟩 🟫



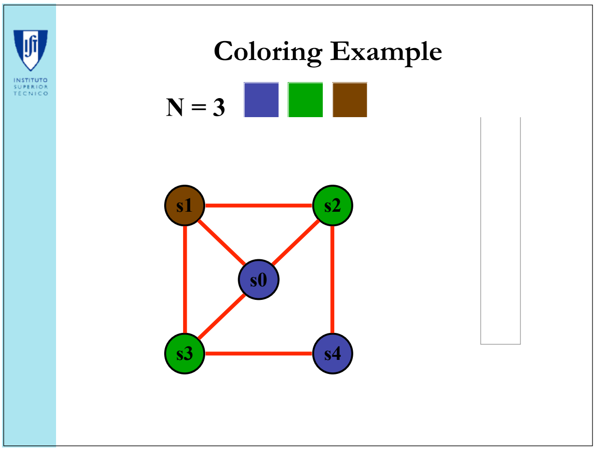
Coloring Example

N = 3 🟦 🟩 🟫



Coloring Example

N = 3 🟦 🟩 🟫

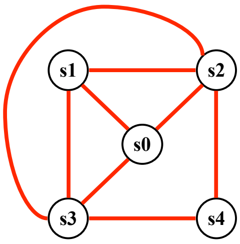


Coloring Example

N = 3 🟦 🟩 🟫

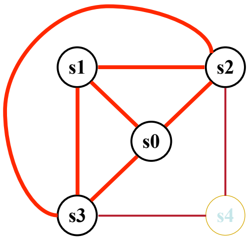
Another Coloring Example

N = 3



Another Coloring Example

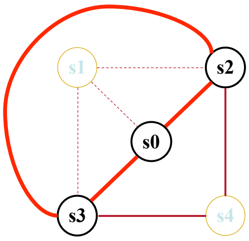
N = 3



s4

Another Coloring Example

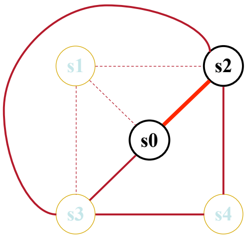
N = 3



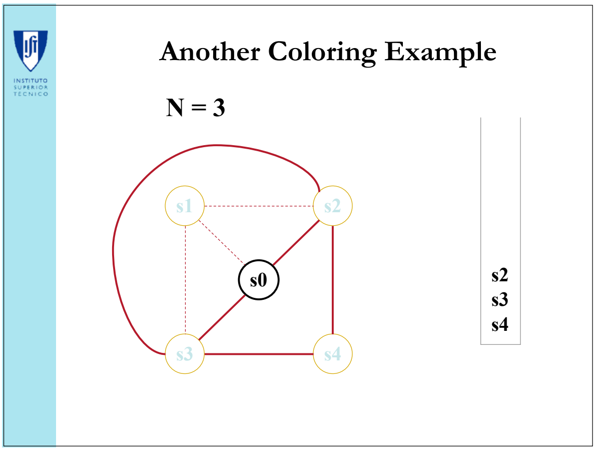
s4

Another Coloring Example

N = 3



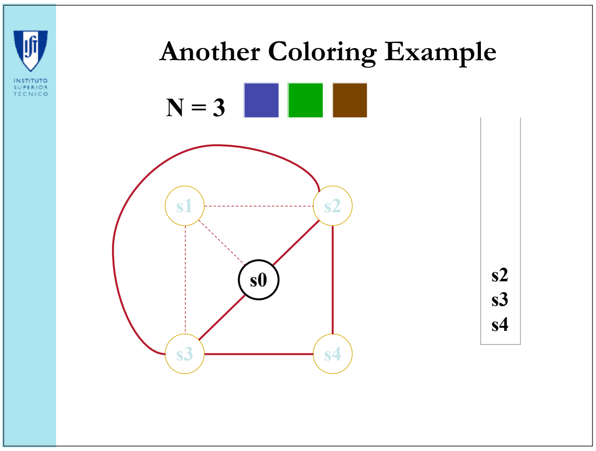
s3
s4



Another Coloring Example

N = 3

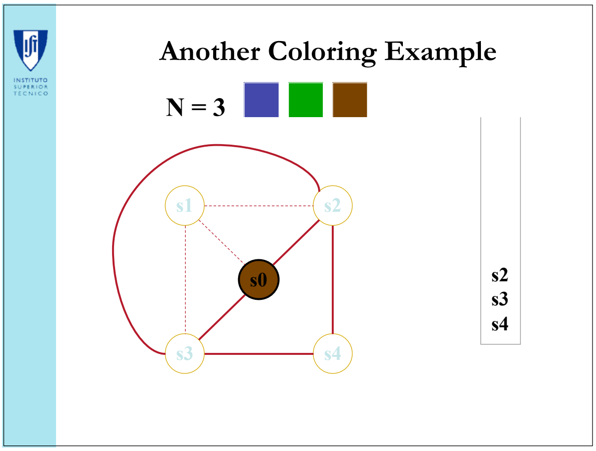
s2
s3
s4



Another Coloring Example

N = 3 🟦 🟩 🟫

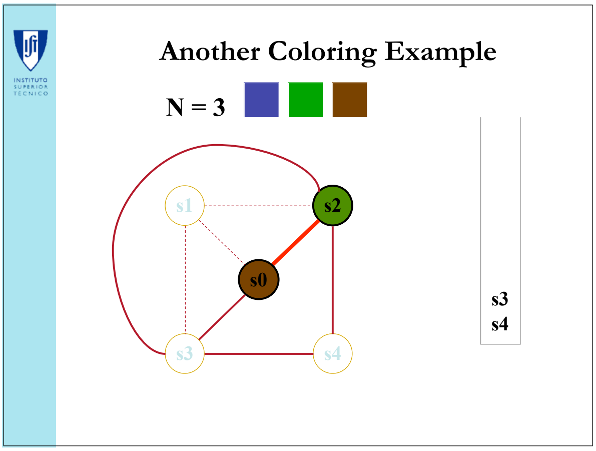
s2
s3
s4



Another Coloring Example

N = 3 🟦 🟩 🟫

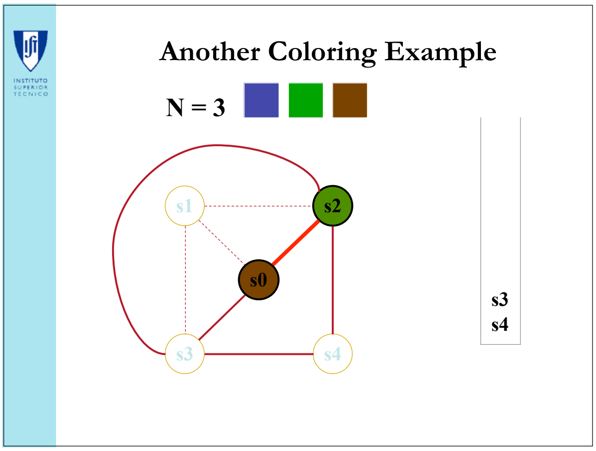
s2
s3
s4



Another Coloring Example

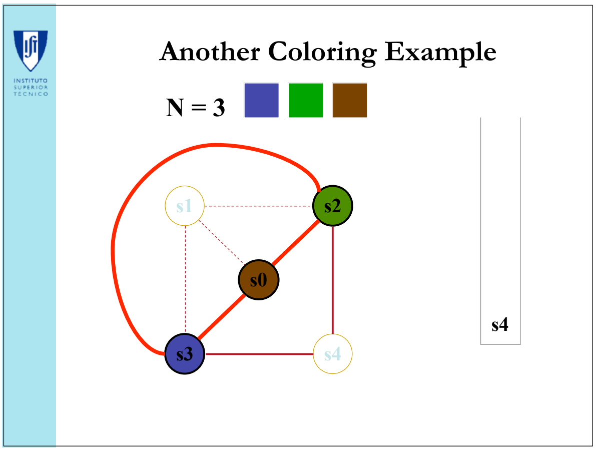
N = 3 🟦 🟩 🟫

s3
s4



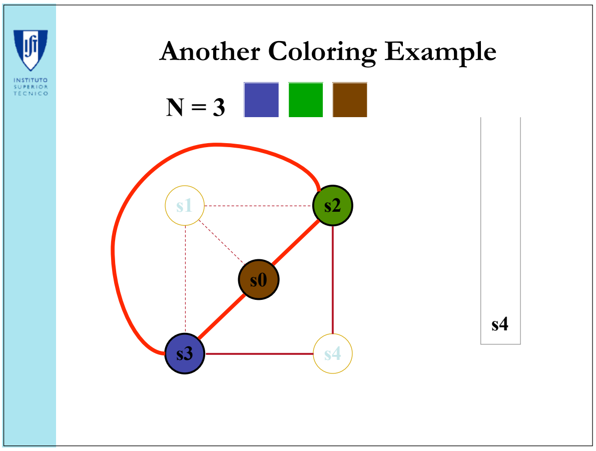
Another Coloring Example

N = 3 🟦 🟩 🟫



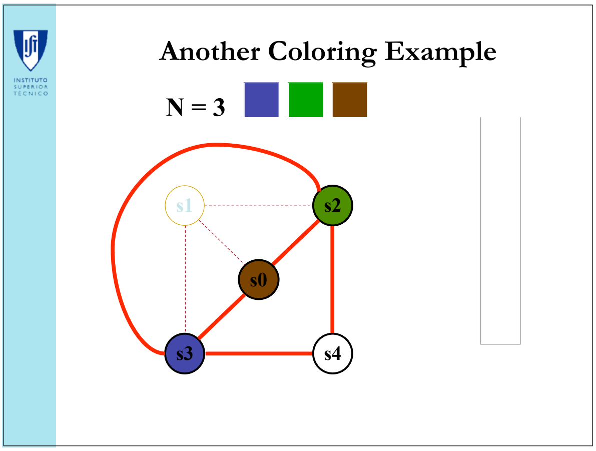
Another Coloring Example

N = 3 🟦 🟩 🟫



Another Coloring Example

N = 3 🟦 🟩 🟫

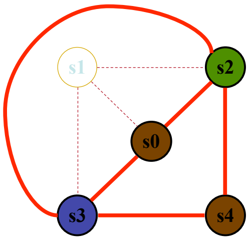


Another Coloring Example

N = 3 🟦 🟩 🟫

Another Coloring Example

N = 3 🟦 🟩 🟫



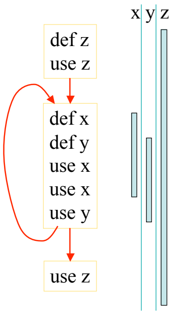
Outline

- What is Register Allocation
- A simple register Allocator
- Webs
- Interference Graphs
- Graph coloring
- Splitting
- More Optimizations

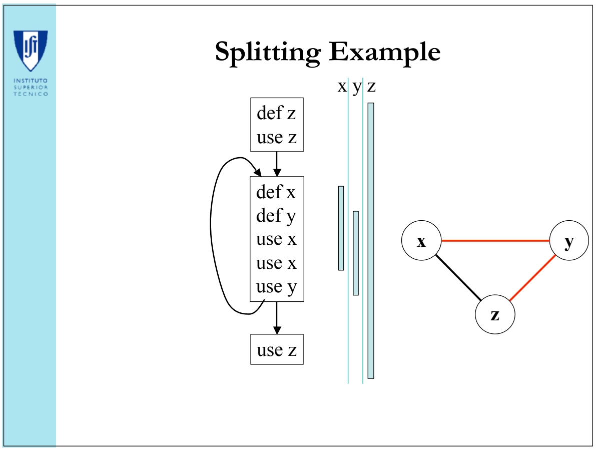
Spilling and Splitting

- When the graph is non-N-colorable
- Select a Web to Spill
  - Find the least costly Web to Spill
  - Use and Defs of that web are read and writes to memory
- Split the web
  - Split a web into multiple webs so that there will be less interference in the interference graph making it N-colorable
  - Spill the value to memory and load it back at the points where the web is split

Splitting Example



Splitting Example

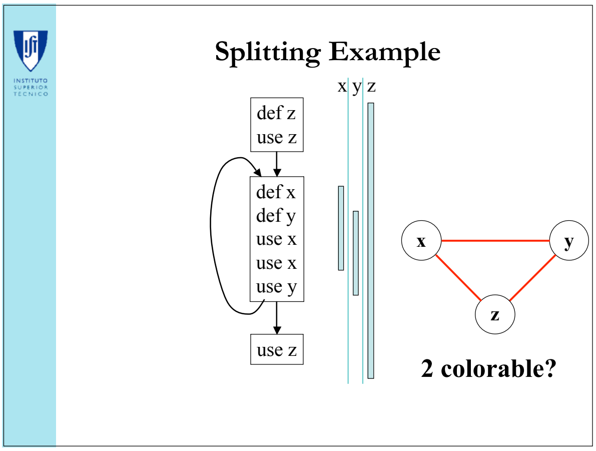


def z
use z

def x
def y
use x
use x
use y

use z

Splitting Example



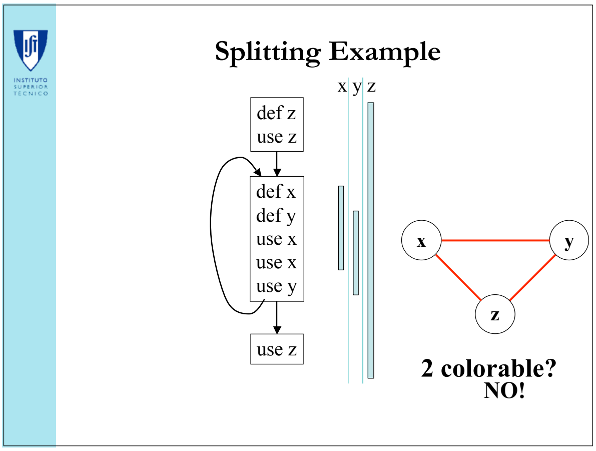
def z
use z

def x
def y
use x
use x
use y

use z

2 colorable?

Splitting Example



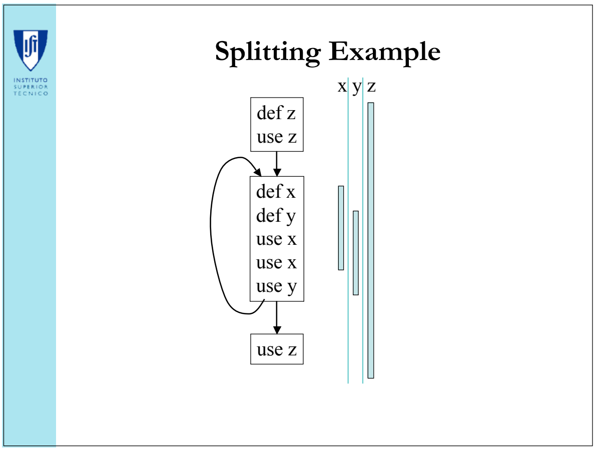
def z
use z

def x
def y
use x
use x
use y

use z

2 colorable?
NO!

Splitting Example

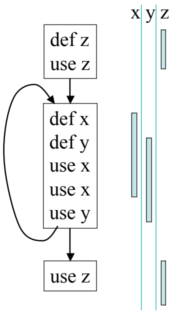


def z
use z

def x
def y
use x
use x
use y

use z

Splitting Example

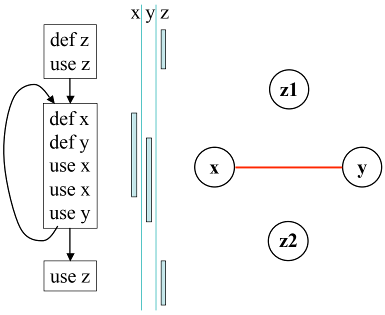


def z
use z

def x
def y
use x
use x
use y

use z

Splitting Example

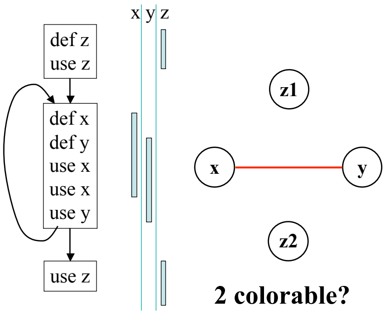


def z
use z

def x
def y
use x
use x
use y

use z

Splitting Example



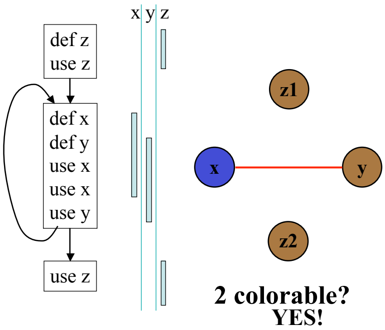
def z
use z

def x
def y
use x
use x
use y

use z

2 colorable?

Splitting Example

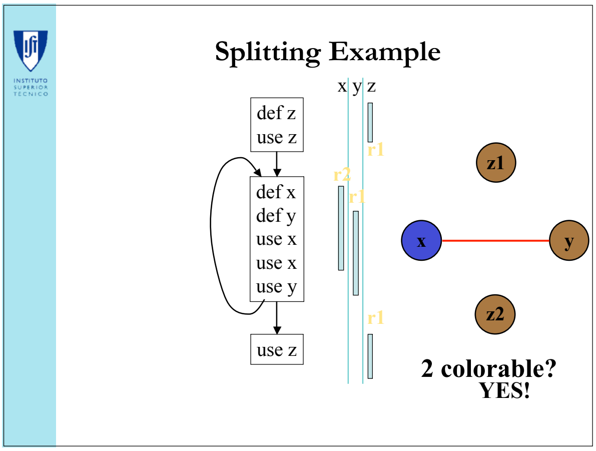


def z
use z

def x
def y
use x
use x
use y

use z

2 colorable?
YES!



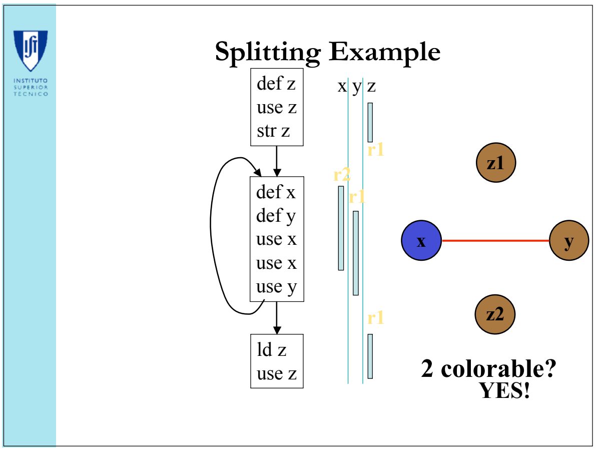
Splitting Example

```
def z
use z
```

```
def x
def y
use x
use x
use y
```

```
use z
```

2 colorable?
YES!



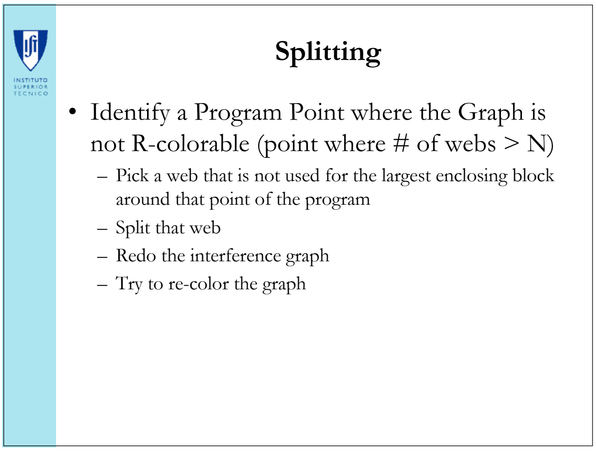
Splitting Example

```
def z
use z
str z
```

```
def x
def y
use x
use x
use y
```

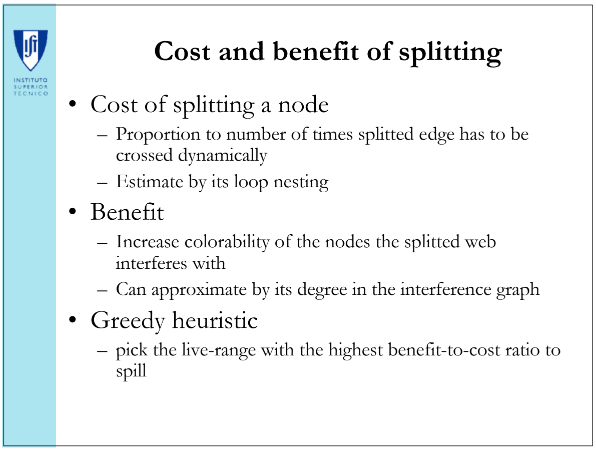
```
ld z
use z
```

2 colorable?
YES!



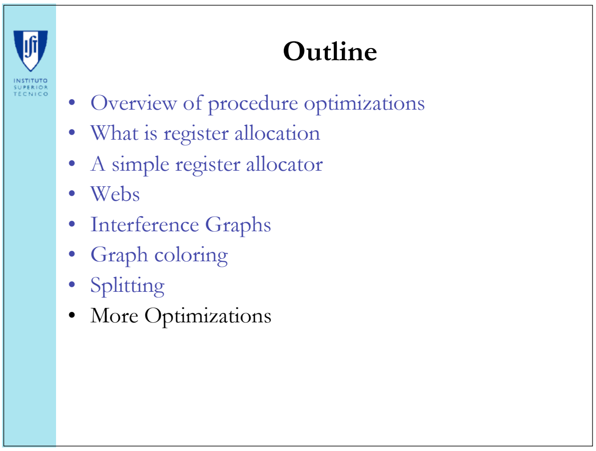
Splitting

- Identify a Program Point where the Graph is not R-colorable (point where # of webs > N)
  - Pick a web that is not used for the largest enclosing block around that point of the program
  - Split that web
  - Redo the interference graph
  - Try to re-color the graph



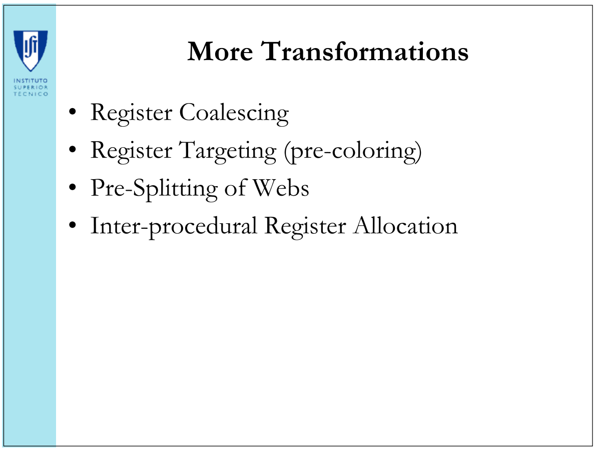
Cost and benefit of splitting

- Cost of splitting a node
  - Proportion to number of times splitted edge has to be crossed dynamically
  - Estimate by its loop nesting
- Benefit
  - Increase colorability of the nodes the splitted web interferes with
  - Can approximate by its degree in the interference graph
- Greedy heuristic
  - pick the live-range with the highest benefit-to-cost ratio to spill



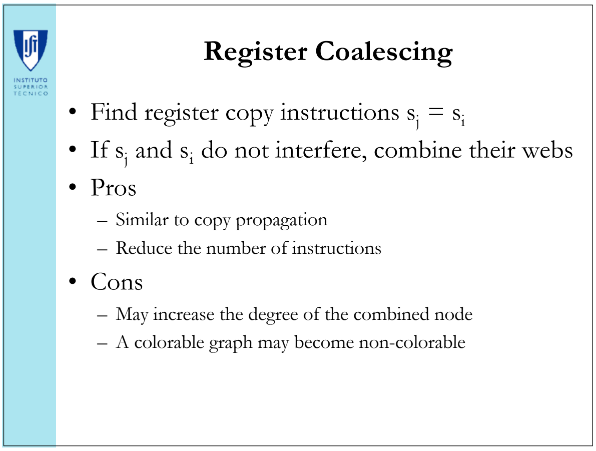
Outline

- Overview of procedure optimizations
- What is register allocation
- A simple register allocator
- Webs
- Interference Graphs
- Graph coloring
- Splitting
- More Optimizations



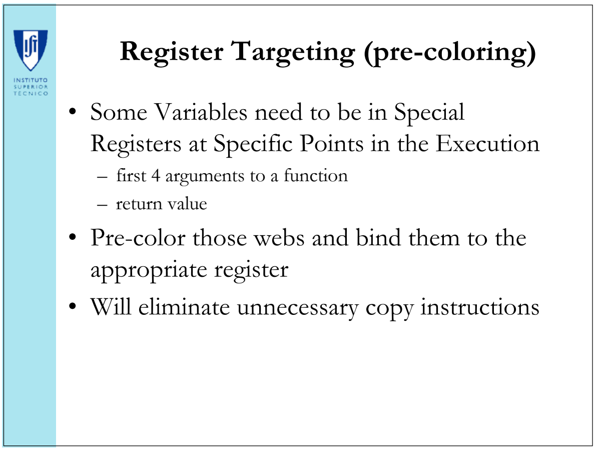
More Transformations

- Register Coalescing
- Register Targeting (pre-coloring)
- Pre-Splitting of Webs
- Inter-procedural Register Allocation



Register Coalescing

- Find register copy instructions $s_j = s_i$
- If s_j and s_i do not interfere, combine their webs
- Pros
  - Similar to copy propagation
  - Reduce the number of instructions
- Cons
  - May increase the degree of the combined node
  - A colorable graph may become non-colorable



Register Targeting (pre-coloring)

- Some Variables need to be in Special Registers at Specific Points in the Execution
  - first 4 arguments to a function
  - return value
- Pre-color those webs and bind them to the appropriate register
- Will eliminate unnecessary copy instructions

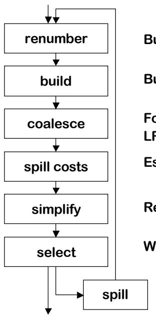
Pre-splitting of the webs

- Some live ranges have very large "dead" regions.
  - Large region where the variable is unused
- Break-up the live ranges
  - need to pay a small cost in spilling
  - but the graph will be very easy to color
- Can find strategic locations to break-up
  - at a call site (need to spill anyway)
  - around a large loop nest (reserve registers for values used in the loop)

Inter-Procedural Register Allocation

- Saving Registers across Procedure boundaries is expensive
  - especially for programs with many small functions
- Calling convention is too general and inefficient
- Customize calling convention per function by doing inter-procedural register allocation

Chaitin-Briggs Allocator



Build SSA, build live ranges, rename

Build the interference graph

Fold unneeded copies
$LR_x \rightarrow LR_y$, and $< LR_x, LR_y > \notin G_I \Rightarrow$ combine LR_x & LR_y

Estimate cost for spilling
each live range

Remove nodes from the graph

```
while N is non-empty
  if ∃ n with n° < k then
      push n onto stack
  else pick n to spill
      push n onto stack
  remove n from G_I
```

While stack is non-empty
pop n, insert n into G_I, & try to color it

Spill uncolored definitions & uses

Briggs' algorithm (1989)

Summary

- Register Allocation and Assignment
  - Very Important Transformations and Optimization
  - In General Hard Problem (NP-Complete)
- Many Approaches
  - Local Methods: Top-Down and Bottom-Up
  - Global Methods: Graph Coloring
    - Webs
    - Interference Graphs
    - Coloring
  - Other Transformations