Register Allocation

Advanced Compiler Techniques

2005

Erik Stenman

Virtutech

Register Allocation

- What is register allocation?
- Different types of register allocators.
- Webs.
- Interference Graphs.
- Graph coloring.
- Spilling.
- Live-Range Splitting.
- More optimizations.

Storing values between defs and uses

- Program computes with values
  - value definitions (where computed)
  - value uses (where read to compute new values)
- Values must be stored between def and use

  First Option:
  - store each value in memory at definition
  - retrieve from memory at each use

  Second Option:
  - store each value in register at definition
  - retrieve value from register at each use

Issues

- On a typical RISC architecture:
  - All computation takes place in registers.
  - Load instructions and store instructions transfer values between memory and registers.
- Add two numbers; values in memory:

```
load r1, 4(sp)
load r2, 8(sp)
add r3,r1,r2
store r3, 12(sp)
```

Issues

- On a typical RISC architecture:
  - All computation takes place in registers.
  - Load instructions and store instructions transfer values between memory and registers.
- Add two numbers; values in registers:

```
add r3,r1,r2
```

Issues

- Fewer instructions when using registers.
  - Most instructions are register-to-register.
  - Additional instructions for memory accesses.
- Registers are faster than memory.
  - Wider gap in faster, newer processors.
  - Factor of about 4 bandwidth, factor of about 3 latency.
  - Could be bigger depending on program characteristics.
- But only a small number of registers available.
  - Usually 32 integer and 32 floating-point registers.
  - Some of those registers have fixed users (r0, ra, sp, fp).

Register Allocation

- Deciding which values to store in a limited number of registers.
- Register allocation has a direct impact on performance.
  - Affects almost every statement of the program.
  - Eliminates expensive memory instructions.
  - # of instructions goes down due to direct manipulation of registers (no need for load and store instructions).
  - This is probably the optimization with the most impact!

What can be put in a register?

- Values stored in compiler-generated temps.
- Language-level values:
  - Values stored in local scalar variables.
  - Big constants.
  - Values stored in array elements and object fields
    - Issue: alias analysis
- Register set depends on the data-type:
  - floating-point values in floating point registers.
  - integer and pointer values in integer registers.

Allocation vs Assignment?

- We sometimes distinguishes between register allocation and register assignment.
- Register allocation deals with the problem to decide which values to store in registers and which to spill to memory.
- Register assignment decides which value goes into which register.

Different Types of Register Allocation

- Local Register allocation.
  - Tree-based approaches:
    - Sethi-Ullman numbering.
  - Basic Block.
- Global Register allocation.
  - Linear Scan.
  - Graph Coloring.
- Inter-procedural allocation.

Web-Based Register Allocation

- Determine live ranges for each value (*web*).
- Determine overlapping ranges (interference).
- Compute the benefit of keeping each web in a register (spill cost).
- Decide which webs get a register (allocation).
- Split webs if needed (spilling and splitting).
- Assign hard registers to webs (assignment).
- Generate code including spills (code gen.).

Webs

- Web is unit of register allocation.
- If web allocated to a given register R:
  - All definitions computed into R.
  - All uses read from R.
- If web allocated to a memory location M:
  - All definitions computed into M.
  - All uses read from M.
- Issue: instructions compute only from registers.
- Reserve some registers to hold memory values.

Convex Sets and Live Ranges

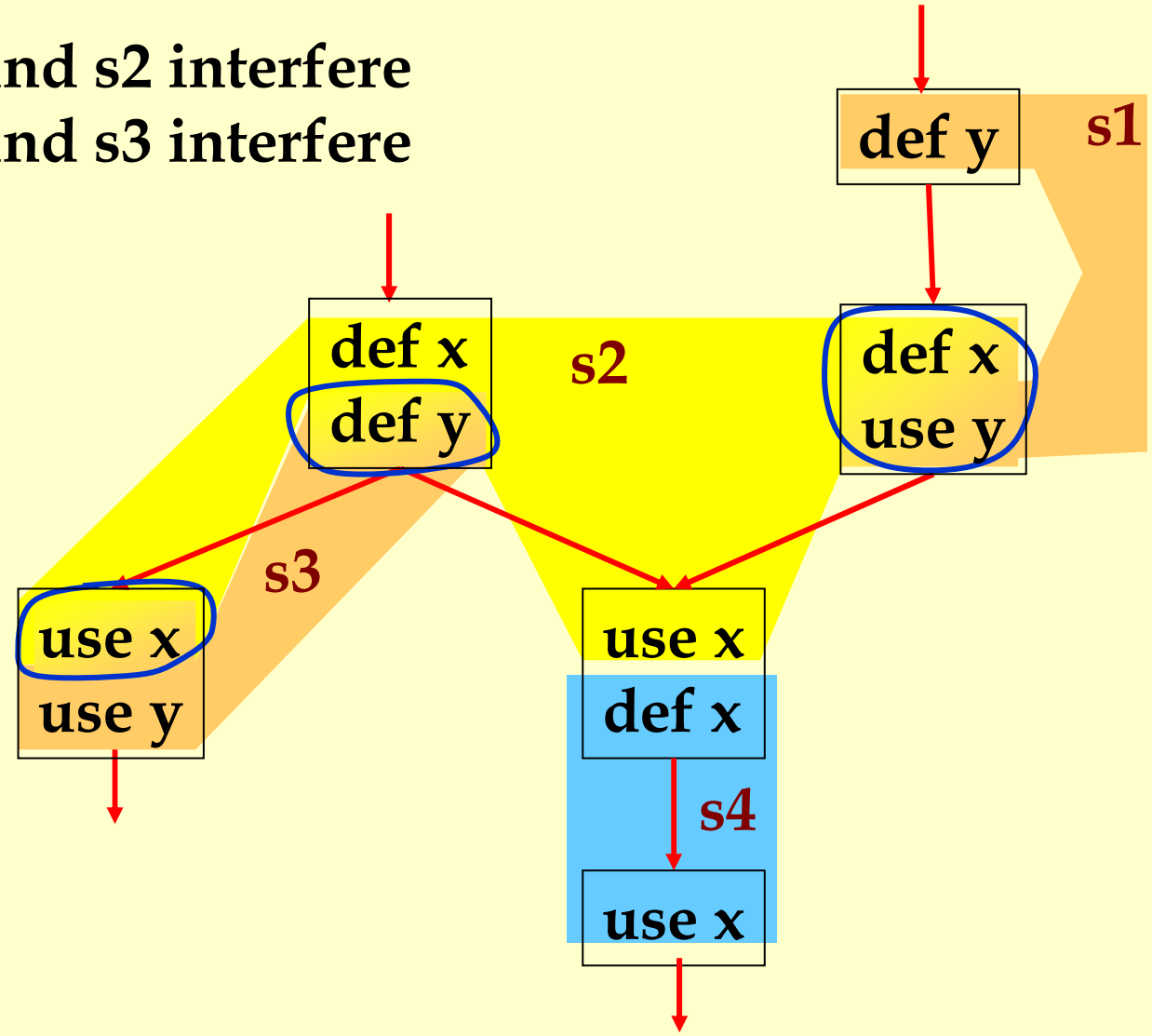
- Concept of convex set.
- A set $\mathbb{S}$ is convex if
  - a, b $\in \mathbb{S}$ and c is on a path from a to b implies c $\in \mathbb{S}$
- Concept of live range of a web.
  - Minimal convex set of instructions that includes all defs and uses in web.
  - Intuitively, region in which web's value is live.

Interference

- Two webs interfere if their live ranges overlap (have a nonempty intersection).
- If two webs interfere, values must be stored in different registers or memory locations.
- If two webs do not interfere, can store values in same register or memory location.

Example

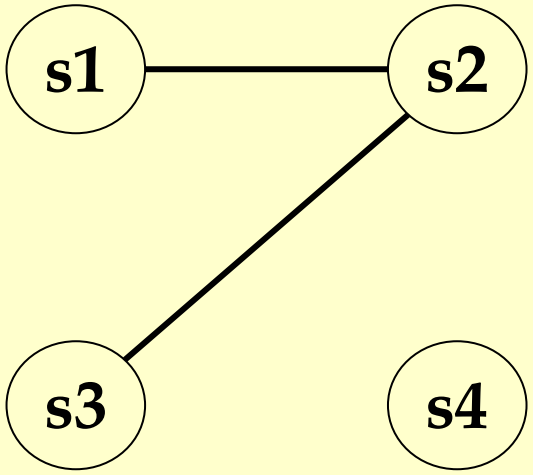
Webs s1 and s2 interfere
Webs s2 and s3 interfere



Interference Graph

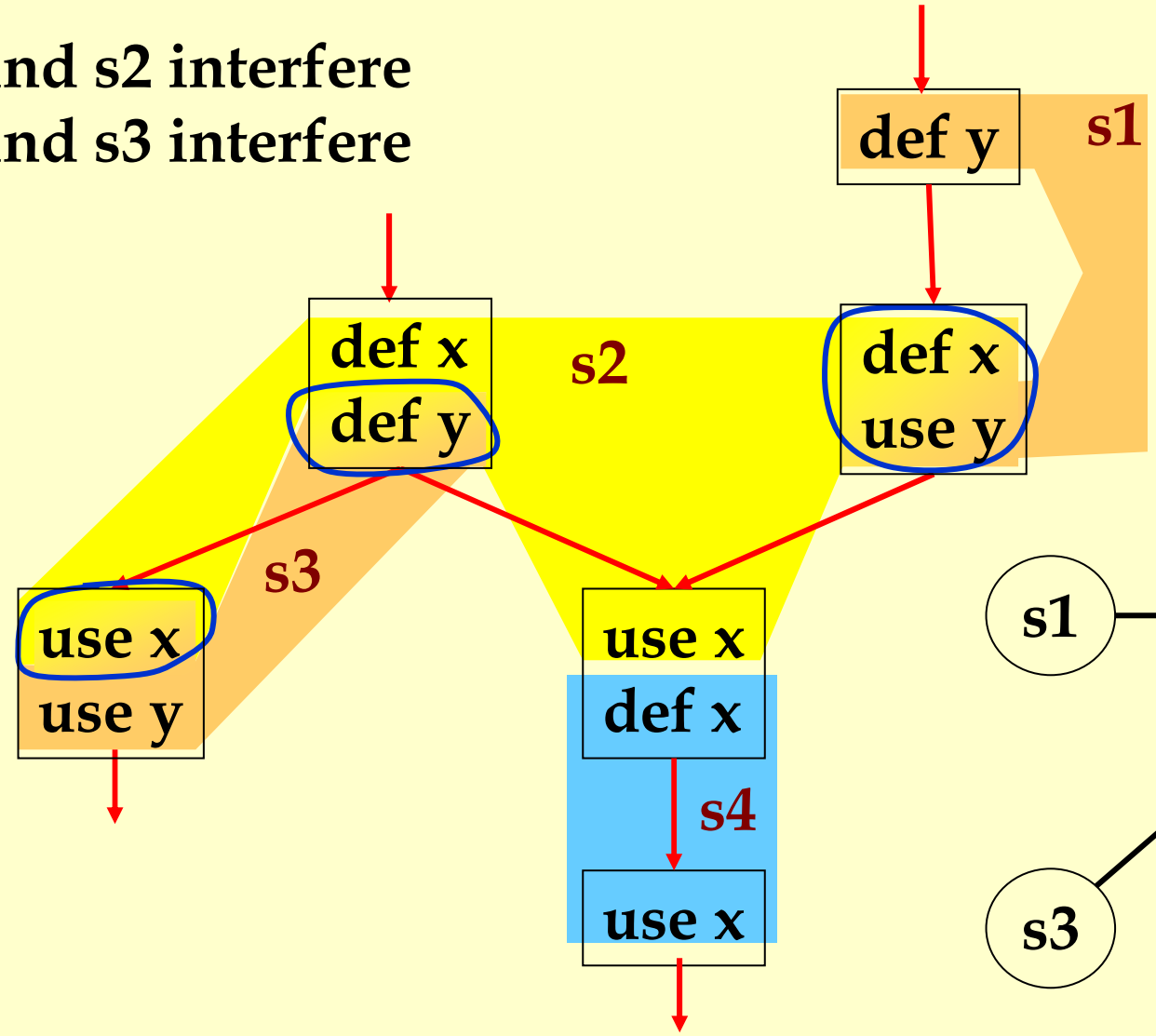
Representation of webs and their interference:

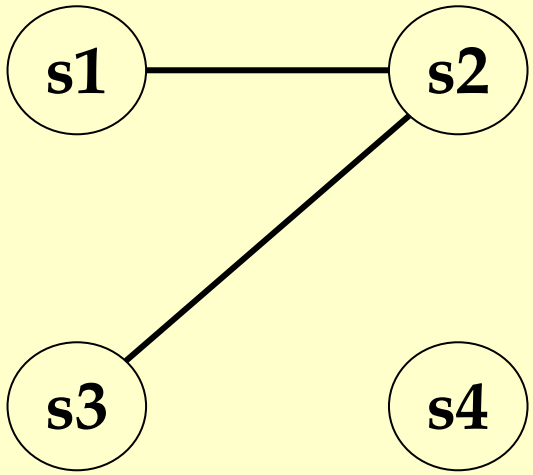
- Nodes are the webs
- An edge exists between two nodes if they interfere:



Example

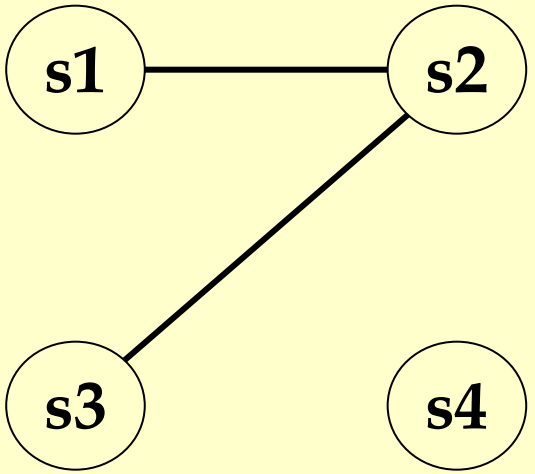
Webs s1 and s2 interfere
Webs s2 and s3 interfere





Register Allocation Using Graph Coloring

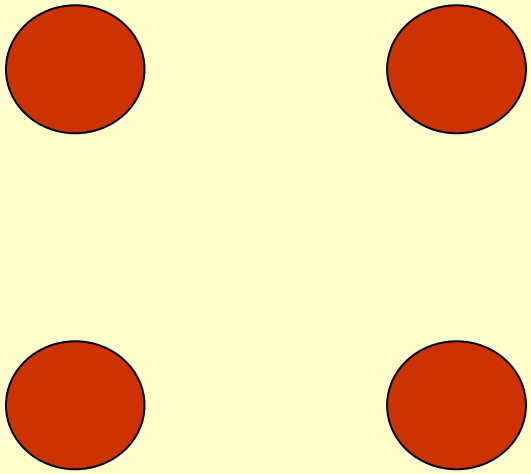
- Each web is allocated to a register.
  - Each node gets a register (color).
- If two webs interfere they cannot use the same register.
  - If two nodes have an edge between them, they cannot have the same color.



Graph Coloring

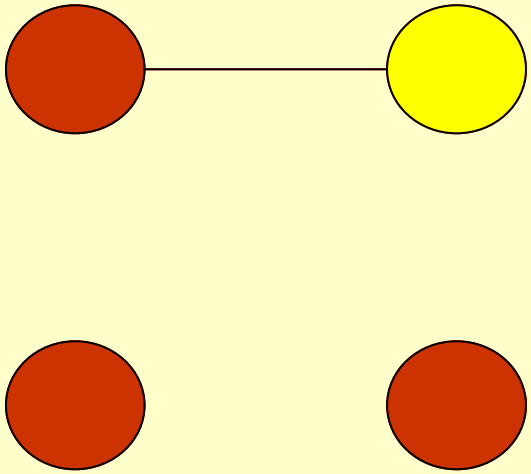
- Assign a color to each node in the graph.
- Two nodes connected to same edge must have different colors.
- Classic problem in graph theory.
- NP complete.
  - But good heuristics exist for register allocation.

Graph Coloring Example



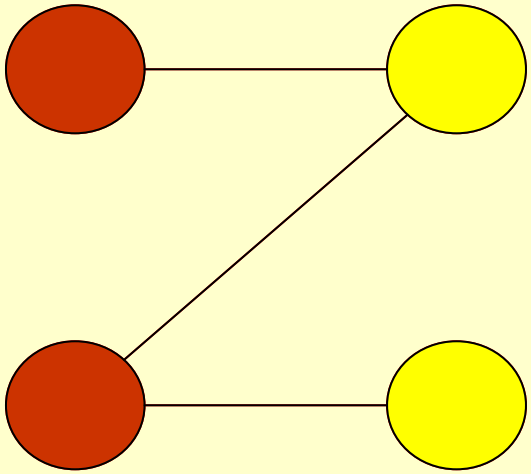
1 Color

Graph Coloring Example



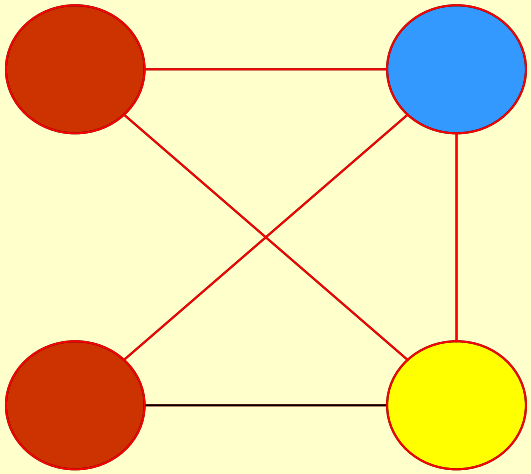
2 Colors

Graph Coloring Example



Still 2 Colors

Graph Coloring Example



3 Colors

Heuristics for Register Coloring

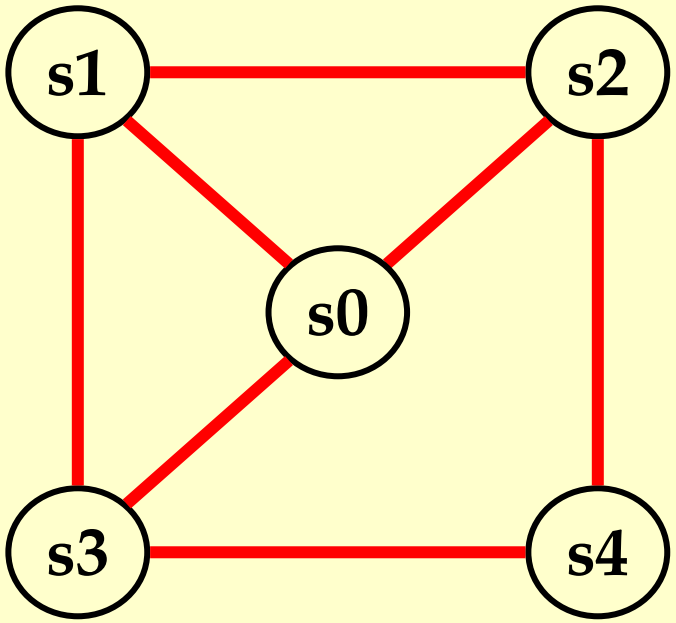
- Coloring a graph with N colors.
- If degree < N (degree of a node = # of edges):
  - Node can always be colored.
  - After coloring the rest of the nodes, there is at least one color left to color the current node.
- If degree >= N:
  - Still may be colorable with N colors. (If some neighbors are colored with the same color.)

Heuristics for Register Coloring

- Remove nodes that have degree < N.
  - Push the removed nodes onto a stack.
- When all the nodes have degree >= N:
  - Find a node to spill (no color for that node).
  - Push that node into the stack.
- When empty, start to color:
  - Pop a node from stack back.
  - Assign it a color that is different from its connected nodes (if possible).

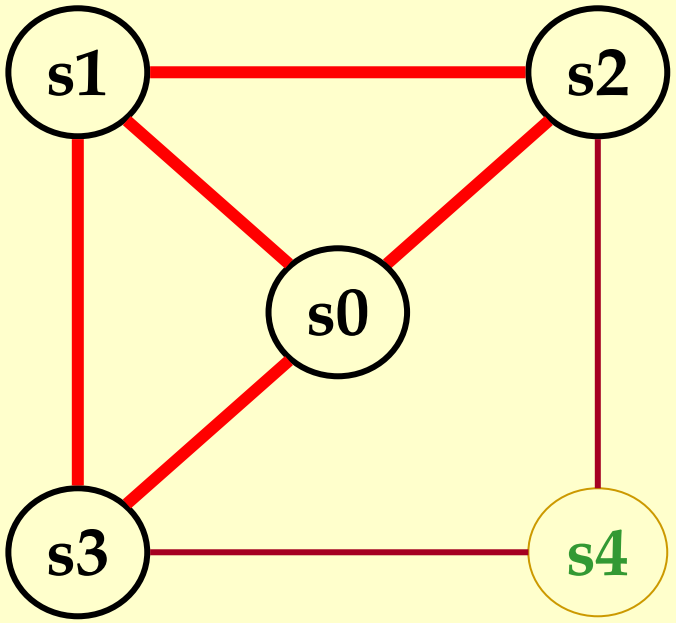
Coloring Example

N = 3



Coloring Example

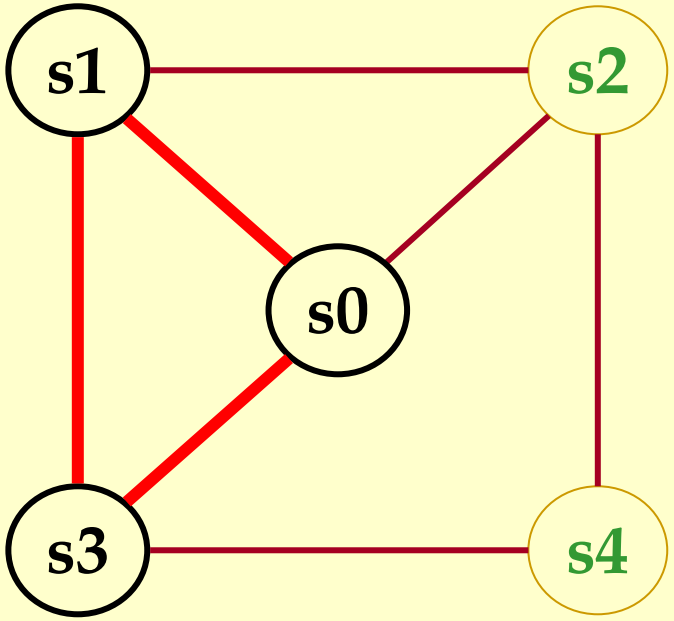
N = 3



s4

Coloring Example

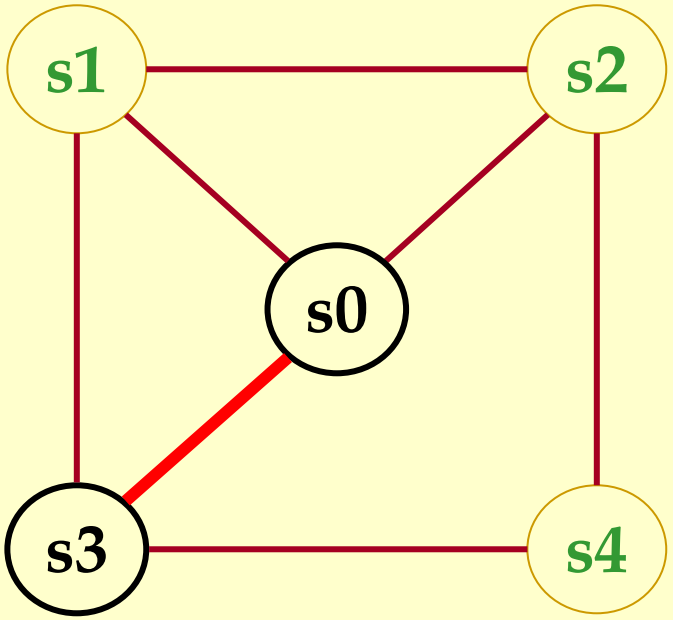
N = 3



s2
s4

Coloring Example

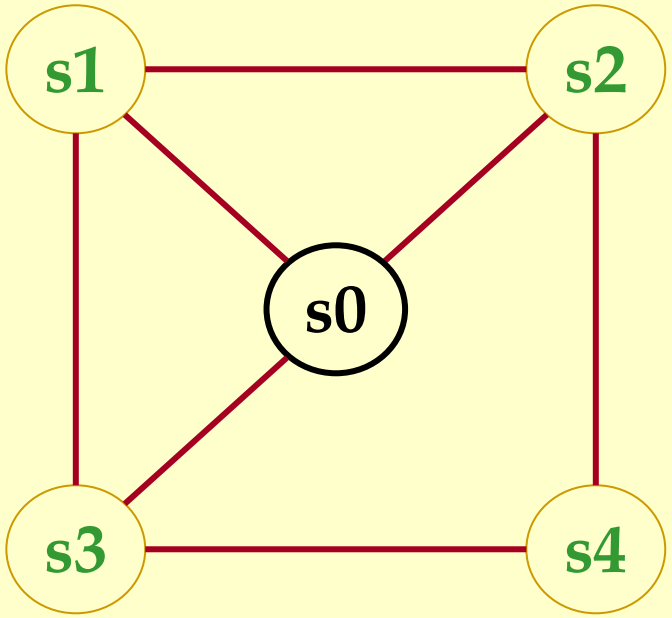
N = 3



s1
s2
s4

Coloring Example

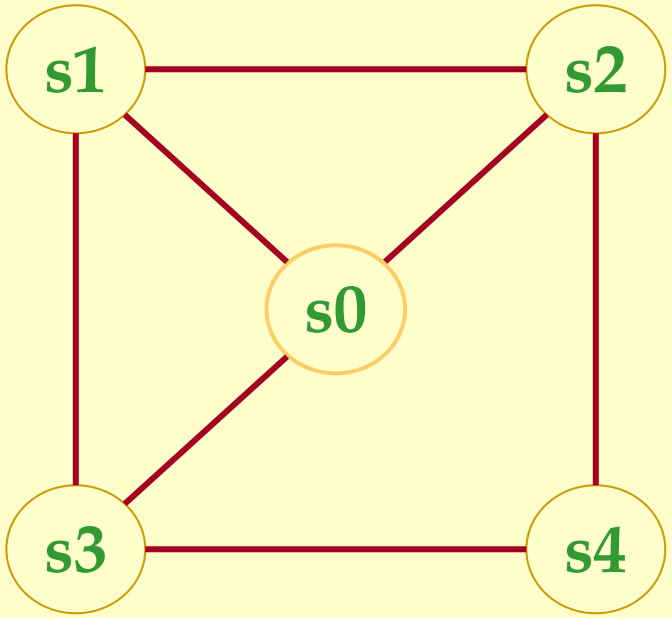
N = 3



s3
s1
s2
s4

Coloring Example

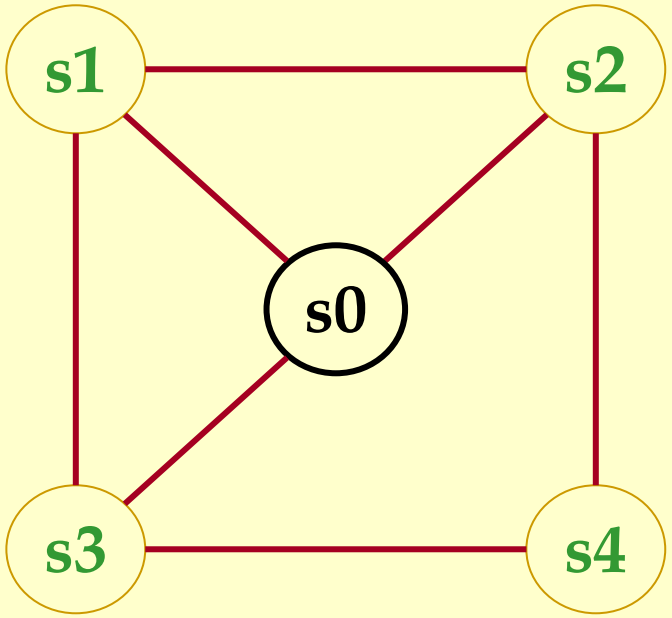
N = 3



s0
s3
s1
s2
s4

Coloring Example

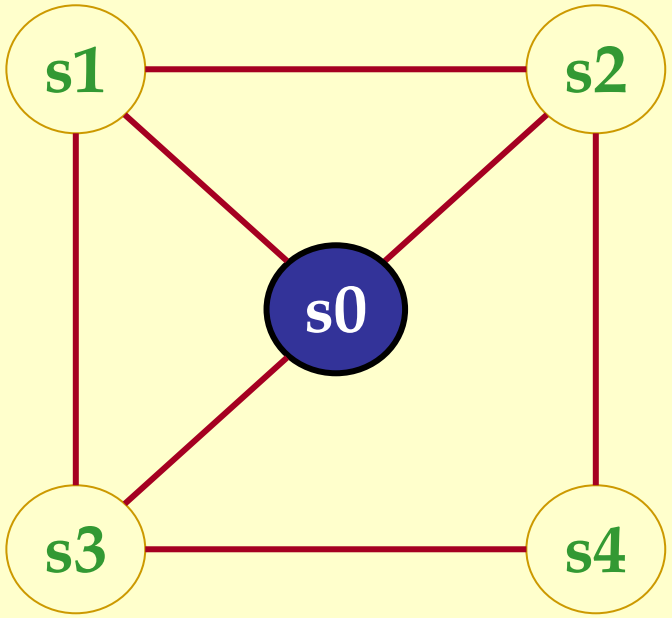
N = 3



s3
s1
s2
s4

Coloring Example

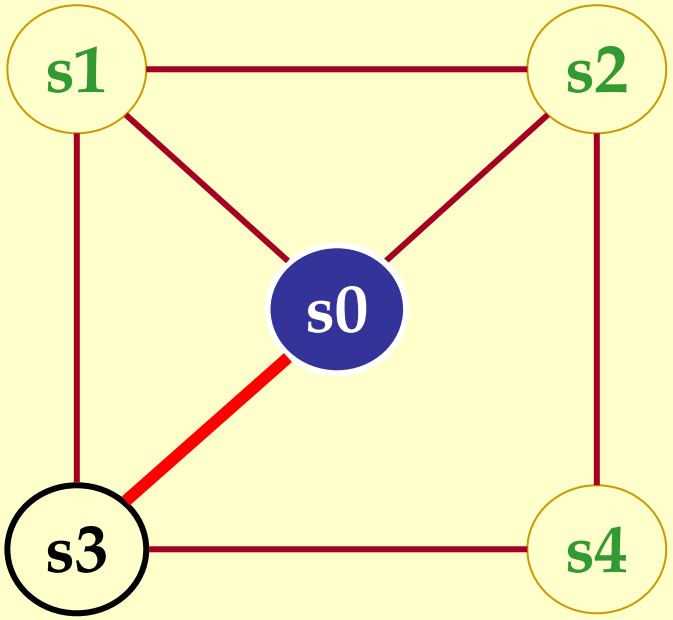
N = 3



s3
s1
s2
s4

Coloring Example

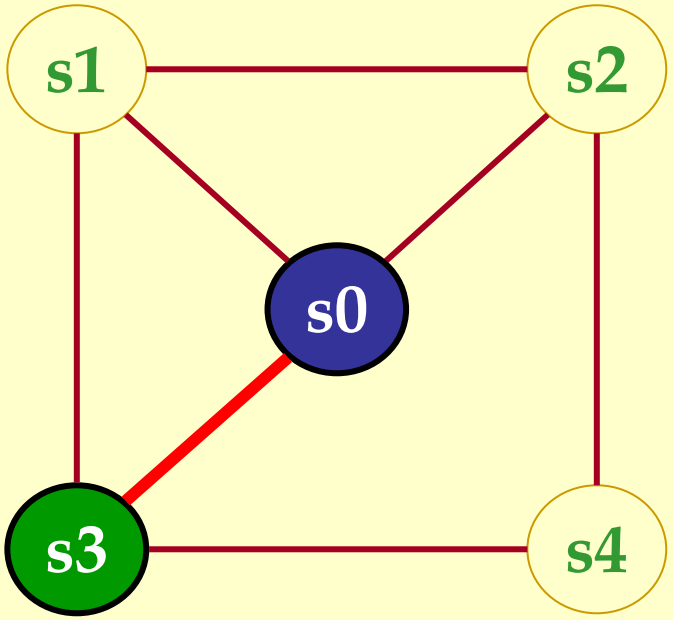
N = 3



s1
s2
s4

Coloring Example

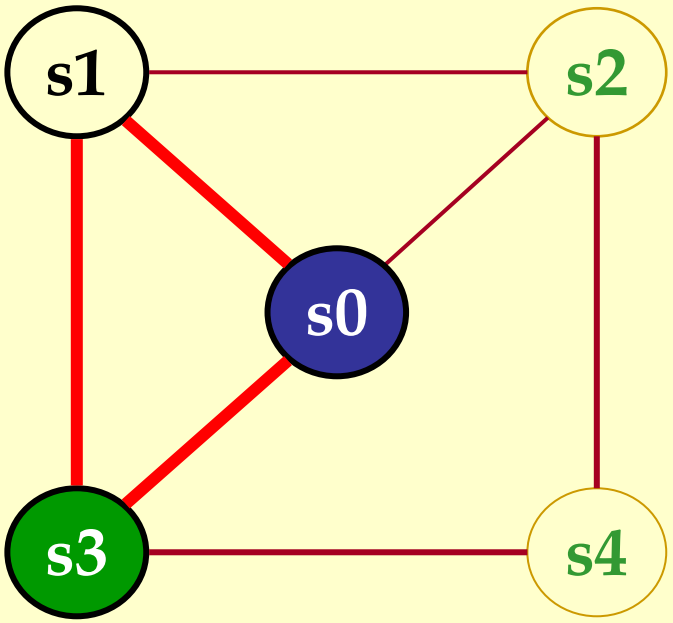
N = 3



s1
s2
s4

Coloring Example

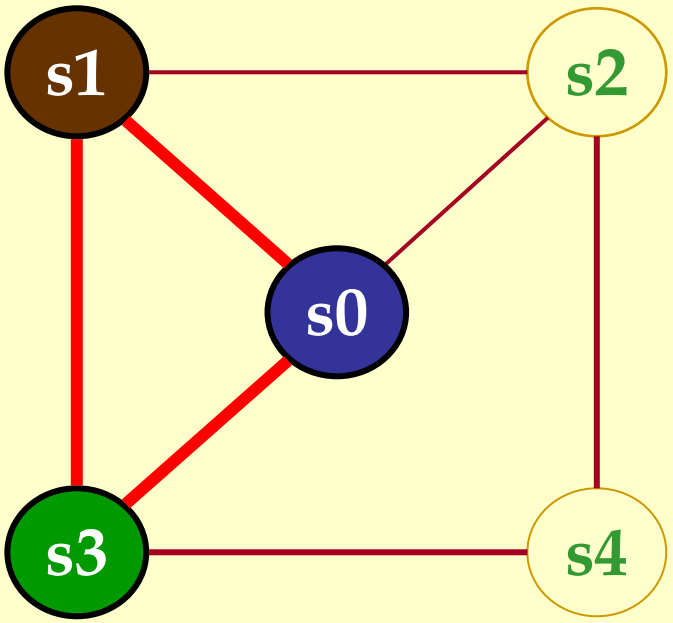
N = 3



s2
s4

Coloring Example

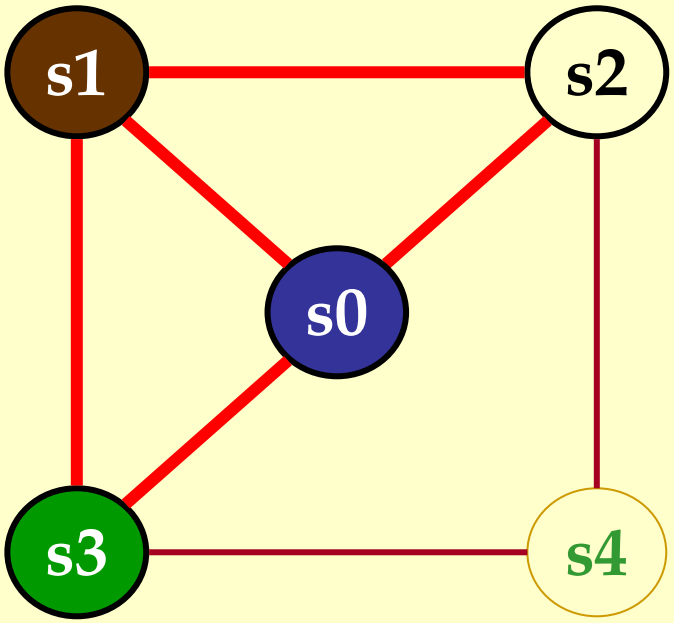
N = 3



s2
s4

Coloring Example

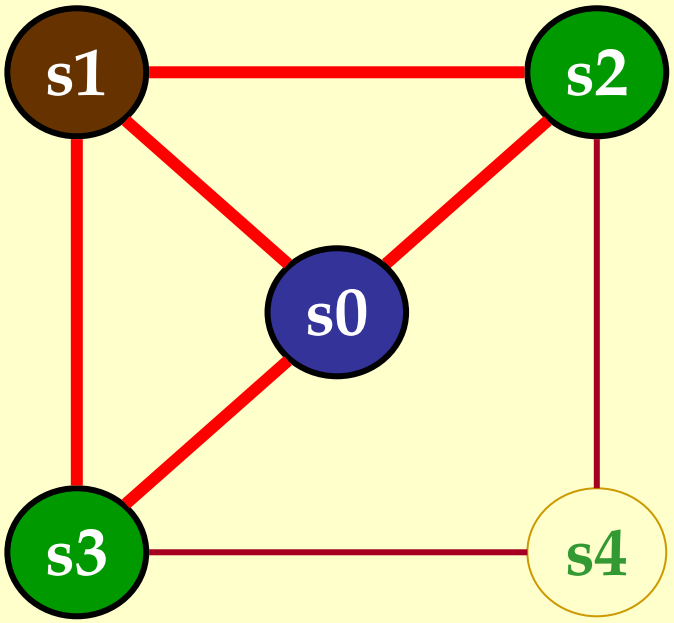
N = 3



s4

Coloring Example

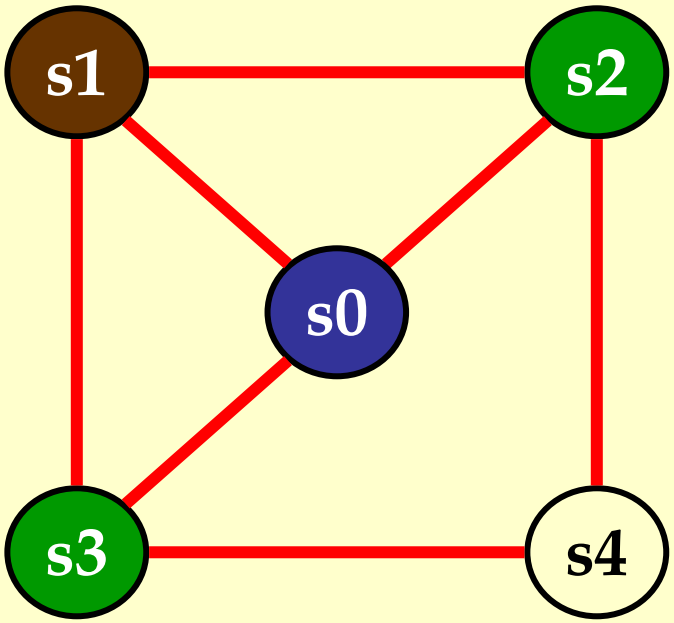
N = 3



s4

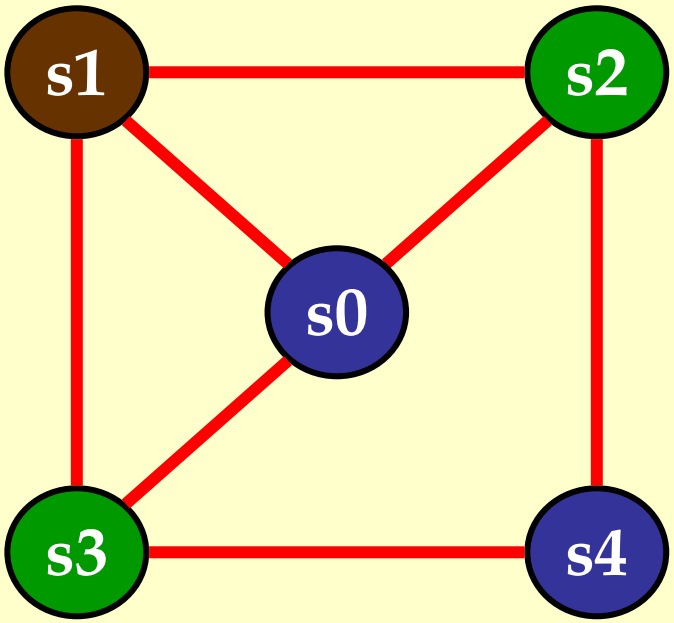
Coloring Example

N = 3



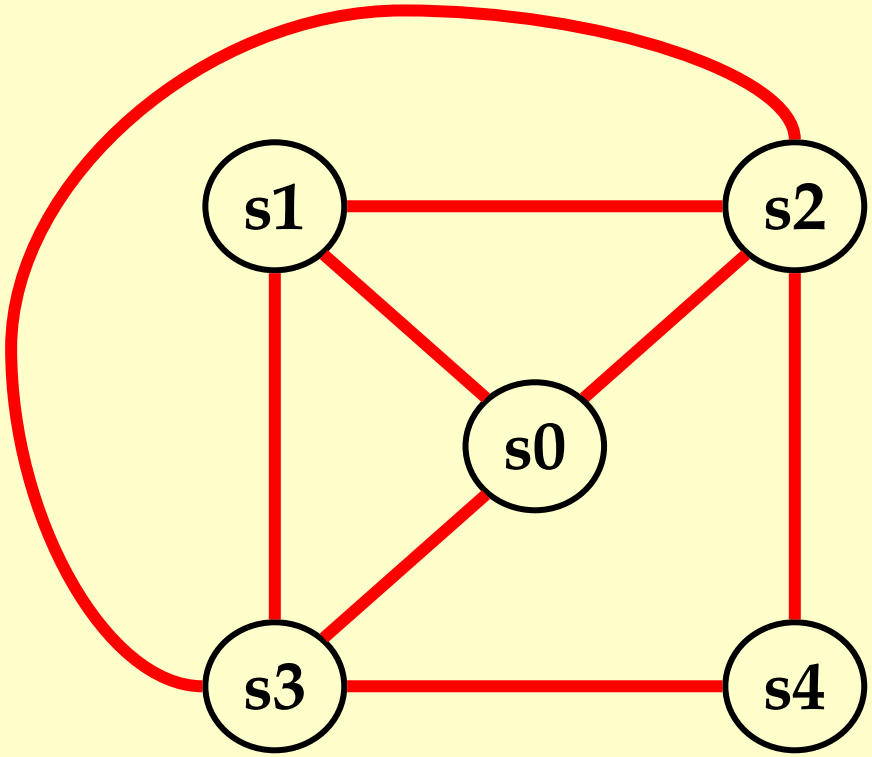
Coloring Example

N = 3



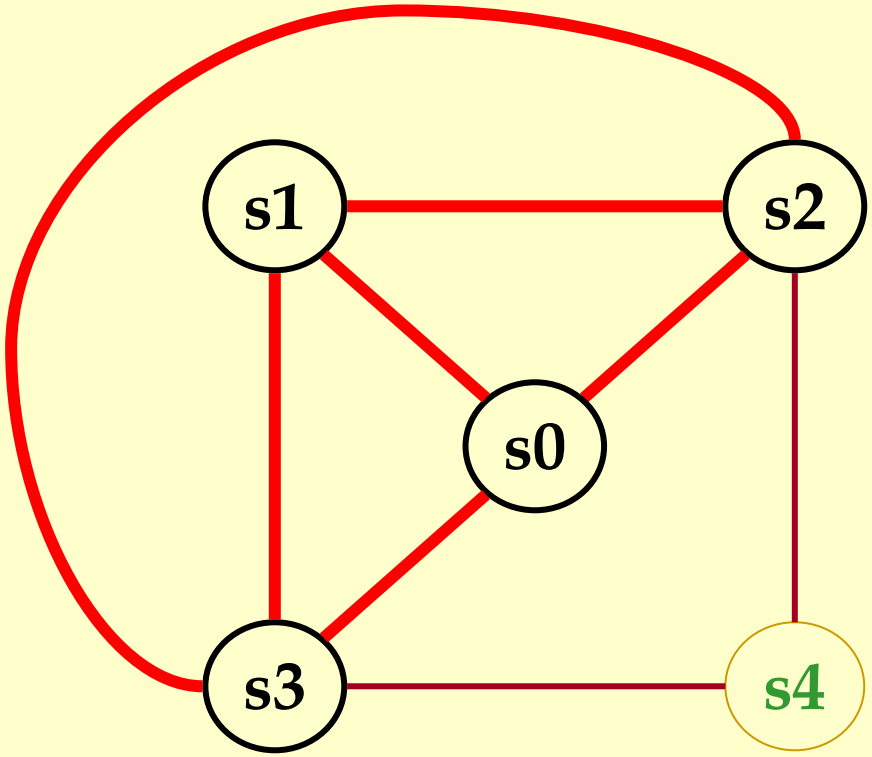
Another Coloring Example

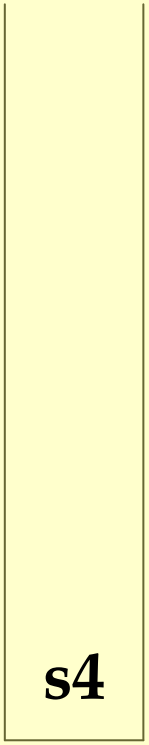
N = 3



Another Coloring Example

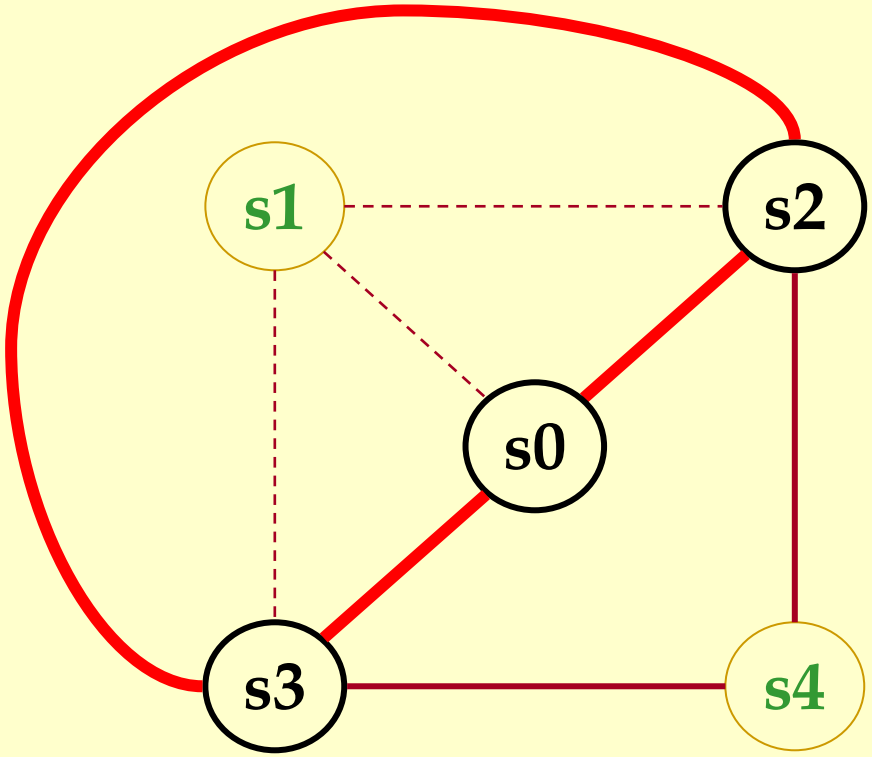
N = 3

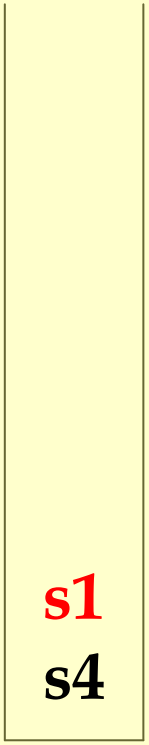




Another Coloring Example

N = 3

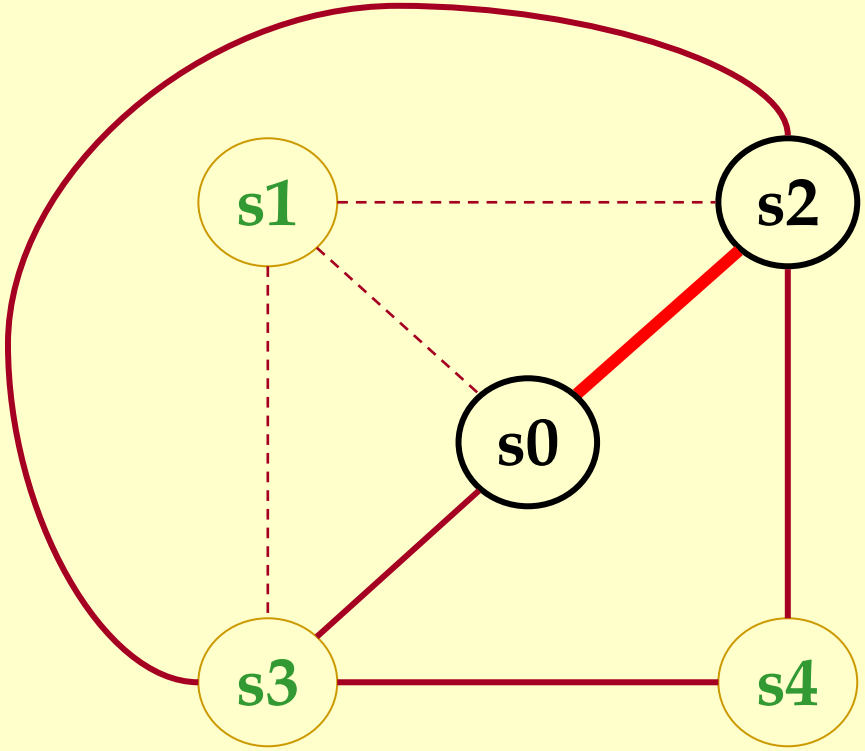




s1: Possible Spill

Another Coloring Example

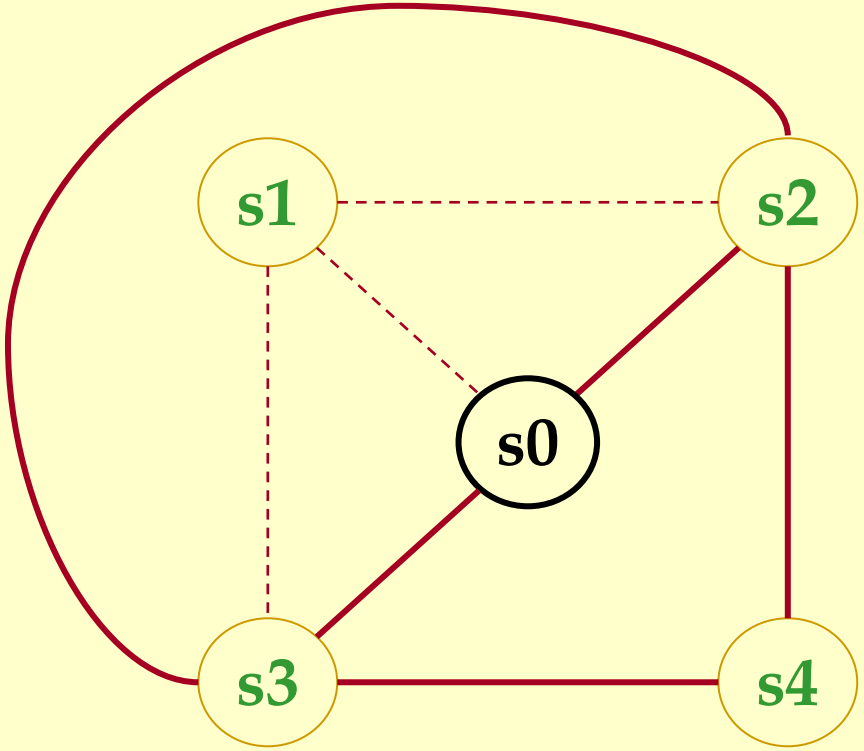
N = 3



s3
s1
s4

Another Coloring Example

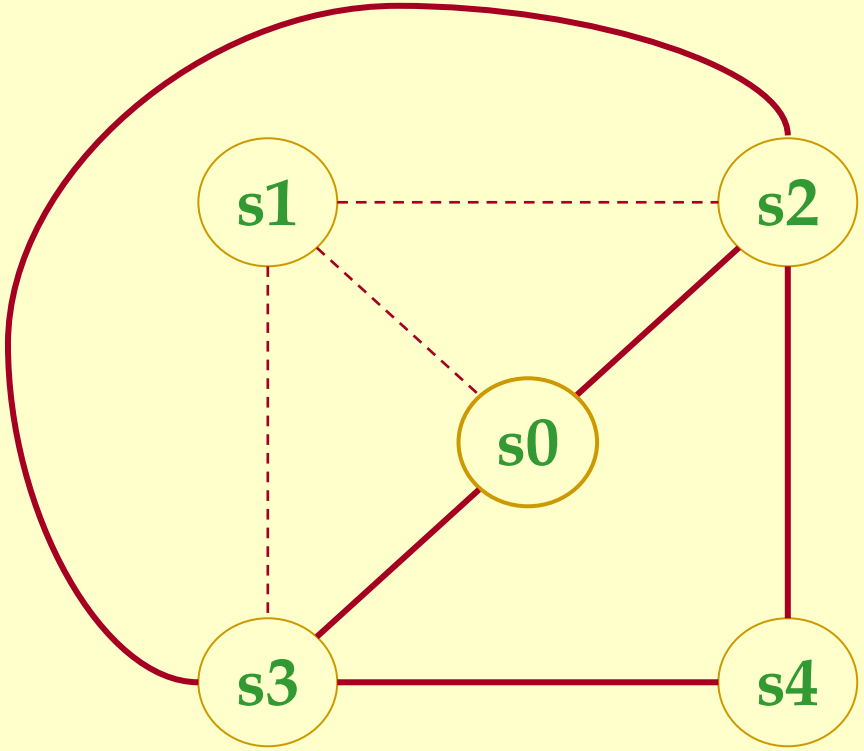
N = 3



s2
s3
s1
s4

Another Coloring Example

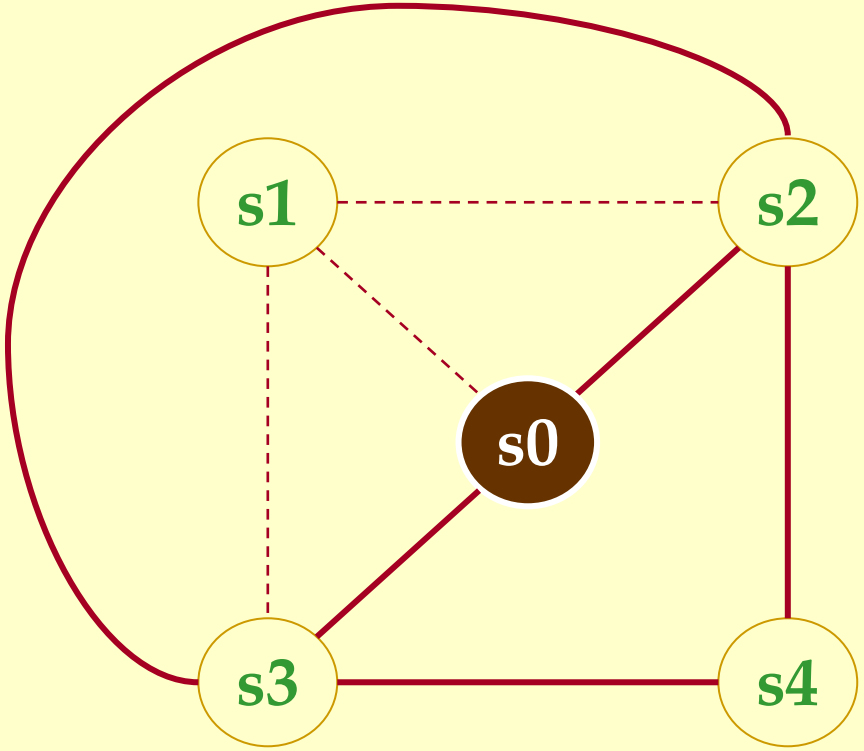
N = 3



s0
s2
s3
s1
s4

Another Coloring Example

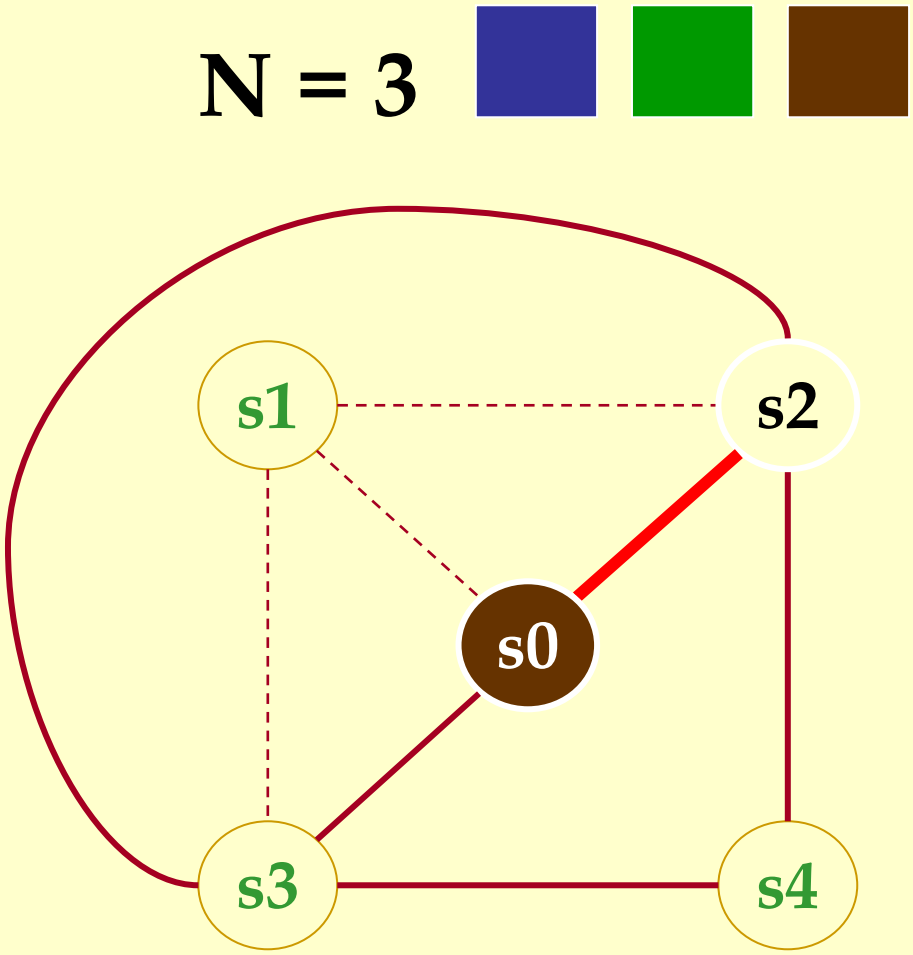
N = 3



s2
s3
s1
s4

Another Coloring Example

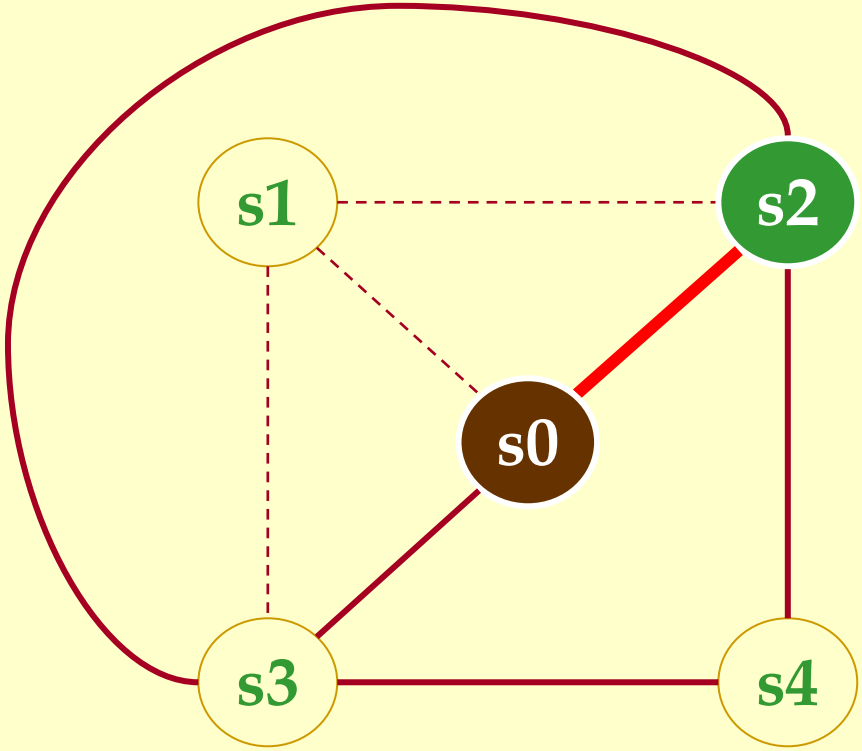
N = 3



s3
s1
s4

Another Coloring Example

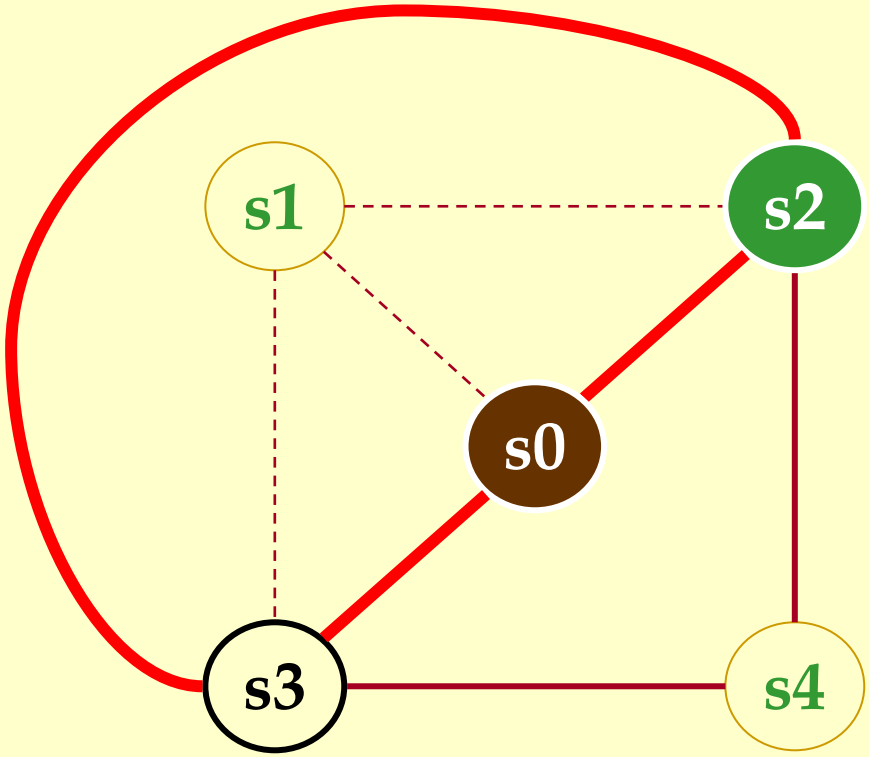
N = 3



s3
s1
s4

Another Coloring Example

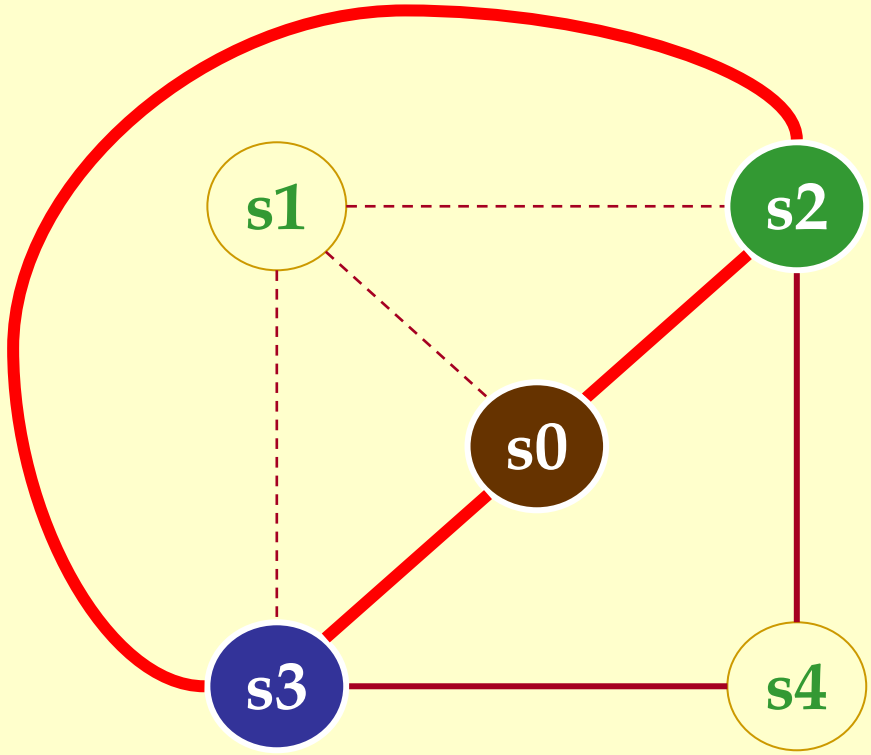
N = 3



s1
s4

Another Coloring Example

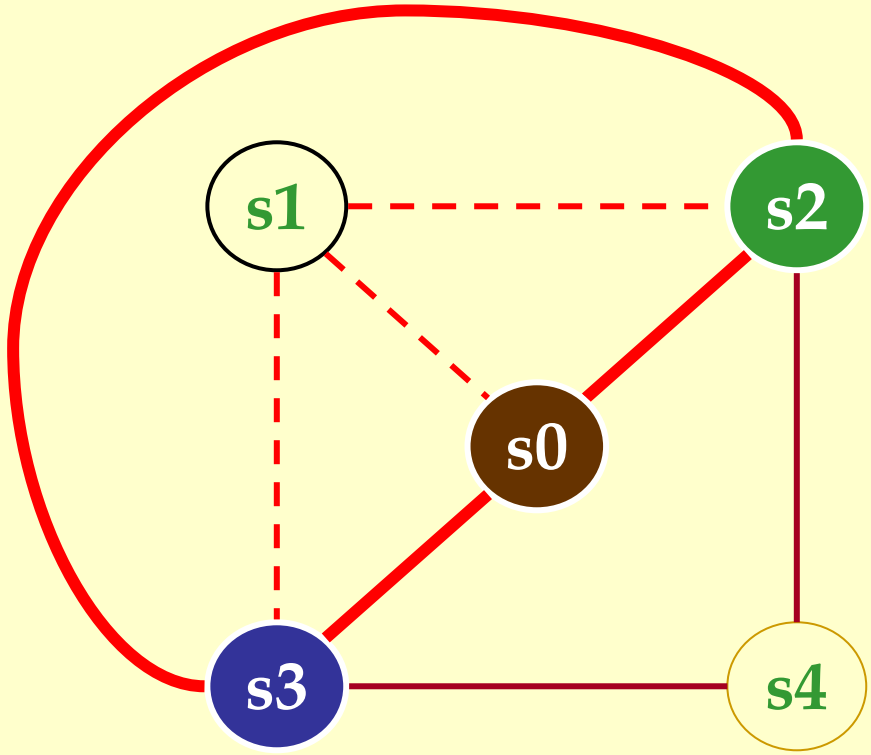
N = 3



s1
s4

Another Coloring Example

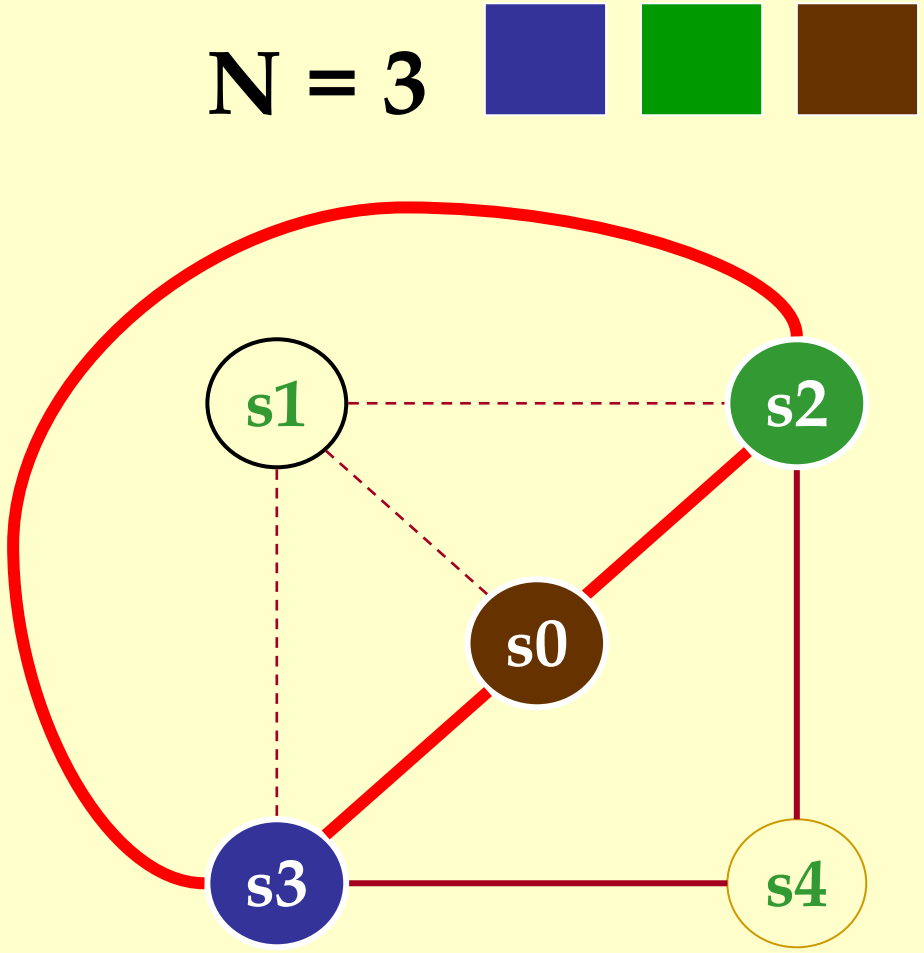
N = 3

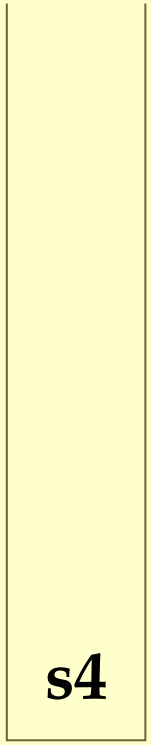


s4

Another Coloring Example

N = 3

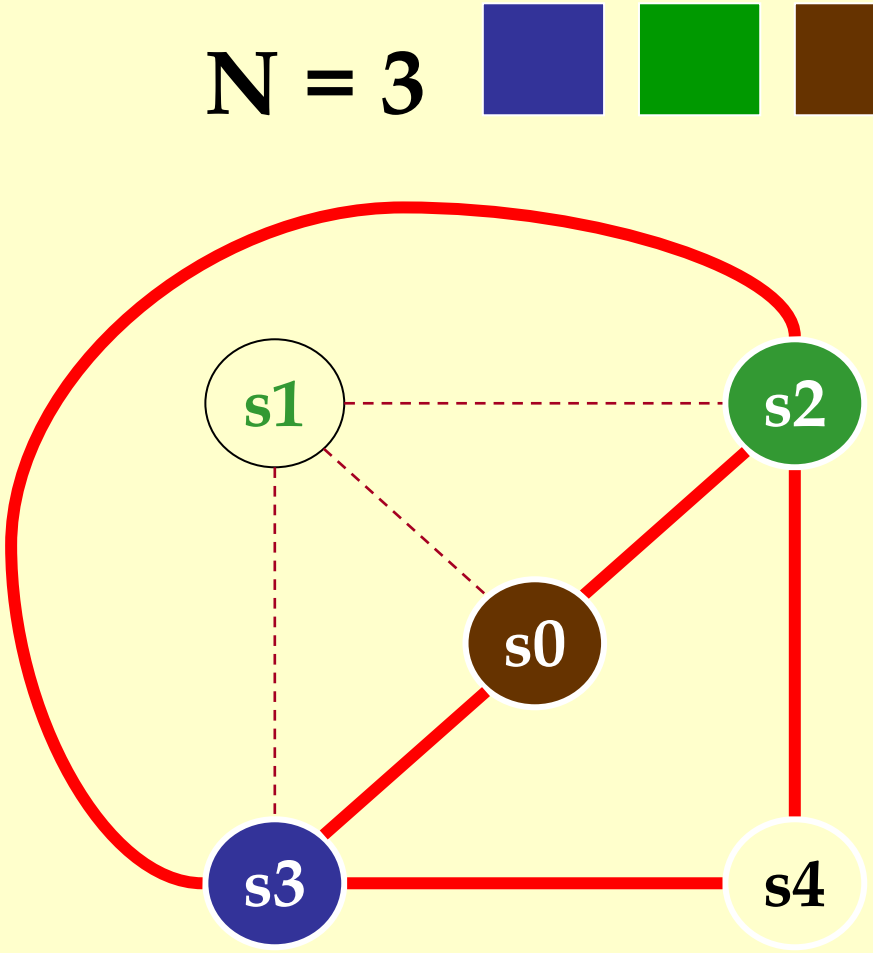




s1: Actual Spill

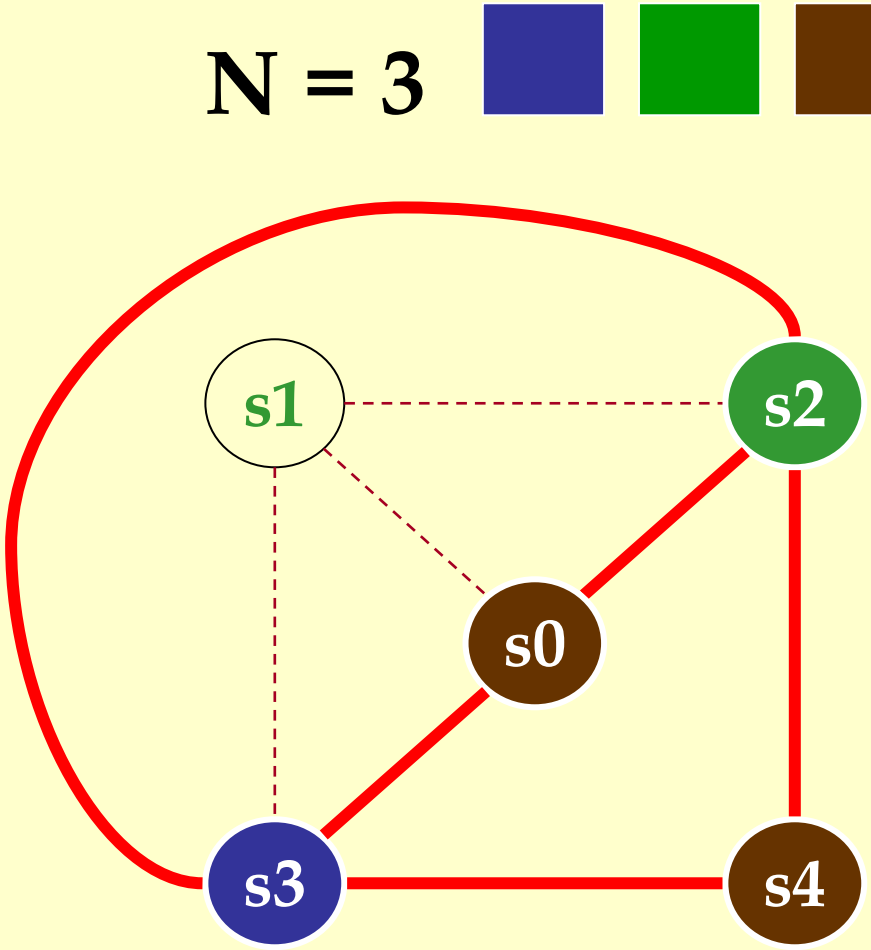
Another Coloring Example

N = 3



Another Coloring Example

N = 3



When Coloring Heuristics Fail...

Option 1:

- Pick a web and allocate value in memory.
- All defs go to memory, all uses come from memory.

Option 2:

- Split the web into multiple webs.

- In either case, will retry the coloring.

Which web to spill?

- One with interference degree >= N.
- One with minimal spill cost (cost of placing value in memory rather than in register).
- What is spill cost?
  - Cost of extra load and store instructions.

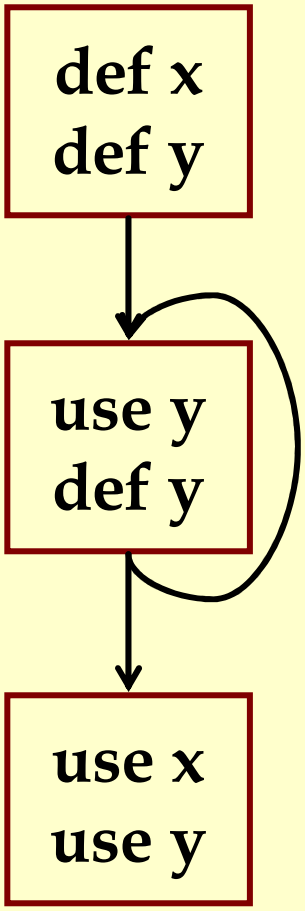
Ideal and Useful Spill Costs

- Ideal spill cost - dynamic cost of extra load and store instructions. Can't expect to compute this.
  - Don't know which way branches resolve.
  - Don't know how many times loops execute.
  - Actual cost may be different for different executions.
- Solution: Use a static approximation.
  - profiling can give instruction execution frequencies.
  - or use heuristics based on structure of control flow graph.

One Way to Compute Spill Cost

- Goal: give priority to values used in loops.
- So assume loops execute 10 (or 8) times.
- Spill cost =
  - sum over all def sites of cost of a store instruction times 8 to the loop nesting depth power, plus
  - sum over all use sites of cost of a load instruction times 8 to the loop nesting depth power.
- Choose the web with the lowest spill cost.

Spill Cost Example



Spill Cost For x
storeCost+loadCost

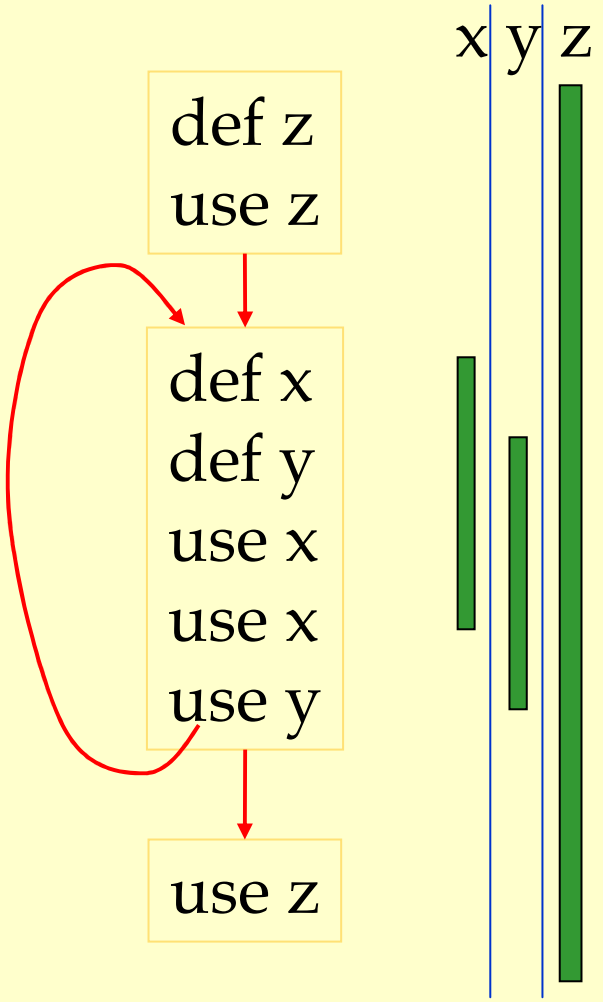
Spill Cost For y
9*storeCost+9*loadCost

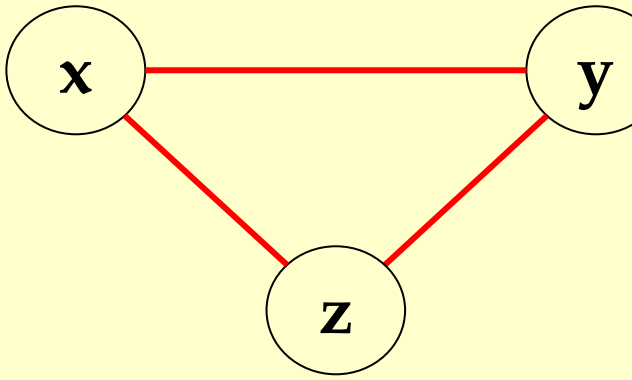
With 1 Register, Which Variable Gets Spilled?

Splitting Rather Than Spilling

- 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.

Live-Range Splitting Example

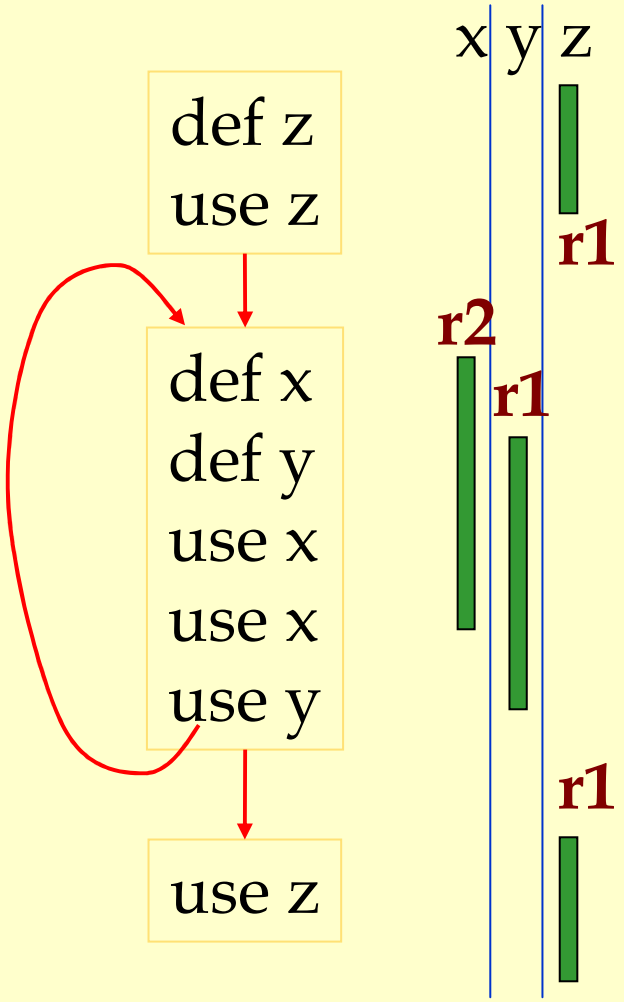


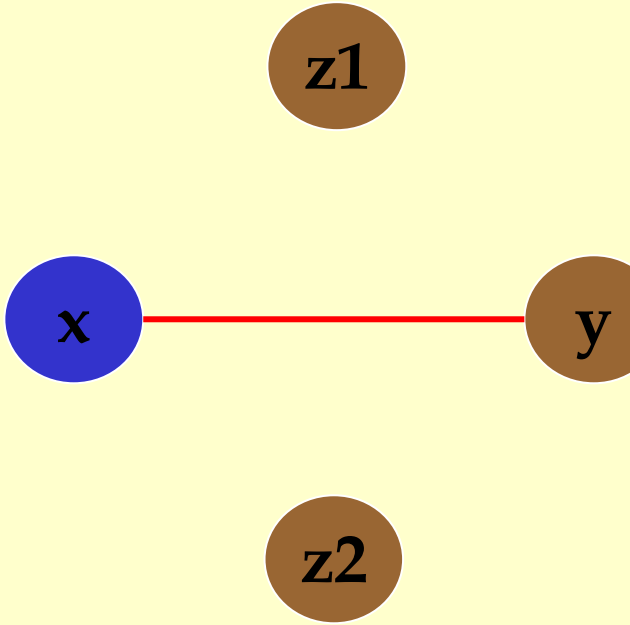


2 colorable?

NO!

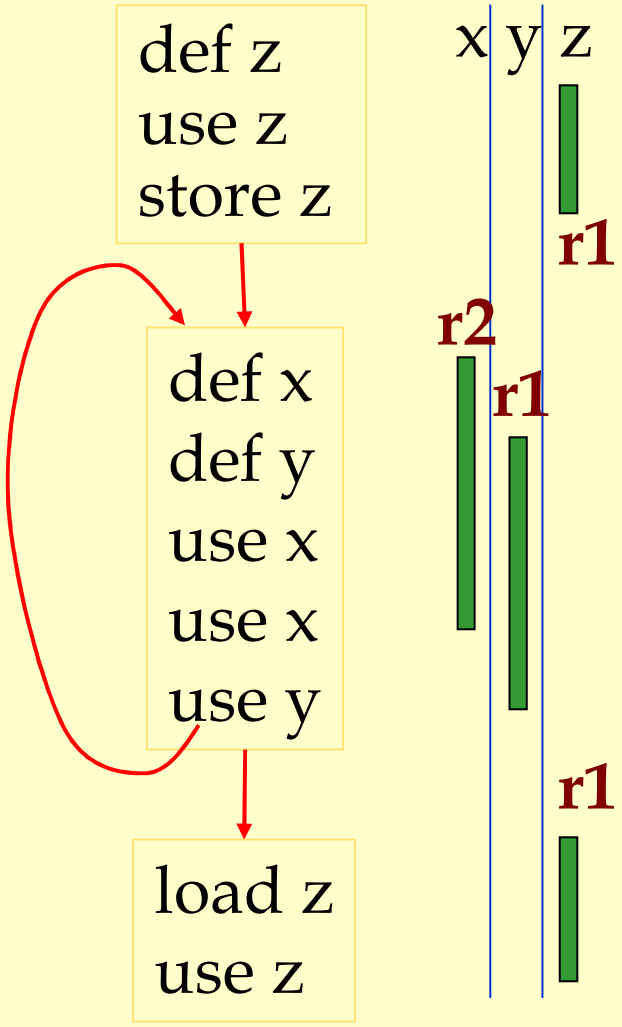
Live-Range Splitting Example

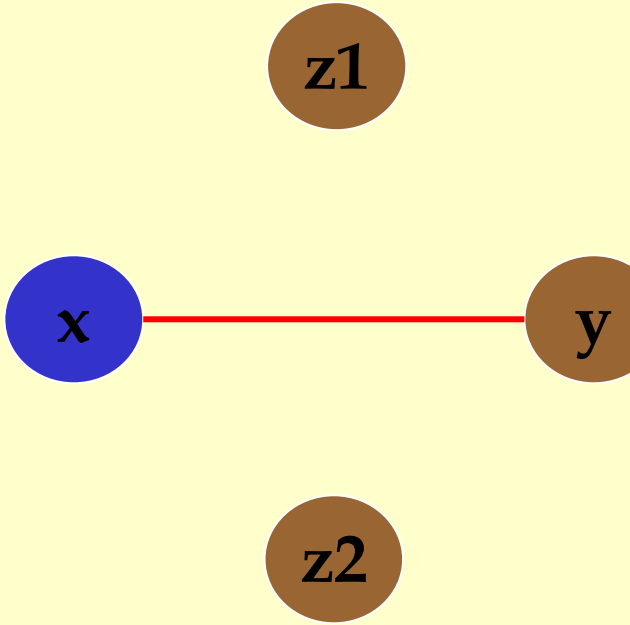




2 colorable?
YES!

Live-Range Splitting Example





2 colorable?
YES!

Live-Range Splitting Heuristic

- Identify a program point where the graph is not N-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 at the corresponding edge.
  - Redo the interference graph.
  - Try to re-color the graph.

Cost and Benefit of Splitting

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

Further Optimizations

- Register coalescing.
- Register targeting (pre-coloring).
- Pre-splitting of webs.
- Interprocedural 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 a given time:
  - First n arguments to a function.
  - The return value.
- Pre-color those webs and bind them to the right 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).

Linear Scan

- Fast (linear) allocation of registers.
- Allocates “live intervals”.
- Algorithm:
  - Calculate liveness.
  - Linearize the code.
  - Calculate the interval where the temp is live.
  - Go through the intervals sorted on startpoint.
    - Try to find a free reg for the interval.
    - Move intervals that end where this interval starts: move them to final list and free their regs.

Interprocedural 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 interprocedural register allocation.

Summary

- The goal of register allocation is to speed up the program by keeping values in registers.
- Usually gives a big impact on performance.
- The most commonly used method is some form of heuristic graph coloring.
- There exists many other methods.