Register Allocation

•A major benefit of higher level language is that the programmer is relieved of assigning storage locations to values the program is processing

•Every processor provides a set of registers with fast access

• Number of such registers is normally from 8 to 32

•Can we create software which can help us do less work?

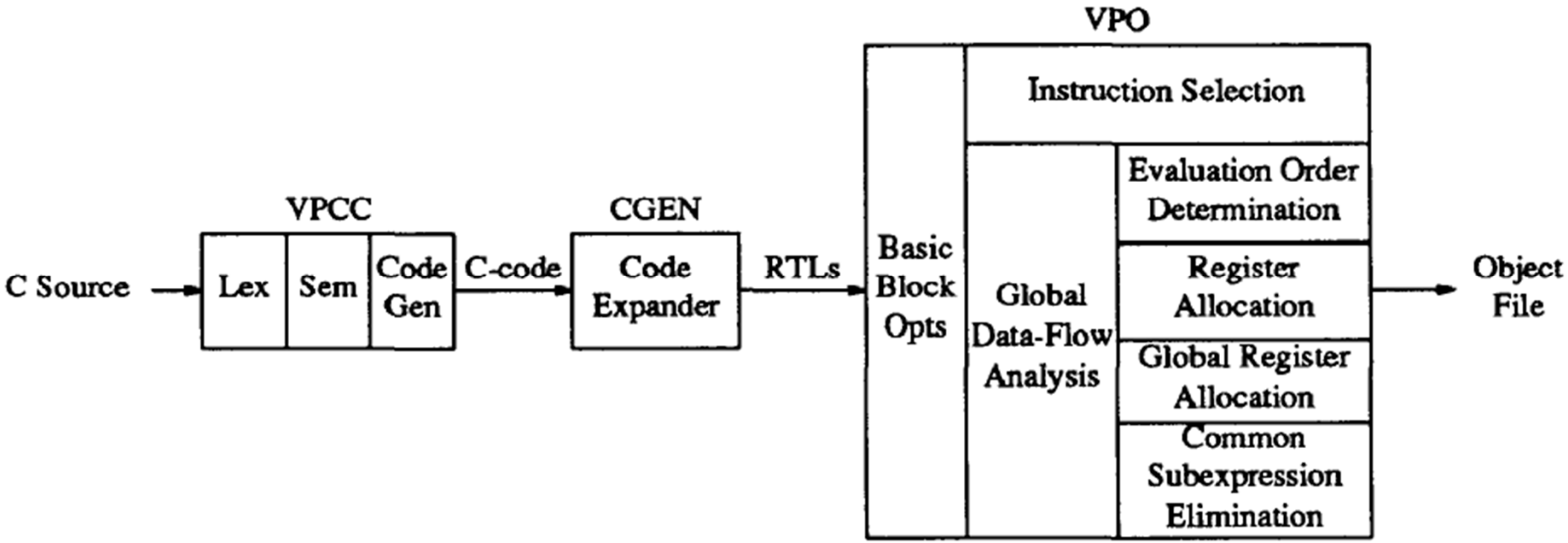
•What kind of problems we end up due to this attitude?

"See all the trouble you started?"
"But I didn't think-" (Alice)
"That's just it. If you don't think, then you shouldn't talk. "

Memory Hierarchy and speed

Memory System	Typical Capacity	Typical Access Time
Registers	1kB	1ns
Cache	1 MB	2 ns
Main Memory	8MB – 16GB	10ns
Disk drive	.1GB to 300GB	10ms

Register Allocation Problem

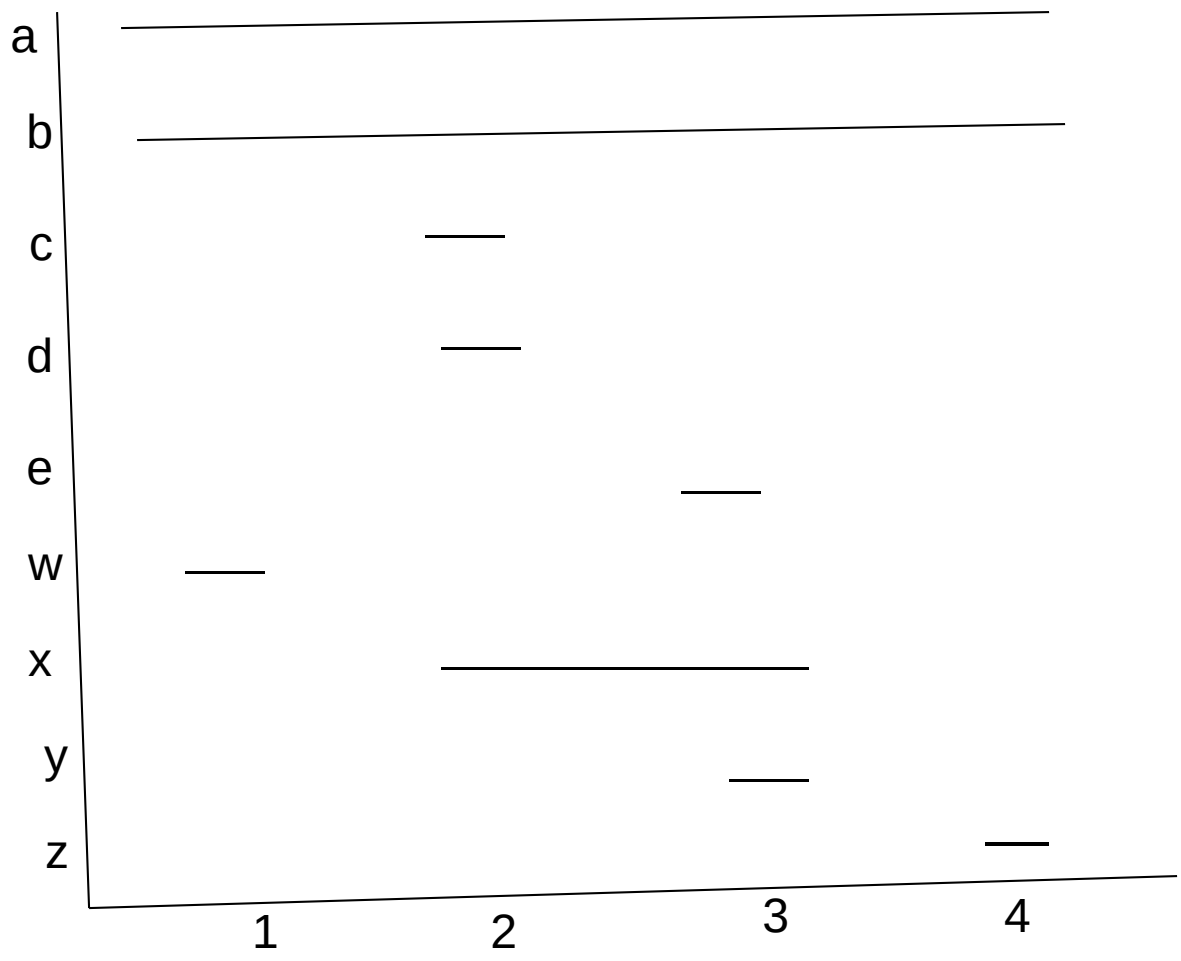


A Portable Global Optimizer and Linker? Manuel E. Benitez , Jack W. Davidson, Department of Computer Science, University of Virginia Charlottesville, VA 22903

Give a Good Algorithm for Register allocation

- A Simple program

- w = a+b
- x = c+ d
- y = x + e
- z = a - b

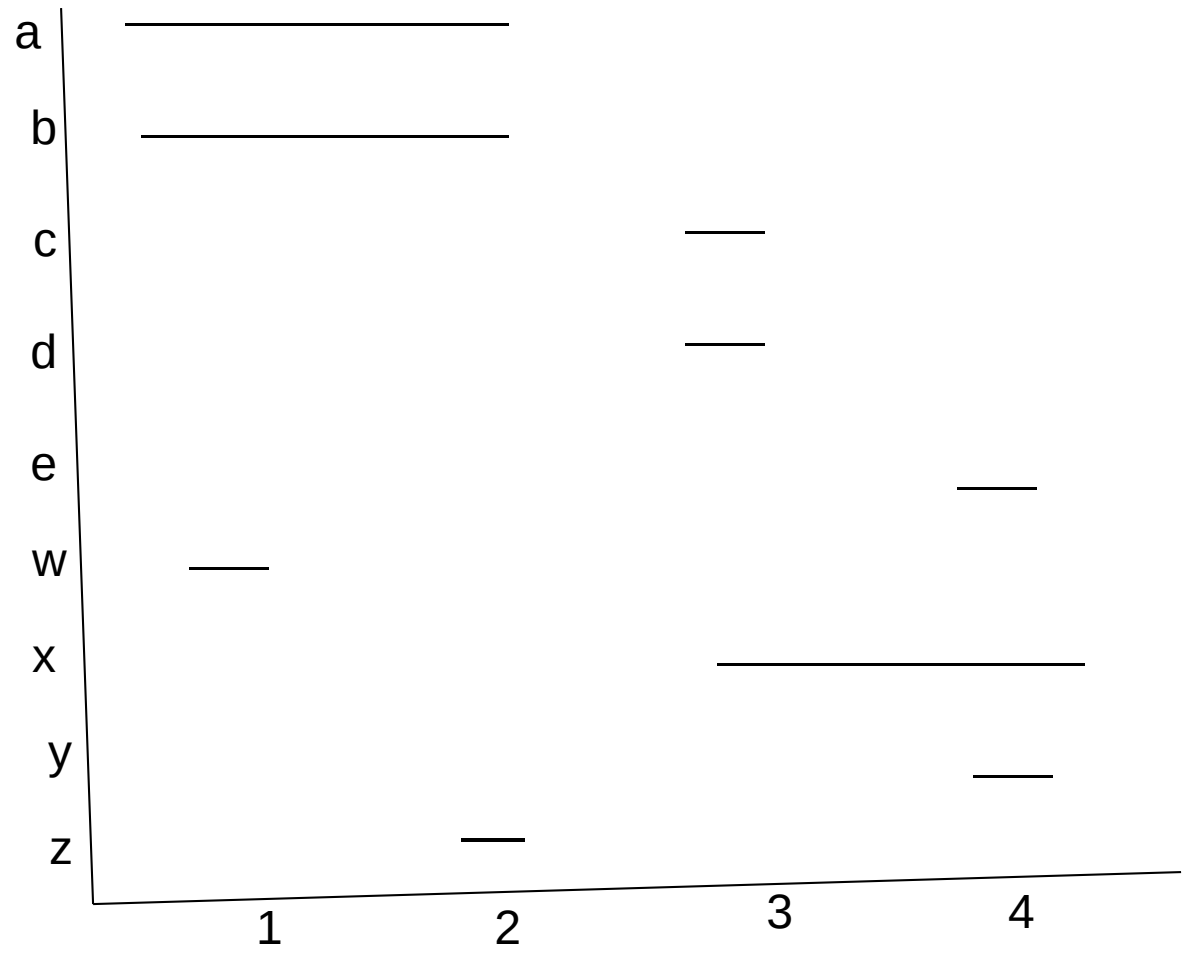


We need 5 Registers

Give a Good Algorithm for Register allocation

- A Changed program
- w = a+b
- z = a - b
- x = c+ d
- y = x + e
-

We need 3 Registers



Register Allocation

- Register allocation is the process of multiplexing a huge number of target program variables onto a small number of on chip registers.
- The ultimate goal is to keep as many operands as possible in registers to minimize the communications between memory and CPU, and in the meantime, maximize the execution speed of program execution.
- register allocation, which is one of the last steps in a compiler before code emission.

Problem Understanding

•Two variables need to be assigned to two different registers if they need to hold two different values at some point in the program.

•In the context of compilers we reduce this to liveness.

• A variable is said to be live at a given program point if it will be used in the remainder of the computation.

• We will not be able to able to accurately predict at compile time whether this will be the case.

• Therefore, try to approximate liveness through some form of dataflow analysis

•This question is undecidable in general for programs with loops

Interference Graph

- The nodes of the interference graph are the variables and registers of the program.

- There is an undirected edge between two nodes if the corresponding variables interfere and should be assigned to different registers.

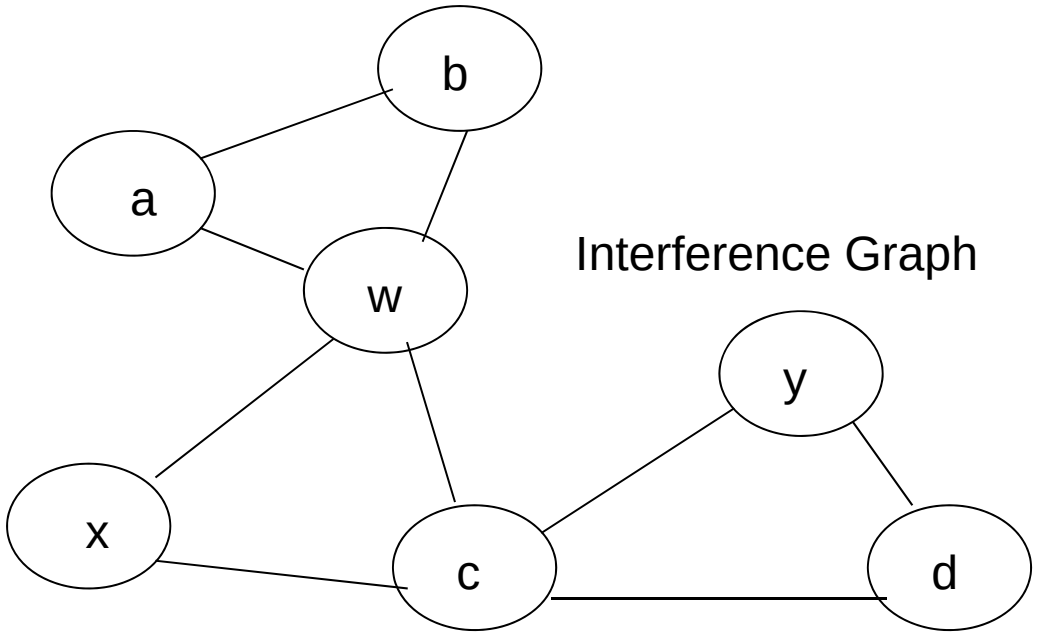
- There are never edges from a node to itself.

“Which road do I take?" (Alice)
"Where do you want to go?"
"I don't know," Alice answered.
"Then, said the cat, it doesn't matter.”

Register Allocation : Example

A Simple program

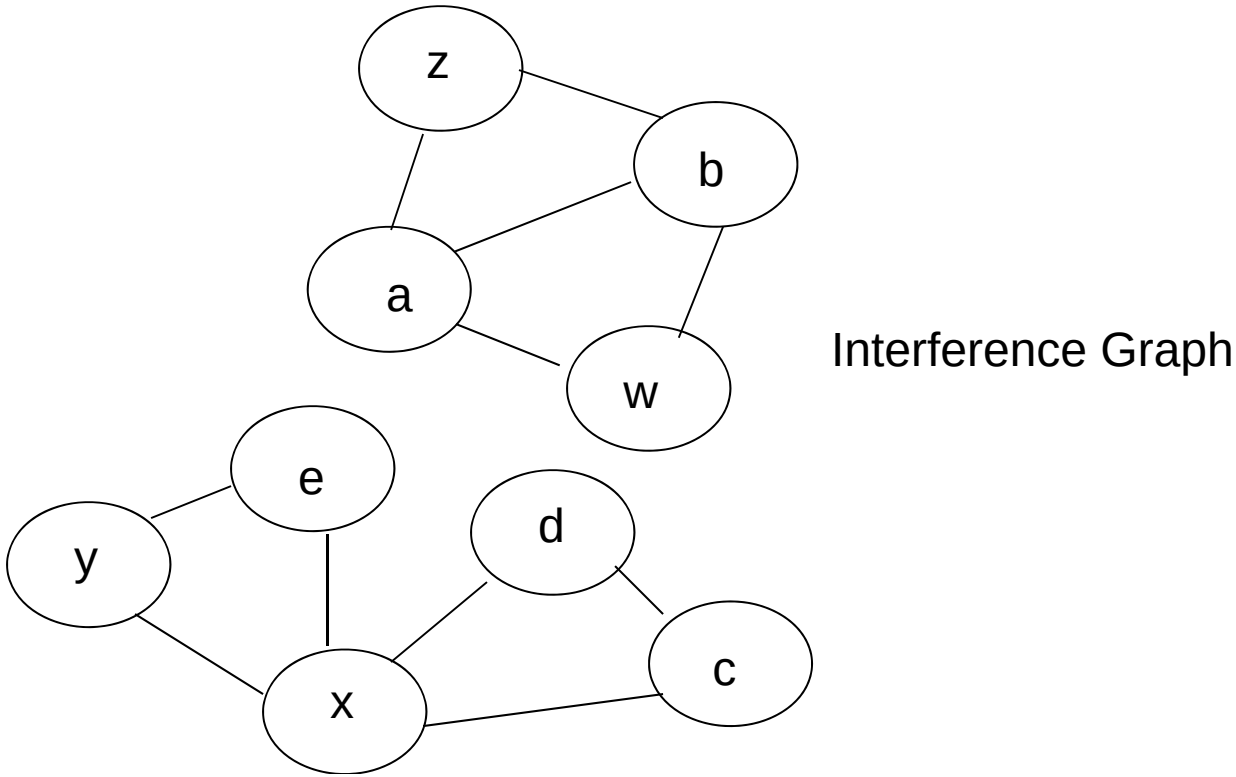
w = a+b
x = c+ w
y = c + d



Register Allocation : Example

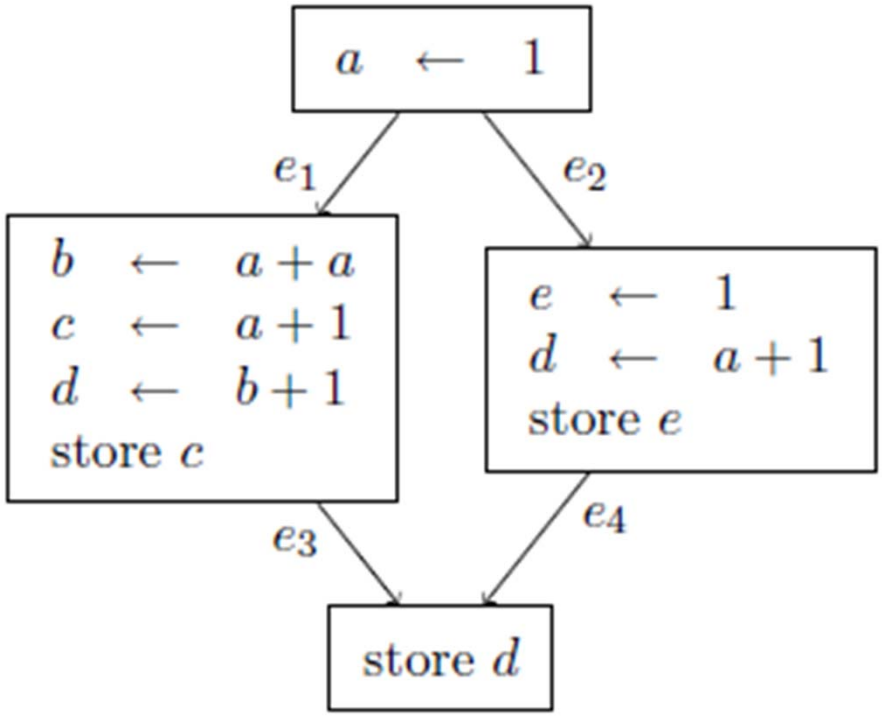
A Simple program

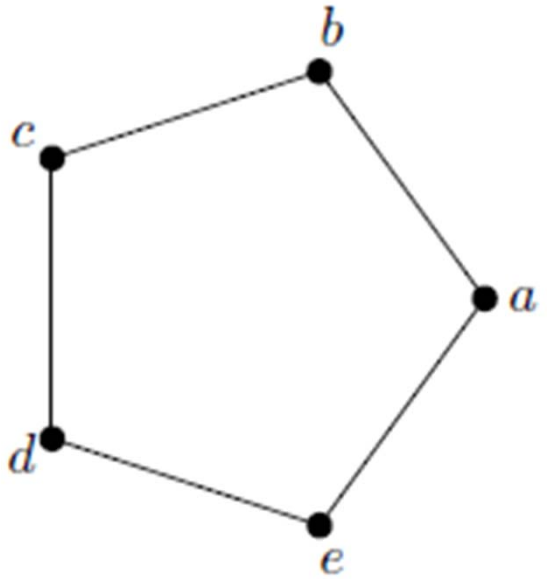
w = a+b
x = c+ d
y = x + e
z = a - b



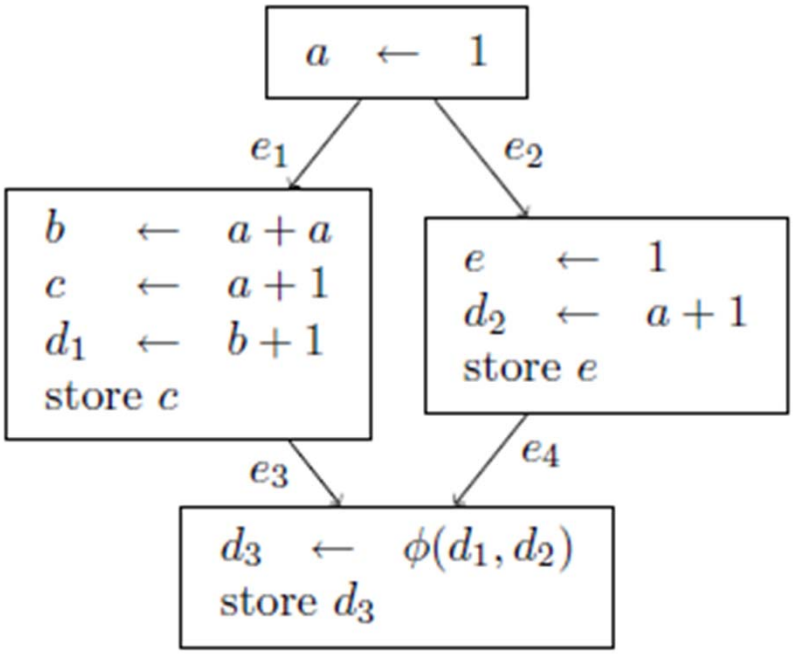
Interference Graph

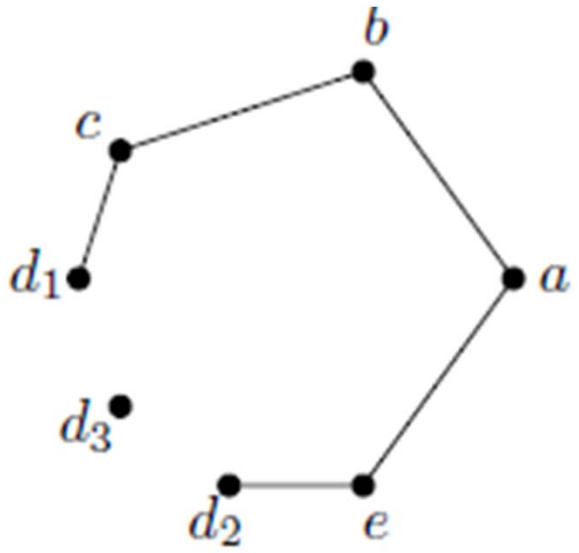
Draw the Interference Graph





Draw the Interference Graph

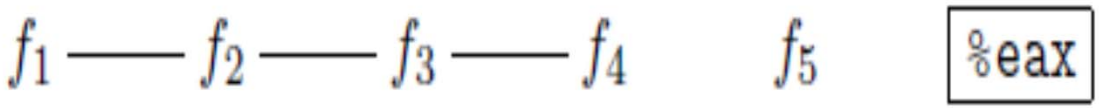




Straight line computation of Fibonacci numbers

f_1	$\leftarrow$	1	.
f_2	$\leftarrow$	1	f_1
f_3	$\leftarrow$	$f_2 + f_1$	f_2, f_1
f_4	$\leftarrow$	$f_3 + f_2$	f_3, f_2
f_5	$\leftarrow$	$f_4 + f_3$	f_4, f_3
`%eax`	$\leftarrow$	f_5	f_5
			`%eax`

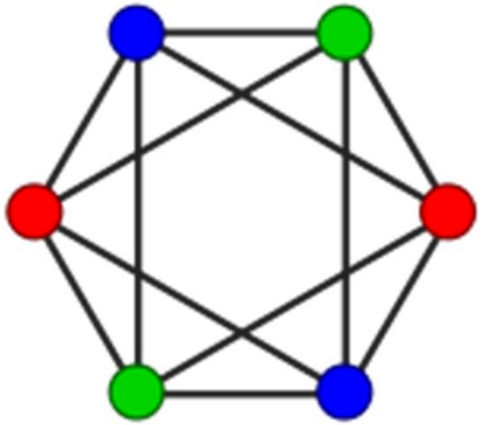
Interference Graph for the example



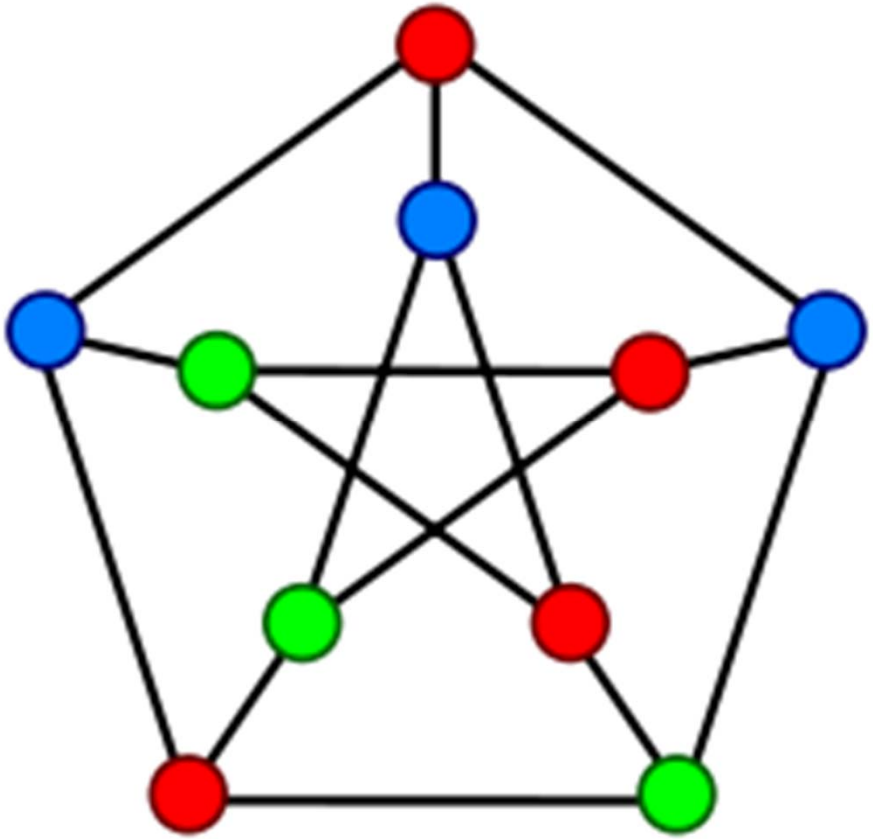
- The register %eax is special, because, as a register, it is already predefined and cannot be arbitrarily assigned to another register. Special care must be taken with predefined registers during register allocation

Register Allocation via Graph Colouring

Once we have constructed the interference graph, we can pose the register allocation problem as follows: construct an assignment of K colours (representing K registers) to the nodes of the graph (representing variables) such that no two connected nodes are of the same colour. If no such colouring exists, then we have to save some variables to memory which is called spilling.

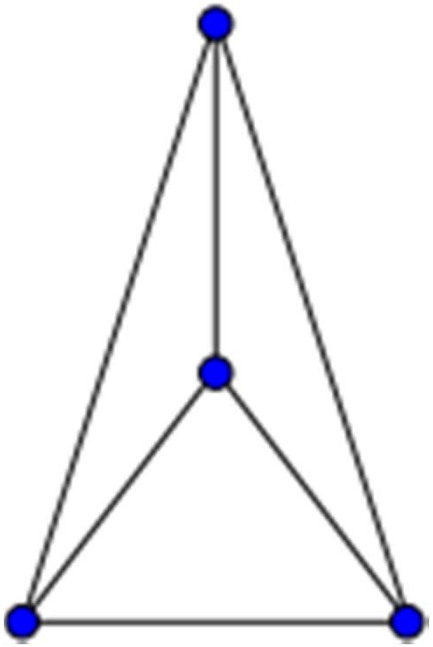


Register Allocation via Graph Colouring

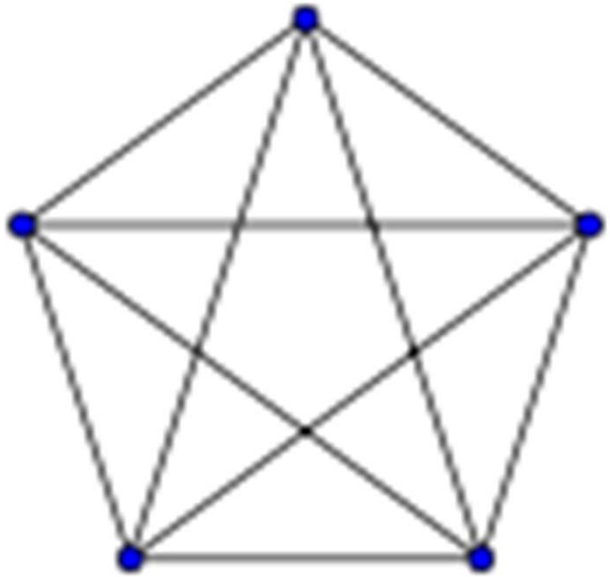


Planar and Nonplanar Graphs

In graph theory, a **planar graph** is a graph that can be embedded in the plane, i.e., it can be drawn on the plane in such a way that its edges intersect only at their endpoints. In other words, it can be drawn in such a way that no edges cross each other



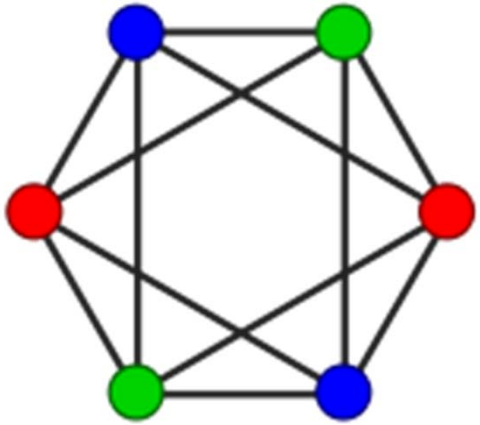
Planar Graph



Non planar Graph

Graph Colouring Problem

In its simplest form, it is a way of coloring the vertices of a graph such that no two adjacent vertices share the same color



“There is no use trying; one can't believe impossible things." (Alice)
"I dare say you haven't had much practice. When I was your age,
I always did it for half an hour a day. Why, sometimes I've believed
as many as six impossible things before breakfast.” -Queen

Register Allocation is NP-Complete

•Unfortunately, the problem whether an arbitrary graph is K-colourable is NP-complete for K>= 3.

•Chaitin (1982) has proved that register allocation is also NP-complete by showing that for any graph G there exists some program which has G as its interference graph.

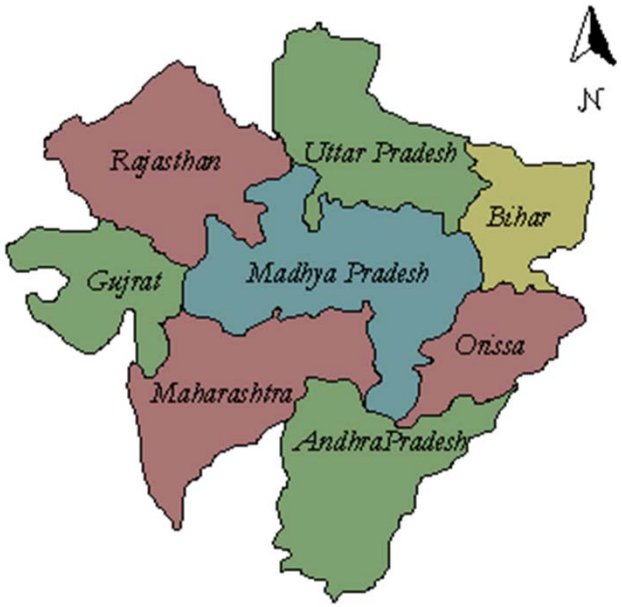
• In other words, one cannot hope for a theoretically optimal and efficient register allocation algorithm that works on all machine programs.

Graph Colouring Problem: Four color theorem

Given any separation of a plane into contiguous regions, producing a figure called a *map*, no more than four colors are required to color the regions of the map so that no two adjacent regions have the same color.

The four color theorem was proven in 1976 by Kenneth Appel and Wolfgang Haken. It was the first major theorem to be proved using a computer. Appel and Haken's approach started by showing that there is a particular set of 1,936 maps, each of which cannot be part of a smallest-sized counterexample to the four color theorem. Appel and Haken used a special-purpose computer program to confirm that each of these maps had this property. In 2005, the theorem was proven by Georges Gonthier with general purpose theorem proving program

Graph Colouring Problem: Four color theorem



Linear scan Algorithm for Register allocation

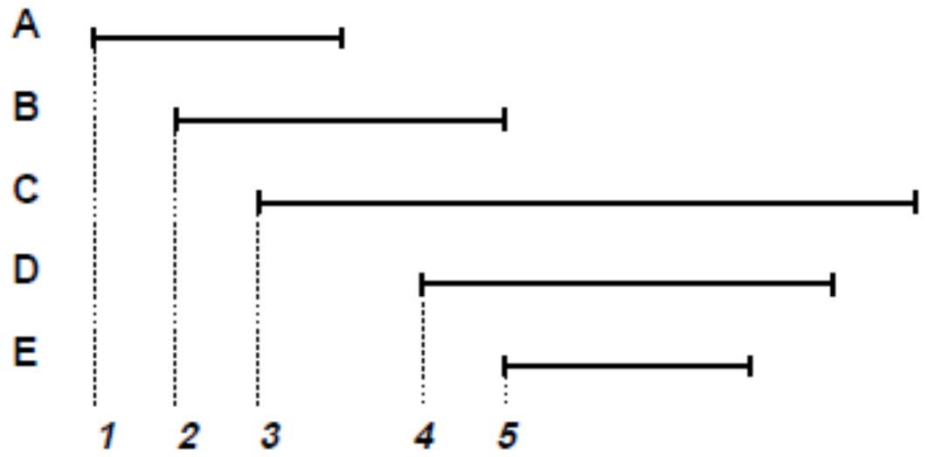
Given live variable information (obtained, for example, via data-flow analysis, live intervals can be computed easily with one pass through the intermediate representation.

Interference among live intervals is captured by whether or not they overlap. Given *R available registers and a list of live intervals, the* linear scan algorithm must allocate registers to as many intervals as possible, such that, no two overlapping live intervals are allocated to the same register.

If *n > R live intervals overlap at any point, then at least n − R of them must reside* in memory.

Linear Scan Register Allocation MASSIMILIANO POLETTO Laboratory for Computer Science, MIT and VIVEK SARKAR IBM Thomas J. Watson Research Centre

Linear scan Algorithm for Register allocation



Letters on the left are variable names; the corresponding live intervals appear to the right.

Linear scan Algorithm: Pathological Examples

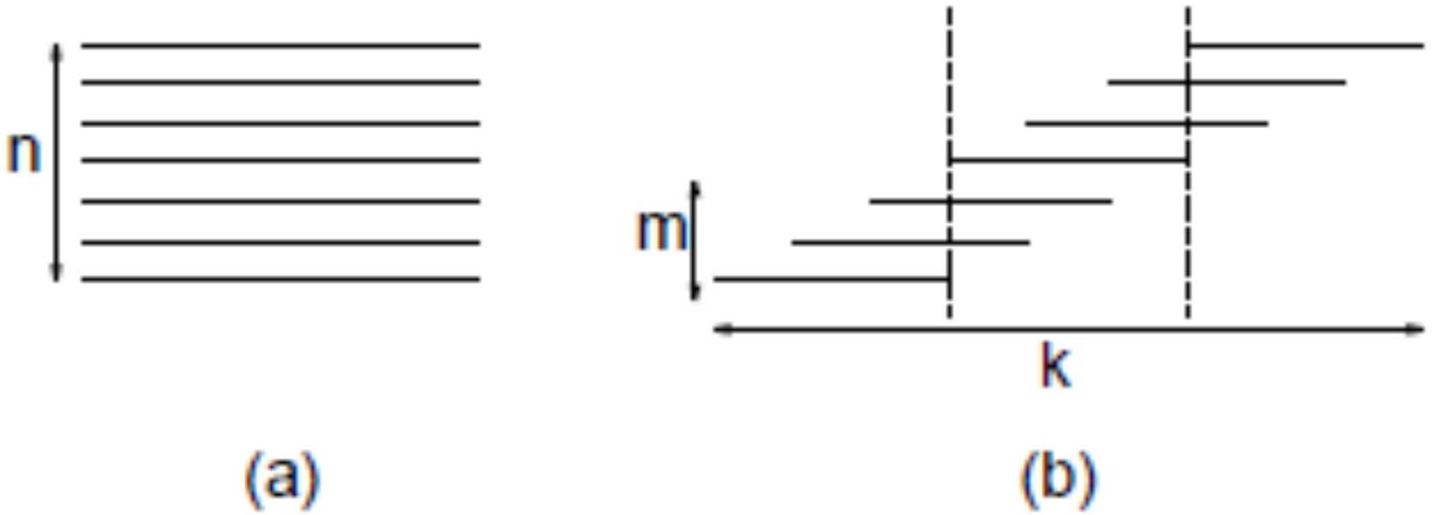
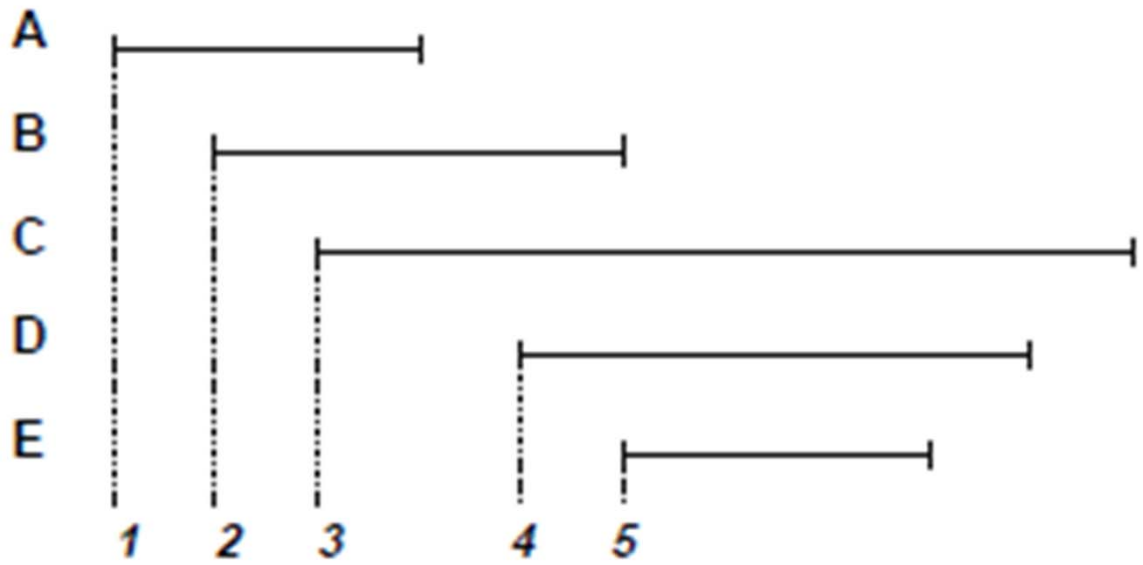


Fig. 4. Two types of pathological programs.

Linear scan Algorithm: An Example



Linear scan Algorithm: An Example

Consider the previous example. Say number of available registers is *R = 2.*

The algorithm performs allocation decisions 5 times,
once per live interval, as denoted by the italicized numbers at the bottom of the figure.

By the end of step *2, active = (A;B) and both A and B are therefore in* registers.

At step *3, three live intervals overlap, so one variable must be spilled.*
The algorithm therefore spills *C, the one whose interval ends furthest away from* the current point, and does not change *active.*

As a result, at step 4, A is expired from *active, making a register available for D, and at step 5, B is expired, making* a register available for *E. Thus, in the end, C is the only variable not allocated to* a register.

Had the algorithm not spilled the longest interval, *C, at step 3, both*
one of *A and B and one of D and E would have been spilled to memory.*

Linear scan Algorithm for Register allocation

- Not based on graph colouring,

- Allocates registers to variables in a single linear-time scan of the variables' live ranges.

- The linear scan algorithm is considerably faster than algorithms based on graph colouring.

- Is simple to implement, and results in code that is almost as efficient as that obtained using more complex and time-consuming register allocators based on graph colouring.

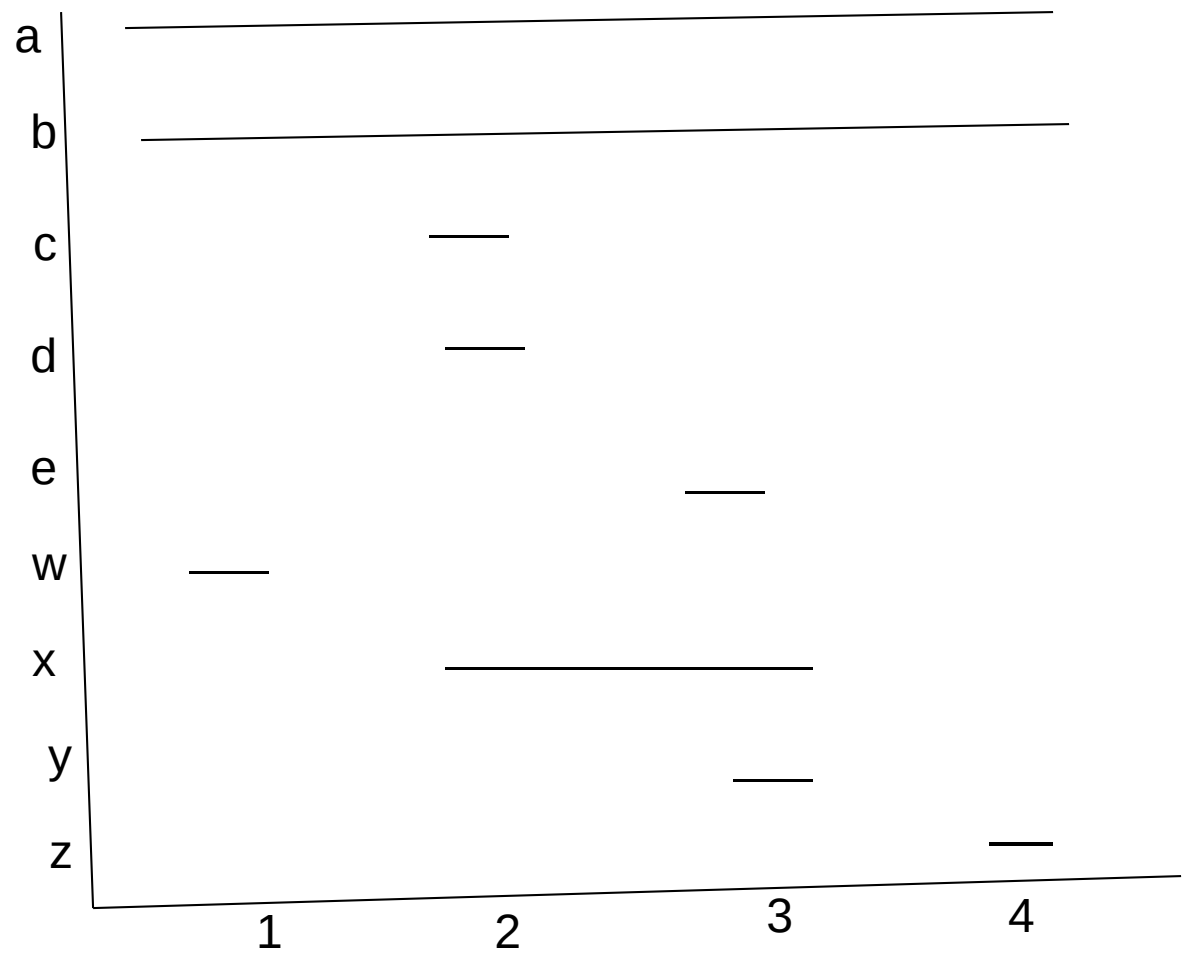
- The algorithm is of interest in applications where compile time is a concern, such as dynamic compilation systems, \just-in-time" compilers, and interactive development environments.

Linear scan Algorithm for Register allocation

- A Simple program

- w = a+b
- x = c+ d
- y = x + e
- z = a - b

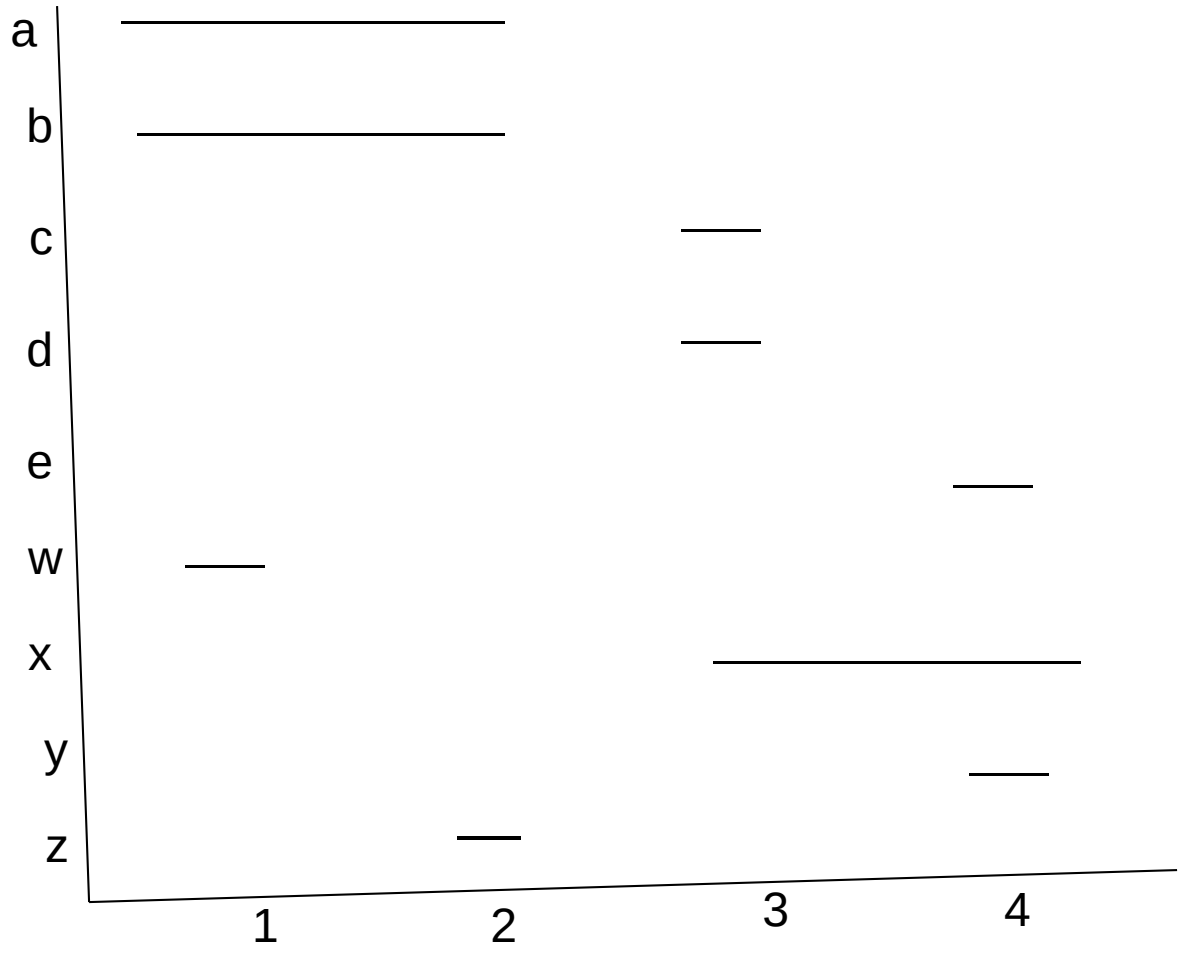
We need 5 Registers



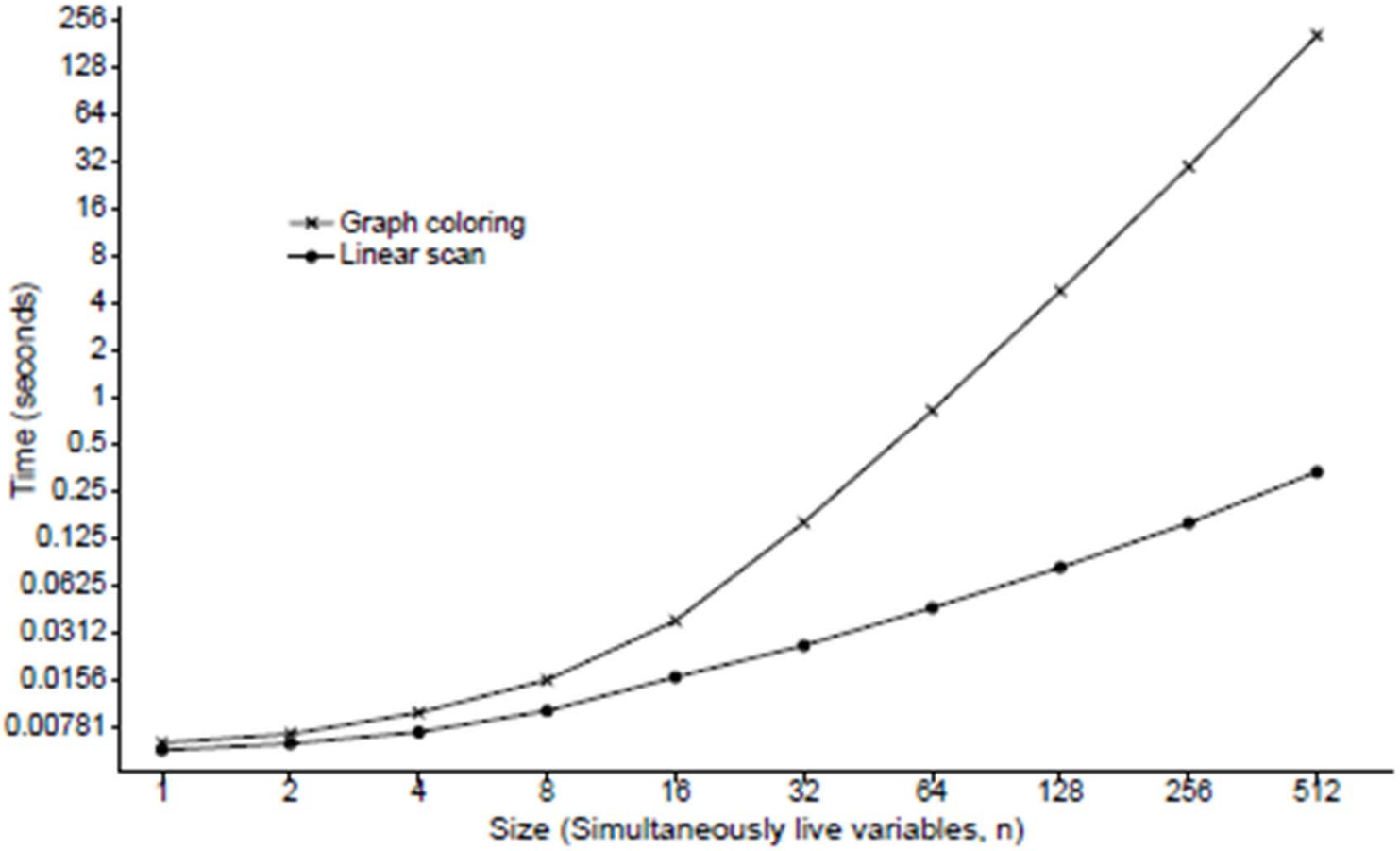
Linear scan Algorithm for Register allocation

- A Changed program
- w = a+b
- z = a - b
- x = c+ d
- y = x + e
-

We need 3 Registers



Speed of Linear scan vs Graph coloring



References

1. Lecture Notes on Register Allocation 15-411: Compiler Design
Frank Pfenning Lecture 3 September 1, 2009

2. A Portable Global Optimizer and Linker? Manuel E. Benitez , Jack W. Davidson, Department of Computer Science, University of Virginia Charlottesville, VA 22903

3. Linear Scan Register Allocation MASSIMILIANO POLETTO
Laboratory for Computer Science, MIT and VIVEK SARKAR
IBM Thomas J. Watson Research Center