Linear Scan Register Allocation

CMPT 379: Compilers

Instructor: Anoop Sarkar

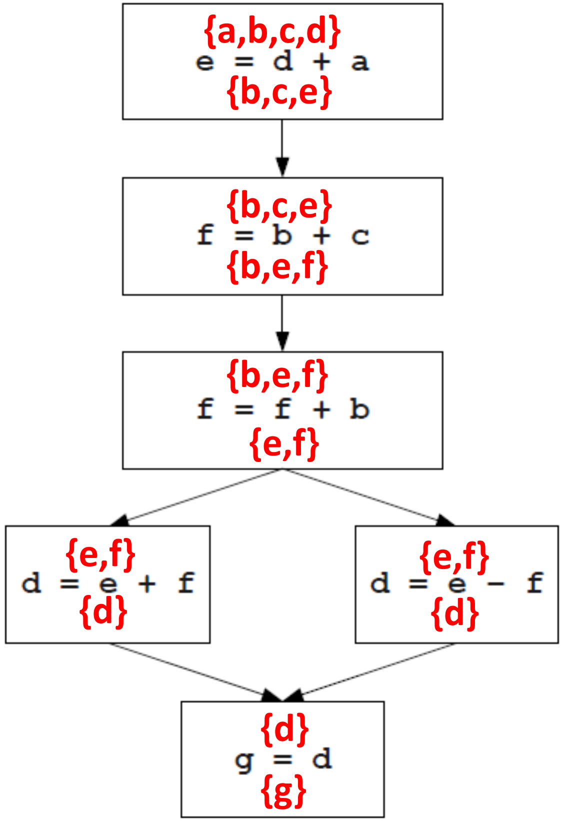
anoopsarkar.github.io/compilers-class

Live Ranges and Live Intervals

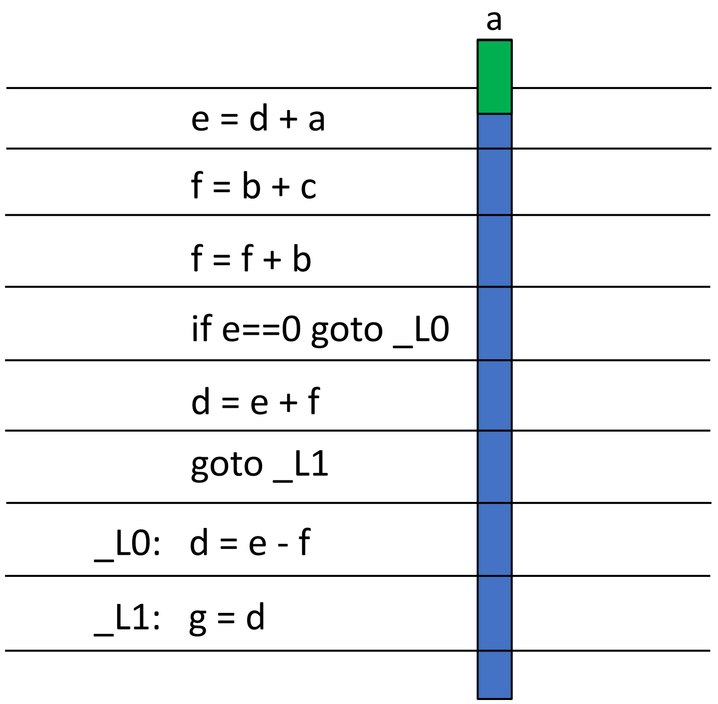
- The live range for a variable is the set of program points at which that variable is live.
- The live interval for a variable is the smallest subrange of the IR code containing all a variable's live ranges.
  - A property of the IR code, not CFG.
  - Less precise than live ranges, but simpler to work with

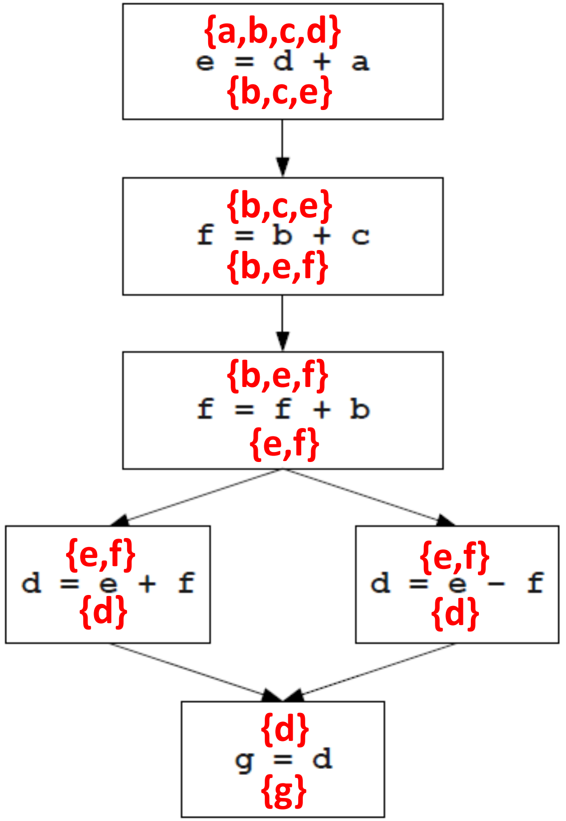
Live Intervals

```
       e = d + a
       f = b + c
       f = f + b
       if e==0 goto _L0
       d = e + f
       goto _L1
_L0:   d = e - f
_L1:   g = d
```

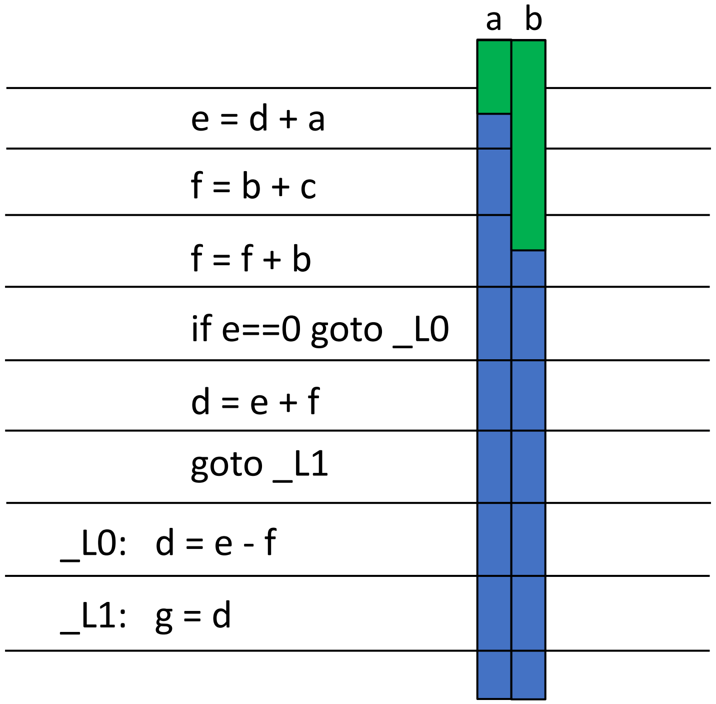


Live Intervals

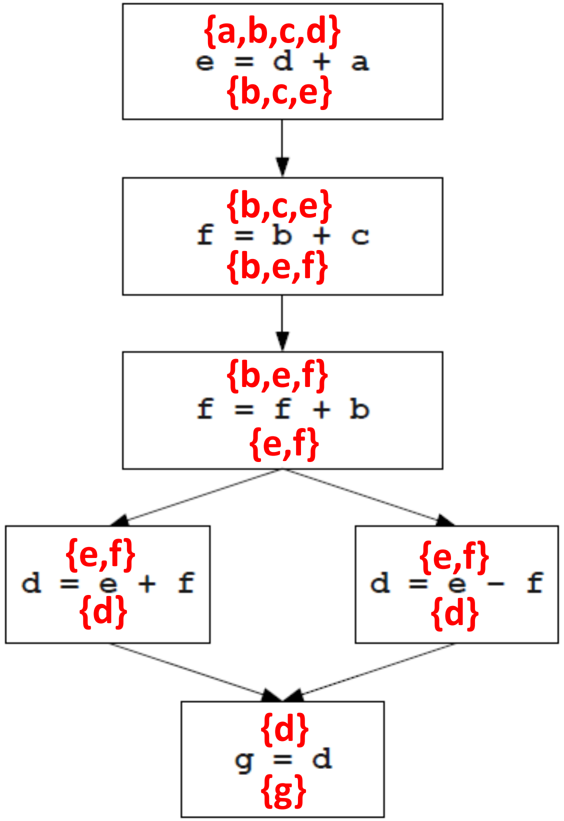




Live Intervals



```
       e = d + a
       f = b + c
       f = f + b
       if e==0 goto _L0
       d = e + f
       goto _L1
_L0:   d = e - f
_L1:   g = d
```



```
{a,b,c,d}
e = d + a
{b,c,e}
```

```
{b,c,e}
f = b + c
{b,e,f}
```

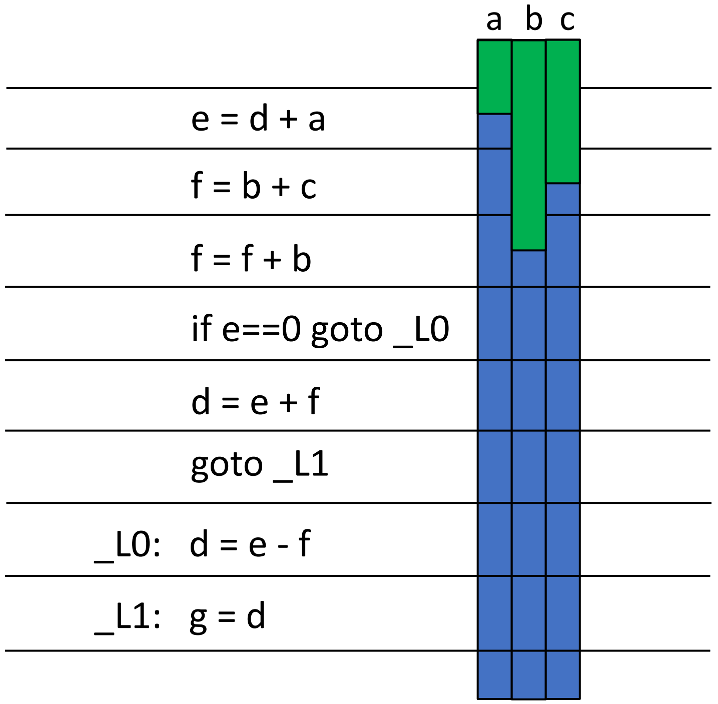
```
{b,e,f}
f = f + b
{e,f}
```

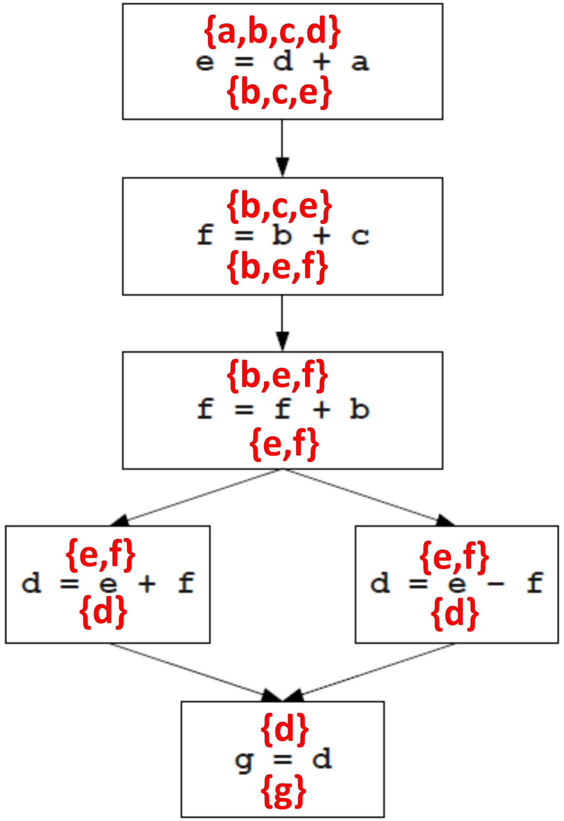
```
{e,f}
d = e + f
{d}
```

```
{e,f}
d = e - f
{d}
```

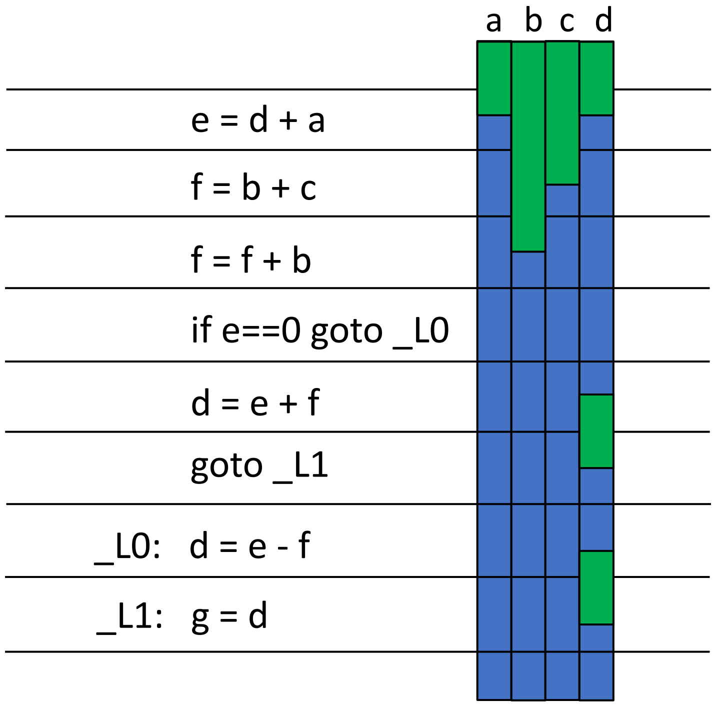
```
{d}
g = d
{g}
```

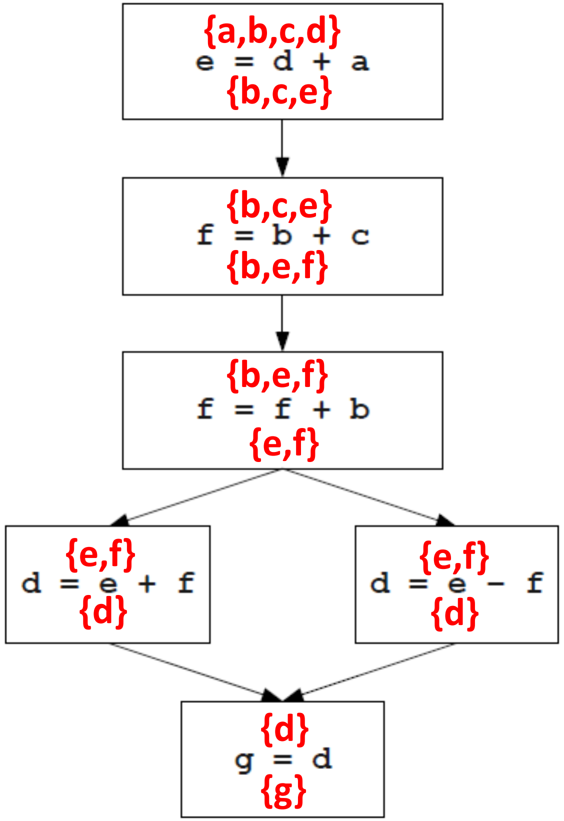
Live Intervals



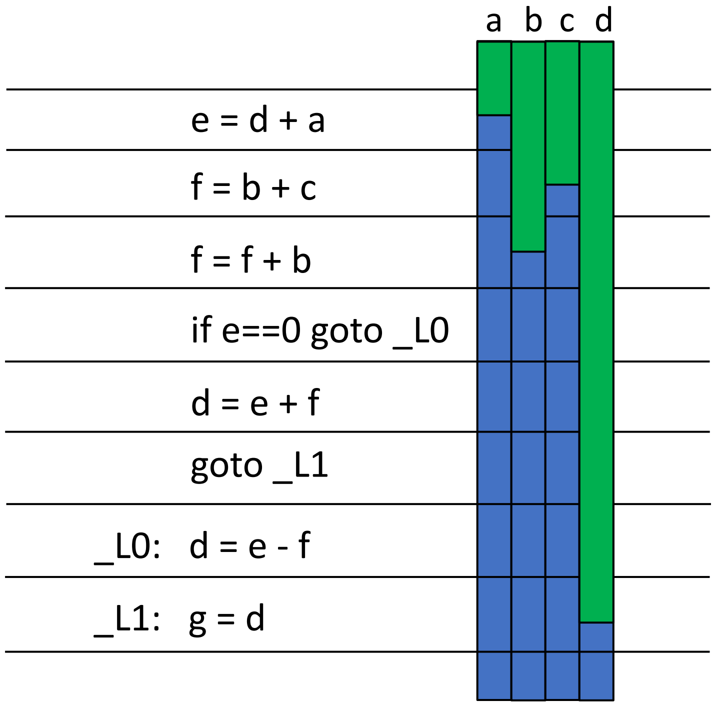


Live Intervals

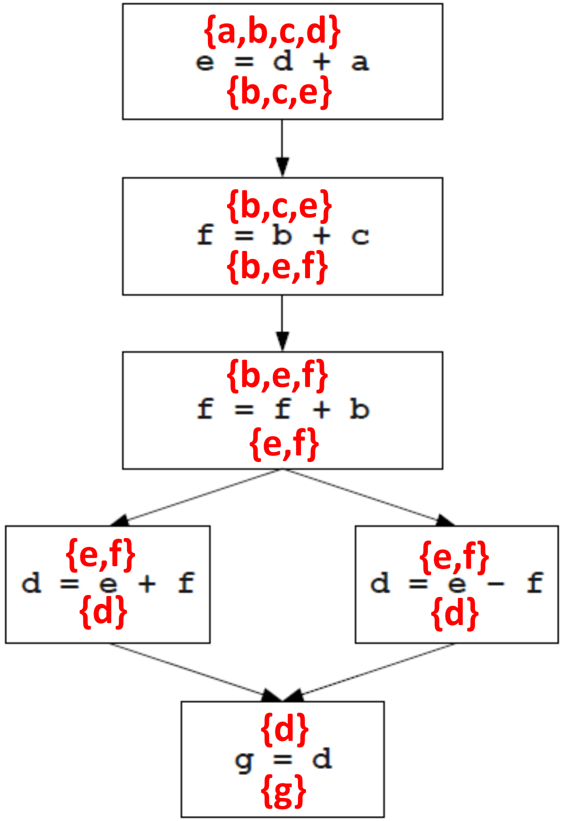




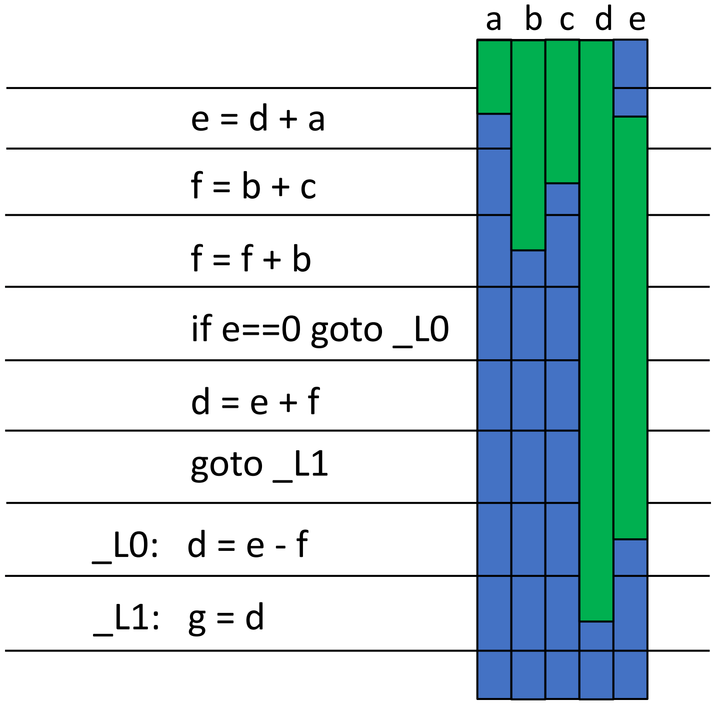
Live Intervals

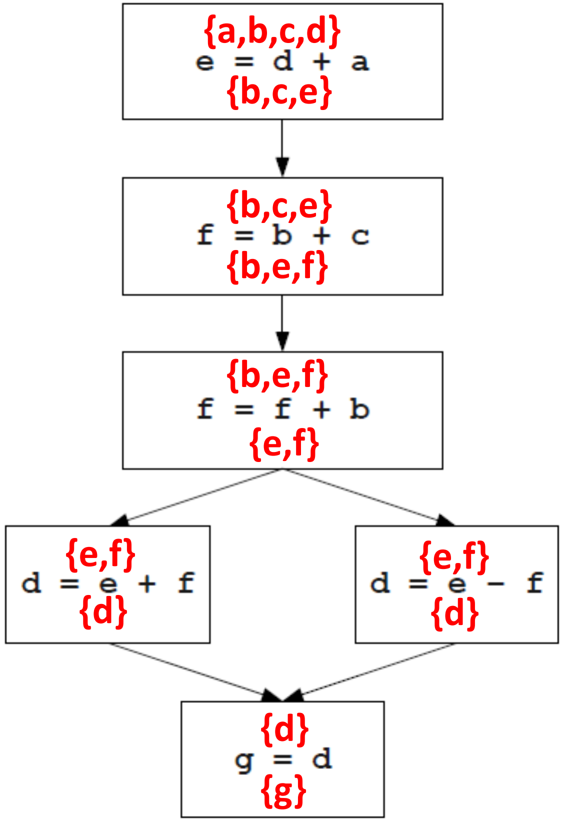


```
        e = d + a
        f = b + c
        f = f + b
        if e==0 goto _L0
        d = e + f
        goto _L1
_L0:    d = e - f
_L1:    g = d
```

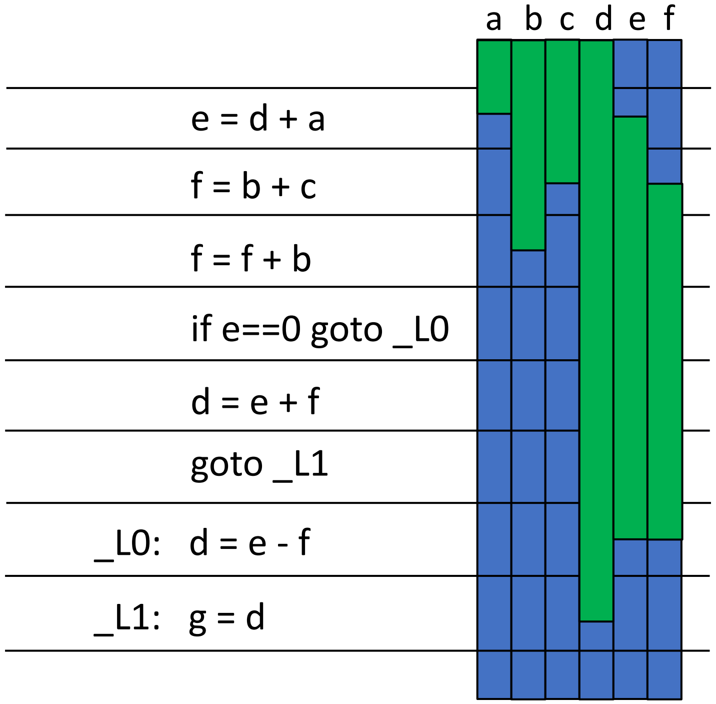


Live Intervals

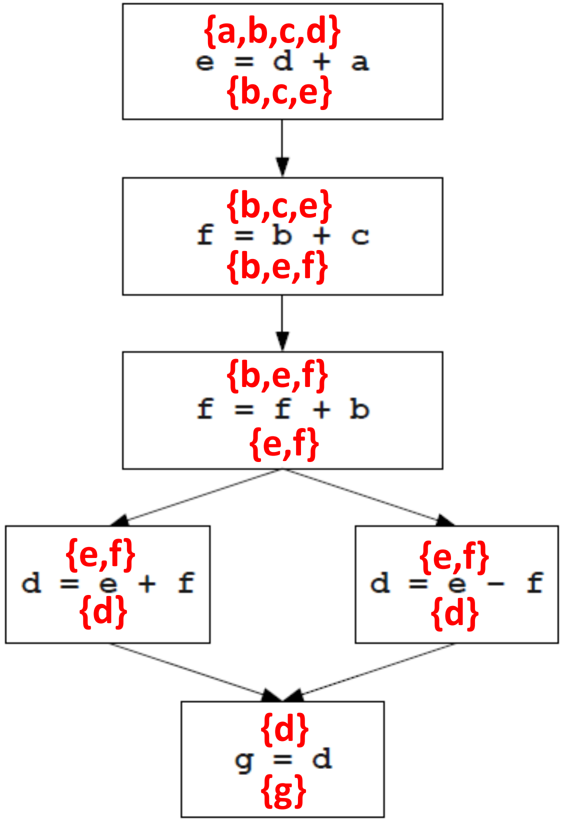




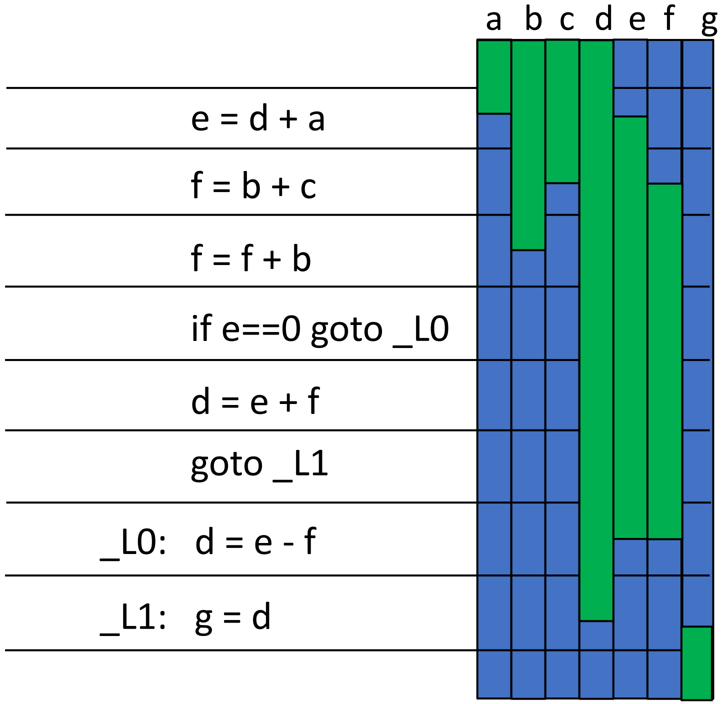
Live Intervals

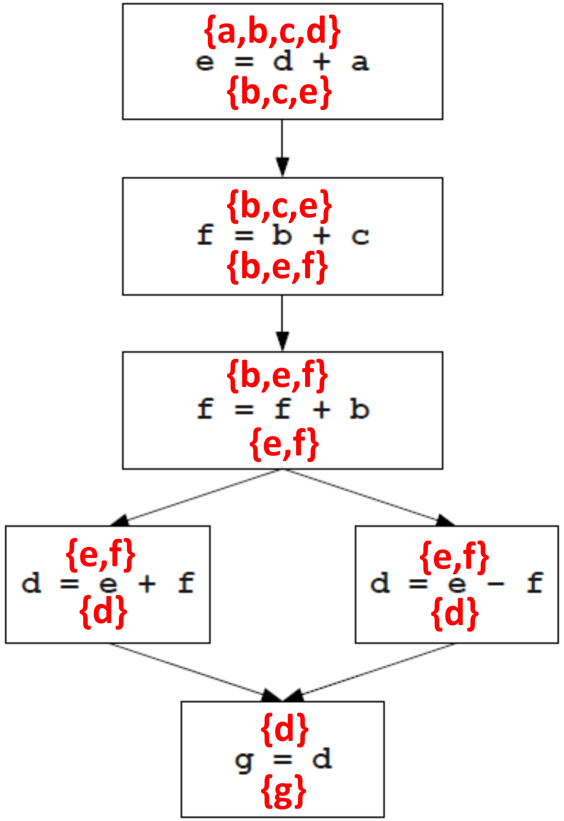


```
      e = d + a
      f = b + c
      f = f + b
      if e==0 goto _L0
      d = e + f
      goto _L1
_L0:  d = e - f
_L1:  g = d
```



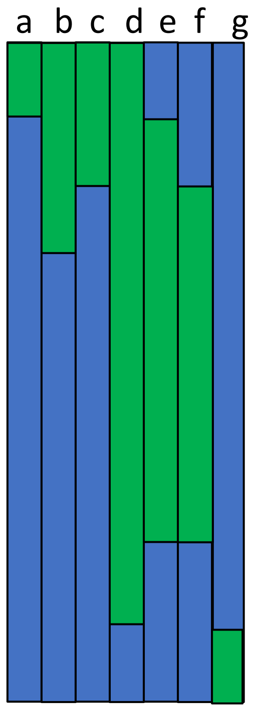
Live Intervals



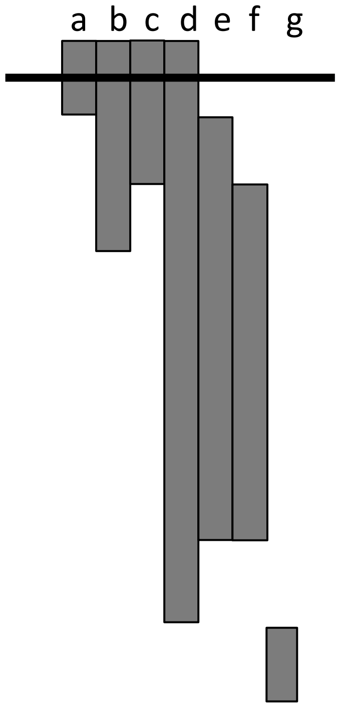


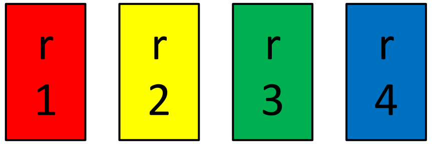
Register Allocation with Live Intervals

- Given the live intervals for all the variables in the program, we can allocate registers using a simple greedy algorithm.
- Idea: Track which registers are free at each point.
- When a live interval begins, give that variable a free register.
- When a live interval ends, the register is once again free.

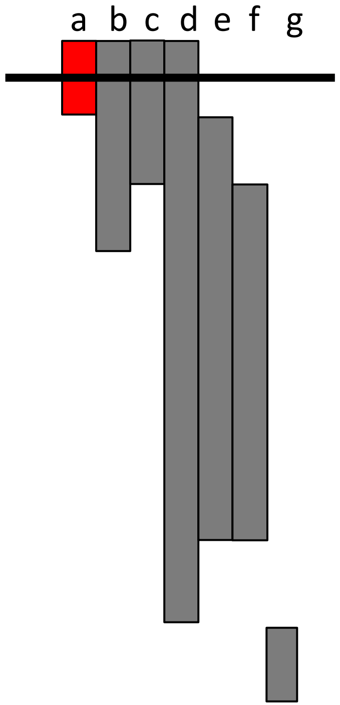


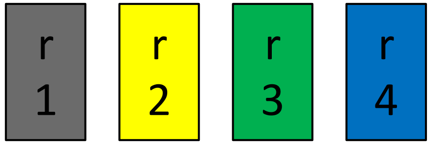
Register Allocation with Live Intervals



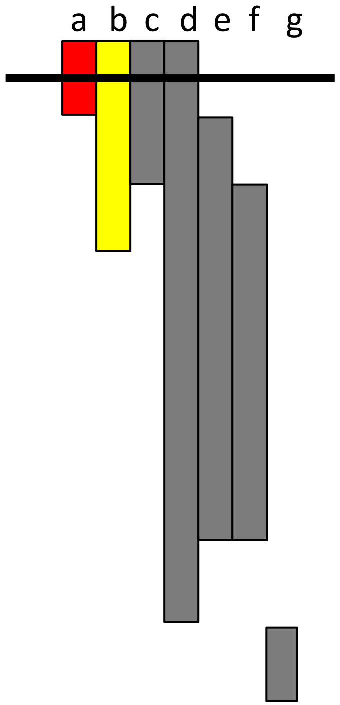


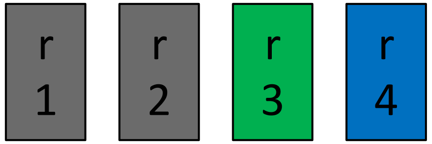
Register Allocation with Live Intervals



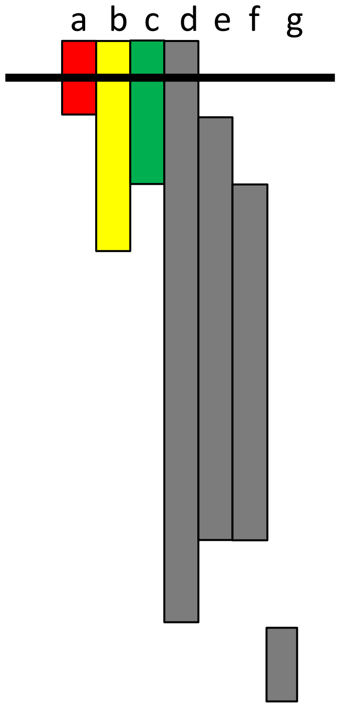


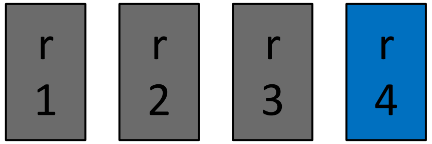
Register Allocation with Live Intervals



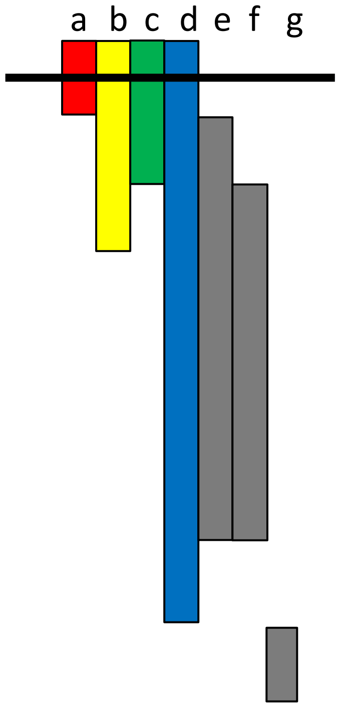


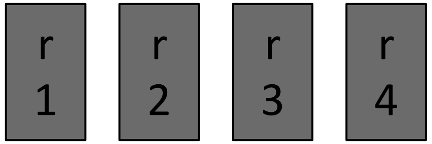
Register Allocation with Live Intervals



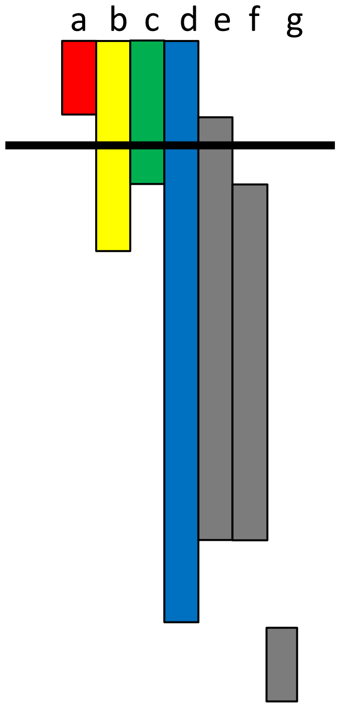


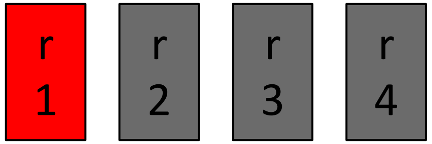
Register Allocation with Live Intervals



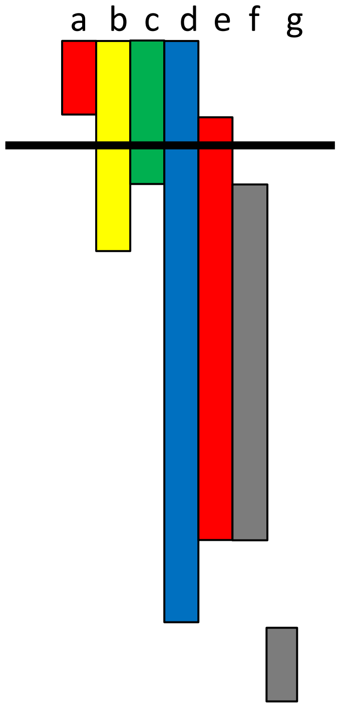


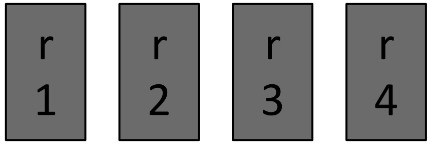
Register Allocation with Live Intervals



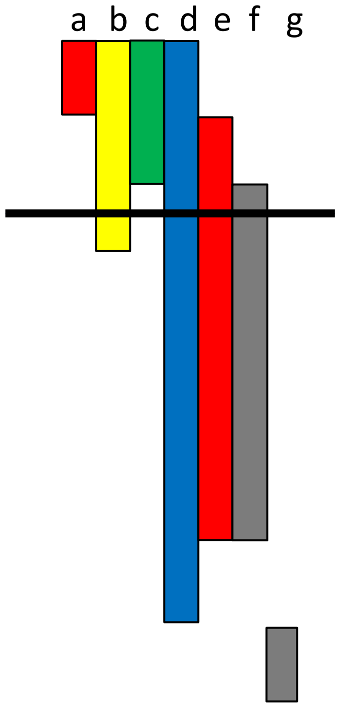


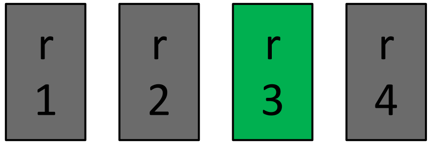
Register Allocation with Live Intervals



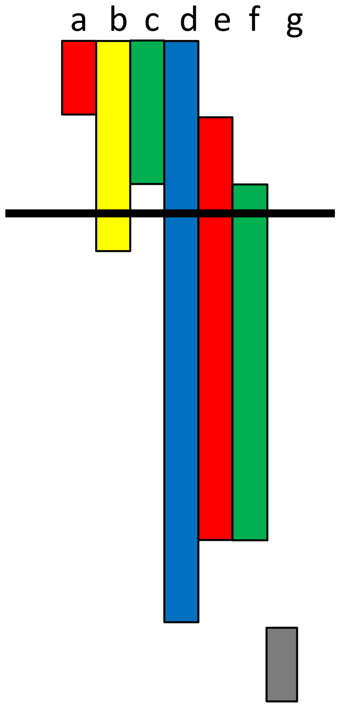


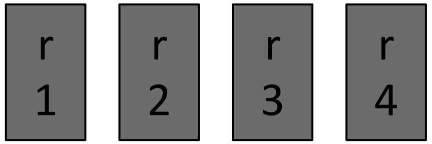
Register Allocation with Live Intervals



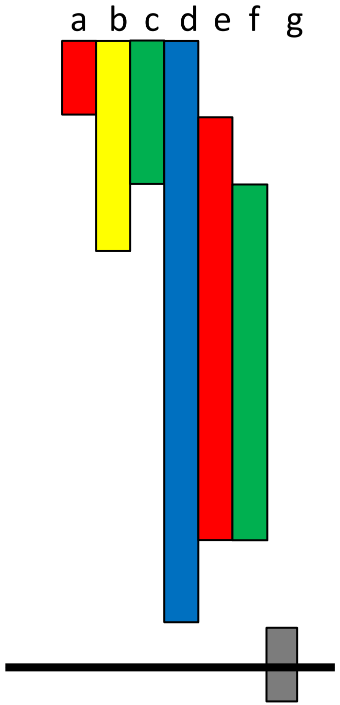


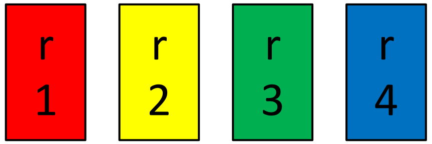
Register Allocation with Live Intervals



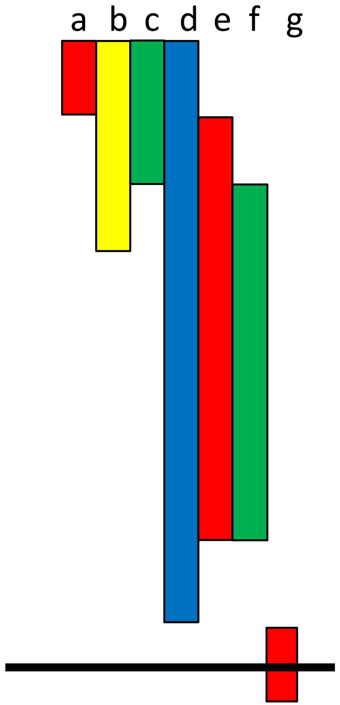


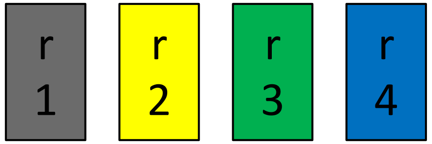
Register Allocation with Live Intervals



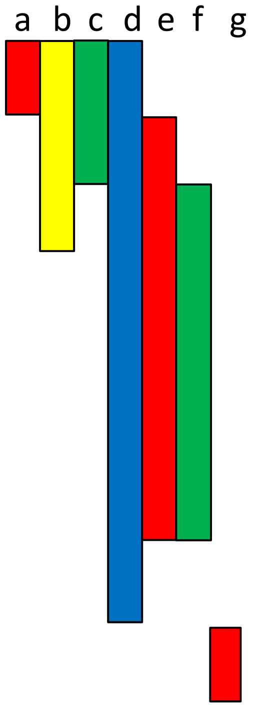


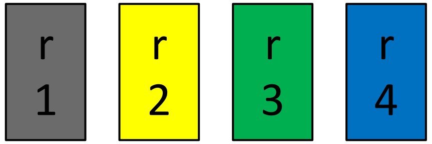
Register Allocation with Live Intervals





Register Allocation with Live Intervals





Linear Scan Register Allocation

- If a register cannot be found for a variable v, we may need to spill a variable.
- This algorithm is called linear scan register allocation
- Requires more up-front work to compute live intervals

Linear Scan Register Allocation

- Pros:
  - Very efficient
  - Works well in many cases
  - Allocation needs one pass, the code can be generated simultaneously
  - Used in JIT compilers like Java HotSpot
- Cons:
  - Produces less efficient code compared to the graph coloring approach

Summary

- Register allocation is a "must have" in compilers, because:
  - Intermediate code uses too many temporaries
  - It makes a big difference in performance
- The liveness at each location can be used for register allocation
- Register allocation as heuristic graph coloring uses live ranges
  - The basis for the technique used in GCC
- Linear scan register allocation uses live intervals
  - Often used in JIT compilers due to efficiency