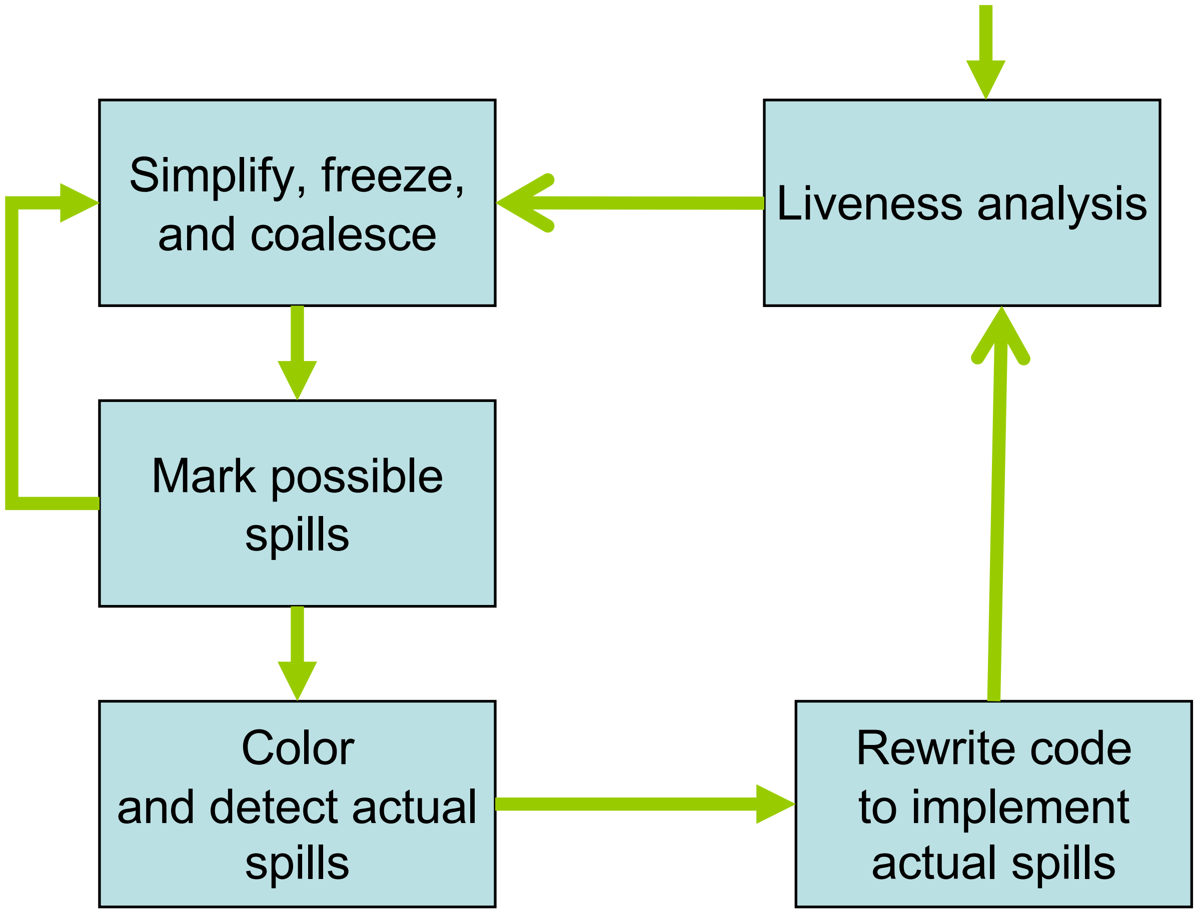
Structure of Graph Coloring Allocator


Simplify, freeze, and coalesce
Liveness analysis
Mark possible spills
Color and detect actual spills
Rewrite code to implement actual spills

Spill-Free (Global) Register Allocation

- Let's look at a somewhat more constrained version of the question.
- [SFRA] Given a set V of symbolic registers (i.e., program variables and temporaries) being used across n instructions, and k physical registers, determine whether it is possible to assign each symbolic register $s \in V$ to a physical register $\mathrm{reg}(s, P)$ at each program point $P \in \{i^+, i^-: 1 \le i \le n\}$ where s is live.
    - If so, report the register assignments, *including any register-register moves that need to be inserted*.
    - If not, report that no feasible solution exists.
- We solved this problem earlier using Kempe's heuristic.
- But the interference graph was constructed around a very coarse notion of live intervals.
    - Can we do better?

Live Intervals and Live Ranges

- Given a numbering of the n instructions, $[i, j]$ is said to be a live interval for variable v if there is no instruction with number $i' < i$ such that v is live at i', and there is no instruction with number $j' > j$ such that v is live at j'.
  - That is, $[i, j]$ is a maximal interval over which v is live.
  - So far, implicitly, this interval has been the trivial interval $[1^-, n^+]$.
- A live interval for v is a conservation approximation to the true lifetime of v, which may be a collection of live ranges along with holes.
- Intuition
  - If we can recognize the fine-grained live ranges of different variables, we may be able to use them to use the same physical register for one variable while another variable has a hole in its lifetime.
  - But we need to honor control flow constraints, so that the assignments of physical registers to variables are rendered oblivious across CFG edges. This can be accomplished with register-register moves.

Example #1 [from Sarkar & Barik 2007]

- Can this situation be handled using two physical registers without spilling?
- Graph Coloring says no.
  - The interference graph is the complete graph on $\{a, b, c\}$, and therefore not 2-colorable.
- But this is easily solvable as a SFRA problem instance, with the following assignments.
  - $\mathrm{reg}(a, [1^+, 3^-]) = r1$, $\mathrm{reg}(b, [1^+, 3^-]) = r2$.
  - $\mathrm{reg}(b, [4^+, 6^-]) = r1$, $\mathrm{reg}(c, [4^+, 6^-]) = r2$.
  - $\mathrm{reg}(a, [7^+, 9^-]) = r1$, $\mathrm{reg}(c, [7^+, 9^-]) = r2$.

```
switch (...) {
  case 0:
    a = ...; // 1
    b = ...; // 2
    ... = a op b;  // 3
    break;
  case 1:
    b = ...; // 4
    c = ...; // 5
    ... = b op c; // 6
    break;
  case 2:
    a = ...; // 7
    c = ...; // 8
    ... = a op c; // 9
    break;
}
```

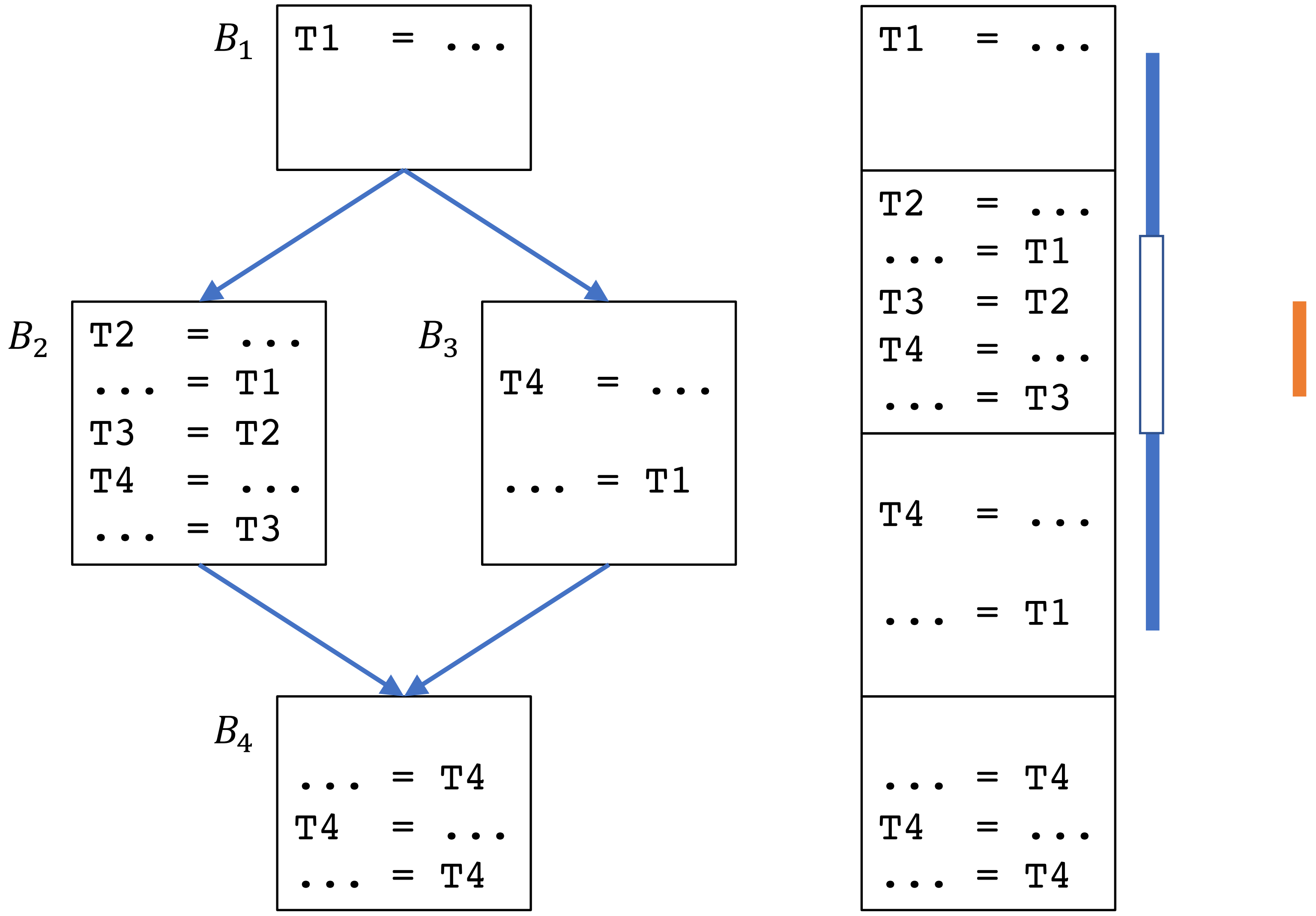
Example #2 [from Sarkar & Barik 2007]

- Can this situation be handled using two physical registers without spilling?
- Graph Coloring says no.
  - The interference graph is the complete graph on $\{a, b, c\}$, and therefore not 2-colorable.
- But this is also solvable as a SFRA problem instance, with the following assignments.
  - $\mathrm{reg}(a, [2^+, 5^-]) = r2.$
  - $\mathrm{reg}(b, [4^+, 7^-]) = r1.$
  - $\mathrm{reg}(c, [1^+, 3^-]) = r1,$
    $\mathrm{reg}(c, [6^+, 8^+]) = r2.$
- Requires the insertion of R-R move instruction `r1 = r2;` between instructions `i8` and `i9`.

```
i1: c = ...
/* Start of loop */
i2: a = ...
i3: ... = c op ...
i4: b = ...
i5: ... = a op ...
i6: c = ...
i7: ... = b op ...
i8: if c <= 0 goto i10

i9: goto i2
/* End of loop */
i10: ...
```

Example #3 [from Traub et al. 1998]



Basic Extended Linear Scan Algorithm

- Applicable to SFRA problem instance.
- Will return:
  - Assignment of physical registers to variables for intervals $[P, Q]$.
  - Modified IR with insertion of reg-reg move instructions to handle cases where different physical registers may be assigned to the same variable in different intervals.
- Guaranteed to find a feasible solution if and only if one exists, with time and space complexity *linear* in size of input SFRA problem instance.
- Intuition
  - Interference among live intervals is captured by whether or not they overlap.
  - Allocate registers to as many intervals as possible without allocating two overlapping live intervals to the same physical register.
  - The number of overlapping intervals changes only at the start and end points of intervals.

Basic Extended Linear Scan: Details (1)

- [Initialize]
  - $\mathcal{I} = \bigcup_{v \in V} \mathcal{I}(v)$, the set of all intervals in program.
  - IEP, the set of endpoints in $\mathcal{I}$.
  - *numlive* = 0. *count*$[P]$ = 0, $\forall P \in IEP$. $avail \ = \ \{1, \dots, k\}$.
- [Calculate register pressure]
  $\forall P \in IEP$, in increasing order:
  - $\forall [x, P] \in \mathcal{I}$: *numlive*--. $\forall [P, y] \in \mathcal{I}$: *numlive*++. *count*$[P]$ = numlive.
- [Assign registers]
  $\forall P \in IEP$, in increasing order:
  - $\forall [x, P] \in \mathcal{I}$: $avail = avail \cup \{r_j\}$, where r_j is the physical register previously allocated to interval $[x, P]$.
  - $\forall [P, y] \in \mathcal{I}$:
    - Let v be the variable corresponding to interval $[P, y]$.
    - Select r_j from $avail$, handling possible liveness of v at P^- and R-R copy cases.
    - $\mathrm{reg}(s, [P, y]) = \ r_j$.
    - $avail \ = \ avail \setminus \{r_j\}$.

Basic Extended Linear Scan: Details (2)

- [Insert needed register move instructions]
  $\forall P \in IEP$:
  $\forall Q \in IEP$ that is a control flow successor to P:
  - $M = \emptyset$.
  - $\forall v \mid v$ is live at P and Q:
    - $\textbf{if } \mathrm{reg}(v, P) \neq \mathrm{reg}(v, Q)\text{: } M = M \cup \{\text{“}\mathrm{reg}(v, Q) = \mathrm{reg}(v, P);\text{”}\}$.
  - [Handle cyclic permutations]
    - Treat the move instructions in M as a directed graph G in which there is an edge from m_1 to m_2 if m_1 reads the register written by m_2.
    - Compute the strongly connected components (SCCs) of G.
    - For each SCC, create a sequence of move and XOR instructions as needed to implement the register moves without using further temporary registers, and insert these instructions on the control flow edge from P to Q.

Register Allocation with Total Spills

- How do we extend the SFRA problem statement to allow for total spills (i.e., identifying a subset of variables for which *all accesses will be performed through memory instead of registers*), with the goal of finding a solution with the smallest spill cost?
- [RATS] Given a set V of symbolic registers (i.e., program variables and temporaries) being used across n instructions, k physical registers, and estimated execution frequencies $f[P]$ for each program point P, determine a solution that minimizes the weighted sum of R-R moves and memory accesses. Return:
  - $\forall v \in V$: a Boolean function $mem(v)$ indicating whether v is kept in memory.
  - $\forall v \in V \mid mem(v) = \mathbf{false}$: a register assignment $\mathrm{reg}(v, P)$ at each program point P where v is live.
- This optimization problem is NP-hard, even ignoring R-R moves.

Extended Linear Scan Algorithm

- Heuristic algorithm for RATS.
- The *spill identification* phase checks all relevant program points in decreasing order of estimated frequency and spills variables on a stack, using a simplified version of Chaitin’s heuristic that doesn’t build the interference graph.
  - Spills happen in *increasing* order of spill cost.
  - Results in a feasible SFRA problem instance, possibly with slack.
- A *spill resurrection* phase then tries to restore any spilled variables (in LIFO order) that still keeps the SFRA problem instance feasible.
  - Resurrection tries to restore costliest spills first.
- Now perform the register assignment phase using the Basic Extended Linear Scan algorithm.

Conclusions on the Design Space

- [Koes & Goldstein 2009] offer the following conclusions on the dimensions of the register allocator design space (assignment, move insertion, coalescing, spilling) based on their systematic empirical investigation covering the x86, x86-64, Armv6, and Thumb ISAs.
  - The ability to insert moves (that is, split live ranges or undo coalescing) has surprisingly little impact on code quality. What benefit there is can primarily be obtained by allowing move insertions only at basic block boundaries. Thus, register allocators need not be designed to explicitly optimize move insertions, although allowing insertions may simplify the design of algorithms for other components.
  - Coalescing, when viewed as a move elimination problem separate from register assignment, is highly separable: it can be performed as a standalone pass without materially degrading code quality. Furthermore, a simple greedy heuristic is nearly as effective as an optimal algorithm.
  - When optimizing for performance on a modern processor, spill code optimization is of paramount importance. Furthermore, the various components of register allocation can be treated separately without significantly impacting performance. Therefore, when targeting processor performance, new register allocator designs should focus on solving the spill code optimization problem as the coalescing, move insertion, and register assignment problems are adequately solved using existing heuristics.
  - Both spill code optimization and register assignment are important contributors when optimizing for code size. If code size is of paramount importance, then integrating these two phases can result in an additional small improvement. Therefore, when targeting code size, new register allocator designs should focus on solving both the spill code optimization and register assignment problems, possibly in an integrated framework.