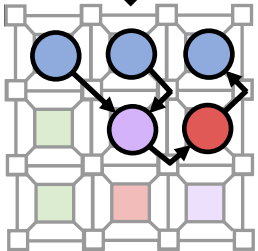


**Complete
system stack**

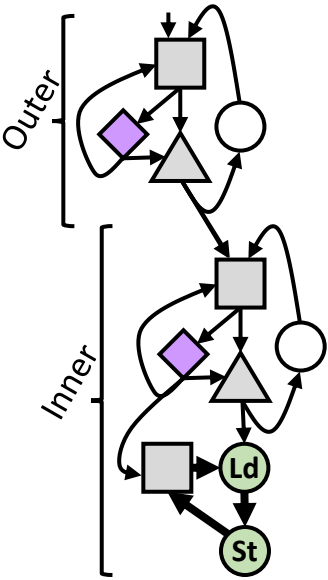
```
int w = 0;  
for (...)  
  w += A[j];  
Z[0] = w;  
Arbitrary code
```

Compiler



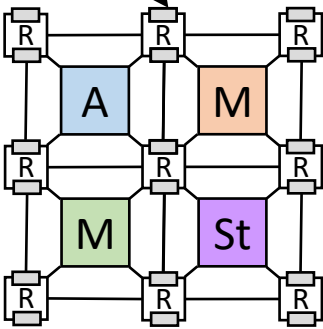
Generated CGRA
hardware

Tag-less dataflow
+ Nested loops
+ Load-store ordering



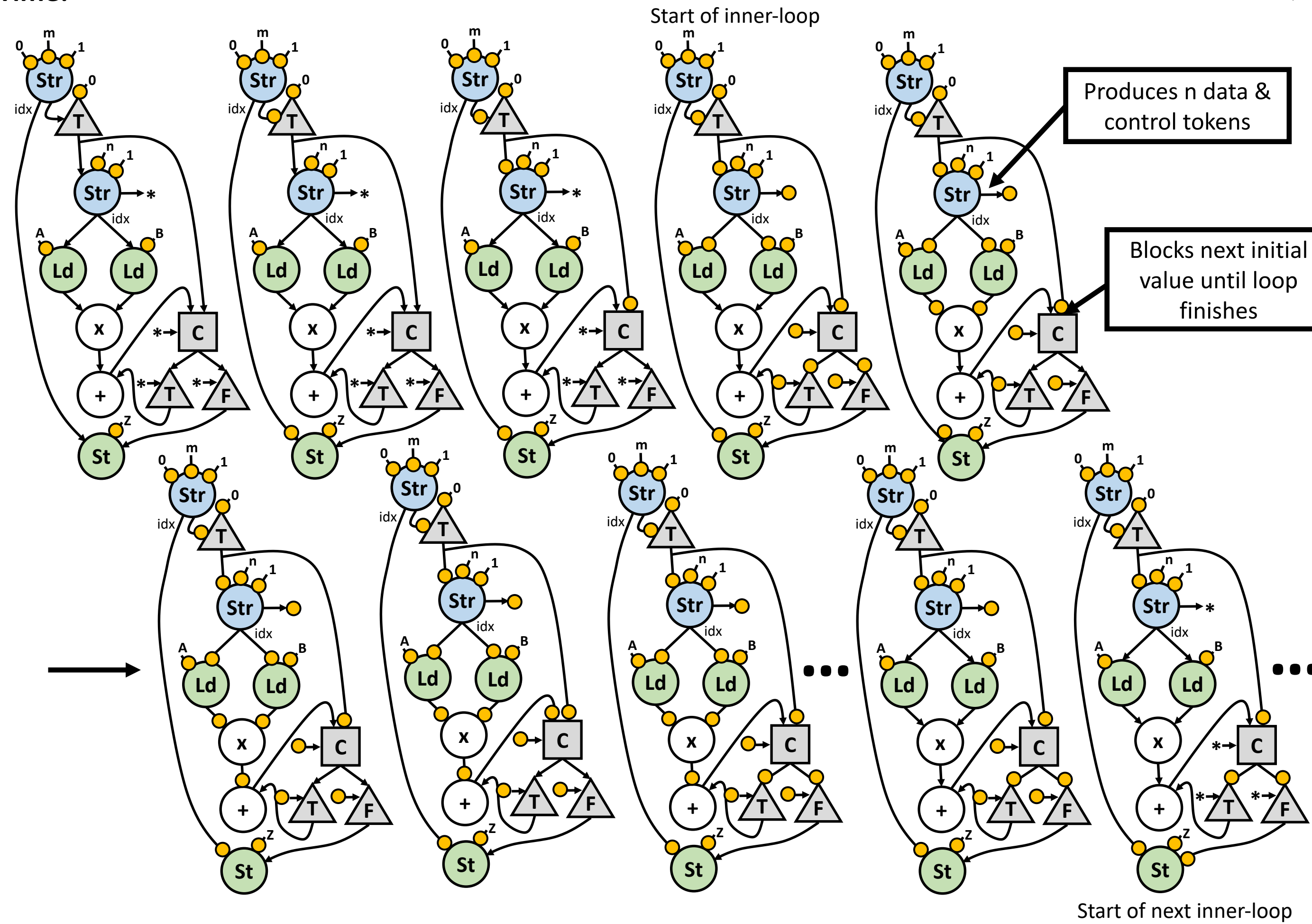
**Control flow
in the NoC**

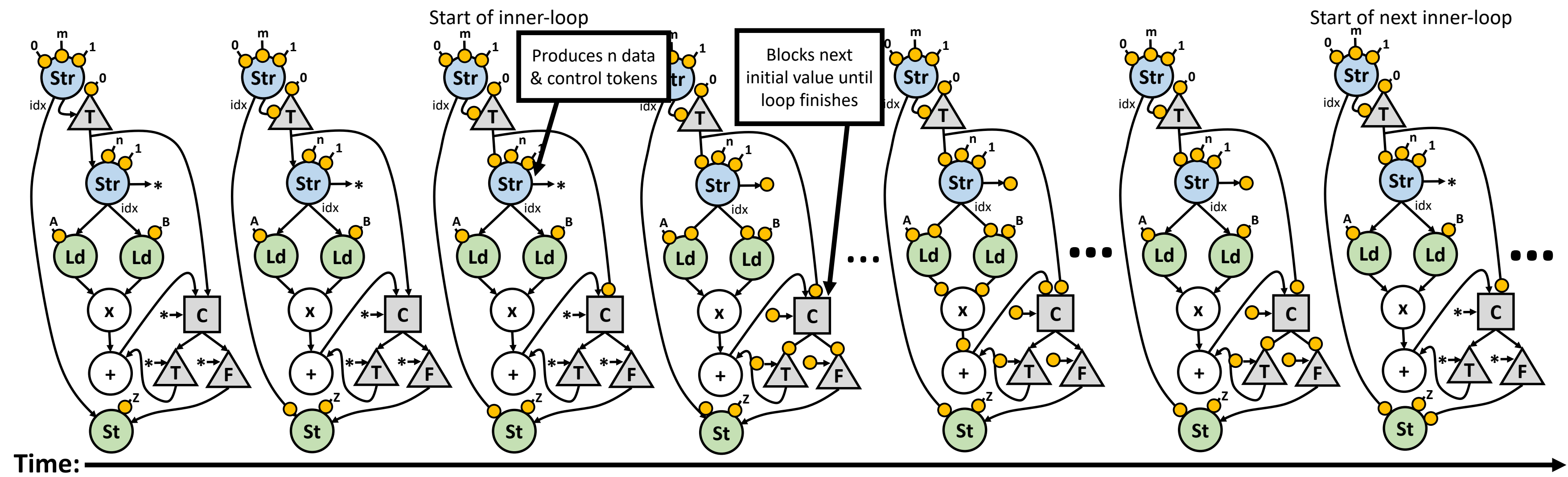
Control-flow ops:
[C, T, O, ...]



Reuses existing
hardware

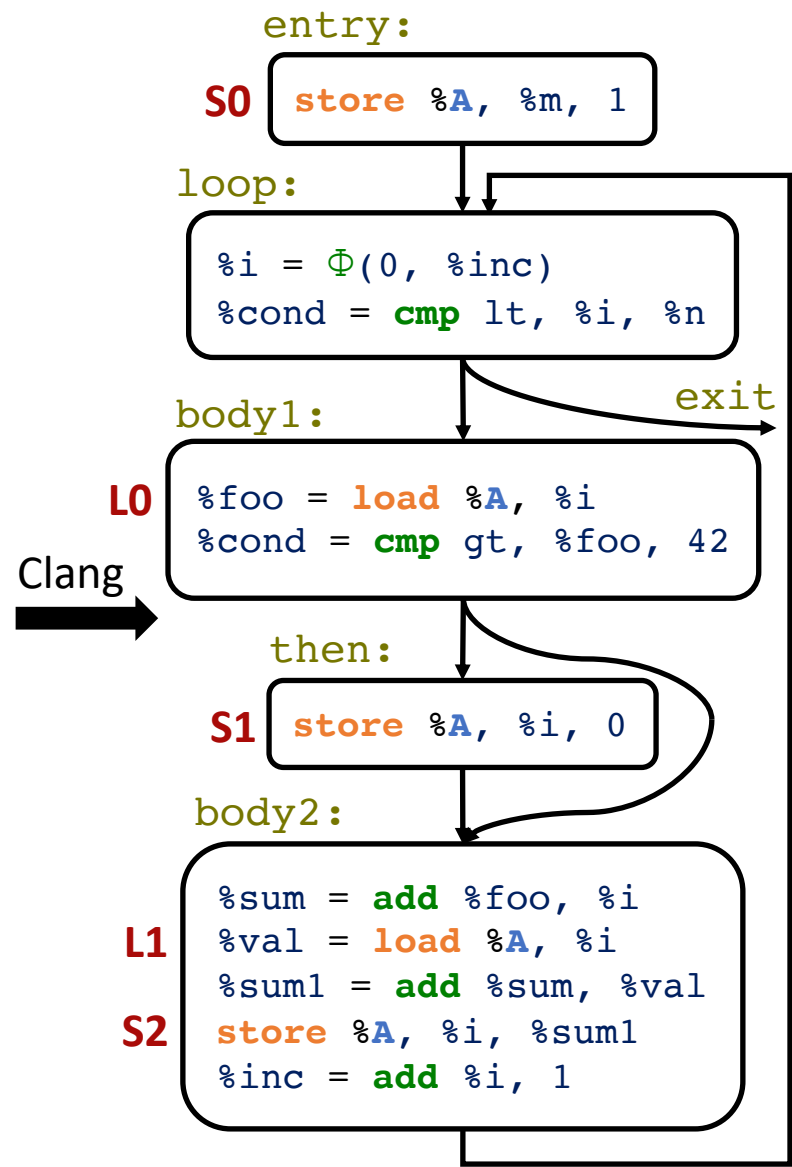
Time: 



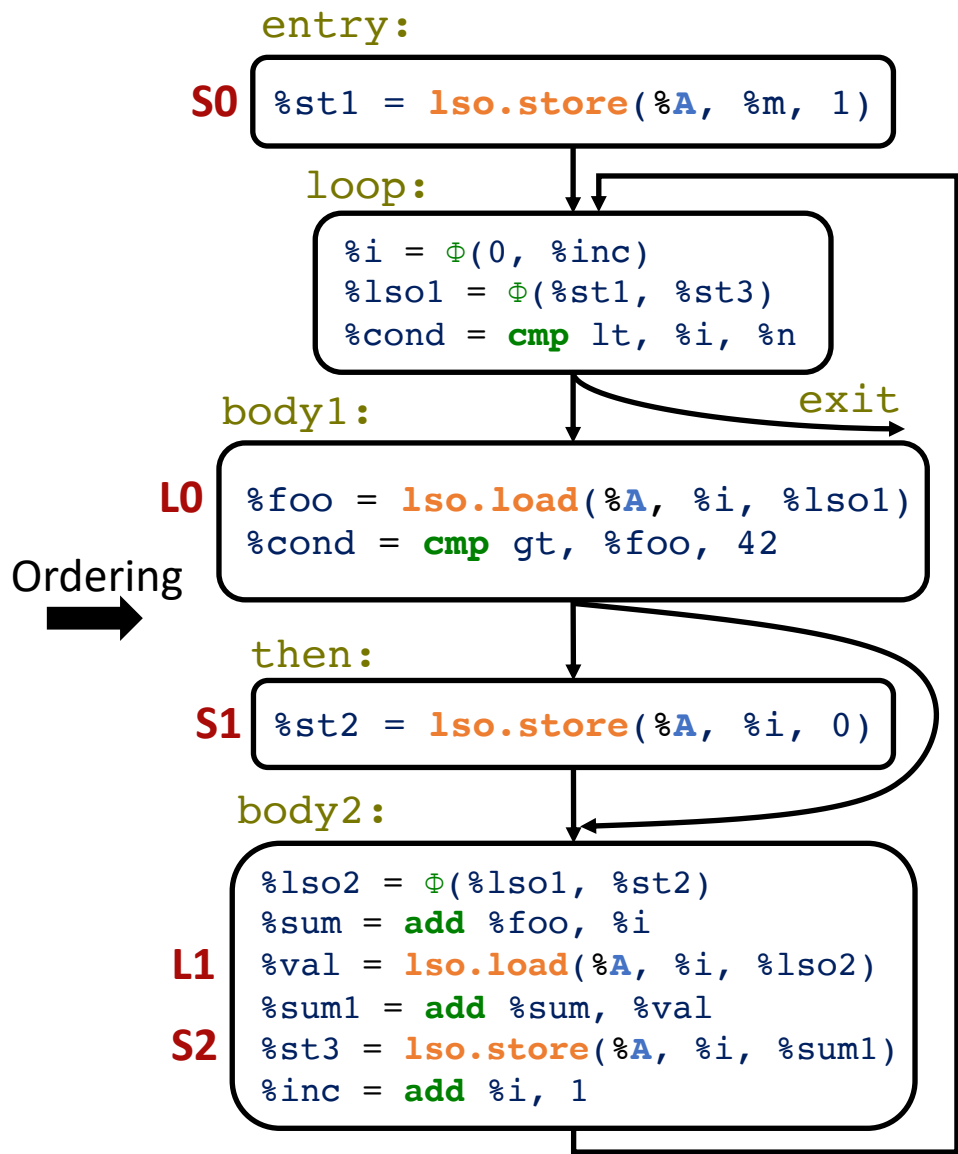


```
void example(  
  int *A, int n, int m  
) {  
  A[m] = 1;  
  for (int i = 0; i < n; i++) {  
    int foo = A[i];  
    if (foo > 42) {  
      A[i] = 0;  
    }  
    A[i] += foo + i;  
  }  
}
```

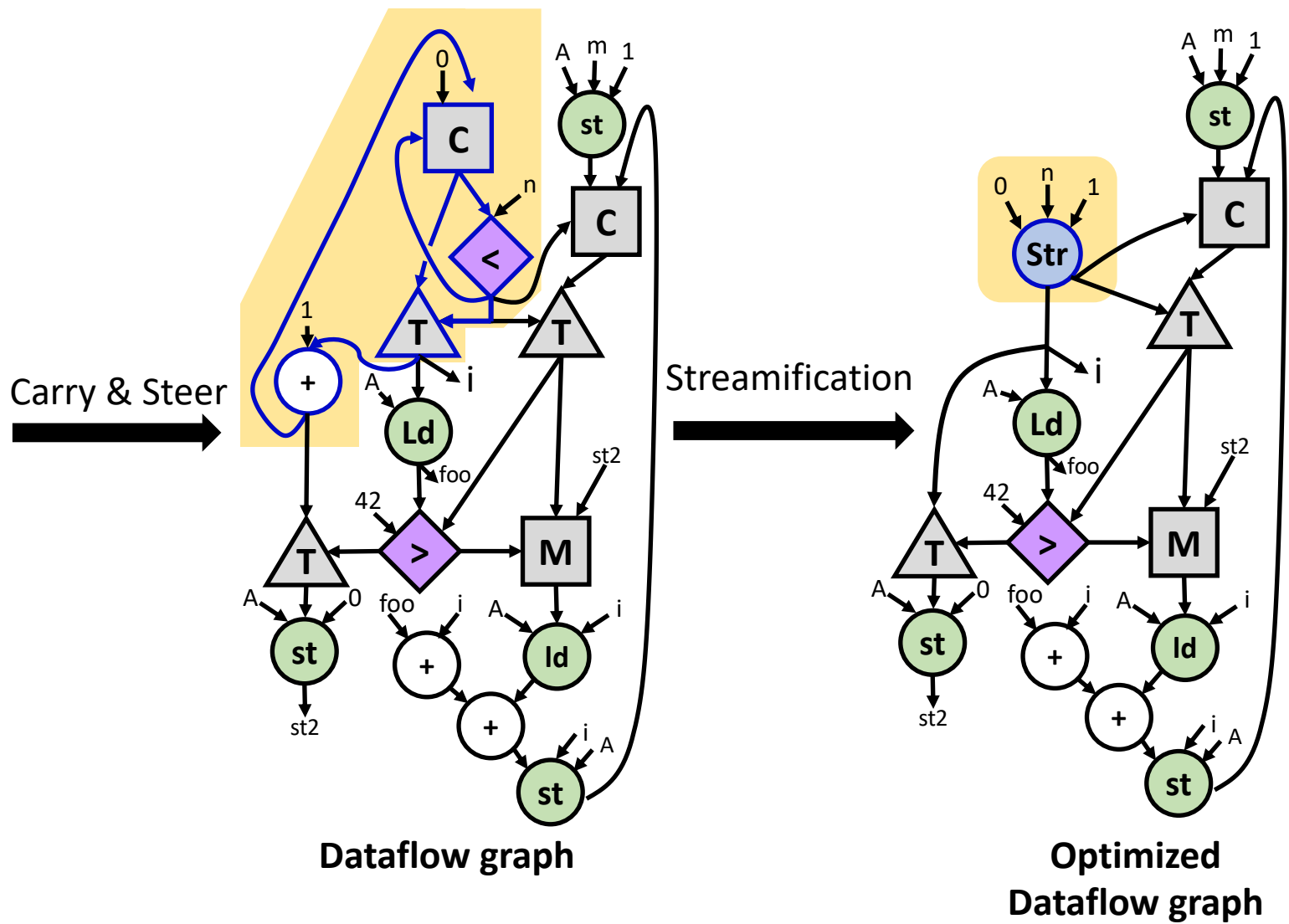
Source Code

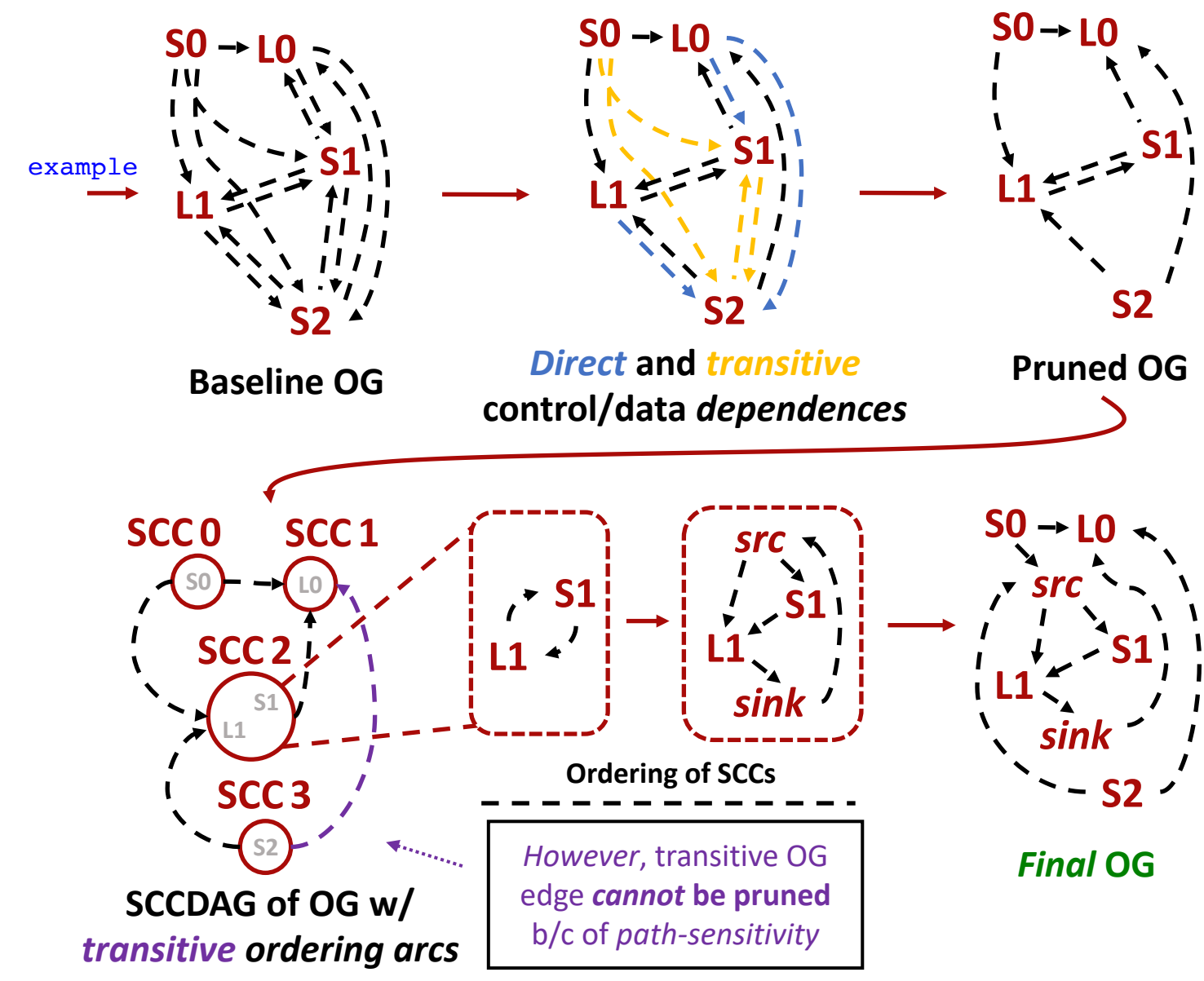


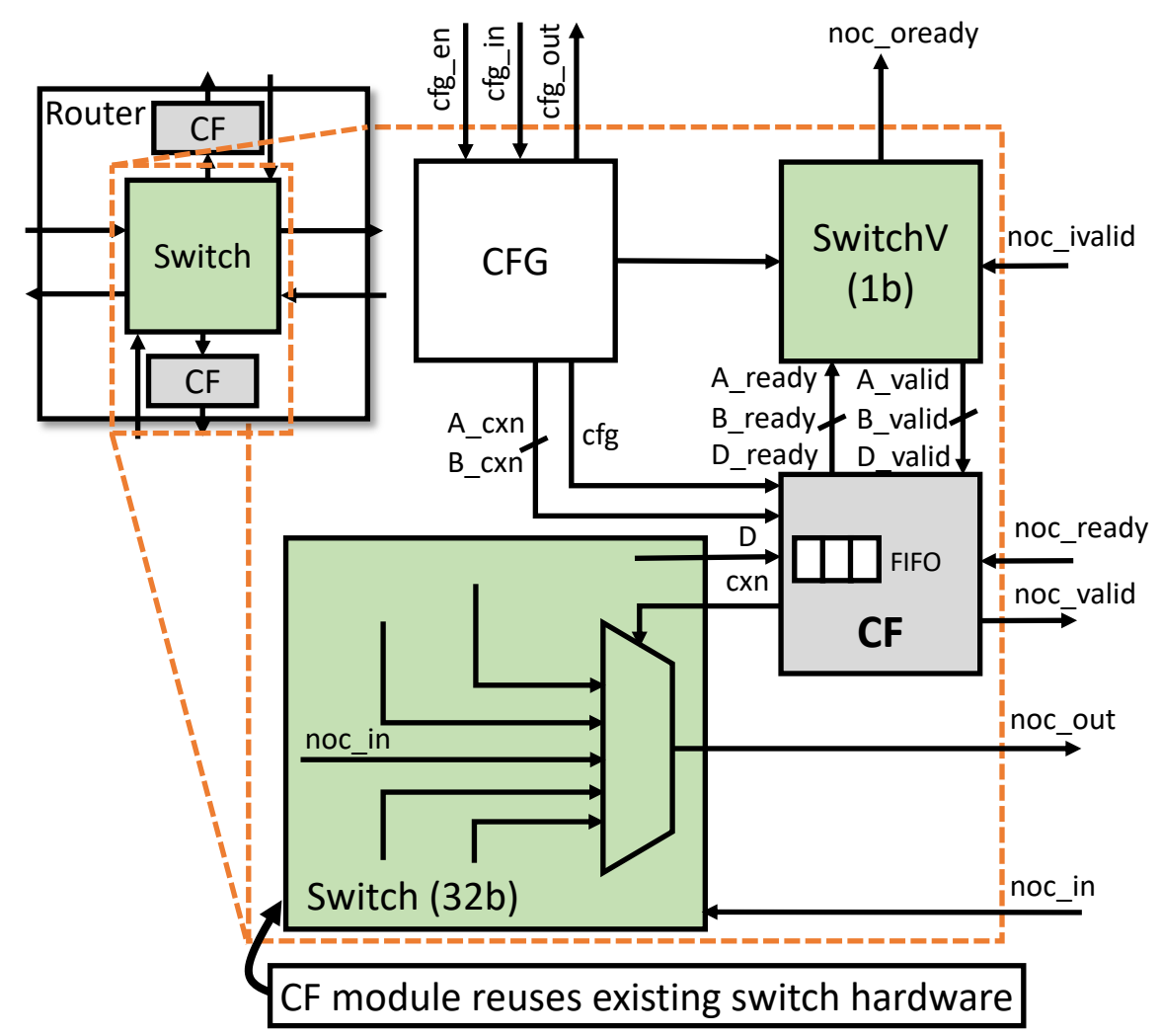
CFG w/ simplified LLVM-IR

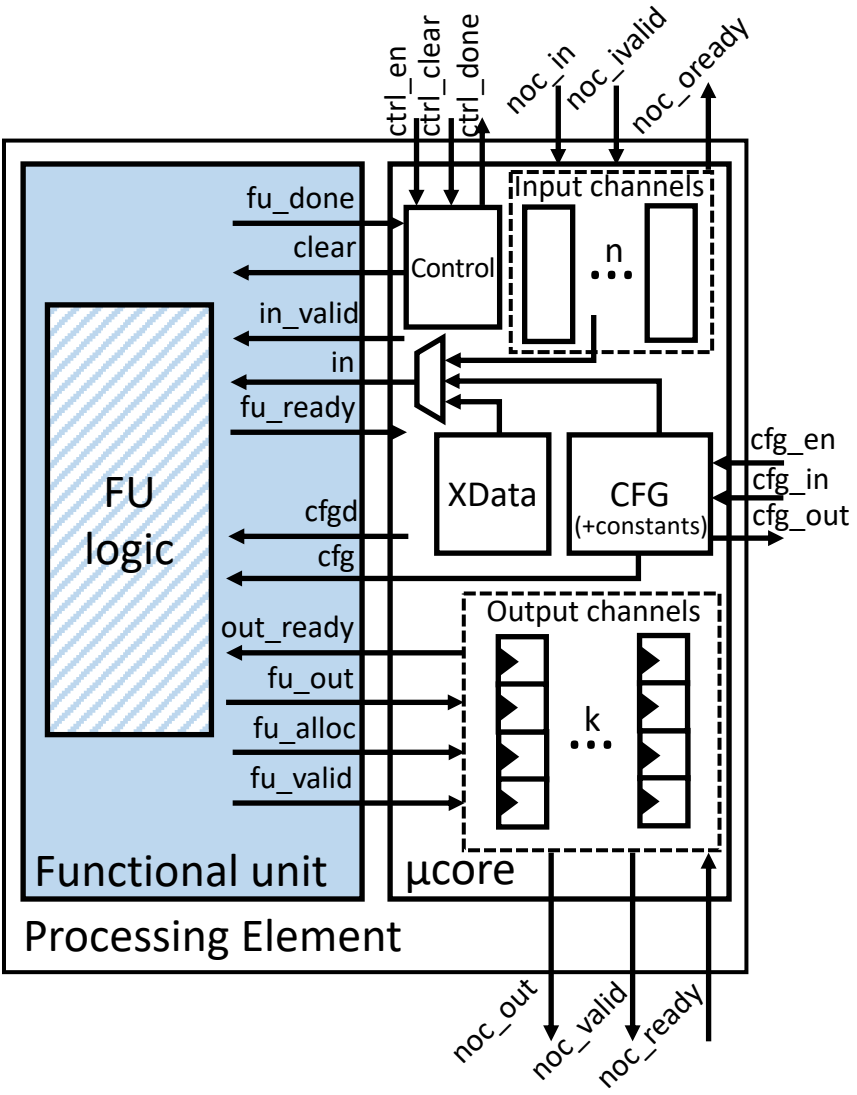


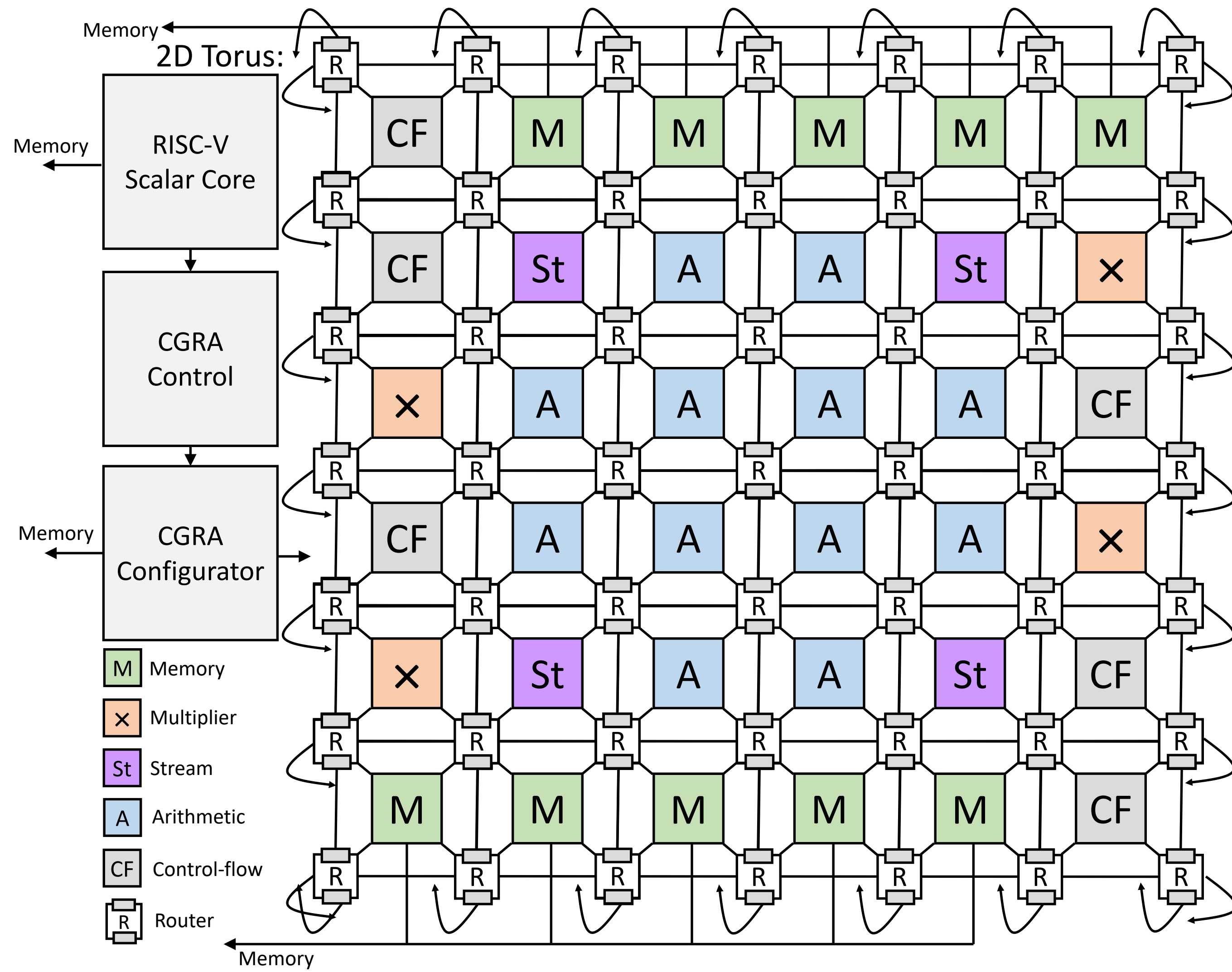
LLVM-IR
(memory ordering enforced)

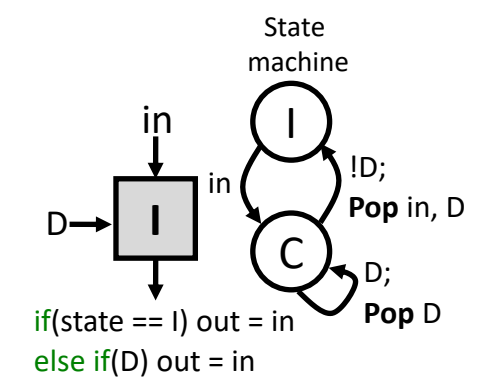


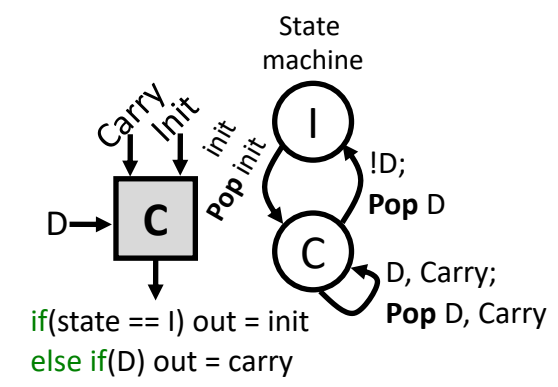


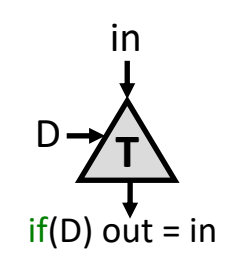


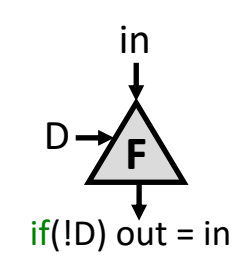


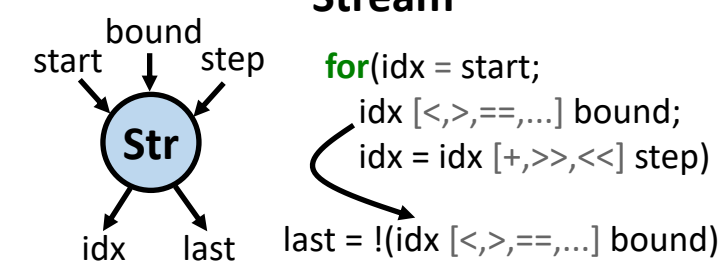
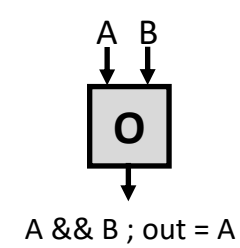
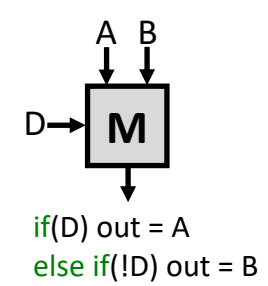
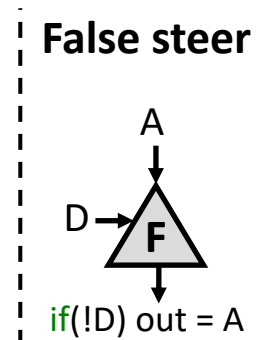
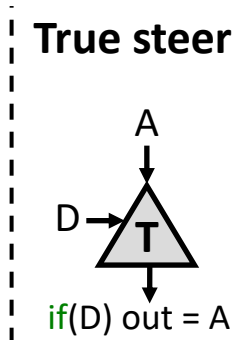
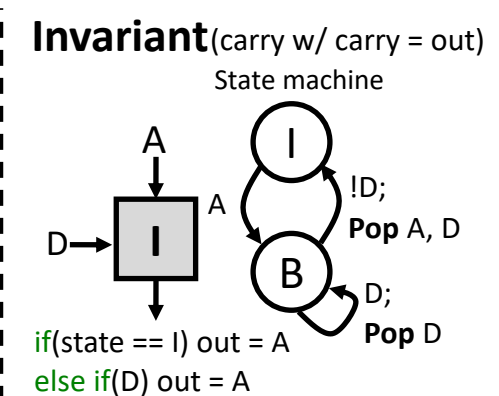
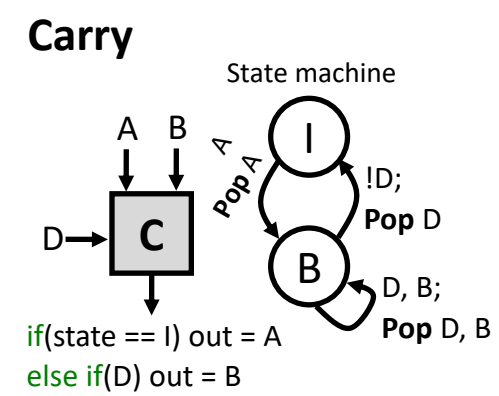


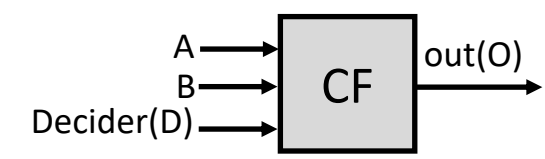








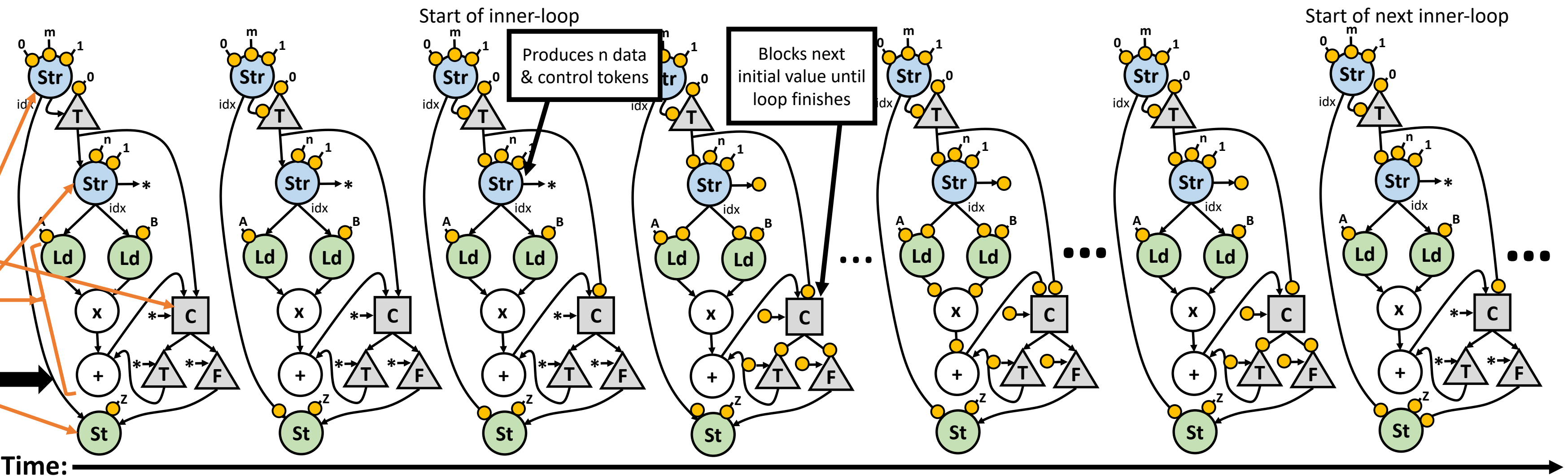




```
void example1(  
    int *A, int *B, int *Z,  
    int m, int n  
) {  
    for(int i = 0; i < m; i++) {  
        int w = 0;  
        for(int j = 0; j < n; j++) {  
            w += A[j] * B[j];  
        }  
        Z[i] = w;  
    }  
}
```

C Code

Time:




```

void example2(
  int *A,
  int n,
  int m
)
{
  A[m] = 1;
  for (int i = 0; i < n; i++) {

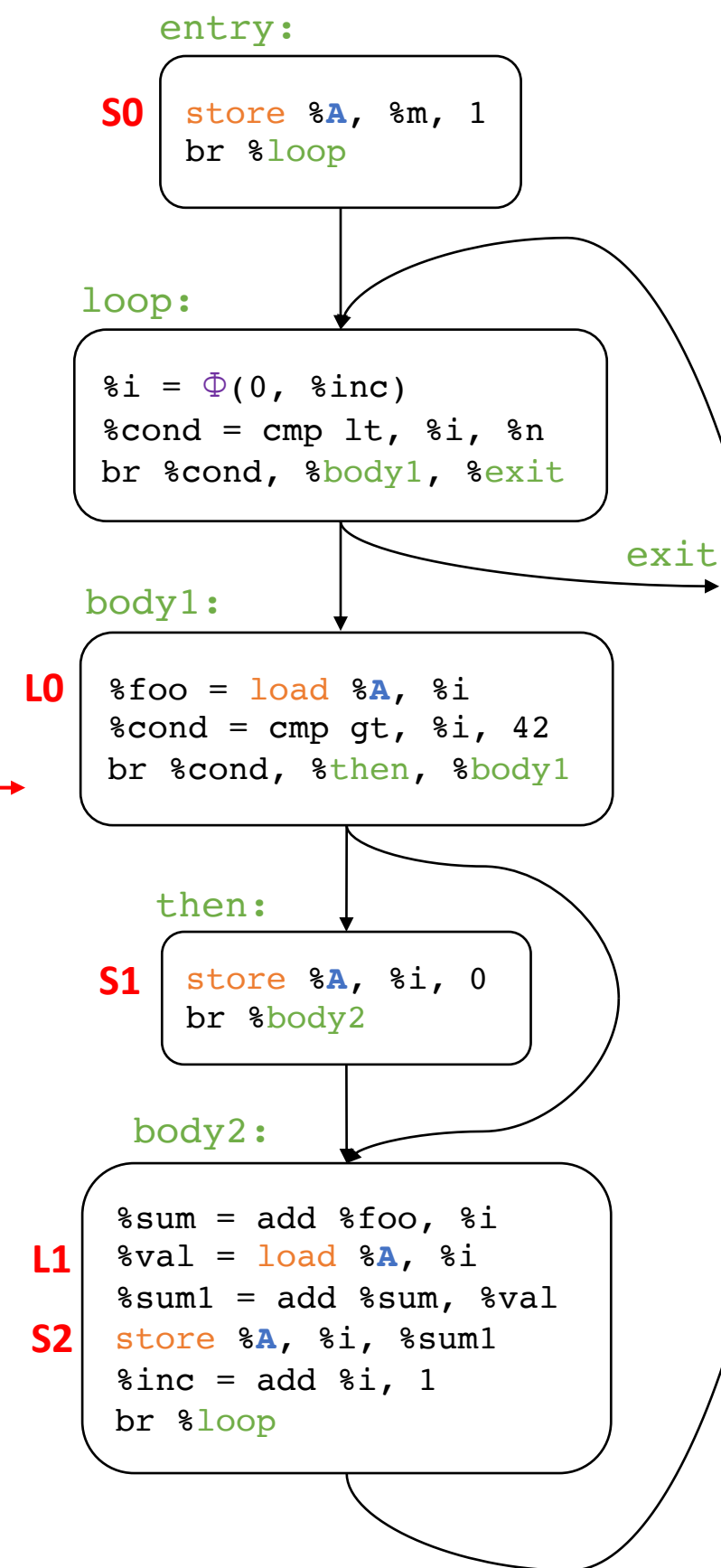
    int foo = A[i];

    if (foo > 42) {
      A[i] = 0;
    }

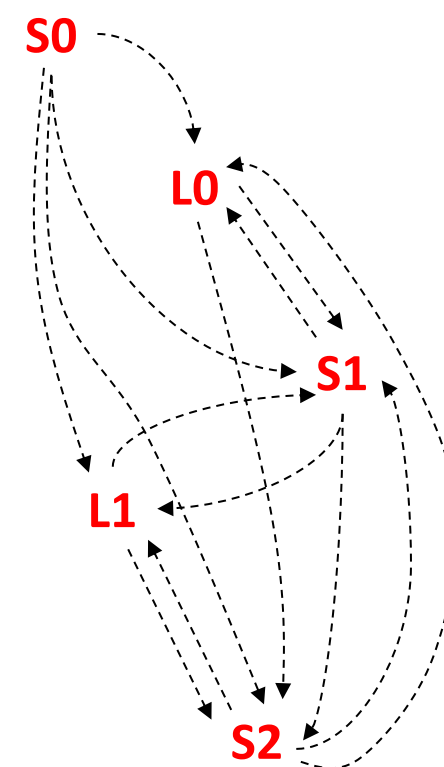
    A[i] += foo + i;
  }
}

```

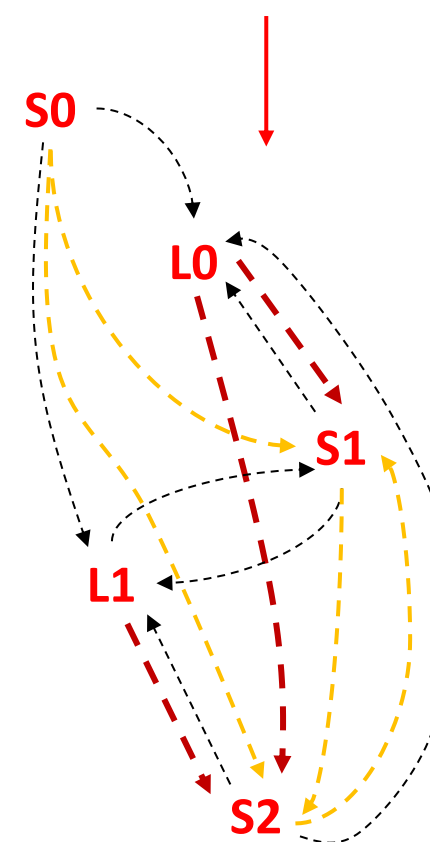
Source Code



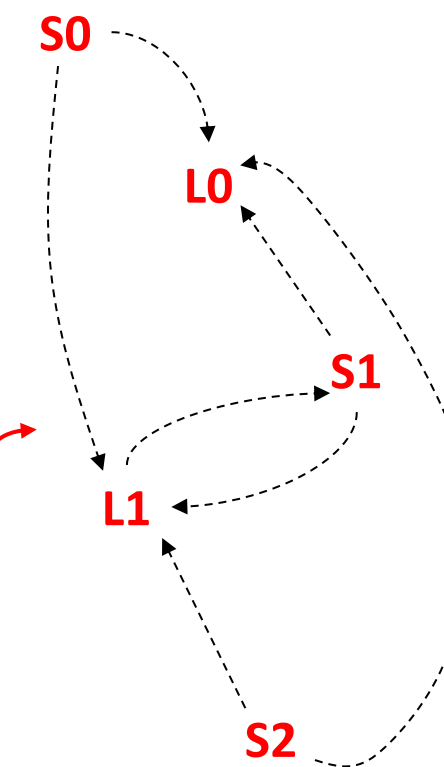
Simplified LLVM-IR



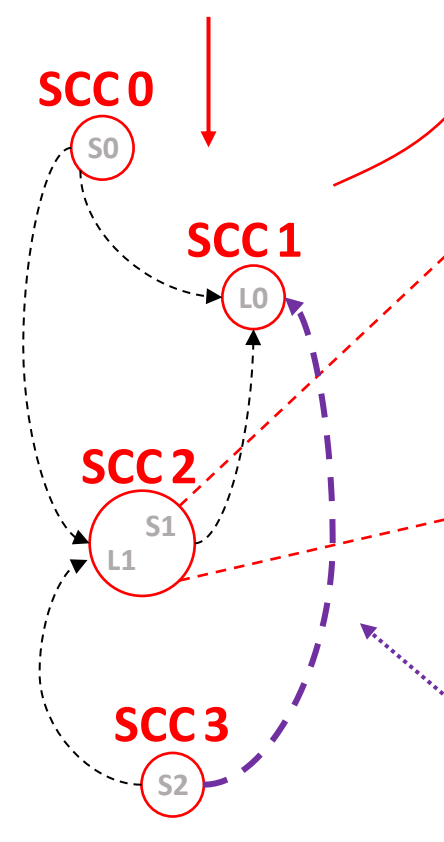
Baseline LSO Graph



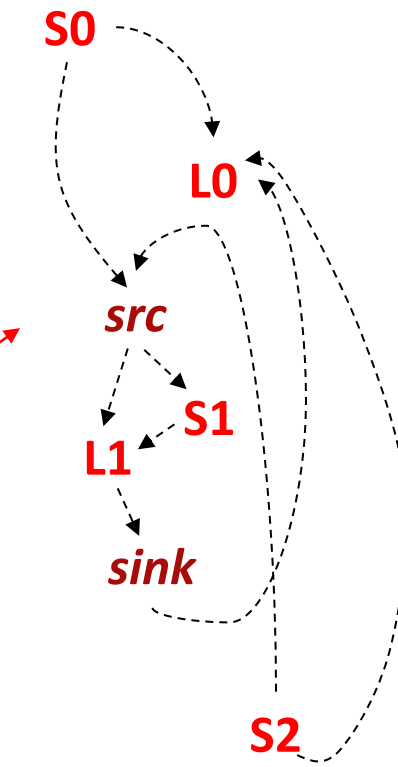
Direct and **transitive**
control/data *dependences*



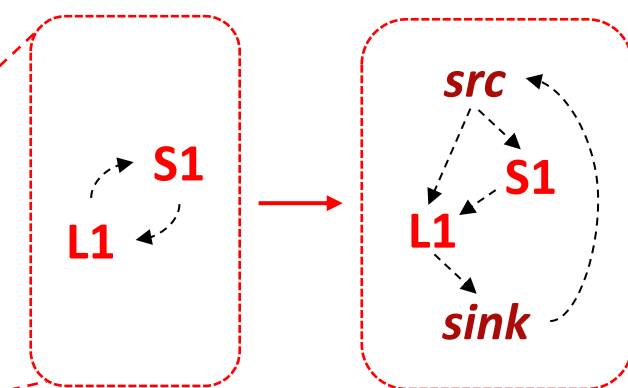
Pruned LSO Graph



Condensed LSO Graph
w/ **transitive** LSO edges

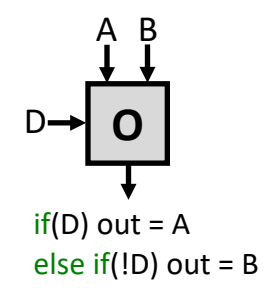


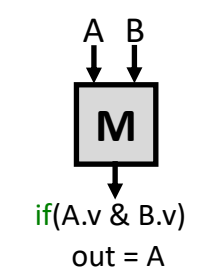
Final LSO Graph

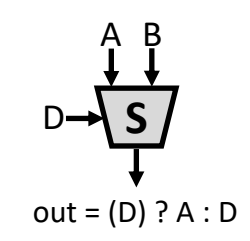


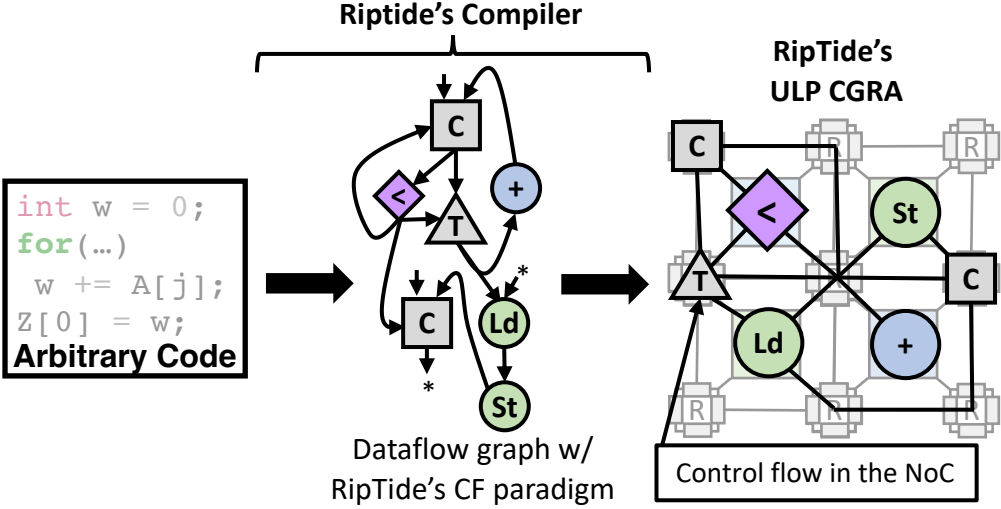
Explicit sequentialization
of SCCs

However, transitive LSO
edge **cannot** be pruned
b/c of *path-sensitivity*







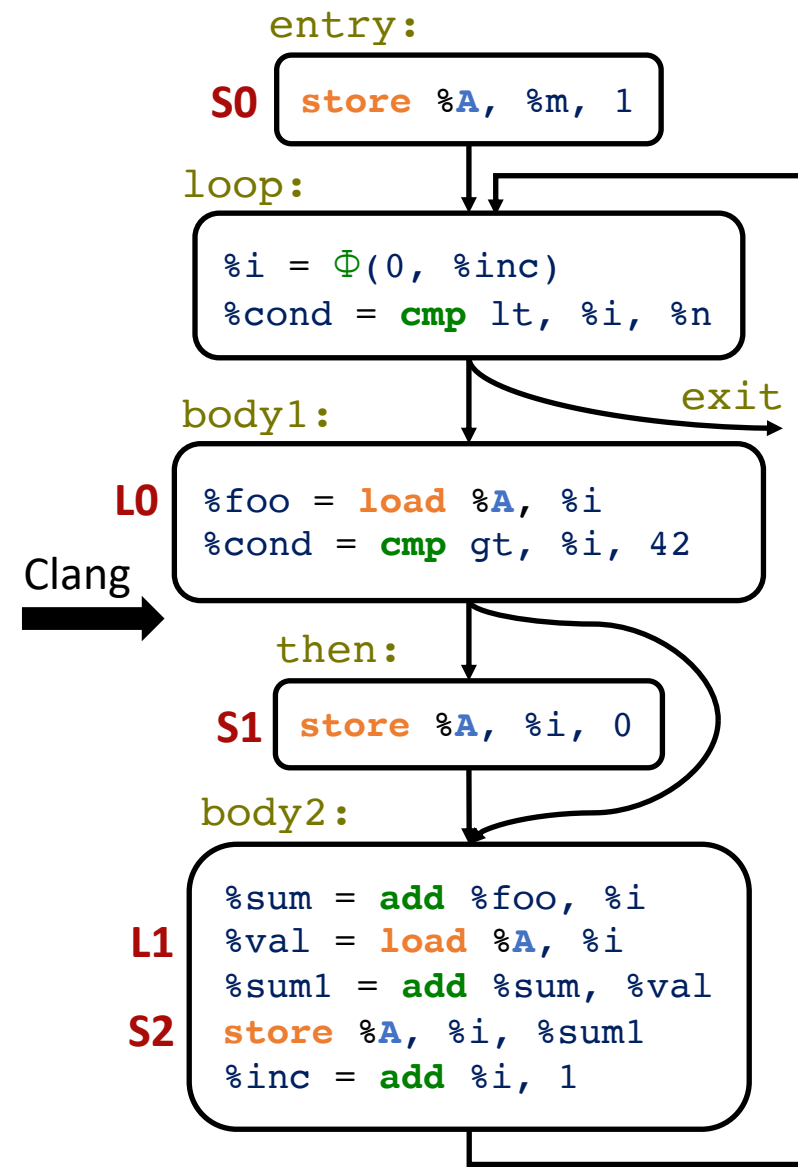


```

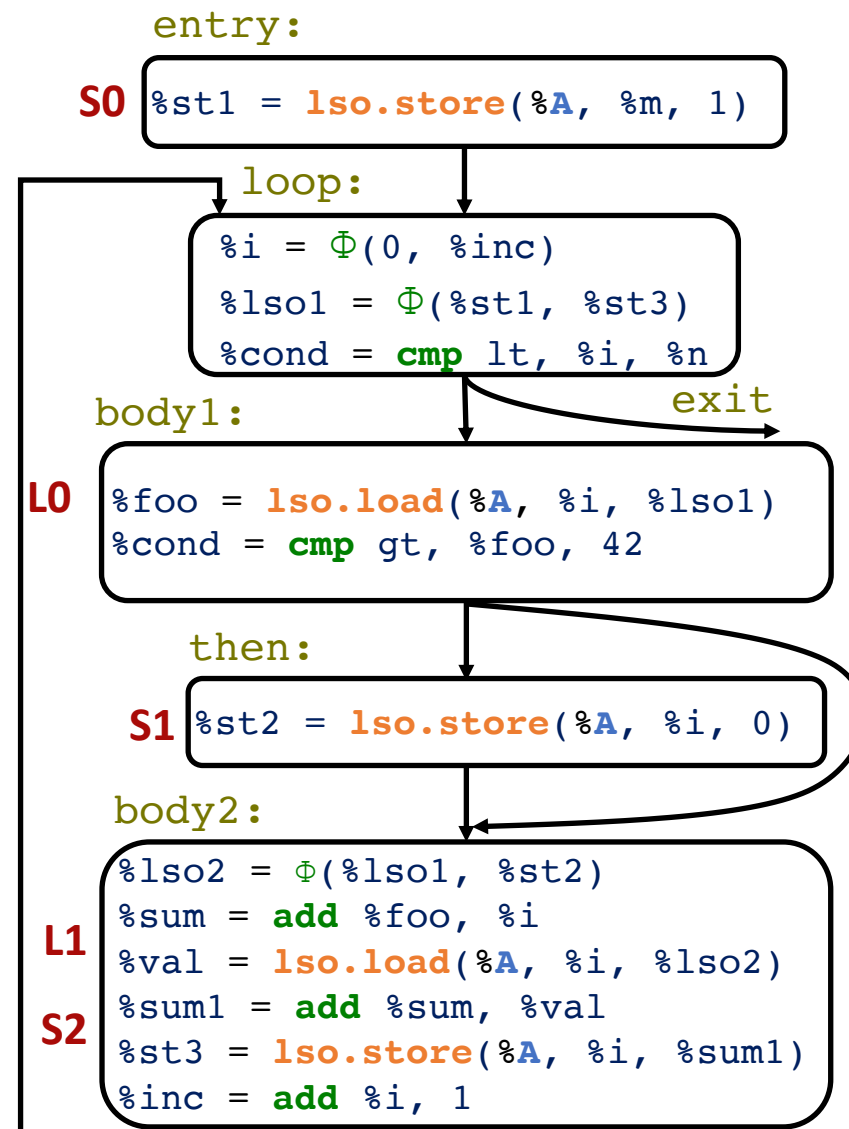
void example2(
  int *A, int n, int m
) {
  A[m] = 1;
  for (int i = 0; i < n; i++) {
    int foo = A[i];
    if (foo > 42) {
      A[i] = 0;
    }
    A[i] += foo + i;
  }
}

```

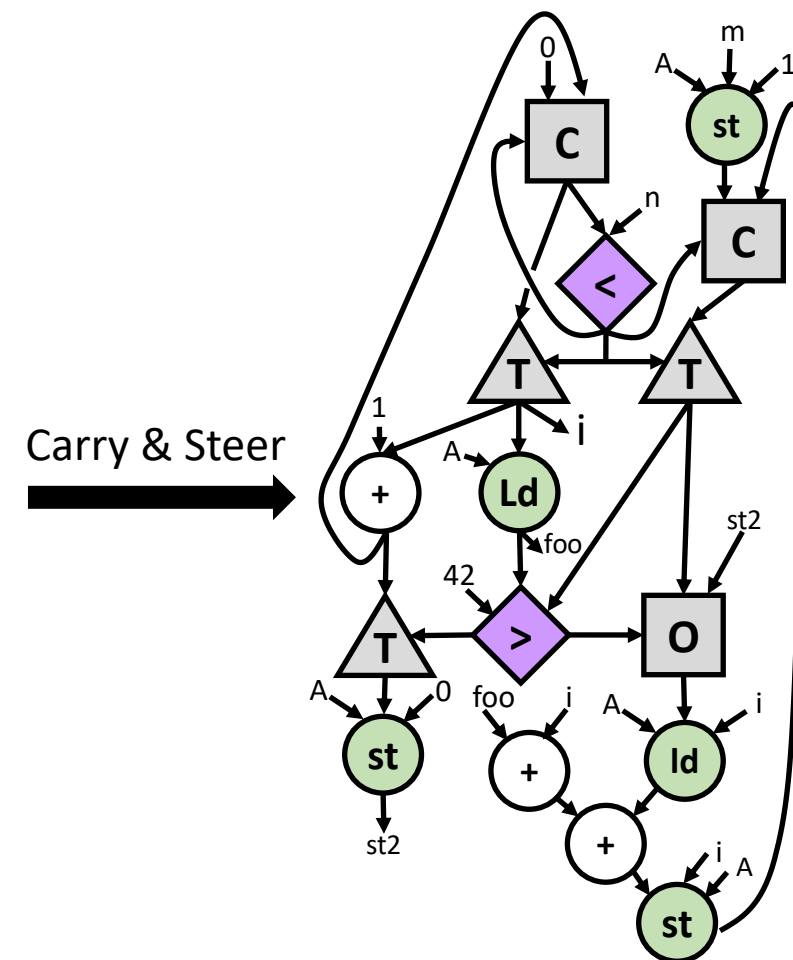
Source Code



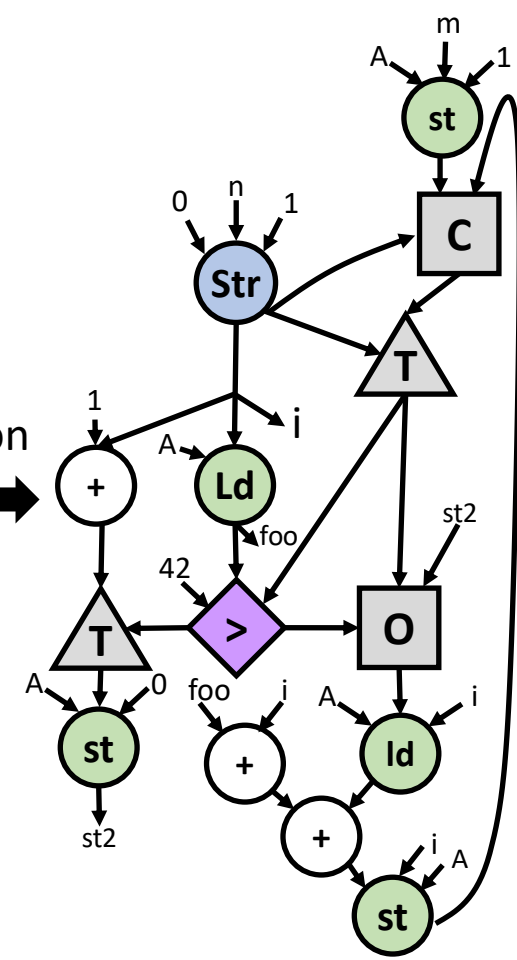
Simplified LLVM-IR

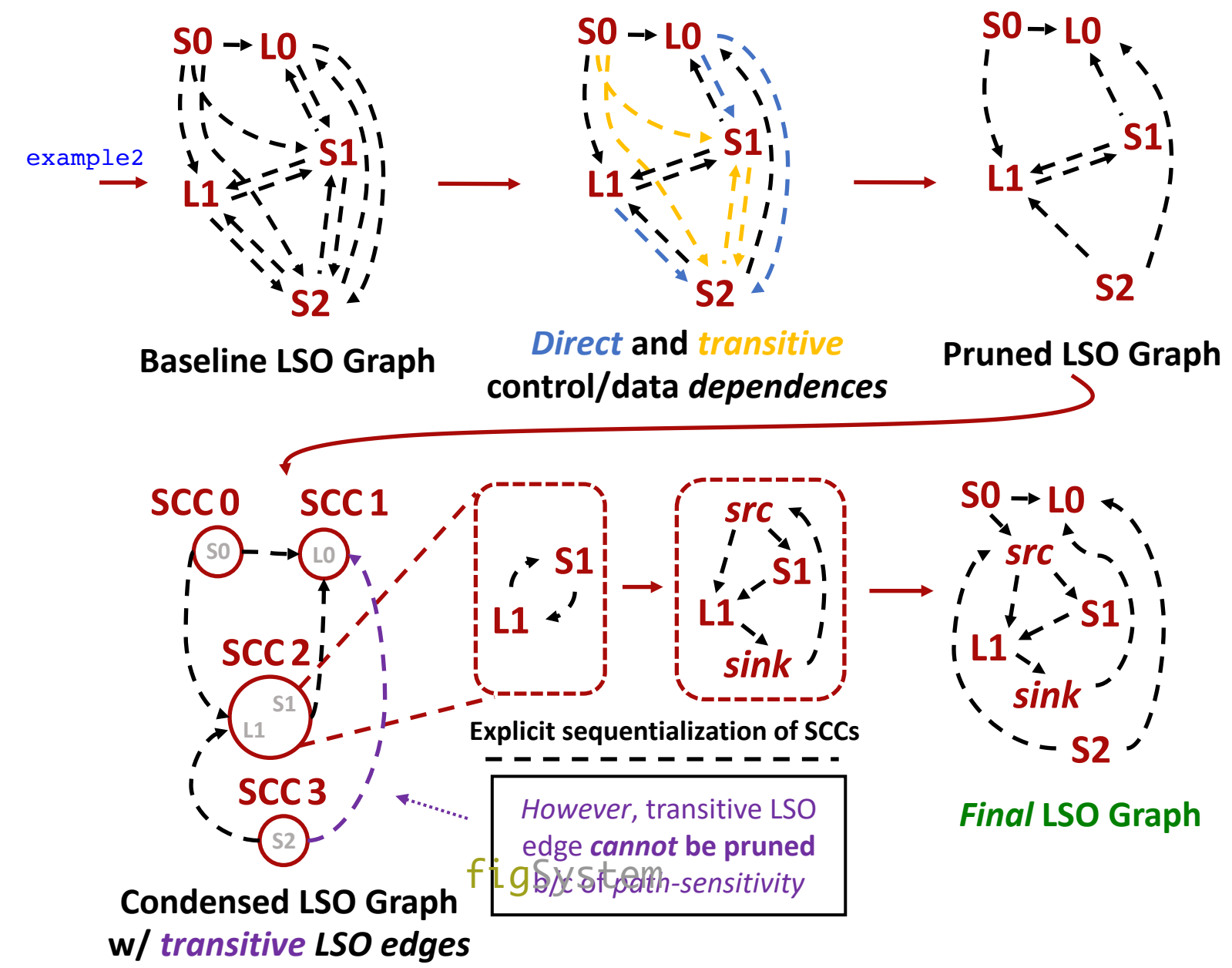


(load-store ordering enforced)



Streamification





Loop annotations (in other works)

```
// REVEL-like pragmas
#pragma config
#pragma stream
#pragma dataflow in(...)
for (...)

// Annotations for 4D-CGRA, etc.
#pragma accelerate
for (...)
...
```

RipTide Code (Native C)

```
#define u16 uint16_t
#define u32 uint32_t
#define res restrict // RipTide annotation

void simple_bfs(
    u16 * res rows, u16 * res cols, // Graph in CSR
    u16 * res queue, u16 * res visited, // Helpers
    u16 * res walk // Output
) {
    while (!stack_empty()) {
        // Record next vertex in @walk
        u16 next = pop();
        add_to_walk(next);

        // Add neighbors to @queue
        for (u32 i = rows[next];
             i < rows[next + 1]; i++) {
            u32 dst = cols[i];
            if (!visited[dst]) {
                push(dst);
                visited[dst] = 1;
            }
        }
    }
}
```

SNAFU assembly excerpt

```
// Config to offload loop
u16 tmp;
u32 start = rows[next];
u32 stop = rows[next + 1];
if ((stop - start) <= 0) continue;
vcfg(((stop - start), _kernel);
vtfr(cols + start, BFS_SNAFU_VTFR0);
vtfr(visited, BFS_SNAFU_VTFR1);
...
vfence();

// SNAFU assembly for loop
vlh(v1, cols);
vlxh(v2, visited, v1);
vseqi(v0, v2, 0);
vsxh(visited, v0.m, v1);
vpresum(v4, v0);
...
```


1 void foo (...) { **a.**
2 ...
3 #pragma target
4 for (i = 0..n)
5 ... = a[i] ...
6 ...
7 }

1 void foo (...) { **b.**
2 ...
3 #pragma target
4 for (i = 0..n)
5 for (j = 0..n)
6 ... = a[i][j] ...
7 ...
8 }

vcfg(...); **e.**
... // Config
vlh(v1, b);
vlxh(v2, a, v1);
veq(v0, v2, 1);
... // Ops for push(i)

1 void foo (...) { **c.**
2 ...
3 #pragma config ...
4 for (i = 0..n)
5 ...
6 #pragma stream
7 #pragma dataflow in(...)
8 for (j = i..n)
9 ... = a[i][j] ...
10 ...
11 }

1 void foo (int * restrict a, **d.**
2 int * restrict b, ...) {
2 ...
4 while (!queue.empty()) {
5 n = queue.pop()
6 for (i = a[n]..a[n + 1])
7 if (b[a[i]]) queue.push(i)
8 ...
9 }
10 ...
11 }

```
void foo (...) {  
    #pragma target  
    for (i = 0..n)  
        ... = a[i]  
}
```

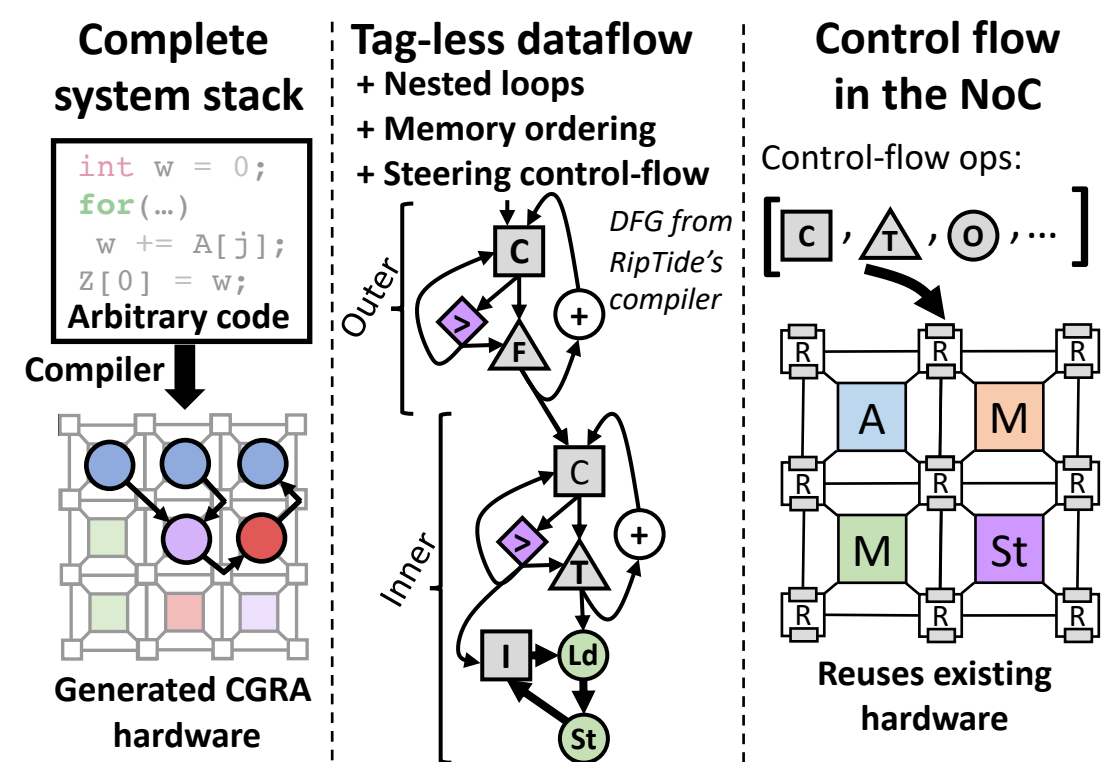
```
void foo (...) {  
    ...  
    #pragma target  
    for (i = 0..n)  
        for (j = 0..n)  
            ... = a[i][j]  
    ...  
}
```

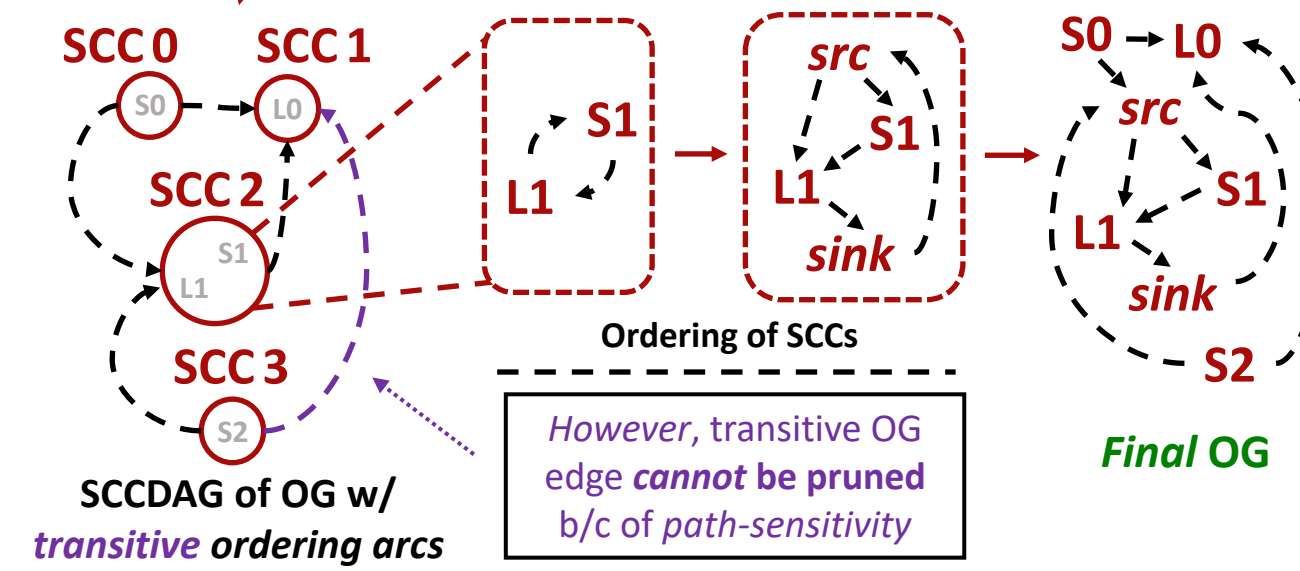
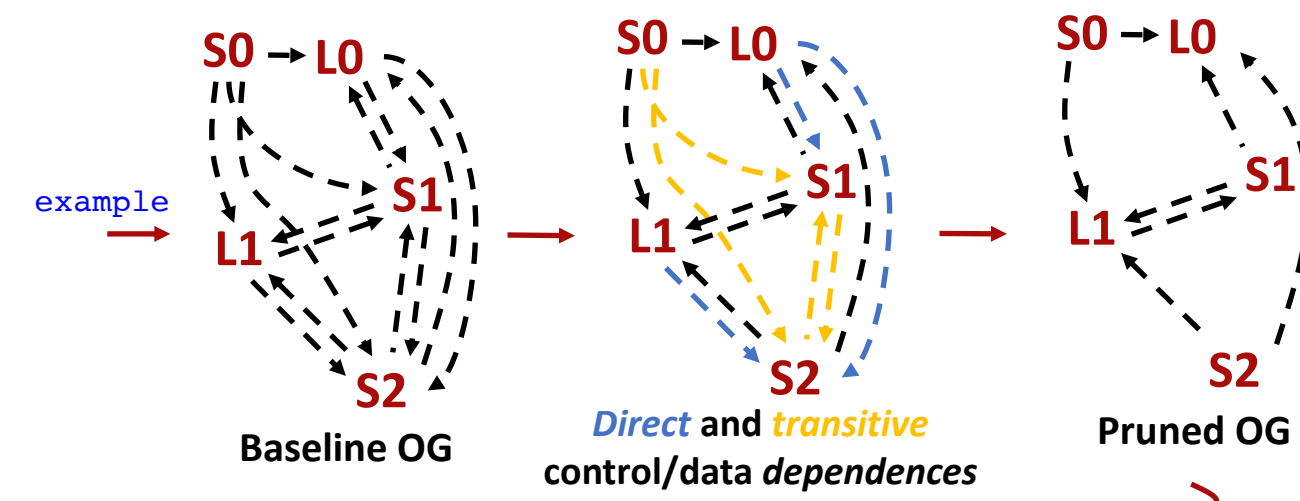
```
void foo (...) {  
    #pragma config ...  
    for (i = 0..n) ...  
        #pragma stream  
        #pragma dataflow  
        for (j = i..n)  
            ... = a[i][j]  
}
```

```
#riptide void foo
(int * restrict a, b) {
    while (!q.empty()) {
        n = q.pop()
        for (i in 0..n)
            if (b[a[i]]) ...
    }
}
```

```
void foo (...) {  
  for (i = 0..n) ...  
    vlh v1, a + i  
    vlh v2, b  
    vadd v3, v1, v2  
    vsh b + i, v3  
  ...  
}
```

```
void foo (int * a,  
         int * b) {  
    while (!q.empty()) {  
        n = q.pop()  
        for (i in 0..n)  
            if (b[a[i]]) ...  
        }  
    }
```









```
void foo (...) {  
    ...  
    for (i = 0..n) {  
        while (a[i] != 0)  
            a[i] = ...  
        ...  
    }  
}
```