CUDA Graphs Conditional Nodes

Stephen Jones, NVIDIA

HiHAT working group meeting, August 20th 2024

Data-Dependent Execution

"Iterate until converged" is an almost universal pattern

```
function ConjugateGradient(A, b, x):
    r = b − A * x
    p = r
    rsold = r * transpose(r)

    do
        Ap = A * p
        alpha = rsold / (Ap * transpose(p))
        x = x + (alpha * p)
        r = r − (alpha * Ap)
        rsnew = r * transpose(r)

        residual = sqrt(rsnew)
        p = r + (rsnew / rsold) * p
        rsold = rsnew
    while(residual > 1e-8)

    return x
end
```

Pseudo-code of the conjugate gradient algorithm
for solving systems of linear equations

Data-Dependent Execution

"Iterate until converged" is an almost universal pattern

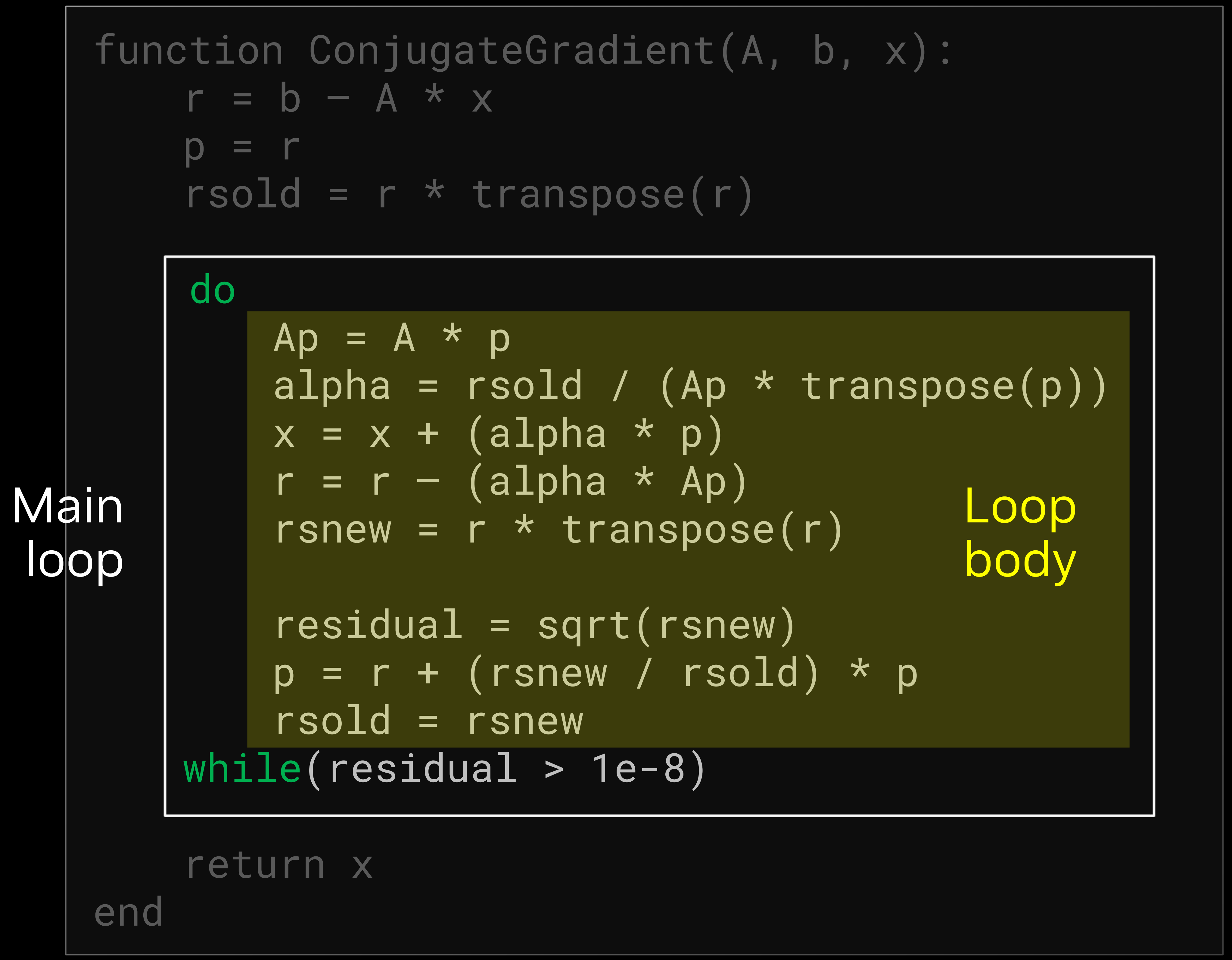
Main loop

```
function ConjugateGradient(A, b, x):
    r = b − A * x
    p = r
    rsold = r * transpose(r)

    do
        Ap = A * p
        alpha = rsold / (Ap * transpose(p))
        x = x + (alpha * p)
        r = r − (alpha * Ap)
        rsnew = r * transpose(r)

        residual = sqrt(rsnew)
        p = r + (rsnew / rsold) * p
        rsold = rsnew
    while(residual > 1e-8)

    return x
end
```

Pseudo-code of the conjugate gradient algorithm for solving systems of linear equations

Data-Dependent Execution

"Iterate until converged" is an almost universal pattern



```
function ConjugateGradient(A, b, x):
    r = b − A * x
    p = r
    rsold = r * transpose(r)

    do
        Ap = A * p
        alpha = rsold / (Ap * transpose(p))
        x = x + (alpha * p)
        r = r − (alpha * Ap)
        rsnew = r * transpose(r)

        residual = sqrt(rsnew)
        p = r + (rsnew / rsold) * p
        rsold = rsnew
    while(residual > 1e-8)

    return x
end
```

Pseudo-code of the conjugate gradient algorithm for solving systems of linear equations

Data-Dependent Execution

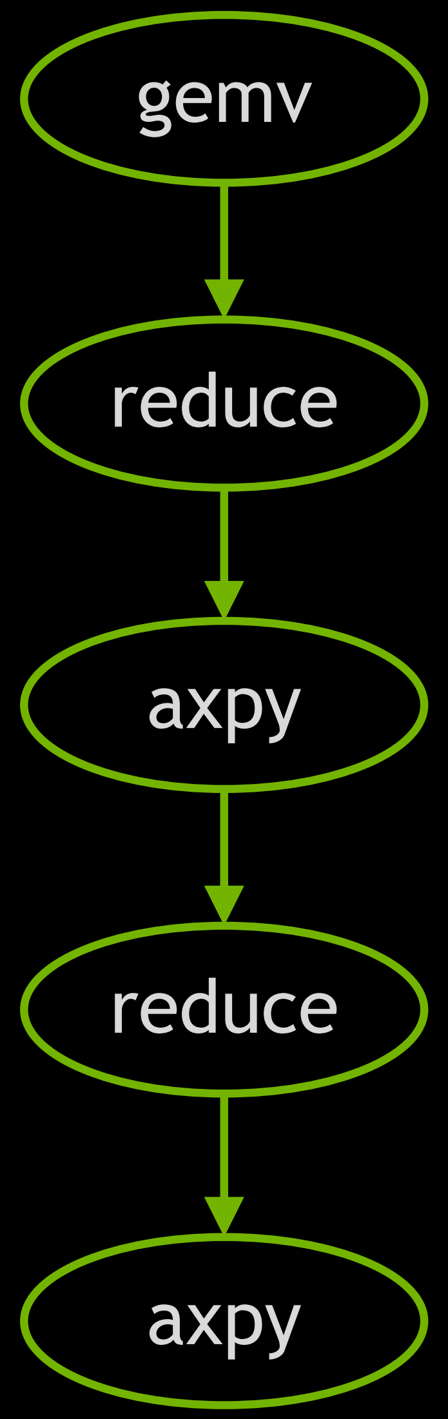
"Iterate until converged" is an almost universal pattern

```
function ConjugateGradient(A, b, x):
    r = b − A * x
    p = r
    rsold = r * transpose(r)

    do
        Ap = A * p
        alpha = rsold / (Ap * transpose(p))
        x = x + (alpha * p)
        r = r − (alpha * Ap)
        rsnew = r * transpose(r)

        residual = sqrt(rsnew)
        p = r + (rsnew / rsold) * p
        rsold = rsnew
    while(residual > 1e-8)

    return x
end
```

Main loop

Loop body

Pseudo-code of the conjugate gradient algorithm for solving systems of linear equations



Convert loop body into a task graph

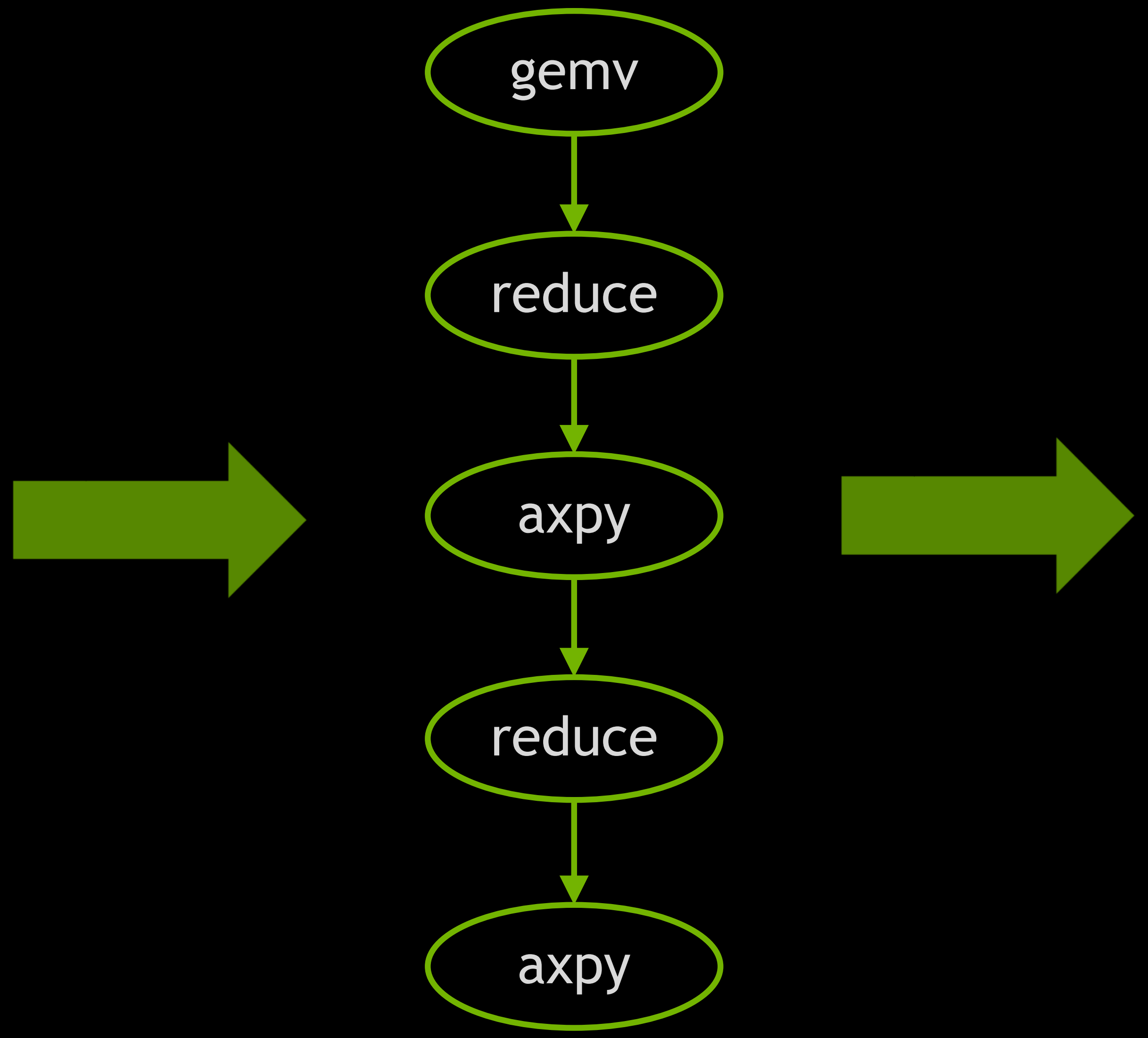
Data-Dependent Execution

"Iterate until converged" is an almost universal pattern

Main loop

```
function ConjugateGradient(A, b, x):
    r = b − A * x
    p = r
    rsold = r * transpose(r)

    do
        Ap = A * p
        alpha = rsold / (Ap * transpose(p))
        x = x + (alpha * p)
        r = r − (alpha * Ap)
        rsnew = r * transpose(r)             Loop
                                             body
        residual = sqrt(rsnew)
        p = r + (rsnew / rsold) * p
        rsold = rsnew
    while(residual > 1e-8)

    return x
end
```

Pseudo-code of the conjugate gradient algorithm for solving systems of linear equations



Convert loop body into a task graph

```
function ConjugateGradient(A, b, x):
    r = b − A * x
    p = r
    rsold = r * transpose(r)

    do


        launch_graph(A, x, r, p, rsold)


    while(residual > 1e-8)

    return x
end
```

Task graph launch optimizes loop body execution

Data-Dependent Execution

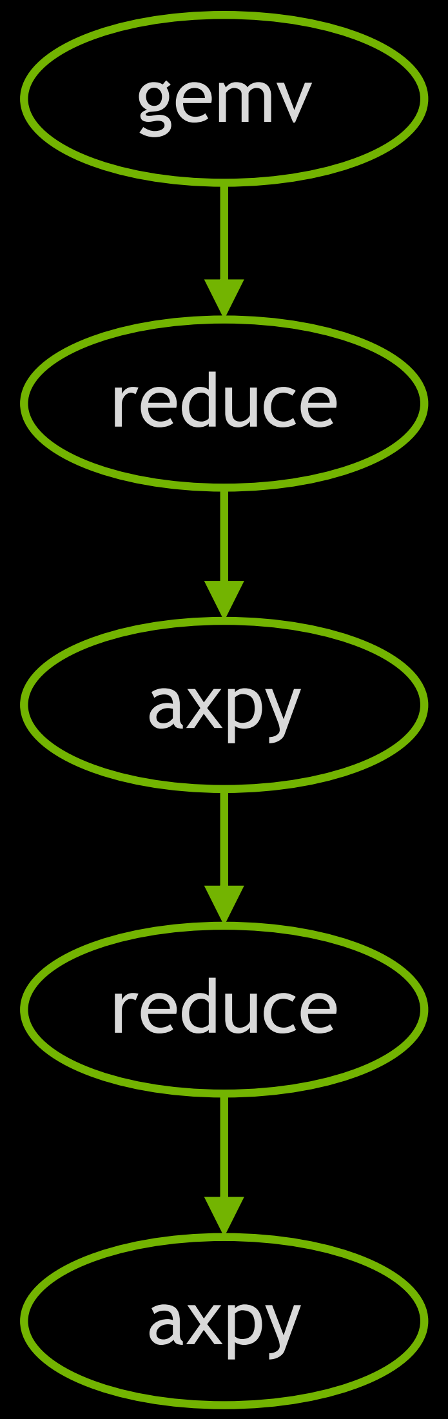
"Iterate until converged" is an almost universal pattern

Main loop

Loop body

```
function ConjugateGradient(A, b, x):
    r = b − A * x
    p = r
    rsold = r * transpose(r)

    do
        Ap = A * p
        alpha = rsold / (Ap * transpose(p))
        x = x + (alpha * p)
        r = r − (alpha * Ap)
        rsnew = r * transpose(r)

        residual = sqrt(rsnew)
        p = r + (rsnew / rsold) * p
        rsold = rsnew
    while(residual > 1e-8)

    return x
end
```

Pseudo-code of the conjugate gradient algorithm for solving systems of linear equations



Convert loop body into a task graph

```
function ConjugateGradient(A, b, x):
    r = b − A * x
    p = r
    rsold = r * transpose(r)

    do



        launch_graph(A, x, r, p, rsold)



    while(residual > 1e-8)

    return x
end
```

Task graph launch optimizes loop body execution

Data-Dependent Execution

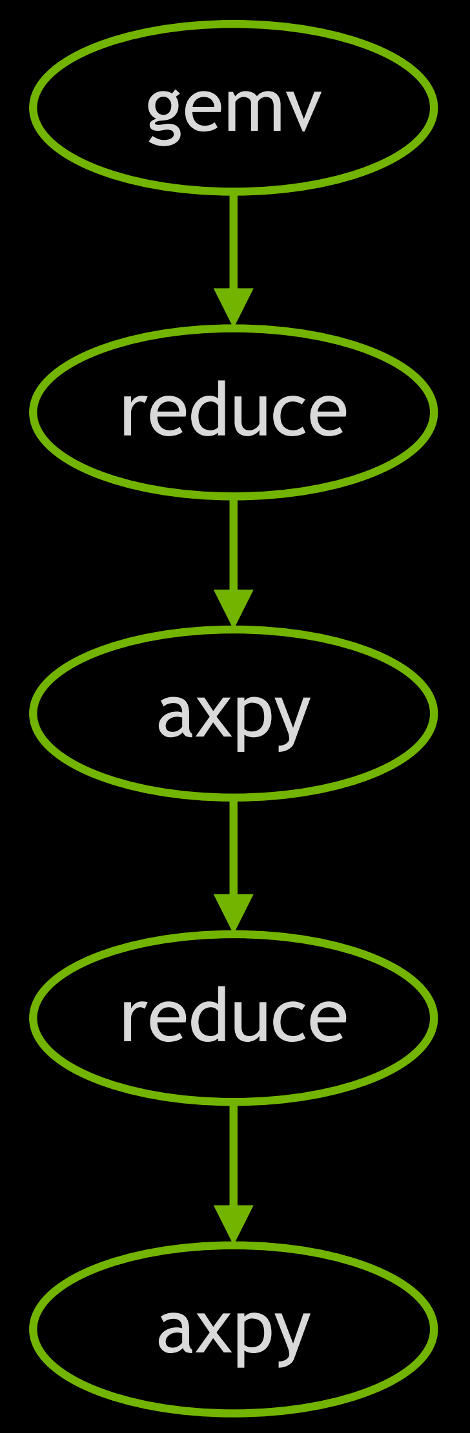
"Iterate until converged" is an almost universal pattern

Main loop

```
function ConjugateGradient(A, b, x):
    r = b − A * x
    p = r
    rsold = r * transpose(r)

    do
        Ap = A * p
        alpha = rsold / (Ap * transpose(p))
        x = x + (alpha * p)
        r = r − (alpha * Ap)
        rsnew = r * transpose(r)

        residual = sqrt(rsnew)
        p = r + (rsnew / rsold) * p
        rsold = rsnew
    while(residual > 1e-8)

    return x
end
```

Loop body

Pseudo-code of the conjugate gradient algorithm for solving systems of linear equations



Convert loop body into a task graph

```
function ConjugateGradient(A, b, x):
    r = b − A * x
    p = r
    rsold = r * transpose(r)

    do



        launch_graph(A, x, r, p, rsold)

        synchronize()
        residual = copy_from_gpu()

    while(residual > 1e-8)

    return x
end
```

Task graph launch optimizes loop body execution but then must return to CPU to evaluate loop again

Data-Dependent Execution On The GPU

"Iterate until converged" is an almost universal pattern

```
function ConjugateGradient(A, b, x):
    r = b − A * x
    p = r
    rsold = r * transpose(r)

    do
        Ap = A * p
        alpha = rsold / (Ap * transpose(p))
        x = x + (alpha * p)
        r = r − (alpha * Ap)
        rsnew = r * transpose(r)

        residual = sqrt(rsnew)
        p = r + (rsnew / rsold) * p
        rsold = rsnew
    while(residual > 1e-8)

    return x
end
```

Main loop

Pseudo-code of the conjugate gradient algorithm for solving systems of linear equations

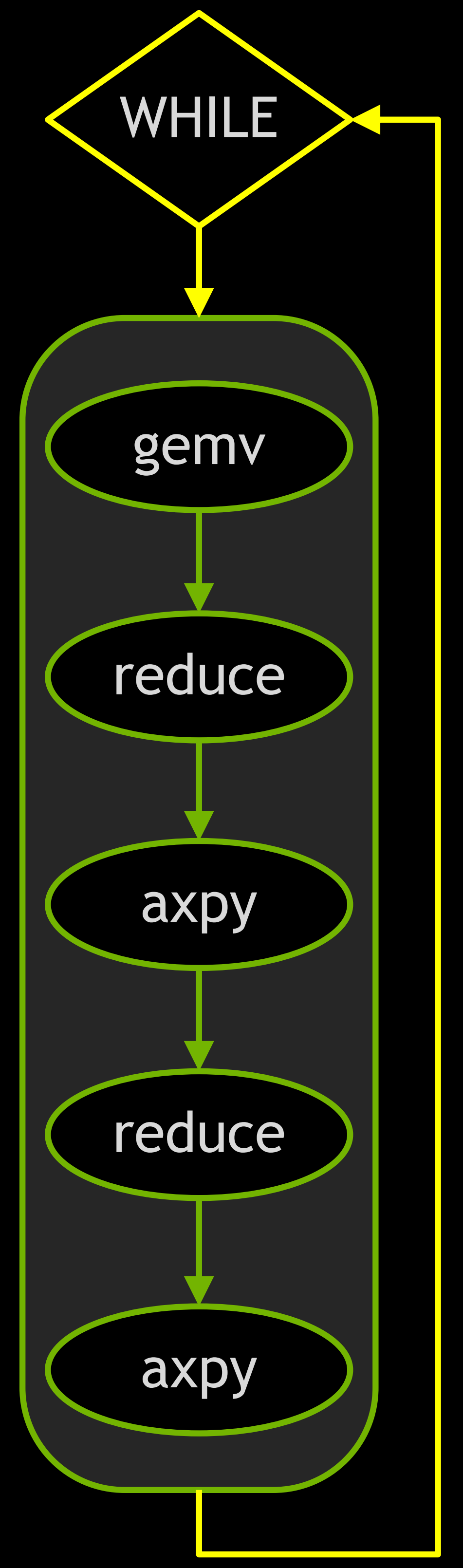
Data-Dependent Execution On The GPU

"Iterate until converged" is an almost universal pattern

Main loop

```
function ConjugateGradient(A, b, x):
    r = b - A * x
    p = r
    rsold = r * transpose(r)

    do
        Ap = A * p
        alpha = rsold / (Ap * transpose(p))
        x = x + (alpha * p)
        r = r - (alpha * Ap)
        rsnew = r * transpose(r)

        residual = sqrt(rsnew)
        p = r + (rsnew / rsold) * p
        rsold = rsnew
    while(residual > 1e-8)

    return x
end
```

Pseudo-code of the conjugate gradient algorithm for solving systems of linear equations



Convert **entire loop** into a task graph using new "**conditional nodes**"

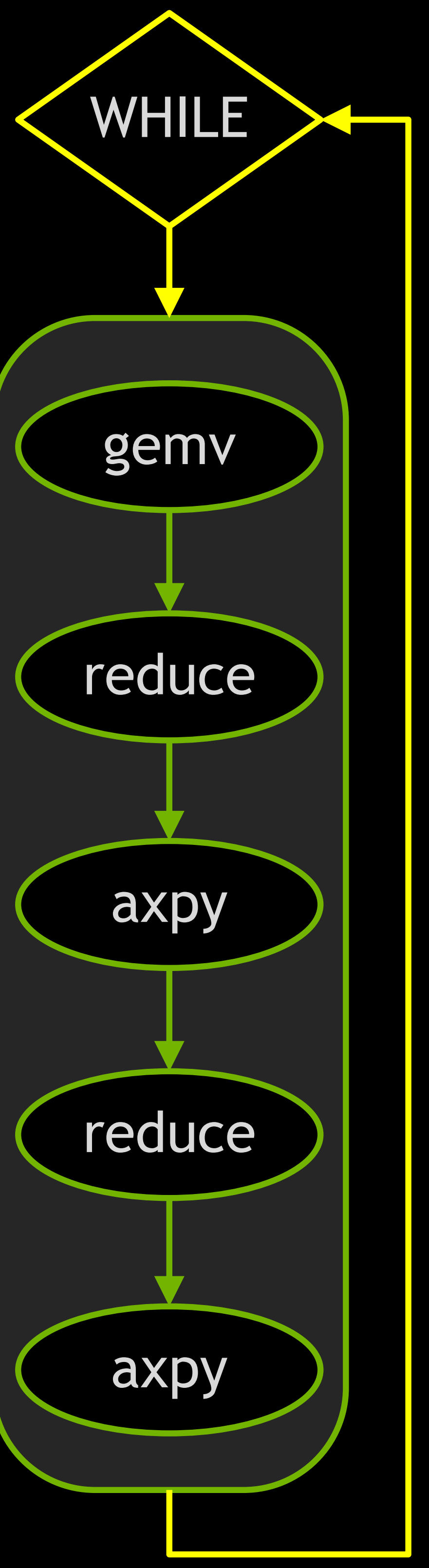
Data-Dependent Execution On The GPU

"Iterate until converged" is an almost universal pattern

Main loop

```
function ConjugateGradient(A, b, x):
    r = b − A * x
    p = r
    rsold = r * transpose(r)

    do
        Ap = A * p
        alpha = rsold / (Ap * transpose(p))
        x = x + (alpha * p)
        r = r − (alpha * Ap)
        rsnew = r * transpose(r)

        residual = sqrt(rsnew)
        p = r + (rsnew / rsold) * p
        rsold = rsnew
    while(residual > 1e-8)

    return x
end
```

Pseudo-code of the conjugate gradient algorithm for solving systems of linear equations



Convert **entire loop** into a task graph using new "**conditional nodes**"

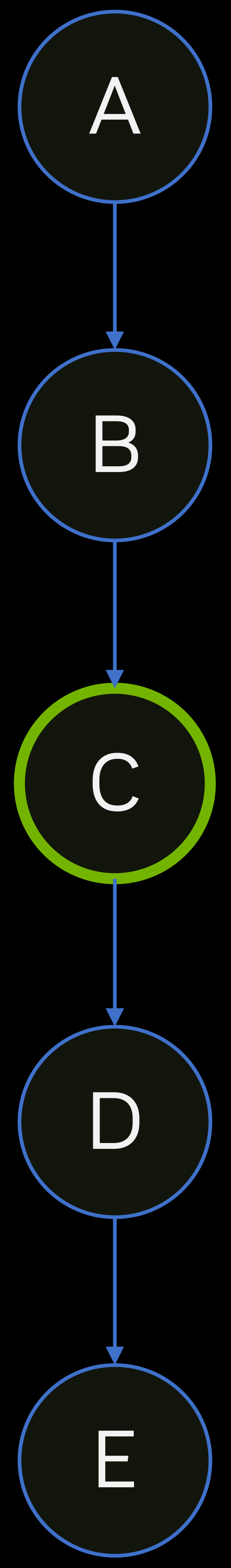
```
function ConjugateGradient(A, b, x):
    r = b − A * x
    p = r
    rsold = r * transpose(r)



    launch_conditional_graph(A, x, r, p,
                             rsold, 1e-8)



    return x
end
```

Entire CG solve runs to completion on GPU **using just one single graph launch**

Conditional Graph Nodes

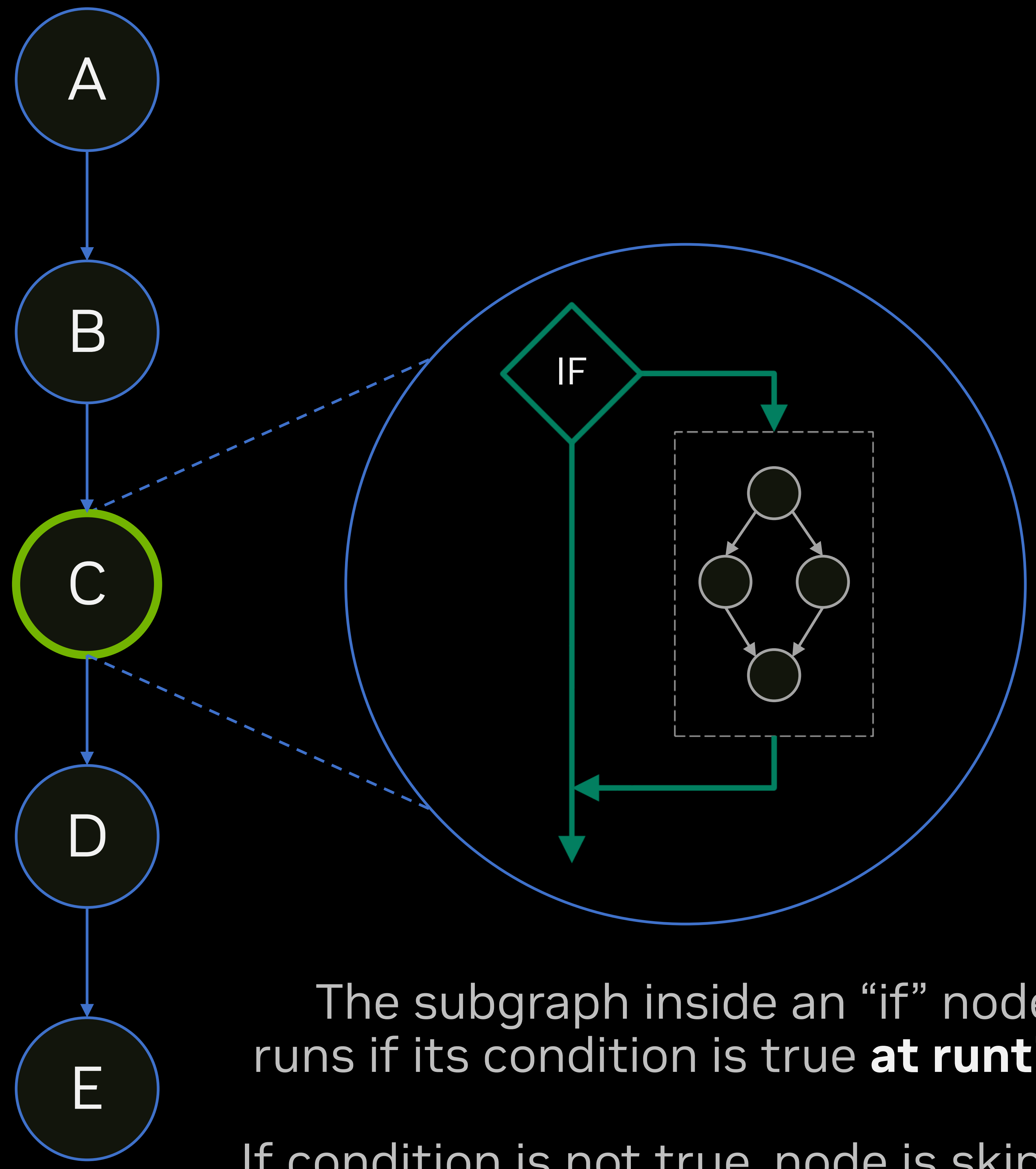
A new type of graph node that contains a subgraph which runs if() or while() a condition is true

A conditional node is just another type of graph node, so graph structure is preserved



Conditional Graph Nodes

A new type of graph node that contains a subgraph which runs if() or while() a condition is true



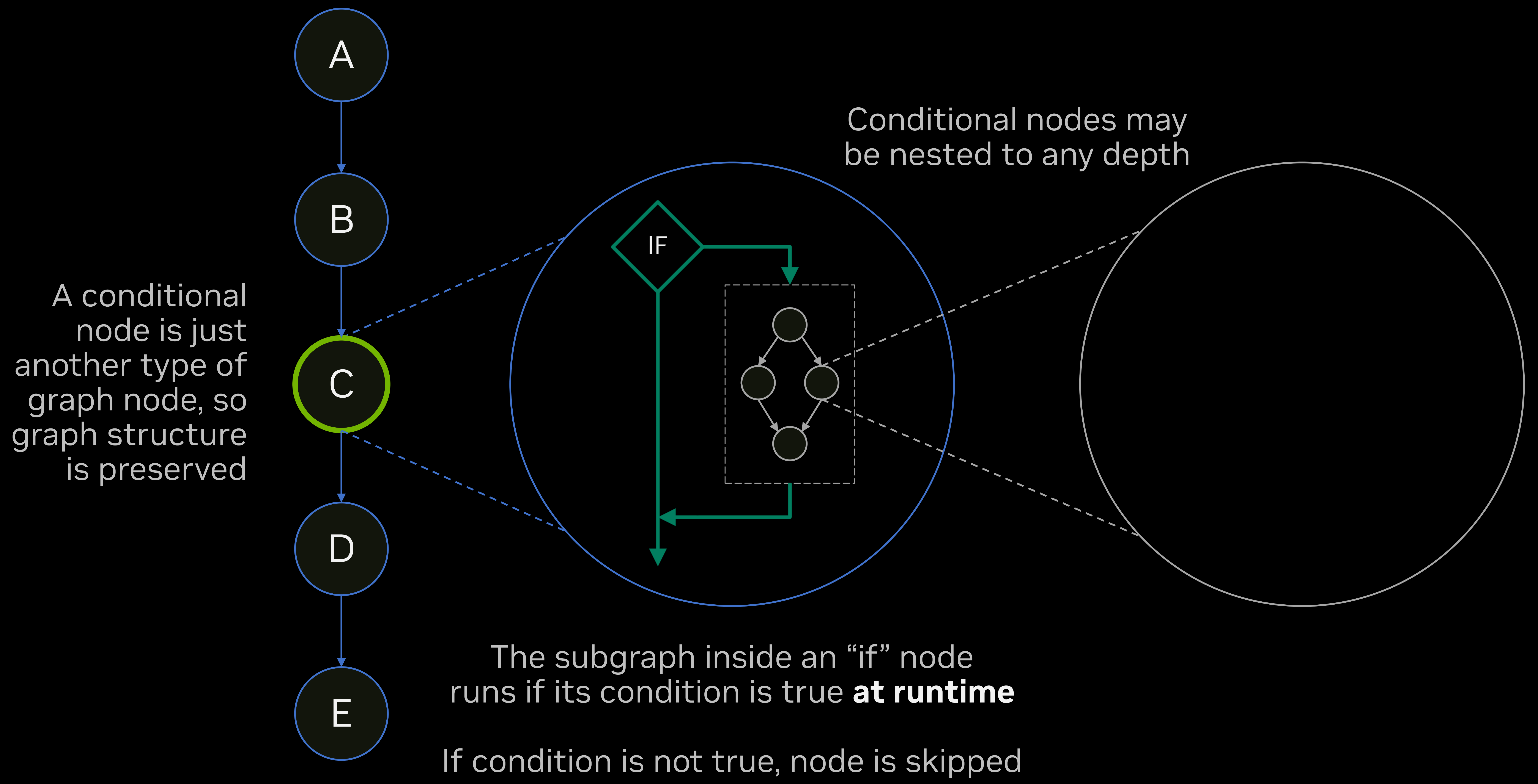
A conditional node is just another type of graph node, so graph structure is preserved

The subgraph inside an "if" node runs if its condition is true **at runtime**

If condition is not true, node is skipped

Conditional Graph Nodes

A new type of graph node that contains a subgraph which runs if() or while() a condition is true

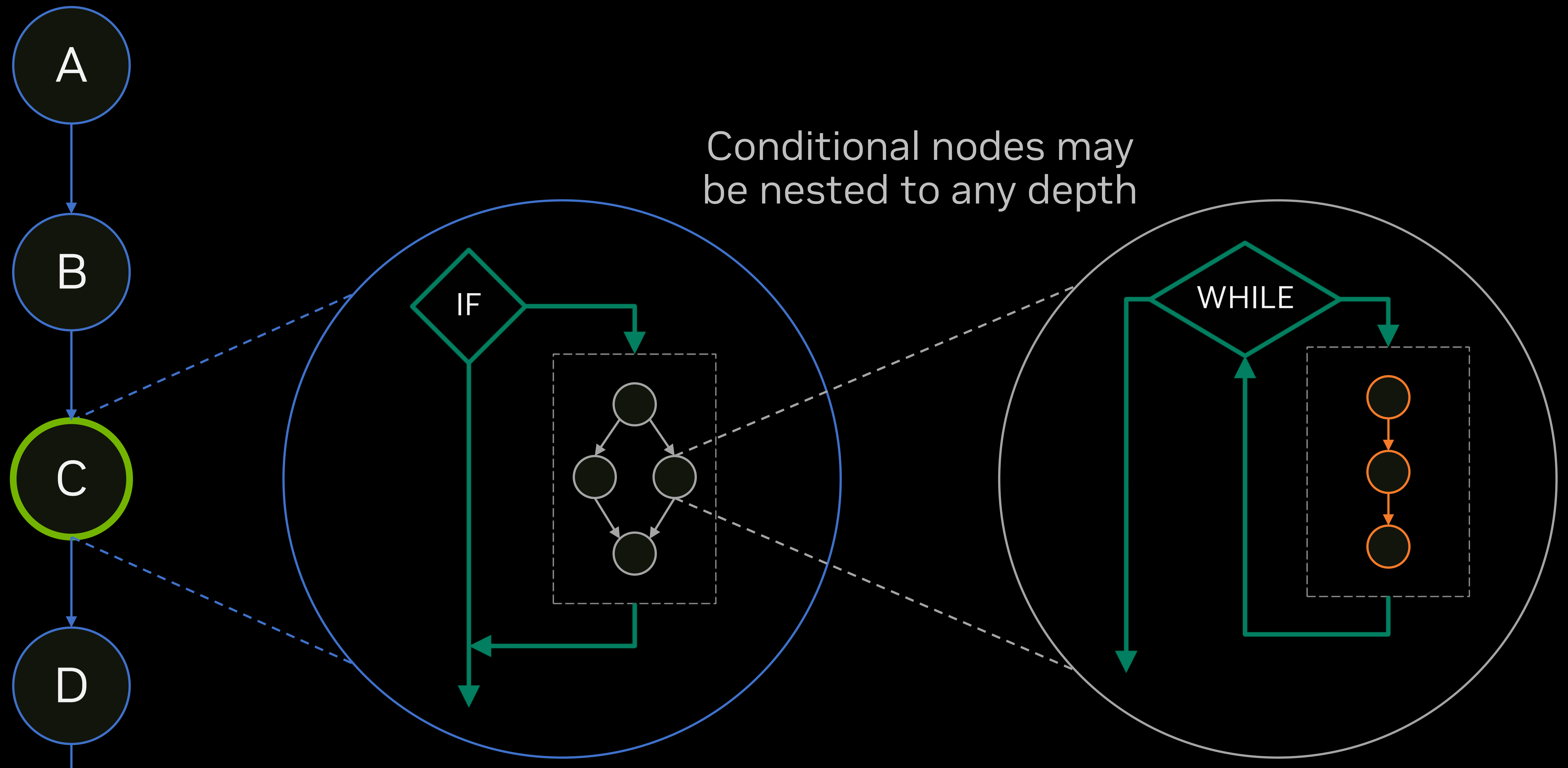

A
B
C
D
E
IF
A conditional node is just another type of graph node, so graph structure is preserved
Conditional nodes may be nested to any depth

The subgraph inside an "if" node runs if its condition is true **at runtime**

If condition is not true, node is skipped

Conditional Graph Nodes

A new type of graph node that contains a subgraph which runs if() or while() a condition is true



A conditional node is just another type of graph node, so graph structure is preserved

Conditional nodes may be nested to any depth

The subgraph inside an "if" node runs if its condition is true **at runtime**

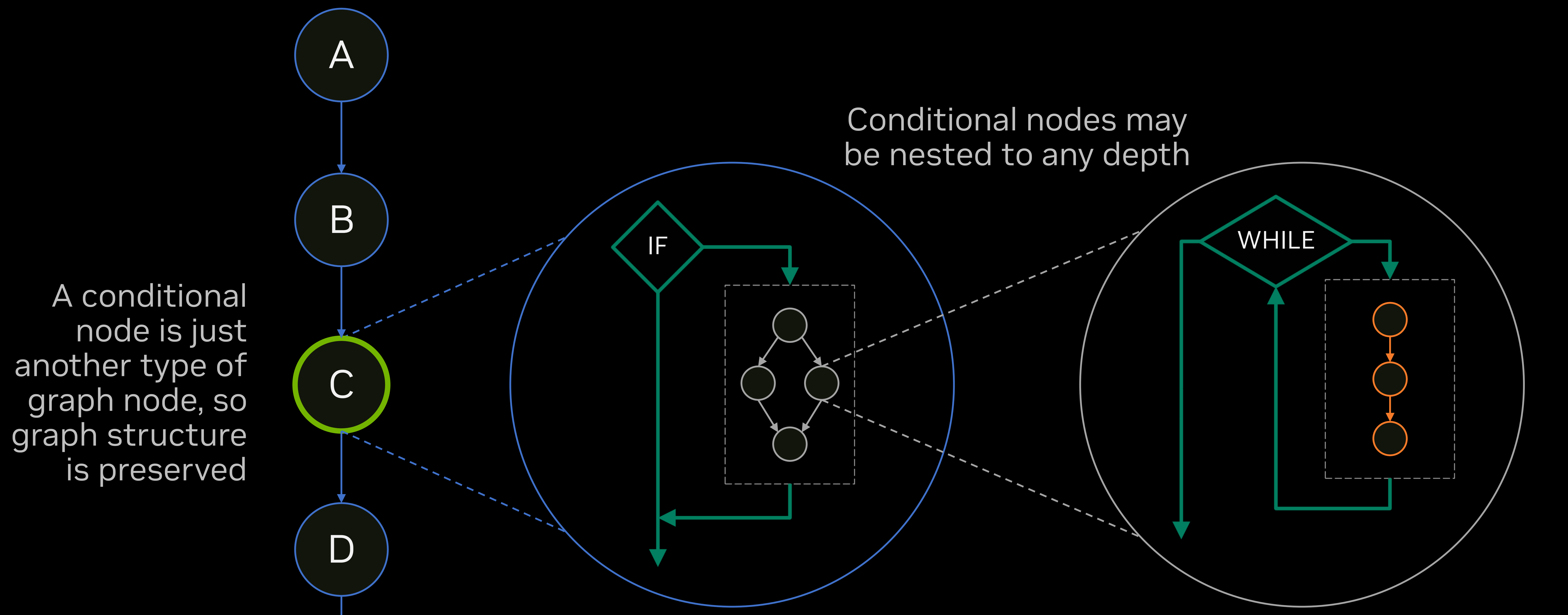
If condition is not true, node is skipped

"while" is like "if" except it **re-evaluates** the while on completion

This allows the while body **to set its own condition**

Conditional Graph Nodes

A new type of graph node that contains a subgraph which runs if() or while() a condition is true

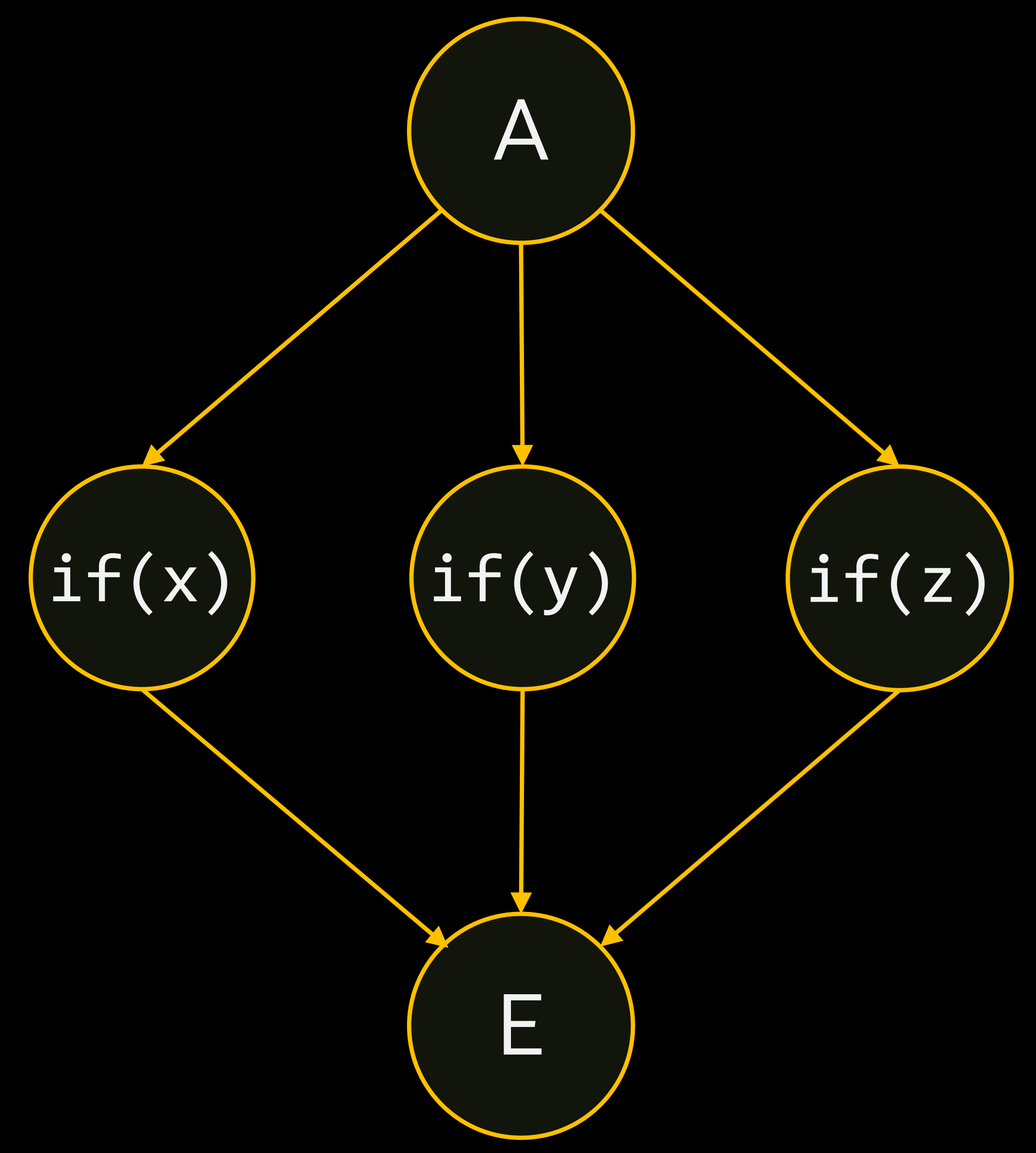

A
B
C
D
E
A conditional node is just another type of graph node, so graph structure is preserved
IF
Conditional nodes may be nested to any depth
WHILE

The subgraph inside an "if" node runs if its condition is true **at runtime**

If condition is not true, node is skipped

"while" is like "if" except it **re-evaluates** the while on completion

This allows the while body **to set its own condition**



Conditional nodes are just graph nodes so multiple "if"s can function like "switch"

Not limited to a single true condition: **zero, one, some or all conditions may run**

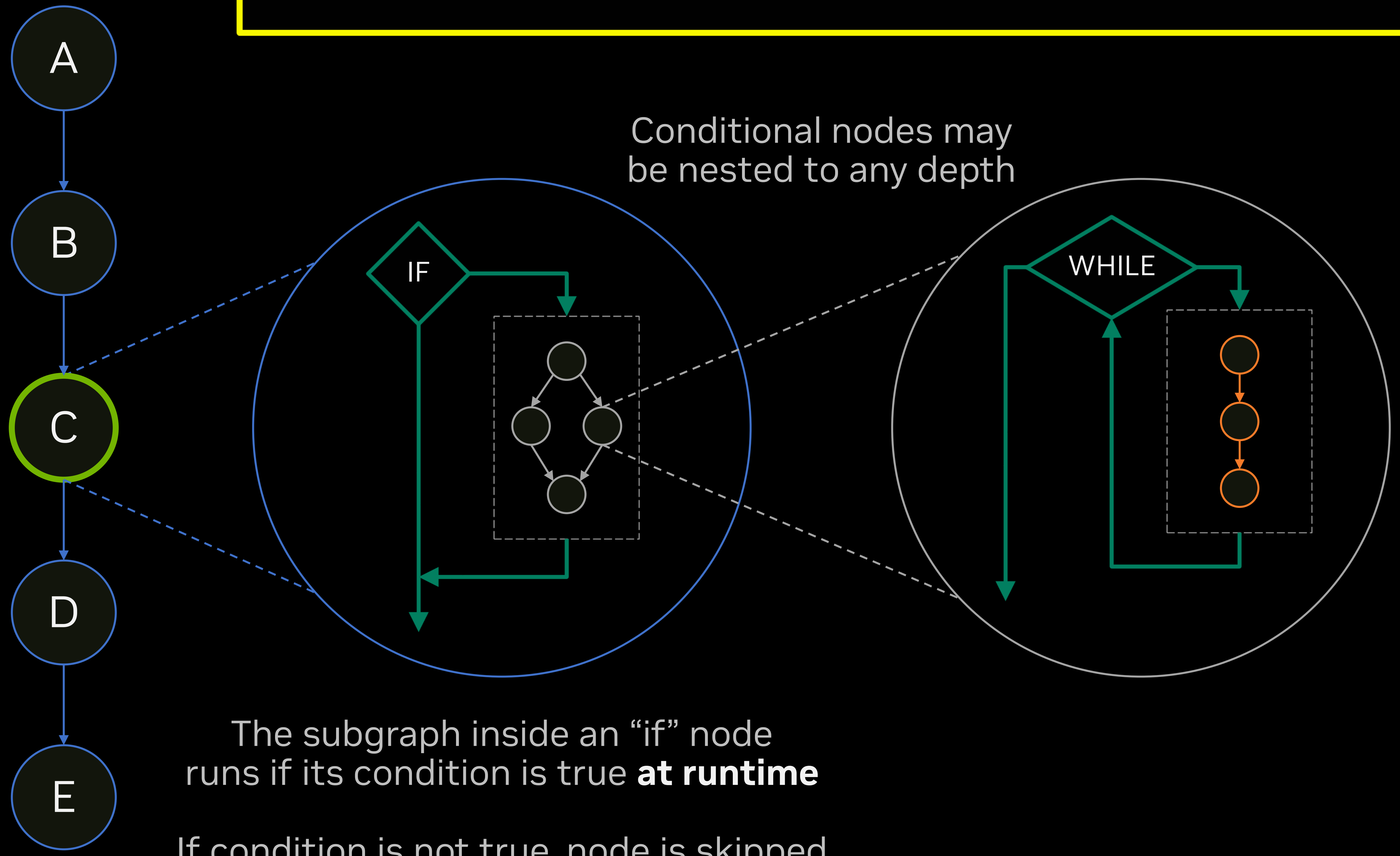
Conditional Graph Nodes

A new type of graph node that contains a subgraph which runs if() or while() a condition is true

Conditional graph nodes available from CUDA 12.4

A conditional node is just another type of graph node, so graph structure is preserved

Conditional nodes may be nested to any depth

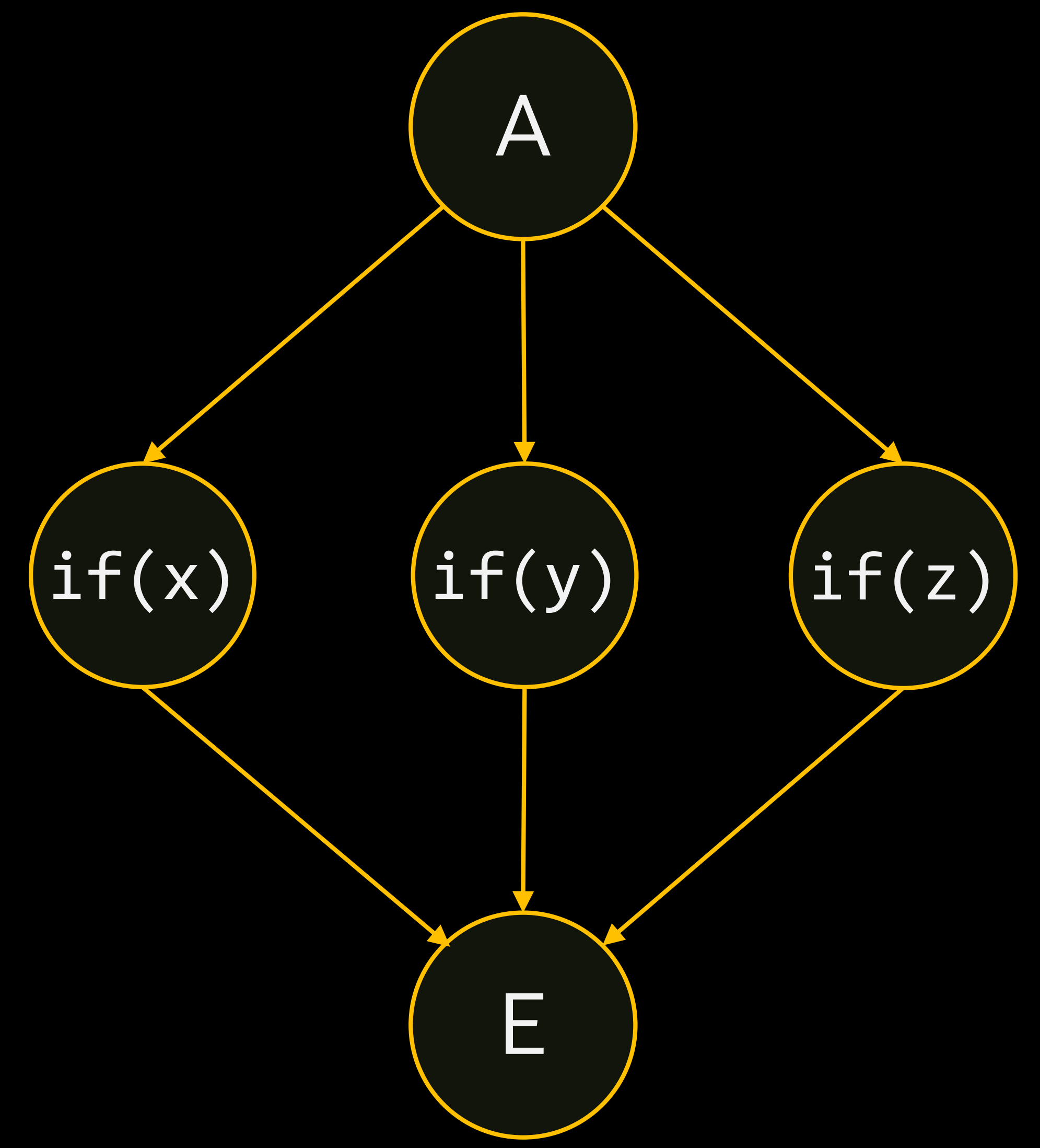


The subgraph inside an "if" node runs if its condition is true **at runtime**

If condition is not true, node is skipped

"while" is like "if" except it **re-evaluates** the while on completion

This allows the while body **to set its own condition**



Conditional nodes are just graph nodes so multiple "if"s can function like "switch"

Not limited to a single true condition: **zero, one, some or all conditions may run**

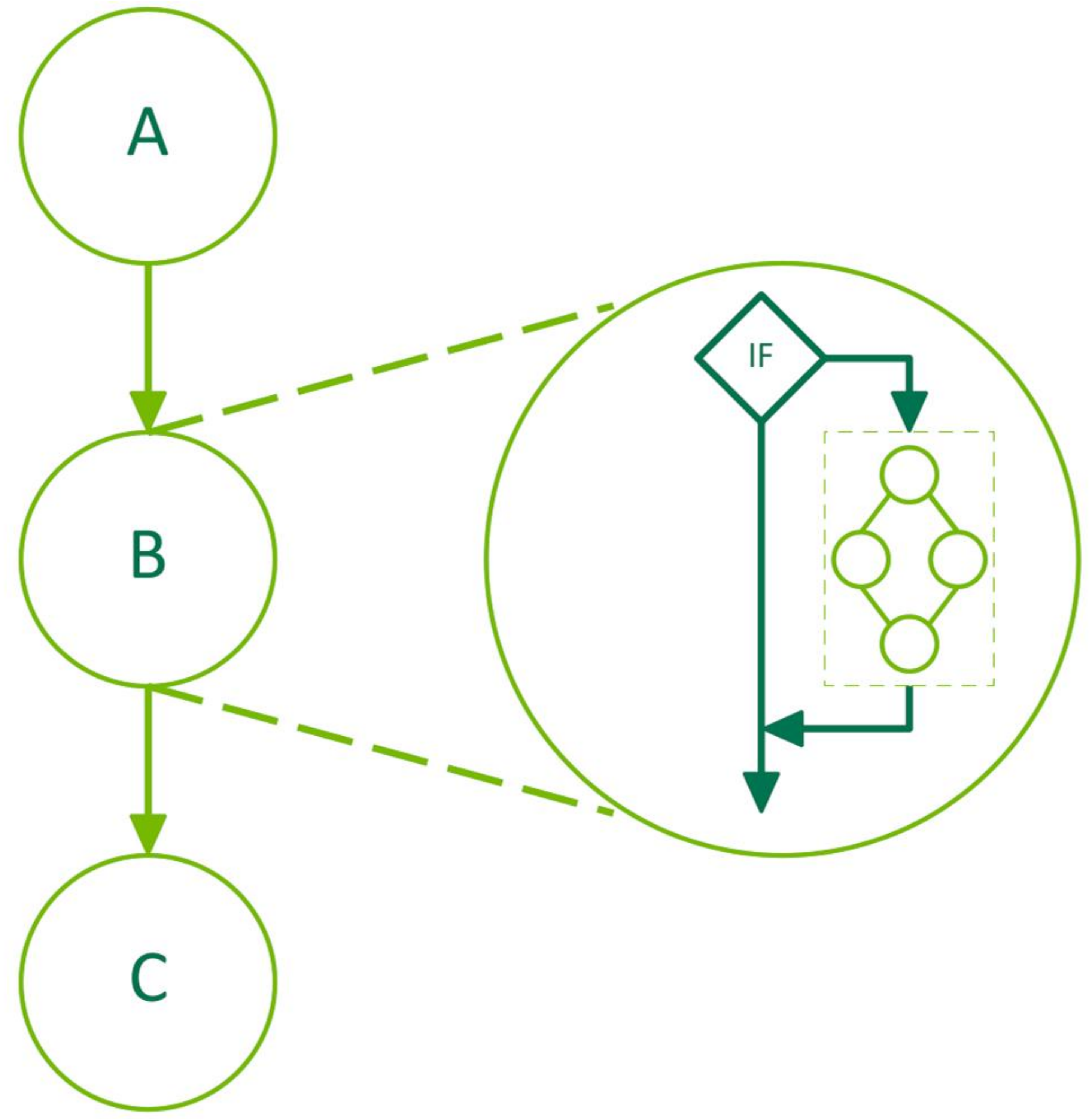
Dynamic Control Flow in CUDA Graphs with Conditional Nodes

Conditional IF Node Example

Device function for Node A:

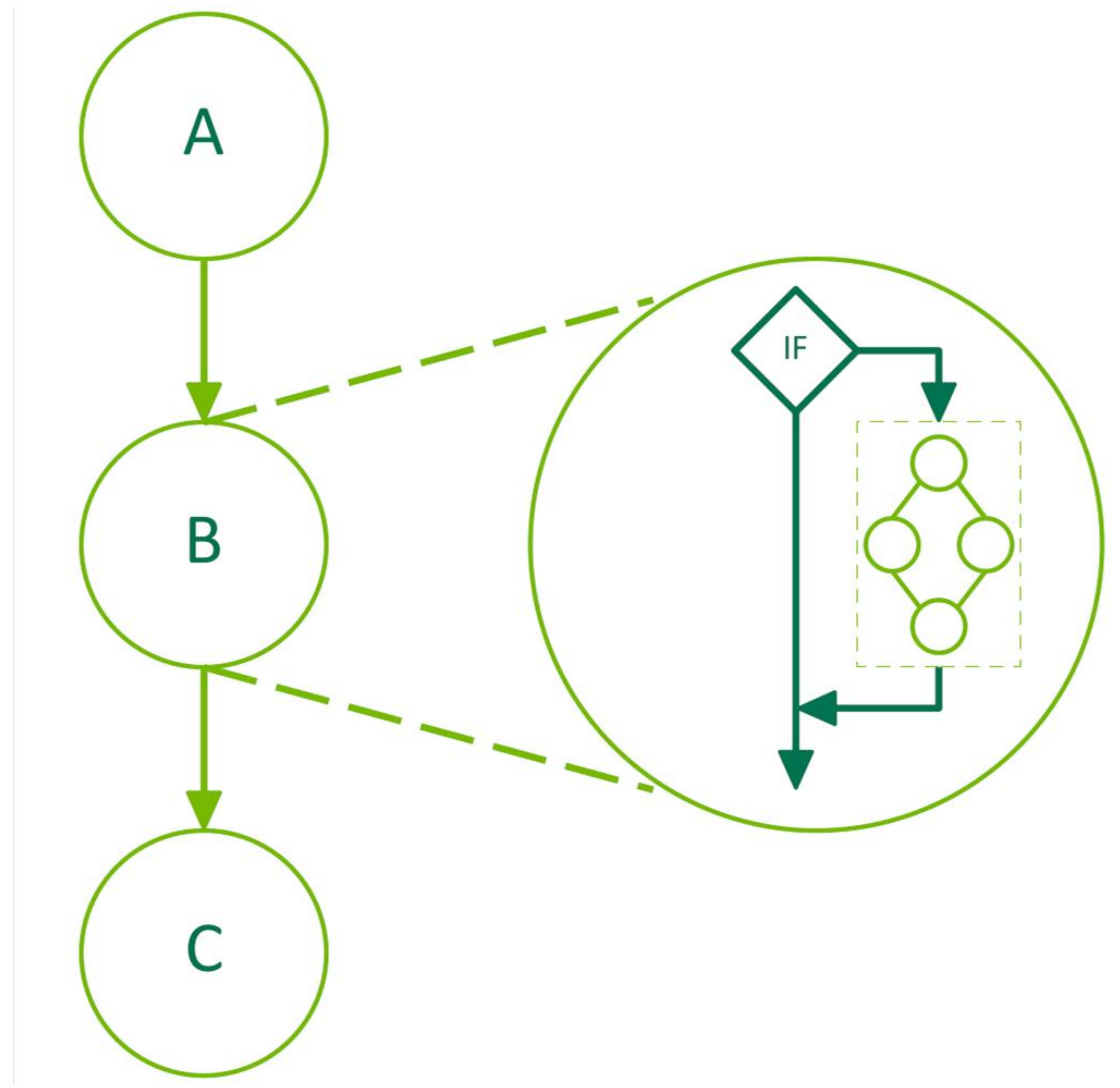
```
__global__ void nodeA(cudaGraphConditionalHandle handle, …) {
   …
   cudaGraphSetConditional(handle, value);
}
```

- Node A must set the condition before Node B is executed
- Application specific code would perform calculations and set ‘value’



Dynamic Control Flow in CUDA Graphs with Conditional Nodes

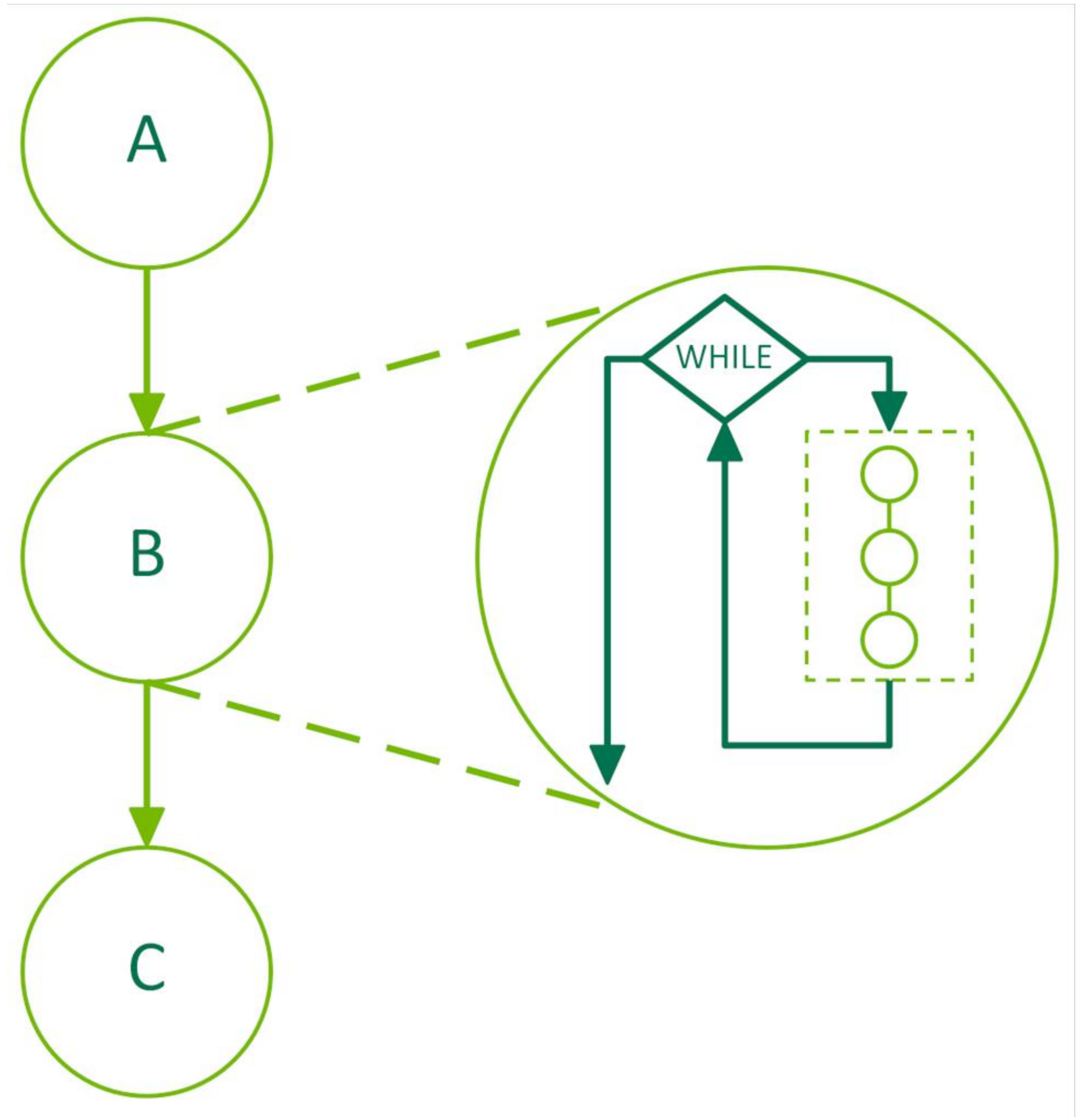
Conditional IF Node Example



```
cudaGraph_t graph;
cudaGraphCreate(&graph, 0);

cudaGraphConditionalHandle handle;
cudaGraphConditionalHandleCreate( &handle, graph );

cudaGraphAddNode( &nodeA, graph, NULL, 0, &params );    ← Parameter setup omitted for brevity

cudaGraphNodeParams cParams = { cudaGraphNodeTypeConditional };
cParams.conditional.handle  = handle;
cParams.conditional.type    = cudaGraphCondTypeIf;
cParams.conditional.size    = 1;

cudaGraphAddNode( &nodeB, graph, &nodeA, 1, &cParams );

cudaGraph_t bodyGraph = cParams.conditional.phGraph_out[0];  ← Body graph returned in params

cudaGraphAddNode( &bodyNodeA, bodyGraph, NULL, 0, &params ); ← Parameter setup omitted
```

Dynamic Control Flow in CUDA Graphs with Conditional Nodes

Conditional WHILE Nodes



3 Node Graph with a Conditional WHILE Node

- Conditional body graph is executed until the condition is zero
- Value will default to 1 to implement a 'Do-While' loop
- Conditional body graph is populated using stream capture
- Complete examples available in the CUDA Samples git repo:

  http://nv/conditionalsamples

Dynamic Control Flow in CUDA Graphs with Conditional Nodes

Conditional Nodes

```
cudaGraphConditionalHandle handle;
cudaGraphConditionalHandleCreate( &handle, graph, 1, cudaGraphCondAssignDefault );

cudaGraphNodeParams cParams = { cudaGraphNodeTypeConditional };
cParams.conditional.handle = handle;
cParams.conditional.type   = cudaGraphCondTypeWhile;
cParams.conditional.size   = 1;
cudaGraphAddNode( &nodeB, graph, &nodeA, 1, &cParams );

cudaGraph_t  bodyGraph = cParams.conditional.phGraph_out[0];
cudaStreamCreate( &captureStream );

cudaStreamBeginCaptureToGraph( captureStream, bodyGraph, … );

loopKernel<<<1, 1, 0, captureStream>>>(handle, …);

cudaStreamEndCapture(captureStream, nullptr);
```