Democratizing High-Performance DSL Development with BuildIt

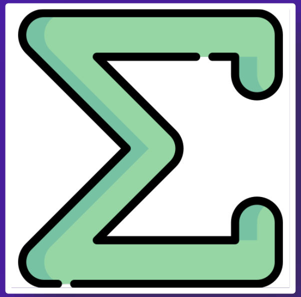
Ajay Brahmakshatriya

CSAIL, MIT

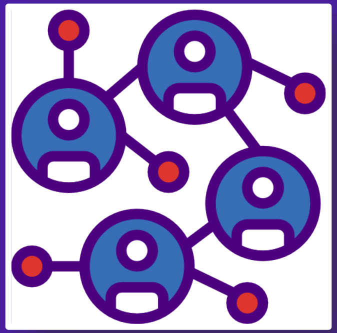
12th November 2024

COMMIT has built a lot of DSLs

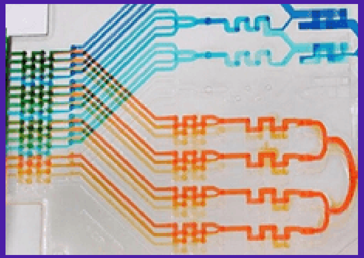
Milk

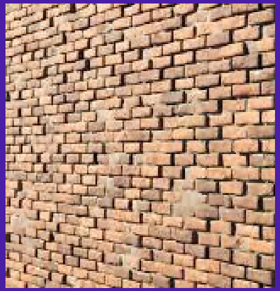

TACO

Finch


GraphIt

SEQ

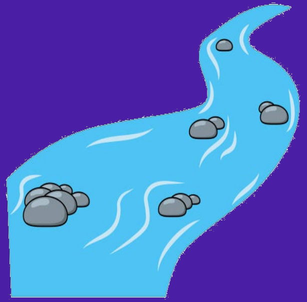

BioStream

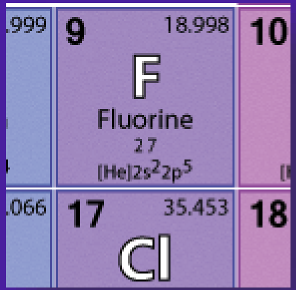

PetaBricks

SimIt

CoLa

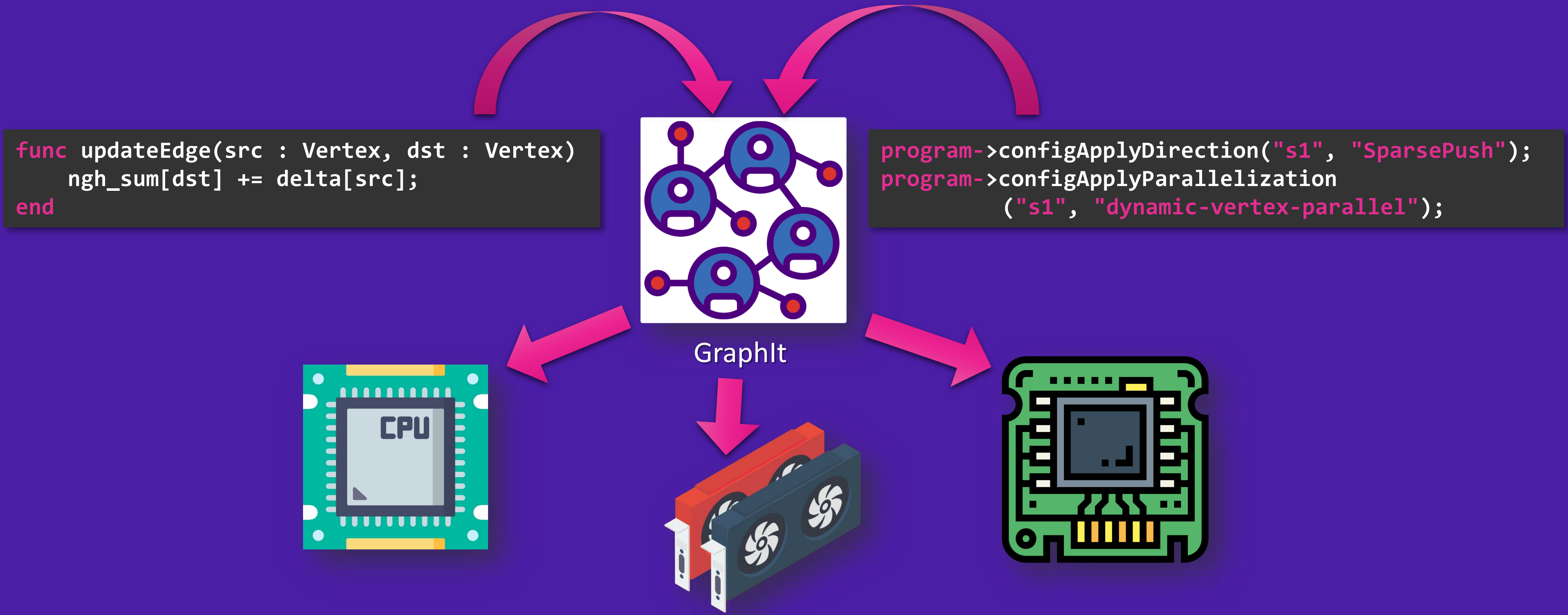
Tiramisu


StreamIt



Halide

DSLs offer control to the user



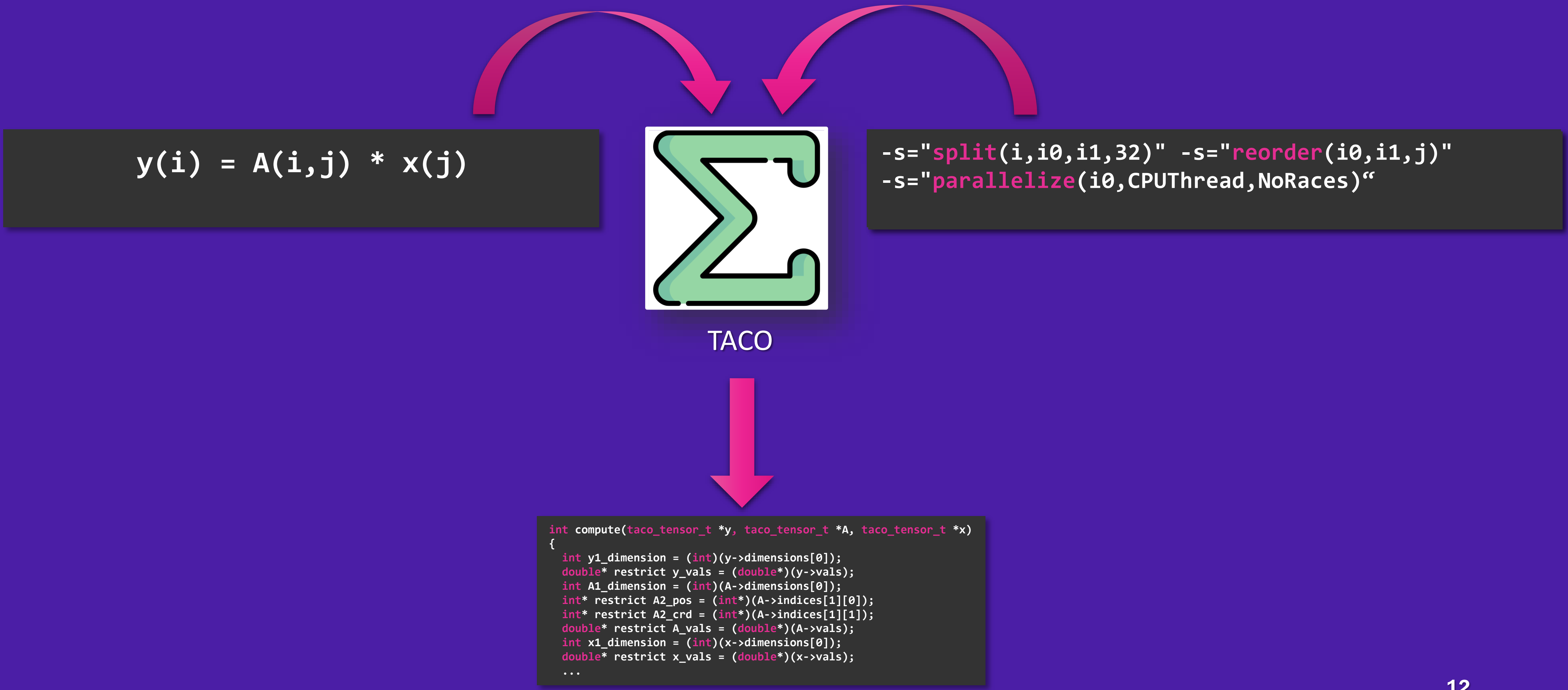
```
func updateEdge(src : Vertex, dst : Vertex)
    ngh_sum[dst] += delta[src];
end
```

```
program->configApplyDirection("s1", "SparsePush");
program->configApplyParallelization
        ("s1", "dynamic-vertex-parallel");
```

[1] **Compiling Graph Applications for GPUs with GraphIt**
Ajay Brahmakshatriya, Yunming Zhang, Changwan Hong, Shoaib Kamil, Julian Shun, Saman Amarasinghe

[2] **Taming the Zoo: A Unified Graph Compiler Framework for Novel Architectures**
Ajay Brahmakshatriya, Emily Furst, Victor Yang, Claire Hsu, Changwan Hong, Max Ruttenberg, Yunming Zhang, Tommy Jung, Dustin Richmond, Michael Taylor, Julian Shun, Mark Oskin, Daniel Sanchez, Saman Amarasinghe

DSLs offer control to the user



```
y(i) = A(i,j) * x(j)
```

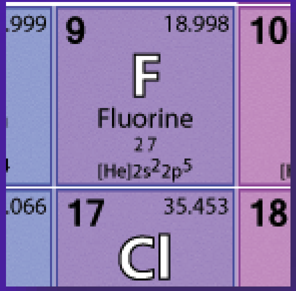
```
-s="split(i,i0,i1,32)" -s="reorder(i0,i1,j)"
-s="parallelize(i0,CPUThread,NoRaces)“
```

TACO

```
int compute(taco_tensor_t *y, taco_tensor_t *A, taco_tensor_t *x)
{
  int y1_dimension = (int)(y->dimensions[0]);
  double* restrict y_vals = (double*)(y->vals);
  int A1_dimension = (int)(A->dimensions[0]);
  int* restrict A2_pos = (int*)(A->indices[1][0]);
  int* restrict A2_crd = (int*)(A->indices[1][1]);
  double* restrict A_vals = (double*)(A->vals);
  int x1_dimension = (int)(x->dimensions[0]);
  double* restrict x_vals = (double*)(x->vals);
  ...
```

DSLs offer control to the user

```
blur_x(x, y) = input(x-1, y) + input(x, y) + input(x+1, y)/3;
blur_y(x, y) = blur_x(x, y-1) + blur_x(x, y) + blur_x(x, y+1)/3;
```



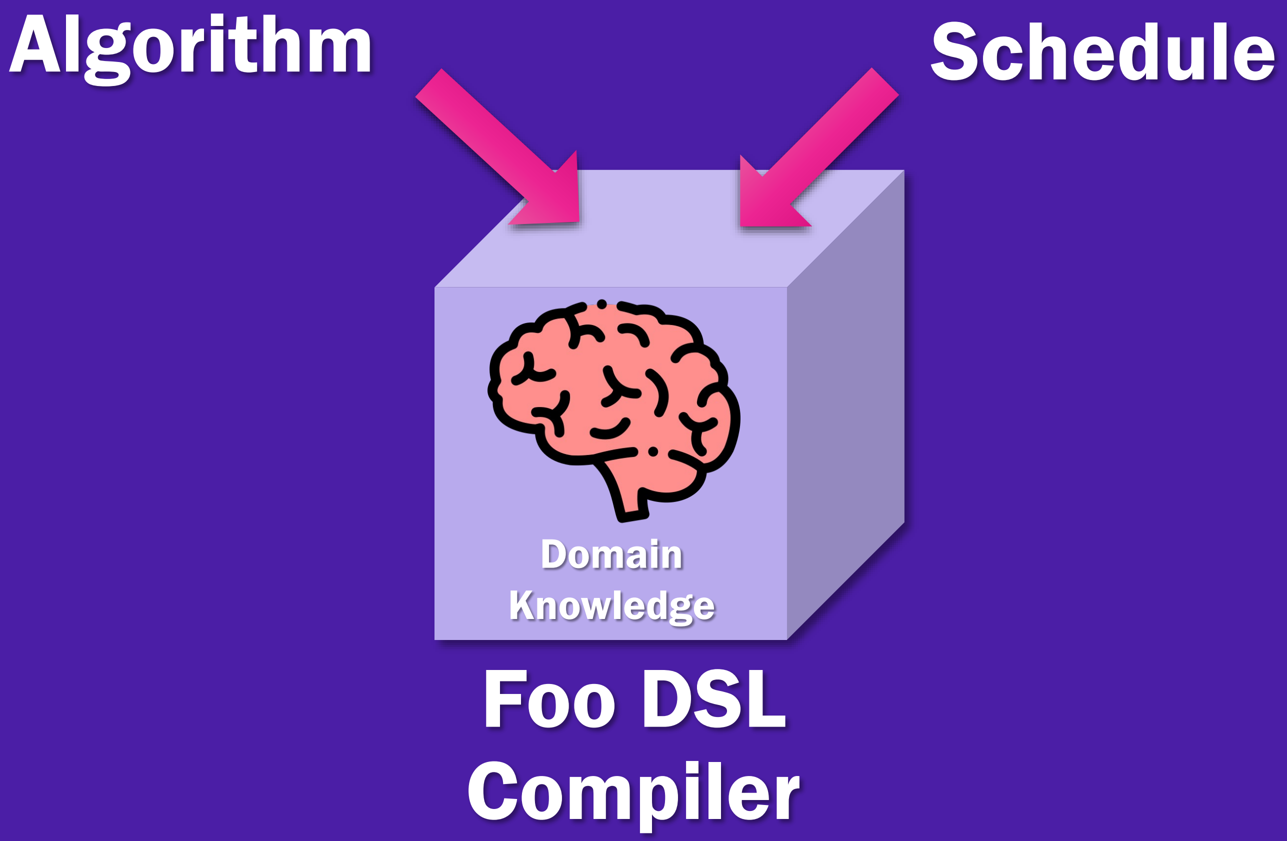
Halide

```
producer.compute_at(consumer, y);
producer.trace_stores();
consumer.trace_stores();
```

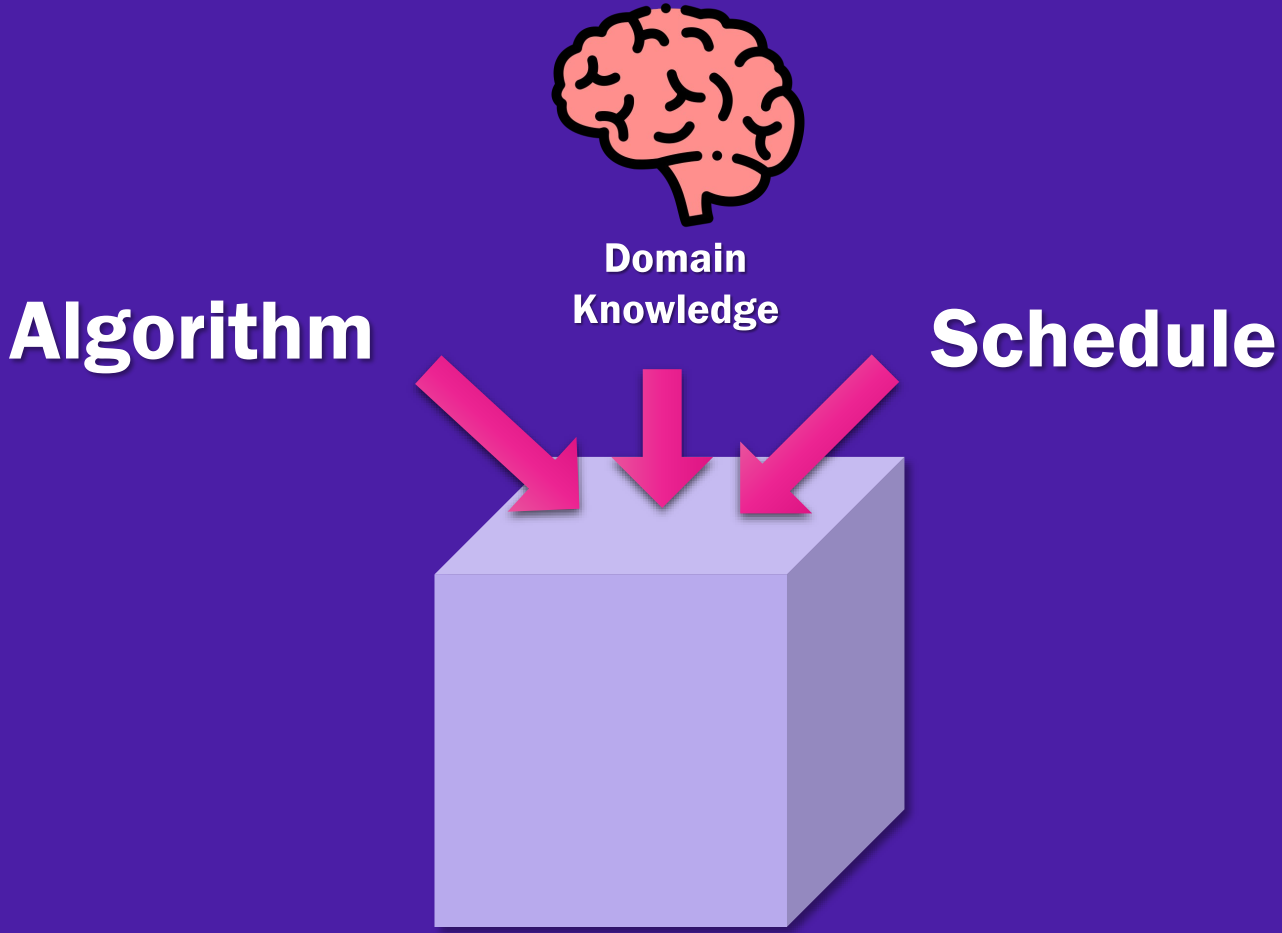
Writing compilers is *Hard*!

DSL	LoC (Total)
GraphIt	100,362
TACO	71,684
Halide	322,822
Tiramisu	349,661

The ONE DSL compiler to rule them all



The ONE DSL compiler to rule them all



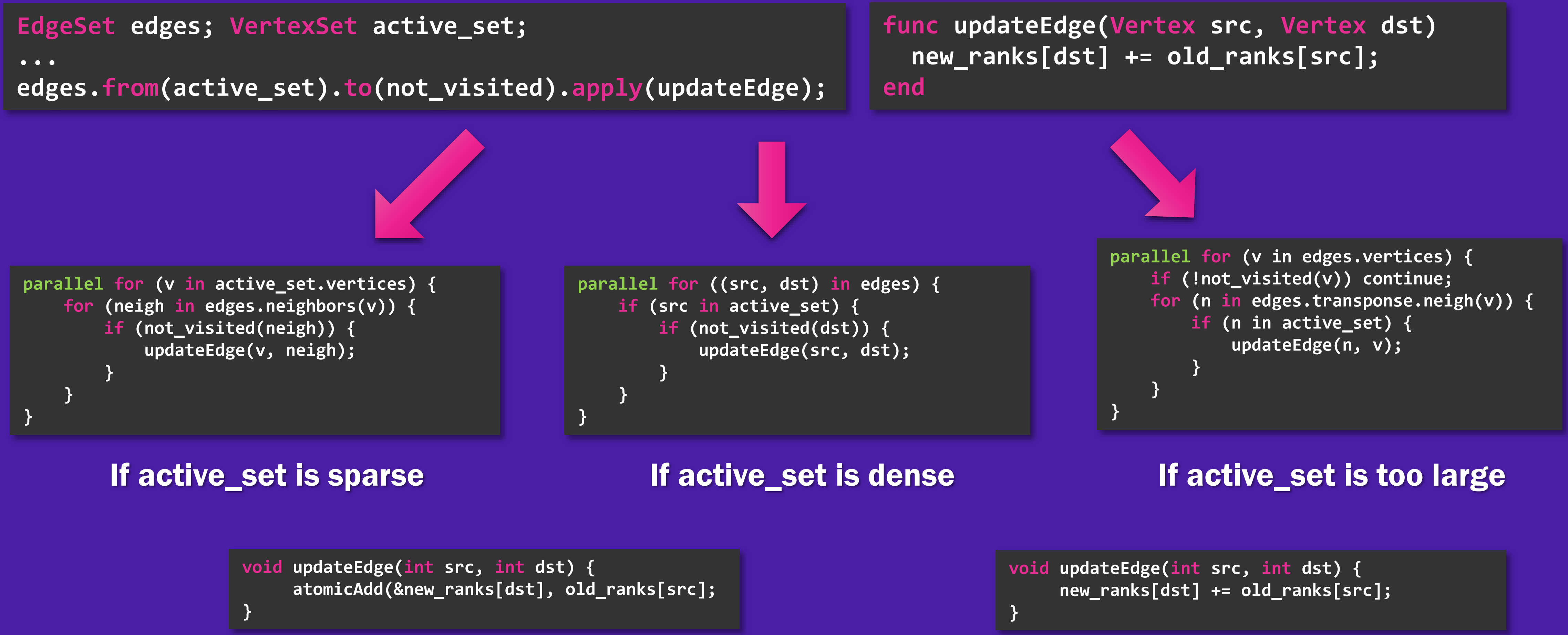
Domain Knowledge

Algorithm

Schedule

Generic DSL Compiler

What *really* is Domain Knowledge?



```
EdgeSet edges; VertexSet active_set;
...
edges.from(active_set).to(not_visited).apply(updateEdge);
```

```
func updateEdge(Vertex src, Vertex dst)
  new_ranks[dst] += old_ranks[src];
end
```

```
parallel for (v in active_set.vertices) {
    for (neigh in edges.neighbors(v)) {
        if (not_visited(neigh)) {
            updateEdge(v, neigh);
        }
    }
}
```

If active_set is sparse

```
parallel for ((src, dst) in edges) {
    if (src in active_set) {
        if (not_visited(dst)) {
            updateEdge(src, dst);
        }
    }
}
```

If active_set is dense

```
parallel for (v in edges.vertices) {
    if (!not_visited(v)) continue;
    for (n in edges.transpose.neigh(v)) {
        if (n in active_set) {
            updateEdge(n, v);
        }
    }
}
```

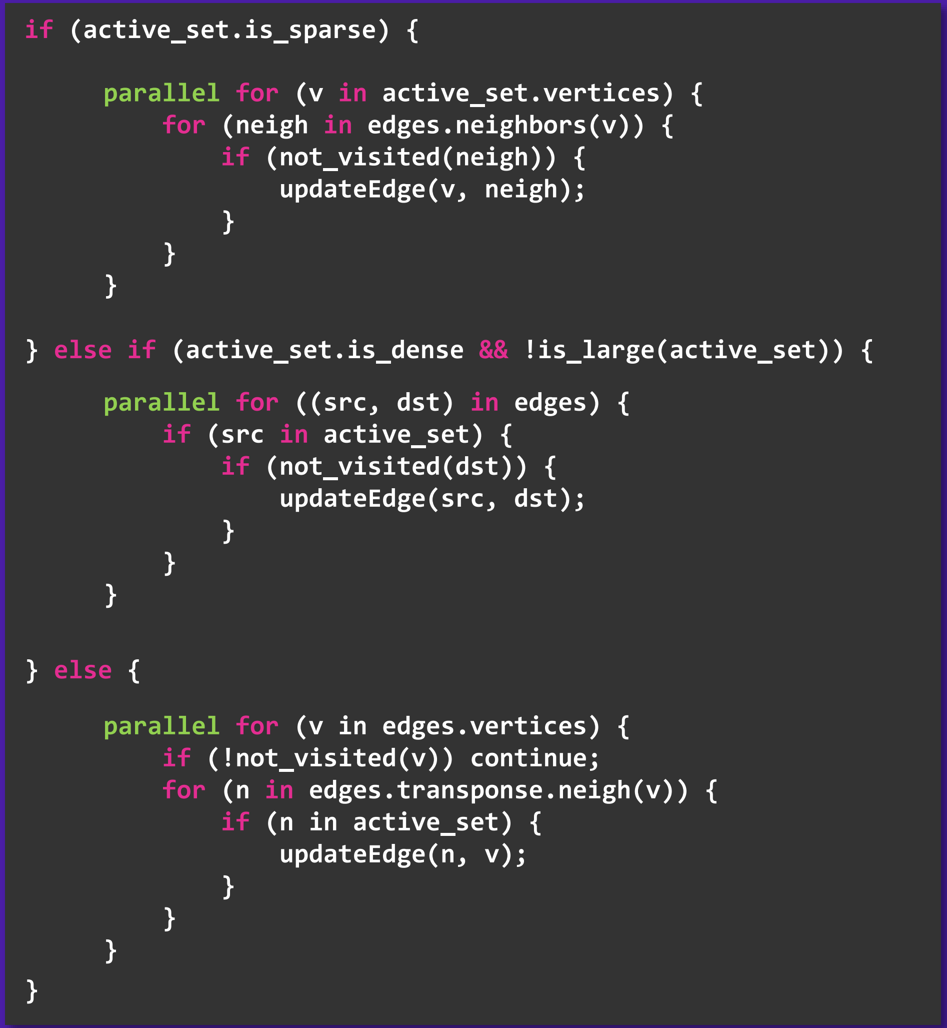
If active_set is too large

```
void updateEdge(int src, int dst) {
    atomicAdd(&new_ranks[dst], old_ranks[src];
}
```

```
void updateEdge(int src, int dst) {
    new_ranks[dst] += old_ranks[src];
}
```

What *really* is Domain Knowledge?

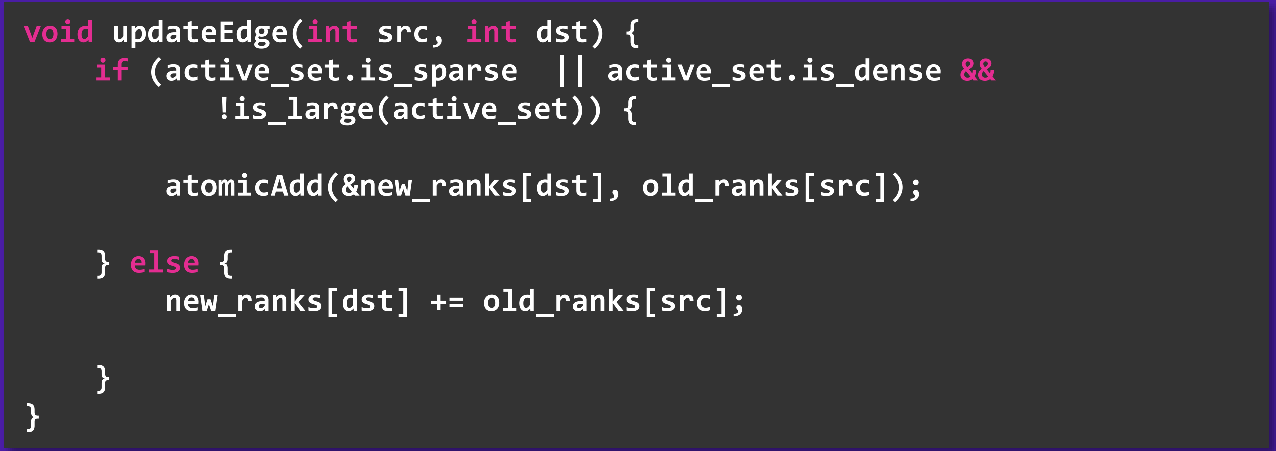
Domain Experts know how to optimized libraries



```
if (active_set.is_sparse) {

    parallel for (v in active_set.vertices) {
        for (neigh in edges.neighbors(v)) {
            if (not_visited(neigh)) {
                updateEdge(v, neigh);
            }
        }
    }

} else if (active_set.is_dense && !is_large(active_set)) {

    parallel for ((src, dst) in edges) {
        if (src in active_set) {
            if (not_visited(dst)) {
                updateEdge(src, dst);
            }
        }
    }

} else {

    parallel for (v in edges.vertices) {
        if (!not_visited(v)) continue;
        for (n in edges.transpose.neigh(v)) {
            if (n in active_set) {
                updateEdge(n, v);
            }
        }
    }
}
```

Runtime conditions with branches is slow

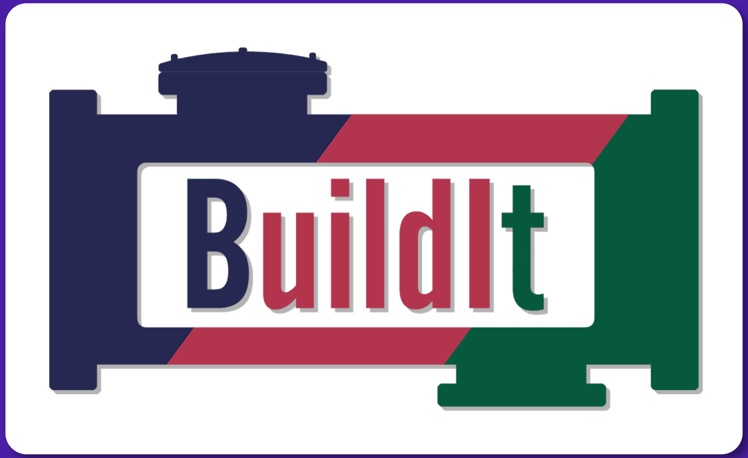
What *really* is Domain Knowledge?



```
void updateEdge(int src, int dst) {
    if (active_set.is_sparse  || active_set.is_dense &&
           !is_large(active_set)) {

        atomicAdd(&new_ranks[dst], old_ranks[src]);

    } else {
        new_ranks[dst] += old_ranks[src];

    }
}
```

Runtime conditions with branches is slow

Enter BuildIt!

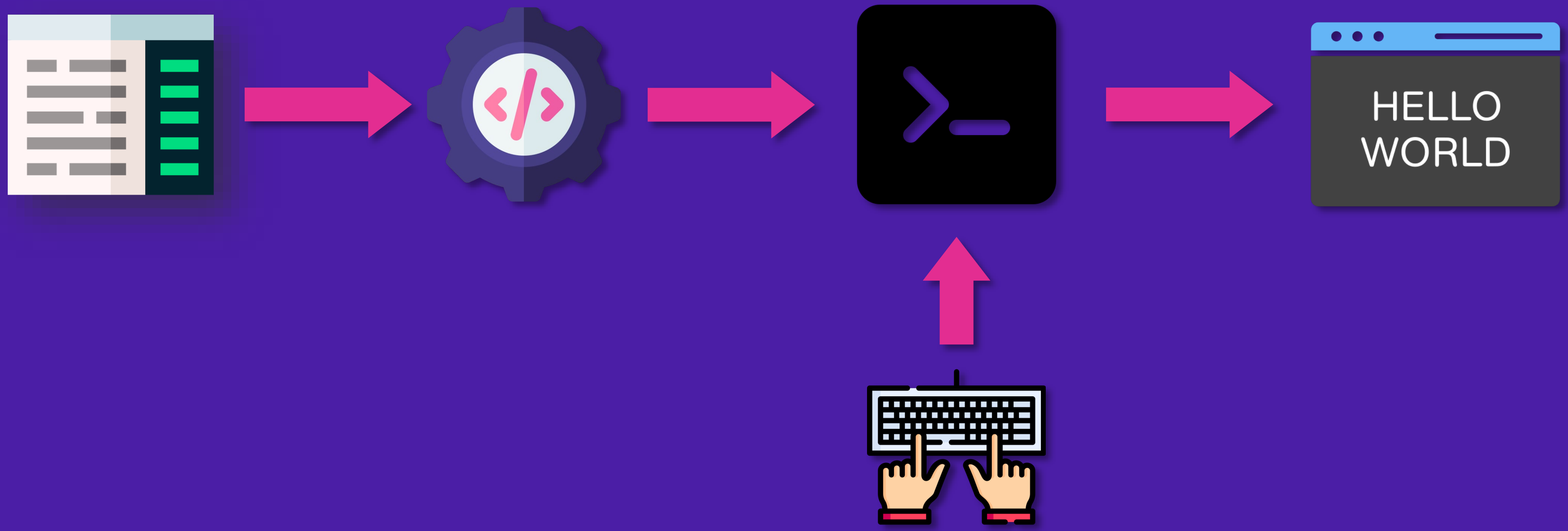


https://buildit.so

BuildIt automatically turns library code into compilers!

Enter BuildIt!

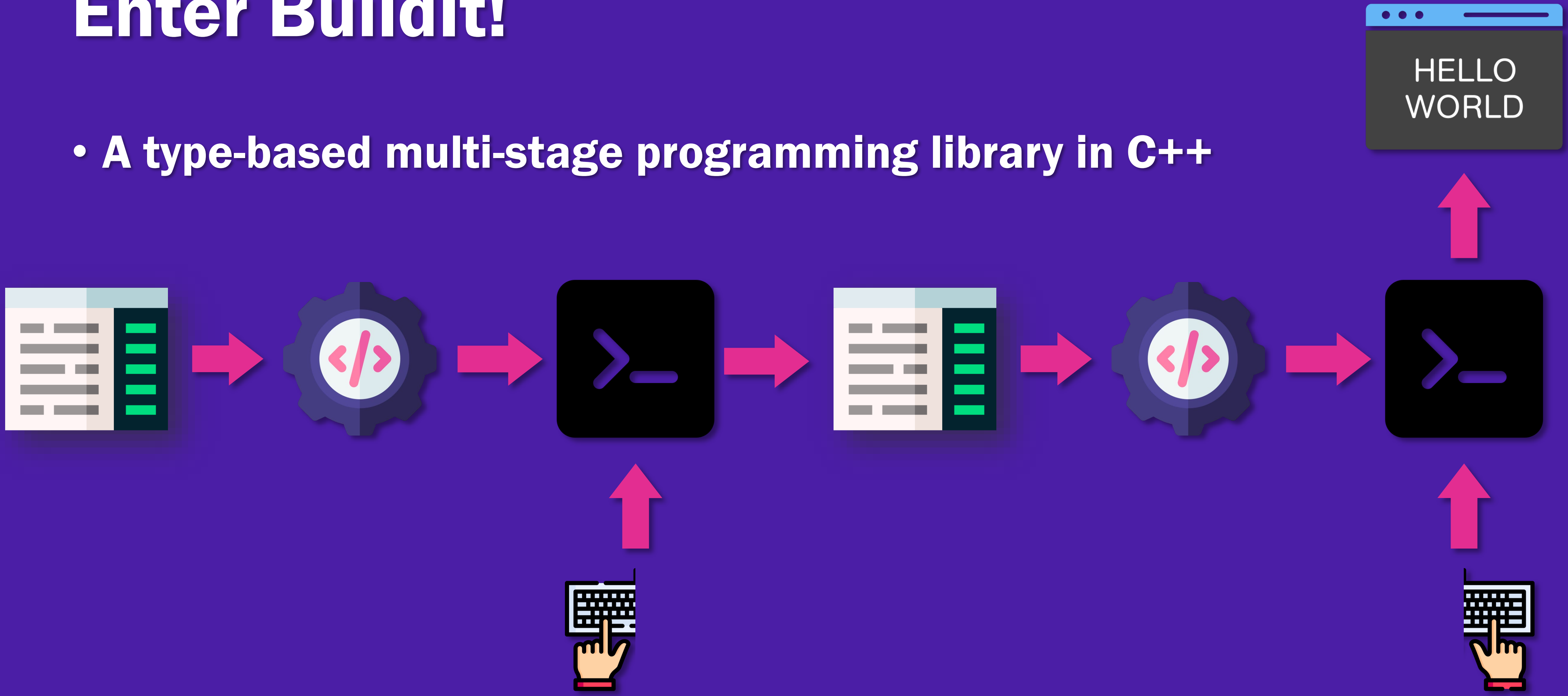
- A type-based multi-stage programming library in C++



1. BuildIt: A Type-Based Multistage Programming Framework for Code Generation in C++
Ajay Brahmakshatriya, Saman Amarasinghe

Enter BuildIt!

- A type-based multi-stage programming library in C++



1. BuildIt: A Type-Based Multistage Programming Framework for Code Generation in C++
Ajay Brahmakshatriya, Saman Amarasinghe

Enter BuildIt!

- Two new types – **dyn_var<T>** and **static_var<T>**

```
// The power function to stage
dyn_var<int> power(dyn_var<int> base, static_var<int> exponent) {
    dyn_var<int> res = 1, x = base;
    while (exponent > 1) {
        if (exponent % 2 == 1)
            res = res * x;
        x = x * x;
        exponent = exponent / 2;
    }
    return res * x;
}
...
context.extract_function(power, “power_15”, 15);
```

```
int power_15 (int arg0) {
  int var0 = arg0;
  int var1 = 1;
  int var2 = var0;
  var1 = var1 * var2;
  var2 = var2 * var2;
  var1 = var1 * var2;
  var2 = var2 * var2;
  var1 = var1 * var2;
  var2 = var2 * var2;
  return var1 * var2;
}
```

Enter BuildIt!

- Two new types – dyn_var<T> and static_var<T>

```
// The power function to stage
dyn_var<int> power(static_var<int> base, dyn_var<int> exponent) {
    dyn_var<int> res = 1, x = base;
    while (exponent > 1) {
        if (exponent % 2 == 1)
            res = res * x;
        x = x * x;
        exponent = exponent / 2;
    }
    return res * x;
}
...
context.extract_function(power, “power_15”, 15);
```

```
int power_15 (int arg1) {
  int var0 = arg1;
  int var1 = 1;
  int var2 = 15;
  while (var0 > 1) {
    if ((var0 % 2) == 1) {
      var1 = var1 * var2;
    }
    var2 = var2 * var2;
    var0 = var0 / 2;
  }
  return var1 * var2;
}
```

Full C++ language support in all stages

How does BuildIt work?

- BuildIt is embedded in C++ as a library, no compiler magic!

Overload all operators!!!

How does BuildIt work?

- BuildIt overloads all operators on dyn_var<T> to generate code instead of running it

How does BuildIt work?

- BuildIt overloads all operators on dyn_var<T> to generate code instead of running it

```
dyn_var<int> x, y = 0;
```

```
int var0;
int var1 = 0;
```

How does BuildIt work?

- BuildIt overloads all operators on dyn_var<T> to generate code instead of running it

```
dyn_var<int> x, y = 0;

x + y * 2;
```

```
int var0;
int var1 = 0;
var0 + var1 * 2;
```

How does BuildIt work?

- BuildIt overloads all operators on dyn_var<T> to generate code instead of running it

```
dyn_var<int> x, y = 0;

x + y * 2;
x = x + 1;
```

```
int var0;
int var1 = 0;
var0 + var1 * 2;
var0 = var0 + 1;
```

How does BuildIt work?

- BuildIt overloads all operators on dyn_var<T> to generate code instead of running it

```
dyn_var<int> x, y = 0;

x + y * 2;
x = x + 1;

foo_bar(z[0], &w, “hello”);
```

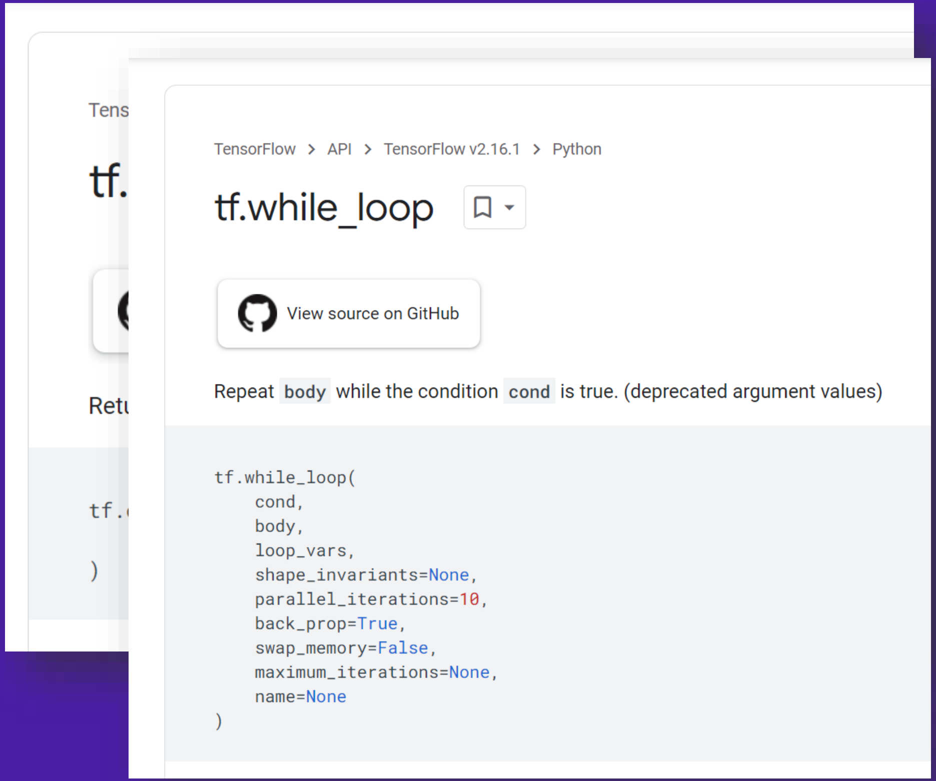
```
int var0;
int var1 = 0;
var0 + var1 * 2;
var0 = var0 + 1;

foo_bar(var3[0], &var2, “hello”);
```

```
if (x == 3) {
```

```
?????
```

How does BuildIt work?

- **Special operators for control-flow**



Tens

tf.

Retu

tf.(

)

TensorFlow > API > TensorFlow v2.16.1 > Python

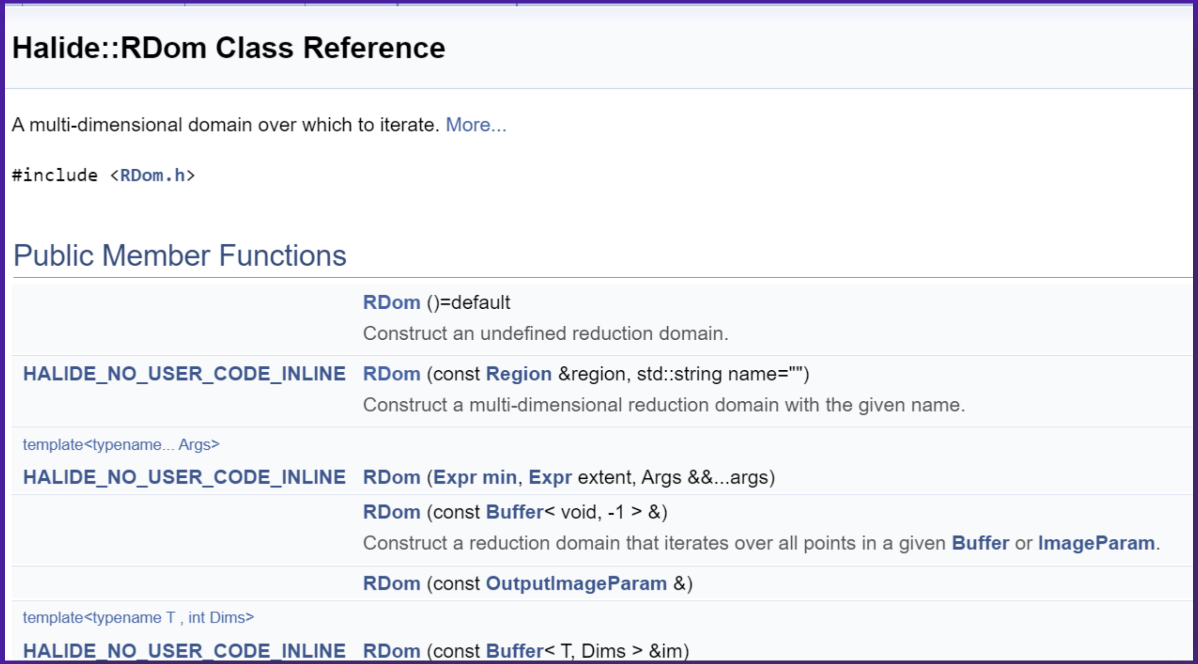
tf.while_loop

View source on GitHub

Repeat `body` while the condition `cond` is true. (deprecated argument values)

```
tf.while_loop(
    cond,
    body,
    loop_vars,
    shape_invariants=None,
    parallel_iterations=10,
    back_prop=True,
    swap_memory=False,
    maximum_iterations=None,
    name=None
)
```

Control flow in Tensorflow



Halide::RDom Class Reference

A multi-dimensional domain over which to iterate. More...

```
#include <RDom.h>
```

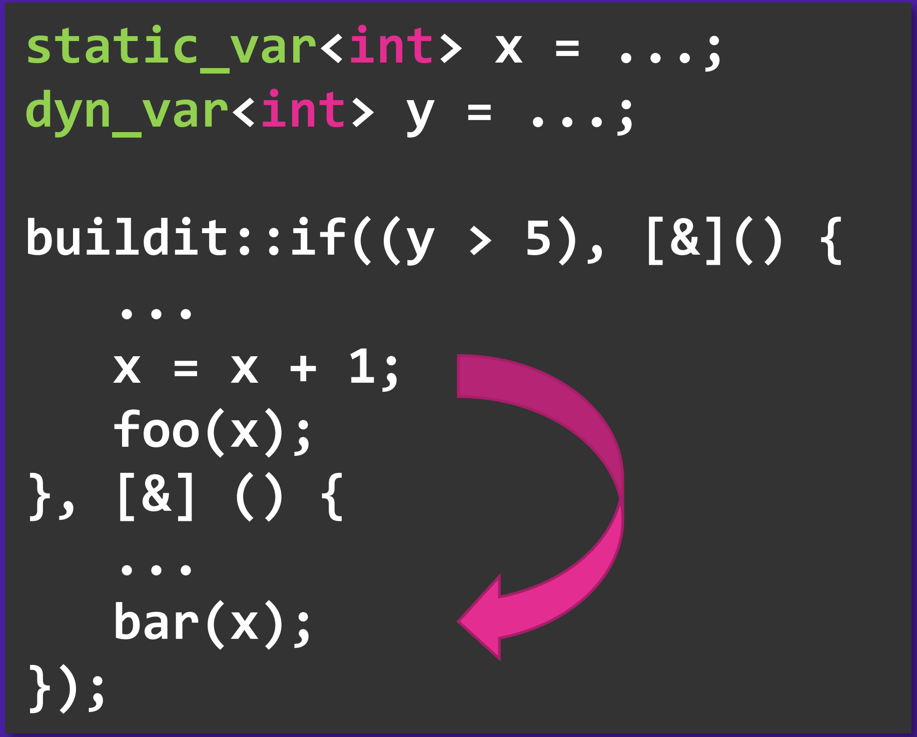
Public Member Functions

	RDom ()=default Construct an undefined reduction domain.
HALIDE_NO_USER_CODE_INLINE	RDom (const Region &region, std::string name="") Construct a multi-dimensional reduction domain with the given name.
template<typename... Args> HALIDE_NO_USER_CODE_INLINE	RDom (Expr min, Expr extent, Args &&...args)
	RDom (const Buffer< void, -1 > &) Construct a reduction domain that iterates over all points in a given Buffer or ImageParam.
	RDom (const OutputImageParam &)
template<typename T , int Dims> HALIDE_NO_USER_CODE_INLINE	RDom (const Buffer< T, Dims > &im)

Control flow in Halide

How does BuildIt work?

- **Special operators break first-stage semantics**



```
static_var<int> x = ...;
dyn_var<int> y = ...;

buildit::if((y > 5), [&]() {
    ...
    x = x + 1;
    foo(x);
}, [&] () {
    ...
    bar(x);
});
```

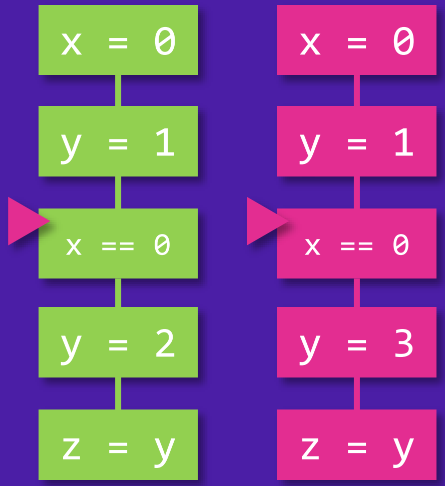
Side effects on static variables leak from disjoint paths

How does BuildIt work?

- Execute multiple times to explore all paths

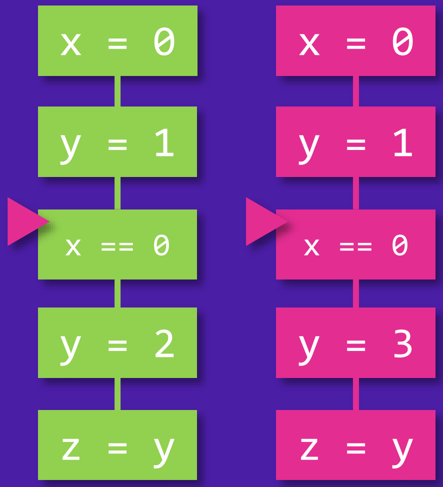
```
explicit dyn_var<T>::operator bool();
```

```
dyn_var<int> x = 0;
dyn_var<int> y = 1;
if (x == 0) {
   y = 2;
} else {
   y = 3;
}
dyn_var<int> z = y;
```



How does BuildIt work?

- Execute multiple times to explore all paths

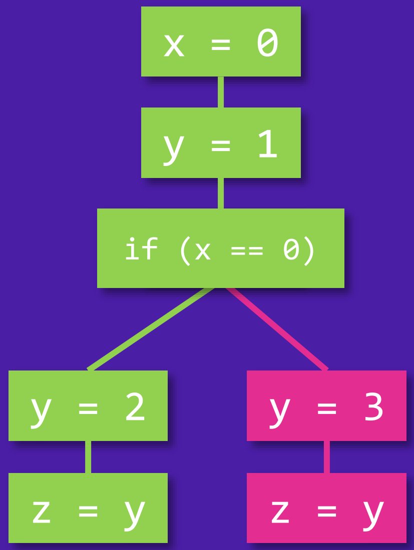


```
x = 0
y = 1
x == 0
y = 2
z = y
```

```
x = 0
y = 1
x == 0
y = 3
z = y
```

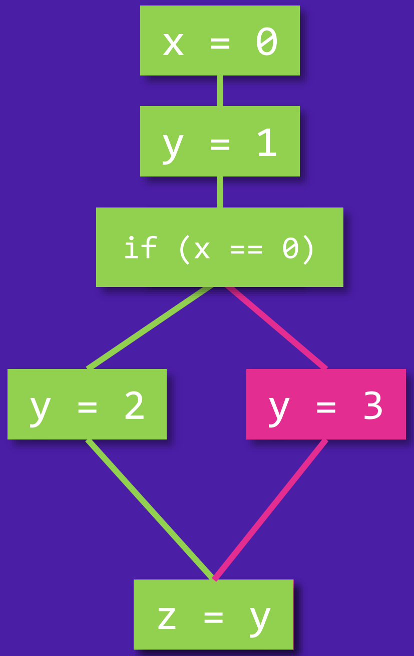
How does BuildIt work?

- Execute multiple times to explore all paths



How does BuildIt work?

- Execute multiple times to explore all paths



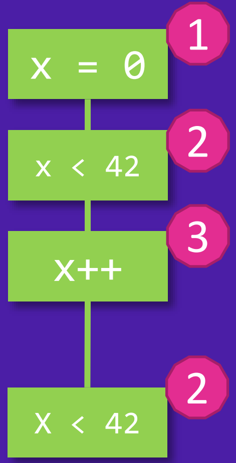
```
int var0 = 0;
int var1 = 1;
if (var0 == 0) {
   var1 = 2;
} else {
   var1 = 3;
}
int var2 = var1;
```

Memoization to improve complexity – details in the paper!

How does BuildIt work?

- What about loops? – use “Static Tags”

```
dyn_var<int> x = 0;

while (x < 42) {
    x++;
}

dyn_var<int> z = x;
```

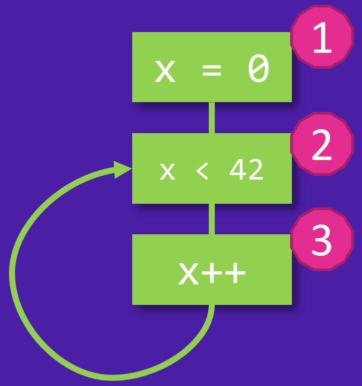


x = 0
1
x < 42
2
x++
3
X < 42
2

How does BuildIt work?

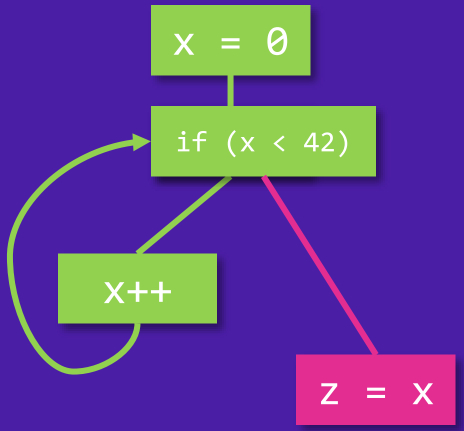
- What about loops? – use “Static Tags”

```
dyn_var<int> x = 0;

while (x < 42) {
   x++;
}

dyn_var<int> z = x;
```



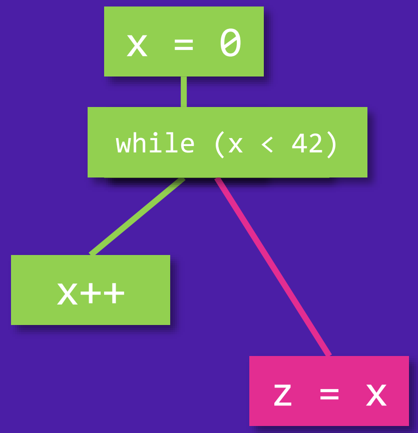
How does BuildIt work?

- What about loops? – use “Static Tags”



How does BuildIt work?

- What about loops? – use "Static Tags"



```
int var0 = 0;

while (var0 < 42) {
    var0++;
}

int var1 = var0;
```

BuildIt to DSLs

```
// active_set.is_sparse - static_var<bool>
// active_set.vertices - dyn_var<vector<int>>
...

if (active_set.is_sparse) {
    for (dyn_var<int> v in active_set.vertices) {
        for (dyn_var<int> neigh in edges.neighbors(v)) {
            if (not_visited(neigh)) {
                updateEdge(v, neigh);
            }
        }
    }
} else if (active_set.is_dense && !is_large(active_set)) {
    for (dyn_var<int> (src, dst) in edges) {
        if (dyn_var<int> src in active_set) {
            if (not_visited(dst)) {
                updateEdge(src, dst);
            }
        }
    }
} else {
    for (dyn_var<int> v in edges.vertices) {
        if (!not_visited(v)) continue;
        for (dyn_var<int> n in edges.transpose.neigh(v)) {
            if (n in active_set) {
                updateEdge(n, v);
            }
        }
    }
}
```

```
for (v in edges.vertices) {
    if (!not_visited(v)) continue;
    for (n in edges.transpose.neigh(v)) {
        if (n in active_set) {
            updateEdge(n, v);
        }
    }
}
```

```
void updateEdge(int src, int dst) {
     new_ranks[dst] += old_ranks[src];
}
```

DSL compilers are more than code generation

- DSL compilers often need to analyze the code before specialized code generation

How do we do analyses without understanding how compilers work?

DSL compilers are more than code generation

- DSL compilers often need to analyze the code before specialized code generation

```
void updateEdge(Vertex src, Vertex dst) {
   new_rank[dst] += old_rank[src];
}
```

- Whether atomics are required depends on if the index at the write access is shared across multiple threads
- Data-flow analysis with a 3 point lattice – SHARED, INDEPENDENT, CONSTANT

DSL compilers are more than code generation

```
struct Vertex {
    dyn_var<int> vid;
    enum access_t {SHARED, INDEPENDENT, CONSTANT};
    static_var<access_t> access;
};
```

```
void Vertex::operator= (const Vertex &rhs) {
    vid = rhs.vid;
    access = rhs.access;
}
Vertex Vertex::operator+ (const Vertex &rhs) {
    Vertex ret; ret.vid = vid + rhs.vid;
    if (rhs.access == CONSTANT) ret.access = access;
    else ret.access = SHARED;
}
```

DSL compilers are more than code generation

```
struct ArrayAccess{
   Vertex index;
   dyn_var<T[]> &array;
}; // For expressions like array[Vertex]

void ArrayAccess::operator+= (const ArrayAccess& rhs) {
    if (index.access == INDEPENDENT)
        array[index] += rhs.array[rhs.index];
    else
        atomicAdd(&array[index], rhs.array[rhs.index]);
}
```

Generates efficient code based on context of invocation

1. GraphIt to CUDA compiler in 2021 LOC: A case for high-performance DSL implementation via staging with BuilDSL
Ajay Brahmakshatriya, Saman Amarasinghe

DSL compilers are more than code generation

```
parallel for (v in edges.vertices) {
    if (!not_visited(v)) continue;
    for (n in edges.transpose.neigh(v)) {
        if (n in active_set) {
            Vertex src(n); Vertex dst(v);
            src.access = SHARED;
            dst.access = INDEPENDENT;
            updateEdge(src, dst);
        }
    }
}
```

```
void updateEdge(Vertex src, Vertex dst) {
     new_ranks[dst] += old_ranks[src];
}
```

```
parallel for (...) {
   for (...) {
       new_ranks[dst] += old_ranks[src];
   }
}
```

DSL compilers are more than code generation

- **Infrastructure support for parallel CPU and GPU code generation**

```
builder::annotate(“pragma: omp parallel for”);
for (dyn_var<int> i = 0; i < N; i++) {
    ...
}
```

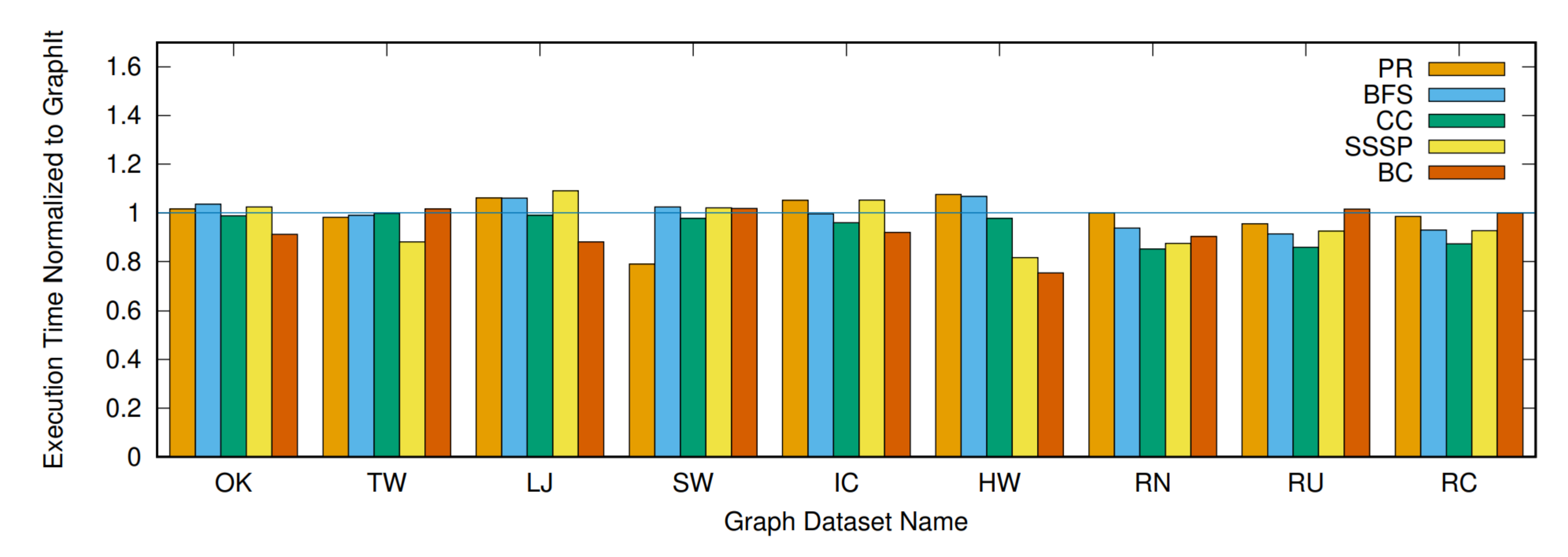
```
#pragma omp parallel for
for (int var0 = 0; var0 < 512; var0++) {
    ...
}
```

```
builder::annotate(“CUDA_KERNEL”);
for (dyn_var<int> i = 0; i < N; i++) {
    for (dyn_var<int> j = 0; j < M; j++) {
        a[i * M + j] = 3.14;
    }
}
```

```
void __global__ cuda_kernel0(float* arg0) {
    arg0[blockIdx.x * 512 + threadIdx.x] = 3.14;
}
...
cuda_kernel0<<<256, 512>>>(a);
cudaDeviceSynchonize();
```

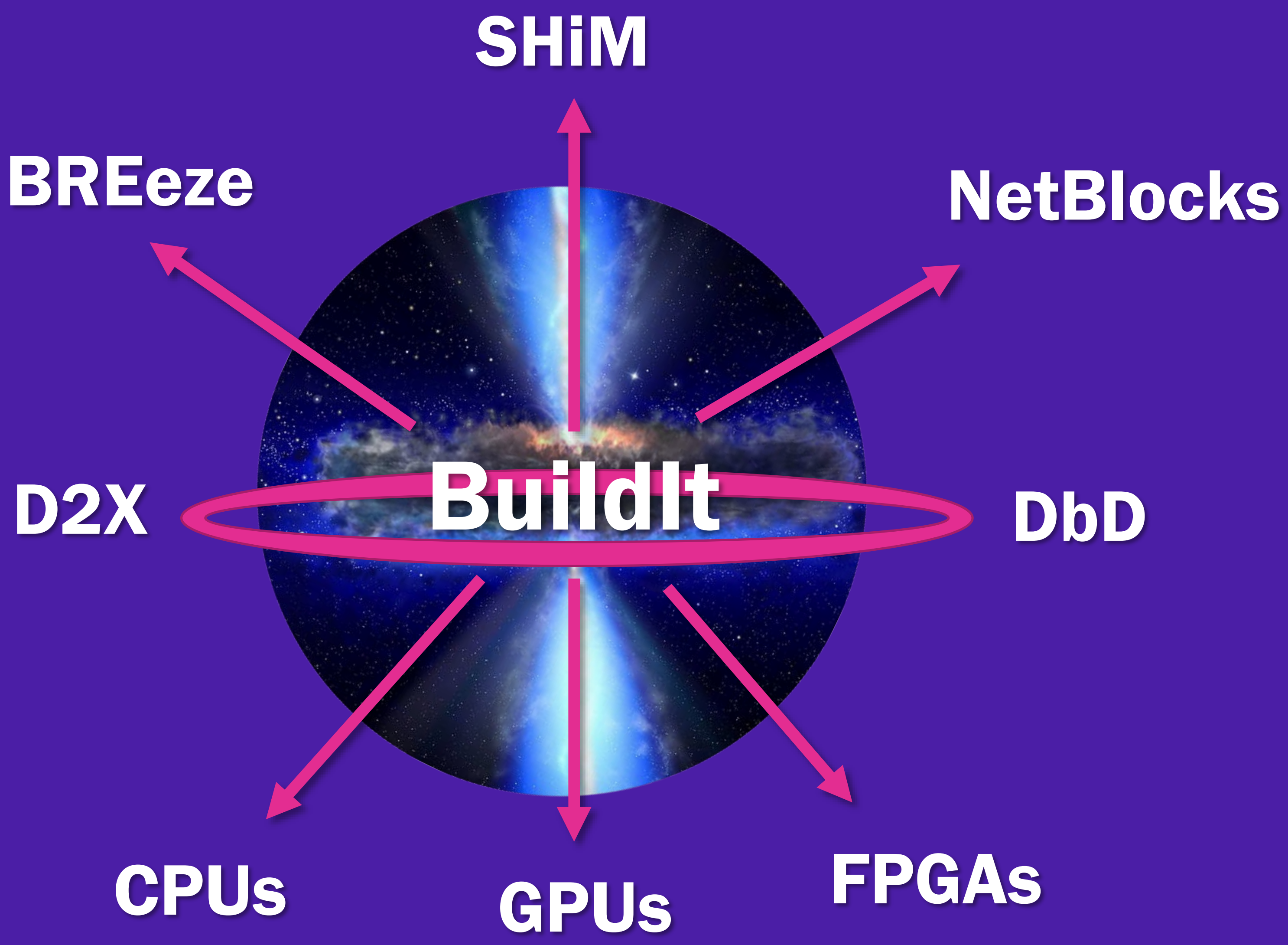
GraphIt to CUDA in 2021 LoC

- Reimplemented the entire GraphIt GPU compiler including all schedules in just 2021 lines of C++ code



1. GraphIt to CUDA compiler in 2021 LOC: A case for high-performance DSL implementation via staging with BuilDSL
Ajay Brahmakshatriya, Saman Amarasinghe

The Quasar of BuildIt



SHiM

BREeze

NetBlocks

D2X

BuildIt

DbD

CPUs

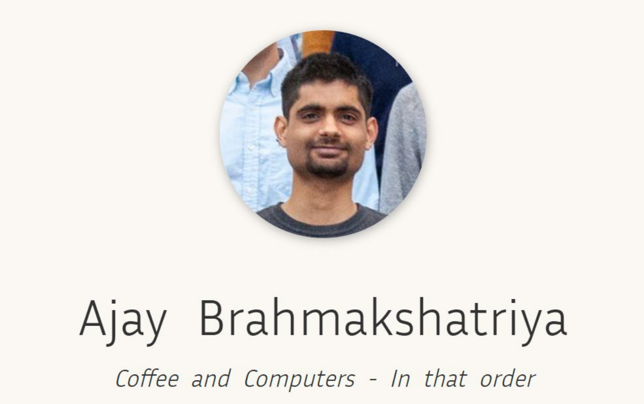
GPUs

FPGAs

https://buildit.so

Contributions welcome!

Democratizing High-Performance DSL Development with BuildIt



Ajay Brahmakshatriya

Coffee and Computers - In that order

https://intimeand.space

https://buildit.so

- If you are interested in building DSLs for your architectures or domains, reach out at ajaybr@mit.edu

Images and icons from https://www.flaticon.com/