# The Intel® Array Building Blocks Virtual Machine

Stefanus Du Toit
<stefanus.du.toit@intel.com>

Software and Services Group

intel Software

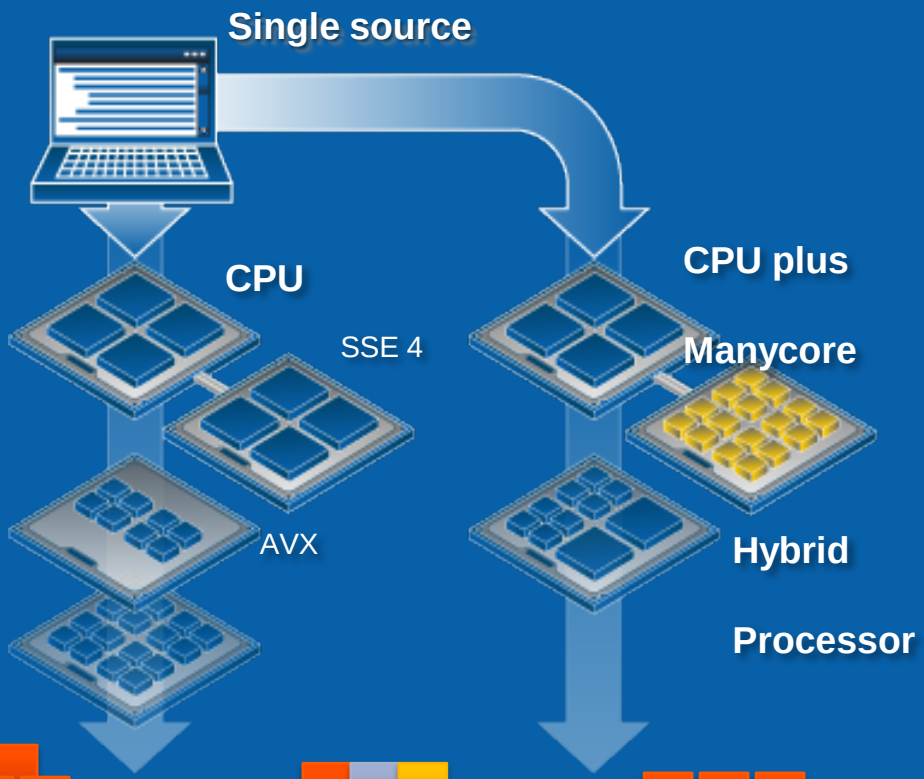
# Agenda

- Intel® Array Building Blocks
- VM Description and Rationale
- Programming Model
- Basic API Functionality and Examples
- Operations in the VM
- Runtime API
- Usage Models
- Finding Out More

# Intel® Array Building Blocks

Key Points:

- Productivity
- Performance
- Portability





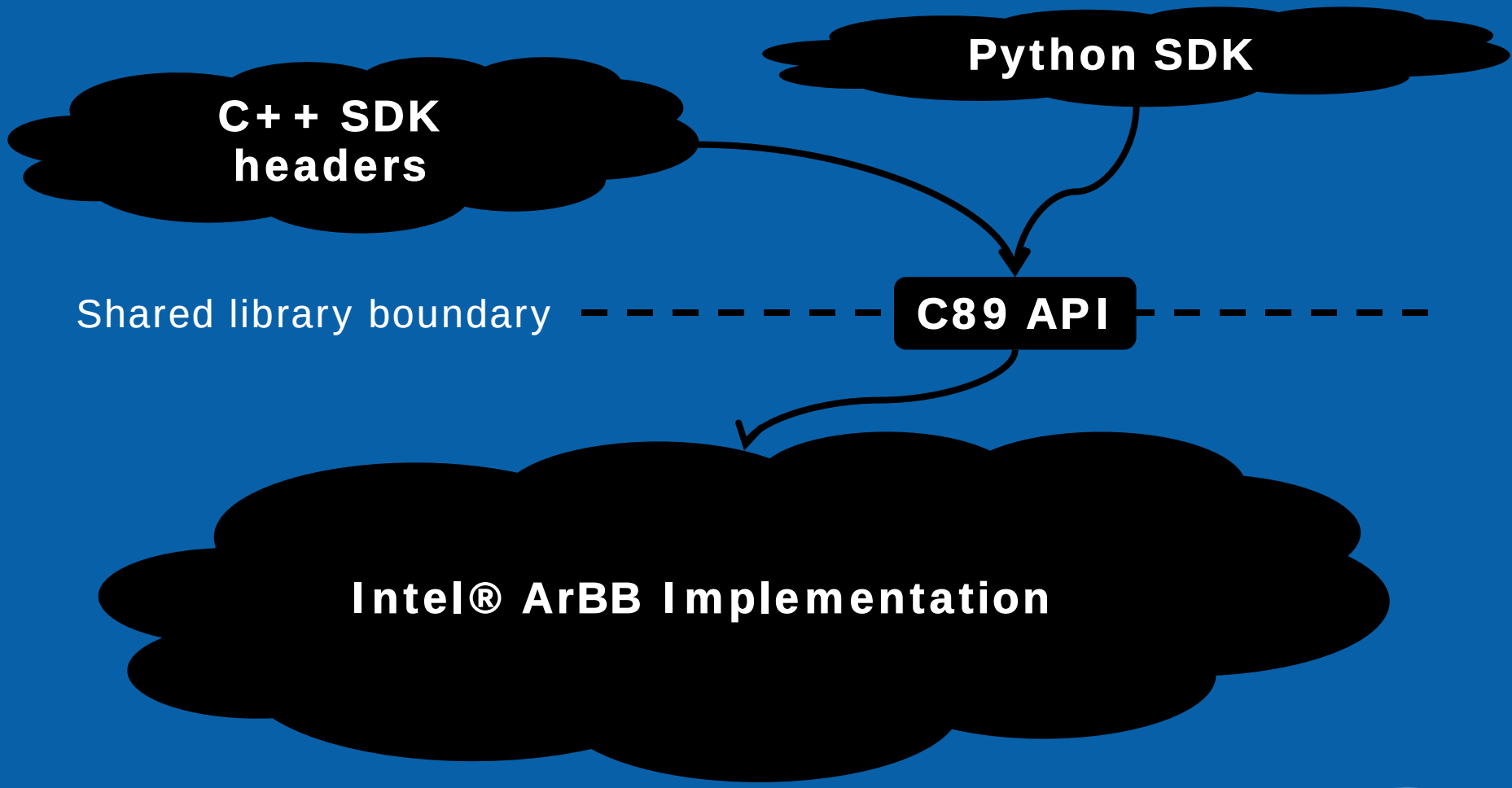




Software and Services Group

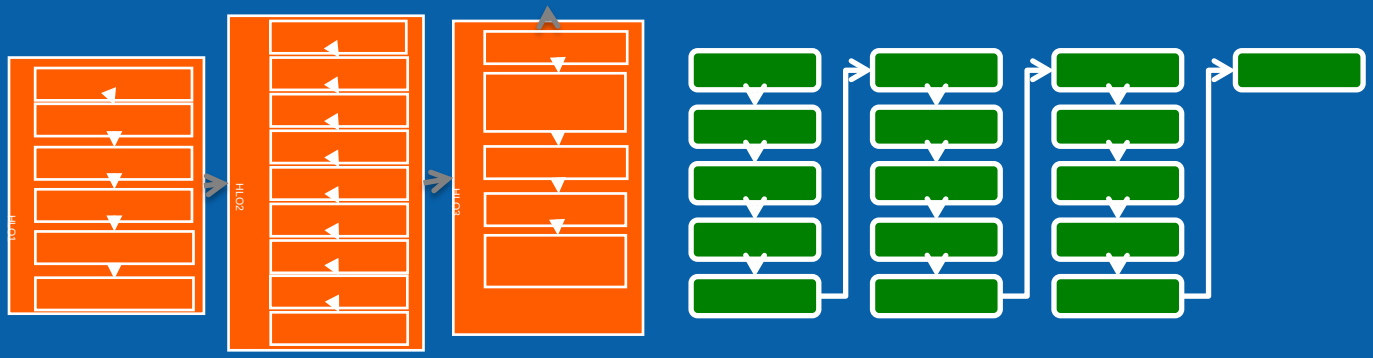
intel Software

# VM Rationale - Implementation

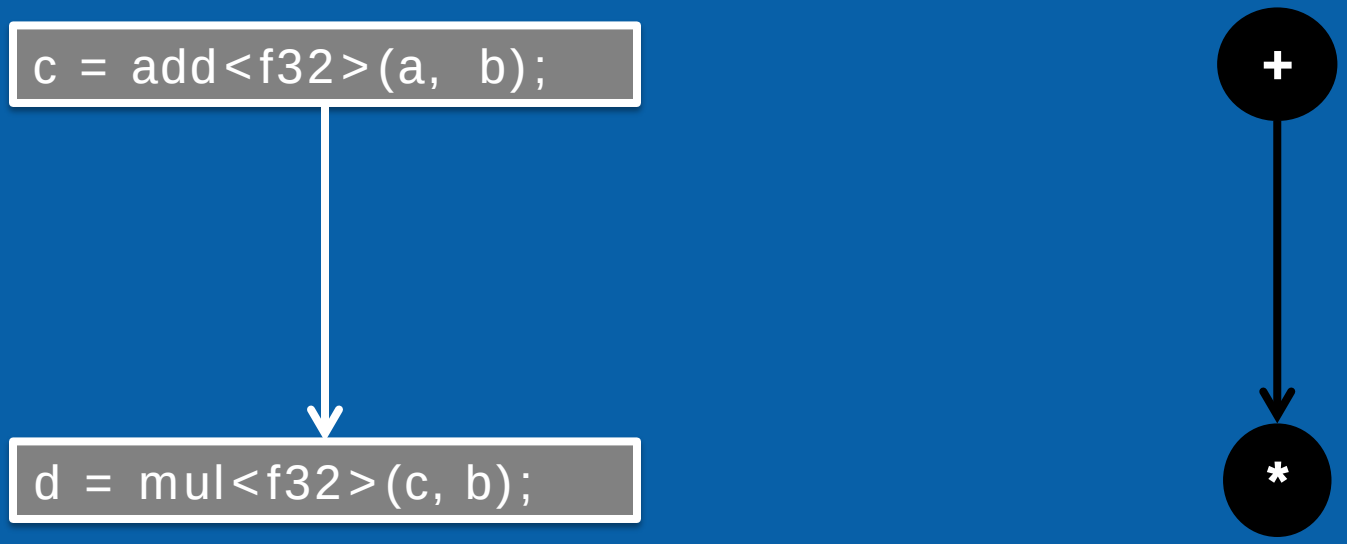


# VM Rationale - Usage

- Deterministic semantics
- Powerful vectorization
- Powerful fusion optimizations
- Distributed memory support
- Dynamic code generation
  - Includes SSE2, SSE3, SSSE3, AVX, Intel MIC architecture

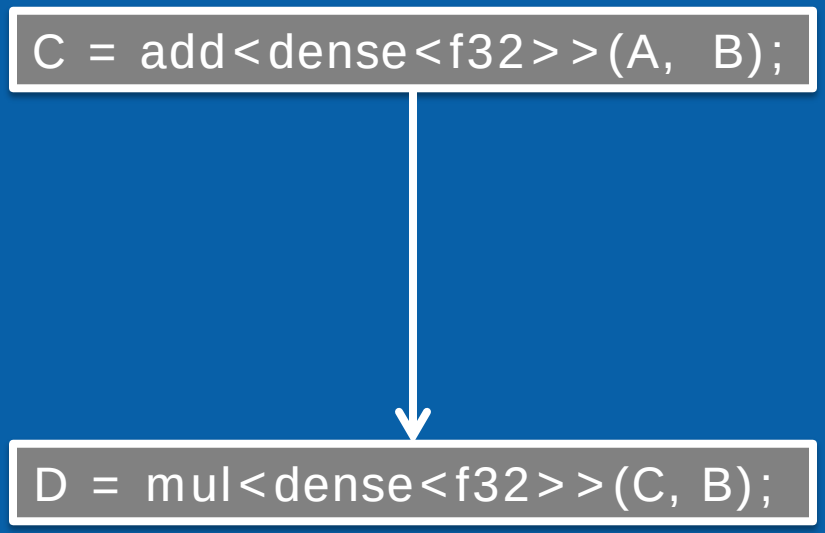


# Programming Model – Serial Code

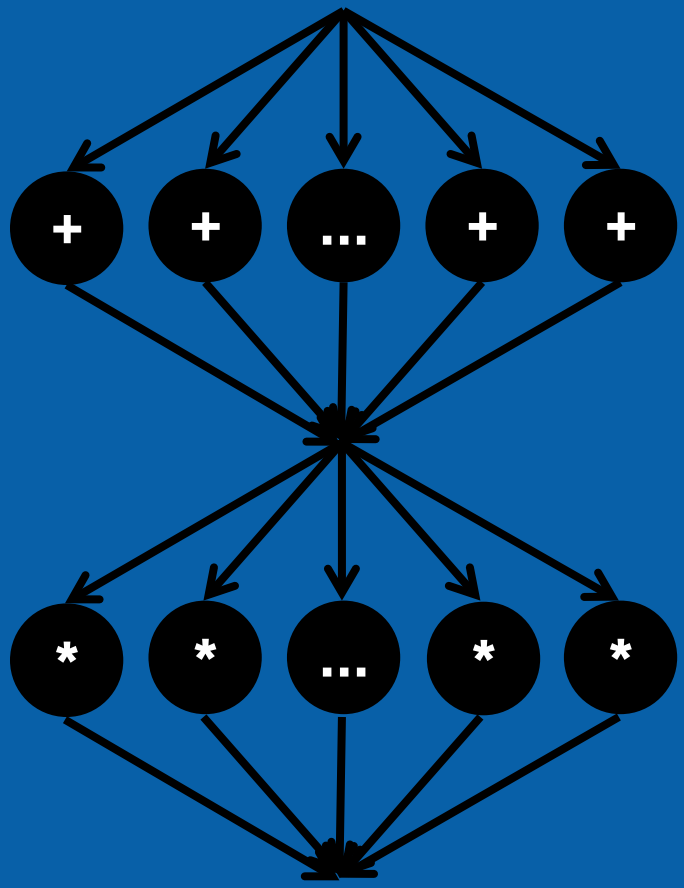


# Programming Model - Parallelism

```
C = add<dense<f32>>(A,  B);
```



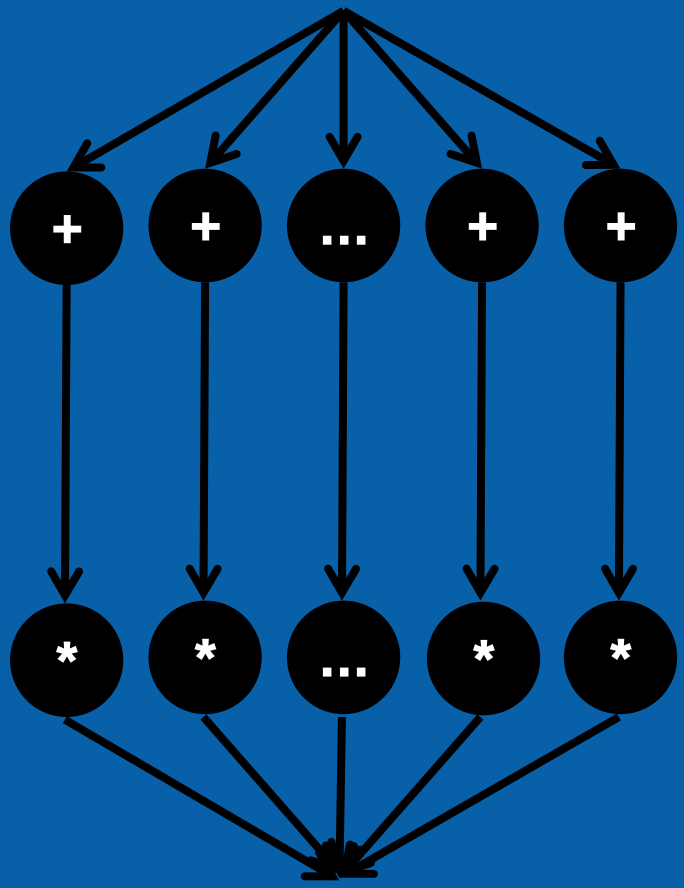
```
D = mul<dense<f32>>(C, B);
```



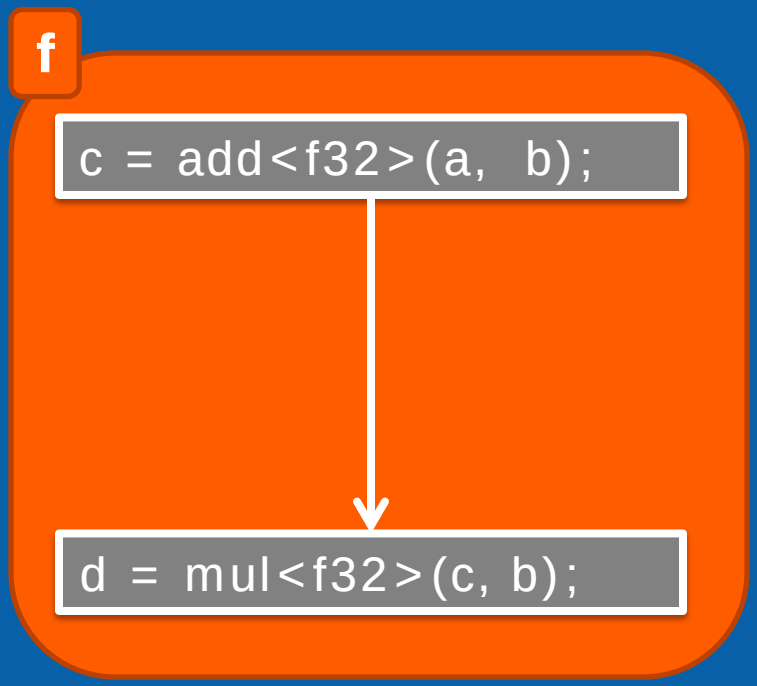
# Programming Model – Optimization

```
C = add<dense<f32>>(A, B);
```

```
D = mul<dense<f32>>(C, B);
```

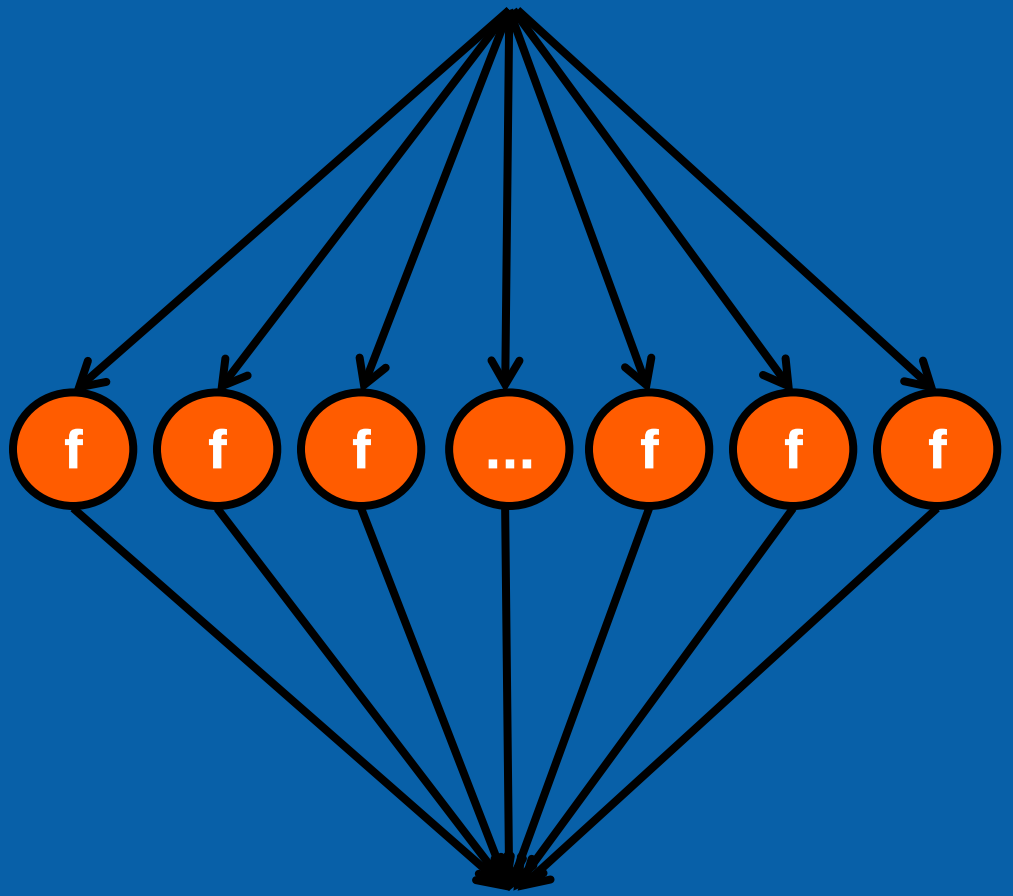


# Programming Model – Functions




# Programming Model – Maps

```
X = map(f, Y, Z)
```



# Basic API Functionality

- Create types
  - Scalar types
  - Dense and nested container types
  - Function types
- Create variables
  - Global to all functions
  - Local to a function
- Create functions and operations

# API Conventions

- All types are enumerations or opaque types
- All functions return an error code & description
- No assumptions about heap
- Only basic C functionality used

```
/// Creates a new local variable within the given function.
///
/// @return An error code depending on the result of the operation:
///  - ::arbb_error_none if the operation succeeded.
///  - ::arbb_error_invalid_argument if @p function is a null object.
///  - ::arbb_error_invalid_argument if @p out_var is a null pointer.
///  - ::arbb_error_invalid_argument if @p type is a null object.
///  - ::arbb_error_invalid_argument if @p type is a function type.
///  - ::arbb_error_scoping if @p function is not being defined.
ARBB_VM_EXPORT
arbb_error_t arbb_create_local(arbb_function_t function,
                               arbb_variable_t* out_var,
                               arbb_type_t type,
                               const char* name,
                               arbb_error_details_t* details);
```

# C API Example: Dot-product

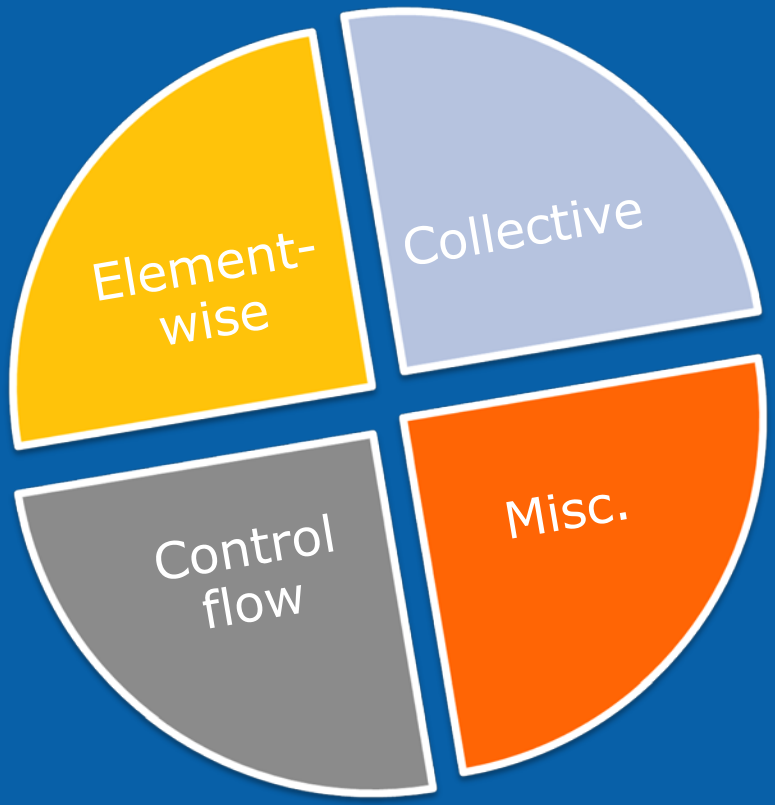
```
arbb_type_t dense_1d_f32;
arbb_get_dense_type(context, &dense_1d_f32, arbb_f32, 1, NULL);

arbb_type_t fn_type;
arbb_get_function_type(context, &fn_type, 1, {arbb_f32},
                       2, {dense_1d_f32, dense_1d_f32}, NULL);

arbb_function_t function;
arbb_begin_function(context, &function, fn_type, “dot”, NULL);
  arbb_variable_t a, b, c, t;
  arbb_get_parameter(function, &a, 0 /* input */,  0, NULL);
  arbb_get_parameter(function, &b, 0 /* input */,  1, NULL);
  arbb_get_parameter(function, &c, 1 /* output */, 0, NULL);

  arbb_create_local(function, &t, dense_1d_f32, “t”, NULL);

  arbb_op(function, arbb_op_mul, {t}, {a, b}, NULL);

  arbb_op(function, arbb_op_reduce_add, {c}, {t}, NULL);
arbb_end_function(function, NULL);
```

# Textual IR Example: Dot-product

```
function _dot(out $f32 _c, in $dense<$f32> _a, in $dense<$f32> _b)
{
  local $dense<$f32> _t;
  _t = mul<$dense<$f32>>(_a, _b);
  _c = reduce_add<$f32>(_t);
}
```

*Syntax not final.*

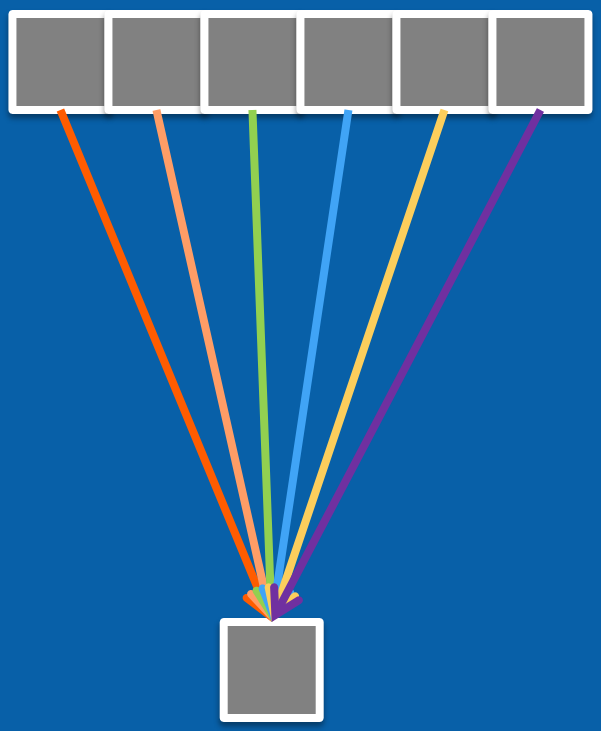
# Operations

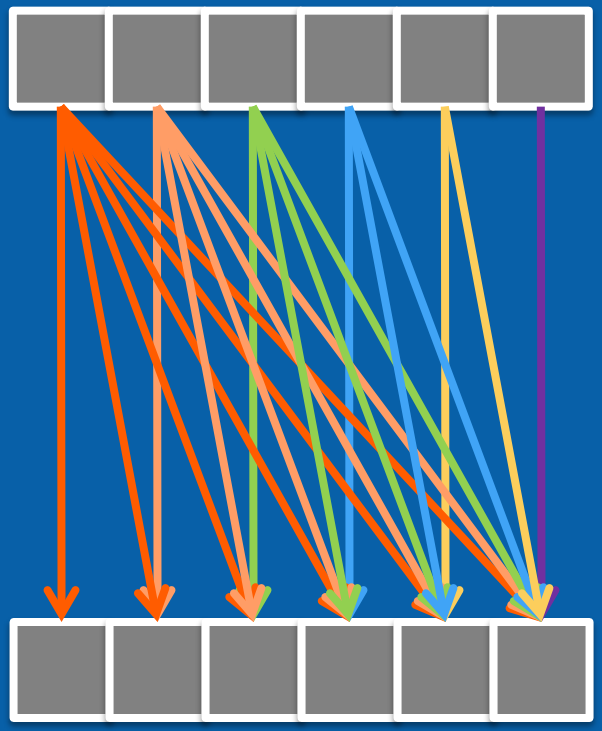


# Element-wise operations

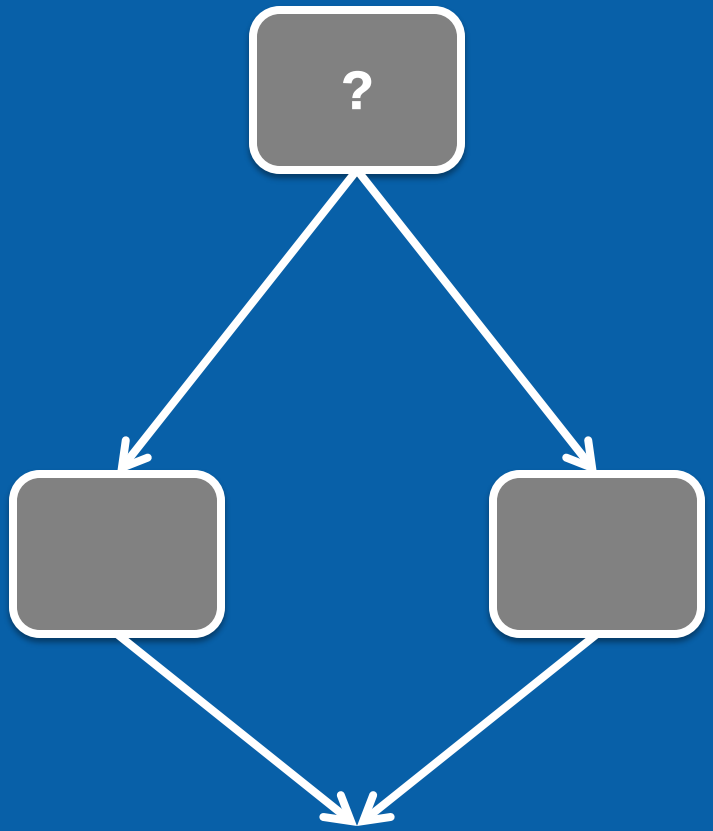
- Any scalar operations are allowed
- Source container shapes must match
  - Destination container size is irrelevant
- Any (but not all) sources can be scalars

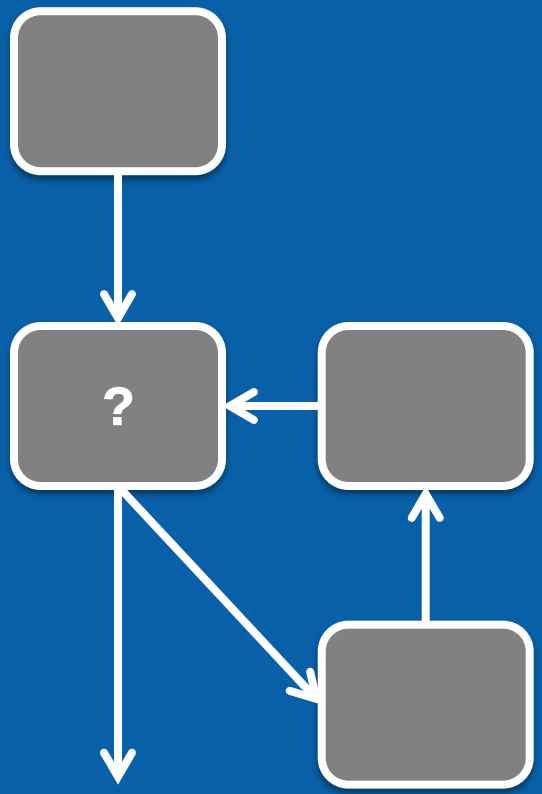
# Collective operations





# Control flow operations



# Miscellaneous operations

- Obtain attributes of values
  - E.g. container dimensions
- Reorder containers
  - E.g. replace a row, scatter, gather, …
- System operations
  - E.g. obtain current time

```
local dense<$f32, 1> _data;
local dense<$f32, 1> _result;
local dense<$isize, 1> _indices;
_result = gather<dense<$f32, 1>>(_data, _indices, $f32(0.0));
```

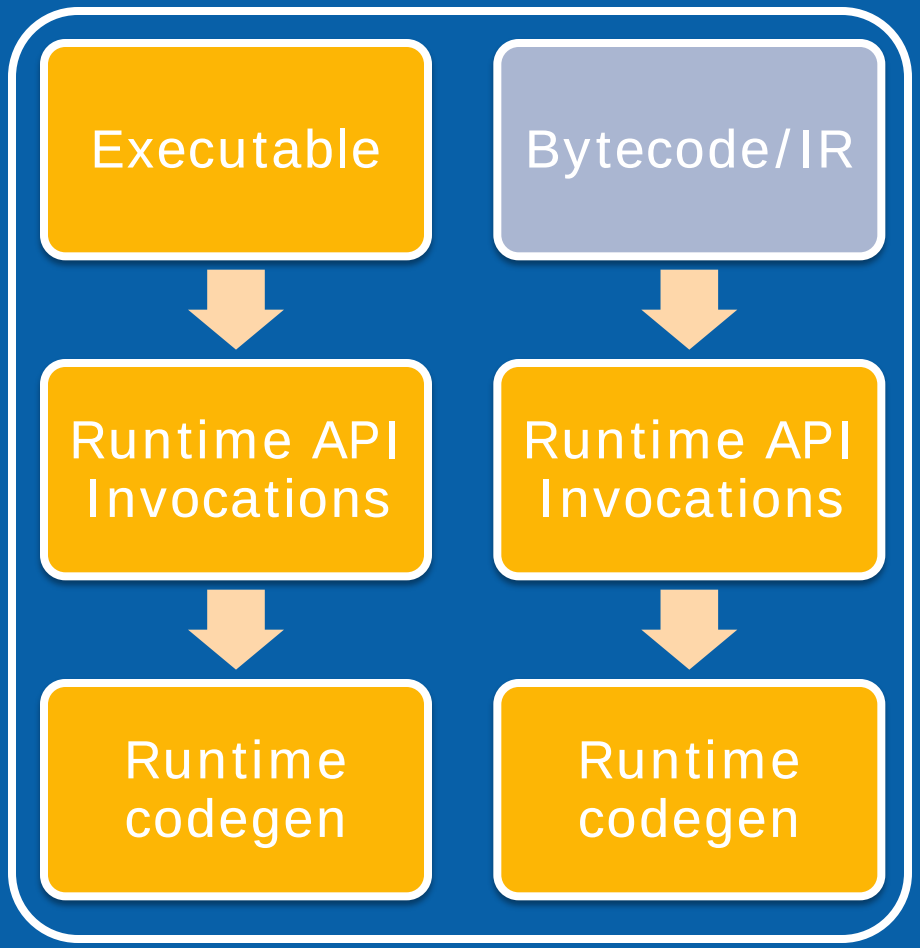
# Runtime API

- Read/write global scalars
- Bind containers to host data
- Map container data into host address space
- Compile functions
- Execute functions
- Manage asynchrony

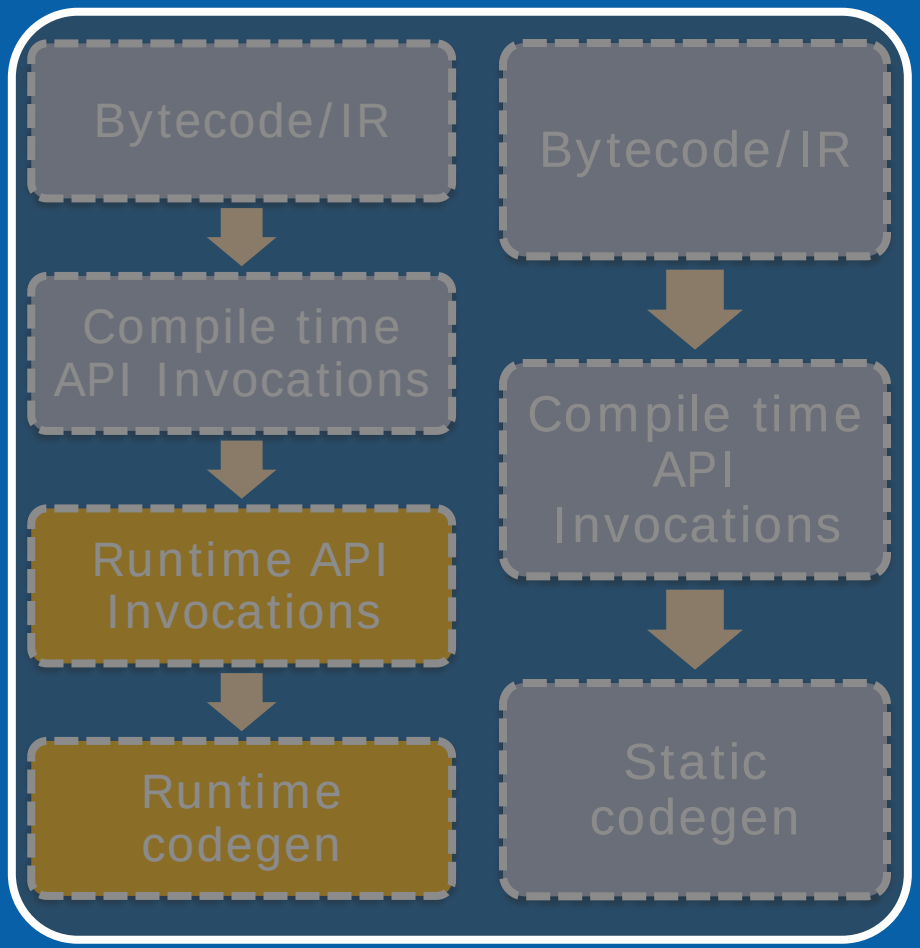
```
/// Maps the global container provided in @p container into the host
/// address space.
///
/// The mapping is valid until the next virtual machine operation
/// which directly or indirectly accesses the given variable.
///
/// @return An error code depending on the result of the operation:
///  - ::arbb_error_none if the operation succeeded.
///  - ::arbb_error_invalid_argument if @p context is a null object.
///  - ::arbb_error_invalid_argument if @p container is a null object.
///  - ::arbb_error_invalid_argument if @p out_data is a null pointer.
///  - ::arbb_error_invalid_argument if @p out_byte_pitch is a null pointer.
///  - ::arbb_error_invalid_argument if @p container is not a global variable.
///  - ::arbb_error_invalid_argument if @p container is not a dense container.
ARBB_VM_EXPORT
arbb_error_t arbb_map_to_host(arbb_context_t context,
                              arbb_variable_t container,
                              void** out_data,
                              uint64_t* out_byte_pitch,
                              arbb_range_access_mode_t mode,
                              arbb_error_details_t* details);
```

# VM Usage Models:

## Present




## Future




static dynamic

# Finding Out More

- VM Specification to be published soon
- Fully implemented, see “arbb_vmapi.h”
- Download Beta 1, give it a try!

http://software.intel.com/en-us/articles/intel-array-building-blocks/