The Intel® Array Building Blocks Virtual Machine

Legal Information

- INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPETY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL® PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.
- Intel may make changes to specifications and product descriptions at any time, without notice.
- All products, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice.
- Intel, processors, chipsets, and desktop boards may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.
- Any code names and other code names featured are used internally within Intel to identify products that are in development and not yet publicly announced for release. Customers, licensees and other third parties are not authorized by Intel to use code names in advertising, promotion or marketing of any product or services and any such use of Intel's internal code names is at the sole risk of the user
- Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.
- Intel, Intel® Streaming SIMD Extensions (Intel® SSE), Intel® Advanced Vector Extensions (Intel® AVX), Intel® Parallel Building Blocks (Intel® PBB), Intel® Threading Building Blocks (Intel® TBB), Intel® Array Building Blocks (Intel® ArBB), Intel® Math Kernel Library (Intel® MKL), Intel® Integrated Performance Primitives (Intel® IPP), Intel® Cilk Plus and the Intel logo are trademarks of Intel Corporation in the United States and other countries.
- *Other names and brands may be claimed as the property of others.
-

Optimization Notice

Intel® compilers, associated libraries and associated development tools may include or utilize options that optimize for instruction sets that are available in both Intel® and non-Intel microprocessors (for example SIMD instruction sets), but do not optimize equally for non-Intel microprocessors. In addition, certain compiler options for Intel compilers, including some that are not specific to Intel micro-architecture, are reserved for Intel microprocessors. For a detailed description of Intel compiler options, including the instruction sets and specific microprocessors they implicate, please refer to the "Intel® Compiler User and Reference Guides" under "Compiler Options". Many library routines that are part of Intel® compiler products are more highly optimized for Intel microprocessors than for other microprocessors. While the compilers and libraries in Intel® compiler products offer optimizations for both Intel and Intel-compatible microprocessors, depending on the options you select, your code and other factors, you likely will get extra performance on Intel microprocessors.

Intel® compilers, associated libraries and associated development tools may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include Intel® Streaming SIMD Extensions 2 (Intel® SSE2), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), and Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3) instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors.

While Intel believes our compilers and libraries are excellent choices to assist in obtaining the best performance on Intel® and non-Intel microprocessors, Intel recommends that you evaluate other compilers and libraries to determine which best meet your requirements. We hope to win your business by striving to offer the best performance of any compiler or library; please let us know if you find we do not.

Notice revision #20101101

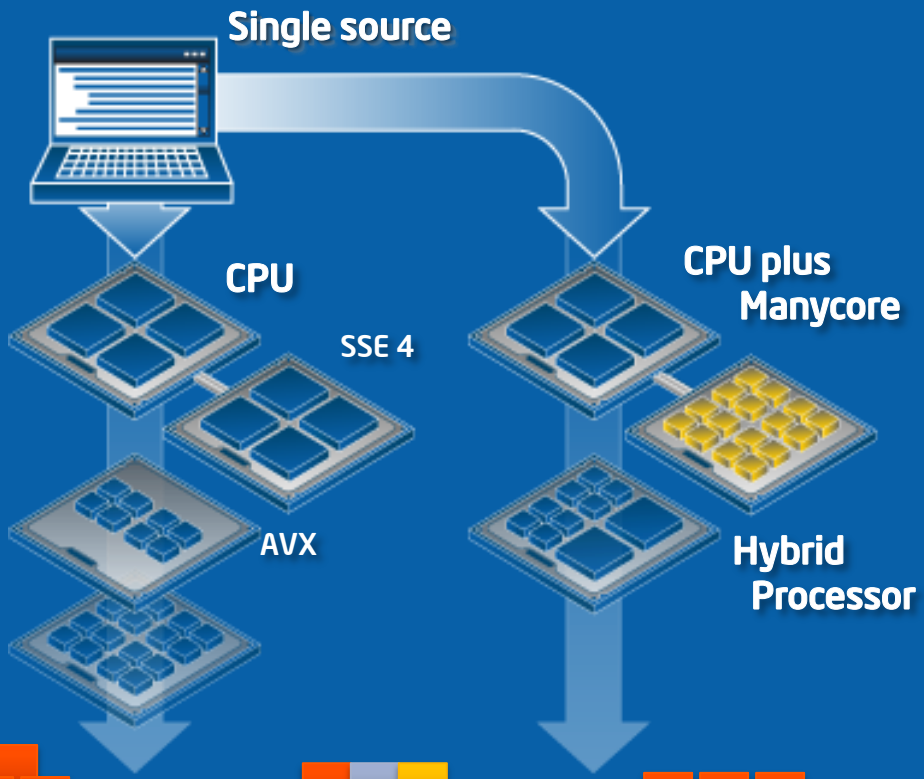
Agenda

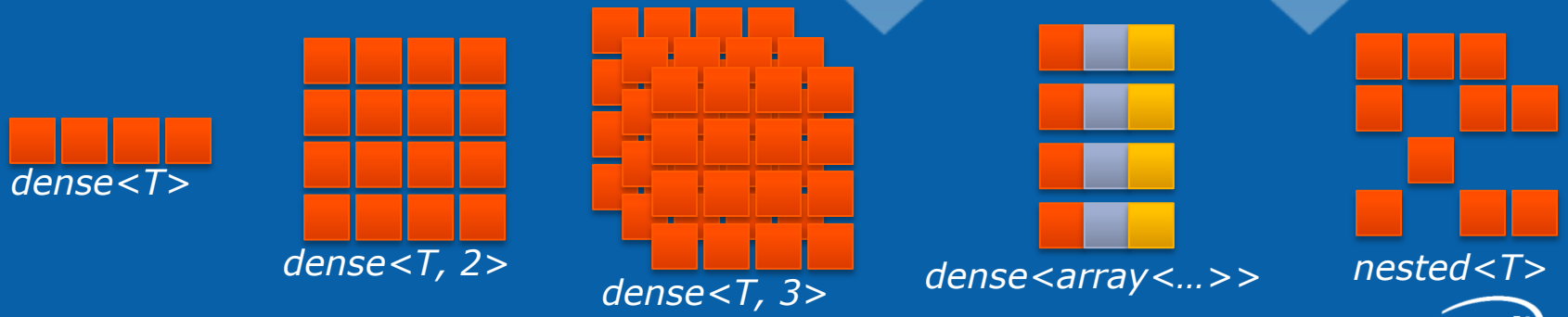
- Intel® Array Building Blocks
- VM Description and Rationale
- Programming Model
- Basic API Functionality and Examples
- Operations in the VM
- Runtime API
- Usage Models
- Finding Out More

Intel® Array Building Blocks

Key Points:

- Productivity
- Performance
- Portability

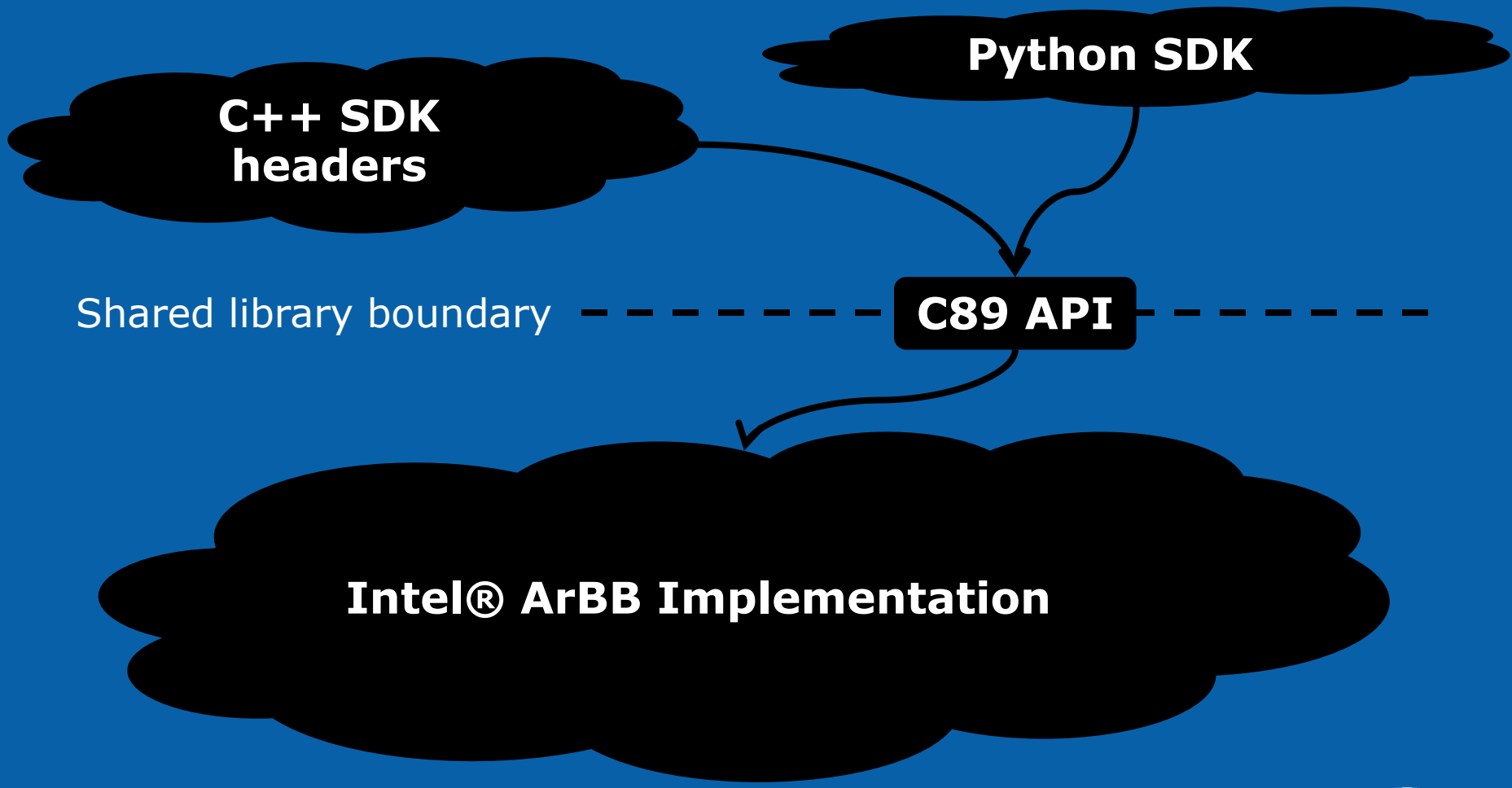




Software and Services Group

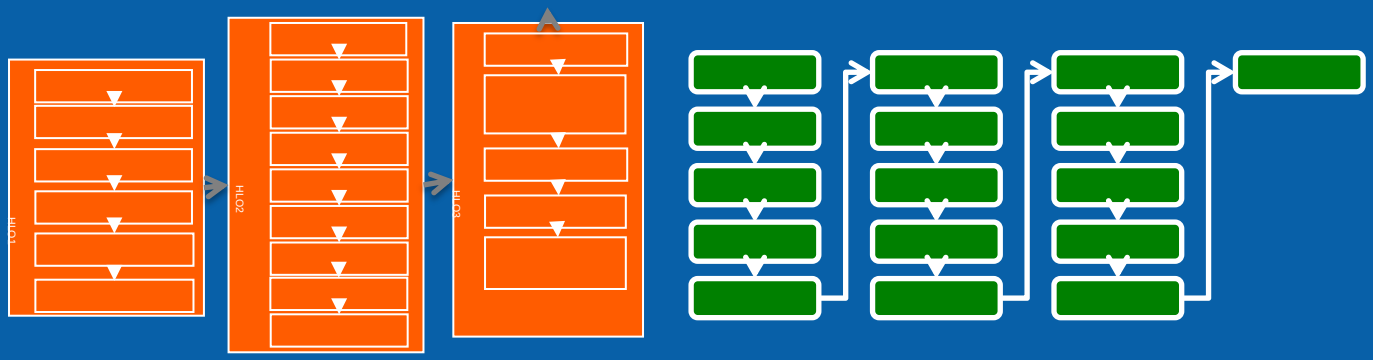
intel Software

VM Rationale - Implementation

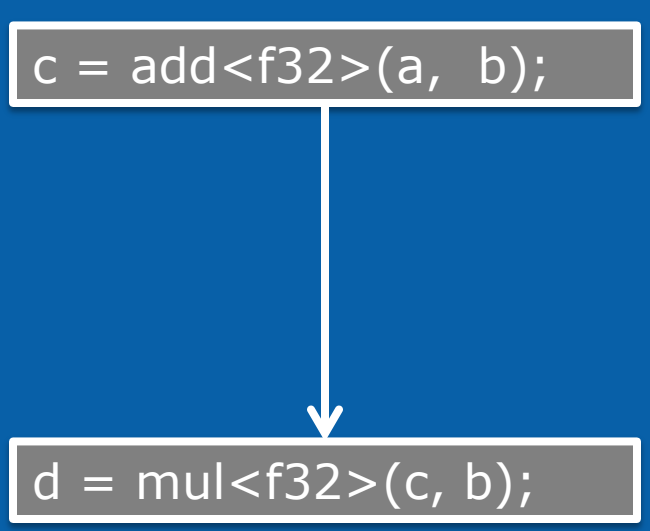


VM Rationale - Usage

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

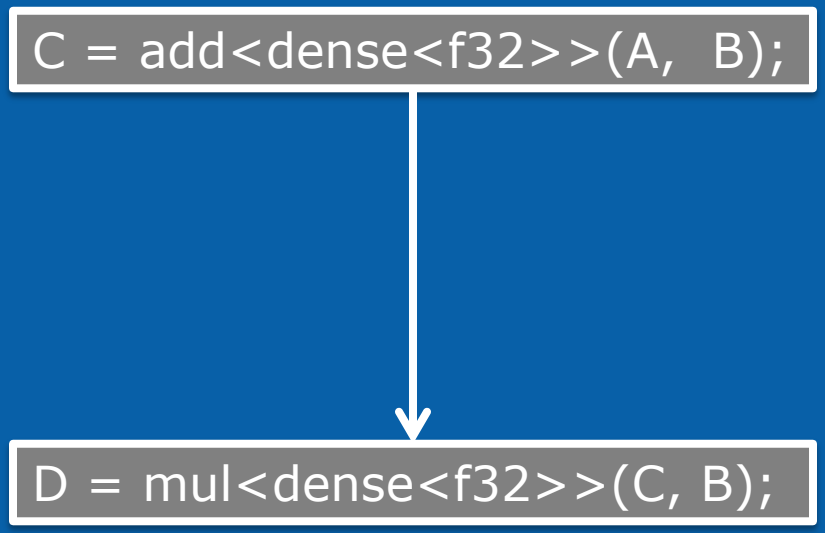


Programming Model – Serial Code

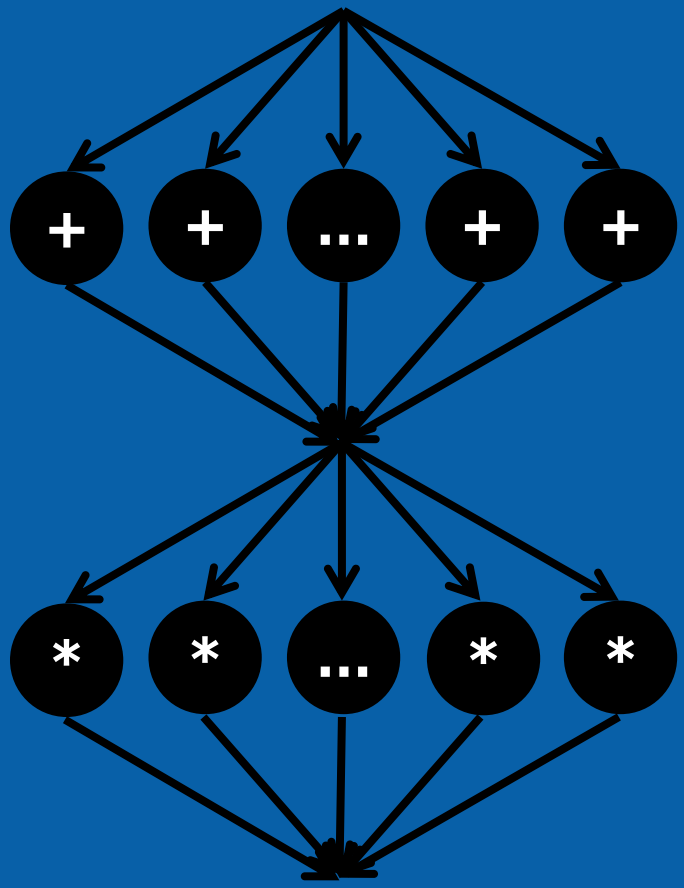


Programming Model - Parallelism

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



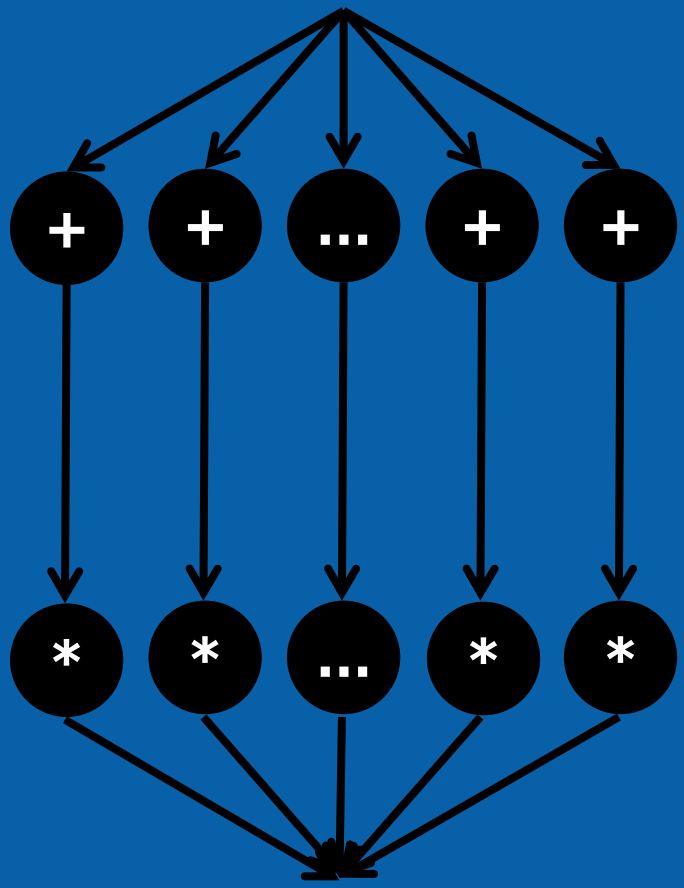
```
D = mul<dense<f32>>(C, B);
```



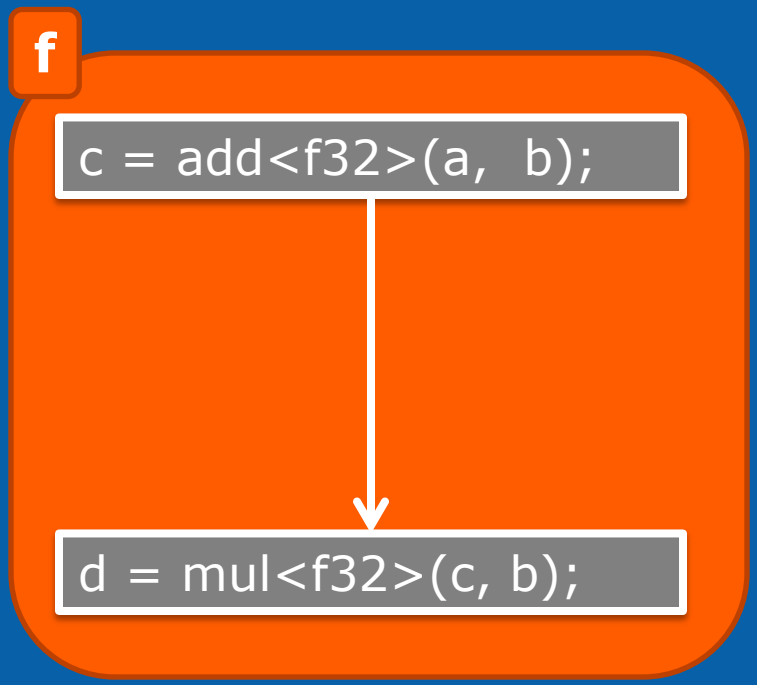
Programming Model – Optimization

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

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

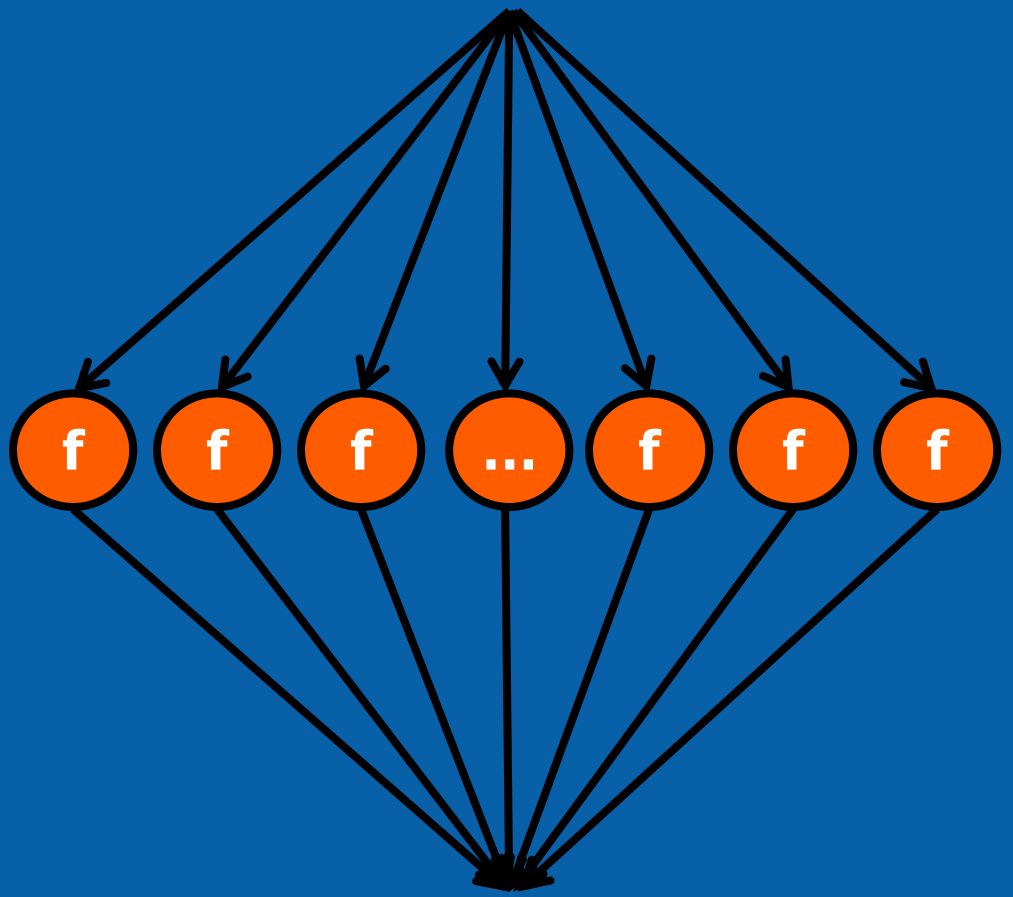


Programming Model – Functions



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

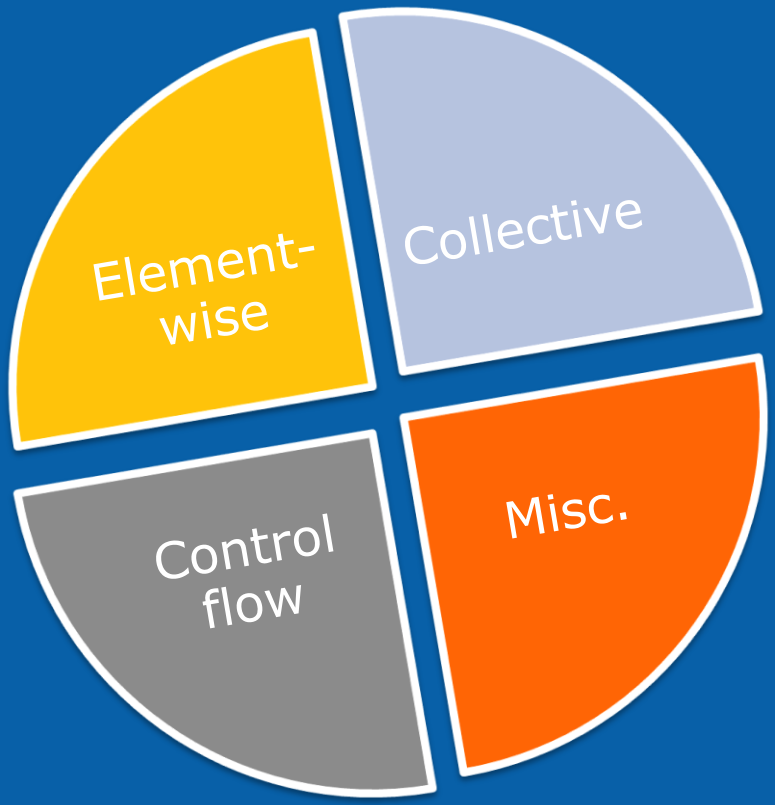
```
arbb_type_t dense_1d_f32;
arbb_get_dense_type(context, &dense_1d_f32, arbb_f32, 1, NULL);

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.

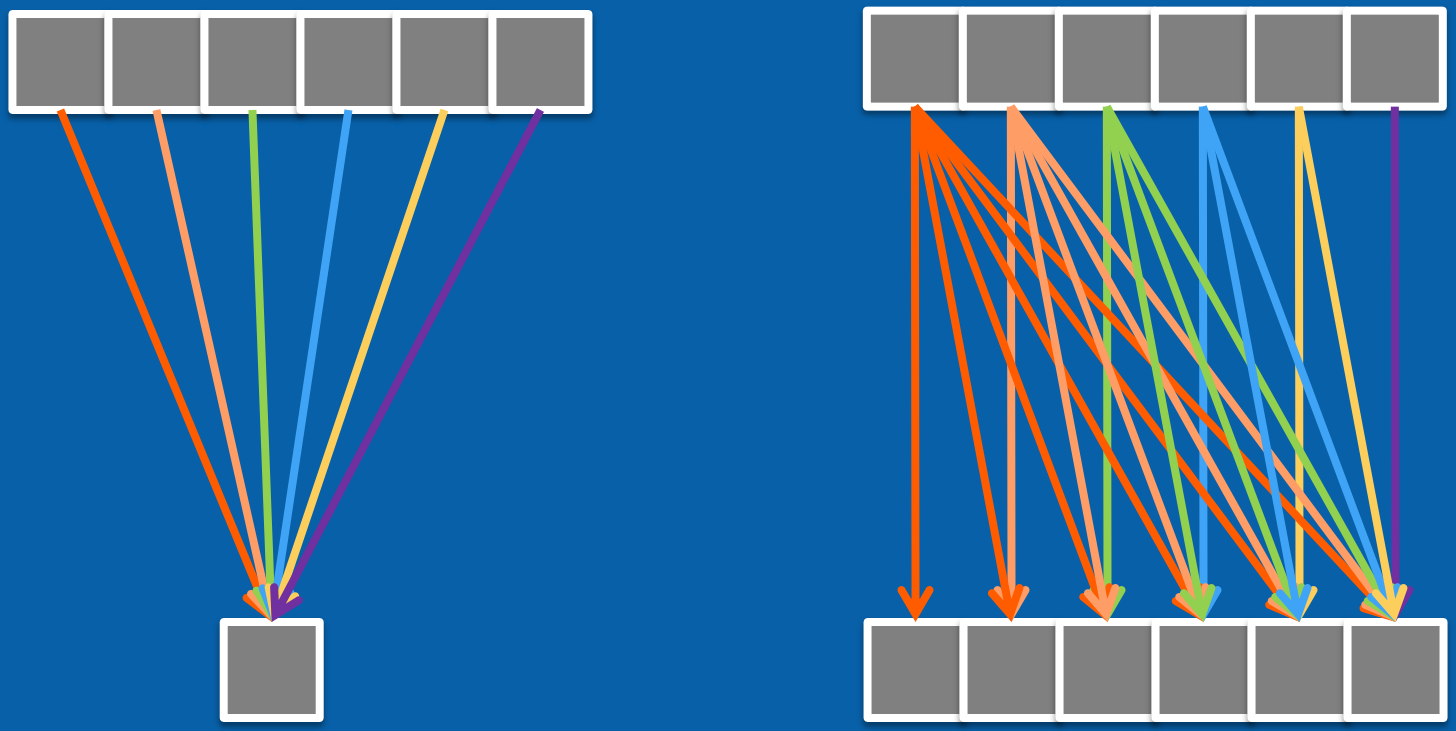
Operations



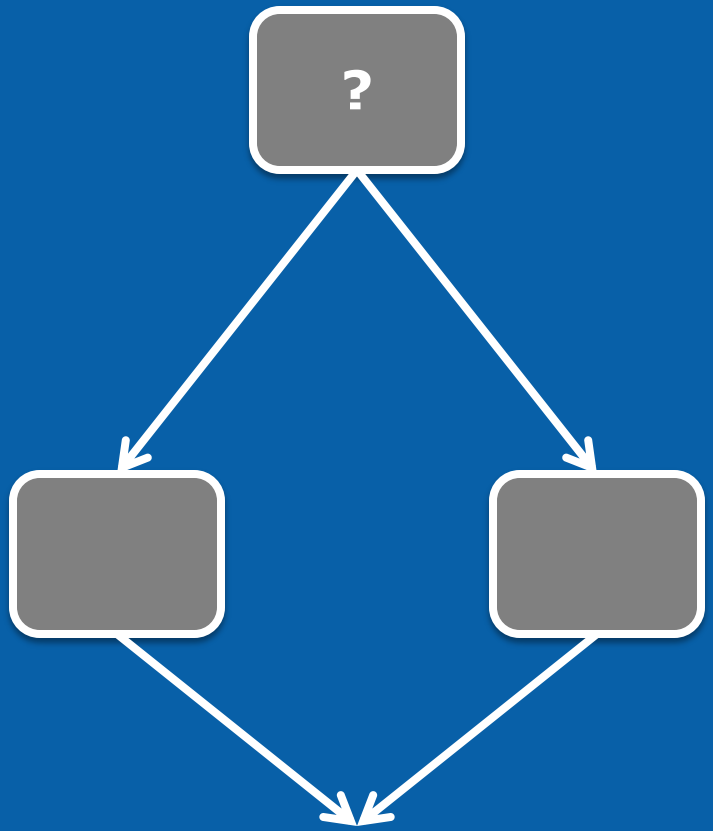
Element-wise operations

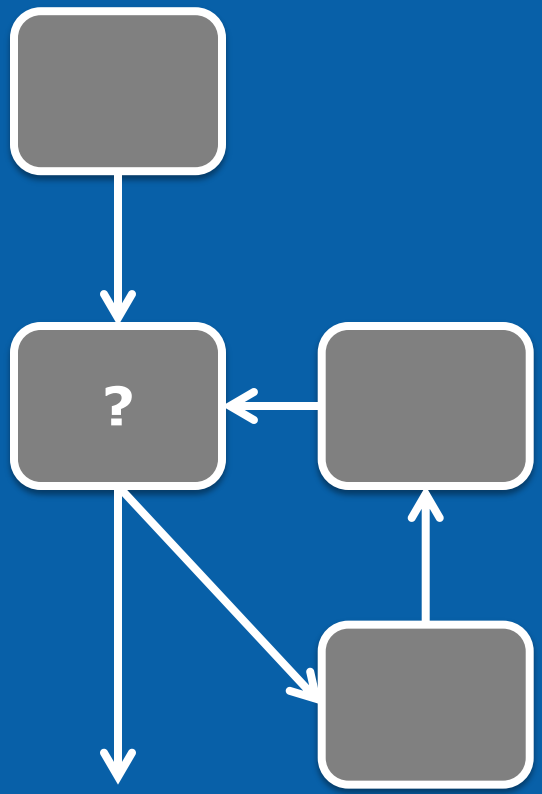
- Any scalar operations are allowed
- Source container shapes must match
  - Destination container size is irrelevant
- Any (but not all) sources can be scalars

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

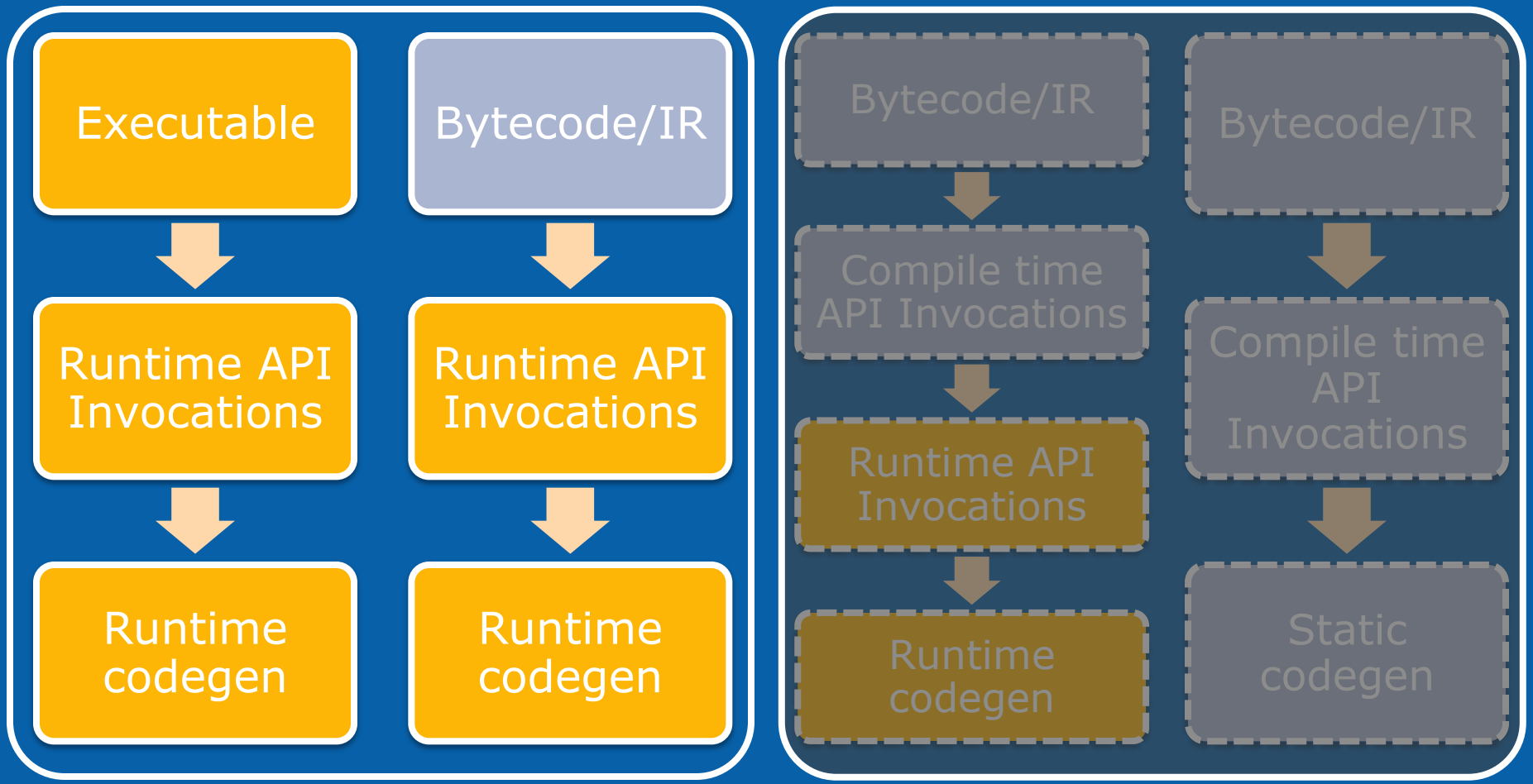
```
local dense<$f32, 1> _data;
local dense<$f32, 1> _result;
local dense<$isize, 1> _indices;
_result = gather<dense<$f32, 1>>(_data, _indices, $f32(0.0));
```


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/