Threading in oneAPI

oneAPI Threading Building Blocks (oneTBB)

oneAPI industry specification

- Specifies interfaces
- Does not prescribe implementation choices, including underlying threading models or specific accelerator support

Intel’s oneAPI product

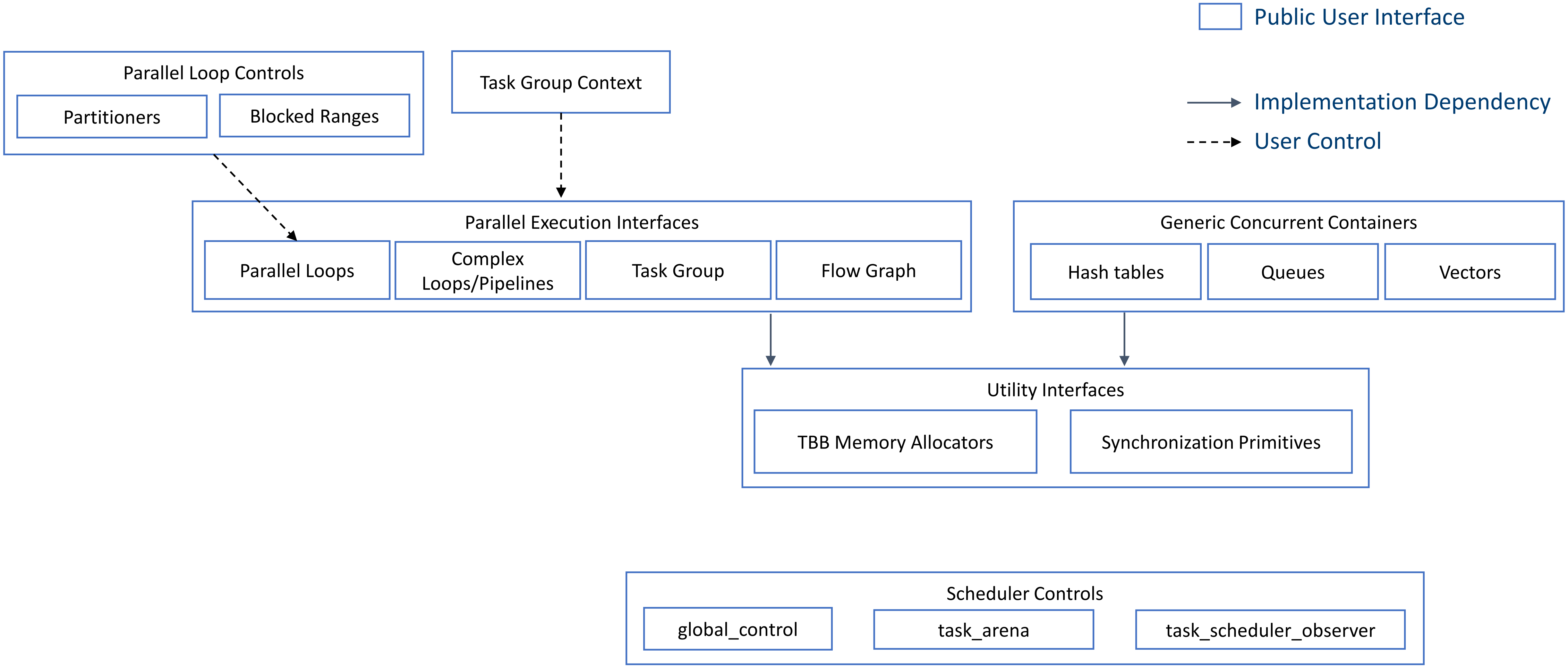
- Is an implementation of the oneAPI specification
- Makes choices about threading models and specific accelerator support

oneAPI specification | Intel’s implementation of oneAPI

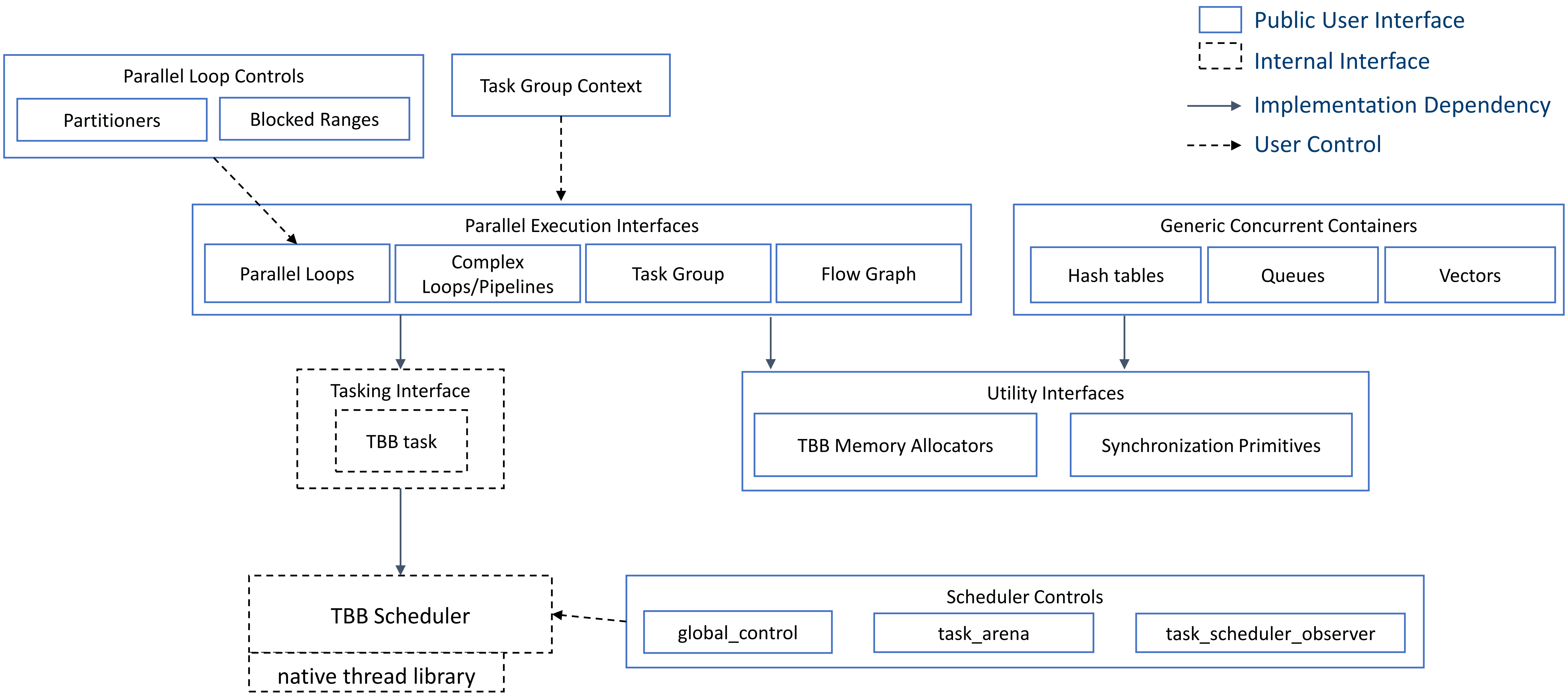
Two benefits of inclusion of oneTBB in oneAPI spec

- oneTBB provides threading features beyond those currently provided by DPC++
  - DPC++ == SYCL plus ISO C++ plus extensions
  - ISO C++ and SYCL are not (yet) sufficient to *easily* express all threading patterns for CPUs
  - oneTBB adds a tasking interface, concurrent containers, scalable memory allocators, flow graph, and additional generic parallel algorithms that work on the host only
- oneTBB can be used as a common target for libraries and user code that wish to compose well with each other on the host
  - The oneTBB specification provides interface guarantees for those that opt-in to oneTBB
  - By using oneTBB as a common threading model, components use oneTBB tasks and can share a single thread pool, increasing composability and avoiding over-subscription
  - Third party code can target oneTBB to compose with oneAPI implementations that target oneTBB on the host

oneAPI Threading Building Blocks (oneTBB) Architecture

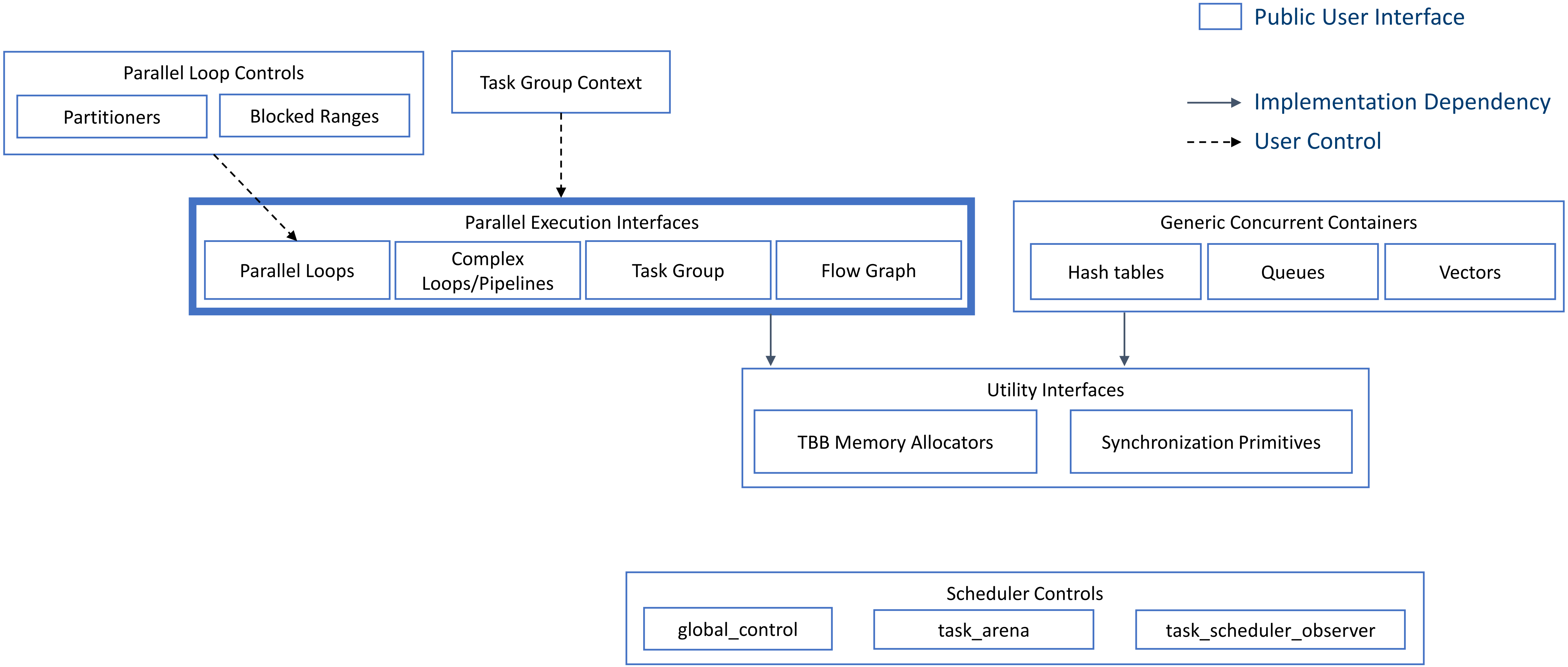


Intel’s implementation of oneTBB



Intel’s implementation of oneTBB

oneAPI Threading Building Blocks (oneTBB) Architecture



oneTBB Generic Parallel Algorithms

Loop parallelization

parallel_for

parallel_reduce

parallel_scan

Parallel sorting

parallel_sort

Parallel function invocation

parallel_invoke

Streaming

parallel_do

parallel_for_each

pipeline / parallel_pipeline

https://spec.oneapi.com/versions/latest/elements/oneTBB/source/algorithms.html

oneAPI specification

A oneTBB parallel_for example

```
for (int i = 0; i < max; i++) {
  for (int j = 0; j < max; j++) {
    p[i][j] = Complex-function(i, j);
  }
}
```

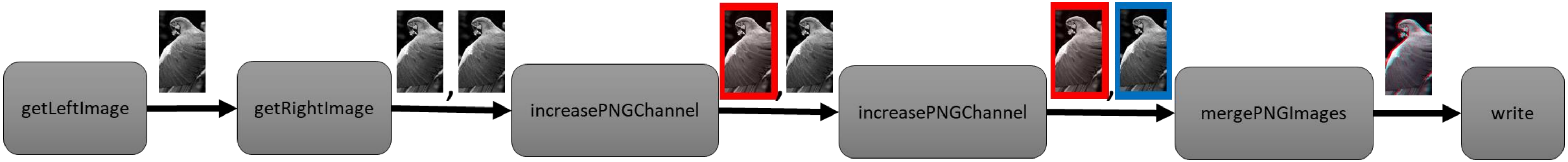
A serial loop that can be made parallel

The parallelism expressed with oneTBB

```
tbb::parallel_for(0, max, [&](int i) {
  for (int j = 0; j < max; j++) {
    p[i][j] = Complex-function(i, j);
  }
}
```

A oneTBB pipeline example

- Use to express linear pipelines of filters



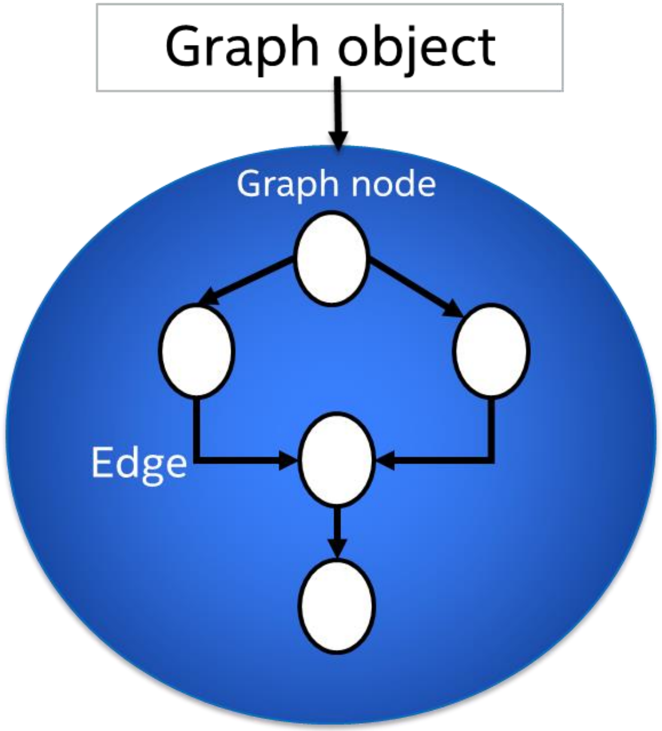
```
tbb::parallel_pipeline(  max_number_of_live_tokens,
                        make_filter<void,I1>(mode0,g0) &
                        make_filter<I1,I2>(mode1,g1) &
                        make_filter<I2,I3>(mode2,g2) &
                        ...
                        make_filter<In,void>(moden,gn) );
```

- The `max_number_of_live_tokens` limits concurrency and resource usage
- Modes are `parallel`, `serial_out_of_order`, or `serial_in_order`
- `g0..gn` are function objects
- The `operator&` concatenates filters.

This image was taken by Elena Adams and is from Halide http://halide-lang.org/tutorials/tutorial_lesson_02_input_image.html

oneTBB flow graph

- For dependency and data flow algorithms
- Used to exploit parallelism at higher levels
- Nodes execute as tasks and so are composable

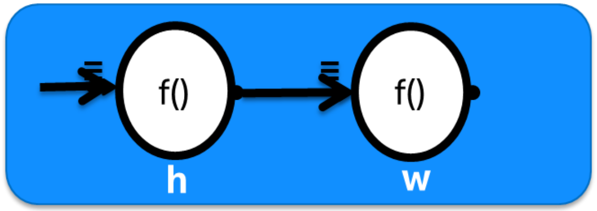


Graph object

Graph node

Edge

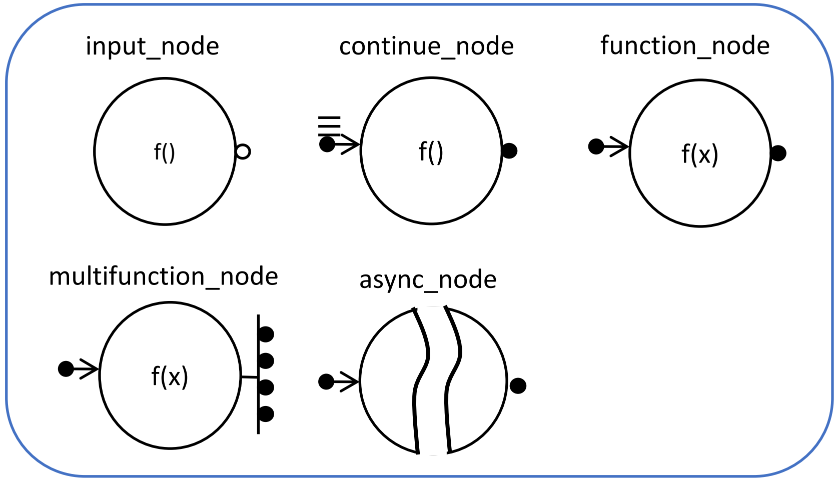
Hello World



```
graph g;
continue_node< continue_msg > h( g,
    []( const continue_msg & ) {
        cout << “Hello “;
    } );
continue_node< continue_msg > w( g,
    []( const continue_msg & ) {
        cout << “World\n“;
    } );
make_edge( h, w );
h.try_put(continue_msg());
g.wait_for_all();
```

oneTBB flow graph node types

Functional



input_node

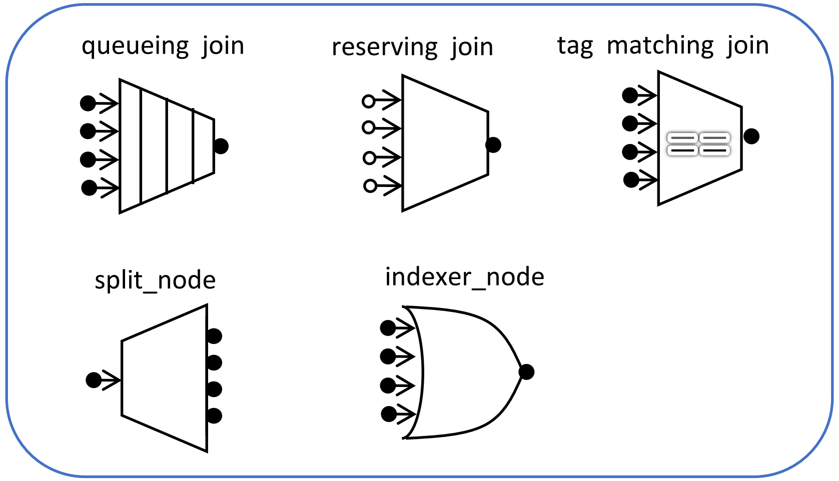
continue_node

function_node

multifunction_node

async_node

Split / Join



queueing join

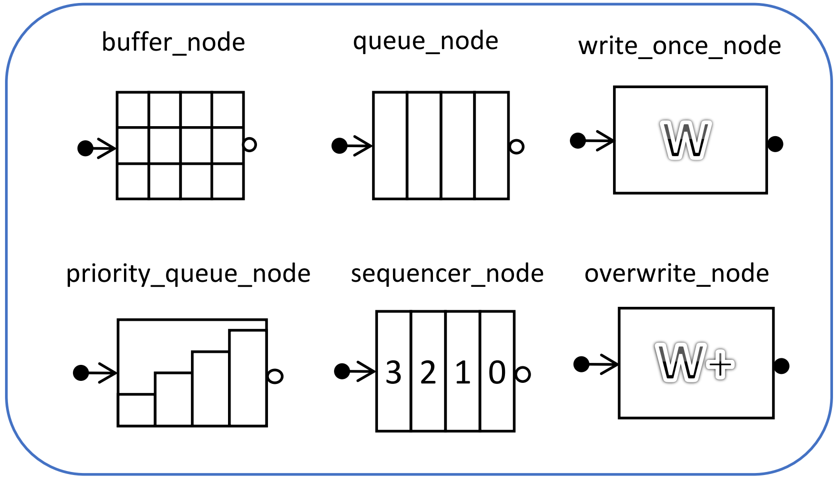
reserving join

tag matching join

split_node

indexer_node

Buffering



buffer_node

queue_node

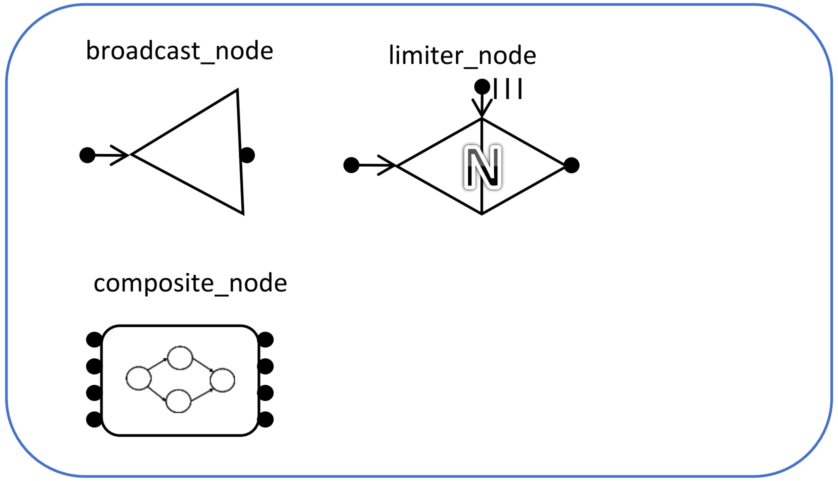
write_once_node

priority_queue_node

sequencer_node

overwrite_node

Other



broadcast_node

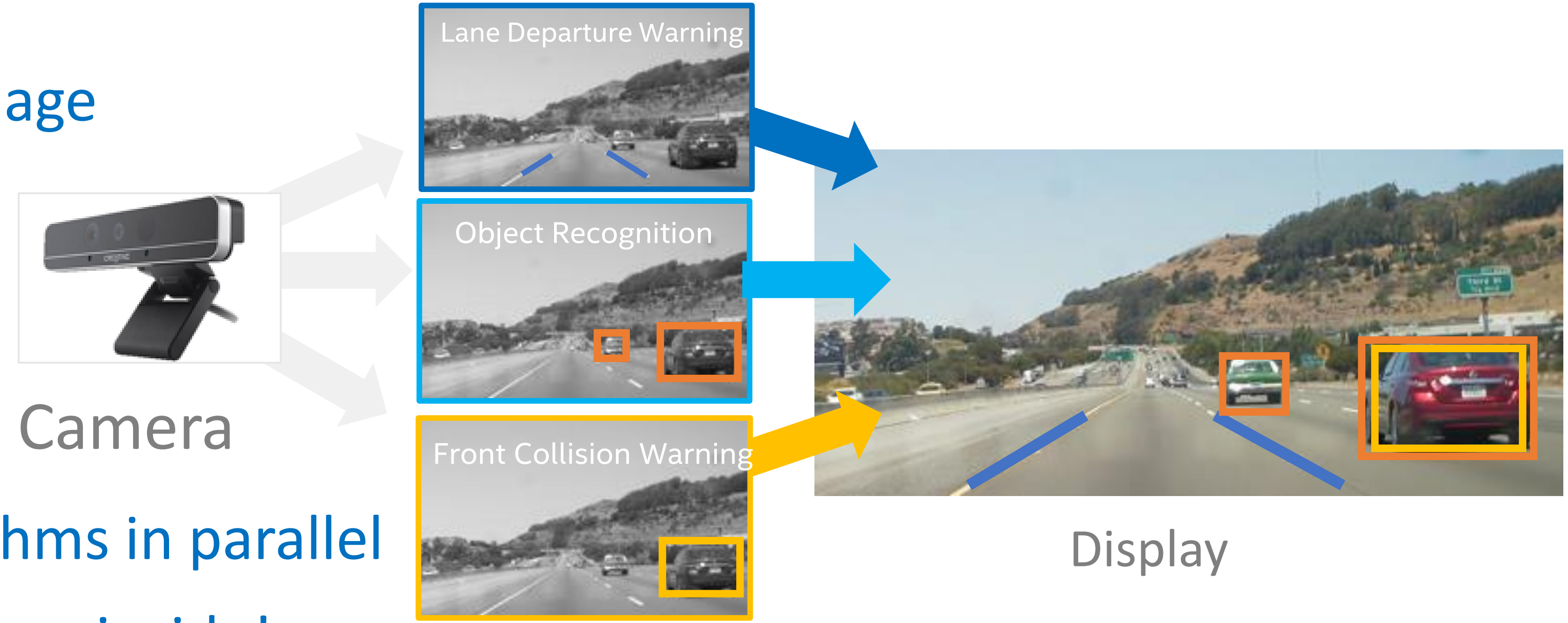
limiter_node

composite_node

A flow graph streaming example:

Advanced driver-assistance systems (ADAS) Application

- Classic Example for an image processing pipeline
  - Read input from sensor
  - Process Algorithms
  - Display result
- Executes different algorithms in parallel
- With nested TBB parallelism inside!

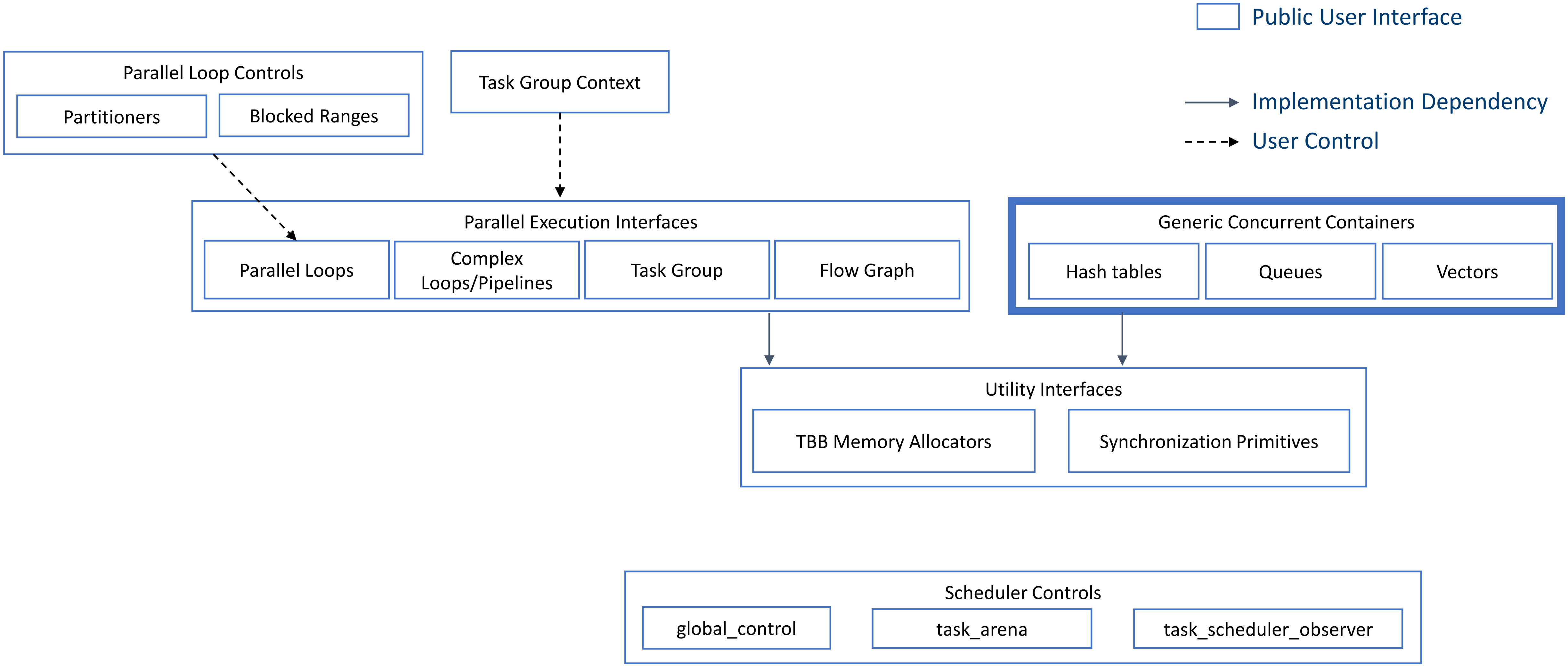


ADAS Use cases

Intel® Parallel Universe Article, Issue 30 (Oct'17):
Vasanth Tovinkere, Pablo Reble, Farshad Akhbari, Palanivel Guruvareddiar,
Driving Code Performance with Intel® Advisor Flow Graph Analyzer

Intel® Parallel Universe Article Issue, 44 (April'21):
Elvis Fefey, Michael Voss, Maxim Shevtsov,
oneTBB Flow Graph and the OpenVINO Inference Engine: Expressing Dependencies across Deep Learning Models in C++

oneAPI Threading Building Blocks (oneTBB) Architecture



oneTBB concurrent container example

```
std::queue q;

…

if (!q.empty()) {
  item = q.front();
  q.pop();
  /* process item */
}
```

std::queue

At this instant, another thread might pop the last element

oneTBB concurrent container example

```
std::queue q;

…

if (!q.empty()) {
  item = q.front();
  q.pop();
  /* process item */
}
```

std::queue

```
tbb::concurrent_queue<T> MyQueue;

…

T item;
if( MyQueue.try_pop(item) ) {
  /* process item */
}
```

tbb::concurrent_queue

oneTBB provides a try_pop function instead.

Migrating from Intel’s TBB 2020 to oneAPI’s oneTBB

oneTBB is NOT backwards compatible with TBB 2020

- Intel’s implementation of oneAPI Threading Building Blocks follows the oneTBB specification, which requires both API and ABI breaks
- Two highlights:
  - oneTBB’s controls for managing the number of threads is cleaner
  - The low-level (error-prone) TBB tasking API is no longer supported
- There is a migration guide: https://docs.oneapi.io/versions/latest/onetbb/tbb_userguide/Migration_Guide.html

oneAPI specification | Intel’s implementation of oneTBB

Migration Example: Spawning Individual Tasks

https://docs.oneapi.io/versions/latest/onetbb/tbb_userguide/Migration_Guide/Task_API.html#spawning-of-individual-tasks

```
#include <tbb/task.h>

int main() {
  // Assuming RootTask, ChildTask1, ChildTask2 are defined.

  RootTask& root = *new(tbb::task::allocate_root())
                      RootTask{};
  ChildTask1& child1 = *new(root.allocate_child())
                          ChildTask1{/*params*/};
  ChildTask2& child2 = *new(root.allocate_child())
                          ChildTask2{/*params*/};

  tbb::task::spawn(child1);
  tbb::task::spawn(child2);
  root.set_ref_count(3);
  root.wait_for_all();
}
```

TBB 2020 Code

```
#include <oneapi/tbb/task_group.h>

int main() {
    // Assuming ChildTask1, and ChildTask2 are defined.
    oneapi::tbb::task_group tg;
    tg.run(ChildTask1{/*params*/});
    tg.run(ChildTask2{/*params*/});
    tg.wait();
}
```

oneTBB option #1

```
#include <oneapi/tbb/parallel_invoke.h>

int main() {
    // Assuming ChildTask1, and ChildTask2 are defined.
    oneapi::tbb::parallel_invoke(
        ChildTask1{/*params*/},
        ChildTask2{/*params*/}
    );
}
```

oneTBB option #2

oneAPI specification | Intel’s implementation of oneTBB

Threading Models of Intel’s oneAPI *implementation*

	Description	Default Threading Model on Host	oneAPI threading resource composability	Support for 3rd party models
oneTBB	Algorithms, containers & task scheduler for threading on host	native threads	provides shared thread pool used by oneAPI components via oneTBB tasks and algorithms	Thread pool is not replaceable
DPC++	Direct programming of XPUs	oneTBB	via oneTBB	Thread pool is not replaceable
oneDPL	Data-parallel library features in the C++ standard plus extensions	oneTBB	via oneTBB or DPC++	Possible in principle via internal abstraction layer
oneMKL	Fundamental math routines	oneTBB	via oneTBB	OpenMP, set at link-time
oneDNN	Performance library for deep learning applications	oneTBB	via oneTBB	OpenMP, set at link-time. Or public thread pool API can change pool at library build time
oneDAL	Performance library for data analytics	oneTBB	via oneTBB	Possible in principle via internal abstraction layer
oneVPL	Performance library for video decoding, encoding and processing	Reuses application threads	Reuses calling threads	Customization support is planned

Intel’s implementation of oneAPI

Additional Resources

- The latest oneTBB specification
  - https://spec.oneapi.io/versions/latest/elements/oneTBB/source/nested-index.html
- An open-source implementation of oneTBB
  - https://github.com/oneapi-src/oneTBB
- Pro TBB: C++ Parallel Programming with Threading Building Blocks
  - An Open Access book that covers TBB 2019
  - https://www.apress.com/gp/book/9781484243978
  - Samples are currently being ported to oneTBB