Creating
World
Changing
Technologies

Creating
World
Changing
Technologies

Exploiting Heterogeneous Computing with Intel oneAPI Threading Building Blocks (oneTBB)

James Tullos
Technical Consulting Engineer, Intel Corporation

Notices & Disclaimers

Intel technologies may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.

Your costs and results may vary.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

Optimization Notice: Intel's compilers 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 SSE2, SSE3, and 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. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice Revision #20110804
https://software.intel.com/en-us/articles/optimization-notice

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors.

Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. See backup for configuration details. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details. No product or component can be absolutely secure.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

intel.

Agenda

- oneTBB Introduction
- oneTBB Task Execution Model
- oneTBB Code Examples

Intel® oneAPI Threading Building Blocks (oneTBB) - Advanced Scaling for Fast Applications

- Flexible C++ Library for Parallelism
- Future Proof & Scale Application Performance
- Compatible with other Threading Packages
- Simplified & Enhanced Application Composability

Part of the Intel® oneAPI Base Toolkit

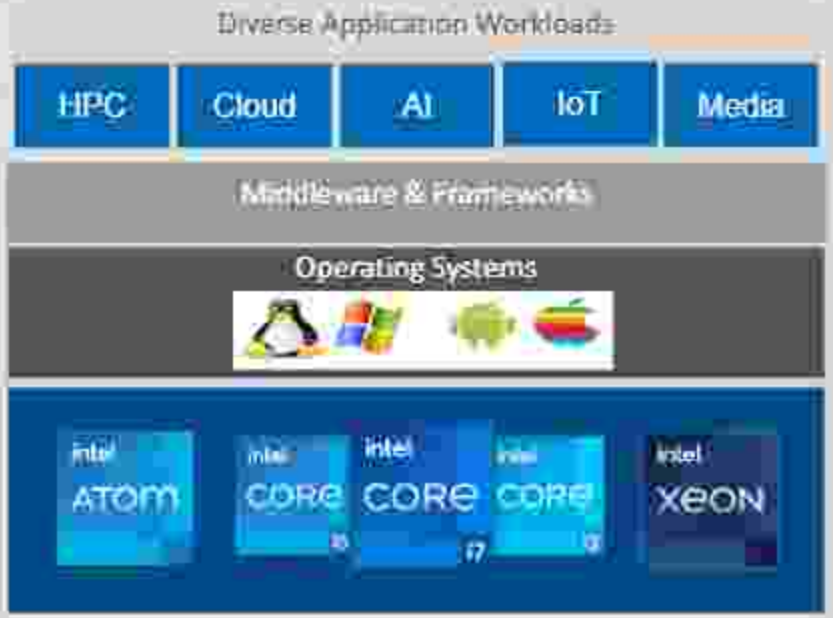
Multi-Threading & Heterogeneous Computing made easy with Intel® oneAPI Threading Building Blocks

What is oneTBB?

A highly templatized C++ library designed to simplify the task of adding CPU parallelism to your application and interoperate with accelerator offload code.

Why should you use oneTBB?

- High Performance
- Easy to use APIs
- Faster Time To Market
- Production Ready & Scalable



How to get oneTBB?

- Intel® oneAPI Base Toolkit
- As standalone component
- Open-source version

Key Applications

- Animation Rendering
- Numeric weather prediction
- Oceanography & Astrophysics
- Artificial intelligence & Automation
- Genetic Engineering
- Medical applications (image processing, MRI reconstruction)
- Remote sensing applications
- Socio Economics
- Financial sector (stock derivative pricing, statistics)
- Bulk updating data files
- Any Big Data problems

Find out more at: http://software.intel.com/intel-tbb
Contact us through our forum:
http://software.intel.com/en-us/forums/intel-threading-building-blocks

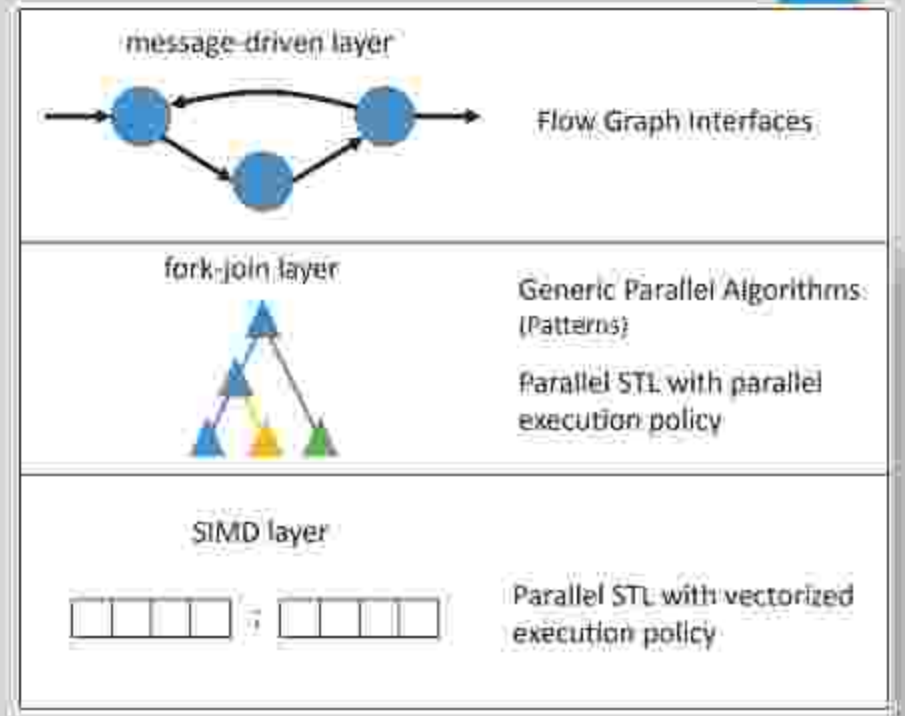
intel.

Advantages of Using Intel® oneAPI TBB over other Threading Models

- Specify tasks instead of manipulating threads.
- oneTBB uses proven , efficient parallel patterns.
- oneTBB uses work stealing to support load balancing.
- Flow graph feature in Intel oneTBB allows developers to easily express dependency and data flow graphs.
- Includes high level parallel algorithms and concurrent containers and low-level building blocks.

Three Parallelism Levels in oneTBB

- Message driven layer - (oneTBB Flow Graph)
- Fork-join layer - (oneTBB Tasks)
- Single Instruction Multiple Data (SIMD) layer

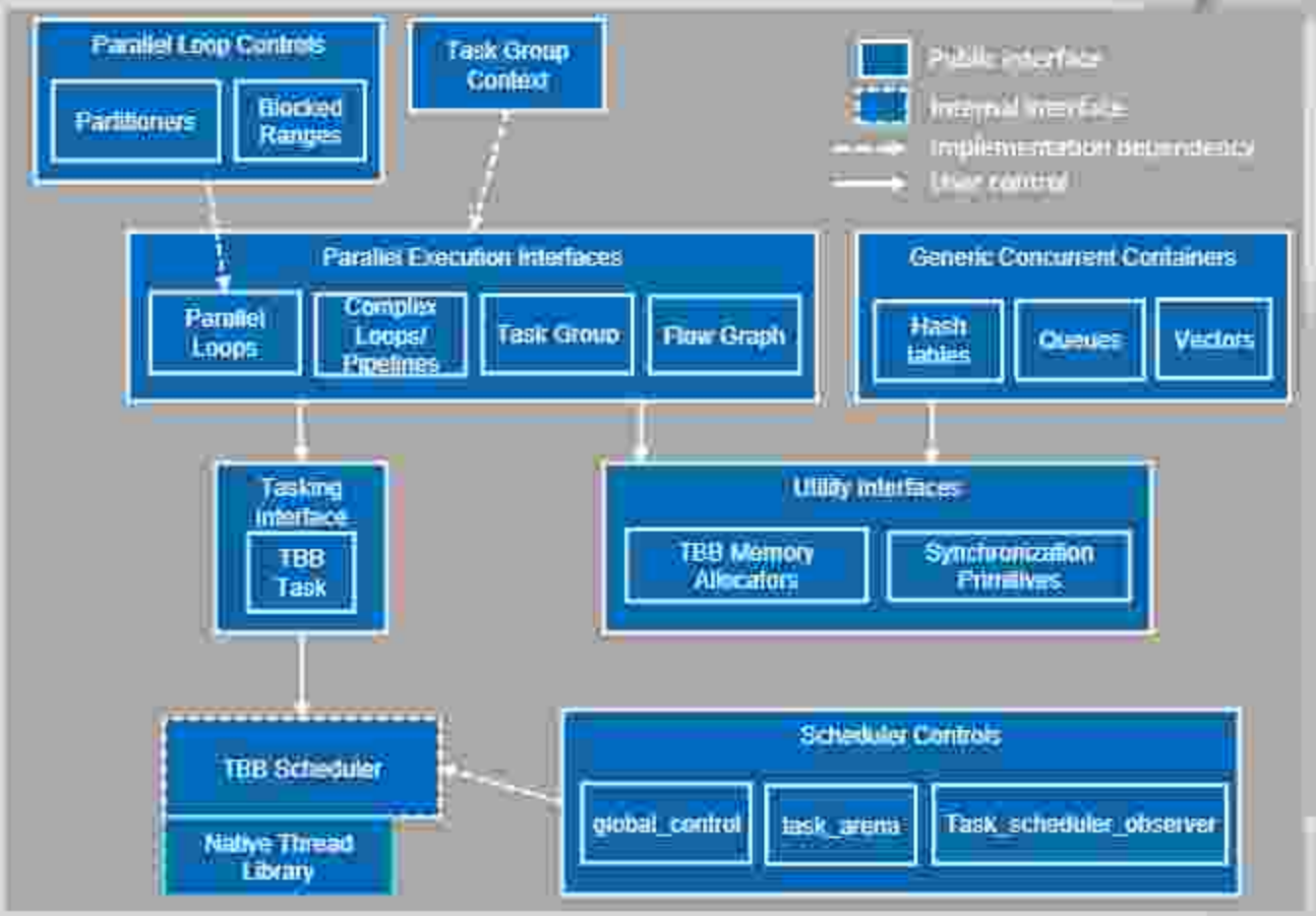


Source: Pro TBB

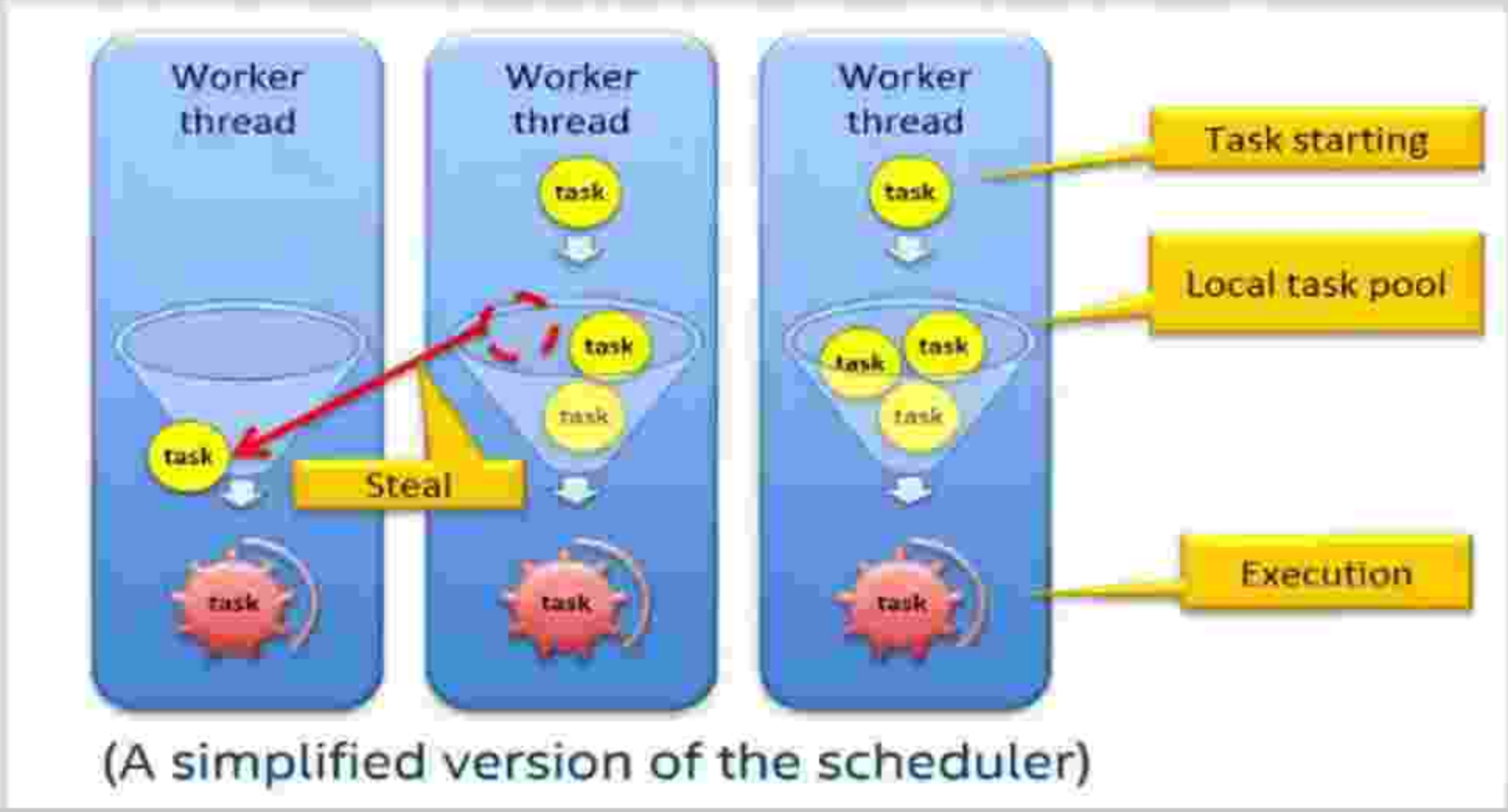
oneTBB Architecture Overview

oneTBB is a Collection of Building Blocks to Develop Highly-scalable Threaded Applications

- oneTBB includes high-level parallel execution interfaces
  - **Parallel Loops**: parallel_for, parallel_reduce, etc.
  - **Complex Algorithms**: pipelines, task groups
  - **Flow Graph**: Expressing data flow independent graphs
- Built on TBB tasks executed on TBB scheduler
- Controls for scheduler and parallel loops
- Concurrent Containers - Queues, Vectors, etc. are thread safe and thread friendly
- Scalable memory allocator, synchronization primitives



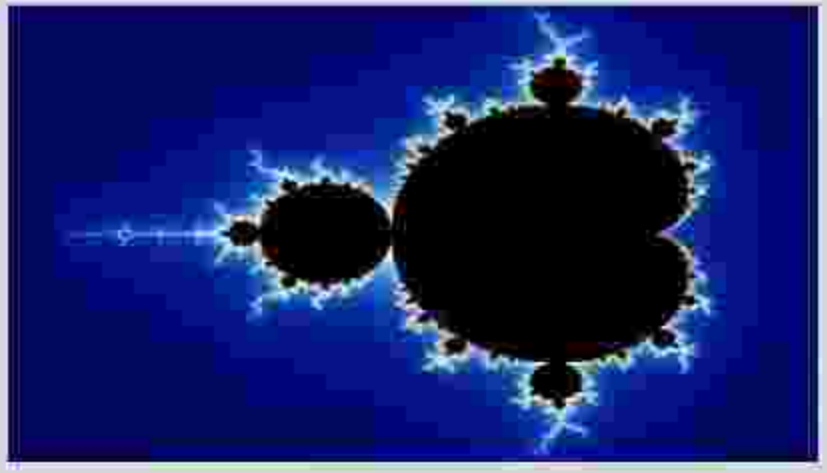
Task Execution In oneTBB



Generic Algorithms Allow Reuse of Proven Parallel Patterns

```
int mandel(Complex c, int max_count) {

 int count = 0; Complex z = 0;

 for (int i = 0; i < max_count; i++) {

   if (abs(z) >= 2.0) break;

   z = z*z + c; count++;
for (int i = 0; i < max_row; i++) {

 for (int j = 0; j < max_col; j++ ) {

   p[i][j] = mandel( Complex(scale(i), scale(j)), depth);

 }

}
```

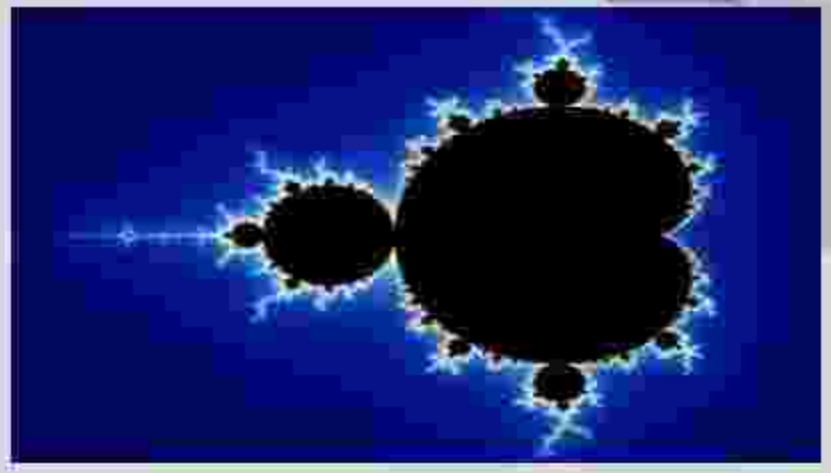
Sequential Version



For each point, is z = z*z + c bounded?

Mandelbrot Speedup

```
int mandel(Complex c, int max_count) {

 int count = 0; Complex z = 0;

 for (int i = 0; i < max_count; i++) {

  if (abs(z) >= 2.0) break;

 Parallel algorithm

 }

 return count;

}
```

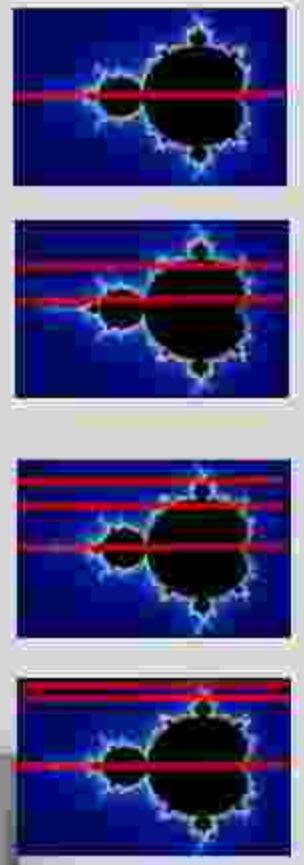


Task is a function object

Use C++ lambda functions to define function object in-line

```
 }

};
```

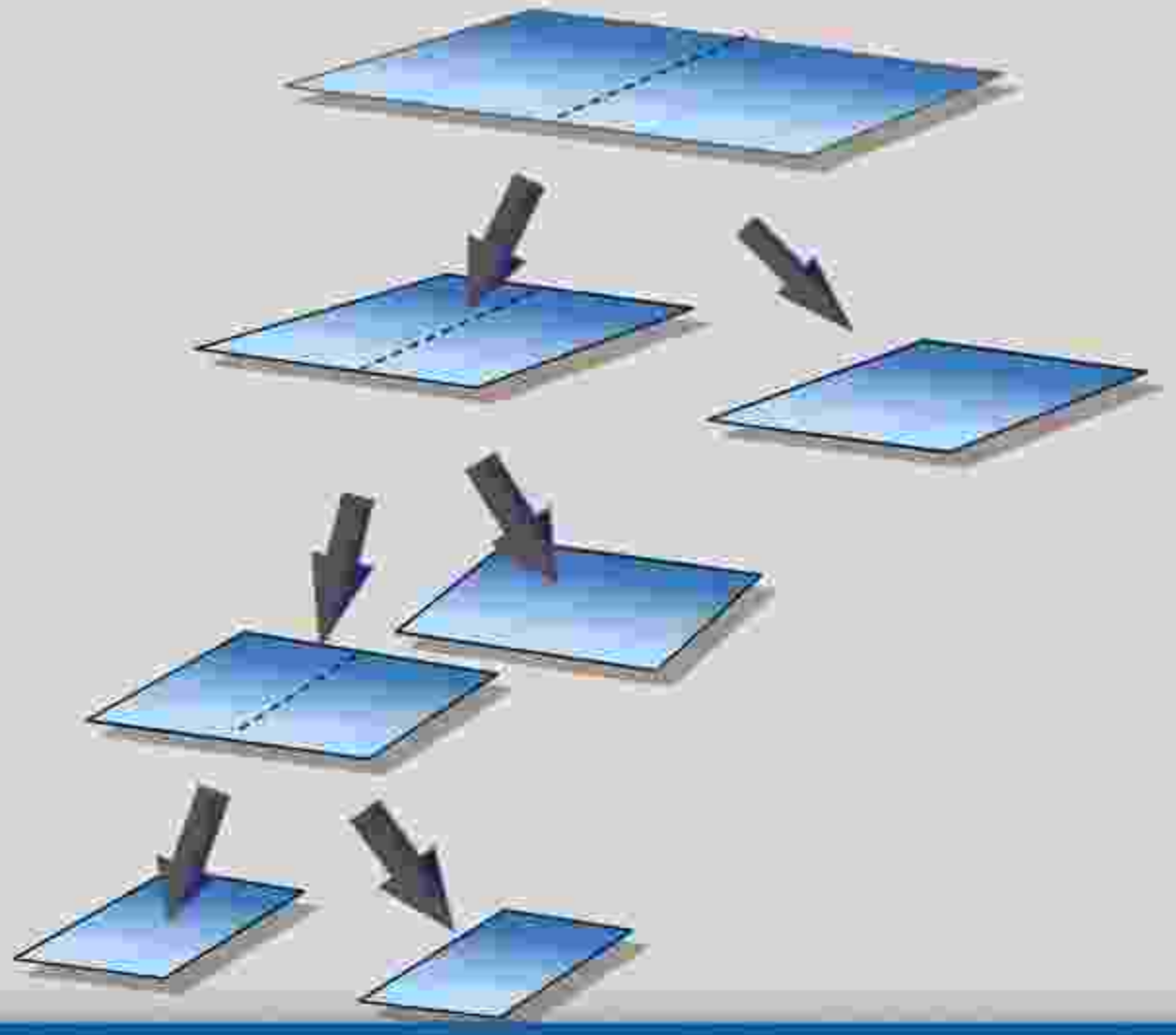
A parallel_for recursively divides the range into subranges that execute as tasks - Intel® oneAPI Threading Building Blocks (oneTBB)



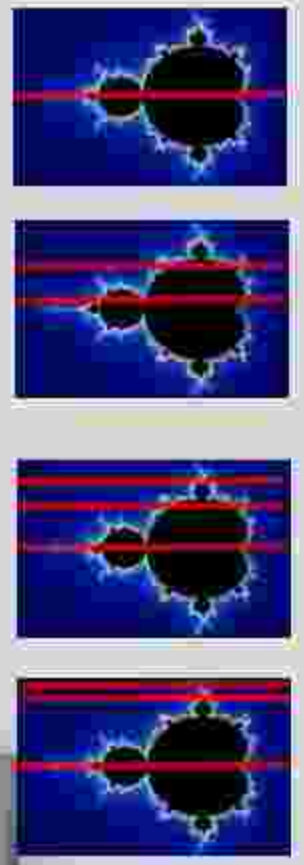
Split range...

.. recursively...

...until ≤ grainsize.



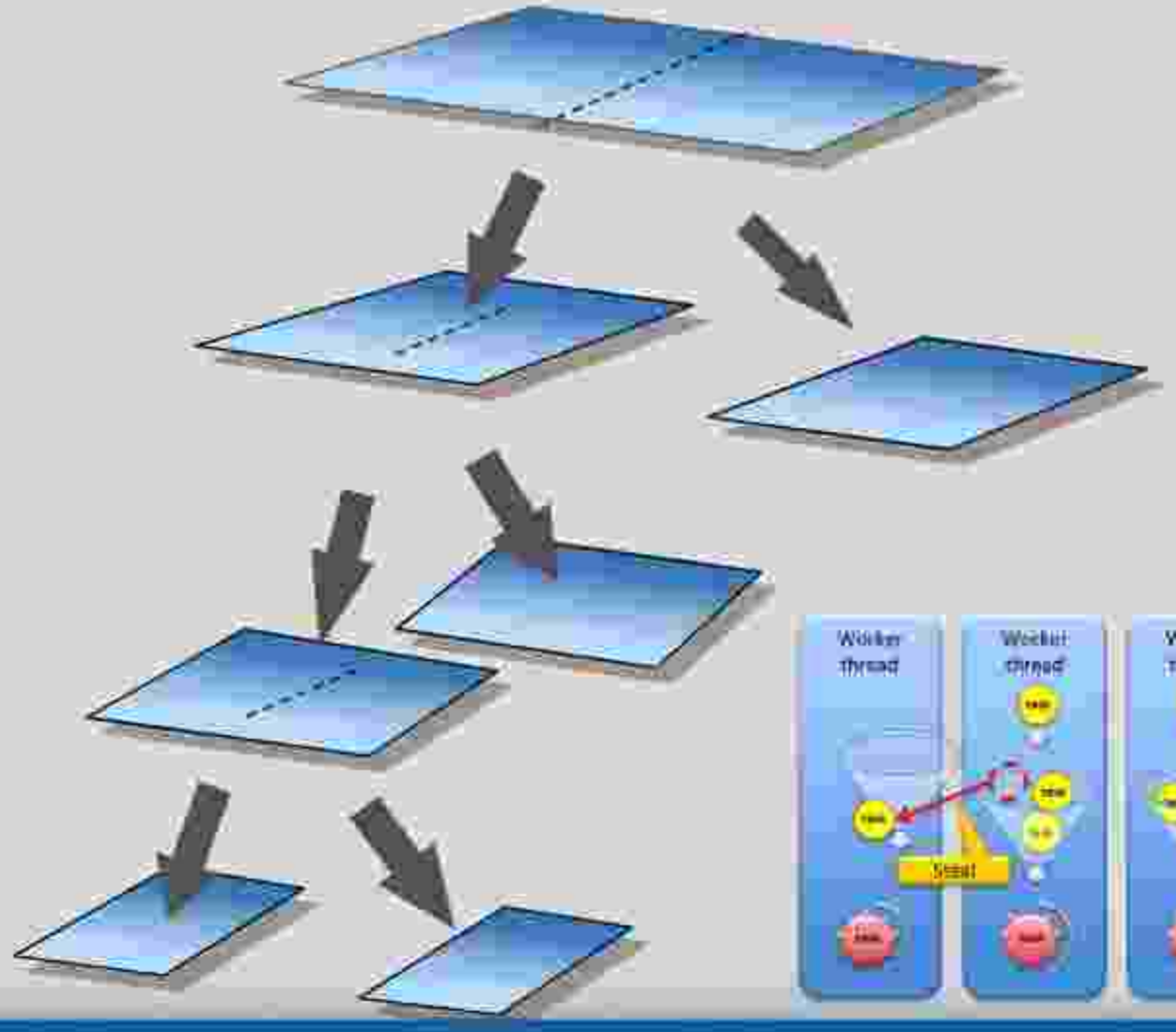
A parallel_for recursively divides the range into subranges that execute as tasks - Intel® oneAPI Threading Building Blocks (oneTBB)



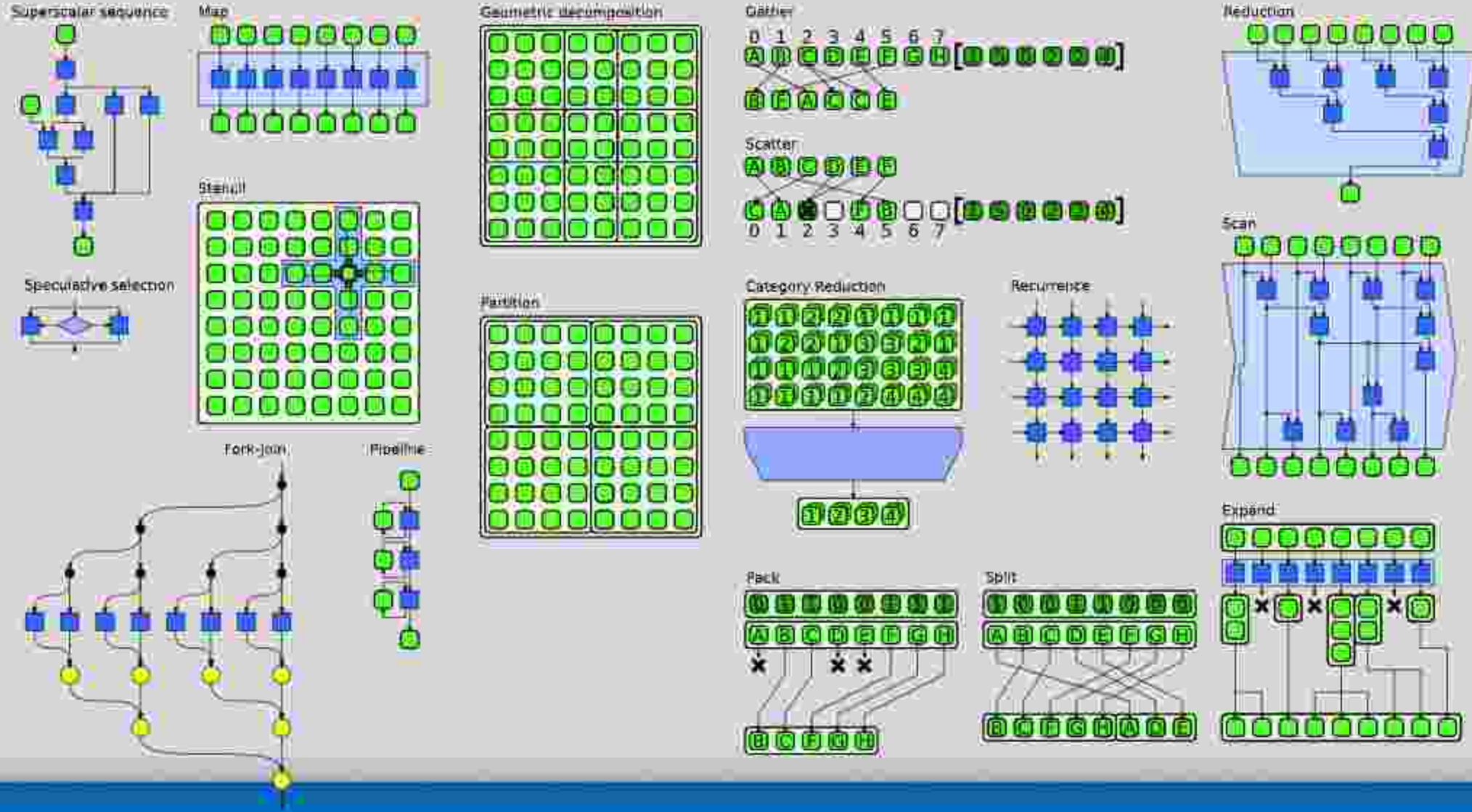
Split range...

.. recursively...

...until ≤ grainsize.



Examples of Parallel Algorithms



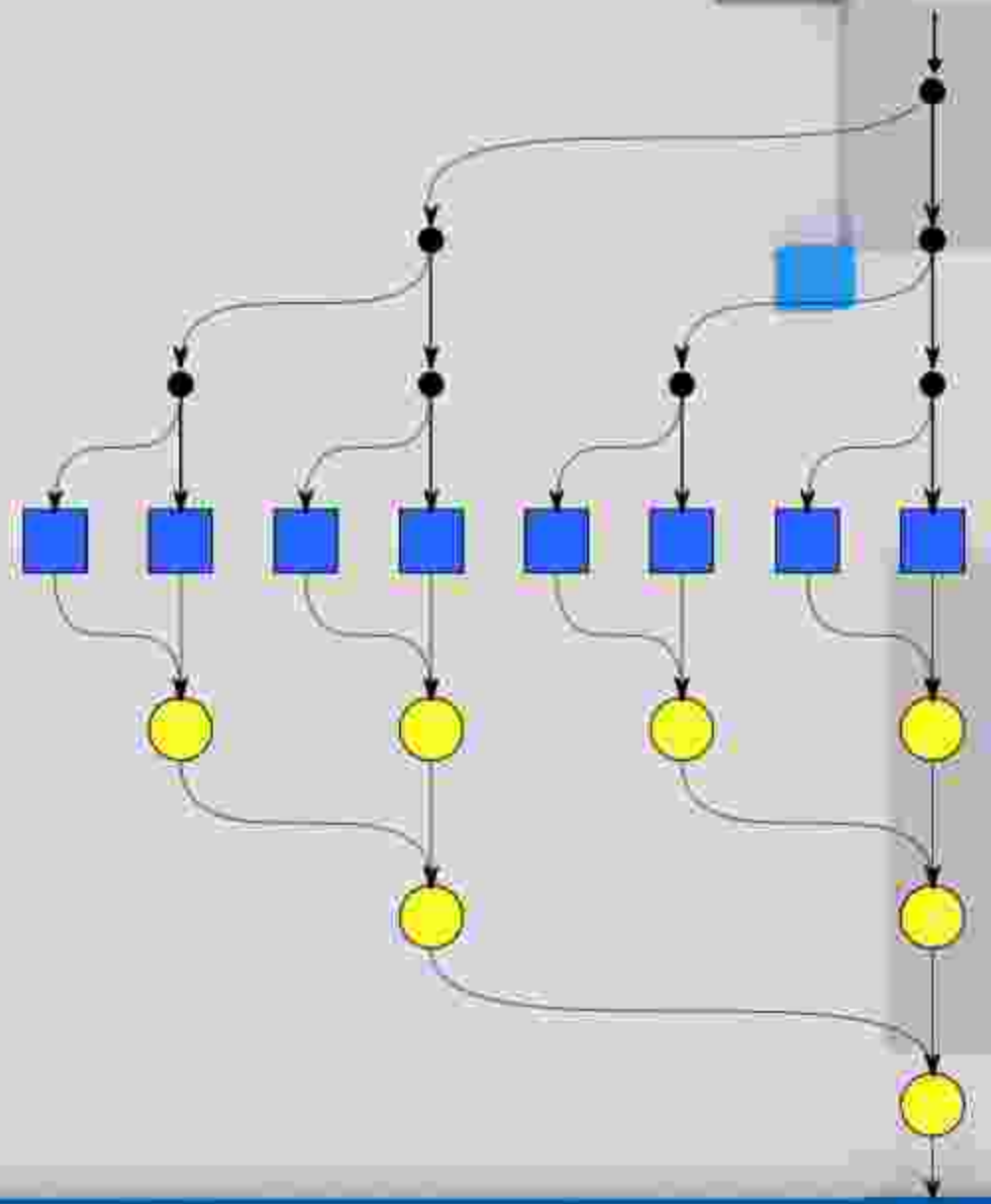
Fork-Join model

For small, known in advance number of tasks

```
parallel_invoke( func1, func2, ...);
```

For large or unknown in advance number of tasks

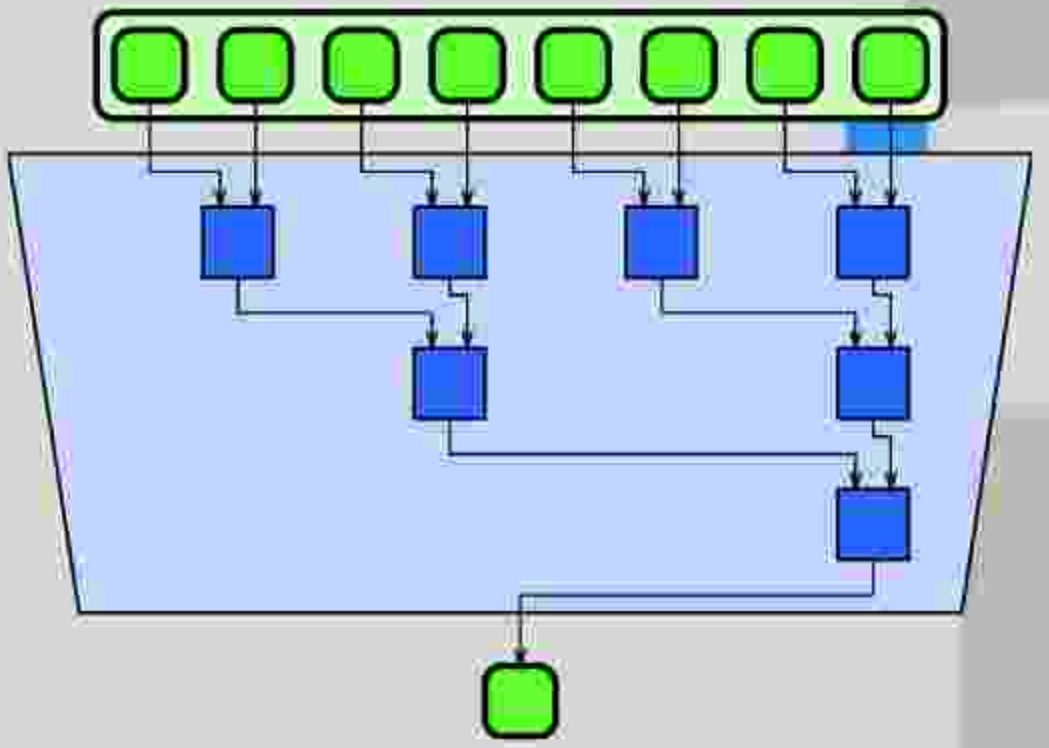
```
task_group g;
...
g.run( func1 );
...
g.run( func2 );
...
g.wait();
```



Reduce Pattern

With **parallel_reduce** function

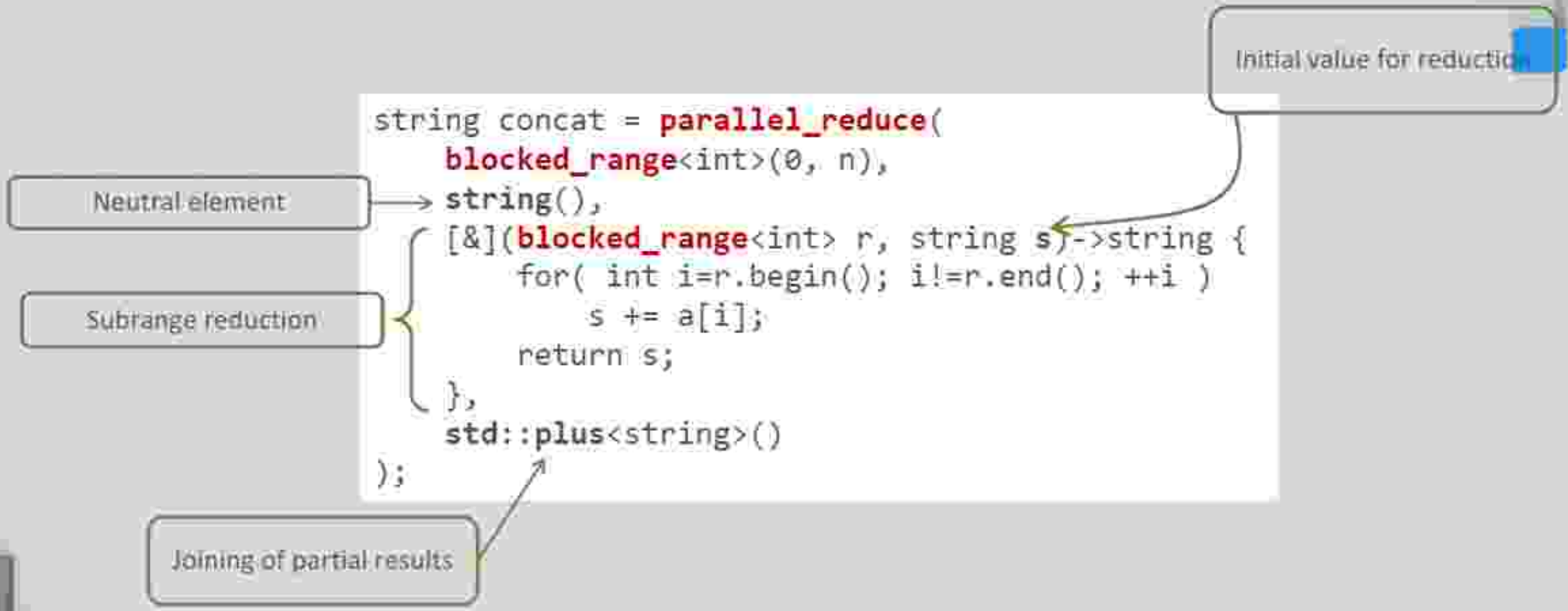
```
T sum = parallel_reduce(
    blocked_range<int>(0,n),
    0.f,
    [&](blocked_range<int> r, T s) -> T {
        for( int i=r.begin(); i!=r.end(); ++i )
            s += a[i];
        return s;
    },
    std::plus<T>()
);
```



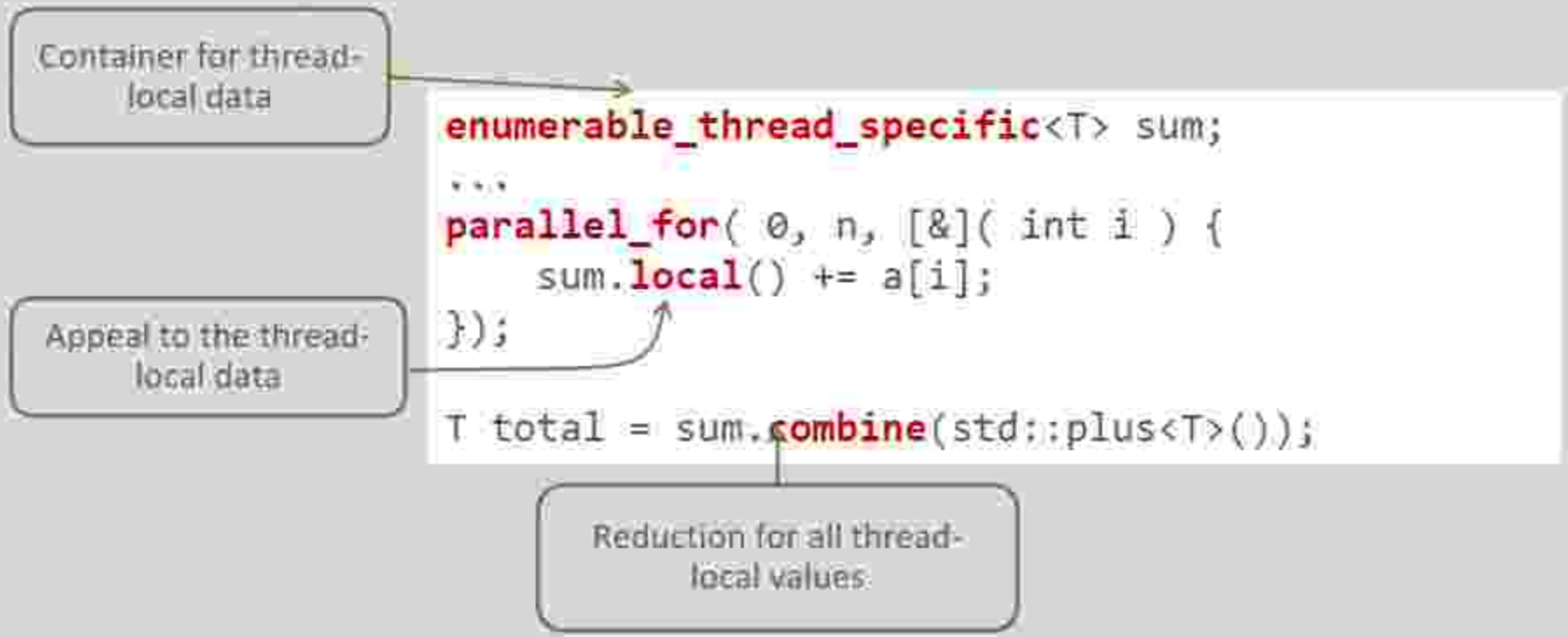
Using **enumerable_thread_specific** class

```
enumerable_thread_specific<T> sum;
parallel_for( 0, n, [&]( int i ) {
    sum.local() += a[i];
});
T total = sum.combine(std::plus<T>());
```

Reduction Using parallel_reduce Function



Reduction Using enumerable_thread_specific class



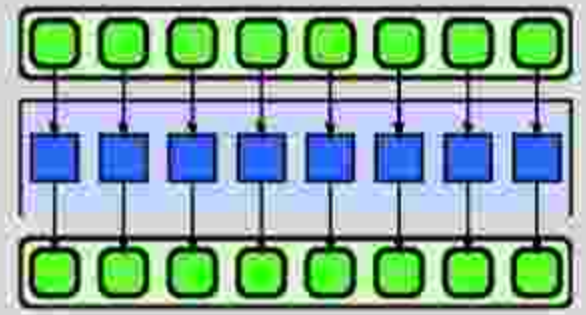
Intel® Threading Building Blocks. Map pattern

Apply *functor(i)* to all $i \in [lower, upper)$

```
parallel_for( lower, upper, func );
```

Apply *functor(i)*, changing *i* in a given step

```
parallel_for( lower, upper, stride, func );
```



Apply *functor(subrange)* to all *subranges* in the *range*

```
parallel_for( range, func );
```

Examples of parallel_for

```
void saxpy( float a, float x[], float (&y)[], size_t n ) {
    tbb::parallel_for( size_t(0), n, [&]( size_t i ) {
        y[i] += a * x[i];
    });
}
```

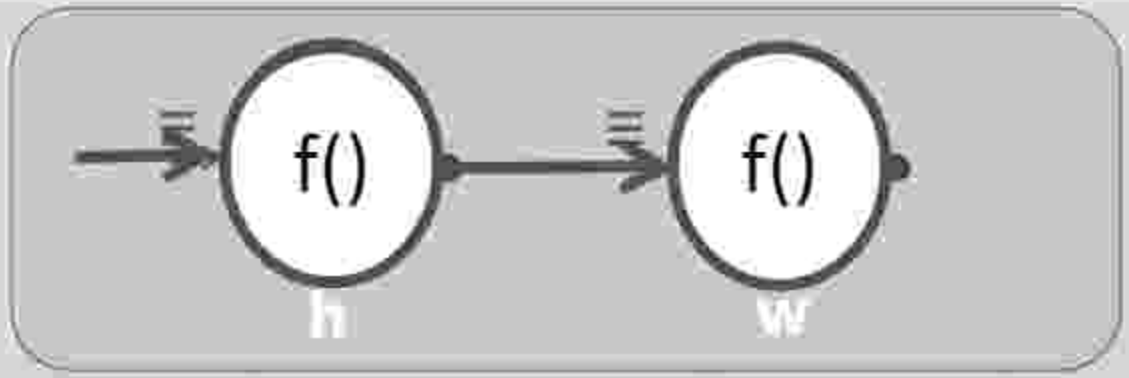
```
void saxpy( float a, float x[], float (&y)[], size_t n ) {
    size_t grain_size = 1000;
    tbb::parallel_for( tbb::blocked_range<size_t>(0, n, grain_size),
    [&]( tbb::blocked_range<size_t> r ) {
        for( size_t i = r.begin(); i != r.end(); ++i )
            y[i] += a * x[i];
    }
    );
}
```

Flow Graph Hello World Example (C++17+preview)

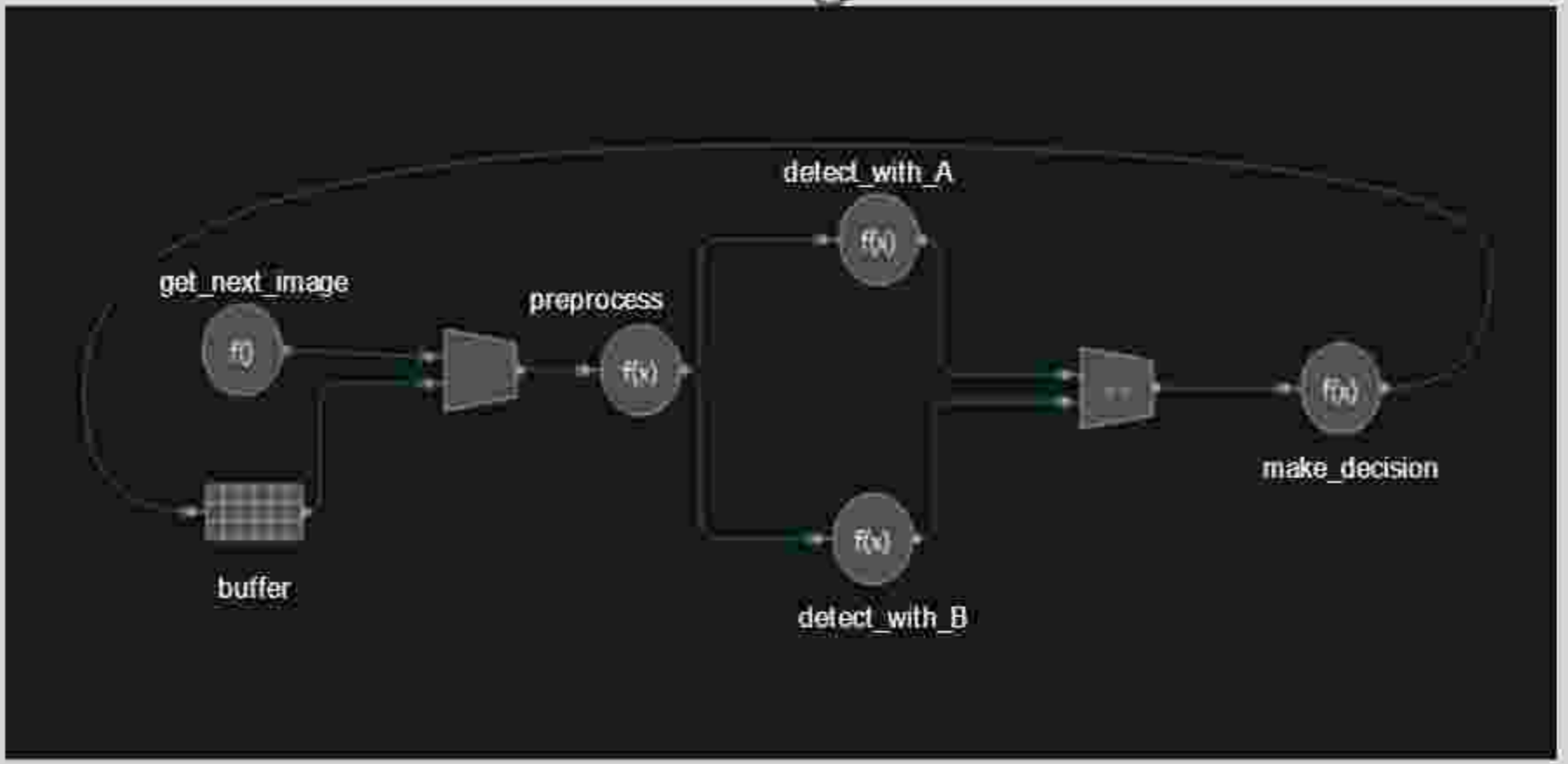
A user creates nodes and edges, interacts with the graph and waits for it to complete

```
tbb::flow::graph g;
tbb::flow::continue_node h( g,
    [](const continue_msg &) { std::cout << "Hello "; } );
tbb::flow::continue_node w( tbb::flow::follows(h),
    [](const continue_msg &) { std::cout << "World\n"; } );
```

```
h.try_put(continue_msg());
g.wait_for_all();
```

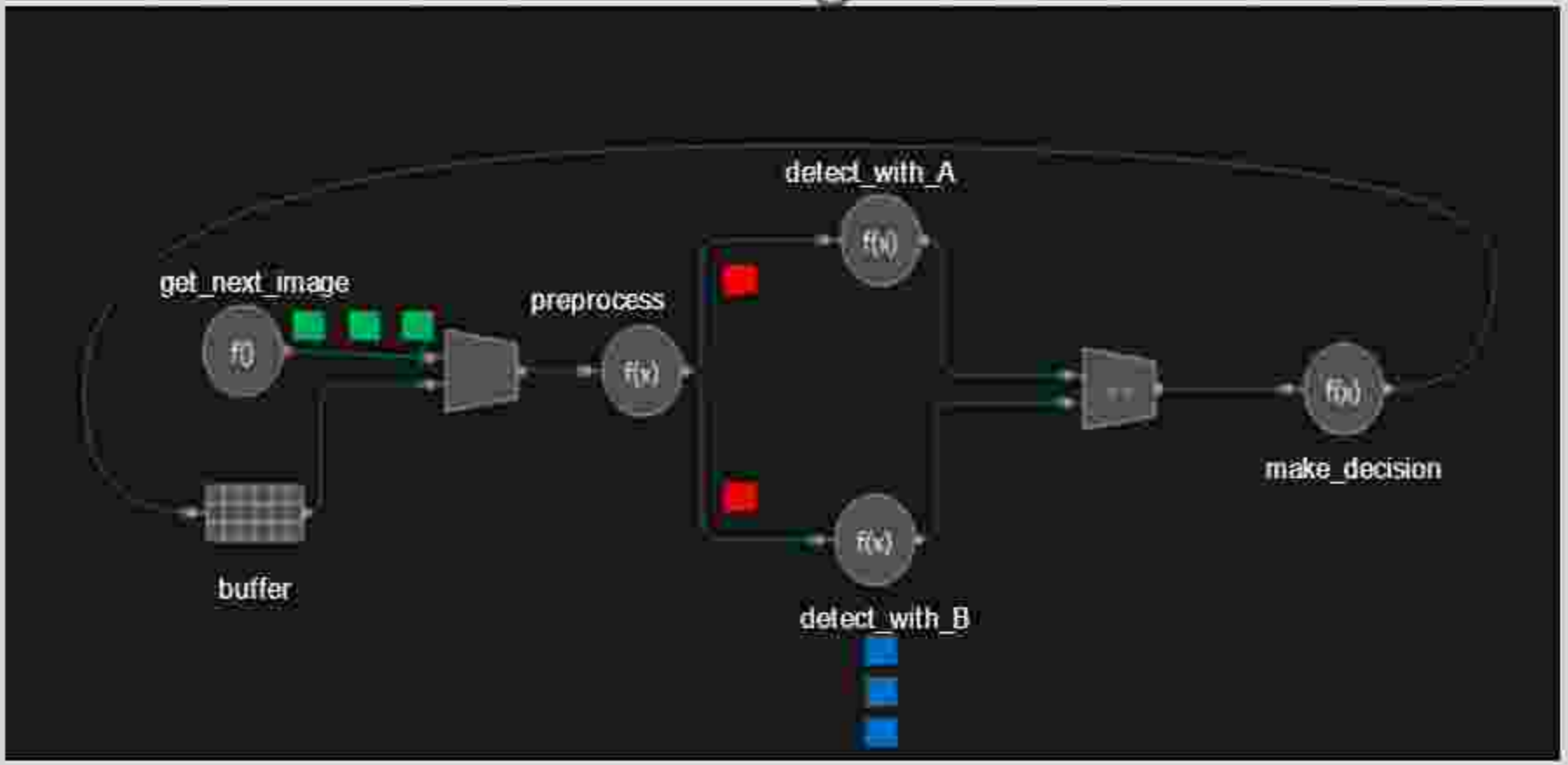


Example: Feature Detection Algorithm



Can express pipelining, task parallelism and data parallelism

Example: Feature Detection Algorithm



Can express pipelining, task parallelism and data parallelism

And supports nested parallelism with Intel® oneAPI Threading Building Blocks, OpenMP, Intel® oneAPI Math Kernel Library, etc.*

Resources

- Intel® oneAPI Base Toolkit
- As standalone component
- Open-source version

C++ Programming with oneTBB

Conclusion

- oneTBB provides flexible parallelism via C++ templates
- Parallelism is achieved by breaking work into tasks and assigning tasks to worker threads


intel