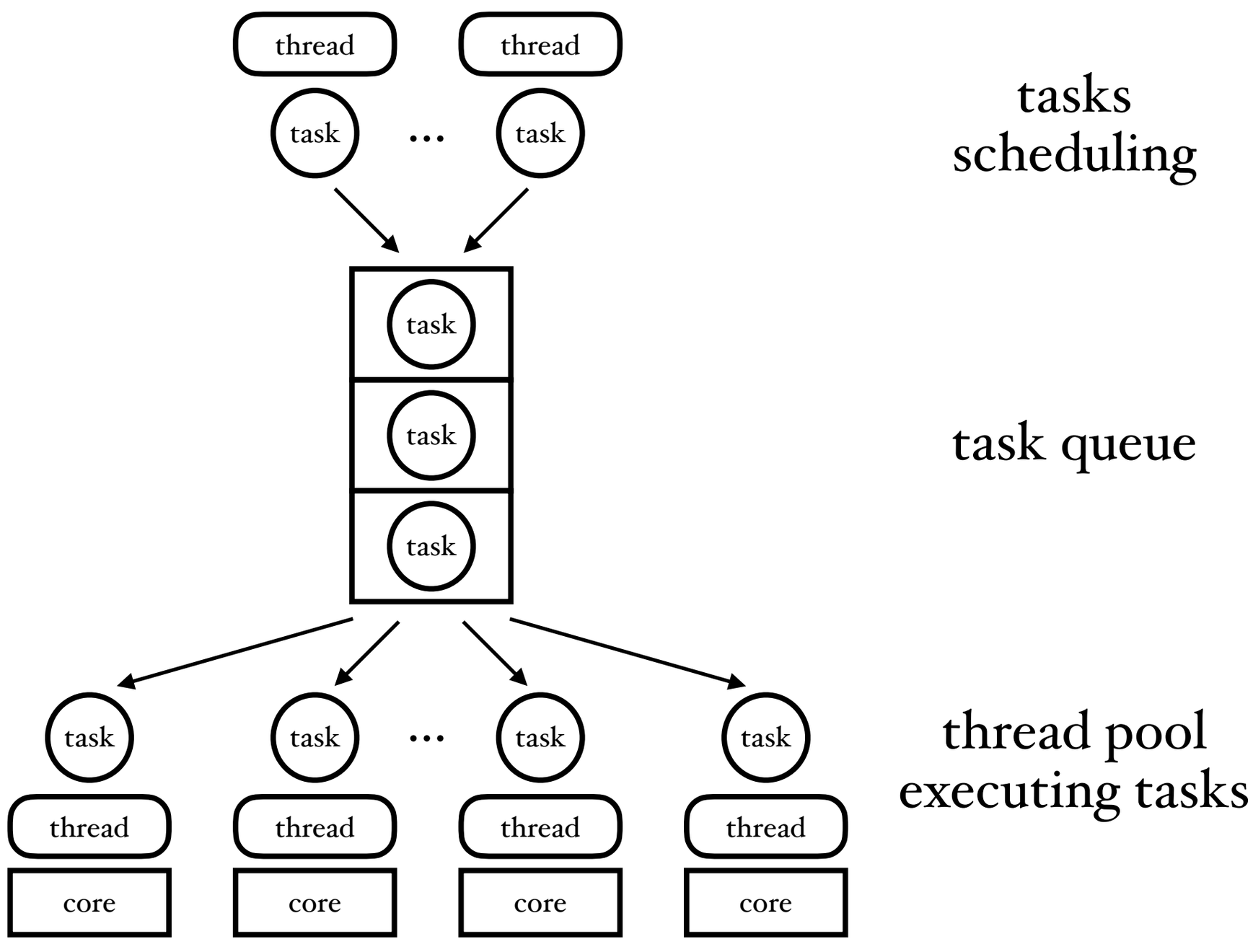
Towards better C++ compilation model

Mateusz Zych

6 July 2020

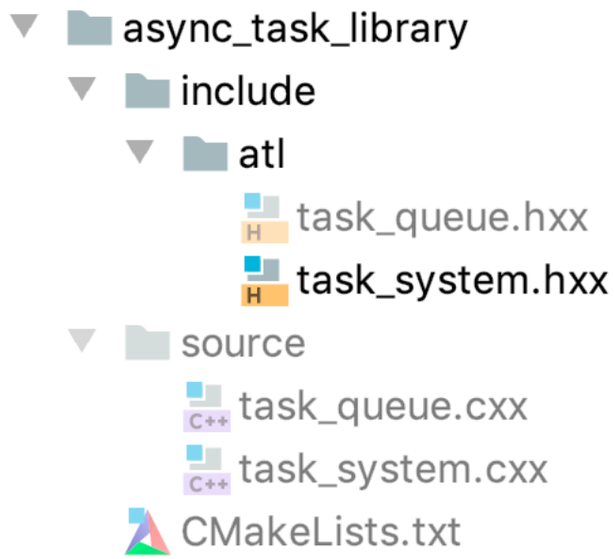
Async Task Library

Diagram



Async Task Library

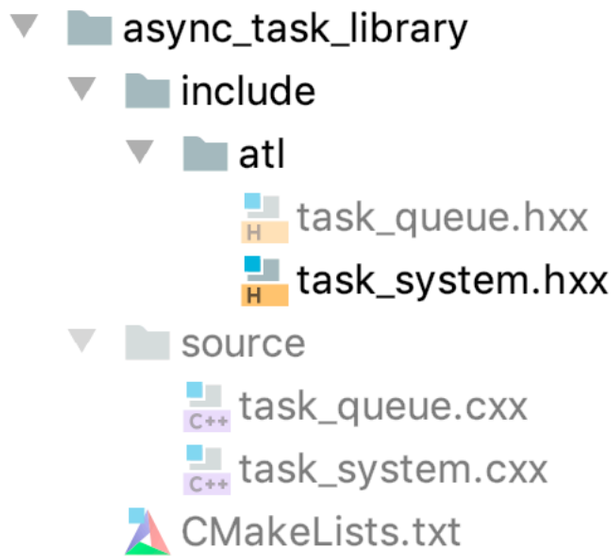
Task System - Interface




```cpp
#ifndef ATL_TASK_SYSTEM
#define ATL_TASK_SYSTEM

#include ...

namespace atl
{
    class task_system
    {
        ...

    public:
        task_system ();
       ~task_system ();

        template <typename callable>
        auto async (callable&& task) -> void { ... }
    };
}

#endif
```

Async Task Library

Task System - Interface

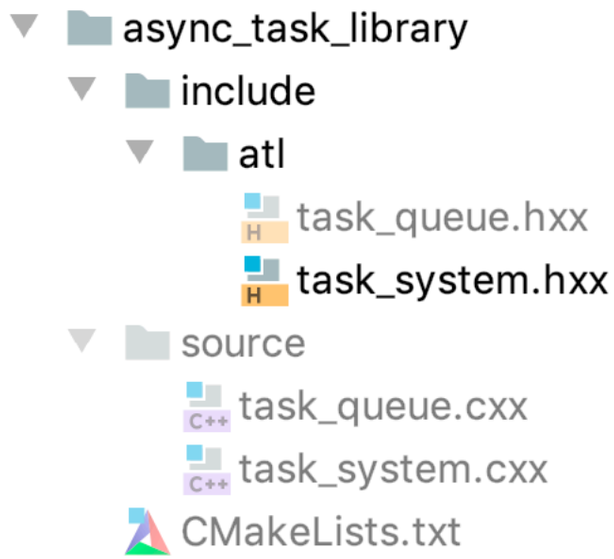



```cpp
#ifndef ATL_TASK_SYSTEM
#define ATL_TASK_SYSTEM

#include ...

namespace atl
{
    class task_system
    {
        ...

    public:
        task_system ();
       ~task_system ();

        template <typename callable>
        auto async (callable&& task) -> void { ... }
    };
}

#endif
```

task_system can be constructed and destructed

Async Task Library

Task System - Interface



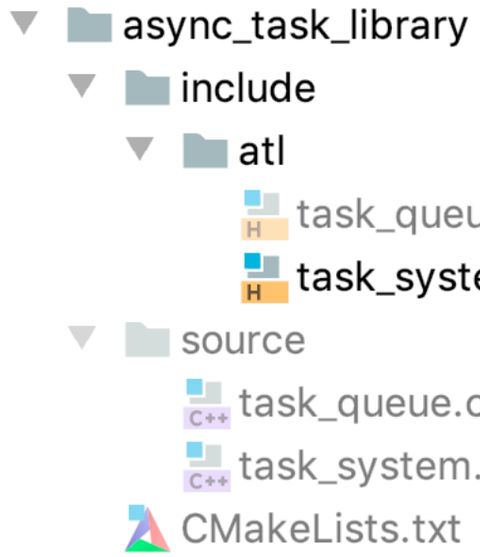

```cpp
#ifndef ATL_TASK_SYSTEM
#define ATL_TASK_SYSTEM

#include ...

namespace atl
{
    class task_system
    {
        ...

    public:
        task_system ();
       ~task_system ();

        template <typename callable>
        auto async (callable&& task) -> void { ... }
    };
}

#endif
```

task_system can be constructed and destructed

scheduling a task is done via async() member function

Async Task Library

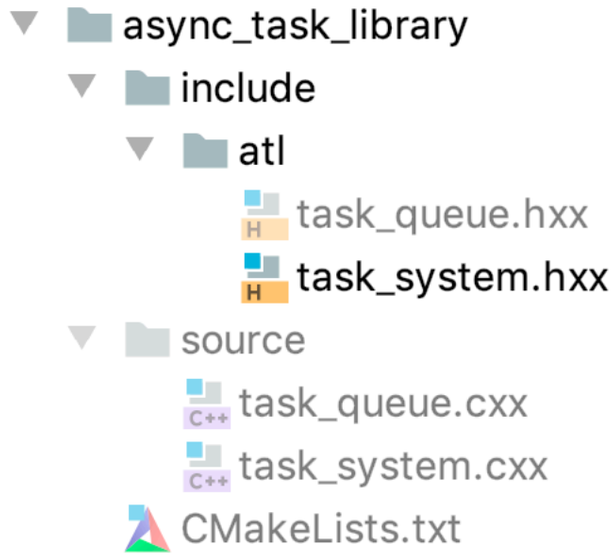
Task System - Implementation




```cpp
#ifndef ATL_TASK_SYSTEM
#define ATL_TASK_SYSTEM

#include <atl/task_queue.hxx>

#include <vector>
#include <thread>

namespace atl
{
    class task_system
    {
    private:
        std::vector<std::thread> threads;
        task_queue               queue;

        ...
    };
}

#endif
```

Async Task Library

Task System - Implementation

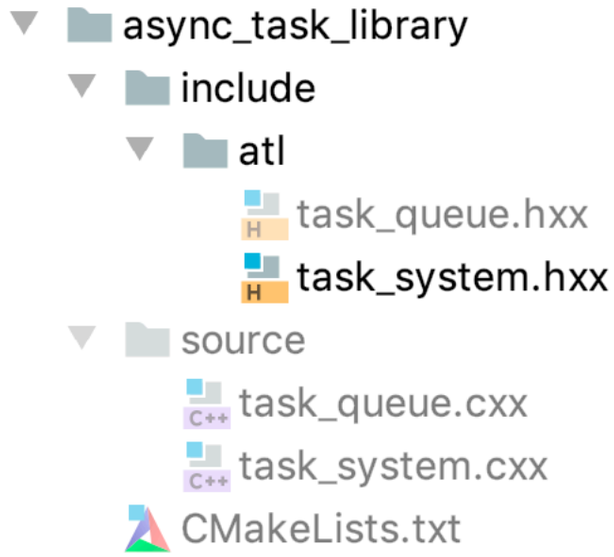



```cpp
#ifndef ATL_TASK_SYSTEM
#define ATL_TASK_SYSTEM

#include <atl/task_queue.hxx>

#include <vector>
#include <thread>

namespace atl
{
    class task_system
    {
    private:
        std::vector<std::thread> threads;
        task_queue               queue;

        ...
    };
}

#endif
```

thread pool, on which tasks will be executed

Async Task Library

Task System - Implementation



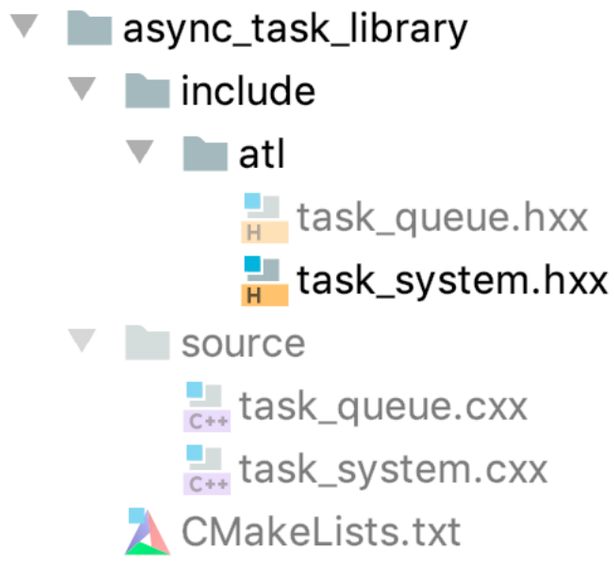

```cpp
#ifndef ATL_TASK_SYSTEM
#define ATL_TASK_SYSTEM

#include <atl/task_queue.hxx>

#include <vector>
#include <thread>

namespace atl
{
    class task_system
    {
    private:
        std::vector<std::thread> threads;
        task_queue               queue;

        ...
    };
}

#endif
```

thread pool, on which tasks will be executed

queue of tasks to be executed

Async Task Library

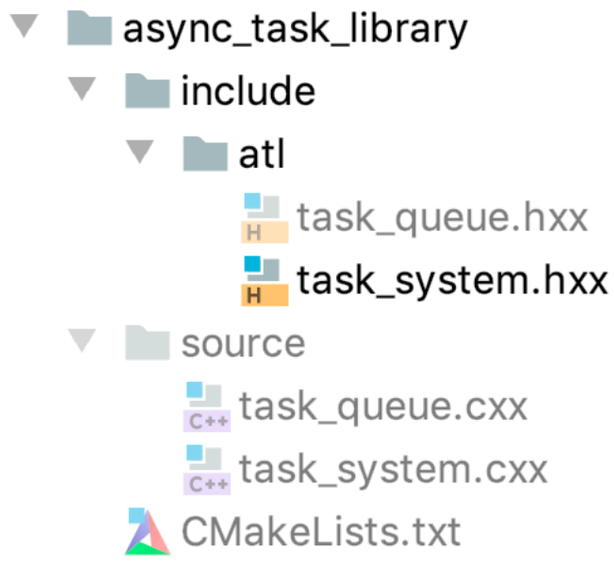
Task System - Implementation




```cpp
#ifndef ATL_TASK_SYSTEM
#define ATL_TASK_SYSTEM

#include ...

namespace atl
{
    class task_system
    {
        ...

        template <typename callable>
        auto async (callable&& task) -> void
        {
            queue.push(std::forward<callable>(task));
        }
    };
}

#endif
```

Async Task Library

Task System - Implementation

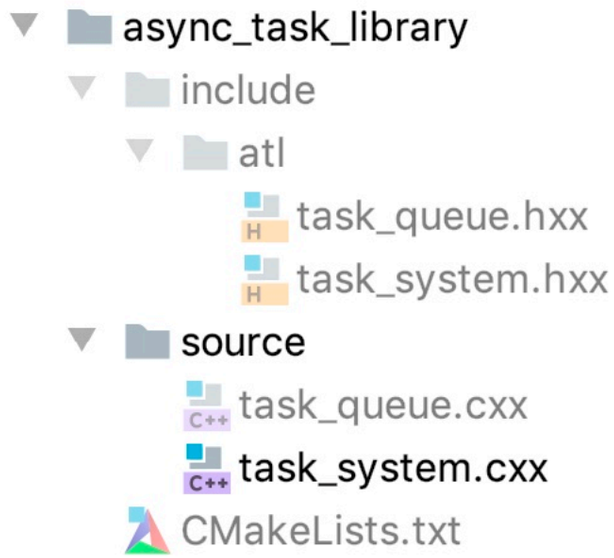



```cpp
#ifndef ATL_TASK_SYSTEM
#define ATL_TASK_SYSTEM

#include ...

namespace atl
{
    class task_system
    {
        ...

        template <typename callable>
        auto async (callable&& task) -> void
        {
            queue.push(std::forward<callable>(task));
        }
    };
}

#endif
```

scheduling a task
pushes it to the queue

Async Task Library

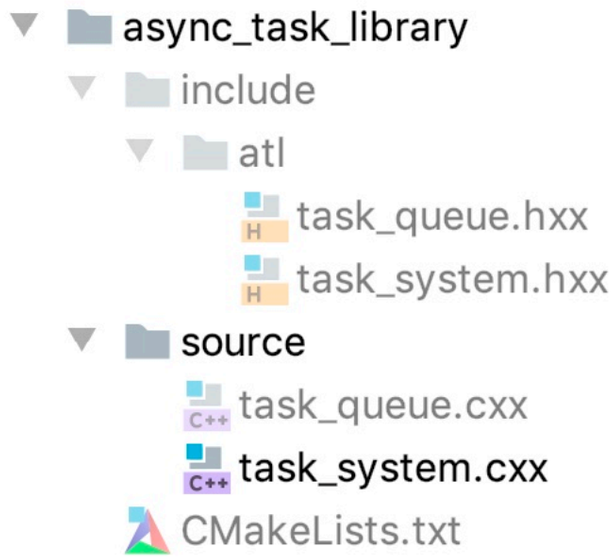
Task System - Implementation




```cpp
#include <atl/task_system.hxx>

namespace atl
{
    ...

    task_system::task_system ()
    {
        auto thread_count = std::thread::hardware_concurrency();

        for (auto i = 0uL; i != thread_count; ++i)
        {
            threads.emplace_back([this] () { run_thread(queue); });
        }
    }

    task_system::~task_system ()
    {
        queue.finish();

        for (auto& thread : threads)
        {
            thread.join();
        }
    }
}
```

Async Task Library

Task System - Implementation

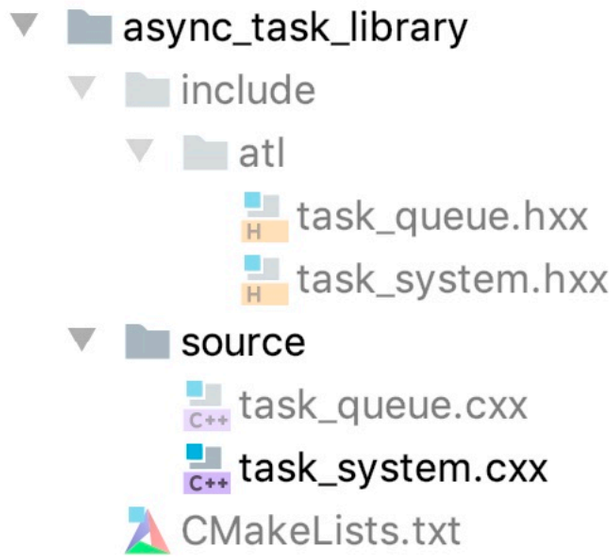



```cpp
#include <atl/task_system.hxx>

namespace atl
{
    ...

    task_system::task_system ()
    {
        auto thread_count = std::thread::hardware_concurrency();

        for (auto i = 0uL; i != thread_count; ++i)
        {
            threads.emplace_back([this] () { run_thread(queue); });
        }
    }

    task_system::~task_system ()
    {
        queue.finish();

        for (auto& thread : threads)
        {
            thread.join();
        }
    }
}
```

spawn as many threads,
as HW is capable of running in parallel

Async Task Library

Task System - Implementation



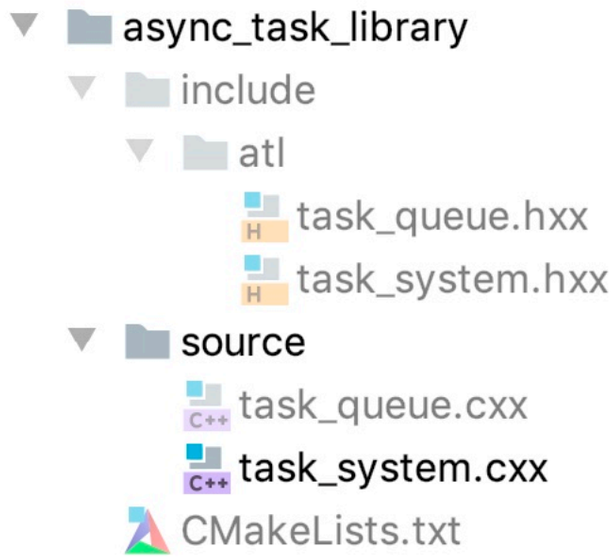

```cpp
#include <atl/task_system.hxx>

namespace atl
{
    ...

    task_system::task_system ()
    {
        auto thread_count = std::thread::hardware_concurrency();

        for (auto i = 0uL; i != thread_count; ++i)
        {
            threads.emplace_back([this] () { run_thread(queue); });
        }
    }

    task_system::~task_system ()
    {
        queue.finish();

        for (auto& thread : threads)
        {
            thread.join();
        }
    }
}
```

spawn as many threads,
as HW is capable of running in parallel

signal to the queue,
that all tasks has been scheduled

Async Task Library

Task System - Implementation




```cpp
#include <atl/task_system.hxx>

namespace atl
{
    ...

    task_system::task_system ()
    {
        auto thread_count = std::thread::hardware_concurrency();

        for (auto i = 0uL; i != thread_count; ++i)
        {
            threads.emplace_back([this] () { run_thread(queue); });
        }
    }

    task_system::~task_system ()
    {
        queue.finish();

        for (auto& thread : threads)
        {
            thread.join();
        }
    }
}
```

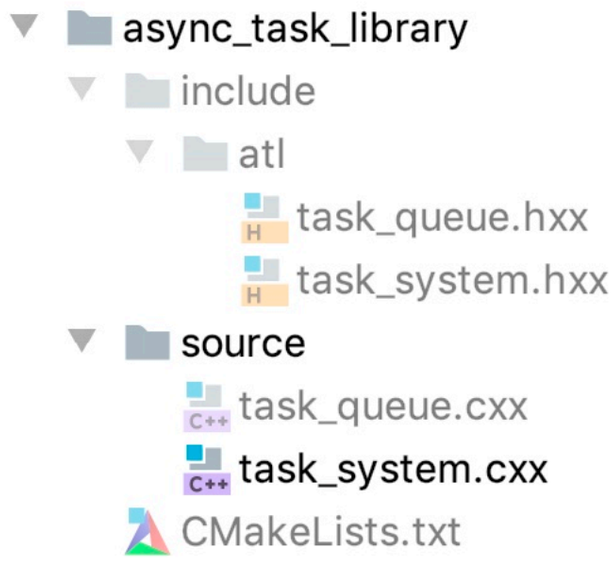
spawn as many threads,
as HW is capable of running in parallel

signal to the queue,
that all tasks has been scheduled

wait on threads,
until they finish all scheduled work

Async Task Library

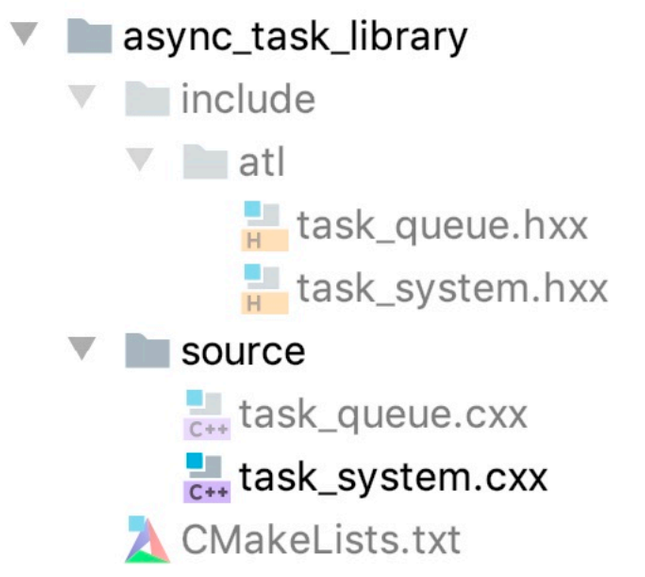
Task System - Implementation




```cpp
#include <atl/task_system.hxx>

namespace atl
{
    namespace
    {
        auto run_thread (task_queue& queue) -> void
        {
            while (true)
            {
                auto task = queue.pop();

                if (task)
                {
                    std::invoke(*task);
                }
                else
                {
                    break;
                }
            }
        }
    }

    ...
}
```

Async Task Library

Task System - Implementation

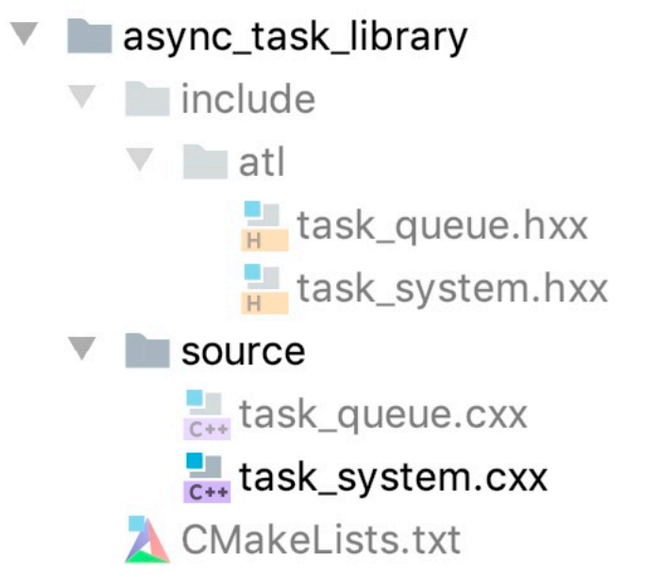



```cpp
#include <atl/task_system.hxx>

namespace atl
{
    namespace
    {
        auto run_thread (task_queue& queue) -> void
        {
            while (true)
            {
                auto task = queue.pop();

                if (task)
                {
                    std::invoke(*task);
                }
                else
                {
                    break;
                }
            }
        }
    }

    ...
}
```

threads pop() tasks from the queue and execute them

Async Task Library

Task System - Implementation



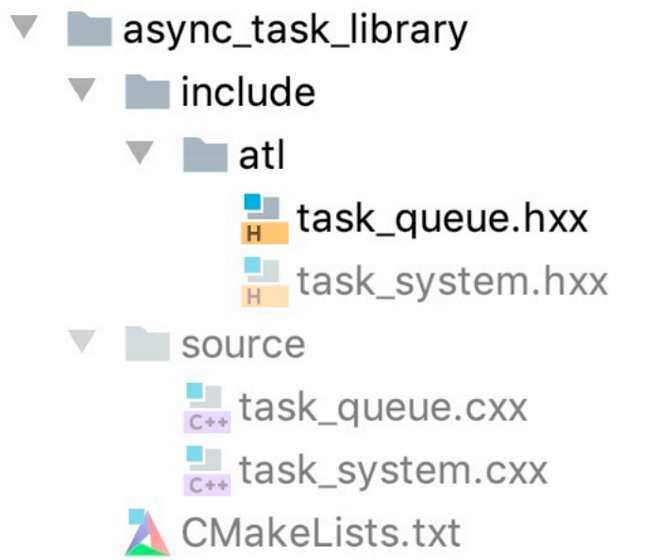

```cpp
#include <atl/task_system.hxx>

namespace atl
{
    namespace
    {
        auto run_thread (task_queue& queue) -> void
        {
            while (true)
            {
                auto task = queue.pop();

                if (task)
                {
                    std::invoke(*task);
                }
                else
                {
                    break;
                }
            }
        }
    }

    ...
}
```

threads pop() tasks from the queue and execute them

threads exit when all tasks has been executed, i.e., there are no tasks to pop()

Async Task Library

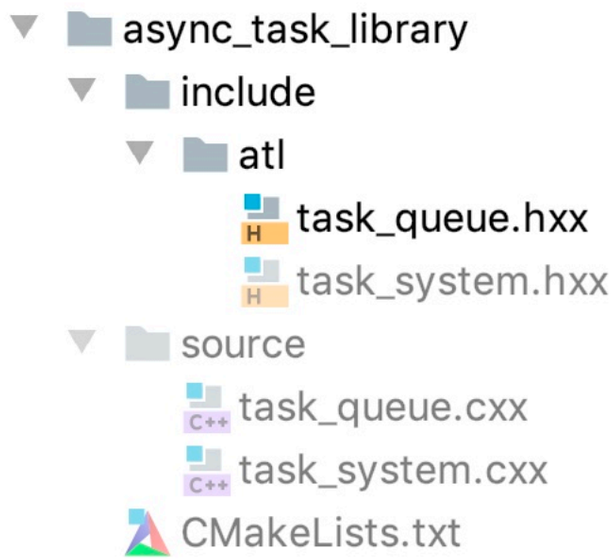
Task Queue - Interface




```cpp
#ifndef ATL_TASK_QUEUE
#define ATL_TASK_QUEUE

#include ...
#include <functional>
#include <optional>

namespace atl
{
    class task_queue
    {
        ...

    public:
        auto finish () -> void;

        auto pop () -> std::optional<std::function<void()>>;

        template <typename callable>
        auto push (callable&& task) -> void { ... }
    };
}

#endif
```

Async Task Library

Task Queue - Interface

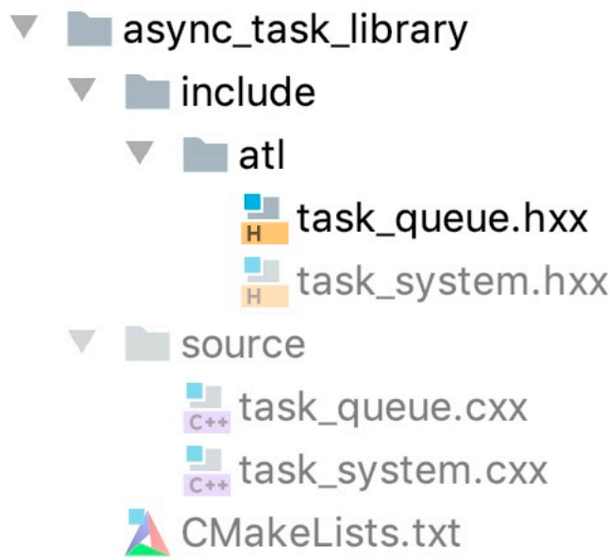



```cpp
#ifndef ATL_TASK_QUEUE
#define ATL_TASK_QUEUE

#include ...
#include <functional>
#include <optional>

namespace atl
{
    class task_queue
    {
        ...

    public:
        auto finish () -> void;

        auto pop () -> std::optional<std::function<void()>>;

        template <typename callable>
        auto push (callable&& task) -> void { ... }
    };
}

#endif
```

push() task to the queue

Async Task Library

Task Queue - Interface



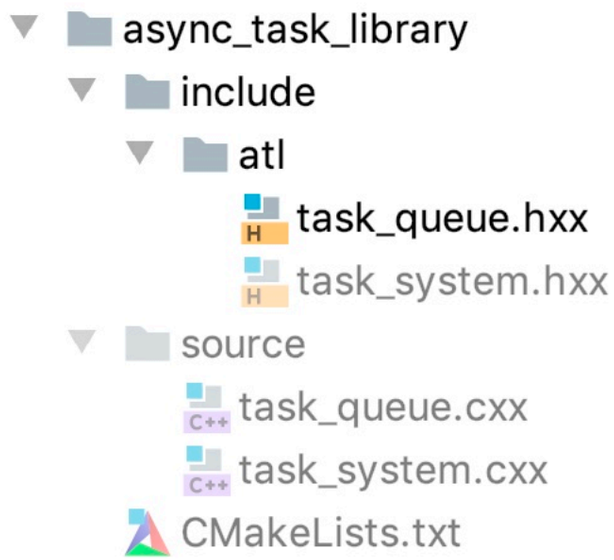

```cpp
#ifndef ATL_TASK_QUEUE
#define ATL_TASK_QUEUE

#include ...
#include <functional>
#include <optional>

namespace atl
{
    class task_queue
    {
        ...

    public:
        auto finish () -> void;

        auto pop () -> std::optional<std::function<void()>>;

        template <typename callable>
        auto push (callable&& task) -> void { ... }
    };
}

#endif
```

push() task to the queue

pop() task from the queue, if available

Async Task Library

Task Queue - Interface




```cpp
#ifndef ATL_TASK_QUEUE
#define ATL_TASK_QUEUE

#include ...
#include <functional>
#include <optional>

namespace atl
{
    class task_queue
    {
        ...

    public:
        auto finish () -> void;

        auto pop () -> std::optional<std::function<void()>>;

        template <typename callable>
        auto push (callable&& task) -> void { ... }
    };
}

#endif
```

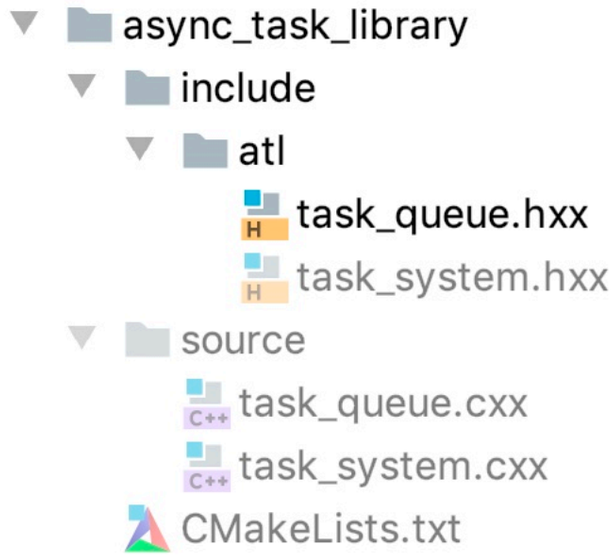
signal to the queue that no more tasks will be scheduled, i.e., push()'ed to the queue

pop() task from the queue, if available

push() task to the queue

Async Task Library

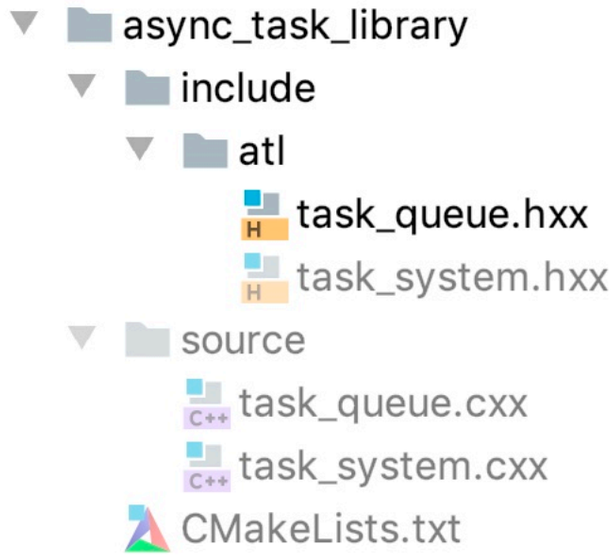
Task Queue - Implementation




```cpp
#ifndef ATL_TASK_QUEUE
#define ATL_TASK_QUEUE

#include <deque>
#include <mutex>
#include <condition_variable>
#include <functional>

namespace atl
{
    class task_queue
    {
    private:
        std::deque<std::function<void()>> tasks;
        std::mutex                        mutex;
        std::condition_variable           ready;
        bool                              done { false };

        ...
    };
}

#endif
```

Async Task Library

Task Queue - Implementation

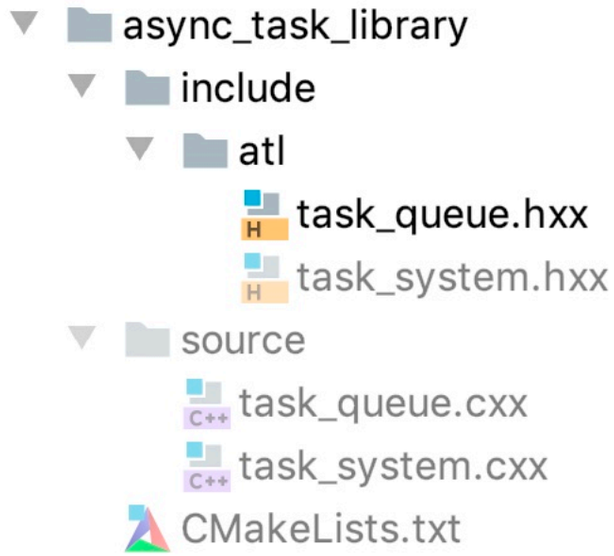



```cpp
#ifndef ATL_TASK_QUEUE
#define ATL_TASK_QUEUE

#include <deque>
#include <mutex>
#include <condition_variable>
#include <functional>

namespace atl
{
    class task_queue
    {
    private:
        std::deque<std::function<void()>> tasks;
        std::mutex                        mutex;
        std::condition_variable           ready;
        bool                              done { false };

        ...
    };
}

#endif
```

tasks to be executed

Async Task Library

Task Queue - Implementation



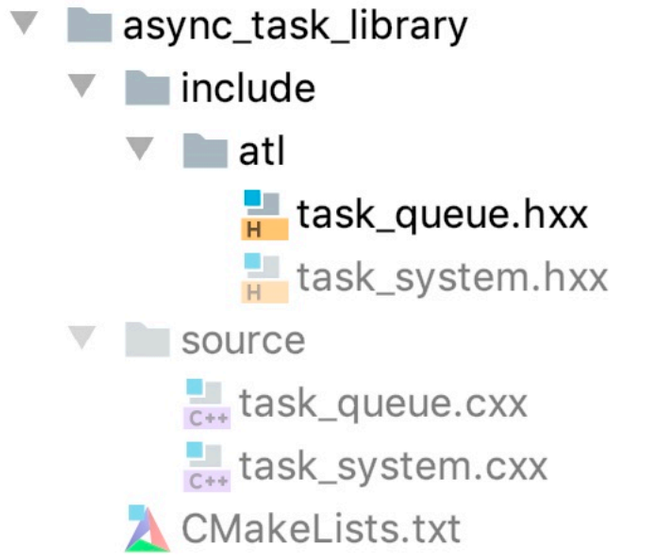

```cpp
#ifndef ATL_TASK_QUEUE
#define ATL_TASK_QUEUE

#include <deque>
#include <mutex>
#include <condition_variable>
#include <functional>

namespace atl
{
    class task_queue
    {
    private:
        std::deque<std::function<void()>> tasks;
        std::mutex                        mutex;
        std::condition_variable           ready;
        bool                              done { false };

        ...
    };
}

#endif
```

tasks to be executed

flag done defining whether
more tasks will be pushed to the queue

Async Task Library

Task Queue - Implementation

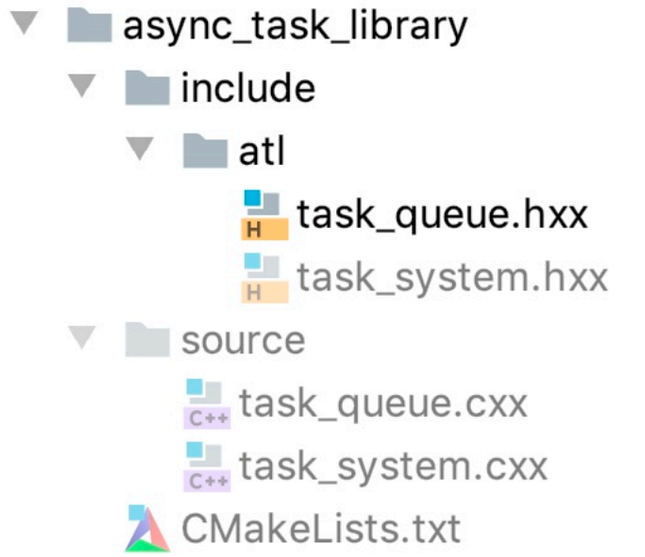



```cpp
#ifndef ATL_TASK_QUEUE
#define ATL_TASK_QUEUE

#include <deque>
#include <mutex>
#include <condition_variable>
#include <functional>

namespace atl
{
    class task_queue
    {
    private:
        std::deque<std::function<void()>> tasks;
        std::mutex                        mutex;
        std::condition_variable           ready;
        bool                              done { false };

        ...
    };
}

#endif
```

mutex protecting shared state, i.e., task queue & done flag

Async Task Library

Task Queue - Implementation



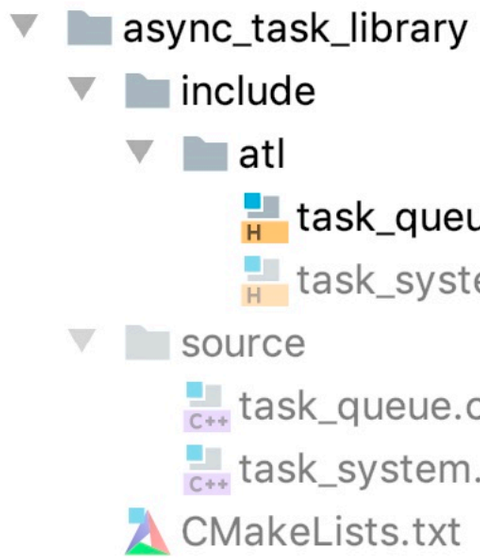

```cpp
#ifndef ATL_TASK_QUEUE
#define ATL_TASK_QUEUE

#include <deque>
#include <mutex>
#include <condition_variable>
#include <functional>

namespace atl
{
    class task_queue
    {
    private:
        std::deque<std::function<void()>> tasks;
        std::mutex                        mutex;
        std::condition_variable           ready;
        bool                              done { false };

        ...
    };
}

#endif
```

mutex protecting shared state, i.e., task queue & done flag

condition variable specifying whether a task is ready to be pop()ed from the queue

Async Task Library

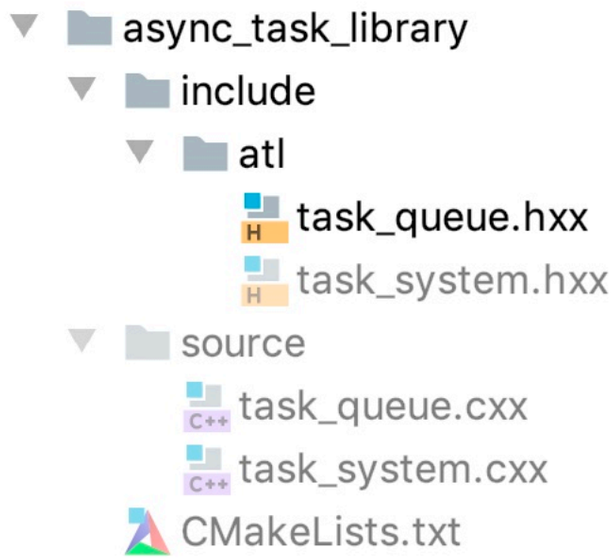
Task Queue - Implementation




```cpp
#ifndef ATL_TASK_QUEUE
#define ATL_TASK_QUEUE

#include ...
#include <utility>

namespace atl
{
    class task_queue
    {
        ...

        template <typename callable>
        auto push (callable&& task) -> void
        {
            {
                auto lock = std::lock_guard<std::mutex> { mutex };
                tasks.emplace_back(std::forward<callable>(task));
            }
            ready.notify_one();
        }
    };
}

#endif
```

Async Task Library

Task Queue - Implementation

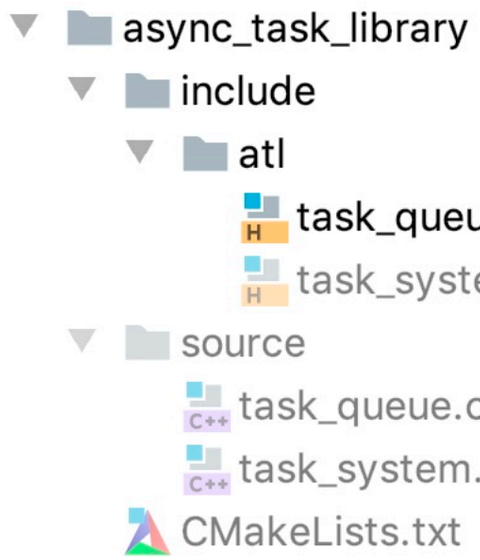



```cpp
#ifndef ATL_TASK_QUEUE
#define ATL_TASK_QUEUE

#include ...
#include <utility>

namespace atl
{
    class task_queue
    {
        ...

        template <typename callable>
        auto push (callable&& task) -> void
        {
            {
                auto lock = std::lock_guard<std::mutex> { mutex };
                tasks.emplace_back(std::forward<callable>(task));
            }
            ready.notify_one();
        }
    };
}

#endif
```

enter critical section

Async Task Library

Task Queue - Implementation



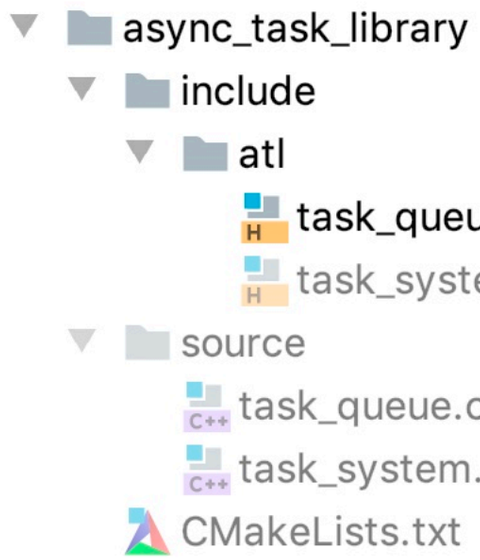

```cpp
#ifndef ATL_TASK_QUEUE
#define ATL_TASK_QUEUE

#include ...
#include <utility>

namespace atl
{
    class task_queue
    {
        ...

        template <typename callable>
        auto push (callable&& task) -> void
        {
            {
                auto lock = std::lock_guard<std::mutex> { mutex };
                tasks.emplace_back(std::forward<callable>(task));
            }
            ready.notify_one();
        }
    };
}

#endif
```

enter critical section

put task at the end of the queue

Async Task Library

Task Queue - Implementation




```cpp
#ifndef ATL_TASK_QUEUE
#define ATL_TASK_QUEUE

#include ...
#include <utility>

namespace atl
{
    class task_queue
    {
        ...

        template <typename callable>
        auto push (callable&& task) -> void
        {
            {
                auto lock = std::lock_guard<std::mutex> { mutex };
                tasks.emplace_back(std::forward<callable>(task));
            }
            ready.notify_one();
        }
    };
}

#endif
```

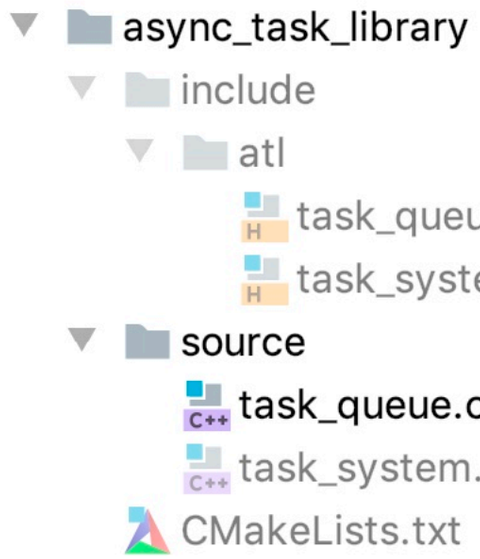
enter critical section

put task at the end of the queue

notify one thread waiting inside pop() that a task is ready to be executed

Async Task Library

Task Queue - Implementation

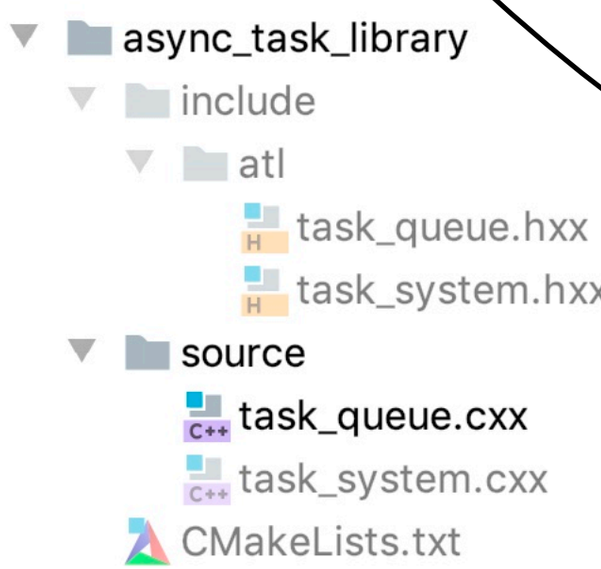



```cpp
#include <atl/task_queue.hxx>

namespace atl
{
    ...

    auto task_queue::pop () -> std::optional<std::function<void()>>
    {
        auto lock = std::unique_lock<std::mutex> { mutex };

        ready.wait(lock, [&] () { return !tasks.empty() || done; });

        if (!tasks.empty())
        {
            auto task = std::move(tasks.front());

            tasks.pop_front();

            return task;
        }
        else
        {
            return std::nullopt;
        }
    }
}
```

Async Task Library

Task Queue - Implementation

enter critical section

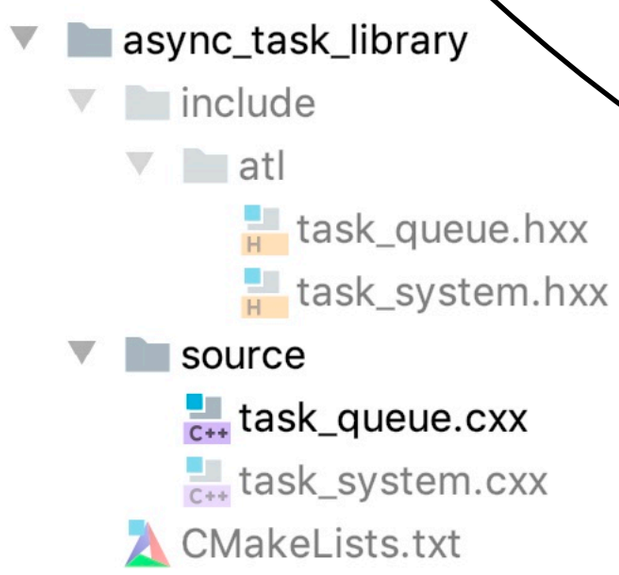



```cpp
#include <atl/task_queue.hxx>

namespace atl
{
    ...

    auto task_queue::pop () -> std::optional<std::function<void()>>
    {
        auto lock = std::unique_lock<std::mutex> { mutex };

        ready.wait(lock, [&] () { return !tasks.empty() || done; });

        if (!tasks.empty())
        {
            auto task = std::move(tasks.front());

            tasks.pop_front();

            return task;
        }
        else
        {
            return std::nullopt;
        }
    }
}
```

Async Task Library

Task Queue - Implementation

enter critical section

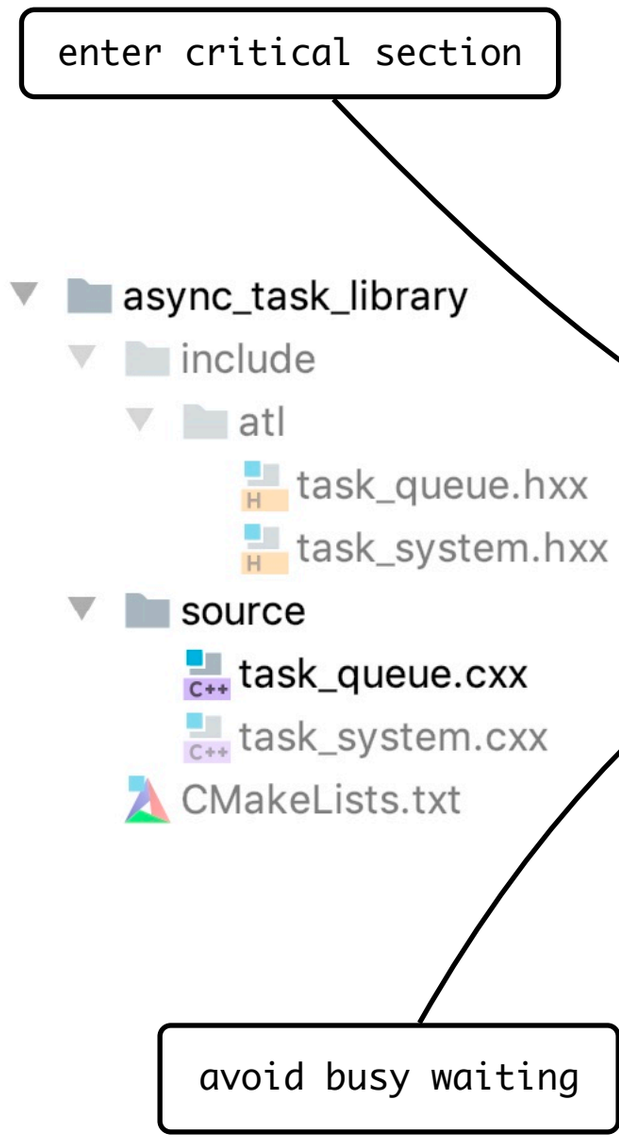



```cpp
#include <atl/task_queue.hxx>

namespace atl
{
    ...

    auto task_queue::pop () -> std::optional<std::function<void()>>
    {
        auto lock = std::unique_lock<std::mutex> { mutex };

        ready.wait(lock, [&] () { return !tasks.empty() || done; });

        if (!tasks.empty())
        {
            auto task = std::move(tasks.front());

            tasks.pop_front();

            return task;
        }
        else
        {
            return std::nullopt;
        }
    }
}
```

wake up if
there is a task to pop() or
scheduling of tasks has finished

Async Task Library

Task Queue - Implementation



enter critical section

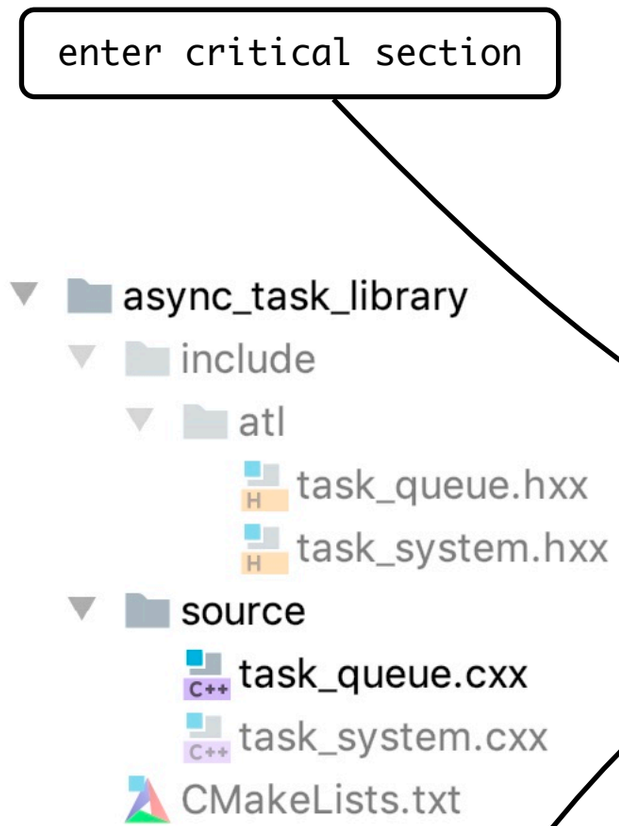
avoid busy waiting

wake up if
there is a task to pop() or
scheduling of tasks has finished

```cpp
#include <atl/task_queue.hxx>

namespace atl
{
    ...

    auto task_queue::pop () -> std::optional<std::function<void()>>
    {
        auto lock = std::unique_lock<std::mutex> { mutex };

        ready.wait(lock, [&] () { return !tasks.empty() || done; });

        if (!tasks.empty())
        {
            auto task = std::move(tasks.front());

            tasks.pop_front();

            return task;
        }
        else
        {
            return std::nullopt;
        }
    }
}
```

Async Task Library

Task Queue - Implementation



enter critical section

avoid busy waiting

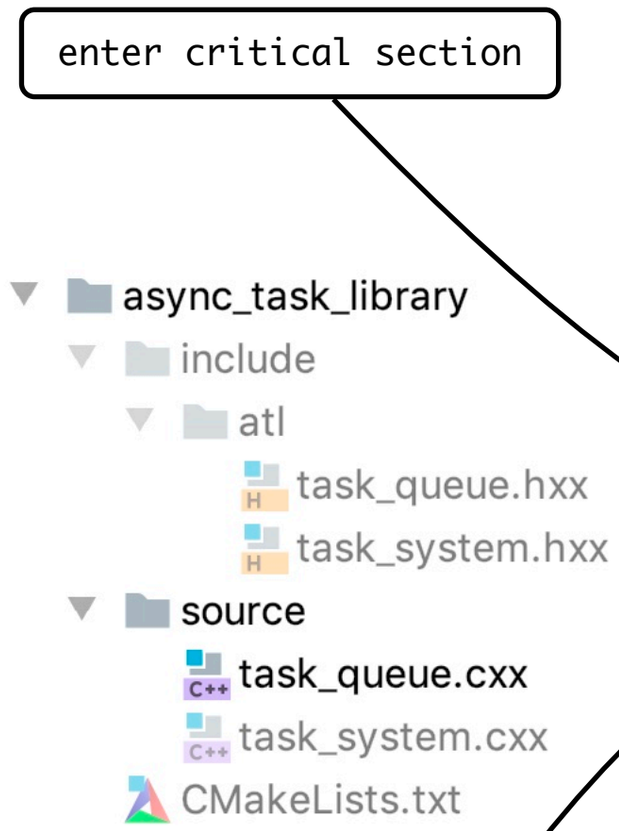
wake up if
there is a task to pop() or
scheduling of tasks has finished

take task from the beginning of the queue

```cpp
#include <atl/task_queue.hxx>

namespace atl
{
    ...

    auto task_queue::pop () -> std::optional<std::function<void()>>
    {
        auto lock = std::unique_lock<std::mutex> { mutex };

        ready.wait(lock, [&] () { return !tasks.empty() || done; });

        if (!tasks.empty())
        {
            auto task = std::move(tasks.front());

            tasks.pop_front();

            return task;
        }
        else
        {
            return std::nullopt;
        }
    }
}
```

Async Task Library

Task Queue - Implementation



enter critical section

avoid busy waiting

wake up if
there is a task to pop() or
scheduling of tasks has finished

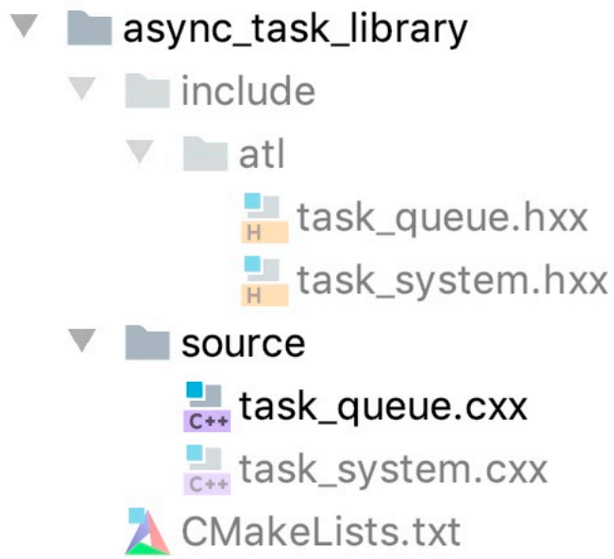
take task from the beginning of the queue

return nothing,
since no tasks are the queue and
no more tasks will be pushed to the queue

```cpp
#include <atl/task_queue.hxx>

namespace atl
{
    ...

    auto task_queue::pop () -> std::optional<std::function<void()>>
    {
        auto lock = std::unique_lock<std::mutex> { mutex };

        ready.wait(lock, [&] () { return !tasks.empty() || done; });

        if (!tasks.empty())
        {
            auto task = std::move(tasks.front());

            tasks.pop_front();

            return task;
        }
        else
        {
            return std::nullopt;
        }
    }
}
```

Async Task Library

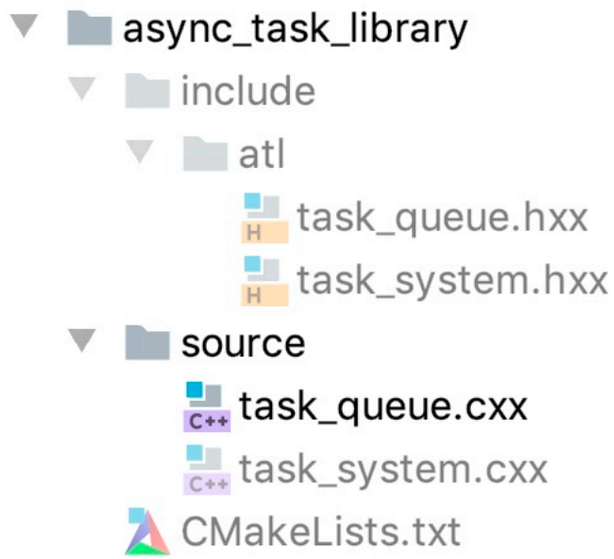
Task Queue - Implementation




```cpp
#include <atl/task_queue.hxx>

namespace atl
{
    auto task_queue::finish() -> void
    {
        {
            auto lock = std::lock_guard<std::mutex> { mutex };
            done = true;
        }
        ready.notify_all();
    }

    ...
}
```

Async Task Library

Task Queue - Implementation

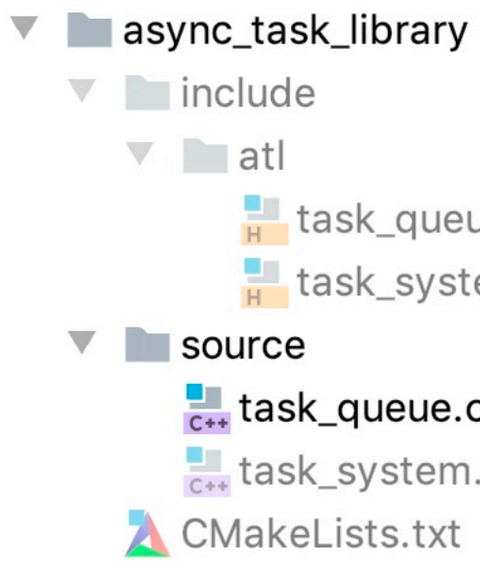



```cpp
#include <atl/task_queue.hxx>

namespace atl
{
    auto task_queue::finish() -> void
    {
        {
            auto lock = std::lock_guard<std::mutex> { mutex };
            done = true;
        }
        ready.notify_all();
    }

    ...
}
```

enter critical section

Async Task Library

Task Queue - Implementation



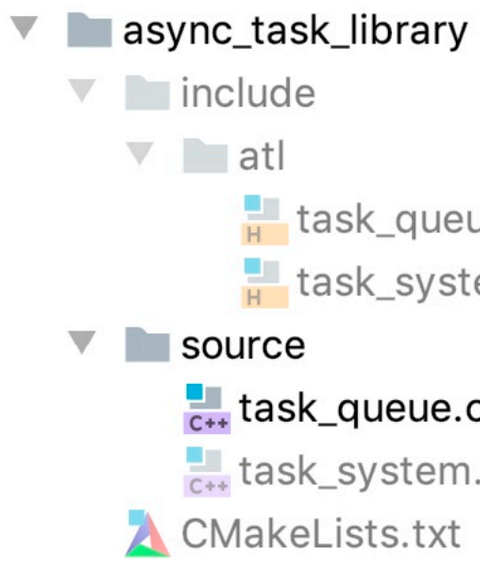

```cpp
#include <atl/task_queue.hxx>

namespace atl
{
    auto task_queue::finish() -> void
    {
        {
            auto lock = std::lock_guard<std::mutex> { mutex };
            done = true;
        }
        ready.notify_all();
    }

    ...
}
```

enter critical section

set done flag

Async Task Library

Task Queue - Implementation




```cpp
#include <atl/task_queue.hxx>

namespace atl
{
    auto task_queue::finish() -> void
    {
        {
            auto lock = std::lock_guard<std::mutex> { mutex };
            done = true;
        }
        ready.notify_all();
    }

    ...
}
```

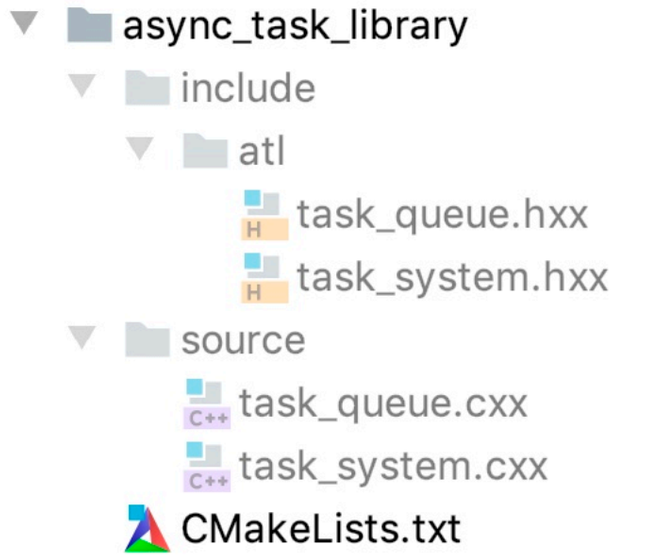
enter critical section

set done flag

notify all threads waiting inside pop(),
that no more tasks will be push()ed to the queue

Async Task Library

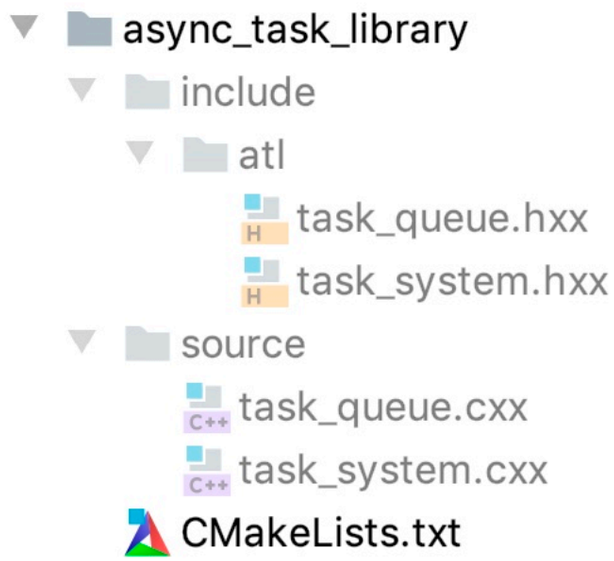
CMake build system




```
cmake_minimum_required (VERSION 3.15)

project(AsyncTaskLibrary VERSION 1.0.0 LANGUAGES CXX)

find_package(Threads REQUIRED)


add_library                (async-task-lib STATIC)

target_compile_features    (async-task-lib PUBLIC  cxx_std_17)

target_include_directories(async-task-lib PUBLIC  include)

target_link_libraries      (async-task-lib PUBLIC
                                           ${CMAKE_THREAD_LIBS_INIT})

target_sources             (async-task-lib PRIVATE
                                       include/atl/task_queue.hxx
                                       include/atl/task_system.hxx
                                            source/task_queue.cxx
                                            source/task_system.cxx)
```

Async Task Library

CMake build system

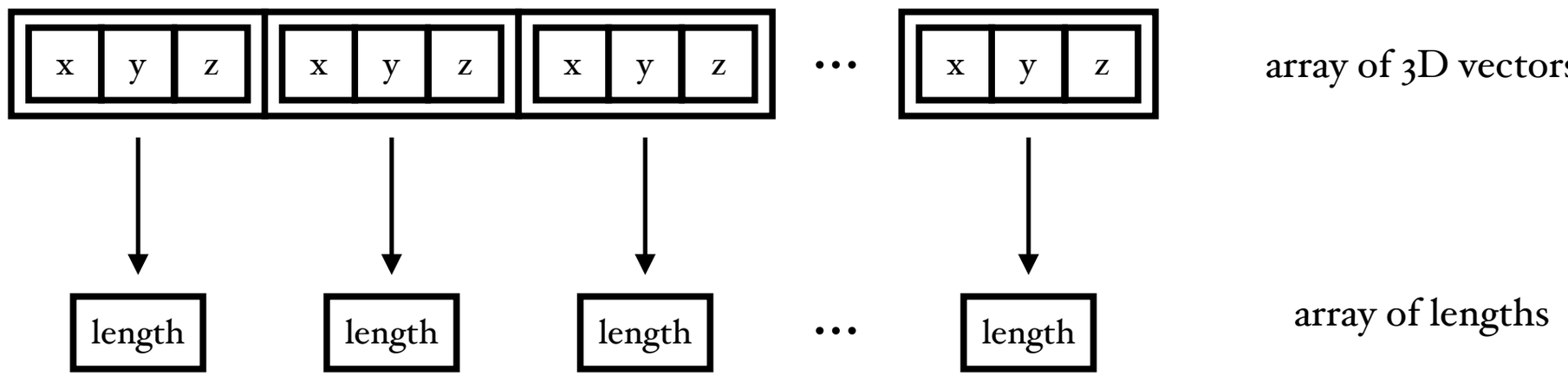



```
cmake_minimum_required (VERSION 3.15)

project(AsyncTaskLibrary VERSION 1.0.0 LANGUAGES CXX)

find_package(Threads REQUIRED)


add_library                (async-task-lib STATIC)

target_compile_features    (async-task-lib PUBLIC  cxx_std_17)

target_include_directories(async-task-lib PUBLIC  include)

target_link_libraries      (async-task-lib PUBLIC
                                           ${CMAKE_THREAD_LIBS_INIT})

target_sources             (async-task-lib PRIVATE
                                       include/atl/task_queue.hxx
                                       include/atl/task_system.hxx
                                            source/task_queue.cxx
                                            source/task_system.cxx)
```

allow clients to #include header files from the include directory

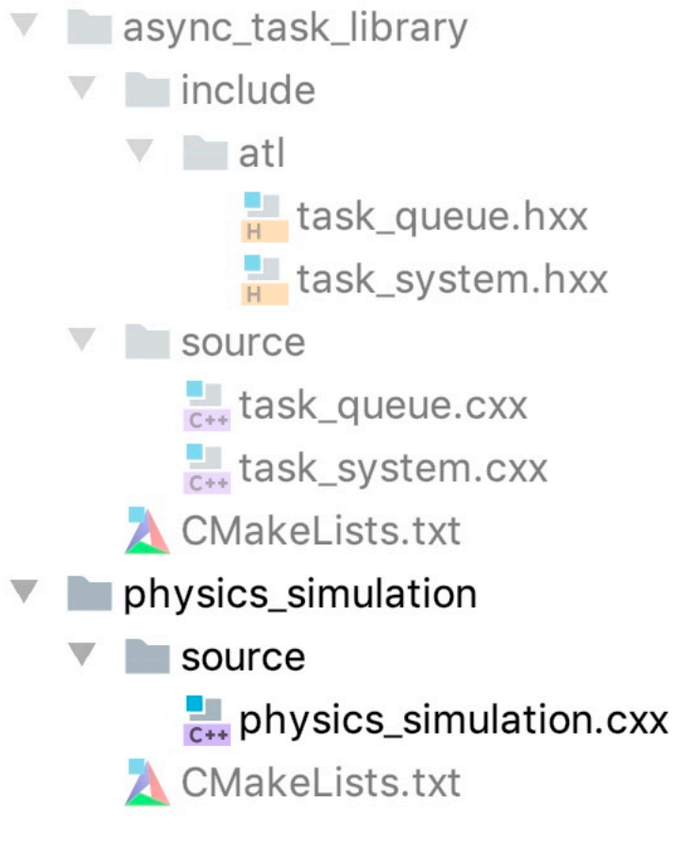
Physics Simulation

Diagram



Physics Simulation

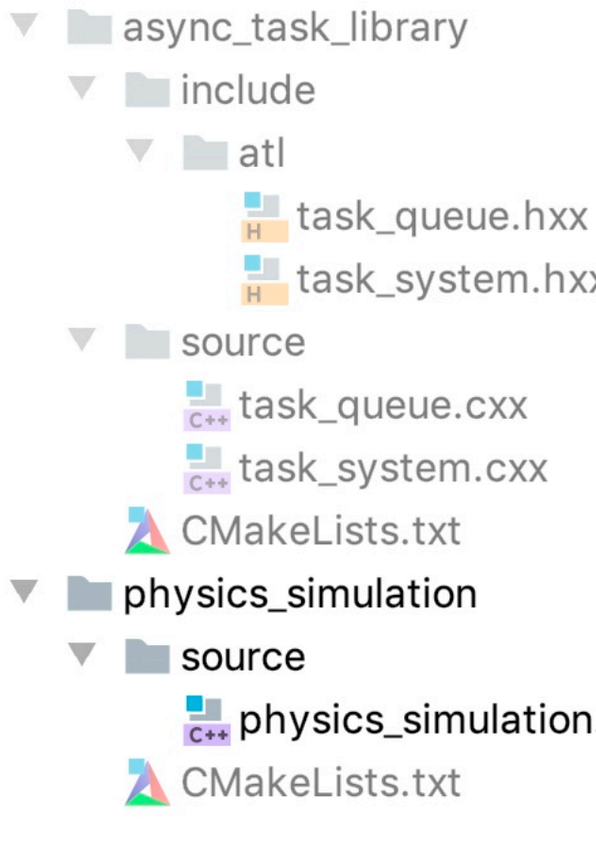
Data Flow




```cpp
#include ...

namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto read_input () -> array<vec3>
    { ... }
    auto compute_lengths (const array<vec3>& vectors) -> array<float>
    { ... }
    auto print_results (const array<float>& lengths) -> void
    { ... }
}

auto main () -> int
{
    const auto vectors = phy_sim::read_input();

    const auto lengths = phy_sim::compute_lengths(vectors);

    phy_sim::print_results(lengths);

    return 0;
}
```

Physics Simulation

Data Flow

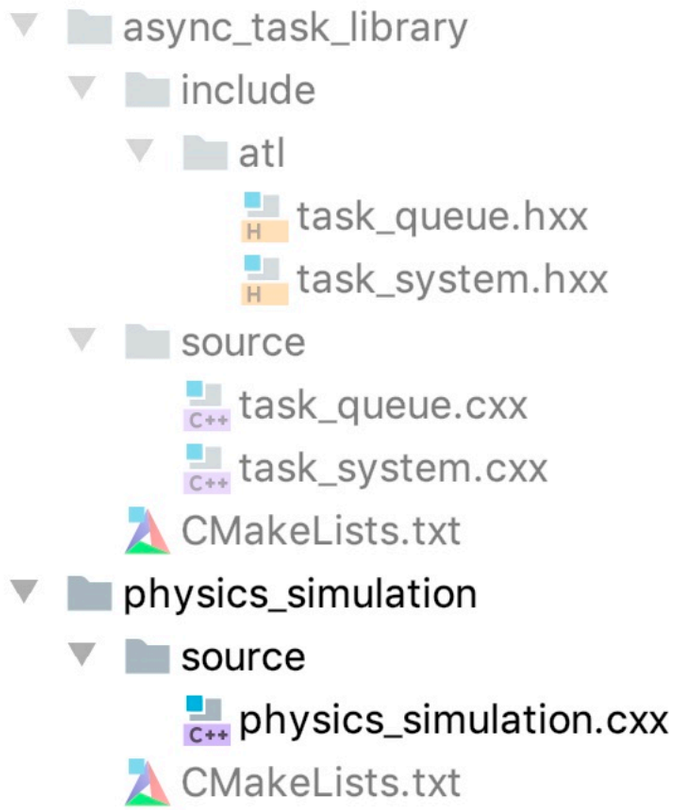



```cpp
#include ...

namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto read_input () -> array<vec3>
    { ... }
    auto compute_lengths (const array<vec3>& vectors) -> array<float>
    { ... }
    auto print_results (const array<float>& lengths) -> void
    { ... }
}

auto main () -> int
{
    const auto vectors = phy_sim::read_input();

    const auto lengths = phy_sim::compute_lengths(vectors);

    phy_sim::print_results(lengths);

    return 0;
}
```

3D vector

Physics Simulation

Data Flow



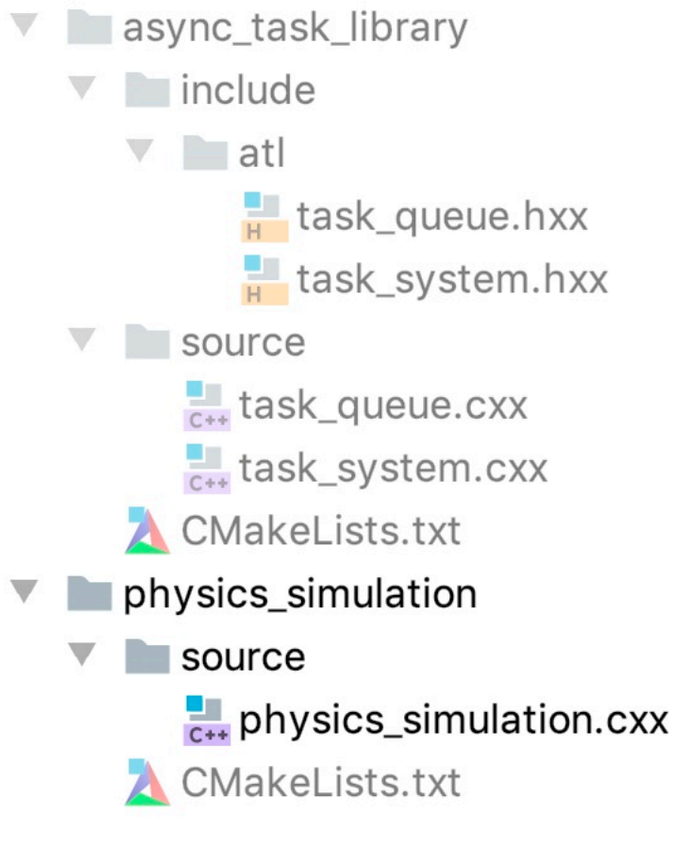

```cpp
#include ...

namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto read_input () -> array<vec3>
    { ... }
    auto compute_lengths (const array<vec3>& vectors) -> array<float>
    { ... }
    auto print_results (const array<float>& lengths) -> void
    { ... }
}

auto main () -> int
{
    const auto vectors = phy_sim::read_input();

    const auto lengths = phy_sim::compute_lengths(vectors);

    phy_sim::print_results(lengths);

    return 0;
}
```

3D vector

obtain input array of 3D vectors

Physics Simulation

Data Flow




```cpp
#include ...

namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto read_input () -> array<vec3>
    { ... }
    auto compute_lengths (const array<vec3>& vectors) -> array<float>
    { ... }
    auto print_results (const array<float>& lengths) -> void
    { ... }
}

auto main () -> int
{
    const auto vectors = phy_sim::read_input();

    const auto lengths = phy_sim::compute_lengths(vectors);

    phy_sim::print_results(lengths);

    return 0;
}
```

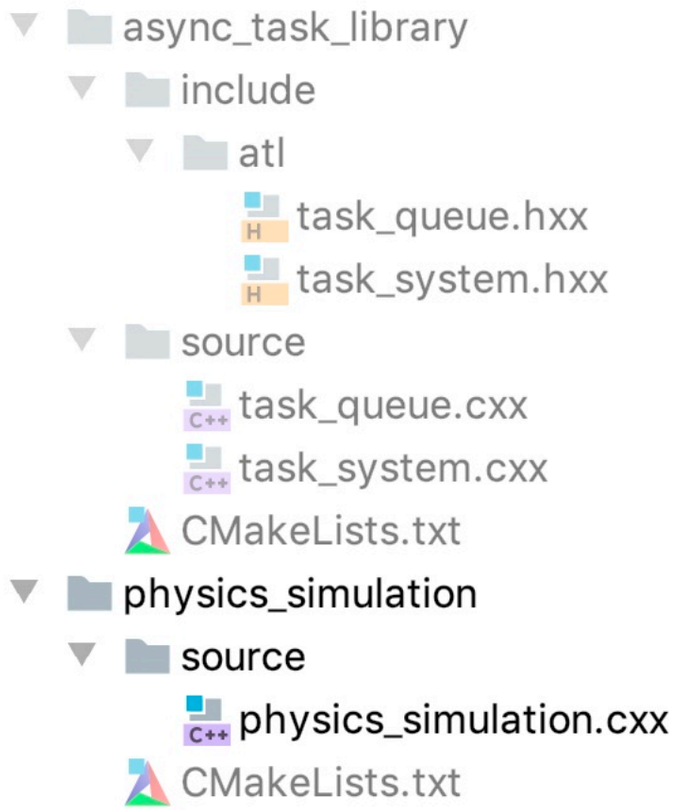
3D vector

obtain input array of 3D vectors

compute lengths of 3D vectors

Physics Simulation

Data Flow




```cpp
#include ...

namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto read_input () -> array<vec3>
    { ... }
    auto compute_lengths (const array<vec3>& vectors) -> array<float>
    { ... }
    auto print_results (const array<float>& lengths) -> void
    { ... }
}

auto main () -> int
{
    const auto vectors = phy_sim::read_input();

    const auto lengths = phy_sim::compute_lengths(vectors);

    phy_sim::print_results(lengths);

    return 0;
}
```

3D vector

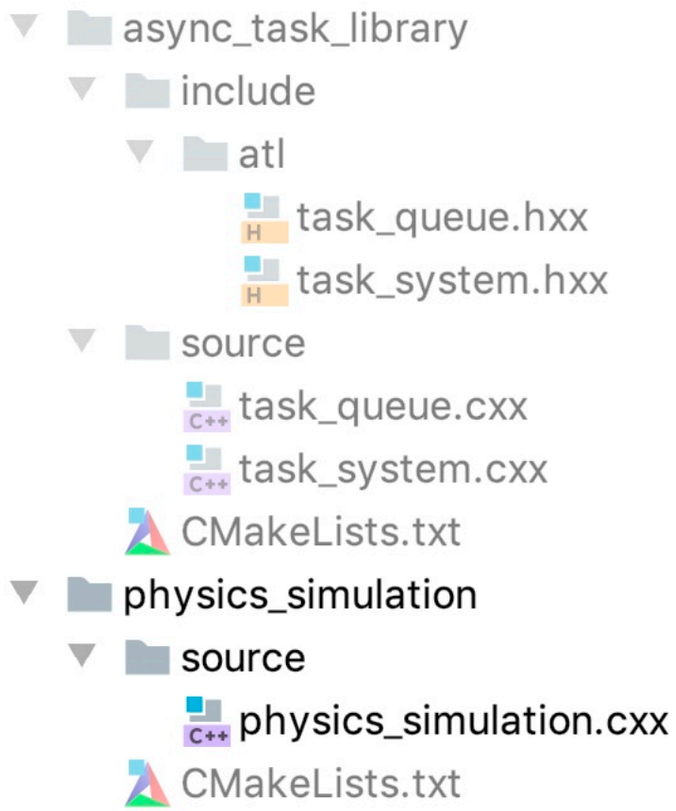
obtain input array of 3D vectors

compute lengths of 3D vectors

report calculated lengths

Physics Simulation

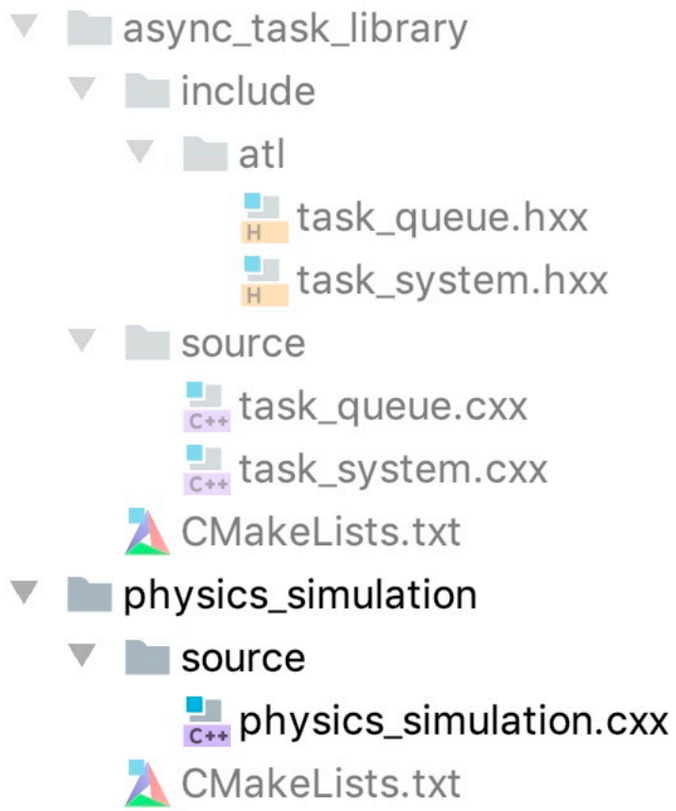
Length of a 3D Vector




```cpp
#include <atl/task_system.hxx>
#include <cmath>

namespace phy_sim
{
    ...

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x + v.y * v.y + v.z * v.z);
    }

    ...
}
```

Physics Simulation

Length of a 3D Vector



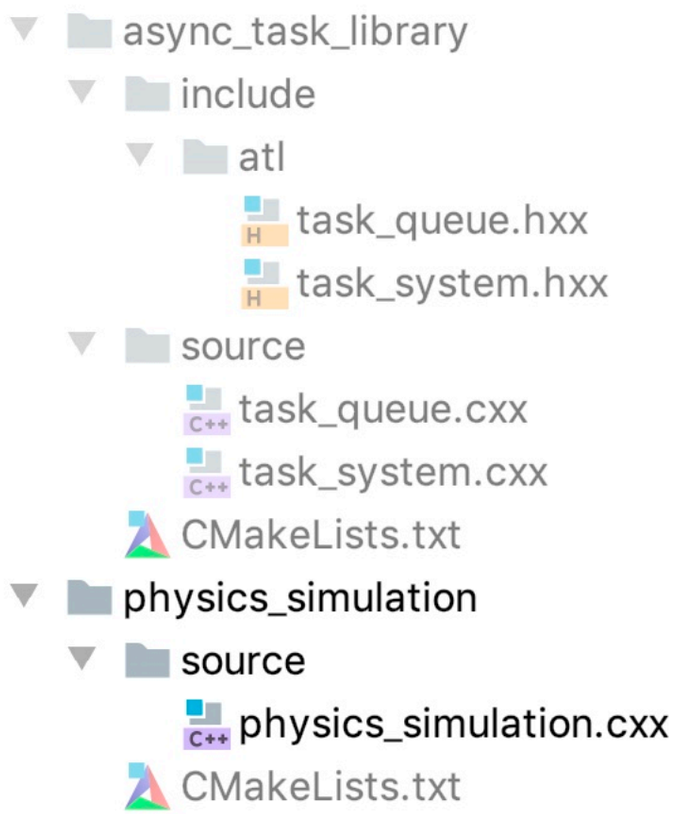

```
#include <atl/task_system.hxx>
#include <cmath>

namespace phy_sim
{
    ...
    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x + v.y * v.y + v.z * v.z);
    }
    ...
}
```

calculate length of a 3D vector
using Euclidean metric:

$$|v| = \sqrt{v_x^2 + v_y^2 + v_z^2}$$

Physics Simulation

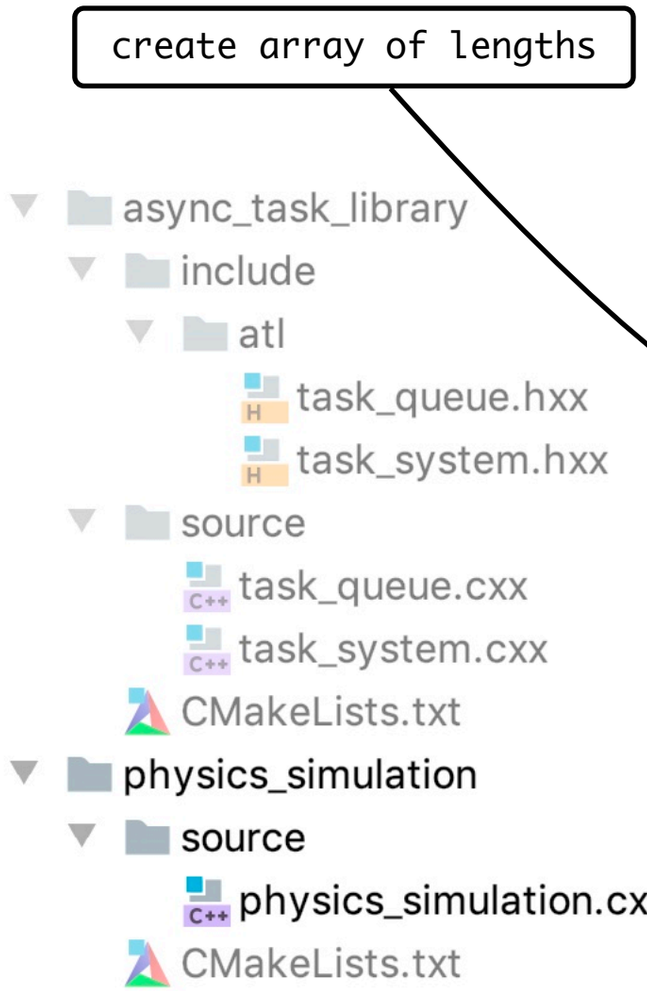
Compute Lengths - Task System Example




```cpp
#include <atl/task_system.hxx>
#include <cmath>

namespace phy_sim
{
    ...

    auto compute_lengths (const array<vec3>& vectors) -> array<float>
    {
        auto lengths = array<float> { };
        {
            auto task_system = atl::task_system { };

            for (auto index = 0; index < vectors.size(); ++index)
            {
                auto task = [index, &vectors, &lengths] ()
                {
                    lengths[index] = length(vectors[index]);
                };
                task_system.async(task);
            }
        }
        return lengths;
    }

    ...
}
```

Physics Simulation

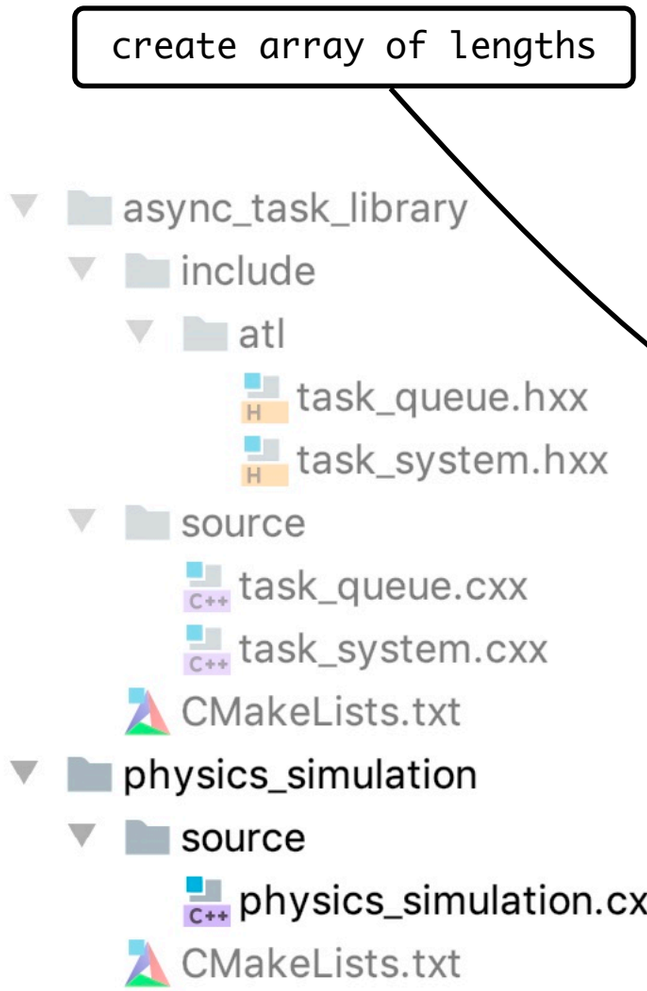
Compute Lengths - Task System Example




```cpp
#include <atl/task_system.hxx>
#include <cmath>

namespace phy_sim
{
    ...

    auto compute_lengths (const array<vec3>& vectors) -> array<float>
    {
        auto lengths = array<float> { };
        {
            auto task_system = atl::task_system { };

            for (auto index = 0; index < vectors.size(); ++index)
            {
                auto task = [index, &vectors, &lengths] ()
                {
                    lengths[index] = length(vectors[index]);
                };
                task_system.async(task);
            }
        }
        return lengths;
    }

    ...
}
```

Physics Simulation

Compute Lengths - Task System Example



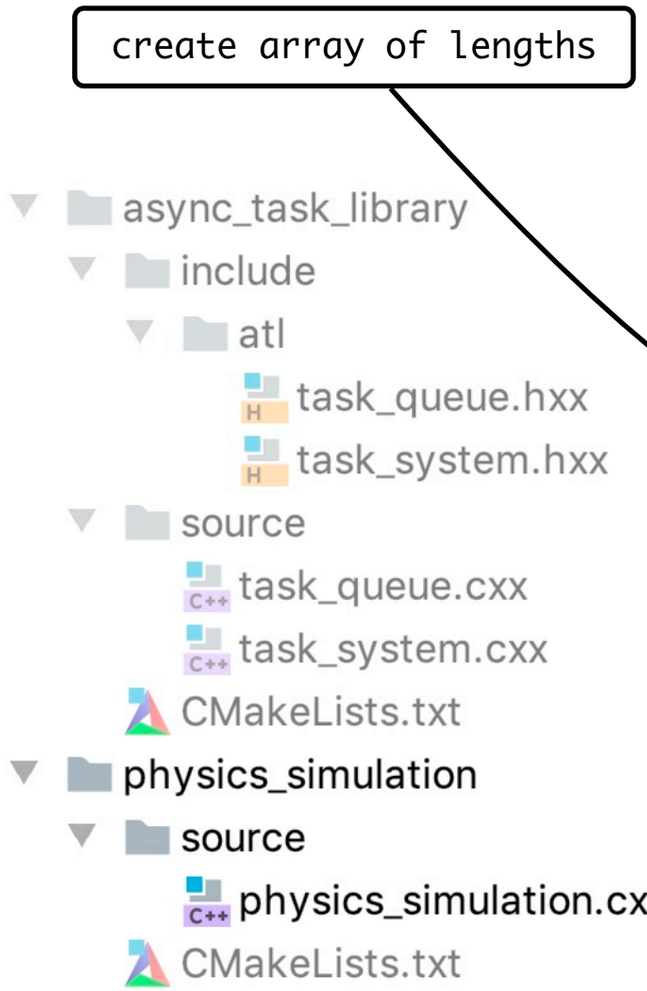
create array of lengths

instantiate task system

```cpp
#include <atl/task_system.hxx>
#include <cmath>

namespace phy_sim
{
    ...

    auto compute_lengths (const array<vec3>& vectors) -> array<float>
    {
        auto lengths = array<float> { };
        {
            auto task_system = atl::task_system { };

            for (auto index = 0; index < vectors.size(); ++index)
            {
                auto task = [index, &vectors, &lengths] ()
                {
                    lengths[index] = length(vectors[index]);
                };
                task_system.async(task);
            }
        }
        return lengths;
    }

    ...
}
```

Physics Simulation

Compute Lengths - Task System Example



create array of lengths

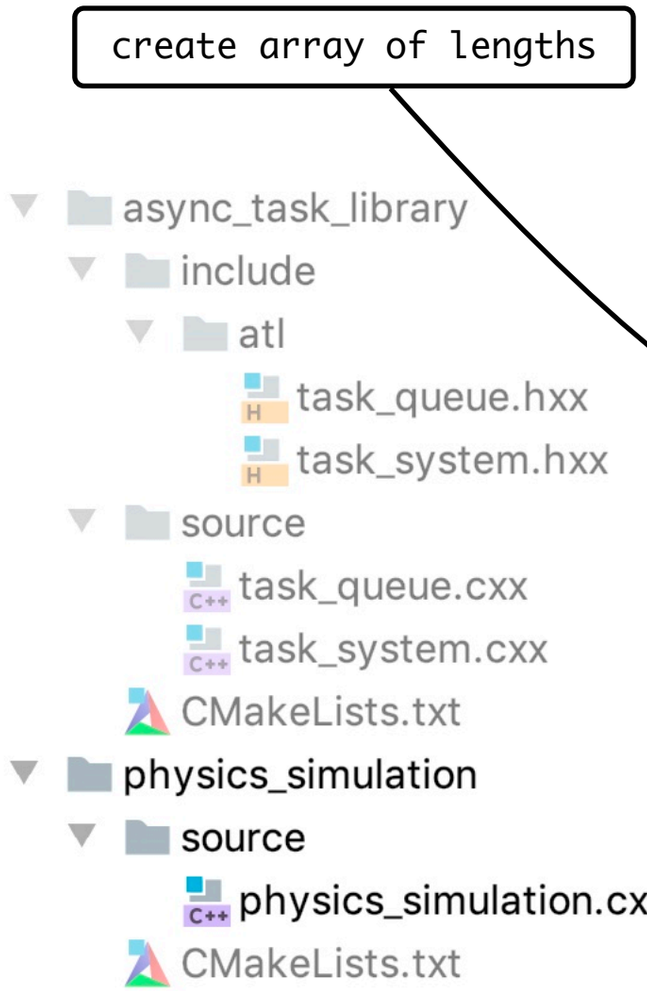
schedule one task per 3D vector

instantiate task system

```cpp
#include <atl/task_system.hxx>
#include <cmath>

namespace phy_sim
{
    ...

auto compute_lengths (const array<vec3>& vectors) -> array<float>
{
    auto lengths = array<float> { };
    {
        auto task_system = atl::task_system { };

        for (auto index = 0; index < vectors.size(); ++index)
        {
            auto task = [index, &vectors, &lengths] ()
            {
                lengths[index] = length(vectors[index]);
            };
            task_system.async(task);
        }
    }
    return lengths;
}

    ...
}
```

Physics Simulation

Compute Lengths - Task System Example



create array of lengths

```cpp
#include <atl/task_system.hxx>
#include <cmath>

namespace phy_sim
{
    ...

auto compute_lengths (const array<vec3>& vectors) -> array<float>
{
    auto lengths = array<float> { };
    {
        auto task_system = atl::task_system { };

        for (auto index = 0; index < vectors.size(); ++index)
        {
            auto task = [index, &vectors, &lengths] ()
            {
                lengths[index] = length(vectors[index]);
            };
            task_system.async(task);
        }
    }
    return lengths;
}

    ...
}
```

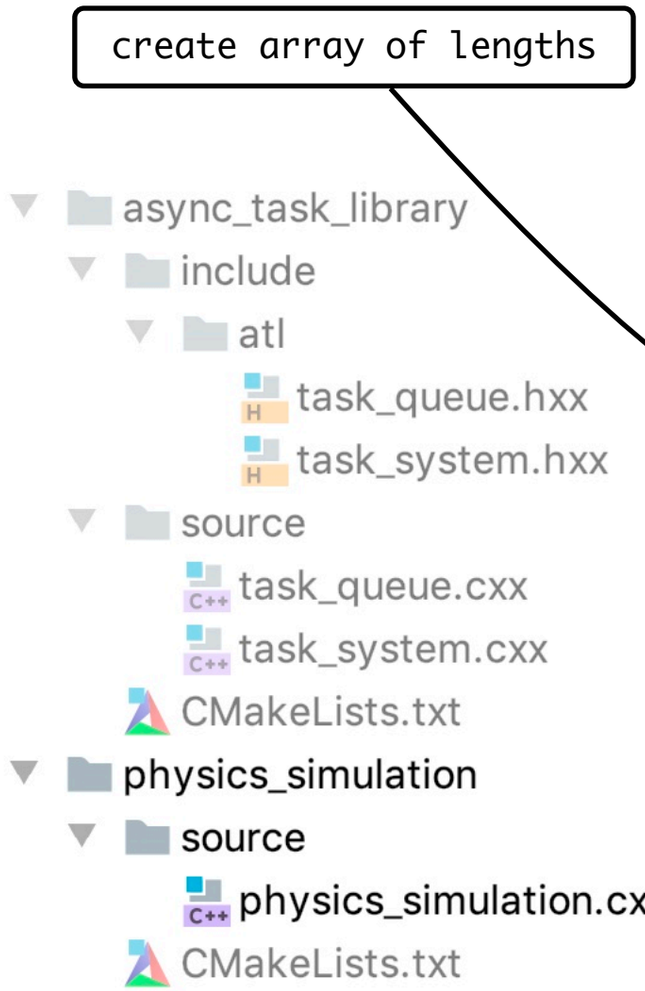
instantiate task system

each task computes length of one 3D vector

schedule one task per 3D vector

Physics Simulation

Compute Lengths - Task System Example



create array of lengths

```cpp
#include <atl/task_system.hxx>
#include <cmath>

namespace phy_sim
{
    ...

auto compute_lengths (const array<vec3>& vectors) -> array<float>
{
    auto lengths = array<float> { };
    {
        auto task_system = atl::task_system { };

        for (auto index = 0; index < vectors.size(); ++index)
        {
            auto task = [index, &vectors, &lengths] ()
            {
                lengths[index] = length(vectors[index]);
            };
            task_system.async(task);
        }
    }
    return lengths;
}

    ...
}
```

instantiate task system

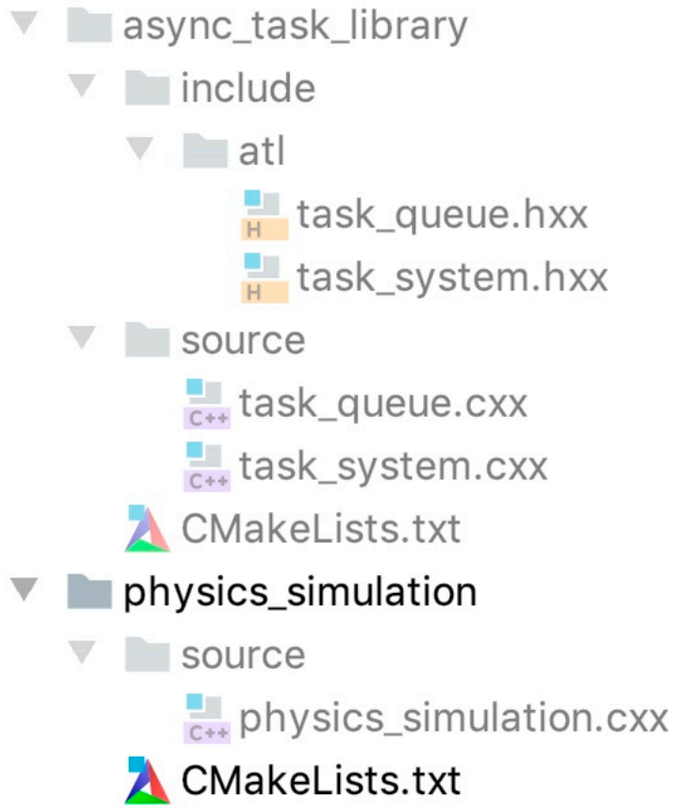
each task computes length of one 3D vector

schedule one task per 3D vector

return lengths array when all scheduled tasks have been completed

Physics Simulation

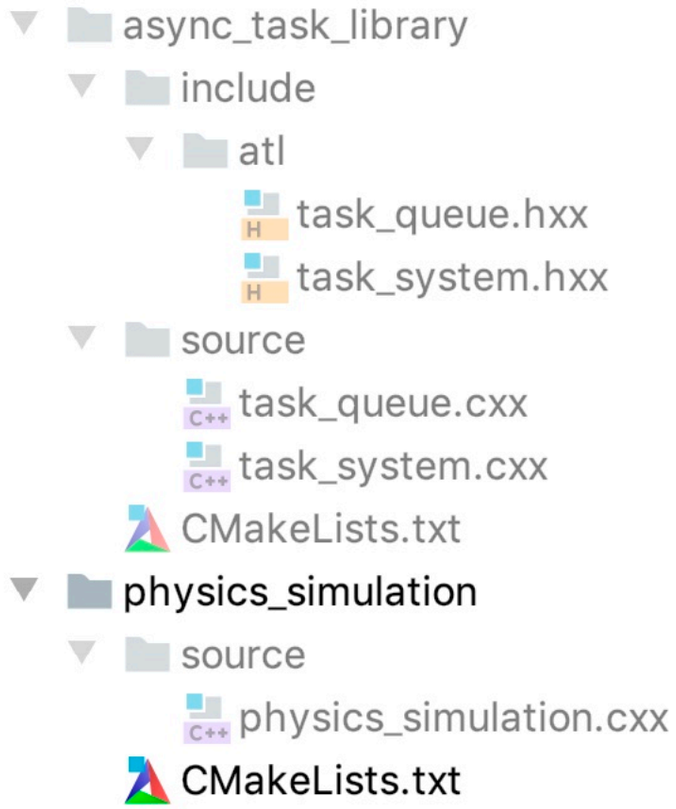
CMake build system




```
cmake_minimum_required (VERSION 3.15)

add_executable              (physics-simulation)

target_compile_features     (physics-simulation PRIVATE cxx_std_17)

target_sources              (physics-simulation PRIVATE
                                    source/physics_simulation.cxx)

target_link_libraries       (physics-simulation PRIVATE async-task-lib)
```

Physics Simulation

CMake build system

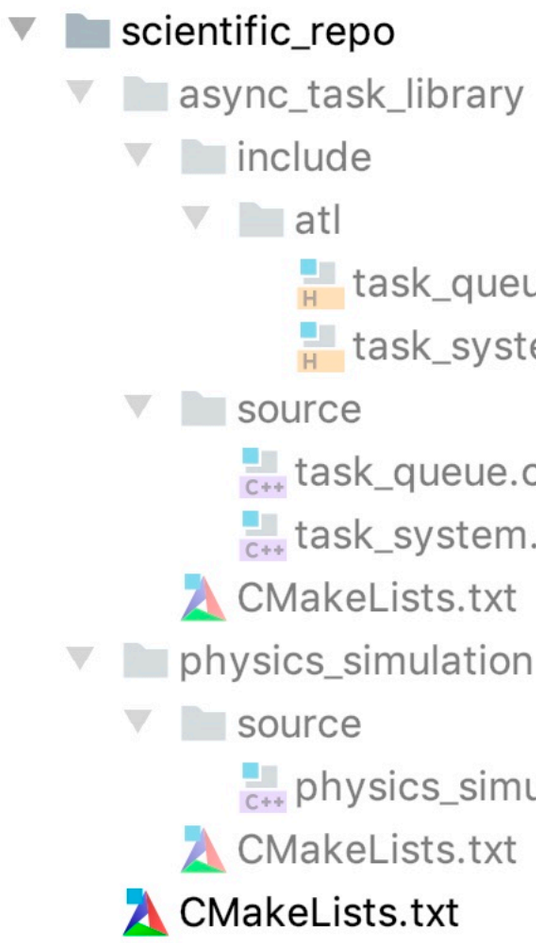



```
cmake_minimum_required (VERSION 3.15)

add_executable              (physics-simulation)

target_compile_features     (physics-simulation PRIVATE cxx_std_17)

target_sources              (physics-simulation PRIVATE
                                source/physics_simulation.cxx)

target_link_libraries       (physics-simulation PRIVATE async-task-lib)
```

define dependency between CMake targets physics-simulation and async-task-lib

Physics Simulation

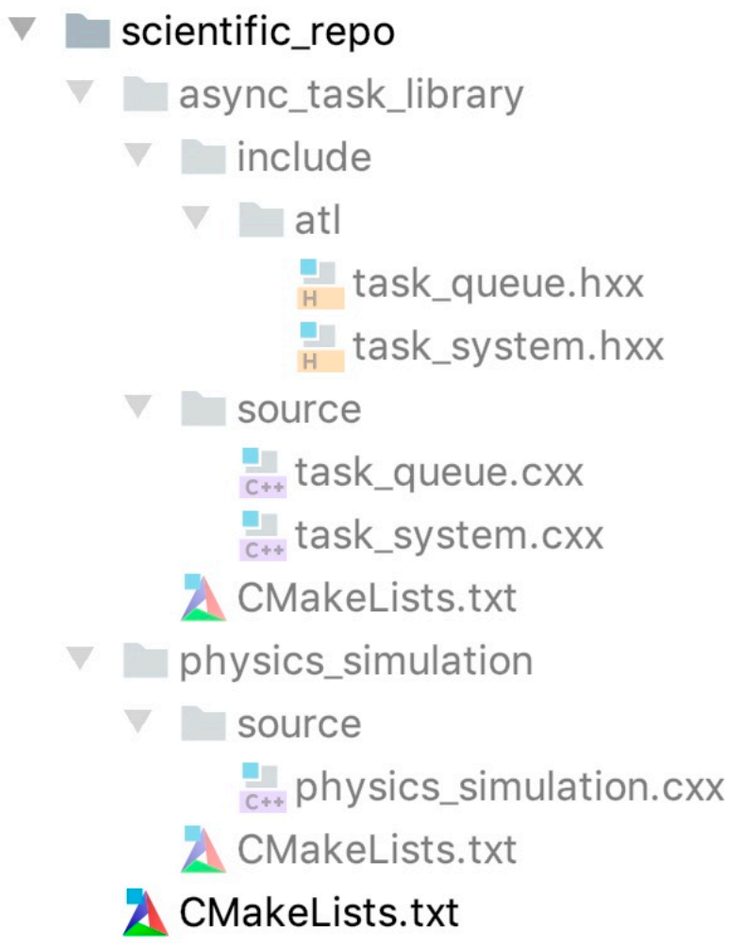
Scientific Repo




```
cmake_minimum_required (VERSION 3.15)

project(PhysicsSimulation VERSION 1.0.0 LANGUAGES CXX)


add_subdirectory(async_task_library)

add_subdirectory(physics_simulation)
```

Physics Simulation

Scientific Repo




```
cmake_minimum_required (VERSION 3.15)

project(PhysicsSimulation VERSION 1.0.0 LANGUAGES CXX)


add_subdirectory(async_task_library)
add_subdirectory(physics_simulation)
```

compose together CMake targets
async-task-lib and physics-simulation
into CMake project PhysicsSimulation

Question

What is wrong with the above code?

Question

What is wrong with the above code?

- Performance of the task system is very poor.
  - The task queue is under a lot of contention,
    since threads scheduling and executing tasks are all trying to access the same task queue.

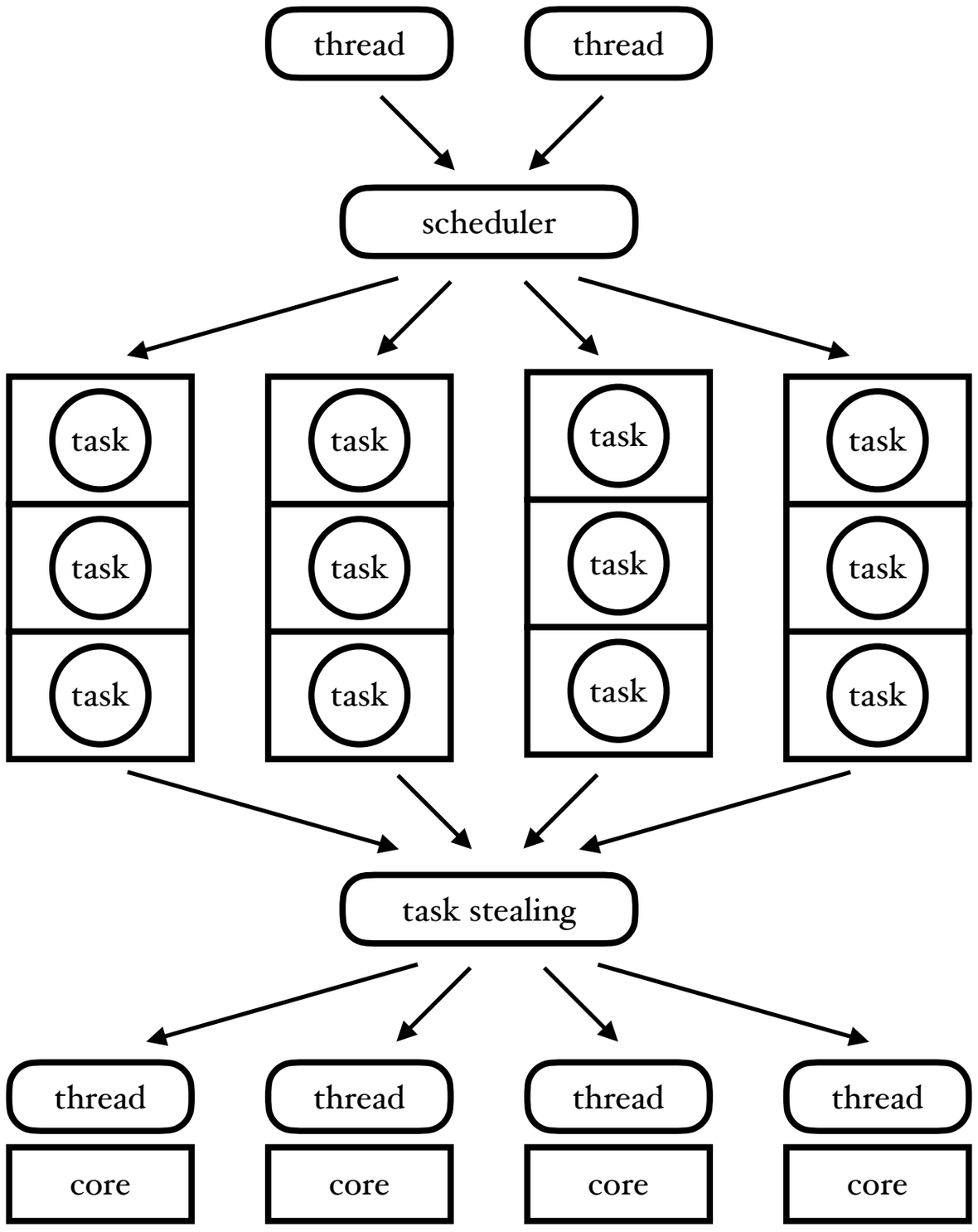
Question

What is wrong with the above code?

- Performance of the task system is very poor.
  - The task queue is under a lot of contention,
    since threads scheduling and executing tasks are all trying to access the same task queue.
  - In a much better task system:
    - ✓ each thread in the thread pool would have its own task queue,
    - ✓ a task scheduler would push tasks into a chosen queue using the round-robin algorithm,
    - ✓ load balancing would be done via task stealing.

Recommendation

Task System - Better Design



The scheduler `push()`es a task into the first non-busy queue.

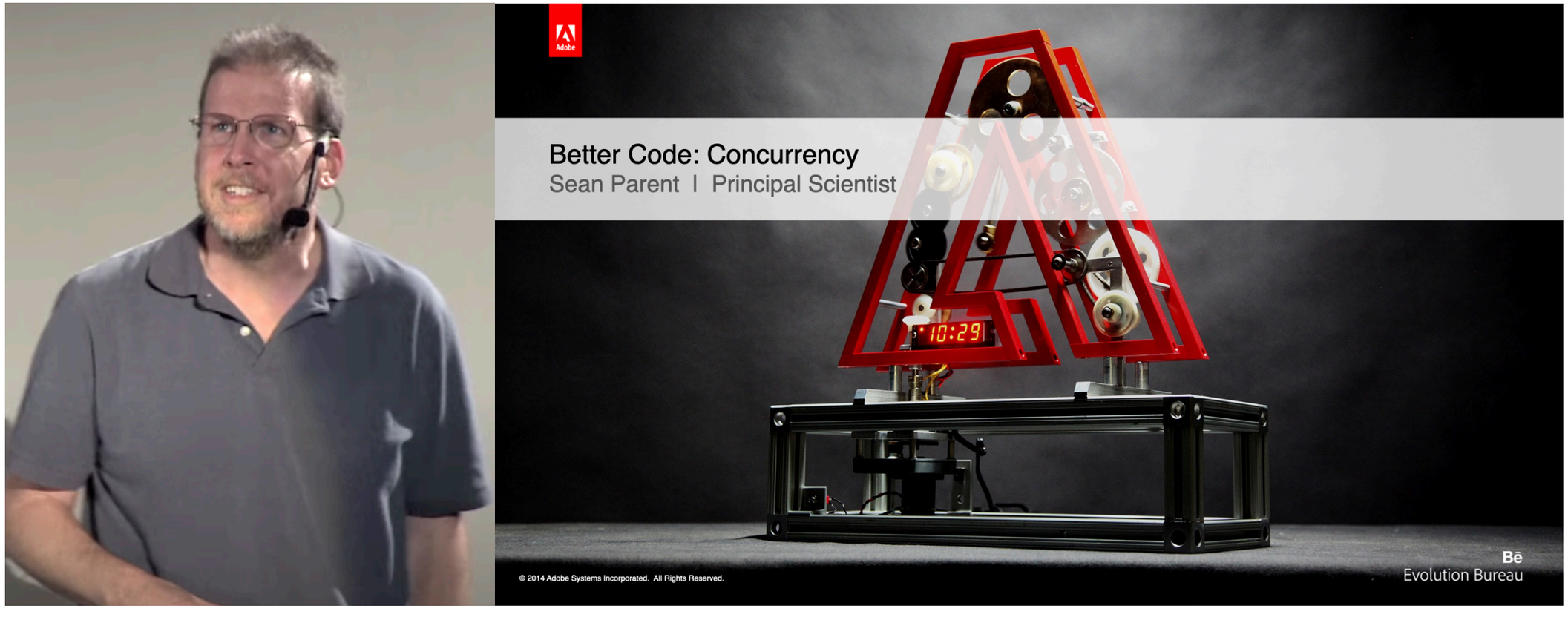
If a thread can't `pop()` a task from its own queue, because its empty or busy, then it will try to `pop()` a task from a different queue.

Recommendation

Task System - Better Design

In fact, do not listen to me!

Instead listen to the Sean Parent's talk "Better Code: Concurrency",
in which he discusses and implements properly such a task system.



Question

What else is wrong with the above code?

- Performance of the task system is very poor.

Question

What else is wrong with the above code?

- Performance of the task system is very poor.
- Interface of the task system is not precise.

```
namespace atl
{
    class task_system
    {
    public:
        task_system ();
        ~task_system ();

        template <typename callable>
        auto async (callable&& task) -> void
        {
            queue.push(std::forward<callable>(task));
        }
    };
}
```

What is a task?

Question

What else is wrong with the above code?

- Performance of the task system is very poor.
- Interface of the task system is not precise.

```
namespace atl
{
    class task_system
    {
    public:
        task_system ();
        ~task_system ();

        template <typename callable>
        auto async (callable&& task) -> void
        {
            queue.push(std::forward<callable>(task));
        }
    };
}
```

What is a task?

Well, it's something we can push into the task queue.

Question

What else is wrong with the above code?

- Performance of the task system is very poor.
- Interface of the task system is not precise.

```
namespace atl
{
    class task_queue
    {
    private:
        std::deque<std::function<void()>> tasks;

    public:
        template <typename callable>
        auto push (callable&& task) -> void
        {
            {
                auto lock = std::lock_guard<std::mutex> { mutex };
                tasks.emplace_back(std::forward<callable>(task));
            }
            ready.notify_one();
        }
    };
}
```

OK. What can be pushed into the task queue?

Question

What else is wrong with the above code?

- Performance of the task system is very poor.
- Interface of the task system is not precise.

```
namespace atl
{
    class task_queue
    {
    private:
        std::deque<std::function<void()>> tasks;

    public:
        template <typename callable>
        auto push (callable&& task) -> void
        {
            {
                auto lock = std::lock_guard<std::mutex> { mutex };
                tasks.emplace_back(std::forward<callable>(task));
            }
            ready.notify_one();
        }
    };
}
```

OK. What can be pushed into the task queue?
Well, something convertible to a `std::function<void()>`.

Recommendation

Task System - Better Design

- Performance of the task system is very poor.
- Interface of the task system is not precise.

```
namespace atl
{
    template <typename type>
    concept task = std::is_invocable_r_v<void, type>;

    class task_system
    {
        ...

    public:
        task_system ();
       ~task_system ();

        auto async (task auto&& t) -> void
        { ... }
    };
}
```

This thinking is all wrong!
Instead of expressing interface in terms of implementation details,
we need to define the **`task concept`**,
which specifies what does it mean to be a task.

Recommendation

Task System - Better Design

Again, do not listen to me!

Instead listen to the Bjarne Stroustrup's talk about interfaces of templates "Concepts: The Future of Generic Programming".

Concepts:
The Future of Generic Programming
(the future is here)

Bjarne Stroustrup
Morgan Stanley and Columbia University
www.stroustrup.com

Question

What else is wrong with the above code?

- Performance of the task system is very poor.
- Interface of the task system is not precise.

Question

What else is wrong with the above code?

- Performance of the task system is very poor.
- Interface of the task system is not precise.
- Implementation details of the task system are exposed to clients.

```cpp
#include <atl/task_system.hxx>

namespace phy_sim
{
    auto function () -> void
    {
        auto task_system = atl::task_system { };
        auto task_queue  = atl::task_queue  { };
   }
}
```

The physics simulation executable has access to the `task_queue` type.

```cpp
#include <atl/task_queue.hxx>

namespace atl
{
    class task_system
    {
        ...
    private:
        task_queue queue;
    };
}
```

Question

What else is wrong with the above code?

- Performance of the task system is very poor.
- Interface of the task system is not precise.
- Implementation details of the task system are exposed to clients.

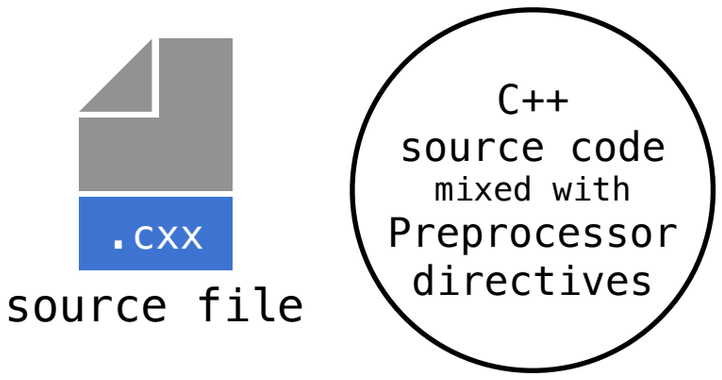
```cpp
#include <optional>
namespace atl
{
    class task_queue
    {
        auto pop () -> std::optional<std::function<void()>>;
  };
}
```

```cpp
#include <atl/task_system.hxx>
namespace phy_sim
{
    auto function () -> void
    {
        auto opt_int = std::optional<int> { 7 };
    }
}
```

The physics simulation executable has also access to the `std::optional` template.

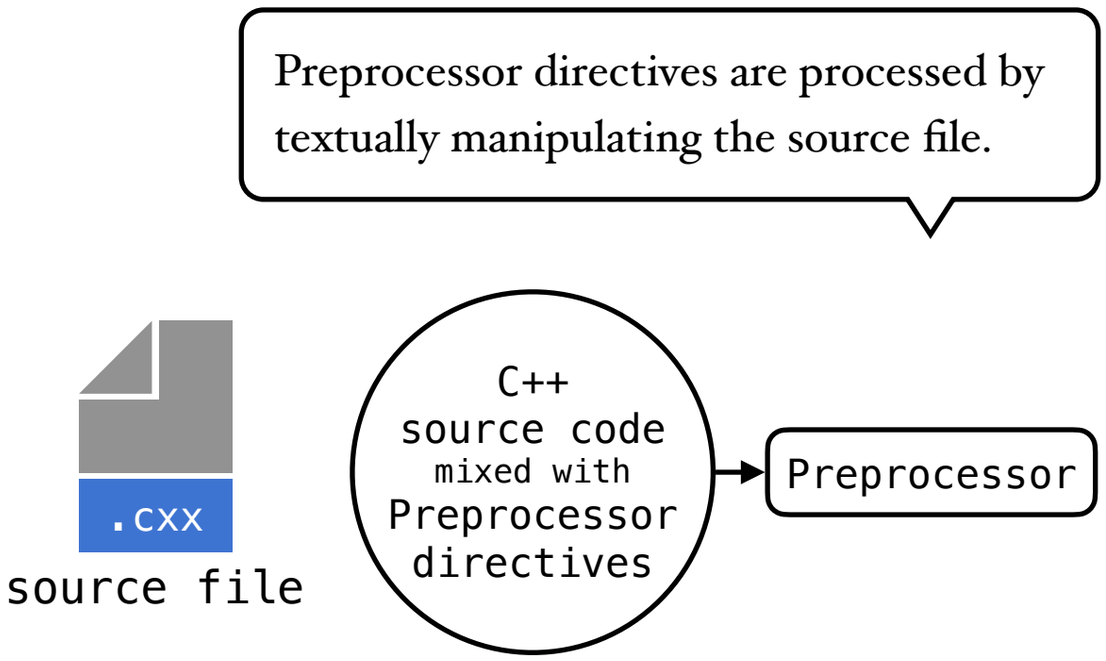
Textual Inclusion Model

Preprocessing & Compilation



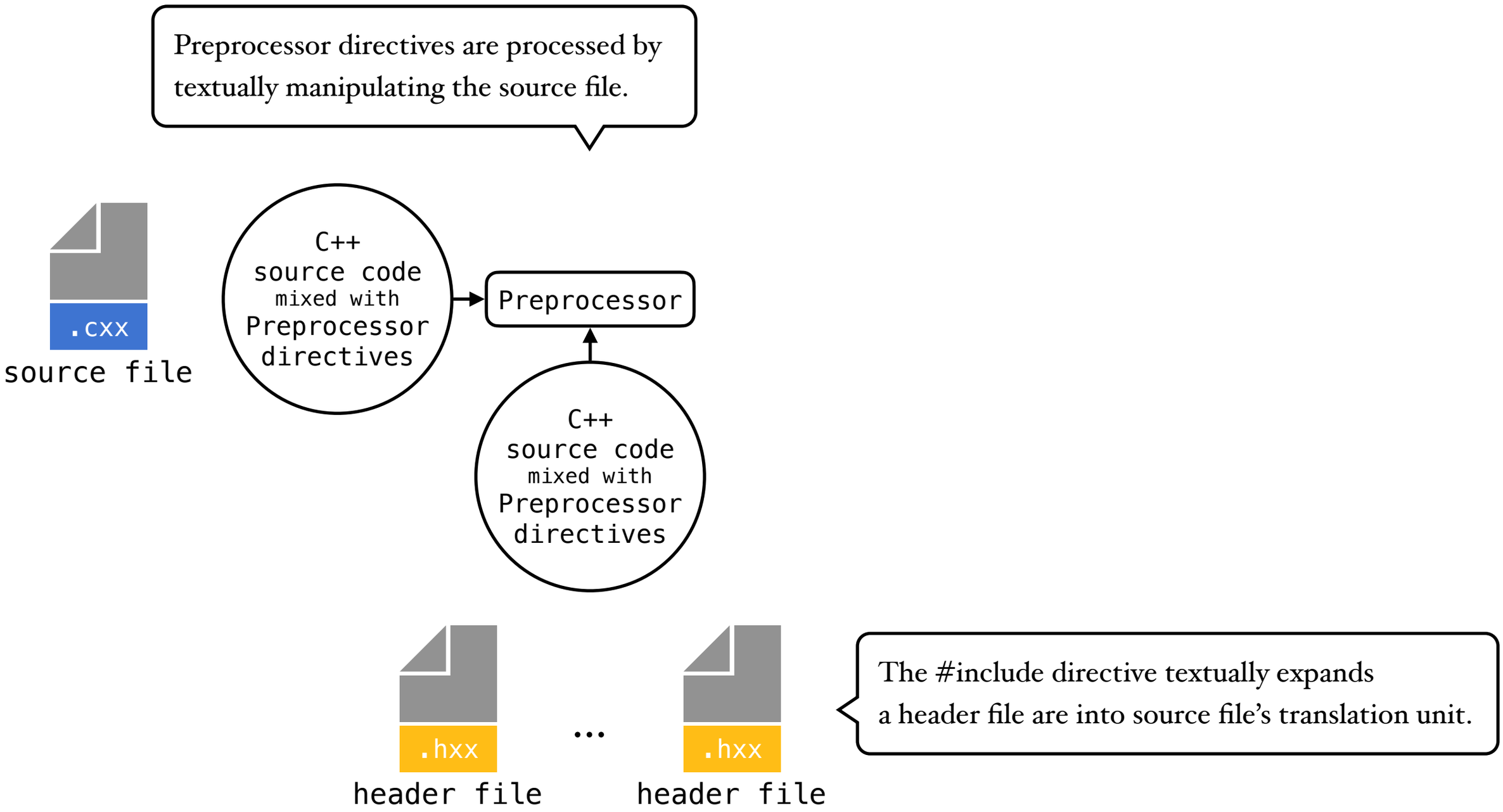
Textual Inclusion Model

Preprocessing & Compilation



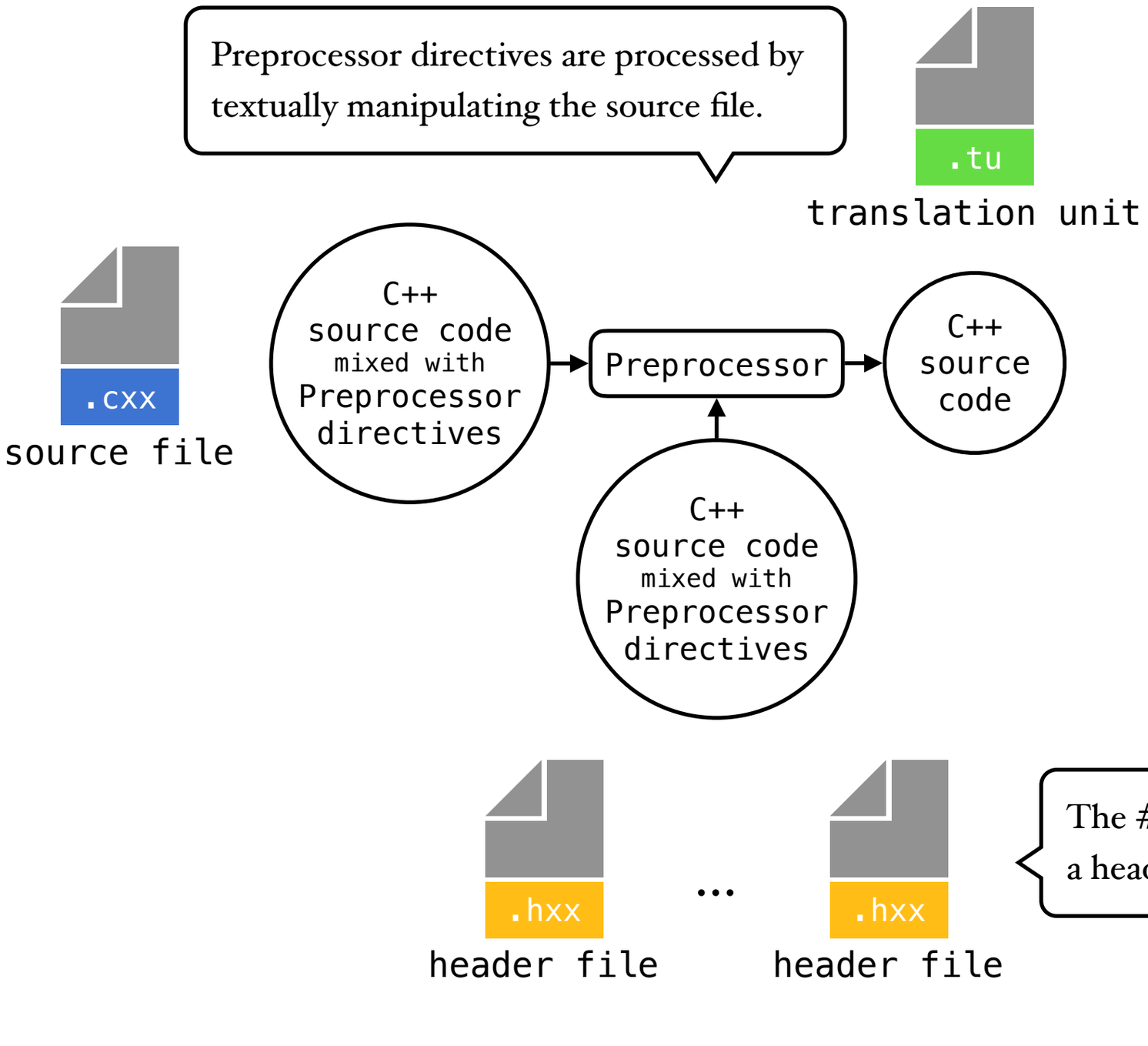
Textual Inclusion Model

Preprocessing & Compilation



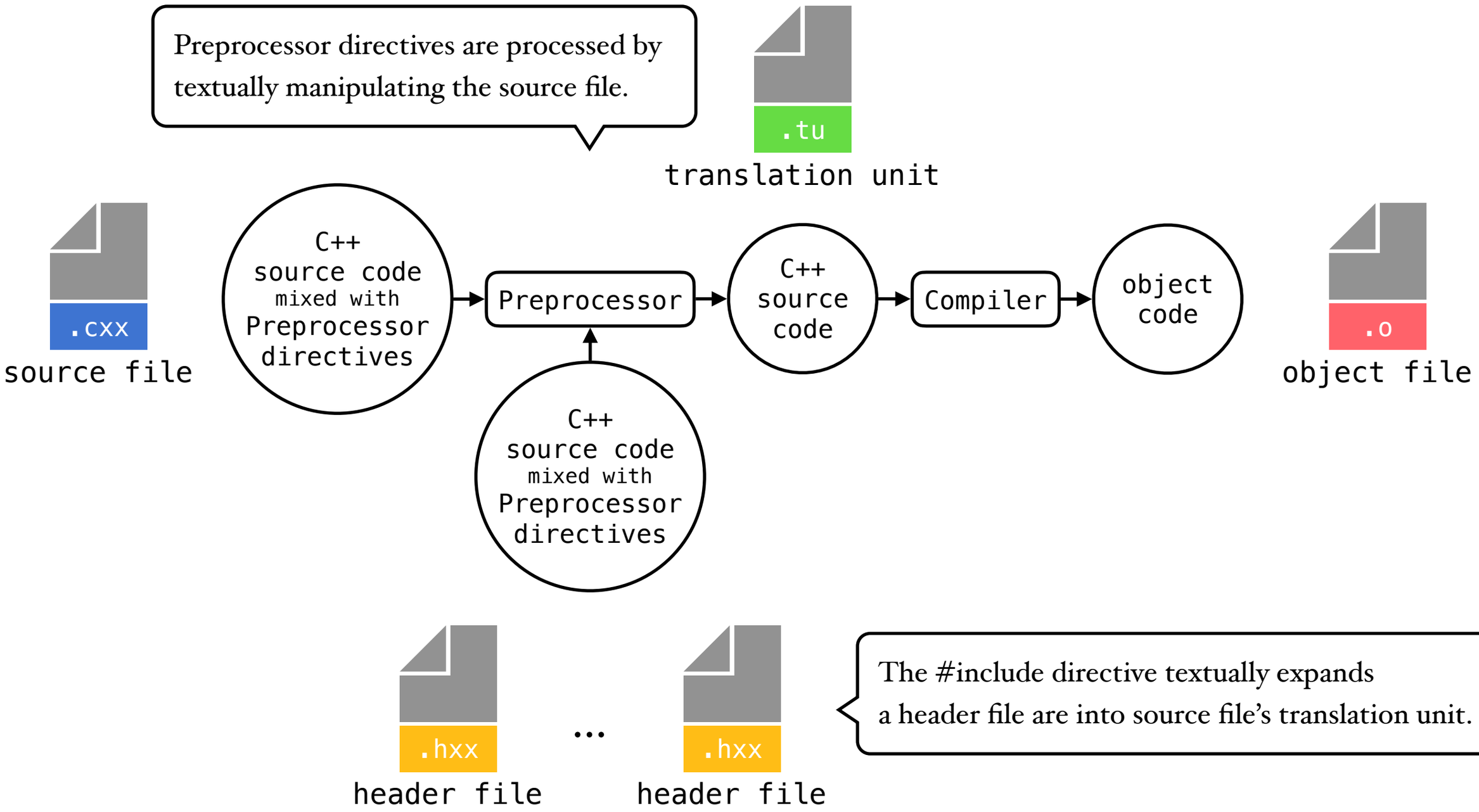
Textual Inclusion Model

Preprocessing & Compilation



Textual Inclusion Model

Preprocessing & Compilation



Textual Inclusion Model

Translation Unit

```cpp
// task_system.hxx
#ifndef ATL_TASK_SYSTEM
#define ATL_TASK_SYSTEM

#include <atl/task_queue.hxx>
...
namespace atl
{
    class task_system
    { ... };
}
#endif
```

```cpp
// task_queue.hxx
#ifndef ATL_TASK_QUEUE
#define ATL_TASK_QUEUE
...
#include <optional>

namespace atl
{
    class task_queue
    { ... };
}
#endif
```

```cpp
// physics_simulation.cxx

#include <atl/task_system.hxx>
...
namespace phy_sim
{ ... }

auto main () -> int
{ ... }
```

```cpp
// optional
#ifndef LIBCPP_OPTIONAL
#define LIBCPP_OPTIONAL
...
namespace std
{
  ...
  template<class T>
  class optional
  { ... };
}
#endif
```

Textual Inclusion Model

Translation Unit

```
// task_system.hxx
#ifndef ATL_TASK_SYSTEM
#define ATL_TASK_SYSTEM

#include <atl/task_queue.hxx>
...
namespace atl
{
    class task_system
    { ... };
}
#endif
```

```
// task_queue.hxx
#ifndef ATL_TASK_QUEUE
#define ATL_TASK_QUEUE
...
#include <optional>

namespace atl
{
    class task_queue
    { ... };
}
#endif
```

```
// physics_simulation.cxx

#include <atl/task_system.hxx>
...
namespace phy_sim
{ ... }

auto main () -> int
{ ... }
```

```
// optional
#ifndef LIBCPP_OPTIONAL
#define LIBCPP_OPTIONAL
...
namespace std
{
  ...
  template<class T>
  class optional
  { ... };
}
#endif
```

```
// physics_simulation.tu
namespace std
{
    ...
    template <class T>
    class optional
    { ... };
}
namespace atl
{
    class task_queue
    { ... };
}
...
namespace atl
{
    class task_system
    { ... };
}
...
namespace phy_sim
{ ... }
auto main () -> int
{ ... }
```

Textual Inclusion Model

Translation Unit

```
$ clang++ -std=c++17                                                \
-I //scientific_repo/async_task_library/include                     \
-E //scientific_repo/physics_simulation/source/physics_simulation.cxx \
-o physics_simulation.tu
```

```
// task_system.hxx
#ifndef ATL_TASK_SYSTEM
#define ATL_TASK_SYSTEM

#include <atl/task_queue.hxx>
...
namespace atl
{
    class task_system
    { ... };
}
#endif
```

```
// task_queue.hxx
#ifndef ATL_TASK_QUEUE
#define ATL_TASK_QUEUE
...
#include <optional>

namespace atl
{
    class task_queue
    { ... };
}
#endif
```

```
// physics_simulation.cxx

#include <atl/task_system.hxx>
...
namespace phy_sim
{ ... }

auto main () -> int
{ ... }
```

```
// optional
#ifndef LIBCPP_OPTIONAL
#define LIBCPP_OPTIONAL
...
namespace std
{
  ...
  template<class T>
  class optional
  { ... };
}
#endif
```

```
// physics_simulation.tu
namespace std
{
    ...
    template <class T>
    class optional
    { ... };
}
namespace atl
{
    class task_queue
    { ... };
}
...
namespace atl
{
    class task_system
    { ... };
}
...
namespace phy_sim
{ ... }
auto main () -> int
{ ... }
```

Textual Inclusion Model

Translation Unit

```
$ clang++ -std=c++17                                                \
-I //scientific_repo/async_task_library/include                     \
-E //scientific_repo/physics_simulation/source/physics_simulation.cxx \
-o physics_simulation.tu
```

Notice that, C++ compilers operate on a different representation of C++ programs than C++ programmers.

```
// task_system.hxx
#ifndef ATL_TASK_SYSTEM
#define ATL_TASK_SYSTEM

#include <atl/task_queue.hxx>
...
namespace atl
{
    class task_system
    { ... };
}
#endif
```

```
// task_queue.hxx
#ifndef ATL_TASK_QUEUE
#define ATL_TASK_QUEUE
...
#include <optional>

namespace atl
{
    class task_queue
    { ... };
}
#endif
```

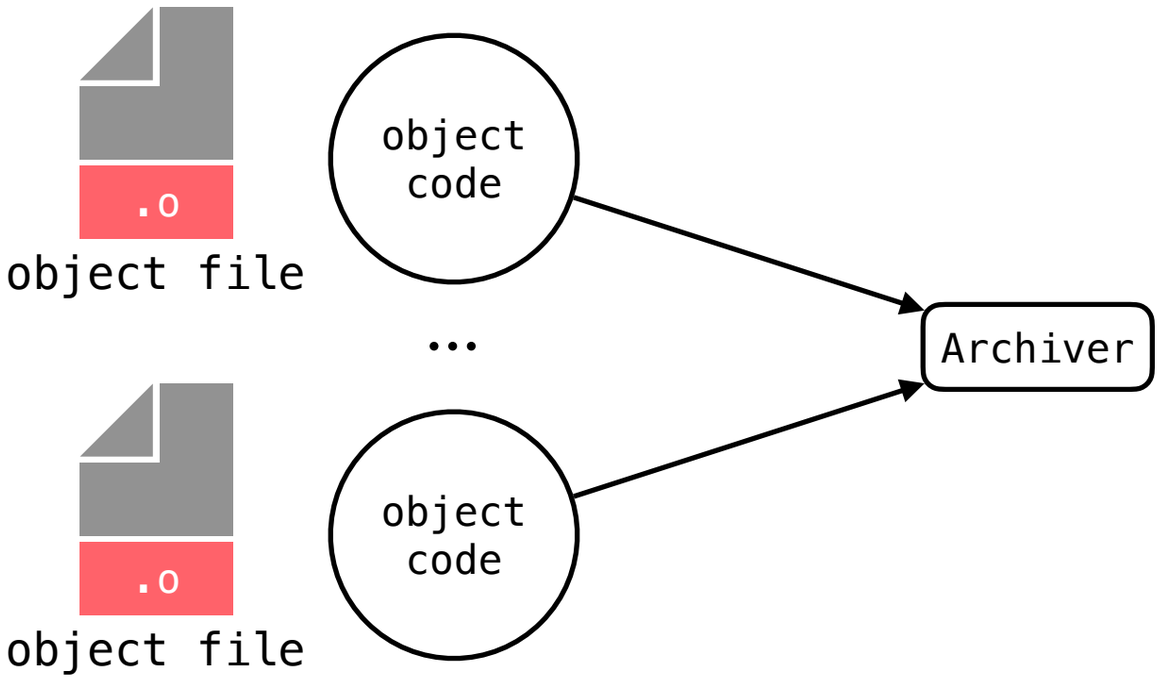
```
// physics_simulation.cxx

#include <atl/task_system.hxx>
...
namespace phy_sim
{ ... }

auto main () -> int
{ ... }
```

```
// optional
#ifndef LIBCPP_OPTIONAL
#define LIBCPP_OPTIONAL
...
namespace std
{
  ...
  template<class T>
  class optional
  { ... };
}
#endif
```

```
// physics_simulation.tu
namespace std
{
    ...
    template <class T>
    class optional
    { ... };
}
namespace atl
{
    class task_queue
    { ... };
}
...
namespace atl
{
    class task_system
    { ... };
}
...
namespace phy_sim
{ ... }
auto main () -> int
{ ... }
```

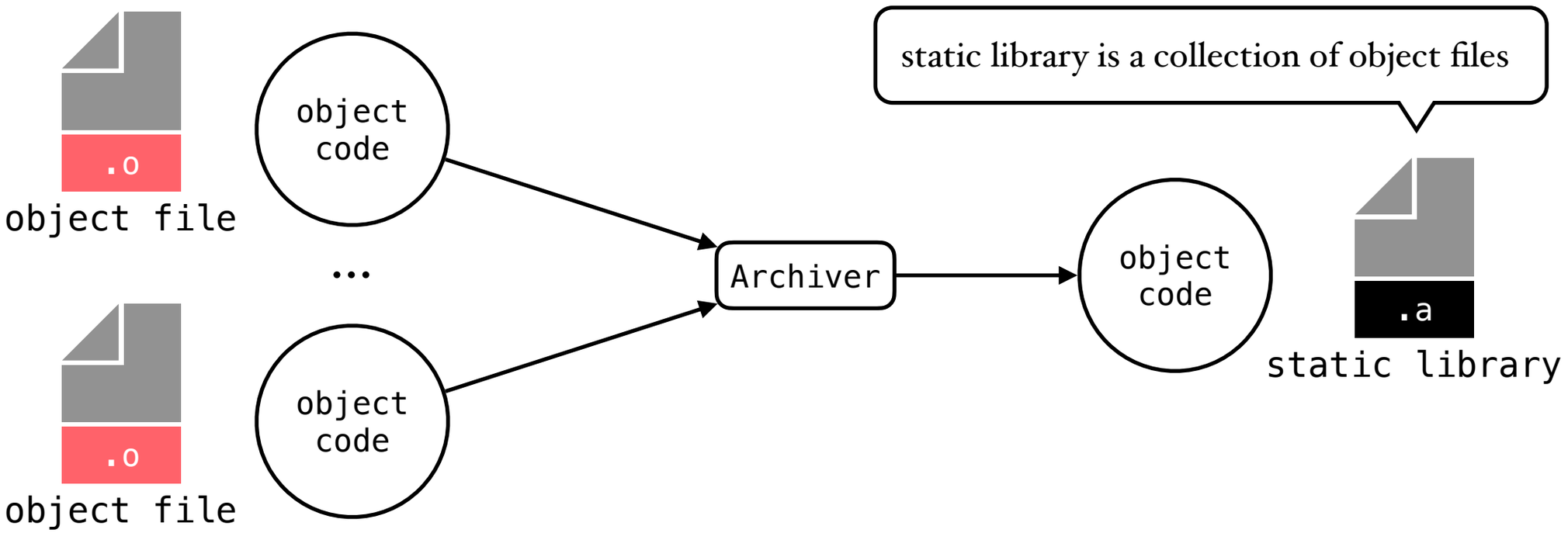
Textual Inclusion Model

Linking



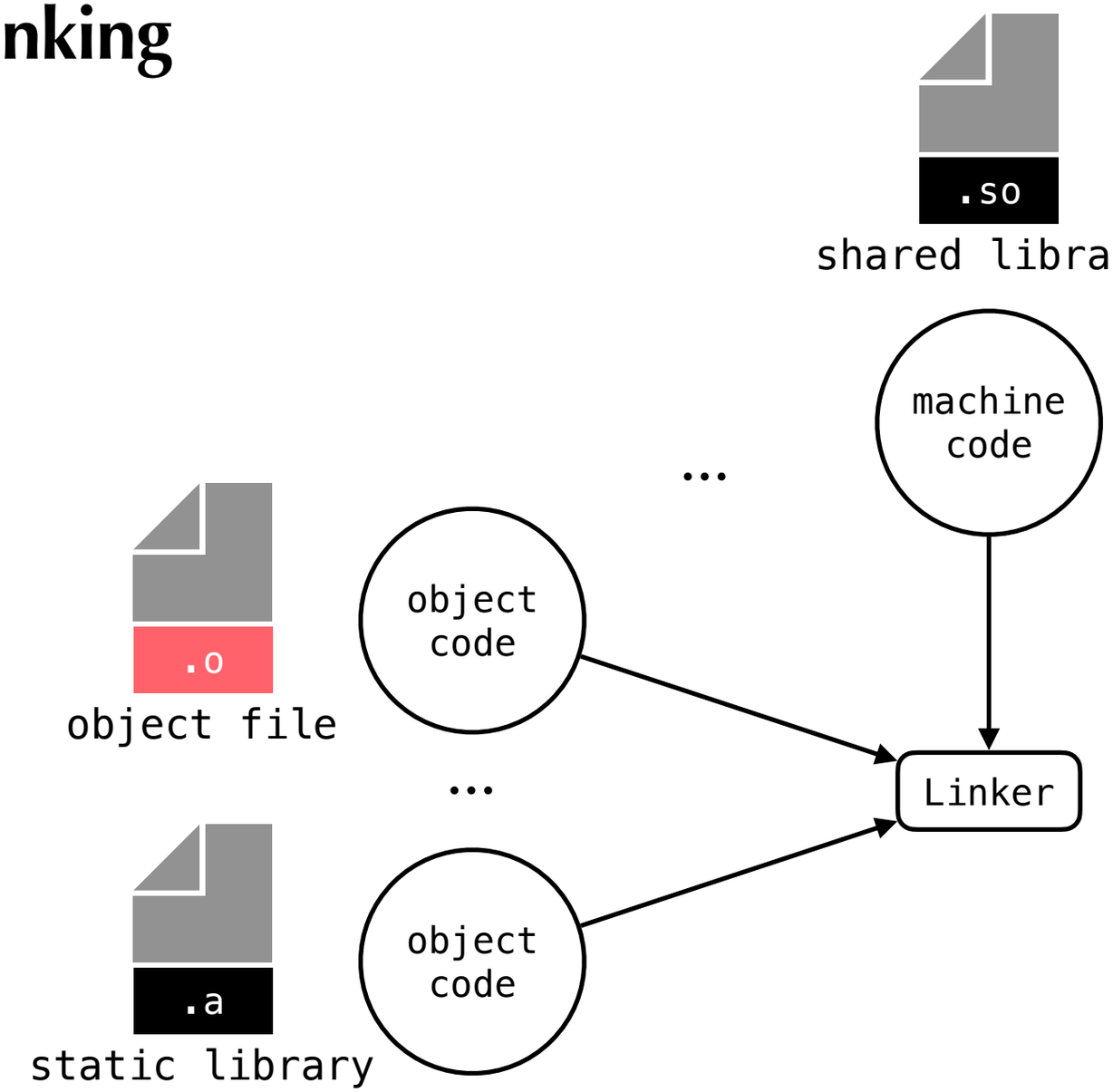
Textual Inclusion Model

Linking



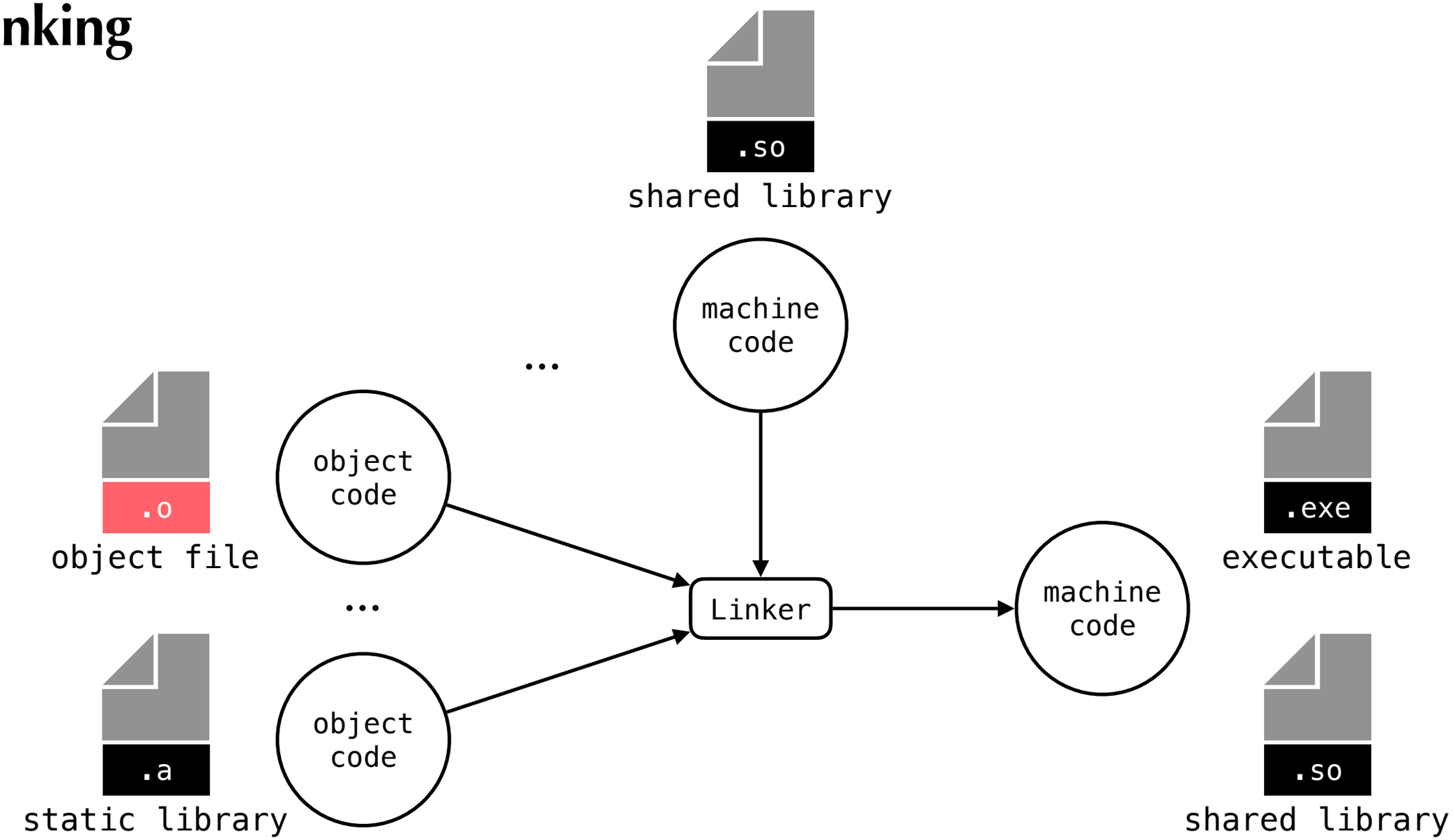
Textual Inclusion Model

Linking



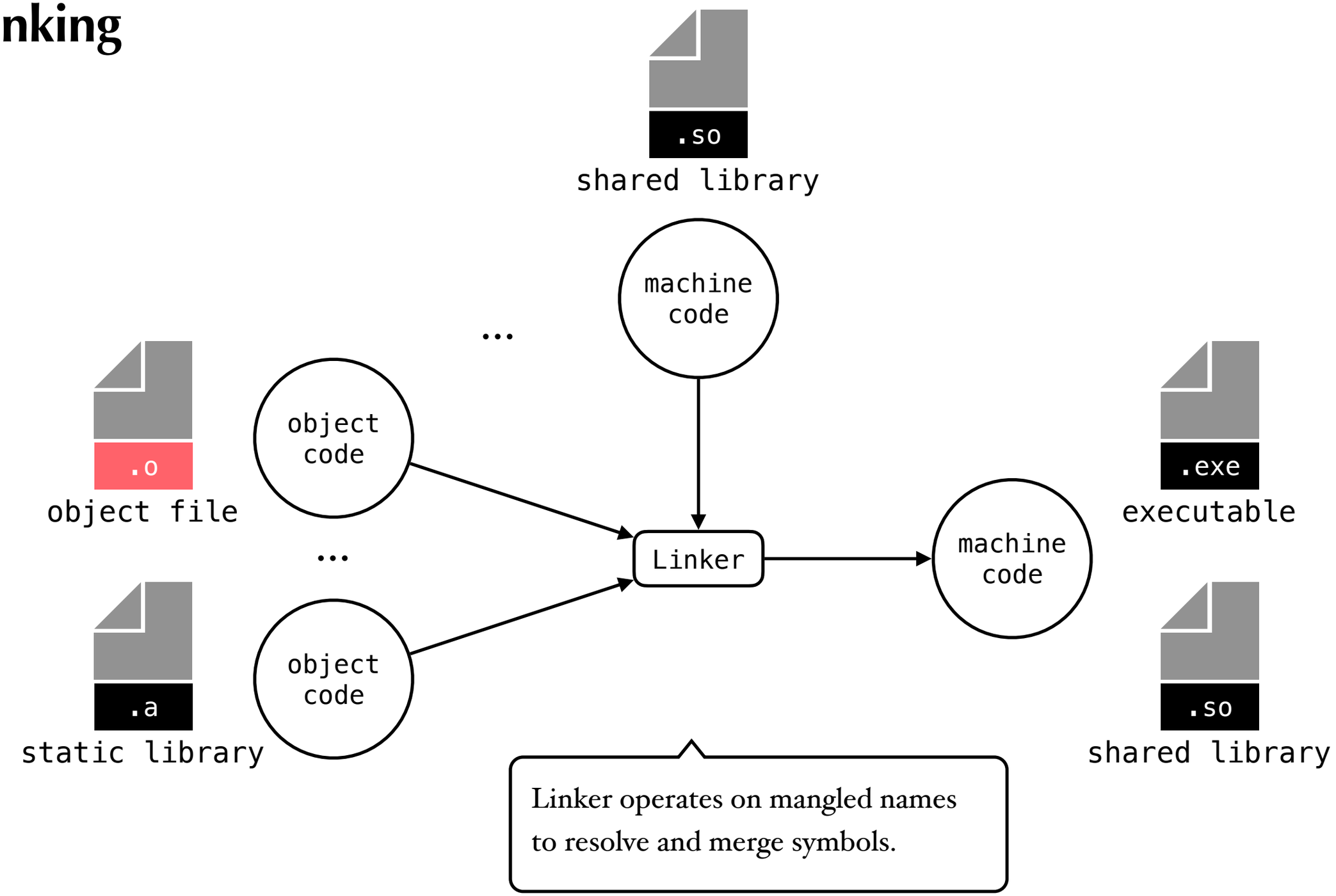
Textual Inclusion Model

Linking



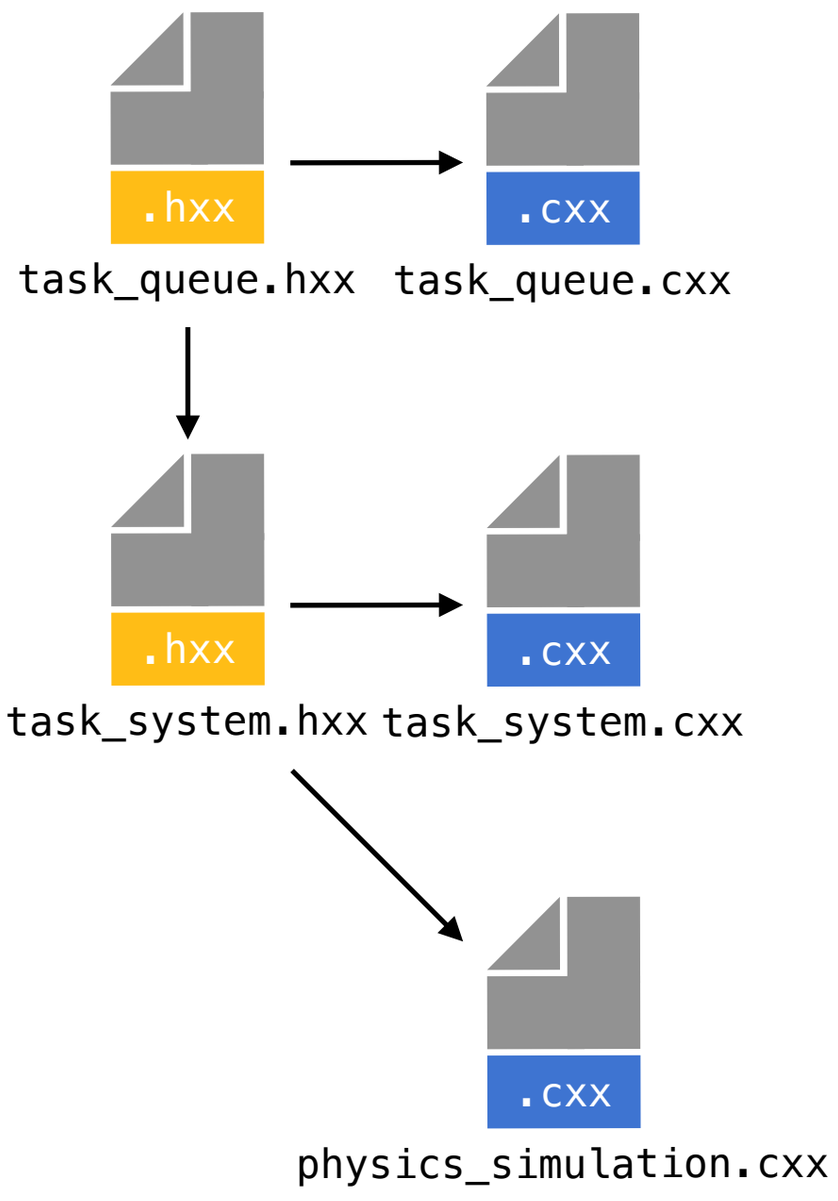
Textual Inclusion Model

Linking



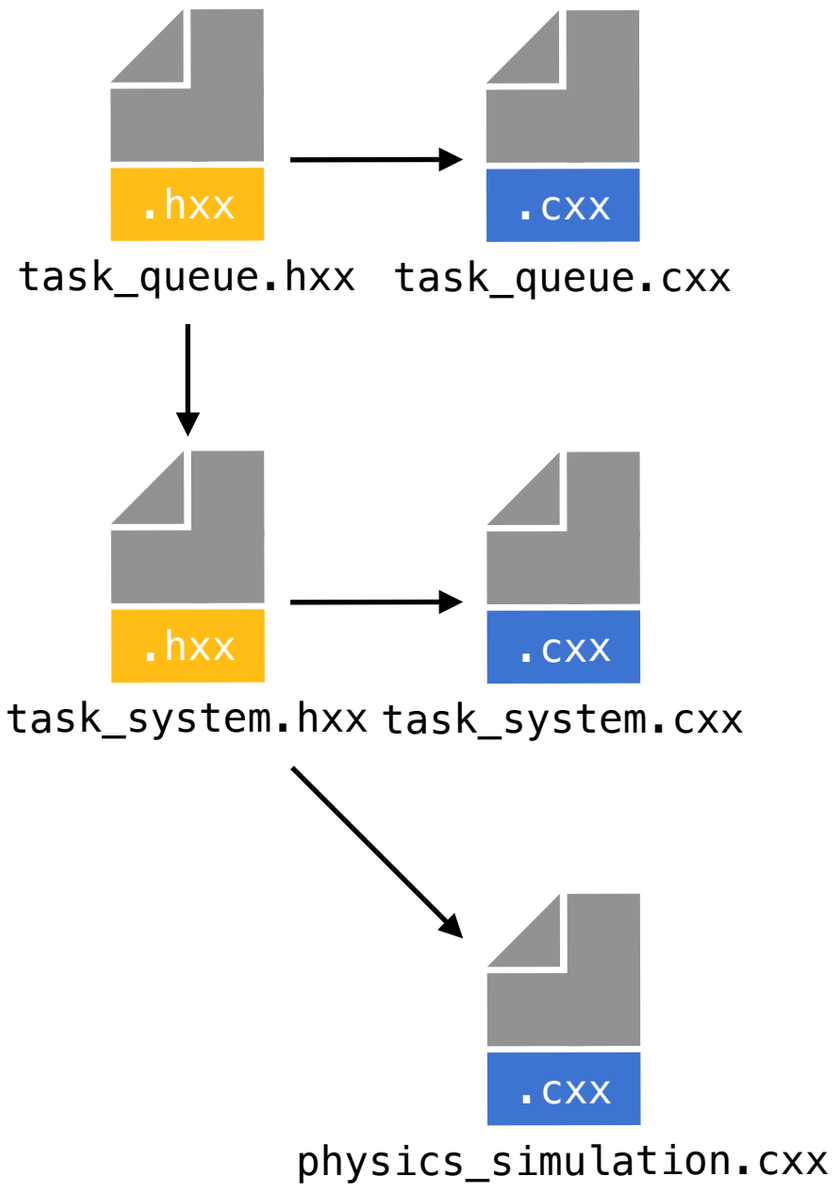
Textual Inclusion Model

Inclusion Graph



Textual Inclusion Model

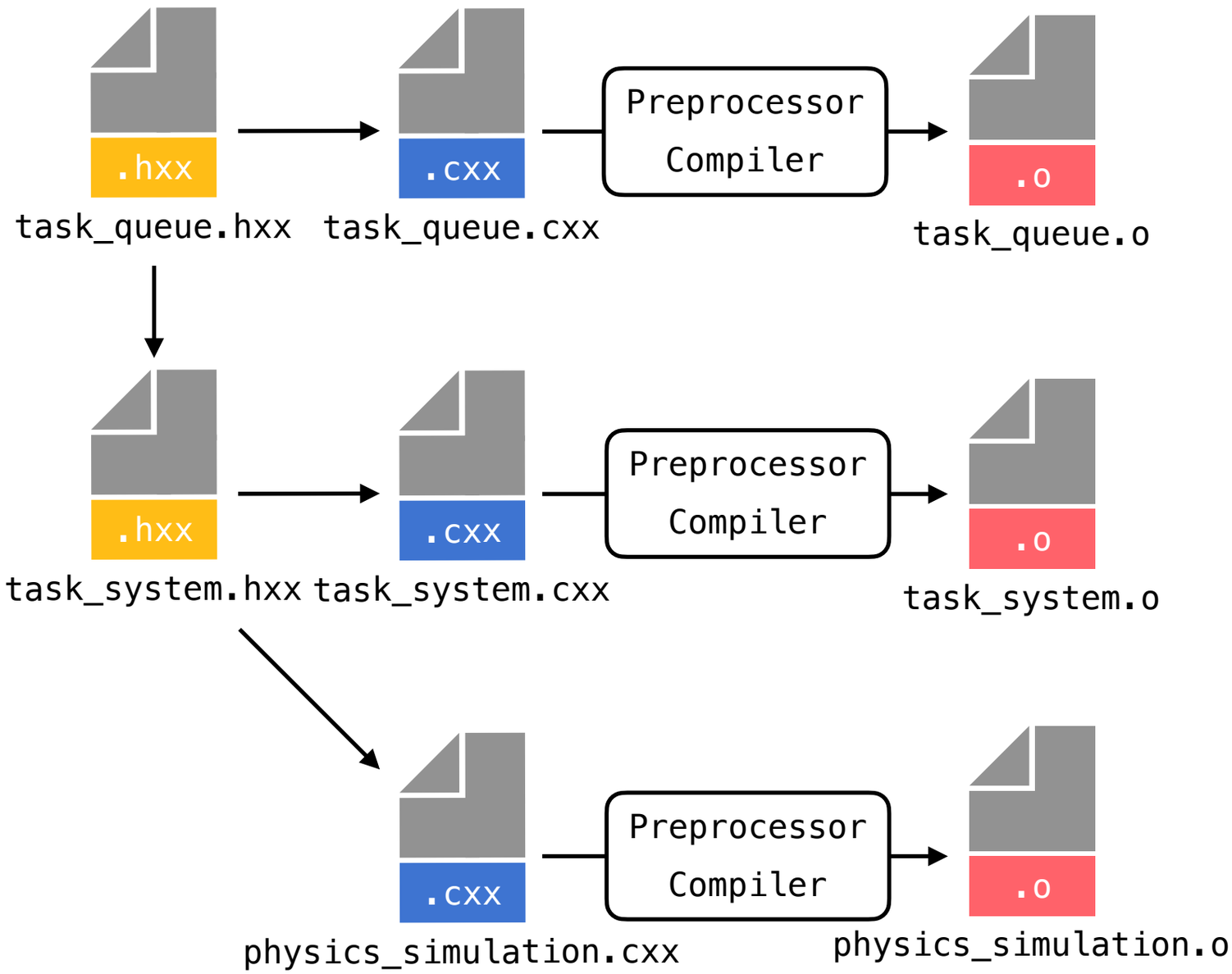
Inclusion Graph



This is very low-level view of the build process,
since it does not define dependencies between components.

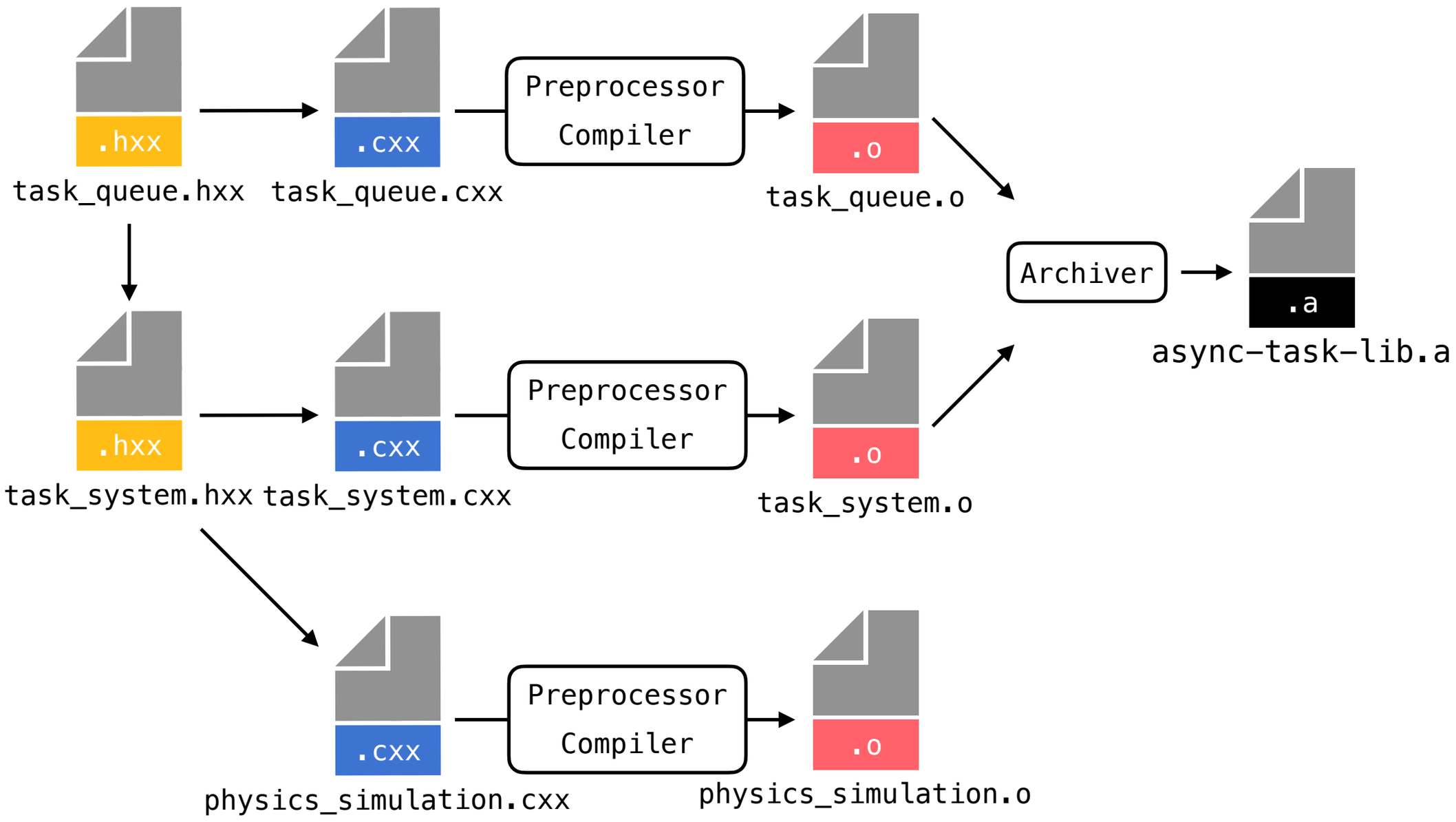
Textual Inclusion Model

Build Workflow



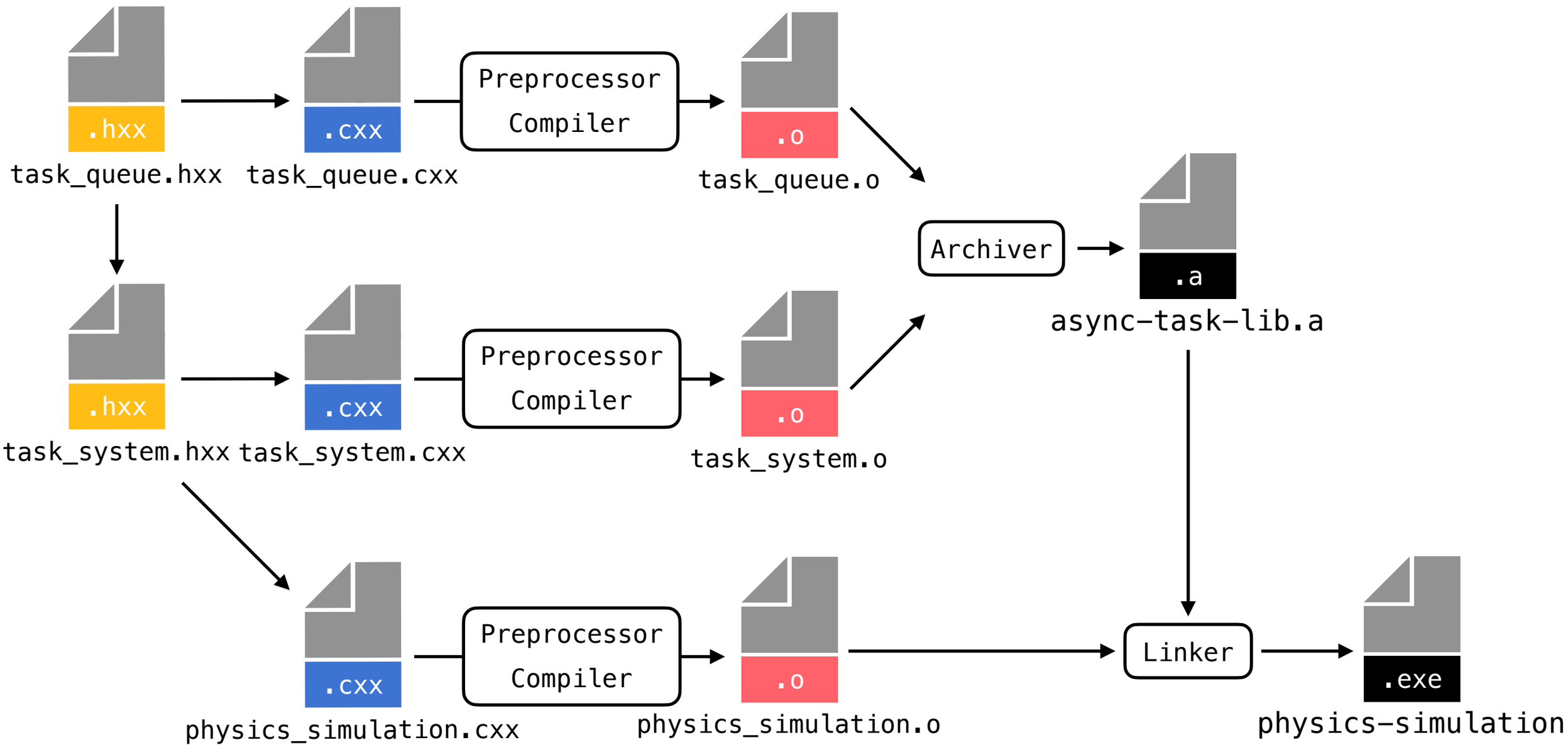
Textual Inclusion Model

Build Workflow



Textual Inclusion Model

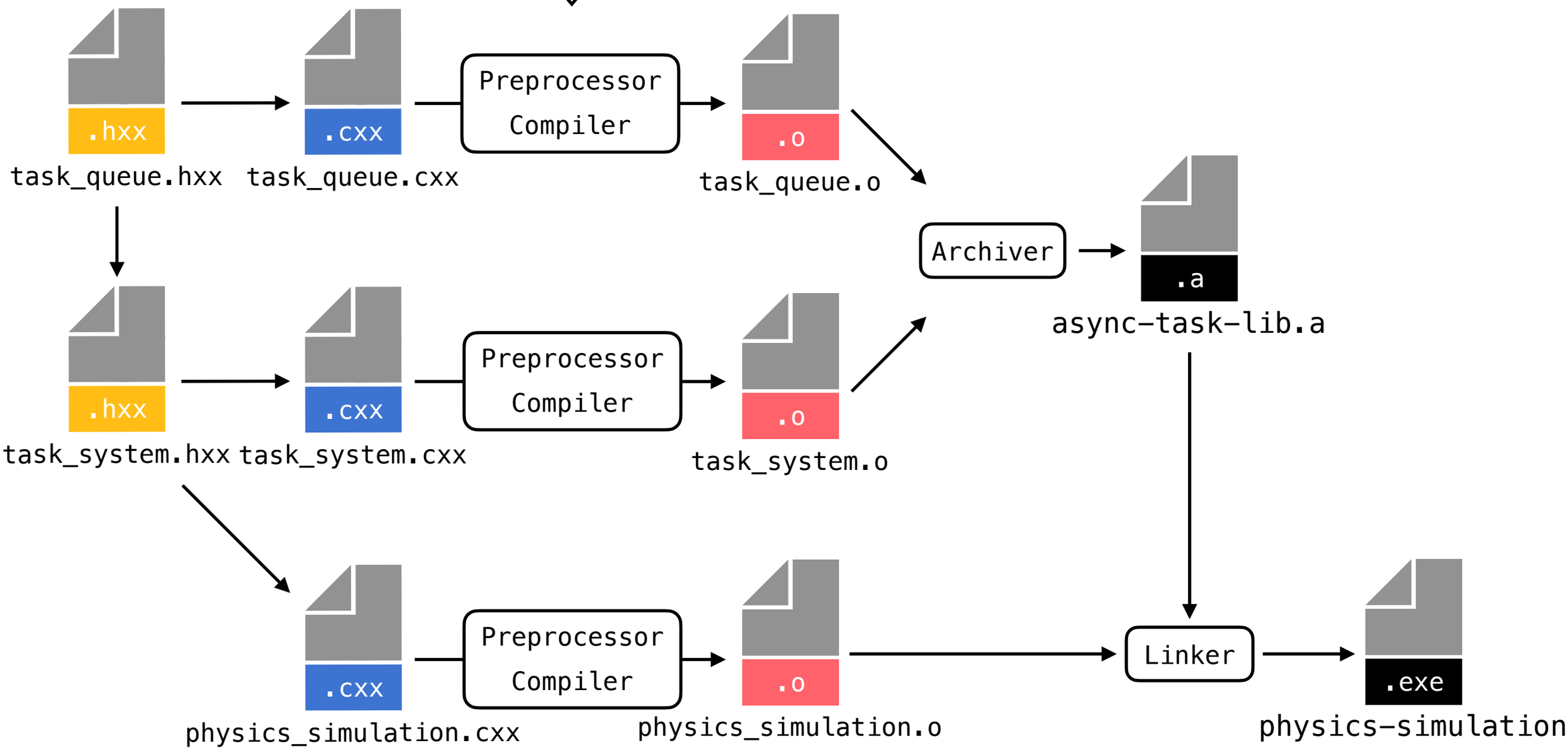
Build Workflow



Textual Inclusion Model

Build Workflow

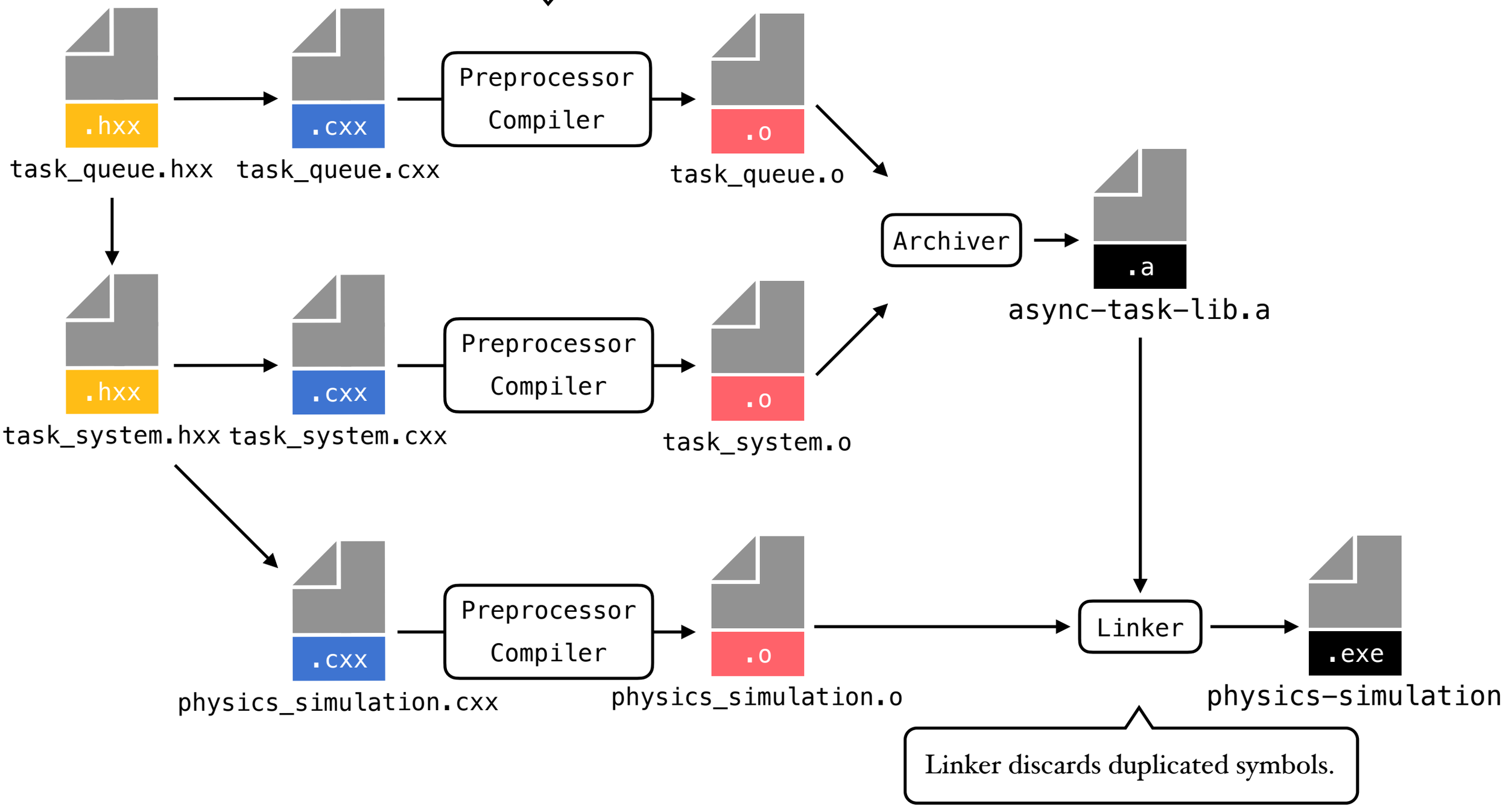
The textual inclusion of header files is very expensive,
because their content is compiled multiple times,
making the textual inclusion model incredibly inefficient and wasteful.



Textual Inclusion Model

Build Workflow

The textual inclusion of header files is very expensive,
because their content is compiled multiple times,
making the textual inclusion model incredibly inefficient and wasteful.



Textual Inclusion Model

Mangled Symbol Names

C++ entities	Linker symbols
`atl::task_queue::task_queue ();`	`__ZN3atl10task_queueC1Ev`
`atl::task_queue::~task_queue ();`	`__ZN3atl10task_queueD1Ev`
`atl::task_queue::pop ();`	`__ZN3atl10task_queue3popEv`
`atl::task_queue::finish ();`	`__ZN3atl10task_queue6finishEv`
`atl::task_system::task_system ();`	`__ZN3atl11task_systemC1Ev`
`atl::task_system::~task_system ();`	`__ZN3atl11task_systemD1Ev`
`phy_sim::length (const vec3&);`	`__ZN7phy_sim6lengthERKNS_4vec3E`
`phy_sim::compute_lengths (` `  const std::array<phy_sim::vec3, 128ul>&);`	`__ZN7phy_sim15compute_lengthsERKNSt3__15arrayINS_4vec3ELm128EEE`

Textual Inclusion Model

Mangled Symbol Names

C++ entities	Linker symbols
`atl::task_queue::task_queue ();`	`__ZN3atl10task_queueC1Ev`
`atl::task_queue::~task_queue ();`	`__ZN3atl10task_queueD1Ev`
`atl::task_queue::pop ();`	`__ZN3atl10task_queue3popEv`
`atl::task_queue::finish ();`	`__ZN3atl10task_queue6finishEv`
`atl::task_system::task_system ();`	`__ZN3atl11task_systemC1Ev`
`atl::task_system::~task_system ();`	`__ZN3atl11task_systemD1Ev`
`phy_sim::length (const vec3&);`	`__ZN7phy_sim6lengthERKNS_4vec3E`
`phy_sim::compute_lengths (` `  const std::array<phy_sim::vec3, 128ul>&);`	`__ZN7phy_sim15compute_lengthsERKNSt3__15arrayINS_4vec3ELm128EEE`

Linker symbol list is global for the whole C++ program,
which implies that each C++ entity can have only one definition.

Textual Inclusion Model

One Definition Rule

```
class task_queue
{
private:
    struct task
    {
        std::function<void ()>                  function;
#ifndef NDEBUG
        std::chrono::steady_clock::time_point arrival;
        std::chrono::steady_clock::time_point departure;
#endif
    };

    std::deque<task> tasks;
    ...
};
```

Textual Inclusion Model

One Definition Rule

```
class task_queue
{
private:
    struct task
    {
        std::function<void ()>                  function;
#ifndef NDEBUG
        std::chrono::steady_clock::time_point arrival;
        std::chrono::steady_clock::time_point departure;
#endif
    };

    std::deque<task> tasks;
    ...
};
```

Let's assume that the task queue in Debug mode reports how long, on average, a task waits for execution in the task queue, that is the task queue latency.

Textual Inclusion Model

One Definition Rule

```
class task_queue
{
private:
    struct task
    {
        std::function<void ()>                  function;
#ifndef NDEBUG
        std::chrono::steady_clock::time_point arrival;
        std::chrono::steady_clock::time_point departure;
#endif
    };

    std::deque<task> tasks;
    ...
};
```

Let's assume that the task queue in Debug mode reports how long, on average, a task waits for execution in the task queue, that is the task queue latency.

To implement this functionality, in Debug mode, each task will store its time of arrival and departure.

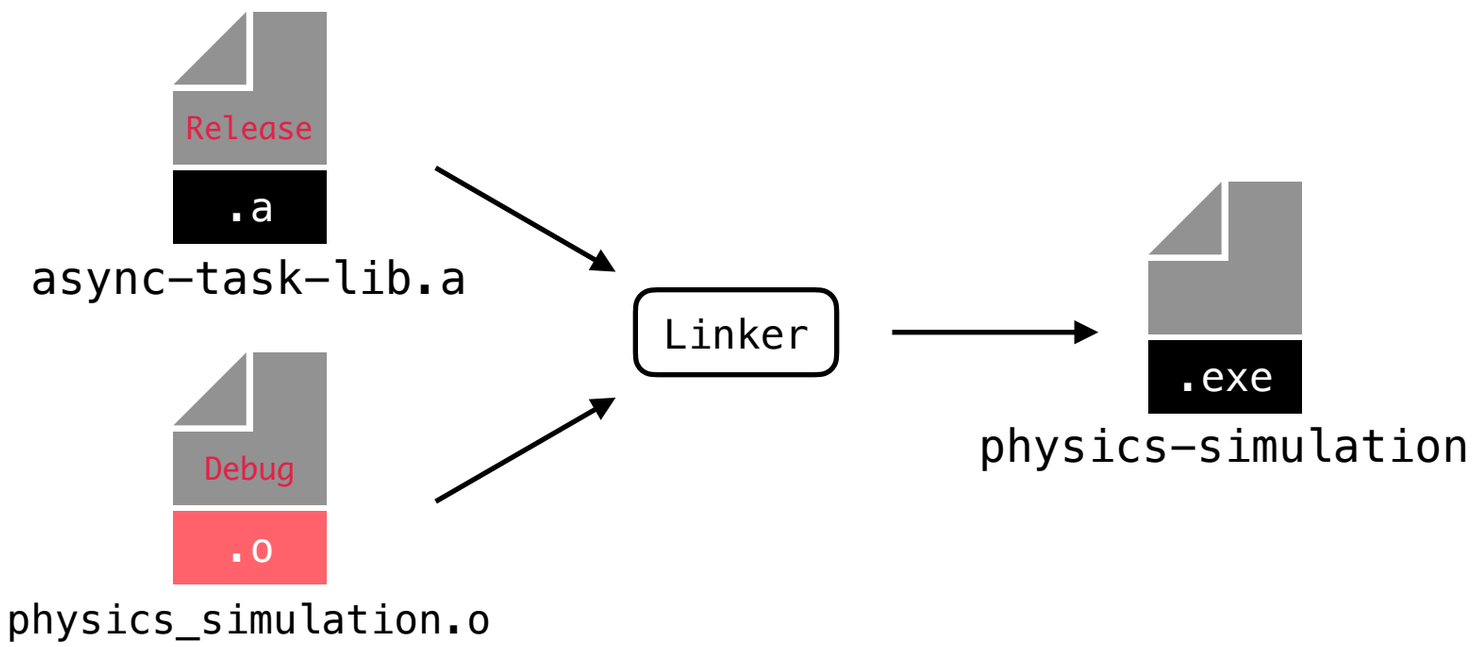
Textual Inclusion Model

One Definition Rule

```
class task_queue
{
private:
    struct task
    {
        std::function<void ()>                  function;
#ifndef NDEBUG
        std::chrono::steady_clock::time_point arrival;
        std::chrono::steady_clock::time_point departure;
#endif
    };

    std::deque<task> tasks;
    ...
};
```

Let's assume that the task queue in Debug mode reports how long, on average, a task waits for execution in the task queue, that is the task queue latency.

To implement this functionality, in Debug mode, each task will store its time of arrival and departure.



Textual Inclusion Model

One Definition Rule

```
class task_queue
{
private:
    struct task
    {
        std::function<void ()>                  function;
#ifndef NDEBUG
        std::chrono::steady_clock::time_point arrival;
        std::chrono::steady_clock::time_point departure;
#endif
    };

    std::deque<task> tasks;
    ...
};
```

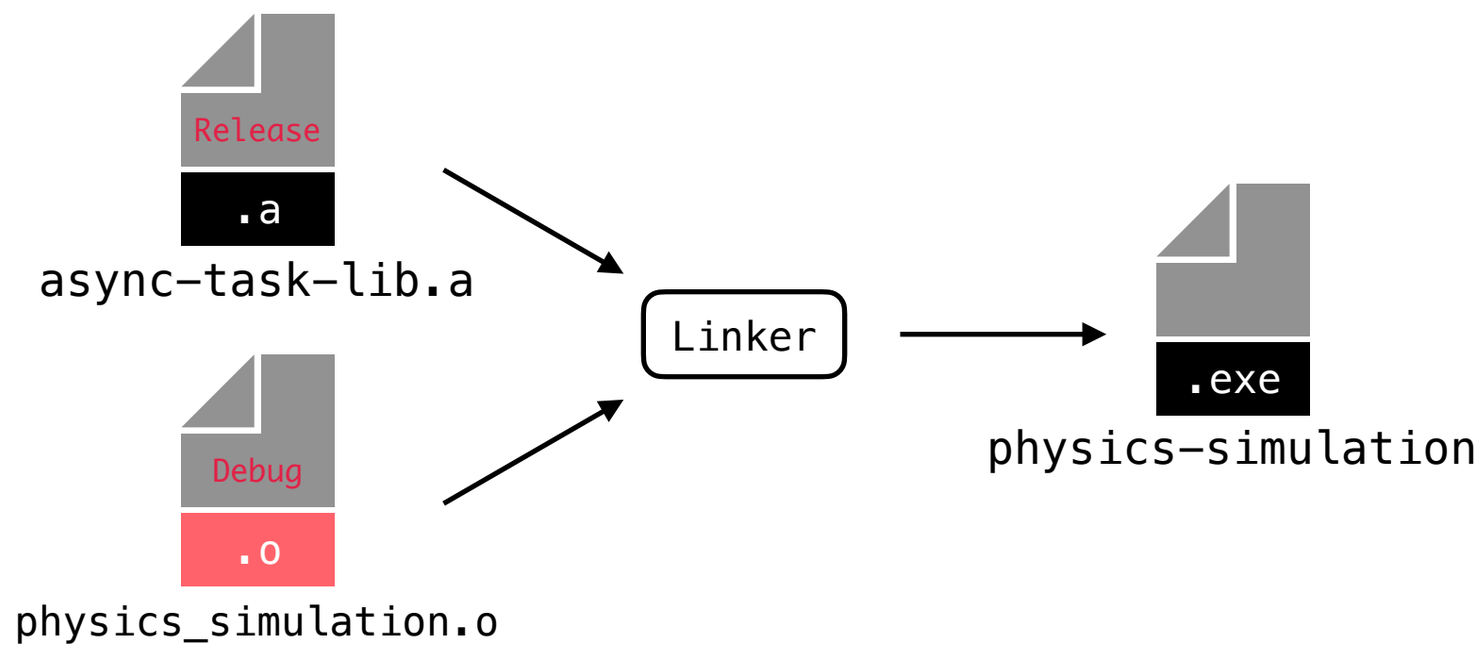
Let's assume that the task queue in Debug mode reports how long, on average, a task waits for execution in the task queue, that is the task queue latency.

To implement this functionality, in Debug mode, each task will store its time of arrival and departure.

Let's also assume that we are debugging an issue in the physics simulation.

To improve responsiveness of debugging we decided to use Release version of the Async Task Library.

The resulting program is **ill-formed**, because translation units will contain conflicting definitions of the task structure!



Textual Inclusion Model

One Definition Rule

```
class task_queue
{
private:
    struct task
    {
        std::function<void ()>                  function;
#ifndef NDEBUG
        std::chrono::steady_clock::time_point arrival;
        std::chrono::steady_clock::time_point departure;
#endif
    };

    std::deque<task> tasks;
    ...
};
```

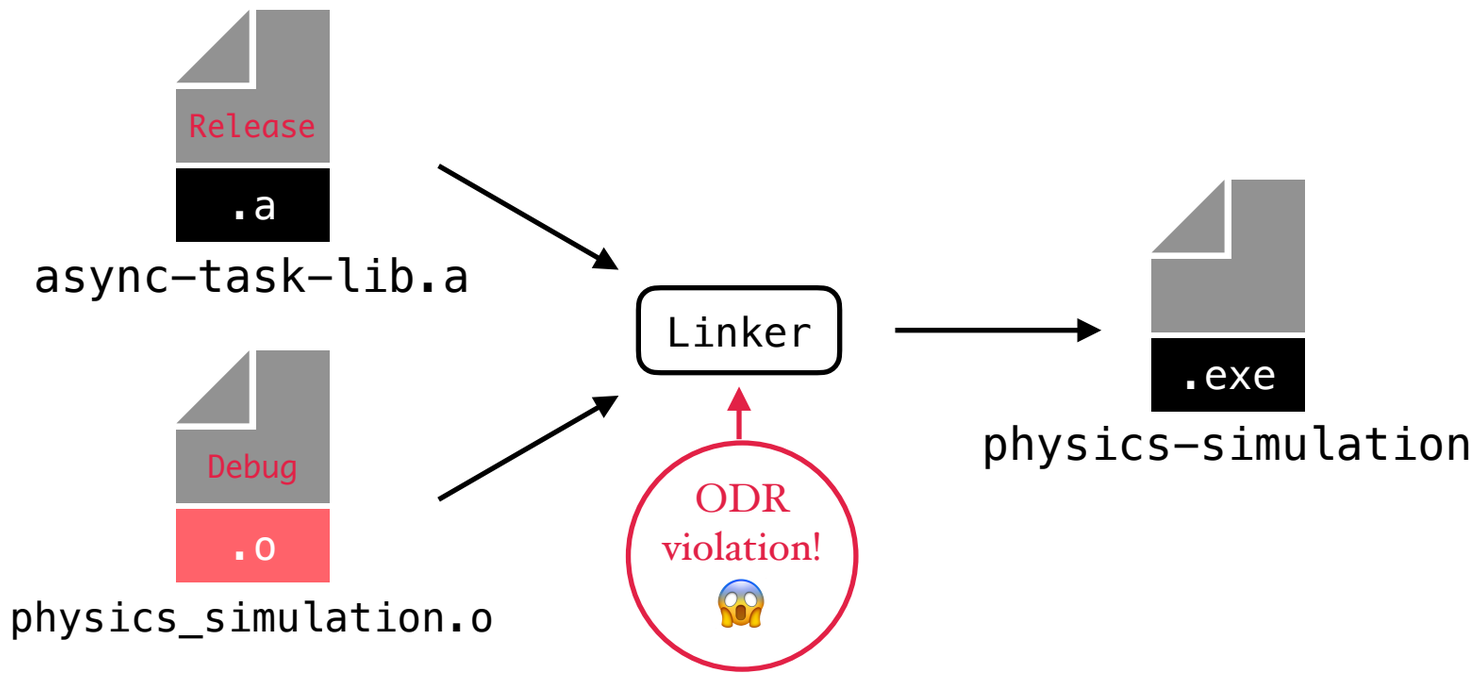
Let's assume that the task queue in Debug mode reports how long, on average, a task waits for execution in the task queue, that is the task queue latency.

To implement this functionality, in Debug mode, each task will store its time of arrival and departure.

Let's also assume that we are debugging an issue in the physics simulation.

To improve responsiveness of debugging we decided to use Release version of the Async Task Library.

The resulting program is **ill-formed**, because translation units will contain conflicting definitions of the task structure!

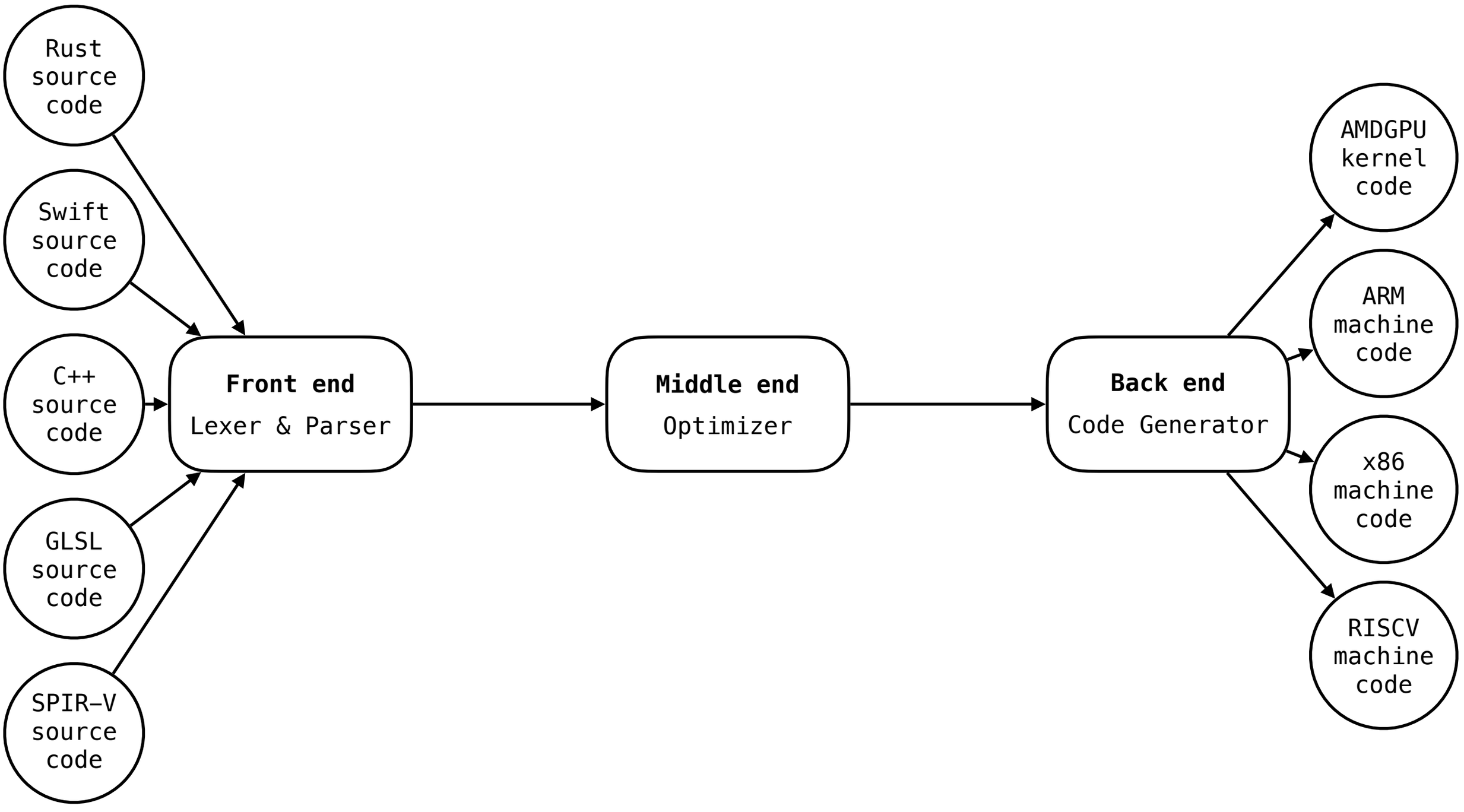


Compiler Architecture

Three stage compiler design

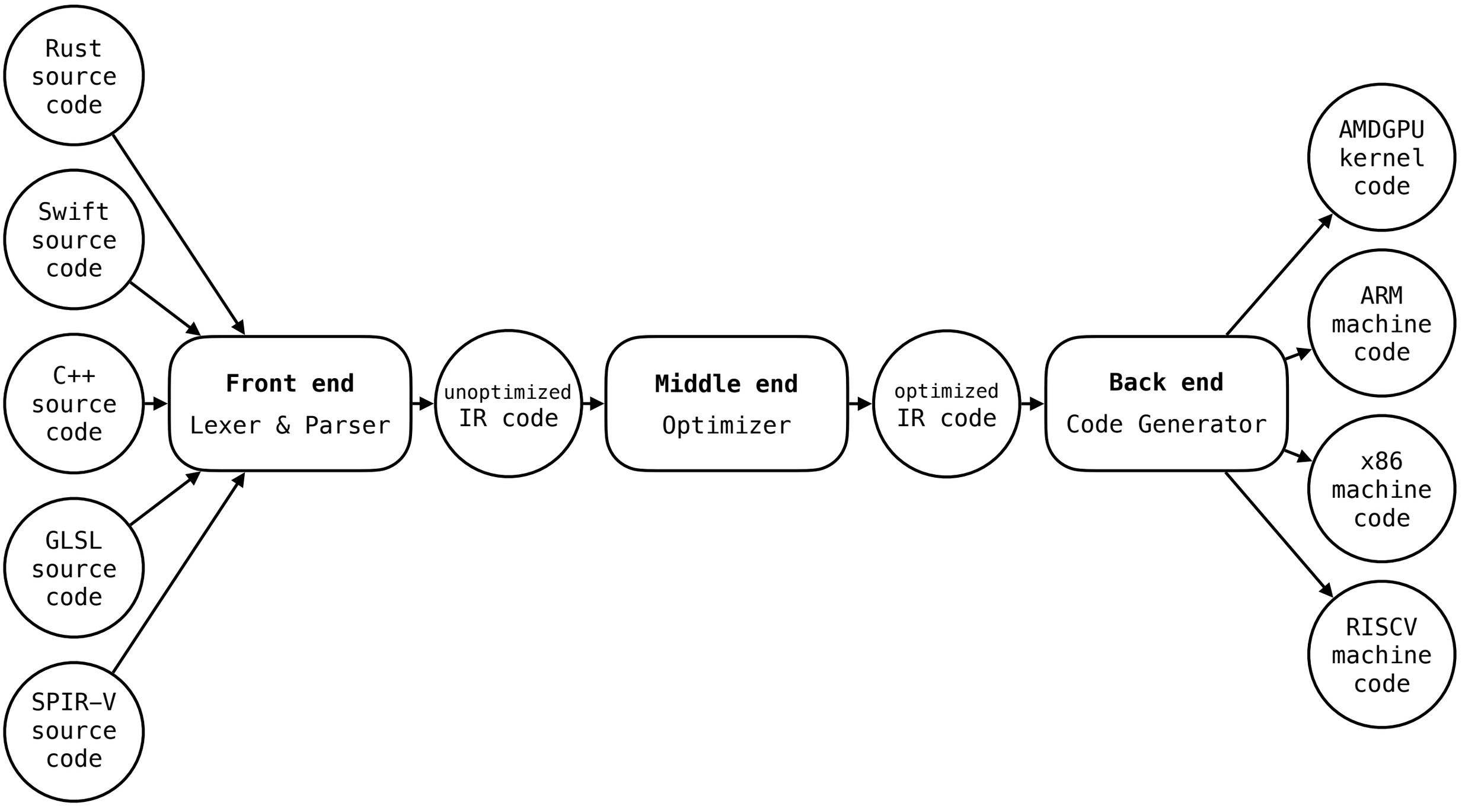
Compiler Architecture

Three stage compiler design



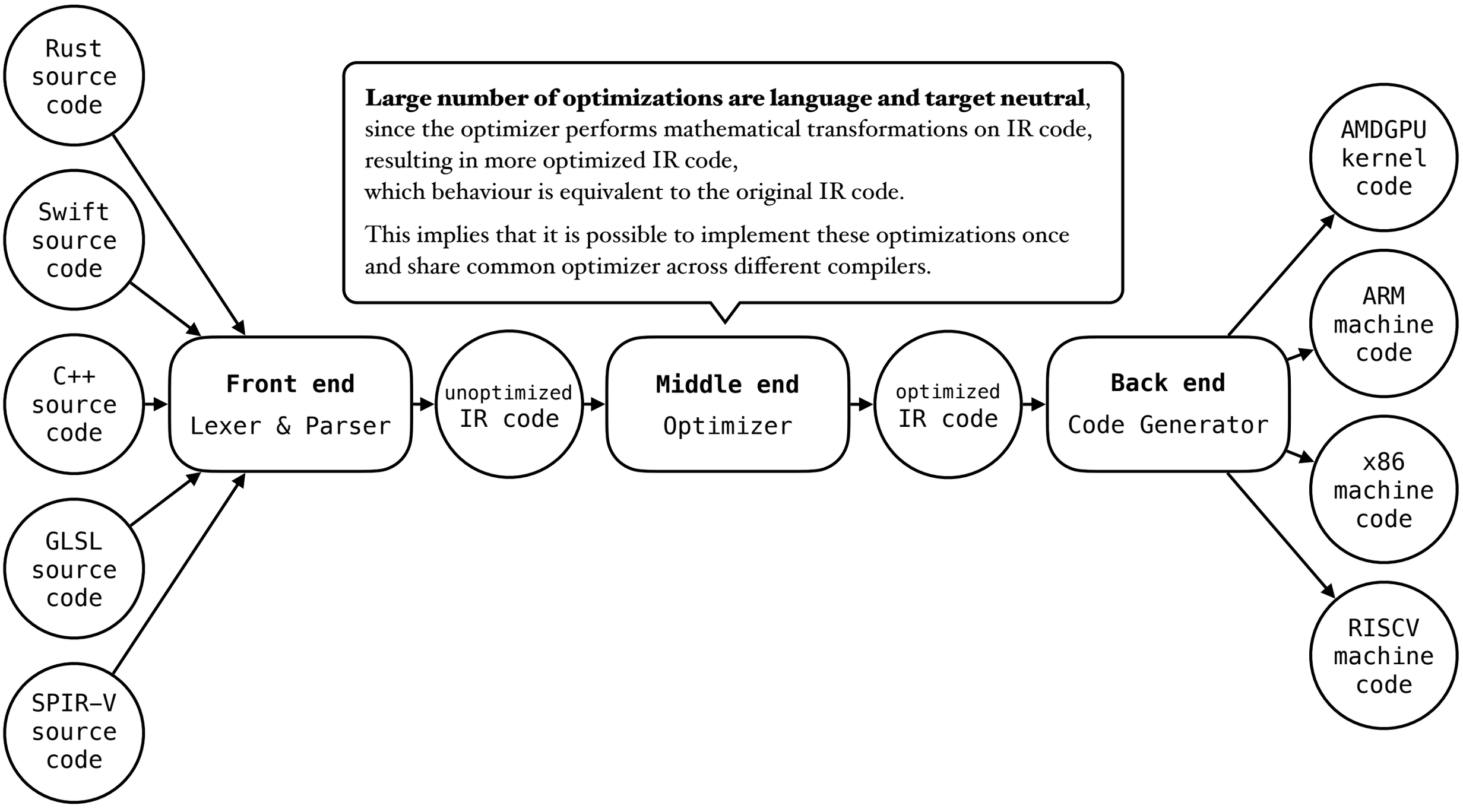
Compiler Architecture

Three stage compiler design



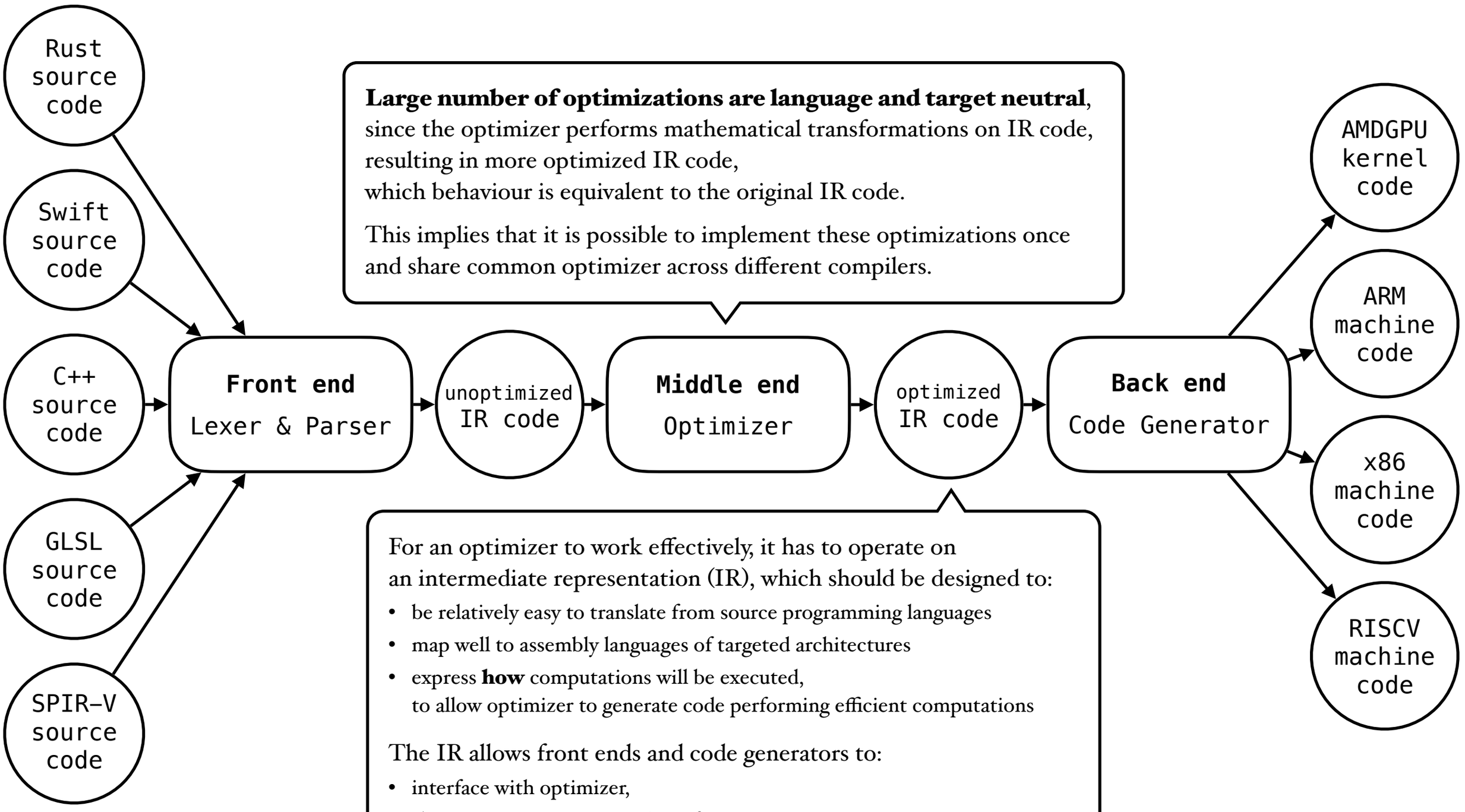
Compiler Architecture

Three stage compiler design



Compiler Architecture

Three stage compiler design



Large number of optimizations are language and target neutral, since the optimizer performs mathematical transformations on IR code, resulting in more optimized IR code,
which behaviour is equivalent to the original IR code.

This implies that it is possible to implement these optimizations once and share common optimizer across different compilers.

For an optimizer to work effectively, it has to operate on an intermediate representation (IR), which should be designed to:

- be relatively easy to translate from source programming languages
- map well to assembly languages of targeted architectures
- express **how** computations will be executed,
  to allow optimizer to generate code performing efficient computations

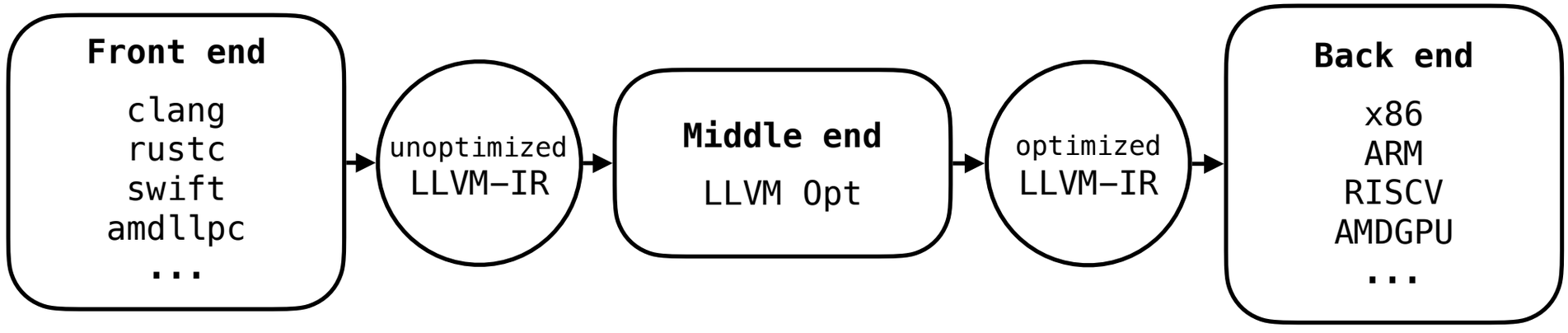
The IR allows front ends and code generators to:

- interface with optimizer,
- share common representation of computer programs,

enabling incredible code reuse between different compilers.

Compiler Architecture

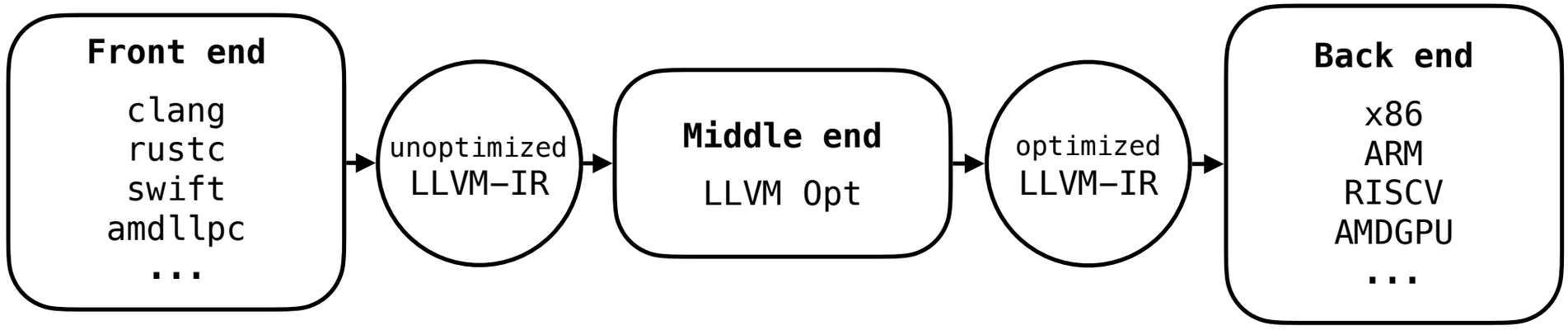
The LLVM Compiler Infrastructure & Clang Compiler



Compiler Architecture

The LLVM Compiler Infrastructure & Clang Compiler

Thriving ecosystem of LLVM-based compilers supporting numerous different programming languages.

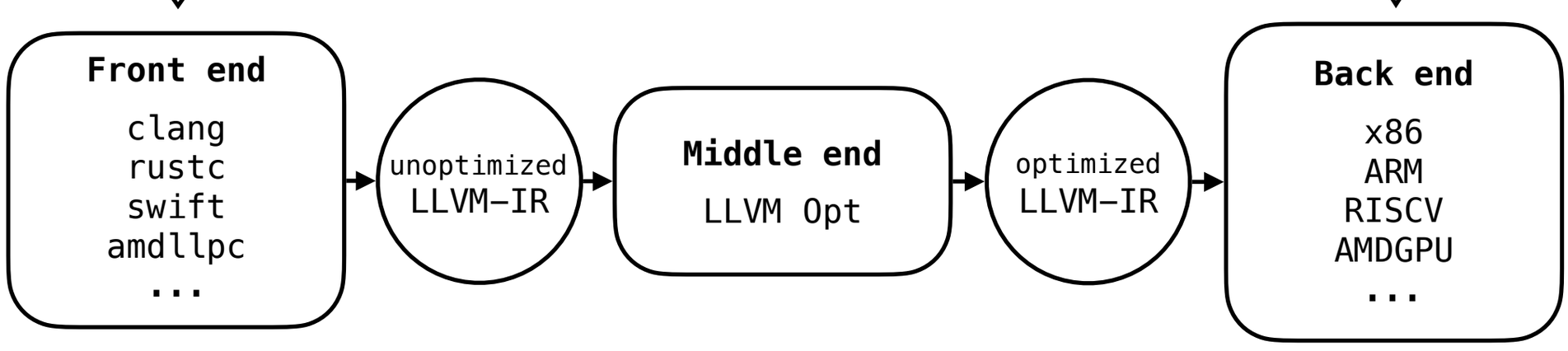


Compiler Architecture

The LLVM Compiler Infrastructure & Clang Compiler

Thriving ecosystem of LLVM-based compilers supporting numerous different programming languages.

Large number of supported architectures, including CPUs and GPUs.

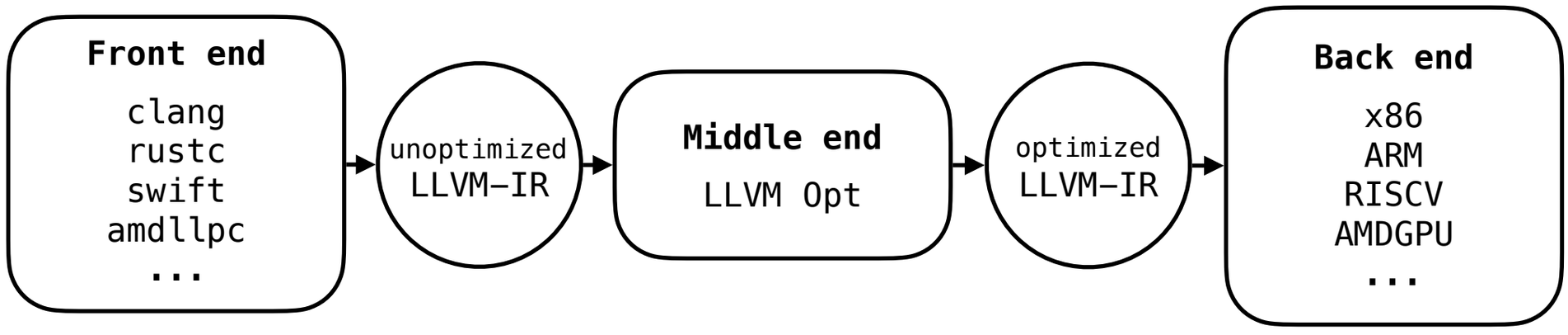


Compiler Architecture

The LLVM Compiler Infrastructure & Clang Compiler

Thriving ecosystem of LLVM-based compilers supporting numerous different programming languages.

Large number of supported architectures, including CPUs and GPUs.



I highly recommend reading the following:

- Architecture of Open Source Applications - LLVM by Chris Lattner
- My First Language Frontend with LLVM Tutorial

References:

- https://llvm.org
- https://github.com/llvm/llvm-project

Compiler Architecture

LLVM-IR

```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```

Compiler Architecture

LLVM-IR

```
$ clang++ -std=c++17 -emit-llvm                    \
-S //physics_simulation/source/vec3_length.cxx \
-o vec3_length.ll
```

```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```

Compiler Architecture

LLVM-IR

```
$ clang++ -std=c++17 -emit-llvm                    \
-S //physics_simulation/source/vec3_length.cxx \
-o vec3_length.ll
```

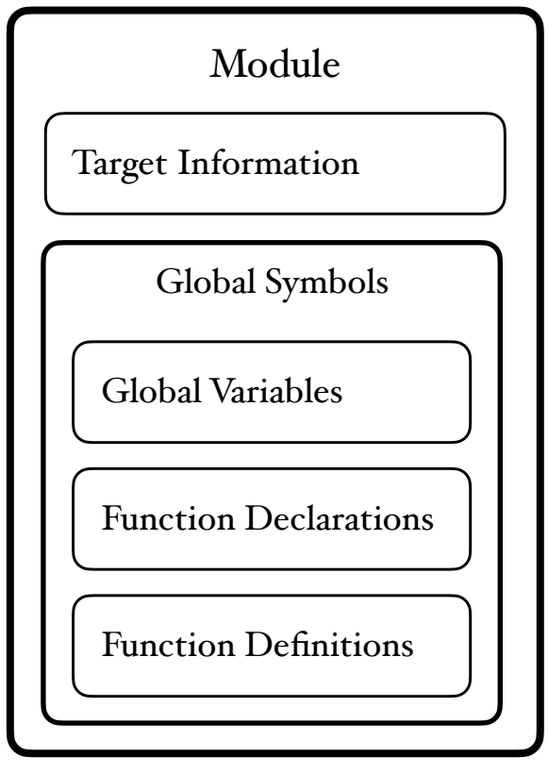
```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```

```
target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple     = "x86_64-apple-macosx10.14.0"

declare float @sqrt (float %x)

%vec3 = type { float, float, float }

define float @_ZN7phy_sim6lengthERKNS_4vec3E (%vec3* %v)
{
entry:
  %v.x_ptr      = getelementptr %vec3, %vec3* %v, i64 0, i32 0
  %v.x          = load float, float* %v.x_ptr
  %v.x2         = fmul float %v.x, %v.x

  %v.y_ptr      = getelementptr %vec3, %vec3* %v, i64 0, i32 1
  %v.yz_ptr     = bitcast float* %v.y_ptr to <2 x float>*
  %v.yz         = load <2 x float>, <2 x float>* %v.yz_ptr
  %v.y2z2       = fmul <2 x float> %v.yz, %v.yz

  %v.y2         = extractelement <2 x float> %v.y2z2, i32 0
  %v.z2         = extractelement <2 x float> %v.y2z2, i32 1

  %v.x2y2_sum   = fadd float %v.x2      , %v.y2
  %v.x2y2z2_sum = fadd float %v.x2y2_sum, %v.z2
  %v.length     = call float @sqrt(float %v.x2y2z2_sum)

  ret float %v.length
}
```

Compiler Architecture

LLVM-IR

```
$ clang++ -std=c++17 -emit-llvm                    \
-S //physics_simulation/source/vec3_length.cxx \
-o vec3_length.ll
```



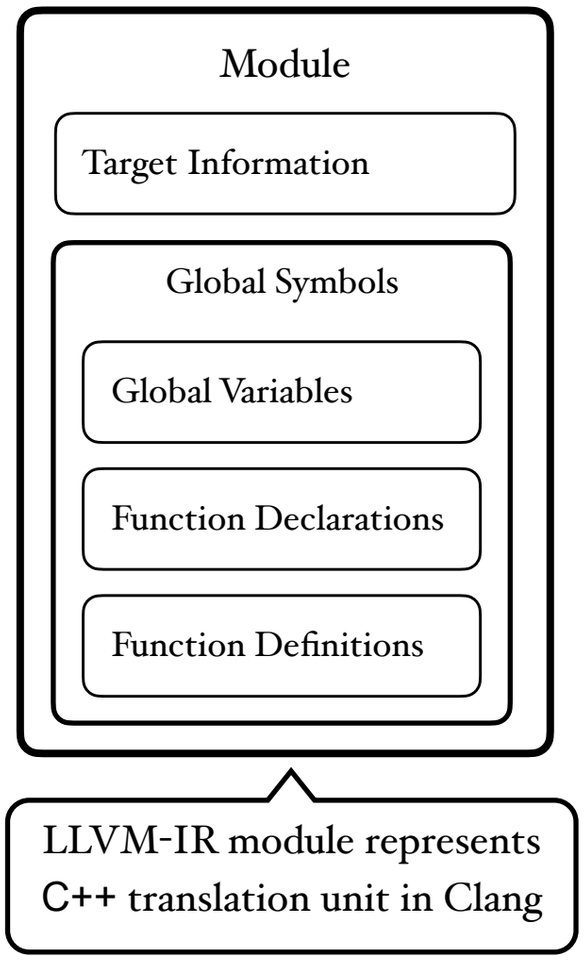

```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```

```
target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple     = "x86_64-apple-macosx10.14.0"

declare float @sqrt (float %x)

%vec3 = type { float, float, float }

define float @_ZN7phy_sim6lengthERKNS_4vec3E (%vec3* %v)
{
entry:
  %v.x_ptr      = getelementptr %vec3, %vec3* %v, i64 0, i32 0
  %v.x          = load float, float* %v.x_ptr
  %v.x2         = fmul float %v.x, %v.x

  %v.y_ptr      = getelementptr %vec3, %vec3* %v, i64 0, i32 1
  %v.yz_ptr     = bitcast float* %v.y_ptr to <2 x float>*
  %v.yz         = load <2 x float>, <2 x float>* %v.yz_ptr
  %v.y2z2       = fmul <2 x float> %v.yz, %v.yz

  %v.y2         = extractelement <2 x float> %v.y2z2, i32 0
  %v.z2         = extractelement <2 x float> %v.y2z2, i32 1

  %v.x2y2_sum   = fadd float %v.x2      , %v.y2
  %v.x2y2z2_sum = fadd float %v.x2y2_sum, %v.z2
  %v.length     = call float @sqrt(float %v.x2y2z2_sum)

  ret float %v.length
}
```

Compiler Architecture

LLVM-IR

```
$ clang++ -std=c++17 -emit-llvm                    \
-S //physics_simulation/source/vec3_length.cxx \
-o vec3_length.ll
```



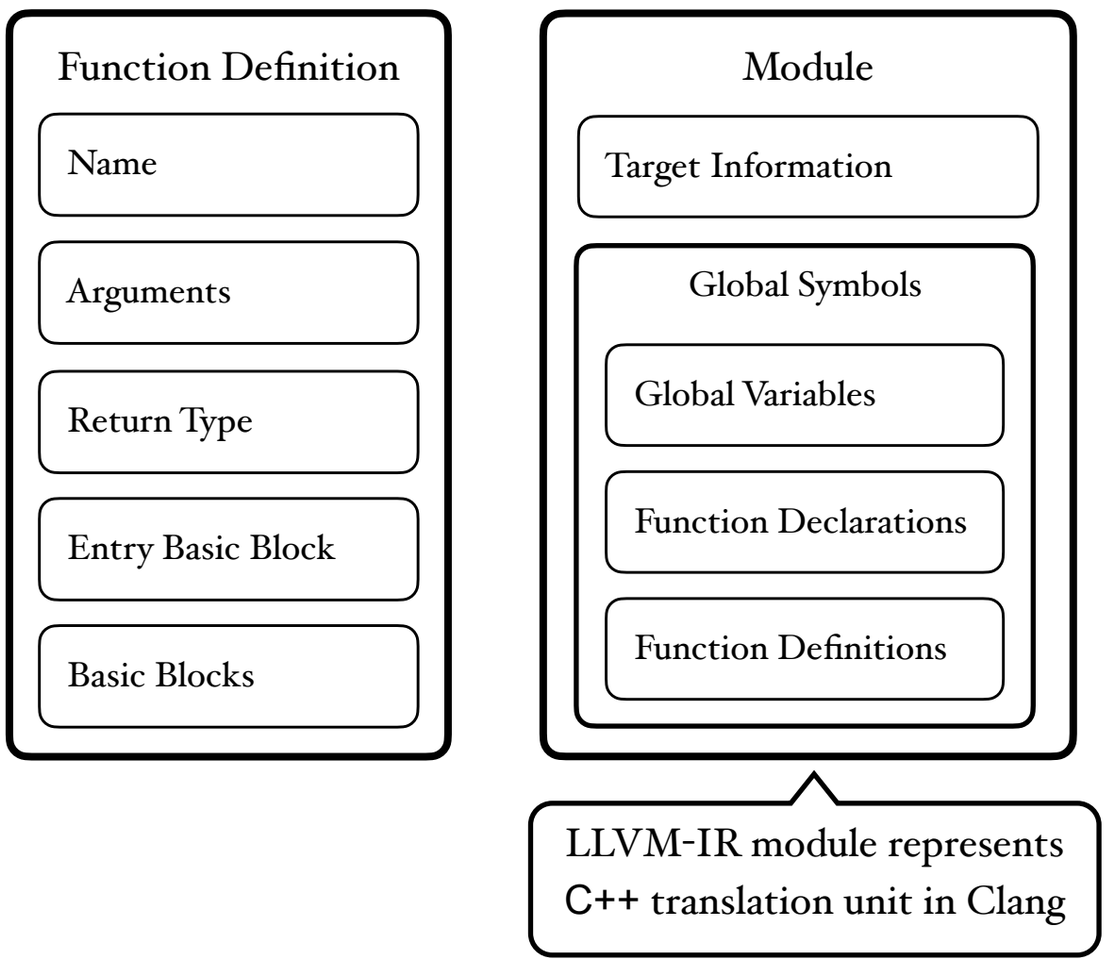

```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```

```
target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple     = "x86_64-apple-macosx10.14.0"

declare float @sqrt (float %x)

%vec3 = type { float, float, float }

define float @_ZN7phy_sim6lengthERKNS_4vec3E (%vec3* %v)
{
entry:
  %v.x_ptr       = getelementptr %vec3, %vec3* %v, i64 0, i32 0
  %v.x           = load float, float* %v.x_ptr
  %v.x2          = fmul float %v.x, %v.x

  %v.y_ptr       = getelementptr %vec3, %vec3* %v, i64 0, i32 1
  %v.yz_ptr      = bitcast float* %v.y_ptr to <2 x float>*
  %v.yz          = load <2 x float>, <2 x float>* %v.yz_ptr
  %v.y2z2        = fmul <2 x float> %v.yz, %v.yz

  %v.y2          = extractelement <2 x float> %v.y2z2, i32 0
  %v.z2          = extractelement <2 x float> %v.y2z2, i32 1

  %v.x2y2_sum    = fadd float %v.x2       , %v.y2
  %v.x2y2z2_sum  = fadd float %v.x2y2_sum, %v.z2
  %v.length      = call float @sqrt(float %v.x2y2z2_sum)

  ret float %v.length
}
```

Compiler Architecture

LLVM-IR

```
$ clang++ -std=c++17 -emit-llvm                    \
-S //physics_simulation/source/vec3_length.cxx \
-o vec3_length.ll
```



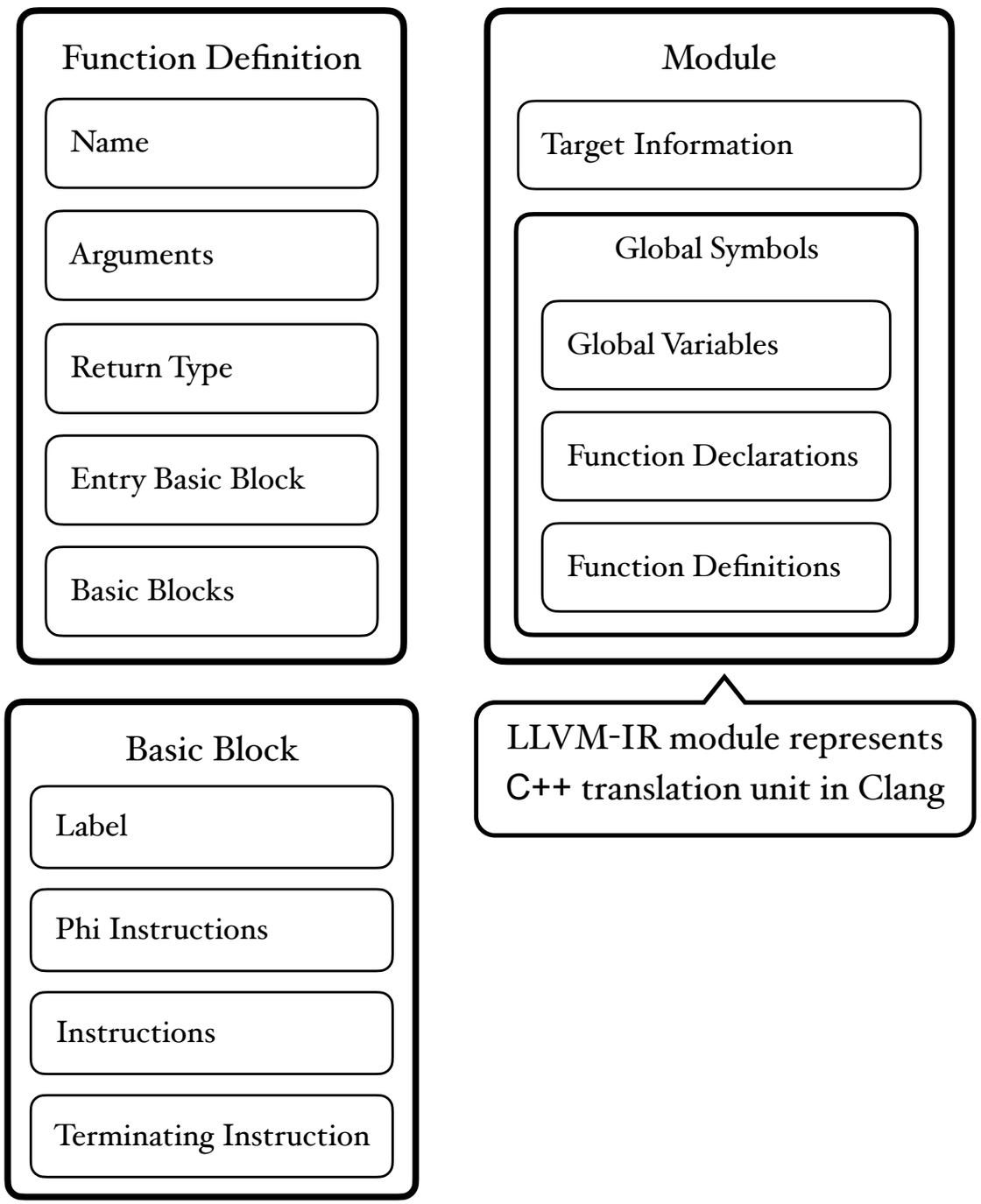

```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```

```
target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple     = "x86_64-apple-macosx10.14.0"

declare float @sqrt (float %x)

%vec3 = type { float, float, float }

define float @_ZN7phy_sim6lengthERKNS_4vec3E (%vec3* %v)
{
entry:
  %v.x_ptr       = getelementptr %vec3, %vec3* %v, i64 0, i32 0
  %v.x           = load float, float* %v.x_ptr
  %v.x2          = fmul float %v.x, %v.x

  %v.y_ptr       = getelementptr %vec3, %vec3* %v, i64 0, i32 1
  %v.yz_ptr      = bitcast float* %v.y_ptr to <2 x float>*
  %v.yz          = load <2 x float>, <2 x float>* %v.yz_ptr
  %v.y2z2        = fmul <2 x float> %v.yz, %v.yz

  %v.y2          = extractelement <2 x float> %v.y2z2, i32 0
  %v.z2          = extractelement <2 x float> %v.y2z2, i32 1

  %v.x2y2_sum    = fadd float %v.x2       , %v.y2
  %v.x2y2z2_sum  = fadd float %v.x2y2_sum, %v.z2
  %v.length      = call float @sqrt(float %v.x2y2z2_sum)

  ret float %v.length
}
```

Compiler Architecture

LLVM-IR

```
$ clang++ -std=c++17 -emit-llvm                    \
-S //physics_simulation/source/vec3_length.cxx \
-o vec3_length.ll
```



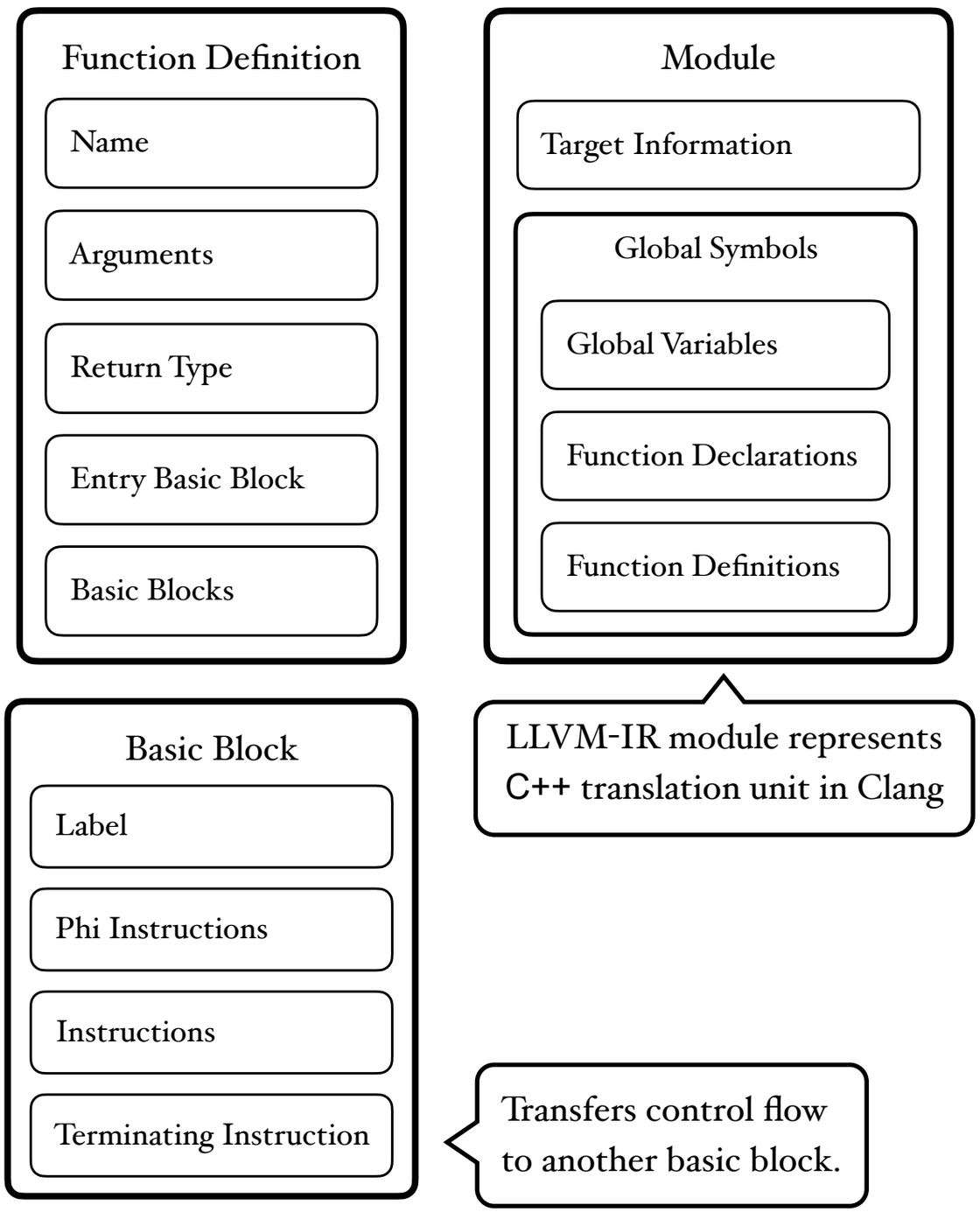

```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```

```
target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple     = "x86_64-apple-macosx10.14.0"

declare float @sqrt (float %x)

%vec3 = type { float, float, float }

define float @_ZN7phy_sim6lengthERKNS_4vec3E (%vec3* %v)
{
entry:
  %v.x_ptr       = getelementptr %vec3, %vec3* %v, i64 0, i32 0
  %v.x           = load float, float* %v.x_ptr
  %v.x2          = fmul float %v.x, %v.x

  %v.y_ptr       = getelementptr %vec3, %vec3* %v, i64 0, i32 1
  %v.yz_ptr      = bitcast float* %v.y_ptr to <2 x float>*
  %v.yz          = load <2 x float>, <2 x float>* %v.yz_ptr
  %v.y2z2        = fmul <2 x float> %v.yz, %v.yz

  %v.y2          = extractelement <2 x float> %v.y2z2, i32 0
  %v.z2          = extractelement <2 x float> %v.y2z2, i32 1

  %v.x2y2_sum    = fadd float %v.x2       , %v.y2
  %v.x2y2z2_sum  = fadd float %v.x2y2_sum, %v.z2
  %v.length      = call float @sqrt(float %v.x2y2z2_sum)

  ret float %v.length
}
```

Compiler Architecture

LLVM-IR

```
$ clang++ -std=c++17 -emit-llvm                    \
-S //physics_simulation/source/vec3_length.cxx \
-o vec3_length.ll
```




```cpp
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```

```
target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple     = "x86_64-apple-macosx10.14.0"

declare float @sqrt (float %x)

%vec3 = type { float, float, float }

define float @_ZN7phy_sim6lengthERKNS_4vec3E (%vec3* %v)
{
entry:
  %v.x_ptr       = getelementptr %vec3, %vec3* %v, i64 0, i32 0
  %v.x           = load float, float* %v.x_ptr
  %v.x2          = fmul float %v.x, %v.x

  %v.y_ptr       = getelementptr %vec3, %vec3* %v, i64 0, i32 1
  %v.yz_ptr      = bitcast float* %v.y_ptr to <2 x float>*
  %v.yz          = load <2 x float>, <2 x float>* %v.yz_ptr
  %v.y2z2        = fmul <2 x float> %v.yz, %v.yz

  %v.y2          = extractelement <2 x float> %v.y2z2, i32 0
  %v.z2          = extractelement <2 x float> %v.y2z2, i32 1

  %v.x2y2_sum    = fadd float %v.x2       , %v.y2
  %v.x2y2z2_sum  = fadd float %v.x2y2_sum, %v.z2
  %v.length      = call float @sqrt(float %v.x2y2z2_sum)

  ret float %v.length
}
```

Compiler Architecture

LLVM-IR

```c
#include <stdio.h>

int main ()
{
    int c;

    while ((c = getchar()) != EOF)
    {
        putchar(c);
    }
}
```

Compiler Architecture

LLVM-IR

```
; copy_file.ll

target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple     = "x86_64-apple-macosx10.14.0"

declare i32 @getchar ()

declare i32 @putchar (i32)

define  i32 @main ()
{
entry:
  %first_char         = call    i32 @getchar()
  %first_char_eq_eof = icmp eq i32 %first_char, -1;EOF

  br i1 %first_char_eq_eof, label %exit, label %loop

loop:
  %out_char           = phi     i32 [  %next_char, %loop  ],
                                    [ %first_char, %entry ]
                        call    i32 @putchar(i32 %out_char)

  %next_char          = call    i32 @getchar()
  %next_char_eq_eof   = icmp eq i32 %next_char, -1;EOF

  br i1 %next_char_eq_eof, label %exit, label %loop

exit:
  ret i32 0
}
```

```
#include <stdio.h>

int main ()
{
    int c;

    while ((c = getchar()) != EOF)
    {
        putchar(c);
    }
}
```

Compiler Architecture

LLVM-IR

LLVM-IR has strong static typing,
supporting floating point and integer types,
as well as pointers, functions, structures, arrays and vectors.

```
; copy_file.ll

target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple     = "x86_64-apple-macosx10.14.0"

declare i32 @getchar ()

declare i32 @putchar (i32)

define  i32 @main ()
{
entry:
  %first_char         = call    i32 @getchar()
  %first_char_eq_eof = icmp eq i32 %first_char, -1;EOF

  br i1 %first_char_eq_eof, label %exit, label %loop

loop:
  %out_char           = phi     i32 [  %next_char, %loop  ],
                                    [ %first_char, %entry ]
                        call    i32 @putchar(i32 %out_char)

  %next_char          = call    i32 @getchar()
  %next_char_eq_eof  = icmp eq i32 %next_char, -1;EOF

  br i1 %next_char_eq_eof, label %exit, label %loop

exit:
  ret i32 0
}
```

```
#include <stdio.h>

int main ()
{
    int c;

    while ((c = getchar()) != EOF)
    {
        putchar(c);
    }
}
```

Compiler Architecture

LLVM-IR

LLVM-IR has strong static typing,
supporting floating point and integer types,
as well as pointers, functions, structures, arrays and vectors.

```
#include <stdio.h>

int main ()
{
    int c;

    while ((c = getchar()) != EOF)
    {
        putchar(c);
    }
}
```

```
; copy_file.ll

target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple     = "x86_64-apple-macosx10.14.0"

declare i32 @getchar ()

declare i32 @putchar (i32)

define  i32 @main ()
{
entry:
  %first_char         = call    i32 @getchar()
  %first_char_eq_eof  = icmp eq i32 %first_char, -1;EOF

  br i1 %first_char_eq_eof, label %exit, label %loop

loop:
  %out_char           = phi     i32 [  %next_char, %loop  ],
                                    [ %first_char, %entry ]
                        call    i32 @putchar(i32 %out_char)

  %next_char          = call    i32 @getchar()
  %next_char_eq_eof   = icmp eq i32 %next_char, -1;EOF

  br i1 %next_char_eq_eof, label %exit, label %loop

exit:
  ret i32 0
}
```

LLVM-IR code has **Static Single Assignment** (SSA) form,
which means that once value is returned from an instruction
and assigned to a virtual register it must never change.

Compiler Architecture

LLVM-IR

LLVM-IR has strong static typing,
supporting floating point and integer types,
as well as pointers, functions, structures, arrays and vectors.

```
; copy_file.ll

target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple     = "x86_64-apple-macosx10.14.0"

declare i32 @getchar ()

declare i32 @putchar (i32)

define  i32 @main ()
{
entry:
  %first_char         = call    i32 @getchar()
  %first_char_eq_eof  = icmp eq i32 %first_char, -1;EOF

  br i1 %first_char_eq_eof, label %exit, label %loop

loop:
  %out_char           = phi     i32 [  %next_char, %loop  ],
                                    [ %first_char, %entry ]
                        call    i32 @putchar(i32 %out_char)

  %next_char          = call    i32 @getchar()
  %next_char_eq_eof   = icmp eq i32 %next_char, -1;EOF

  br i1 %next_char_eq_eof, label %exit, label %loop

exit:
  ret i32 0
}
```

LLVM-IR code has **Static Single Assignment** (SSA) form,
which means that once value is returned from an instruction
and assigned to a virtual register it must never change.

Phi node selects value from
one of predeceasing basic blocks,
depending on control flow.

```
#include <stdio.h>

int main ()
{
    int c;

    while ((c = getchar()) != EOF)
    {
        putchar(c);
    }
}
```

Compiler Architecture

LLVM-IR

LLVM-IR has strong static typing,
supporting floating point and integer types,
as well as pointers, functions, structures, arrays and vectors.

```
; copy_file.ll

target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple     = "x86_64-apple-macosx10.14.0"

declare i32 @getchar ()

declare i32 @putchar (i32)

define  i32 @main ()
{
entry:
  %first_char         = call    i32 @getchar()
  %first_char_eq_eof  = icmp eq i32 %first_char, -1;EOF

  br i1 %first_char_eq_eof, label %exit, label %loop

loop:
  %out_char           = phi     i32 [  %next_char, %loop  ],
                                    [ %first_char, %entry ]
                        call    i32 @putchar(i32 %out_char)

  %next_char          = call    i32 @getchar()
  %next_char_eq_eof   = icmp eq i32 %next_char, -1;EOF

  br i1 %next_char_eq_eof, label %exit, label %loop

exit:
  ret i32 0
}
```

LLVM-IR code has **Static Single Assignment** (SSA) form,
which means that once value is returned from an instruction
and assigned to a virtual register it must never change.

Phi node selects value from
one of predeceasing basic blocks,
depending on control flow.

Each LLVM-IR function defines a **Control Flow Graph** (CFG),
in which vertices are basic blocks, from which that function consists of,
and edges are control flow transfers from one basic block to another.

```
#include <stdio.h>

int main ()
{
    int c;

    while ((c = getchar()) != EOF)
    {
        putchar(c);
    }
}
```

Compiler Architecture

LLVM-IR

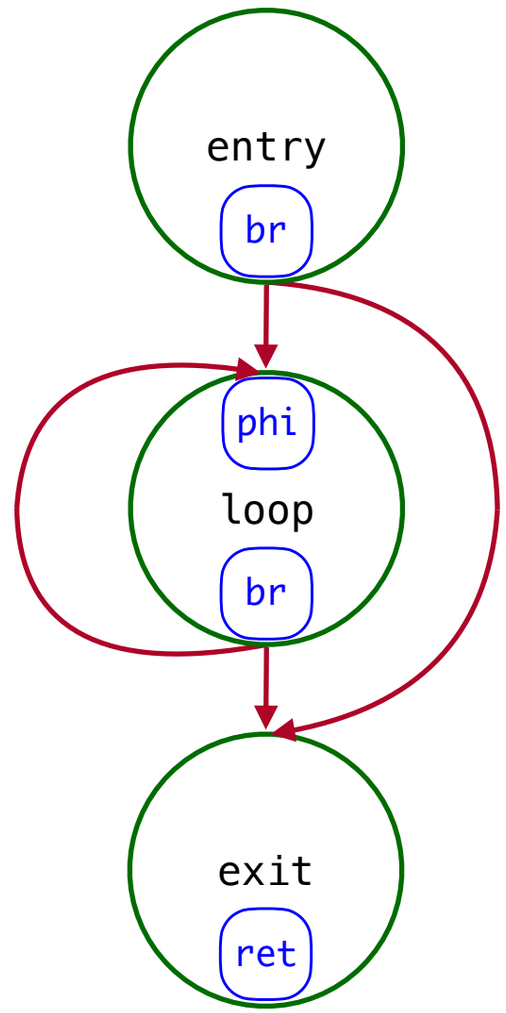
LLVM-IR has strong static typing,
supporting floating point and integer types,
as well as pointers, functions, structures, arrays and vectors.

```
; copy_file.ll

target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple     = "x86_64-apple-macosx10.14.0"

declare i32 @getchar ()

declare i32 @putchar (i32)

define  i32 @main ()
{
entry:
  %first_char         = call    i32 @getchar()
  %first_char_eq_eof  = icmp eq i32 %first_char, -1;EOF

  br i1 %first_char_eq_eof, label %exit, label %loop

loop:
  %out_char           = phi     i32 [  %next_char, %loop  ],
                                    [ %first_char, %entry ]
                        call    i32 @putchar(i32 %out_char)

  %next_char          = call    i32 @getchar()
  %next_char_eq_eof   = icmp eq i32 %next_char, -1;EOF

  br i1 %next_char_eq_eof, label %exit, label %loop

exit:
  ret i32 0
}
```

LLVM-IR code has **Static Single Assignment** (SSA) form,
which means that once value is returned from an instruction
and assigned to a virtual register it must never change.

Phi node selects value from
one of predeceasing basic blocks,
depending on control flow.

```
#include <stdio.h>

int main ()
{
    int c;

    while ((c = getchar()) != EOF)
    {
        putchar(c);
    }
}
```



Each LLVM-IR function defines a **Control Flow Graph** (CFG),
in which vertices are basic blocks, from which that function consists of,
and edges are control flow transfers from one basic block to another.

Compiler Architecture

LLVM-IR

LLVM-IR has strong static typing,
supporting floating point and integer types,
as well as pointers, functions, structures, arrays and vectors.

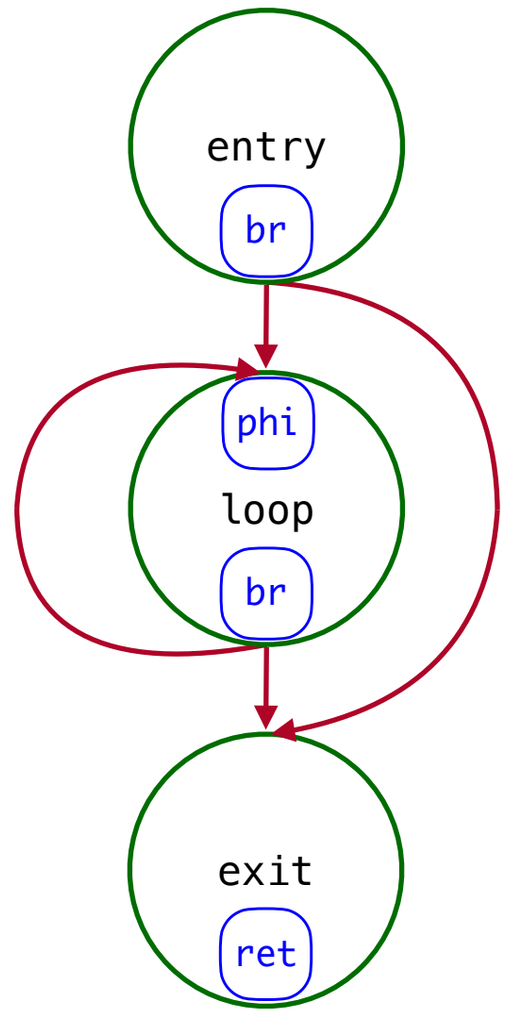
```
; copy_file.ll

target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple     = "x86_64-apple-macosx10.14.0"

declare i32 @getchar ()

declare i32 @putchar (i32)

define  i32 @main ()
{
entry:
  %first_char         = call    i32 @getchar()
  %first_char_eq_eof  = icmp eq i32 %first_char, -1;EOF

  br i1 %first_char_eq_eof, label %exit, label %loop

loop:
  %out_char           = phi     i32 [  %next_char, %loop  ],
                                    [ %first_char, %entry ]
                        call    i32 @putchar(i32 %out_char)

  %next_char          = call    i32 @getchar()
  %next_char_eq_eof   = icmp eq i32 %next_char, -1;EOF

  br i1 %next_char_eq_eof, label %exit, label %loop

exit:
  ret i32 0
}
```

LLVM-IR code has **Static Single Assignment** (SSA) form,
which means that once value is returned from an instruction
and assigned to a virtual register it must never change.

Phi node selects value from
one of predeceasing basic blocks,
depending on control flow.

```
$ llc copy_file.ll -o copy_file.s
$ as  copy_file.s  -o copy_file.o
```

```
#include <stdio.h>

int main ()
{
    int c;

    while ((c = getchar()) != EOF)
    {
        putchar(c);
    }
}
```



Each LLVM-IR function defines a **Control Flow Graph** (CFG),
in which vertices are basic blocks, from which that function consists of,
and edges are control flow transfers from one basic block to another.

Compiler Architecture

Representation of LLVM-IR

LLVM-IR has 3 equivalent representations:

1. **in-memory representation**
   C++ objects representing LLVM-IR code implemented in LLVM Libraries
2. **on-disk representation**
   stream of bits representing LLVM-IR code encoded according to LLVM Bitcode Format
3. **human readable assembly**
   stream of characters representing LLVM-IR code interpreted according to the LLVM Language Reference

Compiler Architecture

Representation of LLVM-IR

LLVM-IR has 3 equivalent representations:

1. **in-memory representation**
   C++ objects representing LLVM-IR code implemented in LLVM Libraries
2. **on-disk representation**
   stream of bits representing LLVM-IR code encoded according to LLVM Bitcode Format
3. **human readable assembly**
   stream of characters representing LLVM-IR code interpreted according to the LLVM Language Reference

```
// llvm.IR
llvm::Module
llvm::GlobalVariable
llvm::Function
llvm::BasicBlock
llvm::Instruction
llvm::PHINode
llvm::CallInst
llvm::AllocaInst
llvm::LoadInst
llvm::StoreInst
llvm::CmpInst
llvm::ConstantInt
llvm::GetElementPtrInst
llvm::BitCastInst
llvm::ExtractElementInst
llvm::BinaryOperator
llvm::BranchInst
llvm::ReturnInst
```

Compiler Architecture

Representation of LLVM-IR

LLVM-IR has 3 equivalent representations:

1. **in-memory representation**
   C++ objects representing LLVM-IR code implemented in LLVM Libraries
2. **on-disk representation**
   stream of bits representing LLVM-IR code encoded according to LLVM Bitcode Format
3. **human readable assembly**
   stream of characters representing LLVM-IR code interpreted according to the LLVM Language Reference

```
// llvm.IR
llvm::Module
llvm::GlobalVariable
llvm::Function
llvm::BasicBlock
llvm::Instruction
llvm::PHINode
llvm::CallInst
llvm::AllocaInst
llvm::LoadInst
llvm::StoreInst
llvm::CmpInst
llvm::ConstantInt
llvm::GetElementPtrInst
llvm::BitCastInst
llvm::ExtractElementInst
llvm::BinaryOperator
llvm::BranchInst
llvm::ReturnInst
```

bitcode.bc

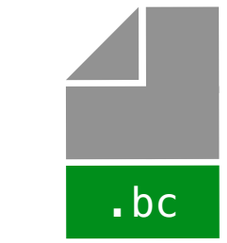
Compiler Architecture

Representation of LLVM-IR

LLVM-IR has 3 equivalent representations:

1. **in-memory representation**
   C++ objects representing LLVM-IR code implemented in LLVM Libraries
2. **on-disk representation**
   stream of bits representing LLVM-IR code encoded according to LLVM Bitcode Format
3. **human readable assembly**
   stream of characters representing LLVM-IR code interpreted according to the LLVM Language Reference

```
// llvm.IR
llvm::Module
llvm::GlobalVariable
llvm::Function
llvm::BasicBlock
llvm::Instruction
llvm::PHINode
llvm::CallInst
llvm::AllocaInst
llvm::LoadInst
llvm::StoreInst
llvm::CmpInst
llvm::ConstantInt
llvm::GetElementPtrInst
llvm::BitCastInst
llvm::ExtractElementInst
llvm::BinaryOperator
llvm::BranchInst
llvm::ReturnInst
```



bitcode.bc

assembly.ll

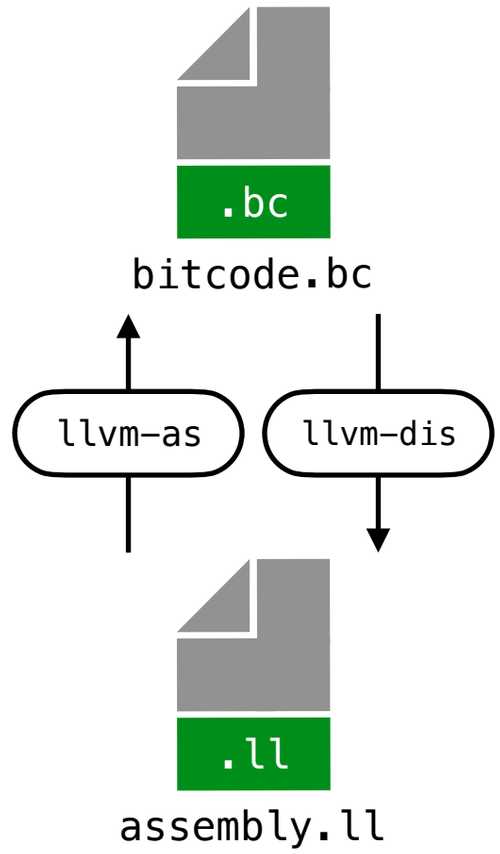
Compiler Architecture

Representation of LLVM-IR

LLVM-IR has 3 equivalent representations:

1. **in-memory representation**
   C++ objects representing LLVM-IR code implemented in LLVM Libraries
2. **on-disk representation**
   stream of bits representing LLVM-IR code encoded according to LLVM Bitcode Format
3. **human readable assembly**
   stream of characters representing LLVM-IR code interpreted according to the LLVM Language Reference

```
// llvm.IR
llvm::Module
llvm::GlobalVariable
llvm::Function
llvm::BasicBlock
llvm::Instruction
llvm::PHINode
llvm::CallInst
llvm::AllocaInst
llvm::LoadInst
llvm::StoreInst
llvm::CmpInst
llvm::ConstantInt
llvm::GetElementPtrInst
llvm::BitCastInst
llvm::ExtractElementInst
llvm::BinaryOperator
llvm::BranchInst
llvm::ReturnInst
```



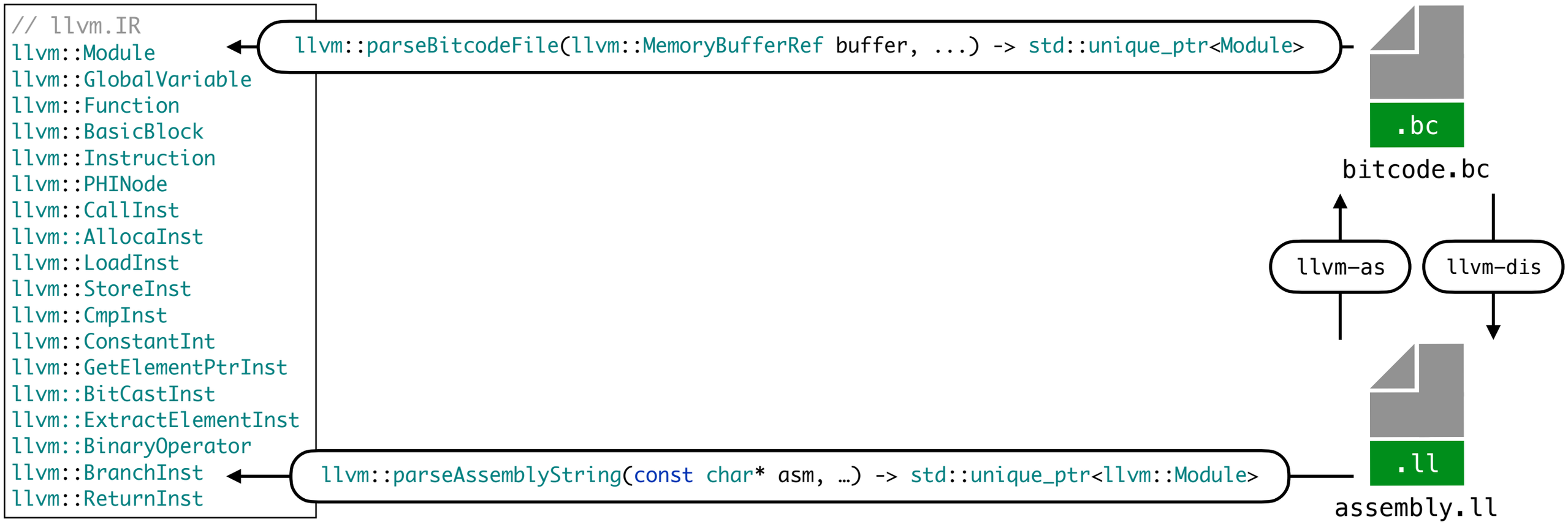
Compiler Architecture

Representation of LLVM-IR

LLVM-IR has 3 equivalent representations:

1. **in-memory representation**
   C++ objects representing LLVM-IR code implemented in LLVM Libraries
2. **on-disk representation**
   stream of bits representing LLVM-IR code encoded according to LLVM Bitcode Format
3. **human readable assembly**
   stream of characters representing LLVM-IR code interpreted according to the LLVM Language Reference

```
// llvm.IR
llvm::Module
llvm::GlobalVariable
llvm::Function
llvm::BasicBlock
llvm::Instruction
llvm::PHINode
llvm::CallInst
llvm::AllocaInst
llvm::LoadInst
llvm::StoreInst
llvm::CmpInst
llvm::ConstantInt
llvm::GetElementPtrInst
llvm::BitCastInst
llvm::ExtractElementInst
llvm::BinaryOperator
llvm::BranchInst
llvm::ReturnInst
```

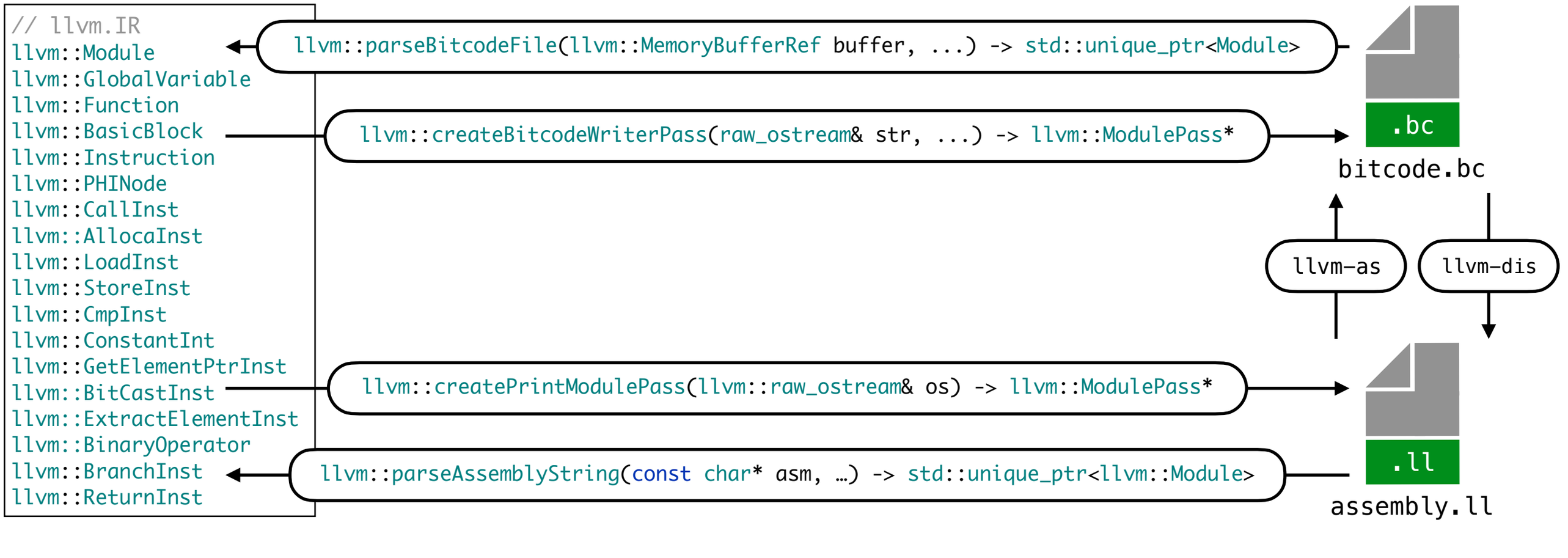


Compiler Architecture

Representation of LLVM-IR

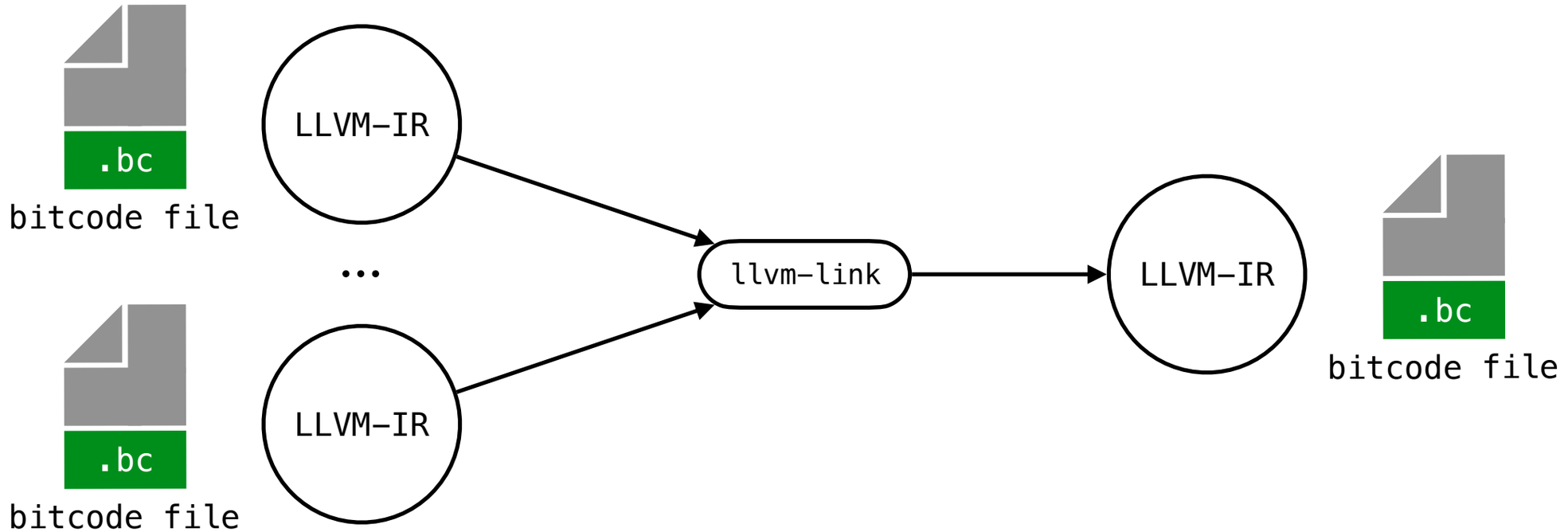
LLVM-IR has 3 equivalent representations:

1. **in-memory representation**
   C++ objects representing LLVM-IR code implemented in LLVM Libraries
2. **on-disk representation**
   stream of bits representing LLVM-IR code encoded according to LLVM Bitcode Format
3. **human readable assembly**
   stream of characters representing LLVM-IR code interpreted according to the LLVM Language Reference



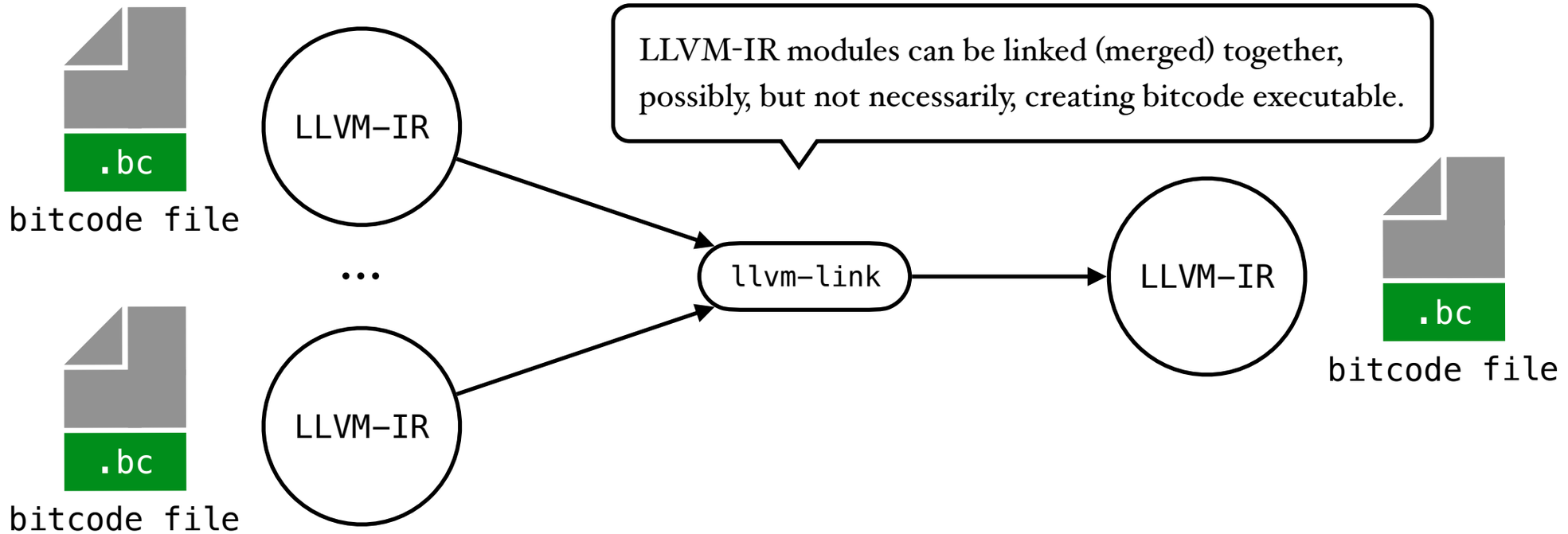
Compiler Architecture

Linking & Interpreting LLVM-IR



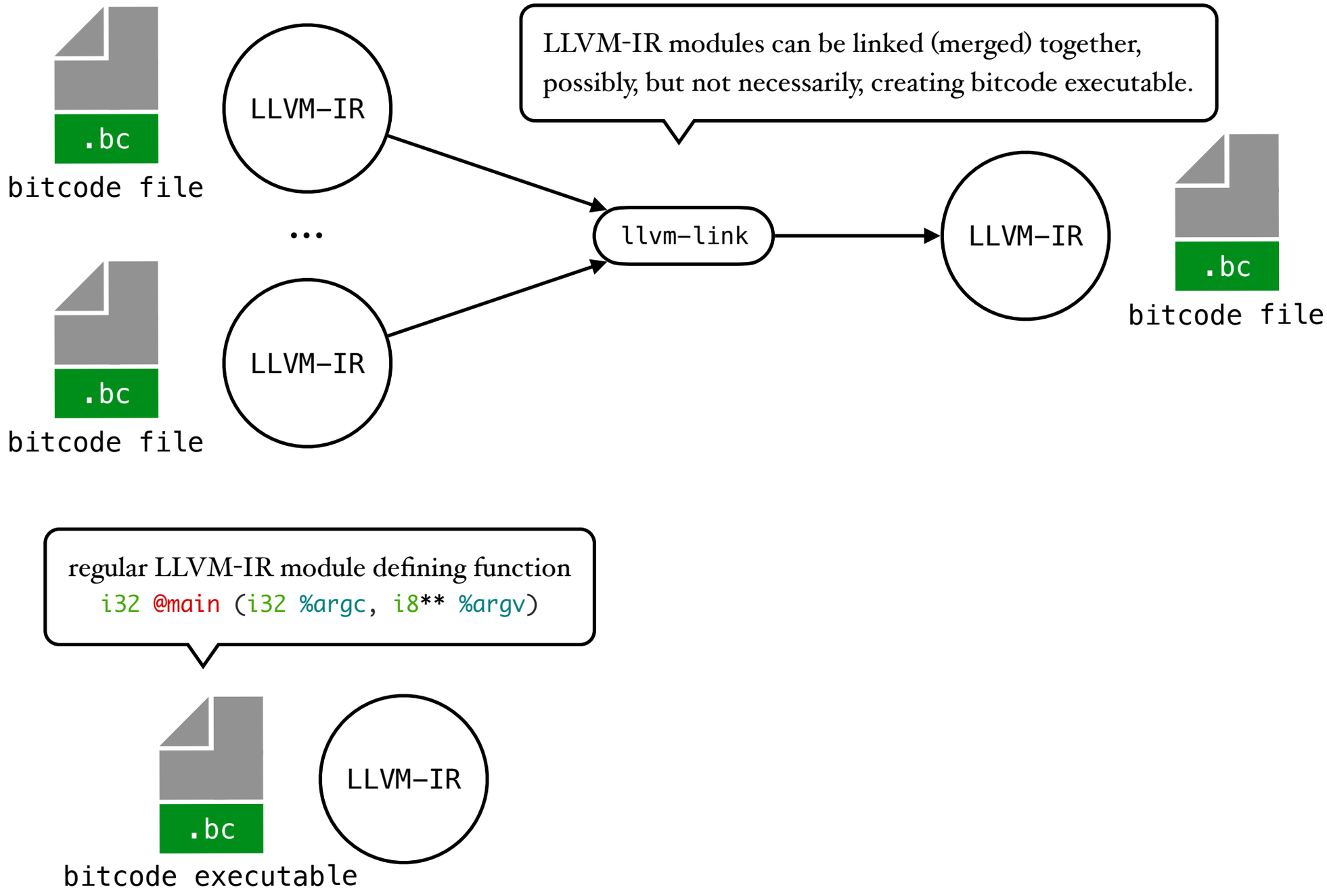
Compiler Architecture

Linking & Interpreting LLVM-IR



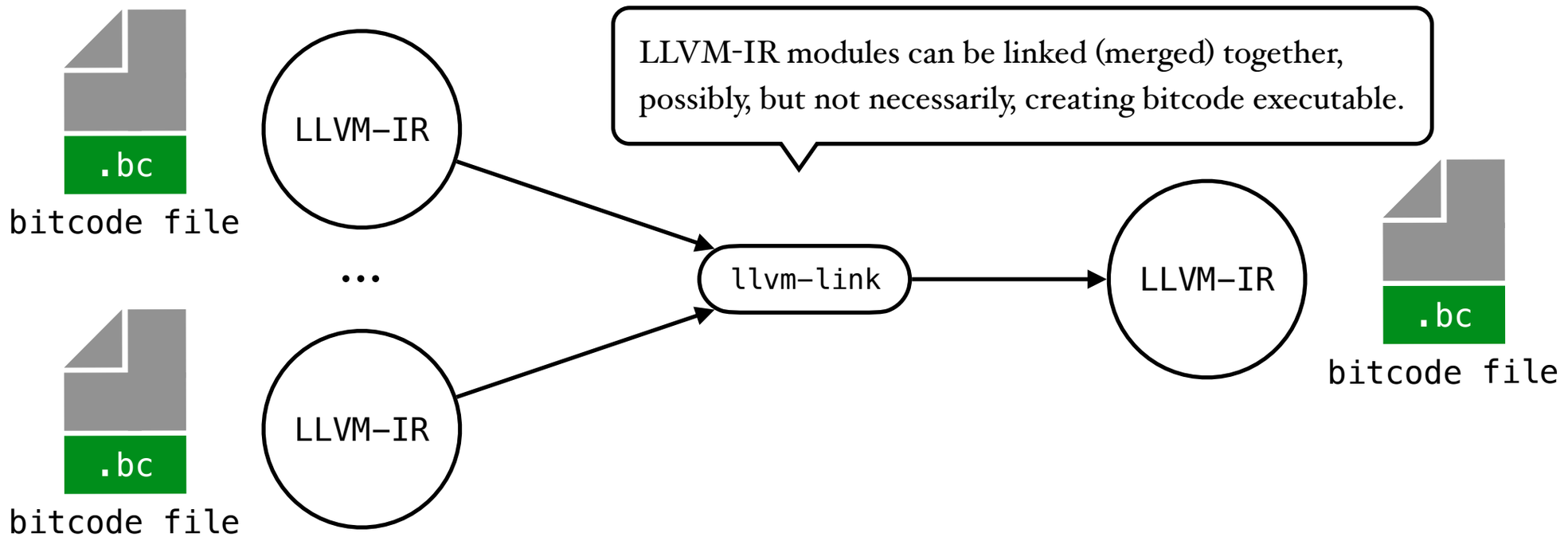
Compiler Architecture

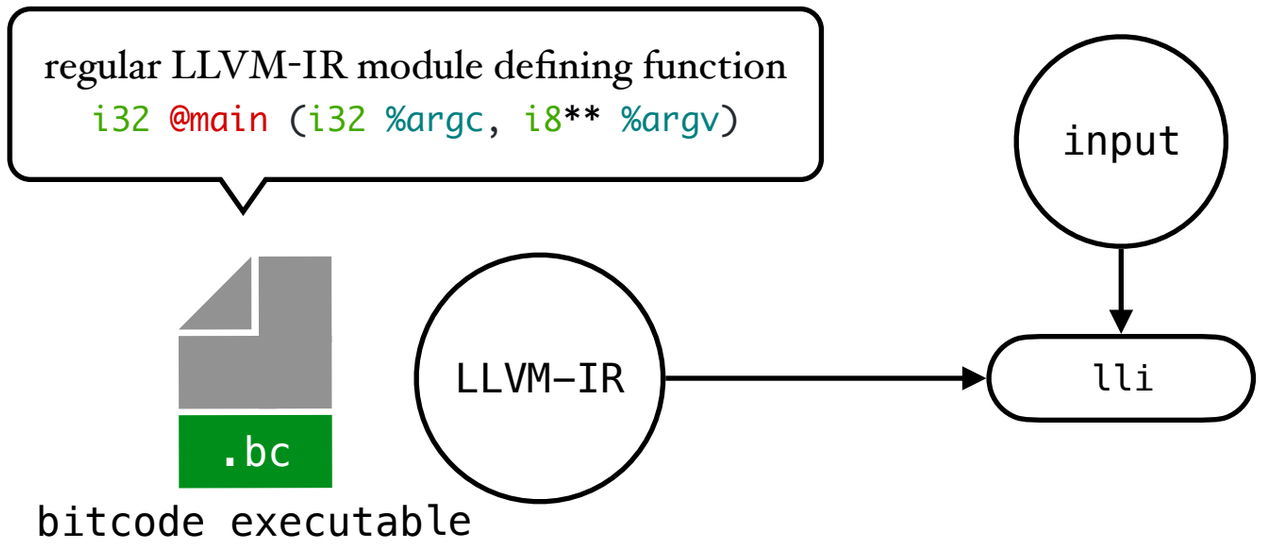
Linking & Interpreting LLVM-IR



Compiler Architecture

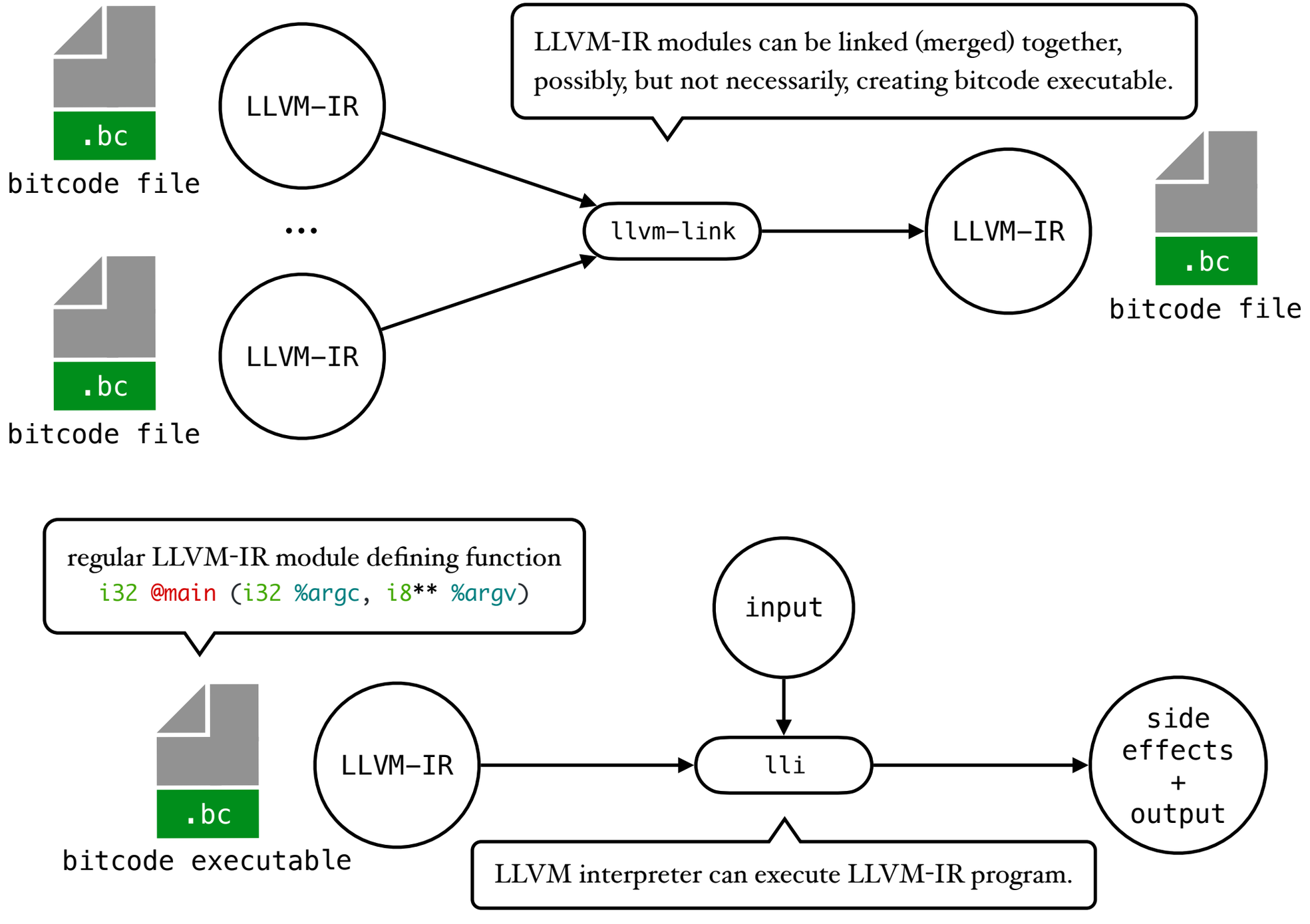
Linking & Interpreting LLVM-IR





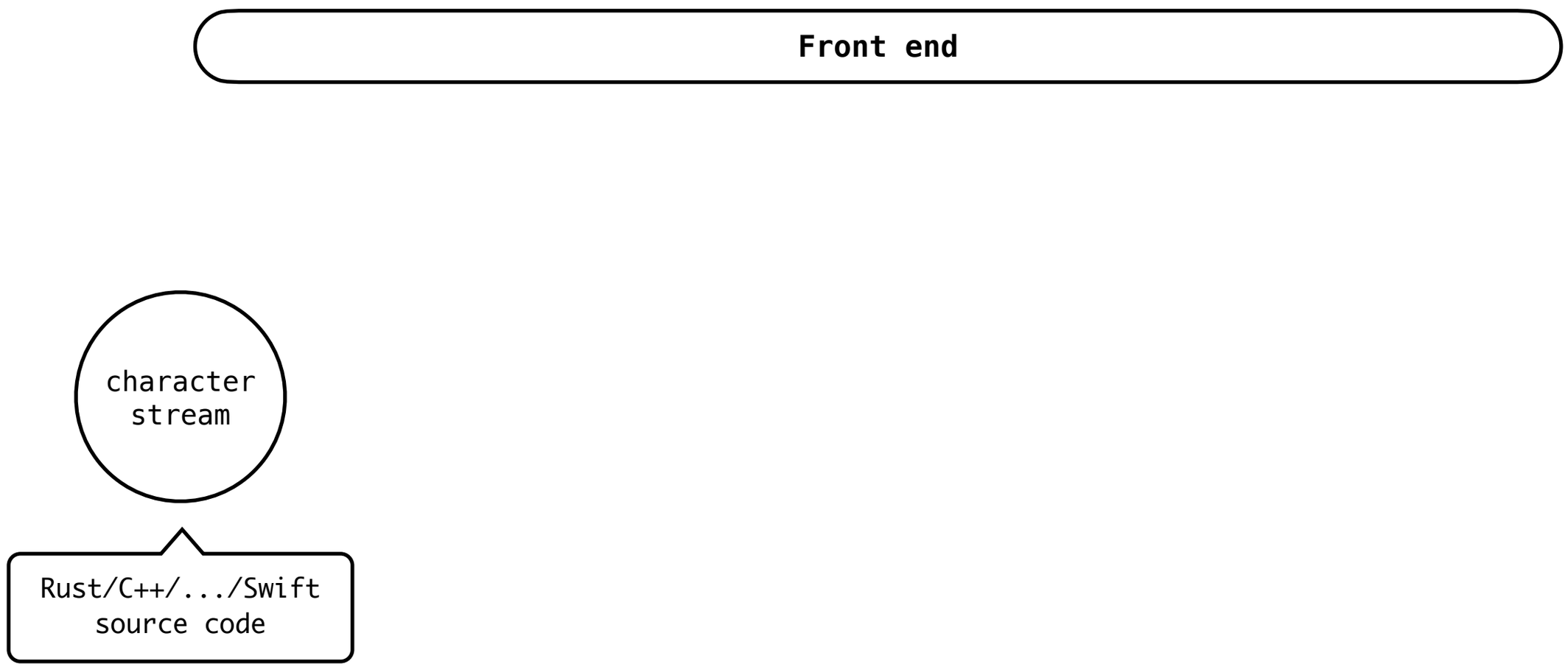
Compiler Architecture

Linking & Interpreting LLVM-IR



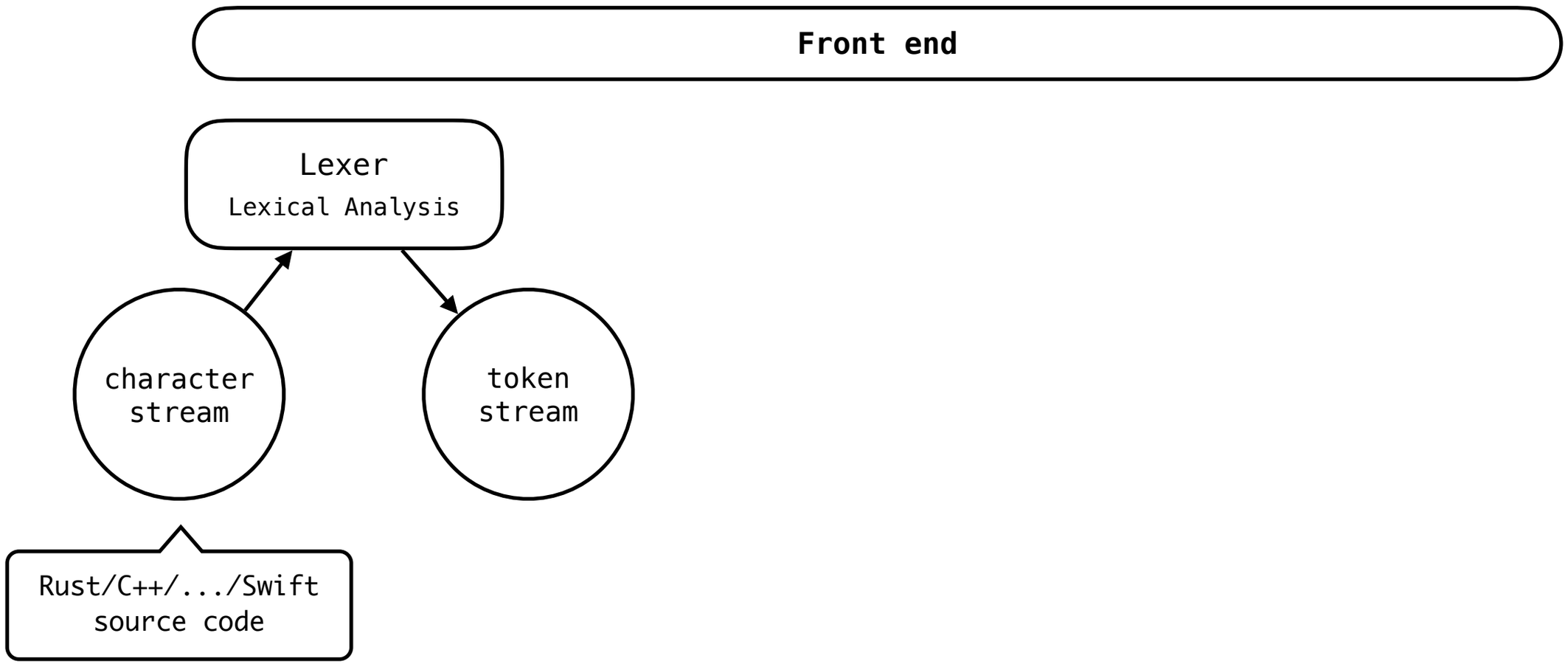
Compiler Architecture

Compiler Front-end



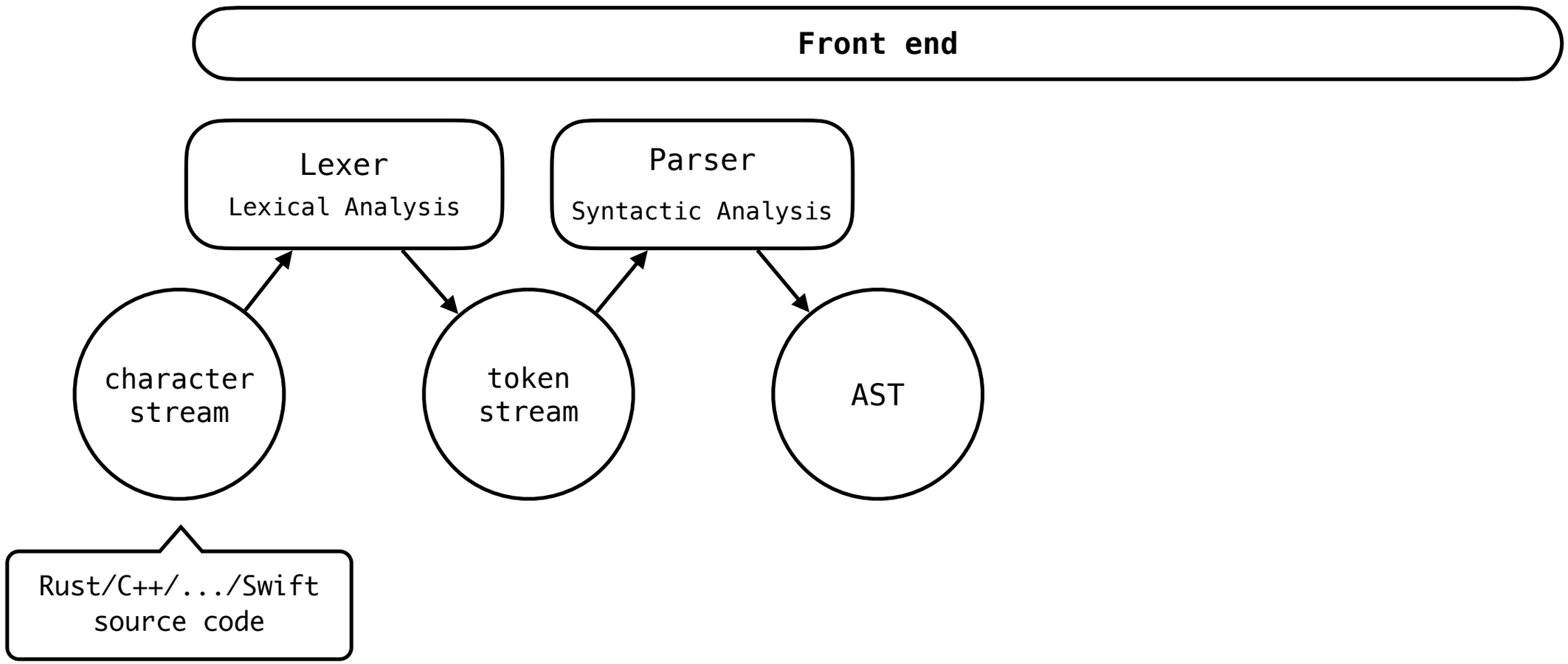
Compiler Architecture

Compiler Front-end



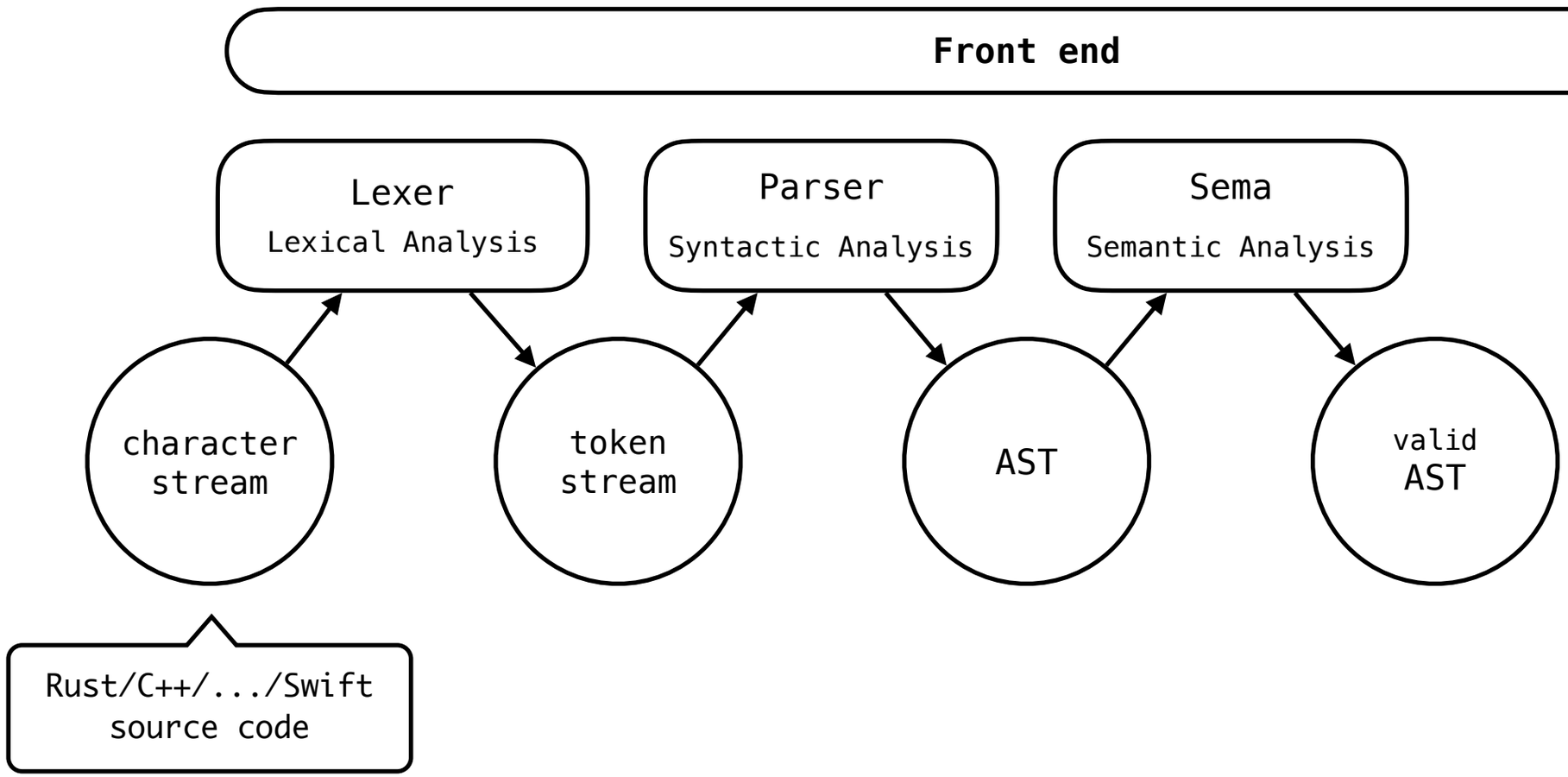
Compiler Architecture

Compiler Front-end



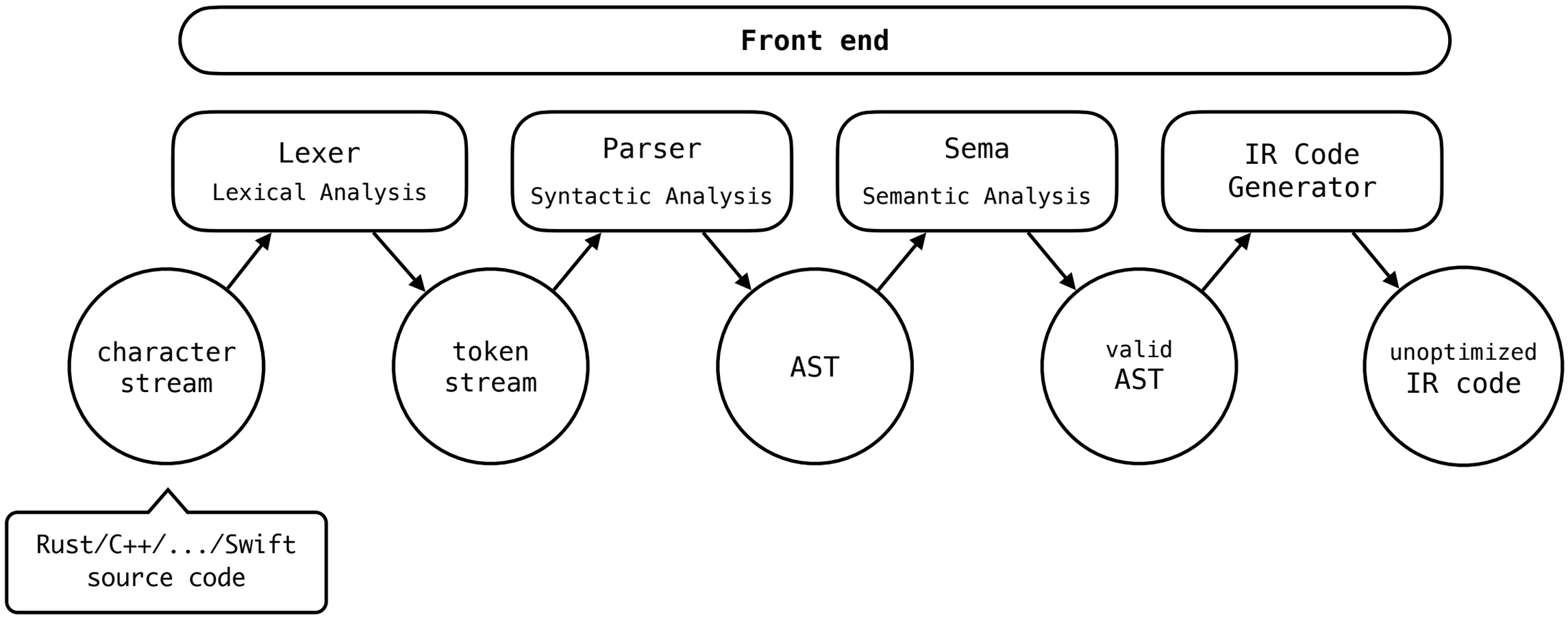
Compiler Architecture

Compiler Front-end



Compiler Architecture

Compiler Front-end



Compiler Architecture

Token

```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```

Compiler Architecture

Token

```
$ clang++ -std=c++17 -Xclang -dump-tokens    \
-c //physics_simulation/source/vec3_length.cxx
```

```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```

Compiler Architecture

Token

```
$ clang++ -std=c++17 -Xclang -dump-tokens    \
-c //physics_simulation/source/vec3_length.cxx
```

Token Kind	Token Value	Token Kind	Token Value	Token Kind	Token Value
namespace	'namespace'	amp	'&'	period	'.'
identifier	'phy_sim'	identifier	'v'	identifier	'y'
l_brace	'{'	r_paren	')'	star	'*'
struct	'struct'	arrow	'->'	identifier	'v'
identifier	'vec3'	float	'float'	period	'.'
l_brace	'{'	l_brace	'{'	identifier	'y'
float	'float'	return	'return'	plus	'+'
identifier	'x'	identifier	'std'	identifier	'v'
comma	','	coloncolon	'::'	period	'.'
identifier	'y'	identifier	'sqrt'	identifier	'z'
comma	','	l_paren	'('	star	'*'
identifier	'z'	identifier	'v'	identifier	'v'
semi	';'	period	'.'	period	'.'
r_brace	'}'	identifier	'x'	identifier	'z'
semi	';'	star	'*'	r_paren	')'
auto	'auto'	identifier	'v'	semi	';'
identifier	'length'	period	'.'	r_brace	'}'
l_paren	'('	identifier	'x'	r_brace	'}'
const	'const'	plus	'+'		
identifier	'vec3'	identifier	'v'		

```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```

Compiler Architecture

Token

```
$ clang++ -std=c++17 -Xclang -dump-tokens    \
-c //physics_simulation/source/vec3_length.cxx
```

Token Kind	Token Value	Token Kind	Token Value	Token Kind	Token Value
namespace	'namespace'	amp	'&'	period	'.'
identifier	'phy_sim'	identifier	'v'	identifier	'y'
l_brace	'{'	r_paren	')'	star	'*'
struct	'struct'	arrow	'->'	identifier	'v'
identifier	'vec3'	float	'float'	period	'.'
l_brace	'{'	l_brace	'{'	identifier	'y'
float	'float'	return	'return'	plus	'+'
identifier	'x'	identifier	'std'	identifier	'v'
comma	','	coloncolon	'::'	period	'.'
identifier	'y'	identifier	'sqrt'	identifier	'z'
comma	','	l_paren	'('	star	'*'
identifier	'z'	identifier	'v'	identifier	'v'
semi	';'	period	'.'	period	'.'
r_brace	'}'	identifier	'x'	identifier	'z'
semi	';'	star	'*'	r_paren	')'
auto	'auto'	identifier	'v'	semi	';'
identifier	'length'	period	'.'	r_brace	'}'
l_paren	'('	identifier	'x'	r_brace	'}'
const	'const'	plus	'+'		
identifier	'vec3'	identifier	'v'		

```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```

Token is a sequence of characters, which has identified semantics.

Tokens represent keywords, operators, literals or identifiers.

Compiler Architecture

Abstract Syntax Tree

```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```

Compiler Architecture

Abstract Syntax Tree

```
$ clang++ -std=c++17 -Xclang -ast-dump       \
-c //physics_simulation/source/vec3_length.cxx
```

```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```

Compiler Architecture

Abstract Syntax Tree

```
$ clang++ -std=c++17 -Xclang -ast-dump       \
-c //physics_simulation/source/vec3_length.cxx
```

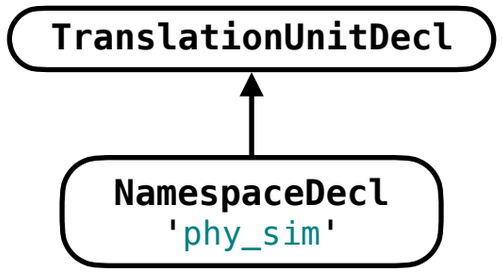
```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```

Compiler Architecture

Abstract Syntax Tree

```
$ clang++ -std=c++17 -Xclang -ast-dump       \
-c //physics_simulation/source/vec3_length.cxx
```

```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```

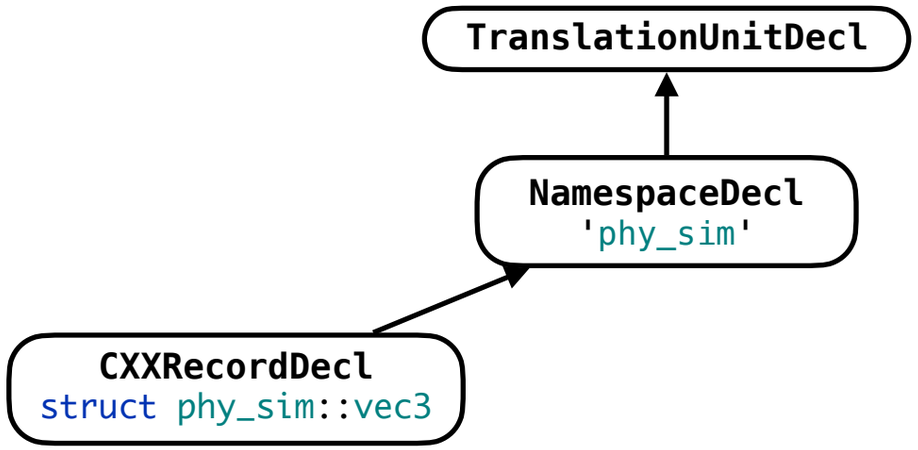


Compiler Architecture

Abstract Syntax Tree

```
$ clang++ -std=c++17 -Xclang -ast-dump       \
-c //physics_simulation/source/vec3_length.cxx
```

```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```

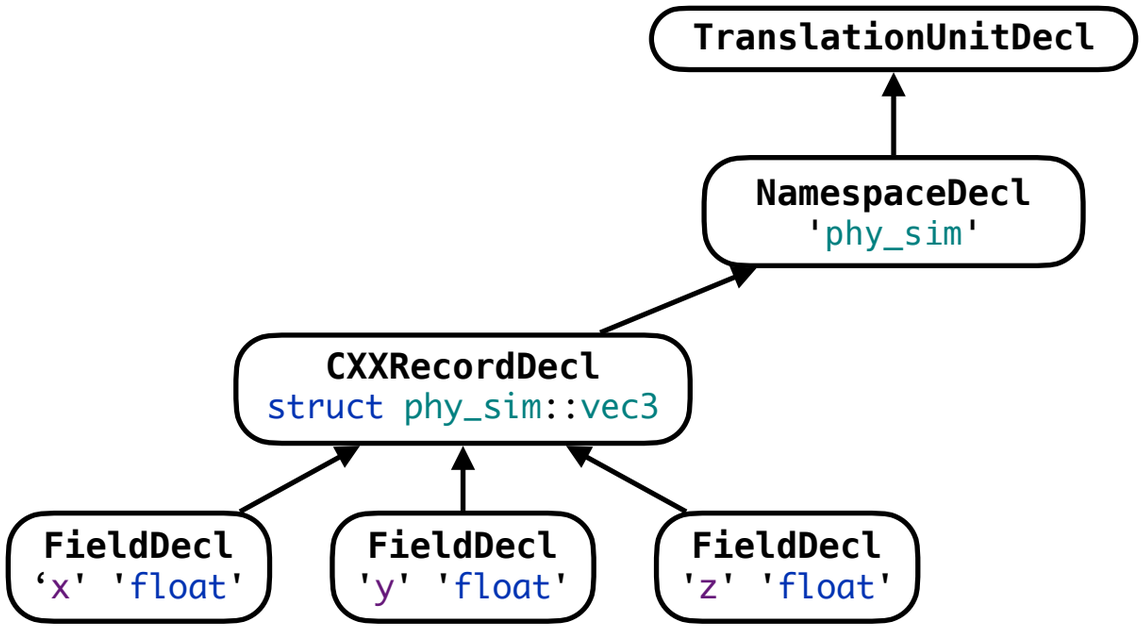


Compiler Architecture

Abstract Syntax Tree

```
$ clang++ -std=c++17 -Xclang -ast-dump       \
-c //physics_simulation/source/vec3_length.cxx
```

```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```

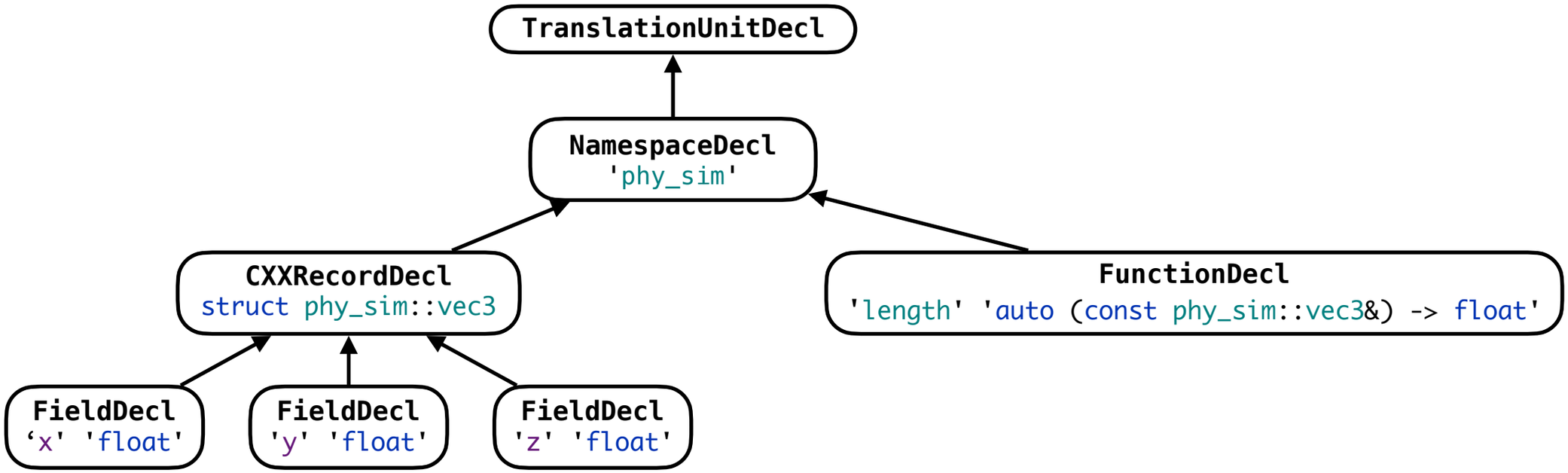


Compiler Architecture

Abstract Syntax Tree

```
$ clang++ -std=c++17 -Xclang -ast-dump       \
-c //physics_simulation/source/vec3_length.cxx
```

```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```

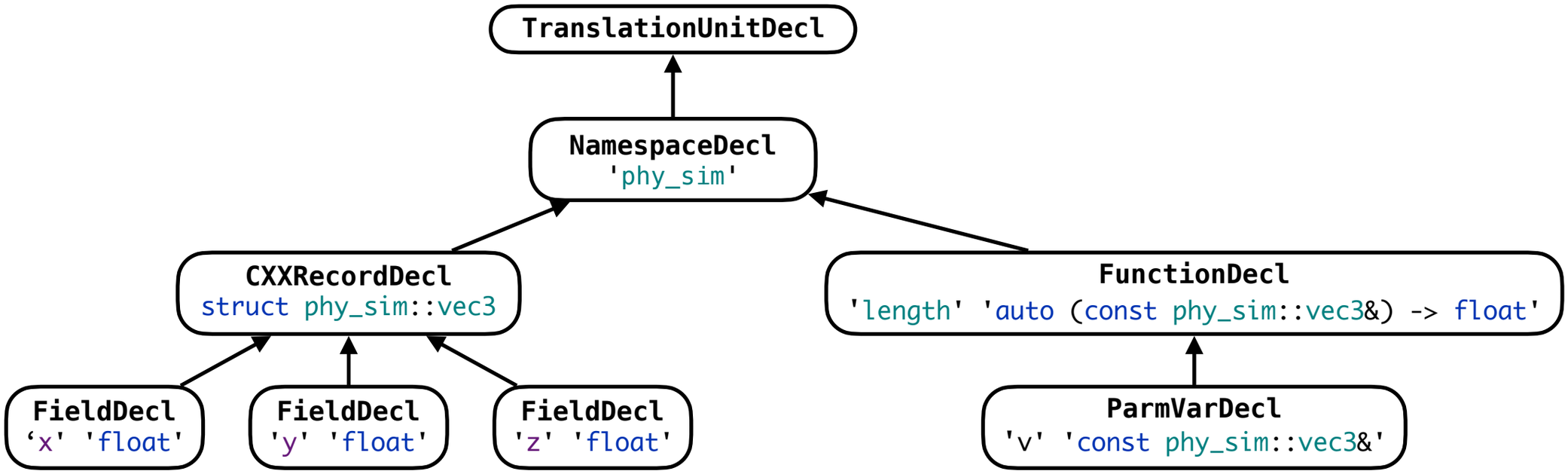


Compiler Architecture

Abstract Syntax Tree

```
$ clang++ -std=c++17 -Xclang -ast-dump       \
-c //physics_simulation/source/vec3_length.cxx
```

```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```



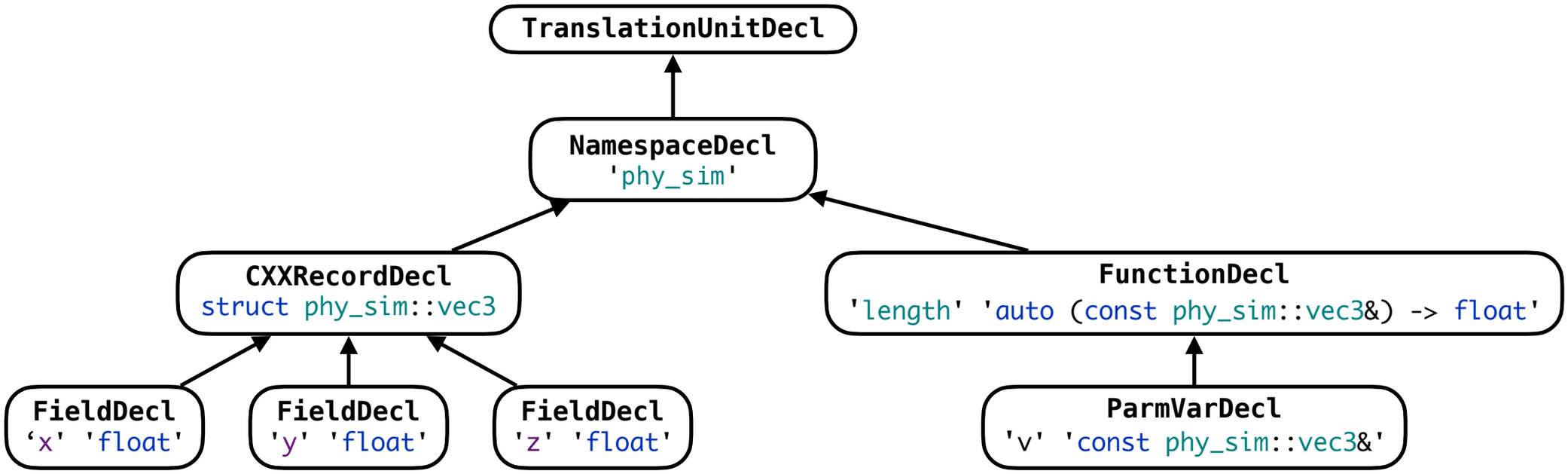
Compiler Architecture

Abstract Syntax Tree

```
$ clang++ -std=c++17 -Xclang -ast-dump       \
-c //physics_simulation/source/vec3_length.cxx
```

```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```

Abstract Syntax Tree (AST) is concrete representation of C++ programs,
in which their syntax & semantics is represented by
a directed graph of C++ entities (vertices) and relations between them (edges).



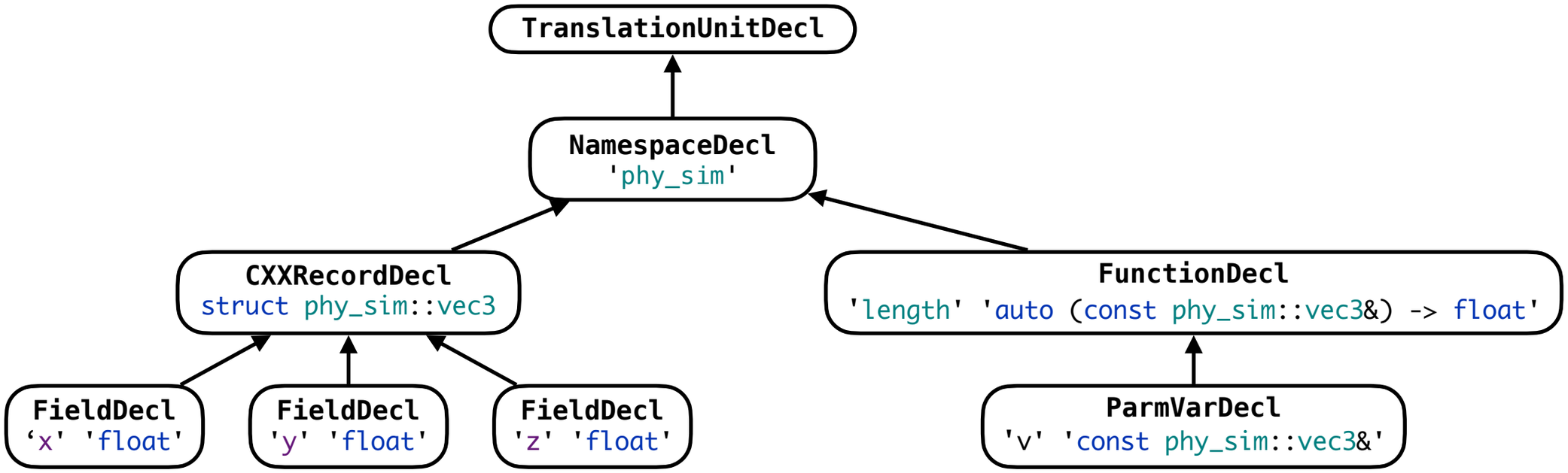
Compiler Architecture

Abstract Syntax Tree

```
$ clang++ -std=c++17 -Xclang -ast-dump       \
-c //physics_simulation/source/vec3_length.cxx
```

```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```

Abstract Syntax Tree (AST) is concrete representation of C++ programs,
in which their syntax & semantics is represented by
a directed graph of C++ entities (vertices) and relations between them (edges).

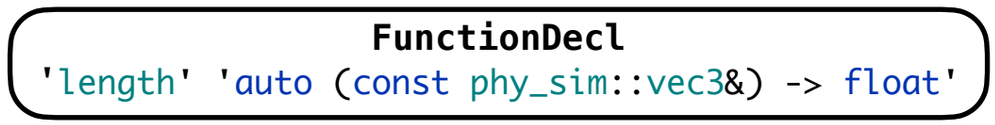


AST is representation of C++ programs alternative to C++ source code,
that is conversion of C++ source code to AST is reversible without losing semantics.

Compiler Architecture

Abstract Syntax Tree

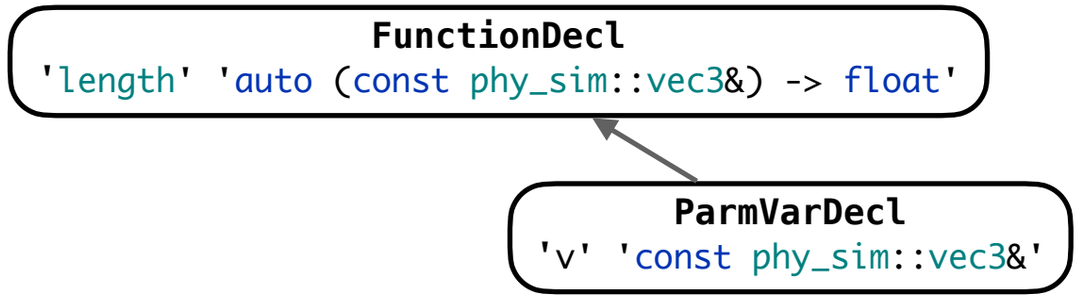
```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```



Compiler Architecture

Abstract Syntax Tree

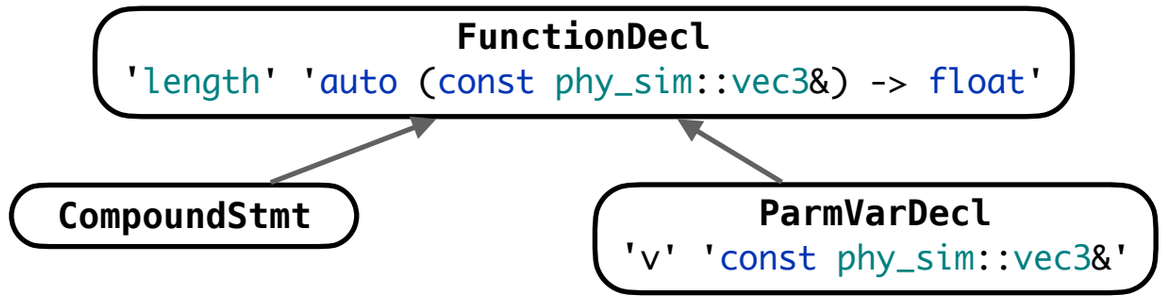
```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```



Compiler Architecture

Abstract Syntax Tree

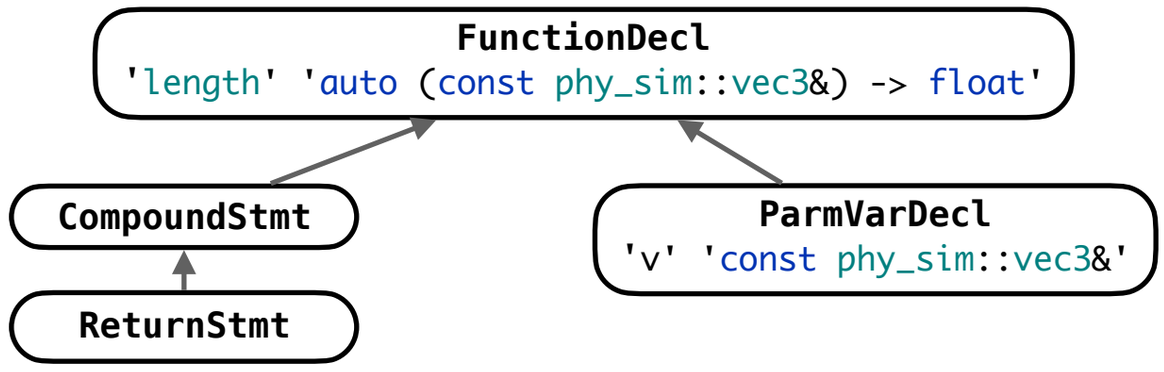
```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```



Compiler Architecture

Abstract Syntax Tree

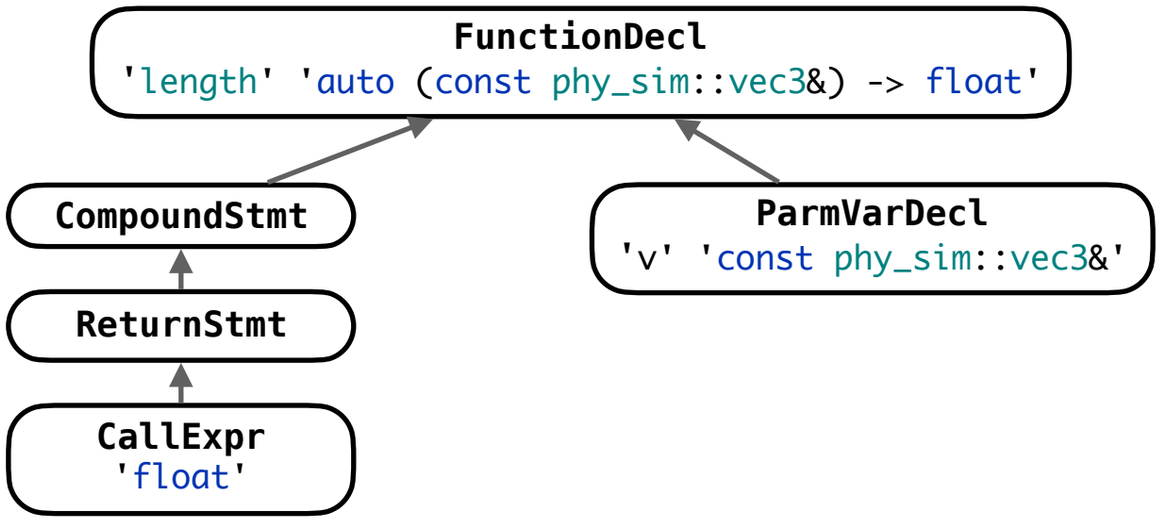
```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```



Compiler Architecture

Abstract Syntax Tree

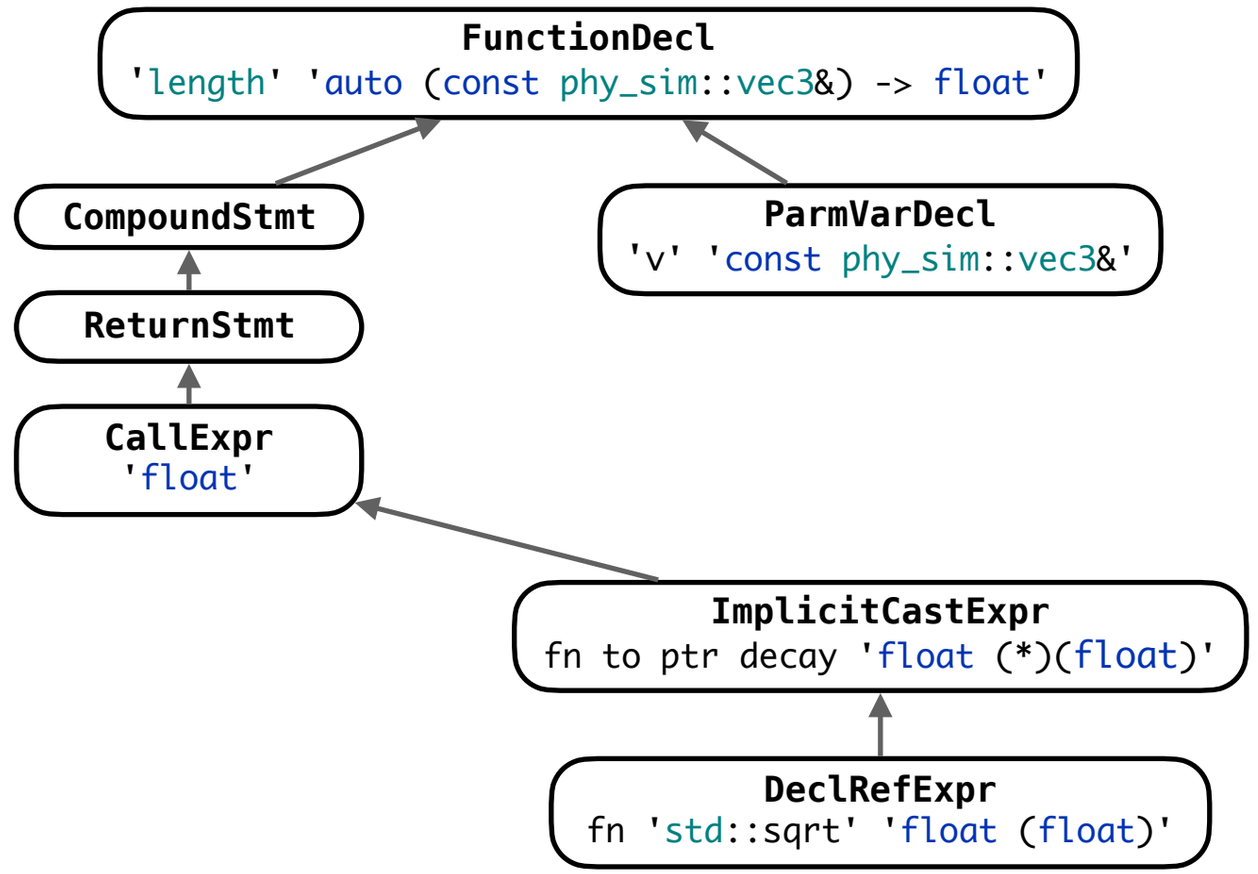
```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```



Compiler Architecture

Abstract Syntax Tree

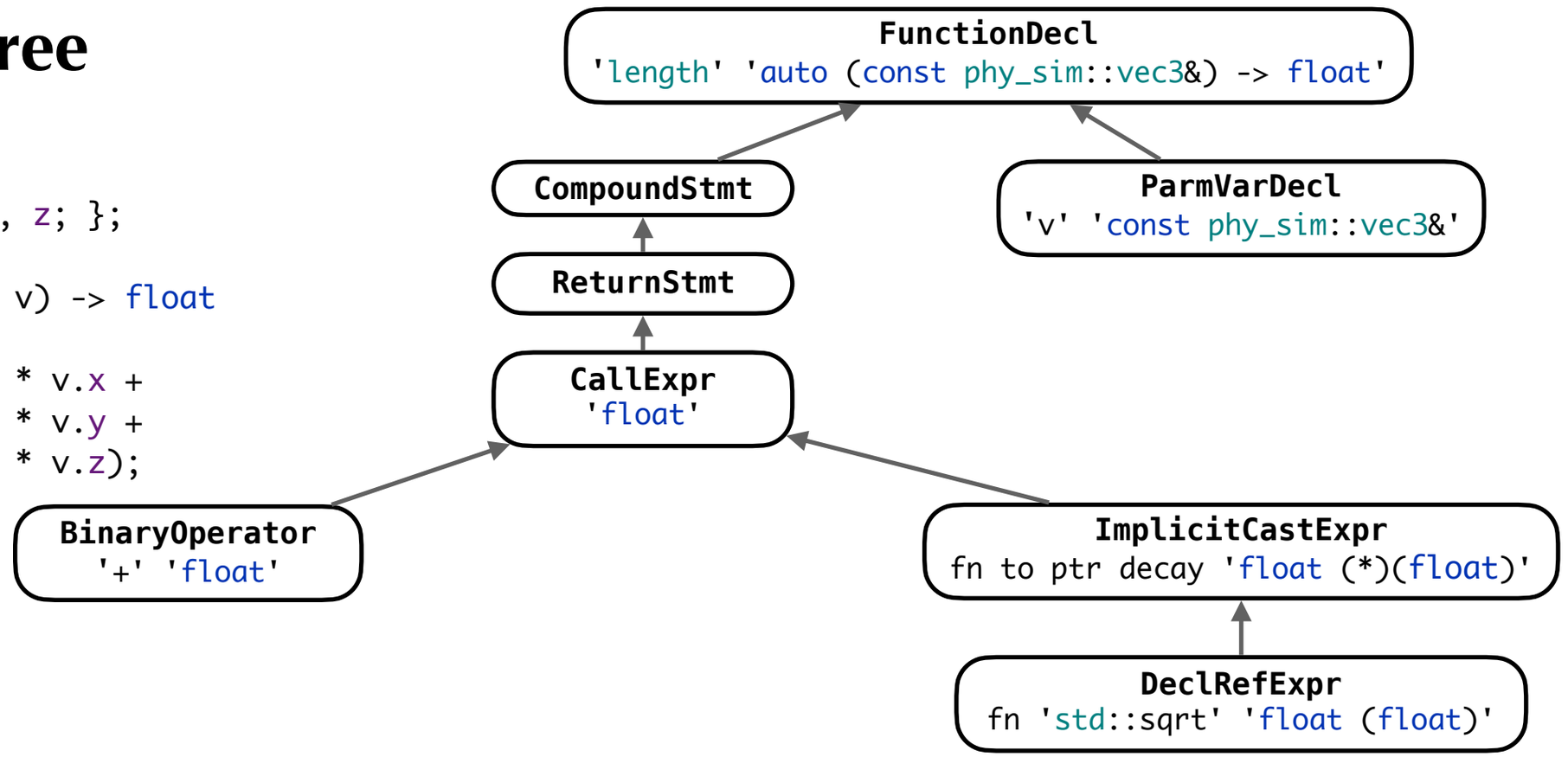
```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```



Compiler Architecture

Abstract Syntax Tree

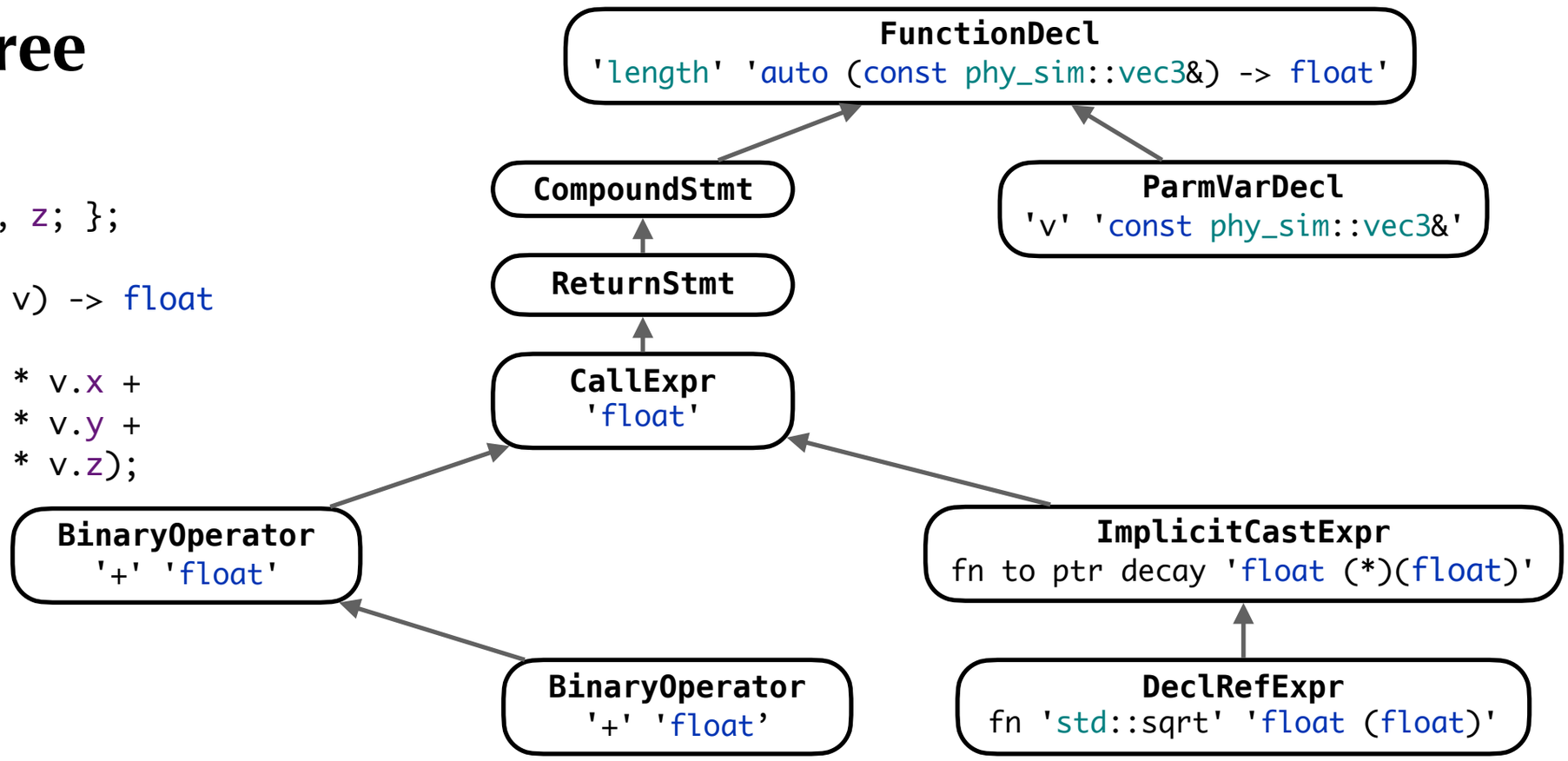
```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```



Compiler Architecture

Abstract Syntax Tree

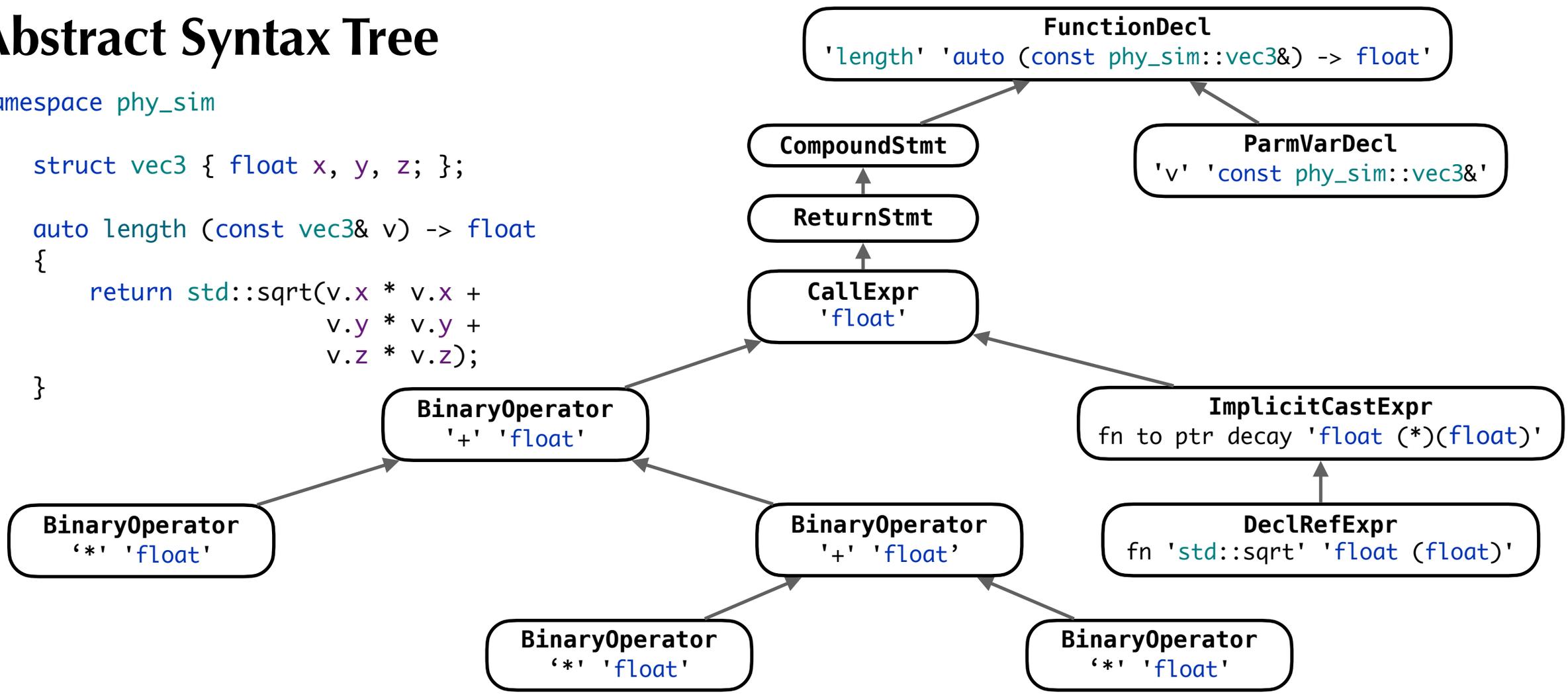
```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```



Compiler Architecture

Abstract Syntax Tree

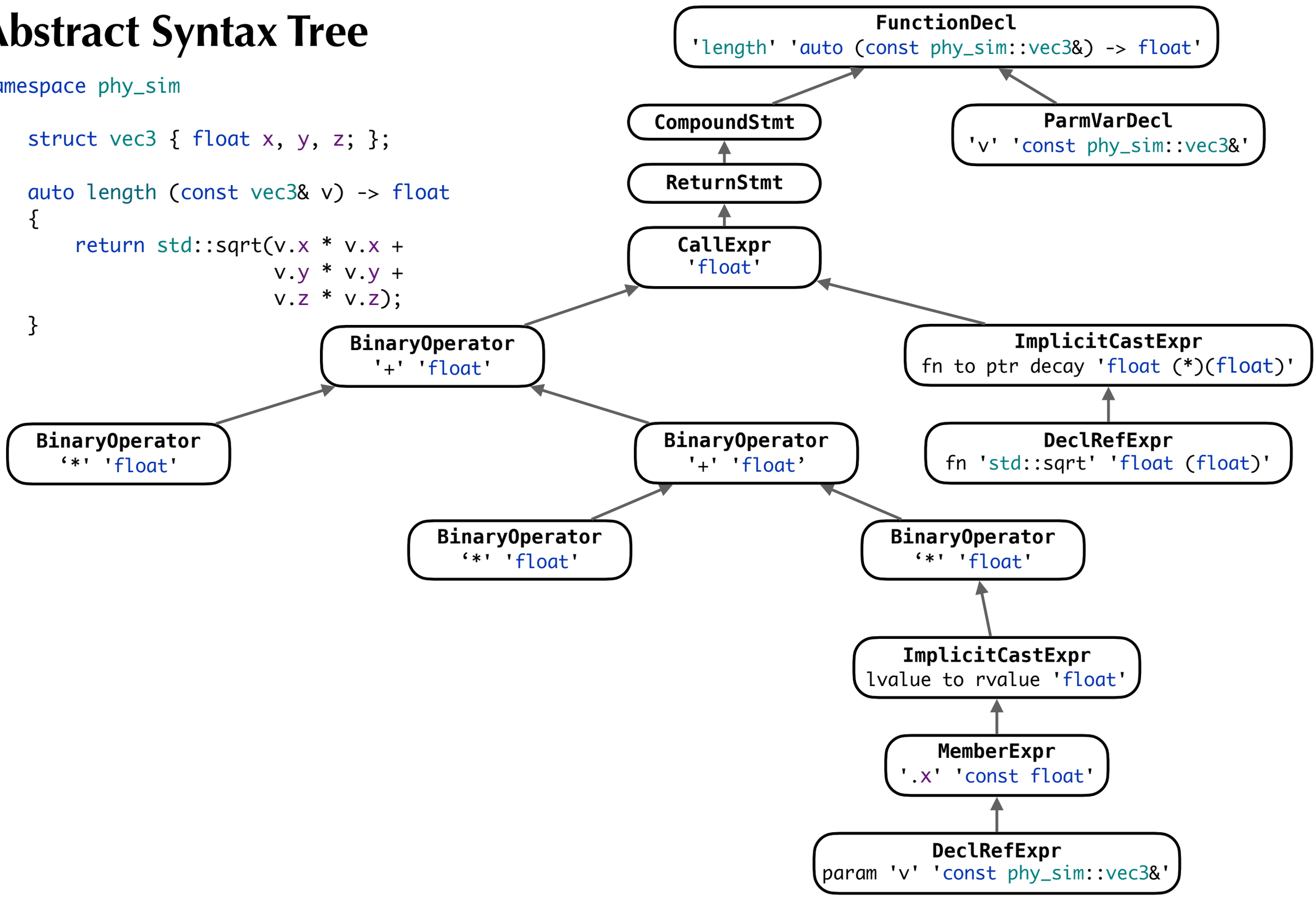
```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```



Compiler Architecture

Abstract Syntax Tree

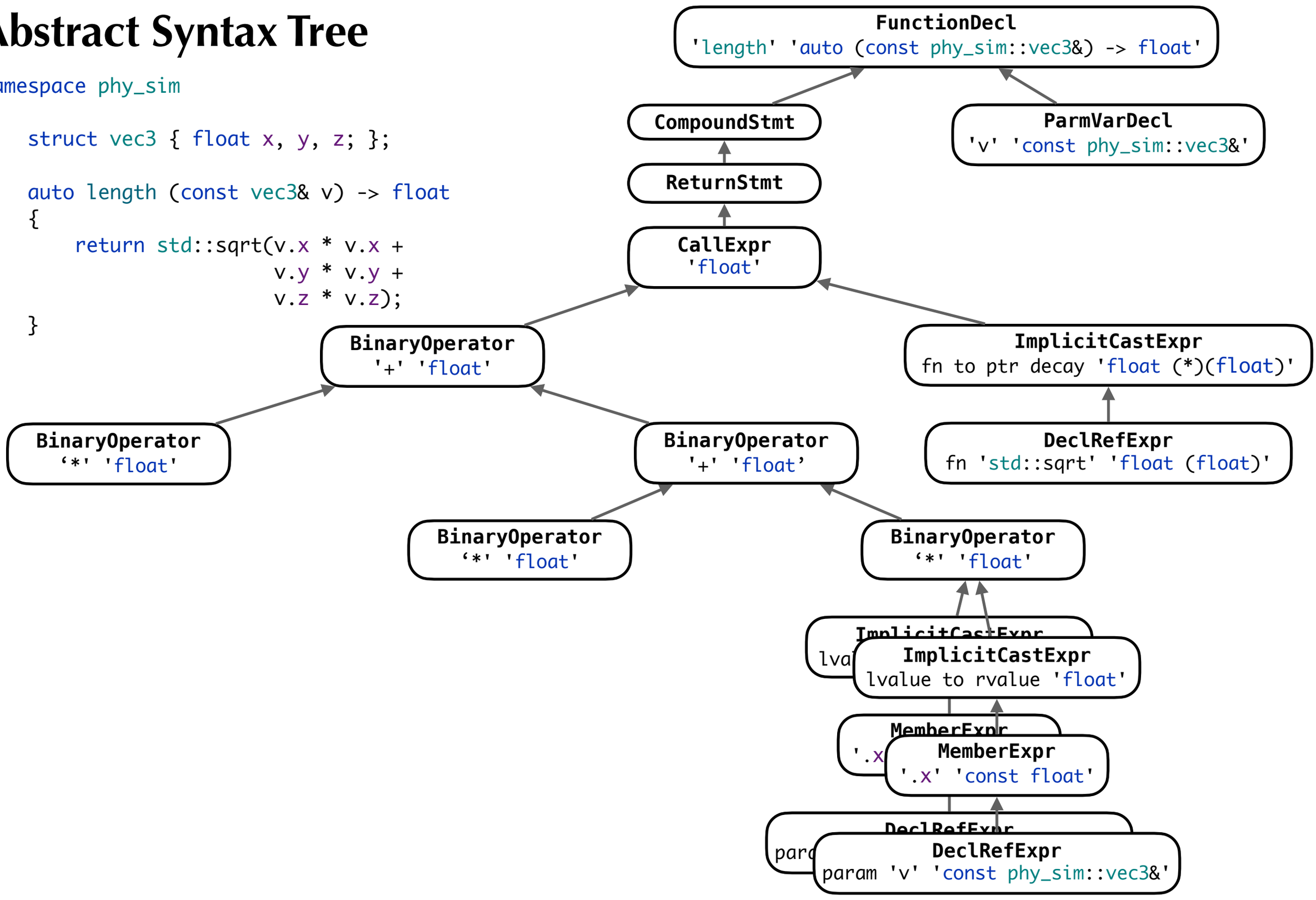
```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```



Compiler Architecture

Abstract Syntax Tree

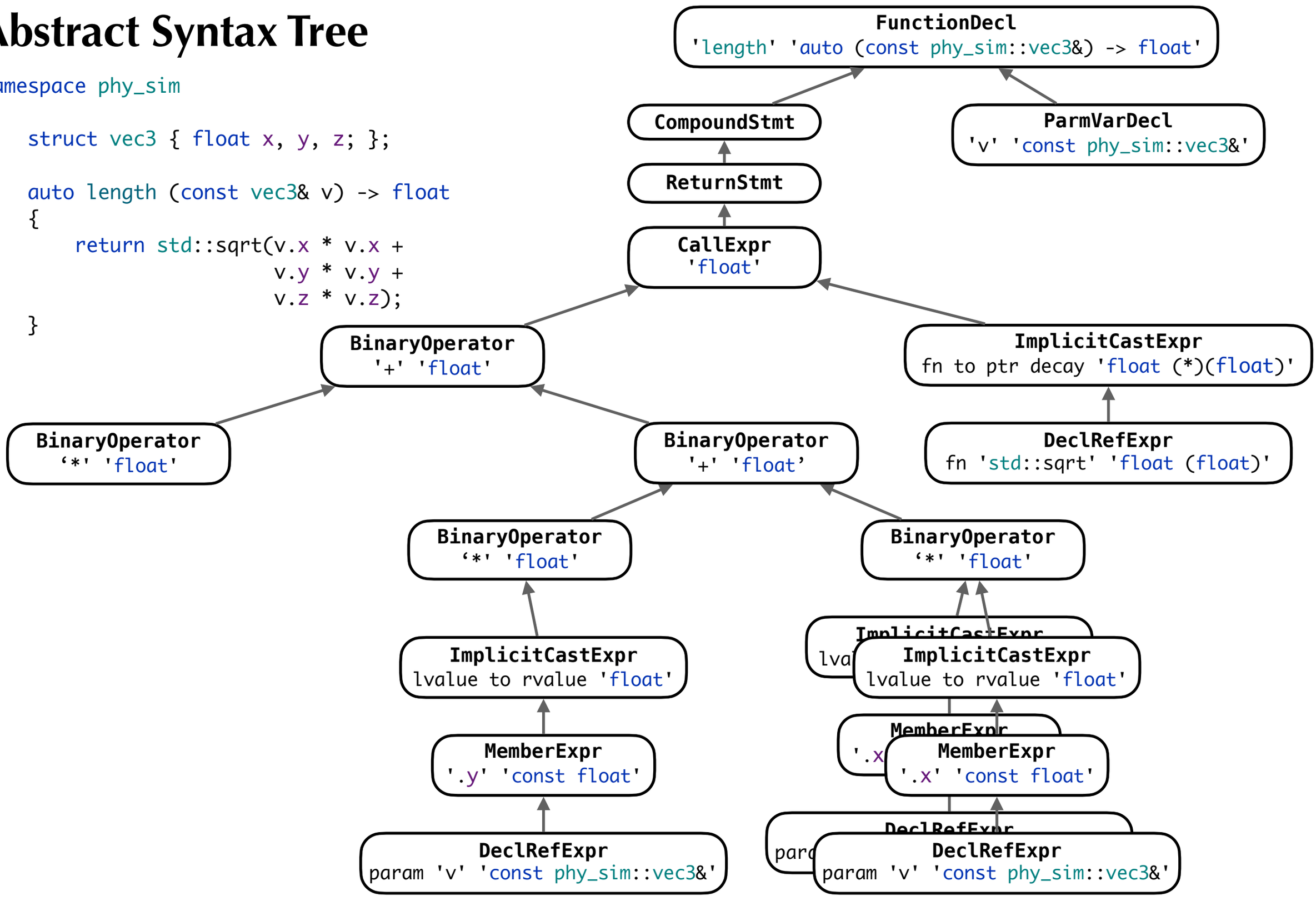
```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```



Compiler Architecture

Abstract Syntax Tree

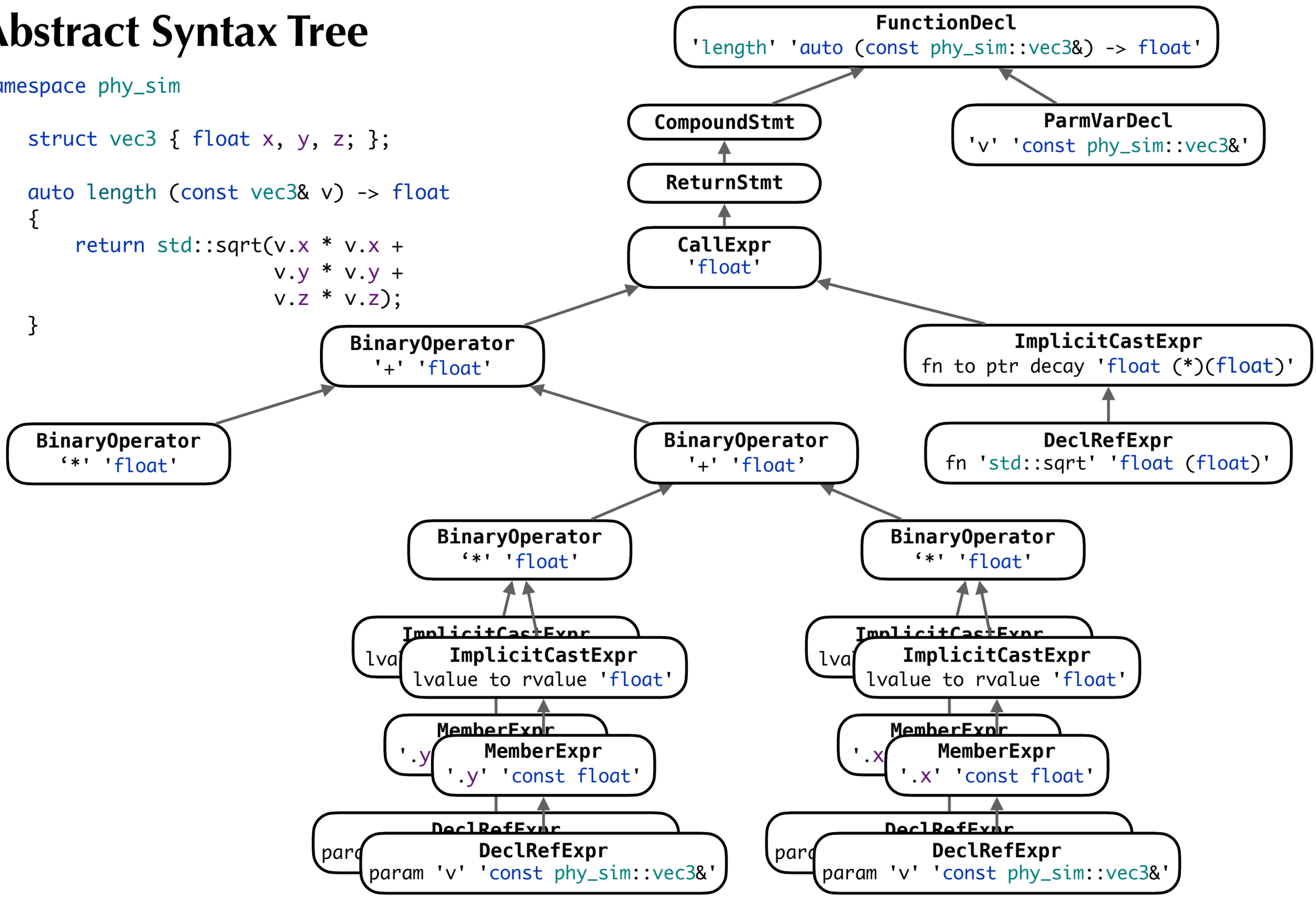
```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```



Compiler Architecture

Abstract Syntax Tree

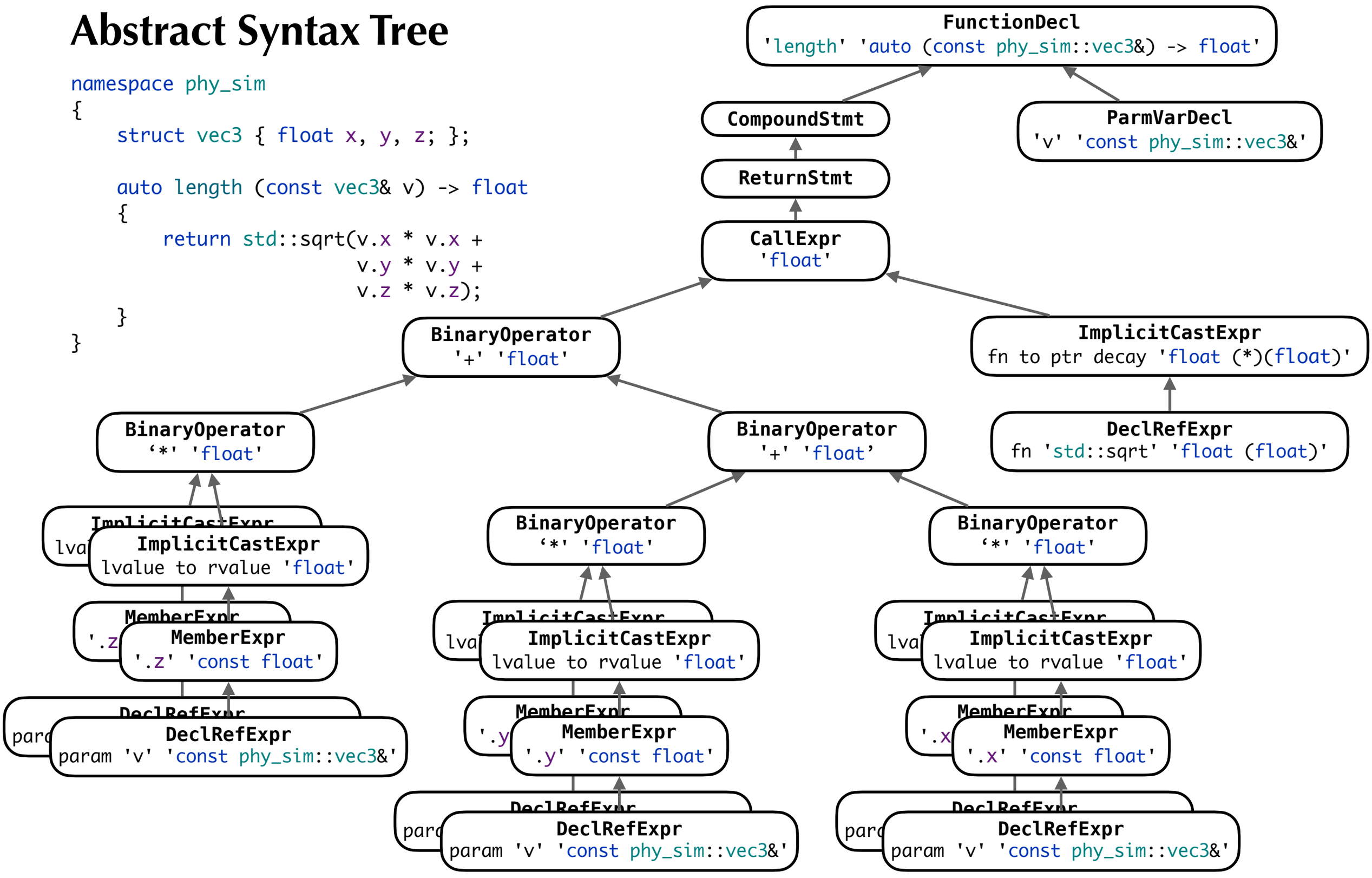
```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```



Compiler Architecture

Abstract Syntax Tree

```
namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float
    {
        return std::sqrt(v.x * v.x +
                         v.y * v.y +
                         v.z * v.z);
    }
}
```



Compiler Architecture

Serialization & Deserialization of Clang AST

```
// clang.AST
clang::TranslationUnitDecl
clang::NamespaceDecl
clang::CXXRecordDecl
clang::FieldDecl
clang::FunctionDecl
clang::VarDecl
clang::Stmt
clang::CompoundStmt
clang::IfStmt
clang::WhileStmt
clang::ReturnStmt
clang::ValueStmt
clang::Expr
clang::DeclRefExpr
clang::IntegerLiteral
clang::FloatingLiteral
clang::StringLiteral
clang::UnaryOperator
clang::CallExpr
clang::MemberExpr
clang::ExplicitCastExpr
clang::ImplicitCastExpr
clang::BinaryOperator
clnag::InitListExpr
```

Compiler Architecture

Serialization & Deserialization of Clang AST

```
clang::ASTWriter::WriteAST(clang::Sema &SemaRef, const std::string &OutputFile, ...) -> clang::ASTFileSignature
```

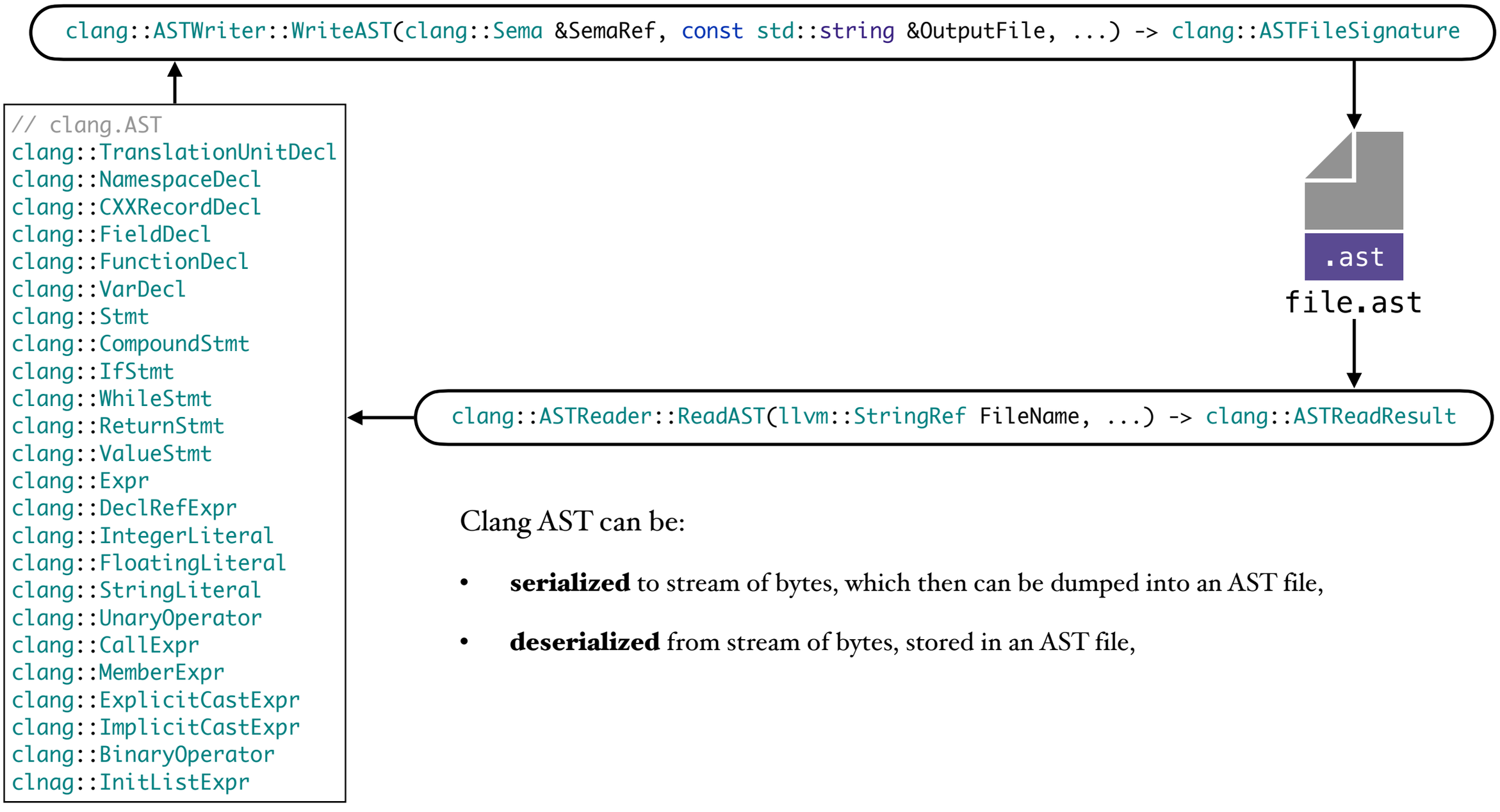
```
// clang.AST
clang::TranslationUnitDecl
clang::NamespaceDecl
clang::CXXRecordDecl
clang::FieldDecl
clang::FunctionDecl
clang::VarDecl
clang::Stmt
clang::CompoundStmt
clang::IfStmt
clang::WhileStmt
clang::ReturnStmt
clang::ValueStmt
clang::Expr
clang::DeclRefExpr
clang::IntegerLiteral
clang::FloatingLiteral
clang::StringLiteral
clang::UnaryOperator
clang::CallExpr
clang::MemberExpr
clang::ExplicitCastExpr
clang::ImplicitCastExpr
clang::BinaryOperator
clnag::InitListExpr
```

Clang AST can be:

- **serialized** to stream of bytes, which then can be dumped into an AST file,

Compiler Architecture

Serialization & Deserialization of Clang AST

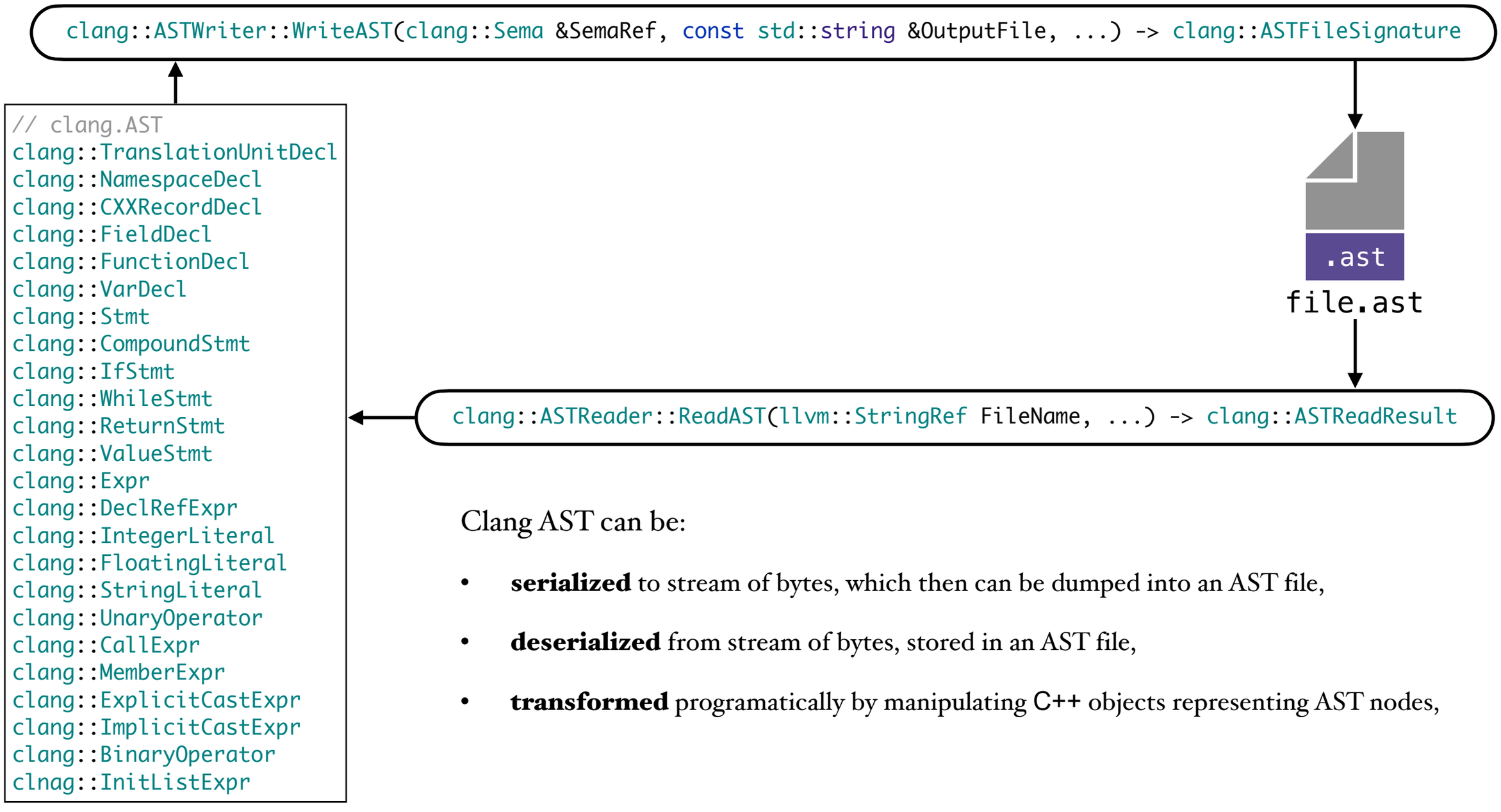


Clang AST can be:

- **serialized** to stream of bytes, which then can be dumped into an AST file,
- **deserialized** from stream of bytes, stored in an AST file,

Compiler Architecture

Serialization & Deserialization of Clang AST

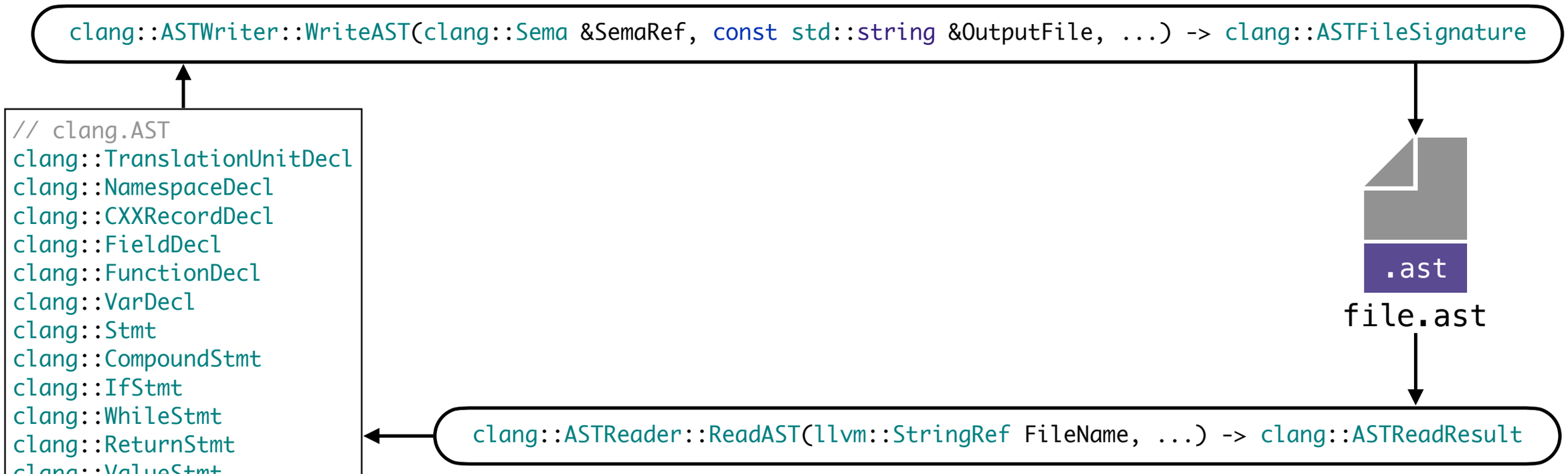


Clang AST can be:

- **serialized** to stream of bytes, which then can be dumped into an AST file,
- **deserialized** from stream of bytes, stored in an AST file,
- **transformed** programatically by manipulating C++ objects representing AST nodes,

Compiler Architecture

Serialization & Deserialization of Clang AST



Clang AST can be:

- **serialized** to stream of bytes, which then can be dumped into an AST file,
- **deserialized** from stream of bytes, stored in an AST file,
- **transformed** programatically by manipulating C++ objects representing AST nodes,
- **merged** with another Clang AST using the `clang::ASTImporter`,
  assuming merged Clang ASTs do not violate C++ ODR,
  that is both could coexist in the same C++ program without any conflicts.

Question

How to design better compilation model for C++?

Question

How to design better compilation model for C++?

Key ideas:

1. Introduce **C++ modules**, a new kind of C++ entities, consisting of **interface** and **implementation units**, to which already existing C++ entities could belong to, to provide true **encapsulation** of implementation details.

Question

How to design better compilation model for C++?

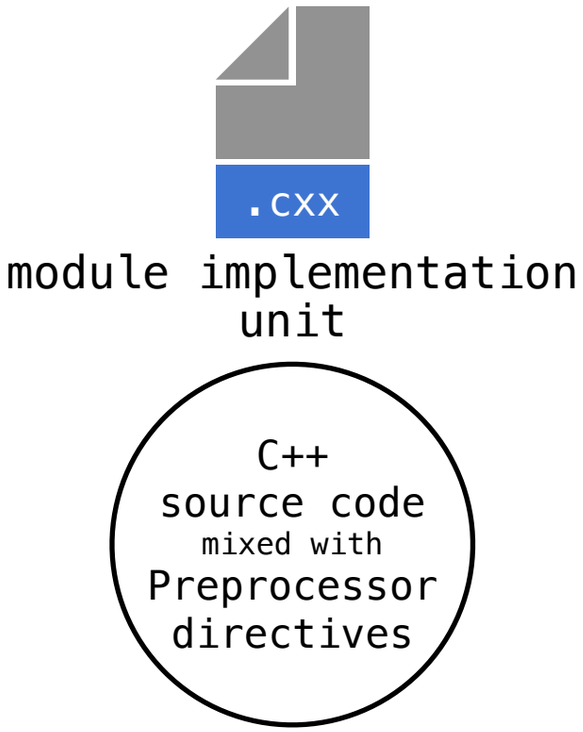
Key ideas:

1. Introduce **C++ modules**, a new kind of C++ entities,
   consisting of **interface** and **implementation units**,
   to which already existing C++ entities could belong to,
   to provide true **encapsulation** of implementation details.

2. Store on a disk **binary representation** of module interface units,
   which will be **efficient to ingest** by a C++ compiler
   in subsequent compilations of dependent module units,
   to **avoid redundant** and **expensive processing**
   performed by the C++ compiler.

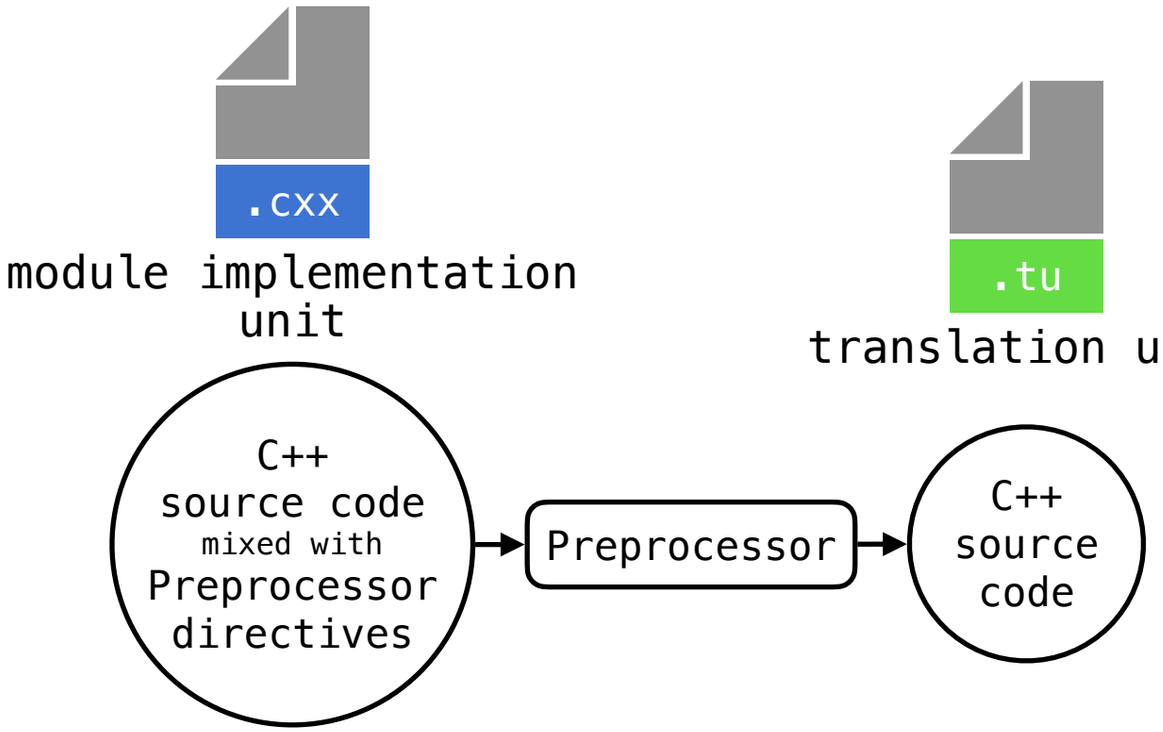
Modular Import Model

Module Implementation Unit - Preprocessing & Compilation



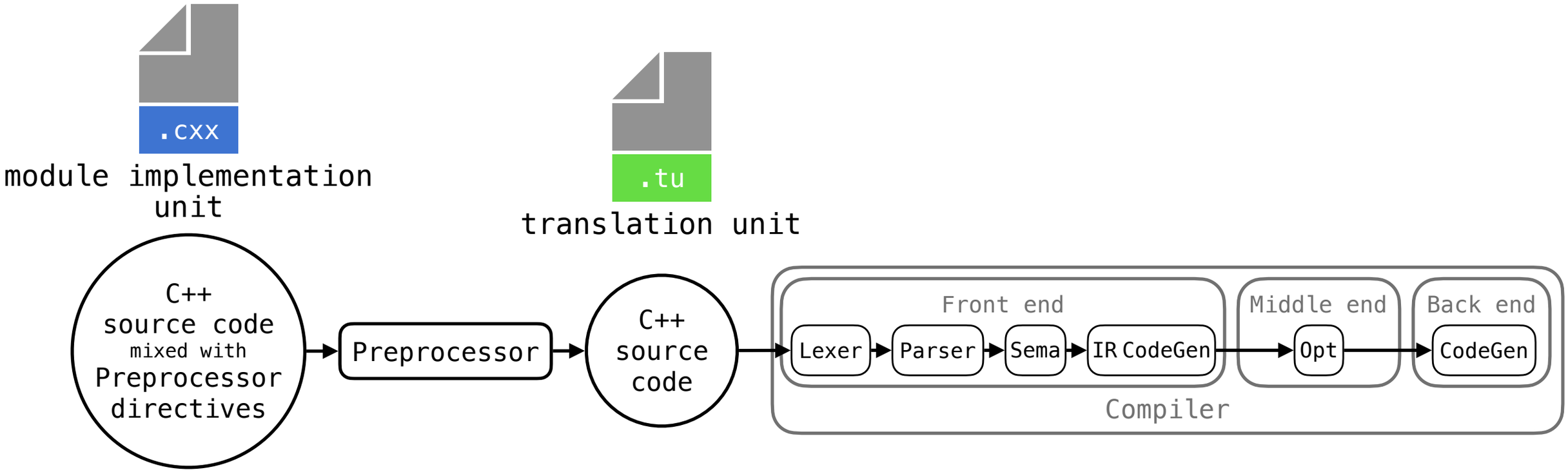
Modular Import Model

Module Implementation Unit - Preprocessing & Compilation



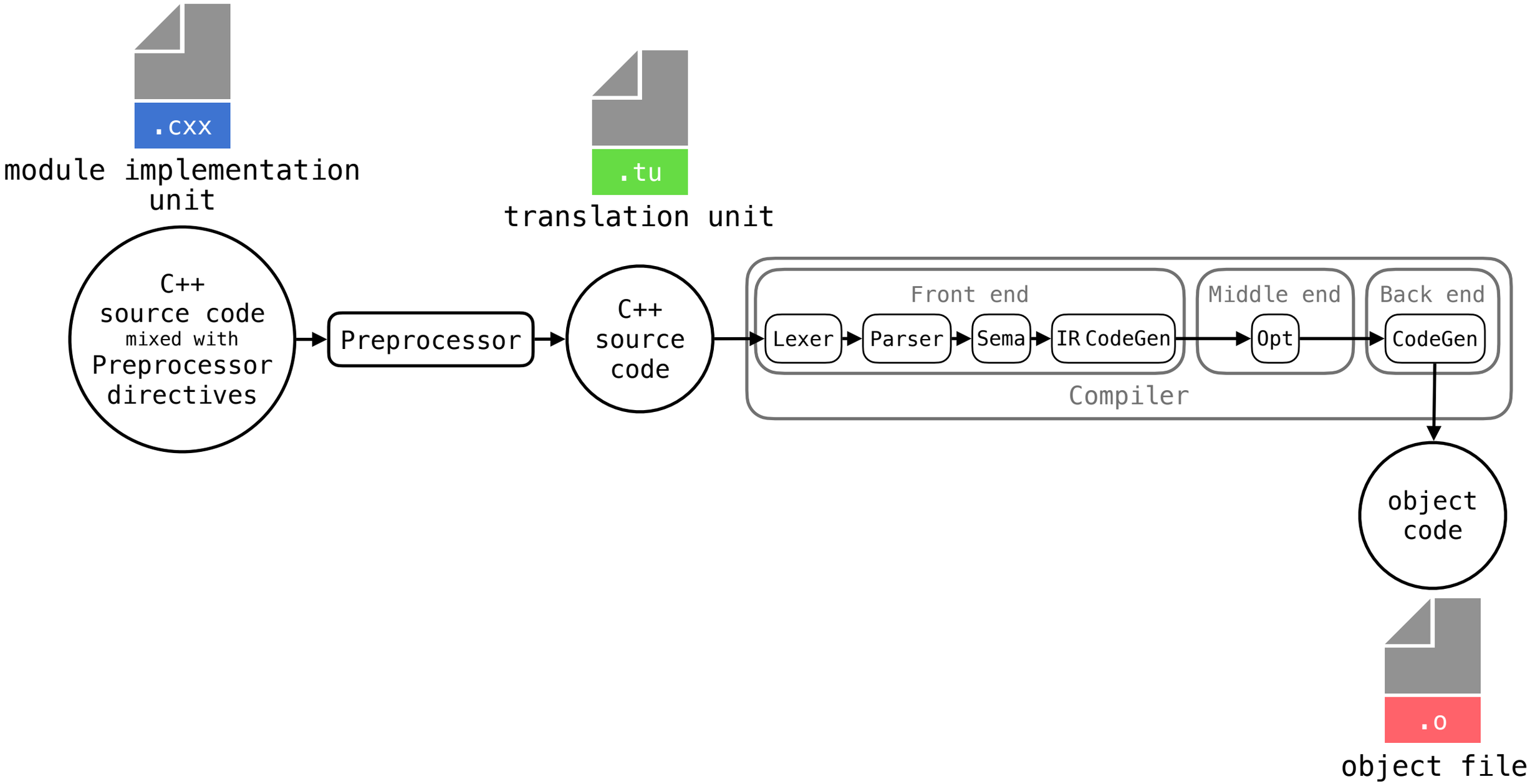
Modular Import Model

Module Implementation Unit - Preprocessing & Compilation



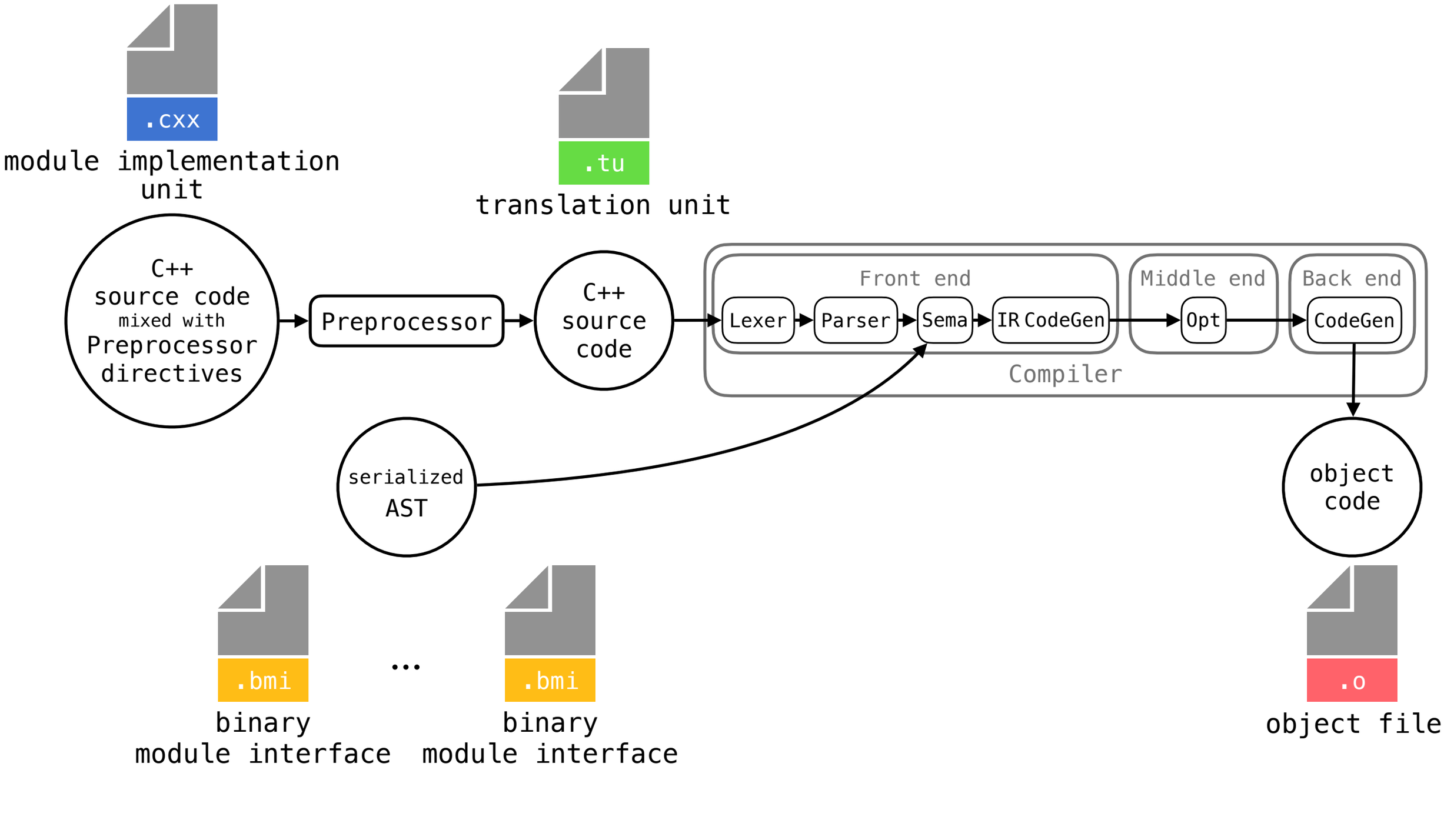
Modular Import Model

Module Implementation Unit - Preprocessing & Compilation



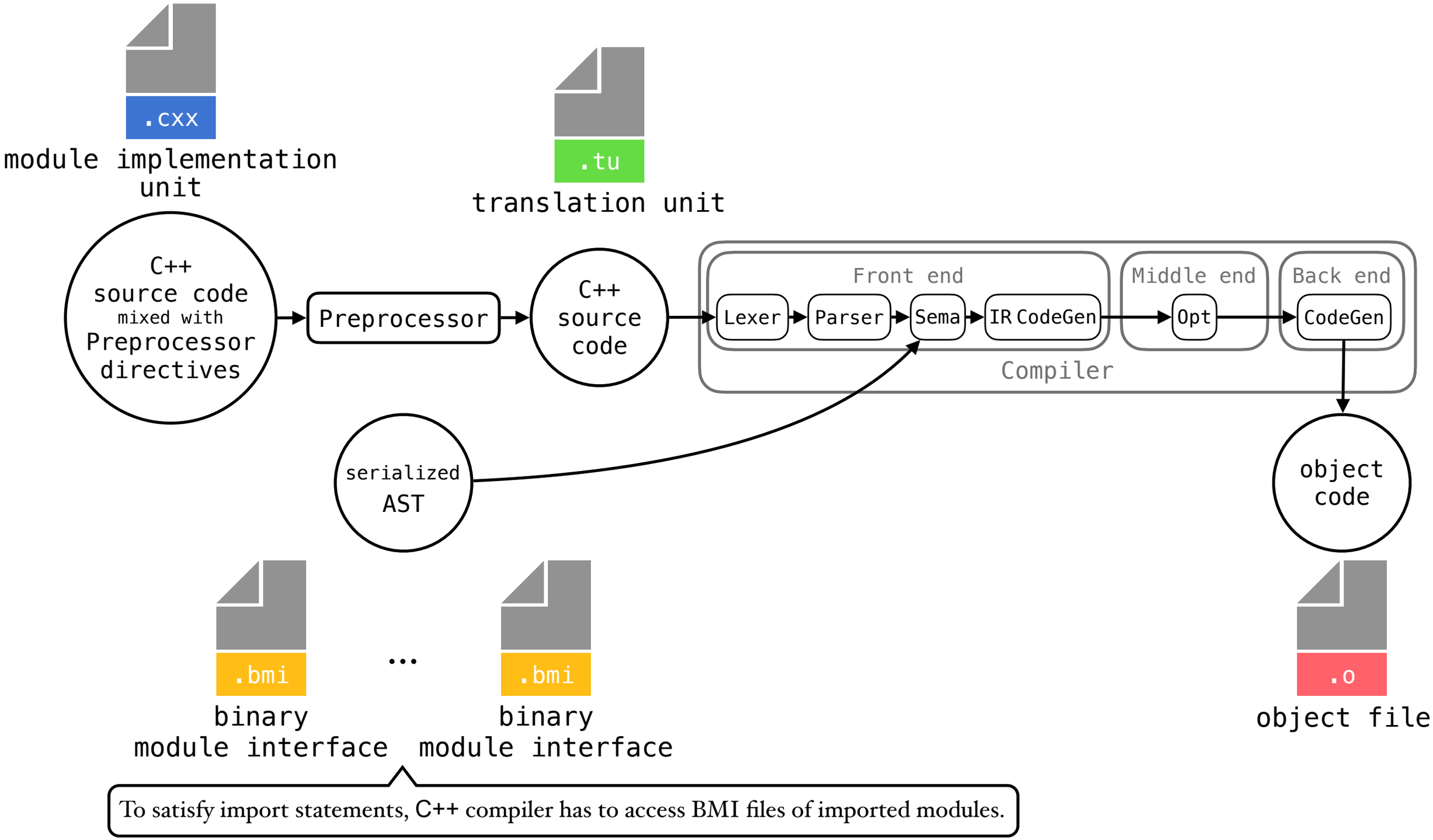
Modular Import Model

Module Implementation Unit - Preprocessing & Compilation



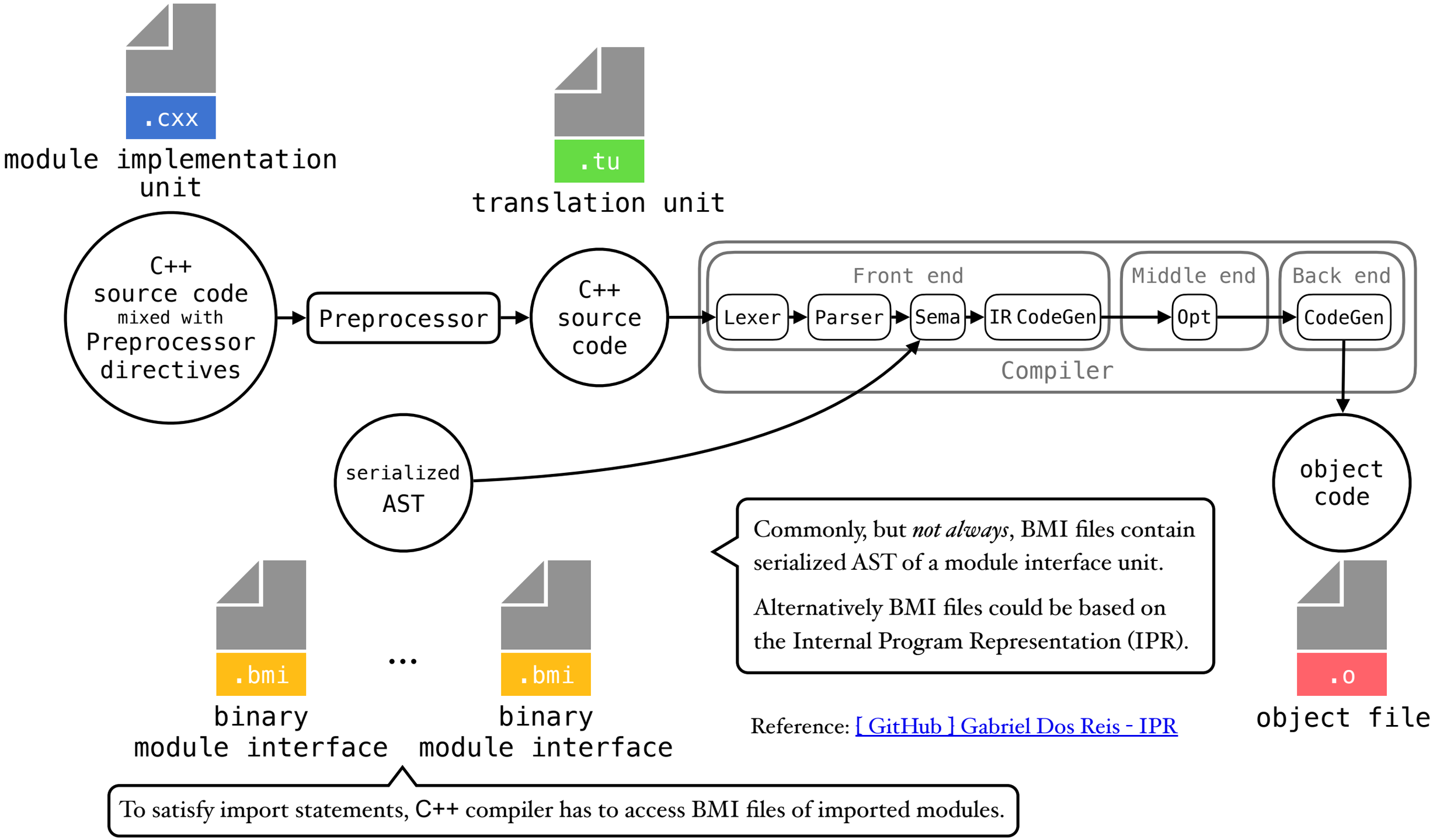
Modular Import Model

Module Implementation Unit - Preprocessing & Compilation



Modular Import Model

Module Implementation Unit - Preprocessing & Compilation

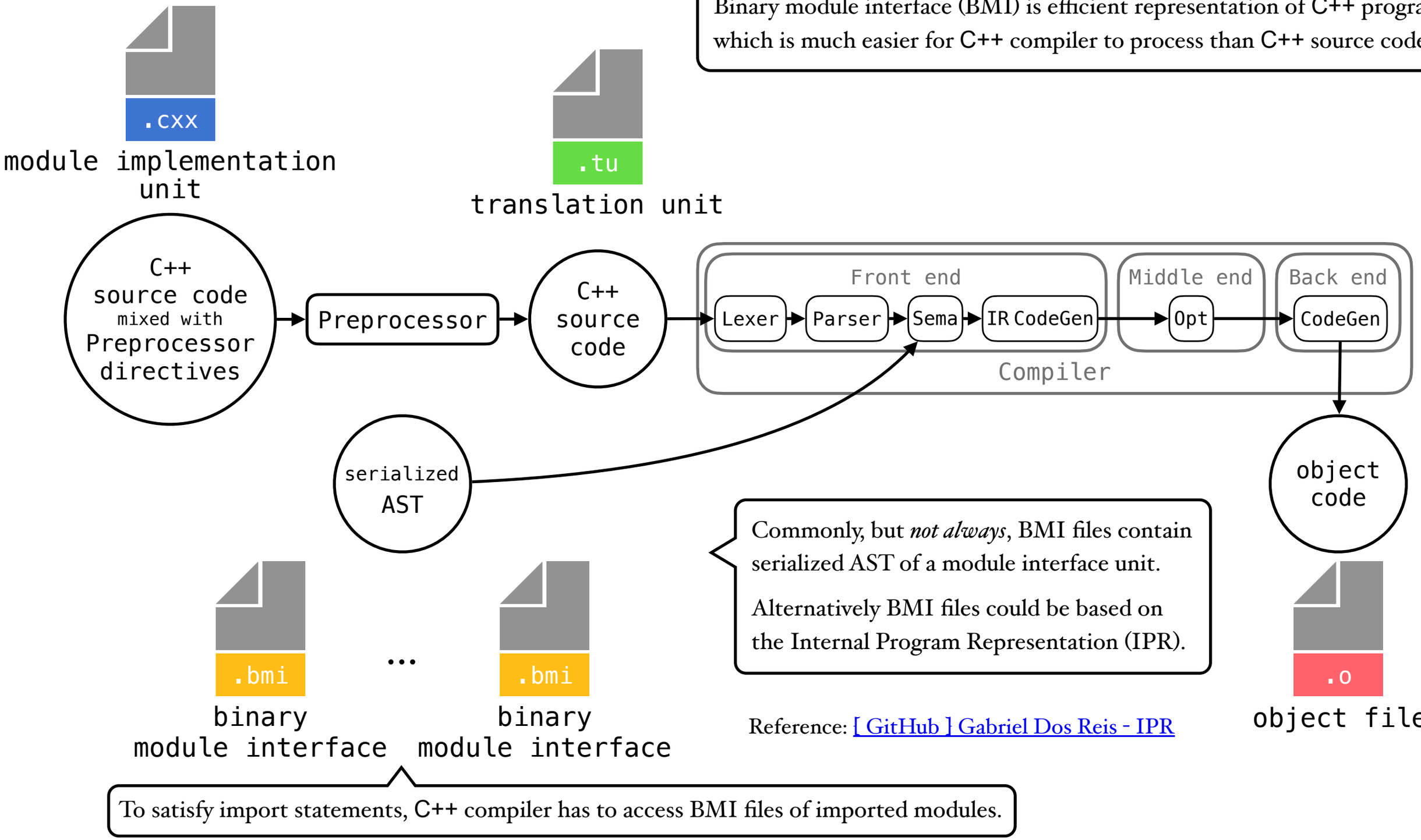


Reference: [GitHub] Gabriel Dos Reis - IPR

Modular Import Model

Module Implementation Unit - Preprocessing & Compilation

Binary module interface (BMI) is efficient representation of C++ programs, which is much easier for C++ compiler to process than C++ source code.



Commonly, but *not always*, BMI files contain serialized AST of a module interface unit.

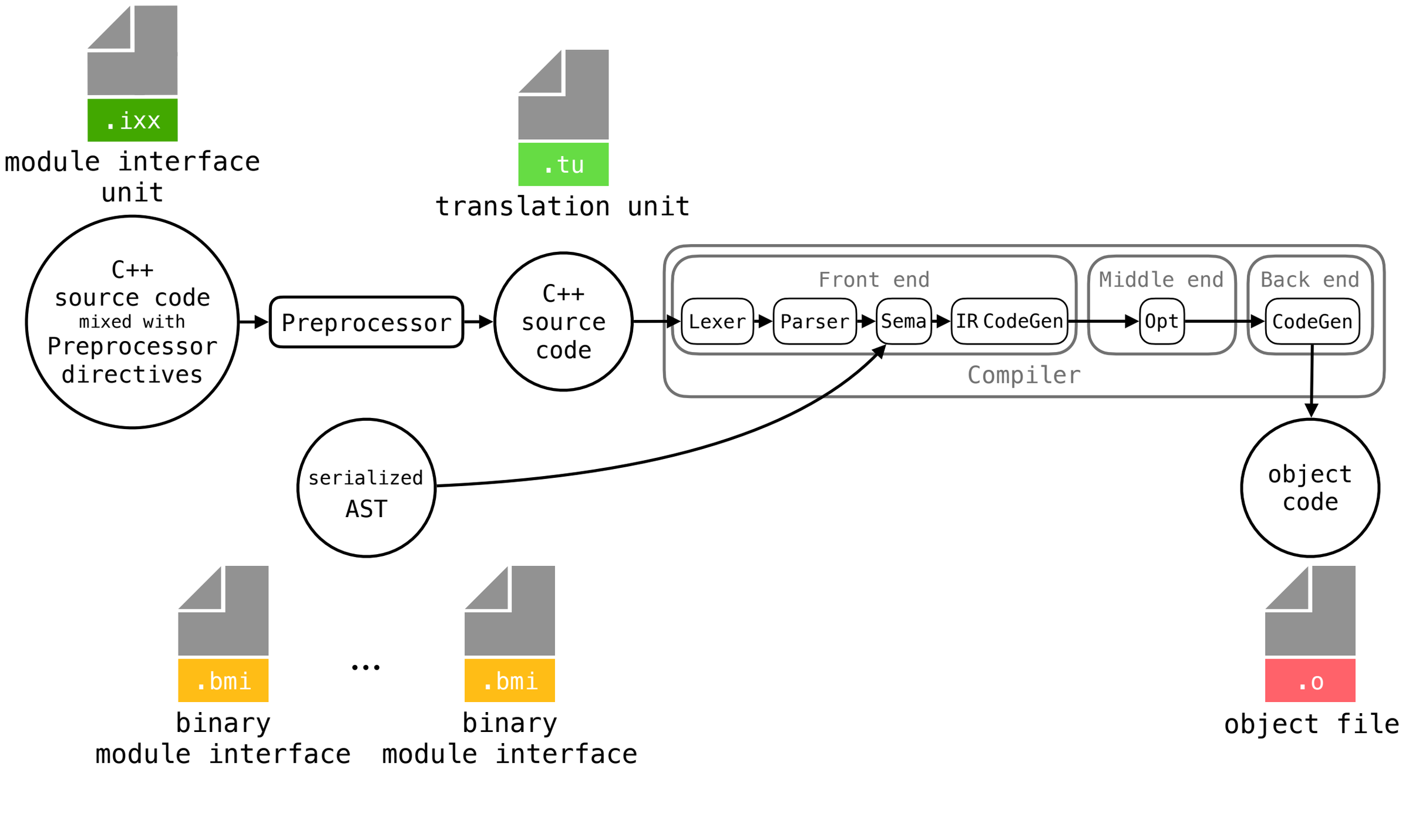
Alternatively BMI files could be based on the Internal Program Representation (IPR).

Reference: [GitHub] Gabriel Dos Reis - IPR

To satisfy import statements, C++ compiler has to access BMI files of imported modules.

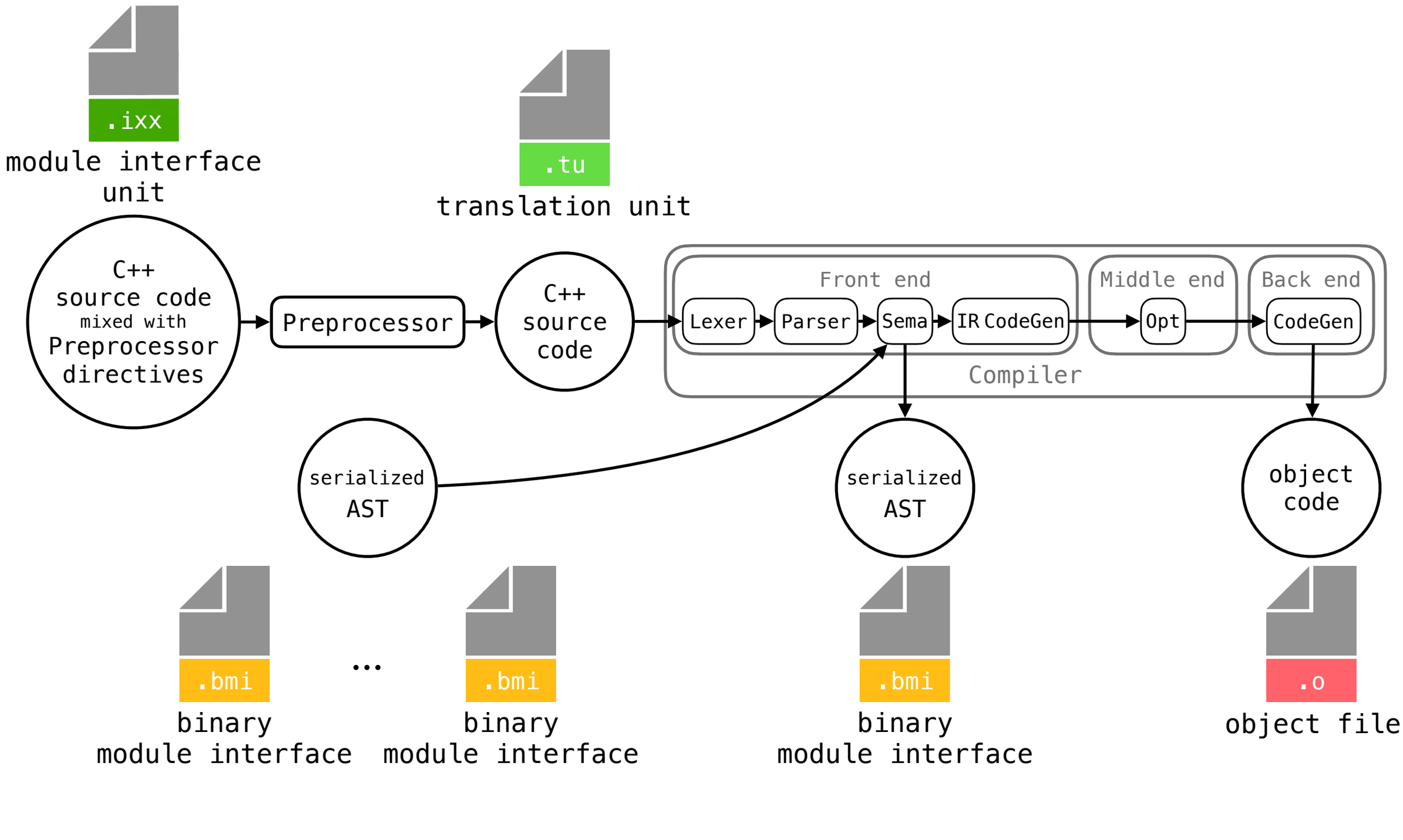
Modular Import Model

Module Interface Unit - Preprocessing & Compilation



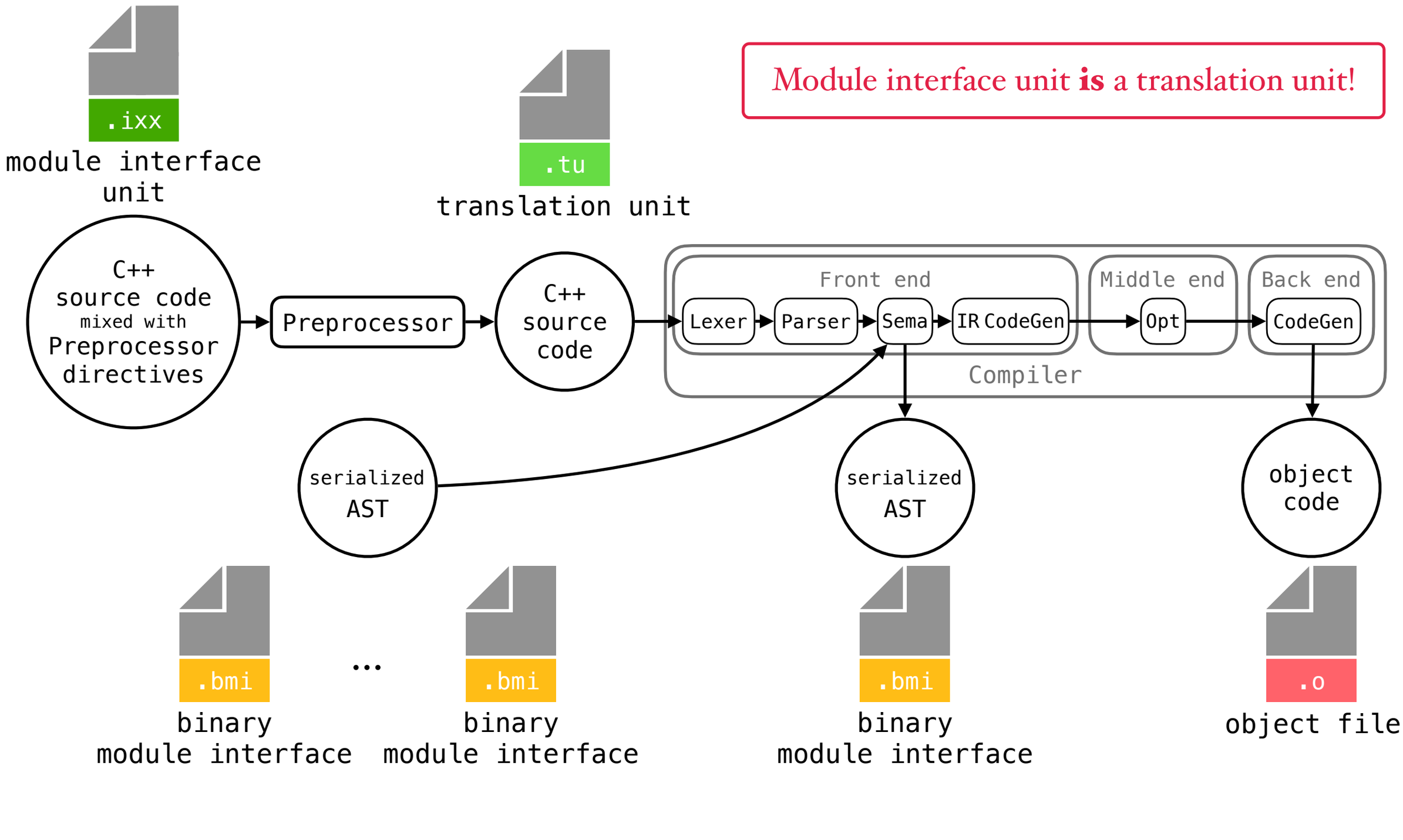
Modular Import Model

Module Interface Unit - Preprocessing & Compilation



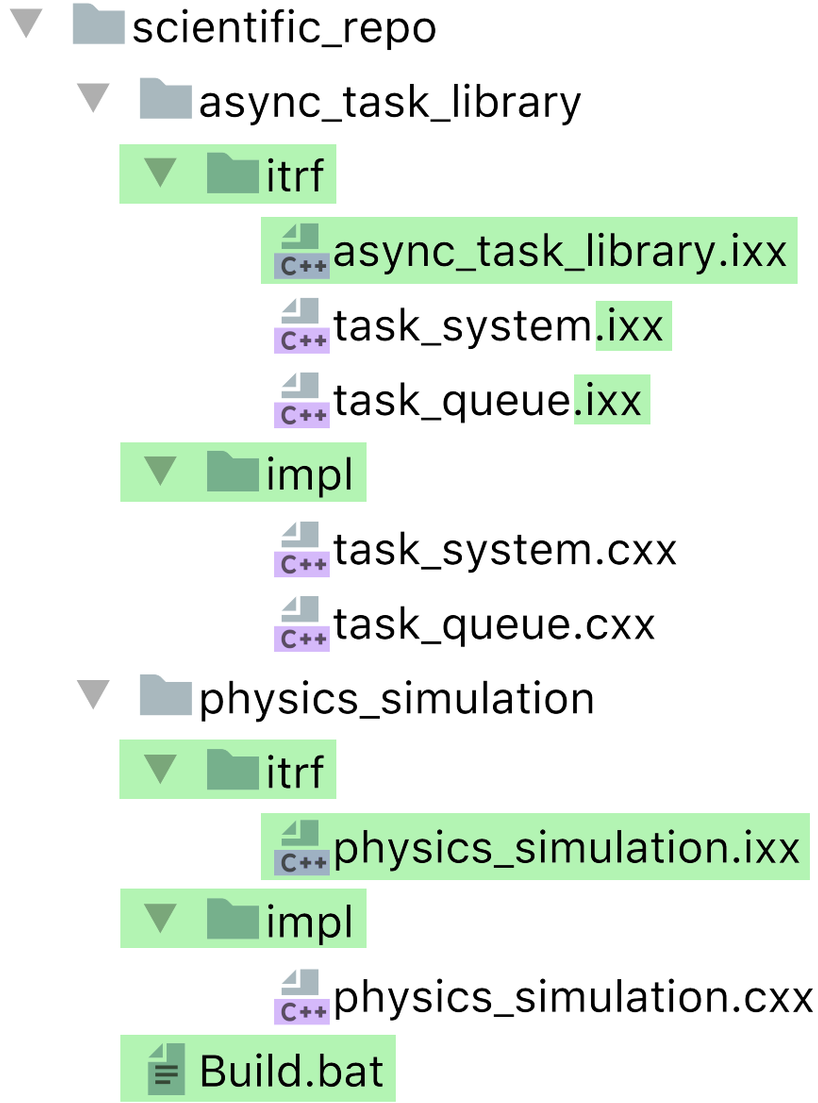
Modular Import Model

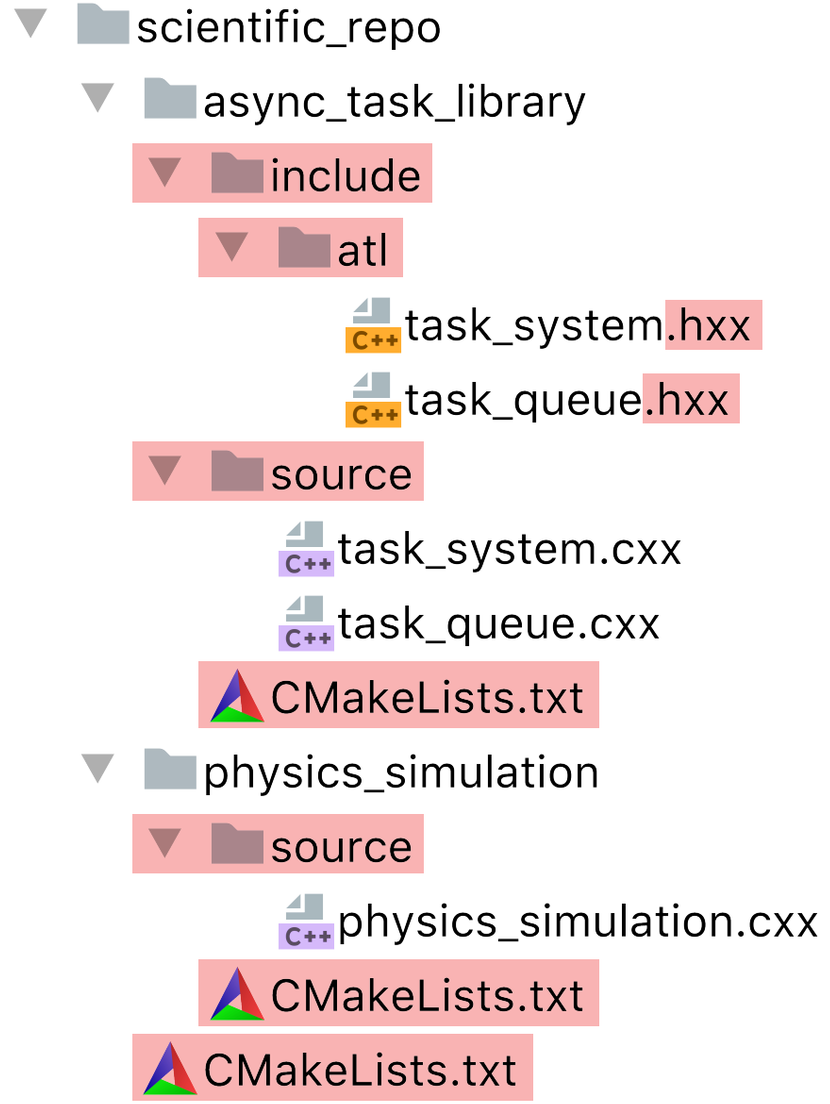
Module Interface Unit - Preprocessing & Compilation



Scientific Repo

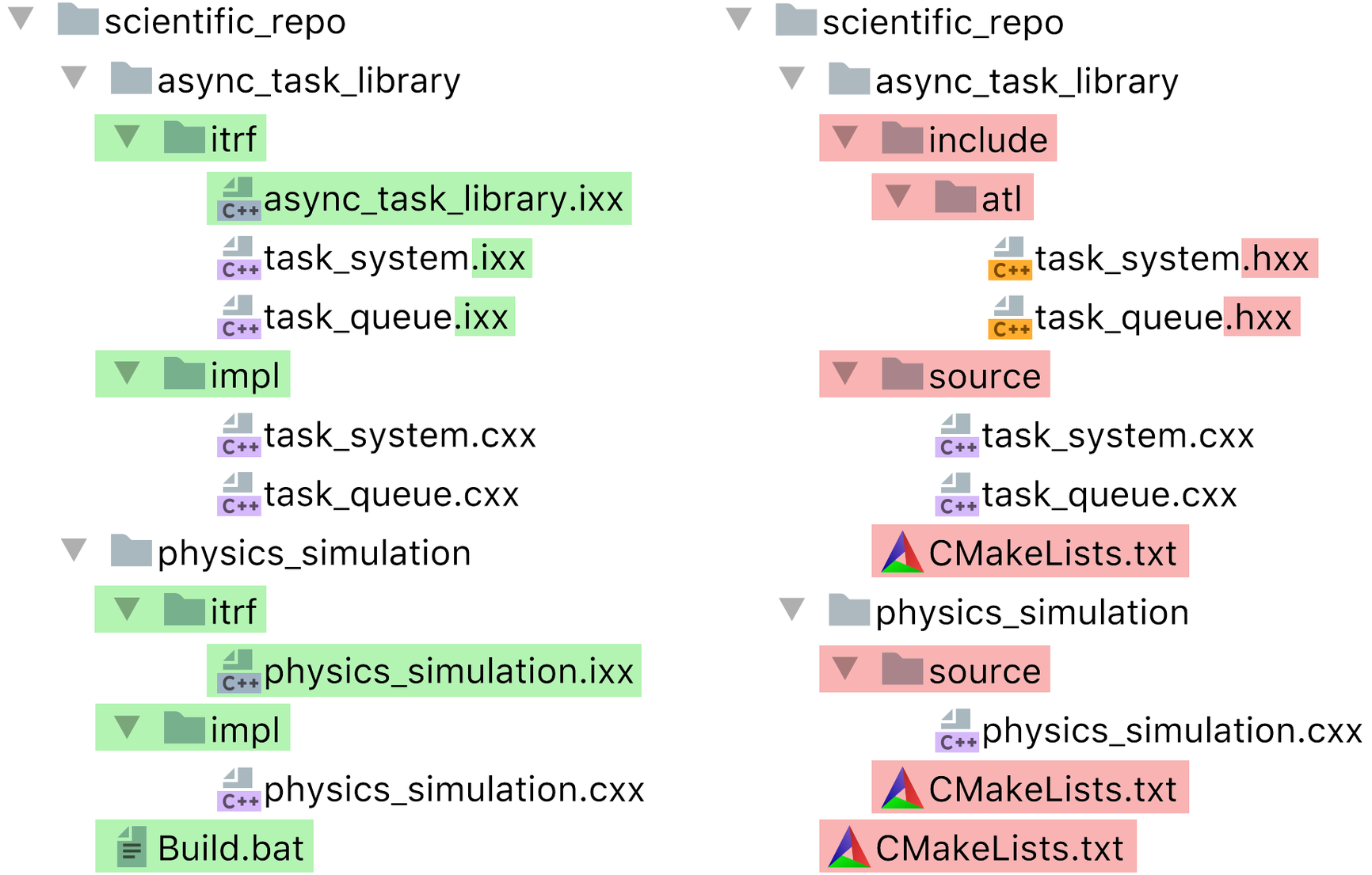
Project Organisation





Scientific Repo

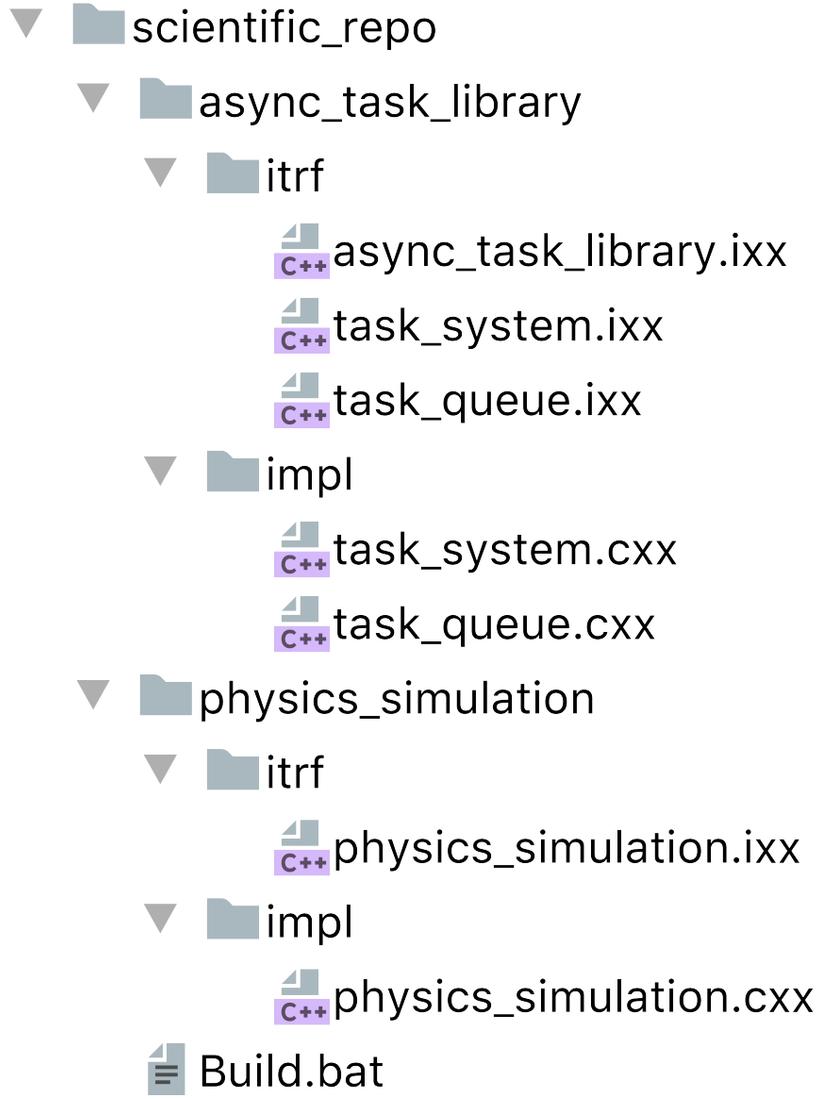
Project Organisation



Currently `build2` is the only build system supporting C++ modules, so as a temporary WA I've implemented batch script `Build.bat`, which calls MSVC compiler and linker directly.

Physics Simulation

Module Interface Unit




```
//physics_simulation/itrf/physics_simulation.ixx
export module phy_sim;

export auto main () -> int;
```

Physics Simulation

Module Implementation Unit

//physics_simulation/source/physics_simulation.cxx

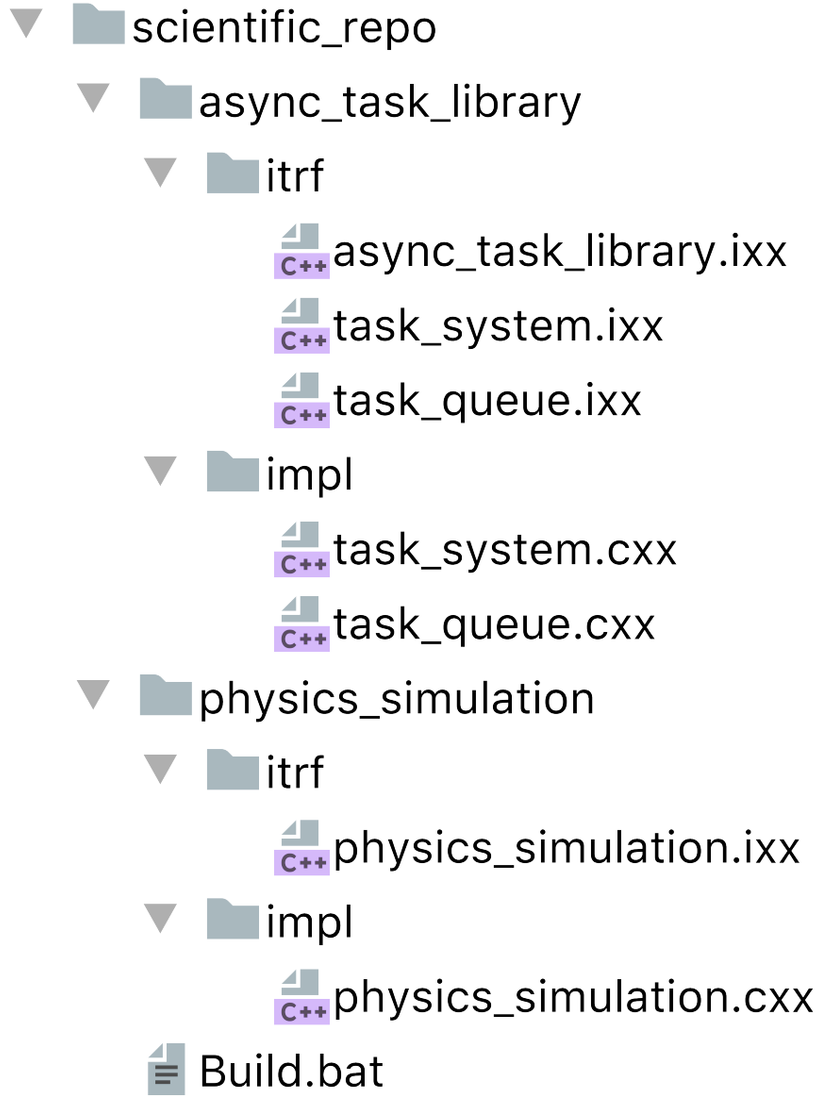
```
#include <atl/task_system.hxx>
#include <cmath>



namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float { ... }

    template <typename element_type>
    using array = ...;

    auto read_input () -> array<vec3> { ... }

    auto compute_lengths (const array<vec3>& vectors)
    -> array<float>
    { ... }

    auto print_results (const array<float>& lengths)
    -> void
    { ... }
}

auto main () -> int
{ ... }
```

//physics_simulation/impl/physics_simulation.cxx

```
module phy_sim;

import atl;
import std.core;

namespace phy_sim
{
    struct vec3 { float x, y, z; };

    auto length (const vec3& v) -> float { ... }

    template <typename element_type>
    using array = ...;

    auto read_input () -> array<vec3> { ... }

    auto compute_lengths (const array<vec3>& vectors)
    -> array<float>
    { ... }

    auto print_results (const array<float>& lengths)
    -> void
    { ... }
}

auto main () -> int
{ ... }
```

Async Task Library

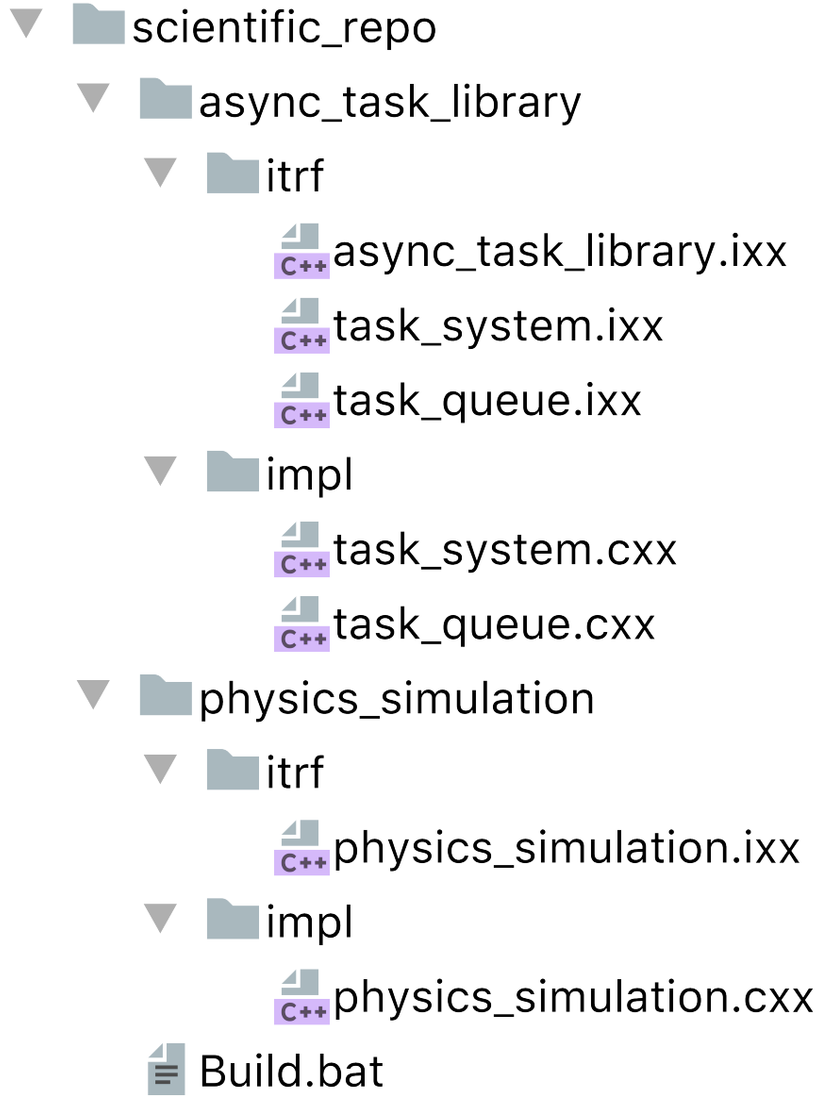
Module Interface Unit




```
//async_task_library/itrf/async_task_library.ixx
export module atl;

export import :task_system;
```

Async Task Library

Module Interface Unit




```
//async_task_library/itrf/async_task_library.ixx
export module atl;

export import :task_system;
```

Define the interface of the whole `atl` module,
by providing to clients interface of the
`atl:task_system` module partition, that is reexport it.

Async Task Library

Task Queue - Module Partition Interface Unit

//async_task_library/include/atl/task_queue.hxx

```
#ifndef ATL_TASK_QUEUE
#define ATL_TASK_QUEUE

#include <deque>
#include <mutex>
#include <condition_variable>
#include <functional>
#include <utility>
#include <optional>


namespace atl
{
    class task_queue
    {
        std::deque<std::function<void()>> tasks;
        std::mutex                        mutex;
        std::condition_variable           ready;
        bool                              done { false };

    public:
        auto finish () -> void;

        auto pop () -> std::optional<std::function<void()>>;

        template <typename callable>
        auto push (callable&& task) -> void { ... }
    };
}
#endif
```

//async_task_library/itrf/task_queue.ixx

```
export module atl:task_queue;

import std.core;
import std.threading;


namespace atl
{
    class task_queue
    {
        std::deque<std::function<void()>> tasks;
        std::mutex                        mutex;
        std::condition_variable           ready;
        bool                              done { false };

    public:
        auto finish () -> void;

        auto pop () -> std::optional<std::function<void()>>;

        template <typename callable>
        auto push (callable&& task) -> void { ... }
    };
}
```

Async Task Library

Tak Queue - Module Partition Implementation Unit

//async_task_library/source/task_queue.cxx

```
#include <atl/task_queue.hxx>

namespace atl
{
    auto task_queue::finish() -> void
    { ... }

    auto task_queue::pop ()
    -> std::optional<std::function<void()>>
    { ... }
}
```

//async_task_library/impl/task_queue.cxx

```
module atl:task_queue;

namespace atl
{
    auto task_queue::finish() -> void
    { ... }

    auto task_queue::pop ()
    -> std::optional<std::function<void()>>
    { ... }
}
```

Async Task Library

Task System - Module Partition Interface Unit

//async_task_library/include/atl/task_system.hxx

```
#ifndef ATL_TASK_SYSTEM
#define ATL_TASK_SYSTEM

#include <atl/task_queue.hxx>
#include <vector>
#include <thread>

namespace atl
{
    class task_system
    {
    private:
        std::vector<std::thread> threads;
        task_queue               queue;

    public:
        task_system ();
        ~task_system ();

        template <typename callable>
        auto async (callable&& task) -> void
        { ... }
    };
}

#endif
```

//async_task_library/itrf/task_system.ixx

```
export module atl:task_system;

import :task_queue;

namespace atl
{
    export class task_system
    {
    private:
        std::vector<std::thread> threads;
        task_queue               queue;

    public:
        task_system ();
        ~task_system ();

        template <typename callable>
        auto async (callable&& task) -> void
        { ... }
    };
}
```

Async Task Library

Task System - Module Partition Interface Unit

//async_task_library/include/atl/task_system.hxx

```
#ifndef ATL_TASK_SYSTEM
#define ATL_TASK_SYSTEM

#include <atl/task_queue.hxx>
#include <vector>
#include <thread>

namespace atl
{
    class task_system
    {
    private:
        std::vector<std::thread> threads;
        task_queue               queue;

    public:
        task_system ();
        ~task_system ();

        template <typename callable>
        auto async (callable&& task) -> void
        { ... }
    };
}

#endif
```

//async_task_library/itrf/task_system.ixx

```
export module atl:task_system;

import :task_queue;
```

Internal dependency inside module between module partitions.

```
namespace atl
{
    export class task_system
    {
    private:
        std::vector<std::thread> threads;
        task_queue               queue;

    public:
        task_system ();
        ~task_system ();

        template <typename callable>
        auto async (callable&& task) -> void
        { ... }
    };
}
```

Async Task Library

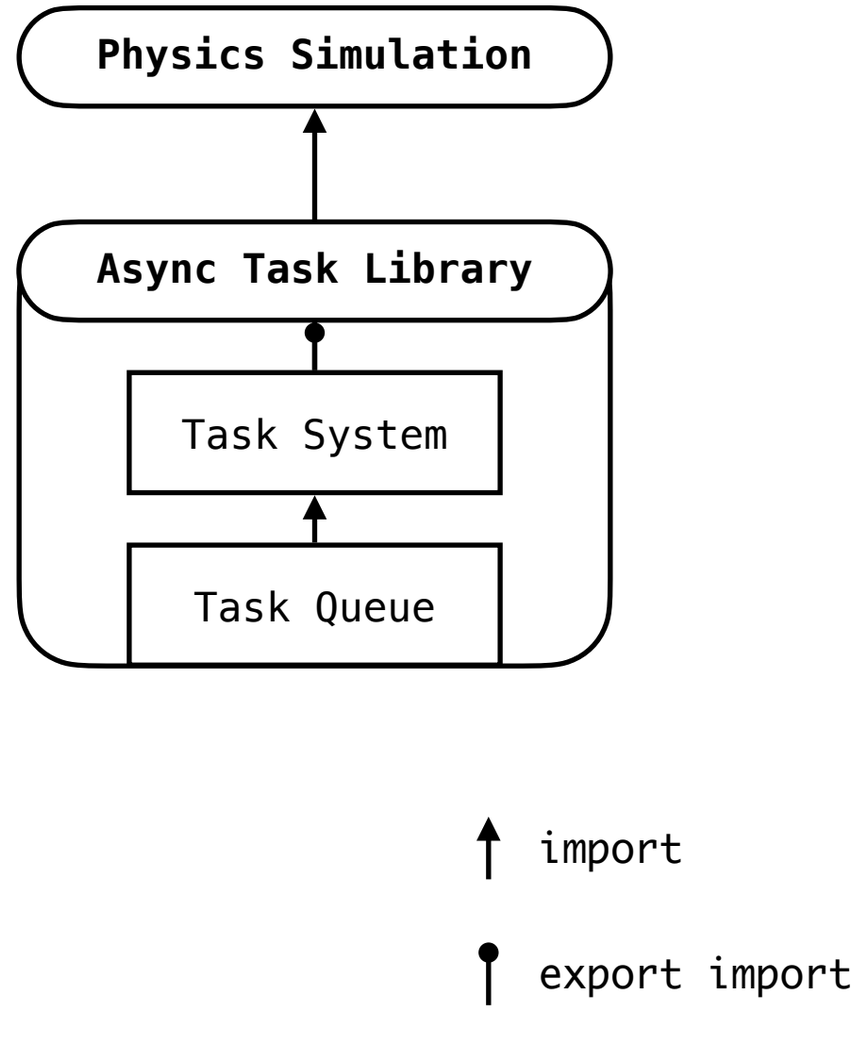
Task System - Module Partition Implementation Unit

```
//async_task_library/source/task_system.cxx
#include <atl/task_system.hxx>

namespace atl
{
    namespace
    {
        auto run_thread (task_queue& queue)
        -> void { ... }
    }

    task_system::task_system ()
    { ... }

    task_system::~task_system ()
    { ... }
}
```

```
//async_task_library/impl/task_system.cxx
module atl:task_system;

namespace atl
{
    namespace
    {
        auto run_thread (task_queue& queue)
        -> void { ... }
    }

    task_system::task_system ()
    { ... }

    task_system::~task_system ()
    { ... }
}
```

Modular Import Model

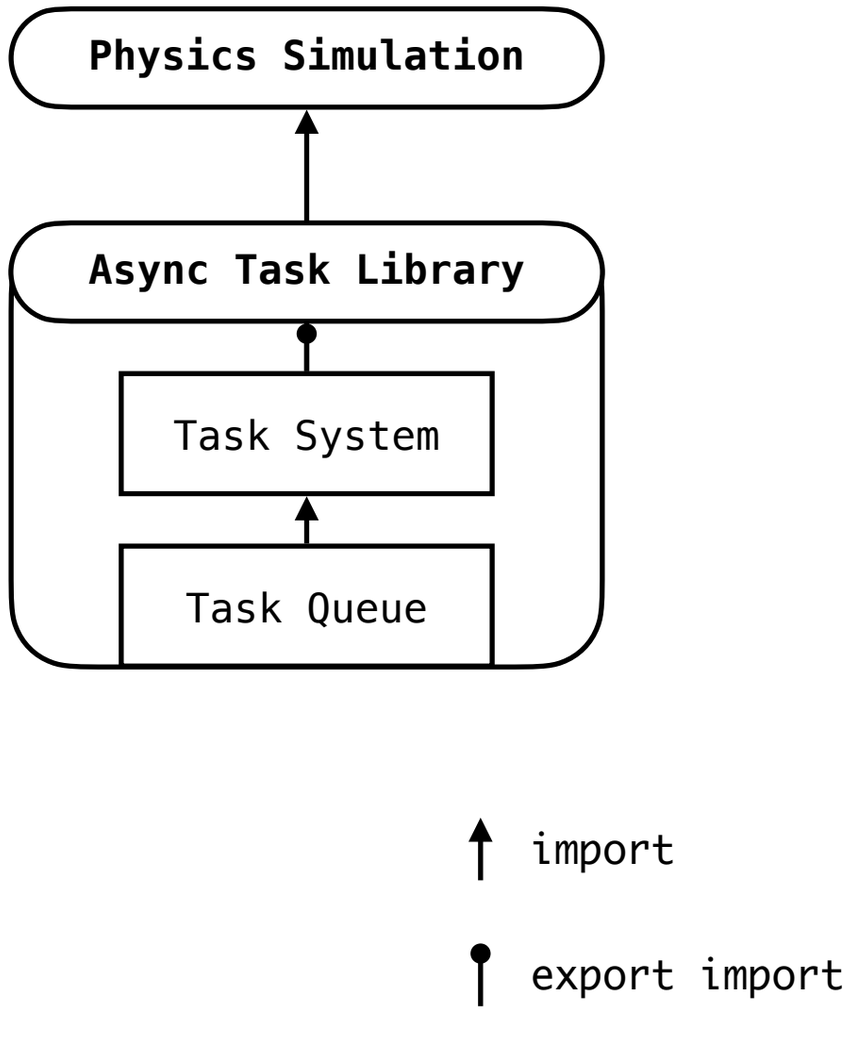
Dependency Graph



Modular Import Model

Dependency Graph

- Import statements form dependency graph between modules.



Modular Import Model

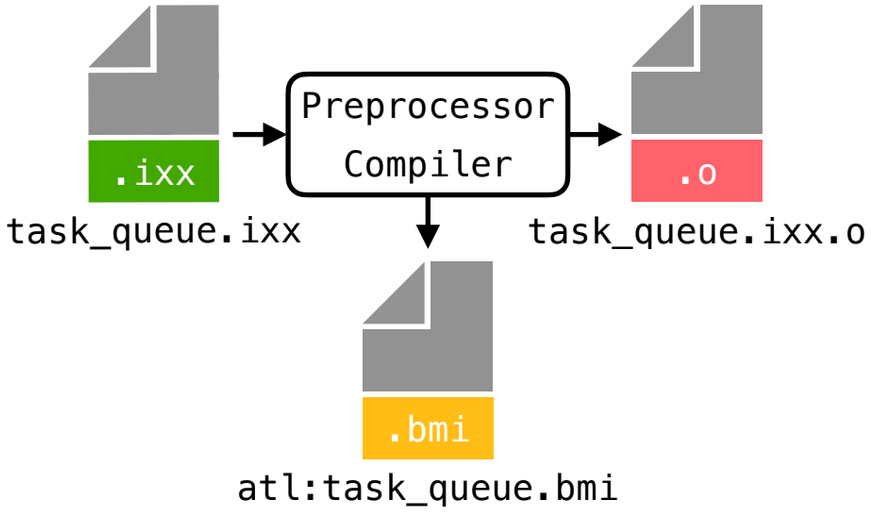
Dependency Graph

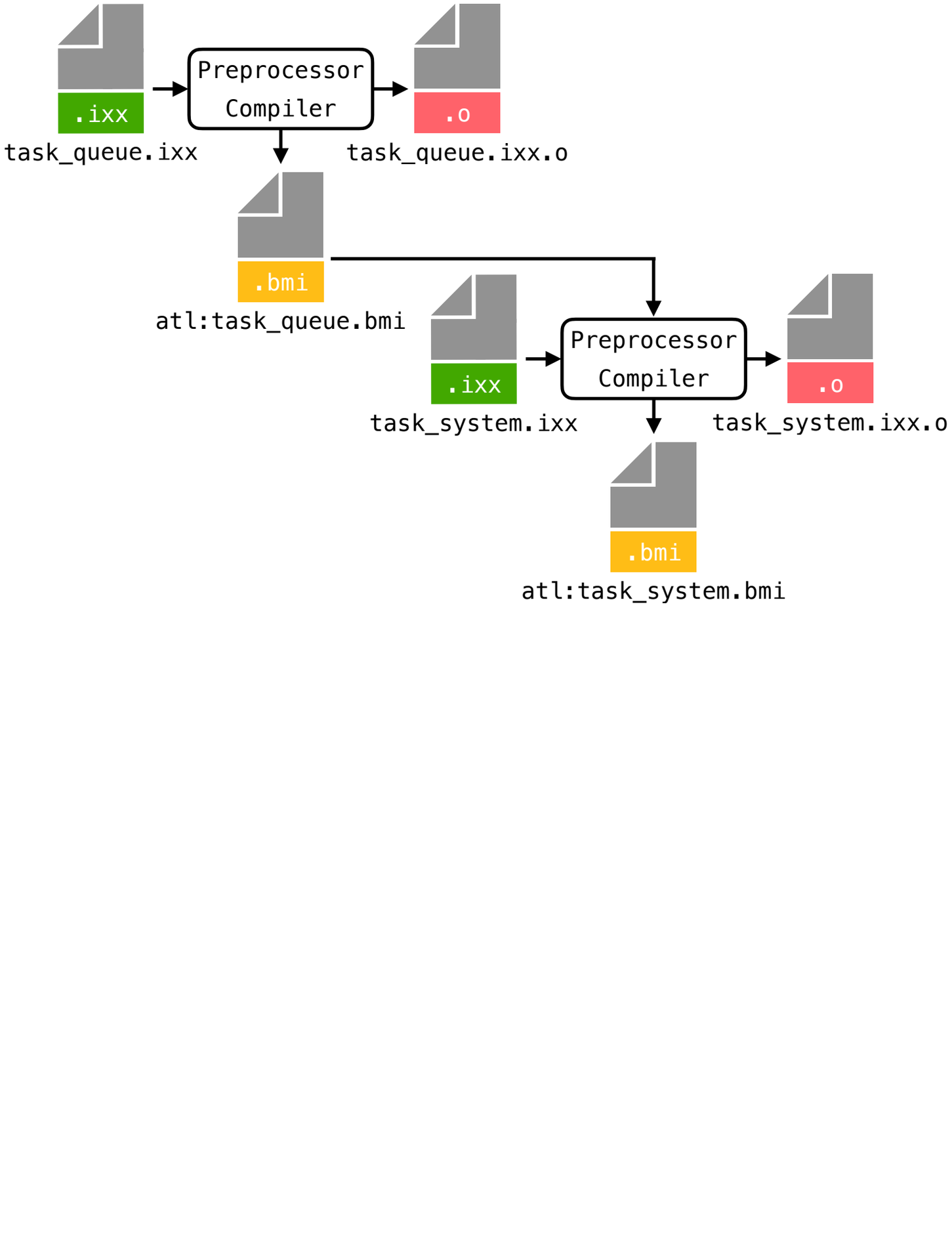
- Import statements form dependency graph between modules.
- Module dependencies cannot be cyclic, which C++ compilers enforce at compile-time.

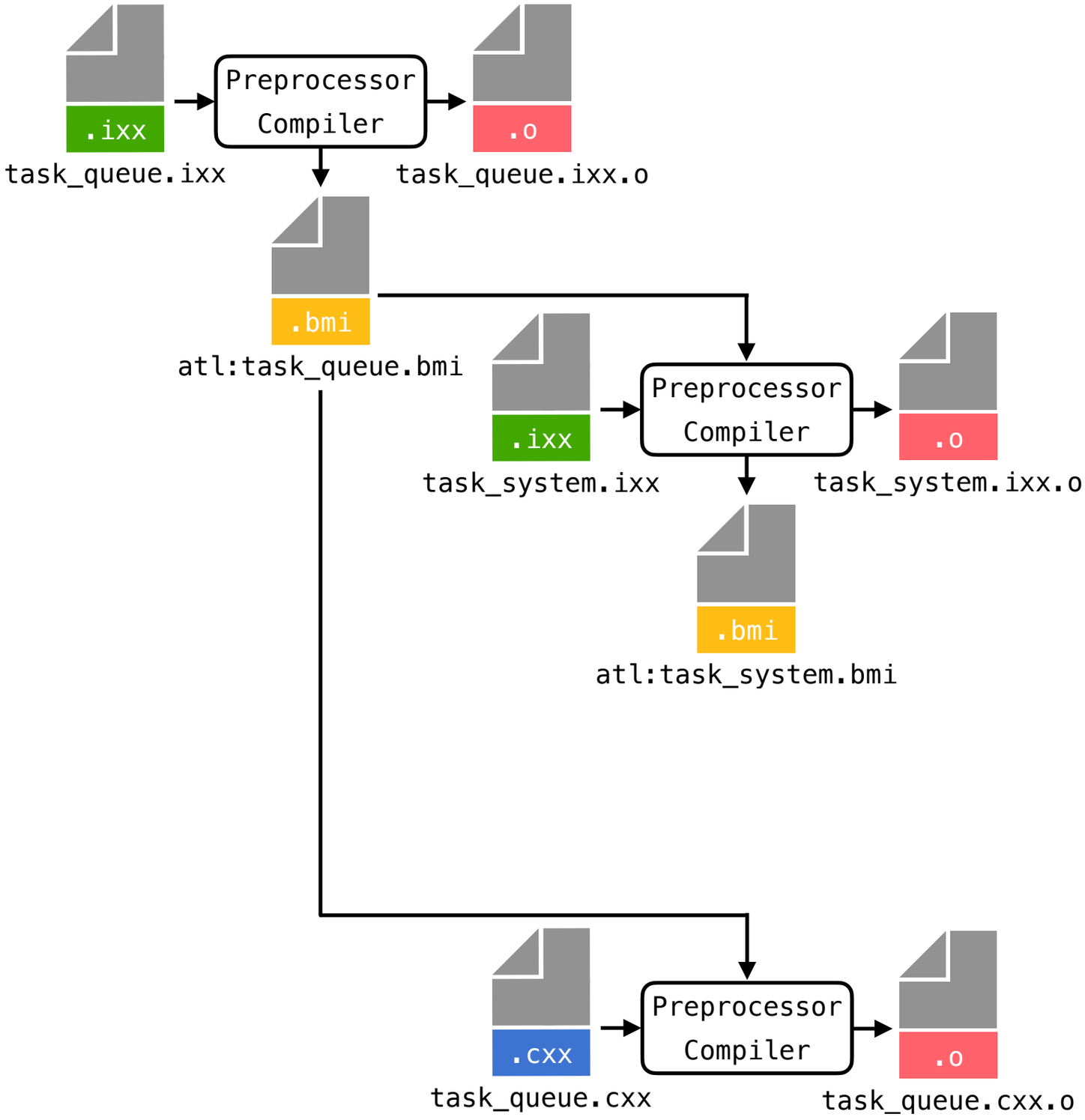
Modular Import Model

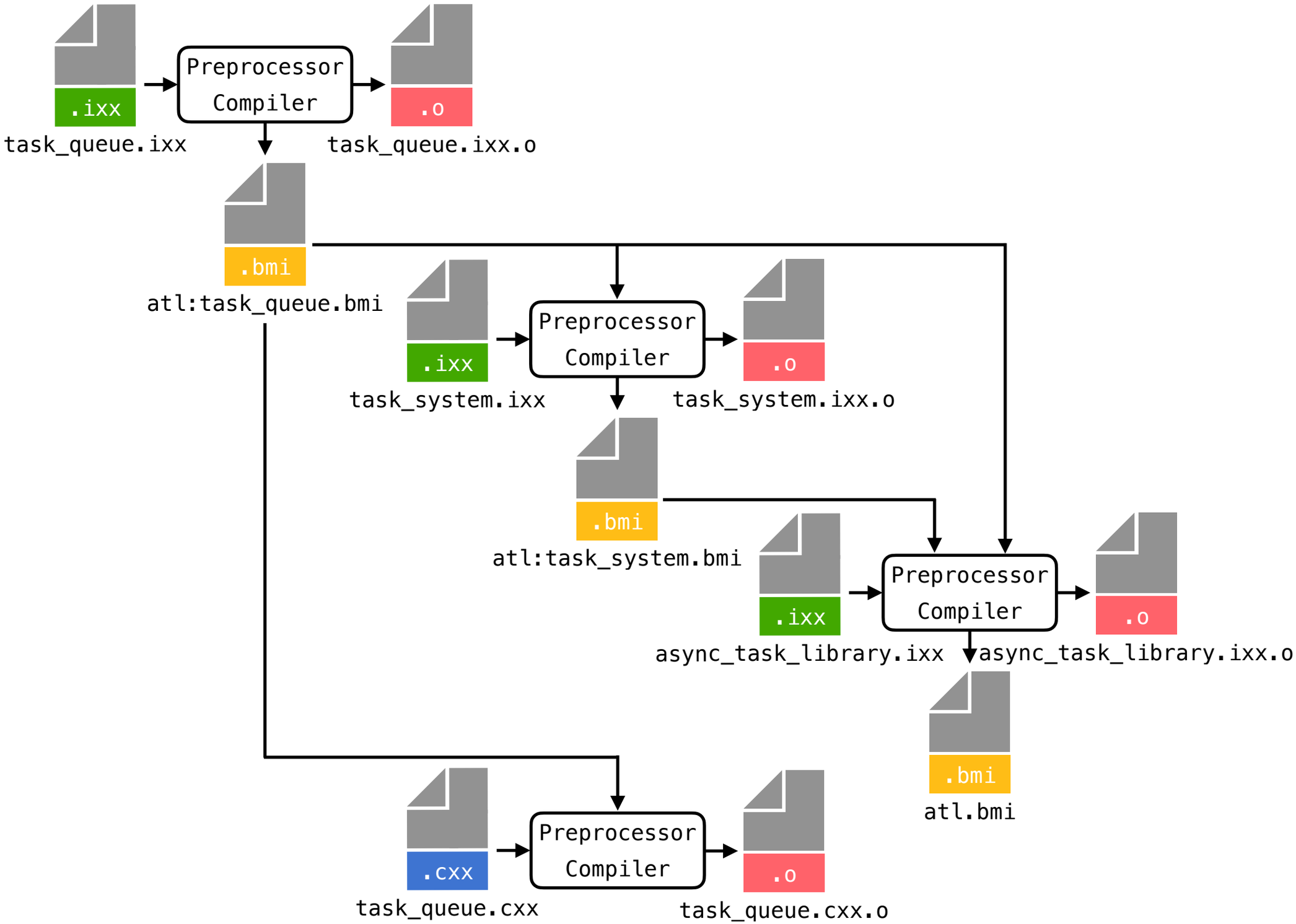
Dependency Graph

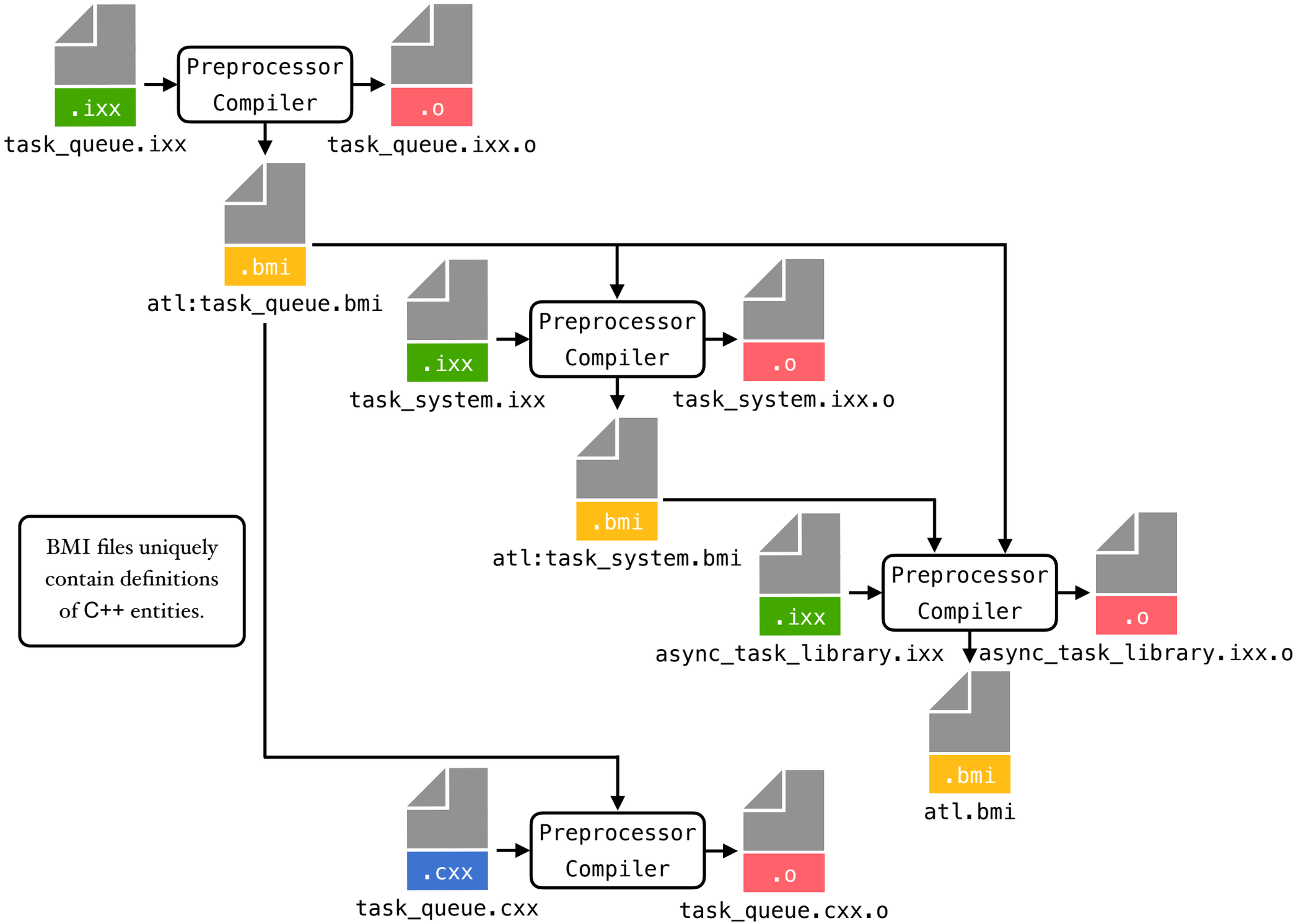
- Import statements form dependency graph between modules.
- Module dependencies cannot be cyclic, which C++ compilers enforce at compile-time.
- In short, modules dependency graphs are DAGs.

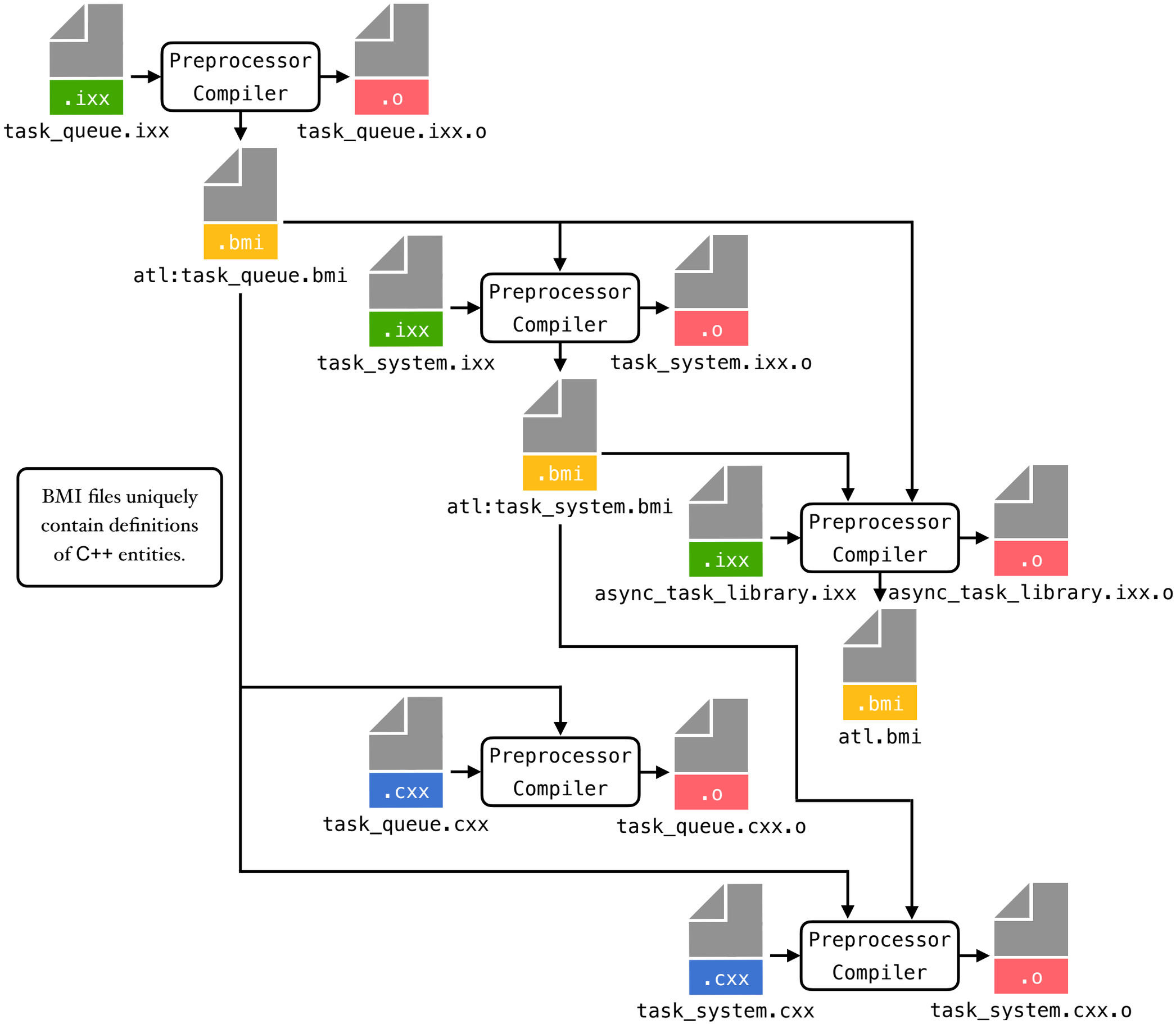

.ixx
Preprocessor
Compiler
.o
task_queue.ixx
task_queue.ixx.o
.bmi
atl:task_queue.bmi

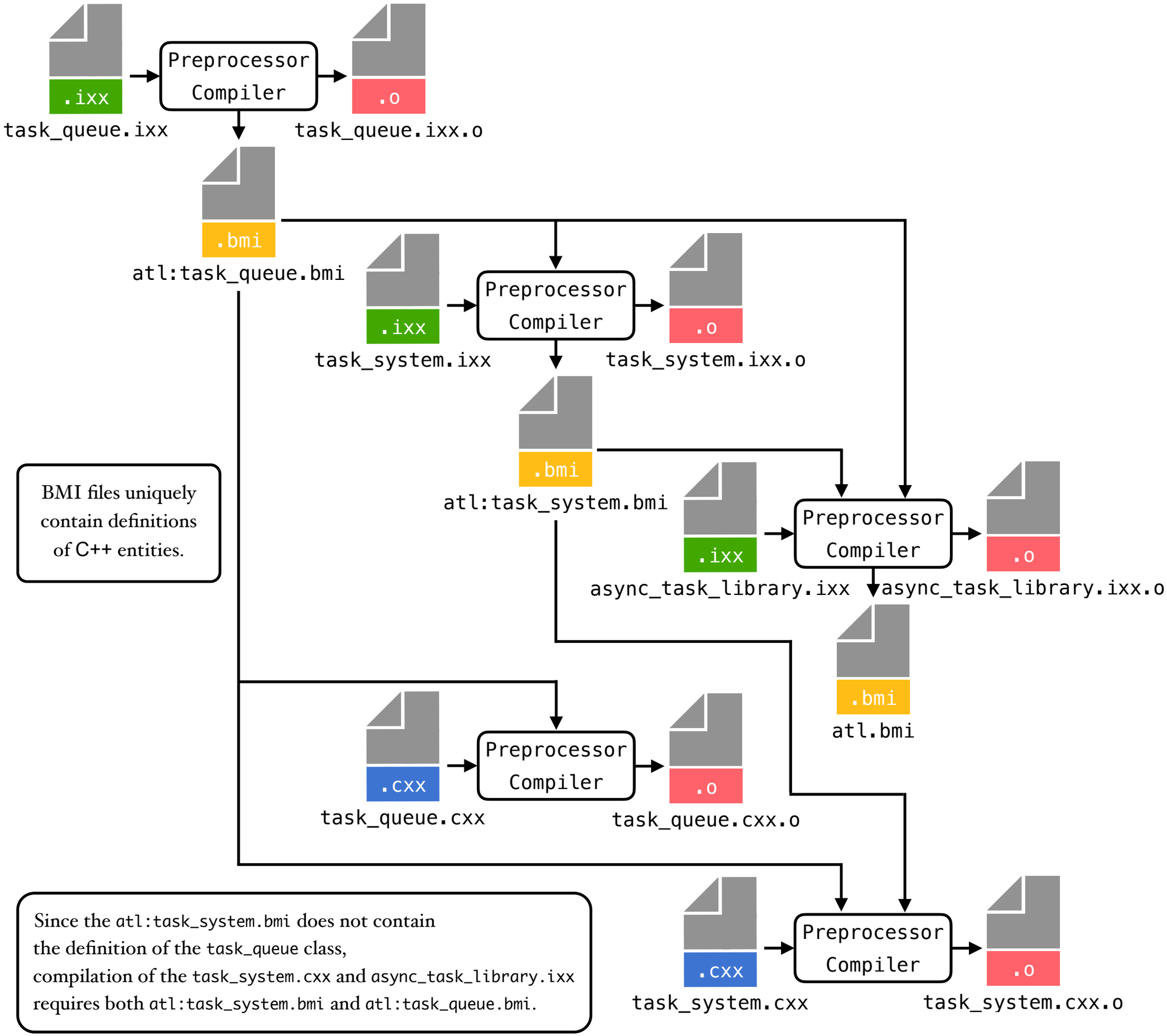

.ixx
Preprocessor
Compiler
.o
task_queue.ixx
task_queue.ixx.o
.bmi
atl:task_queue.bmi
.ixx
Preprocessor
Compiler
.o
task_system.ixx
task_system.ixx.o
.bmi
atl:task_system.bmi

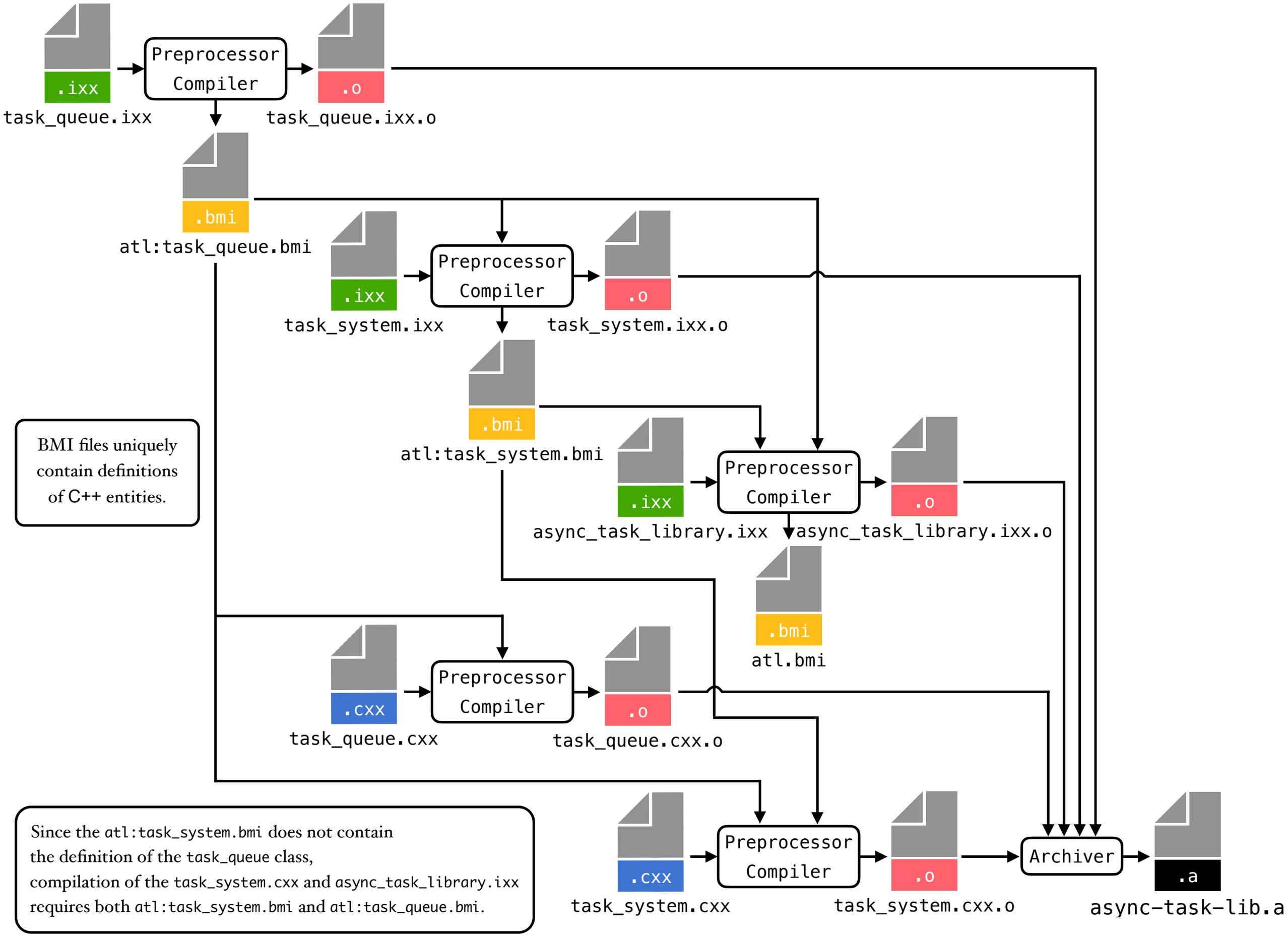

.ixx
Preprocessor
Compiler
.o
task_queue.ixx
task_queue.ixx.o
.bmi
atl:task_queue.bmi
.ixx
Preprocessor
Compiler
.o
task_system.ixx
task_system.ixx.o
.bmi
atl:task_system.bmi
.cxx
Preprocessor
Compiler
.o
task_queue.cxx
task_queue.cxx.o


.ixx
Preprocessor
Compiler
.o
task_queue.ixx
task_queue.ixx.o
.bmi
atl:task_queue.bmi
.ixx
Preprocessor
Compiler
.o
task_system.ixx
task_system.ixx.o
.bmi
atl:task_system.bmi
.ixx
Preprocessor
Compiler
.o
async_task_library.ixx
async_task_library.ixx.o
.bmi
atl.bmi
.cxx
Preprocessor
Compiler
.o
task_queue.cxx
task_queue.cxx.o


.ixx
Preprocessor
Compiler
.o
task_queue.ixx
task_queue.ixx.o
.bmi
atl:task_queue.bmi
.ixx
Preprocessor
Compiler
.o
task_system.ixx
task_system.ixx.o
.bmi
atl:task_system.bmi
.ixx
Preprocessor
Compiler
.o
async_task_library.ixx
async_task_library.ixx.o
.bmi
atl.bmi
BMI files uniquely contain definitions of C++ entities.
.cxx
Preprocessor
Compiler
.o
task_queue.cxx
task_queue.cxx.o

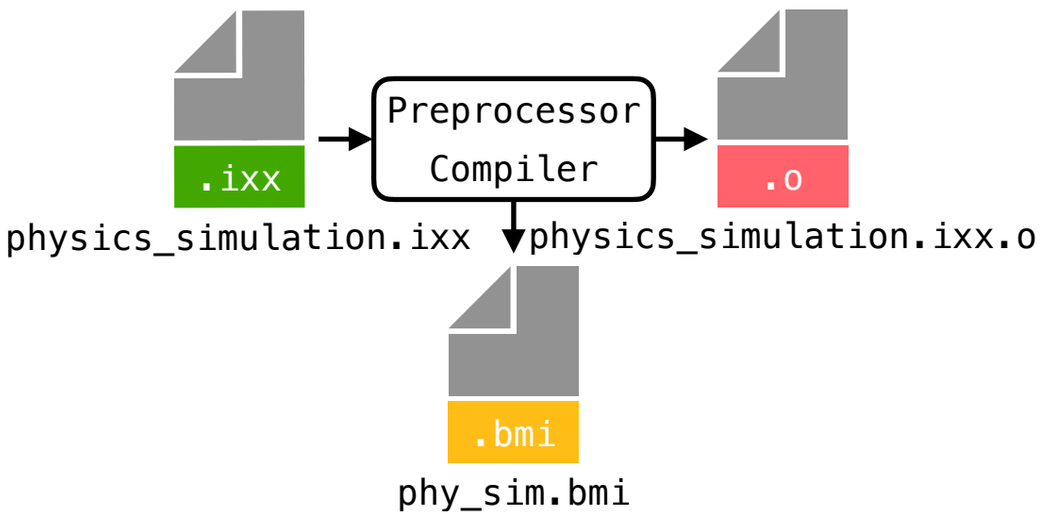

.ixx
Preprocessor
Compiler
.o
task_queue.ixx
task_queue.ixx.o
.bmi
atl:task_queue.bmi
.ixx
Preprocessor
Compiler
.o
task_system.ixx
task_system.ixx.o
.bmi
atl:task_system.bmi
.ixx
Preprocessor
Compiler
.o
async_task_library.ixx
async_task_library.ixx.o
.bmi
atl.bmi
BMI files uniquely contain definitions of C++ entities.
.cxx
Preprocessor
Compiler
.o
task_queue.cxx
task_queue.cxx.o
.cxx
Preprocessor
Compiler
.o
task_system.cxx
task_system.cxx.o


.ixx
Preprocessor
Compiler
.o
task_queue.ixx
task_queue.ixx.o
.bmi
atl:task_queue.bmi
.ixx
Preprocessor
Compiler
.o
task_system.ixx
task_system.ixx.o
.bmi
atl:task_system.bmi
.ixx
Preprocessor
Compiler
.o
async_task_library.ixx
async_task_library.ixx.o
.bmi
atl.bmi
BMI files uniquely contain definitions of C++ entities.
.cxx
Preprocessor
Compiler
.o
task_queue.cxx
task_queue.cxx.o
.cxx
Preprocessor
Compiler
.o
task_system.cxx
task_system.cxx.o
Since the atl:task_system.bmi does not contain the definition of the task_queue class, compilation of the task_system.cxx and async_task_library.ixx requires both atl:task_system.bmi and atl:task_queue.bmi.


.ixx
Preprocessor
Compiler
.o
task_queue.ixx
task_queue.ixx.o
.bmi
atl:task_queue.bmi
.ixx
Preprocessor
Compiler
.o
task_system.ixx
task_system.ixx.o
.bmi
atl:task_system.bmi
.ixx
Preprocessor
Compiler
.o
async_task_library.ixx
async_task_library.ixx.o
.bmi
atl.bmi
BMI files uniquely contain definitions of C++ entities.
.cxx
Preprocessor
Compiler
.o
task_queue.cxx
task_queue.cxx.o
Since the atl:task_system.bmi does not contain the definition of the task_queue class, compilation of the task_system.cxx and async_task_library.ixx requires both atl:task_system.bmi and atl:task_queue.bmi.
.cxx
Preprocessor
Compiler
.o
task_system.cxx
task_system.cxx.o
Archiver
.a
async-task-lib.a

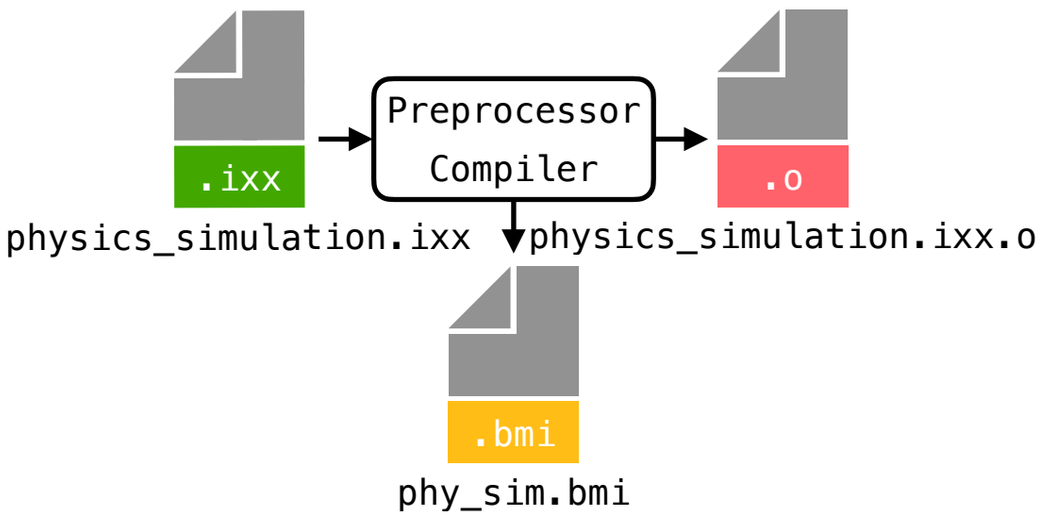
Modular Import Model

Build Workflow



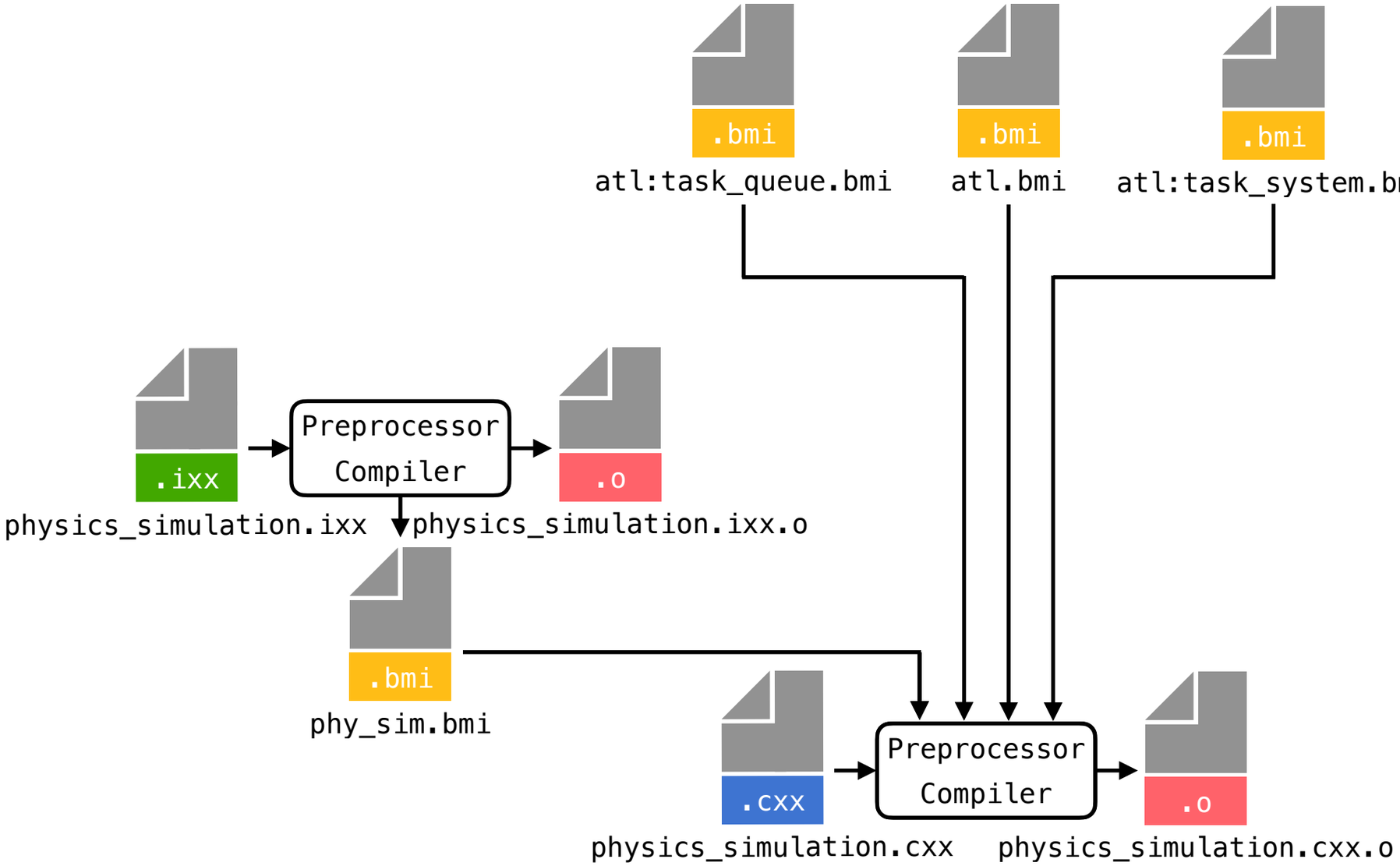
Modular Import Model

Build Workflow



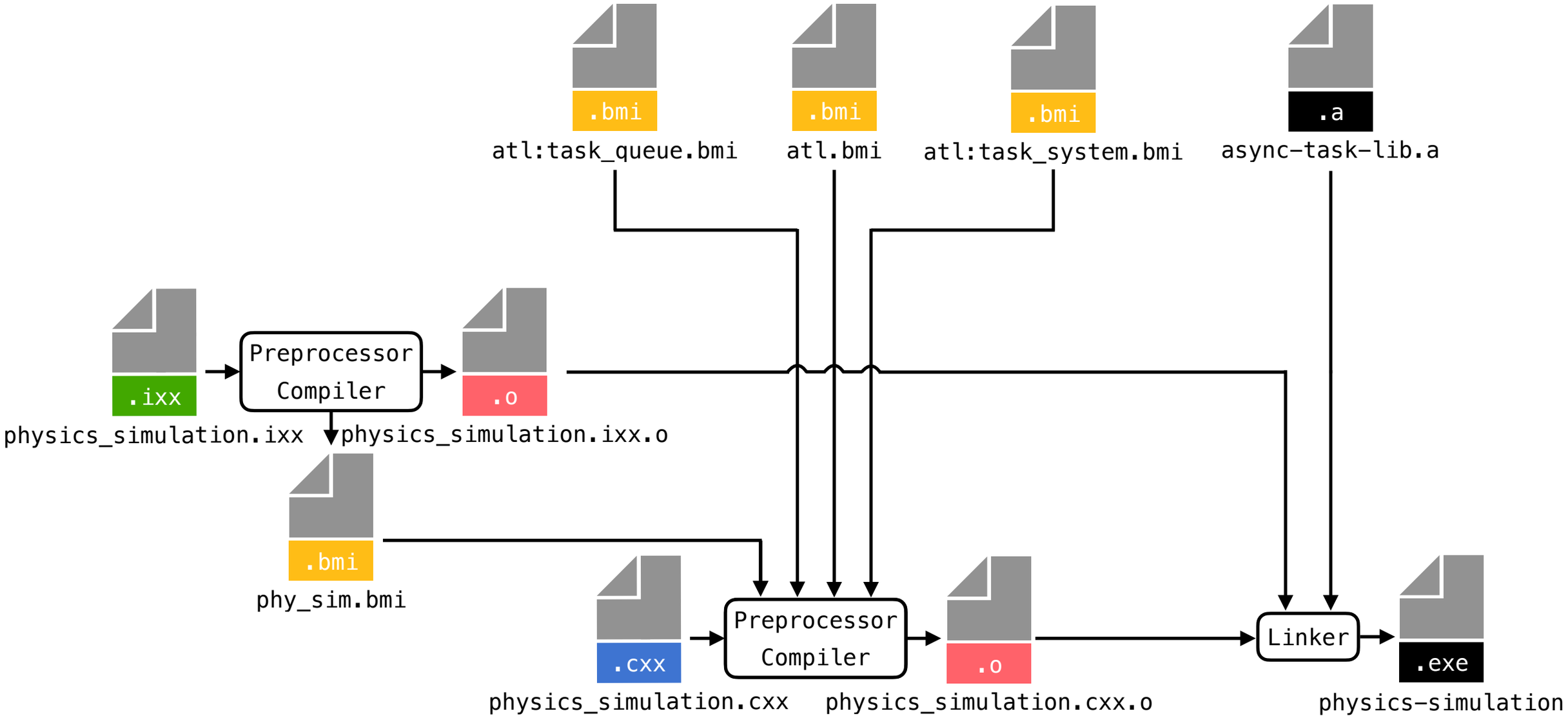
Modular Import Model

Build Workflow



Modular Import Model

Build Workflow



References

Conference Talks

- Core C++ 2019 :: Bryce Adelstein :: Modules are Coming
- Modules are coming - Bryce Adelstein Lelbach - Meeting Cpp 2019
- CppCon 2019: Gabriel Dos Reis “Programming with C++ Modules: Guide for the Working”
- CPPP 2019 - C++ Modules: What You Should Know - Gabriel Dos Reis
- CppCon 2015: Gabriel Dos Reis “Large Scale C++ with Modules: What You Should Know"
- CppCon 2016: Gabriel Dos Reis “C++ Modules: The State of The Union"
- 2019 LLVM Developers’ Meeting: E. Christopher & J. Doerfert “Introduction to LLVM”
- 2019 EuroLLVM Developers’ Meeting: V. Bridgers & F. Piovezan “LLVM IR Tutorial - Phis, GEPs ...”
- 2019 LLVM Developers’ Meeting: S. Haastregt & A. Stulova “An overview of Clang”

References

Documentation

- https://llvm.org/docs/BitCodeFormat.html
- https://llvm.org/docs/CommandGuide/index.html
- https://llvm.org/docs/LangRef.html
- http://llvm.org/docs/GetElementPtr.html
- https://clang.llvm.org/docs/InternalsManual.html
- https://clang.llvm.org/docs/LibASTImporter.html
- https://clang.llvm.org/docs/PCHInternals.html
- https://clang.llvm.org/docs/IntroductionToTheClangAST.html
- https://devblogs.microsoft.com/cppblog/c-modules-in-vs-2015-update-1/
- https://devblogs.microsoft.com/cppblog/cpp-modules-in-visual-studio-2017/
- https://devblogs.microsoft.com/cppblog/c-modules-conformance-improvements-with-msvc-in-visual-studio-2019-16-5/