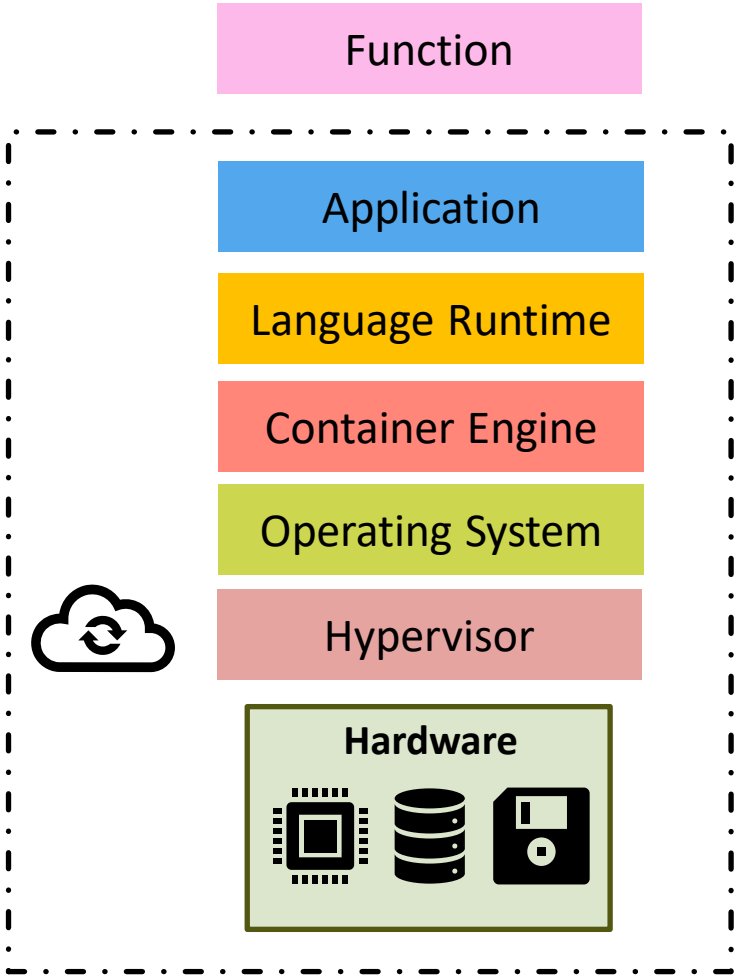
LUKAS MÖLLER, MARCIN COPIK, ALEXANDRU CALOTOIU, TORSTEN HOEFLER

Cppless: Productive and Performant Serverless Programming in C++

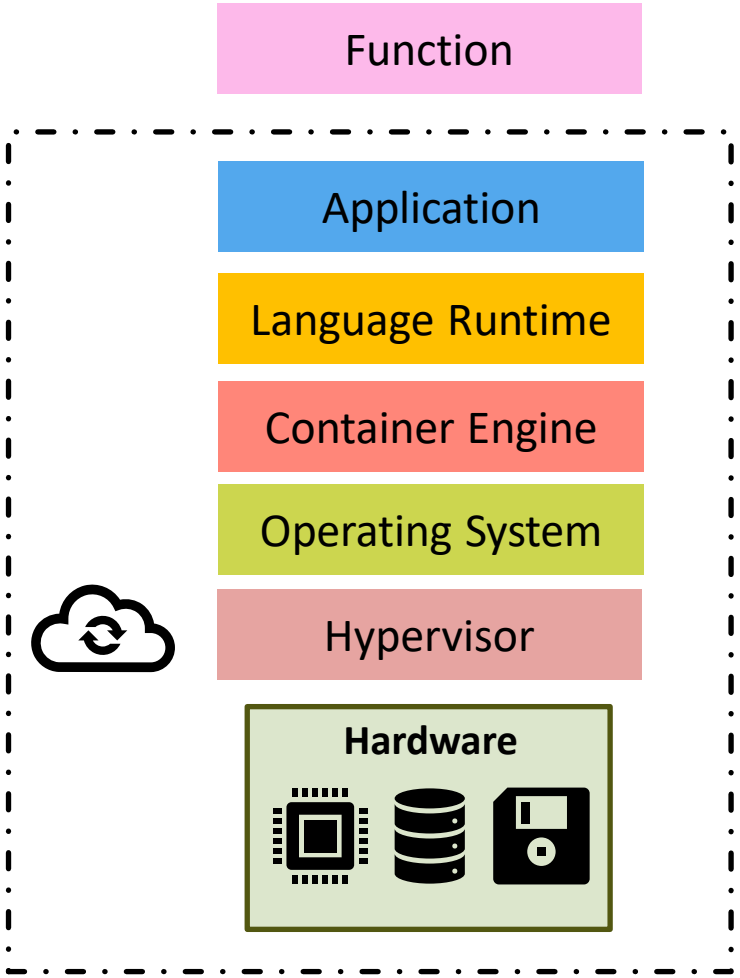
Cloud and Serverless

Cloud and Serverless



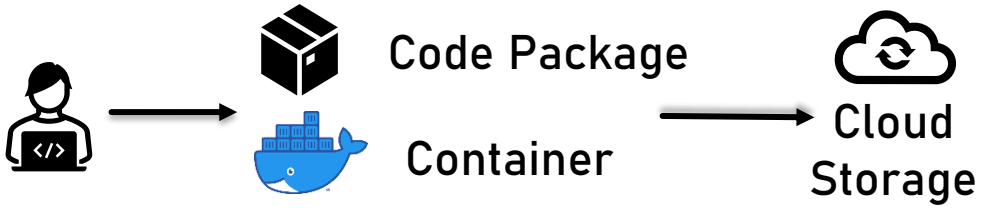
Functions

Cloud and Serverless

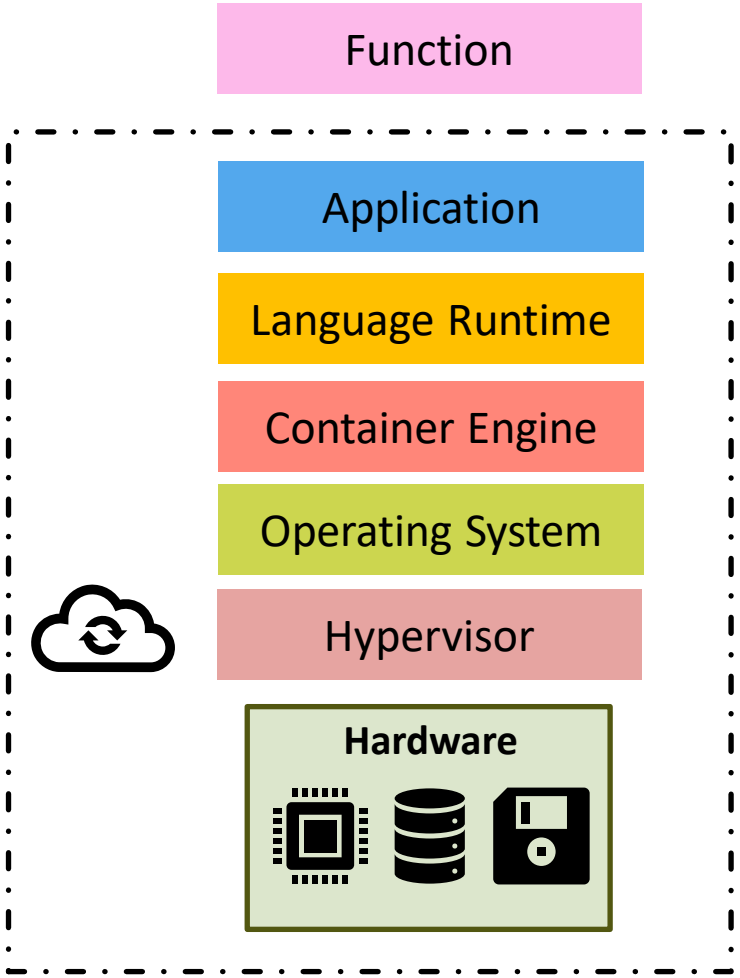


Functions

Compilation: deploying functions to the cloud.

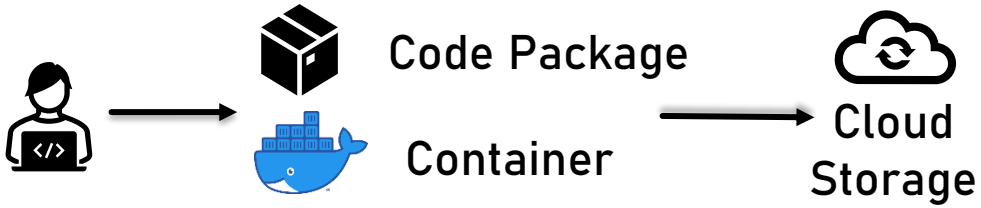


Cloud and Serverless

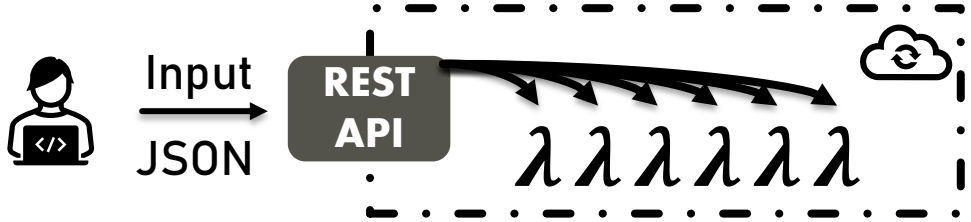


Functions

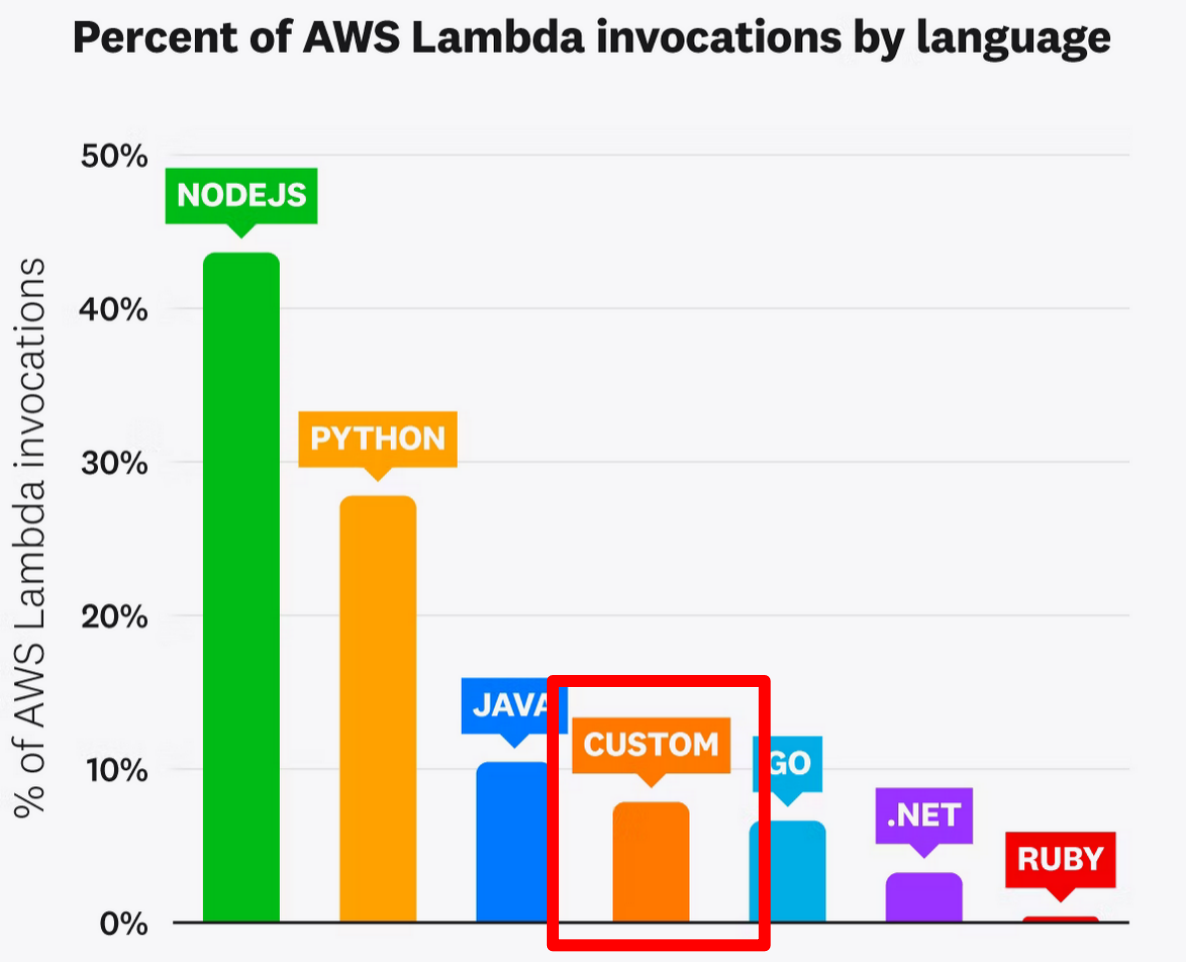
Compilation: deploying functions to the cloud.



Runtime: invoking existing functions.



Cloud and Serverless



Source: DataDog, “State of Serverless 2023”, https://www.datadoghq.com/state-of-serverless/

AWS Lambda with C++ - Function

```
#include <aws/lambda-runtime/runtime.h>
#include <aws/core/utils/json/JsonSerializer.h>
#include <aws/core/utils/memory/stl/SimpleStringStream.h>

using namespace aws::lambda_runtime;

invocation_response my_handler(invocation_request const& request)
{

}
```

AWS Lambda with C++ - Function

```
#include <aws/lambda-runtime/runtime.h>
#include <aws/core/utils/json/JsonSerializer.h>
#include <aws/core/utils/memory/stl/SimpleStringStream.h>

using namespace aws::lambda_runtime;

invocation_response my_handler(invocation_request const& request)
{
  using namespace Aws::Utils::Json;
  JsonValue json(request.payload);
  if (!json.WasParseSuccessful()) {
    return invocation_response::failure(
      "Failed to parse input JSON", "InvalidJSON"
    );
  }





}
```

AWS Lambda with C++ - Function

```
#include <aws/lambda-runtime/runtime.h>
#include <aws/core/utils/json/JsonSerializer.h>
#include <aws/core/utils/memory/stl/SimpleStringStream.h>

using namespace aws::lambda_runtime;

invocation_response my_handler(invocation_request const& request)
{
  using namespace Aws::Utils::Json;
  JsonValue json(request.payload);
  if (!json.WasParseSuccessful()) {
    return invocation_response::failure(
      "Failed to parse input JSON", "InvalidJSON"
    );
  }
  auto iterations = json.GetInt64("iterations");
  auto result = pi_estimation(iterations);
  auto response = std::::to_string(result);



}
```

AWS Lambda with C++ - Function

```
#include <aws/lambda-runtime/runtime.h>
#include <aws/core/utils/json/JsonSerializer.h>
#include <aws/core/utils/memory/stl/SimpleStringStream.h>

using namespace aws::lambda_runtime;

invocation_response my_handler(invocation_request const& request)
{
  using namespace Aws::Utils::Json;
  JsonValue json(request.payload);
  if (!json.WasParseSuccessful()) {
    return invocation_response::failure(
      "Failed to parse input JSON", "InvalidJSON"
    );
  }
  auto iterations = json.GetInt64("iterations");
  auto result = pi_estimation(iterations);
  auto response = std:::to_string(result);
  return invocation_response:::success(
    response, "application/json"
  );
}
```

AWS Lambda with C++ - Function

```
#include <aws/lambda-runtime/runtime.h>
#include <aws/core/utils/json/JsonSerializer.h>
#include <aws/core/utils/memory/stl/SimpleStringStream.h>

using namespace aws::lambda_runtime;

invocation_response my_handler(invocation_request const& request)
{
  using namespace Aws::Utils::Json;
  JsonValue json(request.payload);
  if (!json.WasParseSuccessful()) {
    return invocation_response::failure(
      "Failed to parse input JSON", "InvalidJSON"
    );
  }
  auto iterations = json.GetInt64("iterations");
  auto result = pi_estimation(iterations);
  auto response = std:::to_string(result);
  return invocation_response:::success(
    response, "application/json"
  );
}
```

(Cross) Compile to shared library.

AWS Lambda with C++ - Function

```
#include <aws/lambda-runtime/runtime.h>
#include <aws/core/utils/json/JsonSerializer.h>
#include <aws/core/utils/memory/stl/SimpleStringStream.h>

using namespace aws::lambda_runtime;

invocation_response my_handler(invocation_request const& request)
{
  using namespace Aws::Utils::Json;
  JsonValue json(request.payload);
  if (!json.WasParseSuccessful()) {
    return invocation_response::failure(
      "Failed to parse input JSON", "InvalidJSON"
    );
  }
  auto iterations = json.GetInt64("iterations");
  auto result = pi_estimation(iterations);
  auto response = std:::to_string(result);
  return invocation_response:::success(
    response, "application/json"
  );
}
```

(Cross) Compile to shared library.

↓

Link with custom runtime.

↓

AWS Lambda with C++ - Function

```
#include <aws/lambda-runtime/runtime.h>
#include <aws/core/utils/json/JsonSerializer.h>
#include <aws/core/utils/memory/stl/SimpleStringStream.h>

using namespace aws::lambda_runtime;

invocation_response my_handler(invocation_request const& request)
{
  using namespace Aws::Utils::Json;
  JsonValue json(request.payload);
  if (!json.WasParseSuccessful()) {
    return invocation_response::failure(
      "Failed to parse input JSON", "InvalidJSON"
    );
  }
  auto iterations = json.GetInt64("iterations");
  auto result = pi_estimation(iterations);
  auto response = std:::to_string(result);
  return invocation_response:::success(
    response, "application/json"
  );
}
```

(Cross) Compile to shared library.

Link with custom runtime.

Upload to cloud with all dependencies.

AWS Lambda with C++ - Function

```
bool InvokeFunction(
  const Aws::String& functionName,
  std::shared_ptr<Aws::Lambda::LambdaClient> client,
  int invocations, int &result
)
{
  Aws::Lambda::Model::InvokeRequest invokeRequest;
  invokeRequest.SetFunctionName(functionName);
  invokeRequest.SetInvocationType(
  Aws:::Lambda::Model::InvocationType::RequestResponse);

  std::shared_ptr<Aws::IOStream> payload
        = Aws::MakeShared<Aws::StringStream>();
  Aws::Utils::Json::JsonValue jsonPayload;
  jsonPayload.WithInt64("iterations", iterations);
  *payload <<< jsonPayload.View().WriteReadable();
  invokeRequest.SetBody(payload);
  invokeRequest.SetContentType("application/json");

  …

}
```

(Cross) Compile to shared library.

Link with custom runtime.

Upload to cloud with all dependencies.

AWS Lambda with C++ - Function

```
bool InvokeFunction(
)
{
  {
    …

    auto outcome = client->-Invoke(invokeRequest);

    if (outcome.IsSuccess()) {
      auto &result = outcome.GetResult();
      Aws::IOStream &payload = result.GetPayload();
      Aws::String functionResult;
      std::getline(payload, functionResult);
      result = std::stoi(functionResult);
      return true;
    } else {
      return false;
    }

  }
  …
}
```

(Cross) Compile to shared library.

Link with custom runtime.

Upload to cloud with all dependencies.

AWS Lambda with C++ - Function

```
Aws::SDKOptions options;
Aws::InitAPI(options);
Aws::Client::ClientConfiguration clientConfig;
auto m_client = Aws::MakeShared<Aws::Lambda::LambdaClient>(
  ALLOCATION_TAG,
  clientConfig
);

int n = 10000;
int np = 10;
std::vector<int> results(np);
std::vector<std::thread> threads;
for (int i = 0; i < np; i++) {
  threads.emplace_back([&, i]() {
    InvokeFunction("pi-mc-worker", n / np, results[i]);
  });
}

for (auto &thread : threads) {
  thread.join();
}
auto pi = std:::reduce(results.begin(), results.end()) / np;
```

(Cross) Compile to shared library.

Link with custom runtime.

Upload to cloud with all dependencies.

Challenges

Challenges

Complex, multi-source project setup

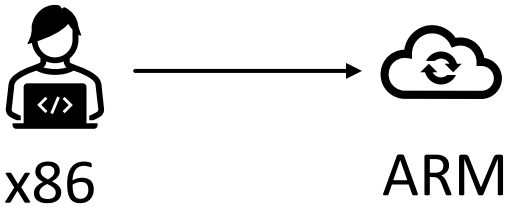
```
#pragma omp but in serverless
for(int i = 0; i < n; ++i)
  pi_mc(i);
```

Challenges

Complex, multi-source project setup

```
#pragma omp but in serverless
for(int i = 0; i < n; ++i)
  pi_mc(i);
```

Cross-compiled environments

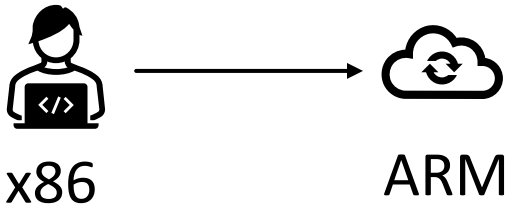


Challenges

Complex, multi-source project setup

```
#pragma omp but in serverless
for(int i = 0; i < n; ++i)
  pi_mc(i);
```

Cross-compiled environments



Lack of static typing

Manual verification?
JSON Schema?

Cppless: single-source C++ compiler for serverless

```
double pi_mc(int n);

double pi_estimate()
{
  const int n = 100000000;
  const int np = 128;

  cppless::aws_dispatcher dispatcher;
  auto aws = dispatcher.create_instance();

  std:::vector<double> results(np);
  auto fn = [=] { return pi_mc(n / np); };

  for (auto& result : results)
        cppless:::dispatch(aws, fn, result);
  cppless:::wait(aws, np);

  auto pi = std:::reduce(
    results.begin(), results.end()
  ) / np;

  return pi;
}
```

Cppless: single-source C++ compiler for serverless

```
double pi_mc(int n);

double pi_estimate()
{
  const int n = 100000000;
  const int np = 128;

  cppless::aws_dispatcher dispatcher;
  auto aws = dispatcher.create_instance();

  std:::vector<double> results(np);
  auto fn = [=] { return pi_mc(n / np); };

  for (auto& result : results)
        cppless:::dispatch(aws, fn, result);
  cppless:::wait(aws, np);

  auto pi = std:::reduce(
    results.begin(), results.end()
  ) / np;

  return pi;
}
```

C++ abstraction for cloud provider APIs.

Avoids the vendor lock-in and simplifies invocations.

Cppless: single-source C++ compiler for serverless

```
double pi_mc(int n);

double pi_estimate()
{
  const int n = 100000000;
  const int np = 128;

  cppless::aws_dispatcher dispatcher;
  auto aws = dispatcher.create_instance();

  std:::vector<double> results(np);
  auto fn = [=] { return pi_mc(n / np); };

  for (auto& result : results)
        cppless:::dispatch(aws, fn, result);
  cppless:::wait(aws, np);

  auto pi = std:::reduce(
    results.begin(), results.end()
  ) / np;

  return pi;
}
```

- **C++ abstraction for cloud provider APIs.**
  Avoids the vendor lock-in and simplifies invocations.
- **Serverless function as C++ lambda expression.**
  Automatically compiled to a cloud function.

Cppless: single-source C++ compiler for serverless

```
double pi_mc(int n);

double pi_estimate()
{
  const int n = 100000000;
  const int np = 128;

  cppless::aws_dispatcher dispatcher;
  auto aws = dispatcher.create_instance();

  std:::vector<double> results(np);
  auto fn = [=] { return pi_mc(n / np); };

  for (auto& result : results)
        cppless:::dispatch(aws, fn, result);
  cppless:::wait(aws, np);

  auto pi = std:::reduce(
    results.begin(), results.end()
  ) / np;

  return pi;
}
```

- **C++ abstraction for cloud provider APIs.**
  Avoids the vendor lock-in and simplifies invocations.
- **Serverless function as C++ lambda expression.**
  Automatically compiled to a cloud function.
- **Integrated invocation of the function.**
  Automatic serialization and type checking.

How it works?

Compile Time

Runtime

How it works?

Compile Time

Source code

int foo(int x)

int bar(int x)

Runtime

How it works?

Compile Time

Source code

int foo(int x)

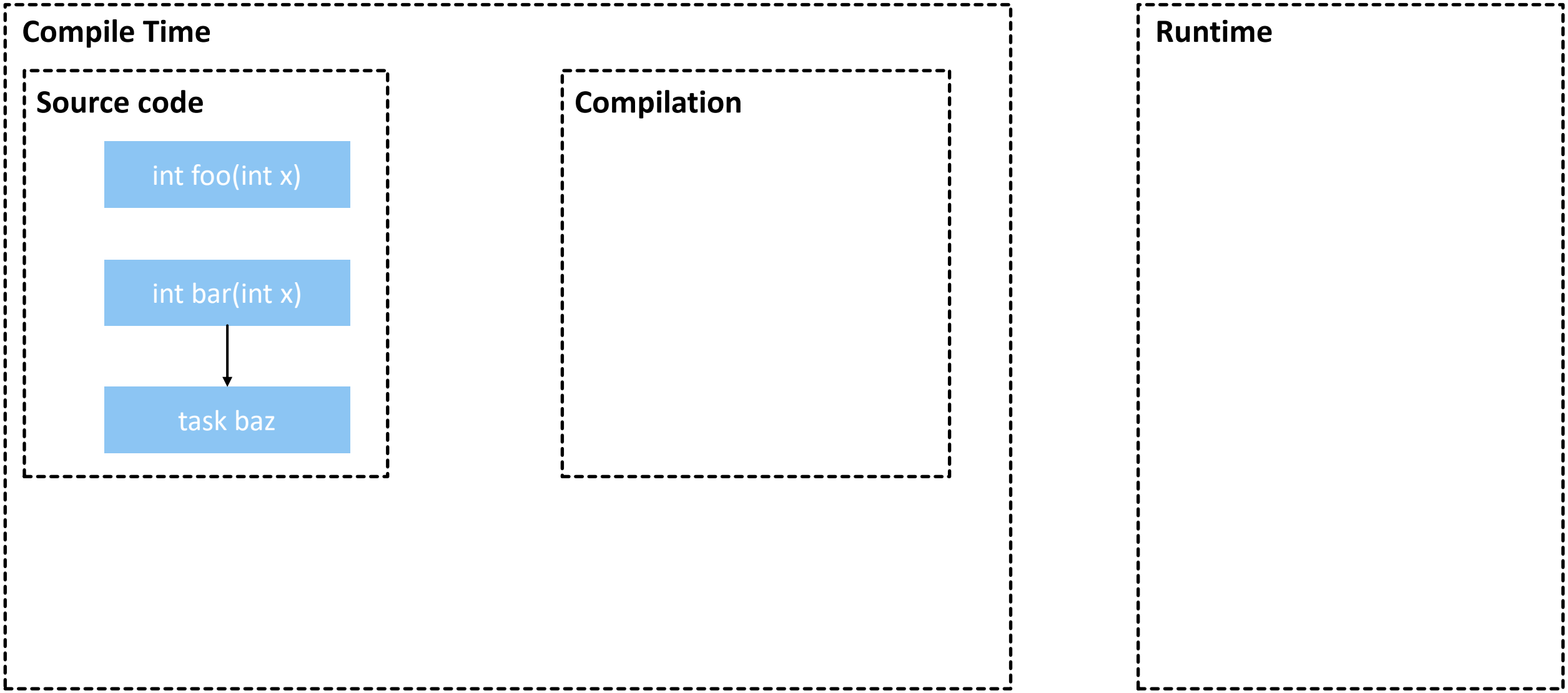
int bar(int x)

↓

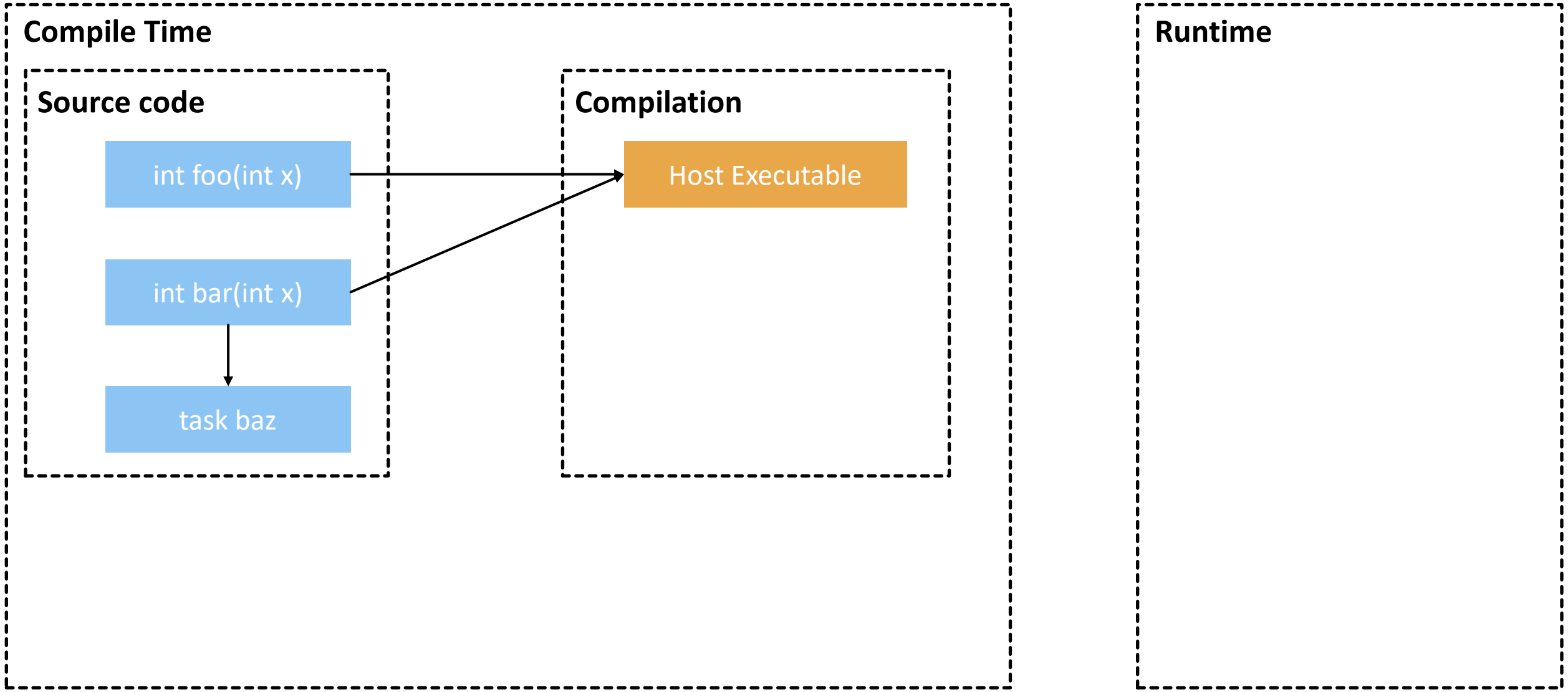
task baz

Runtime

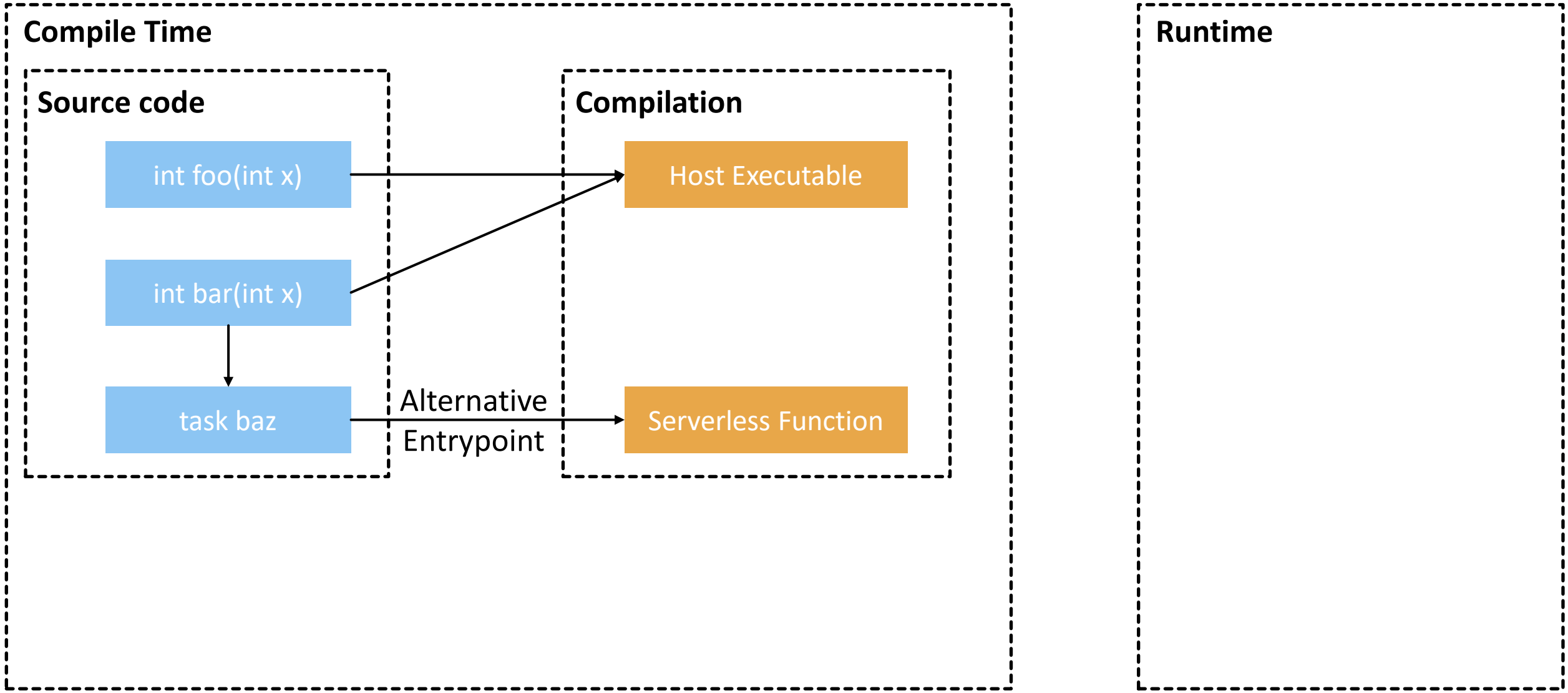
How it works?


Compile Time
Source code
int foo(int x)
int bar(int x)
task baz
Compilation
Runtime

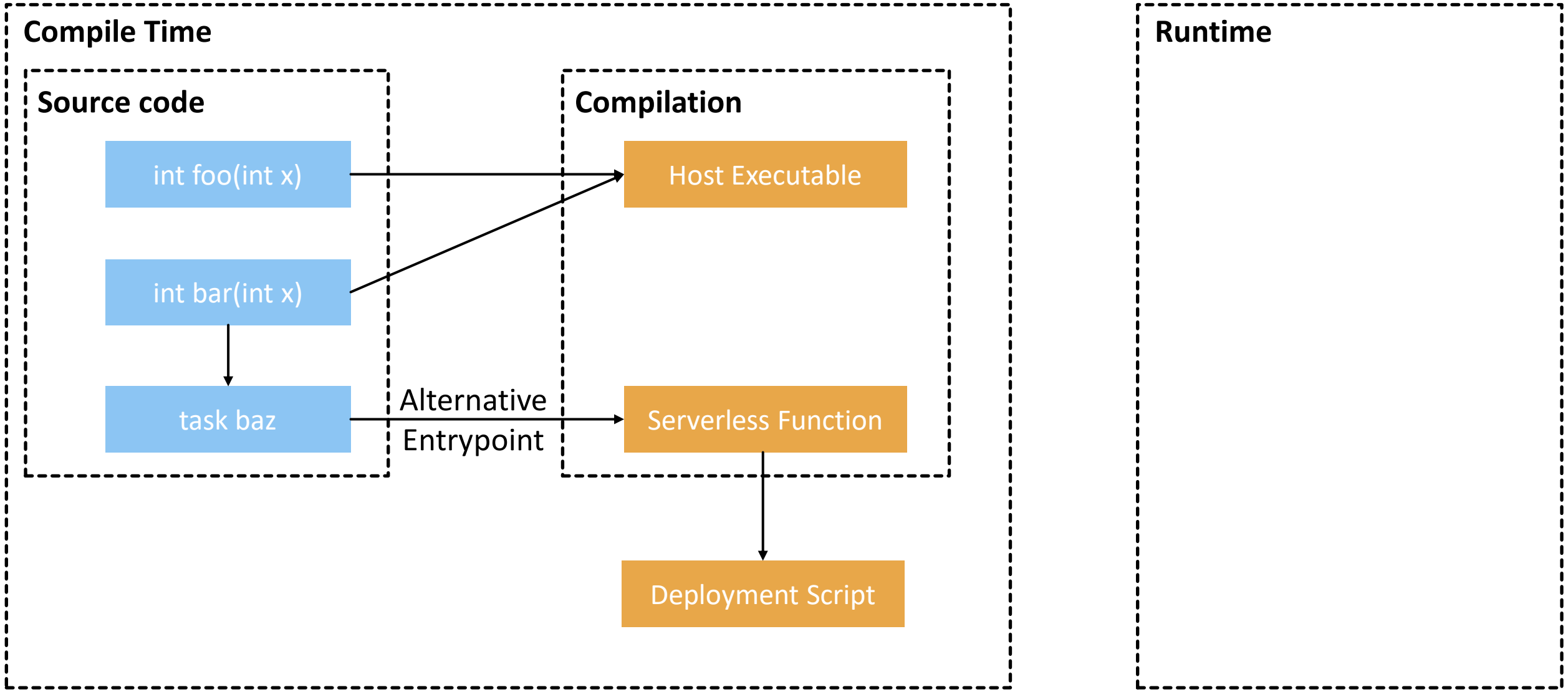
How it works?


Compile Time
Source code
int foo(int x)
int bar(int x)
task baz
Compilation
Host Executable
Runtime

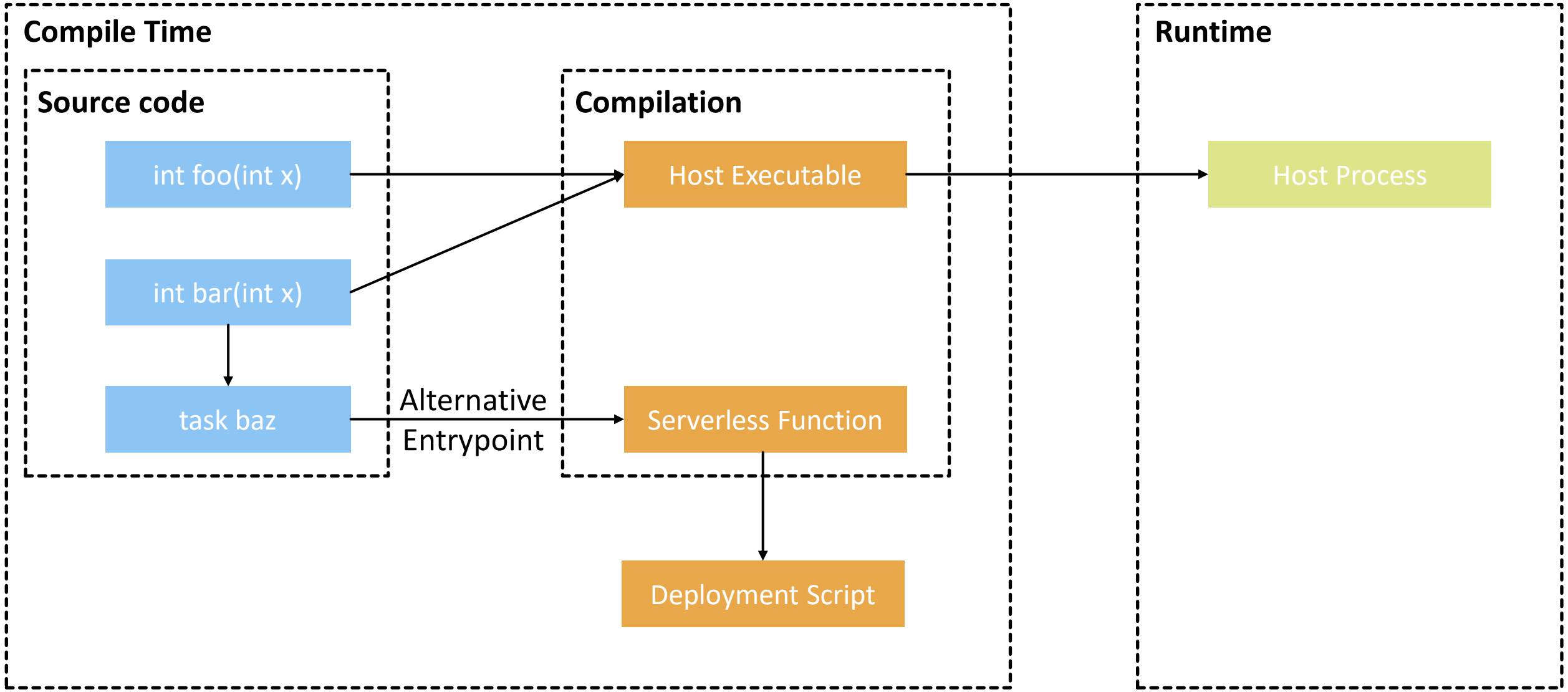
How it works?


Compile Time
Source code
int foo(int x)
int bar(int x)
task baz
Alternative
Entrypoint
Compilation
Host Executable
Serverless Function
Runtime

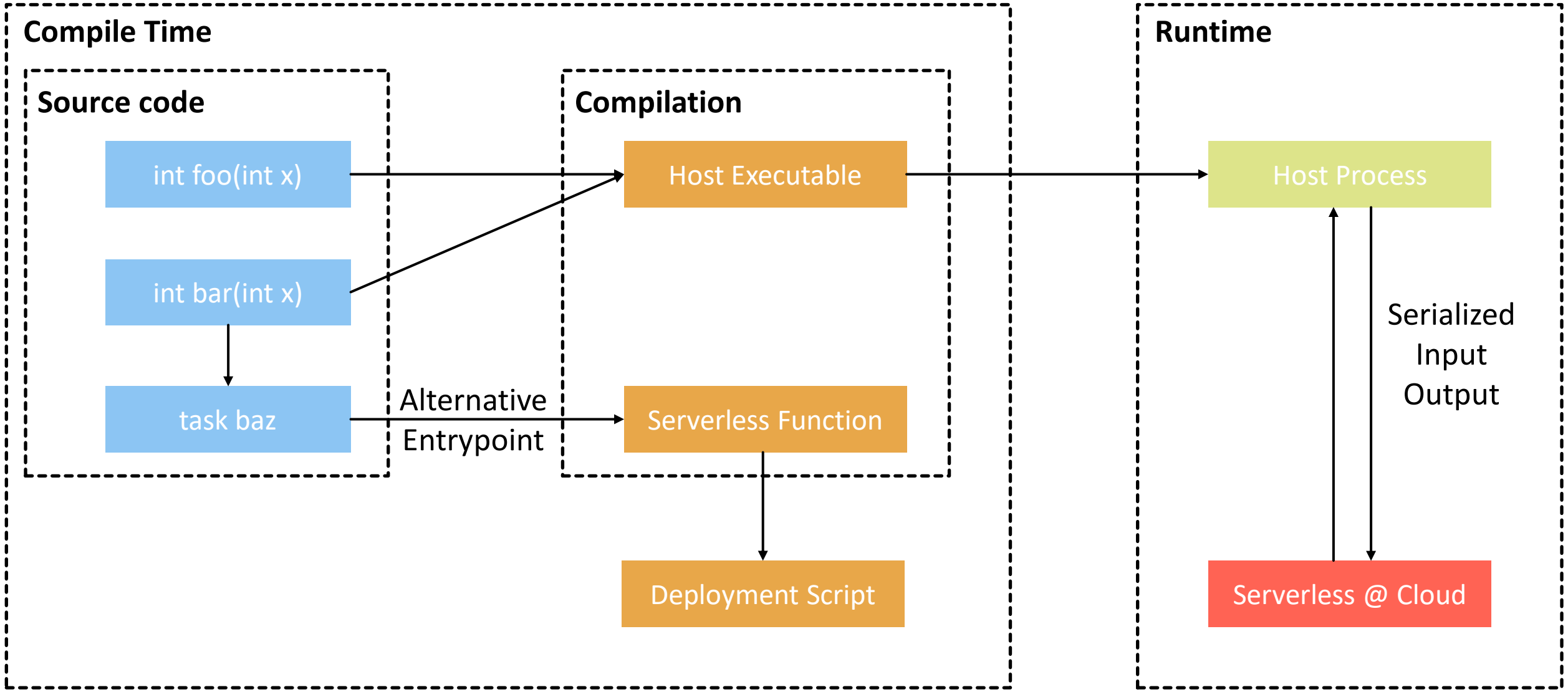
How it works?


Compile Time
Source code
int foo(int x)
int bar(int x)
task baz
Alternative
Entrypoint
Compilation
Host Executable
Serverless Function
Deployment Script
Runtime

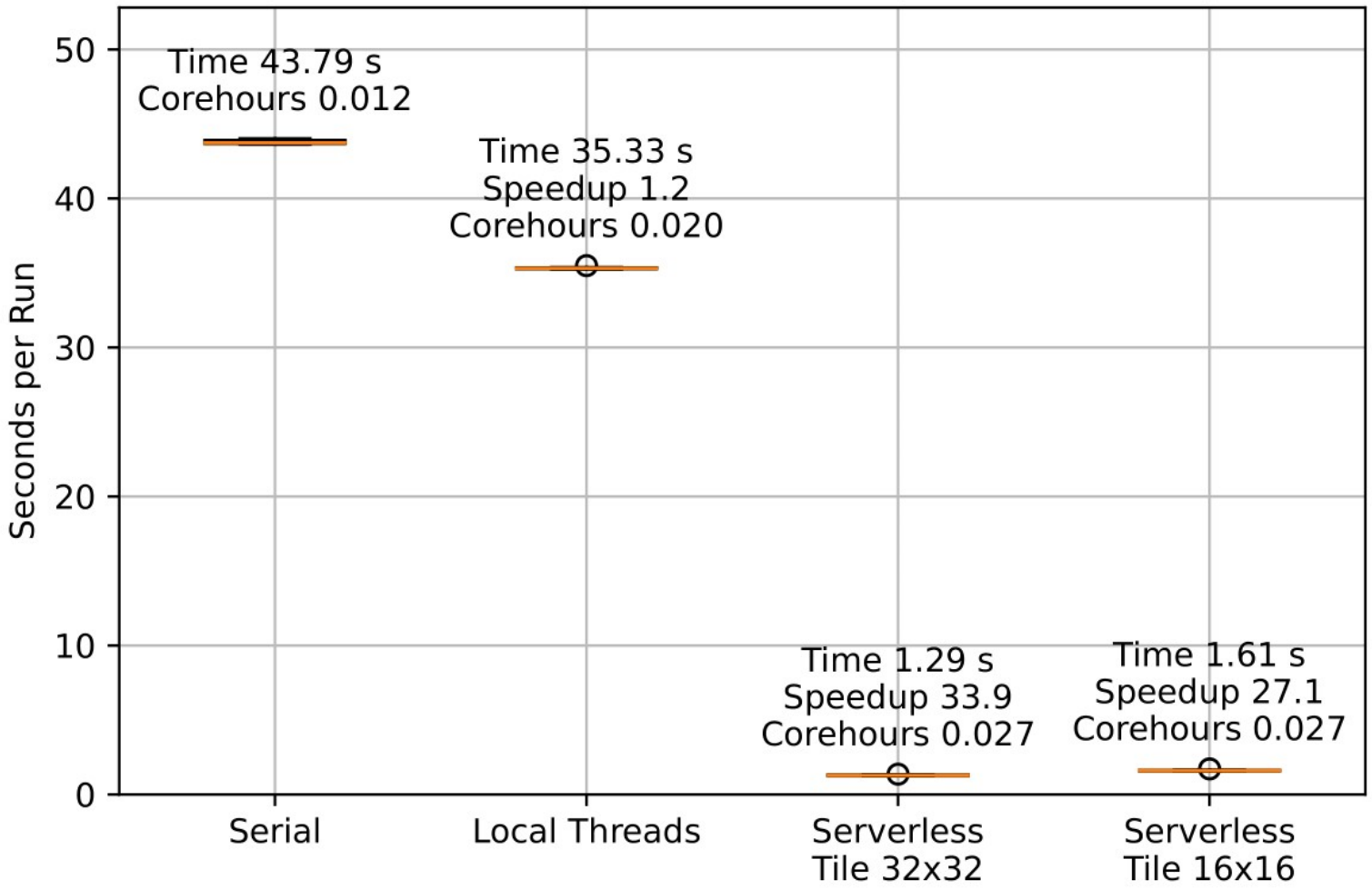
How it works?


Compile Time
Source code
int foo(int x)
int bar(int x)
task baz
Alternative
Entrypoint
Compilation
Host Executable
Serverless Function
Deployment Script
Runtime
Host Process

How it works?


Compile Time
Source code
int foo(int x)
int bar(int x)
task baz
Alternative
Entrypoint
Compilation
Host Executable
Serverless Function
Deployment Script
Runtime
Host Process
Serialized
Input
Output
Serverless @ Cloud

Serverless ray tracing: from small VM to many functions



Conclusions

More of SPCL's research:

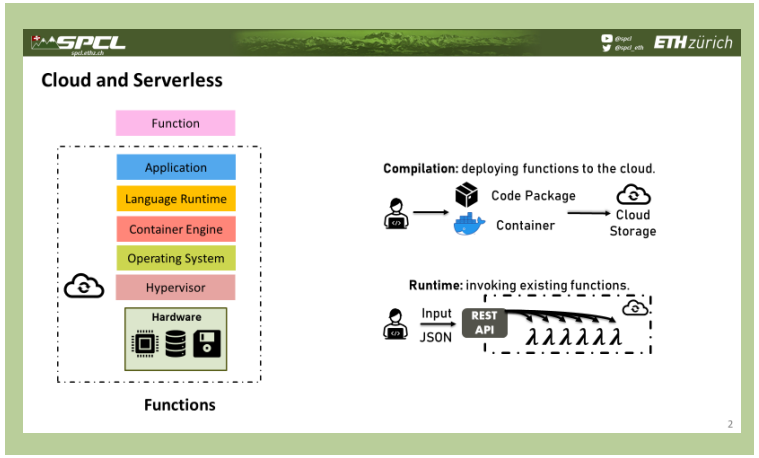
youtube.com/@spcl — 150+ Talks

twitter.com/spcl_eth — 1.2K+ Followers

github.com/spcl — 2K+ Stars

... or spcl.ethz.ch

Conclusions



More of SPCL's research:

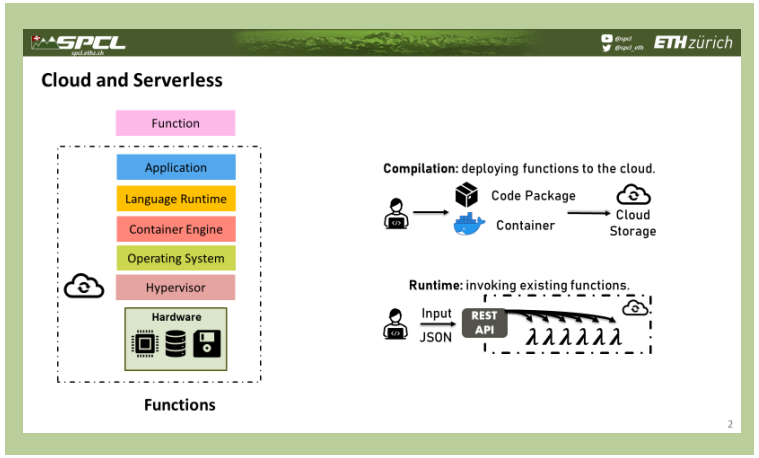
youtube.com/@spcl — 150+ Talks

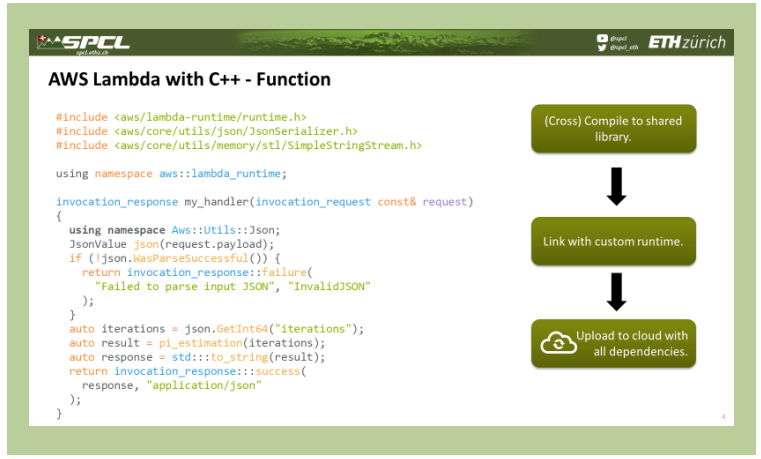
twitter.com/spcl_eth — 1.2K+ Followers

github.com/spcl — 2K+ Stars

… or spcl.ethz.ch

Conclusions





More of SPCL's research:

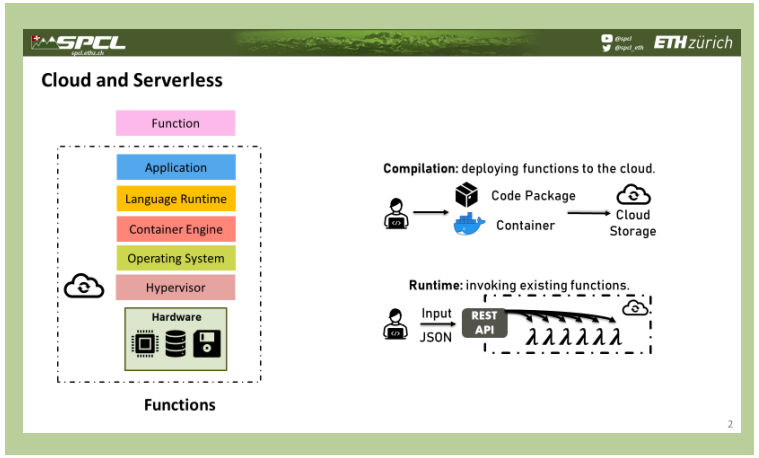
youtube.com/@spcl — 150+ Talks

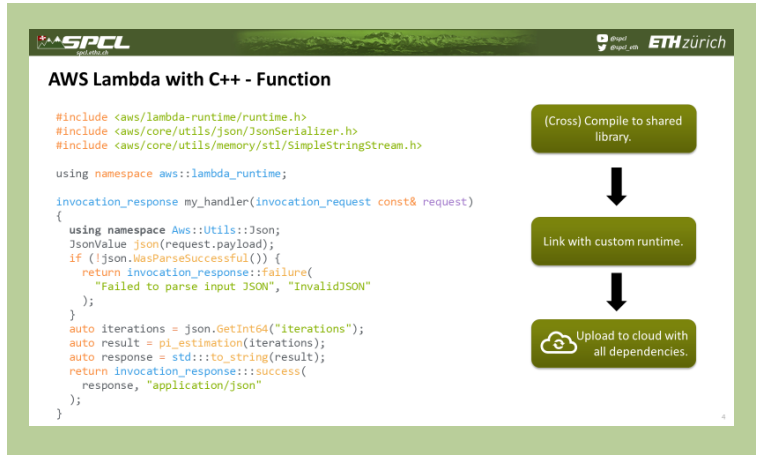
twitter.com/spcl_eth — 1.2K+ Followers

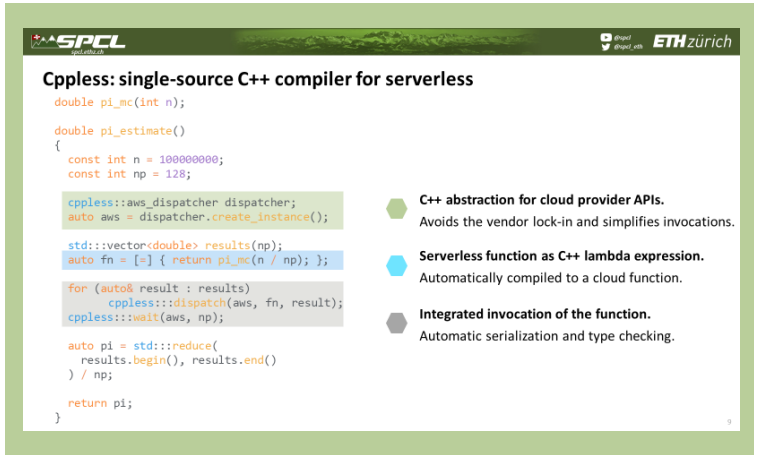
github.com/spcl — 2K+ Stars

… or spcl.ethz.ch

Conclusions







More of SPCL's research:

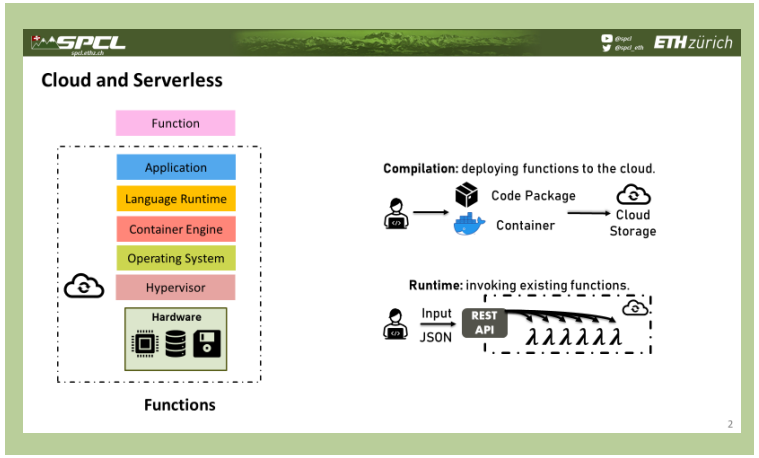
youtube.com/@spcl — 150+ Talks

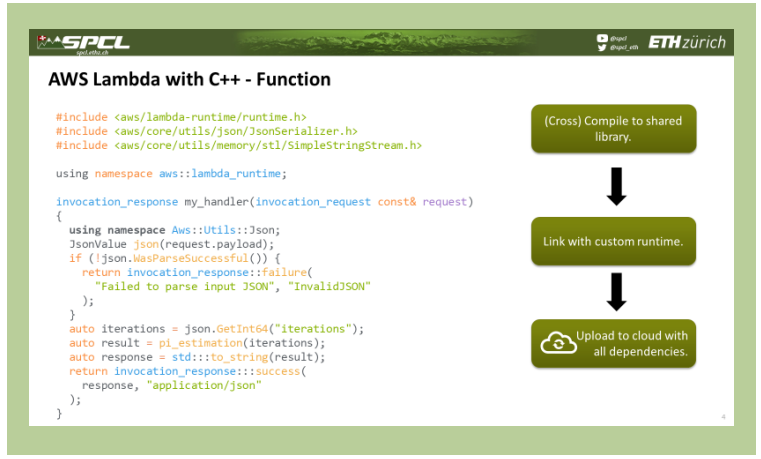
twitter.com/spcl_eth — 1.2K+ Followers

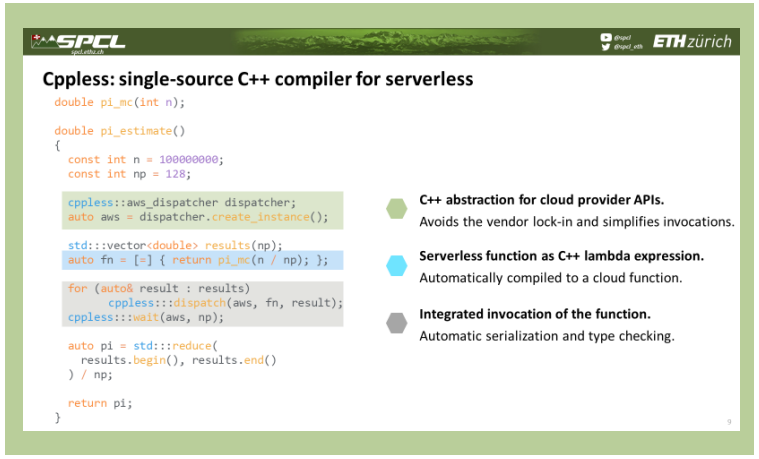
github.com/spcl — 2K+ Stars

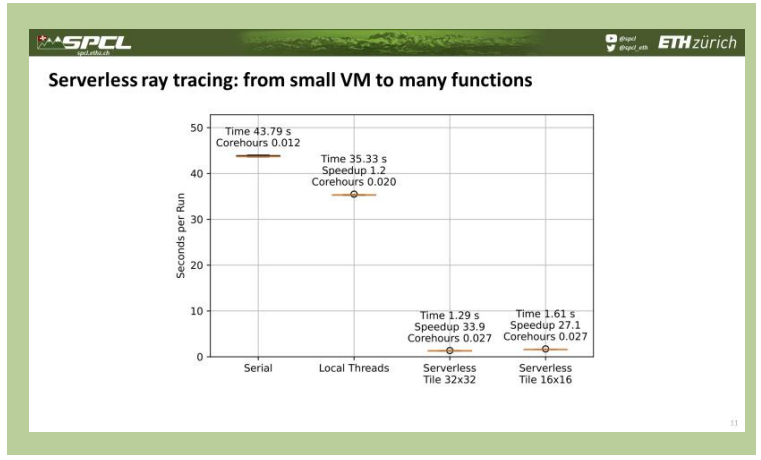
... or spcl.ethz.ch

Conclusions









More of SPCL's research:

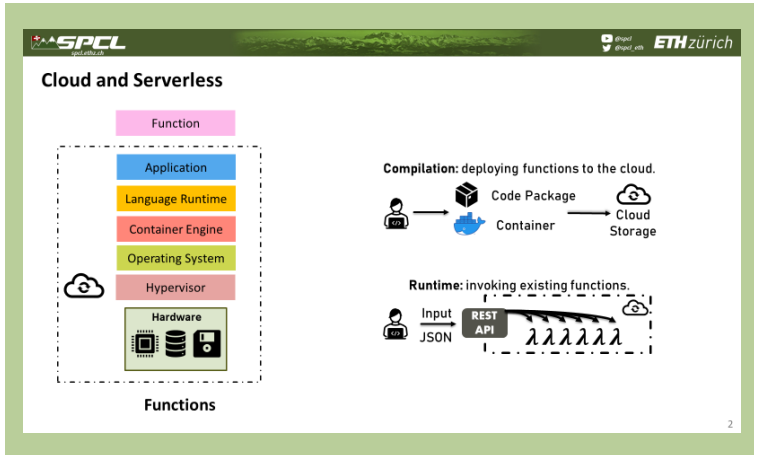
youtube.com/@spcl — 150+ Talks

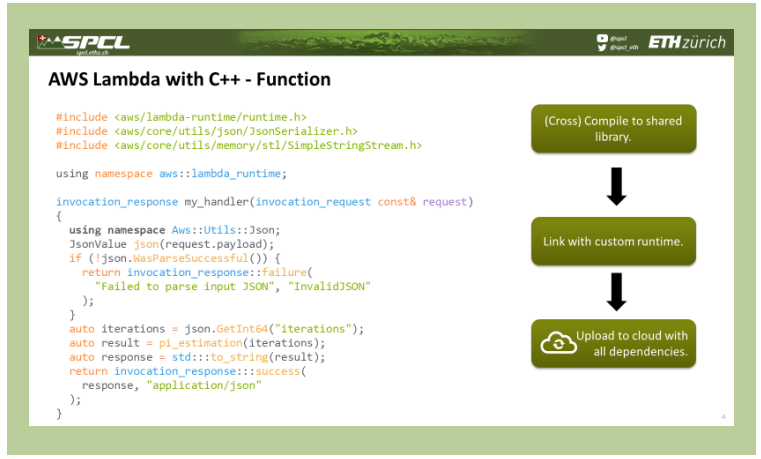
twitter.com/spcl_eth — 1.2K+ Followers

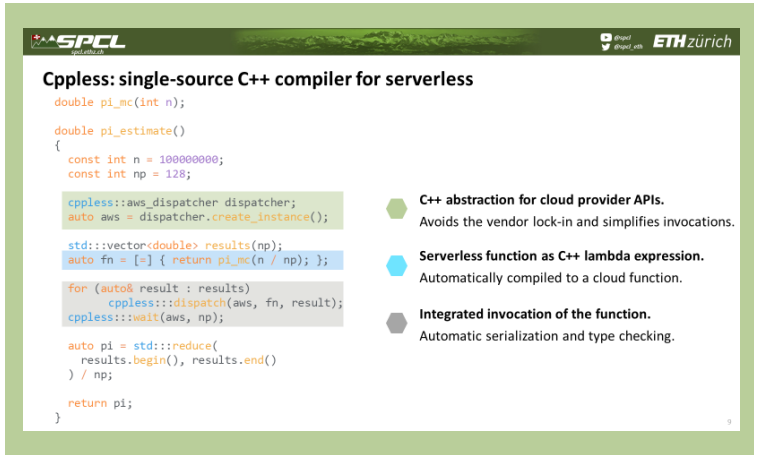
github.com/spcl — 2K+ Stars

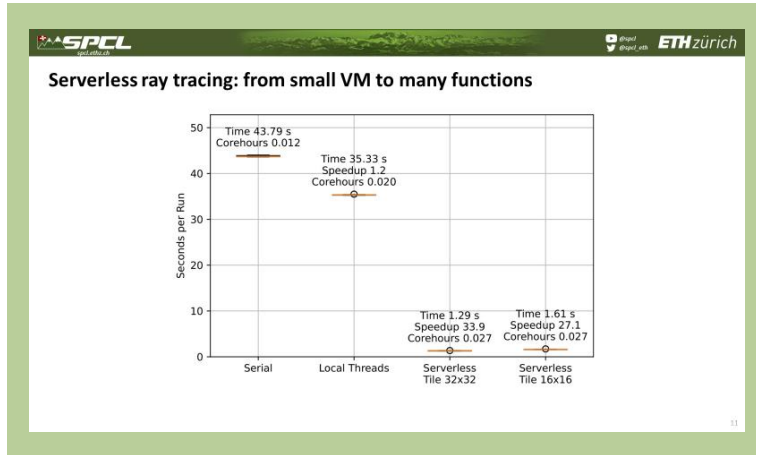
... or spcl.ethz.ch

Conclusions









More of SPCL's research:

youtube.com/@spcl — 150+ Talks

twitter.com/spcl_eth — 1.2K+ Followers

github.com/spcl — 2K+ Stars

... or spcl.ethz.ch

GitHub

spcl/cppless

Paper preprint

mcopik.github.io/projects/cppless