ROBIN MILNER

Computer Science Department, University of Edinburgh, Edinbur

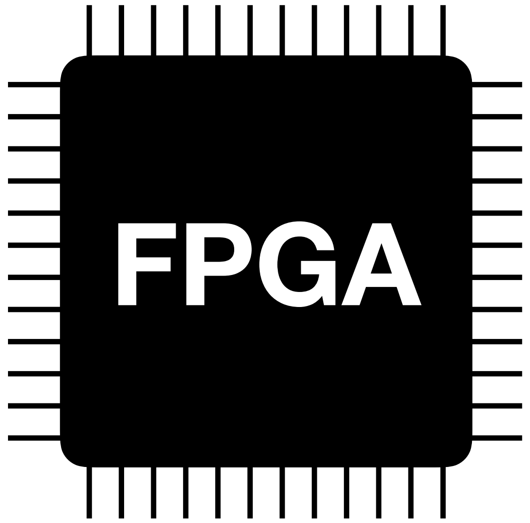
Received October 10, 1977; revised April 19, 1978

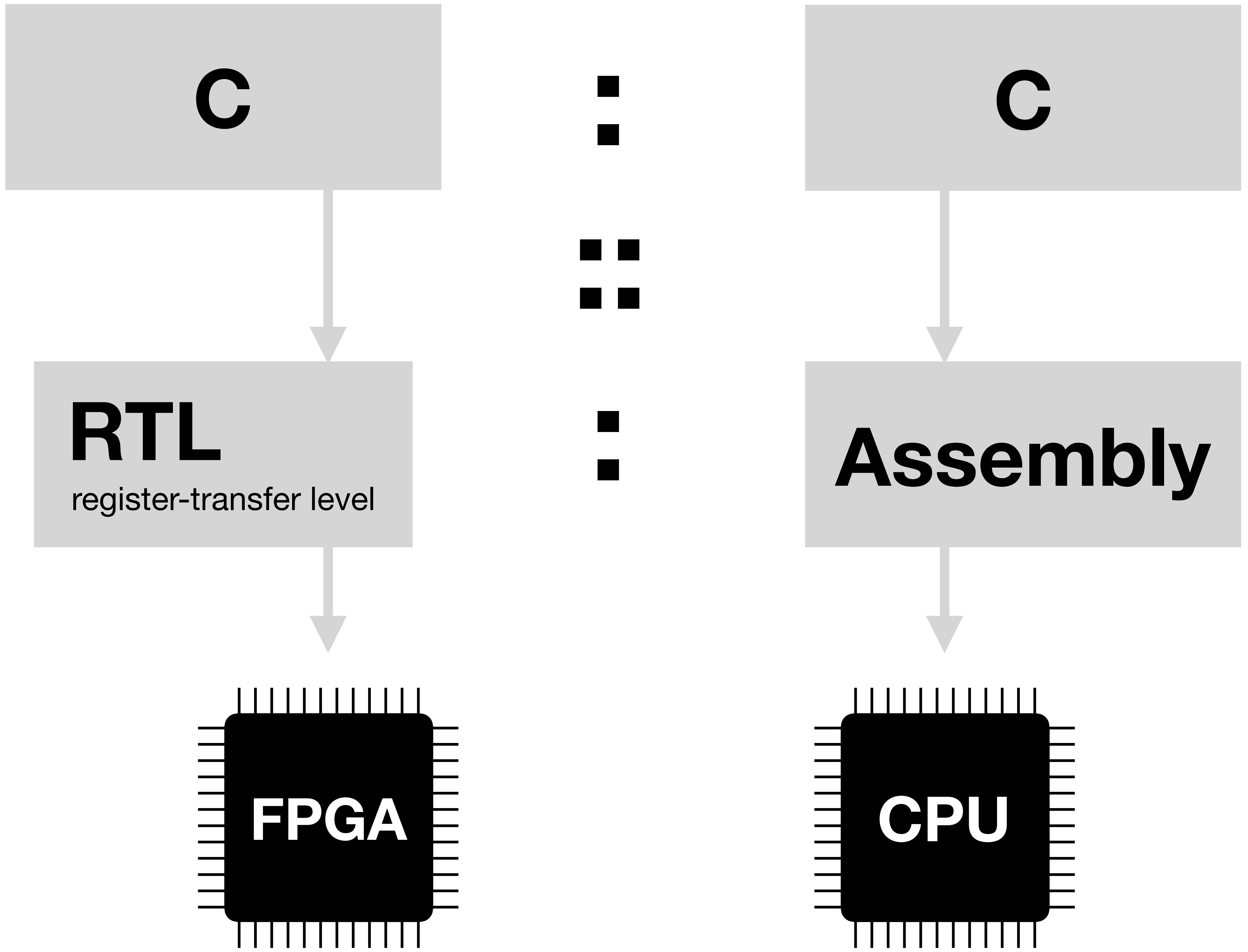
he aim of this work is largely a practical one. A widely employed sty
icularly in structure-processing languages which impose no d
ils defining procedures which work well on objects of a wide va
nal type discipline for such polymorphic procedures in the conte
nming language, and a compile time type-checking algorithm $\mathscr{W}$
ipline. A Semantic Soundness Theorem (based on a formal semant
s that well-type programs cannot "go wrong" and a Syntactic S
s that if $\mathscr{W}$ accepts a program then it is well typed. We also disc
lts to richer languages; a type-checking algorithm based on $\mathscr{W}$
lemented and working, for the metalanguage ML in the Edinbur

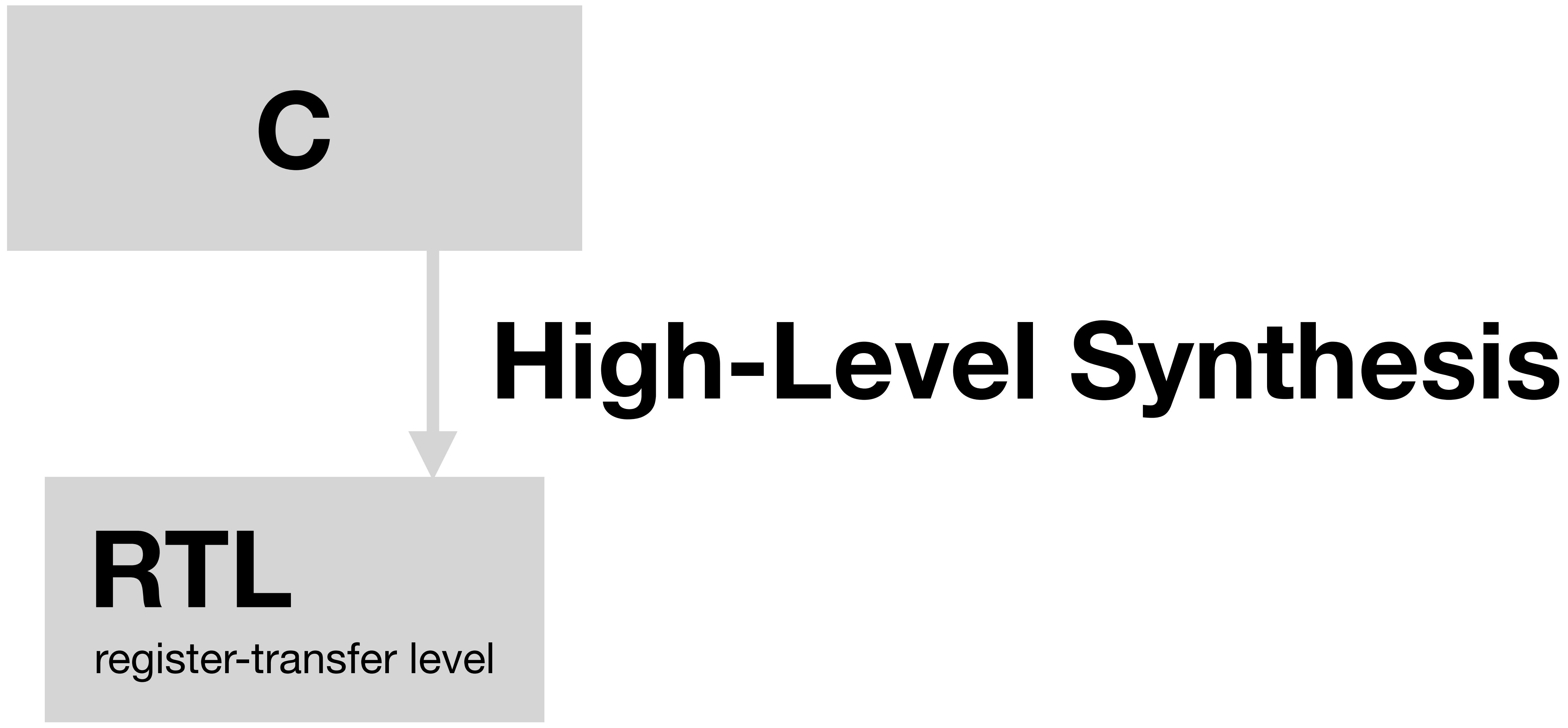
RTL

register-transfer level

Verilog VHDL Bluespec Chisel




C
:
C
RTL
register-transfer level
::
Assembly
:
FPGA
CPU


C
High-Level Synthesis
RTL
register-transfer level

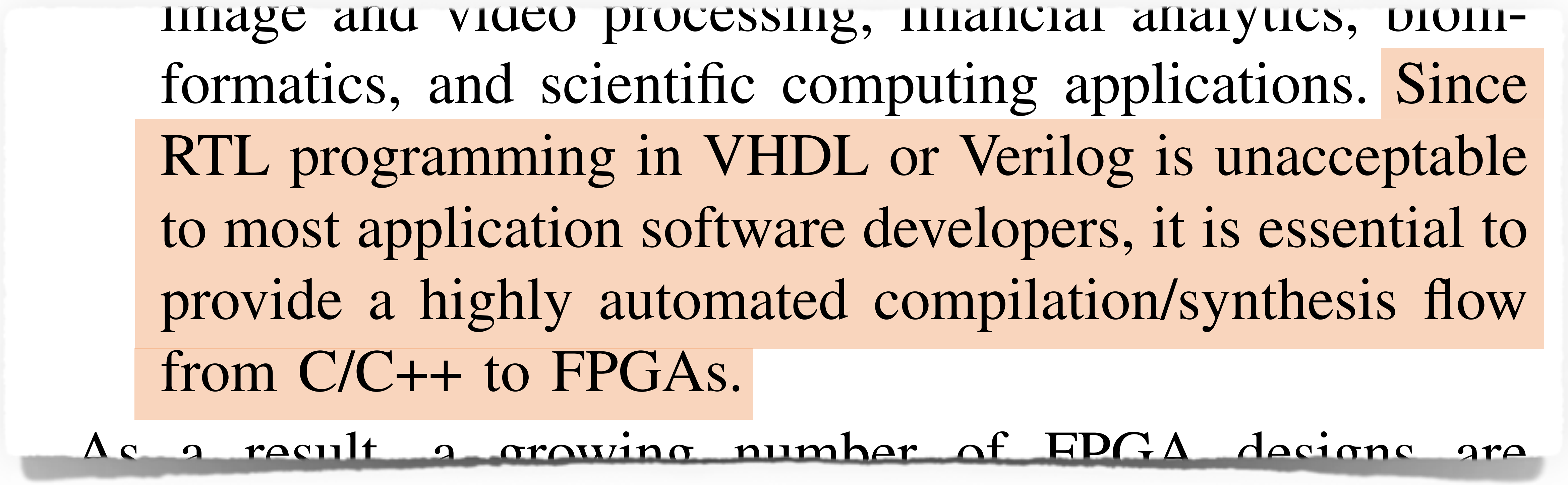
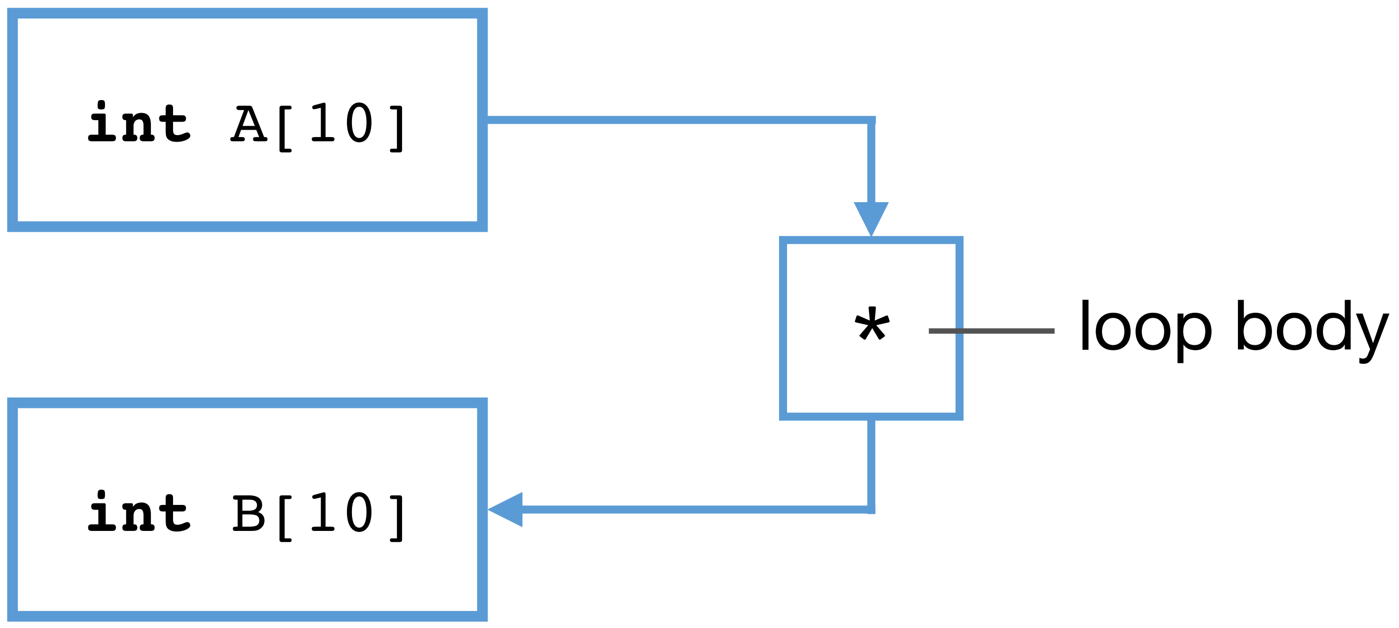


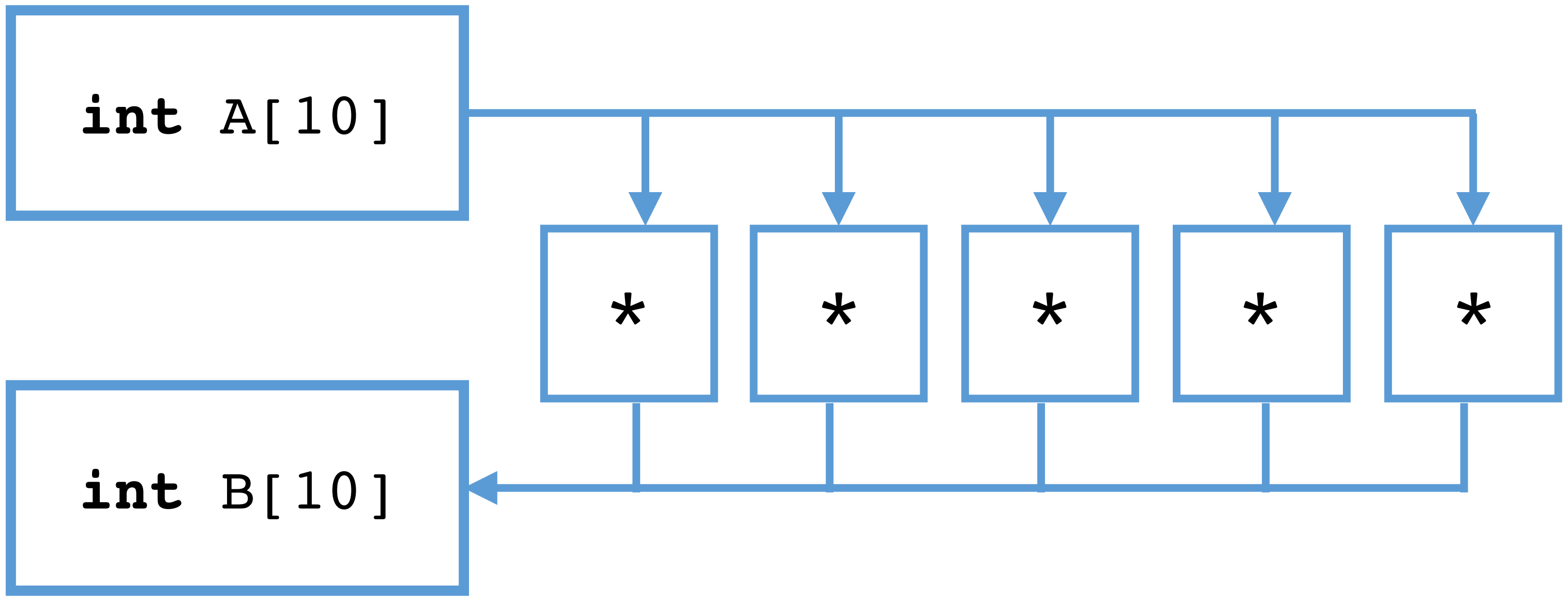
image and video processing, financial analytics, bioinformatics, and scientific computing applications. Since RTL programming in VHDL or Verilog is unacceptable to most application software developers, it is essential to provide a highly automated compilation/synthesis flow from C/C++ to FPGAs.

As a result, a growing number of FPGA designs are

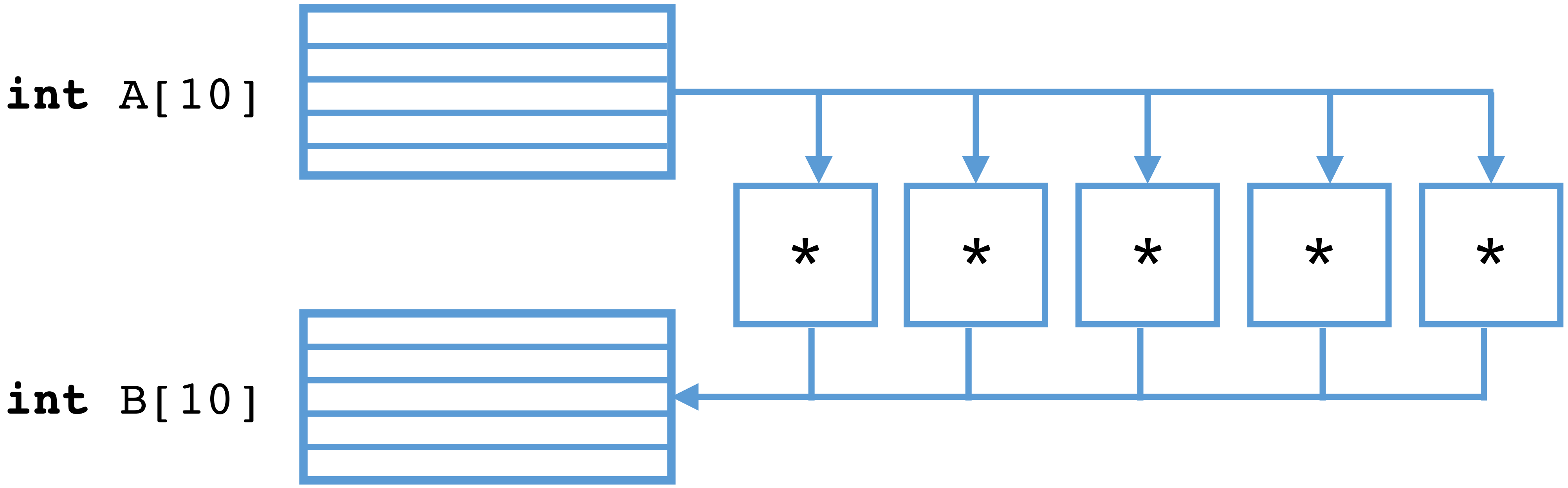
Verilog is unacceptable $\nRightarrow$ we must program FPGAs in C

```
int A[10];
int B[10];
for (int i = 0; i < 10; i++) {
    int x = A[i];
    int y = x * 5;
    B[i] = y;
}
```


int A[10]
*
loop body
int B[10]


int A[10]
*
*
*
*
*
int B[10]

```
#pragma HLS ARRAY_PARTITION variable=A factor=5
#pragma HLS ARRAY_PARTITION variable=B factor=5
int A[10];
int B[10];
for (int i = 0; i < 10; i++) {
    #pragma HLS UNROLL factor=5
    int x = A[i];
    int y = x * 5;
    B[i] = y;
}
```


int A[10]
int B[10]
*
*
*
*
*

```
#pragma HLS ARRAY_PARTITION variable=A factor=5
#pragma HLS ARRAY_PARTITION variable=B factor=5
int A[10];
int B[10];
for (int i = 0; i < 10; i++) {
    #pragma HLS UNROLL factor=5
    int x = A[i];
    int y = x * 5;
    B[i] = y;
}
```

Seashell:

A type system that guarantees that a high-level program has equivalent semantics in an FPGA implementation.

```
decl A: bit<32>[10];
for (...) {
  let x = A[i];
  A[i+1] = x + 1;
}
```

Memory declaration.
(Not just an array declaration.)

Banking is part of the memory’s type.

```
decl A: bit<32>[10 bank 5];
for (...) {
  let x = A[i];
  A[i+1] = x + 1;
}
```

```
decl A: bit<32>[10];
for (...) {
  let x = A[i];
  A[i+1] = x + 1;
}
```

Type error: `A` already used in this context

```
A ; B
```

Allow A and B to run in *parallel.*

```
let x = A[i];
A[i + i] = x + 1;
```

Conflict! Cannot read from and write to A simultaneously.

```
decl A: bit<32>[10 bank 5];
for (...) {
  let x = A[i]
  ---
  A[i+1] = x + 1;
}
```

A logical time step.

```
A --- B
```

Run A and then run B, sequentially.

```
let x = A[i];
---
A[i + 1] = x + 1;
```

No conflict. Statements are guaranteed to run in separate cycles.

```
{ A --- B };
{ C --- D };
     E
```

Specify complex parallel behavior using --- and ;.

```
decl A: bit<32>[10 bank 5];
decl B: bit<32>[10 bank 5];
for (...) unroll 5 {
  B[i] = A[i] + 1;
}
```

Unrolling affects the type of `i`
to indicate that it touches 5 locations.

```
decl A: bit<32>[10 bank 5];
let sum = 0;
for (...) unroll 5 {
  let x = A[i] + 1;
} combine {
  sum += x;
}
```

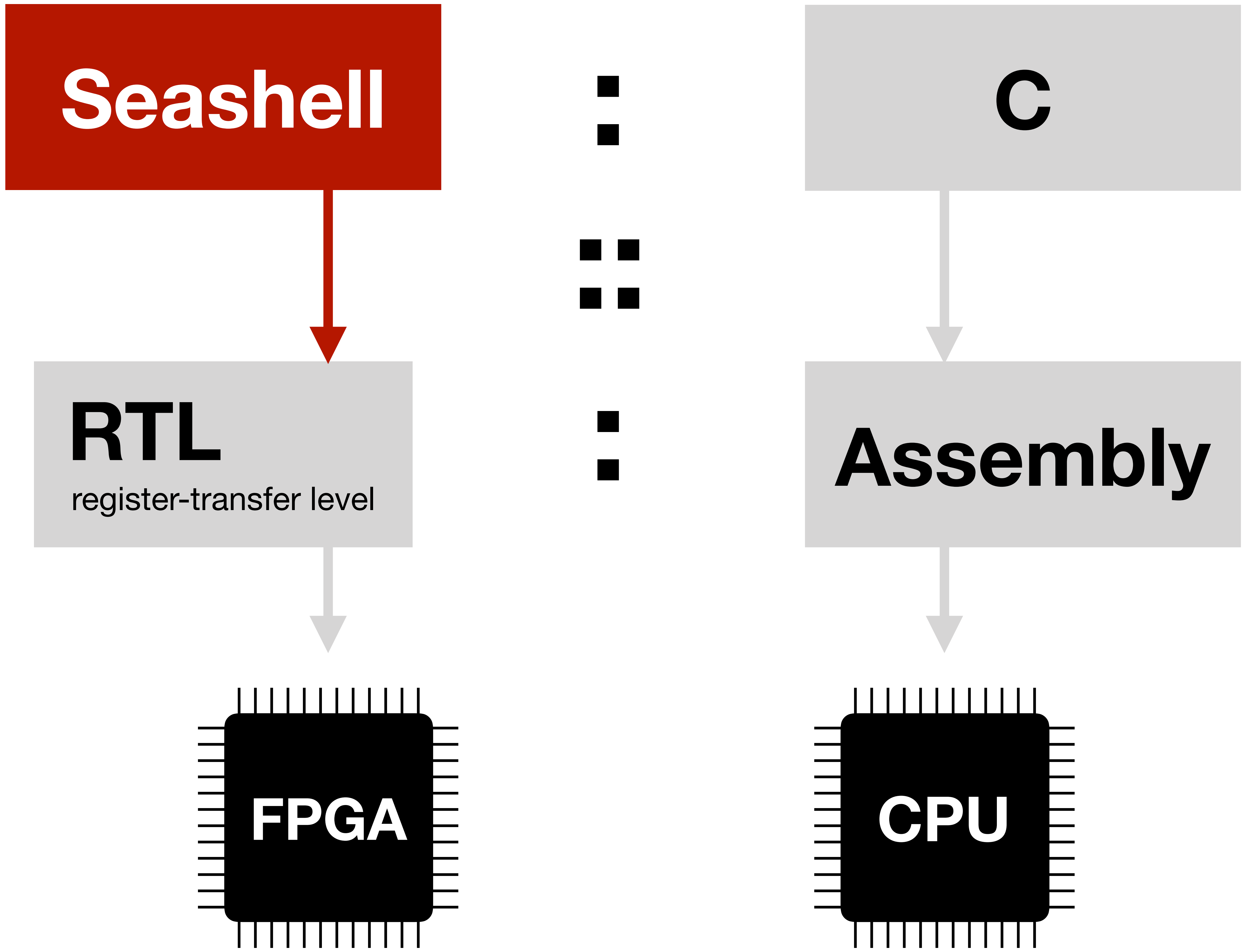
Explicitly delimit non-parallelizable computations such as reductions.

```
decl A: bit<32>[10 bank 5];
let sum = 0;
for (...) unroll 5 {
  let x = A[i] + 1;
} combine {
  sum += x;
}
```

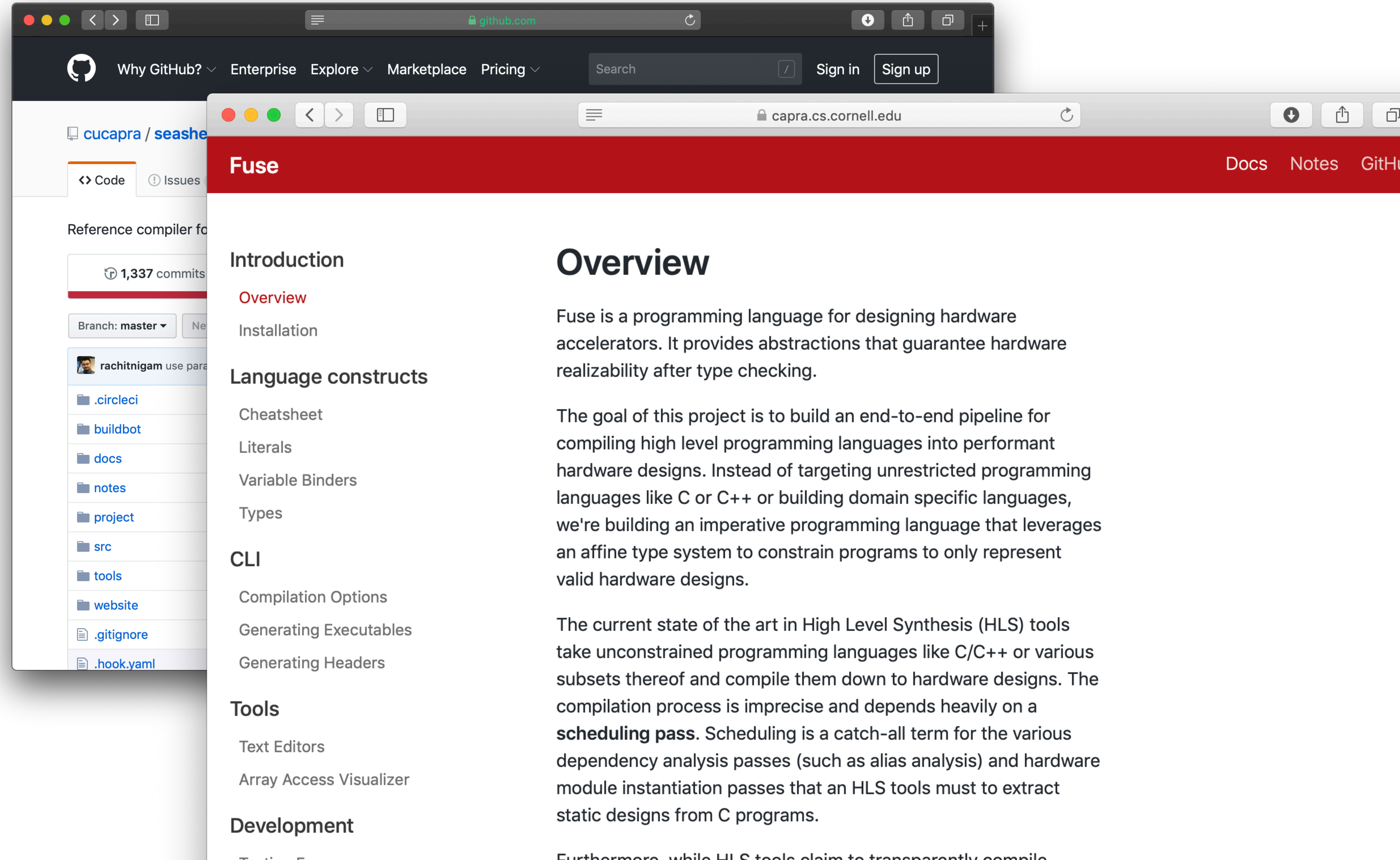
Well-typed programs preserve the semantics of the unannotated program.

```
decl A: bit<32>[10 bank 5];
let sum = 0;
for (...) unroll 5 {
  let x = A[i] + 1;
} combine {
  sum += x;
}
```

Well-typed programs preserve the semantics of the unannotated program.


Seashell
:
C
::
RTL
register-transfer level
:
Assembly
FPGA
CPU

capra.cs.cornell.edu/fuse



Introduction
- Overview
- Installation

Language constructs
- Cheatsheet
- Literals
- Variable Binders
- Types

CLI
- Compilation Options
- Generating Executables
- Generating Headers

Tools
- Text Editors
- Array Access Visualizer

Development

Overview

Fuse is a programming language for designing hardware accelerators. It provides abstractions that guarantee hardware realizability after type checking.

The goal of this project is to build an end-to-end pipeline for compiling high level programming languages into performant hardware designs. Instead of targeting unrestricted programming languages like C or C++ or building domain specific languages, we're building an imperative programming language that leverages an affine type system to constrain programs to only represent valid hardware designs.

The current state of the art in High Level Synthesis (HLS) tools take unconstrained programming languages like C/C++ or various subsets thereof and compile them down to hardware designs. The compilation process is imprecise and depends heavily on a **scheduling pass**. Scheduling is a catch-all term for the various dependency analysis passes (such as alias analysis) and hardware module instantiation passes that an HLS tools must to extract static designs from C programs.