P4 “suffers” from one significant limitation: the **voluminosity** of the code.
P4 “suffers” from one significant limitation: the *voluminosity* of the code.
P4’s voluminosity is due to two reasons: verbosity and limited parametrization
P4’s voluminosity is due to two reasons: **verbosity** and limited parametrization

// P4 (tofino)

RegisterAction (...)(my_register) inc = {
    void apply (...) {
        val = val + 1;
        ...
    }
};

// C++

(* my_register)++;
P4’s voluminosity is due to two reasons: verbosity and limited parametrization

```p4
// P4

table my_table_0 {
    key = { field_0 : exact; }
    actions = { my_action; }
}

table my_table_1 {
    key = { field_1 : exact; }
    actions = { my_action; }
}

table my_table_2 {
    key = { field_2 : exact; }
    actions = { my_action; }
}
```
P4’s voluminosity makes it harder to write, debug, deploy, and maintain code.

Voluminous code takes longer to process.

Larger programs have higher probability of errors.

Errors can propagate across code repetitions.
P4 users rely on various tools to overcome the language voluminosity

<table>
<thead>
<tr>
<th>Tool</th>
<th>Number of Programmers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scripts</td>
<td>4</td>
</tr>
<tr>
<td>Templating tools</td>
<td>8</td>
</tr>
<tr>
<td>Macros</td>
<td>14</td>
</tr>
<tr>
<td>Copy-pasting</td>
<td>24</td>
</tr>
</tbody>
</table>
Recent works have proposed higher-level programming languages

- Directly fitting P4 into target hardware: Lyra, Chipmunk, P4All
- Adding macro–like annotations into P4: pcube
Recent works have proposed higher-level programming languages

- Directly fitting P4 into target hardware: Lyra, Chipmunk, P4All
- Adding macro–like annotations into P4: pcube

However, such languages ... extend expressivity
lose fine–grained control
Can we reduce P4 language’s voluminosity by introducing higher-level constructs?
Can we reduce P4 language’s voluminosity by introducing higher-level constructs…

... while preserving ... language expressivity, and fine-grained control?

Yep!

Introducing P4
O4 extends P4 with three simple constructs: *arrays, loops, and factories*

<table>
<thead>
<tr>
<th>Constructs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrays</td>
<td>Group variables of the same type</td>
</tr>
<tr>
<td>Loops</td>
<td>Reduce code-block repetitions</td>
</tr>
<tr>
<td>Factories</td>
<td>Enable parametrization, “everywhere”</td>
</tr>
</tbody>
</table>
With O4, we go from this ...

acc-turbo.p4: ~2K lines of code
... to this

acc-turbo.o4: ~350 lines of code
O4 arrays group variables of the same type

```
// O4

bit<32>[2][2] my_array;

my_array = [[0, 1], [2, 3]];

my_array[0][0];
```

```
// P4

bit<32> my_array_0_0;
bit<32> my_array_0_1;
bit<32> my_array_1_0;
bit<32> my_array_1_1;

my_array_0_0 = 0;
my_array_0_1 = 1;
my_array_1_0 = 2;
my_array_1_1 = 3;

my_array_0_0;
```
O4 loops reduce repetitions of code-blocks

// O4

for (int index in [0,1,2,3]){
    my_call(index);
}

// P4

my_call(0);
my_call(1);
my_call(2);
my_call(3);
O4 factories enable parametrization in “any” P4 construct

```c
// O4

factory my_factory (bit <8> field){
    table my_table {
        key = { field : exact; }
        actions = { my_action; }
    }
    return my_table ;
}

my_table_0 = my_factory (field_0);
my_table_1 = my_factory (field_1);
my_table_2 = my_factory (field_2);

// P4

table my_table_0 {
    key = { field_0 : exact; }
    actions = { my_action; }
}

table my_table_1 {
    key = { field_1 : exact; }
    actions = { my_action; }
}

table my_table_2 {
    key = { field_2 : exact; }
    actions = { my_action; }
}
```

O4 factories enable parametrization in “any” P4 construct
We implemented an **O4 compiler using Racket**
We evaluated O4 on several state-of-the-art P4 programs

We built a dataset of p4-16 programs
Open Tofino and P4-Learning
v1model and tna architectures

We evaluated …
... voluminosity reduction
... verbosity reduction
... code-clone reduction
We evaluated O4 on several state-of-the-art P4 programs

<table>
<thead>
<tr>
<th>Results</th>
<th>Volume</th>
<th>-42.4%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verbosity</td>
<td>-32.6%</td>
<td></td>
</tr>
<tr>
<td>Number of Clones</td>
<td>-56.3%</td>
<td></td>
</tr>
</tbody>
</table>
O4 performs **better for larger programs**
O4 performs *similarly to state-of-the-art* higher-level programming languages.
The O4 compiler is fast

Compilation time from P4 to Tofino

Compilation time from O4 to P4

~3.5%
Reducing P4 Language’s Voluminosity using Higher-Level Constructs

P4 language is highly voluminous
usually requiring thousands of lines of code

O4 introduces three high-level constructs
arrays, loops, and factories

O4 significantly decreases code volumes
preserving expressivity and low-level control

github.com/nsg-ethz/O4