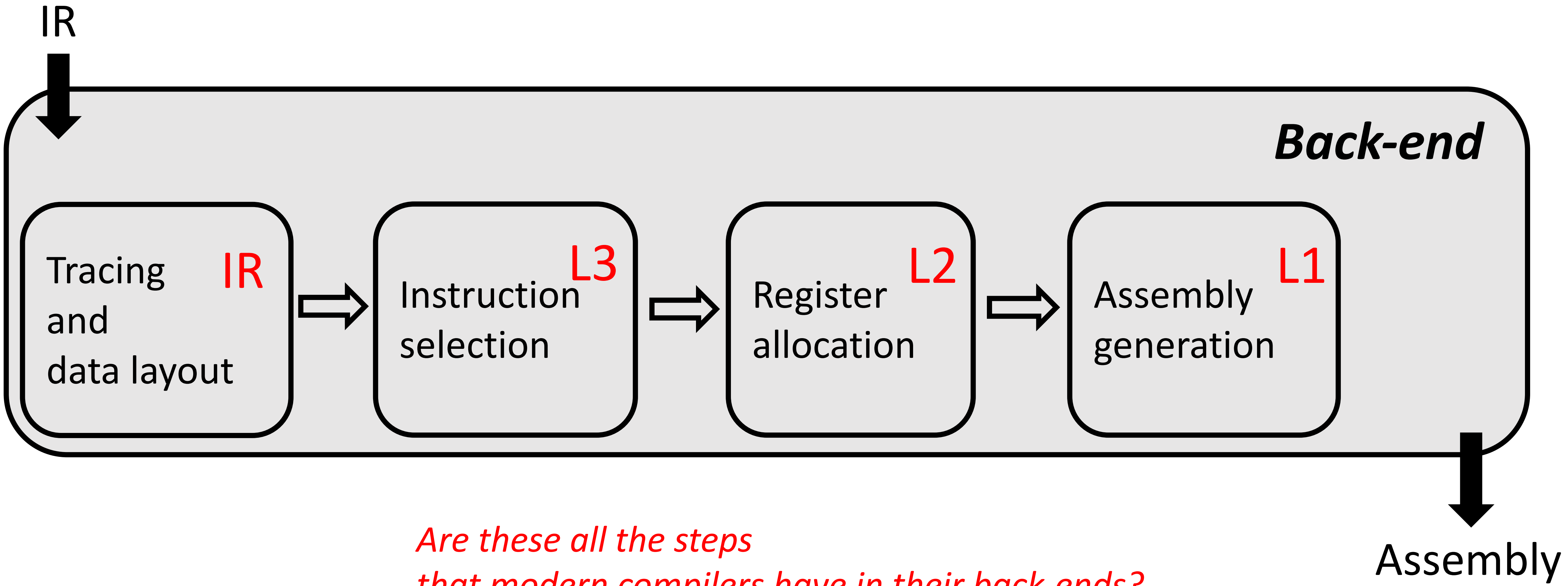
Compiler Construction

Back-end missing pieces

Simone Campanoni
simone.campanoni@northwestern.edu

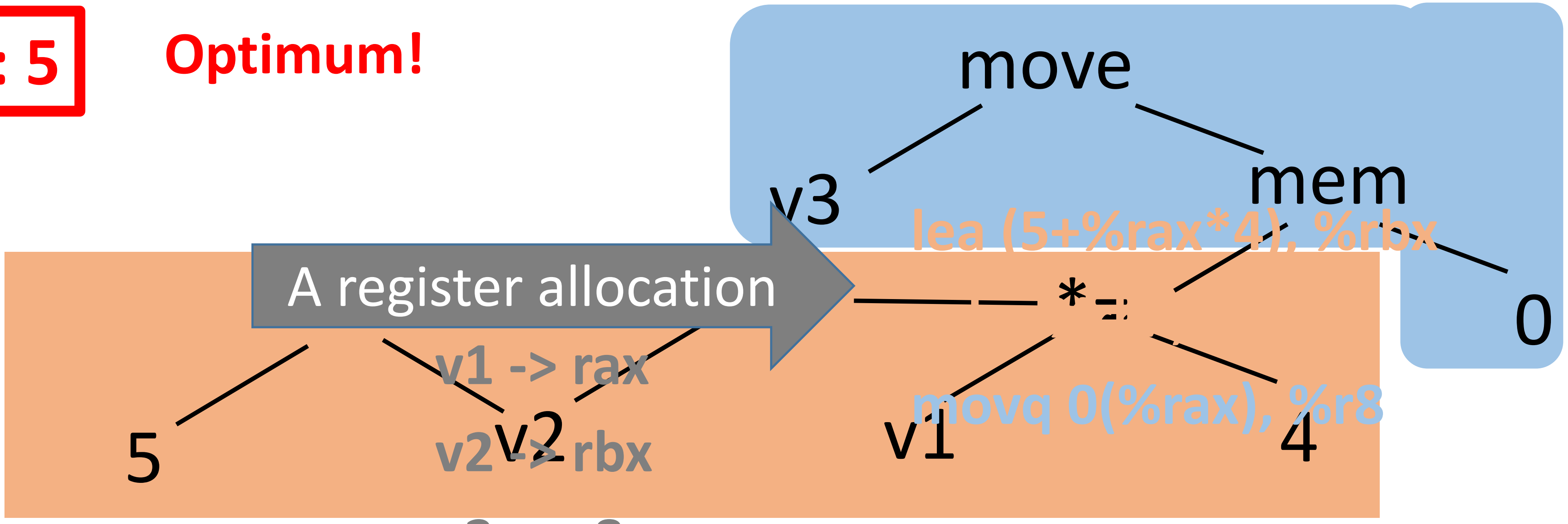
Instruction selection is part of the backend



Are these all the steps
that modern compilers have in their back-ends?
No

Register allocation after instruction selection

Total cost: 5 **Optimum!**



lea (5+%rax*4), %rbx

movq 0(%rax), %r8

v1 -> rax

v2 -> rbx

v3 -> r8

```
v1 *= 4
v2 <- v1
v2 += 5
v3 <- mem v1 0
```

```
lea (5+%v1*4), %v2
subq %v2, %v1
movq 0(%v1), %v3
```

Register allocation after instruction selection

```
lea (5+%v1*4), %v2
subq %v2, %v1
movq 0(%v1), %v3
```

A register allocation

v1 -> rax

v2 -> rbx

v3 -> stack O

Temporary register

```
lea (5+%rax*4), %rbx
subq %rbx, %rax
movq 0(%rax), %r10
movq %r10, O(%rsp)
```

v3

Register allocation after instruction selection

```
lea (5+%v1*4), %v2
subq %v2, %v1
movq 0(%v1), %v3
movq %v3, %v4
```

A register allocation

v1 -> rax

v2 -> rbx

v3 -> stack 0

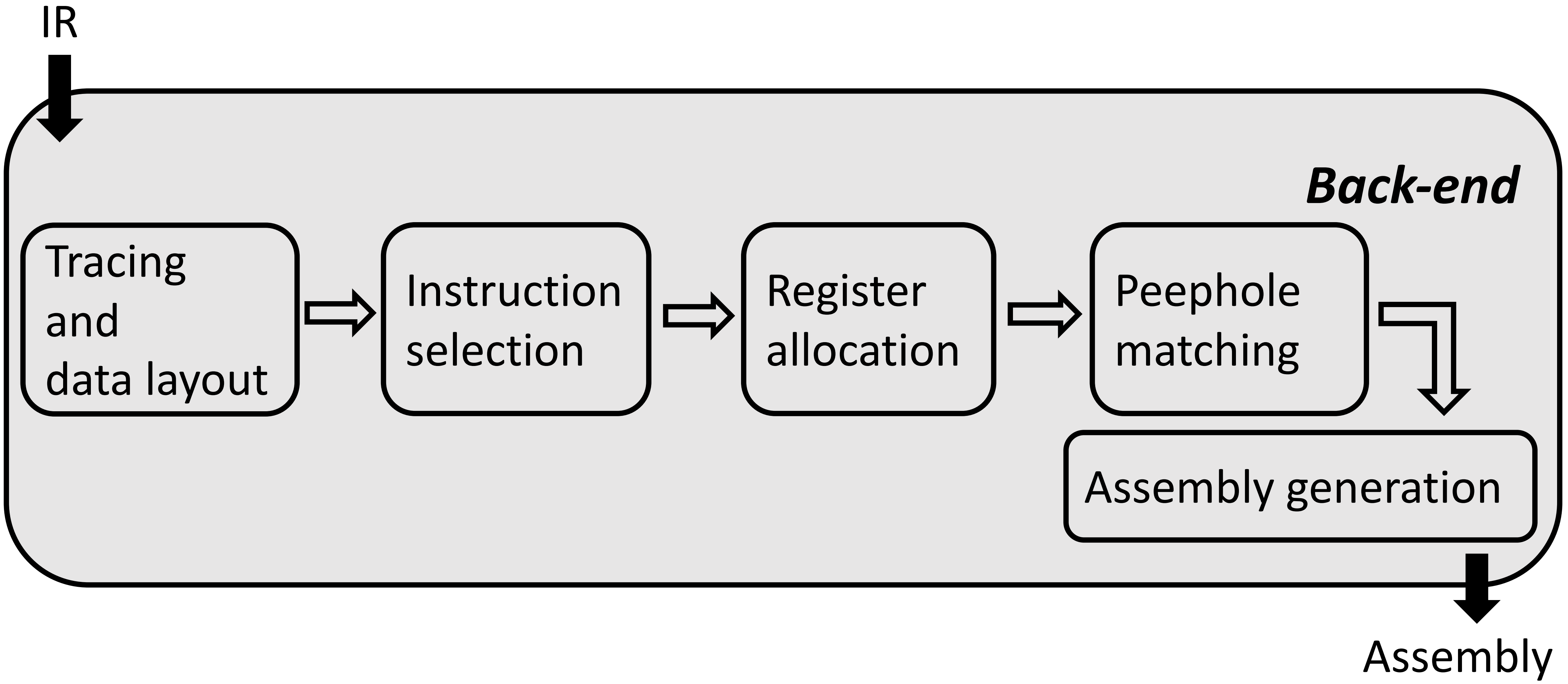
v4 -> r8

```
lea (5+%rax*4), %rbx
subq %rbx, %rax
movq 0(%rax), %r10
movq %r10, 0(%rsp)
movq 0(%rsp), %r8
```

Peephole matching

Wait, I thought we found the optimum …

Instruction selection is part of the backend

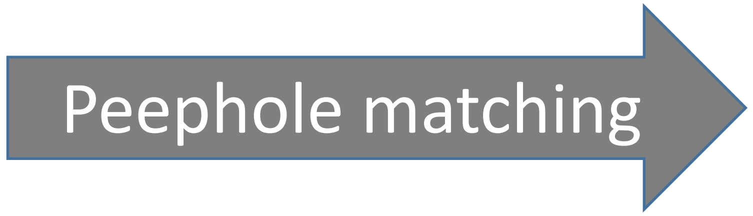


Peephole matching

- Basic idea: compiler can discover local improvements locally
  - Look at a small set of adjacent operations
  - Move a “peephole” over code & search for improvement

- Example: store followed by load

```
movq %r10, 0(%rsp)
movq 0(%rsp), %r8
```



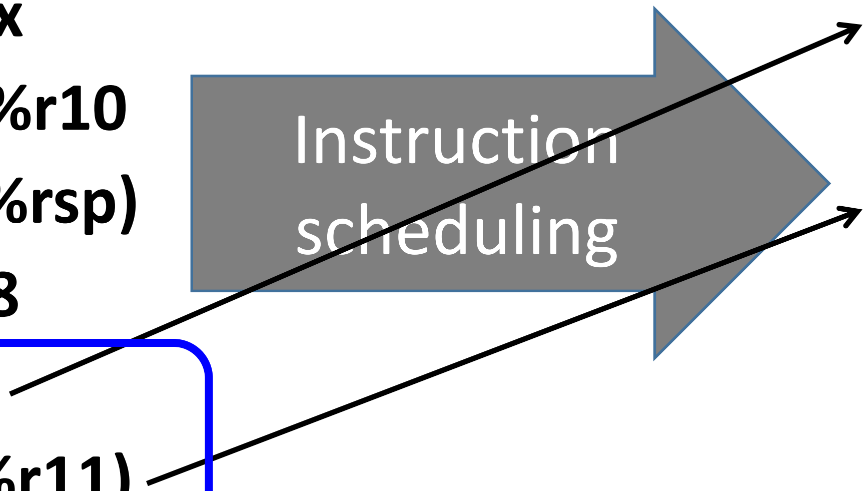
```
movq %r10, 0(%rsp)
movq %r10, %r8
```

Are we happy now with the generated assembly?

Of course NOT!

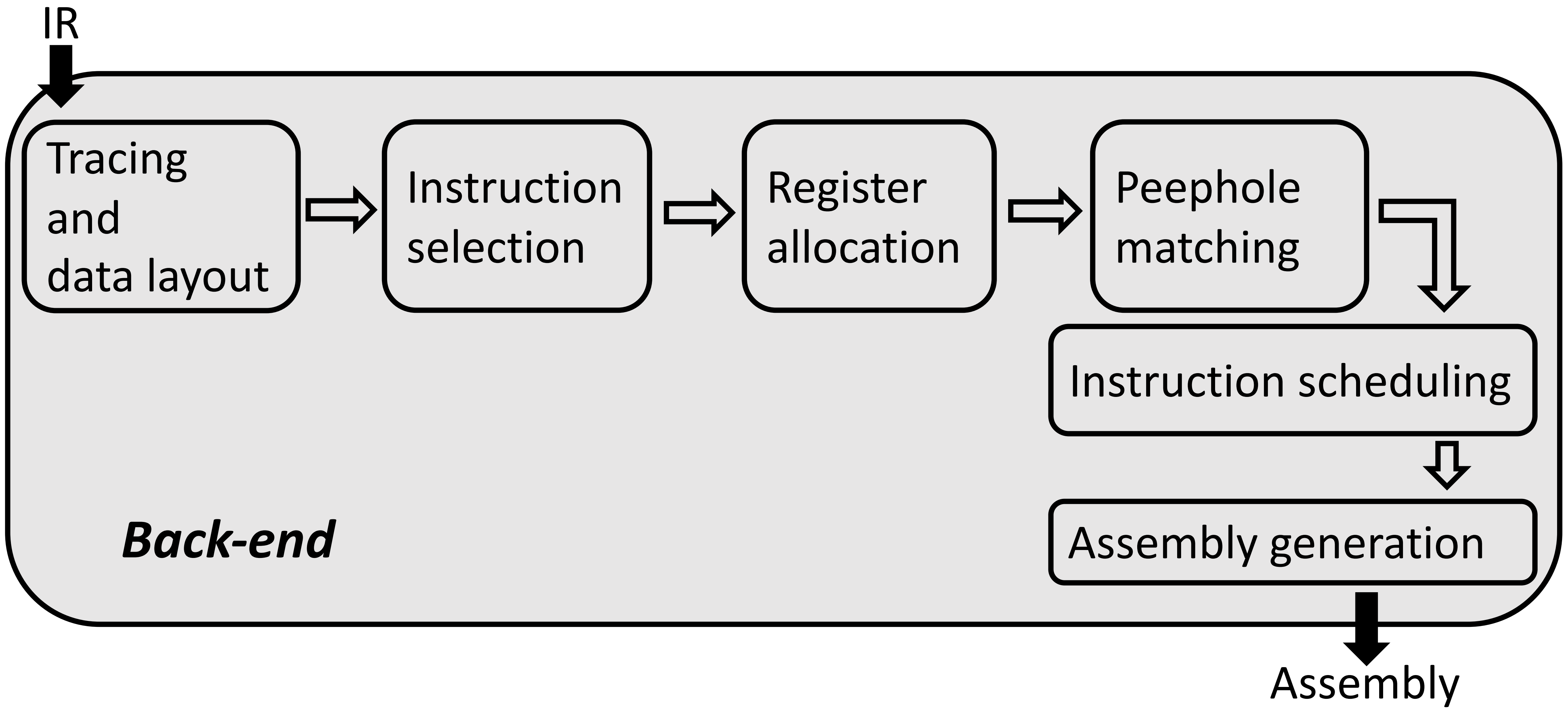
The problem left

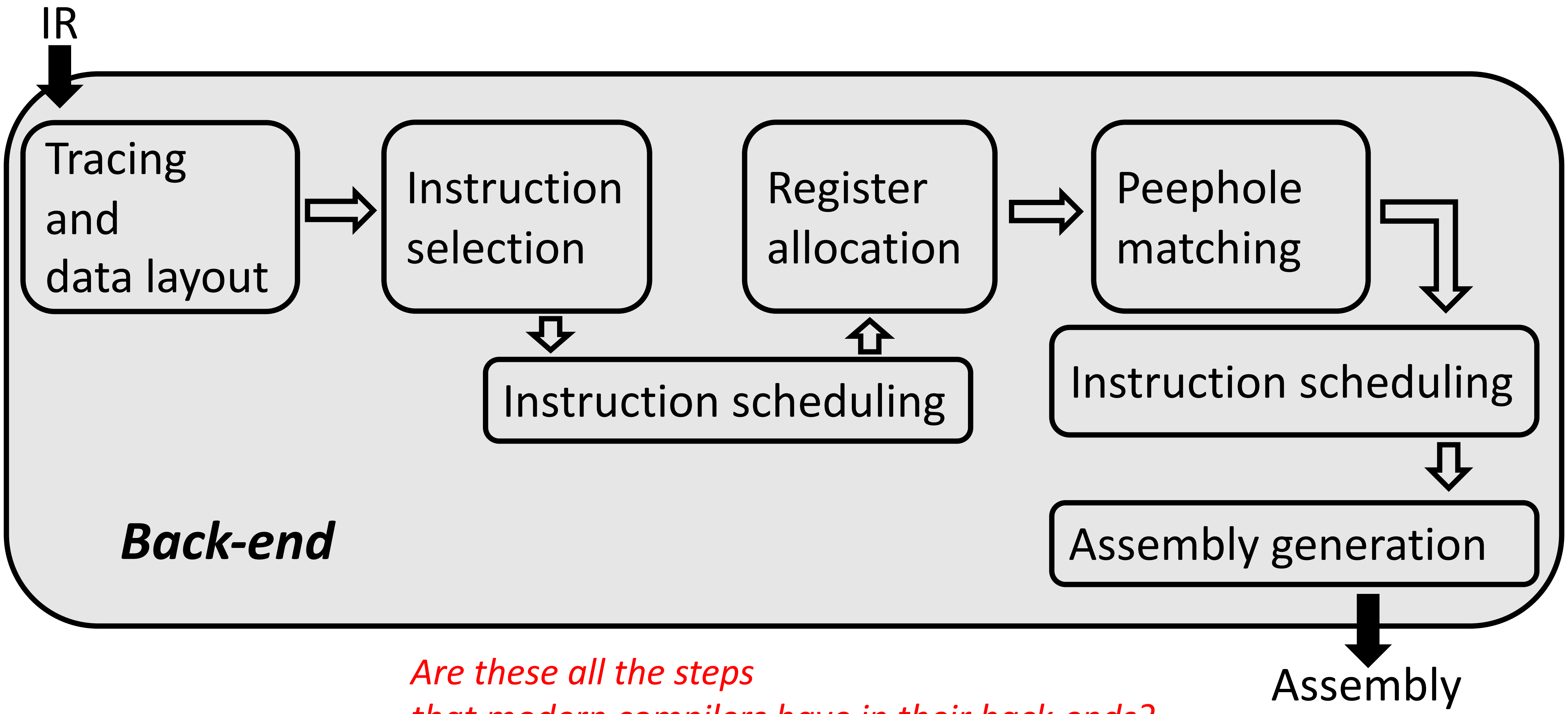
```
lea (5+%rax*4), %rbx
subq %rbx, %rax
movq 0(%rax), %r10
movq %r10, O(%rsp)
movq %r10, %r8
subq %r9, %r10
movq %r10, 0(%r11)
```



```
lea (5+%rax*4), %rbx
subq %r9, %r10
subq %rbx, %rax
movq %r10, 0(%r11)
movq 0(%rax), %r10
movq %r10, O(%rsp)
movq %r10, %r8
```

Better schedule of instructions


IR
Tracing and data layout
Instruction selection
Register allocation
Peephole matching
Instruction scheduling
Assembly generation
Back-end
Assembly

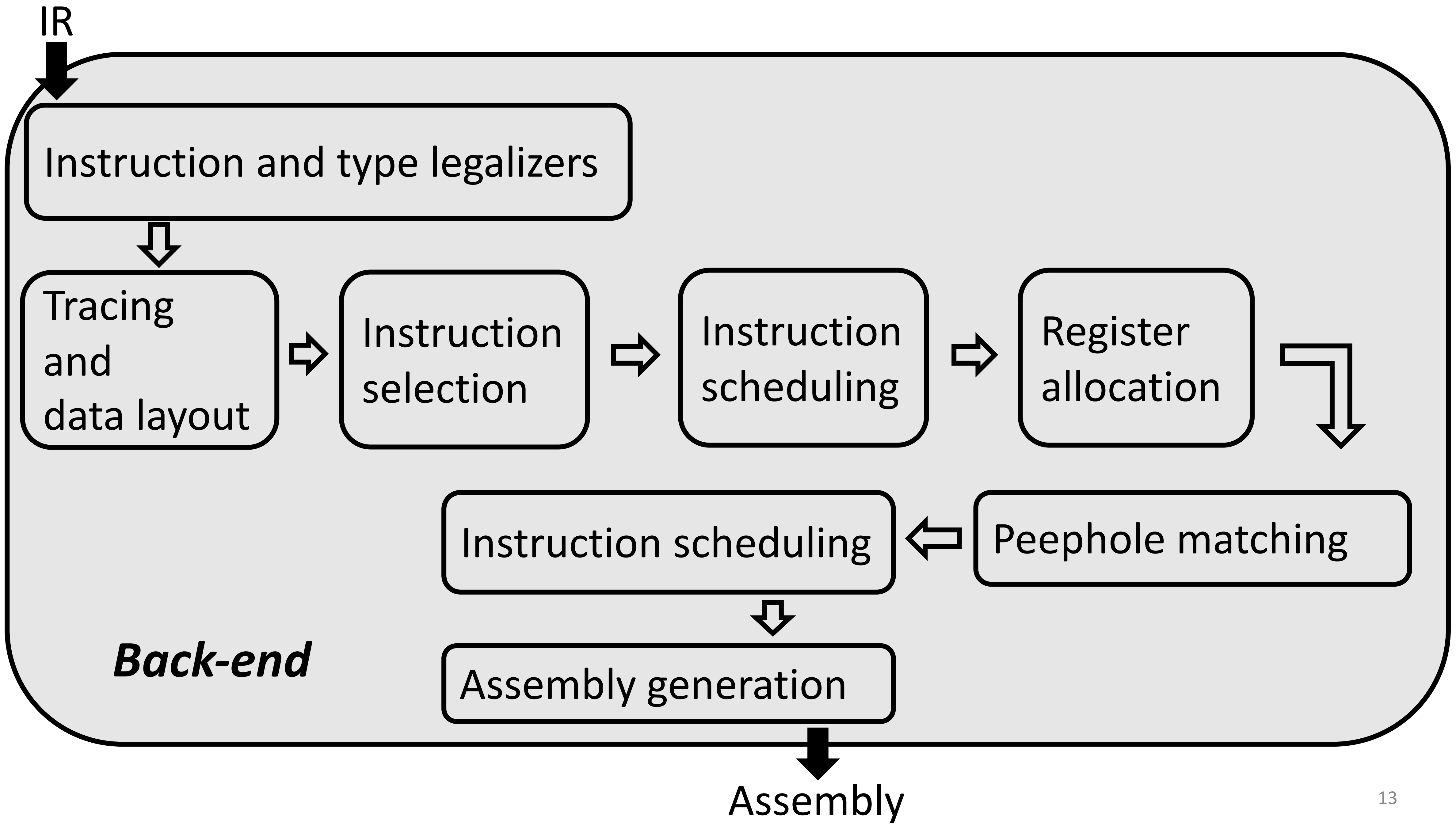


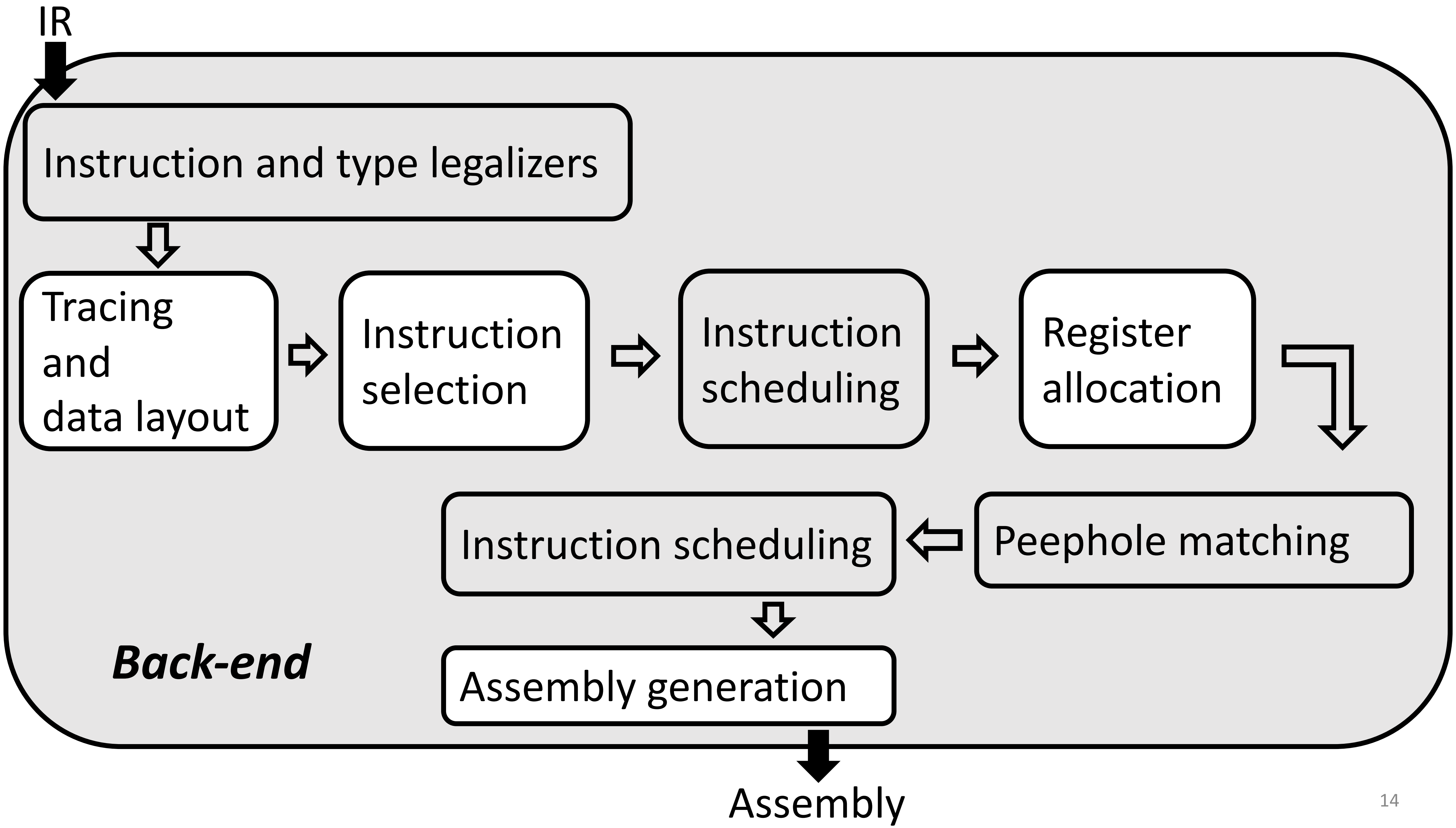
Are these all the steps
that modern compilers have in their back-ends?
Almost

Legalizer

- Some instructions and/or data types
  might not be valid at lower-level representations (e.g., L3)
  - E.g., 8-bits variables in IR exist,
    but all variables in lower-level representation are only at 32 bits

- Solution 0: instruction selection
  - Advantage: no extra compilation step
  - Disadvantage: it makes a complex compilation step even more complex
- Solution 1: legalizer
  - Changes the code to replace data types and instructions that are ”illegal” for the target architecture with “legal” data types and instructions


IR
Instruction and type legalizers
Tracing and data layout
Instruction selection
Instruction scheduling
Register allocation
Peephole matching
Instruction scheduling
Assembly generation
Back-end
Assembly


IR
Instruction and type legalizers
Tracing and data layout
Instruction selection
Instruction scheduling
Register allocation
Peephole matching
Instruction scheduling
Assembly generation
Back-end
Assembly

Always have faith in your ability

Success will come your way eventually

Best of luck!