UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

Compilers & Domain-Specific Languages

Roland Leißa

Introduction

Myth: Computers are fast enough

Monsters University, 2013, Pixar

Myth: Computers are fast enough

Monsters University, 2013, Pixar

- Rendering a single frame: >24 hours

Myth: Computers are fast enough

Monsters University, 2013, Pixar

- Rendering a single frame: >24 hours
- Rendering whole movie on a supercomputer:
  - 2000 machines
  - 24,000 cores
  - 2 years to render

Myth: Computers are fast enough

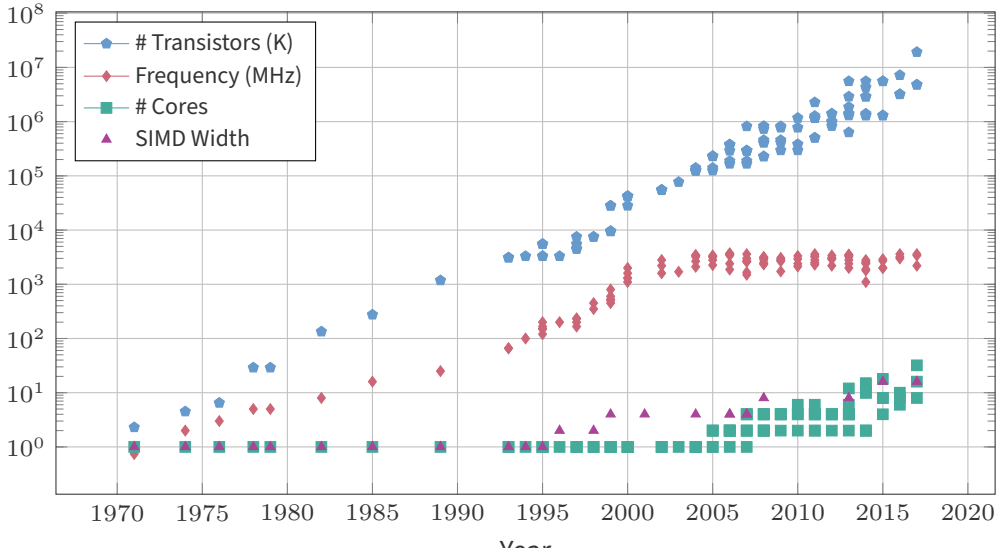
Monsters University, 2013, Pixar

- Rendering a single frame: >24 hours
- Rendering whole movie on a supercomputer:
  - 2000 machines
  - 24,000 cores
  - 2 years to render
- Doing this in real time?

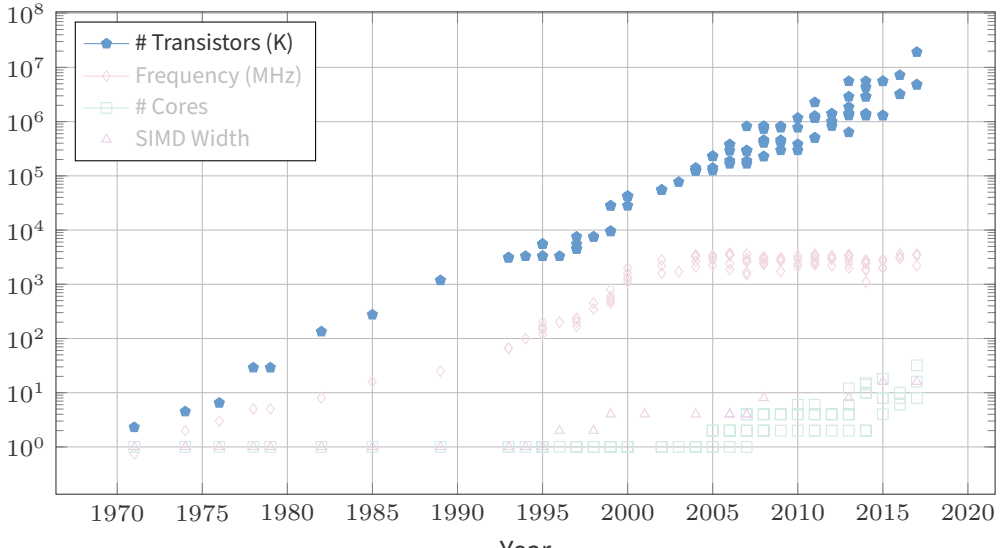
$$24\,h \cdot \frac{1}{2}^{\frac{t}{2}} = \frac{1}{60}\,s$$

$$t = 2 \cdot log_2(86400 \cdot 60) \approx 44.6 \text{ years}$$

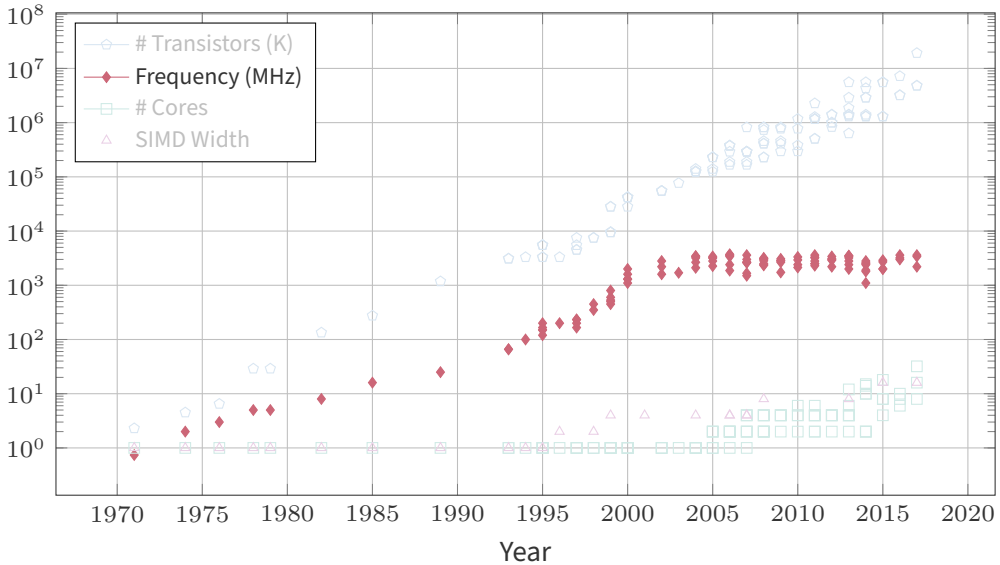
Moore and more



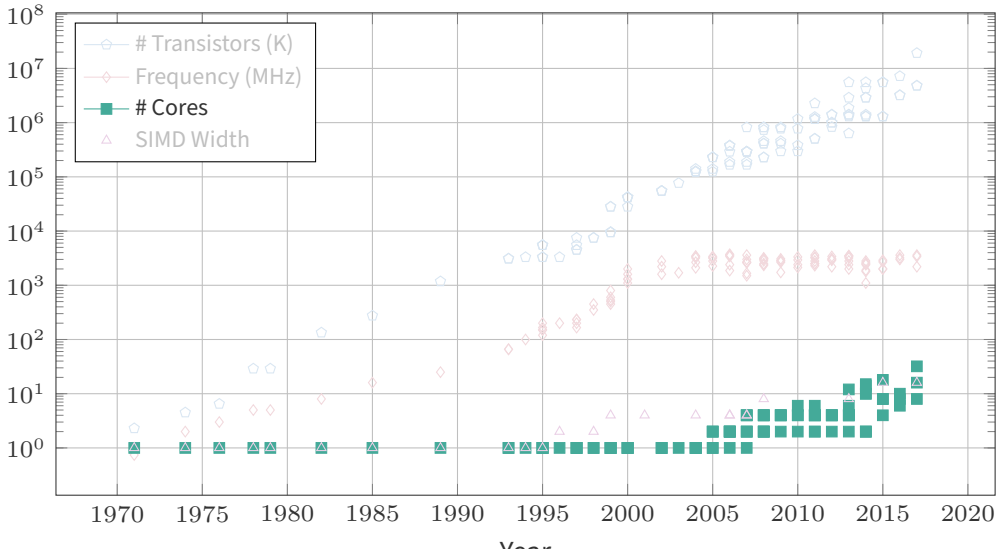
Moore and more



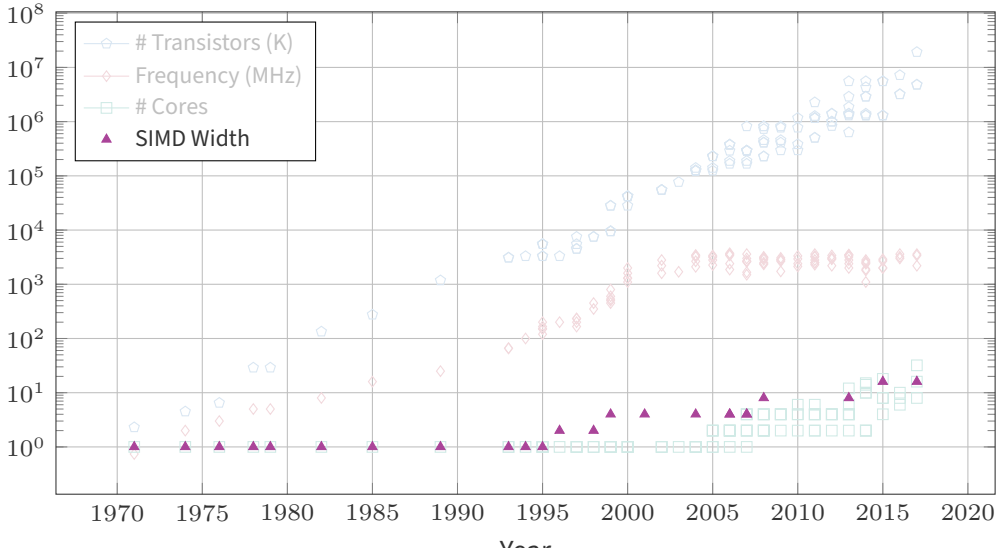
Moore and more



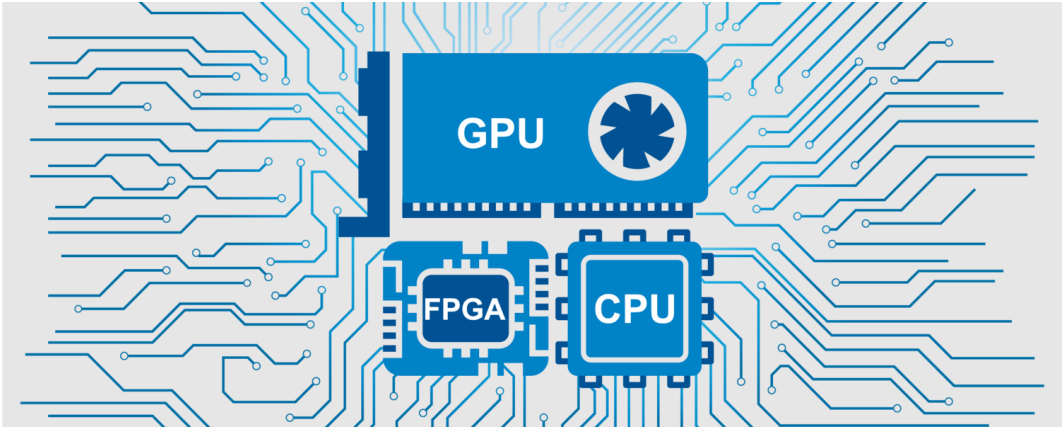
Moore and more



Moore and more



Heterogeneous Hardware



Training AI

Model	Training end	Chip type	TFLOP/s (max)	Chip count	Wall clock (days)	Total time (years)	Cost (US$)	MMLU ▲
GPT-3	Apr/2020	V100	130	10,000	15 days	405 years	$9M	43.9
Llama 1	Jan/2023	A100	312	2,048	21 days	118 years	$4M	63.4
Llama 2	Jun/2023	A100	312	*2,048*	35 days	196 years	$7M	68.0
GPT-4	Aug/2022	A100	312	25,000	95 days	6,507 years	$224M	86.4
Gemini	Nov/2023	TPUv4	275	57,000	100 days	15,616 years	$440M	90.0
GPT-5	*Apr/2024*	H100	989	*50,000*	*120 days*	*16,438 years*	*$612M*	
Llama 3	*Apr/2024*	H100	989					
Olympus	*Aug/2024*	H100	989					
Gemini 2	*Nov/2024*	TPUv5	393					

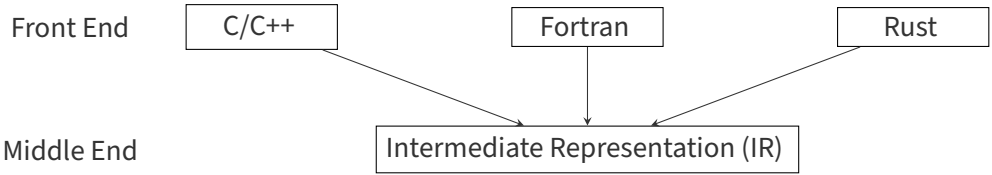
Alan D. Thompson. Feb/2024. LifeArchitect.ai

Table. Google DeepMind Gemini training compute (see working, with sources[8]).

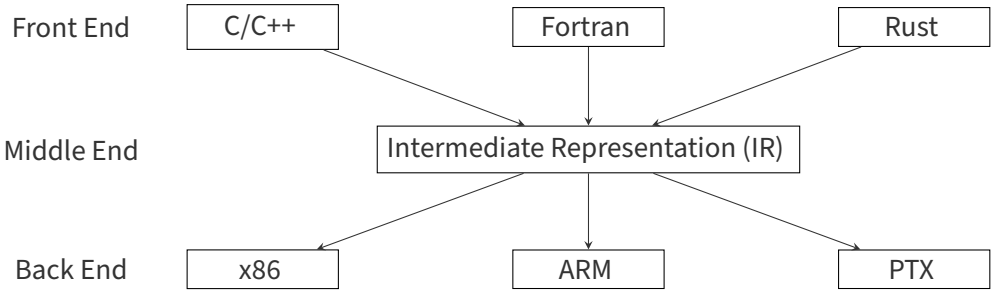
Structure of a Compiler

Architecture

Architecture



Architecture



Running Example

```
float f(float in) {
    float out;
    out = 2 * in;
    return out;
}
```

Running Example

```
float f(float in) {
    float out;
    out = 2 * in;
    return out;
}
```

Lexical Analysis

- **Input:** Sequence of Characters

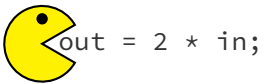
Lexical Analysis

- **Input:** Sequence of Characters
- **Tasks:**
  - Deconstruct incoming text into meaningful chunks called Tokens
  - Discard useless characters (whitespace, comments)

Lexical Analysis

- **Input:** Sequence of Characters
- **Tasks:**
  - Deconstruct incoming text into meaningful chunks called Tokens
  - Discard useless characters (whitespace, comments)
- **Output:** Sequence of tokens

Lexical Analysis



Lexical Analysis

o  ut = 2 * in;

Lexical Analysis

ou  t = 2 * in;

Lexical Analysis

⟨**id**, `out`⟩  `= 2 * in;`

Lexical Analysis

⟨**id**, `out`⟩  `= 2 * in;`

Lexical Analysis

⟨**id**, `out`⟩ ⟨**=**⟩  `2 * in;`

Lexical Analysis

⟨**id**, `out`⟩ ⟨**=**⟩  `2 * in;`

Lexical Analysis

⟨**id**, `out`⟩ ⟨**=**⟩ ⟨**lit**, `2`⟩  `* in;`

Lexical Analysis

⟨**id**, `out`⟩ ⟨**=**⟩ ⟨**lit**, `2`⟩  `* in;`

Lexical Analysis

⟨**id**, `out`⟩ ⟨**=**⟩ ⟨**lit**, `2`⟩ ⟨*****⟩  `in;`

Lexical Analysis

⟨**id**, `out`⟩ ⟨**=**⟩ ⟨**lit**, `2`⟩ ⟨*****⟩  `in;`

Lexical Analysis

⟨**id**, `out`⟩ ⟨**=**⟩ ⟨**lit**, `2`⟩ ⟨*****⟩ `i`  `n;`

Lexical Analysis

⟨**id**, `out`⟩ ⟨**=**⟩ ⟨**lit**, `2`⟩ ⟨*****⟩ ⟨**id**, `in`⟩

Lexical Analysis

⟨**id**, `out`⟩ ⟨**=**⟩ ⟨**lit**, `2`⟩ ⟨*****⟩ ⟨**id**, `in`⟩ ⟨**;**⟩

Syntax Analysis

- **Input:** Sequence of Tokens

Syntax Analysis

- **Input:** Sequence of Tokens
- **Tasks**
  - Checks if input adheres to context-free grammar of source language
  - Detects hierarchical structure of input

Syntax Analysis

- **Input:** Sequence of Tokens
- **Tasks**
  - Checks if input adheres to context-free grammar of source language
  - Detects hierarchical structure of input
- **Output:** Abstract syntax tree (AST)

Syntax Analysis

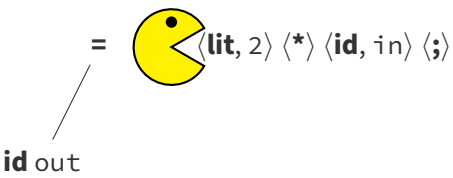
⟨**id**, `out`⟩ ⟨**=**⟩ ⟨**lit**, `2`⟩ ⟨*****⟩ ⟨**id**, `in`⟩ ⟨**;**⟩

Syntax Analysis

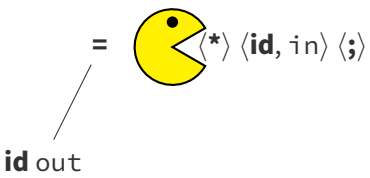
⟨**=**⟩ ⟨**lit**, 2⟩ ⟨***⟩ ⟨**id**, `in`⟩ ⟨**;**⟩

id `out`

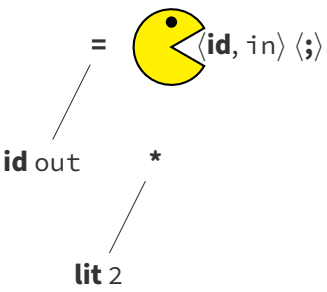
Syntax Analysis



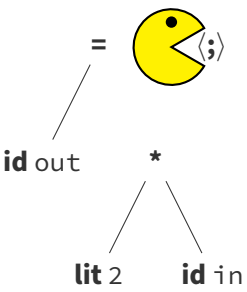
Syntax Analysis



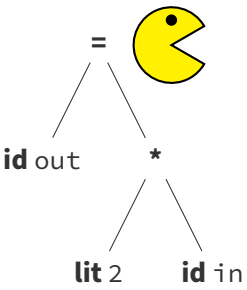
Syntax Analysis



Syntax Analysis



Syntax Analysis



Semantic Analysis

- **Input:** AST

Semantic Analysis

- **Input:** AST
- **Tasks:**
  - **Name analysis:** Relate uses to definitions of names
  - **Type analysis:** Elaborate and check the types of entities

Semantic Analysis

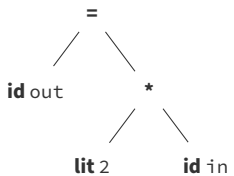
- **Input:** AST
- **Tasks:**
  - **Name analysis:** Relate uses to definitions of names
  - **Type analysis:** Elaborate and check the types of entities
- **Output:** Attributed AST

Semantic Analysis

```
float f(float in) {
    float out;
    out = 2 * in;
    return out;
}
```

Semantic Analysis

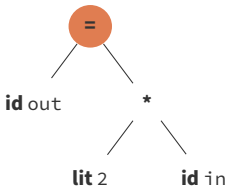
```
float f(float in) {
    float out;
```




```
    return out;
}
```

Semantic Analysis

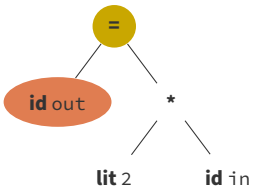
```
float f(float in) {
    float out;
```




```
    return out;
}
```

Semantic Analysis

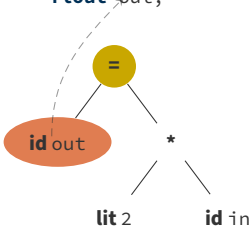
```
float f(float in) {
    float out;
```




```
    return out;
}
```

Semantic Analysis

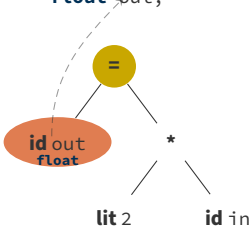
```
float f(float in) {
    float out;
```




```
    return out;
}
```

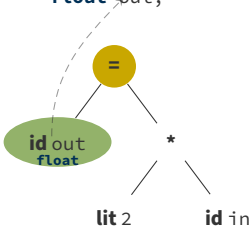
Semantic Analysis

```
float f(float in) {
    float out;

    return out;
}
```



Semantic Analysis

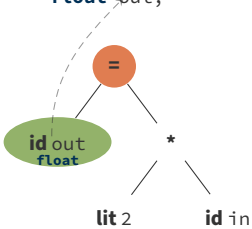
```
float f(float in) {
    float out;
```




```
    return out;
}
```

Semantic Analysis

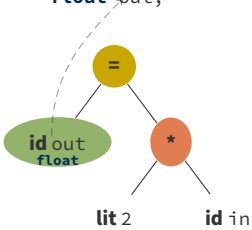
```
float f(float in) {
    float out;
```




```
    return out;
}
```

Semantic Analysis

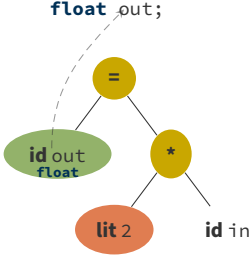
```
float f(float in) {
    float out;
```




```
    return out;
}
```

Semantic Analysis

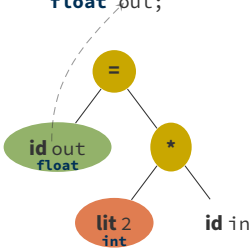
```
float f(float in) {
    float out;
```




```
    return out;
}
```

Semantic Analysis

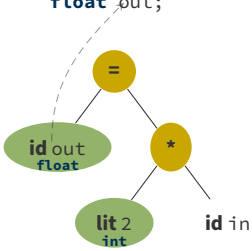
```
float f(float in) {
    float out;
```




```
    return out;
}
```

Semantic Analysis

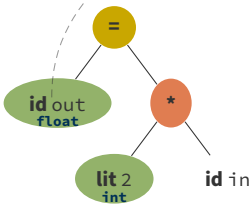
```
float f(float in) {
    float out;
```




```
    return out;
}
```

Semantic Analysis

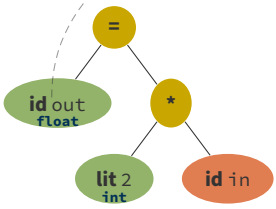
```
float f(float in) {
    float out;
```




```
    return out;
}
```

Semantic Analysis

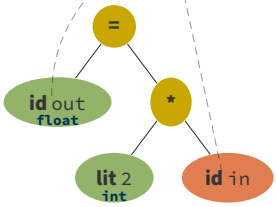
```
float f(float in) {
    float out;
```




```
    return out;
}
```

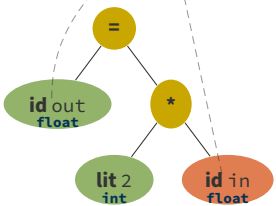
Semantic Analysis

```
float f(float in) {
    float out;

    return out;
}
```



Semantic Analysis

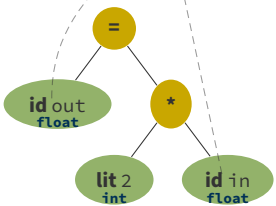
```
float f(float in) {
    float out;
```




```
    return out;
}
```

Semantic Analysis

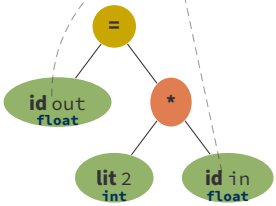
```
float f(float in) {
    float out;
```




```
    return out;
}
```

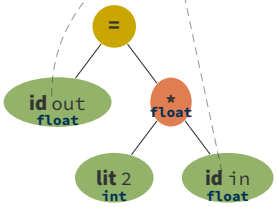
Semantic Analysis

```
float f(float in) {
    float out;

    return out;
}
```



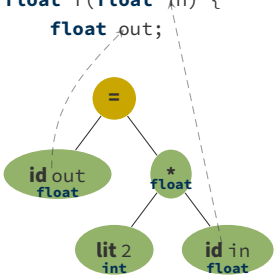
Semantic Analysis

```
float f(float in) {
    float out;

    return out;
}
```



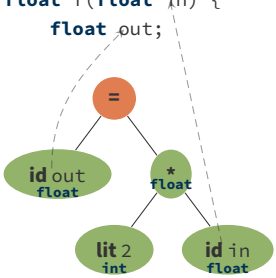
Semantic Analysis

```
float f(float in) {
    float out;

    return out;
}
```



Semantic Analysis

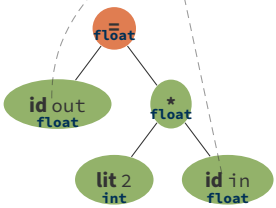
```
float f(float in) {
    float out;
```




```
    return out;
}
```

Semantic Analysis

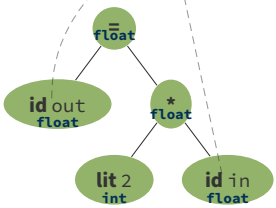
```
float f(float in) {
    float out;
```




```
    return out;
}
```

Semantic Analysis

```
float f(float in) {
    float out;

    return out;
}
```



Transformation

- **Input:** Attributed AST

Transformation

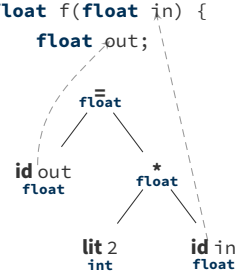
- **Input:** Attributed AST
- **Tasks:** Lower high-level language constructs to target code

Transformation

- **Input:** Attributed AST
- **Tasks:** Lower high-level language constructs to target code
- **Output:** Machine code or intermediate representation (IR)

Transformation

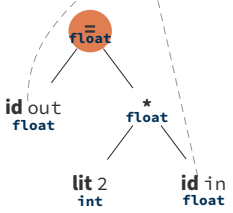
```
float f(float in) {
    float out;
```




```
    return out;
}
```

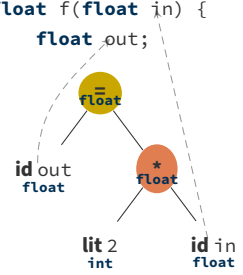
Transformation

```
float f(float in) {
    float out;

    return out;
}
```



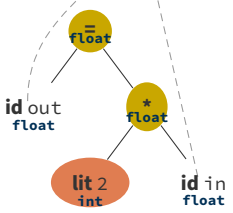
Transformation

```
float f(float in) {
    float out;

    return out;
}
```



Transformation

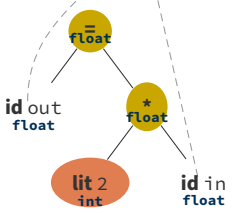
```
float f(float in) {
    float out;
```




```
    return out;
}
```

Transformation

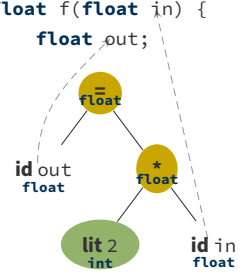
```
float f(float in) {
    float out;

    return out;
}
```



2

Transformation

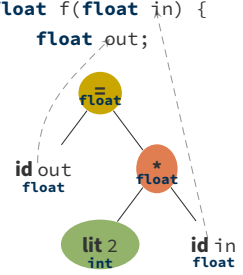
```
float f(float in) {
    float out;

        return out;
    }
```



2

Transformation

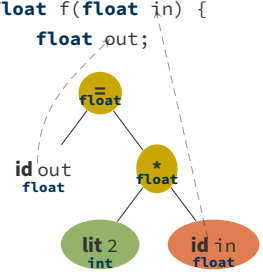
```
float f(float in) {
    float out;

    return out;
}
```




```
t1 = inttofloat 2
```

Transformation

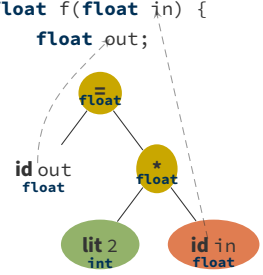
```
float f(float in) {
    float out;

    return out;
}
```




```
t1 = inttofloat 2
```

Transformation

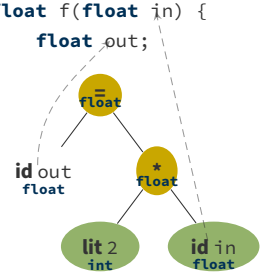
```
float f(float in) {
    float out;

    return out;
}
```




```
t1 = inttofloat 2
t2 = load in
```

Transformation

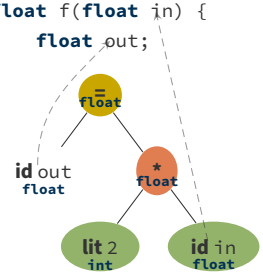
```
float f(float in) {
    float out;

    return out;
}
```




```
t1 = inttofloat 2
t2 = load in
```

Transformation

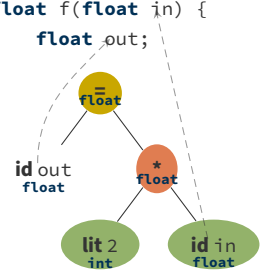
```
float f(float in) {
    float out;

    return out;
}
```




```
t1 = inttofloat 2
t2 = load in
```

Transformation

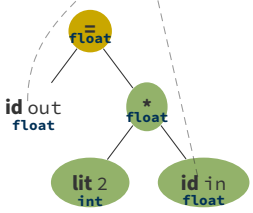
```
float f(float in) {
    float out;

    return out;
}
```




```
t1 = inttofloat 2
t2 = load in
t3 = mul t1, t2
```

Transformation

```
float f(float in) {
    float out;
```

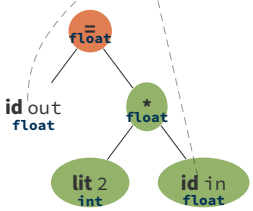



```
    return out;
}
```

```
t1 = inttofloat 2
t2 = load in
t3 = mul t1, t2
```

Transformation

```
float f(float in) {
    float out;
```

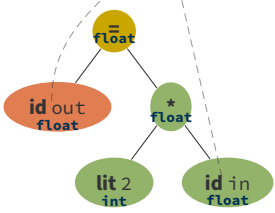



```
    return out;
}
```

```
t1 = inttofloat 2
t2 = load in
t3 = mul t1, t2
```

Transformation

```
float f(float in) {
    float out;
```

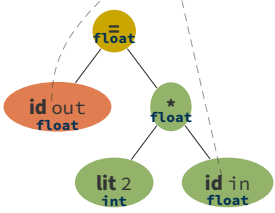



```
    return out;
}
```

```
t1 = inttofloat 2
t2 = load in
t3 = mul t1, t2
```

Transformation

```
float f(float in) {
    float out;
```

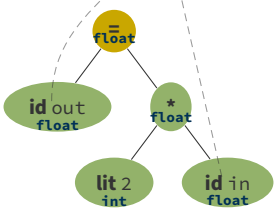



```
    return out;
}
```

```
t1 = inttofloat 2
t2 = load in
t3 = mul t1, t2
          out
```

Transformation

```
float f(float in) {
    float out;
```

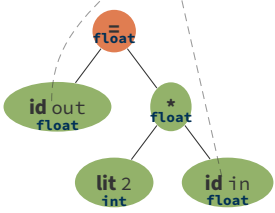



```
    return out;
}
```

```
t1 = inttofloat 2
t2 = load in
t3 = mul t1, t2
          out
```

Transformation

```
float f(float in) {
    float out;
```

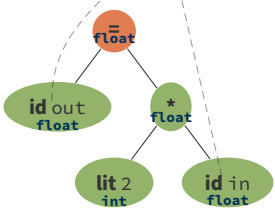



```
    return out;
}
```

```
t1 = inttofloat 2
t2 = load in
t3 = mul t1, t2
          out
```

Transformation

```
float f(float in) {
    float out;
```

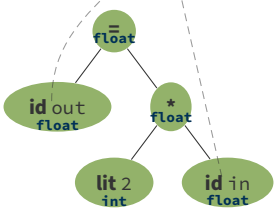



```
    return out;
}
```

```
t1 = inttofloat 2
t2 = load in
t3 = mul t1, t2
store t3, out
```

Transformation

```
float f(float in) {
    float out;
```




```
    return out;
}
```

```
t1 = inttofloat 2
t2 = load in
t3 = mul t1, t2
store t3, out
```

Optimization

- **Input:** IR

Optimization

- **Input:** IR
- **Tasks:**
  - Reduce resource consumption: execution time, memory
  - Preserve semantics

Optimization

- **Input:** IR
- **Tasks:**
  - Reduce resource consumption: execution time, memory
  - Preserve semantics
- **Output:** IR

Optimization

```
t1 = inttofloat 2
t2 = load in
t3 = mul t1, t2
store t3, out
```

Optimization

```
t1 = 2.f
t2 = load in
t3 = mul t1, t2
store t3, out
```

Optimization

```
t2 = load in
t3 = mul 2.f, t2
store t3, out
```

Optimization

```
t2 = load in
t3 = add t2, t2
store t3, out
```

Code Generation

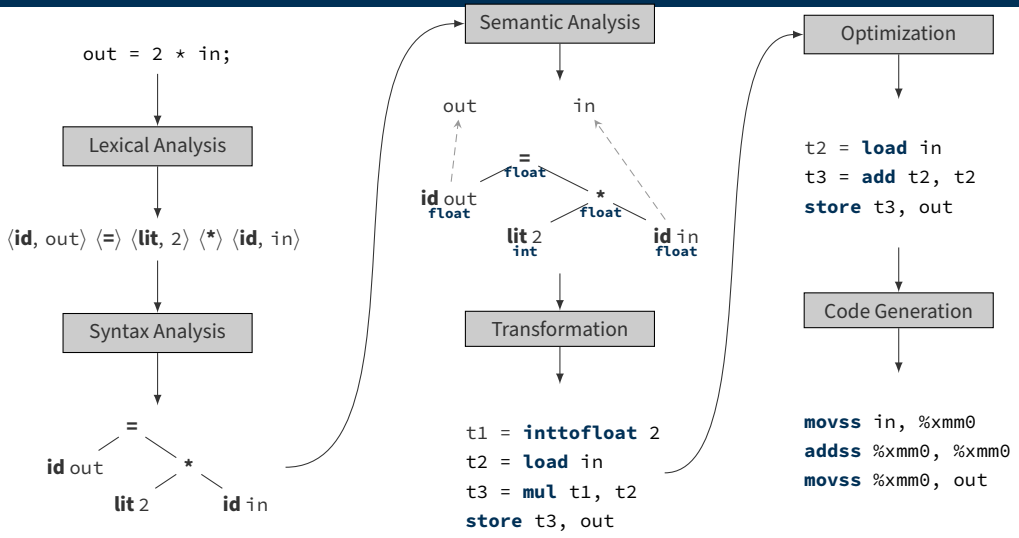
- **Input:** IR
- **Tasks:**
  - Instruction selection: use ISA of target arch
  - Scheduling: re-order instructions
  - Register allocation: allocate registers to IR vars
- **Output:** Machine code

Code Generation

```
t2 = load in
t3 = add t2, t2
store t3, out
```

```
movss in, %xmm0
addss %xmm0, %xmm0
movss %xmm0, out
```

Summary


out = 2 * in;
Lexical Analysis
⟨id, out⟩ ⟨=⟩ ⟨lit, 2⟩ ⟨*⟩ ⟨id, in⟩
Syntax Analysis
=
id out
*
lit 2
id in
Semantic Analysis
out
in
=
float
id out
float
*
float
lit 2
int
id in
float
Transformation
t1 = inttofloat 2
t2 = load in
t3 = mul t1, t2
store t3, out
Optimization
t2 = load in
t3 = add t2, t2
store t3, out
Code Generation
movss in, %xmm0
addss %xmm0, %xmm0
movss %xmm0, out

Domain-Specific Languages

General Purpose vs. Domain-Specific Languages

General Purpose Language

- Java
- C/C++
- Rust
- Fortran
- Python
- Perl

Domain-Specific Languages

- HTML
- SQL
- MATLAB
- RenderMan, GLSLang, HLSL, …
- GraphViz
- MLL

Embedded DSLs

SQL

```
SELECT name, age
    FROM person
    WHERE age >= 18
```

C++ → SQL

```
string sql =
"SELECT name, age
    FROM person
    WHERE age >= 18";
mysql_query(conn, sql);
```

C++ → sqlpp11 (Embedded DSL)

```
select(person.name, person.age)
    .from(person)
    .where(person.age >= 18)
```

Embedded DSLs

- SQL libraries
- Halide
- PyTorch
- TensorFlow
- JAX
- Z3 (SMT solver)
- MLL

Not DSL

```
class Vec2:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, v):
        return Vec2(self.x + v.x, self.y + v.y)
```

```
r = v + w
```

Embedded DSL

```
class Vec2:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, v):
        return Add(self, v)

class Add:
    def __init__(self, lhs, rhs):
        self.lhs = lhs
        self.rhs = rhs

r = v + w
```

Scaled Exponential Linear Units (SELU)

```
def selu(x, alpha=1.67, lmbda=1.05):
    return lmbda * jnp.where(x > 0, x, alpha * jnp.exp(x) - alpha)

x = random.normal(key, (1000000,))
%timeit selu(x).block_until_ready()
```

```
100 loops, best of 5: 2.05 ms per loop
```

Scaled Exponential Linear Units (SELU)

```
def selu(x, alpha=1.67, lmbda=1.05):
    return lmbda * jnp.where(x > 0, x, alpha * jnp.exp(x) - alpha)

x = random.normal(key, (1000000,))
%timeit selu(x).block_until_ready()
```

```
100 loops, best of 5: 2.05 ms per loop
```

Just-In-Time Compilation (JIT)

```
selu_jit = jax.jit(selu)
selu_jit(x).block_until_ready() # Warm up

%timeit selu_jit(x).block_until_ready()
```

```
10000 loops, best of 5: 150 µs per loop
```

Grad

```
def sum_logistic(x):
    return jnp.sum(1.0 / (1.0 + jnp.exp(-x)))

x_small = jnp.arange(3.)
derivative_fn = grad(sum_logistic)
print(derivative_fn(x_small))
```

```
[0.25       0.19661194 0.10499357]
```

Grad

```
def sum_logistic(x):
    return jnp.sum(1.0 / (1.0 + jnp.exp(-x)))

x_small = jnp.arange(3.)
derivative_fn = grad(sum_logistic)
print(derivative_fn(x_small))
```

```
[0.25       0.19661194 0.10499357]
```

Grad + JIT

```
derivative_fn_jit = jax.jit(grad(sum_logistic))
```

Logistic Function

$$\frac{1}{1 + \exp(-x)}$$

Java

```
var dag         = new DAG();
var logistic    = dag.lit1().div(dag.lit1().add(x().neg().exp()));
var derivative  = logistic.backwards();
derivative.llvm("logistic.ll"); // "JIT"
```

Logistic Function

$$\frac{1}{1 + \exp(-x)}$$

Java

```
var dag         = new DAG();
var logistic    = dag.lit1().div(dag.lit1().add(x().neg().exp()));
var derivative  = logistic.backwards();
derivative.llvm("logistic.ll"); // "JIT"
```

Python

```
dag        = mll.DAG()
logistic   = 1.0 / (1.0 + (dag.exp(-dag.x)))
derivative = logistic.backwards()
derivative.llvm("logistic.ll") # "JIT"
```

LLVM

LLVM

- open source
- large, active research community
- used in industry:

  Apple, Google, Intel, NVIDIA, Sony, …
- front-ends for many languages:

  C/C++, Fortran, Rust, Swift, Julia, Haskell, …
- back-ends for many architectures:

  X86(-64), ARM, RISC-V, MIPS, WebAssembly, …
- HUGE

LLVM

- open source
- large, active research community
- used in industry:

  Apple, Google, Intel, NVIDIA, Sony, …
- front-ends for many languages:

  C/C++, Fortran, Rust, Swift, Julia, Haskell, …
- back-ends for many architectures:

  X86(-64), ARM, RISC-V, MIPS, WebAssembly, …
- HUGE

LLVM

- open source
- large, active research community
- used in industry:

  Apple, Google, Intel, NVIDIA, Sony, …
- front-ends for many languages:

  C/C++, Fortran, Rust, Swift, Julia, Haskell, …
- back-ends for many architectures:

  X86(-64), ARM, RISC-V, MIPS, WebAssembly, …
- HUGE

LLVM Intermediate Representation

- SSA-based representation of control flow graphs
- dumpable in human-readable, assembly-like form (`*.ll`)
- dumpable as compact bitcode (`*.bc`)

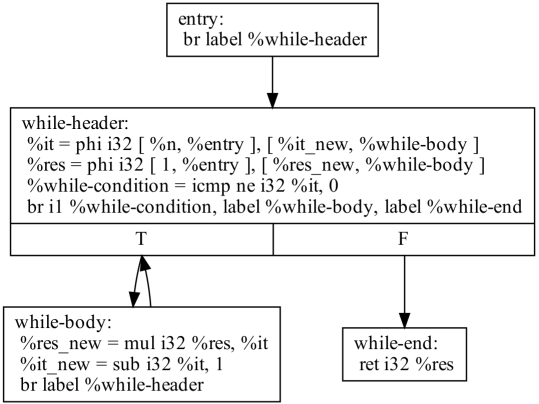
MLL: generate human-readable `*.ll` file

LLVM Intermediate Representation — Example

```
define i32 @fac(i32 %n) {
entry:
  br label %while-header

while-header:
  %it  = phi i32 [ %n, %entry ], [ %it_new,  %while-body ]
  %res = phi i32 [  1, %entry ], [ %res_new, %while-body ]
  %while-condition = icmp ne i32 %it, 0
  br i1 %while-condition, label %while-body, label %while-end

while-body:
  %res_new = mul i32 %res, %it
  %it_new  = sub i32 %it, 1
  br label %while-header

while-end:
  ret i32 %res
}
```

```
int fac(int n) {
    int it  = n;
    int res = 1;

    while (it != 0) {


        res = res * it;
        it  = it - 1;
    }


    return res;
}
```

LLVM Intermediate Representation — Example



CFG for 'fac' function

Useful Commands

- Generate (human readable) LLVM-IR from C/C++ input:

  ```
  clang -Xclang -disable-O0-optnone -emit-llvm -c -S -o OUT.ll IN.c
  ```

- Draw CFG of function `foo` from dumped LLVM-IR module:

  ```
  opt -dot-cfg IN.ll; dot -Tpdf cfg.foo.dot > OUT.pdf
  ```

  requires `dot/graphviz`

- Execute dumped LLVM-IR module:

  ```
  lli IN.ll <argv arguments>
  ```

- Create binary from dumped LLVM-IR module:

  ```
  clang -o OUT IN.ll
  ```

- Create architecture-specific assembly:

  ```
  llc -o OUT.s IN.ll
  ```

- Create binary from architecture specific-assembly:

  ```
  cc -o OUT IN.s
  ```

- Get more help:

  ```
  <TOOL> --help
  ```

Getting Help

- General language reference manual:

  `http://llvm.org/docs/LangRef.html`

- Doxygen code documentation:

  (well accessible via Google/Bing/DuckDuckGo/…)

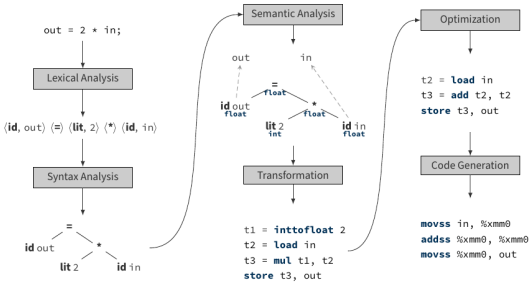
  `http://llvm.org/doxygen/index.html`

- Full command line tools guide:

  `http://llvm.org/docs/CommandGuide/`

Conclusion

Conclusion



SQL

```
SELECT name, age
    FROM person
    WHERE age >= 18
```

C++ → SQL

```
string sql =
"SELECT name, age
    FROM person
    WHERE age >= 18";
mysql_query(conn, sql);
```

C++ → sqlpp11 (Embedded DSL)

```
select(person.name, person.age)
    .from(person)
    .where(person.age >= 18)
```

Thank You!
Questions?