Declarative Semantics Definition

Code Generation

Guido Wachsmuth

Assessment

last lecture

Assessment

last lecture

Discuss the need for static analysis in the definition of software languages.

Assessment
last lecture

Discuss the need for static analysis in the definition of software languages.

- decidability & efficiency of the word problem
- context-free languages
- software languages typically not context-free
- liberal context-free grammar for superset of the language
- static analysis to restrict superset
- semantic editor services

Assessment

last lecture

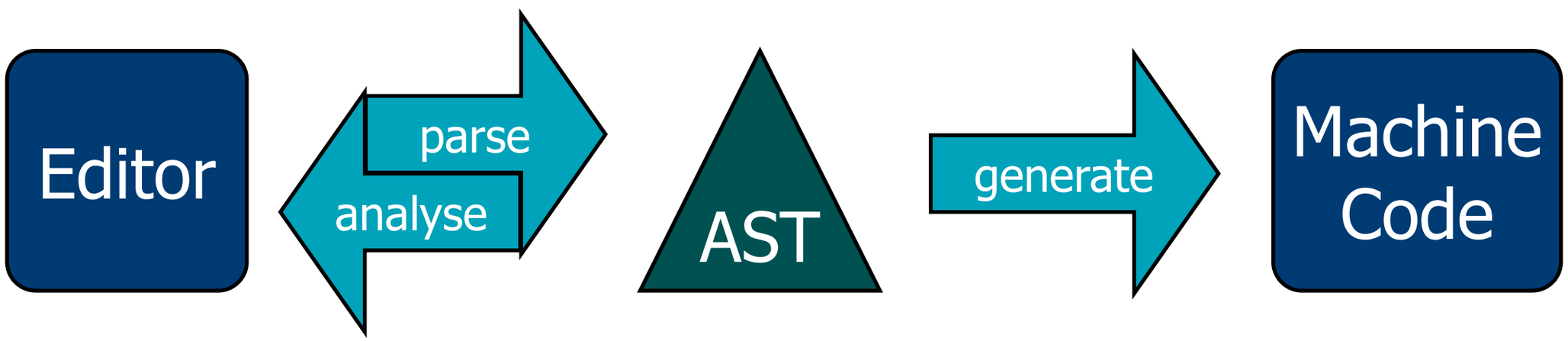
Discuss the need for static analysis in the definition of software languages.

- decidability & efficiency of the word problem
- context-free languages
- software languages typically not context-free
- liberal context-free grammar for superset of the language
- static analysis to restrict superset
- semantic editor services

How does static analysis contribute to a compiler w.r.t. its architecture?

Recap: Modern Compilers in IDEs

architecture



Overview
today's lecture

Java Virtual Machine

- name analysis
- Java Bytecode instructions
- execution

code generation

- text vs. code generation
- Tiger to Java Bytecode

pretty-printing

- string interpolation
- generic pretty-printing

I

Java Virtual Machine

Java Virtual Machine

bytecode instructions

method area		
pc: 00		
00	A7	goto
01	00	
02	04	04
03	00	nop
04	A7	goto
05	FF	
06	FF	03

Java Virtual Machine

bytecode instructions

method area		
pc: 04		
00	A7	goto
01	00	
02	04	04
03	00	nop
04	A7	goto
05	FF	
06	FF	03

Java Virtual Machine

bytecode instructions

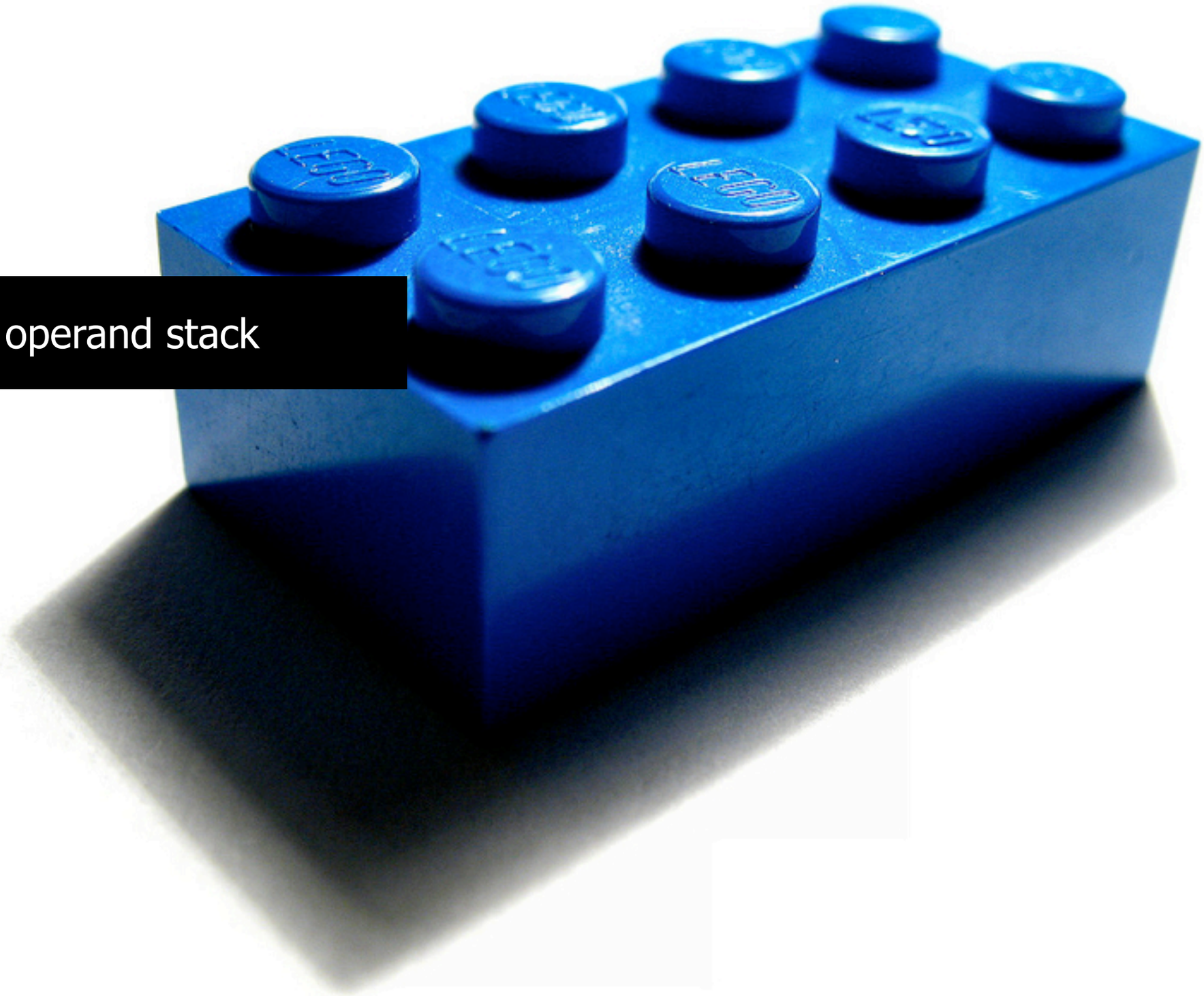
method area		
pc: 03		
00	A7	goto
01	00	
02	04	04
03	00	nop
04	A7	goto
05	FF	
06	FF	03

Java Virtual Machine

bytecode instructions

method area		
pc: 04		
00	A7	goto
01	00	
02	04	04
03	00	nop
04	A7	goto
05	FF	
06	FF	03

the operand stack



Java Virtual Machine

the operand stack

method area		
pc: 00		
00	04	iconst_1
01	05	iconst_2
02	10	bipush
03	2A	
04	11	sipush
05	43	
06	03	

stack
optop: 00
00
01
02
03
04
05
06

Java Virtual Machine

the operand stack

method area	
pc: 01	
00	04 iconst_1
01	05 iconst_2
02	10 bipush
03	2A
04	11 sipush
05	43
06	03

stack
optop: 01
00 0000 0001
01
02
03
04
05
06

Java Virtual Machine

the operand stack

method area		
pc: 02		
00	04	iconst_1
01	05	iconst_2
02	10	bipush
03	2A	
04	11	sipush
05	43	
06	03	

stack	
optop: 02	
00	0000 0001
01	0000 0002
02	
03	
04	
05	
06	

Java Virtual Machine

the operand stack

method area		
pc: 04		
00	04	iconst_1
01	05	iconst_2
02	10	bipush
03	2A	
04	11	sipush
05	43	
06	03	

stack
optop: 03
00 0000 0001
01 0000 0002
02 0000 002A
03
04
05
06

Java Virtual Machine

the operand stack

method area	
pc: 07	
00	04 iconst_1
01	05 iconst_2
02	10 bipush
03	2A
04	11 sipush
05	43
06	03

stack	
optop: 04	
00	0000 0001
01	0000 0002
02	0000 002A
03	0000 4303
04	
05	
06	

Java Virtual Machine

the operand stack

method area	
pc: 07	
07	60 iadd
08	68 imul
09	5F swap
0A	64 isub
0B	9A ifne
0C	FF
0D	F5 00

stack
optop: 04
00 0000 0001
01 0000 0002
02 0000 002A
03 0000 4303
04
05
06

Java Virtual Machine

the operand stack

method area	
pc: 08	
07	60 iadd
08	68 imul
09	5F swap
0A	64 isub
0B	9A ifne
0C	FF
0D	F5 00

stack
optop: 03
00 0000 0001
01 0000 0002
02 0000 432D
03
04
05
06

Java Virtual Machine

the operand stack

method area	
pc: 09	
07	60 iadd
08	68 imul
09	5F swap
0A	64 isub
0B	9A ifne
0C	FF
0D	F5 00

stack
optop: 02
00 0000 0001
01 0000 865A
02
03
04
05
06

Java Virtual Machine

the operand stack

method area		
pc: 0A		
07	60	iadd
08	68	imul
09	5F	swap
0A	64	isub
0B	9A	ifne
0C	FF	
0D	F5	00

stack
optop: 02
00 0000 865A
01 0000 0001
02
03
04
05
06

Java Virtual Machine

the operand stack

method area	
pc: 0B	
07	60 iadd
08	68 imul
09	5F swap
0A	64 isub
0B	9A ifne
0C	FF
0D	F5 00

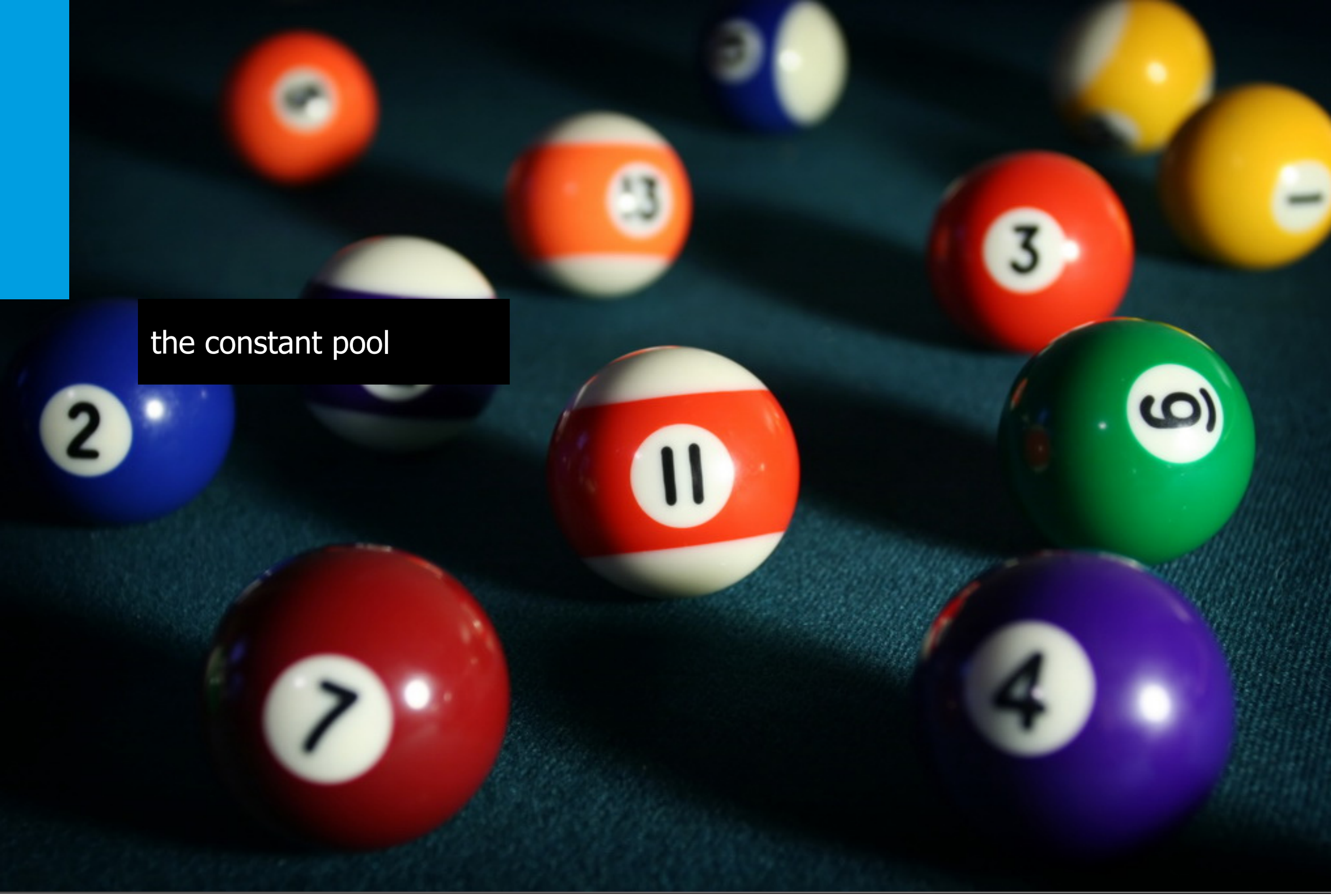
stack
optop: 01
00 0000 8659
01
02
03
04
05
06

Java Virtual Machine

the operand stack

method area		
pc: 00		
00	04	iconst_1
01	05	iconst_2
02	10	bipush
03	2A	
04	11	sipush
05	43	
06	03	

stack
optop: 00
00
01
02
03
04
05
06



the constant pool

Java Virtual Machine

the constant pool

method area	
pc: 00	constant pool
00 12 ldc	00 0000 002A
01 00 00	01 0000 4303
02 12 ldc	02 0000 0000
03 01 01	03 0000 002A
04 14 ldc2_w	04
05 00	05
06 02 02	06

stack
optop: 00
00
01
02
03
04
05
06

Java Virtual Machine

the constant pool

method area	
pc: 02	constant pool
00 12 ldc	00 0000 002A
01 00 00	01 0000 4303
02 12 ldc	02 0000 0000
03 01 01	03 0000 002A
04 14 ldc2_w	04
05 00	05
06 02 02	06

stack
optop: 01
00 0000 002A
01
02
03
04
05
06

Java Virtual Machine

the constant pool

method area	
pc: 04	constant pool
00 12 ldc	00 0000 002A
01 00 00	01 0000 4303
02 12 ldc	02 0000 0000
03 01 01	03 0000 002A
04 14 ldc2_w	04
05 00	05
06 02 02	06

stack
optop: 02
00 0000 002A
01 0000 4303
02
03
04
05
06

Java Virtual Machine

the constant pool

method area	
pc: 07	constant pool
00 12 ldc	00 0000 002A
01 00 00	01 0000 4303
02 12 ldc	02 0000 0000
03 01 01	03 0000 002A
04 14 ldc2_w	04
05 00	05
06 02 02	06

stack
optop: 04
00 0000 002A
01 0000 4303
02 0000 0000
03 0000 002A
04
05
06

local variables

Java Virtual Machine

local variables

method area		
pc: 00		
00	04	iconst_1
01	3B	istore_0
02	1A	iload_0
03	3C	istore_1
04	84	iinc
05	01	01
06	01	01

stack	
optop: 00	local variables
00	00
01	01
02	02
03	03
04	04
05	05
06	06

Java Virtual Machine

local variables

method area		
pc: 01		
00	04	iconst_1
01	3B	istore_0
02	1A	iload_0
03	3C	istore_1
04	84	iinc
05	01	01
06	01	01

stack	
optop: 01	local variables
00 0000 0001	00
01	01
02	02
03	03
04	04
05	05
06	06

Java Virtual Machine

local variables

method area		
pc: 02		
00	04	iconst_1
01	3B	istore_0
02	1A	iload_0
03	3C	istore_1
04	84	iinc
05	01	01
06	01	01

stack	
optop: 00	local variables
00	00 0000 0001
01	01
02	02
03	03
04	04
05	05
06	06

Java Virtual Machine

local variables

method area		
pc: 03		
00	04	iconst_1
01	3B	istore_0
02	1A	iload_0
03	3C	istore_1
04	84	iinc
05	01	01
06	01	01

stack	
optop: 01	local variables
00 0000 0001	00 0000 0001
01	01
02	02
03	03
04	04
05	05
06	06

Java Virtual Machine

local variables

method area		
pc: 04		
00	04	iconst_1
01	3B	istore_0
02	1A	iload_0
03	3C	istore_1
04	84	iinc
05	01	01
06	01	01

stack	
optop: 00	local variables
00	00 0000 0001
01	01 0000 0001
02	02
03	03
04	04
05	05
06	06

Java Virtual Machine

local variables

method area		
pc: 07		
00	04	iconst_1
01	3B	istore_0
02	1A	iload_0
03	3C	istore_1
04	84	iinc
05	01	01
06	01	01

stack	
optop: 00	local variables
00	00 0000 0001
01	01 0000 0002
02	02
03	03
04	04
05	05
06	06


the heap

Java Virtual Machine

the heap

method area	
pc: 00	constant pool
00 12 ldc	00 4303 4303
01 00 00	01 0000 0004
02 19 aload	02
03 00 00	03
04 12 ldc	04
05 01 01	05
06 2E iaload	06

stack	
optop: 00	local variables
00	00 002A 002A
01	01
02	02
03	03
04	04
05	05
06	06

heap	
4303 4303 "Compilers"	002A 002A [20,01,40,02,42]

Java Virtual Machine

the heap

method area	
pc: 02	constant pool
00 12 ldc	00 4303 4303
01 00 00	01 0000 0004
02 19 aload	02
03 00 00	03
04 12 ldc	04
05 01 01	05
06 2E iaload	06

stack	
optop: 01	local variables
00 4303 4303	00 002A 002A
01	01
02	02
03	03
04	04
05	05
06	06

heap	
4303 4303 "Compilers"	002A 002A [20,01,40,02,42]

Java Virtual Machine

the heap

method area

pc: 04	constant pool
00 12 ldc	00 4303 4303
01 00 00	01 0000 0004
02 19 aload	02
03 00 00	03
04 12 ldc	04
05 01 01	05
06 2E iaload	06

stack

optop: 02	local variables
00 4303 4303	00 002A 002A
01 002A 002A	01
02	02
03	03
04	04
05	05
06	06

heap

4303 4303 "Compilers" 002A 002A [20,01,40,02,42]

Java Virtual Machine

the heap

method area

pc: 06	constant pool
00 12 ldc	00 4303 4303
01 00 00	01 0000 0004
02 19 aload	02
03 00 00	03
04 12 ldc	04
05 01 01	05
06 2E iaload	06

stack

optop: 03	local variables
00 4303 4303	00 002A 002A
01 002A 002A	01
02 0000 0004	02
03	03
04	04
05	05
06	06

heap

4303 4303 "Compilers" 002A 002A [20,01,40,02,42]

Java Virtual Machine

the heap

method area

pc: 07	constant pool
00 12 ldc	00 4303 4303
01 00 00	01 0000 0004
02 19 aload	02
03 00 00	03
04 12 ldc	04
05 01 01	05
06 2E iaload	06

stack

optop: 02	local variables
00 4303 4303	00 002A 002A
01 0000 0042	01
02	02
03	03
04	04
05	05
06	06

heap

4303 4303 "Compilers" 002A 002A [20,01,40,02,42]

stack frames

Java Virtual Machine

stack frames

method area

pc: 03

```
00    2A      aload_0
01    10      bipush
02    40
03    B6      invokevirtual
04    00
05    01      01
06    AC      ireturn
```

stack

optop: 02

```
00 4303 4303
01 0000 0040
02
03
04
05
06
```

local variables

```
00 4303 4303
01
02
03
04
05
06
```

heap

Java Virtual Machine

stack frames

method area

pc: 80

80	2B	iload_1
81	59	dup
82	68	imul
83	AC	ireturn
84	00	
85	00	
86	00	

stack

optop: 00

	local variables
00	00 4303 4303
01	01 0000 0040
02	02
03	03
04	04
05	05
06	06

heap

Java Virtual Machine

stack frames

method area

```
pc: 81

80    2B      iload_1
81    59      dup
82    68      imul
83    AC      ireturn
84    00
85    00
86    00
```

stack

```
optop: 01

00 0000 0040
01
02
03
04
05
06
```

local variables

```
00 4303 4303
01 0000 0040
02
03
04
05
06
```

heap

Java Virtual Machine

stack frames

method area

pc: 81

```
80    2B      iload_1
81    59      dup
82    68      imul
83    AC      ireturn
84    00
85    00
86    00
```

stack

optop: 02

local variables
00 4303 4303
01 0000 0040
02
03
04
05
06

00 0000 0040
01 0000 0040
02
03
04
05
06

heap

Java Virtual Machine

stack frames

method area

```
pc: 82

80    2B      iload_1
81    59      dup
82    68      imul
83    AC      ireturn
84    00
85    00
86    00
```

stack

```
optop: 01
```

local variables
00 4303 4303
01 0000 0040
02
03
04
05
06

00 0000 1000
01
02
03
04
05
06

heap

Java Virtual Machine

stack frames

method area

pc: 06

00	2A	aload_0
01	10	bipush
02	40	
03	B6	invokevirtual
04	00	
05	01	01
06	AC	ireturn

stack

optop: 01

00 0000 1000	00 4303 4303
01	01
02	02
03	03
04	04
05	05
06	06

local variables

heap

II

class files

Recap: Traditional Java Compiler

example

Recap: Traditional Java Compiler
example

ls

Course.java

Recap: Traditional Java Compiler
example

ls

Course.java

javac -verbose Course.java

[parsing started Course.java]
[parsing completed 8ms]
[loading java/lang/Object.class(java/lang:Object.class)]
[checking university.Course]
[wrote Course.class]
[total 411ms]

Recap: Traditional Java Compiler example

ls

Course.java

javac -verbose Course.java

[parsing started Course.java]
[parsing completed 8ms]
[loading java/lang/Object.class(java/lang:Object.class)]
[checking university.Course]
[wrote Course.class]
[total 411ms]

ls

Course.class Course.java

Class Files

format

magic number `CAFEBABE`

class file version (minor, major)

constant pool count + constant pool

access flags

this class

super class

interfaces count + interfaces

fields count + fields

methods count + methods

attribute count + attributes

Intermediate Language

```
.class public Exp

   .method public static fac(I)I

            iload 1
            ifne else

            iconst_1
            ireturn

      else: iload 1
            dup
            iconst_1
            isub
            invokestatic Exp/fac(I)I
            imul
            ireturn
   .end method
```

III

code generation

Printing Code

```
to-jbc = ?Nil() ; <printstring> "aconst_null\n"
to-jbc = ?NoVal() ; <printstring> "nop\n"
to-jbc = ?Seq(es) ; <list-loop(to-jbc)> es

to-jbc =
   ?Int(i);
   <printstring> "ldc ";
   <printstring> i;
   <printstring> "\n"


to-jbc = ?Bop(op, e1, e2) ; <to-jbc> e1 ; <to-jbc> e2 ; <to-jbc> op

to-jbc = ?PLUS() ; <printstring> "iadd\n"
to-jbc = ?MINUS() ; <printstring> "isub\n"
to-jbc = ?MUL() ; <printstring> "imul\n"
to-jbc = ?DIV() ; <printstring> "idiv\n"
```

Composing Strings

```
to-jbc: Nil()   -> "aconst_null\n"
to-jbc: NoVal() -> "nop\n"
to-jbc: Seq(es) -> <concat-strings> <map(to-jbc)> es

to-jbc: Int(i) -> <concat-strings> ["ldc ", i, "\n"]

to-jbc: Bop(op, e1, e2) -> <concat-strings> [ <to-jbc> e1,
                                              <to-jbc> e2,
                                              <to-jbc> op ]


to-jbc: PLUS()  -> "iadd\n"
to-jbc: MINUS() -> "isub\n"
to-jbc: MUL()   -> "imul\n"
to-jbc: DIV()   -> "idiv\n"
```

String Interpolation

```
to-jbc: Nil()   -> $[aconst_null]
to-jbc: NoVal() -> $[nop]
to-jbc: Seq(es) -> <map-to-jbc> es

map-to-jbc: [] -> $[]
map-to-jbc: [h|t] ->
   $[[<to-jbc> h]
     [<map-to-jbc> t]]

to-jbc: Int(i) -> $[ldc [i]]

to-jbc: Bop(op, e1, e2) ->
   $[[<to-jbc> e1]
     [<to-jbc> e2]
     [<to-jbc> op]]

to-jbc: PLUS()  -> $[iadd]
to-jbc: MINUS() -> $[isub]
to-jbc: MUL()   -> $[imul]
to-jbc: DIV()   -> $[idiv]
```

coffee break

IV

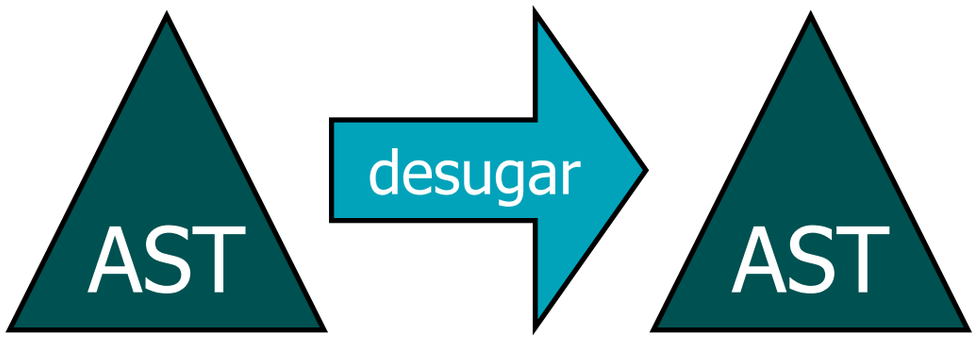
generation by transformation

Staged Compilation

transformation steps

Staged Compilation

transformation steps

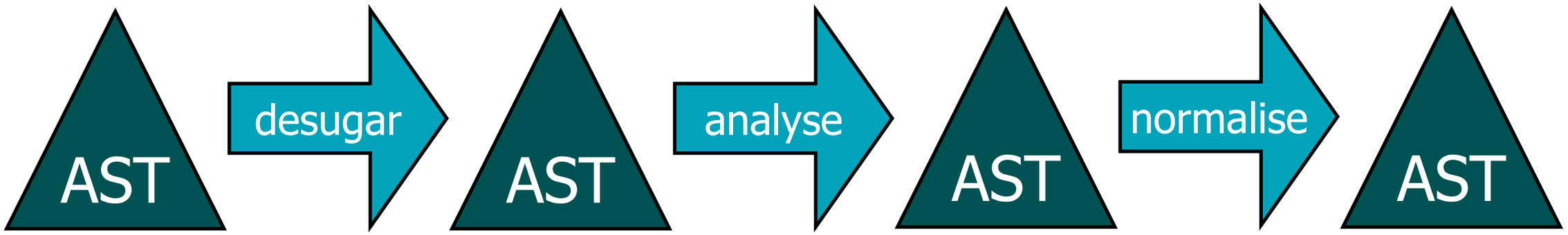


Staged Compilation

transformation steps

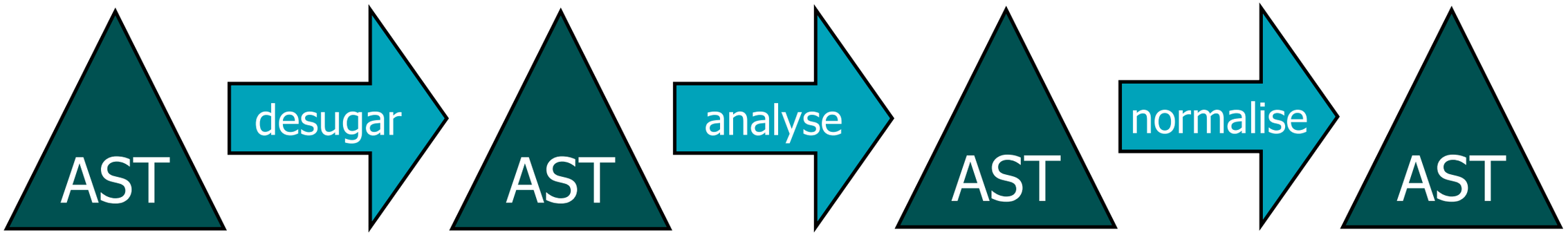
Staged Compilation

transformation steps



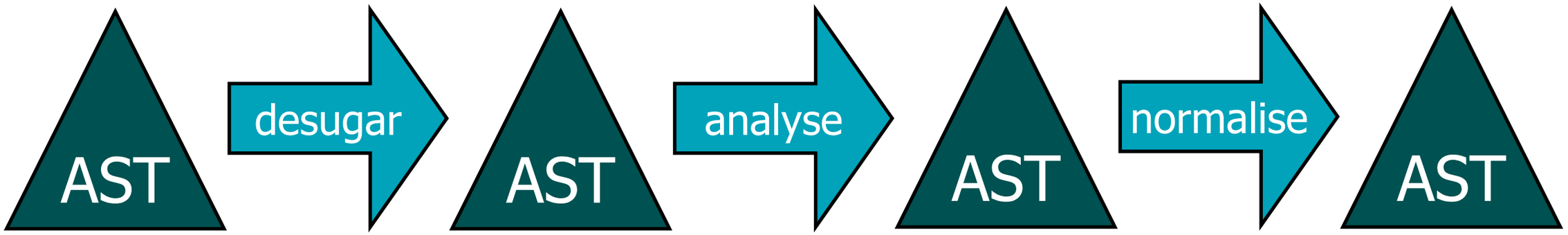
Staged Compilation

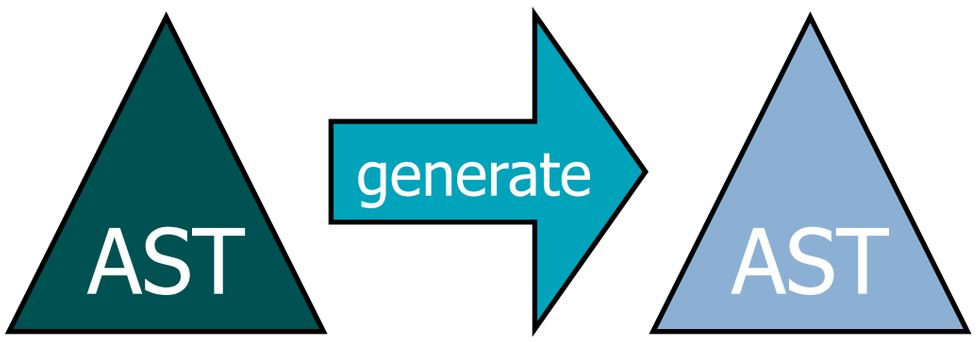
transformation steps



Staged Compilation

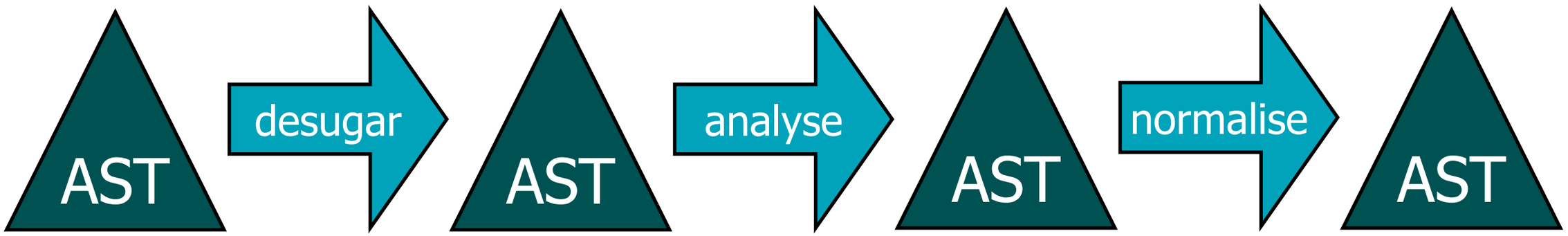
transformation steps

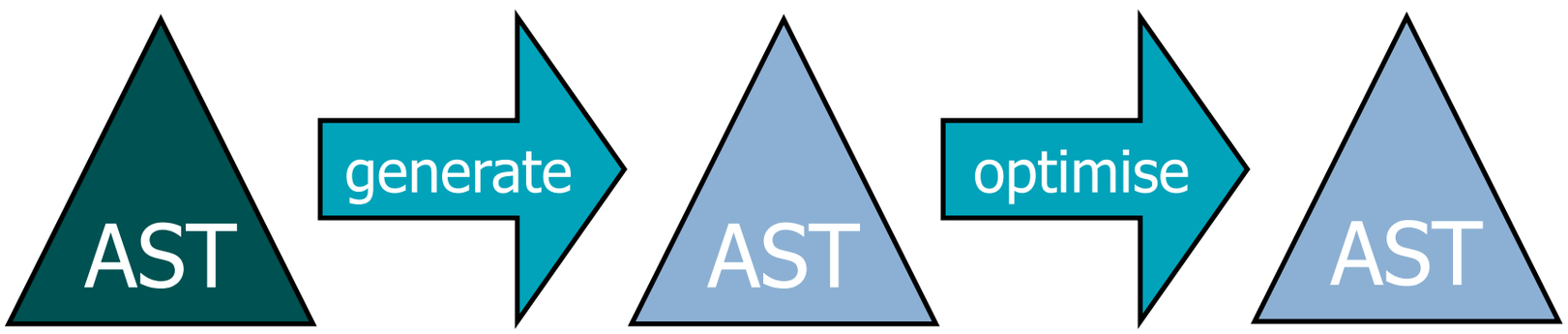




Staged Compilation

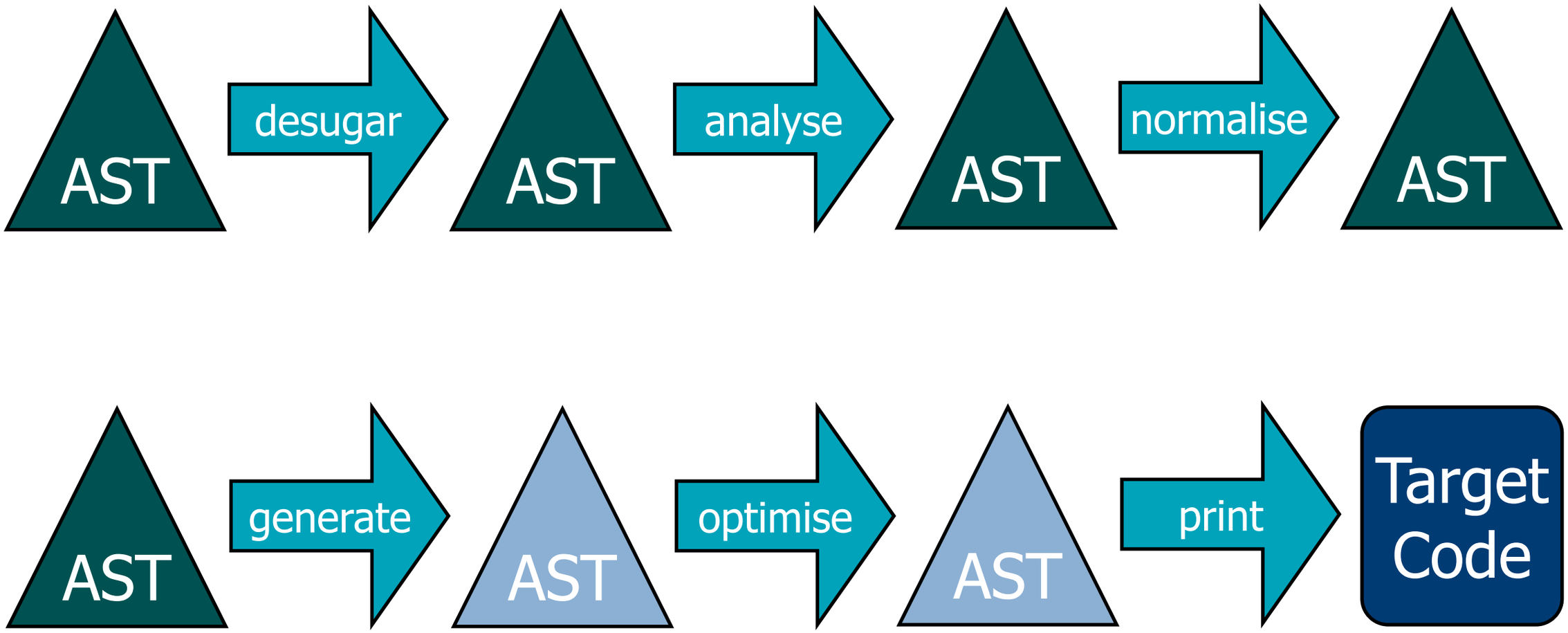
transformation steps





Staged Compilation

transformation steps



Pretty-Printing

```
pp-jbc: ACONST_NULL() -> $[aconst_null]

pp-jbc: NOP()  -> $[nop]

pp-jbc: LDC(Int(i))    -> $[ldc [i]]
pp-jbc: LDC(String(s)) -> $[ldc [s]]

pp-jbc: IADD() -> $[iadd]
pp-jbc: ISUB() -> $[isub]
pp-jbc: IMUL() -> $[imul]
pp-jbc: IDIV() -> $[idiv]
```

Recap: Desugaring

```
if x then
   printint(x)
```

```
if x then
   printint(x)
else
   ()
```

```
IfThen(
  Var("x")
, Call(
    "printint"
  , [Var("x")]
  )
)
```

```
IfThenElse(
  Var("x")
, Call(
    "printint"
  , [Var("x")]
  )
, NoVal()
)
```

```
desugar: IfThen(e1, e2) -> IfThenElse(e1, e2, NoVal())
```

Recap: Desugaring

```
desugar: Uminus(e)      -> Bop(MINUS(), Int("0"), e)


desugar: Plus(e1, e2)   -> Bop(PLUS(), e1, e2)
desugar: Minus(e1, e2)  -> Bop(MINUS(), e1, e2)
desugar: Times(e1, e2)  -> Bop(MUL(), e1, e2)
desugar: Divide(e1, e2) -> Bop(DIV(), e1, e2)


desugar: Eq(e1, e2)  -> Rop(EQ(), e1, e2)
desugar: Neq(e1, e2) -> Rop(NE(), e1, e2)
desugar: Leq(e1, e2) -> Rop(LE(), e1, e2)
desugar: Lt(e1, e2)  -> Rop(LT(), e1, e2)
desugar: Geq(e1, e2) -> Rop(LE(), e2, e1)
desugar: Gt(e1, e2)  -> Rop(LT(), e2, e1)


desugar: And(e1, e2) -> IfThenElse(e1, e2, Int("0"))
desugar: Or(e1, e2)  -> IfThenElse(e1, Int("1"), e2)
```

```
signature constructors
  PLUS:   BinOp
  MINUS:  BinOp
  MUL:    BinOp
  DIV:    BinOp

  EQ:  RelOp
  NE:  RelOp
  LE:  RelOp
  LT:  RelOp

  Bop: BinOp * Expr * Expr -> Expr
  Rop: RelOp * Expr * Expr -> Expr
```

Recap: Analysis

```
let
   type t = u
   type u = int
   var x: u := 0
in
   x := 42 ;
   let
      type u = t
      var y: u := 0
   in
      y := 42
   end
end
```

```
let
   type t0 = u0
   type u0 = int
   var x0: u0 := 0
in
   x0 := 42 ;
   let
      type u1 = t0
      var y0: u1 := 0
   in
      y0 := 42
   end
end
```

```
rename-all = alltd(rename)
```

Normalisation

```
normalise : VarDec(v, e)    -> VarDec(v, <type-of> e, e)
normalise : VarDec(v, t, e) -> VarDec(v, <type-of> t, e)

normalise : FunDec(f, as, e)    -> FunDec(f, as, NO_VAL(), e)
normalise : FunDec(f, as, t, e) -> FunDec(f, as, <type-of> t, e)

normalise :
    For(v, e1, e2, e3) -> Let([decl], [while])
    where
        !VarDec(v, INT(), e1) => decl ;
        !Rop(LE(), Var(v), e2) => check ;
        !Assign(Var(v), Bop(PLUS(), Var(v), Int("1"))) => inc ;
        !While(check, Seq([e3, inc])) => while

normalise : Seq([])  -> NoVal()
normalise : Seq([e]) -> e

normalise : Seq([NoVal()|es])   -> Seq(es)
normalise : Seq([e1,e2,e3|es])  -> Seq([e1, Seq([e2, e3|es])])
normalise : Seq([Seq(es1)|es2]) -> Seq(<conc> (es1, es2))
```

Normalisation

```
let
   type t0 = u0
   type u0 = int
   var x0: u0 := 21
in
   x0 := 42 ;
   let
      function fac0(n0: int): int=
         if n0 = 0 then
            1
         else
            n0 * fac0(n0 - 1)

      type u1 = t0
      var y0: u1 := fac0(x0)
   in
      y0 := 42
   end
end
```

```
let
   var x0: int := 0
   var y0: int := 0

   function fac0(n0: int): int=
      if n0 = 0 then
         1
      else
         n0 * fac0(n0 - 1)
in
   x0 := 21 ;
   x0 := 42 ;
   y0 := fac0(x0) ;
   y0 := 42
end
```

Code Generation

```
to-jbc: Nil()   -> [ ACONST_NULL() ]
to-jbc: NoVal() -> [ NOP() ]
to-jbc: Seq(es) -> <mapconcat(to-jbc)> es

to-jbc: Int(i)    -> [ LDC(Int(i)) ]
to-jbc: String(s) -> [ LDC(String(s)) ]

to-jbc: Bop(op, e1, e2) -> <mapconcat(to-jbc)> [ e1, e2, op ]

to-jbc: PLUS()  -> [ IADD() ]
to-jbc: MINUS() -> [ ISUB() ]
to-jbc: MUL()   -> [ IMUL() ]
to-jbc: DIV()   -> [ IDIV() ]

to-jbc: Assign(lhs, e) -> <concat> [ <to-jbc> e, <lhs-to-jbc> lhs ]

to-jbc:     Var(x) -> [ ILOAD(x) ]  where <type-of> Var(x) => INT()
to-jbc:     Var(x) -> [ ALOAD(x) ]  where <type-of> Var(x) => STRING()
lhs-to-jbc: Var(x) -> [ ISTORE(x) ] where <type-of> Var(x) => INT()
lhs-to-jbc: Var(x) -> [ ASTORE(x) ] where <type-of> Var(x) => STRING()
```

Code Generation

```
to-jbc:
   IfThenElse(e1, e2, e3) -> <concat> [ <to-jbc> e1,
                                        [ IFEQ(LabelRef(else)) ],
                                        <to-jbc> e2,
                                        [ GOTO(LabelRef(end)),
                                          Label(else) ],
                                        <to-jbc> e3,
                                        [ Label(end) ] ]
   where <newname> "else" => else
   where <newname> "end"  => end

to-jbc:
   While(e1, e2) -> <concat> [ [ GOTO(LabelRef(check)),
                                 Label(body) ],
                               <to-jbc> e2,
                               [ Label(check) ],
                               <to-jbc> e1,
                               [ IFNE(LabelRef(body)) ] ]
   where <newname> "test" => check
   where <newname> "body" => body
```

Code Generation

```
function fac0(n0: int): int=
   if
      n0 = 0
   then
      1
   else
      n0 * fac0(n0 - 1)
```

```
.method public static fac0(I)I

            iload 1
            ldc 0
            if_icmpeq label0
            ldc 0
            goto label1
   label0:  ldc 1
   label1:  ifeq else0
            ldc 1
            goto end0
   else0:   iload 1
            iload 1
            ldc 1
            isub
            invokestatic
               Exp/fac0(I)I
            imul
   end0:    ireturn
.end method
```

Optimisation

```
.method public static fac0(I)I

          iload 1
          ldc 0
          if_icmpeq label0
          ldc 0
          goto label1
   label0: ldc 1
   label1: ifeq else0
          ldc 1
          goto end0
    else0:  iload 1
          iload 1
          ldc 1
          isub
          invokestatic
             Exp/fac0(I)I
          imul
    end0:   ireturn
.end method
```

```
.method public static fac0(I)I

          iload_1
          ifne else0

          iconst_1
          ireturn

    else0:  iload_1
          dup
          iconst_1
          isub
          invokestatic
             Exp/fac0(I)I
          imul
          ireturn
.end method
```

V

summary

Summary
lessons learned

Summary
lessons learned

What are Java Bytecode and the Java Virtual Machine?

- bytecode instructions
- architecture
- class file format
- intermediate language

Summary
lessons learned

What are Java Bytecode and the Java Virtual Machine?

- bytecode instructions
- architecture
- class file format
- intermediate language

How do you generate code by transformation?

- desugaring, decoration, normalisation
- transforming generated code

Summary
lessons learned

What are Java Bytecode and the Java Virtual Machine?

- bytecode instructions
- architecture
- class file format
- intermediate language

How do you generate code by transformation?

- desugaring, decoration, normalisation
- transforming generated code

How do you transform abstract syntax trees into text again?

- string interpolation
- pretty-printing

Literature

learn more

Literature

learn more

Java Virtual Machine

Tim Lindholm, Frank Yellin: The Java Virtual Machine Specification, 2nd edition. Addison-Wesley, 1999.

Bill Venners: Inside the Java 2 Virtual Machine. McGraw-Hill, 2000.

Literature

learn more

Java Virtual Machine

Tim Lindholm, Frank Yellin: The Java Virtual Machine Specification, 2nd edition. Addison-Wesley, 1999.

Bill Venners: Inside the Java 2 Virtual Machine. McGraw-Hill, 2000.

Generation by Transformation

Zef Hemel, Lennart C. L. Kats, Danny M. Groenewegen, Eelco Visser: Code generation by model transformation. A case study in transformation modularity. SoSym 9(3), 2010

Outlook
coming next

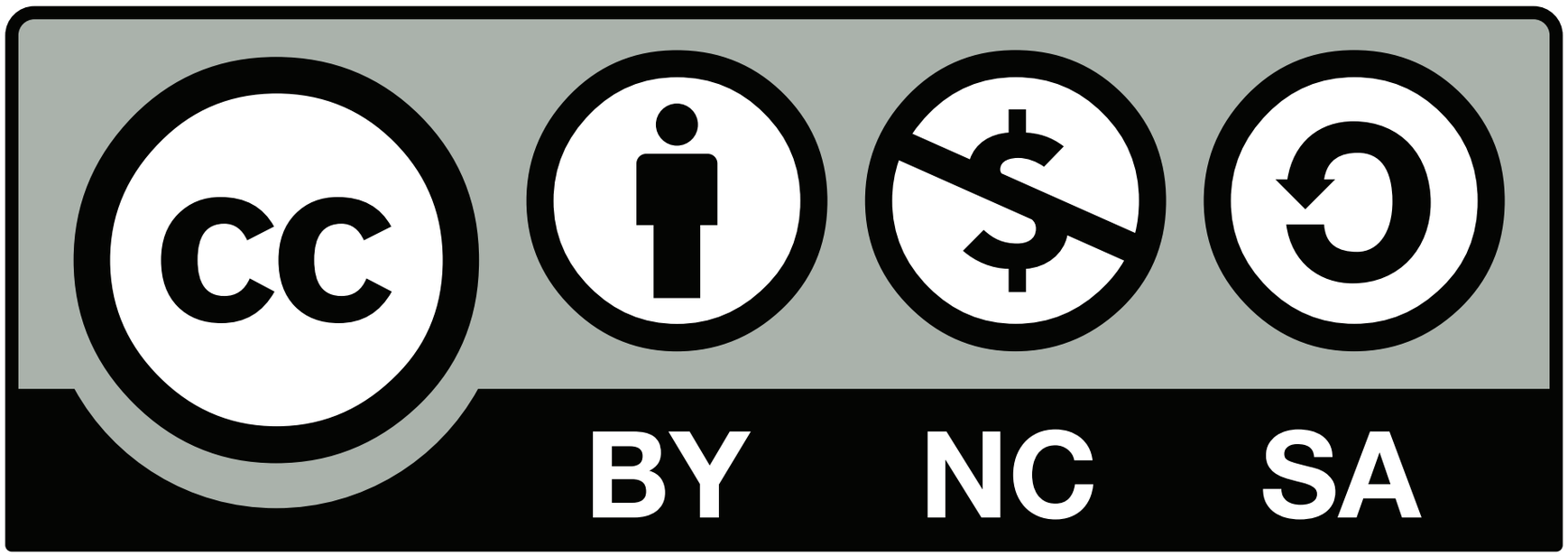
imperative and object-oriented languages

- Activation Frames
- Liveness Analysis
- Register Allocation
- Garbage Collection

Lab Oct 18

- type projection & type checking

copyrights & credits


BY
NC
SA

Pictures

copyrights

Slide 1:
Matrix by Gamaliel Espinoza Macedo, some rights reserved

Slide 7:
L is for Lego by Don Solo, some rights reserved

Slide 9:
Billiard Balls by Darren Hester, some rights reserved

Slide 11:
Autoturm by m.prinke, some rights reserved

Slide 13:
Spy Hill Landfill by D'Arcy Norman, some rights reserved

Slide 15:
Framed by LexnGer, some rights reserved

Slide 25:
Benders by Dominica Williamson, some rights reserved

Slide 41:
Leuchtturm by herr.g, some rights reserved