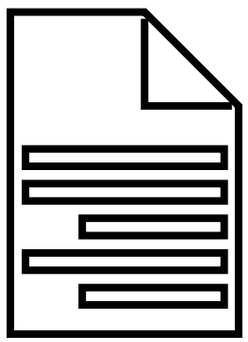
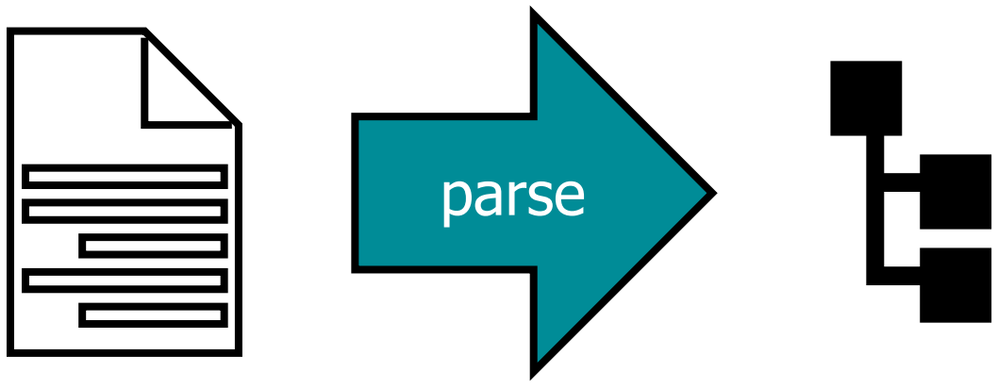
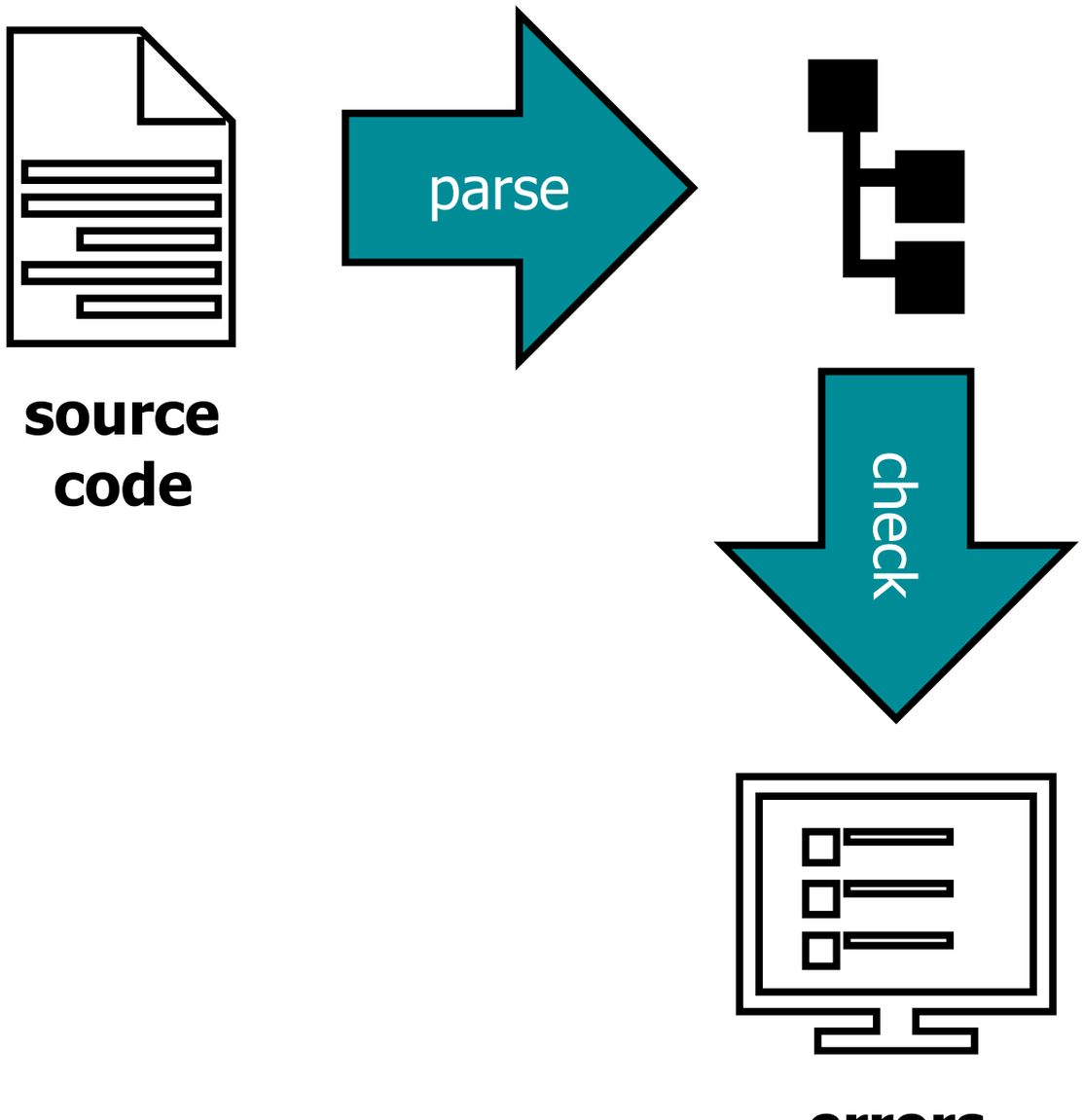
Virtual Machines

Guido Wachsmuth, Eelco Visser



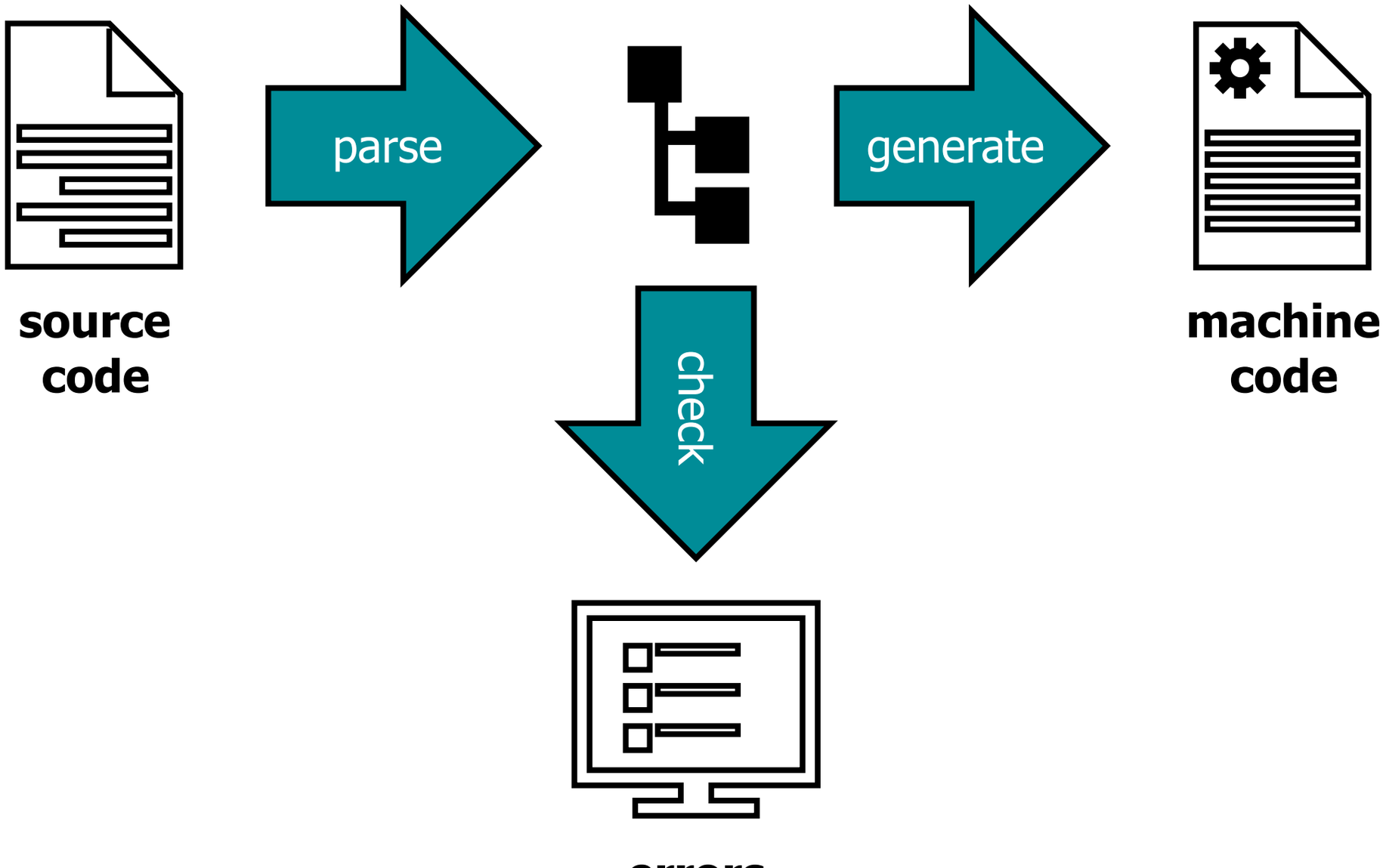
source code

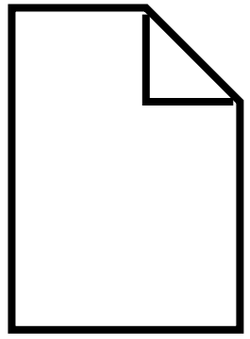

parse
source
code



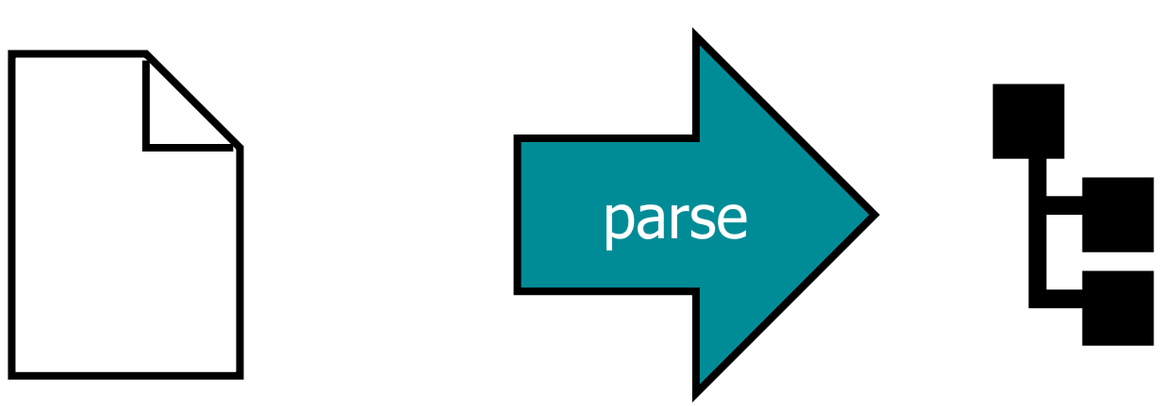
source code

errors

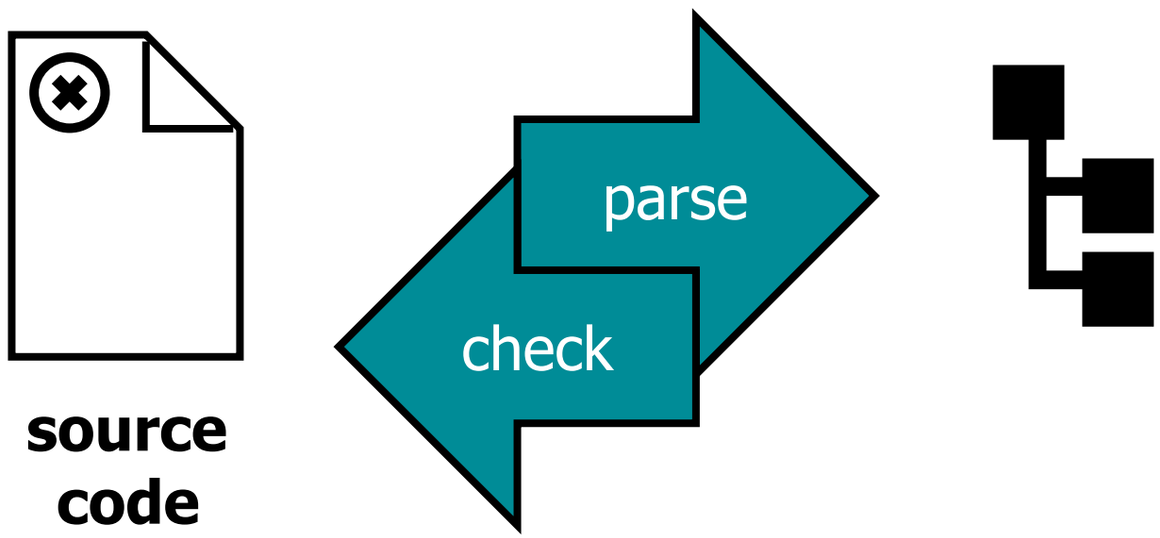

parse
generate
check
source code
machine code
errors

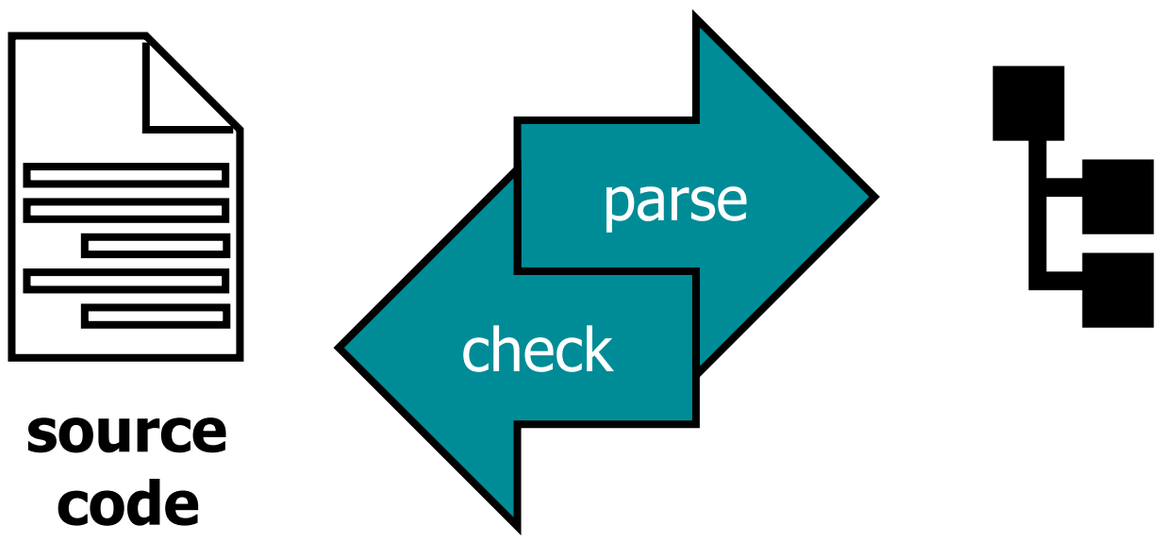


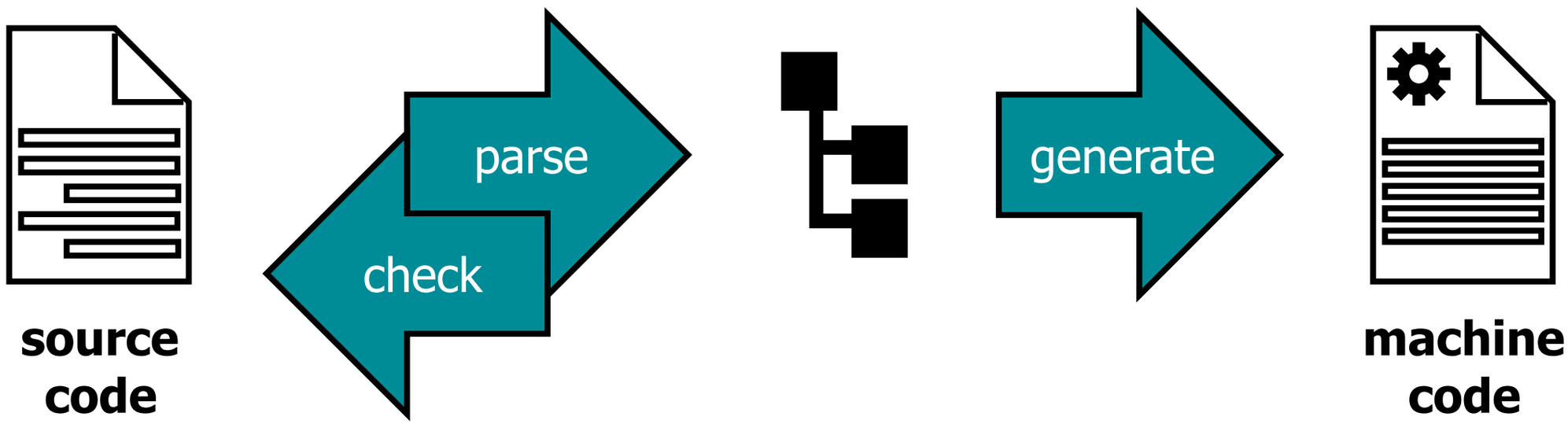
source code

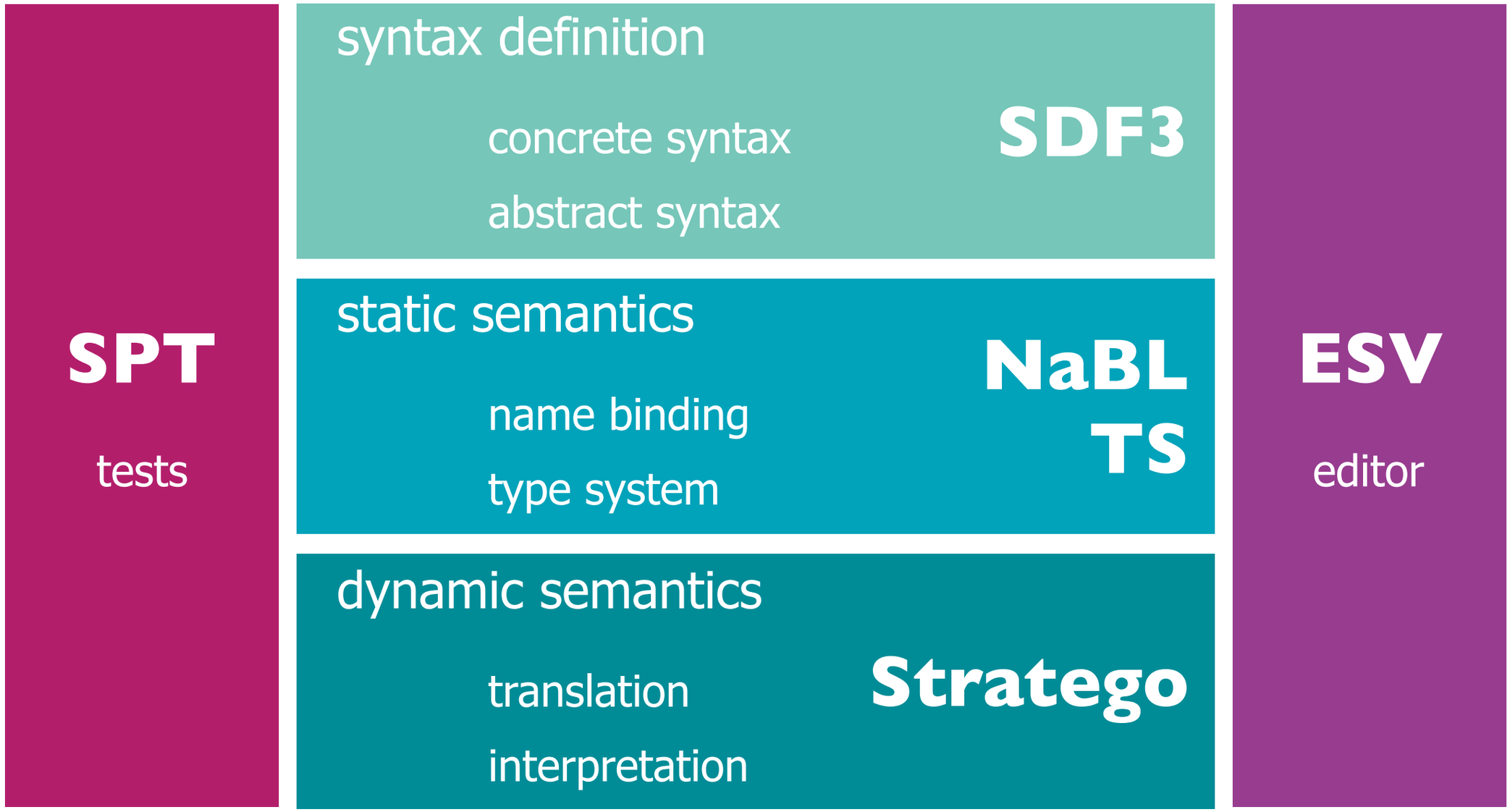


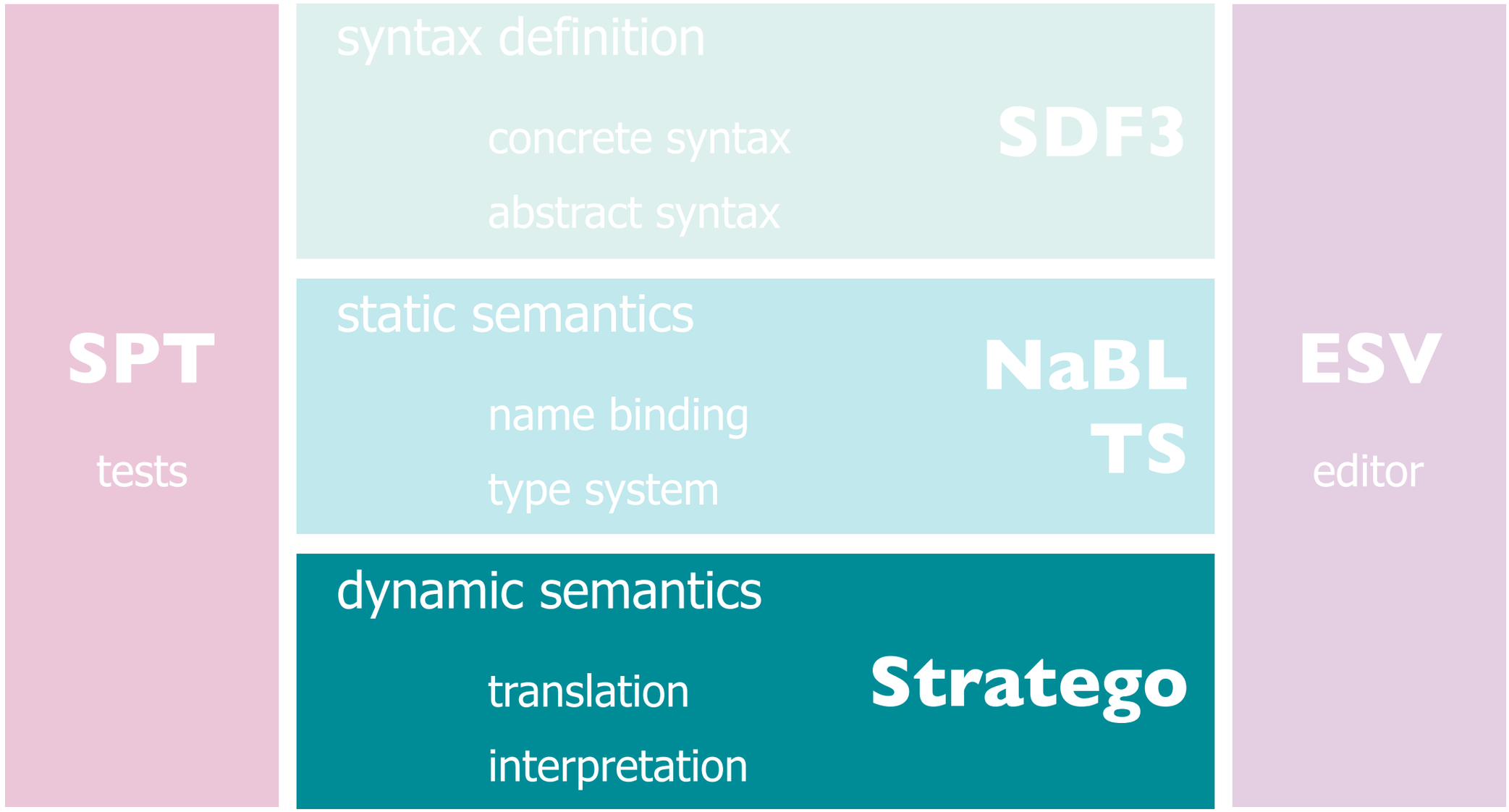
source code


parse
check
source
code


parse
check
source
code


parse
check
generate
source code
machine code


SPT
tests
syntax definition
concrete syntax
abstract syntax
SDF3
static semantics
name binding
type system
NaBL
TS
dynamic semantics
translation
interpretation
Stratego
ESV
editor


SPT
tests
syntax definition
concrete syntax
abstract syntax
SDF3
static semantics
name binding
type system
NaBL
TS
dynamic semantics
translation
interpretation
Stratego
ESV
editor

JVM

operand stack

constant pool

local variables

heap

stack frames

class files

code generation

printing strings

string concatenation

string interpolation

transformation

calling conventions

JVM

register-based machines

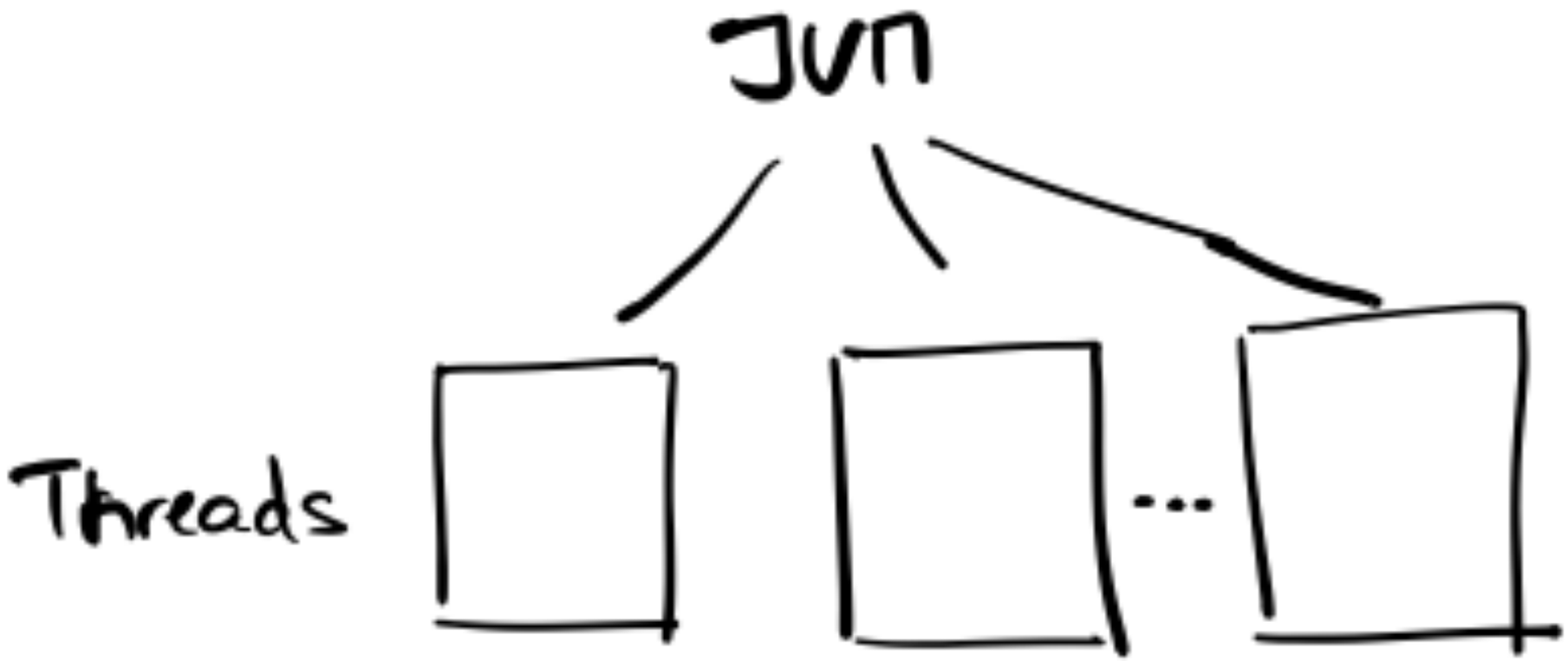
Java Virtual Machine

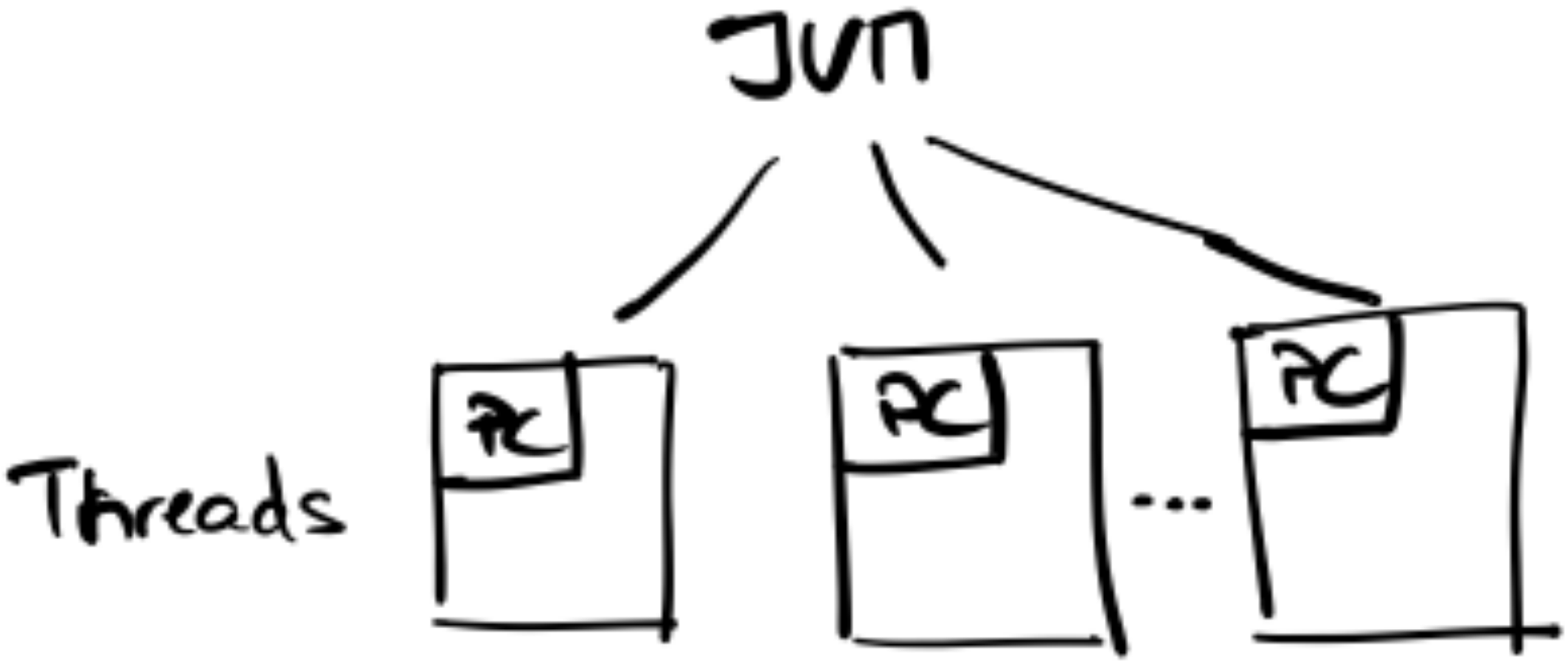
The Java® Virtual Machine Specification

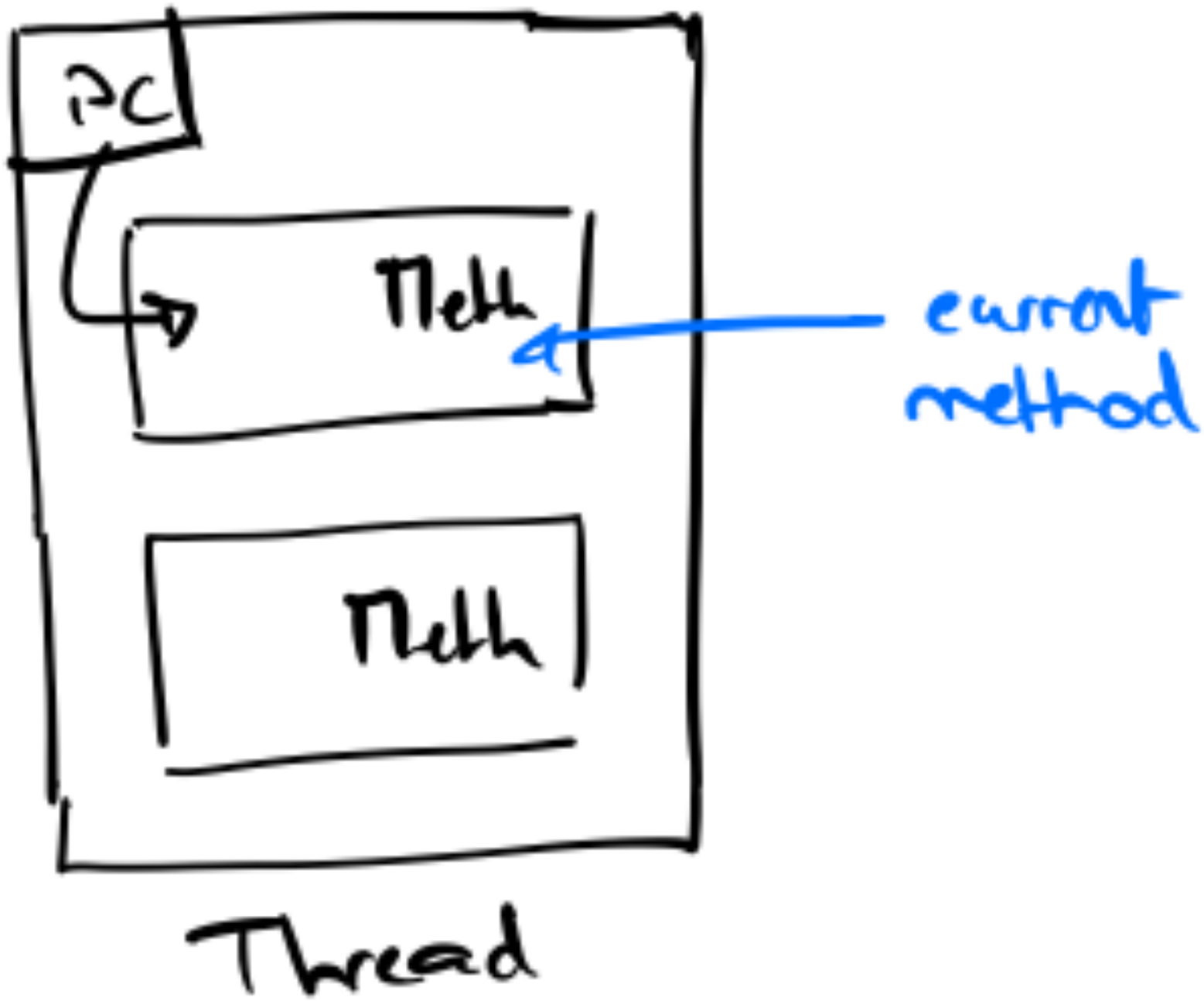
Java SE 8 Edition

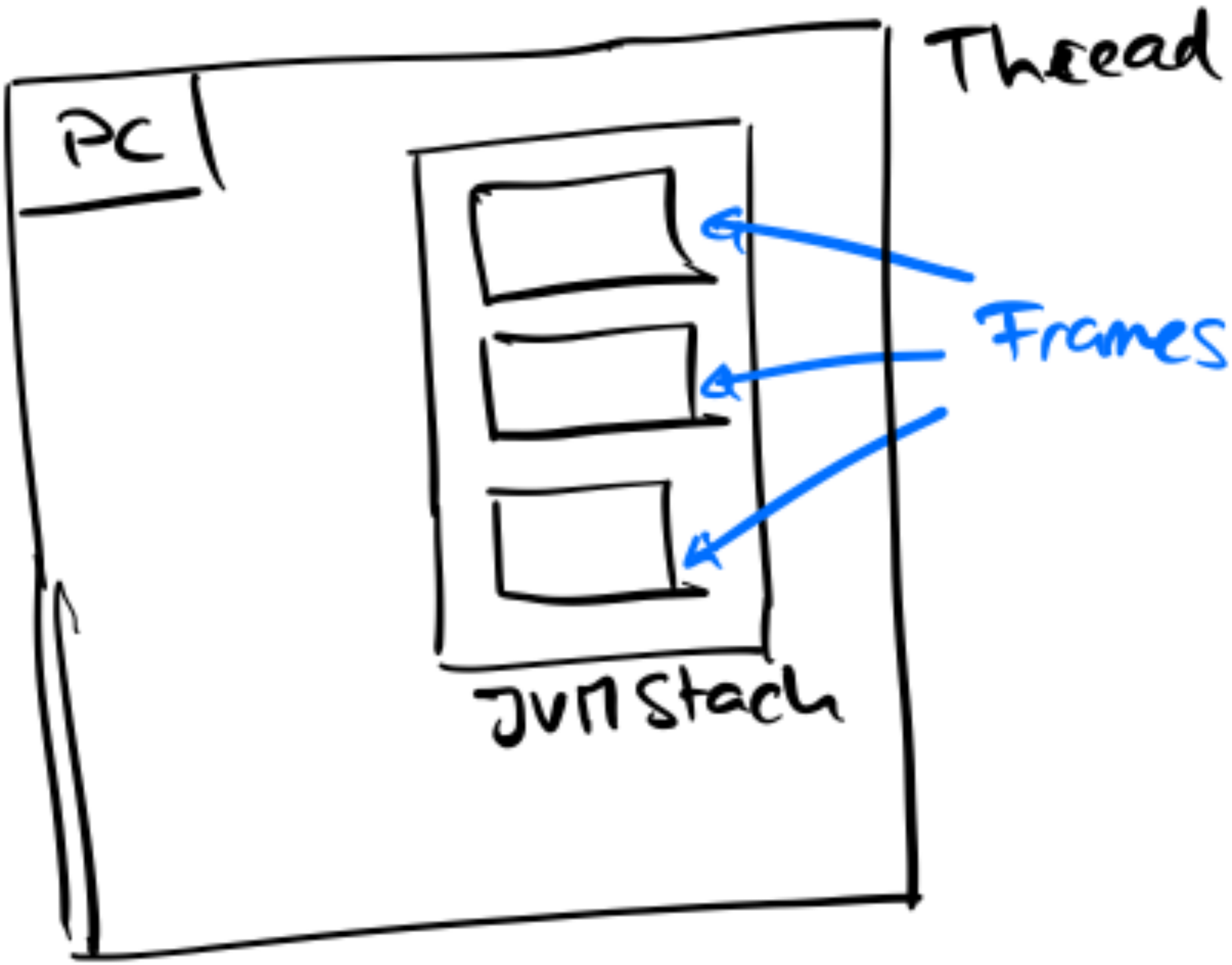
Tim Lindholm
Frank Yellin
Gilad Bracha
Alex Buckley

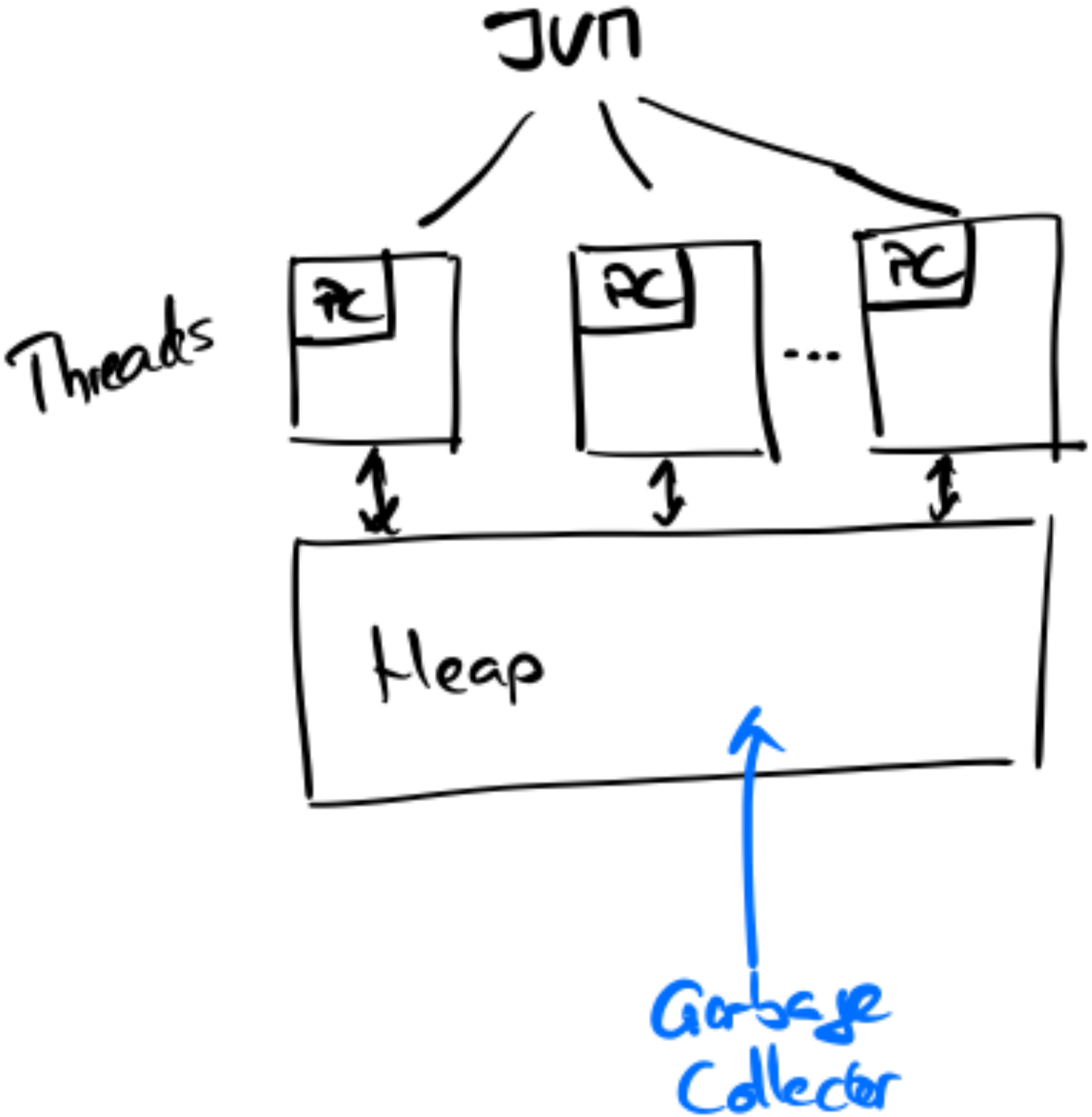
2015-02-13

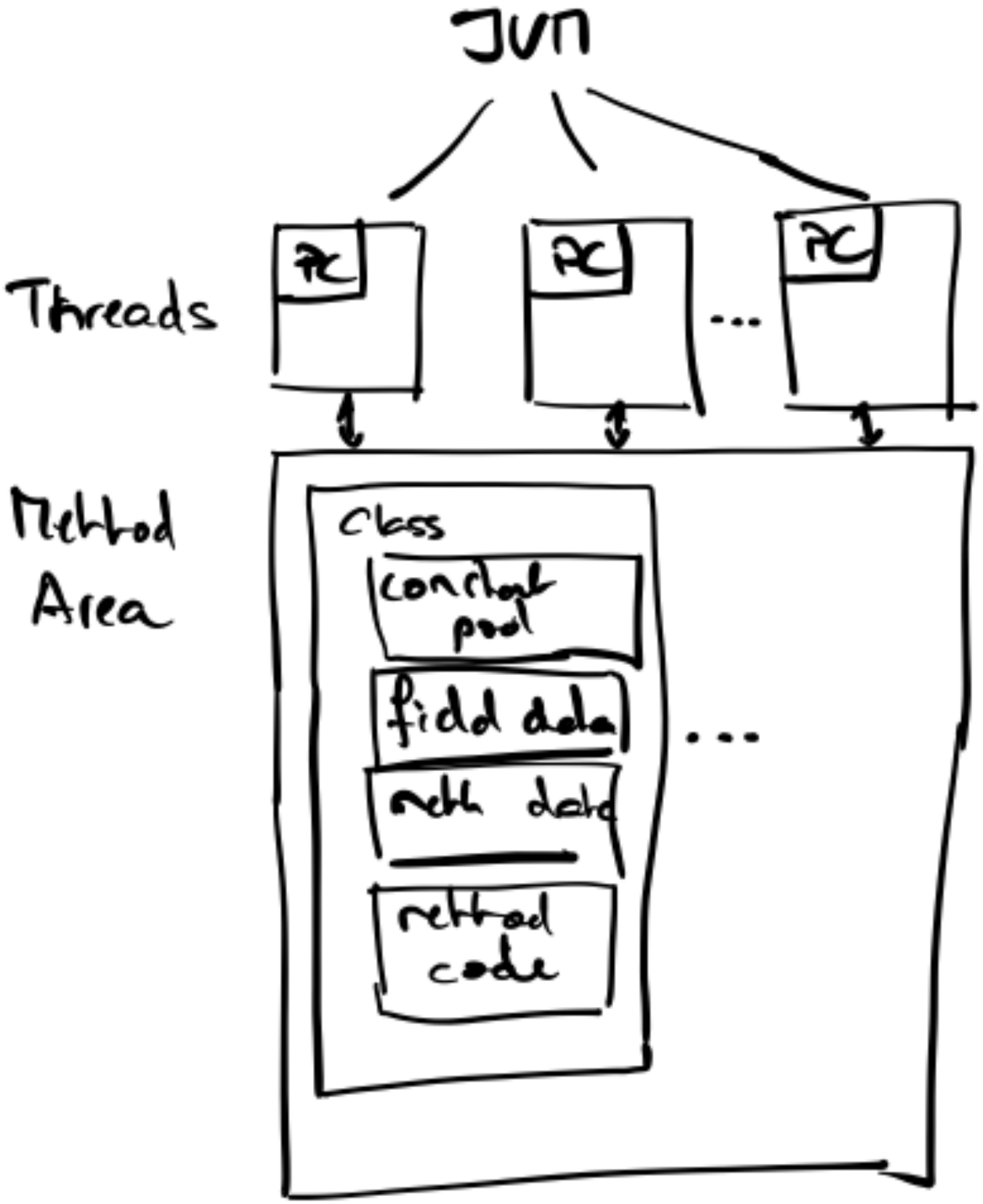

JVM
Threads

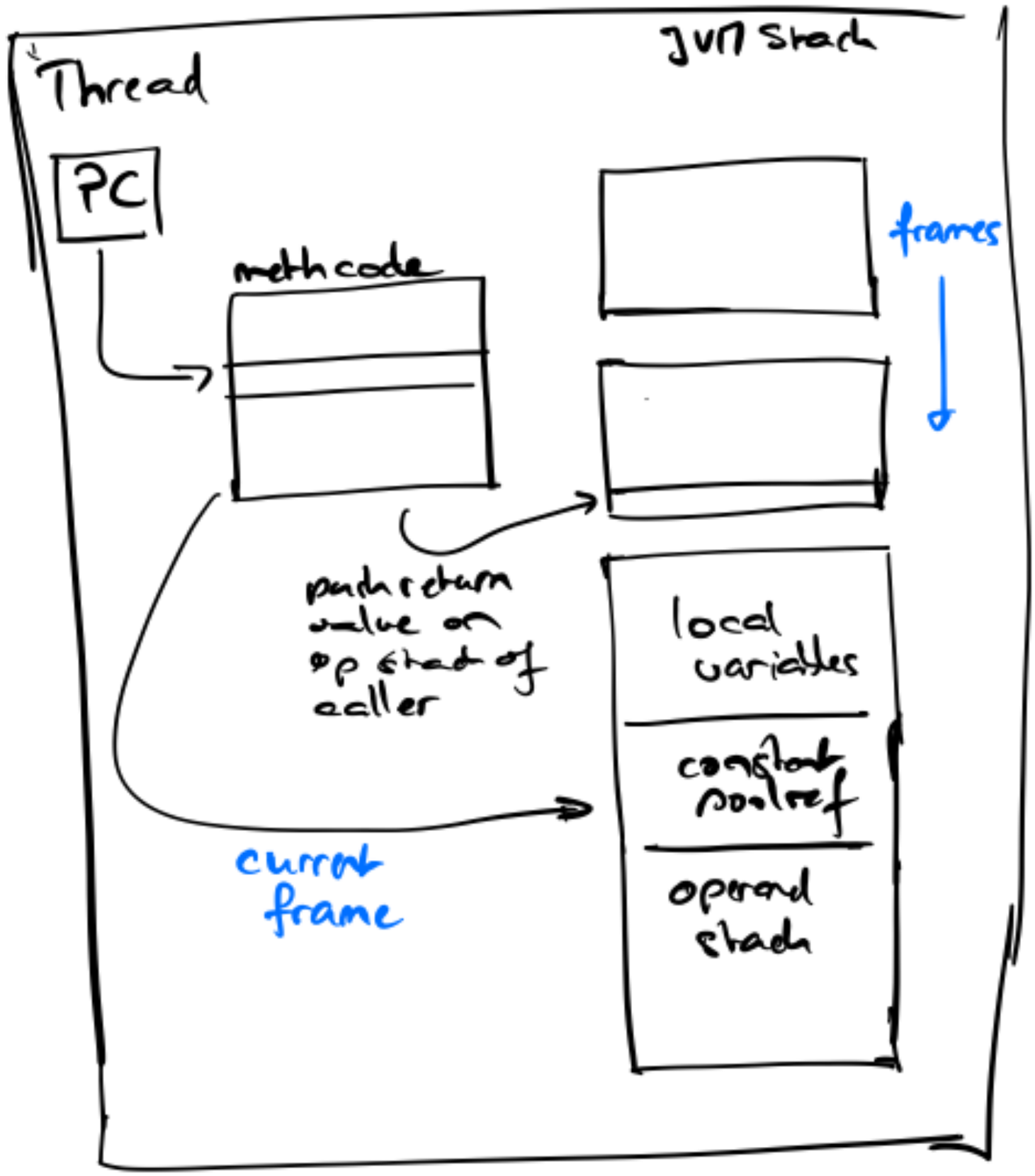

JVM
Threads


PC
Meth
Meth
current method
Thread


Thread
PC
Frames
JVM Stack


JVM
Threads
PC
PC
PC
...
Heap
Garbage
Collector


JVM
Threads
PC
PC
PC
...
Method
Area
class
constant
pool
field data
meth data
method
code
...


JVM Stack
Thread
PC
meth code
frames
push return value on op stack of caller
local variables
constant pool ref
operand stack
current frame

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

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

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

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

operand stack

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

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	

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	

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	

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	

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

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	

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	

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

operand stack

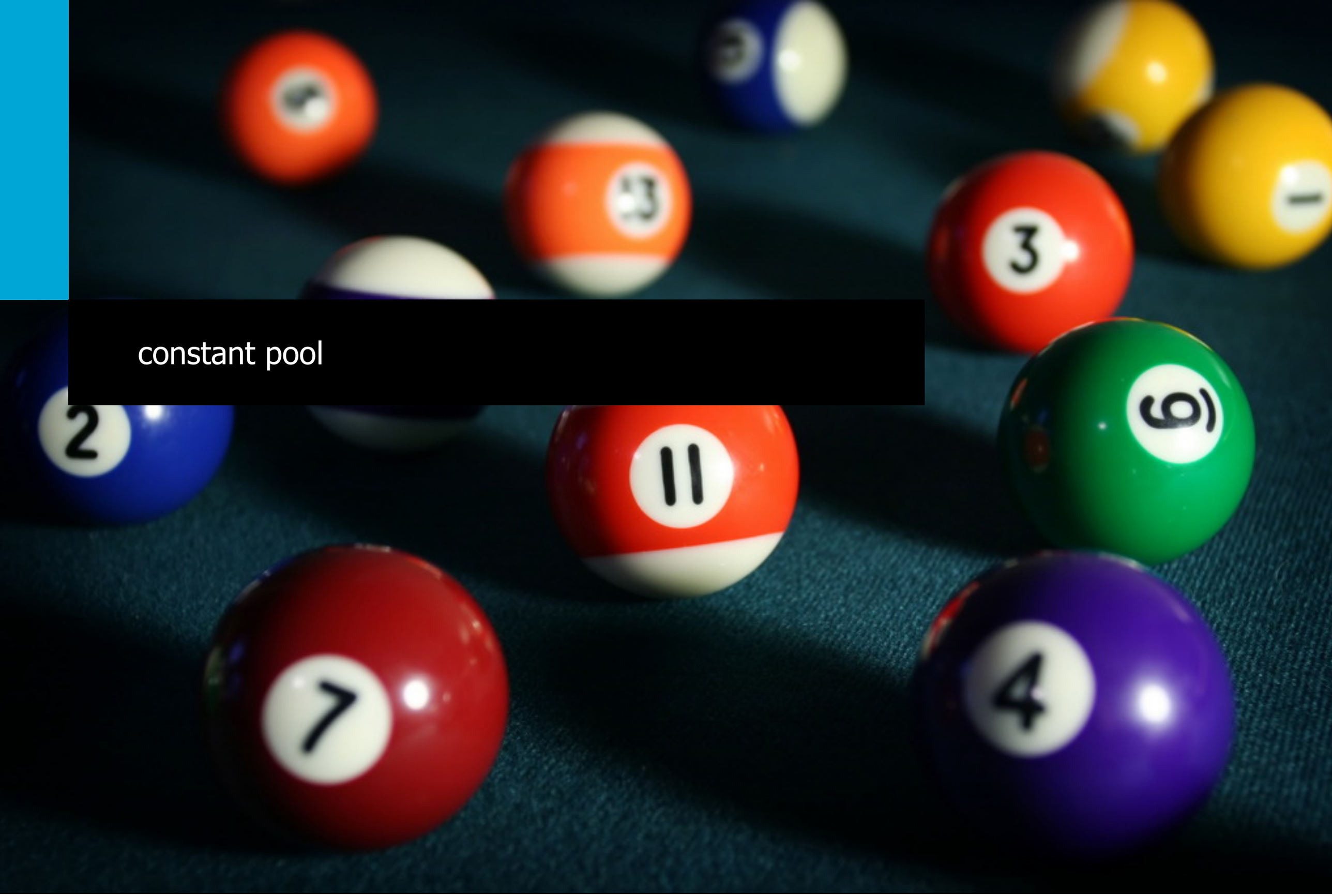
method area		
pc: 0B		
07	60	iadd
08	68	imul
09	5F	swap
0A	64	isub
0B	9A	ifne
0C	FF	
0D	F5	00

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

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



constant pool

constant pool

method area

pc: 00

00	12 ldc
01	00 00
02	12 ldc
03	01 01
04	14 ldc2_w
05	00
06	02 02

constant pool

00	0000 002A
01	0000 4303
02	0000 0000
03	0000 002A
04	
05	
06	

stack

optop: 00

00
01
02
03
04
05
06

constant pool

method area

pc: 02

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

constant pool

stack

optop: 01

00 0000 002A
01
02
03
04
05
06

constant pool

method area

pc: 04

00 12	ldc
01 00	00
02 12	ldc
03 01	01
04 14	ldc2_w
05 00	
06 02	02

constant pool

00	0000 002A
01	0000 4303
02	0000 0000
03	0000 002A
04	
05	
06	

stack

optop: 02

00	0000 002A
01	0000 4303
02	
03	
04	
05	
06	

constant pool

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

local variables

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

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

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

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

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

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

heap

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]

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]

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]

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

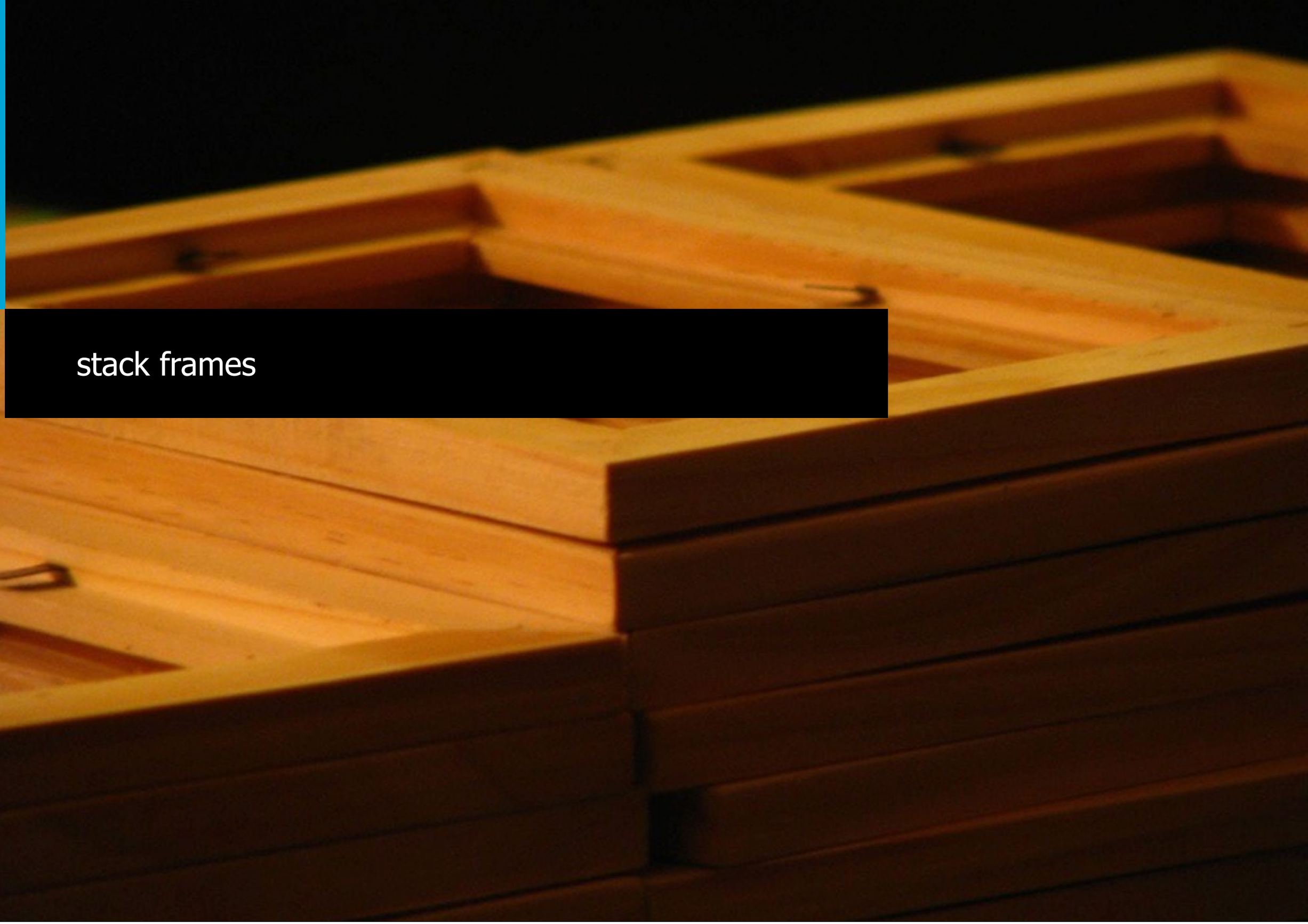
heap	
4303 4303 "Compilers"	002A 002A [20,01,40,02,42]

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

recap

static vs. dynamic dispatch

dispatch

- link method call to method

static dispatch

- type information at compile-time

dynamic dispatch

- type information at run-time
- single dispatch: one parameter
- multiple dispatch: more parameters

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

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

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

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

00 0000 0040
01
02
03
04
05
06

heap

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

heap

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 0000 1000	00 4303 4303
01	01 0000 0040
02	02
03	03
04	04
05	05
06	06

heap

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

Java Virtual Machine

class files

recap

traditional compilers

```
> 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

Jasmin

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
```

Code Generation

strings

Printing Strings

```
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"
```

String Concatenation

```
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]]
```

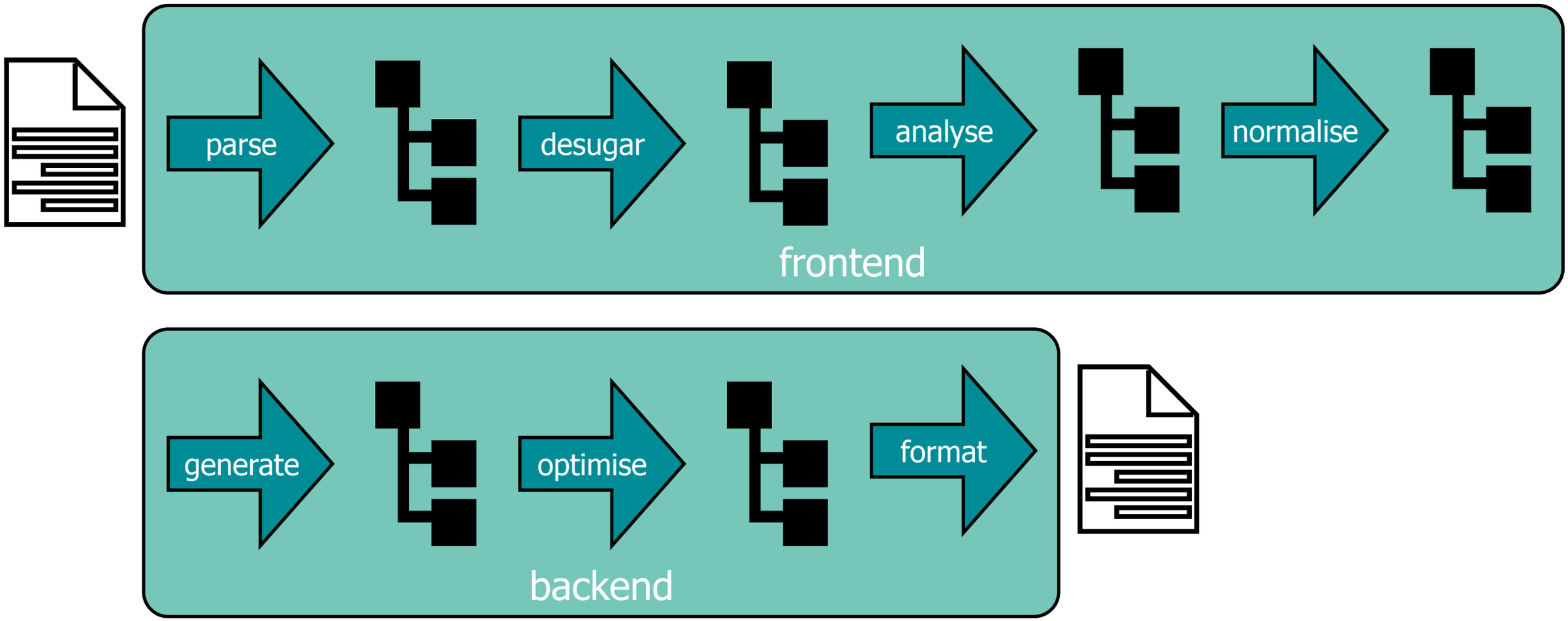
```
to-jbc: Bop(op, e1, e2) ->
  $[[<to-jbc> e1]
    [<to-jbc> e2]
    [<to-jbc> op]]

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

Code Generation transformation

Recap

compilation by transformation



Transformation

```
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()
```

Transformation

```
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
```

Transformation

example

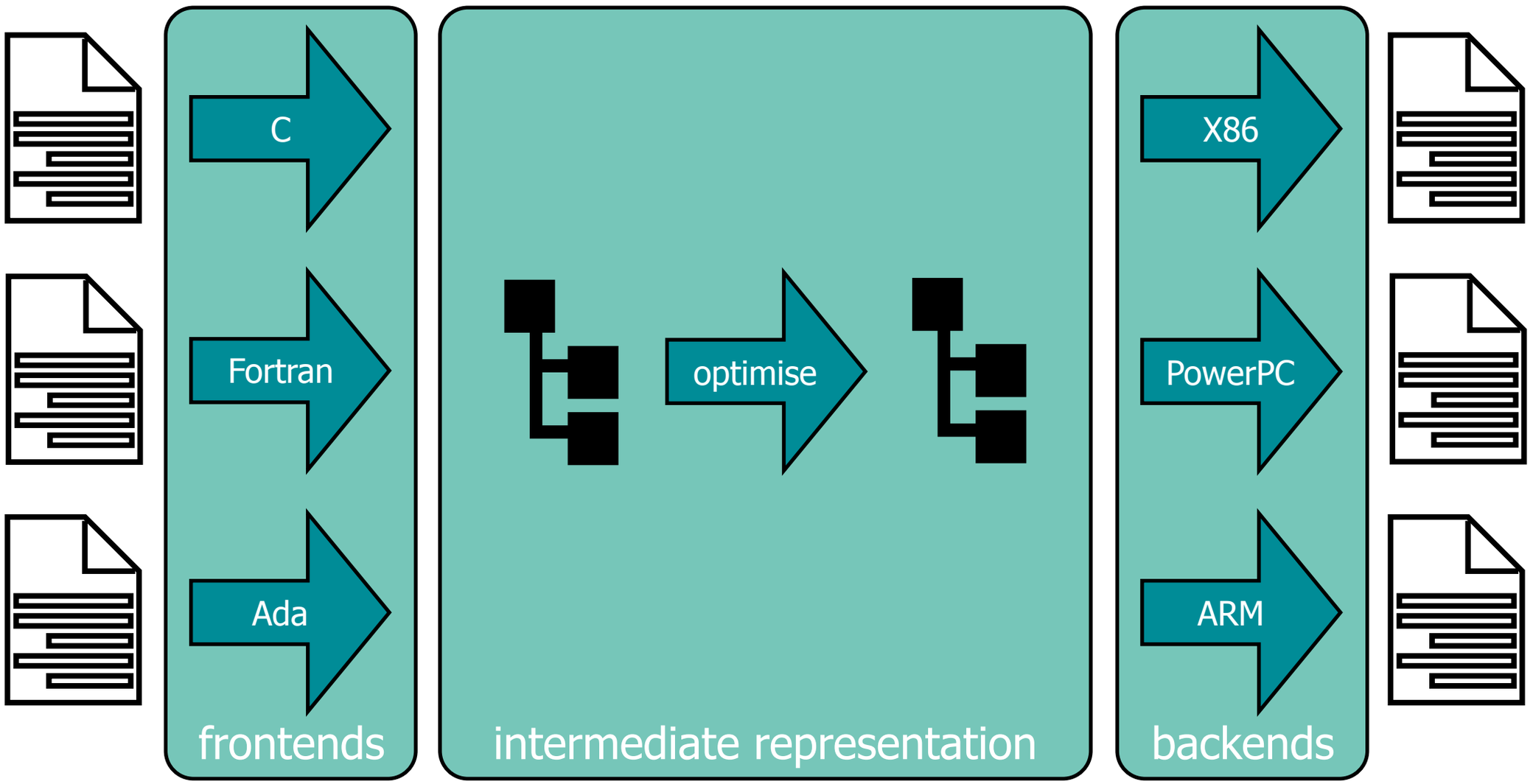
```
function fac(n: int): int=
   if
      n = 0
   then
      1
   else
      n * fac(n - 1)
```

```
.method public static fac(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/fac(I)I
            imul
      end0:   ireturn
.end method
```

LLVM

compiler infrastructure



Calling Conventions

Java Virtual Machine

Example

static call

```
function fac(n: int): int=
    if
        n = 0
    then
        1
    else
        n * fac(n - 1)
```

```
.class public Exp

    .method public static fac(I)I

                iload 1
                ifne else
                iconst_1
                ireturn

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

Example

dynamic call

```
function fac(n: int): int=
   if
      n = 0
   then
      1
   else
      n * fac(n - 1)
```

```
.class public Exp

   .method public fac(I)I

                iload 1
                ifne else
                iconst_1
                ireturn

          else: iload 1
                iload 0
                iload 1
                iconst_1
                isub
                invokevirtual Exp/fac(I)I
                imul
                ireturn
   .end method
```

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

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

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

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

00 0000 0040
01
02
03
04
05
06

heap

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

heap

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

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

Responsibilities

method call

Caller

- push object
- push parameters left-to-right
- call method

Virtual machine on call

- allocate space (frame data, operand stack, local variables)
- store frame data (data pointer, return address, exception table)
- store parameters as local variables
- dynamic dispatch
- point pc to method code

Responsibilities

return from method call

Callee

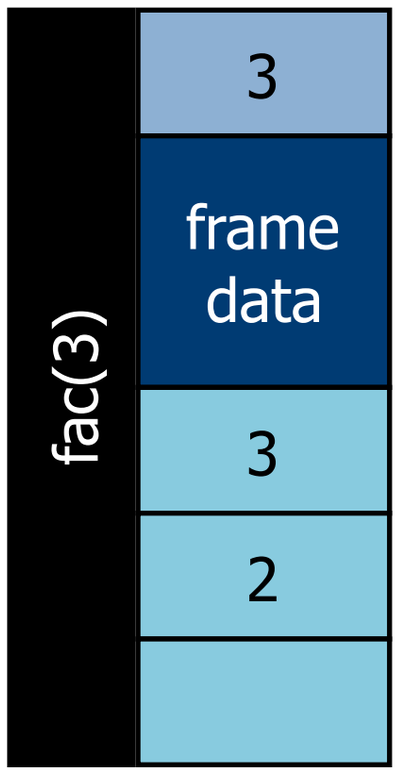
- parameters in local variables
- leave result on operand stack
- return to caller

Virtual machine on return

- push result on caller's operand stack
- point pc to return address
- destroy frame

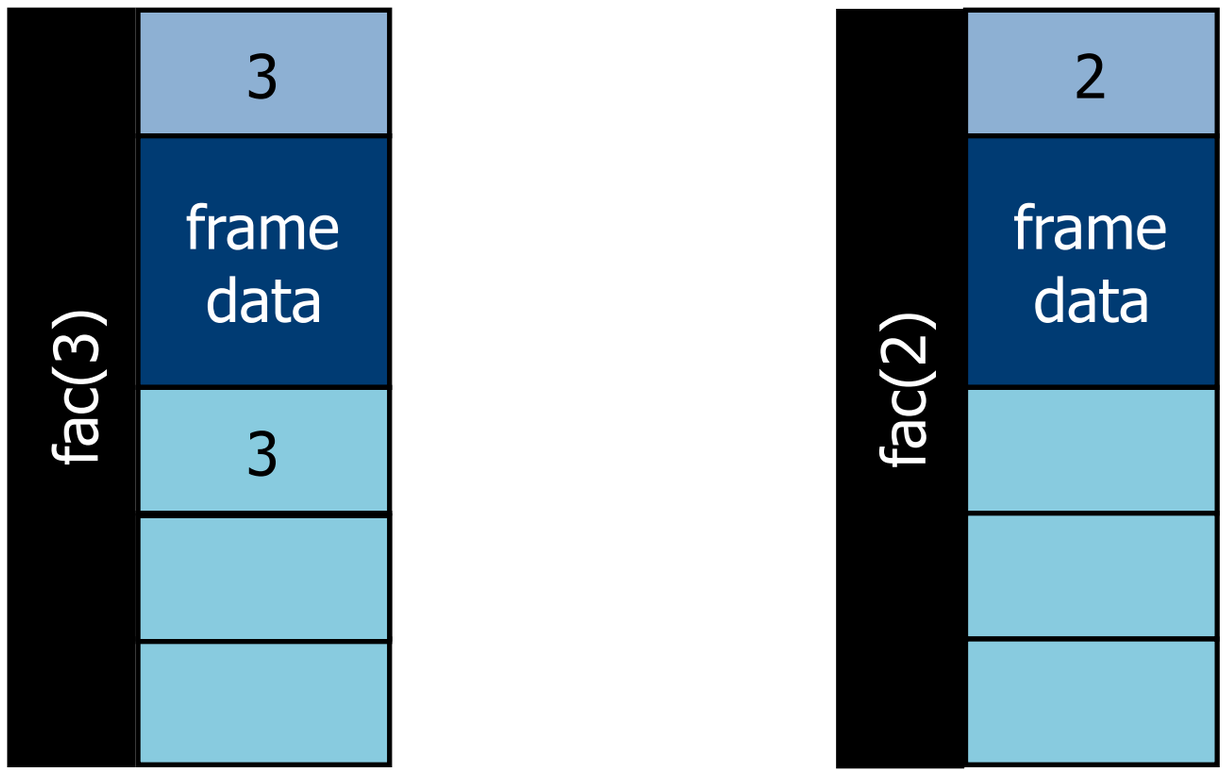
Implementation

heap-based



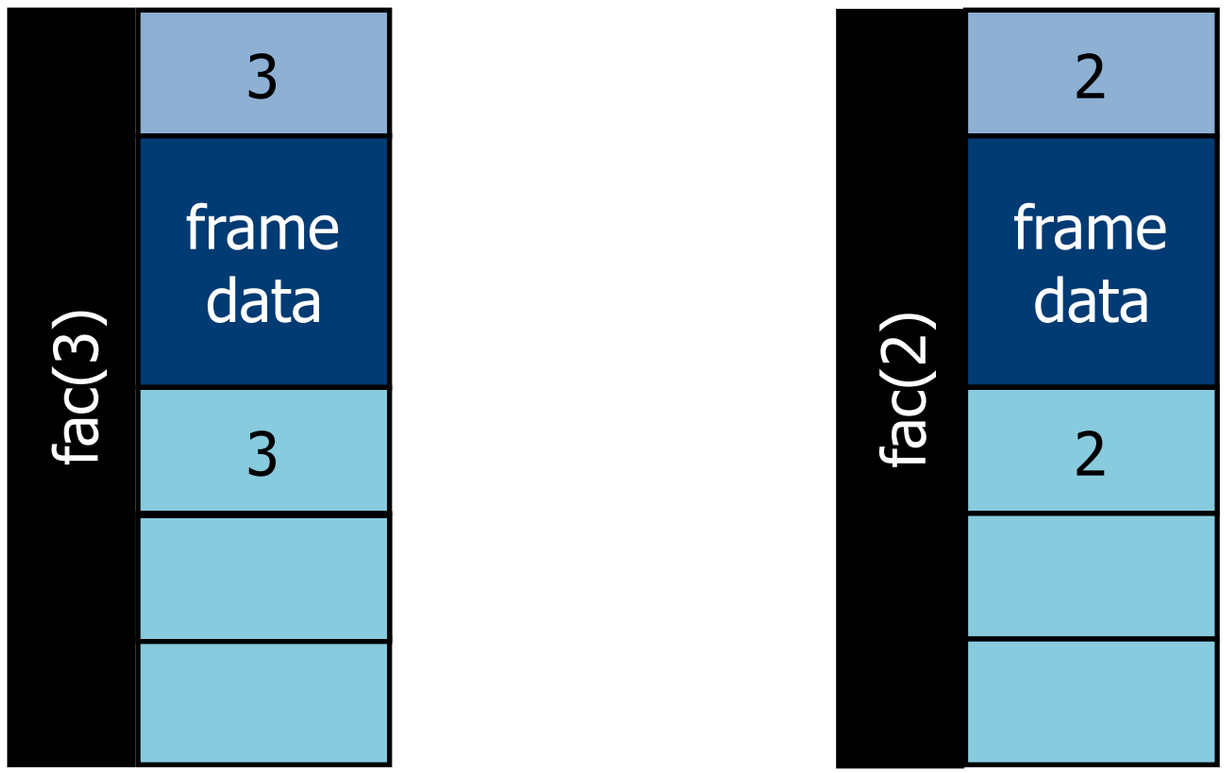
Implementation

heap-based



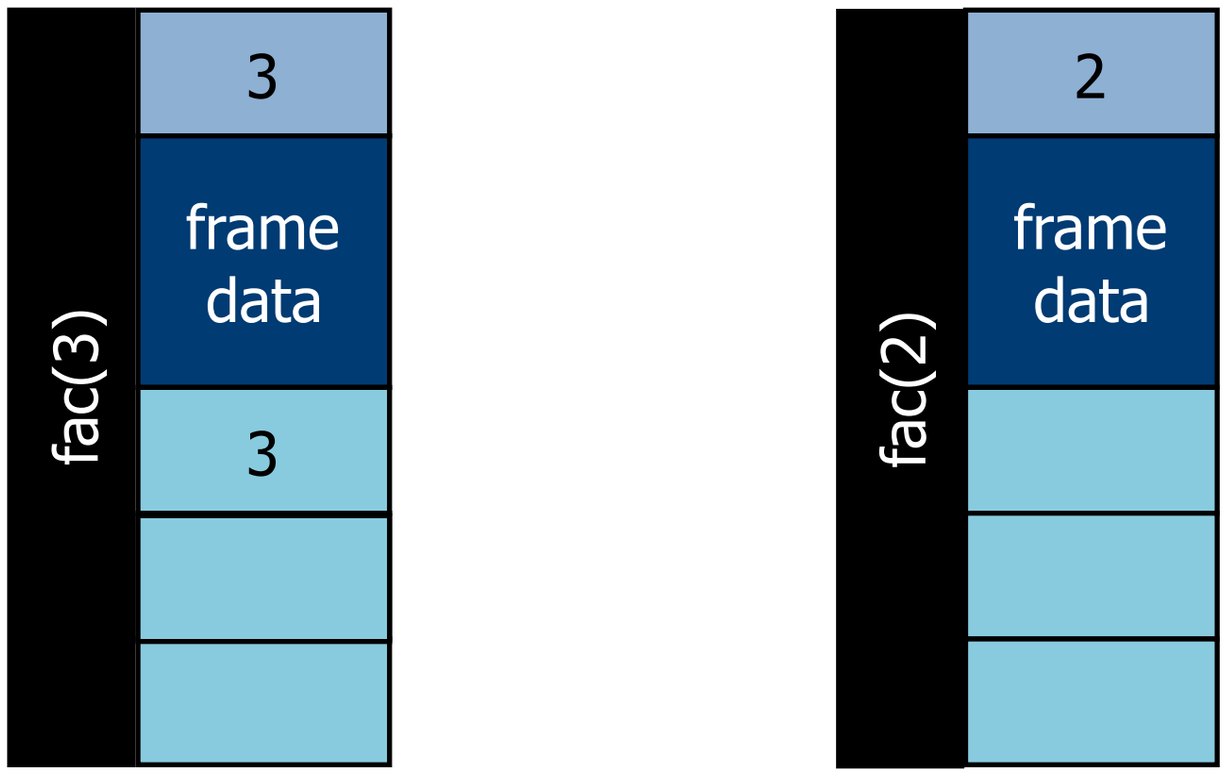
Implementation

heap-based



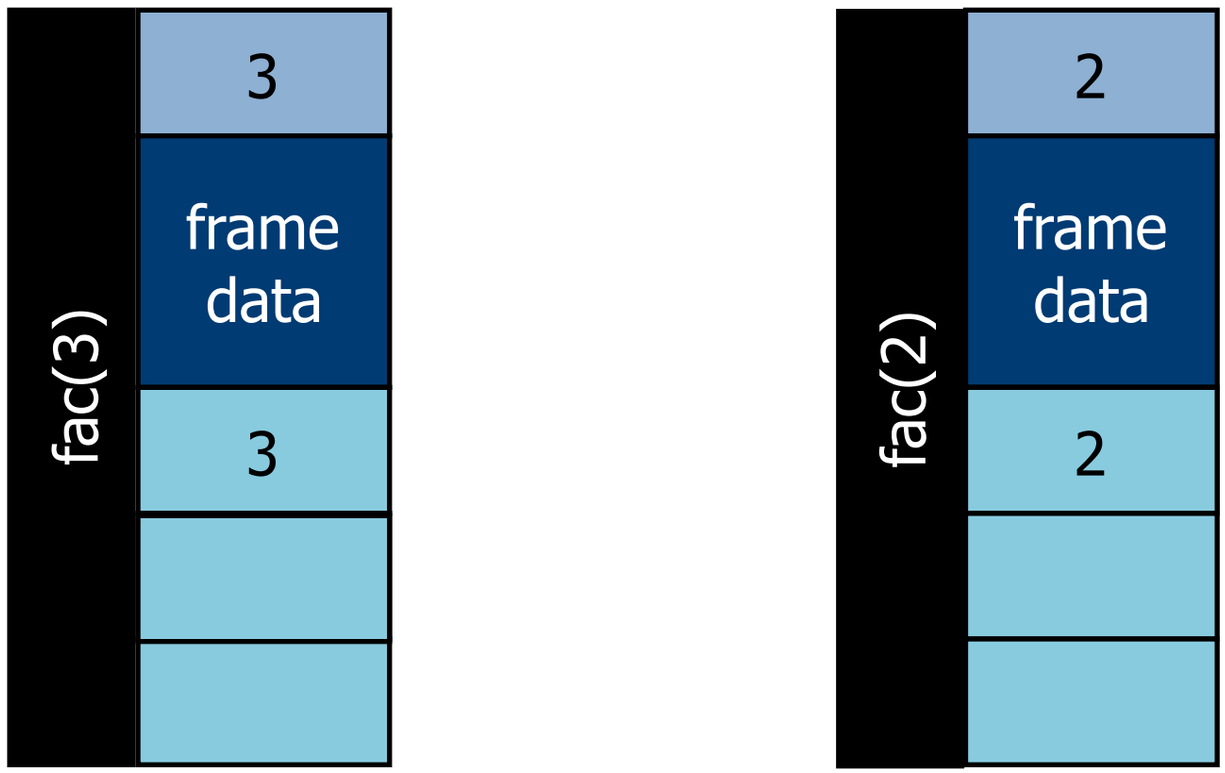
Implementation

heap-based



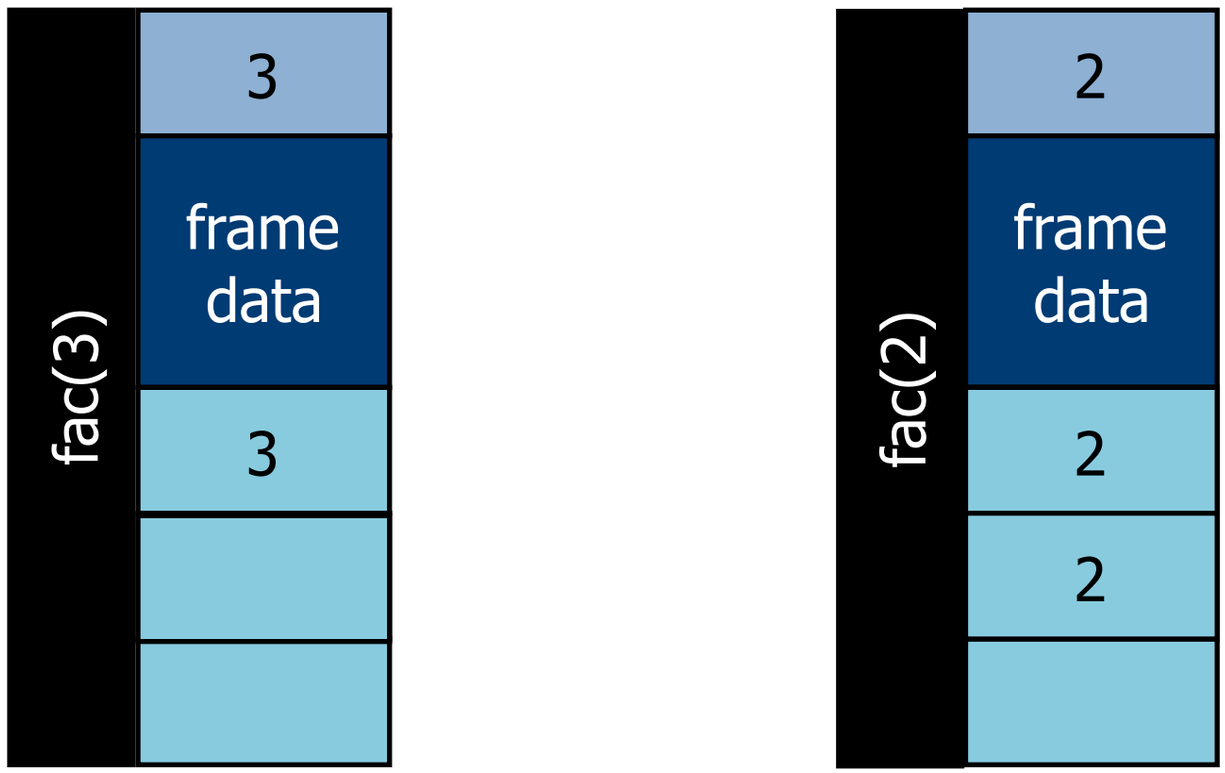
Implementation

heap-based



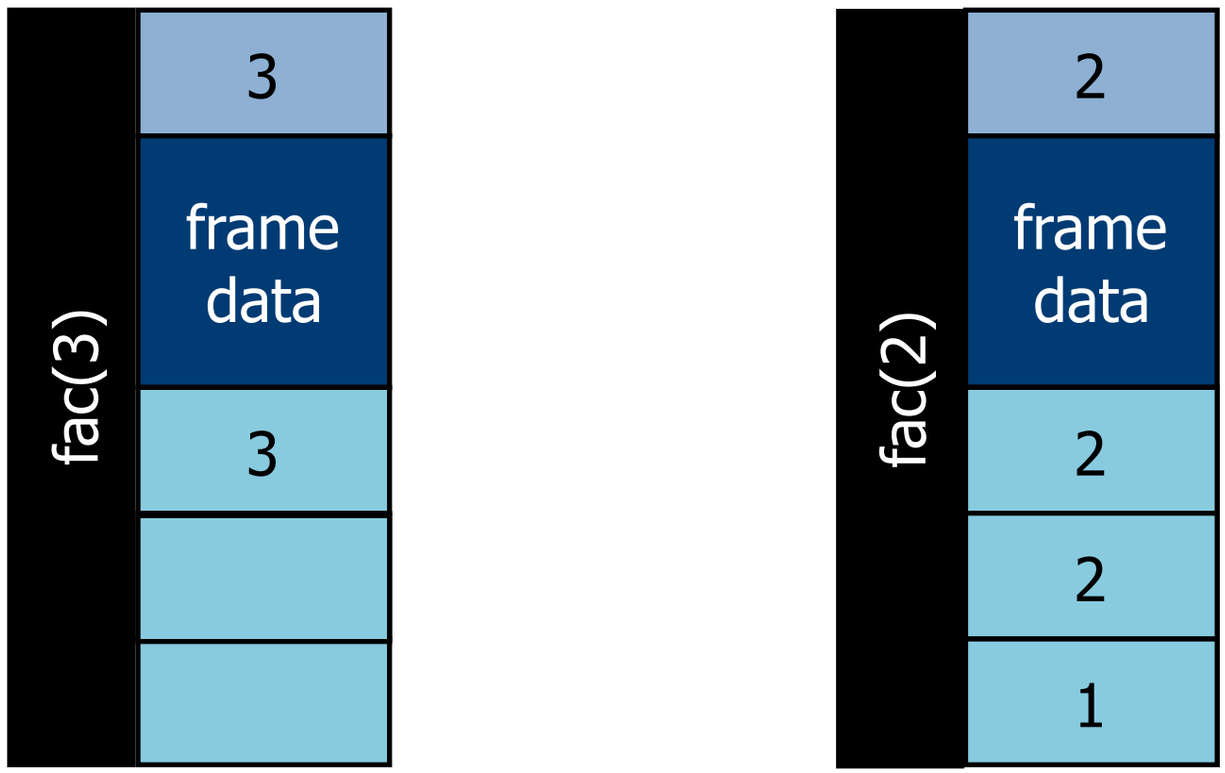
Implementation

heap-based



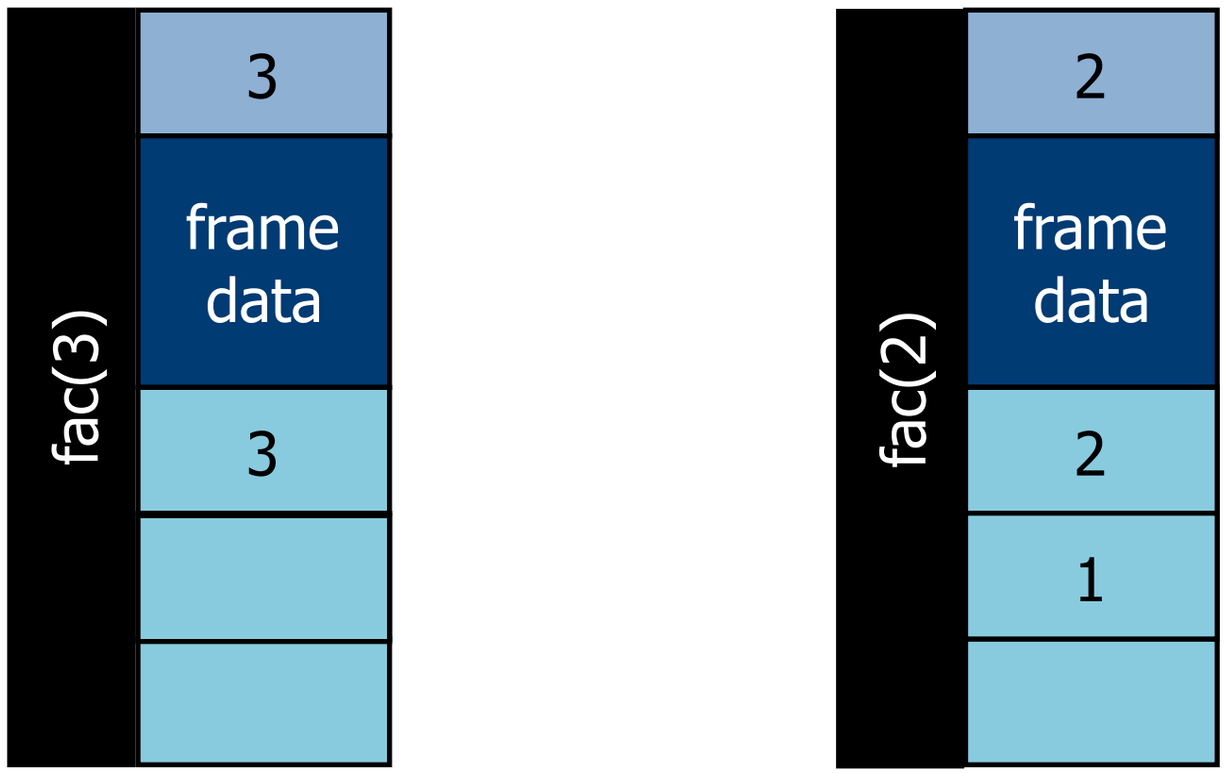
Implementation

heap-based



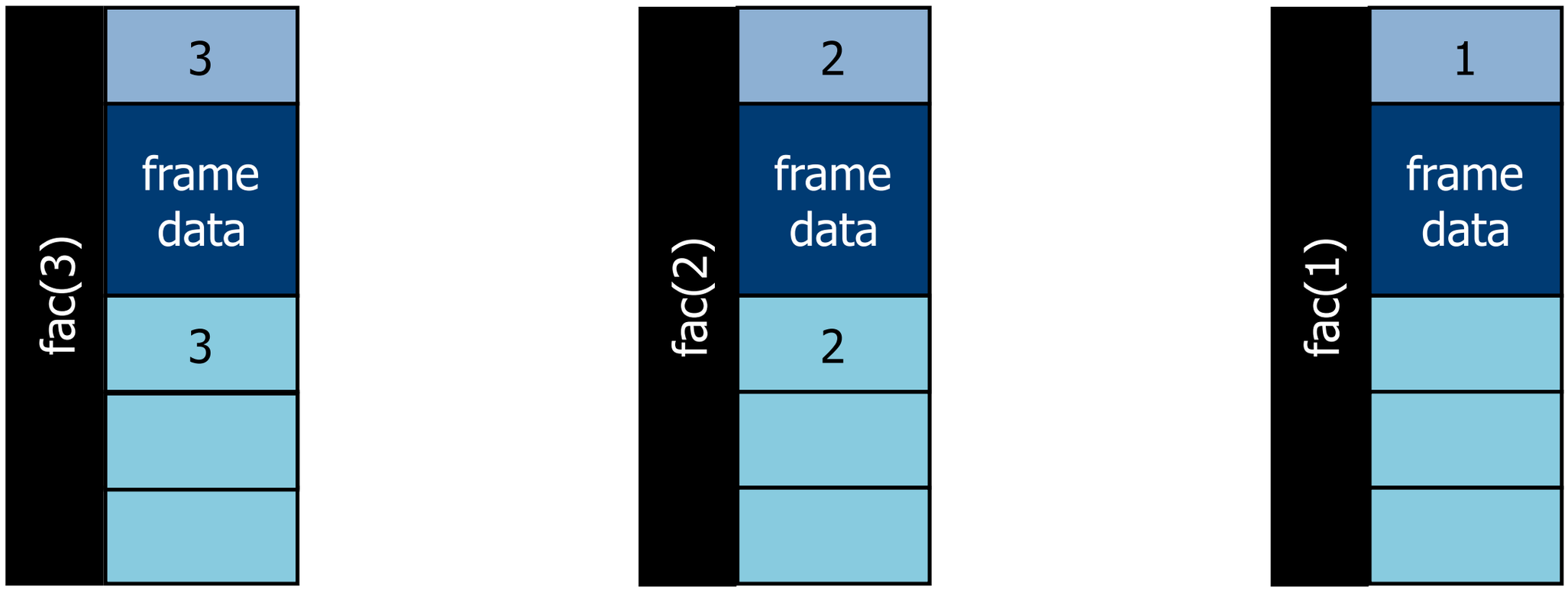
Implementation

heap-based



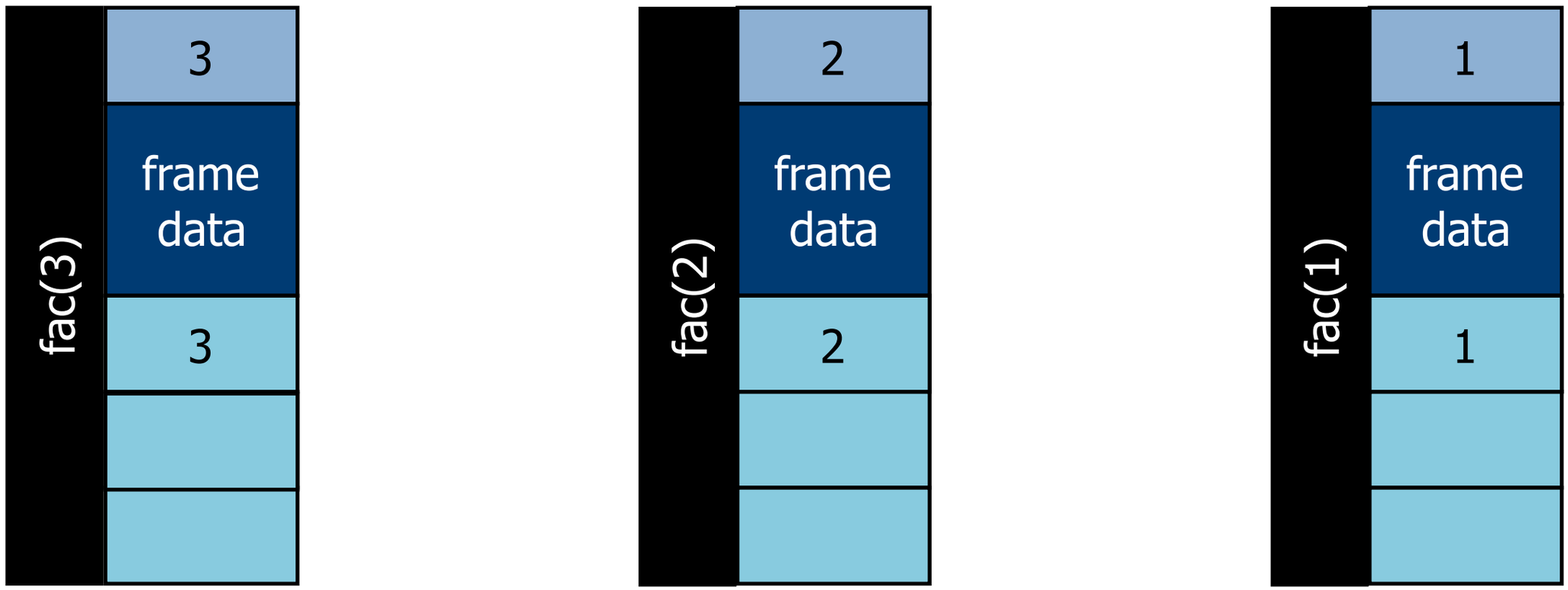
Implementation

heap-based



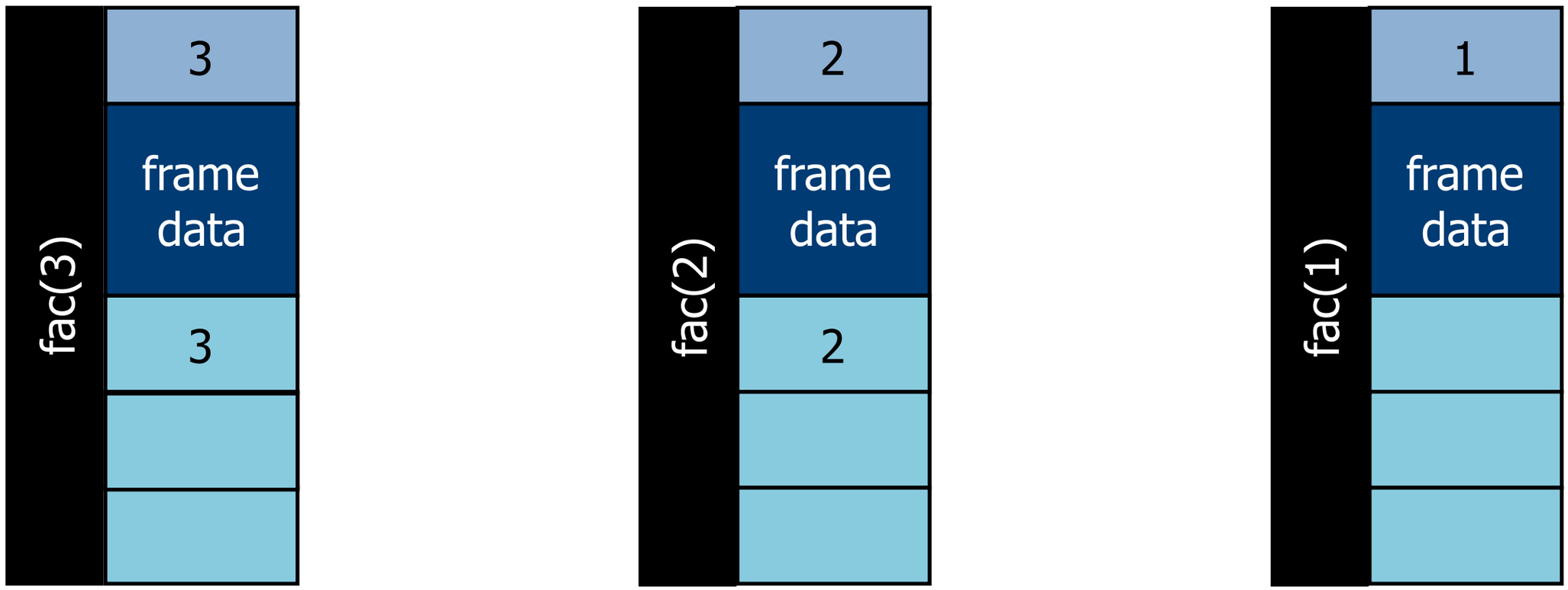
Implementation

heap-based



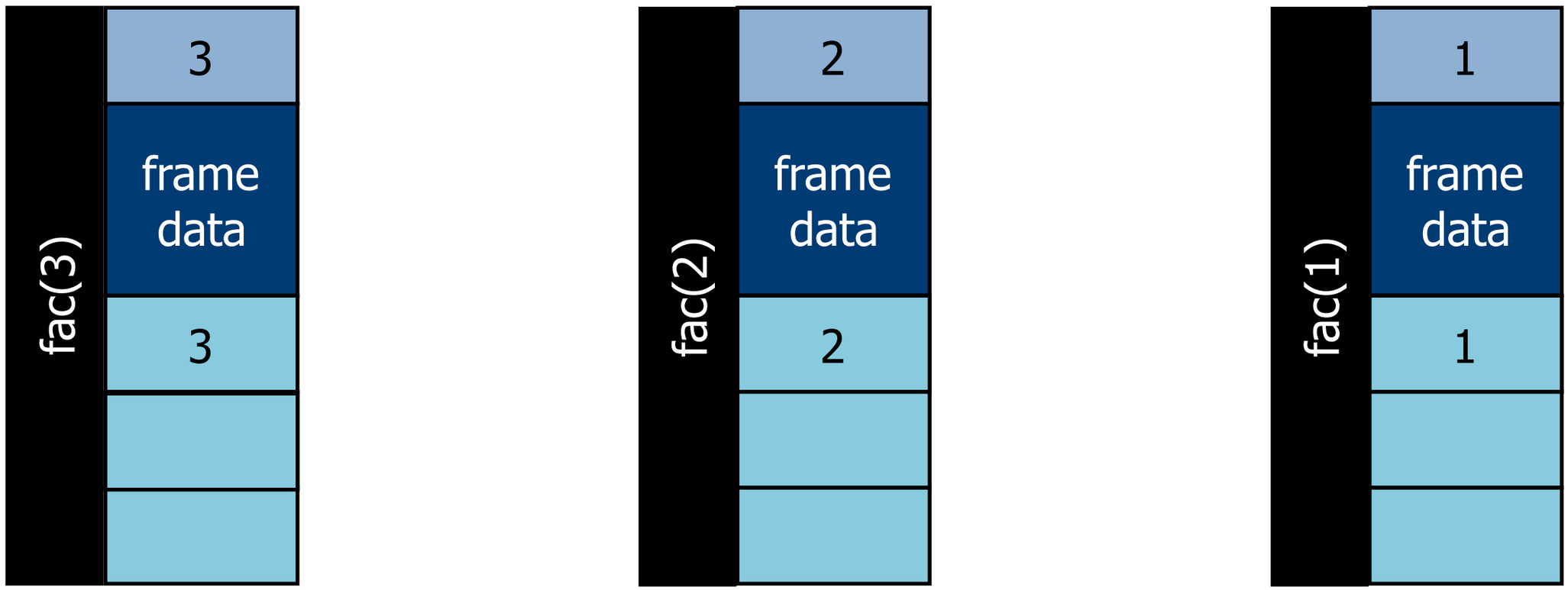
Implementation

heap-based



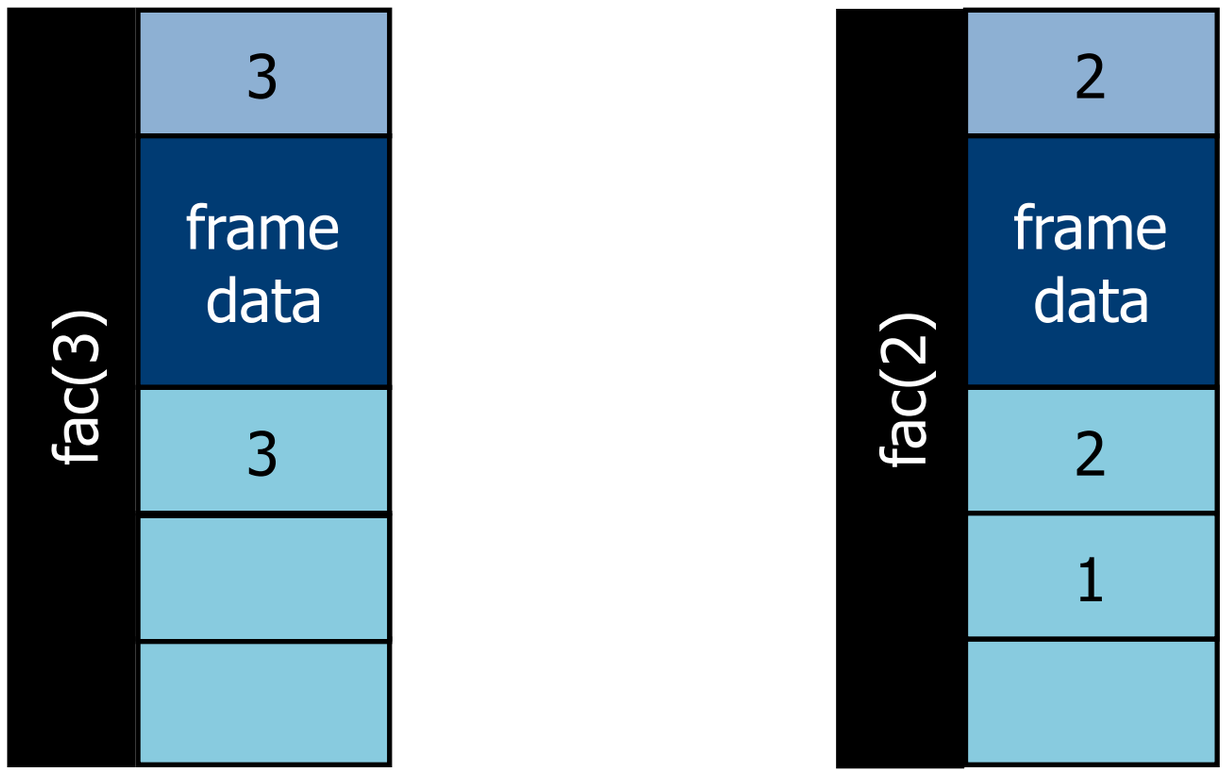
Implementation

heap-based



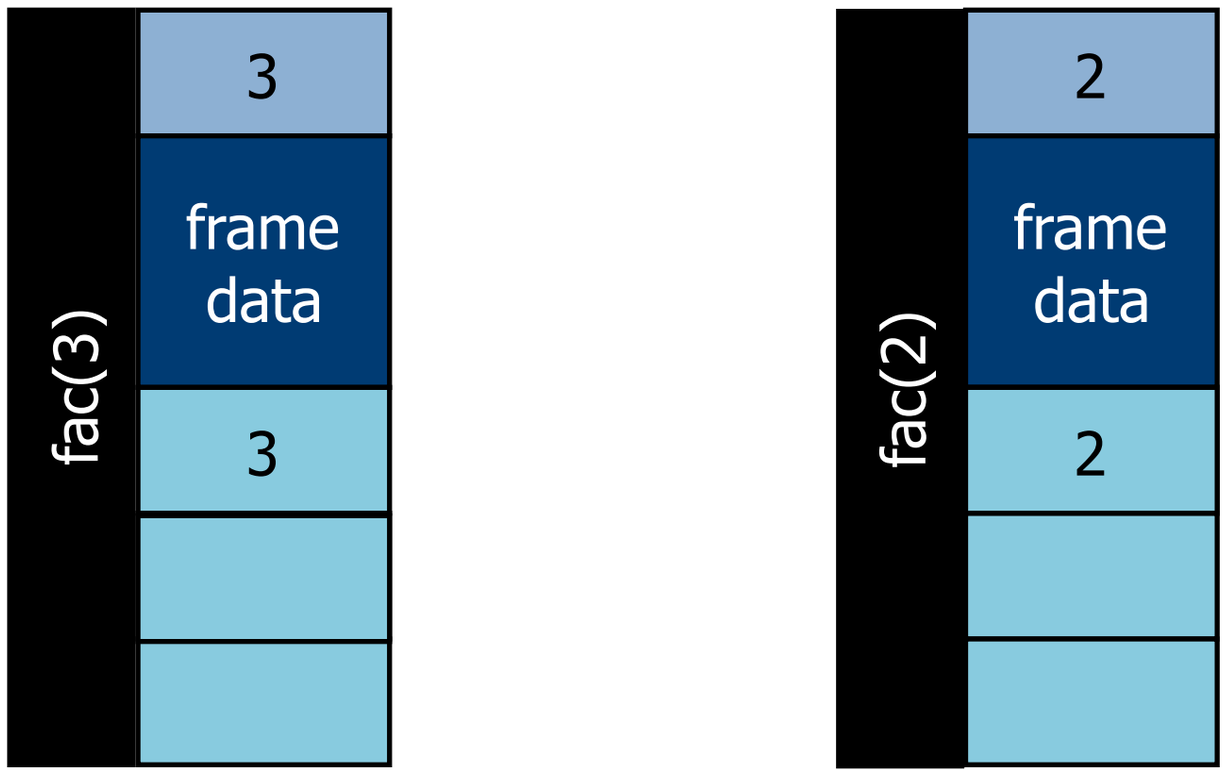
Implementation

heap-based



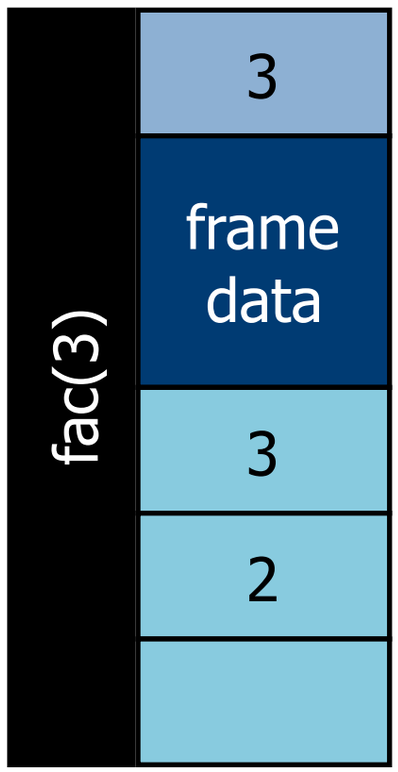
Implementation

heap-based



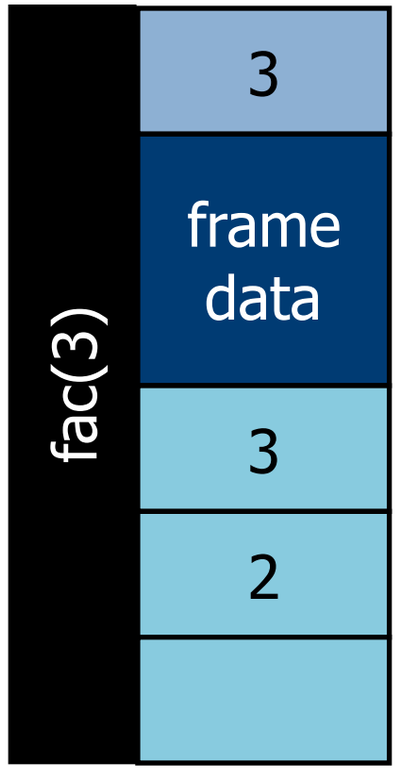
Implementation

heap-based



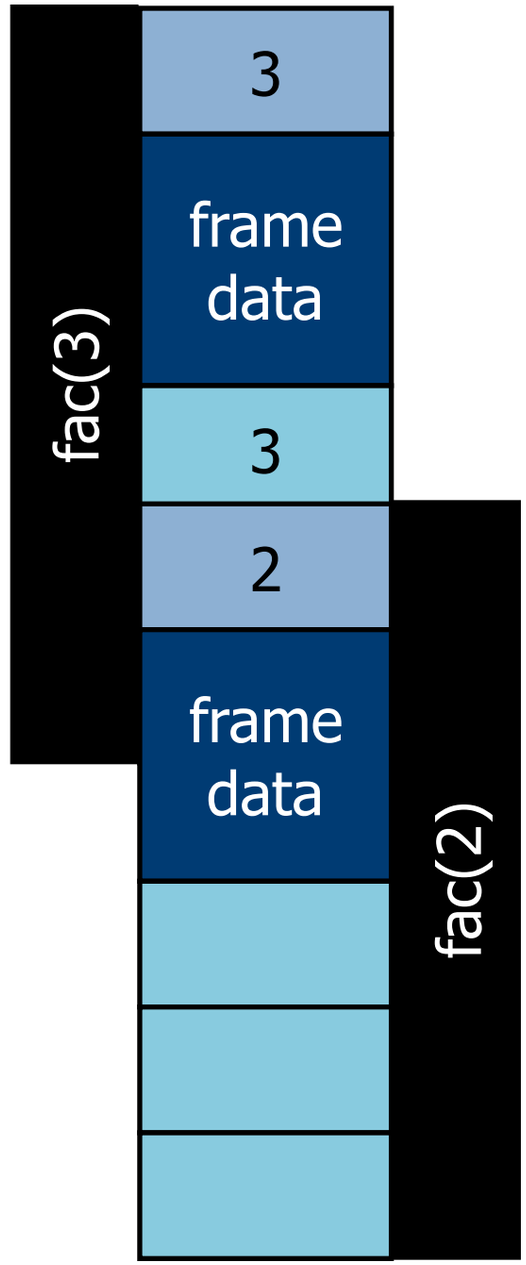
Implementation

stack-based



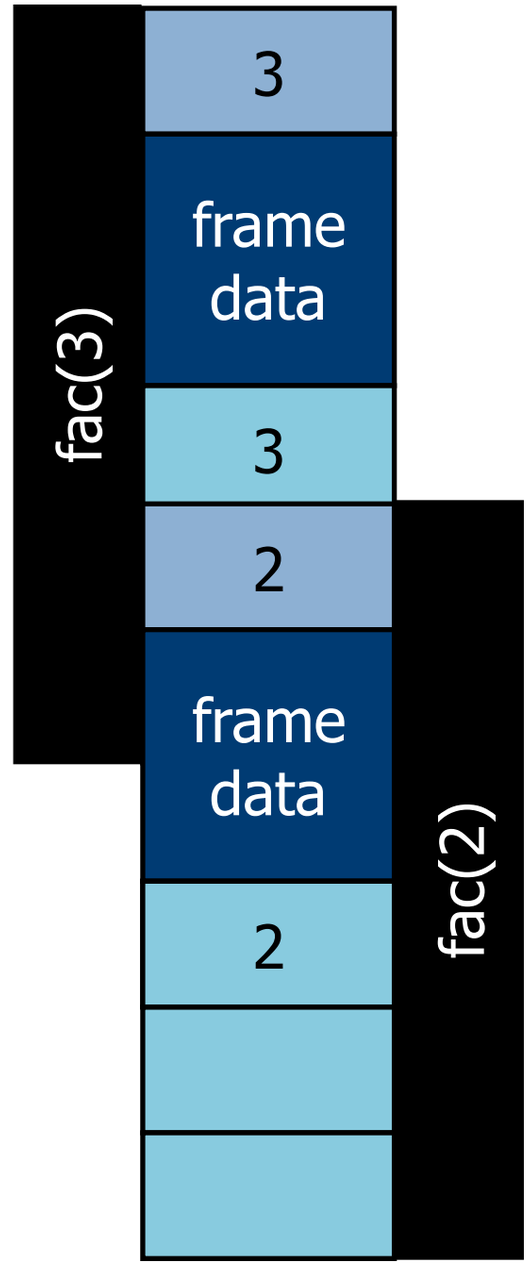
Implementation

stack-based



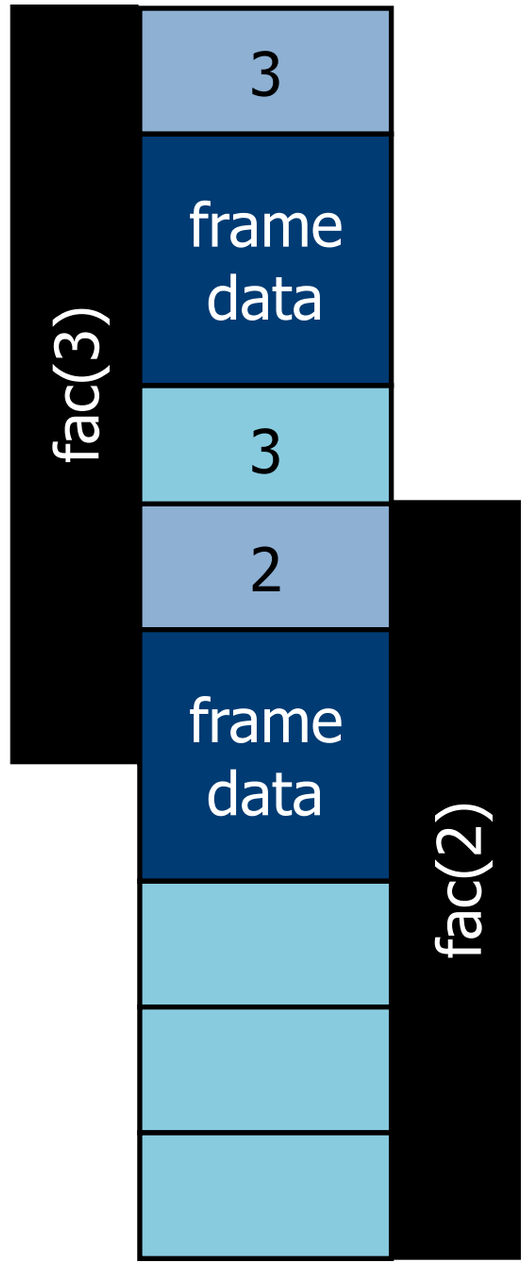
Implementation

stack-based



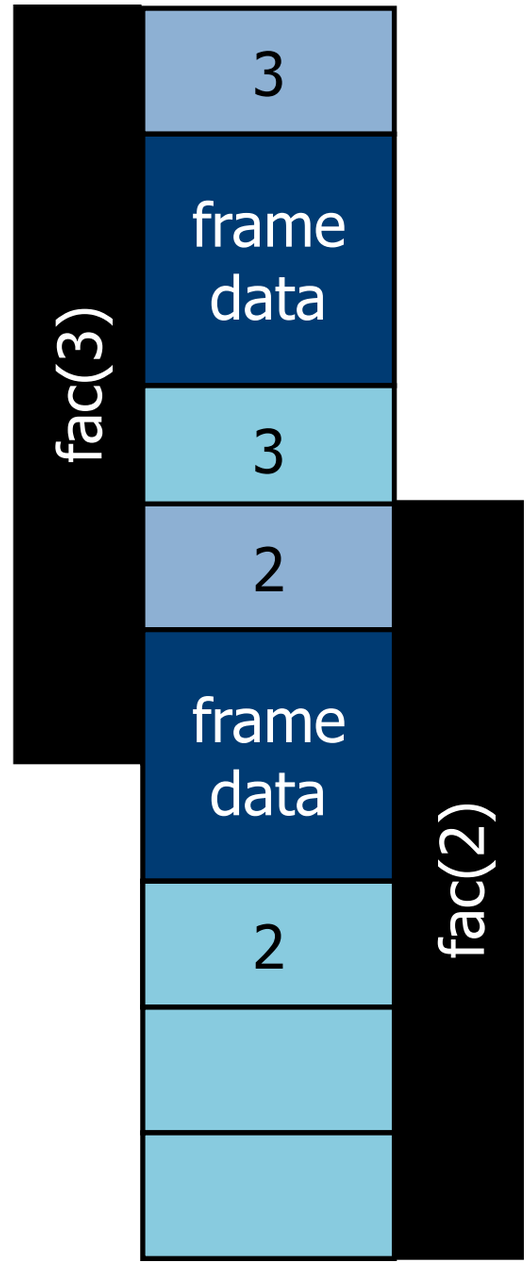
Implementation

stack-based



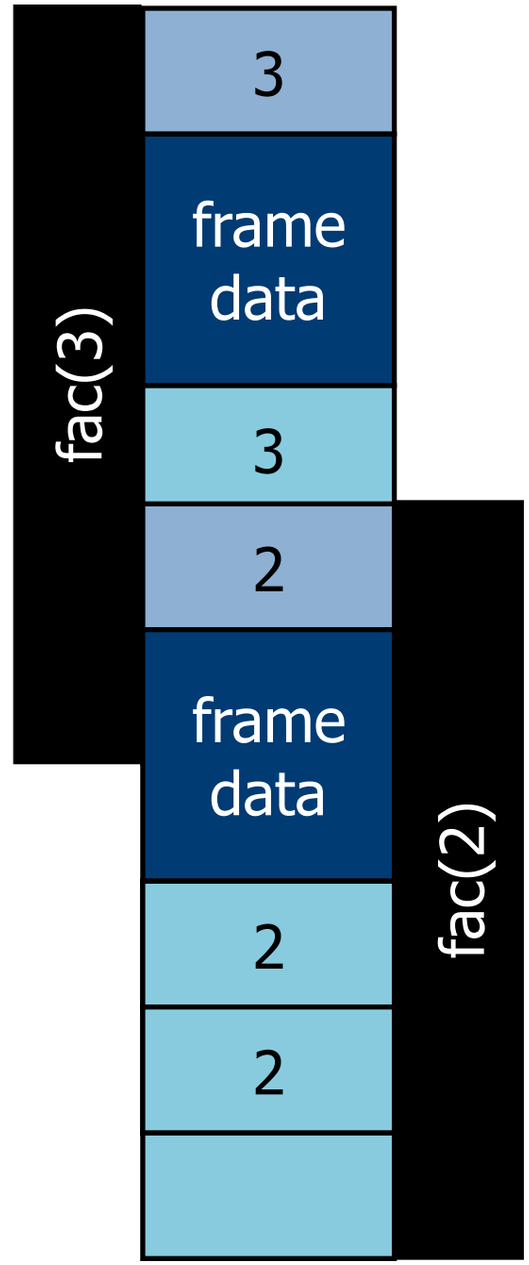
Implementation

stack-based



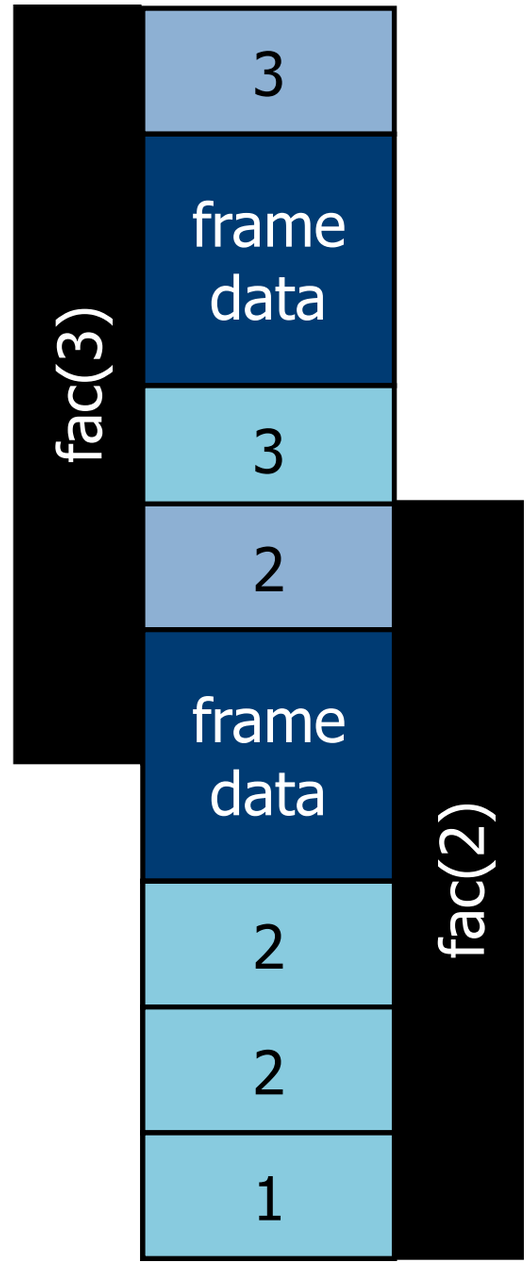
Implementation

stack-based



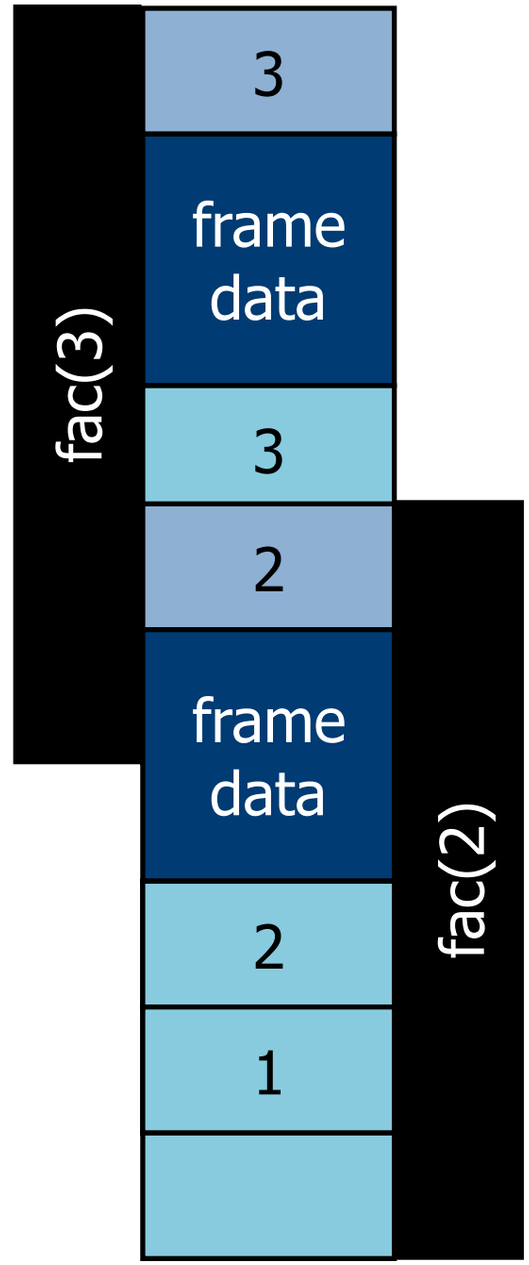
Implementation

stack-based



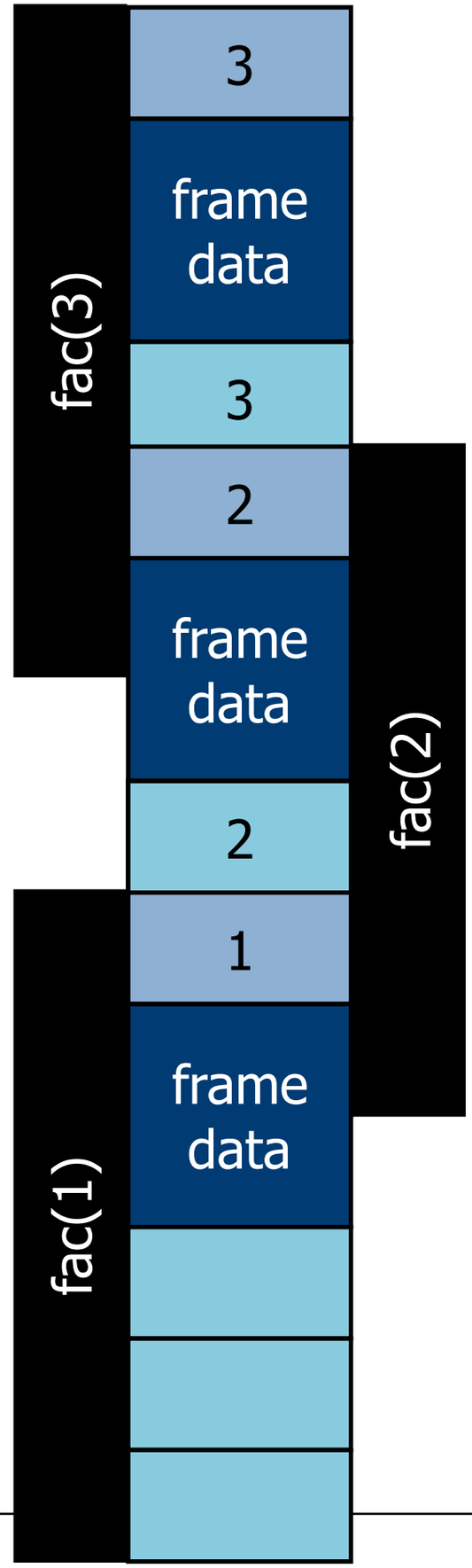
Implementation

stack-based



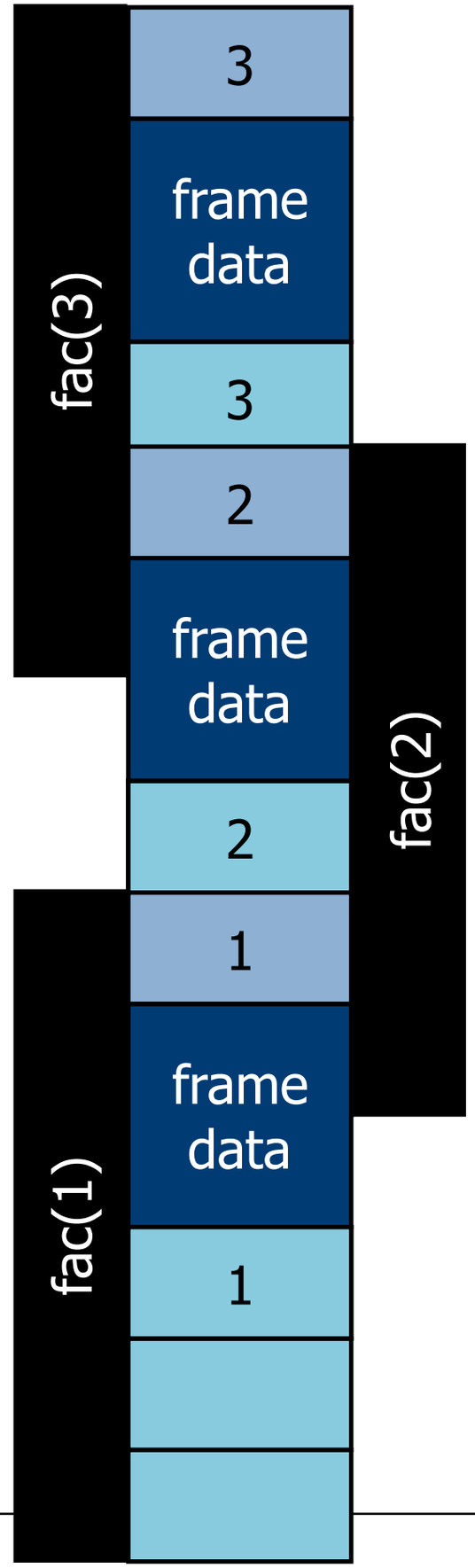
Implementation

stack-based



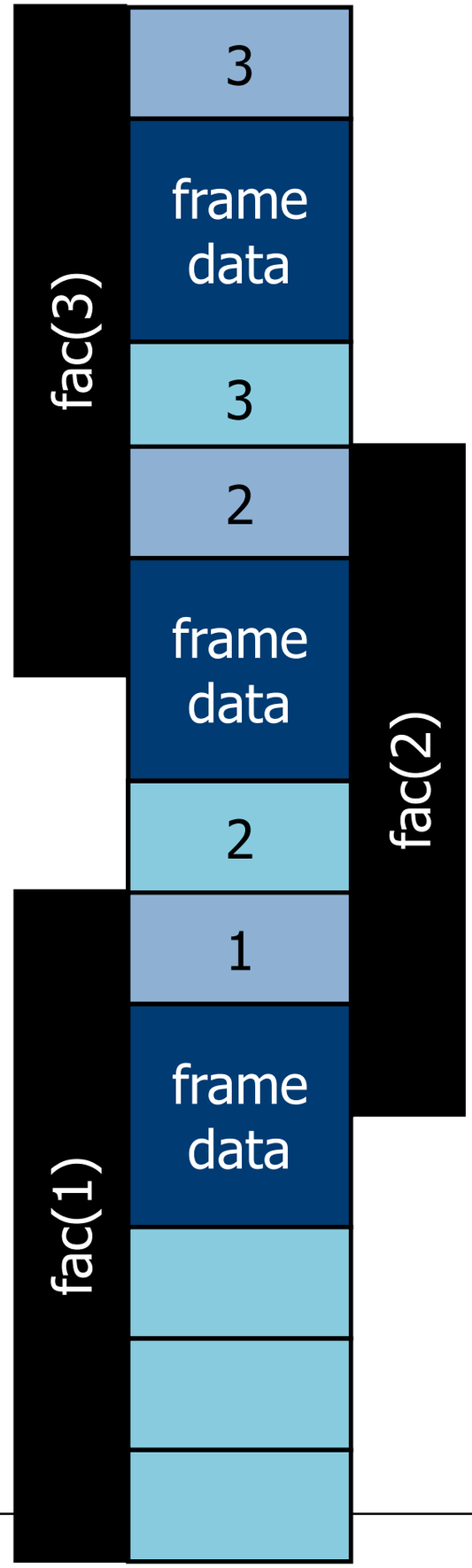
Implementation

stack-based



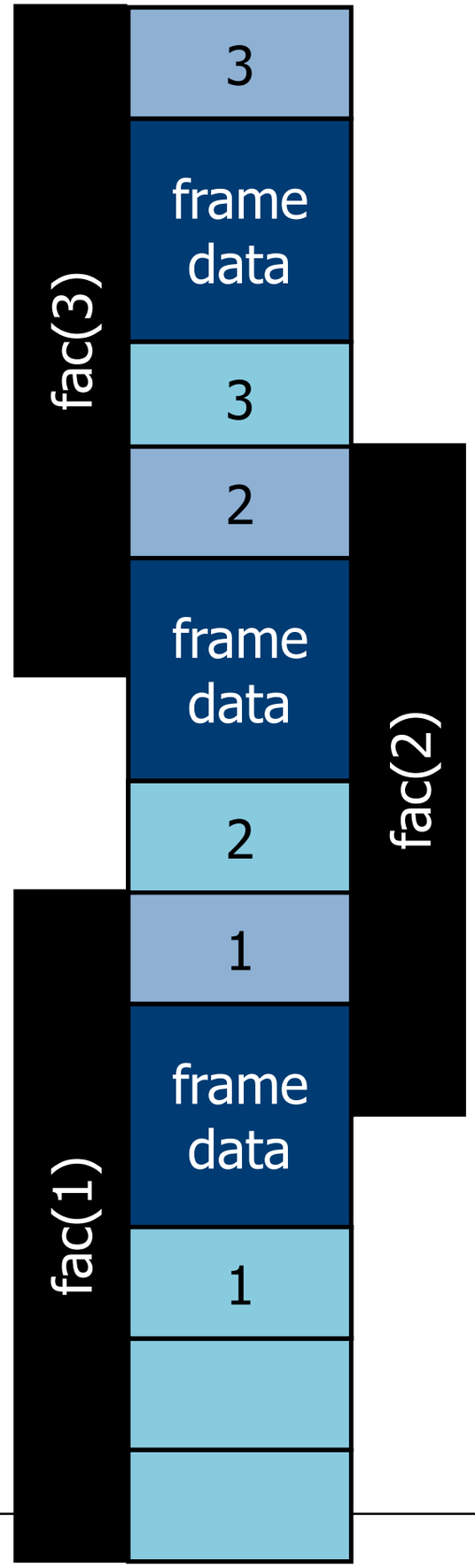
Implementation

stack-based



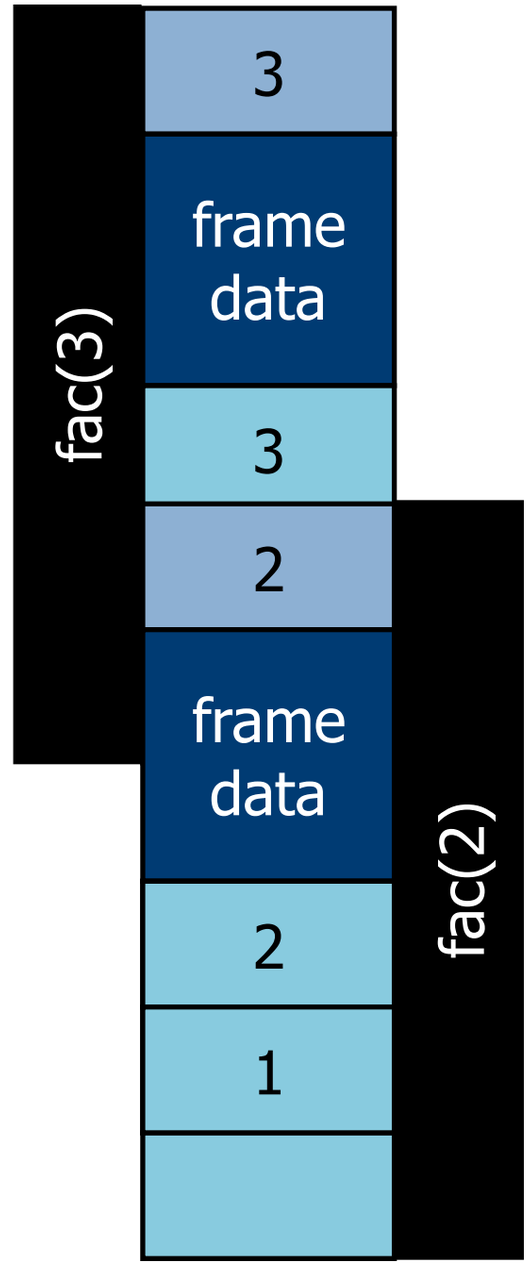
Implementation

stack-based



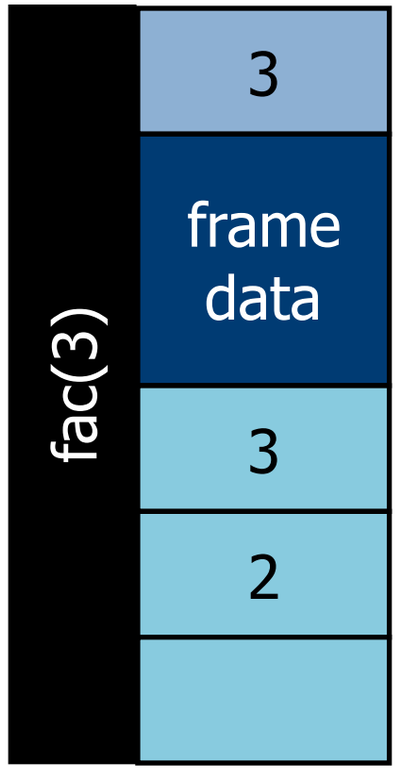
Implementation

stack-based



Implementation

stack-based



Calling Conventions

register-based machines

Recap

x86 family

General purpose registers

- accumulator AX - arithmetic operations
- counter CX - shift/rotate instructions, loops
- data DX - arithmetic operations, I/O
- base BX - pointer to data
- stack pointer SP, base pointer BP - top and base of stack
- source SI, destination DI - stream operations

Special purpose registers

- segments SS, CS, DS, ES, FS, GS
- flags EFLAGS

Stack and Stack Frames

Stack

- temporary storage
- grows from high to low memory addresses
- starts at SS

Stack frames

- return address
- local variables
- parameters
- stack base: BP
- stack top: SP

Calling Conventions

CDECL

Caller

- push parameters right-to-left on the stack
- clean-up stack after call

Callee

- save old BP
- initialise new BP
- save registers
- return result in AX
- restore registers
- restore BP

Calling Conventions

CDECL

Caller

- push parameters right-to-left on the stack
- clean-up stack after call

```
push 21
push 42
call _f
add  ESP 8
```

Callee

- save old BP
- initialise new BP
- save registers
- return result in AX
- restore registers
- restore BP

```
push EBP
mov  EBP ESP
mov  EAX [EBP + 8]
mov  EDX [EBP + 12]
add  EAX EDX
pop  EBP
ret
```

Calling Conventions

STDCALL

Caller

- push parameters right-to-left on the stack

Callee

- save old BP, initialise new BP
- save registers
- return result in AX
- restore registers
- restore BP
- cleans up the stack

Calling Conventions

STDCALL

Caller

push parameters right-to-left on the stack

```
push 21
push 42
call _f@8
```

Callee

save old BP, initialise new BP

save registers

return result in AX

restore registers

restore BP

cleans up the stack

```
push EBP
mov  EBP ESP
mov  EAX [EBP + 8]
mov  EDX [EBP + 12]
add  EAX EDX
pop  EBP
ret  8
```

Calling Conventions

FASTCALL

Caller

- pass parameters in registers
- push remaining parameters right-to-left on the stack

Callee

- save old BP, initialise new BP
- save registers
- return result in AX
- restore registers
- restore BP
- cleans up the stack

Calling Conventions

FASTCALL

Caller

- pass parameters in registers
- push remaining parameters right-to-left on the stack

```
mov  ECX 21
mov  EDX 42
call @f@8
```

Callee

- save old BP, initialise new BP
- save registers
- return result in AX
- restore registers
- restore BP
- cleans up the stack

```
push EBP
mov  EBP ESP
mov  EAX ECX
add  EAX EDX
pop  EBP
ret
```

Calling Conventions

compilation

Method declarations

- in principle: full freedom
- project constraints
- target platform constraints

Method calls

- need to match method declarations

Precompiled libraries

- avoid recompilation
- source code not always available

attribution

slide	title	author	license
1	Matrix	Gamaliel Espinoza Macedo	CC BY-NC 2.0
2, 3, 59, 63	PICOL icons	Melih Bilgil	CC BY 3.0
12	Gray Legos wallpaper	monohex	CC BY-NC-SA 2.0
24	Billiard Balls	Darren Hester	some rights reserved
29	Autoturm	m.prinke	CC BY-SA 2.0
36	Spy Hill Landfill	D'Arcy Norman	some rights reserved
42	Framed	LexnGer	CC BY-NC 2.0