Activation Records

Guido Wachsmuth

Assessment
last lecture

Given an optimisation step, explain

- the optimisation,
- its validity,
- its benefits.

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
```

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

Optimisation

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

            iload 1
            ifneq else0
            ldc 1
            ireturn
   else0:   iload 1
            dup
            ldc 1
            isub
            invokestatic
               Exp/fac0(I)I
            imul
   end0:    ireturn
.end method
```

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

            iload_1
            ifneq else0
            ldc 1
            ireturn
   else0:   iload_1
            dup
            ldc 1
            isub
            invokestatic
               Exp/fac0(I)I
            imul
   end0:    ireturn
.end method
```

Optimisation

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

            iload_1
            ifneq else0
            ldc 1
            ireturn
   else0:   iload_1
            dup
            ldc 1
            isub
            invokestatic
               Exp/fac0(I)I
            imul
   end0:    ireturn
.end method
```

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

            iload_1
            ifneq else0
            iconst_1
            ireturn
   else0:   iload_1
            dup
            iconst_1
            isub
            invokestatic
               Exp/fac0(I)I
            imul
   end0:    ireturn
.end method
```

Overview
today's lecture

activation records

- procedures in imperative and object-oriented languages
- Java Virtual Machine
- register-based machines
- calling conventions

second assignment

- general remarks
- namespace library
- environment library
- reports

I

Java Virtual Machine

Recap: Modularity
procedures

imperative languages

- subroutines, routines, procedures, functions, methods
- scoping: local variables
- declarations with parameters (formal parameters)
- calls with arguments (actual parameters)
- pass by value, pass by reference

Recap: Modularity
procedures

imperative languages

- subroutines, routines, procedures, functions, methods
- scoping: local variables
- declarations with parameters (formal parameters)
- calls with arguments (actual parameters)
- pass by value, pass by reference

machine code

- jumps: call and return
- call stack: return address, parameters, private data
- procedure prologue and epilogue

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

	local variables
00 0000 0040	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: 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 0000 1000	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: 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

Example: static call

```
.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
```

Example: dynamic call

```
.class public Exp

   .method public fac(I)I

            iload 1
            ifne else

            iconst_1
            ireturn

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

Code Pattern

dynamic method call

caller

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

Code Pattern

dynamic 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

Code Pattern

return from method call

callee

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

Code Pattern

return from method call

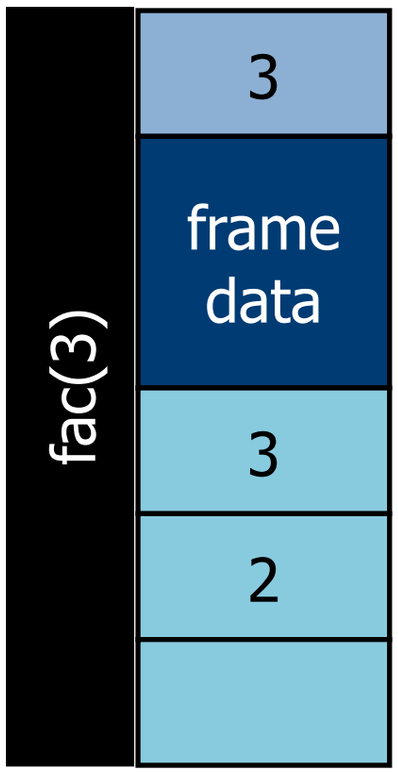
callee

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

virtual machine on return

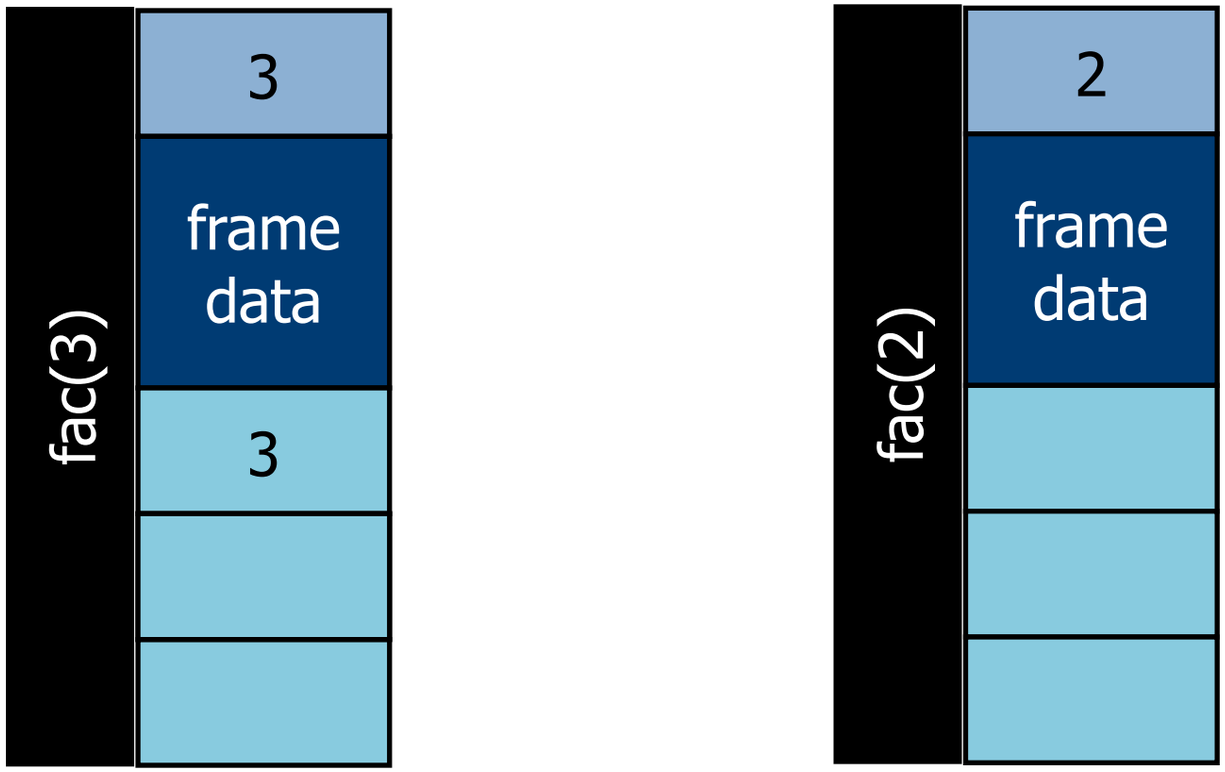
- push result on caller's operand stack
- point pc to return address
- destroy frame

Implementation
heap-based



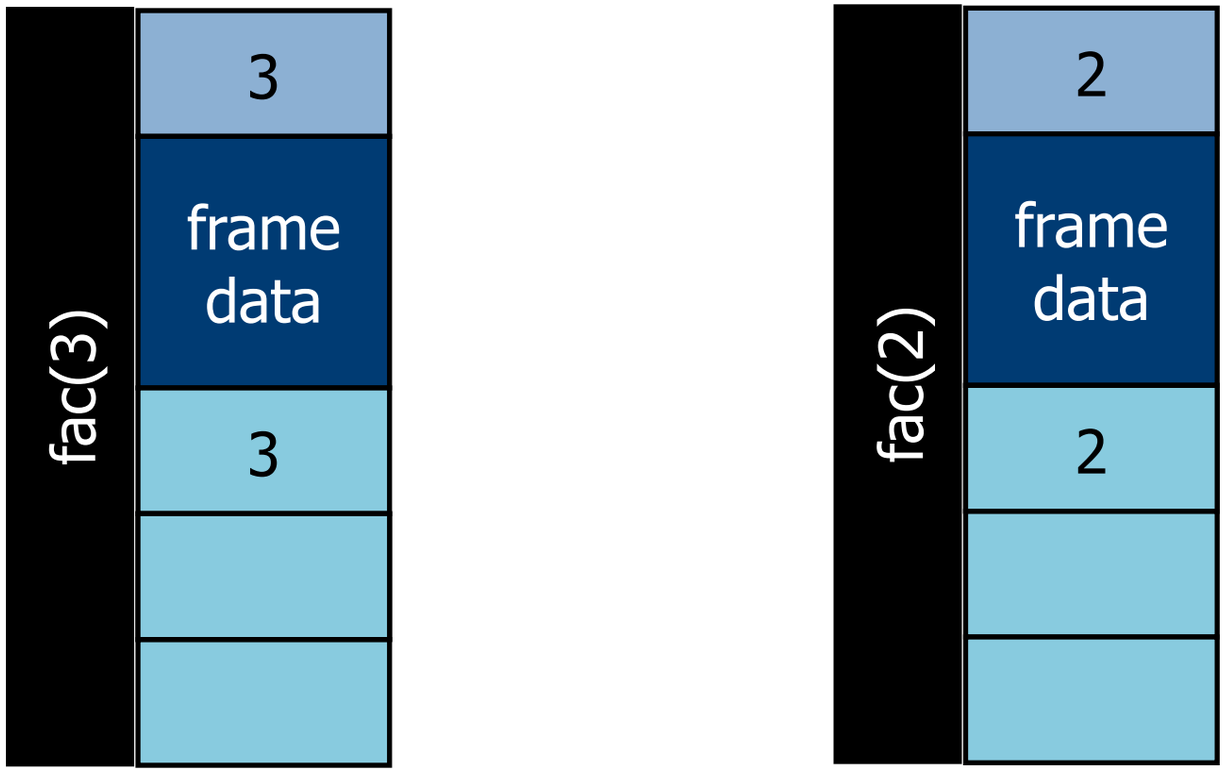
Implementation

heap-based



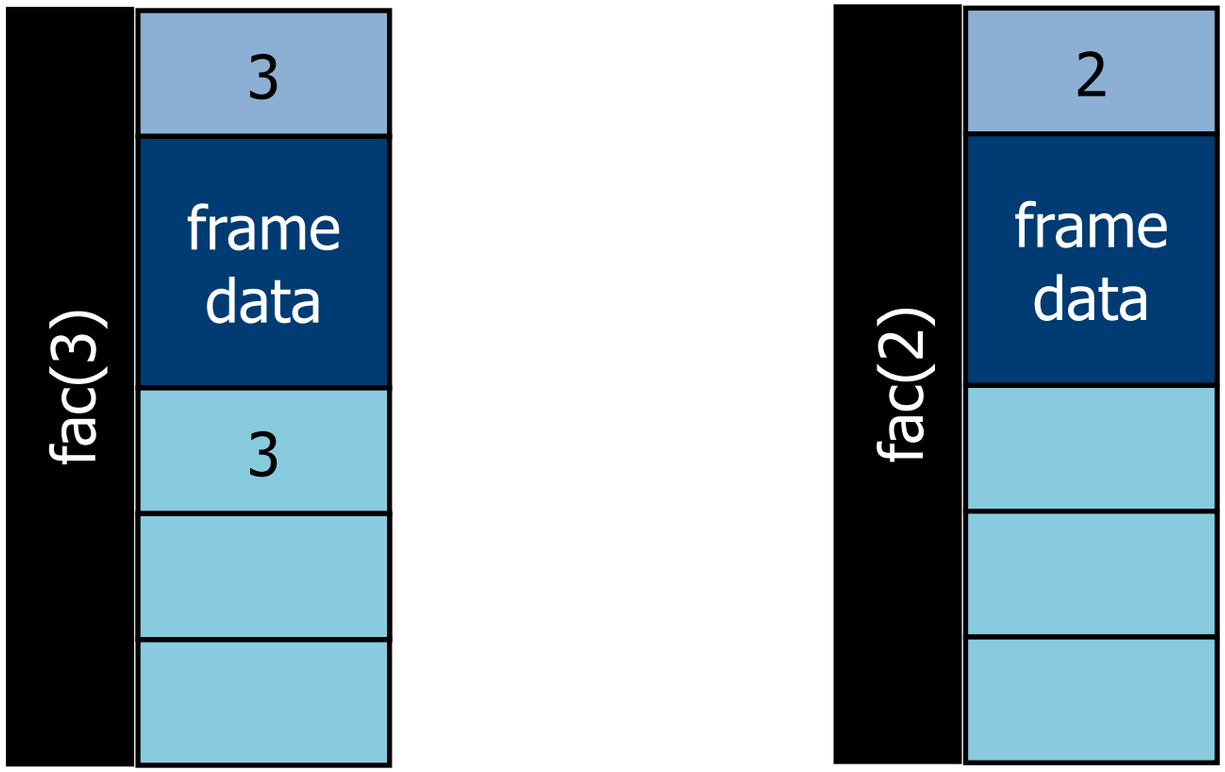
Implementation

heap-based



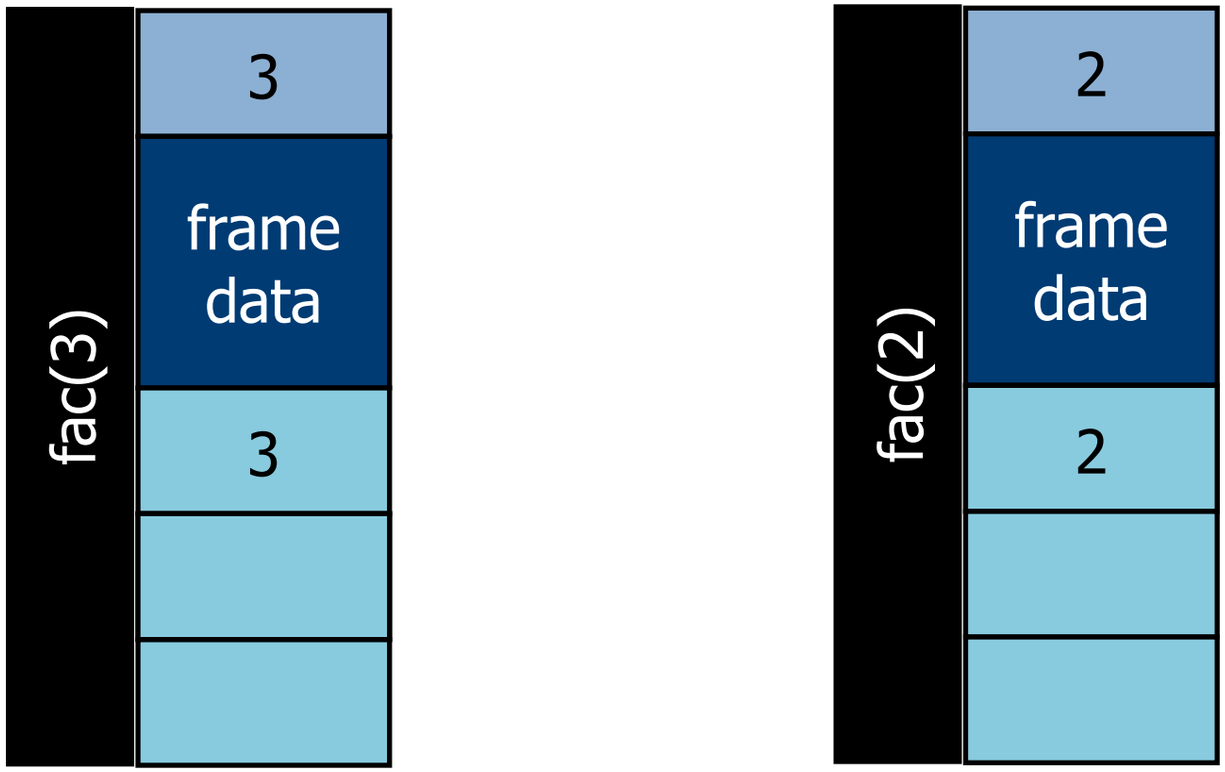
Implementation

heap-based



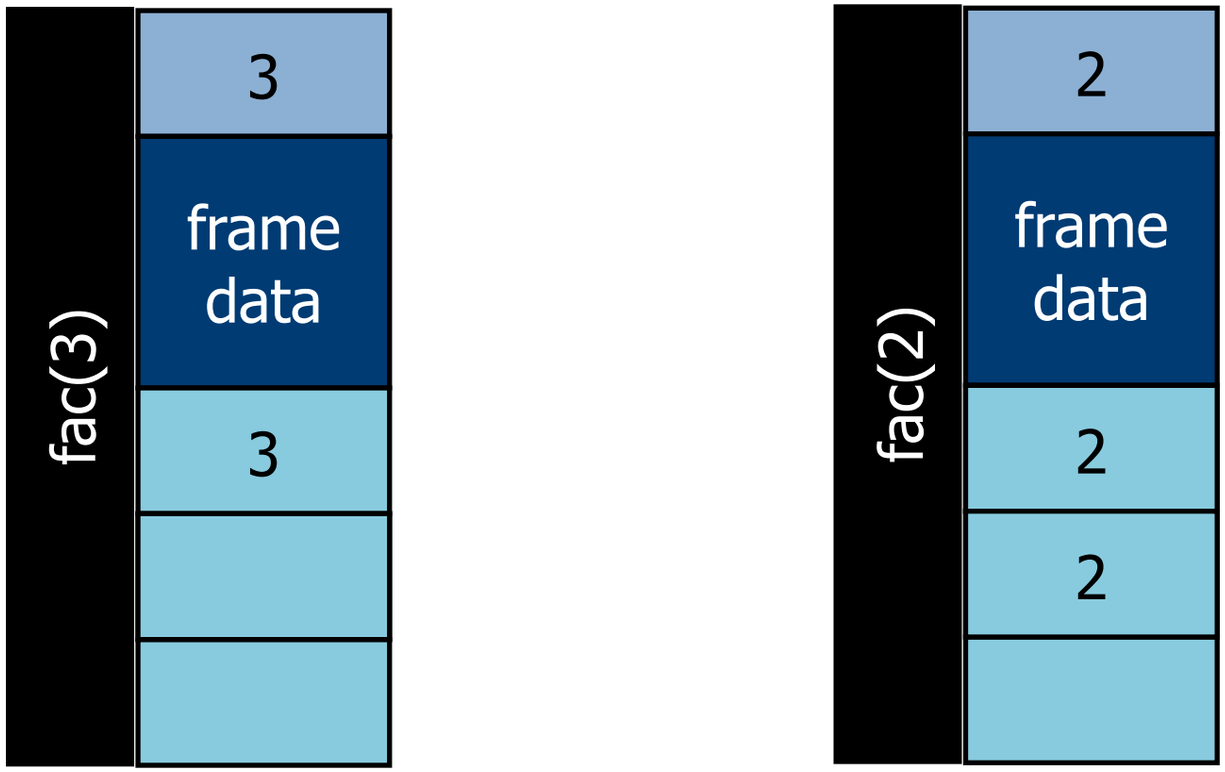
Implementation

heap-based



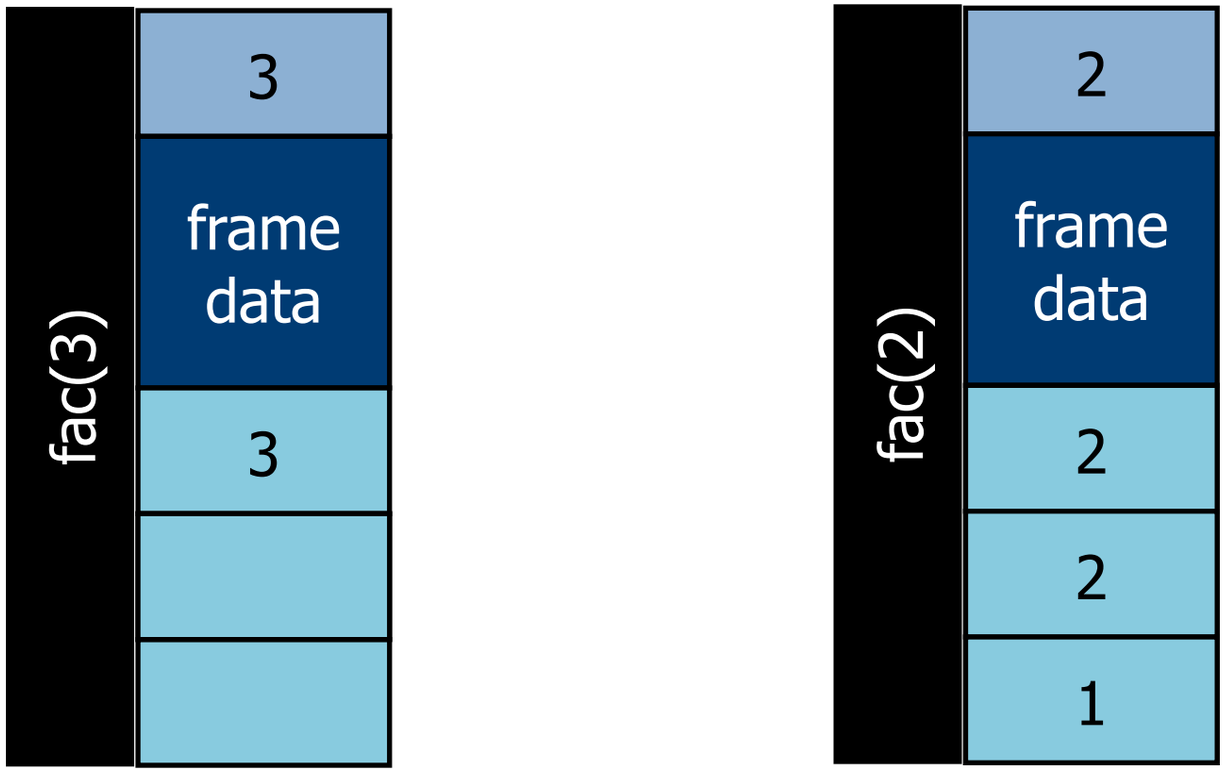
Implementation

heap-based



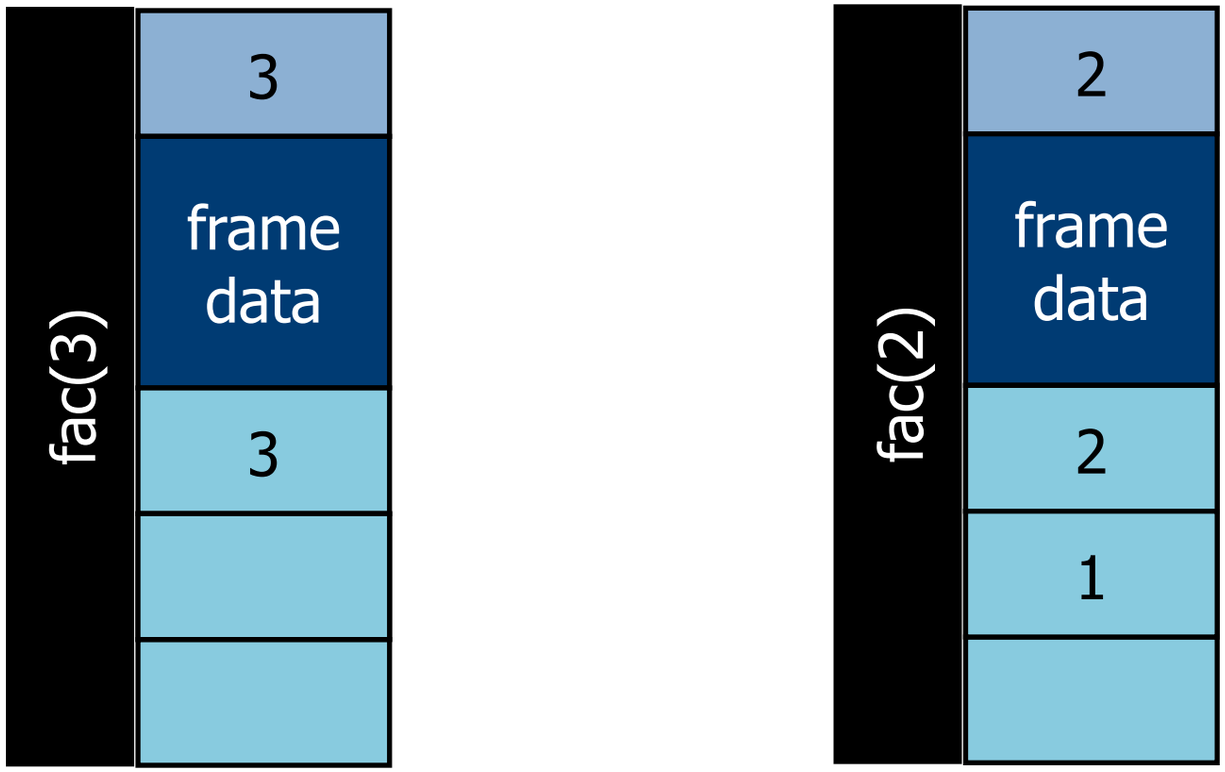
Implementation

heap-based



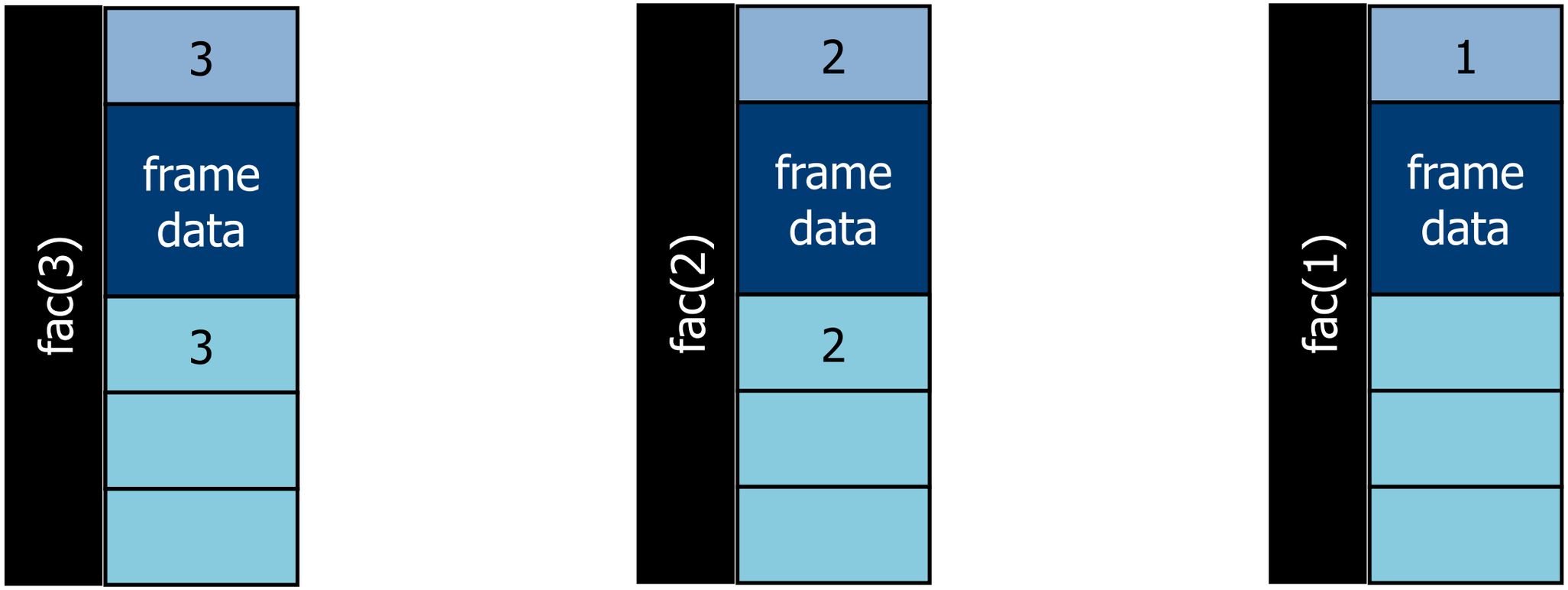
Implementation

heap-based



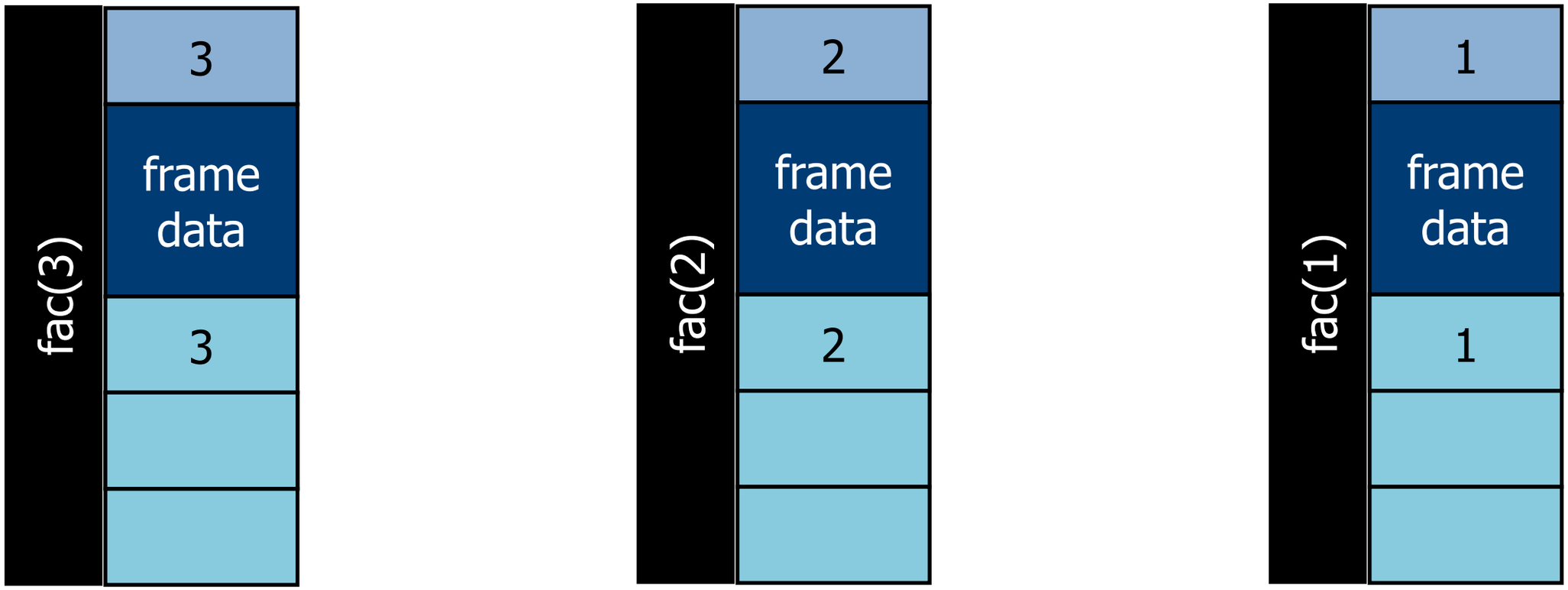
Implementation

heap-based



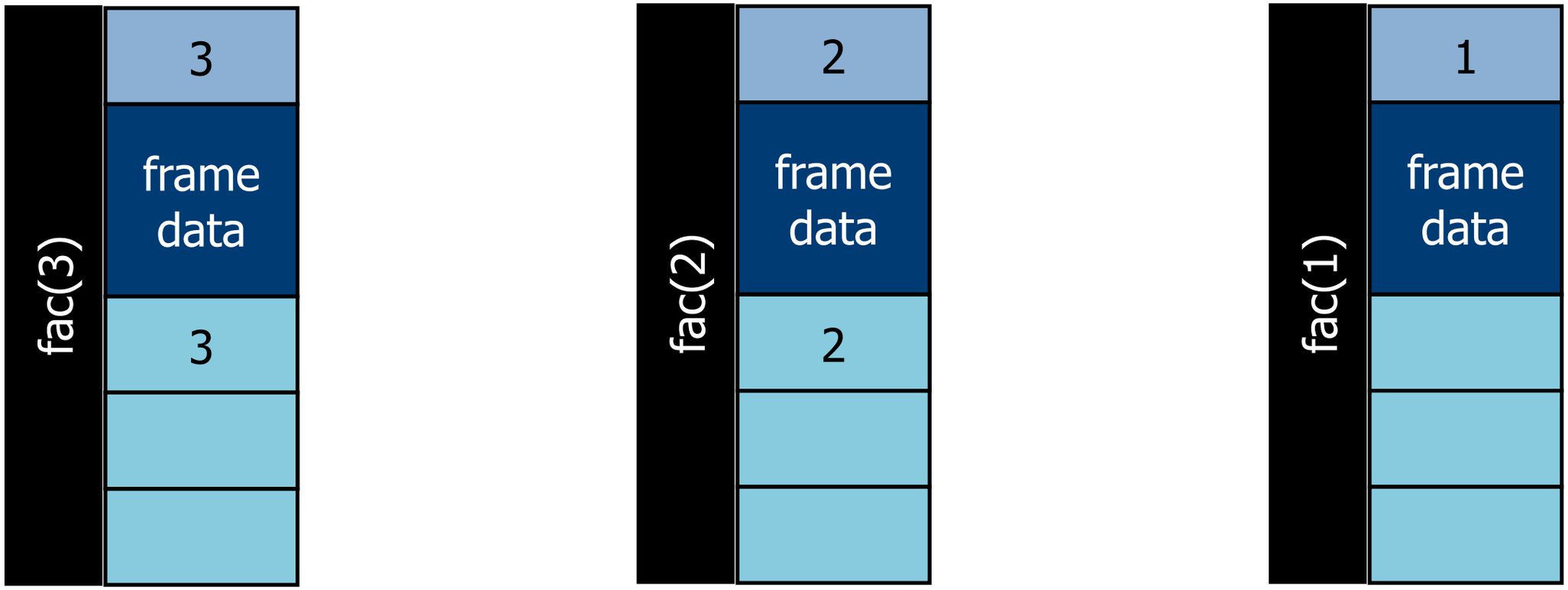
Implementation

heap-based



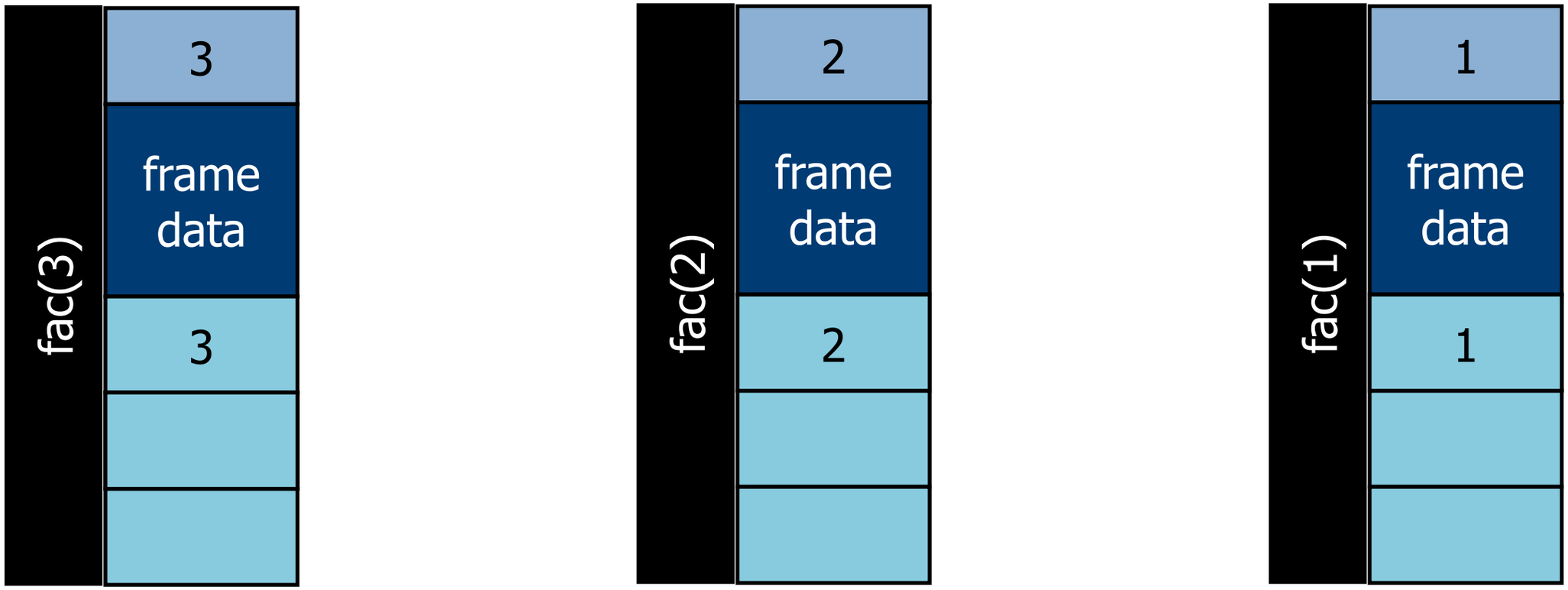
Implementation

heap-based



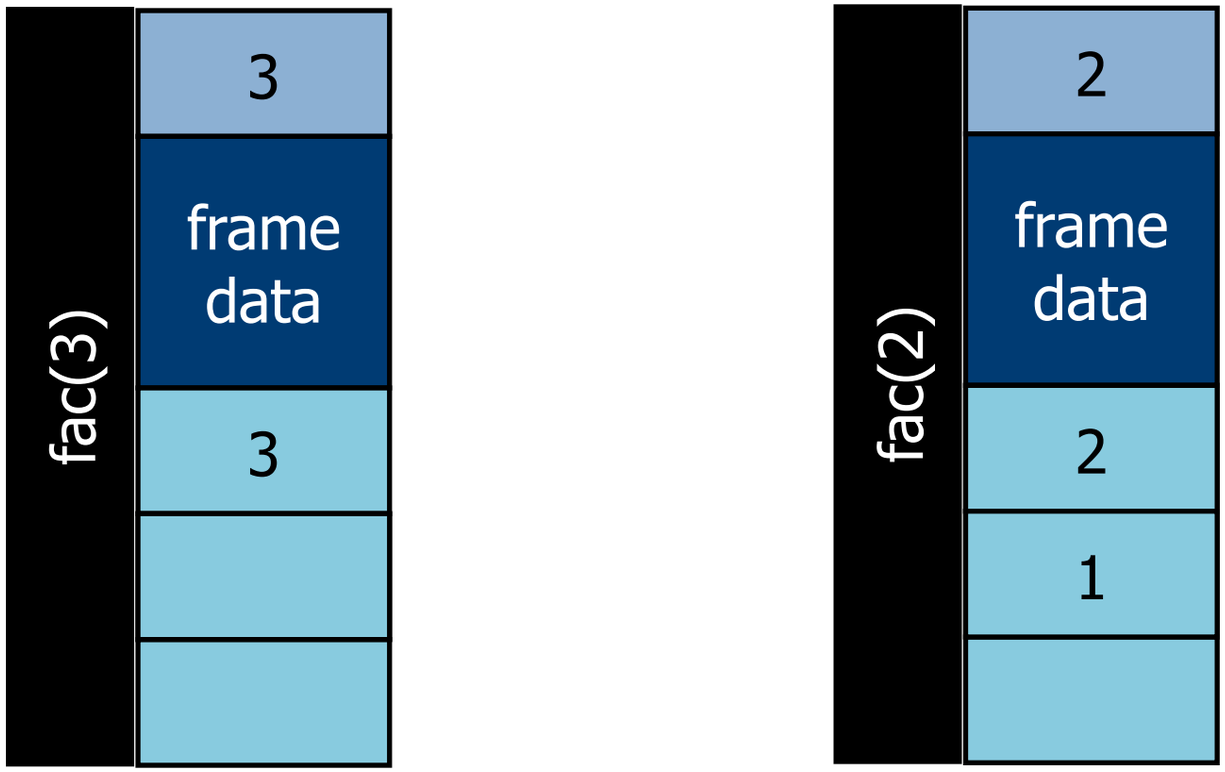
Implementation

heap-based



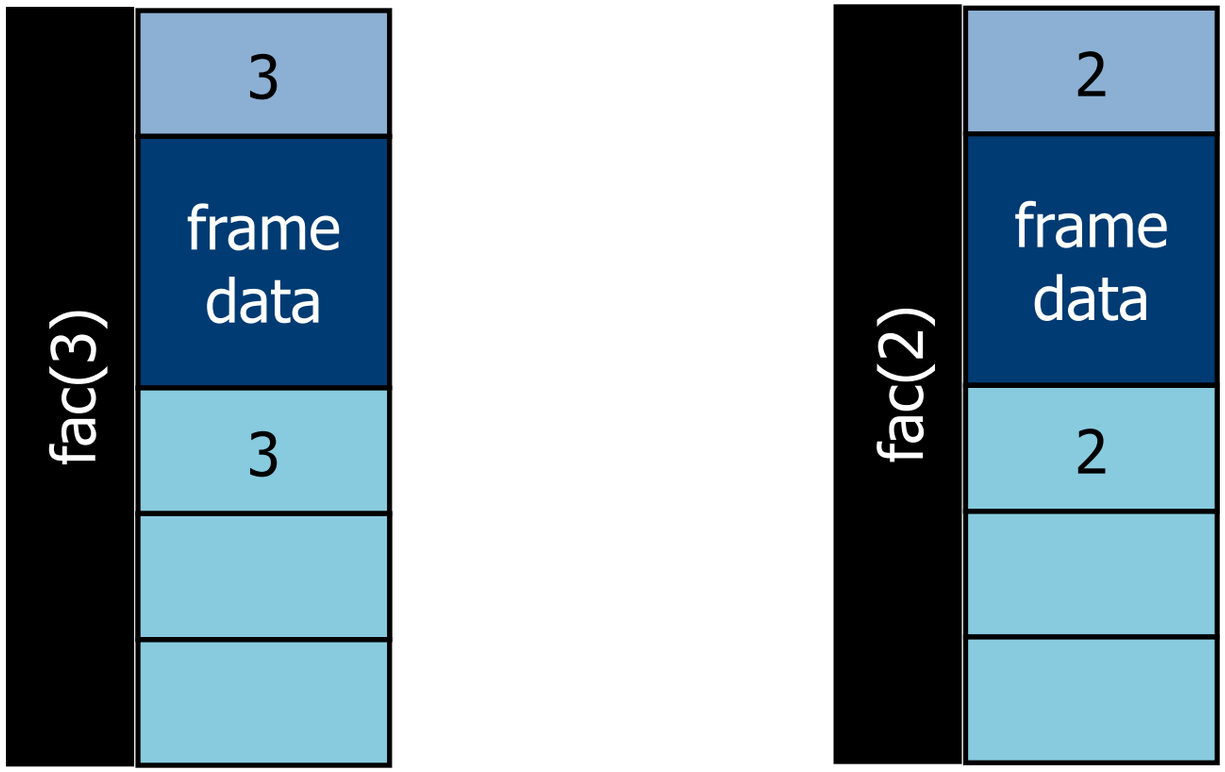
Implementation

heap-based

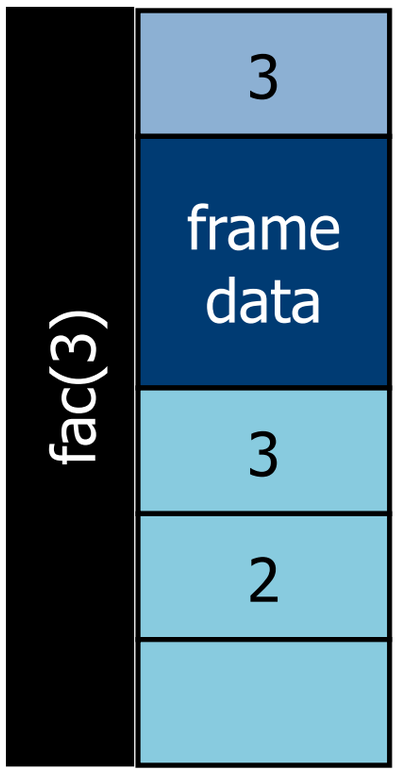


Implementation

heap-based

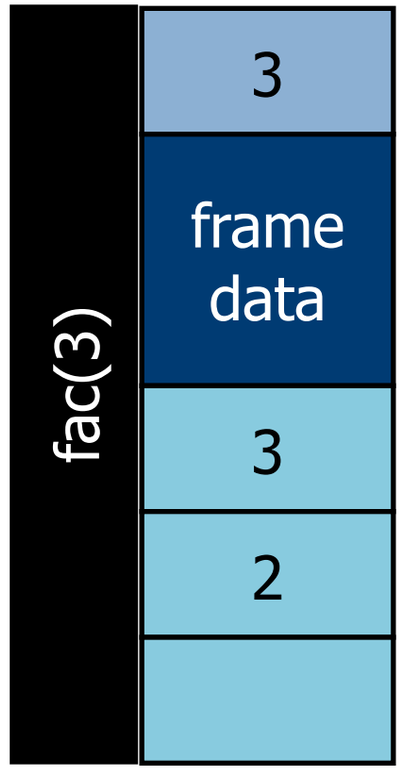


Implementation
heap-based



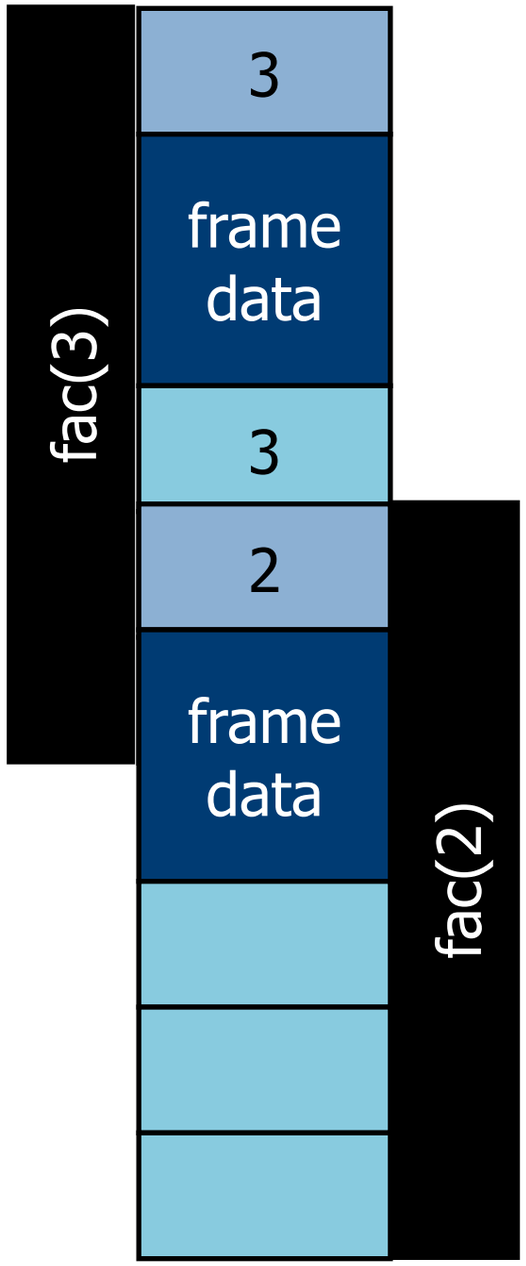
Implementation

stack-based

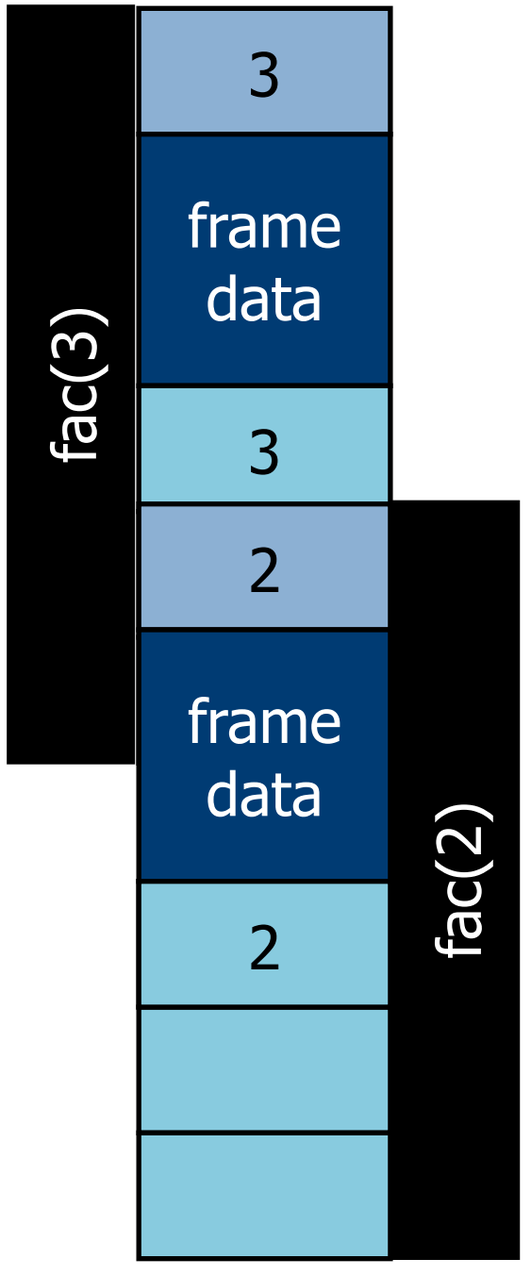


Implementation

stack-based

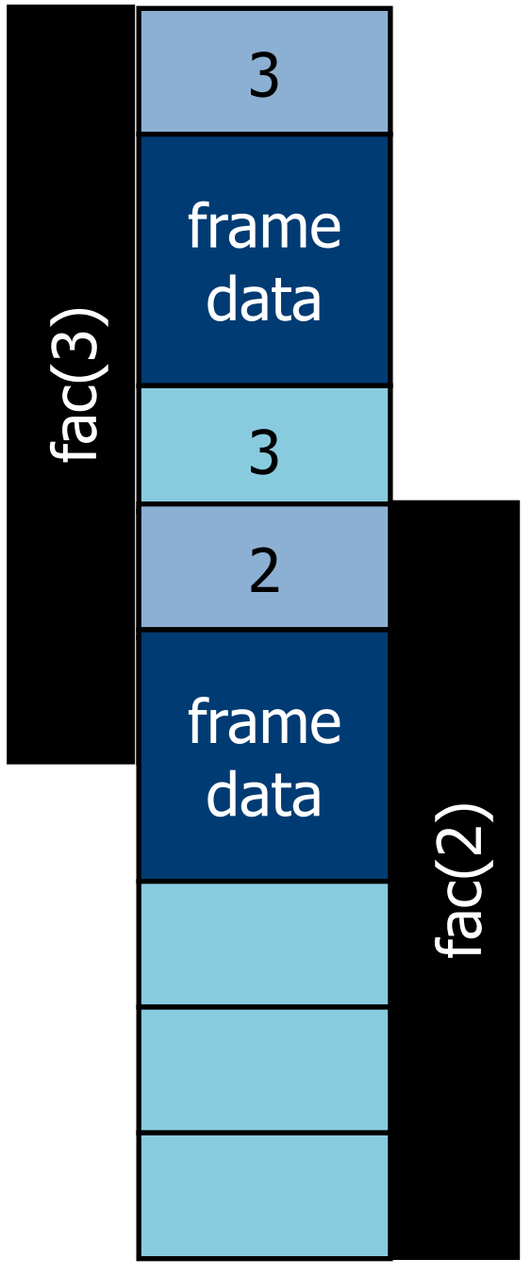


Implementation
stack-based

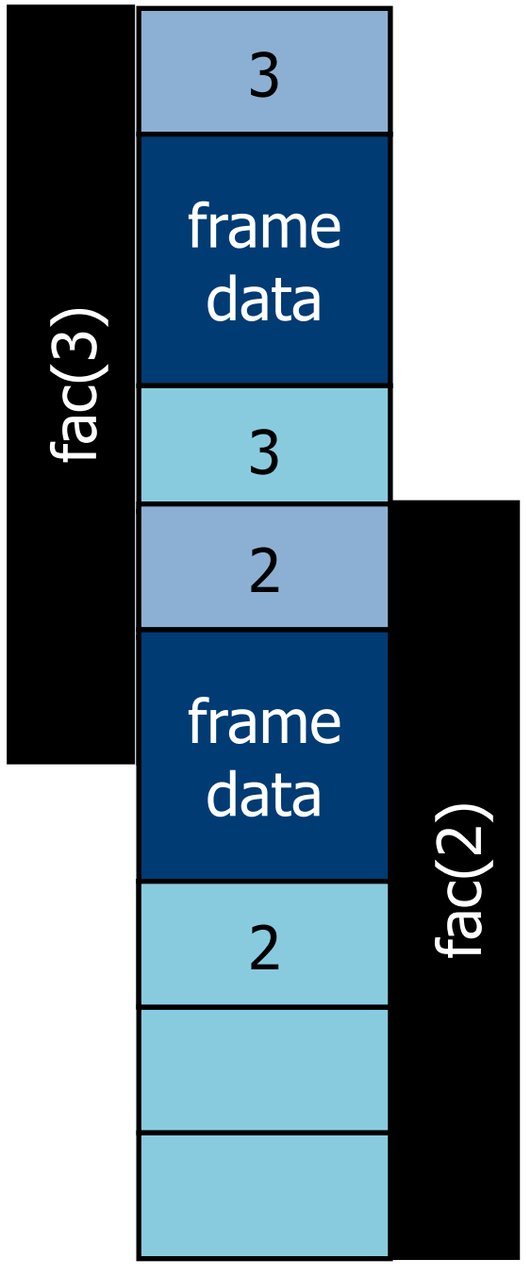


Implementation

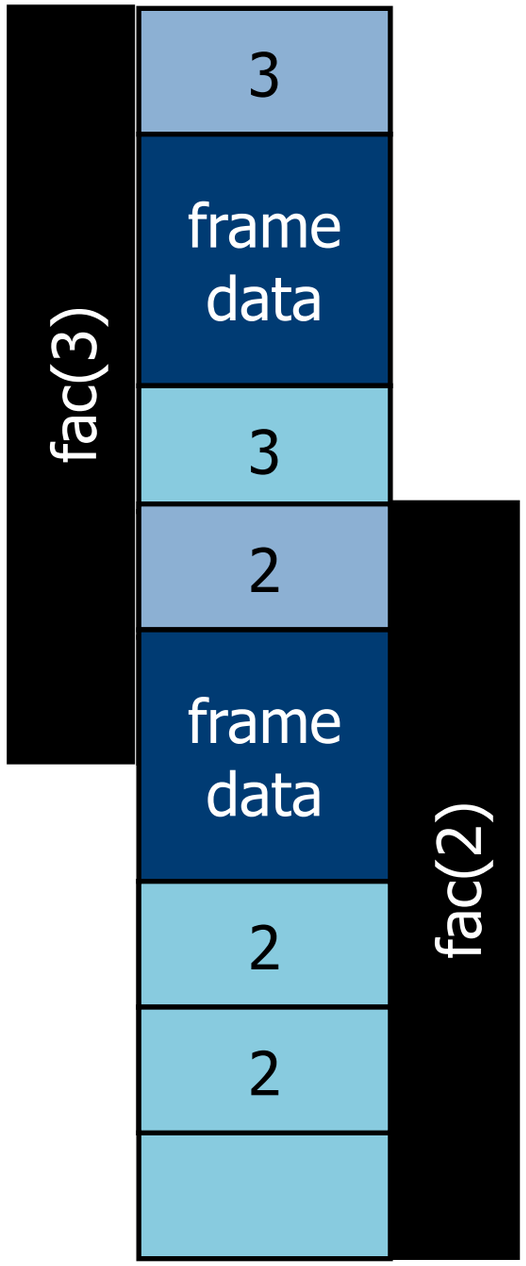
stack-based



Implementation
stack-based

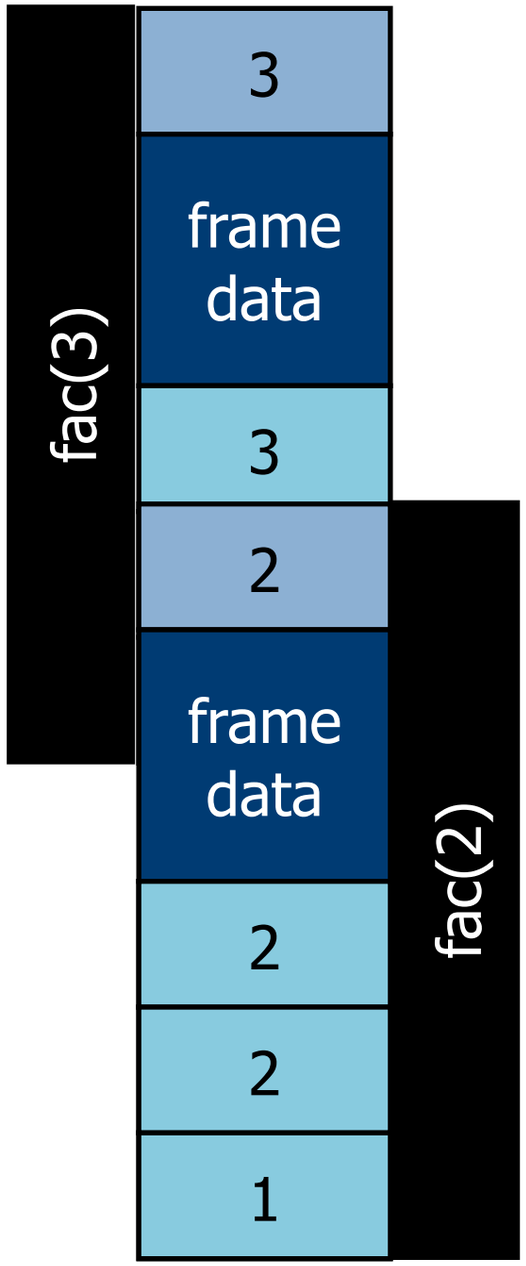


Implementation
stack-based



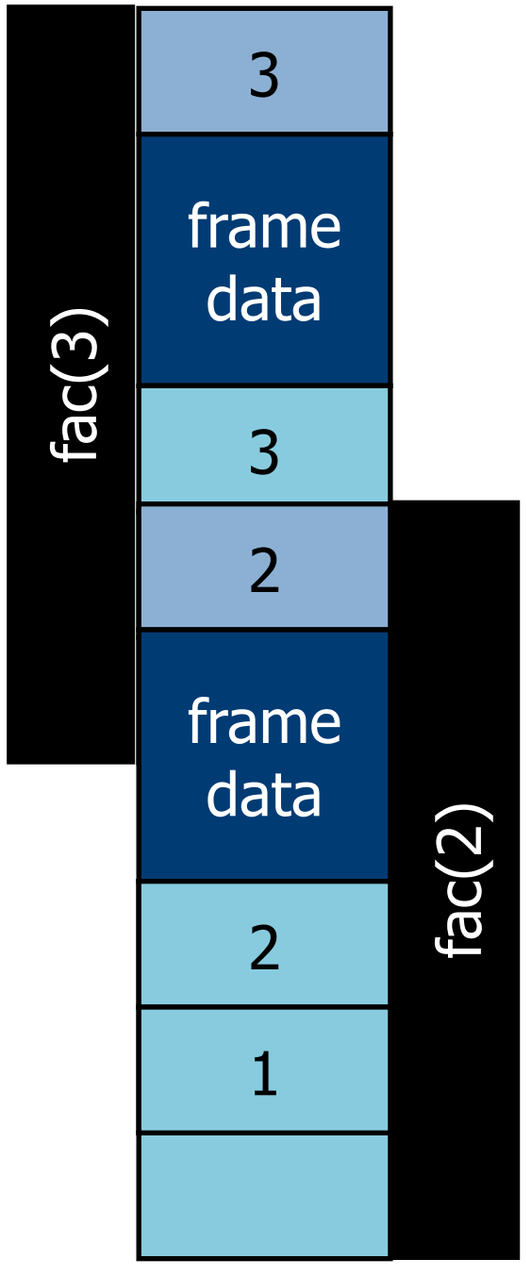
Implementation

stack-based

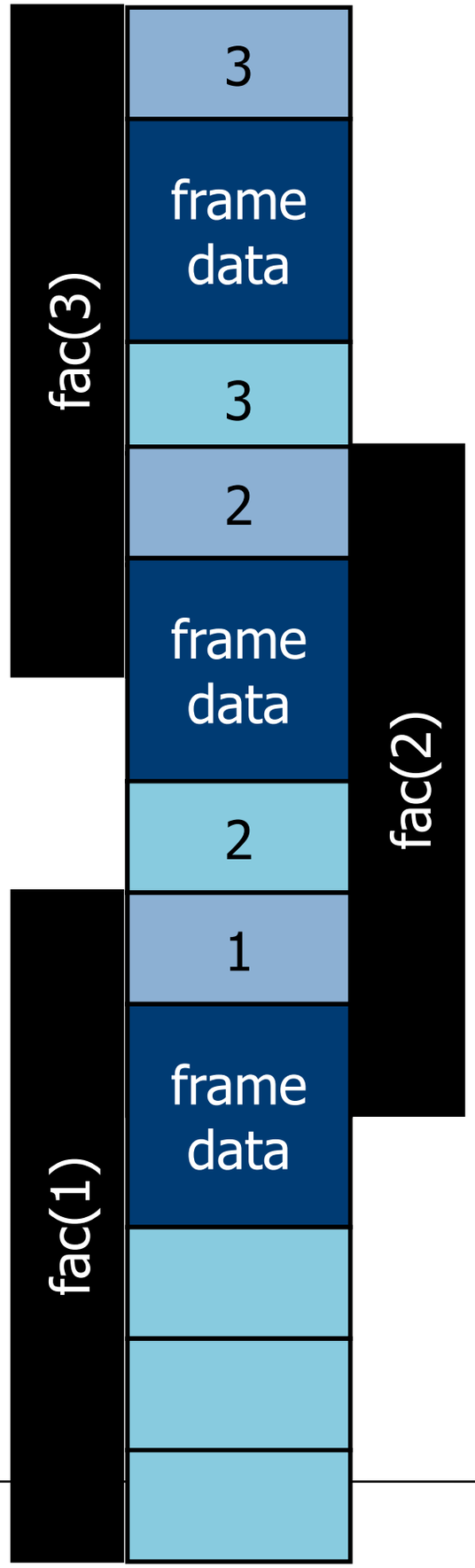


Implementation

stack-based

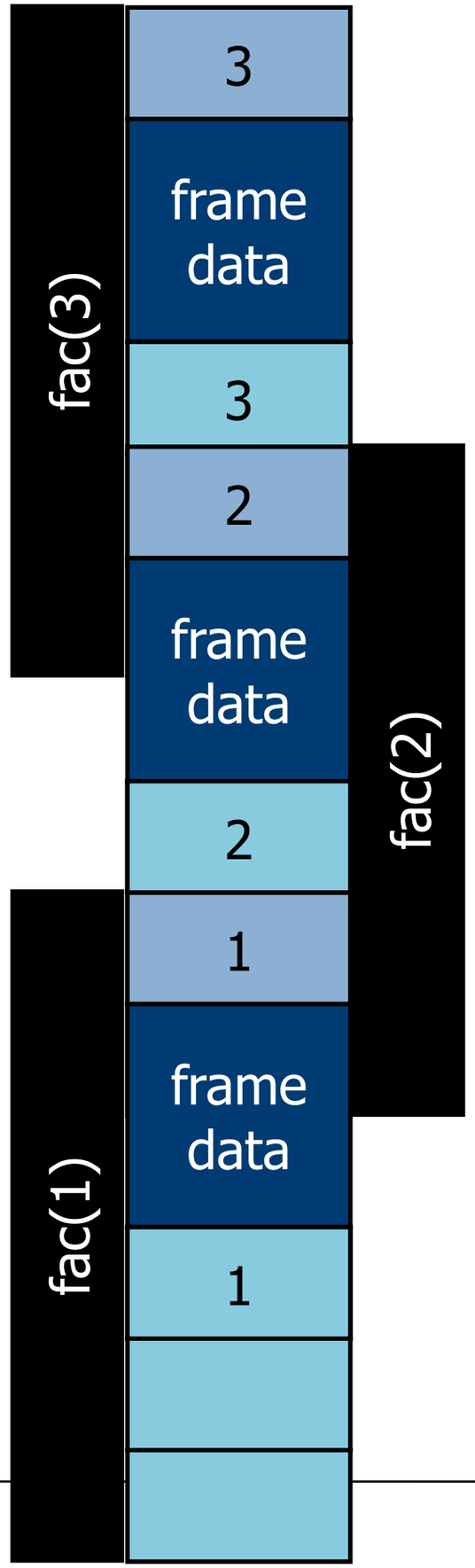


Implementation
stack-based

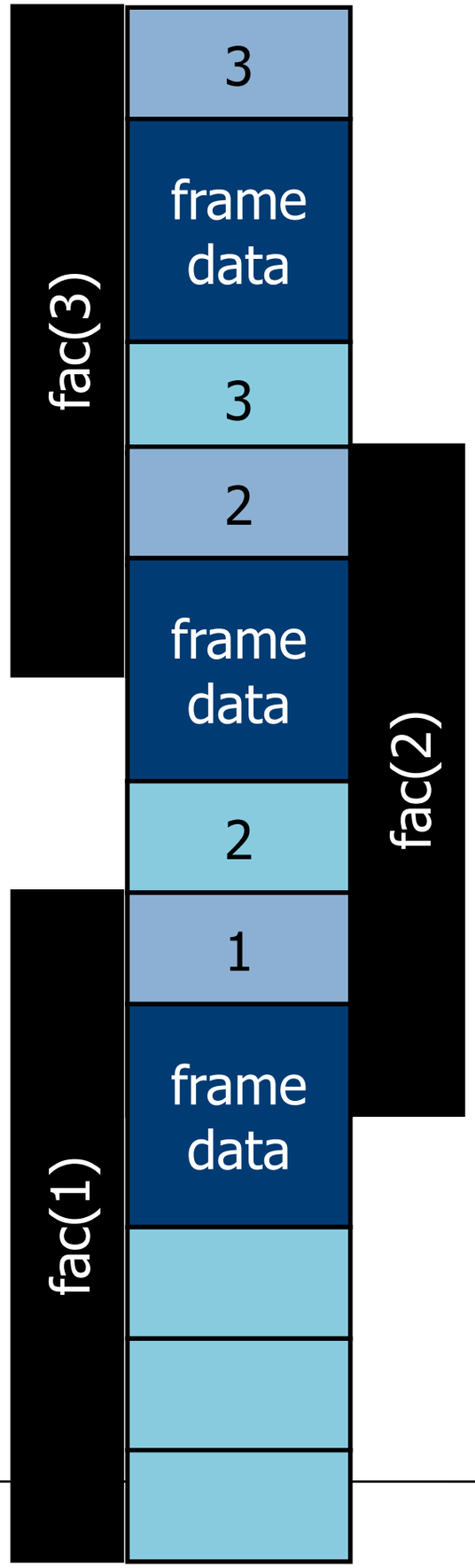


Implementation

stack-based

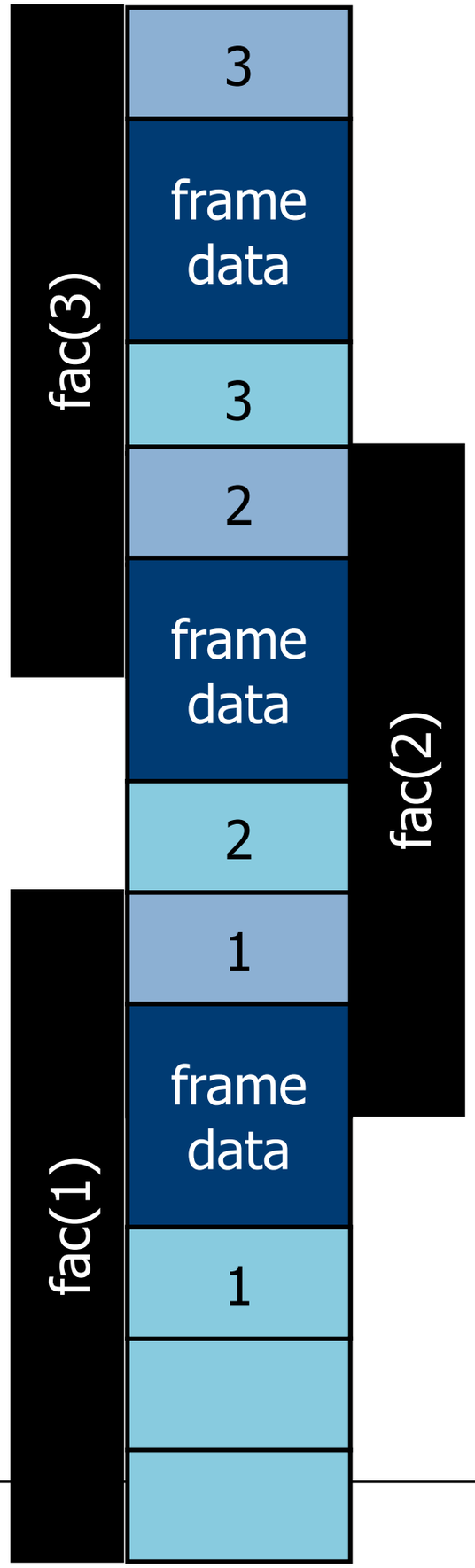


Implementation
stack-based



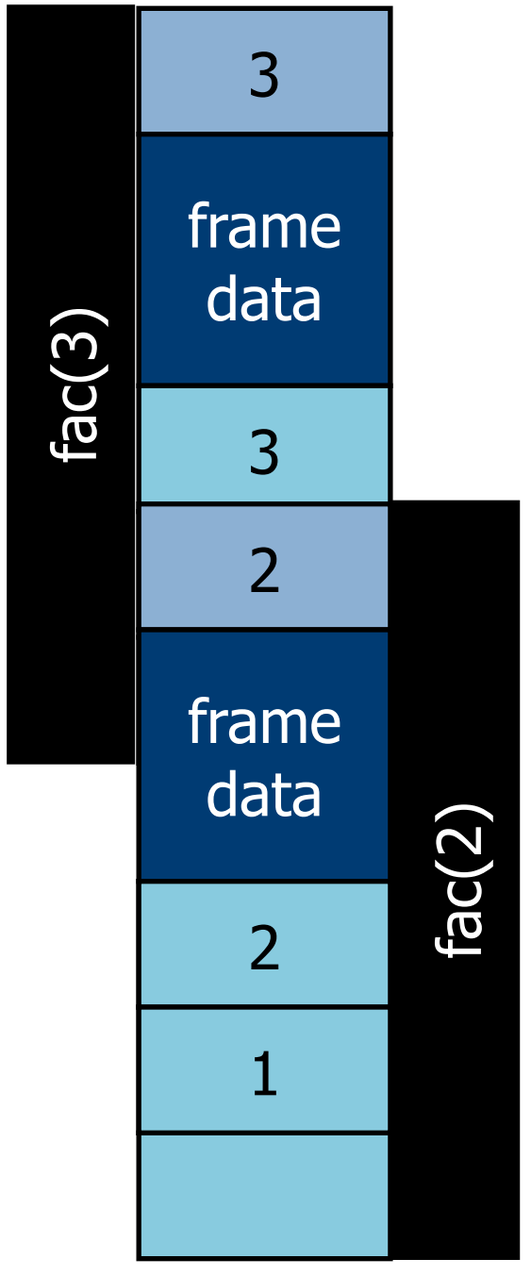
Implementation

stack-based



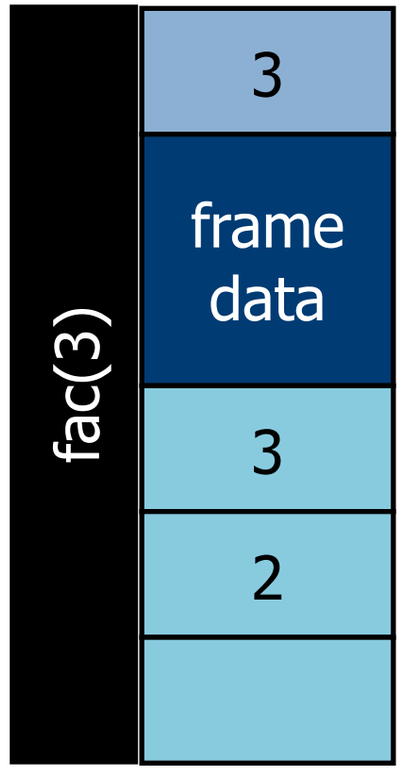
Implementation

stack-based



Implementation

stack-based



coffee break

II

register-based machines

Registers
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

Registers
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 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

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

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

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

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

Calling Conventions
FASTCALL

caller

- passes parameters in registers
- pushes additional 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

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

- passes parameters in registers
- pushes additional 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

- passes parameters in registers
- pushes additional 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
```

III

optimisations revisited

Optimisations

reasons

- code overhead
- execution overhead

Optimisations

reasons

- code overhead
- execution overhead

inlining

- replace calls by body of the procedure
- source code level

Optimisations

reasons

- code overhead
- execution overhead

inlining

- replace calls by body of the procedure
- source code level

tail recursion

- replace recursive calls by loops or jumps
- source or machine code level

Example: Tail Recursion

```
.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
```

Example: Tail Recursion

```
.class public Exp

   .method public static fac(II)I

            iload 1
            ifne else

            iload 2
            ireturn

      else: iload 1
            iconst_1
            isub
            iload 1
            iload 2
            imul
            invokestatic Exp/fac(II)I
            ireturn
   .end method
```

Example: Tail Recursion

```
.class public Exp

   .method public static fac(II)I

      strt: iload 1
            ifne else

            iload 2
            ireturn

      else: iload 1
            iconst_1
            isub
            iload 1
            iload 2
            imul
            istore 2
            istore 1
            goto strt
```

IV

summary

Summary
lessons learned

stack frames in the Java Virtual Machine

- parameter passing, returning results
- implementation strategies

Summary
lessons learned

stack frames in the Java Virtual Machine

- parameter passing, returning results
- implementation strategies

stack frames in register-based machines

- registers x86 family
- manipulating stack registers
- calling conventions

Summary
lessons learned

stack frames in the Java Virtual Machine

- parameter passing, returning results
- implementation strategies

stack frames in register-based machines

- registers x86 family
- manipulating stack registers
- calling conventions

optimisations

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.

Activation Records

Andrew W. Appel, Jens Palsberg: Modern Compiler Implementation in Java, 2nd edition. 2002

Outlook
coming next

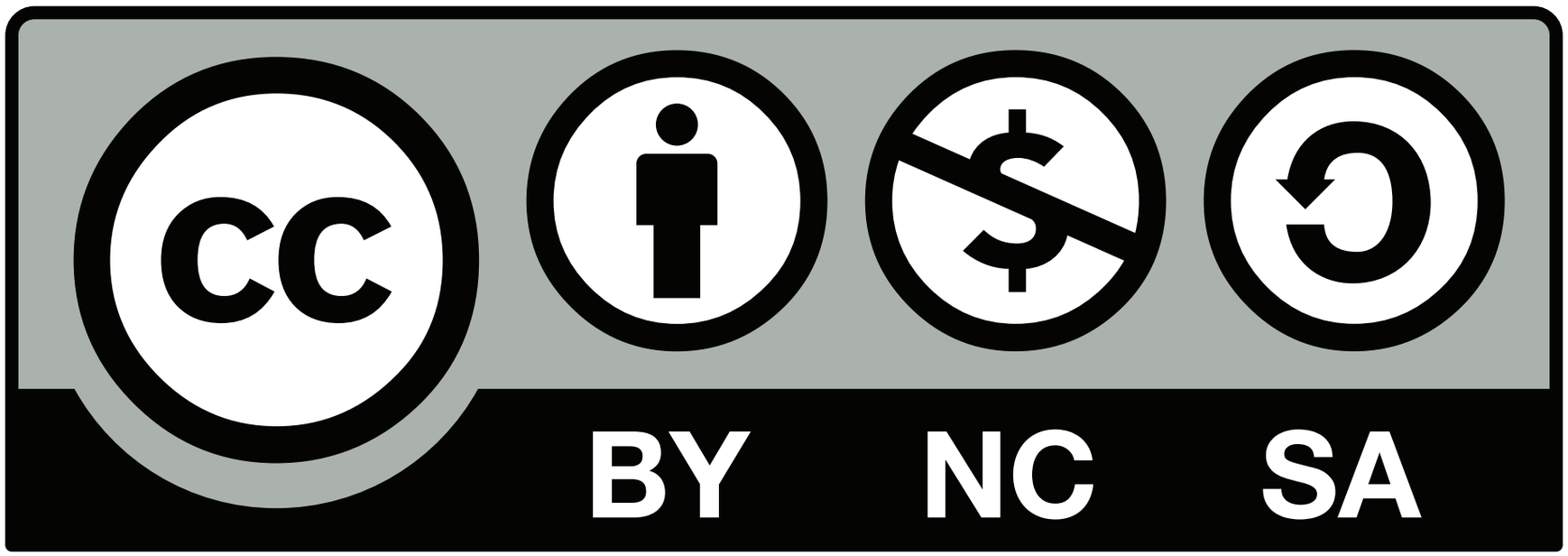
imperative and object-oriented languages

- Lecture 10: Dataflow Analysis Nov 13
- Lecture 11: Register Allocation Nov 20
- Lecture 12: Garbage Collection Nov 27

Lab Oct 25

- name analysis
- name-based errors
- type analysis for named elements
- type-based errors

copyrights & credits


BY
NC
SA

Pictures

copyrights

Slide 1:

Framed by LexnGer, some rights reserved

Slide 37:

The Lemon Tree by Dominica Williamson, some rights reserved

Slide 46:

Oude Kerk by M.M. R, some rights reserved