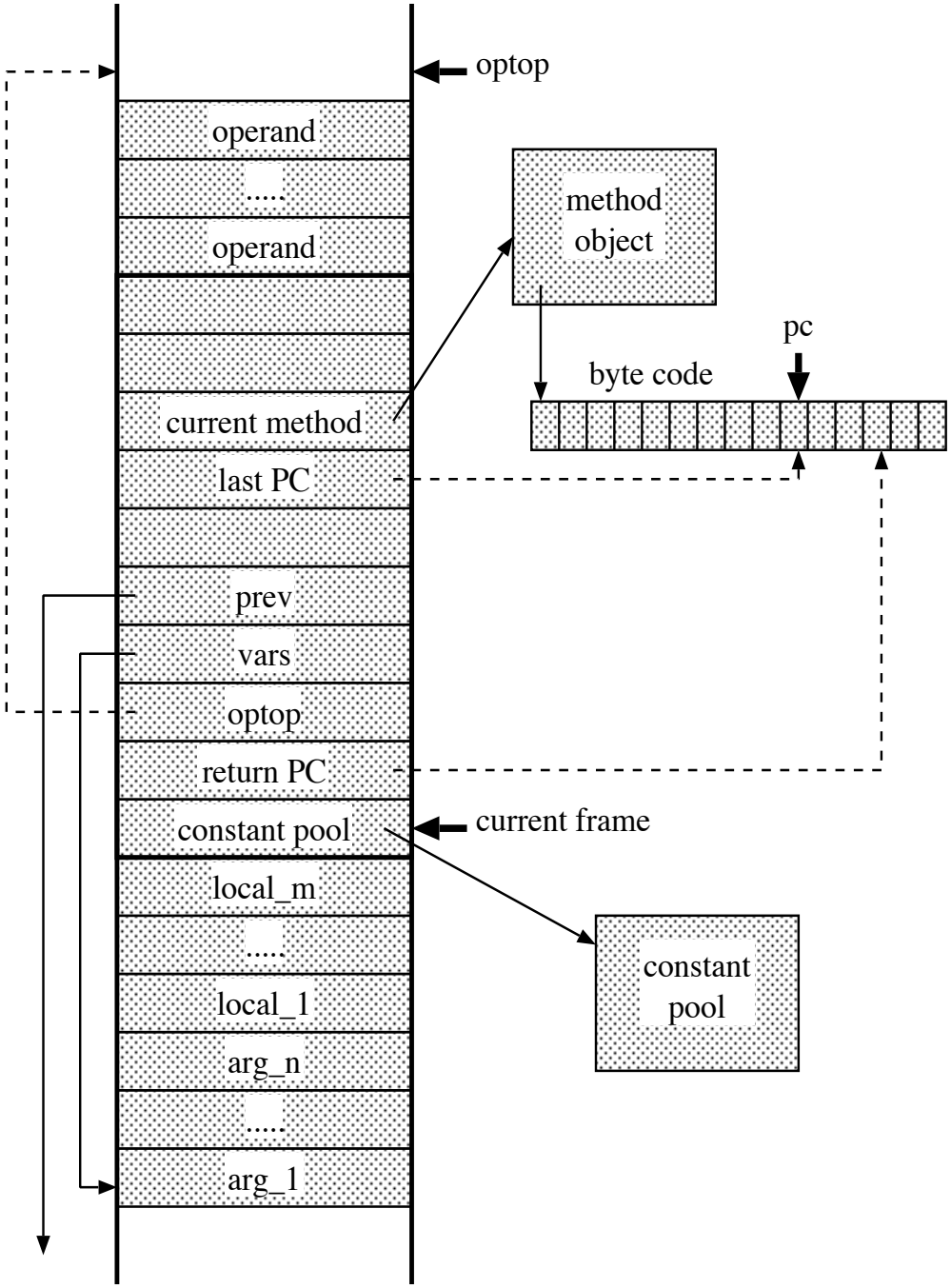
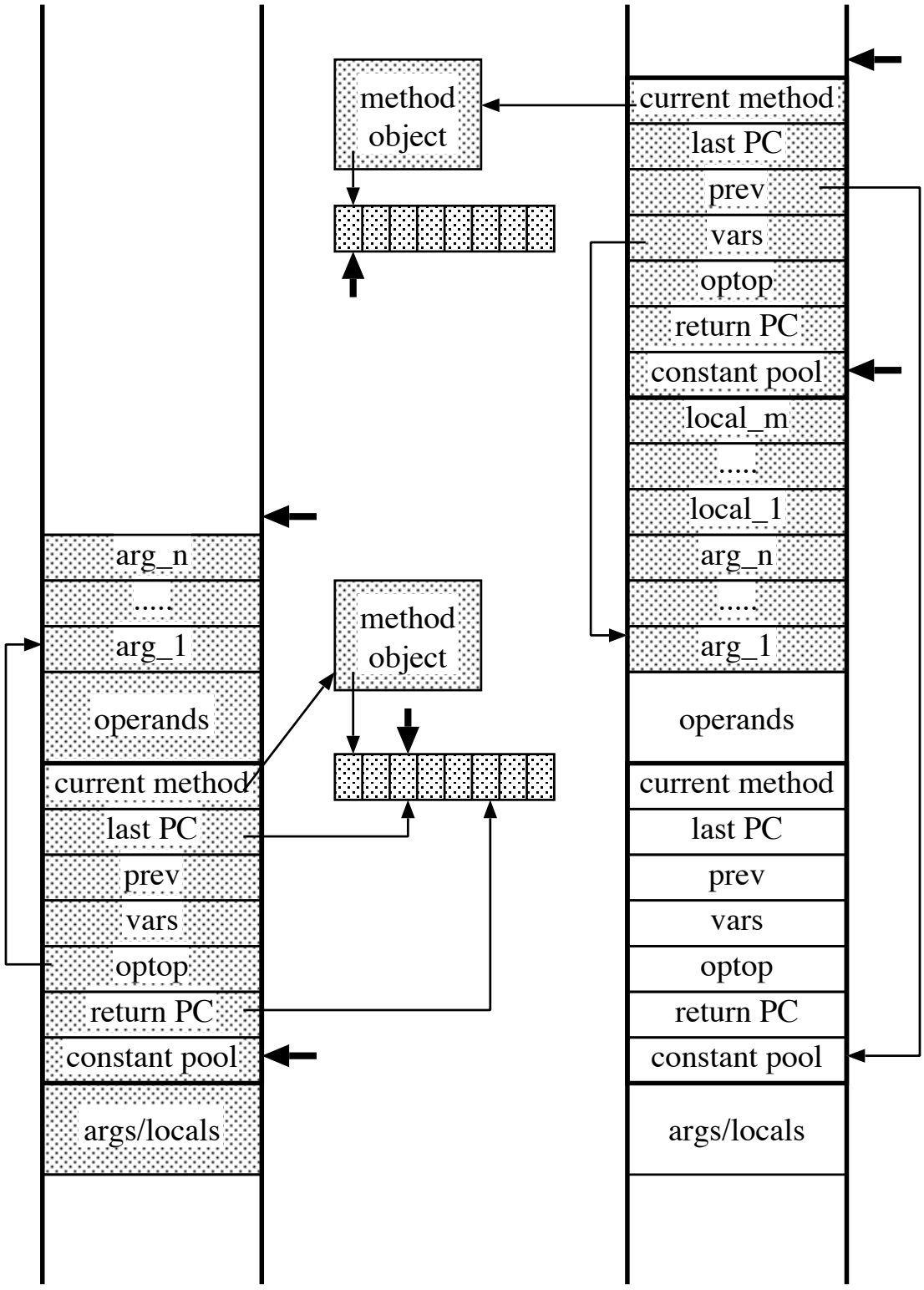
JVM Code generation

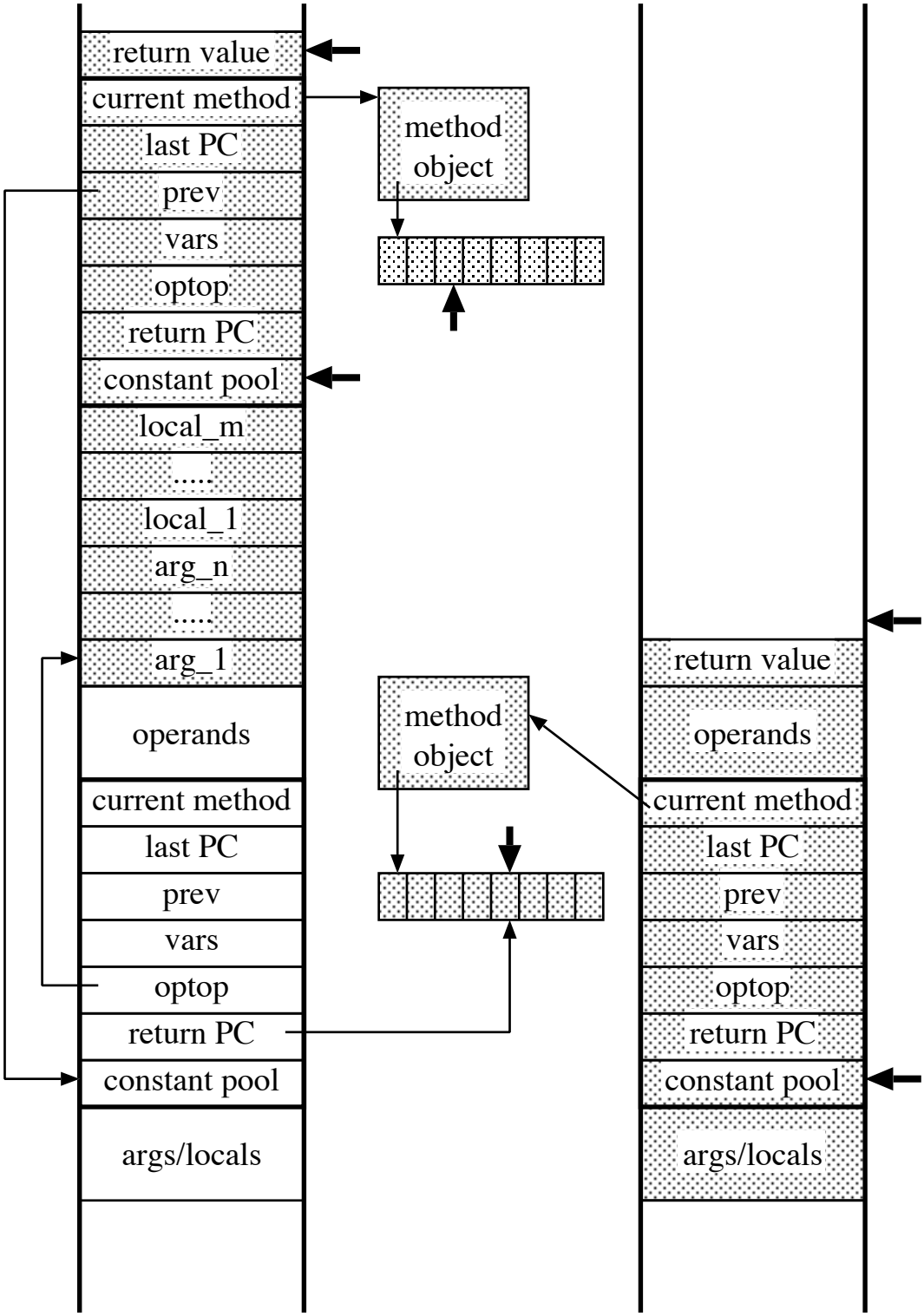
An implementation of frames (SUN JDK)



Method call


method object
current method
last PC
prev
vars
optop
return PC
constant pool
local_m
.....
local_1
arg_n
.....
arg_1
operands
current method
last PC
prev
vars
optop
return PC
constant pool
args/locals
method object
arg_n
.....
arg_1
operands
current method
last PC
prev
vars
optop
return PC
constant pool
args/locals

Return from method



JVM main loop (in C)

```
ExecuteJava() {
  for (;;) {
    switch (*pc) {

    case 0 /* nop */:
      pc++;
      break;

    case 1 /* aconst_null */:
      *optop++ = NULL;
      pc++;
      break;

    case 2 /* iconst_m1 */:
    ...

    case 201 /* jsr_w */:
      offset = pc[1]<<24 | pc[2]<<16
               | pc[3]<<8 | pc[4];
      *optop++ = pc + 5;
      pc += offset;
      break;

    case 254 /* impdep1 */
      ...
    case 255 /* impdep2 */
      ...
    }
  }
}
```

Instructions for method invocation

```
ExecuteJava() {
  for (;;) {
    switch (*pc) {

    case 0 /* nop */:
    ...
    case 184 /* invokestatic */:
      Solve constant pool info pc+1 & pc+2 represent;
      Load & initialize class if necessary;
      Update pc[1] & pc[2];
      *pc = 254;
      break;

    case 185 /* invokeinterface */:
    ...
    case 254 /* invokestatic_quick */:
      Read the initialized method info from the constant
      pool pc+1 & pc+2 represent;
      c_frame->optop = optop - Number of arguments;
      c_frame->lastPC = pc;
      c_frame->returnPC = pc + 3;
      Push new frame of the method to be called;
      pc = start address of the method to be called;
      break;

    case 255 /* impdep2 */:
    ...
    }
  }
}
```

Return instruction

```
ExecuteJava() {
  for (;;) {
    switch (*pc) {
    case 0 /* nop */:
    ...

    case 172 /* ireturn */:

      int val = *--optop;
      c_frame = c_frame->prev;
      optop = c_frame->optop;
      *optop++ = val;
      pc = c_frame->returnPC;
      break;

    case 173 /* lreturn */:
    ...
    }
  }
}
```

Receiving arguments

Example (instance method):

```
 int addTwo(int i, int j) {
     return i + j;
 }

 Method int addtwo(int,int)
    0 iload_1
    1 iload_2
    2 iadd
    3 ireturn
```

Example (static method):

```
 static int addTwoStatic(int i, int j) {
     return i + j;
 }

 Method int addTwoStatic(int,int)
    0 iload_0
    1 iload_1
    2 iadd
    3 ireturn
```

Code of static-method call expression $f(e_1, e_2, \ldots, e_n)$

```
Evaluate e1, then push result.
Evaluate e2, then push result.
...
Evaluate en, then push result.
invokestatic c.f(t1...tn)t

Example: Code of f(1, g(2,3), 4),
  0 iconst_1   // First argument to f
  1 iconst_2   // First argument to g
  2 iconst_3   // Second argument to g
  3 invokestatic cg.g(II)I
  6 iconst_4   // Third argument to f
  7 invokestatic cf.f(III)I

Example:
  int add12and13()  {
    return addTwoStatic(12,13);
  }

  Method int add12and13()
   0 bipush 12
   2 bipush 13
   4 invokestatic c.addTwoStatic(II)I
   7 ireturn
```

Code of instance-method call expression $x.f(e_1, \ldots, e_n)$

Evaluate x, then push result.
Evaluate e_1, then push result.
…
Evaluate e_n, then push result.
invokevirtual $c.f(t_1 \ldots t_n)t$

Example:

```
int add12and13() {
     return this.addTwo(12,13);
}

Method int add12and13()
  0 aload_0
  1 bipush 12
  3 bipush 13
  5 invokevirtual c.addTwo(II)I
  8 ireturn
```

Example (super and private)

```
class Near {
  int it;
  public int getItNear() {
    return getIt();
  }
  private int getIt() {
    return it;
  }
}
class Far extends Near {
  int getItFar() {
    return super.getItNear();
  }
}

Method int getItNear()
  0 aload_0
  1 invokespecial Near.getIt()I
  4 ireturn

Method int getIt()
  0 aload_0
  1 getfield Near.it I
  4 ireturn

Method int getItFar()
  0 aload_0
  1 invokespecial Near.getItNear()I
  4 ireturn
```

Code generation of statement

Code of if statement

if (e) s_1 else s_2

Evaluate e; if the result is false, jump to L_1
Execute s_1
Jump to L_2
L_1: Execute s_2
L_2:

Example:

```
int lessThan100(double d) {
  if (d < 100.0)
    return 1;
  else
    return -1;
}
```

Method int lessThan100(double)

```
 0 dload_1
 1 ldc2_w #4  // #4 = 100.0
 4 dcmpg
 5 ifge 10    // if (d>=100.0)
 8 iconst_1
 9 ireturn
10 iconst_m1
11 ireturn
```

Code of while statement

Code of while statement

while (*e*) *s*

Jump to L_2

L_1: Execute *s*

L_2: Evaluate *e*; if the result is true, jump to L_1

Example:

```
void whileInt() {
  int i = 0;
  while (i < 100)
    i++;
}
```

```
Method void whileInt()
 0 iconst_0
 1 istore_1
 2 goto 8
 5 iinc 1 1
 8 iload_1
 9 bipush 100
11 if_icmplt 5
14 return
```

```
Comparison
 0 iconst_0
 1 istore_1
 2 iload_1
 3 bipush 100
 5 if_icmpge 14
 8 iinc 1 1
11 goto 2
14 return
```

Code of assignment expression

Assignment expression to variable *v*

```
  v = e
```

* when discarding the value

```
    Evaluate e (the result in optop)
    istore_v
```

* when using the value later

```
    Evaluate e (the result in optop)
    dup
    istore_v
```

Example: x = y = z+1;

```
  0 iload_z
  1 iconst_1
  2 iadd
  3 dup
  4 istore_y
  5 istore_x
```

Code of arithmetic expression

Arithmetic expression $e_1 \bigcirc e_2$

  Evaluate e_1 (the result in optop)

  Evaluate e_2 (the result in optop)

  inst

Example: x+y

```
  iload_x
  iload_y
  iadd
```

Example: a*b+y

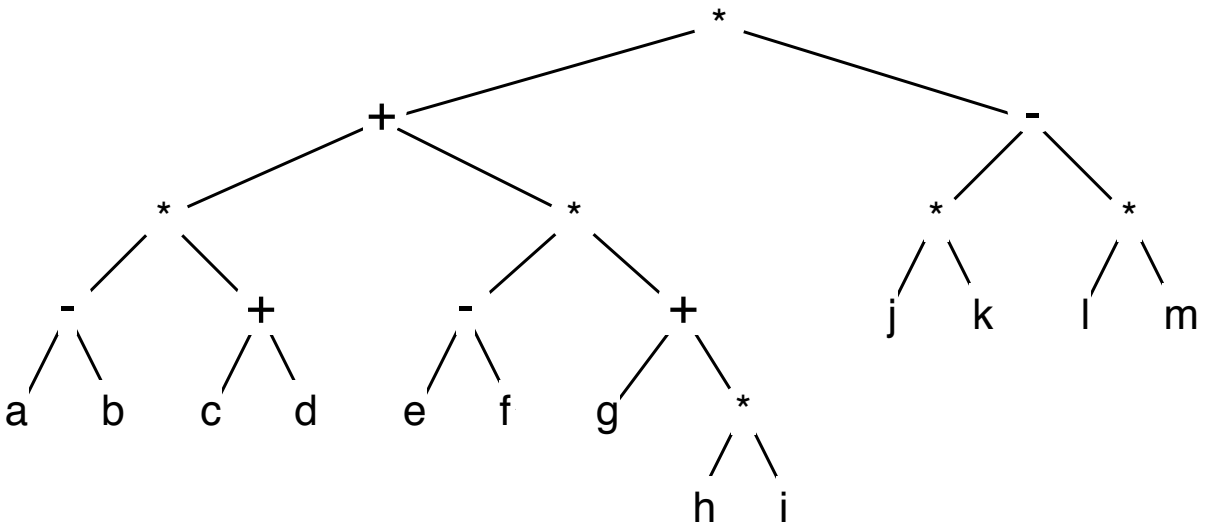
```
  iload_a
  iload_b
  imul
  iload_y
  iadd
```

Example: a*b+x*y

```
  iload_a
  iload_b
  imul
  iload_x
  iload_y
  imul
  iadd
```

Example

```
((a-b)*(c+d)+(e-f)*(g+h*i))*(j*k-1*m)
```




```
iload_a    iload_e    iadd      iload_m
iload_b    iload_f    imul      imul
isub       isub       iadd      isub
iload_c    iload_g    iload_j   imul
iload_d    iload_h    iload_k
iadd       iload_i    imul
imul       imul       iload_l
```

Example:

```
  int align2grain(int i, int grain) {
    return ((i+grain-1) & ~(grain-1));
  }

  Method int align2grain(int,int)
   0 iload_1
   1 iload_2
   2 iadd
   3 iconst_1
   4 isub
   5 iload_2
   6 iconst_1
   7 isub
   8 iconst_m1
   9 ixor
  10 iand
  11 ireturn
```

Code generation of conditional jump

"Evaluate *e*; if the result is false, jump to *L*"

```
  Evaluate e (the result in optop)
  ifeq L; // Jump if the value in optop is 0
```

"Evaluate *e*; if the result is true, jump to *L*"

```
  Evaluate e (the result in optop)
  ifne L; // Jump if the value in optop is non-0
```

Example: if (f()) x = 10;

```
  invokestatic c.f()Z
  ifeq L
  bipush 10
  istore_x
L:
```

"If e_1 >= e_2, then jump to *L*"

```
  Evaluate e1 (the result in optop)
  Evaluate e2 (the result in optop)
  if_icmpge L
```

Calculation code of return value

```
return e;

  Evaluate e (the result in optop)
  ireturn

Example: Code for return a*b-x*y;
  iload_a
  iload_b
  imul
  iload_x
  iload_y
  imul
  isub
  ireturn
```

Class instance

Creation of class instance

```
  Object create() {
    return new Object();
  }

  Method java.lang.Object create()
    0 new #1  // java.lang.Object
    3 dup
    4 invokespecial #4
      //java.lang.Object.<init>()V
    7 areturn
```

Instance processing

```
int i;  // instance variable
MyObj example() {
  MyObj o = new MyObj();
  return silly(o);
}
MyObj silly(MyObj o) {
  if (o != null)
    return o;
  else
    return o;
}

Method MyObj example()
  0 new #2             // Class MyObj
  3 dup
  4 invokespecial #5  // MyObj.<init>()V
  7 astore_1
  8 aload_0
  9 aload_1
 10 invokevirtual #4
     // Example.silly(LMyObj;)LMyObj;
 13 areturn

Method MyObj silly(MyObj)
  0 aload_1
  1 ifnull 6
  4 aload_1
  5 areturn
  6 aload_1
  7 areturn
```

Reference to instance variable

```
int i;  // instance variable
void setIt(int value) {
  i = value;
}
int getIt() {
  return i ;
}

Method void setIt(int)
  0 aload_0
  1 iload_1
  2 putfield #4  // Example.i I
  5 return

Method int getIt()
  0 aload_0
  1 getfield #4  // Example.i I
  4 ireturn
```

Code of throw statement

```
throw e;

  Evaluate e (the result in optop)
  athrow

Example:
  void cantBeZero(int i) throws TestExc {
    if (i == 0)
      throw new TestExc();
  }

Method void cantBeZero(int)
   0 iload_1
   1 ifne 12
   4 new #1             //Class TestExc
   7 dup
   8 invokespecial #7  // TestExc.<init>()V
   11 athrow
   12 return
```

Code of try-catch statement

try *s* catch (*class v*) s_1

L_1: Execute *s*
L_2: goto *L*
L_3: astore_*v*
  Execute s_1
  goto *L*
L:

Exception table:

From	To	Target	Type
L_1	L_2	L_3	*class*

Code of try-catch statement

try s catch (*class* v) s_1

L_1: Execute s
L_2: goto L
L_3: astore_v
    Execute s_1
    goto L
L:

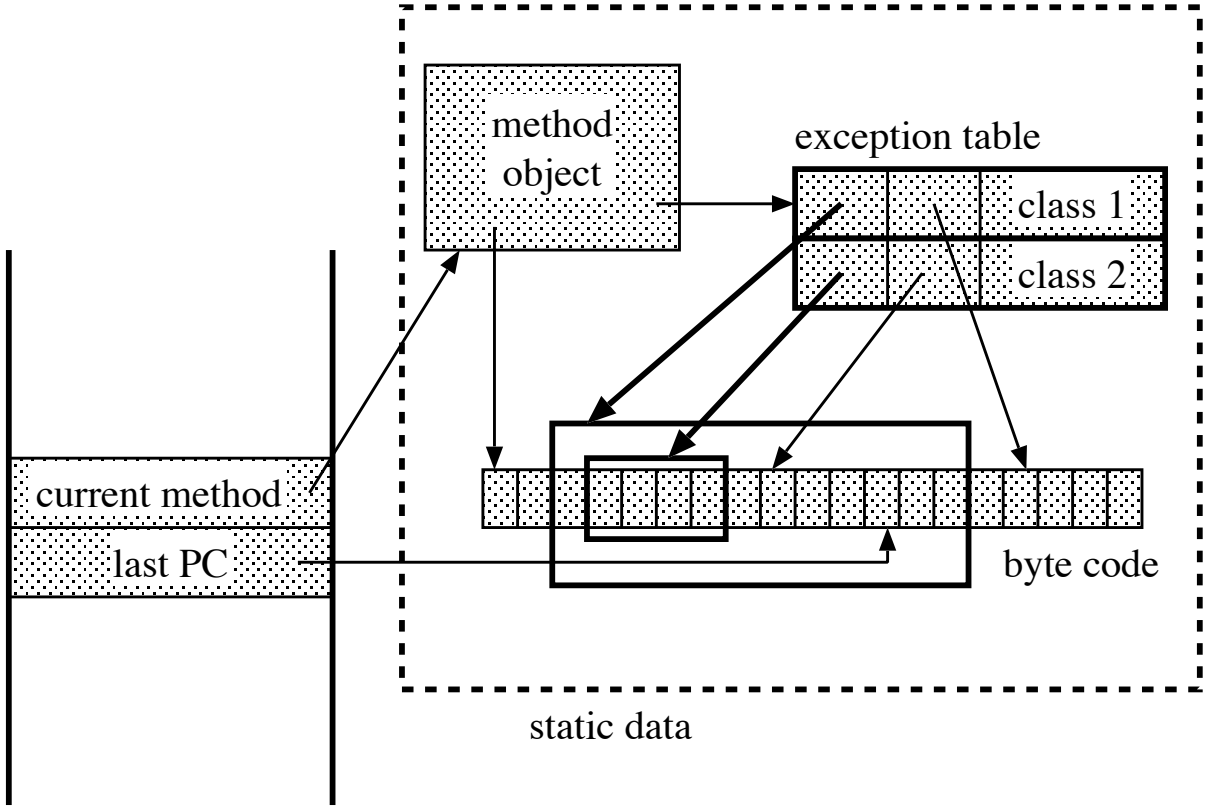
Example:

```
void catchOne() {
  try {
    tryItOut();
  }catch (TestExc e) {
    handleExc(e);
  }
}
```

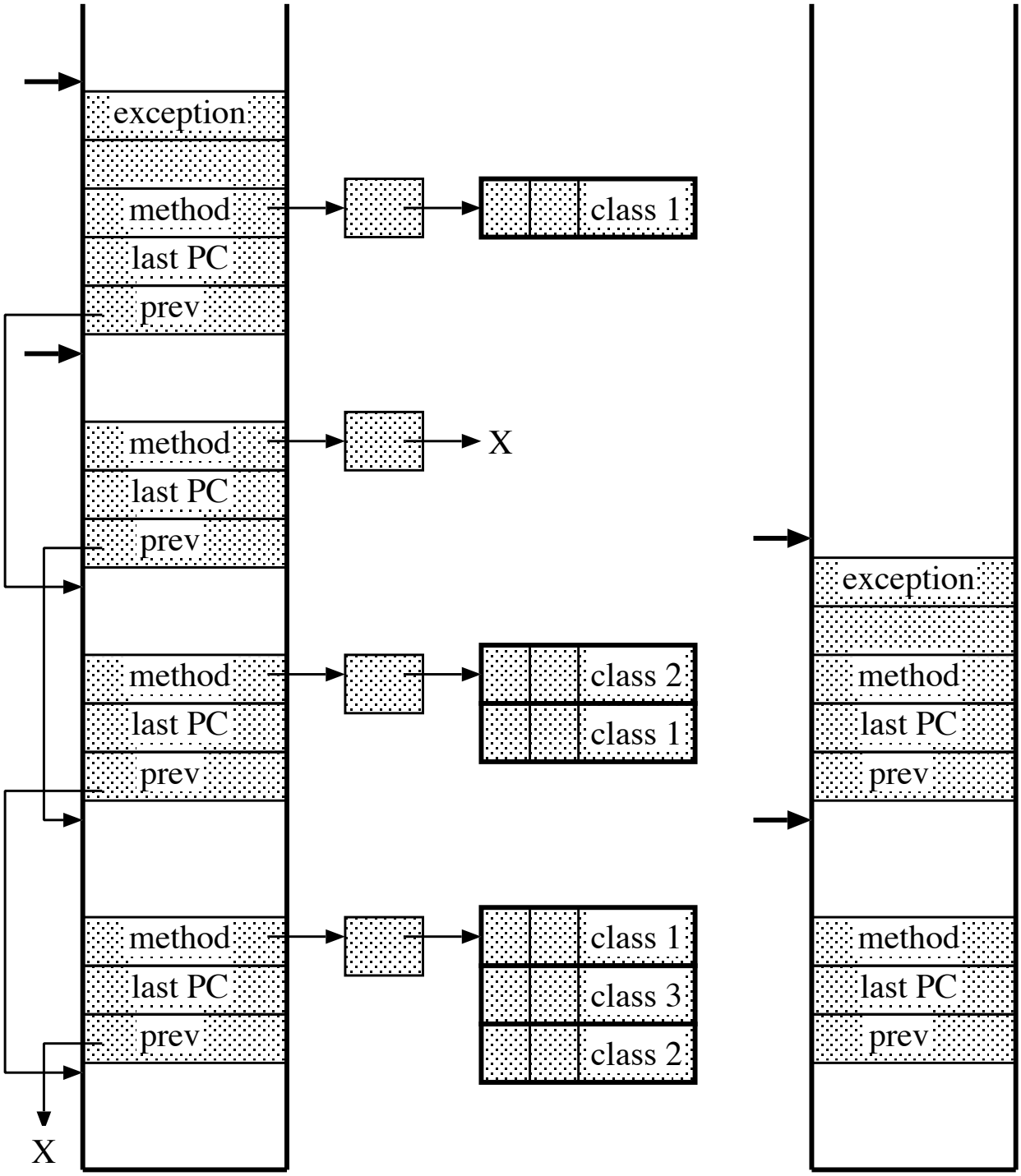
```
Method void catchOne()
  0 aload_0
  1 invokevirtual #6
  4 return
  5 astore_1
  6 aload_0
  7 aload_1
  8 invokevirtual #5
 11 return
Exception table:
 From To Target Type
 0    4  5      TestExc
```

Exception table

```
void m() {
  try {
    ...
    try {
        ...
    } catch(class_2 e) {
         ...
    }
     ...
  } catch(class_1 e) {
    ...
  }
}
```



Exception throw



athrow instruction

```
ExecuteJave() {
  for (;;) {
    switch (*pc) {
    ...
    case 191 /* ashrow */:
      Exception e = *--optop;
      for (fr = c_frame; fr != null; fr = fr->prev)
        {
        for each x in fr->c_method->eTable
        {
          if (fr->lastPC >=x.From
                  && fr->lastPC < x.To
             && e instanceof x.Type)
          {
            optop = fr + OPSTACK;
            *optop++ = e;
               c_frame = fr;
               pc = x.Target;
               break switch;
          }
        }
      }
        error("no catch");

    case 192 /* checkcast */:
      ...
    }
  }
}
```

Multiple catch clauses,

try s catch(c_1 v_1) s_1 catch (c_2 v_2) s_2

L_1: Execute s
L_2: goto L
L_3: astore_v_1
    Execute s_1
    goto L
L_4: astore_v_2
    Execute s_2
    goto L
L:

Exception table:

From	To	Target	Type
L_1	L_2	L_3	c_1
L_1	L_2	L_4	c_2

```
Example:
  void catchTwo() {
     try {
       tryItOut();
     } catch (TestExc1 e) {
       handleExc(e);
     } catch (TestExc2 e) {
       handleExc(e);
     }
   }

Method void catchTwo()
  0 aload_0
  1 invokevirtual #5
  4 return
  5 astore_1
  6 aload_0
  7 aload_1
  8 invokevirtual #7
  11 return
  12 astore_1
  13 aload_0
  14 aload_1
  15 invokevirtual #7
  18 return

Exception table:
  From To Target Type
  0    4  5      TestExc1
  0    4  12     TestExc2
```

Nesting of try-catch

```
void nestedCatch() {
  try {
     try {
       tryItOut();
     } catch (TestExc1 e) {
       handleExc1(e);
     }
   } catch (TestExc2 e) {
     handleExc2(e);
   }
}

Method void nestedCatch()
   0 aload_0
   1 invokevirtual #8
   4 return
   5 astore_1
   6 aload_0
   7 aload_1
   8 invokevirtual #7
   11 return
   12 astore_1
   13 aload_0
   14 aload_1
   15 invokevirtual #6
   18 return

Exception table:
   From To Target Type
   0    4  5      TestExc1
   0    12 12     TestExc2
```

Code of try-finally

try s finally s_1

```
  L1: Execute s
  L2: jsr Lf
      goto L
  L3: astore_v
      jsr Lf
      aload_v
      athrow
  Lf: astore_r
    Execute s1
      ret r
   L:
```

Exception table

From	To	Target	Type
L_1	L_2	L_3	any

Example:

```
void tryFinally() {
  try {
    tryItOut();
  } finally {
    wrapItUp();
  }
}

Method void tryFinally()
    0 aload_0
    1 invokevirtual #6
    4 jsr 14
    7 return
    8 astore_1
    9 jsr 14
    12 aload_1
    13 athrow
    14 astore_2
    15 aload_0
    16 invokevirtual #5
    19 ret 2

Exception table:
  From To Target Type
  0    4  8      any
```

try s catch (c v) s_1 finally s_2

L_1: Execute s
L_2: jsr L_f
goto L
L_3: asotre_v
Execute s_1
L_4: jsr L_f
goto L
L_5: asotre_v
jsr L_f
aload_v
athrow
L_f: astore_r
Execute s_2
ret r
L:

Exception table :

From	To	Target	Type
L_1	L_2	L_3	c
L_1	L_4	L_5	any

```
void tryCatchFinally() {
  try {
    tryItOut();
  } catch (TestExc e) {
    handleExc(e);
  } finally {
    wrapItUp();
  }
}

Method void tryCatchFinally()
   0 aload_0
   1 invokevirtual #4
   4 goto 16
   7 astore_3
   8 aload_0
   9 aload_3
  10 invokevirtual #6
  13 goto 16
  16 jsr 26
  19 return
  20 asotore_1
  21 jsr 26
  24 aload_1
  25 athrow
  26 astore_2
  27 aload_0
  28 invokevirtual #5
  31 ret 2

Exception table:
   From To Target Type
   0    4  7      TestExc
   0    16 20     any
```