Lecture 11

Interpretation and Emulation

Executing Java programs

Java bytecode

JVM

Interpretation

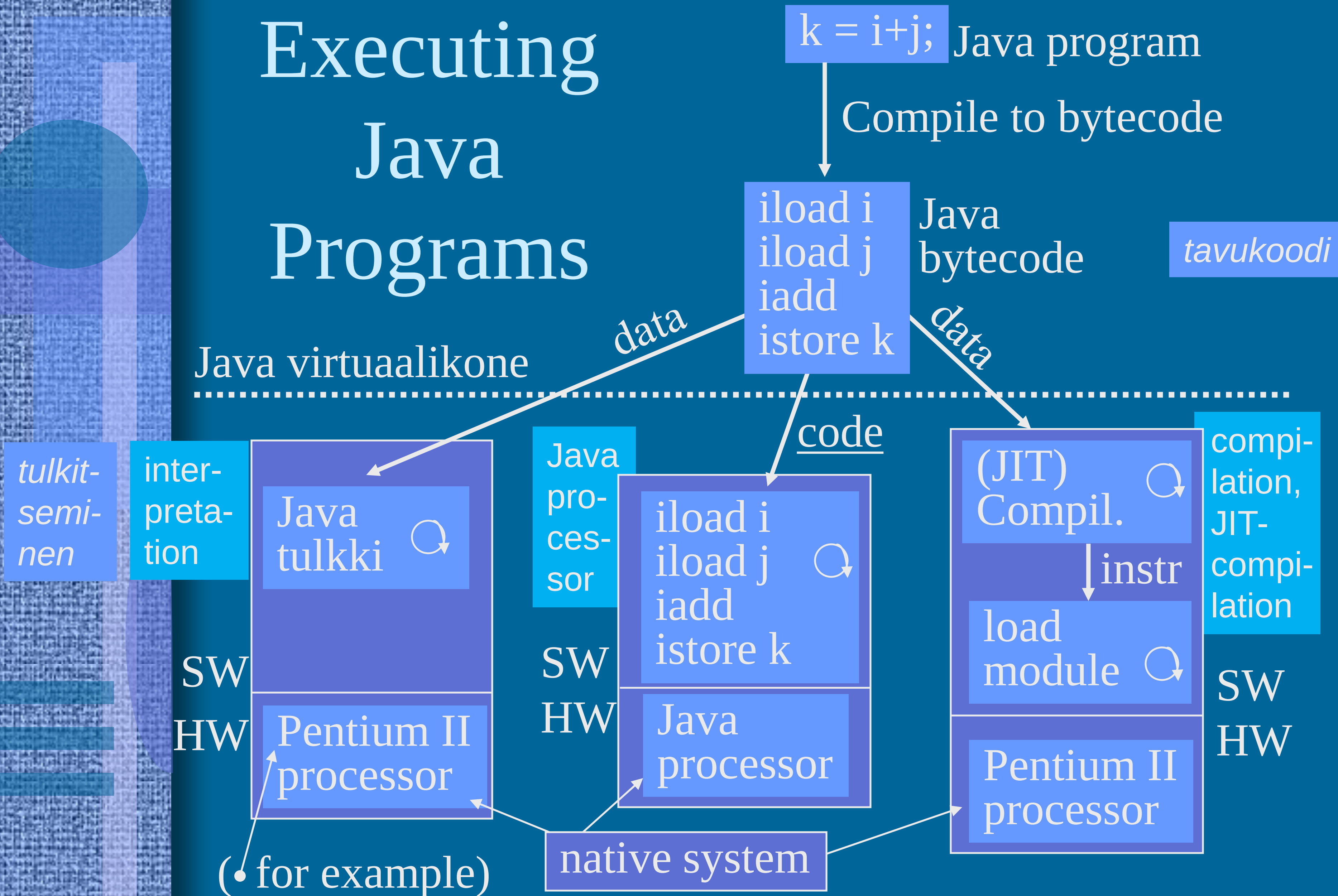
Java-processor

Compilation

JIT-compilation

JVM vs. ttk-91

Executing Java Programs



Java Virtual Machine (JVM)

- Hypothetical processor, implemented in various ways
- Generic. ”Easy” to emulate with all real processors
  - Execution based on compilation or interpretation
- Many threads can be in execution concurrently
  - Alternating or simultaneously on multiple cores
- Data structures
  - JVM “registers”, memory blocks, etc
  - Created when JVM is started
- Instruction
  - JVM (symbolic) machine instructions
  - 226 instructions

JVM data Structures

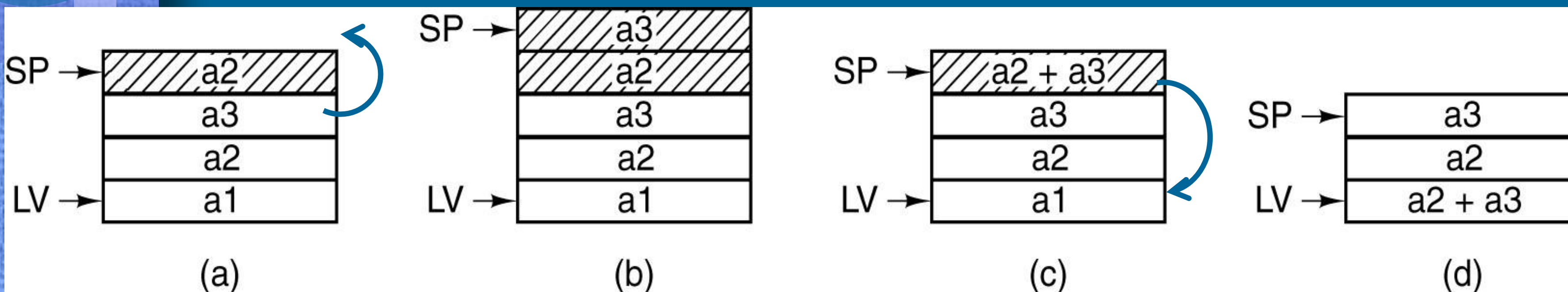
Figs 4-8, 4-9, 4-10 [Tane13]

- JVM stack
  - Like ordinary activation record stack
  - Consists of multiple *frames* (activation records) and *operand stack*
  - Use: only push/pop operations for frames
    also push/pop operations for operand stack elements
  - No need for shared memory area
  - Allocated from heap
  - Finite size or dynamically extendable (based on implementation)
    - Out of space $\Rightarrow$ StackOverflowError, OutOfMemoryError

kehys

http://java.sun.com/docs/books/vmspec/2nd-edition/html/VMSpecTOC.doc.html

Fig 4-9 [Tane13]. Stacks (2)

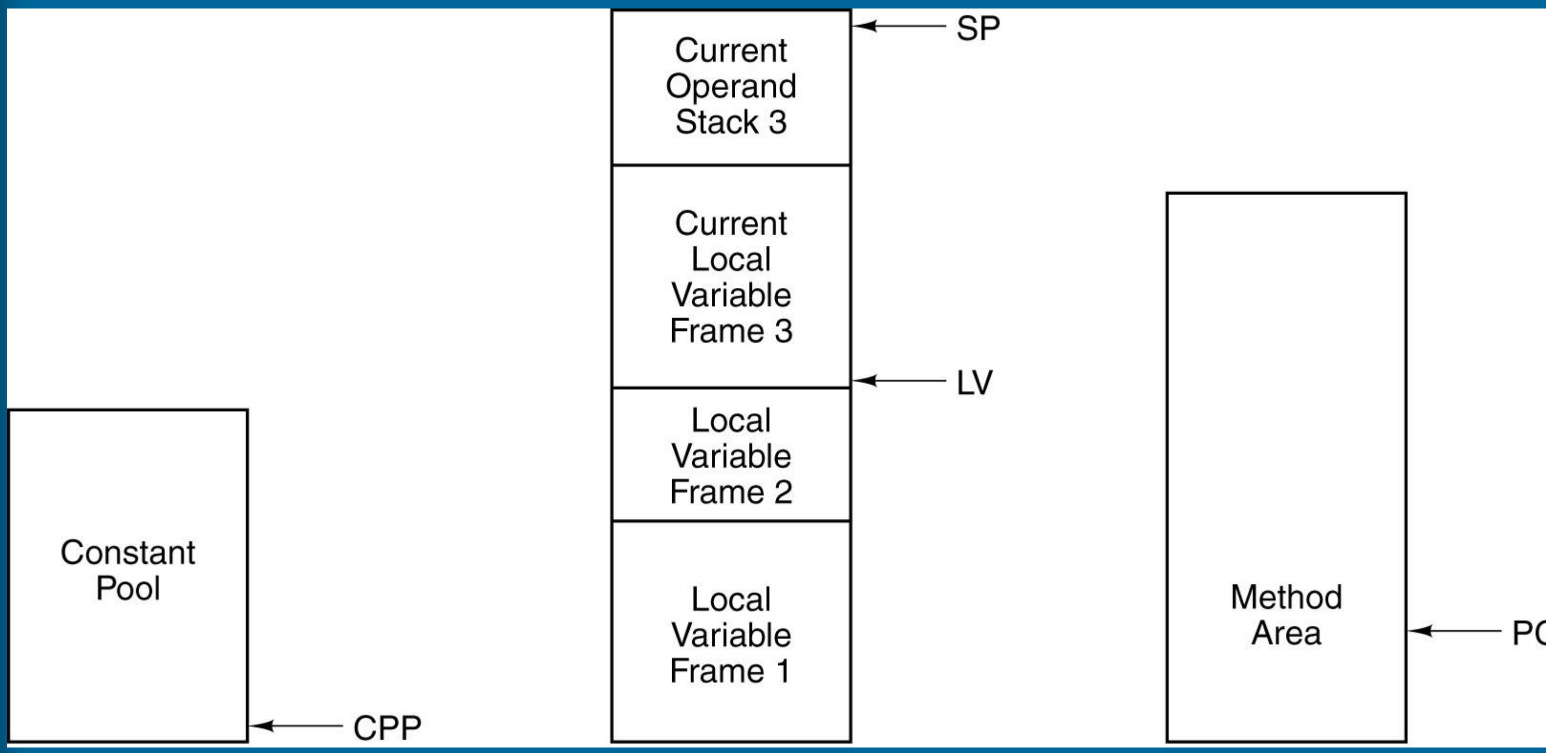


LOAD	ADD	STORE
iLOAD 2	*iADD*	*iSTORE 0*

Use of an operand <u>stack</u> (<u>not registers</u>) for doing an (e.g., integer) <u>arithmetic computation</u>.

Discuss

Fig 4-10 [Tane13]



The various parts of the IJVM memory.

JVM Data Structures (contd)

keko

roskien keruu

- JVM heap
  - Shared for all threads in one JVM
  - Automatic garbage collection
    - Unused (implicitly deallocated) memory is made available for reuse (free)
    - No need for special *free* operation in Java programs
    - May slow down execution at any time (problem in real time systems)
  - Finite size or dynamically extendable from native system heap (based on implementation)
    - Out of space ⇒ OutOfMemoryError

JVM:n tietorakenteet (jatkuu)

Fig. 4-10 [Tane13]

- JVM Method Area
  - Shared for all threads in one JVM
  - Corresponds to ordinary code segement
  - Logically part of JVM heap
  - Finite size or dynamically extendable from native system heap (based on implementation)
    - Out of space ⇒ OutOfMemoryError

JVM Data Structures (contd)

Fig. 4-10 [Tane13]

vakioallas

- Java runtime constant pool
  - For each class and each interface
  - Execution time representation for *class constant_pool* table
  - Corresponds somewhat to symbol table
  - Many different constants (compilation time literals, attributes to be solved at execuition time, etc)
  - Saved in JVM method area
  - Out of space $\Rightarrow$ OutOfMemoryError

JVM Data Structures (contd)

- Native Method Stacks
  - Implementation may use ordinary stacks ("C stacks") to support such native methods that are not written in Java
  - Used also to implement Java Interpreter
  - Not in JVM implementations without non-native methods
  - Finite size or dynamically extendable (based on implementation)
    - Out of space ⇒ StackOverflowError, OutOfMemoryError

JVM Data Structures (contd)

Fig. 4-10 [Tane13]

- JVM registers
  - PC points somewhere in JVM method area
  - CPP points to current constant pol
  - LV is the base address for local variables (vs. FP in ttk-91)
  - SP points to the top of JVM operand stack
  - All registers are implicit, they are not named in JVM machine instructions

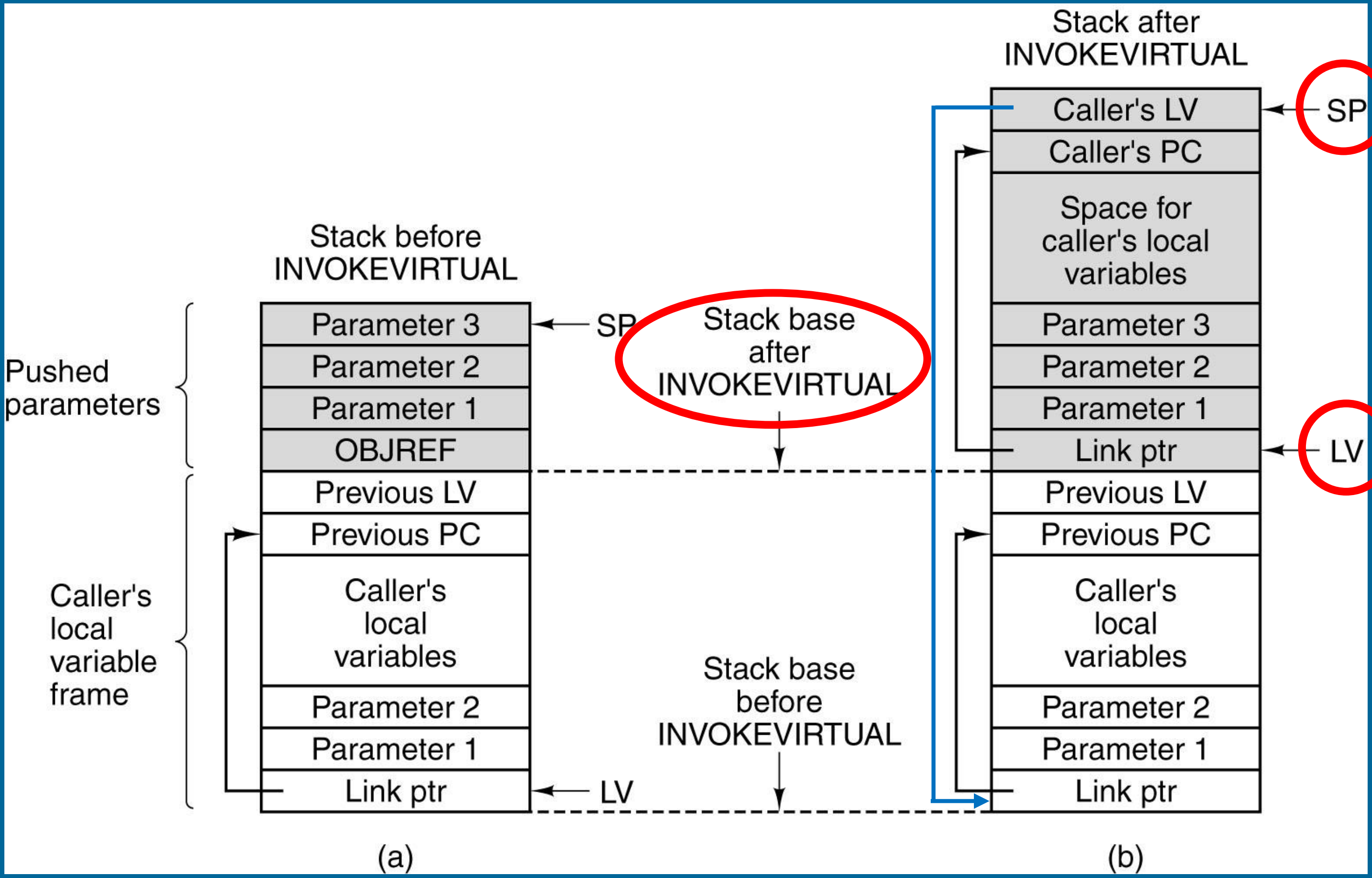
JVM Data Structures (contd)

Figs 4-12, 4-13 [Tane13]

kehys, raami

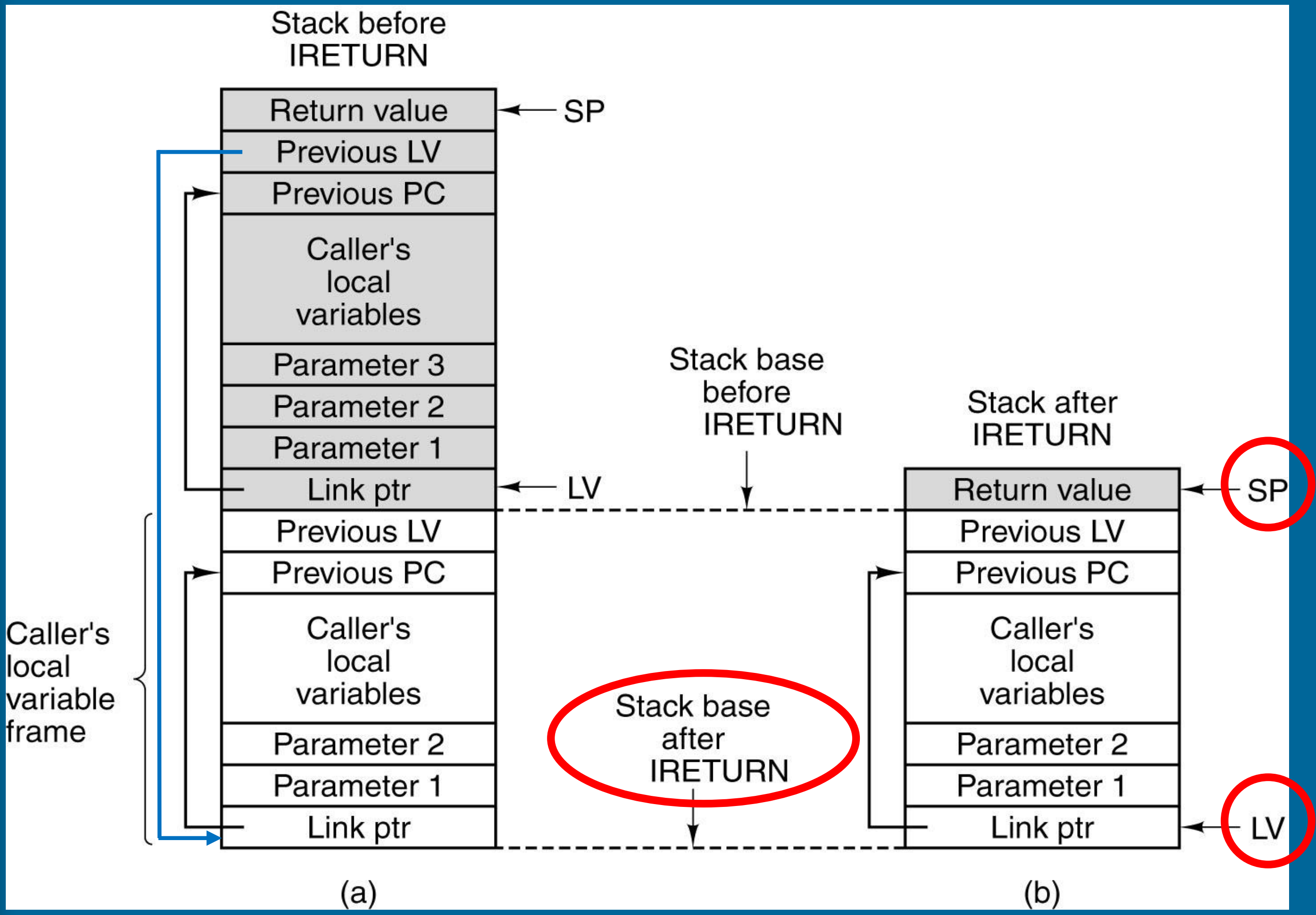
- JVM frame
  - Saved in JVM stack, created with method call, deallocated on method exit
  - All local data structures
  - Parameters, return value, intermediate results
  - Implementation tool for dynamic linking
  - Implementation tool for interrupts/exceptions

Fig 4-12 [Tane13]
The IJVM Instruction Set (2)



- Memory before executing INVOKEVIRTUAL.
- After executing it.

Fig 4-13 [Tane13]
The IJVM Instruction Set (3)



- Memory before executing IRETURN.
- After executing it.

JVM Frame Data

Fig. 4-13 [Tane13]

- All local variables (and other local data structures)
  - References are indexes (0, 1, 2, …) relative to LV
  - Indexes refer to words
  - Two word variable (long, double) is placed into two adjacent (32 bit) words
  - big-endian storage
- The operand stack containing parameters, return value, and intermediate results
  - SP points to top of stack
  - Stack architecture (vs. register architecture)

JVM Data Reference Modes (4)

- Immediate operand

iINC 2 34

Java: xLoc += 34;

- Indexed operand

iINC 2 34

Effective mem addr (LV) + 2

- Stack operand(s)

iADD

Java: a1 = a2+a3;

Replace two integers on top of stack with their sum

Fig. 4-9 [Tane13]

- Array reference via stack

aLOAD 1
iLOAD 2
iALOAD
iSTORE 3

Replace array starting address and index in stack with array element value

Java: a = T[i];

JVM Instructions

Fig. 4-11 [Tane13]

- Basic arithmetics
  - add, sub, mul, div, rem, neg
- Boolean
  - and, or, xor, shl, shr, ushr
- Stack ops
  - dup, pop, swap, create arrays, repres changes
- Load/Store
  - load, aload, store, astore, push instructions
- Comparisons
- Control transfers
- Other

Fig 4-11 [Tane13]
The IJVM Instruction Set (1)

Hex	Mnemonic	Meaning
0x10	BIPUSH *byte*	Push byte onto stack
0x59	DUP	Copy top word on stack and push onto stack
0xA7	GOTO *offset*	Unconditional branch
0x60	IADD	Pop two words from stack; push their sum
0x7E	IAND	Pop two words from stack; push Boolean AND
0x99	IFEQ *offset*	Pop word from stack and branch if it is zero
0x9B	IFLT *offset*	Pop word from stack and branch if it is less than zero
0x9F	IF_ICMPEQ *offset*	Pop two words from stack; branch if equal
0x84	IINC *varnum const*	Add a constant to a local variable
0x15	ILOAD *varnum*	Push local variable onto stack
0xB6	INVOKEVIRTUAL *disp*	Invoke a method
0x80	IOR	Pop two words from stack; push Boolean OR
0xAC	IRETURN	Return from method with integer value
0x36	ISTORE *varnum*	Pop word from stack and store in local variable
0x64	ISUB	Pop two words from stack; push their difference
0x13	LDC_W *index*	Push constant from constant pool onto stack
0x00	NOP	Do nothing
0x57	POP	Delete word on top of stack
0x5F	SWAP	Swap the two top words on the stack
0xC4	WIDE	Prefix instruction; next instruction has a 16-bit index

The IJVM instruction set. The operands *byte*, *const*, and *varnum* are 1 byte. The operands *disp*, *index*, and *offset* are 2 bytes.

Fig 4-11 [Tane10], Compiling Java to IJVM (1)

```
i = j + k;
if (i == 3)
    k = 0;
else
    j = j – 1;
```

(a)

1		ILOAD j	// i = j + k	0x15 0x02
2		ILOAD k		0x15 0x03
3		IADD		0x60
4		ISTORE i		0x36 0x01
5		ILOAD i	// if (i == 3)	0x15 0x01
6		BIPUSH 3		0x10 0x03
7		IF_ICMPEQ L1		0x9F 0x00 0x0D
8		ILOAD j	// j = j – 1	0x15 0x02
9		BIPUSH 1		0x10 0x01
10		ISUB		0x64
11		ISTORE j		0x36 0x02
12		GOTO L2		0xA7 0x00 0x07
13	L1:	BIPUSH 0	// k = 0	0x10 0x00
14		ISTORE k		0x36 0x03
15	L2:			

(b) (c)

a) A Java fragment.

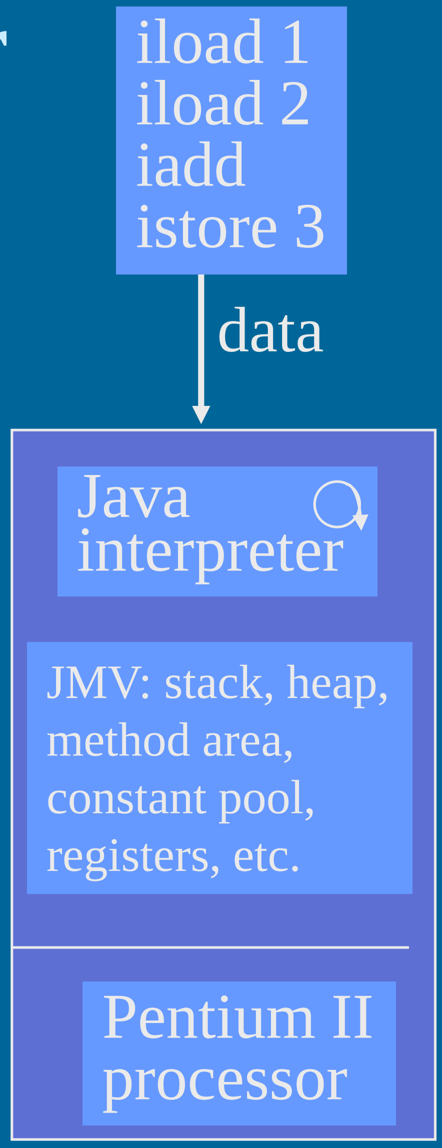
b) The corresponding Java assembly language.

c) The IJVM program in hexadecimal.

Discuss

Interpreter can be called JVM !!

Java Interpreter

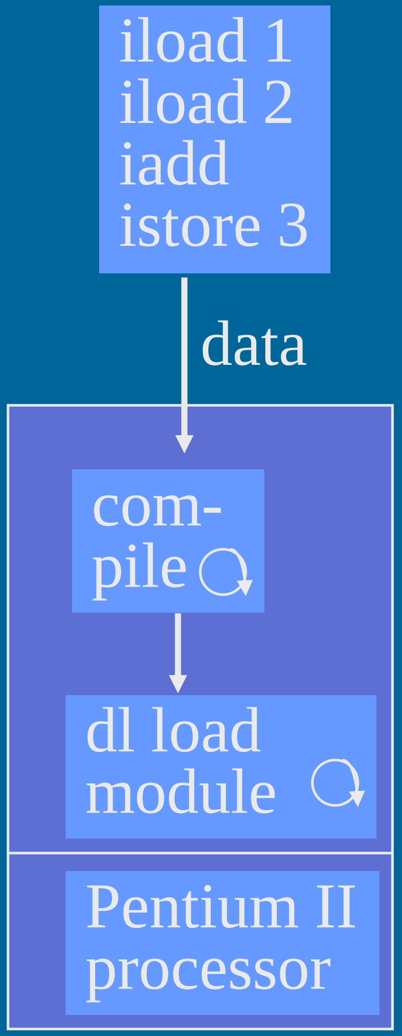


- Emulate JVM machine language (byte code) instructions
- One (byte code) instruction at a time
- JVM registers and memory areas implemented as interpreter data structures in memory
  - Compare to Titokone and ttk-91
- Slow, but flexible

Compile to Native System

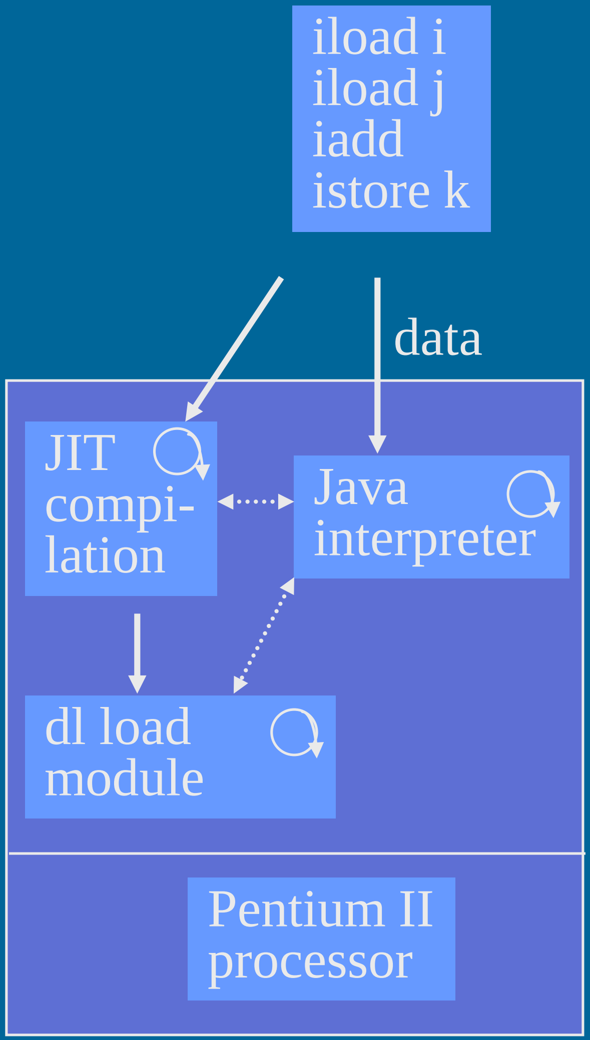
Different ways to implement compilation

- (a) Compile byte code directly to target system native machine language and execute it normally
- (b) Compile byte code first to HLL (e.g., C), which is then compiled with standard compiler to native machine language
  - First part is relatively easy
  - Last part exists already
- Problem: no dynamic linking



JIT Compilation

- JIT = Just-in-Time
- Emulate and/or compile depending on situation
- Compile Java class to dynamically linkable module in native machine language and link it, but only just before class method is called (1st time?)
- Need lots of memory
- May slow down execution (compared to interpretation) if compilation and linking takes more time than interpretation
  - Use interpretation if only called once?
  - Compile only on 2nd call time?
- JVM registers and memory areas implemented as interpreter data structures which are also used by native code

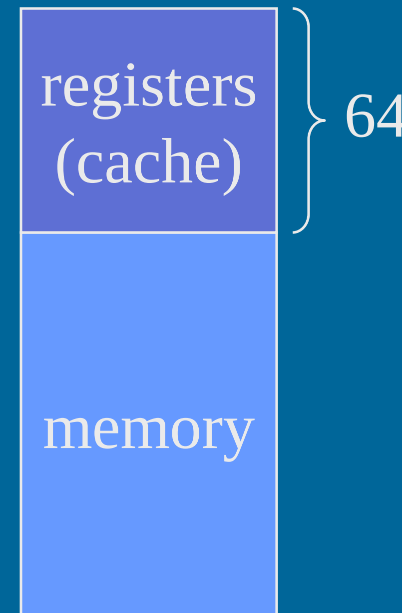


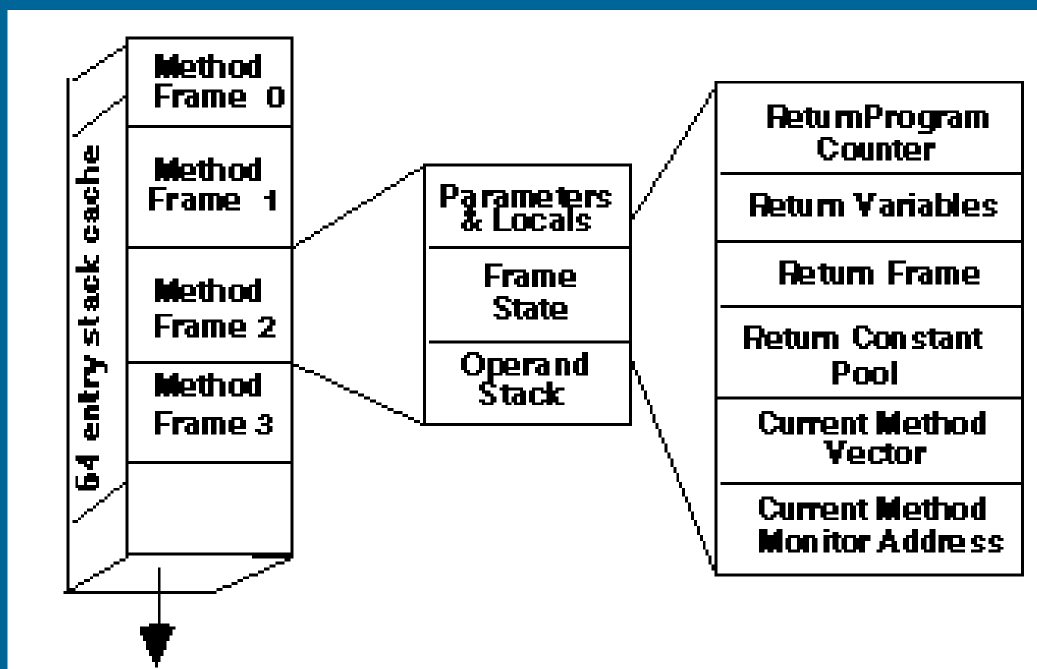
Java Processor: Sun PicoJAVA II

- Processor definition for a system where byte code programs can be executed as is
- Elective cache and floating point processor
- All 226 JVM machine instructions
  - Some machine instructions implemented as subroutines which are activated via interrupts
- Also 115 other machine instructions to effectively implement operating systems and (other) programming language compilers
  - C and C++

PicoJAVA II Stack

- 64 (cache-) device register JVM to store top of JVM stack
  - Rest of JVM stack is in memory





Shawn Lauzon,
Survey of the JavaChip

PicoJAVA II Registers

- 25 registers á 32 bits
  - PC, LV, CPP, SP (stack grows to smaller addresses)
  - OPLIM lower limit for SP; reference below causes (stack overflow) interrupt
  - FRAME points to return address stored after local vars
  - PSW (status register)
  - Register to manage top of stack special registers
  - 4 registers to manage interrupts and break points
  - 4 registers to manage threads
  - 4 registers for implementation of C and C++ programs
  - 2 bounds registers to define current memory segment
  - CPU version number and configuration registers

PicoJAVA Extra Instructions

- Read/write for extra registers
- Pointer manipulation instructions
  - Any memory location can be directly referenced
  - Needed for C/C++
- C/C++ subroutine calls and returns
- Native HW manipulation
  - Clear cache (partly? Completely=), ...
- Other instructions
  - power on/off, ...

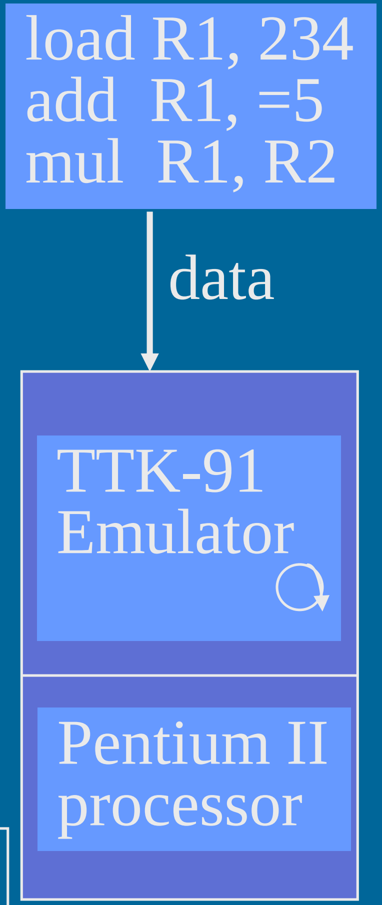
Other Java-suorittimia

- JEM (Rockwell Collins)
- PSC1000 (Patriot Scientific)
  - dSys (Germany), medical devices
- MJ501 (LG Semicon)
  - TV, smart cards
- JSR-001, Real-Time Specification for Java (Java Community Process, ”Sun Microsystems”)
  - aJile: aJ-80, aJ-100, smart mobile devices

- Komodo, SHAP, jHISC, Cjip, ARM926EJ-S, ObjectCore, …

TTK-91 Emulation

- TTK-91 emulation
- Part of Titokone
- Emulate one ttk-91 machine instruction at a time
- TTK-91 registers and memory emulated as data structures in Titokone



See simulator code, project Titokone
http://www.cs.helsinki.fi/group/nodes/kurssit/tito/2012s/Interpreter.java
http://www.cs.helsinki.fi/group/nodes/kurssit/tito/2012s/Processor.java

Discuss

-- End --

- Cache (1965, Maurice Wilkes)
  - IBM S/360 Model 85
    - 1968
    - 256 lohkoa á 64 tavua