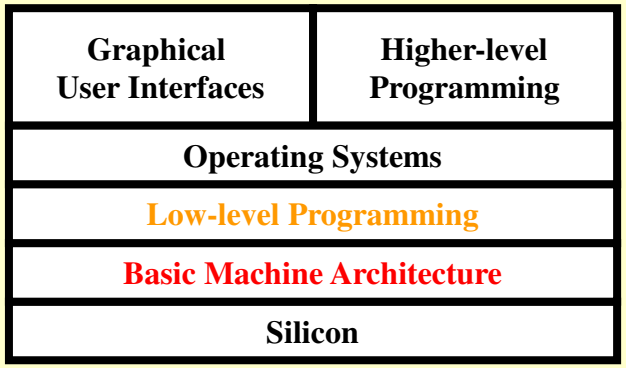
CSCU9V4
Systems

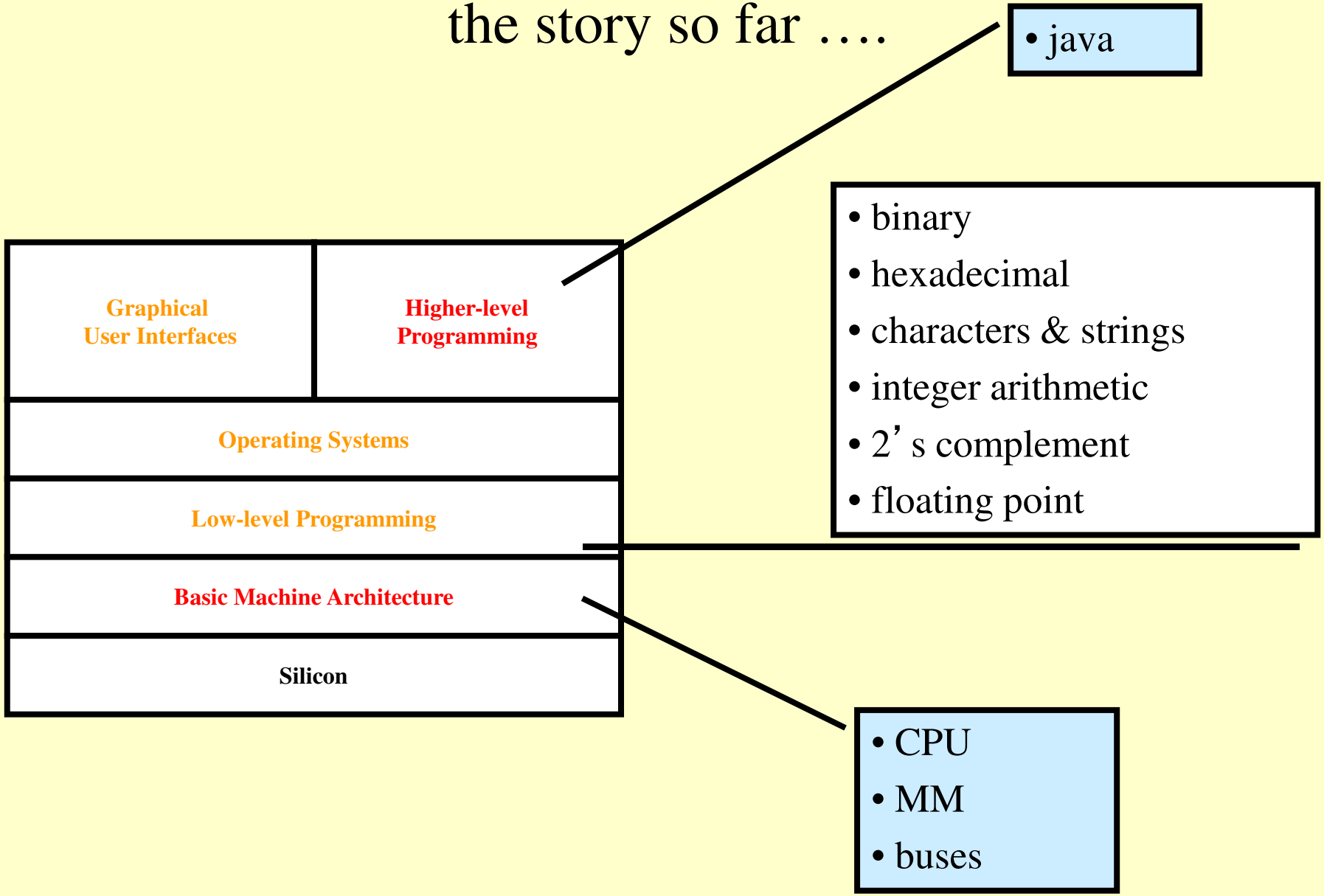
Systems lecture 10

Computer Organisation

A Virtual Machine

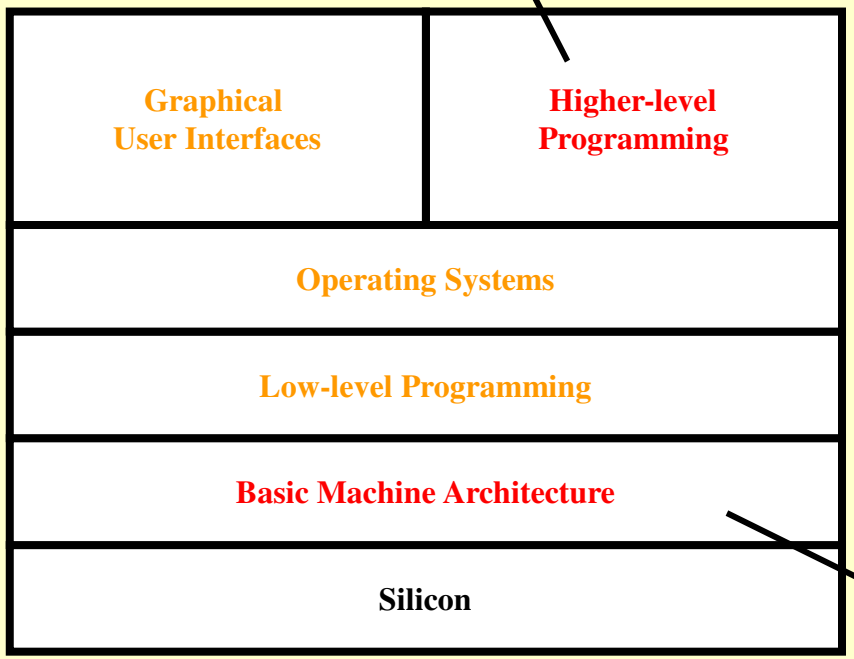


the story so far ….



how do we run programs ?

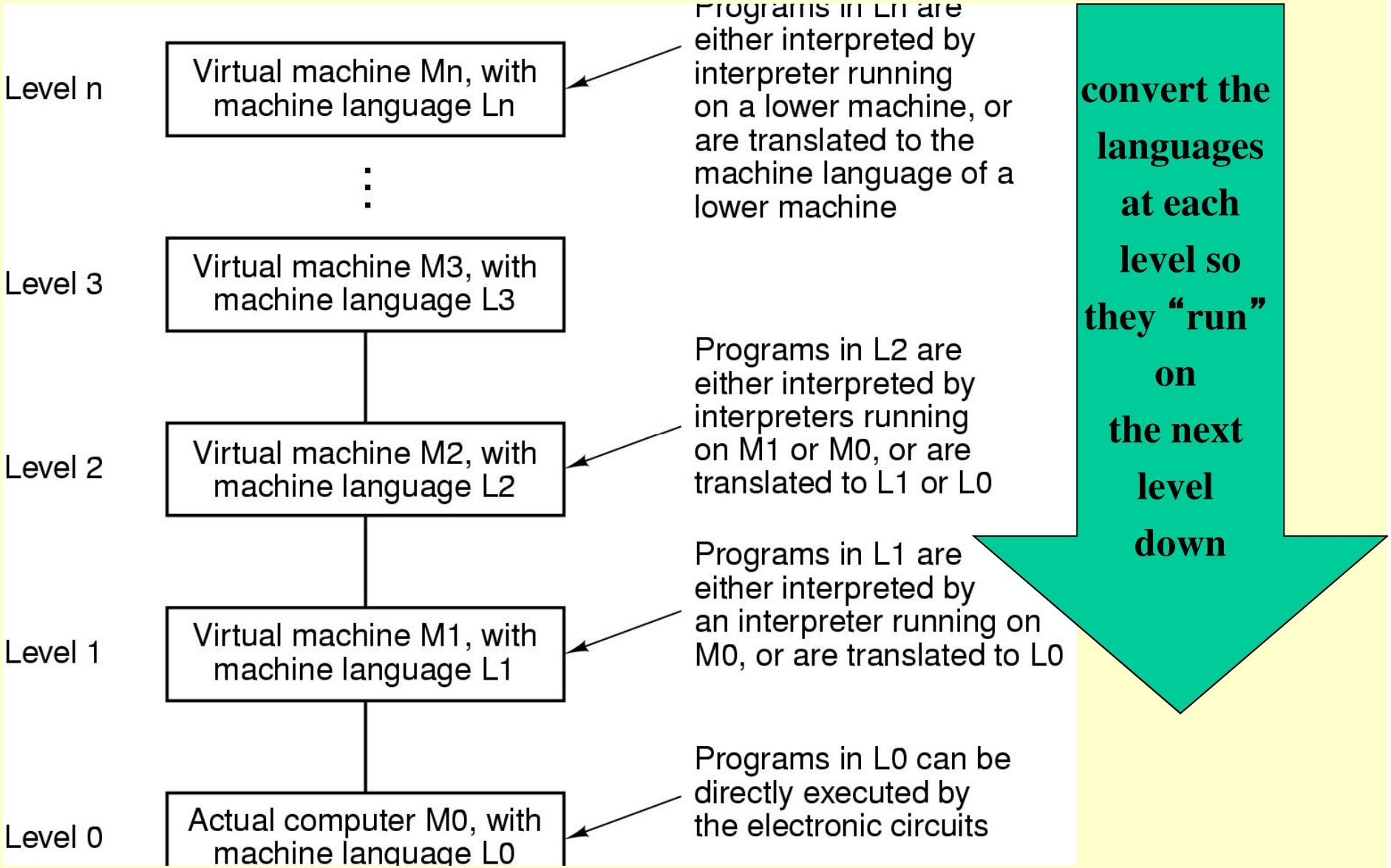
convenient for people



Convert the languages at each level so they "run" on the next level down

convenient for the technology

general case: multilevel machine (Tanenbaum)

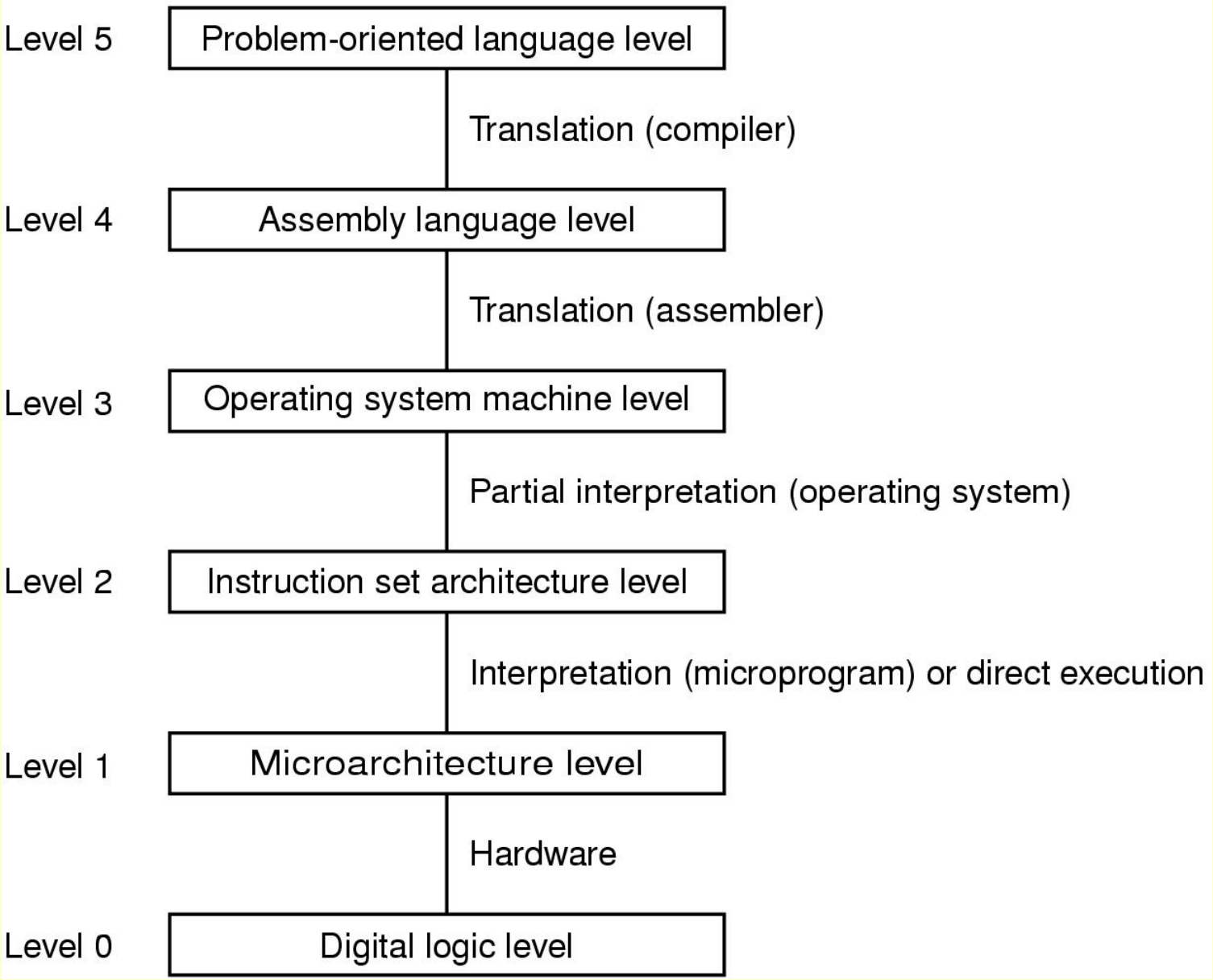


What happens at each stage?

- If we have a language, say L_n we want to "run" on M_n
- How do we "implement" M_n ? (only m_0 is implemented directly!)
- Two ways:
  - Replace each instruction of program written in L_n with an equivalent set of L_{n-1} instructions
  - M_{n-1} executes the L_{n-1} instructions (**Translation**)

  - Write a program in L_{n-1} which takes programs in L_n as input data
  - program executes the equivalent set of L_{n-1} instructions for every L_n instruction (**Interpretation**)
  - Both are widely used: generally translation is more efficient, but interpretation easier to debug
    - Combinations of both are also used
- Many layers may be used.

An example computer (Tanenbaum)

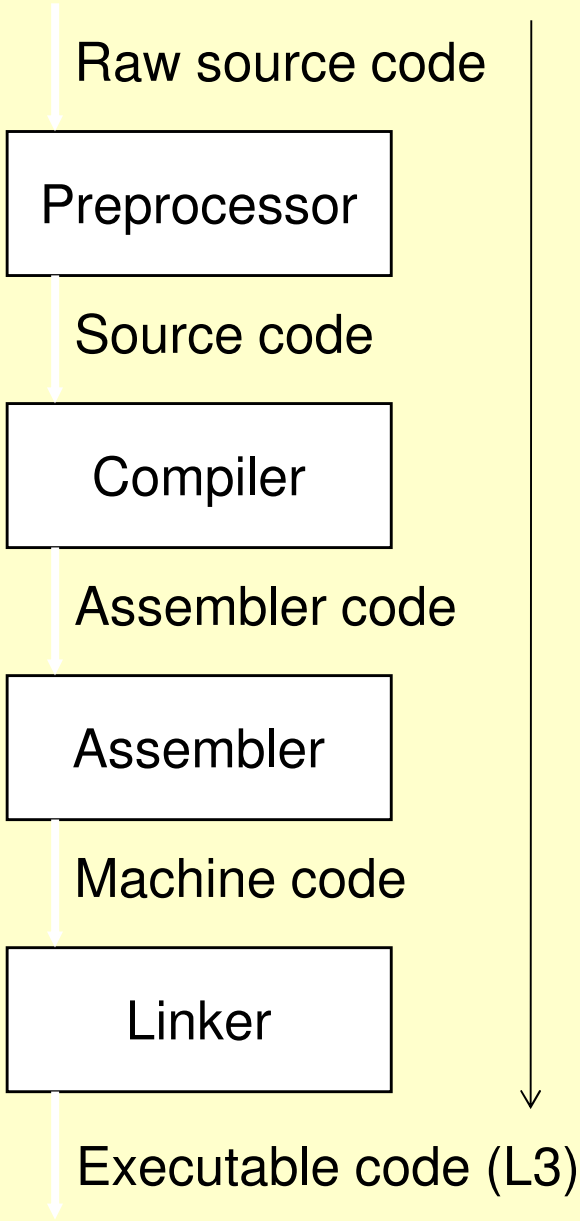


Problem-oriented High-level Languages

- There are many different **types** of language, and many different languages of each.
- Imperative
  - Basic, C, Pascal, Ada, COBOL, FORTRAN, ...
- Object-oriented
  - C++, C#, Java, Objective C, Eiffel,...
- Functional
  - Miranda, Lisp, Haskell,...
- Logic
  - Prolog
- There are ***hundreds*** of different programming languages.
  - There are also many **scripting** languages: usually directly executed by some software package
- The language used is chosen to suit the application domain
  - (it must, of course, be available for the target hardware and O.S.).

Compiler & Linker

- Compiler translates a high level language to level 3 or 4
- Compiler usually come with a group of tools
- Pre-processor
  - Putting together modules of source code
  - Translating macros
- Compiler
  - Lexical analysis, Syntax check, semantic check
  - Generate intermediate code (simplified)
  - Code optimising, assembler code generator
- Assembler
  - Sometimes included in compiler
  - Generates machine code and symbol table
- Linker
  - Changes movable addresses to absolute addresses
  - Combines several parts of a program (library)
  - May originate from various compiler runs



An Example

- High-level language:
  - `Xlevel = ylevel + 2;`
- Assembly language

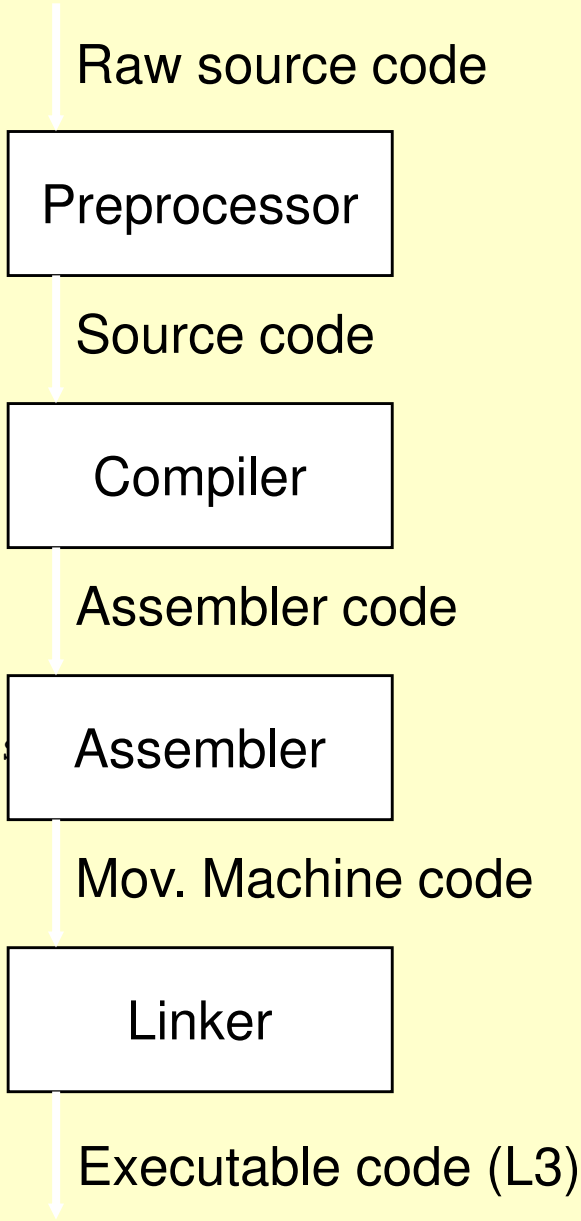
```
LOAD      R1, 44(R7)
ADD       R1, #2
STORE     R1, 36(R7)
```

- Relocatable machine code

```
0001 01 00 00000000 *(load,normal address)
0011 01 10 00000010 * (add,direct mode)
0010 01 00 00000100 * (store,normal addres
```

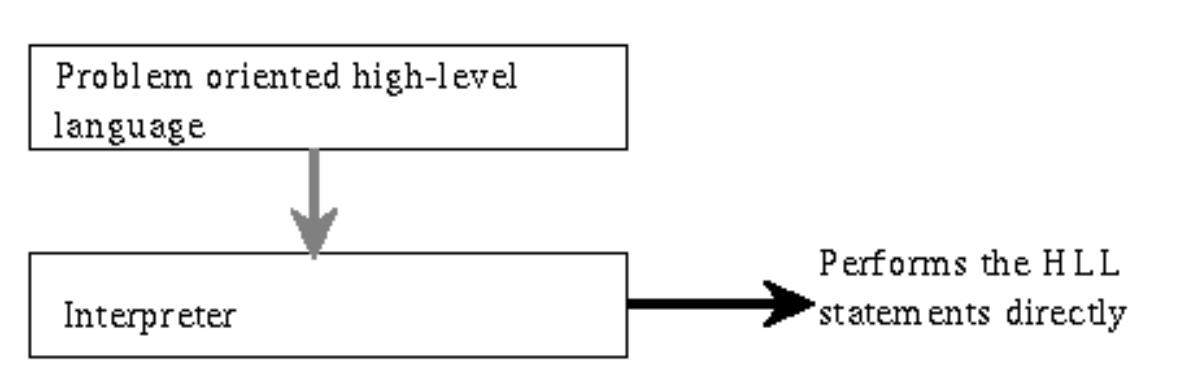
- Executable machine code (start at `Locn=00001111`)

```
0001 01 00 00001111 (load, 15)
0011 01 10 00000010 (add, 2)
0010 01 00 00010011 (store, 19)
```



Interpreter

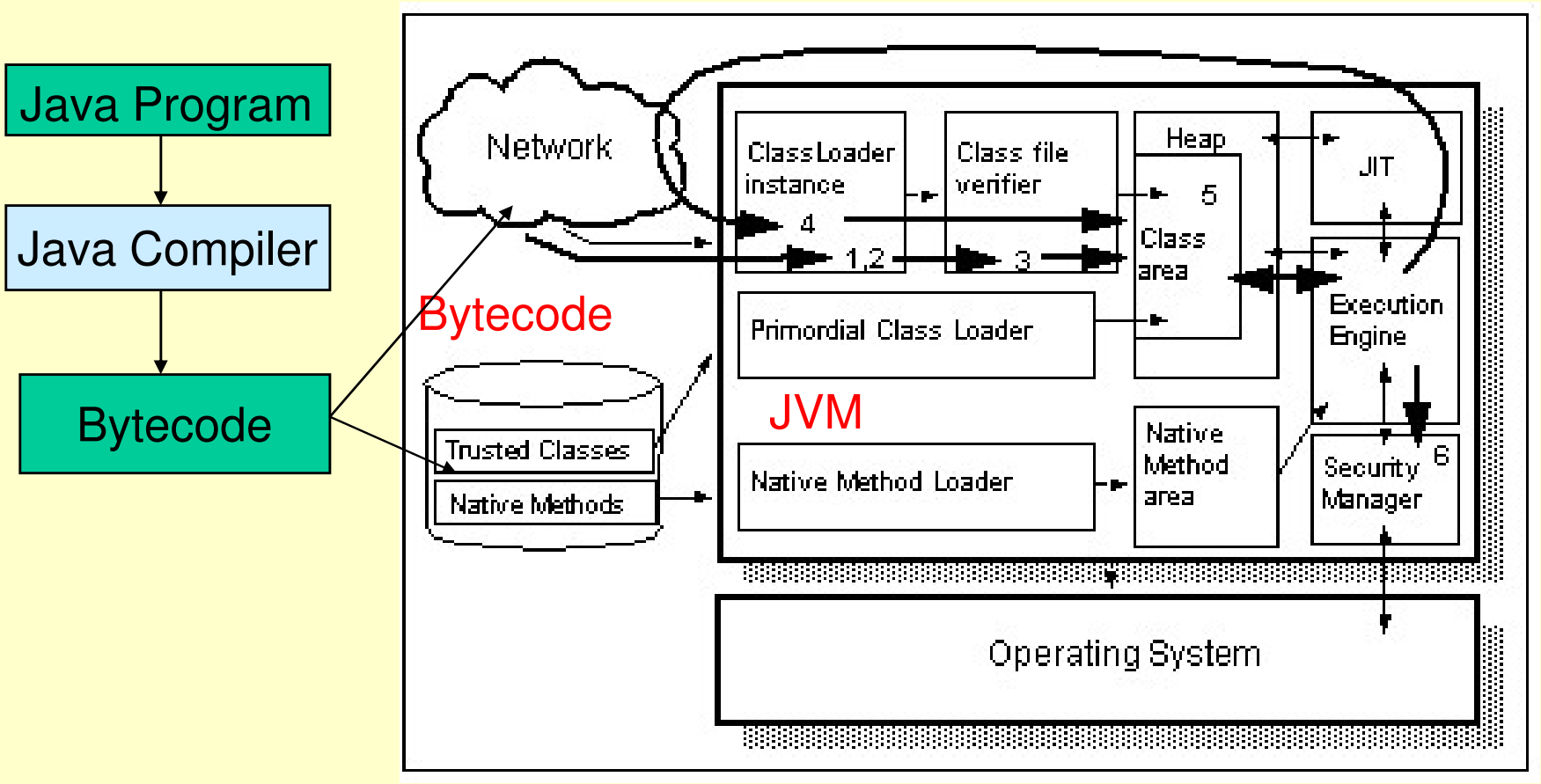
- An *interpreter* is a program which reads in statements in some language, and directly executes them.



- BASIC
- Java virtual machine is an interpreter
  - Does not execute Java directly, but interprets the Java bytecode
  - Portable (i. e. runs on a wide variety of machine types)
- Most scripting languages are of this form (PERL, R, PYTHON, JavaScript, …)

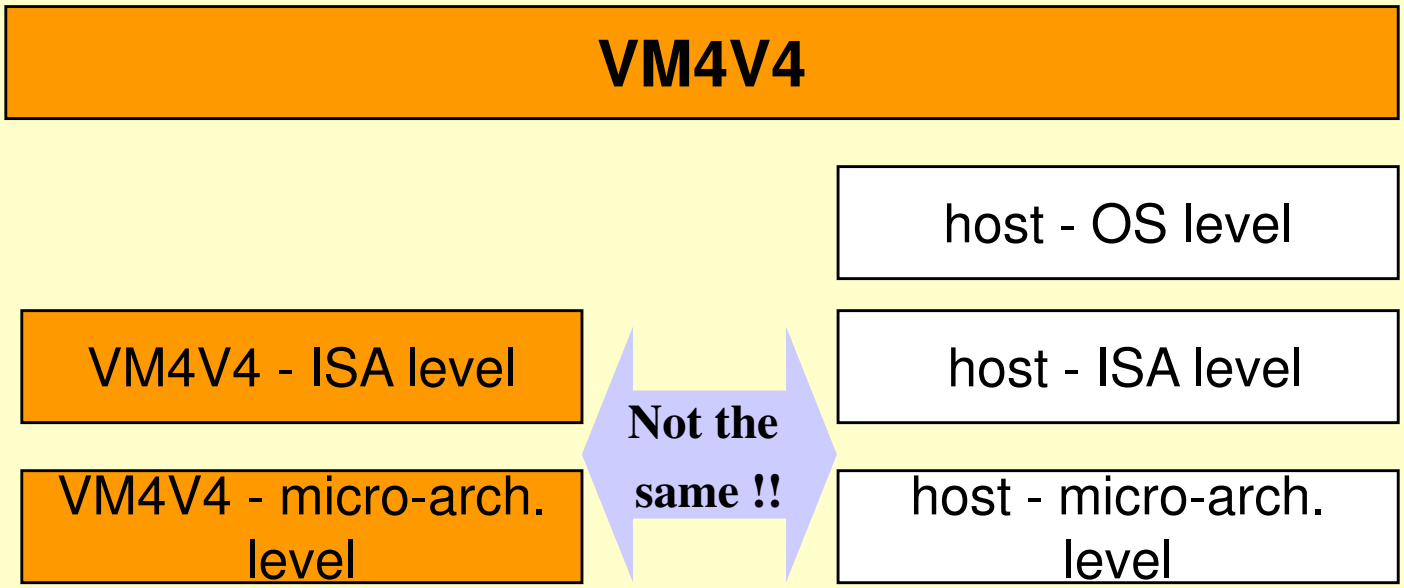
Interpretation and Java

- The Java virtual machine is one well known example of an interpreter.
- It doesn't actually execute Java directly, but it interprets the bytecode produced by the Java compiler.

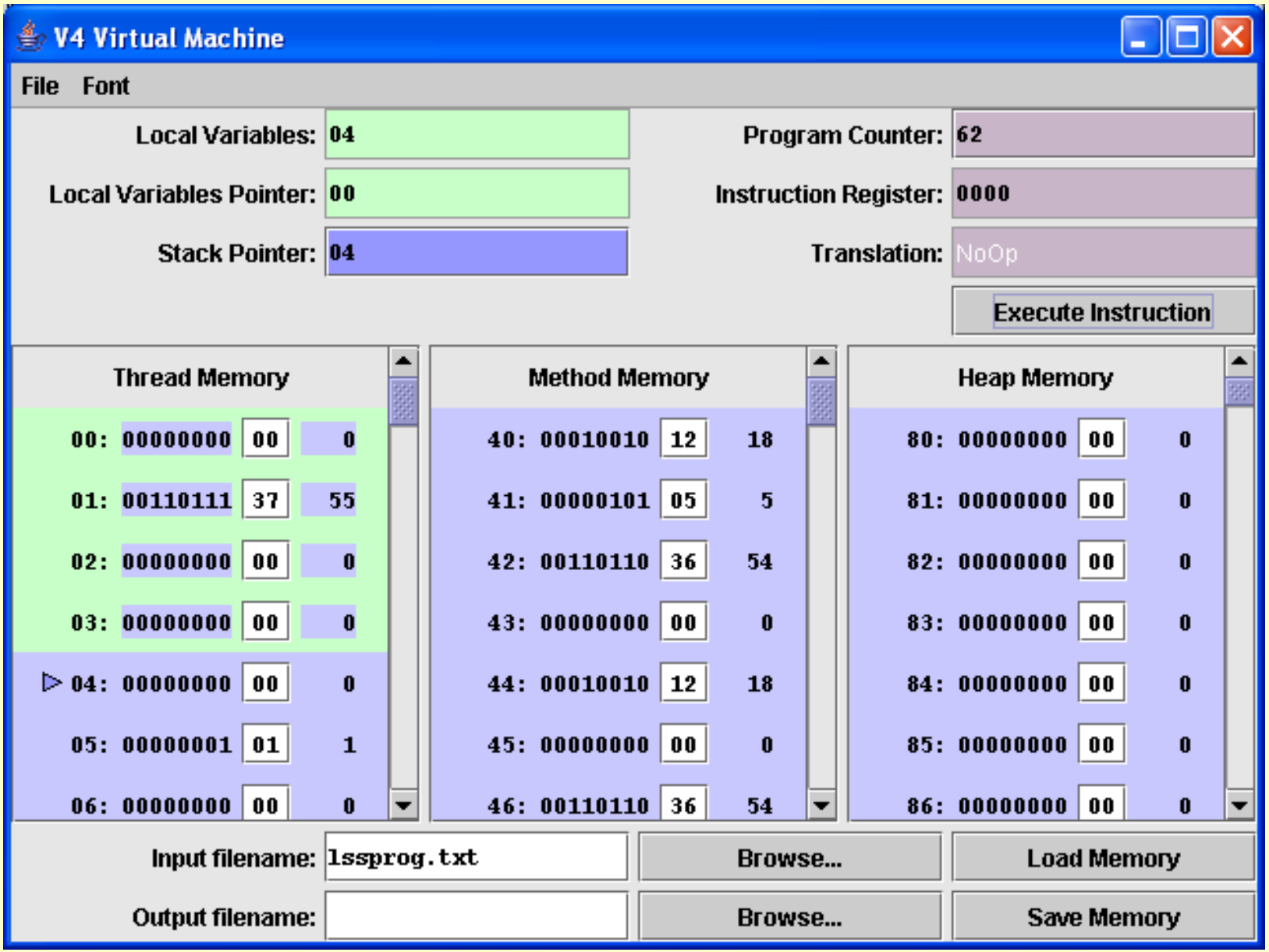


VM4V4

- based upon the Java virtual machine - but *much* simpler
- VM4V4 is a virtual machine which is physically above level 3
- yet it reflects an imaginary machine with its *instruction set architecture* (level 2) and *micro-architecture level* (level 1).
- however - these imaginary levels are useful as examples.



V4 Virtual machine

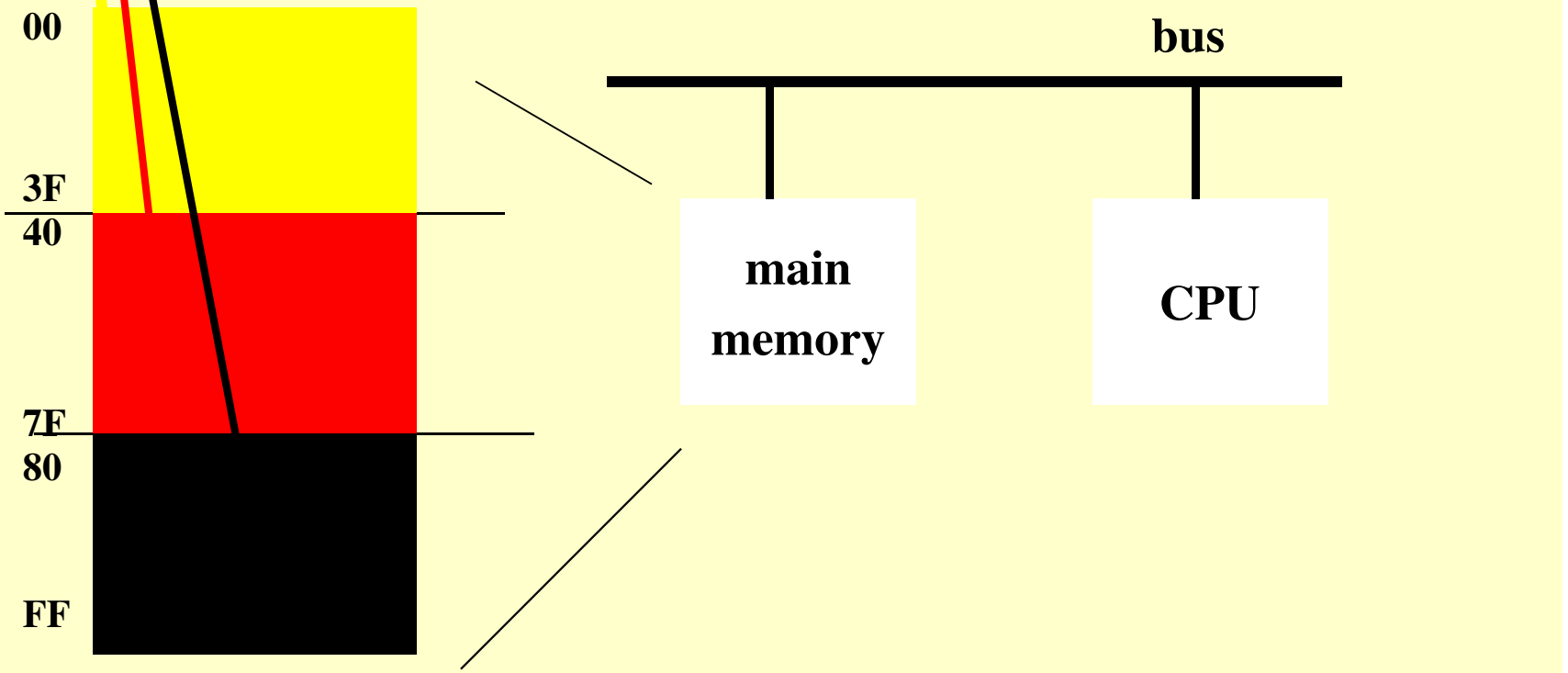


VM4V4 - micro-architectural level

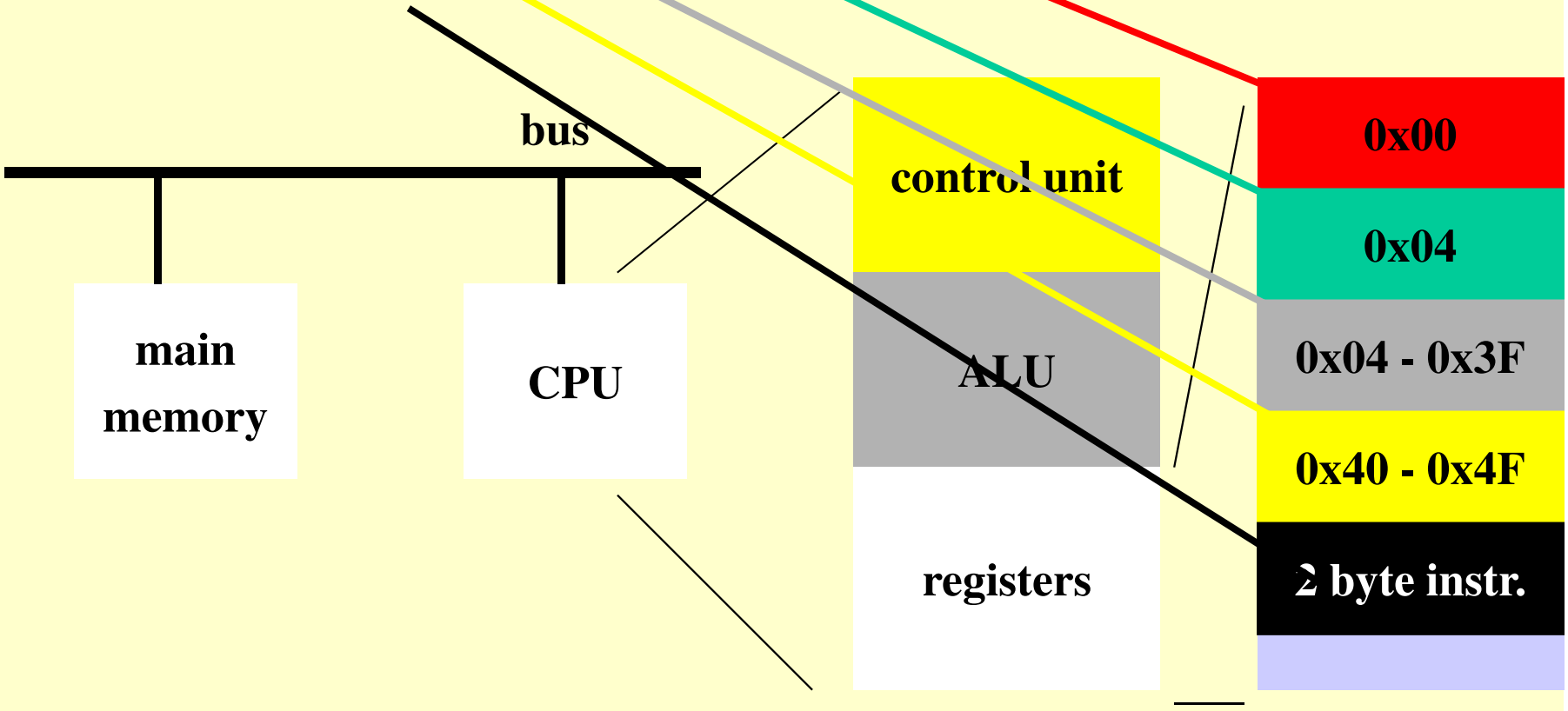
- memory
  - frame area - information for current thread (thread memory)
  - method area - where the program is stored (method memory)
  - heap - spare memory for dynamic allocation of memory (heap memory)

- registers
  - local variables pointer - points to start of current frame, fixed at 0x00
  - number of local variables - how much memory is reserved for local vars.
  - Stack pointer - points to top of stack
  - program counter - points to the *next* instruction
  - instruction register - holds current instruction
    - (a translation of this instruction is provided)

- memory
  - frame area - information for current thread (thread memory)
  - method area - where the program is stored (method memory)
  - heap - spare memory for dynamic allocation of memory (heap memory)



- registers
  - local variables pointer - points to start of current frame, fixed at 0x00
  - number of local variables - how much memory is reserved for local vars
  - Stack pointer - points to top of stack
  - program counter - points to the *next* instruction
  - instruction register - holds current instruction



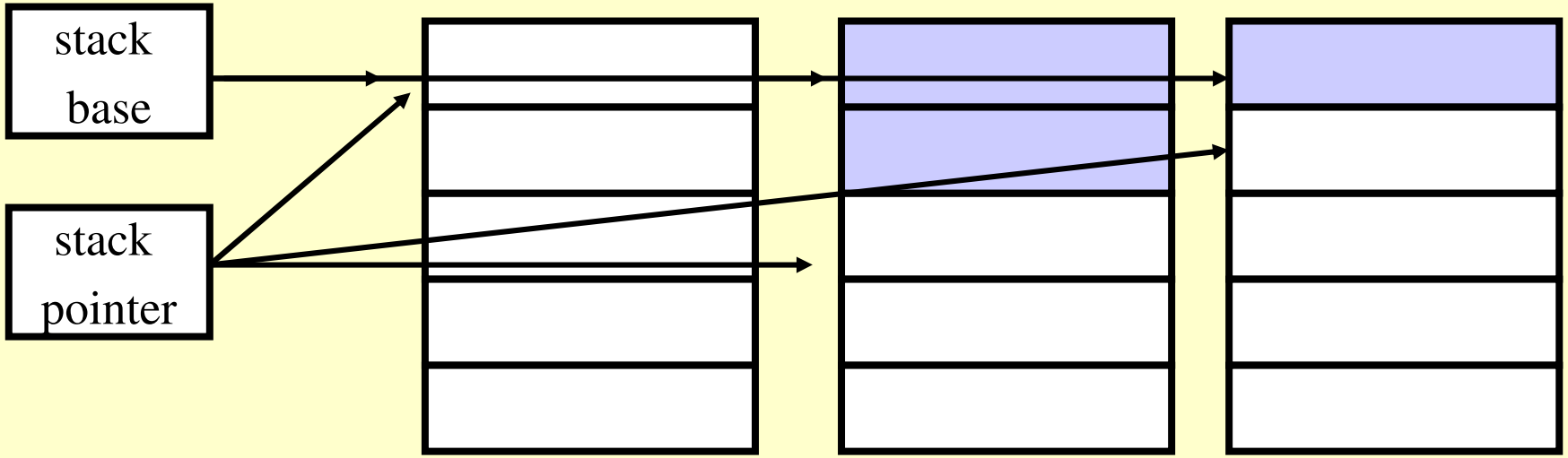
VM4V4 - instruction set architecture (isa) level

- Instructions
  - 2 byte instructions - 1 for opcode, 1 for operand
  - integers - 2's comp - 1 byte
  - logical operators - AND, OR
  - arithmetic - add, sub, multiple
  - operations are stack-based
  - branching - conditional & unconditional

opcode	operand
iadd	
60xx	

- **pops 2 numbers from stack**
- **adds them**
- **pushes answer back onto stack**

what is a stack ?

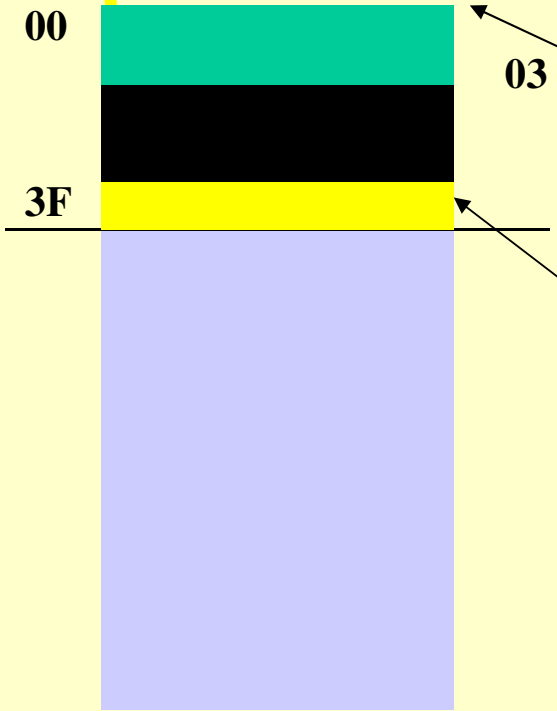


empty **push on 2 cells** **pop off 1 cell**

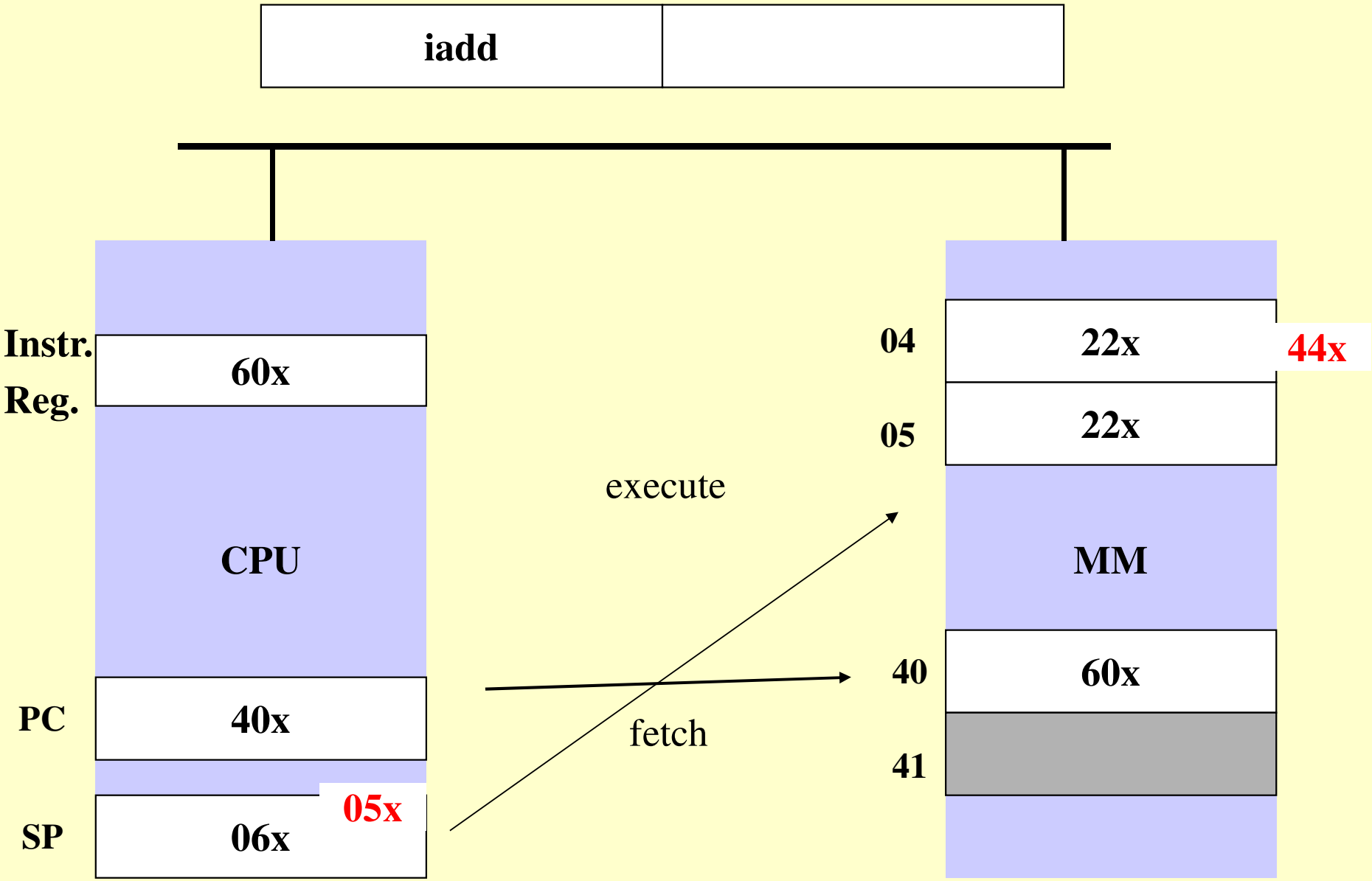
- working memory
- intermediate results
- storage of “local” variables
- dynamic - changing
- SB often fixed

where is the VM4V4 stack ?

- memory
  - frame area - information for current thread
  - method area - where the program is stored
  - heap - spare memory for dynamic allocation of memory



- registers
  - local variables pointer - points to start of current frame, fixed at 0x00
  - number of local variables - how much memory is reserved for local vars.
  - Stack pointer - points to top of stack
  - program counter - points to the *next* instruction
  - instruction register - holds current instruction



End of lecture