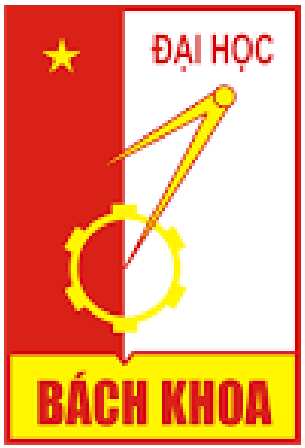
Nature of Programming Languages

Topic 1. Introduction

Han, Nguyen Dinh (han.nguyendinh@hust.edu.vn)

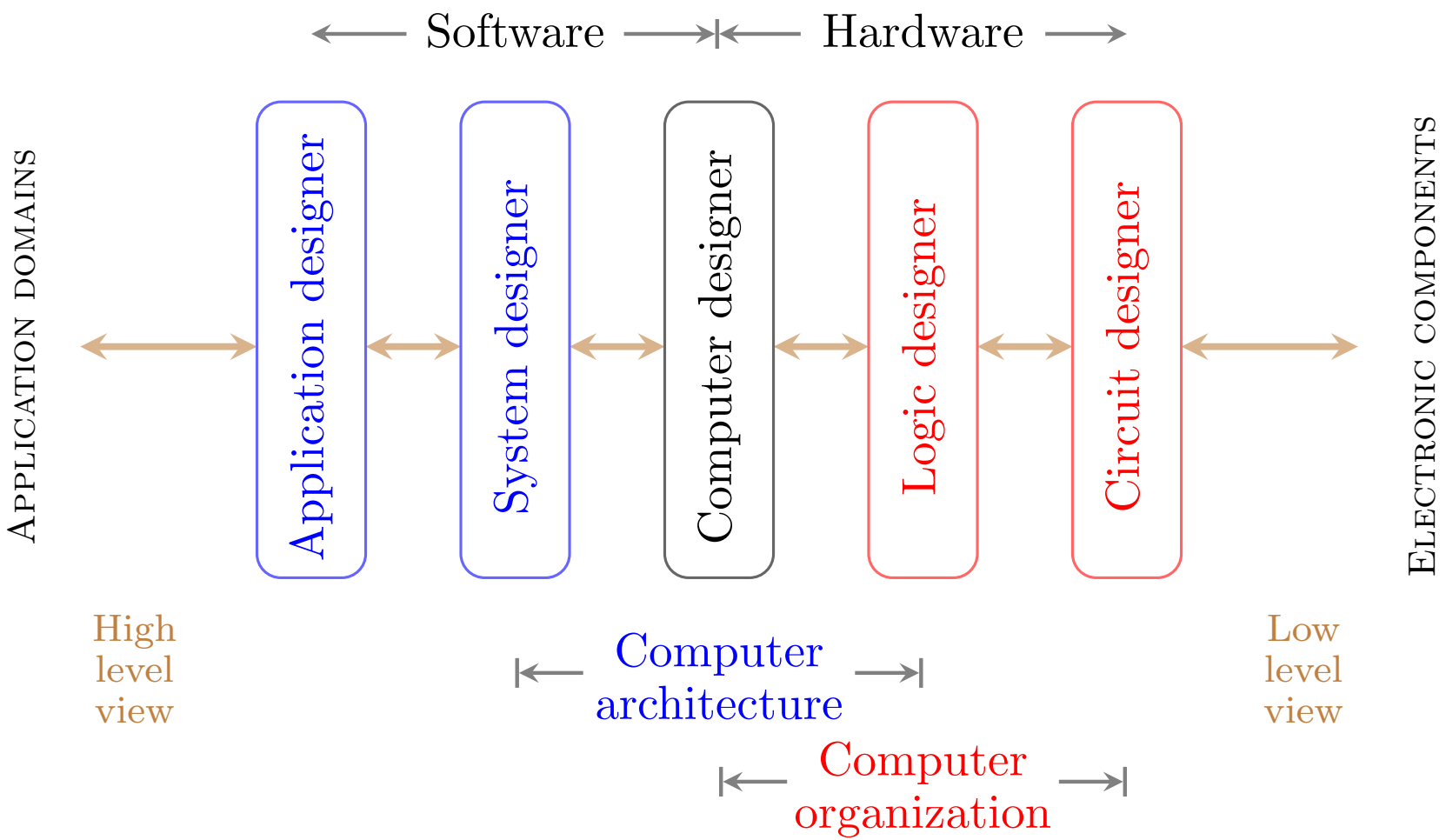


Faculty of Mathematics and Informatics
Hanoi University of Science and Technology

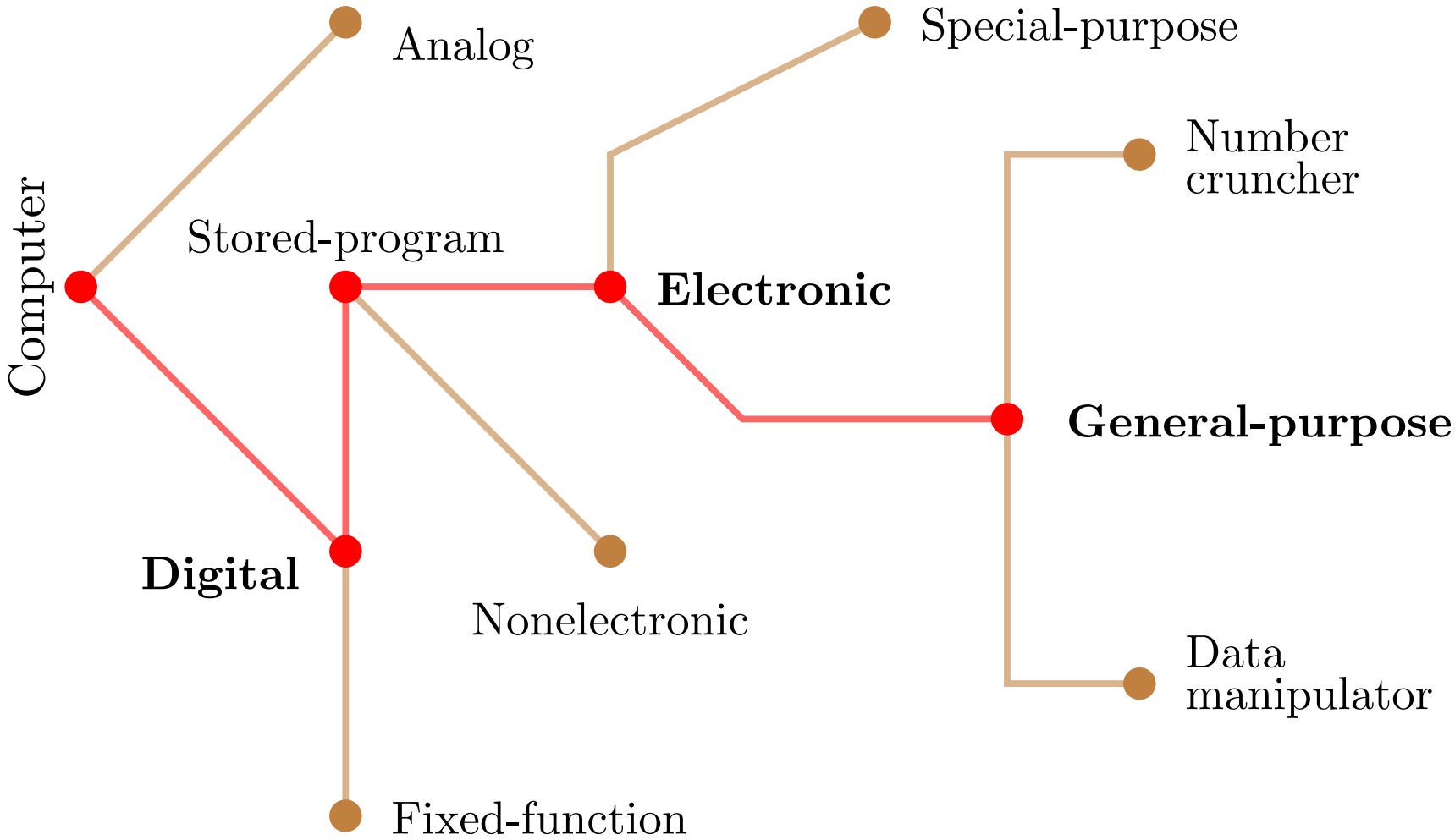
Outline

1. The origins of programming languages

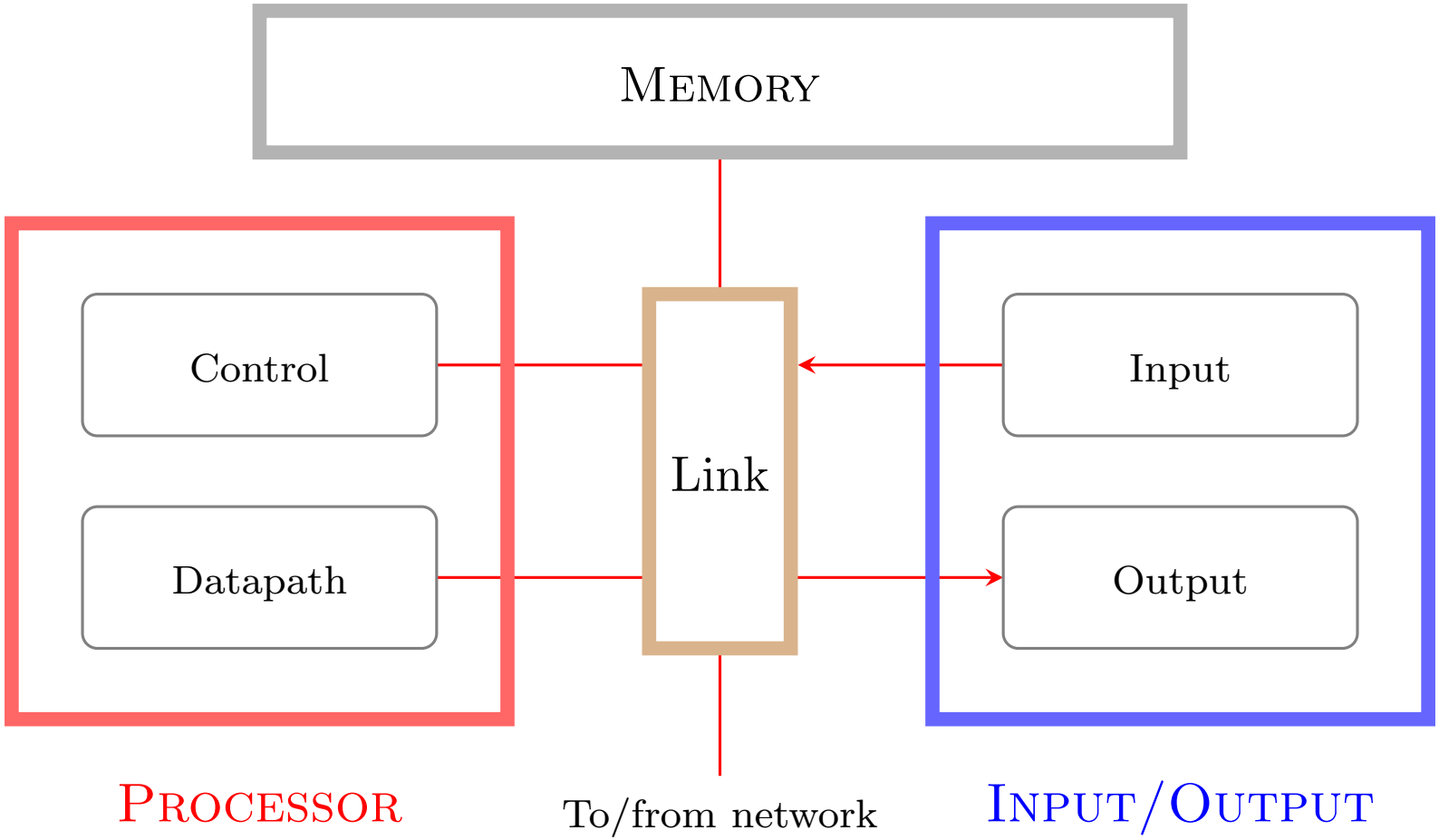
Views in computer system engineering



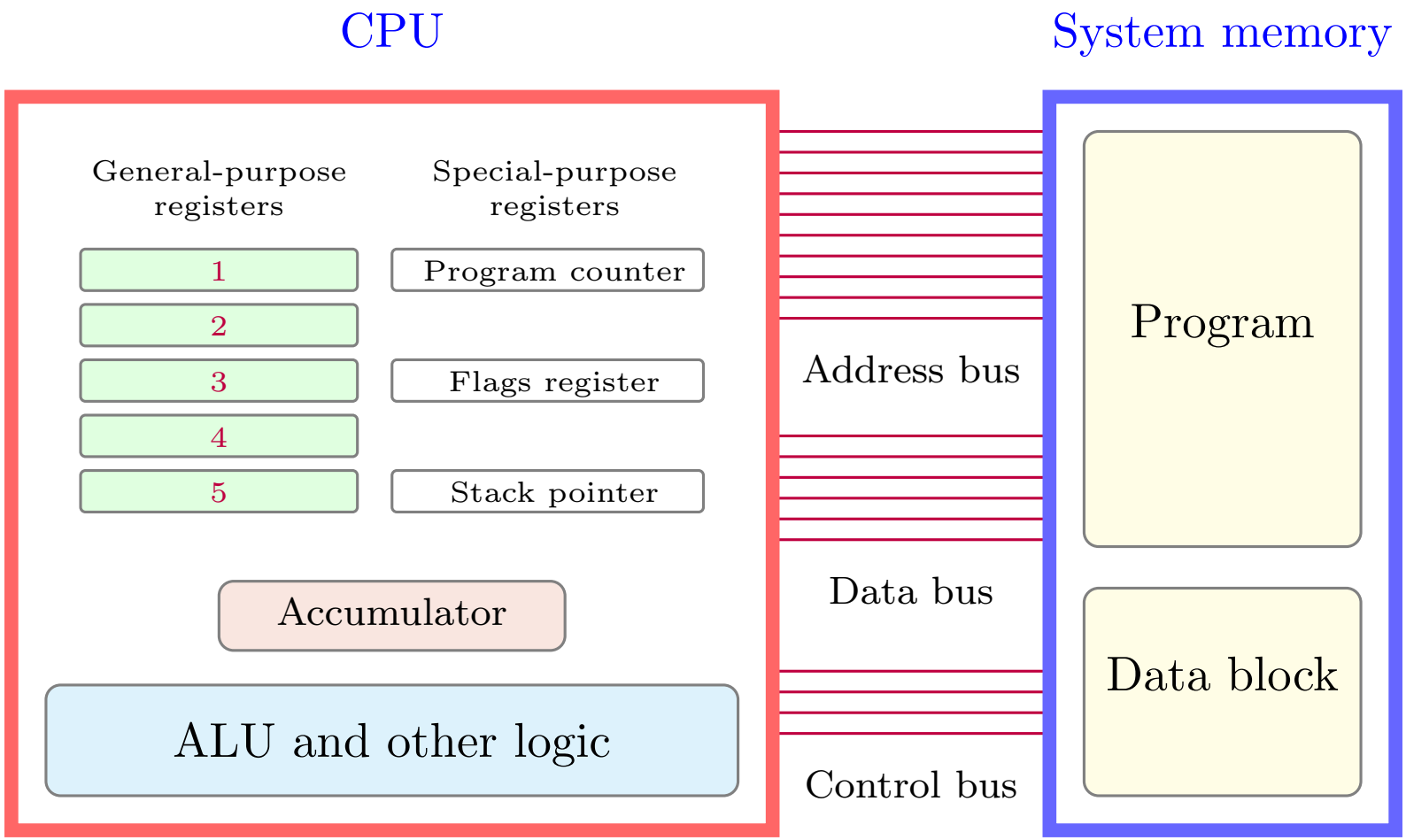
The world of computer hardware



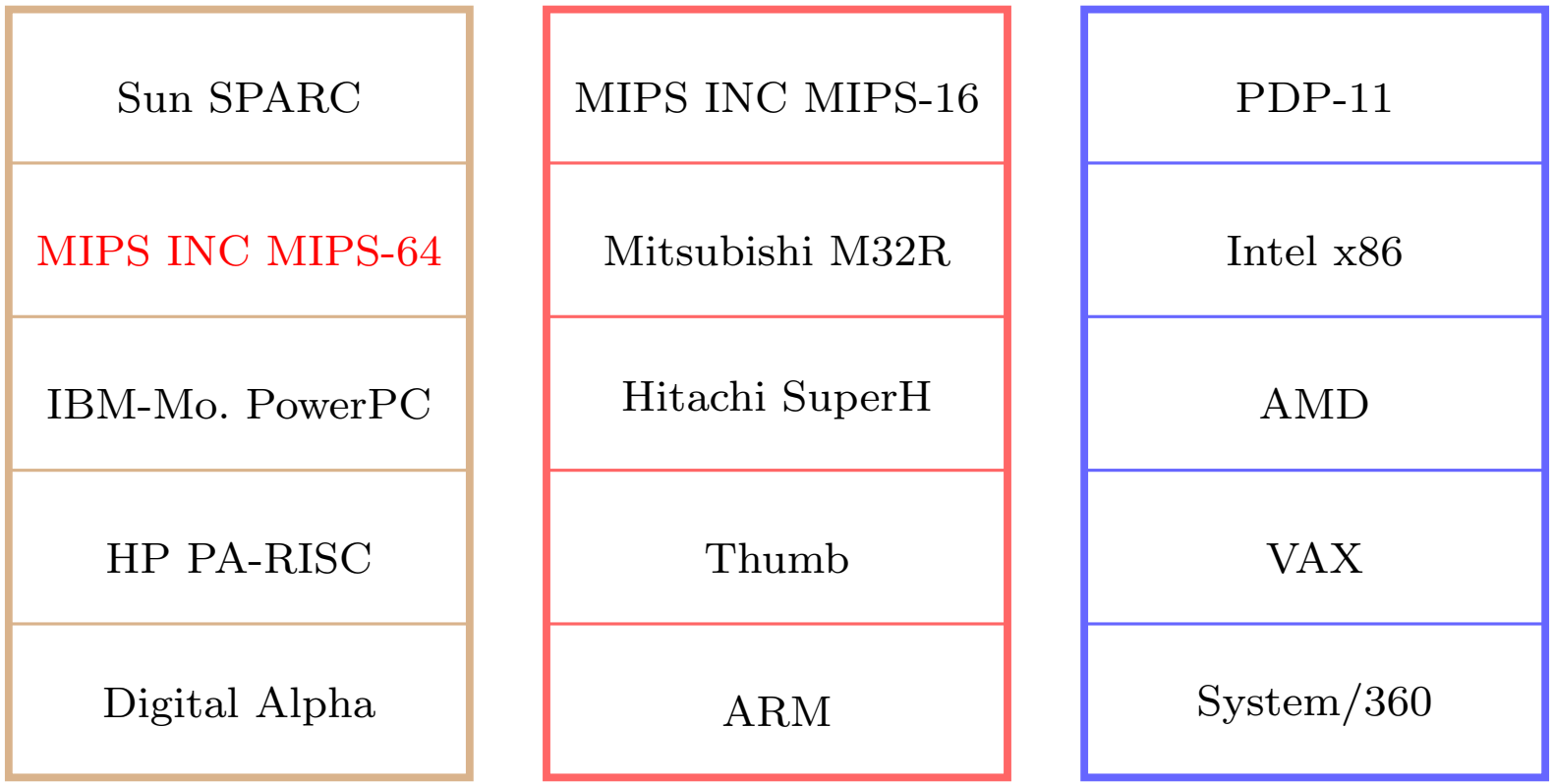
The (three to six) main units of a digital computer



A simplifed modern computer



Instructions: language of the computer

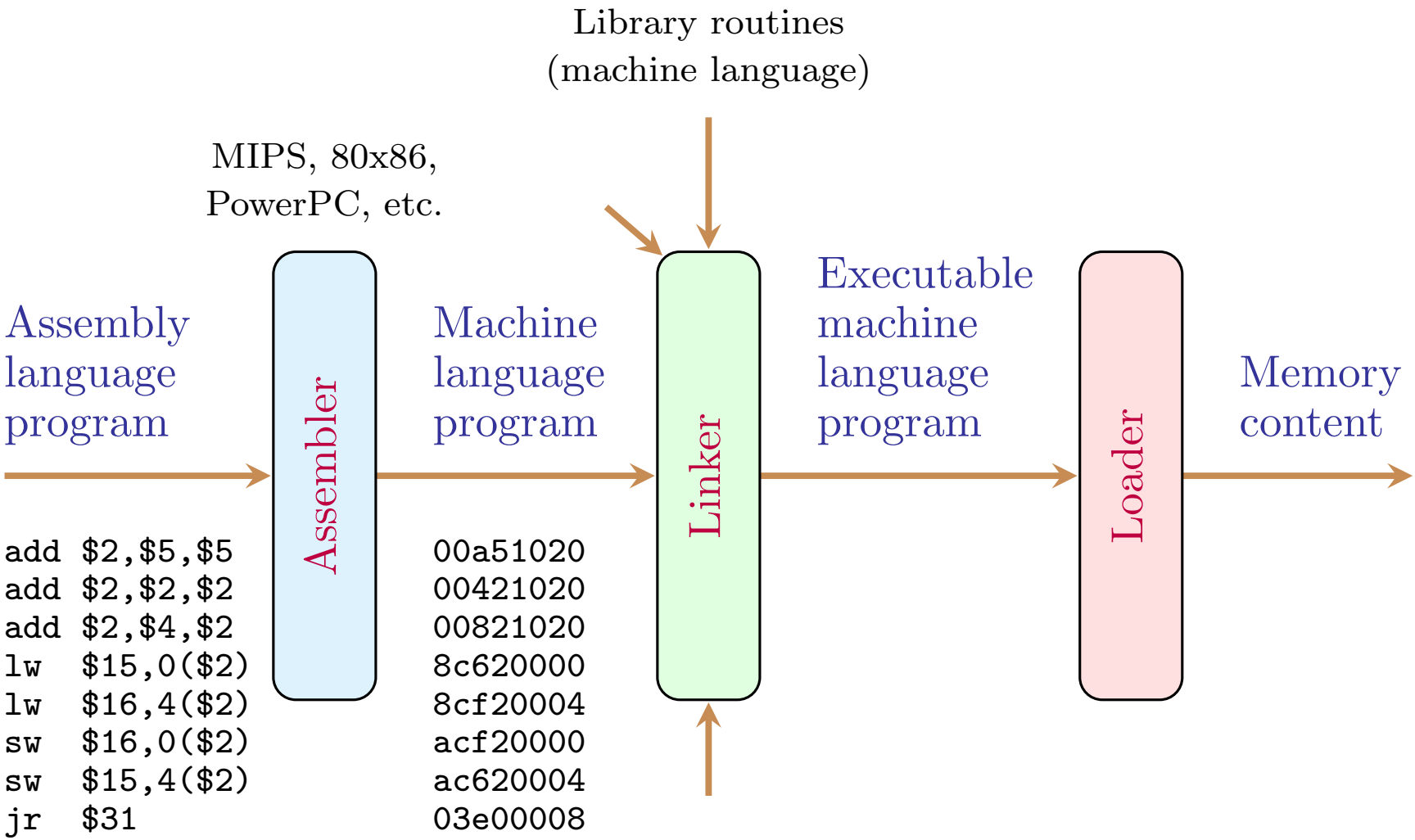


RISC Instruction Set (desktops and servers)

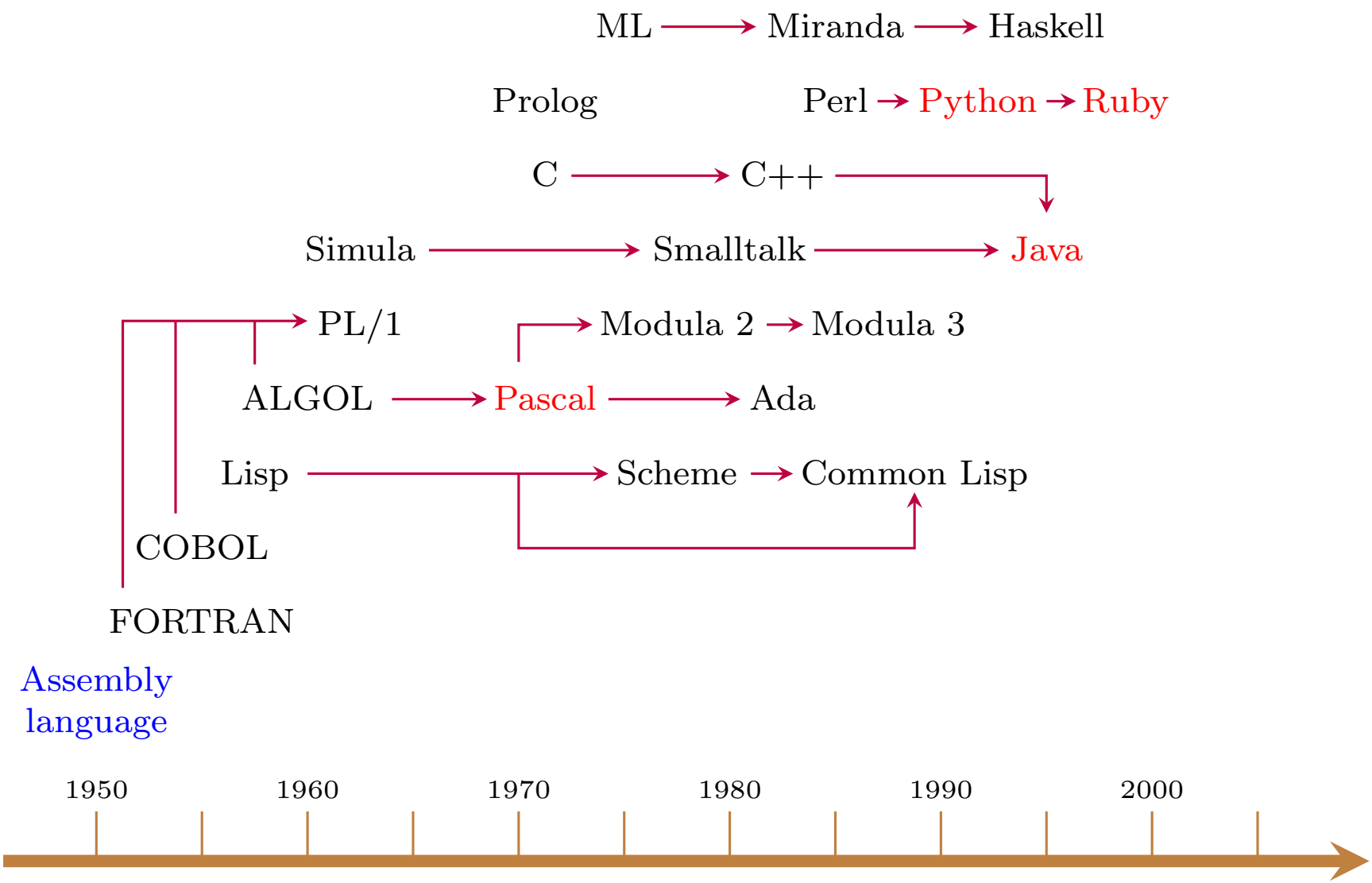
RISC Instruction Set (embedded computers)

CISC Instruction Set

Assembly and machine languages



A programming language timeline



The origins of programming languages

Can anyone teach us
how to write and run some simple programs
in your favorite languages?

Your first assignment

Students work individually to carry out the following tasks:

1. Design a simple calculation program that allows users to add, subtract, multiply and divide two integer numbers
2. Give a brief description of the program and its flowcharts
3. Implement your program in Assembly, Pascal, Python, JavaScript and C#
4. Compare all obtained implementations and give your own judgments (on their running performance, language features, etc.)
5. Report (in Word or Latex) and share your progress and results

NOTE ⇝ The report should have sections such as Description, Flowchart, Implementation, Comparison and Judgment, and relevant sections. Please convert your report to PDF when you submit it

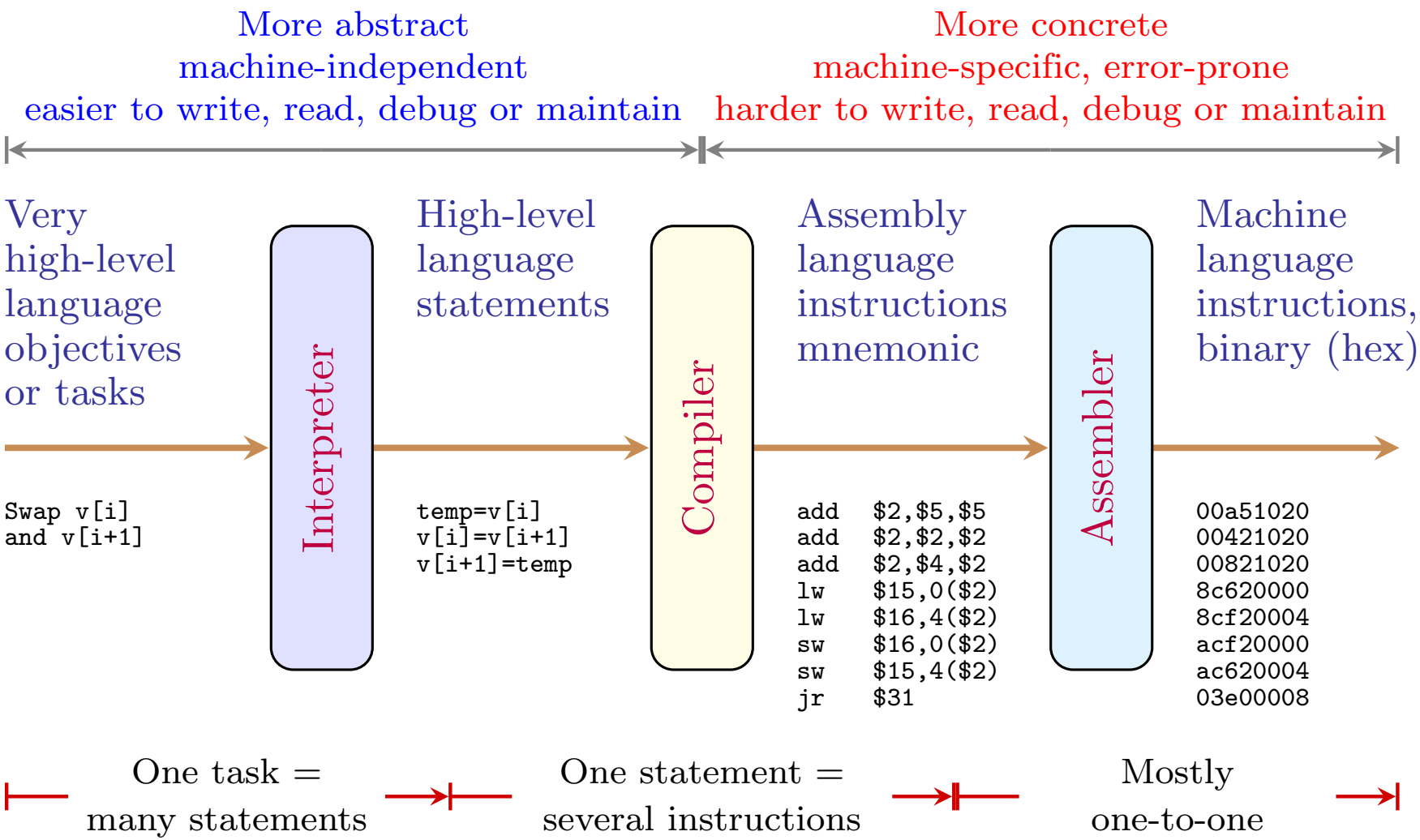
2. Abstractions in programming languages

Abstraction concepts and principle

- *Abstraction* is a mode of thought by which we concentrate on general ideas rather than on specific manifestations of these ideas
- In systems analysis, abstraction is the discipline by which we concentrate on essential aspects of the problem on hand, and ignore all extraneous aspects
- In programming, abstraction alludes to the distinction we make between (a) *what* a piece of program does and (b) *how* it is implemented. A programming language itself consists of constructs that are ultimately abstractions of machine code

NOTE ⇝ Here, we define an abstraction to be an entity that embodies a computation. It is possible to construct abstractions over any syntactic class, provided only that the phrases of that class specify some kind of computation

Models and abstractions in programming

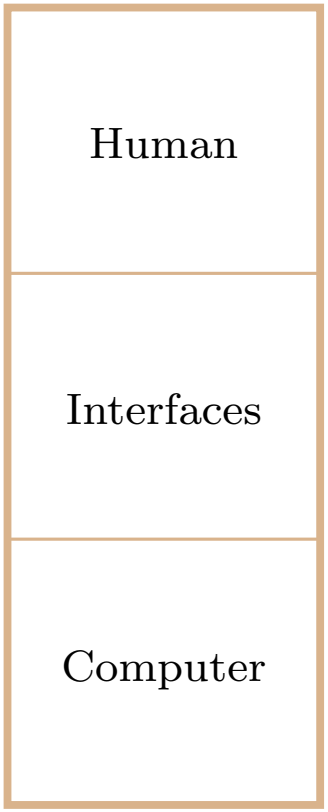


More concepts of programming abstractions

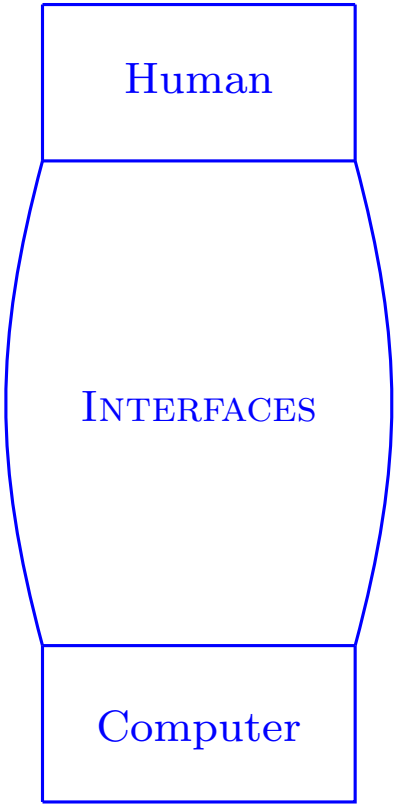
- *A function abstraction* is an abstraction over an *expression* (i.e. it embodies an expression to be evaluated, and when called will yield a result value)
- *A procedure abstraction* is an abstraction over a *command* (i.e. it embodies a command to be executed, and when called will update variables)
- *A selector abstraction* is an abstraction over a *variable access* (i.e. it has a body that is a variable access, and this will yield a reference to a variable)
- *A generic abstraction* is an abstraction over a *declaration* (i.e. it has a body that is a declaration, and this will produce bindings)

3. Computational paradigms

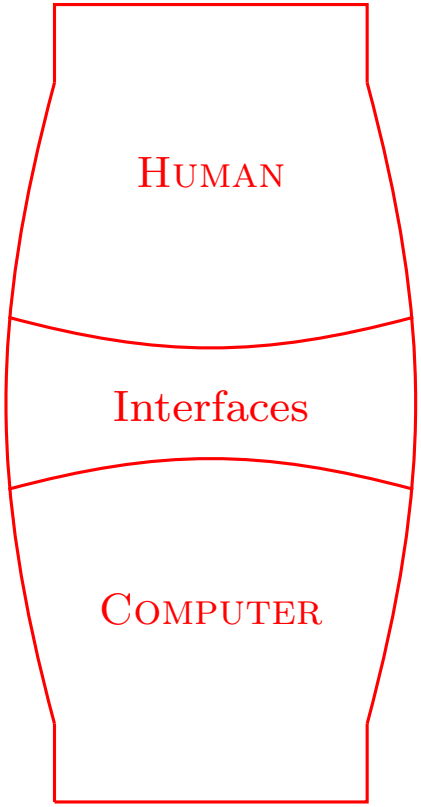
Different views of computational paradigms



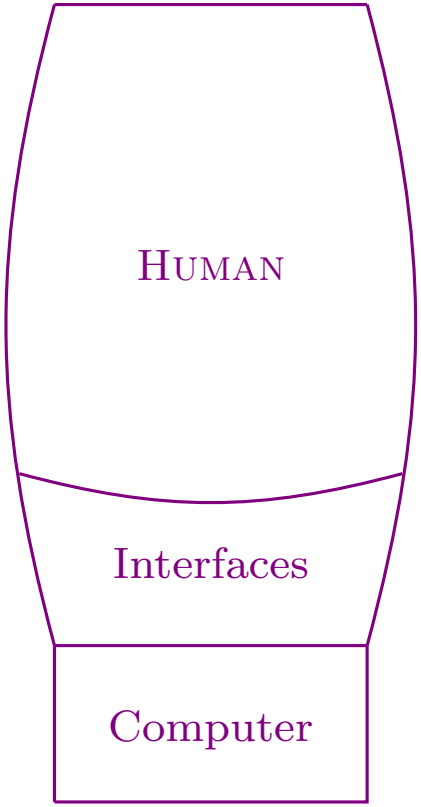
Textbooks



Wiki

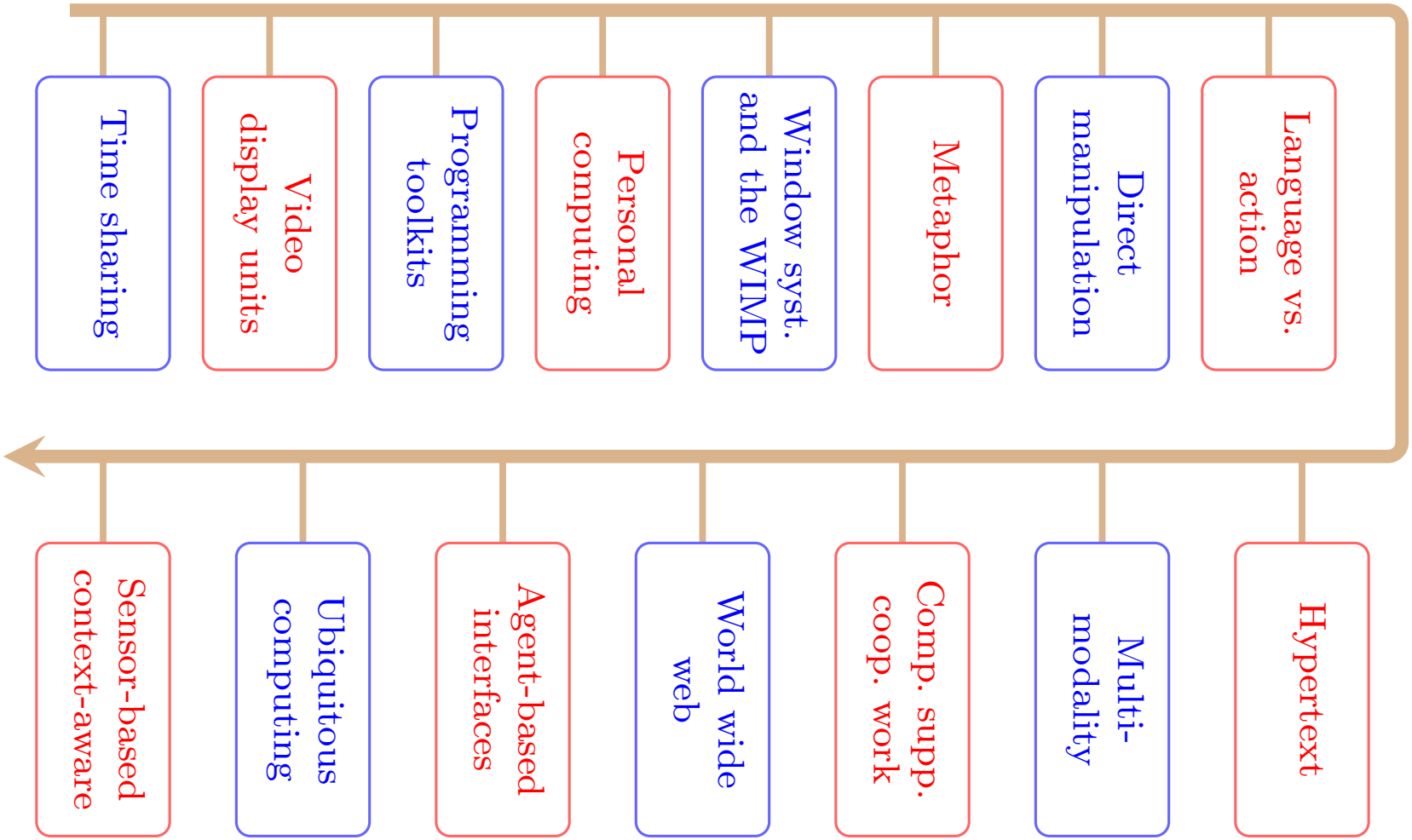


ACM



Business

Computational paradigms for UI/UX



The Internet **does not provide services**.

Instead, the Internet **only provides communication**, and

application programs provide all services!

Two Internet communication paradigms

Stream paradigm	**Message paradigm**
Connection-oriented	Connection-less
1-to-1 communication	Many-to-many communication
Sequence of individual bytes	Sequence of individual messages
Arbitrary length transfer	Each message limited to 64 Kbytes
Used by most applications	Used for multimedia applications
Built on TCP protocol	Built on UDP protocol

Client/Server model of interaction

- Used by applications to establish communication
- One application acts as a *server*
  - Starts execution first
  - Awaits contact
- The other application becomes a *client*
  - Starts after server is running
  - Initiates contact
- Important concept: once communication has been established, data (e.g. requests and responses) can flow in either direction between a client and server

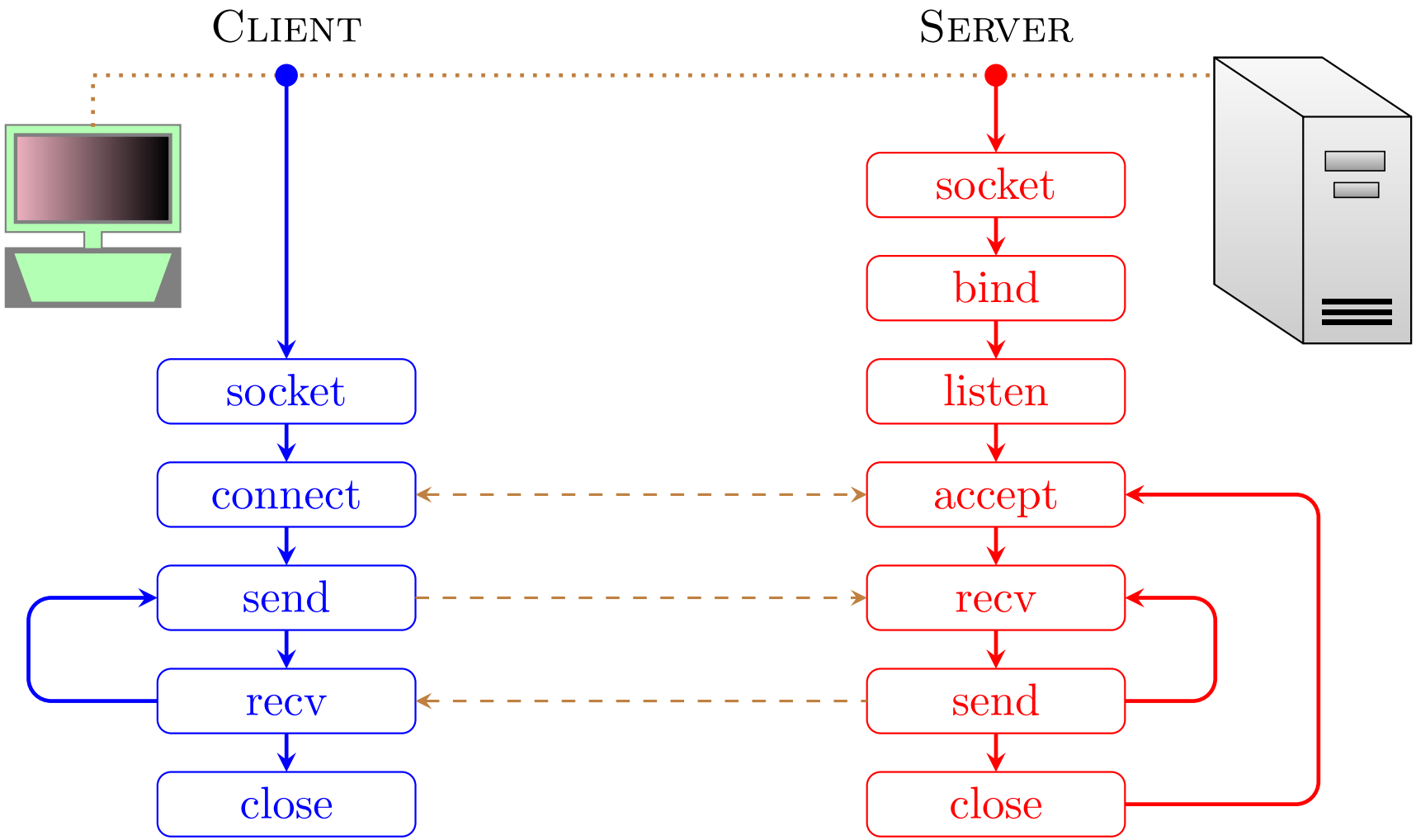
Network or Internet programming

- General term that refers to the creation of client and server applications that communicate over a network
- Programmer uses an *Application Program Interface* (*API*)
  - Set of functions
  - Include control as well as data transfer functions (e.g. establish and terminate communication)
- Defined by the operating system; not part of the Internet
- *Socket API* has become a de facto standard

Some simplified API

Operation	**Meaning**
await_contact	Used by a server to wait for contact from a client
make_contact	Used by a client to contact a server
appname_to_appnum	Used to translate a program name to an equivalent internal binary value
cname_to_comp	Used to translate a computer name to an equivalent internal binary value
send	Used by either client or server to send data
recv	Used by either client or server to receive data
send_eof	Used by both client and server after they have finished sending data

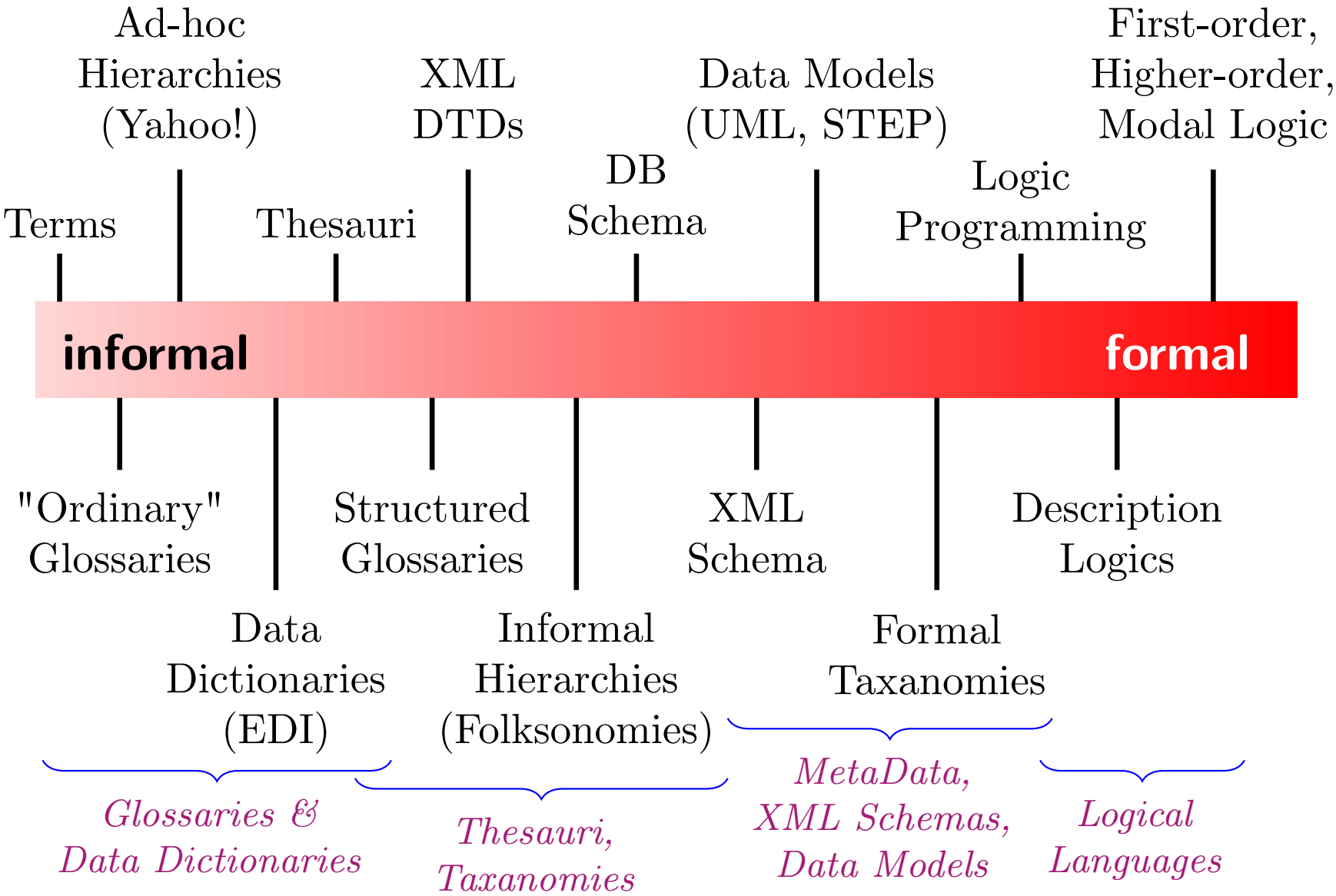
Example socket calls for stream communication



Exercise 1.1 Execute client/server programs in the zipped file, namely **client-server-app.rar**, and report the results

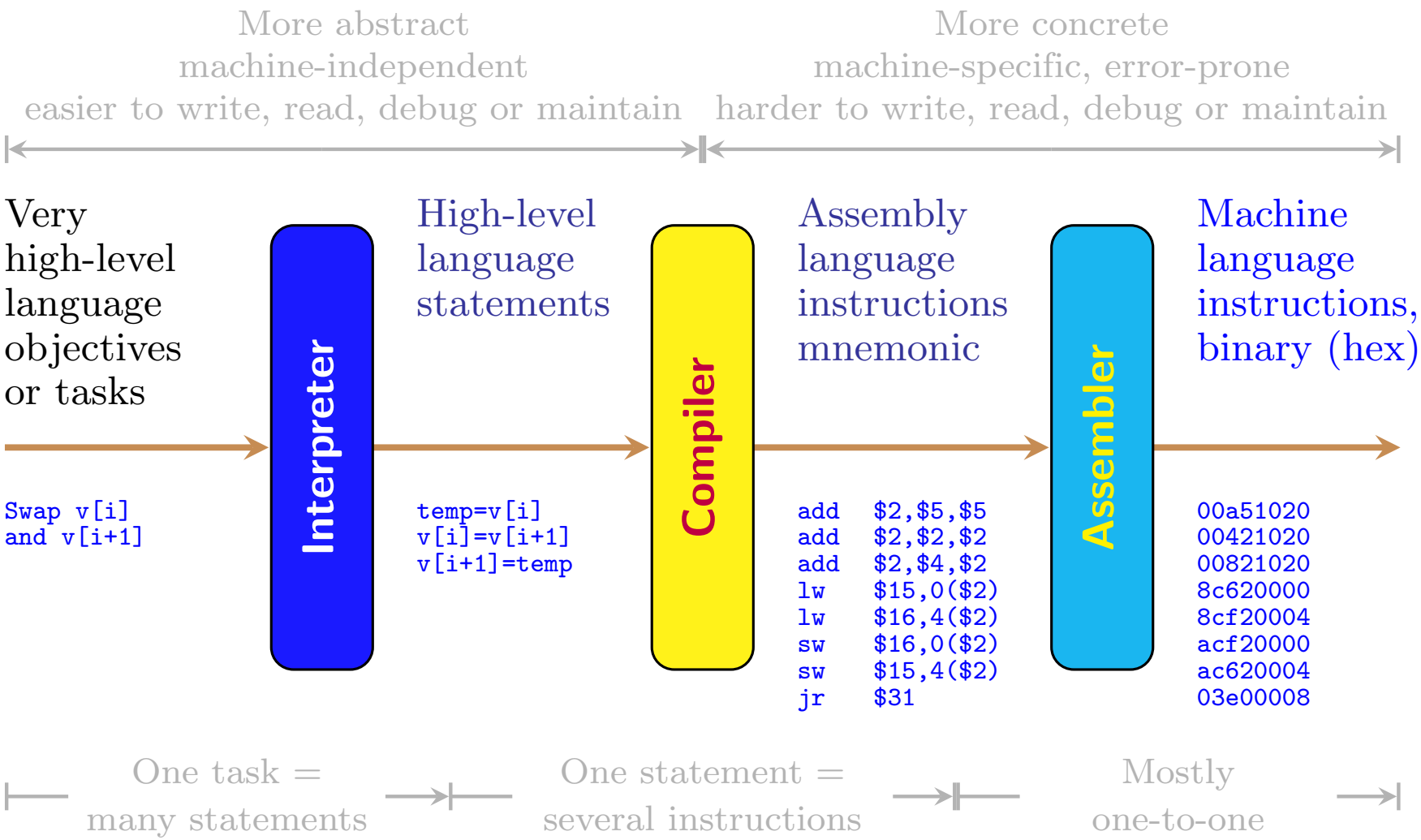
4. Language definition

Means of knowledge representation



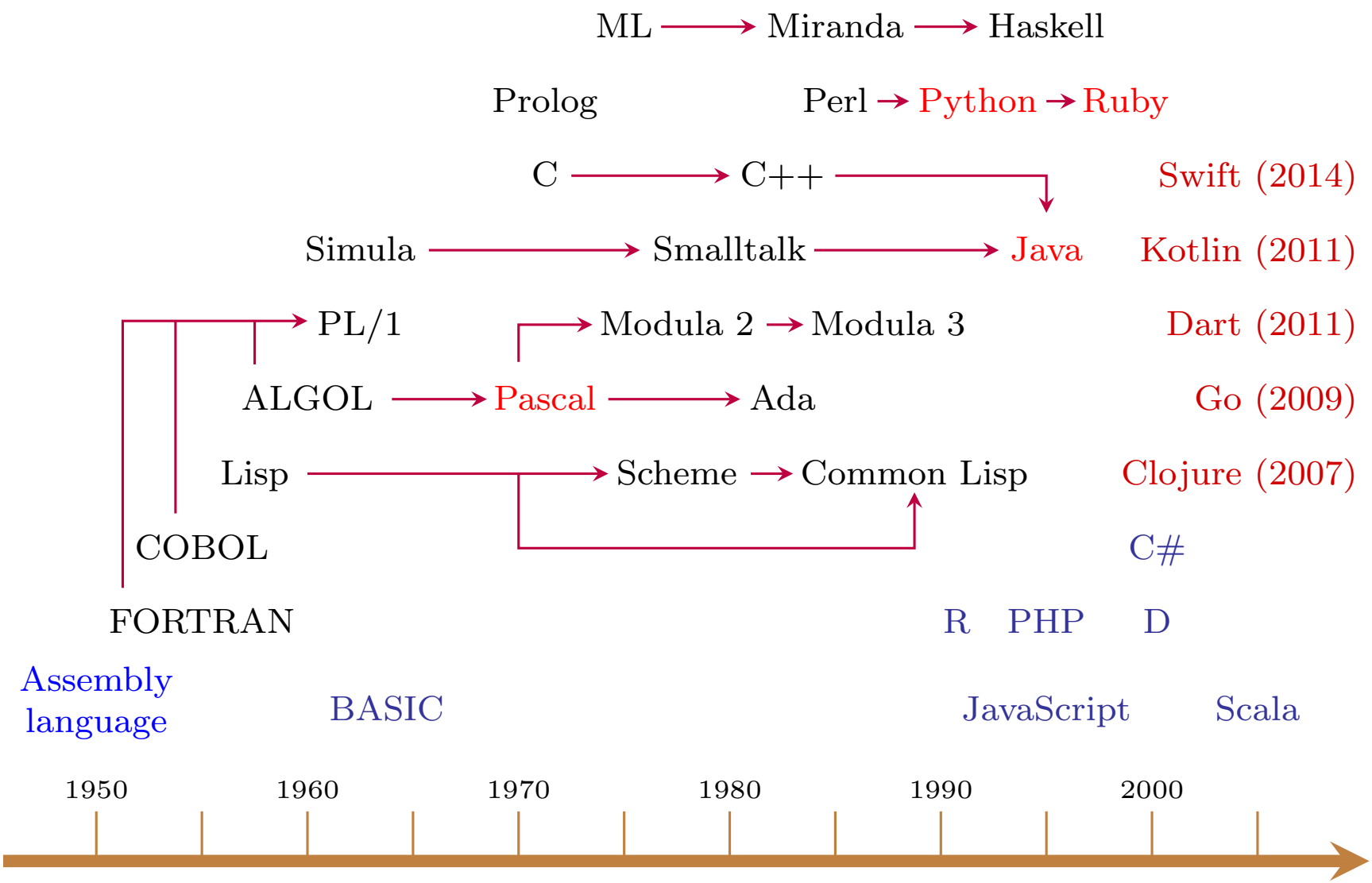
5. Language translation

Language translation

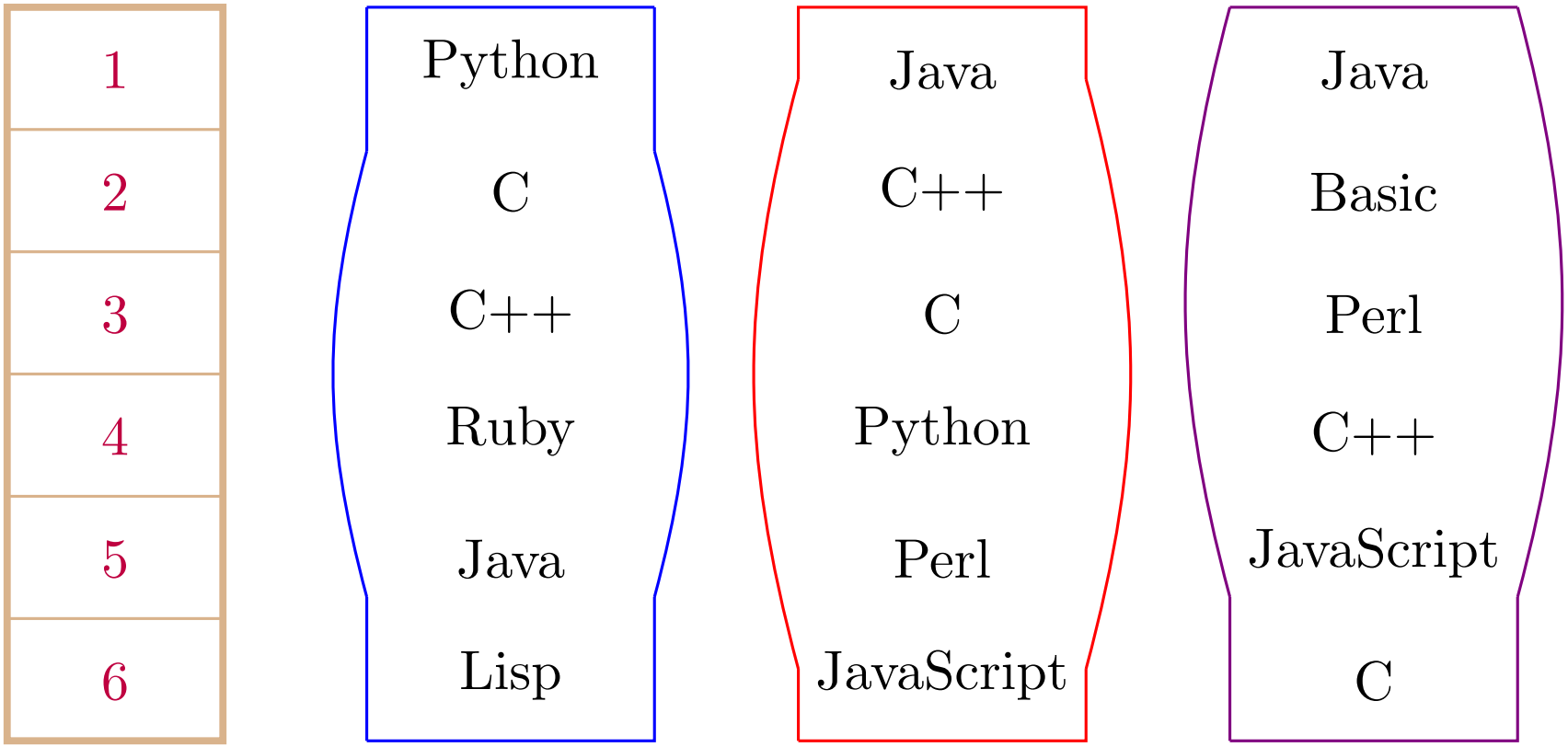


6. The future of programming languages

The future of programming languages



The future of programming languages



The future of programming languages

Can you share with us
your opinion about
the future of programming languages?

Thank you very much for your attention!