JSM: A small Java Processor Core for Smart Cards and Embedded Systems

Frank Golatowski,
Hagen Ploog, Nico Bannow,Dirk Timmermann

Institute of Applied Microelectronics and Computer Science
Department of Electrical Engineering and Information Technology, University of Rostock

Outline

- Goals
- Smart cards
- Java on smart cards
- Design and Functionality of Java processor
- Rapid-Prototyping-System
- Outview

Goals

- Development of a Java processor for small embedded systems, especially smart cards
- Increase of development time for smart card applications
- Speed up execution time of Javacard applications (Javacard applets)
- Usage in domains with security concerns
- Access to external devices
- Reducing chip´s die size and power consumption retaining high performance

Smart card

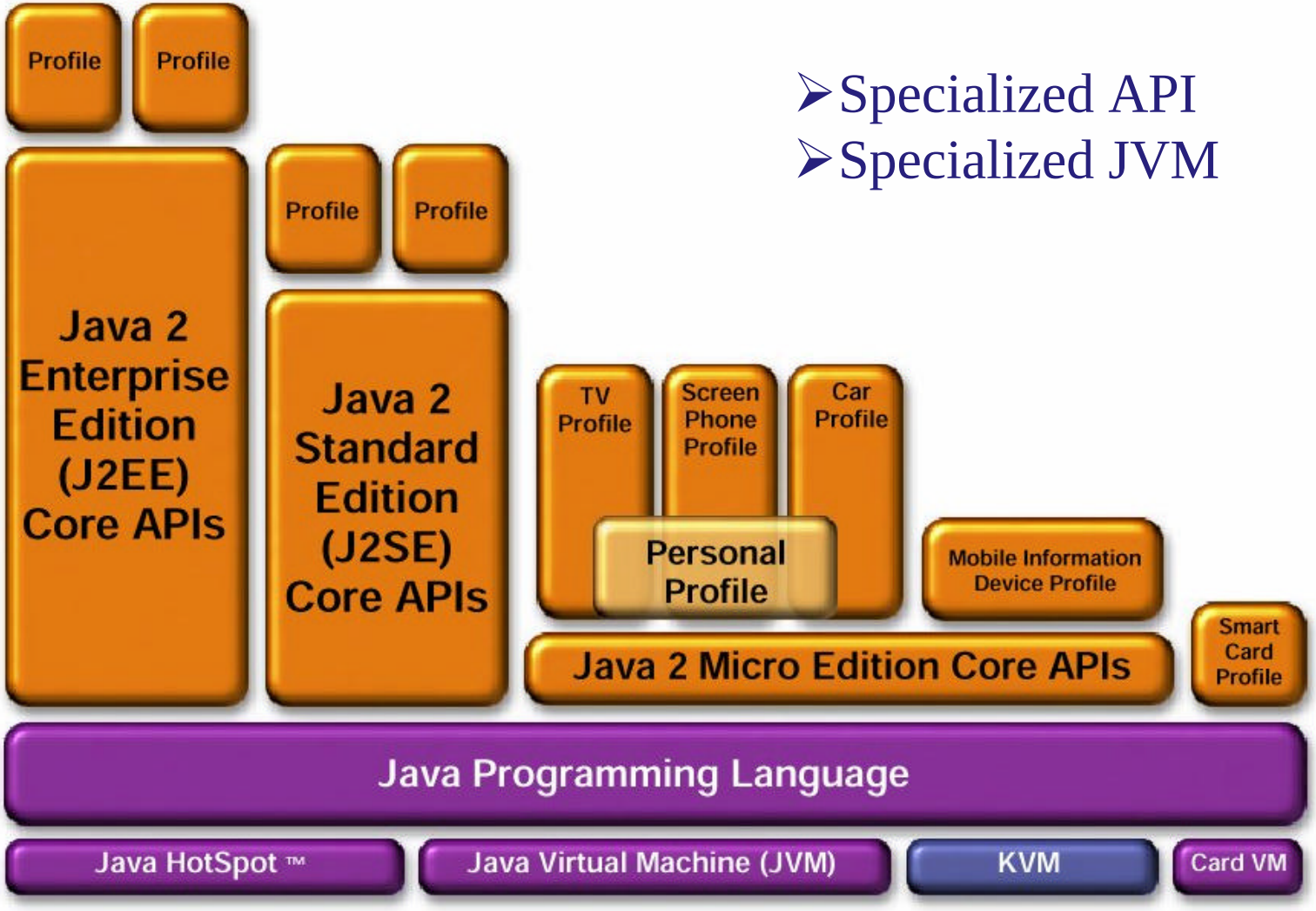
Plastic

SmartCard-Chip

Java Card

- Smart cards based on Java Card technology

Java2 platform



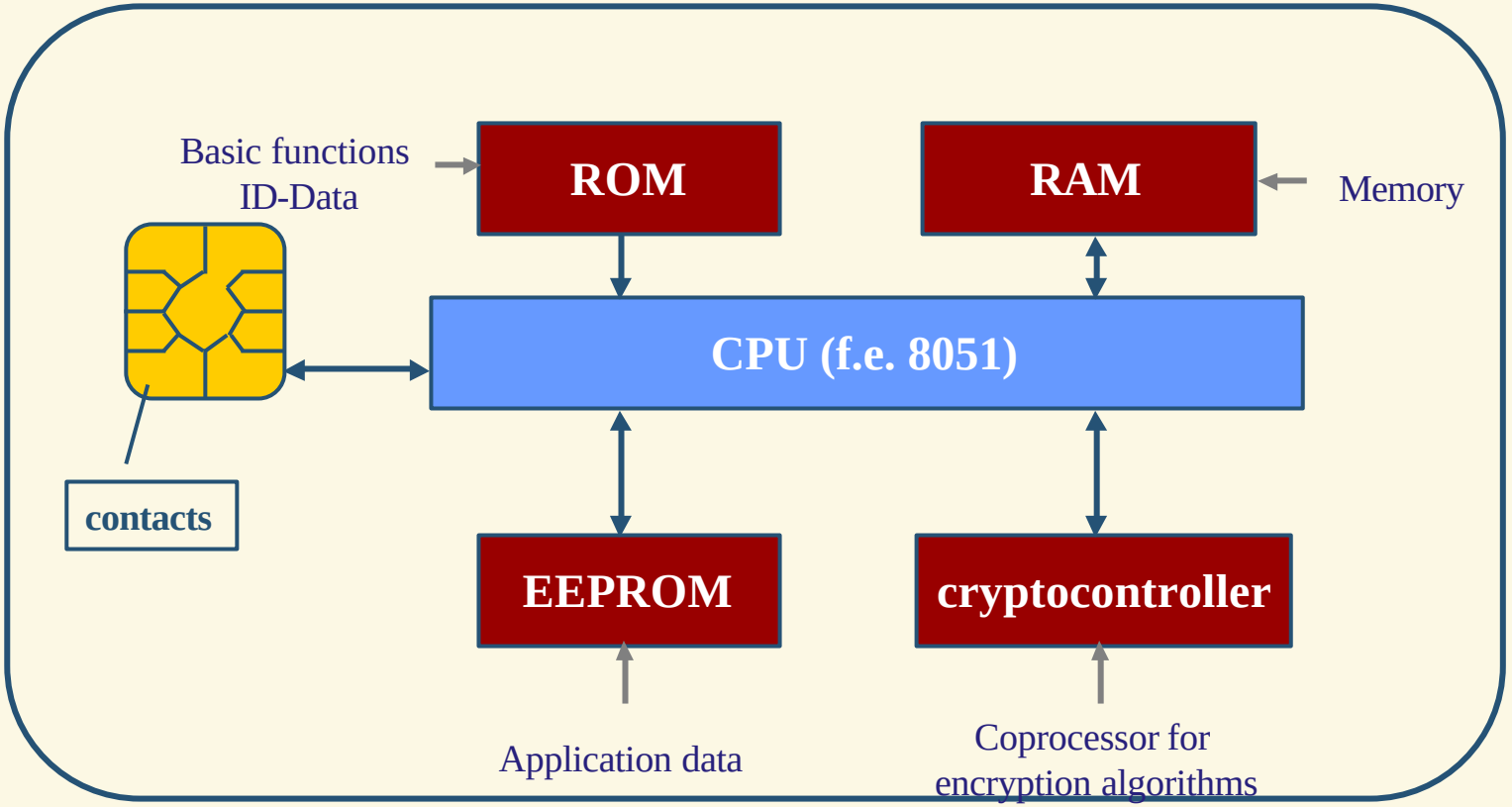
- Specialized API
- Specialized JVM

Application domains

- Financial services
- Health care
- Telecommunication
- Public entities
- Network security
- Retail
- Industry

- Authority and coupons
- Stored values
- Personal information
- Security
- Payment
- Access / Identification

Smart card microcomputer architecture



Advantages

- Security
- Easy to use

Disadvantages

- Application development is time consuming and often hard
- Applications for SC mostly written in Assembler and C
- Applications masked in ROM
  - Changing of applications not possible
- Applications not portable

One Solution

→Java on Smartcards

Javacard

...but with long run times

→ Our Solution: Java in Hardware

Advantages (Java Card)

- Secure value-added services in ubiquitous world
- Support additional services and applications after the card has been issued
- Rapid development of smart card applications

Why Java

- Write Once, Run Anywhere
  - Portability
- Security of Applications
- Usage of standard tools
- Multi application cards

Problems

- Limited resources

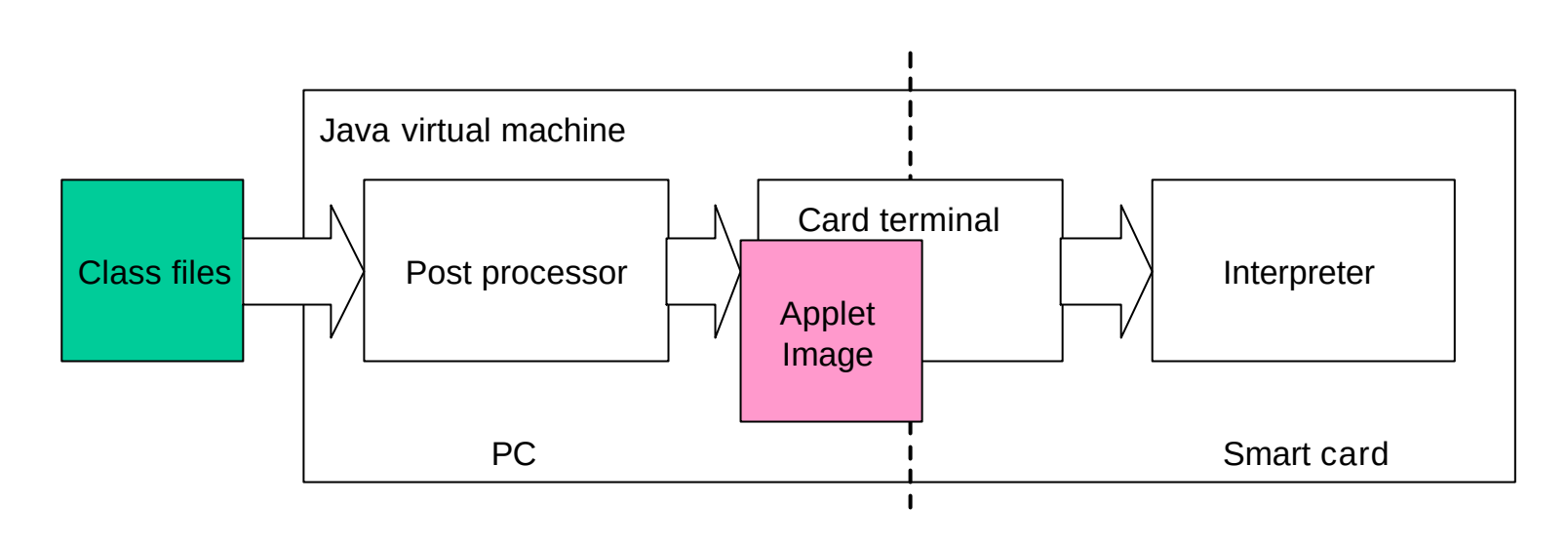
Solution

- Limited instruction set
- Split virtual machine

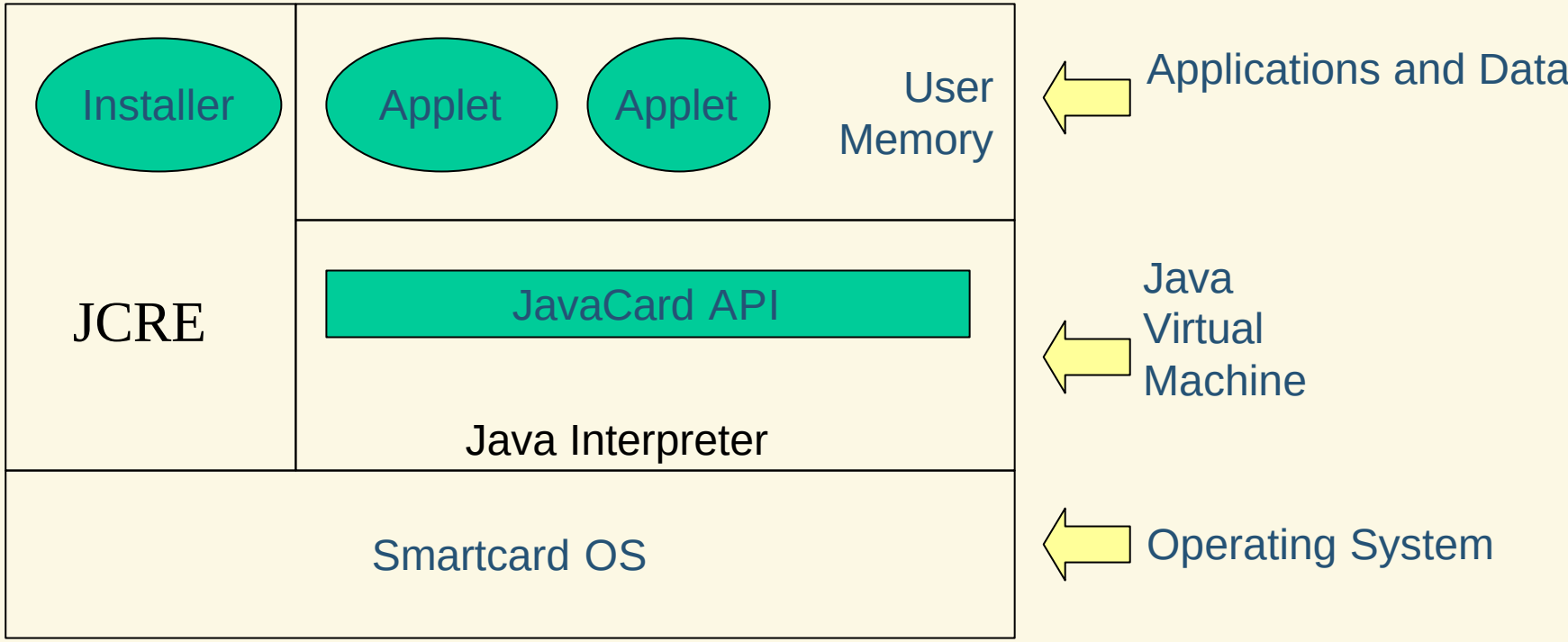
Java for smart cards: Limitations

- Adaptation of API and instruction set to the limited resources of smartcards
- Limit the size of classes, objects and stackframes
- Reload of Classes not possible
- Applets and Objects can not be deleted
- …but deviate from the minimum requirements smart cards can be extended at will

Split virtual machine



JavaCard


Installer
Applet
Applet
User Memory
Applications and Data
JCRE
JavaCard API
Java Virtual Machine
Java Interpreter
Smartcard OS
Operating System

Split virtual machine

- One part of the Java virtual machine executes off-card,
- Preparing the executable code that is executed in the other part of the VM, on the card
- Reasons:
  - ↓ size of applet image downloaded to the card
  - ↓ run-time memory requirement

Split virtual machine

Converter

Verifier

- Off-card:
  - Linking classes and resolving references
  - Converter
    - Produces applet image (*.cap – Files)
  - Verifier
    - Verify CAP-Files
  - Reading in all of the Java executable codes to produce applet image (Java Card loading format, CAP-File)
  - CAP-File may be downloaded to a Java Card
  - The code will then be executed by the on-card part of the JCVM
- On-card:
  - JCVM, JCRE, JC-API´s

Split virtual machine

- Off-card:

Converter

Verifier

  - Reading in all of the Java executable codes to produce applet image (Java Card loading format, CAP-File)
  - CAP-File may be downloaded to a Java Card
  - The code will then be executed by the on-card part of the JCVM
- On-card:
  - JCVM, JCRE, JC-API´s

Java Card-Security

Derived from the Java Programming Language

- Object-oriented programming language
- Name space mgmt. for type and procedure name
- Reuse if code that has already tested
- Strongly typed language
- Java byte code strictly related to specification
- No pointers as in C and C++
- Access to memory beyond allocated array boundaries not possible with array indices
- Java provides transparent storage allocation

Java Card-Security

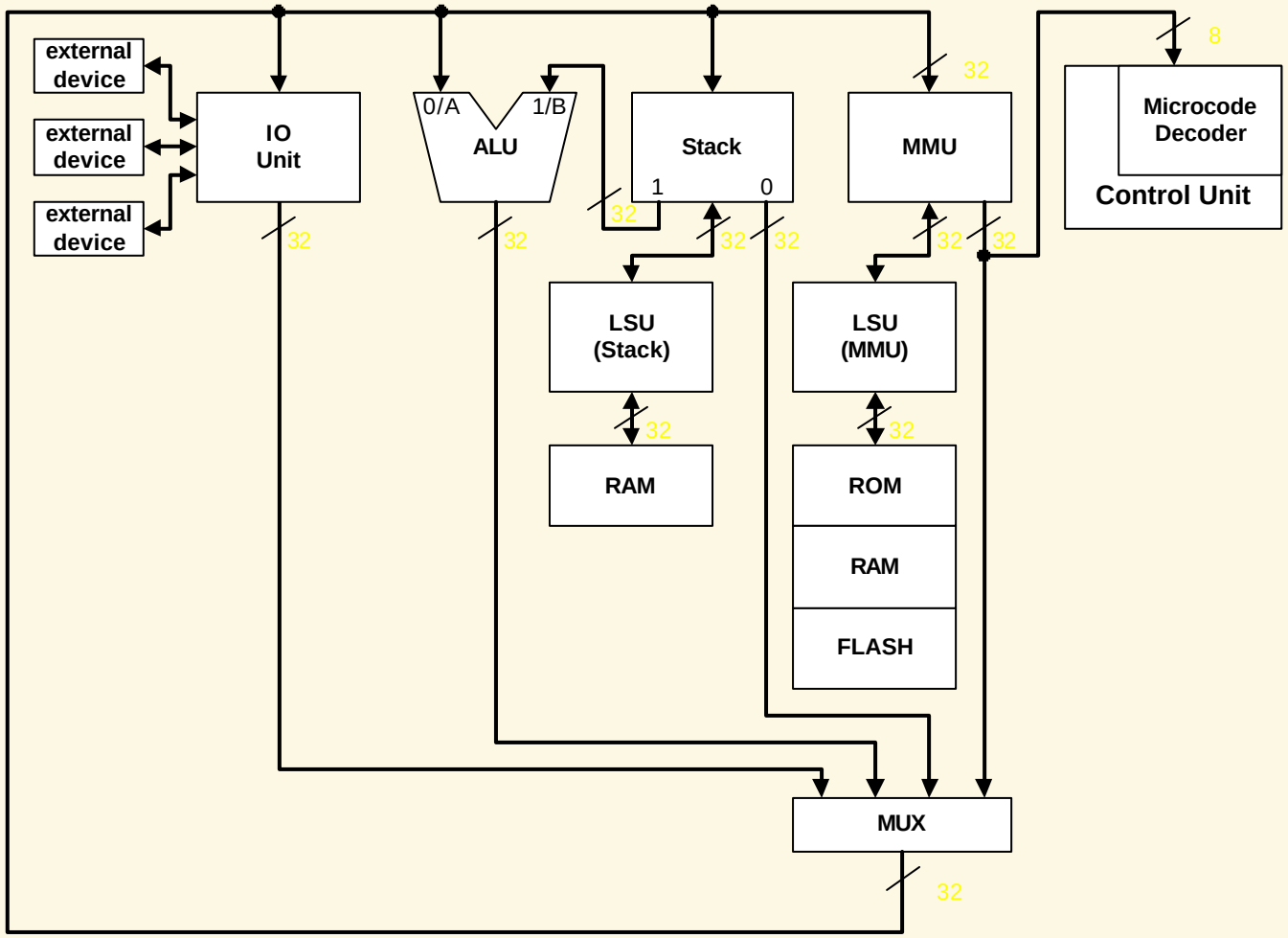
through Java Card platform security enhancements

- Transaction atomicity
- Applet firewall
- Security and cryptographic classes

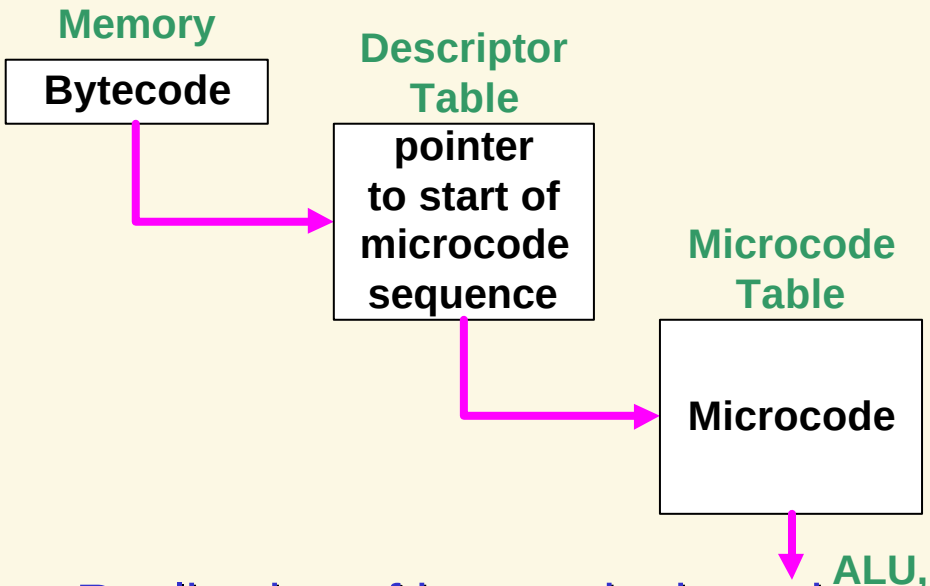
Features of JSM

- 32bit
- Synthesizeable Code
- Microcode control
- Stack machine
- Security tests in parallel to program execution
- Additional bytecodes
- Embedded Triple DES Crypto–Coprocessor

Design of JSM



Microcode controlled processor



Microcode
ALU, MMU, Stack, IO-Unit, Control-Unit

Realization of byte code through
RISC – alike Microcode instructions

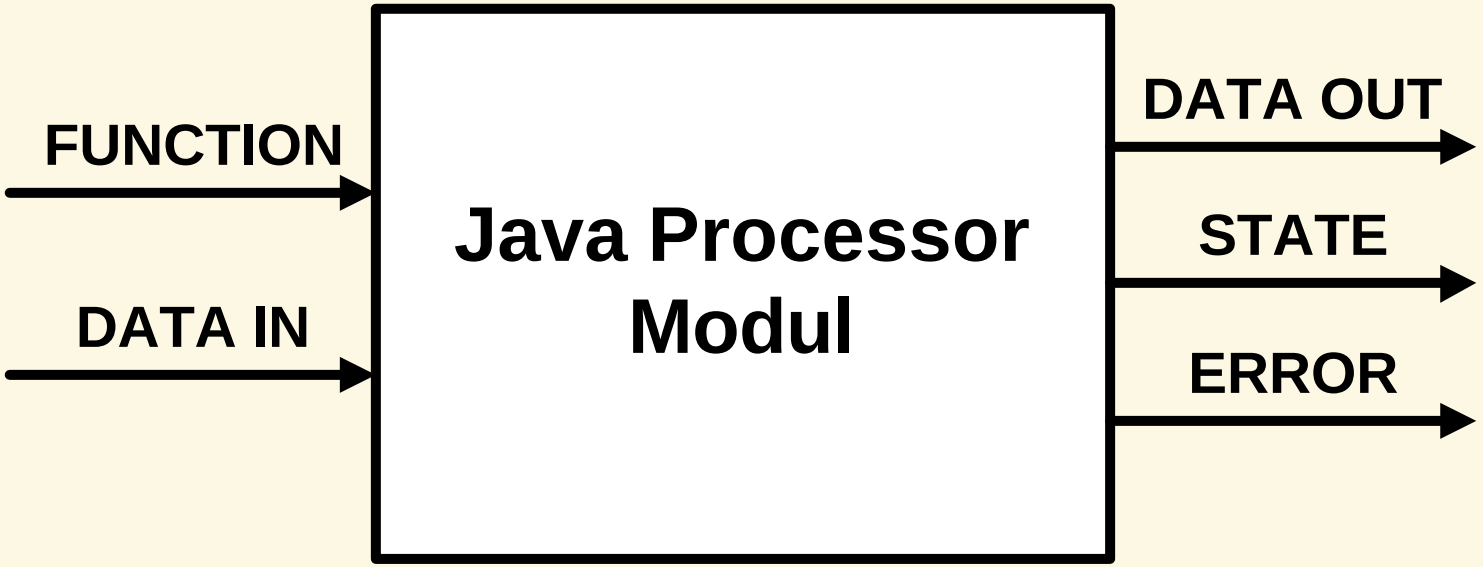
- Internally translation from bytecode to microcode
- Jumps, conditional jumps and execution of subroutines possible
- Very efficient regarding realization,execution speed, modifications
- Clearly arranged

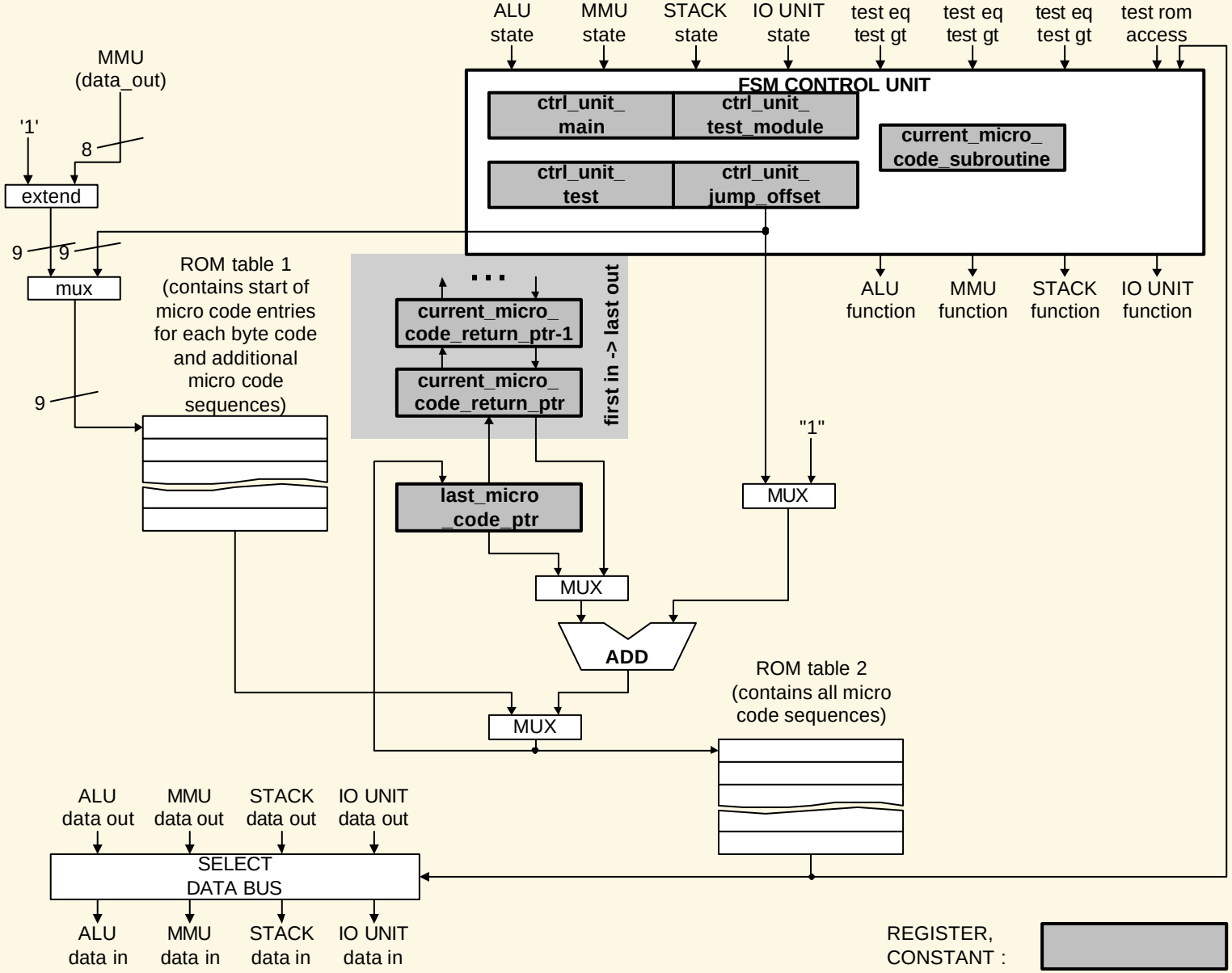
Software encapsulation vs Hardware encapsulation

- Software objects
  - states are defined by variables
  - modifications through operands and bytecodes
  - operands transmits data

- Hardware modules
  - states are defined by registers and memory
  - modifications through input signals and internal logic
  - output signals transmit data

General structure JSM´s modules

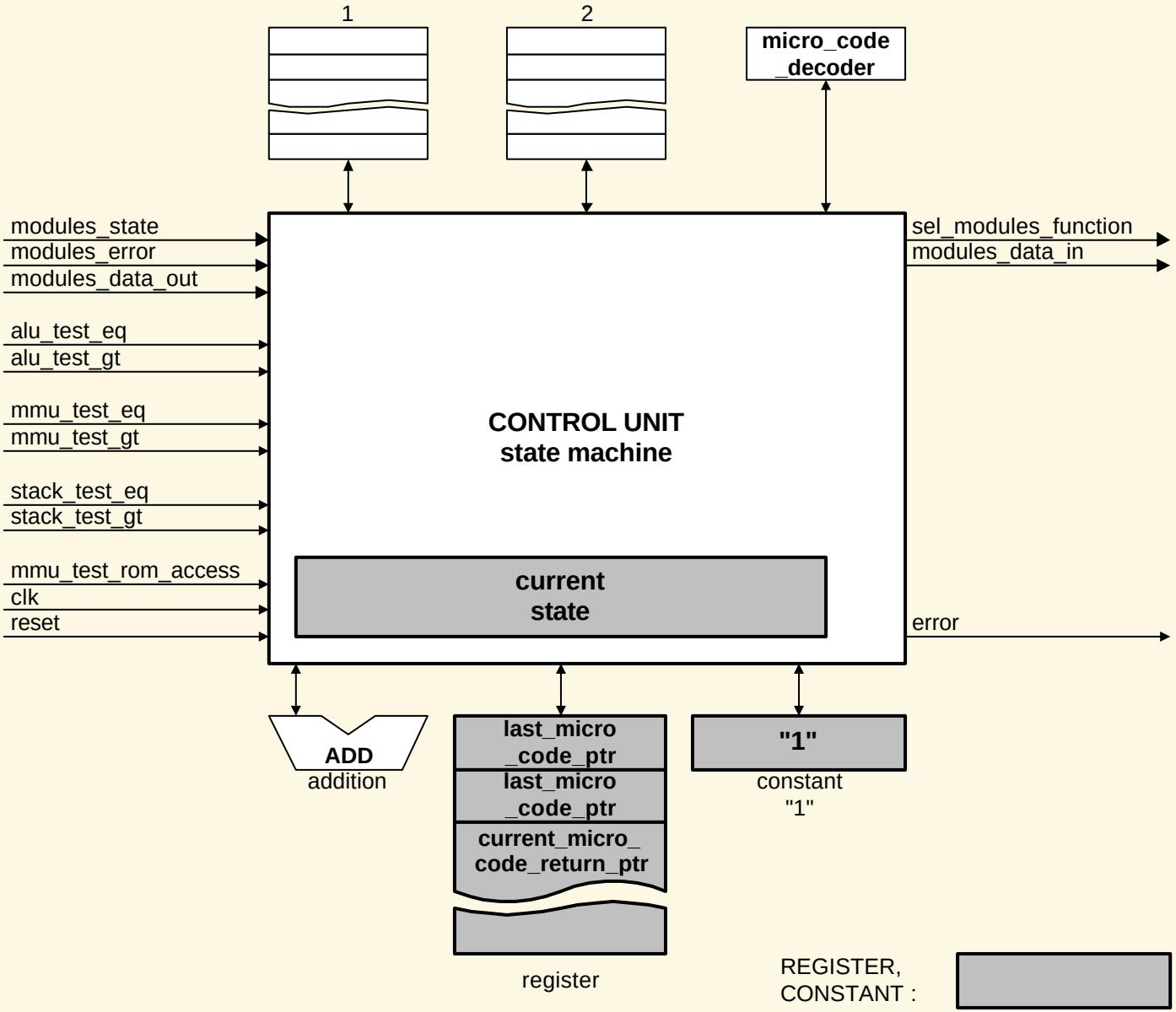




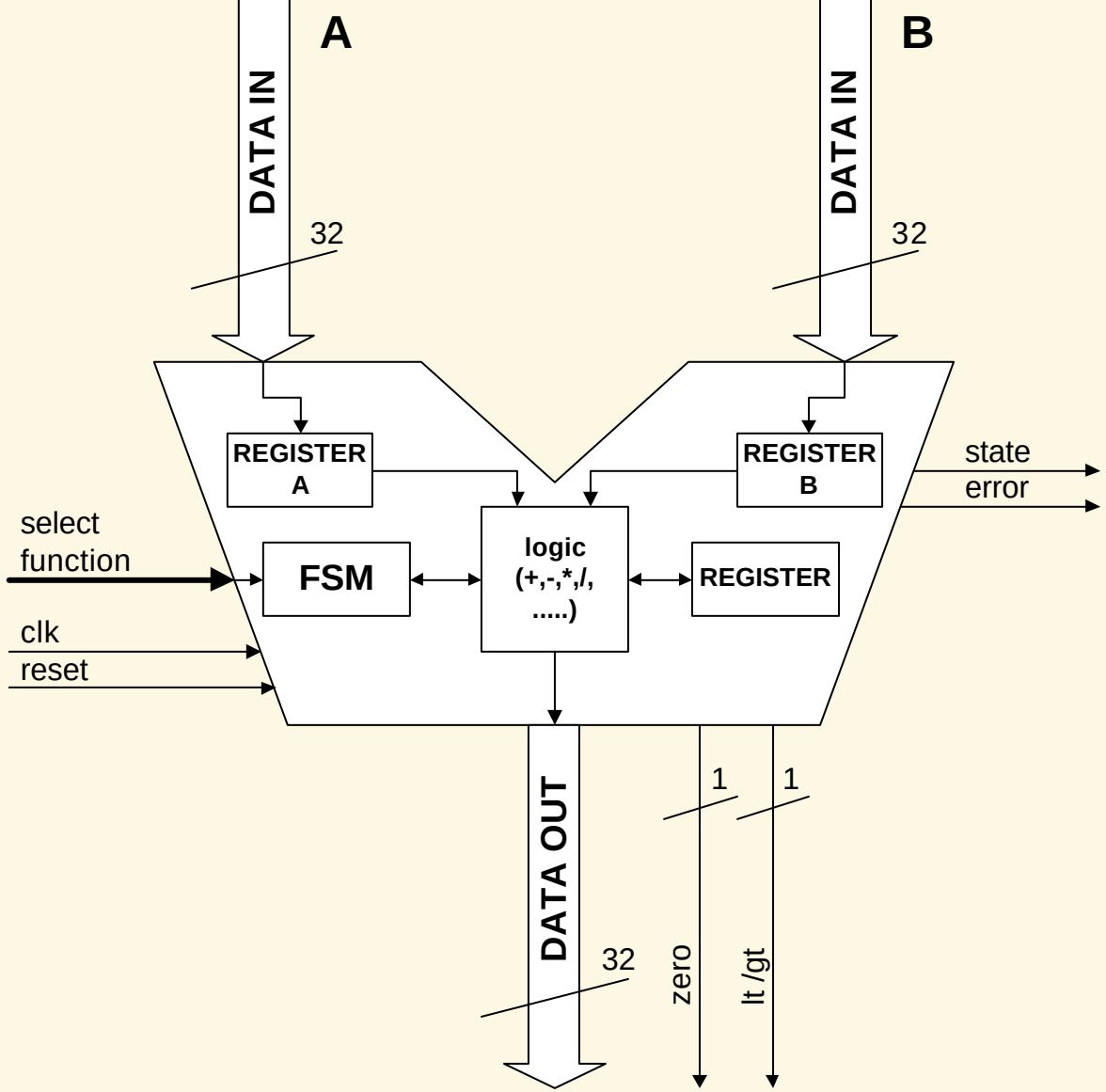
CU

Control Unit

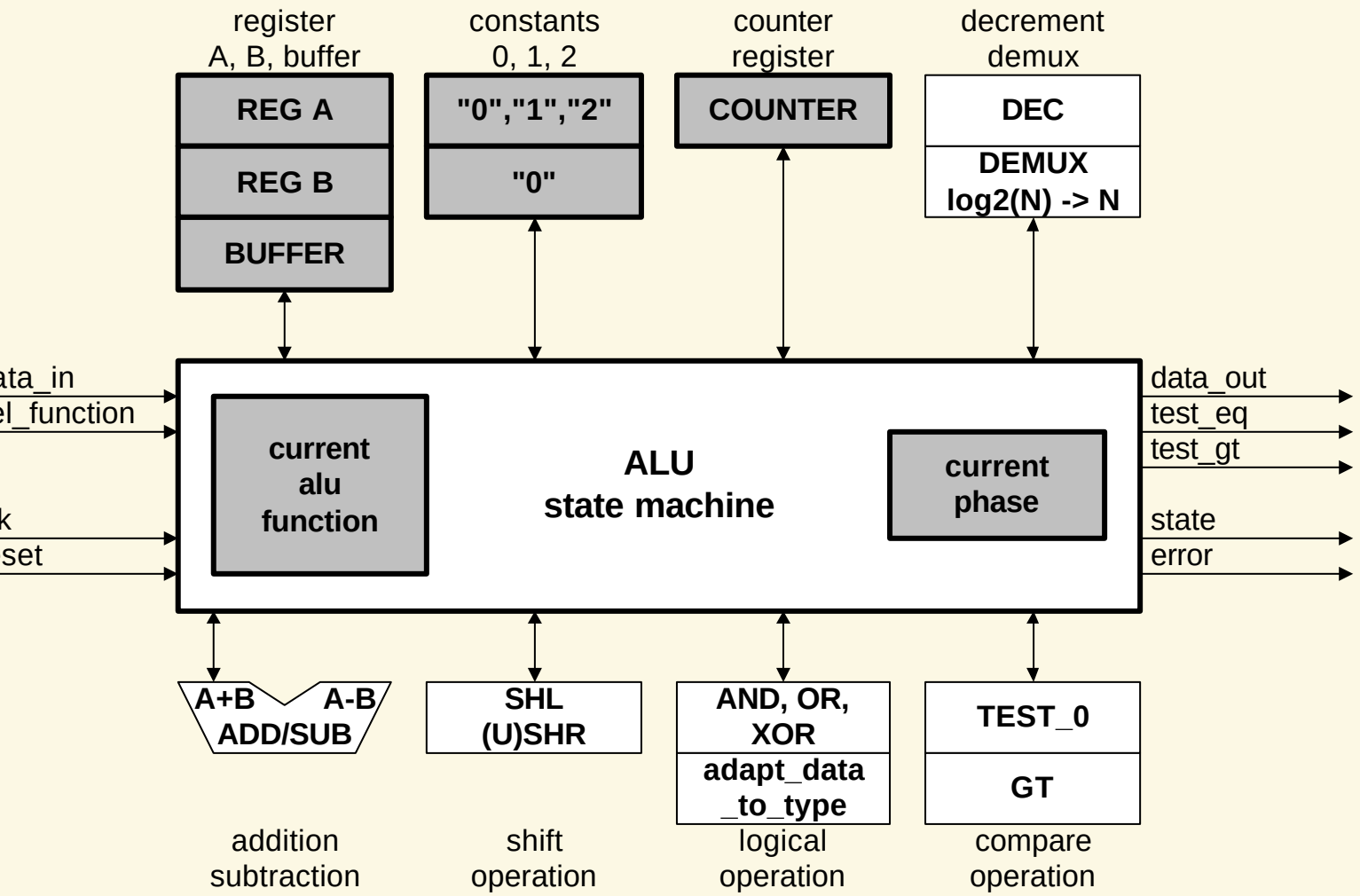
IO & Components CU



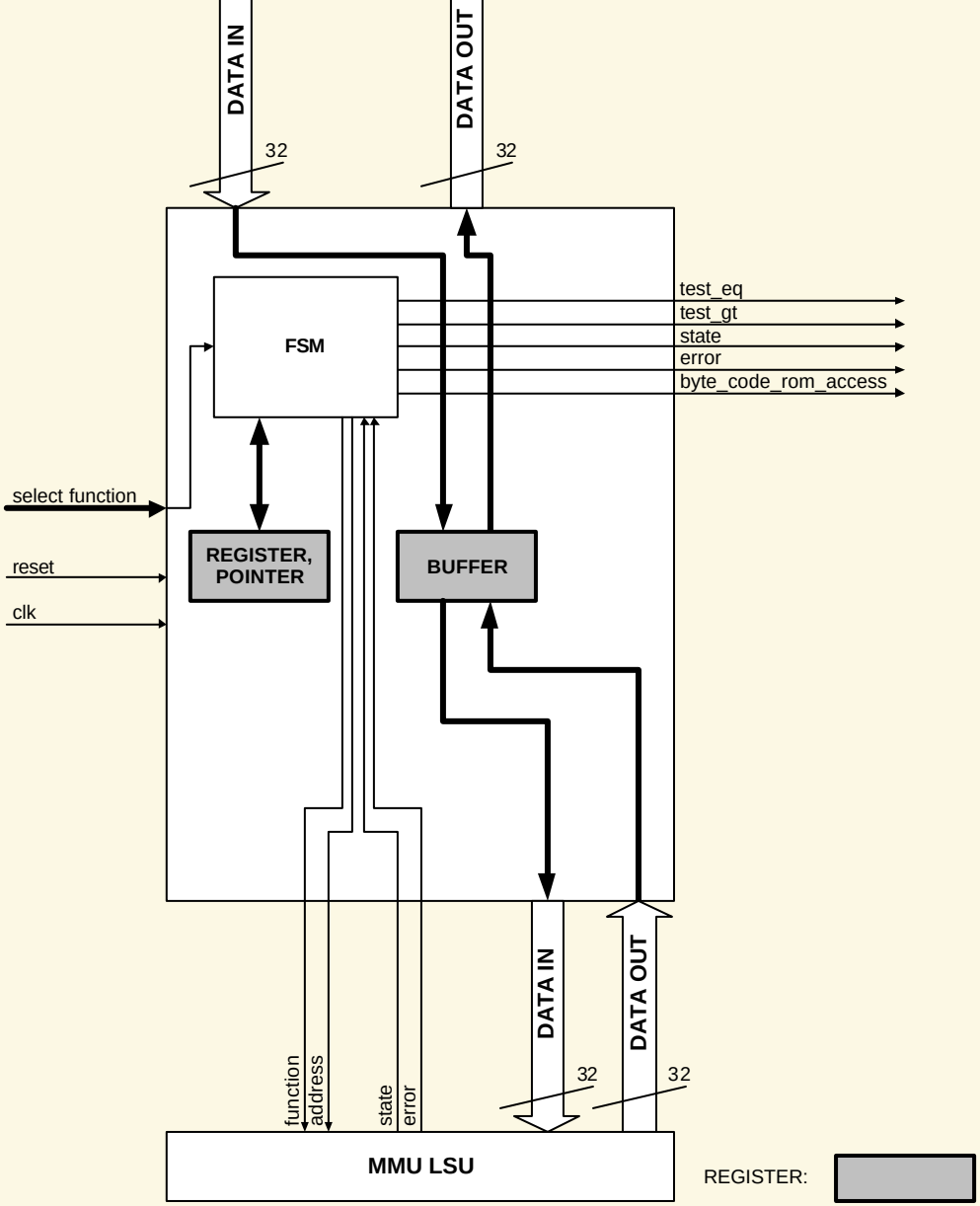
ALU


A
B
DATA IN
DATA IN
32
32
REGISTER A
REGISTER B
state
error
select function
FSM
logic (+,-,*,/, ...)
REGISTER
clk
reset
DATA OUT
1
1
32
zero
lt /gt

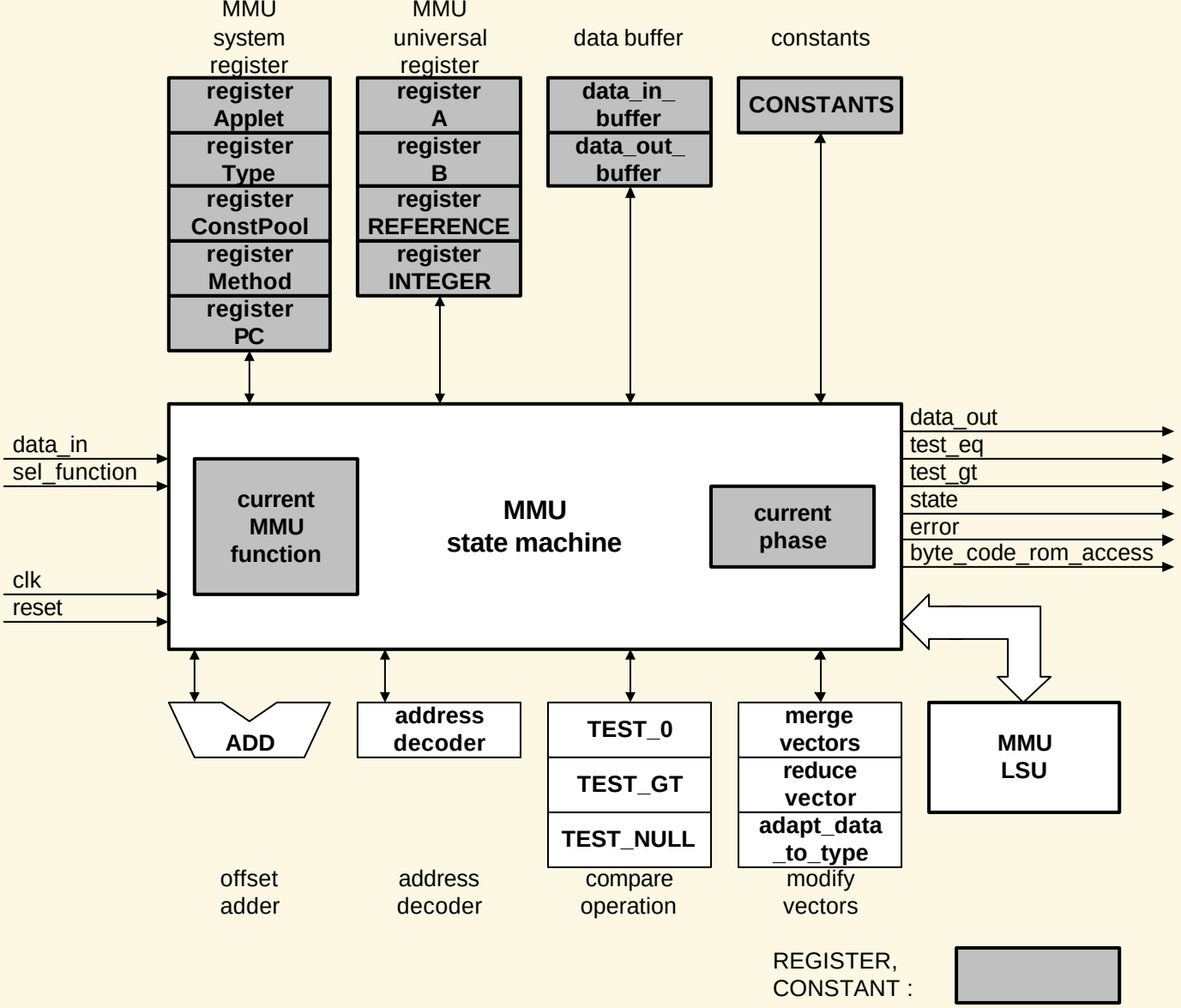
IO & ComponentsALU



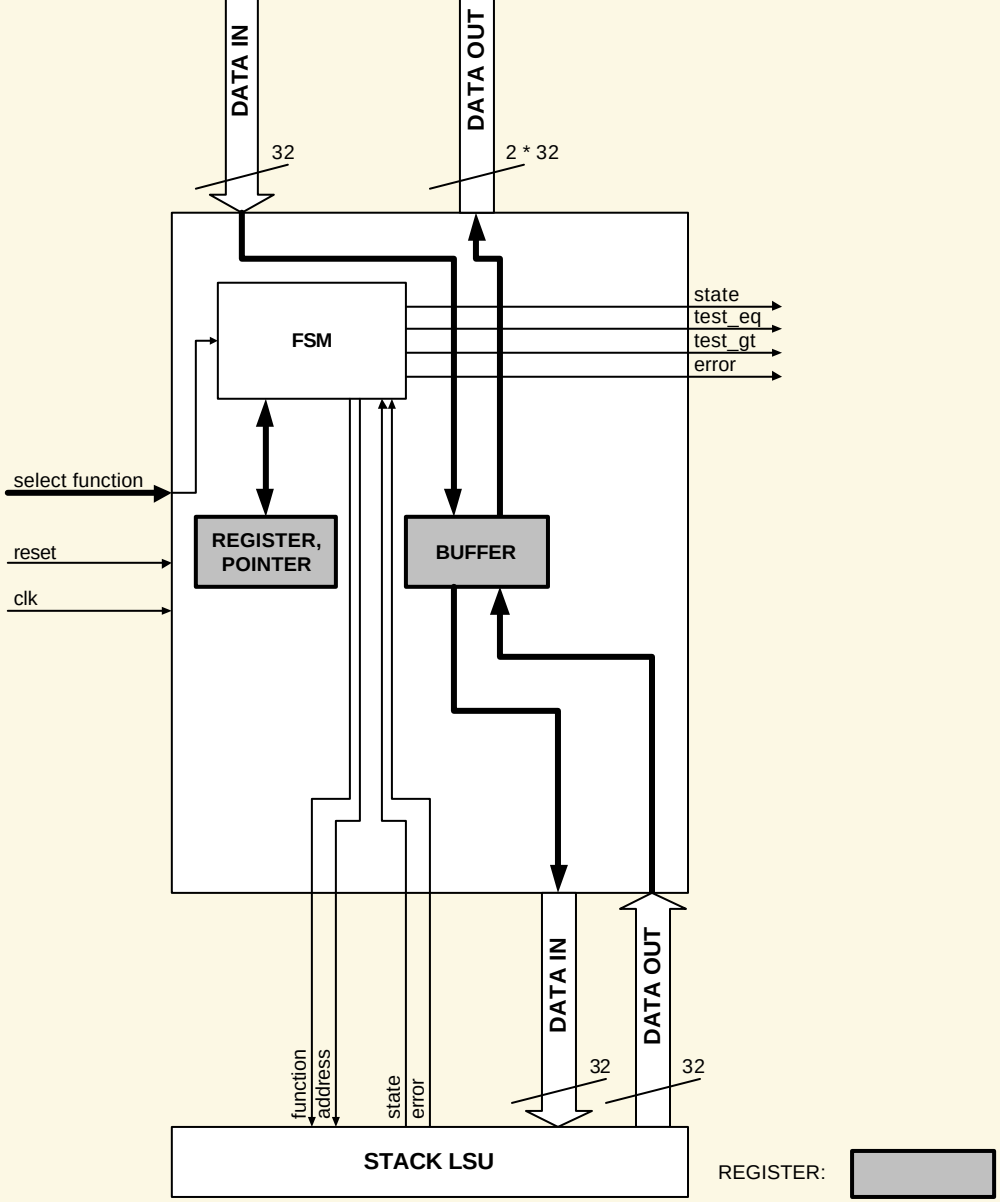
MMU


DATA IN
32
DATA OUT
32
FSM
test_eq
test_gt
state
error
byte_code_rom_access
select function
reset
clk
REGISTER, POINTER
BUFFER
function address
state error
DATA IN
32
DATA OUT
32
MMU LSU
REGISTER:

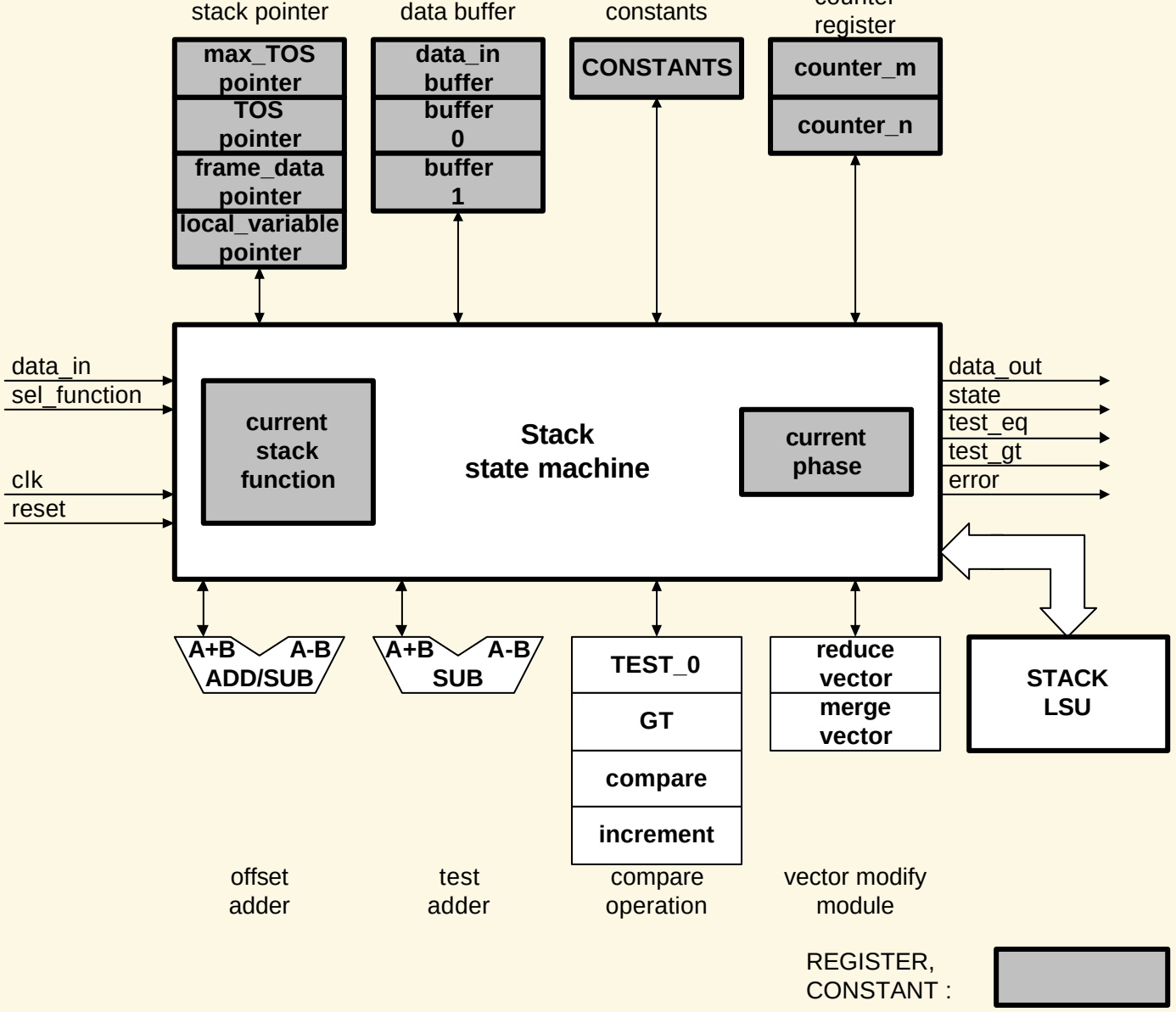
IO & Components MMU



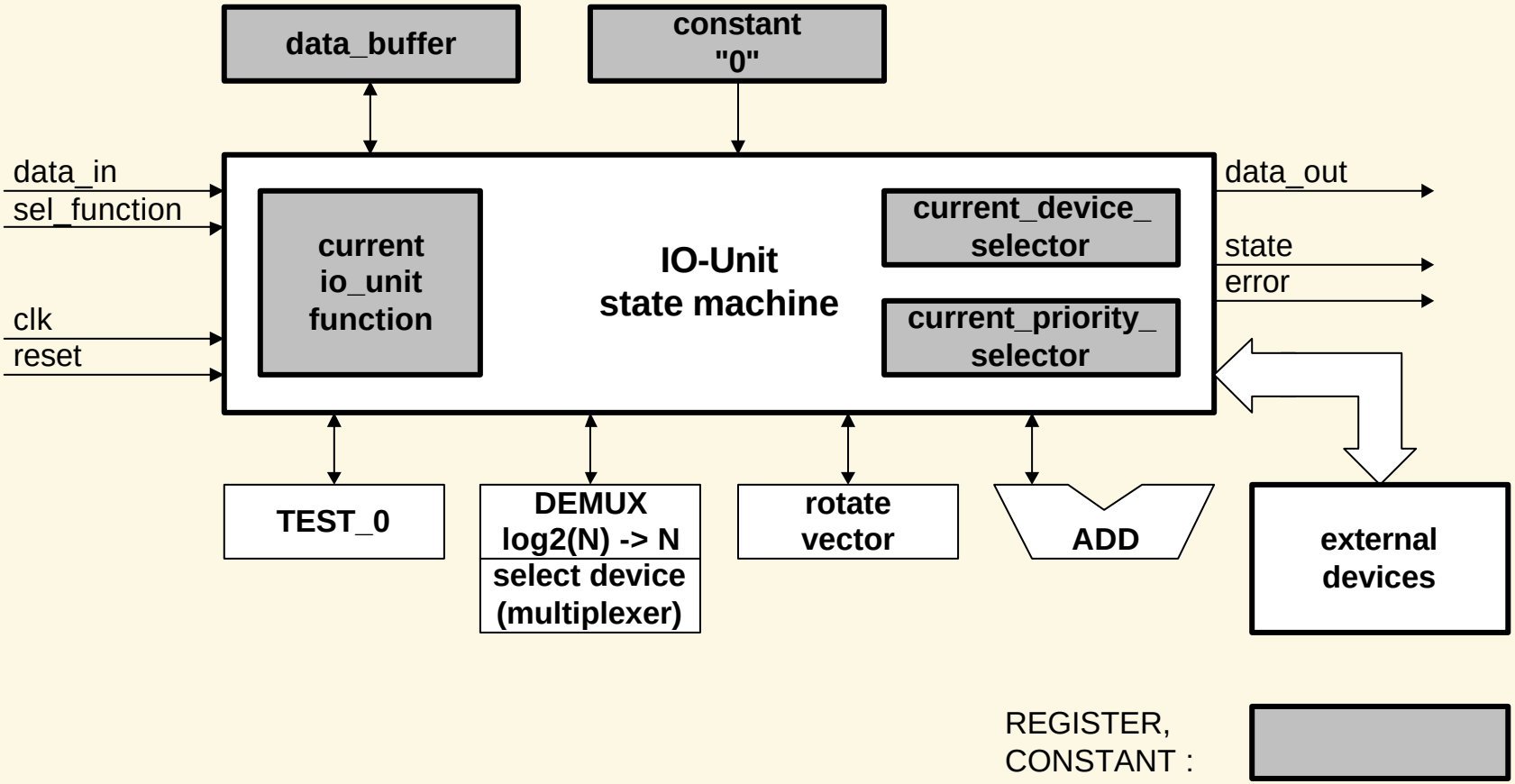
STACK


DATA IN
32
DATA OUT
2 * 32
FSM
state
test_eq
test_gt
error
select function
reset
clk
REGISTER, POINTER
BUFFER
DATA IN
DATA OUT
32
32
function
address
state
error
STACK LSU
REGISTER:

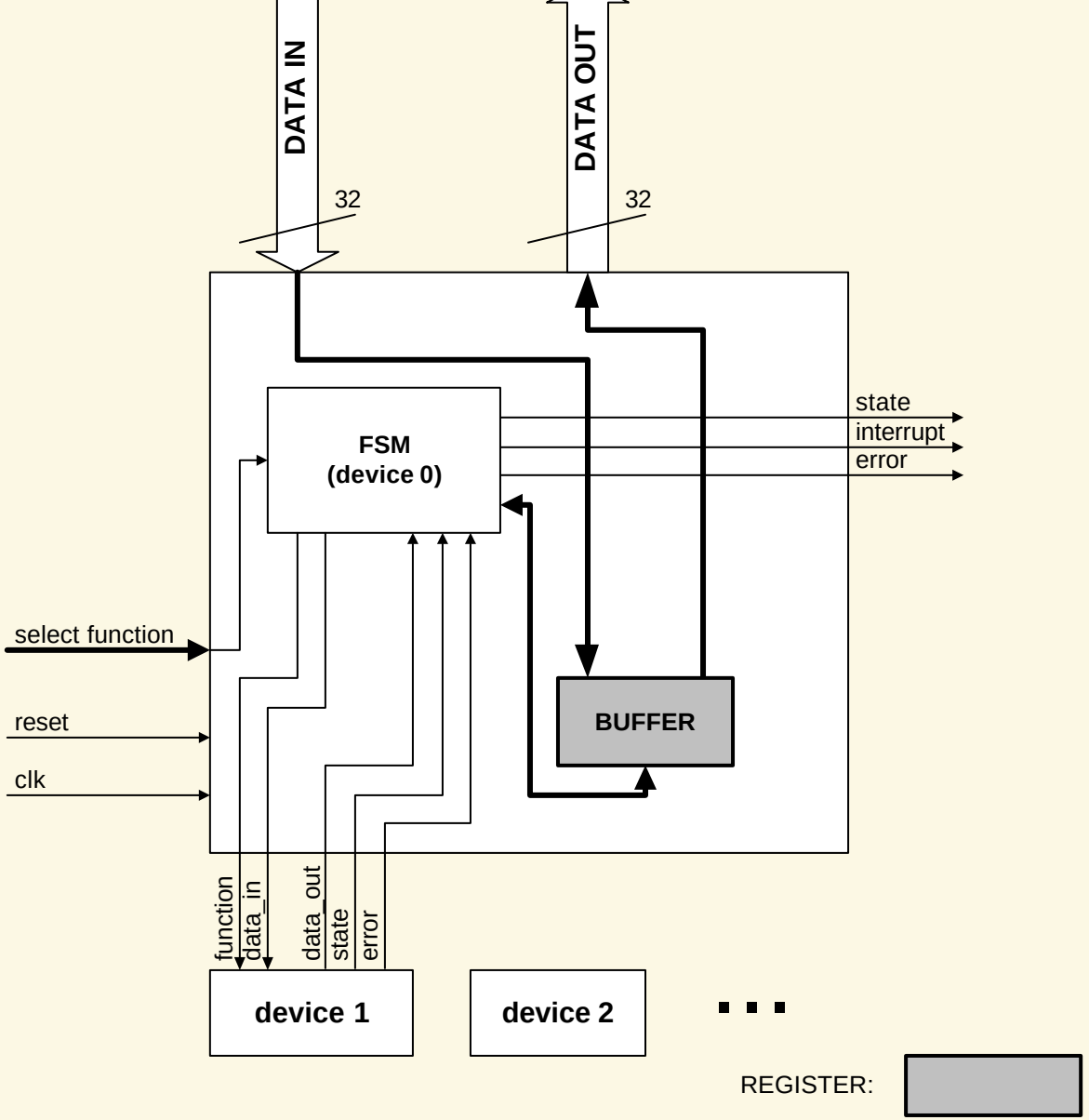
IO & Components Stack



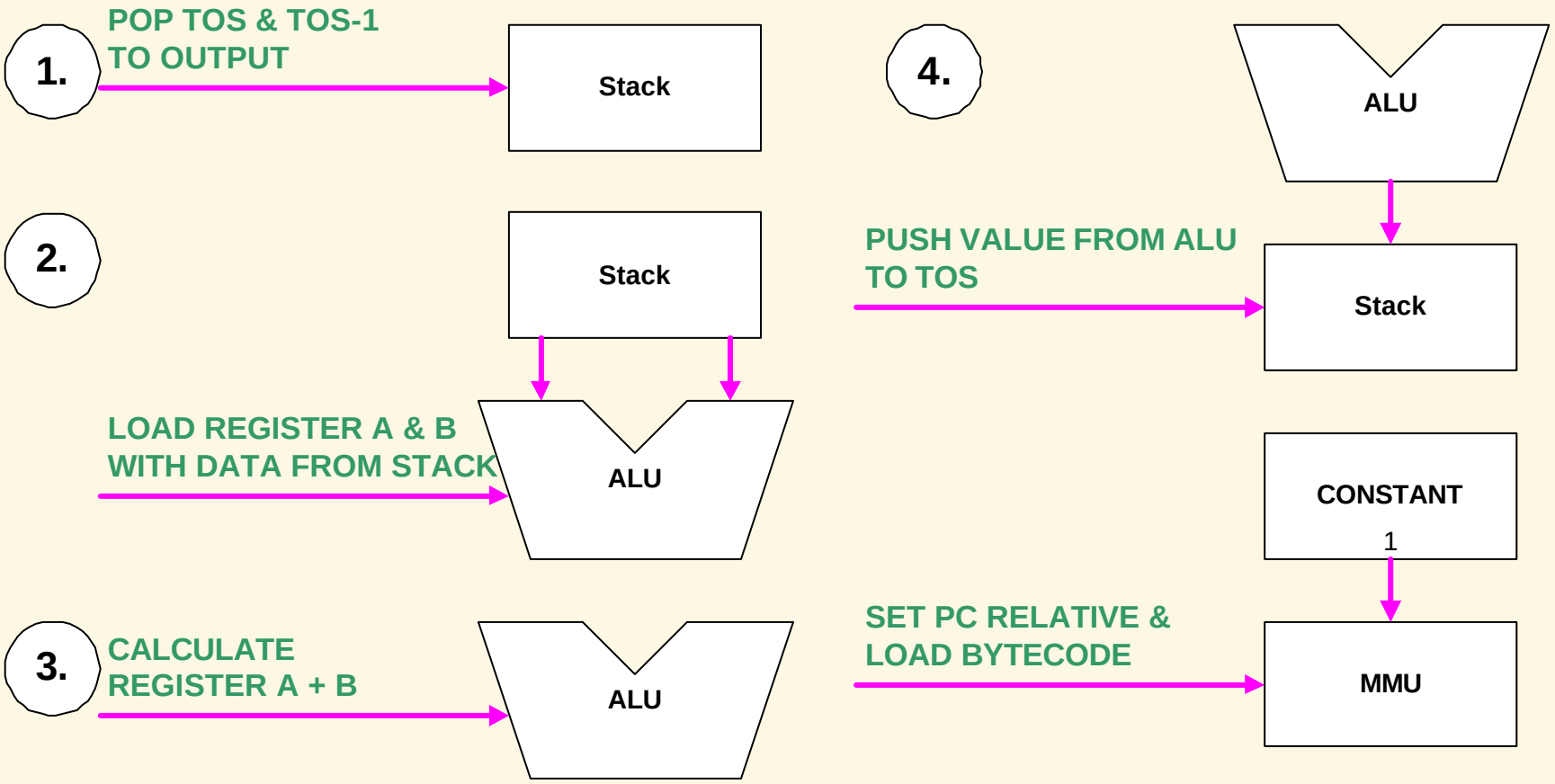
Input / Output and Components of IO-Unit



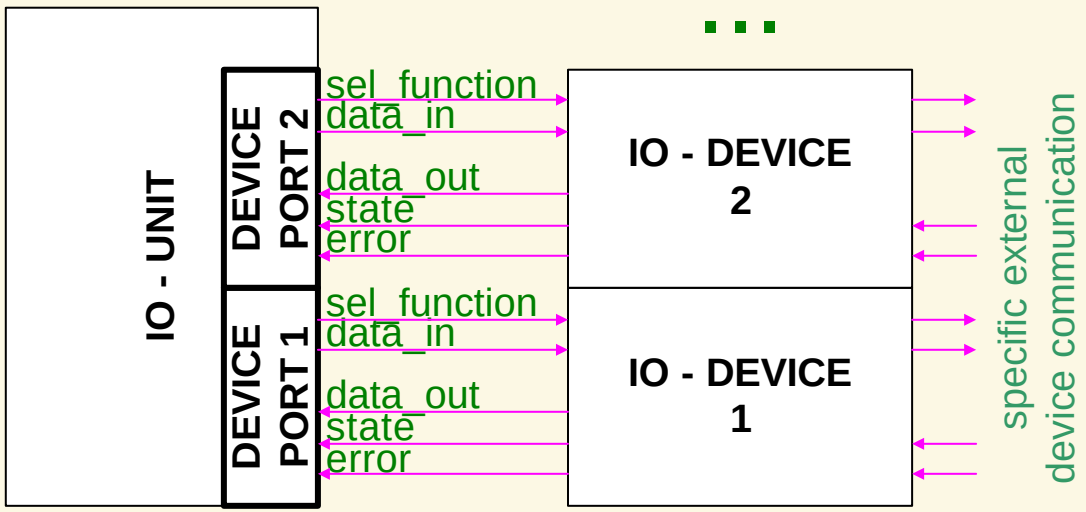
I/O-Unit


DATA IN
32
DATA OUT
32
FSM
(device 0)
state
interrupt
error
select function
reset
clk
BUFFER
function
data_in
data_out
state
error
device 1
device 2
REGISTER:

iadd : mikrocode sequence Example



Access to external devices



- Access to external devices over IO-Unit
- Every Device (IO-Device) has 1 I/O-Bus
- Buffering of data must be done in devices (ISR is used)

New bytecodes

New bytecodes	Operation
start_applet	Preparation of an applet.
read_applet	Reading a word from memory and save it on stack.
write_applet	Writes a word to memory. Value and reference are on stack.
select_io_device	Selects a device from I/O modules.
read_io_device	Reads value(s) from I/O device and pushes it on top of stack.
write_io_device	Outputs a value to I/O device. Value is on stack.

Error recognization and handling

Java has many opportunities to check an applet for run time errors

- Strength type verification
- Unauthorized bytecode instructions
- Range overflow
- Internal errors
- Runtime errors

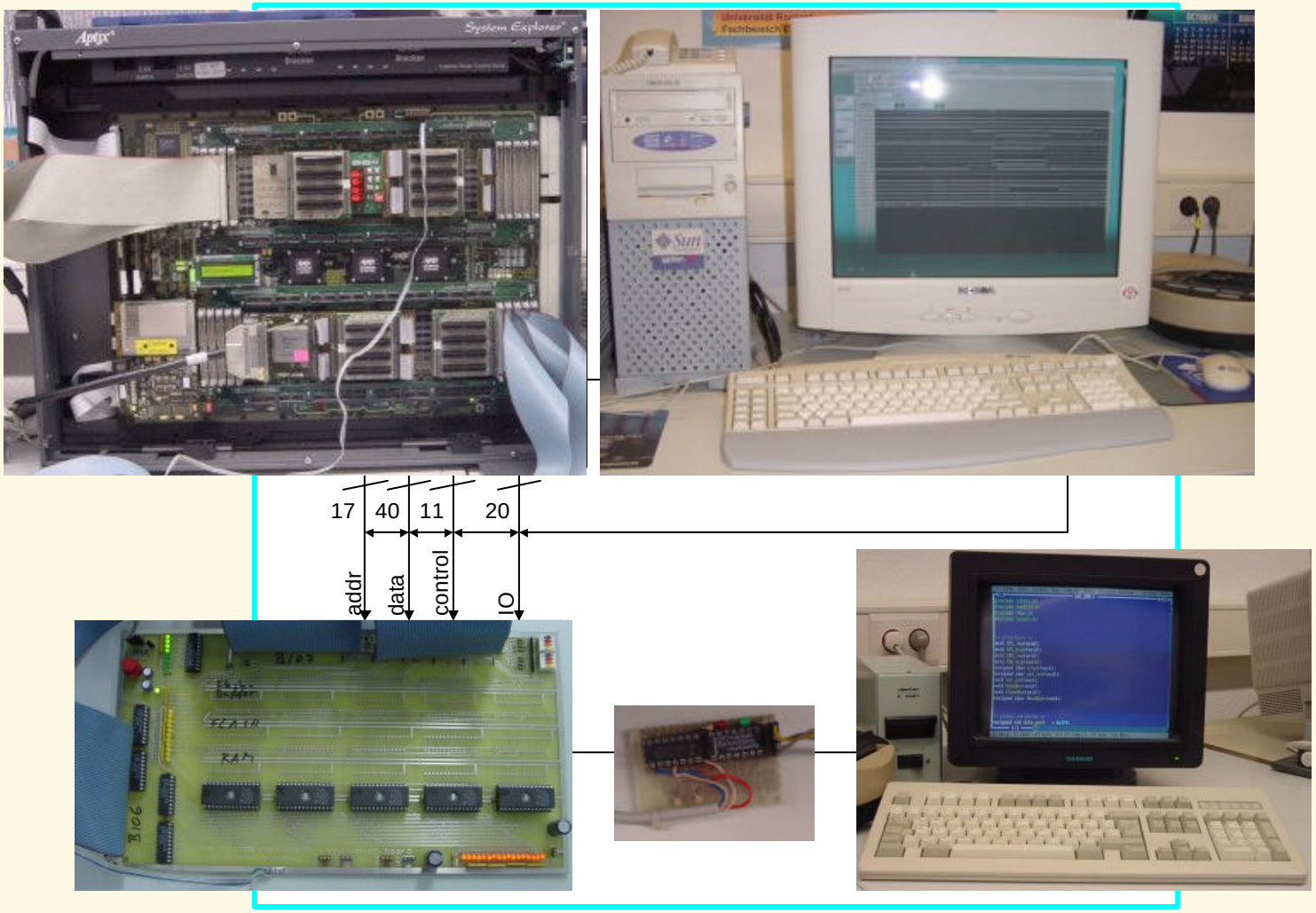
- Now every error results in a stop of the processor

Results

- Area: ~ 2400 slices without optimization
- Utilization of Area within Virtex: ~ 20%
- Speed: ~ 7.5 MHz without optimization
- Size of microcode: 52 kBit

- Estimated die size: ~ 0.2 mm^2 @ 0,18µ

Development environment

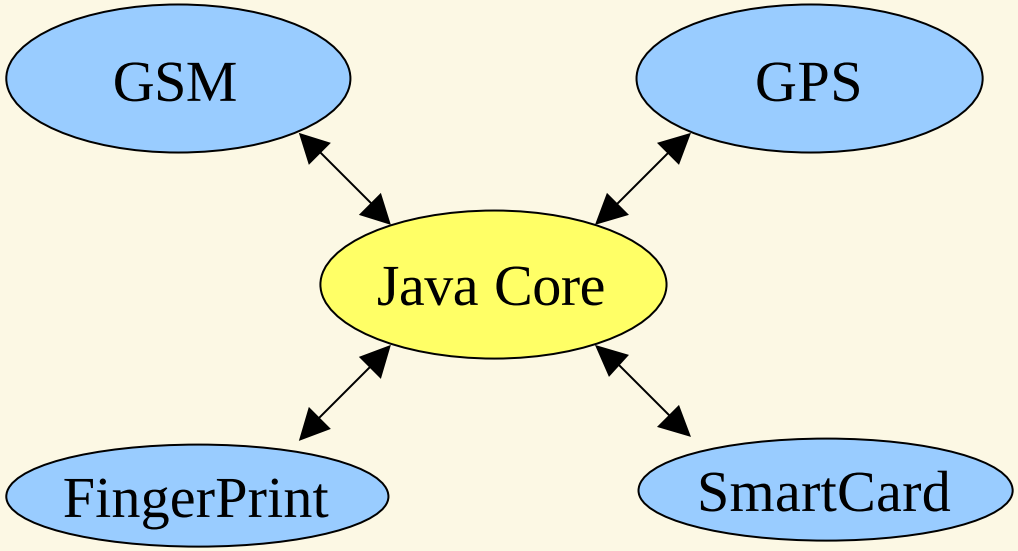


Outview

- JSM in smart devices
- JavaCard API in wireless Systems

JSM in smart devices

- Integration of Java Silicon Machine into Smartdevices
- Mobile Communication Unit for Distributing Intelligence



Future architectures

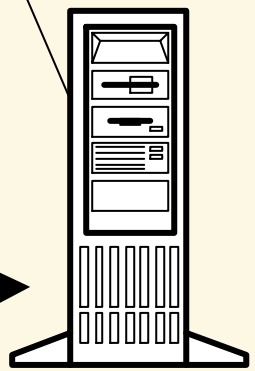
GSM, 3G,
Javacard VM
Applets

Wireless
network
WAP Gateway
Internet

J2EE
HTML / WML
Applet Server
Servlets

J2ME, CLDC
GSM, WAP, 3G
KVM
Java applications

Network



3GPP 3rd Generation Portable Phones
USIM Universal Subscriber Identity Module
CLDC Connected, Limited Device Configurat
CDC Connected Device Configuration

Future architectures

J2ME, CLDC
GSM, WAP, 3G
KVM
Java applications

3GPP 3rd Generation Portable Phones
USIM Universal Subscriber Identity Module
CLDC Connected, Limited Device Configurat
CDC Connected Device Configuration

Future architectures

GSM, 3G,
Javacard VM
Applets

APDU

3GPP 3rd Generation Portable Phones
USIM Universal Subscriber Identity Module
CLDC Connected, Limited Device Configurat
CDC Connected Device Configuration

Future architectures

Wireless Network
WAP Gateway
Internet

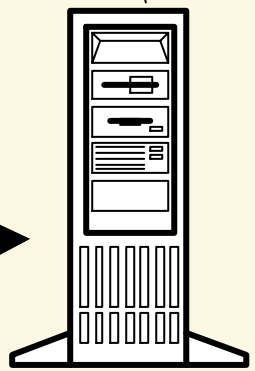
Network

3GPP 3rd Generation Portable Phones
USIM Universal Subscriber Identity Module
CLDC Connected, Limited Device Configurat
CDC Connected Device Configuration

Future architectures

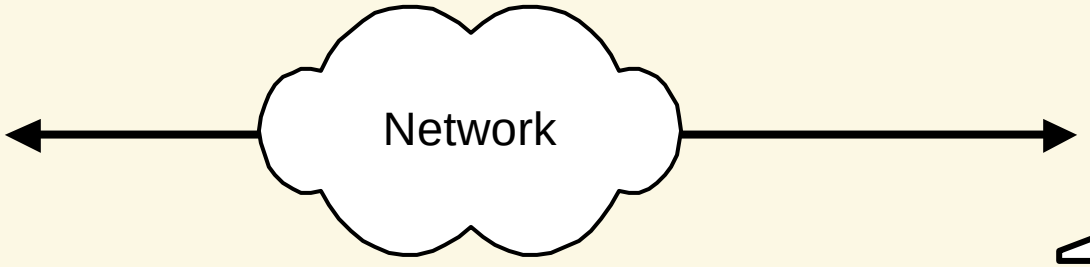
J2EE
HTML / WML
Applet Server
Servlets

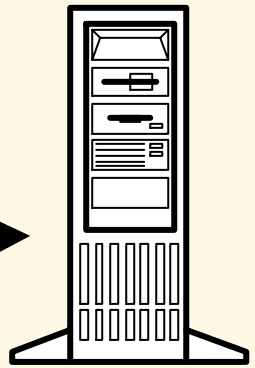
Network



3GPP 3rd Generation Portable Phones
USIM Universal Subscriber Identity Module
CLDC Connected, Limited Device Configurat
CDC Connected Device Configuration

Future architectures





3GPP 3rd Generation Portable Phones
USIM Universal Subscriber Identity Module
CLDC Connected, Limited Device Configurat
CDC Connected Device Configuration

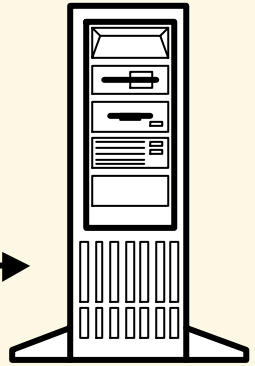
Future architectures

RMI

RMI

Applications are independent from protocols

RMI



...thank you