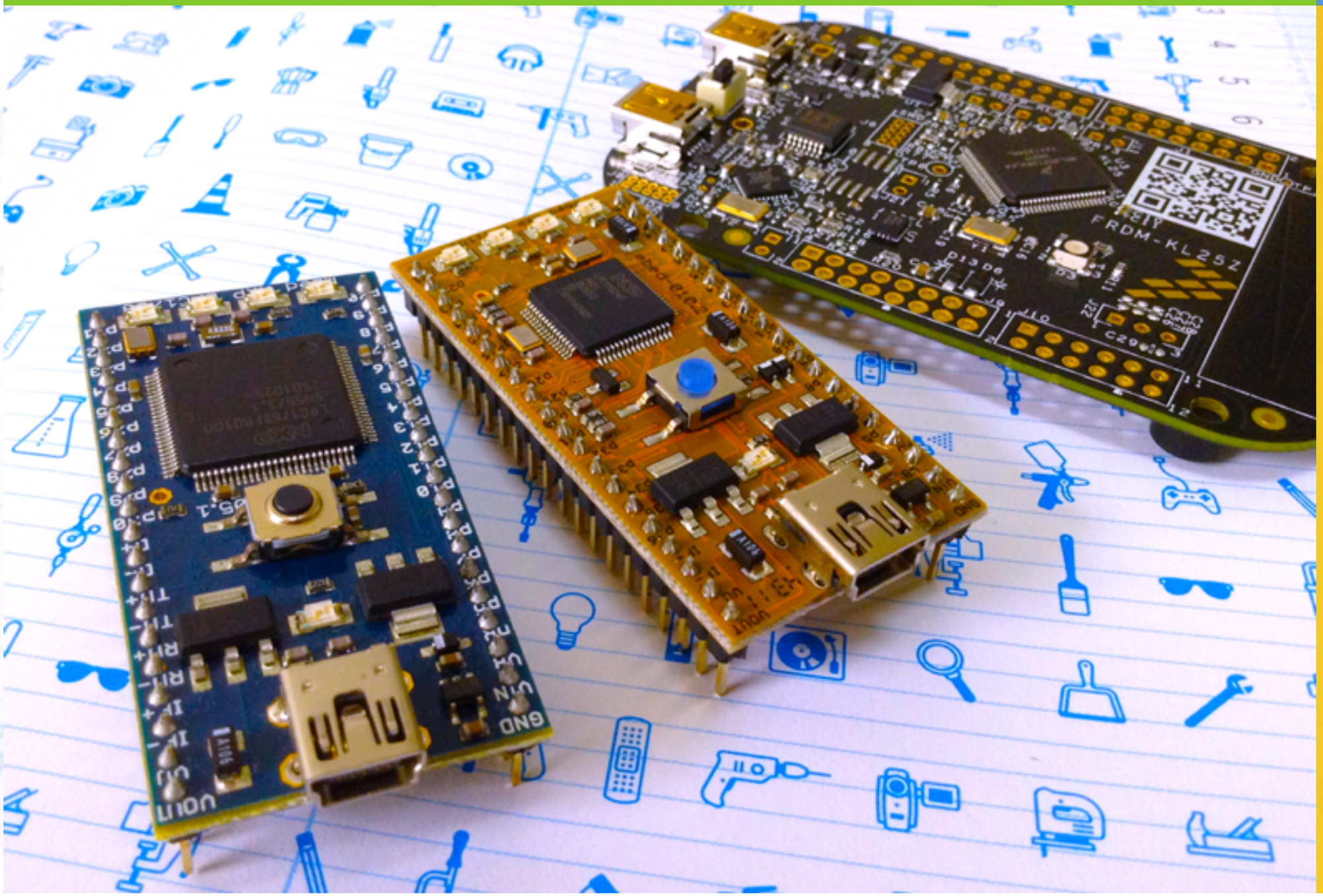
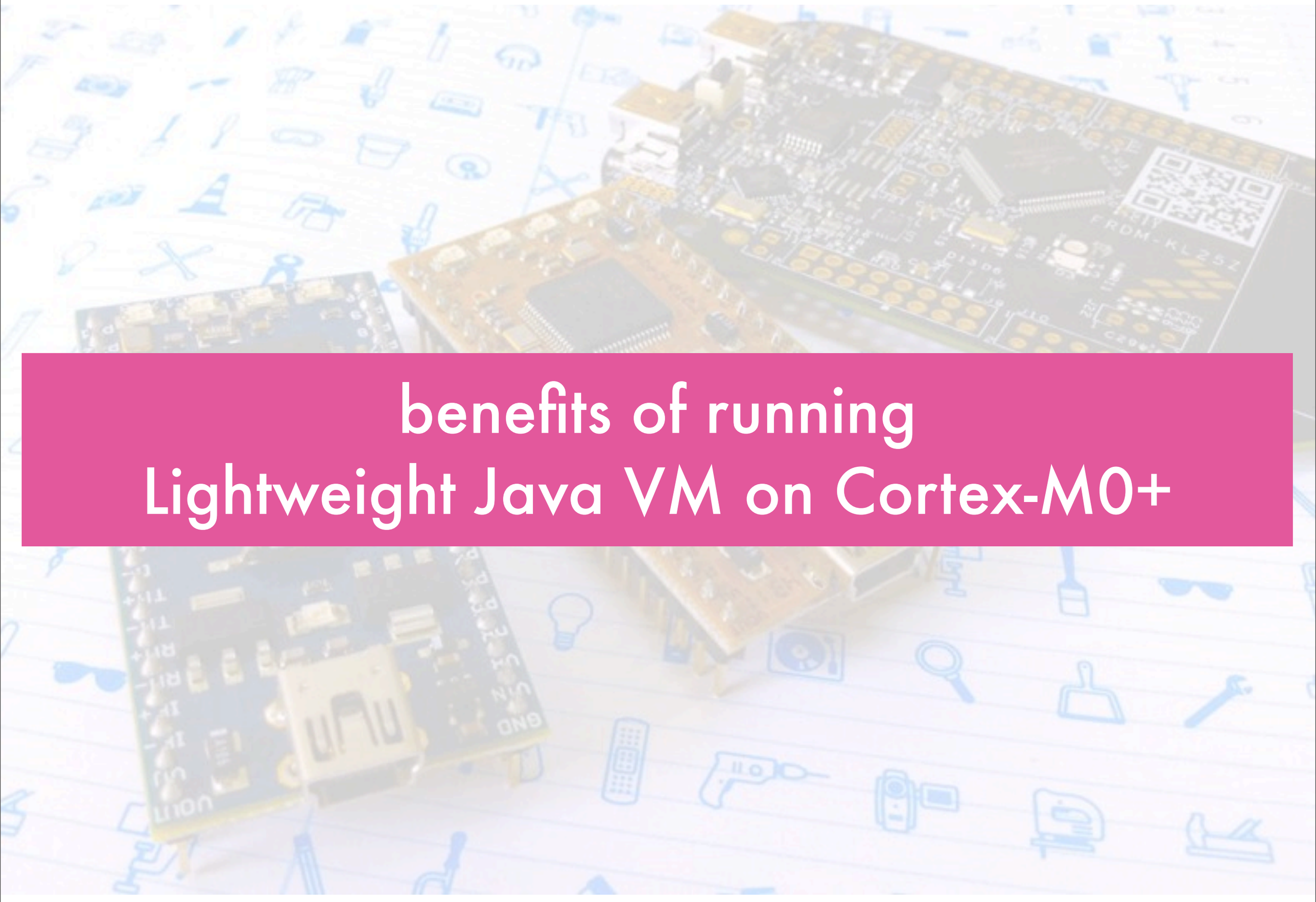
mbed fest 2013

- Nagoya, Yokohama, Sapporo JPN





benefits of running Lightweight Java VM on Cortex-M0+

Agenda

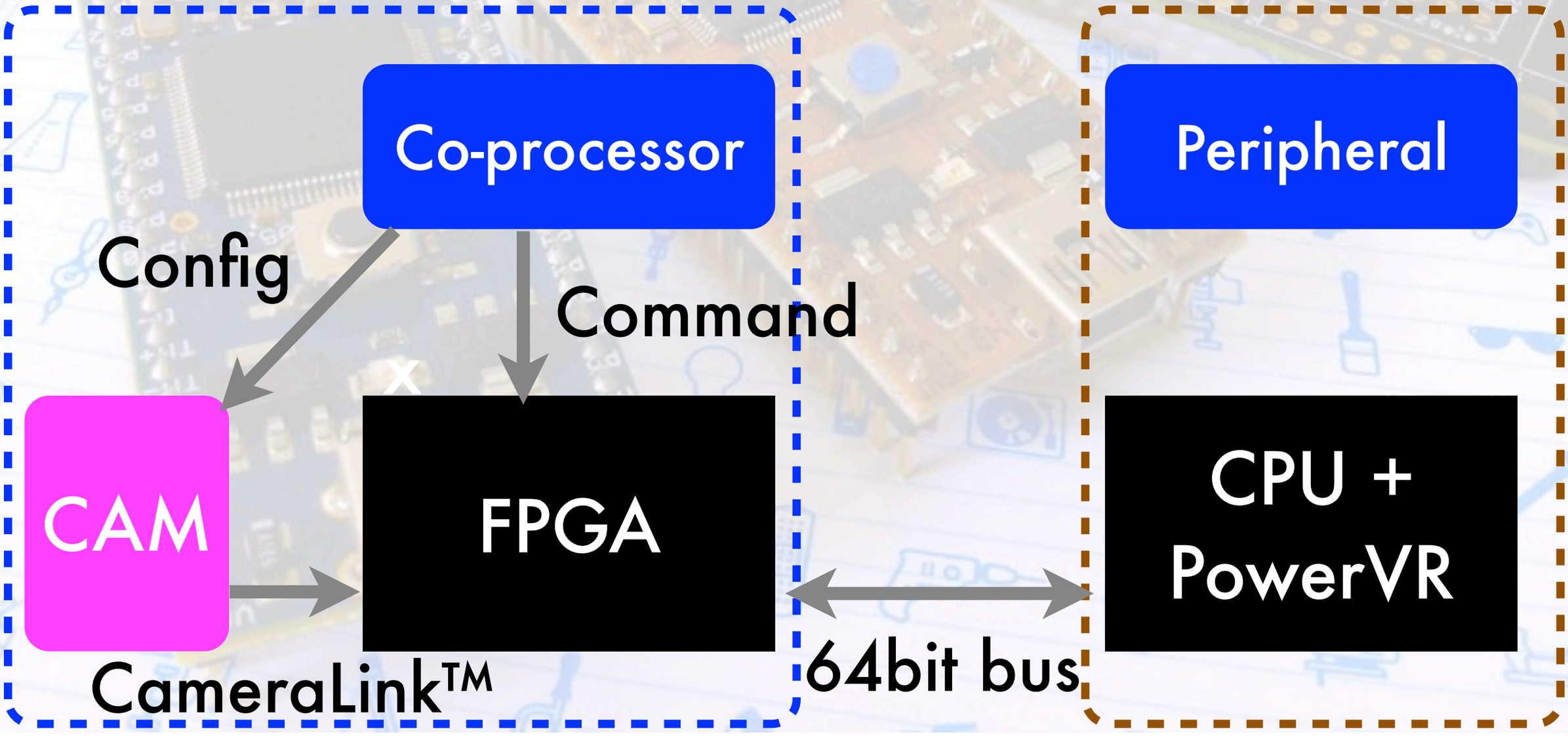
- Device-independent Development Environment
- the “DIY-interpreter” make us unhappy
- Lists of Major VM
- Why do we need “device-independent”?
- RaVem JVM

Agenda

- Device-independent Development Environment
- the "DIY-interpreter" make us unhappy
- lists of Major VM
- Why do we need "device-independent"?
- RaVem JVM

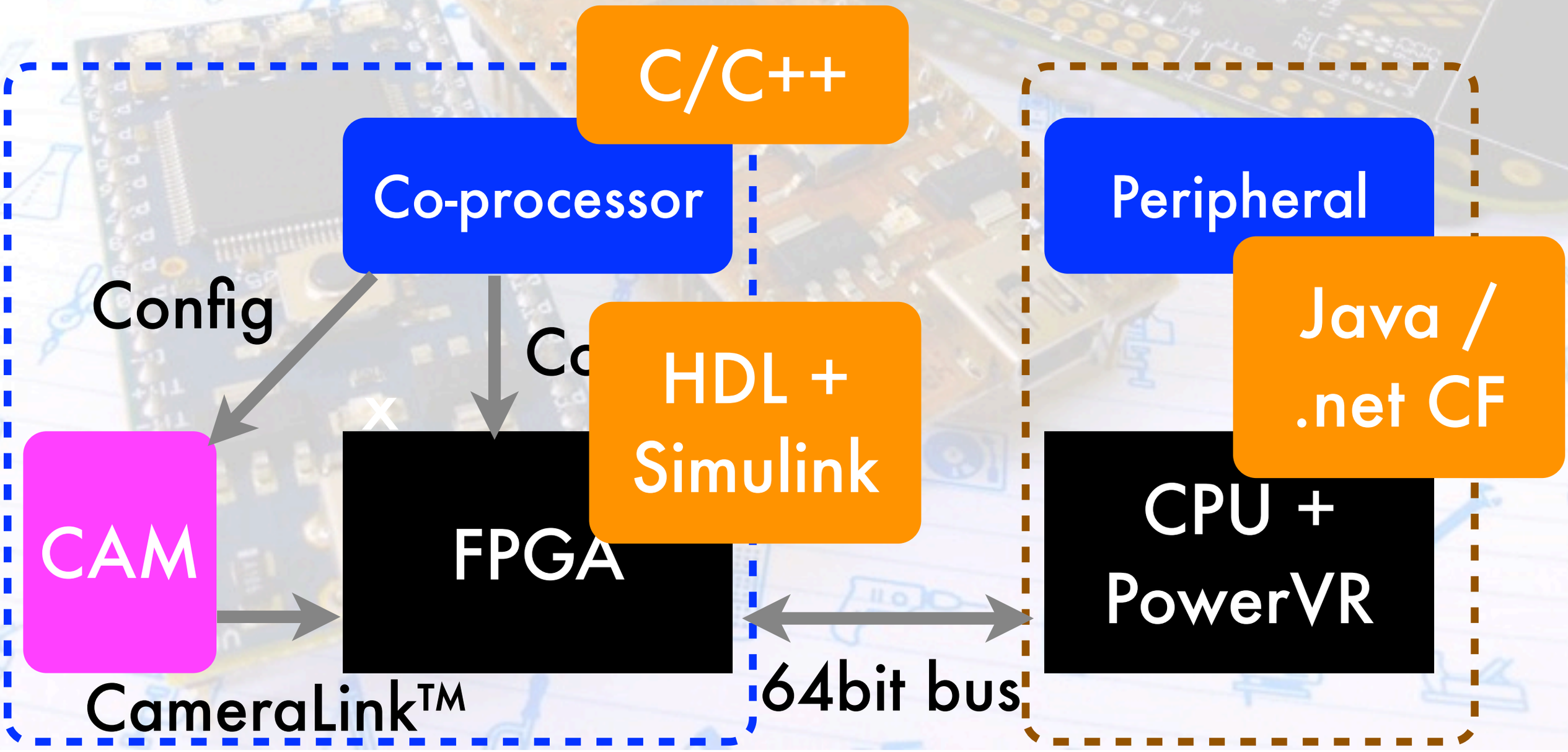
Processor-independent devel. environment

- ex:developers in different layer, different Language



Processor-independent devel. environment

- ex:developers in different layer, different Language



Agenda

- Device-independent Development Environment
- the “DIY-interpreter” make us unhappy
- lists of Major VM
- Why do we need “device-independent”?
- RaVem JVM

the “DIY-interpreter” make us unhappy

- “Uh Oh We're In Trouble!”
  - Interactive shell or managed code generation
  - error occurs!
    - coding mistake?
    - or interpreter’s error?
    - arghhh! I must remake my interpreter!!
- and, You must tell your friends how to use this VM.
- OMG!

Agenda

- Device-independent Development Environment
- the "DIY-interpreter" makes us "mad"
- Lists of Major VM
- Why we need "device-independent"?
- Ro

So...

- use one's VM standards

Language	for embedded	requirements
Python	P14P(ex.PyMite)	Flash:55kB~,RAM:8kB~
Lua	eLua	Flash:256kB~,RAM:64kB~
Ruby	mruby	Flash:512kB,RAM:1MB?
Java	Java ME and so on...	Flash:130kB,RAM:8kB

VM

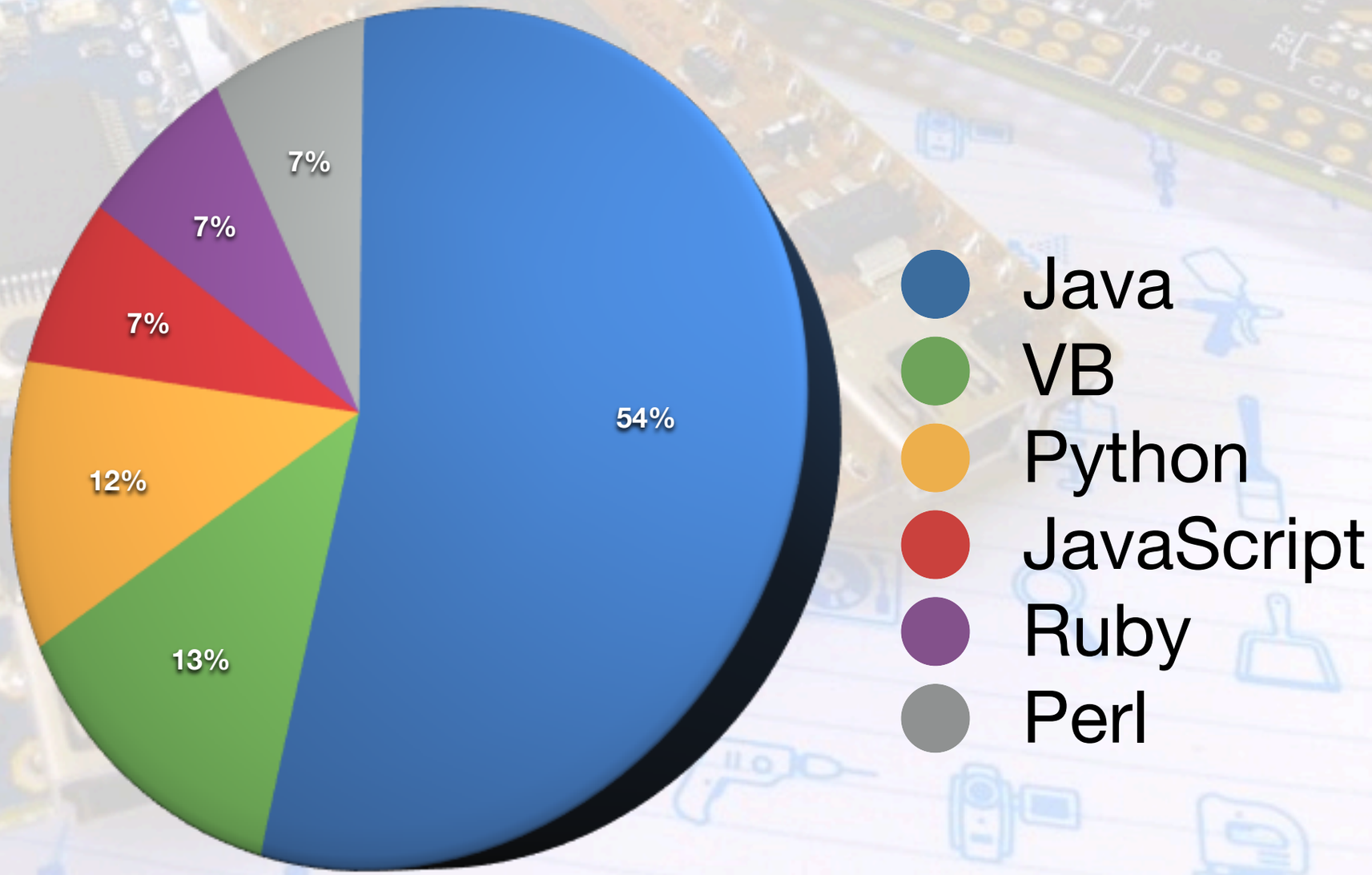
- use one's VM standards

- Java
- VB
- Python
- JavaScript
- Ruby
- Perl

VM

- use one's VM standards

Popular VM languages(Tiobe Programming Community,Aug 2013)



Variants Java VM

- Java for embedded

JVM	System requirements	
Oracle Java ME Embedded for STM32F	Flash :130kB~ RAM: ?	Binary only
uJ	Flash :64kB~ RAM:192bytes~	thread,runnable,GC, original String type
nanoVM	Flash :8kB~ RAM: 1kB~	for AVR Single thread
RaVem	Flash :5kB~ RAM: 256bytes~	thread,runnable, Integer only

Agenda

- Device-independent Development Environment
- the "DIY-interpreter" make up unit
- Lists of Major VM
- Why do we need "device-independent"?
- RaVem JVM

C compiler

- Just C, but each processor needs their unique compiler.

ARM

intel

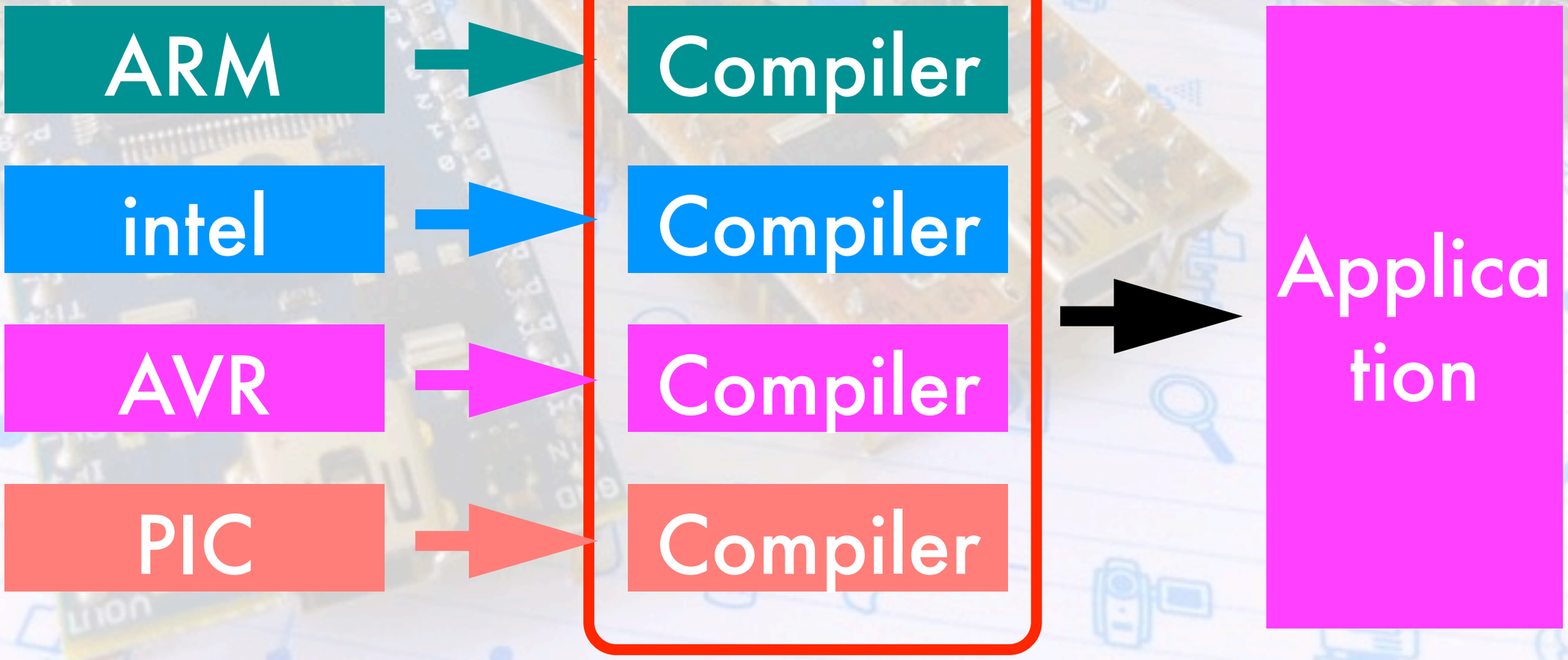
AVR

PIC

Applica
tion

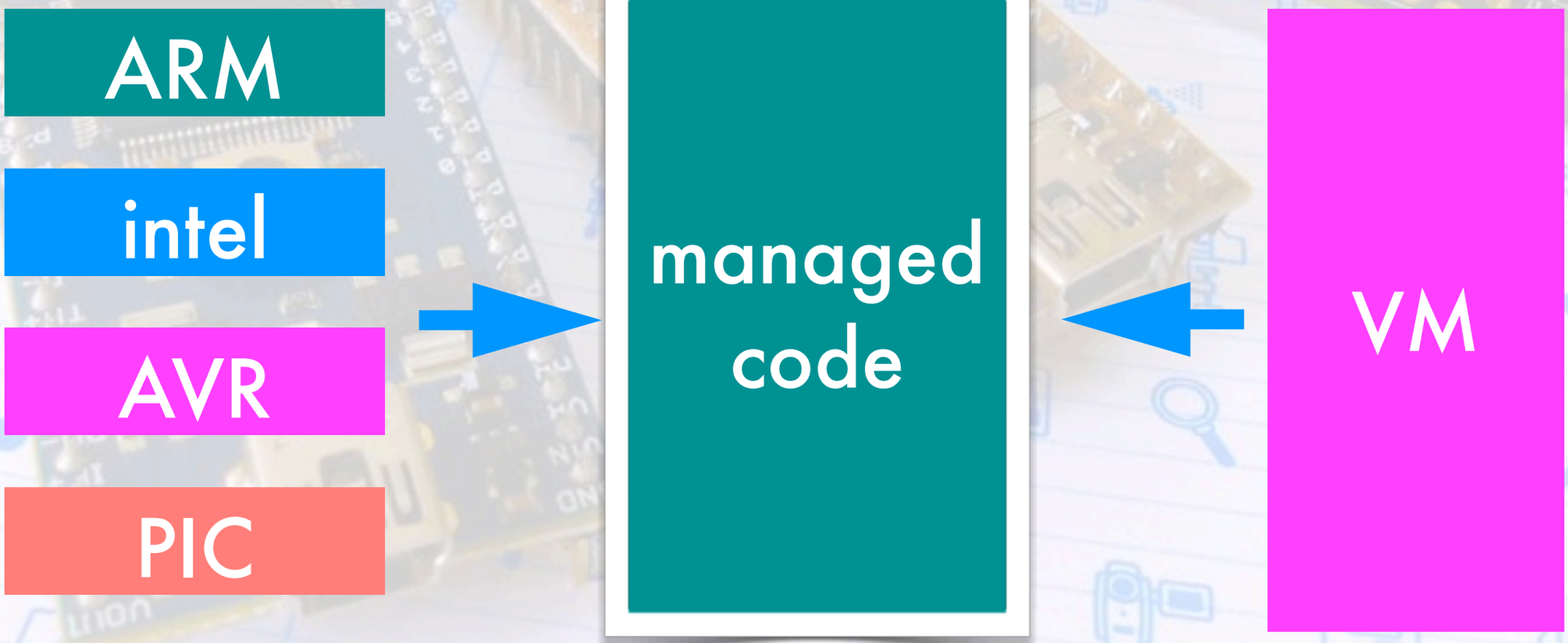
C compiler

- Just C, but each processor needs their unique compiler.



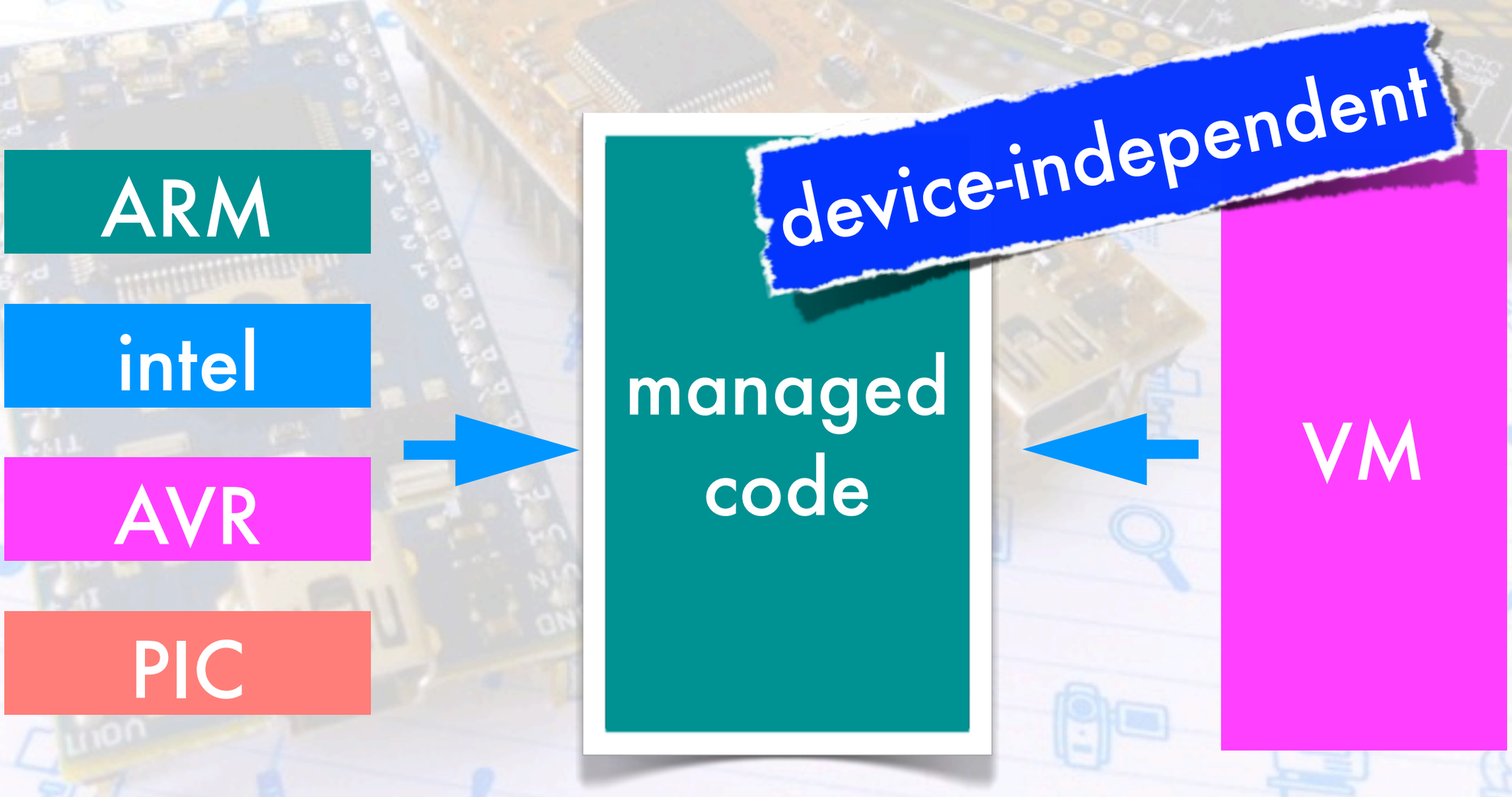
Java

- Java:Write VM on processor once, same managed code,and run anywhere.



Java

- Java:Write VM on processor once, same managed code,and run anywhere.



Java

- Java:Write VM on processor once, same managed code,and run anywhere.

device-independent

managed

VM

write once run anywhere

PIC

VM

- How to implementing system that interpret byte code?
- How can we read Java byte code?

reading byte code

CA FE BA BE.....

.....

.....

.....

.....

.....

.....

.....

.....

- Java byte consists...
  - Constant Pool
  - Code Attribute(instructions)
  - Exception

- all literal is written in big-endian.

reading byte code

CA FE BA BE.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

- Java byte consists...
  - Constant Pool
  - Code Attribute(instructions)
  - Exception

- all literal is written in big-endian.

reading byte code

CA FE BA BE.....

.....

.....

.....

.....

.....

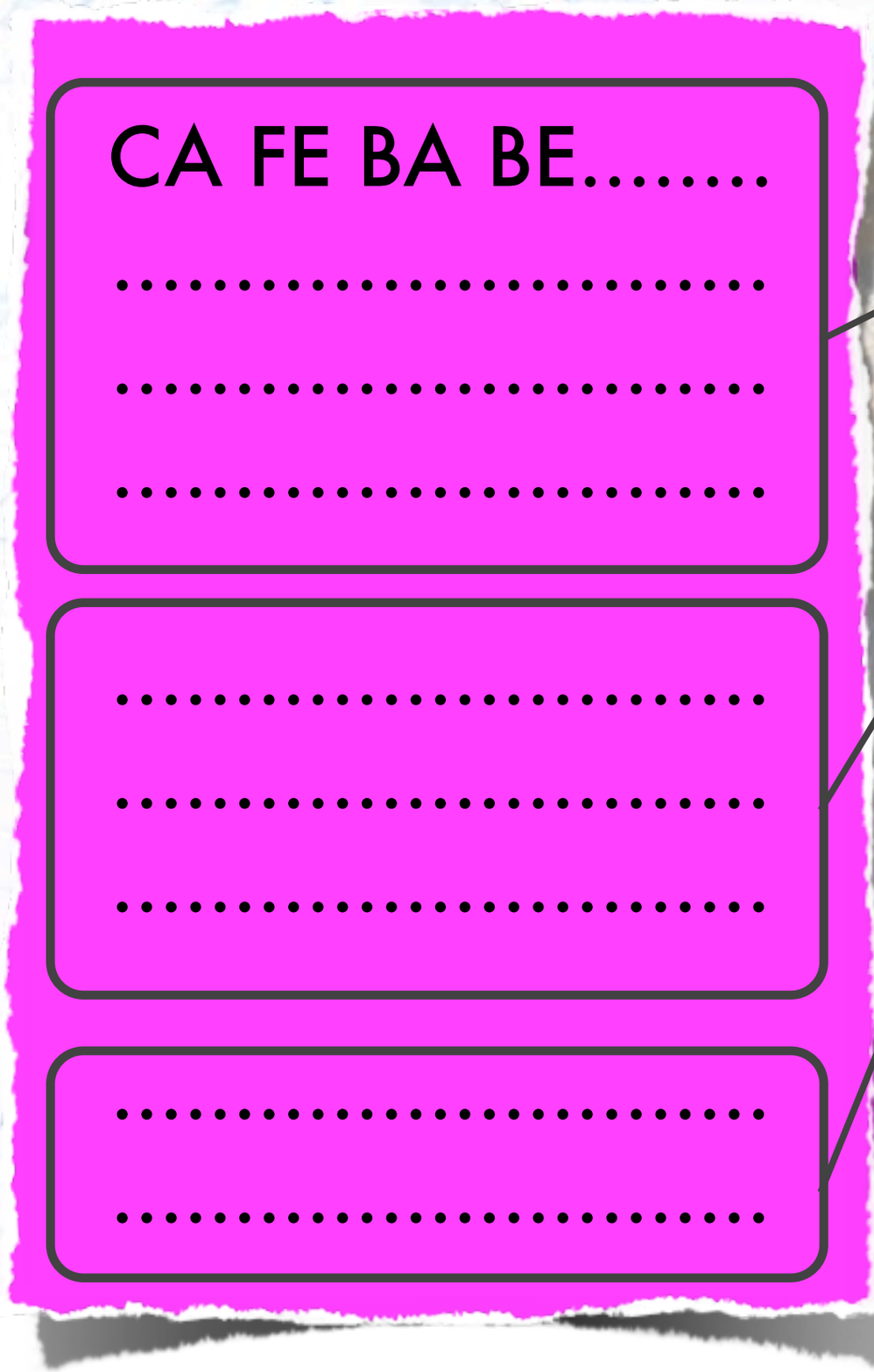
.....

.....

- Java byte consists...
  - Constant Pool
  - Code Attribute(instructions)
  - Exception

- all literal is written in big-endian.

reading byte code



- Java byte consists...
  - Constant Pool
  - Code Attribute(instructions)
  - Exception
- all literal is written in big-endian.

byte code

cp[1]=.....

.....

cp[

.....

load cp[20],stack
call “println”

```
System.out.println(“hello world”);
```

- exam:hello world
- Constant pool #20→”hello world”(for example)
- stack CP#20 to Operand Stack
- call println methods

byte code

cp[1]=.....

.....

cp[20]="hello world"

.....

load cp[20],stack

call "println"

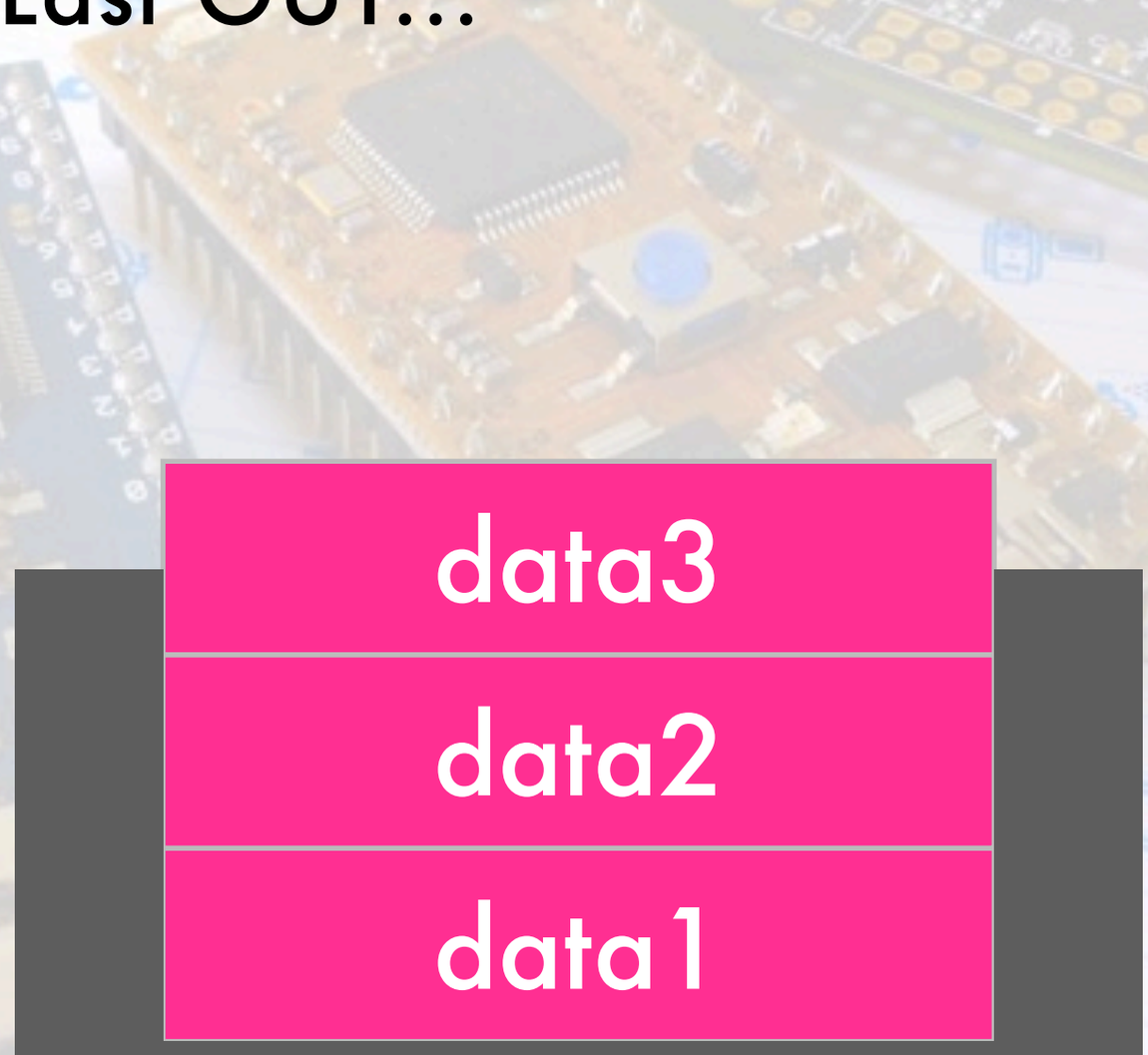
- exam:hello world
- Constant pool #20→"hello world"(for example)
- mnemonics
  - stack CP#20 to Operand Stack
  - call println methods

stack

- Operand Stack
  - First IN, Last OUT...

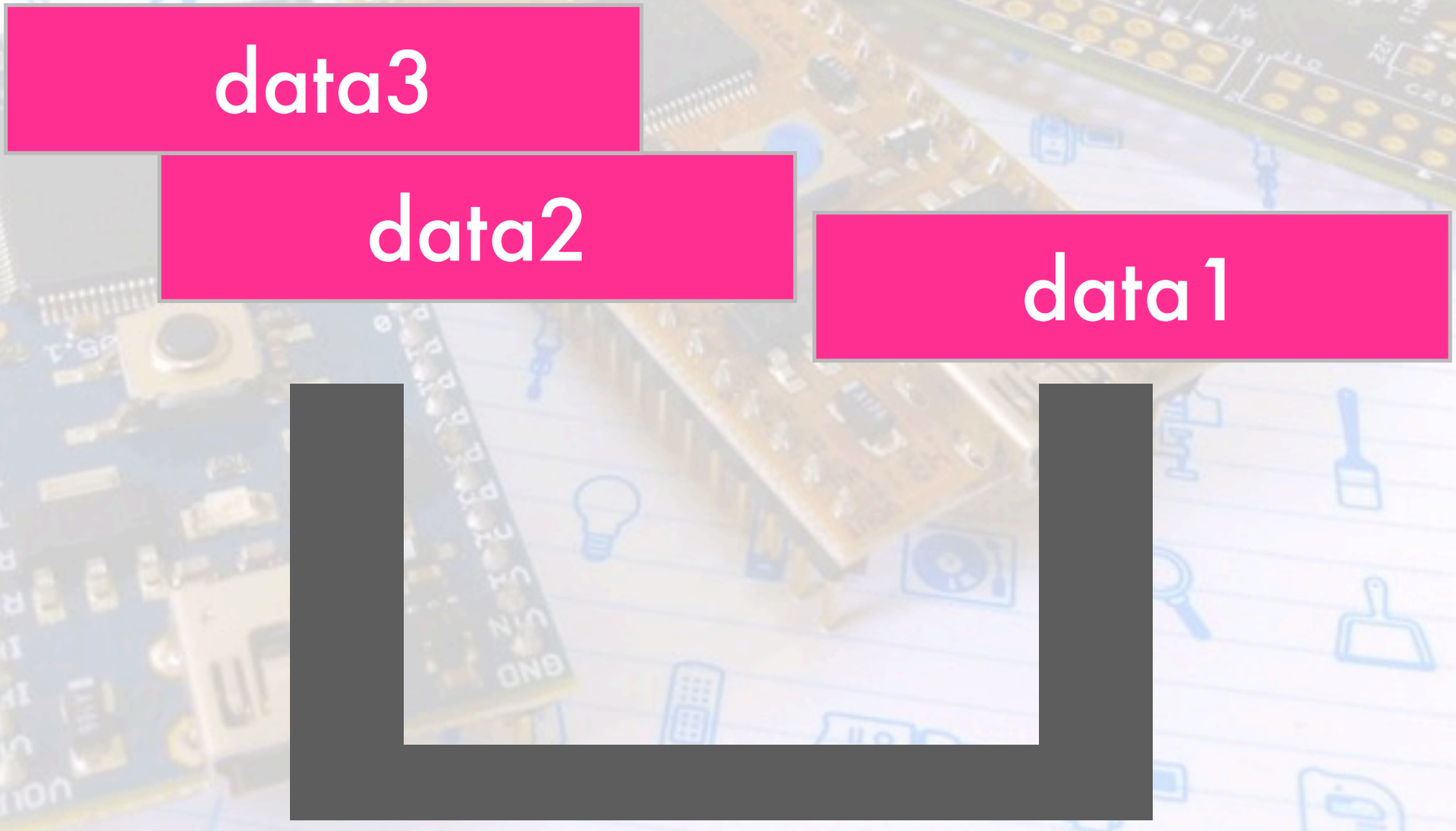
stack

- Operand Stack
  - First IN, Last OUT...



stack

- Operand Stack
  - First IN, Last OUT...



example: add

- Exam. of Using Operand Stack(Addition)

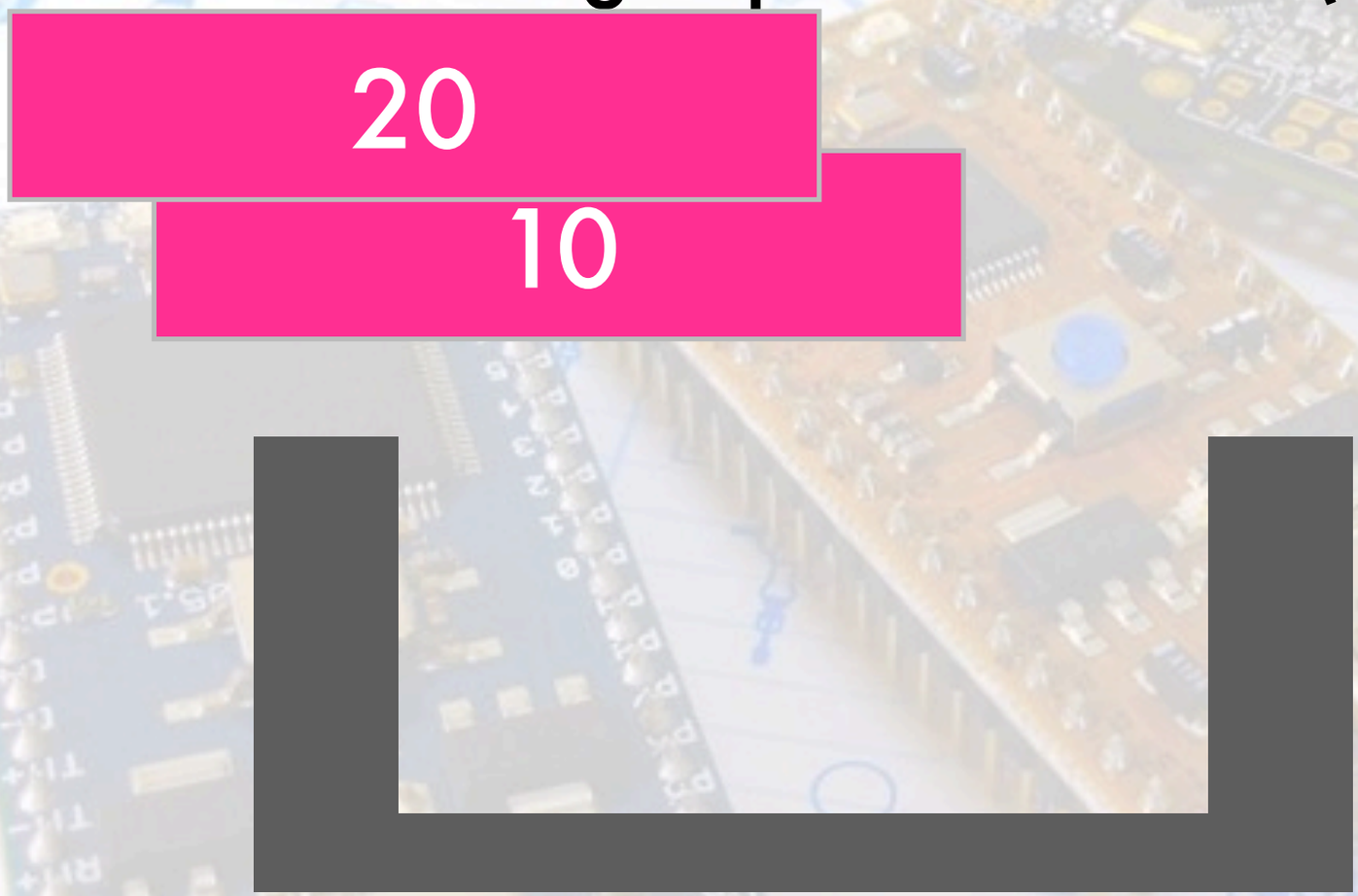
```
int x, y, z;

x = 10;
y = 20;
z = x + y;
System.out.println(z);
```

example: add

- Exam. of Using Operand Stack(Addition)

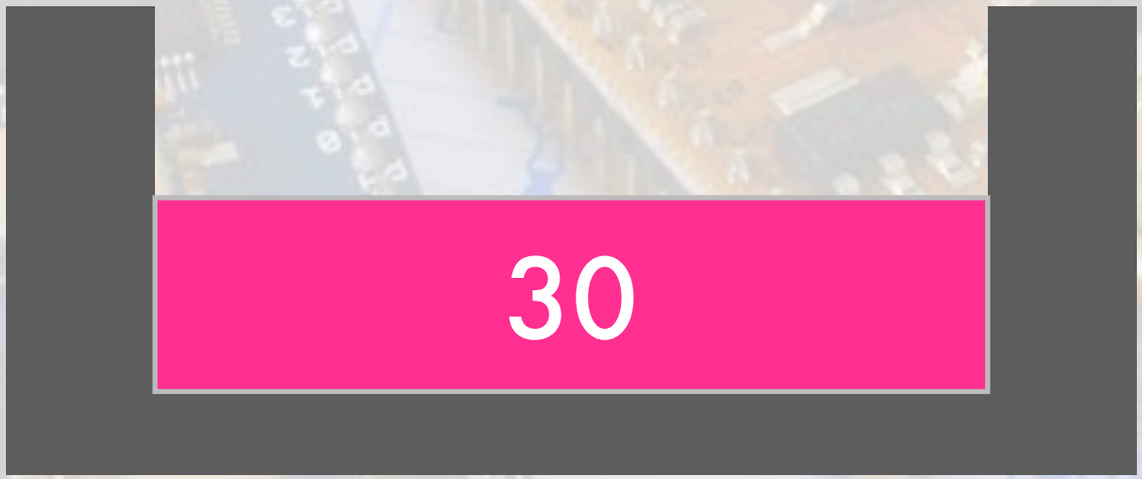
example: add

- Exam. of Using Operand Stack(Addition)



example: add

- Exam. of Using Operand Stack(Addition)



Java VM

- Java VM needs only
  - Mechanism to quote the Constant Pool,
  - Mechanism to manage the Operand Stack,
  - and Mechanism for executing instructions.

Java VM

- Java VM needs only
  - Mechanism to quote the Constant Pool,
  - Mechanism to manage the Operand Stack,
  - and Mechanism for executing instructions.

we've just implemented JavaVM on mbed!

Agenda

- Device-independent Development Environment
- the "DIY-interpreter" maker's journal
- Lists of Major VM
- Why do we need "device-independent"?
- RaVem JVM

RaVem JVM

- Requirements
  - Flash 5kbytes or more
  - RAM 128bytes or more
  - int type only

- all code written in C (about 700 sloc)

- github (for LPC812)
  - https://github.com/lynxeyed-atsu/RaVem

- port on mbed
  - http://mbed.org/users/lynxeyed_atsu/code/FRDM_RaVem_JVM

blink LEDs with JVM

- Method of control GPIOs: portWrite(bit, value)
- Write the code Turning a LED on / off
- make that code to work in 4 threads (mbed has 4LEDs)

blink LEDs with JVM

- Method of control GPIOs: portWrite(bit, value)
- Write the code Turning a LED on / off
- m as 4LEDs)

```
mbed.portWrite(port_bit, 0);
Thread.sleep(time);
mbed.portWrite(port_bit, 1);
Thread.sleep(time);
```

blink LEDs with JVM

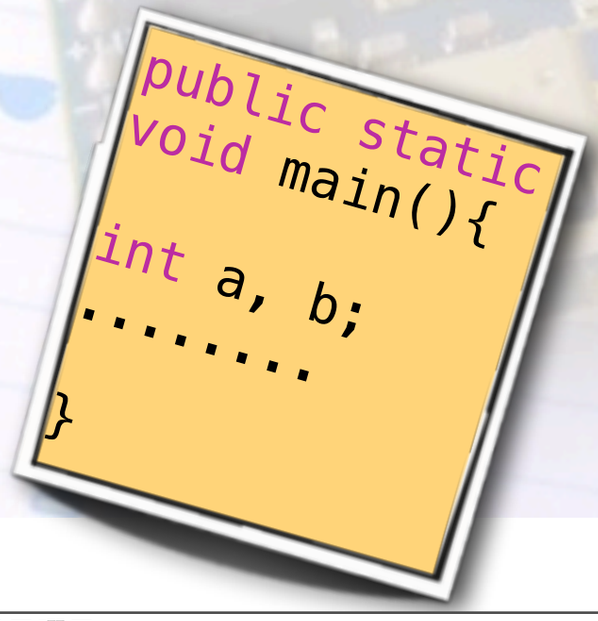
- Method of control GPIOs: portWrite(bit, value)
- Write the code Turning a LED on / off
- m…s)

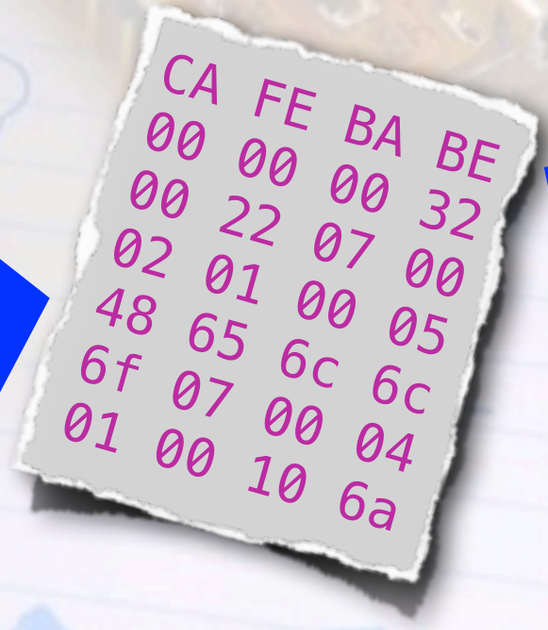
```
mb
Th
mb
Th
```

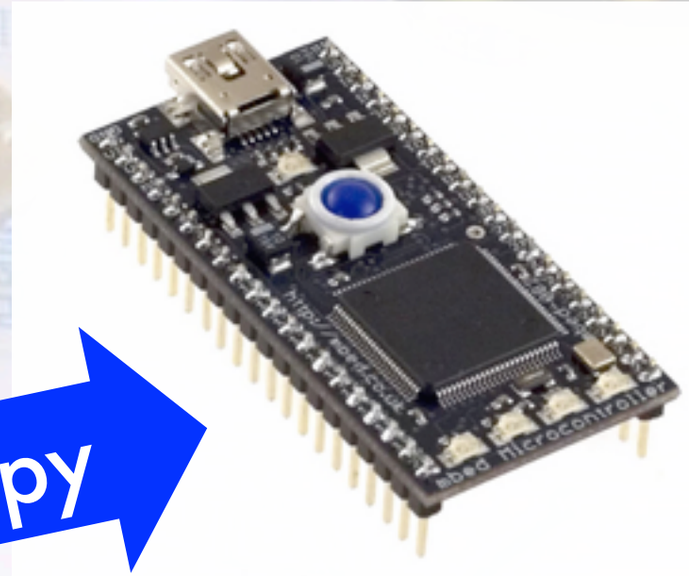
```
BlinkLED LED1 = new BlinkLED(0, 90);
BlinkLED LED2 = new BlinkLED(1, 100);
……

Thread th1 = new Thread(LED1);
Thread th2 = new Thread(LED2);
……

th1.start();
th2.start();
```

blink LEDs with JVM

- Write VM(.bin file) to mbed.
- getting byte code compiling on javac.
  - javac make us *.class file.
  - rename that “Test.cla” (8.3 extension),
- Copy to mbed,
- Reset to Start!



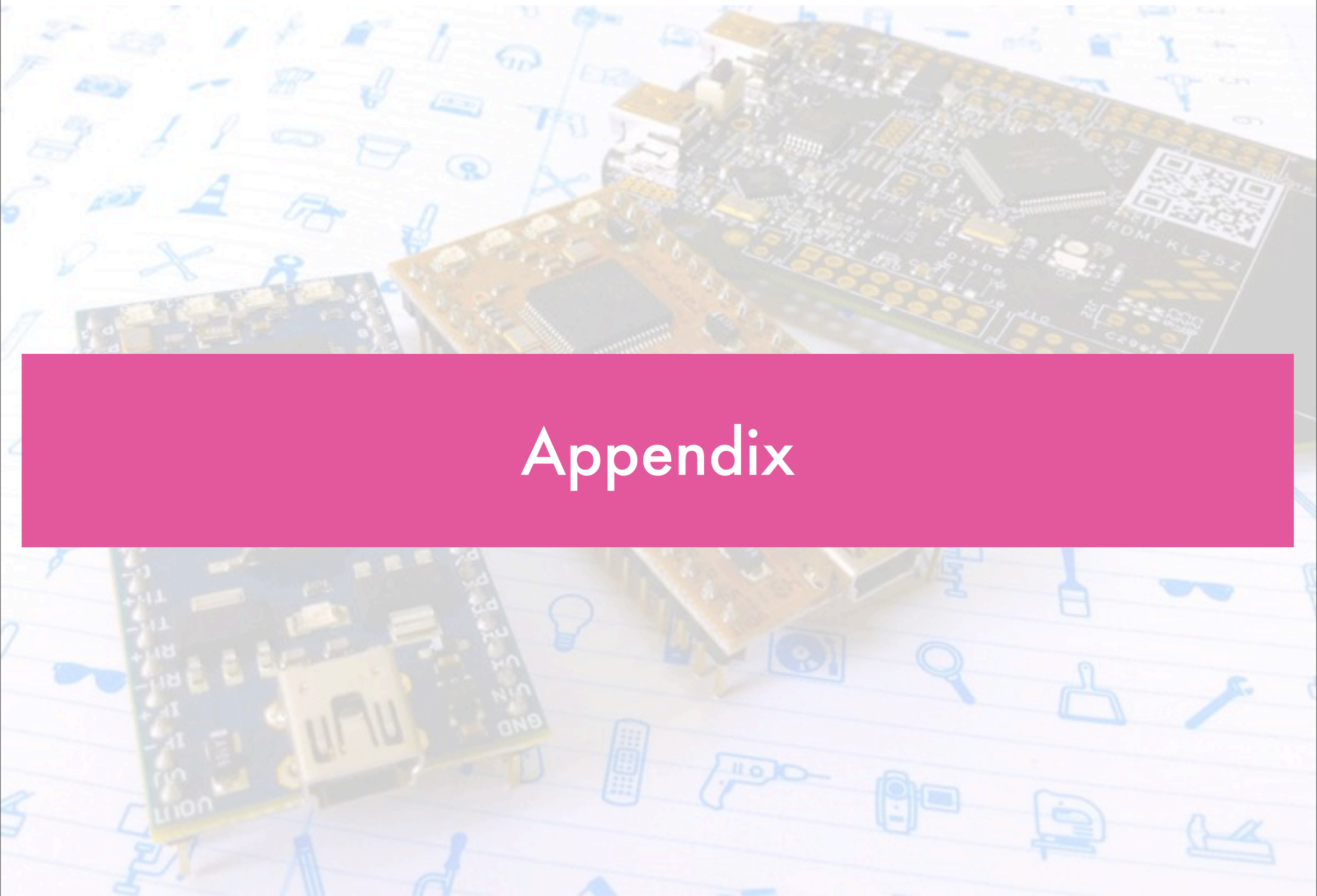




Conclusion

- Implemented Java VM to mbed or LPC81x.
  - mbed:D&D Java byte code, and execute.
  - others:make byte code to array, compile together with ALL JVM code.
- benefits: “write once run anywhere”.
- See also (written in Japanese only..)
  - http://lynxeyed.hatenablog.com/

Thank you for listening!



Appendix

RaVem JVM

- adding new method named “foobar”
- adding code in invokestatic_callFunction (in ravem.c)
  - if(strncmp(func_name,"foobar",6) == 0){.... }

- Adding “fake” methods in Java
  - You may not have to write the actual operation. if the method needs return value, you can write just return NULL :-)