(small) embeded system? use C++ !

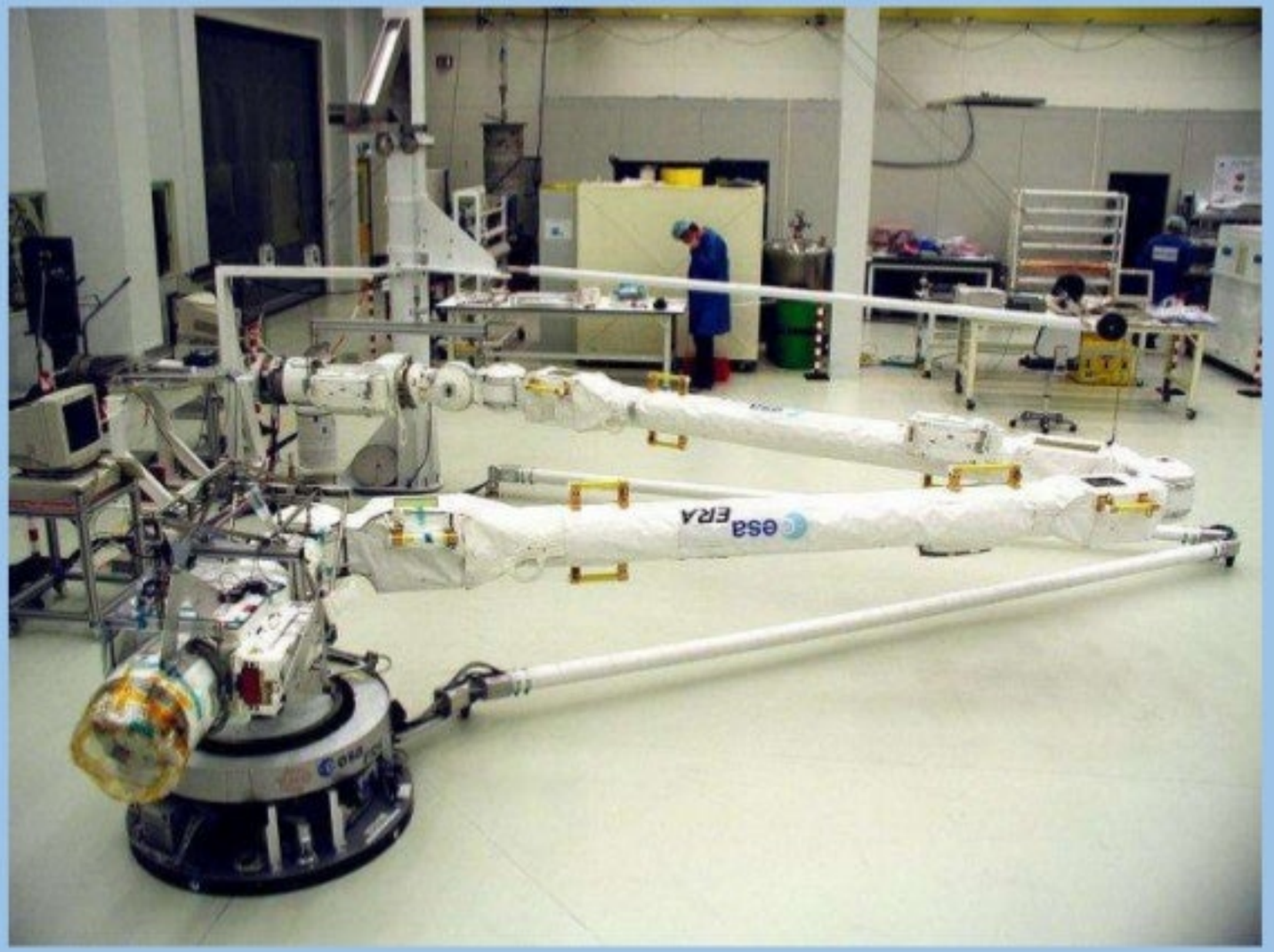
Wouter van Ooijen MSc (ir)
Hogeschool Utrecht, Netherlands,
senior lecturer Computer Engineering
SG14 member
wouter.vanooijen@hu.nl

CoreHard 2018 Minsk

A detour …..

Some 20 years ago I worked for a small Dutch space company

… on the the European Robotic Arm project



Which was delayed, delayed and delayed,

in-orbit replaceable units (ERUs) of ERA

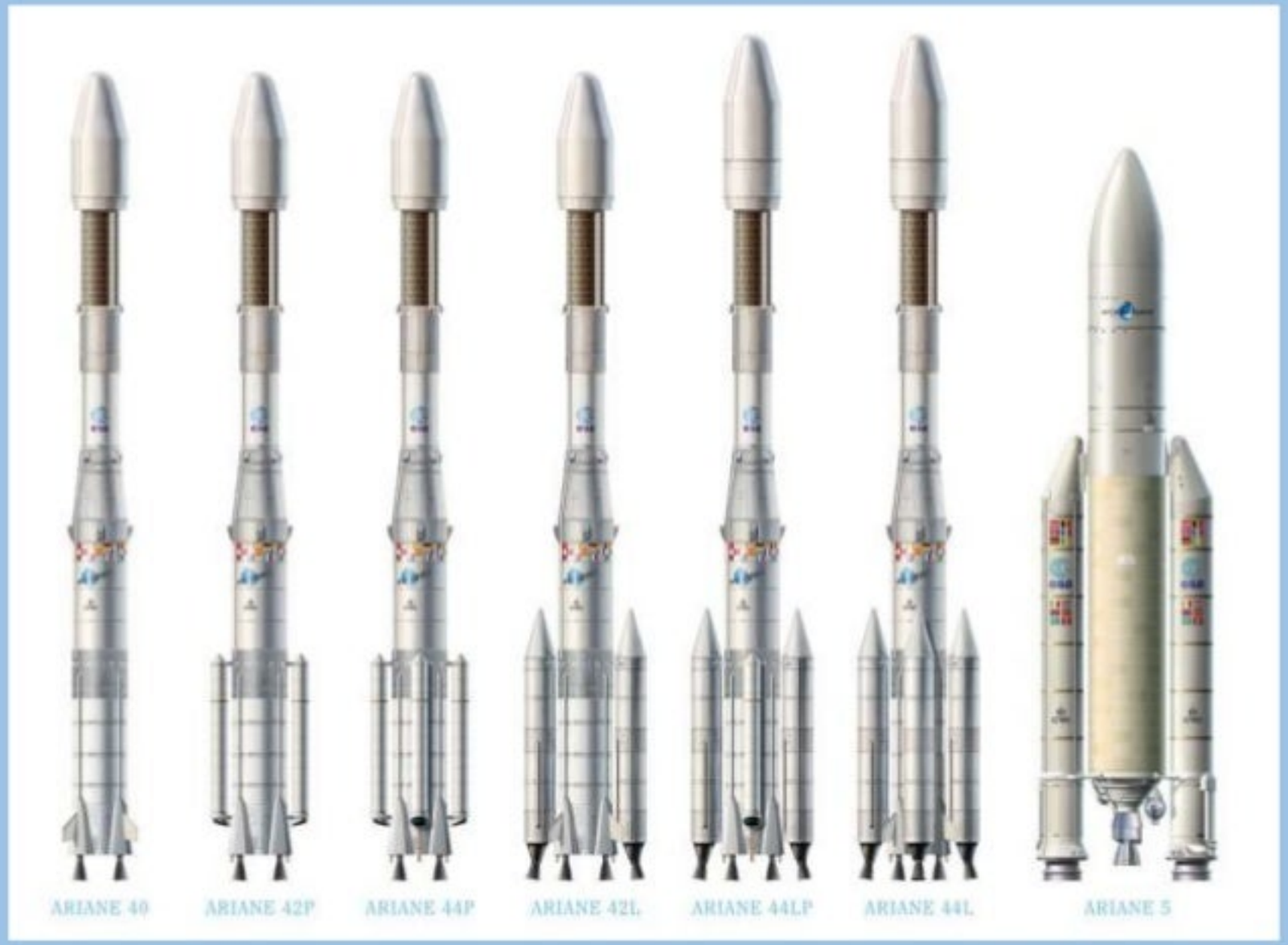
Project status [edit]

- 2005 - final qualification and delivery to the customer
- May 2010 - STS-132 - preemptive launch of a spare elbow joint with 2 limbs for ERA to the International Space Station
- Mid 2018 (?)- Launch of ERA with the Russian Multipurpose Laboratory Module on a Proton-M rocket[3][4]

and finally mothballed for lack of a purpose.

This page was last edited on 7 July 2017, at 19:20.

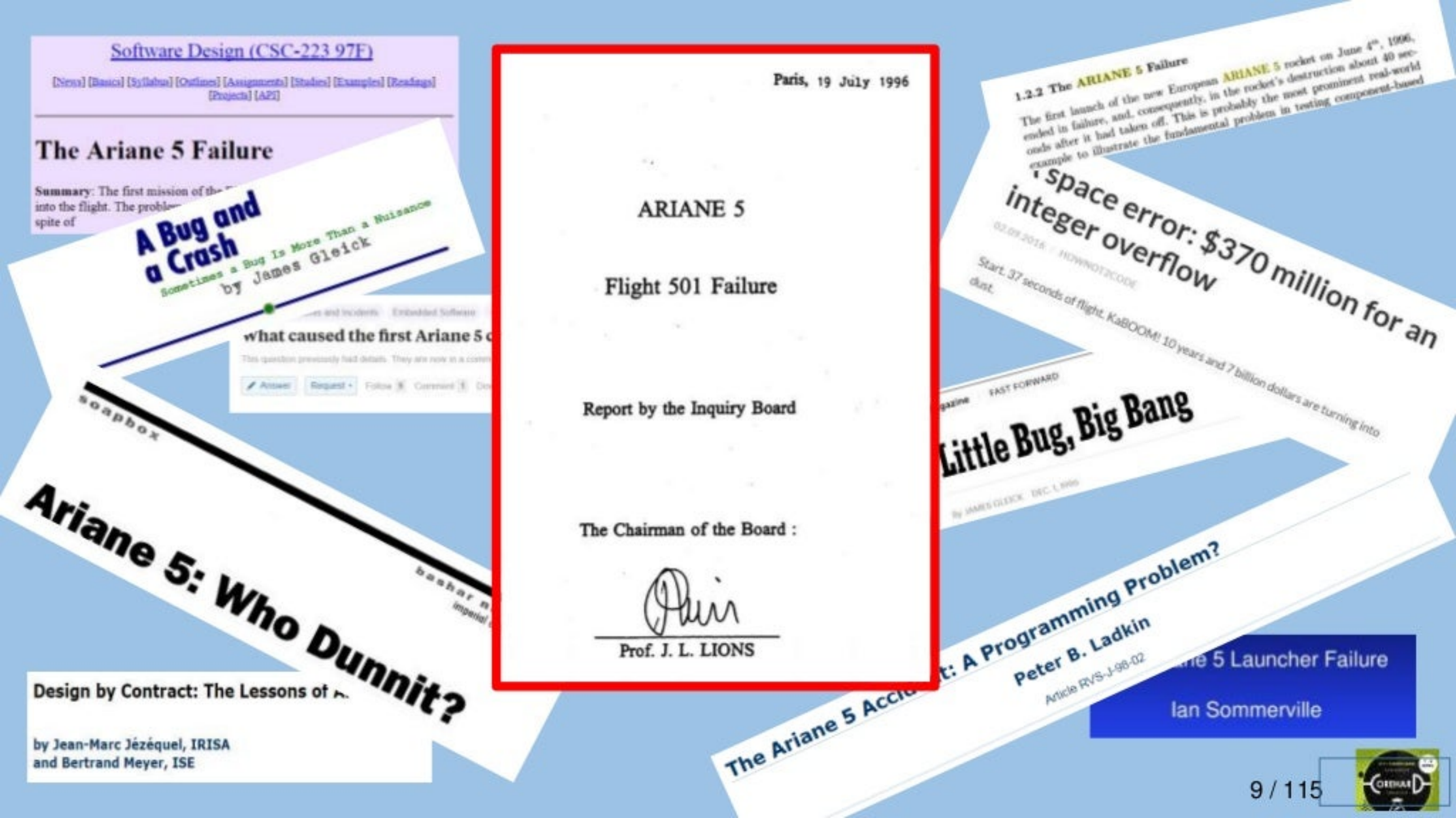
Ariane 4 ➔ Ariane 5

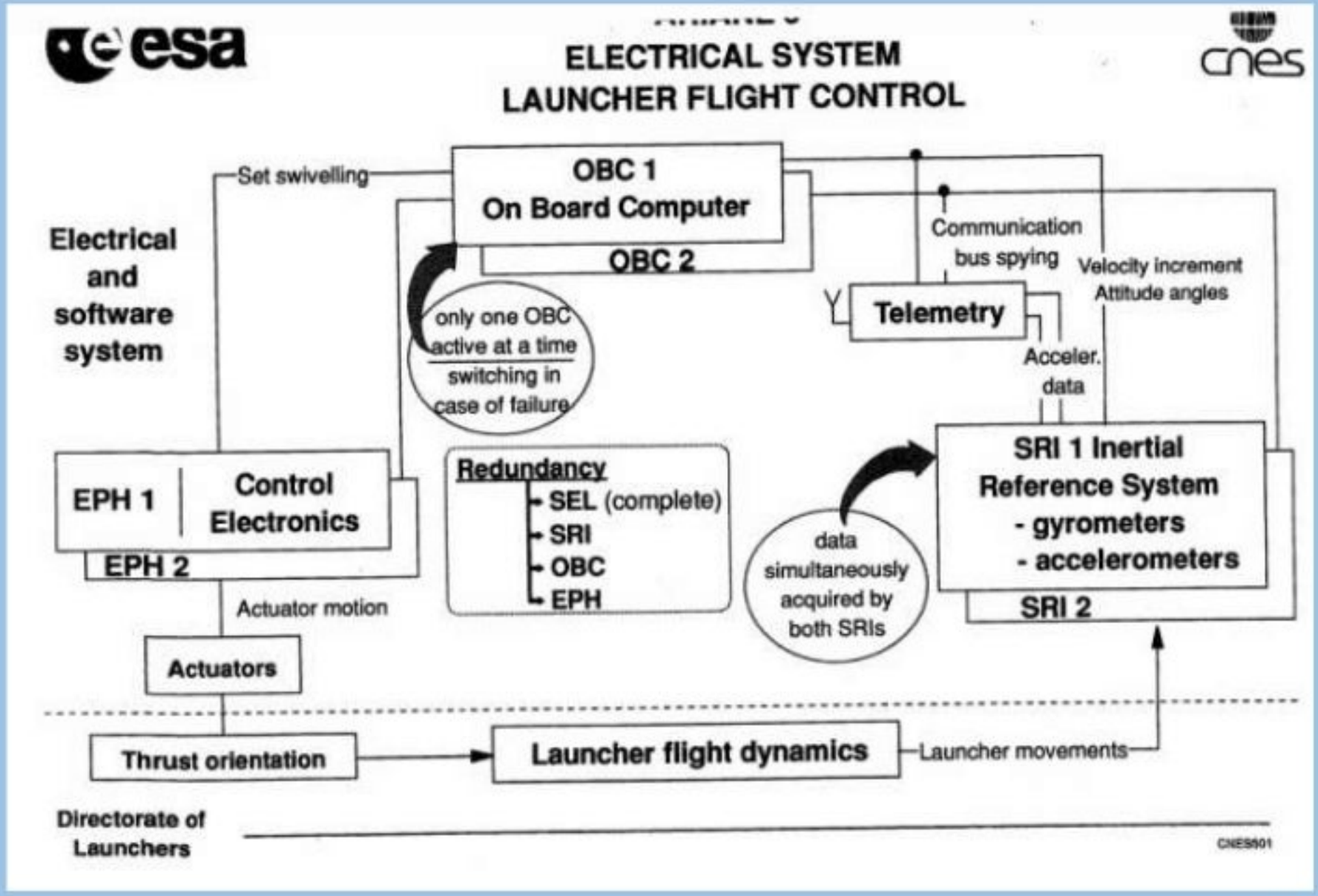


Big day
Ariane 5 maiden flight
Payment!


-00:00:17

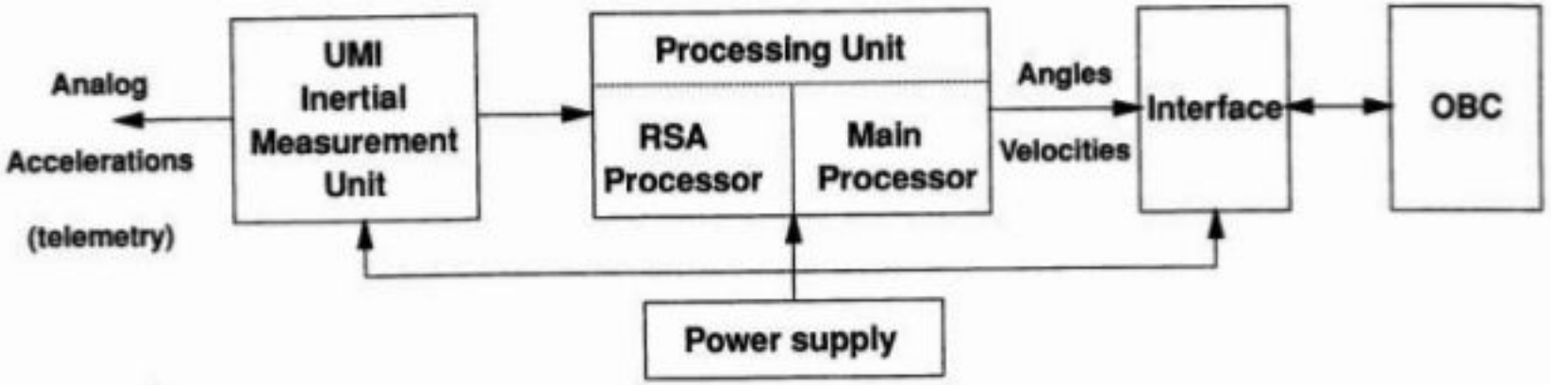

CHAMPAGNE
MONSIGNY
BRUT


Software Design (CSC-223 97F)
[News] [Basics] [Syllabus] [Outlines] [Assignments] [Studies] [Examples] [Readings] [Projects] [API]
The Ariane 5 Failure
Summary: The first mission of the
into the flight. The proble
spite of
A Bug and a Crash
Sometimes a Bug Is More Than a Nuisance
by James Gleick
what caused the first Ariane 5
Answer
Request
soapbox
Ariane 5: Who Dunnit?
Design by Contract: The Lessons of
by Jean-Marc Jézéquel, IRISA
and Bertrand Meyer, ISE
Paris, 19 July 1996
ARIANE 5
Flight 501 Failure
Report by the Inquiry Board
The Chairman of the Board :
Prof. J. L. LIONS
1.2.2 The ARIANE 5 Failure
The first launch of the new European ARIANE 5 rocket on June 4th, 1996, ended in failure, and, consequently, in the rocket's destruction about 40 seconds after it had taken off. This is probably the most prominent real-world example to illustrate the fundamental problem in testing component-based
'space error: $370 million for an integer overflow
Start. 37 seconds of flight. KaBOOM! 10 years and 7 billion dollars are turning into dust.
FAST FORWARD
Little Bug, Big Bang
The Ariane 5 Acci
t: A Programming Problem?
Peter B. Ladkin
Article RVS-J-96-02
5 Launcher Failure
Ian Sommerville



the Flight Control system

ARCHITECTURE



The SRIs were developed in common for Ariane 4 and Ariane 5
The only differences between the Ariane 4 and Ariane 5 versions are as follows:

	ARIANE 4	ARIANE 5
UMI	3 laser gyros 3 accelerometers	3 laser gyros 4 accelerometers
Power supply	28 V	55 V
Interface	Parallel 16 bits	Series 1553 B

Directorate of Launchers

CNES501

the Inertial Reference System

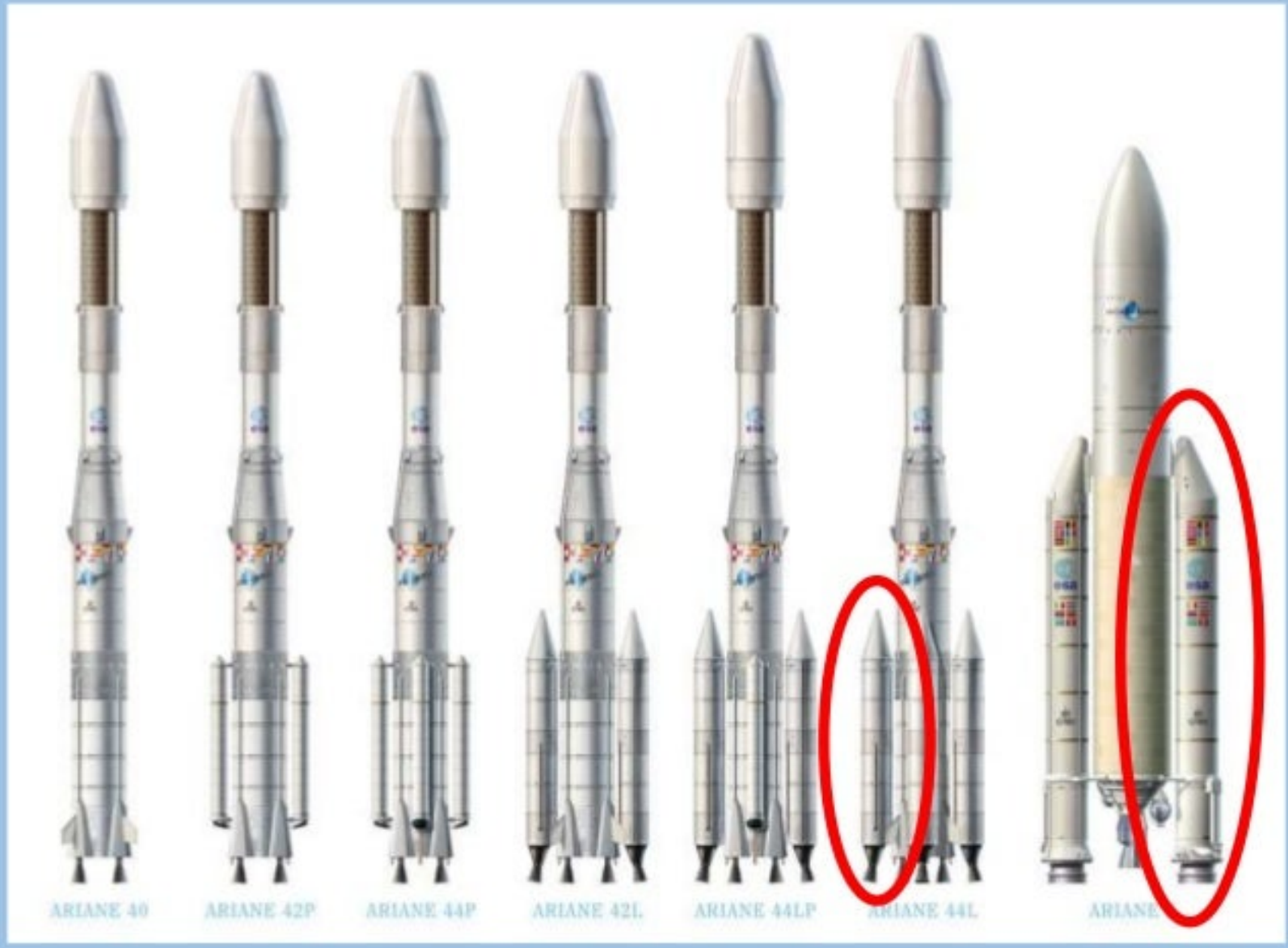
Design process…

- Requirement: CPU load at launch must be < 80%
- IRS CPU load is too high
- Analysis: physically HVB can’t be out of range
- Agreed solution: remove the checks for HVB

```
-- Vertical velocity bias as measured by sensor
L_M_BV_32 :=
   TBD.T_ENTIER_32S ((1.0/C_M_LSB_BV) *
   G_M_INFO_DERIVE(T_ALG.E_BV));
-- Check, if measured vertical velocity bias has be
-- converted to a 16 bit int. If so, then convert
if L_M_BV_32 > 32767 then
   P_M_DERIVE(T_ALG.E_BV) := 16#7FFF#;
elsif L_M_BV_32 < -32768 then
   P_M_DERIVE(T_ALG.E_BV) := 16#8000#;
else
   P_M_DERIVE(T_ALG.E_BV) :=
            UC_16S_EN_16NS(TDB.T_ENTIER_16S(L_M_BV_32));
end if;
-- Horizontal velocity bias as measured by sensor
-- is converted to a 16 bit  int without checking
P_M_DERIVE(T_ALG.E_BH) :=
   UC_16S_EN_16NS (TDB.T_ENTIER_16S ((1.0/C_M_LSB_BH) *
   G_M_INFO_DERIVE(T_ALG.E_BH)));
```

Chain of events

- Floating-point HRS value is out of 16-bits bounds
- Conversion exception occurs
- Is caugth at main level, application shutdown
- Switch to backup IRS fails
- Diagnostic information is put on the databus
- Is interpreted as flight data by the main computer
- Causes extreme steering
- Boosters break loose
- Self-destruct engages
- (Flight control commands self-destruct)

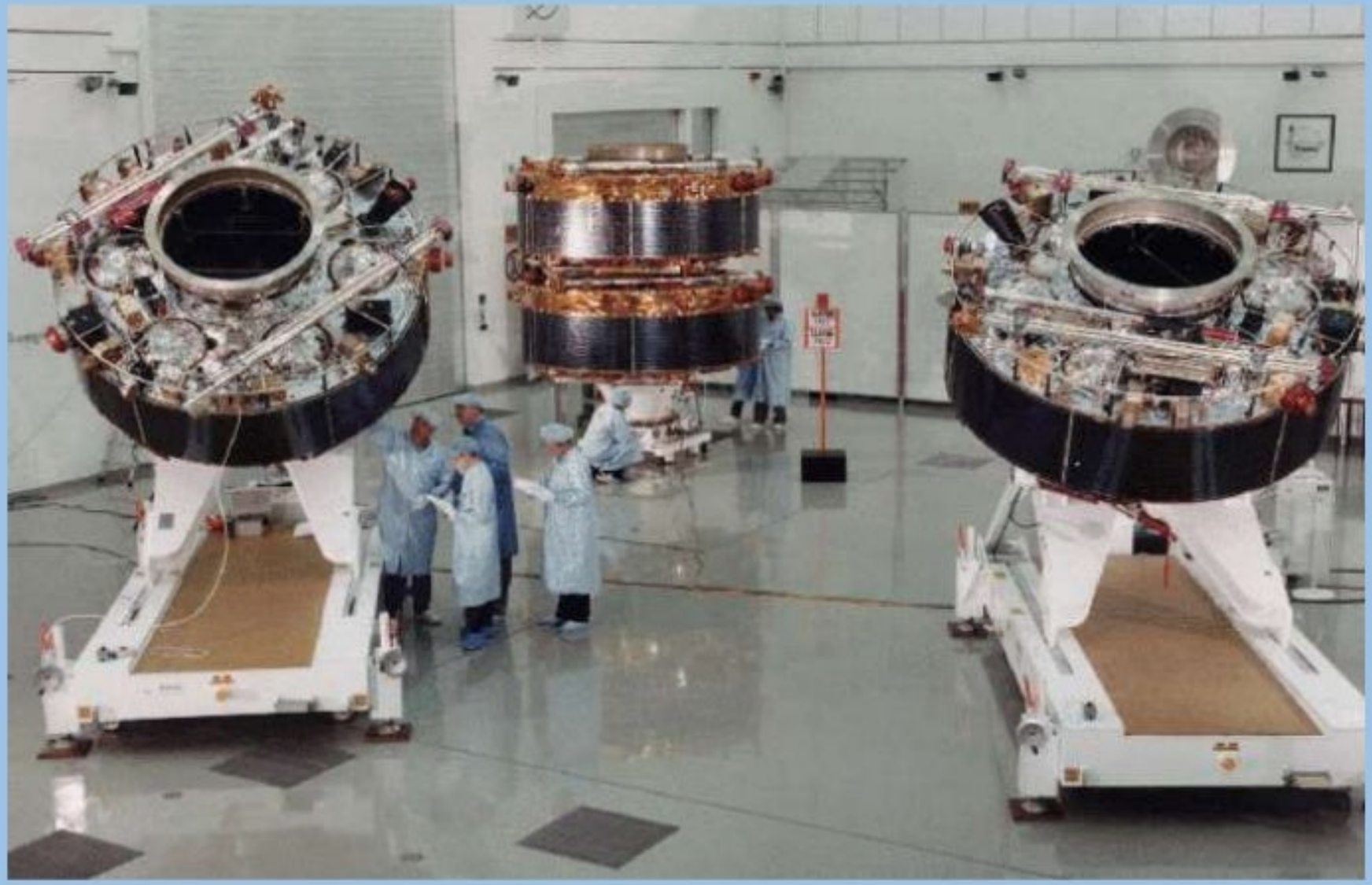


Spot the important difference

So who/what is to blame?

- Task running when no longer needed
- < 80 % CPU utilization requirement
- Using underpowered hardware
- Redundancy in HW but not in SW
- Stopping all tasks when an unexpected exception occurs
- Fail-totally instead of fail-gradually
- Re-using Ariane 4 software in Ariane 5 (untested)
- Having only ‘positive’ requirements
- Using Ada
- …… add your own?

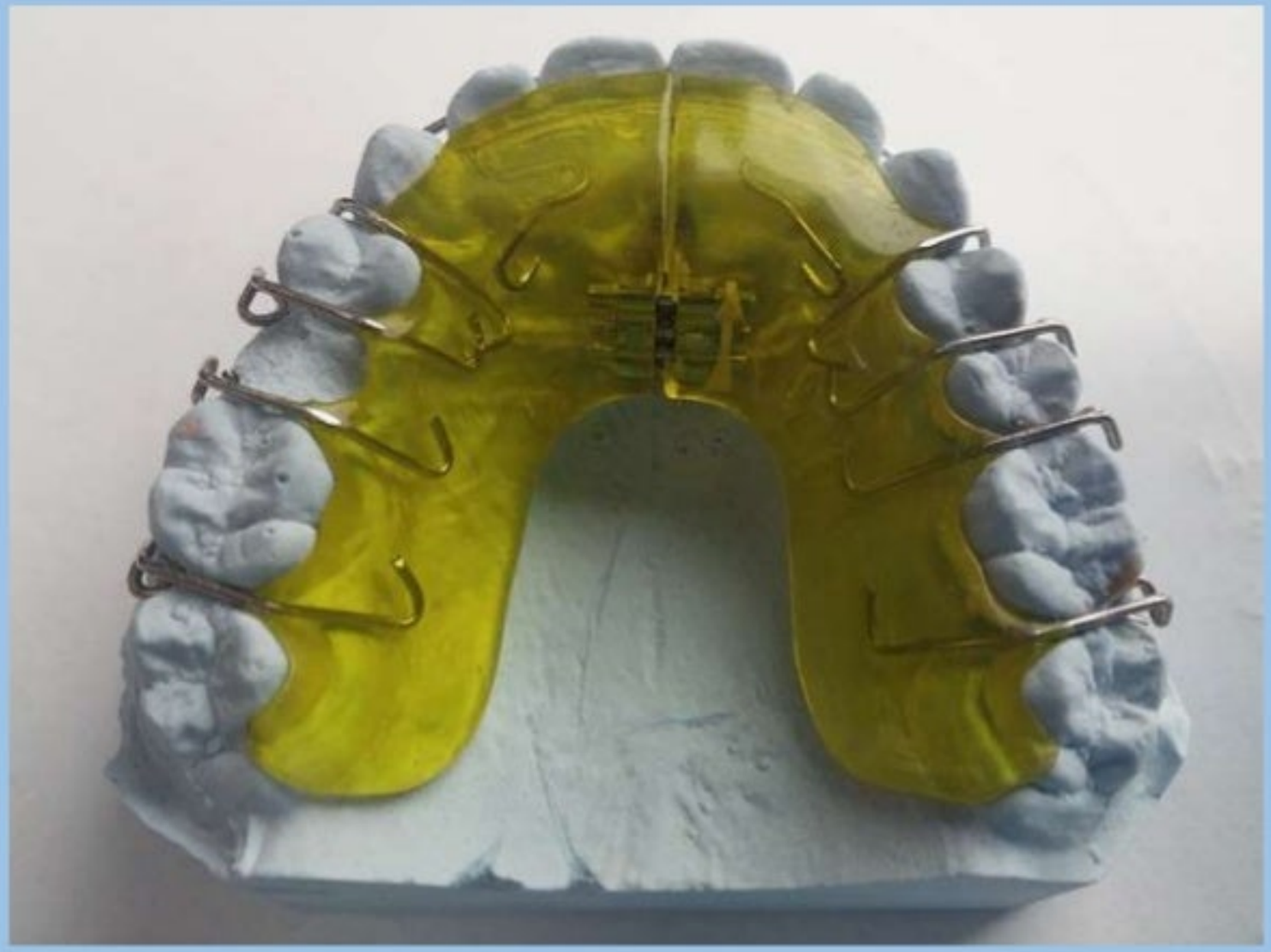
The cluster



Welcome to the embedded world – where things can go kaboom!

Not always *that* serious

Dental brace retainer



Airbag

Late might even be worse than not at all…

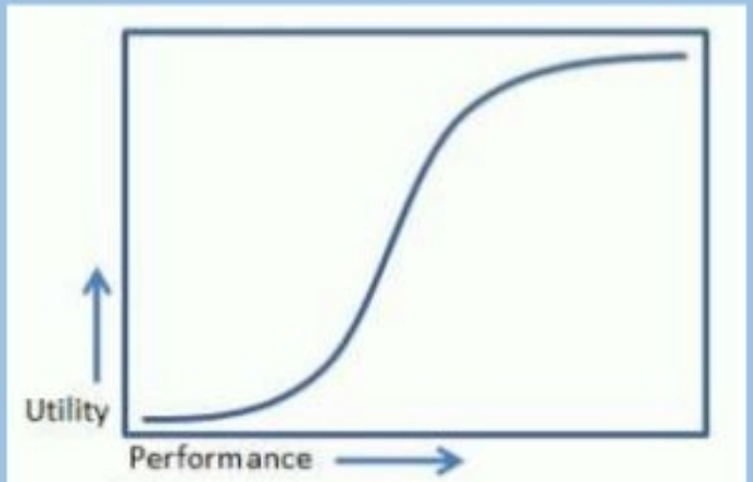
Wide span of seriousness

Wide span of response time

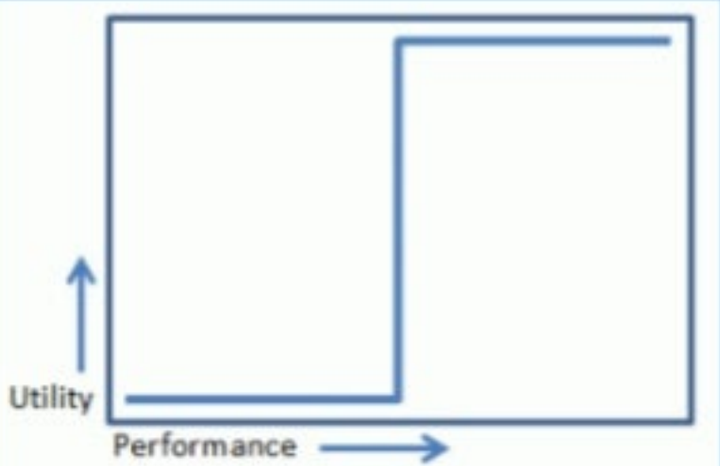
µs ms s

Real-time

A throughput-oriented system



An interactive system



A real-time system

Wide span of replication

1’000’000

100’000

10’000

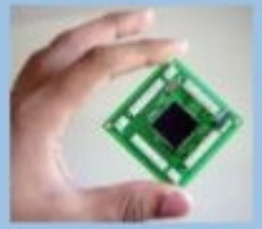
100

1

Disclaimer: all figures are just wild guesses

Wide span of hardware

1KiB 1MiB 1GiB



4-bit

64-bit

8-bit : 8051, PIC, AVR, …

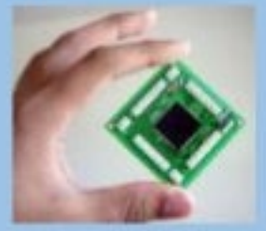
No full standard C/C++

16-bit : MSP430, PIC, …

32-bit : ARM, MIPS, Tensilca, Intel, …

Wide span of support software

1KiB    1MiB  1GiB



Bare metal:

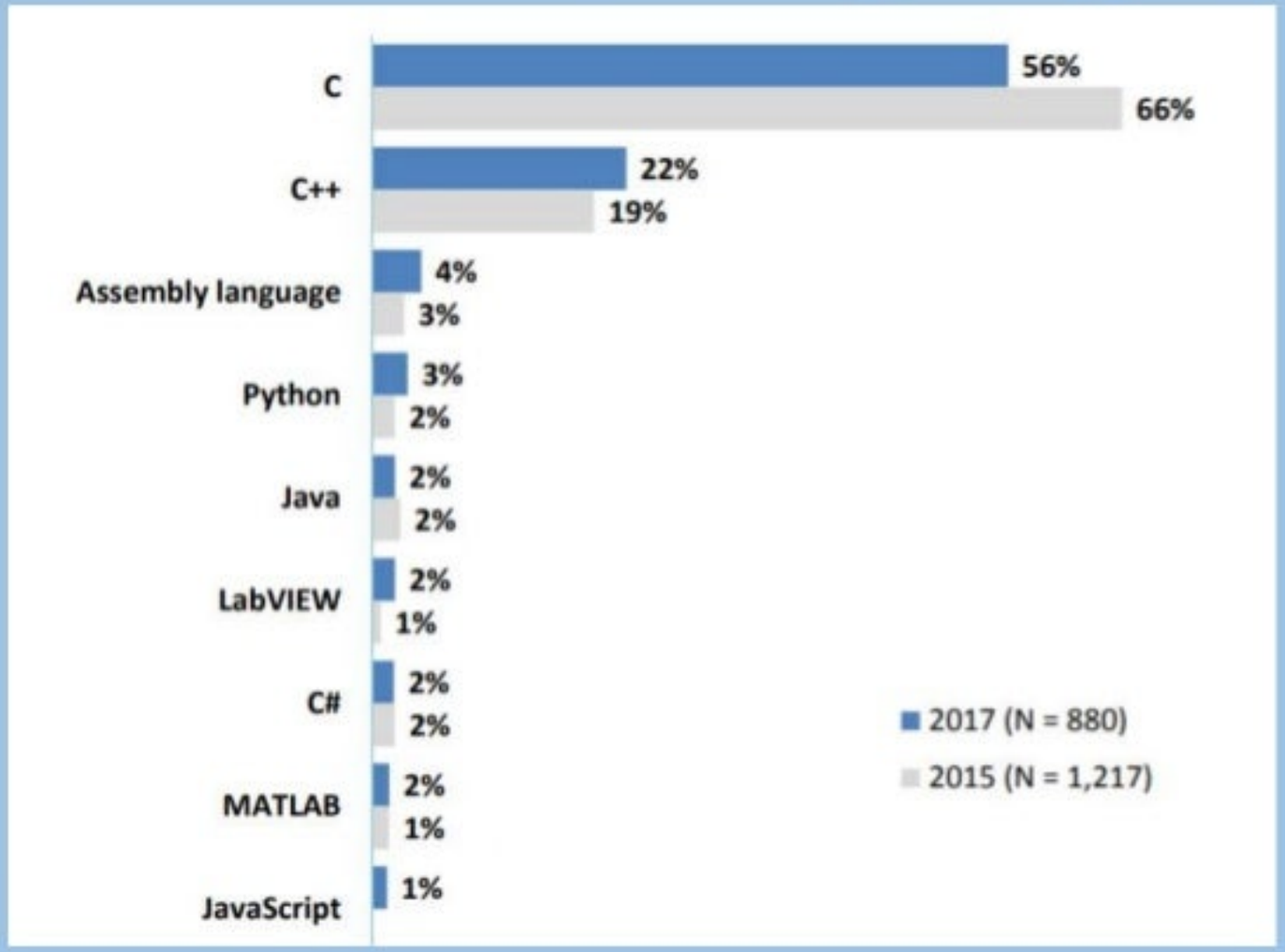
- libraries
- no separate OS

- Interpreter
- RTOS

Desktop OS:

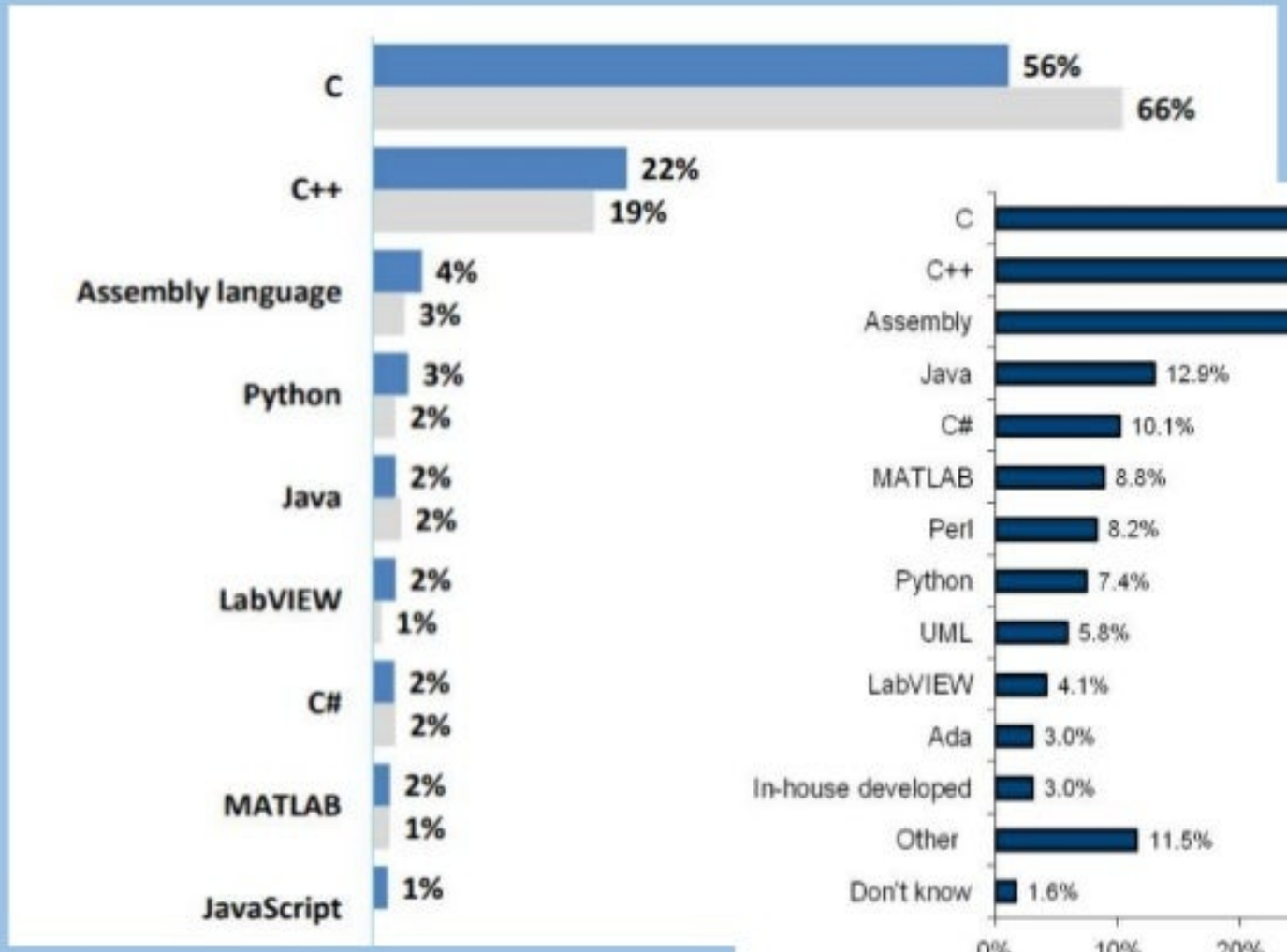
- Linux
- Windows

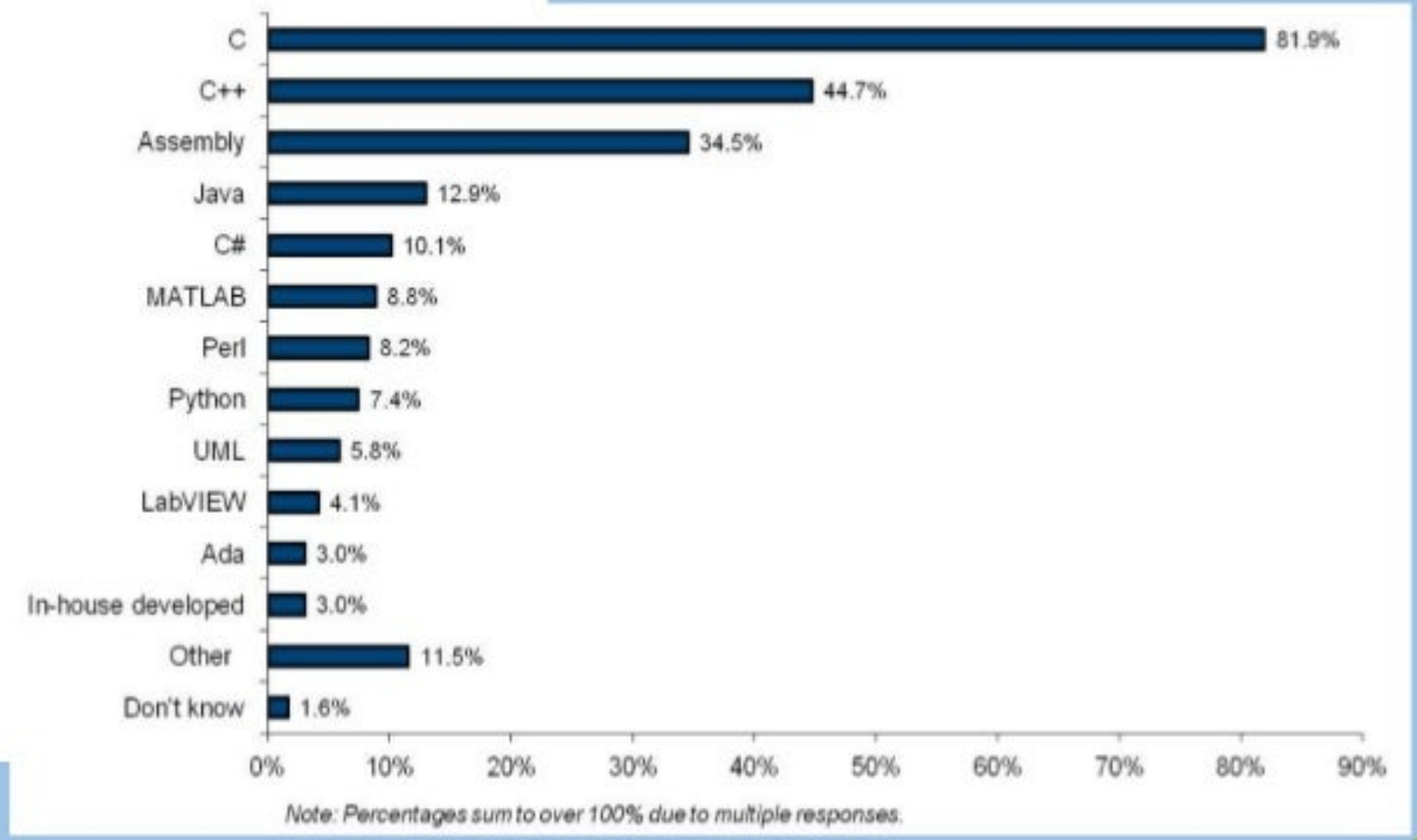
Embedded languages



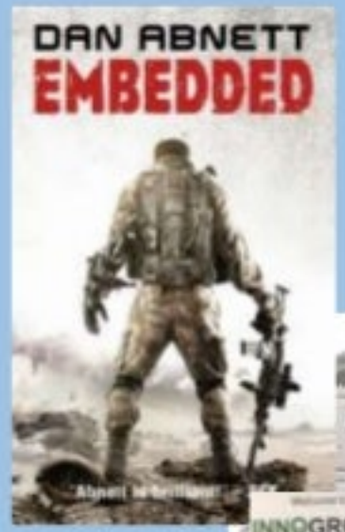
AspenCore: my current embedded project is programmed mostly in…

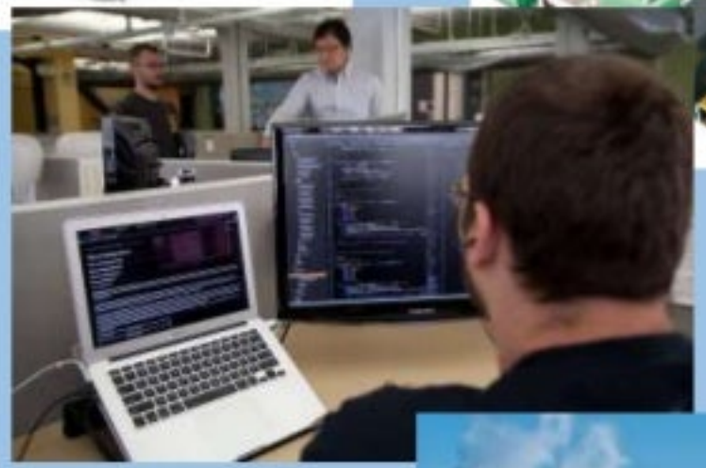
Embedded languages





2010 VDC survey



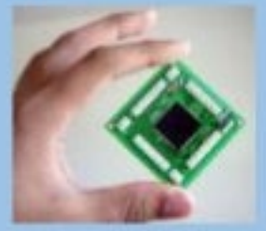


Embedded

=

Not a usable term to define a (single) programming style

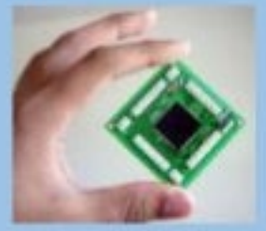
Let's quantify



	10F200	Arduino Uno	Arduino Due	Raspberry Pi	Desktop
Code	375 B	32 KiB	512 KiB	8 GiB	1 TiB
Data	16 B	2 KiB	96 KiB	1 GiB	4 GiB
CPU	8 bit	8 bit	32 bit	32 bit	64 bit
Clock	4 MHz	16 MHz	84 MHz	900 MHz	3 GHz

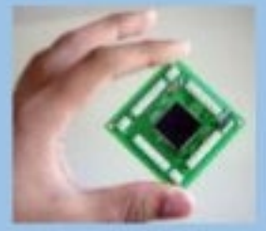
Real numbers

1KiB **1MiB** **1GiB**



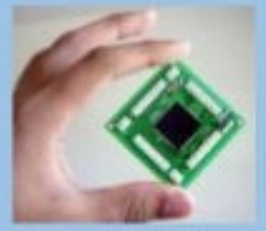
	10F200	Arduino Uno	Arduino Due	Raspberry Pi	Desktop
Code	375	32768	524288	8589934592	1099511627776
Data	16	2048	98304	1073741824	4294967296
CPU	1000000	16000000	168000000	3600000000	30000000000

Relative



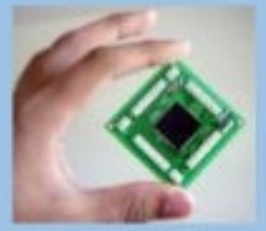
	10F200	Arduino Uno	Arduino Due	Raspberry Pi	Desktop
Code	1	87	1398	22906492	2932031007
Data	1	128	6144	67108864	268435456
CPU	1	16	128	3600	30000

10 * Log



	10F200	Arduino Uno	Arduino Due	Raspberry Pi	Desktop
Code	0.000000	44.70282	72.42870	169.46931	217.98961
Data	0.000000	48.52030	87.23231	180.21827	194.08121
CPU	0.000000	27.72588	51.239639	81.886891	103.089526

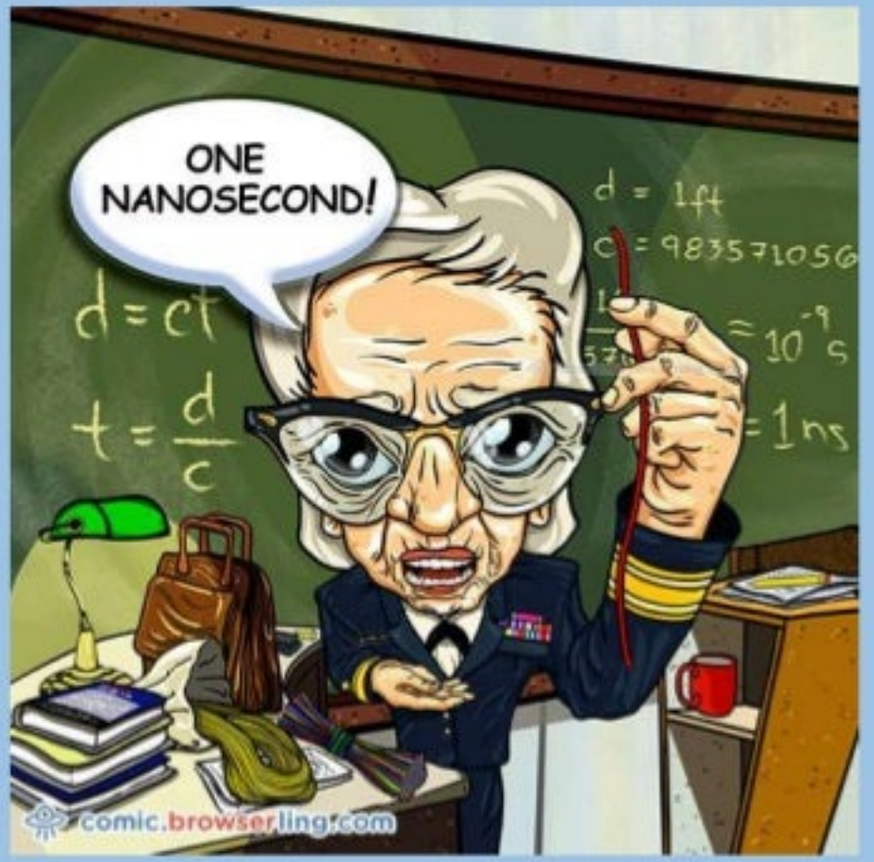
Rounded 10 * log



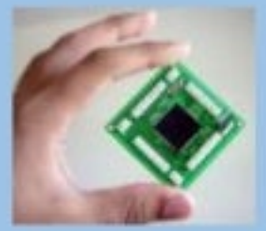
	10F200	Arduino Uno	Arduino Due	Raspberry Pi	Desktop
Code	0	45	72	169	218
Data	0	49	87	180	194
CPU	0	28	51	82	103

Unit of (log) size?

Unit of (log) speed?



Relative, in dMore and dHopper



	10F200	Arduino Uno	Arduino Due	Raspberry Pi	Desktop
code dMoore	0	+45	+27	+97	+49
data Moore	0	+49	+38	+93	+14
dHopper	0	+28	+31	+31	+21

relative to previous column

No alternative

	10F200	Arduino Uno	Arduino Due	Raspberry Pi	Desktop
code dMoore	0	+45	+27	+97	+49
data Moore	0	+49	+38	+93	+14
dHopper	0	+28	+31	+31	+21

Assembler

I am actually doing electronics diguised as software.

Maybe I'll consider C

You can afford

RTOS

GP OS

	10F200	Arduino Uno	Arduino Due	Raspberry Pi	Desktop
code dMoore	0	+45	+27	+97	+49
data Moore	0	+49	+38	+93	+14

Interpreter – I run desktop code in a small chip

Where does C++ fit in?

Assembler

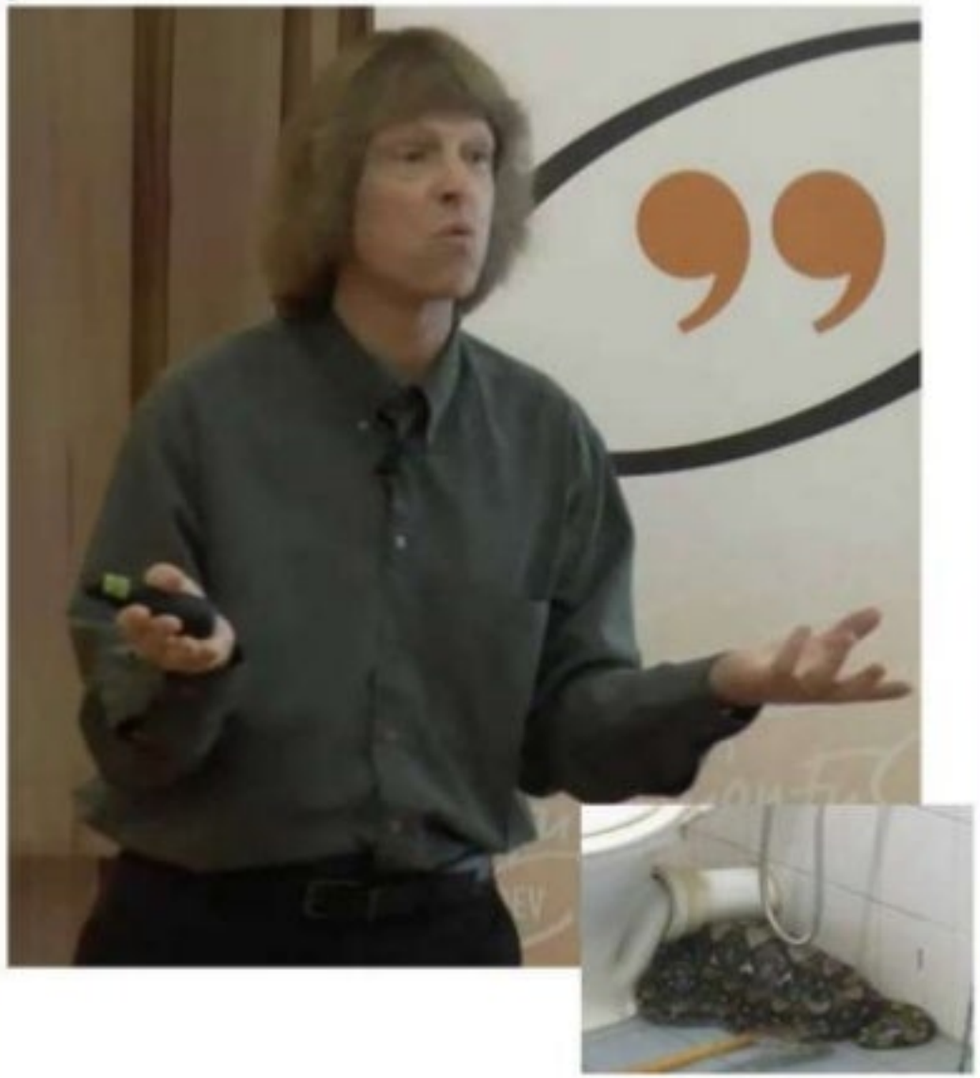
Interpreter

RTOS

GP OS

	10F200	Arduino Uno	Arduino Due	Raspberry Pi	Desktop
code dMoore	0	+45	+27	+97	+49
data Moore	0	+49	+38	+93	+14
dHopper	0	+28	+31	+81	+21

If you need performance ...



If you're not at all interested in performance, shouldn't you be in the Python room down the hall?

[Scott Meyers, Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14]

Overall Ranking

Python is the No. 1

Language Types (click to hide)

Web | Mobile | Enterprise | Embedded

Language Rank	Types	Spectrum Ranking
1. Python	Web, Enterprise	100.0
2. C	Mobile, Enterprise, Embedded	99.7
3. Java	Web, Mobile, Enterprise	99.4
4. C++	Mobile, Enterprise, Embedded	97.2
5. C#	Web, Mobile, Enterprise	88.6
6. R	Enterprise	88.1
7. JavaScript	Web, Mobile	85.5
8. PHP	Web	81.4
9. Go	Web, Enterprise	76.1
10. Swift	Mobile, Enterprise	75.3
11. Arduino	Embedded	73.0

Vs.

Only Embedded

What happened to Python?

Language Types (click to hide)

Web | Mobile | Enterprise | Embedded

Language Rank	Types	Spectrum Ranking
1. C	Mobile, Enterprise, Embedded	99.7
2. C++	Mobile, Enterprise, Embedded	97.2
3. Arduino	Embedded	73.0
4. Assembly	Embedded	72.1
5. Haskell	Enterprise, Embedded	48.5
6. D	Web, Embedded	38.8
7. VHDL	Embedded	36.7
8. LabView	Enterprise, Embedded	32.6
9. Verilog	Embedded	32.0
10. Erlang	Enterprise, Embedded	28.0
11. Ada	Enterprise, Embedded	26.4

C++ versus Python

Actually compiled versus interpreted

	Penalty	
Size	3 .. 13 dMoore	
Speed	3 .. 23 dHopper	

C++ versus Python

Actually compiled versus interpreted

	Penalty	
Size	3 .. 13 dMoore	(factor 2 .. 20)
Speed	3 .. 23 dHopper	(factor 2 .. 200)

Where does C++ fit in?

Assembler Interpreter RTOS GP OS

	10F200	Arduino Uno	Arduino Due	Raspberry Pi	Desktop
code dMoore	0	+45	+27	+97	+49
data Moore	0	+49	+38	+93	+14
dHopper	0	+28	+31	+81	+21

small-embedded C++ alley

Size & speed margin

Very low margin:
Use Assembler

Comfortable margin:
Consider using an interpreter

dMoore	0	+10	+20		
dHopper	0	+10	+20		

small-embedded C++ alley

MiB, GiB
GHz, multi-core
Memory latency, caches
Worry about latency

large-embedded

	10F200	Arduino Uno	Arduino Due	Raspberry Pi	Desktop

small-embedded

bytes, KiB, worry about size
MHz, Interrupts
Hardware peripherals
Battery life-time


fast trading
gaming
simulation
SG14:
low-latency
large-embedded
small-embedded

What does C++ do for embedded?

- ADTs: abstraction, data hiding, type-checking, enum class
- Overloading, operators
- Namespaces
- Argument defaults, references, std::array<>
- Const, constexpr, constexpr if
- }, RAII
- Templates
- DSLs (using templates, operators, overloading)

- ADTs: abstraction, data hiding, type-checking, enum class
- Overloading, operators
- Namespaces
- Argument defaults, references, std::array<>
- Const, constexpr, constexpr if
- }, RAII
- Templates
- DSLs (using templates, operators, overloading)

Note that most:

- **Are zero cost**
- **Were NOT included for embedded**
- **Help to reduce MACRO'S**

Less usefull?

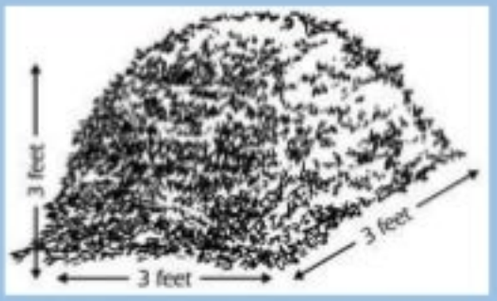
- Heap : (small-) embedded is often ‘mission-critical’
- Exceptions : idem
- Classic OO : not free
- Bit-fields : not enough control, and can be done adequately (and encapsulated!) with other means
- The libraries (large - but unspecified! - parts can’t be used)

Annoying

All (library) assumptions that are hard-coded

- **Extra space is allocated on *the heap***
- **Integer calculations are done using *int***
- **Floating-point calculations are done using *double***
- **Special situations are reported by *exceptions***
- **This function must be *fast* / *small***
- **Caching determines speed, *so avoid linked lists***
- **All that matters is *amortized complexity***

Heap



The heap is flexibility with respect to the amount of data, at the cost of (some) unpredictability in run-time and maximum available memory (fragmentation).

A typical small embedded system

- **has a rigidly prescribed task, including the size of the data involved**
- **Must meet real-time constraints**
- **Should be certain to meet work OK all the time (not just for a certain sequence of actions).**

→A heap and a small embedded system don't match very well. Better:

- **Global allocation (!)**
- **Fixed size pools or fixed size containers**

Exceptions

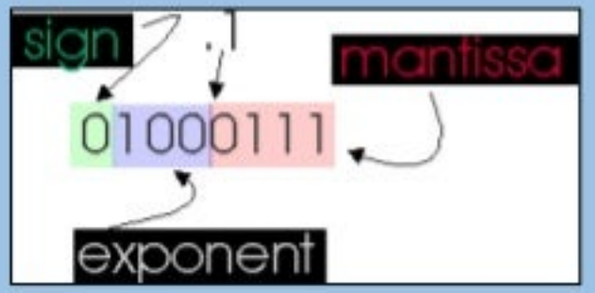
Exceptions are great to handle a local problem that requires a (more) global response.

- A small system often has a rather simple, rigidly defined task
  → there are no exceptions, only different situations.

- No heap ➔ one reason less to use exceptions.
- Exception implementation often uses the heap….

Straw man: exceptions are slow and have unpredictable timing.

Floating point



FP is useful when a wide dynamic range is needed. For an embedded application the ranges are often very well known. (but… Ariane 5?)

Small micro-controllers often don't have a hardware FPU. A software FP library

- Is considerably slower
- Takes up ROM

Make your software 'open':

```
template< typename value_type = double >
class your_amazing_library {
   . . .
```

”Just” print

```
struct location { int x, y; };
```

```
void print_location( FILE * f, location v ){
   fprintf( f, "(%d,%d)", v.x, v.y );
}
```

```
std::ostream & operator<<( std::ostream & out, const location & v  ){
   return out << '(' << v.x << ',' << v.y << ')';
}
```

```
template< typename OUT >
auto operator<<( OUT & out, const location & v  )
   -> decltype( out << 'x' ) &
{
   return out << '(' << v.x << ',' << v.y << ')';
}
```

General
embedded / low-latency / standalone ‘profile’

- No heap use (in the critical code)
- No exceptions thrown or caught (in the critical code)
- No RTTI

Small-embedded profile

- No heap included
- Compiled without exceptions
- No RTTI
- No floating point
- No OS support, no file system, no std::cout

Embedded 'paranoia' profiles

- No heap included
- Compiled without exceptions
- No RTTI
- No floating point
- No OS support, no file system, no std::cout
- Analyzable stack size: no recursion, no indirection
- Analyzable run-time: no unbounded loops

What does C++ NOT (yet) do for small-embedded

- Comprehensible, single-way-of-doing-things language
- Usable set of libraries
- Examples (books, websites, blogs, etc.) that show *the* (or even *an*) appropriate modern programming style

Why don’t they all use C++?

- No compiler available (PIC, 8051)
- No programmers available
- Not allowed
- Existing code base (often in C)
- The style is inappropriate
- It is too complex (doesn’t show what happens)
- Code bloat
- No advantages

Why don’t they all use C++?

- No compiler available (PIC, 8051)
- No programmers available
- Not allowed
- Existing code base (often in C)
- The style is inappropriate
- It is too complex (doesn’t show what happens)
- Code bloat
- No advantages

Remark from The Embedded Muse (salary) survey:

At my job we are developing embedded software using C++11 and there are exceedingly few resources on how to develop embedded systems using modern C++. We try to stick to industry best practices, but a lot of the time there is a lack of standards (or even discussion) regarding parts of C++ that are now 7+ years old. I would love to see more discussion and use of modern C++ in the embedded community.

http://www.ganssle.com/tem/tem347.html

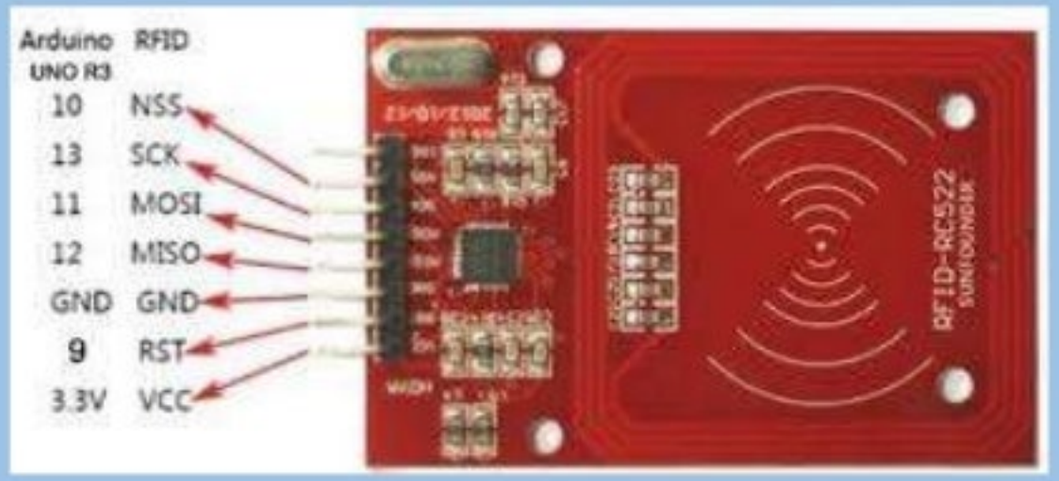
What's in the pipeline

- Concepts
- Class types in non-type template parameters
- Freestanding
- Ranges

Samll example: take a C-style ‘C++’ library

Arduino library for

- Reading RFID tags
- Using MRFC522 chip



- SPI interface
- Can read/write registers on the chip
- Special command register

Use enum classes

C-style 'C++'

```
#define     CommandReg          0x01
#define     CommIEnReg          0x02
#define     DivlEnReg           0x03
#define     CommIrqReg          0x04
#define     DivIrqReg           0x05


void writeMFRC522( uint8_t addr, uint8_t val );

writeMFRC522( DivlEnReg, 0x80 );

writeMFRC522( 0x80, DivlEnReg );
```

Modern C++

```
enum class reg : uint8_t {
     Command          = 0x01,
     ComIEn           = 0x02,
     DivIEn           = 0x03,
     ComIrq           = 0x04,
     DivIrq           = 0x05,
     . . .

void write( reg r, uint8_t d );

write( reg::DivlEnReg, 0x80 );

write( 0x80, reg::DivlEnReg );
```

- Unscoped #define -> scoped enum class
- Enum class name is part of the name
- No name clashes
- Strong typing

Strong typing + overloading

C-style 'C++'

```
#define     CommandReg              0x01
#define     CommIEnReg              0x02
#define     DivlEnReg               0x03
#define     CommIrqReg              0x04
#define     DivIrqReg               0x05


void writeMFRC522( uint8_t addr, uint8_t val );

writeMFRC522( DivlEnReg, 0x80 );


#define PCD_IDLE                0x00
#define PCD_AUTHENT             0x0E
#define PCD_CALCCRC             0x03


writeMFRC522( CommandReg, PCD_CALCCRC );
```

Modern C++

```
enum class reg : uint8_t {
     Command            = 0x01,
     ComIEn             = 0x02,
     DivIEn             = 0x03,
     ComIrq             = 0x04,
     DivIrq             = 0x05,
     . . .

void write( reg r, uint8_t d );

write( reg::DivlEnReg, 0x80 );

enum class cmd : uint8_t {
     Idle                   = 0x00,
     MFAuthent              = 0x0E,
     CalcCRC                = 0x03,
     . . .

void write( cmd d );

write( cmd::CalcCRC );
```

Define the function in the header -> compiler can inline when appropriate

Template parameter, const, implicit size, overloading

C-style ‘C++’

```
void writeMFRC522( uint8_t addr, uint8_t val );


void RFID::calculateCRC(
   uint8_t *pIndata, uint8_t len
{
   uint8_t i, n;

   for( i=0; i<len; i++ )
      writeMFRC522( FIFODataReg, *(pIndata+i) );
   . . .
```

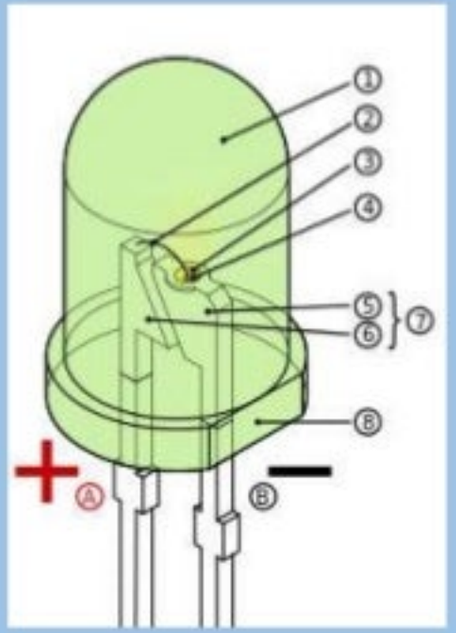
Modern C++

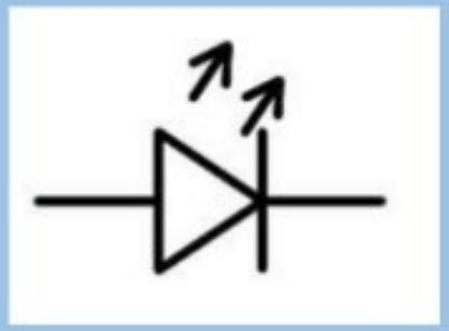
```
void write( reg r, uint8_t d );

template< size_t n >
void write(
   reg r,
   const std::array< uint8_t, n > & data );
```

Watch out for template bloat -> maybe (!) use flyweight pattern

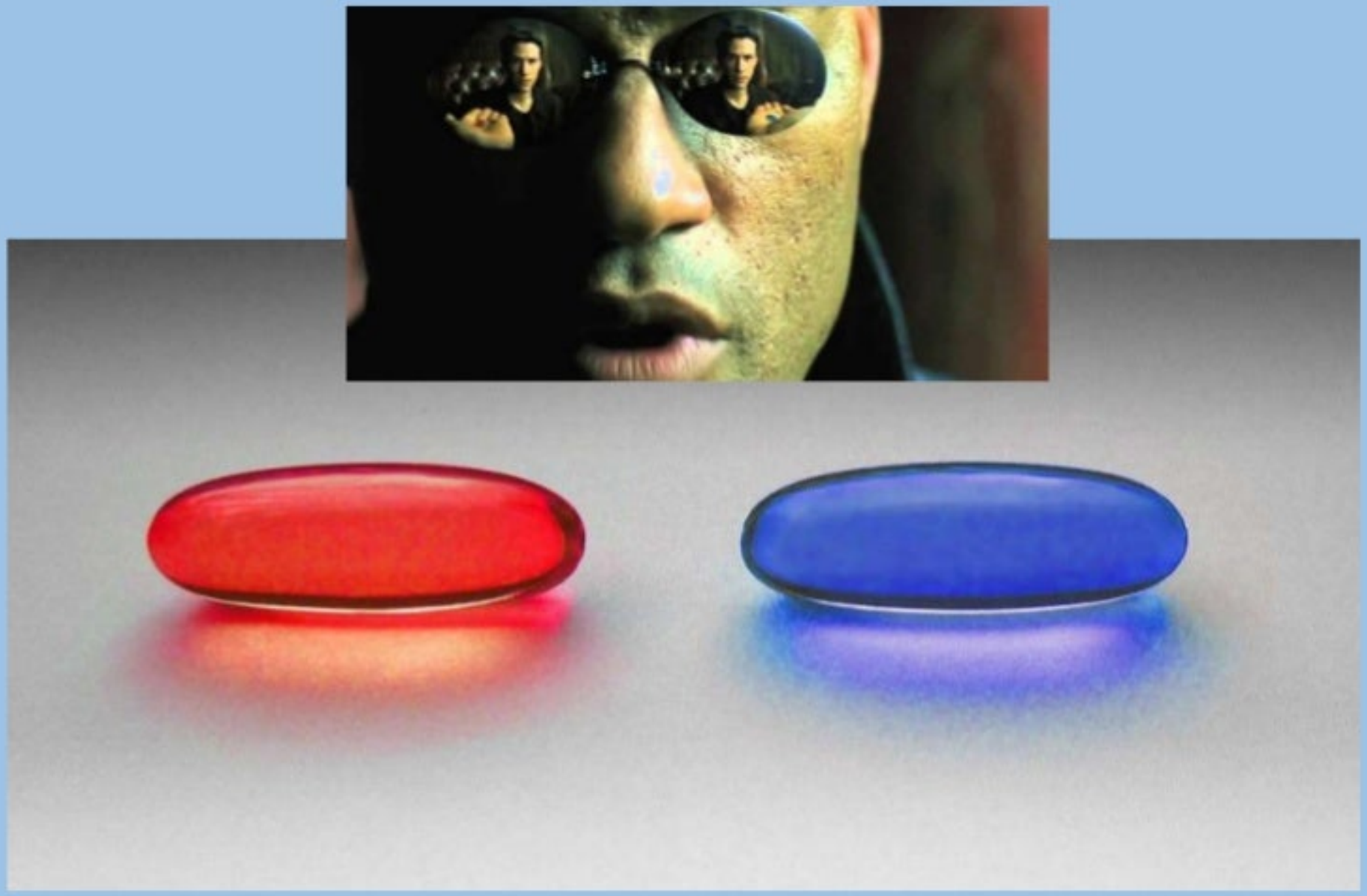
(The) hard-embedded programming example:





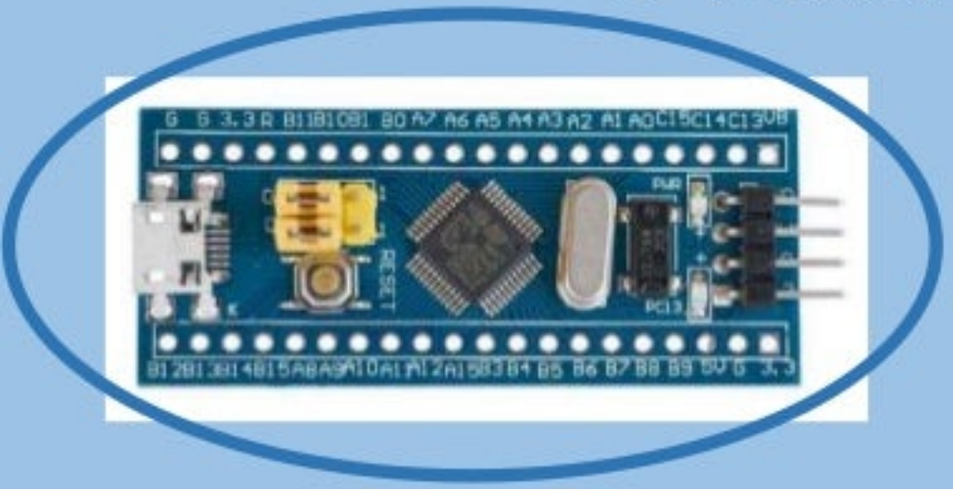
```
for(;;){
    led.set( true );
    wait();
    led.set( false );
    wait();
};
```

Let’s blink a LED!

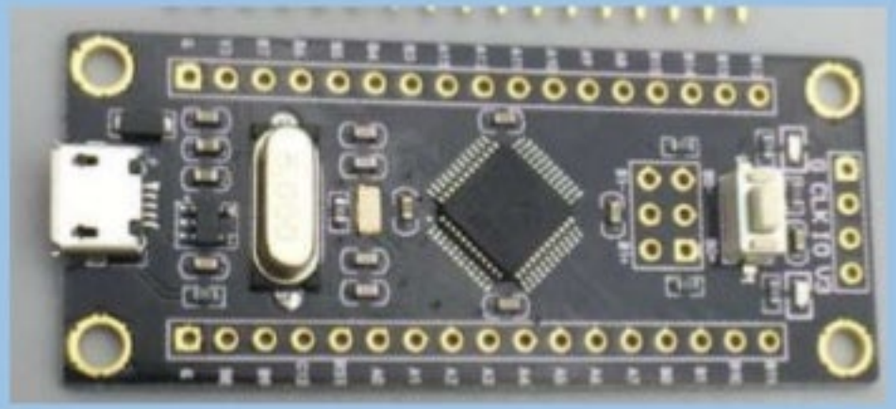


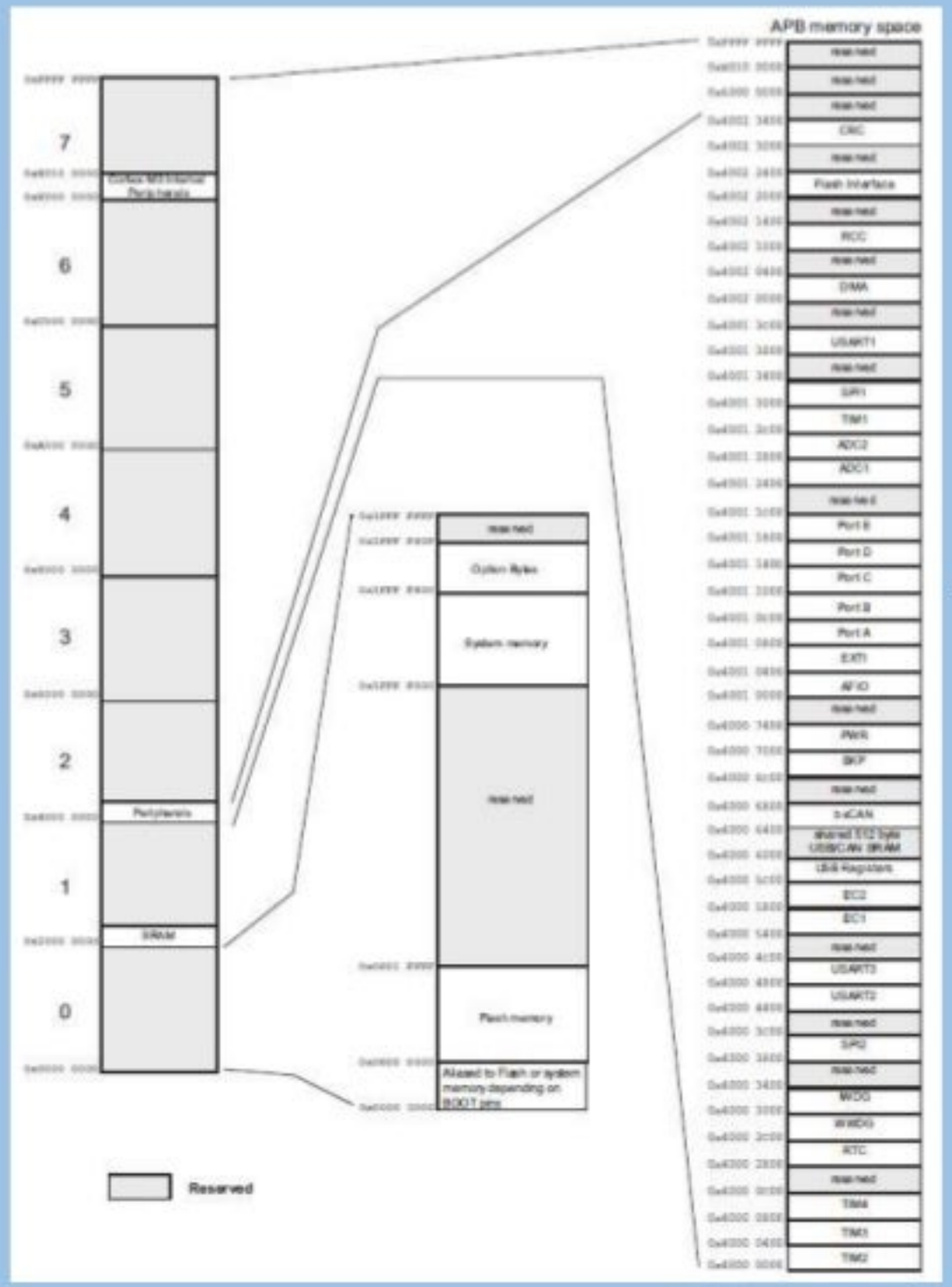
In this case, take the blue pill

STM32Duino ”Blue pill”



- STM32F103C8T6, Cortex-M3 32 bit CPU, 72 MHz
- 64 Kbyte FLASH (code & contants memory)
- 20 Kbyte RAM
- 32 I/O pins (chip has max 37)
- < E 2.00 (aliexpress)





Memory map

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	GPIOx_CRL	CNF7 [1:0]		MODE7 [1:0]		CNF6 [1:0]		MODE6 [1:0]		CNF5 [1:0]		MODE5 [1:0]		CNF4 [1:0]		MODE4 [1:0]		CNF3 [1:0]		MODE3 [1:0]		CNF2 [1:0]		MODE2 [1:0]		CNF1 [1:0]		MODE1 [1:0]		CNF0 [1:0]		MODE0 [1:0]	
	Reset value	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0
0x04	GPIOx_CRH	CNF15 [1:0]		MODE15 [1:0]		CNF14 [1:0]		MODE14 [1:0]		CNF13 [1:0]		MODE13 [1:0]		CNF12 [1:0]		MODE12 [1:0]		CNF11 [1:0]		MODE11 [1:0]		CNF10 [1:0]		MODE10 [1:0]		CNF9 [1:0]		MODE9 [1:0]		CNF8 [1:0]		MODE8 [1:0]	
	Reset value	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0
0x08	GPIOx_IDR	Reserved																IDRy															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	GPIOx_ODR	Reserved																ODRy															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	GPIOx_BSRR	BR[15:0]																BSR[15:0]															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x14	GPIOx_BRR	Reserved																BR[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x18	GPIOx_LCKR	Reserved															LCKK	LCK[15:0]															
	Reset value																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

GPIO registers

How to make a pin high or low

9.2.5 Port bit set/reset register (GPIOx_BSRR) (x=A..G)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:16 **BRy:** Port x Reset *bit y (y= 0 .. 15)*

These bits are write-only and can be accessed in Word mode only.

0: No action on the corresponding ODRx bit

1: Reset the corresponding ODRx bit

Note: If both BSx and BRx are set, BSx has priority.

Bits 15:0 **BSy:** Port x *Set bit y (y= 0 .. 15)*

These bits are write-only and can be accessed in Word mode only.

0: No action on the corresponding ODRx bit

1: Set the corresponding ODRx bit

In C - directly

```
// blink the on-board LED at PORTC, pin 13

#include "stm32f103x6.h"

void wait();

int main( void ){

   // enable the PORTC peripheral clock
   RCC->APB2ENR |= RCC_APB2ENR_IOPCEN;

   // make the pin an output
   GPIOC->CRH &= ~( 0x0F << 20 ); // CRH if pin > 7;
   GPIOC->CRH |=  ( 0x03 << 20 ); // 20 = 4 * ( 13 % 8 )

   for(;;){
       GPIOC->BSRR |= ( 0x01 << 13 ); // 1 << 13 for reset
       wait();
       GPIOC->BSRR |= ( 0x01 << 29 ); // 1 << ( 13 + 16 ) for set
       wait();
   };

}
```

9.2.5 Port bit set/reset register (GPIOx_BSRR) (x=A..G)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:16 **BRy:** Port x Reset *bit y (y= 0 .. 15)*
These bits are write-only and can be accessed in Word mode only.
0: No action on the corresponding ODRx bit
1: Reset the corresponding ODRx bit
Note: If both BSx and BRx are set, BSx has priority

Bits 15:0 **BSy:** Port x *Set bit y (y= 0 .. 15)*
These bits are write-only and can be accessed in Word mode only.
0: No action on the corresponding ODRx bit
1: Set the corresponding ODRx bit

In C - directly

```
for(;;){
    GPIOC->BSRR = ( 0x01 << 13 );
    wait();
    GPIOC->BSRR = ( 0x01 << 29 );
    wait();
};
```

Instructions	Direct
Call	1*
Library	-

```
        mov     r6, #8192
        mov     r5, #536870912
        ldr     r4, .L3+4
        . . .
.L2:
        str     r6, [r4, #16]
        bl      _Z4waitv
        str     r5, [r4, #16]
        bl      _Z4waitv
        b       .L2
        . . .
.L3:
        .word   1073876992
        .word   1073811456
```

gcc 6.2.0, -mcpu=cortex-m3 -Os -std=c11

http://www.github.com/wovo/2017-berlin

In C – use a macro

```
// set the pin port.pin to value
#define PIN_SET( port, pin, value ) . . .
```

Abstraction of an I/O pin

- **+ Efficient**
- **- It is a macro! (scope, textual parameters, …)**
- **- Can’t easily be replaced or augmented by the user**
- **- not easy to separate *what* from *how***

In C - functions

```
// blink the on-board LED at PORTC, pin 13
#include "stm32f103x6.h"

void wait();
void pin_set( int, bool );

int main( void ){

   RCC->APB2ENR |= RCC_APB2ENR_IOPCEN;

   GPIOC->CRH &= ~( 0x0F << 20 ); // CRH if pin > 7
   GPIOC->CRH |=  ( 0x03 << 20 ); // 20 = 4 * ( 13 % 8 )

   for(;;){
       pin_set( 77, true );
       wait();
       pin_set( 77, false );
       wait();
   };
}
```

```
#include "stm32f103x6.h"

GPIO_TypeDef * port_registers[] = {
   GPIOA, GPIOB, GPIOC, GPIOD };

// pin == 32 * port + pin
void pin_set( int pin, bool value ){
   port_registers[ pin / 32 ]->BSRR =
      0x01 << ( ( pin % 16 ) + ( value ? 0 : 16 ) );
}
```

In C - functions

9.2.5 Port bit set/reset register (GPIOx_BSRR) (x=A..G)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:16 **BRy:** Port x Reset *bit y (y= 0 .. 15)*
These bits are write-only and can be accessed in Word mode only.
0: No action on the corresponding ODRx bit
1: Reset the corresponding ODRx bit
Note: If both BSx and BRx are set, BSx has priority.

Bits 15:0 **BSy:** Port x *Set bit y (y= 0 .. 15)*
These bits are write-only and can be accessed in Word mode only.
0: No action on the corresponding ODRx bit
1: Set the corresponding ODRx bit

```
#include "stm32f103x6.h"

GPIO_TypeDef * port_registers[] = {
   GPIOA, GPIOB, GPIOC, GPIOD };

// pin == 32 * port + pin
void pin_set( int pin, bool value ){
   port_registers[ pin / 32 ]->BSRR =
      0x01 << ( ( pin % 16 ) + ( value ? 0 : 16 ) );
}
```

In C - functions

```
for(;;){
    pin_set( 77, true );
    wait();
    pin_set( 77, false );
    wait();
};
```

```
void pin_set( int pin, bool value ){
   port_registers[ pin / 32 ]->BSRR =
      0x01 << ( ( pin % 16 ) + ( value ? 0 : 16 ) );
}
```

```
.L2:
        movs    r1, #1
        movs    r0, #77
        bl      _Z7pin_setib
        bl      _Z4waitv
        movs    r1, #0
        movs    r0, #77
        bl      _Z7pin_setib
        bl      _Z4waitv
        b       .L2
```

```
_Z7pin_setib:
        movs    r3, #32
        sdiv    r3, r0, r3
        ldr     r2, .L4
        ldr     r2, [r2, r3, lsl #2]
        rsbs    r3, r0, #0
        and     r3, r3, #15
        and     r0, r0, #15
        it      pl
        rsbpl   r0, r3, #0
        cmp     r1, #0
        ite     ne
        movne   r3, #0
        moveq   r3, #16
        add     r3, r3, r0
        movs    r0, #1
        lsls    r0, r0, r3
        str     r0, [r2, #16]
        bx      lr
```

Instructions	Direct	Function
Call	1*	3
Library	-	18

In C++ - objects

```
. . .
#include "pin.hpp"

int main( void ){
    . . .

    auto led = pin( 2, 13 );

    for(;;){
        led.set( true );
        wait();
        led.set( false );
        wait();
    };

}
```

```
// pin.hpp
class pin {
private:
    int port, num;
public:
    pin( int port, int num );
    void set( bool value );
};
```

```
// pin.cpp
#include "stm32f103x6.h"
#include "pin.hpp"

GPIO_TypeDef * port_registers[] = {
    GPIOA, GPIOB, GPIOC, GPIOD };

pin::pin( int port, int num ):
    port( port ), num( num ){}

void pin::set( bool value ){
    port_registers[ port ]->BSRR =
        0x01 << ( ( num % 16 ) + ( value ? 0 : 16 ) );
}
```

In C++ - objects

```
. . .
#include "pin.hpp"

int main( void ){
    . . .

    auto led = pin( 2, 13 );

    for(;;){
        led.set( true );
        wait();
        led.set( false );
        wait();
    };

}
```

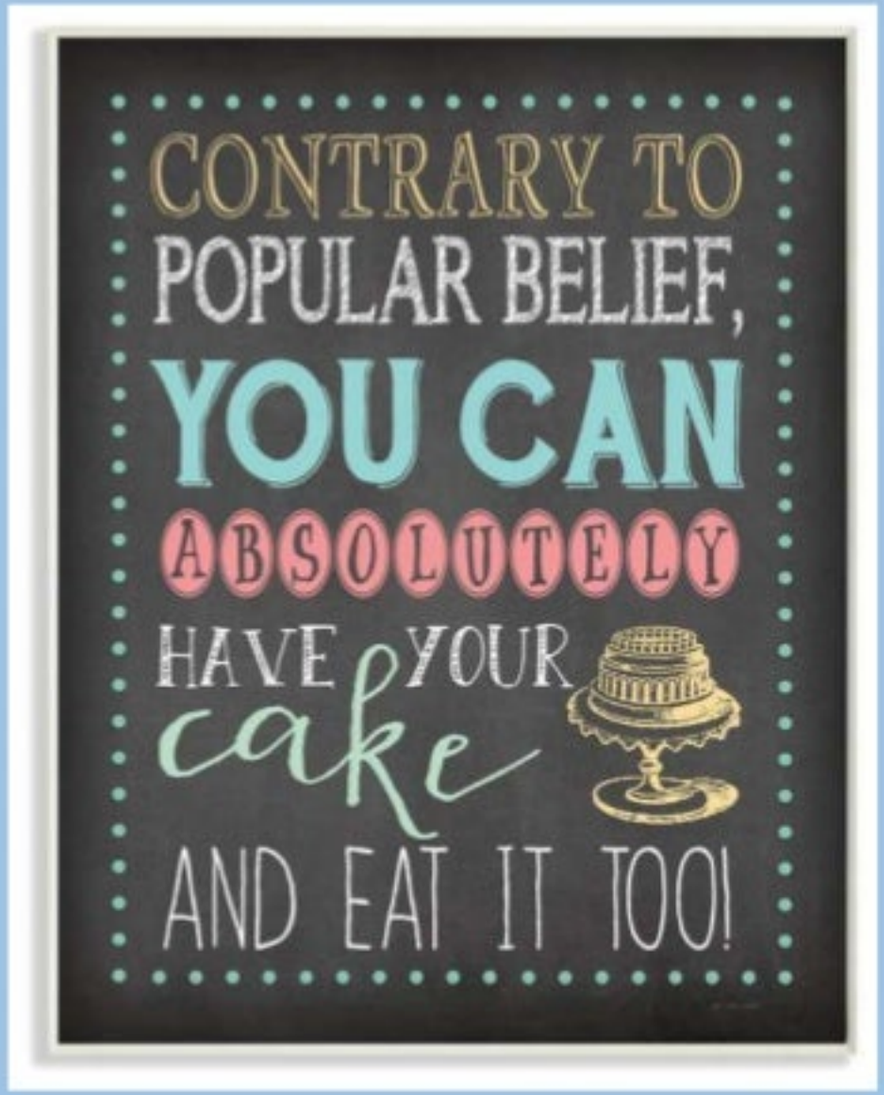
```
        movs    r2, #13
        movs    r1, #2
        mov     r0, sp
        bl      _ZN3pinC1Eii
.L2:
        mov     r0, sp
        movs    r1, #1
        bl      _ZN3pin3setEb
        bl      _Z4waitv
        movs    r1, #0
        mov     r0, sp
        bl      _ZN3pin3setEb
        bl      _Z4waitv
        b       .L2
```

In C++ - objects

```
void pin::set( bool value ){
   port_registers[ port ]->BSRR =
      0x01 << ( ( num % 16 ) + ( value ? 0 : 16 ) );
}
```

Instructions	Direct	Function	OO
Call	1*	3	3*
Library	-	18	19

```
_ZN3pin3setEb:
        ldr     r2, [r0]
        ldr     r3, .L5
        push    {r4, lr}
        ldr     r4, [r3, r2, lsl #2]
        ldr     r3, [r0, #4]
        rsbs    r2, r3, #0
        and     r2, r2, #15
        and     r3, r3, #15
        it      pl
        rsbpl   r3, r2, #0
        cmp     r1, #0
        ite     ne
        movne   r2, #0
        moveq   r2, #16
        add     r2, r2, r3
        movs    r3, #1
        lsls    r3, r3, r2
        str     r3, [r4, #16]
        pop     {r4, pc}
```



…… Well, in some situations.

In C++ - class templates

```
. . .
#include "pin.hpp"

int main( void ){
    . . .

    using led = gpio< 2, 13 >;

    for(;;){
        led::set( true );
        wait();
        led::set( false );
        wait();
    };

}
```

```
// pin.hpp
#include "stm32f103x6.h"

GPIO_TypeDef * port_registers[] = {
    GPIOA, GPIOB, GPIOC, GPIOD };

template< int port, int pin >
class gpio {
public:
    static void set( bool value ){
        port_registers[ port ]->BSRR =
            0x01 << ( ( pin % 16 ) + ( value ? 0 : 16 ) );
    }
};
```

In C++ - class templates

```
. . .
#include "pin.hpp"

int main( void ){
    . . .

    using led = gpio< 2, 13 >;

    for(;;){
        led::set( true );
        wait();
        led::set( false );
        wait();
    };

}
```

```
        mov     r6, #8192
        mov     r5, #536870912
        ldr     r4, .L3+4
        . . .
.L2:
        ldr     r3, [r4, #8]
        str     r6, [r3, #16]
        bl      _Z4waitv
        ldr     r3, [r4, #8]
        str     r5, [r3, #16]
        bl      _Z4waitv
        b       .L2
        . . .
.L3:
        .word   1073876992
        .word   .LANCHOR0
        .word   1073811456
```

Instructions	Direct	Function	OO	Template
Call	1*	3	3*	2*
Library	-	18	19	-

compile-time polymorphism

- Static classes (or structs) can be used as compile-time encapsulation of data and operations.
- Template parameters can be used to pass such an encapsulation to another one.
- These are compile-time-only actions, hence fully visible to the optimizer.

Run-time polymorphism

```
struct pin_out {
   virtual void set( bool ) = 0;
};


struct lpc1114_gpio : public pin_out {
   lpc1114_gpio( . . . ) . . .
   void set( bool x ) override { . . . }
};

struct blink {
   pin_out & pin;
   blink( pin_out & pin ): pin( pin ){}
   void run(){
      . . .
      pin.set( 1 );
   }
}

int main(){
   blink( lpc1114_gpio( 1, 2 )).run();
}
```

Compile-time polymorphism

```
struct pin_out_archetype {
   typedef void has_pin_out;
   static void set( bool value ){}
}

template< int port, int pin >
struct lpc1114_gpio : public pin_out_archetype {
   static void set( bool value ){ . . . }
};

template< typename pin >
struct blink {
   void run(){
      . . .
      pin::set( 1 );
   }
}


int main(){
   blink< lpc1114_gpio< 1, 2 >>::run();
}
```

As before, lots of ugly details (initialization, timing, …) are omitted.

Four pin archetypes

```
struct pin_in_archetype {
   typedef void has_pin_in;
   static void init();
   static bool get();
};
```

```
struct pin_out_archetype {
   typedef void has_pin_out;
   static void init();
   static void set( bool value );
};
```

```
struct pin_in_out_archetype {
   typedef void has_pin_in_out;
   static void init();
   static void direction_set_input();
   static void direction_set_output();
   static bool get();
   static void set( bool value );
};
```

```
struct pin_oc_archetype {
   typedef void has_pin_oc;
   static void init();
   static bool get();
   static void set( bool value );
};
```


Blink: some (ugly?) details

```
#include "targets/lpc1114fn28.hpp"
typedef hwcpp::lpc1114fn28<> target;

int main(){
   hwcpp::blink< target::gpio_1_2, target::waiting >::run();
}
```

```
template<
   typename arg_pin,
   typename arg_timing
> struct blinking {

   typedef pin_out_from< arg_pin > pin;
   typedef waiting_from< arg_timing > timing;

   typedef typename timing::duration duration;

   static void run( const duration t = duration::ms( 500 ) ){
      timing::init();
      pin::init();
      for(;;){
         pin::set( 1 );
         timing::wait( t / 2 );
         pin::set( 0 );
         timing::wait( t / 2 );
      }
   }
};
```

Type narrowing

Explicit initialization

pin_out_from< . . . > template

```
template< class unsupported, class dummy = void > struct pin_out_from {
   static_assert( sizeof( unsupported ) == 0, . . . ); };

template< class pin > struct pin_out_from < pin, typename pin::has_pin_out > :
   public pin_out_archetype
{
   static void init(){ pin::init(); }
   static void set( bool value ){ pin::set( value ); }
};

template< class pin > struct pin_out_from < pin, typename pin::has_pin_oc > :
   public pin_out_archetype
{
   static void init(){ pin::init(); }
   static void set( bool value ){ pin::set( value ); }
};

template< class pin > struct pin_out_from < pin, typename pin::has_pin_in_out > :
   public pin_out_archetype
{
   static void init(){ pin::init(); pin::direction_set_output(); }
   static void set( bool value ){ pin::set( value ); }
};
```

Type narrowing

```
class target {
   typedef . . . pin_a0;
}

typedef target::pin_a0 alarm;

int main(){
   alarm::init();
   alarm::direction_set_output();
   alarm::set( false );

   if( . . . ){
      alarm::set( true );
   }
}
```

```
class target {
   typedef . . . pin_a0;
}

typedef pin_out_from< target::pin_a0 > alarm;

int main(){
   alarm::init();
   alarm::direction_set_output();
   alarm::set( false );

   if( . . . ){
      alarm::set( true );
   }
}
```

- ❑ The template parameter is type-checked
- ❑ The template adapts to the parameter type
- ❑ Initialization is ‘complete’
- ❑ The code can’t accidentally use an inappropriate method

invert< . . . >

Don’t say it in a comment

decorator

```
namespace target {
   typedef . . . pin_a0;
}

typedef target::pin_a0 alarm_led;

   // the alarm LED pin is active low
const bool alarm_led_active = false;

int main(){
   alarm_led::set( ! alarm_led_active );
   if( . . . ){
      alarm_led::set( alarm_led_active );
   }
}
```

Say it in the code

```
namespace target {
   typedef . . . pin_a0;
}

typedef invert< target::pin_a0 > alarm_led;

int main(){
   alarm_led::set( false );
   if( . . . ){
      alarm_led::set( true );
   }
}
```

Pin invert< . . . >

```
template< class unsupported, class dummy = void > struct invert {
    static_assert( sizeof( unsupported ) == 0, ". . ." ); };

template< class pin > struct invert< pin, typename pin::has_pin_in > :
  public pin_in_archetype {
    static void init(){ pin::init(); }
    static bool get(){ return ! pin::get(); } };

template< class pin > struct invert< pin, typename pin::has_pin_out > :
  public pin_out_archetype {
    static void init(){ pin::init(); }
    static void set( bool x ){ pin::set( ! x ); } };

template< class pin > struct invert< pin, typename pin::has_pin_in_out > :
  public pin_in_out_archetype {
    static void init(){ pin::init(); }
    static void direction_set_input(){ pin::direction_set_input(); }
    static void direction_set_output(){ pin::direction_set_output(); }
    static void set( bool x ){ pin::set( ! x );      }
    static bool get(){ return ! pin::get(); } };

template< class pin > struct invert< pin, typename pin::has_pin_oc > :
  public pin_oc_archetype {
    static void init(){ pin::init(); }
    static void set( bool x ){ pin::set( ! x ); }
    static bool get(){ return ! pin::get(); } };
```

Often:
ZERO cost!

Some more pin & port decorators

```
// create the stated pin or port type from p
pin_in_from< p >
pin_out_from< p >
pin_in_out_from< p >
pin_oc_from< p >
port_in_from< p >
port_out_from< p >
port_in_out_from< p >
port_oc_from< p >

// create a port of the stated type from pins
port_in_from_pins< p, ... >
port_out_from_pins< p, ... >
port_in_out_from_pins< p, ... >

// invert the value read from or written to p
invert< p >

// pin or port that writes to all pins or ports
all< p, ... >
```


Using static class templates: consequences

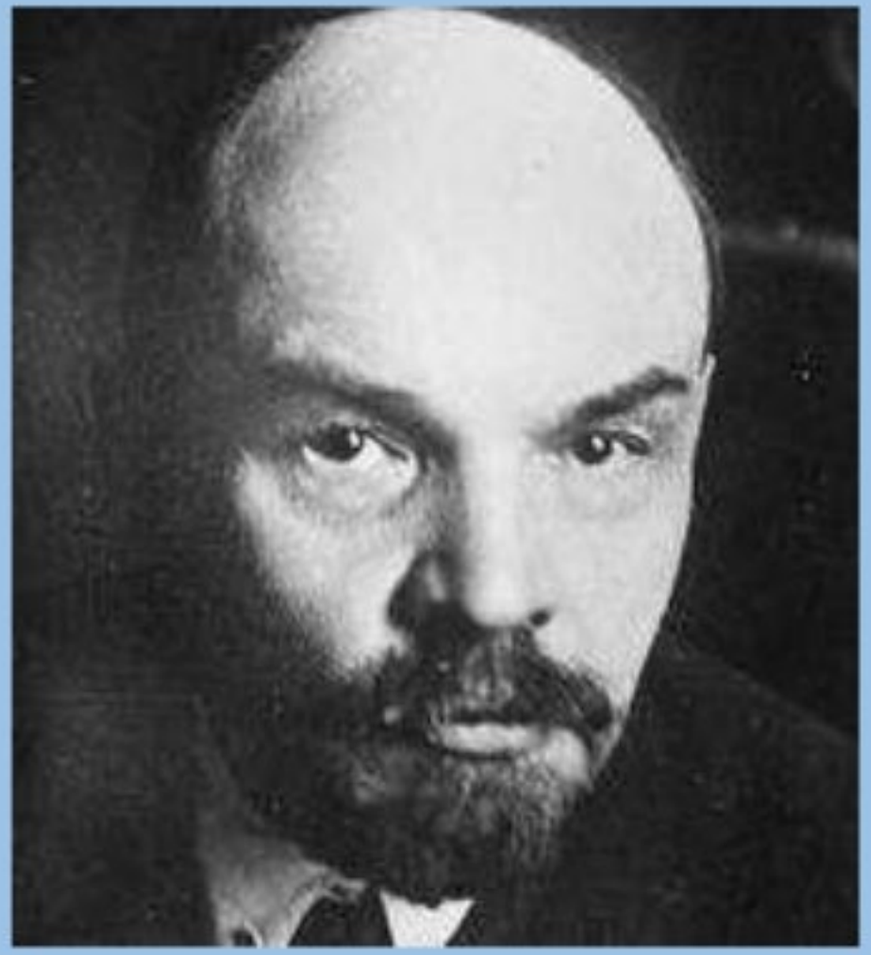
- Abstraction tool is the class template with only static contents (classes, methods, attributes)
- Composition is by template instantiation
- Method call syntax is class::method(…) instead of object.method(…)
- Header-only

Using static class templates: more consequences

- ❑ No objects (of such classes) are created
- ❑ No memory management, no dangling references
- ❑ Collections don’t contain unused elements
- ❑ Sharing of identical constructs is automatic
- ❑ No automatic construction: explicit init() calls
- ❑ Abstraction overhead can be zero

What does a programming language want?

“Give me just one generation of youth, and I'll transform the whole world.”

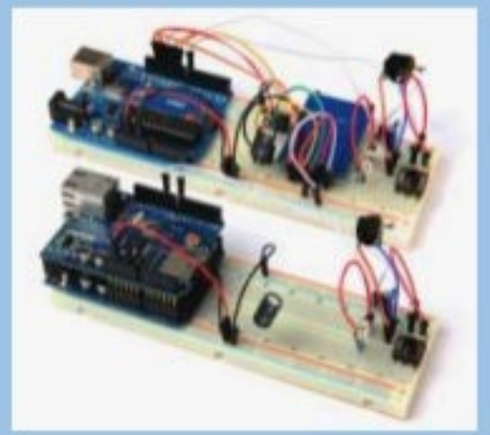


What motivates the next generation of programmers?

- Websites : (that isn't real programming)
- Mobile : mostly Android, Java, Java-frameworks
- Easy power : RaspberryPi, Python
- (Micro Bit, Lego) : graphic programming
- Cheap, low-energy, hardware interfacing : Arduino 

The Arduino ecosystem

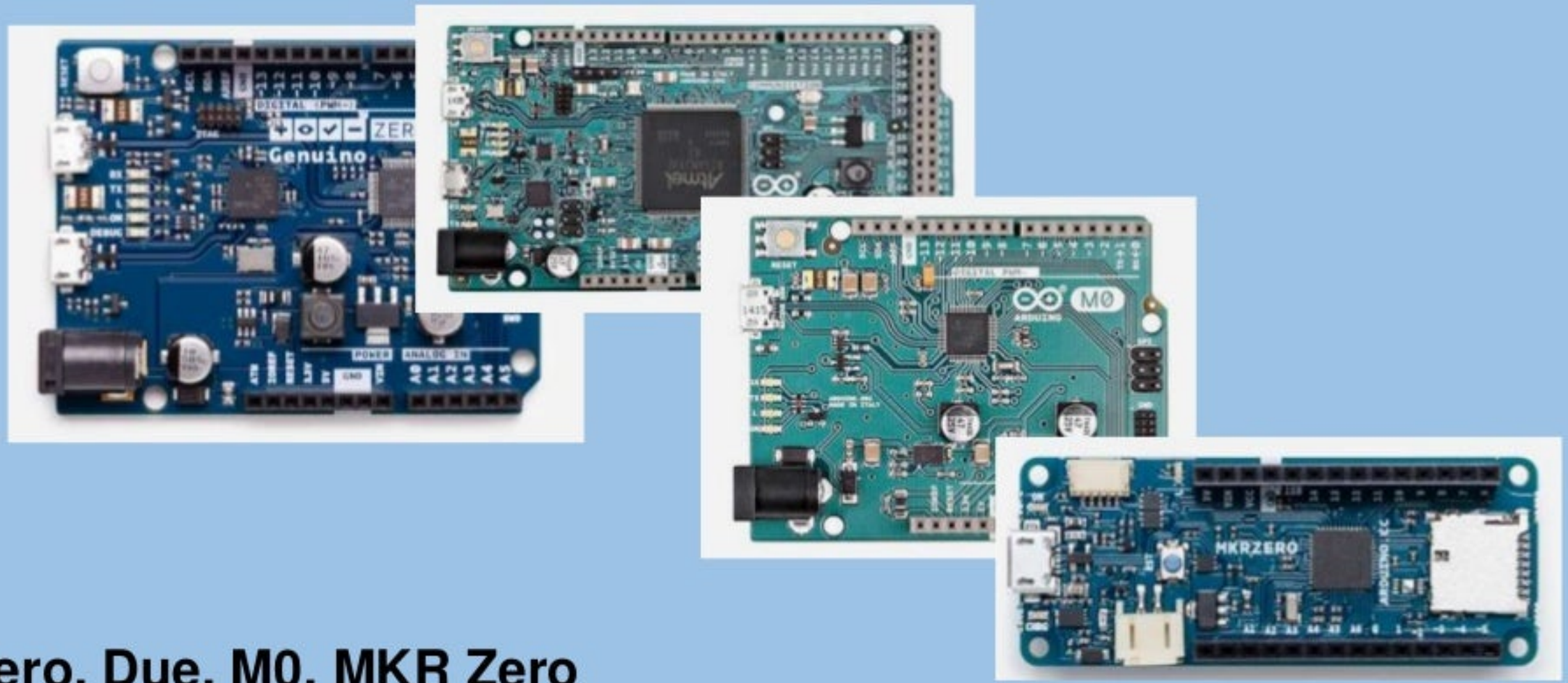
- Hardware designs & webshop
- Bootloaders
- IDE with toolchain, libraries, download tools
- Third-party libraries
- ‘Counterfeit’ hardware
- Web coverage



Official hardware designs - AVR

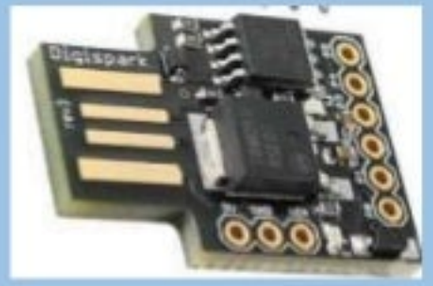
Uno, Leonardo, 101, esplora, micro, nano, mini, mega, ethrente, lilypad

Official hardware designs - Cortex



Zero, Due, M0, MKR Zero

Some other boards



Blue Pill, ATTiny85, ESP8266, ESP32

Arduino software

- Crappy IDE, but it works quite reliably
- Some behind-the-scenes magic
- Uses C++ compiler, but almost no C++ features
- Lots of overlapping third-party libraries
- Configuration is often by editing the library source
  (each LCD driver has its own graphics library)
- And yet: tremendously popular!

- Lots of black magic & cargo-cult beliefs
- Lots of duplicate efforts (overlapping libraries)
- Lack of leverage (no component-based culture)

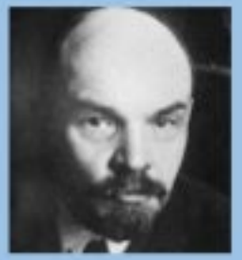
Opportunities!

A well-structured set of C++ libraries, books, websites, clubs, etc. could make these folks love C++ for their life!

Target audience?

- Kids
- Professional C++ programmers in their spare time
- non-EE/CE professionals (artists!)

Grab the next generation



[&kids]{

and don’t let them escape!

}

Takaways

- Embedded? Please be more specific!
- The C++ is *the* language for (small-) embedded
- but ’they’ don’t know that yet ☹
- and we don’t yet agree on *how to use it*
- C++ libraries could be more embedded-friendly