std::map<Code,Performance> myMCU{?}

World Map (1459)

World Map (1525)

People admitted they don't know.

The Beginning of Modern Science

1. Admit ignorance

ignorance | ˈɪɡn(ə)r(ə)ns |

noun *[mass noun]*

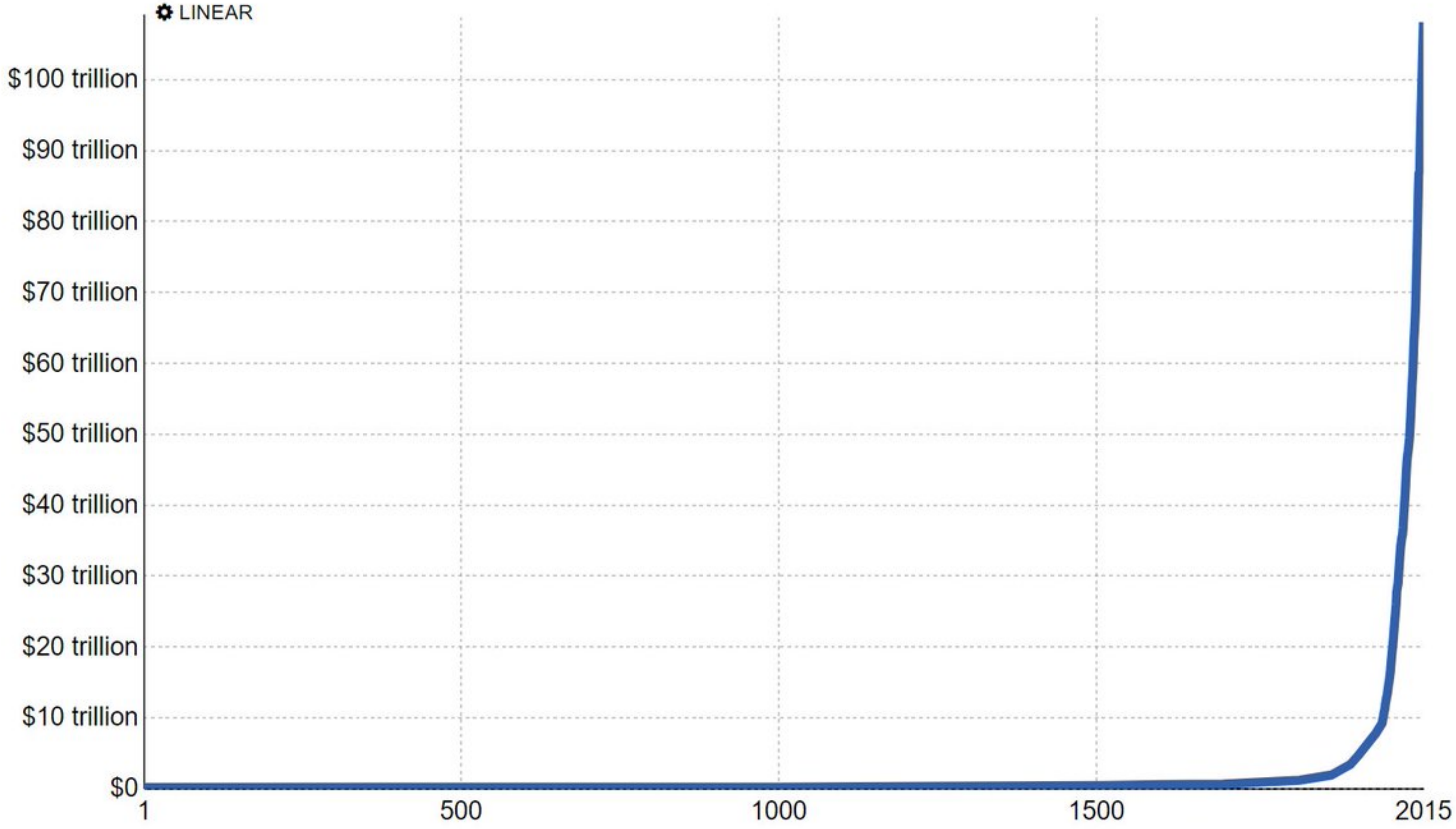
lack of knowledge or information: *he acted in* ***ignorance of*** *basic procedures*.

2. Observations

- Measure and gather data.
- Connect data into comprehensive theories.

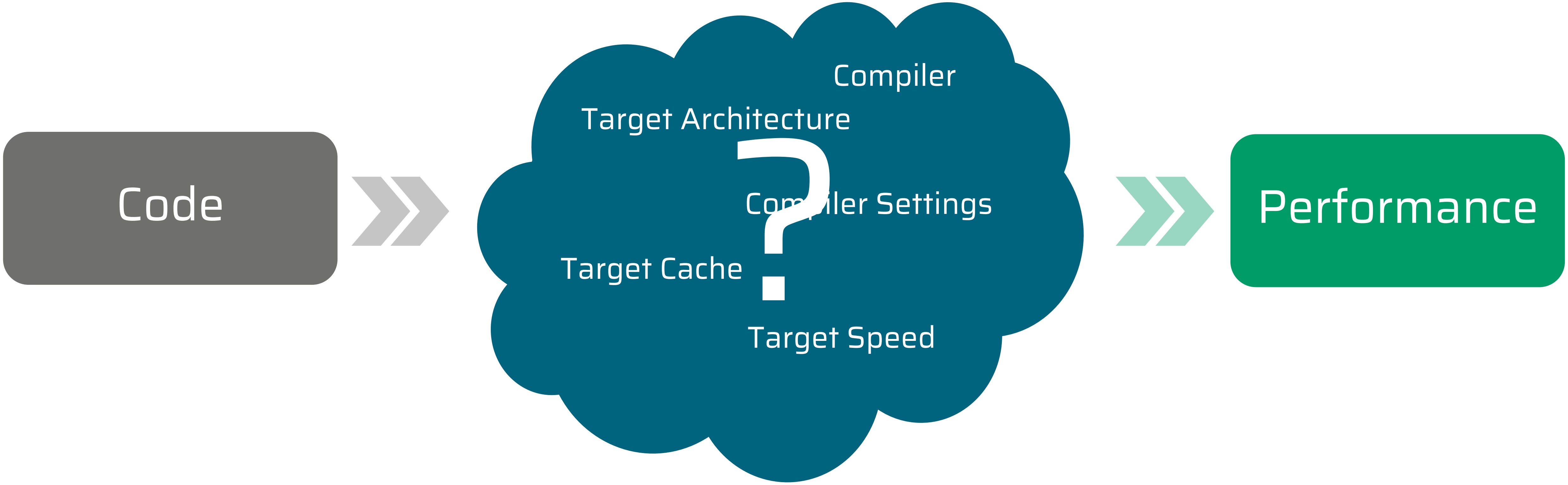
World GDP over the last two millennia

Total output of the world economy; adjusted for inflation and expressed in 2011 international dollars.



Source: World GDP (Our World In Data based on World Bank & Maddison)

Embedded & Ignorance



Possibly a highly complex and interdependent mapping!

Consequences

Prejudices prevail

Mistrust against libraries

Low code quality

Performance suffers

Let’s admit our ignorance.

Observations in Embedded

Profiling

- Top Down Process.
- Great to identify bottlenecks.
- Bad to create specific understanding.

Build knowledge bottom up

- Start with small code blocks.
- Observe performance.
- Create heuristics.

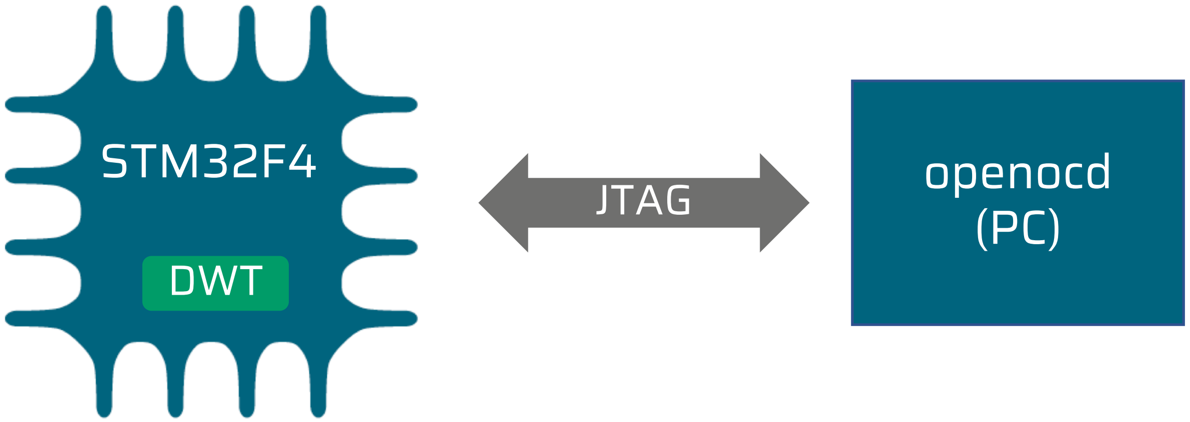
Code Performance for armv7m

Architecture widely used (Cortex-M3/M4)

Provides **D**ata **W**atchpoint and **T**race Unit

CMSIS Register	**Description**
DWT_CYCCNT	Cycle Count Register
DWT_CPICNT	CPI Count Register
DWT_EXCCNT	Exception Overhead Count Register
DWT_SLEEPCNT	Sleep Count Register
DWT_LSUCNT	LSU Count Register
DWT_FOLDCNT	Folded-instruction Count Register

Measure Cycles

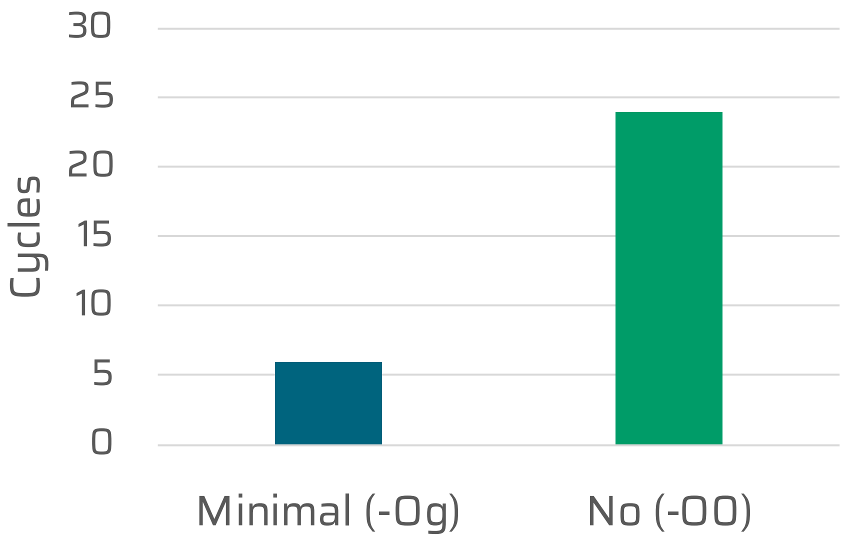



```
BKPT //< Read CYCCNT
CodeUnderTest(<Parameter>)
BKPT //< Read CYCCNT
```

Let’s make observations.

Example 1: Basic Optimization

```c
int square(int x) {
    return x*x;
}
```



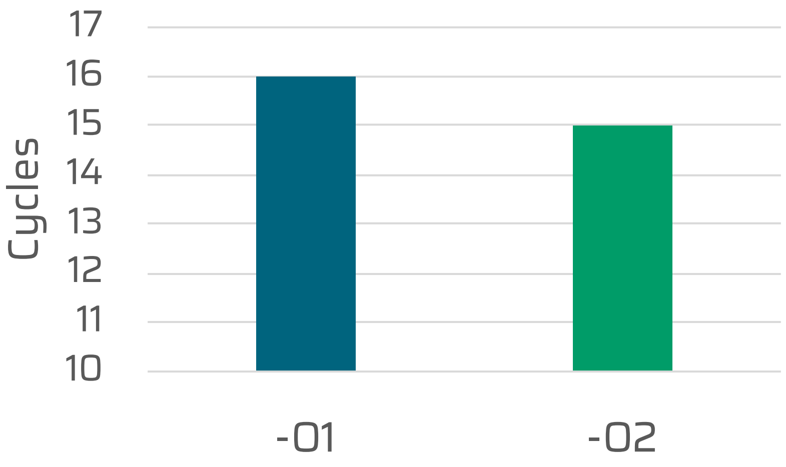

```
square(int):
        mul     r0, r0, r0
        bx      lr
```

```
square(int):
        push    {r7}
        sub     sp, sp, #12
        add     r7, sp, #0
        str     r0, [r7, #4]
        ldr     r3, [r7, #4]
        ldr     r2, [r7, #4]
        mul     r3, r2, r3
        mov     r0, r3
        adds    r7, r7, #12
        mov     sp, r7
        ldr     r7, [sp], #4
        bx      lr
```

Heuristic #1
The difference between minimal and no optimization is huge.

Example 2: Pipeline

```
int DependentOps(int x) {
    int tmp = x/3;
    int tmp2 = x/7;
    return tmp+tmp2;
}
```




```
DependentOps_O1(int):
    ldr     r3, .L2
    smull   r2, r3, r3, r0
    asrs    r1, r0, #31
    subs    r3, r3, r1
    ldr     r2, .L2+4
    smull   ip, r2, r2, r0
    add     r0, r0, r2
    rsb     r0, r1, r0, asr #2
    add     r0, r0, r3
    bx      lr
.L2:
    .word   1431655766
    .word   -1840700269
```

```
DependentOps_O2(int):
    ldr     r3, .L3
    ldr     r1, .L3+4
    smull   r2, r3, r3, r0
    add     r3, r3, r0
    asrs    r2, r0, #31
    smull   r1, r0, r1, r0
    rsb     r3, r2, r3, asr #2
    subs    r0, r0, r2
    add     r0, r0, r3
    bx      lr
.L3:
    .word   -1840700269
    .word   1431655766
```

Heuristic #2
In low-level assembly, the compiler is probably smarter than you.

Example 3: FPU vs Soft-FPU

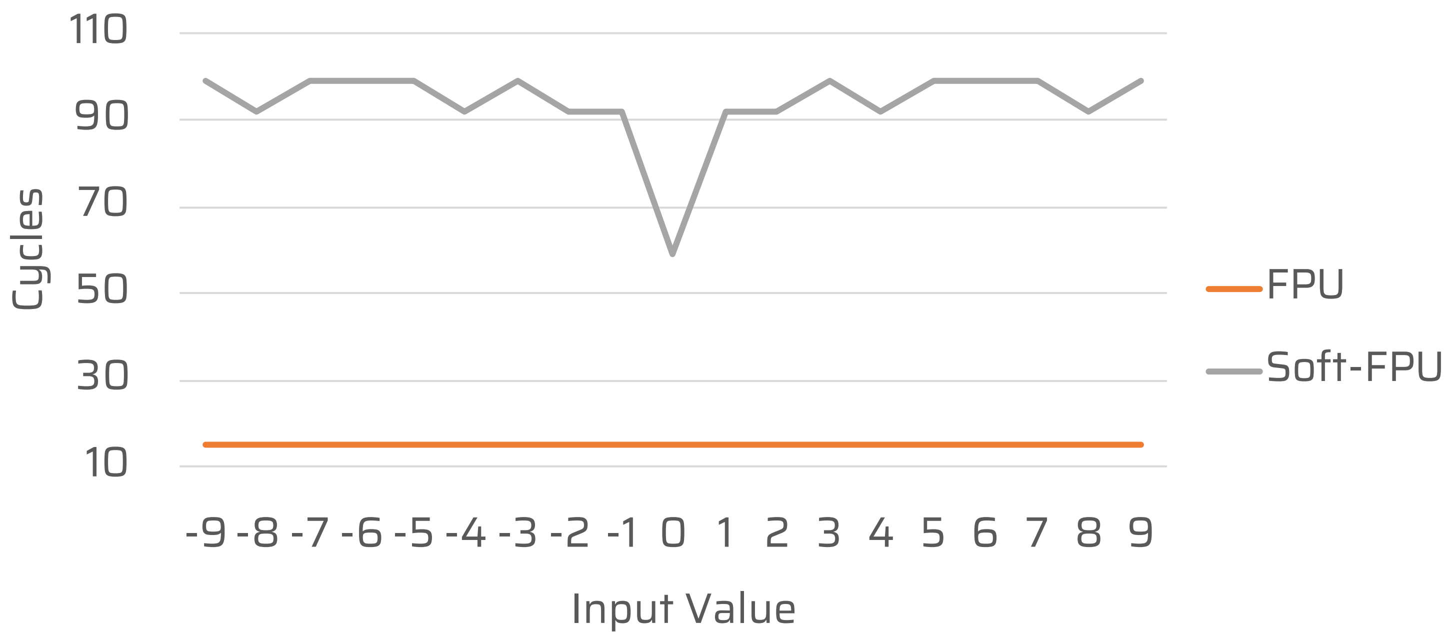
```c
int MultiplyWithPi(int input) {
    return input * 3.14159265359f;
}
```

```
MultiplyWithPi_FPU(int):
  vmov    s15, r0 @ int
  vldr.32 s14, .L3
  vcvt.f32.s32    s15, s15
  vmul.f32        s15, s15, s14
  vcvt.s32.f32    s15, s15
  vmov    r0, s15 @ int
  bx      lr
.L3:
  .word   1078530011
```

```
MultiplyWithPi_SoftFPU(int):
  push    {r3, lr}
  bl      __aeabi_i2f
  ldr     r1, .L4
  bl      __aeabi_fmul
  bl      __aeabi_f2iz
  pop     {r3, pc}
.L4:
  .word   1078530011
```

Example 3: FPU vs Soft-FPU

```c
int MultiplyWithPi(int input) {
    return input * 3.14159265359f;
}
```

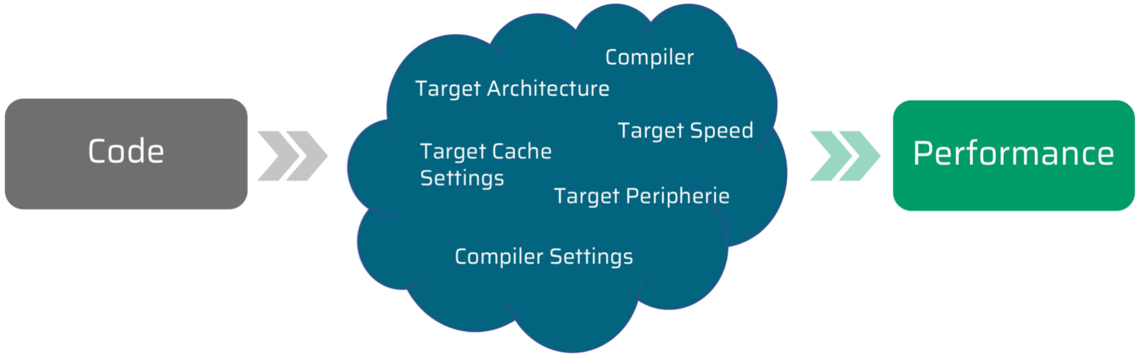


Heuristic # 3
Software-FPU ~ 6x slower and not deterministic.

Example 4: CRC Computation

Cyclic Redundancy Check

- Direct Computation
- Lookup-Table
- Hardware-Support



Online Benchmarking

- Execute on real hardware.
- Technical Preview Stage.
- https://barebench.com

barebench.com
- Demo -

Heuristic #4

Performance *may* be dependent on clock speed.

Heuristic #5
Caching is essential for high clock speeds.

Conclusion

Admit lack of knowledge.

Measure performance.

Use measurements to form heuristics.

Share heuristics.

Use heuristics instead of prejudices.

Let‘s make embedded systems better!