

# Compiling C to CLI for Heterogeneous Multicore SoCs



Erven Rohou  
Andrea C. Ornstein  
Marco Cornero

# Agenda

 Software and hardware of embedded systems

 Virtualization

 Rationale for C and CLI

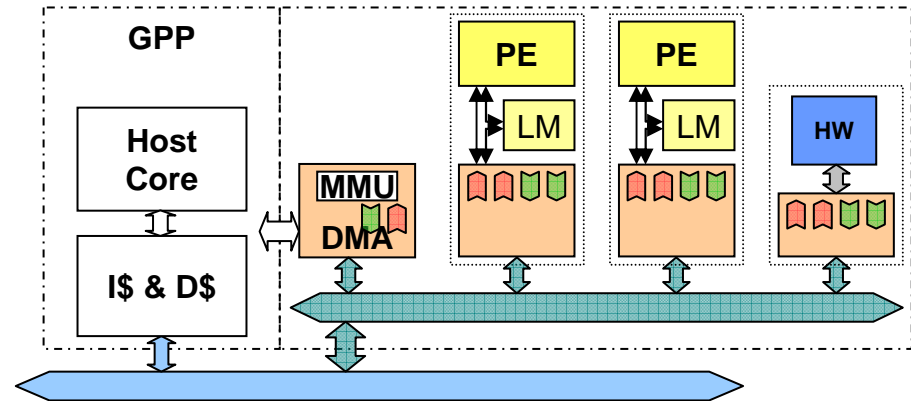
 Some issues

# Embedded Software

- ▣ Software content increasing 2x every 10 months
- ▣ Typical complex SoC software today:
  - ▣ several millions lines of code
  - ▣ 50-300 people involved
  - ▣ ASP < \$10!
- ▣ C language is *de facto* standard
  - ▣ performance
  - ▣ legacy

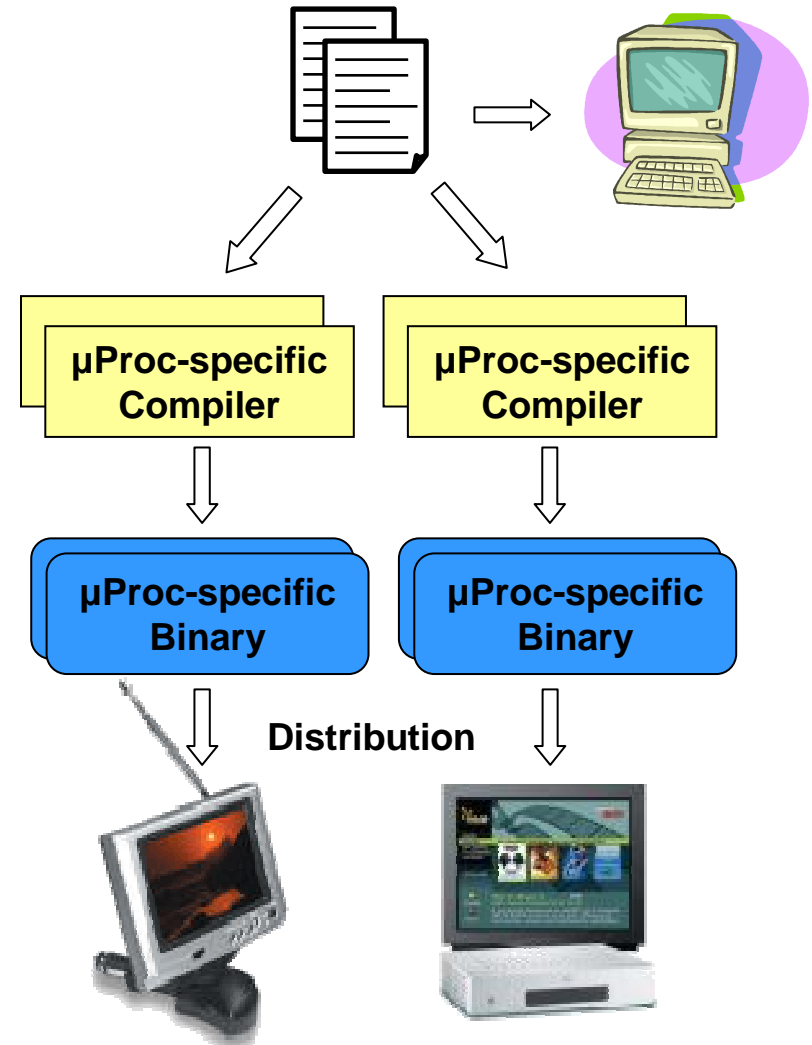
# Embedded Hardware

- Is heterogeneous
  - and will be so for a while...
- For many reasons
  - performance
  - cost
  - power consumption
- Becomes programmable
  - cost
  - time to market



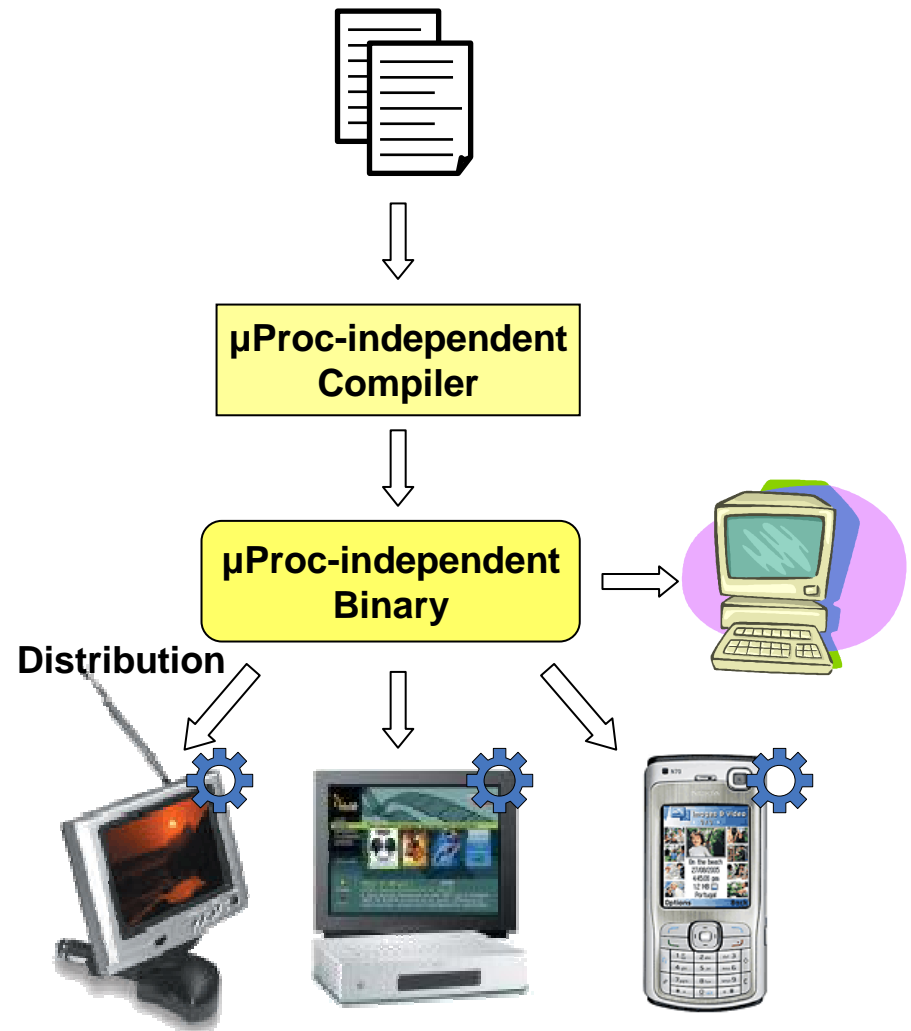
# Embedded SoCs are Difficult to Program

- Many compilers/toolchains
  - per product
  - per chip
  - per OS
  - likely different technologies (GCC, Cosy, Pro64, proprietary)
    - different flags, behaviors ...
  - maintenance/upgrade nightmare
- Code full of `#ifdef` and intrinsics
  - poor readability
  - what you run is NOT what you debugged
- ISV have no access to tools
  - cannot exploit media processor



# Proposal: Processor Virtualization in C

- Not a new concept (Java, .NET)
- C (C++)
- High performance
- Program the entire system
  - including DSP, media processors
- Postpone code generation to device
  - same format deployed to all systems
- One compiler/toolchain
  - easier maintenance
  - what you run IS what you debug
- Open platform: ISV can exploit the whole hardware
- Will run on future systems
  - reduce legacy
- Of course, extra technology needed in device



# What is CLI?

- ▣ Stands for *Common Language Infrastructure*
  - ▣ aka Microsoft .NET
- ▣ Standardized ECMA 335 and ISO/IEC 23271:2006
- ▣ Processor independent format
  - ▣ stack based, no registers
  - ▣ object oriented
  - ▣ garbage collection
  - ▣ and support for unmanaged code (pointer arithmetic,...)
- ▣ Common Type System
- ▣ Standard library
- ▣ Interface to native code
  - ▣ hand optimized library, proprietary code, ...

# Why C to CLI?


## Why C?

-  legacy

-  performance (low level C programming)


## Why CLI?

-  target independent

-  standard (unlike Java)

  -  well defined

  -  stable

-  can express managed but also unmanaged (i.e. C or C++ like) code (unlike the Java bytecode)

  -  can deliver performance (C)

  -  can improve productivity (C#)

-  can integrate with existing optimized native libraries

-  directly executable

  -  good for development, debugging

-  large community (including open source)

# Benefits

- ▣ Simplify toolchains mess
- ▣ 3<sup>rd</sup> party can easily program entire platform
  - ▣ no need for proprietary/internal tools
  - ▣ have access to DSP
    - ▣ the most powerful part of the system
- ▣ Reduce hardware legacy
  - ▣ manufacturer can change processor
- ▣ Run time advantages
  - ▣ load balancing
  - ▣ fault tolerance
  - ▣ code sharing
- ▣ Zero risk entry point with static compilation
  - ▣ no loss of performance, similar code size
  - ▣ opens the door to dynamic optimizations

## Benefits (cont'd)

- ▣ Dynamic Optimization
  - ▣ exploits run-time information
    - ▣ profile-driven optimizations
    - ▣ code specialization
  - ▣ potential to improve on static
  - ▣ embedded domain is friendly to dynamic optimizations
    - ▣ media processing runs for a very long time!
    - ▣ must be real-time upfront (e.g. load-time compilation), then optimize (e.g. for low power)
- ▣ Dynamic environment can accommodate for hardware reconfiguration
- ▣ Higher Level Languages
  - ▣ virtual machine for managed languages (Java, C#) for control code
    - ▣ much increased software productivity!
    - ▣ observability
  - ▣ smoothly integrated with unmanaged languages (C/C++) for high-performance

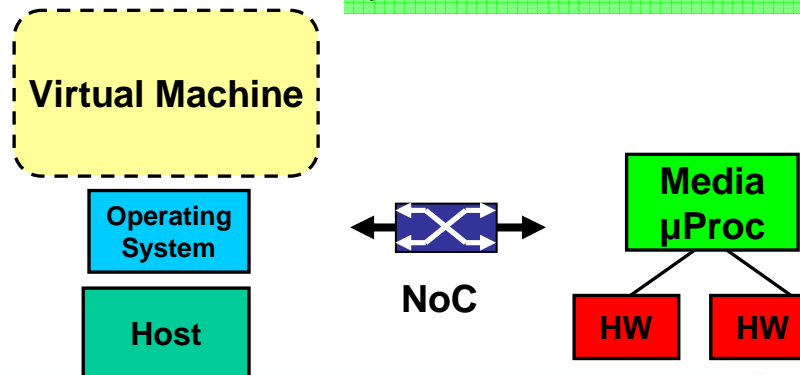
# Heterogeneous Platform Virtualization

**C#, Java, ...**  
(control/OS-dependent code)

```
open(stream);  
...  
if (a < b) {  
    bla bla;  
}  
  
filter(in, out, n);  
...  
update_GUI(new_skin)  
...  
close(stream);  
...
```

**Unmanaged C#, C, C++, ...**  
(efficient OS-independent code)

```
[ProcessorAttribute("Media",  
    "in:input.n",  
    "out:output.n")]  
void filter(short* input,  
            short* output,  
            int n) {  
    for (int i = 0; i < n; i++) {  
        (wild pointer fantasy ☺)  
    }  
}
```

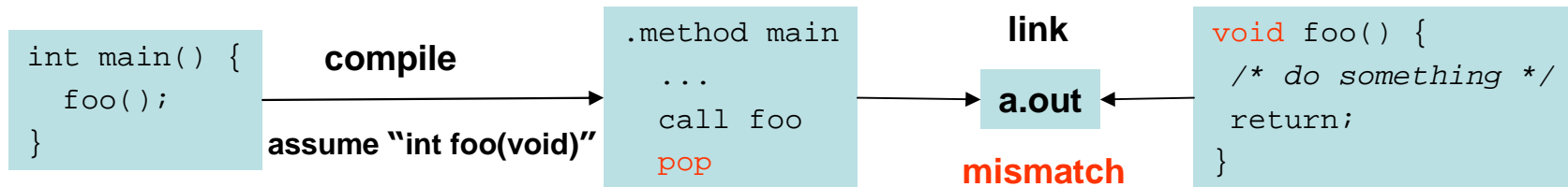


# Issues

- ❏ CLI stricter than C
  - ❏ prototypes
  - ❏ strongly typed
  - ❏ varargs
- ❏ Full portability is challenging
  - ❏ size of `native int` exposed in layout of aggregates
- ❏ Interaction with native code
- ❏ Interaction between native and CLI libraries

# CLI / C mismatches

## Missing function prototypes



mandatory in C99

CLI has no bitfields

CLI has no type qualifiers (`const`, `restrict`, `volatile`)

asm keyword: CIL or native instructions?

## Size of native int

- CLI defines the type `System.IntPtr` (or `i`)
  - size of a pointer on target machine
- C typically exposes the layout of aggregate types
  - layout done at compilation time in C
- Would require symbolic addressing
  - `foo[2]` becomes `*(@foo + 2*sizeof(node))`
- Resulting code would be
  - large
  - inefficient (missing optimizations, IV, folding, ...)

```
struct node {  
    void* data;  
    struct node* next;  
} foo[10];
```

# Interaction with Native Code

## Reasons to do it

- hand optimized code
- 3<sup>rd</sup> party code (no source available)

## *pinvoke* helps, but:

- layout of structures/unions
  - CLI layout vs native ABI
  - CLI wrapper around call
- ELF allows libraries to refer to global variables
  - what CLI name for x in library?
- global variable in library

```
extern void libfoo(void);
int x;
int main() {
    x = 2;
    libfoo();
}
```

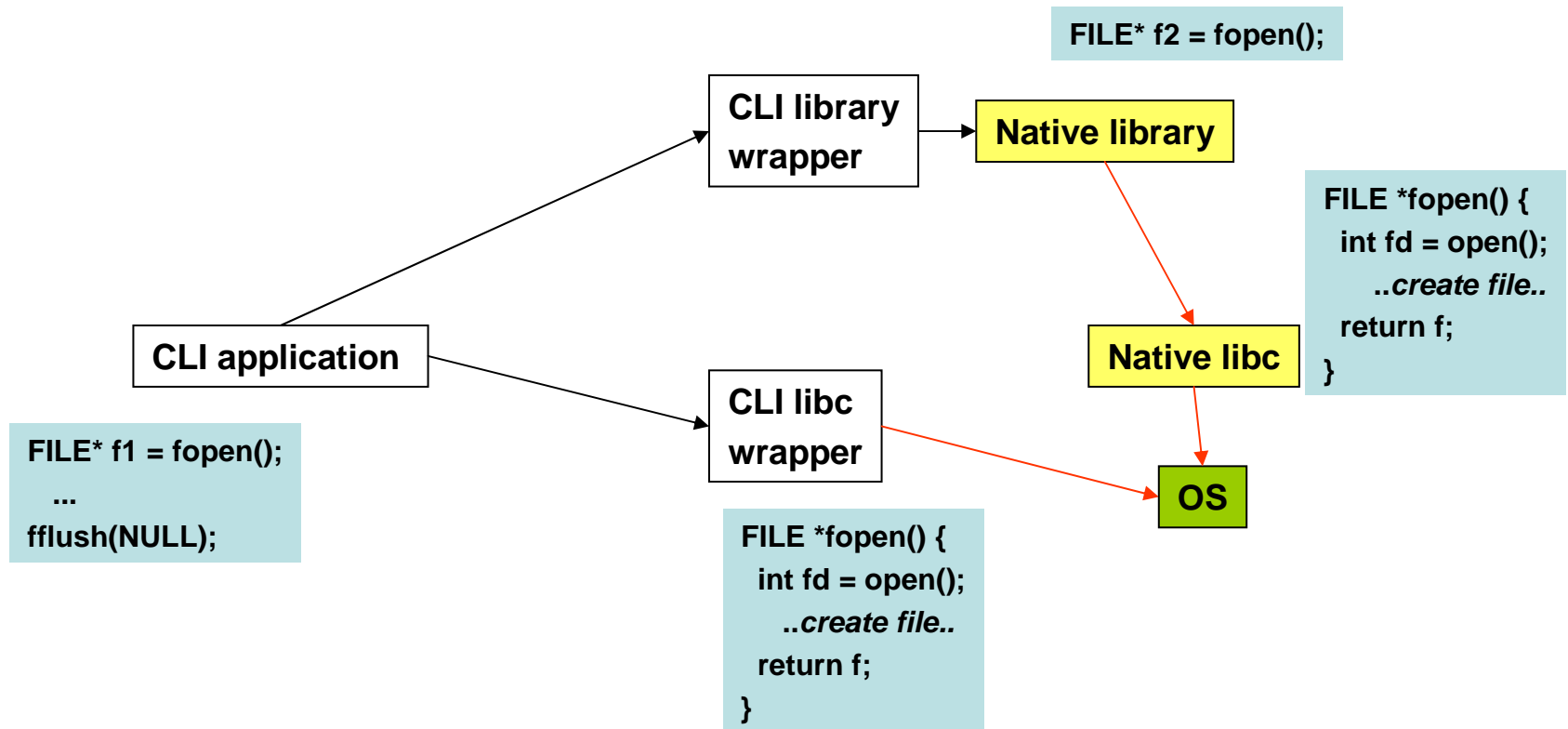
```
extern int x;
void libfoo() {
    use(x);
}
```

```
extern int x;
int main () {
    if (x)
        . . .
}
```

```
/* global state */
int x;
void foo() {
    x=17;
}
```

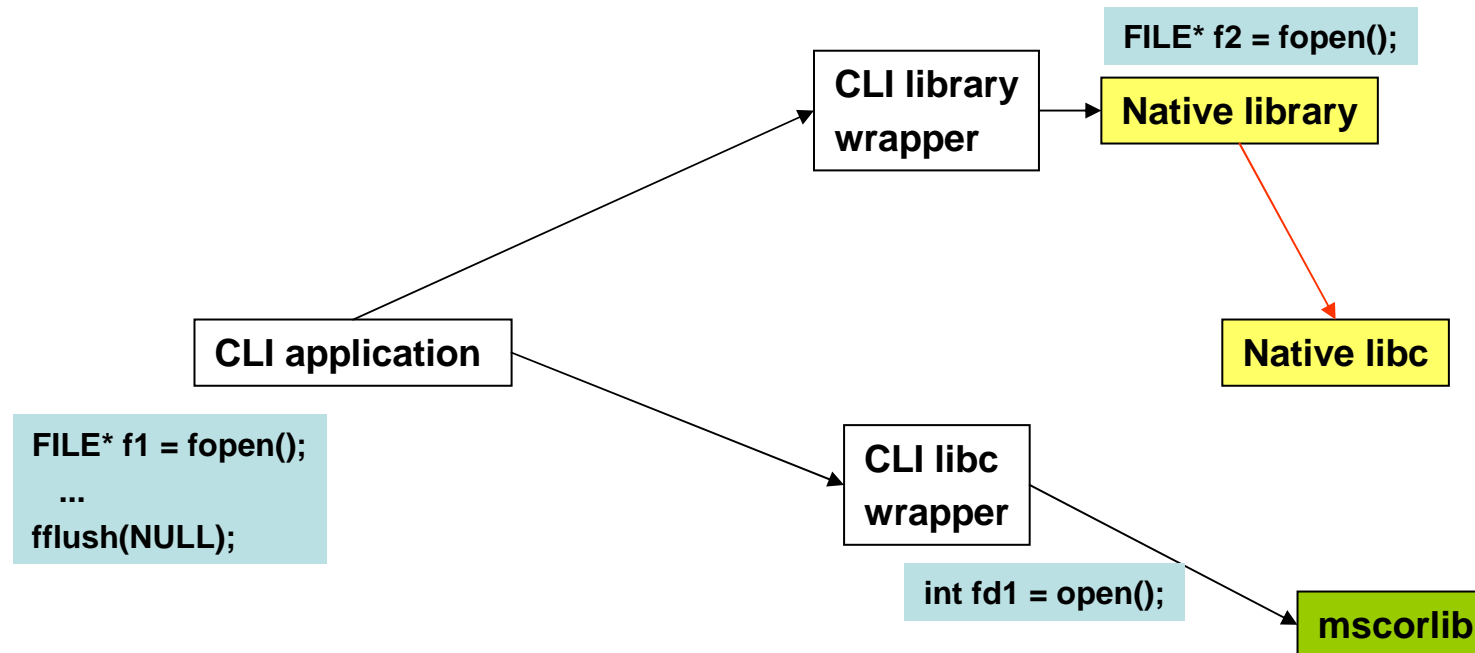
# Library Interactions – Native libc

Wrappers must be stateless



# Library Interactions - Portable libc

- ▣ libc written on top of mscorlib
- ▣ Native and CLI libc coexist
- ▣ Risk of inconsistency!





# Status

- ▣ C99, except
  - ▣ setjmp/longjmp
  - ▣ complex numbers (incomplete)
  - ▣ variable-sized arrays in structs
  - ▣ debug information
    - ▣ Line numbers
    - ▣ Information on local variables
- ▣ Binutils based on Mono project
- ▣ Missing GCC extensions
  - ▣ some cases of nested functions
  - ▣ attribute *packed*
  - ▣ ELF directives (section, weak, alias)
  - ▣ asm

# Thank you!

## Give it a try

 <http://gcc.gnu.org/projects/cli.html>

 `svn co svn://gcc.gnu.org/svn/gcc/branches/st gcc`

## In case you are interested, let us know

 [erven.rohou@st.com](mailto:erven.rohou@st.com)

 [andrea.ornstein@st.com](mailto:andrea.ornstein@st.com)

 [marco.cornero@st.com](mailto:marco.cornero@st.com)