Topic 1:
Introduction to Microprogrammed Embedded Systems

Systèmes Embarqués Microprogrammés

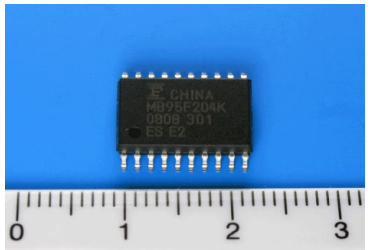
Prof. David Atienza Alonso, SEL-STI

Content of Session

- Microprogrammed Embedded Systems: what, why and when
  - Comparisons with traditional embedded systems
  - Main application fields and design constraints
  - System-level architectures: hardware and software
  - C-based cross-development frameworks
- Case study: Nintendo DS (NDS) Lite
  - Hardware architecture
  - Software design: C-based cross-compilation flow
  - Practical work 2: C programming on the Nintendo DS
  - Practical Work 3: Advanced C Programming on the Nintendo DS

What has changed in “embedded Systems”?

- Definition of (traditional) embedded system:
  - Computer system designed to perform one (or few) dedicated functions, often with limited computation complexity, but real-time computing constraints. It is part of a complete device typically including hardware and mechanical parts.

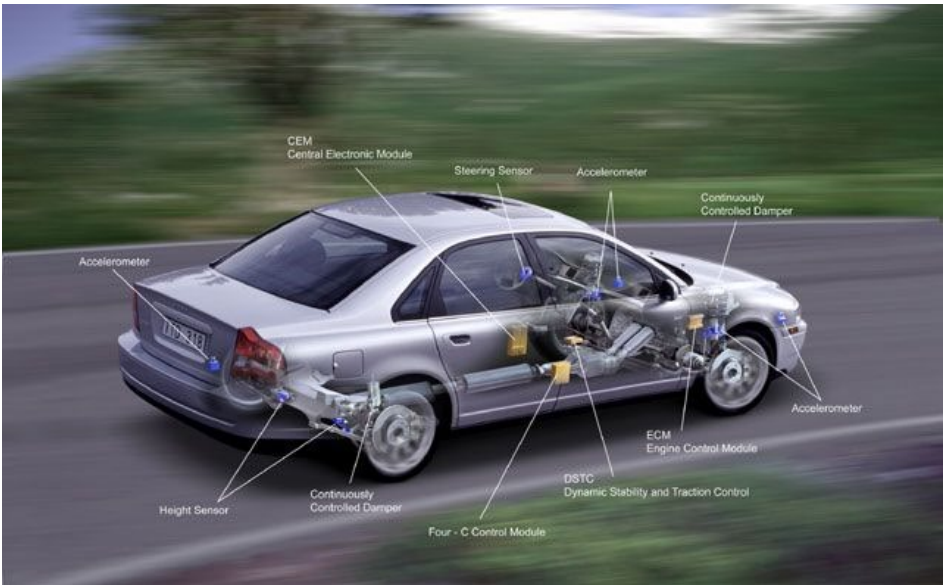


ASICs or simple microcontrollers (8- or 16-bit PICs)

- Required functionality features: ARMS
  - **Availability A(t)=** probability system working at time t
  - **Reliability R(t) =** probability system working correctly at time t
  - **Maintainability M(d) =** probability of system working correctly d time units after an error occurred.
  - **Safety =** no harm to be caused

- Required design constraints:
  - Small size, but low cost (large production volume)
  - Low power use, minimum set of HW possible

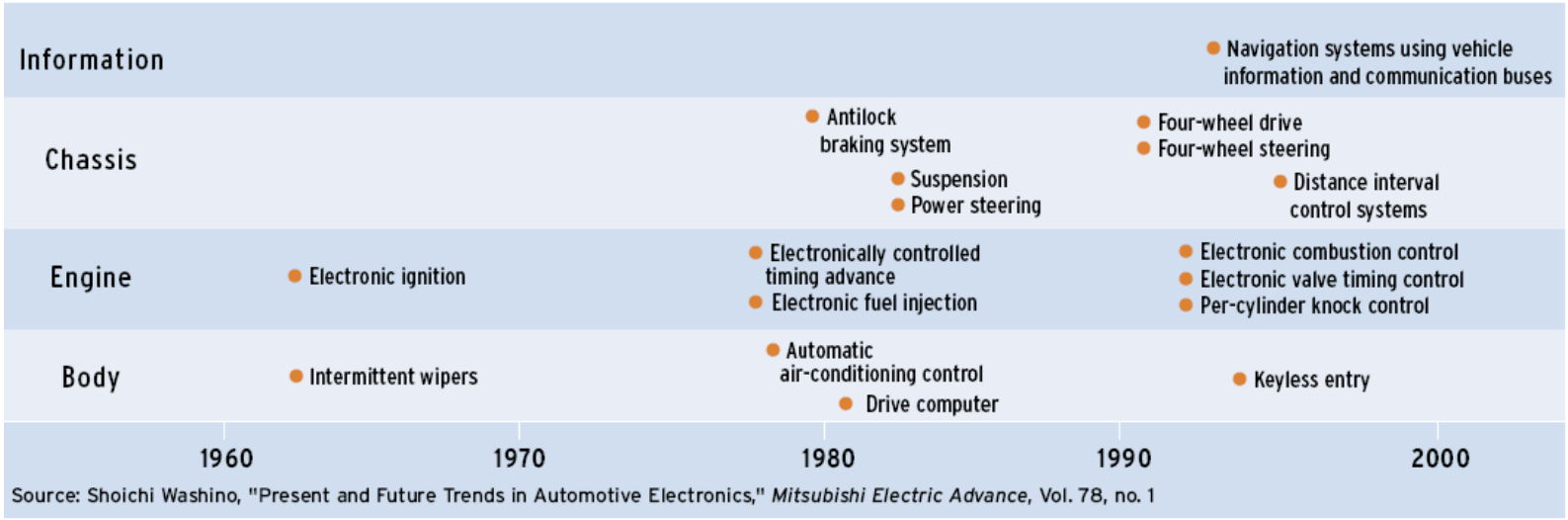
Key area for the evolution: increase of automotive electronics



What is “automotive electronics”?

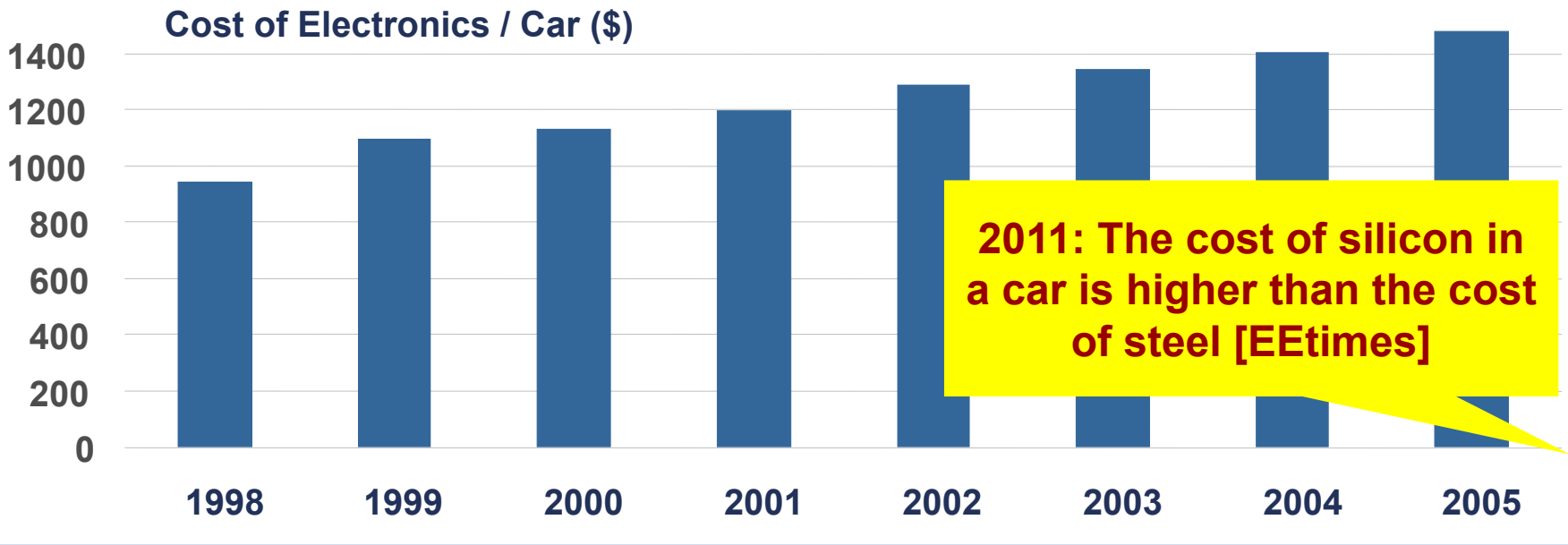
- Vehicle functions implemented with embedded systems
  - Body electronics
  - System electronics: chassis, engine
  - Information/entertainment

Automotive Electronics Through the Years



Source: Shoichi Washino, "Present and Future Trends in Automotive Electronics," *Mitsubishi Electric Advance*, Vol. 78, no. 1

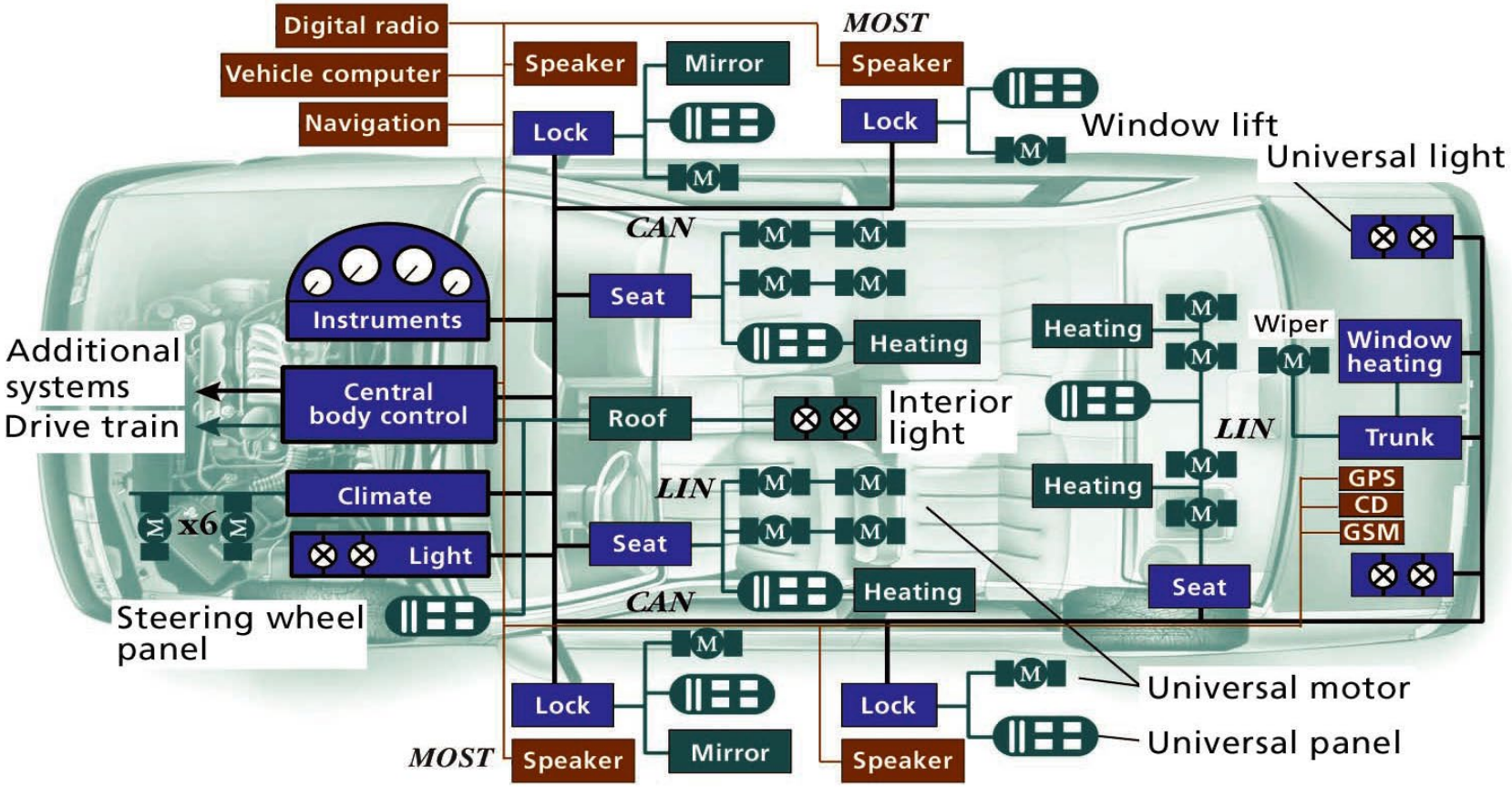
Automotive electronics market size



Market ($billions)	8.9	10.5	13.1	14.1	15.8	17.4	19.3	21.0

70% of future innovations in our vehicles will be only possible through improvements of the embedded electronic systems

Latest automotive electronics: complex set of microprogrammed embedded systems

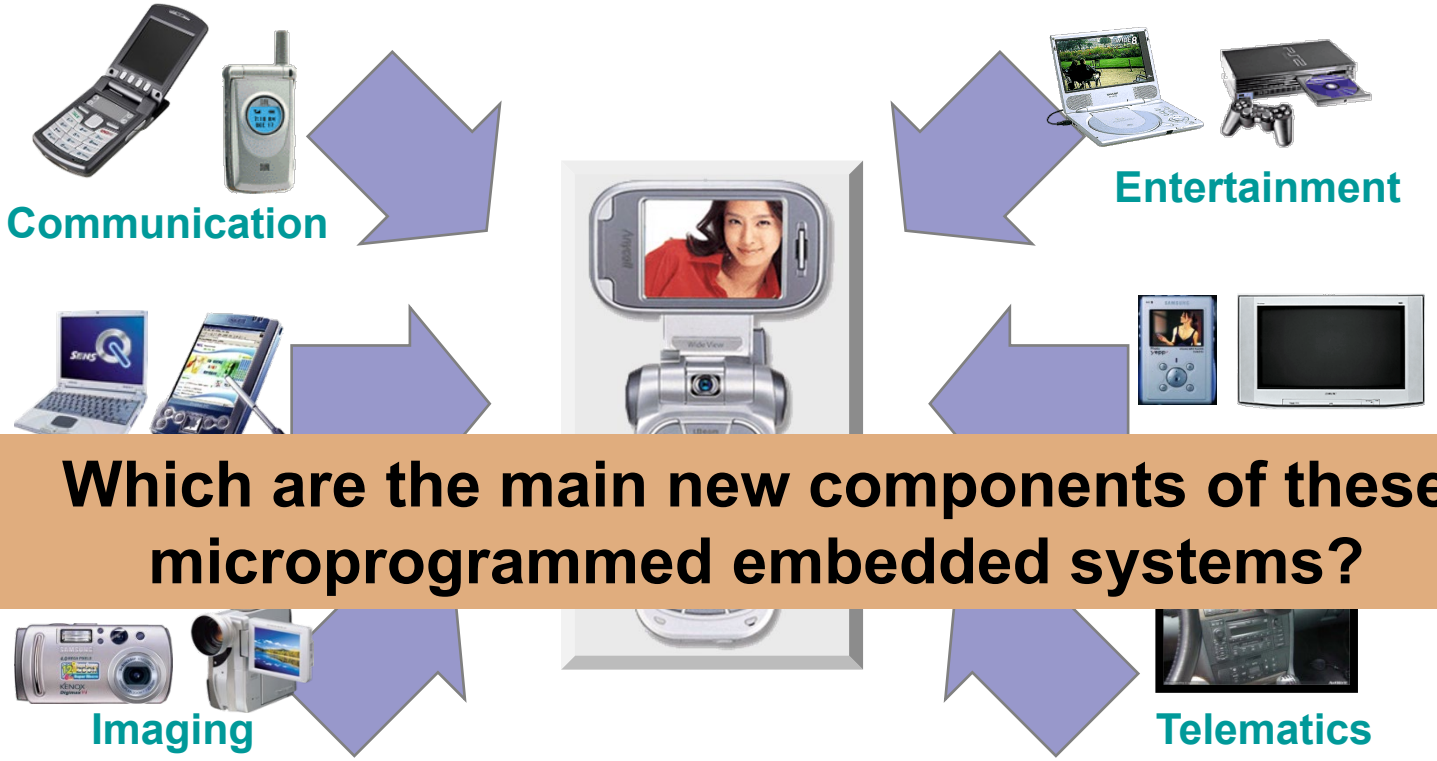


CAN Controller area network
GPS Global Positioning System
GSM Global System for Mobile Communications
LIN Local interconnect network
MOST Media-oriented systems transport

Latest BMWs contain more than 150 embedded microprocessors

Source: Expanding automotive electronic systems, IEEE Computer, Jan. 2010

Latest market: digital convergence, the “smart” mobile example

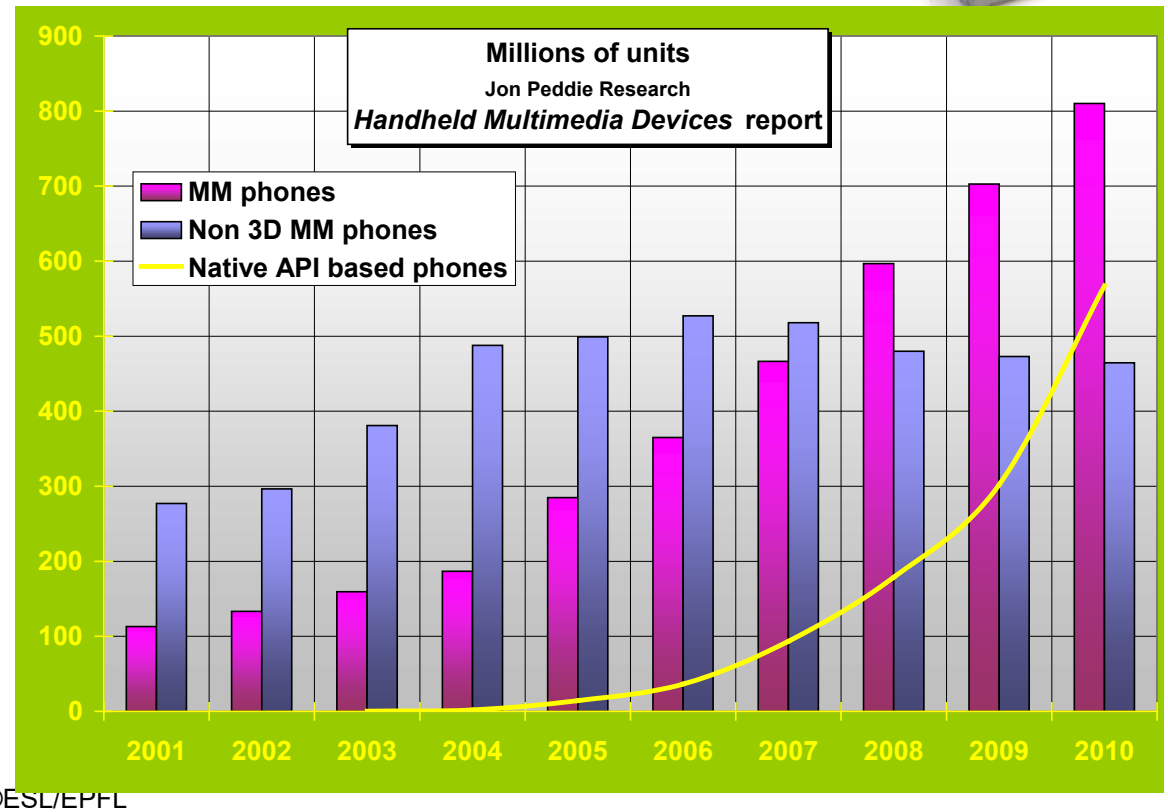


Which are the main new components of these microprogrammed embedded systems?

- One device, multiple functions
- Center of ubiquitous media and social networks
- Smart mobiles: next drive for semiconductor. Industry

1st new key component) **IMAGE** Mobile graphics enable games, videos...

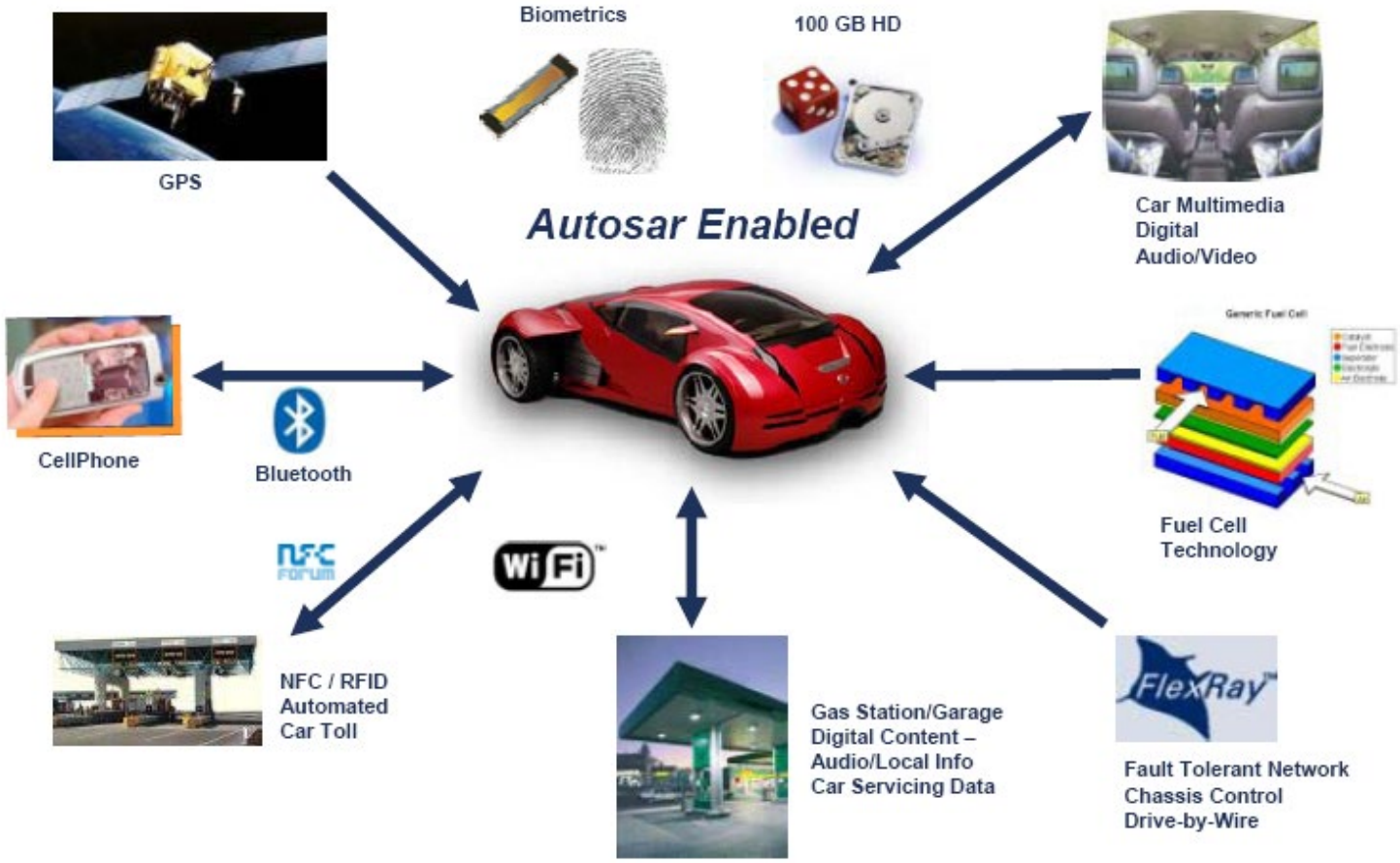
- Resolution started with ~176x208 (2001)
  - In Japan, QVGA (320x240) is the minimum now
  - Nokia series 90 is 640x320
  - iPhone4 is 960x640



Less than VGA first, but almost 1024x768 already here, this has made the market grow

New applications and services

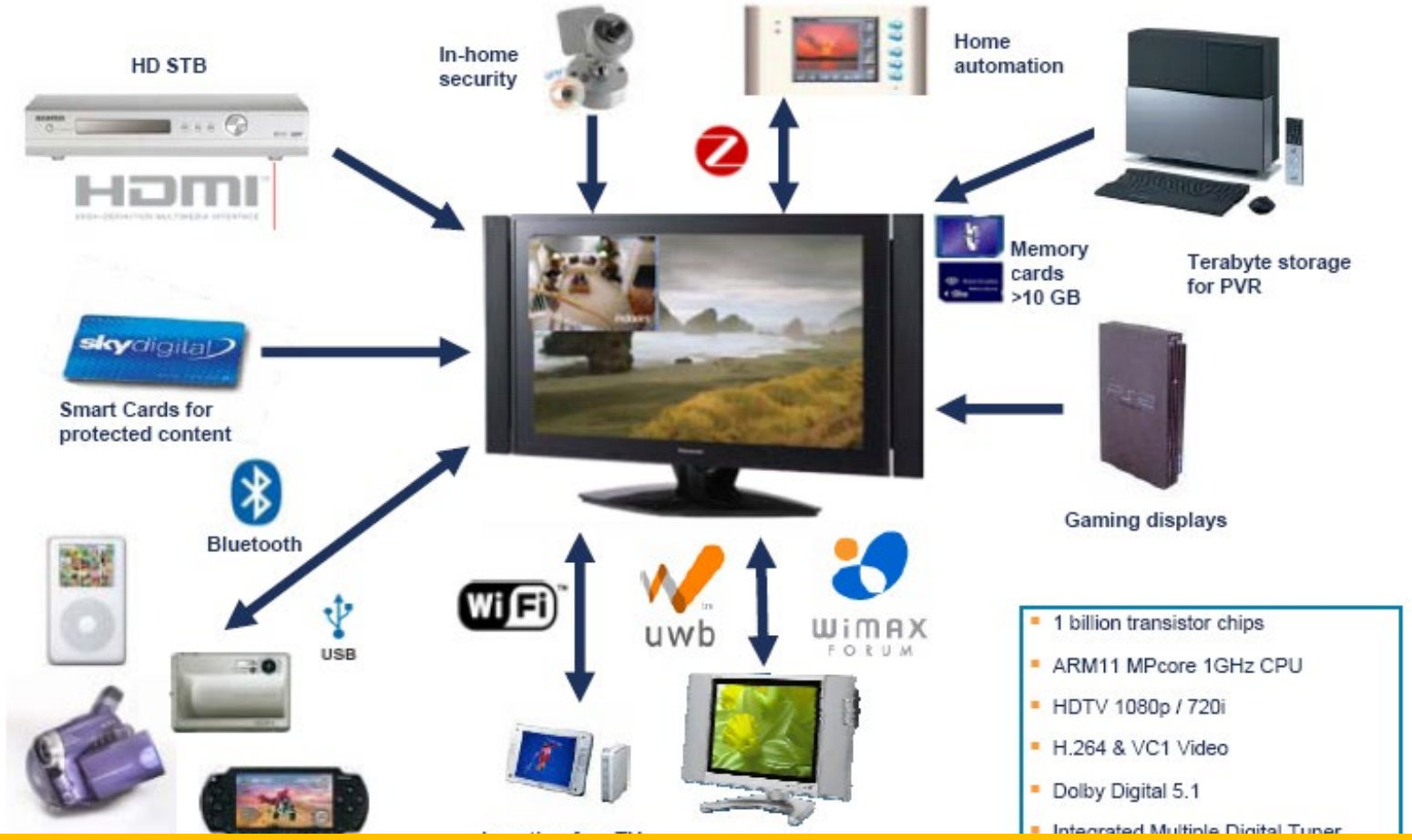
2nd new key component) **COMMUNICATION** First in automotive applications!



"At least 9 out of 12 of new planed subsystems for next-generation cars need networks of microprogrammed embedded systems"

[Toyota-ARM10]

But also in “traditional” consumer/home applications…



“22% of all TVs shipped worldwide in 2010 were Network-enabled; 25M units in 2010 and 122M expected in 2014 (approximately 75%)”

[TechNews, Dec. 2010]

3rd new key component) **I/O VERSATILITY** Multiple types of I/O interfaces are needed

- Traditionally, none or just keyboards and command-line

"I see no advantage whatsoever to a graphical user interface (GUI)" **B. Gates – Microsoft, 1983**

- Jan. 24, 1984, 1st Macintosh: a mouse and a GUI for the first individual computer
- Dec. 1996: ***"Mobile phone vendors and entertainment providers will turn to a virtual multi-IO display-based smart phone to offer powerful new services, and differentiate themselves from competitors."*** **[Reflection Tech., Inc]**

Nokia 7710, 2003-04

Wii console, Nov. 2006

iPhone, Jun. 2007

Nintendo 3DS, Feb. 26, 2011

What has changed in “embedded Systems”?

- Definition of microprogrammed embedded system:
  - Computer system designed to perform a domain-specific set of functions (including video, communication and large set of I/O devices), but with limited computation and soft real-time constraints. It is a portable device that includes hardware, software and mechanical parts.

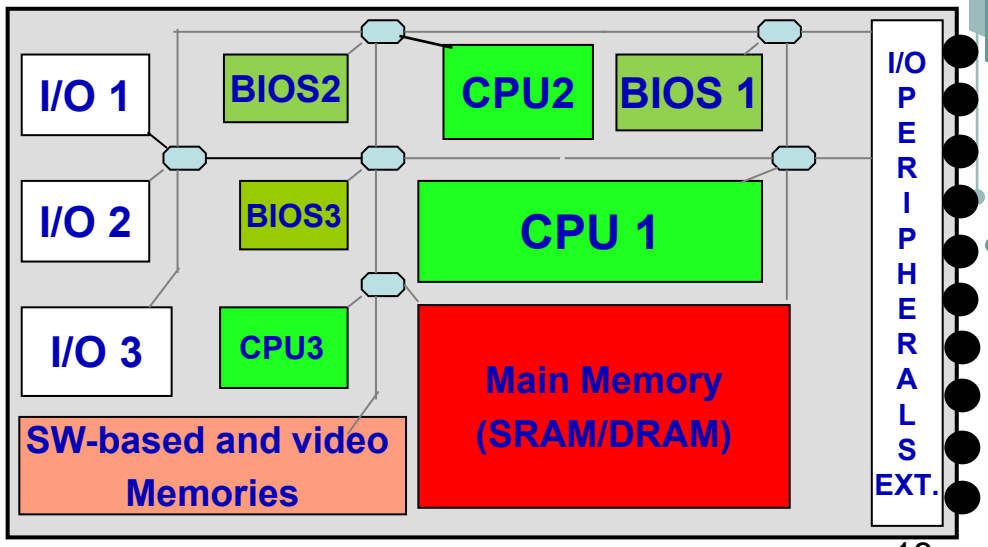
- ARMS functionality features important, but not essential
  - No lives really at risk, only customers unhappiness…
- Not general purpose computing! Still required embedded design constraints not to lose the market
  - Small size/weight, lower cost possible (medium production vol.)
  - **Low power integration is the most important constraint**:
    1. HW: minimum set of components for target functionality
    2. SW: efficient use of components in different application domains

“Current 3G phones can hardly be operated nowadays for more than an hour, if data is really being transmitted” **[Financial Times, 2010]**

Hardware Architectures of microprogrammed embedded systems

- Four fundamental hardware components
  1. Central Processing unit/s (CPUs): 32-bit **R**educed **I**nstruction **S**et **C**omputers (RISC)
  2. Memory hierarchy: 1-2 level of cache, fast and main memories, and external solid-state memories
  3. Interconnects: One or more multiple buses (different number of bit widths or speeds)
  4. I/O devices: Multiple, custom set per domain

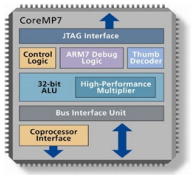
- NDS Lite architecture
  - 2 CPUs: ARM9 and ARM7
  - D-/I-Cache, 4MB main mem., 656KB of VRAM, two 16KB fast RAM
  - 17 16-bit and eight 32-bit buses
  - 1 TFT LCDs, 1 TFT touch-screen, 8 buttons, 4-direction pad, microphone, speaker, Wifi and GBA cart-based flash card, …



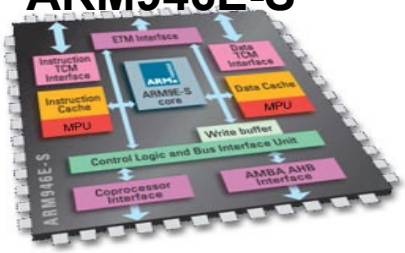
NDS processing and I/O management

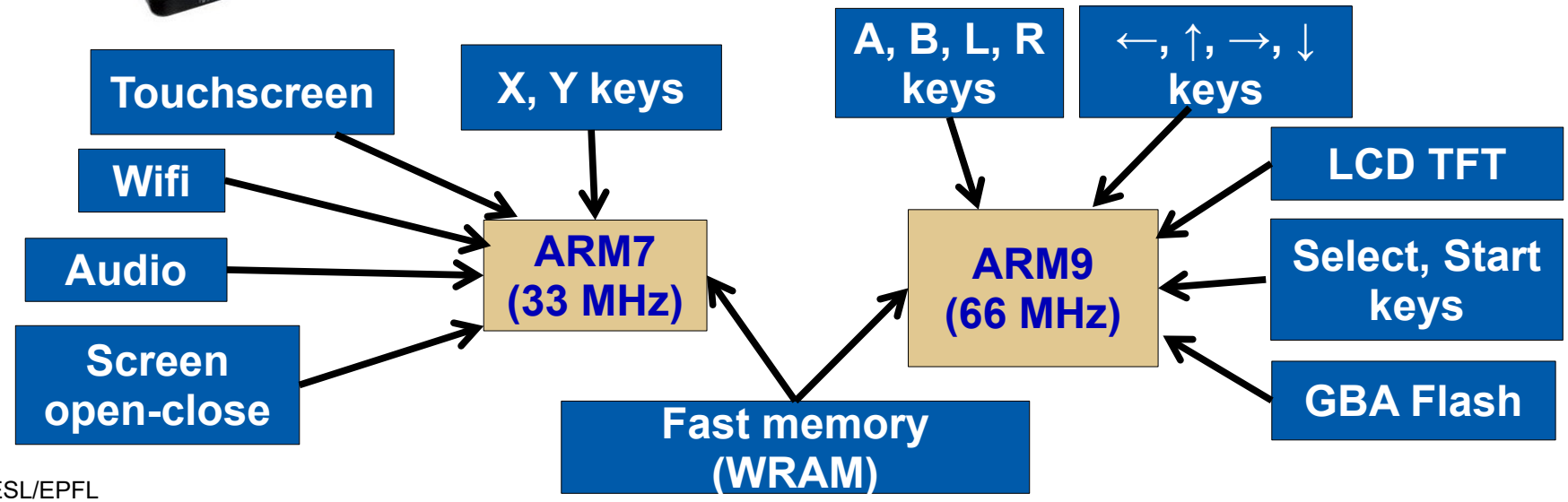
- Two asymmetric 32-bit Advanced RISC Microprocessors (ARM) cores
  - ARM 946E-S: user software processing and main I/O peripherals
  - ARM 7TDMI-S: dedicated I/O support (sound, wifi and specific keys)

ARM7TDMI-S



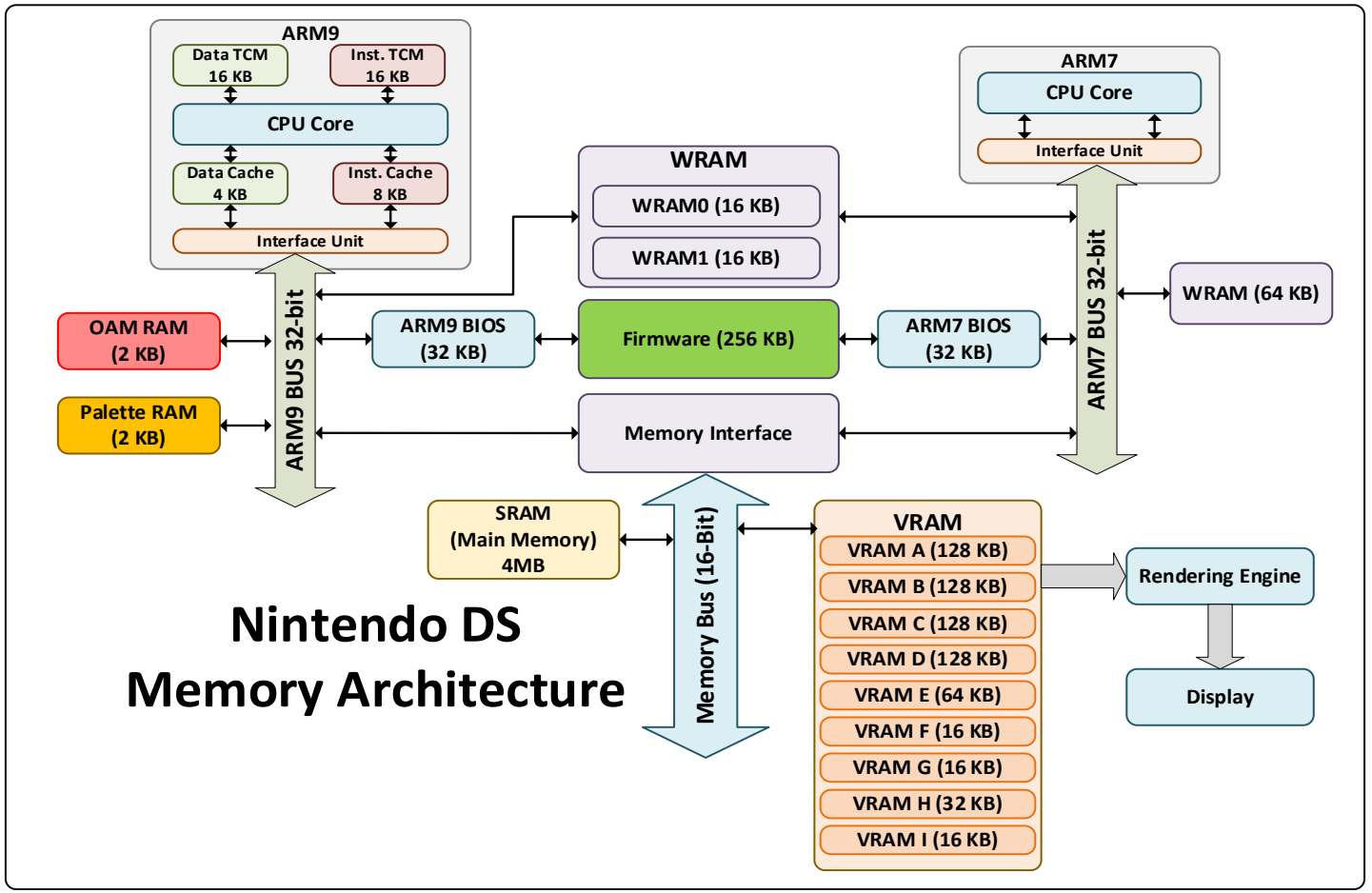
ARM946E-S





Nintendo DS detailed memory map

- Complex mem. Hierarchy, and interconnects widths for each I/O peripheral
  - ARM9 SW-controlled memories: D-/I- tightly coupled memories (TCMs)
  - Shared memories between both processors: WRAM 0-1
  - Several video-related memories (VRAM, Palette and OAM RAM)



SW side: controlling/programming microprogrammed embedded systems

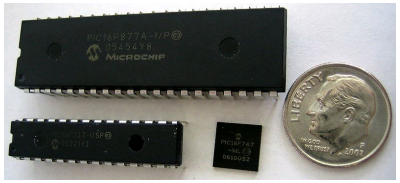
- Multiple levels of abstraction possible, the choice depends on required I/O efficiency and complexity in general of embedded system validation

Abstraction level

MCS-48 (8048, 8035, 8748) [Intel Corp., 1976]
(8-bit architecture, 64-128B RAM, 1MHz)

\-

Low level = *Assembly Language,* each line is a minimum fundamental operation to be done by microcontroller/microprocessor



PIC16x84 families [Microchip, 1993]
(14-bit architectures, 512B-2.5KB RAM, up to 20MHz)

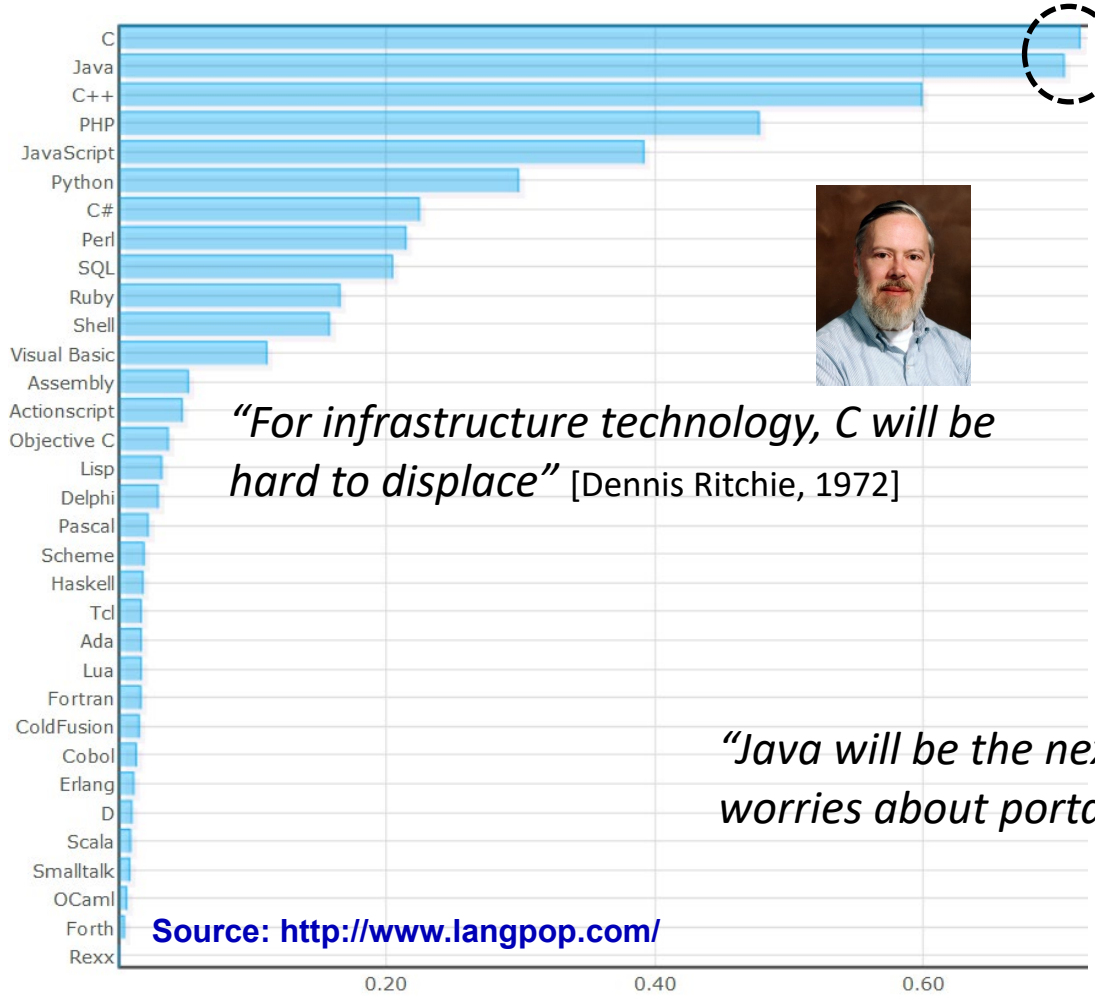
High-level abstraction languages

1967	BCPL	Martin Richards	Cambridge
1969	B	Ken Thomson	Bell Labs
1972	**C**	**Dennis Ritchie**	**Bell Labs**
1983	C++	Bjarne Stroustrup	Bell Labs
1995	Java	James Gosling	Sun

\+

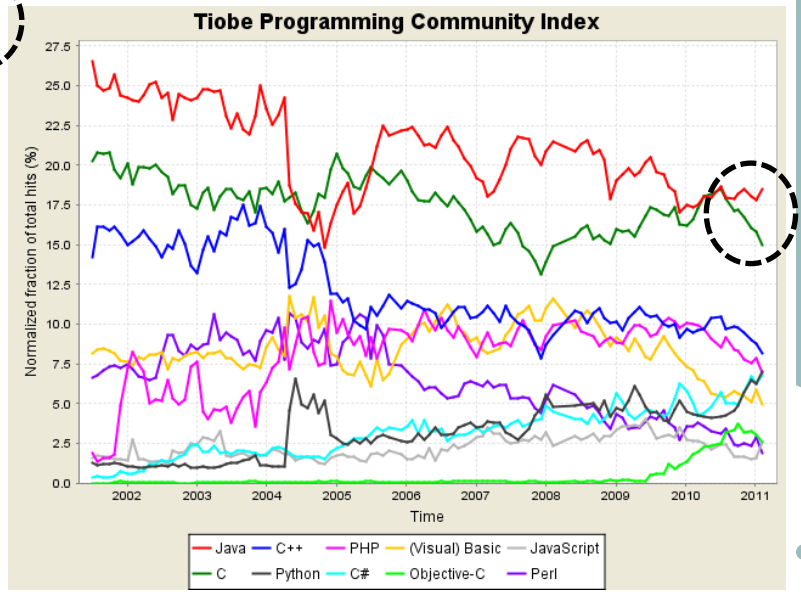
Java and C: trendiest languages for microprogrammed embedded systems

- Very different objectives, but both have found their reason to be there
  - Java: most retargettable solution (Virtual machine), interpreted language
  - C: closest to HW language, very optimized compilation technology



Source: http://www.langpop.com/

"For infrastructure technology, C will be hard to displace" [Dennis Ritchie, 1972]



Source: http://www.tiobe.com/

"Java will be the next wave in computing: no worries about portability" [James A. Gosling, 1995]

Cross-development for microprogrammed embedded systems

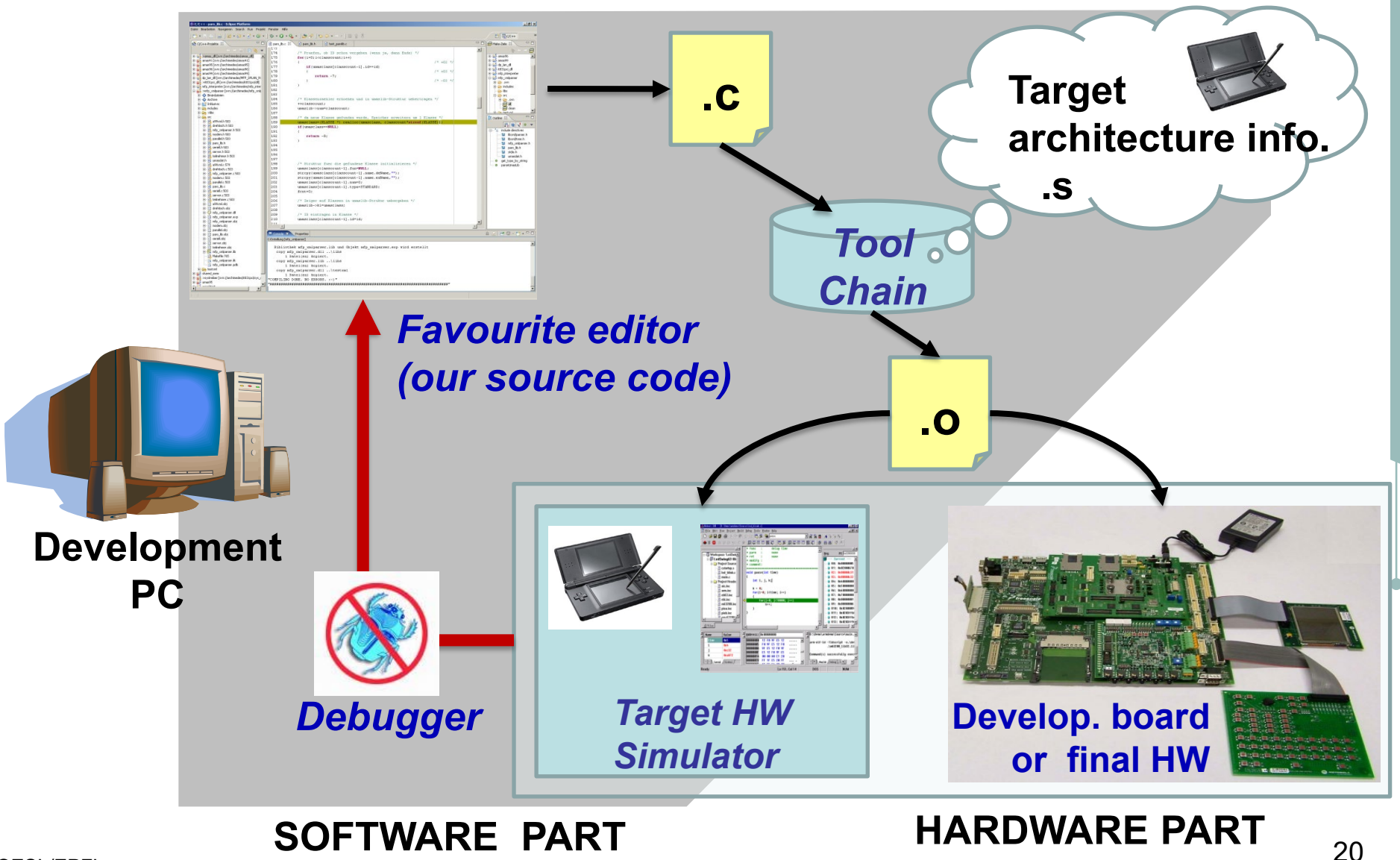
- Microprogrammed embedded systems are devices with limited resources, and are typically not powerful enough to run a compiler, a file system or a development environment
- Cross development is the separation of the system build environment from the target environment
- Benefits
  - Faster compilation of applications
  - Debugging and testing with more resources than available on target embedded system

Cross-toolchain for microprogrammed embedded systems based on C language

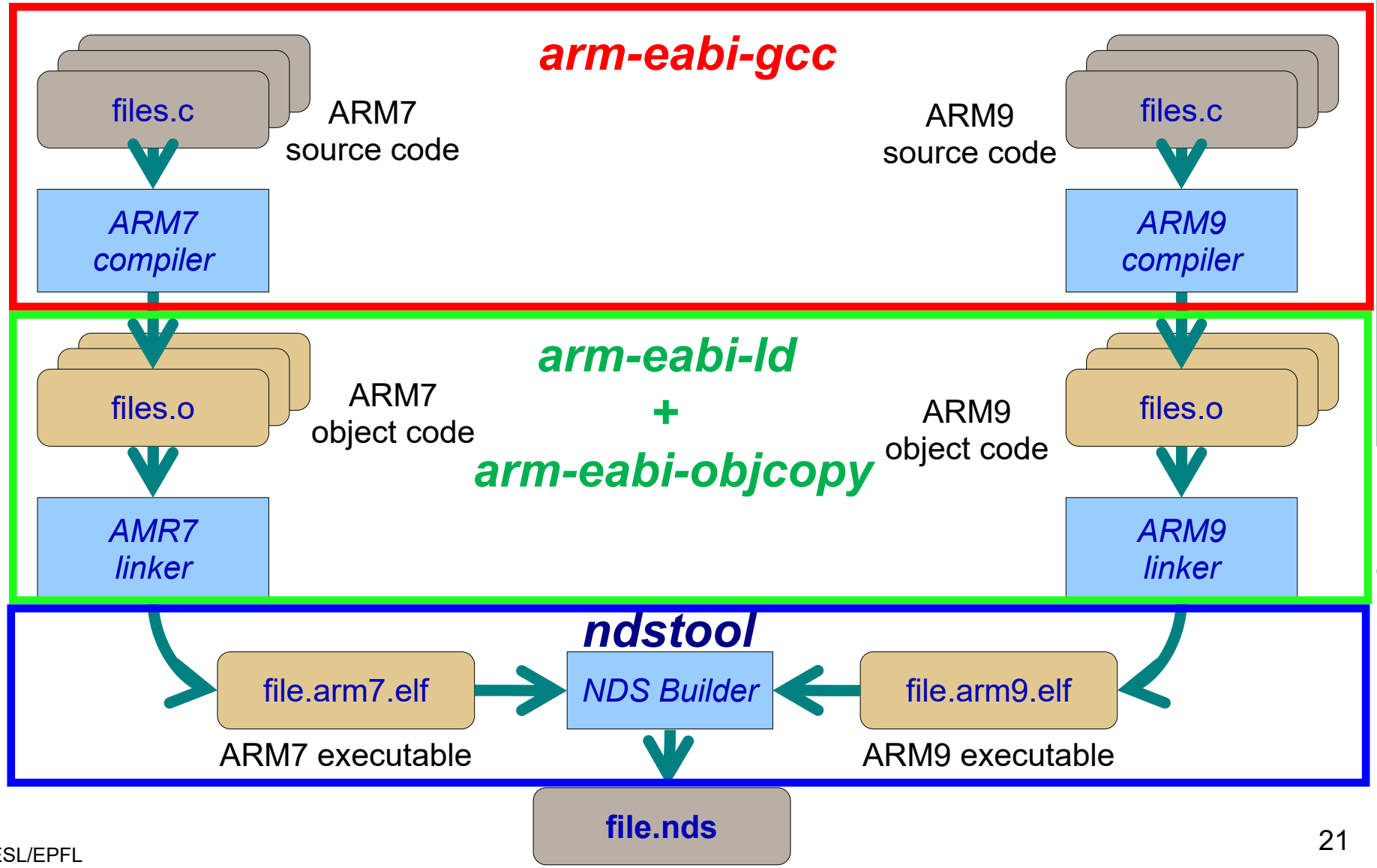
- The complete flow to generate the final binary for the target platform and its validation for the target microprogrammed embedded systems requires a cross-compilation toolchain

- It is a set of tools running on a host machine, used to
  1. Pre-process header files (*#include*) and macros (*#define*) and Compile high-level source code (*.c*) the target object code (*.o*)
     - Possible to get assembly output as intermediate step (*.s*)
  2. Link pre-existing collections or libraries of object files (*.o*) to obtain a final executable object (*.elf*) with all the necessary object code
  3. Pack and format the executable object into a format that can run with the target memory hierarchy and I/O subsystem (*.nds*)

NDS: System co-design framework for cross-development with C language

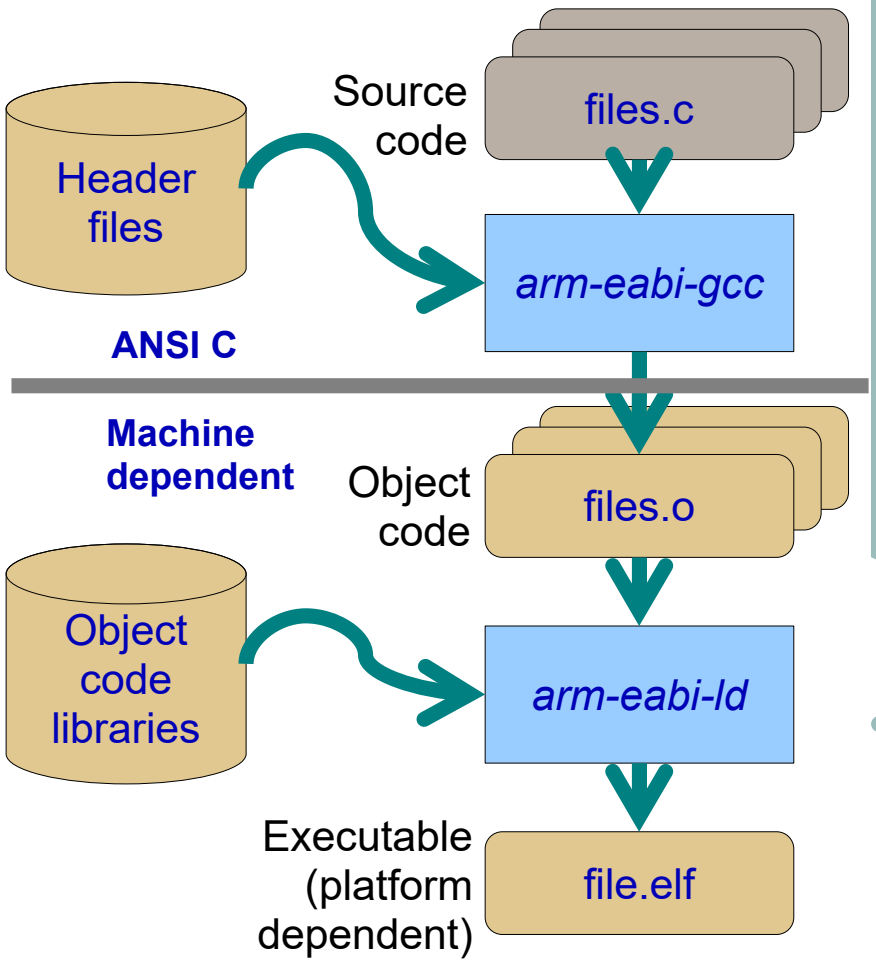


C-Based cross-compilation flow for Nintendo DS: 2 processors – 2 flows



Seeking compatibility in compilation for microprogrammed embedded systems

- Two separated phases
  1. Source code is easy to migrate
     - Standard source language (e.g., ANSI C)
     - Standard multi-platform libraries (e.g., *stdio*, *stdlib*, etc.)
  2. Object code is not portable, not even among compilers for the same architecture
     - Third-party object code linked in platform-dependent phase, e.g.:
       - *libnds*
       - *dswifi*
       - *PAlib*



Example: compiling C source code for NDS

```
#include <nds.h>
#include <stdio.h>
#define TRUE 1
#define FALSE 0
```

Handled by the pre-processor

```
int main(void)
{
    int i;
    i = 5 * 2;
```

Handled by the compiler

```
    consoleDemoinit();
    printf("5 times 2 is %d.\n", i);
    printf("TRUE is %d.\n", TRUE);
    printf("FALSE is %d.\n", FALSE);
}
```

Implemented in C library for the target platform (NDS)

Final packing process according to microprogrammed embedded system I/O

1. Tool to pack binaries in 1 file (**file.nds**) for NDS cartridge format: *ndstool*
   - Headers
   - Both executables

2. Additional tool to enables booting from Slot2 (e.g., GameBoy Advance): *dsbuild*
   - Different headers
   - Adds a loader

Field	Start	End	Size
Game title	0x000	0x00B	12
Game code	0x00C	0x00F	4
Maker code	0x010	0x011	2
Unit code	0x012	0x012	1
Device code	0x013	0x013	1
Card size	0x014	0x014	1

. . . **(39 fields)**

Use of Git for Lab. Sessions and Project: Introduction to Git

- Git is a version control system
  - It allows us to keep track of changes in our software projects

- Key features
  - History of projects developments
  - Efficient multi-users working environment
  - Traceability

Why using Git?

- Changes are efficiently tracked
  - You don’t need to rename several files to trace different (updated) versions
- Multiple developers can work in parallel
  - Each one on his local environment
  - Users see which files have been modified by other colleagues
  - Only desired files from different users can be merged together
- Repository on cloud
  - Local versions for current updates are stored locally
  - The main repository is on the cloud
  - No risk of loosing your work

Mandatory use for final NDS projects in EE-310 course!

How to install git repositories – Linux/Unix

- Install git - Native Ubuntu OS
  - Install with:
    - **sudo apt-get update**
    - **sudo apt-get install git**
  - Reference – other distributions
    Git (git-scm.com)

- Check installation success with
  - **git –version**
  - (should return the number of the installed git version)

We strongly encourage you to use these commands from Ubuntu in the virtual machine directly!

How to install git repositories – Mac OS

- Install git - Mac OS
  - Already installed if you have Xcode package
  - Install with:
    - **brew install git**
  - Reference
    Git - Downloading Package (git-scm.com)

- Check installation success with
  - **git –version**
  - (should return the number of the installed git version)

How to install git repositories - Windows

- Install git - Windows PowerShell
  - Power Shell environment for Git: ***posh-git*** module
  - Check all required packages are already installed
  - Install reference commands
    Git - Git in PowerShell (git-scm.com)
  - Download installer
    Git - Downloading Package (git-scm.com)

- Check installation success with
  - **git –version**
  - (should return the number of the installed git version)

How to manage git repositories

- Main git commands
  - **git clone**
  - **git status**
  - **git fetch**
  - **git pull**
  - **git push**
  - **git add /my_path/my_file_to_add**
  - **git remove /my_path/my_file_to_add**
  - **git commit –m “short description of commit goal”**

NDS Git repository

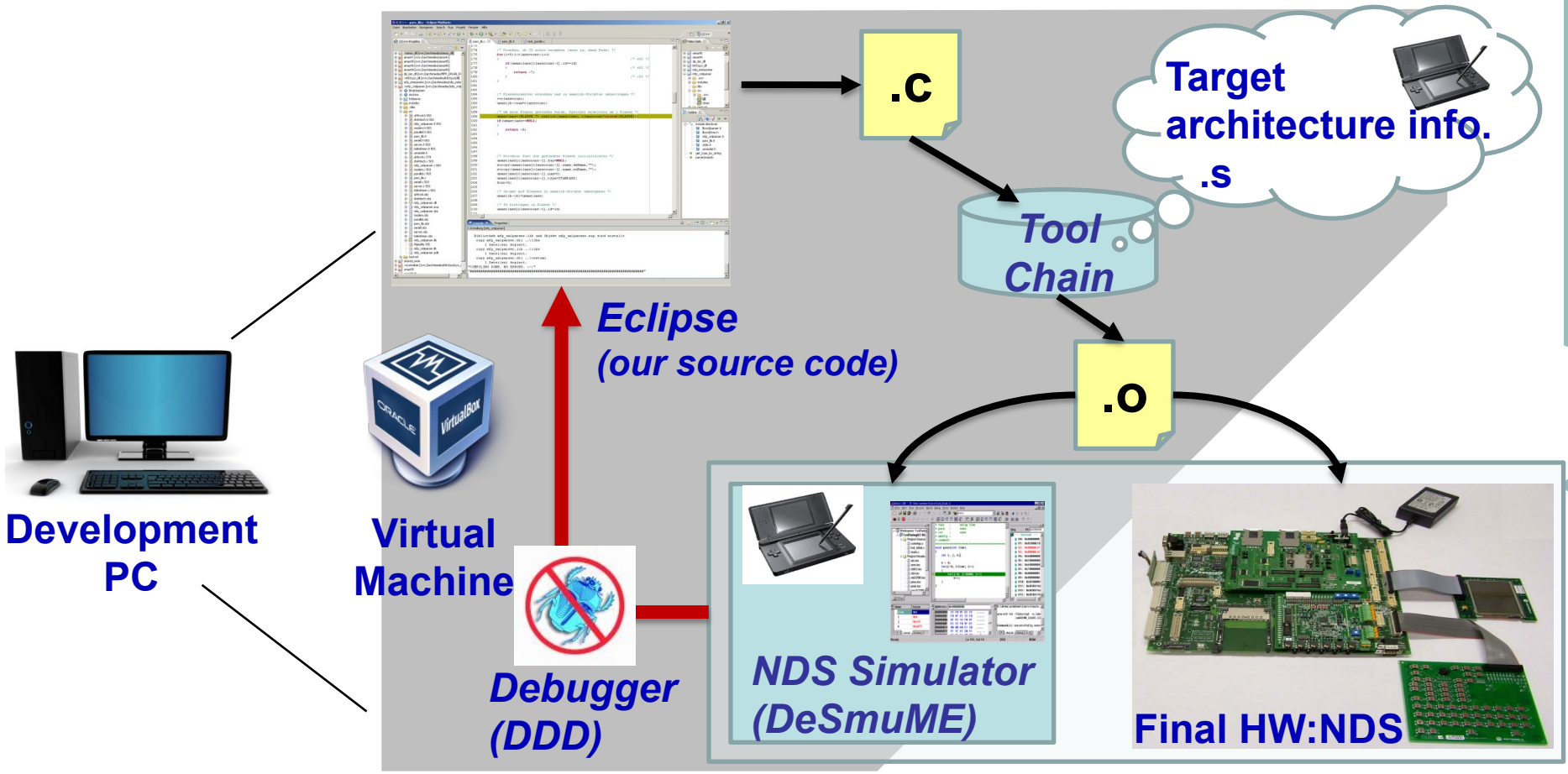
- Our course's repository is on EPFL GitLab:
  https://gitlab.epfl.ch/syst-mes-embarqu-s-microprogramm-s1/practical-works

- All you need for the practical work sessions is there
  - **Instructions**
  - **Source code**
  - **Solutions (of previous week)**

- Repo is updated every Friday after the lab session!
  - **Practical work of following week**
  - **Solutions of current week**

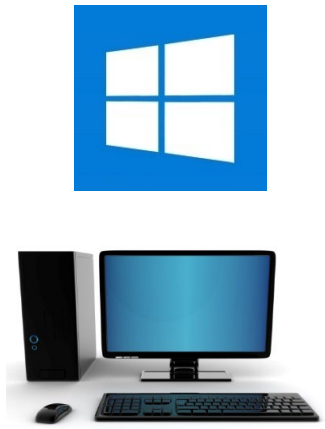
Practical Work 2: Getting familiar with the environment and starting with C for NDS

- Part A: First steps with the Virtual Machine for the NDS
  - Use of VirtualBox: http://www.virtualbox.org



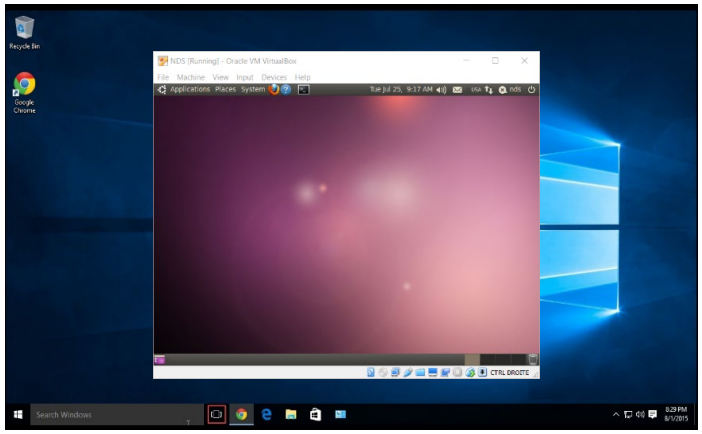
Practical Work 2: Getting familiar with the environment and starting with C for NDS

- Part A: First steps with the Virtual Machine for the NDS
  - Use of VirtualBox: http://www.virtualbox.org



Development PC

Virtual Box



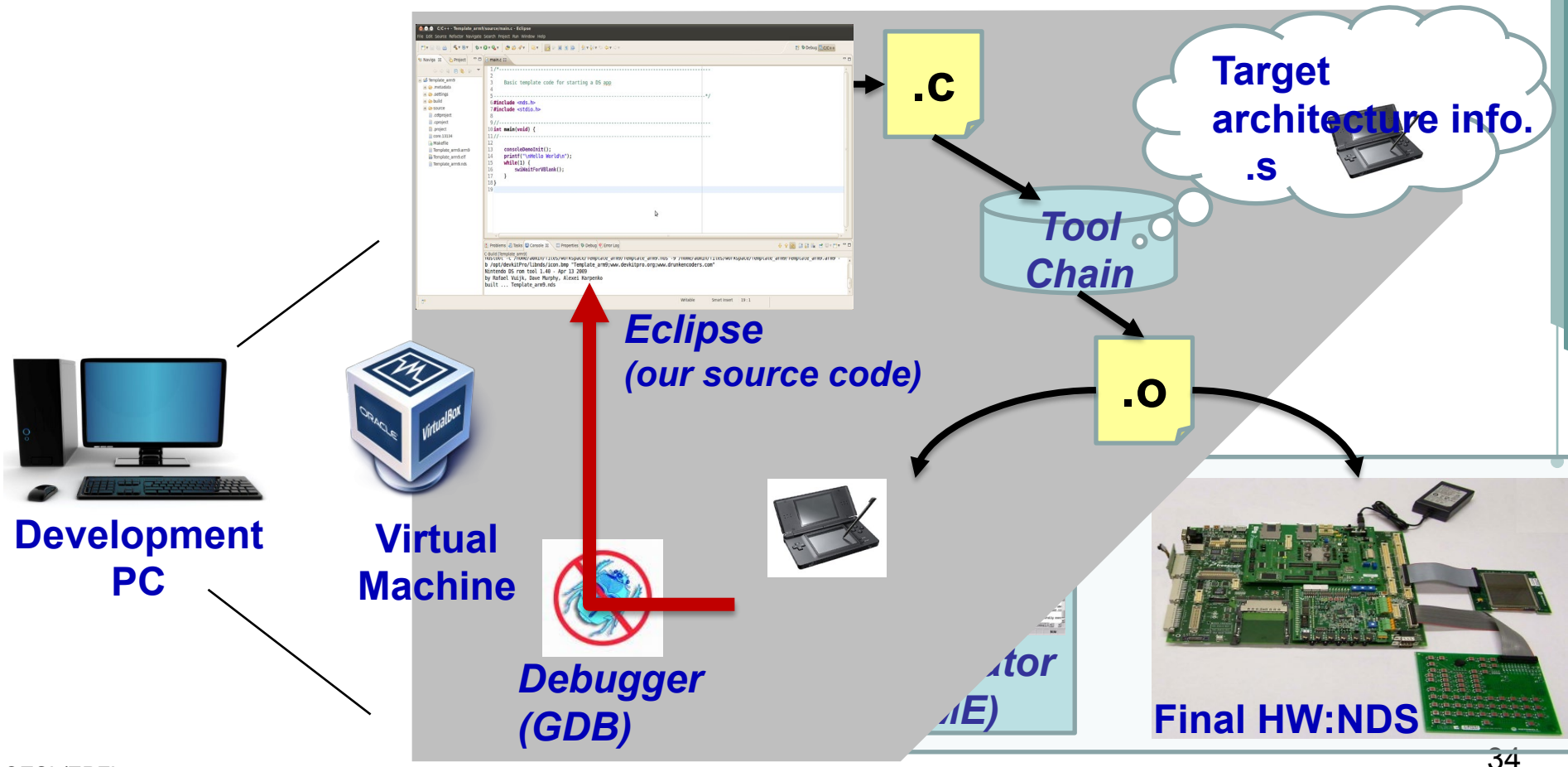
Current environment

Virtual Box allows to install new OS on top of the current OS

Working environment

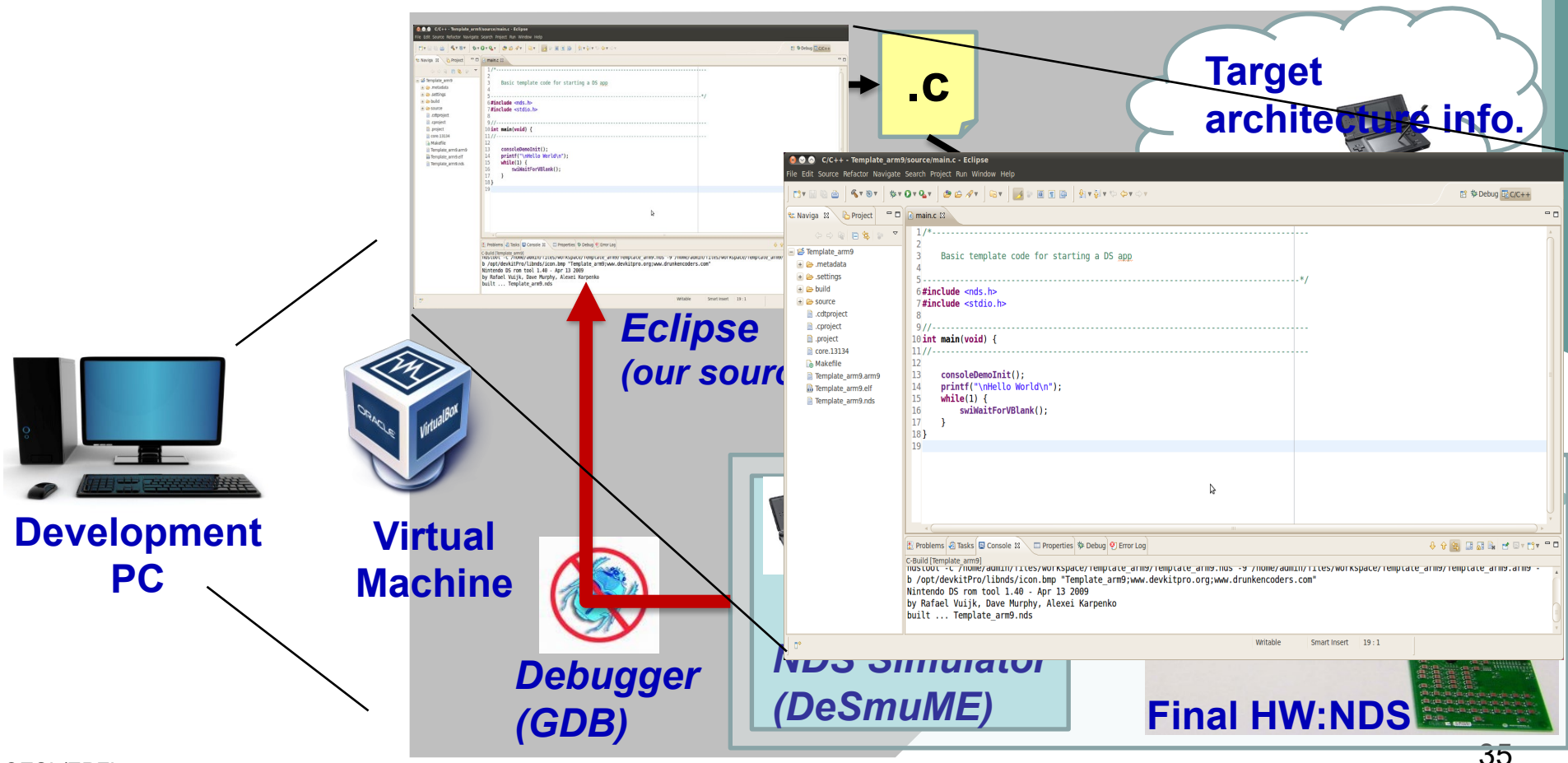
Practical Work 2: Getting familiar with the environment and starting with C for NDS

- Part A: First steps with the Virtual Machine for the NDS
  - Use of VirtualBox: http://www.virtualbox.org
- Part B: Getting in touch with C in the NDS: “Hello World”



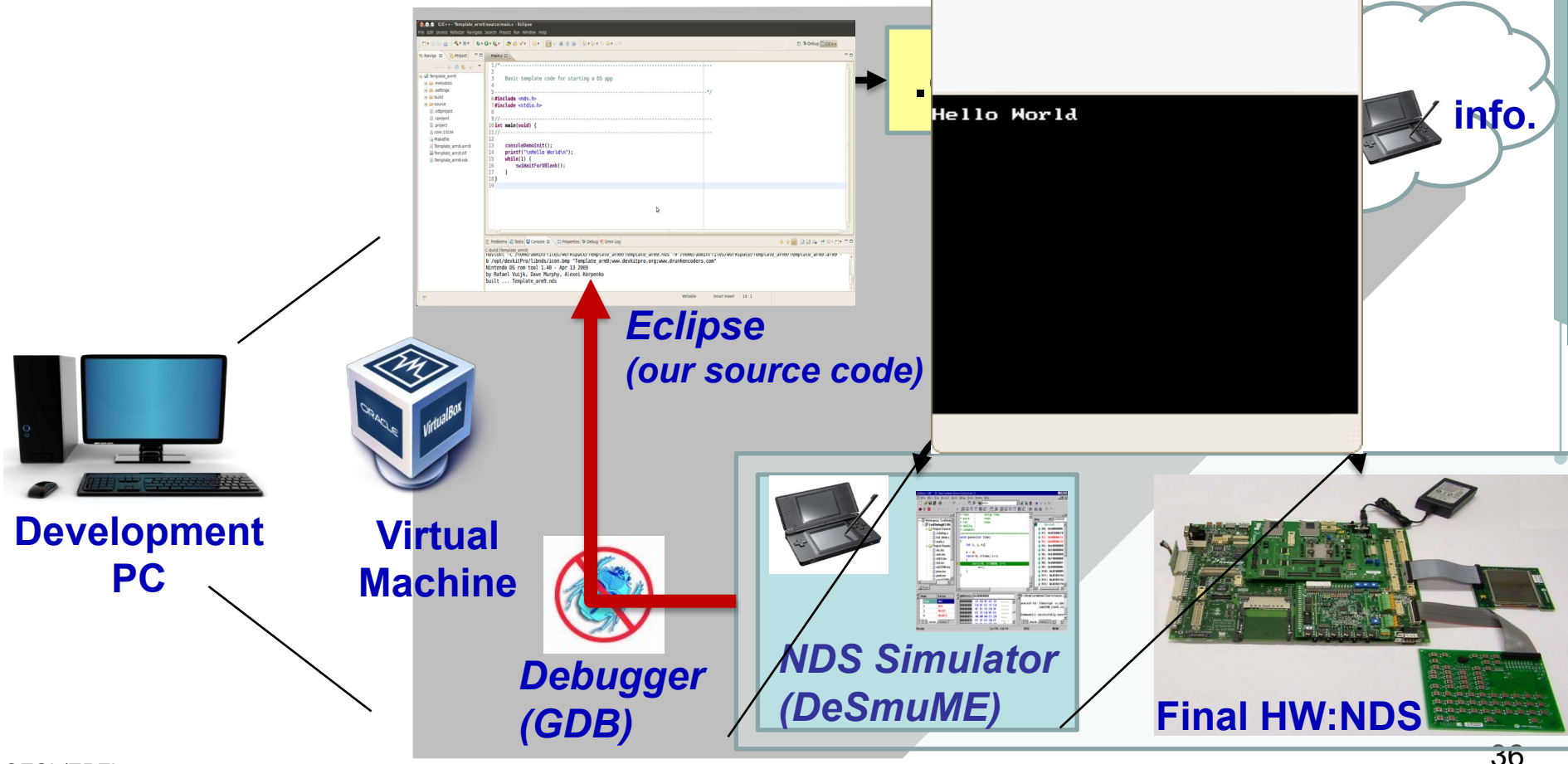
Practical Work 2: Getting familiar with the environment and starting with C for NDS

- Part A: First steps with the Virtual Machine for the NDS
  - Use of VirtualBox: http://www.virtualbox.org
- Part B: Getting in touch with C in the NDS: “Hello World”



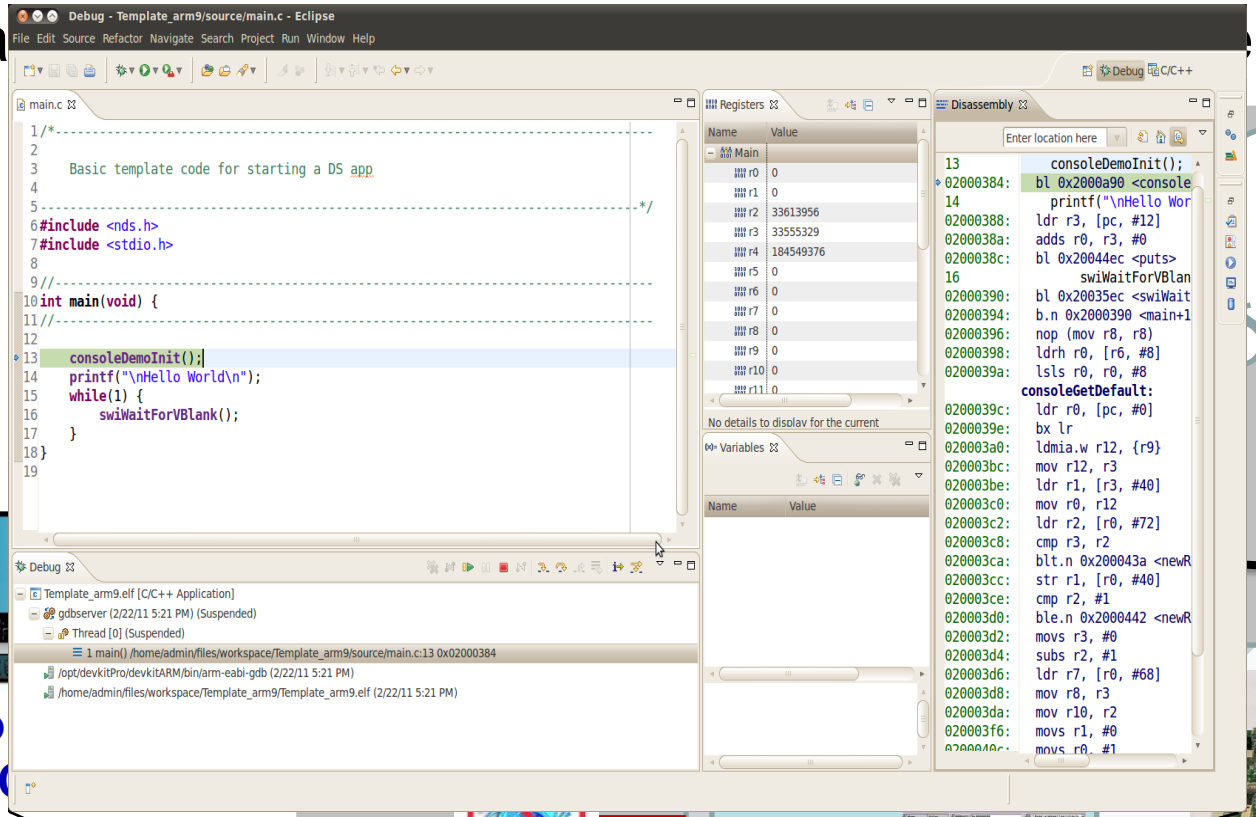
Practical Work 2: Getting familiar with the environment and start … NDS

- Part A: First steps with the Virtual M… S
  - Use of VirtualBox: http://www.virtualbox…
- Part B: Getting in touch with C in the … d”



Practical Work 2: Getting familiar with the environment and starting with C for NDS

- Part A: First steps with the Virtual Machine for the NDS
  - Use of VirtualBox: http://www.virtualbox.org
- Pa… …llo World”



Practical Work 2: Getting familiar with the environment and starting with C for NDS

- Part A: First steps with the Virtual Machine for the NDS
  - Use of VirtualBox: http://www.virtualbox.org
- Part B: Getting in touch with C in the NDS: “Hello World”
- Additional exercises
  - Exercise 1: Use of *printf()*. Change the ¨Hello world¨ message by another one
  - Exercise 2: Find the maximum, the minimum, and the average among the values of a pre-initialized array of integers.
  - Exercise 3: For a given integer *n* find the factorial (*n!*) with an iterative function and/or an recursive one.
  - Exercise 4: Given 2 integer numbers, find their Greatest Common Divisor (GCD).

Practical Work 3: Advanced C Programming on the Nintendo DS

- Exercises (and homework)
  - Exercise 1 – Separating function prototypes from implementations
  - Exercise 2 – Displaying matrices on NDS console
  - Exercise 3 – Initialization of matrices
  - Exercise 4 – Summation of arrays and matrices
  - Exercise 5 – Vector sorting
  - Exercise 6 – Matrices multiplication
  - *Exercise 7 – Run-time resources errors on the NDS
  - *Exercise 8 – Managing the different types of memory resources in the NDS
  - *Exercise 9 – Allocating/deallocating memory
  - *Exercise 10 – Understanding how arguments are passed between NDS functions

* Data management for the NDS

Questions?

Let’s create our first program for NDS