BG003 : Renesas Technical Training

Development Tools for Micro Computer Products

Renesas Design Vietnam Co., Ltd.
Training Center

October 12, 2016

Company confidential

Contents

0. Preface

Key words

1. Embedded system overview
2. Embedded system development flow
3. Operating System
4. Real-time Operating System
5. Integrate development environment
6. Cross software
7. Simulation tools
8. Evaluation boards
9. Emulation tools

0. Preface

Words in slides 1

User

- Renesas sells semiconductor devices to our customers, such as electronics products manufacturer, automotive manufacturer, industrial products manufacturer, etc.
- Customers design, manufacture, and sell their products to end customers.
- Sometimes we take the word ‘user’ as customer, not end customer.

- User system (target system): Our customer’s system. Electronic system.
- User program (target program): Our customer’s program.
- User interface: Interface between customer’s system and our (Renesas) tool

MCU and MPU

- MCU : Micro Controller Unit

Integrated ‘single chip’ device, such as M16C, H8/H8S, SH-2(2A)
CPU, memory(ROM and RAM), and IO in one silicon

- MPU : Micro Processing Unit

Integrated device including CPU, cache, and IO, such as SH-4(4A).
External large size memory, Flash and DRAM

Words in slides 2

Firmware

- Original meaning is ‘software stored in some hardware which is hard to rewrite, ie. software stored in ROM. In old days, memory is clearly divided into two type, one is RAM and the other one is ROM.
- Recently flash memory is used widely for software storage, and due to above historical reason, software stored in flash memory is also called ‘firmware’ in some case.

Middleware

- Original meaning is ‘some software which is placed between higher layer and lower layer’.
- In embedded application, higher layer means ‘application’, and lower layer means ‘driver’.
- Example of middleware is, audio/video codec(WMA / WMV / MP3 / MPEG / H.264), telecommunication codec(G.72x), protocol stack(TCP/IP), file system.

Words in slides 3

BSP (Board Support Package)

- In embedded system, wide variety of different hardware (IO) is used, there is need for software package which is dedicated to specific hardware platform. Software package for such dedicated hardware is called ‘BSP’, as this is prepared for specific board.

IPL (Initial Program Loader)

- IPL is firmware software which runs just after hardware power on.

BIOS (Basic Input/Output System)

- BIOS is firmware software for IBM-PC compatible computer which runs just after hardware power on. This program identify and initiate IO hardware, such as HDD, CD drive, etc., and provide standard IO access software interface to higher level software, such as Windows OS.
- BIOS is very close to compact/simple OS(Operating System), but BIOS is not OS.

CMOS (Complementary Metal-Oxide Semiconductor)

- P-MOS and N-MOS pair semiconductor digital logic circuit. Commonly used for current digital semiconductor devices.
- This word has no relation to any Operating System.

Words in slides 4

- **MMU (Memory Management Unit)**
  - A part of CPU
  - Provides address translation from logical (virtual) address to physical address
- **OS (Operating System)**
- **RTOS (Real-Time Operating System)**
  - Operating System for embedded application.
  - Hard real-time, multi-task
- **Real-time**
  - One of major demand in embedded system.
  - Time sensitive.
  - Something must be done within pre-determined specific time period.
- **Task**
  - A part of application. Usually task runs on the device without MMU.
- **Process**
  - A part of application. Usually process runs on the device with MMU

1. Embedded system overview

What is embedded system?

User knows this is computer.

Standard hardware configuration, such as CPU, memory, HDD, CD/DVD drive, USB, Ethernet (or wireless), keyboard, mouse, microphone, speaker, etc.

It is possible to get different function by changing software (install new software) without changing hardware.

Internal architecture is computer system, but user doesn't know it (doesn't pay attention for it).

Application specific hardware configuration.
Dedicated and limited function for specific operation.
Smaller & Lower price than standard computer system.
User doesn't know internal software.

Usually, timing constrains are harder than standard computer system, such as start up time, quick response against input, seamless operation.

Differences between Computer and embedded software (1)

- Value of performance
  - On PC, performance means ‘total data through put’, i.e. performance means ‘how many data calculation/processing is performed in specific time period’. ‘Time period’ is longer than embedded system, 1 to some sec or more.
  - On embedded system, ‘real-time performance’ is more important than ‘data processing performance’, i.e. how long (second, or microsecond) it takes from (against) specific input, such as ‘from power on to system ready’, ‘from key tough to reaction(from power off key tough to power down)’, etc. Specific ‘job’ must be finished within proper time period.

- Software visibility
  - On PC, user knows (understands) which software he/she use, such as IE, word, excel.
  - On embedded system, user does not know what kind of software behind the system. Most of person do not know the software in mobile phone, automobile, TV, DSC, Video recorder.

- Limited hardware resources
  - PC has rich hardware resources(relatively rich against embedded system), such as rich memory, HDD(file system), keyboard, mouse, graphical display, network, PCI, USB, etc.
  - On embedded system, due to limited target functionality, cost/price reduction, size limitation, smaller power consumption, hardware resources are not rich, not only external IO devices, but also smaller memory size and lower CPU clock.

Differences between Computer and embedded software (2)

- Self (native) development and cross development
  - On PC, software development is done on PC, i.e., source code is compiled on x86 CPU, and object code(binary code) runs on x86 CPU.
  - On embedded system, source code is compiled on x86 CPU(in most cases), and object code runs on target CPU, such as SH, H8, M16C, etc.
  However, Linux has two compile mode, one is cross (compile on x86), the other one is native (compile on SH CPU, Linux OS).

- Special hardware assist requirement for debug
  - PC has common hardware. No need for hardware debug (if you are not PC manufacturer). Debugger software is ready and running on PC. No special hardware request for debug.
  - On embedded system, some target system does not have display(such as remote controller for consumer products), some target system does not have character input device(such as several controllers(ECU) for automotive). Thus, for embedded system debug, there is special need for debug, i.e. ICE: In-Circuit Emulator.

Reference Numbers (Typical Example)
- Architecture Comparison -

Architecture	8 Bit CPU	16 Bit CPU	32 Bit CPU
ALU data width	8 bit	16 bit	32 bit
Maximum Linear Address Space	64K (16 bit addressing)	16M (24 bit addressing)	4G (32 bit addressing)
Implemented Address Space (typical)	up to 64 K (on silicon)	128K to 1M or more (on silicon)	64M to 512M (external)
MMU (OS)	No	No	Yes
CPU Clock	10MHz	40 MHz	400MHz

Typical OS kernel size:

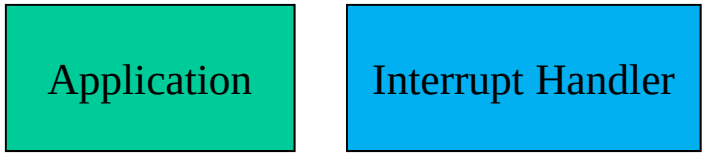
RTOS (uITRON)	Linux
10 ~ 20 KB	1 ~ 4 MB

ALU: Arithmetic logic unit
MMU: Memory management unit

Typical Example of Software Structure

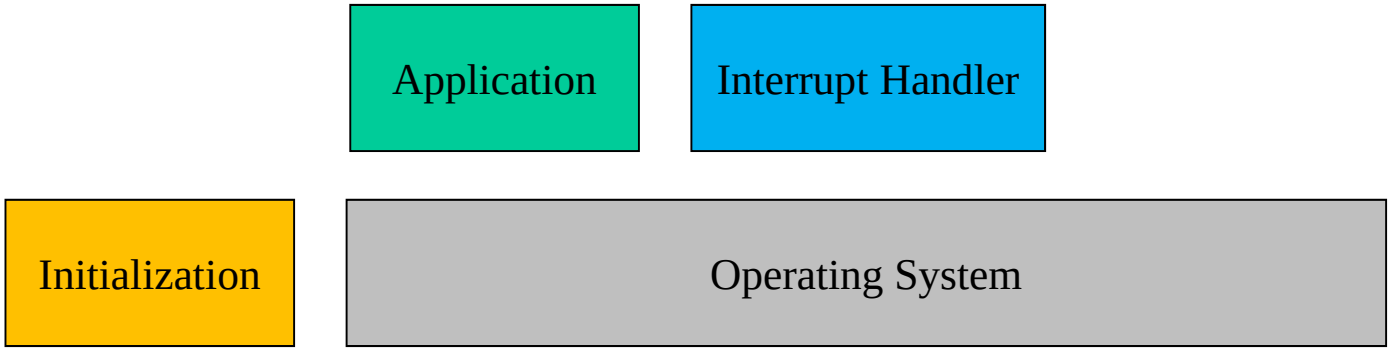
1. Just One component

2. Application and Interrupt



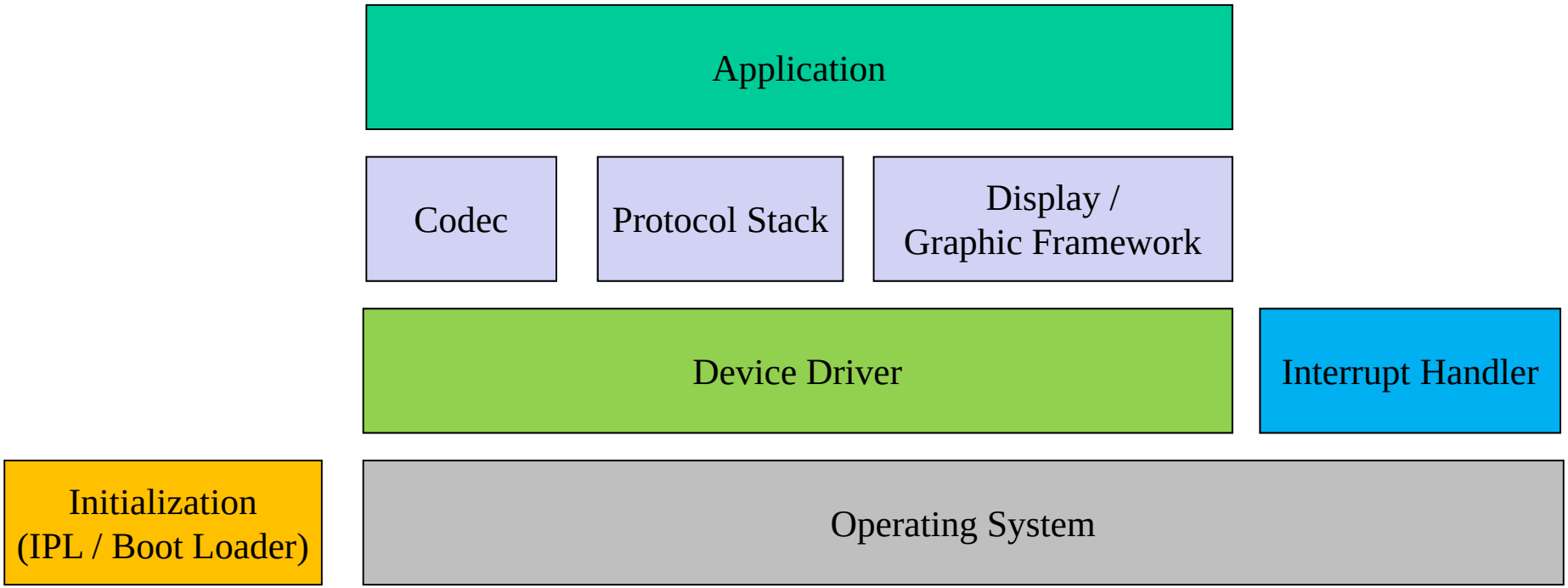
3. Initialization, Application and Interrupt

4. Initialization, Operating System, Application and Interrupt

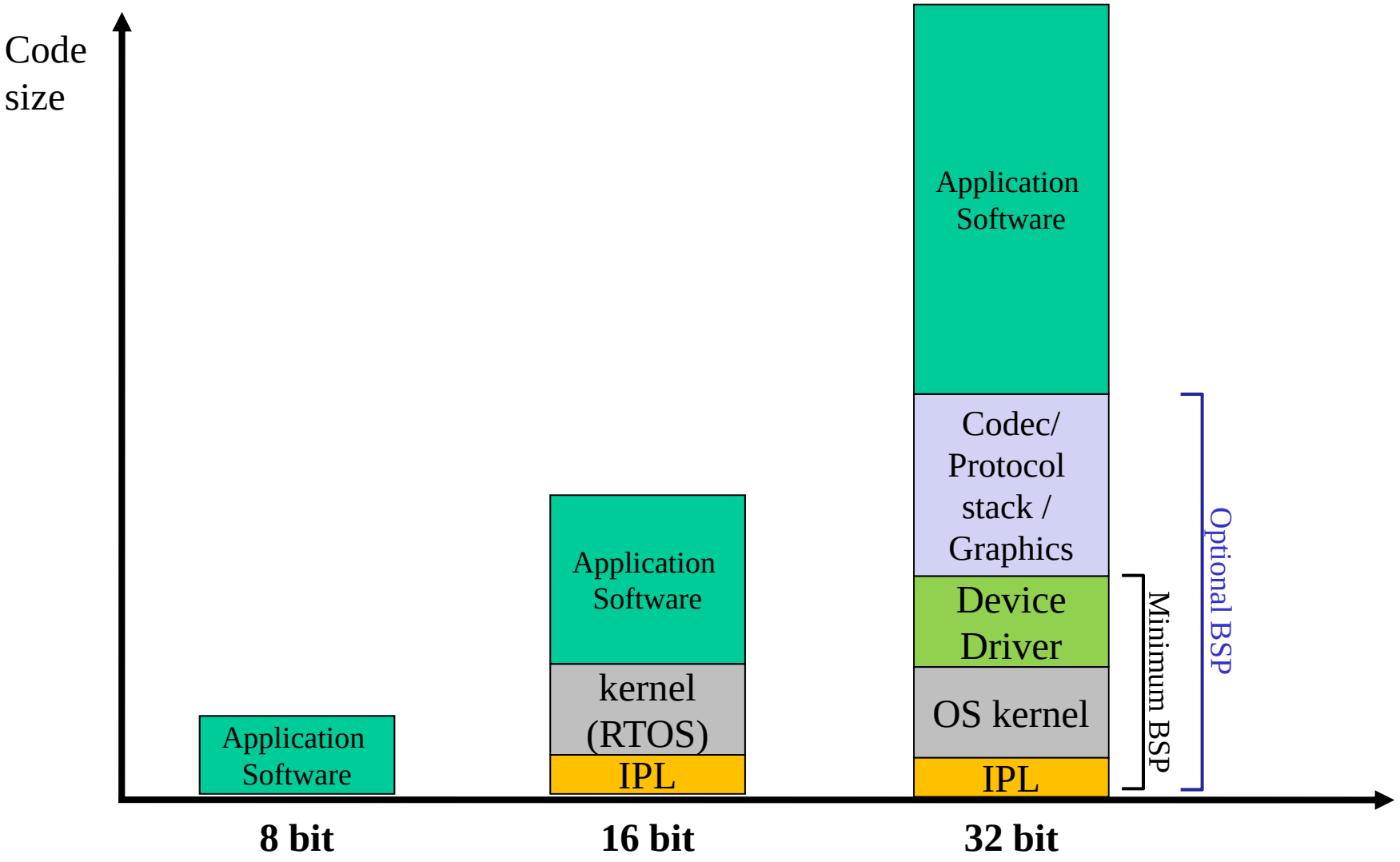


Typical Example of Software Structure

5. More Complex Software Structure



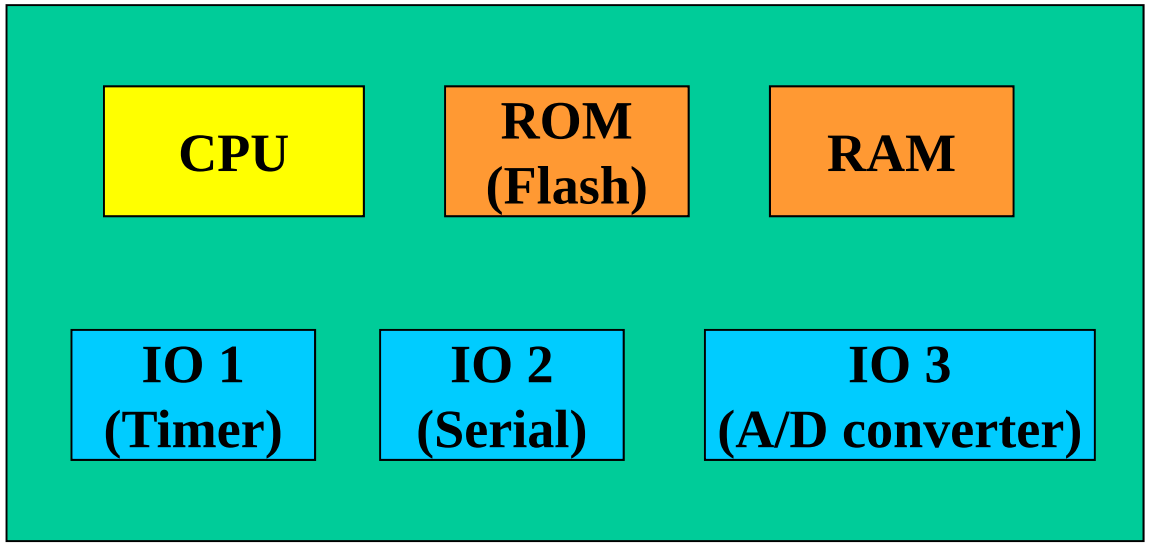
Code size and software structure examples - differences between devices -



BSP: Board support package

Type of Device
- MCU and MPU -

MCU : Micro Controller Unit (Or 'Single Chip Micro Computer')
All component is in one silicon ; CPU, memory (ROM and RAM), and IO



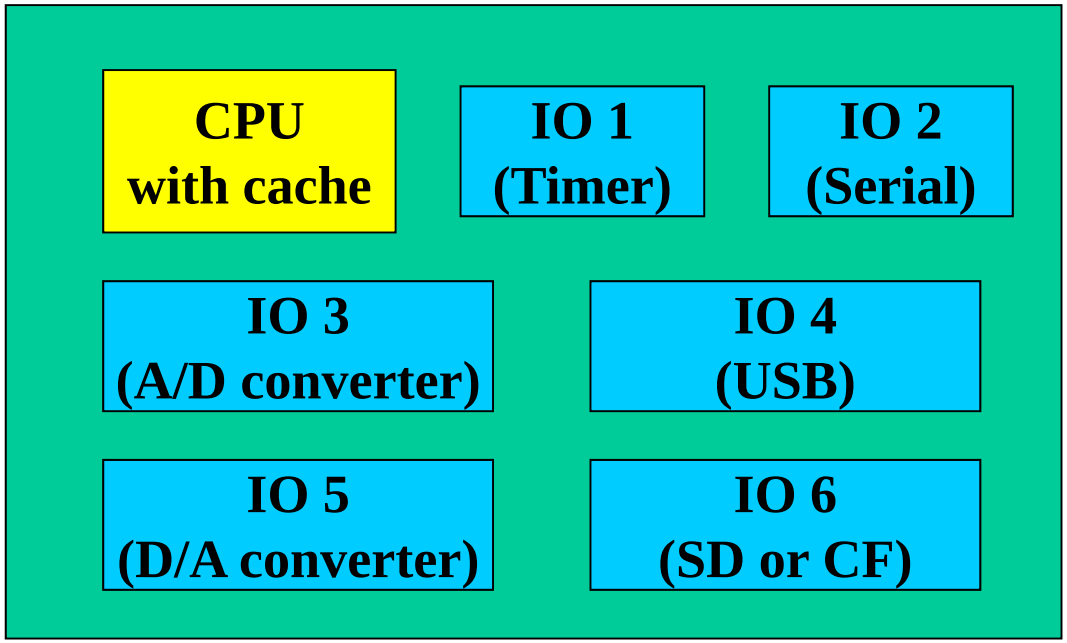
Smaller size of memory
Compact IO

MCU Device (Example)

Type of Deice
- MCU and MPU -

MPU : Micro Processing Unit
CPU, cache memory and IO, external memory

Larger size of memory
Rich IO



MPU Device (Example)

ROM (Flash)

RAM (DRAM)

External IO 1 (Network)

External IO 2 (Display Device)

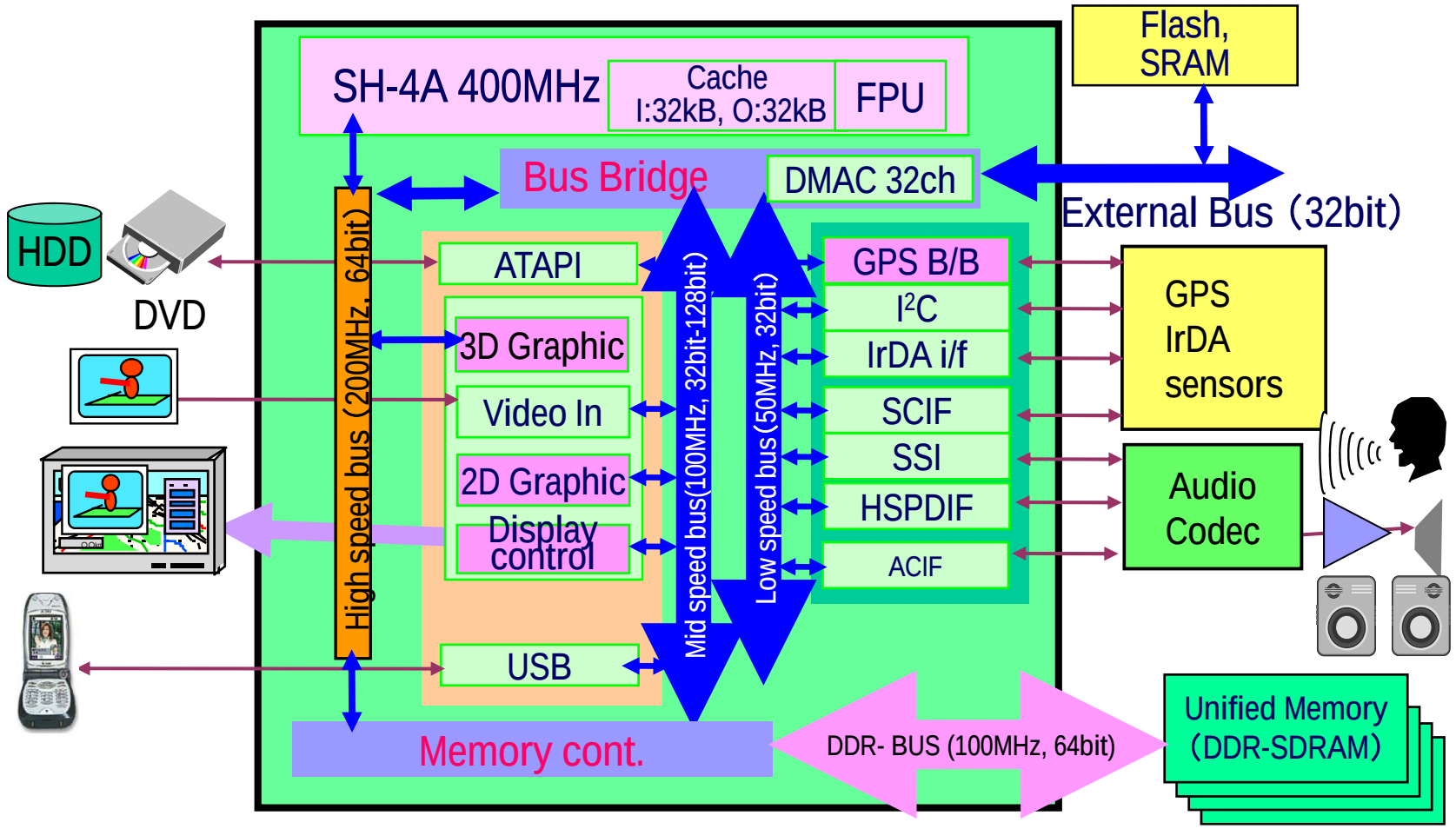
External IO 3 (Input Device)

External IO 4 (Storage Device)

MPU Device Example
SH-Navi I (SH7770) Overview

- SH-4A CPU core (upward compatible from SH-4)
- Various IP module
- Effective Bus structure

Effective Usage of software asset
Unified Hardware platform approach
(from low to High end system)

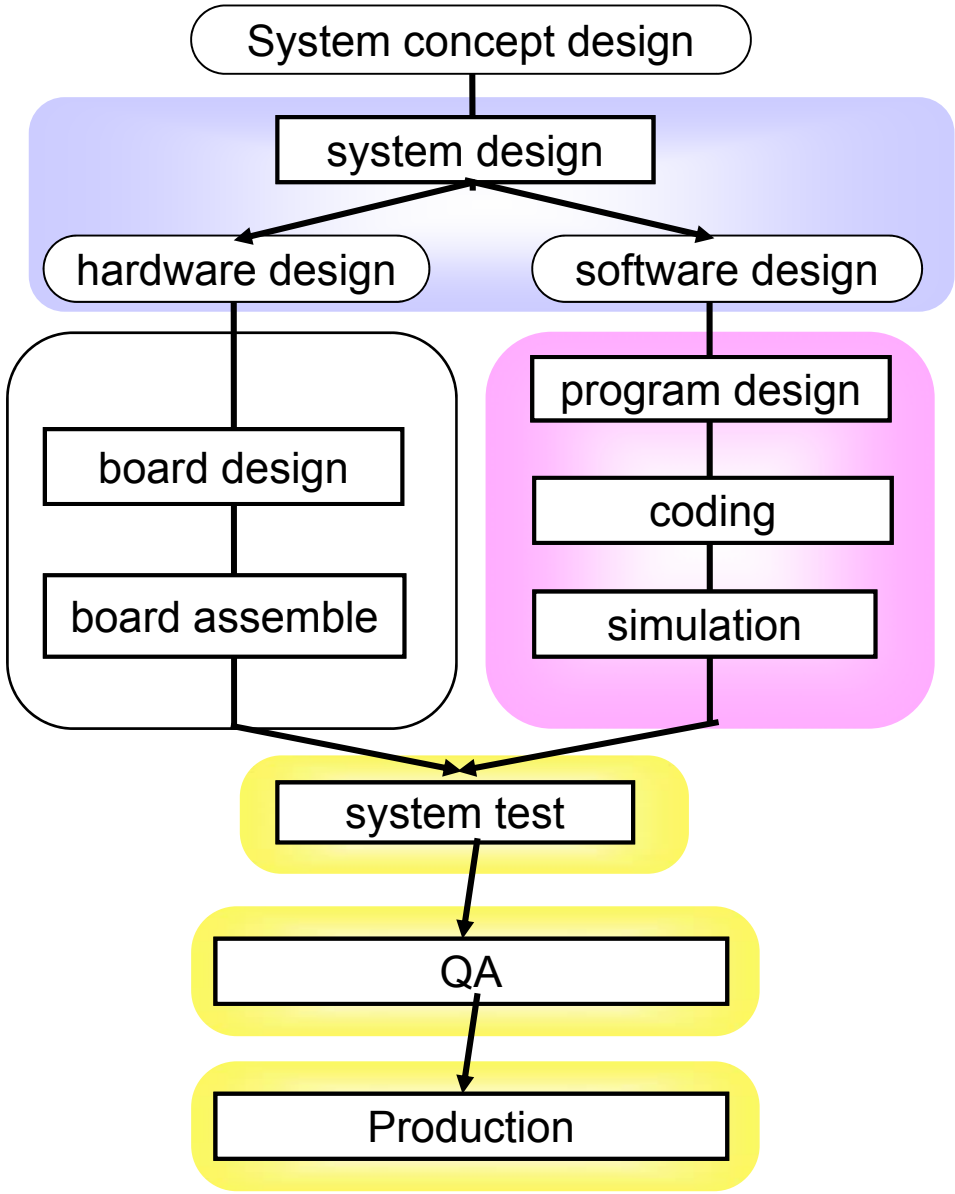


2. Embedded system development flow

System development flow

When you design the micro computer embedded system, you should design your system outline at first. And have to consider about……

- How to reduce the product cost?
- How to reduce the development cost?
- How to improve the system quality?



System design target

When you design the micro computer embedded system, your targets are…

(1) How to reduce the *product cost*?

- ✓ Choose the *cost-competitive device* as a Renesas micro processor!
- ✓ Choose the valued parts to reduce the *production cost*.
- ✓ Re-programmable device as micro controllers with on-chip flash memory.

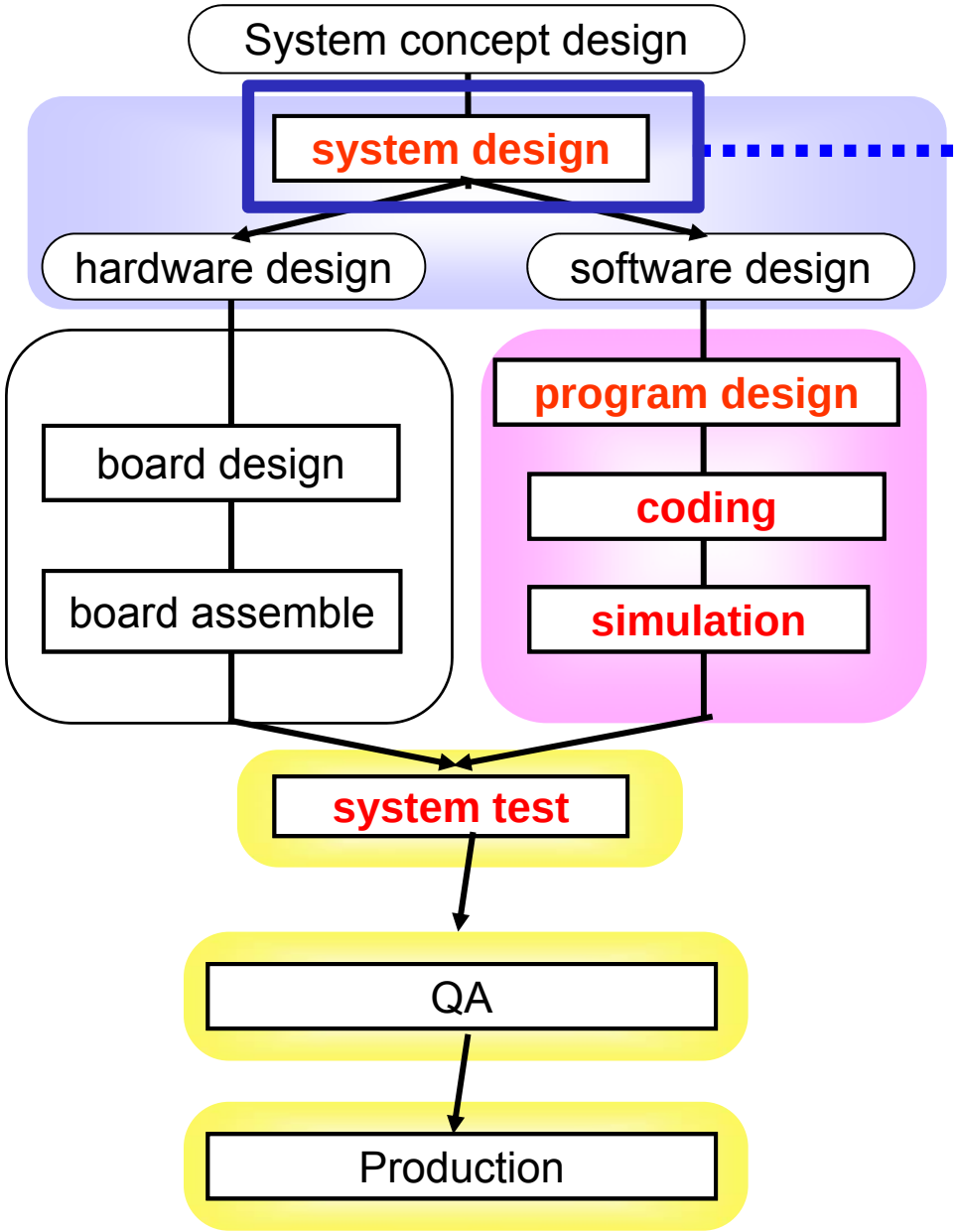
(2) How to reduce the *development cost*?

- ✓ You should concentrate on your application. You do not design all of your system. *Your system value is in your application!*
- ✓ Choose the proper *operating system* that is suitable for your system.
- ✓ Choose the proper *middleware* that is suitable for your system.

(3) How to improve the *system quality*?

- ✓ Choose the proper tool and method *which conducts your system higher quality*.
- ✓ Choose *proper debug tools* that show the debug information on your demand.

Development tools help your system design

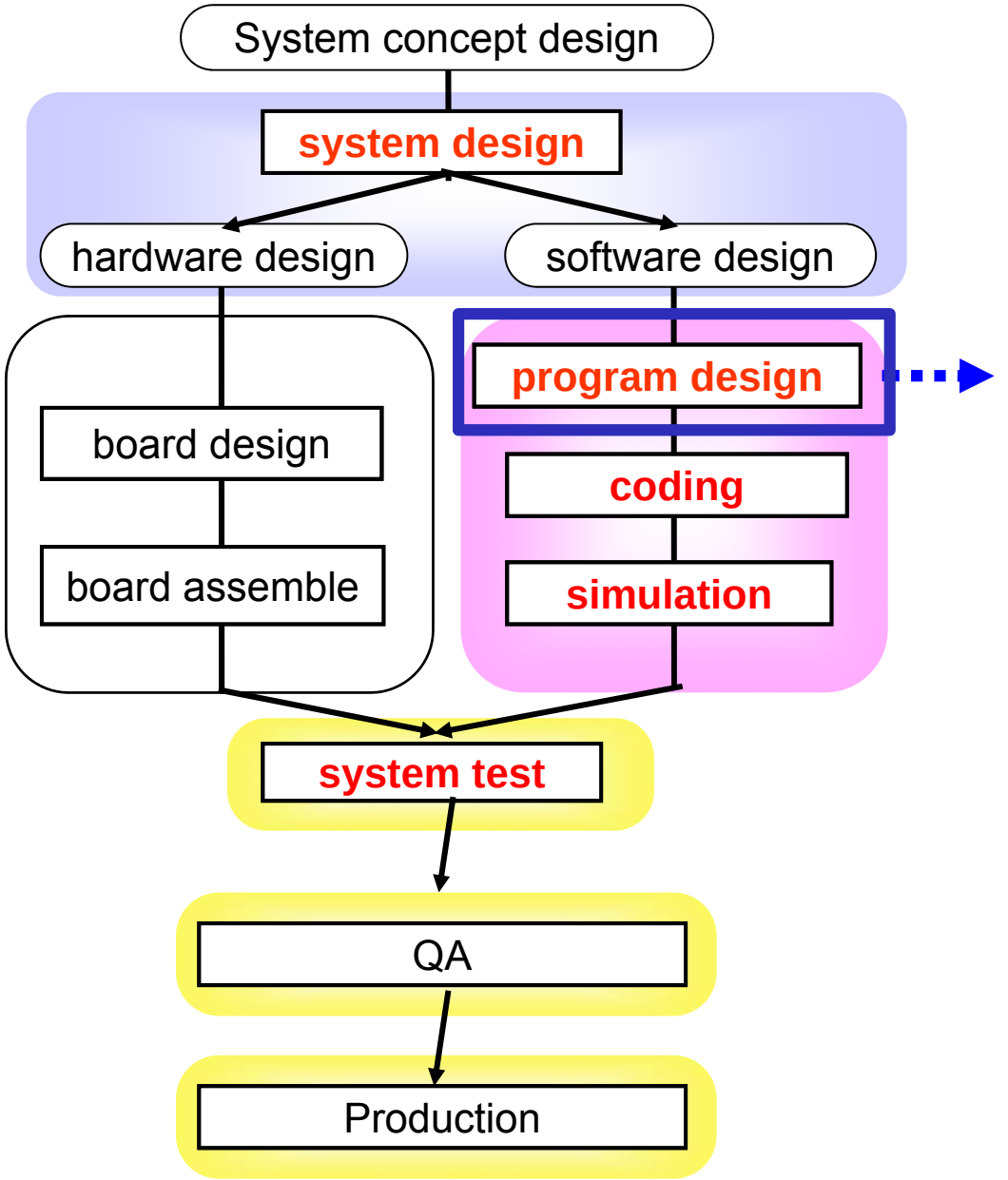


Not only function but also performance evaluation on reference platform is important.

- ✓ The reference platform is a board with microprocessor, memory and IOs.
- ✓ BSP is useful package for system / software evaluation.

T-Engine, an open and standardized platform, improves development efficiency of ubiquitous equipment.

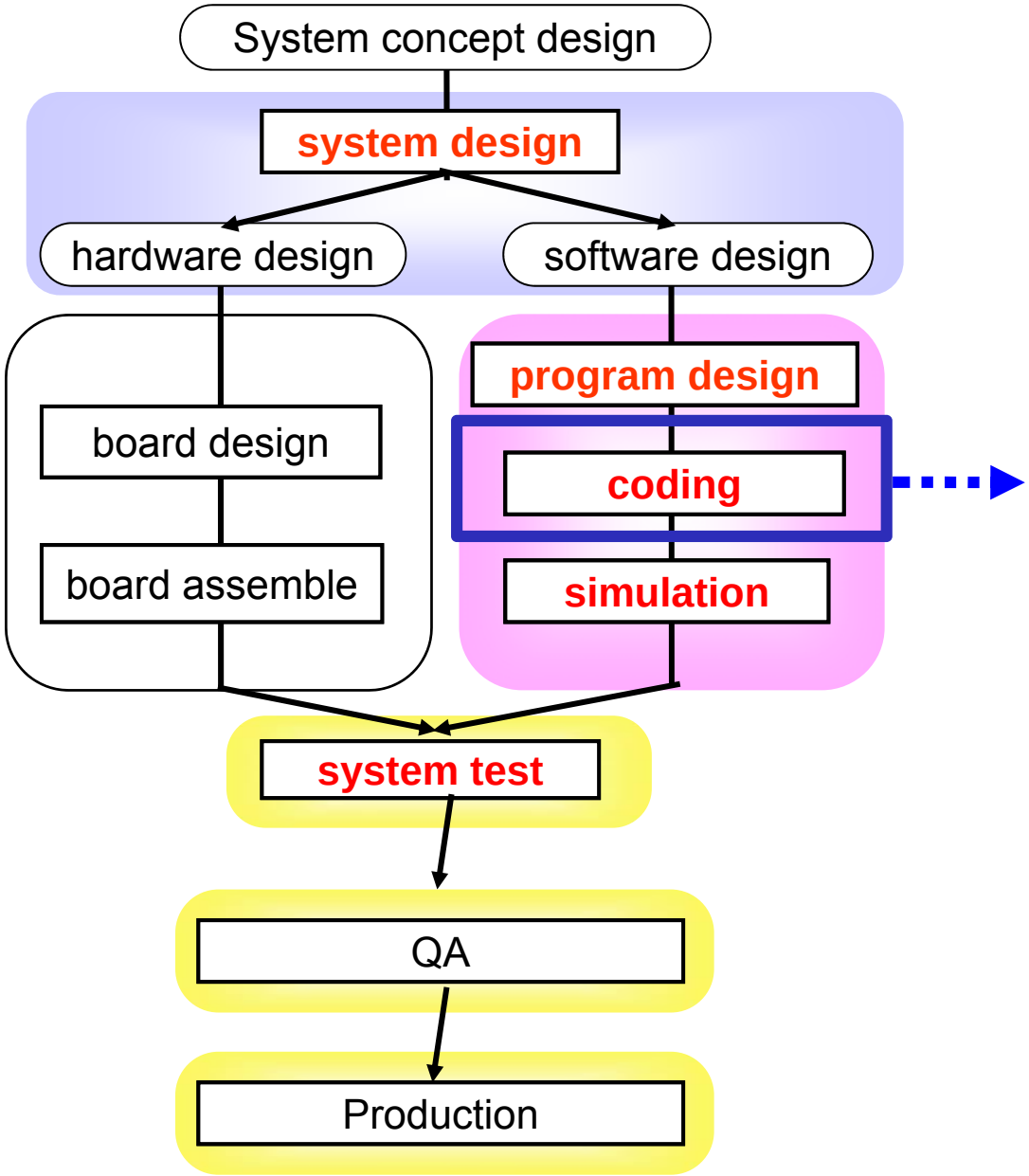
Development tools help your program design



When you start designing the software, it is possible to take proper software sub-component to reduce your software development cost and time, such as commercially available software package, legacy software in your previous product.

- ✓ If you will develop complex and large software system, OS (operating system) helps you to develop complex application quickly.

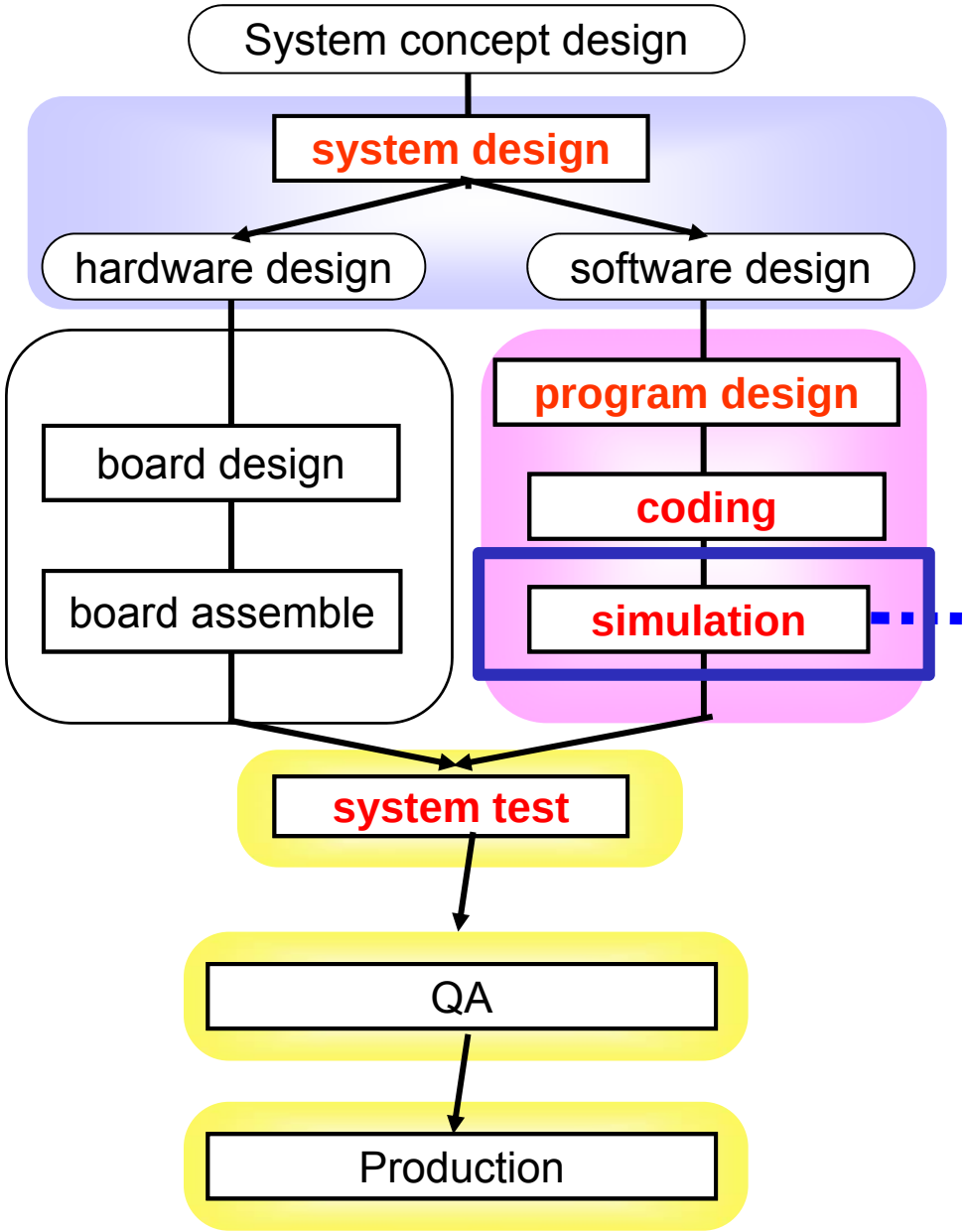
Development tools help your software coding



You have to pay attention to reduce the code size for saving the memory space (in small memory size system), or to higher performance, as hardware resources in embedded system is not rich (to reduce cost).

- ✓ C/C++ compiler provides you the code optimization to generate smaller and/or high performance code.

Development tools help your software simulation



Simulator enables you to run your code (which is compiled for your target CPU, such as H8, M16C, SH-4, etc.) on host PC.

It is better to test/debug your source code on simulator, to get better software quality.

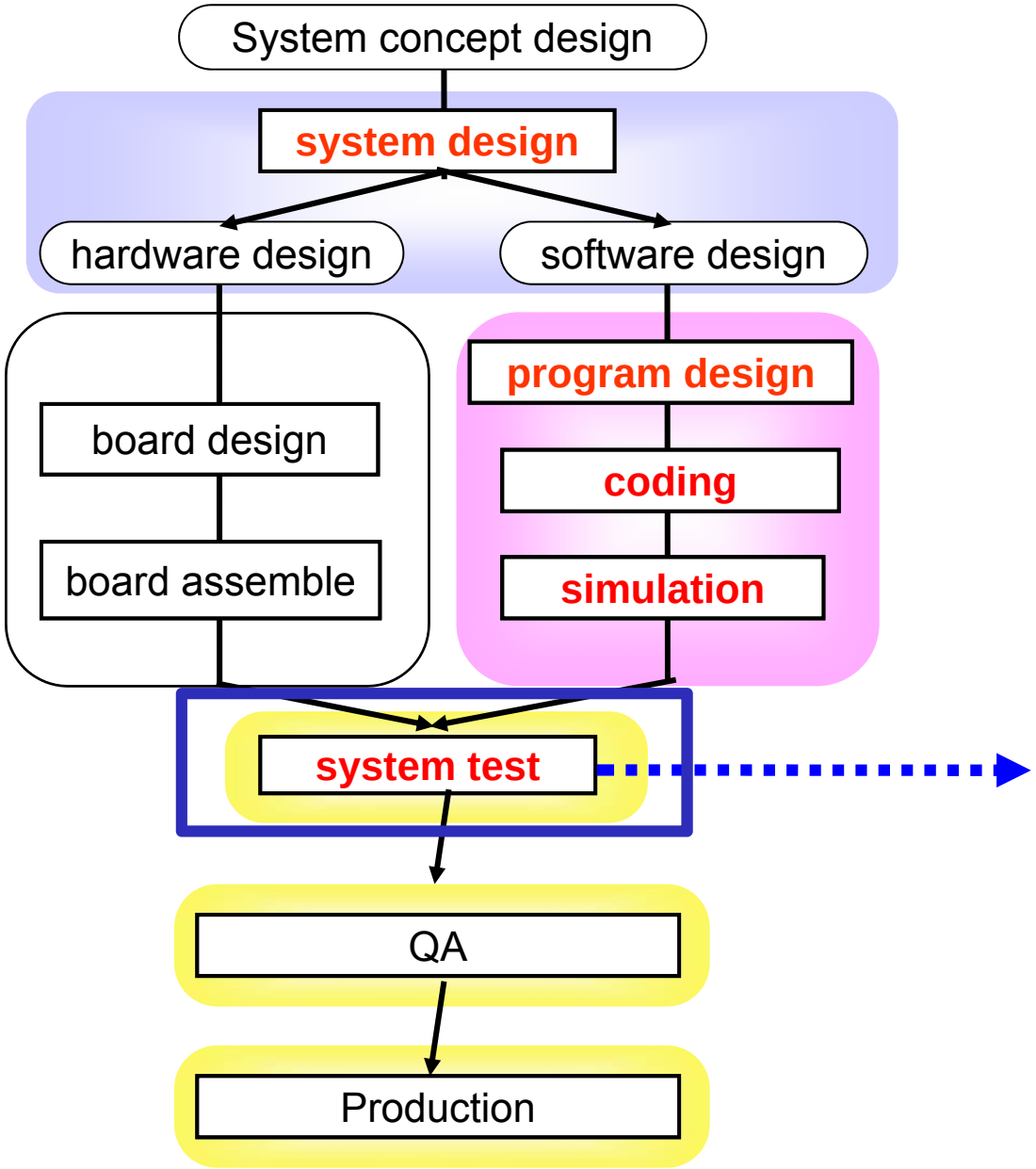
Sometime, your target hardware is not available when you finished your coding, then simulator is only one way to test your code.

✓Software must be tested in smaller unit first.

✓Simulator gives you whole CPU resource view.

✓Cycle accurate simulator gives you the certain software performance.

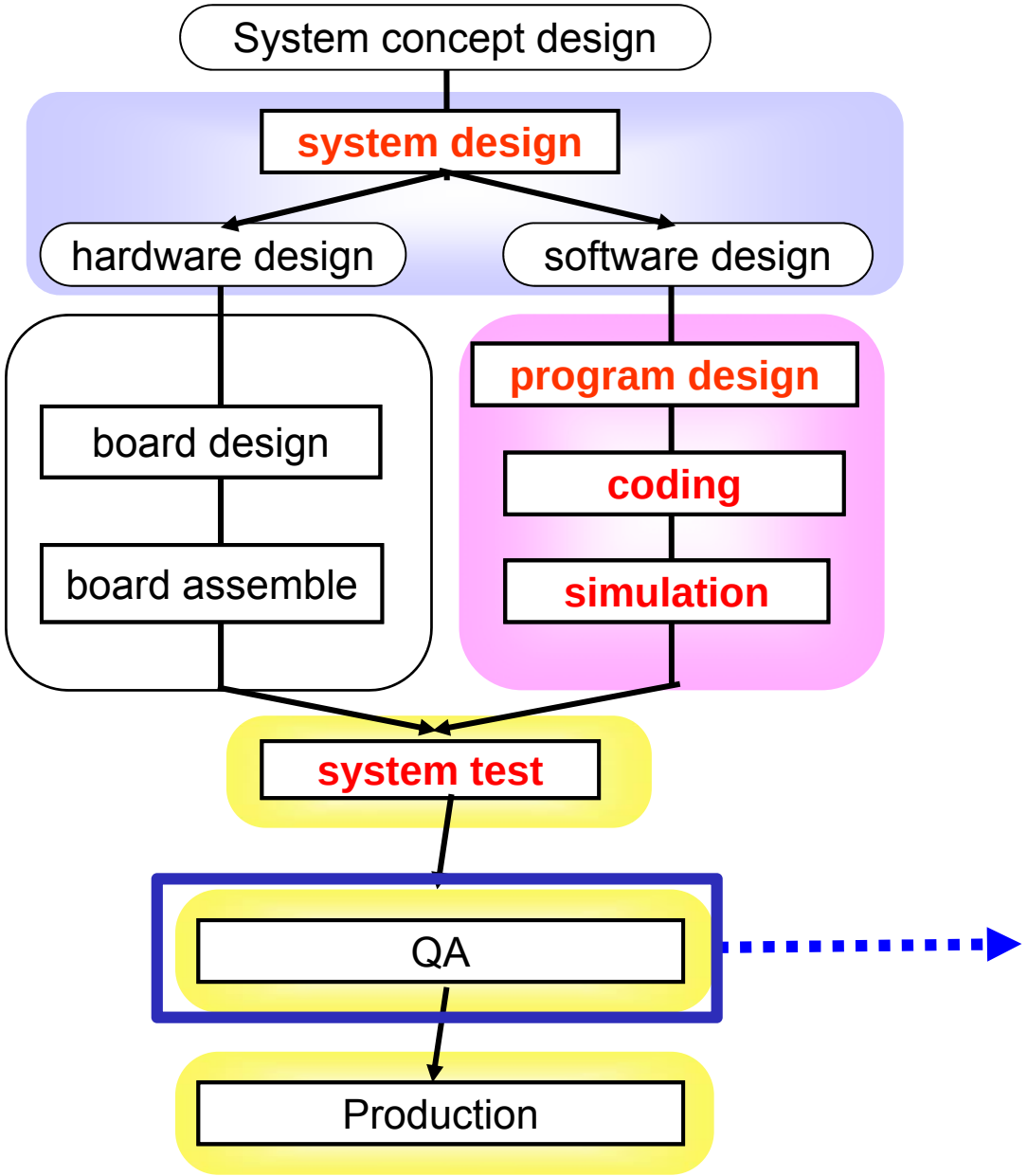
Development tools help your system test



When you finish software debug (and hardware debug), system test (run software on target hardware) must be done in properly.

- ✓ ICE (In-Circuit Emulator) gives you visibility of target hardware internal status, and gives you the control of target hardware.
- ✓ Basic hardware debug must be completed, before starting system test.

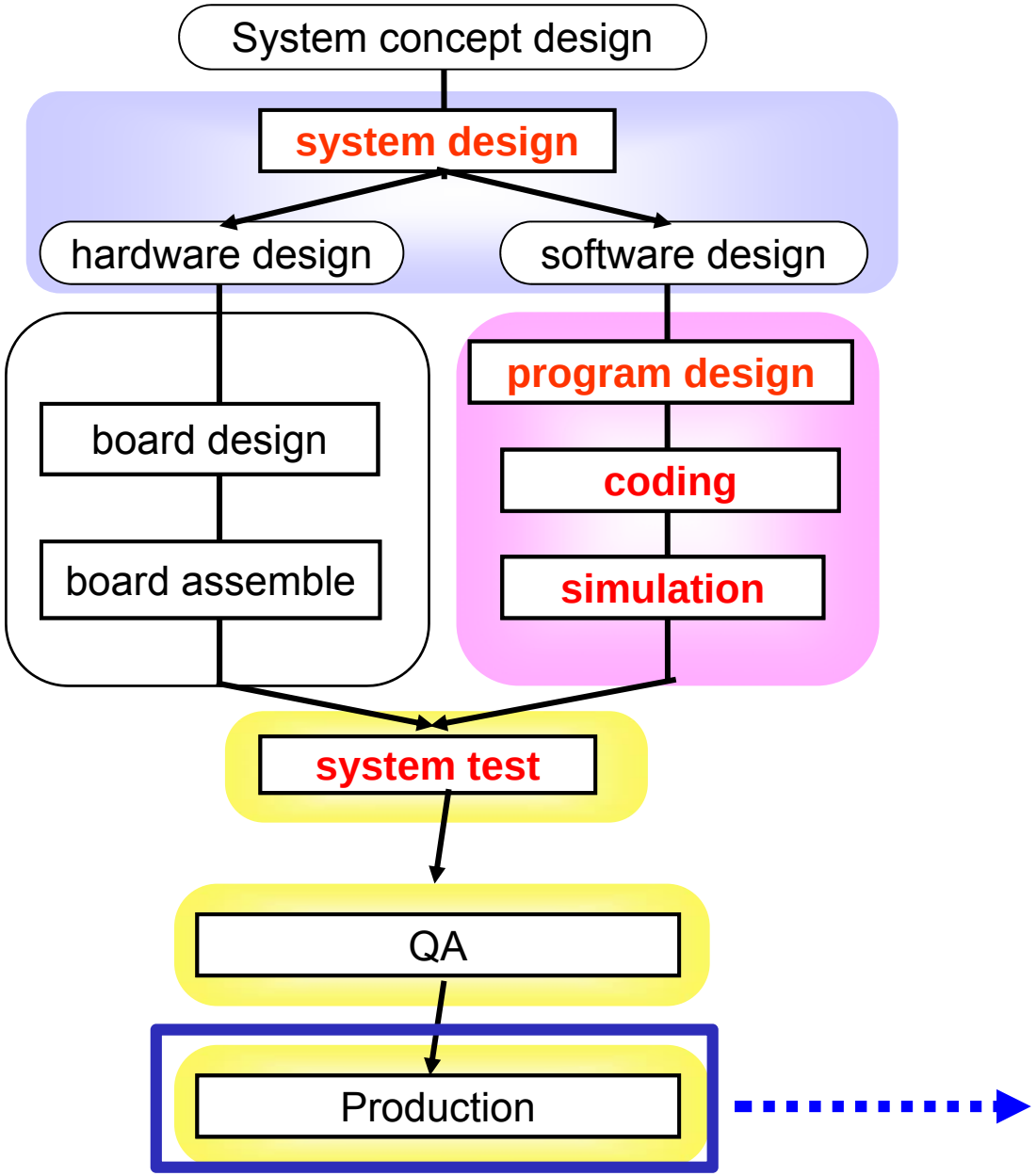
Development tools help your system test



You must define test scenario to get proper quality suitable for production.

- ✓ Code coverage is one of software quality index in test phase.
- ✓ Code coverage information tells you how many instructions are tested (covered) in specific test suites against all instructions in your source code.
- ✓ One important test criteria for high quality software is to achieve 100% coverage.

Development tools for manufacturing



When your product will go into mass production, your code must be stored in target hardware (many number of hardware).
In recent embedded system, most object code is stored in flash memory.

- ✓ Flash programming tool helps you to transfer object code from host PC to target memory (especially for flash memory in MCU) in production.

3. Operating System

Contents

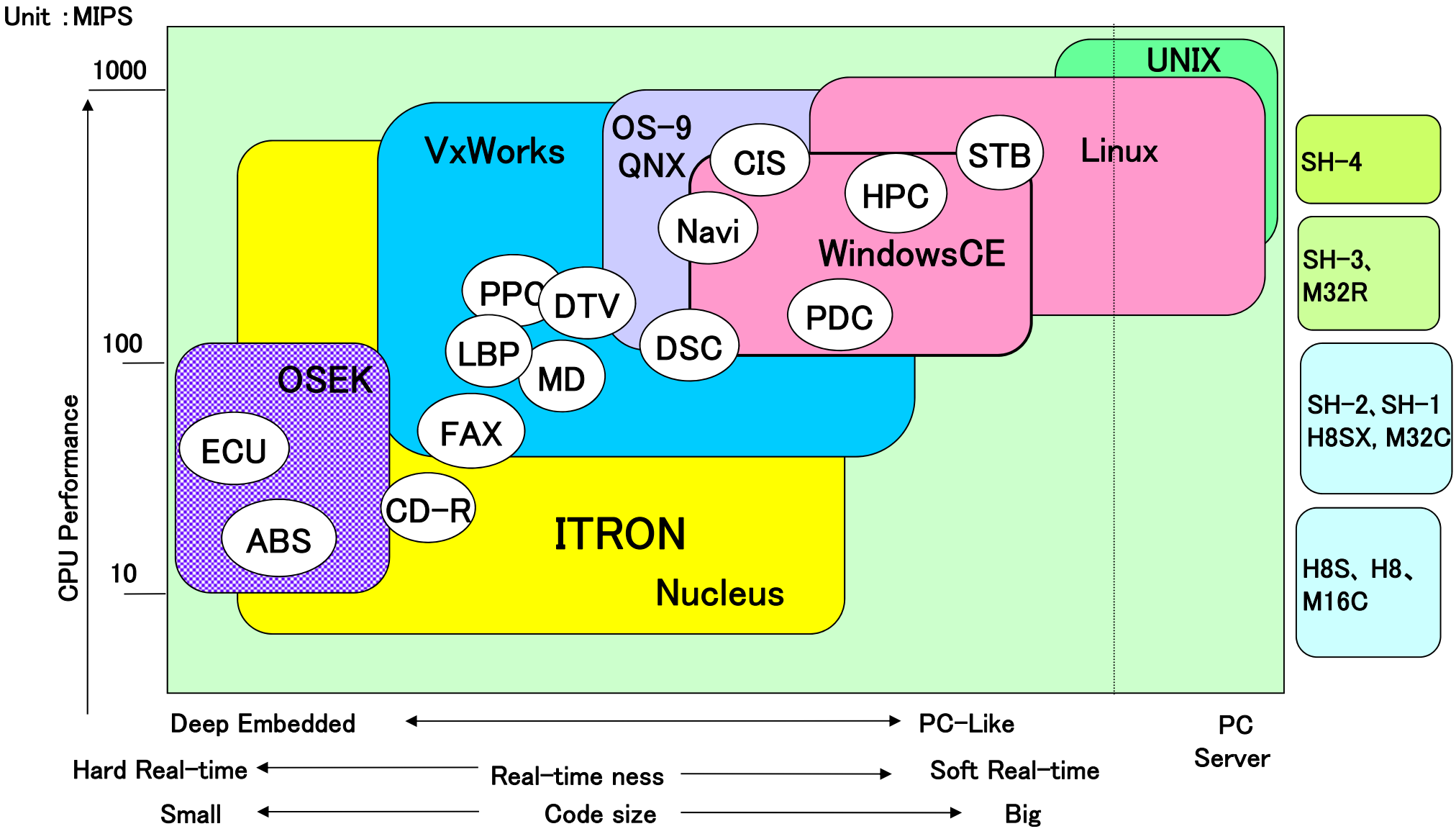
3.1 Embedded Operating System Overview
3.2 Windows Embedded CE
3.3 Linux

3.1 Embedded Operating System Overview

Index

- Embedded Operating Systems for Renesas MCU and MPU
- Reasons for OS selection
- Kernel architecture difference
- Process, task and thread

Embedded Operating Systems for Renesas MCU and MPU



Criteria for OS selection

- **Software richness**
  - How many drivers, middlewares, protocols are provided/supported
- **Foot print**
  - Size of memory.
- **Realtimeness, performance**
  - Response latency against interrupts, speed of task switch (process re-scheduling), etc.
- **Tool chain**
  - Not only compiler/debugger, but also other useful tools, such as system configuration tool, performance analyzer, etc.
- **Robustness and security**
  - MMU protection, kernel/driver stability
  - Secure protocol support
- **Technical support, BSP support, maintenance, update service**
- **Easiness : Easy to port, easy to develop**
- **Price**
- **Legal (GPL, patents, 3rd party intellectual property rights, etc.)**
- **Others : There might be many other points**

What is OS major function

❑ Resource management

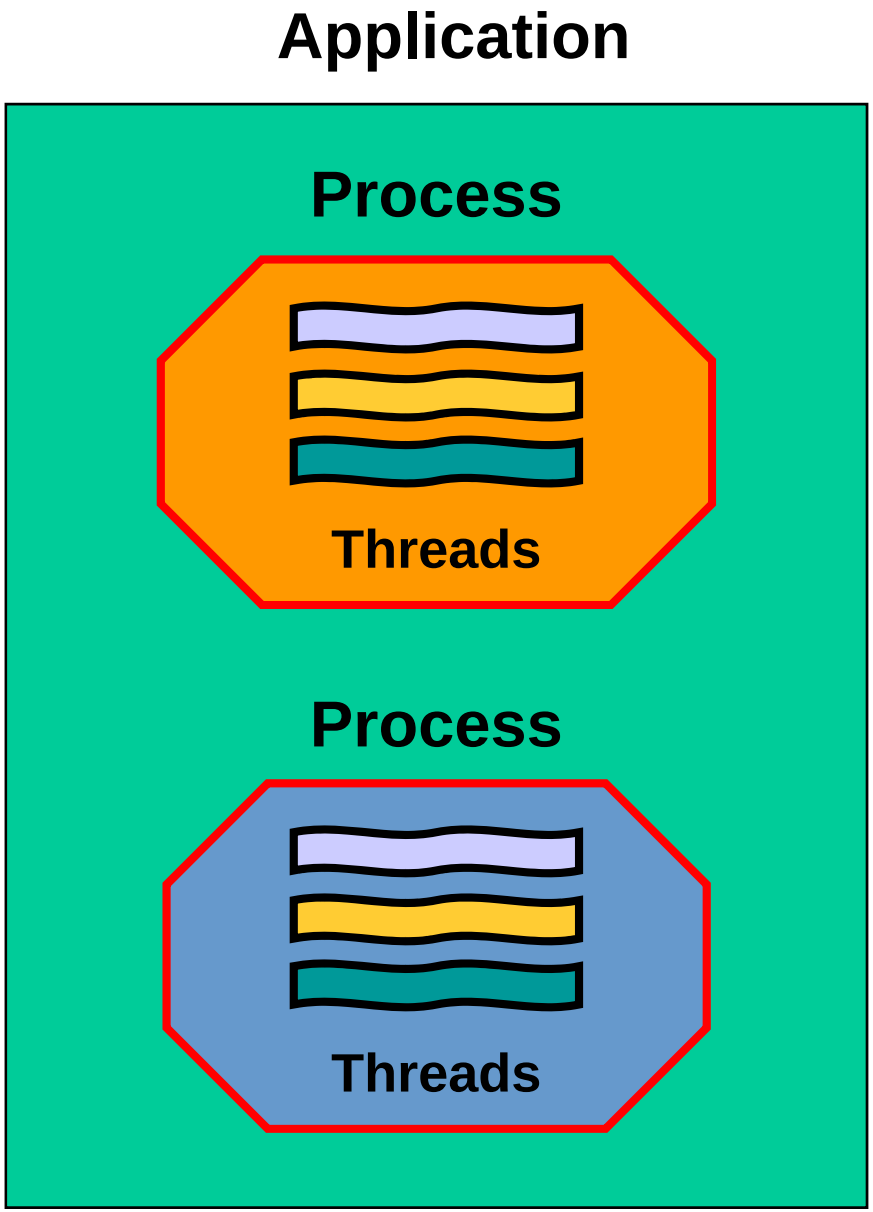
Each process/task uses memory and IO resources independently. OS has to manage resource allocation properly, otherwise resource conflict makes fatal error on the system

❑ Scheduling

Each process/task must be executed in proper order, to achieve low cost/high performance embedded system.

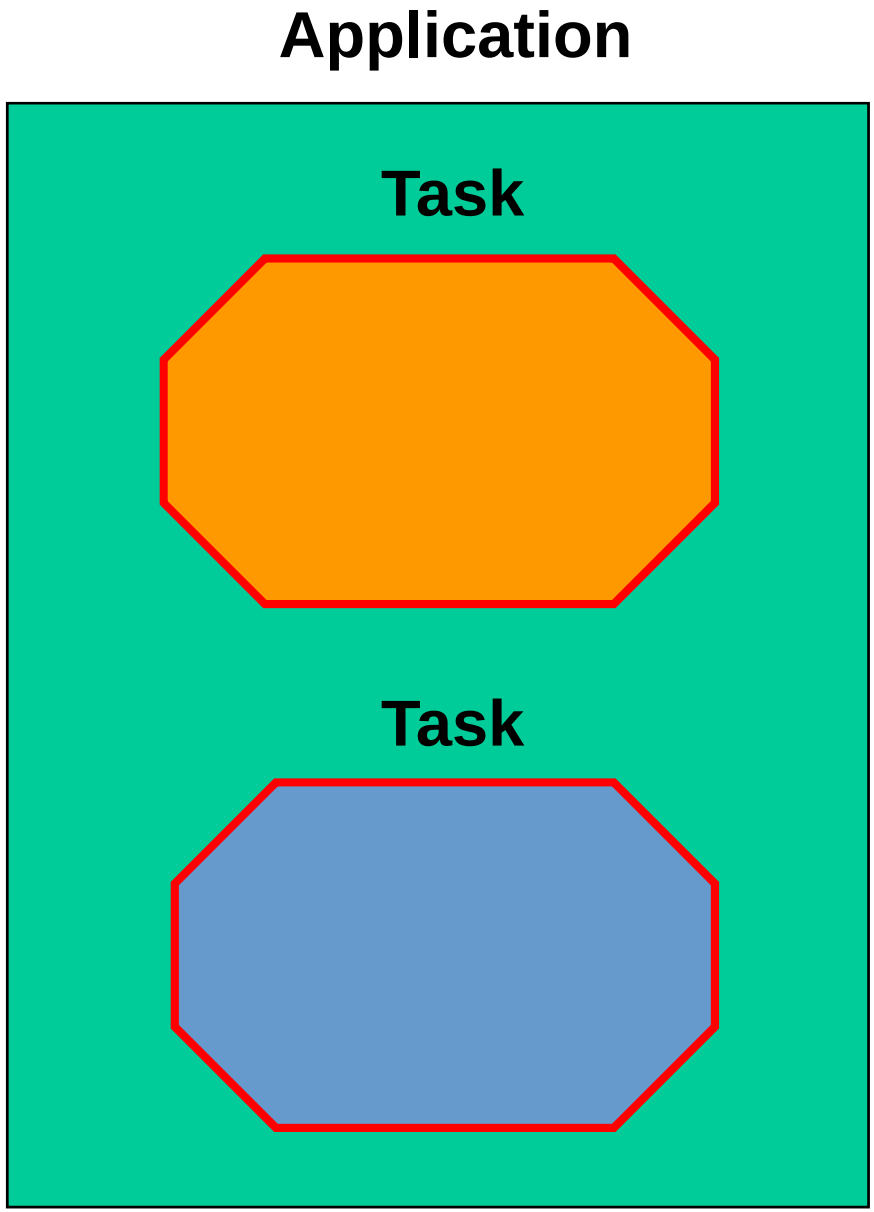
Process and thread

- Application is divided into memory protected units called processes
- Process is further divided into internal, schedulable units called threads
- Threads share all of the same resources (memory space included)



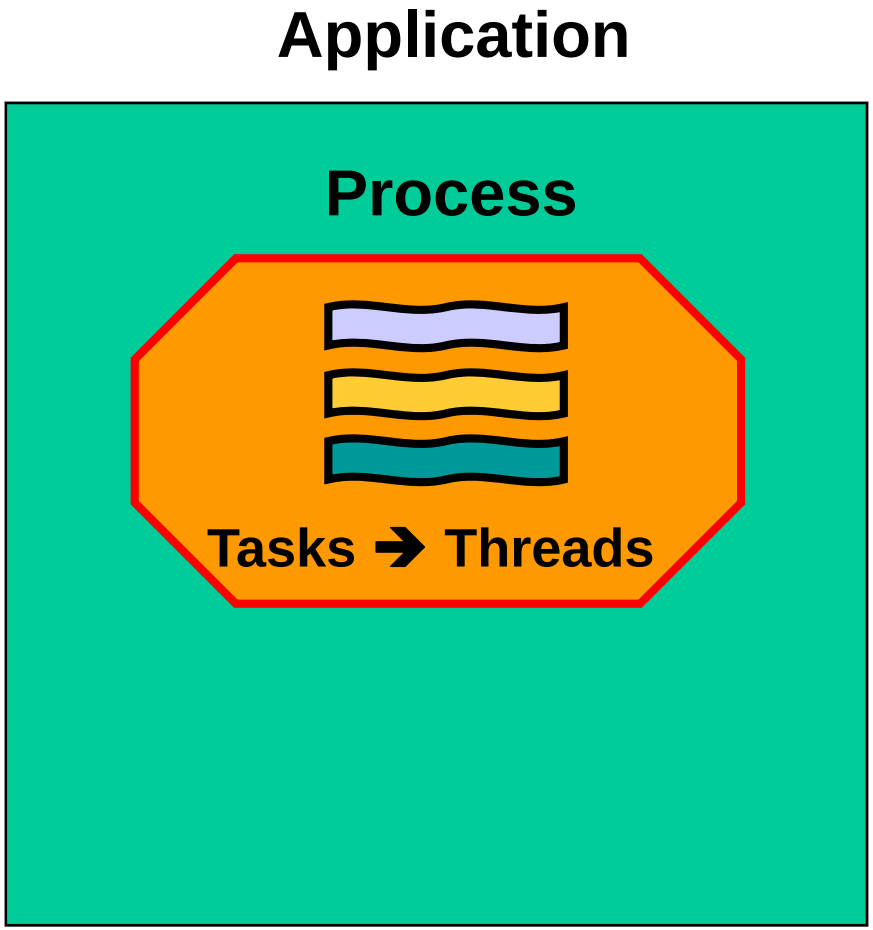
Task in real-time kernel

- **Application is divided into non memory protected units called task**
- **Tasks share all of the same resources (memory space included)**



Process and thread

- **Application porting from real-time kernel based system to Linux based one, such as ITRON → Linux**
  - To share resources, each task is mapped as thread
  - One process in one application
  - Application itself may run as designer expected, but this implementation is not based on original process idea (Unix POSIX interface)
  - Performance (or other) problem may occur on this kind of migration



Local variable and global variable

Local variable

- Locates on stack
- Unique address in specific function
- Allocated at function start, released at function end
- Same variable (variable name) can be used in different function (as different variable)
- Access beyond function is not possible (not allowed)

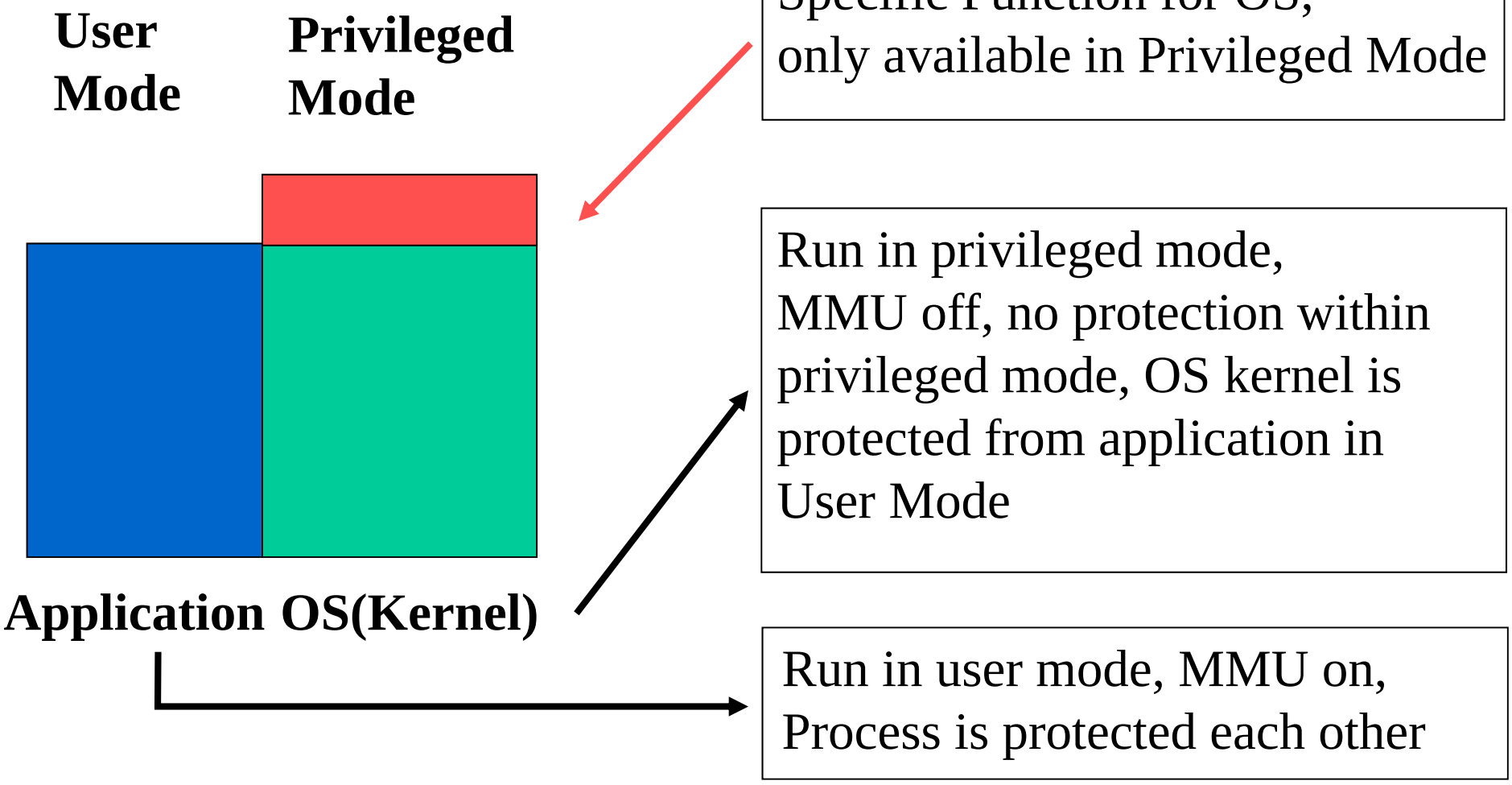
Local variable and global variable

Global variable

- Locates on specific address through all function
- Common address between function
- Same variable (variable name) can be used in different function (as same variable)
- Access beyond function is possible (allowed)

Programming model for High-end MPU

Programming model for SH-4, generic



MMU

MMU (with OS) provides memory protection and dynamic loading by using following function

- Address translation, from logical (virtual) address to physical address
  → dynamic loading, physical address extension beyond 32 bit (32 bit CPU)
- Address translation is done in small address size unit called page, typical page size is 4 kByte/page
- Memory access type setting for page entry, RO/RW/EX, PV/US
  → protect the access beyond process,
  protect the kernel from illegal access caused by application

RO : Read Only, RW : Read and Write, EX : Execute only
PV : Privileged mode, US : User mode

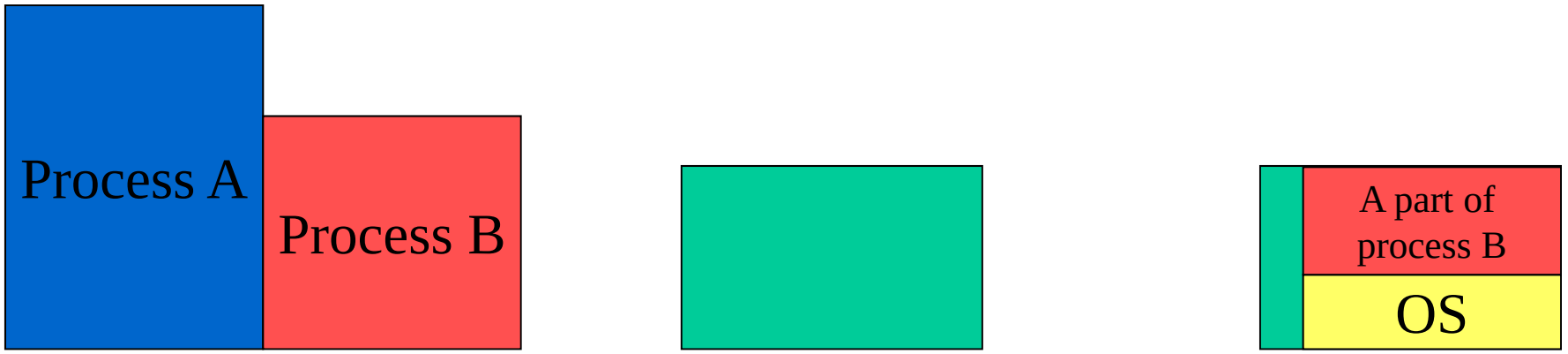
Example of memory mapping

Traditional model
(memory is smaller than process in application)

Application (Logical Address)

Memory (Physical address)

Application allocation in memory



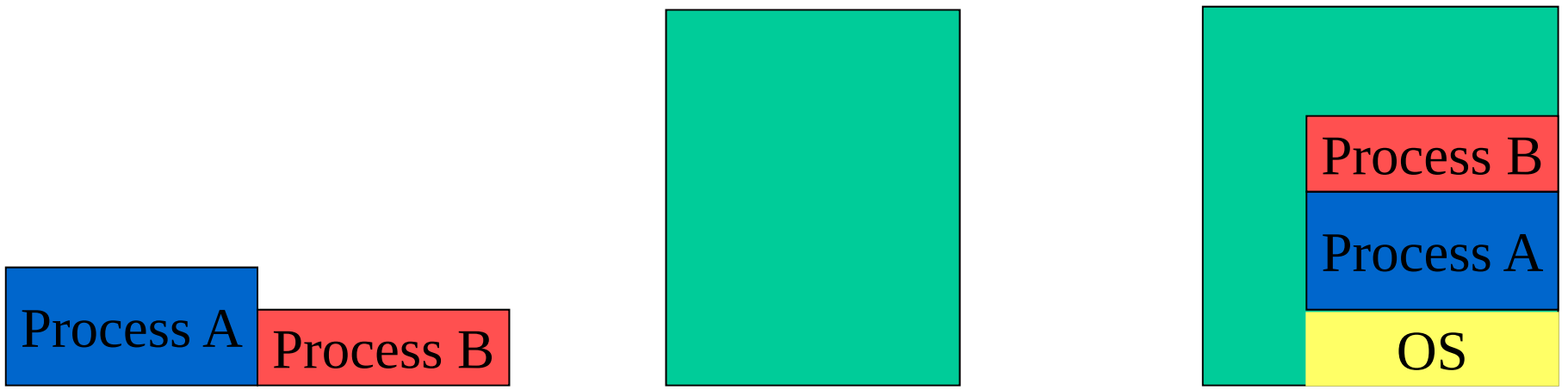
Example of memory mapping

Memory model in modern embedded RTOS
(memory is large enough to place multiple process)

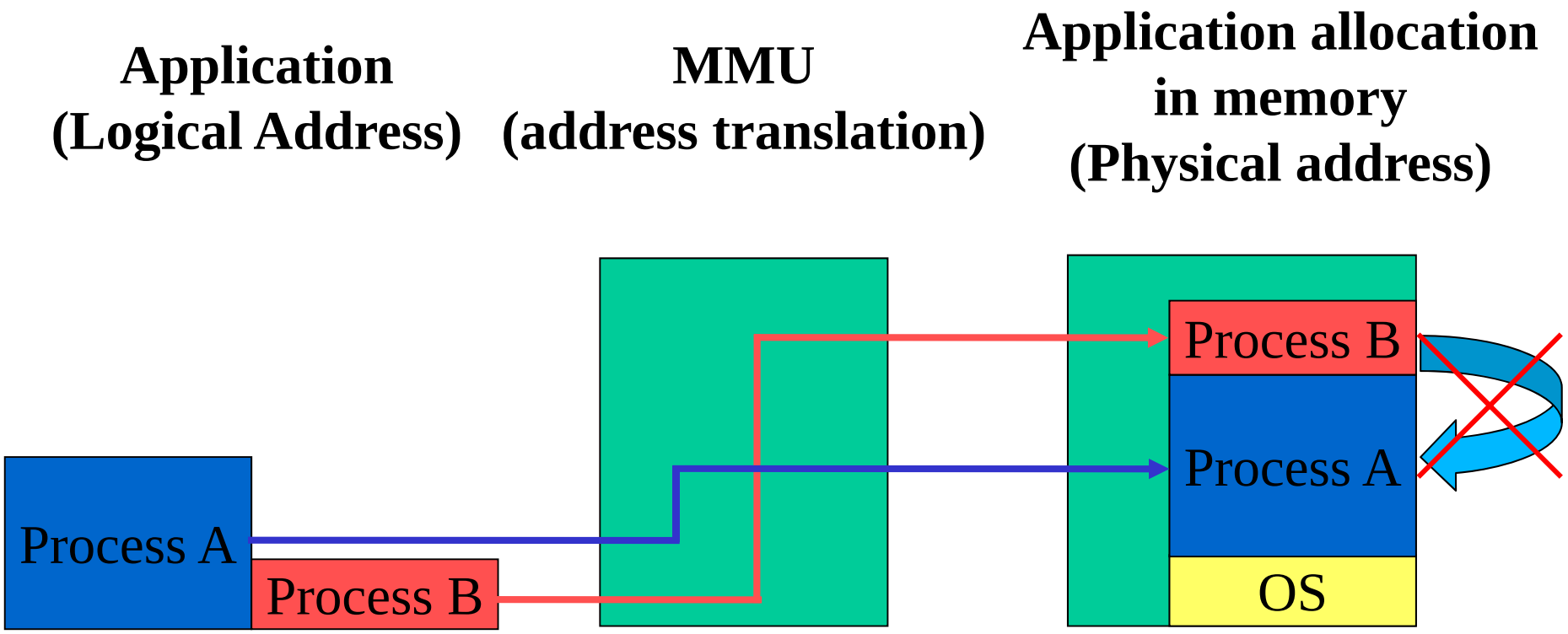
Application (Logical Address)

Memory (Physical address)

Application allocation in memory



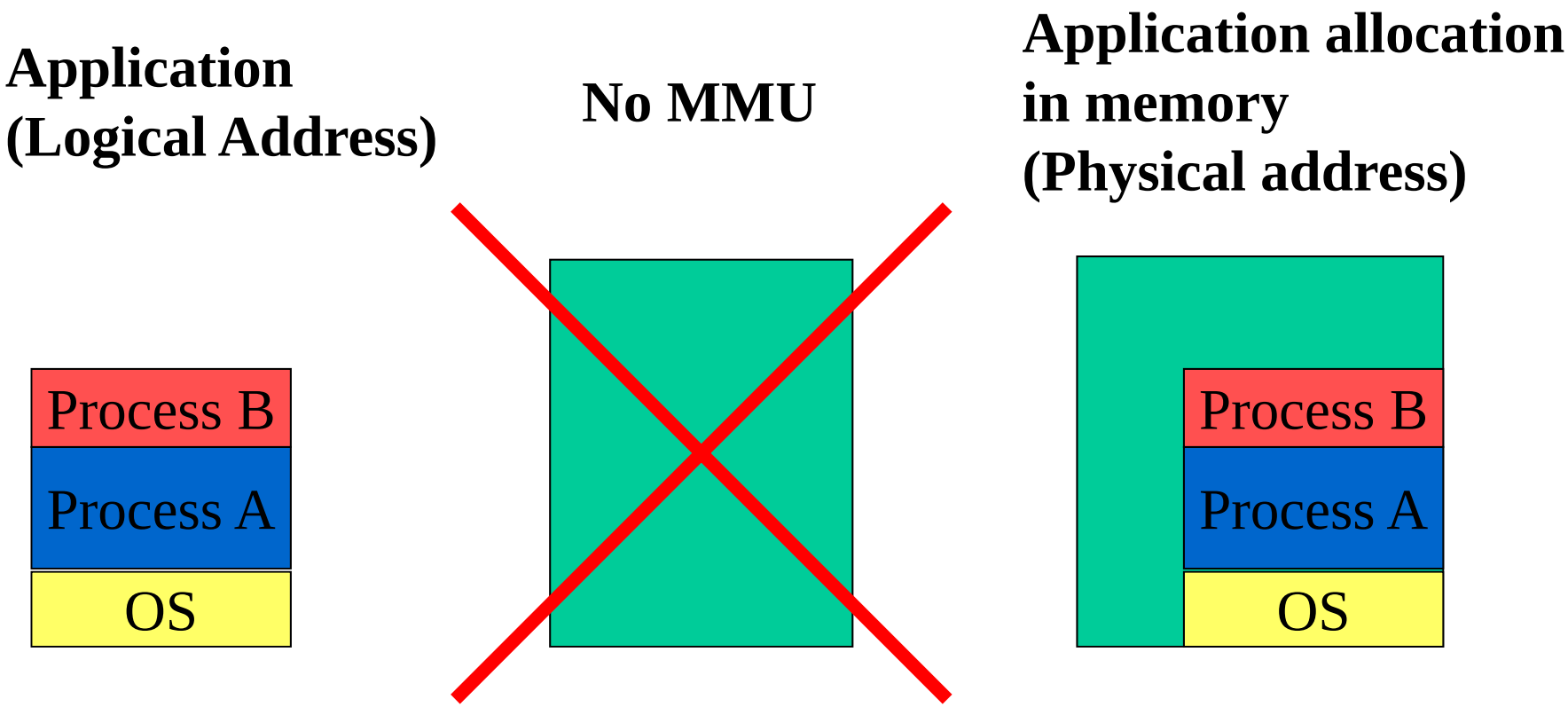
Address translation by MMU



Access beyond process is NOT allowed

Address allocation in non-MMU device

Application software engineer have to decide proper physical address for each application and OS, when he/her compile and link the program. Logical address must be same as physical address.



Dynamic Link and Static Link

❑ OS (Device) with MMU

- o Application software engineer does not need to think about physical address. Just compile, link and go.
- o OS takes care of physical address resolution. OS places each process into proper physical address through MMU. → Dynamic link

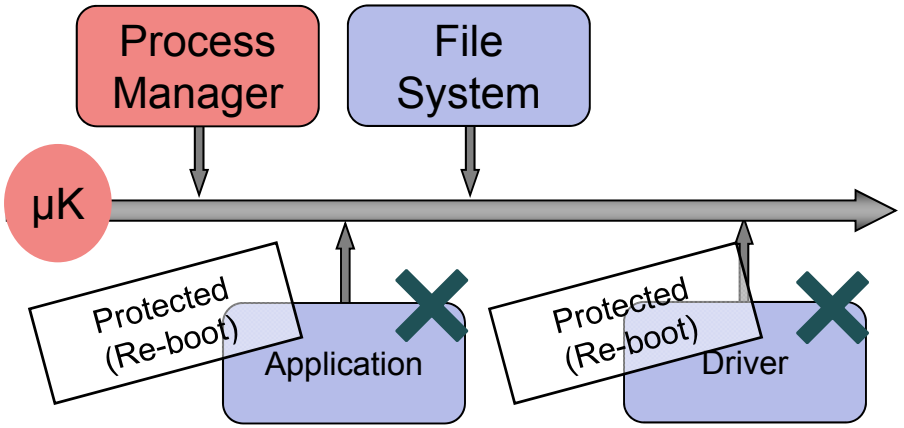
❑ OS (Device) without MMU

- o Application software engineer have to think about physical address for his/her application.
- o Application software engineer have to specify correct physical address on compile and link. → Static link

Kernel architecture example

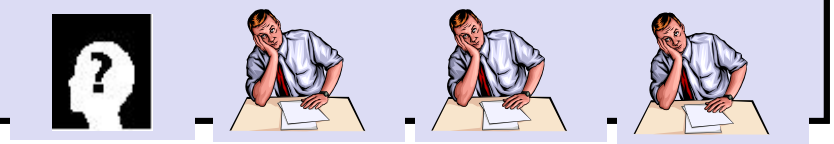
Real-time kernel with MMU protection

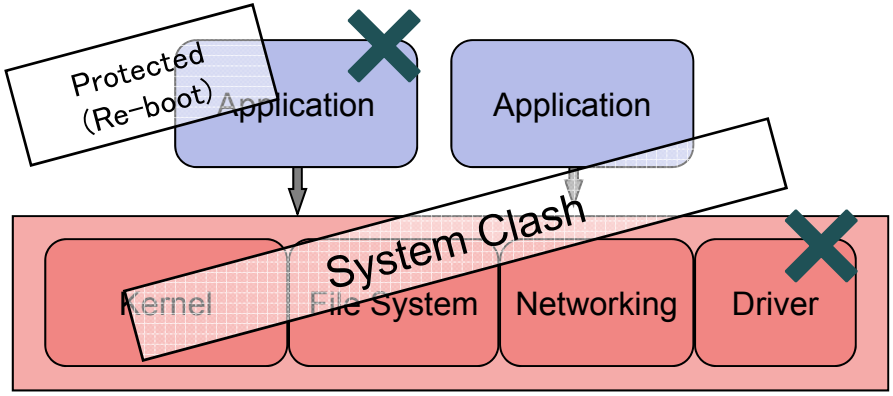
- Application, Driver, protocols are protected



Monolithic kernel (NT / Unix / Linux)

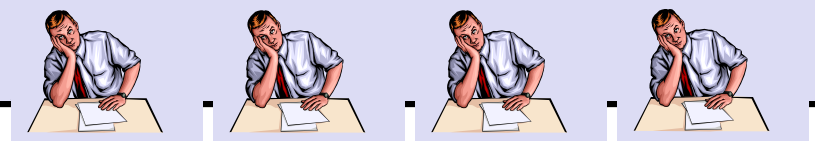
- Partial MMU protection
- Application is protected

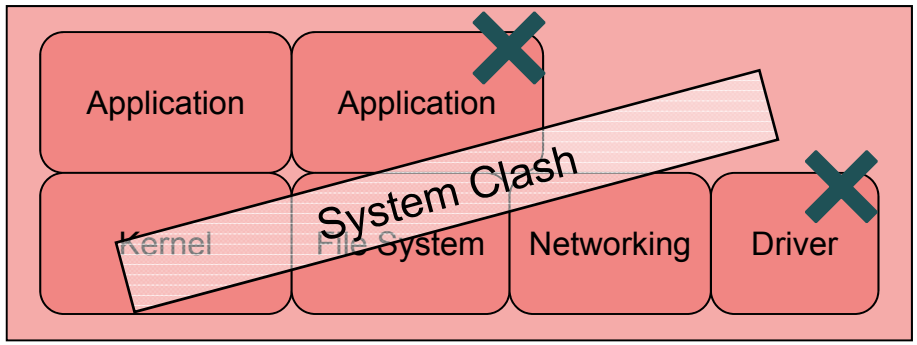




Real-time kernel without MMU protection

- No MMU protection
- Application, driver, protocols are in kernel space





OS comparison

- **Compact real-time kernel (real-time OS) vs. OS (with MMU protection)**
  - Small size vs. larger size
  - No MMU protection vs. MMU protection
  - Fast vs. slow
  - Poor driver/protocols vs. rich driver/protocols
  - Lower price vs. higher price
  - Reliability, stability is also need to be considered
  - OS (or kernel) specification is different each by each, above classification is example for study purpose.

3.2 Windows CE

Index

- History
- Windows Embedded CE 6.0 overview
- Development Environment and debug

Windows CE History
- Windows CE, core OS version -

• Windows CE 1.0	1996.11	Pegasus
• Windows CE 2.0	1997. 9	Alder
• Windows CE 3.0	2000. 4	Cedar
• Windows CE .NET 4.0	2002. 1	Talisker
• Windows CE .NET 5.0	2004. 7	Macallan
• Windows Embedded CE 6.0	2006. 9	Yamazaki
• …		

出典:「Windows Embedded CE 6.0 組み込みOS構築技法入門」
日経BPソフトプレス社刊

Windows CE
- Application specific package -

❑ Windows Automotive (WA)V. 5.0 / V5.5 (2008)

Developed by Microsoft Japan\
Widely used in Japan, and Asia

❑ Microsoft Auto (MS Auto) V. 3.0 (2008) / V. 4.0 (2009)

Developed by Microsoft US\
Used in Europe, North America

❑ New OS for Automotive (2010)

Based on WinCE V7 (SMP capable)\
Merged release of MS Auto and WA\
Multi-core (SMP) and single-core support\
Silverlight technology based new graphics system and tools\
Automotive System Tool (Readyguard, Snap Shot Boot, etc.)\
Mobile and multimedia capability inherited from MS Auto

Windows Embedded CE 6.0
- Feature -

- Win32 API
- New kernel : up to 2GB address space, up to 32000 process
  - High performance
  - High Reliability
- Supports 4 CPU architectures : x86, SH-4, ARM, MiPS
- Microsoft quality
- Network media device support
- DVR Engine : multiple digital video stream support

MiPS: Microprocessor without Interlocked Pipeline Stages

出典:「Windows Embedded CE 6.0 組み込みOS構築技法入門」
日経BPソフトプレス社刊

Windows Embedded CE 6.0
- Development environment -

- Host Machine : Windows 2000, Windows XP, or Windows Vista
- Microsoft Visual Studio 2005
- Windows Embedded CE 6.0 Platform Builder
- Additional software
  - Install BSP, etc., for specific device/platform support

出典:「Windows Embedded CE 6.0 組み込みOS構築技法入門」
日経BPソフトプレス社刊

Windows Embedded CE 6.0
- Debug -

- KITL (Kernel Independent Transport Layer)
  - Connection interface between target hardware and Host PC
    (Platform Builder)
  - Serial or ethernet connection
- Debug function on Platform Builder
  - Kernel debugger
  - Remote tool
  - Target control (command line for debug)
- eXDI (eXtended Debug Interface)
  - hardware assist (by ICE) debug function

出典:「Windows Embedded CE 6.0 組み込みOS構築技法入門」
日経BPソフトプレス社刊

Index

- Why Linux? What is Linux?
- Linux vs. embedded OS
- Porting Linux from x86 to SH
- Linux package
- Open source vs. commercial distribution
- Embedded Linux
- GPL
  - What is ‘free’?
  - What is ‘open source’?
  - What is ‘as is’?

Why Linux?

- Network, file system are supported
- High compatibility between CPUs
- Many peripherals, protocols are supported
- Many existing software source code files are available
- Relatively easy to build prototype system
- Open source
- Free of charge

What is Linux?

The word of ‘Linux’ has several different meanings

- Linux OS kernel, just kernel only
- Linux OS kernel binary, including drivers, etc.
- Linux command, such as bash
- Open source applications, such as samba, apache, running on Linux
- Gnu development environment for Linux
- Combination of above

Linux vs. traditional embedded RTOS

Pros

- Wide range source code availability
  Everyone has access to whole source code through internet
- Copy left : everyone can use it, no limitation for using Linux
- Free of charge (or seems to be free of charge)

Cons

- See next page

Linux vs. traditional embedded RTOS

Cons

- Kernel structure design concept, best match for PC, but…
  - Linux is TSS (Time Sharing System) based architecture
  - Slow latency against interrupts, process/task switch, etc.
  - Designed for x86 based PC architecture, not for embedded application
- Specification change / incompatibility
  - Linux specification is changing rapidly. Upward compatibility is not guaranteed. I.e., if you port some software onto another kernel, you must test it. Compatibility (that software should work on another kernel) is not guaranteed.
- Quality, guarantee
  - Linux quality is getting better, but no one provides formal guarantee
- GPL

Linux package components

- **Kernel (vmlinux)**
  - Driver binary is statically linked to kernel
  - Running in privileged mode on SuperH architecture (SH-4)
- **RootFS (Root File System, or called 'User Land')**
  - On PC Linux, rootFS is placed on HDD.
    In embedded application, flash file system is popular than HDD file system
  - Running in user mode on SuperH architecture (SH-4)
- **Tool chain (gcc, binutils, glibc, etc.)**
  - Cross (on x86 Linux) and native (on SH-4 Linux) tool chain are available.
  - Note that some gcc distribution includes newlib, not glibc, such as KPIT SH GCC.
- **IPL (Initial Program Loader)**
  - Boot up, initialization of the system, loading kernel, etc.

Open source vs. commercial distribution

- **Open source (kernel.org, fsf.org, etc.)**
  - Free of charge
  - No warranty in any kind, user's own risk
- **Commercial distribution (Red Hat Enterprise Linux, Montavista Linux, etc.)**
  - NOT free of charge
  - Limited warranty (depends on contract/price)
  - Bug fix, update, consulting services are available

GNU GPL (GNU General Public License)

- Most of Linux software is covered by GNU GPL (GPL v2/v3) license
- As Linux kernel binary file includes driver binaries, driver is also covered by GPL, even if the driver is coded from scratch by yourself.
- Please refer original GPL carefully at http://www.gnu.org/copyleft/gpl.html, when you are in charge for Linux based software development

What is ‘free’ in GPL

- **Free is not only price in common understanding**
- **But also**
  - Freedom of use
  - Freedom of distribution
  - Freedom of source code modification
- **You have a choice to charge for your Linux based software package**

What is ‘free’ in GPL?

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

What is ‘open source’ in GPL

Not necessary to open ‘source code’ on web site

- When you are using object code (modified by yourself) for your private use only, you do not need to open your source code

- When you provide object code (modified by yourself) to limited person, such as Renesas employee, you need to open source code to the person who received object code. Not necessary to open source code to other person.

What is ‘open source’ in GPL

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

What is ‘as is’ and ‘warranty’ in GPL

- **Program is provided ‘as is’ based, without any kind of warranty.**
  - No guarantee / commitment to achieve any kind of functionality / performance
- **Entire risk is with you : the person who use Linux.**
- **Copyright holder nor any person (including the person who distributes Linux) never take any kind of responsibility.**
  - Not necessary to fix bug / modify source code
  - Not necessary to investigate any problem
  - Not necessary to answer against any question
  - All re-action is based on voluntary

What is ‘as is’ and ‘warranty’ in GPL

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

4. Real-time Operating System (Real-time multi-task kernel)

4.1 Real-time kernel vs. OS

Real-time kernel (or Real-time Operating System)

- mainly used for 16/32 bit MCU, such as M16C, H8/H8S, SH-2(2A)
- compact, smaller object size
- relatively fast
- No MMU support
- simple configuration
- static link

OS

- mainly used for 32 bit MPU, such as SH-4(4A)
- rich function support, larger object size
- relatively slow
- MMU support
- rich software sub-component support
- dynamic link

Note: above classification is example for study purpose, each OS/kernel has different function/specification/feature.

4.2 Real-time performance and multi-task control

Realtime performance

- Follow the constantly changing world we live in and control it with computer

PC without realtime performance

Need to wait for PC to ready to use for a few minutes after starting-up.

>>User has to wait for the computer getting the process done.

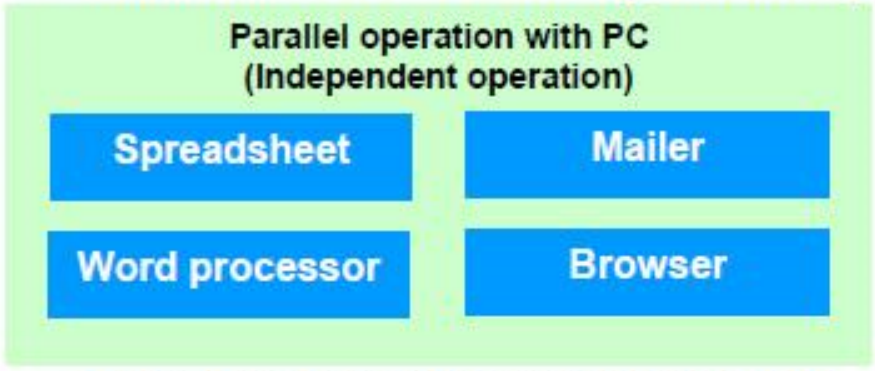
DVC with realtime performance

You can use immediatery after power-on. Video tape recording starts just from the moment you click the record button. The picture and voice are synchronized with.

Task

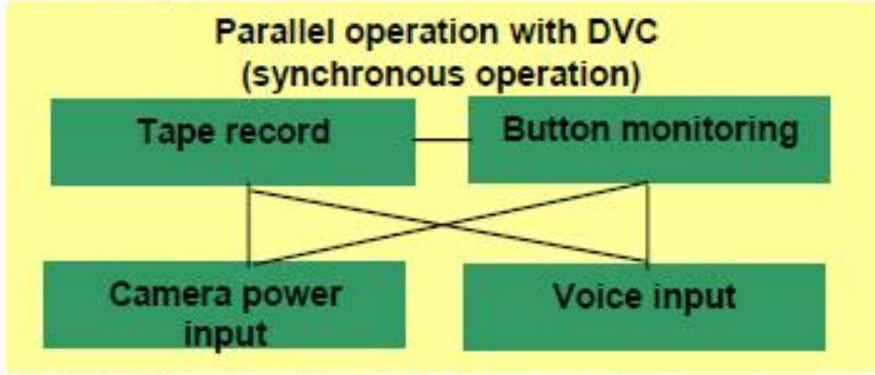
A task (process) is a execution unit.

Multi-task means executing the multiple tasks in parallel.



Tasks operate one by one independently

(Each task has its own task)



Multi-task operates synchronously

(Embedded system need this synchronous operation)

4.3 Real time performance

Value of performance

- On PC, performance means ‘total data through put’, i.e. performance means ‘how many data calculation/processing is performed in specific time period’. ‘Time period’ is longer than embedded system, 1 to some sec or more.
- On embedded system, ‘real-time performance (or response)’ is more important than ‘data processing performance’, i.e. how long (second, or microsecond) it takes from (against) specific input, such as ‘from power on to system ready’, ‘from key tough to reaction(from power off key tough to power down)’, etc. Specific ‘job’ must be finished within proper time period.

- Real-time performance depends on latency against interrupts

  Old system (without interrupts) :

  Key scan (polling) → “if key in = ‘y’” → specific job → return

  OS(kernel) less system :

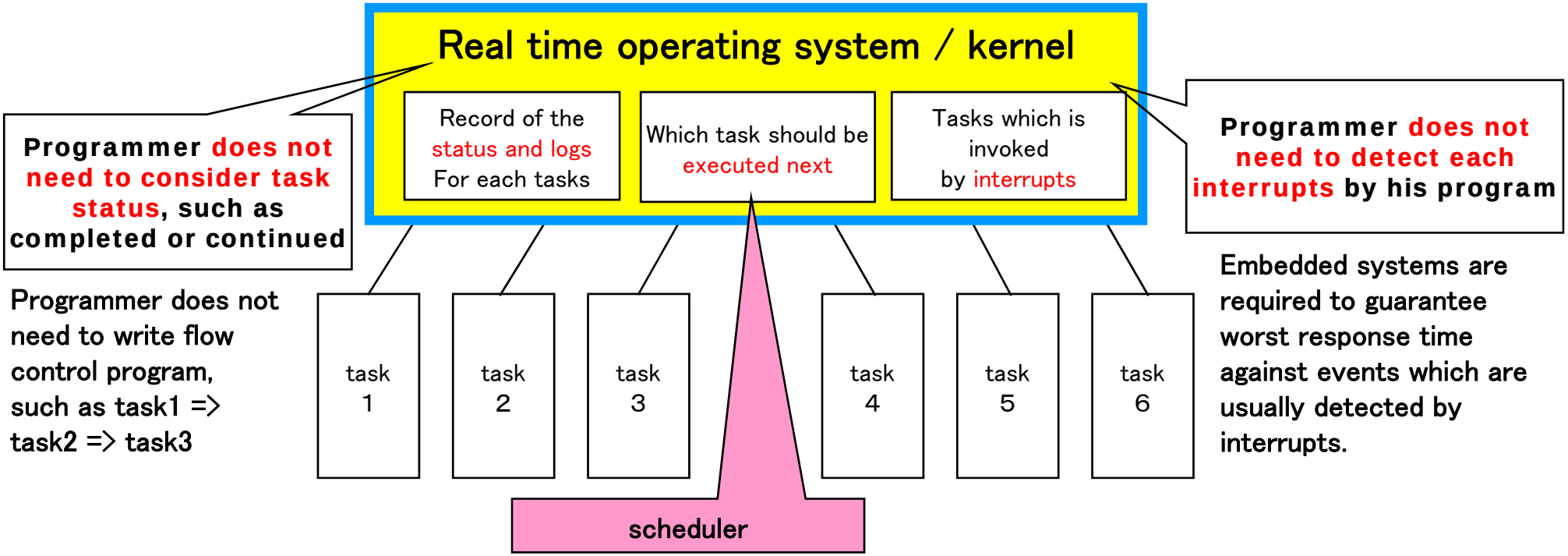
  Wait interrupts (waiting software loop) → interrupts → specific job → wait

  OS (kernel) based system :

  Interrupts → interrupt handler → change status (request) table → task re-scheduling → specific ‘task’ runs

4.4 Parallel processing under embedded real-time OS, kernel

- Need to register software groups (named 'task' which can be executed independently) to OS table
- Events (key input, timer overflow, etc.) generates interrupts and invokes related tasks

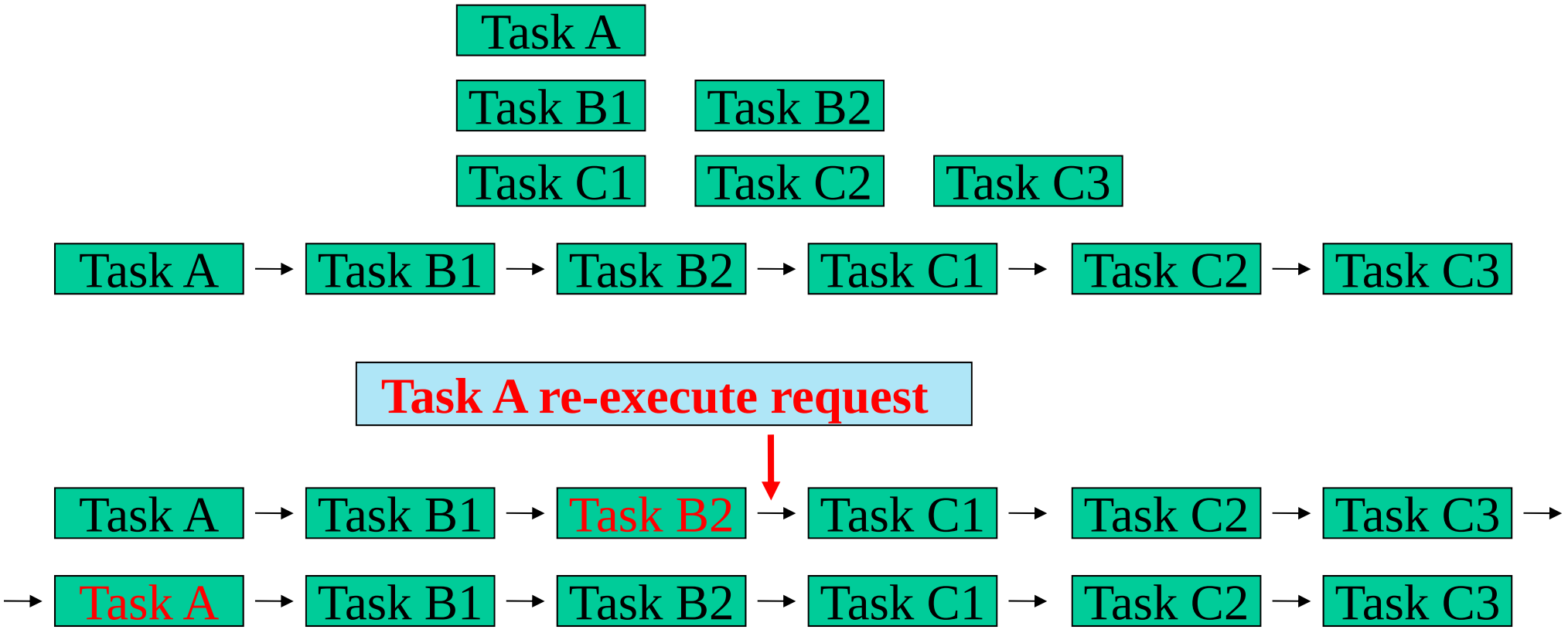


Application program is divided into several small groups which is called 'task'.
Operating system provides task control function, i.e. CPU power is time sliced and in each small time slice, task is executed under pre-defined order.
There are two major algorithm for time slice control:

- Time shearing system is popular in PC application.
- Priority control system is popular in embedded application.

4.5 Scheduling: Priority and Round-robin

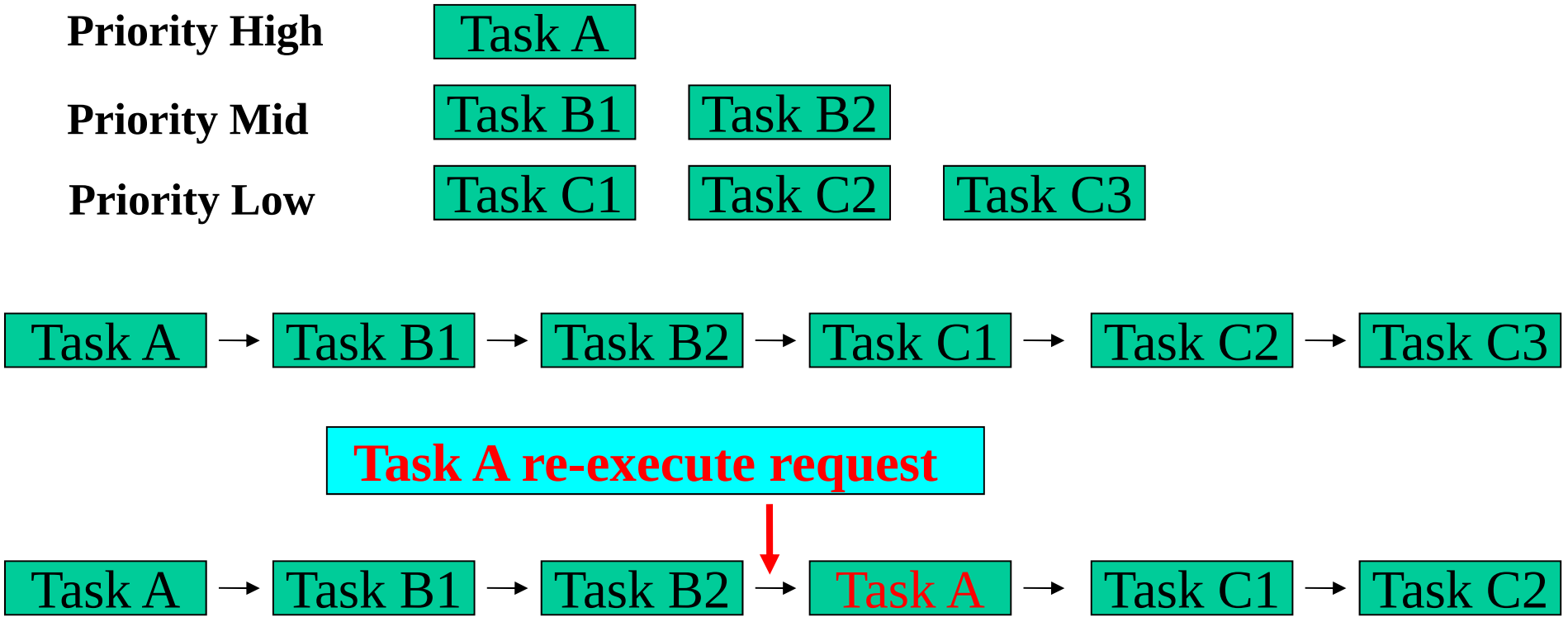
Round-robin scheduling : Common algorithm in computer system (or TSS: Time Sharing System)



Every task will be executed in long term time period, but response time (from request to complete) is longer than priority control.

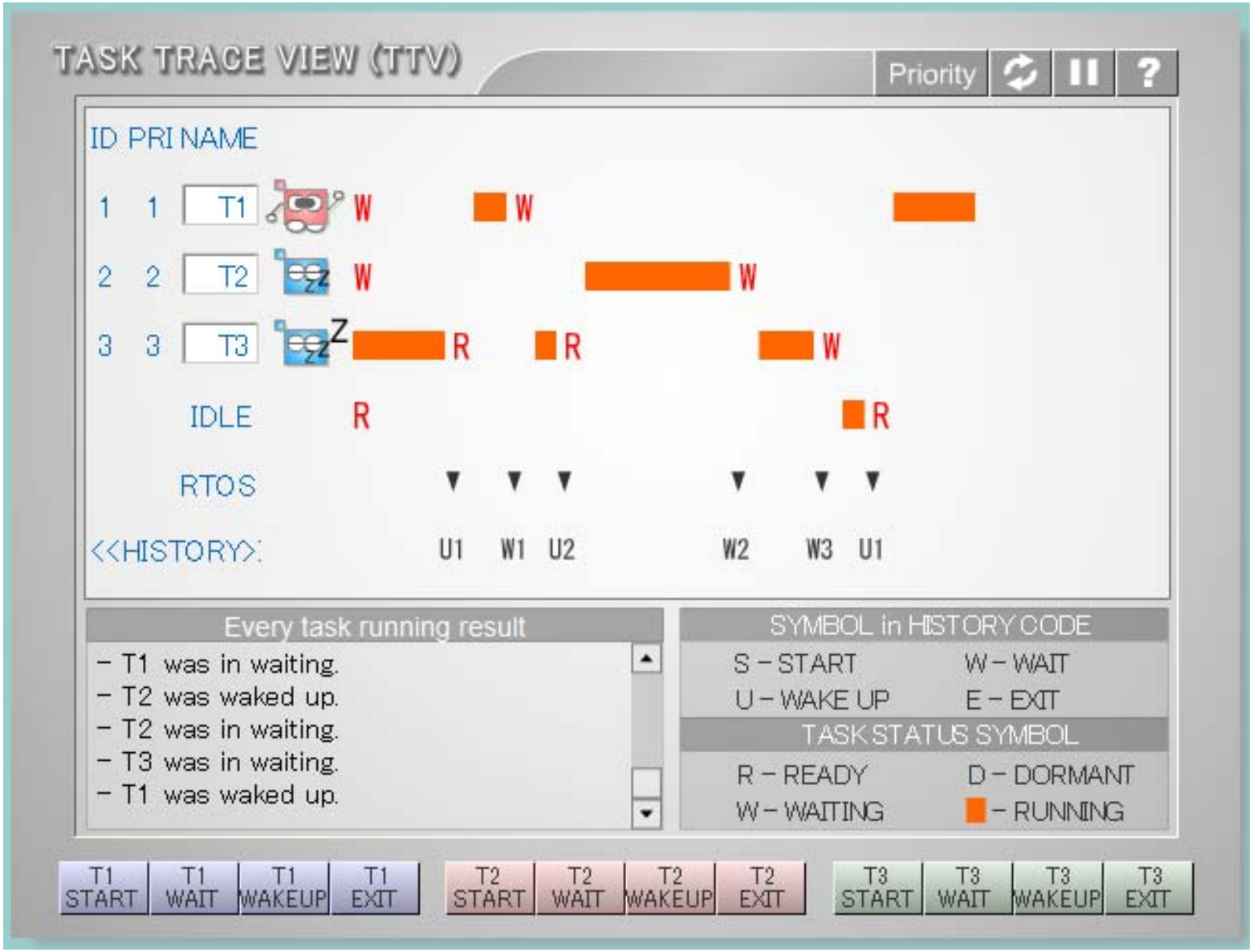
Scheduling : Priority and Round-robin

Priority scheduling : Major algorithm in embedded system



Response time(from request to complete) is shorter than round-robin, but lower priority task may not executed forever if higher priority task runs frequently.

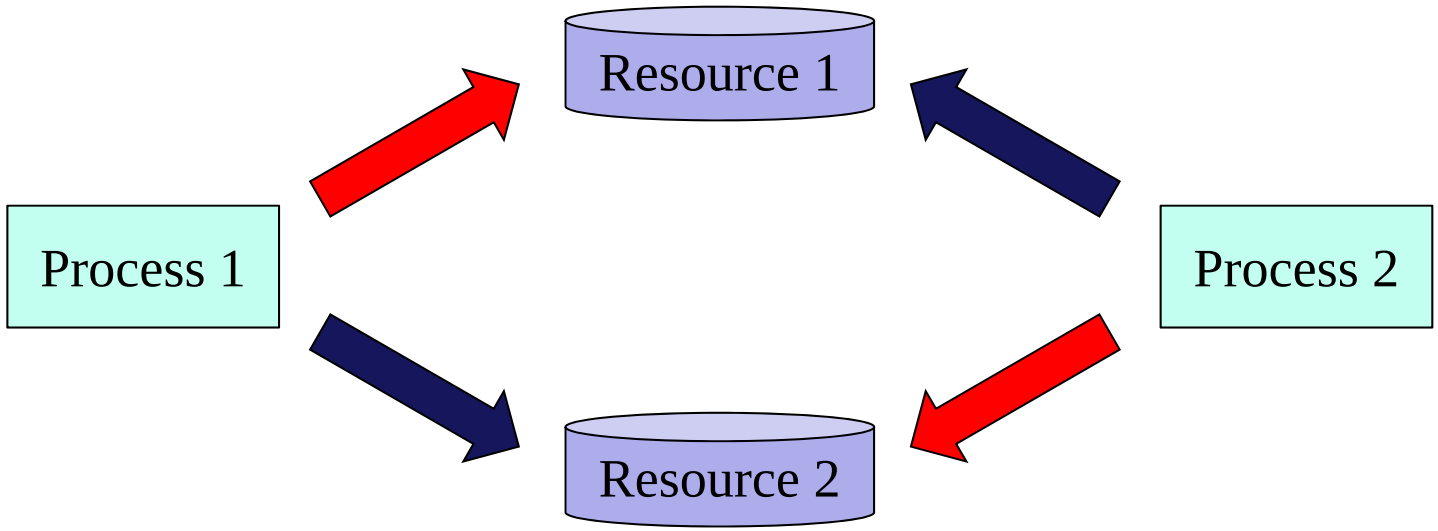
Scheduling: Demo



4.6 Resource control

- **Deadlock**

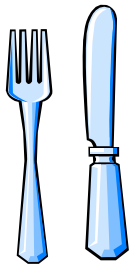
"If more than one resources are accessed from different process (task), there is always risk for deadlock"



Keeps to executing

Needs to finish

4.6 Resource control – Deadlock



Mr. A

Osechi – Japanese new year foods
(http://en.wikipedia.org)

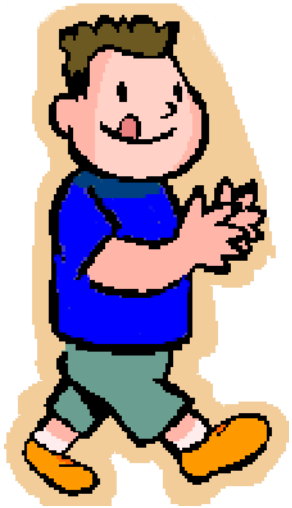
Mr. B

E.g. 2 (two) persons try to eat dinner,
with sharing one fork and one knife.

4.6 Resource control – Deadlock

1 Mr. A takes knife

2 Mr. B takes fork



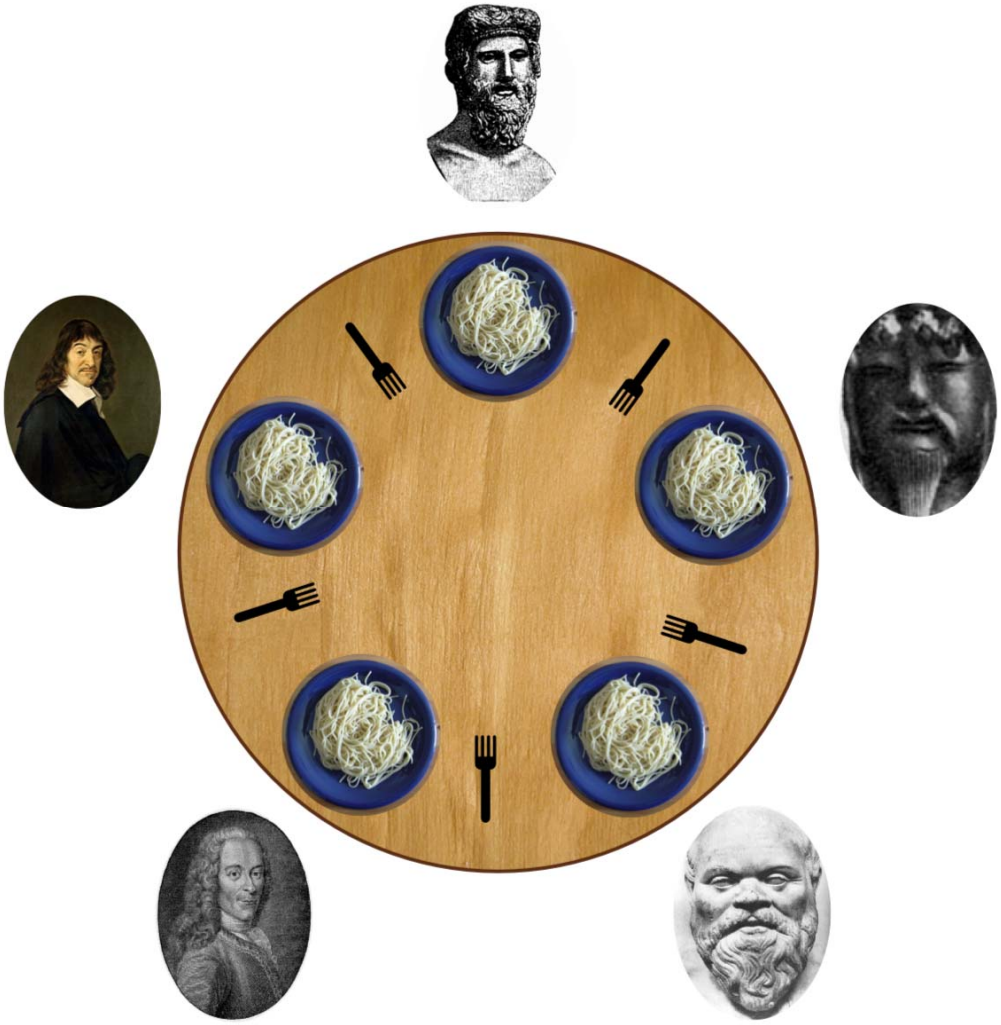
Mr. A

Osechi – Japanese new year foods (http://en.wikipedia.org)

Mr. B

3 Mr. A need fork, but fork is not available!
Mr. B need knife, but knife is not available!
No one can eat!

4.6 Resource control – Deadlock



An illustration of the dining philosophers problem.
Philosophers: Plato, Confucius, Socrates, Voltaire and Descartes
(http://en.wikipedia.org)

4.6 Resource control – Deadlock

❑ Condition of deadlock ("Coffman conditions")

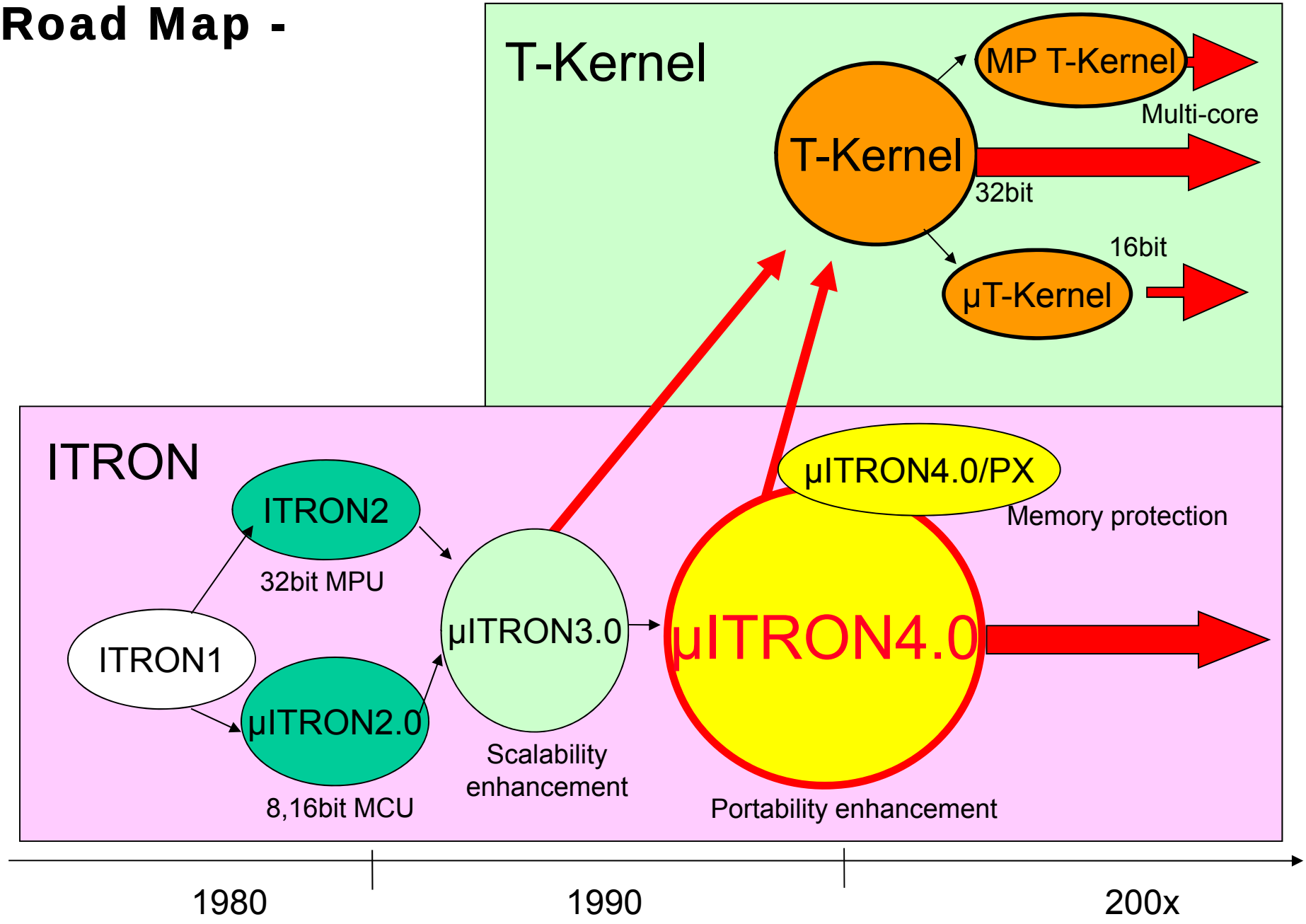
- **Mutual Exclusion:** At least one *resource must be held in a non-shareable mode*. Only one process can use the resource at any given instant of time.
- **Hold and Wait or Resource Holding:** A process is currently *holding at least one resource and requesting additional resources* which are being held by other processes.
- **No Preemption:** A resource can be *released only voluntarily* by the process holding it.
- **Circular Wait:** Processes must be *waiting for resources* which is being held by others *in circle*. In general, there is a set of waiting processes, P = {P1, P2, ..., Pn}, such that P1 is waiting for a resource held by P2, P2 is waiting for a resource held by P3 and so on until Pn is waiting for a resource held by P1.

4.7 uITRON outline

University of Tokyo Professor, Ken Sakamura advocated TRON project in 1984. The µITRON specification is a one of the specification of the real-time operating systems which was standardized in the ITRON project established as a subproject of the TRON project. The µITRON specification is now a de facto standard in the Japanese embedded industry.

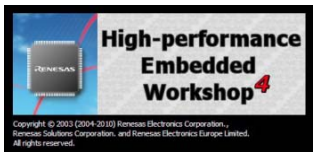
Renesas Electronics Corp., has been participated in the TRON project up to today since the project was established. [HI series OS] is a real-time operating system for the Renesas H8/SH microcomputers which is compliant with uITRON specifications. Refer to the TRON Association website for detailed information on the ITRON specification and the µITRON specification.

uITRON outline
- Road Map -

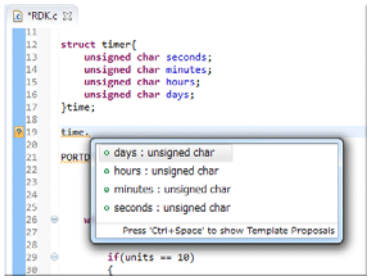


5. Integrated development environment

Development tools system configuration



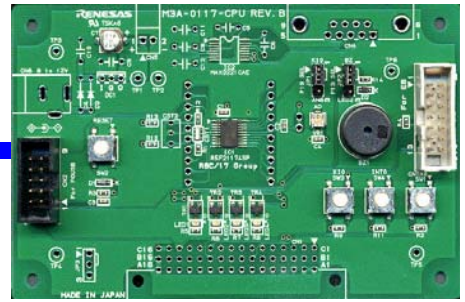
IDE



C/C++ source code debugger

Emulator

Target board



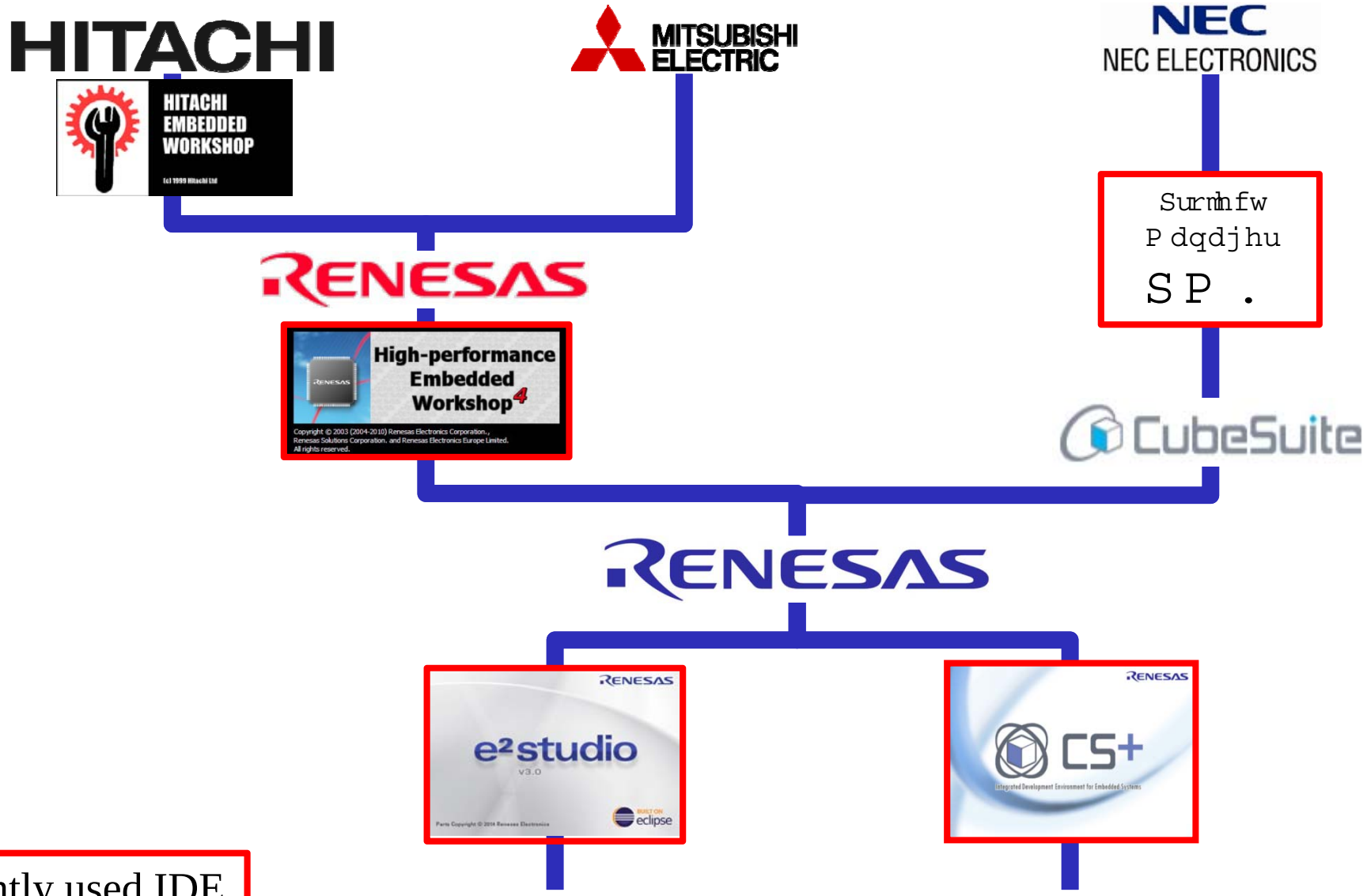
Simulator

C/C++ compiler, linkage editor, source code editor
Project manager, source code navigator
Profile analyzer, stack analyzer

3rd party tools（Compiler, Debugger, CASE tool, system simulator, program management tools)

5.1 Renesas IDE (Integrated Development Environment)

Timeline



Currently used IDE

5.1 Renesas IDE (Integrated Development Environment)

Development information

	Surmfw P dqdjhu SP .	High-performance Embedded Workshop 4	CS+	e² studio v3.0
Latest version	V6.3x (2007)	V.4.09.01 (Jun, 2012)	V3.02.00 (Octt, 2015)	V.4.3.0.0 (Jan, 2016)
Plan of next version	*none*	*none*	V4.00.00 (Apr, 2016)	V5.0.0.0 (Apr, 2016)
Copyright	Full	Full	Full	Part
Development status	*Maintenance*	*Maintenance*	In development	In development

5.1 Renesas IDE (Integrated Development Environment)

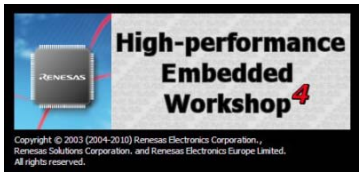
Supported devices (current version)

Surmhfw P dqdjhu SP.	High-performance Embedded Workshop 4	CS+	e²studio v3.0
RENESAS V850 RENESAS 78K	RENESAS H8 Tiny RENESAS M16C Platform RENESAS M32R RENESAS SuperH RENESAS RX RENESAS 740 RENESAS R8C	RENESAS RL78 RENESAS RX RENESAS V850 RENESAS RH850 RENESAS 78K RENESAS R8C …	RENESAS RL78 RENESAS RX RENESAS SuperH RENESAS RH850 RENESAS RZ RENESAS synergy™ …

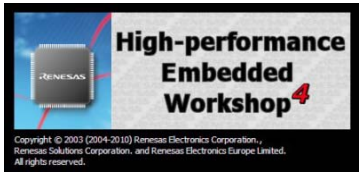
➔ Choose best device for product first. Then, selected suitable IDE.

5.1 Renesas IDE (Integrated Development Environment)

Supported Compilers

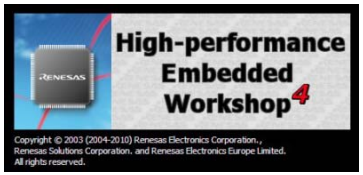
Surmhfw Pdqdjhu SP.			
Renesas Compilers	Renesas Compilers	Renesas Compilers	Renesas Compilers 3rd Party Compilers KPIT GNU IAR Green Hills

Operating System Environment

Surmhfw Pdqdjhu SP.			
Windows® XP Windows® 2000	Windows® 7 Windows Vista® Windows® XP	Windows® 8.1 Windows® 8 Windows® 7 Windows Vista®	Windows® 8.1 Windows® 8 Windows® 7

5.1 Renesas IDE (Integrated Development Environment)

Supported Renesas Debugger Tools

Surmhfw P dqdjhu SP .			
IECUBE MINICUBE2	E200F E100 PC7501 PC4701 M30870T2-CPE R0E521000CPE00 M38000T2-CPE E10A E1/E20 E10T E30A ...	IECUBE IECUBE2 E1/E20 MINICUBE MINICUBE2	E1/E20 IECUBE E10A Segger J-Link

➔ Refer section 8 and 9 for more information about Renesas Debugger Tool

5.1 Renesas IDE (Integrated Development Environment)

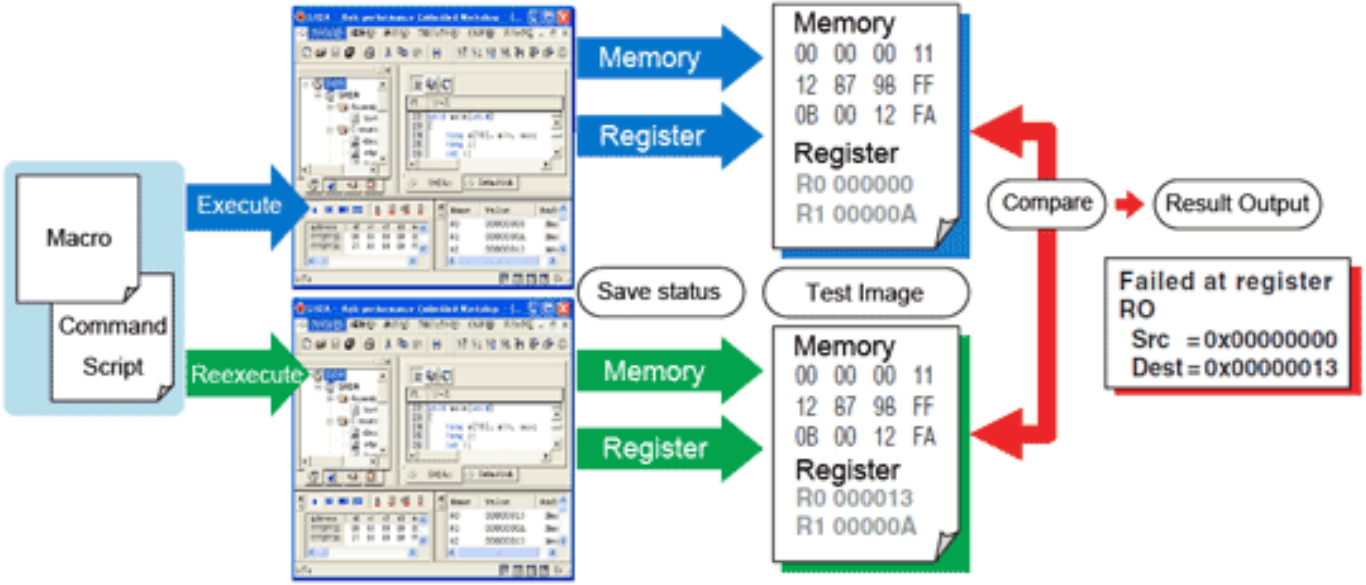
Porting projects among integrated development environments

Projects can be ported between the several IDEs in following directions:

Special feature – HEW

HEW has long development time, it is powerful and supports advanced debugging features such as code profiling, performance analysis, code coverage, trace features…

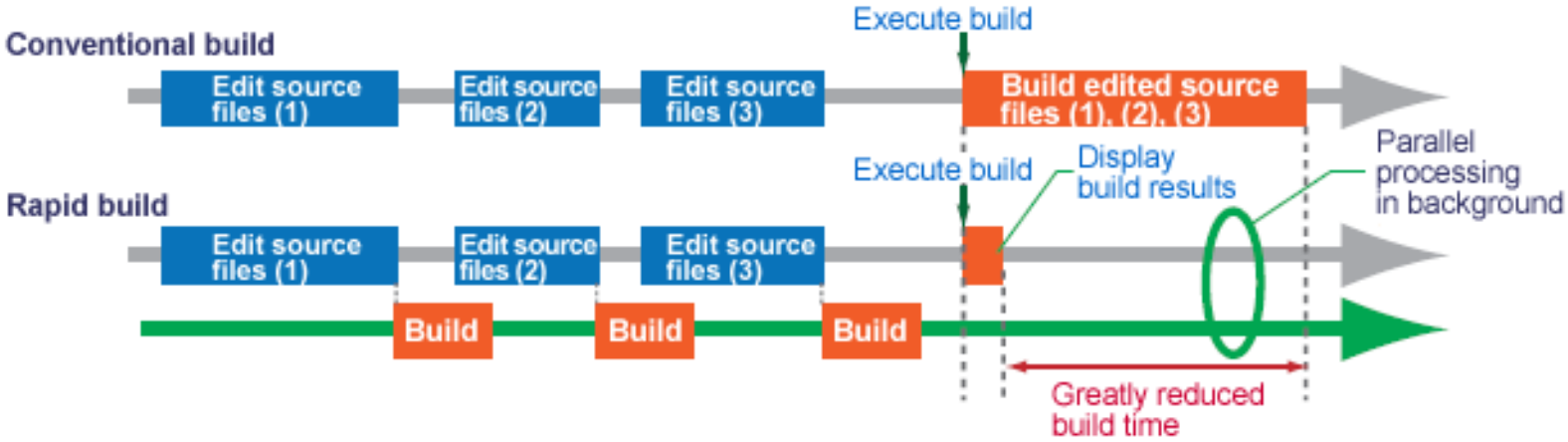
➔ We are selecting the most suitable features to implement for newer IDE (CubeSuite+, e^2 studio)



The test support function, if combined with **the macro generation support function**, becomes even more convenient. For example, if you have a test procedure prepared in a macro file and the expected value of test prepared in a test image file available, a series of repeat tests comprised of test execution and comparison of test result can be performed efficiently.

Special feature – CubeSuite+

CubeSuite+ can be used with the newly added RH850 family as well as the conventional MCUs, RL78 family, RX family, V850 family, 78K0R, and 78K0. Renesas will continue to expand the number of development tools supported by CubeSuite+, persistently making your development environment better and faster.



Rapid build enables greatly reduced build time

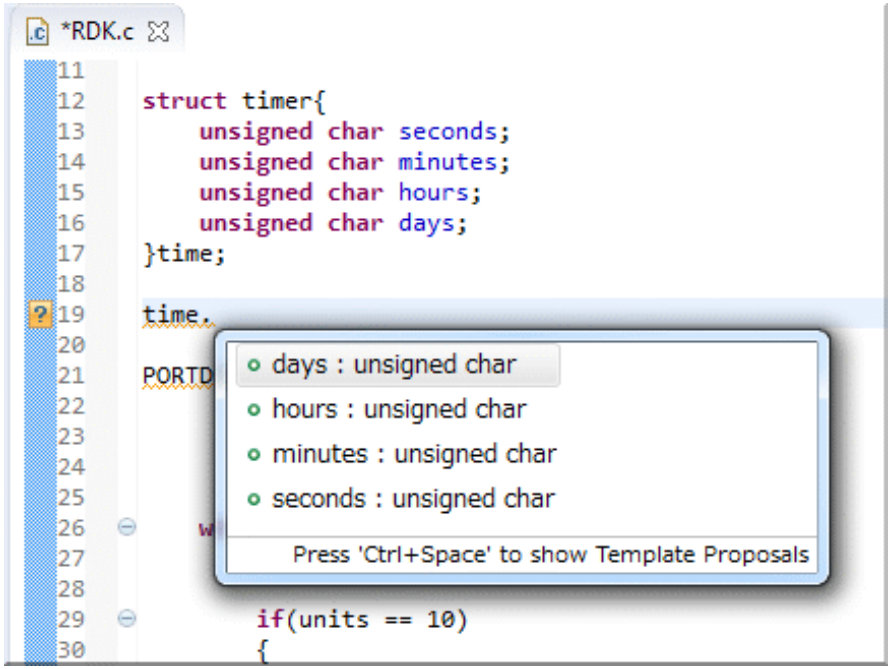
Conventional development environments require the user to edit all source files first, and then execute a build for the entire program, resulting in a lengthy build time.

CubeSuite+ features a "Rapid Build" function that automatically runs the build function in the background each time a source file is changed or saved, greatly reducing overall build time.

5.1 Renesas IDE (Integrated Development Environment)

Special feature – e² studio

A wide range of compilers can be integrated into e² studio to ensure customer has the choice of tools to match their project requirements. e² studio offers a state of the art coding environment for Renesas embedded controllers.



Eclipse CDT Editor

- Automatic code completion
- Keyword color coding of source code
- Built in spell checker
- Powerful code navigation
- Comment and code folding options
- Jump to declarations
- Automated code formatting (brace matching, comment blocks etc.)
- Code templates
- Automated code constructs (if, while, do…while etc.)
- In edit pre-processor checking (#ifdef code low-lighted if not true)
- Auto variable completion while writing

6. Cross software

Contents

6. Cross software

6.1 C/C++ compiler

6.2 Linkage editor

6.3 MISRA-C rule checker

6.1 C/C++ compiler

Outline

- The Renesas C Compiler is an optimizing *ANSI C and ANSI C++* compiler for the microprocessors.
- In addition to full ANSI C support, the compiler provides *#pragma language extensions* and command-line switches to support target specific features and extended compiler functionality.
- The C compiler has powerful and reliable code generation facilities for the targets. A variety of *optimization* features allow you to generate highly optimized PROM code. In particular, code can be optimized for size or speed to match the requirements of the particular application being developed.

6.2 Linkage editor

Outline

- Linkage editor input the object programs from the compiler or assembler and *output the load modules or library files*.

Optimization

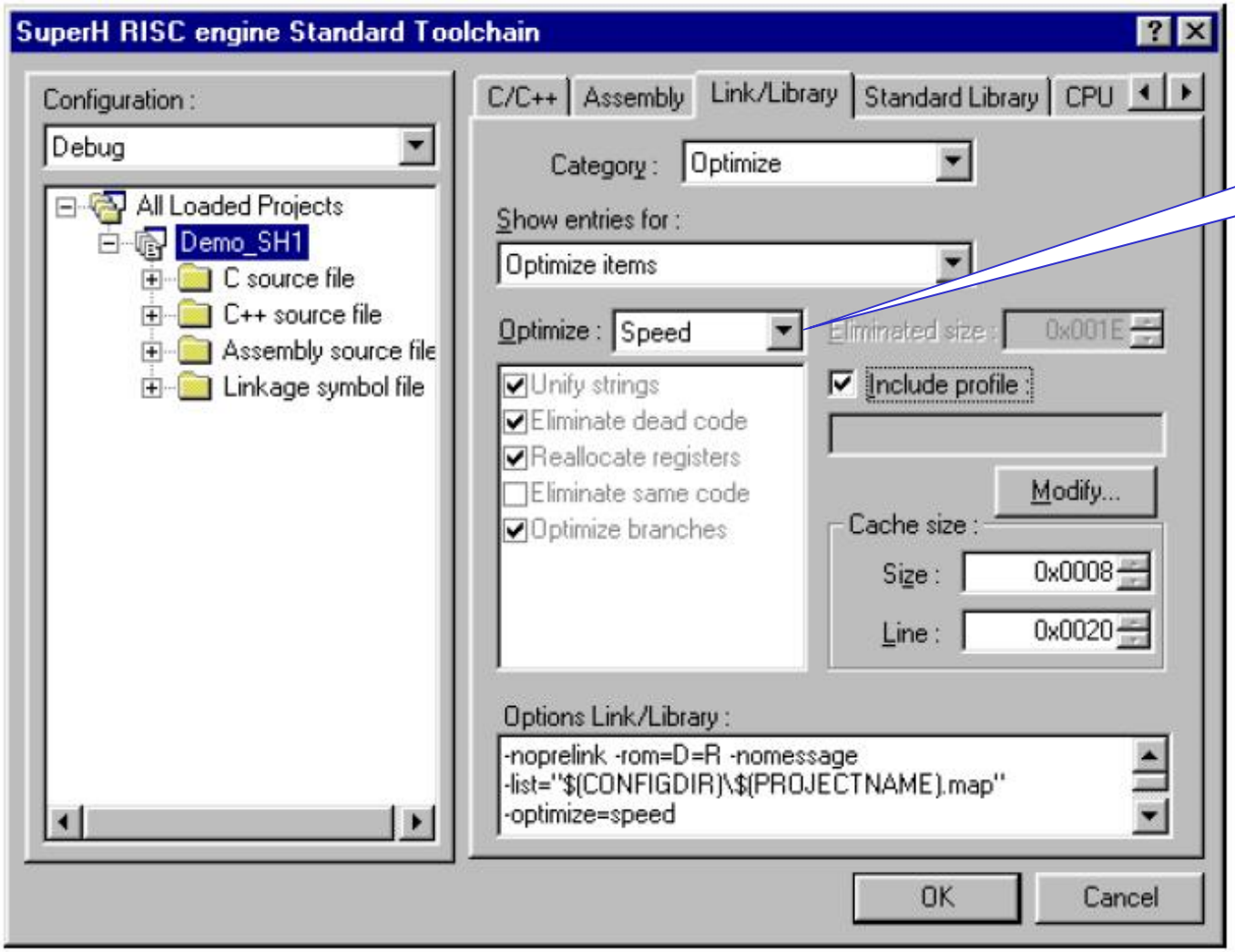
- Linkage editor analyzing the relation ship *between the object programs*.
- MAP optimizing: Optimizing linker re-compile the program using symbol allocated address defined by compiling or linking.
- Global optimization: Combine all compiler objects to an unique object file.

Output file

- Linkage editor output the file in the specified format: Absolute(ELF), Motorola S-type, HEX type, Binary type and symbol reference list.

Linkage editor

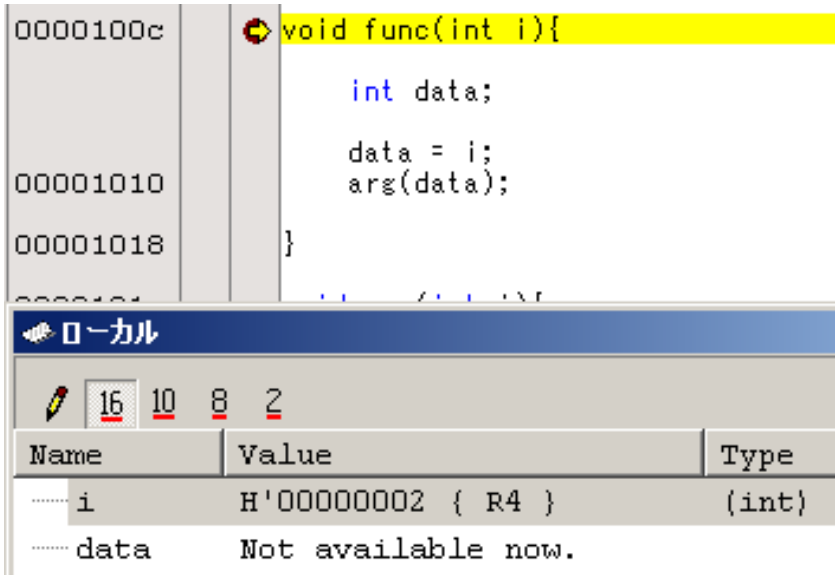
Optimization (1) : Selection of speed optimize or size optimization



Linkage editor

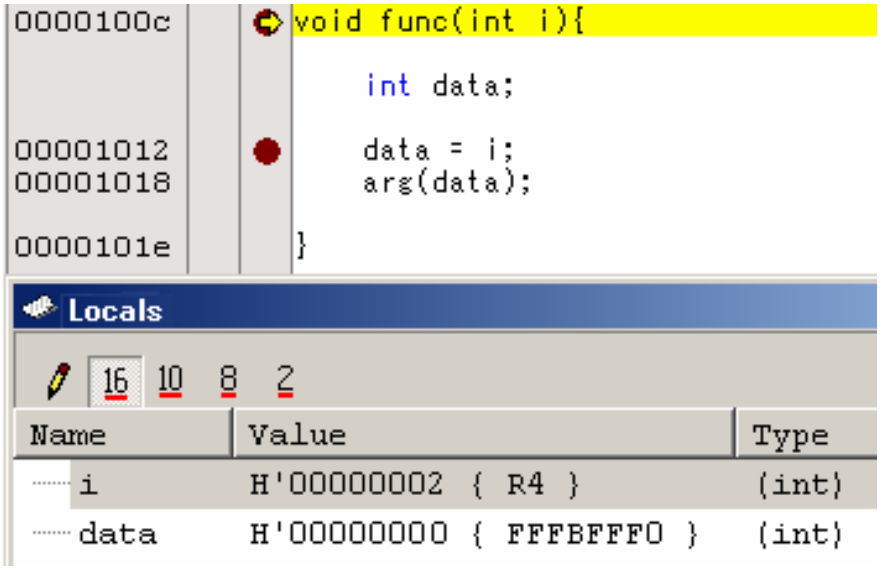
Optimization (2) : Selection of debug easiness or optimized code

[optimize=0]



Local variables may not be visible in some case, but code is best optimized.
Suitable for final object generation

[optimize=debug_only]



Local variables are always visible, but code is not best optimized.
Suitable for algorithm debug.

6.3 MISRA-C rule checker

Overview

SQMlint is a tool to inspect C source codes according to MISRA C rules. Because SQMlint assists developers in source code review through automatic inspection, it helps to develop *high-quality C source codes* efficiently. Furthermore, because improper codes can be detected by only adding compile options, the user can correct such codes easily when correcting source code pointed by compiler error messages.

MISRA C is a set of software development guidelines for the C programming language developed by MISRA (**M**otor **I**ndustry **S**oftware **R**eliability **A**ssociation). Its aims are to **facilitate code safety, security, portability and reliability** in the context of embedded systems. MISRA has evolved as a widely accepted model for best practices by leading developers in sectors including **aerospace, telecom, medical devices, defense, railway**, and others. There is also a set of guidelines for MISRA C++.
MISRA C is not an open standard; the guideline documents must be bought by users or implementers.

MISRA-C rule checker

Features

SQMlint is supplied as an additional function to Renesas compilers, so that it provides the following features:

- Inspection is possible by only adding options when compiling source files.
  - Source files can be inspected at the same time they are compiled.
  - There is no need to build an inspection-purpose environment.
  - Adaptable to special options of the Renesas C compiler.
- Can inspect in about less than half the compile time.
- Does not affect the compile results at all.
- Usable in Renesas integrated development environments : HEW, CS+, e^2 studio

- For more detail, please refer ‘MISRA C Rule Checker SQMlint User’s Manual’ (rej10j1314_sqmlnt_u.pdf) or related C compiler application note, such as ‘SuperH C/C++ Compiler Package Application Note’ (rej05b0463_superh.pdf), Section 10 MISRA C.

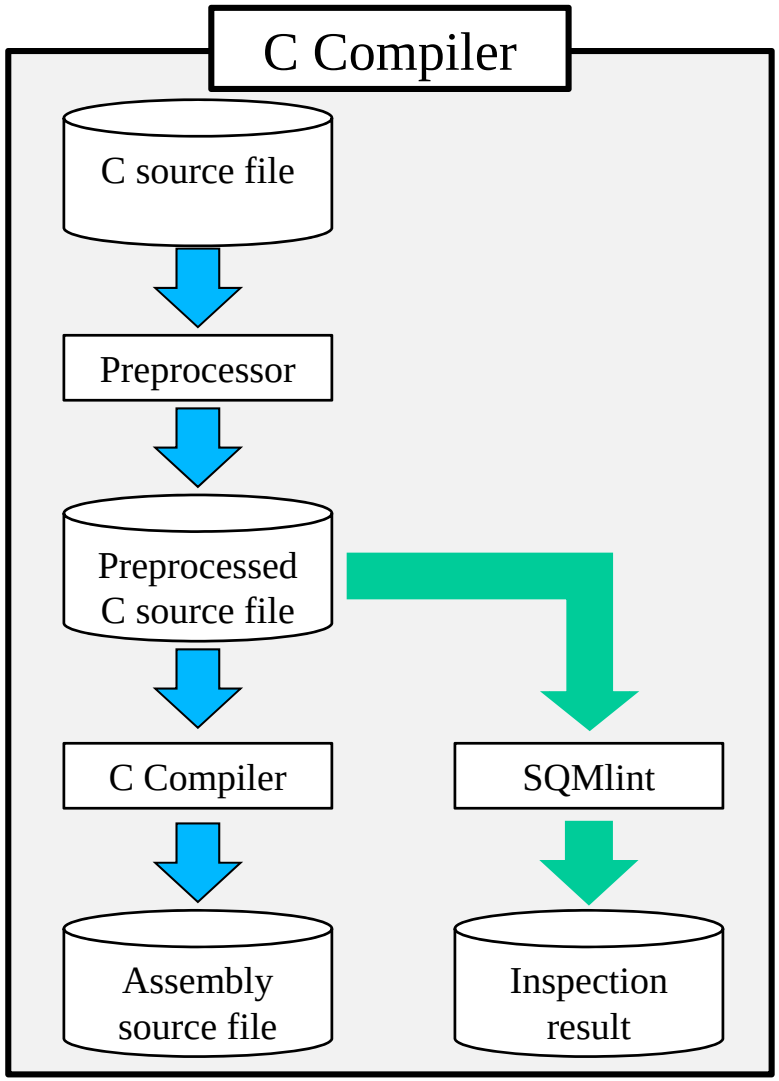
MISRA-C rule checker

Functions

- MISRA C rule inspection during compilation
  - Source codes can be inspected against MISRA C rules by only specifying options when compiling the source files.
- Adaptable to special options of the Renesas C compiler
  - If special options such as one that handles the double type as float type are used, C source codes are inspected by taking the effects of those options into consideration.
- Usable in Renesas integrated development environment
  - You can make a tag jump in Renesas integrated development environment on the output results and correct the C source codes on the spot.
  - Therefore, deviations from MISRA C rules can be corrected in the same way as when correcting compile errors.
- Inspection results output in CSV format
  - The inspection results are output in CSV format usable in spread sheet software, allowing you to put the inspection results in order easily.

MISRA-C rule checker

Number of MISRA C Rules which can be Inspected by **SQMlint**

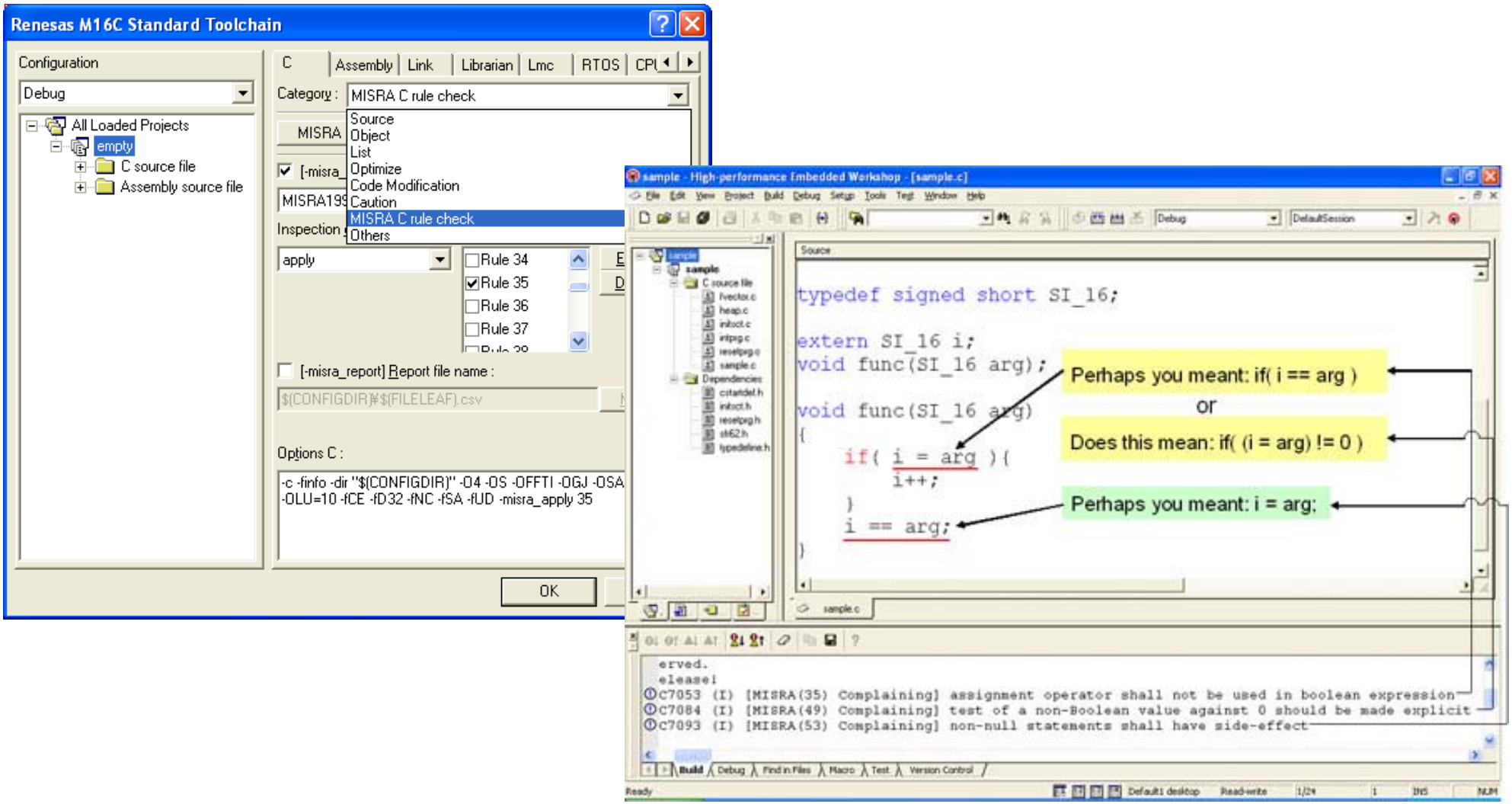


Rule classification	MISRA C 1998	MISRA C 2004	MISRA C 2012
Required	67/93	79/122	
Advisory	19/34	13/20	
Total	**86/127**	**92/142**	**79/143**

MISRA-C:2012 contains 143 rules and 16 "directives"; each of which is classified logically into a number of categories.

MISRA-C rule checker

SQM lint integrated in HEW



7. Simulation tools

Contents

7. Simulation tools
7.1 Simulator outline
7.2 Simulator features

7.1 Simulator outline

The Renesas simulator enables you to debug/test software modules without target hardware. Useful debug function is provided, such as go/break, step, coverage and interrupt handler debug support.

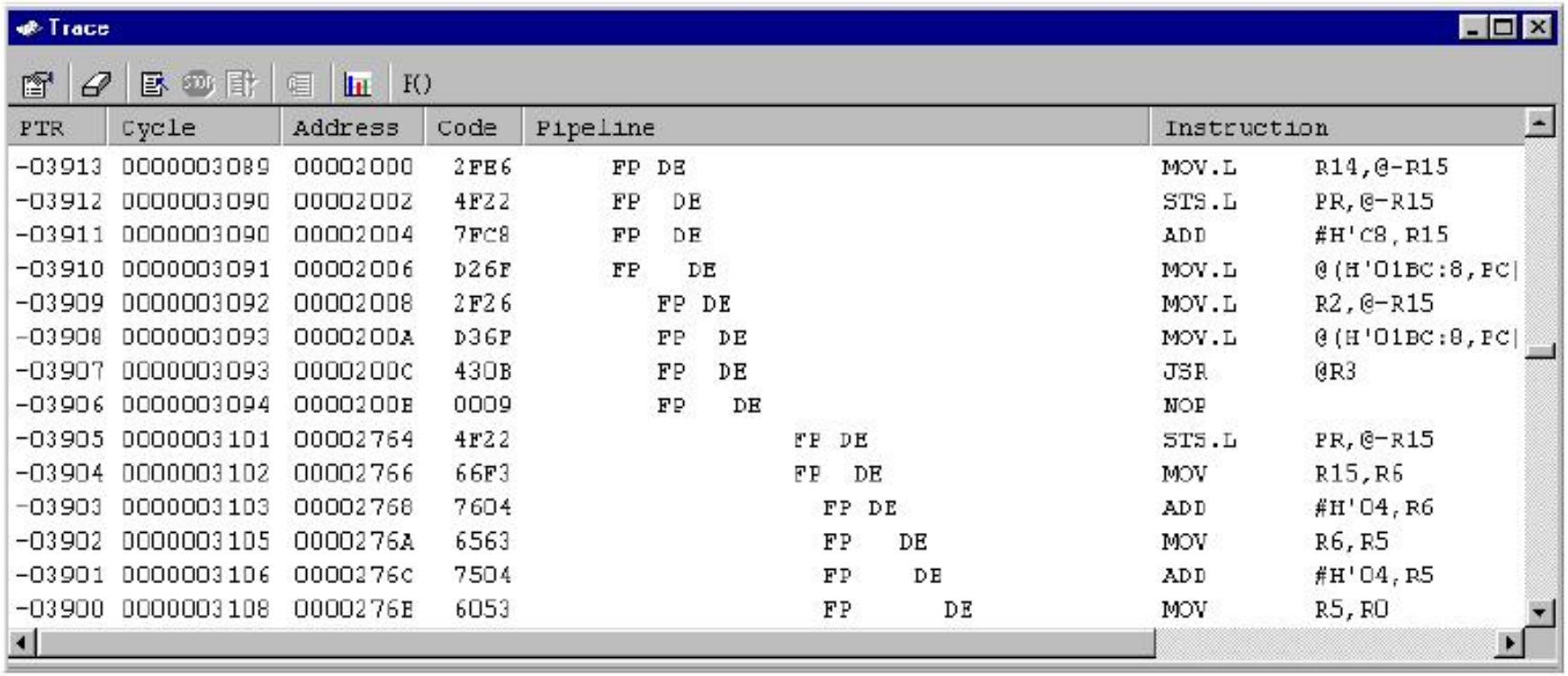
HEW provides consistent HMI(Human Machine Interface) both in build and debug.

7.2 Simulator features

- Support for ELF/DWARF object format
- Support graphic display of *execution cycle* count and called count in function units
- Provide *stack trace* function
- Display *cache hit rate*
- Display *state of pipeline*
- Provide extensive break point functionalities (*Pseudo interrupt* is available)
- Display *coverage* information in assembly source level
- Provide debug function in visual display with *images and waveforms*
- Display *memory access* counts in memory type units (embedded-memory or external memory)

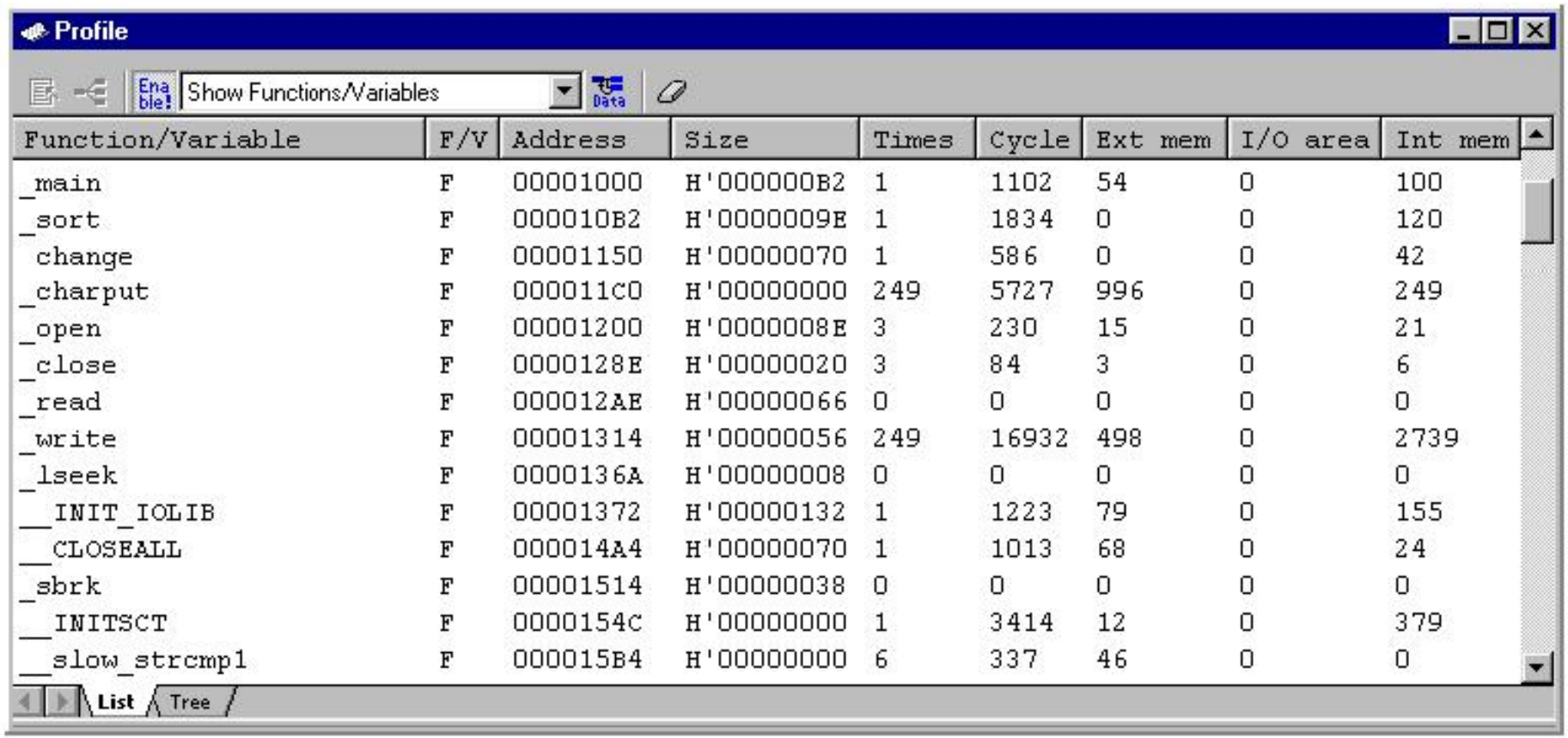
HEW window example (1)

Trace window with pipeline execution status



HEW window example (2)

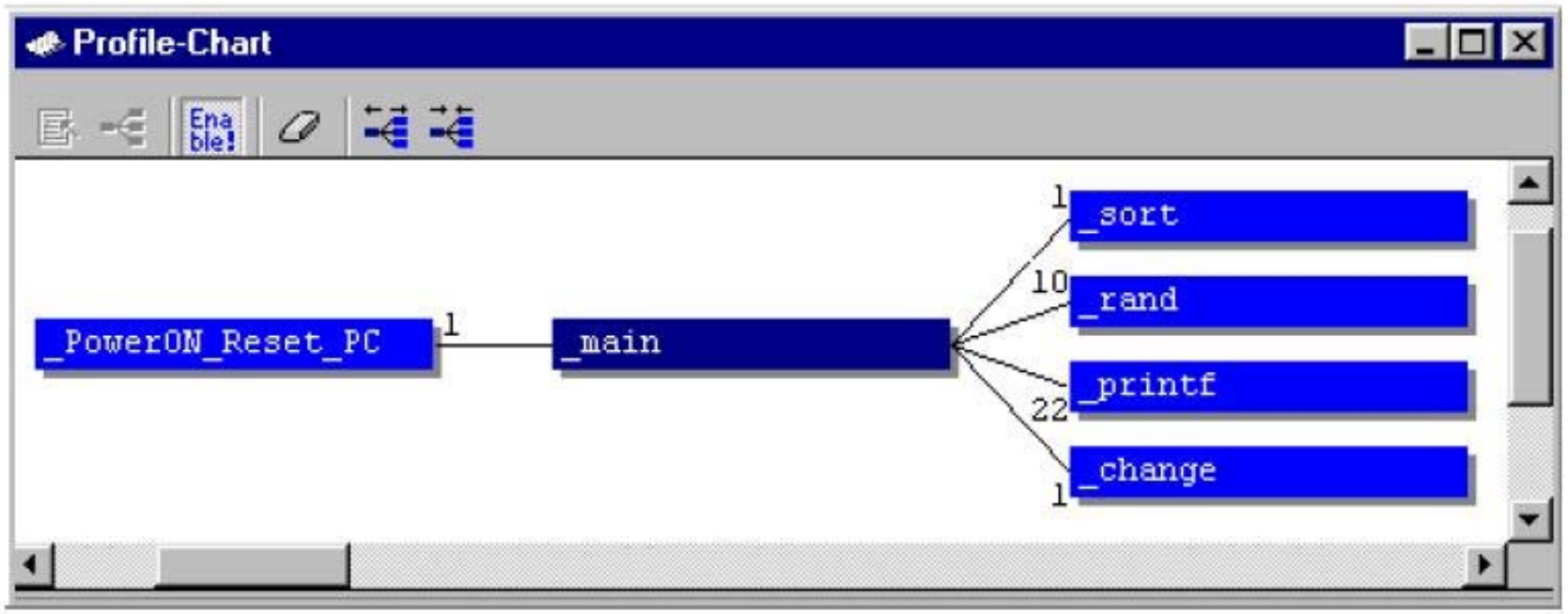
Profile window after execute the program by simulator



Function/Variable	F/V	Address	Size	Times	Cycle	Ext mem	I/O area	Int mem
_main	F	00001000	H'000000B2	1	1102	54	0	100
_sort	F	000010B2	H'0000009E	1	1834	0	0	120
change	F	00001150	H'00000070	1	586	0	0	42
_charput	F	000011C0	H'00000000	249	5727	996	0	249
_open	F	00001200	H'0000008E	3	230	15	0	21
_close	F	0000128E	H'00000020	3	84	3	0	6
_read	F	000012AE	H'00000066	0	0	0	0	0
_write	F	00001314	H'00000056	249	16932	498	0	2739
_lseek	F	0000136A	H'00000008	0	0	0	0	0
__INIT_IOLIB	F	00001372	H'00000132	1	1223	79	0	155
__CLOSEALL	F	000014A4	H'00000070	1	1013	68	0	24
_sbrk	F	00001514	H'00000038	0	0	0	0	0
__INITSCT	F	0000154C	H'00000000	1	3414	12	0	379
__slow_strcmp1	F	000015B4	H'00000000	6	337	46	0	0

HEW window example (3)

Trace profile chart after simulation



8. Evaluation boards

Contents

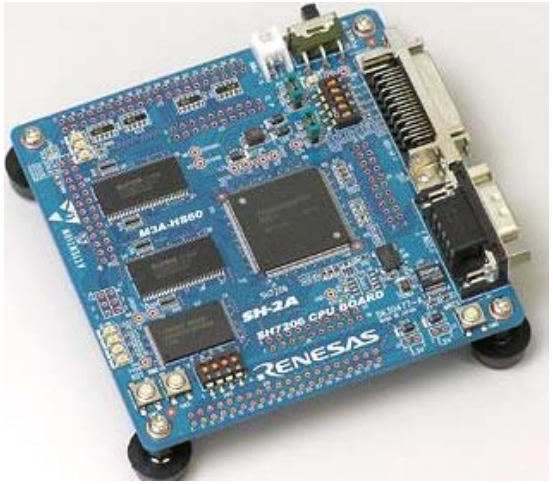
8. Evaluation board

8.1 Renesas evaluation board

8.1 Renesas evaluation board

Renesas design and sell the evaluation board for the customers.
When Renesas start new product promotion, we recommend Renesas evaluation board, because this is available earlier than other board from other company.

Renesas evaluation board include the minimum hardware and no monitor program.
When you use, E10A-USB or other H-UDI interface emulator is required.



SH7206 evaluation board

R-Car evaluation board

9. Emulation tools

Contents

9. Emulation tools

9.1 Emulator overview

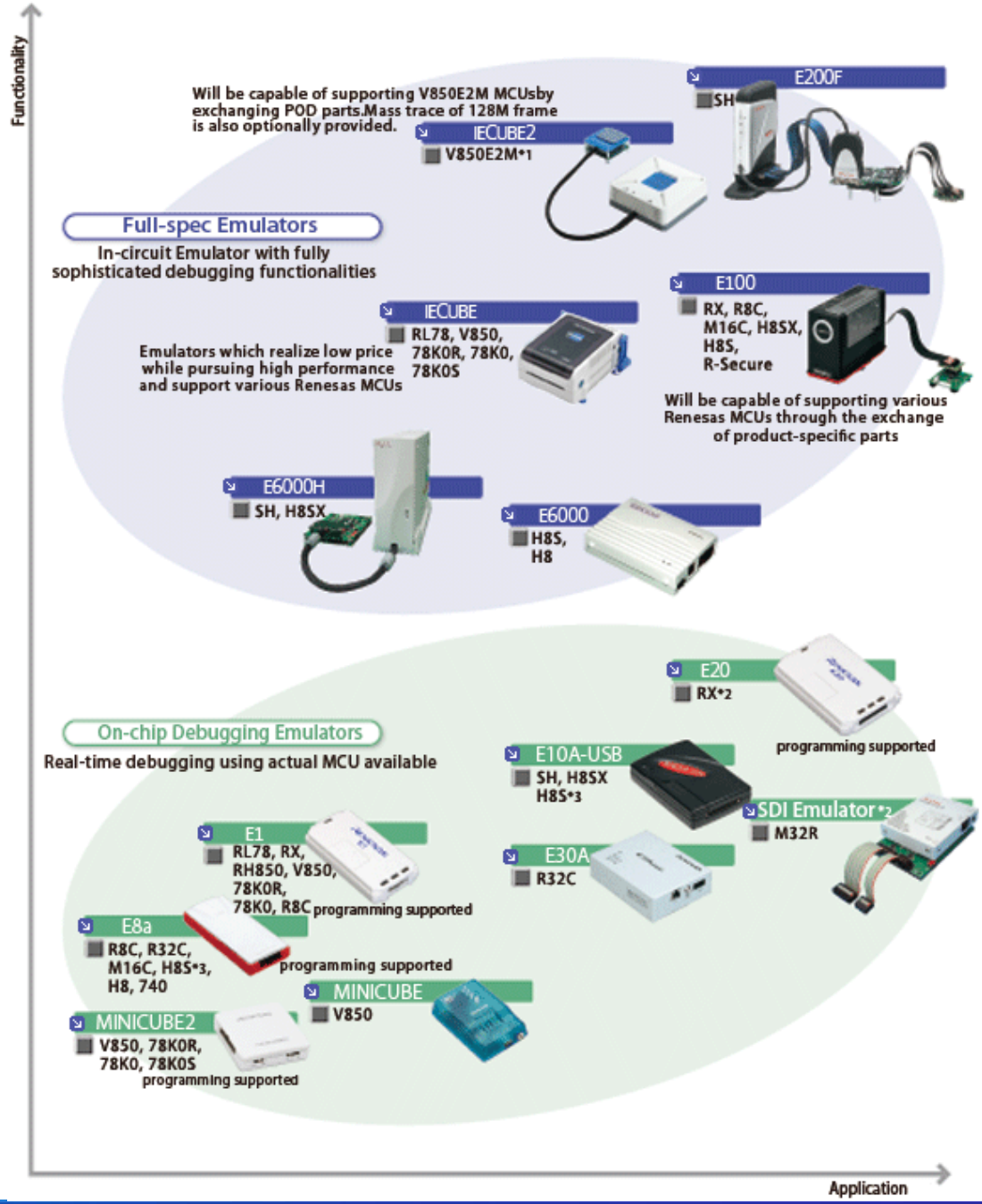
9.2 Popular debug function

9.3 OCD (On-chip debugger) emulator

9.4 Monitor debugger

9.5 Test Support Function

What is Emulator (ICE : In-Circuit Emulator)?



What is Emulator (ICE : In-Circuit Emulator)?

ICE is special debug tool for embedded software debug

- ICE provides debug capability without OS, without standard IO
- Just target hardware and no software, this is enough start point to use ICE
- Connect ICE to target hardware (and to PC) and start debugger software, then you have access to all resource in target hardware, such as memory, peripheral, etc.
- ICE allows you to download object code from host PC to the memory in target hardware.
- You can start program execution at any point you need (GO)
- You can stop program execution at any point you need (BREAK)
- You can execute program step by step, if necessary (Single STEP)
- You can modify variable data value at any place (Memory)
- You have access to execution log, to confirm that the program is executed as you expected (Trace)

What is Emulator's advantage against Simulator?

– Debug with target hardware
– Software for physical layer access, such as driver, ipl, etc.
– Combinational bus operation by CPU and DMA (or any other bus master).
– Interrupt collision detection
– Non synchronous communication between two different hardware.
– Any other timing critical scenario

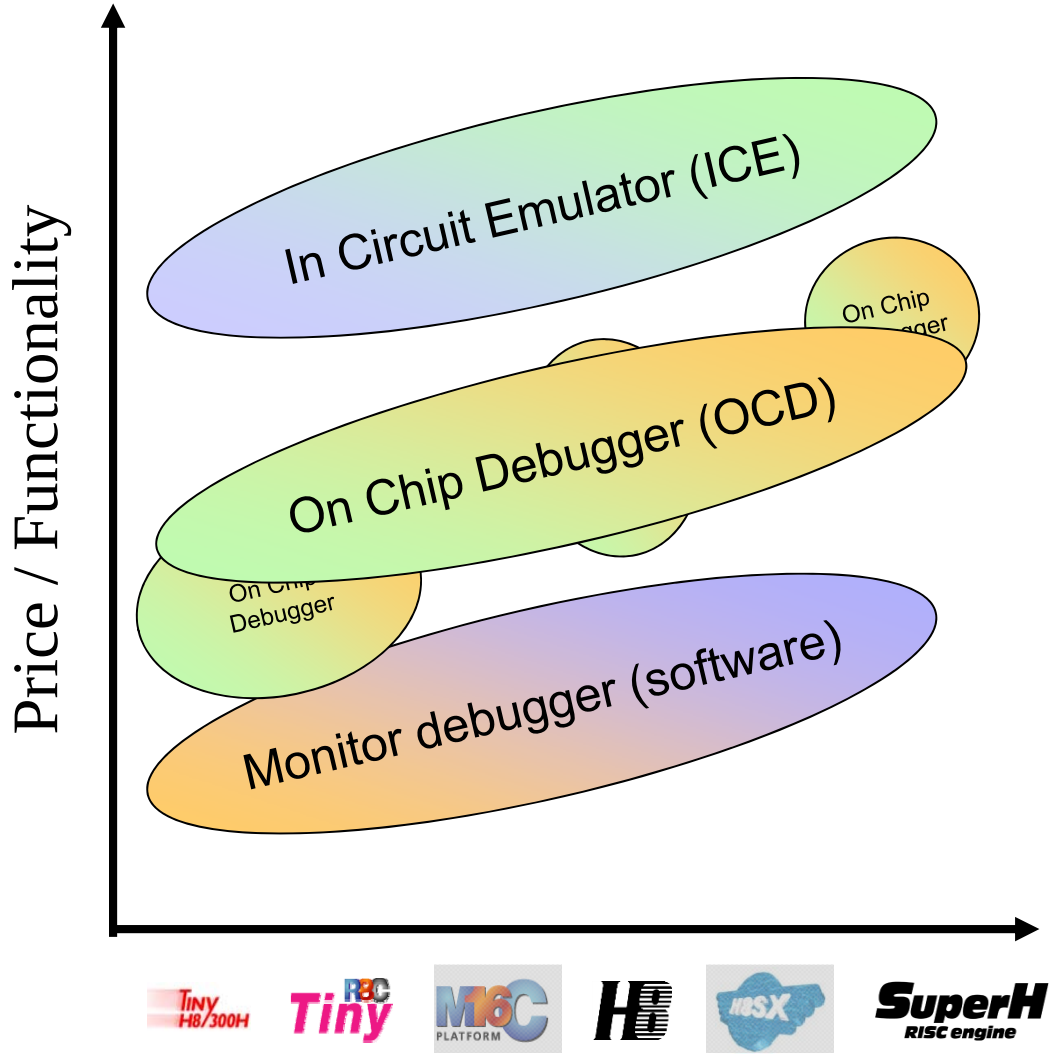
What is Simulator's advantage against Emulator?

– Debug without target hardware (before target hardware ready)
– Low cost
– Precise CPU internal view, such as CPU pipeline operation view
– Concentrate on software debug, independent from hardware bug

9.1 Emulator products

Debug tool who provides debug function with target hardware, is divided into three (3) type of product.

- ✓ In-circuit emulator is high functional/high price debug tool.
- ✓ OCD(On-chip debugger) is low cost ICE with reduced debug function. Currently most popular debug tool in the world.
- ✓ Monitor debugger is lowest price debug solution by software. Lower debug capability than ICE/OCD.



Differences between ICE, OCD, and Monitor

ICE (In-Circuit Emulator)

- Traditional debug tool
- High price, high functionality debug tool
- Target interface : IC socket on the target board
- Due to CPU clock speed up, it becomes hard to develop ICE for high-end MPU, such as SH-4(4A)

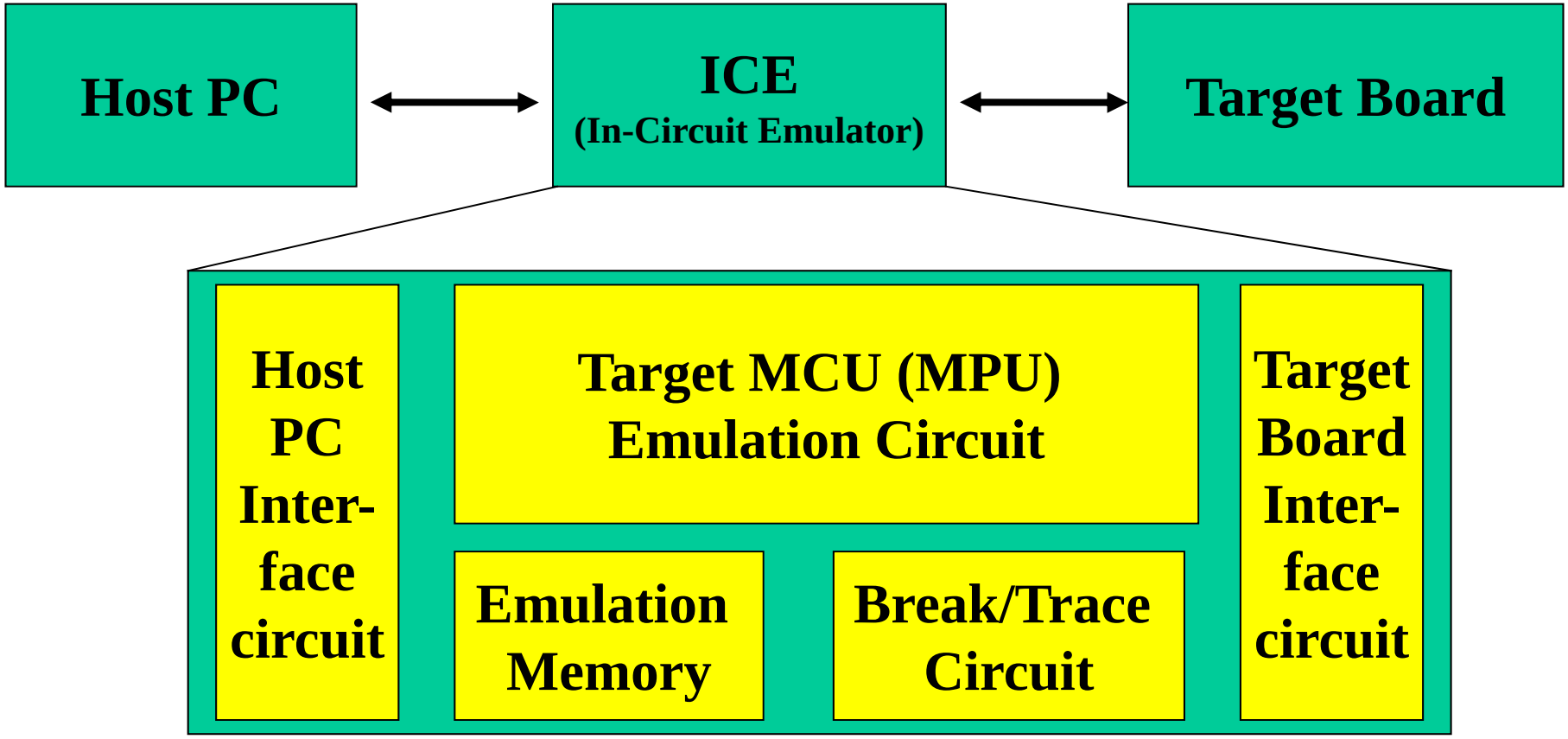
OCD (On Chip Debugger)

- Mid price, mid functionality debug tool
- Target interface : JTAG connection (or serial)
- Production device includes DM(Debug Module)

Monitor

- Low price, low functionality debug tool
- Target interface : serial (or Ethernet)
- Traditional debug tool made by software.
- No specific hardware is required, but debug function is limited.

What is ICE?



Sample:

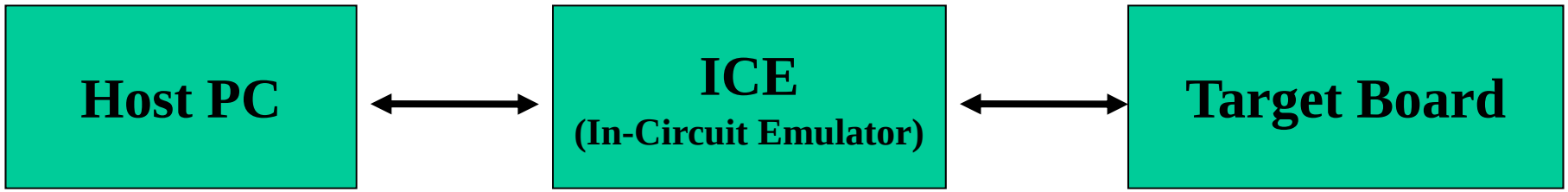
E600H

M32100T5-SDI-E

E100

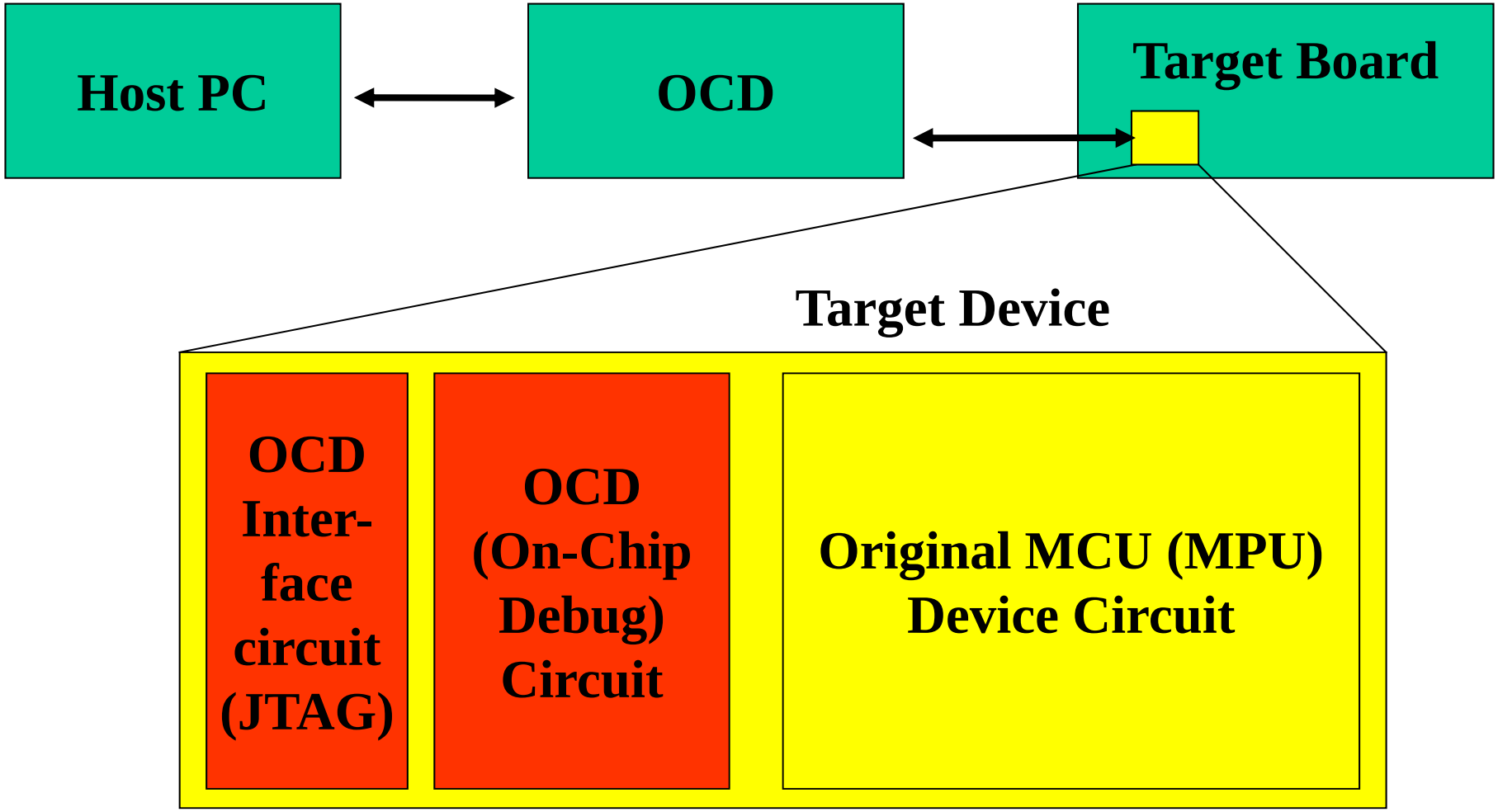
IECUBE2

What is ICE?



- If the target board does not work correctly, ICE works fine without any problem. As this is independent hardware to target board.
- ICE has its memory (emulation memory) for target program, it is possible to test software without target board.

What is OCD?



Sample:

What is OCD?

- If target board hardware has problem, then OCD emulator does not work.
- There is no way to run target program, if target board does not work.

9.2 Popular debug function

Breakpoint

Function to stop your program execution at the just point you want to stop

- Hardware Breakpoint
  Break function supported by dedicated hardware. In most devices, complex break condition, such as PC+operand address/data+read/write, is supported.

- Software breakpoint
  Break function supported by code replacement. Complex break condition is not supported, just PC break only.

Trace

- PC(Program Counter) trace
  Shows back trace log of PC(Program Counter). After the break, you may find that PC is not the value of your expectation. Then you need to find why (and from where) program comes and stops here. Trace function gives you the useful information for analyzing the program flow.
  Data trace is also available.

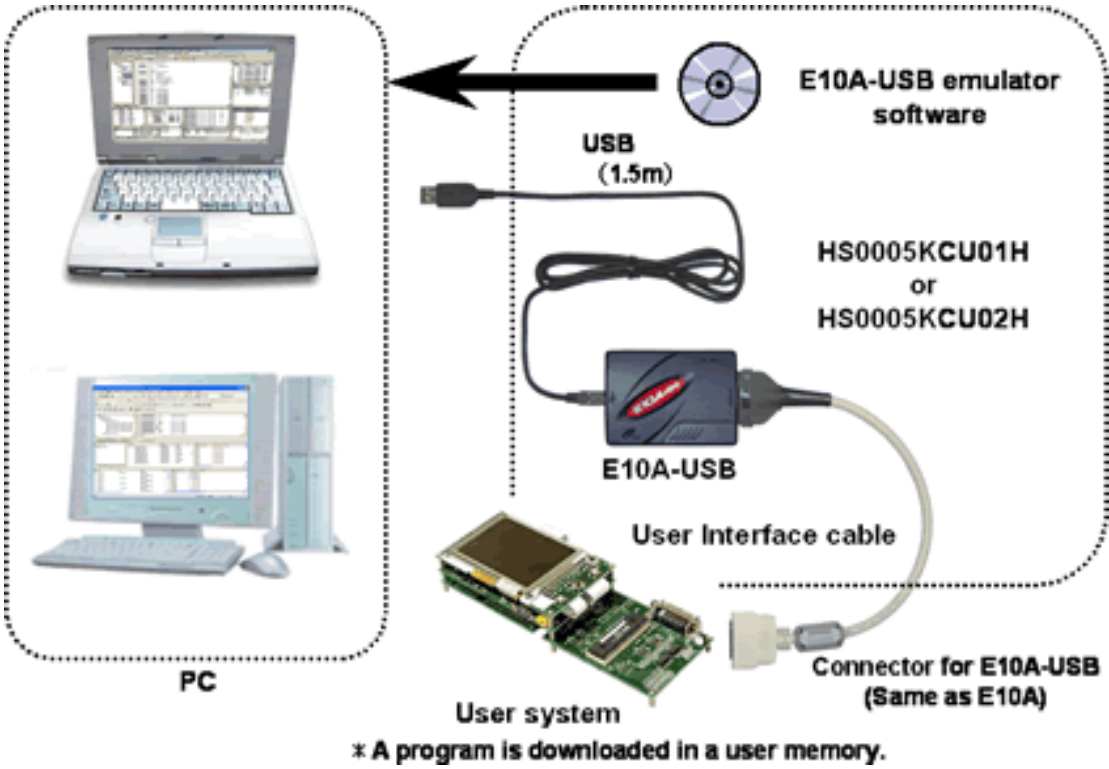
9.3 OCD emulator

The OCD emulator offers lower price than ICE. Renesas OCD emulator price is from $100 to $2000.

There are several debug interfaces, as follows,

(1) JTAG(H-UDI): SH, H8SX, H8S

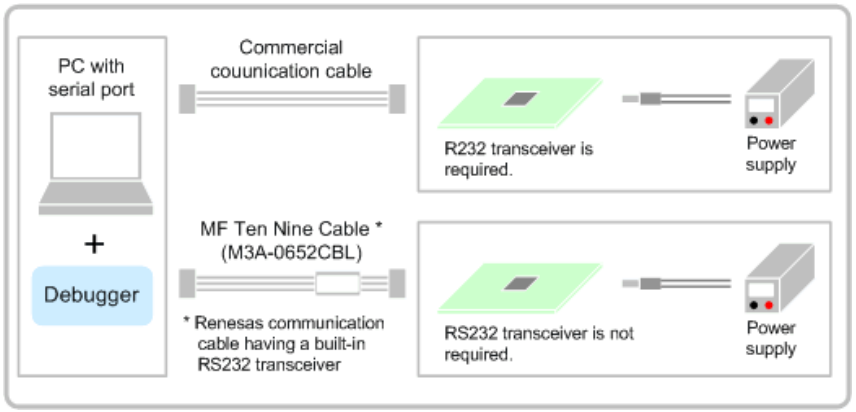
(2) UART:H8/Tiny, H8/SLP, R8C, M16C



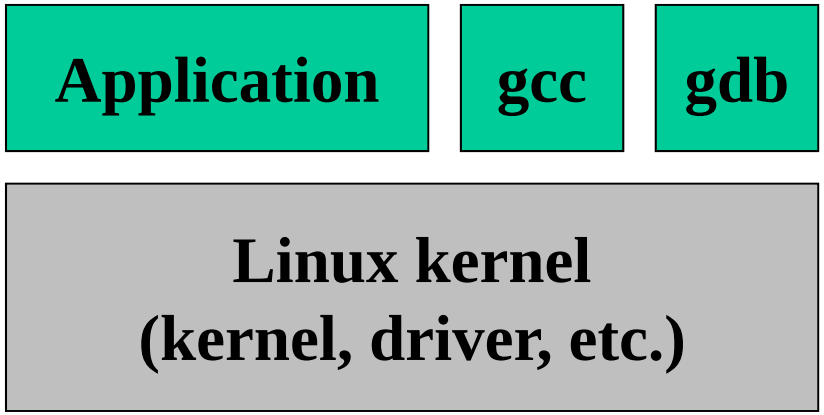
9.4 Monitor debugger

Monitor debugger does not need any hardware assistance (except interface cable between target hardware and host PC). After the monitor program is downloaded into the target hardware, the program starts communication to host PC. Most popular interface is UART.

Monitor program does not work, when communication channel hardware is not working.



Debug function in OS



On Linux system, debugger (gdb) is one application software.

Gdb works with kernel to provide debug function for application software engineer.

Standard IO (keyboard/mouse and display) is used for debug information communication.

Target hardware must have such standard IO, if debug function is required.

9.5 Test Support Function

Primary goal of debug is remove bug and make target program work (functional debug).
Beyond functional debug, you may need to work further in following two points

- Quality Assurance
- Performance enhancement

Emulator provides debug function not only for functional debug, but also for software quality improvement and performance enhancement.

9.5.1 Code Coverage

Code coverage is one of software quality index in test phase.
One important test criteria for high quality software is to achieve 100% C0 and/or C1 coverage.

Some emulator supports C0 coverage function.

For example, E200F Emulator (for SH-4A) support following coverage function.

- C0 coverage (C1 coverage is not supported)
- 4M Byte address range (default)
- Prefetch cancel
- Source code view with coverage result (executed / not executed)

C0 and C1 Coverage

- C0 coverage
  - How many lines (or instructions) are executed against all source lines (or all instructions)
- C1 coverage
  - How may branches are taken (or not taken) against all branch cases

- *Coverage definition in C source level (statement) and assembly level (instruction) is different*

C0 and C1 Comparison

- C0 coverage 100%, but C1 coverage is not 100%

```
          Statement A
          Conditional Branch B
             (if true branch to 'Lable1')
          Statement C
Label1:   Statement D
```

If conditional branch B is not taken at the test case, all statement is executed

➔ C0 is 100%.

But, 'branch B taken' is not tested in such test case

➔ C1 is NOT 100%

C0 and C1 Comparison

- C0 coverage is not 100%, but C1 coverage is 100%

```
            Statement A
            Conditional Branch B
               (if true branch to 'Lable1')
            Statement C
            Branch always
               (always branch to 'Lable2')
            Statement D
Label1:     Statement E
Label2:     Statement F
```

Statement D can NOT be executed in any test case.

➔ C0 is NOT 100%

Statement D must be removed from source code, or need to change source code in proper way.

9.5.2 Performance / Profile

Performance enhancement can be achieved by hardware upgrade (takes higher frequency, wider bus, etc.) or software tuning. In high-end application, there are many rooms for performance enhancement by software tuning.

Some emulator provides performance (or profile) function for software performance tuning. This function provides several useful information for performance enhancement, such as execution time information in specific function or specific address range (from start point to end point).

- CPU clock count (or number of executed instructions) from specific start address to specific end address
- number of i-cache hit (or number of i-cache miss hit)
- number of o-cache hit (or number of o-cache miss hit)
- number of interrupt (or number of exception)

Renesas Design Vietnam Co., Ltd.