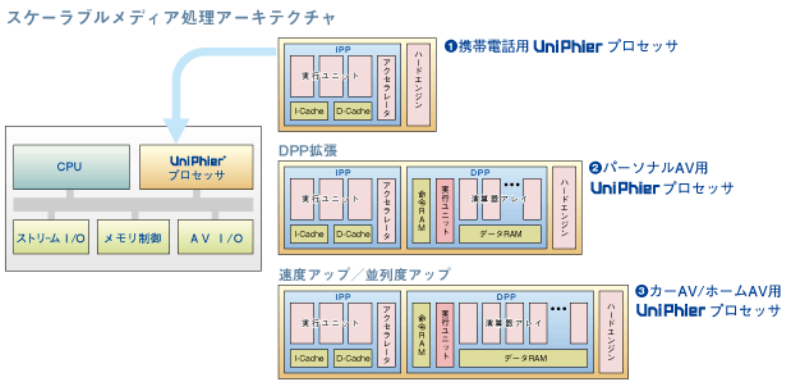
April 6, 2018 by Edahiro

- Introduction
- Our Target
- On this Class
  - 9 Lectures + 3 Exercise
  - Lecture: Lecture＋Quiz　(About 2:1)
    - Handouts (J+E, only E today), Slides (E), Talk (J)
    - When you cannot finish the quiz in class, submit it in Report Box within the day of class.
    - Points of Quiz : Attendance=5, Quiz (Submit=1, Q1=3, Q2=1)
  - Questionnaires twice (Each 10 points)
  - Exercise: Parallel Programming
  - WWW: http://www.pdsl.jp/class/utyo2018/
- First Questionnaire
- What's Multi-Core?

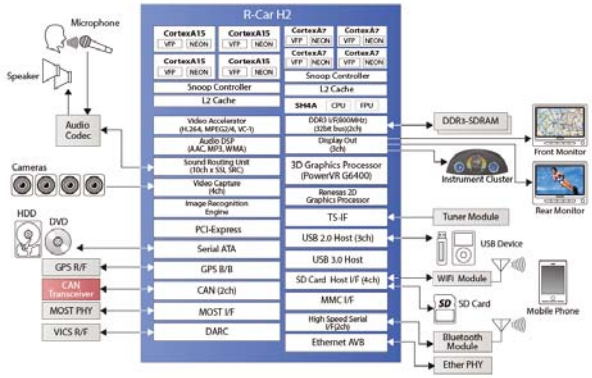
Graduate School of Information Science, Nagoya Univ.
Parallel & Distributed Systems Lab.
PDSL

Multi-Core Everywhere

Multi-Core Processor: Multiple Processors on a SoC



Digital Appliance（Uniphier, Panasonic）



Car Informatics（R-Car H2, Renesas）

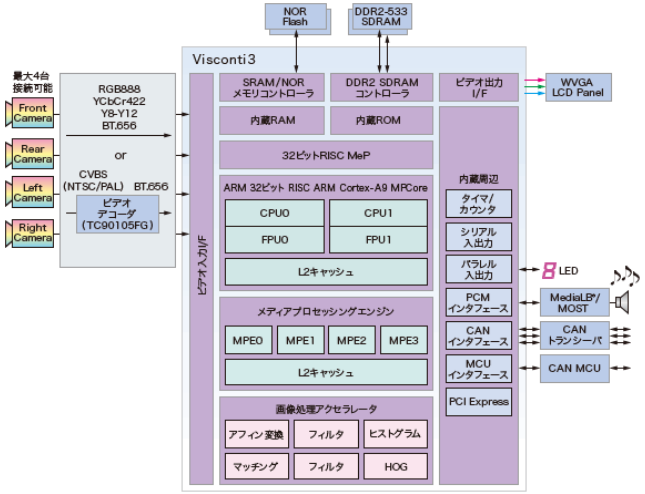
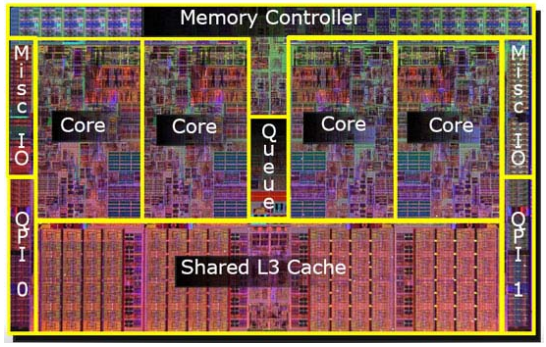


Image Recognition（Visconti, Toshiba）



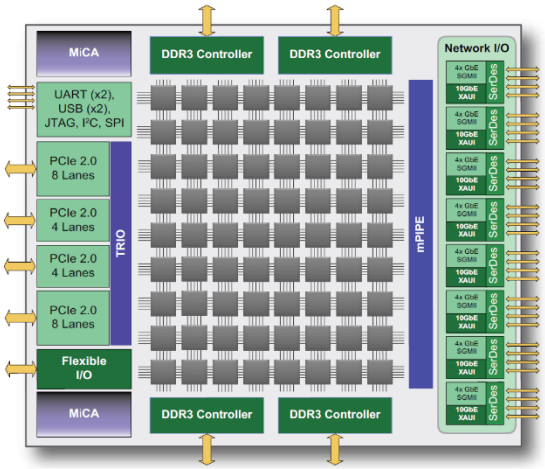
PC, Server（Core i7, Intel）

Many-Core, too

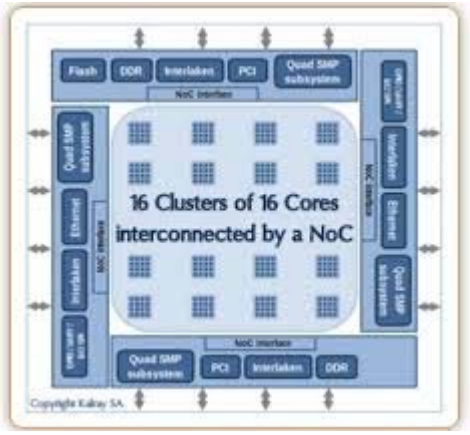
Many-Core Processors: Many Processors on a SoC

Kepler GK210 (Nvidia, 2496 cores)

PEZY-SC (PEZY, 1024 cores)



TILE-Gx72 (Tilera, 72 cores)



MPPA-256 (Kalray, 256 cores)

Multi-Core, Everywhere

- Single CPU: Limit of Performance Scaling
  - Device Minitualization Continues.
  - BUT SELECT “Lower Power” OR “Higher Frequency”
- Multi-Core, Everywhere
  - Severe Power Limitation
  - Multiple Low-Power CPUs
    - Servers, Personal Computers, High-end Embedded Systems
- Software: No More Performance Scaling WITHOUT PARALLELISM
  - Algorithms SHOULD BE SCALABLE for Future Computer Systems

What is Scalable Algorithm?

1. *P-times Smaller Time Complexity with P CPUs*

2. *Comparable Processing Time on a CPU* compared with optimized algorithms for single-CPU processors

3. *Higher Speed-Up with Multiple CPUs (is desirable)*

Barriers to Speed-up

- Not Easy to Achieve Higher Speed-Up on Actual LSI
- Issues to be Care About to Achieve Higher Speed-Up
  - Time Complexity
  - Memory Access
  - Inter-Core Communication
  - Granularity
  - Load Balancing
  - etc.

Contents of This Class

- Our Target
  - Understand Systems and Algorithms on “Multi-Core” processors
- Schedule (Tentative)
  - #1 April 6 (＝Today) What’s “Multi-Core”?
  - #2 April 13 : Parallel Programming Languages (Ex. 1)
  - April 20, 27, May 4, 11, 18: NO CLASS
  - #3 May 25 : Parallel Algorithm Design
  - #4 June 1 (Fri) : Laws on Multi-Core
  - #5 June 8 : Examples of Parallel Algorithms (1) (Ex. 2)
  - June 15: NO CLASS
  - #6 June 22 : Examples of Parallel Algorithms (2)
  - #7 June 29 : Examples of Parallel Algorithms (3)
  - #8 July 6 : Examples of Parallel Algorithms (4)
  - #9 July 13 : Examples of Parallel Algorithms (5) (Ex. 3)
  - (July 20)

Graduate School of Information Science, Nagoya Univ.
Parallel & Distributed Systems Lab.
PDSL

Systems & Algorithms for Multi-/Many-Core Processors

- Introduction
- Our Target
- On this Class
  - 9 Lectures + 3 Exercise
  - Lecture: Lecture＋Quiz　(About 2:1)
    - Handouts (J+E, only E today), Slides (E), Talk (J)
    - When you cannot finish the quiz in class, submit it in Report Box within the day of class.
    - Points of Quiz : Attendance=5, Quiz (Submit=1, Q1=3, Q2=1)
  - Questionnaires twice (Each 10 points)
  - Exercise: Parallel Programming
  - WWW: http://www.pdsl.jp/class/utyo2018/
- First Questionnaire
- What's Multi-Core?

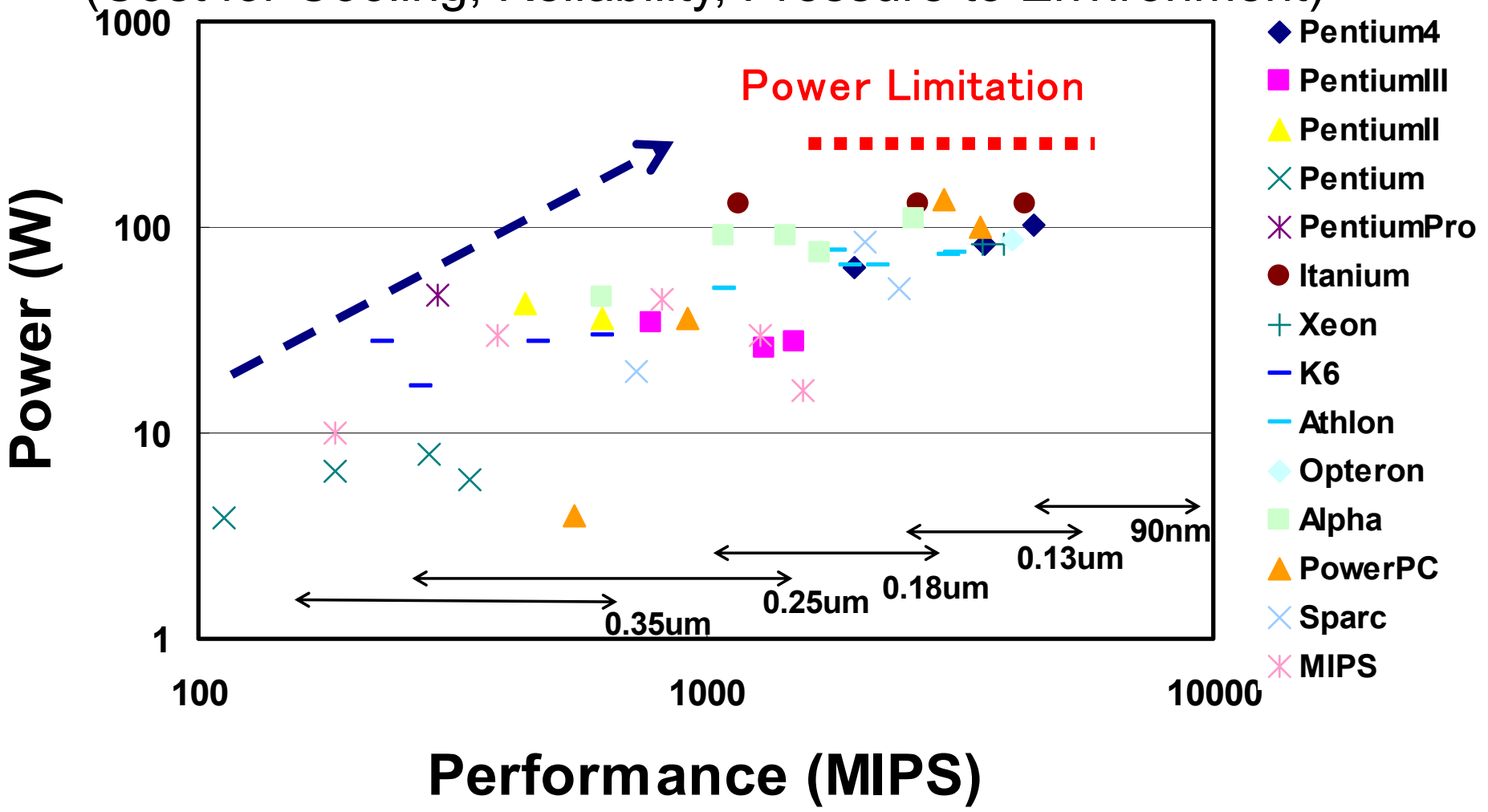
Graduate School of Information Science, Nagoya Univ.
Parallel & Distributed Systems Lab.
PDSL

Outline

- Background
- Multi-Core Architecture and Software Model
- Software Development for Multi-Cores

Background: Power Trend of Processors

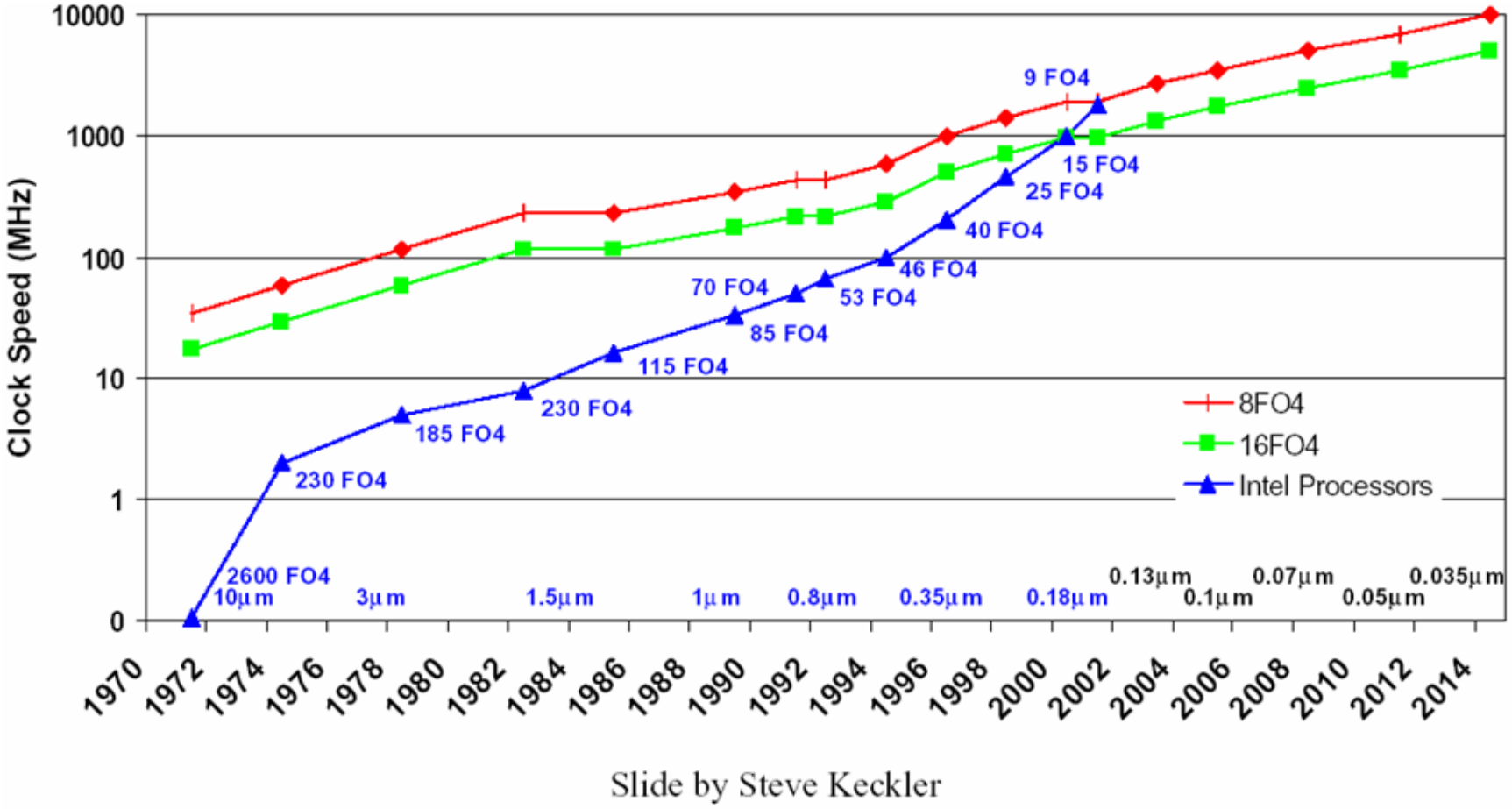
- Achieve 100W. Cannot Increase Power Any More
  (Cost for Cooling, Reliability, Pressure to Environment)



Graduate School of Information Science, Nagoya Univ.
Parallel & Distributed Systems Lab.
PDSL

Background: No More Clock Up

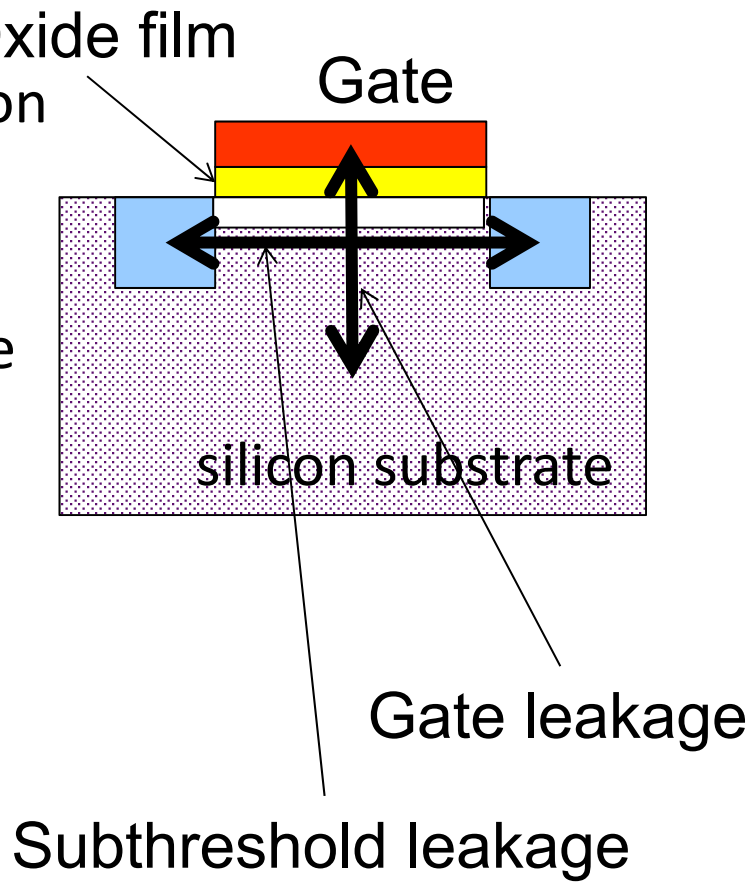
- Already 9FO4 (9 Fan-Out 4) for High-end CPU
- 4FO4 in Critical Path (e.g. ALU)



Slide by Steve Keckler

Background: Leakage Current

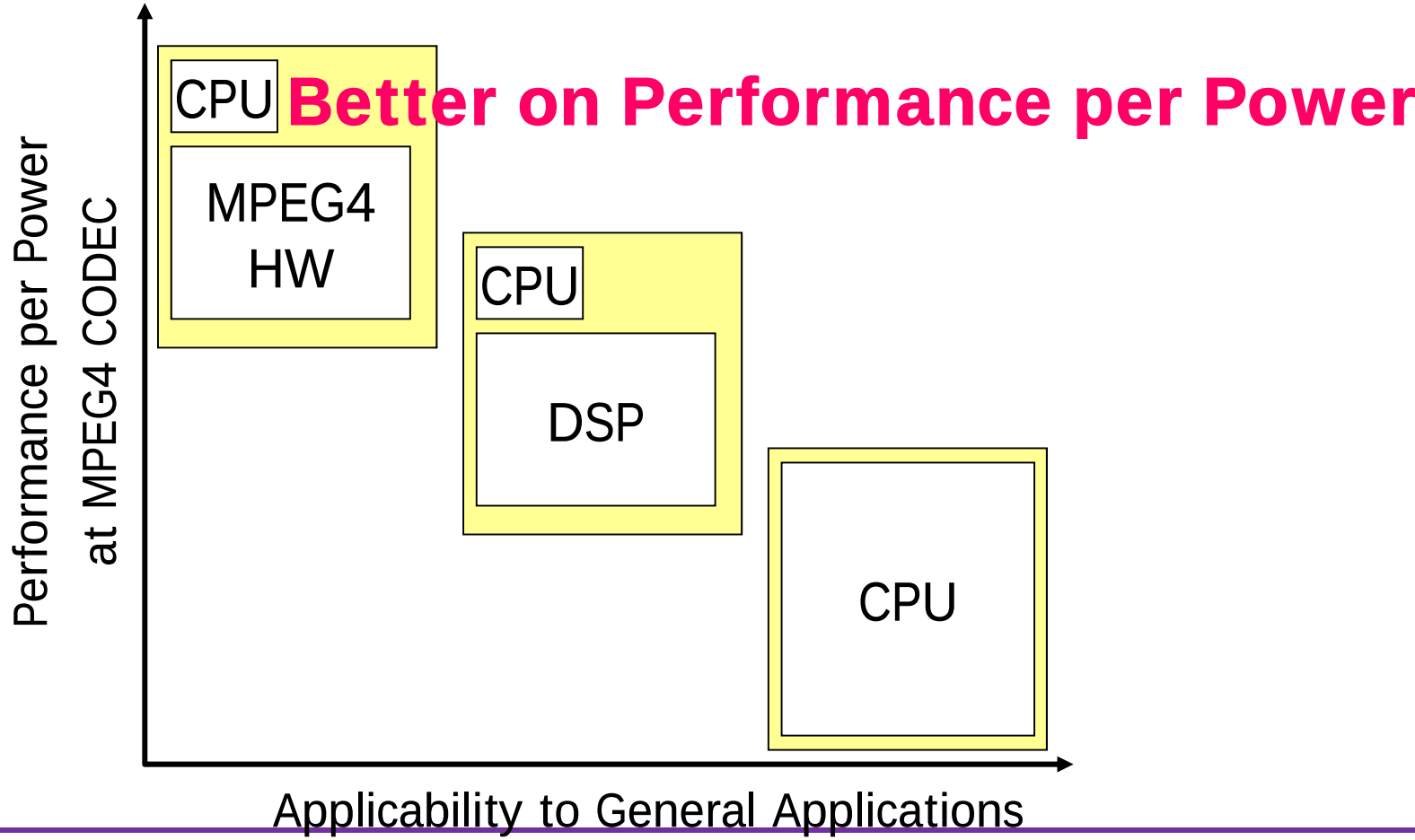
- Leakage Current
  - Leakage of current by miniaturization of transistors. Power dissipation even a transistor is OFF.
  - Large Leakage for high-performance transistors



- Reduction of Leakage Current
  - Use multiple low-power processors with low-leakage transistors
  - All processors are ON for high-performance execution. Otherwise, some of processors are OFF.

Necessity of Multi-Core(Performance per Power)

To Decrease Power for a Specified Functionality, Heterogeneous Multiprocessor is better than a High Performance Single Processor.



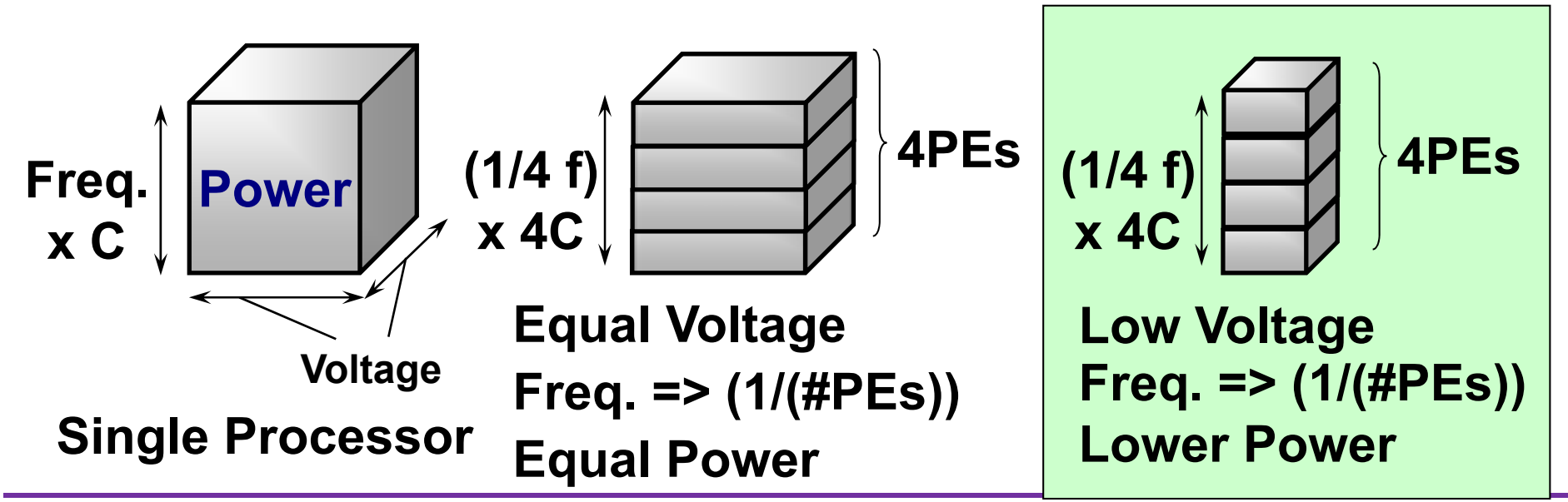
Necessity of Multi-Core (Low Power)

Homogeneous Multi-Core:
Lower Power per Performance
Keep High Performance with Parallelism,
Reduce Power with Low Voltage

Classical Explanation

Power ≅ f x C x V^2

$f \times C \times V^2$ | $(1/4\ f) \times (4 \times C) \times V^2$ | $(1/4\ f) \times (4 \times C) \times (\text{Lower } V)^2$



Necessity of Multi-Core (Low Power)

Multi-Core (N processors):
Peak Performance O(N), Power O(N)
⇒Efficiency（Performance/W) O(1)

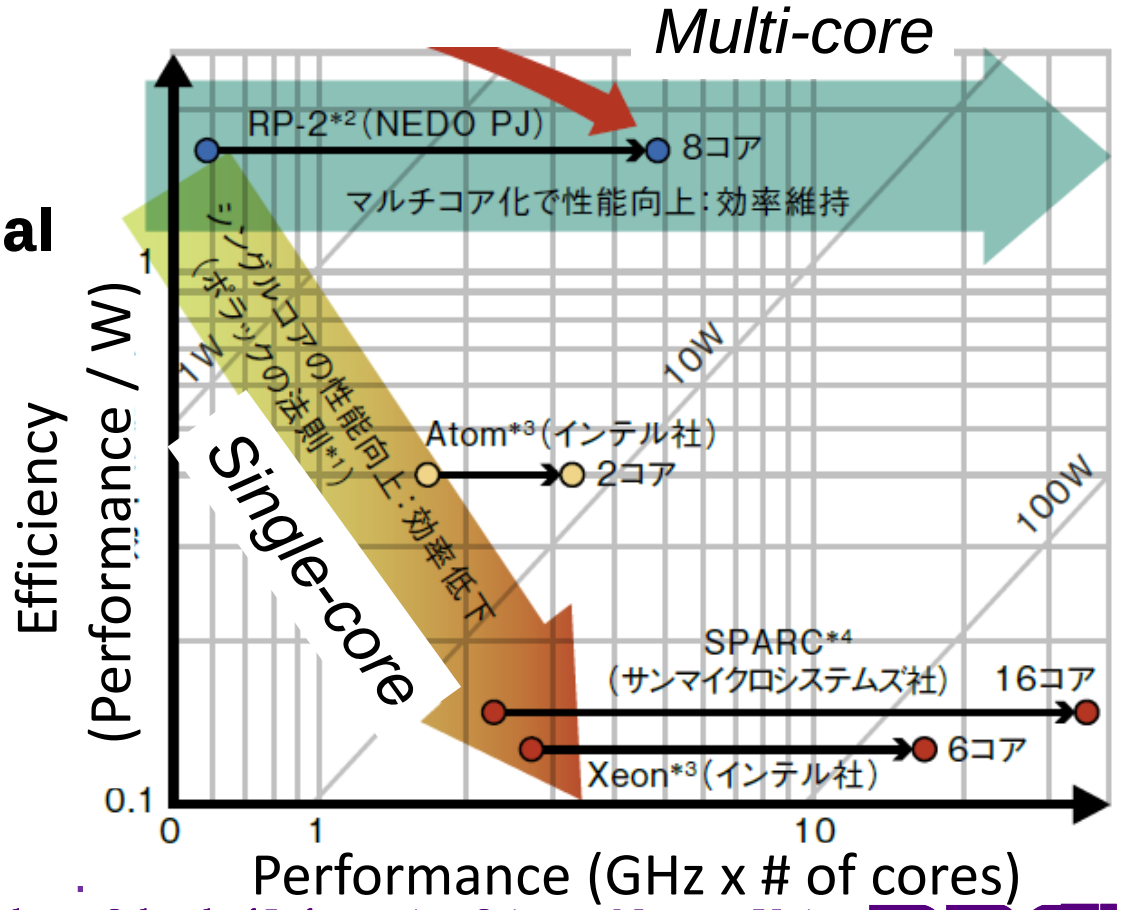
Explanation at
Post-Scaling Era

Pollack's Rule

Performance of Single Processor is proportional to square root of its complexity.

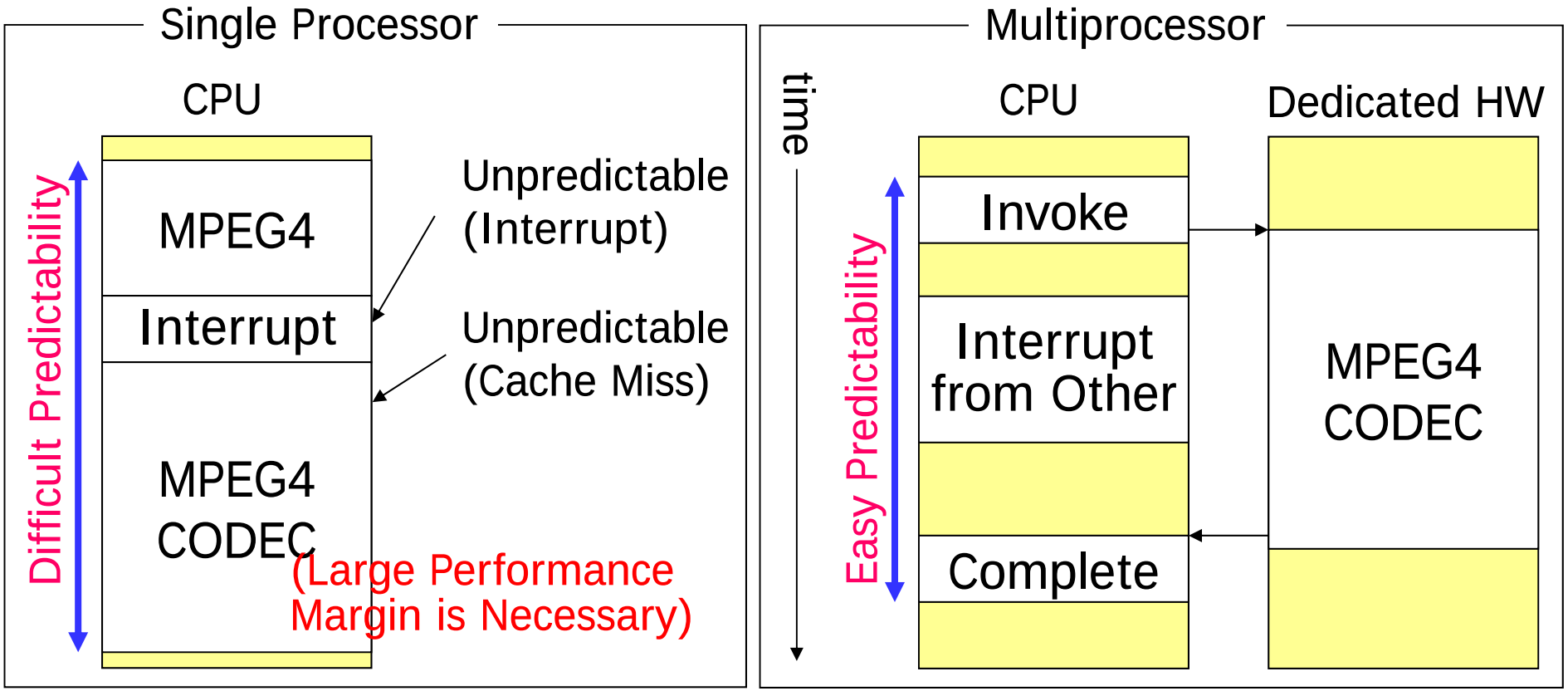
(Empirical)

Single-Core of N times Performance
⇒Complexity (Area, Power) $O(N^2)$
⇒Efficiency O(1/N)



Necessity of Multi-Core (Real-Time Performance)

Heterogeneous Multiprocessor has better Real-Time Performance than a High Performance Single Processor.

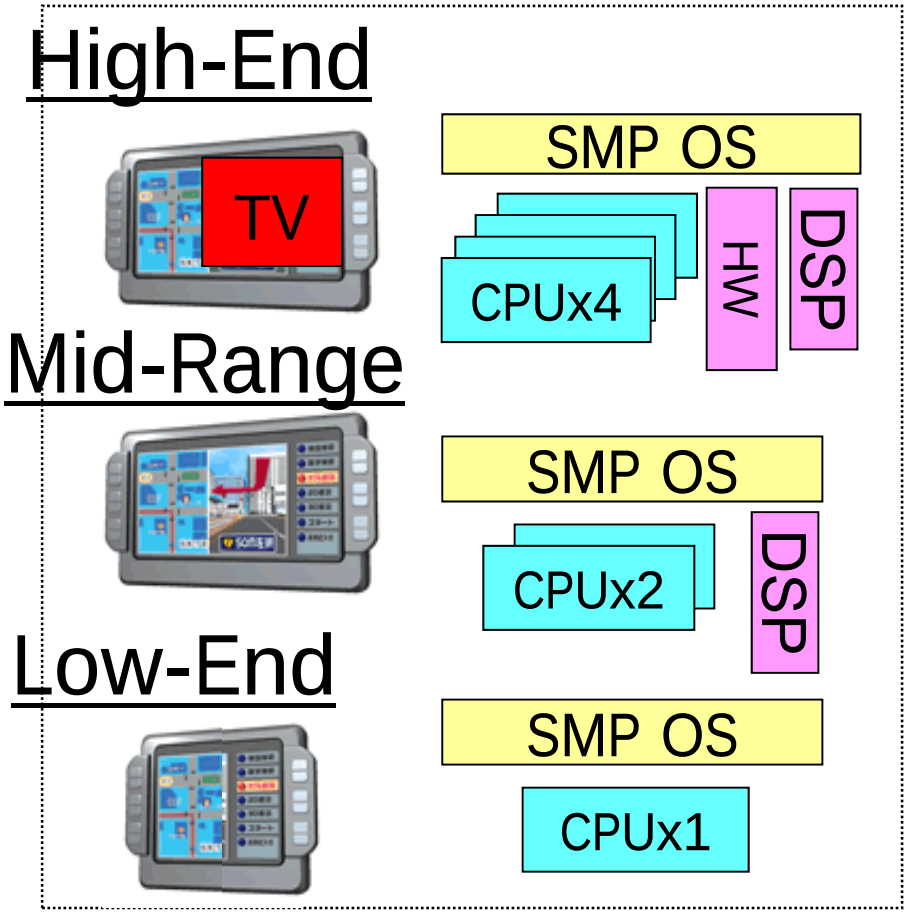


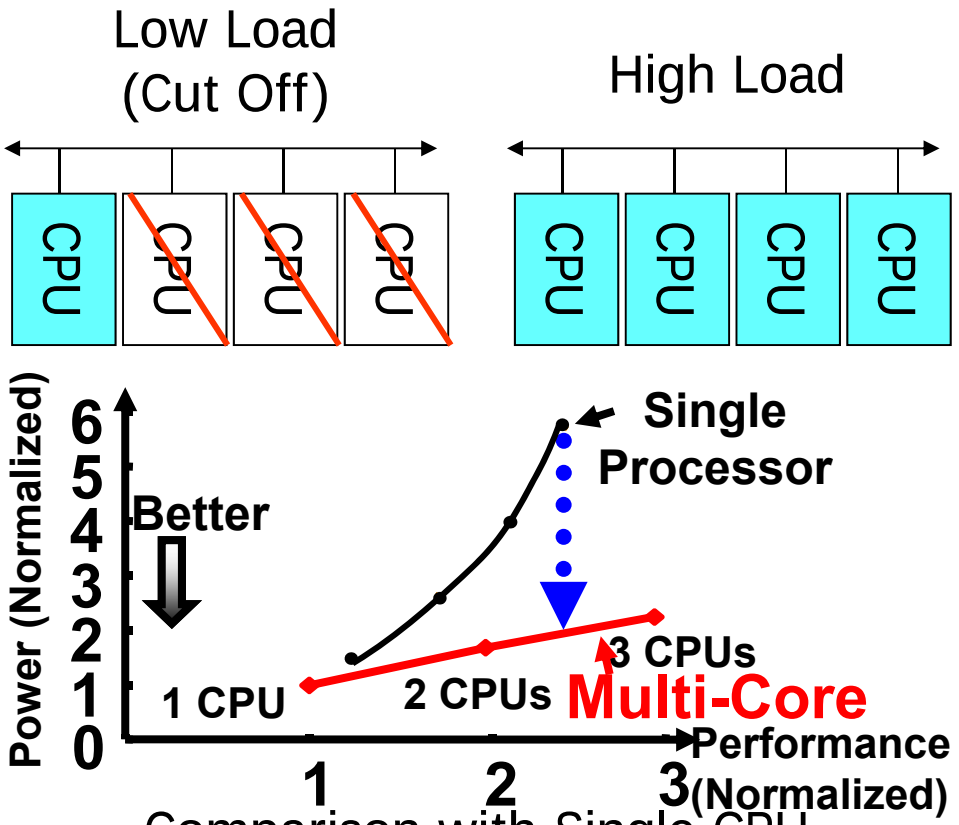
Predictability is Important for Embedded Systems.

Graduate School of Information Science, Nagoya Univ.
Parallel & Distributed Systems Lab.
PDSL

Necessity of Multi-Core (Scalability)

- Single Software Platform for All Products
- Easy Power Control especially for Leakage





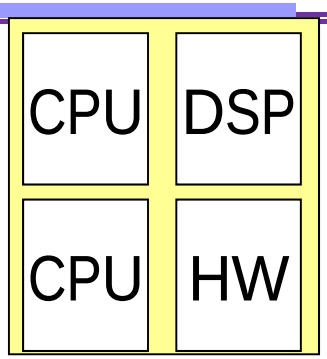
Comparison with Single CPU
on Performance/Power Scalability

Outline

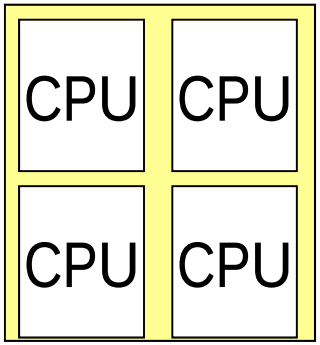
- Background
- Multi-Core Architecture and Software Model
- Software Development for Multi-Cores

Types of Multi-Cores

- Heterogeneous vs. Homogeneous (AMP vs. SMP in Hardware Architecture)
  - Heterogeneous: Multiple Kinds of Cores
  - Homogeneous : Same Cores
- AMP vs. SMP (in System Architecture)
  - AMP (Asymmetric Multi-Processor)
    - Each Core executes Each Software (Functionality Distribution)
  - SMP (Symmetric Multi-Processor)
    - SMP OS distributes Multiple of Software into Cores (SMP is only Homogeneous.)
- Cache Coherency Support is different between Homogeneous-AMP and Homogeneous-SMP
- LSIs for Embedded Systems are MIXTURE of several types. LSIs have several Cores (CPU, DSP, and Hardware Engines), and Some Cores may be Homogeneous AMP/SMP.



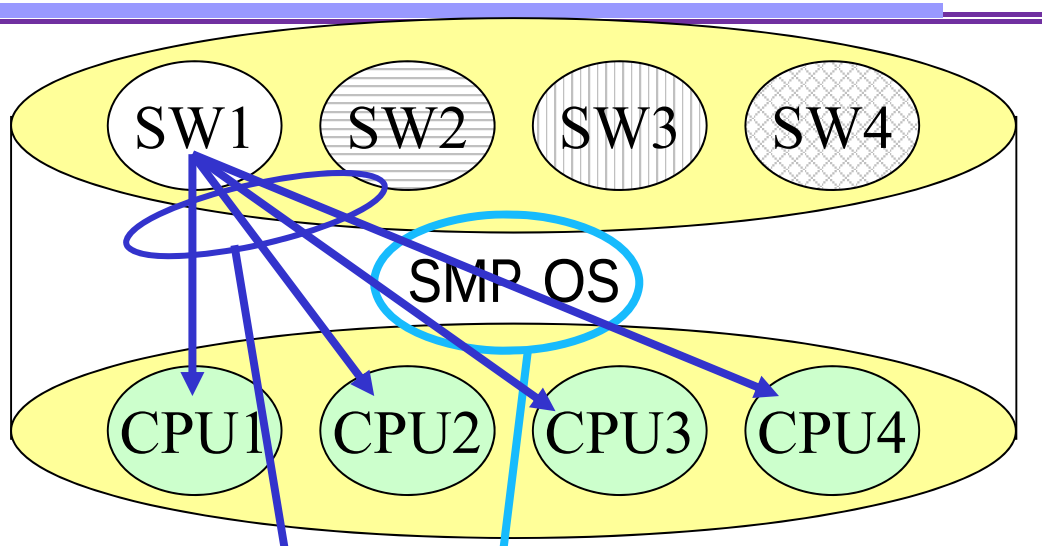
Heterogeneous



Homogeneous

Definition of SMP (Symmetrical Multi-Processing)

- Several types of definition
  Definition here is:
  - SMP="Parallel processing methods, managed by a single OS, on which all processes (tasks, threads, etc.) are symmetrically executable on all CPUs"
  - AMP = not SMP
- Binary codes have to execute on all CPUs.
  - All CPUs should be the same (at least instruction set) = Homogeneous
  - Difference between homogeneous AMP and SMP is Cache coherency
  - Cache Coherency?



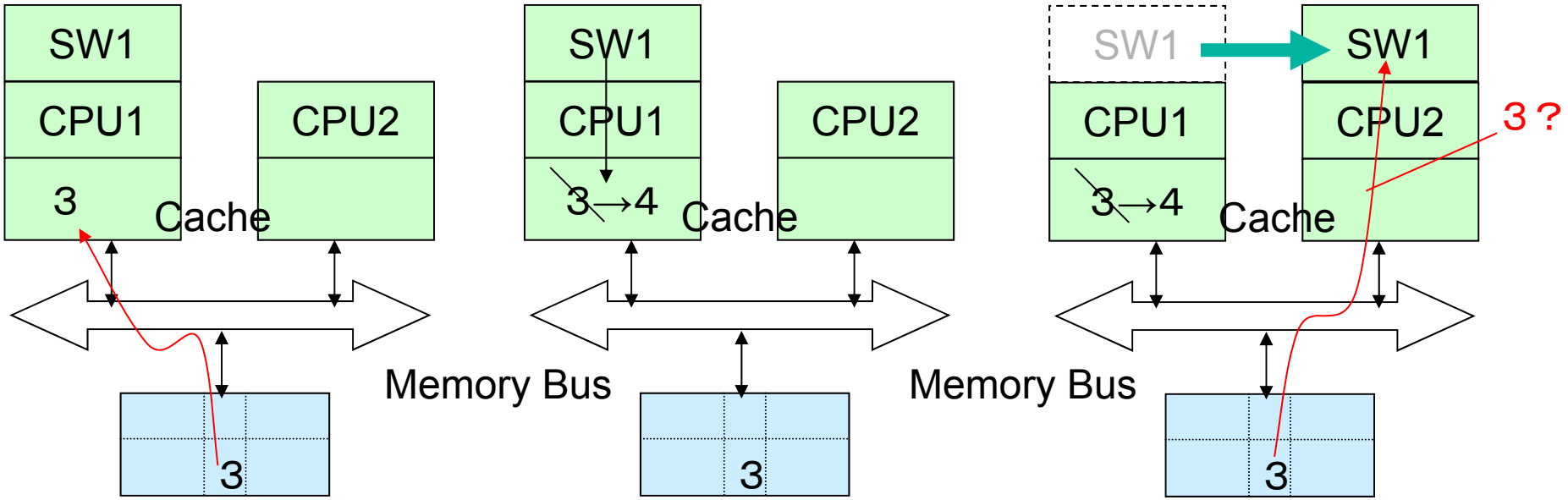
Point #1: Single OS

Point #2:
Symmetrically Executable = Migratable to other CPUs

(Tasks suspended at CPU1 can be restart at CPU2. A problem can be data on cache.)

Cache Behavior and Symmetrical Execution

- When SW1 is migrated from CPU1 to CPU2, an error could take place if cache has data for SW1.



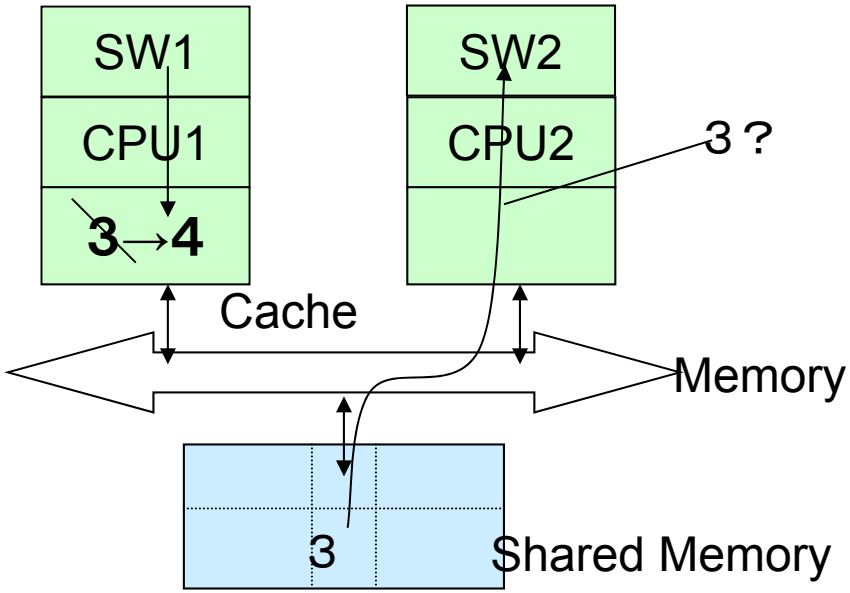
1. SW1 read value 3 from memory.

2. SW1 update the value to 4.

3. SW1 read wrong value from the memory after SW1 is migrated from CPU1 to CPU2 if the value 4 remains in the cache.

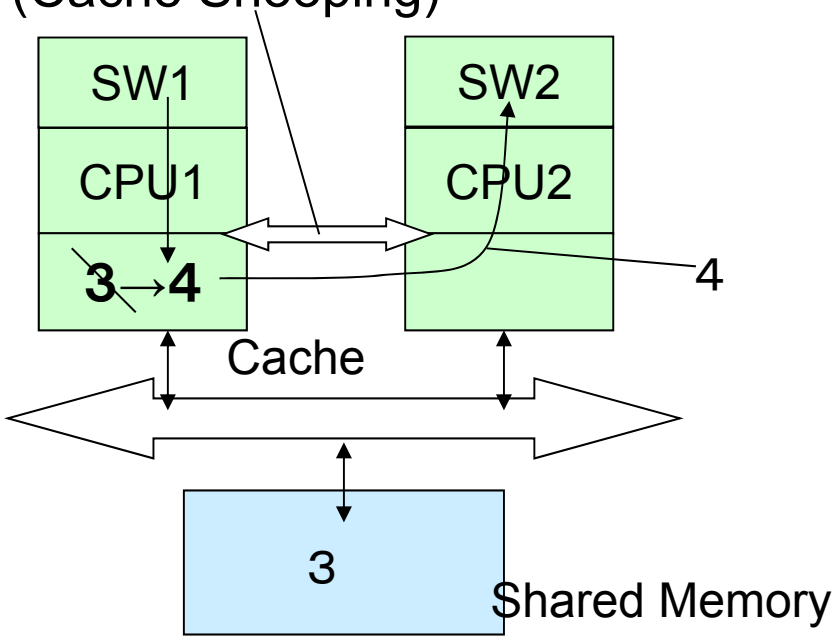
Difference between AMP and SMP (Cache Coherency 1)

AMP:
(No Hardware Support)



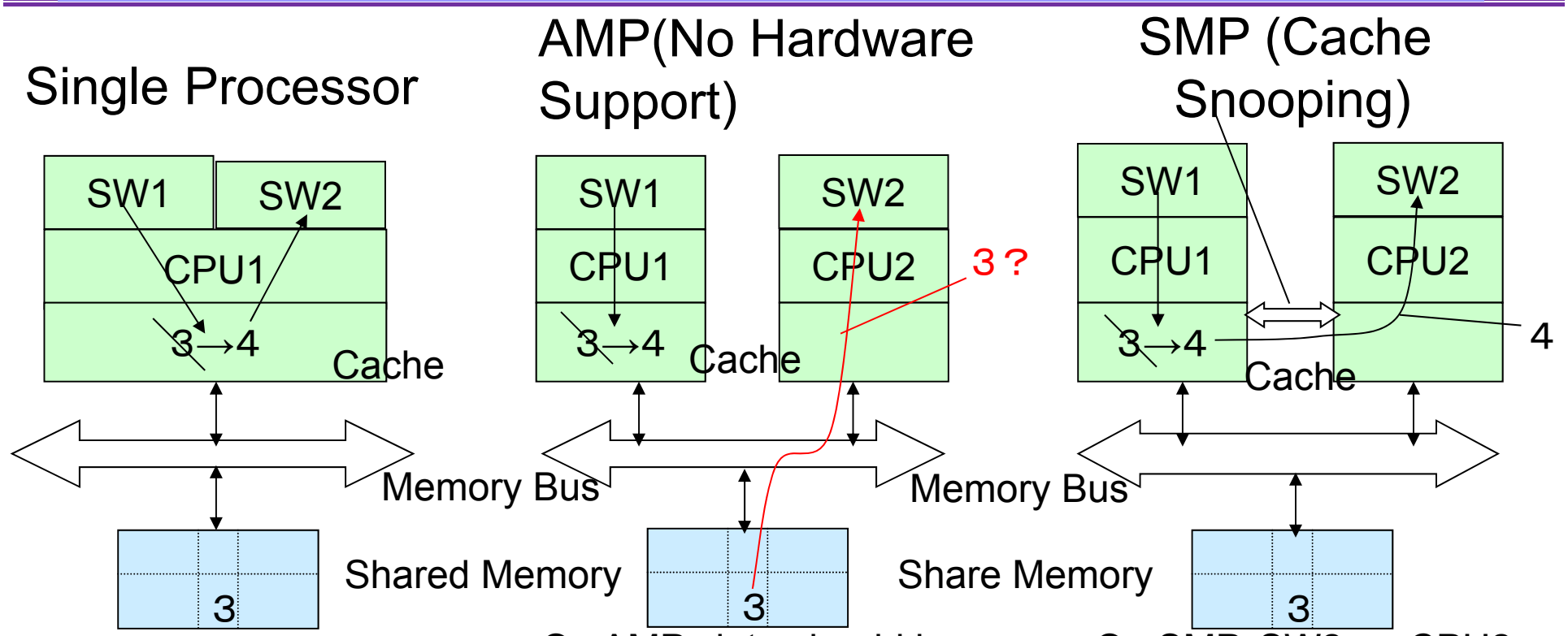
When SW2 uses data generated by SW1, SW1 MUST write the data to shared memory.
The operation should be written in software, and may have large overhead on AMP.
(No Hardware Support)

SMP:
(Cache Snooping)



On SMP, Software does not need to care about Cache Coherency because of Hardware Support such as Snooping mechanism. (You still may care about some latency of inter-cache communication.)

Difference between AMP and SMP (Cache Coherency 2)



On Single Processors, SW1 and SW2 easily share data without overhead because they share cache.

On AMP, data should be stored to memory from cache before SW2 uses. Programmers need to write “communication” from SW1 to SW2 explicitly.

On SMP, SW2 on CPU2 can share data with CPU1 by snooping. Programmer do not have to rewrite their programs and there is less overhead than AMP.

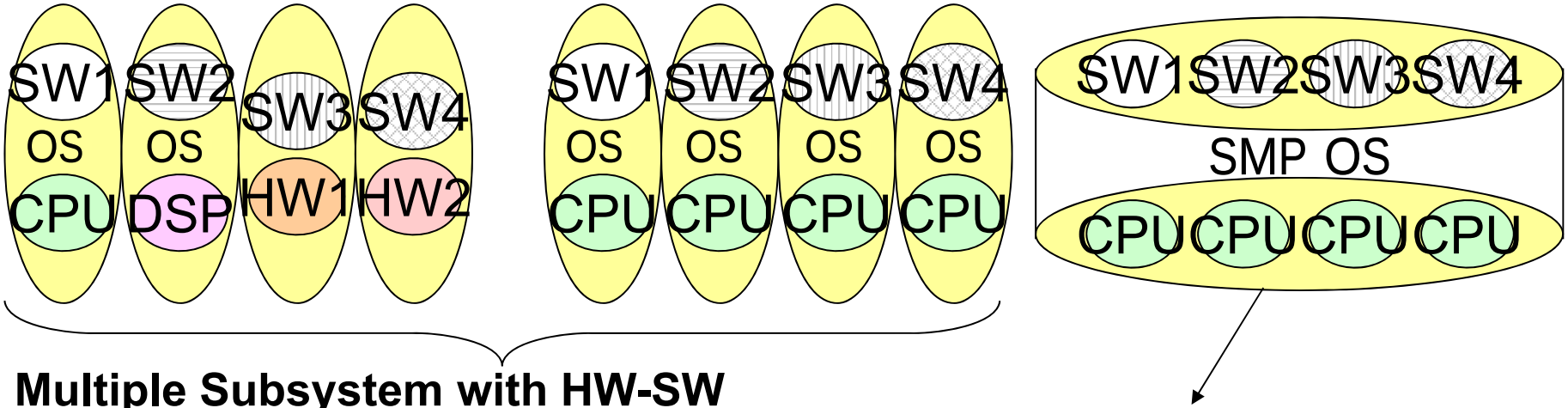
Difference between AMP and SMP (Summary)

- SMP="Parallel processing methods, managed by a single OS, on which all processes (tasks, threads, etc.) are symmetrically executable on all CPUs"
  - SMP should be homogeneous.
  - With SMP, all software is executed on all CPUs under dynamic load balancing by a single OS.
  - On AMP, each CPU has OS(*), and software statically assigned to CPUs.

    * Including monitors, BIOSs
  - Therefore, on AMP, each CPU becomes a subsystem.

Types of Multi-Cores

- Heterogeneous-AMP
- Homogeneous-AMP
- Homogeneous-SMP



Multiple Subsystem with HW-SW

- Change into Subsystem has a small Influence on Others. → Good for Real-Time Performance, Test of Subsystem
- Select Best HW for each Subsystem
- Homogeneous-AMP is used for Separat of Subsystems with a single Software Development Environment.
- Fixed Assignment between SW and HW → may cause Load Imbalances

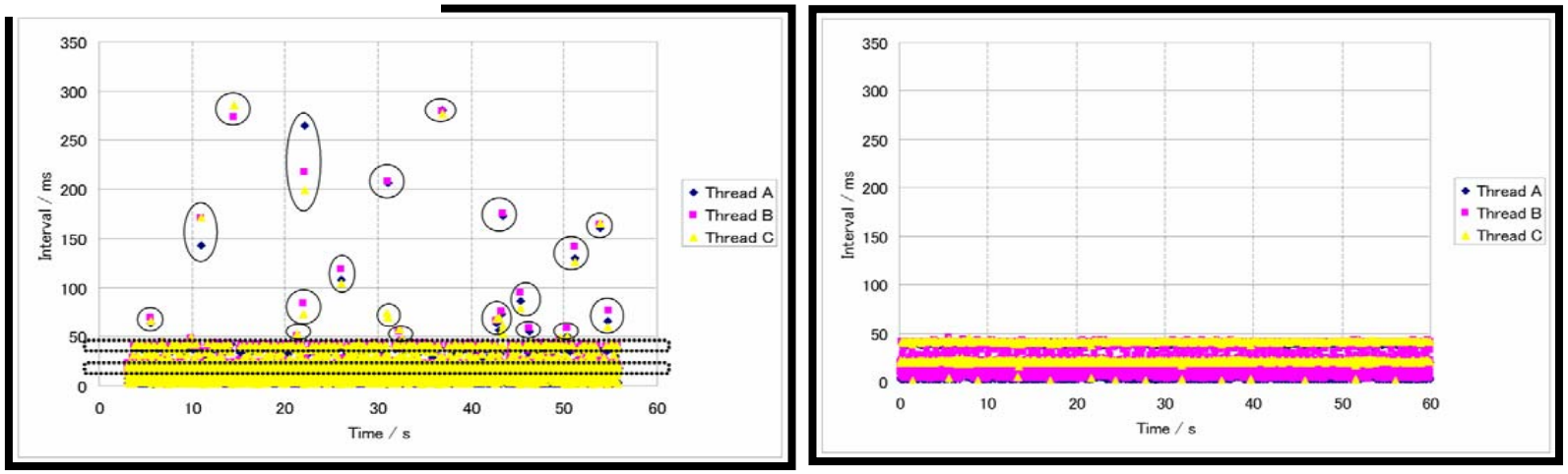
SMP OS Dynamically Maps SW Modules to HW Modules.

- Change of SW Module Influences on a whole of System → Bad for Real-Time Performance, Test of Subsystem
- Dynamic Mapping between SW and HW is effective to Better Load Balances.

Real-Time Performance on AMP-type Multi-Core

Delay from Scheduled Time (= Points above 40ms)
→Discontinuity of Audio & Video

Execute Time for Periodical Processes



a) 1CPU → Time ↔ b) 3CPU

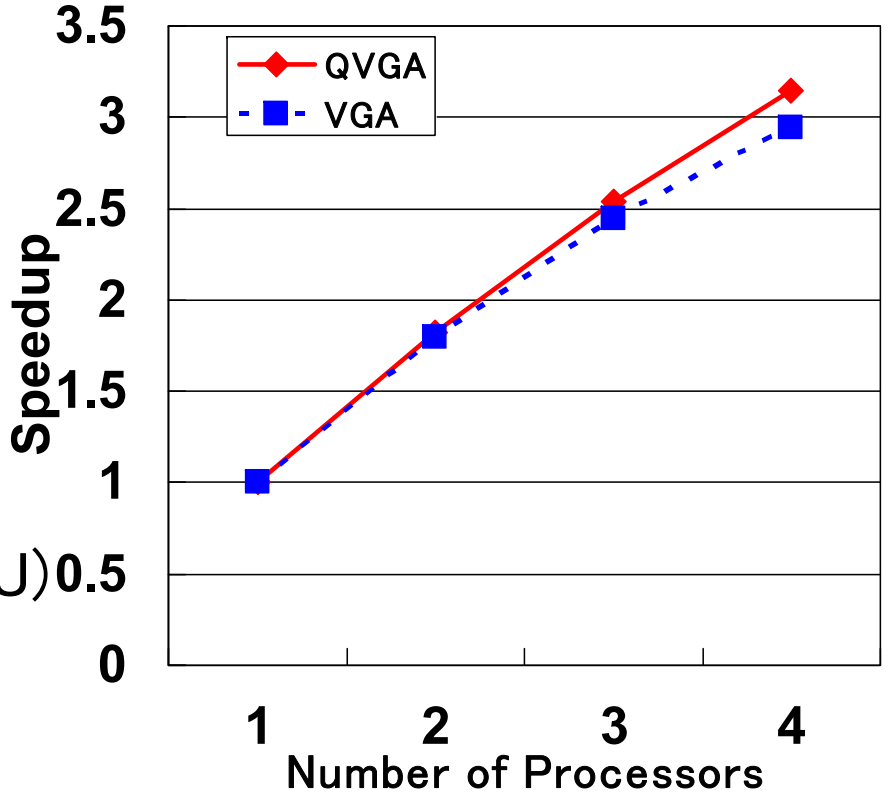
Many Delays

NO DELAY

Scalability on SMP-type Multi-Core

- Image Stabilizer
  - Slow Shutter → Blurry Image (I)
  - Fast Shutter → Dark Image (II)
  - Multiple Shots + Image Processing (III)
  - Over one-giga Operations

- Speed-Up Ratio (1CPU vs. 4CPU)
  --- VGA: 2.94, QVGA: 3.15



(I) slow shutter: bright but blurry

(II) fast shutter: dark but not blurry

Image Stabilizer

(III) bright and not blurry

Heterogeneous-AMP: UniPhier (Panasonic)

(Heterogeneous Configurable Array Media Processors)

Media Processor (Uniphier Processor) with Instruction Parallel Processors (IPP), Array Extendible DPP, and Hardware Accelerator, Controlled by Single Host Processor.

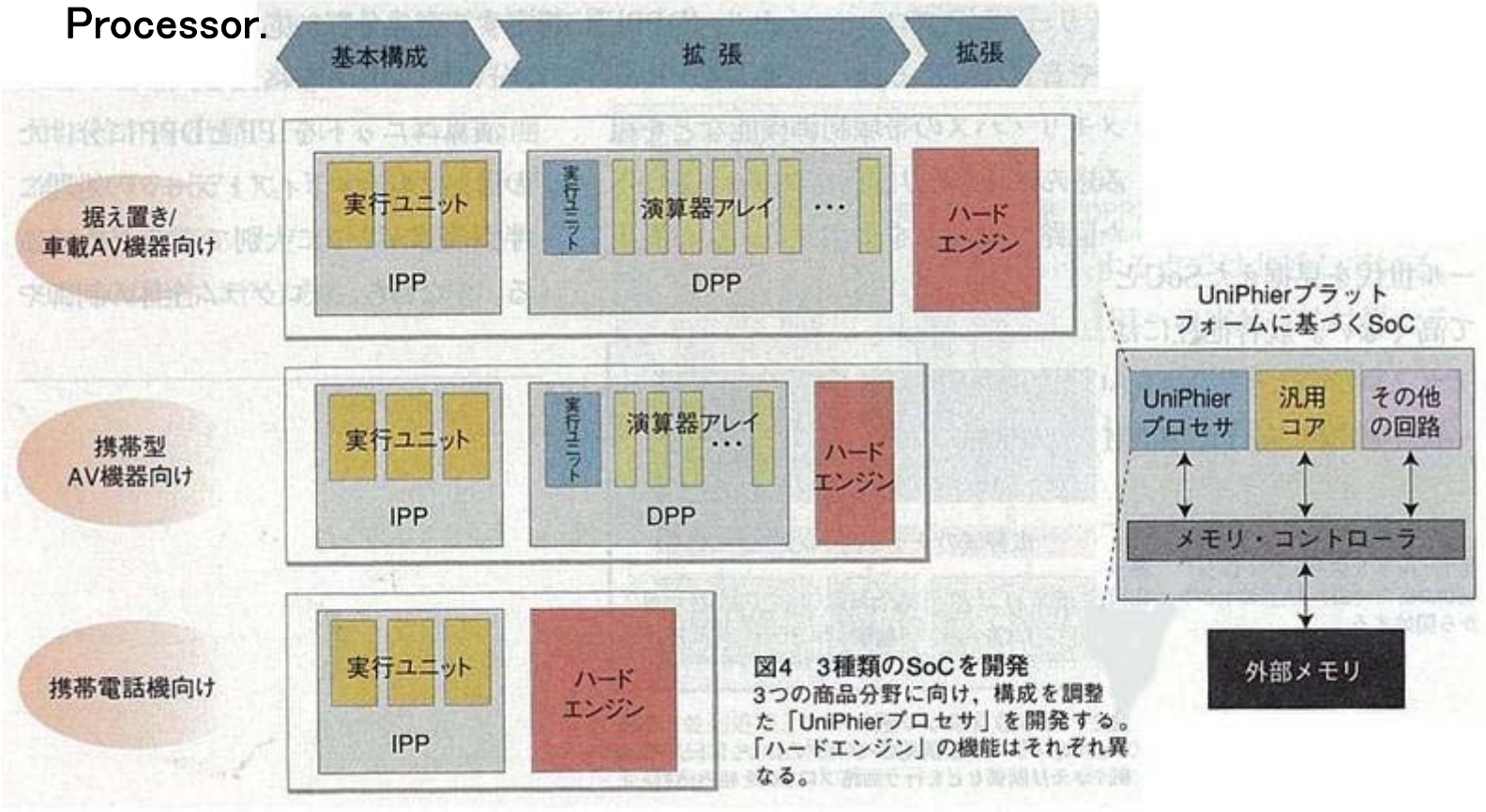


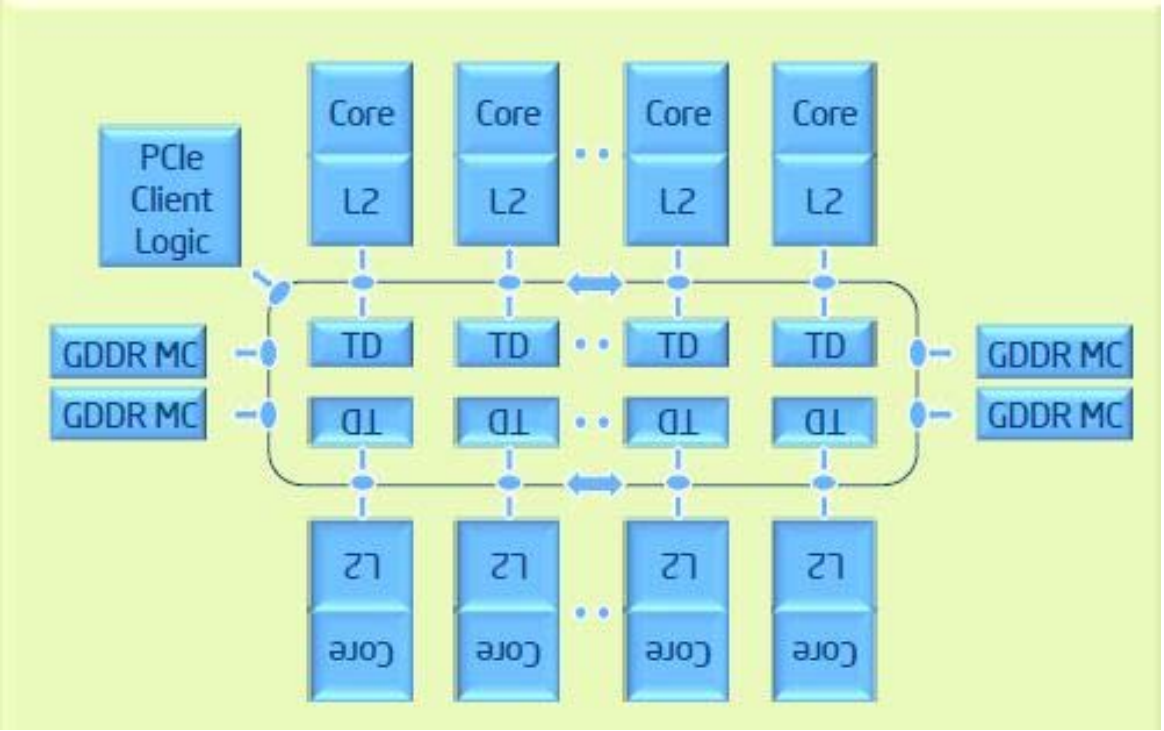
図4 3種類のSoCを開発
3つの商品分野に向け，構成を調整した「UniPhierプロセサ」を開発する。「ハードエンジン」の機能はそれぞれ異なる。

- Low Power, Low Cost, Single-Chip Embedded Processors

Nikkei Electronics 2004.10.11

Homogeneous-SMP: Intel MIC (Many Integrated Core)

- 57 – 61 cores (Xeon Phi) connected by coherent cache



http://wccftech.com/intel-xeon-phi-coprocessor-architecture-details-revealedintel-xeonphi-coprocessor-architecture-details-revealed/

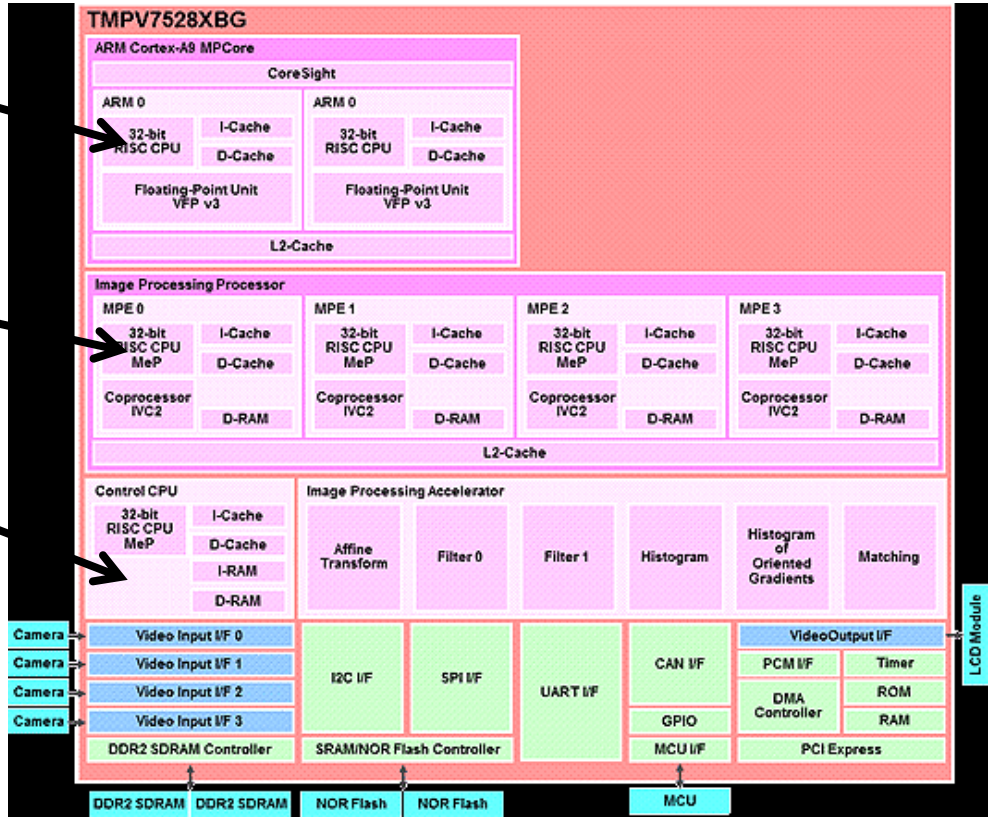
Toshiba Visconti3

- **Image Recognition SoC for Automobile**

Homogeneous SMP with 2 cores of ARM Cortex-A9

Homogeneous AMP with 4 cores of Media Processing Engine (MPE)

Heterogeneous AMP with control CPU and several engines

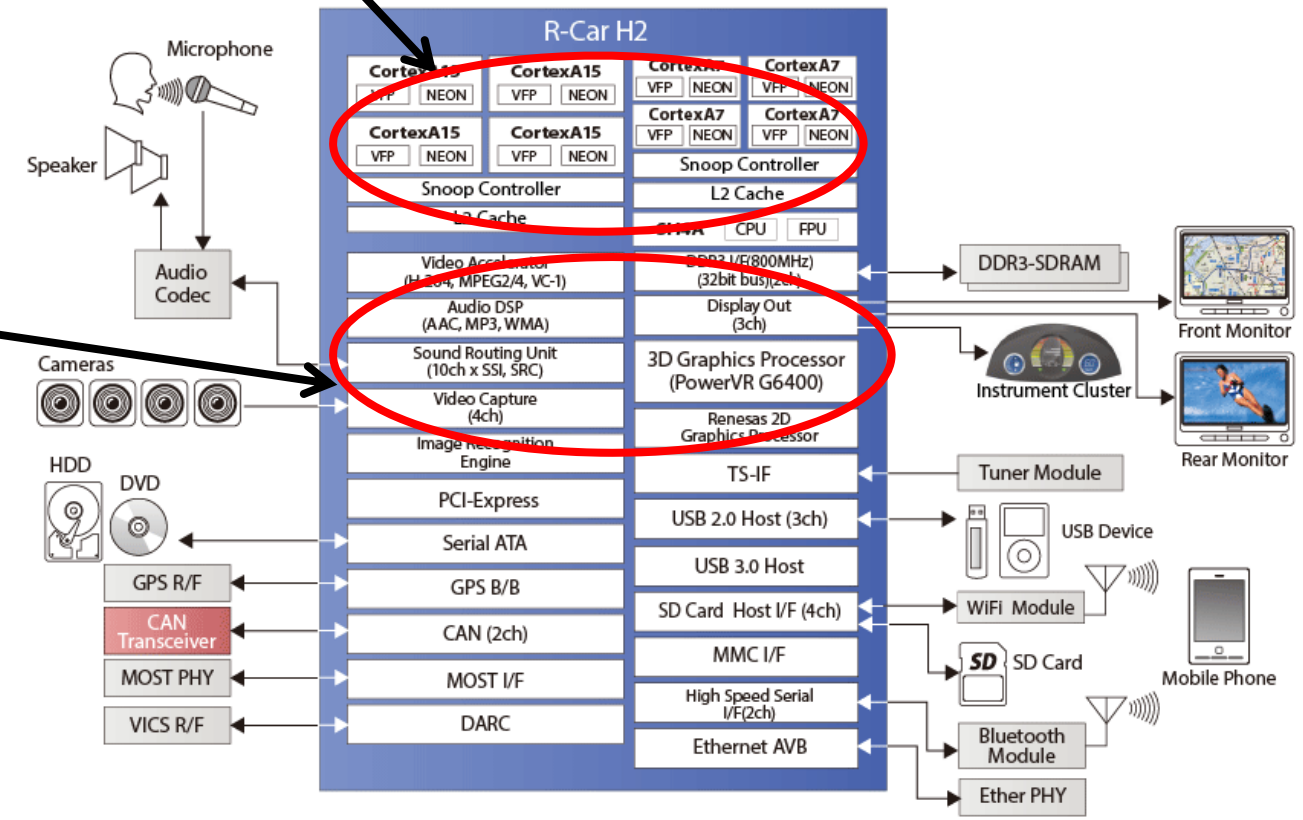


http://toshiba.semicon-storage.com/jp/product/automotive/image-recognition.html

Renesas Electronics R-Car H2（Car Navigation SoC）

Homogeneous SMP with 4 cores of ARM Cortex-A15 or A7.
Switch between A15 (High-Performance) and A7 (Low Power)

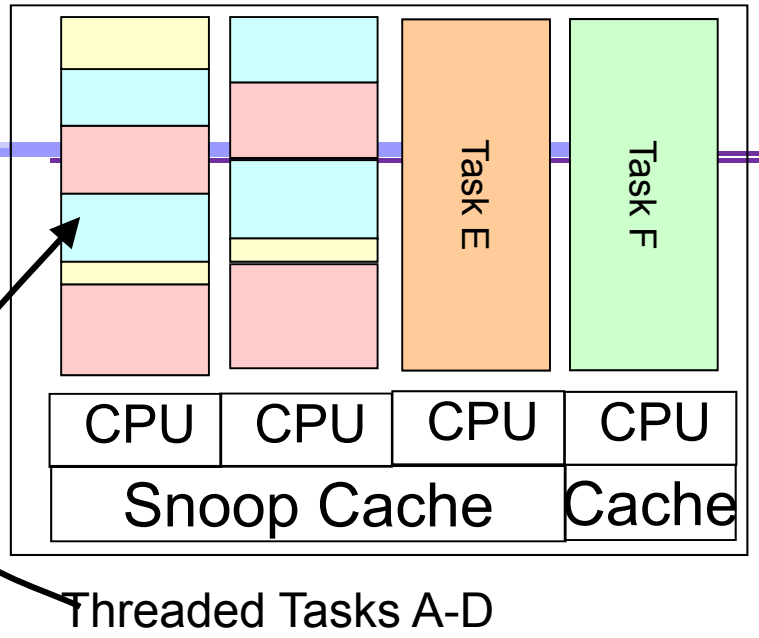
Renesas SH-4A CPU controls engines (Heterogeneous AMP)

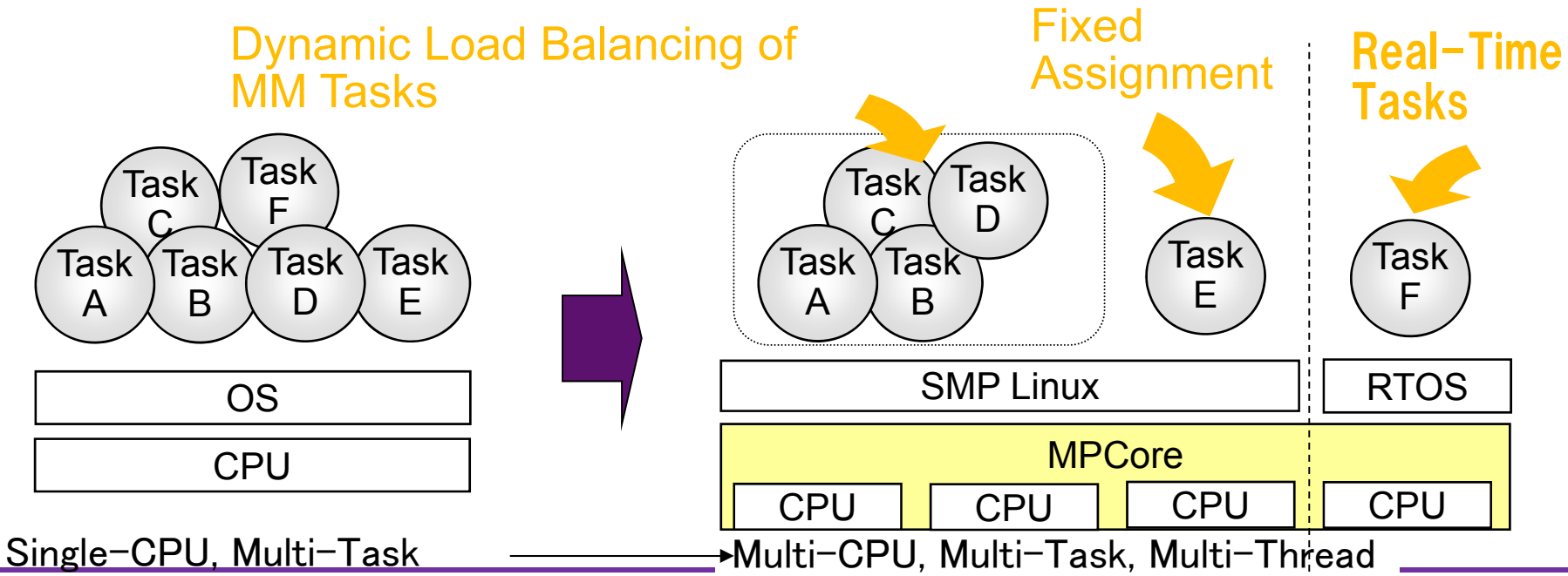


http://japan.renesas.com/applications/automotive/cis/cis_highend/rcar_h2/index.jsp

Achieve Both Scalability and Separation

- Affinity of SMP Linux: Assignment of Tasks to Fixed CPU.
- AMP-SMP Hybrid
- Good for Future Embedded Systems that have both Real-Time Tasks and Heavy Multimedia Tasks.





Outline

- Background
- Multi-Core Architecture and Software Model
- Software Development for Multi-Cores

Software is Key Issue for Multi-Cores

- "Multi-Cores" is needless when software does not utilize multiple cores.
- Concurrent / Parallel Execution of Systems or Software
  - Concurrent execution of multiple software in systems on multiple cores (Functional Distribution)
  - Parallel execution of a single software
- Software Development to utilize multiple cores effectively
  - Algorithms or Logics of Systems need to be parallelizable or to have concurrency.
  - Program codes have to be written for parallel / concurrent execution.

Parallelizable Algorithm (1)

- Speed-up on multi-cores ONLY IF algorithms are parallelizable
  - Example: Summation (1 to 1000000)

```
int i, sum;
for (i=1,sum=0; i<=1000000; i++) {
        sum += i;
}
return sum;
```

  - Difficult to parallelize because of dependency: *sum* at *i=k* is calculated with *sum* at *i=k-1*.

Parallelizable Algorithm (2)

–Example: Summation (1 to 1000000 on 4 cores)

```
int i, j, p, sum;
for (p=0,sum=0; p<4; p++) {
    for(i=0,j=p*250000+1,s=0; i<=250000; i++,
j++) {
        s += j;
    }
    sum += s;
}
return sum;
```

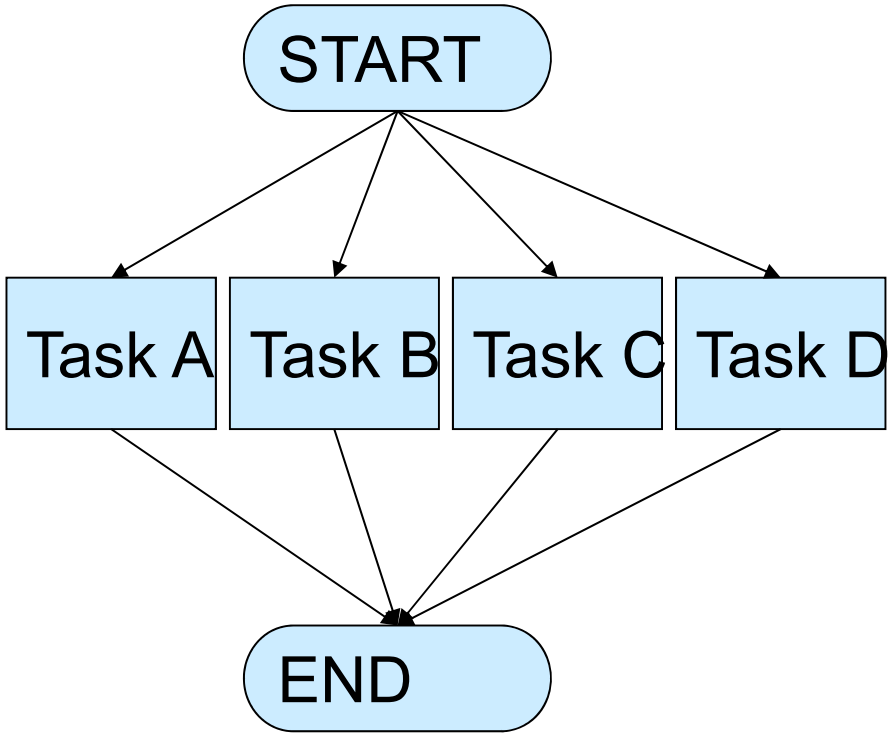
– Executable in Parallel on 4 cores. Take local variables *i, j, s* at each core, and calculate *sum* with results of cores at last.

◆Note

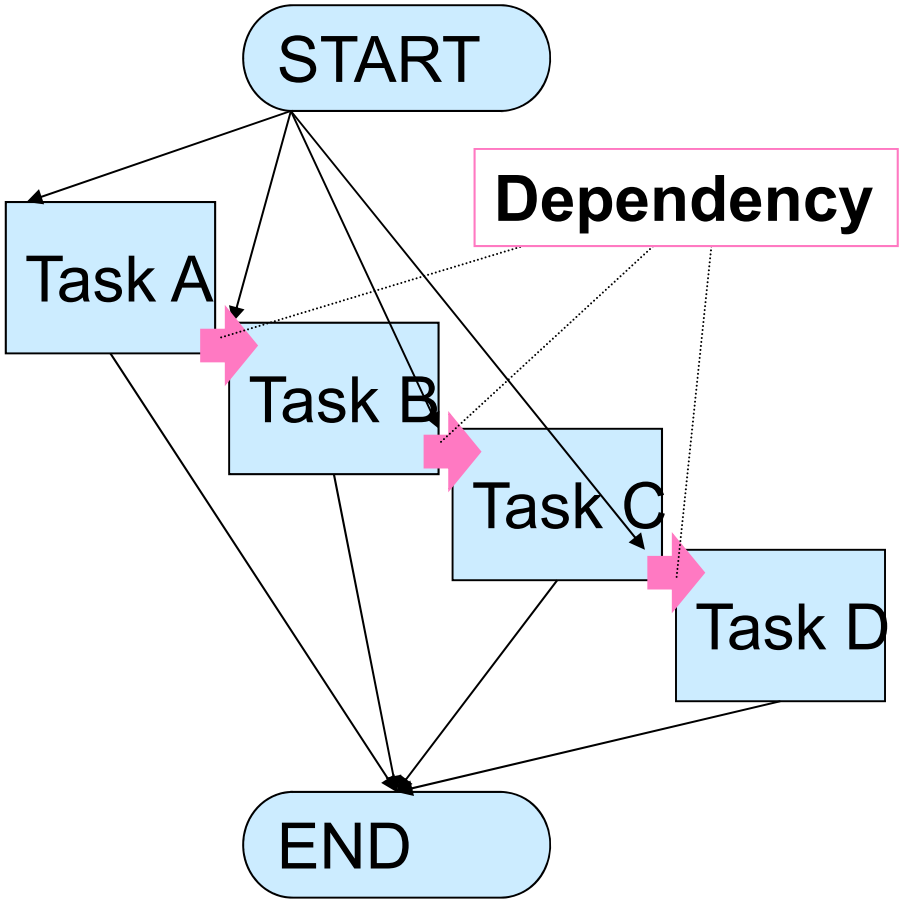
Current automatic parallelizing compiler can parallelize above example. Software Engineers need to rewrite their programs for more complicated cases.

Concurrent System

Speed-Up

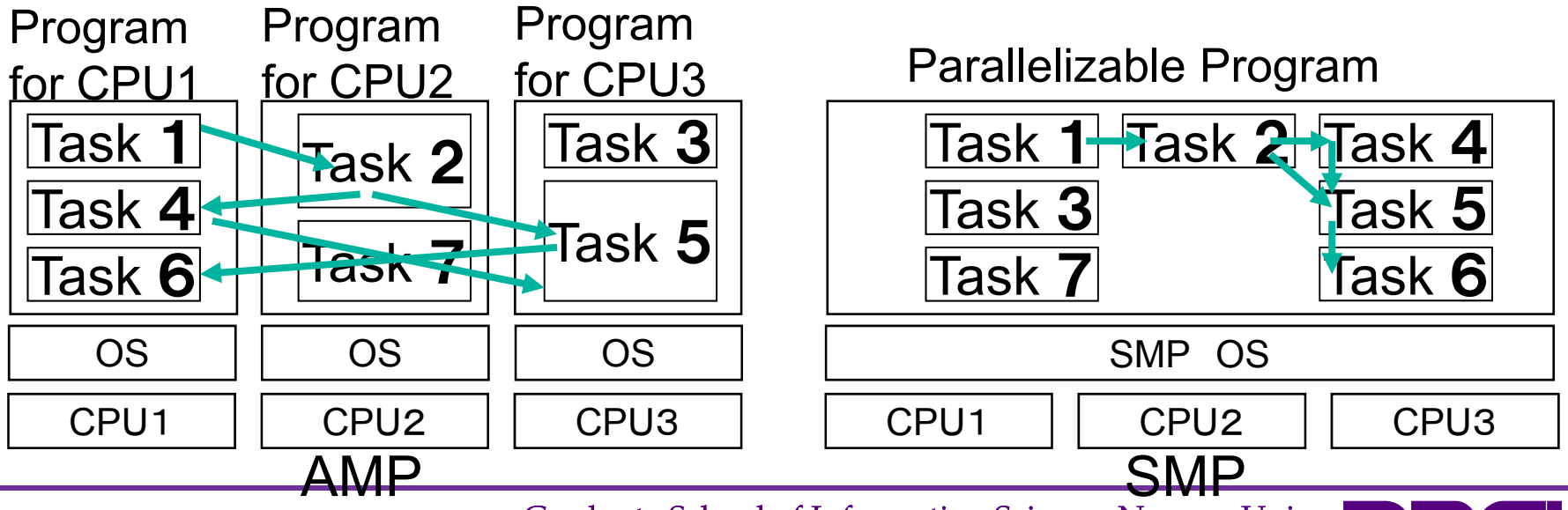


Not Speed-Up



Parallel / Concurrent Executable Program Codes

- AMP and SMP
  - In AMP, each CPU has separated program (on each OS), and synchronization / communication between programs are written. Therefore, tasks are statically assigned to CPUs.
  - In SMP, a single program execute a single SMP-OS on multiple CPUs. Synchronization / Communication are written with parallel program languages or APIs. SMP-OS dynamically assigns tasks (threads) to multiple CPUs.

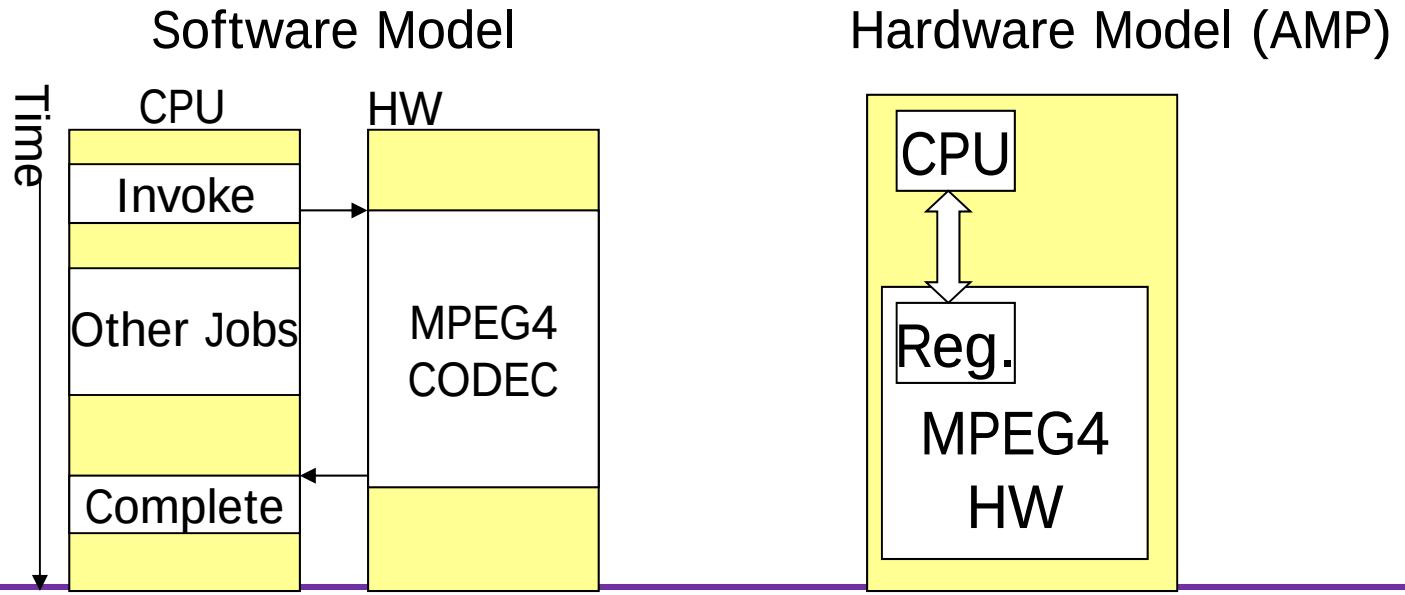


Parallel / Concurrent Executable Program Codes

- AMP
  - Ordinary program for single CPU except synchronization / communication
    - e.g. Task Invocation / Completion, Communication using Shared Memory
- SMP
  - Thread programming
- Other
  - For Heterogeneous Multi-Cores
    - Ex. OpenCL
  - For GPGPU
    - Ex. CUDA

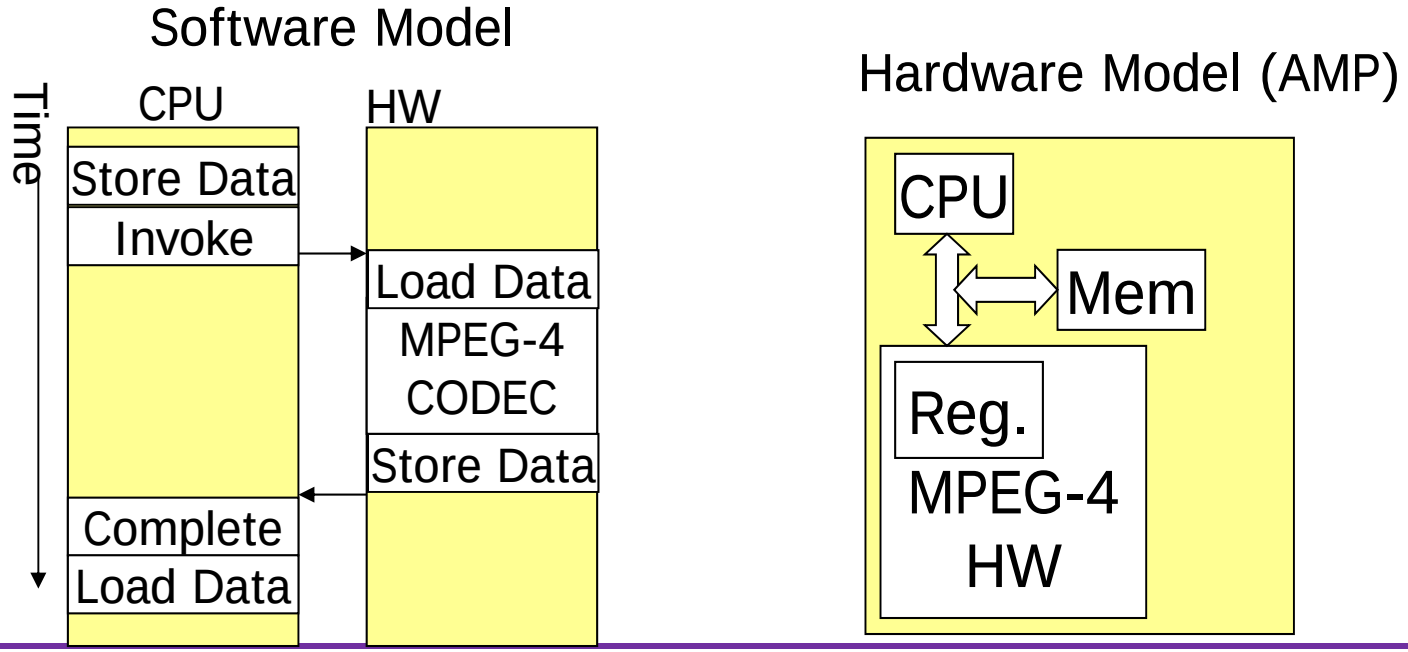
Task Invocation and Completion for AMP

- Invoke: Write to a register in Dedicated HW
- Complete:
  - Interrupt CPU from HW
  - Poll HW Register from CPU
- Invocation/Completion require some time overhead.



Communication using Shared Memory for AMP

- Data Send/Receive among Processors
  - Before using Data at HW, the Data should be written in Memory from Cache
  - Require some time overhead due to Cache Invalidation etc.



Thread Programming for SMP

- Examples of Thread Library
  - pthread
    - IEEE POSIX Section 1003.1c
      POSIX: Portable Operating System Interface
    - Standard in Linux, etc.
  - Windows API (for Windows)
  - Java Thread
    - Define in Java Language
  - OpenMP
    - Directives in C/C++/FORTRAN that enable to write 'Parallel'
    - Developed by Compiler Vendors in US
    - Supported by Software Development Tools for PCs

References: Nichols, Buttlar, and Farrell: Pthreads Programming, O'REILLY, 1998.
: Oaks and Wong: Java Thread Programming, O'REILLY, 1997.
: http://www.openmp.org/