
# Embedded Operating Systems

Che-Wei Chang

chewei@mail.cgu.edu.tw

Department of Computer Science and Information Engineering, Chang Gung University

# Grading and Resources

- Midterm: 30%
- Discussion and Quiz: 20%
- Exercises: 20%
- Final Project Presentation and Report: 30%
- Course Website: http://www.csie.cgu.edu.tw/~chewei/teaching.html
- Moved to https://icechewei.github.io/webpage/teaching.html



# Paper Reading and Topic Survey

- You can cover all contents of the paper or survey the topic by yourself
- 10 minutes for each presentation
- 5 extra minutes for Q&A
- Number of slides is from 10 to 20



# Reading List

| Paper | Name | Student ID |
|---|---|---|
| 1 My VM is Lighter (and Safer) than your Container | M0629013 | 盧昱宏 |
| 2 NetCache: Balancing Key-Value Stores with Fast In-Network Caching | M0629010 | 呂毓軒 |
| 3 Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms | M0629012 | 陳志榮 |
| 4 Corey: An Operating System for Many Cores | M0529004 | 張孝荃 |
| 5 Scaling a file system to many cores using an operation log | M0529016 | 許鏡照 |
| 6 Dynamo: Amazon's Highly Available Key-value Store | M0529011 | SABIH AHMAD KHAN |
| 7 SVE: Distributed Video Processing at Facebook Scale | M0629009 | 沈凱明 |
| 8 NEVE: Nested Virtualization Extensions for ARM | D9921002 | 詹程凱 |
| 9 Multiprogramming a 64 kB Computer Safely and Efficiently | M0629014 | 王宥憲 |
| 10 NOVA-Fortis: A Fault-Tolerant Non-Volatile Main Memory File System | M0629018 | 章齊信 |




# Case Study:

## Real-Time Partitioned Scheduling on Multi-Core Systems with Local and Global Memories


# Real-Time Partitioned Scheduling on Multi-Core Systems with Local and Global Memories

Che-Wei Chang, Jian-Jia Chen, Tei-Wei Kuo and Heiko Falk

*18th Asia and South Pacific Design Automation Conference, Jan. 22-25, Yokohama, Japan*

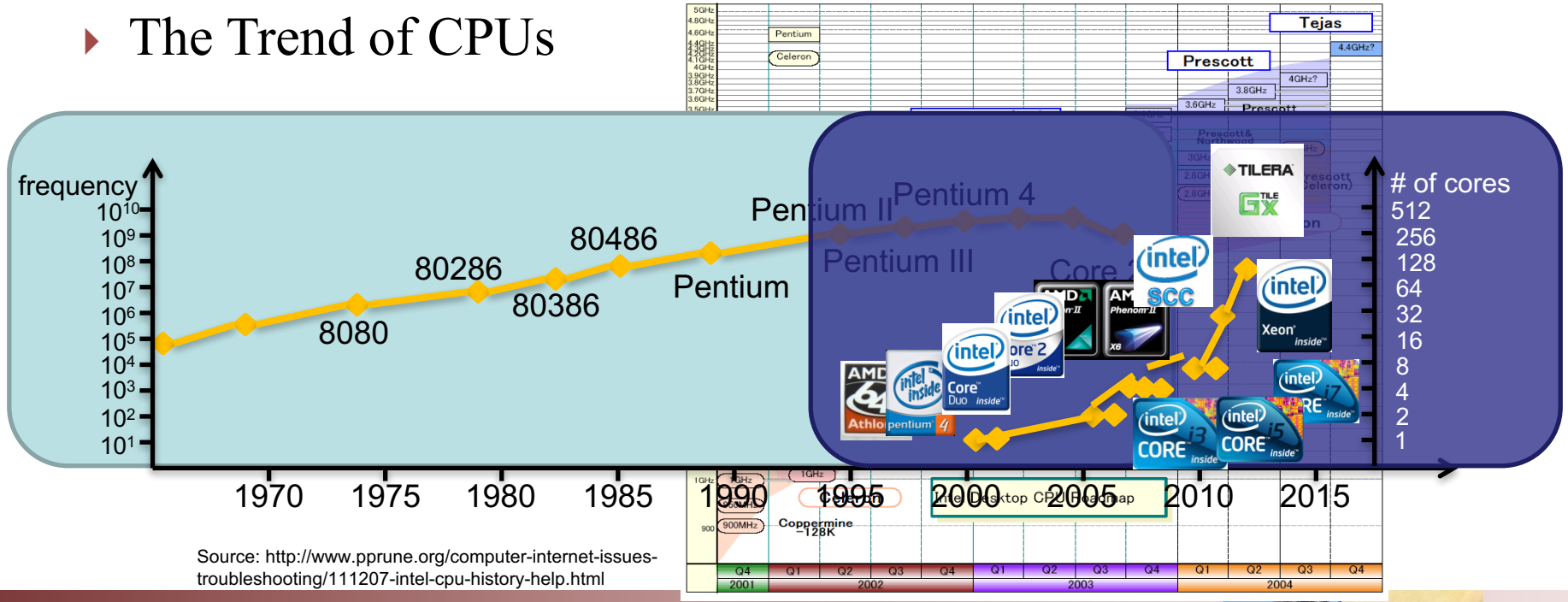
# Agenda

- Introduction and Motivation
- System Architecture of Multiprocessor Platforms with Local and Global Memories
- A Scheduling Algorithm with Considerations of Real-Time and Memory-Space Constraints
- Performance Evaluation with Benchmark Suites
- Conclusion



# Introduction: Multi-Core Systems

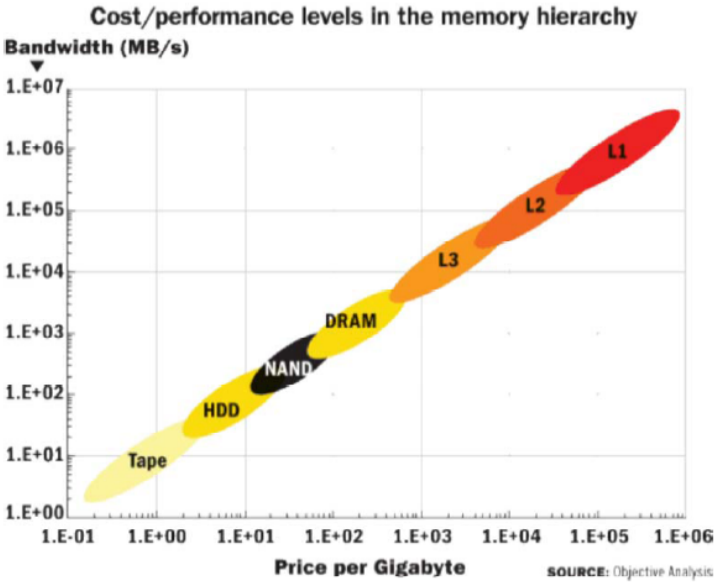
- Emerging Problems of Computing System Designs
  - Limited battery power for embedded systems
  - Thermal problems of integrated chips
  - Frequency constraints of high-performance processors
- The Trend of CPUs

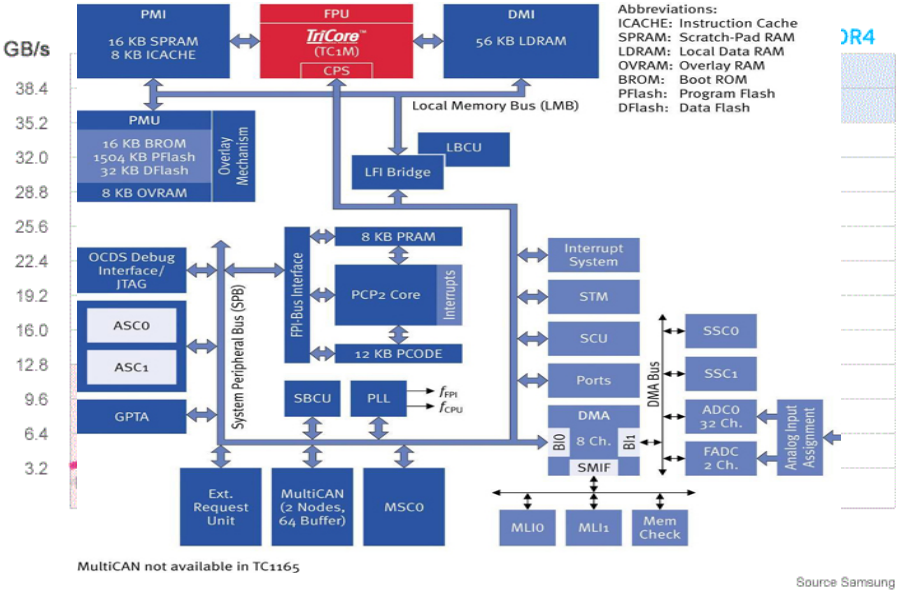



Source: http://www.pprune.org/computer-internet-issues-troubleshooting/111207-intel-cpu-history-help.html

# Motivation: Memory Wall

- The Increasing CPU Performance and Number of Cores
- The Popularity of Heterogeneous Memory Designs
- The Conservative Strategies of Traditional Real-Time Scheduling Algorithm Designs



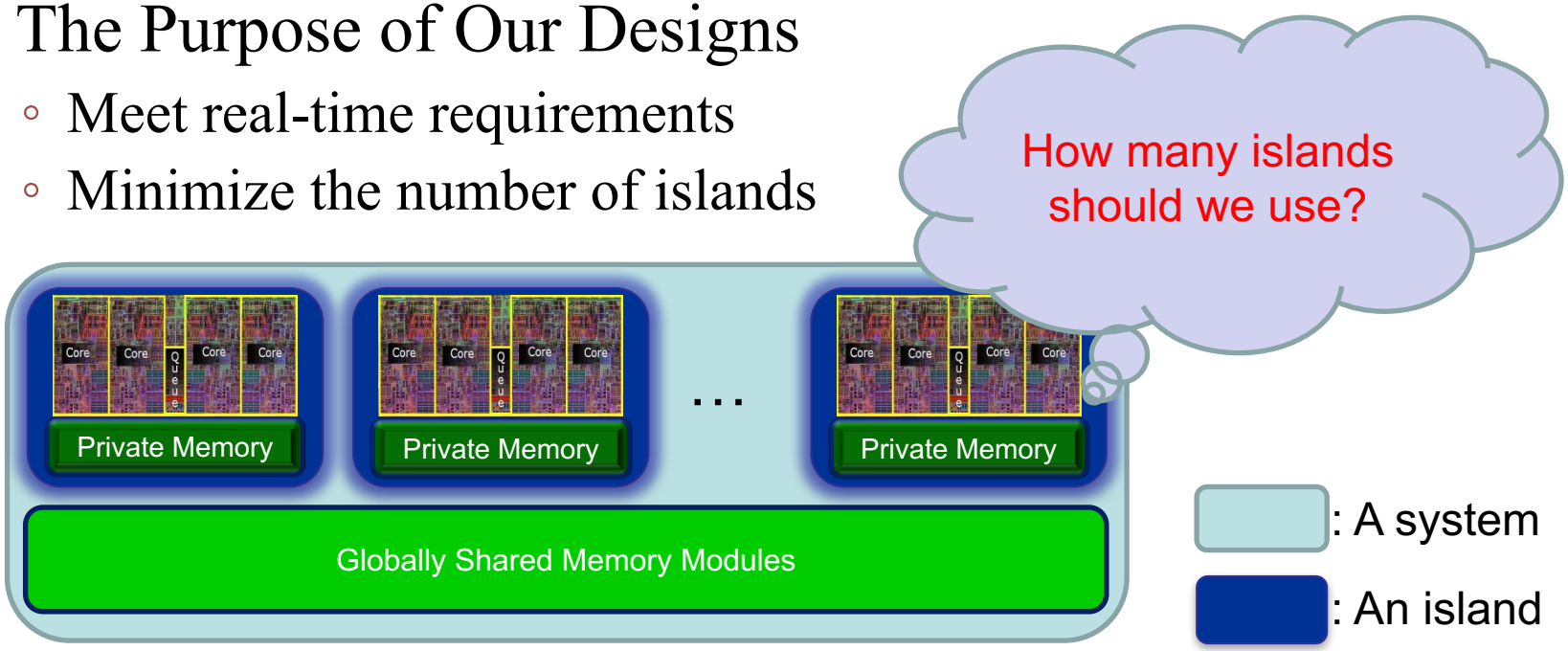




- Source :http://members.tripod.com/pc_tutor1/memory.htm , http://www.infostor.com/index/articles/display/3844378763/articles/infostor/volume-13/Issue_10/Features/Where_do_SSDs_fit__Servers_or_arrays_.html , http://www.samsung.com/global/business/semiconductor/product/computing-dram/overview



# System Model

- An Island-Based Multi-Core Platform
  - A global memory pool and multiple islands
  - A policy to turn on/off islands to manage resources
- The Purpose of Our Designs
  - Meet real-time requirements
  - Minimize the number of islands





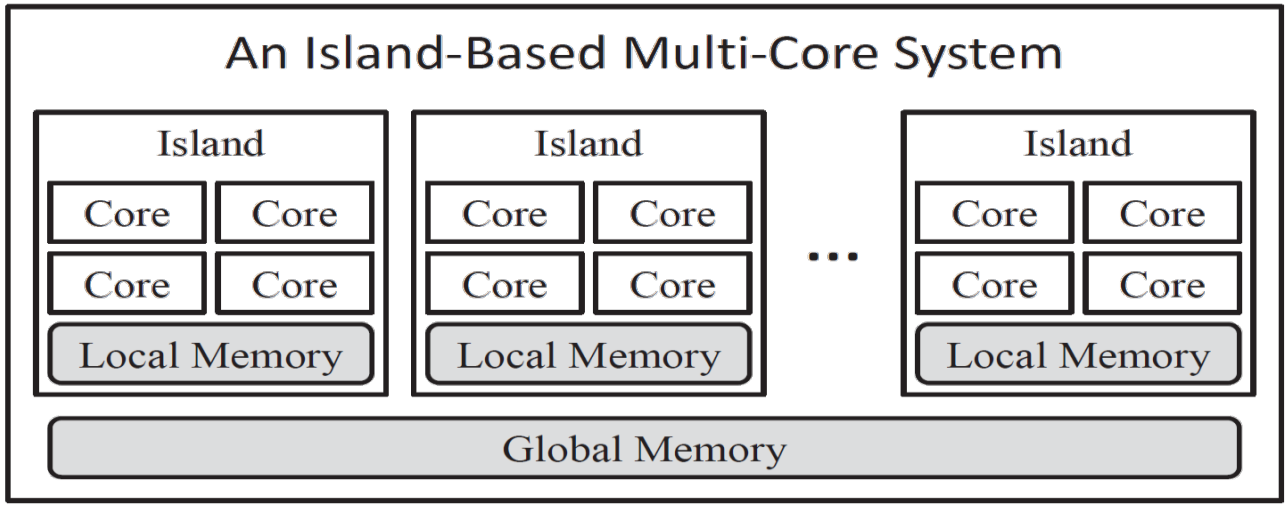

# Problem Definition– Real-Time Tasks

- Goal:
  - Minimize the number of required islands
- Constraints:
  - All implicit-deadline sporadic tasks meet their deadlines
  - No memory space limitation is violated
- Task Model– Implicit-Deadline Sporadic Tasks:
  - A relative deadline
  - The minimum inter-arrival time
  - The worst case execution time



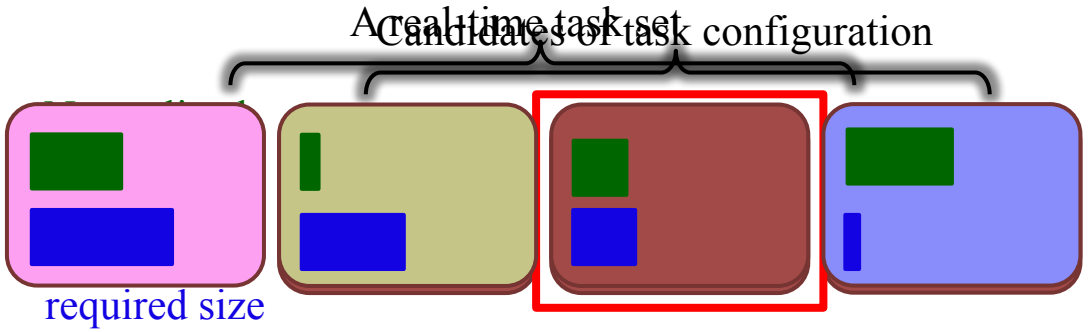
# Problem Definition– Hardware Platforms

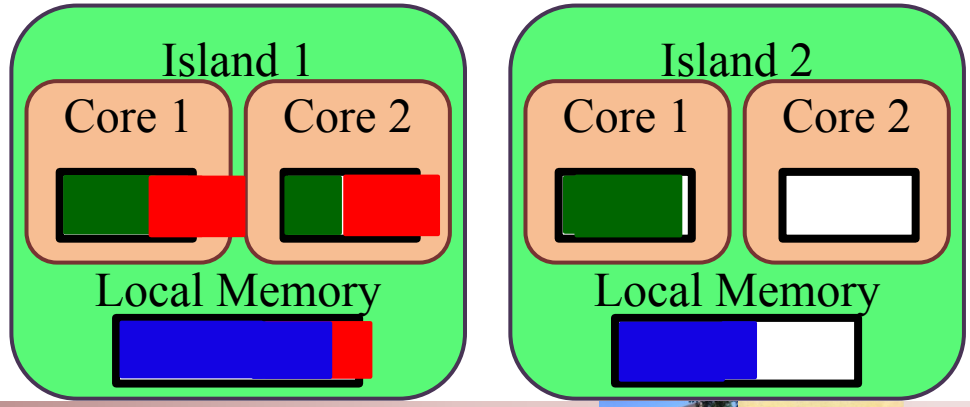
- Hardware Model– Island-Based Multi-Core Platforms:
  - Cores in an island share a local memory pool
  - An island consists of multiple homogeneous cores
  - A system consists of multiple islands
  - A system consists of a large global memory pool



# Proposed Algorithms

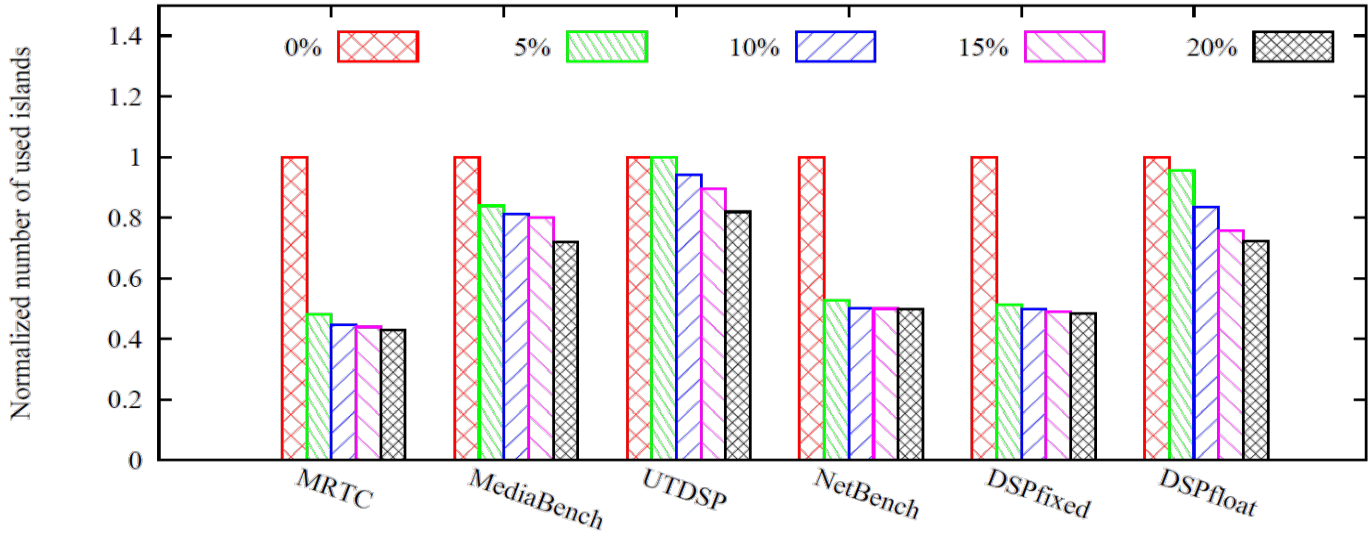
1. Minimize the value of $\frac{U_i^j}{M} + \frac{j}{B_L}$ for each task
2. Sort all tasks in a non-increasing order by the required number of the local memory blocks
3. Partition all tasks to islands by the first fit strategy
4. Assign tasks onto cores by the first fit strategy
5. Allocate a new island if there is no feasible assignment for a task



# Performance Evaluation

- 82 real-life benchmarks from MRTC, MediaBench, UPDSP, NetBench, and DSPstone
- The worst-case execution time of each task is generated by aiT which is based on Infineon TriCore TC1797
- The number of cores in an island is 4



# Conclusion

- Summary
  - Minimize the needed number of islands
  - Meet the real-time required
  - Provide a polynomial-time asymptotic 4-approximation algorithm
- Insights
  - The design of the memory architecture of multi-core systems is with increasing importance
  - CPU is not the only resource which should be considered in multi-core real-time scheduling




# Summarizing this Semester

# Common Performance Metrics

- Unit Cost: the monetary cost of manufacturing each copy of the system
- NRE Cost (Non-Recurring Engineering cost): the one-time monetary cost of designing the system
- Size: the physical space required by the system
- Performance: the execution time or throughput of the system
- Power: the amount of power consumed by the system
- Flexibility: the ability to change the functionality of the system without incurring heavy NRE cost
- Time-to-Market: the time required to develop a system to the point that it can be released and sold to customers
- Maintainability: the ability to modify the system after its initial release



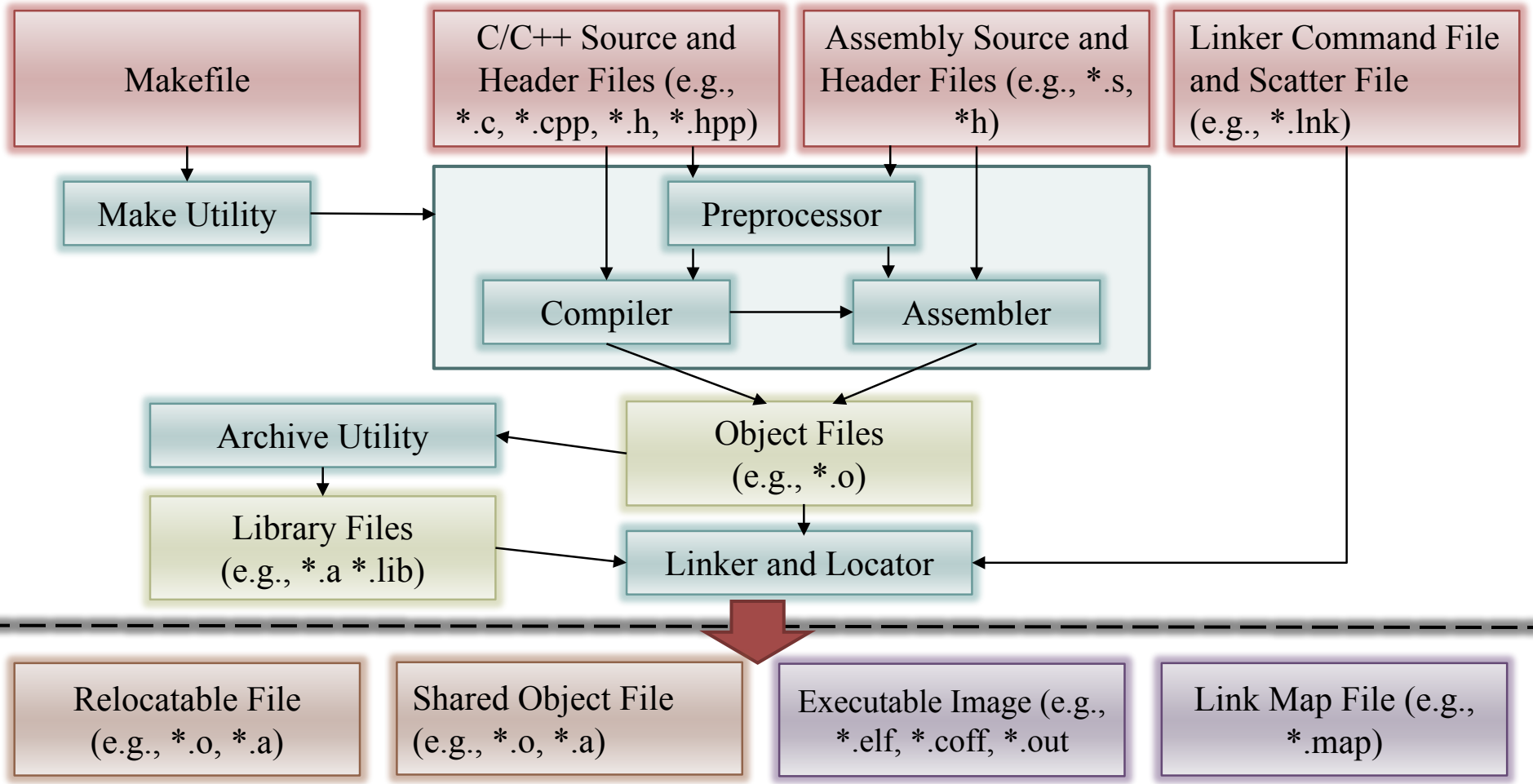
# Definition of Embedded Systems

- It is difficult to define embedded systems
  - An embedded system is a digital system
  - An embedded system has computing processors
  - An embedded system runs dedicated functions
  - An embedded system is frequently used as a controller

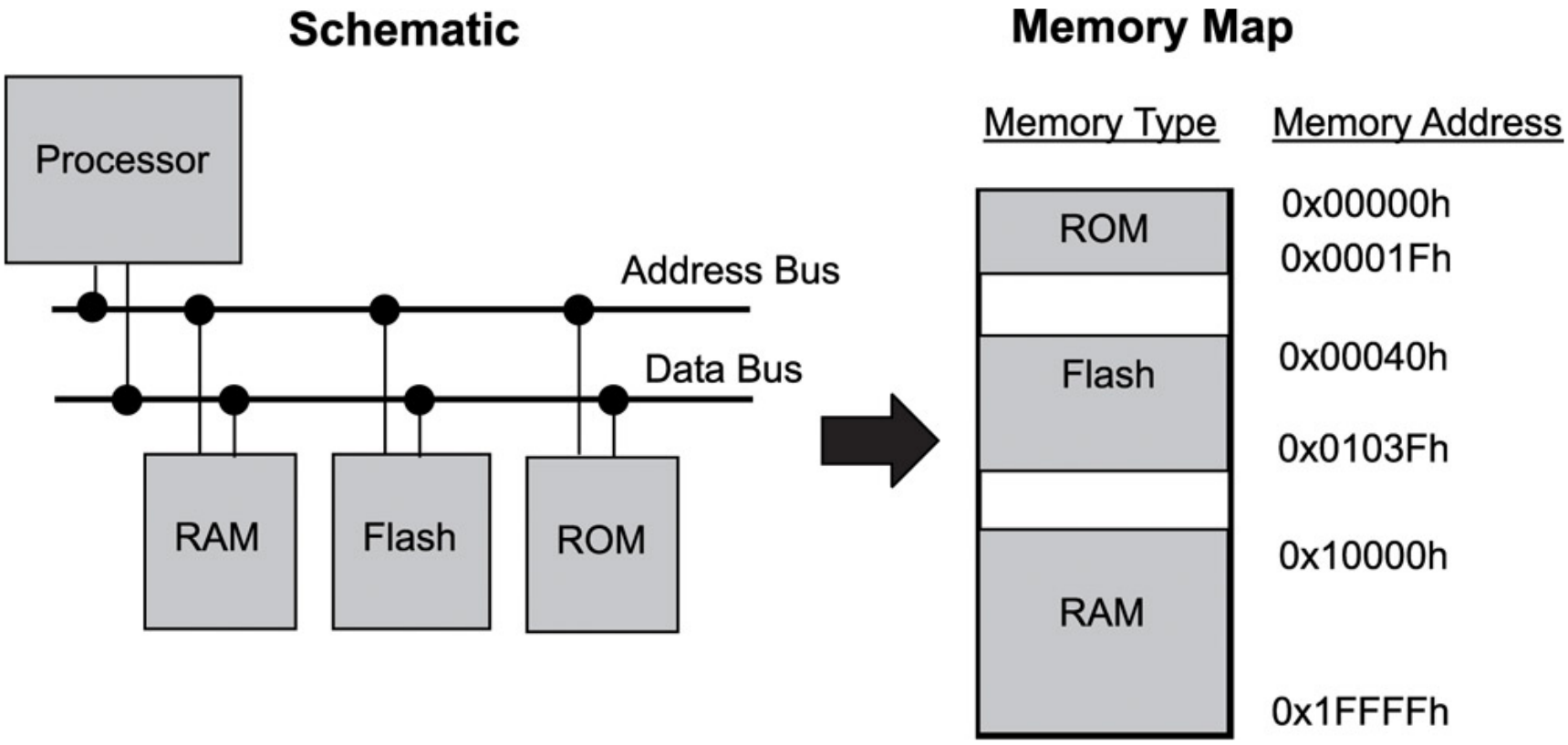
- An embedded system is a computer system with some dedicated functions within a larger mechanical or electrical system, often with real-time computing constraints. - From Wikipedia



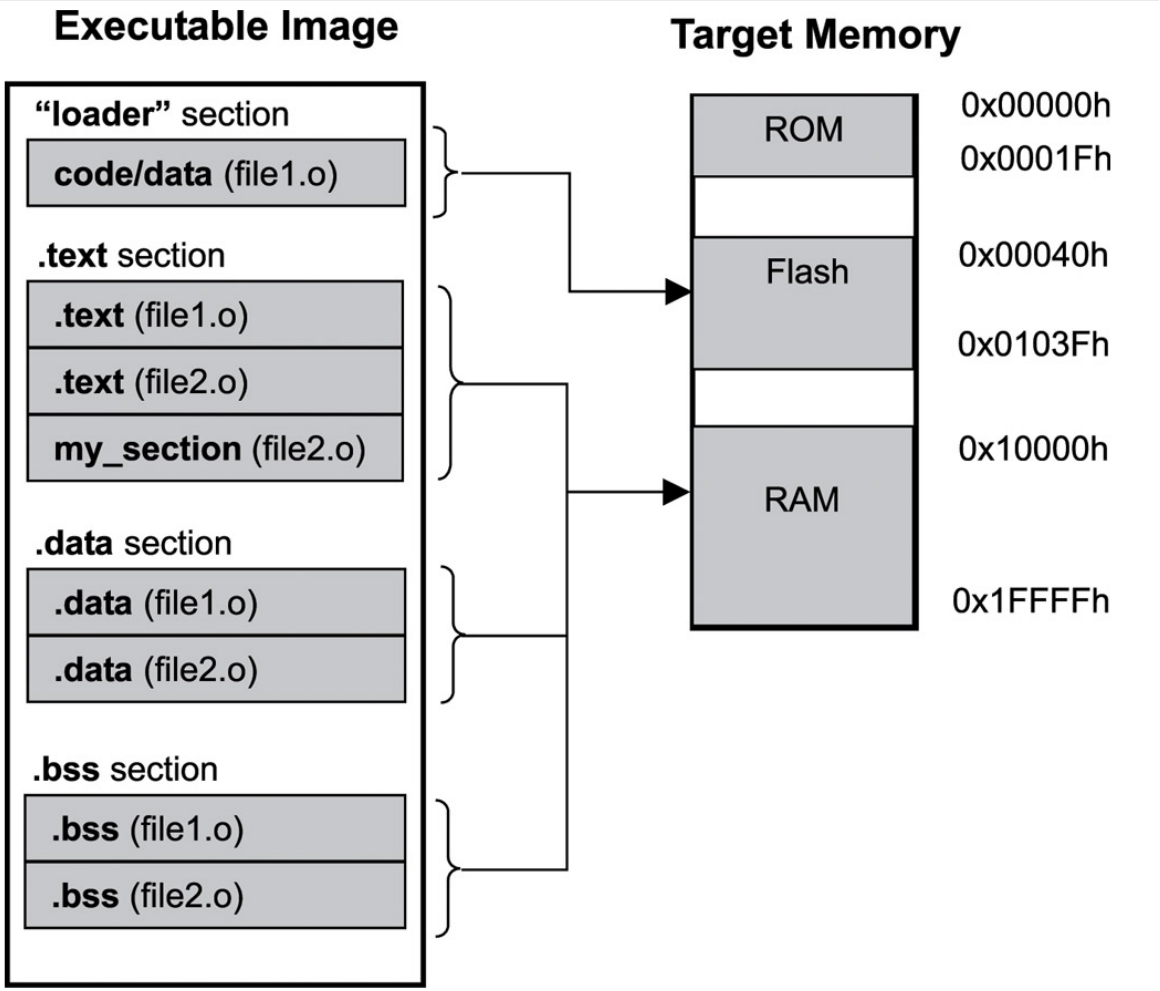
# Creating an Image for a Target Embedded System



# Simplified Schematic and Memory Map for a Target System

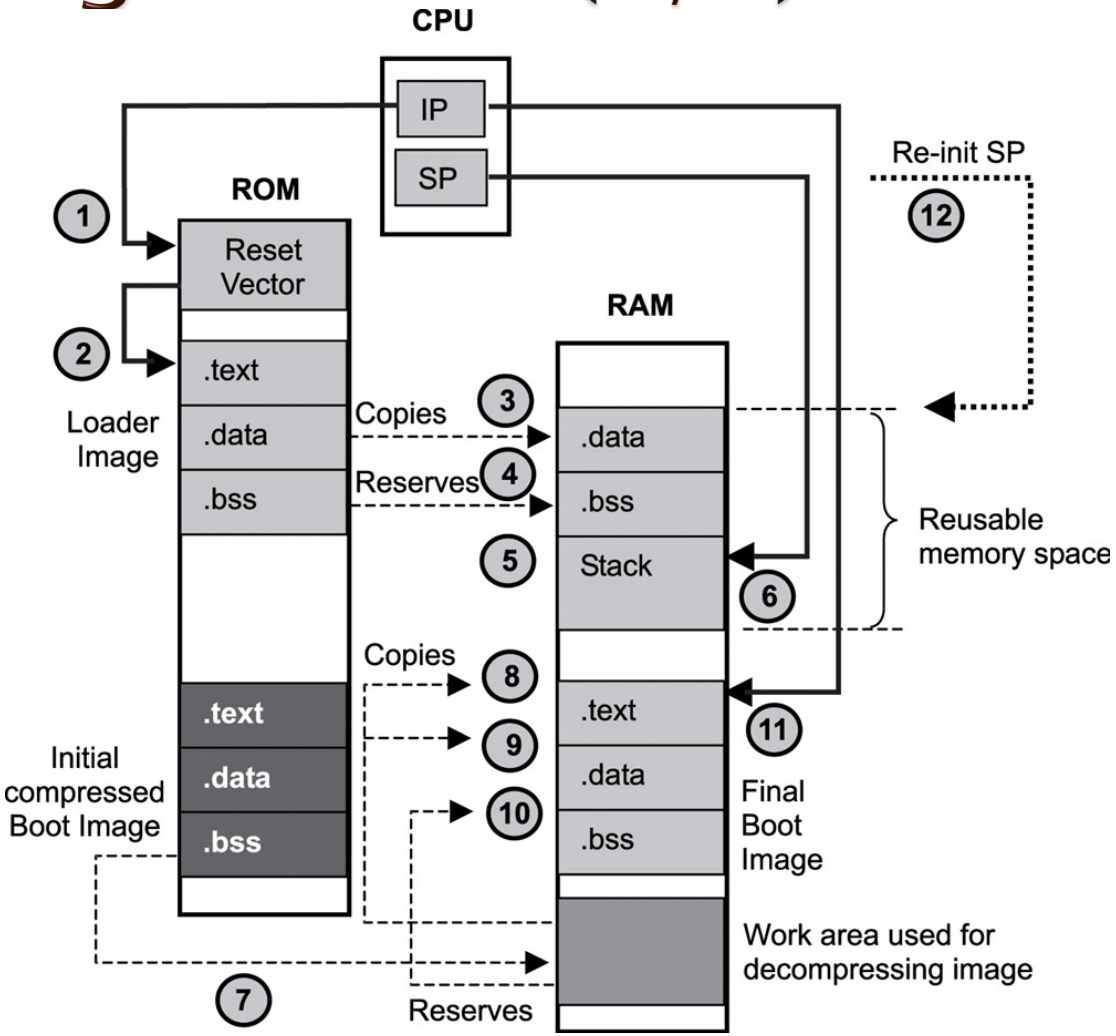


# Mapping an Executable Image into the Target System






# Image Transferred from ROM Running on RAM (1/2)



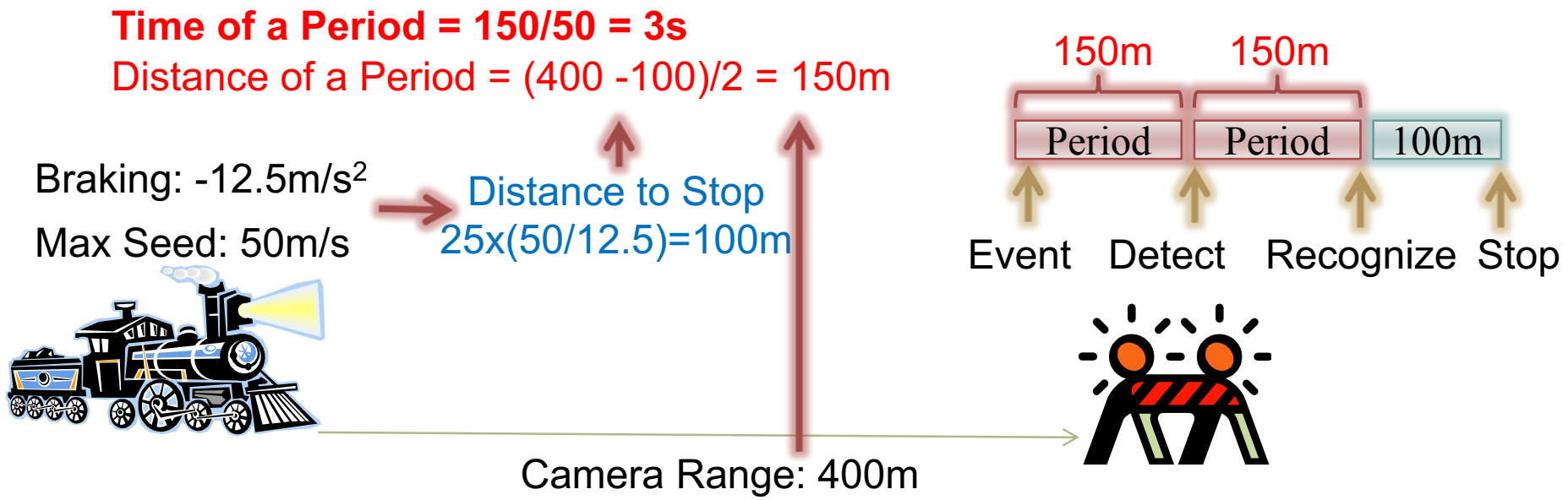
# Image Transferred from ROM Running on RAM (2/2)

- The boot loader transfers an application image from ROM to RAM for execution
  - The application image is usually compressed in ROM to reduce the storage space required
- Boot sequence
  7. Copy the Compressed application image from ROM to RAM in a work area
  8. Decompress and initialize the application image(1)
  9. Decompress and initialize the application image(2)
  10. Decompress and initialize the application image(3)
  11. The loader transfers control to the image using a processor-specific jump instruction
  12. Recycle the memory area occupied by the loader and the work area
      - May also reinitialize the SP to point to the memory area occupied by the loader to use it as the stack space



# An Example of Real-Time Designs

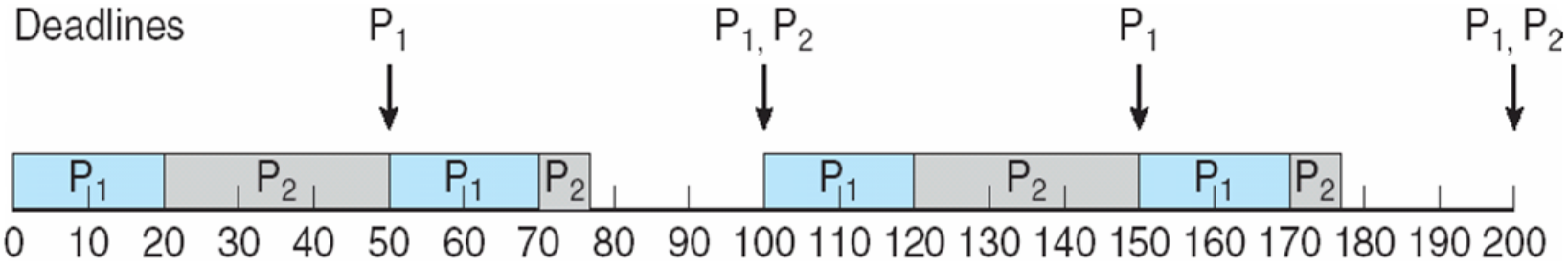
- A camera periodically takes a photo
- The image recognition result will be produced before the next period
- If there is an obstacle, the train automatically brakes



# A Static Scheduling Algorithm—Rate Monotonic Scheduling

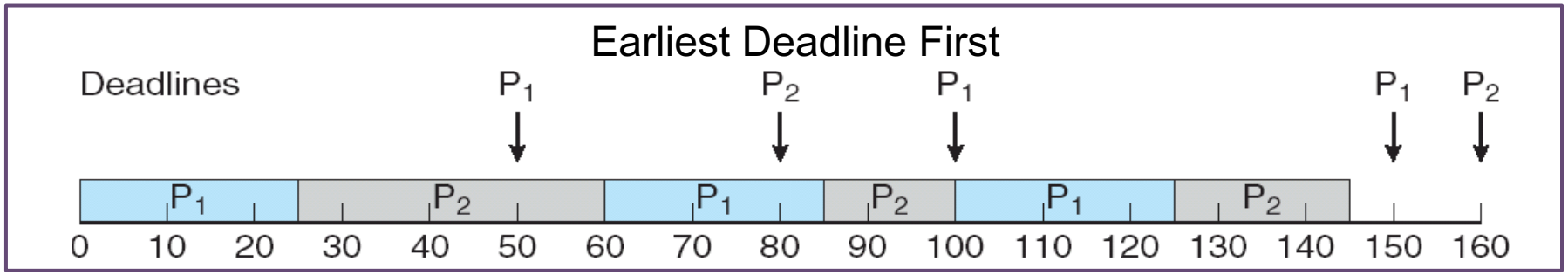
- A static priority is assigned to each task based on the inverse of its period
  - A task with shorter period ➔ higher priority
  - A task with longer period ➔ lower priority
  - For example:
    - $P_1$ has its period 50 and execution time 20
    - $P_2$ has its period 100 and execution time 37

    ➔$P_1$ is assigned a higher priority than $P_2$



# A Dynamic Scheduling Algorithm—Earliest Deadline First Scheduling

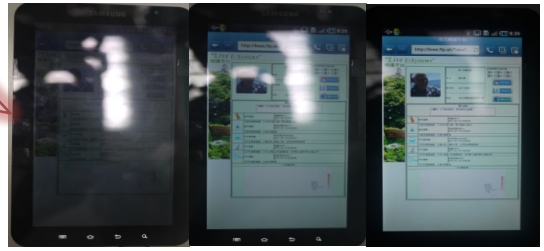
- Dynamic priorities are assigned according to deadlines
  - The earlier the deadline, the higher the priority
  - The later the deadline, the lower the priority
  - For example:
    - $P_1$ has its period 50 and execution time 25
    - $P_2$ has its period 80 and execution time 35



# Two Approaches to Reduce the Power Consumption of Devices

- Dynamic Voltage and Frequency Scaling (DVFS)
  - Scale down the voltage and/or frequency to reduce the processor power consumption
  - An example: reducing 17% of the energy consumption for H.264 decoding over TI DaVince DM6446
- Dynamic Power Management (DPM)
  - Change to an energy-efficient state to reduce the power consumption of peripheral devices
  - An example: reducing 15% of the energy consumption for the browser over Android Galaxy Tab

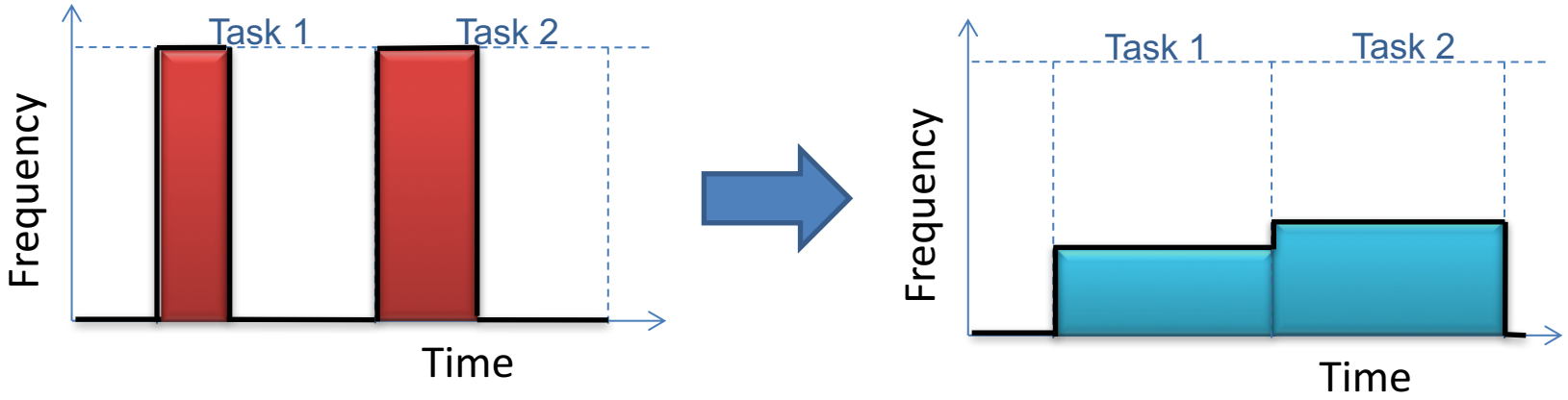
| Power Consumption | 1625mW | 2023mW | 3166mW |
|---|---|---|---|



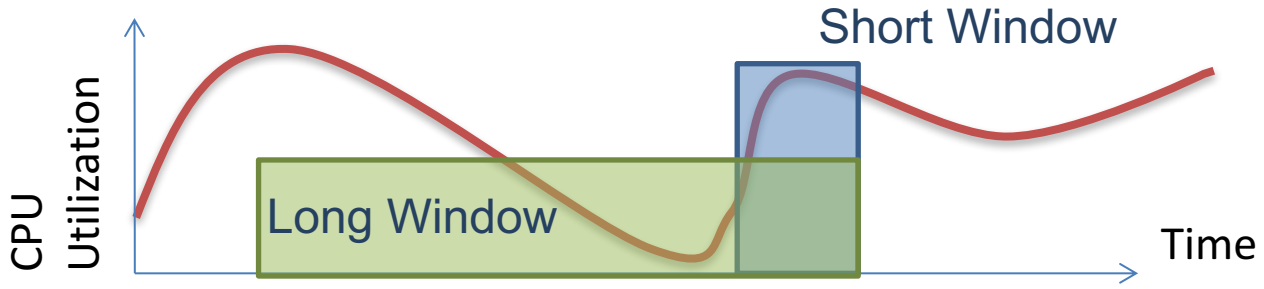



# Dynamic Frequency and Voltage Scaling Designs

- The Observation: It is energy-efficient to scale down the processor frequency dynamically to fit current workloads




- The Approach: It keeps a window or buffer to estimate the workload



# Introduction of µC/OS-II (1/2)

- The name is from micro-controller operating system, version 2
- µC/OS-II is certified in an avionics product by FAA in July 2000 and is also used in the Mars Curiosity Rover
- It is a very small real-time kernel
  - Memory footprint is about 20KB for a fully functional kernel
  - Source code is about 5,500 lines, mostly in ANSI C
  - It's source is open but not free for commercial usages
- Preemptible priority-driven real-time scheduling
  - 64 priority levels (max 64 tasks)
  - 8 reserved for µC/OS-II
  - Each task is an infinite loop




# Introduction of µC/OS-II (2/2)

- Deterministic execution times for most µC/OS-II functions and services
- Nested interrupts could go up to 256 levels
- Supports of various 8-bit to 64-bit platforms: x86, 68x, MIPS, 8051, etc.
- Easy for development: Borland C++ compiler and DOS (optional)
- However, uC/OS-II still lacks of the following features:
  - Resource synchronization protocol
  - Soft-real-time support


# The μC/OS-II File Structure

Application Code (Your Code!)

**Processor Independent Implementations**

- Scheduling policy
- Event flags
- Semaphores
- Mailboxes
- Event queues
- Task management
- Time management
- Memory management

**Application Specific Configurations**

- OS_CFG.H
- Max # of tasks
- Max Queue length
- …

uC/OS-II Port for Processor Specific Codes

***Software***

---

***Hardware***

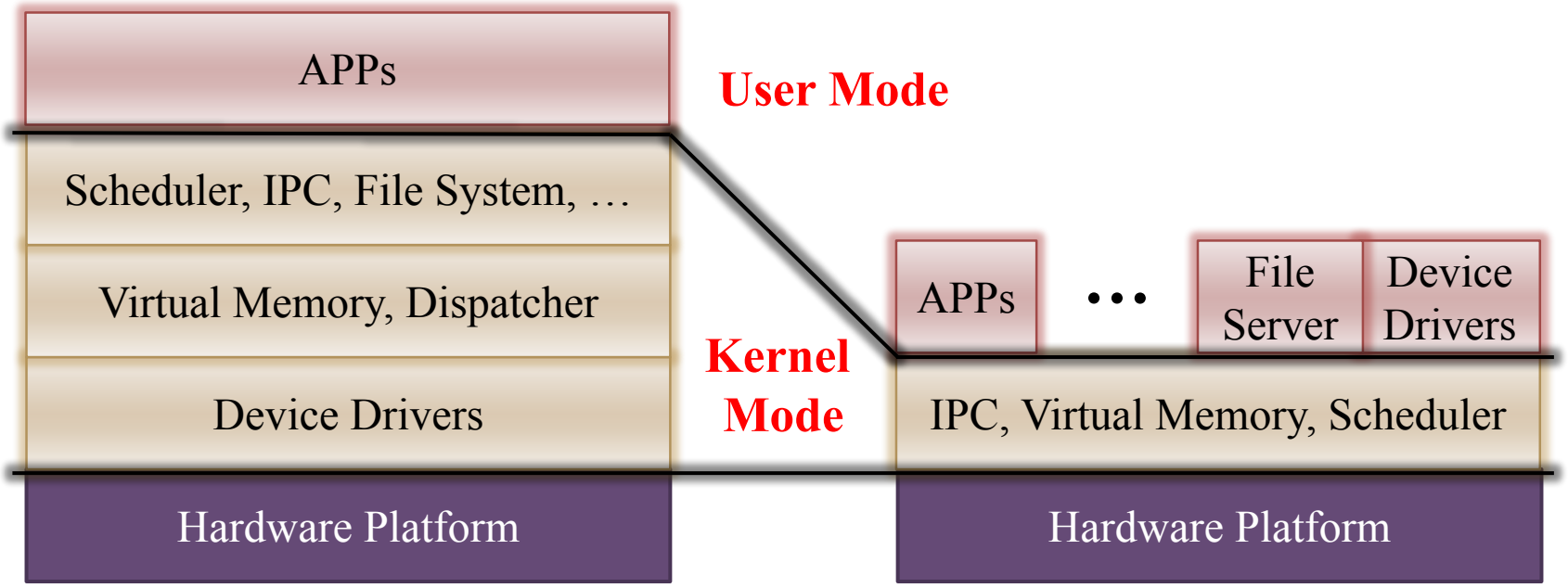
CPU | Timer



# Monolithic Kernel and Microkernel

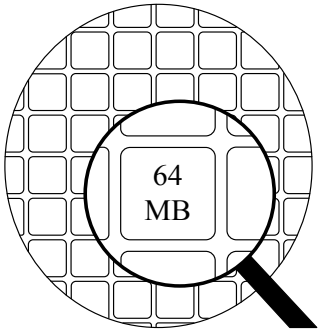
Monolithic Kernel

Microkernel



# Single-Level Cell (SLC) vs Multi-Level Cell (MLC) Flash

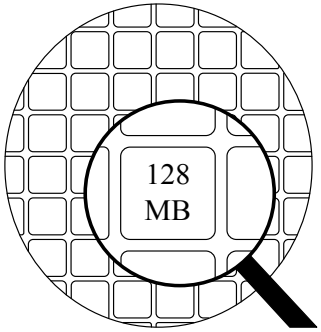
SLC Flash

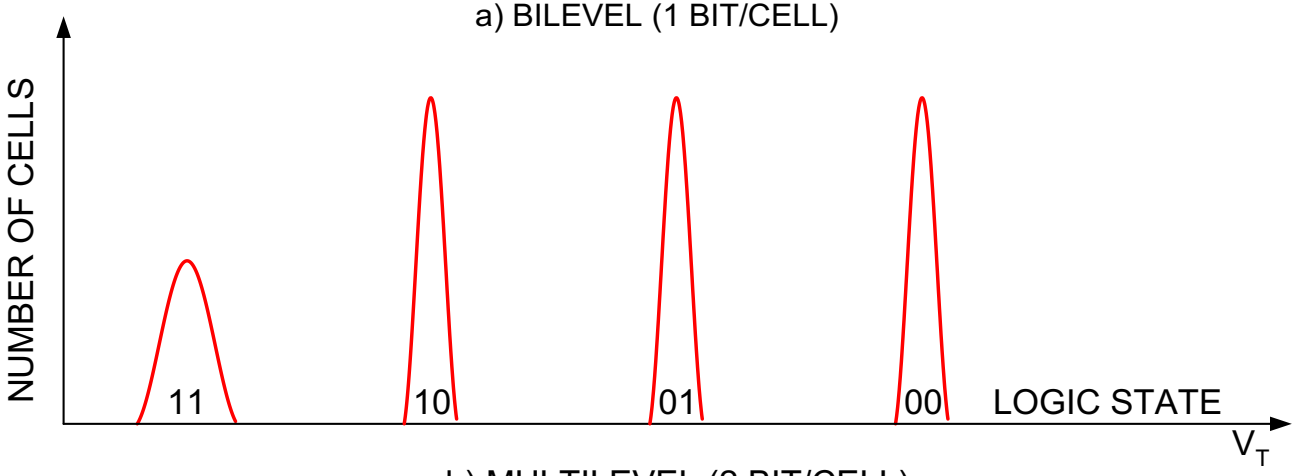




a) BILEVEL (1 BIT/CELL)

MLC Flash

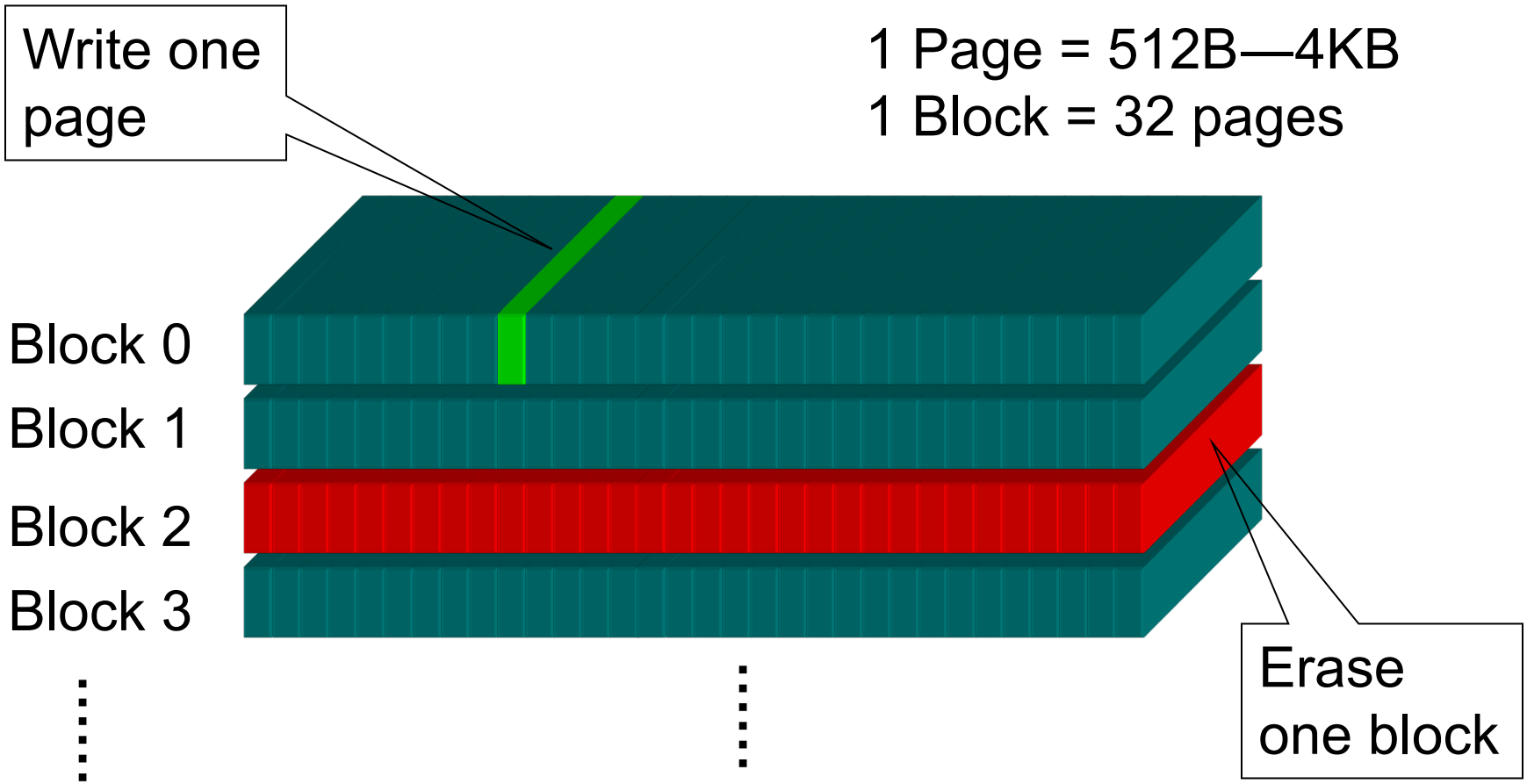



b) MULTILEVEL (2 BIT/CELL)



# Page Write and Block Erase



# Lab 1: Android Tools

- Goal: run android applications on Qualcomm evaluation boards
- Install JDK
- Install Android Studio
- Create a project and use API level 19 (for kitkat)
- Run the program on an Android emulator
- Export the apk file and upload it to somewhere
- Download it to the evaluation board and run it



# Lab 2: Embedded System Tools and Multi-thread Programs

- Build the Cross Development Toolchain
- Build the Linux Kernel
- Setup a TFTP Server
- Setup NFS Server
- Setup the Target Board
- Download the Linux Kernel
- Write Multi-thread programs on TI OMAP Evaluation Board
- Use Mutex to protect your program




# Thank You!