Últimas Tendencias en Sistemas Empotrados

How to Identify an ES

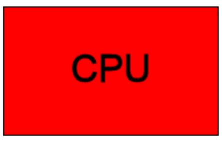
- Embedding of an information processing system into a somewhat greater product
- Typical applications:
  - Automotive
  - Aeronautic
  - Telecommunication
  - Sensor networks
  - etc.

Applications

- Monitoring/ Control of manufacturing process
- Production management
- Control of power plants
- Control of railway systems
- Aeronautic control systems
- Observation and logging of environmental data
- Space flight
- Military systems
- Telecom systems
- Robotics
- Team Robotics and autonomic vehicles
- Virtual and augmented reality

Point of View of an Computer Architect

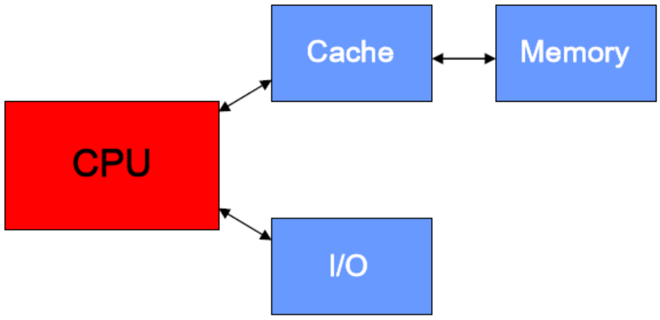
Figure of merit: Performance



Source: J. Kaiser

Point of View of an Computer Architect with ideas ahead

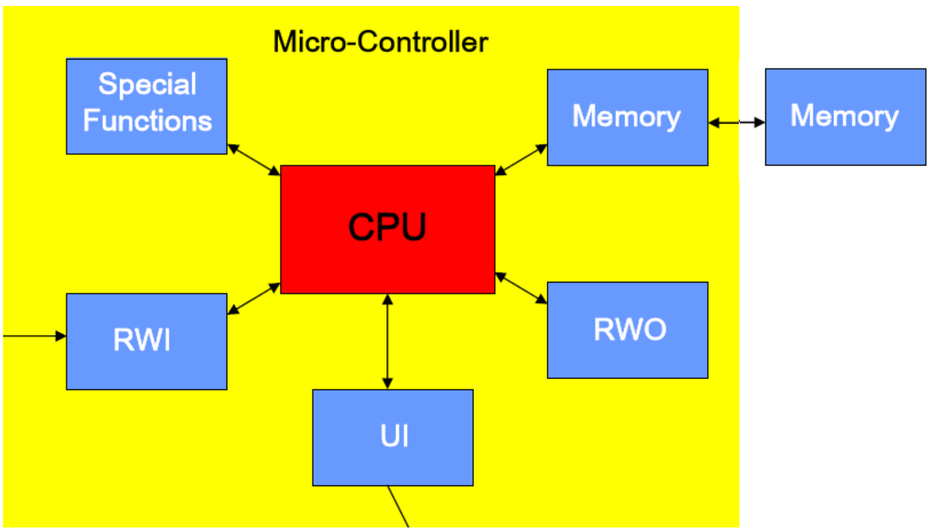
Figure of merit: Performance and costs



Source: J. Kaiser

Point of View of an ES Architect

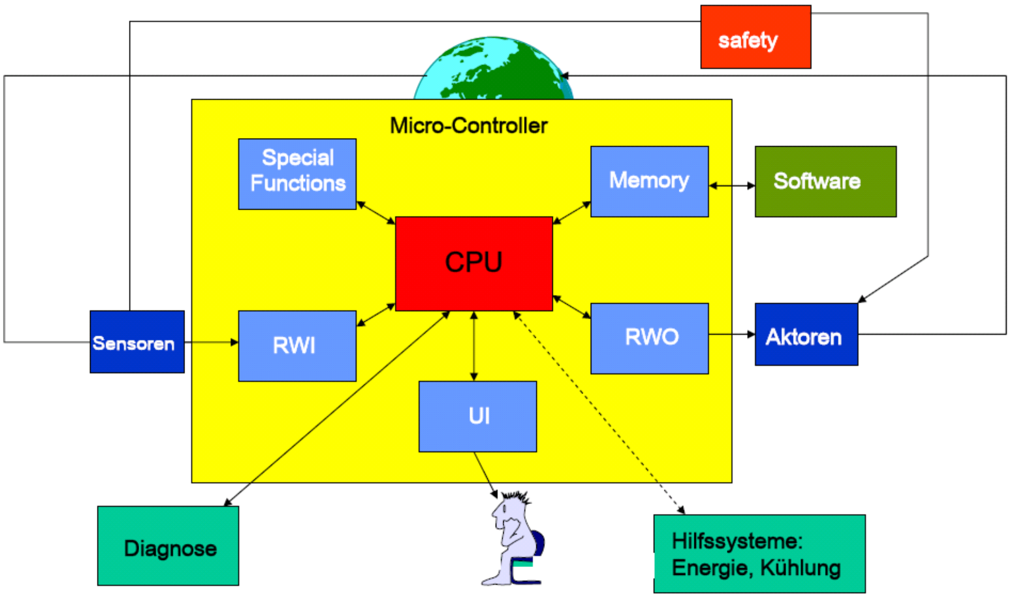
Figure of merit:
costs, I/O-options, Memory size, Performance



Source: J. Kaiser

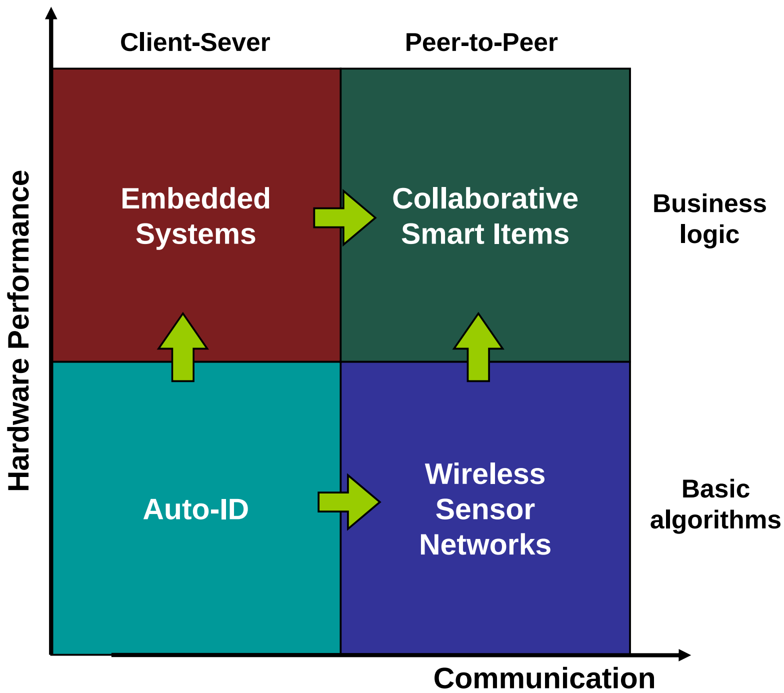
Point of View of an ES Architect for Control Applications

Figure of merit:
costs, Time-to-Market, features, costs, costs…

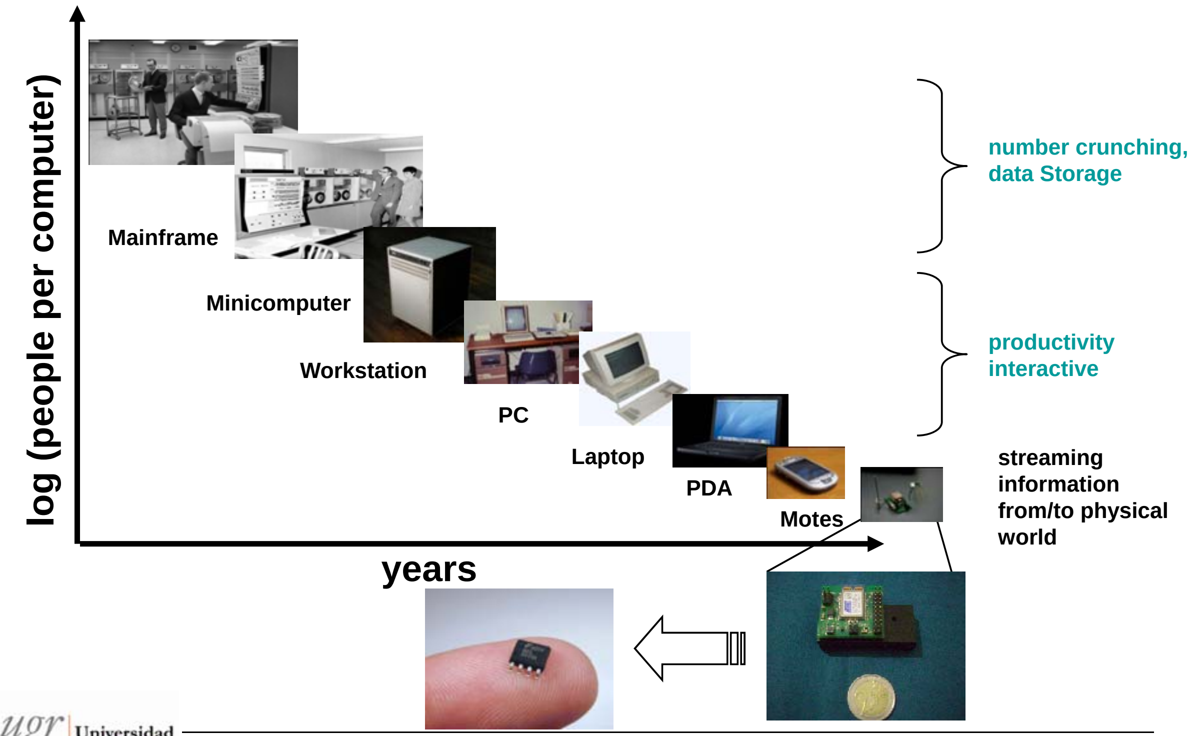


Source: J. Kaiser

Trends: Collaborative Smart Items

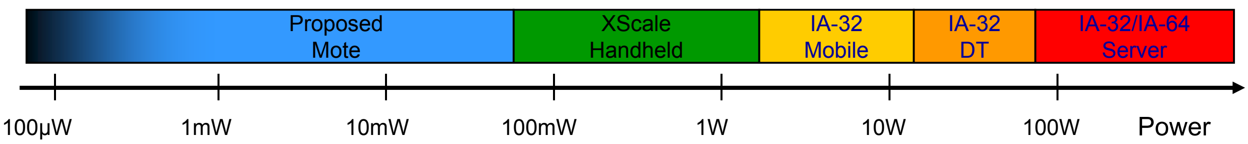


Trends: A new class of computing (Moore's Law)



Motes overview

- A "mote" is a tiny wireless computing platform
  - CPU, memory, FLASH, I/O, radio components
  - Low power operation, often battery operated

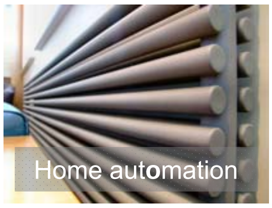


- Motes are used to build wireless mesh networks
  - Self configuring and maintaining connectivity
  - Distributed sensing of environmental data
  - Distributed computation capabilities
  - Bandwidth and resources scale with network size

Source: Intel

Sensor network applications

- Environmental monitoring
  - Habitat monitoring
  - Precision agriculture
  - Heating Ventilation Air-Conditioning (HVAC) systems
  - Security, surveillance
- Structure and equipment monitoring
  - Structural dynamics
  - Condition-based maintenance
  - Emergency response
- Supply chain monitoring
  - Manufacturing flows, asset tracking
- Context aware computing
  - Information beacons



How Many Embedded Systems?

- In the average American household: around 40 microprocessors; not counting:
  - PCs, which contribute another 5-10 each
  - cars, which typically contain a few dozen
- Will rise 100X over next couple of decades
- Most people don’t know what “embedded” means

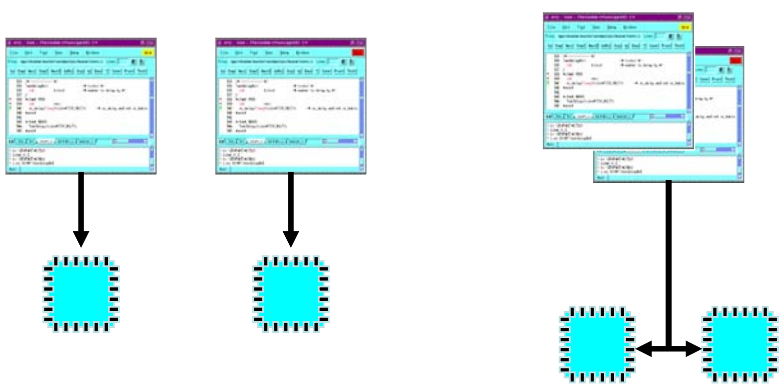
Source: Collin Walls

Development Challenges

- Multiple processors:
  - A digital camera typically has two: one deals with image processing and the other looks after the general operation of the camera.
  - Debugging of multiple processors is one of the biggest challenge
- Limited memory:
  - Embedded systems almost always have limited memory
  - Although the amount of memory may not be small, it typically cannot be added on demand
- User interface:
  - The user interface on any device is critically important
  - Its quality can have a very direct influence on the success of a product

Multiple Processors

- Key challenge is debugging
- Need multi-core support



Limited Memory

- May not be small, but probably not extendable
- Cost and power consumption issues
- Understand optimization
- C++ requires skill and the right tools

User Interface

- Critically important
- Mainly implemented in software
- Ideal steps:
  - design the hardware
  - make some prototypes
  - implement the software [UI]
  - try the device with the UI and refine/re-implement as necessary

UI Development

- Hardware not available
- Design may not even be complete
- Need to use prototyping/simulation technology to model on host computer

Re-usable Software

- Used to be a “start from scratch” approach
- Now software is too big and too complex
- Nobody can have all the expertise
- Time to market pressure drives short development cycles
- Reuse widely accepted in hardware design – same needed in software

Software Components

Examples:

- Real-time operating system
- File system
- USB
- Graphics
- Networking

Real Time Operating Systems

- 200 products on the market
- Still common to implement in-house
- Need to understand selection criteria

RTOS Selection Factors

- Hard Real Time
- Royalty Free
- Support
- Tools
- Ease of Use
- Networking
- Broad CPU Support

RTOS Standards

- Many proprietary
- Some standards available:
  - OSEK/VDX [automotive/transportation]
  - µiTRON [Japan]
  - POSIX [migration from UNIX host]

File System

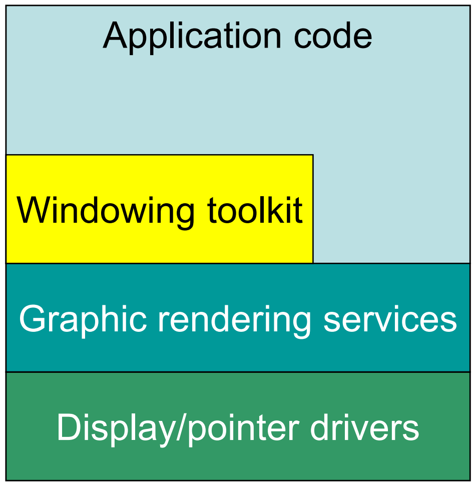
- Persistent storage
- Magnetic, optical or NVRAM [flash]
- Standards-based approach best
  - interoperability issues
  - data transfer
- MS-DOS the easiest standard to adopt

USB

- Implementation is very complex
  - hence ease of use
- Smart part is software not hardware
- Support already done for host computer
- Needed for embedded devices
- USB On-The-Go becoming available

Graphics

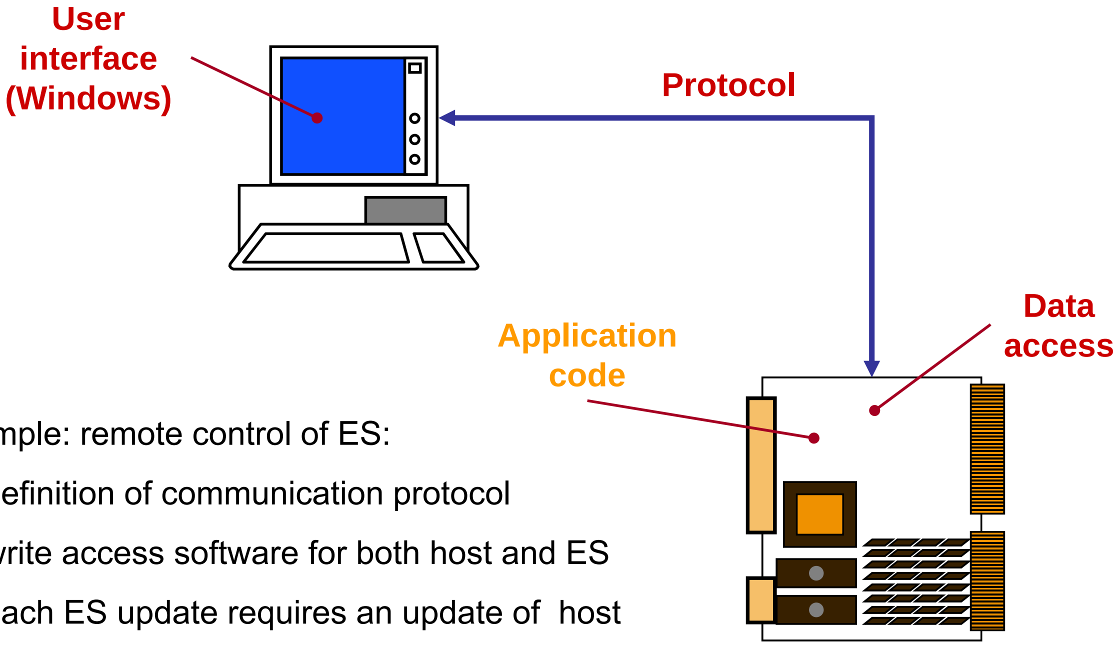
- LCD panel may have 2 functions
  – graphic output
  – user interface
- Doing graphics seems easy, but can quickly become complex
  – simplified with graphics library
- GUI is typically another library on top of regular graphics



Networking

- At least a third of embedded system are connected
- May be wired or wireless
- TCP/IP is quite straightforward to program
  – additional applications and protocols are challenging

Who Needs a Web Server?



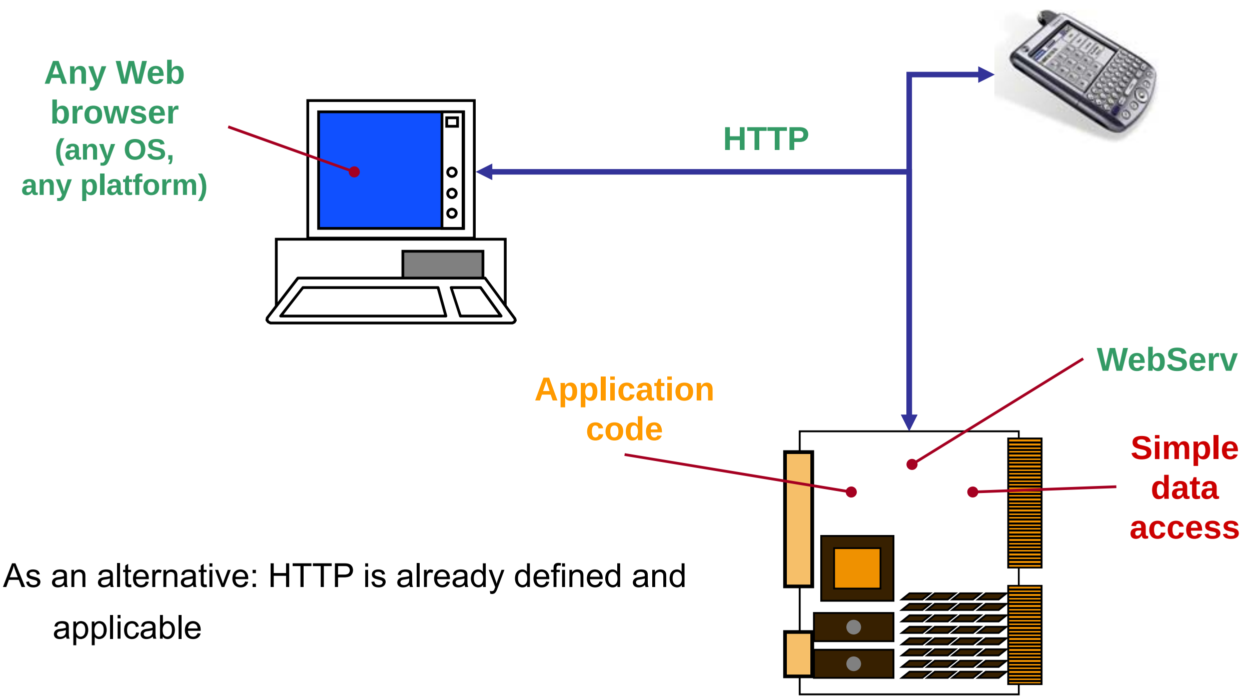
Example: remote control of ES:

→ definition of communication protocol

→ write access software for both host and ES

→ each ES update requires an update of host software

Web Server Solution



As an alternative: HTTP is already defined and applicable

Memory in Embedded Systems

Memory

- Memory growing relentlessly
- PCs started with 16K
  - 640K max
  - current 512M norm
- Address bus size?
  - 32 not enough
  - 64 possible
  - 128 is overkill

What is Memory?

Hardware engineer:

"Memory is a chip in which you can keep bits of data. There are really two kinds: ROM and RAM. These, in turn, come in two varieties each. There is masked programmed ROM and programmable devices, which you can program yourself. RAM may be static, which is easy to use, but has less capacity; dynamic is denser, but needs support circuits."

What is Memory?

Software engineer:

"Memory is where you run your program. The code and data are read off of the disk into memory, and the program executed. You do not need to worry too much about the size, as virtual memory is effectively unlimited."

What is Memory?

Embedded systems programmer:

"Memory comes in two varieties: ROM, where you keep code and constants, and RAM, where you keep the variable data [but which contains garbage on startup]."

What is Memory?

C compiler designer:

"There are lots of kinds of memory: there is some for code, variable data, literals, string constants, initialized statics, uninitialized statics, stack, heap, some is really I/O devices, and so forth.

ROMable Code

- Code will execute correctly from ROM
  - no copy to RAM necessary
  - but RAM may be faster
- Code and data must not be mixed
  - except for constant data
- Compiler/linker should accommodate these requirements

Program Sections

- Names units of memory
- Address assigned at link time
- Each may be contributed to by multiple modules
- Minimal requirement is 2 sections:
  - ROM and RAM

Static Variables

- Any variable not on stack or in a register
- May have an initial value
- Defaults to 0
- Problems with random values in RAM

Static Variables Options

- 3 options:
  – only do explicit assignments to initialize
  – map statics to ROM [makes them into constants]
  – map variables to RAM and values to ROM and copy at start-up

Embedded Tools Solution

- **Typical selection of sections:**
  - **`code`** – the program code
  - **`zerovars`** – uninitialized static
  - **`initvars`** – initialized statics
  - **`const`** – variables declared **const**
  - **`strings`** – constant text strings
  - **`literals`** – compiler generated literals
  - **`tags`** – compiler generated tags

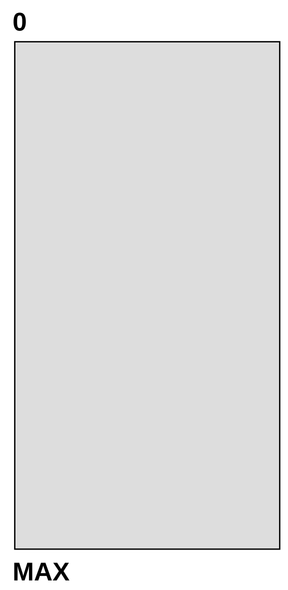
Memory Architectures

Memory Architectures

- Flat single-space
- Segmented
- Bank-switched
- Multiple-space
- Virtual

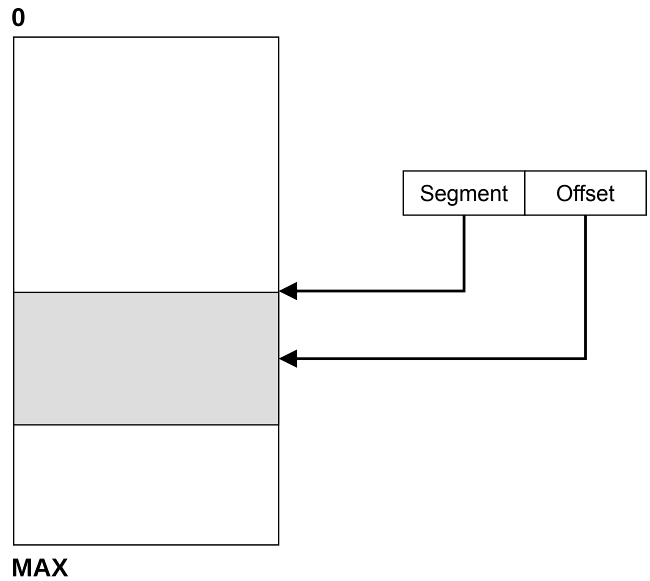
Flat Single-space Memory

- Simple
- Examples: 68K, Z80
- Space may be discontinuous
- Assumed by C
- Care with address 0



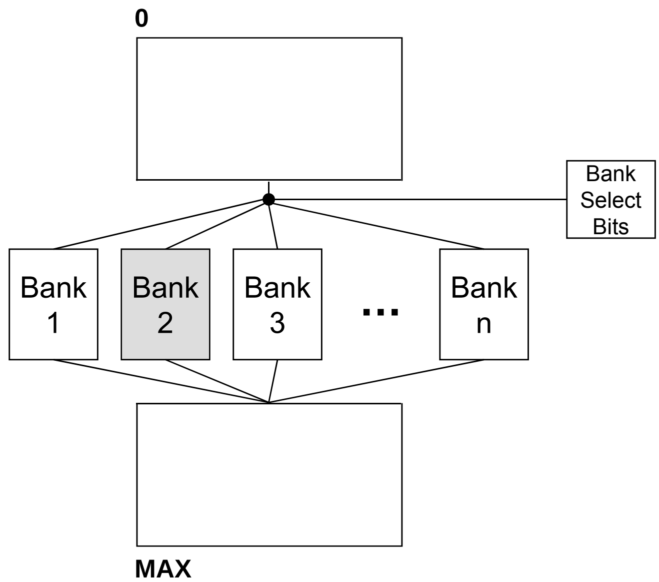
Segmented Memory

- Increased address space
- Example: Intel x86
- 2 part address:
  - segment
  - offset
- Need C extension: `near` and `far`

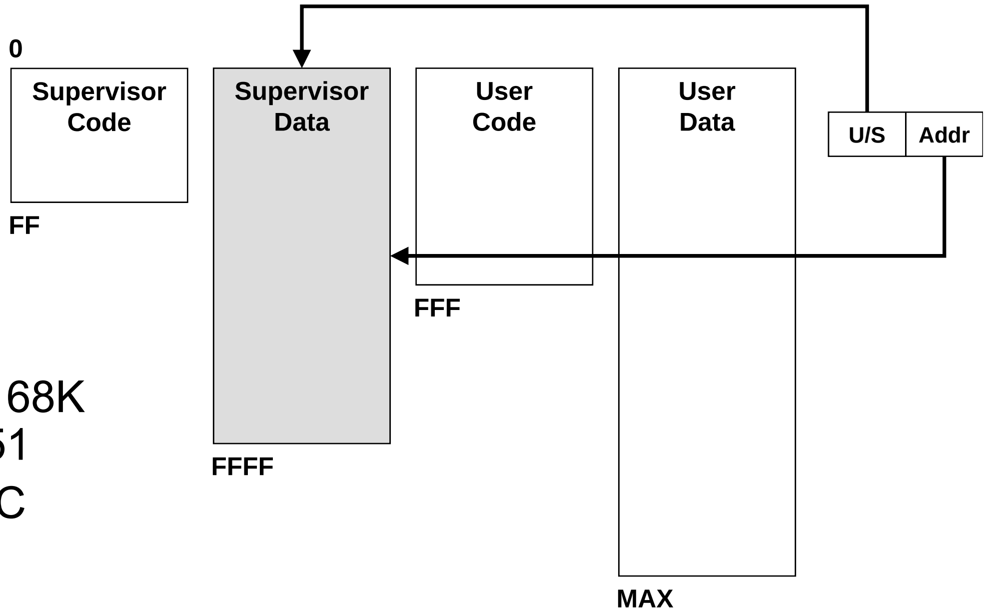


Bank-switched Memory

- Can be added to any processor
- Window into larger address space
- Linker support useful



Multiple-space Memory



- Examples: 68K option, 8051
- May need C language extensions

Virtual Memory

- Increase apparent memory size
- Swap data on/off of disk
- Not used much in embedded systems
- Not real time

Cache Memory

- Not strictly a memory architecture
- May often be ignored by programmers
- Optimization key to effective use

Memory Management Units

- Mainly 32 bit processors
- May be built-in or option
- Provides protection of memory
- Generally managed by RTOS
- 2 approaches:
  - blocking [write-protecting] memory areas
  - process model implementation

Memory Architecture – Conclusion

- Understanding the memory architecture of a chip is essential to determine its appropriateness for a specific application
- For very large or very small applications, flat memory is usually best
- For small programs with a lot of data, bank-switched memory may be particularly suitable