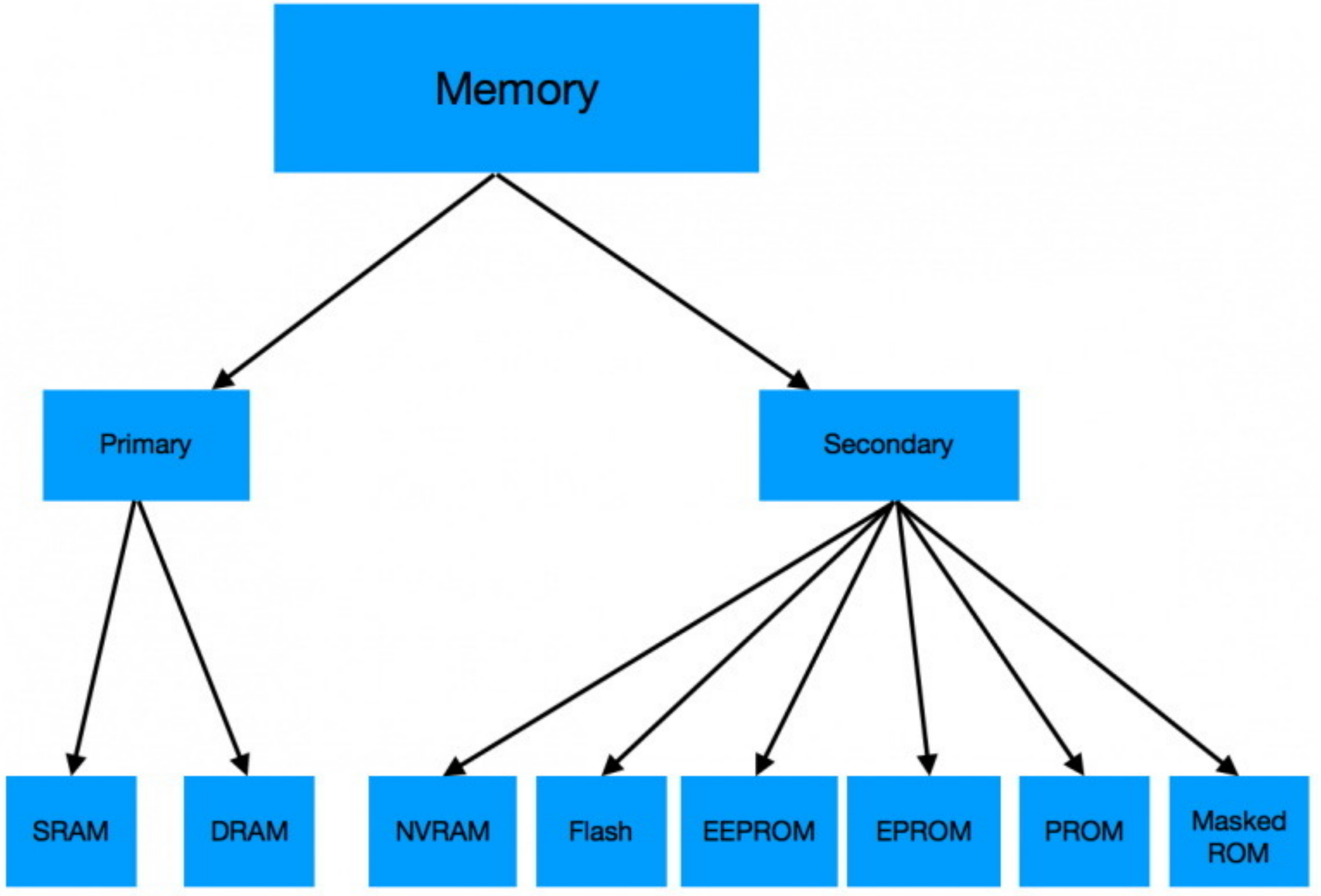
Memory

Memory Classification

1. SRAM
2. DRAM
3. Masked ROM
4. PROM
5. EPROM
6. EEPROM
7. flash memory and
8. NVRAM



Distinctive Features

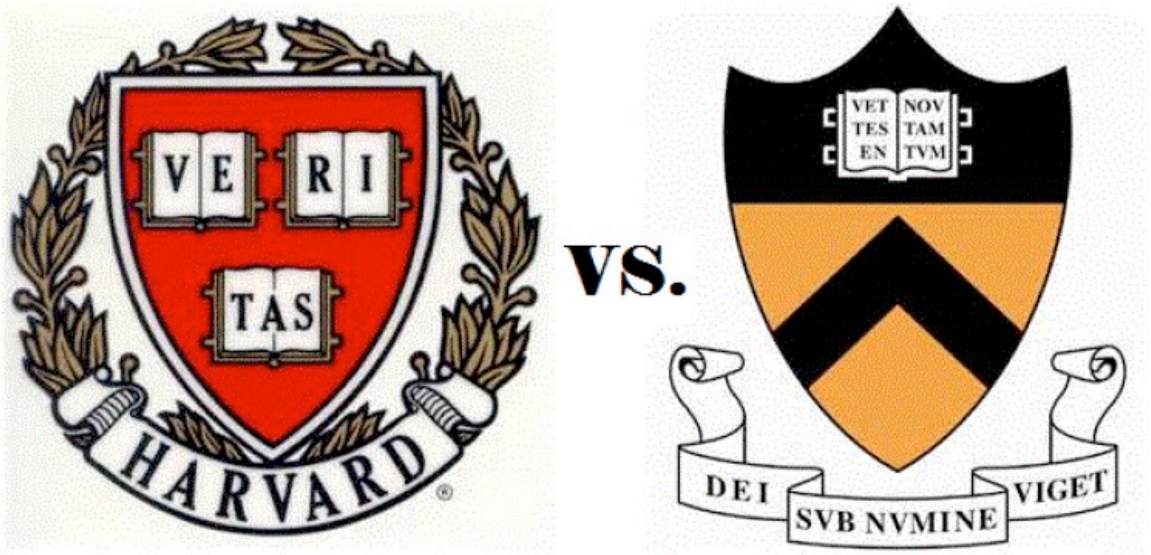
- Primary Memory very fast, but it cannot hold data without power , while Secondary is relatively slow but can hold data without power.
- SRAM is ~4x times the speed of DRAM , higher complexity 6 transistors (cross coupled flip flops) design , more expensive and eventually more power drain.(row select)
- DRAM requires a refresh for its 1 capacitor & 1 transistor circuit (by DRAM controller) every few milliseconds or else it will end up erased. (tri state flip flop top access cell)
- Usually SRAM is smaller capacity than DRAM in most microcontrollers.

Distinctive Features

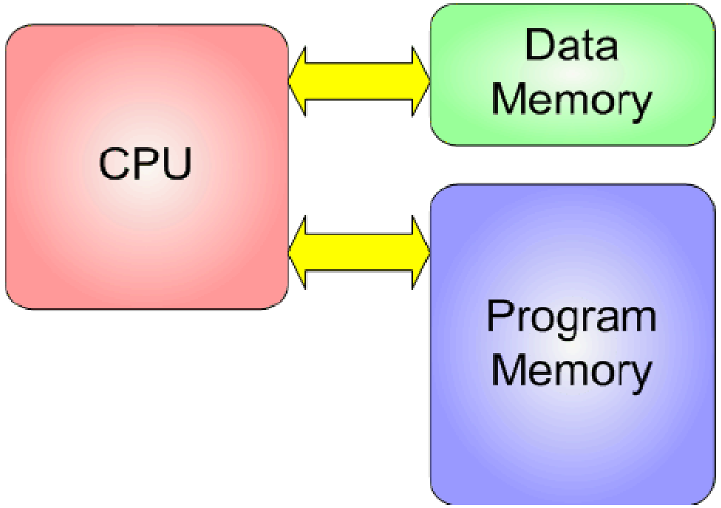
- A Masked ROM is manufactured with data that can not be changed (manufacture attributes) , while a PROM is a one time programmable read only memory (firmware).
- EPROM is erasable using UV(glass window over chip) , EEPROM is electrically erasable programmable read only memory (which made it read/write) (runtime constants and updatable firmware).
- Selecting your chip memory capacities is affected by how optimized your code is. The higher the capacity the higher the system cost.

Memory Architectures

- Von Neumann vs Harvard's
  - Why in PC ?
- Single Data/Program Path



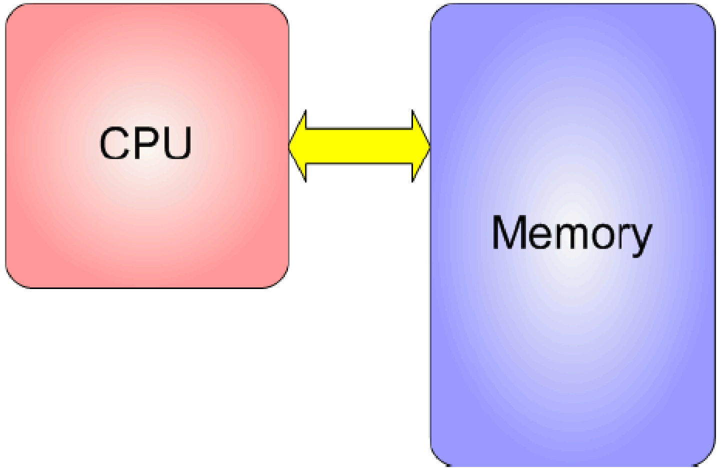
- Separate Paths (Harvard)



CPU

Data Memory

Program Memory



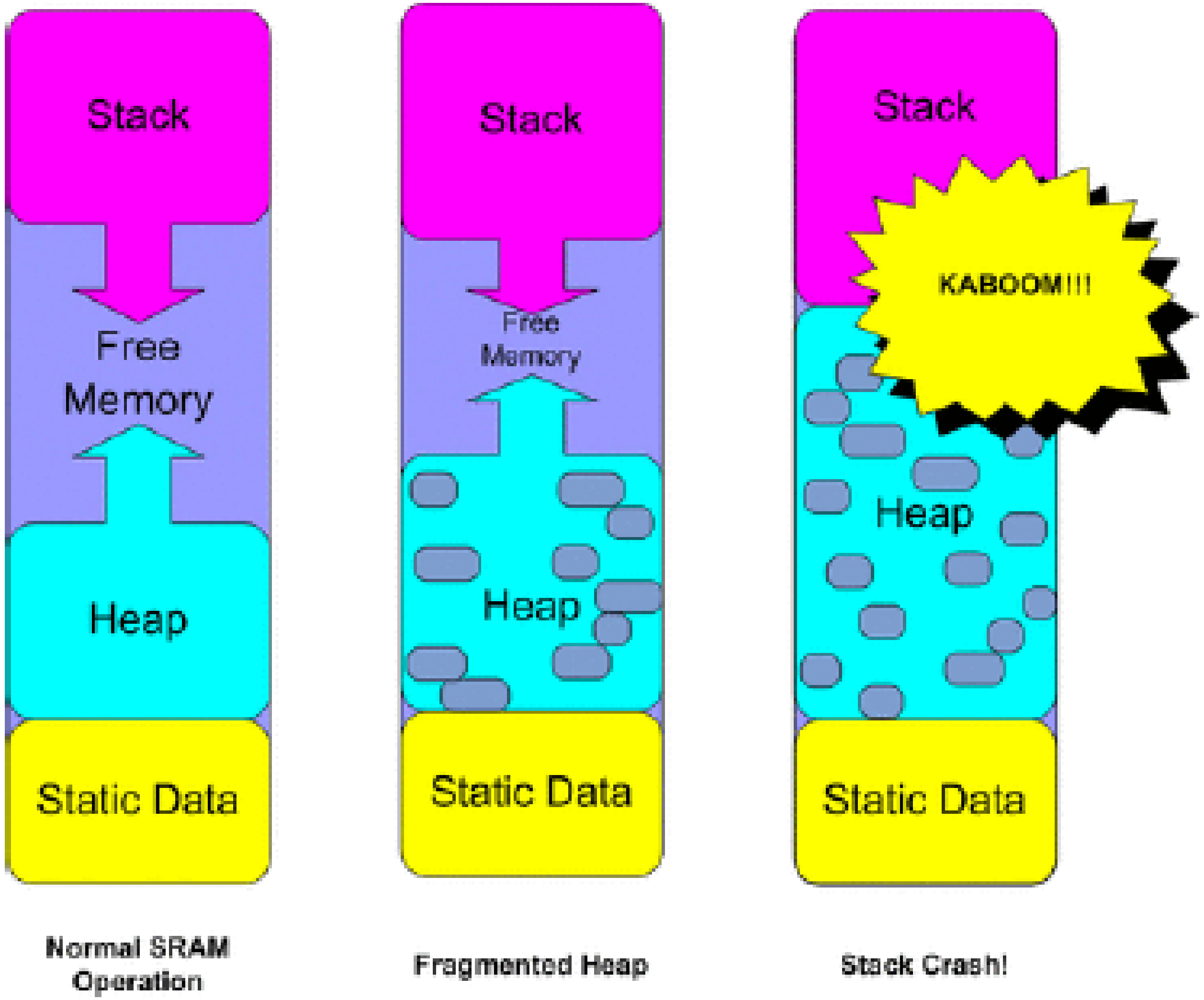
CPU

Memory

- Flexibility vs Performance

Memories of AVR / Arduino

Coding like on a PC ?

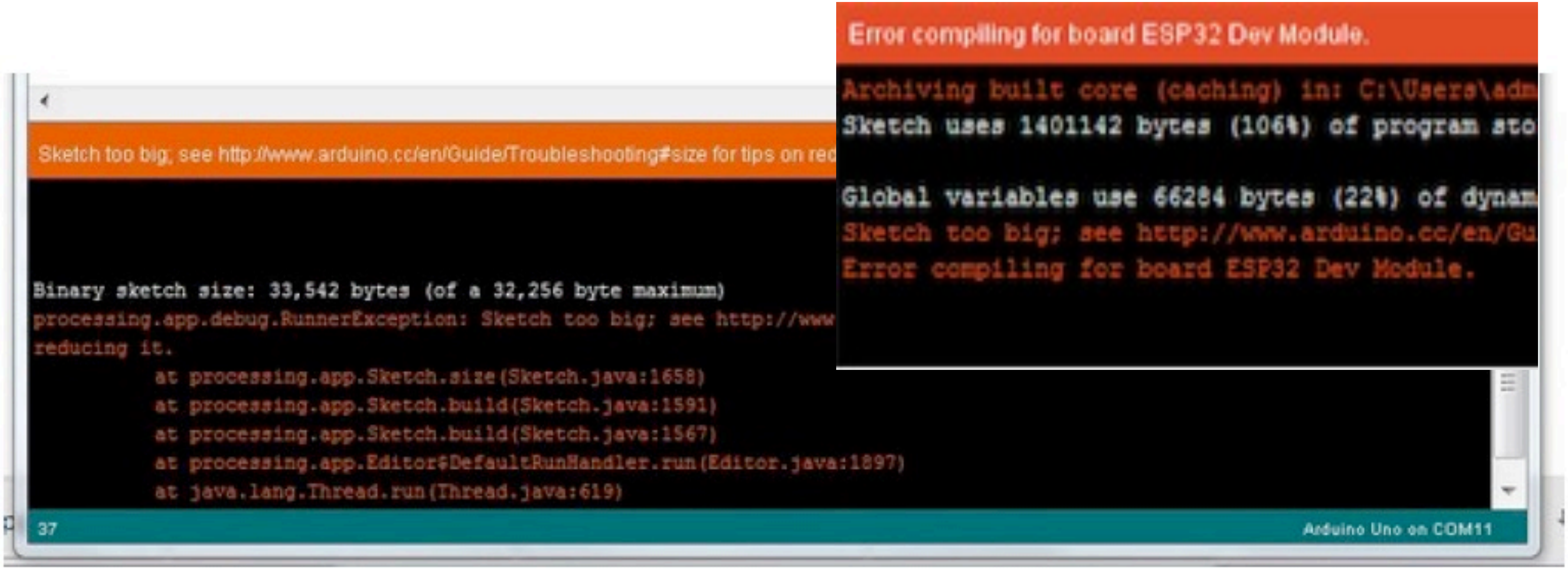


Normal SRAM Operation

Fragmented Heap

Stack Crash!

Sketch Too Big



Sketch too big; see http://www.arduino.cc/en/Guide/Troubleshooting#size for tips on red

```
Binary sketch size: 33,542 bytes (of a 32,256 byte maximum)
processing.app.debug.RunnerException: Sketch too big; see http://www
reducing it.
        at processing.app.Sketch.size(Sketch.java:1658)
        at processing.app.Sketch.build(Sketch.java:1591)
        at processing.app.Sketch.build(Sketch.java:1567)
        at processing.app.Editor$DefaultRunHandler.run(Editor.java:1897)
        at java.lang.Thread.run(Thread.java:619)
```

37

Arduino Uno on COM11

Error compiling for board ESP32 Dev Module.

```
Archiving built core (caching) in: C:\Users\adm
Sketch uses 1401142 bytes (106%) of program sto

Global variables use 66284 bytes (22%) of dynam
Sketch too big; see http://www.arduino.cc/en/Gu
Error compiling for board ESP32 Dev Module.
```

My Program was fine until :

- "Included one more library"
- "Added some more LED pixels"
- "Opened a file on the SD card"
- "Initialized a graphical display"
- "Merged in another sketch"
- "Added a new function

AVR Architecture for atmega328p

- Harvard architecture
- Flash for sketch (program)
- SRAM for data

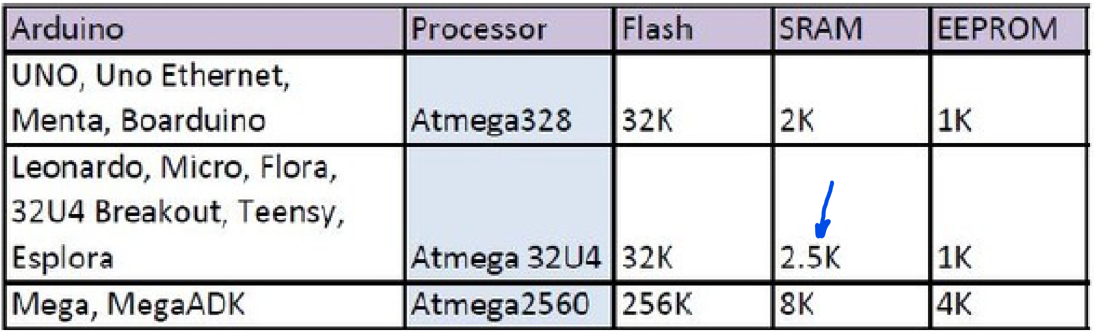
- Compiler and Runtime system (Arduino boot loader) works to handle allocation and deallocation , as well as task space /segment protection. (and Swapping in other OSs)

Getting into Memory Management

- Flash 32 KB
- SRAM 2 KB
- EPROM 1KB

- 10,000 to 100,000 times less than PC memory

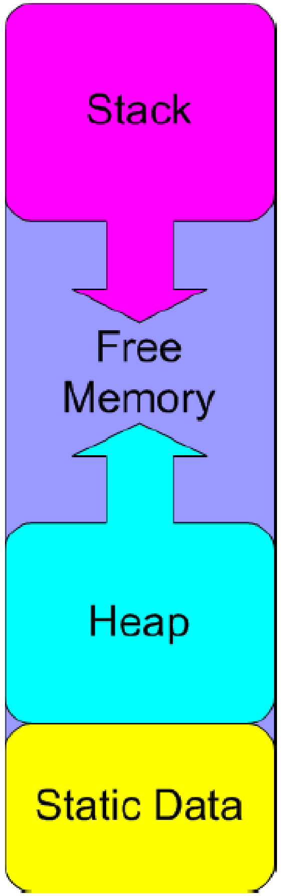
Differences



Arduino	Processor	Flash	SRAM	EEPROM
UNO, Uno Ethernet, Menta, Boarduino	Atmega328	32K	2K	1K
Leonardo, Micro, Flora, 32U4 Breakout, Teensy, Esplora	Atmega 32U4	32K	2.5K	1K
Mega, MegaADK	Atmega2560	256K	8K	4K

men byt7t fen

Flash	SRAM hwaa hwa da el by5zn el data el dynamic - drgt 7rart el tkyeef hwa dae	EPROM
Program byt7t fl flash	Data Static Data Reserved Block Constants - Globals – Initial values	Byte by Byte operation
Static , 10 , 000 write cycles , fused down when written .	Volatile and Dynamic Heap : Dynamically allocated data items growing up Stack : function calls and interrupr handler calls growing down	Slower than SRAM , with 100,000 write cycles



Free Memory

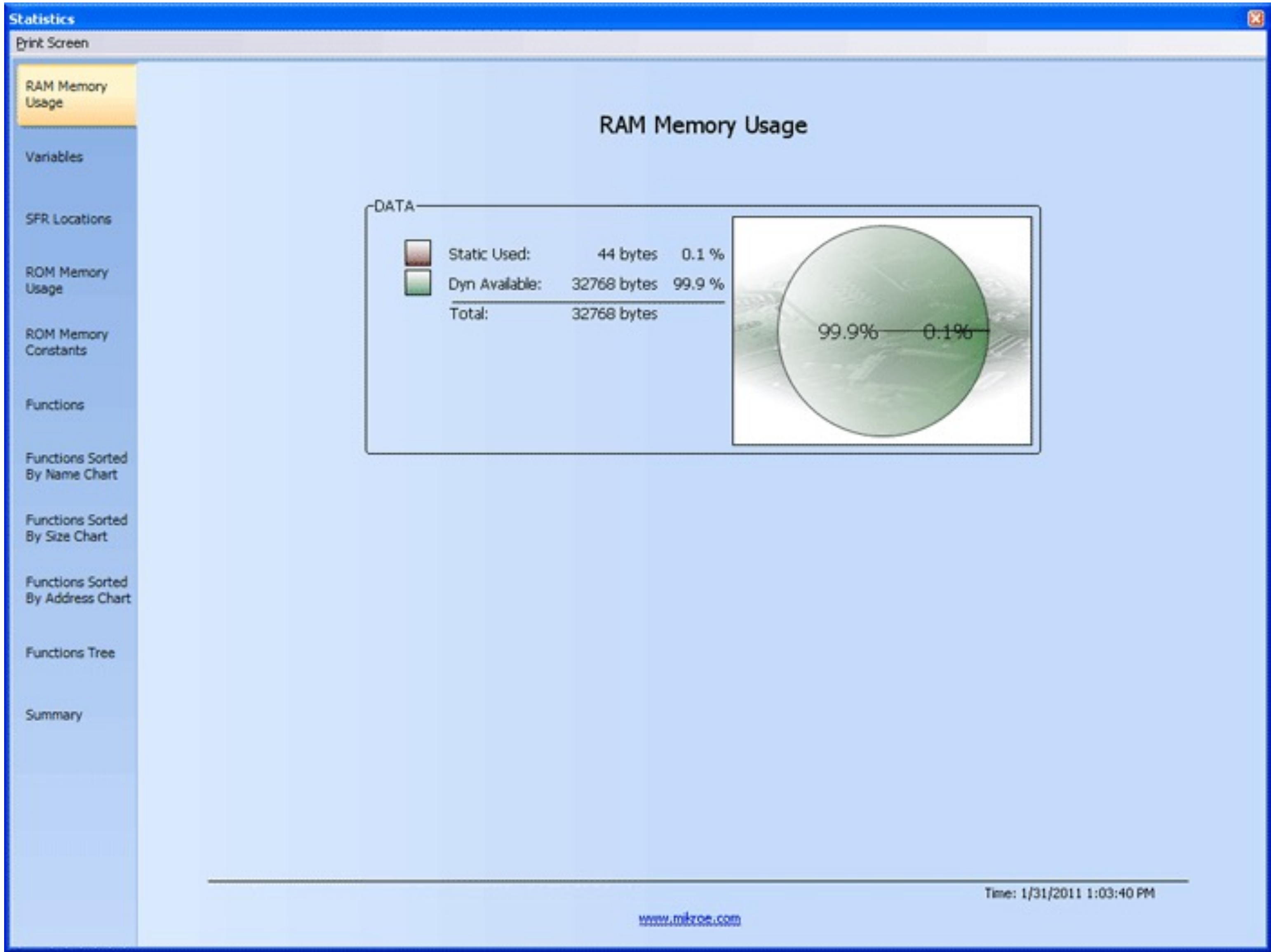
- Distance between Heap and Stack
- Defragmentation of Heap computer architecture algorithms
    - FF (first fit) algorithm, where the list is scanned from the beginning for the first “hole” that is large enough.
    - NF (next fit) where the list is scanned from where the last search ended for the next “hole” that is large enough.
    - BF (best fit) where the entire list is searched for the hole that best fits the new data.
    - WF (worst fit), which places data in the largest available “hole.”
    - QF (quick fit) where a list is kept of memory sizes and allocation is done from this information.

Measuring Free Memory

- MikroC Statistics Library
- freeMemory()

routine for Arduino

https://github.com/mpflaga/Arduino-MemoryFree



RAM Memory Usage

DATA

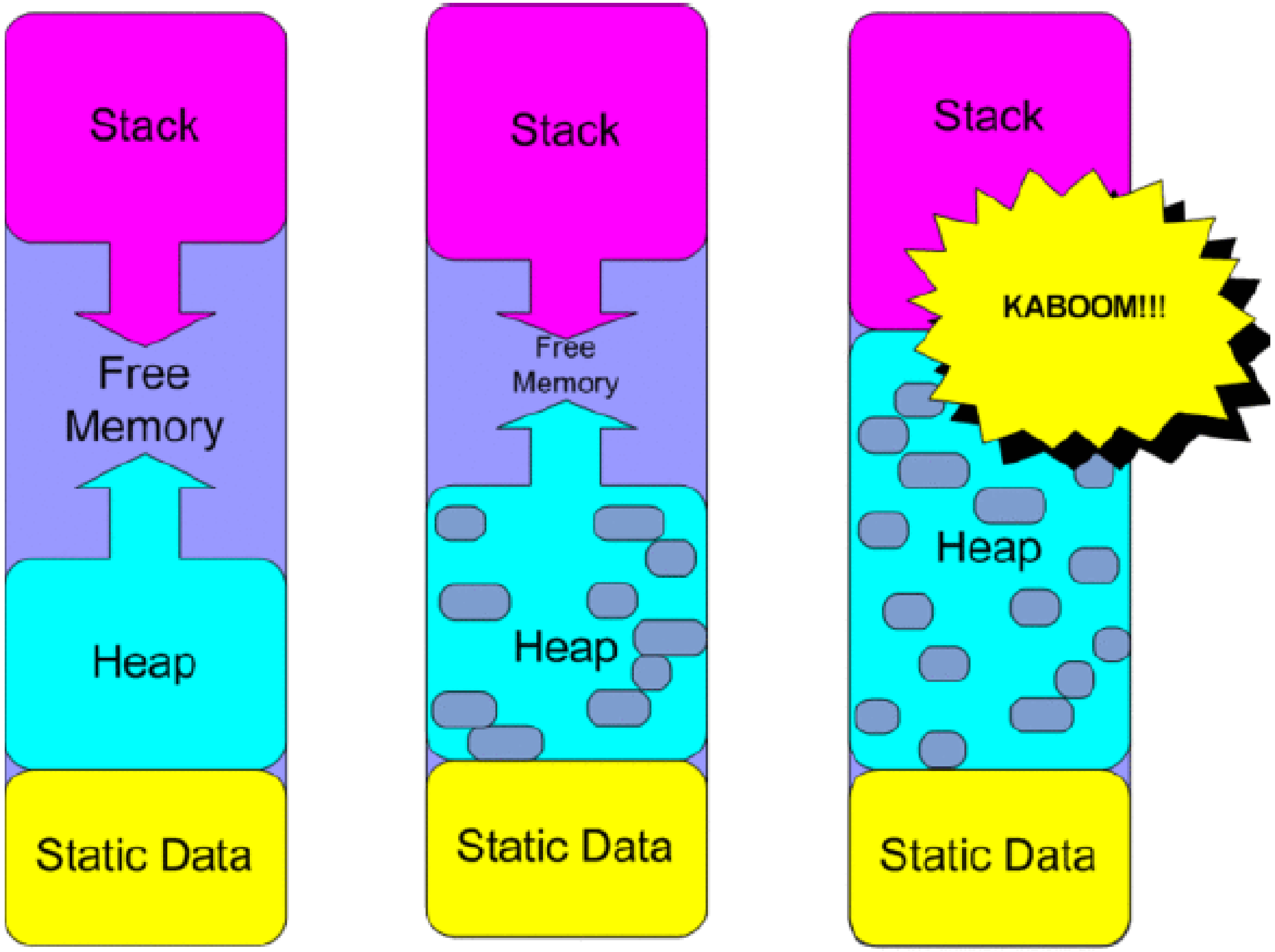
🟥	Static Used:	44 bytes	0.1 %
🟩	Dyn Available:	32768 bytes	99.9 %
	Total:	32768 bytes	

Time: 1/31/2011 1:03:40 PM

High SRAM usage

- 3 bytes per pixel in RGB
    - Would you run a 32x32 pixel RGB display on Uno 328p ?
- 1 Byte per 8 pixels in Monochrom
- Communication buffers reserved for I/O

Optimizing Memory Usage



Normal SRAM Operation

Fragmented Heap

Stack Crash!

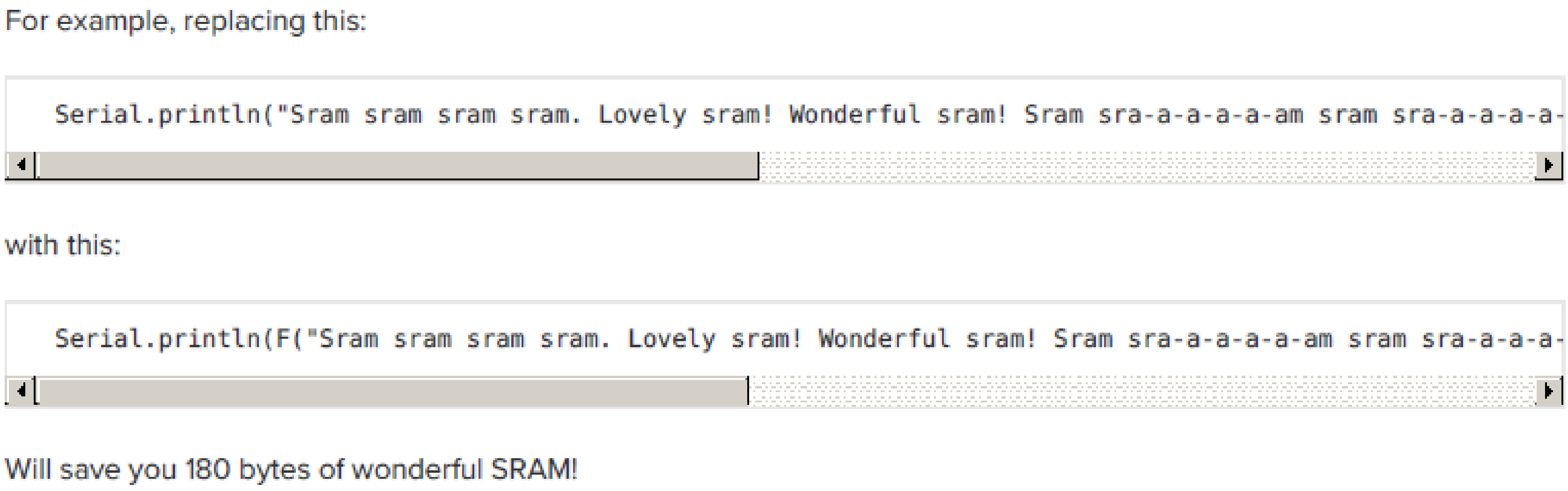
Optimizing Flash Usage – Application Code

- Remove Dead Code
- Unused Libraries (just comment and compile)
- Unused functions
- Unused variables
- Unreachable code

- Refactor (repeated code)
- Eliminate the Bootloader ** special considerations.

Optimizing SRAM Usage – Data and Dynamic Allocations

- Remove unused variables
- F() fixed Strings to pointers to base address in Flash



For example, replacing this:

```
Serial.println("Sram sram sram sram. Lovely sram! Wonderful sram! Sram sra-a-a-a-a-am sram sra-a-a-a-
```

with this:

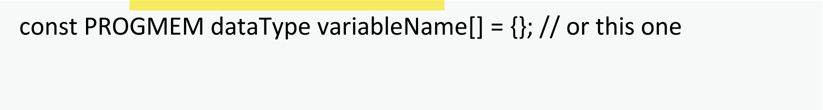
```
Serial.println(F("Sram sram sram sram. Lovely sram! Wonderful sram! Sram sra-a-a-a-a-am sram sra-a-a-a-
```

Will save you 180 bytes of wonderful SRAM!

- Reserve() for Growing Strings

Optimizing SRAM Usage – Data and Dynamic Allocations

- Moving Constants to PROGMEM

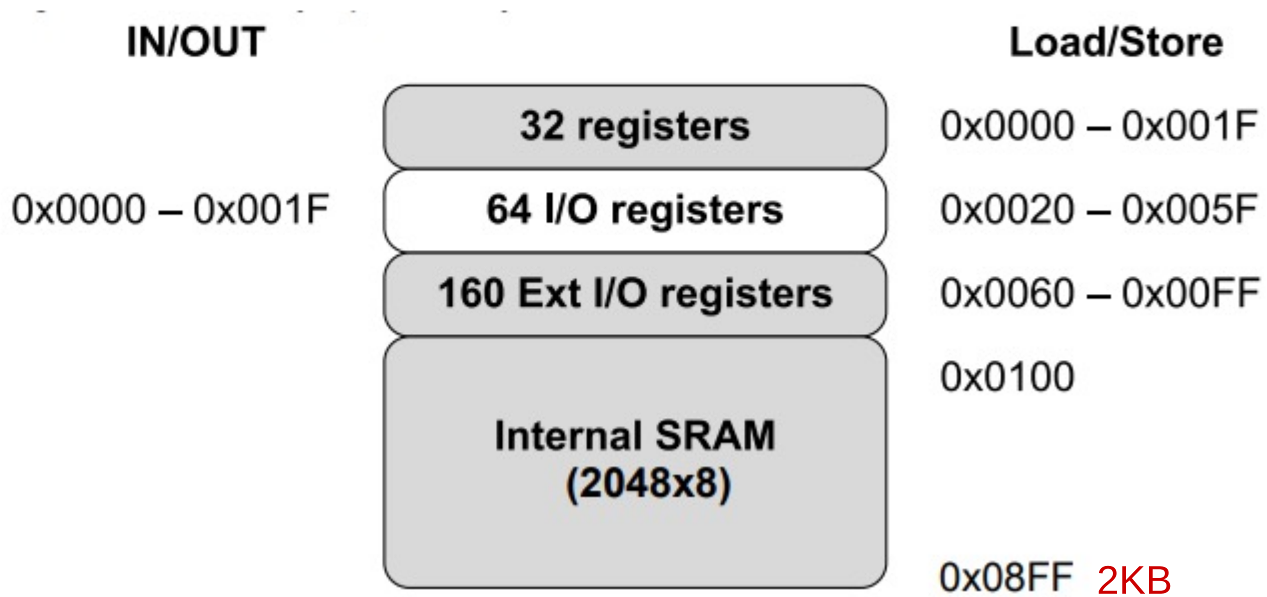


```
const PROGMEM dataType variableName[] = {}; // or this one
```

- Reducing unused buffer sizes or Data types
  - Serial Buffer Size for fast communication 64 Bytes , make it 32 bytes
- Allocate local variables in a function scope to ensure cleanup with stack pop.
- Avoid dynamic heap allocations - These can quickly fragment the limited heap-space.
- Prefer local to global allocation - Stack variables only exist while they are being used. If you have variables that only are used in a small section of your code, consider making that code into a function and declaring the variables local to the function.

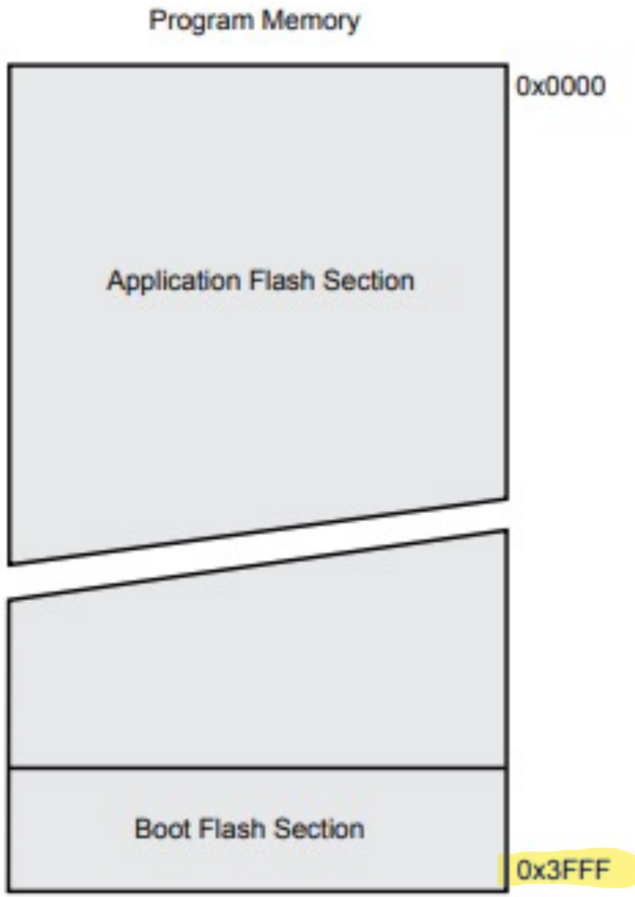
Data & Program Memory Maps of Atmega328p

- Data Memory Map



IN/OUT		Load/Store
	32 registers	0x0000 – 0x001F
0x0000 – 0x001F	64 I/O registers	0x0020 – 0x005F
	160 Ext I/O registers	0x0060 – 0x00FF
	Internal SRAM (2048x8)	0x0100
		0x08FF

2KB



EEPROM from 0 to 1023

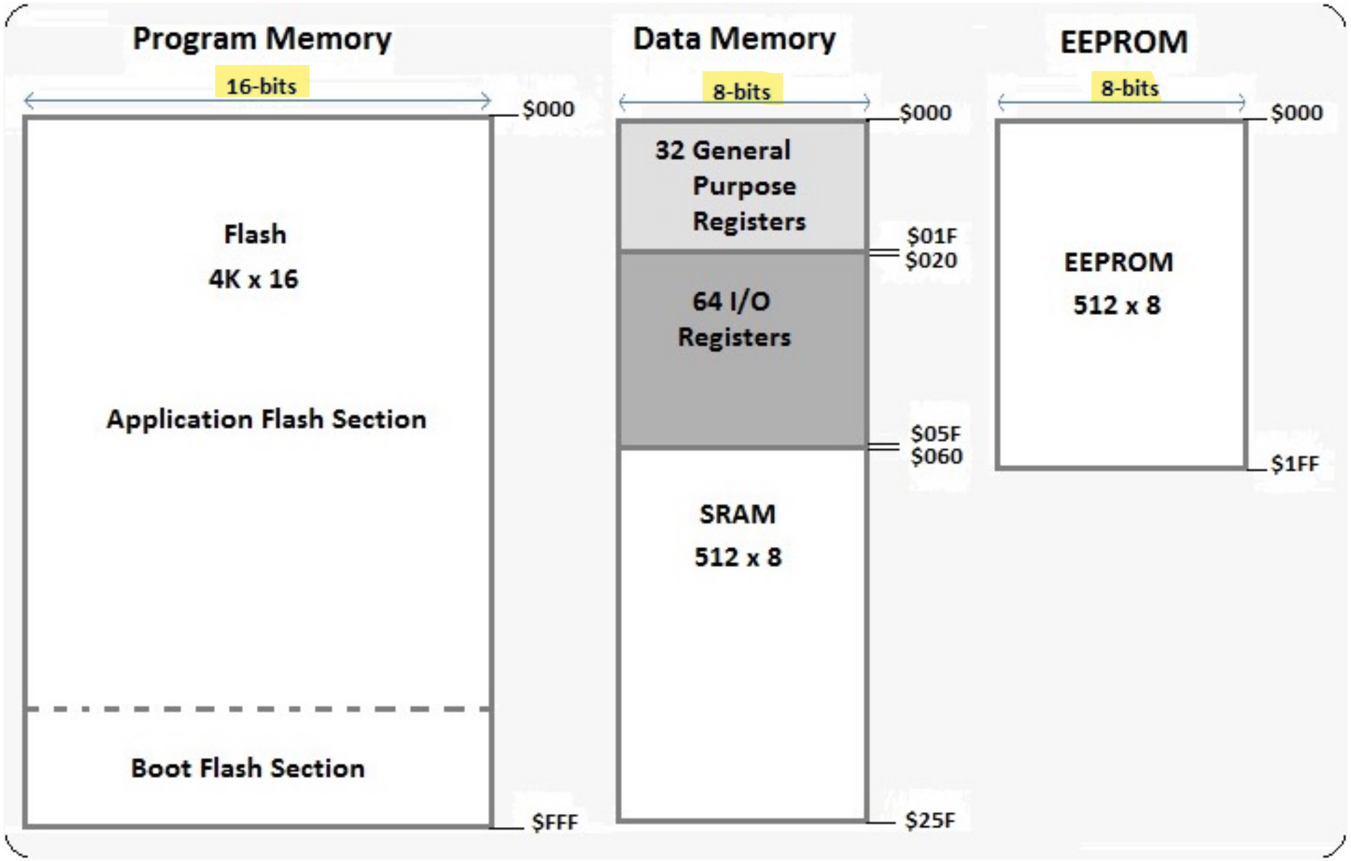
el size = 32KB calculated

https://cdn.sparkfun.com/assets/c/a/8/e/4/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf

Other Microcontrollers

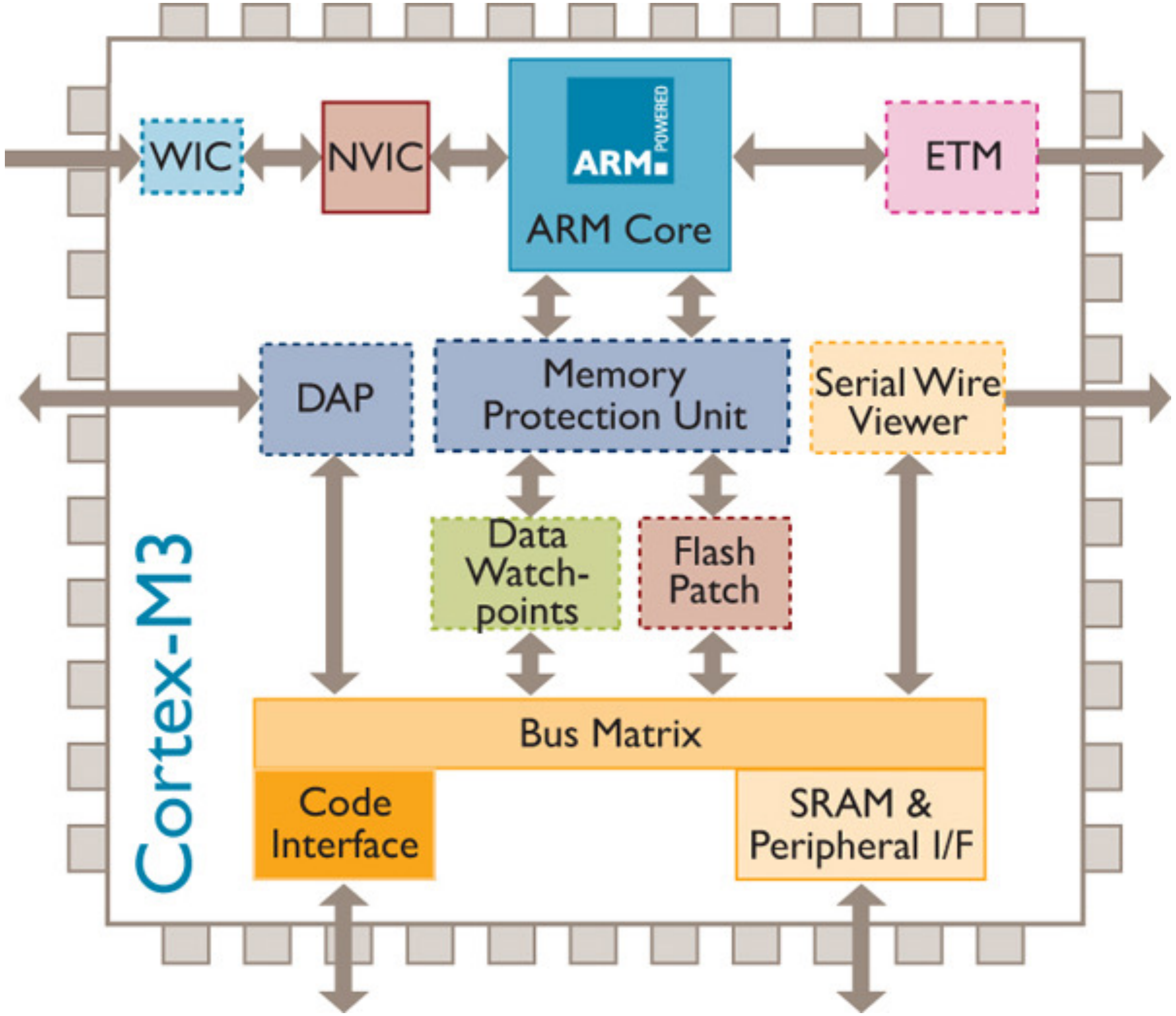
- **ATMega8515**
- *Data Memory Contains:*
  - 32 8-bits General Purpose Regs.
  - 64 8-bits Input/Output Registers
  - 512 8-bits SRAM space
- *Program Memory Contains:*
  - 8K byte Flash Memory
  - Organized as 4K-16bits

  Ending by FFF = 4095 ie 4096 Locations = 4 x 1024



AVR Tutorials Diagrams
www.AVR-Tutorials.com

Memory Management Unit



Memory Management Unit

- Managing the mapping between logical (physical) memory and task memory references.
- Determining which processes to load into the available memory space.
- Allocating and deallocating of memory for the processes that make up the system.
- Supporting memory allocation and deallocation of code requests (within a process), such as the C language “alloc” and “dealloc” functions, or specific buffer allocation and
- Tracking the memory usage of system components.
- Ensuring cache coherency (for systems with cache).
- Ensuring process memory protection.