The Abdus Salam
International Centre for Theoretical Physics

United Nations
Educational, Scientific and
Cultural Organization

IAEA
International Atomic Energy Agency

**1875-2**

# First Latin American Regional Workshop on Distributed Laboratory Instrumentation in Physics

***7 January - 4 February, 2008***

## TINI Platform.

Anthony J. Wetherilt

*Director, Administration*
*UNIDO-ICHET,*
*Sabri Ulker Sok, 38/4,*
*Cevizlibag, Zeytinburnu,*
*34015 Istanbul*

# The TINI Platform

*A.J. Wetherilt*

*UNIDO,*

*International Centre for Hydrogen Energy Technologies,*

*Istanbul,*

*Turkey*

# The TINI Platform

- *Introduction*
- *The TINI hardware*
- *The Runtime Environment*
- *The TINI boot sequence*
- *Using TINI for the first time*
- *Programming TINI*
- *The native packages*
- *Serial port programming*
- *Networking with the ethernet adapter*
- *The 1-Wire network*

# The TINI Hardware (1)
# The DS90C390

***DS90C390 hardware comprises:***

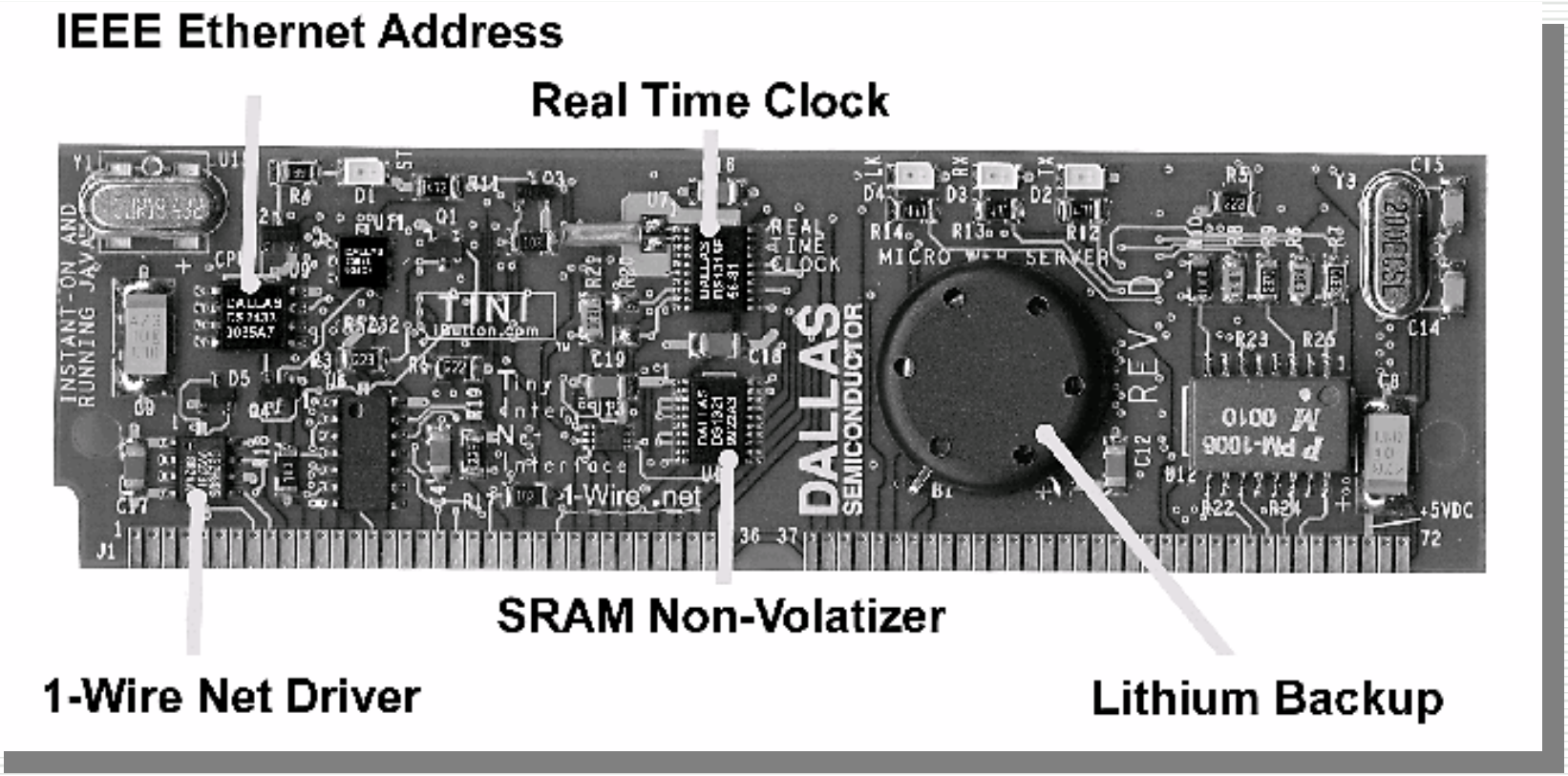
- ***Extended 8051 operating at 120MHz (40MHz xtal)***
- ***Up to 4Mbytes address space (22 bits)***
- ***4 kbytes internal SRAM***
- ***Dedicated maths accelerator for 32 bit arithmetic (40 bits accumulator)***
- ***3 Timers, 2 serial ports, Watchdog, IrDA, 2 CAN controllers***

# The TINI Hardware (2) DSTINI1

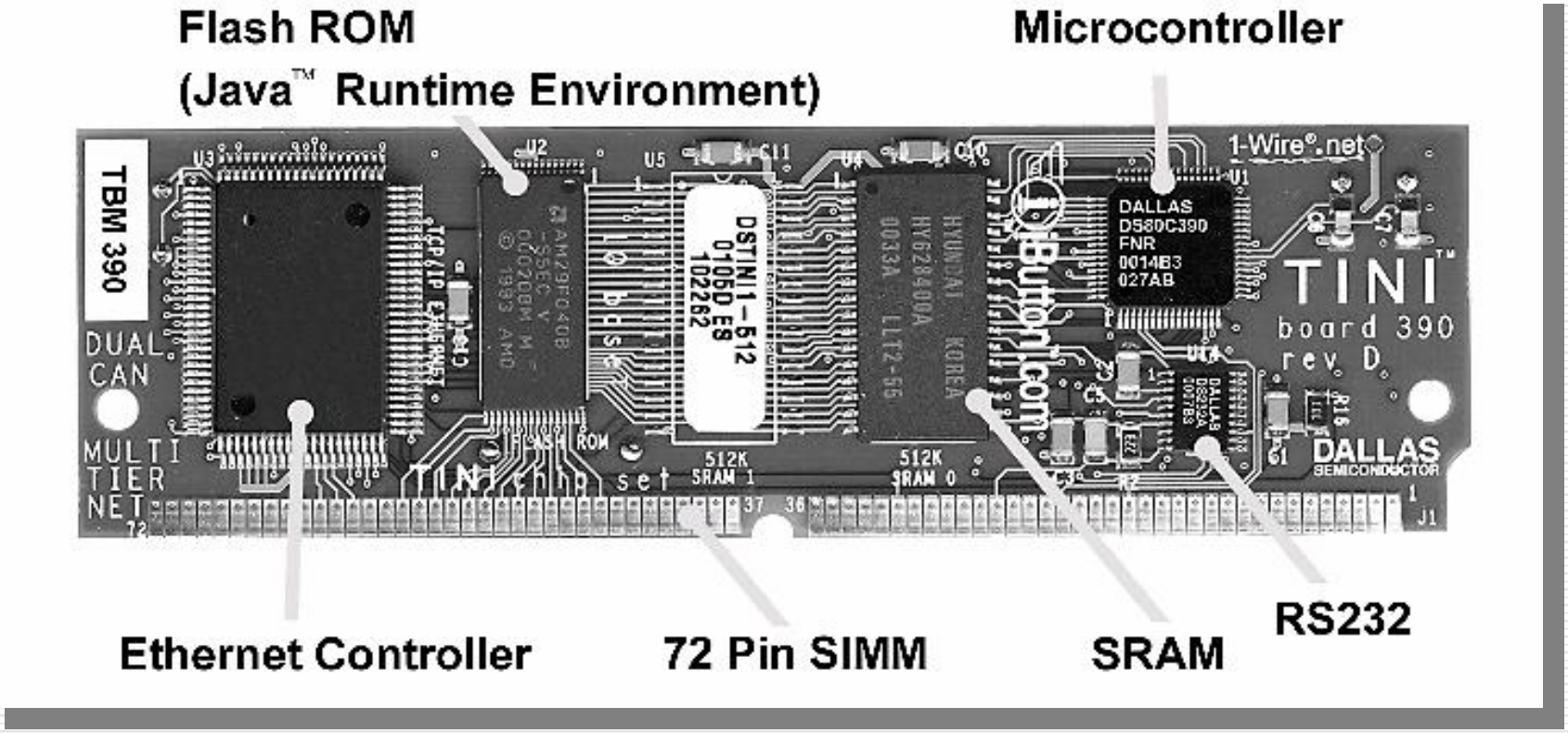
***External components:***

- ***2 x 512 kbyte static RAM***
- ***512k Flash (expandable to 1M)***
- ***10Base-T Ethernet controller***
- ***Serial communications (RS232, 1-wire)***
- ***Battery SRAM non-volatiser***
- ***Fits on to card with 72 pin SIMM connector***

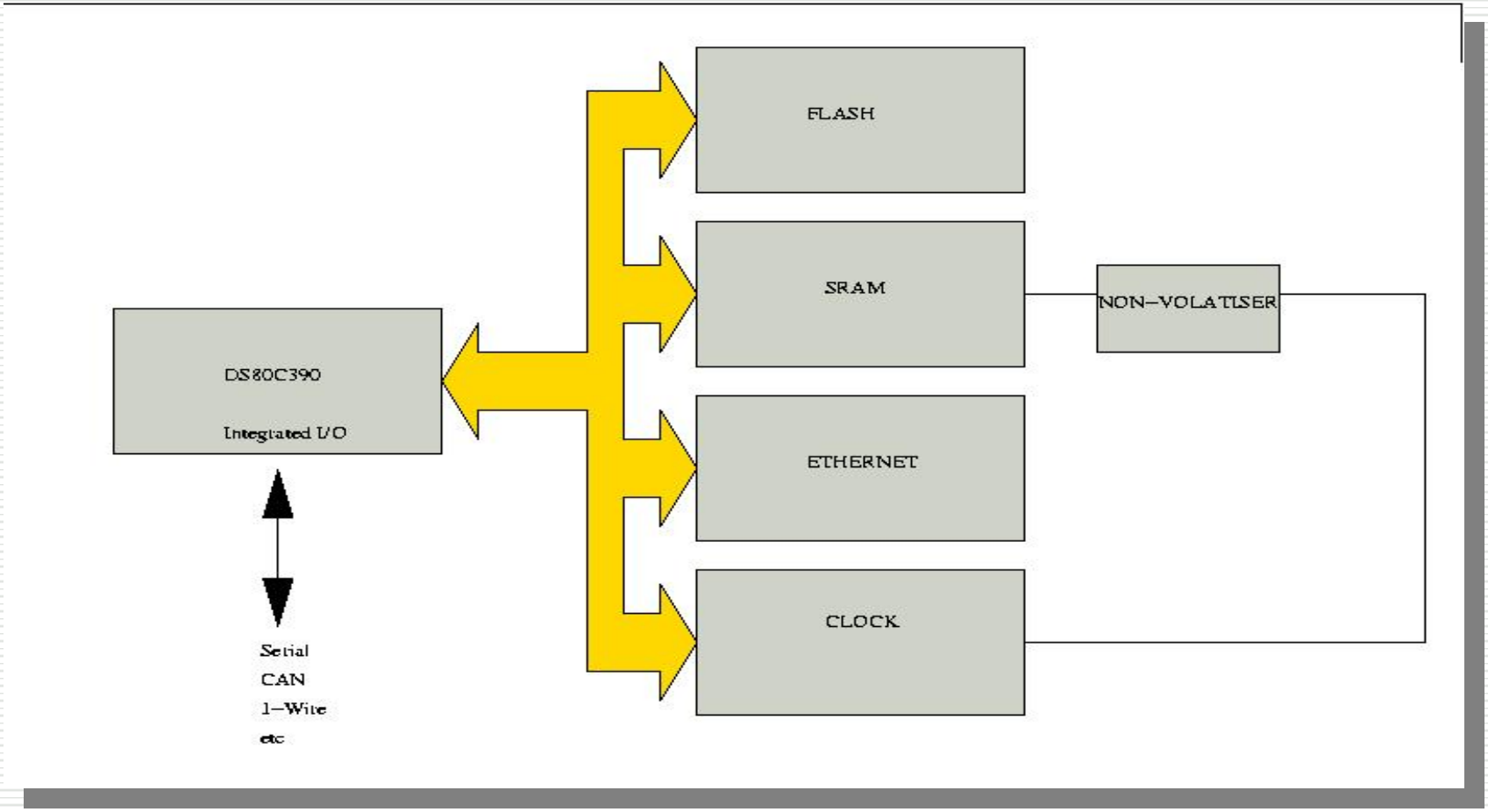
# The TINI Hardware (3)
# DSTINI1 - A



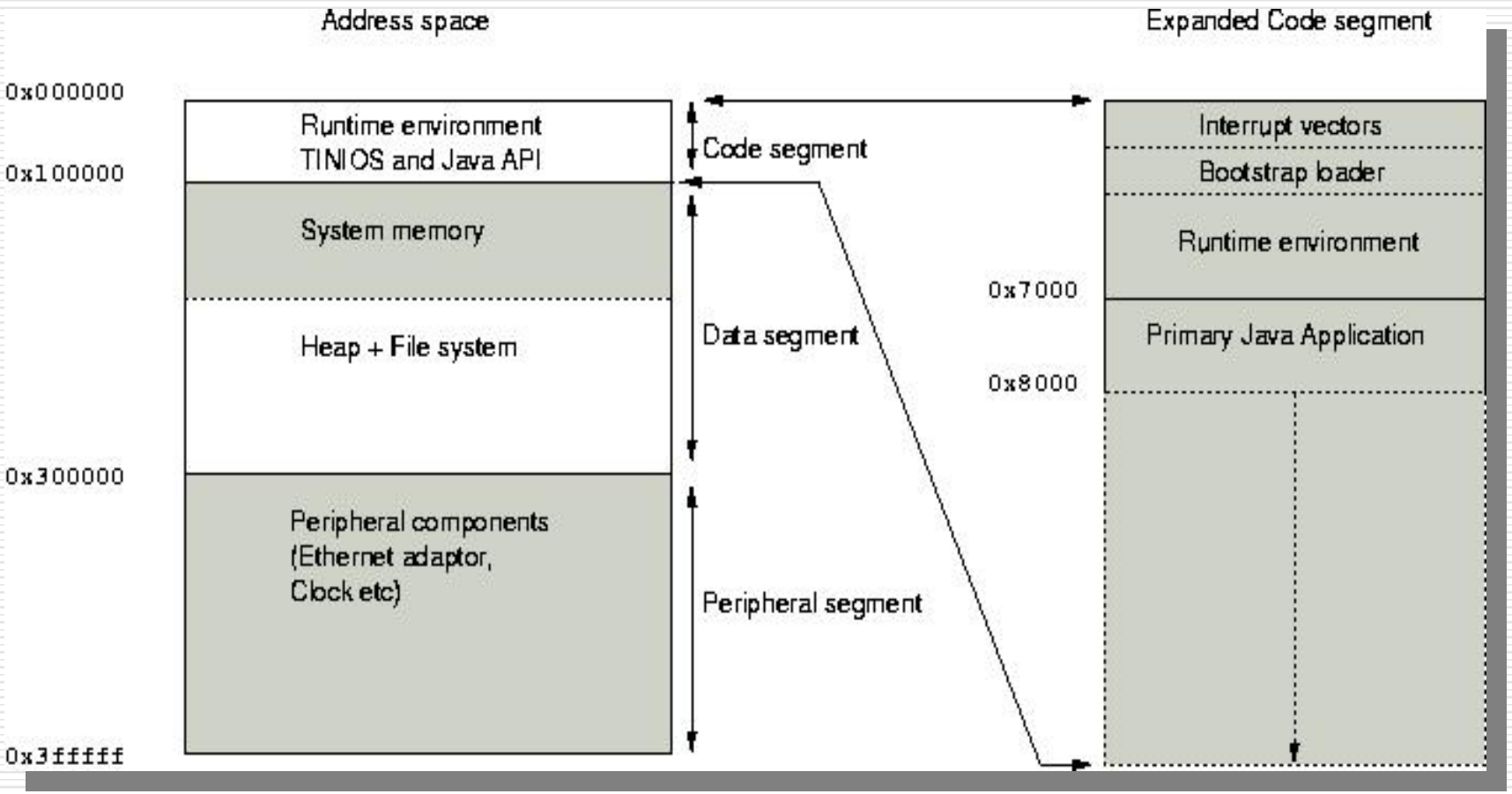
# The TINI Hardware (4)
# DSTINI1 - B



# The TINI Hardware (5)
# DSTINI1-functionality



# The TINI Hardware (6)
# DSTINI1 – Memory map



# The TINI Hardware (7) DSTINI1

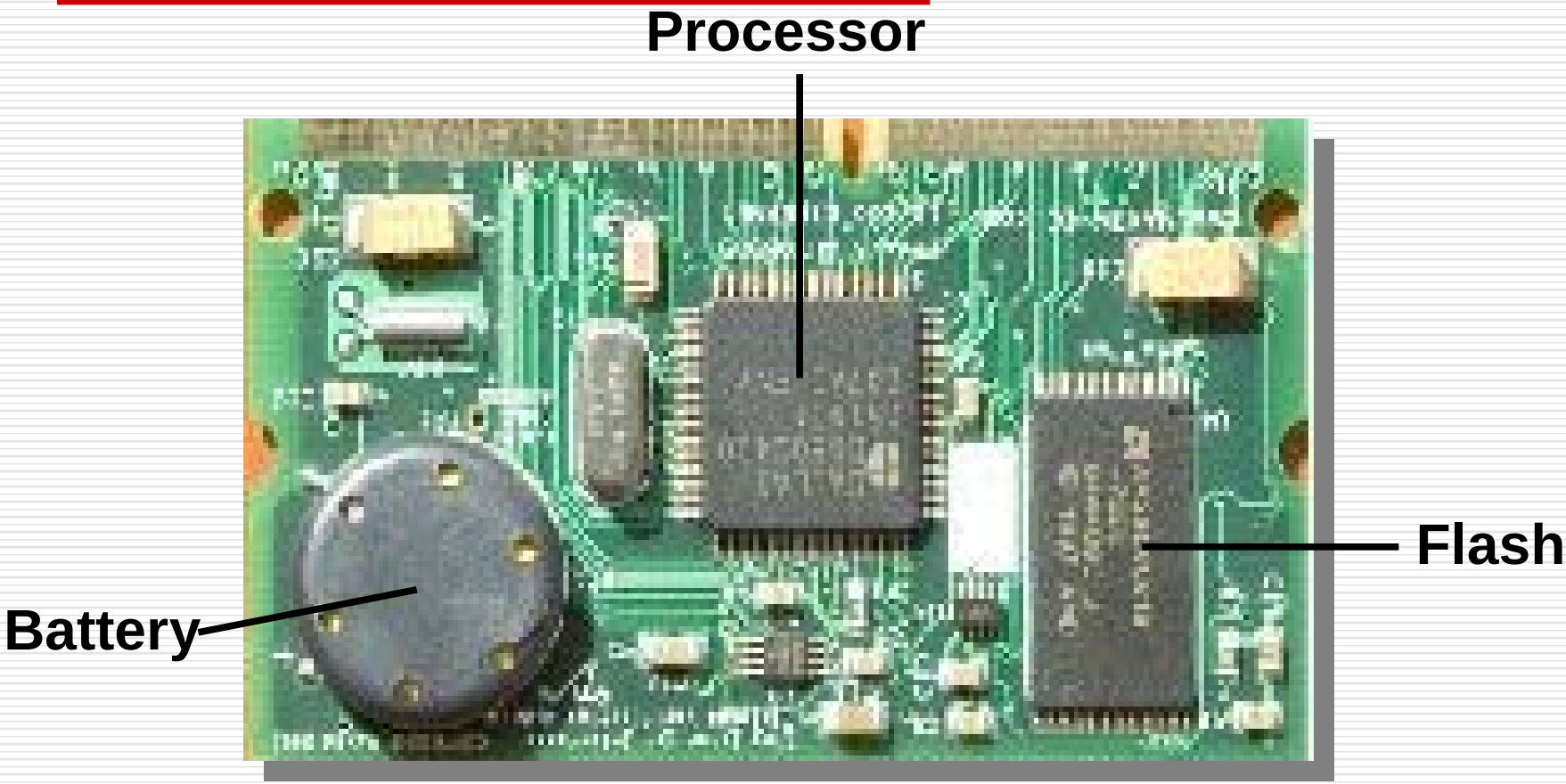
- *Variety of expansion boards commercially available*
- *We use ones provided by Dallas Semiconductor (DSTINIs-500/600)*
- *Single dc power supply with on-board regulation (s-600 only)*
- *RJ45 Ethernet, RJ11 1-wire, 2xDB9 connectors*

# The TINI Hardware (8)
# The DS90C400

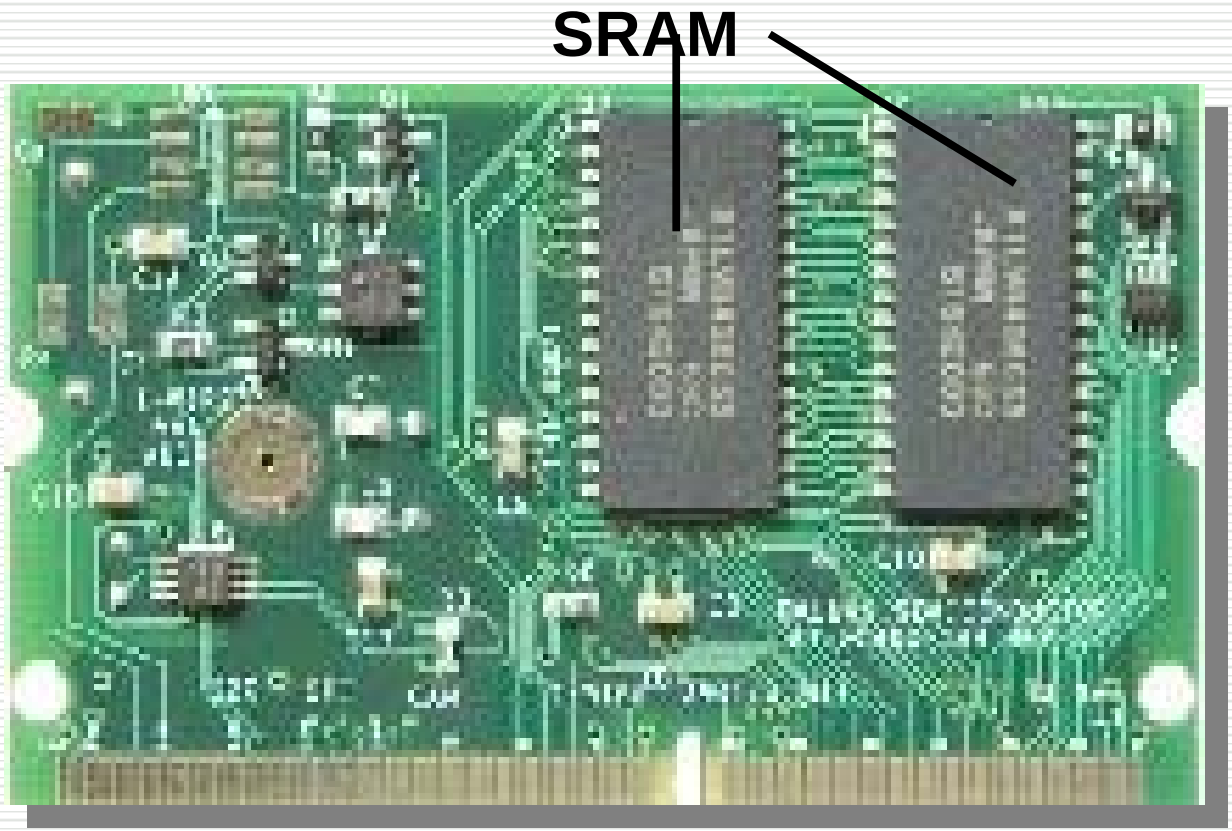
The DS90C400 extends the 390:

- *Max operating frequency of 75MHz*
- *16Mbytes addressable linear memory*
- *On board Ethernet controller*
- *Data instructions optimised*
- *1-Wire bus master implemented in h/w*
- *1 extra serial port*
- *ROM containing TCP/IP stack, boot loader etc.*

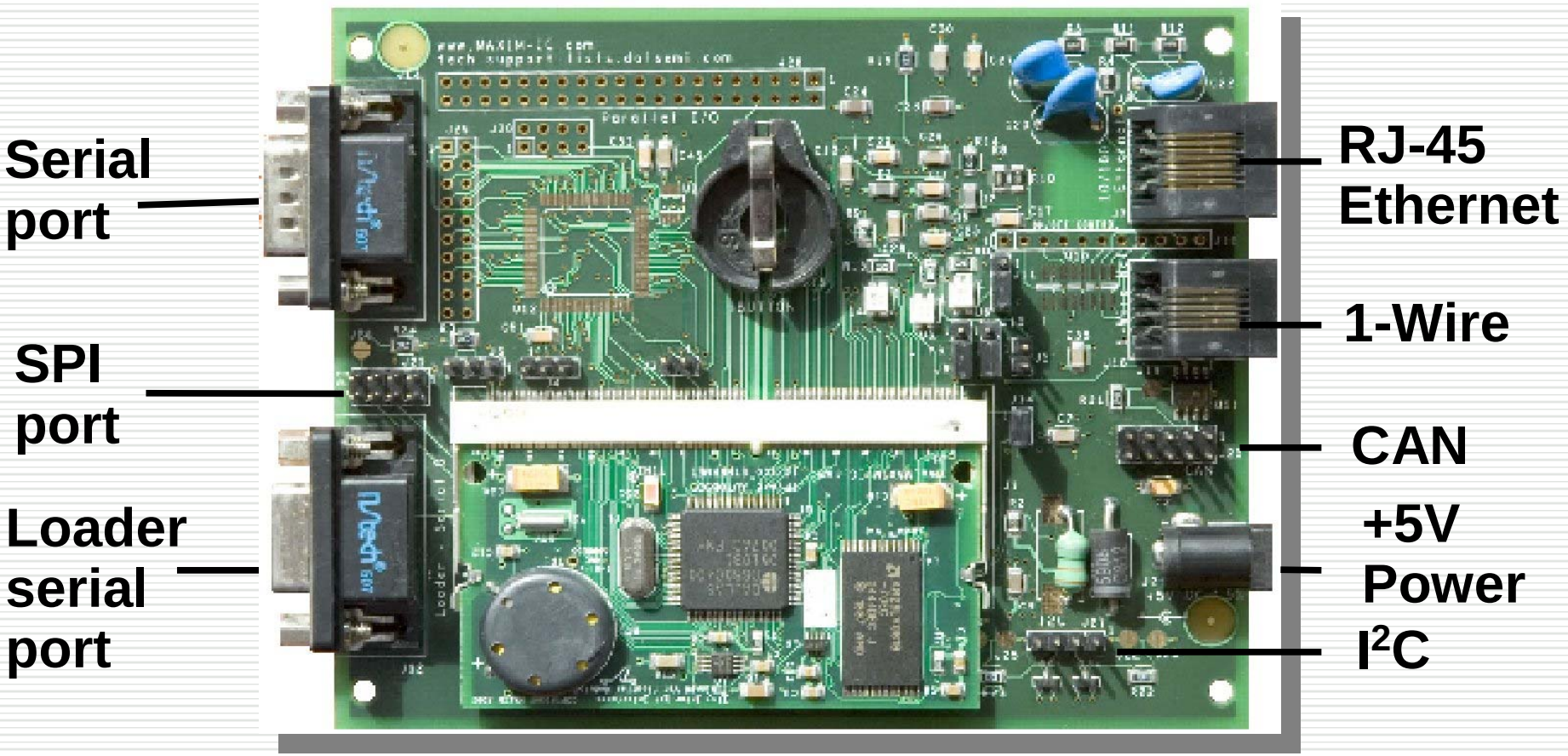
# The TINI Hardware (9)
# The DS90C400 - A



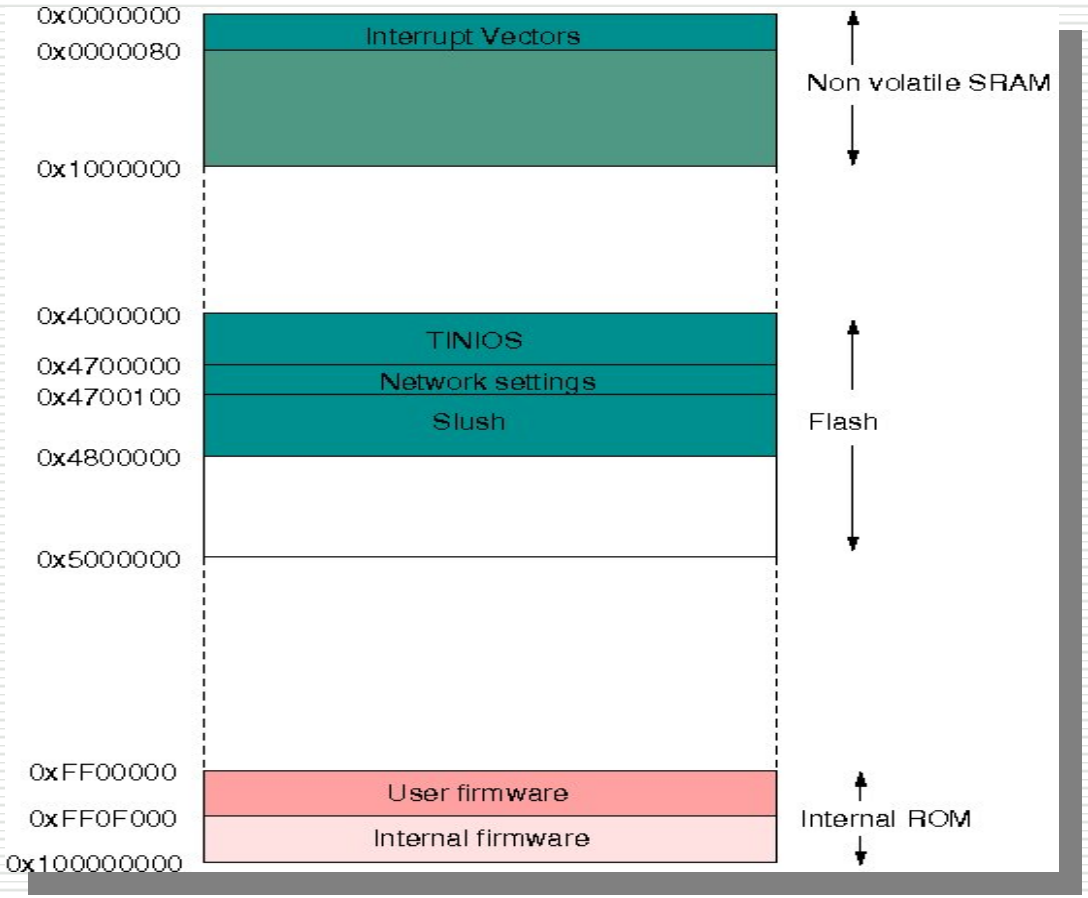
# The TINI Hardware (10)
# The DS90C400 - B



# The TINI Hardware (11)
# The DSTINIs400



# The TINI Hardware (12)
# The DS90C400 – Memory map



# The Development cycle with TINI

*Develop using TINI + extension board:*

- Add hardware to peripheral IO area as needed
- Develop drivers and other software to access hardware
- Develop system as needed
- Design new hardware layout once problems resolved
- Move software to new board

# The Runtime Environment

The Runtime Environment comprises:

- **API (Java + platform specific)**
- **Java Virtual Machine (JVM)**
- **TINI Native Interface (TNI)**
- **Operating system (TINIOS)**
- **Drivers to native hardware**

# The Runtime Environment (2)

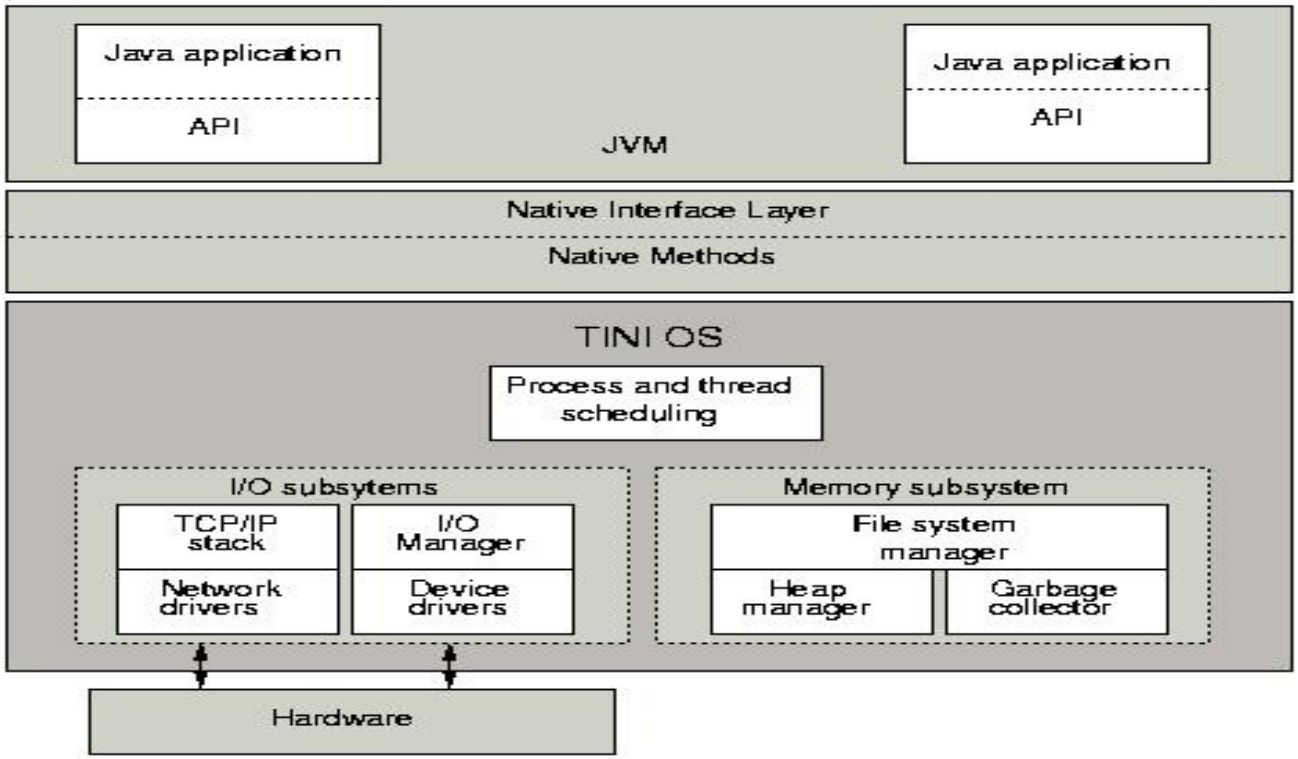
**API: Classes defined in JDK + platform specific**

- ***java.lang*** ***classes fundamental to java***
- ***java.io*** ***system i/o through file system***
- ***java.net*** ***networking***
- ***java.util*** ***miscellaneous utilities***

**Warning: Differences between these and JDK (see APIDiffs.txt)**

- ***com.dalsemi*** ***all platform specific classes***

# The Runtime Environment (3)



# The Runtime Environment (4)

*The Java Virtual Machine:*

- ***Occupies ~40kbytes***
- ***Full support for:*** ***Threads (16/32 max per proc), primitive types, strings***
- ***Not supported:*** ***Finalisation, (all) dynamic class loading,reflection, serialization (TINI1 only)***
- ***All classes must be defined either in API or compiled in directly during binary file creation using TINIConvertor/BuildDependency***

# The Runtime Environment (5)

*The TINI Native Interface (TNI):*

- ***Rarely needed directly but can be accessed using*** `loadlibrary(libname)`
- ***A very thin layer that acts as an interface between JVM and operating system***

  ***from***

  `java.lang.Runtime`

# The Runtime Environment (6)

*The TINI Operating System (TINIOS):*

- ***Scheduling for process and threads***
- ***Memory management***
- ***I/O management***

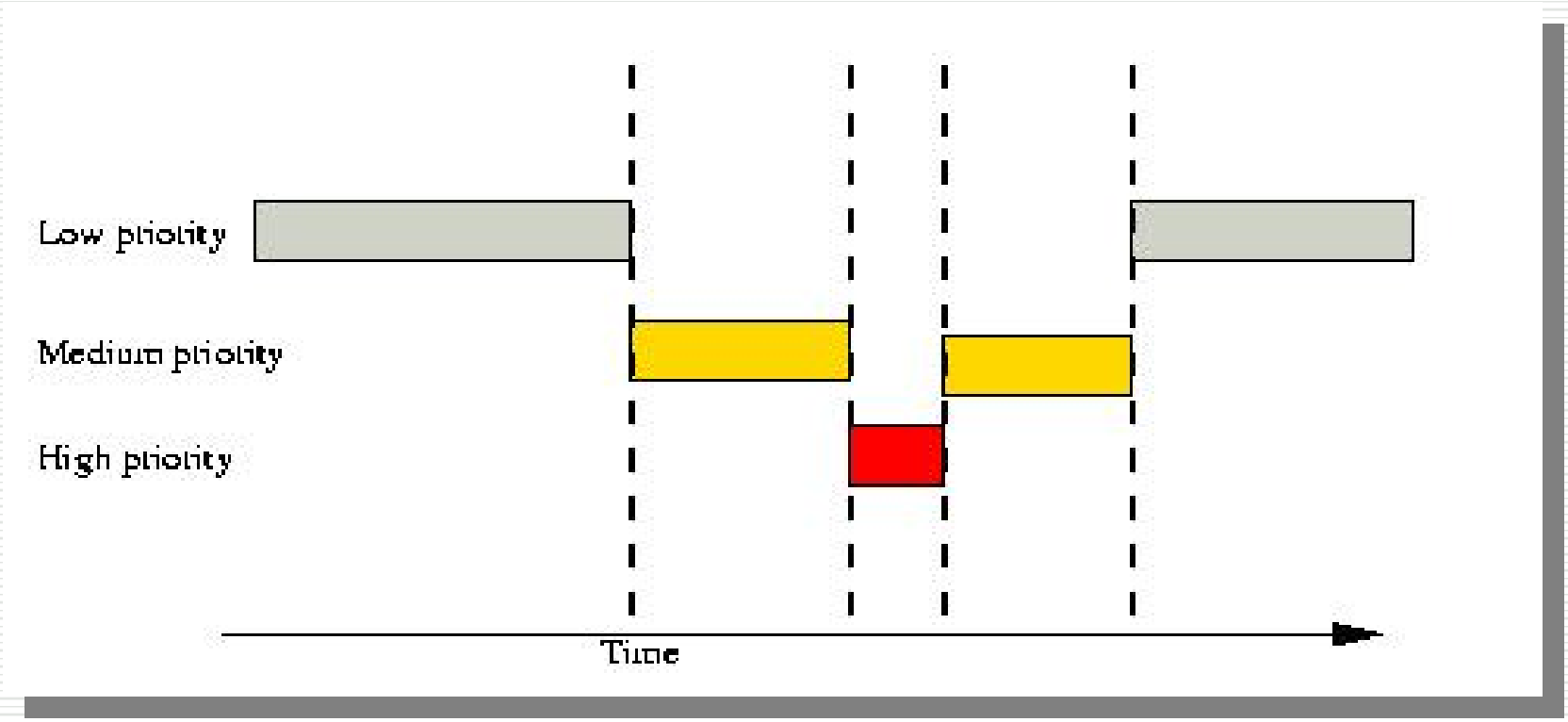
# The Runtime Environment (7)

*Scheduling:*

- ***In multitasking systems, need to switch between processes (context switching)***
- ***Various strategies involving tradeoffs between responsiveness to critical events and sharing processor time***
- ***Round robin scheduling with a 1ms clock***

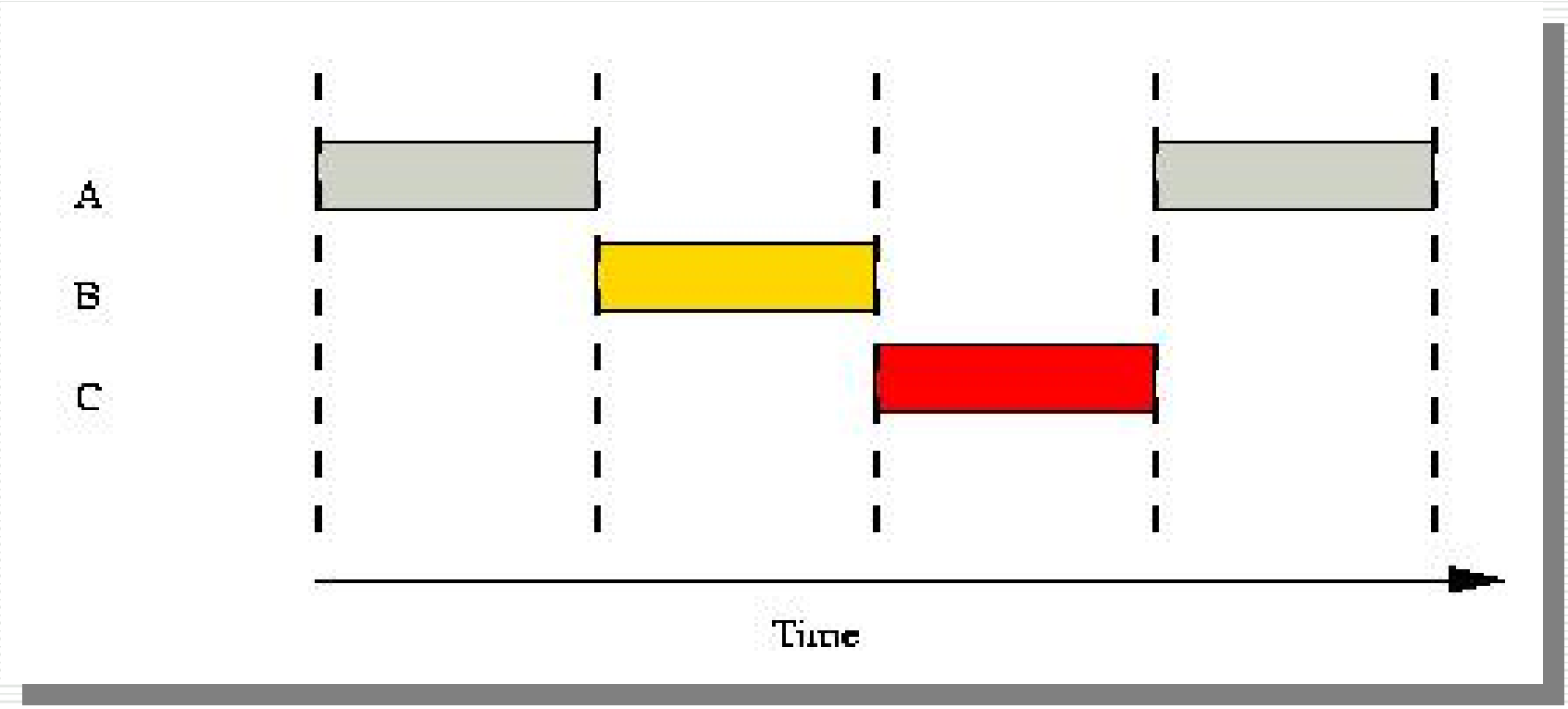
# The Runtime Environment (8)

Priority scheduling (Real-Time)



# The Runtime Environment (9)

Round robin scheduling:



# The Runtime Environment (10)

*TINIOS distinguishes between processes and threads:*

- **A process is expensive in terms of processor work and time. Also need support from OS for IPC**
- **A thread (or lightweight process) is a sub-process and needs much less code and time. Also since all variables in same process easy to access from separate threads**
- **Time slices: 8ms for processes, 2ms for threads, 4ms for kernel processes (devices)**

# The Runtime Environment (11)

*Memory manager:*

- **Allocates memory from heap for all processes**
- **Automatic garbage collection**
- **File system**

# The Runtime Environment (12)

*The Garbage Collector:*

- **Is the only non-Java process, and runs in background**
- **Is invoked**

  ***(i) Explicitely using***
  ***`java.lang.System.gc()`***

  **(ii) When heap space drops below 64 kbytes**

  **(iii) When a process terminates**

# The Runtime Environment (13)

*The File System:*

- **The file system is situated in SRAM on heap (not flash - TINI1)**
- **Consists of linked lists of 512 byte blocks**
- **Files must be made contiguous in order to be interpreted by JVM**
- **Is non-volatile (battery backup)**

# The Runtime Environment (14)

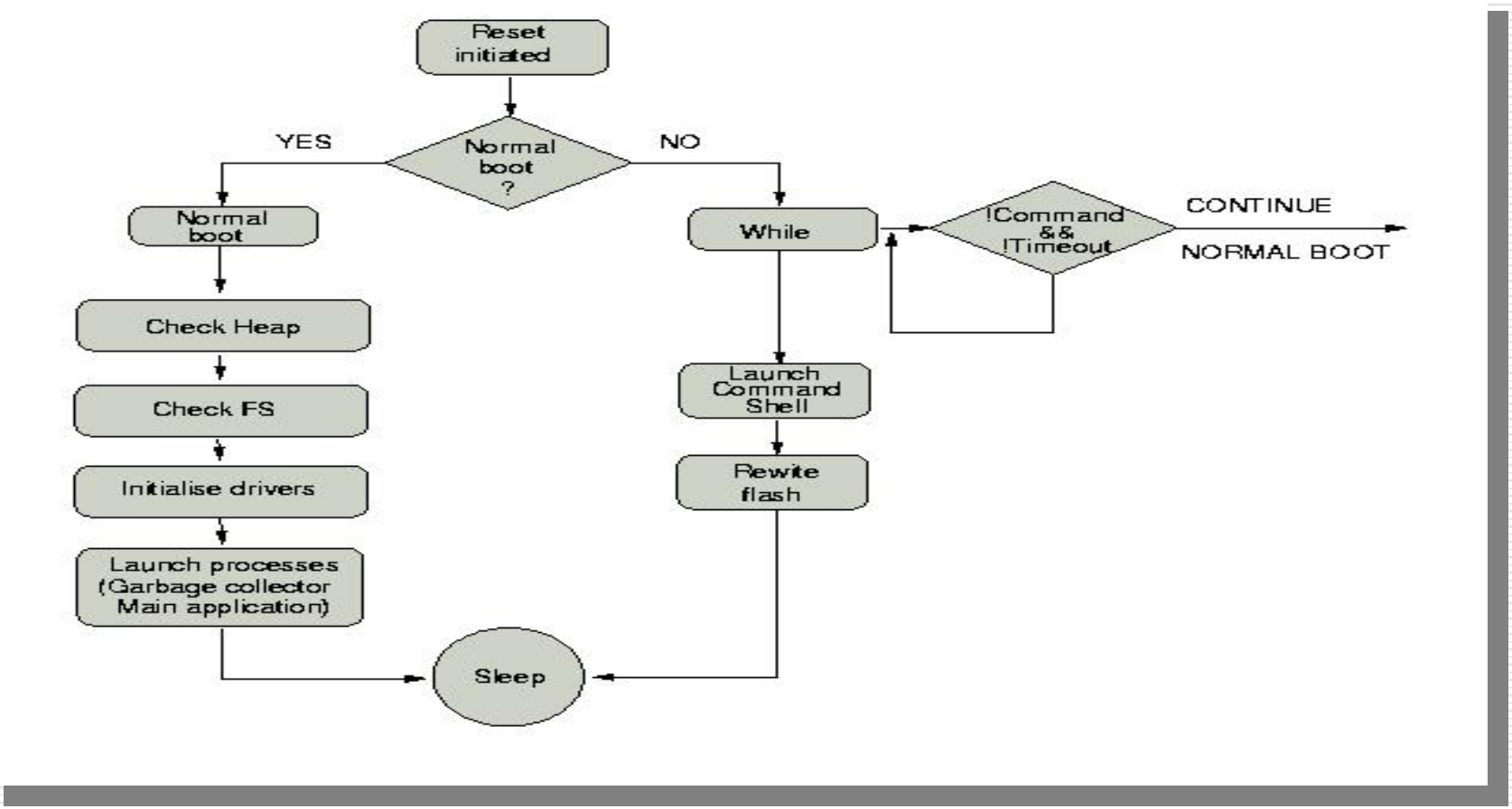
*The Input/Output system manager:*

- **TCP/IP stack (all networking)**

  ***(In ROM on DSTINIm400)***

- **Non-networking I/O (Serial, CAN, parallel bus etc)**

# The TINI Boot Sequence



# Using TINI

*Host system requirements:*

***Hardware:***

- ***Ethernet adaptor, RS232 port***

***Software:***

- ***Java Development Environment***
- ***Java Communications API***
- ***TINI Software Development Kit***

***OS:***

- ***Linux, Solaris, Windows***

# Using TINI (2)

*TINI SDK:*

- **Download from** www.ibutton.com/TINI
- **tini.jar** — JavaKit, TINIConvertor and BuildDependency
- **tiniclasses.jar** — TINI API class files
- **tini.tbin/tini400.tbin** — TINI Runtime binary image
- **slush.tbin/ slush400.tbin** — Binary image of shell

# Using TINI for the First Time

*Loading the Runtime Environment:*

- **Connect straight RS232 cable to TINI and apply power**
- **Run JavaKit [-flash 40 -400] on PC and connect to TINI**
- **Download tini.tbin and slush.tbin [tini400.tbin / slush400.tbin]**
- **Type b18 followed by f0 [b0 f0]**
- **Type EXIT**

# Using TINI (3)

*Slush is a command shell for interacting with TINIOS:*

- **Login with root and tini**
- **Use Unix like commands to view file system**

  **cat, ls, cd, cp etc**
- **Get help by typing**

  ***help [topic]***

# Using TINI (4)

*Configuring the Ethernet adapter for IPv4:*

***`ipconfig`***

- ***-a xx.xx.xx.xx*** ***Set ip address***
- ***-m xx.xx.xx.xx*** ***Set subnet mask***

***or***

- ***-d*** ***Get DHCP to issue ip address and mask***

***On 'OK to proceed?' prompt type 'y'***

***Test with ping, and telnet***

# Programming TINI

## Running a programme on TINI:

- Create source file on host using text editor
- Run `javac Progname.java`
- Run TINIConvertor to create a binary image

```
java [TINI_PATH]/tini.jar TINIConvertor –f
   Progname.class –d [TINI_PATH]/tini.db –o
   Progname.tini
```

- Download image to TINI via ftp put in binary mode
- Telnet to TINI and run

```
java Progname.tini
```

# Programming TINI
# Use of BuildDependency (1)

**Converting OneWireContainers + all classes NOT in flash:**

- **Because of complicated dependencies, use of TINConvertor becomes difficult**

```
java -classpath [TINI PATH]/bin/tini.jar
   BuildDependency
   -x  [TINI PATH]/owapi_dep.txt
   -p [TINI PATH]/owapi_dependencies_TINI.jar
   -f input_file.class
   -o output_file.tini
   -d {TINI_PATH]/tini.db
   -add OneWireContainer01;OneWireContainer02;...
```

- **For full list of options type:**

```
java [TINI PATH]/tini.jar BuildDependency
```

# Programming TINI
# Use of BuildDependency (2)

*Packages needing -add flag [m400]:*

- *All 1-Wire containers:* **OneWireContainer01**
  **etc [ALL]**
- *URLs:* *HTTP, FTP, FILE, MAILTO*
- *HTTP server:* *HTTP SERVER*
- *FTP clients:* *FTPCLIENT*
- *Communications:* *IIC, CAN, PPP, SPI*

# The Native Packages

**Some com.dalsemi packages:**

| | |
|---|---|
| `com.dalsemi.comm` | *CAN and serial ports* |
| `com.dalsemi.fs` | *Extensions to* `java.io.File` |
| `com.dalsemi.onewire` | *1-Wire classes* |
| `com.dalsemi.shell` | *FTP and telnet shells* |
| `com.dalsemi.system` | *Native hardware access* |
| `com.dalsemi.tininet` | *Network hardware support* |

# Serial IO (1)

*Using* `javax.comm`:

**Obtain a serial port using:**

```
static CommPortIdentifier
    getPortIdentifier(String portName)
```

**where**

- ***`serial0` is serial port,***
- **`serial1` is 1-wire port and,**
- **`serial4` is 2nd serial port [m400 only]**

**and then open it using the `CommPortIdentifier` method**

```
CommPort open(String appname, int timeout)
```

**Here `timeout` is the time (in ms) to wait before giving up**

**Note that the `CommPort` object must be cast to a `SerialPort` object before it can be used**

# Serial IO (2)

*Configure the port :*

```
void SetSerialPortParams(
   int baudrate, int databits,
   int stopbits, int parity)
```

Can configure to (7, 2, 0), (7,1,1), (8,1,0), (8,1,1)

*(data, stop, parity)*

Set handshaking

```
int getFlowControlMode()
void setFlowControl(int flowcontrol)
```

# Serial IO (3)

Next obtain the streams associated with the port:

```
InputStream getInputStream()

OutputStream getOutputStream()
```

and

```
byte read(void)
void write(byte b)
```

# Serial IO (4): Code snippet

```
// Step 1: Obtain the port object
CommPortIdentifier cpi =
    CommPortIdentifier.getPortIdentifier(“serial0”);

// Step 2: Open the port object and set timeout
SerialPort sPort = (SerialPort)cpi.open(“MyApp”, 5000);

// Step 3: Set serial port parameters
sPort.setSerialPortParams(9600, 8, 1, 0);

// Step 4: Get streams
InputStream  sIn  = sPort.getInputStream();
OutputStream sOut = sPort.getOutputStream();

// Step 5: Use the streams
while (true) {
    byte b = (byte)sIn.read();
    sOut.write(b);
}
```

# Networking (1)

Can perform networking on Ethernet using following packages:

| | |
|---|---|
| ***com.dalsemi.tininet.http*** | ***Simple http server*** |
| ***com.dalsemi.tininet.icmp*** | ***Error and control*** |
| ***com.dalsemi.tininet.dhcp*** | ***Dynamic Host Configuration protocol*** |
| ***com.dalsemi.tininet.dns*** | ***Domain Name System*** |

# Networking (2)

*Communicating with sockets:*

- **Standard java.net package**
- **Create a new socket with constructor**

```
Socket(String serverIP, int port)
```

- **Get streams**

```
InputStream getInputStream()
OutputStream getOutputStream()
```

- **Use streams**

```
int read(byte[] buffer, int offset, int length)
void write(byte[] buffer, int offset, int length)
```

# Networking (3): Code snippet

```
Socket client;
byte buffer[1024];

// Step 1: Connect to server
client = new Socket(sServerIP, port);

// Step 2: Set up IO streams as needed
InputStream  sIn  = client.getInputStream();
OutputStream sOut = client.getOutputStream();

// Step 3: Process data
while (true) {
    // Read from the input stream
    int nchars = sIn.read(buffer, 0, buffer.length);

    // … and echo back
    sOut.write(buffer, 0, nchars);
}
```

# Networking (4)

*A simple HTTP server:*

```
import com.dalsemi.tininet.http.HTTPServer;

// Step 1: Construct a server object on a given port
HTTPServer server = new HTTPServer(80);

// Step 2: Configure the server index file and root directory
server.setIndexPage("index.html");
server.setHTTPRoot("/html");

// Step 3: Handle service requests
while (true) {
    server.serviceRequests();
}
```

# Networking (5)

*index.html:*

```
<html>
   <head><title> Hello Trieste </title>
      <body>
         Hello Trieste
      </body>
   </head>
</html>
```

# 1-Wire Devices

- **1-Wire devices have single active line (+return) for signaling**
- **Each 1-Wire device has factory defined unique address with last byte giving family of device (memory, thermometer, adc etc) i.e.:**
  - ***DS18B20 digital thermometer has id of 0x28***
  - ***DS2505 memory has id 0x0b***

**TINI has two 1-Wire ports**

- ***Internal : Used for ethernet MAC address only***
- ***External: Used for everything else***
  - Connected to *serial1*
  - ***Can drive many devices***

- ***All classes needed for 1-Wire networking found in***

  ***com.dalsemi.onewire***

# 1-Wire devices (2)

*Adapters:*

*`DSPortAdapter`* is superclass with `TINIExternalAdapter` used for specific properties

**Instance of adapter created by:**

***(i) `new TINIExternalAdapter()`***

or calling

(ii) `DSPortAdapter getDefaultAdapter()`

from `OneWireAccessProvider` class

Note: 1-Wire devices have two speeds: Regular and Overdrive.

Set with

`DSPortAdapter.setSpeed()`

# 1-Wire devices (3)

- In a multi-threaded environment will need to lock and unlock the adapter:

  ***boolean beginExclusive()***

  ***endExclusive()***

- Before use it is necessary to determine what devices are on network using

  ***boolean findFirstDevice()***

  ***and in loop***

  ***boolean findNextDevice()***

# 1-Wire devices (4)

- **At each stage of the device identification can get address of devices in various forms:**

  ***`String getAddressAsString()`***

  ***`long getAddressAsLong()`***

  ***etc***

- **Can often use this to identify device unambiguously using family id**

# 1-Wire devices (5)

**Can communicate through:**

```
boolean getBit()
int getByte()

byte[] getBlock(int length)
```

**and**

```
void putBit(boolean bit)

putByte(int value

void dataBlock(byte[] data, int offset,
               int length)
```

**All these throw `OneWireException`, and `OneWireIOException`**

# 1-Wire devices (6)

```
import com.dalsemi.onewire.*;

static final READ_MEMORY   = 0xf0;
static final DS2502_FAMILY = 0x89;
DSPortAdapter adapter;
byte memory[] = new byte[128];

// Step 1: obtain an adapter
adapter =
 OneWireAccessProvider.
    getDefaultAdapter();

// Step 2: Get exclusive access to
    the adapter
adapter.beginExclusive(true);

// Set speed of adapter (*******)
adapter.setSpeed(
    adapter.SPEED_REGULAR);
```

```
// Step 3: iterate through the devices on
    the network

if (adapter.findFirstDevice()) {
    // Obtain the device address and
    // check that it is in the
    // correct family. DS2502 have a
    // family id of 0x89
    long address =
        adapter.getAddressAsLong();
    if (address & 0x0ff == DS2502_FAMILY)
        getMemory();
    else while (adapter.findNextDevice()) {
        if (address & 0x0ff ==
                DS2502_FAMILY) {
            getMemory();
            break;
        }
    }
}
// Close the lock on the adapter
adapter.endExclusive();
```

# 1-Wire devices (7)

```
// Read a block of memory from DS2502
// Method to get the 1024 bits of the DS2502 memory
void getMemory(void) {
    byte [] command;

    // Form the command to send to the device
    command = new byte[3];
    command[0] = READ_MEMORY;
    command[1] = 0;
    command[2] = 0;

    // Step 4: Send the command
    adapter.dataBlock(command, 0, 3);
    // Get the response
    memory = adapter.getBlock();
}
```

# 1-Wire devices (7)

- Containers through `OneWireContainer` superclass provide a much more convenient way of accessing devices

- Each container formed by `OneWireContainerXX` where `XX` is family id

# 1-Wire devices (8)

Containers exist for:

| | |
|---|---|
| ADCContainer | Analog measuring operations. |
| ClockContainer | Real-Time clocks. |
| HumidityContainer | Humidity measuring operations. |
| MemoryBank | Basic memory communication. |
| MissionContainer | Analog measuring operations. |
| OneWireSensor | Basic sensor operations. |
| OTPMemoryBank | OTP Memory bank interface. |
| PagedMemoryBank | Paged Memory bank interface. |
| PagedMemoryBank | Password protection interface. |
| PotentiometerContainer | Basic potentiometer operations. |
| SwitchContainer | Switch device interface |
| TemperatureContainer | Temperature measuring devices |

# 1-Wire devices (9)

**Accessed using:**

```
OneWireContainer getFirstDeviceContainer()
getNextDeviceContainer()
Enumeration getAllDeviceContainers()
```

- Throw `OneWireIOException, OneWireException`
- Once a container has been obtained the functionality offered by the device can be accessed
- Each family has different physical properties reflecting in specific methods for its container class i.e.

  *DS18B20 is a digital thermometer with a family id of 0x28. It has methods:*

  ```
  void doTemperatureConvert(byte[] state)
  double getTemperature(byte[] state)
  double getMaxTemperature()
  ```

  *etc.*

# 1-Wire devices (10)

```
import com.dalsemi.onewire.container.OneWireContainer;

DSPortAdapter adapter;

// Get adapter etc as before

 …
// Obtain containers
container = adapter.getFirstDeviceContainer();
while (container != null) {
    // Do something with it
    System.out.println("Got " + container.getName());
    …

    // Get next container
    container = adapter.getNextContainer();
}
```

# Summary

- The TINI is a versatile device for of data all forms of embedded data acquisition and control
- It is easy to programme using standard java and html
- It is reasonably cheap and many different boards can be found
- The 1-Wire network allows simple, modular data acquisition