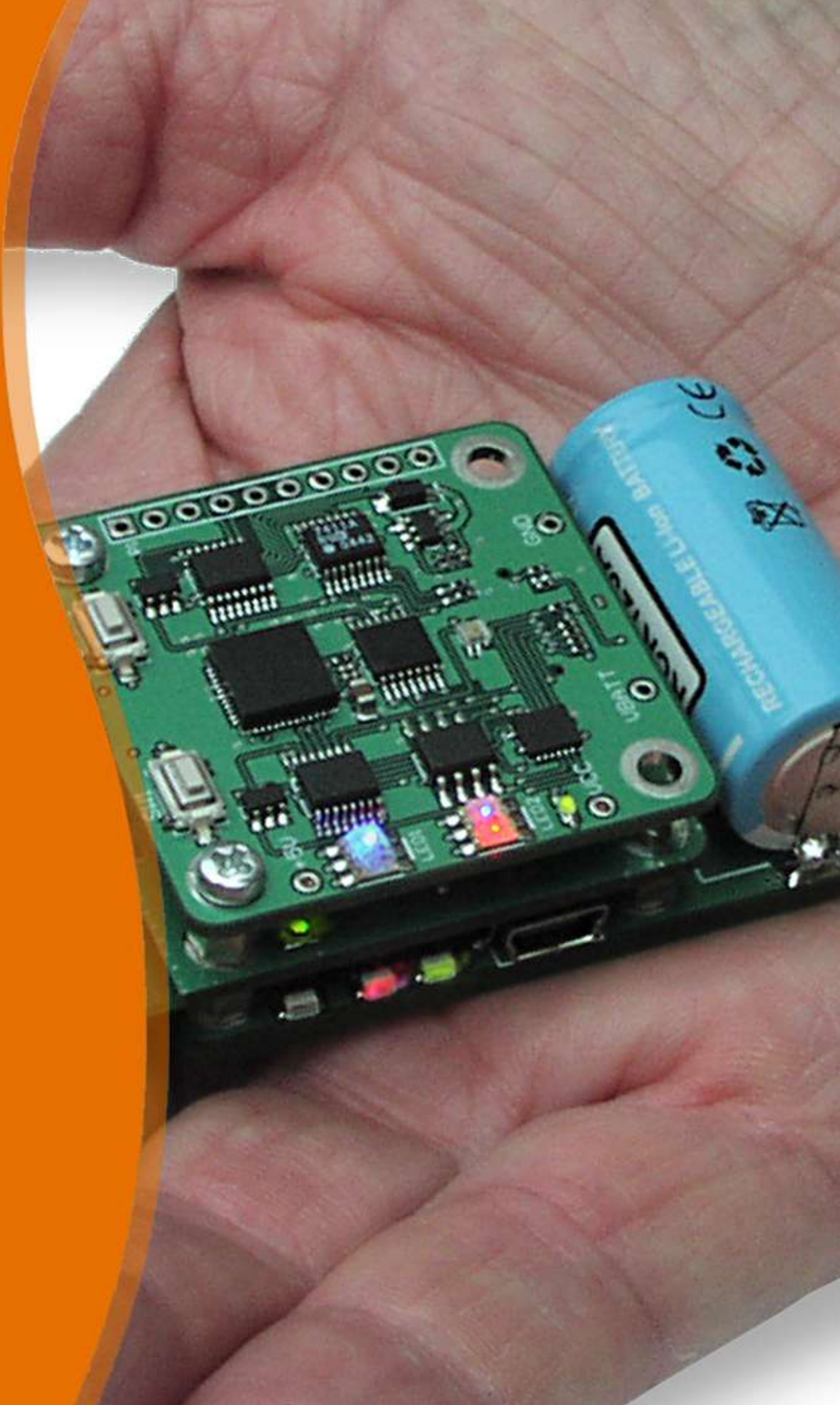
JAVA™ ON WIRELESS SENSORS

Cristina Cifuentes

Sun Labs

February 2006



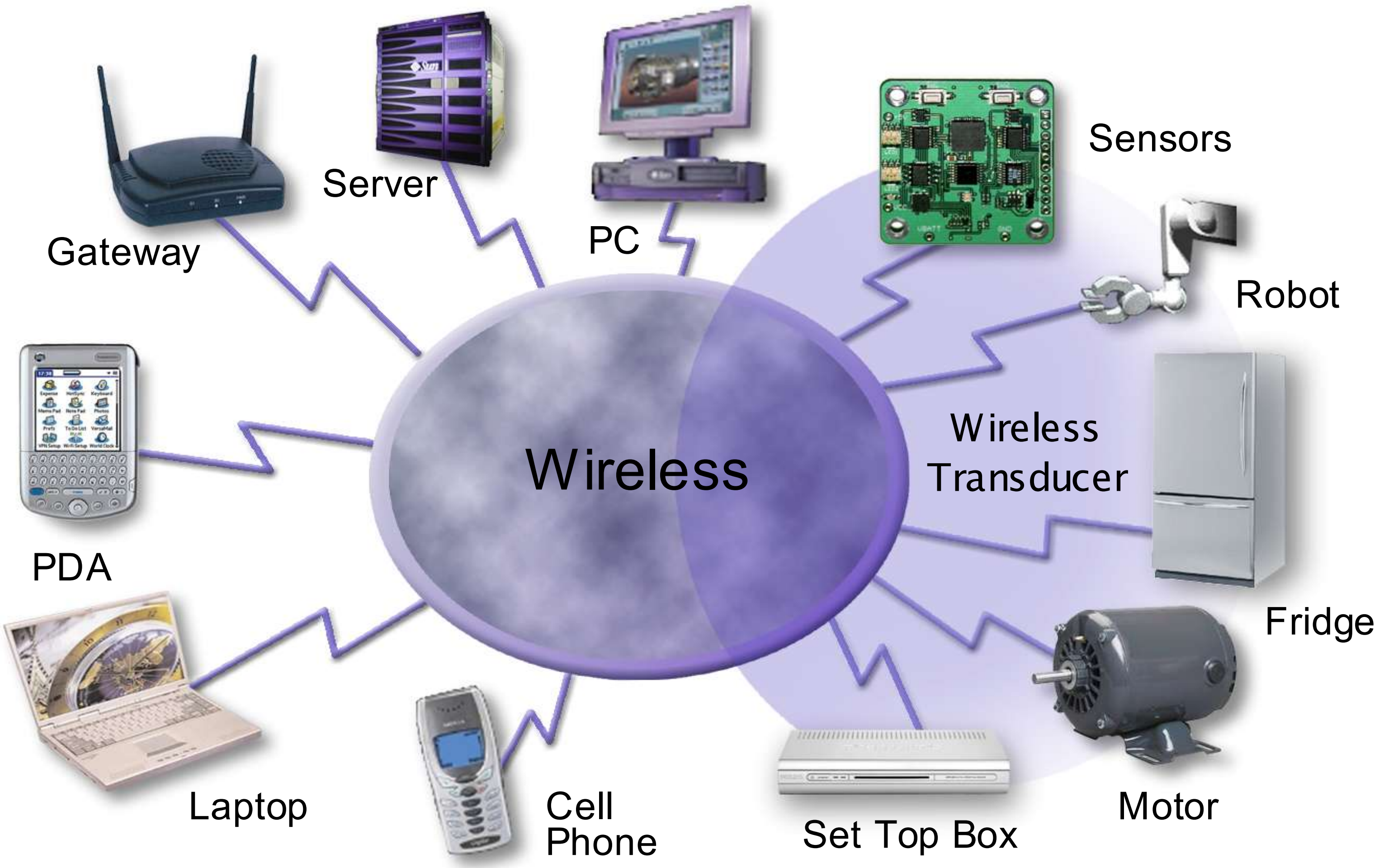
Agenda

- Wireless Sensor Networks
- Proposed Solution and Demo
- The Squawk Java^{TM} Virtual Machine
- The Sun^{TM} Small Programmable Object Technology (SPOT) System
- The Wireless API
- Results

Agenda

- Wireless Sensor Networks
- Proposed Solution and Demo
- The Squawk Java™ Virtual Machine
- The Sun™ Small Programmable Object Technology (SPOT) System
- The Wireless API
- Results

Wireless Networks



The State of the Art

- Ideas of “Smart Dust”
- Berkeley motes, TinyOS, IEEE 802.15.4
- Sun Labs Anteater project
  - Research into impact and meaning of such systems to Sun Microsystems
  - Major customer advantage seen as economics and flexibility
- Most of the work is aimed at infrastructure issues
  - Size, power, and networking (mesh networking)

Applications: Chicken and Egg

- Hard to develop applications using current technologies
  - > Low-level C-like languages
  - > Unproductive development tools
    - Hardly any debugging support
  - > Too many low-level concerns in current systems
    - Most high-level software developers do not know how hardware works, or even have an appreciation any more
  - > Not accessible to majority of software developers

Agenda

- Wireless Sensor Networks
- Proposed Solution and Demo
- The Squawk Java^{TM} Virtual Machine
- The Sun^{TM} Small Programmable Object Technology (SPOT) System
- The Wireless API
- Results

Proposed Solution

Squawk Java VM

CLDC libraries

Libraries for driving hardware: radio, sensor board

Network libraries: 802.15.4, GCF implementation

Desktop libraries: connect from J2SE VMs to wireless devices

Hardware: 32-bit ARM core, Chipcon CC2420 based wireless platform, SPI based peripherals

From conception to implementation: 6 months

Demonstration

- Reactomatic
- Ectoplasm
- Theremin

Sample Code: React-o-matic

```
public static void main(String[] args) throws IOException {
        // Setup and read from accelerometer
        Accelerometer3D acc = DemoSensorBoard.getAccelerometer();
        ((LIS3L02AQAccelerometer)acc).set6GScale();
        RangeInput x = acc.getX(), y = acc.getY(), z = acc.getZ();


        SensorBoardColouredLED led = SensorBoardColouredLED.getLed1();
        led.setOn();          //switch it on
        led.setRGB(0,0,0);    //...but black it out


        // Display red/green/blue on LED based on motion difference
        int lastX = 0, lastY = 0, lastZ = 0;
        while(true) {
                int xValue = x.getValue(), yValue = y.getValue(), zValue = z.getValue();

                int r = Math.abs(xValue-lastX) > 10 ? 255:0;
                int g = Math.abs(yValue-lastY) > 10 ? 255:0;
                int b = Math.abs(zValue-lastZ) > 10 ? 255:0;
                led.setRGB(r,g,b);

                lastX = xValue; lastY = yValue; lastZ = zValue;

        }
 }
```

The Sun SPOT System

- Hardware
  - 32-bit ARM core
  - Chipcon CC2420 based wireless platform
  - SPI based peripherals
  - Simple demo sensor board

- Software
  - Squawk: Java VM
  - Desktop build and deploy scripts
  - Libraries for
    - Driving hardware: radio, sensor boards, ...
    - Basic 802.15.4 network functionality
  - SpotWorld: graphical desktop interface

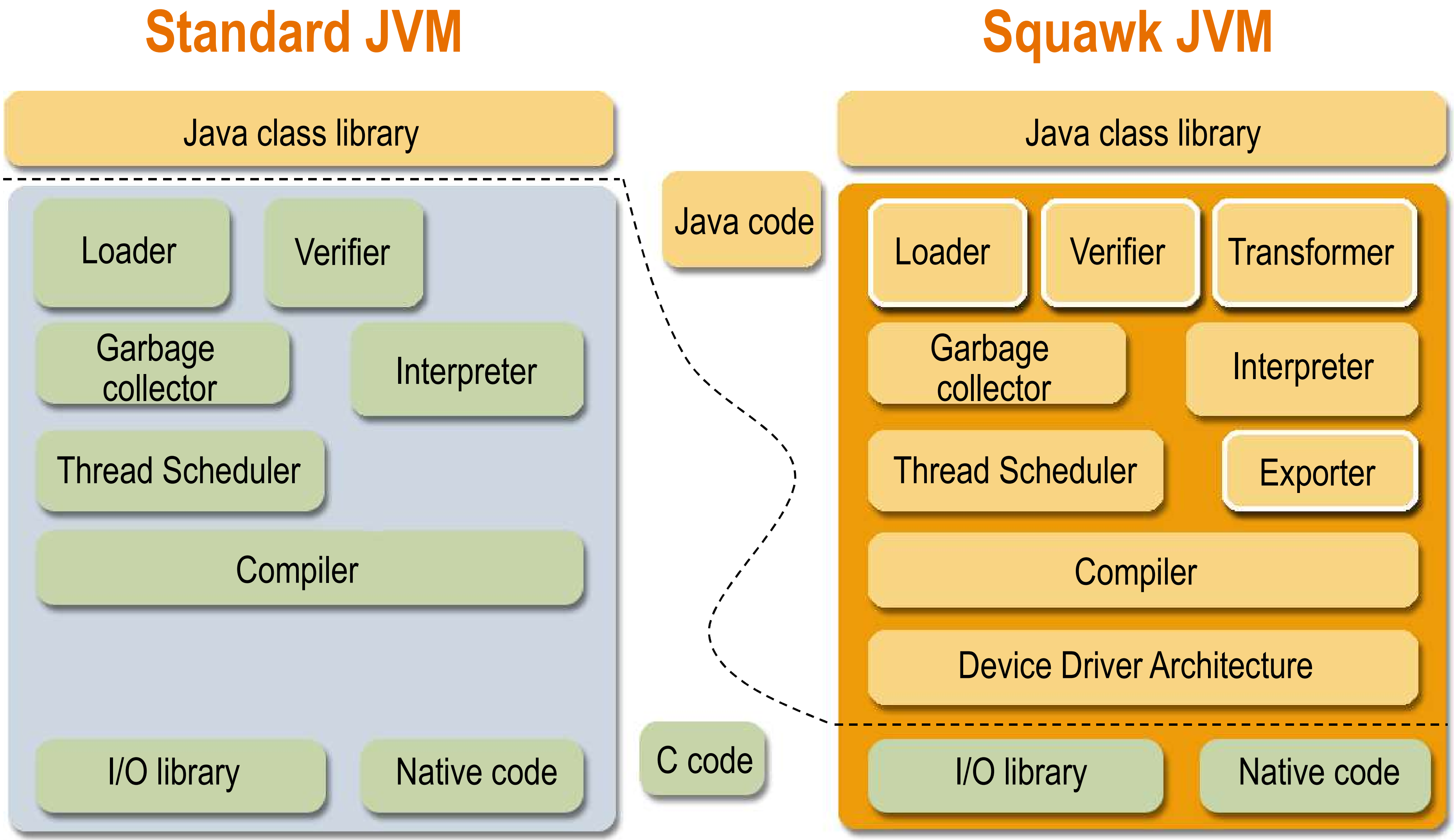
Agenda

- Wireless Sensor Networks
- Proposed Solution and Demo
- The Squawk Java™ Virtual Machine
- The Sun™ Small Programmable Object Technology (SPOT) System
- The Wireless API
- Results

Origins of the Squawk JVM Project

- Prior experience with Spotless and KVM
  - C-based
- Q: Is there an alternative way to construct JVMs?
- A: Write it in Java
  - Pointer safety, exceptions, garbage collection, ...
  - More portable
  - Ease of development
- Design for memory constrained devices

Standard JVM vs Squawk JVM



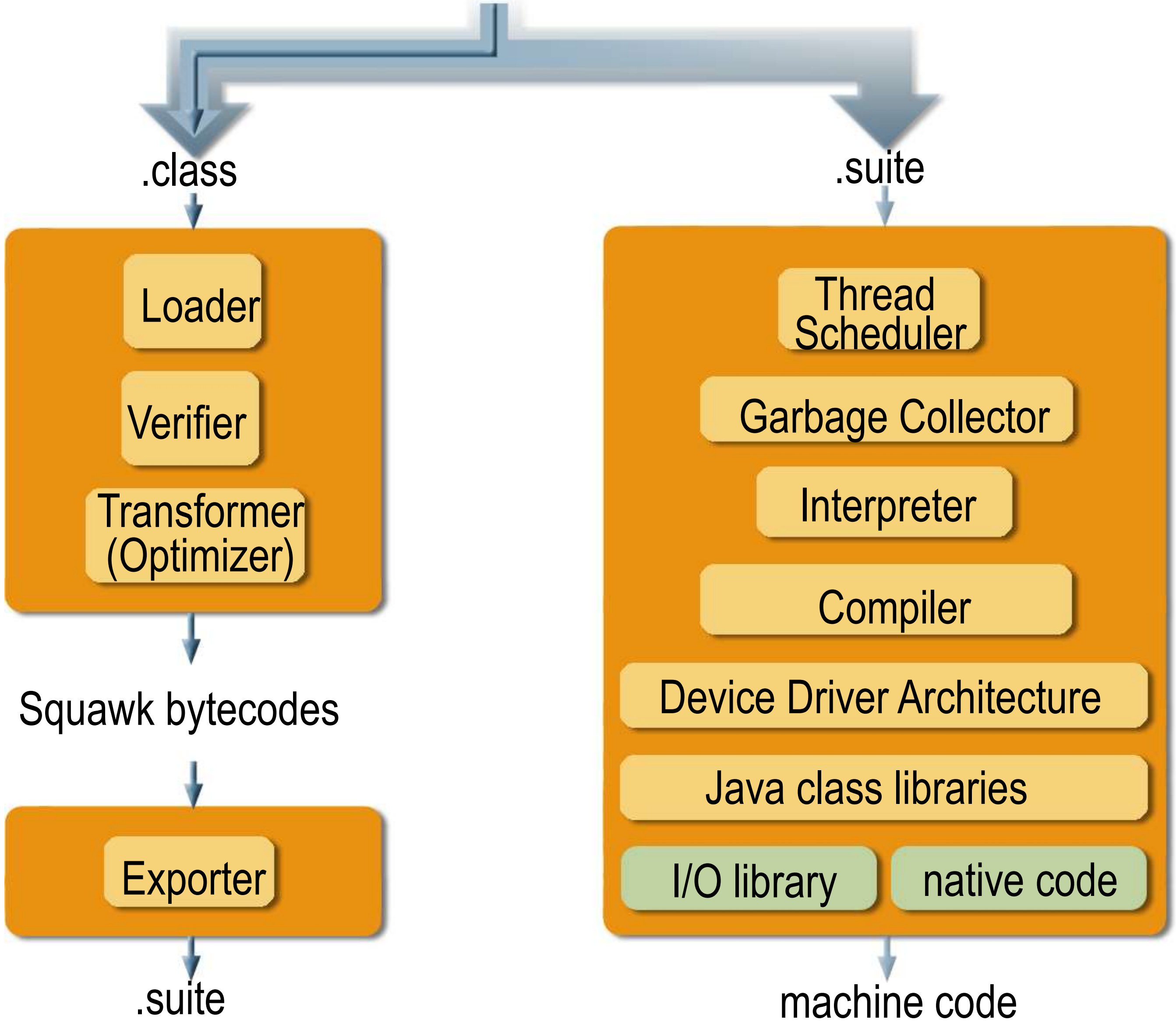
The Squawk JVM

- Java VM mainly written in Java
  - Interpreter written in C
  - Garbage collector translated from Java to C
- J2ME level VM
  - CLDC 1.0/1.1 libraries
- Extra features
  - Runs on the bare ARM without an underlying OS
  - Interrupts and device drivers written in Java
  - Support application migration (extension to isolates),
- Memory footprint on the ARM:
  - 80K RAM for VM
  - 270K flash for libraries

Squawk Features for Wireless Sensor Devices

- Designed for memory constrained devices
- Runs on the bare metal on the ARM
- Represents applications as objects
- Runs multiple applications in the one VM
- Migrates applications from one device to another
- Authenticates deployed applications on device

Squawk's Split VM Architecture



Squawk Features for Wireless Sensor Devices

- Designed for memory constrained devices
- Runs on the bare metal on the ARM
- Represents applications as objects
- Runs multiple applications in the one VM
- Migrates applications from one device to another
- Authenticates deployed applications on device

Squawk bytecodes: Designed for Memory Constrained Devices

Squawk Bytecode Property	Benefit
Commonly used bytecodes are 2 bytes instead of 3 bytes	More compact
References to fields and methods resolve into physical offsets	More efficient for interpretation
Local variables are typed	More efficient for compilation
One OOP map per method, nothing on the operand stack at GC points	Simplifies garbage collection Eliminates need for static interpretation to decipher activation frames

Suite Files

- Preprocessed set of classfiles
- Internally fully linked
  - Pointers to other classes in suite or parent(s) only
  - Chain of suites is a transitive class closure
- Uses Squawk bytecode set

Classfiles vs Suite Files Size Comparison

Application	JAR	Suite	Suite/JAR
CLDC	458,291	149,542	0.33
cubes	38,904	16,687	0.42
hanoi	1,805	835	0.46
delta blue	30,623	8,144	0.27
mpeg	100,917	54,888	0.54
manyballs	12,017	6,100	0.51
pong	17,993	7,567	0.42
spaceinvaders	50,854	25,953	0.51
tilepuzzle	18.516	7,438	0.40
wormgame	23,985	9,131	0.38
Total	**753,905**	**286,285**	**0.38**

Squawk Features for Wireless Sensor Devices

- Designed for memory constrained devices
- Runs on the bare metal on the ARM
- Represents applications as objects
- Runs multiple applications in the one VM
- Migrates applications from one device to another
- Authenticates deployed applications on device

Interrupt Handling and Device Driver Support

- Device driver sets up the interrupt controller
- Interrupt handler thread blocks waiting for the VM to signal an event
- When an interrupt occurs, an assembler interrupt handler sets a bit in an interrupt status word and disables the interrupt
- At each VM reschedule point, the bit is detected, the event signaled, the scheduler resumes the interrupt handler thread, which handles the interrupt and re-enables it

- Device driver written in Java

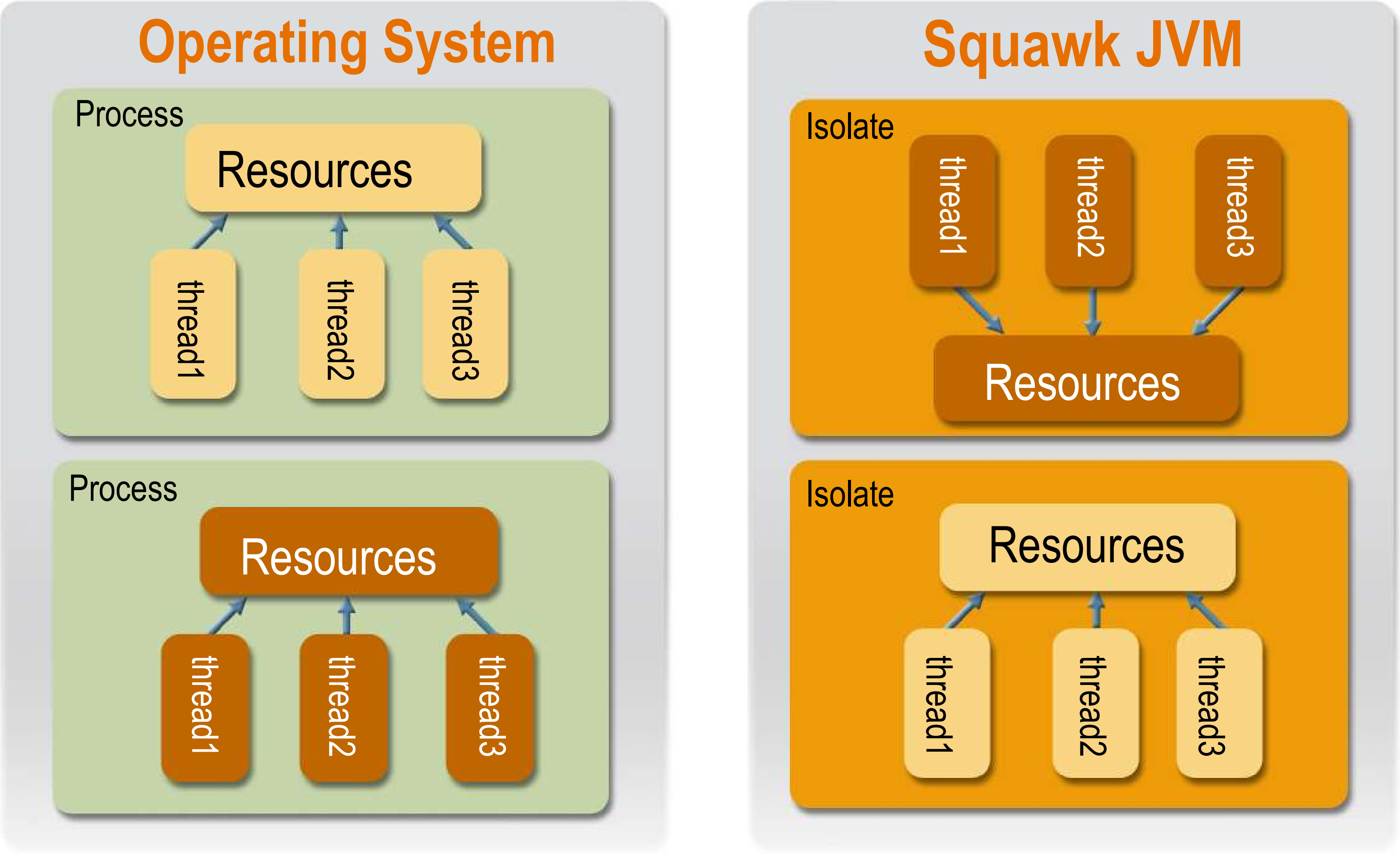
Interrupt Latency

- Dependent on the time from the global interrupt handler running until the next VM schedule
- Optimal case
  - VM is idle => no penalty
- Average case
  - VM is executing bytecodes in another thread => VM reschedules after a certain number of back branches
- Worst case
  - VM is executing a GC => VM reschedules after the GC completes; in practice, < 1 msec for heap size of 8MB

Squawk Features for Wireless Sensor Devices

- Designed for memory constrained devices
- Runs on the bare metal on the ARM
- Represents applications as objects
- Runs multiple applications in the one VM
- Migrates applications from one device to another
- Authenticates deployed applications on device

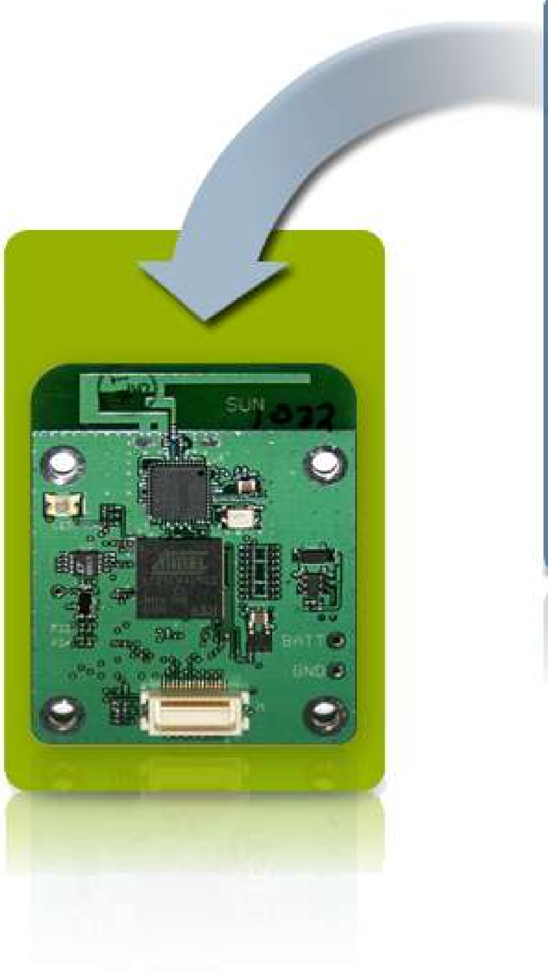
OS Processes and Isolates Analogy



Squawk Isolates

- Each application is represented by an Isolate object
- Similar to JSR 121 Isolate API
  - Each isolate has resources that are shared amongst the threads of that isolate
  - Immutable state (e.g., methods, string constants, parts of classes) is shared
    - Non-shared class state includes static fields, class initialization state, and class monitors
  - Different support for inter-isolate communication
    - Uses generic connection framework
- Allows for reification of applications
  - Can start(), pause(), resume(), and exit()

Sample Isolate Code



```
public void run() throws Exception {
   ...
   Isolate isolate = new Isolate
     ("com.sun.spots.SelfHibernator",
      url());
   isolate.start();
   send(isolate, outStream);
   ...
}
```

Uses of Isolate Migration

- Load balancing
  - Radio, power, performance, space
- Simplifies maintenance
  - Field replacement of hardware
- Debugging on the go
  - Local debugging of remote application
- Seamless client-server programming model

```
localVar = getLocalData( );
goto (server);
localVar.storeOn (file);
```

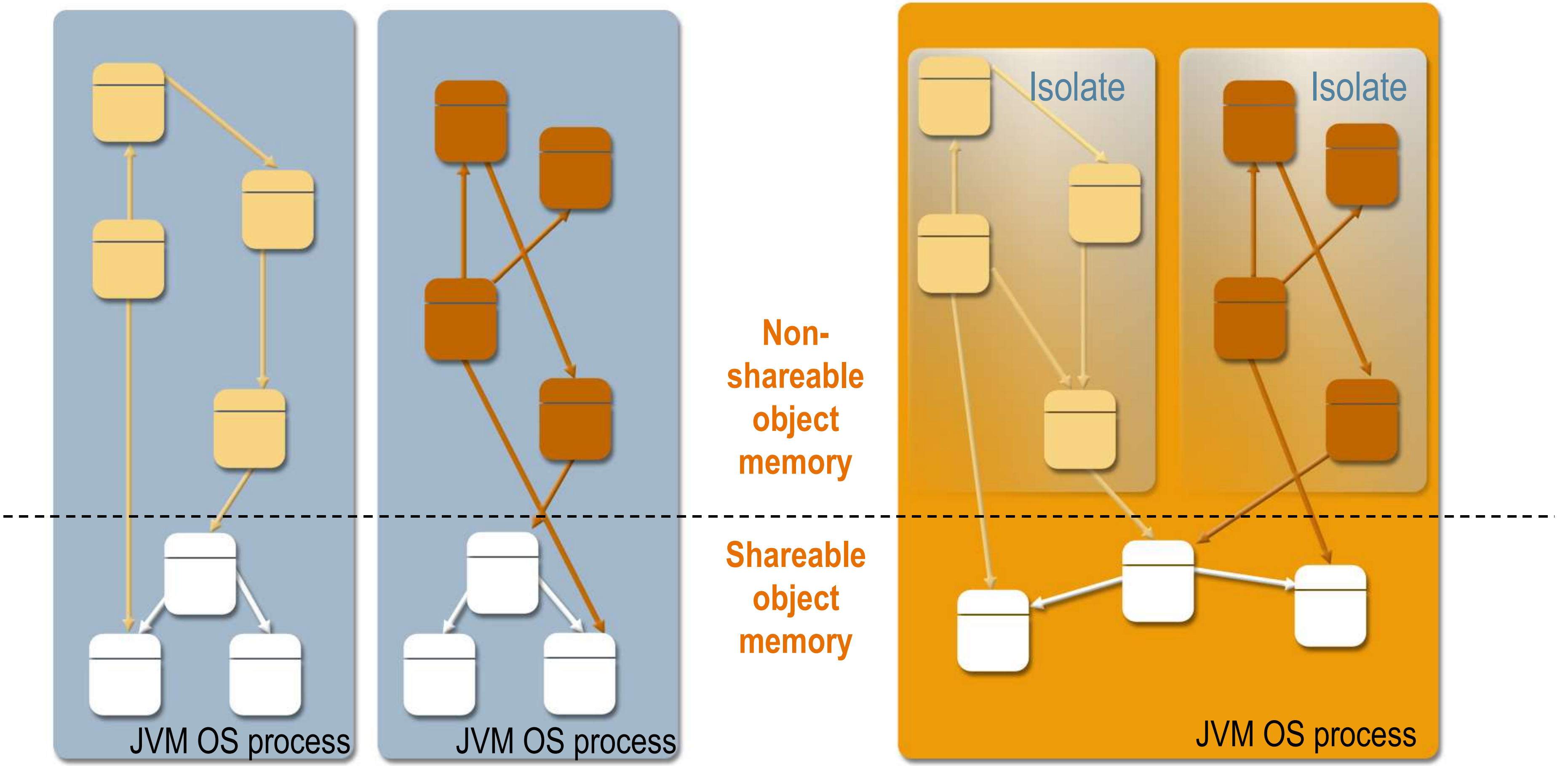
Squawk Features for Wireless Sensor Devices

- Designed for memory constrained devices
- Runs on the bare metal on the ARM
- Represents applications as objects
- Runs multiple applications in the one VM
- Migrates applications from one device to another
- Authenticates deployed applications on device

Multiple Isolates (Applications) on the One JVM

Standard JVM

Squawk JVM



Isolate

Isolate

Non-shareable object memory

Shareable object memory

JVM OS process

JVM OS process

JVM OS process

Squawk Features for Wireless Sensor Devices

- Designed for memory constrained devices
- Runs on the bare metal on the ARM
- Represents applications as objects
- Runs multiple applications in the one VM
- Migrates applications from one device to another
- Authenticates deployed applications on device

Isolate (Application) Migration

- Isolates can be migrated
  - > Migrates state of a running application, to continue running on a target device
  - > Target device must have binary code (suite) of the application
  - > Migration uses same object serialization mechanism as the suite creator
  - > Constraints on external state
    - Must be none, or
    - Must be homogeneous at both ends (Sun SPOT Squawk), or
    - Must be serializable (desktop Squawk)

Squawk Features for Wireless Sensor Devices

- Designed for memory constrained devices
- Runs on the bare metal on the ARM
- Represents applications as objects
- Runs multiple applications in the one VM
- Migrates applications from one device to another
- Authenticates deployed applications on device

Secure Suite Deployment

- Digitally sign suites on the desktop and verify signature on the Sun SPOT at installation time
- Why?
  - > Ensure that split VM architecture does not compromise verified Java applications on the desktop
    - Suite originated at a trusted source
    - Suite was received in the state intended
- For a user, each device is bound to one or more Sun SPOT SDK installations

Secure Suite Deployment

- Each SDK has an associated public/private key pair
  - Key pair automatically generated on the background first time the SDK is installed
- Private key used for signing
  - Stored on desktop in password protected file
- Public key stored on device
- Signed suites can also be migrated from device to device

Agenda

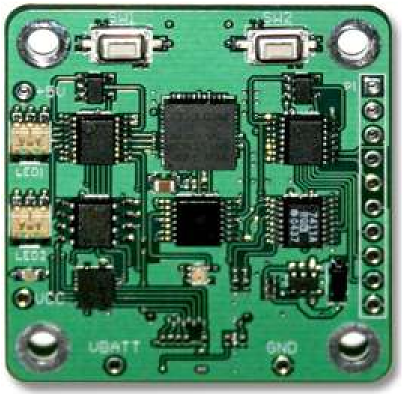
- Wireless Sensor Networks
- Proposed Solution and Demo
- The Squawk Java^{TM} Virtual Machine
- The Sun^{TM} Small Programmable Object Technology (SPOT) System
- The Wireless API
- Results

Sun SPOT Hardware

- ARM7 core
  - 256K RAM/2M ROM
- CC2420 radio
  - Strip antenna
- Single LED
- Double sided connector for stackable boards
- Can be powered from single 1.5V battery
  - Requires 25-90mA depending on operation
- 35x25 mm in size
- Supporting testboard with USB connection to desktop

Sun SPOT Demo Sensor Board

- Demo sensor board
  - > 3D accelerometer
  - > 9 I/O Pins (PWM capable)
  - > Temperature sensor
  - > Light sensor
  - > IRDA serial connection



- All SPI driven peripherals
- Users can build own transducer boards
  - > Experimental board available

Sun SPOT Build and Deploy Scripts

- Full range of developer tools
  - Use standard IDEs to create Java code
  - Build and deploy scripted to make as simple as possible
    - Ant based
  - JDWP debugger

- Deploy host applications via
  - Serial, USB, over-the-air (OTA)

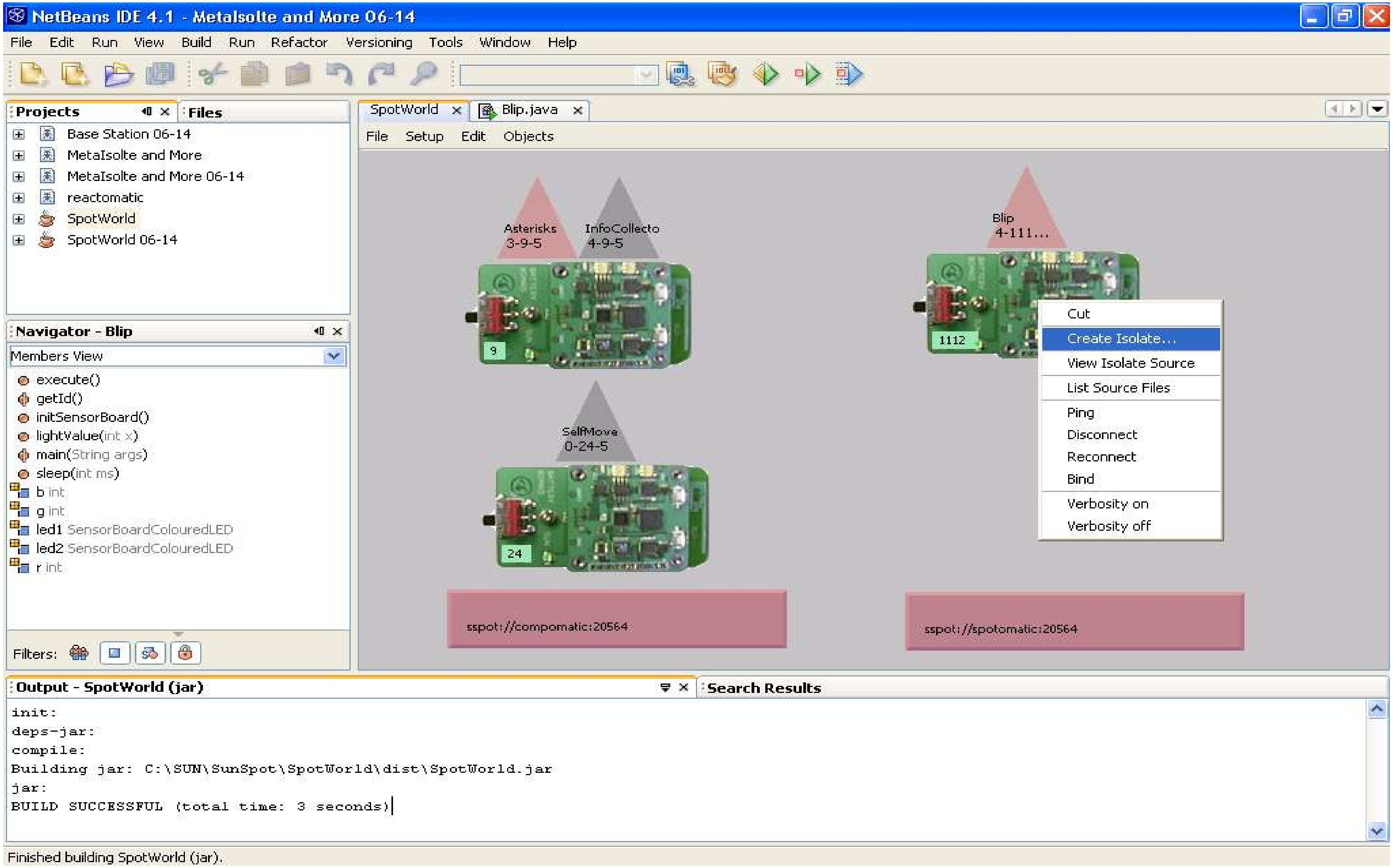
Sun SPOT Software Libraries

- Standard J2ME Java libraries
  - CLDC 1.0/1.1
- Hardware libraries
  - SPI, AIC, TC, PIO drivers all in Java
  - Sensor board hardware driven by Java (no C)
    - ADCs, GPIO, IRDA, etc.
- Radio libraries
  - Drive Chipcon CC2420 hardware from Java (no C)

Software Libraries (2)

- Network libraries
  - 802.15.4 MAC layer in Java (no C)
  - Simple GCF implementations of connections
- Desktop libraries
  - Create connections from standard J2SE VMs to wireless devices
  - Utilize one Sun SPOT as a gateway (base station)

Sun SPOT Graphical UI: SpotWorld



```
init:
deps-jar:
compile:
Building jar: C:\SUN\SunSpot\SpotWorld\dist\SpotWorld.jar
jar:
BUILD SUCCESSFUL (total time: 3 seconds)
```

Finished building SpotWorld (jar).

Agenda

- Wireless Sensor Networks
- Proposed Solution and Demo
- The Squawk Java^{TM} Virtual Machine
- The Sun^{TM} Small Programmable Object Technology (SPOT) System
- The Wireless API
- Results

The Wireless API Building Blocks

- IEEE 802.15.4
  - Low data rates (250 kbps, 40 kbps, and 20 kbps)
  - Multi-month to multi-year battery life
  - Low complexity
- Generic connection framework
  - Part of J2ME
  - Hierarchy of interfaces and classes that create connections (e.g., HTTP, datagram) and perform I/O
  - `javax.microedition.io` package

The Wireless API - Radio

```
radio://{address}:{port}
```

- Streaming radio connection
- `address`: unique IEEE address of the device
- `port`: channel to be used

Radio Example

- Output number 5 to device 1020 on channel 42

```
StreamConnection conn = (StreamConnection)
  Connector.open (“radio://” + 1020 + “:42”);
DataOutputStream output =
  conn.openDataOutputStream();
output.writeInt(5);
output.flush();
```

The Wireless API – Radiogram

```
radiogram://{address}|broadcast:{port}
```

- Datagram-style communication
- Point-to-point
- Can also broadcast to multiple listeners

Radiogram Example

- Send radiogram to device 1020 on channel 42 and wait for receiving a datagram from remote device

```
StreamConnection conn = (StreamConnection)
  Connector.open (“radiogram://” + 1020 + “:42”);
Datagram dg = conn.newDatagram
  (conn.getMaximumLength());
dg.writeUTF (“Hello world”);
conn.send (dg);
conn.receive (dg);
```

Radiogram Broadcast Example

- Listen for radiograms on port 51 from all neighbours and broadcast the received signal strength indicator (RSSI) to any listeners by broadcasting on port 52

```
public NeighbourhoodSender() throws IOException {
  listenerConn = (DatagramConnection)
     Connector.open (“radiogram://:51”);
  inputPacket = (Radiogram)
     listenerConn.newDatagram(0);
  senderConn = (DatagramConnection) Connector.open
     (“radiogram://broadcast:52”);
  outputPacket = senderConn.newDatagram
     (senderConn.getMaximumLength());
}
```

Agenda

- Wireless Sensor Networks
- Proposed Solution and Demo
- The Squawk Java^{TM} Virtual Machine
- The Sun^{TM} Small Programmable Object Technology (SPOT) System
- The Wireless API
- Results

Experimental Results

Benchmark	.class	.suite	Sampling	(samples/sec)
Richards (Gibbons)	11,770	4,584	ARM PIO lines	11,760
Richards (Deutsch)	19,655	6,788	Sensor board input lines	300-800
Delta Blue	27,520	9,724		
Game of Life	7,390	3,396	Radio range:	90 mts

Benchmark	LOC	ms on ARM7 EB40 board
Richards (Gibbons)	410	5,277
Richards (Deutsch)	456	8,382
Delta Blue	984	4,766
Game of Life	354	4,032

Conclusions

- Java on “wireless sensor networks” is here
  - > Small Java-based VM
    - Java runs on the bare metal, no underlying OS needed
  - > Better developer experience than the state-of-the-art
    - Standard Java development and debugging tools
    - Simple out-of-the-box experience (SpotWorld)
  - > Mid-level sensor device that can be battery powered
    - Enable exploratory programming
    - Enable more on device computation and reduce network traffic
    - Enable over-the-air programming

Future

- Collaborate with qualifying partners
- Use within Sun Labs
  - > Gesture based interfaces, building instrumentation, self-organising systems, etc.
- Iterate hardware design
  - > Smaller chips, lower power, cheaper, etc.
- Iterate VM
  - > Smaller footprint, faster, smarter interrupts, power management, etc.
- Open schematics and VM to the community?

The Teams

- Squawk
  - > Nik Shaylor (alumni)
  - > Bill Bush (alumni)
  - > Doug Simon (alumni)
  - > Cristina Cifuentes
  - > Derek White
  - > Eric Arseneau
- Squawk ARM Support
  - > John Daniels
  - > Dave Cleal
  - > Duncan Pierce
  - > Rachel Davies
  - > John Wilcox
- Sun SPOT Hardware
  - > Bob Alkire
  - > John Nolan (alumni)
  - > Del Peck
- Sun SPOT Software
  - > Randy Smith
  - > Bernard Horan (alumni)
  - > Vipul Gupta
  - > Samita Chakrabarti
  - > Rob Tow
  - > David Simmons
- Managers
  - > Roger Meike (Sun SPOT)
  - > Dan Ingalls (Squawk)

Questions

cristina.cifuentes@sun.com