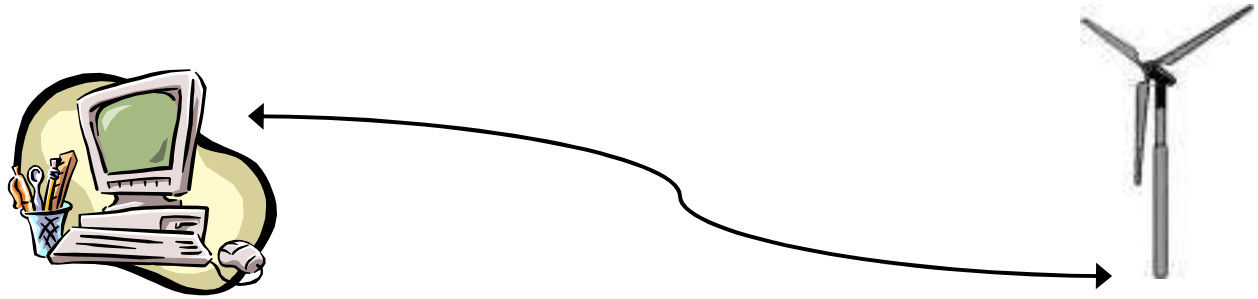
A New Approach to Developing Robust Embedded Software

oovm

by Lars Bak OOVM A/S



Purpose of the Presentation

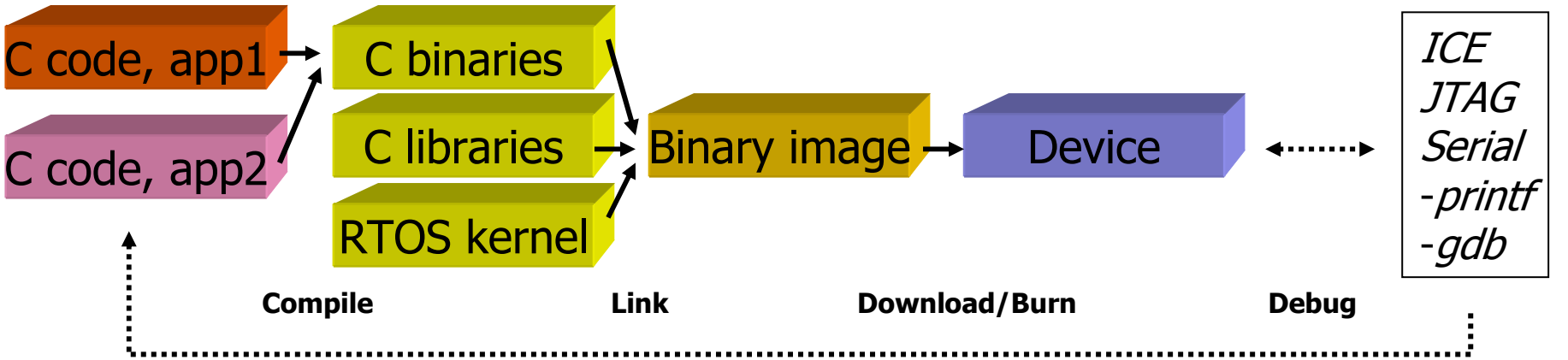
oovm

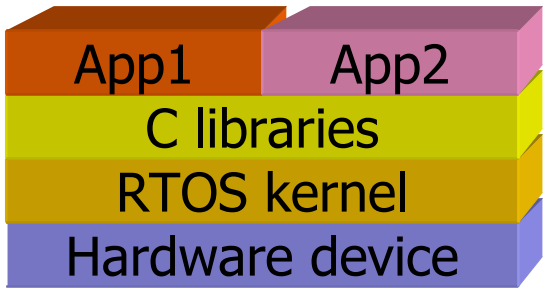
To argue for and present a new compact and fast embedded object-oriented system that can run both hosted and on the bare metal

The Company: OOVM A/S

- Venture financed startup company dedicated to creating a simpler and more reliable software platform for embedded systems
- Founders
  - Lars Bak, CEO
  - Steffen Grarup, CTO
  - Kasper Lund
  - Jakob Andersen

Embedded Software Today





- Slow development
- Low productivity
- Unsafe programming language
- No servicability
- Very static model

Demands of the Embedded Industy

oovm

- Increased reliability
- Low cost -> resource constraints
- Dynamic software updates in the field
- Real-time capabilities
- Rapid development cycles

Is embedded Java the solution?

Why not Embedded Java?

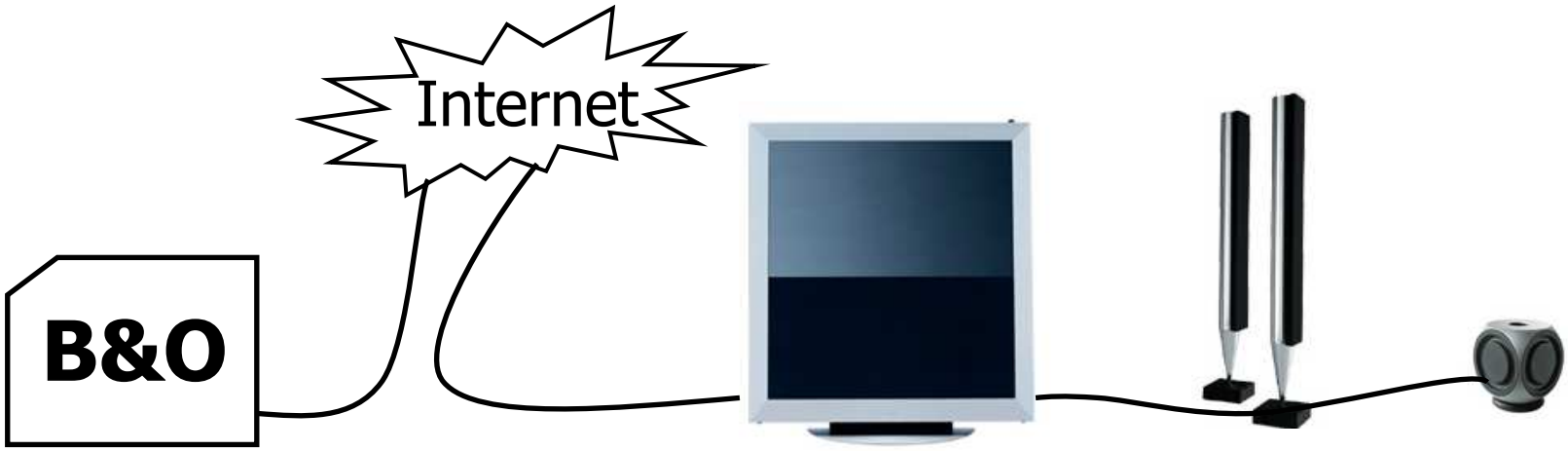
oovm

- Does not support incremental execution
- Virtual machine specification is very complicated
- Bytecodes not designed for speed and compactness
- Configurations for embedded systems too big
  - CLDC and CDC
  - And the configurations are growing...

What Can We Do It Better?

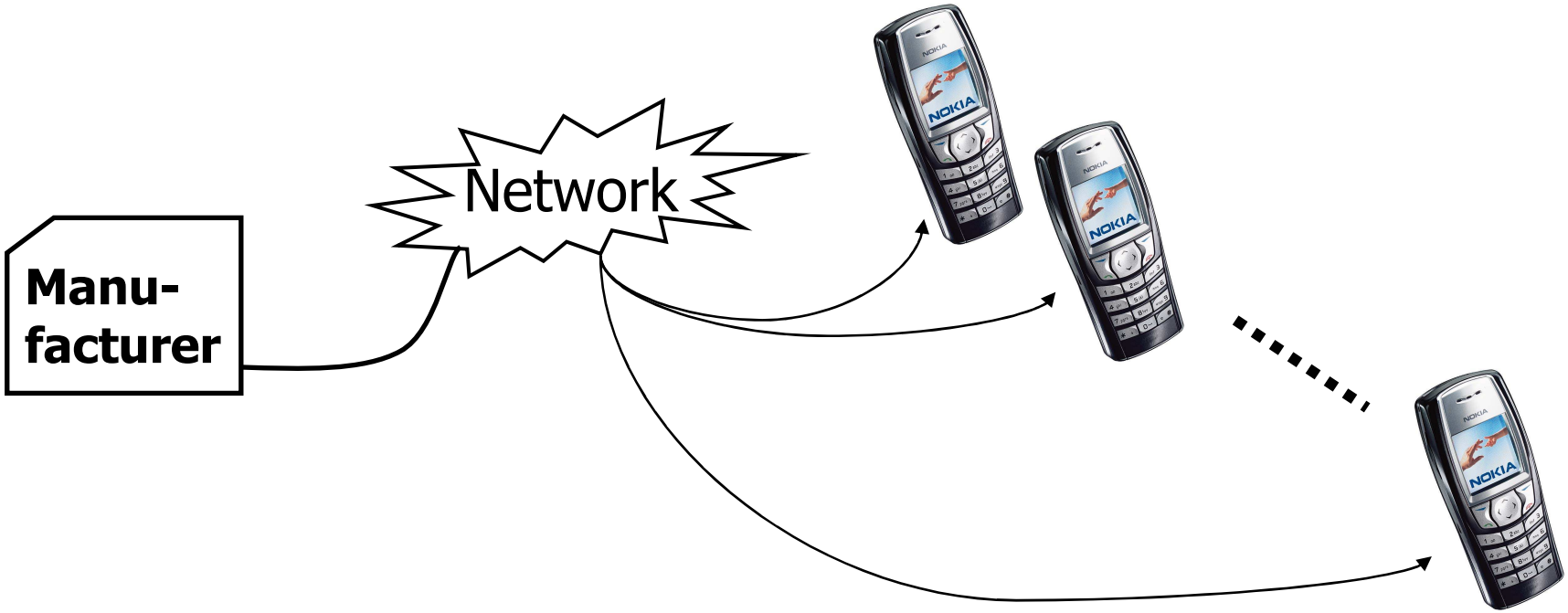
- Use of safe dynamic programming language
- Increase productivity
  - Connect programming environment to running system
  - Provide incremental execution
- Provide serviceablility
  - Debugging supported in production
  - On-the-fly software update

Scenario: Product Serviceability



- Product serviceability enables manufacturer to identify and fix potential problems at customer site
- In a pilot project with Bang & Olufsen, Resilient™ was used successfully to test serviceability of the controlling software in a digital speaker

Scenario:
Transparent Software Updates



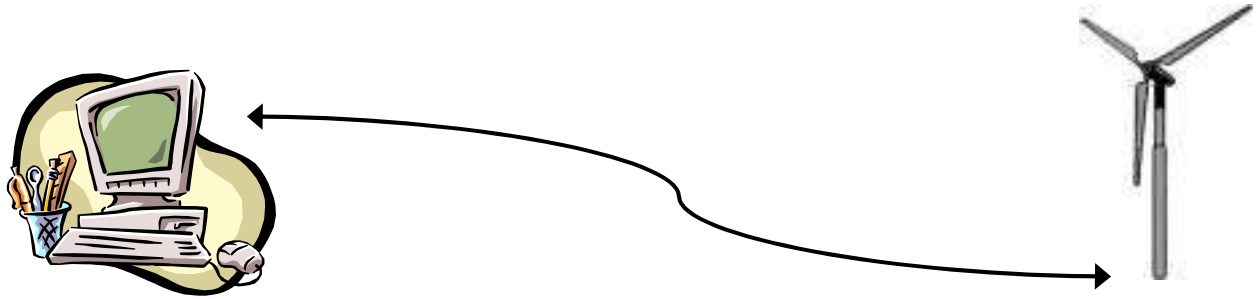
- Updates are minimal in size
- Automatically applied while software is running

Presentation Outline

- Resilient in embedded systems
- A different Smalltalk system
  - Reflection vs. execution
  - Atomic test and store statement
  - Namespaces
  - Typed LIFO blocks
  - Threading and scheduling
- Benchmarks
- Product licensing

The Resilient™ Product

- A software platform for creating secure, object-oriented, real-time software for embedded devices
  - Requires minimal memory
  - Increases development productivity
  - Enables analysis and correction of software problems in the field
  - Enables minimal updates of deployed software

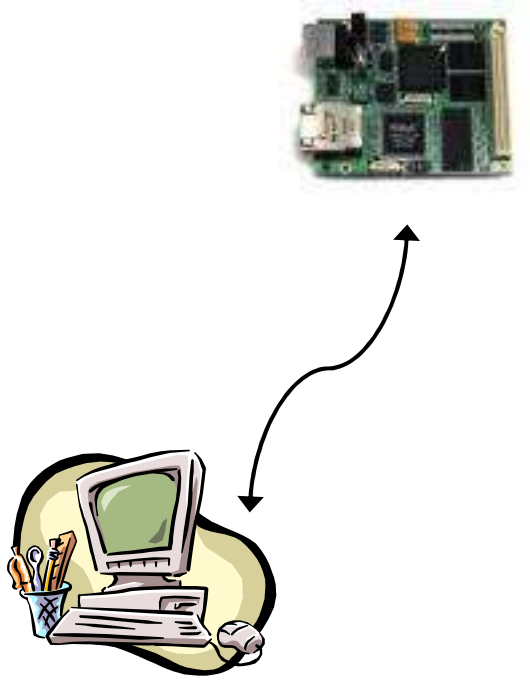


Resilient™ IDE

Resilient™ Embedded Platform

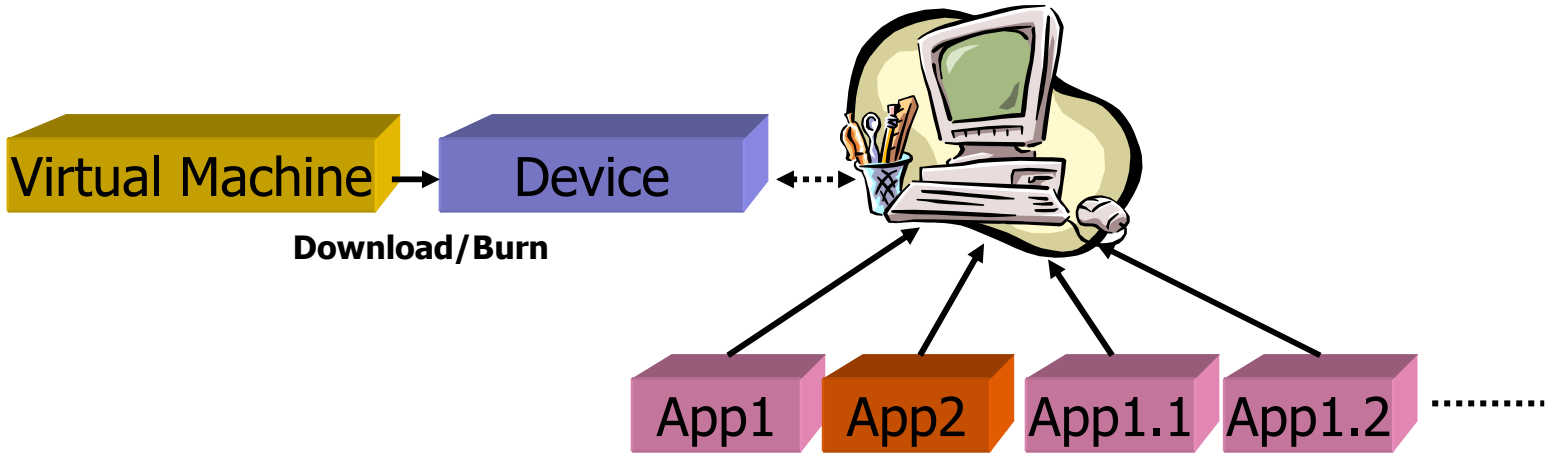
Product Components

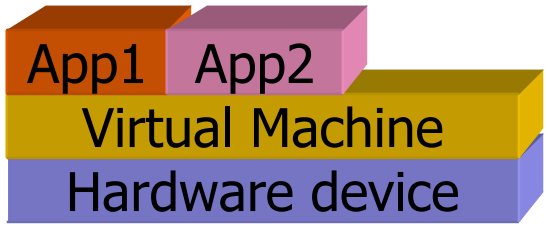
- **Resilient™ Embedded Platform**
  - Object-based operating system
  - Pure object-oriented system
  - Platform-independent mobile code
  - Libraries for device drivers and networking
  - Unified real-time resource management
- **Resilient™ IDE**
  - Connects to running system
  - Supports true incremental execution
  - Full serviceability and visibility



IDE

Embedded Software Using Resilient

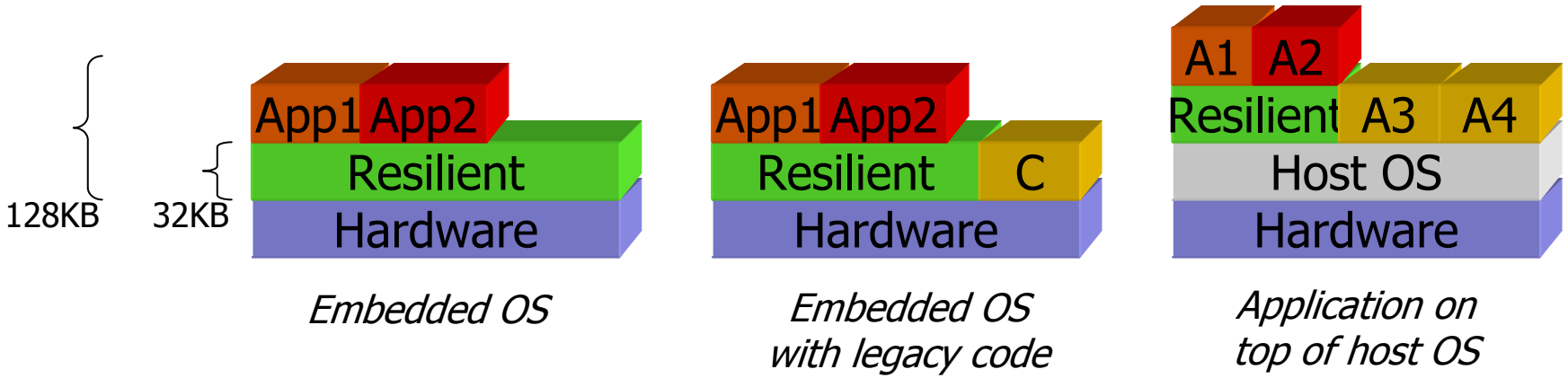


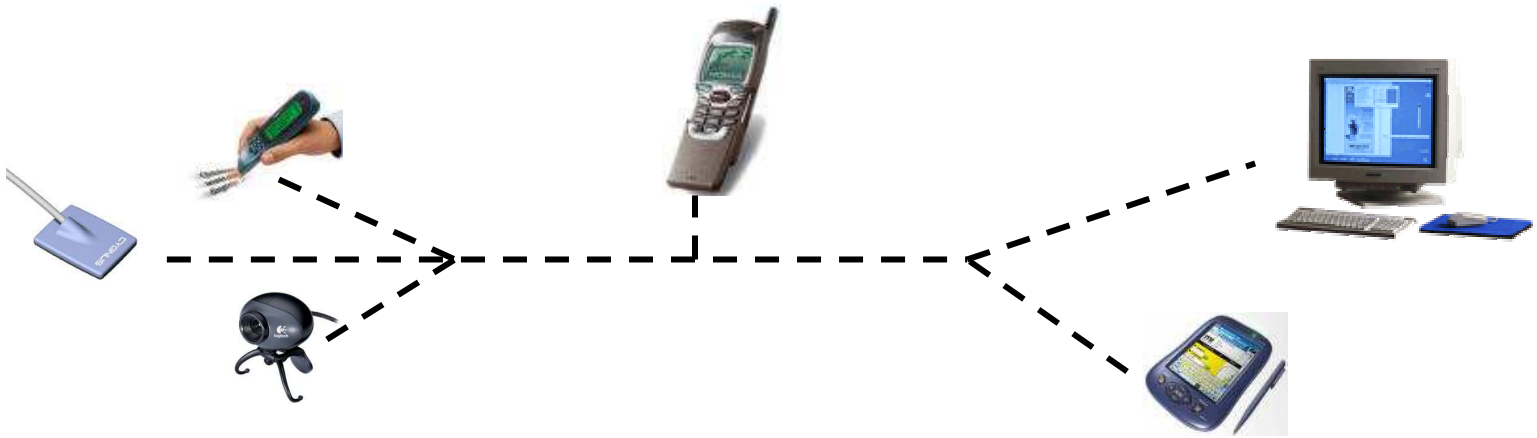


Deployed Product

- Runs directly on hardware
- Fully dynamic system
  - Full application isolation
  - Change anything on-the-fly
- Unified resource management
- Very small memory footprint

Resilient Deployment Options

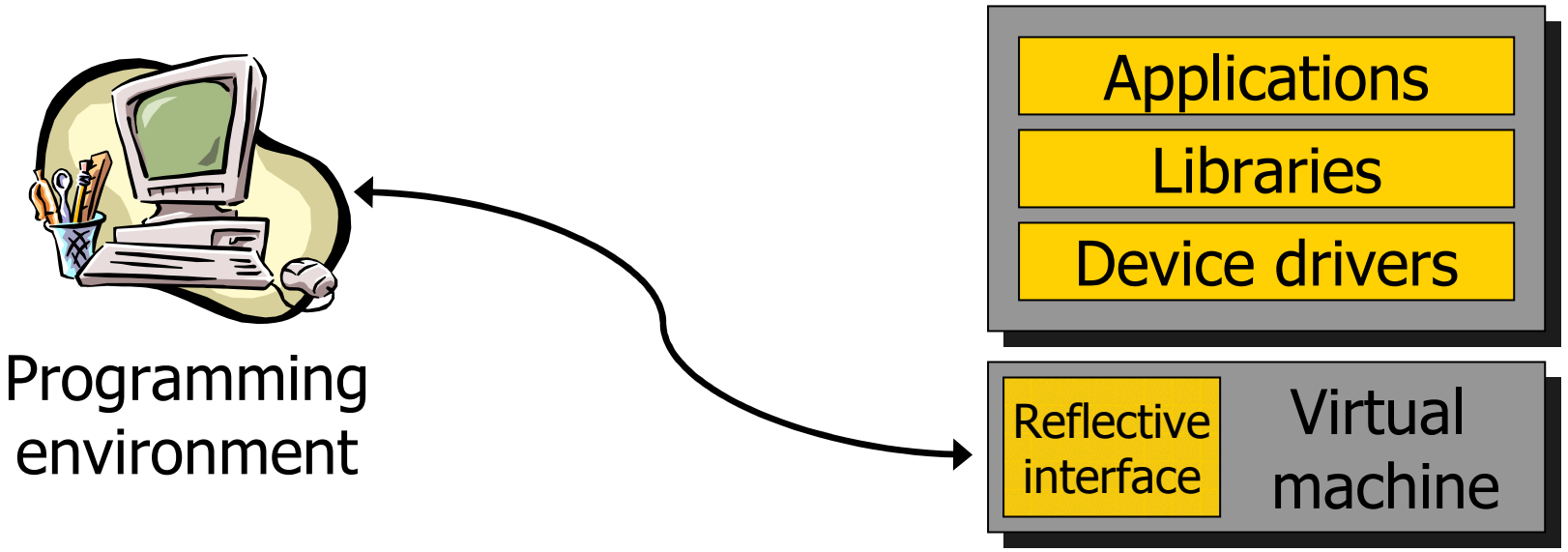




The Resilient System

oovm

- Programming environment provides all reflective behavior
- Virtual machine provides simple reflective interface



The Programming Environment

oovm

Eclipse UI customized for Resilient

- Programming support
- Bytecode compiler
- Debugging
- Profiling
- Introspection

Embedded system

The Virtual Machine

oovm

- Basic philosophy: simplicity
- 32bit virtual machine
- Scalable object heap
- Compact object model
  - 1-word object headers
- New bytecode set for Smalltalk
  - 20 bytecodes with uniform format
- Portable design

The Virtual Machine

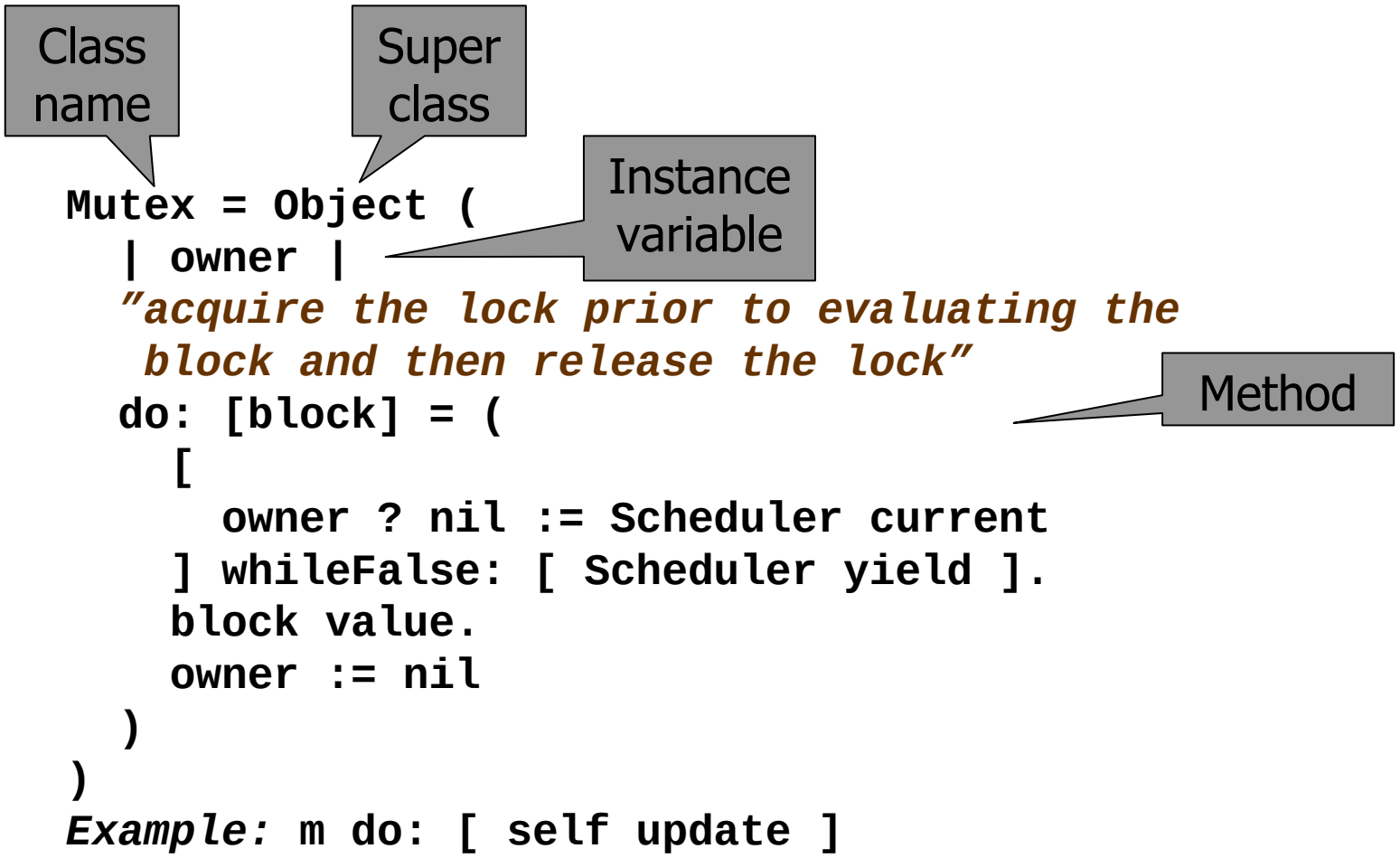
oovm

- Safe and fast control of
  - Memory mapped devices
  - Interrupts
- Unified automatic resource management
  - Real-time garbage collection
  - Policy based, user definable
  - Guaranteed allocation/scheduling behavior per thread/application
- Serviceability
  - True incremental program execution
  - Dynamic updating of user and system software with running apps
  - Full introspection even when running in production

The Programming Language

- Smalltalk with a few twists
  - Introduced
    - Syntax for full classes
    - Atomic test and store statement
    - Namespaces
    - Typed LIFO blocks
  - ... and removed
    - Pool variables
    - Class instance variables

Test and Store Example



Semaphore Implementation

```
Semaphore = Object (
  | count |

  acquire = (
    [ | c |
      c := count - 1.
      c < 0 ifTrue: [ ^Scheduler acquire: self ].
      count ? c + 1 := c
    ] whileFalse
  )

  release = (
    [ | c |
      c := count + 1.
      c < 1 ifTrue: [ ^Scheduler release: self ].
      count ? c - 1 := c
    ] whileFalse
  )
)
```

Class Libraries

oovm

- Minimal set of classes to provide basic execution behavior
- No reflective behavior
  - Only the programming environment can create classes
  - **`perform:`** is not supported
- Scheduler and device drivers
- Networking libraries
  - TCP/IP (SLIP, NIC, Firewire)

Integer Class Hierarchy

- Object
  - Integer
    - SmallInteger (30 bits)
    - LargeInteger (32 bits)

- Writing device drivers on a 32 bit computer requires 32 bit arithmetic

Collection Class Hierarchy

- Object
  - Collection
    - OrderedCollection
      - IndexableCollection
        - Interval
        - String
          - CompactString
          - UnicodeString
        - UpdatableIndexableCollection
          - Array
          - ByteArray
          - ObjectArray
      - UpdatableOrderedCollection
        - List
        - Tree
    - UnorderedCollection
      - Dictionary

Namespaces

- Desirable for modularizing code and dynamic application loading
- The namespace consists of nested classes
- Any class can be a namespace
- Examples:
  - **`Services::DebuggerAgent install`**
  - **`::Network::Services::DebuggerAgent install`**

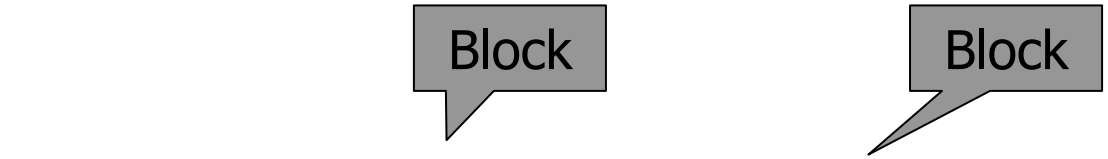
Achilles Heel of Smalltalk Performance

- Allocation of block contexts
  - Inlining of basic control structures
  - Flattening of code (ex. Collection hierarchy)
- Interpretation overhead
- Slow method invocation
  - Results in breaking down code abstractions

... or apply advanced inlining compiler

Typed LIFO Blocks

- Stack allocated contexts require no-escape-guarantee
- Blocks cannot be returned nor stored into heap objects
- Example from Collection

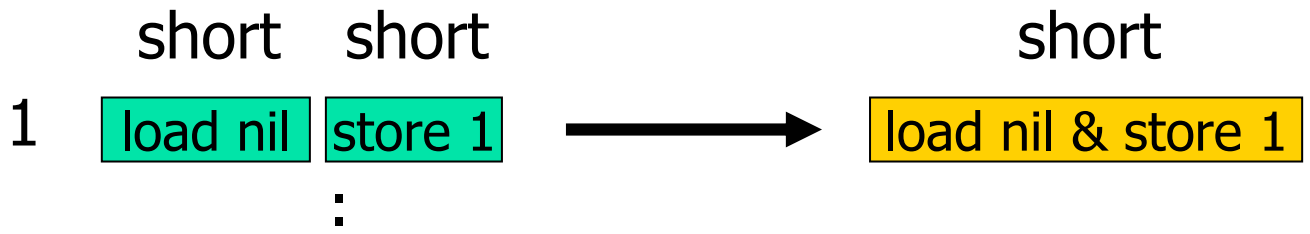



```
collect: [collect] do: [block] = (
  self do: [ :e | block value: (collect value: e) ].
)
```

Making Bytecodes Compact

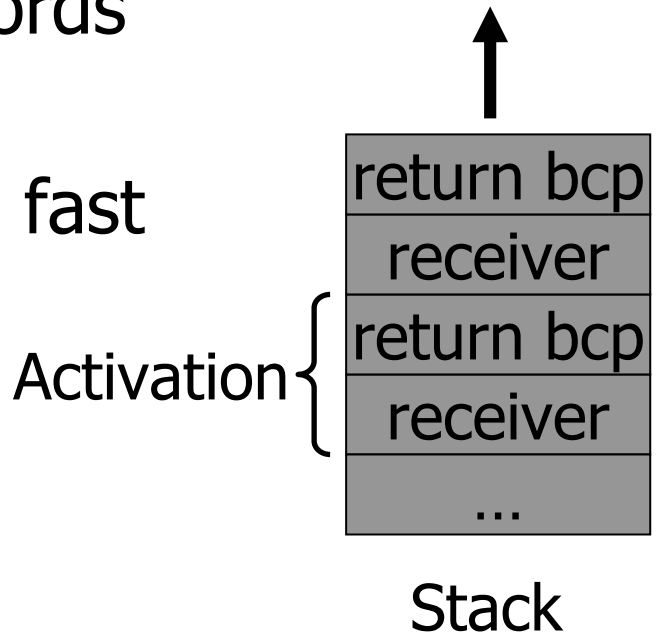
oovm

- Bytecode set designed for compactness
  - 20 simple bytecodes
- Methods with identical bytecodes are shared
  - Saves 10% of space used by methods
- Super bytecodes are computed based on static bytecode-pair-histograms
  - Reduces the bytecodes with 45%



Compact Execution Stacks

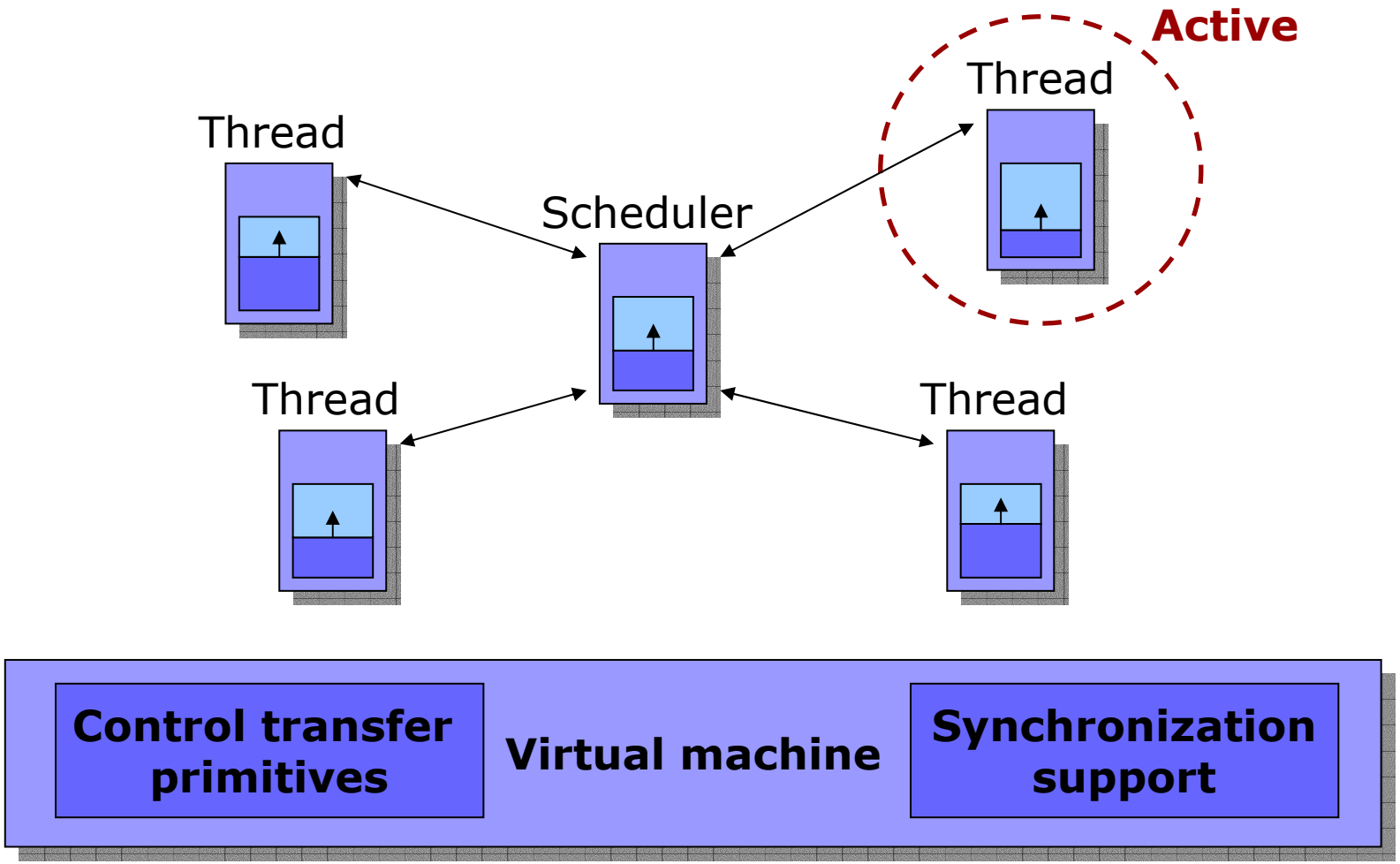
- Stacks contain activations but are also objects
- Initial size is 512 bytes but grows as needed
- Minimal activation size is 2 words
  - No frame pointers
- Send bytecodes then become fast



Resilient OS Layer Services

- Threads
- Stacks
- Synchronization
- Device driver API
  - Memory mapped I/O
  - Interrupts

Threads and Scheduling

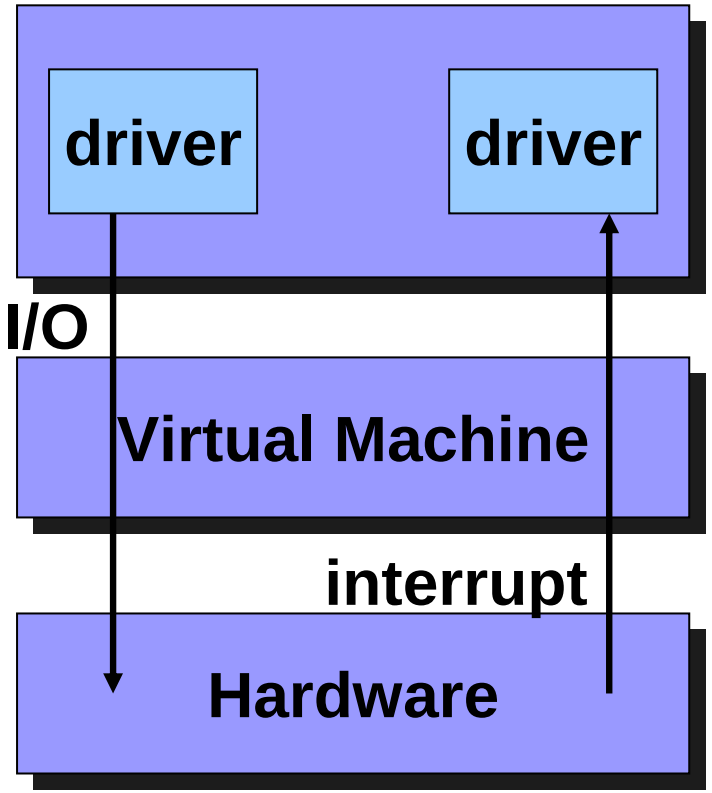


Threads and Scheduling

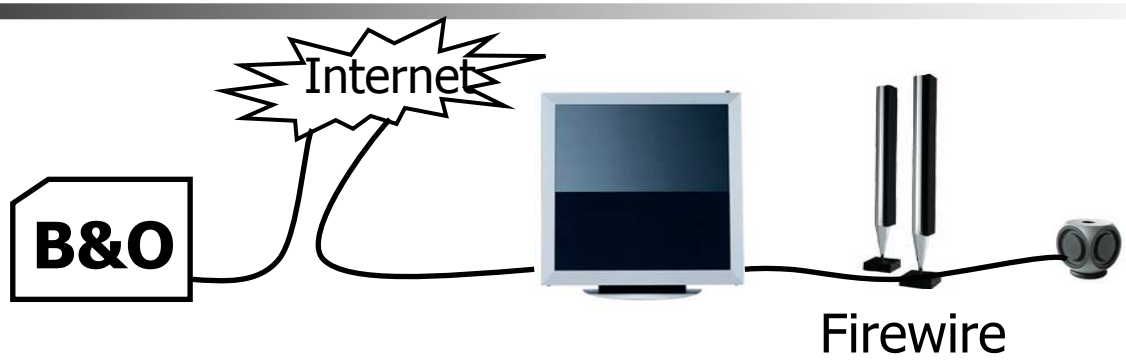
- Quasi-parallel coroutine system
  - **Asymmetrical**: Control is transferred between the supervisor coroutine and the other coroutines
  - **Preemptive**: Control transfers can be asynchronous, i.e. not initiated by the active coroutine

Device Driver API

- Simple design, mimics raw hardware
- Virtual machine is hardware abstraction layer
- Provides all necessary functionality for implementing device drivers



Serviceability



- Programming environment connect to running embedded system
- Enables debugging and updating
- A set of changes can be “atomically” applied to preserve integrity of system

Is Interpretation Fast Enough?

oovm

- Resilient interpreter will be 2x the speed of the fastest interpreted JVM
- Profiled based compilation is possible for performance critical code
- However, too much compiled code will compromise memory footprint

Resilient System Characteristics

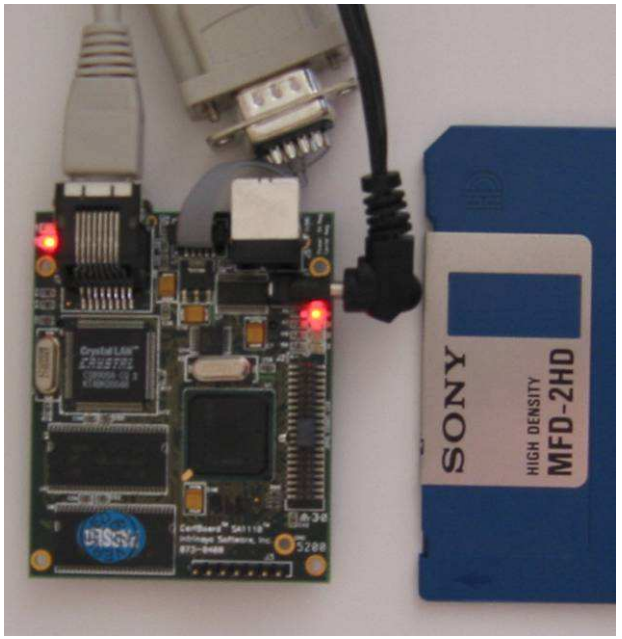
- The system runs all the time!
- Compact memory footprint
  - Minimal system executes in 128KB
  - Smaller than all OS+Java systems
- High performance
  - 2x fastest interpreted JVM
- Minimal power consumption
  - Performance is important for battery life
  - Battery size often determine the product size

Resilient Demo

oovm

Resilient Supported Platforms

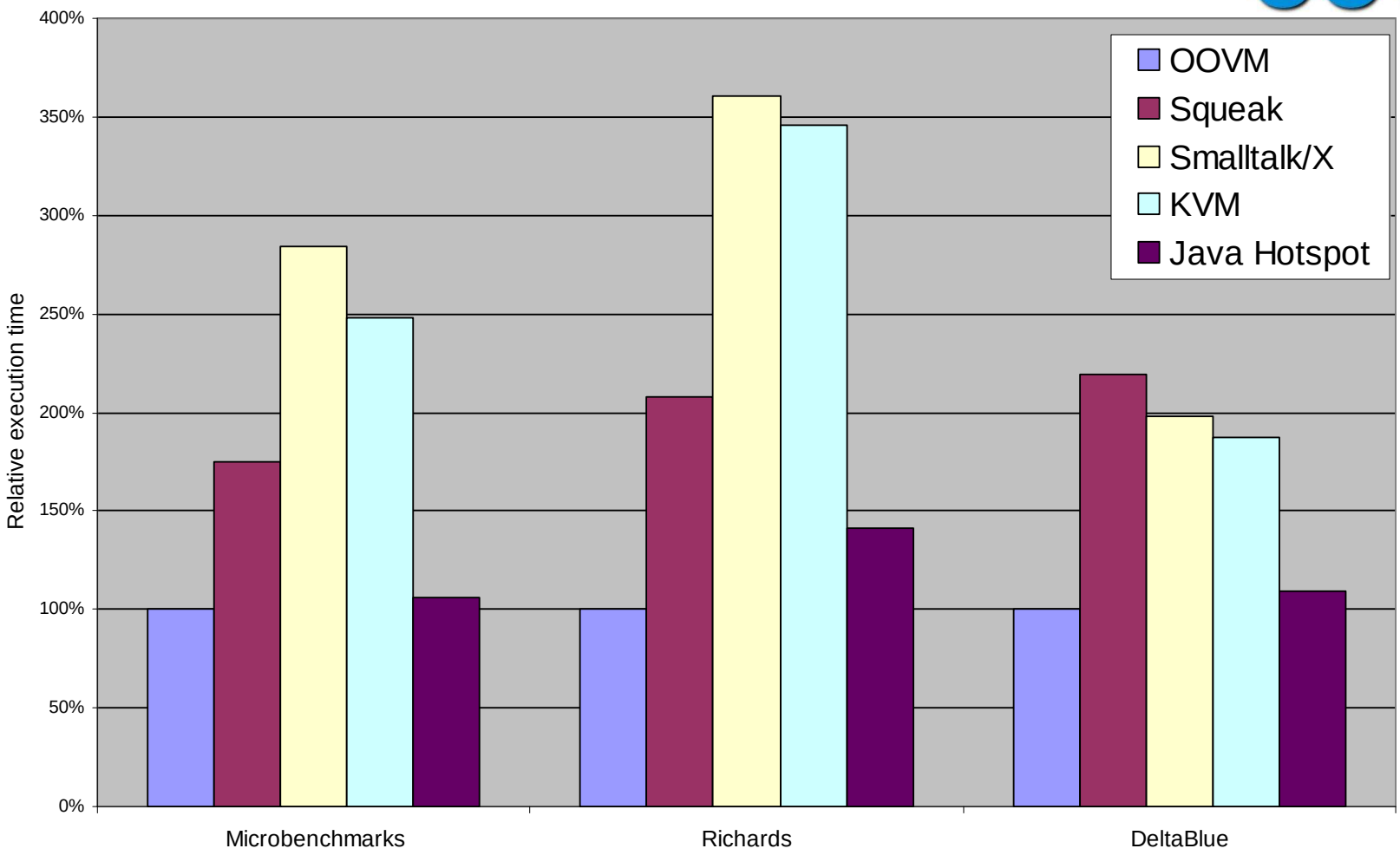
- Embedded platforms
  - Intrinsyc Cerfcube eval board
    - 200MHz StrongARM
    - 32MB RAM, 16MB Flash
  - ICE Lynx from TI
    - 50MHz StrongARM
    - 256KB RAM, no flash
    - Firewire
    - Audio and video streaming
- Hosted platforms
  - IA32/Linux
  - StrongARM/Embedded Linux
  - Windows



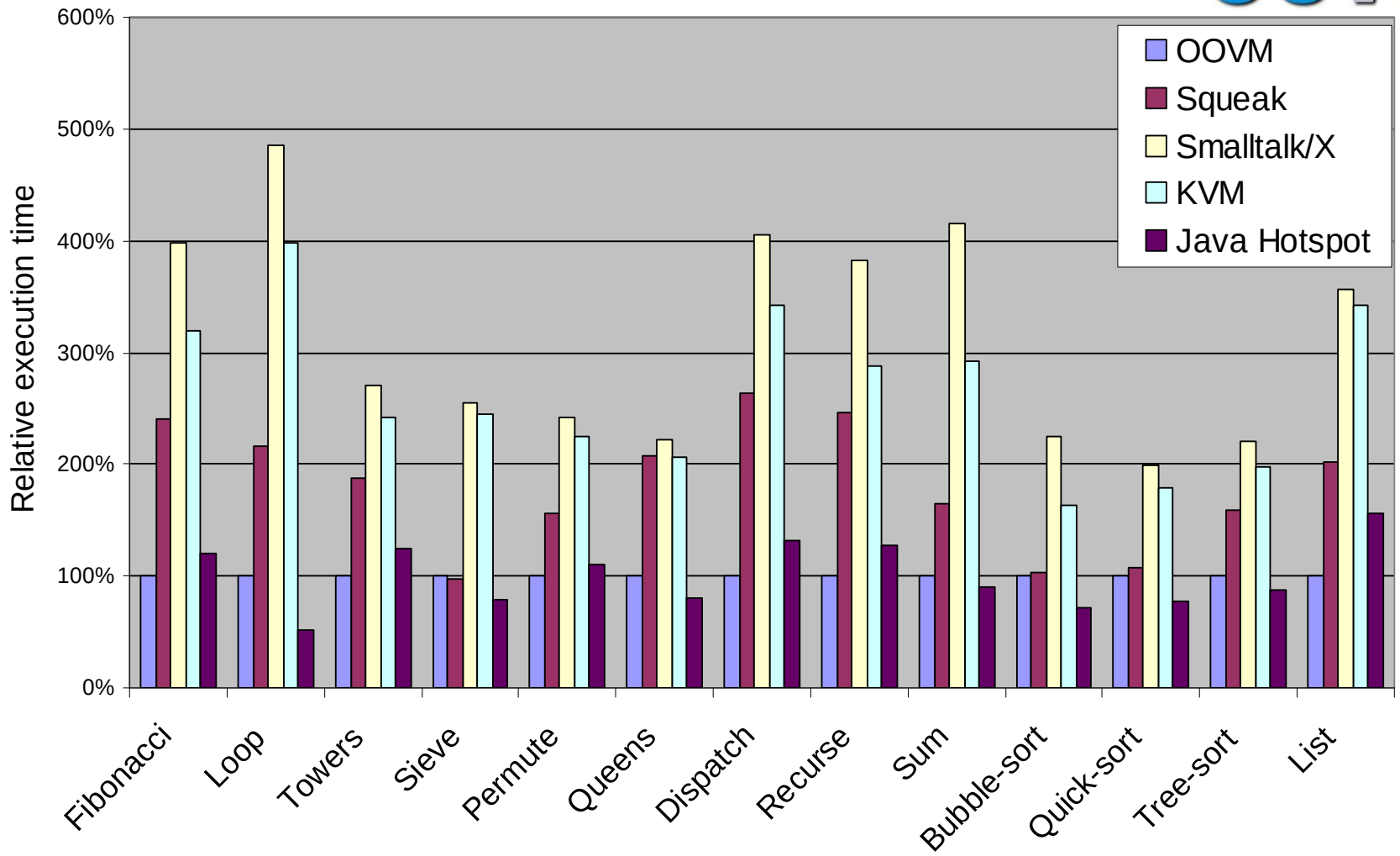
Benchmarking Resilient

- Benchmarks
  - Microbenchmark: Stanford integer benchmarks suite
  - DeltaBlue
  - Richards
- Resilient compared to
  - Smalltalk-X version 4.1.7 (JIT disabled)
  - Squeak version 3.2-4
  - Java KVM version 1.0.4
  - Java HotSpot version 1.4.0 (JIT disabled)
- Benchmarking platform
  - Red Hat Linux 7.3
  - Intel IA-32 PIII 1133MHz

Benchmark Results



Microbenchmark Results



Resilient Availablity

- Technology preview available today from **www.oovm.com**
- Resilient version 1.0 scheduled for July 2004
- Dual licensing model
  - Free non-commercial use
  - Commercial use requires a license

Summary

- Resilient™ is the next-generation platform for embedded systems
  - Enables smaller, lower power, lower cost devices
  - Delivers product serviceability
  - Alleviates skyrocketing software development costs
  - Delivers more reliable and secure products