

JEM Architecture Revisited

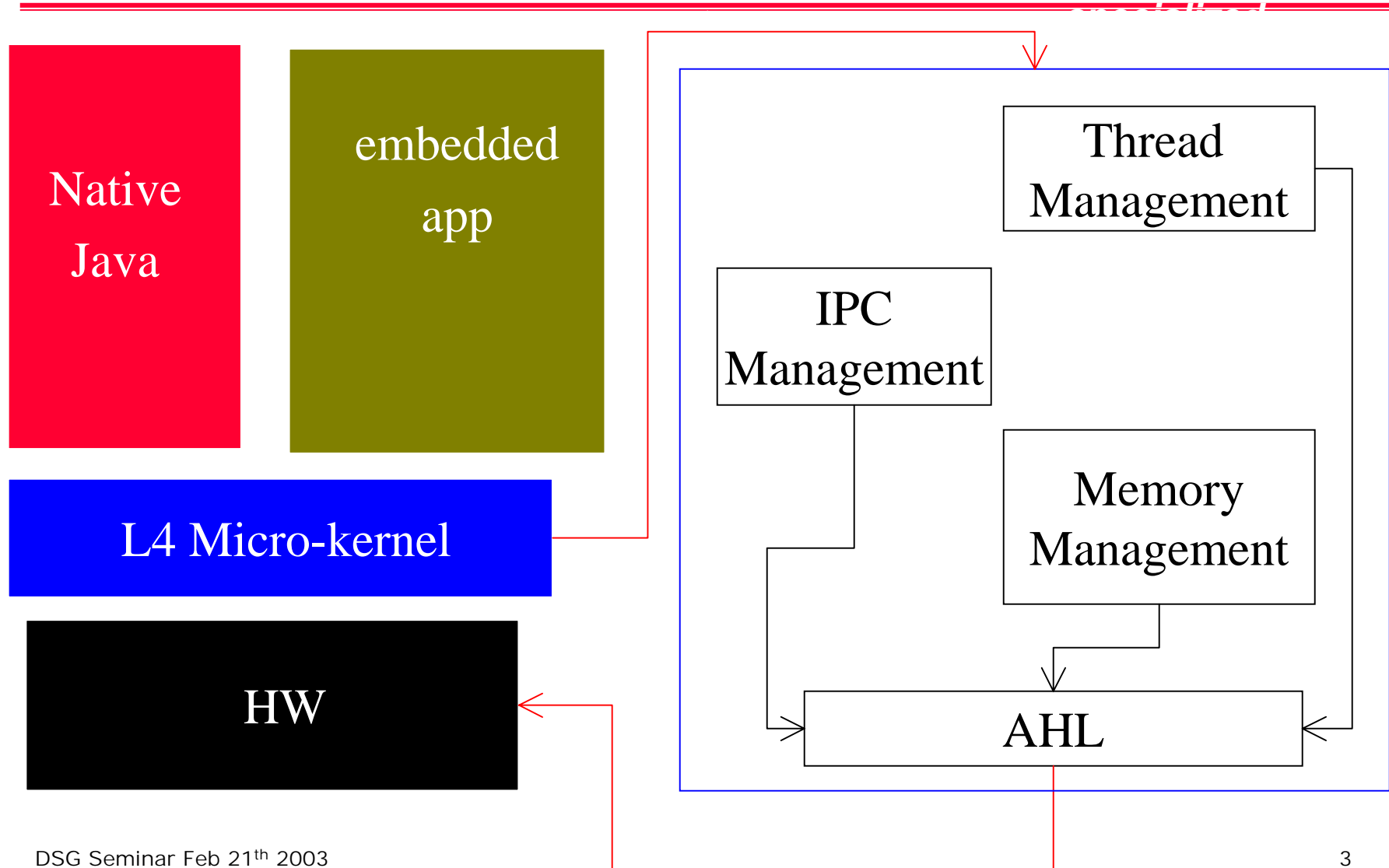
Hong Ong
Distributed Systems Group
University of Portsmouth

hong.ong@port.ac.uk
<http://dsg.port.ac.uk/hong>

Outline

- Part I:
 - Some changes to JEM design (JEM+ +?).
 - Some implementation issues.
- Part II:
 - Some general issues.
 - Some open questions.

Recall Original JEM Model



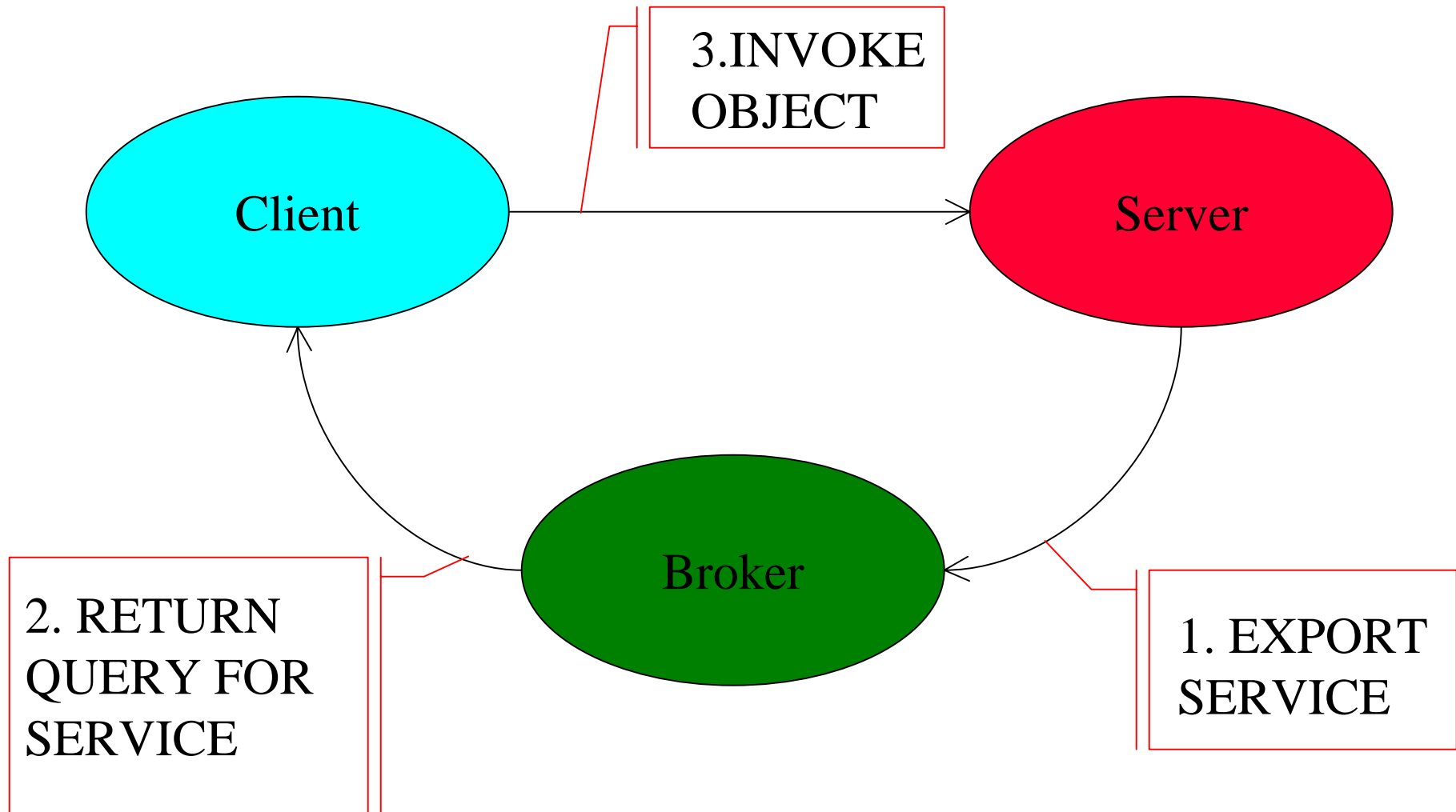
“Problems” with the original Model

- Pre-defined kernel philosophy.
 - Why can't I build an exokernel or application specific kernel?
- Pre-defined kernel core functionality.
 - What kernel functionalities must we provide?
 - e.g., Are memory, thread, and IPC abstraction sufficient to accommodate different requirements?
- Pre-defined configuration.
 - Can we reconfigure what we want either at compile time or run time?

What we really want? Gee, again!?

- Java as an implementation language.
- Abstract Hardware Layer.
 - Provide hardware independence function.
 - Separate language and hardware architecture.
- No predefined kernel philosophy.
- No predefined kernel functionality.
- Systematically applied the concept of module.

Inspiration – Trading Service!



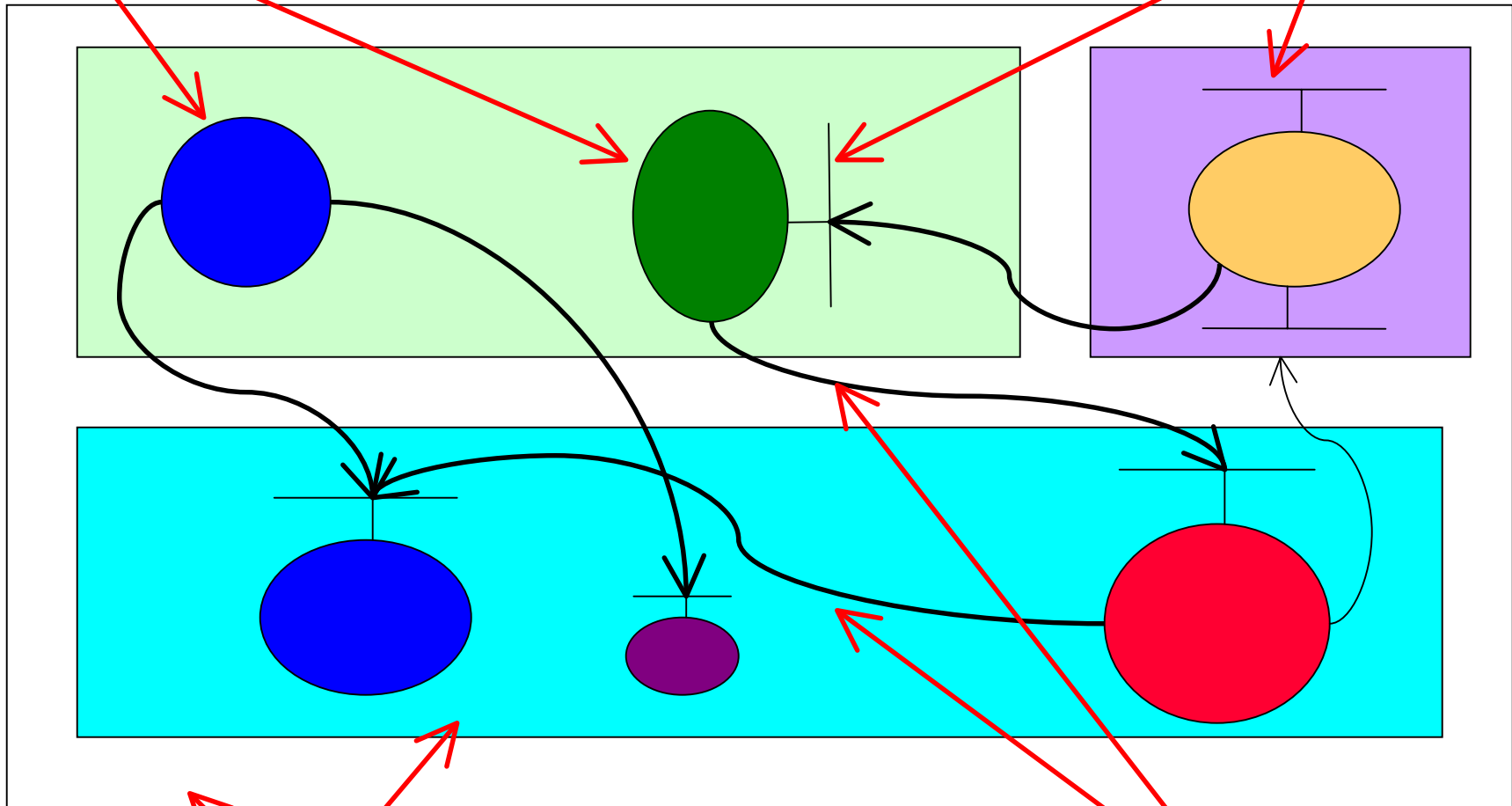
“New” Concepts/Definitions

- Module.
 - A finite set of well defined interfaces.
 - A finite set of Global and Private data.
 - A unit of configuration and deployment.
- Interfaces
 - A finite set of methods.
- Bindings.
 - The end result of establishing a communication channel between two or more objects.
- Domain.
 - A finite set of modules.
 - A unit of protection.

Modules

"New" Framework

Interface



Domains

Binding

Implementation - framework

```
interface Reference {}
interface Name {
    Context getContext();
    byte[] encode();
}
interface Context {
    Name export(Reference ref);
    Name decode(byte[] stream);
}
interface Binding {
    Reference bind(Name name)
}
```

Implementation – Memory Example

```
interface Allocator {
    byte[] alloc(int size);
    void free(byte[] addr);
}
interface Space{
    void map(byte[] virtpage, int physpage,
            int wimg, int pp);
    void unmap(byte[] virtpage);
    int tophys(byte[] virtpage);
    byte[] ioremap(int physaddr, int size,
                  int wimg);
    void iounmap(byte[] virtaddr, int size);
    int getid();
    void setspace();
}
```

Implementation

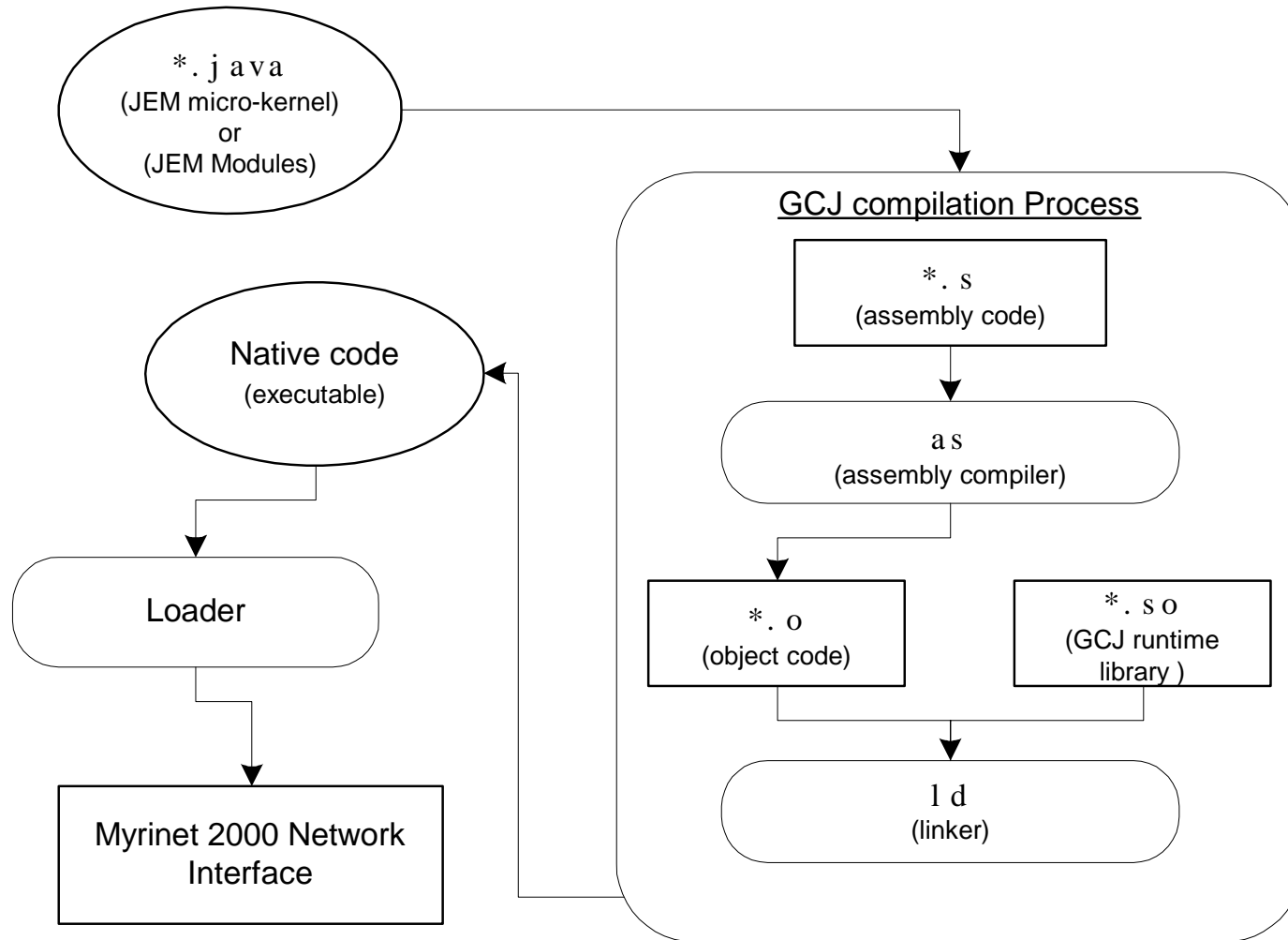
- Tools
 - Code generation – a J2C compiler
 - Extends IBM jikes compiler to generate C code.,
 - produce interface and class reference (a pointer)
 - Generate “stub” to glue the modules together.
 - Configuration
 - Determine the necessary modules by reading meta-data written in XML syntax

Implementation – Memory Example

```
<module name="buddy" class="memory">  
<file name="buddy"/>  
<produces interface="page"/>  
<consumes interface="bind"/>  
<consumes interface="trader"/>
```

```
<module name="allocator" class="memory">  
<file name="dlmalloc"/>  
<produces interface="allocator"/>  
<consumes interface="sbrk"/>  
  <consumes interface="trader"/>  
</module>
```

Implementation Strategy



L4-kernel

```
<module name="L4">
```

```
<description>
```

This module implemented the L4-like micro-kernel.

```
</description>
```

```
<file name="l4" />
```

```
<requires name="emu_boot" />
```

```
<requires name="emu_allocator" />
```

```
<requires name="emu_property" />
```

```
<requires name="emu_thread" />
```

```
<requires name="emu_lowscheduler" />
```

```
<requires name="emu_trap" />
```

```
<requires name="roundrobin" />
```

```
<requires name="space" />
```

```
<requires name="allocator" />
```

```
<requires name="trader" />
```

```
<requires name="localbind" />
```

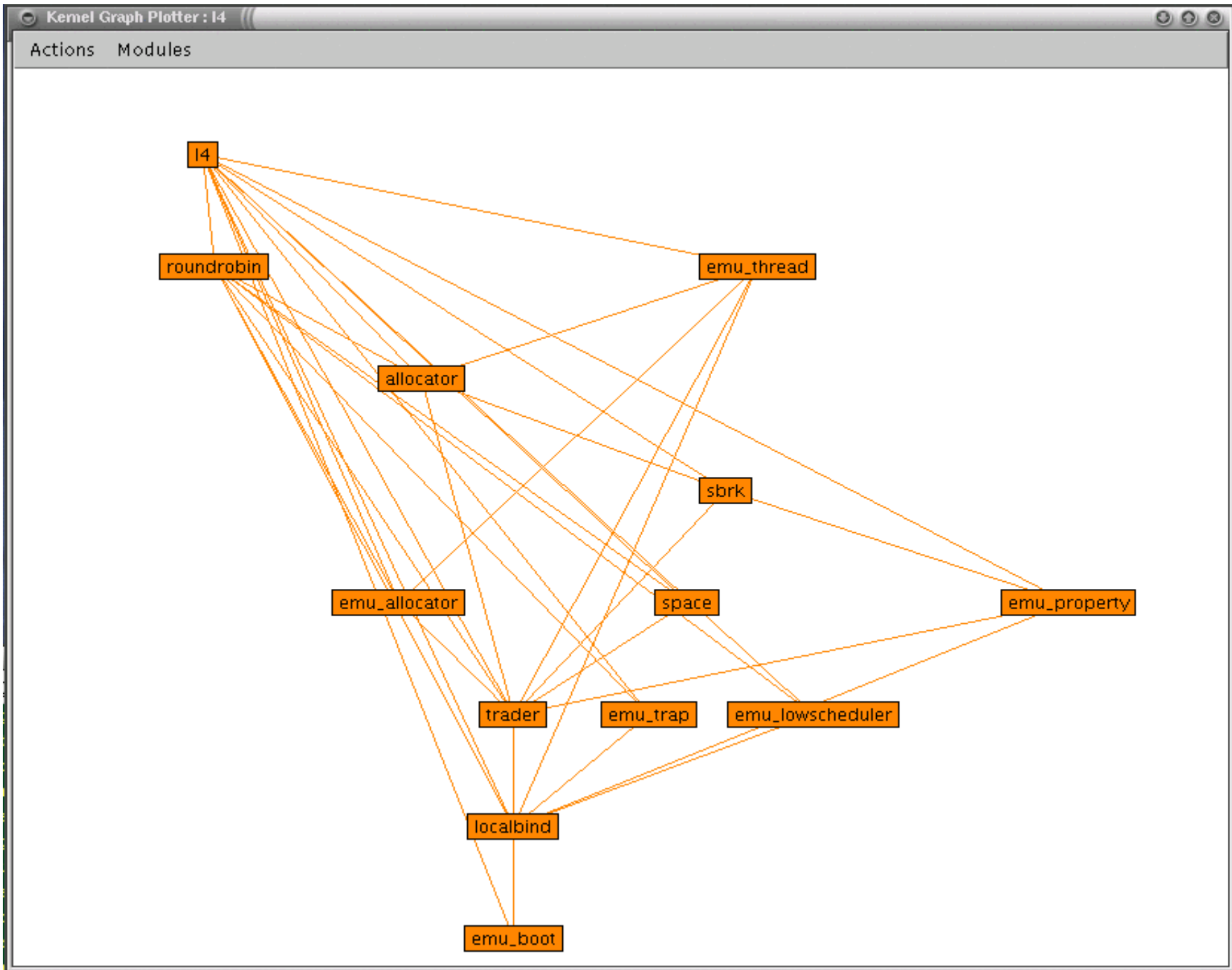
```
<requires name="sbrk" />
```

```
</module>
```

L4-Kernel cont'

```
/** L4-like Micro kernel */
#include <jem.h>
/** Main method call by jem */
unsigned char kernelstack[4096];
unsigned int kernelstacksize = 4096;

void prekernelstart(unsigned long offset) {}
void l4Probe(void) { while(1); }
```



Part II. DISCUSSION ...

Some Issues - General

- Is the name “JEM” denote/represent our framework?
 - Need a better name to encapsulate the framework!!
- Are “Module” and “Interface” defined correctly?
 - Behavior, synchronization, QoS
- Is the framework capable of self analysis?
 - Correctness and performance
- How is the module connected?
 - COM, DCOM, COBRA, etc ...
- Is dynamic (re)configuration possible?

Some Open Questions

- Configuration process.
 - How to guide developer in choosing the right module?
 - How to analyze the resultant composition of modules is correct and meet the requirements?
- Software development.
 - How to develop lightweight interface?
 - How to define metrics and developing technique for categorizing modules along memory size, security, and QoS?
 - How to develop and save configuration information?
 - Is there really a generic "Infrastructure"? Is it possible to have a generic "Infrastructure"?

Some Open Questions (cont')

- Reconfigurable hardware.
 - What is the impact on OS and application services for both functional and non-functional attributes?
 - How to determine when the performance gain is worth the cost?