The Flux OSKit:
A Substrate for Kernel and Language Research

Bryan Ford Godmar Back
Greg Benson Jay Lepreau
Albert Lin Olin Shivers

Computer Systems Laboratory
University of California, Davis
MIT AI Lab

flux@cs.utah.edu

http://www.cs.utah.edu/projects/flux/

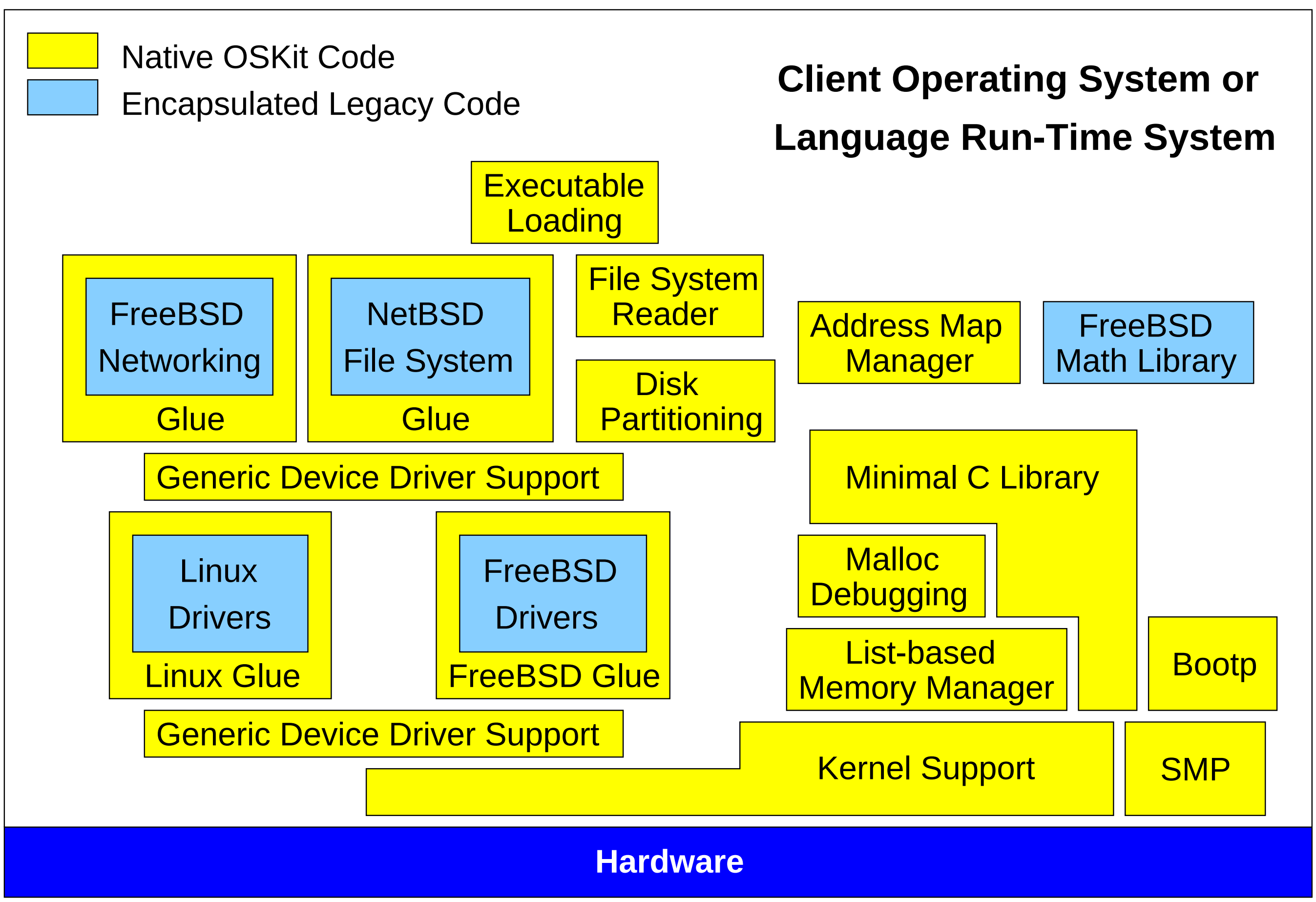
October 6, 1997

Motivation

OS research and development has a high cost of entry due to mundane infrastructure:

- Bootstrapping
- Basic kernel runtime environment
- Device drivers for diverse hardware
- Compatibility with existing systems

Reusable Components for OS Development



Key Concepts

Our approach to component-based OS's:

- Don't create a new OS; instead create components that can be used in *other* OS's.

- Don't rewrite from scratch when possible; reuse existing OS code in a maintainable way by *encapsulating* it within glue code.

- Emphasis on usability and practicality, not religion or buzzword-compliance.

Reusable Components for Arbitrary Environments

Component must have *some* expectations of its environment.

For reusability, expectations should be:

- Simple
- Well-defined
- Unconstraining

Important Properties of OSKit Components

Inter-component interfaces based on Microsoft's Component Object Model (COM).

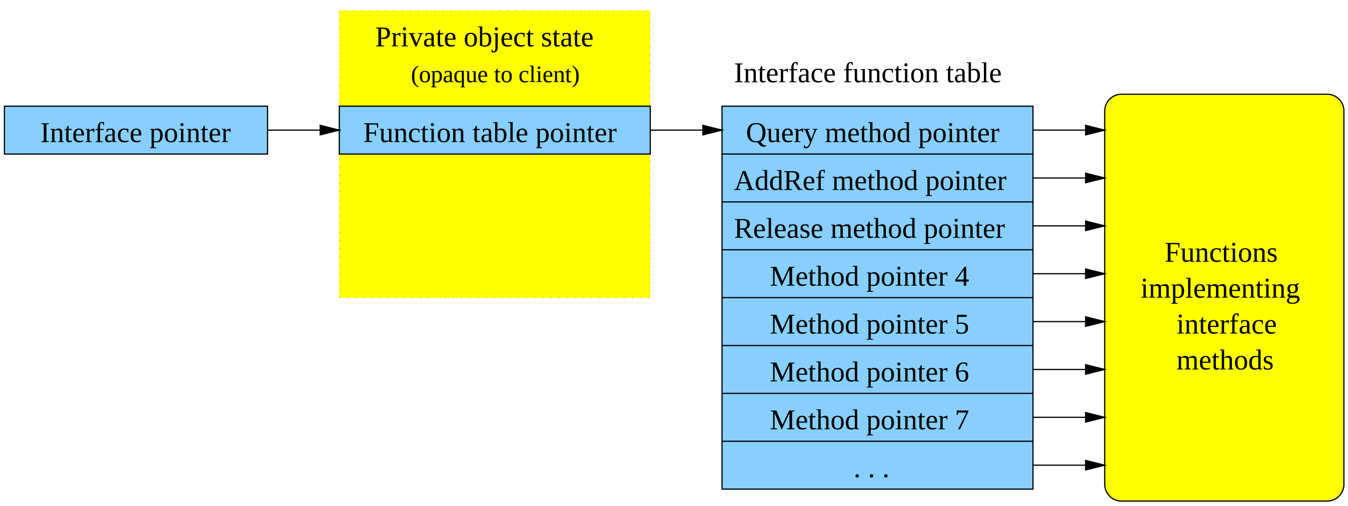
Minimal interdependencies, *no* mandatory global infrastructure.

Common uniprocessor/blocking concurrency model.

COM interfaces

- Similar to Java interfaces
- Standardized and well-known in industry
- Separates interface from implementation
- Supports independent interface extension and evolution
- No required runtime support code

Diagram of a COM Interface



No Implicit Dependencies

Components depend on only a handful of well-defined, easily reimplementable functions:

- Memory allocation
- Synchronization primitives
- Error printing/logging
- Hardware access (for device drivers)

Other facilities used by particular components are parameterized through COM interfaces.

No Implicit Dependencies

e.g., contrasts with:

- BSD's VFS and networking architecture: requires common `vnode`/`mbuf` code.

- Win32-based COM environment: requires various parts of the Win32 API

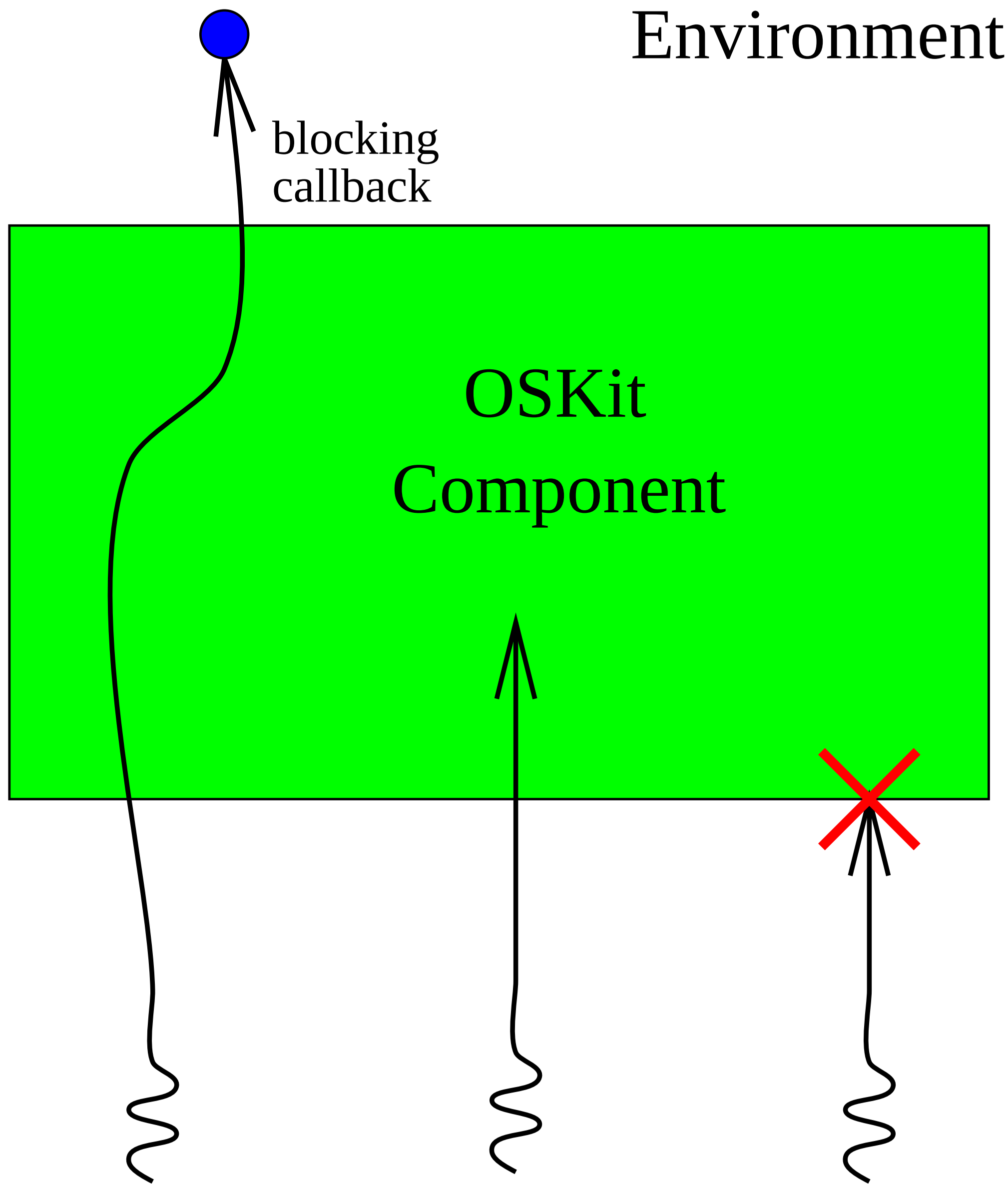
OSKit Concurrency Model

Defines:

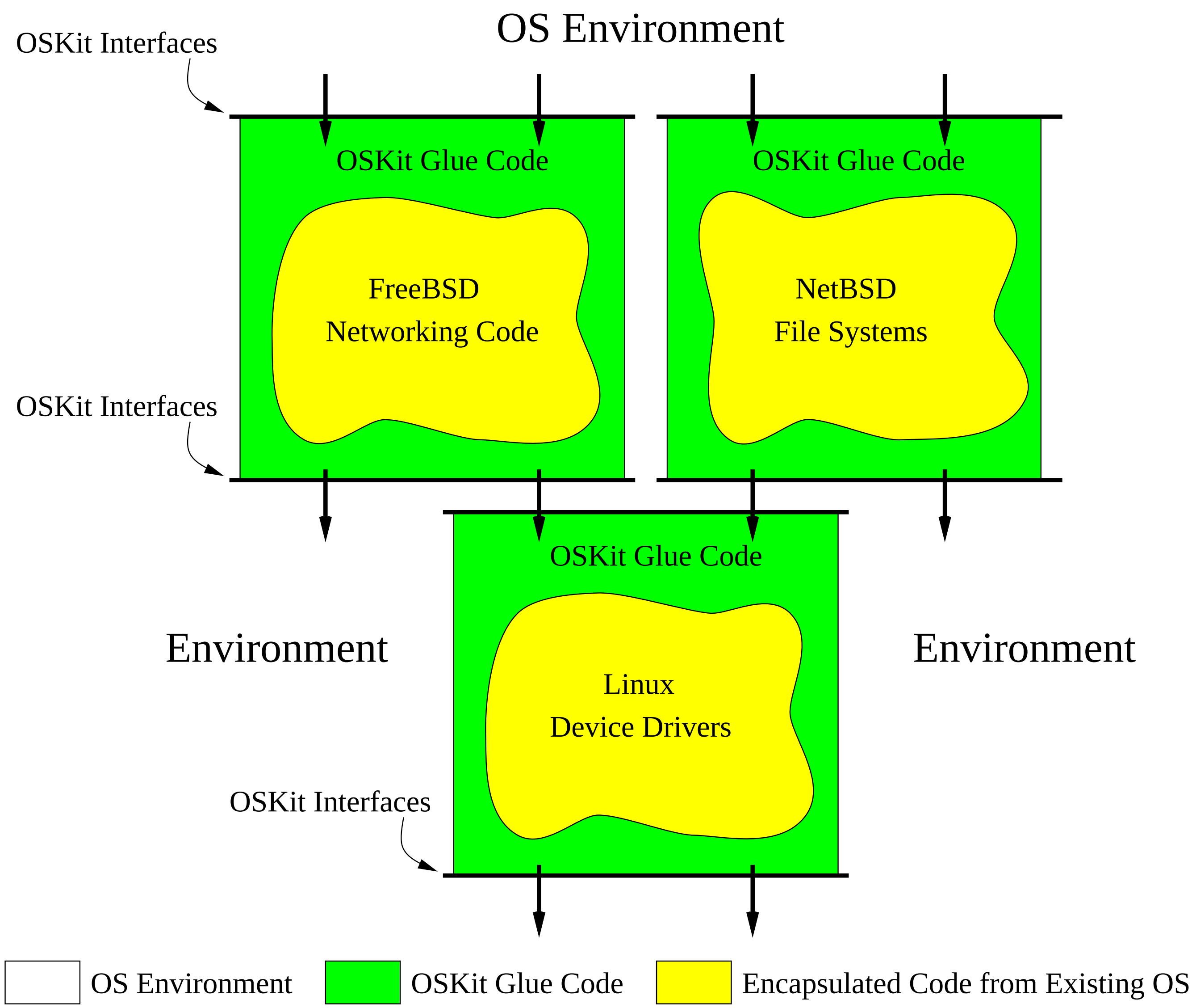
- How and when component can be invoked
- How and when the component can make callbacks to its surrounding environment.

OSKit uses the well-known blocking model, carefully defined and documented in a *component-centric* way.

OSKit Concurrency Model



Encapsulation of Legacy Code



Challenges for Encapsulation

Imported code makes many assumptions:

- `proc`/`task` structures
- The "current process" variable
- Memory allocation and mapping facilities
- Sleep/wakeup facilities
- Interrupt priority levels
- `mbuf`, `skbuff`, `vnode` infrastructure, etc.

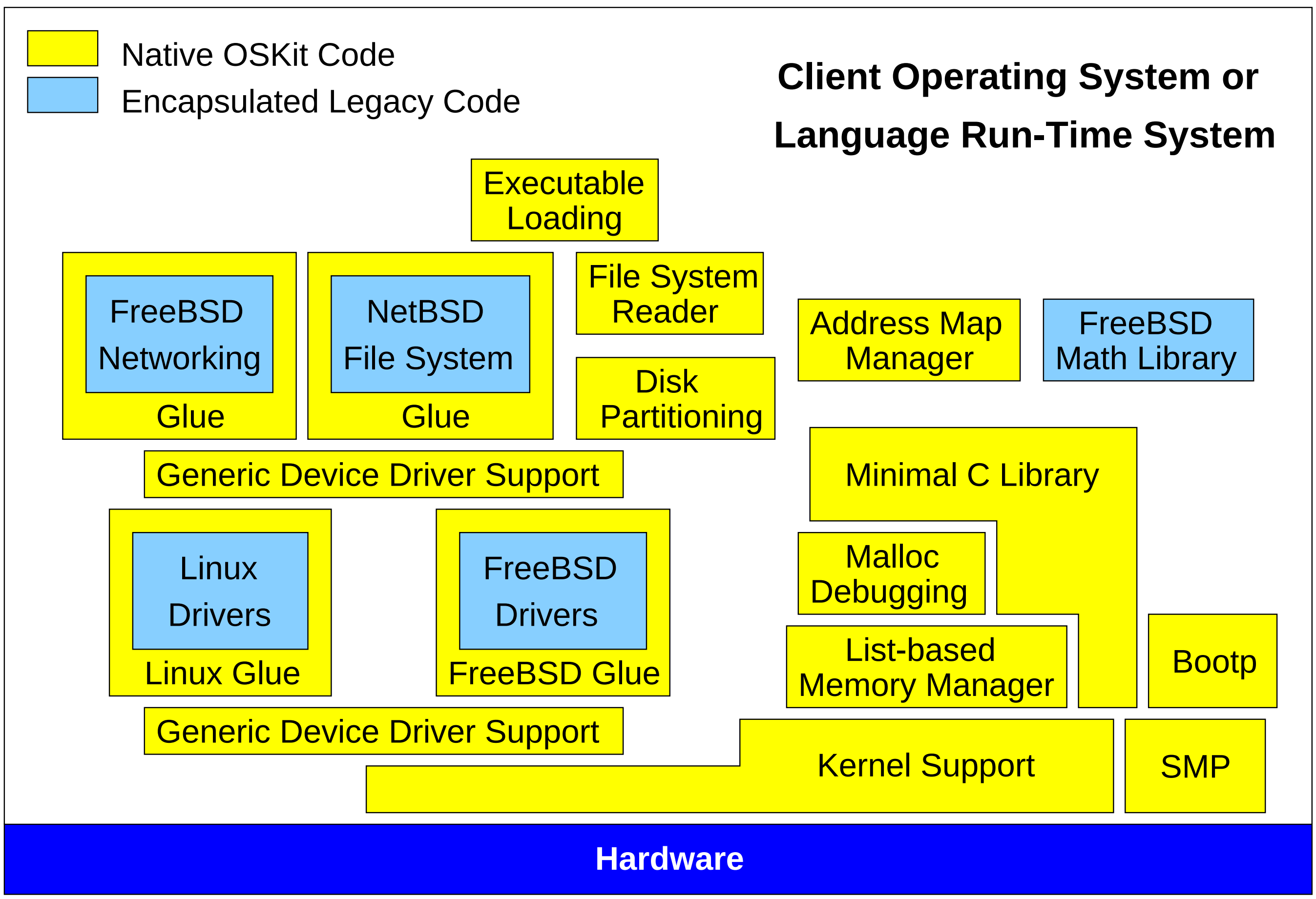
Solution: Lots of Ugly Magic

To avoid changing the imported code, all of these assumptions must be emulated:

- Glue routines translate memory allocation, synchronization, and other primitives.
- Create dummy `proc` structures on entry, destroy them on return.
- Preprocessor magic to ensure namespace cleanliness
  (e.g., `tsleep` → `oskit_freebsd_tsleep`).

It's ugly, but the ugliness is confined!

Current OSKit Components



Efficiency

TCP throughput (Mbit/sec):

	Receiver: Linux	FreeBSD	OSKit
Sender:			
Linux	**72.4**	71.2	71.3
FreeBSD	60.0	**78.6**	78.7
OSKit	56.4	68.3	**68.2**

TCP latency (μsec):

	Server: Linux	FreeBSD	OSKit
Client:			
Linux	**152**	168	180
FreeBSD	168	**197**	210
OSKit	180	210	**222**

Experiences

- Fluke OS
- ML-based OS
- SR-based OS
- Java-based network PC
- ...other users

Fluke

First and most closely bound OSKit customer

Over half of Fluke comes from the OSKit:

- C library
- Debugging
- File systems (as user-mode servers)
- Networking (as user-mode servers)
- Device drivers (in supervisor and user mode)

ML-based OS

ML is a high-level functional language:
Lisp with strong typing and a syntax.

ML/OS created at MIT AI Lab as first external client of the OSKit; took a few months.

Only uses OSKit's bootstrap support and C library; everything else written in ML.

Unique language runtime features that benefit from direct hardware access:

- Stackless implementation

- Continuation-based multithreading

SR-based OS

Parallel/distributed programming language.

SR/OS developed by Greg Benson from U.C. Davis, working at Utah.

Initial implementation took one week; network support took another week.

Uses Arizona's *x*-kernel for networking, but with the OSKit's Linux network drivers.

Java

Developed by Godmar Back at Utah.

Uses Kaffe, a free JVM.

Took 14 hours to get "Hello World" running; JIT compiler took another day; multithreaded Jigsaw web server running in three weeks.

Functionally similar to JavaOS, but uses stable native components instead of rewriting everything in Java.

Status

Fully functional and fairly well documented.

Preliminary release was made earlier this year.

Latest version available at
`http://www.cs.utah.eduh/projects/flux`.

Future Work

- Interoperability with typesafe languages such as Java and ML.

- Direct support for multithreaded code and multithreaded environments

- IDL compiler support for COM interfaces

Conclusion

Key ideas:

- New reusable OS components instead of new OS's
- Encapsulation allows unmodified legacy code to present clean interfaces
- Emphasis on practicality and usability
- Catalyzes OS research and specialized OS development.

Example COM Interface

```
typedef struct blkio {
  struct blkio_ops *ops;
} blkio_t;

struct blkio_ops {
  error_t  (*query)(blkio_t *io,
                    const struct guid *iid,
                    void **out_ihandle);
  unsigned (*addref)(blkio_t *io);
  unsigned (*release)(blkio_t *io);
  unsigned (*getblocksize)(blkio_t *io);
  error_t  (*read)(blkio_t *io, void *buf,
                   off_t offset, size_t amount,
                   size_t *out_actual);
  error_t  (*write)(blkio_t *io, const void *buf,
                    off_t offset, size_t amount,
                    size_t *out_actual);
  error_t  (*getsize)(blkio_t *io, off_t *out_size);
  error_t  (*setsize)(blkio_t *io, off_t new_size);
};

#define BLKIO_IID GUID(0x4aa7df81, 0x7c74, 0x11cf, \
    0xb5, 0x00, 0x08, 0x00, 0x09, 0x53, 0xad, 0xc2)
```

Related Work

- Extensible systems (SPIN, VINO, exo)
- Embedded systems (QNX, VxWorks)
- Object-oriented OS's (Choices, Taligent)

Typical problems:

- New, incompatible OS environments.
- Little reuse of existing OS code.