The Flux Project

Jay Lepreau

University of Utah

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

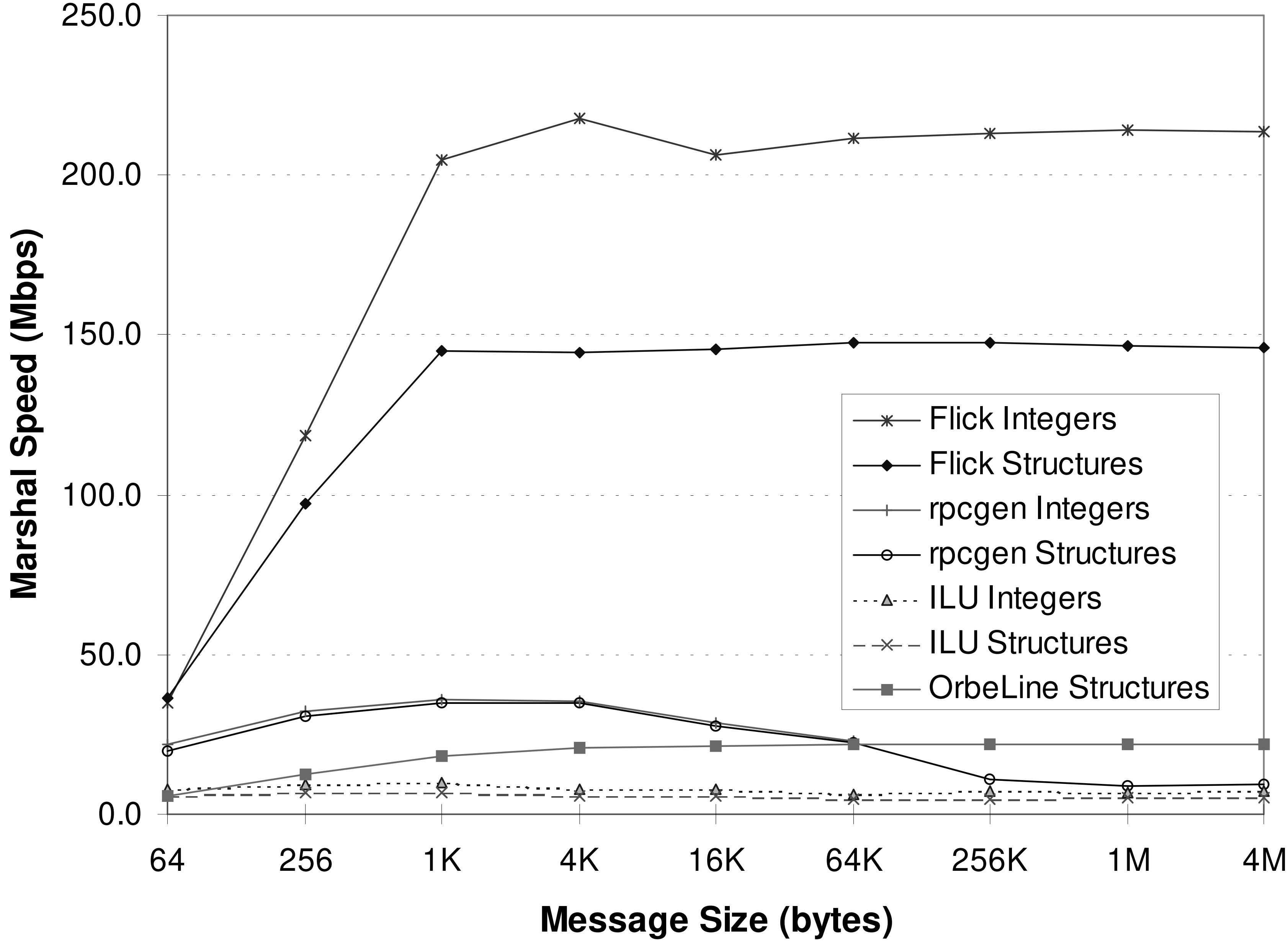
December 11, 1996

Major Projects

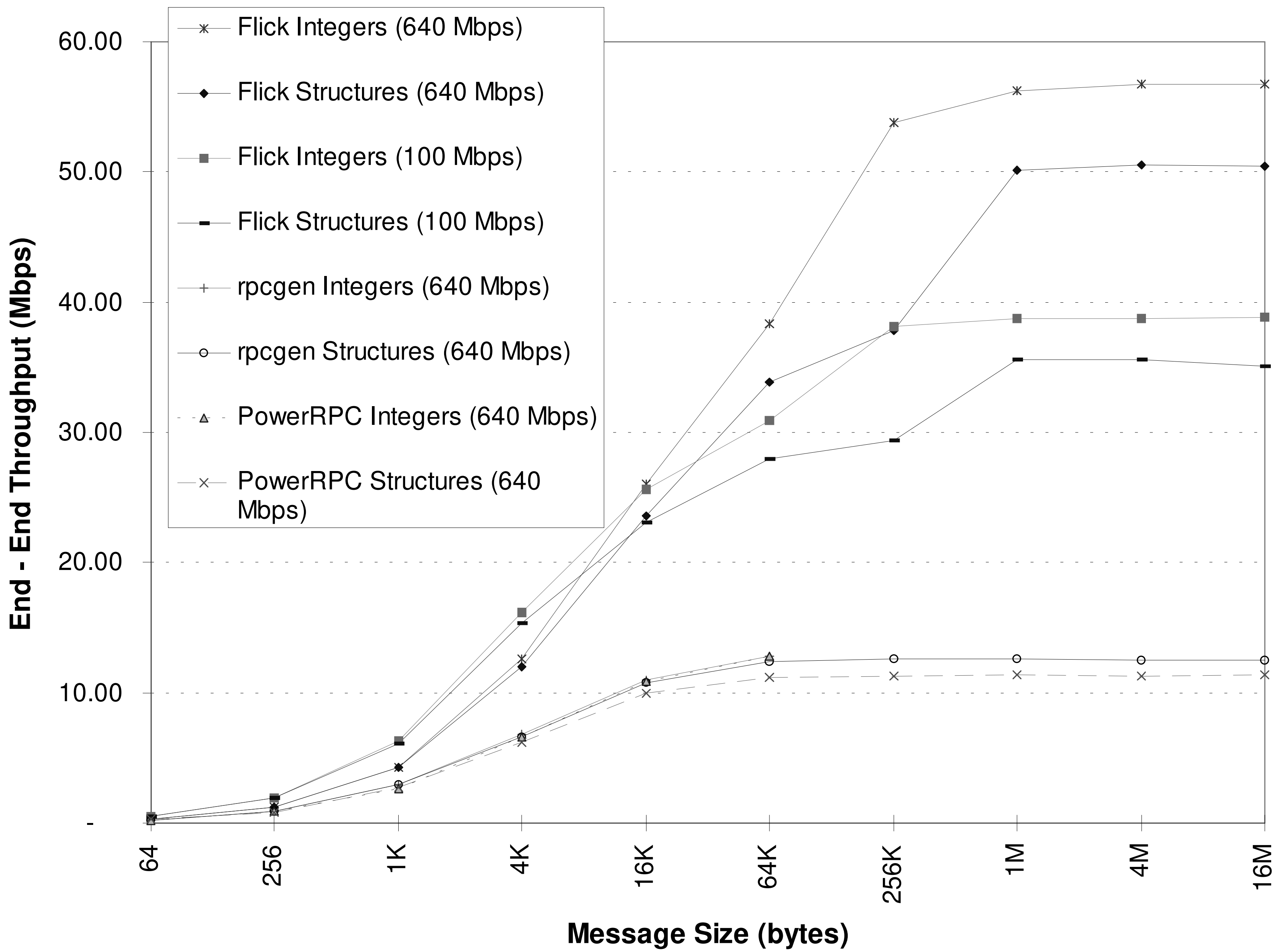
- Fluke: model, kernel, and operating system
- High confidence: resource management, high assurance, information security
- Flick: flexible, optimizing IDL compiler supporting interface annotation
- OS Toolkit: modularized infrastructure for OS's

Digression:
Flick - Optimizing IDL Compiler

- (talk off of Flick picture)
- IDL: interface defintion language
- Presentation layer (data marshaling, encoding, conversion) becoming bottleneck as transport overhead plummets and OS gets clever
- Apply traditional compiler techniques
- Use powerful intermiedate representations
- ....



250.0
200.0
150.0
100.0
50.0
0.0
Marshal Speed (Mbps)
64
256
1K
4K
16K
64K
256K
1M
4M
Message Size (bytes)
Flick Integers
Flick Structures
rpcgen Integers
rpcgen Structures
ILU Integers
ILU Structures
OrbeLine Structures


Flick Integers (640 Mbps)
Flick Structures (640 Mbps)
Flick Integers (100 Mbps)
Flick Structures (100 Mbps)
rpcgen Integers (640 Mbps)
rpcgen Structures (640 Mbps)
PowerRPC Integers (640 Mbps)
PowerRPC Structures (640 Mbps)
60.00
50.00
40.00
30.00
20.00
10.00
-
End - End Throughput (Mbps)
64
256
1K
4K
16K
64K
256K
1M
4M
16M
Message Size (bytes)

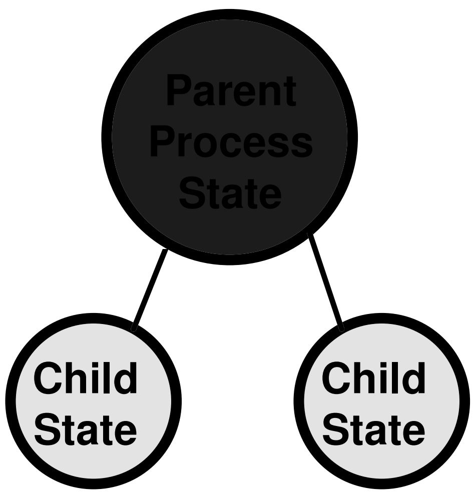
Fluke Model: Motivations

- Modularity, extensibility, maintability, security, assurance e.g, DOCT project and local OS “security”
- Virtual machine model!
- Wrap services around applications
- Hierarchy enabled: provides isolation and composition

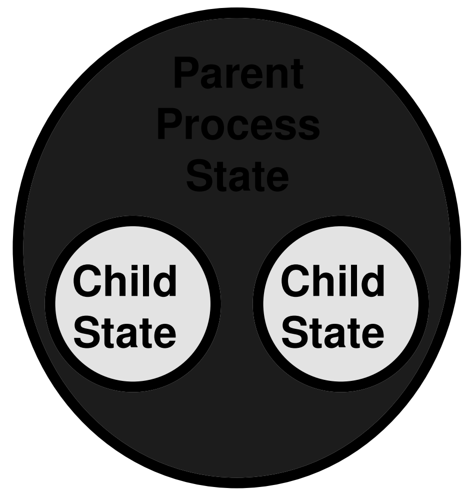
The Nested Process Model

Child process is **encapsulated** in its parent.

Traditional Process Model



Nested Process Model



Parent has **complete control** over the child.

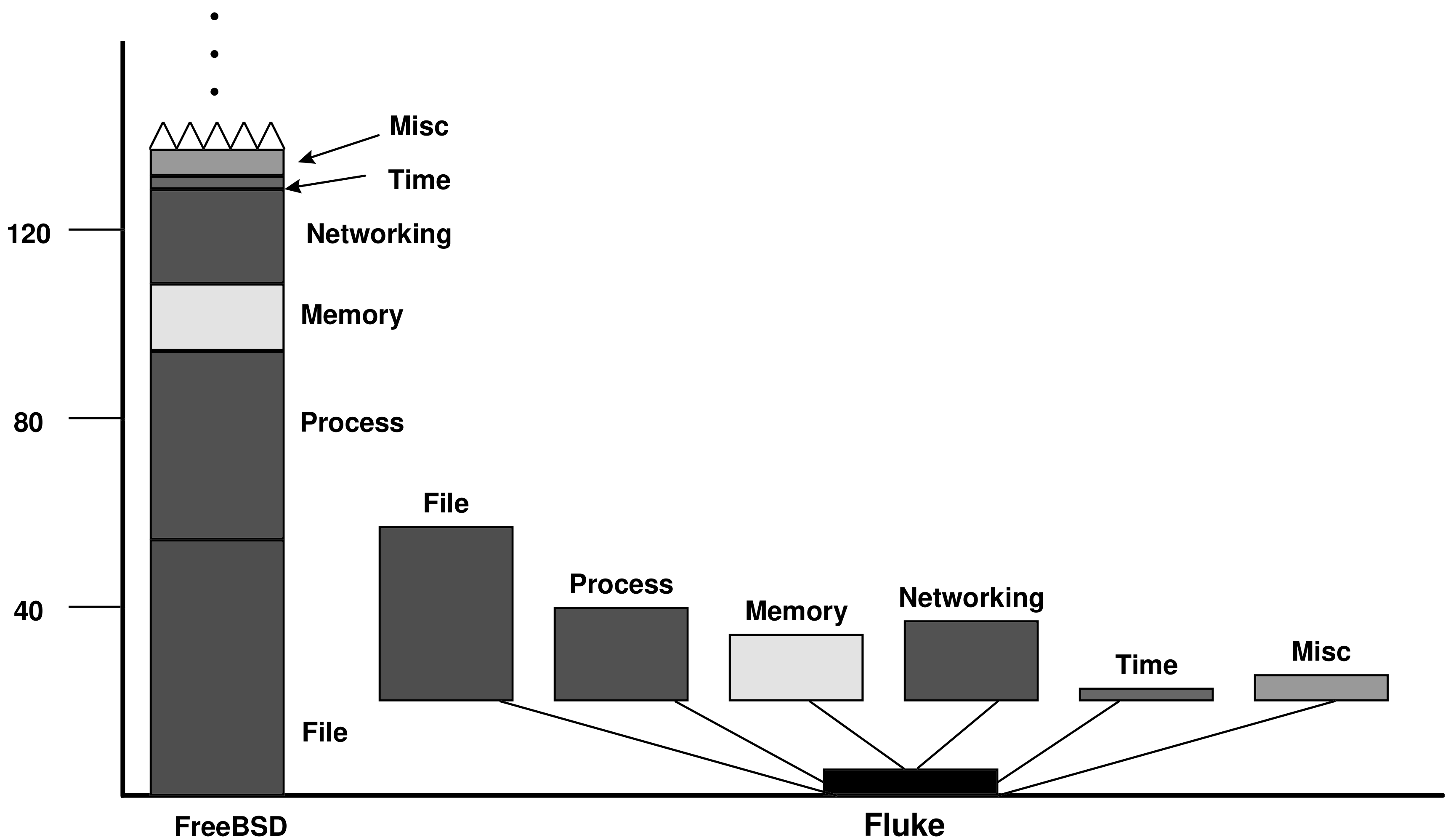
Nested Process Model and Protection

- **Provides a new way to *use* protection domains**
- **Can use various protection *mechanisms*:**
  - **Hardware (current)**
  - **Type-safe languages or software fault isolation (planned)**
  - **Proof-carrying code (planned)**

Structured Interfaces

- Three sets:
  - Low-level: irrelevant to external control needs
  - “Parent -- the universal service service” - sole access to higher level services
  - Higher level services : traditional memory, file, process, network, ..., services
- Provides control hierarchy: single point of control over service access
- Modular control (memory vs. filesys vs. networking ....)

Interface Size



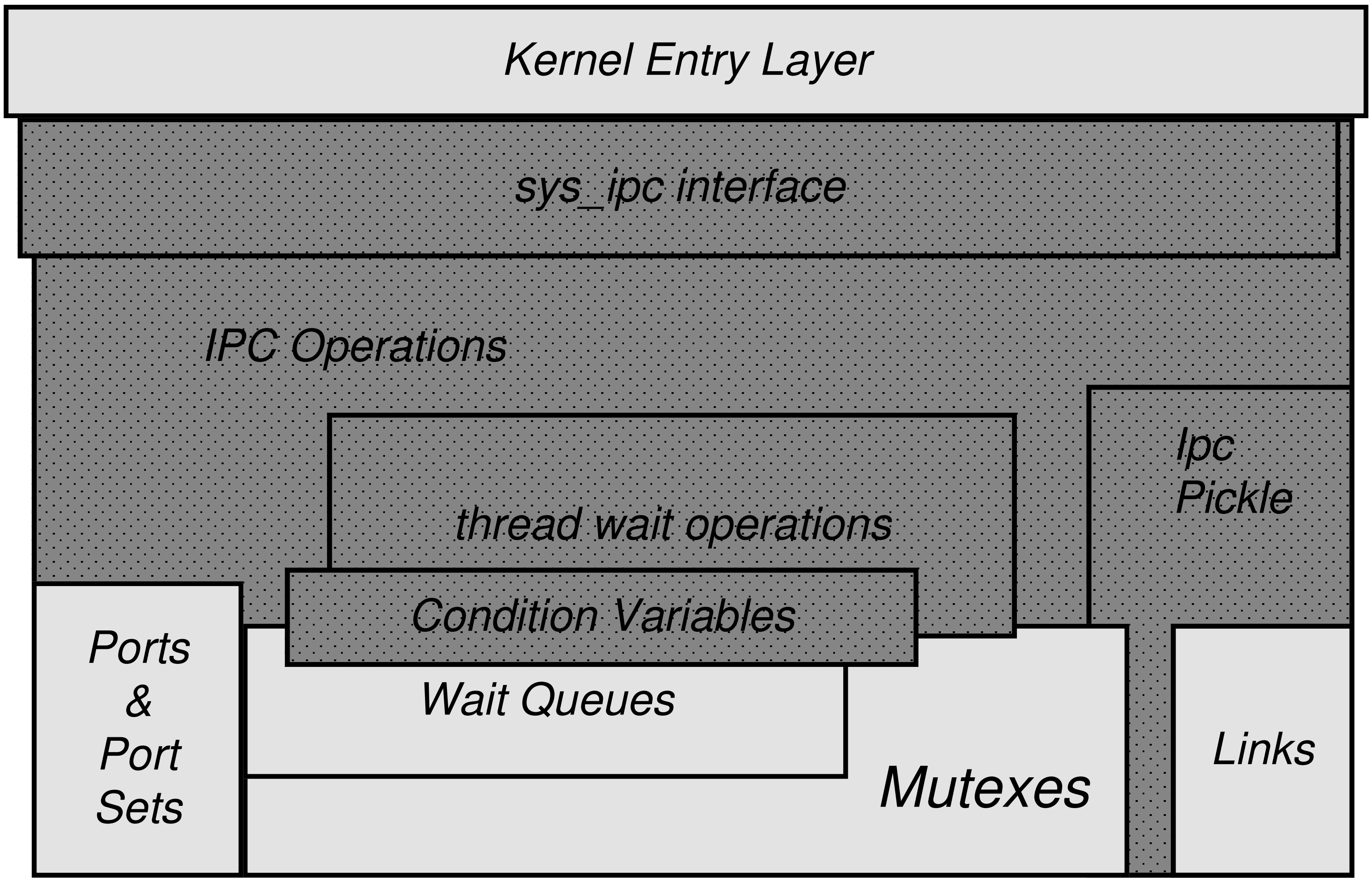
Assurance

- Have a *model* with specific properties
- Have *structure*: design, interfaces, implementation
- *Documented*: model, properties, interfaces
- Kernel implementation is *layered, small* (90k)
- Applying *formal methods*

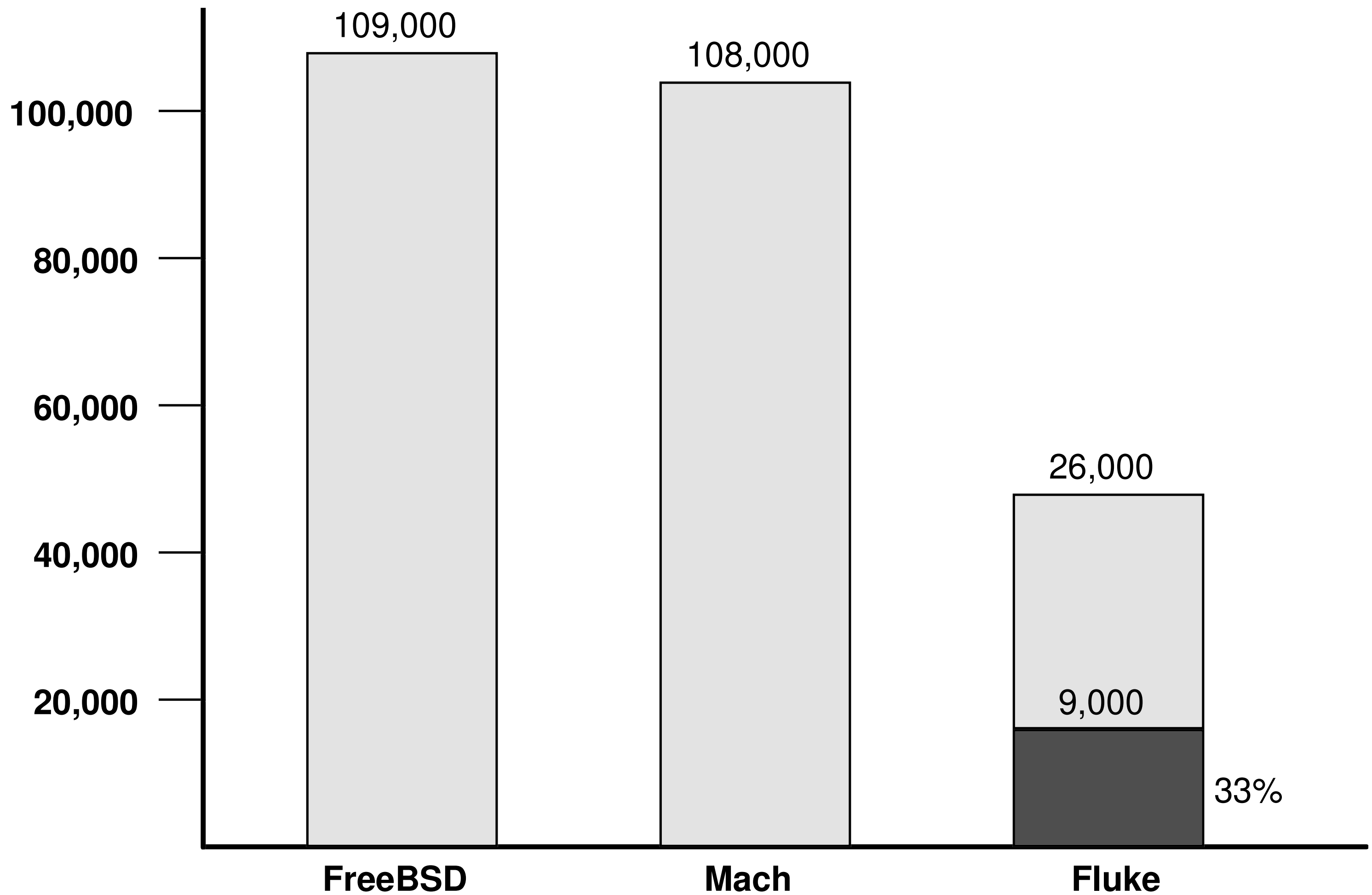
The Big Picture

- Fluke IPC path has interesting and tricky aspects
- Using formal methods, model the IPC path and verify the model has various properties.   E.g.
  - No deadlock
  - That IPC can be cancelled at arbitrary points, without leaving involved threads deadlocked or lost
- Increases assurance; maybe finds bugs

Modeled IPC Structure



Code Size and Percentage Verified



Providing QoS and Adaptation

- Services can wrap themselves around apps and provide resource mgmt/QoS
- Respond to QoS requests from “smart” applications
- Provide to existing or naive appplications transparently
- “Drag and drop” executing app into a resource manager
- Virtual machine model is fundamental: extend from multi nodes through local system

Nested Processes on Other OS's?

- Range of control depending on how much support from base OS
  - New kernel (eg ours): complete
  - Implement on top of hi-speed ukernel (eg L4): probably complete
  - Integrate into kernel: provide Fluke *and* conventional processes (FleeBSD, Flinux, FliNT, SPIN)
  - Implement on top of hardware abstraction kernel (eg Exokernel) or traditional OS: provide unenforced Fluke semantics
- Many possibilities, especially if not concerned with errant or malicious apps.

Fluke Collaborators

- NSA R23 research (ongoing):
  - High trust system in/on Fluke
  - Formal Security Policy Model
  - Formal Top Level Specs for kernel, virtual memory mgr, process mgr
  - Formal methods:
    - On-going exploration
    - Apply to OS Toolkit (probable)
- Secure Computing Corporation (probable):
  - Refine PVS composition framework and apply to Fluke nesters
  - Parts of FSPM and FTLS
- CMU Fox project (planned): test new proof-carrying code technologies in Fluke, and possibly hi-level programming languages

Collaborators (cont'd)

- OGI (planned): test specialization technologies on Fluke layers
- CMU RT (planned): test real-time scheduling policies on Fluke
- DEC SRC (planned): develop resource alloc. & scheduling algorithms on Fluke
- Sandia (ongoing): add high trust to Fluke distributed IPC
- CLI (potential): major project involving formal specification, modeling, verification, and code synthesis of Fluke

Conclusion

- Carefully structured interfaces provide modular control easily
- Strong and flexible management of all resources
- High assurance and assurability
- Performance will come through our own work and leveraging of recent research in varied protection domain mechanisms
- Nesting/virtual machine model useful for QoS management
- A *model* and *API*, not an implementation: can provide in other OS's