

• • • • • • • • • •

EROS: A Fast Capability System



Jonathan S. Shapiro
Jonathan M. Smith
David J. Farber

• • • • • • • • • •

EROS Goals

- Priorities
 1. Security & Integrity
 2. High availability
 3. Fault Tolerance
 4. Evolvability
 5. Performance
- This ordering has architectural and performance implications.

How is EROS Different?

- Pure capability system
- Transparently persistent
- Recovers rapidly (< 30 seconds)
- Thoroughly paranoid implementation
 - Consistency checks to prevent snapshot of bad states
 - Implementation tries to be “fail fast”
 - Think: kernel *always* compiled for debugging
- Some emphasis on discretionary security

What is a Capability?

- A capability is an (object, permissions) pair

- Unforgeable, so a basis for protection
 - Transferable, so a basis for authorization

- This can be generalized to (object, type)

- An object version number makes reallocation simple.

- The resulting representation is straightforward.

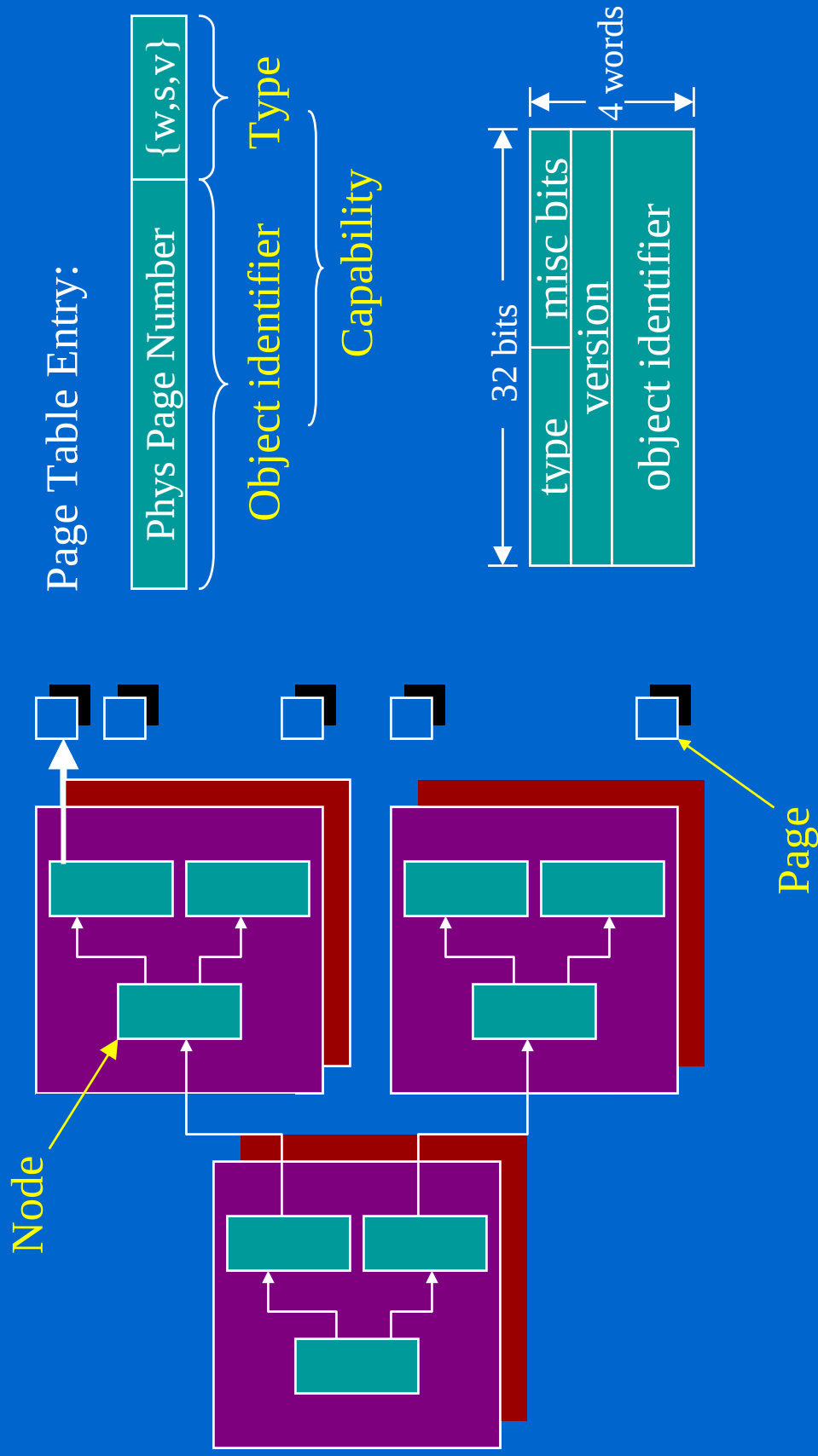
(spaceroot, rw-space)
(spaceroot, node)



Comparison to Other Capability Systems

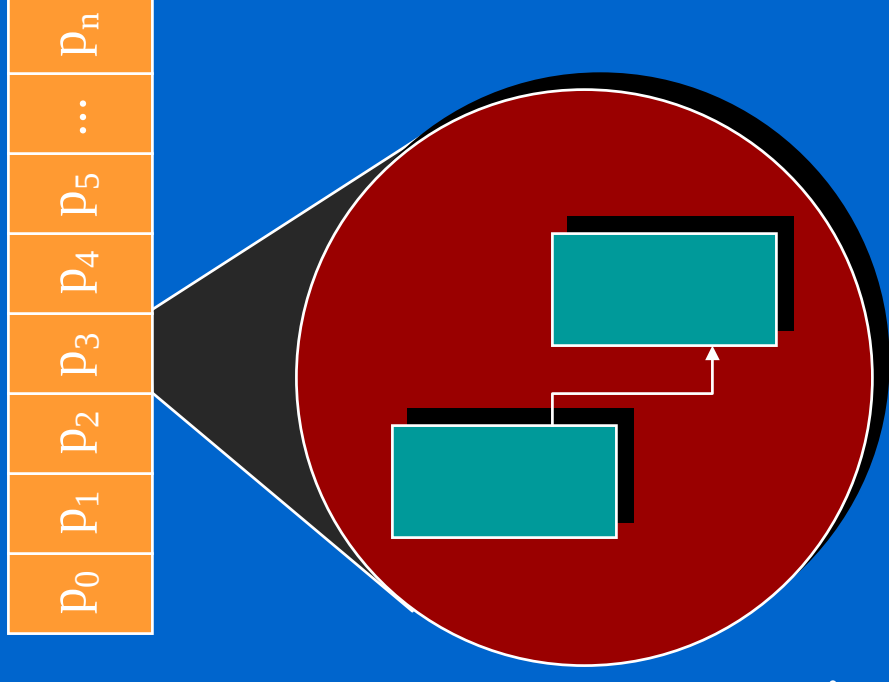
<i>System</i>	<i>HW/SW</i>	<i>Store</i>	<i>Persist</i>	<i>Cap Prot</i>	<i>Mem Model</i>	<i>IPC</i>
<i>Cal TSS</i>	SW	File	Explicit	Partition	Byte Segments	Buffered, Unbounded
<i>CAP</i>	HW	Object	Explicit	Partition	Byte Segments	Prot. Procedure Call
<i>Hydra</i>	Mixed	File	Explicit	Partition	Byte Segments	Prot. Procedure Call
<i>S/38 (AS/400)</i>	HW + Compiler	Object	Transparent	Tagging	Byte Segments	Prot. Procedure Call
<i>i432</i>	HW	Object	Explicit	Partition	Byte Segments	Prot. Procedure Call
<i>Mach</i>	SW	App. Defined	Explicit	Partition	Page Regions	Buffered, Unbounded
<i>Amoeba</i>	SW	Object	Explicit	Sparsity	Page Regions	Buffered, Bounded
<i>KeyKOS/EROS</i>	SW	Object	Transparent	Partition	Pages + Nodes	Unbuffered, Bounded

Memory Mapping



Processes

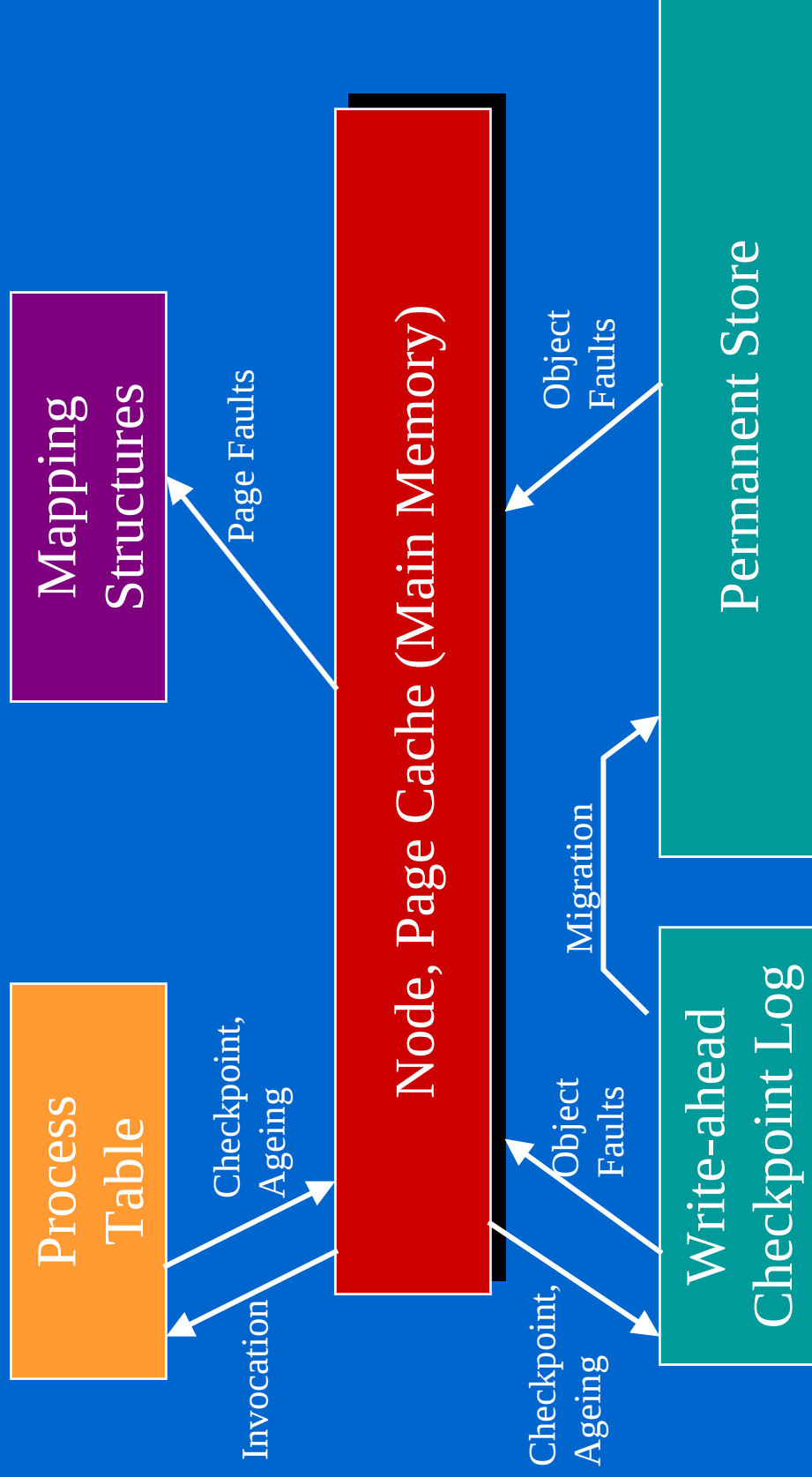
- Processes have user-mode machine state plus supervisor-implemented capability registers.
- Kernel implements a machine-specific process table
 - Used to *cache* active processes (c.f. Cache Kernel, Fluke).
 - Fast-path IPC uses this structure.
 - General capability invocation path uses both representations.
- Process state is recorded in nodes.



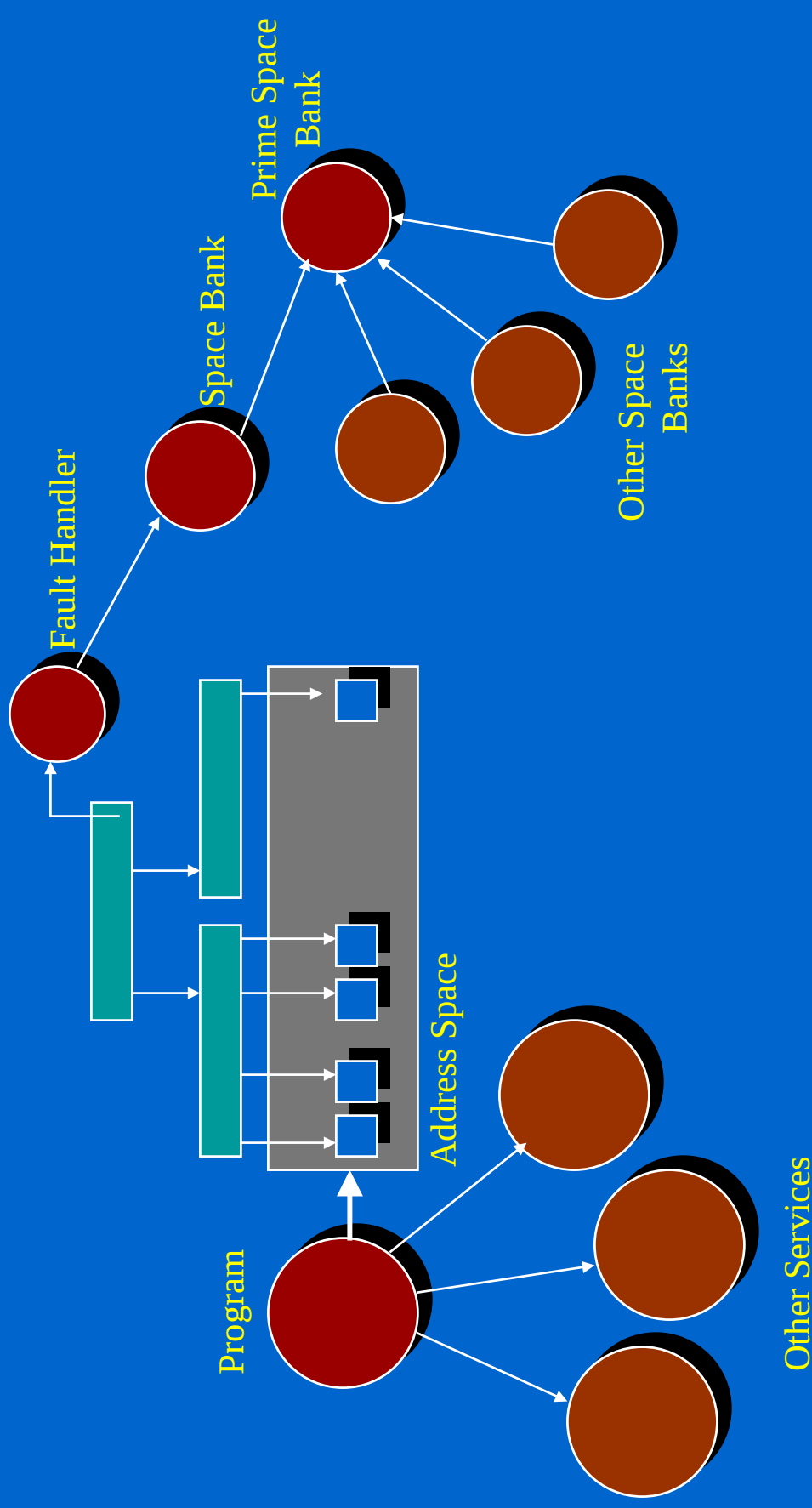
Properties of this Design

- Everything (all resources) is named by a uniform naming mechanism: capabilities.
- The protection state of the system can be directly realized by the hardware.
- All user-visible state is stored in pages and nodes
 - This plus “run list” is all you need to define a recoverable system state.
- Object reference is a protected operation
 - Conventional operating system services can therefore be implemented outside the kernel.

Transparent, Global Persistence



Typical Program Structure

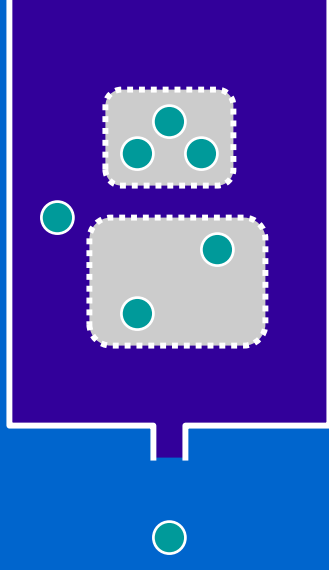


User-Mode Services

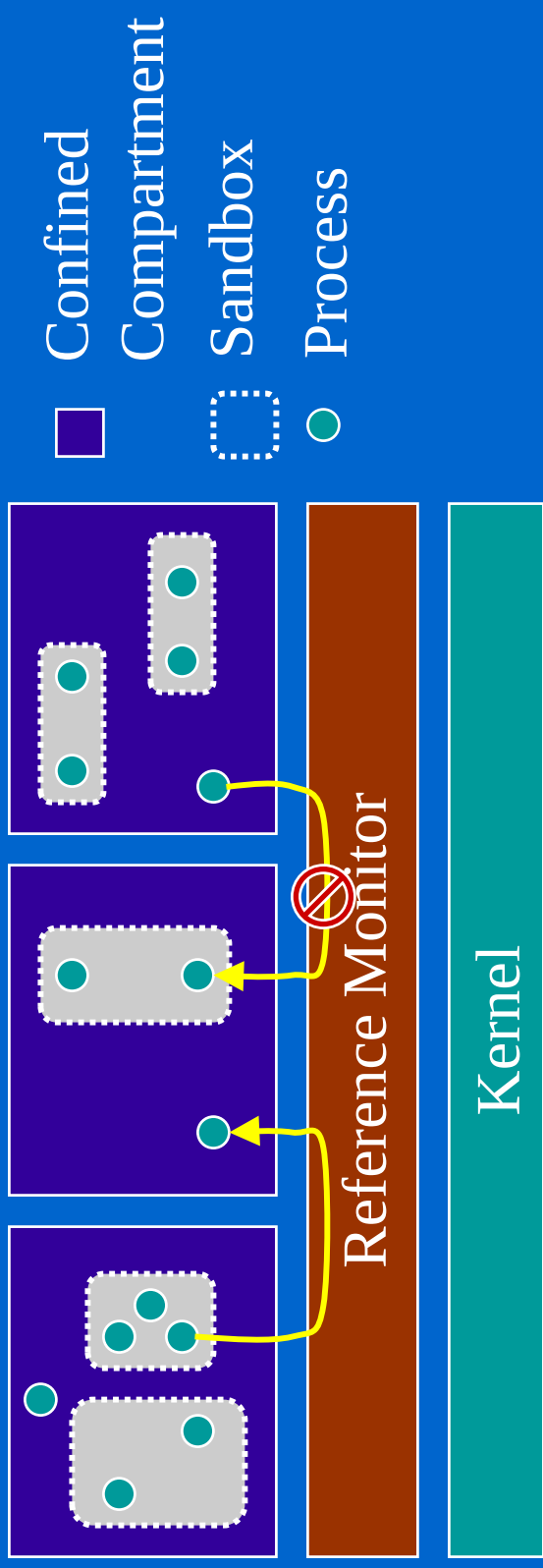
- Memory fault handlers
- Storage allocator (space bank)
- Files and Directories
- Pipes
- Constructor (confinement implementation)
- Reference monitor

Confinement (Lampson '73)

- Initial Conditions:
 - Client has exclusive access to service.
 - Confined entity has no unauthorized channels.
- Confined entity can be a complex subsystem.
- Client therefore completely controls communication.

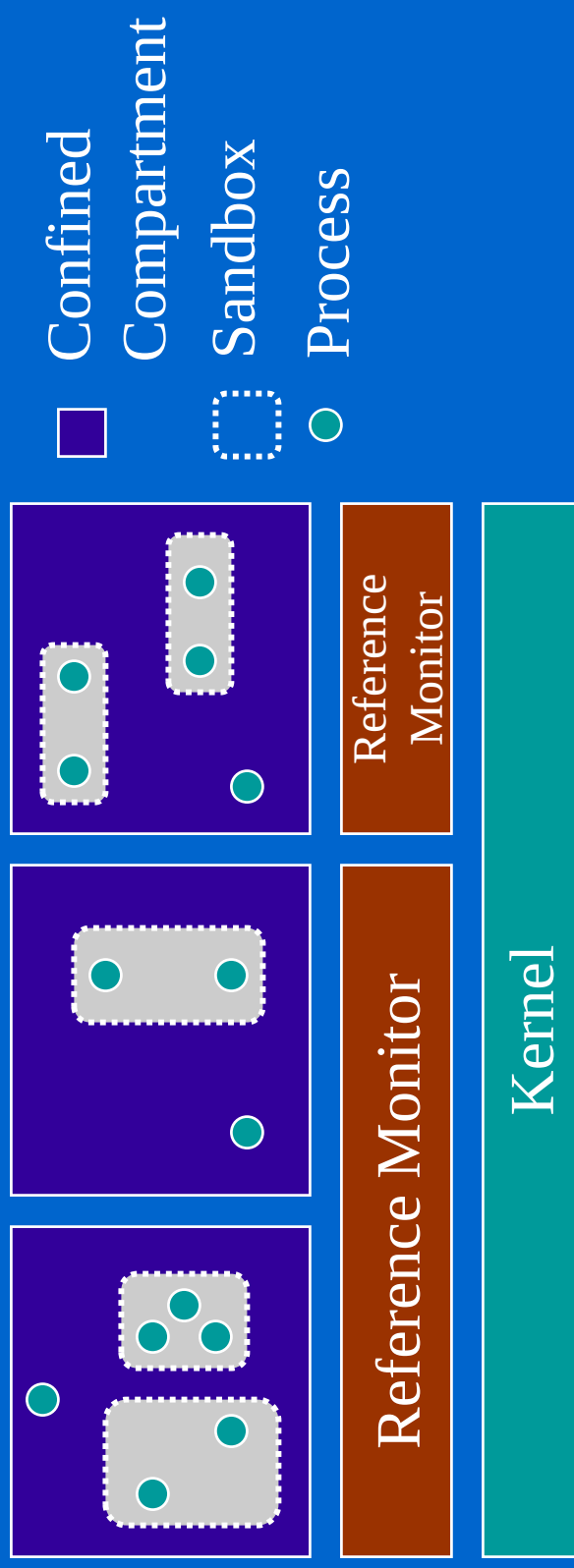


Reference Monitor w/Confinement



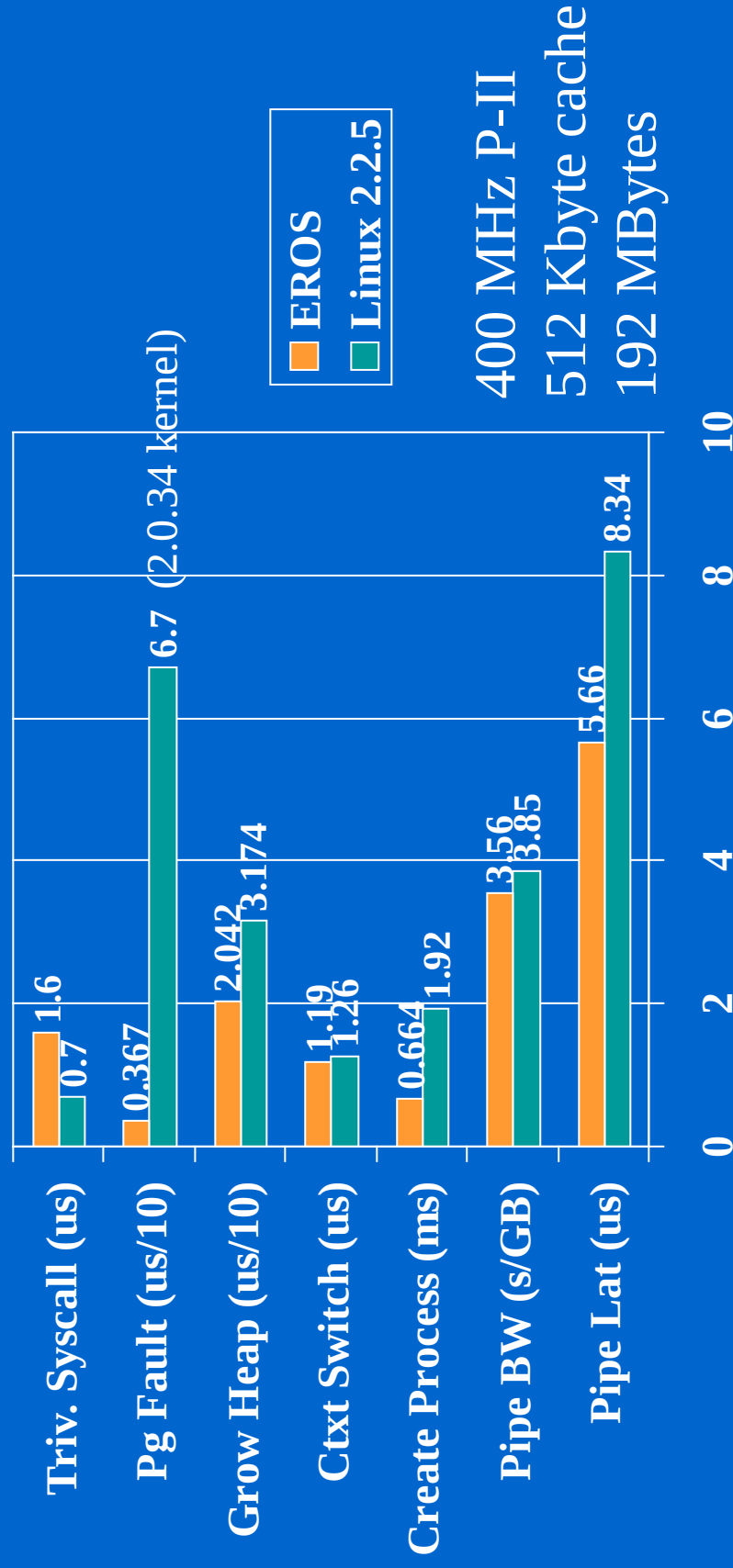
- Reference monitor knows object semantics.
- Interposes transparent forwarding objects where appropriate.
- Can be evolved as new object types are introduced.

Disjoint Reference Monitors



- Multiple reference monitors can securely manage disjoint logical systems on the same hardware.
- **Remote Hot Standby**

Microbenchmark Performance



Note: 2.2.x kernel introduced a temporary performance bug in page fault handling.

IPC Semantics: EROS and L4

<i>Property</i>	<i>L4</i>	<i>EROS</i>	<i>Issue</i>
<i>Registers saved</i>	Most	All	Covert Channel
<i>Payload</i>	31 x 4M	1 x 64k	Resource Exhaustion,
<i>Target name</i>	Thread ID	Capability	Encapsulation
<i>Authority Xfer</i>	Permissions for Pages	Capabilities	Access Control, Channel audits
<i>Atomicity</i>	No: Preemption, Page Faults	Yes	Bounding resources and time
<i>Missing page strategy</i>	Timeout, then discard	Discard	Covert Channel

IPC Semantics: EROS and L4

<i>Property</i>	<i>L4</i>	<i>EROS</i>	<i>Issue</i>
<i>Registers saved</i>	Most	All	Covert Channel
<i>Payload</i>	31 x 4M	1 x 64k	Resource Exhaustion,
<i>Target name</i>	Thread ID	Capability	Encapsulation
<i>Authority Xfer</i>	Permissions for Pages	Capabilities	Access Control, Channel audits
<i>Atomicity</i>	No: Preemption, Page Faults	Yes	Bounding resources and time
<i>Missing page strategy</i>	Timeout, then discard	Discard	Covert Channel
<i>Latency</i>	454 cycles	640 cycles	Large spaces

Conclusions

- It *appears* possible to build a high-performance capability system.
- Persistence greatly simplifies some components, and therefore assurance.
- Capabilities provide a sufficient primitive protection mechanism to implement other security policies at user level.
- Using performance as the only evaluation criterion can obscure important issues, including security.

Research Questions

- How can a capability system be distributed securely and efficiently?
- How is multiparty administration and just-in-time software provisioning to be managed?
- How can assurance be achieved using an open development model?
- Compatibility and (r)evolution
- System structure – design and architecture
- Language integration: how to do it successfully

The Future: *Cougar*

- IBM Research has started the *Cougar* project to investigate secure, high-performance underpinnings for pervasive devices and their supporting servers.
- *Cougar* will be capability based, and will borrow from both the L4 and the EROS architectures.