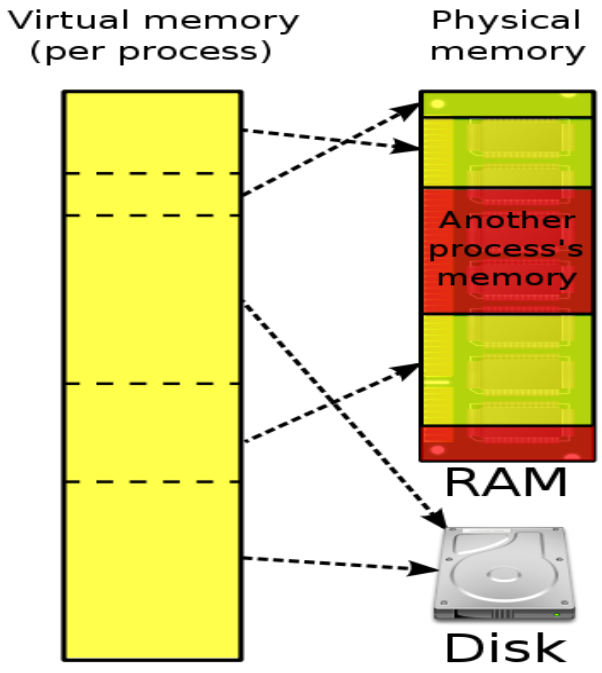
VIRTUAL MEMORY

CS6410
9/22/2011
DAN DENG

Virtual Memory

- Separates
  - Logical memory as perceived by users
  - Physical memory in the system
- Provides for
  - Larger programs
  - Process isolation
  - Memory sharing



Comparing Asbestos and Mach VM

- Similarities
  - Underlying IPC mechanism
- Mach VM
  - Simplify support for diverse architectures
  - Efficiently support memory sharing
- Asbestos
  - Applications specify protection policies
  - Kernel enforces policies to isolate private data

Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures

Richard Rashid - Chief Executive Officer, Microsoft Research (joined Microsoft in 1991)
Lead developer of Mach

Avadis Tevanian – Chief Software Technology Officer (2003-2006) and Senior VP of Software Engineering (1997-2003) at Apple Inc., Witness for the United States DoJ testifying against Microsoft in *United States v. Microsoft*,

William Bolosky (CMU) – Microsoft Research
David Black (CMU)
Michael Young (CMU)
David Golub (CMU)
Robert Baron (CMU)
Jonathan Chew (CMU)

Mach Takeaways

- Hardware-independent virtual memory (VM) is possible
  - Hardware-dependent structures can be kept small
  - Mach can be easily adapted to a variety of architectures

- Flaws
  - Incomplete Evaluation – microbenchmarks
  - Use of IPC for all tasks turned out costly
  - Memory remapping between programs became more expensive

Context

- Successor to Accent
  - Accent pioneered many novel OS concepts
  - Accent limited by inability to execute UNIX applications
  - Accent had strong ties to a single hardware architecture
    - Explosion of hardware architectures and memory structures
    - Many of them were multiprocessor systems
- Solution
  - Mach – cleanly defined, UNIX-based, highly portable OS

Design Principles

- Support for diverse architectures
- Simplified kernel structure
- Distributed operation
- Integrated memory management and IPC
- Heterogeneous system support

Approach

- Simple, extensible kernel
  - As little as possible inside the kernel
  - What is in the kernel must be powerful
- Concentrate on communication facilities
  - IPC handles all kernel requests and data movement among tasks
  - System-wide protection by protecting the IPC mechanism
- Key to efficiency
  - Integration of memory management and IPC

Abstractions

- Task
  - Unit of resource allocation
- Thread
  - Unit of execution
- Port
  - Communication channel for IPC
- Message
  - Collection of objects used in communication
- Memory Object
  - Collection of data provided and managed by a task

Improving Portability

- Perform software memory management
  - Manage all important virtual memory information in software
  - Communicate with hardware using a well-defined interface
- Machine dependent code
  - Implement the bare minimum to allow the kernel to run
  - Export a clean interface to the software

Implementation

- Split hardware-independent and hardware-dependent memory management
- Hardware-independent:
  - Resident Page Table
  - Address Map
  - Memory Object
- Hardware-dependent:
  - Pmap
- Hardware-independent parts are the driving force

Hardware-Independent

- Resident Page table
  - Indexed by physical page number
  - Entries are linked into
    - Memory object list
    - Memory allocation queue
    - Object/offset hash bucket
- Address map
  - Sorted, doubly-linked list of entries
  - Maps a range of virtual addresses to area of a memory object
- Memory Object
  - Resembles a UNIX file
  - Managed by a pager

Hardware-Dependent

- Pmaps – management of physical address maps
  - VAX – pmap corresponds to VAX page table
  - IBM RT PC – pmap corresponds to allocated segment registers
- Pmaps are small
  - Code is confined to a single module
  - Interface with Mach is relatively small
  - Modifying pmap requires little knowledge of Mach
  - Pmap does not have to keep track of all valid mappings

VM-IPC Integration

- Memory management based on use of memory objects, which are represented by ports
- IPC in memory management
  - Memory objects may reside on remote systems
  - Kernel caches the contents of memory objects in local memory
- Memory-management use in IPC
  - Mach passes messages by moving pointers to shared memory objects
  - Mach uses VM remapping to transfer large messages

Adapting Mach

- Code for VM originally ran on VAX machines
- IBM RT PC
  - Approx. 3 months for pmap module
- Sequent Balance
  - 5 weeks – bootable system
- Sun 3, Encore MultiMAX

Performance

in the Intel Hypercube.

As yet, Mach, like UNIX, has been ported only to multiprocessors with uniform shared memory. Mach does, however, possess mechanisms unavailable in UNIX for integrating more loosely coupled computing systems. An important way in which Mach differs from previous systems is that it has integrated memory management and communication. In a tightly coupled multiprocessor, Mach implements efficient message passing through the use of memory management "tricks" which allow lazy-evaluation of by-value data transmission. It is likewise possible to implement shared

differs from these previous systems in that it provides sophisticated virtual memory features without being tied to a specific hardware base. Moreover, Mach's virtual memory mechanisms can be used either within a multiprocessor or extended transparently into a distributed environment.

9. Conclusion

An intimate relationship between memory architecture and software made sense when each hardware box was expected to run its own manufacturer's proprietary operating system. As the computer science community moves toward UNIX-

Table 7.1
The cost of various measures of virtual memory performance for Mach, ACIS 4.2a, SunOS 3.2, and 4.3bsd UNIX

Table 7.2
Cost of compiling the entire Mach kernel and a set of 13C programs on a VAX 8650 with 36 MB of memory under both Mach and 4.3bsd UNIX

Perspective

- Achieved Goals
  - Sophisticated, hardware-independent VM system possible
  - Good micro-benchmark performance
- Ongoing research
  - Later evaluation demonstrated poor performance

Labels and Event Processes in the Asbestos Operating System

Petros Efstathopoulos – Researcher at Symantec, works on OS kernel Projects, PhD from UCLA in 2008

Maxwell Krohn– Founder and Chief Scientist of OkCupid.com (built on OKWS), Creator SFSLite and OK Web Server, PhD from MIT in 2008

Steve VanDeBogart – PhD from UCLA in 2009

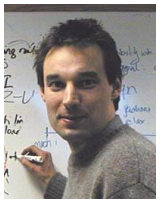
Cliff Frey – CS Meng from MIT in 2005, Meraki

David Ziegler – CS Meng from MIT in 2005

Labels and Event Processes in the Asbestos Operating System

Eddie Kohler – CS Professor at Harvard. Creator of Click Modular Router.

David Mazieres – CS Professor at Standard University, Secure Systems Group. Creator of SFS and libasync.



Francs Kaashoek – CS Professor at MIT, member of CSAIL, Parallel and Distributed Operating Systems group. Creator of Chord.

Robert Morris – CS Professor at MIT, member of CSAIL, Parallel and Distributed Operating Systems group. Creator of Chord.

Asbestos Takeaways

- Asbestos operating system
  - Applications use labels to define their own isolation policies
  - Kernel enforces isolation policies
  - Isolate processes by tracking and limiting the flow of information
- OK web server on Asbestos
  - Performs comparably to Apache
  - Provides better security properties than Apache

- Lessons/Flaws
  - Label checking scales linearly with number of compartments
  - Kernel must be invoked to update send labels

Context

- Problem
  - Web servers and networked systems are insecure
    - Lack of good primitives for security
    - Monolithic code with many privileges
    - Exploits: XSS, SQL Injection, Buffer Overrun
  - Lack of isolation of user data
    - Breaches divulge private information on a massive scale
    - TJX – 45 **million** stolen CC numbers in 2007
    - Sony PSN – $171 **million** in stolen accounts

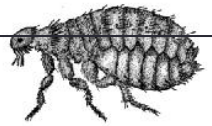
Example

POST

Alice

Bob

Apache



Write()

Kernel

/alice_private_data.dat

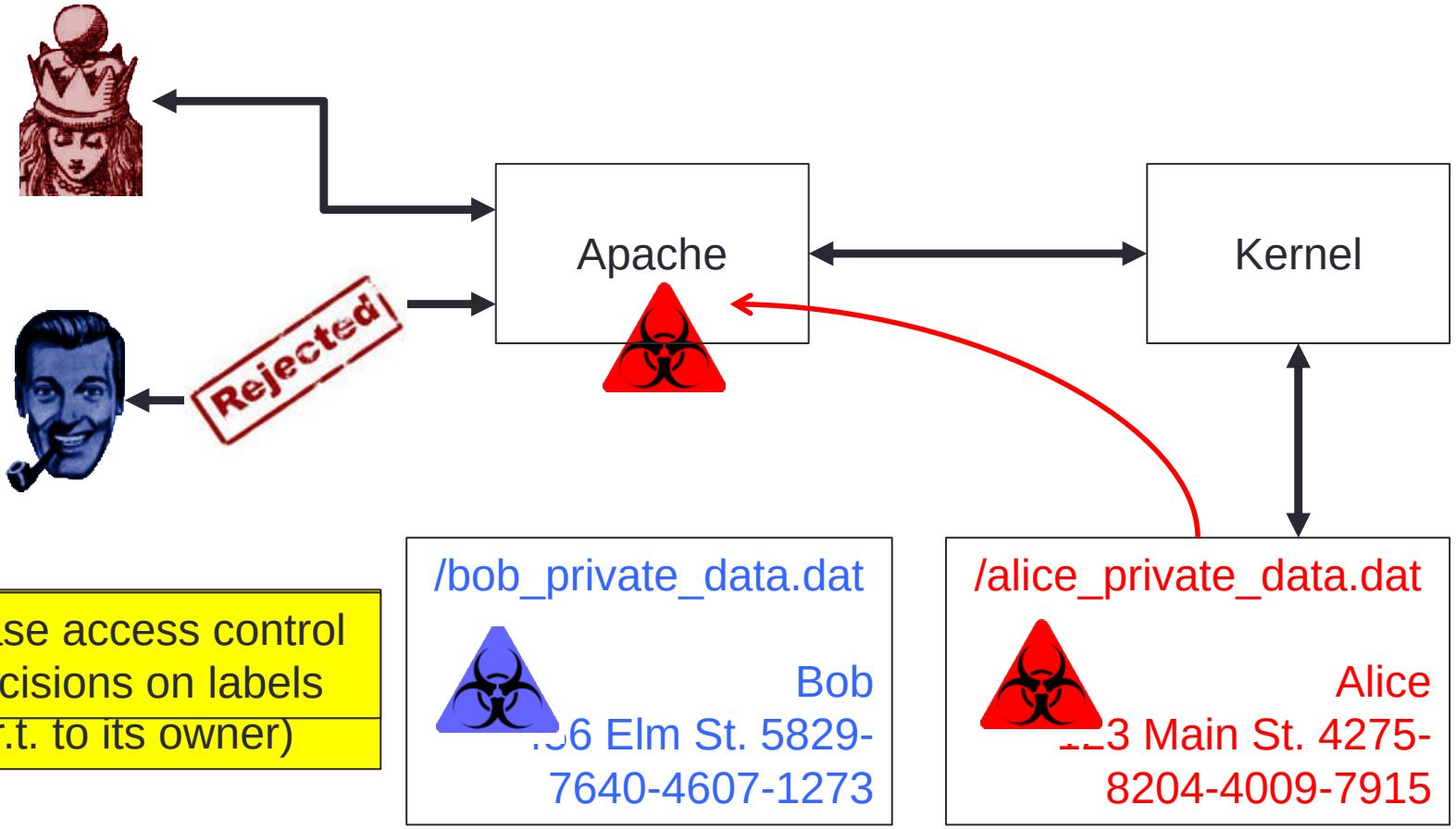
Alice
123 Main St. 4275-8204-4009-7915

By exploiting a software flaw, Bob can access Alice's data

Goal

- Bob can’t access Alice's data without Alice's permission
  - Alice’s and Bob’s data are isolated
- Complications
  - Bugs in the applications
  - Alice's data may travel through several processes
  - There may be hundreds of thousands of users
- To isolate, must prevent inappropriate data flow

Information Flow Control



Asbestos Goals

- Large-scale
  - Changing population of 100,000s of users
- Efficient
- Unprivileged
  - Minimum privilege required to do its job
- Isolated
  - One user can’t gain inappropriate access to other users’ data
- Isolation policy
  - Application-defined, OS-enforced

Compared to MAC

- Mandatory access control systems
  - Inflexible, administrator-established policies
  - Fixed number of levels and compartments
  - Central authority, no privilege delegation
- Asbestos
  - Applications can define flexible policies
  - Almost unlimited* number of compartments
  - Any application can dynamically create compartments

Asbestos Overview

- Processes communicate using ports
- Asbestos labels
  - Each process has a send label and a receive label
  - Labels determine where messages can be sent
  - Contamination of receiver updated
- Event processes
  - Efficiently support/isolate many concurrent users

Asbestos Labels

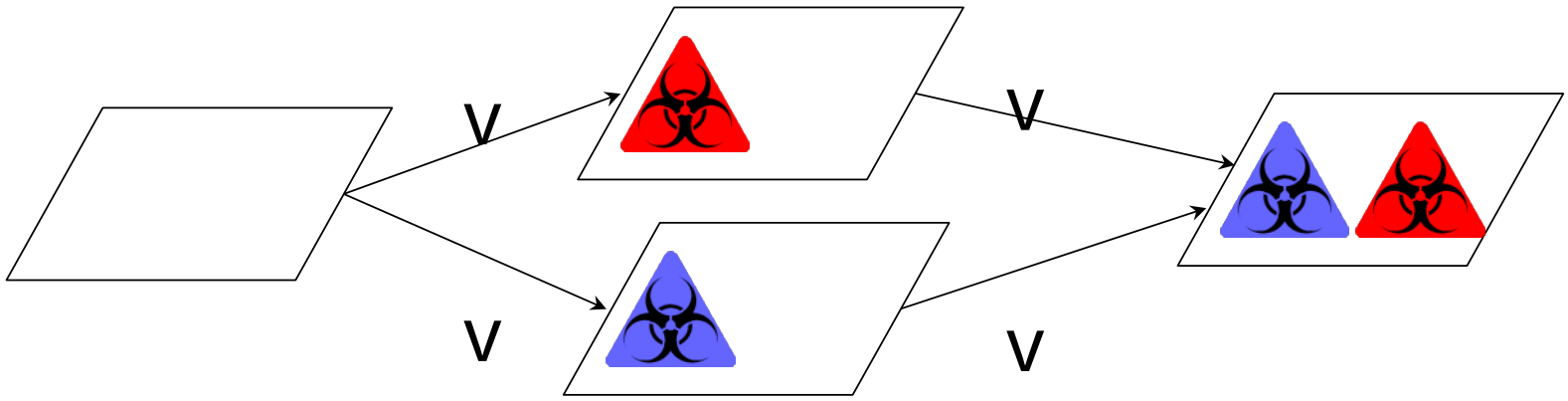
- Applications have send and receive labels
  - Labels consist of strings of handles and levels
- Handle
  - 61-bit # identifying a compartment
  - Applications can create compartments
  - Creating a compartment returns its handle
- Level
  - Contamination & Privilege = level (*, 0-3)
  - Example send label – {   } = {A 3, B 3, 1}
- Label size linear in # compartments

Contamination Levels

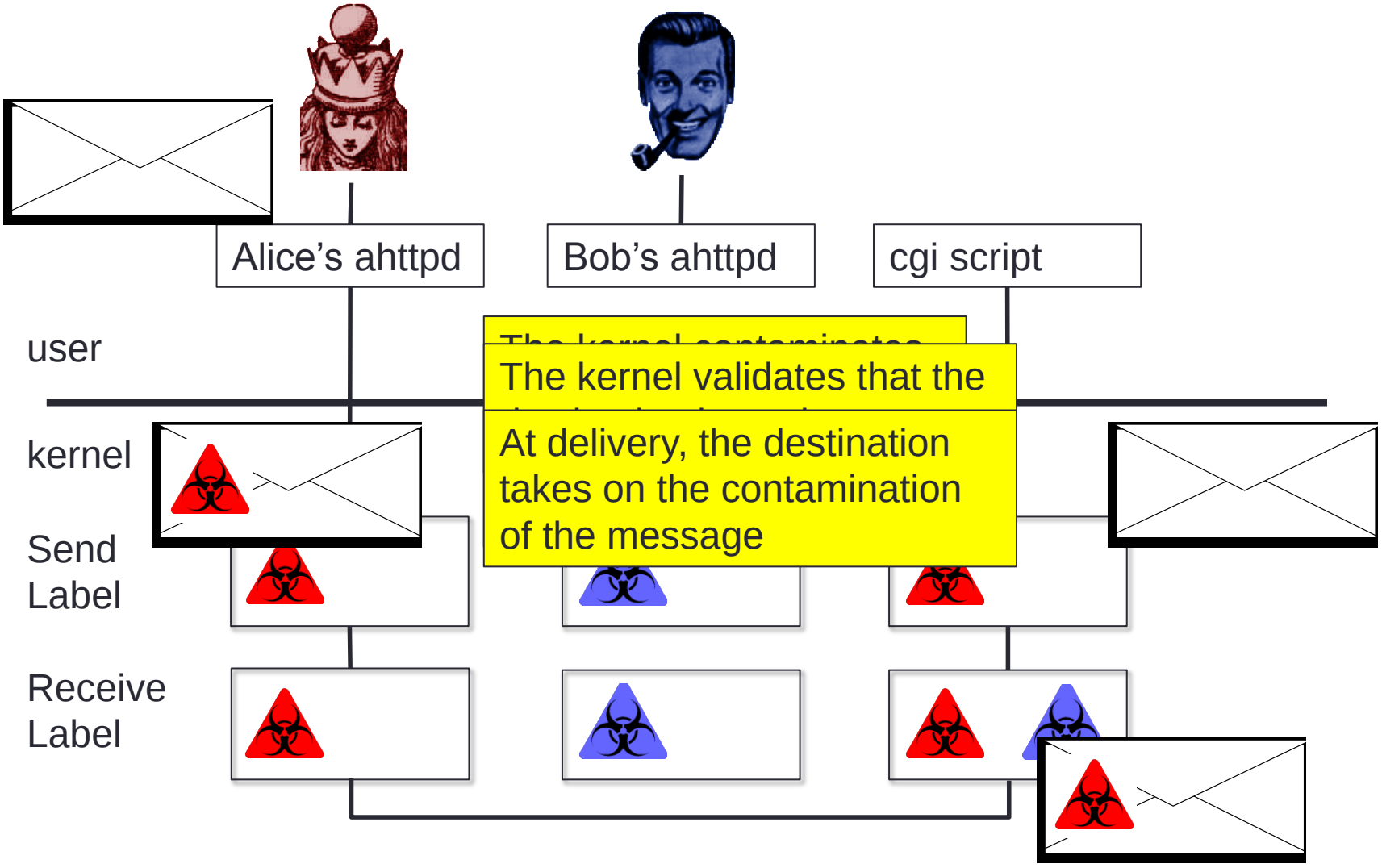
- Asbestos's four levels
  - 2 levels for send(1)/receive(2) defaults, 1 above & below defaults
- User messages using P_S of 3
  - System defaults (2) denies user-tainted messages
  - Raising P_R requires special privilege
- User messages using P_S of 2
  - System defaults (2) allows communication
  - must explicitly lower the P_R to deny user messages

Policy enforcement

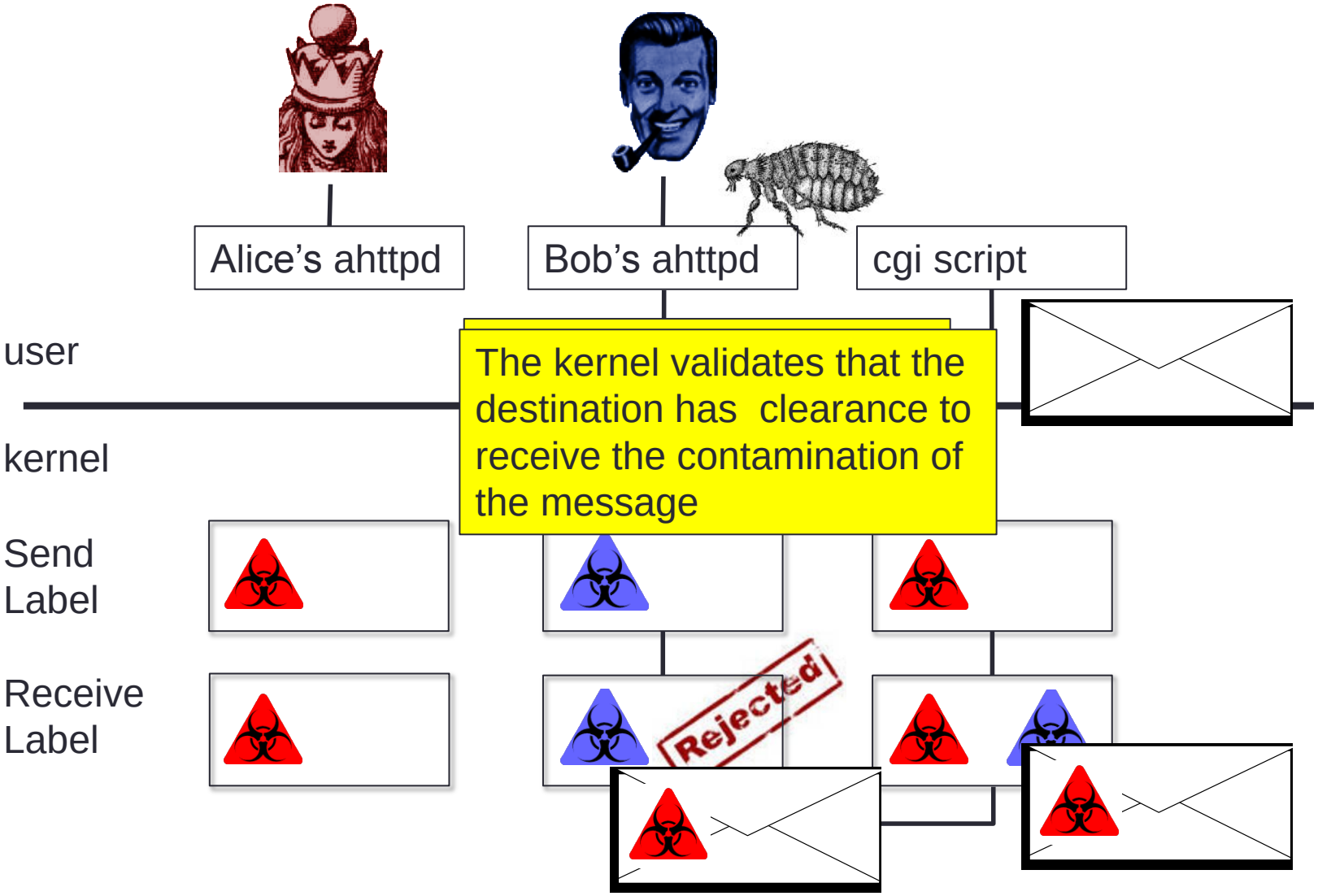
- Kernel performs clearance checks on message send
- Labels form a lattice
- Partial ordering
  - Sender's send label $\sqsubseteq$ destination's receive label
  - $L1 \sqsubseteq L2$ iff $L1(h) \leq L2(h)$ for all h
- Send label updated with a least upper bound operator



Return to example



Return to example



Efficient Isolation

- Threads don’t provide enough isolation
  - One process per user
- Processes consume too much memory
  - Solution is an event-process abstraction

```
ep_checkpoint(&msg);
if (!state.initialized) {
        initialize_state(state);
        state.reply = new_port();
}
process_msg(msg, state);
ep_yield();
```

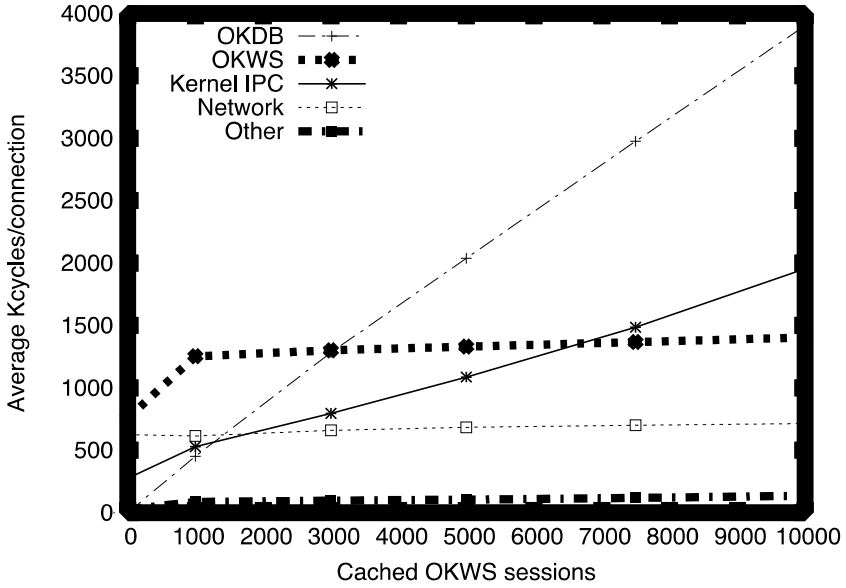
Event Process Abstraction

- Fork memory state for each new session
  - No event process will run until message is available
  - Kernel scheduling cost is little higher than a single process
  - Small differences anticipated, stored efficiently
- Event loop allows shared execution state
  - Light weight context switches
- Event process isolates state
  - Each event process is only contaminated by one user
  - One process per service with one event process per user

Evaluation

- Asbestos web server
  - Based on the OKWS
  - Enforces user isolation within workers via event processes
- Apache web server
  - Version 1.3.33
- Application
  - Response is a string of characters whose length depends on client's parameters
- OKWS had comparable throughput and latency to Apache

Label Costs



- Size of labels in the system increases with # sessions
- IPC operations take more time as # sessions increase
- Linear performance degradation as labels increase in size

Perspective

- Asbestos labels make MAC (mandatory access control) tractable
  - Labels provide decentralized policy specification
  - Kernel enforces information flow control
  - Event processes efficiently supports many users
- The OK web server on Asbestos
  - Performs comparably to Apache
  - Provides better security properties than Apache

Backup slides

Delegating to user processes

- Using IPC, a task can
  - Allocate a region of VM
  - De-allocate a region of VM
  - Set the protection status of a region of VM
  - Specify the inheritance of a region of VM
  - Create and manage a memory object

Sharing Memory

- Copy-on-write for sharing memory objects
  - Shadow objects
  - Rely on original object for unmodified data
  - Shadow objects may also be shadowed
- Read/write sharing must use the sharing map
  - Sharing map is identical in organization to address map

Multiprocessor Issues

- TLBs Consistency
- Solutions
  - Force interrupts to all CPUs
  - Wait until timer interrupt
  - Temporarily allow inconsistency