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

And

Labels and Event Processes in the Asbestos Operating System

Presented by Petko Nikolov

9/22/09

Mach

- Problem
  - OS portability suffers due to diff. memory structures
- Solution
  - Portable, multiprocessor OS – Mach
  - Few assumptions about memory hardware
    - Just recover from page faults

Mach VM

- Supports:
  - Large, sparse virtual address spaces
  - Copy-on-write virtual copy operations
  - Copy-on-write and read-write memory sharing
  - Memory mapped files
  - User-provided backing store objects and pagers

Mach Design

- Task
- Thread
- Port
- Message
- Memory object

VM Operations

- A task can:
  - Allocate a region of VM on a page boundary
  - Deallocate a region of VM
  - Set the protection status of a region
  - Specify the inhertance of a region
  - Create and manage a memory object

Implementation

- 4 basic memory management data structures:
  - Resident page table
  - Address map
  - Memory object
  - Pmap
- Machine dependent vs independent

Resident Memory

- Physical memory – cache for virtual memory objects
- Physical page entries linked into:
  - Memory object list
  - Memory allocation queues
  - object/offset hash bucket

Address Maps

- Doubly-linked list of address map entries
- Map range of virtual addresses to area in virtual object
  - Contiguous
- Efficient for most frequent operations:
  - Page fault lookups
  - Copy/protection operations on address ranges
  - Allocation/deallocation of address ranges

Memory Objects

- Repository for data, indexed by byte
  - Resembles a UNIX file
- Reference counters allow garbage collection
- Pager – memory object managing task
  - Handles page faults, page-out requests outside of kernel

Sharing Memory

- Copy-on-write
  - Shadow objects
  - Remembers modified pages
- Read/write sharing
  - Memory object not appropriate for this
  - Must use sharing maps

Object Tree

- Must prevent large chains of shadow objects
  - Utilize GC for shadow objects
- Unnecessary chains occurs during heavy paging
  - Cannot be detected easily
- Complex locking rules

pmap

- Management of physical address maps
  - Only machine-dependent module
  - Implement page-level operations
  - Ensure hardware map is operational
  - Need not keep track of all currently valid mappings
- Machine-independent parts are the driving force of Mach VM operations

Porting Mach VM

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

Performance

Performance of Mach VM Operations

Operation		Mach	UNIX
zero fill 1K (RT PC)		.45ms	.58ms
zero fill 1K(uVAX II)		.58ms	1.2ms
zero fill 1K(SUN 3/160)	.23ms	.27ms	
fork 256K (RT PC)		41ms	145ms
fork 256K (uVAX II)		59ms	220ms
fork 256K (SUN 3/160)	68ms	89ms	
read 2.5M file(VAX 8200)	(system/elapsed sec)		
first time		5.2/11sec	5.0/11sec
second time		1.2/1.4sec	5.0/11sec
read 50K file (VAX 8200)	(system/elapsed sec)		
first time		.2/.3sec	.2/.5sec
second time		.1/.1sec	.2/.2sec

Table 7-1:

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

Summary

- Sophisticated, hardware-independent VM system possible
- Can achieve good performance in some cases

Asbestos

Labels and Event Processing in the Asbestos Operating System

With slides borrowed from SOSP 2005 Asbestos presentation

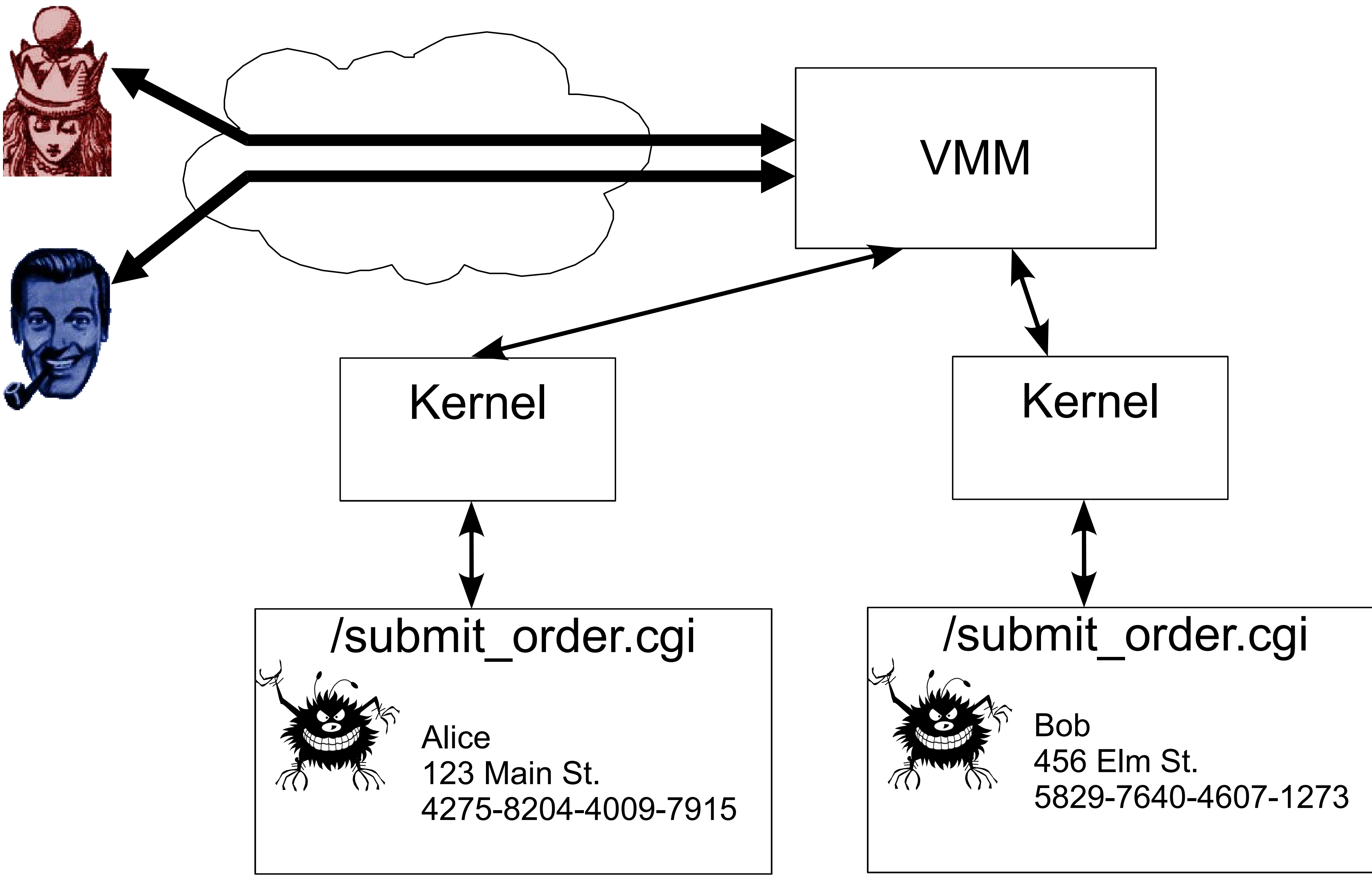
Asbestos Outline

- Why is it needed?
- Other models
  - Virtual machines
- Asbestos OS
  - Labels
  - Event processes
- Asbestos OKWS
- Performance

The Problem

- Web servers have exploitable software flaws
  - SQL injection, buffer overrun
- Private information leaked
  - Credit card #'s, SS #'s
  - All data potentially exposed due to single flaw
- Lack of isolation of user data
- Unconstrained information flow

Virtual Machine Isolation



Problem with VM Isolation

- Course-grained sharing/isolation
- Heavy on resources
- Clumsy way to handle problem
  - Requires separate instance of OS for each label
  - Should really have support for this in OS

Information Flow Control Systems

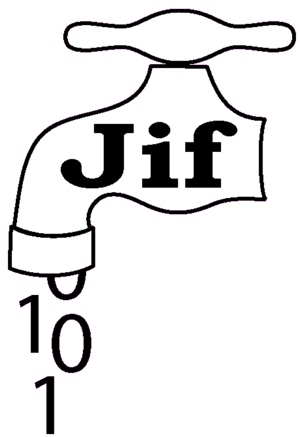
- Conventional multi-level security
  - Kernel-enforced information flow control across processes
  - A handful of *levels* and *compartments*: “secret, nuclear”
  - Inflexible, administrator-established policies
  - Central authority, no privilege delegation
- Language-enforced information flow (Jif)
  - Applications can define flexible policies at compile time
  - Enforced within one process
- **Asbestos**
  - Applications can define flexible policies
  - Kernel-enforced across all processes

Approaches

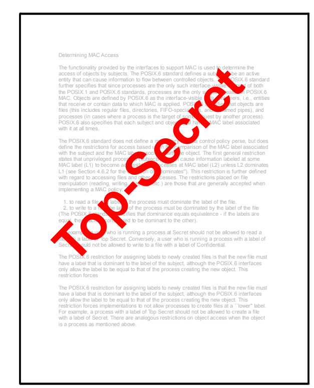
Policy defined by:

Application

Kernel



Asbestos



Conventional MLS

Within a process

Across processes

Asbestos Goal

Asbestos should support efficient, unprivileged, and large-scale server applications whose application-defined users are isolated from one another by the operating system, according to application policy.

Asbestos Goal

- Large-scale
  - Changing population of thousands
- Efficient
  - Cache user data, while keeping it isolated
- Unpriviliged
  - Minimum privilege required
- Application defines notion of user
- Isolation of users' data
- Application policy
  - Application-defined, OS-enforced

Asbestos Overview

- IPC similar to that of Mach
  - Messages sent to ports
  - Asynchronous, unreliable
- Asbestos labels
  - Track, limit flow of information
- Event processes
  - Efficiently support/isolate many concurrent users

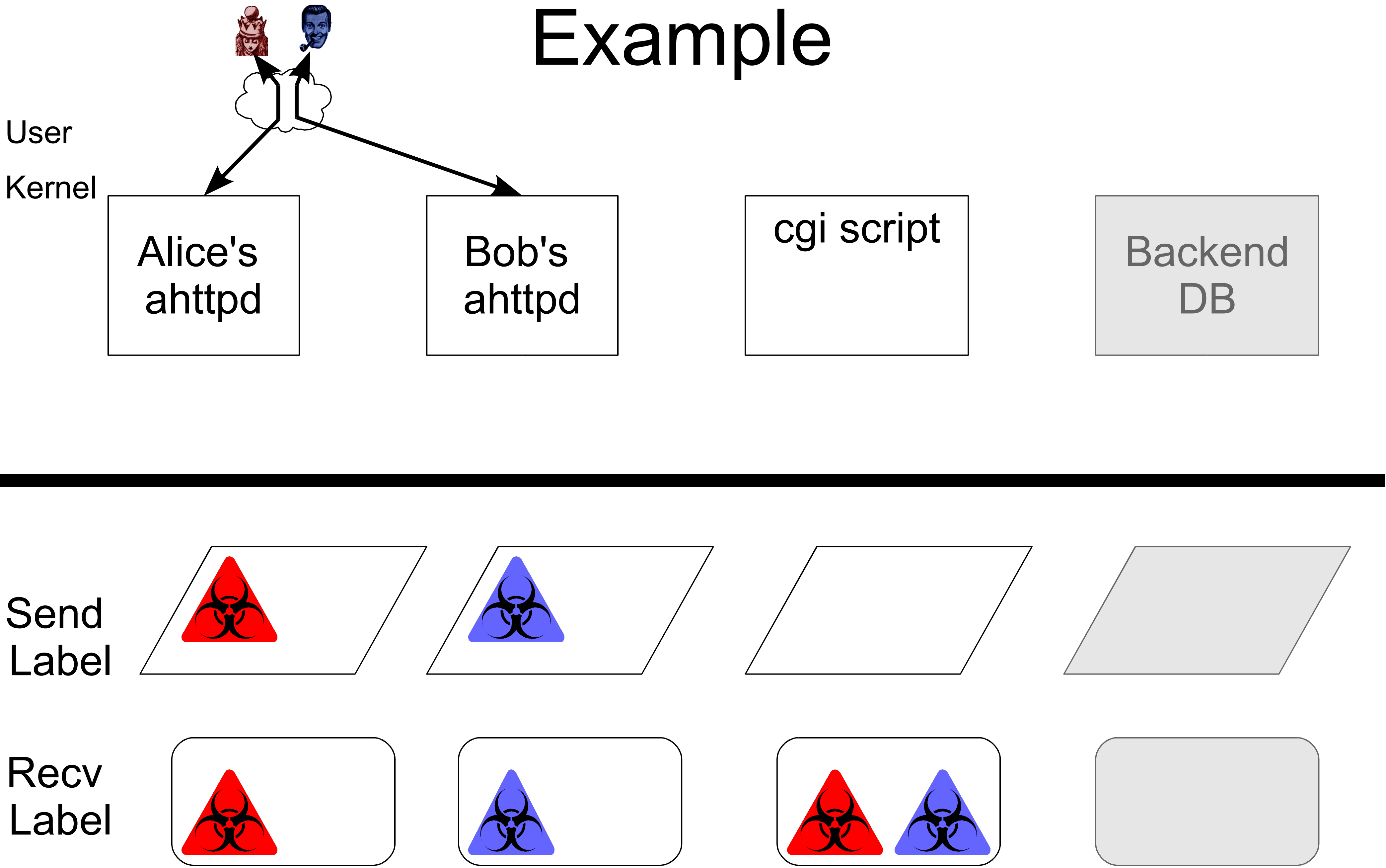
Compartments

- Contamination / label type
  - Mike's data, Michele's data, Peter's business data
- Created by application
  - Creator process can delegate rights

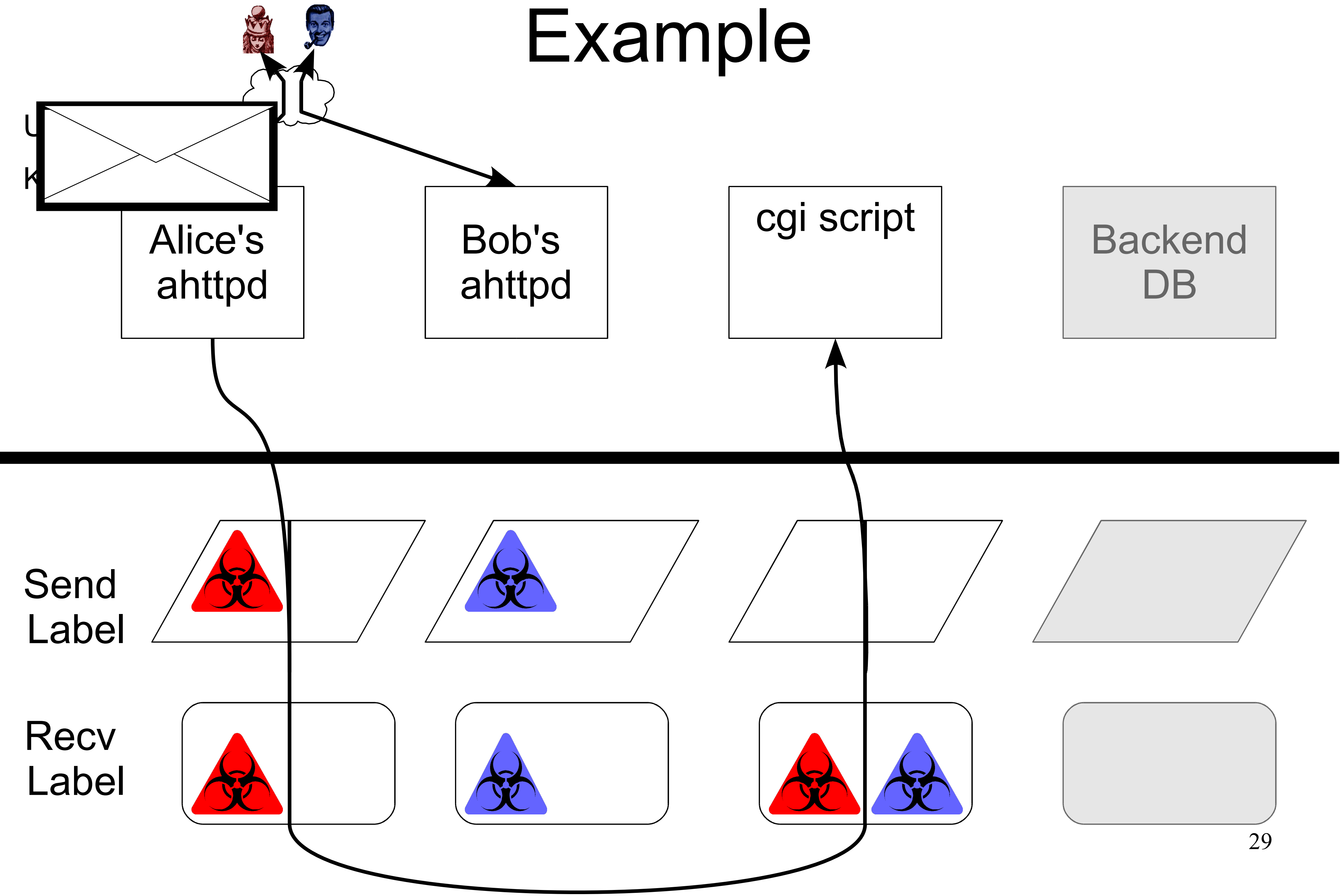
Labels

- Each process has send and receive label
  - Send label track current contamination
  - Receive label tracks max contamination (clearance)
- Rules enforced when messages are sent
- Contamination of receiver updated

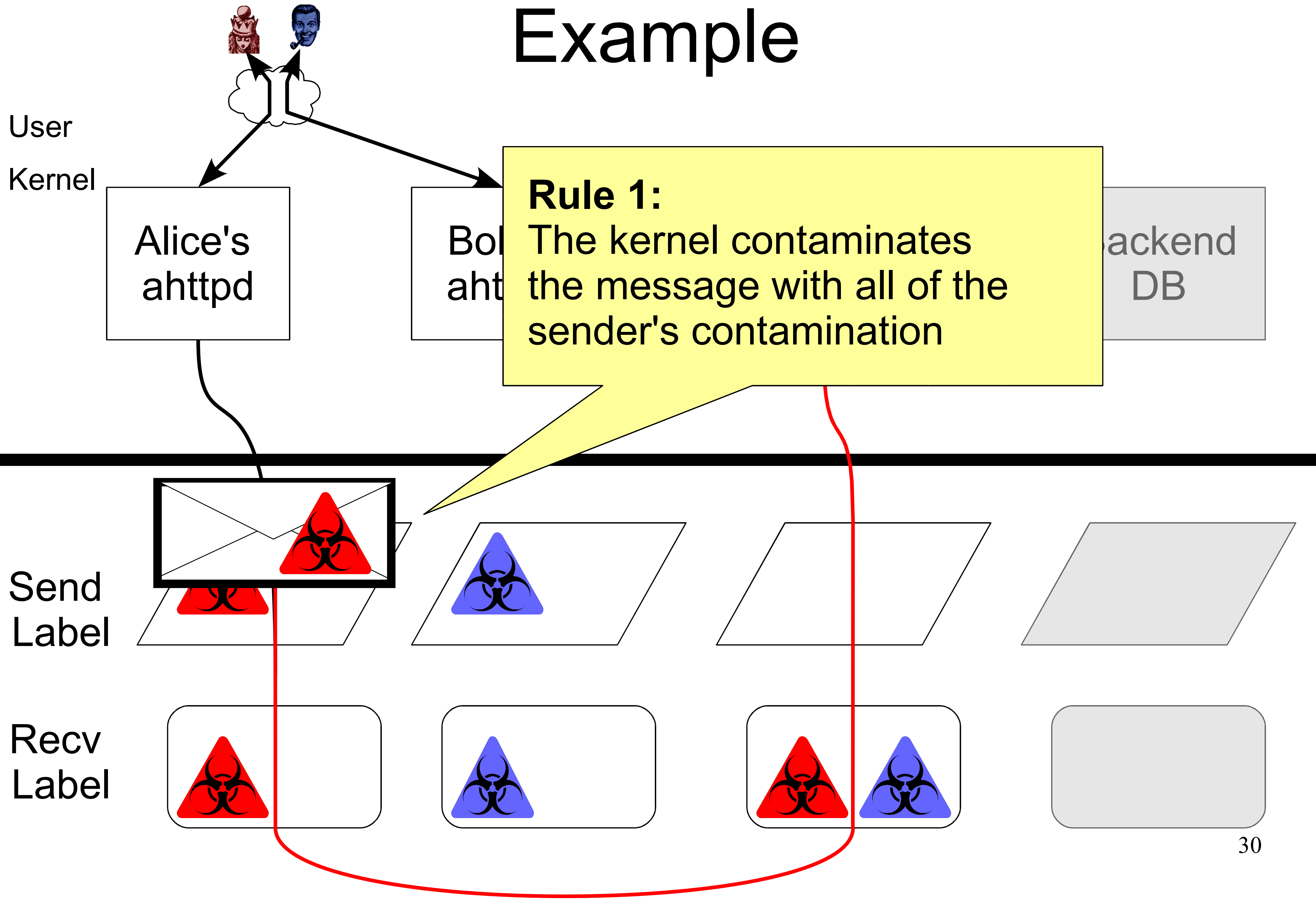
Basic Example



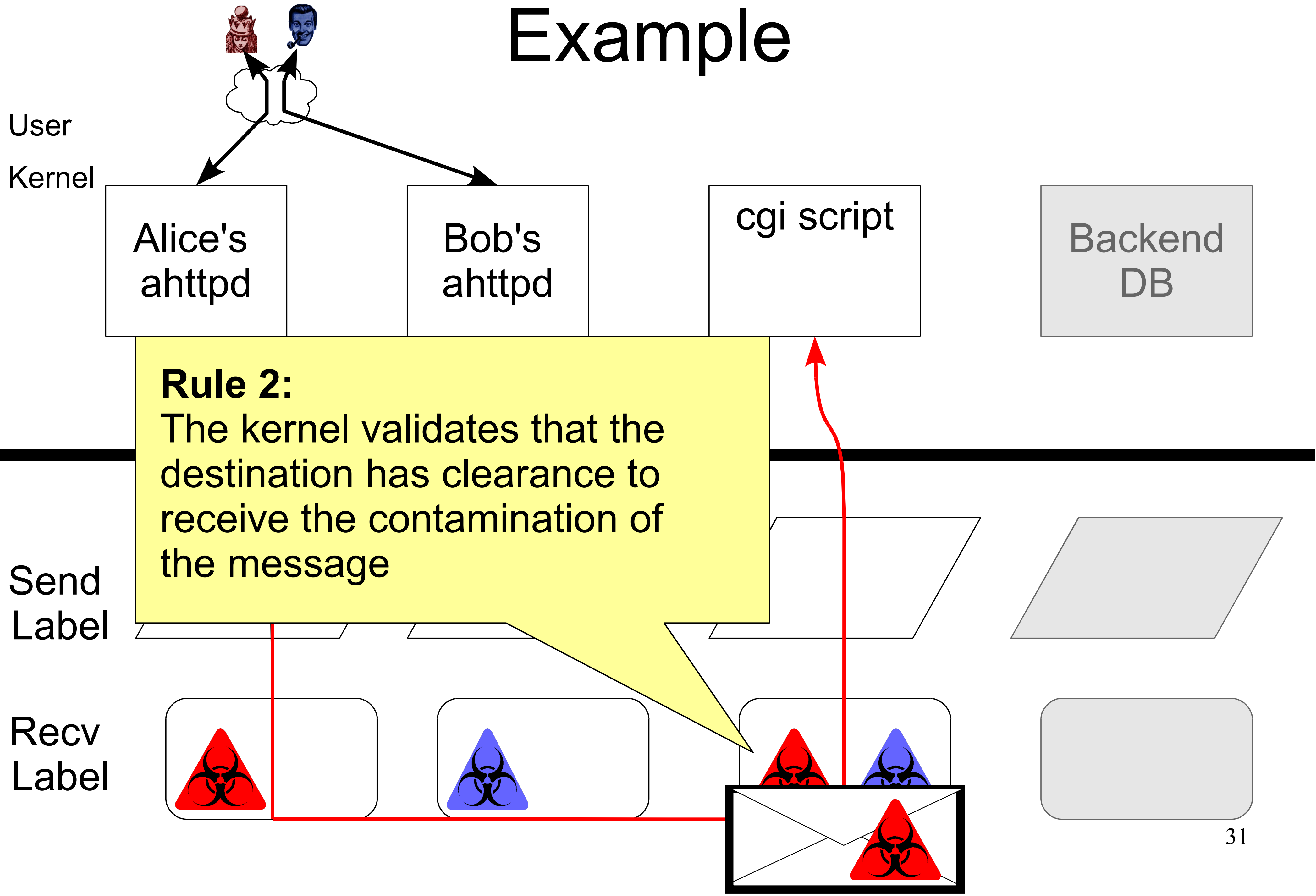
Basic Example



Basic Example



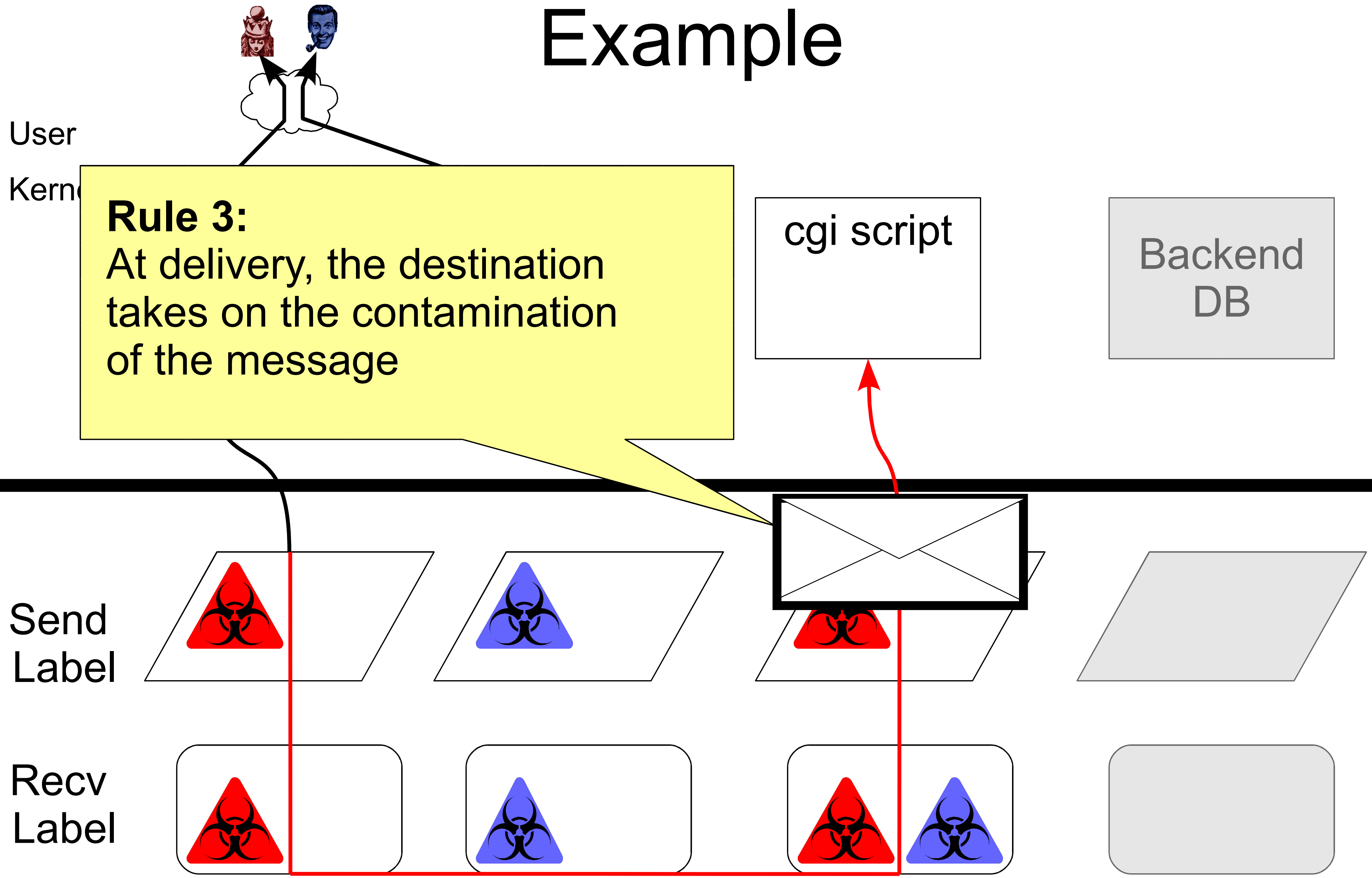
Basic Example



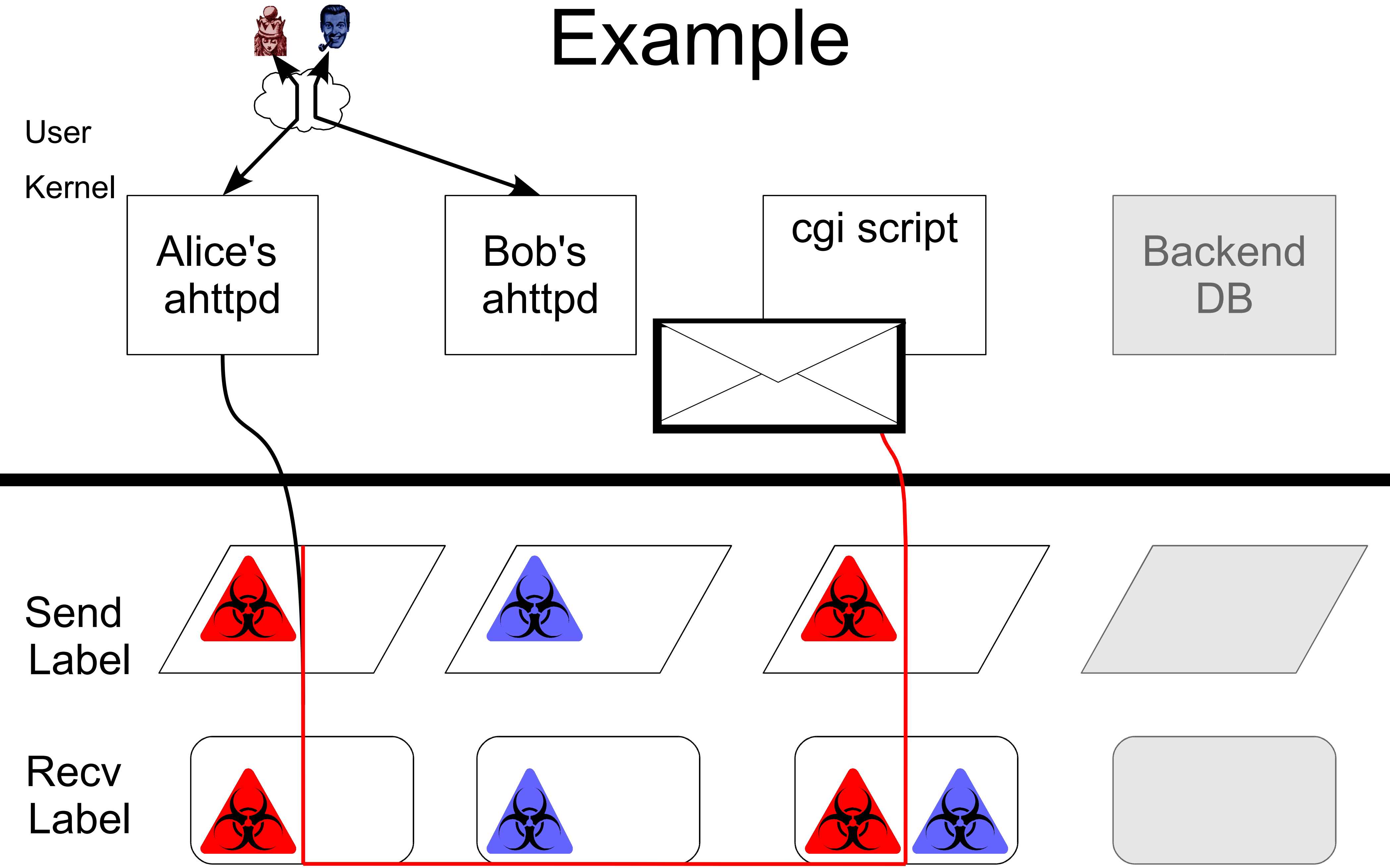
Rule 2:
The kernel validates that the destination has clearance to receive the contamination of the message

Basic Example

Rule 3:
At delivery, the destination takes on the contamination of the message

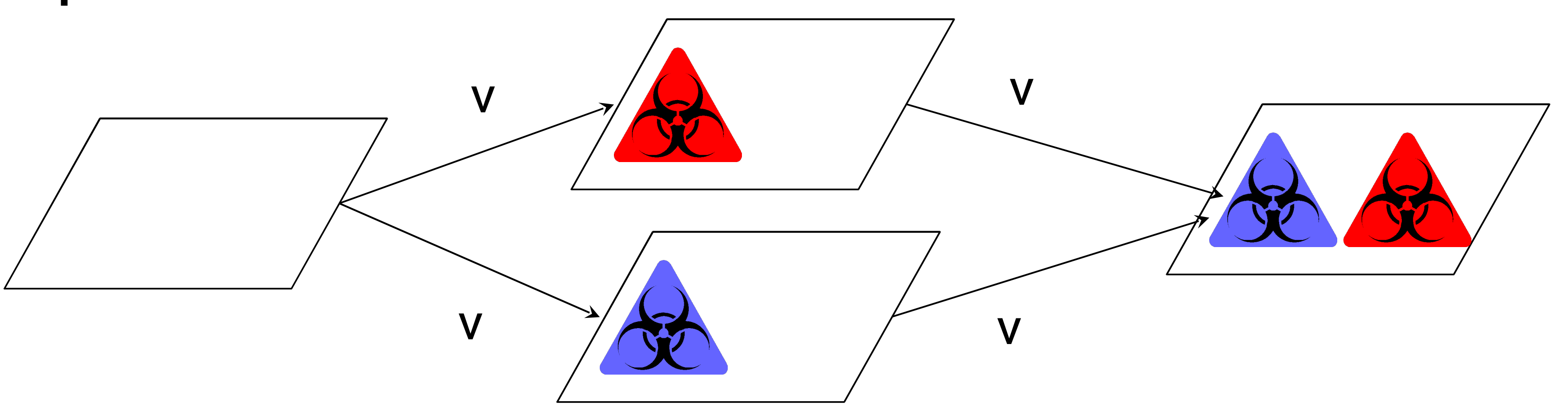


Basic Example

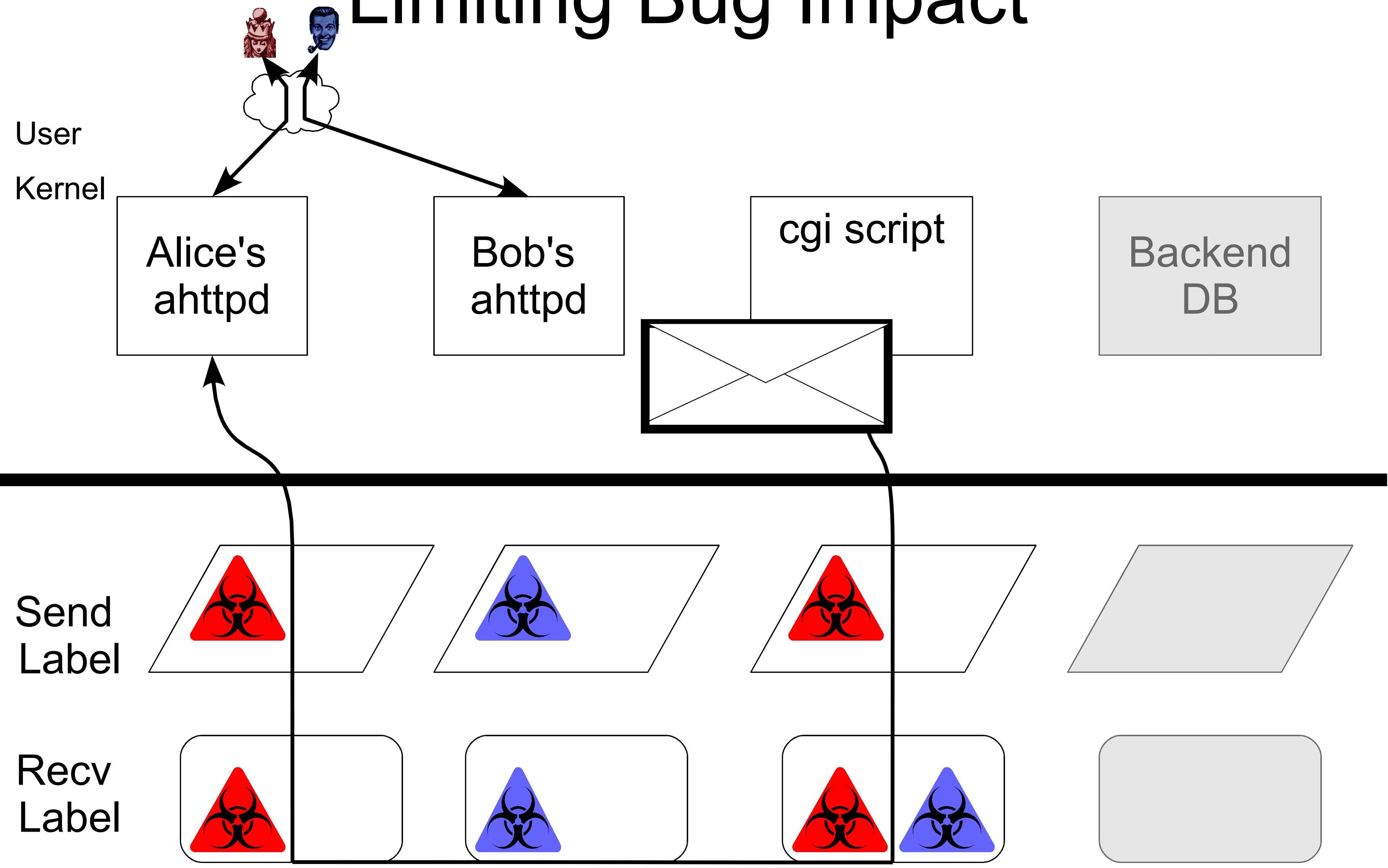


Implementing Clearance Checks

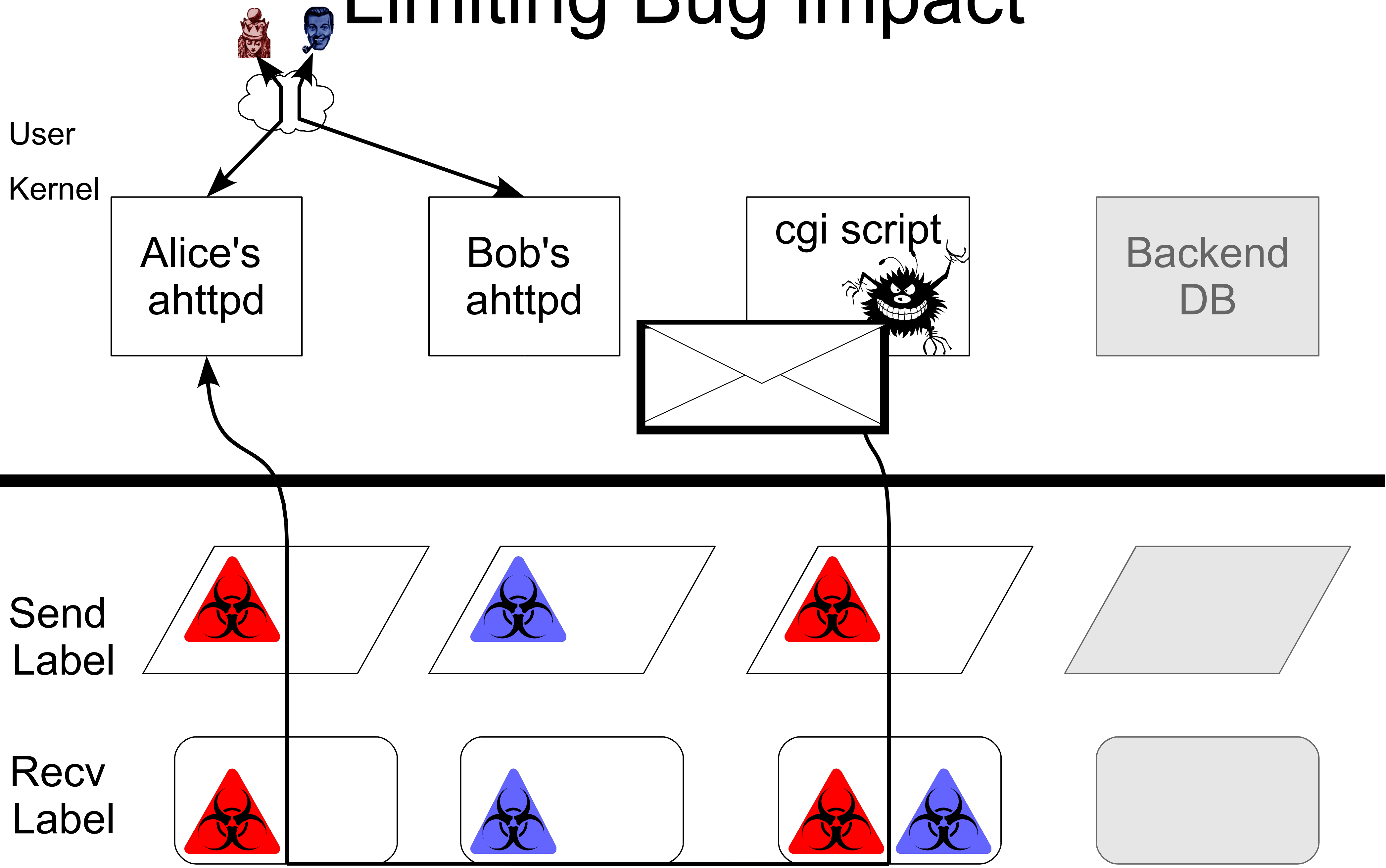
- How does the clearance check work?
- Labels form a lattice
- Partial ordering
  - Sender's send label must be less than or equal to the destination's receive label
- Send label updated with a least upper bound operator



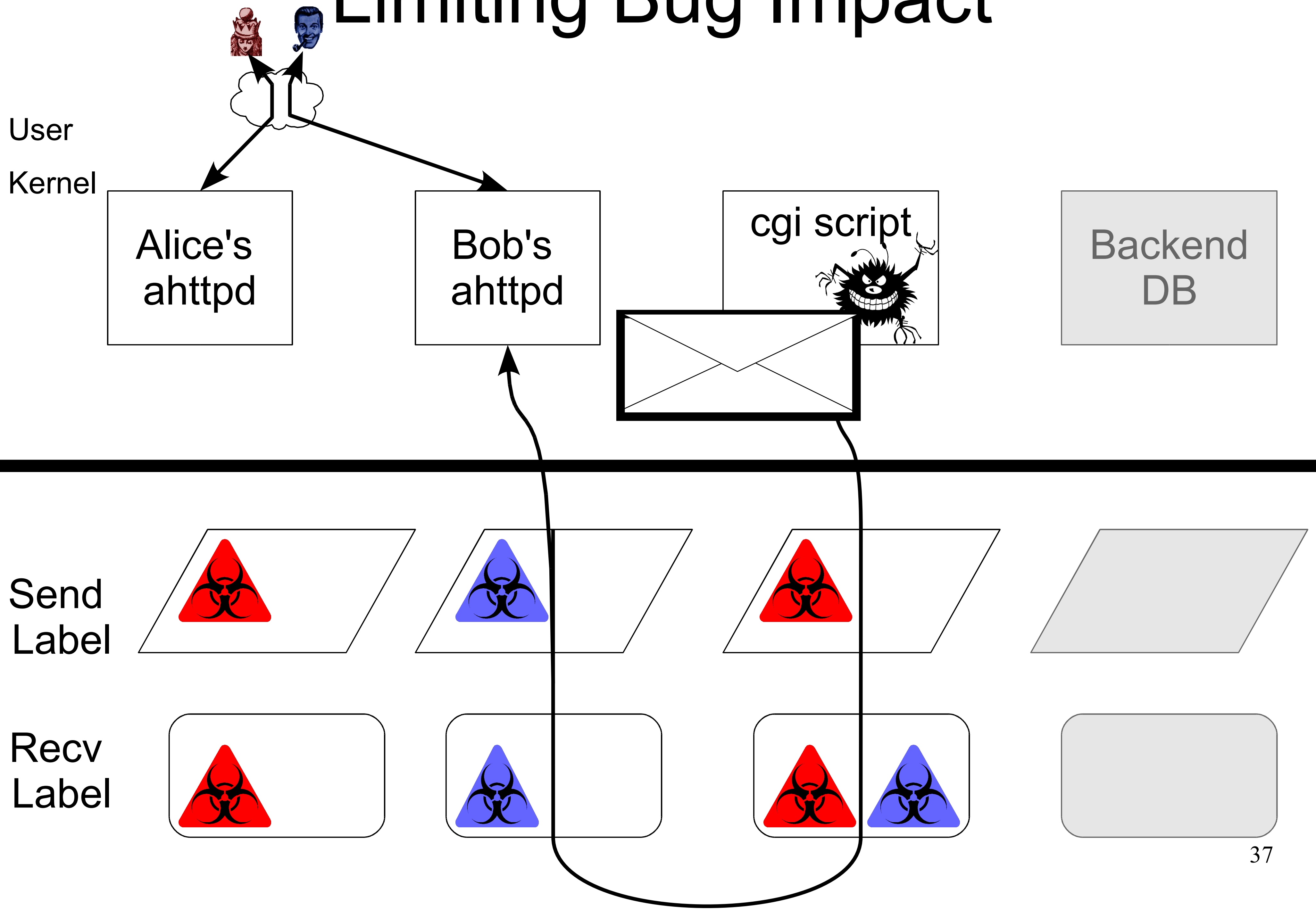
Limiting Bug Impact



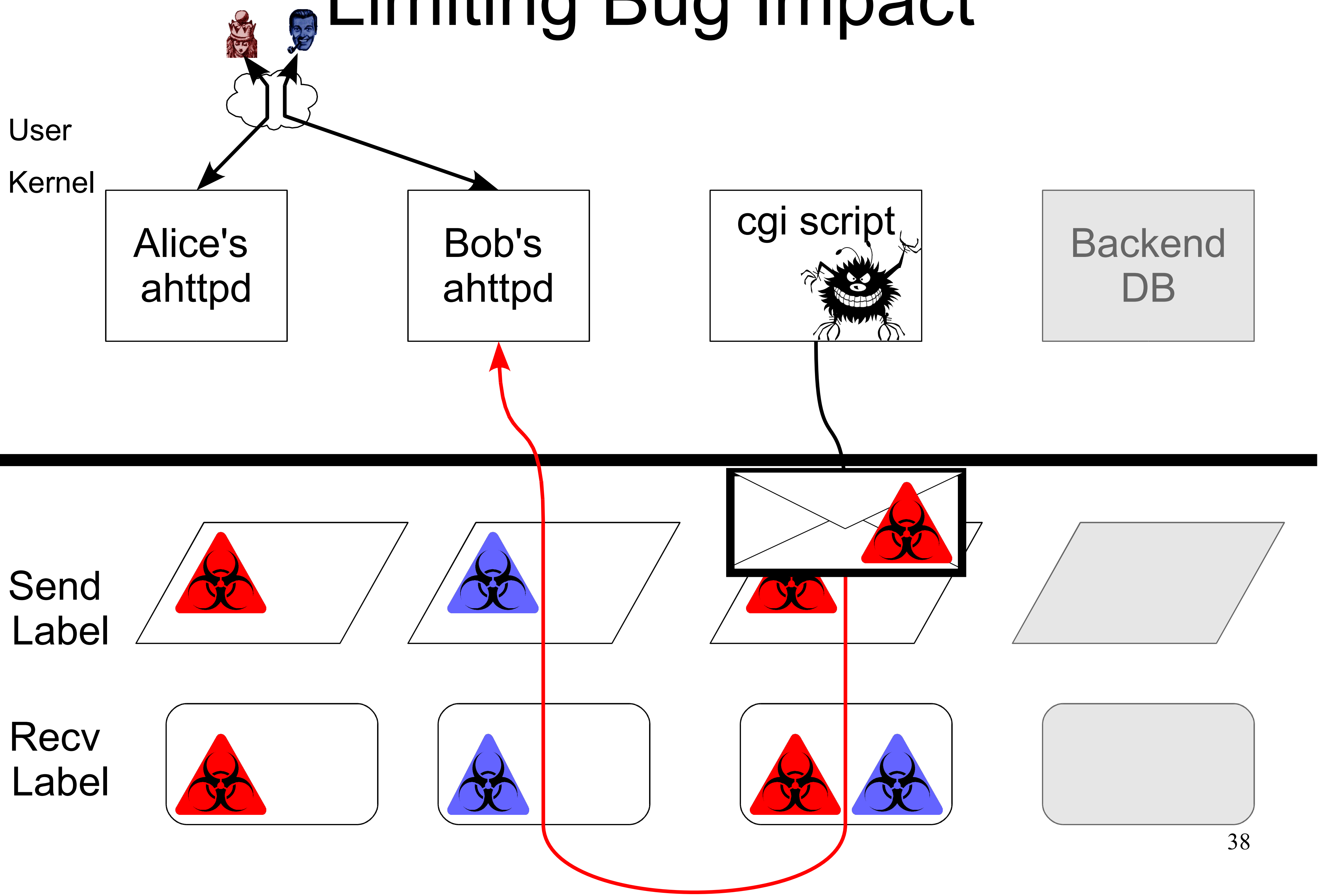
Limiting Bug Impact



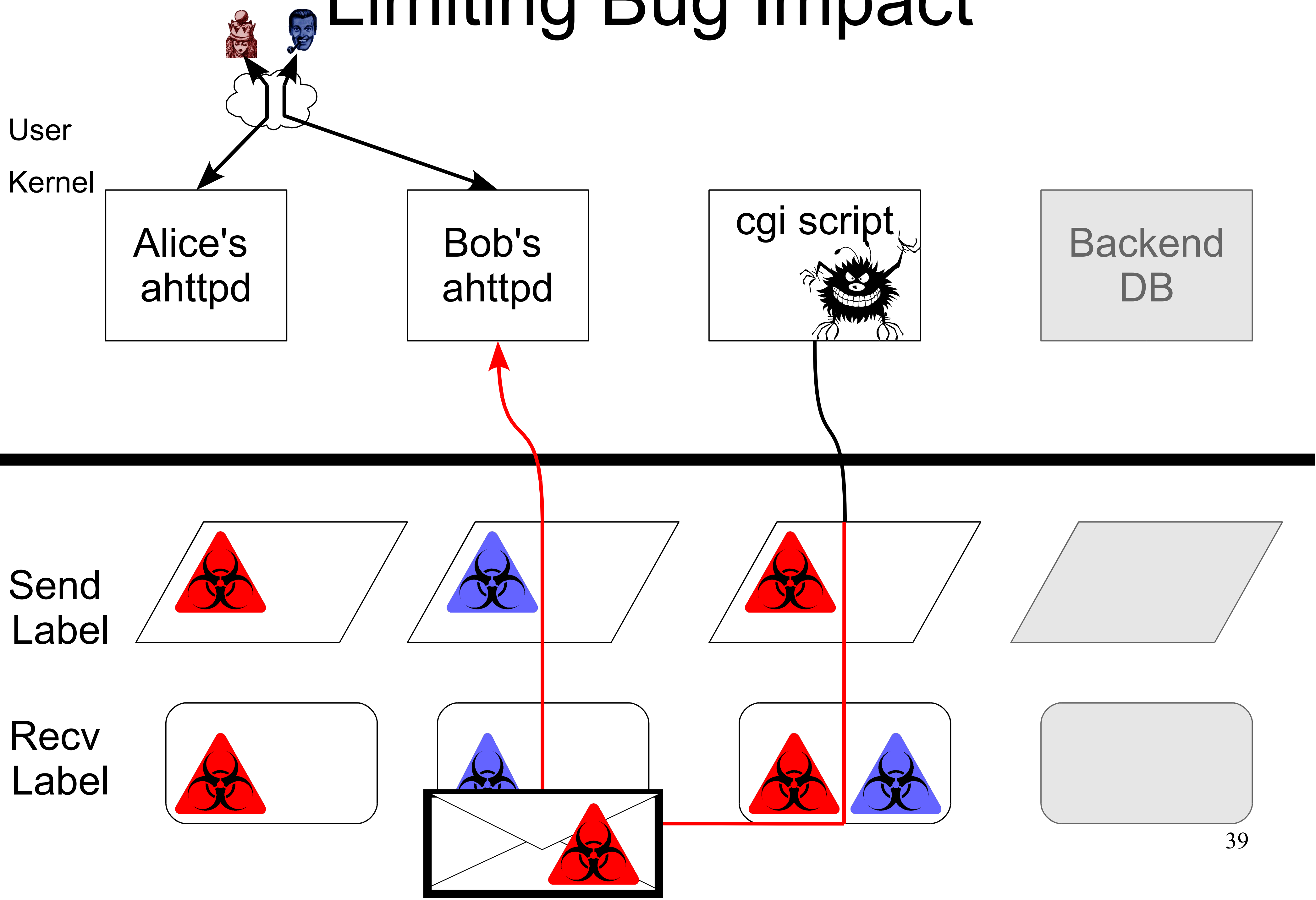
Limiting Bug Impact



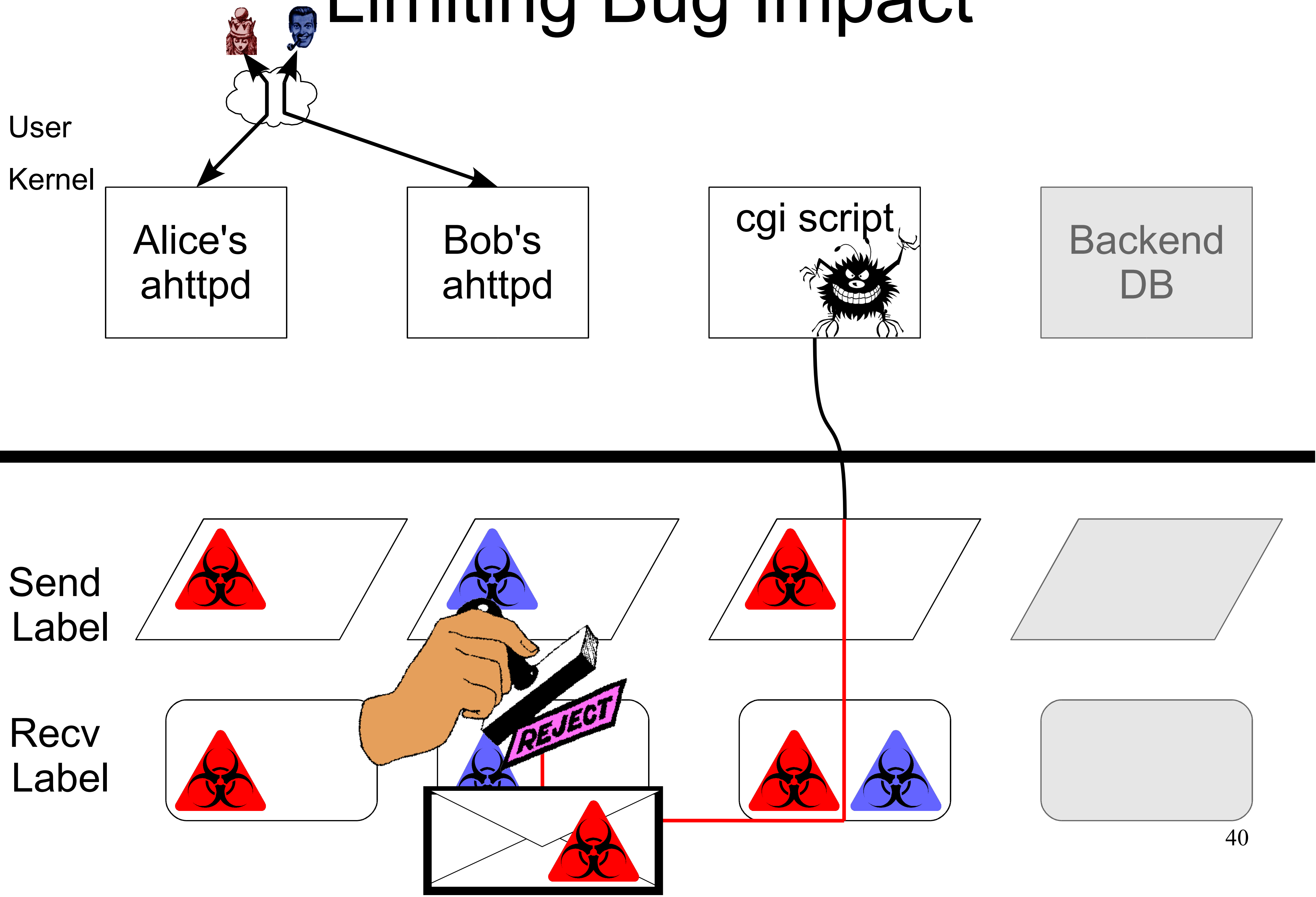
Limiting Bug Impact



Limiting Bug Impact



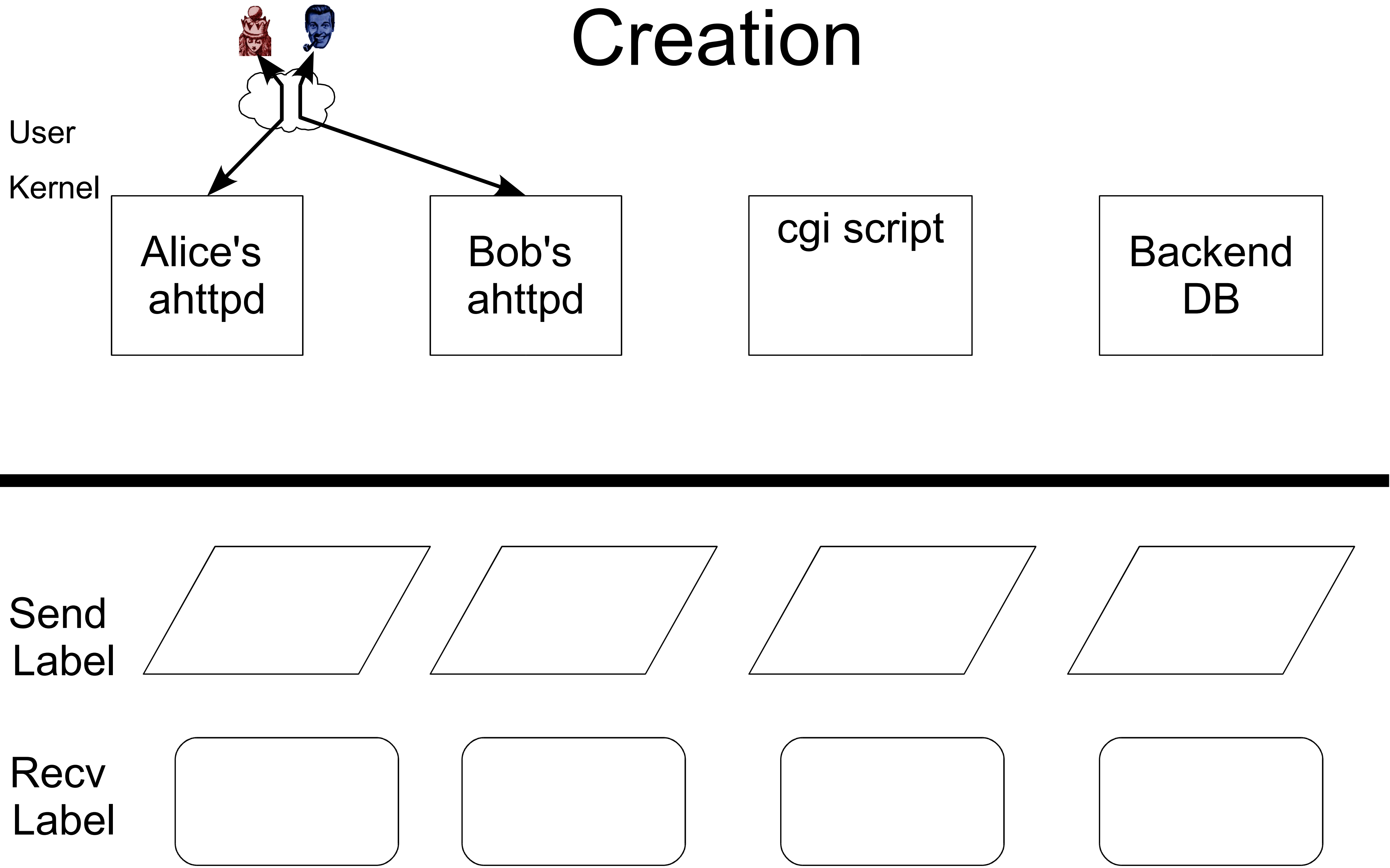
Limiting Bug Impact



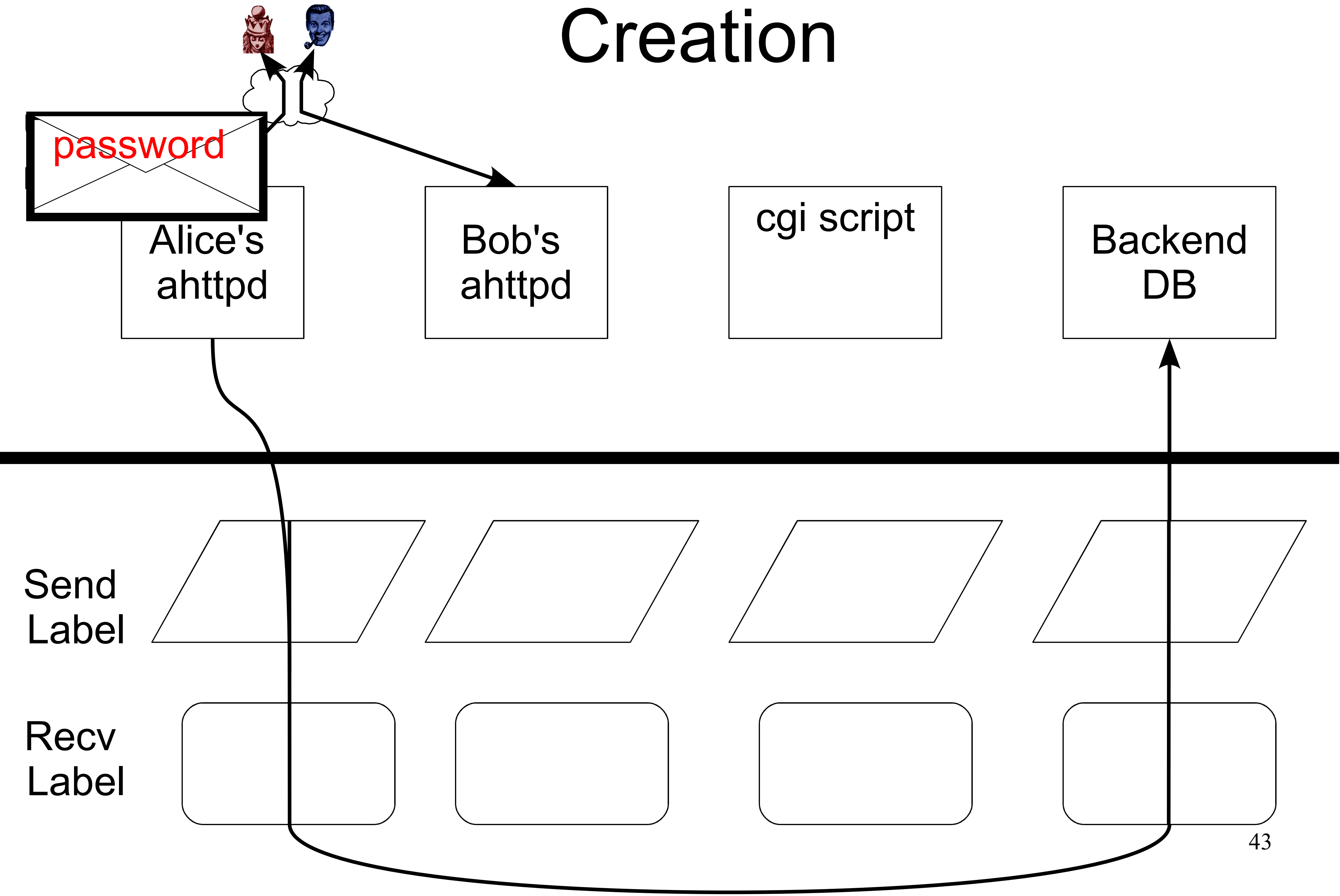
Application Defined Policies

- Where did the compartments come from?
- How did the labels get set the way they are?
- In traditional multi-level security systems, the system operator does these things
- Asbestos labels provide a decentralized and unprivileged method to set these initial conditions

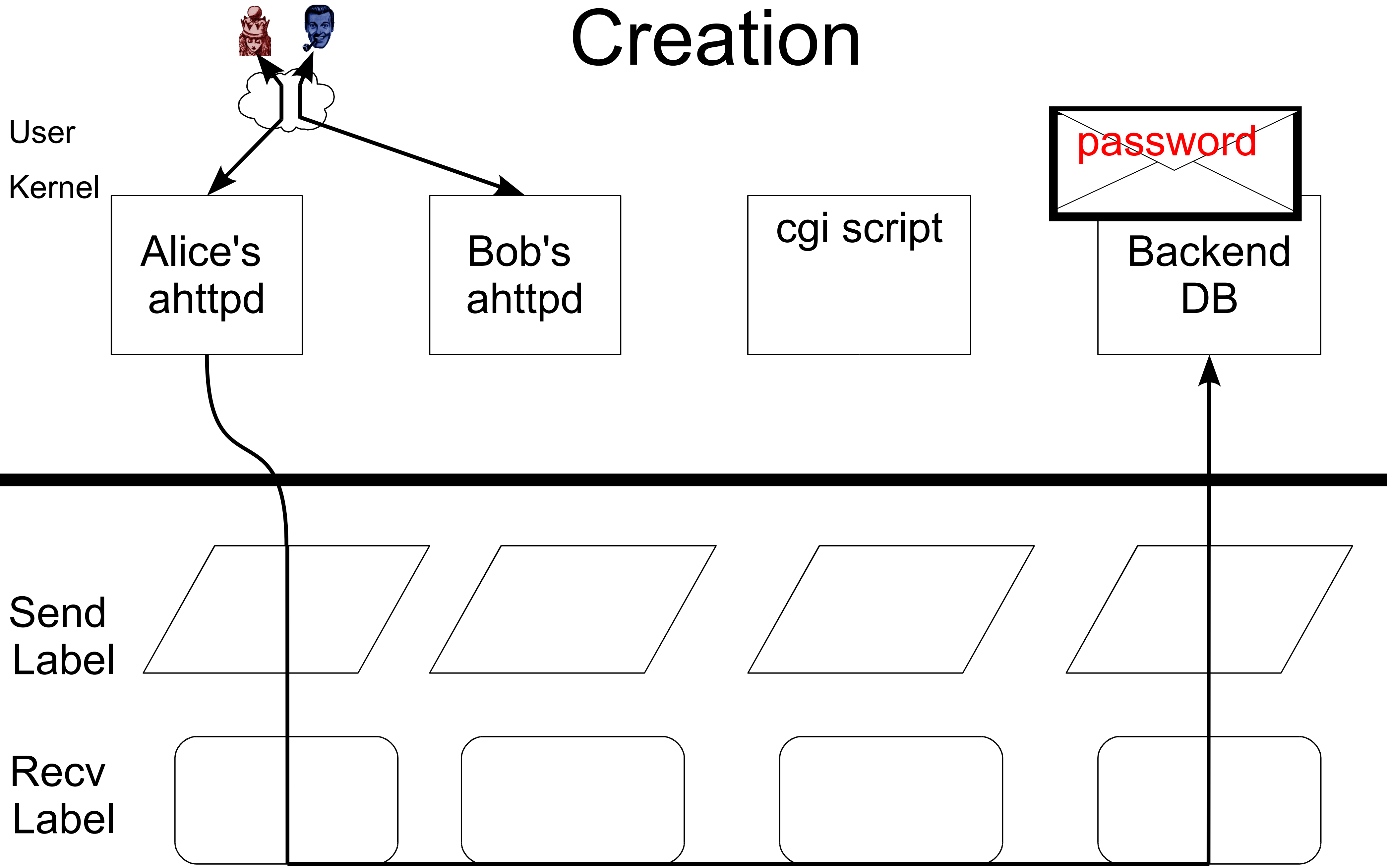
Compartment Creation



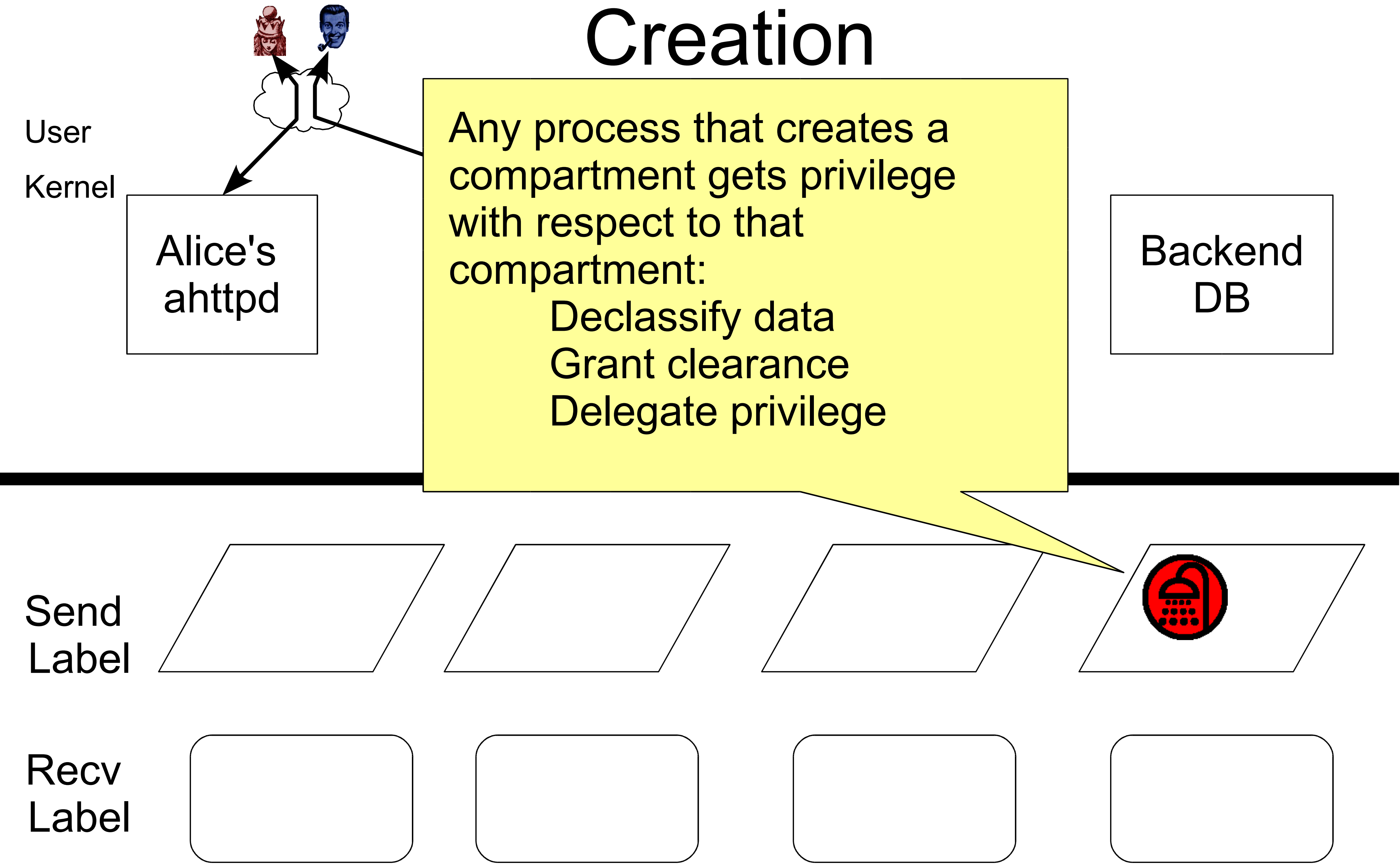
Compartment Creation



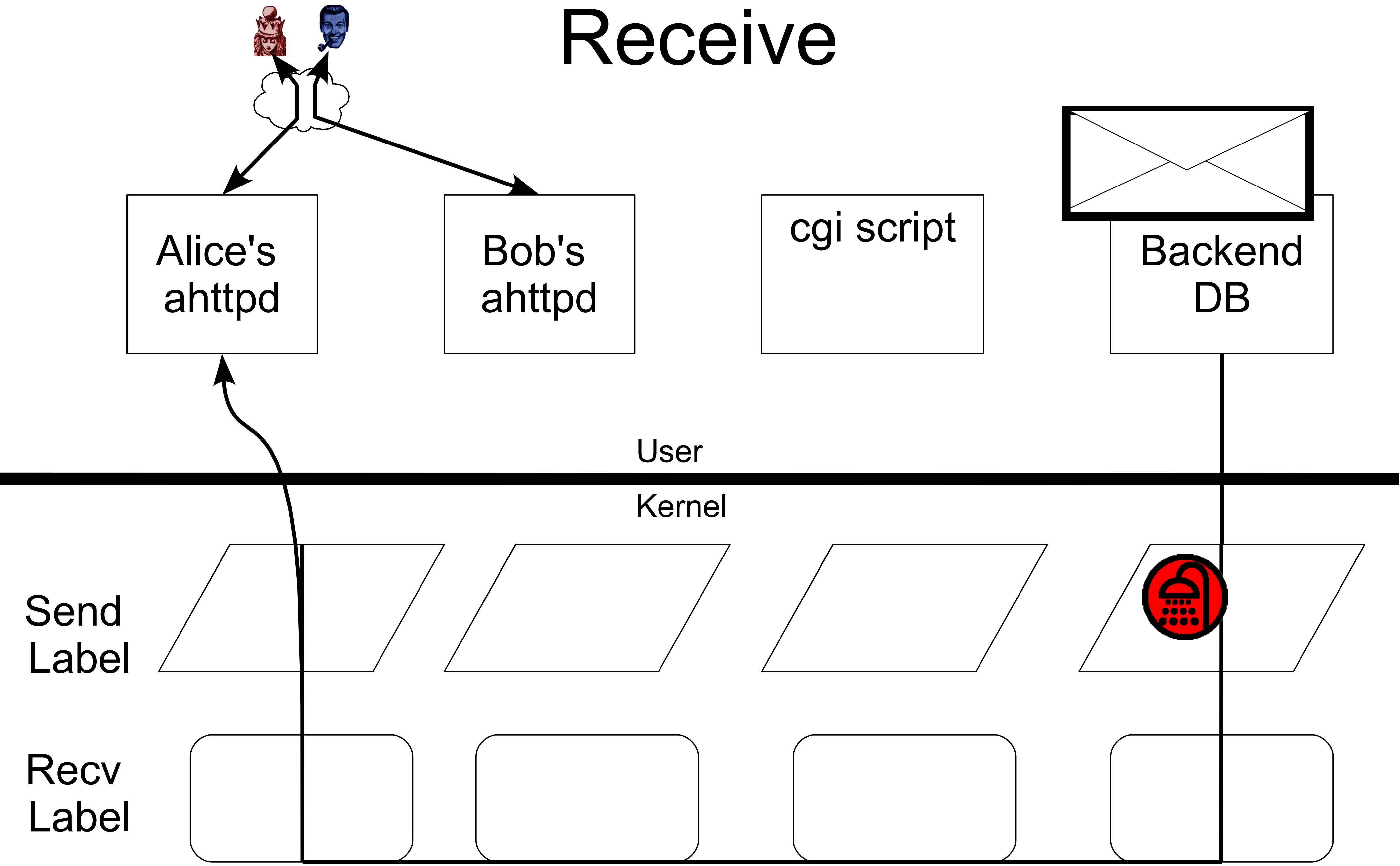
Compartment Creation



Compartment Creation



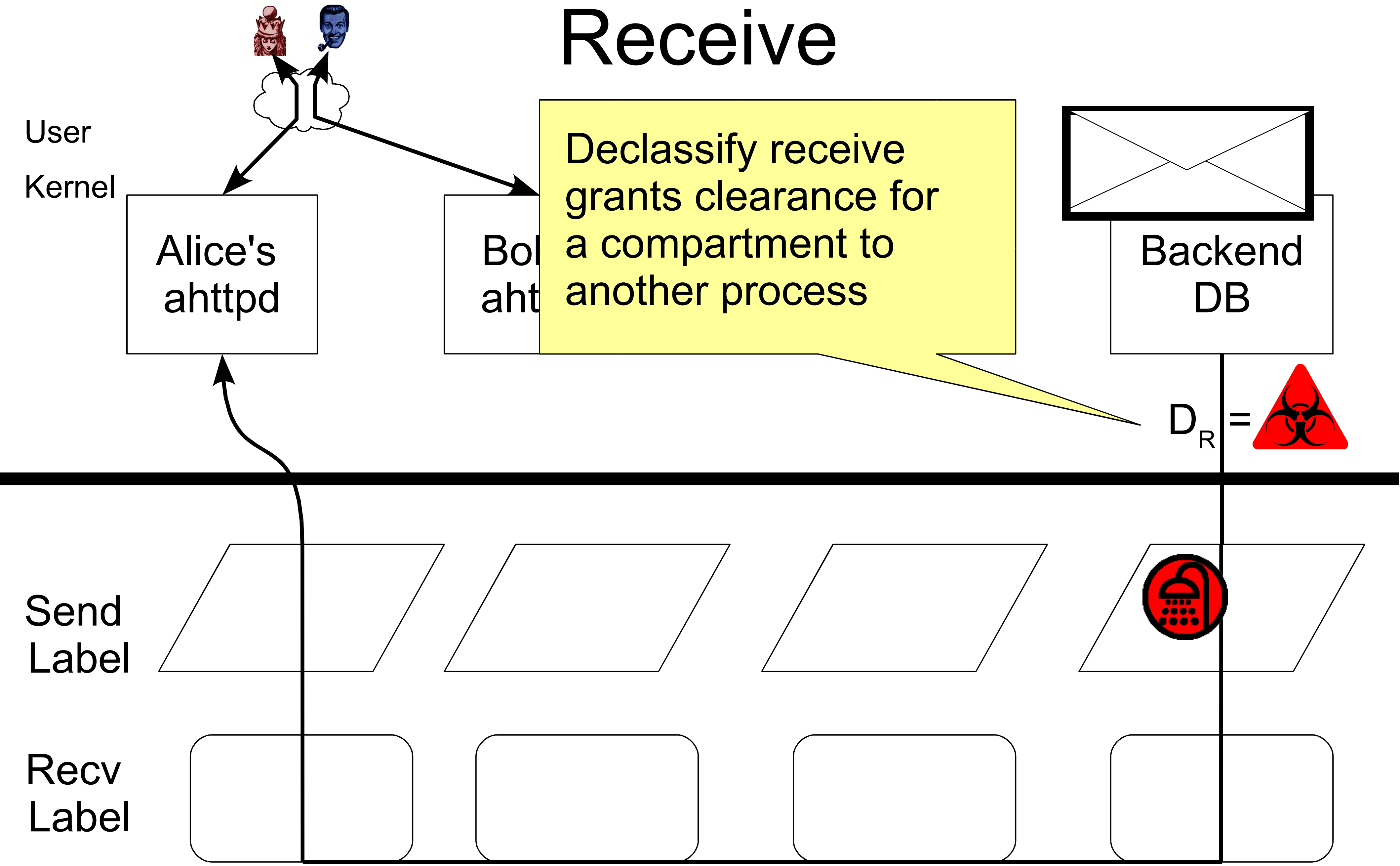
Declassify
Receive



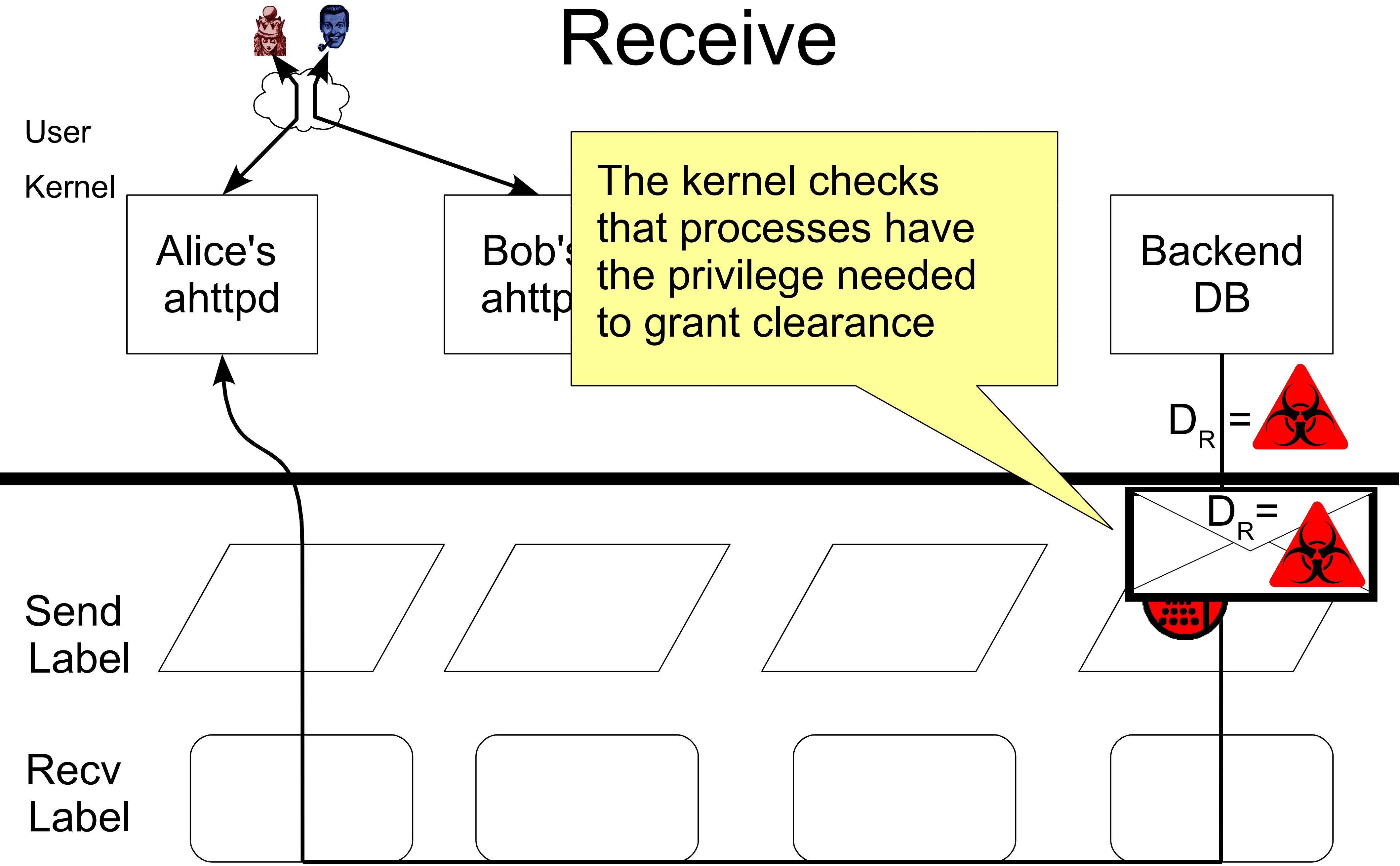
Optional Labels

- Process can attach optional (discretionary) labels to messages
  - C_S – Contaminate Send
  - D_R – Declassify Receive
  - D_S – Declassify Send
  - V – Verify

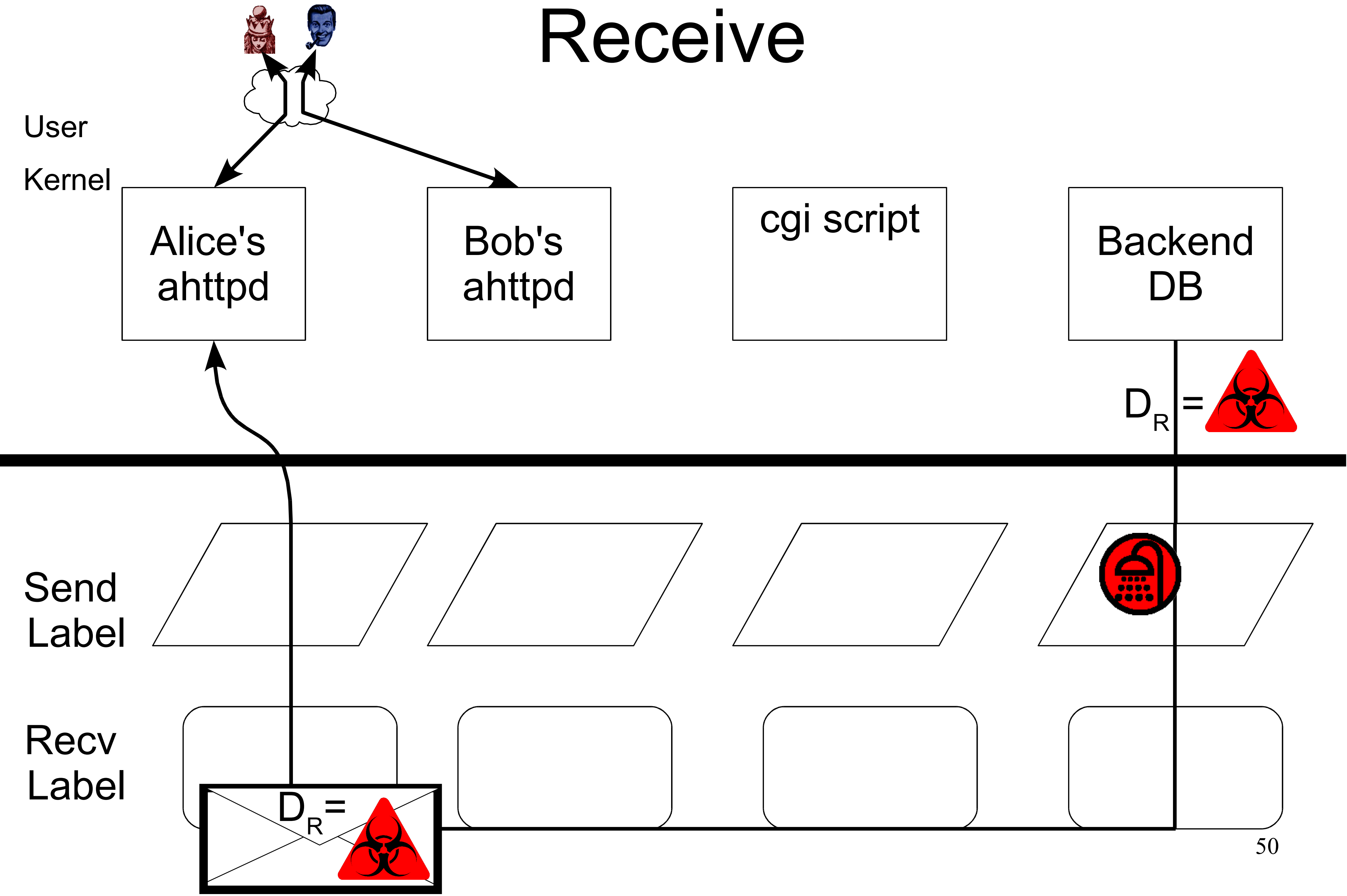
Declassify Receive



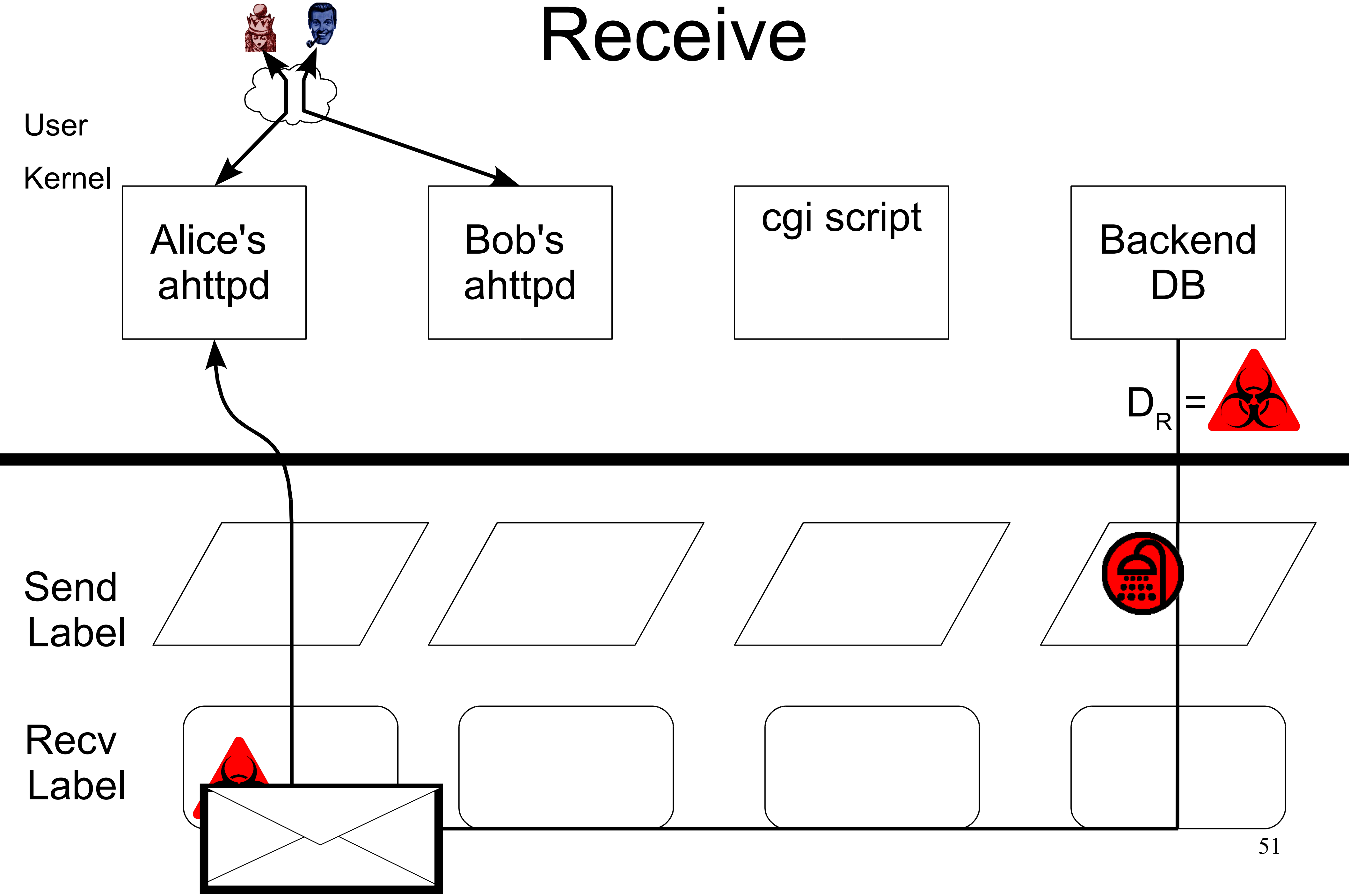
Declassify Receive



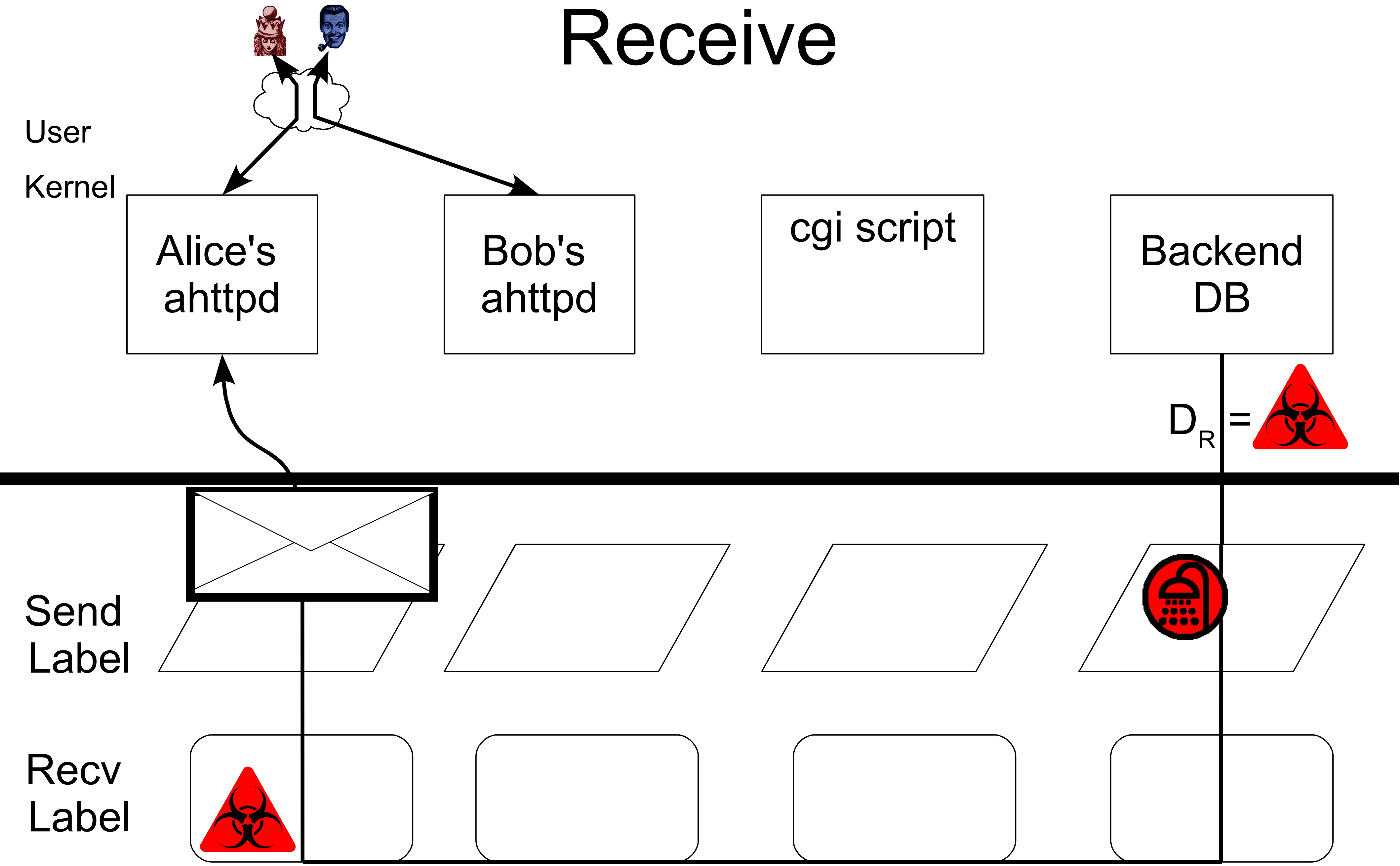
Declassify Receive



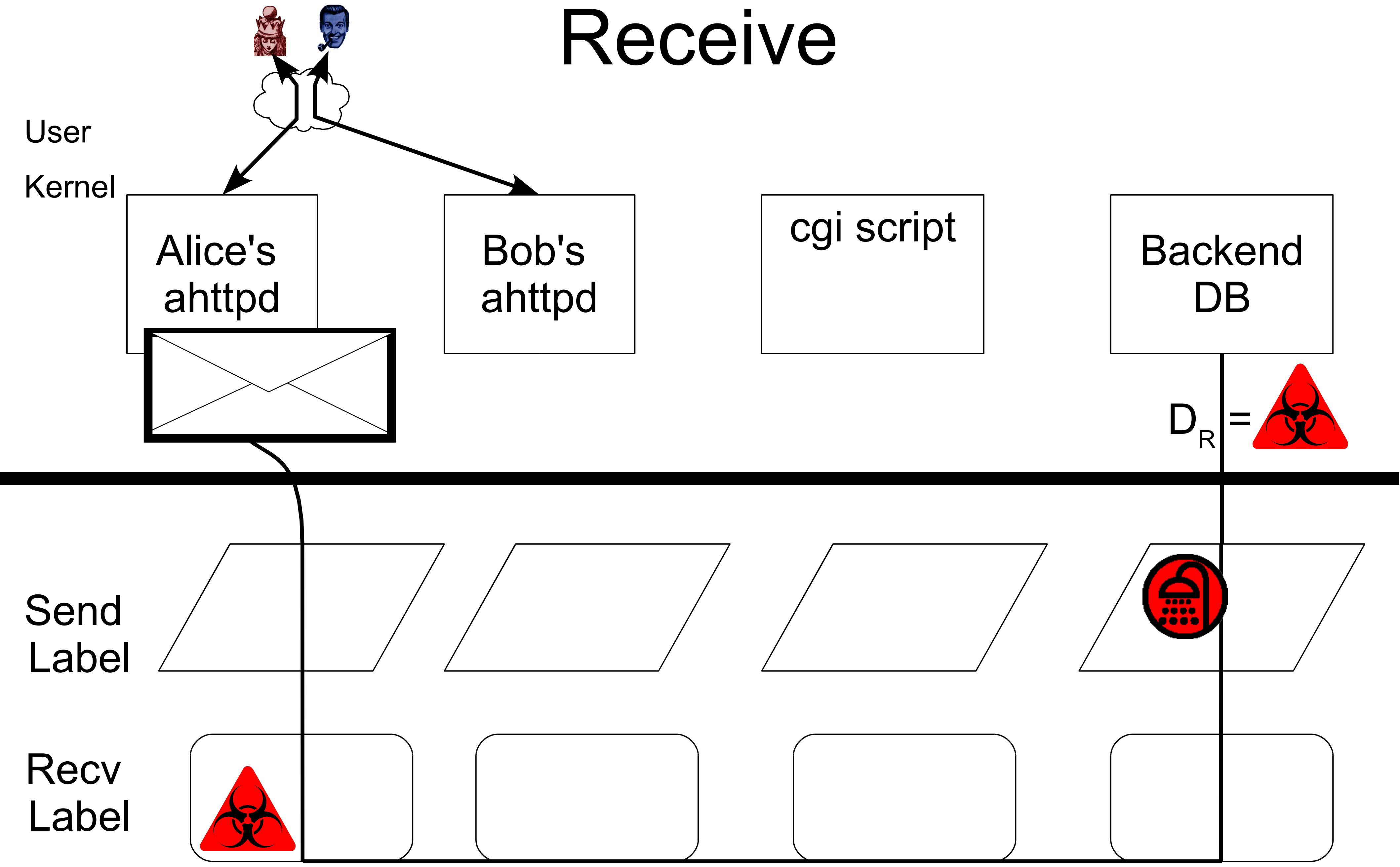
Declassify Receive



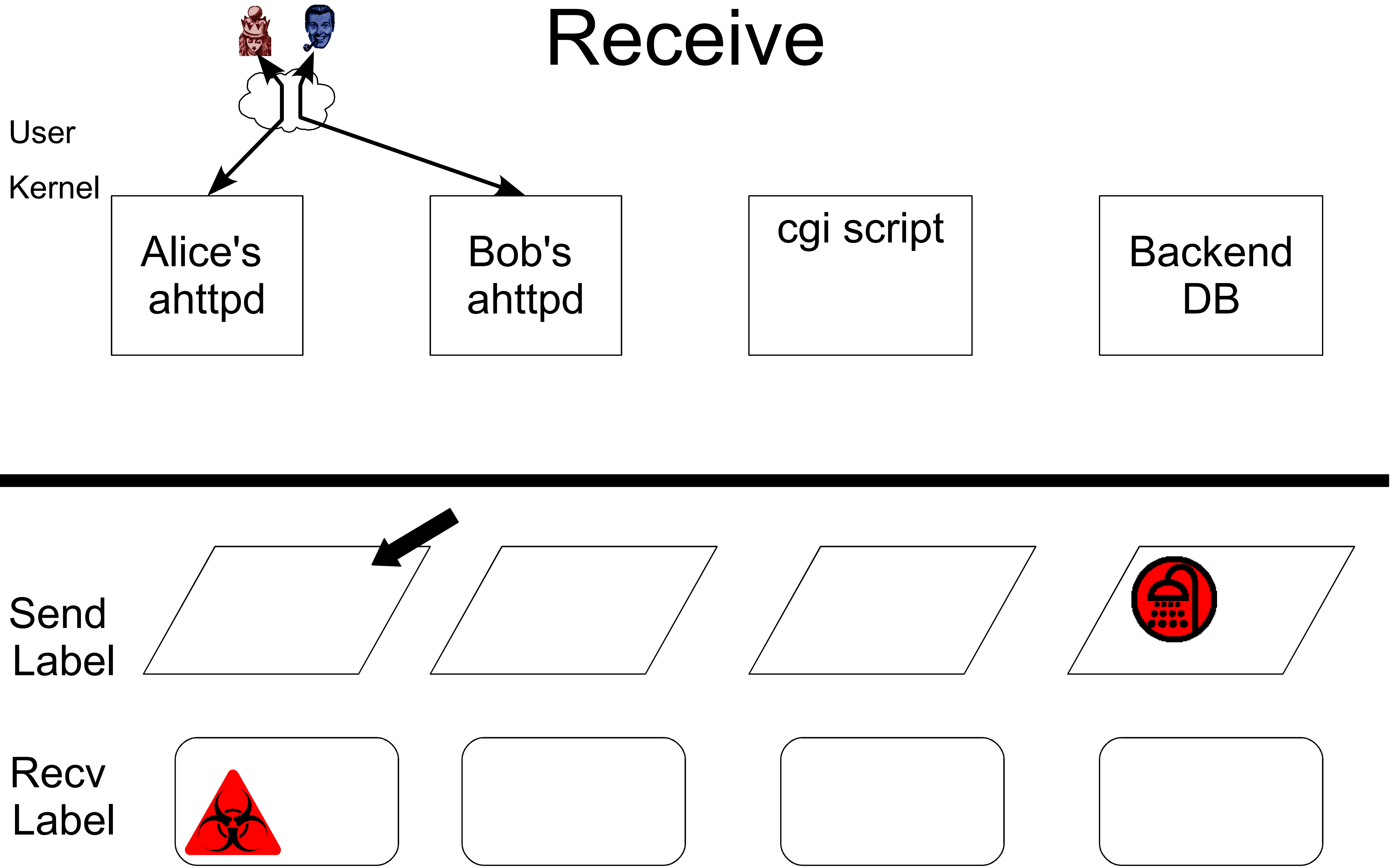
Declassify Receive



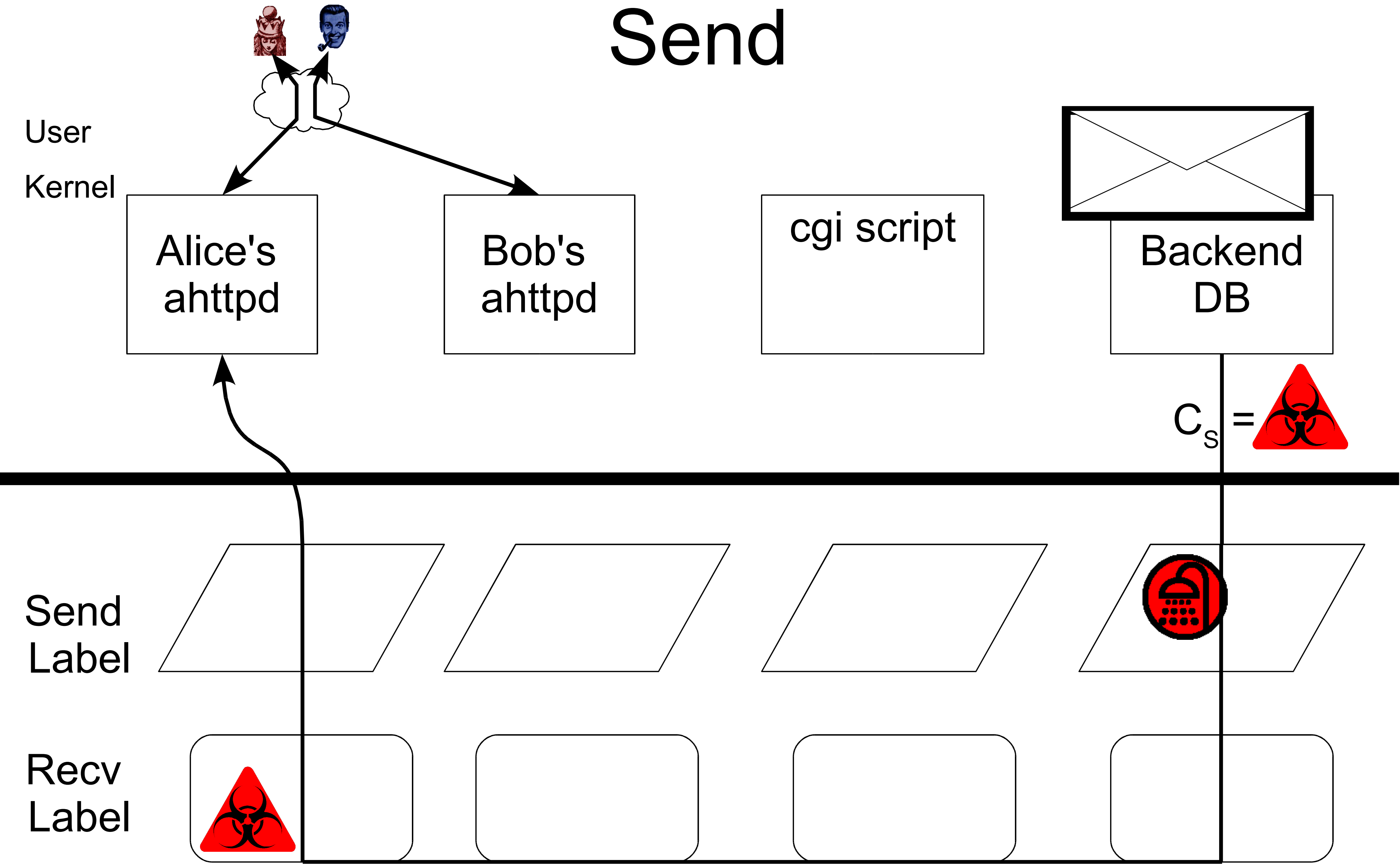
Declassify Receive



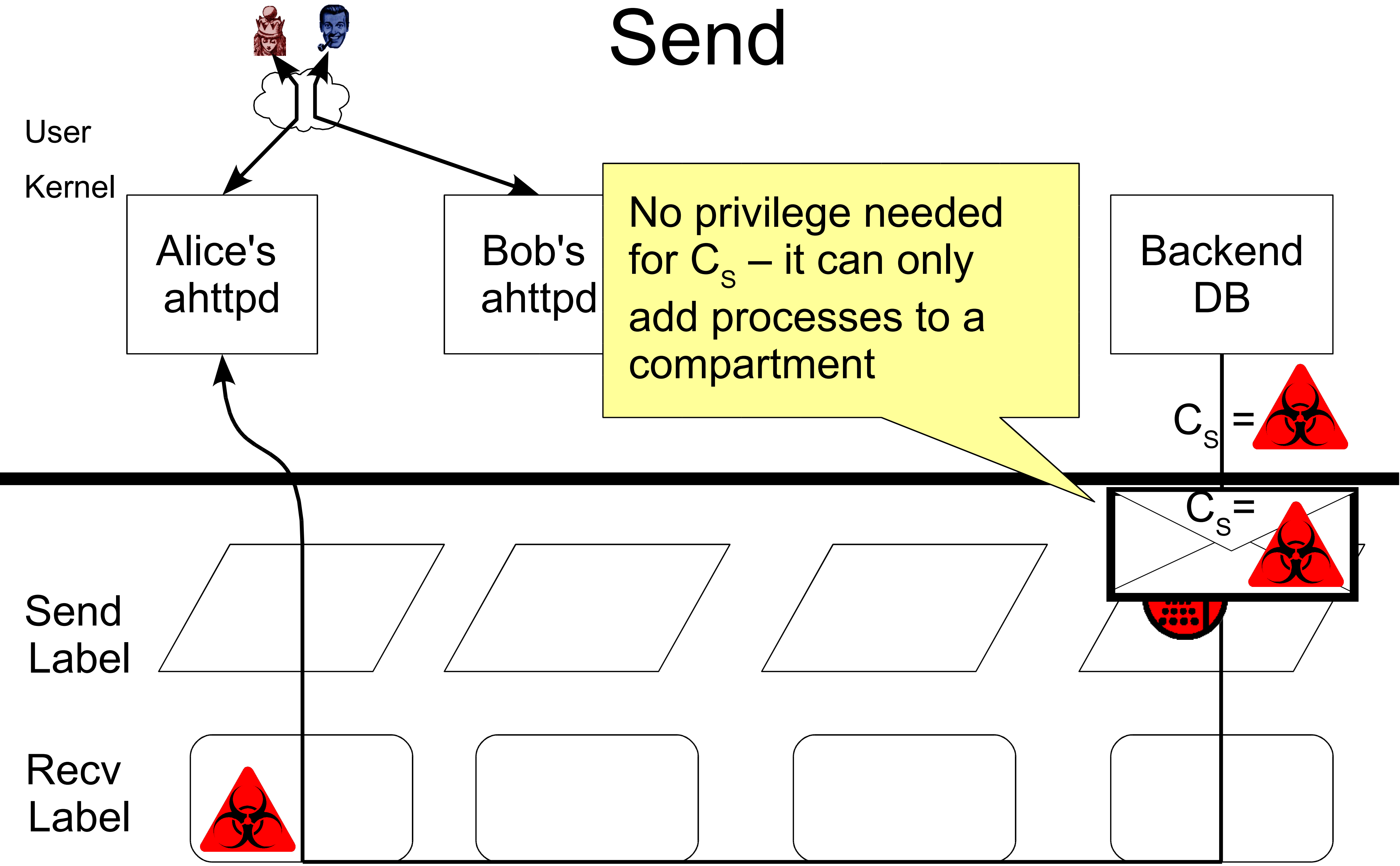
Declassify Receive



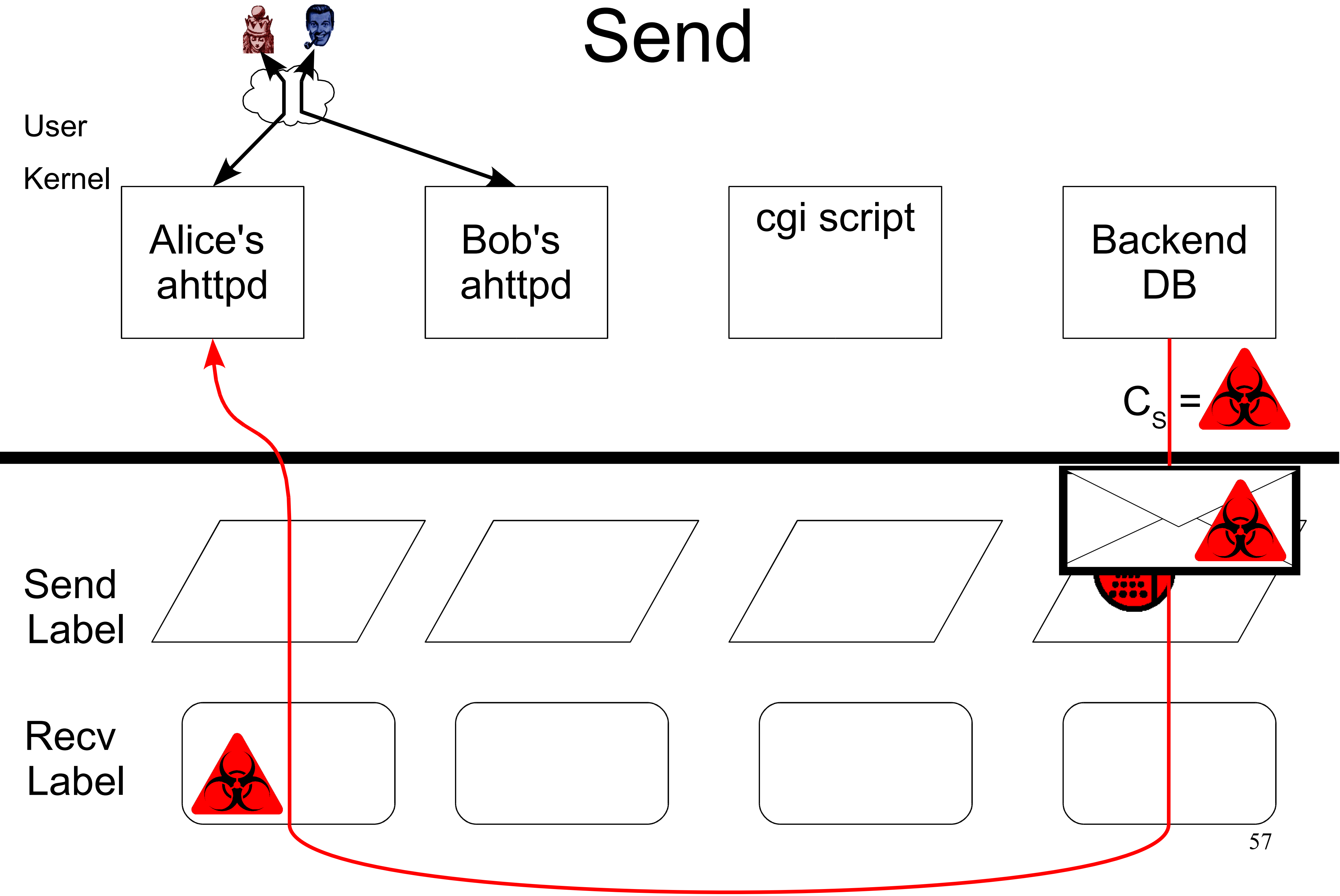
Contaminate Send



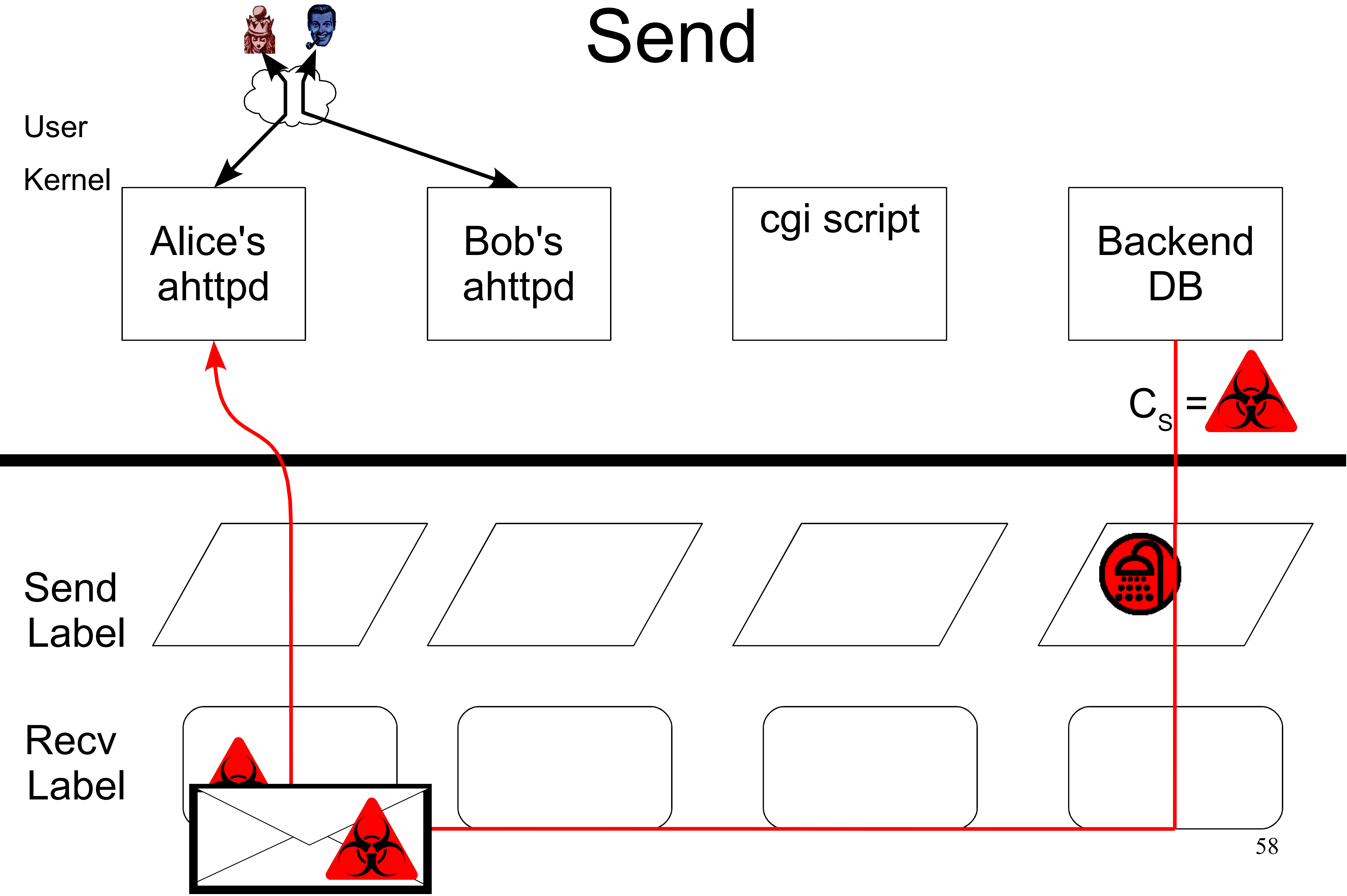
Contaminate Send



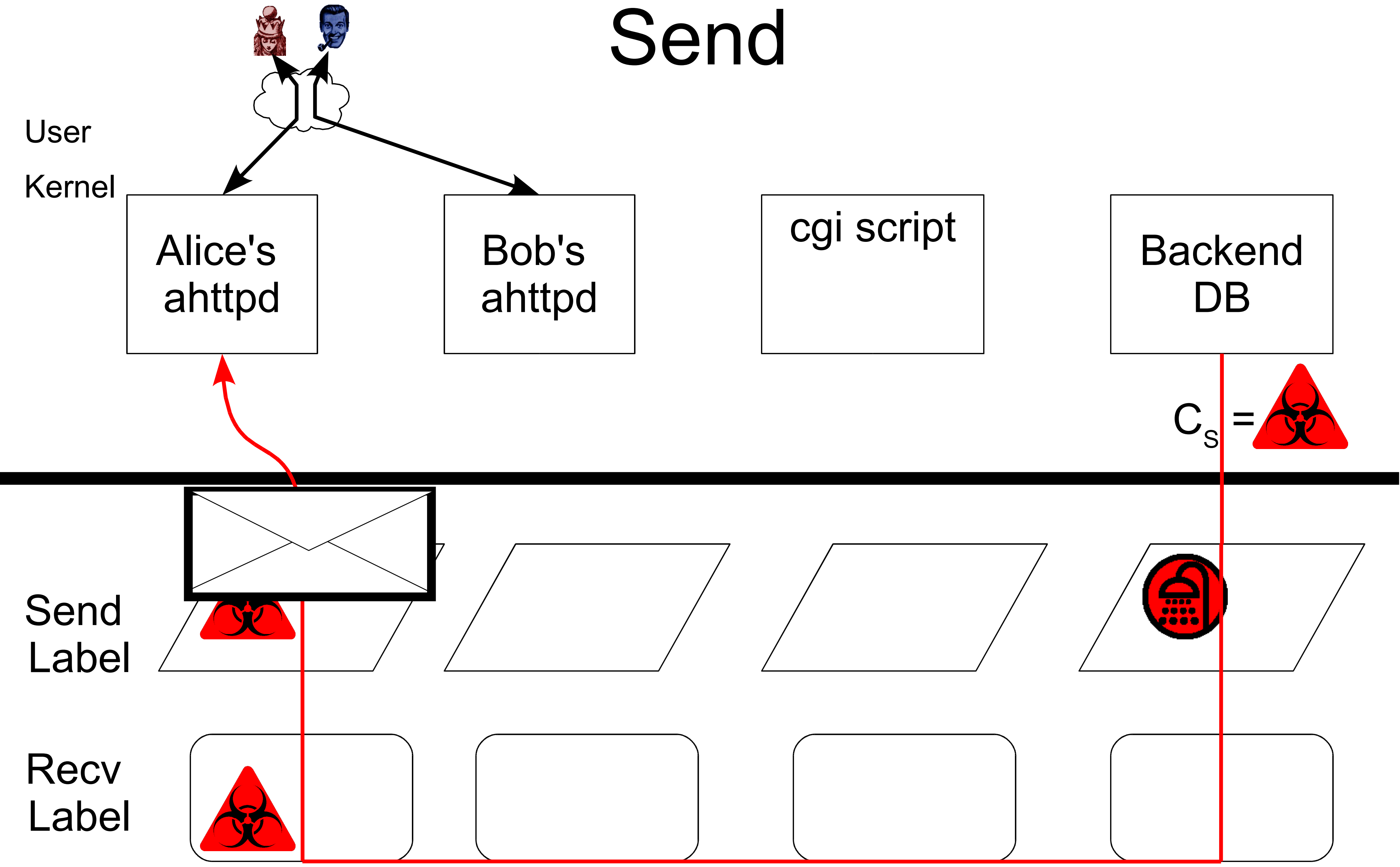
Contaminate Send



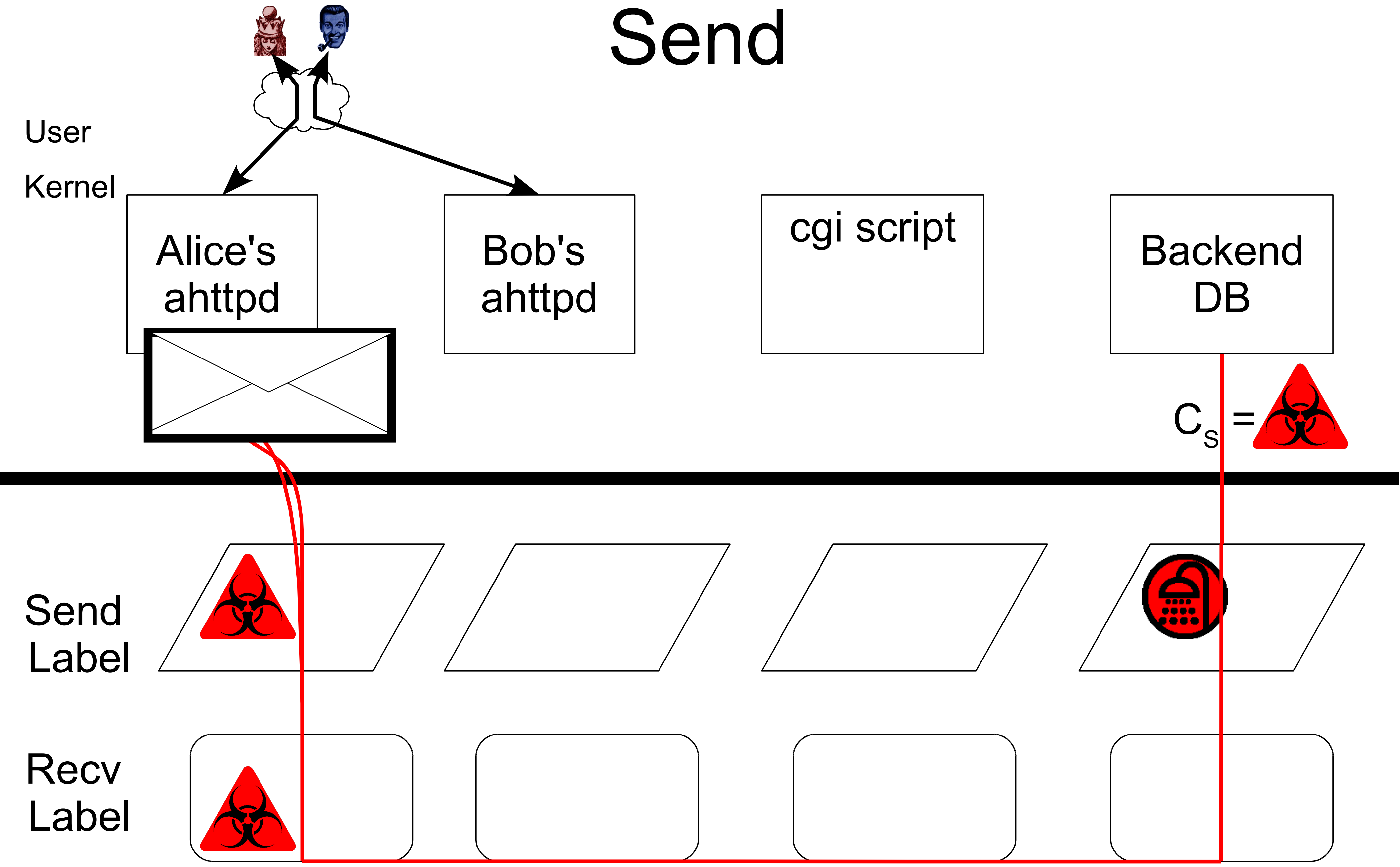
Contaminate Send



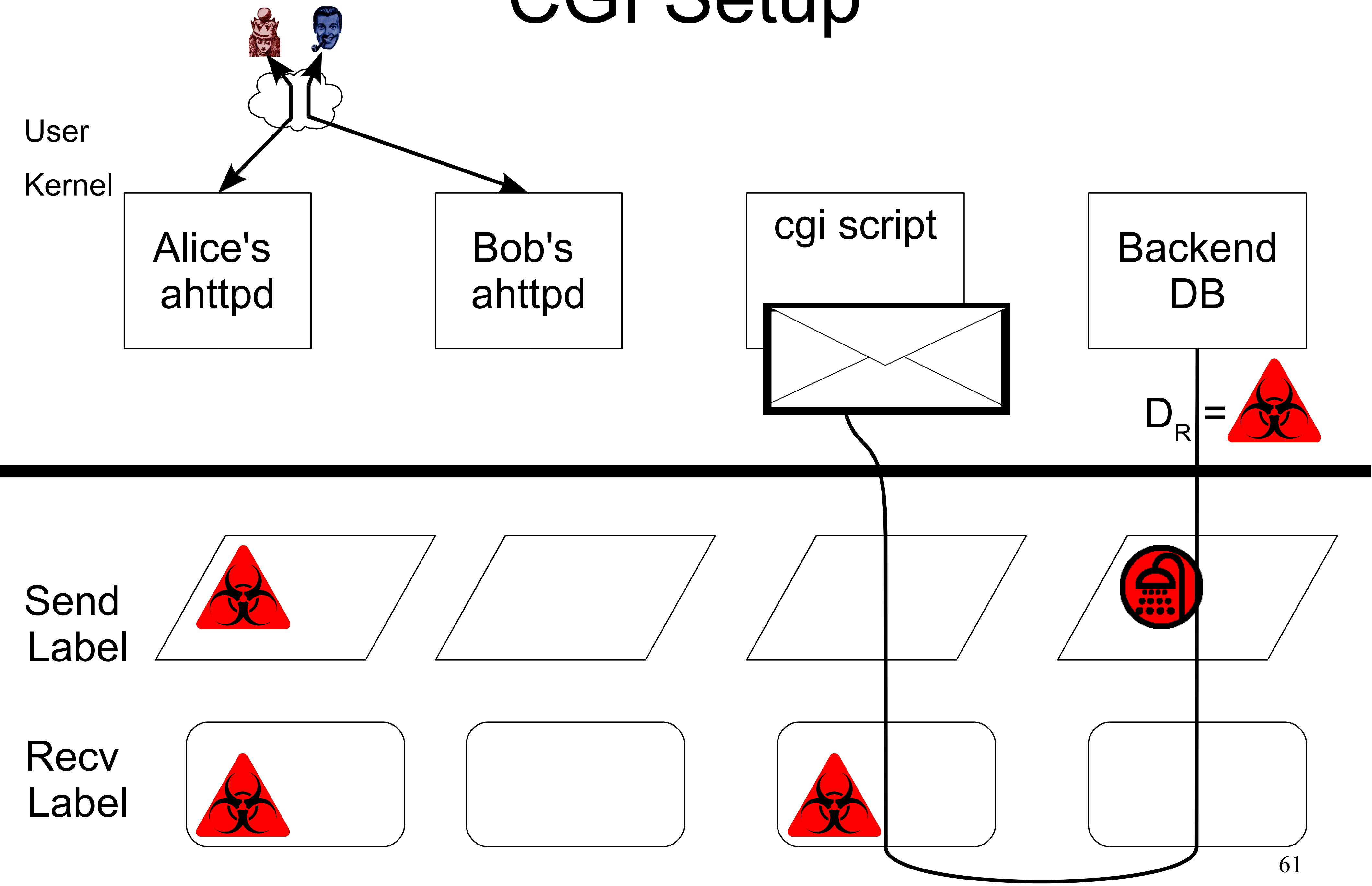
Contaminate Send



Contaminate Send

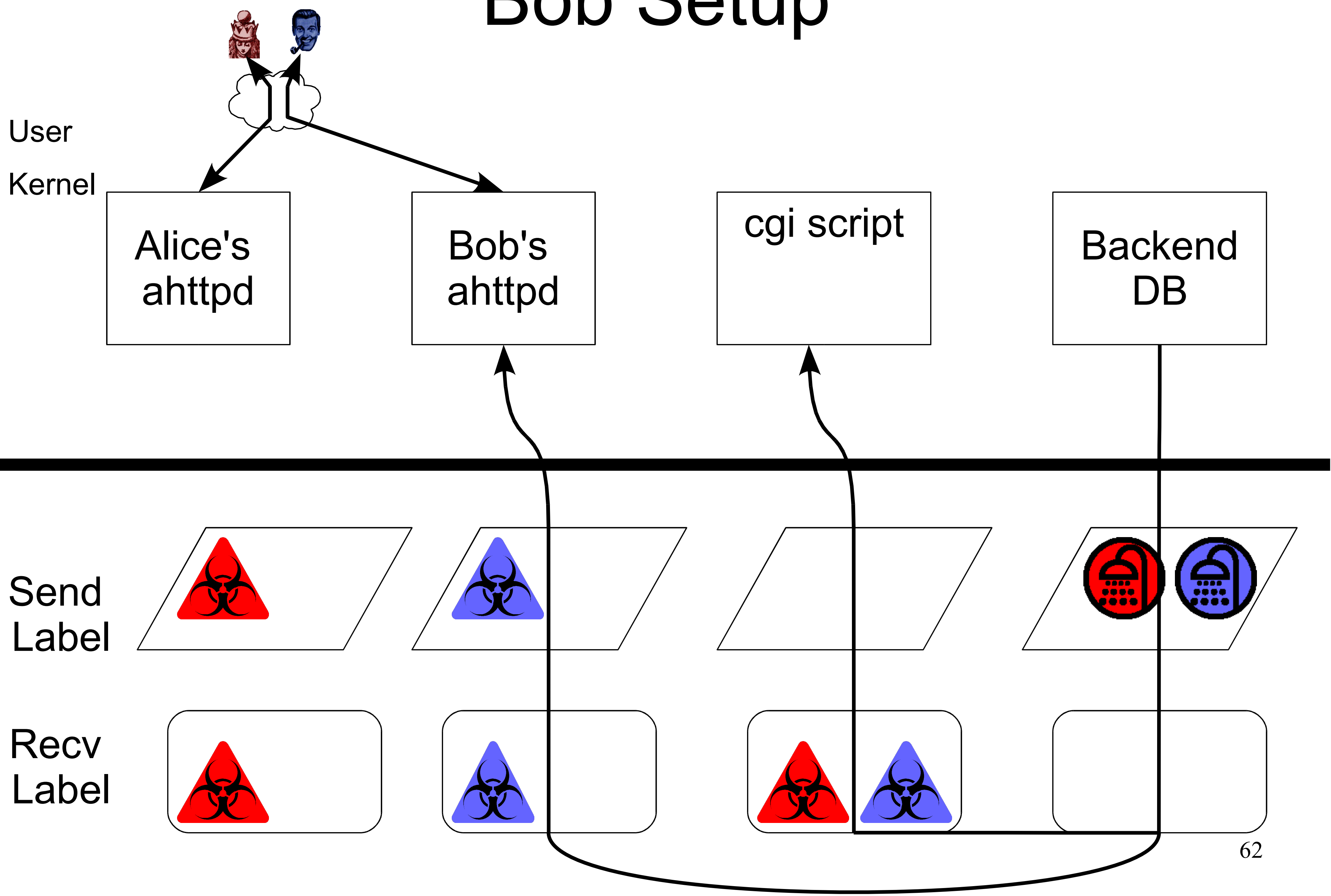


CGI Setup



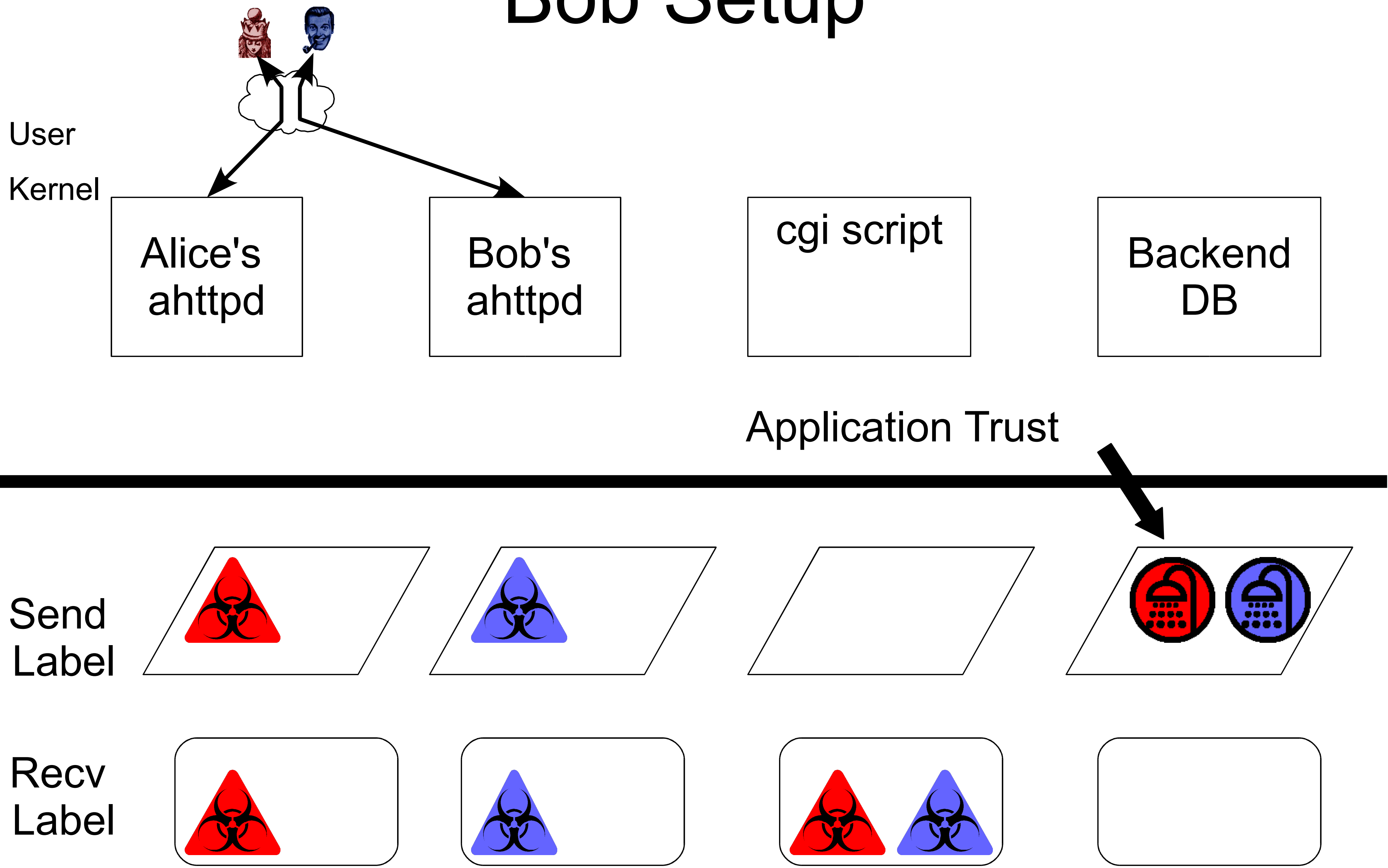
User
Kernel
Alice's ahttpd
Bob's ahttpd
cgi script
Backend DB
D_R =
Send Label
Recv Label

Bob Setup

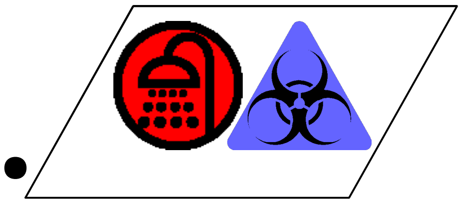


User
Kernel
Alice's ahttpd
Bob's ahttpd
cgi script
Backend DB
Send Label
Recv Label

Bob Setup


User
Kernel
Alice's ahttpd
Bob's ahttpd
cgi script
Backend DB
Application Trust
Send Label
Recv Label

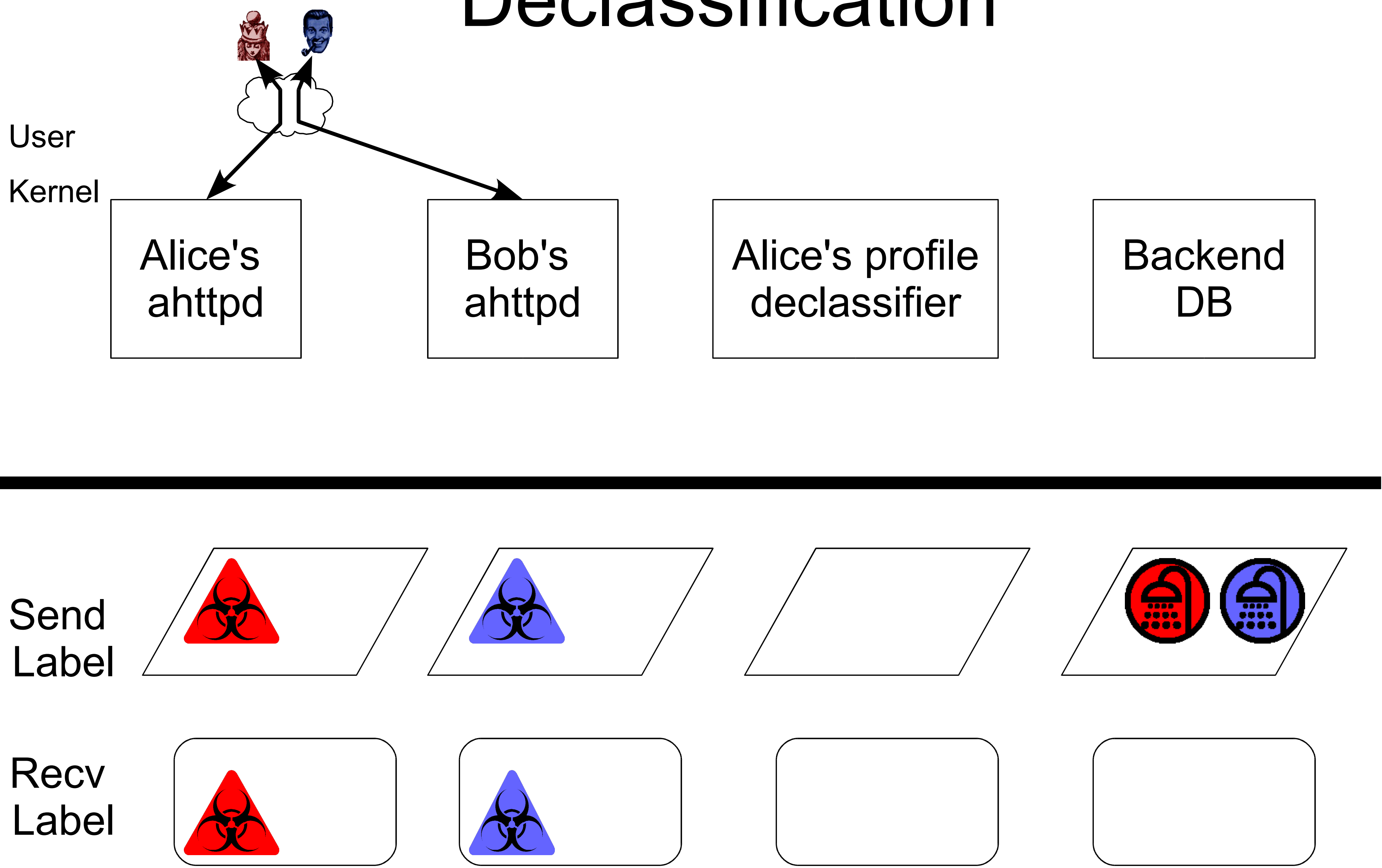
Label Implementation

- Contamination & Privilege = Label level (*, 0-3)
-  = {A *, B 3, 1}
- A & B are compartment names
- Trailing 1 = Neutral in all other compartments

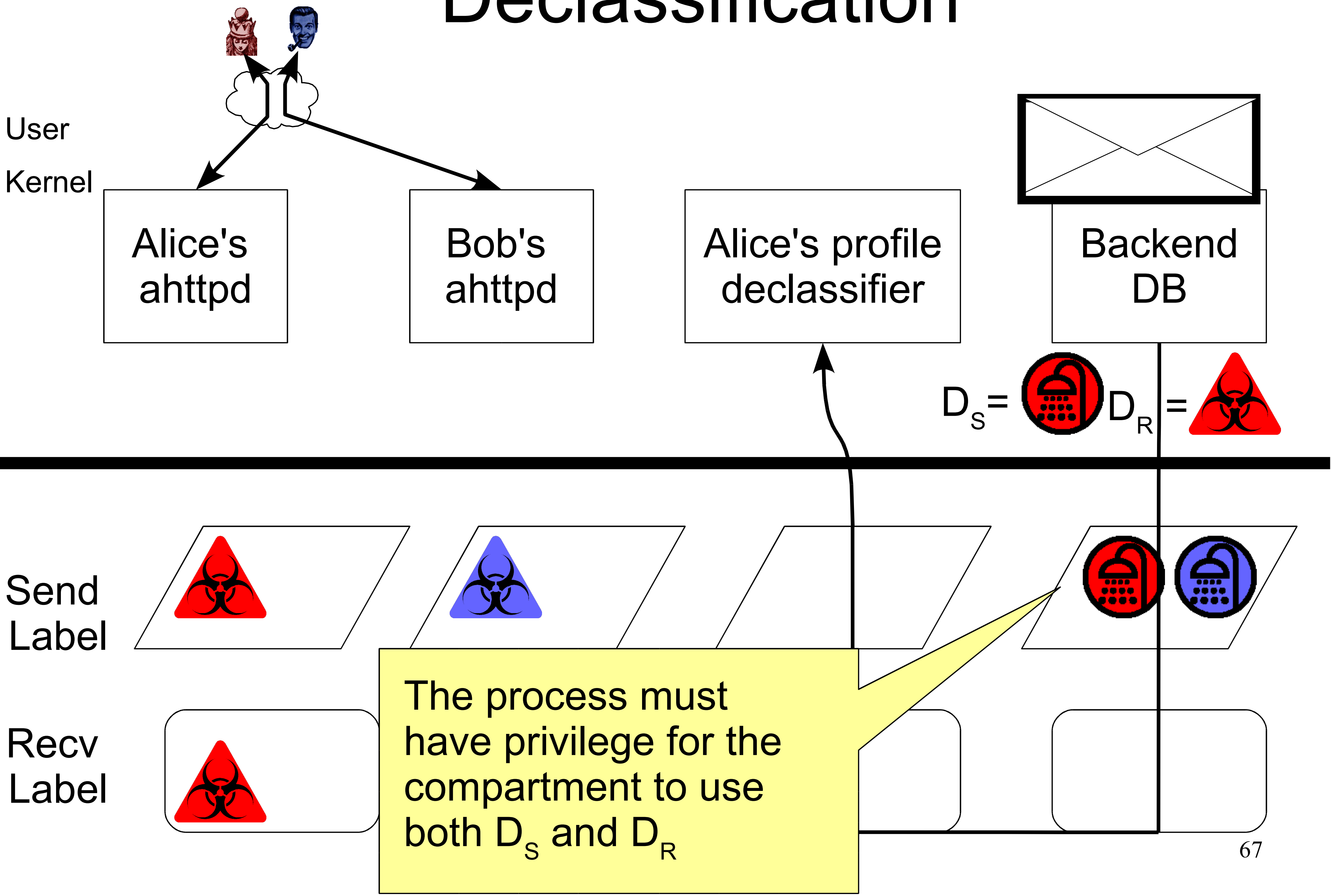
Declassification

- Information flow control keeps users data completely disjoint
- Alice wants to export **some** of her data, like her profile
  - But **all** her data is in her compartment
- How can she safely declassify her data?
- Alice must trust all processes that can do so
- To minimize declassification bugs, we build declassifiers as simple, single purpose programs

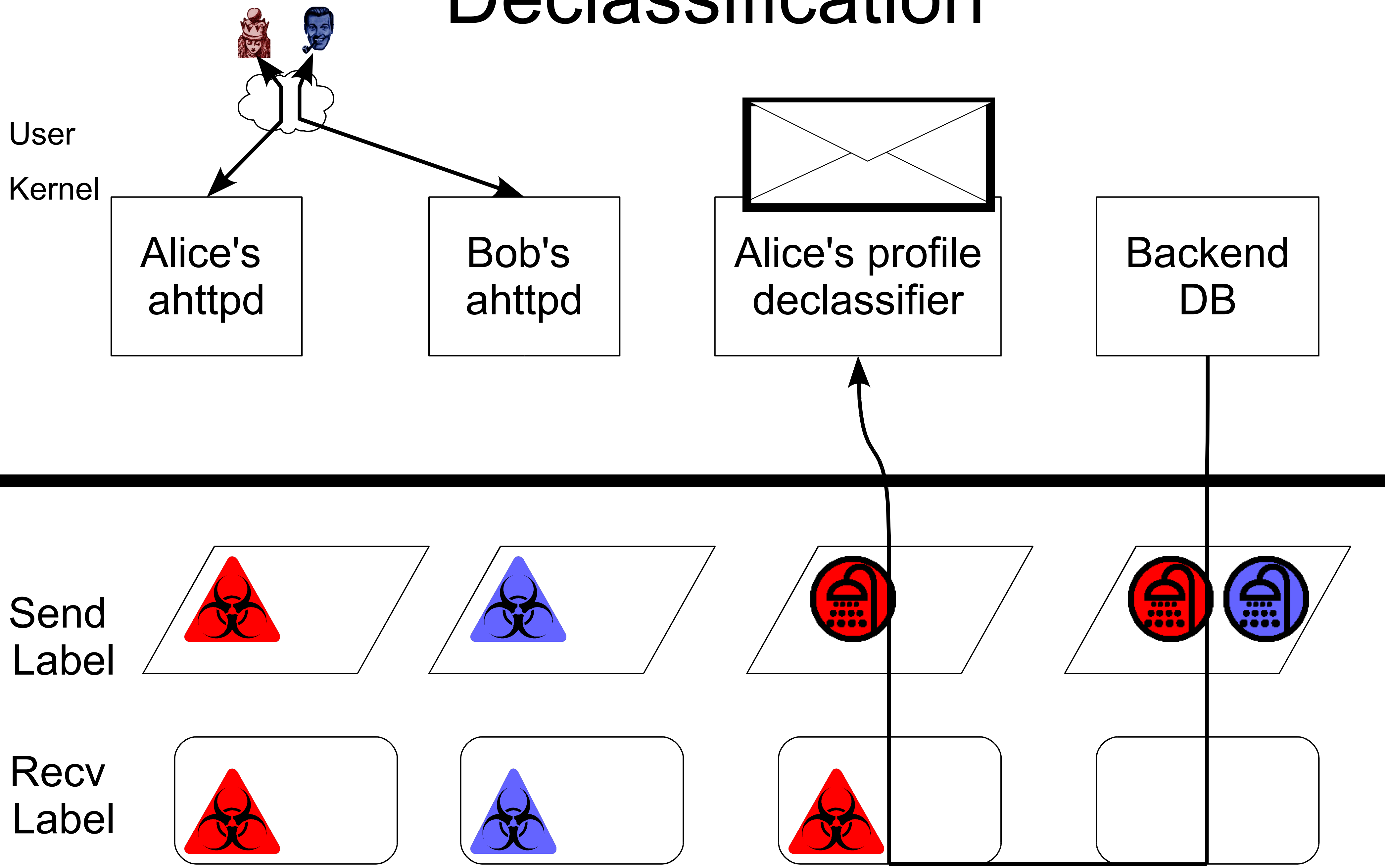
Declassification



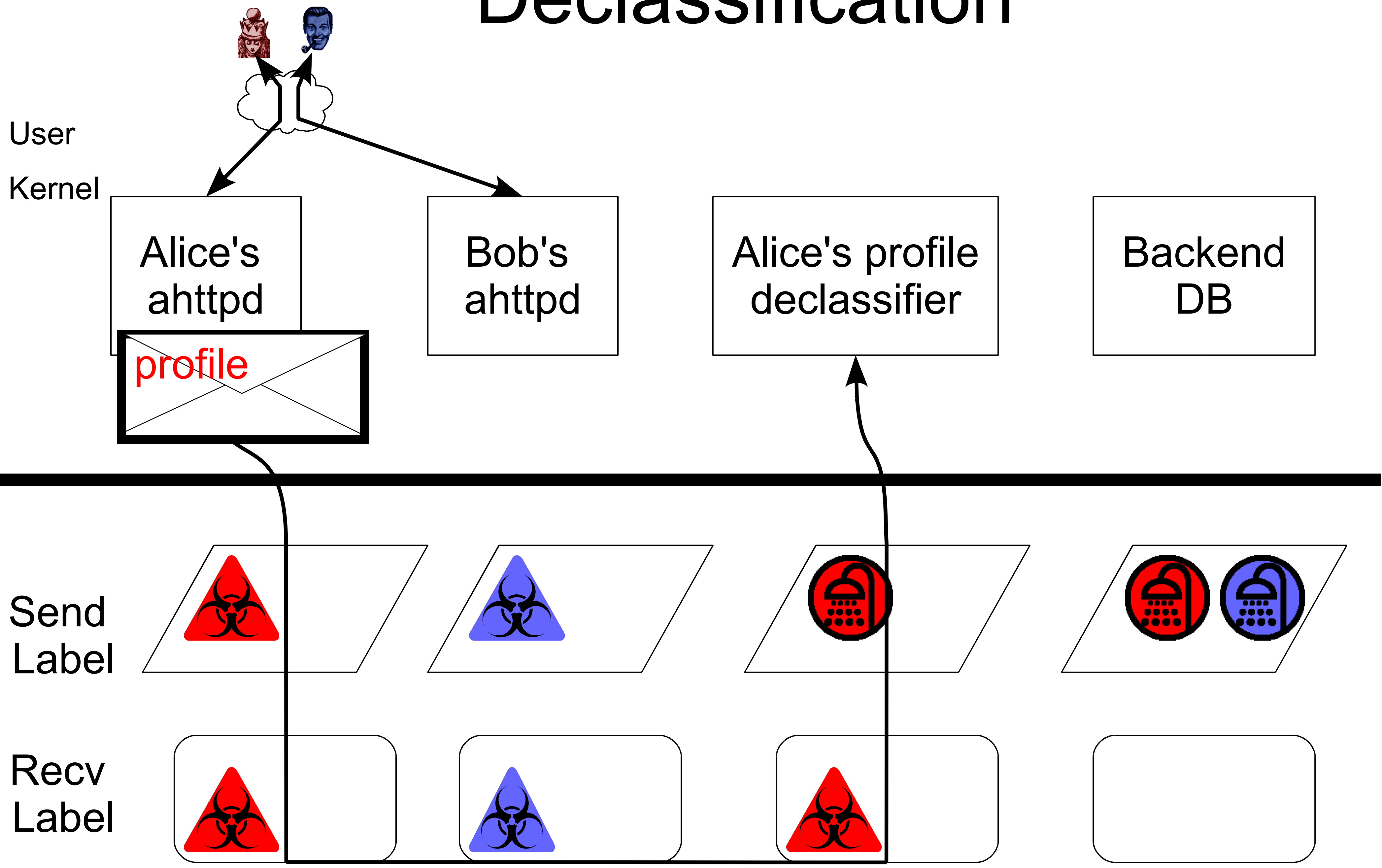
Declassification



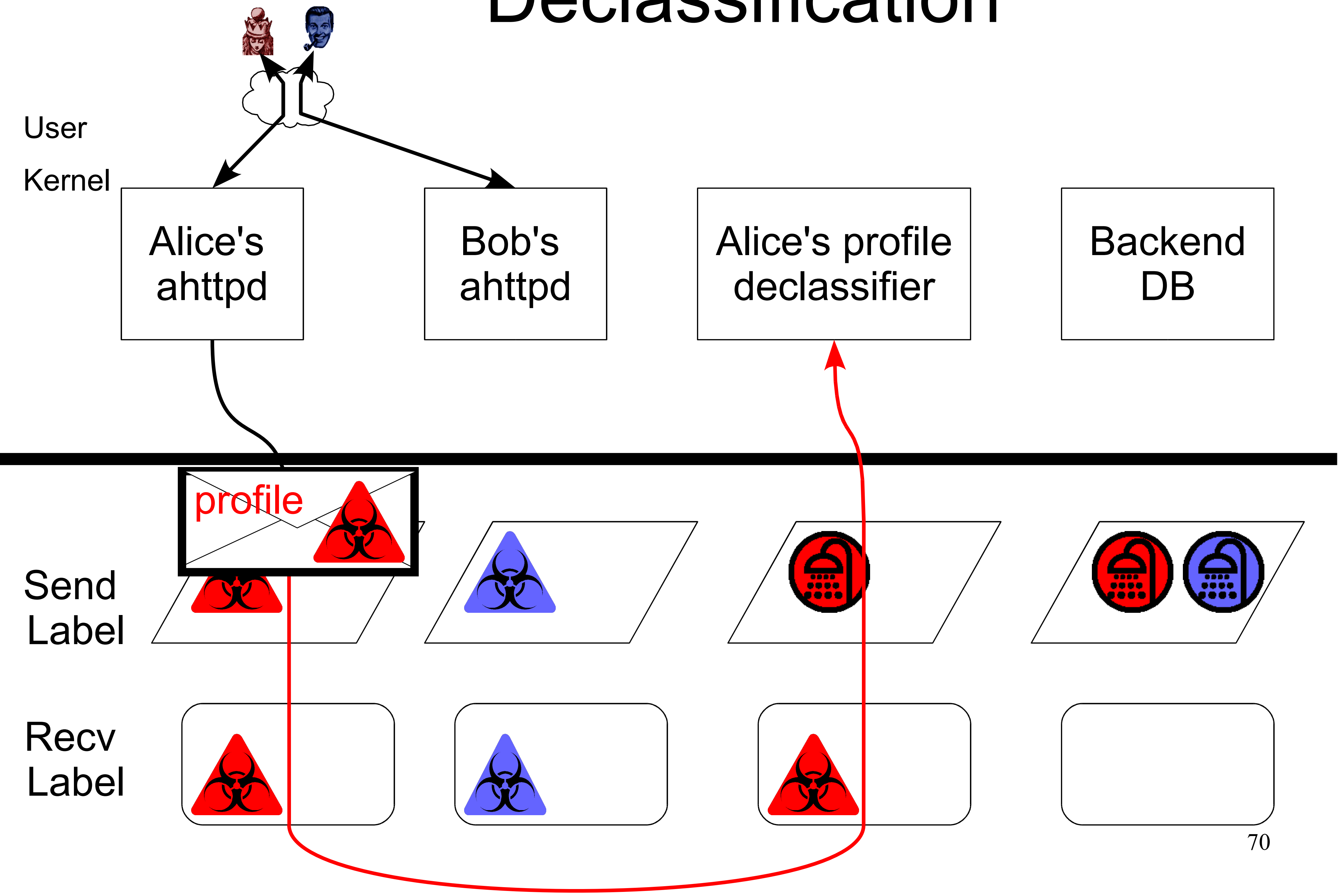
Declassification



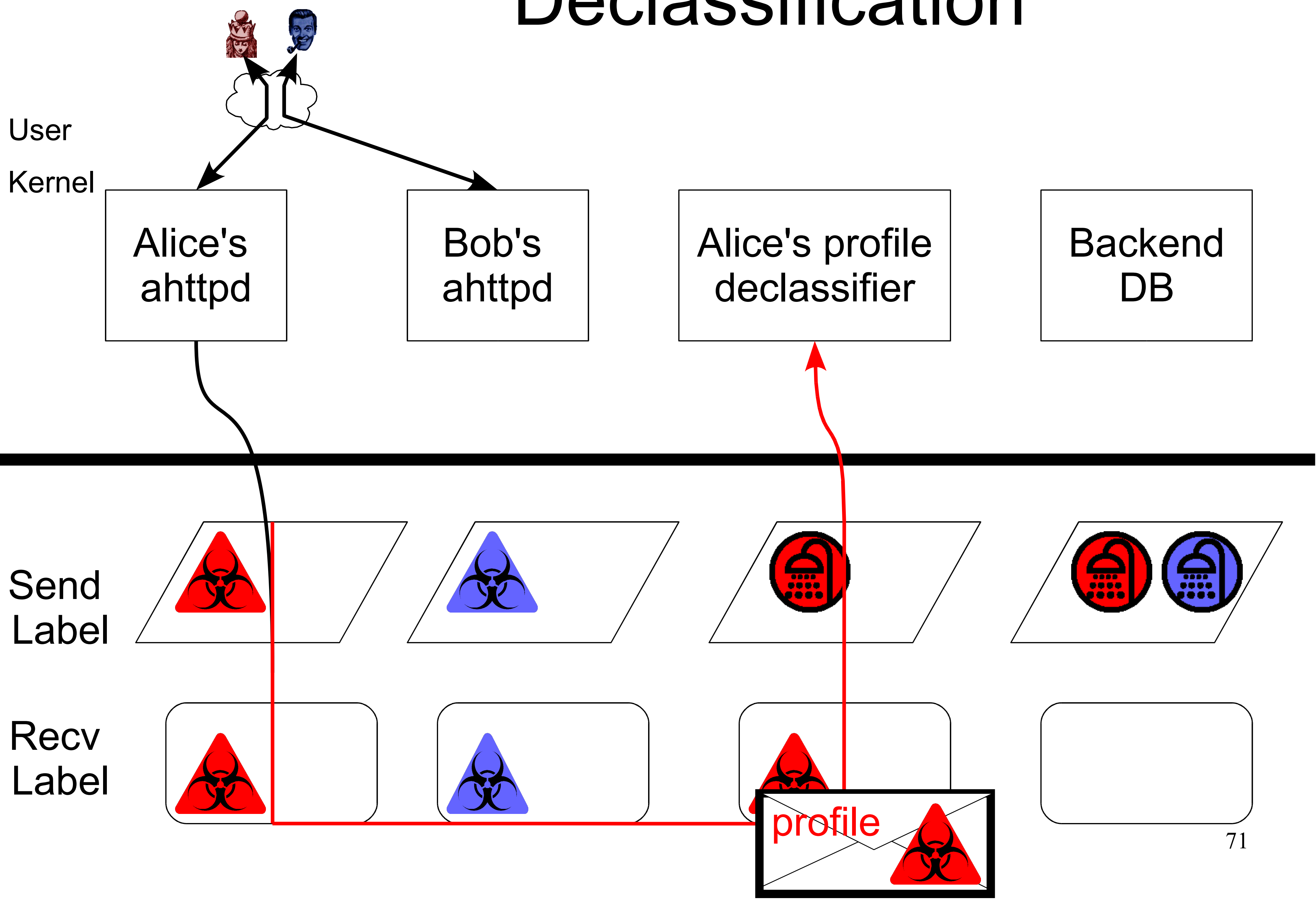
Declassification



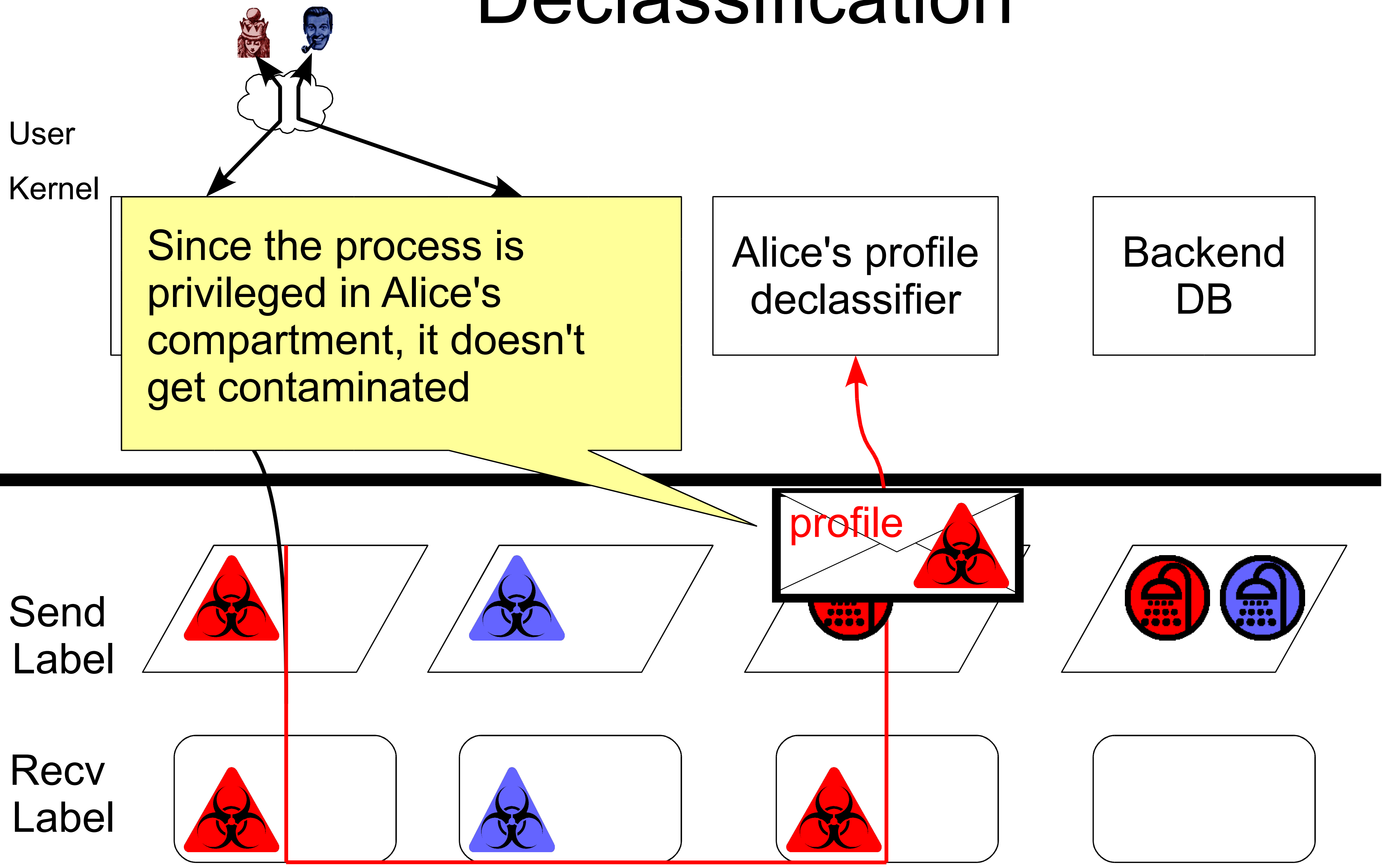
Declassification



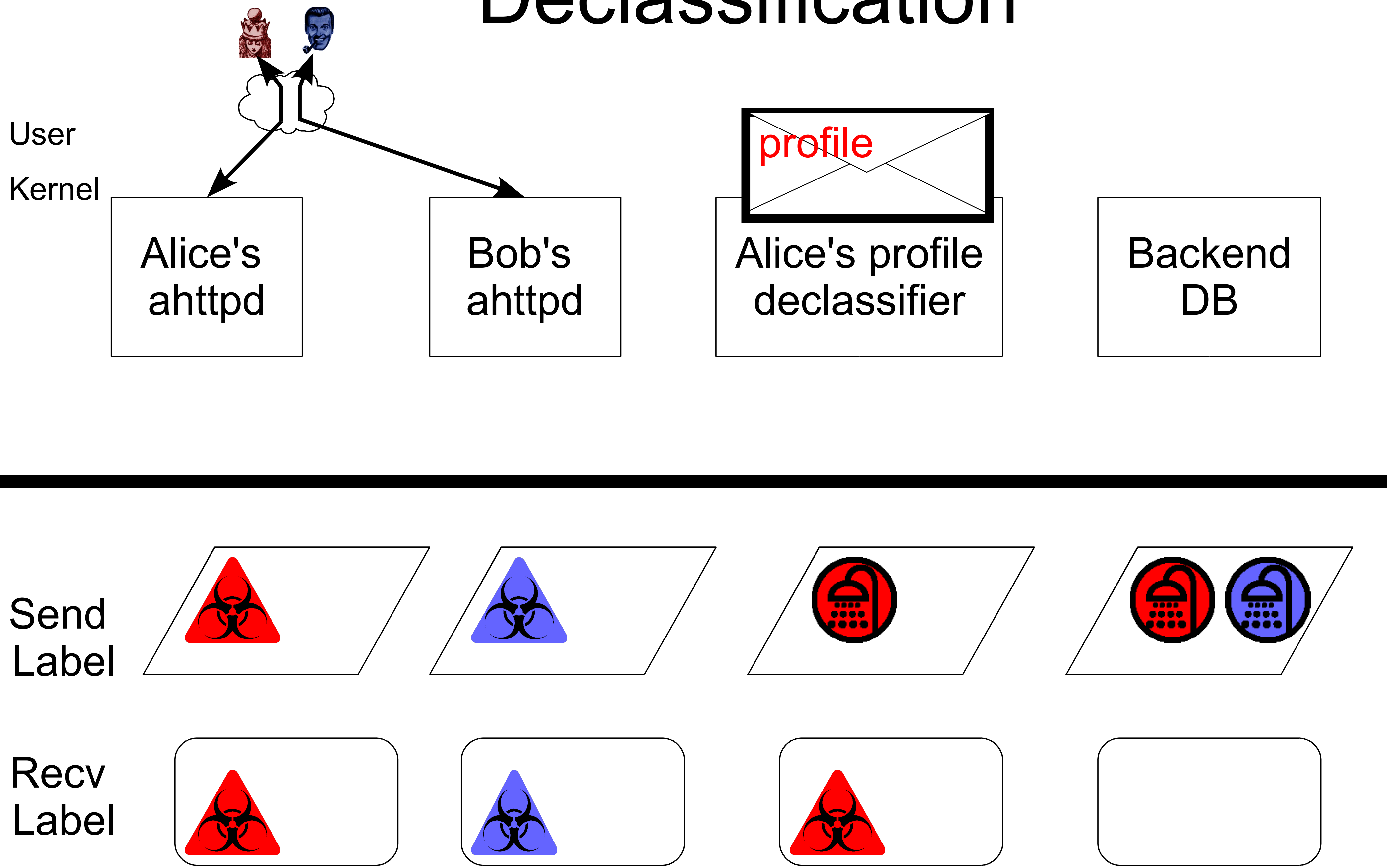
Declassification



Declassification



Declassification



Other Label Features

- Verify label on messages
  - Allows a process to prove it has labels at specific levels
- Integrity tracking
  - Enabled by level 0
- Different default level for send & receive labels
  - Enables interesting isolation policies

Preventing Contamination

- Ports
  - Associated with receive label
  - Verification imposed by receiver
  - Deny decontamination of receive labels beyond certain point
  - Receiver can grant rights to processes to send
  - Prevents arbitrary processes from sending to it

Combating Process Over-Contamination

- One process per user per service
  - Lots of heavy weight context switches
  - Lots of memory
- Combine processes to get one process per service?
  - Become too contaminated to function
  - Or **too** privileged
- Many processes are similar
- Programming style help?

Event Loop

```
while (1) {
    event = get_next_event();
    user = lookup_user(event);
    if (user not yet seen)
        user.state = create_state();
    process_event(event, user);
}
```

- State isolated to data structures
- Stack not used from event to event
- Execution state has nice preemption points

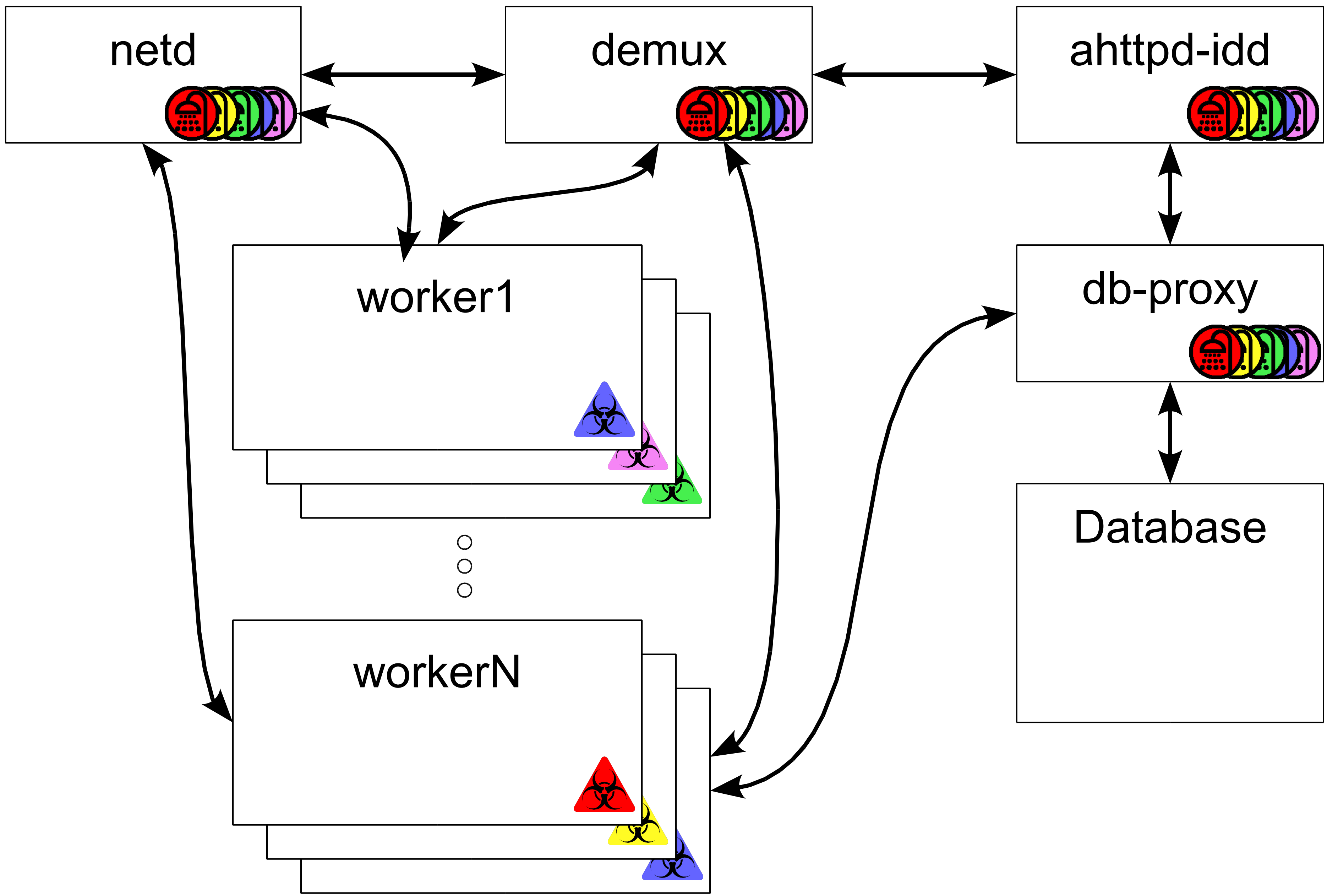
Event Process Abstraction

l

```
ep_checkpoint(&msg);
if (!state.initialized) {
    initialize_state(&state);
    state.reply = new_port();
}
process_message(&msg, &state);
ep_yield(); // revert to chkpointed memory
```

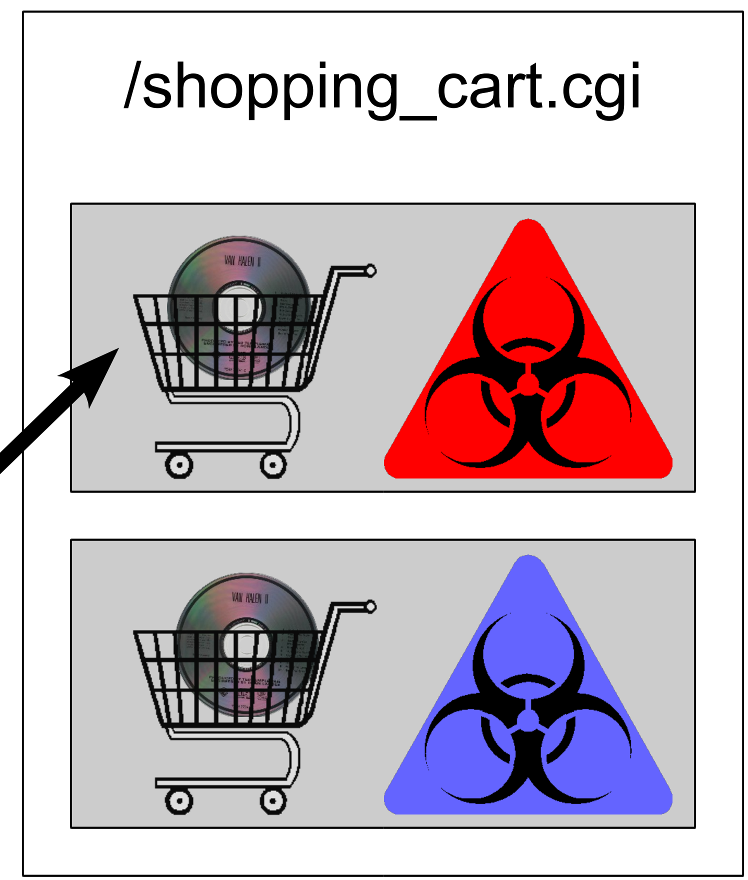
- Fork memory state for each new session
    - Memory isolation is the same as fork
    - Small differences anticipated, stored efficiently (diff)
- Event loop allows shared execution state
    - Allows light weight context switches

Web Server Architecture

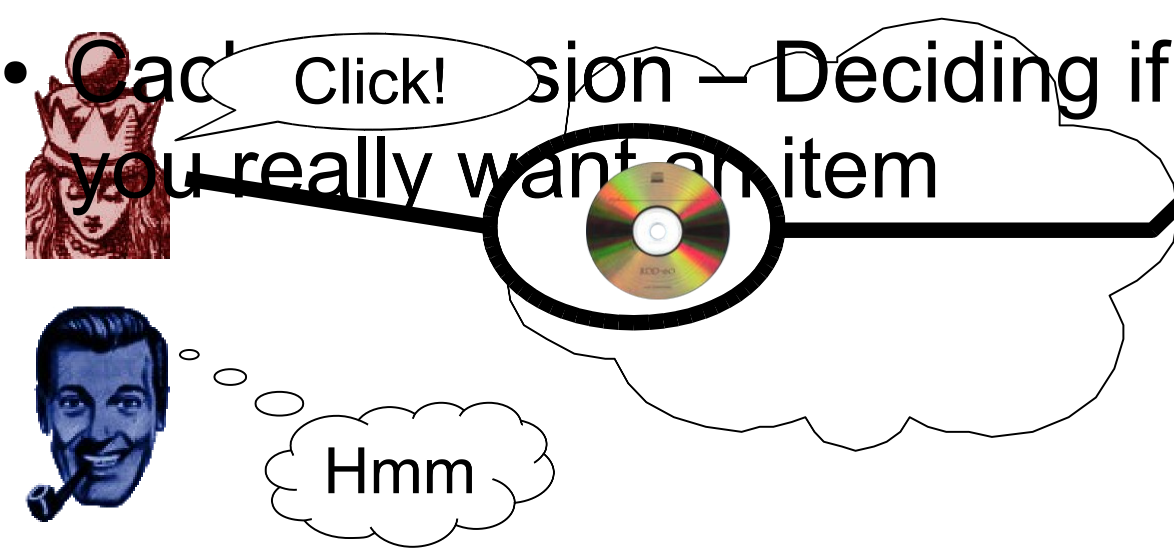


Experimental Setup – Memory

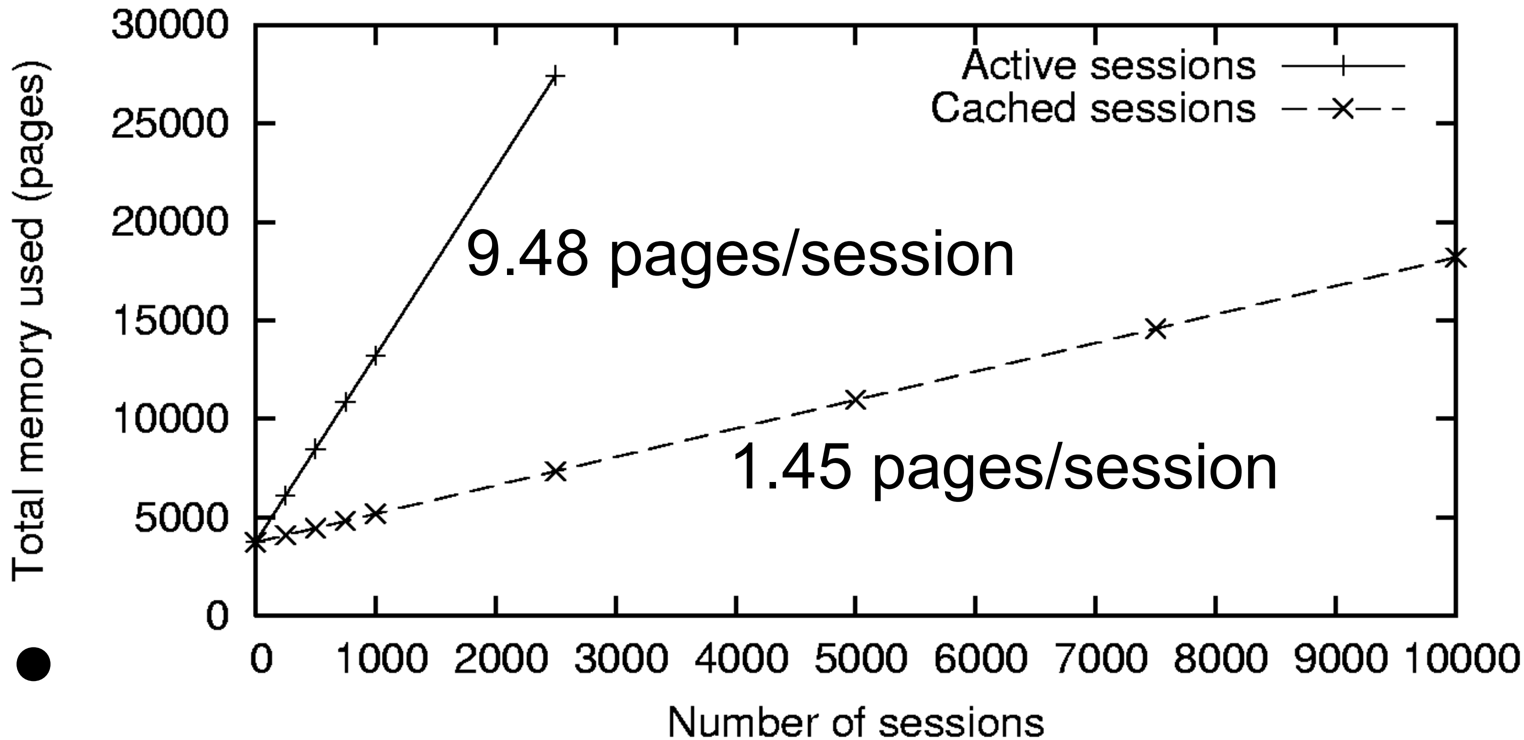
- How much memory do event processes use?
- Shopping cart application
  - Session state stored in event process
  - One event process per user



- Active session – Adding an item to the shopping cart
- Cache session – Deciding if you really want an item



Event Processes Conserve Memory

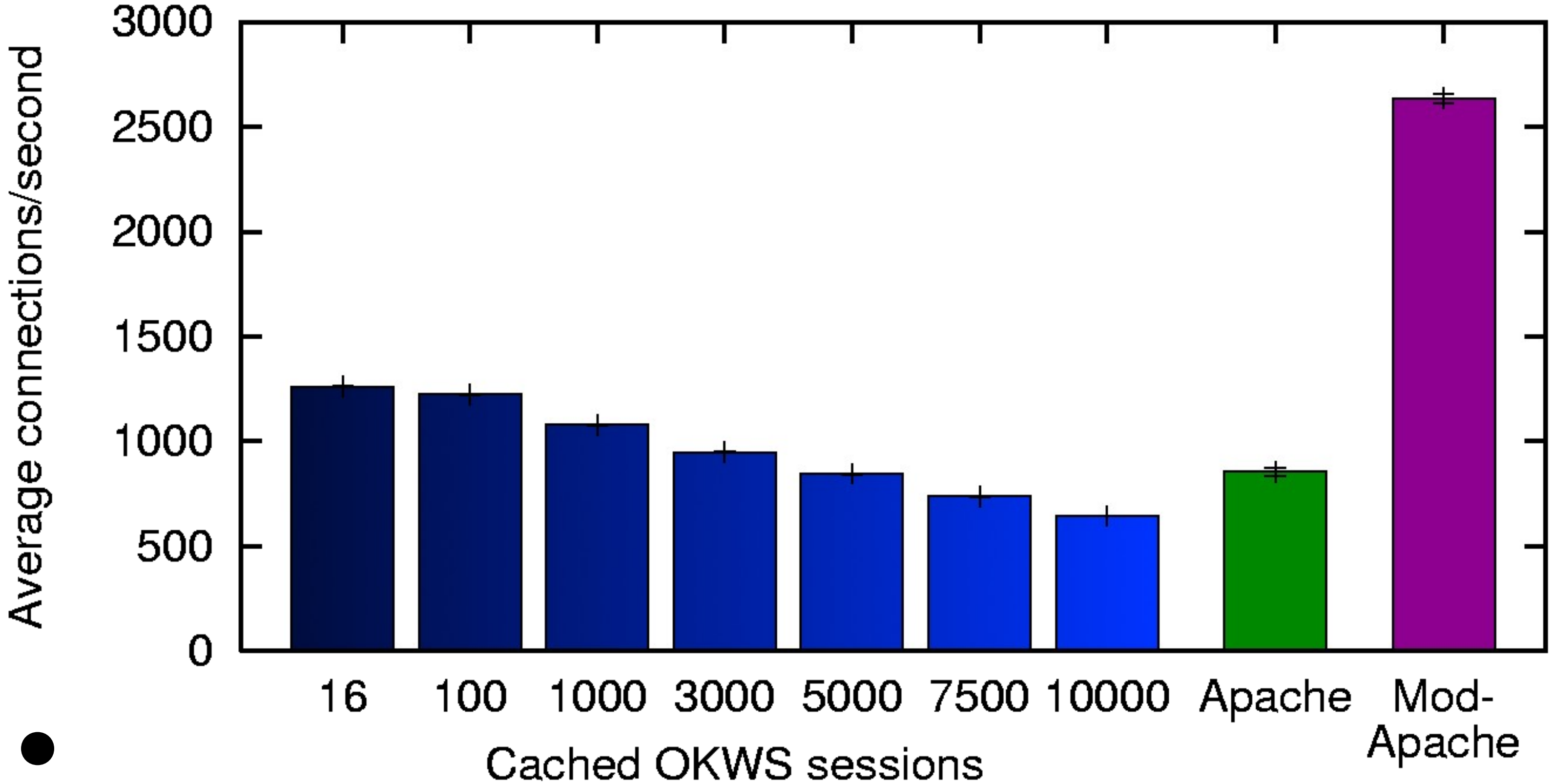


- Includes user and kernel memory
- Not too many active sessions on a large website

Experimental Setup – Throughput

- Simple character generation service
  - Not interested in application overhead
  - One event process per session (user)
- Compare to Apache & Mod-Apache
  - Varied concurrency to get best case performance
- Apache
  - Service runs as a CGI script
  - Connections are isolated into processes
  - Processes are not isolated or jailed on the system
- Mod-Apache
  - Service runs inside Apache process

Good Throughput



- For 16 sessions, 150% of Apache
- For 10,000 session, 75% of Apache

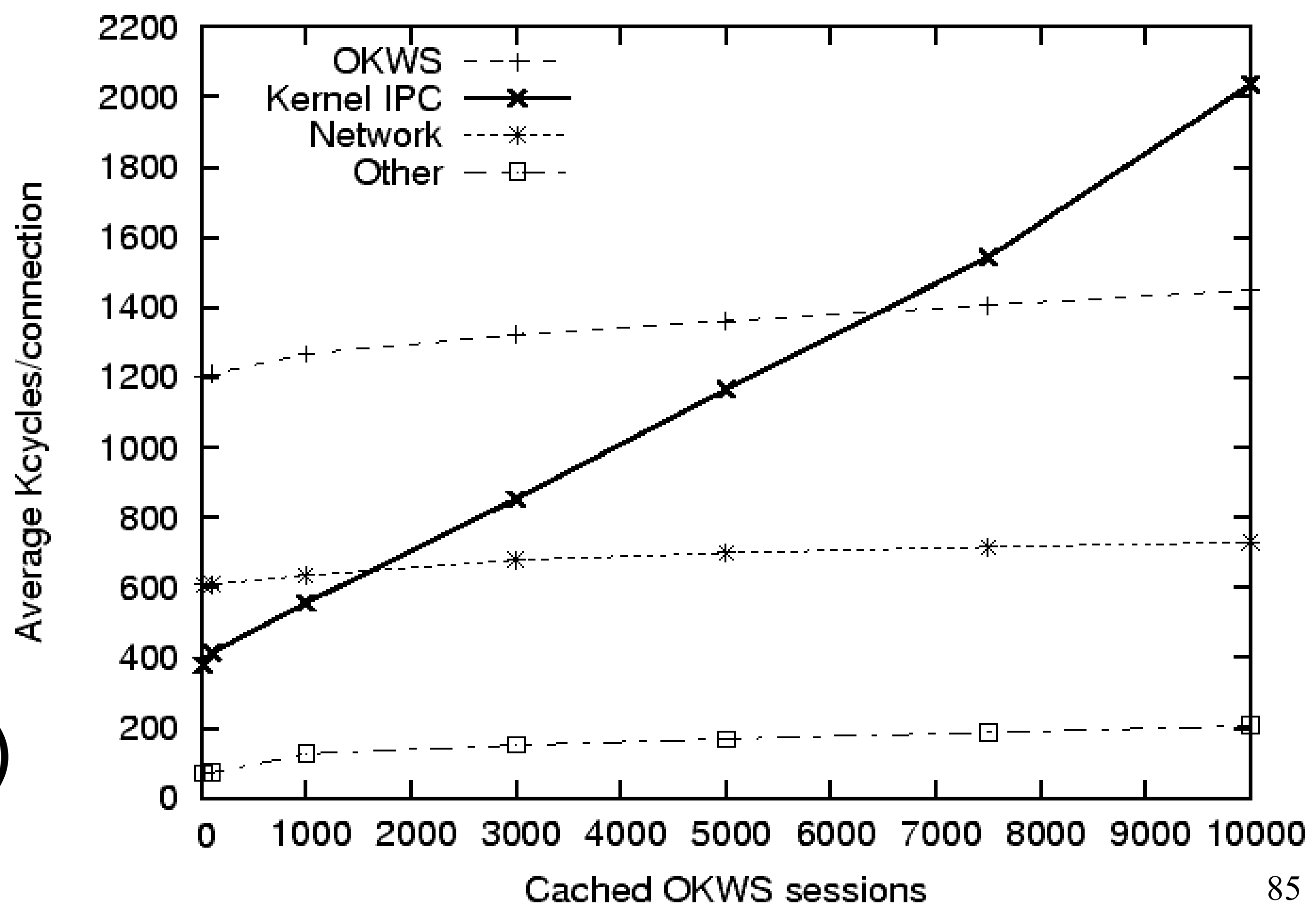
Latency

Server	Latency (μs) Median	90th Percentile
Mod-Apache	999	1,015
Apache	3,374	5,262
OKWS, 1 session	1,875	2,384
OKWS, 1000 sessions	3,414	6,767

Figure 8: The median and 90th percentile latencies of requests to various server confi gurations.

Label Cost Linear in Label Size

- Label cost starts small but outstrips OKWS cost around 6500 sessions
- Declassifiers label size O(#sessions)



Conclusion

- Asbestos labels make MAC more practical
  - Labels provide decentralized compartment creation & privilege
  - Event processes avoid accumulation of contamination
- The OK web server on Asbestos
  - Performs comparably to Apache
  - Provides better security properties than Apache