Virtual Memory:
Mach and Asbestos

Presented by Hakim Weatherspoon

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

Richard Rashid, Avadis Tevanian, Michael Young, David Golub, Robert Baron, David Black, William Bolosky, and Jonathan Chew

- Richard Rashid
  - Lead developer of Mach
  - Microsoft Research
- William Bolosky
  - Microsoft Research

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

Takeaway

- Hardware-independent virtual memory (VM) is not only possible, but can be elegant
  - Hardware dependent structures contained to pmap
  - VM functionality can be delegated to user process
  - Mach works with uniprocessors, multiprocessors, One- and two- level page tables, and inverted page tables
- Lessons/Flaws
  - Macrobenchmark performance missing
  - Performance revisited over next 10+ years

Mach Virtual Memory

- 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 Abstractions

- Task
  - Basic unit of resource allocation
  - Virtual address space, communication capabilities
- Thread
  - Basic unit of computation
- Port
  - Communication channel for IPC
- Message
  - May contain port capabilities, pointers
- Memory Object

Virtual Memory 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 Map

- 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 Virtual Memory

- 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

Multiprocessor Issues

- TLB Consistency
  - Force interrupts to all CPU’s
  - Wait until timer interrupt
  - Temporarily allow inconsistency

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.

Perspective

- Achieved Goals
  - Sophisticated, hardware-independent VM system possible
  - Can achieve good (microbenchmark) performance
- Lessons/Flaws
  - Macrobenchmark performance missing
  - Performance revisited over next 10+ years

Labels and Event Processes in the Asbestos Operating SystemPetros

Petros Efstathopoulos, Maxwell Krohn, Steve VanDeBogart, Cliff Frey,

David Ziegler, Eddie Kohler, David Mazières, Frans Kaashoek, Robert Morris

- Frans Kaashoek and Robert Morris
  - MIT Faculty. Creators of Chord.
  - Academic father and grandfather to other authors and many more
- Maxwell Krohn
  - Creator of OK Cupid dating Service (ugrad @ Harvard)
  - Creator SFSLite and OK Web Server
- David Mazières
  - Stanford Faculty
  - Creator of SFS and libasync
- Eddie Kohler
  - UCLA Faculty
  - Creator of Click Modular Router
- Rest were students at MIT or UCLA

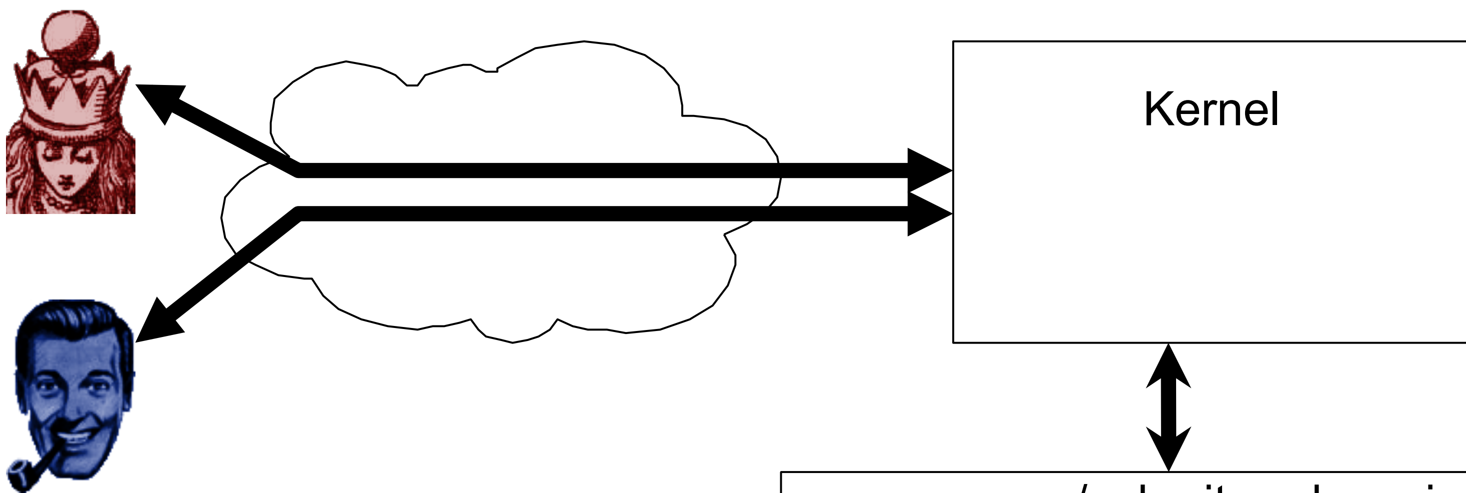
Asbestos Outline

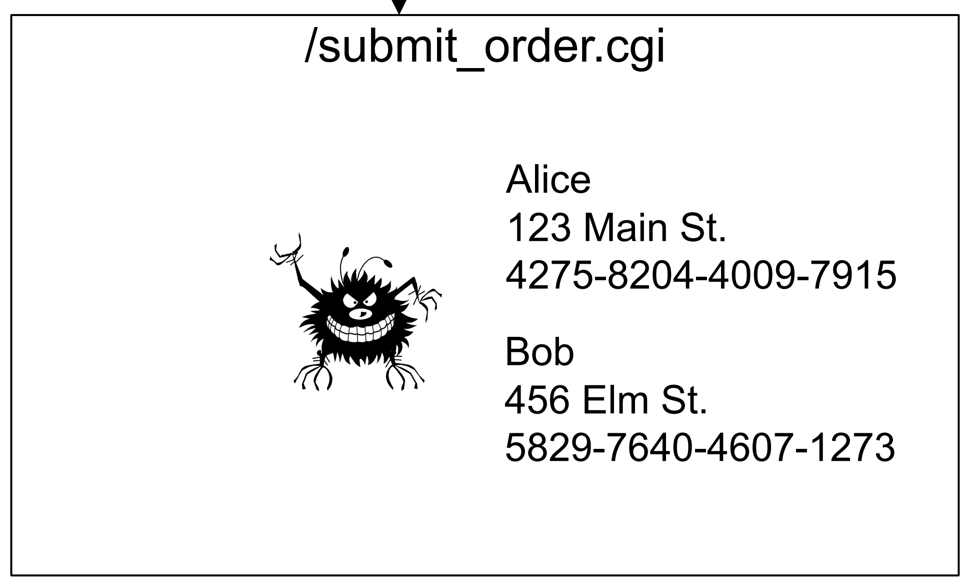
- Why is it needed?
- Other models
  - Virtual machines
- Asbestos OS
  - Labels
  - Event processes
- Asbestos OKWS
- Performance

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

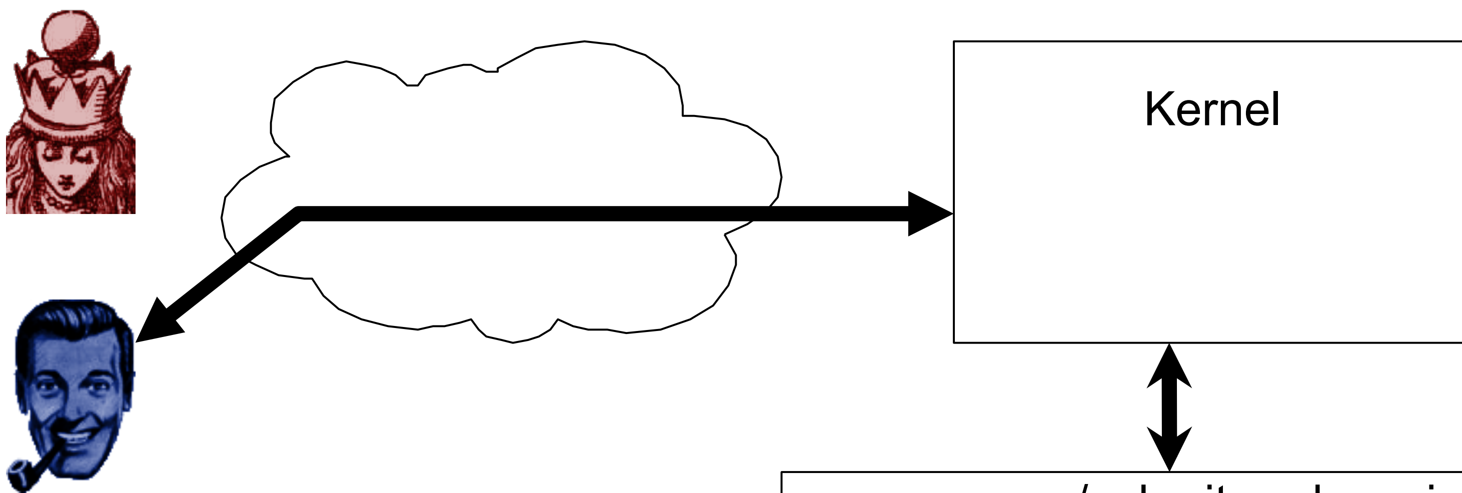
The Problem

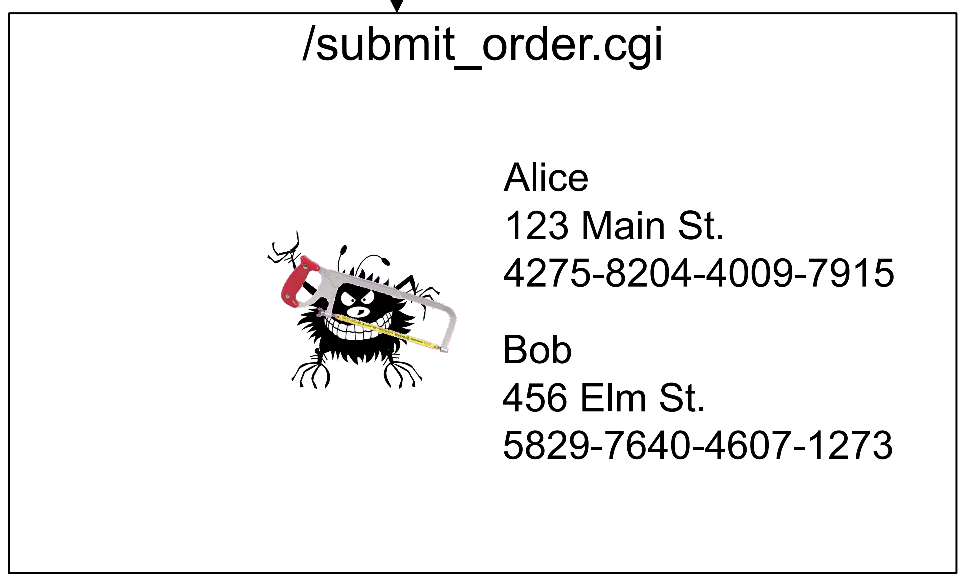




- If Bob compromises the system, he can access Alice's data

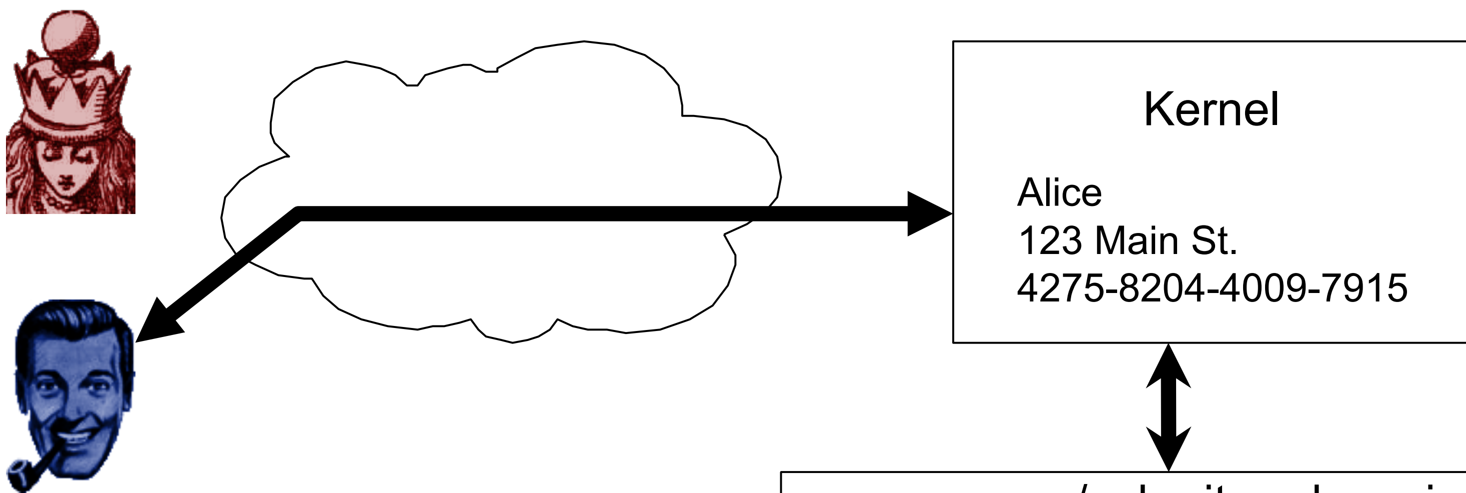
The Problem

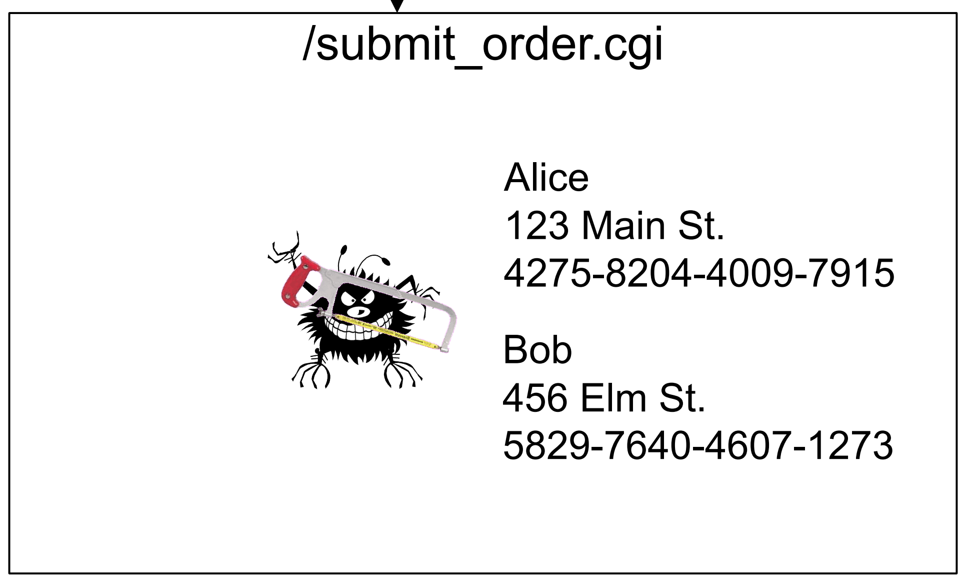




- If Bob compromises the system, he can access Alice's data

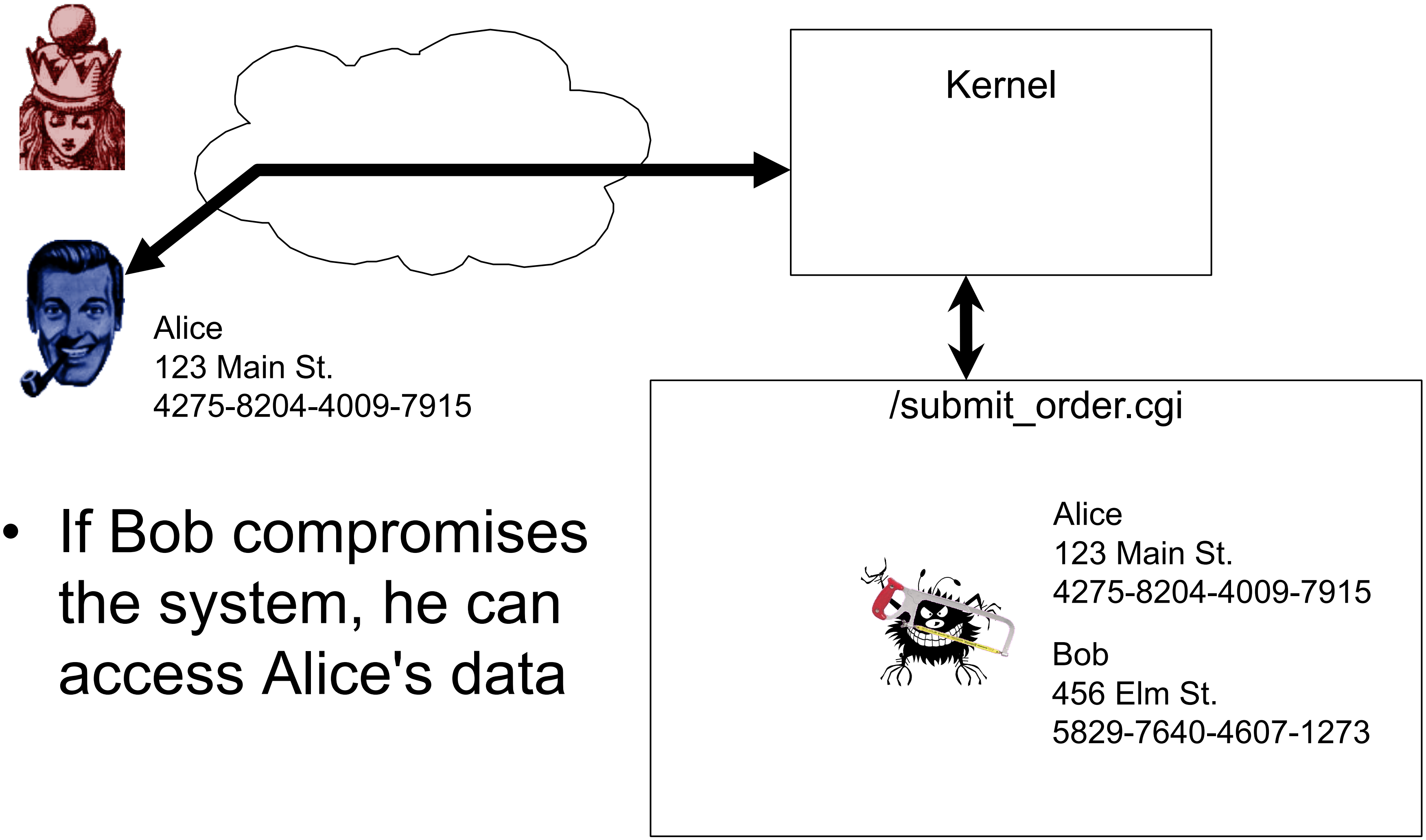
The Problem





- If Bob compromises the system, he can access Alice's data

The Problem

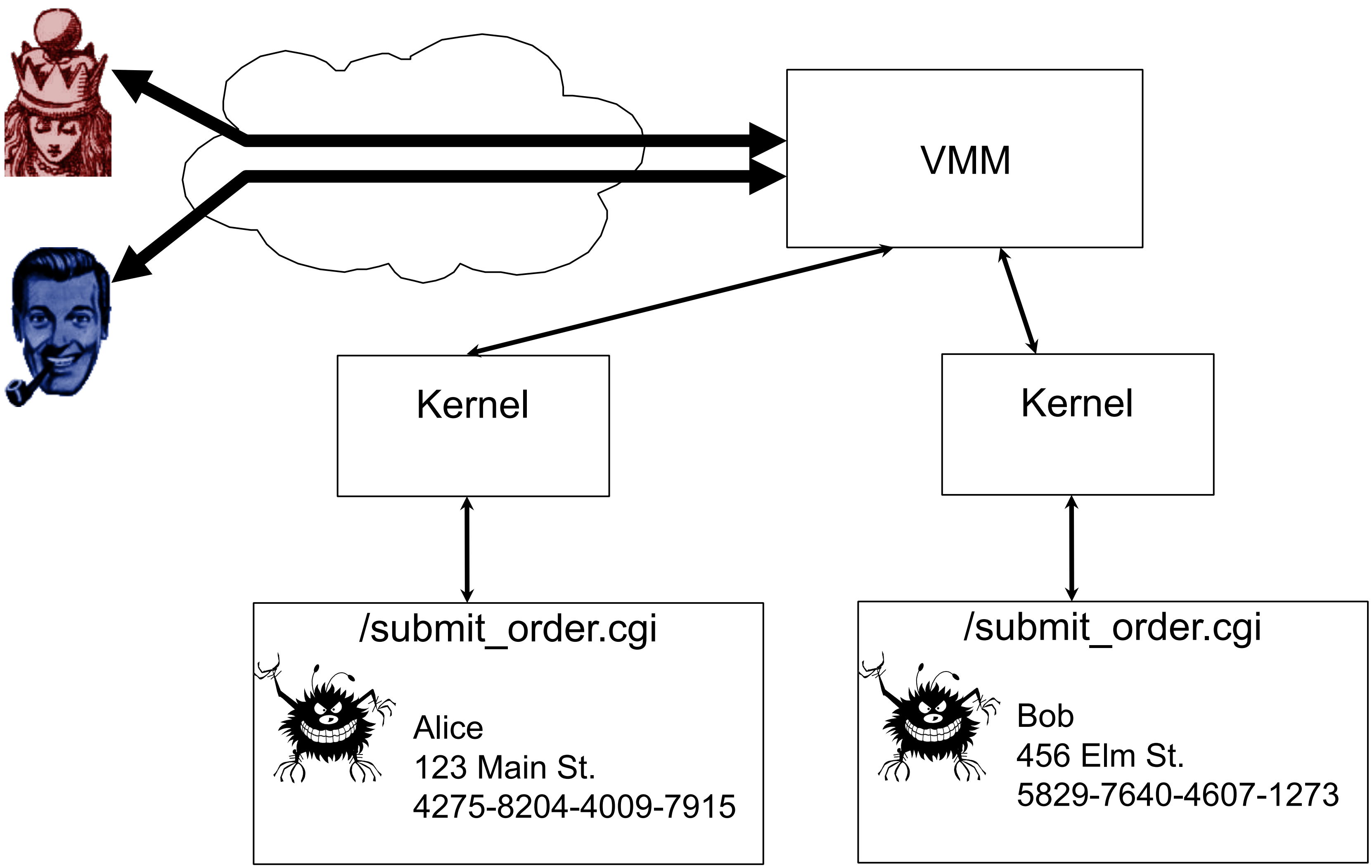


- If Bob compromises the system, he can access Alice's data

The Goal: User Isolation

- Bob should not be able to access Alice's data without Alice's permission
  - Alice and Bob’s data is **isolated**
- Complications
  - Even if there are bugs in the applications
  - Alice's data may travel through several processes
- To isolate, must prevent inappropriate data flow
- Application designer defines inappropriate

Virtual Machine Isolation



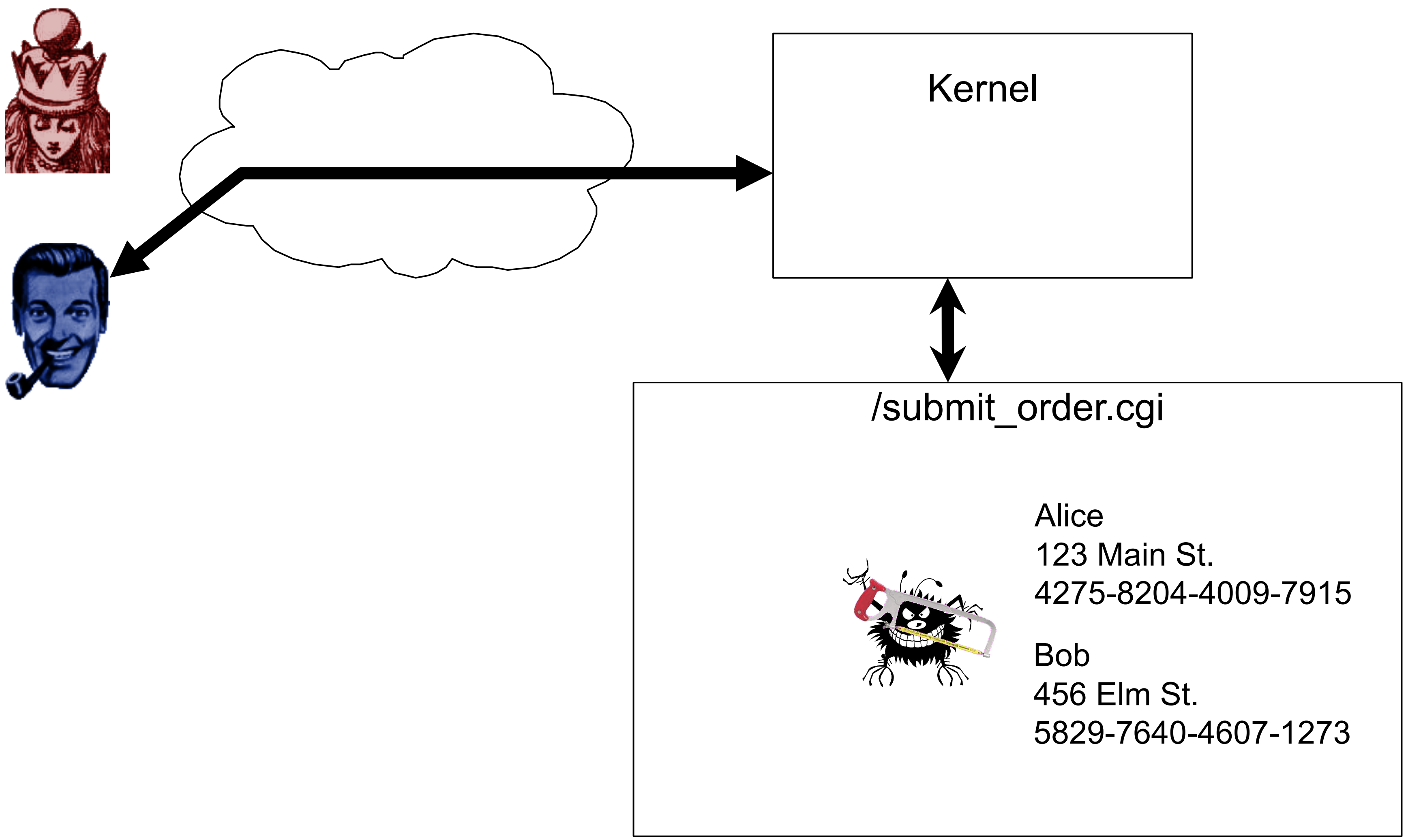
Virtual Machine Tradeoffs

+ Strict partitioning of off-the-shelf software

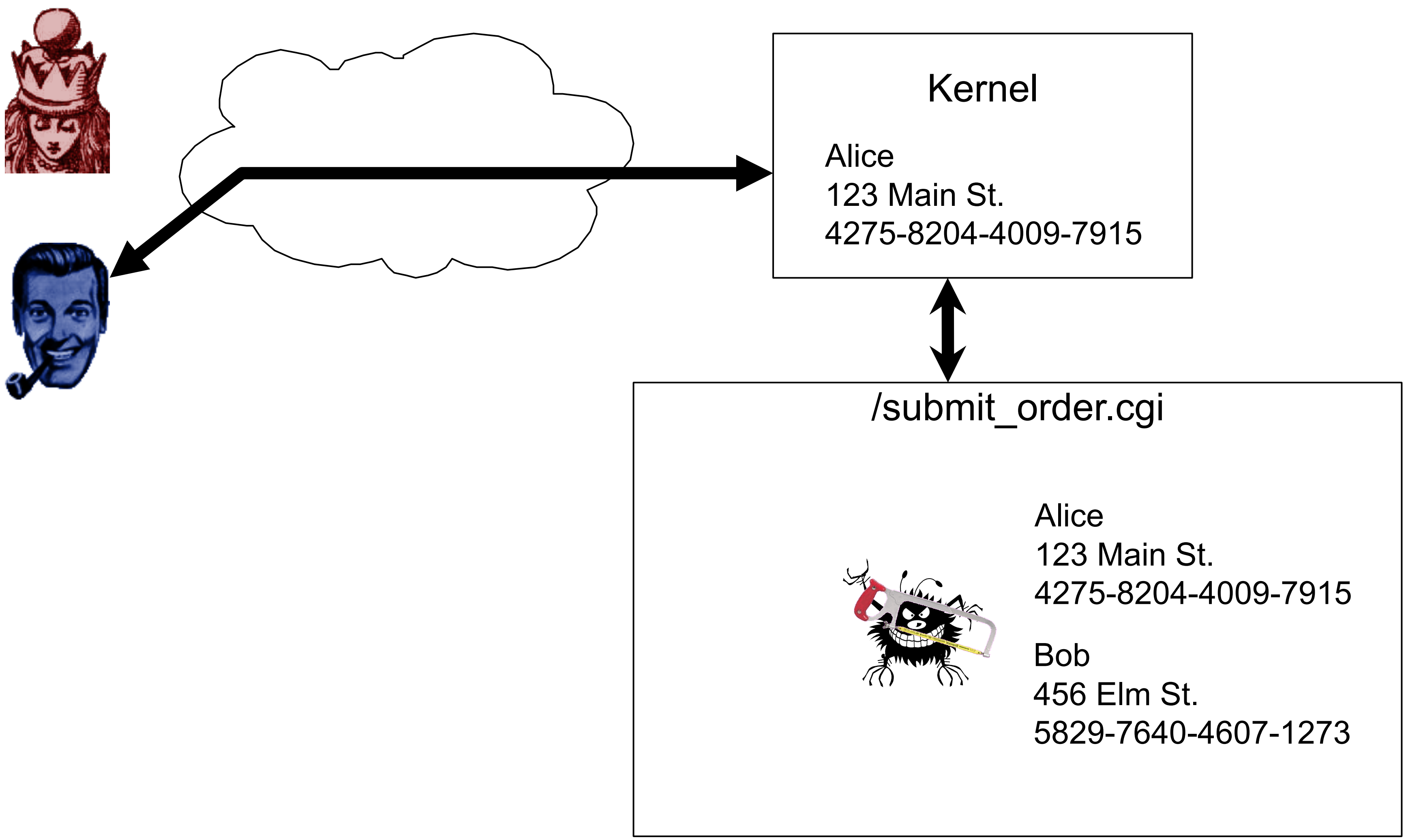
+ But…
  - Coarse-grained sharing
  - Resource challenges

- Isolation should be an OS feature

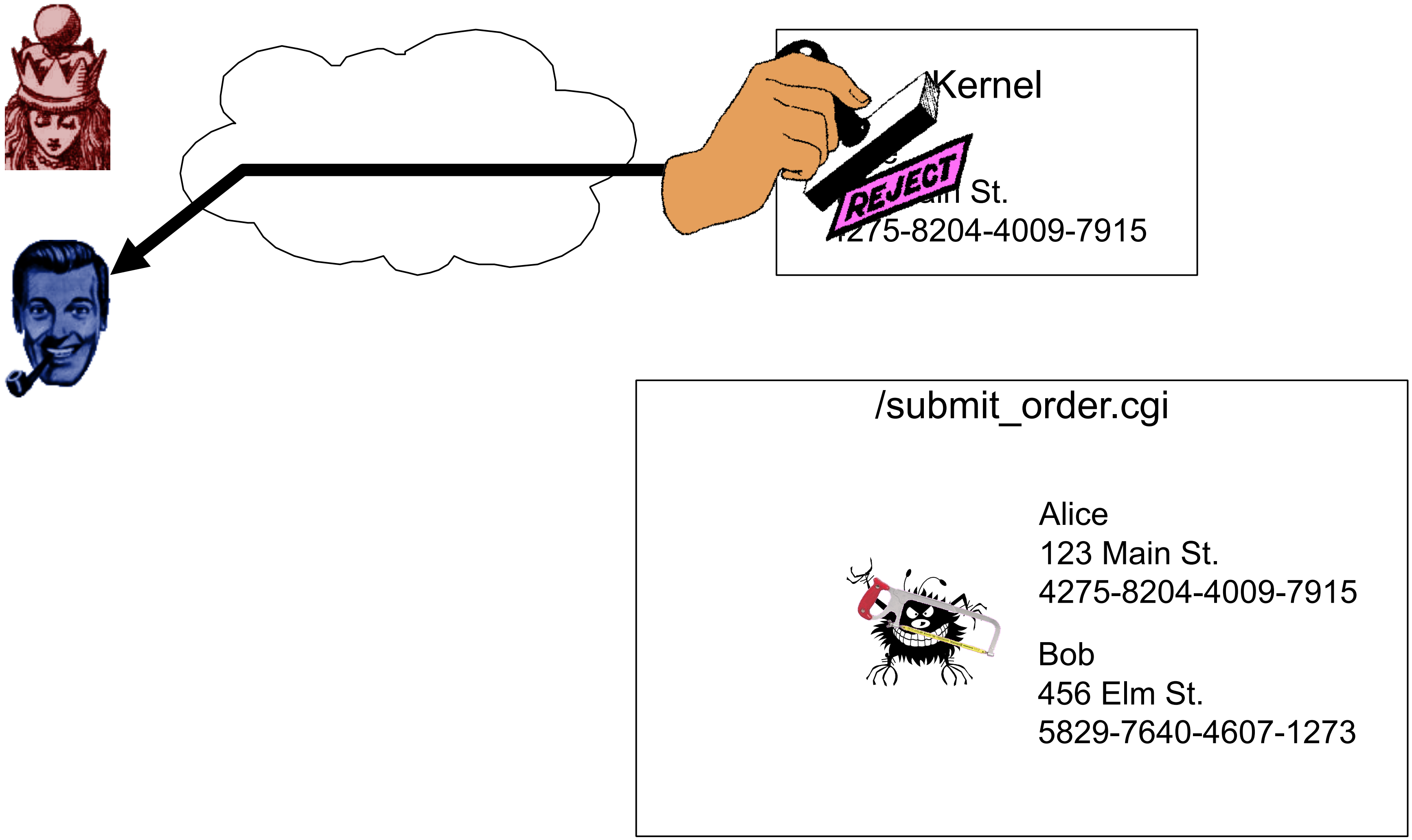
Desired Behavior



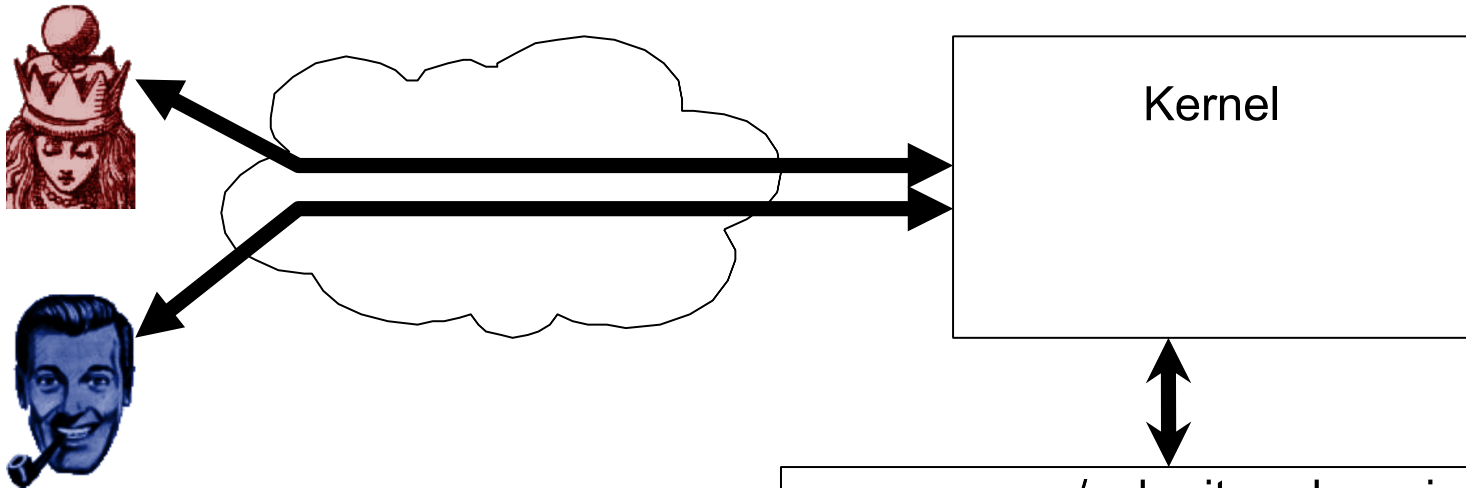
Desired Behavior

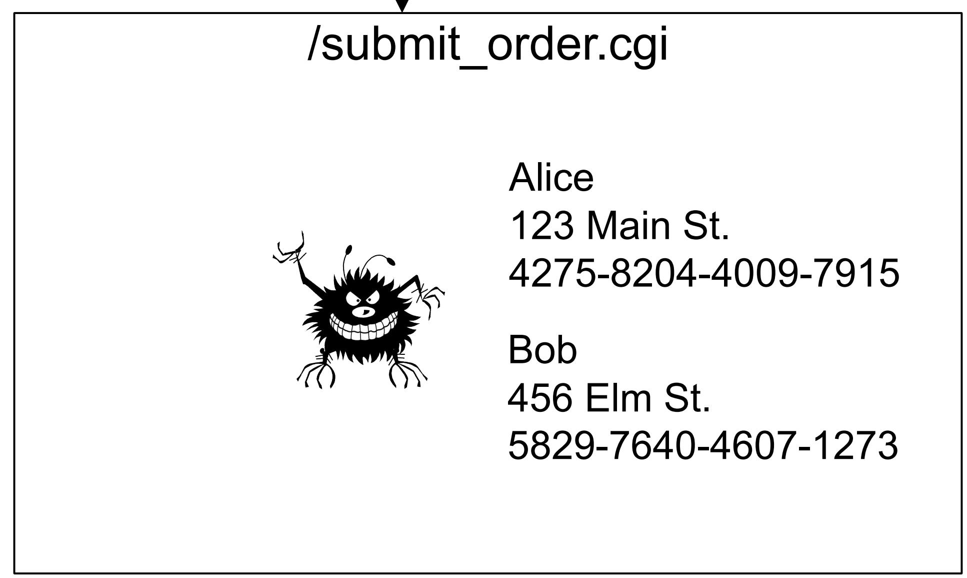


Desired Behavior



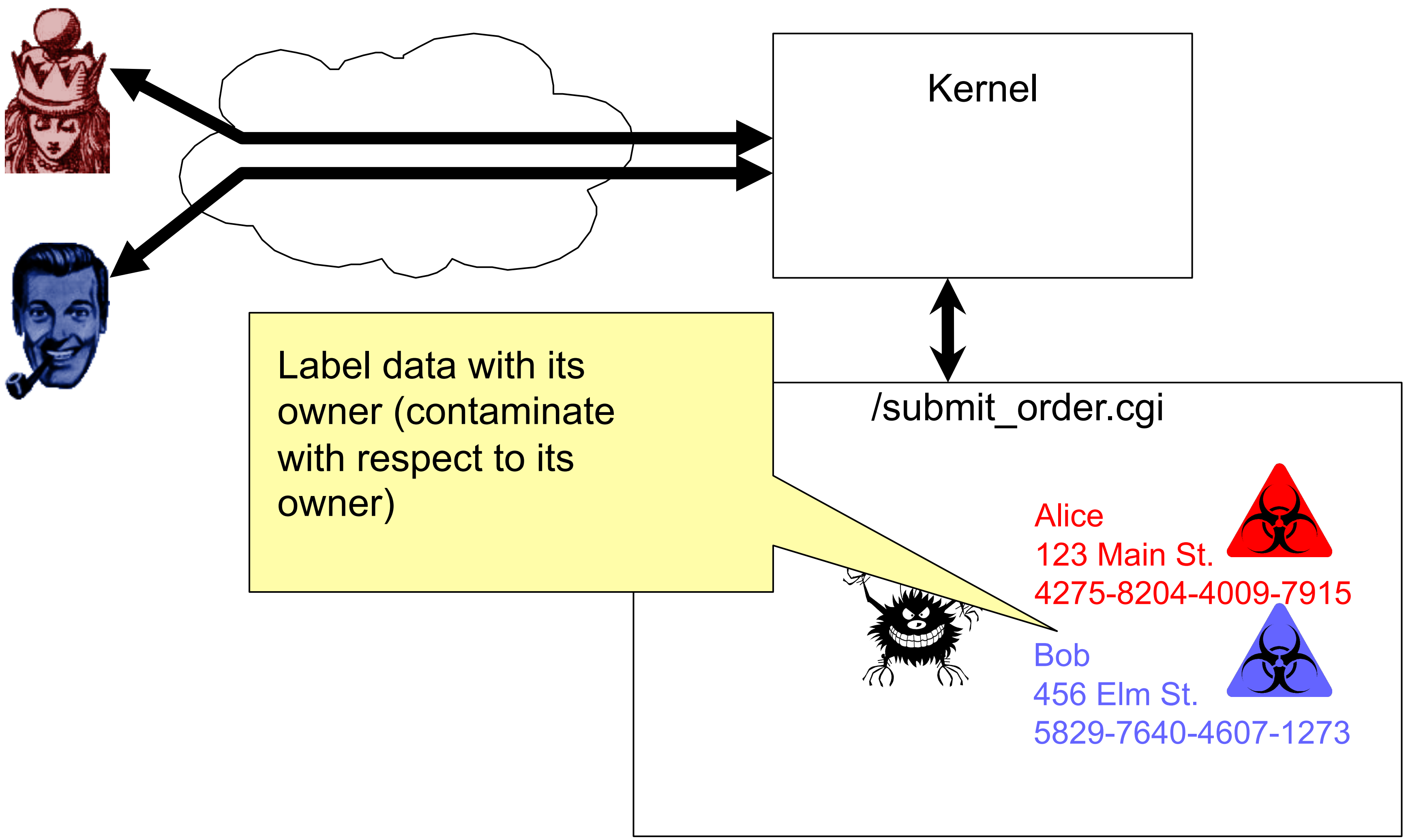
Information Flow Control



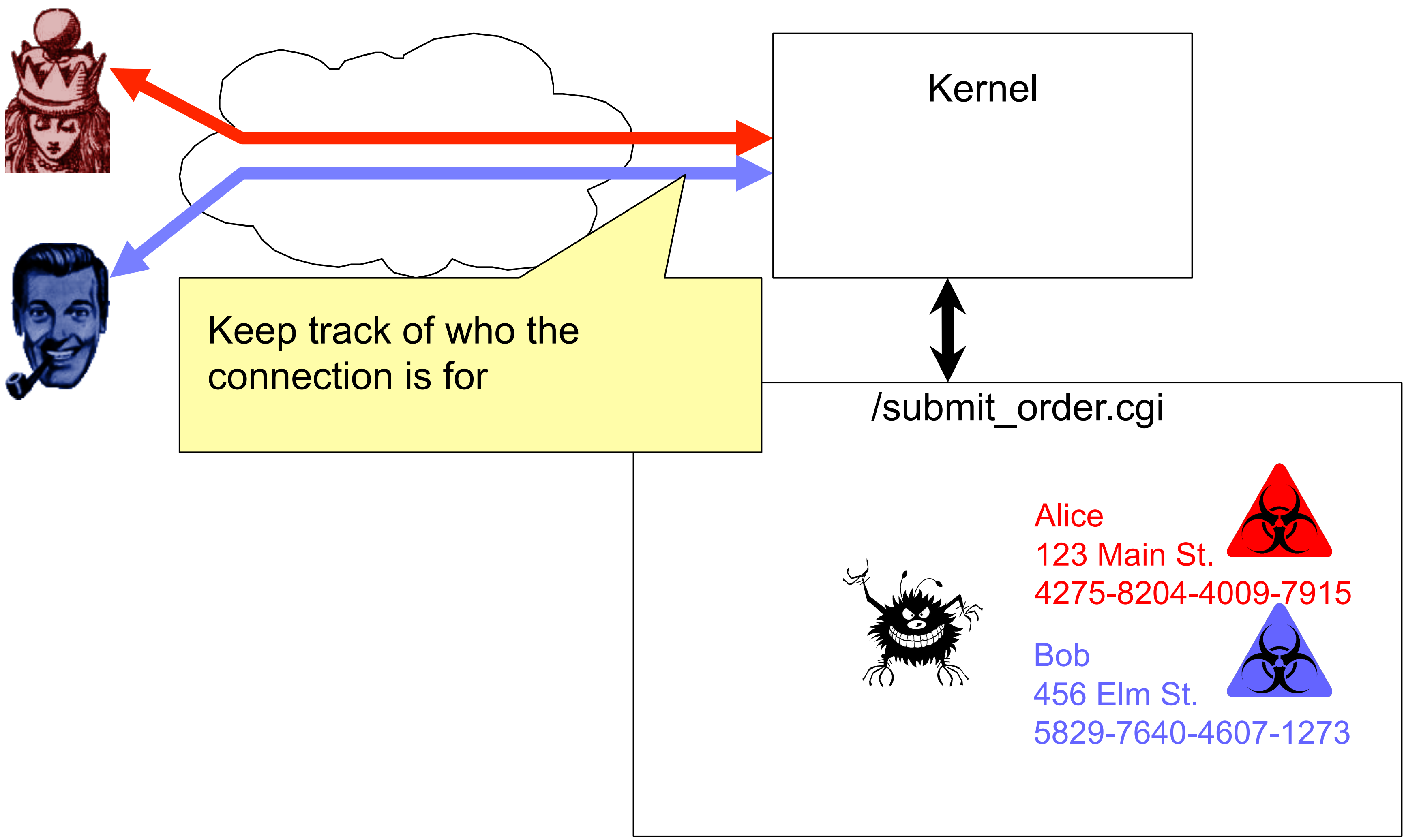


- Information flow control solves this kind of problem

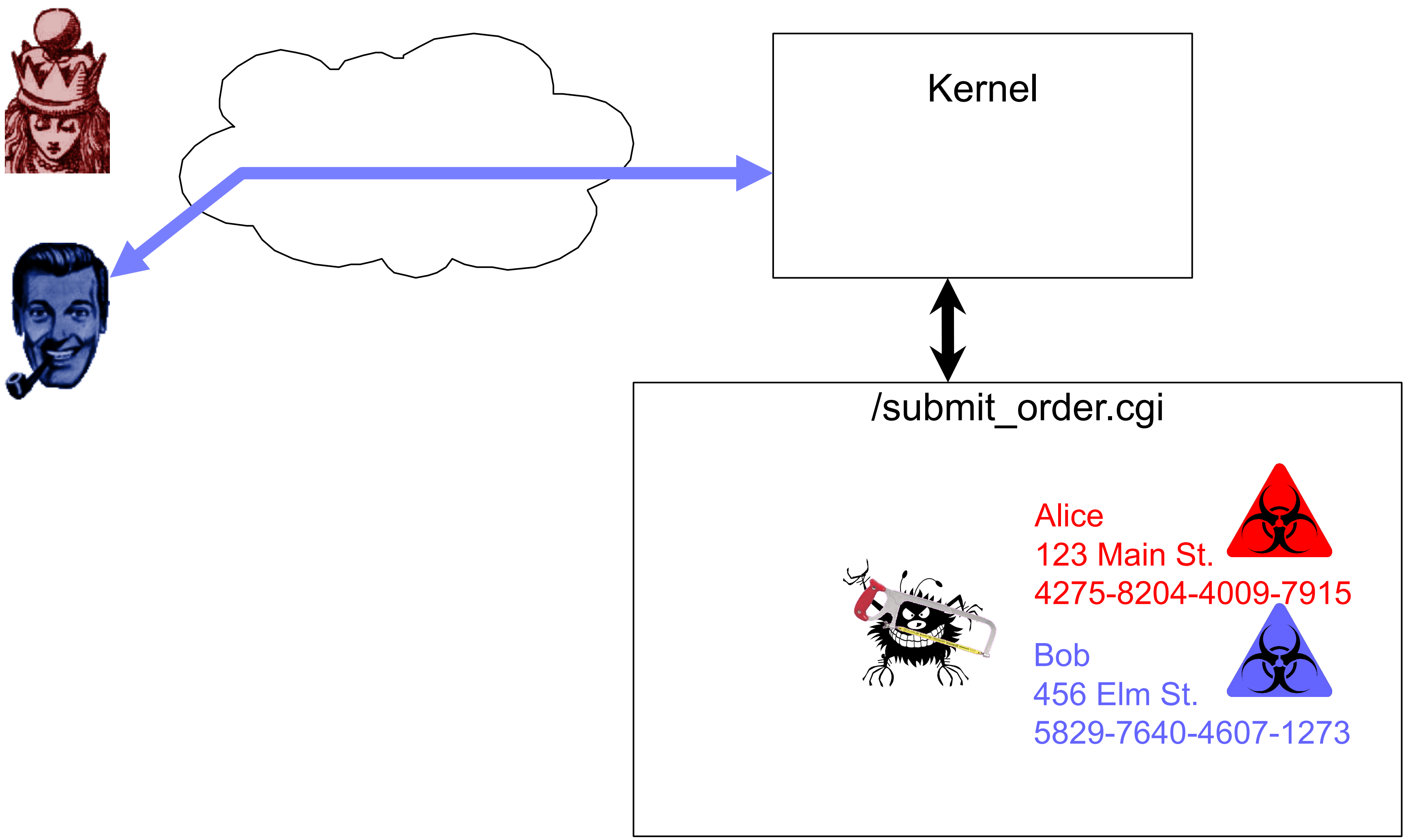
Information Flow Control



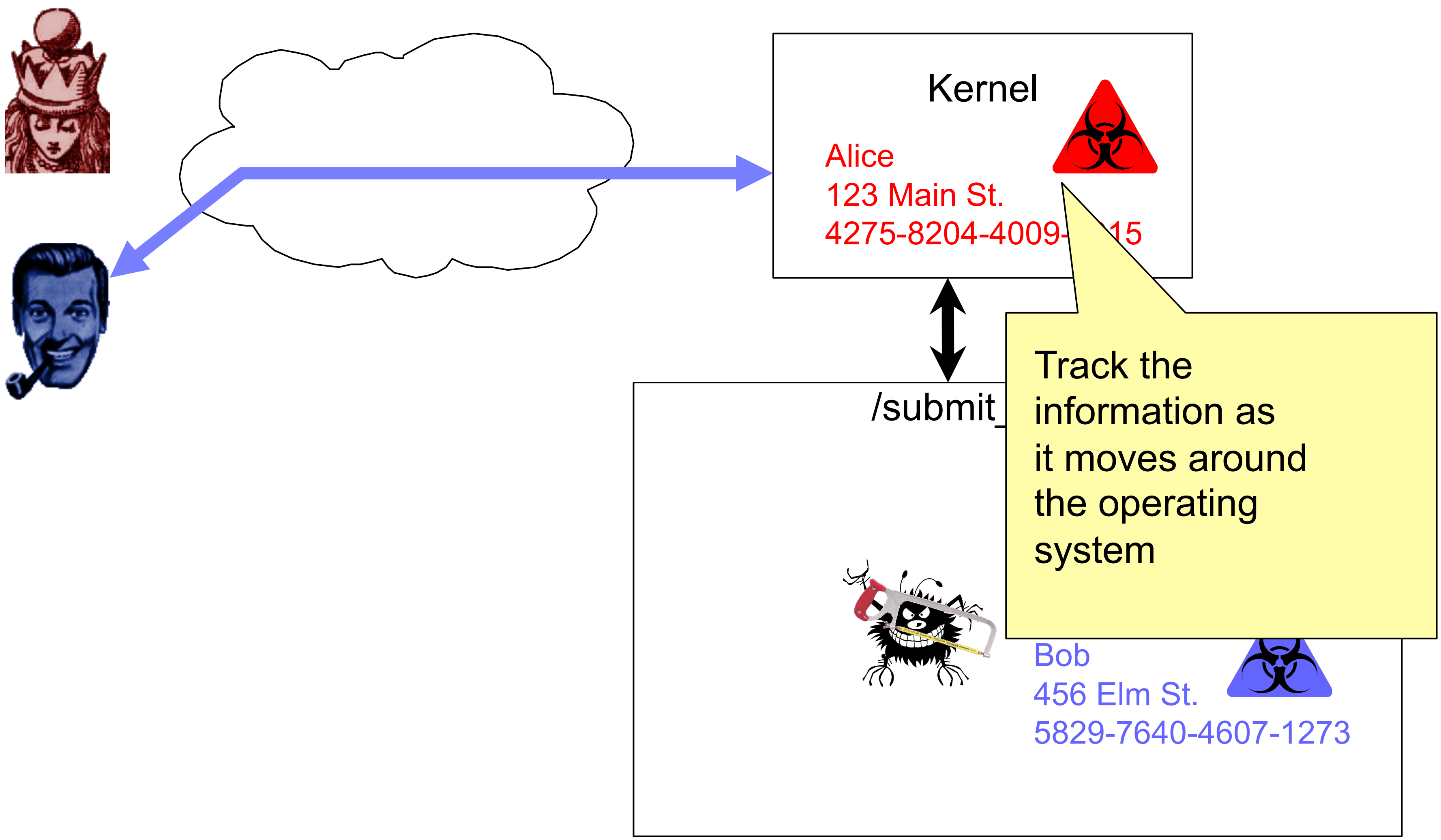
Information Flow Control



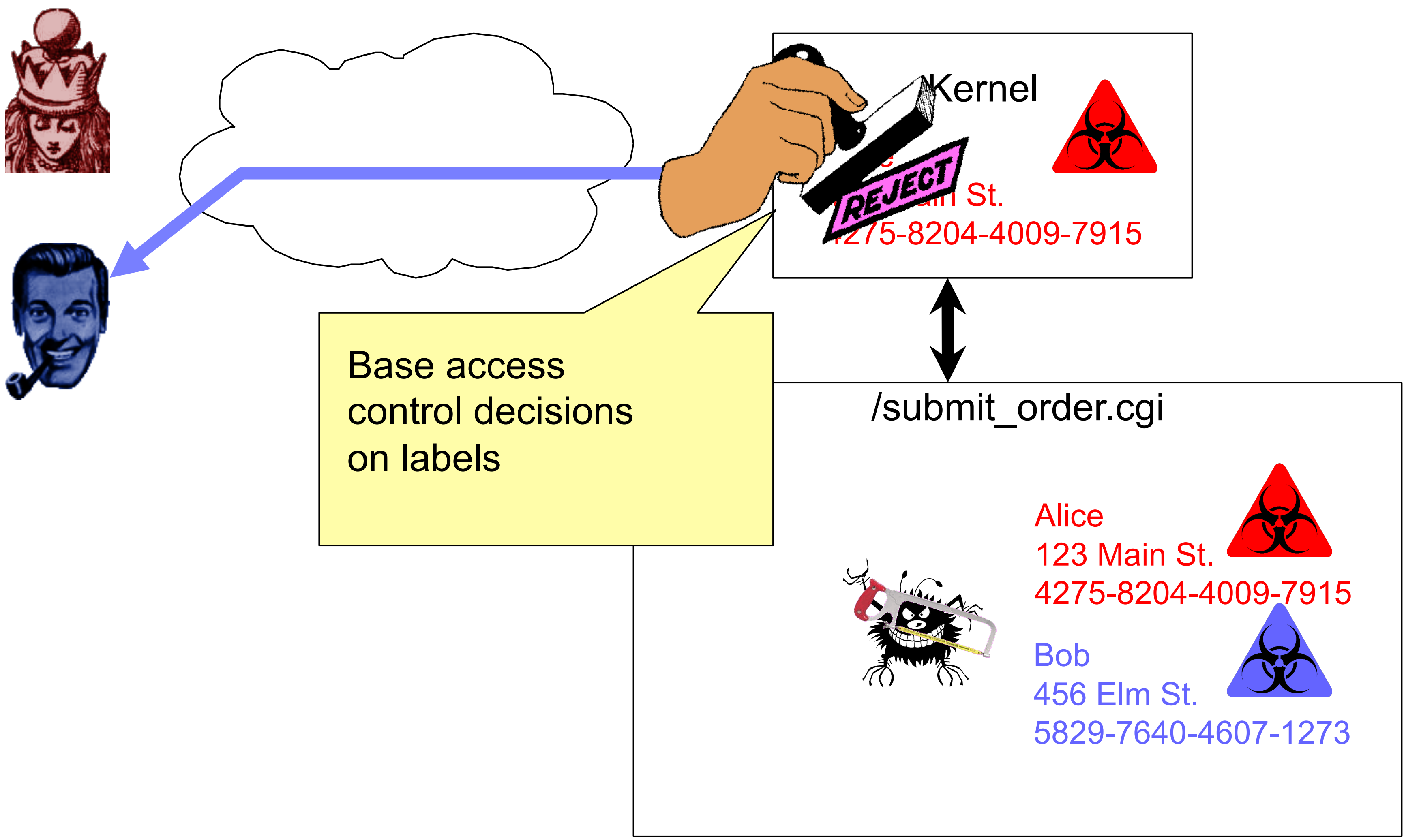
Information Flow Control



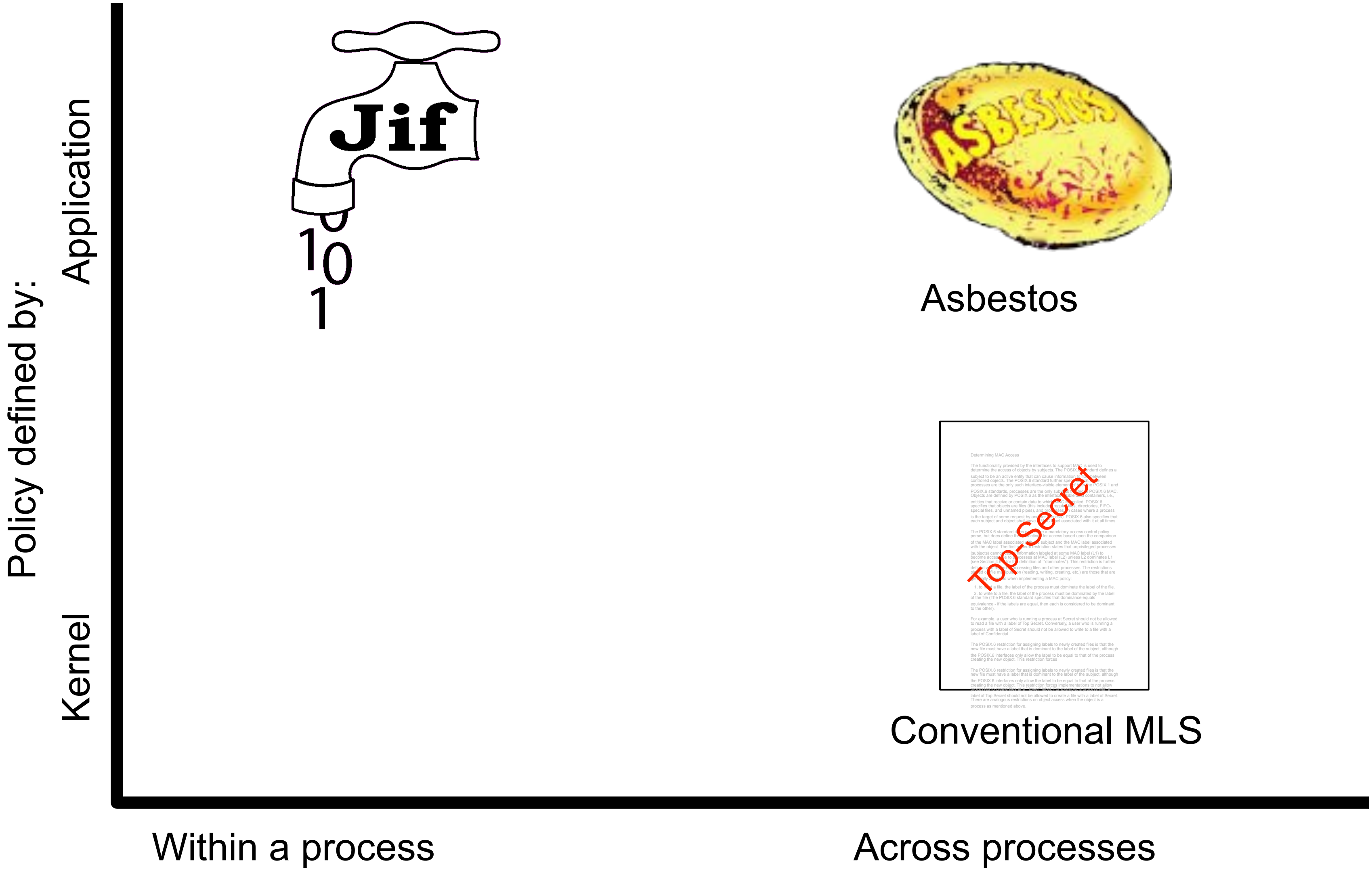
Information Flow Control



Information Flow Control



Approaches:
Information Flow Control Systems



Aproaches:
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

Asbestos Goals

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

Asbestos Goals

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

Asbestos Goals

- Large-scale
  - Changing population of thousands
- Efficient
  - Cache user data, while keeping it isolated
- Unprivileged
  - 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

Asbestos Compartments

- Contamination / label type
  - Mike's data, Michele's data, Peter's business data
  - Example had two compartments: Alice & Bob
- Created by application
  - Creator process can delegate rights
  - Kernel enforces compartment policy

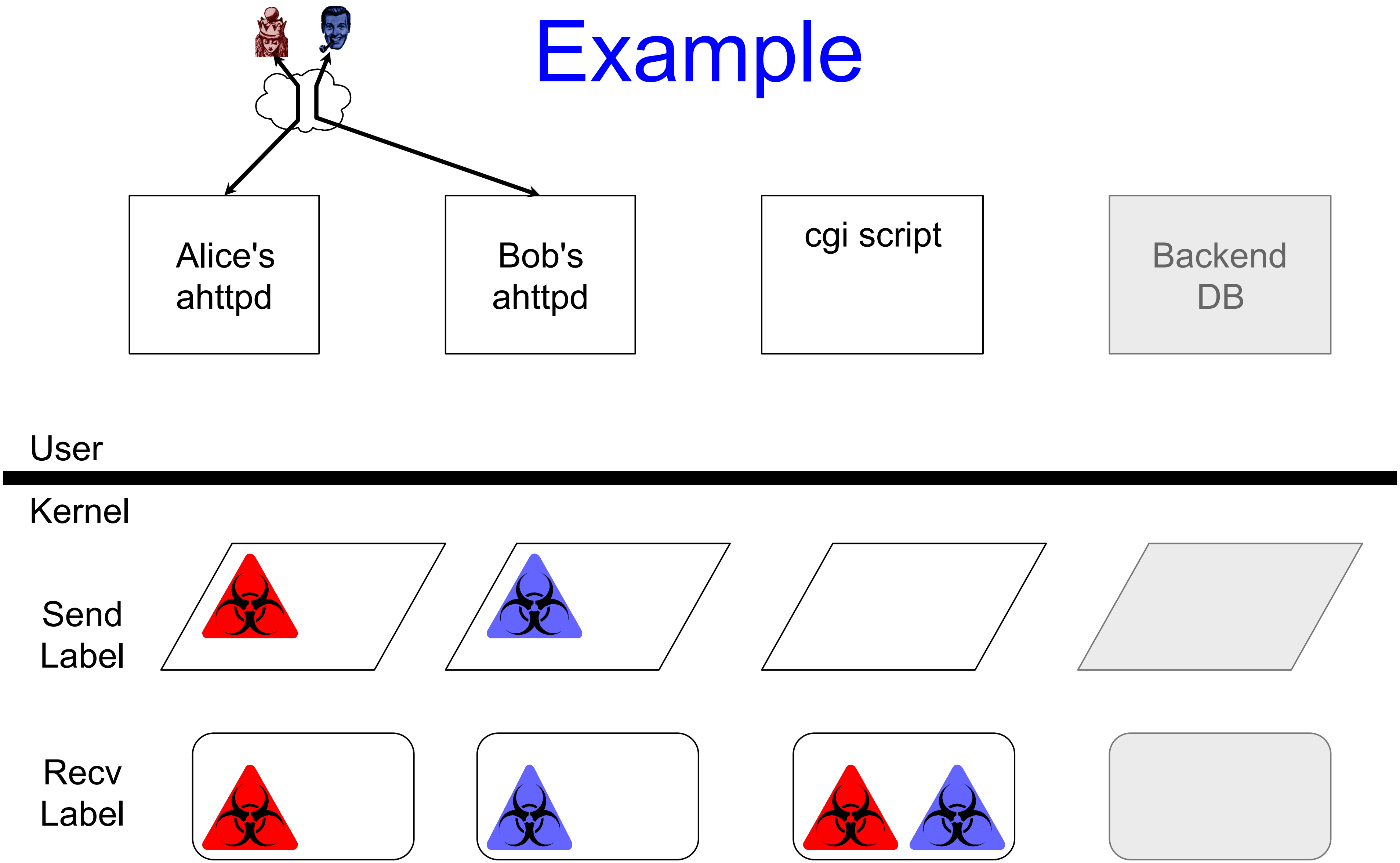
Asbestos Labels

- 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

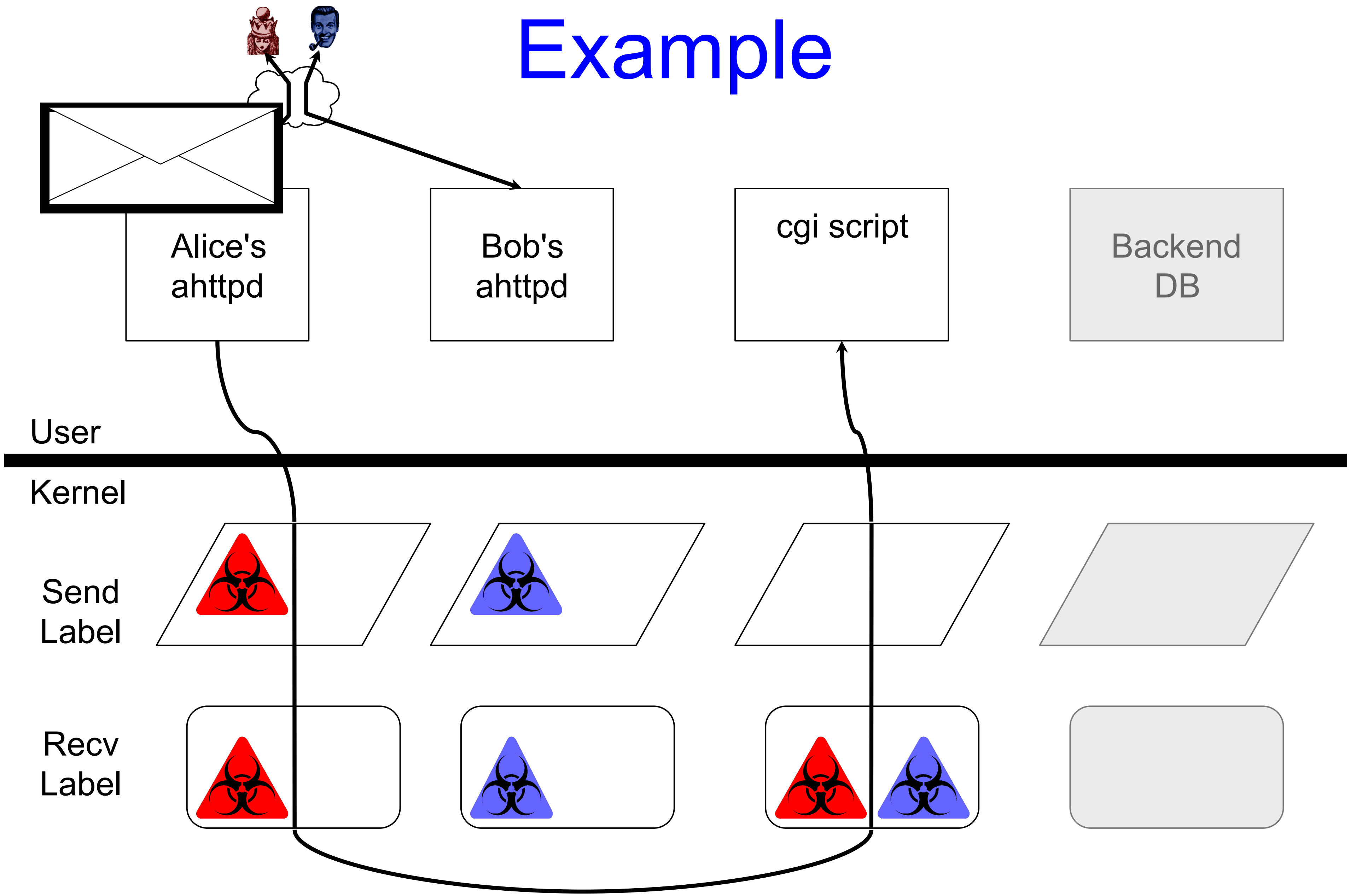
Asbestos Labels

- Application can create compartments without privilege
  - Application created users are isolated with the same mechanism as login users
  - Applications can easily sub-divide privilege
- Applications can *delegate* rights for compartments
  - Decentralized declassification like Jif
- Applications can choose different policies
  - Mandatory Access Control
  - Discretionary Access Control
  - Capabilities
  - More...

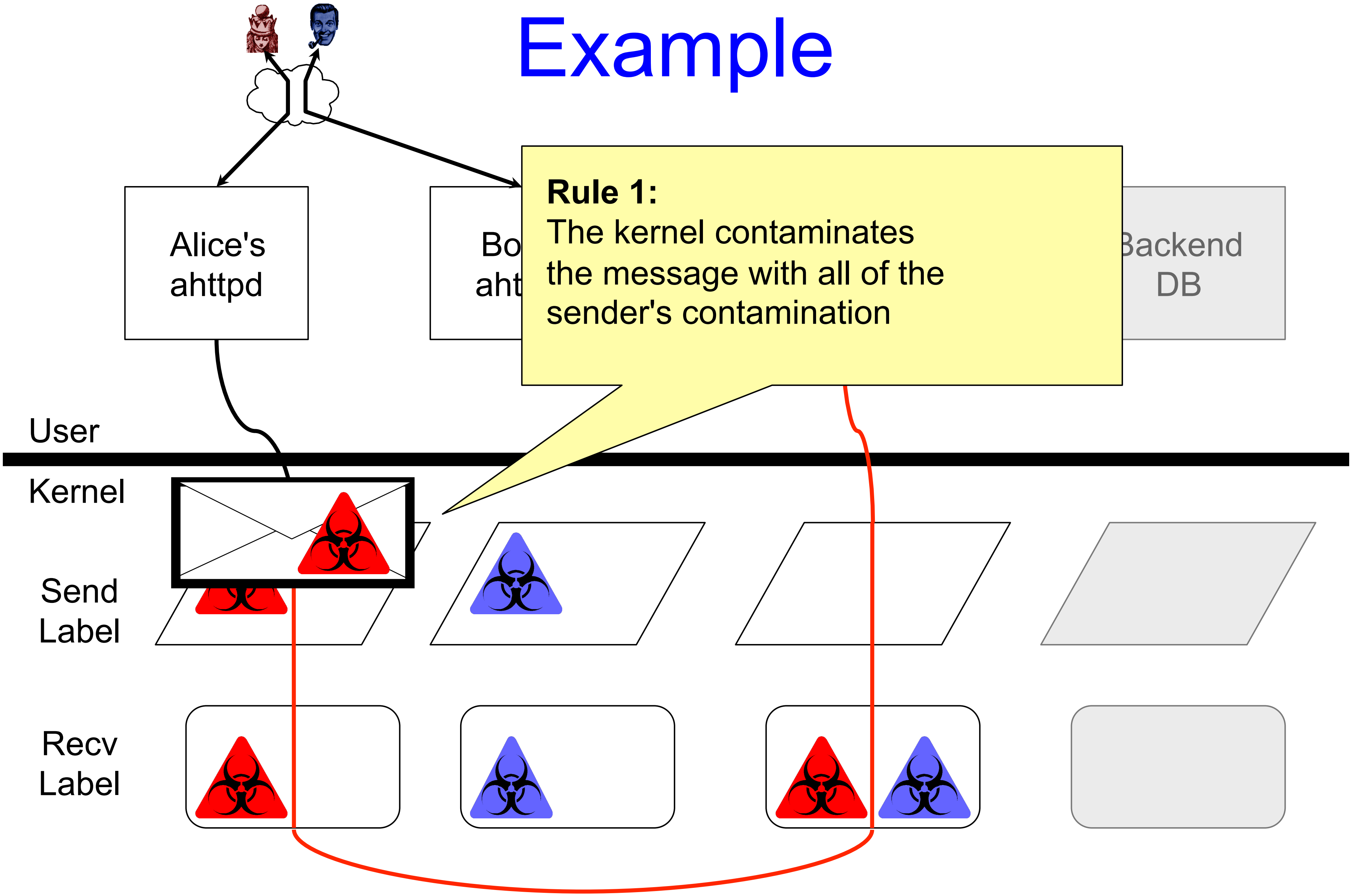
Basic Label Example



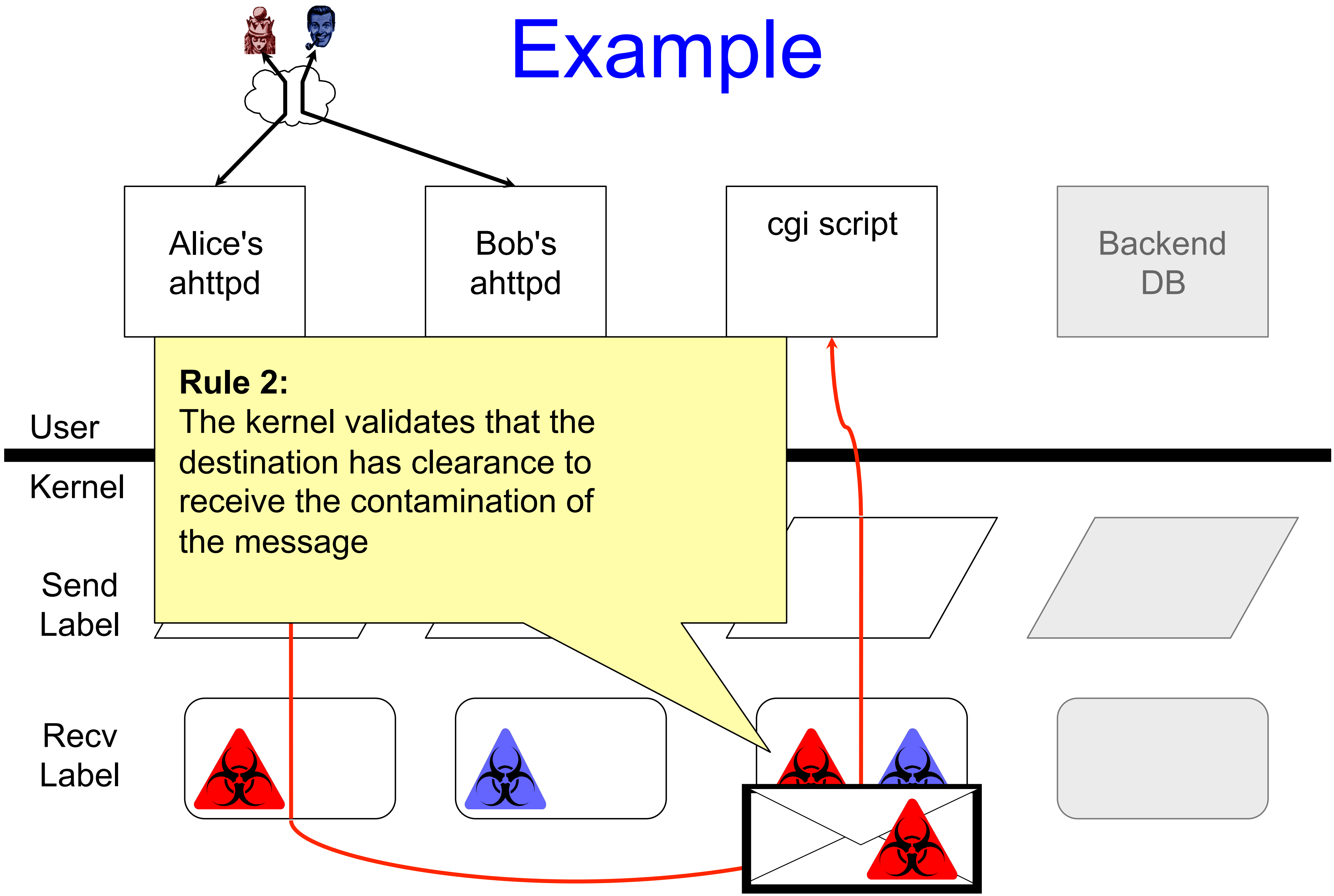
Basic Label Example



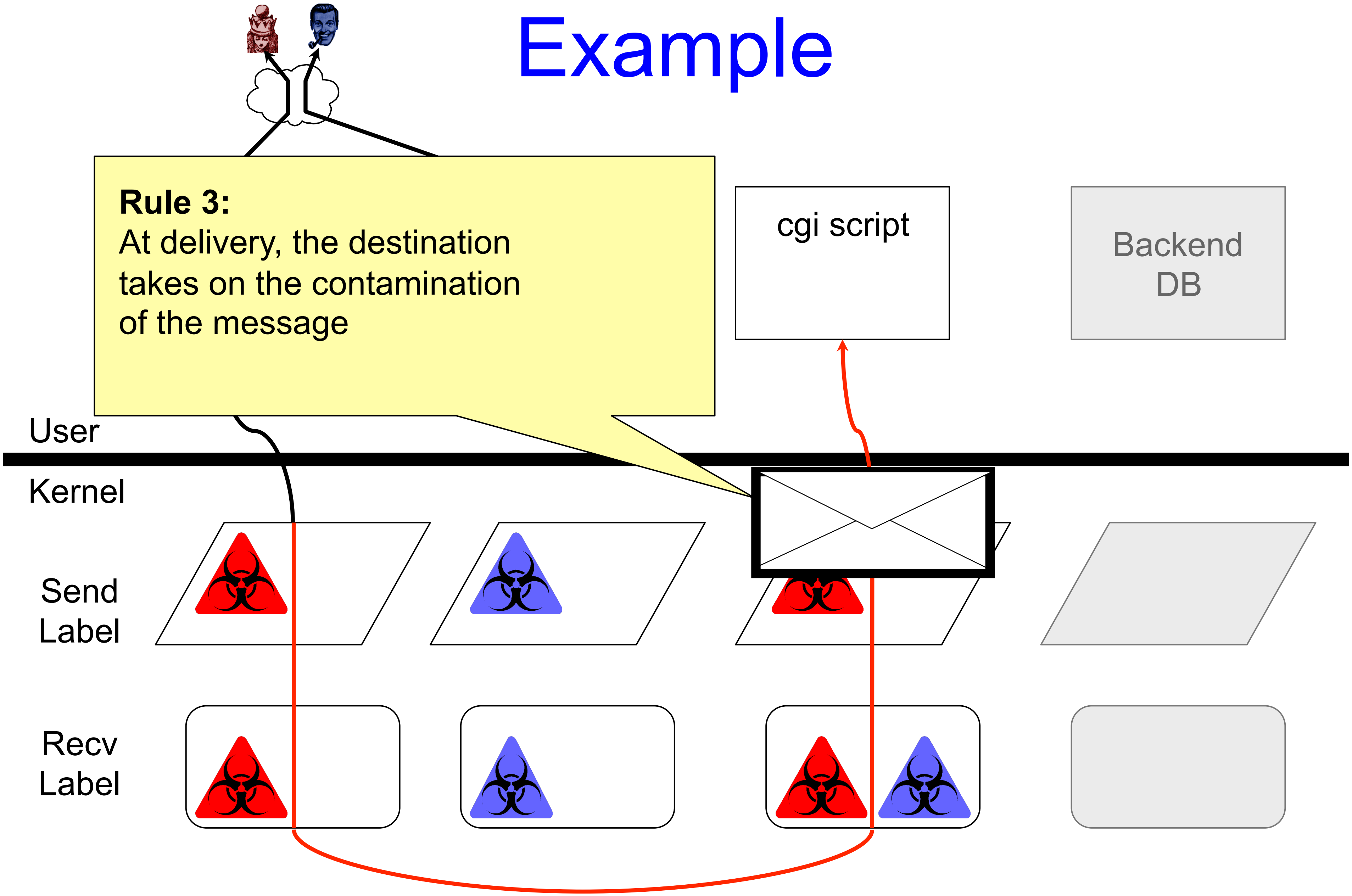
Basic Label Example



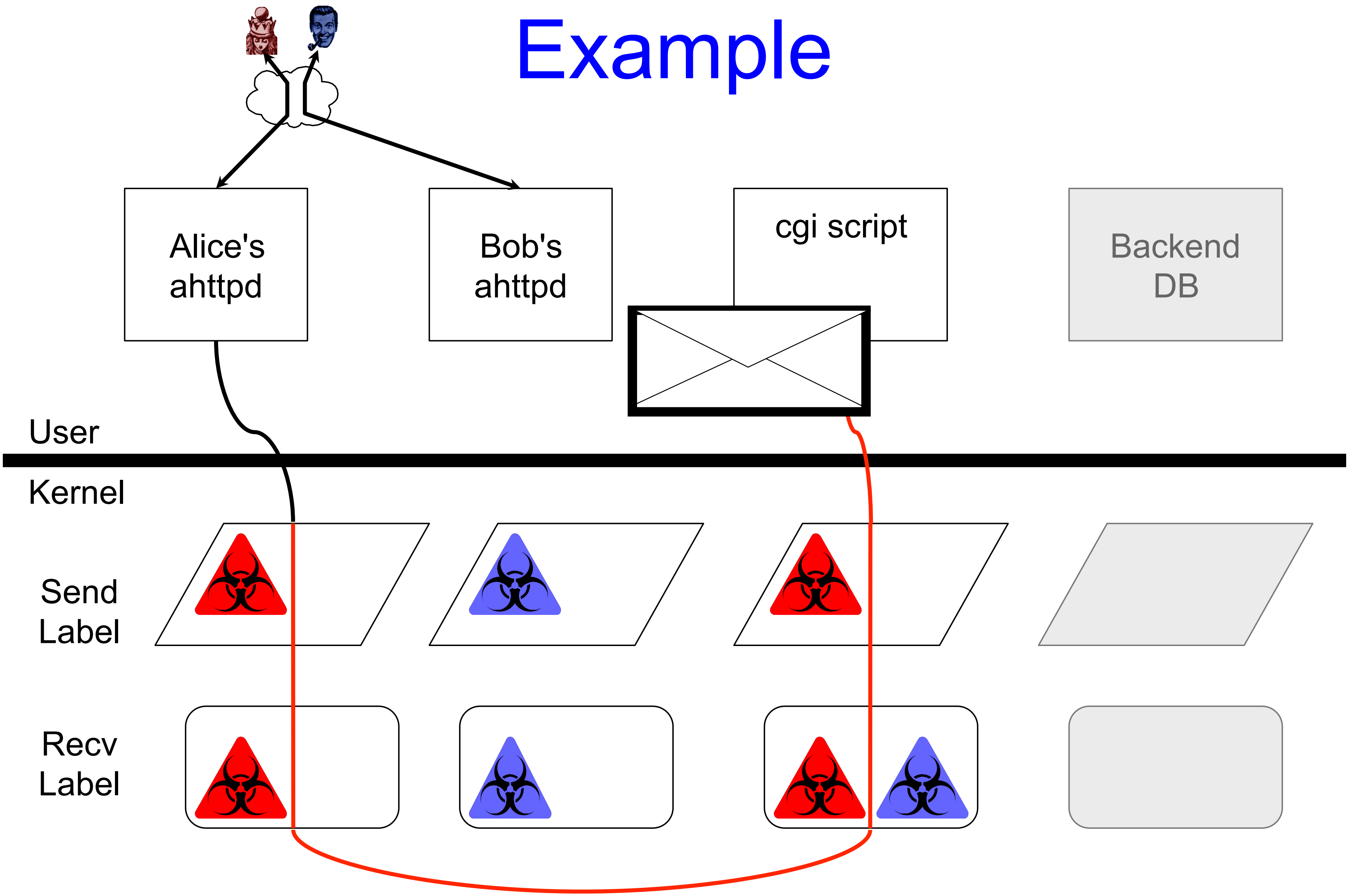
Basic Label Example



Basic Label Example

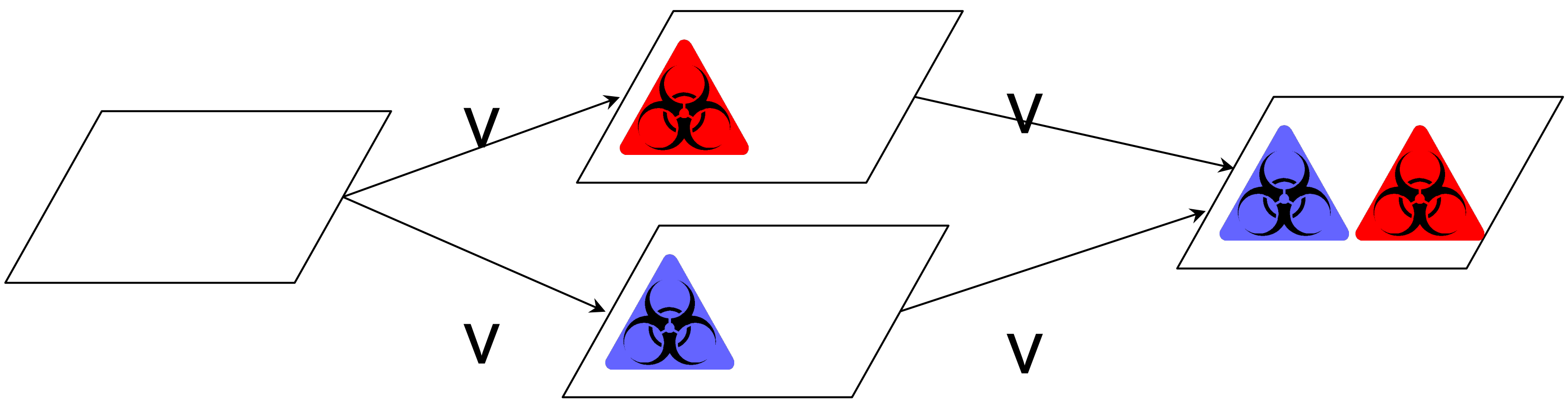


Basic Label Example

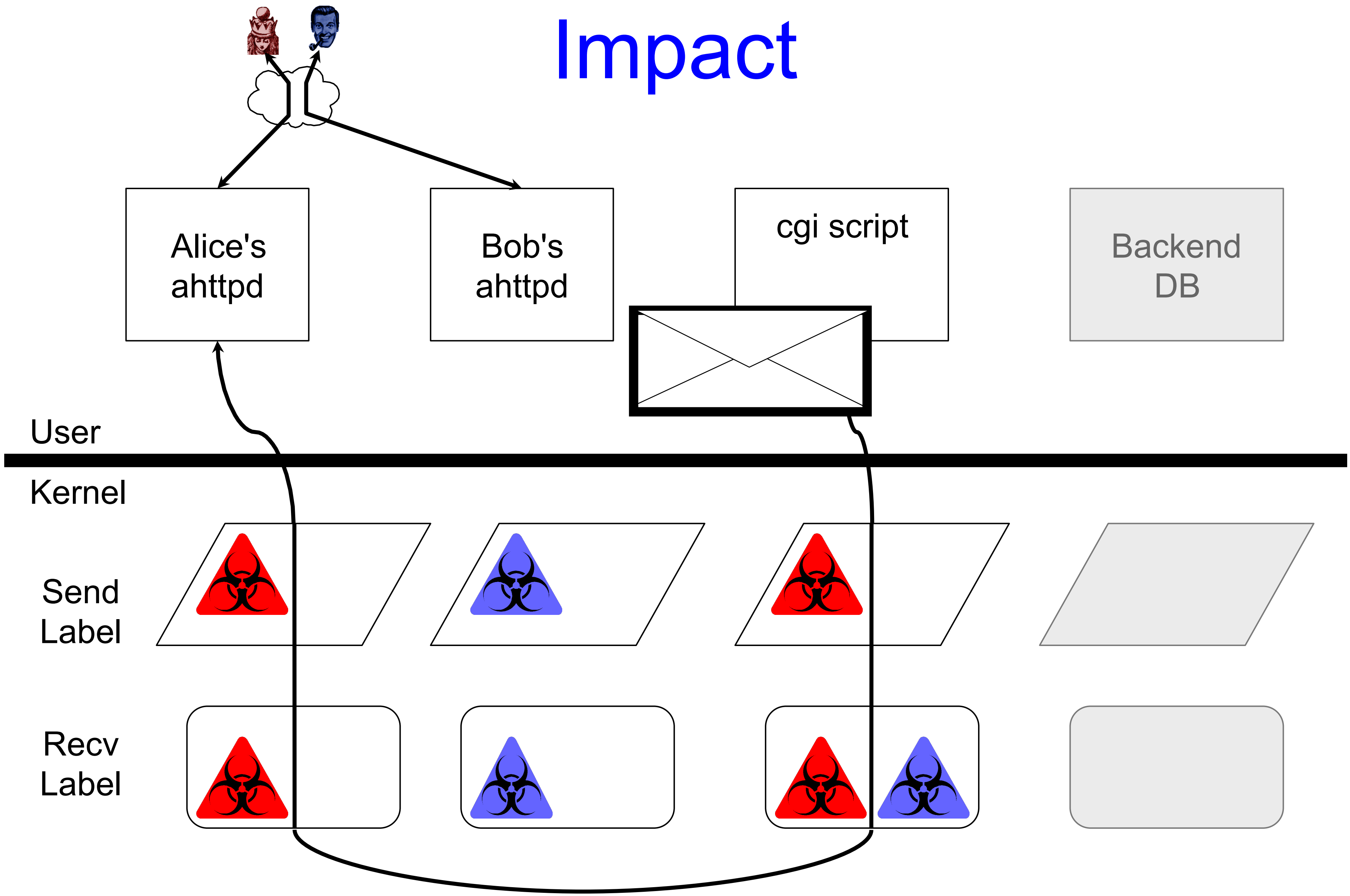


Implementing Clearance Checks

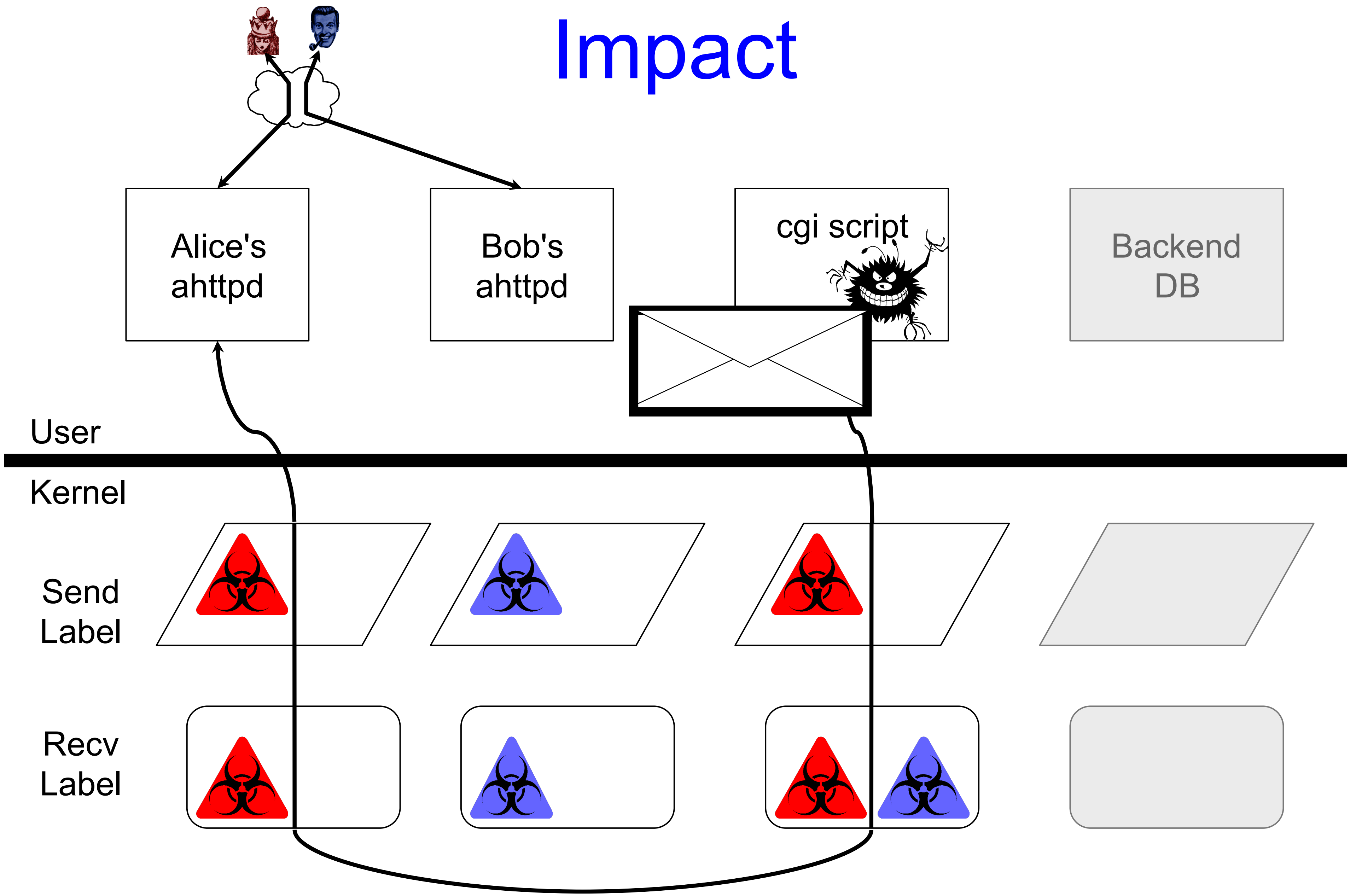
- How does the clearance check work?
- Labels form a lattice
- Partial ordering
  - Sender's send label must be less than or equal to the destination's receive label
- Send label updated with a least upper bound operator



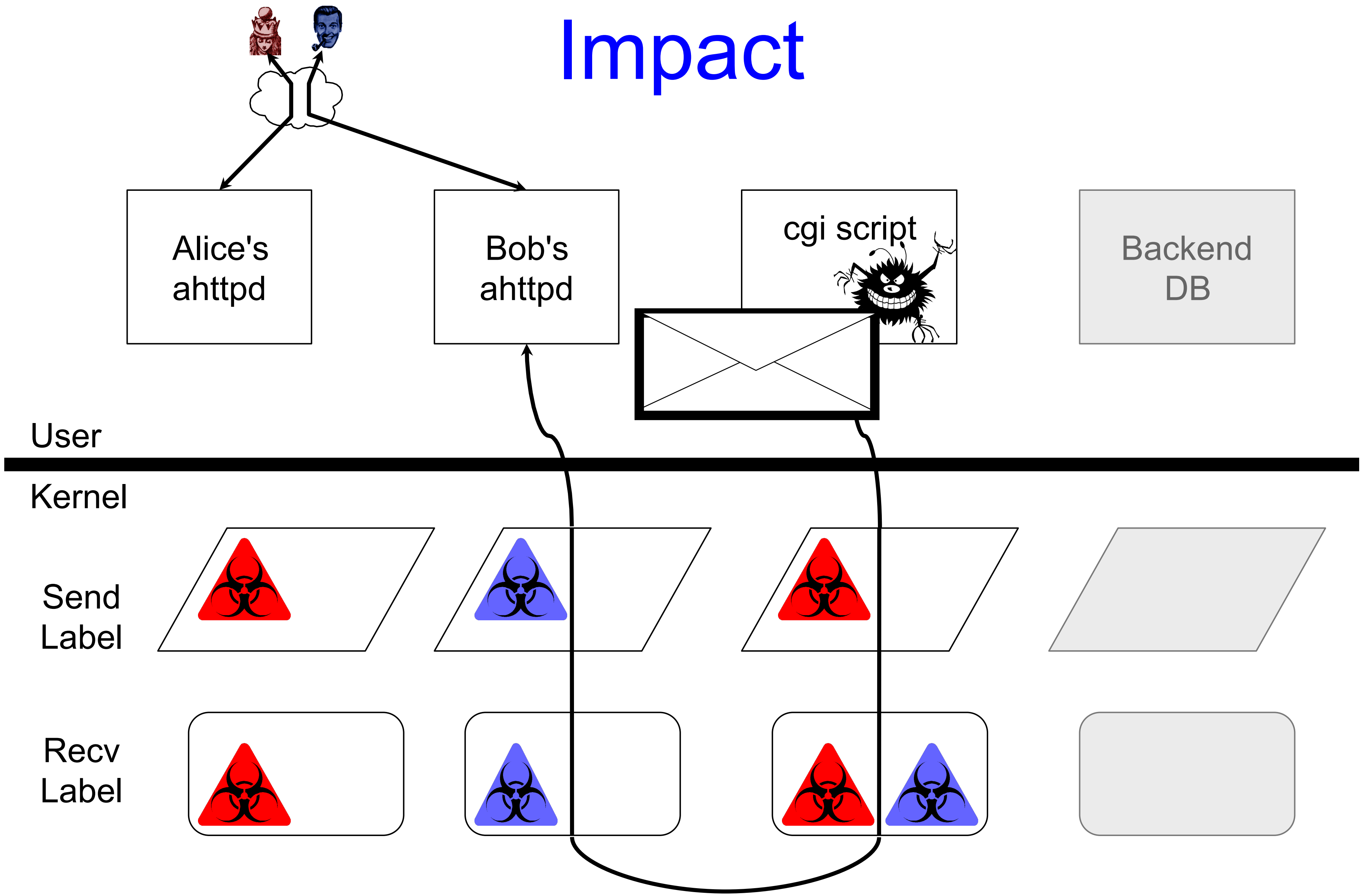
Limiting Bug Impact



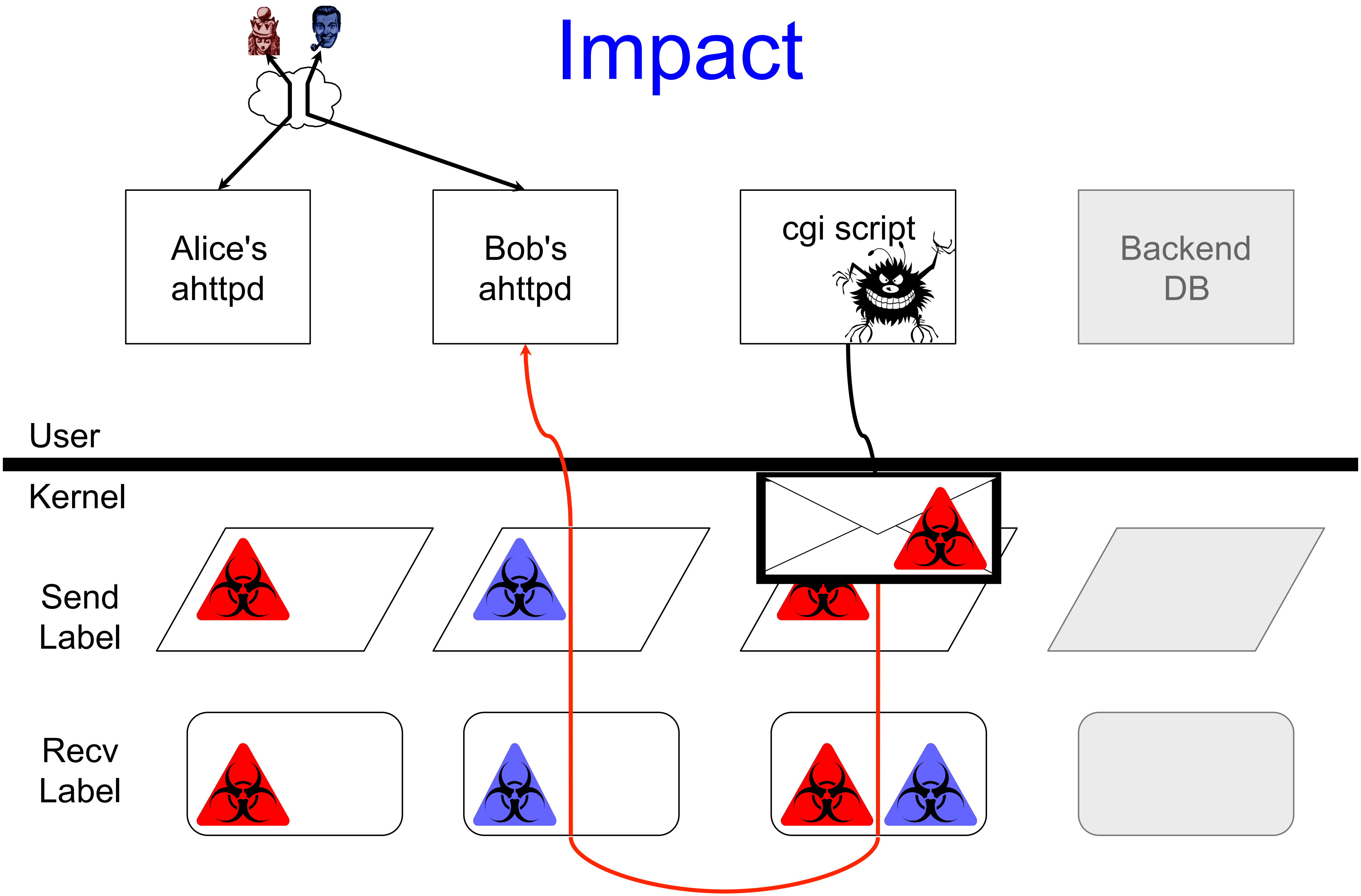
Limiting Bug Impact



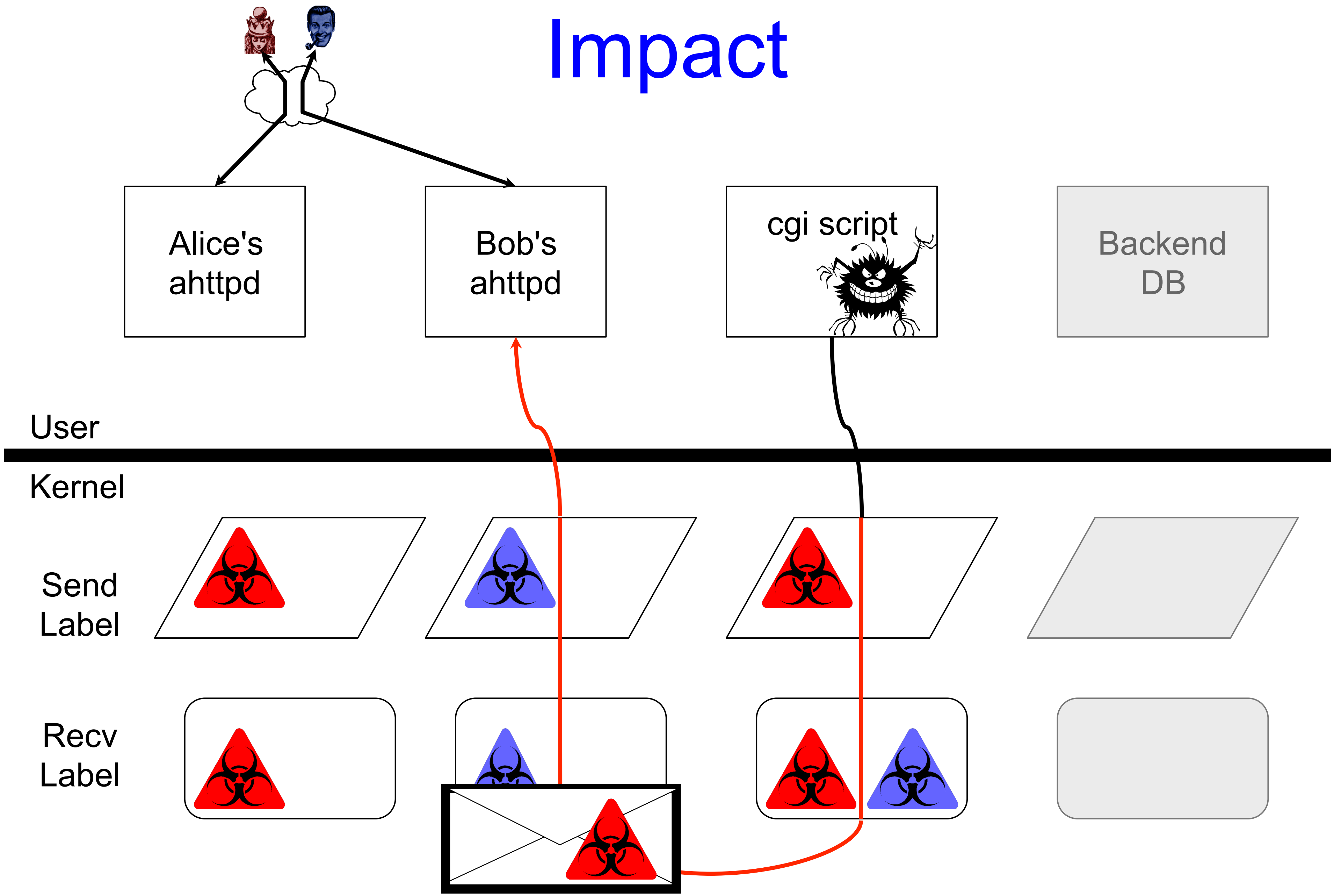
Limiting Bug Impact



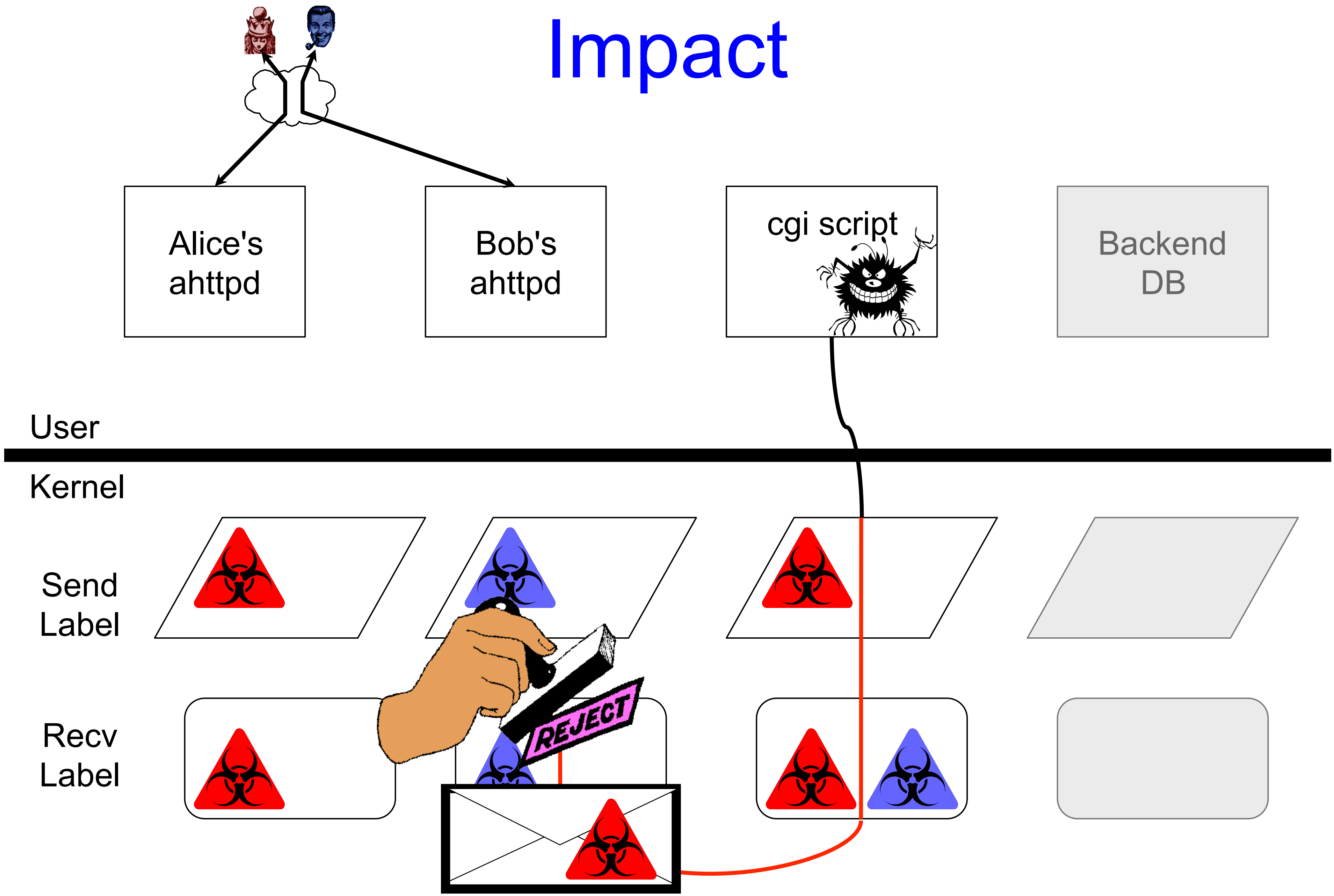
Limiting Bug Impact



Limiting Bug Impact



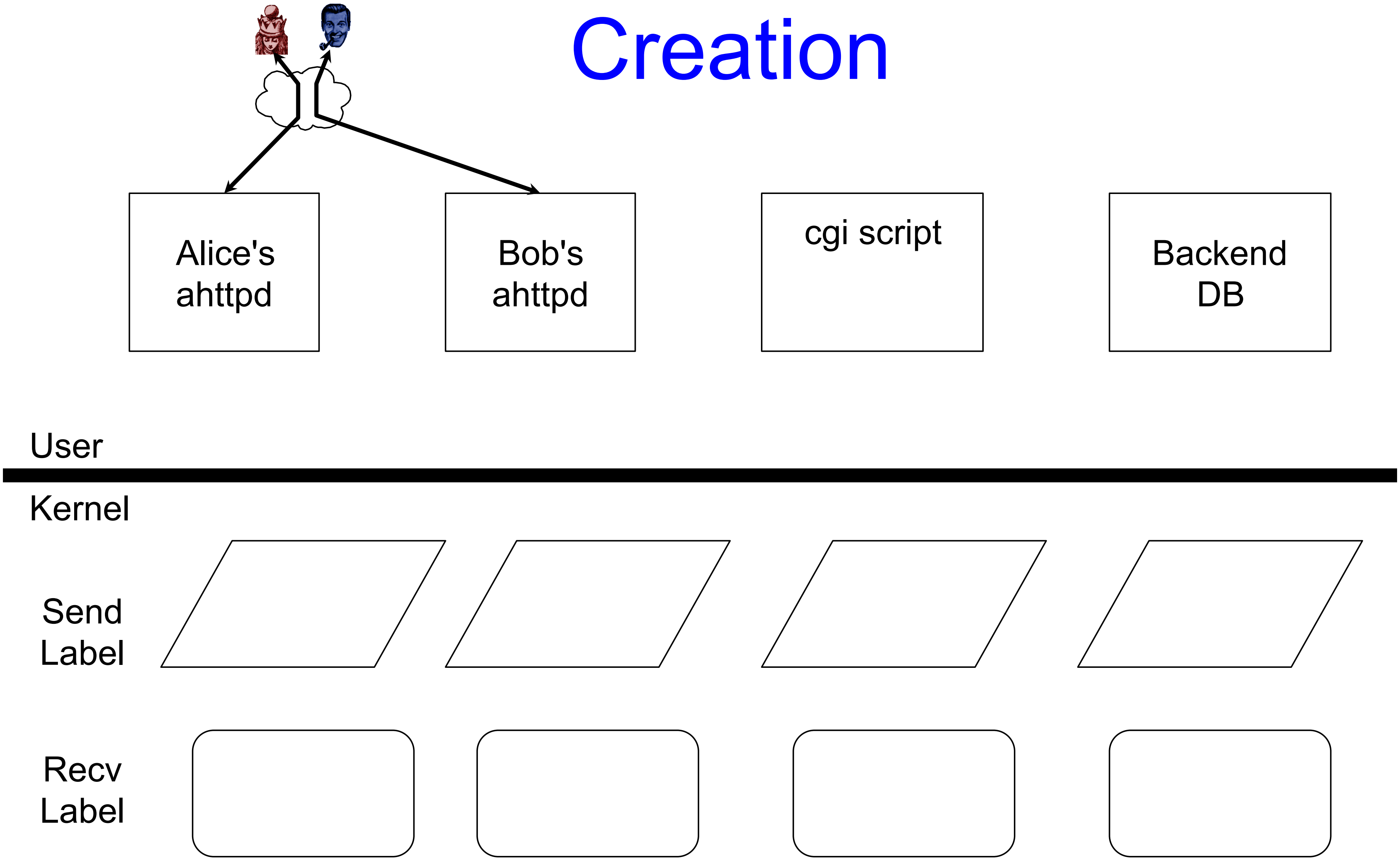
Limiting Bug Impact



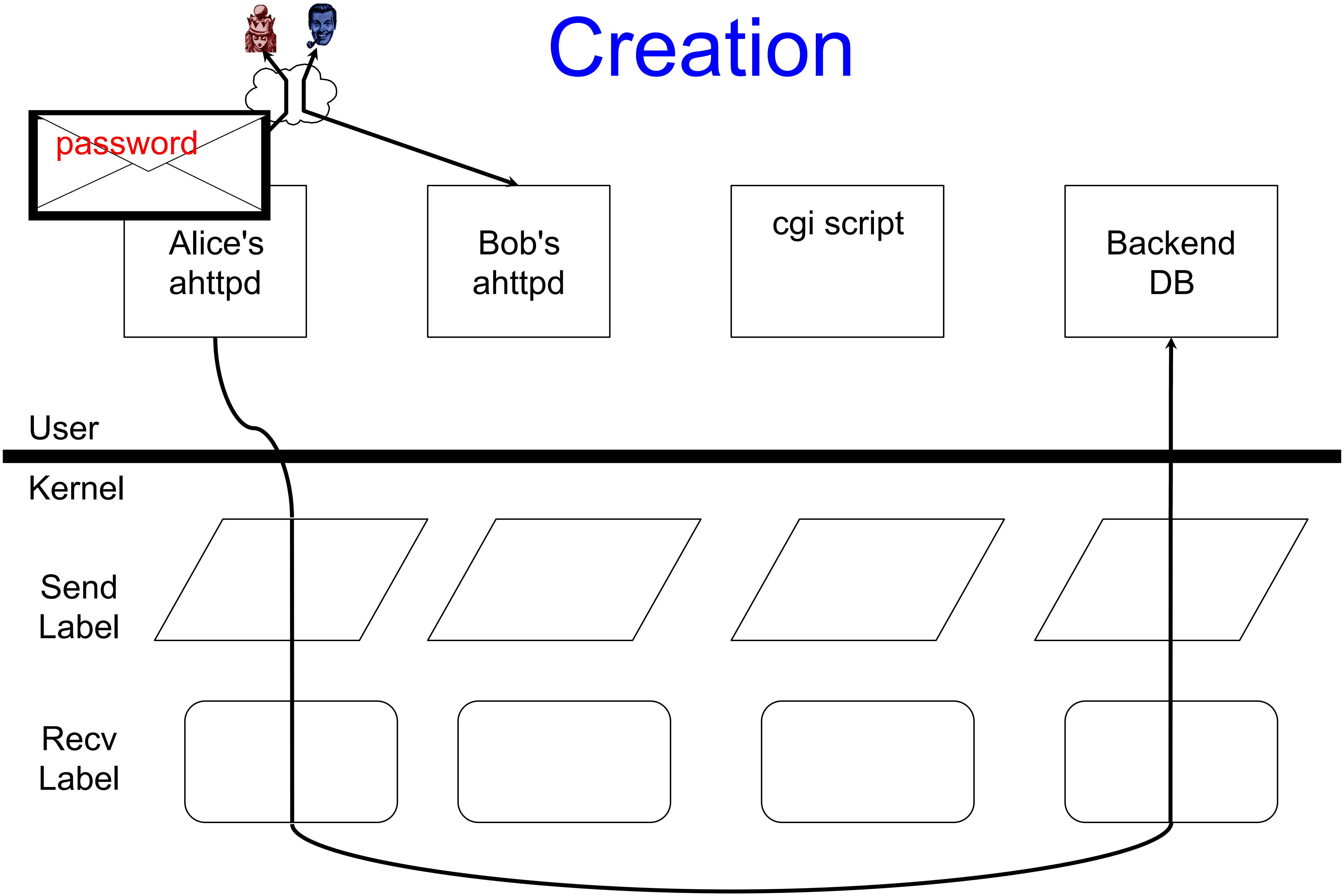
Application Defined Policies

- 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

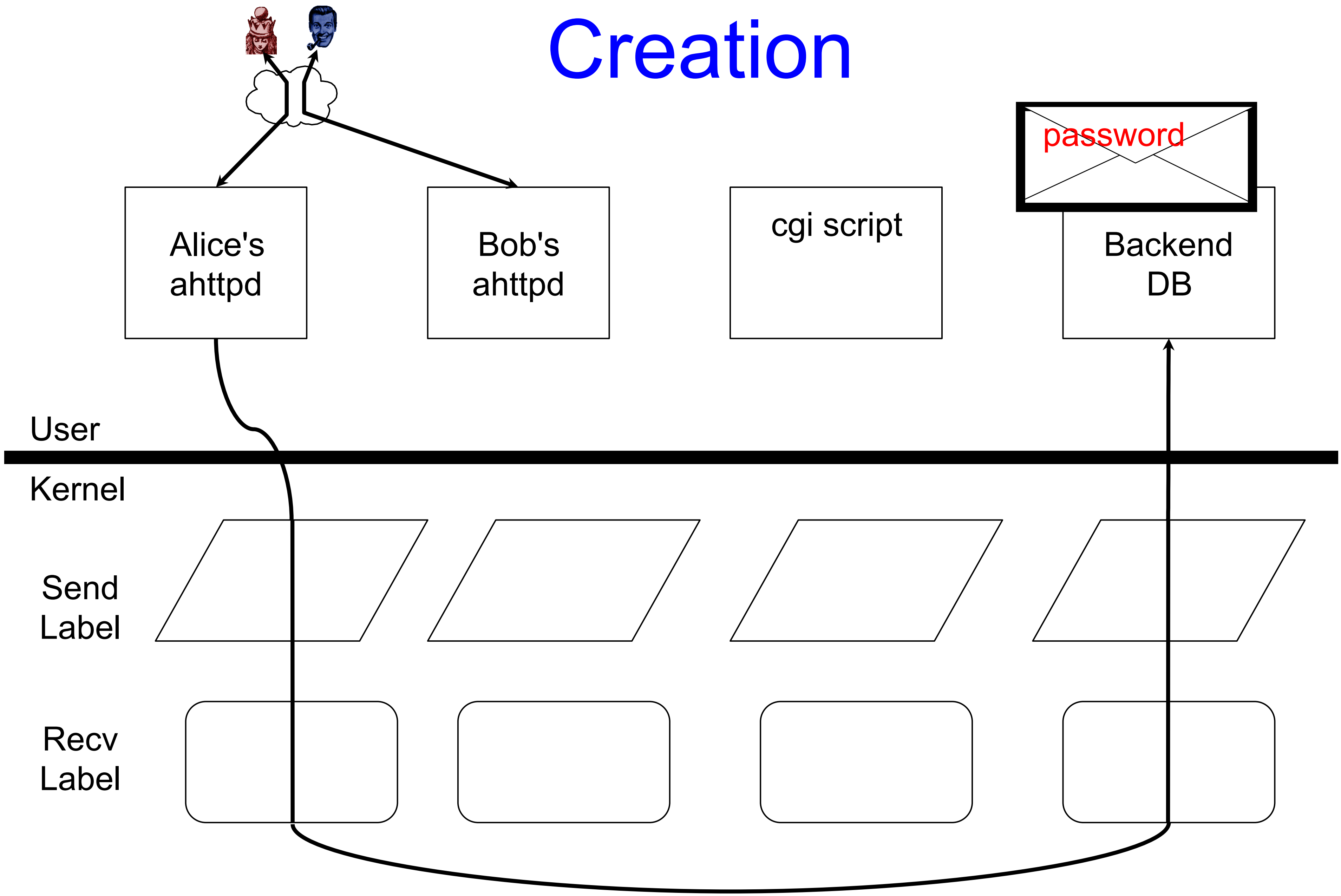
Compartment Creation



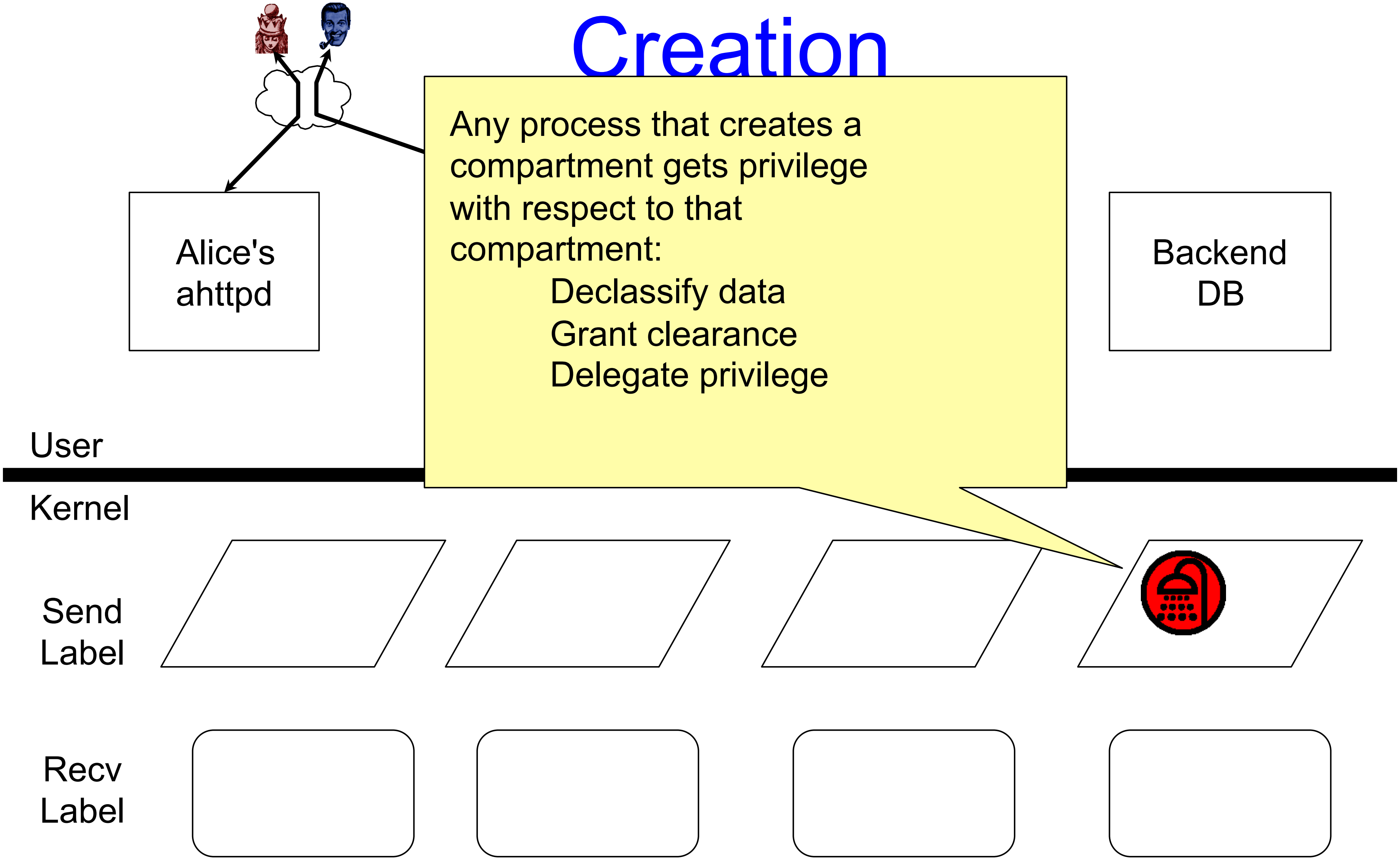
Compartment Creation



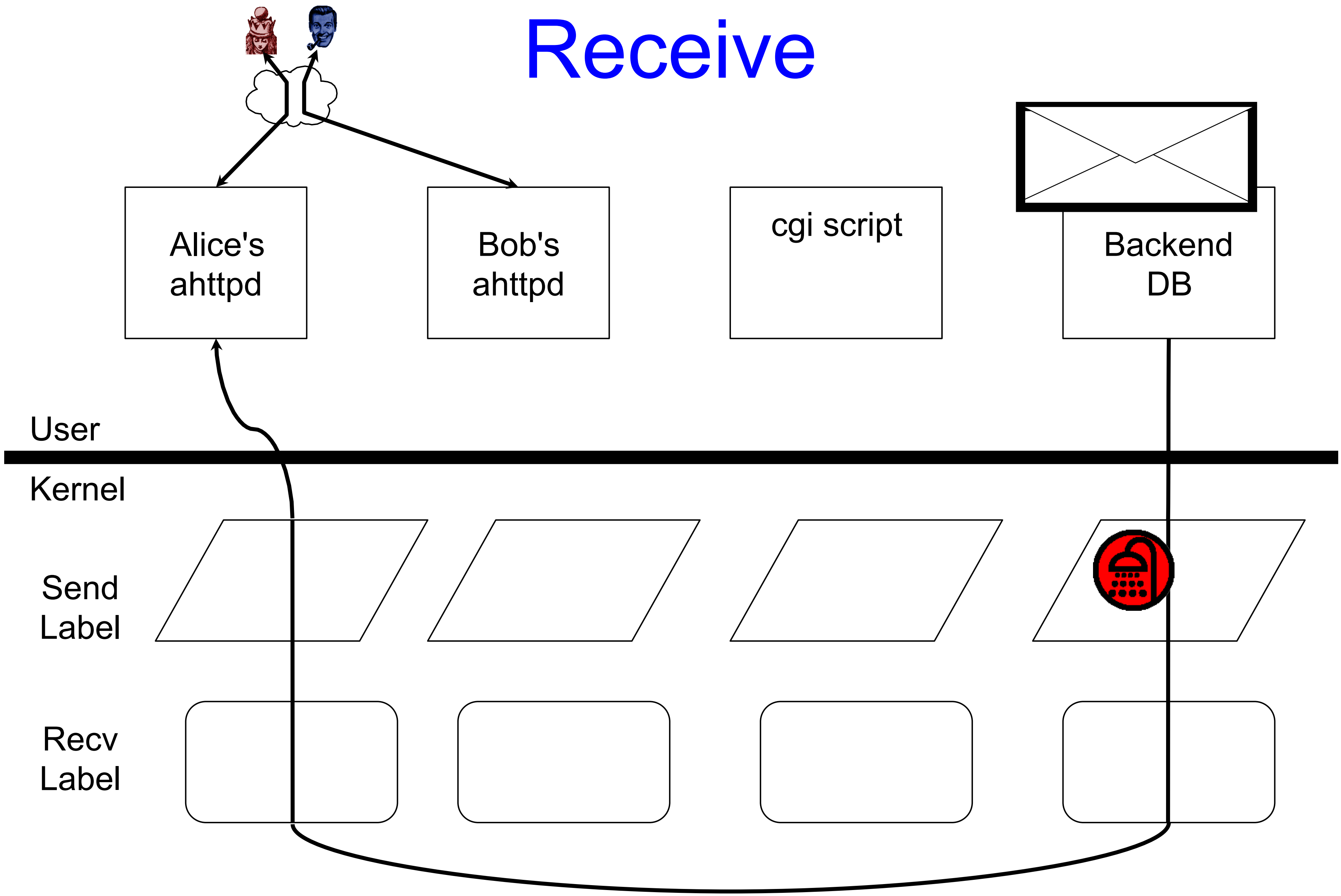
Compartment Creation



Compartment Creation



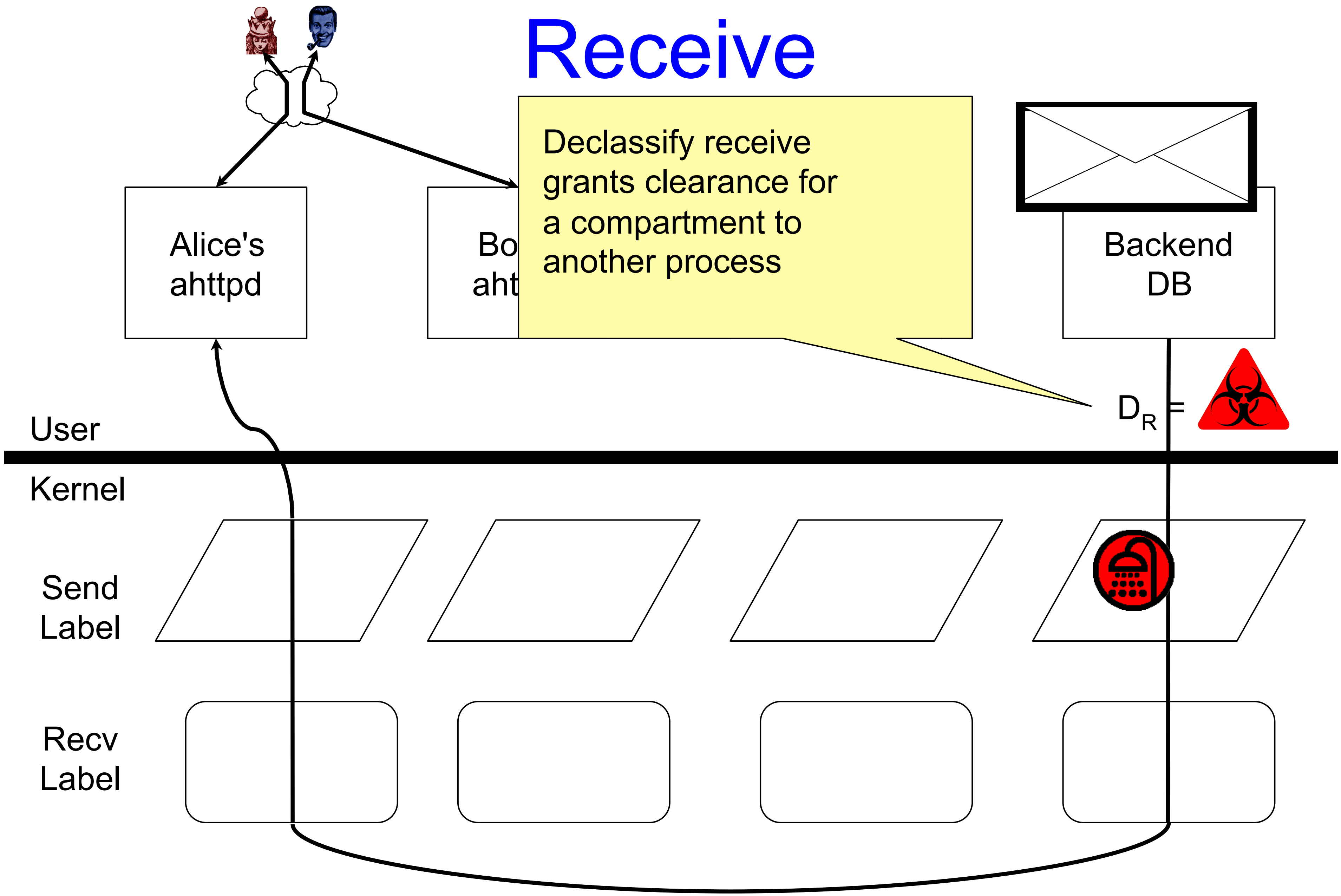
Declassify Receive



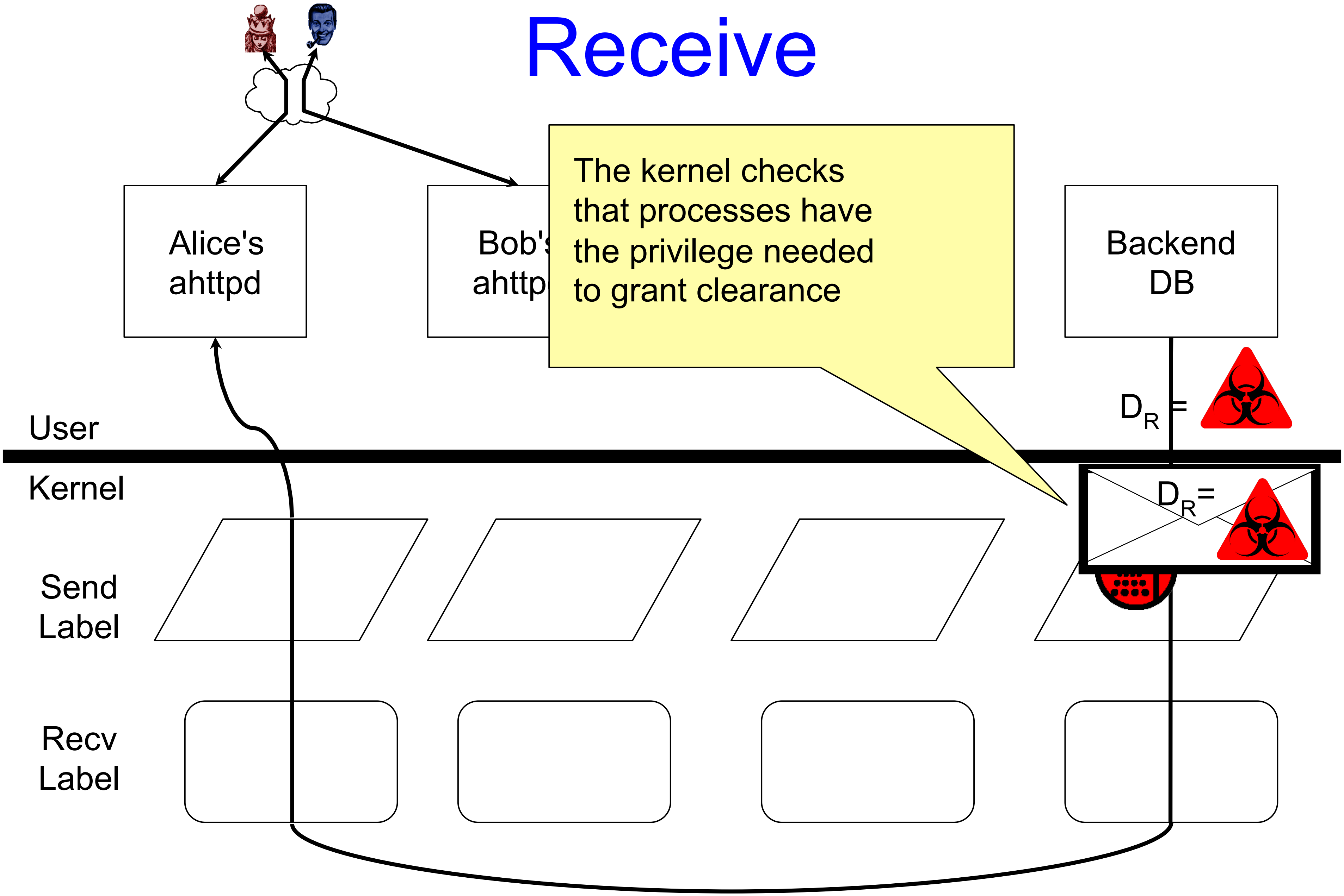
Optional Labels

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

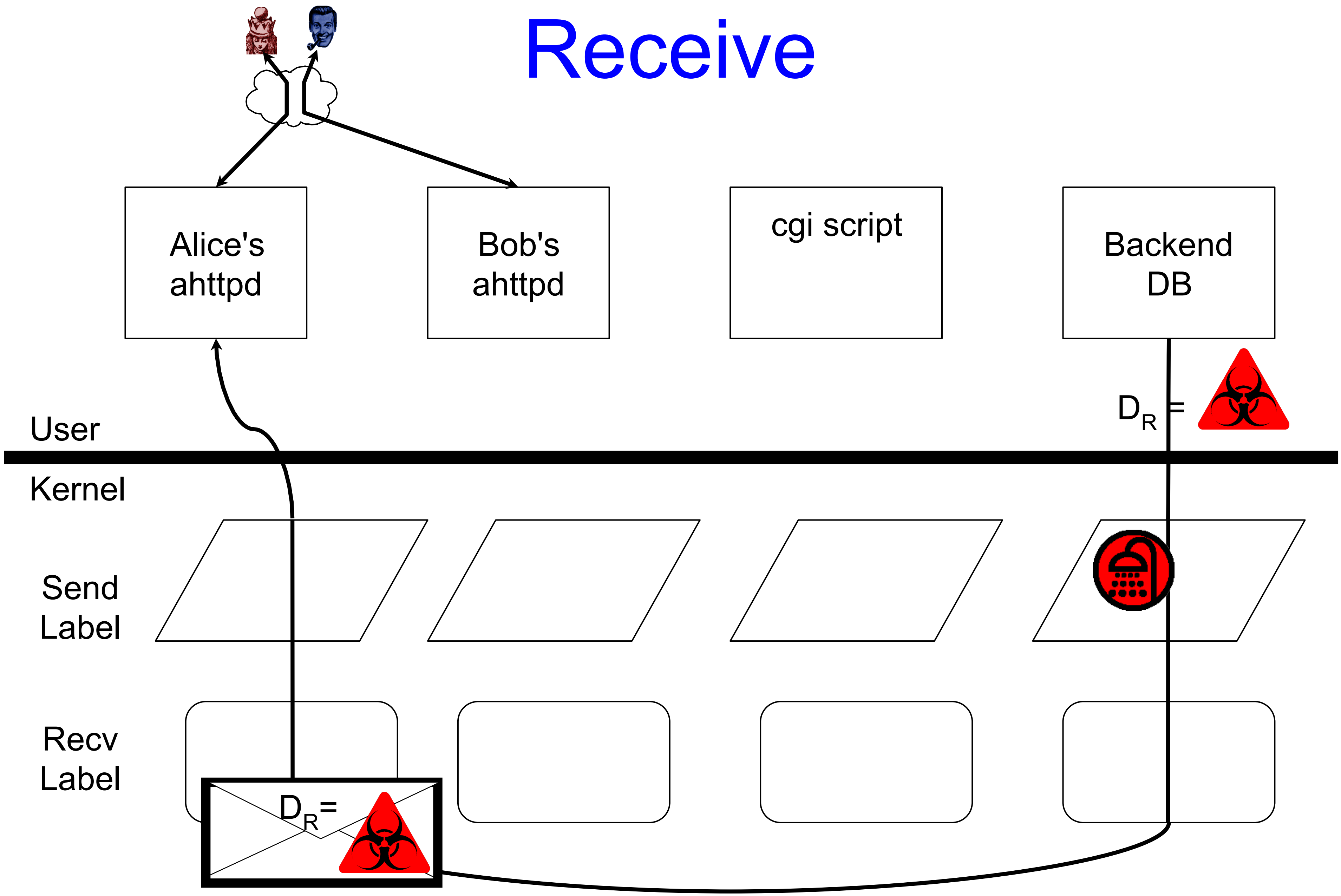
Declassify Receive



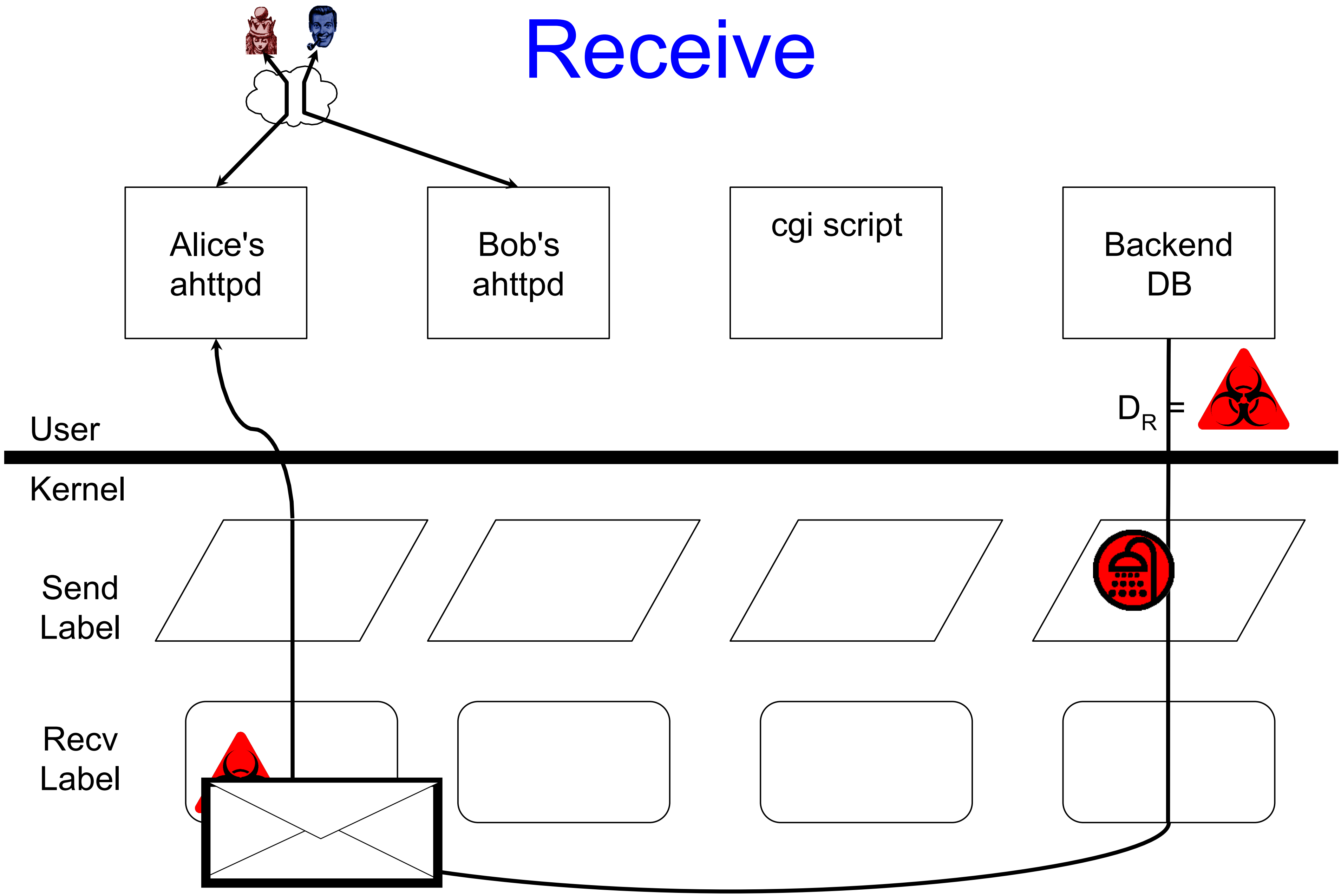
Declassify Receive



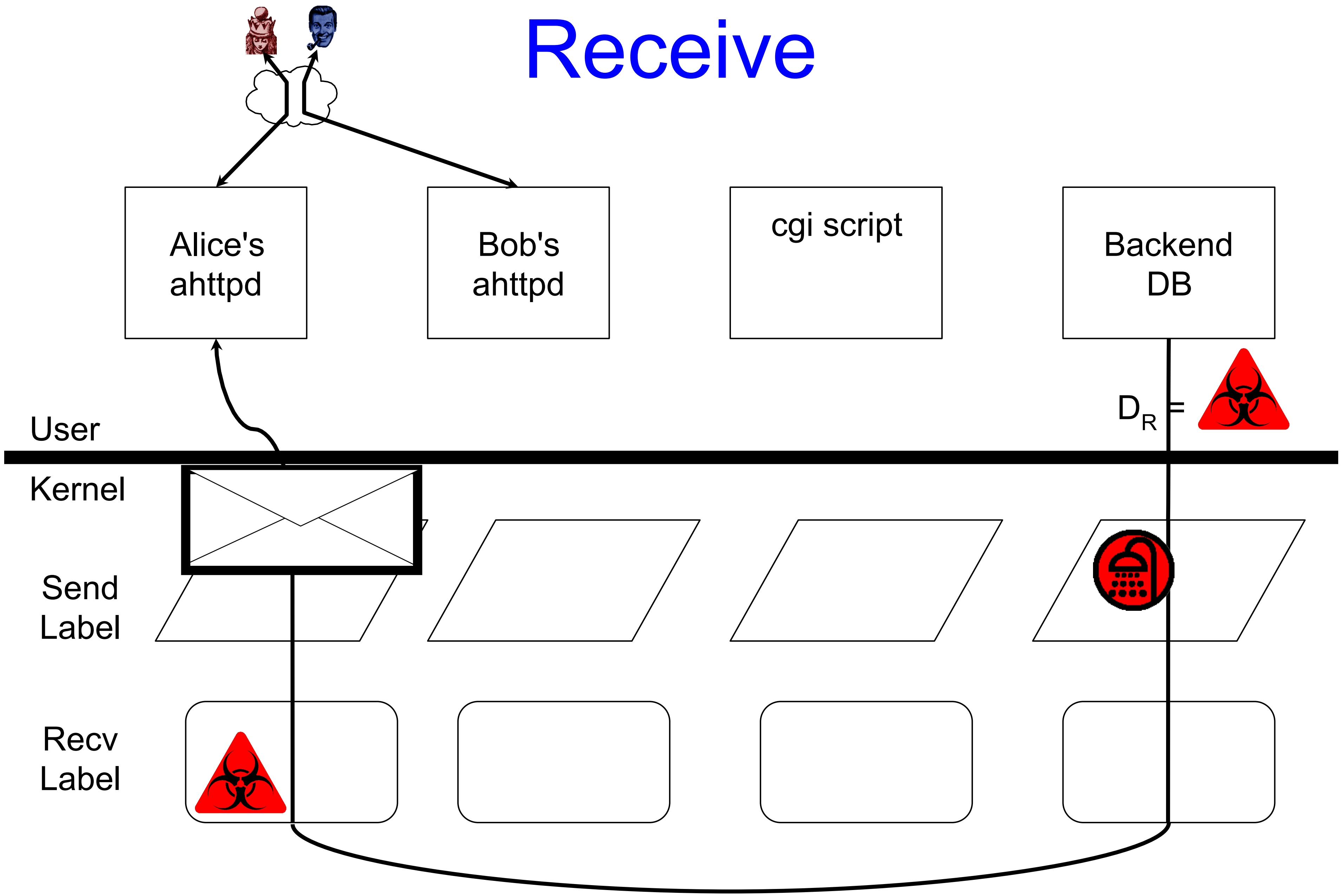
Declassify Receive



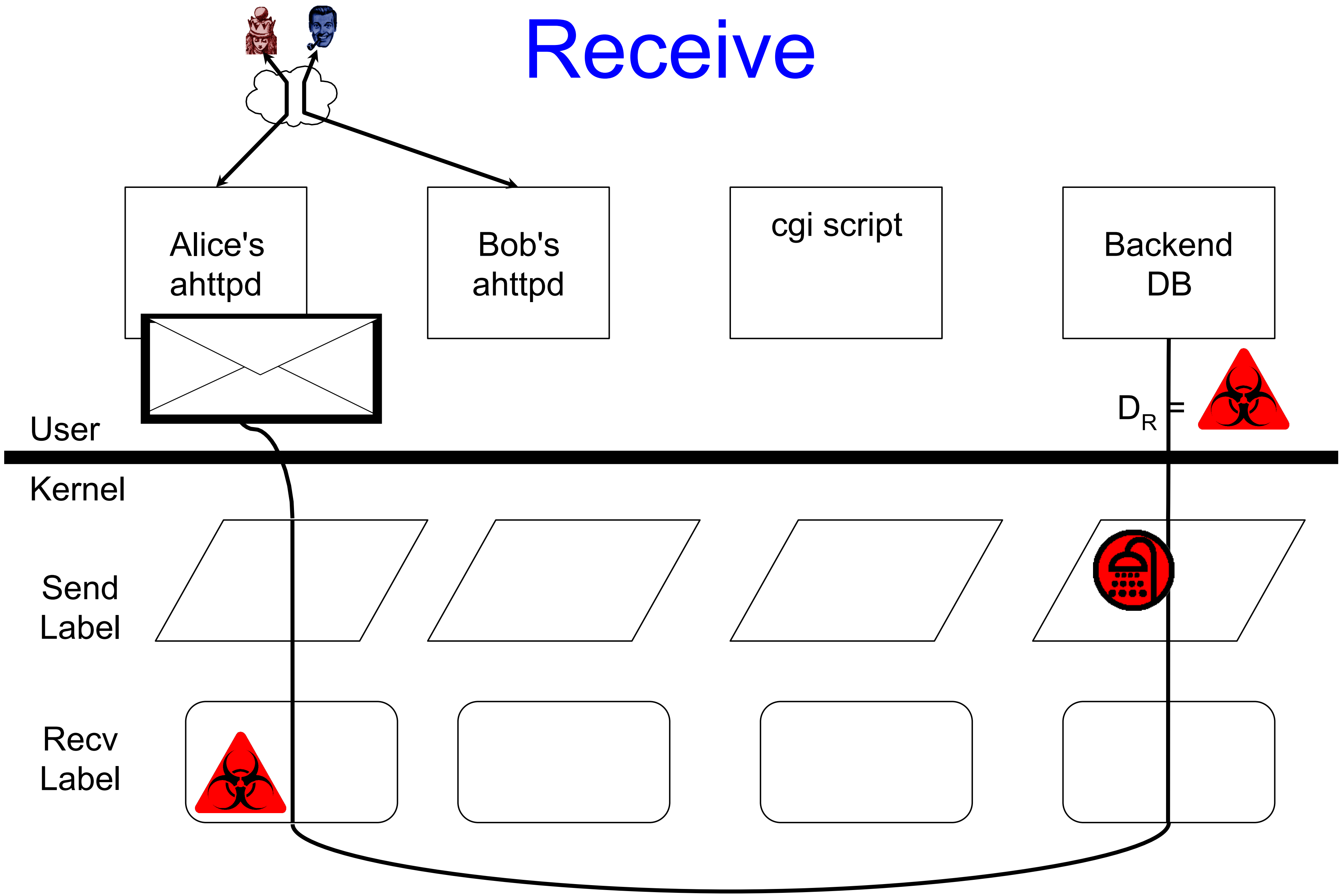
Declassify Receive



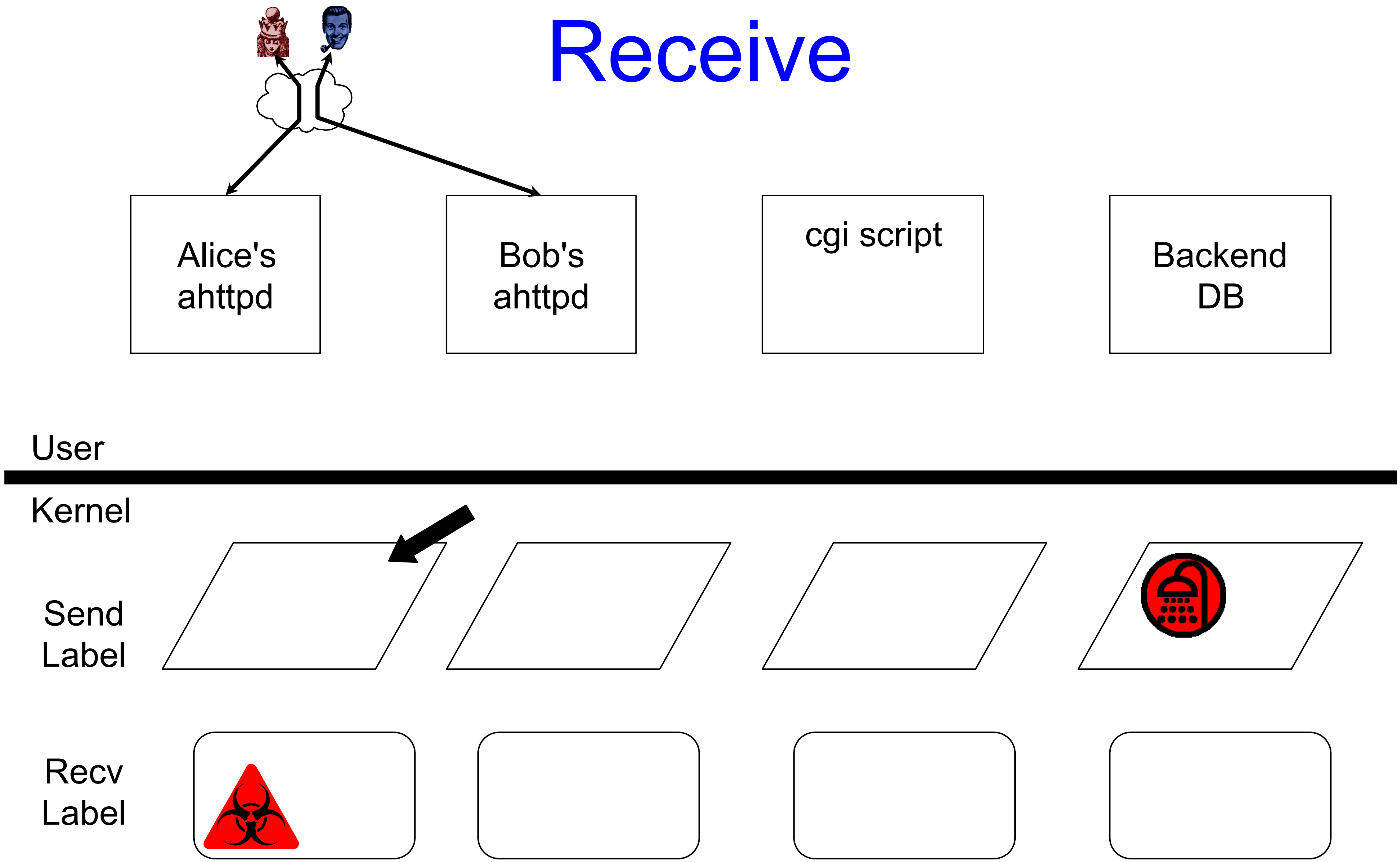
Declassify Receive



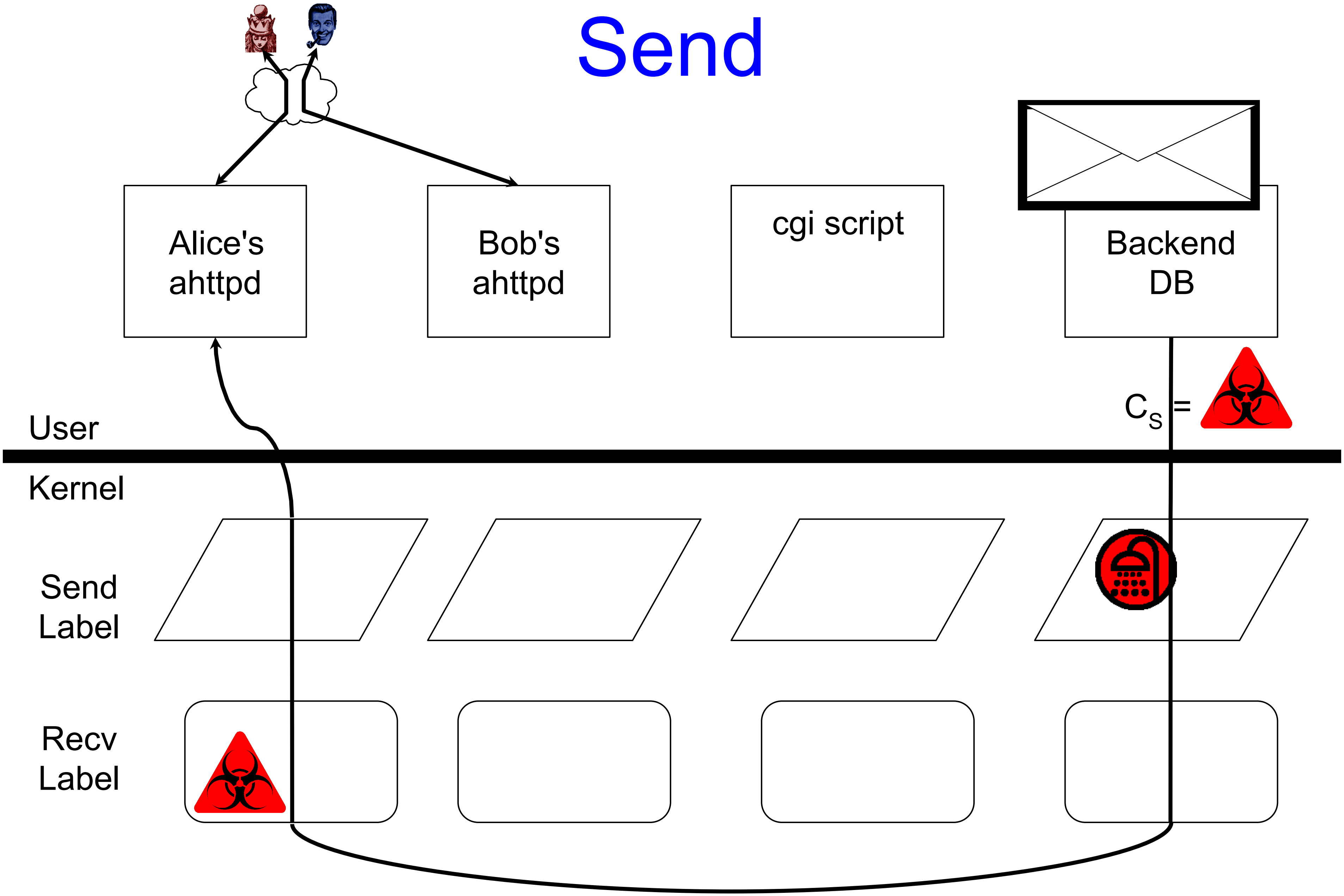
Declassify Receive



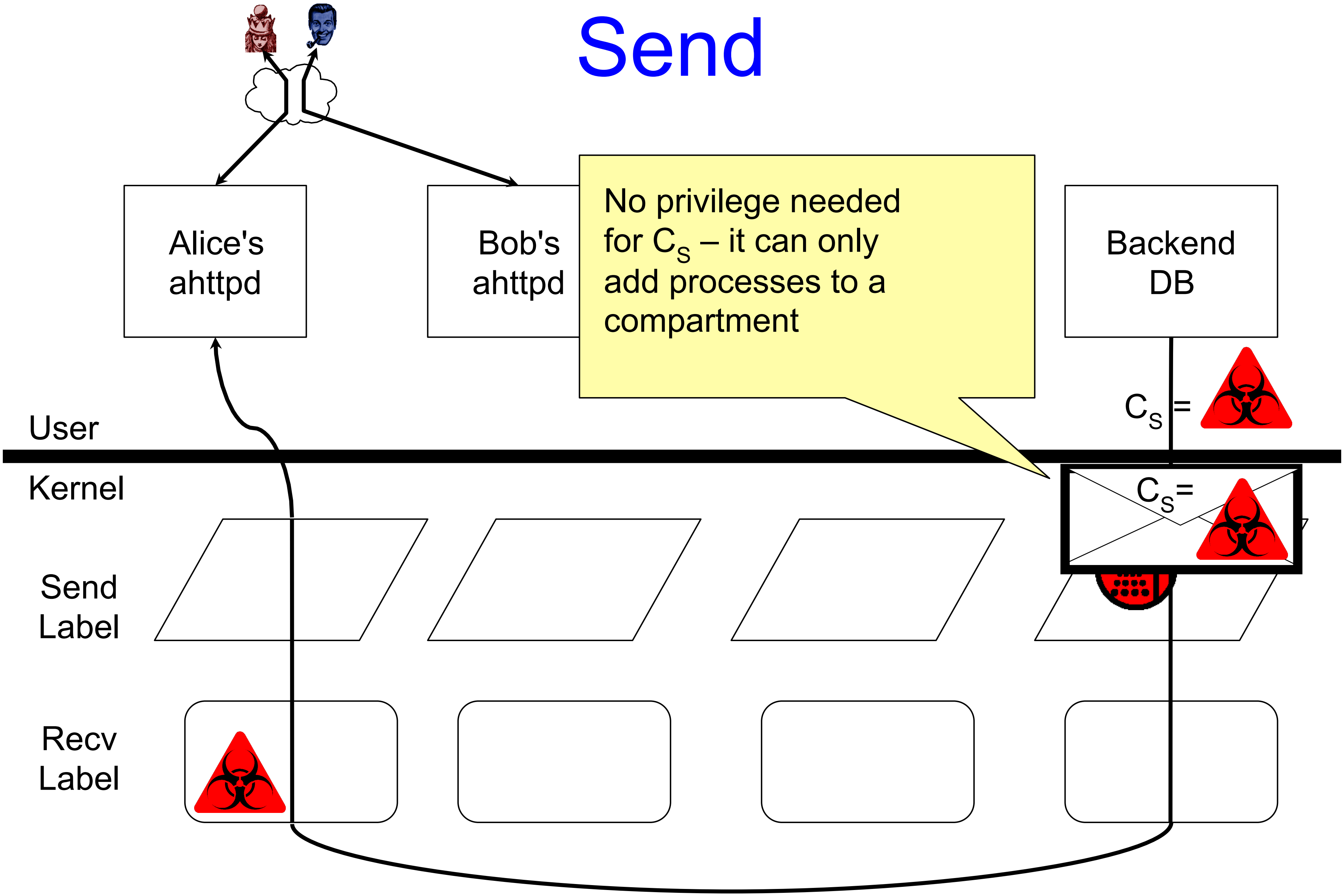
Declassify Receive



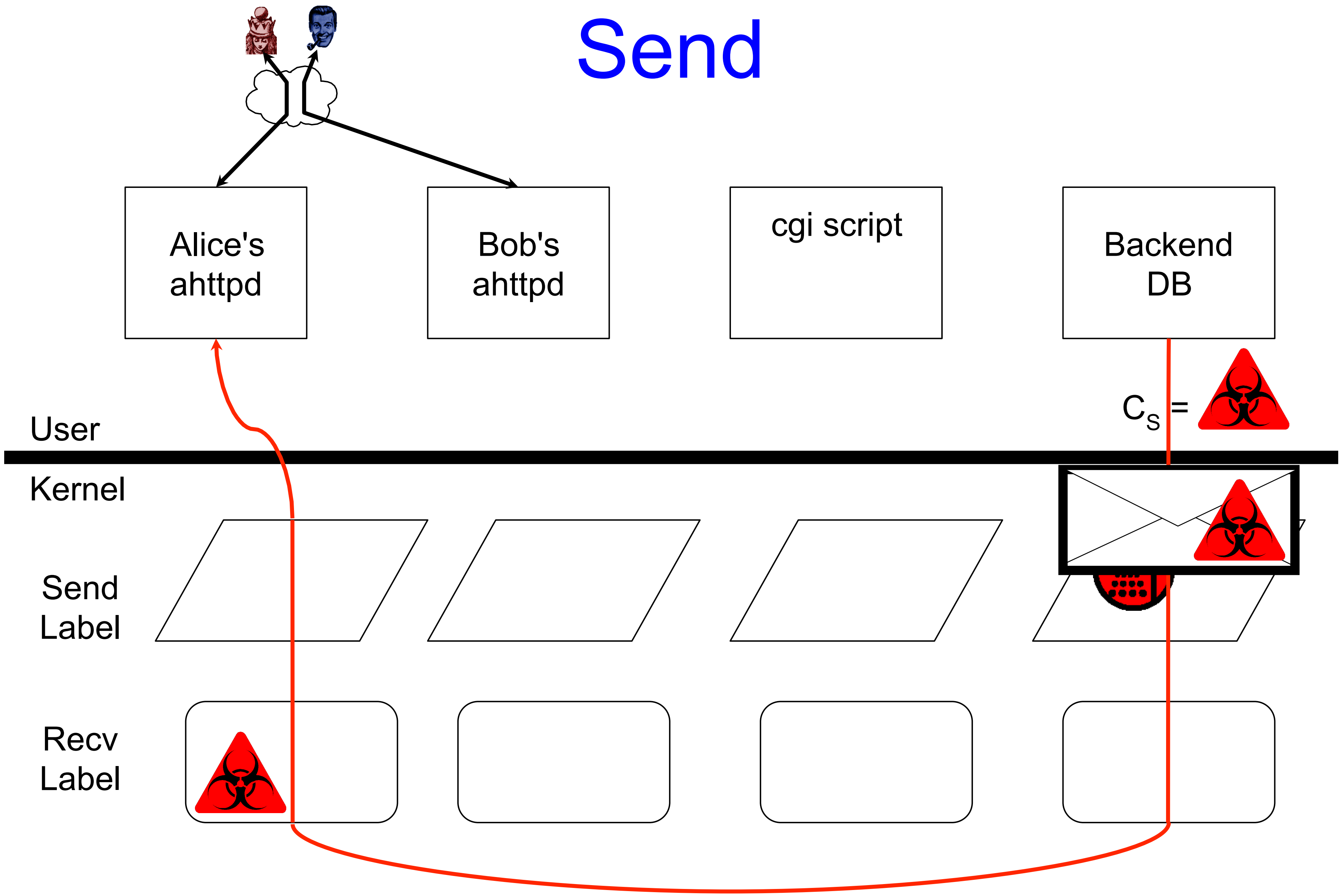
Contaminate Send



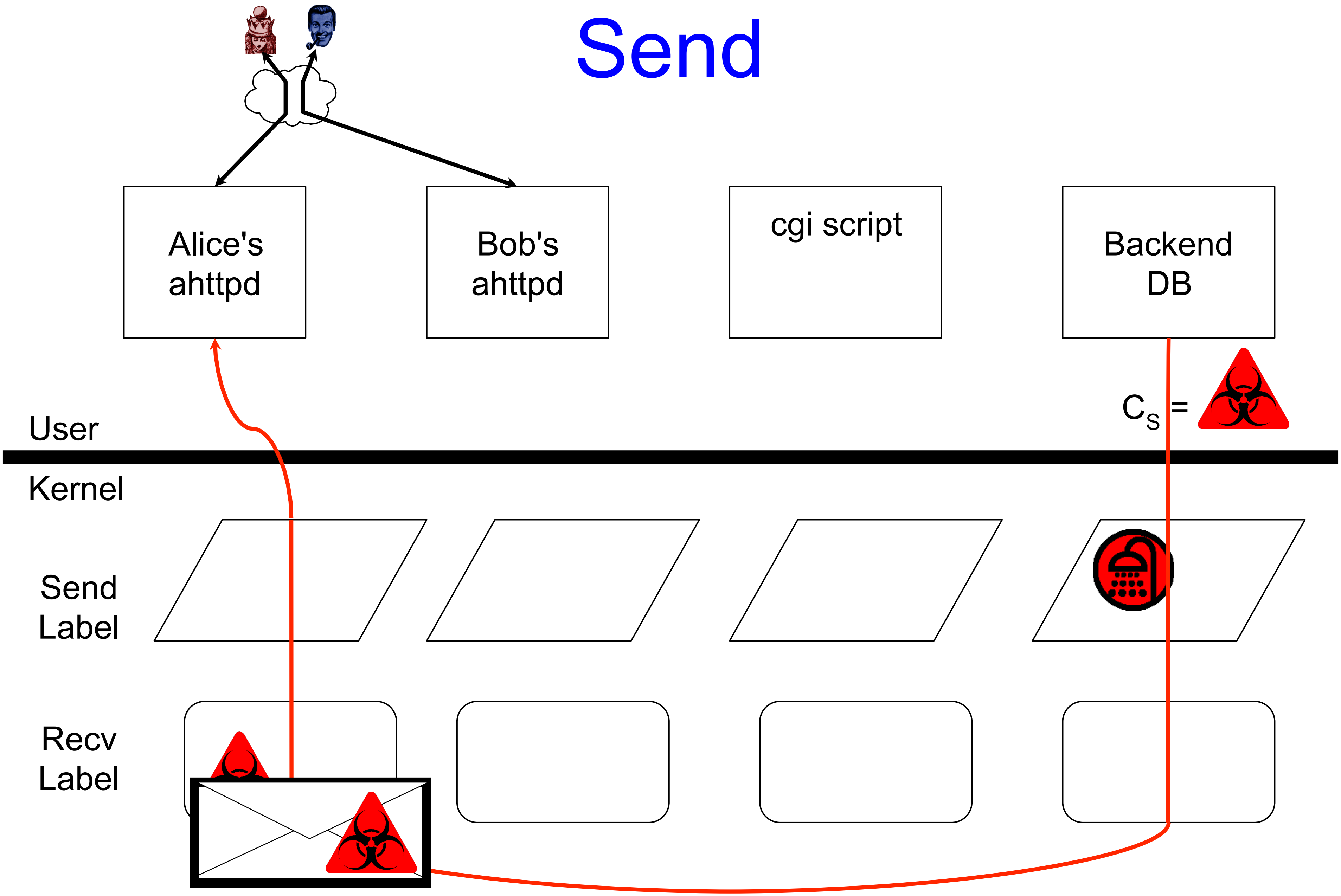
Contaminate Send



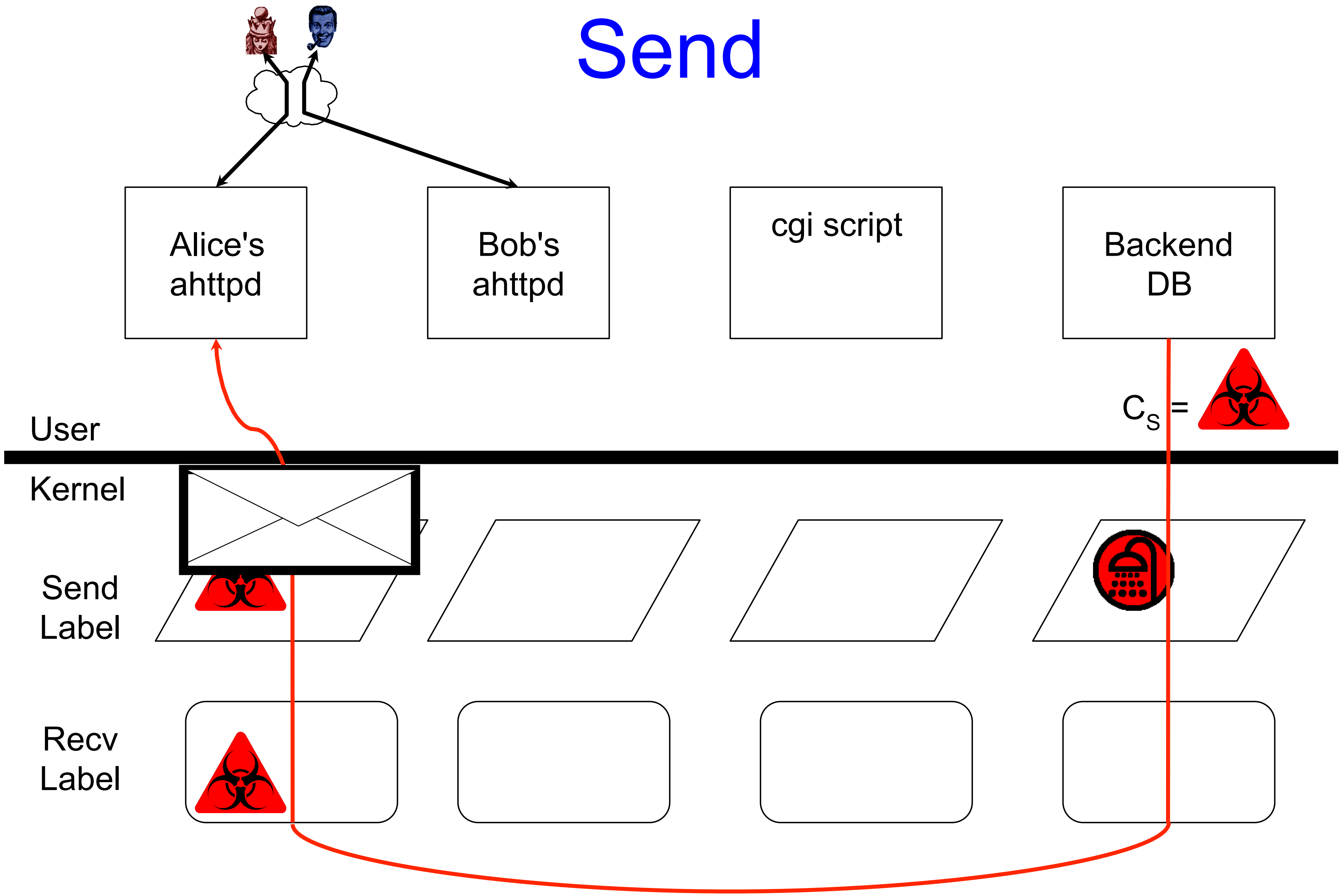
Contaminate Send



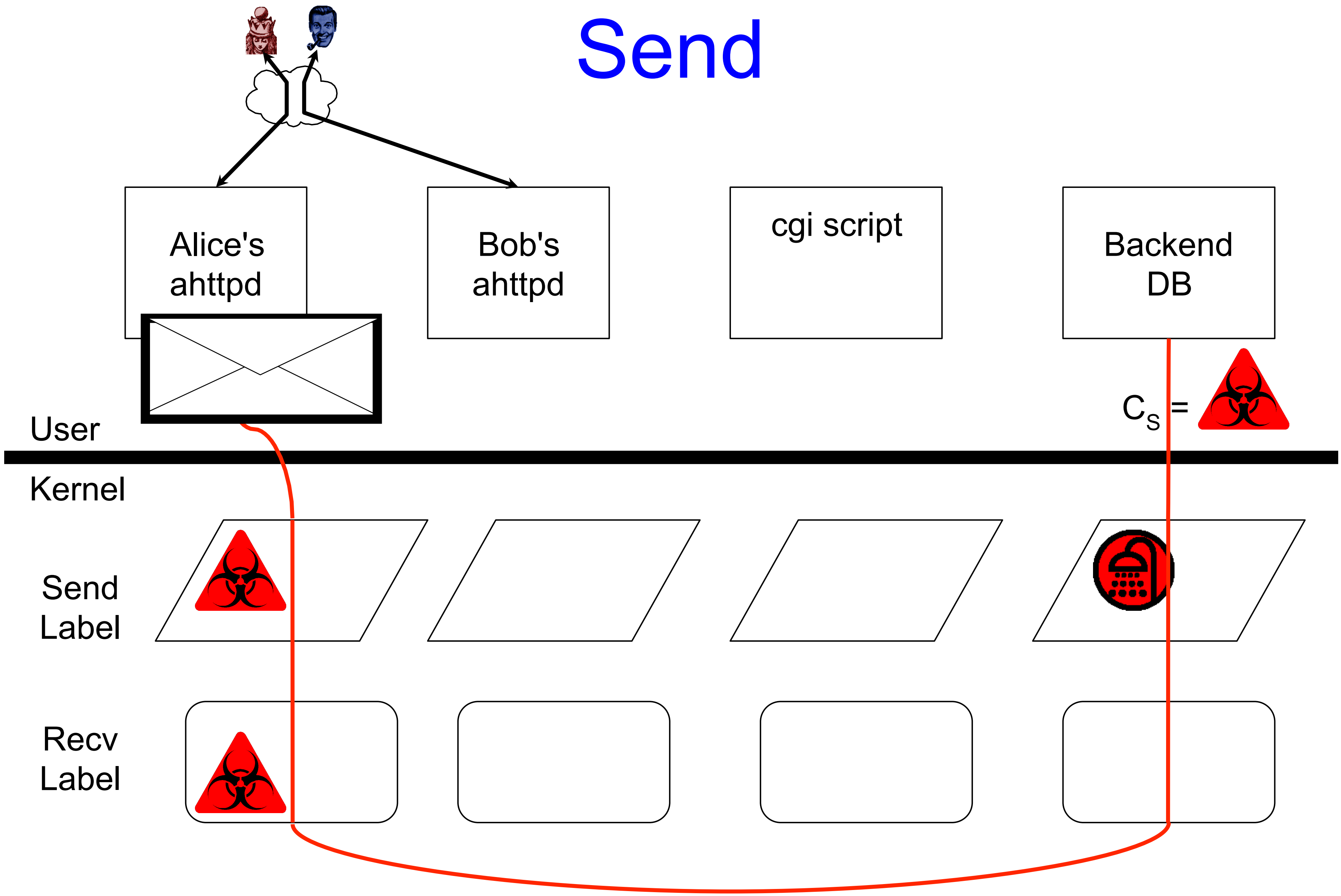
Contaminate Send



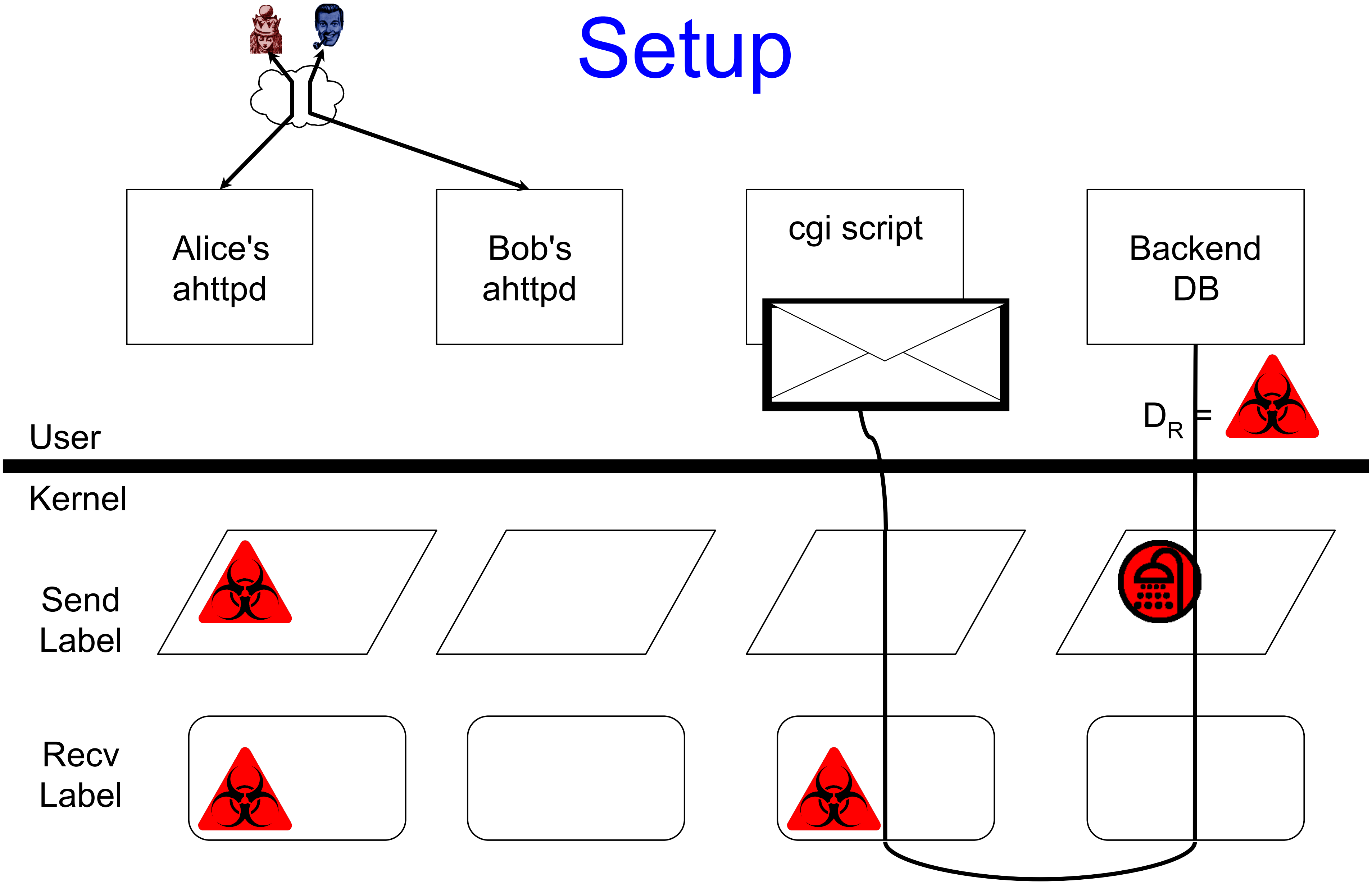
Contaminate Send



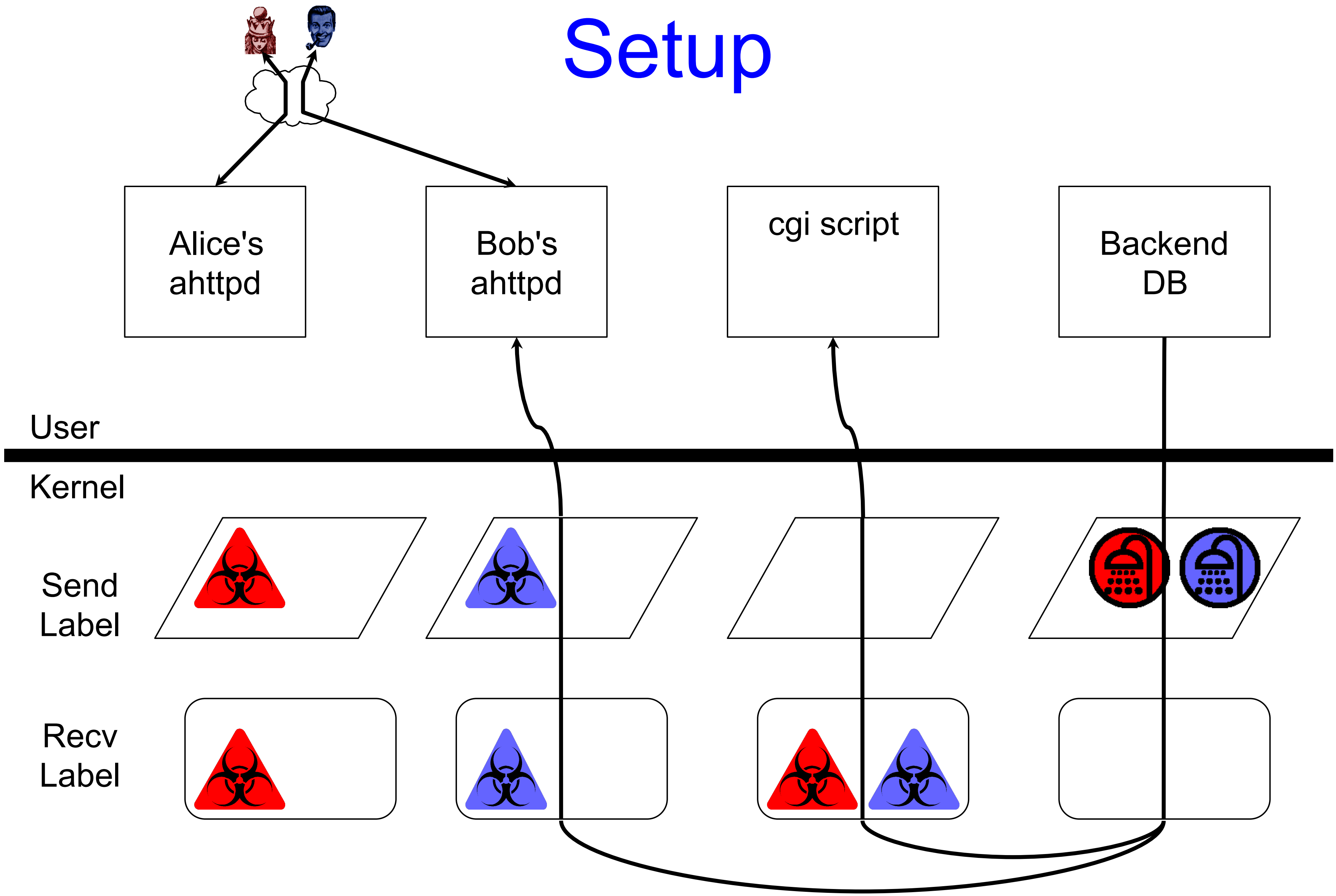
Contaminate Send



CGI Setup

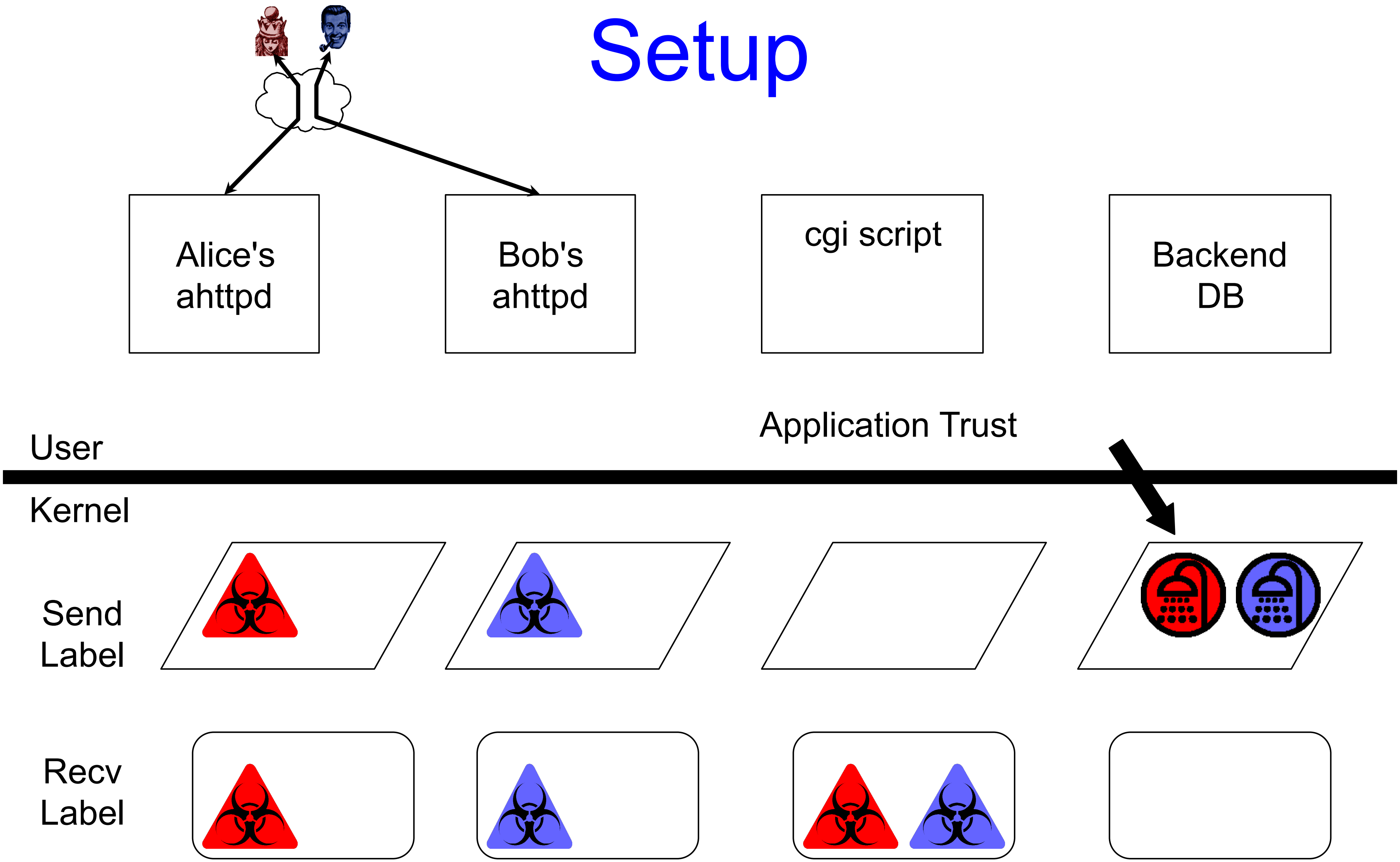

Alice's ahttpd
Bob's ahttpd
cgi script
Backend DB
D_R =
User
Kernel
Send Label
Recv Label

Bob Setup



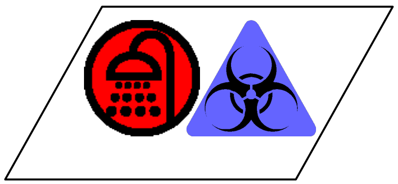
Alice's ahttpd
Bob's ahttpd
cgi script
Backend DB
User
Kernel
Send Label
Recv Label

Bob Setup


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

Label Implementation

- Contamination & Privilege = Label level (*, 0-3)

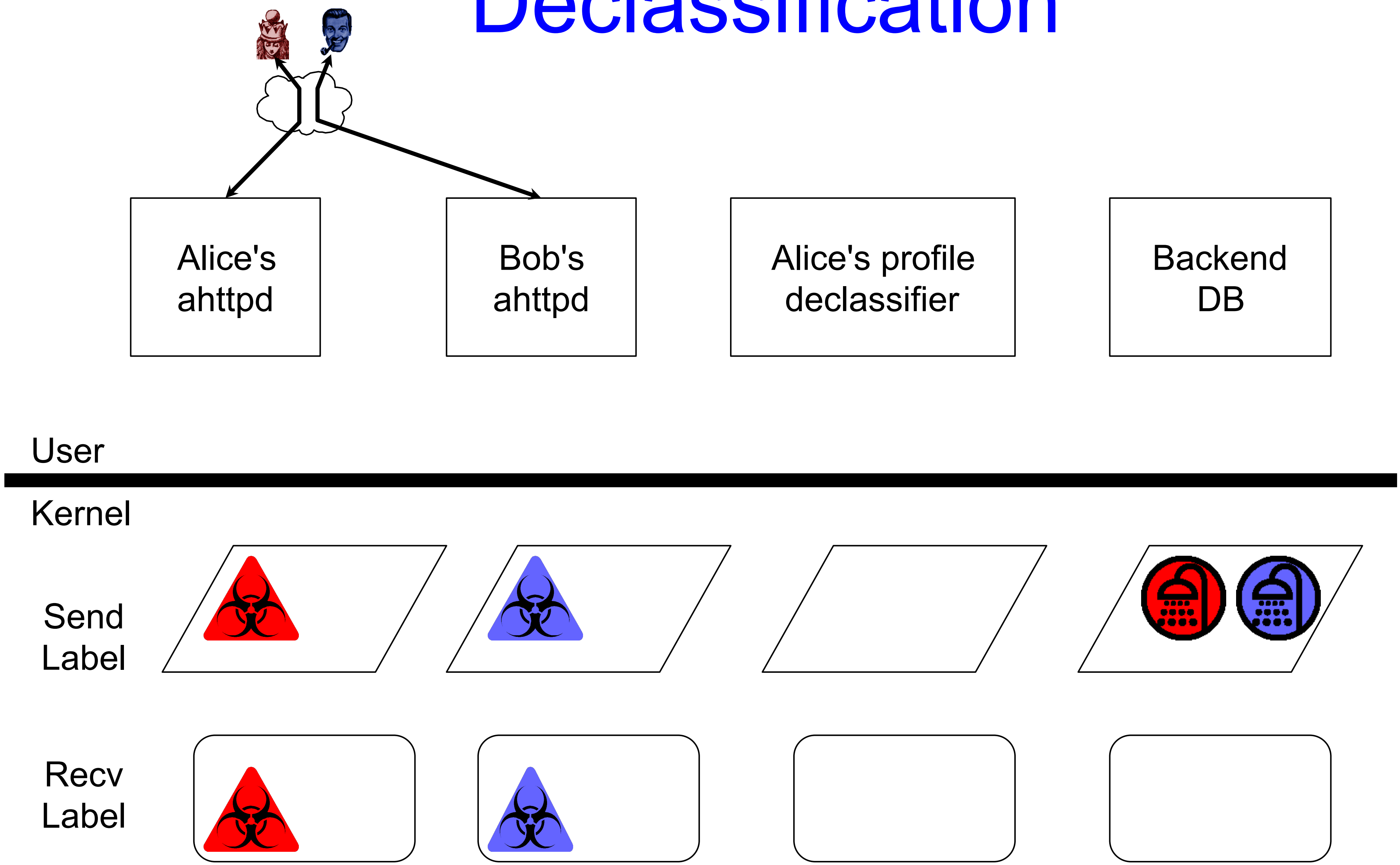


- = {A *, B 3, 1}
- A & B are compartment names
- Trailing 1 = Neutral in all other compartments
  - Including those that haven't been created yet
- Label representation linear in # compartments

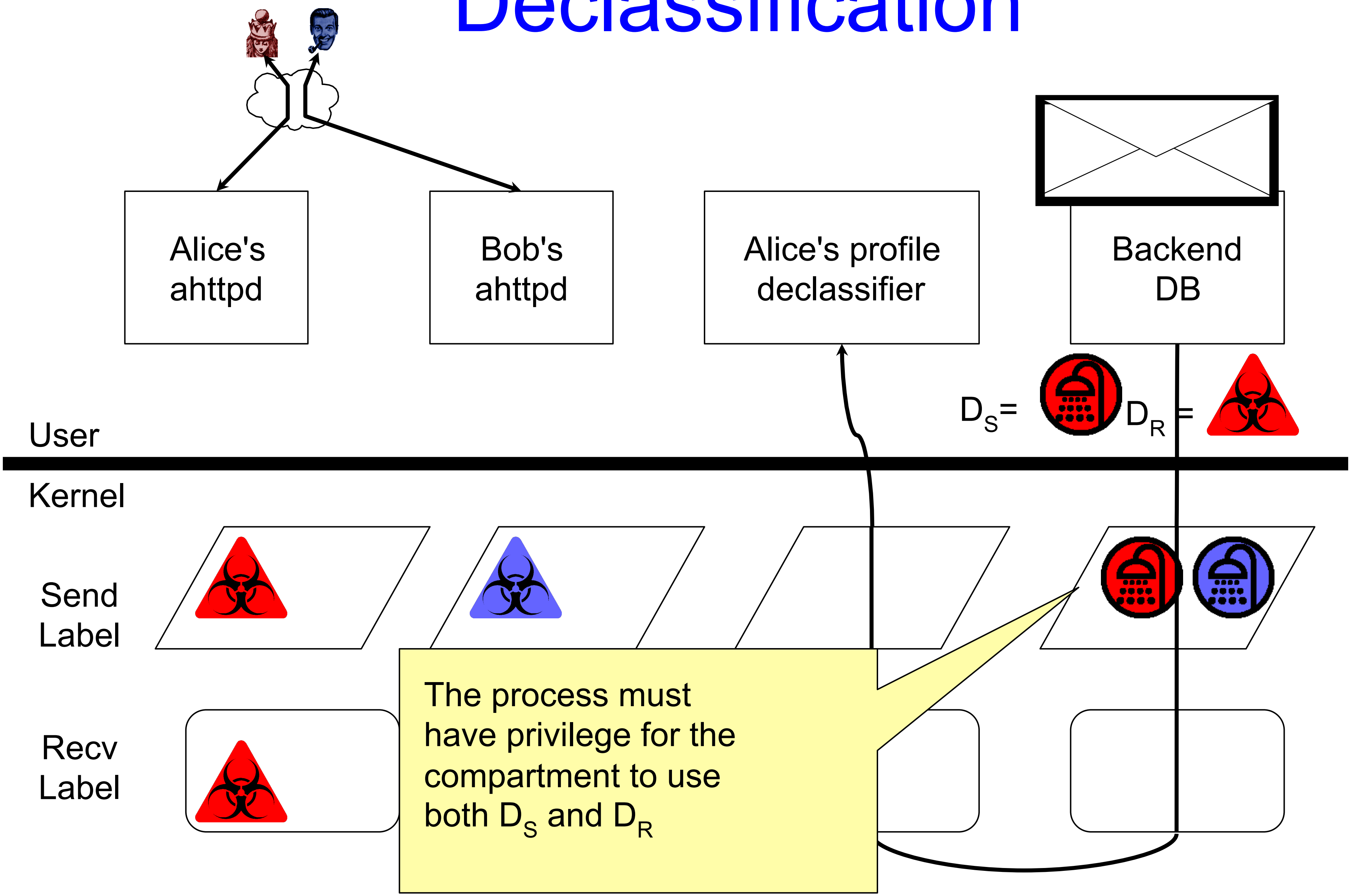
Declassification

- 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 process that can do so
- To minimize declassification bugs, we build declassifiers as simple, single purpose programs

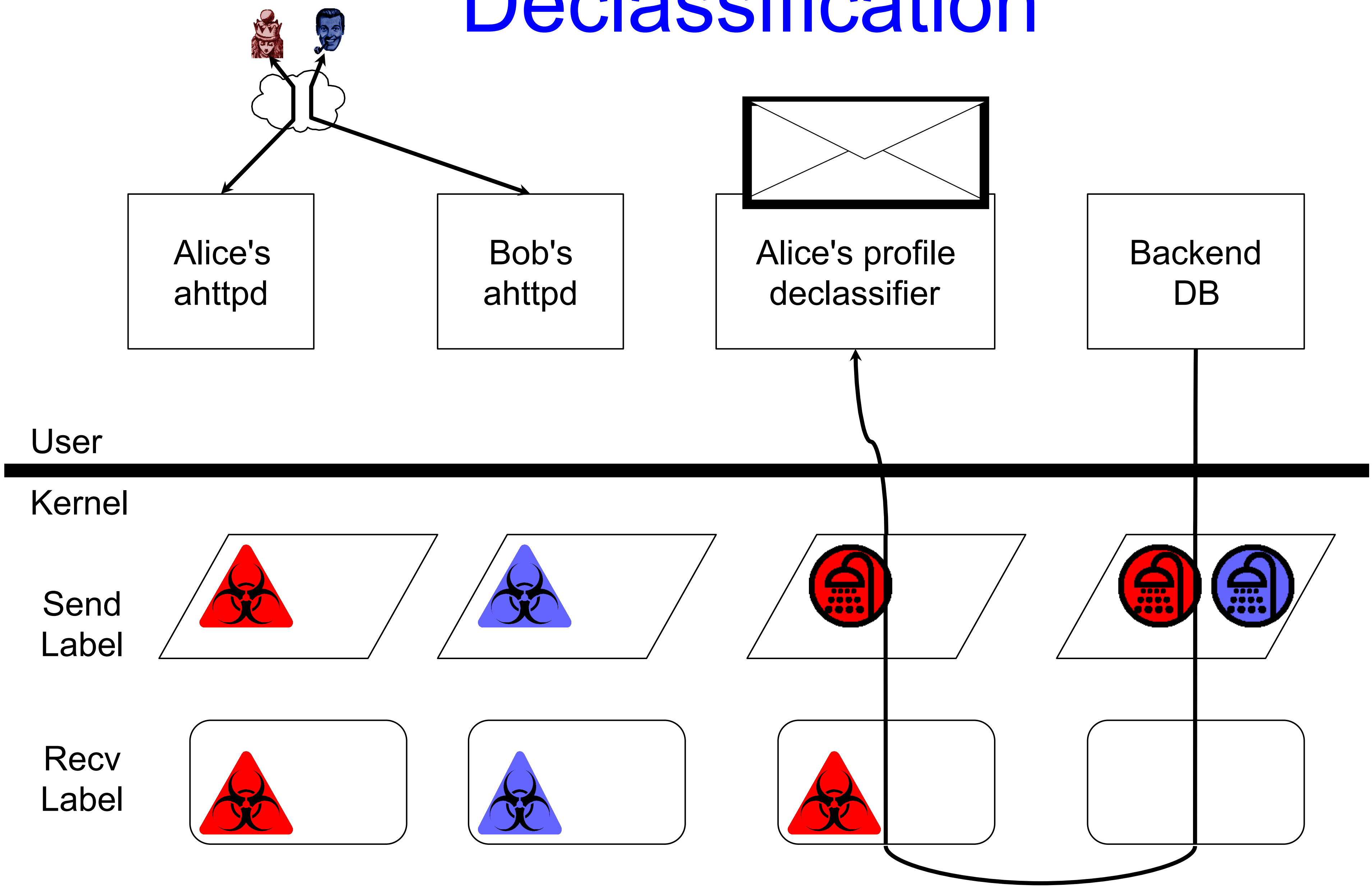
Declassification



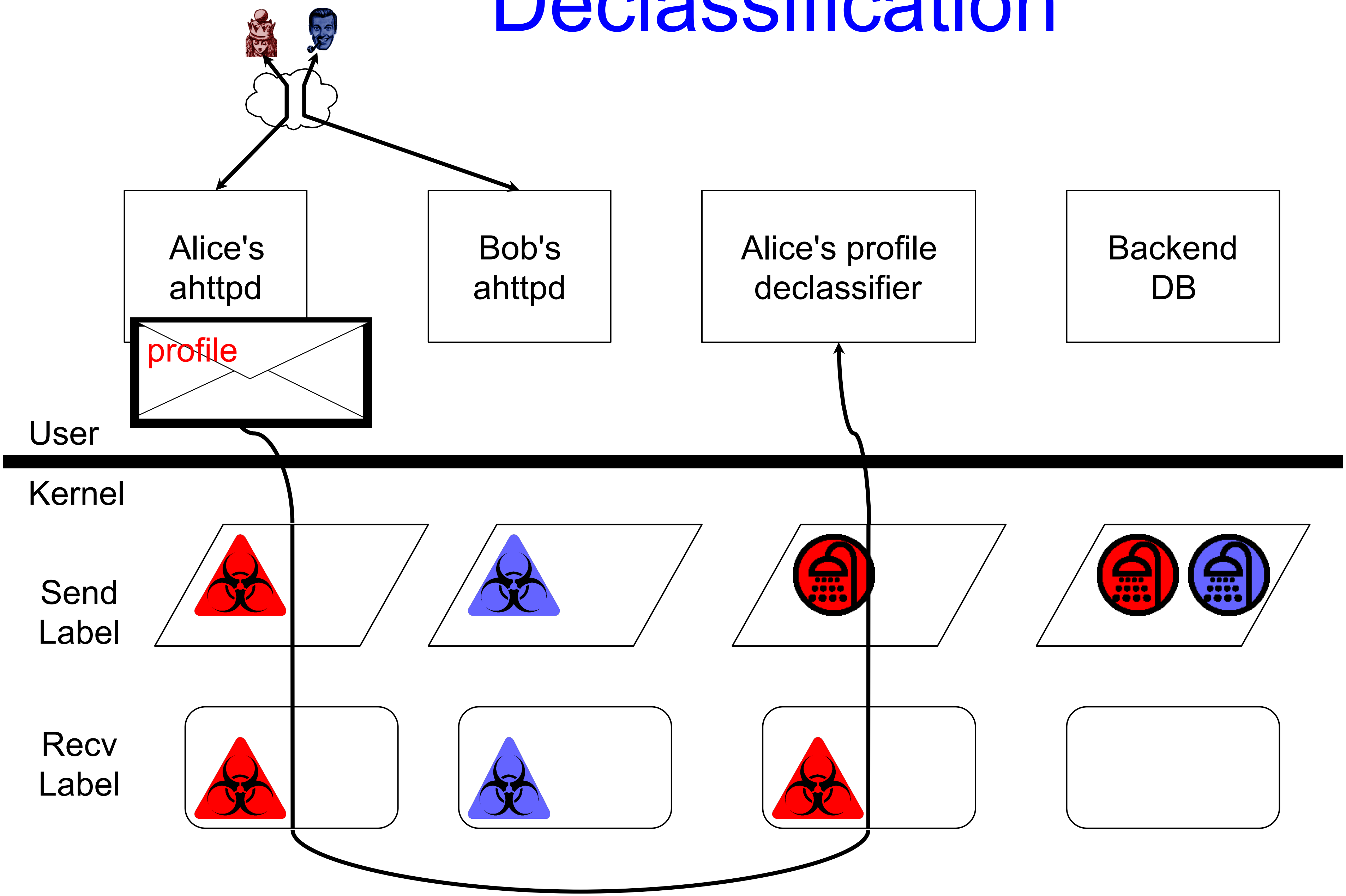
Declassification



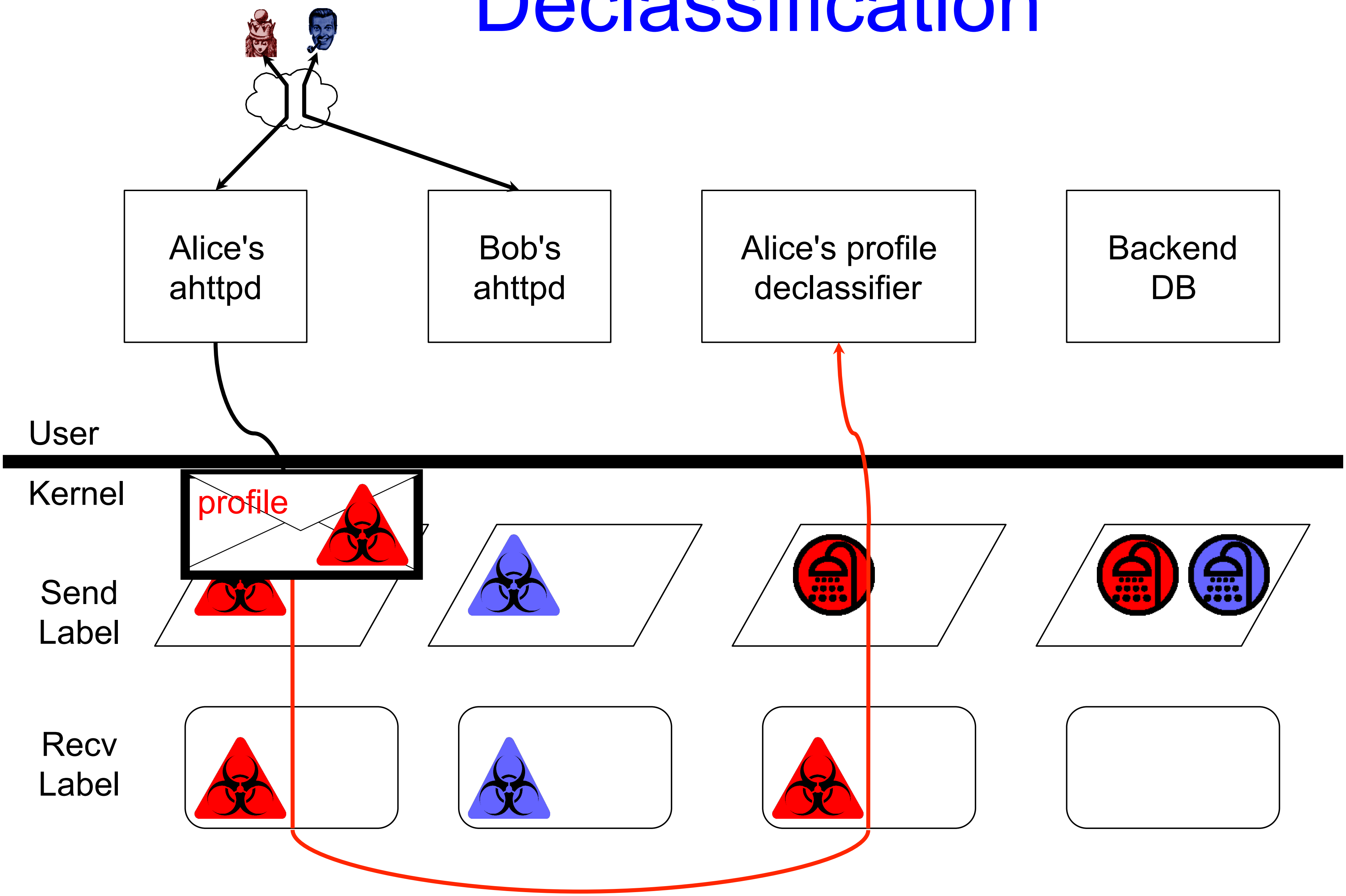
Declassification



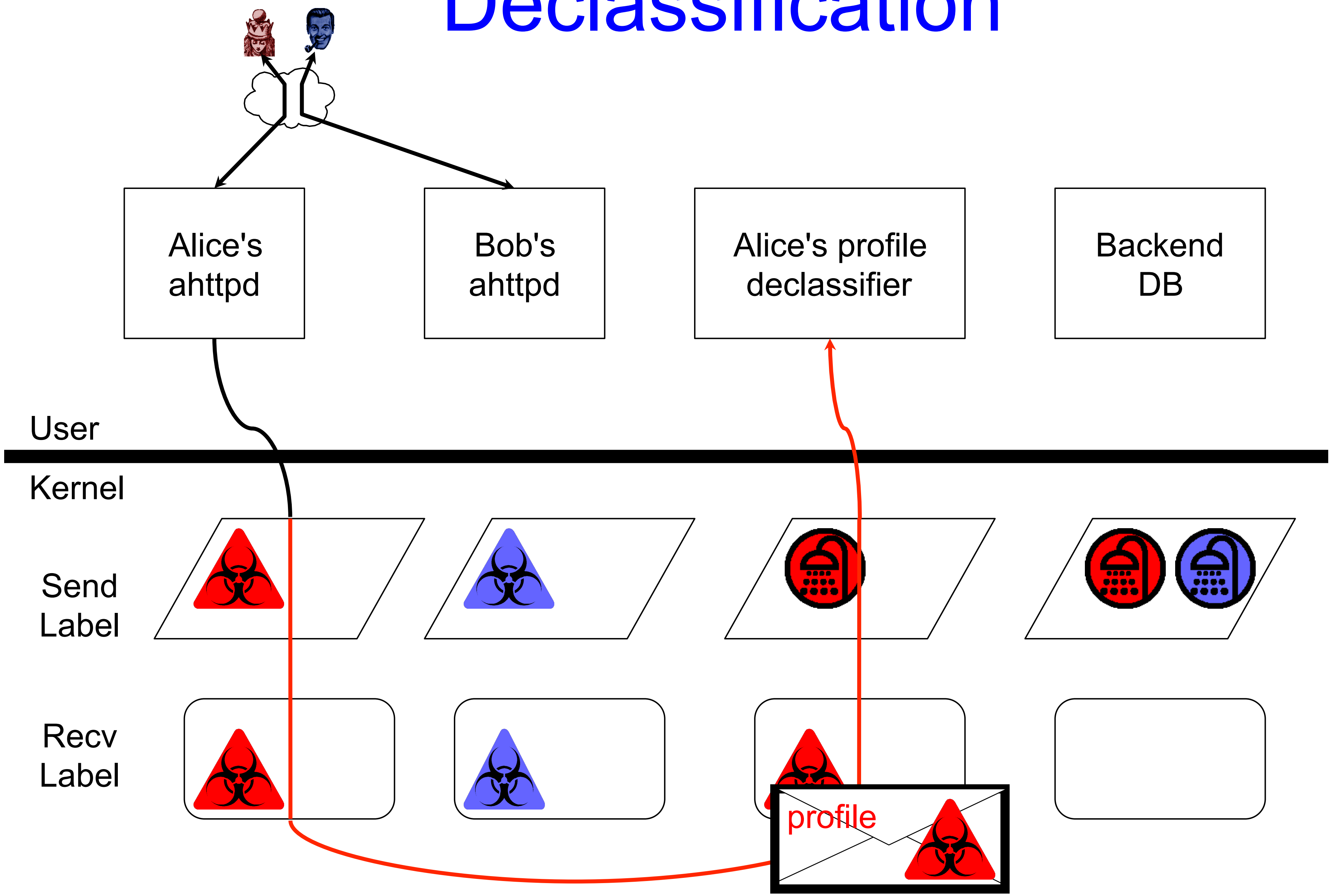
Declassification



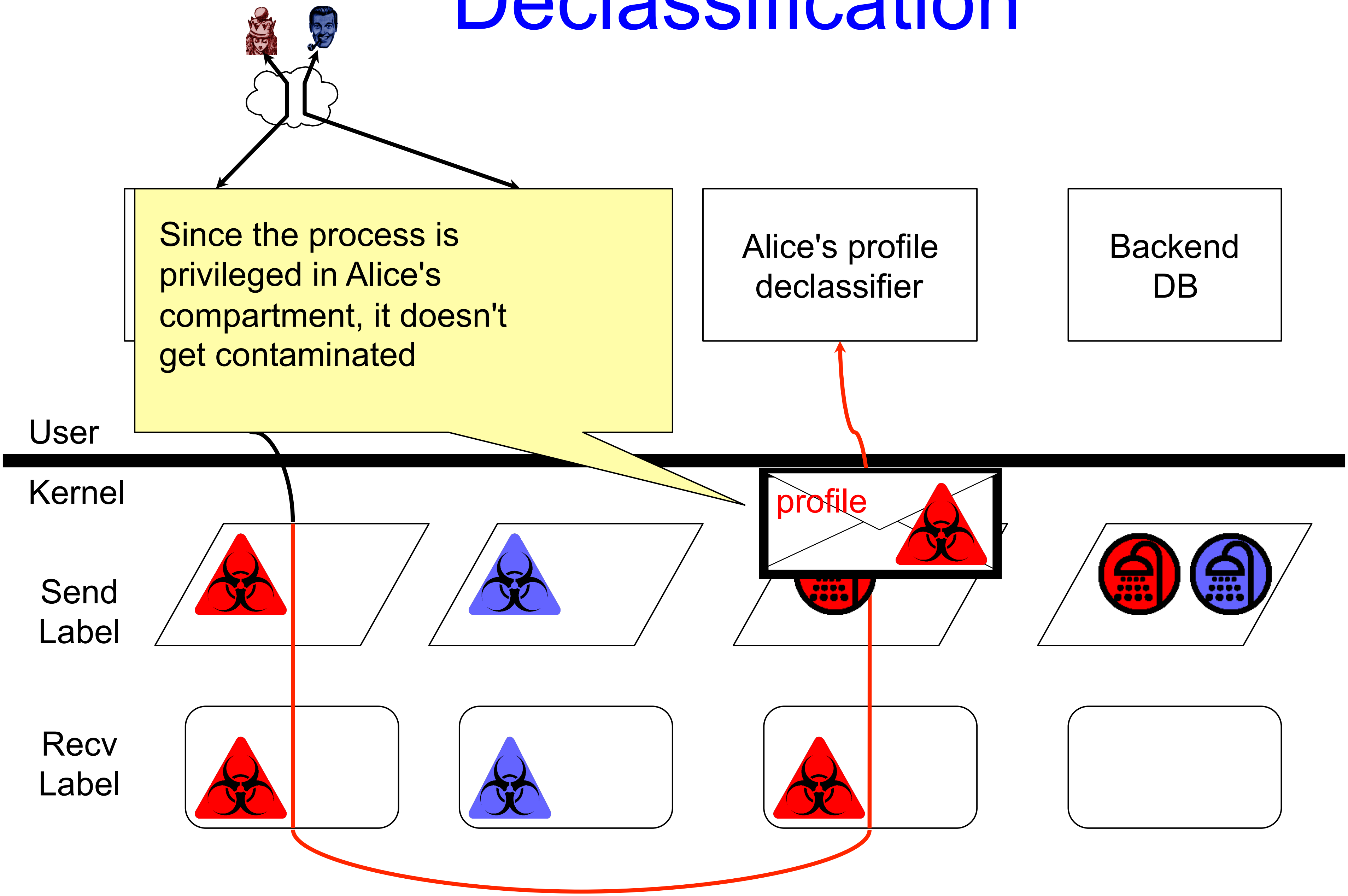
Declassification



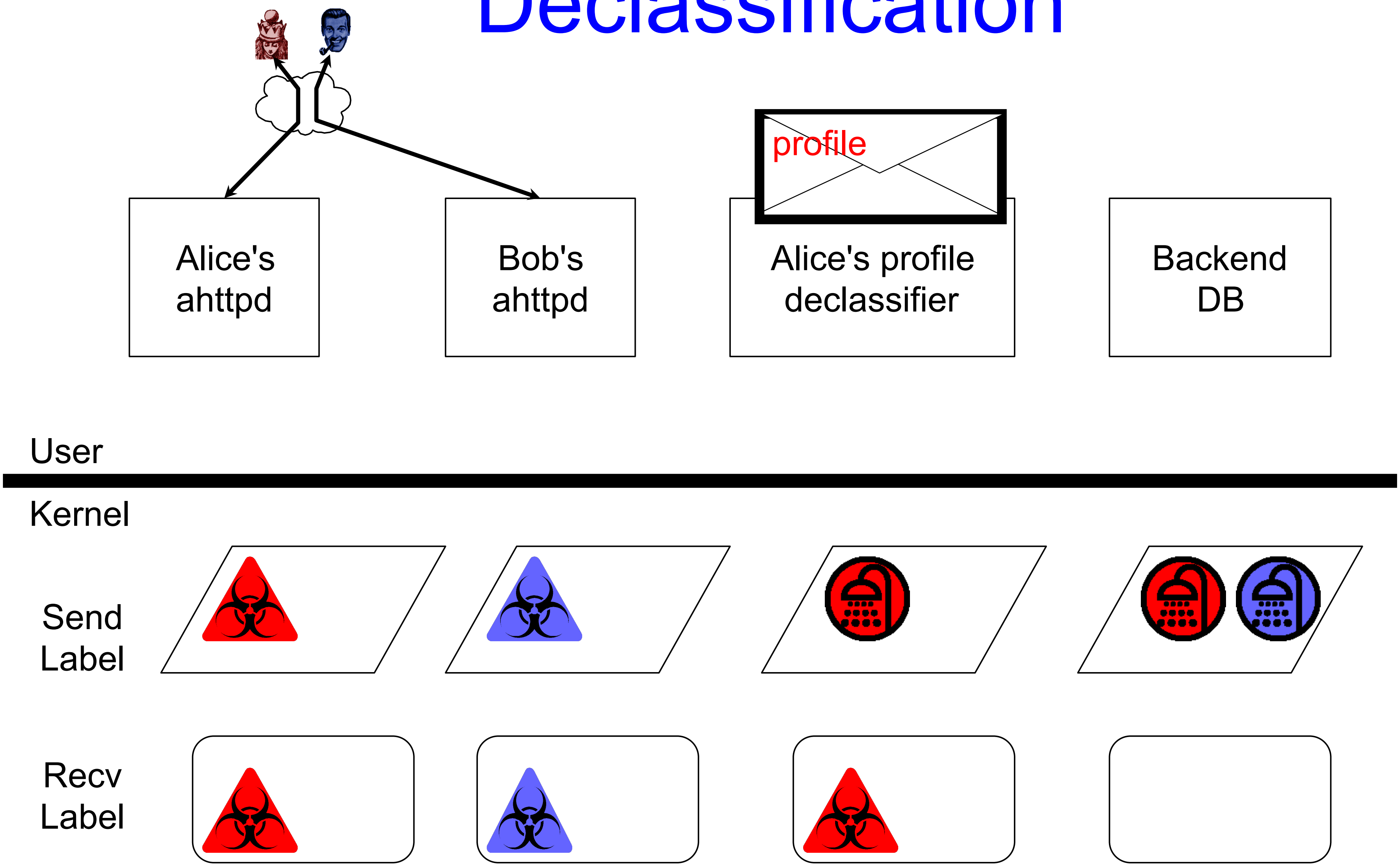
Declassification



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

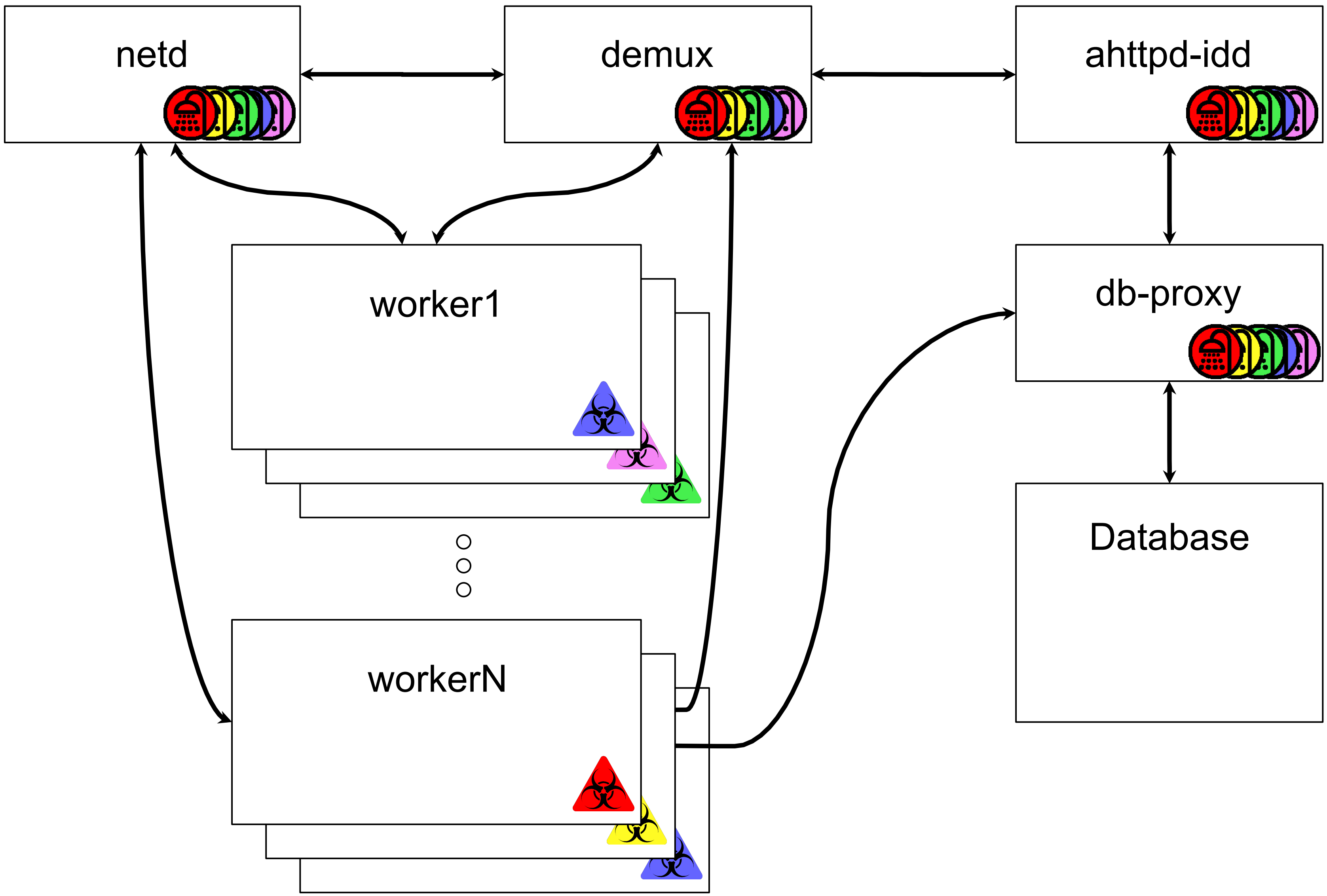
```
ep_checkpoint(&msg);
if (!state.initialized) {
    initialize_state(&state);
    state.reply = new_port();
}
process_message(&msg, &state);
ep_yield(); // revert to chkpointed memory
```

- 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

Event Processes Abstraction

- Event process isolate state
  - Used so that each event process is only contaminated by one user
  - One process per service with one event process per user
- Even at 10,000 event processes, state is stored efficiently
- Little additional programmer overhead because event processes fit into event driven programming style

Web Server Architecture



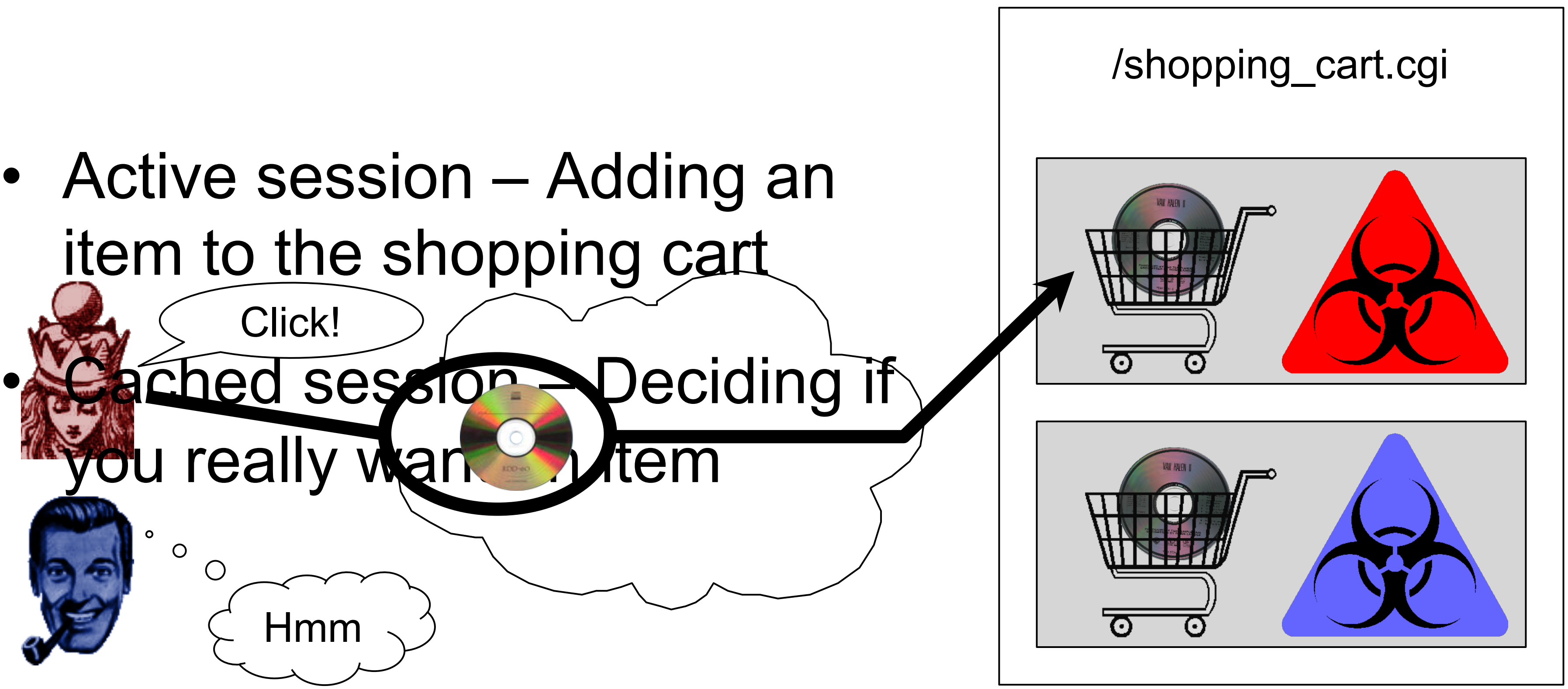
Performance Hypotheses

- Is the memory overhead from event processes mild, even at 10,000 sessions?
- Despite better security properties, is the performance of the OK web server on Asbestos comparable with Apache?
- Does the per connection kernel overhead increase at most linearly with the number of sessions?

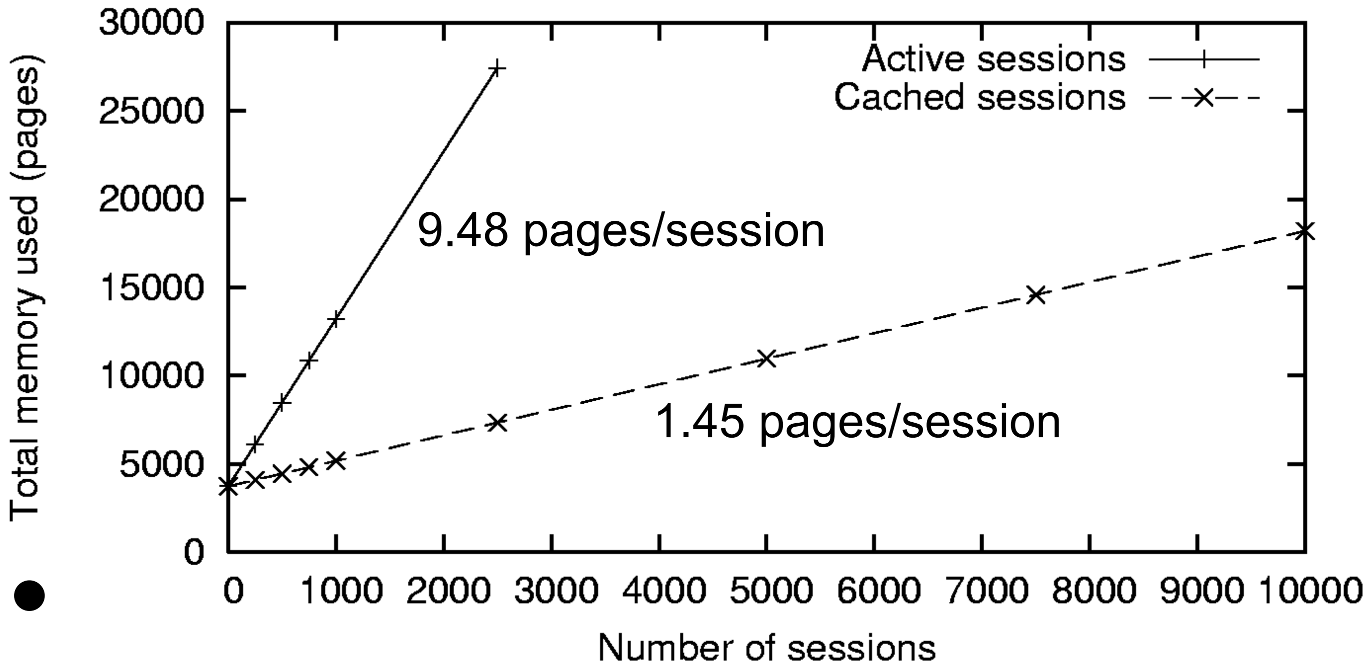
Experimental Setup – Memory

- 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
- Cached session – Deciding if you really want an item



Event Processes Conserve Memory

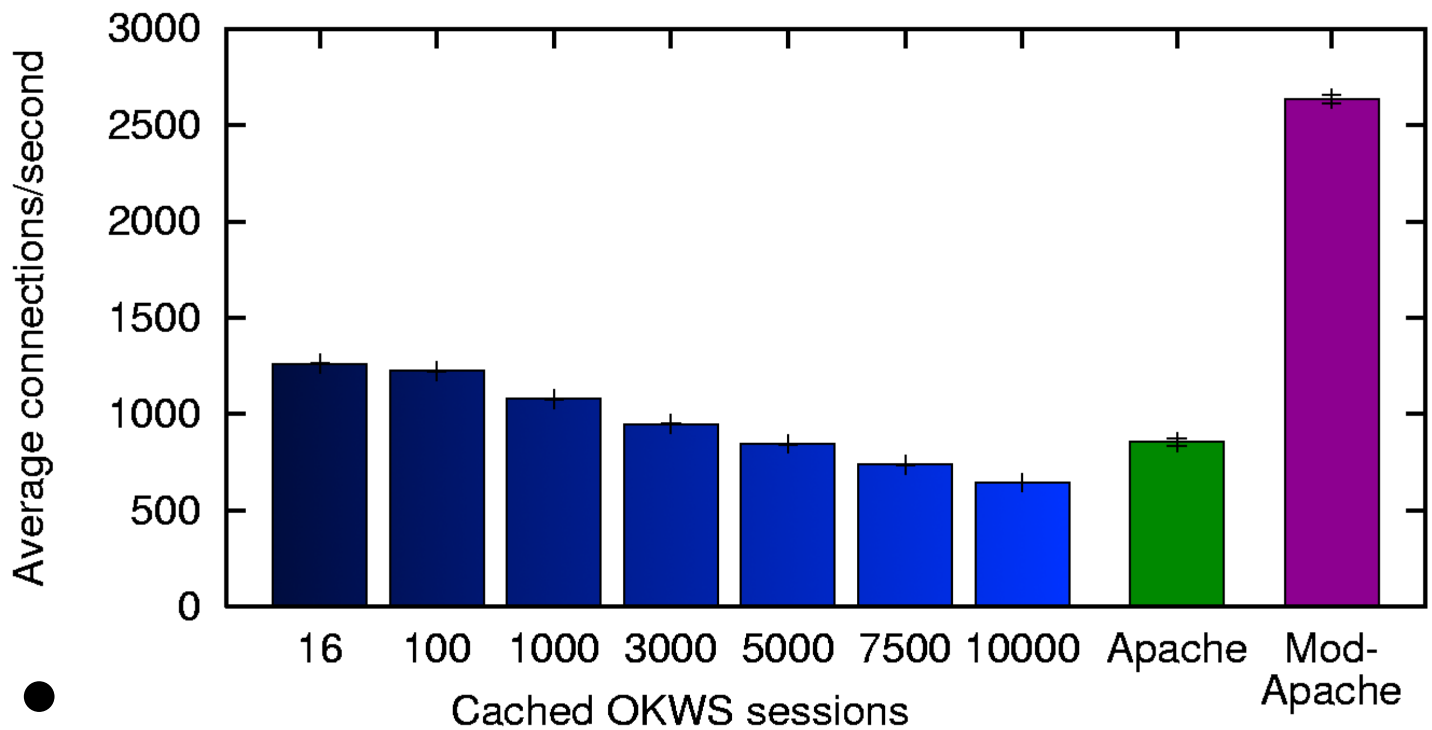


- Includes user and kernel memory
- Not too many active sessions on a large website

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
  - i.e. did *not* fork a worker 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

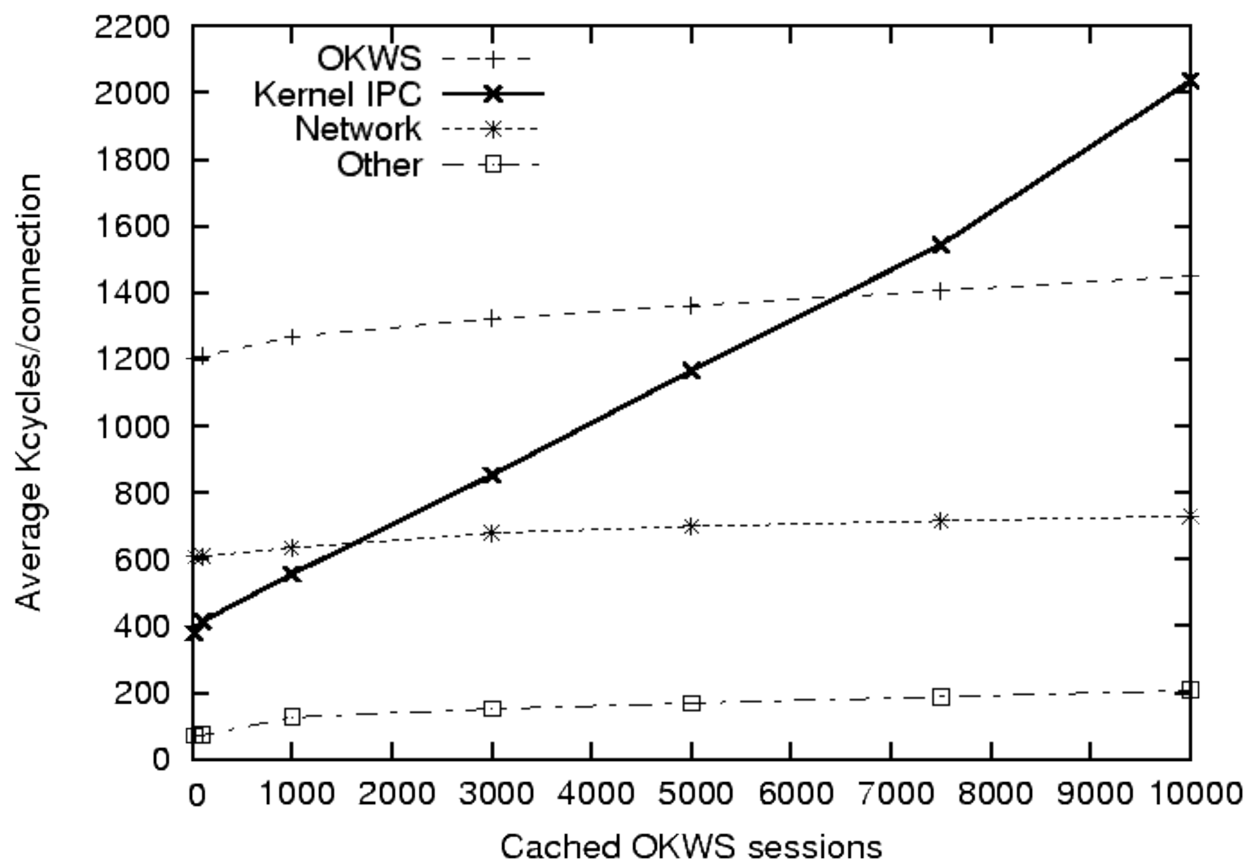
Figure 8: The median and 90th percentile latencies of requests to various server confi gurations.

Label Cost Linear in Label Size

- Throughput benchmark
- DB performance fixed

- Label cost starts small but outstrips OKWS cost around 6500 sessions

- Declassifiers label size O (#sessions)



Future Work

- Minimizing label costs
- Easing programmability
- Label persistence
- More applications

Perspective

- Asbestos labels make MAC (mandatory access control) tractable
  - 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
- Lessons/Flaws
  - Increased cached sessions decrease performance
  - Label checking scales linearly with number of labels
    - “at least not quadratic or exponential”!

Next Time

- Read and write review:
  - *Exokernel: an operating system architecture for application-level resource management*, Dawson R. Engler, M. Frans Kaashoek, and James O'Toole, Jr. 15th ACM symposium on Operating systems principles (SOSP), December 1995, pages 251—266
  - *Extensibility, Safety and Performance in the SPIN Operating System*, Brian N. Bershad, Stefan Savage, Przemyslaw Pardyak, Emin Gun Sirer, Marc E. Fiuczynski, David Becker, Craig Chambers, Susan Eggers. 15th ACM symposium on Operating systems principles (SOSP), December 1995, pages 267--283.

Next Time

- Read and write review:

- Project Proposal
  - Return comments later today

- Project Survey Paper due next Friday

- Check website for updated schedule