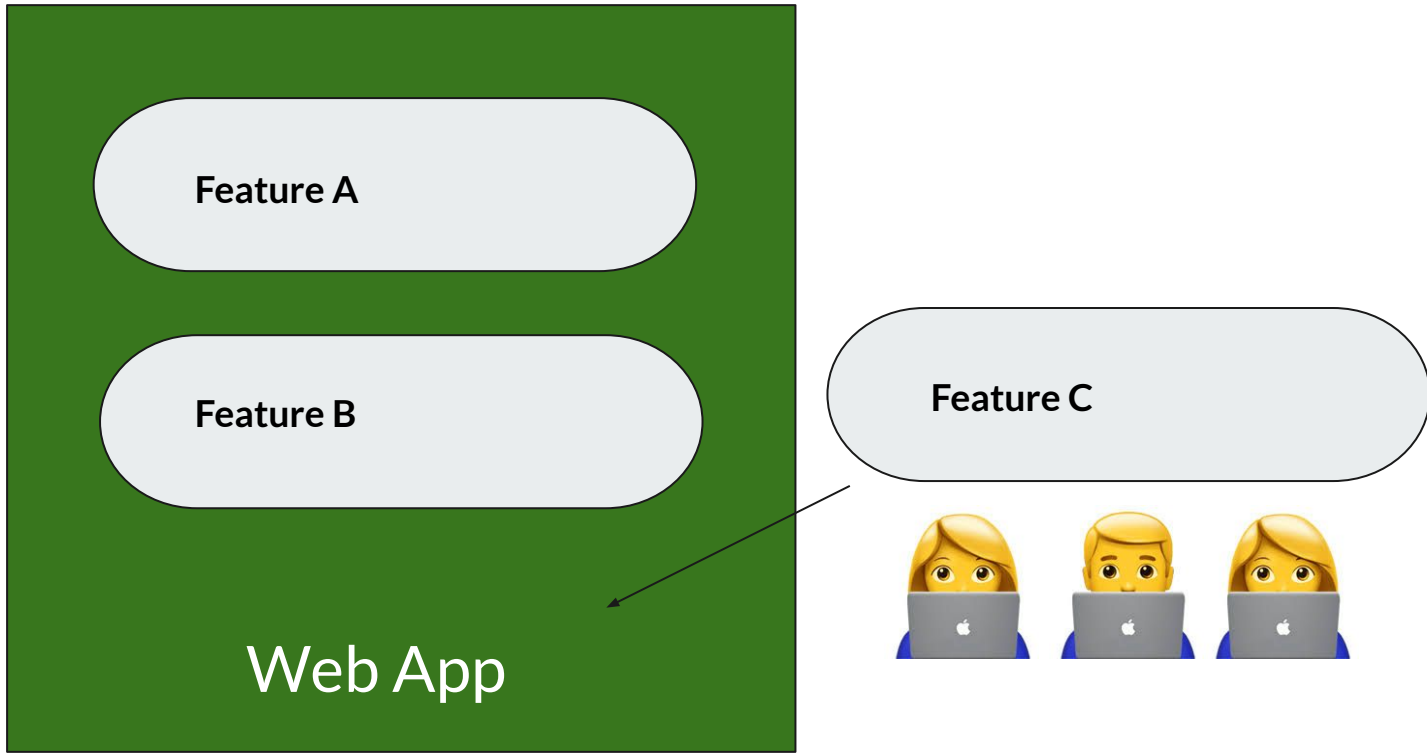
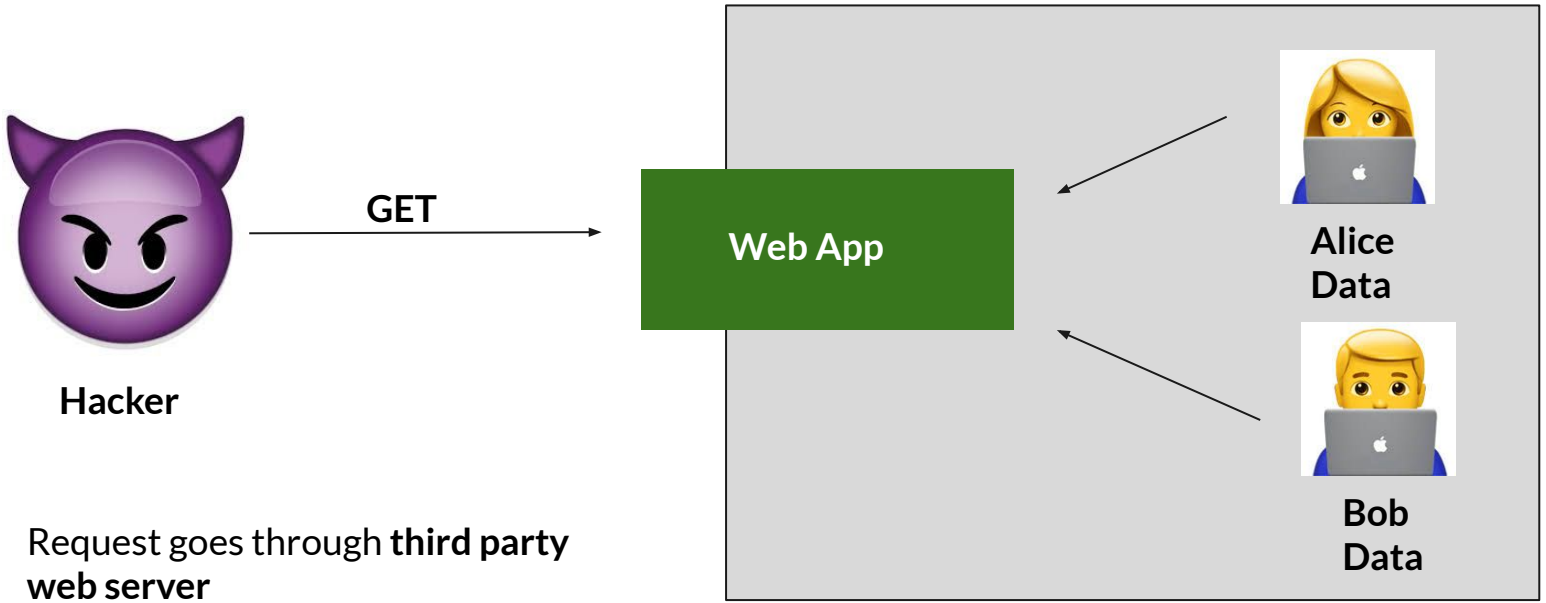
Securing Distributed Systems with Information Flow Control

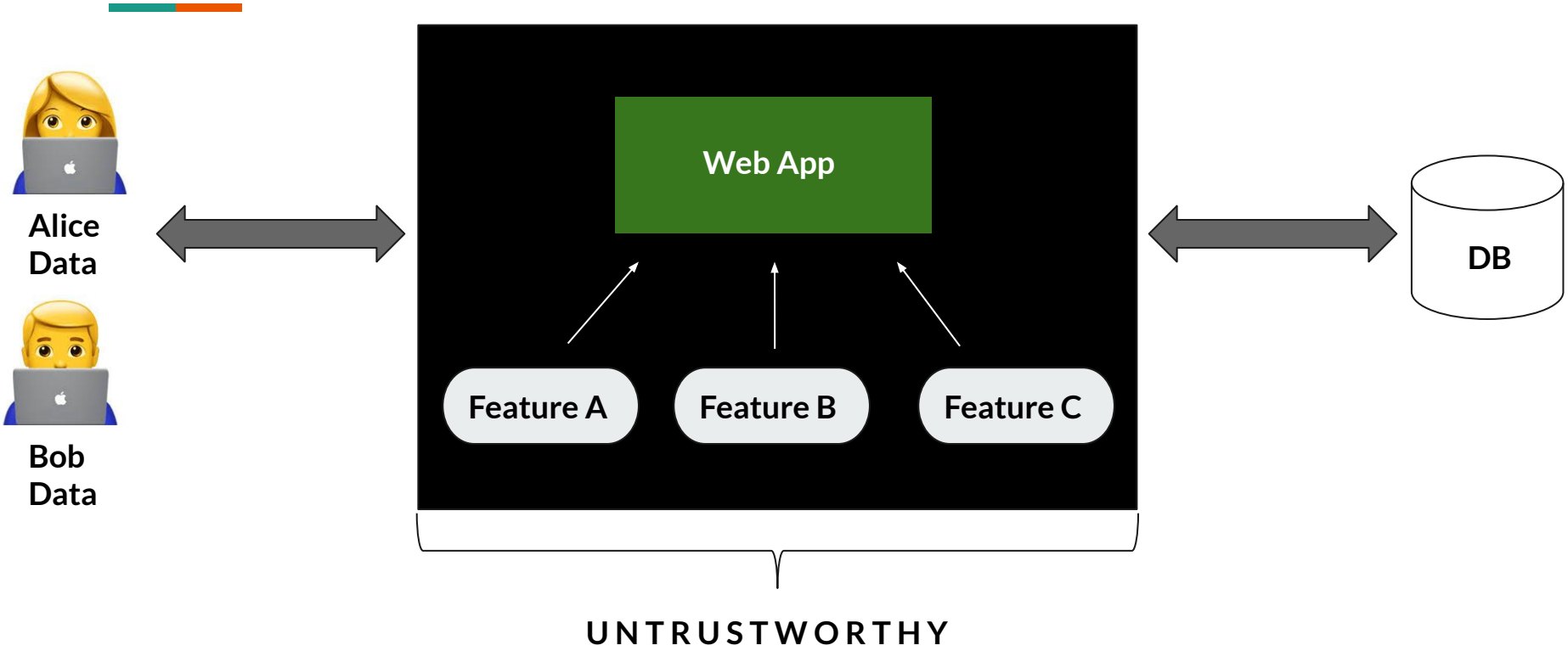
Ankita

Web Apps these days...





- Request goes through **third party web server**
- (Possibility) of server side malware
- Hacker gets client data


Alice
Data
Bob
Data
Web App
Feature A
Feature B
Feature C
DB
UNTRUSTWORTHY

Key Terms 

- IPC - Interprocess Communication
- DIFC - Decentralized Information Control
- IFC - Information Flow Control
- DIFC OSes - Asbestos, HiStar, Flume
- PayMaxx

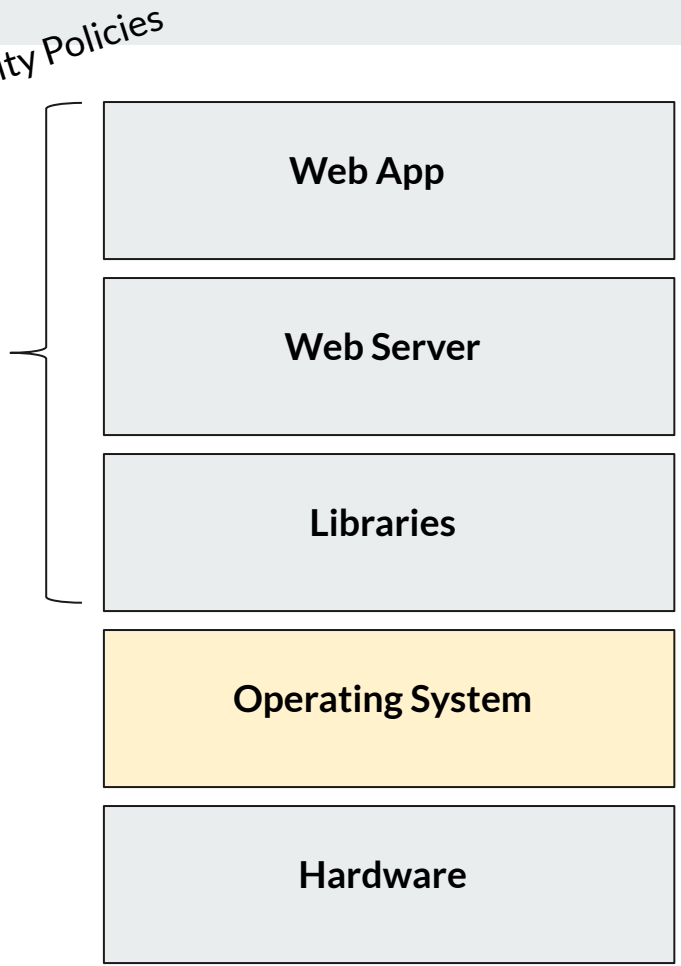
Design systems that remain secure despite untrustworthy code

If security is about code being correct, we need to fix the code **BUT** not this is not practical: code will never be perfect and bug free!

As long as Alice's data is not going to Bob and vice versa, we don't really care about what the code does.

How do we keep data secure?

- Enforce data movement
- IFC enforced at the **lowest possible component**
  - All other protection mechanisms built **on top of this**
- **Associate a label with data** 💡
  - Labels follow data when it moves around
  - Labels specify what happens to the data
    - How information flows between objects
  - MOST of the other code does not have to worry about security

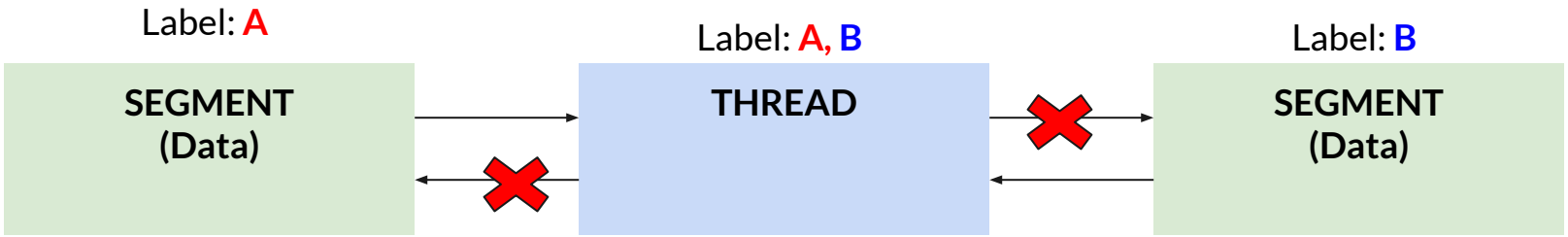


Labels

- Communication permissions
- $L_S \sqsubseteq L_R$
  - $\sqsubseteq$ = **“can flow to”**
  - Is this bidirectional?

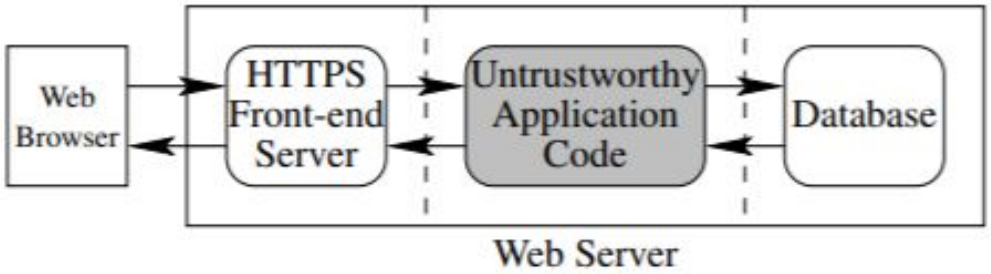
HiStar

- Kernel enforces security
  - Associate protection with data and **not** files or processes



High level: Back to PayMaxx

- PayMaxx runs buggy application code for each user to generate a tax form
- A DIFC OS would **prevent** the application from communicating with any other component



DIFC OSes Short Comings

- Works when all processes are running on the same machine
- This does not scale
- A site like PayMaxx may uses several machines for their frontend HTTPS servers
- We **need a network protocol** that supports DIFC

DStar

- Protocol and framework that leverage OS-level protection on DIFC machines
- Control how information flows between processes on different machines
- Interaction between mutually distrustful components
- Every user's tax information can be individually tracked and protected as it flows through the network

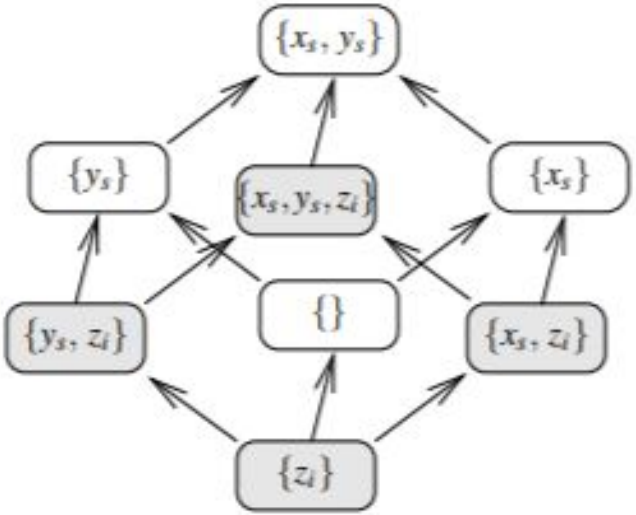
Labels in a Distributed System

- Cannot observe labels of processes on different machines
  - DStar **labels messages**
  - Enforces $\mathbf{L_S} \sqsubseteq \mathbf{L_M} \sqsubseteq \mathbf{L_R}$
  - Actually enforces $\mathbf{L_S} \sqsubseteq \mathbf{O_S}\,\mathbf{L_M} \sqsubseteq \mathbf{O_R}\,\mathbf{L_R}$
- Why should PayMaxx have a different value of L_M?

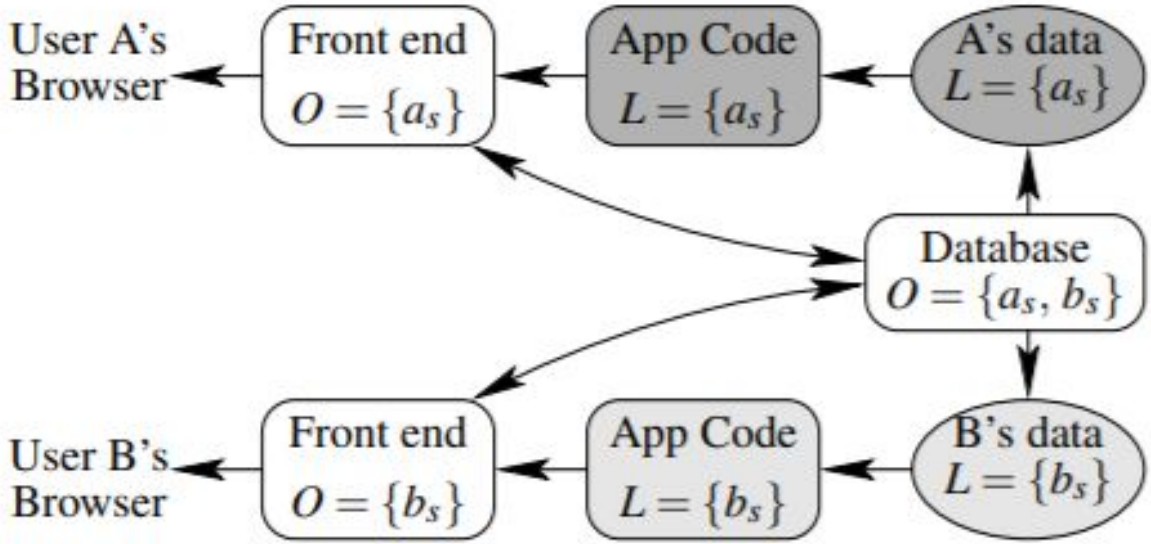
Categories

- DStar label is a **set of categories**
- There are **two types of categories**
  - Secrecy
  - Integrity
- Any process can allocate a category, gain ownership of it, grant ownership to another process
- What category should the PayMaxx database server have for every user?

$L_1 \sqsubseteq L_2$ iff L1 contains all the integrity categories in L_2 and L_2 contains all the secrecy categories in L_1



PayMaxx + Categories + Labels

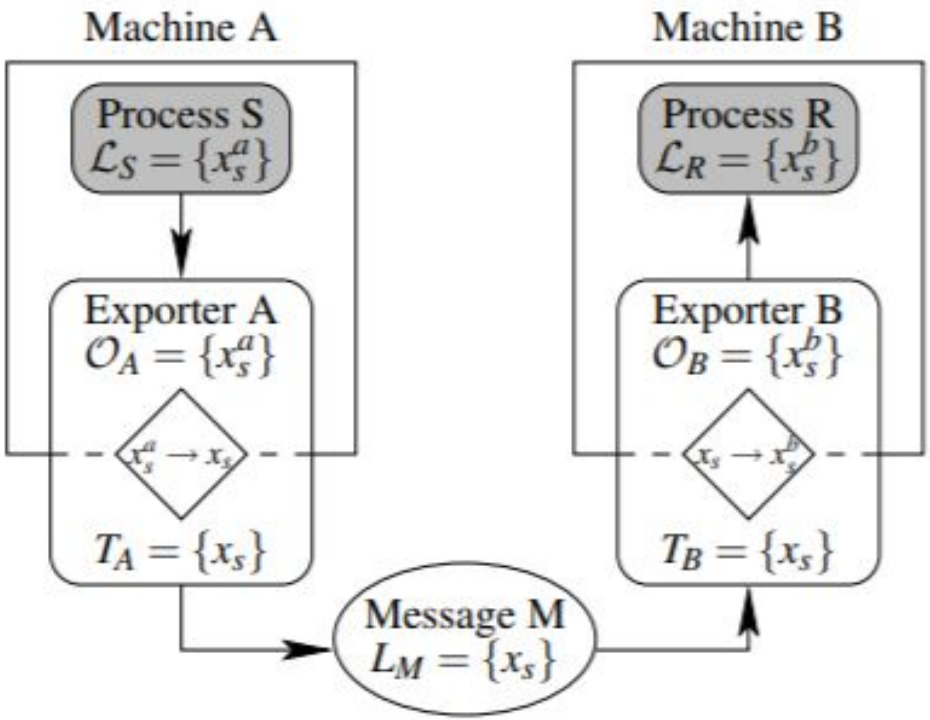


DStar Exporter

Allow processes that lack direct network access to communicate across machines

Only process that sends and receives DStar messages over the network

Enforces flow restrictions implied by message labels



Exporter Properties

Delegation certificates: determine when an exporter with a particular public key can receive a message with a particular message label

- A process is only as trustworthy as its exporter
  - Process P creates a category C, adds C to its local exporter's trust set
- How does DStar use self-certifying category names?

```
struct category_name {
    pubkey creator;
    category_type type;
    uint64_t id;
};
```

- How does one exporter verify the membership of a category in a remote exporter's trust set with **no external communication**?

```
struct dstar_message {
    pubkey recipient_exporter;
    slot recipient_slot;
    category_set label, ownership, clearance;
    cert_set certs;
    mapping_set mapset;
    opaque payload;
};
```

Info flow restrictions of the message

Delegation certificates proving to recipient exporter that sender is trusted with all categories in ownership and clearance

Mappings for recipient exporter to map DStar categories to OS categories

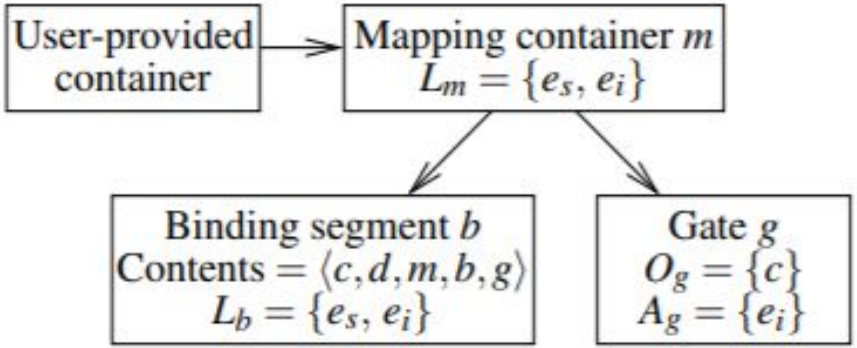
Exporter Properties Contd.

- How do all exporters know other exporter’s network addresses and public keys?

HiStar Exporter

- Containers
  - Provide resources like storage and CPU time
- Threads
  - Any thread can allocate a category
- Gates
  - IPC

Exporters need ownership over the local HiStar Category



Single machine communicating over DStar

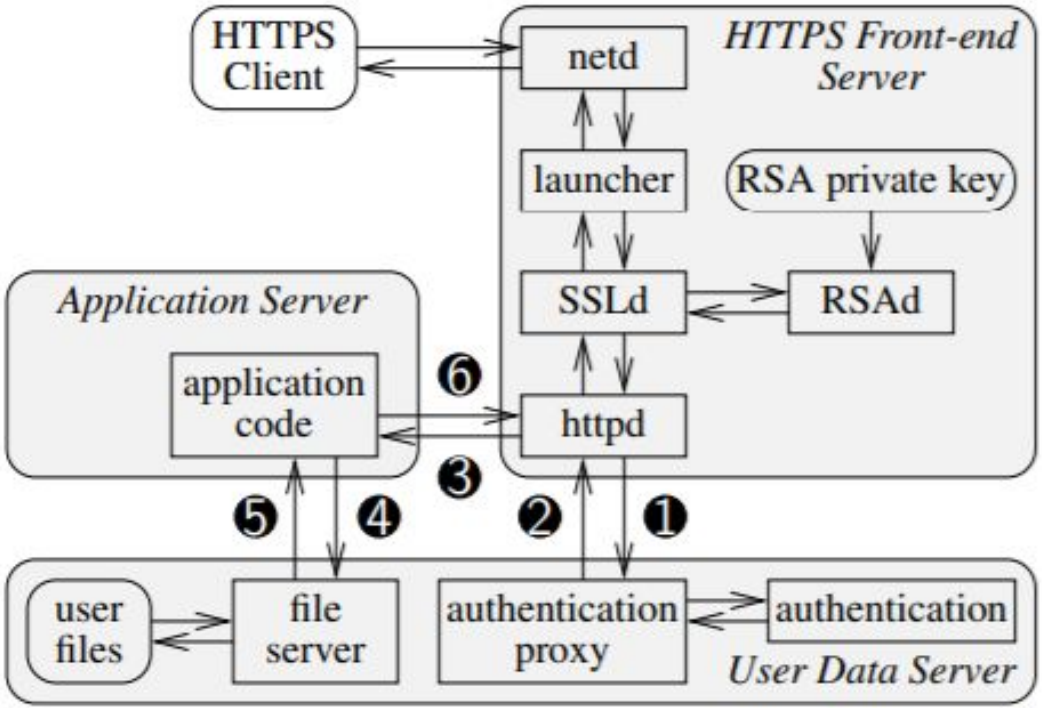
(1) a mapping from the local category to a DStar category

(2) certificates proving that the remote exporter is trusted by that DStar category

(3) mapping from the DStar category to a local category on the remote machine

HiStar + DStar - 3 tiered web application

- Communicates with one data server per user
- For a web service that generates tax forms, allow multiple companies to provide their own data server. Each company can trust to web service to generate their tax form, but the company does not trust anyone other than themselves with employee data.
    - If servers handling tax data were compromised, credit card data would not necessarily be exposed
    - Better to NOT lose everything with this isolation


HTTPS Client
netd
HTTPS Front-end Server
launcher
RSA private key
SSLd
RSAd
Application Server
application code
httpd
6
3
5
4
2
1
user files
file server
authentication proxy
authentication
User Data Server

What Does the App Need to Do?

- Must explicitly define trust between different machines in the distributed system - creating and distributing appropriate delegation certificates
- Applications need to explicitly allocate resources such as containers and category mappings on different machines to communicate/execute code remotely
- Provide memory and CPU resources for all messages and processes

Adding a new physical machine to DStar cluster

- Requires the manual transfer of category names and public keys
  - Like we do in SSH

Calling machine	Linux	HiStar	Linux	Linux	Linux
Execution machine	same	same	Linux	HiStar	Flume
Communication	none	none	TCP	DStar	DStar
Throughput, req/sec	505	334	160	67	61
Latency, msec	2.0	3.0	6.3	15.7	20.6

Figure 13: Throughput and latency of executing a "Hello world" perl script in different configurations.

- DStar pros and cons
- Network mechanisms to improve app security
- Web browser security mechanisms
- What are some future directions for building secure systems?