CS 114: Network Security

Lecture 18 - Firewalls and Intrusion Detection Systems

Prof. Daniel Votipka
Spring 2023

(some slides courtesy of Prof. Micah Sherr and Patrick McDaniel)

Plan for today

- Administrivia
- Anonymity Review
- Network Defense
  - Firewalls
  - Intrusion Detection Systems
- Honeypots/Malware Analysis

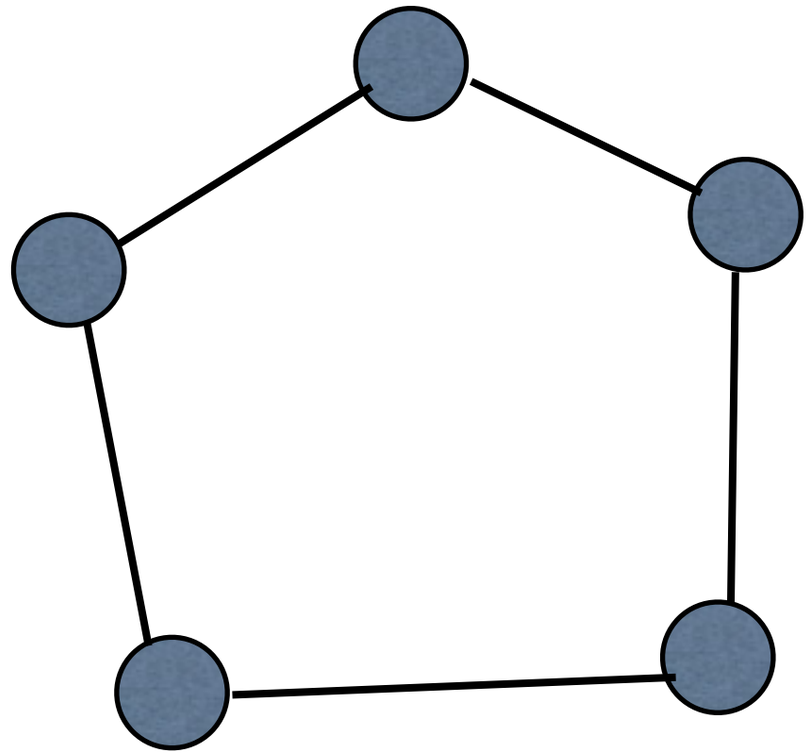
Administrivia

- Homework 1, part 3 is due tonight
- Exam 2 is next Thursday
  - Authentication -> Anonymity
  - Same format as last time

Anonymity Review

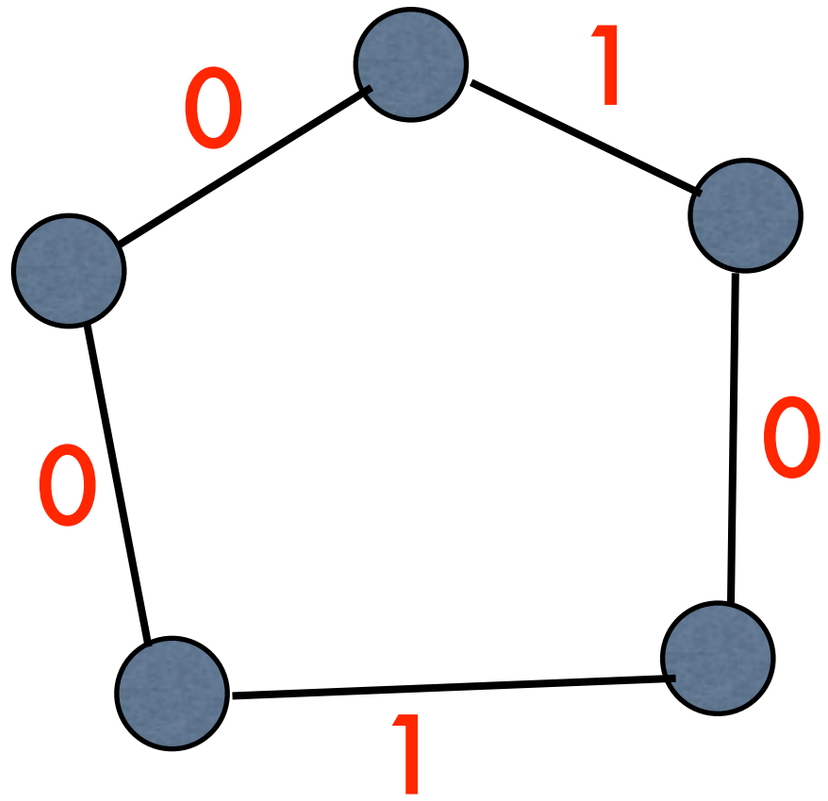
DC-Net

- Phase I: Each diner exchanges secret coin flip with neighbor
- Phase II:
  - If diner didn't pay, announces xor of local coin flips
  - If diner did pay, announces inverse of xor
- If xor of the announced xors is 0, then no one inverted and NSA paid; otherwise, a diner paid.



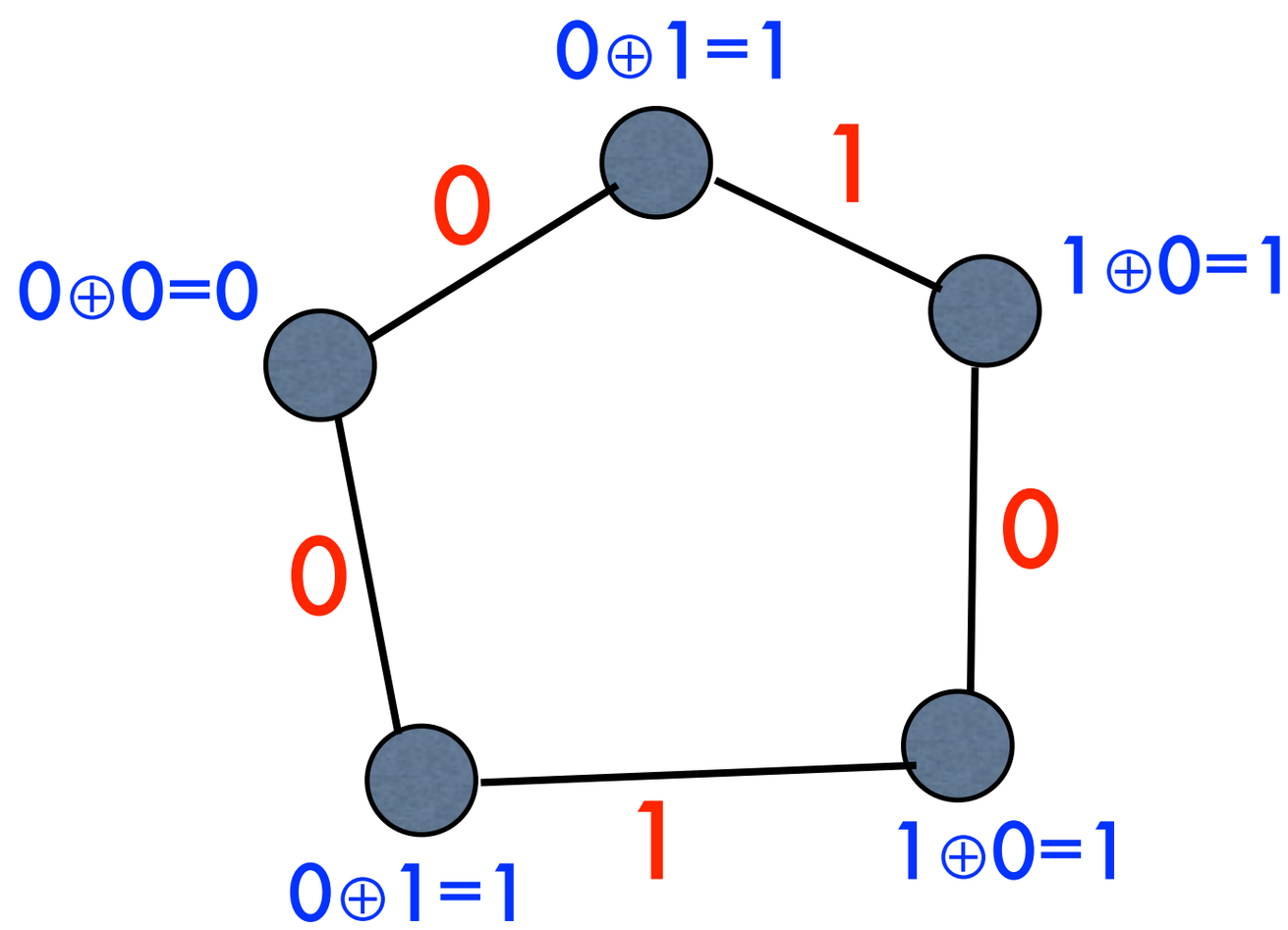
DC-Net

- Phase I: Each diner exchanges secret coin flip with neighbor
- Phase II:
  - If diner didn't pay, announces xor of local coin flips
  - If diner did pay, announces inverse of xor
- If xor of the announced xors is 0, then no one inverted and NSA paid; otherwise, a diner paid.



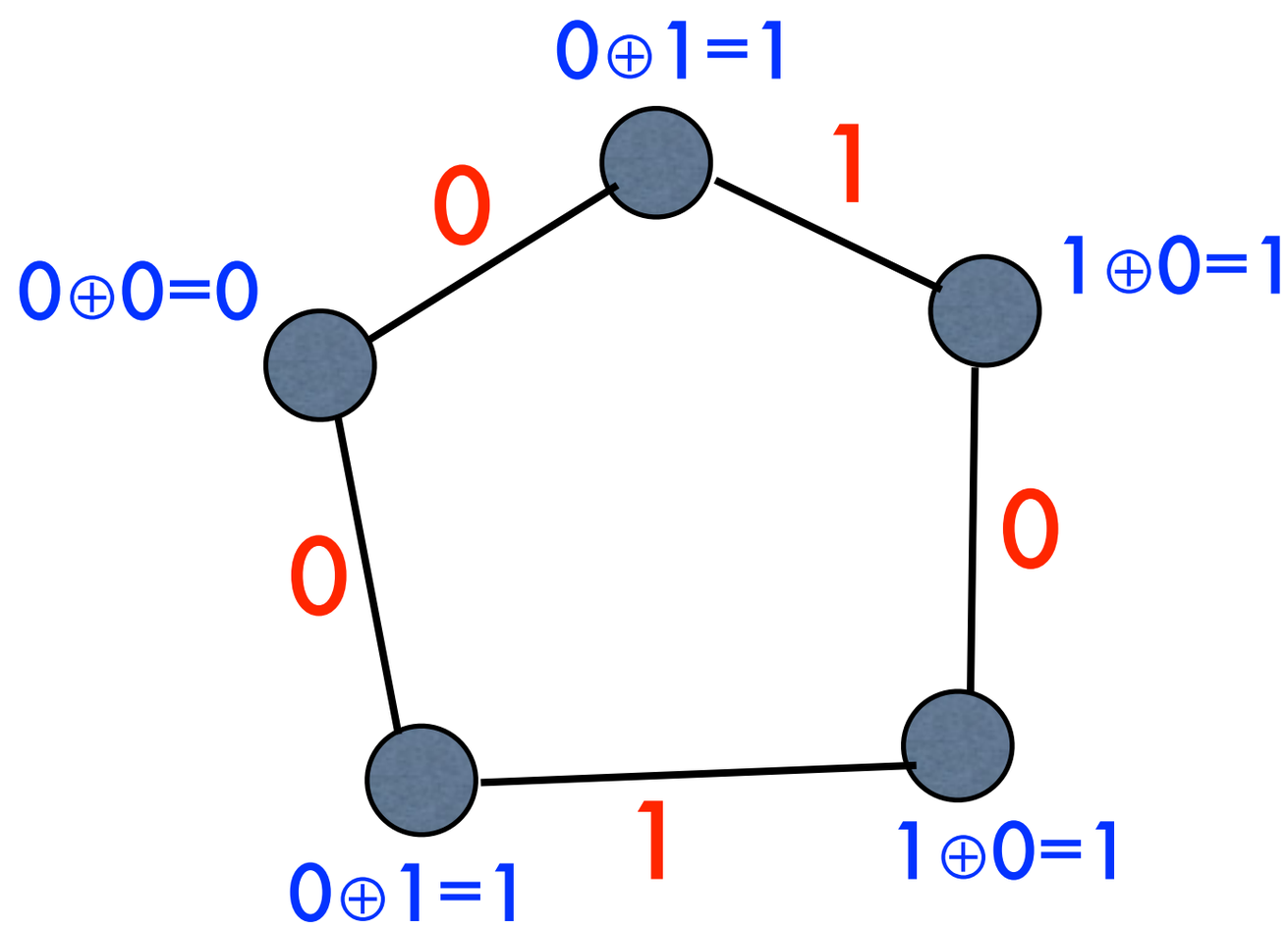
DC-Net

- Phase I: Each diner exchanges secret coin flip with neighbor
- Phase II:
  - If diner didn't pay, announces xor of local coin flips
  - If diner did pay, announces inverse of xor
- If xor of the announced xors is 0, then no one inverted and NSA paid; otherwise, a diner paid.



DC-Net

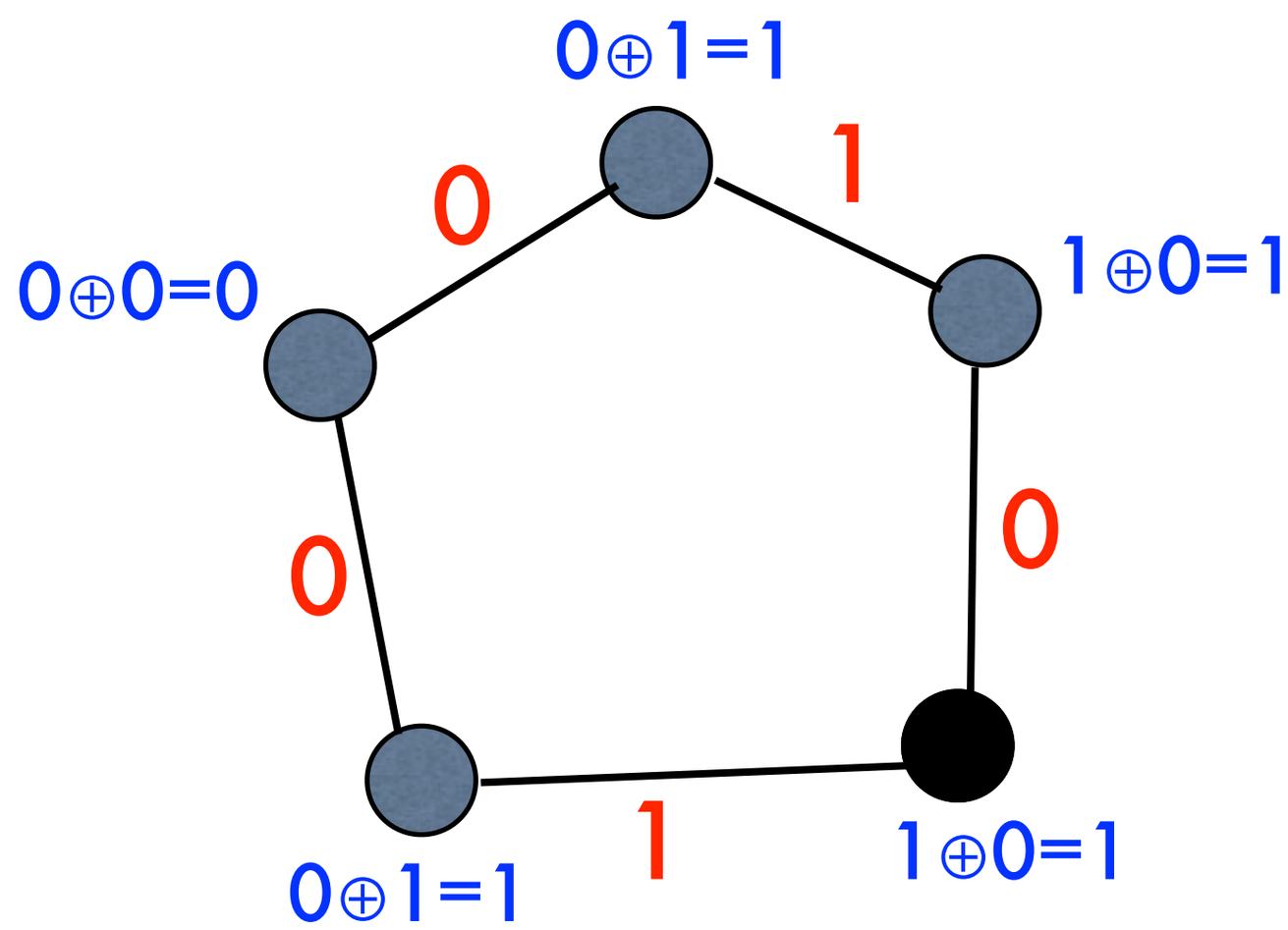
- Phase I: Each diner exchanges secret coin flip with neighbor
- Phase II:
  - If diner didn't pay, announces xor of local coin flips
  - If diner did pay, announces inverse of xor
- If xor of the announced xors is 0, then no one inverted and NSA paid; otherwise, a diner paid.



$$0 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 0$$

DC-Net

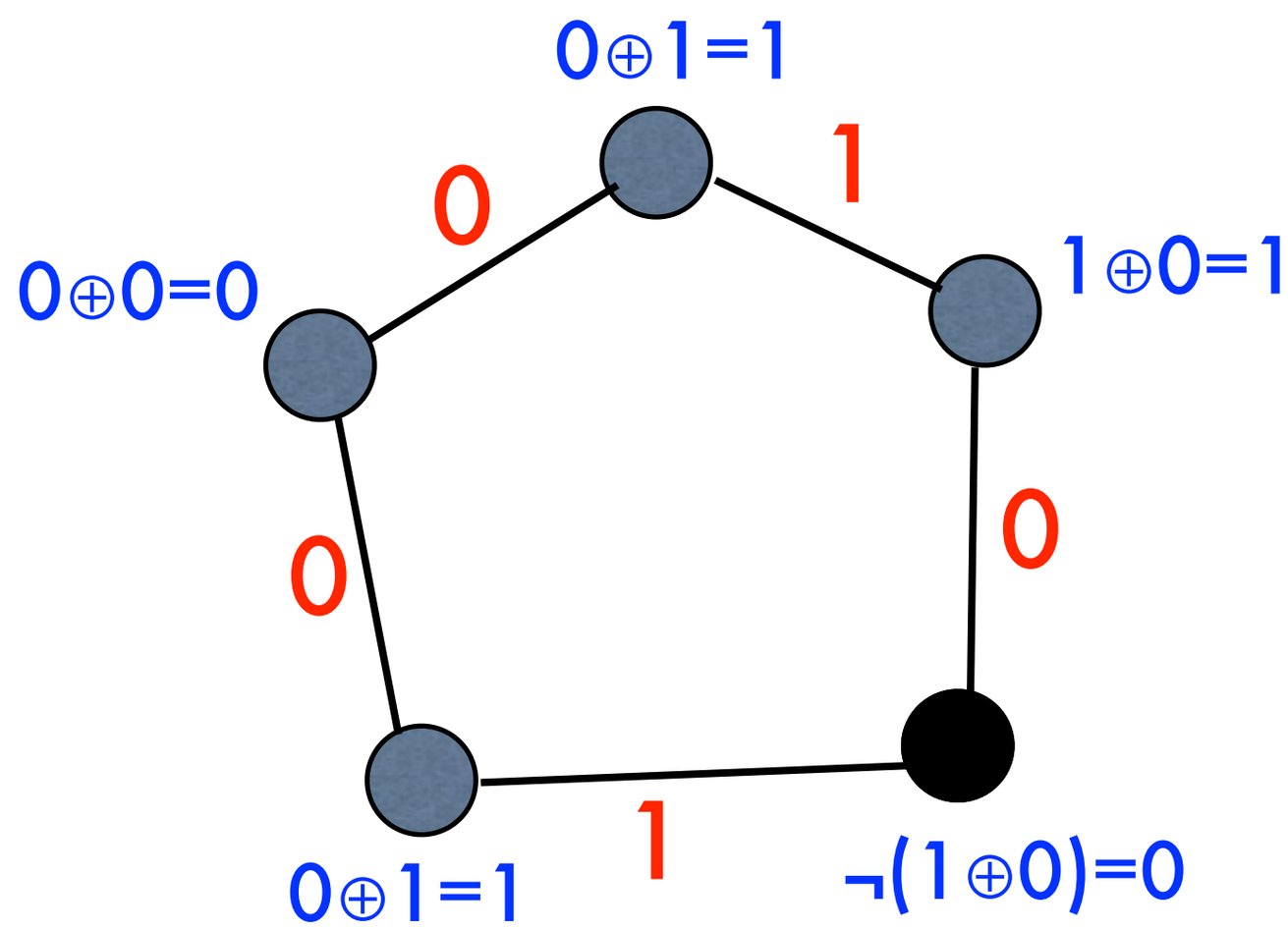
- Phase I: Each diner exchanges secret coin flip with neighbor
- Phase II:
  - If diner didn't pay, announces xor of local coin flips
  - If diner did pay, announces inverse of xor
- If xor of the announced xors is 0, then no one inverted and NSA paid; otherwise, a diner paid.



$$0 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 0$$

DC-Net

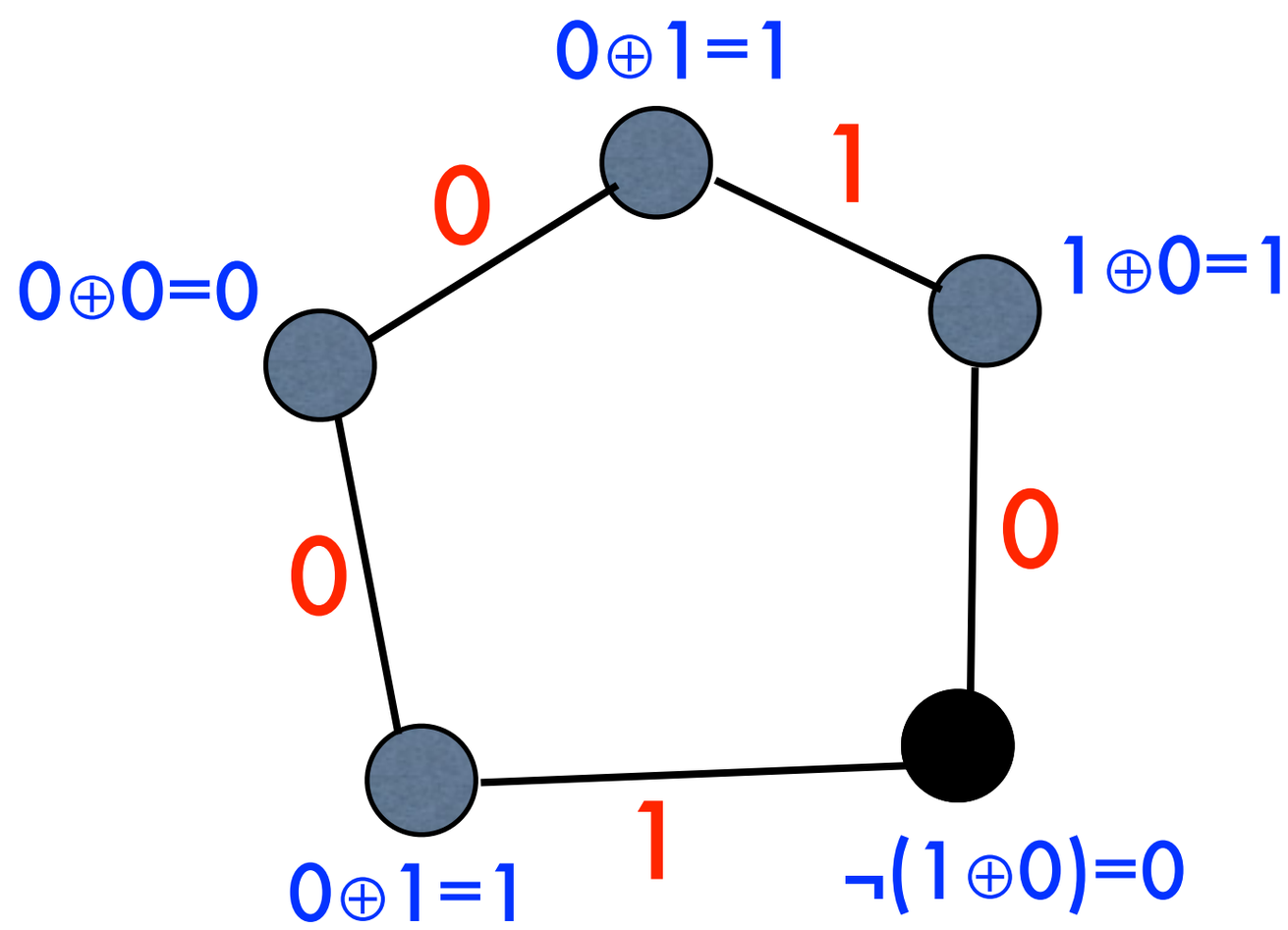
- Phase I: Each diner exchanges secret coin flip with neighbor
- Phase II:
  - If diner didn't pay, announces xor of local coin flips
  - If diner did pay, announces inverse of xor
- If xor of the announced xors is 0, then no one inverted and NSA paid; otherwise, a diner paid.



$$0 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 0$$

DC-Net

- Phase I: Each diner exchanges secret coin flip with neighbor
- Phase II:
  - If diner didn't pay, announces xor of local coin flips
  - If diner did pay, announces inverse of xor
- If xor of the announced xors is 0, then no one inverted and NSA paid; otherwise, a diner paid.



$$0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 = 1$$

DC-Nets

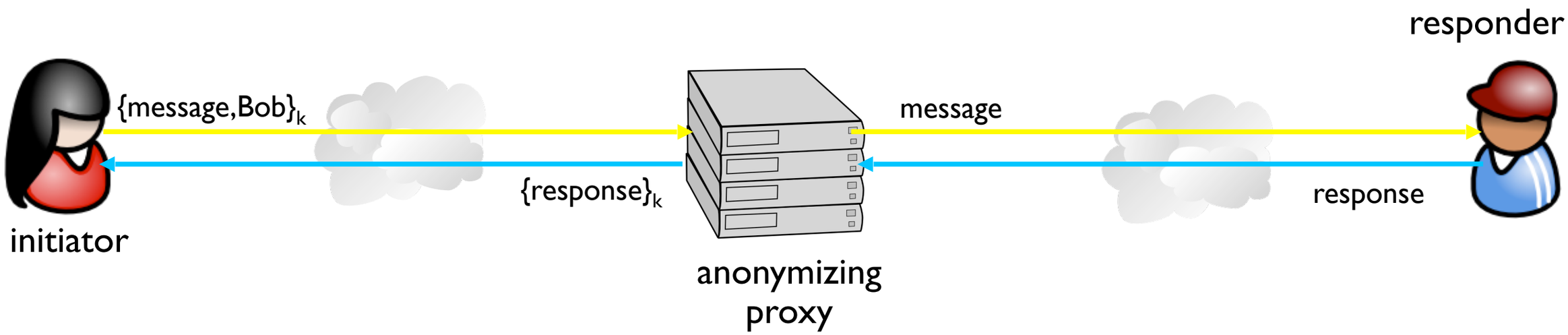
- Achieves information-theoretic anonymity (under certain conditions)
- Limitations:
  - Subject to collisions (what if two diners pay?)
  - Requires pairwise secret keys
  - Last diner who announces message gets to choose the result

DC-Nets

- Achieves information-theoretic anonymity (under certain conditions)
- Limitations: https://dedis.cs.yale.edu/dissent/
  - Subject to collisions (what if two diners pay?)
  - Requires pairwise secret keys
  - Last diner who announces message gets to choose the result

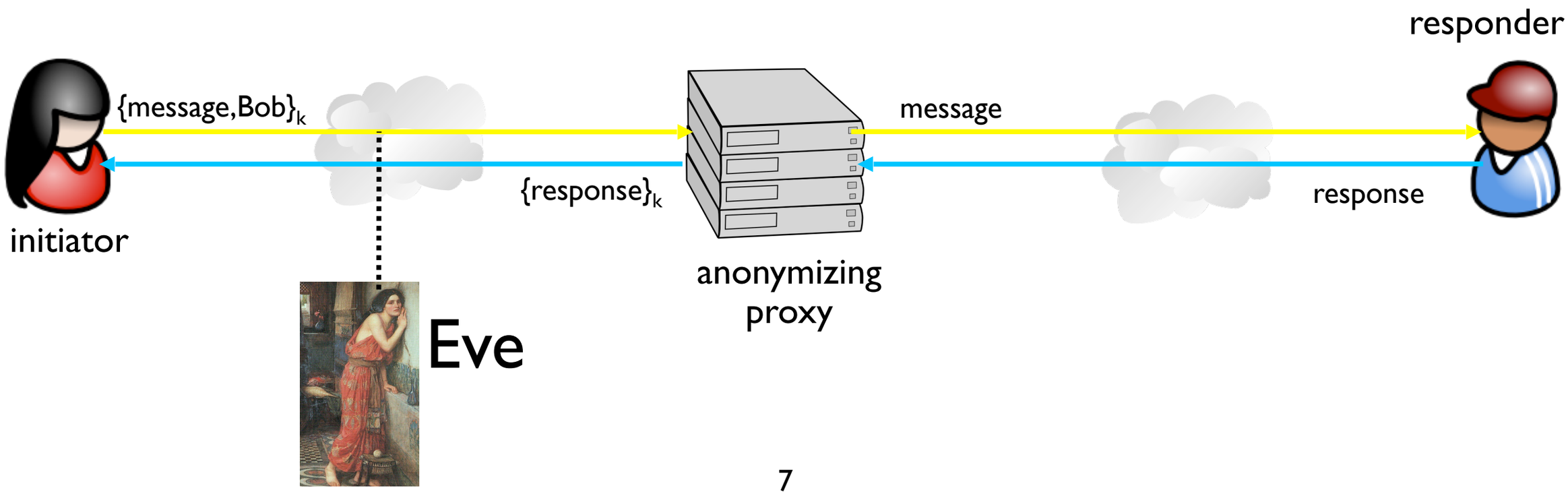
Anonymizing proxies

If eavesdroppers collude, Eve can correlate ingress and egress proxy traffic to identify Alice and Bob



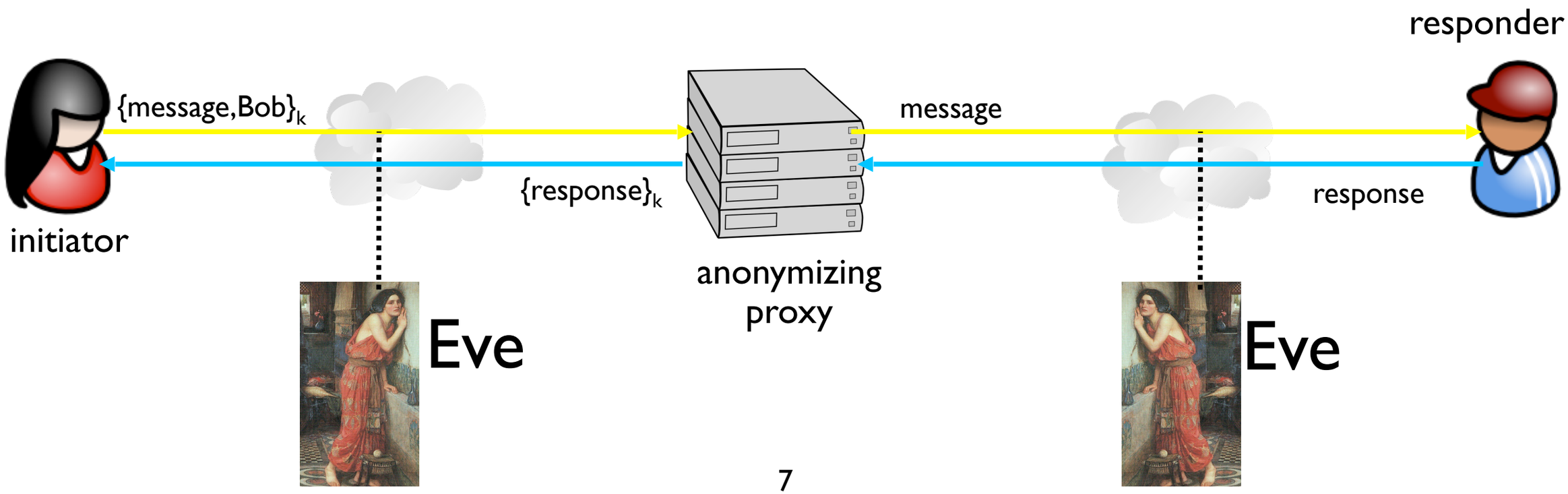
Anonymizing proxies

If eavesdroppers collude, Eve can correlate ingress and egress proxy traffic to identify Alice and Bob



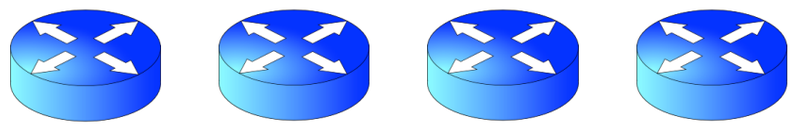
Anonymizing proxies

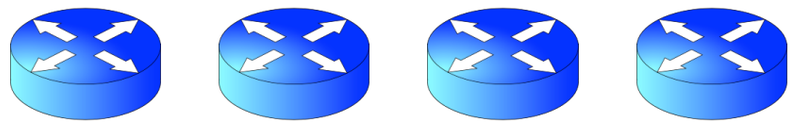
If eavesdroppers collude, Eve can correlate ingress and egress proxy traffic to identify Alice and Bob



Crowds

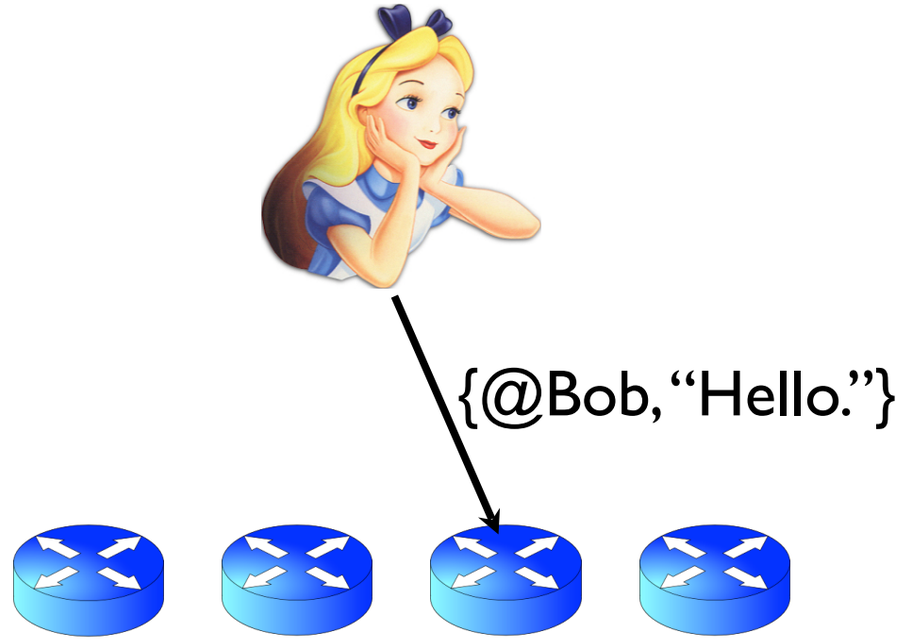
- Algorithm:
  - Relay message to random jondo
  - With probability p, jondo forwards message to another jondo
  - With probability 1-p, jondo delivers message to its intended destination

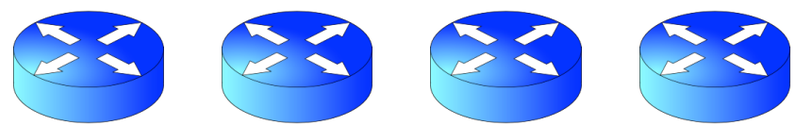




Crowds

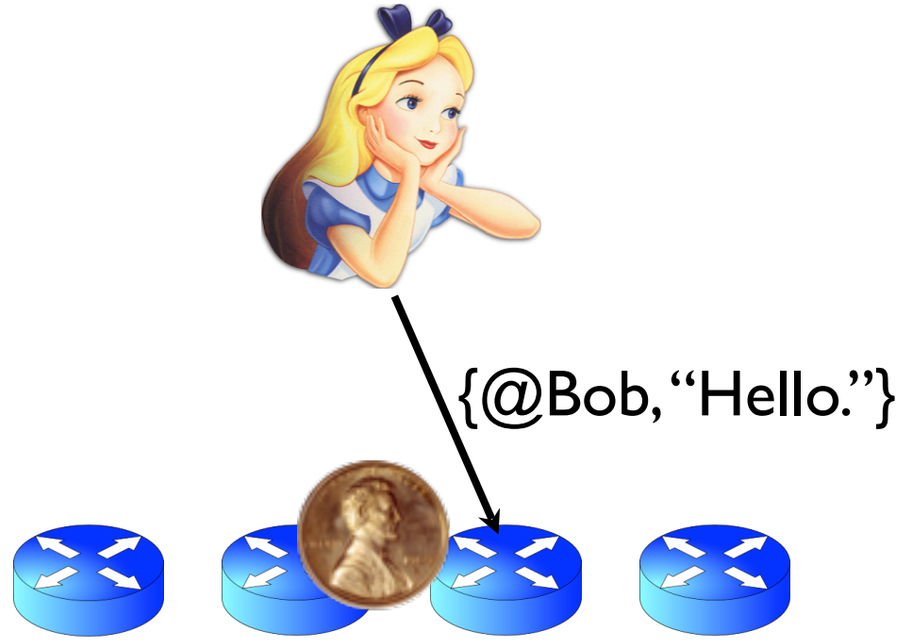
- Algorithm:
  - Relay message to random jondo
  - With probability p, jondo forwards message to another jondo
  - With probability 1-p, jondo delivers message to its intended destination

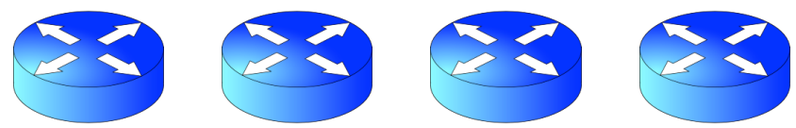




Crowds

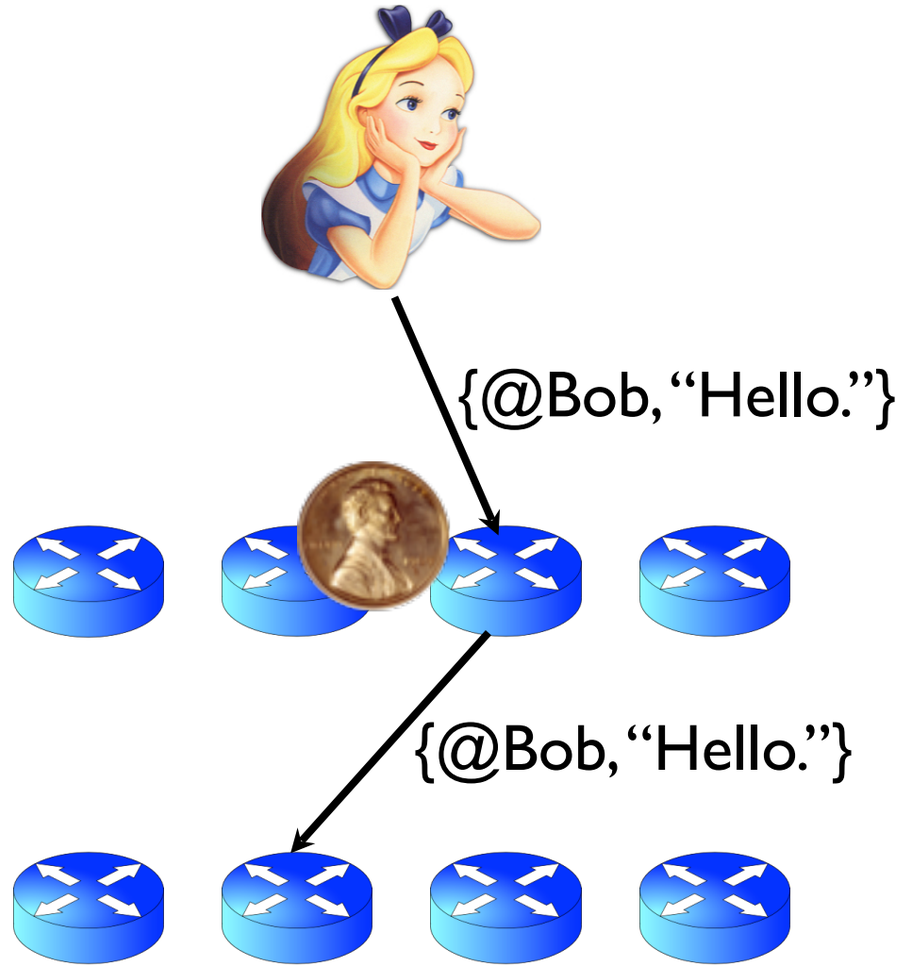
- Algorithm:
  - Relay message to random jondo
  - With probability p, jondo forwards message to another jondo
  - With probability 1-p, jondo delivers message to its intended destination





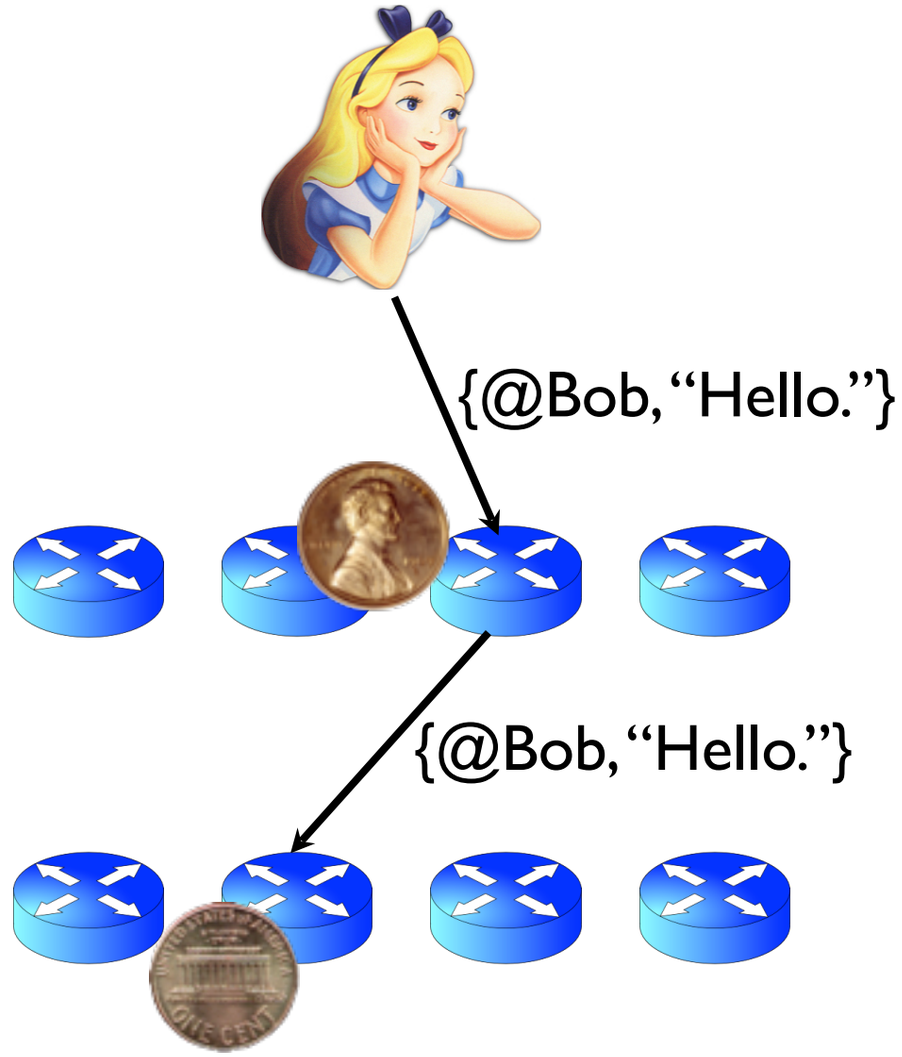
Crowds

- Algorithm:
  - Relay message to random jondo
  - With probability p, jondo forwards message to another jondo
  - With probability 1-p, jondo delivers message to its intended destination



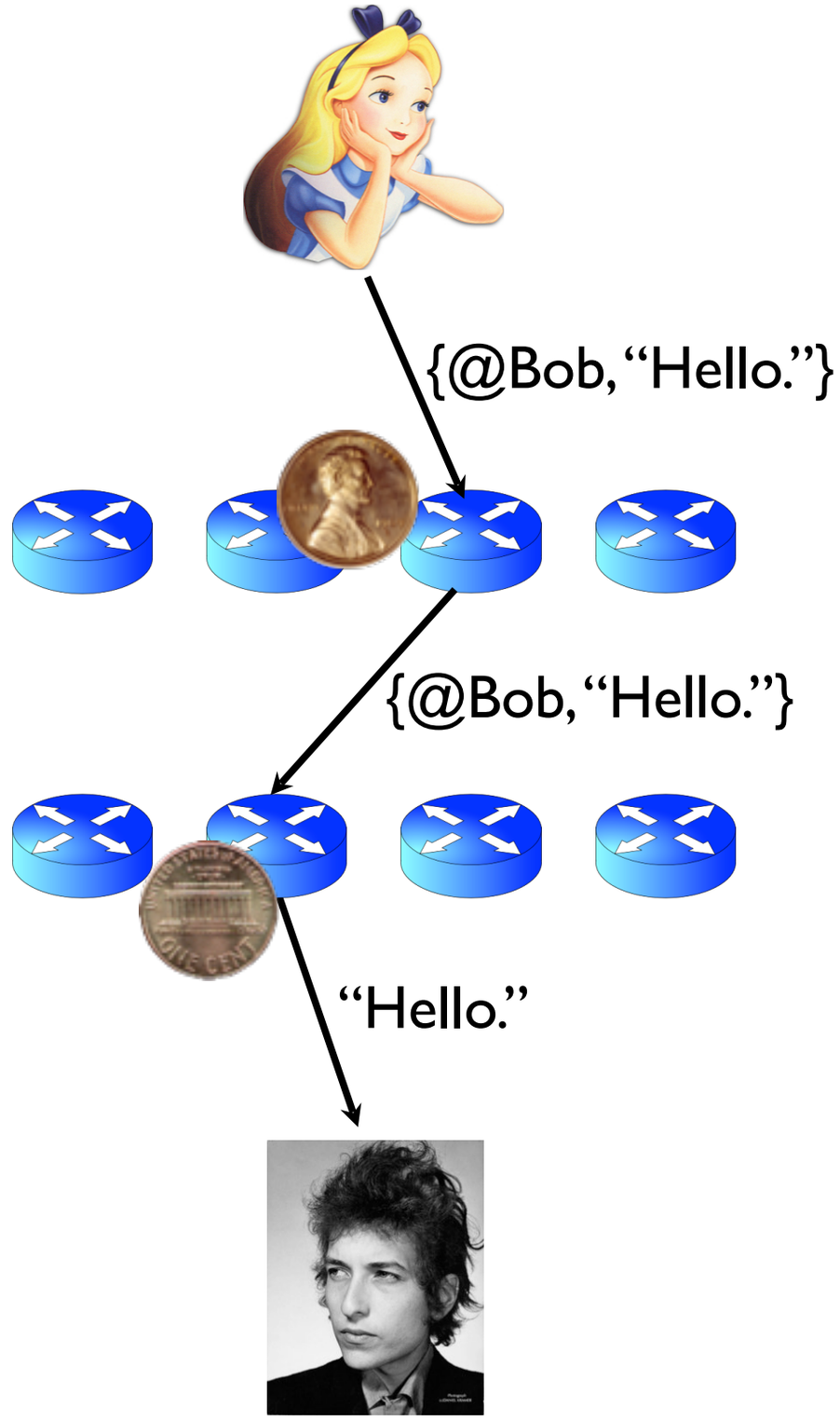
Crowds

- Algorithm:
  - Relay message to random jondo
  - With probability p, jondo forwards message to another jondo
  - With probability 1-p, jondo delivers message to its intended destination

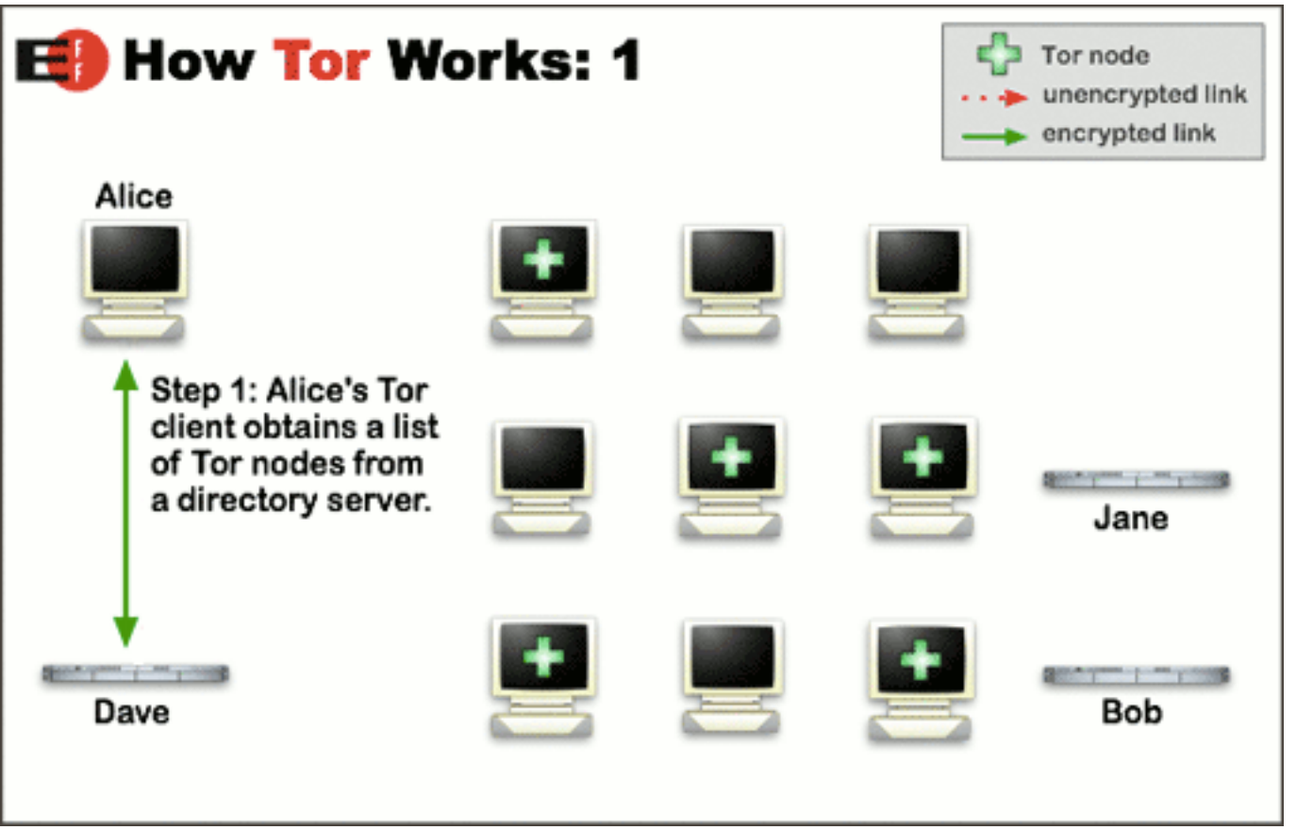


Crowds

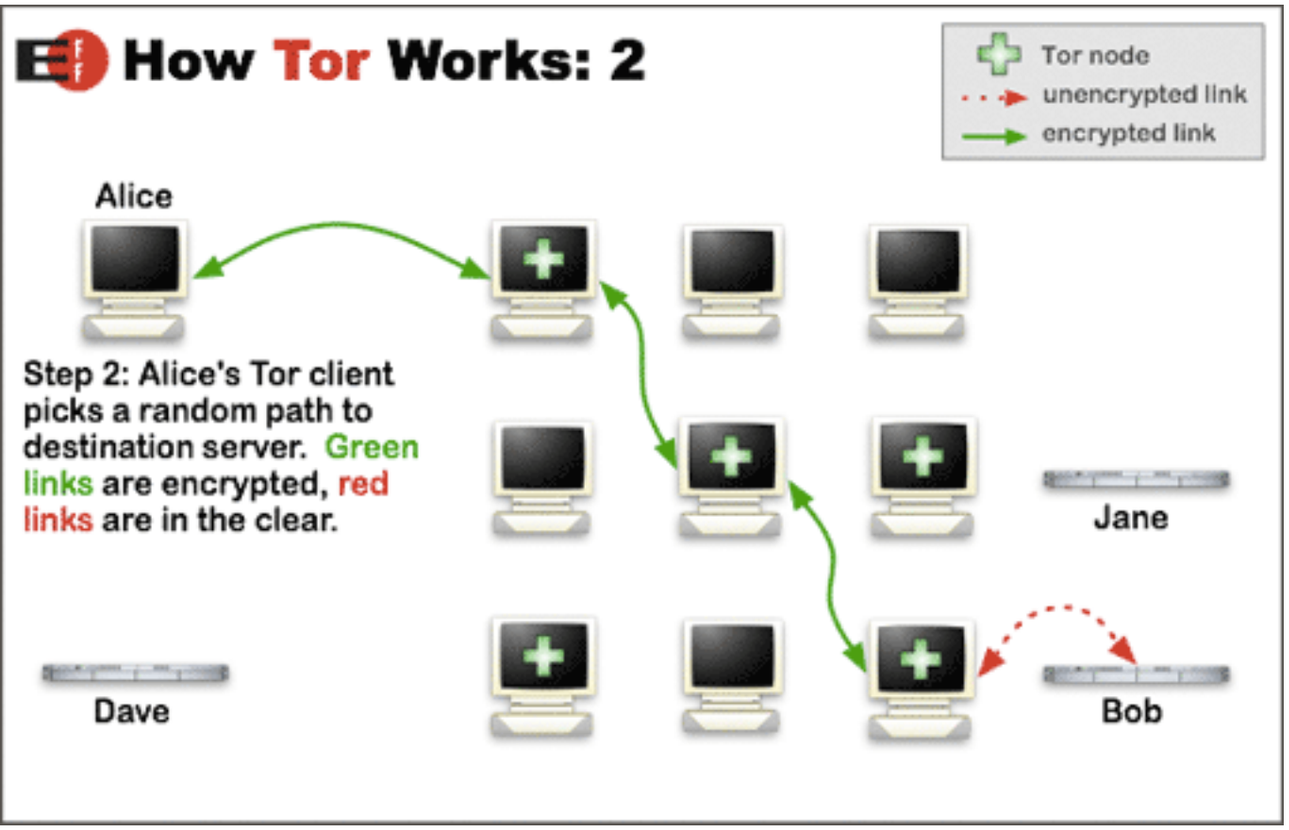
- Algorithm:
  - Relay message to random jondo
  - With probability p, jondo forwards message to another jondo
  - With probability 1-p, jondo delivers message to its intended destination



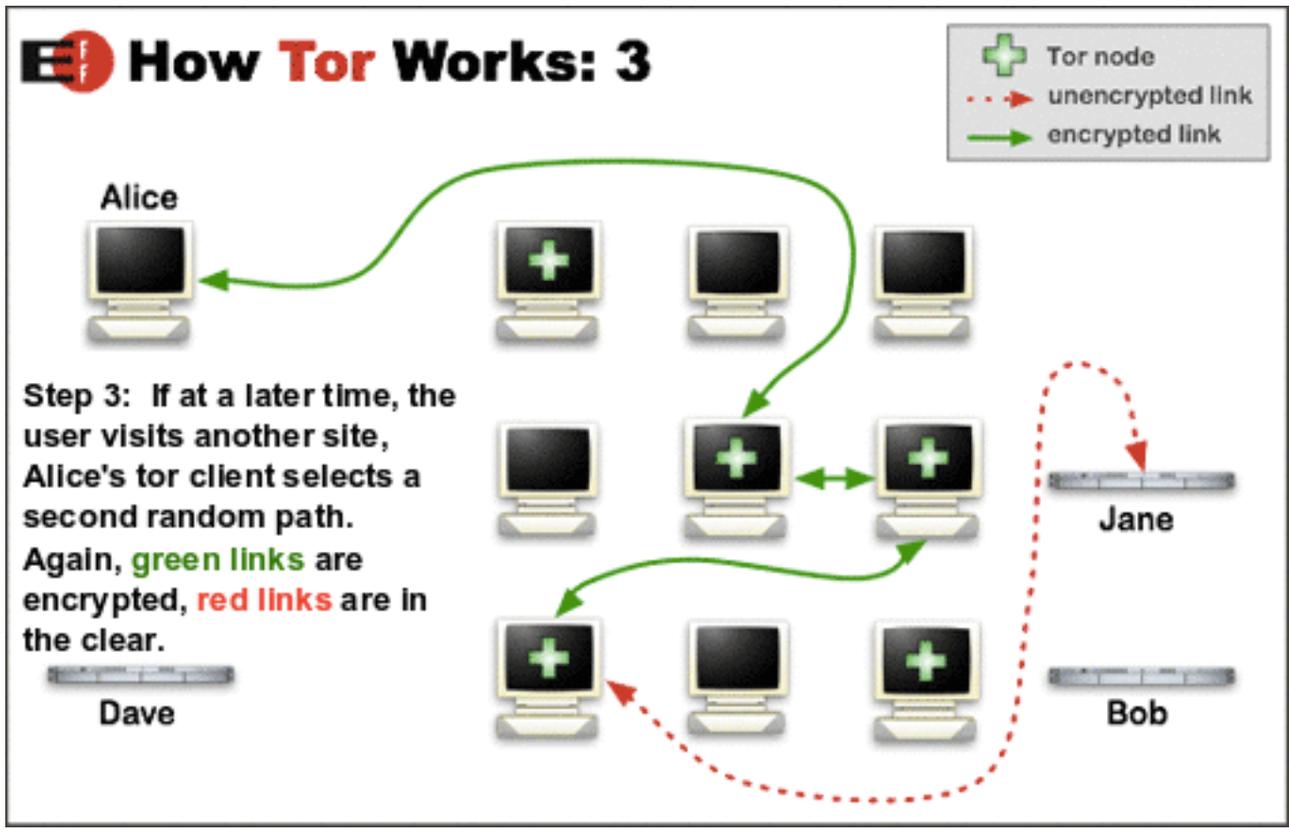
How Tor Works: 1



How Tor Works: 2

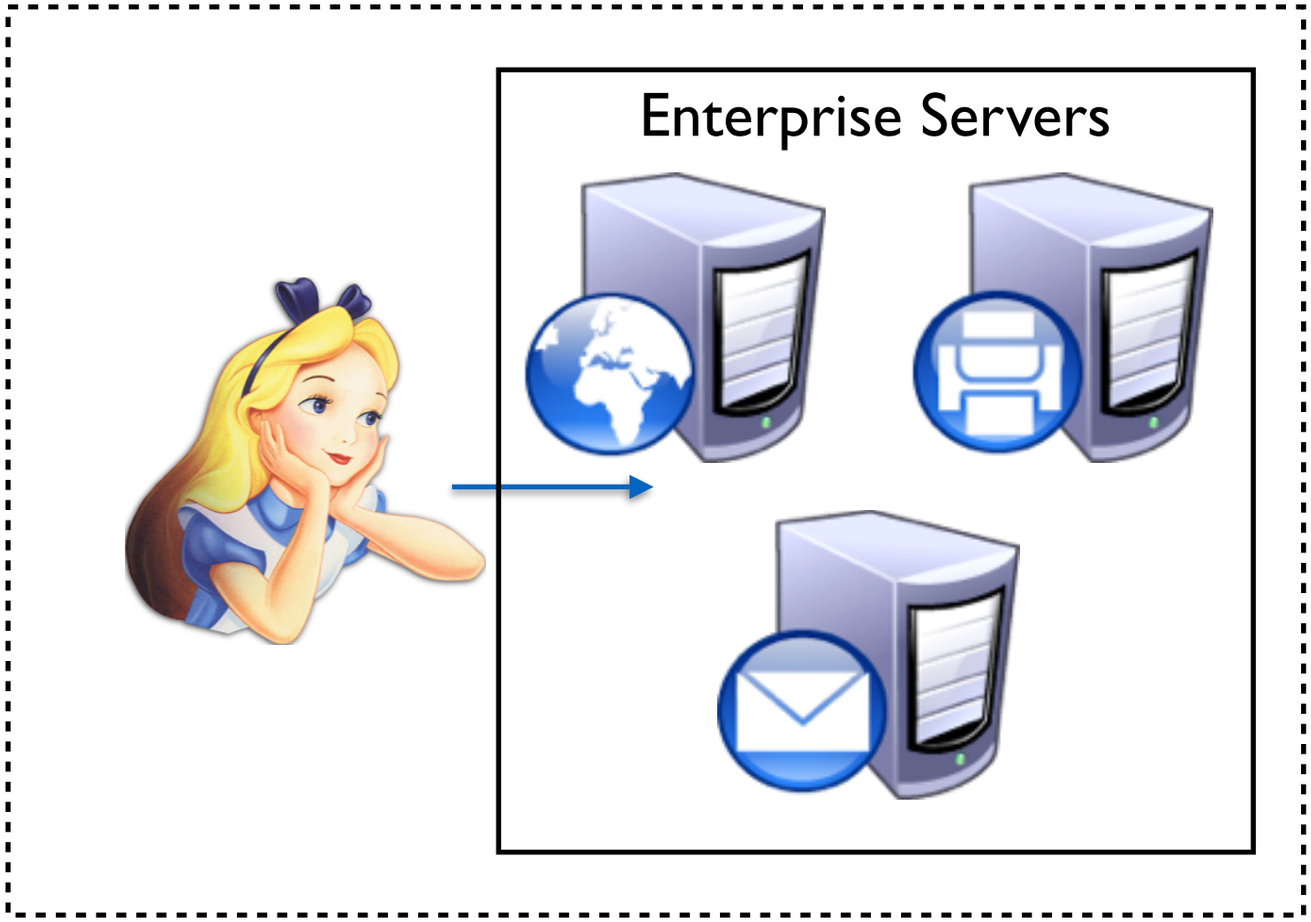


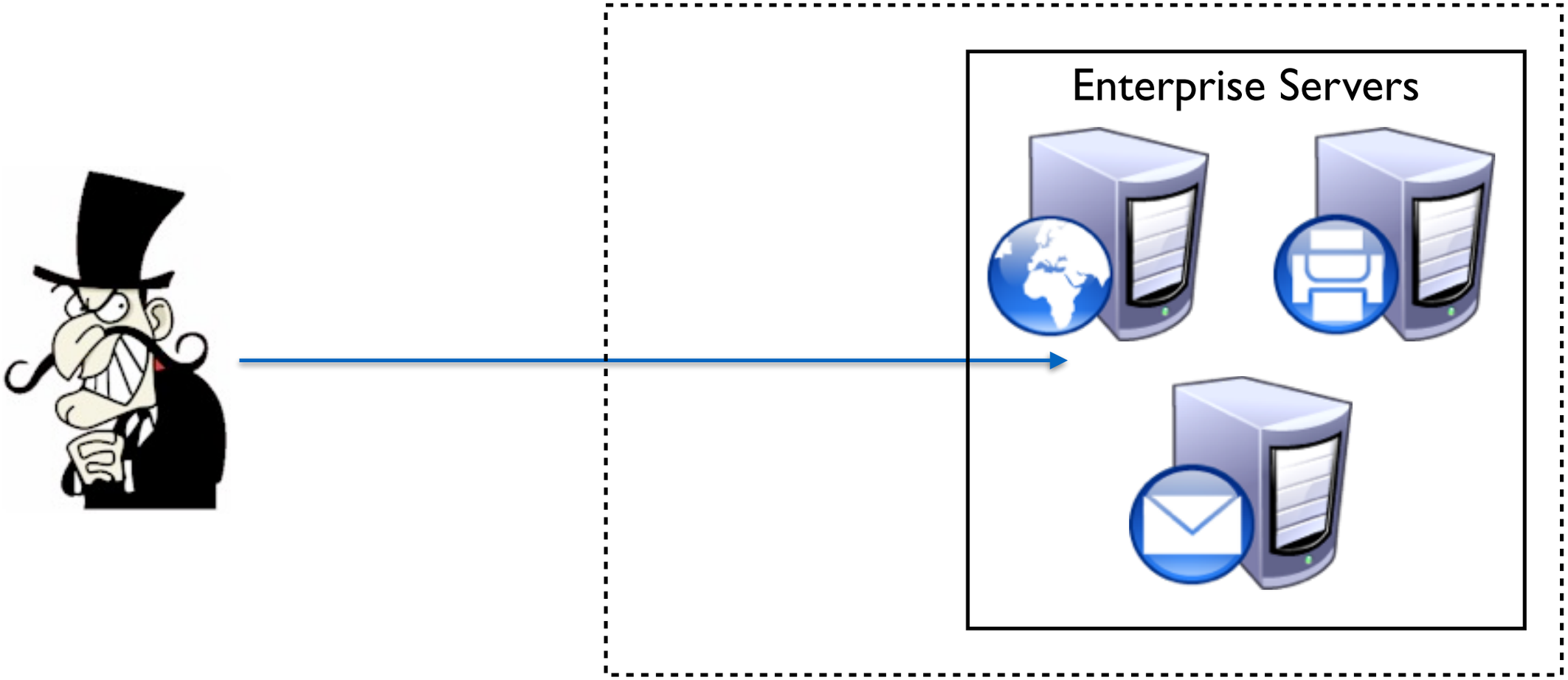
How Tor Works: 3

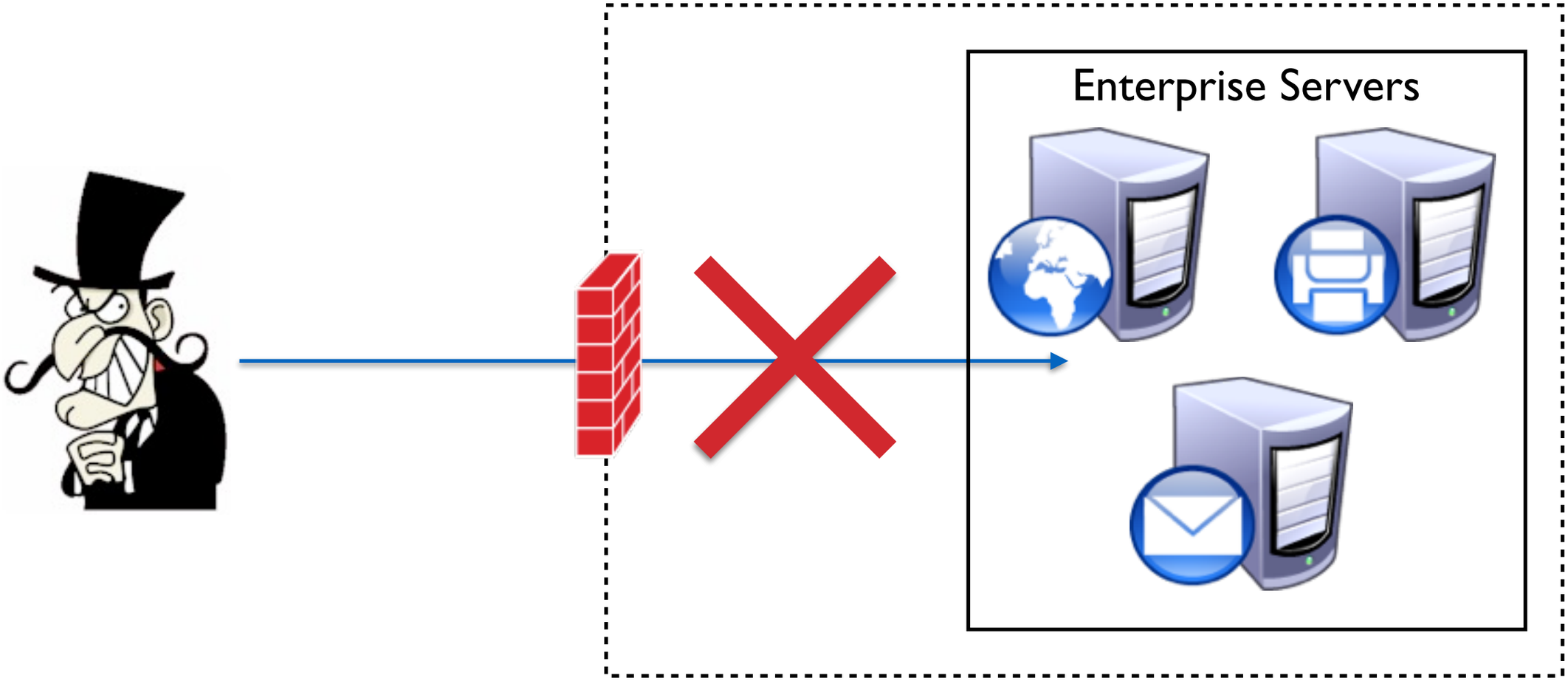


Plan for today

- Administrivia
- Anonymity
- Network Defense
  - Firewalls
  - Intrusion Detection Systems
- Exam 2 review


Enterprise Servers


Enterprise Servers


Enterprise Servers

Filtering: Firewalls

- Filtering traffic based on **policy**
  - Policy determines what is acceptable traffic
  - Access control over traffic
  - Accept or deny
- May perform other duties
  - Logging (forensics, SLA)
  - Flagging (intrusion detection)
  - QoS (differentiated services)

IP Firewall Policy

- Specifies what traffic is (not) allowed
  - Maps attributes to address and ports
  - Example: HTTP should be allowed to any external host, but inbound only to web-server
  - Rules typically refer to IP addresses, not hostnames **-- WHY?**

Source		Destination		Protocol	Flags	Actions
Address	Port	Address	Port			
*	*	1.1.1.1	80	TCP	SYN	Accept
1.1.1.*	*	*	80	TCP	SYN	Accept
*	80	1.1.1.*	*	TCP		Accept
*	*	*	*	TCP		Deny

Default accept vs. Default deny

- Default policy specifies what to do if no other policy applies
- Most OSes default to default accept
- Most organizations default to default deny

Source		Destination		Protocol	Flags	Actions
Address	Port	Address	Port			
*	*	1.1.1.1	80	TCP	SYN	Accept
1.1.1.*	*	*	80	TCP	SYN	Accept
*	80	1.1.1.*	*	TCP		Accept
*	*	*	*	TCP		Deny

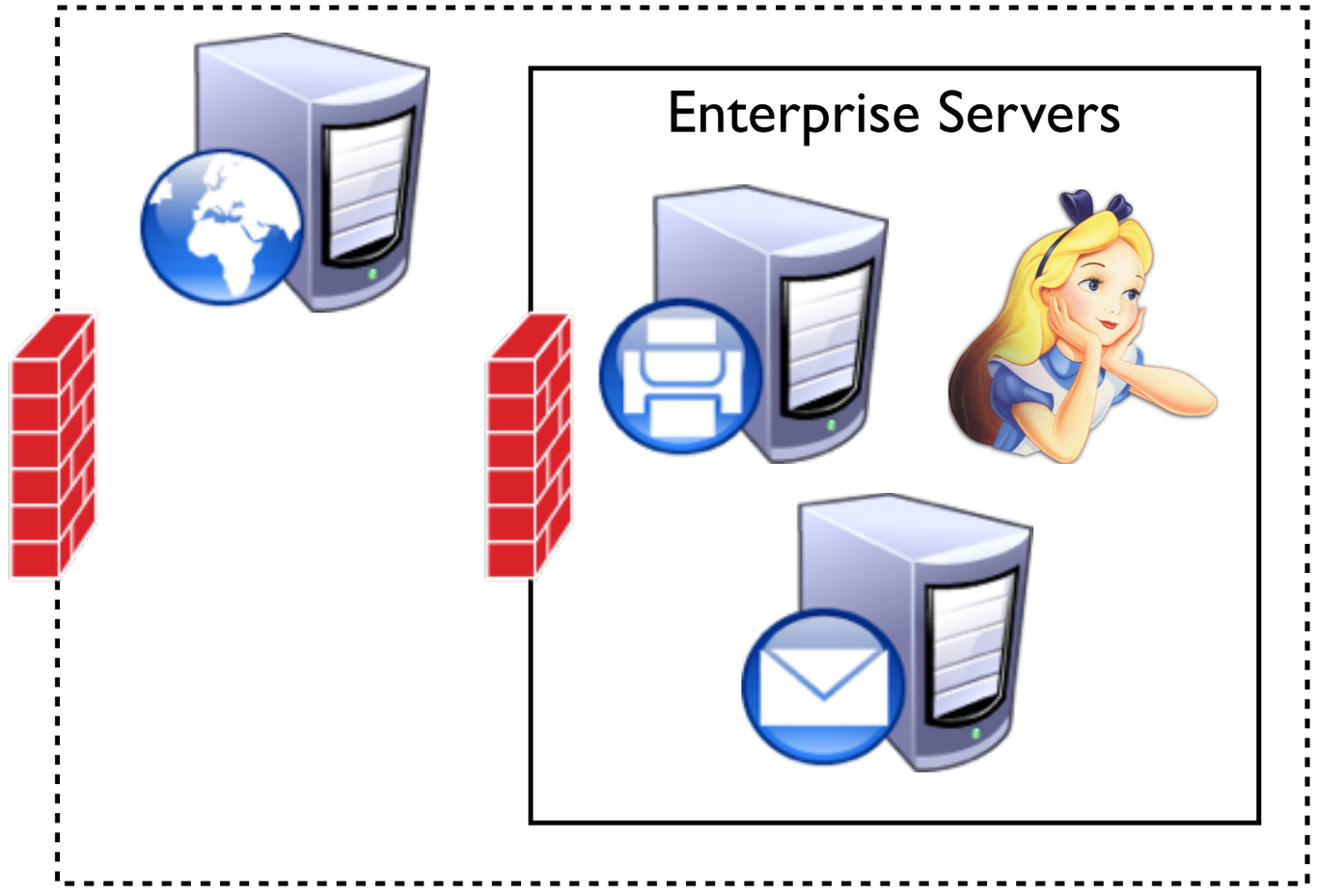
- **Deny list (blacklist)**
  - Specifies connectivity that is explicitly disallowed
  - E.g., prevent connections from badguys.com
- **Accept list (whitelist)**
  - Specifies connectivity that is explicitly allowed
  - E.g., allow connections from goodguys.com

Stateless vs. Stateful

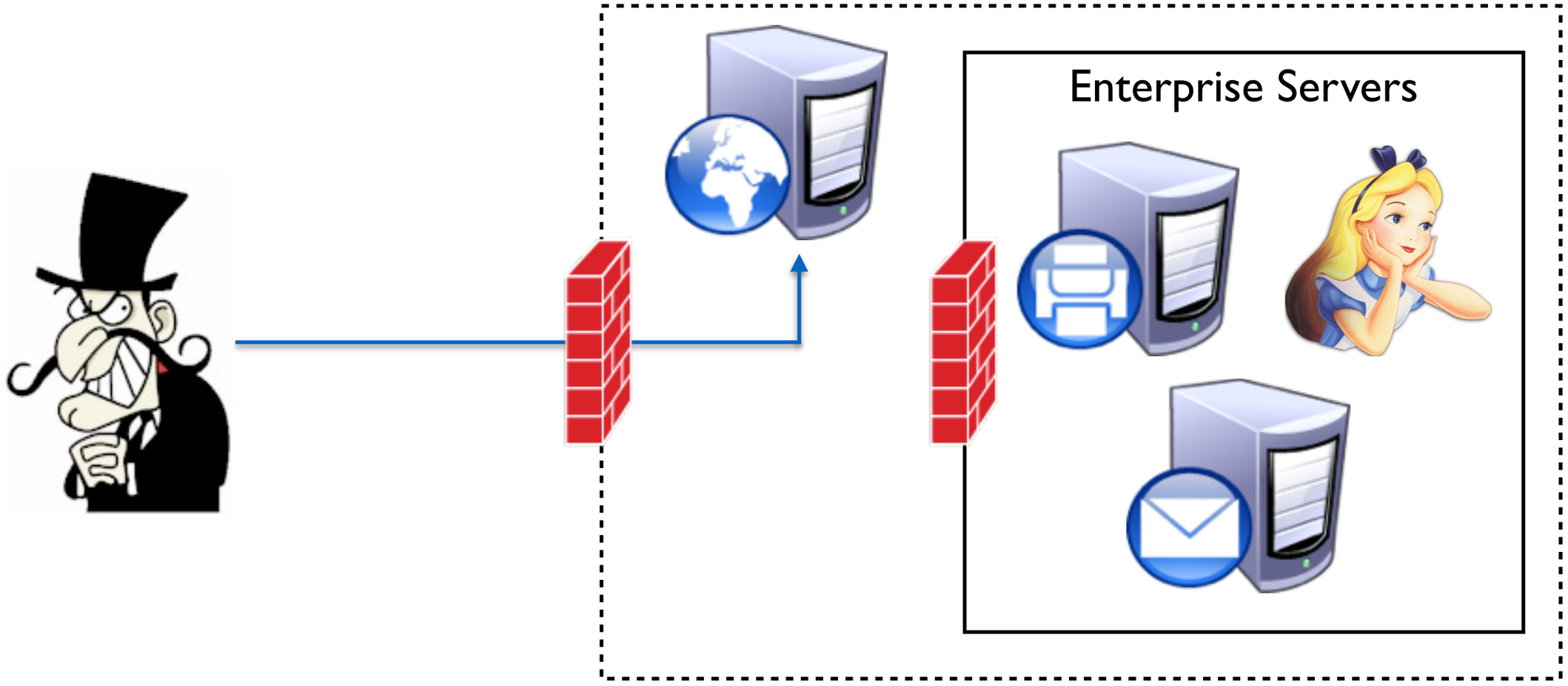
- **Stateless:** each packet considered in isolation
- Single packet contains insufficient data to make access control decision

- **Stateful:** allows historical context consideration
  - Firewall collects data over time
    - e.g., TCP packet is part of established session
- Q: What are the advantages/disadvantages of stateless and stateful?

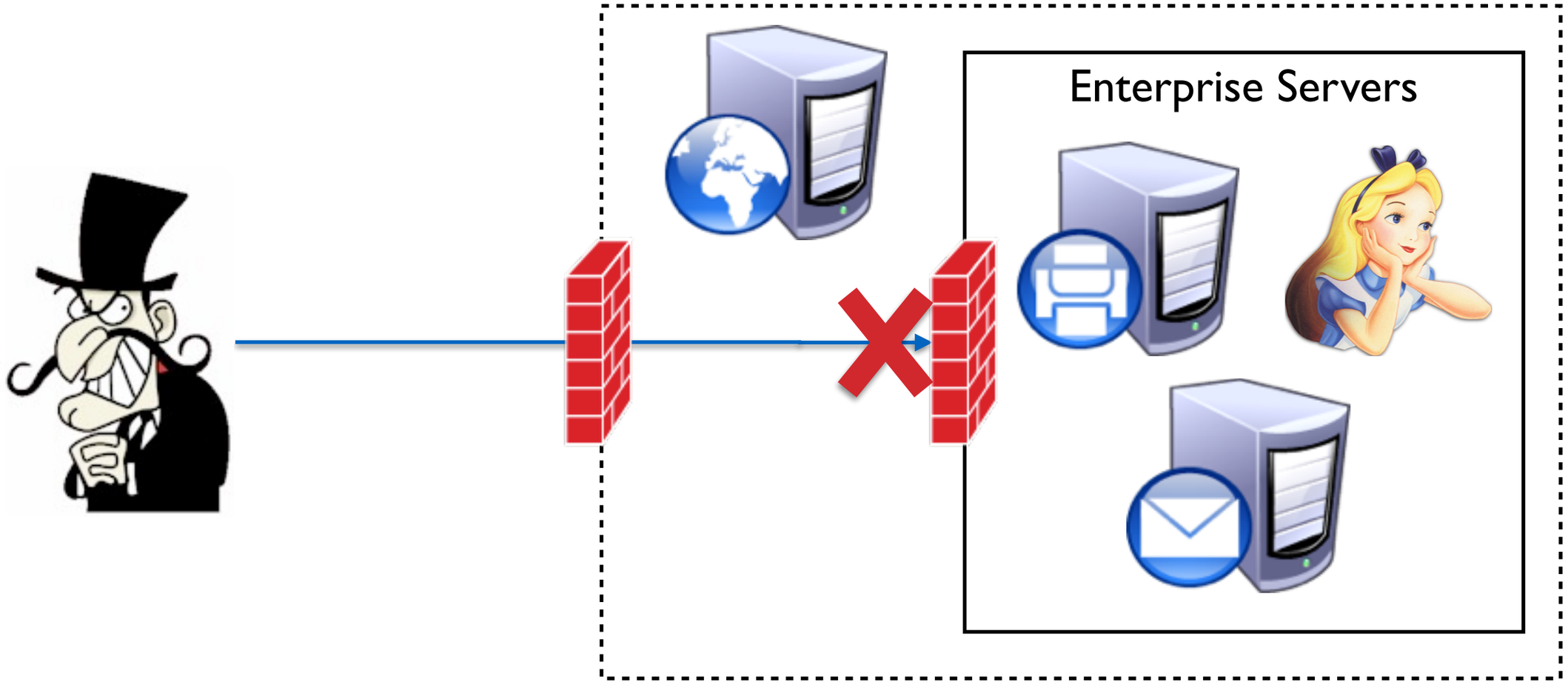
DMZ (De-militarized Zone)



DMZ (De-militarized Zone)



DMZ (De-militarized Zone)



Practical Issues and Limitations

- Network layer firewalls are dominant
  - DMZs allow multi-tiered firewalling
  - Tools are widely available and mature
  - Personal firewalls gaining popularity
- Issues
  - Network perimeters not quite as clear as before: e.g., telecommuters, VPNs, wireless
  - Every access point must be protected
  - Hard to debug, maintain consistency and correctness
  - Often seen by non-security personnel as impediment
    - E.g., Just open port X so I can use my wonder widget

Firewall Implementations

- Linux **iptables** (http://www.netfilter.org/documentation/HOWTO/packet-filtering-HOWTO.html)
  - also referred to as Netfilter
  - lots of GUIs, higher-level apps, etc.
  - e.g., ufw
- **PF**: OpenBSD Packet Filter
- Mac OS X Application Firewall
- Windows firewall thingy

iptables Concepts

The iptables firewall looks in the firewall *table* to see if the *chain* associated with the current hook *matches* a packet, and executes the *target* if it does.

- *Table*: all the firewall rules
- *Chain*: list of rules associated with the chain identifier, e.g., hook name
- *Match*: when all of a rule's field match the packet
- *Target*: operation to execute on a packet given a match

iptables Rule Parameters

- Non-comprehensive list of things you can match on:
  - Destination/Source
    - Specific IPs, or
    - IP address range and netmask
  - Protocol of packet: ICMP, TCP, etc
  - Fragmented only
  - Incoming/outgoing interface

Per Protocol Options

- Specialized matching options for rules that are specific to a particular protocol
- E.g., for TCP:
  - Source/destination ports (also for UDP)
  - SYN
  - TCP flags

Targets

- Define what to do with the packet at this time
- ACCEPT/DROP
- QUEUE for user-space application
- LOG any packet that matches
- REJECT drops and returns error packet
- RETURN enables packet to return to previous chain

Examples

```
iptables -A INPUT -s 200.200.200.2 -j ACCEPT

iptables -A INPUT -s 200.200.200.1 -j DROP

iptables -A INPUT -s 200.200.200.1 -p tcp -j DROP

iptables -A INPUT -s 200.200.200.1 -p tcp --dport telnet -j
 DROP

iptables -A INPUT -p tcp --dport telnet -i eth0 -j DROP
```

Deep Packet Inspection

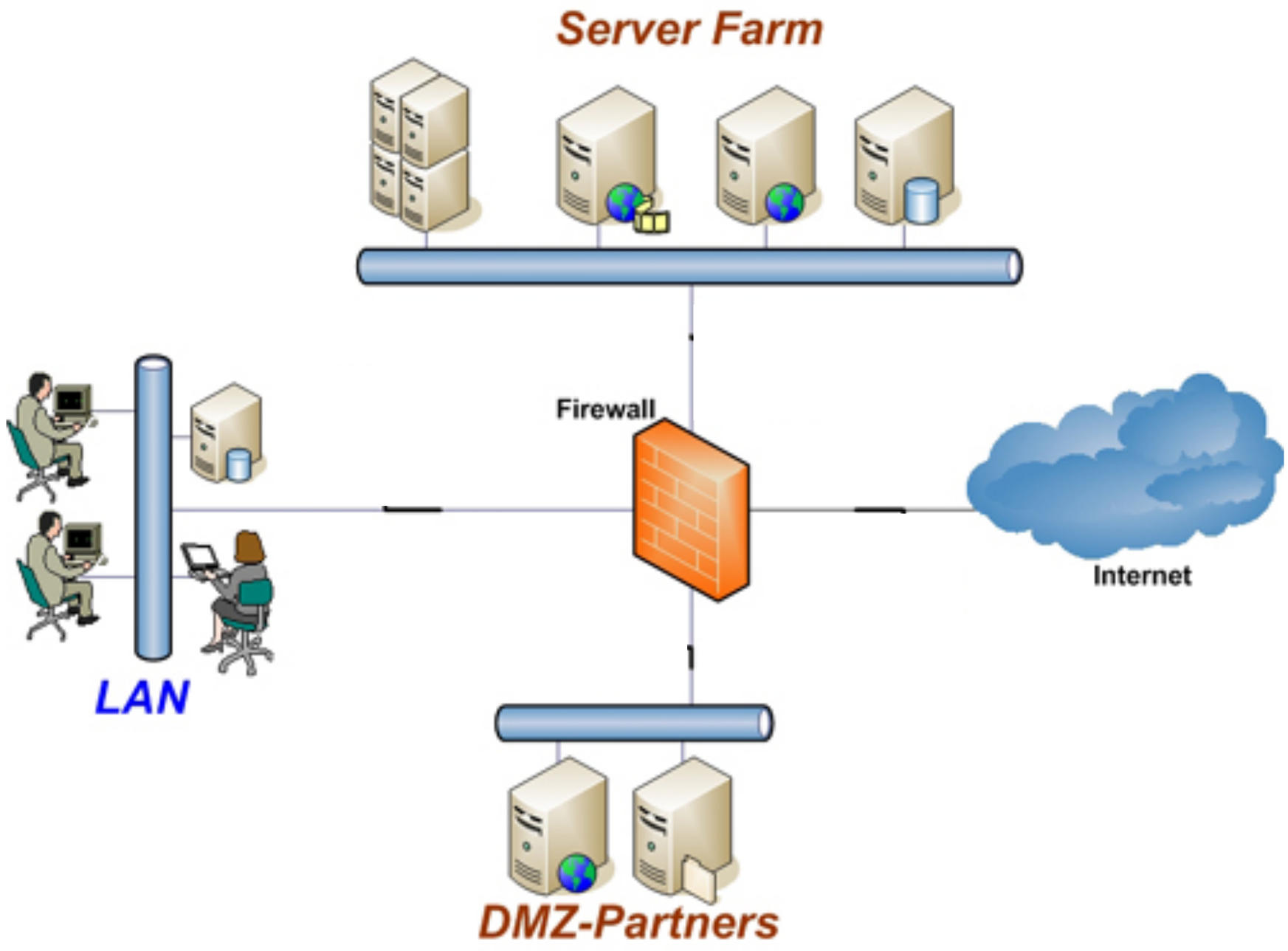
- **Deep packet inspection** looks into the internals of a packet to look for some application/content context
  - e.g., inspect HTTP for URLs that point to malicious websites
  - Can have serious privacy issues if done by, say, Comcast
- To specify a match in iptables
  - `iptables -A INPUT -p tcp -m string --algo bm --string 'exe'`
    - matches packet with content containing ‘exe’
  - `iptables -A INPUT -p tcp -m length --length 10:100`
    - matches packet with length between 10 and 100 bytes

Network Intrusion Detection Systems (NIDS)

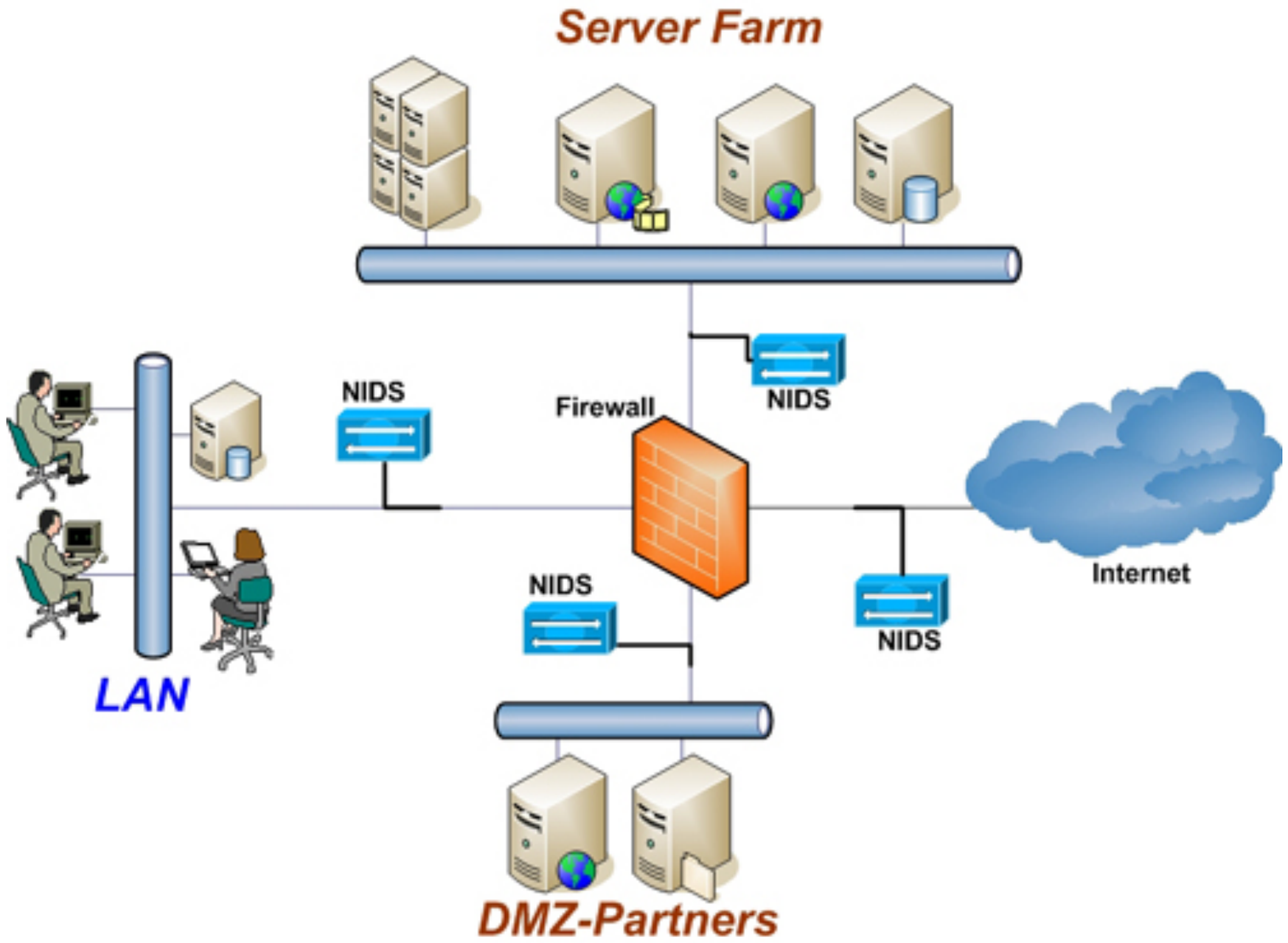
Intrusion Detection Systems

- Authorized eavesdropper that listens in on network traffic
- Makes determination whether traffic contains malware
  - usually compares payload to virus/worm signatures
  - usually looks at only incoming traffic
- If malware is detected, IDS somehow raises an alert
- Intrusion detection is a **classification problem**

Example Setup



Example Setup



Detection via Signatures

- Signature checking
  - does packet match some signature
    - suspicious headers
    - suspicious payload (e.g., shellcode)
- great at matching known signatures
- Problem: not so great for zero-day attacks -- **Q: WHY?**

Detection via Machine Learning

- Use ML techniques to identify malware
- Underlying assumption: malware will look different from non-malware
- **Supervised learning**
  - IDS requires learning phase in which operator provides pre-classified ***training data*** to learn patterns
  - Sometimes called **anomaly detection (systems)**
  - {good, 80, “GET”, “/”, “Firefox”}
  - {bad, 80, “POST”, “/php-shell.php?cmd=’rm -rf /’”, “Evil Browser”}
  - ML technique builds model for classifying never-before-seen packets
  - Problem: is new malware going to look like training malware?

Detection via Machine Learning

Everybody's Got ML, Tell Me What Else You Have: Practitioners' Perception of ML-Based Security Tools and Explanations

Jaron Mink*, Hadjer Benkraouda*, Limin Yang*, Arridhana Ciptadi†, Ali Ahmadzadeh‡, Daniel Votipka, Gang Wang*
*University of Illinois at Urbana-Champaign † Truera ‡ Blue Hexagon † Tufts University
{jaronmm2, ll6, natamb2}@illinois.edu, ofigueira@alumni.scu.edu, {yvw, gangw}@illinois.edu

***Abstract*—Significant efforts have been investigated to develop machine learning (ML) based tools to support security operations. However, they still face key challenges in practice. A generally perceived weakness of machine learning is the lack of explanation, which motivates researchers to develop machine learning explanation techniques. However, it is not yet well understood how security practitioners perceive the benefits and pain points of machine learning and corresponding explanation methods in the context of security operations. To fill this gap and understand "what is needed", we conducted semi-structured interviews with 18 security practitioners with diverse roles, duties, and expertise. We find practitioners generally believe that ML tools should be used in conjunction with (instead of replacing) traditional rule-based methods. While ML's output is perceived as difficult to reason, surprisingly, rule-based methods are not strictly easier to interpret. We also find that only few practitioners considered security (robustness to adversarial attacks) as a key factor for the choice of tools. Regarding ML explanations, while recognizing their values in model verification and understanding security events, practitioners also identify gaps between existing explanation methods and the needs of their downstream tasks. We collect and synthesize the suggestions from practitioners regarding**

operations centers (SOCs) have deployed machine learning-based tools, they are not rated among the most effective [18].

More recently, the machine learning and the security communities have investigated *machine learning explanations* to improve machine learning based tools' usability [14]–[17], [19]–[21]. For example, an explanation method can highlight key features in a binary (e.g., specific bytecode gadgets, API calls) to explain its classification as malware. However, most existing efforts focus on designing technical solutions to generate accurate and robust explanations, without proactively engaging stakeholders who use, manage, or implement these tools in the field.

In this paper, we aim to fill this gap by exploring security practitioners' perceptions of machine learning based security tools and the corresponding explanation methods. The goal is to provide a deeper understanding of "what is needed" and facilitate further research into usable machine learning and explanation methods for security operations. We seek to answer three main research questions:

RQ1 Where and how is machine learning used in security operations centers (SOC)?

RQ2 What are the perceived benefits and challenges in using machine learning in practical security operations?

RQ3 How are existing machine learning explanation tech-

Base Rate Fallacy

- Occurs when we assess P(X|Y) without considering prior probability of X and the total probability of Y
- Example:
  - *Base rate* of malware is 1 packet in a 10,000
  - Intrusion detection system is 99% accurate
    - 1% false positive rate (benign marked as malicious 1% of the time)
    - 1% false negative rate (malicious marked as benign 1% of the time)
  - Packet X is marked by the NIDS as malware. What is the probability that packet X actually is malware?

Base Rate Fallacy

- 1% false positive rate (benign marked as malicious 1% of the time); TPR=99%
- 1% false negative rate (malicious marked as benign 1% of the time)
- *Base rate* of malware is 1 packet in 10,000
- Find Pr(IsMalware|MarkedAsMalware)

Base Rate Fallacy

- 1% false positive rate (benign marked as malicious 1% of the time); TPR=99%
- 1% false negative rate (malicious marked as benign 1% of the time)
- *Base rate* of malware is 1 packet in 10,000
- Find Pr(IsMalware|MarkedAsMalware)
- Pr(Is|Marked) = Pr(Marked|Is)Pr(Is) / Pr(Marked)

Base Rate Fallacy

- 1% false positive rate (benign marked as malicious 1% of the time); TPR=99%
- 1% false negative rate (malicious marked as benign 1% of the time)
- *Base rate* of malware is 1 packet in 10,000
- Find Pr(IsMalware|MarkedAsMalware)
- Pr(Is|Marked) = Pr(Marked|Is)Pr(Is) / Pr(Marked)
  - Pr(Marked|Is)Pr(Is) = 0.99*1/10,000

Base Rate Fallacy

- 1% false positive rate (benign marked as malicious 1% of the time); TPR=99%
- 1% false negative rate (malicious marked as benign 1% of the time)
- *Base rate* of malware is 1 packet in 10,000
- Find Pr(IsMalware|MarkedAsMalware)
- Pr(Is|Marked) = Pr(Marked|Is)Pr(Is) / Pr(Marked)
  - Pr(Marked|Is)Pr(Is) = 0.99*1/10,000
  - Pr(Marked) = Pr(Marked|Is)Pr(Is) + Pr(Marked|IsNot)Pr(IsNot)

Base Rate Fallacy

- 1% false positive rate (benign marked as malicious 1% of the time); TPR=99%
- 1% false negative rate (malicious marked as benign 1% of the time)
- *Base rate* of malware is 1 packet in 10,000
- Find Pr(IsMalware|MarkedAsMalware)
- Pr(Is|Marked) = Pr(Marked|Is)Pr(Is) / Pr(Marked)
  - Pr(Marked|Is)Pr(Is) = 0.99*1/10,000
  - Pr(Marked) = Pr(Marked|Is)Pr(Is) + Pr(Marked|IsNot)Pr(IsNot)
    - Pr(Marked) = (.99*1/10,000) + (0.01*9,999/10,000)

Base Rate Fallacy

- 1% false positive rate (benign marked as malicious 1% of the time); TPR=99%
- 1% false negative rate (malicious marked as benign 1% of the time)
- *Base rate* of malware is 1 packet in 10,000
- Find Pr(IsMalware|MarkedAsMalware)
- Pr(Is|Marked) = Pr(Marked|Is)Pr(Is) / Pr(Marked)
  - Pr(Marked|Is)Pr(Is) = 0.99*1/10,000
  - Pr(Marked) = Pr(Marked|Is)Pr(Is) + Pr(Marked|IsNot)Pr(IsNot)
    - Pr(Marked) = (.99*1/10,000) + (0.01*9,999/10,000)
- Pr(Is|Marked) = 0.98%

Problems with IDSes

- VERY difficult to get both good recall and precision
- Malware comes in small packages
- Looking for one packet in a million (billion? trillion?)
- If insufficiently sensitive, IDS will miss this packet (low recall)
- If overly sensitive, too many alerts will be raised (low precision)

Snort

- Open source IDS
- Signature detection
- Lots of available rulesets
  - alert tcp $EXTERNAL_NET any -> $SQL_SERVERS 3306 (msg:"MYSQL root login attempt"; flow:to_server,established; content:"|0A 00 00 01 85 04 00 00 80|root|00|"; classtype:protocol-command-decode; sid:1775; rev:2;)
  - alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-PHP Setup.php access"; flow:to_server,established; uricontent:"/Setup.php"; nocase; reference:bugtraq,9057; classtype:web-application-activity; sid:2281; rev:2;)

Defenses thus far

- **Firewalls** and **Intrusion Prevention Systems** prevent malicious packets from entering the network (in theory)
- **Intrusion Detection Systems** alert network administrators to intrusion attempts
- Both systems work best when malware is well-understood and easily fingerprinted

How do we learn about and study malware?

Plan for today

- Administrivia
- Anonymity Review
- Network Defense
- Honeypots
  - Overview
  - Malware analysis
  - Setting up honeypots

Honeypots

- **Honeypot:** a controlled environment constructed to trick malware into thinking it is running in an unprotected system
  - collection of decoy services (fake mail, web, ftp, etc.)
  - decoys often mimic behavior of unpatched and vulnerable services

Honeypots

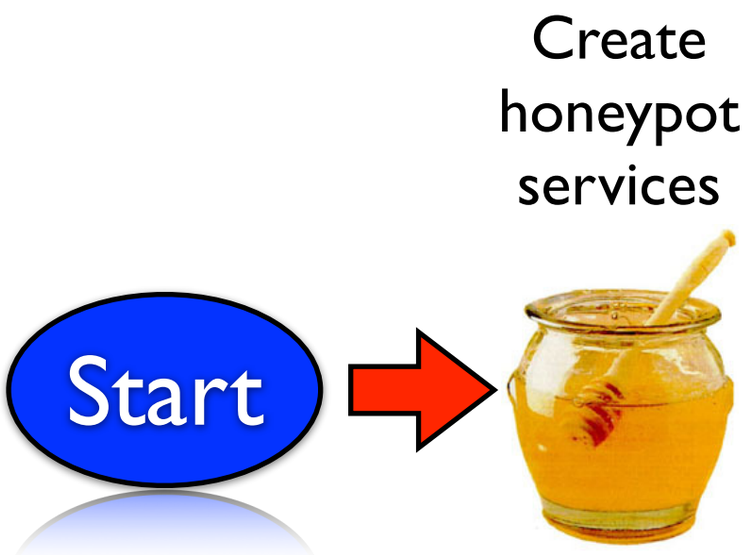
- Three main uses:
  - forensic analysis: better understand how malware works; collect evidence for future legal proceedings
  - risk mitigation:
    - provide "low-hanging fruit" to distract attacker while safeguarding the actually important services
    - tarpits: provide very slow service to slow down the attacker
  - malware detection: examine behavior of incoming request in order to classify it as benign or malicious

Honeypots

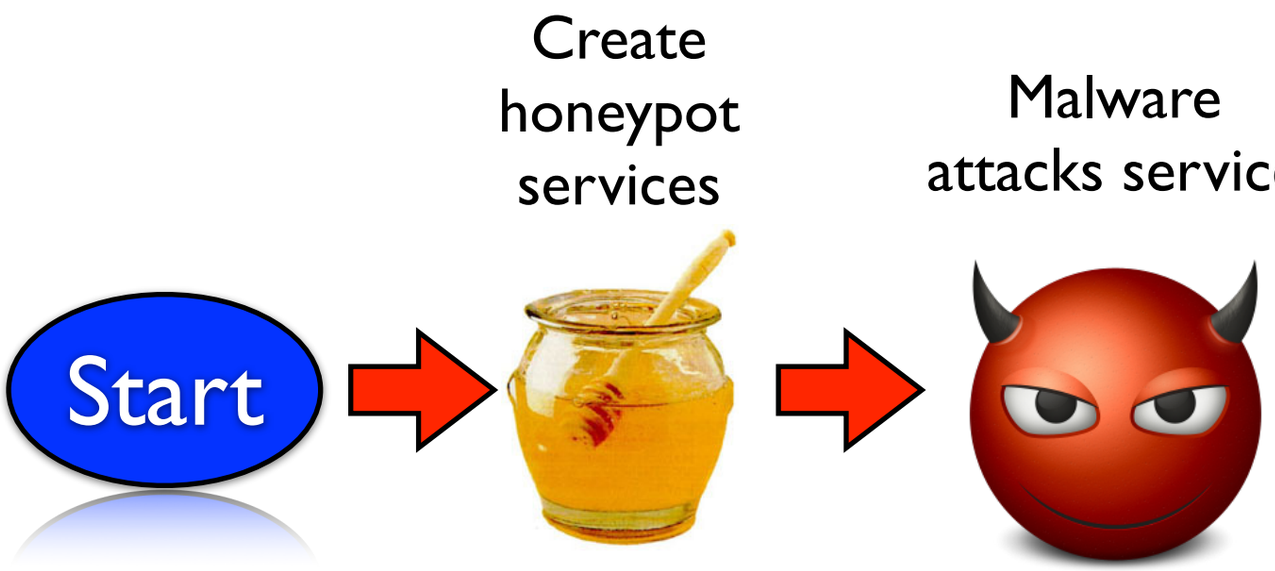
- Two main types:
  - **Low-interaction:** emulated services
    - inexpensive
    - may be easier to detect
  - **High-interaction:** no emulation; honeypot maintained inside of real OS
    - expensive
    - good realism

Example Honeypot Workflow

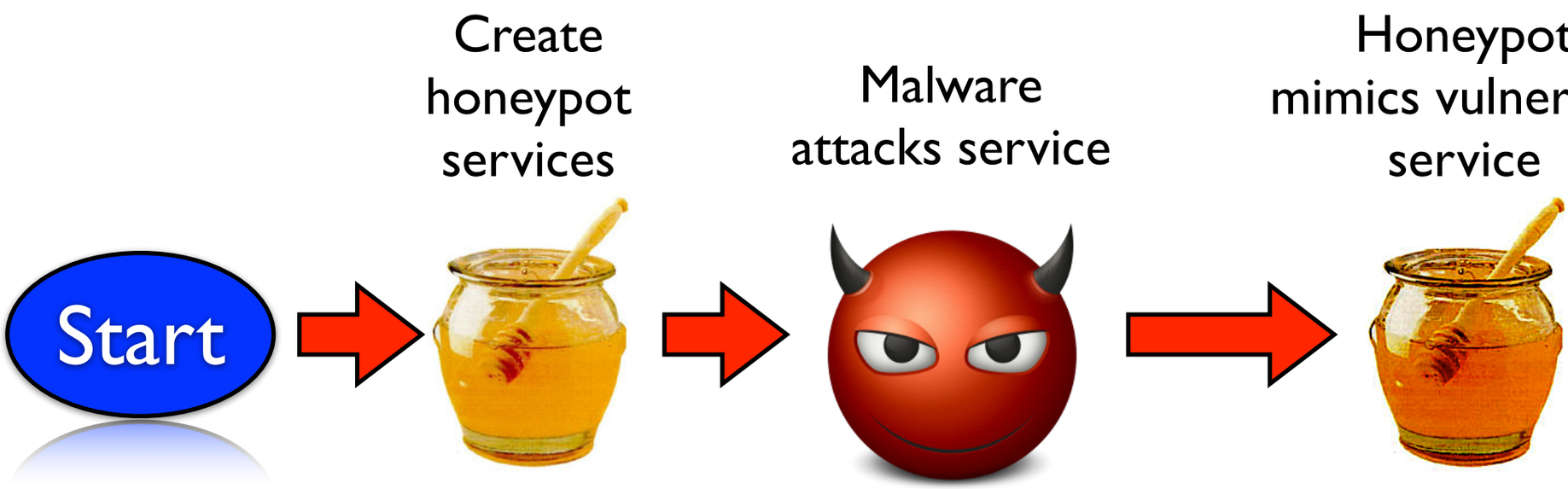
Example Honeypot Workflow



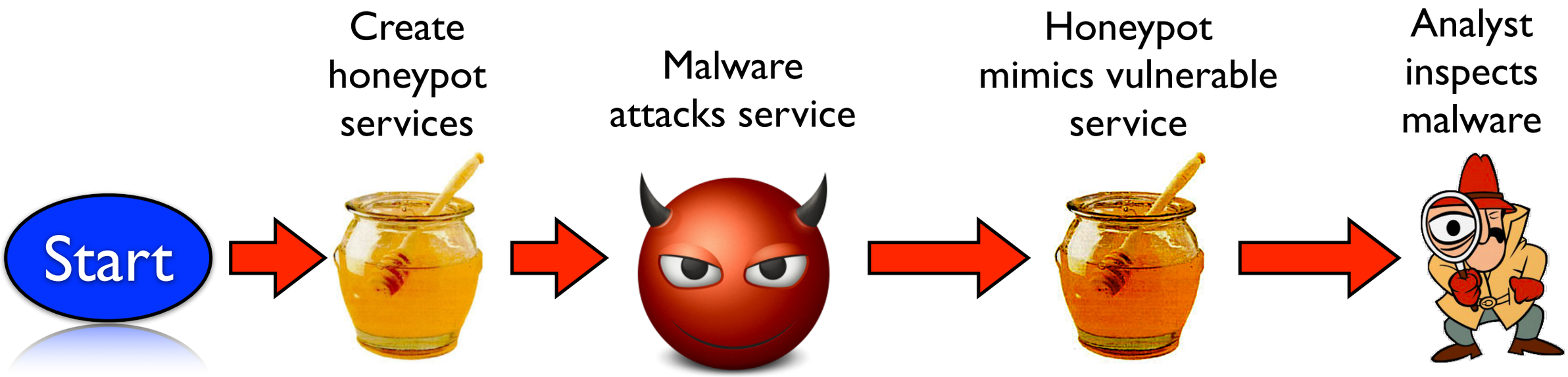
Example Honeypot Workflow



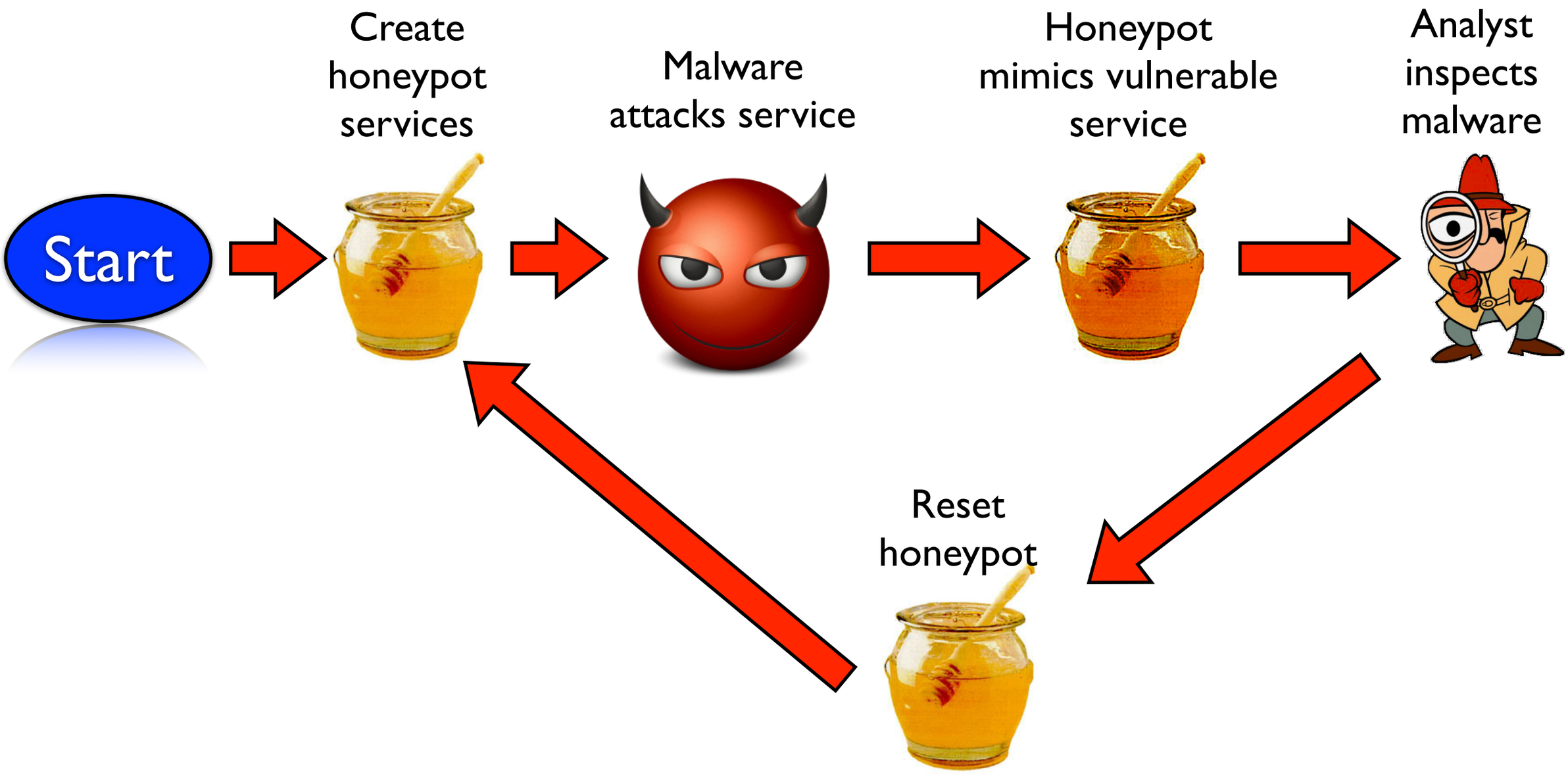
Example Honeypot Workflow



Example Honeypot Workflow



Example Honeypot Workflow



Plan for today

- Administrivia
- Anonymity Review
- Network Defense
- Honeypots
  - Overview
  - Malware analysis
  - Setting up honeypots

Examining Malware

- **Trace system calls:**
  - most OSes support method to trace sequence of system calls
    - e.g., ptrace, strace, etc.
  - all "interesting" behavior (e.g., networking, file I/O, etc.) must go through system calls
  - capturing sequence of system calls (plus their arguments) reveals useful info about malware's behavior

Examining Malware

- **Observe filesystem changes and network IO:**
  - "diff" the filesystem before and after
    - which files are the malware reading/ writing?
  - capture network packets
    - to whom is the malware communicating

Internet Background Radiation

- **Internet Background Radiation** or **Backscatter:** Traffic that is sent to addresses on which no device is set up (these unused portions of the Internet are called **darknets**)
- Backscatter primarily originates from spam, worms, and port scans
- Estimated at 5.5Gbps
- Estimated that 70% of background radiation due to Conficker Worm

Plan for today

- Administrivia
- Anonymity Review
- Network Defense
- Honeypots
  - Overview
  - Malware analysis
  - Setting up honeypots

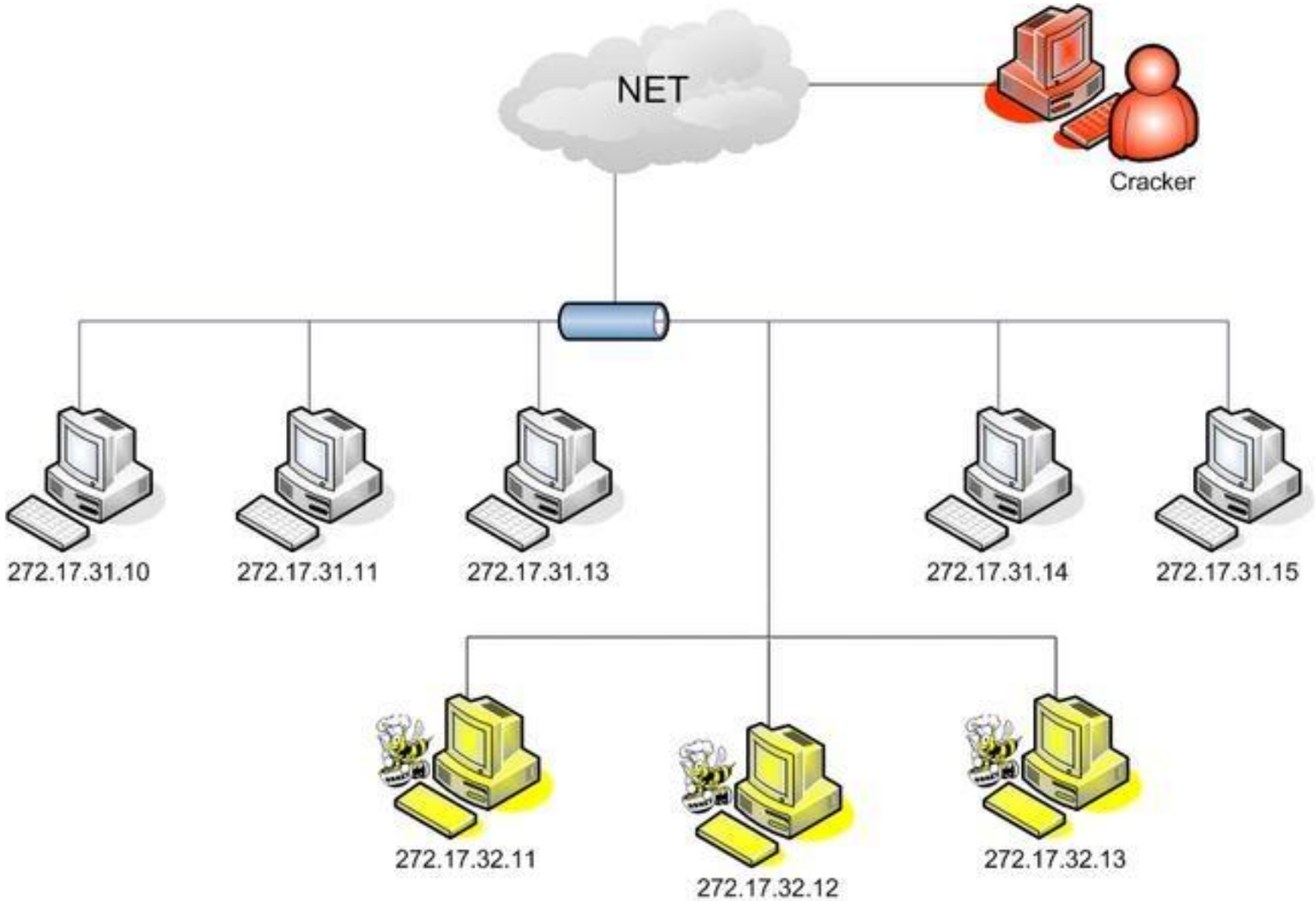
Challenges

- Honeypot must resemble actual machine
  - simulate actual services (Apache, MySQL, etc.)
  - but not too much... bad form to actually help propagate the worm (legal risks!)
- Some worms do a reasonably good job of detecting honeypots

Honeynets

- **Honeynet:** also called **honeyfarms**
  - Collection of honeypots that simulate a network; or
  - Single honeypot that emulates services on multiple emulated "machines" (that is, on a network)

Example Deployment



honeyd

- Open-source virtual honeynet
- creates virtual hosts on network
- services actually run on a single host
- scriptable services

honeyd example:
FTP service (ftp.sh)

```
echo "$DATE: FTP started from $1 Port $2" >> $log
echo -e "220 $host.$domain FTP server (Version wu-2.6.0(5) $DATE) ready."
...
case $incmd_nocase in

        QUIT* )
            echo -e "221 Goodbye.\r"
            exit 0;;
        SYST* )
            echo -e "215 UNIX Type: L8\r"
            ;;
    HELP* )
            echo -e "214-The following commands are recognized (* =>'s unimplemented).\r"
            echo -e "   USER   PORT   STOR   MSAM*  RNTO   NLST   MKD    CDUP\r"
            echo -e "   PASS   PASV   APPE   MRSQ*  ABOR   SITE   XMKD   XCUP\r"
            echo -e "   ACCT*  TYPE   MLFL*  MRCP*  DELE   SYST   RMD    STOU\r"
            echo -e "   SMNT*  STRU   MAIL*  ALLO   CWD    STAT   XRMD   SIZE\r"
            echo -e "   REIN*  MODE   MSND*  REST   XCWD   HELP   PWD    MDTM\r"
            echo -e "   QUIT   RETR   MSOM*  RNFR   LIST   NOOP   XPWD\r"
            echo -e "214 Direct comments to ftp@$domain.\r"
            ;;
```

honeyd example:
FTP service (ftp.sh)

```
echo "$DATE: FTP started from $1 Port $2" >> $log
echo -e "220 $host.$domain FTP server (Version wu-2.6.0(5) $DATE) ready."
...
case $incmd_nocase in

        QUIT* )
            echo -e "221 Goodbye.\r"
            exit 0;;
        SYST* )
            echo -e "215 UNIX Type: L8\r"
            ;;
    HELP* )
            echo -e "214-The following commands are recognized (* =>'s unimplemented).\r"
            echo -e "   USER   PORT   STOR   MSAM*  RNTO   NLST   MKD    CDUP\r"
            echo -e "   PASS   PASV   APPE   MRSQ*  ABOR   SITE   XMKD   XCUP\r"
            echo -e "   ACCT*  TYPE   MLFL*  MRCP*  DELE   SYST   RMD    STOU\r"
            echo -e "   SMNT*  STRU   MAIL*  ALLO   CWD    STAT   XRMD   SIZE\r"
            echo -e "   REIN*  MODE   MSND*  REST   XCWD   HELP   PWD    MDTM\r"
            echo -e "   QUIT   RETR   MSOM*  RNFR   LIST   NOOP   XPWD\r"
            echo -e "214 Direct comments to ftp@$domain.\r"
            ;;
```

Virtual Machines

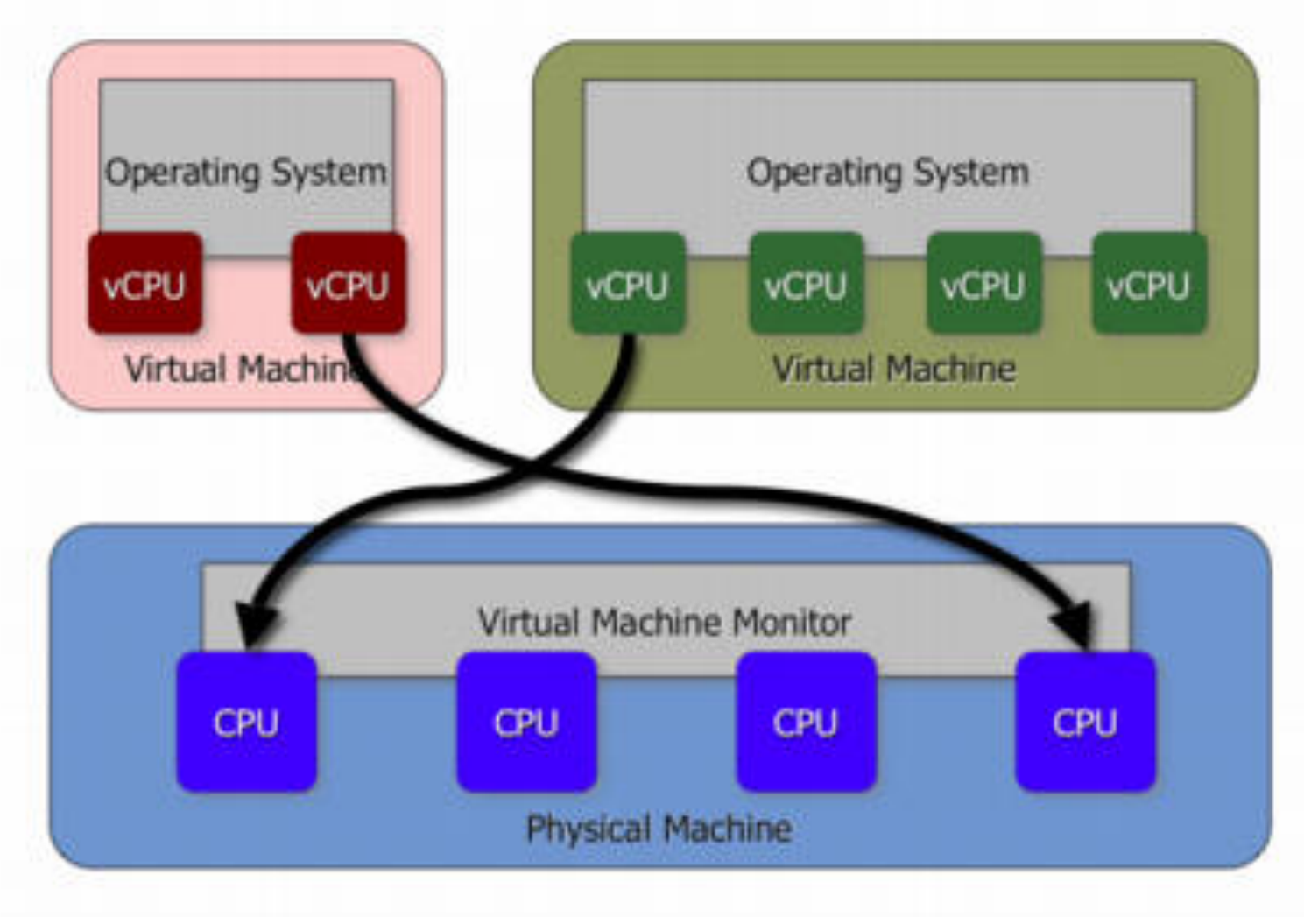
Virtual Machines

- **Virtual machine:** isolated virtual hardware running within a single operating system
  - i.e., a software implementation of hardware
  - usually provides emulated hardware which runs OS and other applications
    - i.e., a computer inside of a computer
- What's the point?
  - extreme software isolation -- programs can't easily interfere with one another if they run on separate machines

Virtual Machines

- **Virtual machine:** isolated virtual hardware running within a single operating system
  - i.e., a software implementation of hardware
  - usually provides emulated hardware which runs OS and other applications
  - i.e., a computer inside of a computer
- What's the point?
  - extreme software isolation -- programs can't easily interfere with one another if they run on separate machines
  - much better hardware utilization than with separate machines
  - power savings
  - easy migration -- no downtime for hardware repairs/improvements

Virtual Machines



Honeypots and Virtual Machines

- Most virtual machines provide checkpointing features
  - **Checkpoint** (also called **snapshot**) consists of all VM state (disk, memory, etc.)
  - In normal VM usage, user periodically creates snapshots before making major changes
  - Rolling back ("restoring") to snapshot is fairly inexpensive
- **Checkpointing features are very useful for honeypots**
  - Let malware do its damage
  - Pause VM and safely inspect damage from virtual machine monitor
  - To reset state, simply restore back to the checkpoint

Honeypots and Virtual Machines

- Virtual Machines are also very useful for analyzing malware:
  - execute malware one instruction at a time
  - pause malware
  - easily detect effects of malware by looking at "diffs" between current state and last snapshot
  - execute malware on one VM and uninfected software on another; compare state

Detecting VMs

- Lots of research into detecting when you're in a virtual machine
  - examine hardware drivers
  - time certain operations
  - look at ISA support
- Malware does this too!
  - if not in VM, wreak havoc
  - if in VM, self-destruct

Plan for today

- Administrivia
- Anonymity Review
- Network Defense
- Honeypots
  - Overview
  - Malware analysis
  - Setting up honeypots