CSE227 – Graduate Computer Security

Software Security Catchup + Side Channels pt. 1

UC San Diego

Housekeeping

General course things to know

- Due by **4/10** at 11:59
  - Project intention form: https://forms.gle/bgaAtD742X8YSxgw5
  - #FinAid Canvas quiz: https://canvas.ucsd.edu/courses/74259/quizzes/247982
- Project specification released here: https://kumarde.com/cse227-sp26/spec.pdf
- **4/14;** made one paper optional (Light Commands)

News

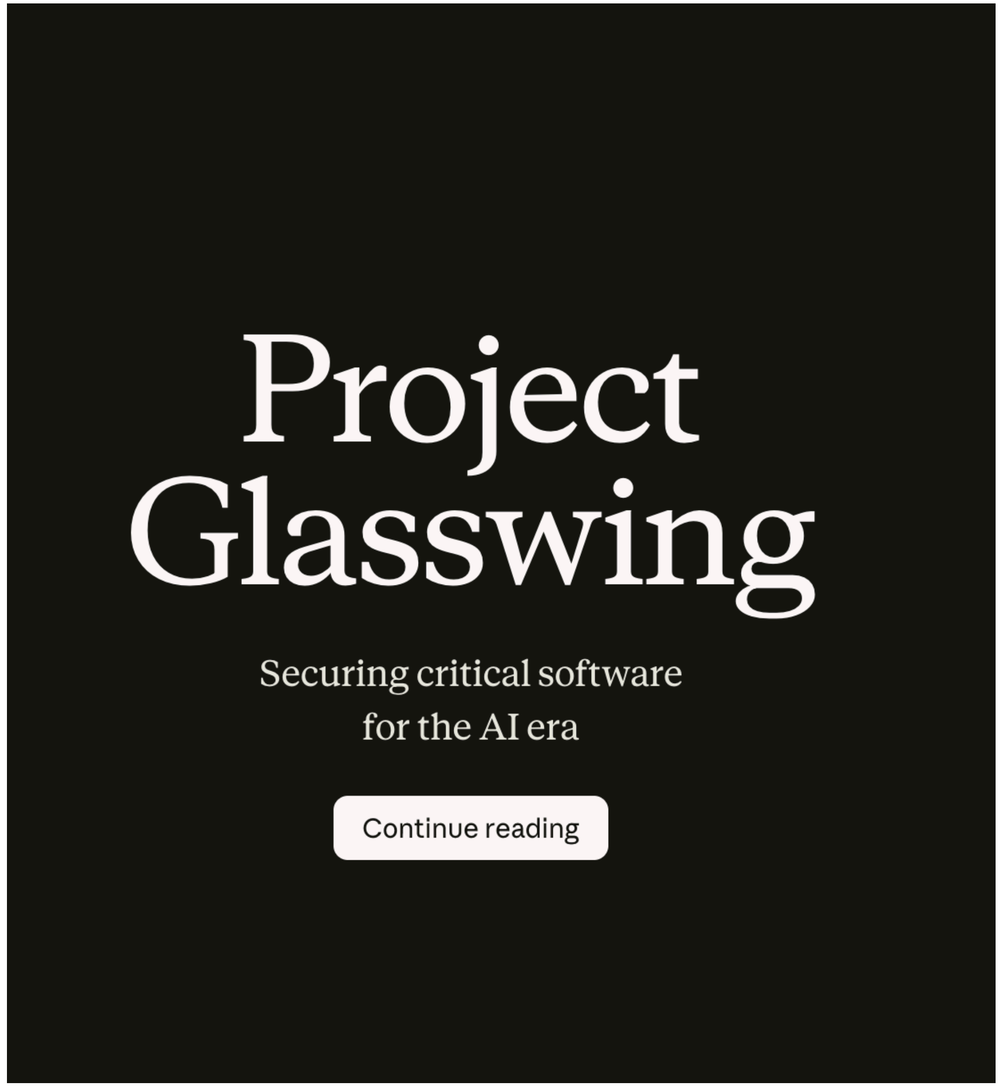
Claude keeps Claude-ing

Assessing Claude Mythos Preview's cybersecurity capabilities

Mythos Preview is capable of ***identifying*** *and then* ***exploiting*** *zero-day vulnerabilities in every* ***major operating system and every major web browser*** *when directed by a user to do so.*

…not just run-of-the-mill stack-smashing exploits… web browser exploit that chained together four vulnerabilities… autonomously obtained local privilege escalation exploits on Linux and other operating systems… autonomously wrote a remote code execution exploit on FreeBSD's NFS server that granted full root access to unauthenticated users by splitting a ***20-gadget ROP chain*** *over multiple packets.*

Claude keeps Claude-ing



My take on all this uncertainty

- Not the first time a mystical box has produced vulnerabilities!
  - Fuzzing, symbolic execution, dynamic tracing, etc.
- Still… the complexity here is **much more vast**; somewhat surprising me, and there's definitely something new here
- Note: The solution to solving these problems is… **partnering with companies (aka people).** Many of our core security problems are still *people* problems
- Good outcome: Defenders get to critical software before attackers, and make the software we rely on more secure
- Bad (maybe likely) outcome: Attackers get to vulns before defenders can, and wreak 10x the amount of havoc before defenders fix the problems

Back to class

Quick recap

- We talked about the basic layout of C programs, how memory is laid out, and a basic primer on x86 assembly code
- We then detailed the high level of a *buffer overflow attack* — an attack where a copy into a buffer can be *overwritten* to manipulate control flow
- We highlighted *why* some of these attacks are possible… and a fundamental security issue: with **code** vs. **data**

Defenses against code vs. data

- W^X (W *xor* X), or DEP
  - Memory protection policy whereby every *page* in an address space is either writeable or executable but **not both**
- You can break DEP by conducting a **return-to-libc** attack
  - Recall… **parts of `libc` is loaded in process memory**
  - Remember `execve()`? If we can return control there, we can start any process…
    - Called a **return-to-libc** attack

Return-to-libc

- What is a return-to-libc attack? *Return control to* ***system functions*** *to execute shellcode*

How to handle return-to-libc attacks?

- Remove all code you don’t need!
  - If you don’t use `execve`, don’t load it in process memory
  - **This can be done at compile time**
- But not perfect…
  - Some dependencies are really challenging to find and remove and reduce functionality
  - Fundamental tradeoff between flexibility and security

Limitations with return-to-libc attacks

- “Straight line limited”
  - Means you can only enter into one libc function after another
- “Removal limited”
  - If you remove libc function that aren’t useful, you can seriously hamper attackers

Limitations with return-to-libc attacks

- “Straight line limited”
  - Means you can only enter into one libc function after another
- “Removal limited”
  - If you remove libc function that aren’t useful, you can seriously hamper attackers
- **…turns out… those assumptions are wrong, you don’t even need functions!**

Return-Oriented Programming

- What is return-oriented programming?

Return-Oriented Programming

- This paper demonstrates that you don't even need function calls, but all you need are ***micro sequences of instructions*** to mess with control flow of a program
- What is the fundamental insight about x86 that enables this attack?

Return-Oriented Programming

- This paper demonstrates that you don't even need function calls, but all you need are *micro sequences of instructions* to mess with control flow of a program
- What is the fundamental insight about x86 that enables this attack?
  - x86 instructions are **ambiguous** and **dense**, so shifting by a single byte often leads to interesting strings of instructions
  - All you need is **ret** to chain gadgets together

Return-Oriented Programming

- What is return-oriented programming?
- How do you execute return-oriented programming?

Return-Oriented Programming

- What is return-oriented programming?
- How do you execute return-oriented programming?
  - Every **ret** takes you to your next sequence of microinstructions (gadget) that does something that you want…. in a way, you're turning %esp (stack pointer) into your instruction pointer

Gadgets Galore

- What is a *useful* gadget in this paper?

Gadgets Galore

- What is a *useful* gadget in this paper?
  - Anything that ends w/ **ret** and doesn't alter control flow is good (because we can set up the stack the way we like!)

Gadgets Galore

- What is a *useful* gadget in this paper?
  - Anything that ends w/ **ret** and doesn't alter control flow is good (because we can set up the stack the way we like!)
- Why do these gadgets exist?

Gadgets Galore

- What is a *useful* gadget in this paper?
  - Anything that ends w/ **ret** and doesn't alter control flow is good (because we can set up the stack the way we like!)
- Why do these gadgets exist?
  - Compilers commonly add them at the end of a function! Very hard to avoid.

```
$otool -t /bin/ls |grep c3
0000000100000f70        39 48 38 7f 07 b8 ff ff ff ff 7d 02 5d c3 48 83
0000000100000fc0        00 00 7d 02 5d c3 48 83 c6 68 49 83 c0 68 48 89
0000000100001010        c3 48 83 c7 68 48 83 c6 68 5d e9 6b 35 00 00 55
0000000100001050        b8 01 00 00 00 7d 02 5d c3 48 83 c6 68 49 83 c0
00000001000010a0        7d 02 5d c3 48 83 c7 68 48 83 c6 68 5d e9 d8 34
00000001000010e0        48 7f 07 b8 01 00 00 00 7d 02 5d c3 48 83 c6 68
0000000100001120        7d 02 5d c3 48 83 c7 68 48 83 c6 68 5d e9 58 34
0000000100001150        b8 01 00 00 00 7d 02 5d c3 48 83 c6 68 48 83 c1
00000001000011a0        7d 02 5d c3 48 83 c7 68 48 83 c6 68 5d e9 d8 33
00000001000011e0        58 7f 07 b8 01 00 00 00 7d 02 5d c3 48 83 c6 68
0000000100001870        c0 09 c8 8a 0d ab 3c 00 00 89 c3 81 cb 80 00 00
0000000100001b70        5d d4 89 de e8 57 29 00 00 48 89 c3 48 85 db 0f
0000000100001c30        03 39 00 00 01 e9 52 01 00 00 0f b7 c0 83 f8 0d
0000000100001dd0        c3 48 8d 35 91 2d 00 00 eb 07 48 8d 35 c0 2d 00
0000000100001e20        36 0f b7 56 58 83 fa 07 75 02 5d c3 44 0f b7 c9
0000000100001ec0        00 48 8d 3d e2 2c 00 00 e8 21 26 00 00 48 89 c3
0000000100001f70        34 48 83 c3 02 80 f9 3a 75 19 80 7b fe 3a 75 13
0000000100001fa0        c3 84 c9 75 d0 44 89 b5 78 fb ff ff 45 89 e6 80
00000001000023b0        fb ff ff 74 5c 8b 78 74 e8 ef 20 00 00 48 89 c3
00000001000023e0        00 00 48 89 c3 48 85 db 0f 84 9a 04 00 00 48 89
0000000100002520        66 18 4d 8b 7e 20 41 8b 5e 30 48 63 c3 48 8d 34
0000000100002560        20 49 63 4e 30 41 89 04 8f 41 8b 5e 30 ff c3 41
0000000100002870        38 05 00 00 5b 41 5c 41 5d 41 5e 41 5f 5d c3 48
0000000100002970        c3 48 8d 3d 9e 22 00 00 48 8d 35 a1 22 00 00 48
0000000100002a30        ed 48 83 c3 68 48 89 df e8 4f 0b 00 00 89 c3 45
0000000100002a90        0f b7 7c 24 04 e8 28 0b 00 00 01 c3 89 d8 48 83
0000000100002aa0        c4 08 5b 41 5c 41 5d 41 5e 41 5f 5d c3 55 48 89
0000000100002c30        4f 28 48 8b 46 08 eb 0f 85 c0 45 8b 4f 38 48 8b
0000000100003200        00 48 83 c3 18 48 81 fb a8 01 00 00 75 84 bb 10
0000000100003260        45 89 fd 4c 8d bd b0 f7 ff ff 48 83 c3 18 48 83
00000001000032e0        5b 41 5c 41 5d 41 5e 41 5f 5d c3 48 8d 35 c5 18
0000000100003350        48 83 c4 08 5b 5d c3 48 8d 3d c6 1b 00 00 31 c0
00000001000034a0        c4 70 5b 41 5e 5d c3 e8 a0 0f 00 00 55 48 89 e5
0000000100003550        00 89 d8 48 83 c4 08 5b 5d c3 66 90 7e ff ff ff
00000001000035f0        00 00 00 5d c3 81 c1 00 60 00 00 81 e1 00 f0 00
00000001000036a0        75 06 48 83 c3 10 eb 69 48 8d 7b 68 e8 f7 0e 00
0000000100003740        5e 41 5f 5d c3 55 48 89 e5 41 57 41 56 41 55 41
0000000100003930        5c f0 ff ff 89 c3 48 8d 05 1b 1d 00 00 8b 08 85
0000000100003970        7c 04 85 c9 75 40 41 89 d4 89 c3 48 8d 05 b6 1c
0000000100003990        45 f8 e8 c3 0b 00 00 42 8d 04 2b 23 45 c8 44 89
00000001000039f0        83 c4 38 5b 41 5c 41 5d 41 5e 41 5f 5d c3 31 ff
0000000100003ac0        00 00 48 89 c3 8a 04 1a 88 45 d6 48 83 ca 01 48
0000000100003b00        f8 80 f9 30 75 36 83 c3 d0 41 89 1f 66 bb 01 00
0000000100003b40        9f 80 f9 07 77 08 83 c3 9f 41 89 1f eb 4e 89 c1
0000000100003b50        80 c1 bf 80 f9 07 77 12 83 c3 bf 41 89 1f 48 8b
0000000100003bd0        41 5d 41 5e 41 5f 5d c3 55 48 89 e5 41 56 53 41
0000000100003c30        c6 08 00 00 89 c7 44 89 f6 5b 41 5e 5d e9 e2 08
0000000100003c60        31 c0 48 83 c4 10 5d c3 55 48 89 e5 e8 e9 08 00
0000000100003c70        00 31 c0 5d c3 55 48 89 e5 41 56 53 89 f8 48 8d
0000000100003d10        5e 5d e9 b5 08 00 00 5b 41 5e 5d c3 55 48 89 e5
0000000100003e40        ff ff 4c 89 e6 4c 89 f9 e8 f5 06 00 00 48 89 c3
```

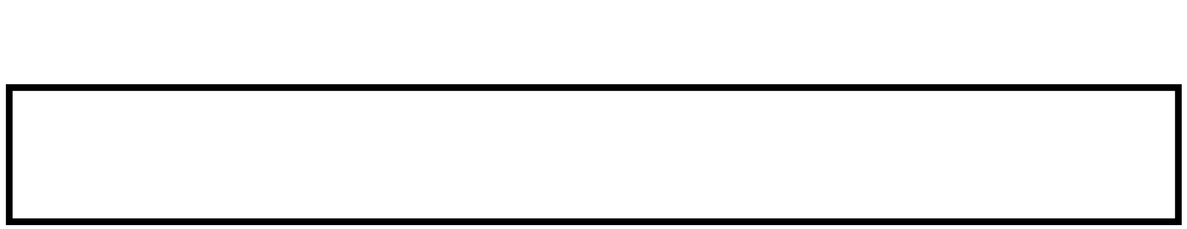
Gadgets Galore

- What is a *useful* gadget in this paper?
  - Anything that ends w/ **ret** and doesn't alter control flow is good (because we can set up the stack the way we like!)
- Why do these gadgets exist?
  - Compilers commonly add them at the end of a function! Very hard to avoid.
- Can build arbitrary new bad programs that are made completely out of "known good" instructions (e.g., libc)

Simple example

What does this piece of assembly do?

```
mov $5, %edx
```



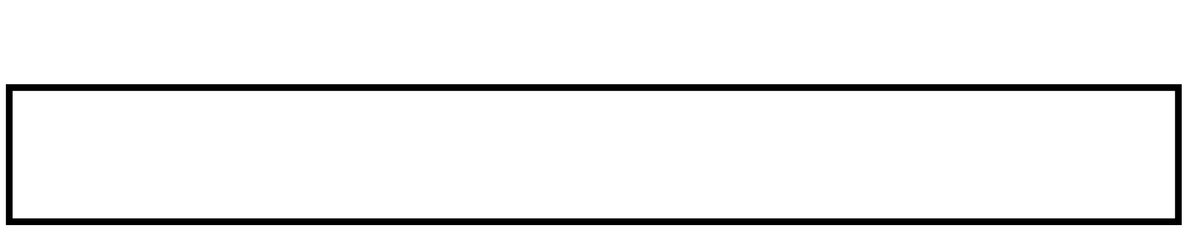
0x00 0xffffffff

stolen w/ love from UMD

Simple example

What does this piece of assembly do?

```
mov $5, %edx
```

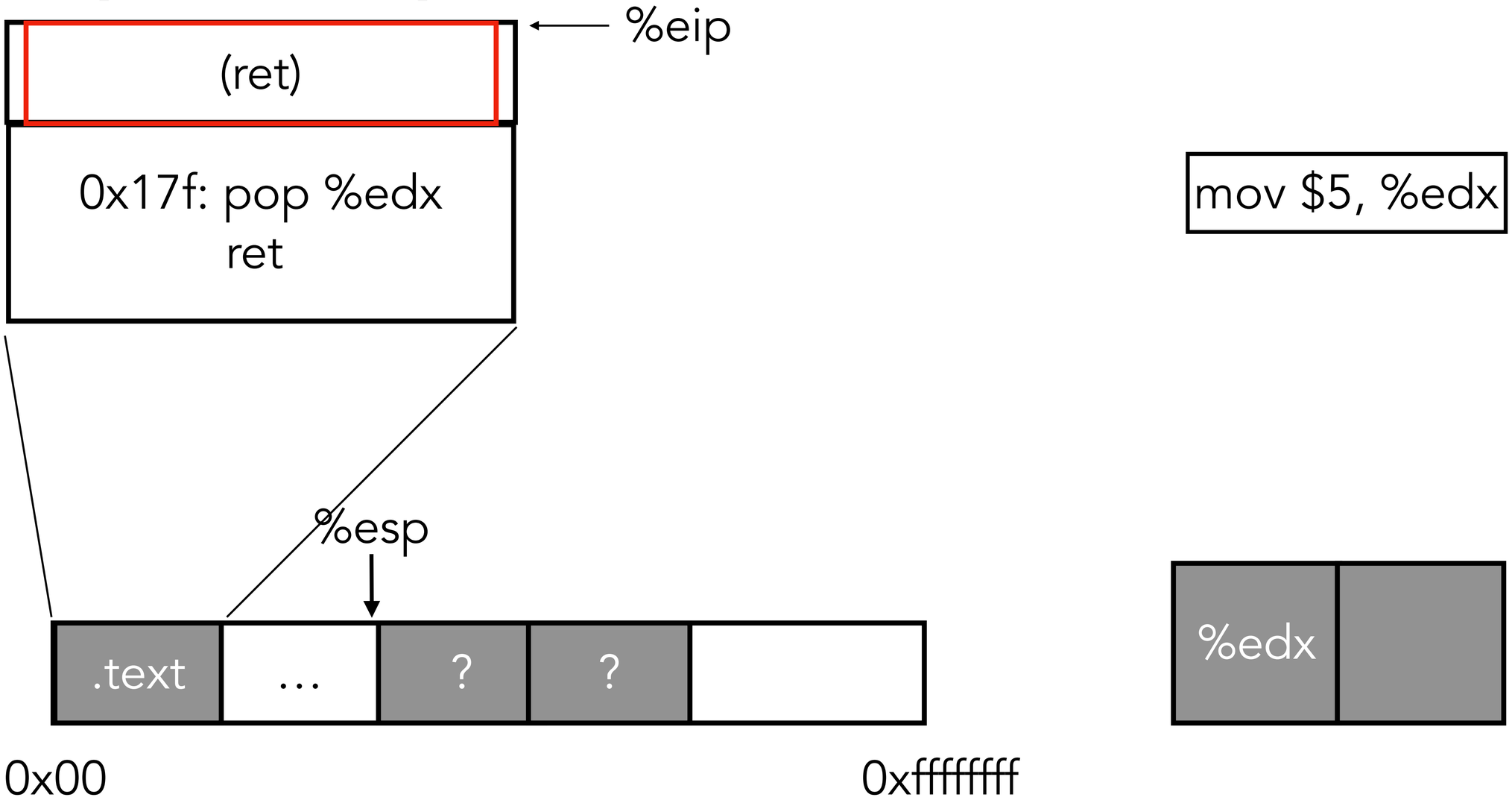


0x00

0xffffffff

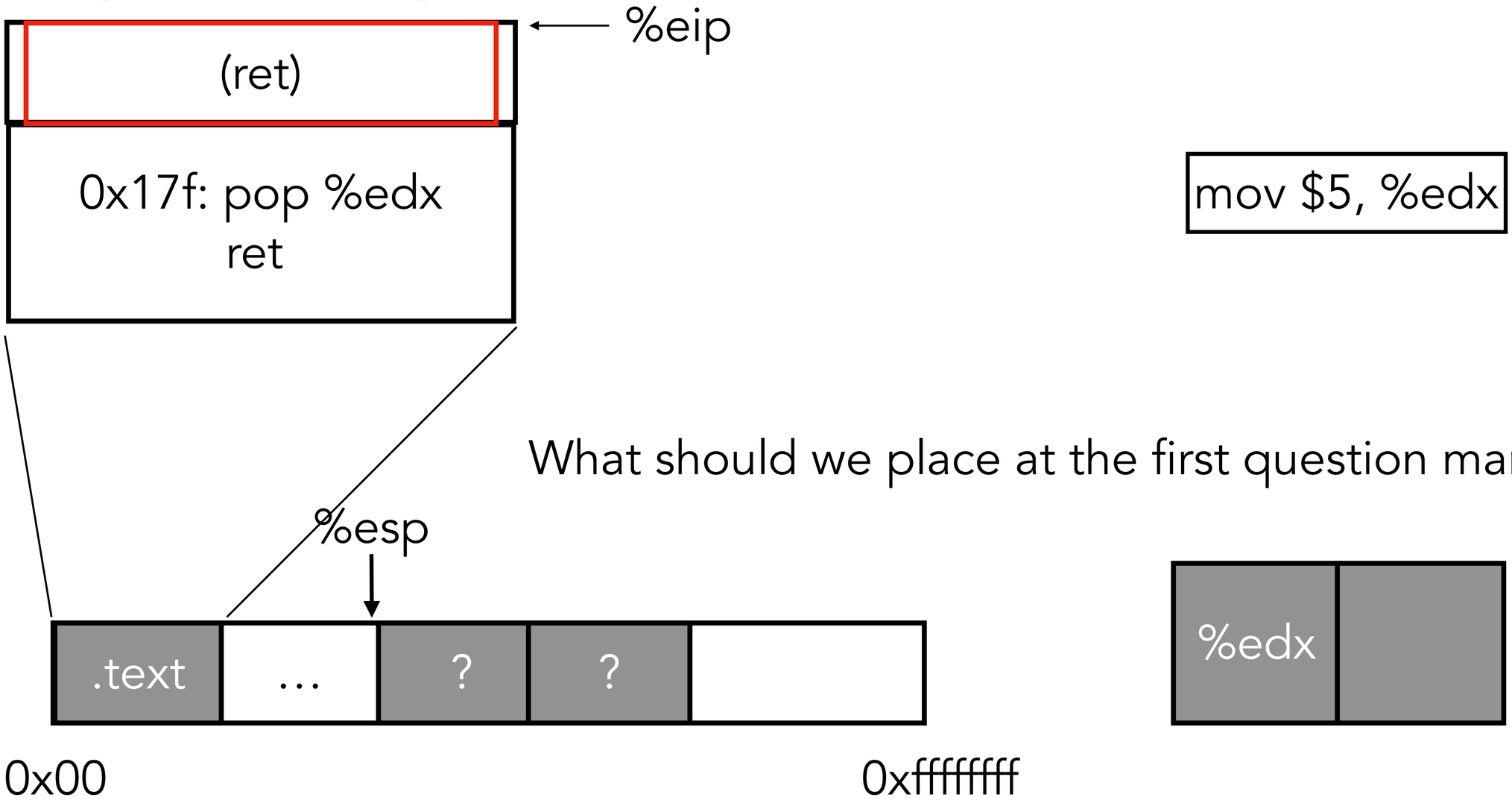
stolen w/ love from UMD

Simple example



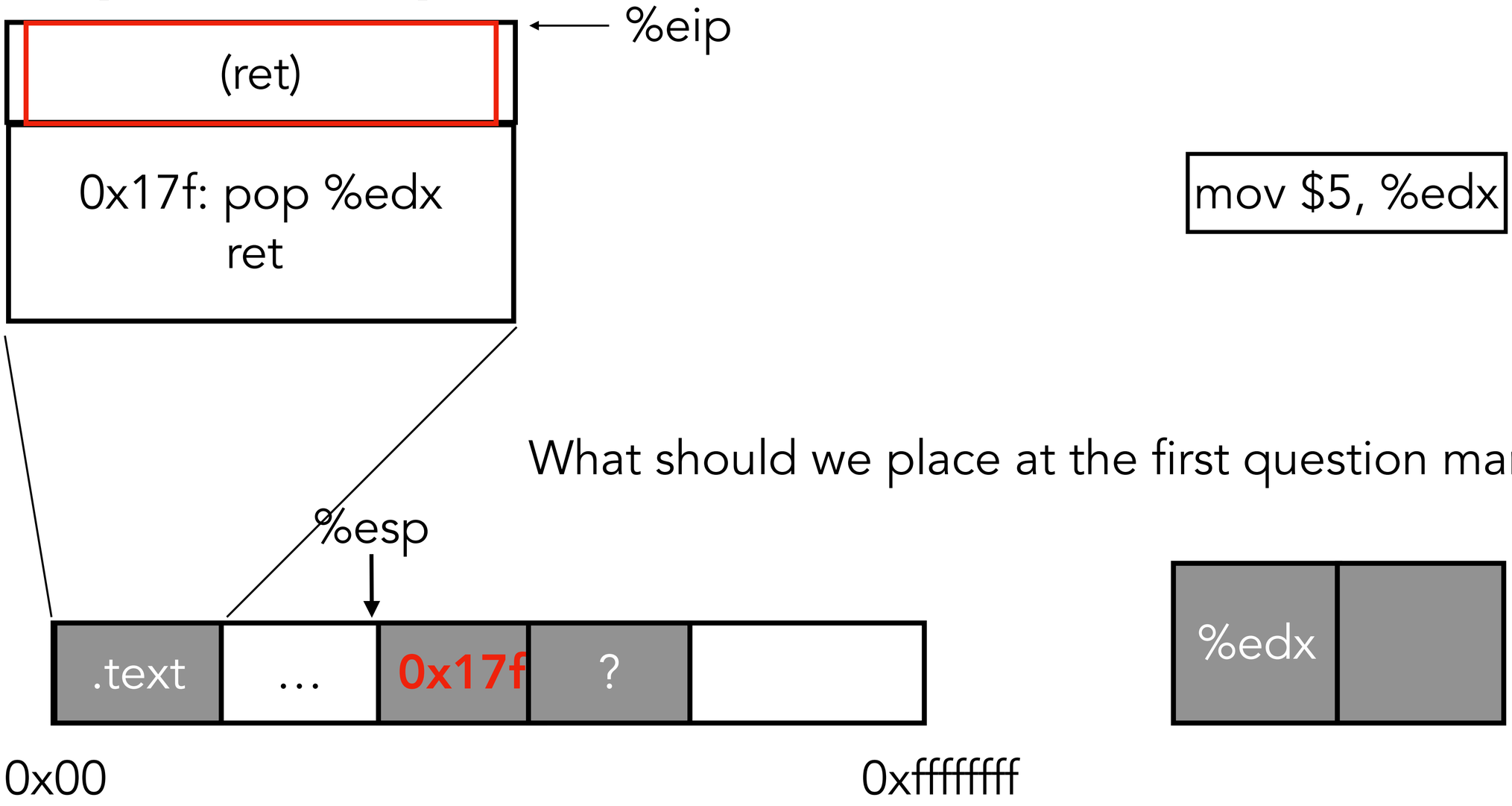
stolen w/ love from UMD

Simple example



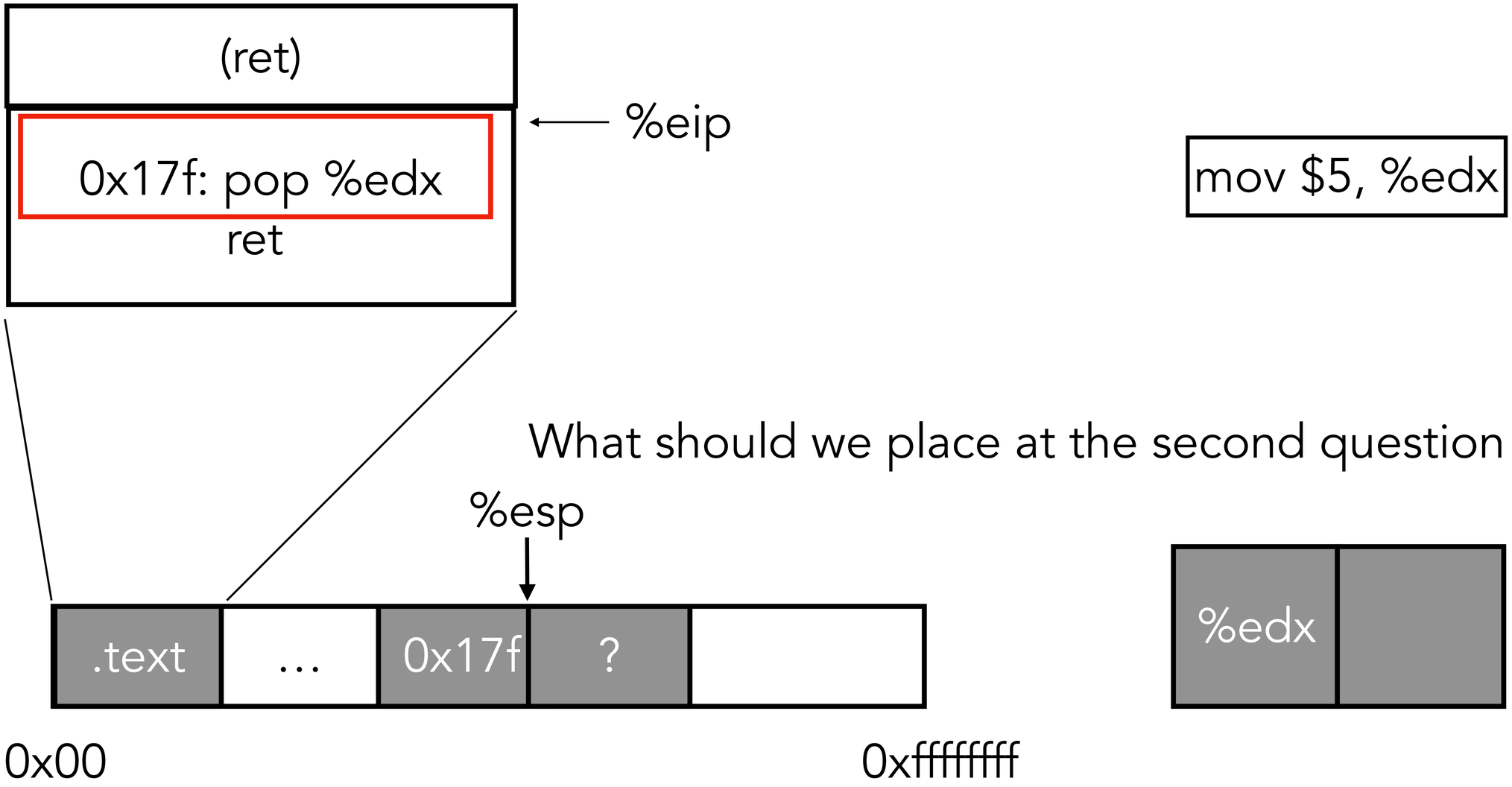
stolen w/ love from UMD

Simple example



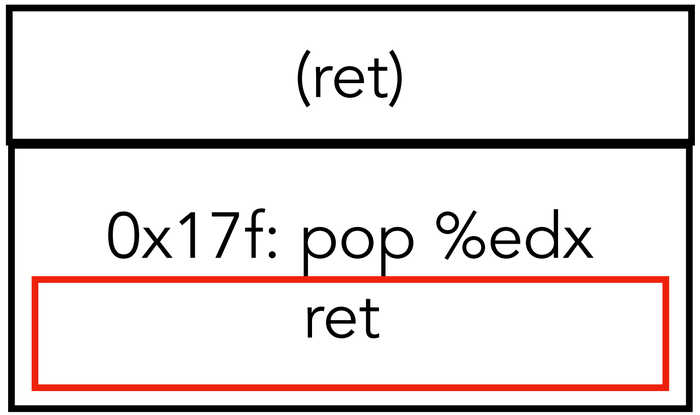
stolen w/ love from UMD

Simple example



stolen w/ love from UMD

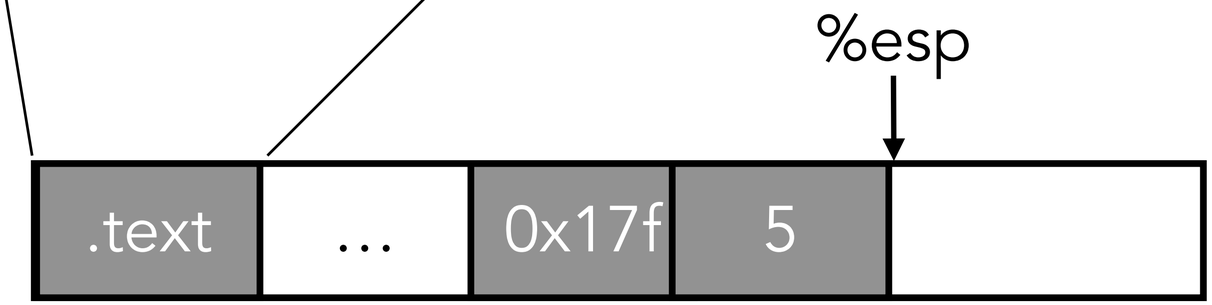
Simple example



%eip

mov $5, %edx

What should we place at the second question mark?

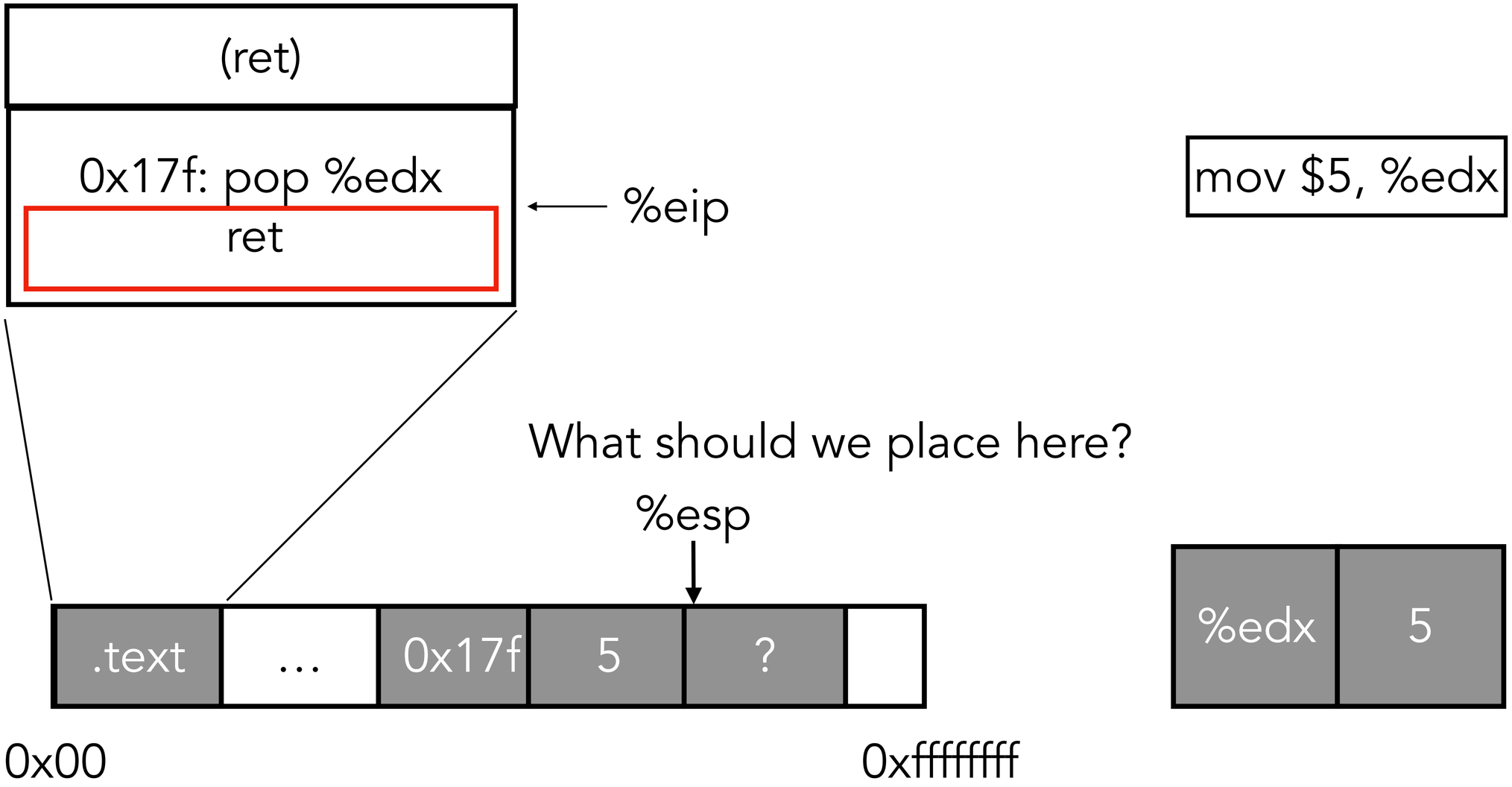


0x00

0xffffffff

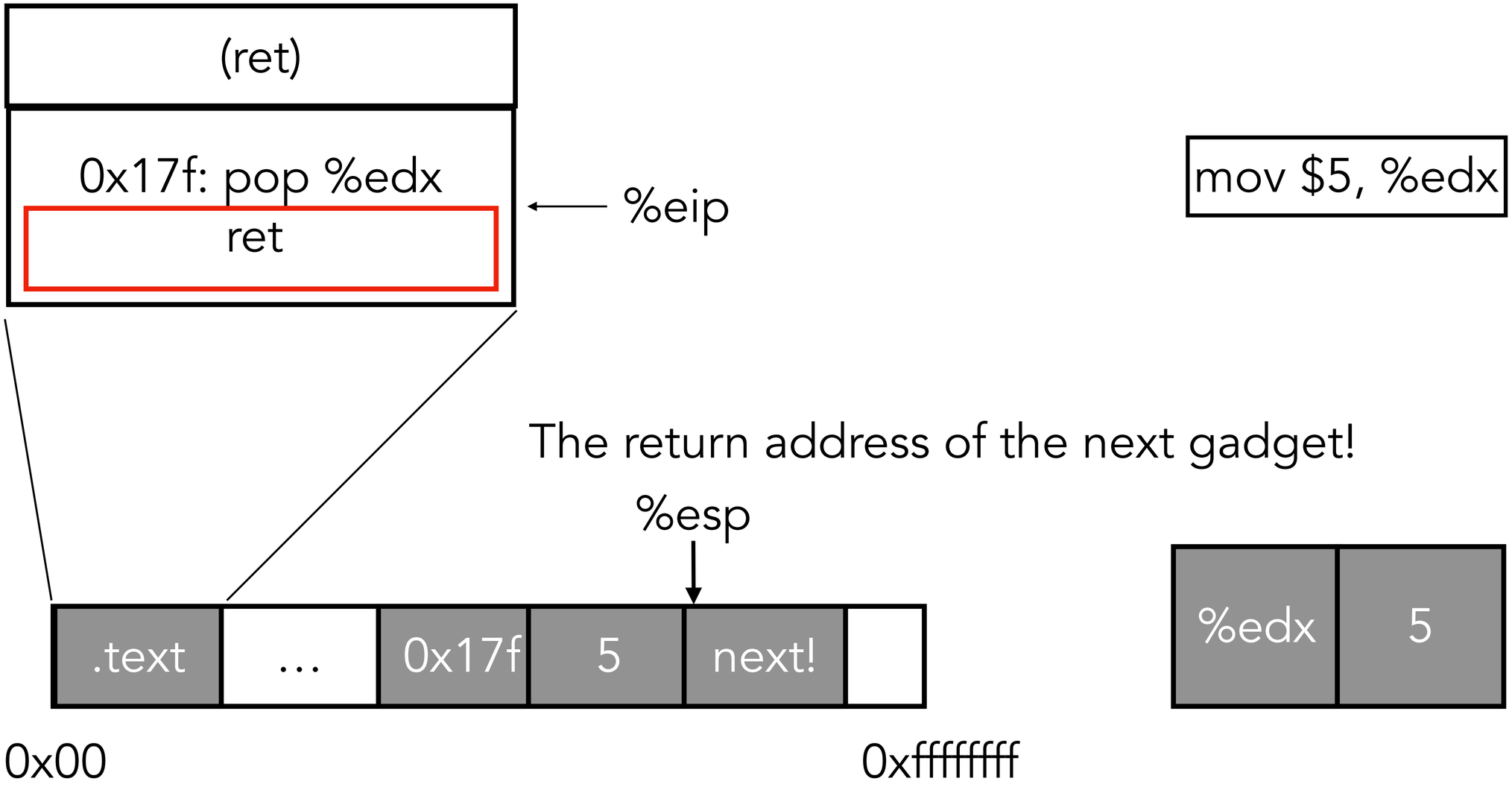
stolen w/ love from UMD

Simple example



stolen w/ love from UMD

Simple example



stolen w/ love from UMD

Making ROP Hard

- What are some assumptions made about the *location* of libc functions that make ROP possible?

Making ROP Hard

- What are some assumptions made about the *location* of libc functions that make ROP possible?
  - libc is in a fixed location: **not true with Address Space Layout Randomization (ASLR)**

Making ROP Hard

- What are some assumptions made about the *location* of libc functions that make ROP possible?
  - libc is in a fixed location: **not true with Address Space Layout Randomization (ASLR)**
- Control flow integrity (CFI)
  - Check at run-time if the execution path is allowed by the original program
  - Insert “tags” before each branch target when branching, and first check the target’s tag matches expectation


Return-Oriented Programming
is a lot like a ransom note. But instead of cutting cut letters from magazines, you are cutting out instructions from .text segments

What about these attacks *surprised* you?

What do these attacks teach us about *trust?*

Break Time + Attendance

https://tinyurl.com/cse227-attend

Codeword:
ROP-is-Hard

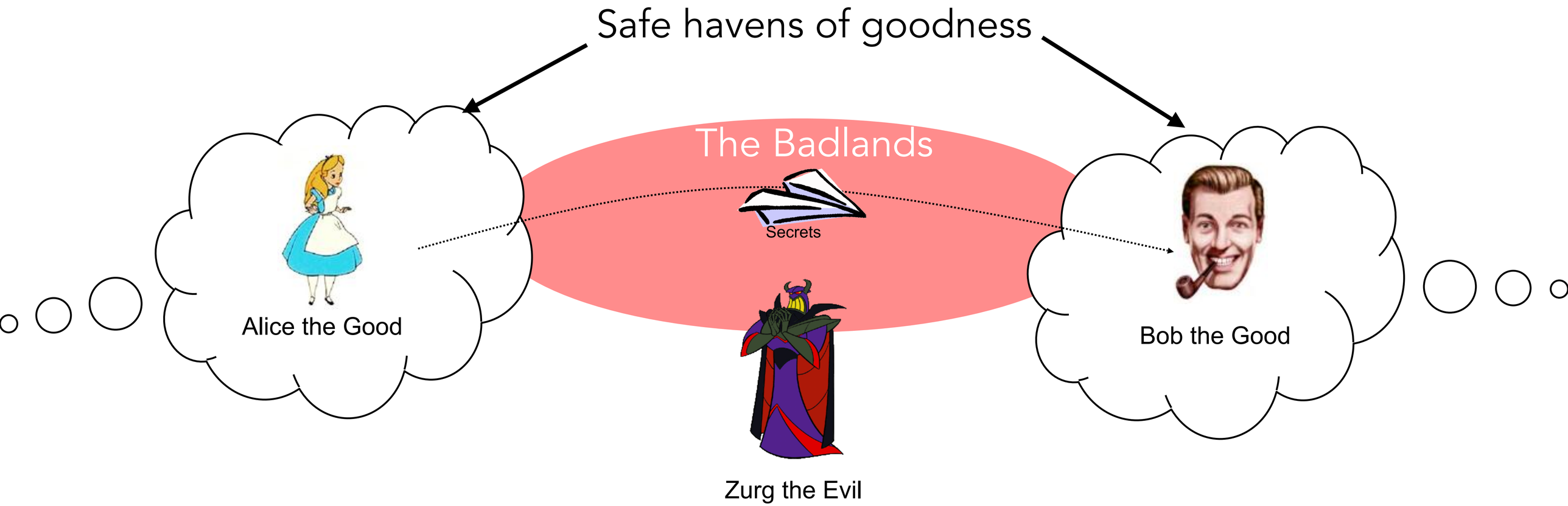
Side Channels

Learning Objectives

- Learn what a side channel is, why side channels exist, and how side channels gets implemented or exploited in practice
- Walk through some modern versions of side channels and understand how to find them in the wild
- Discuss the “keyboard emanations” attack
- Discuss the “Cold boot” attack on DRAM

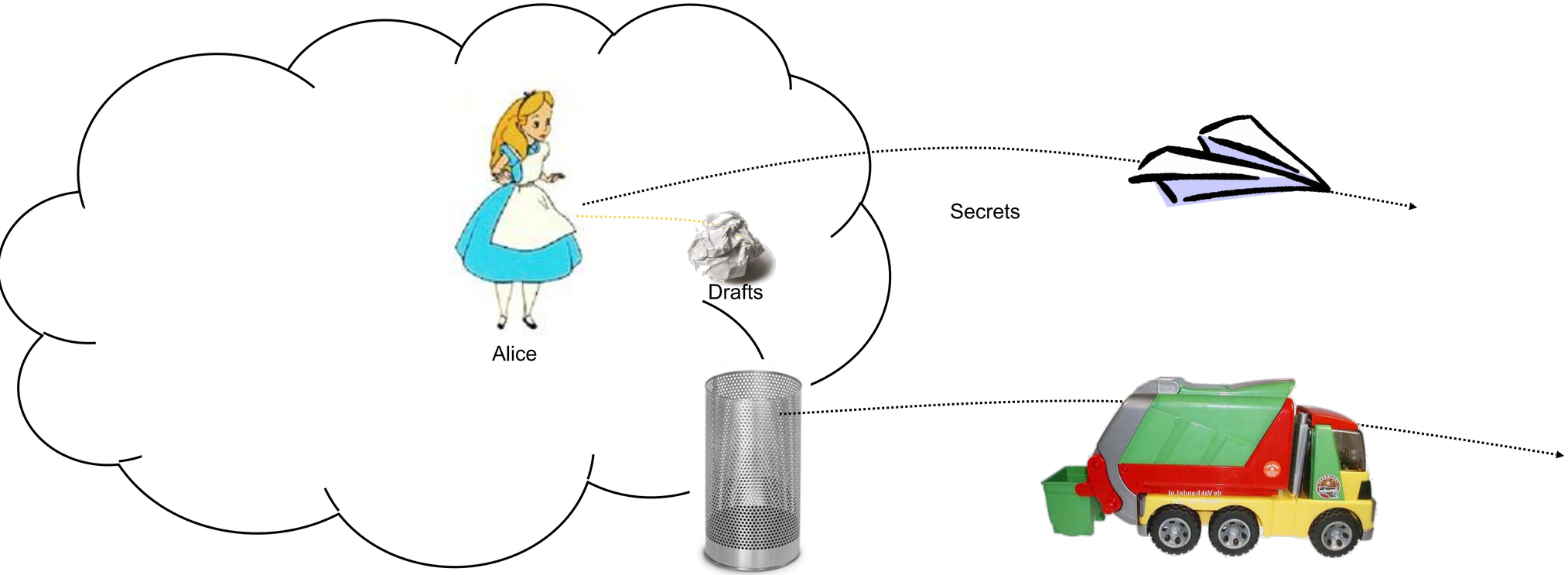
Preliminaries

Our typical model of security



Things are more complicated in practice…

Side channel: Secrets may leak *outside* the protocol because of how its implemented in practice



Side Channel Scenario

Figure 2. The basic setting: The monitor faces away from the window in an attempt to hide the screen's content.

From: Compromising Reflections – or – How to Read LCD Monitors Around the Corner

How might you read the computer screen from the window?

Figure 2. The basic setting: The monitor faces away from the window in an attempt to hide the screen's content.

From: Compromising Reflections – or – How to Read LCD Monitors Around the Corner

Reflections are your friend!

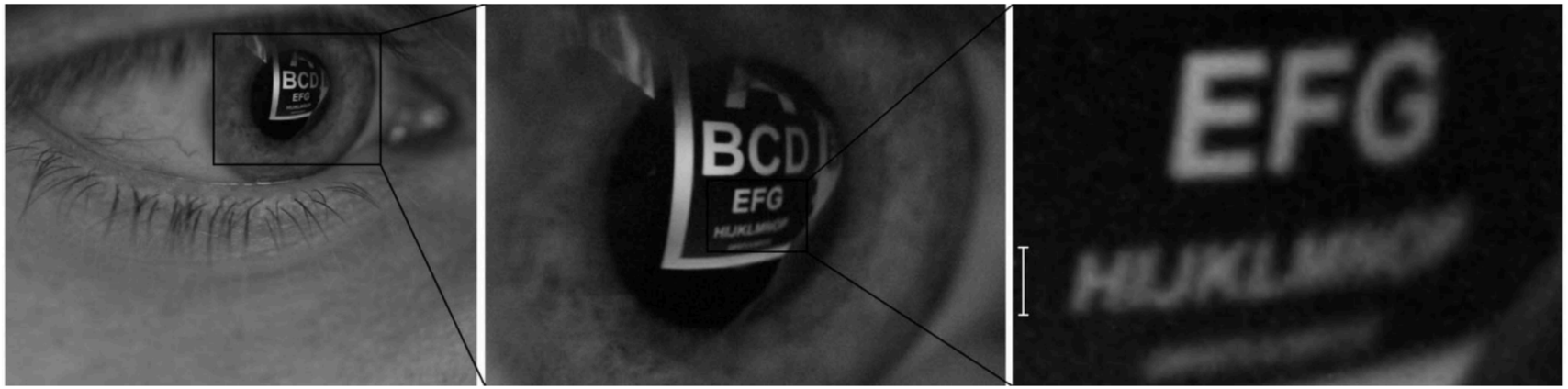


Figure 1. Image taken with a macro lens from short distance; the distance between the eye and the monitor was reduced for demonstration. Readability is essentially limited by the camera resolution.

From: Compromising Reflections – or – How to Read LCD Monitors Around the Corner

Reflections are your friend!

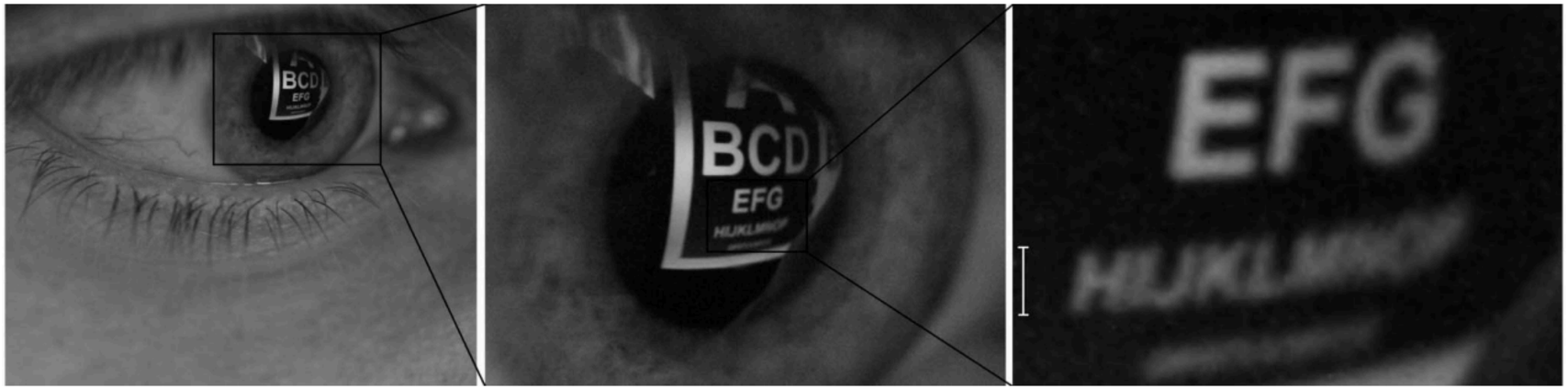


Figure 1. Image taken with a macro lens from short distance; the distance between the eye and the monitor was reduced for demonstration. Readability is essentially limited by the camera resolution.

Figure 5. Reflections in a tea pot, taken from a distance of 10m. The 18pt font is readable from the reflection.

The craziest reflection of them all…

Figure 12. Reflections in a 0.5l plastic Coca-Cola bottle, taken from a distance of 5m. Because of the irregular surface, only parts of the text are readable.

From: Compromising Reflections – or – How to Read LCD Monitors Around the Corner

Keyboard Acoustic Emanations

What is an acoustic emanation?

What is an acoustic emanation?

Acoustic emanation: A sound made by an object in the course of normal use of that object

Keyboard acoustic emanations

What is the attack the authors want to conduct?

Keyboard acoustic emanations

What is the attack the authors want to conduct?

Keyboard acoustic emanations

What is the attack the authors want to conduct?

Covertly or overtly record keystrokes

Keyboard acoustic emanations

What is the attack the authors want to conduct?

Attacker can use keystrokes to recover passwords or other secrets

Keyboard acoustic emanations

What is the attack the authors want to conduct?

What makes the attack possible? What's the side channel here?

Keyboard acoustic emanations

What is the attack the authors want to conduct?

What makes the attack possible? What's the side channel here?

Why do keystrokes make different sounds?

Keyboard acoustic emanations

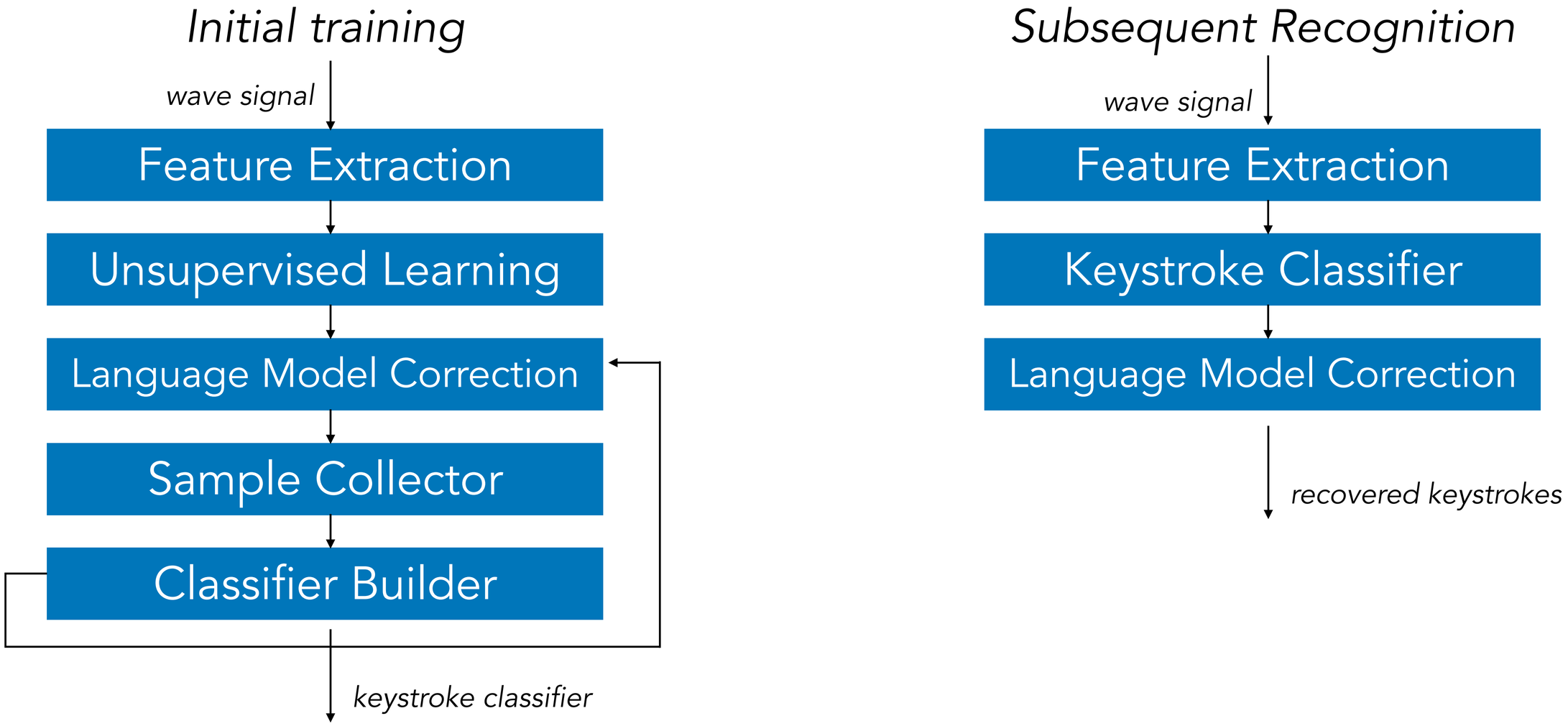
What is the attack the authors want to conduct?

What makes the attack possible? What's the side channel here?

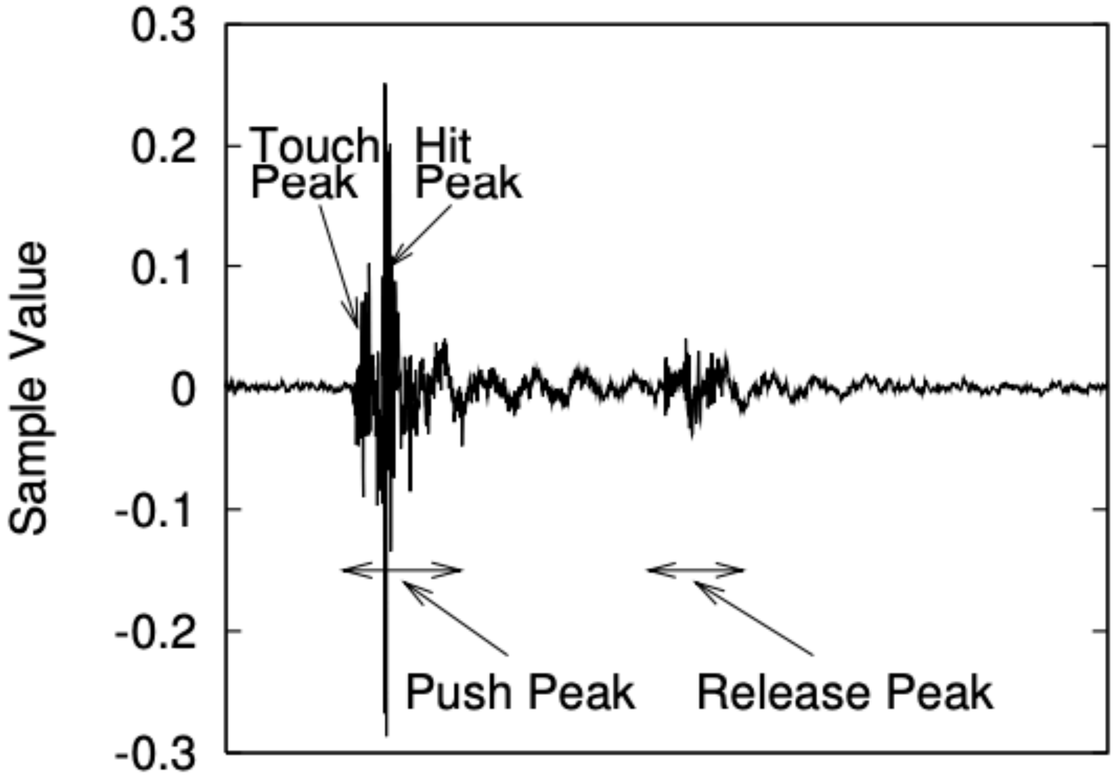
Why do keystrokes make different sounds?

What capabilities (think threat model) are required of the attacker?

The attack flow

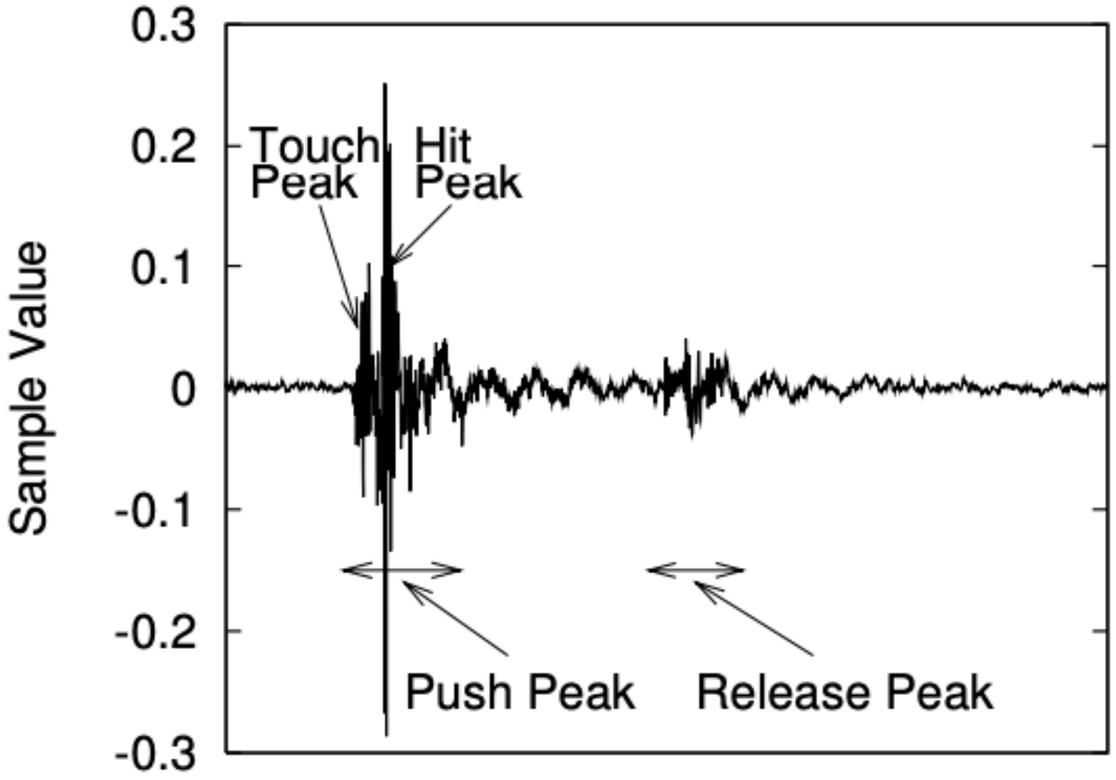


Extracting Keystrokes



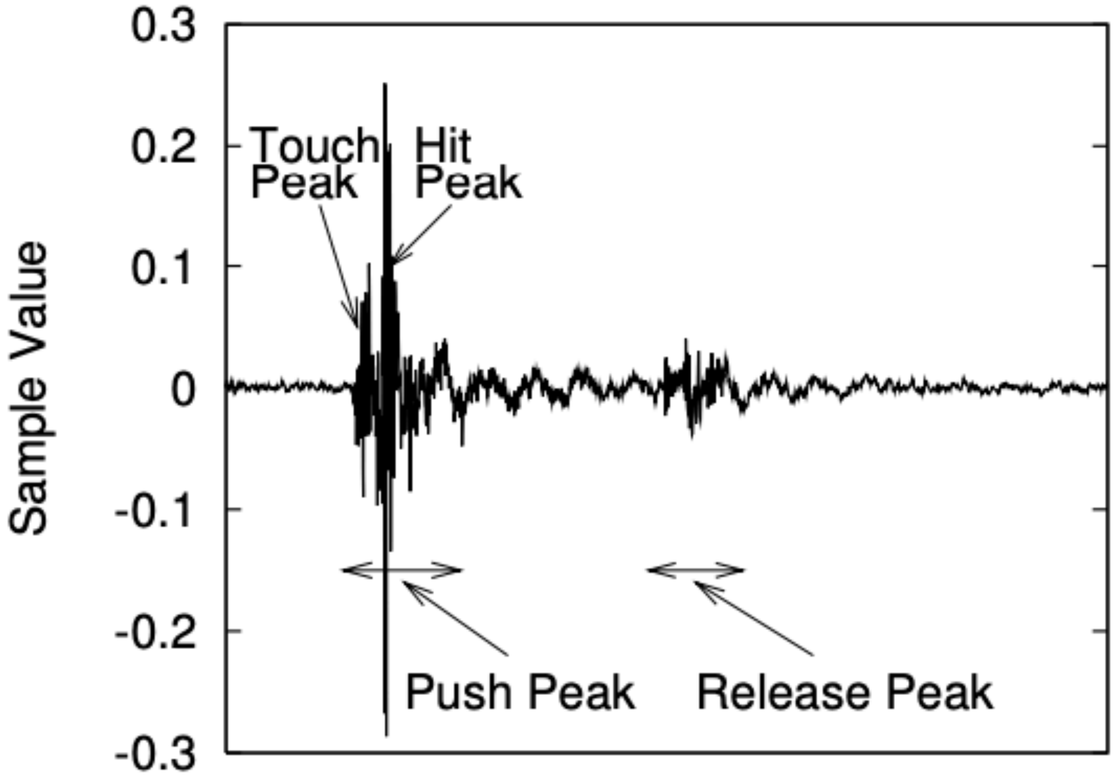
Extracting Keystrokes

- What is the touch peak?



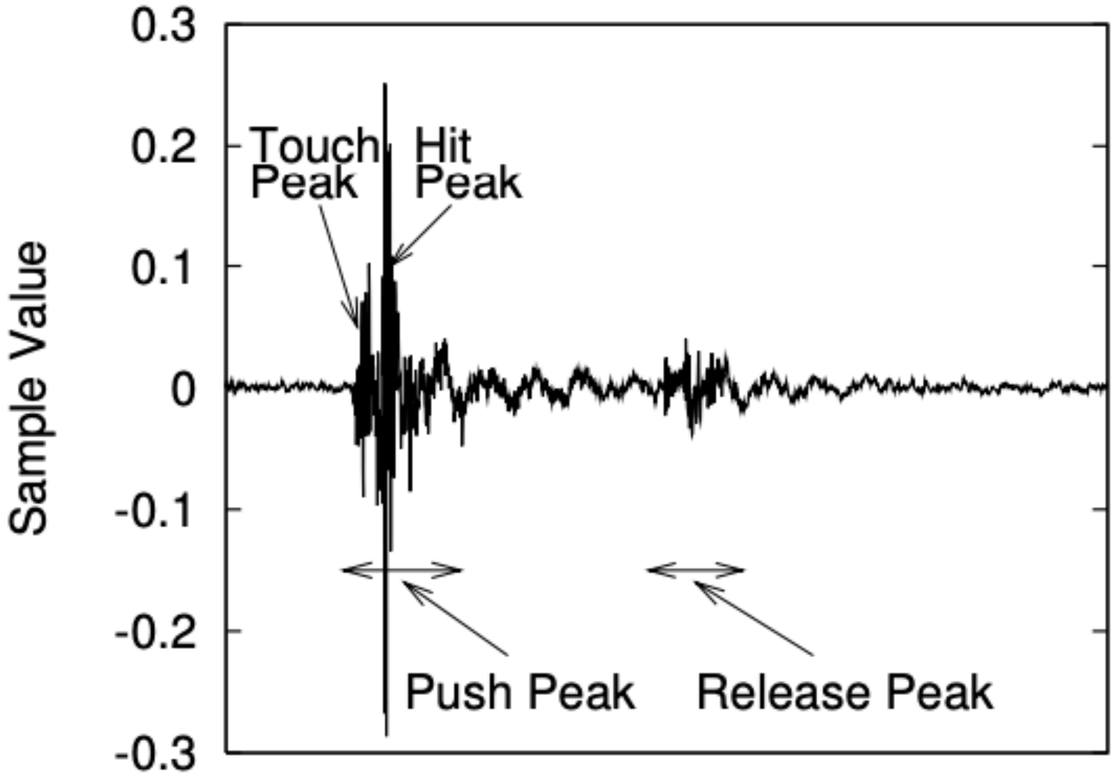
Extracting Keystrokes

- What is the touch peak?
- What is the hit peak?



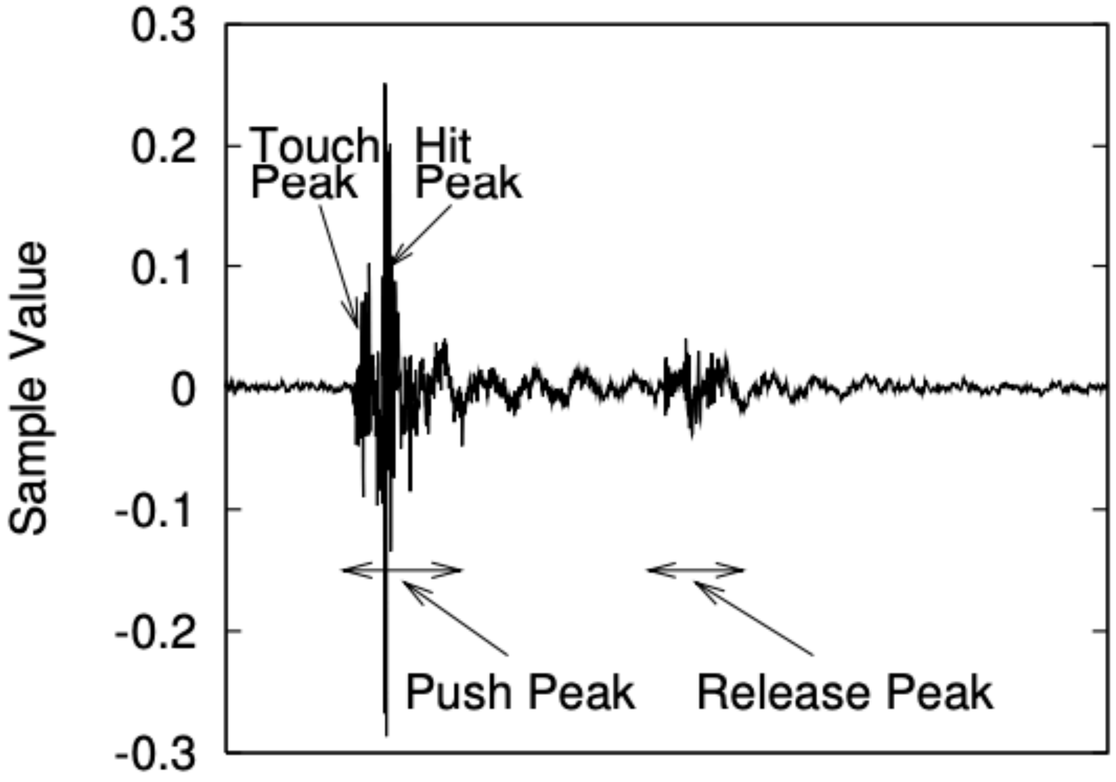
Extracting Keystrokes

- What is the touch peak?
- What is the hit peak?
- What is the release peak?



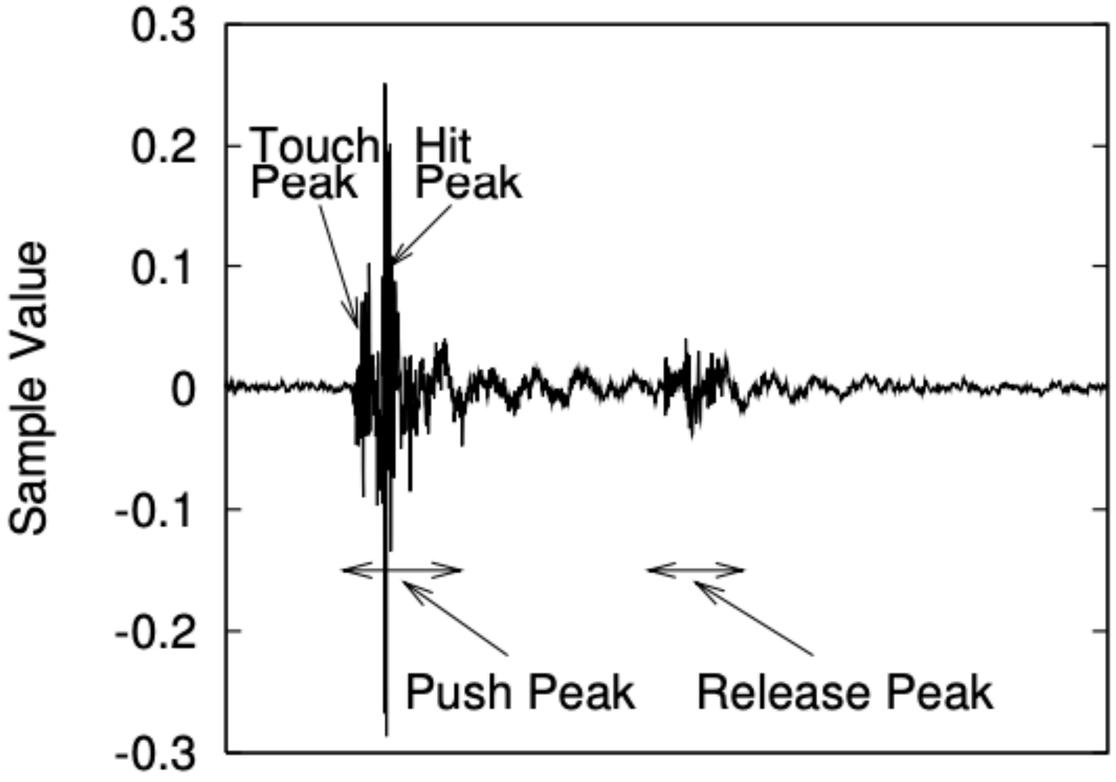
Extracting Keystrokes

- What is the touch peak?
- What is the hit peak?
- What is the release peak?
- How much time from push to release, on average?



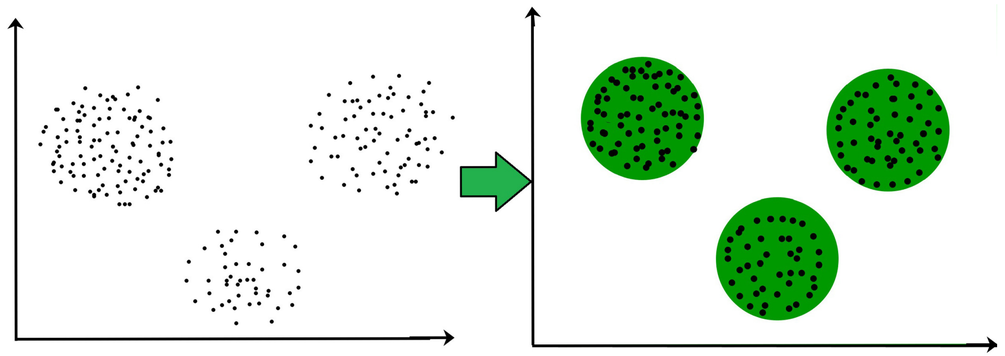
Extracting Keystrokes

- What is the touch peak?
- What is the hit peak?
- What is the release peak?
- How much time from push to release, on average?
  - **100 milliseconds… enough for a computer to discern!**



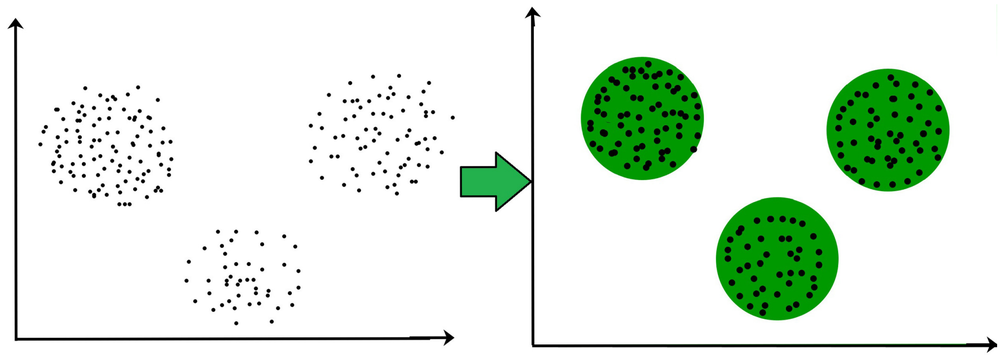
Unsupervised Learning

- What is unsupervised learning?



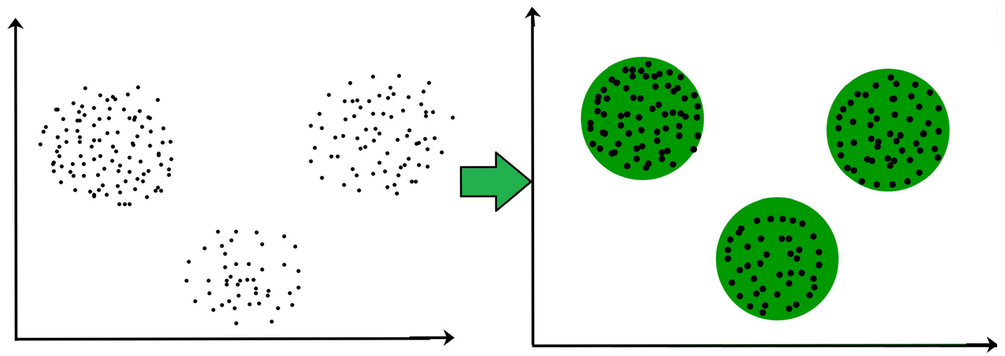
Unsupervised Learning

- What is unsupervised learning?
- What is clustering?



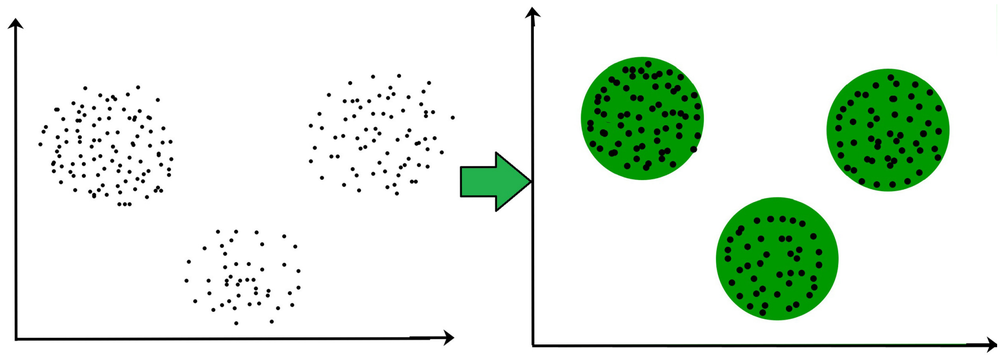
Unsupervised Learning

- What is unsupervised learning?
- What is clustering?
- How do the authors use unsupervised learning in their paper?



Unsupervised Learning

- What is unsupervised learning?
- What is clustering?
- How do the authors use unsupervised learning in their paper?
- Why is clustering *hard* in this context?



Recovering text from clusters

- How do the authors map clusters to letters?

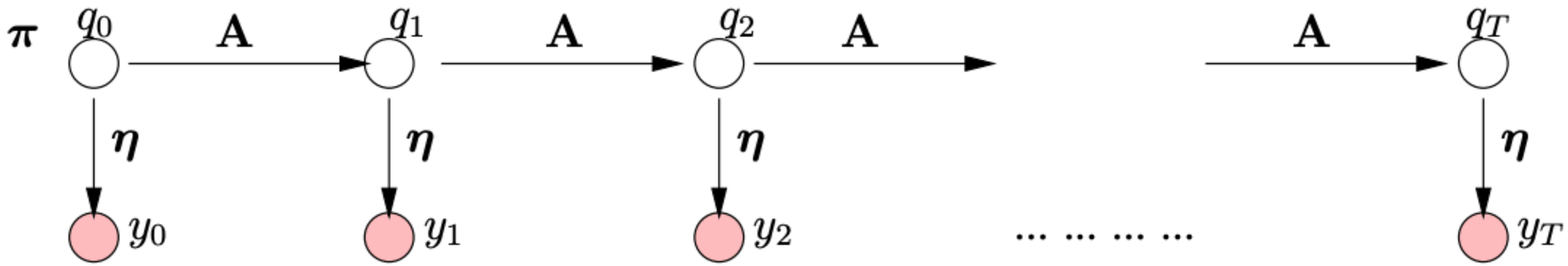
Recovering text from clusters

- How do the authors map clusters to letters?
- What is a Hidden Markov Model?

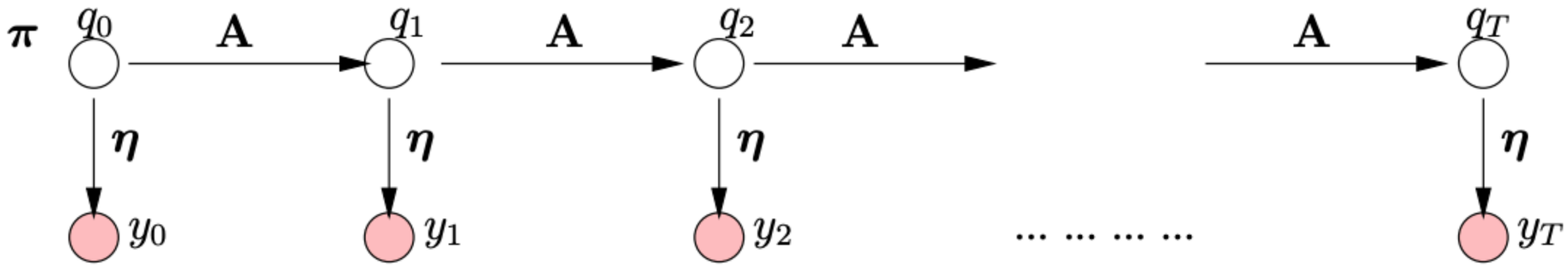
Recovering text from clusters

- How do the authors map clusters to letters?
- What is a Hidden Markov Model?
  - Authors “embed” English probabilities here – **th** is much more likely than **tj**

Bi-grams of characters

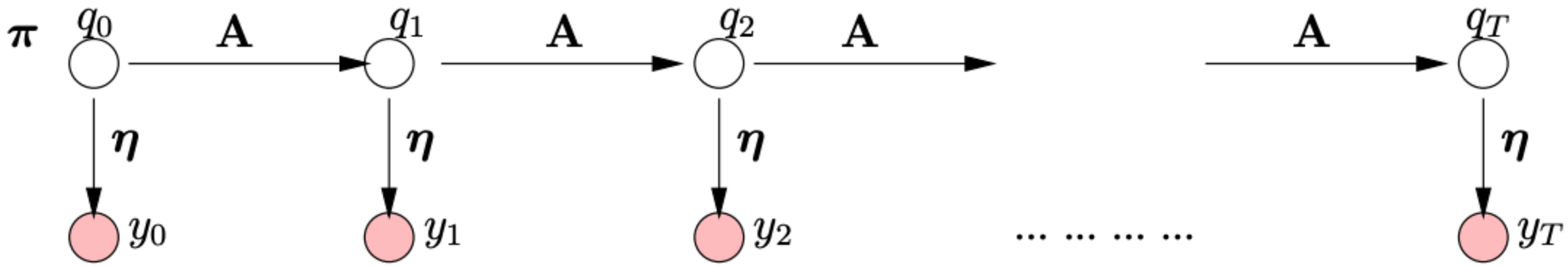


Bi-grams of characters



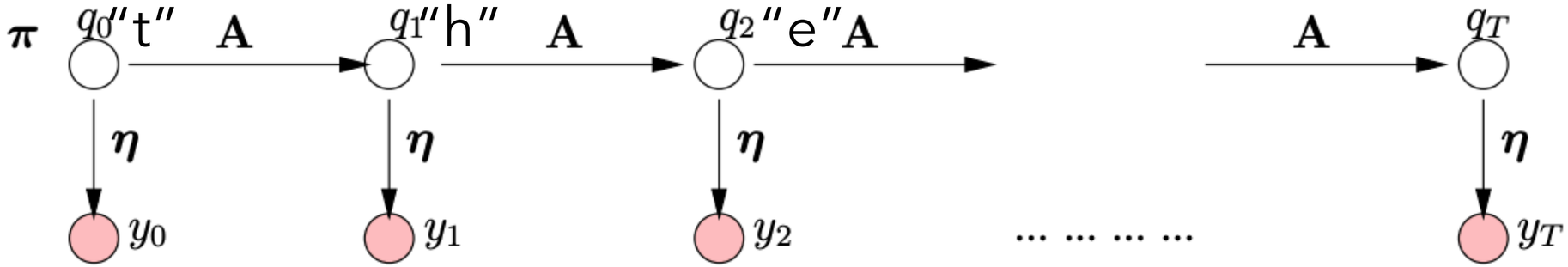
- What do the circles represent in the model? What are shaded and unshaded?

Bi-grams of characters



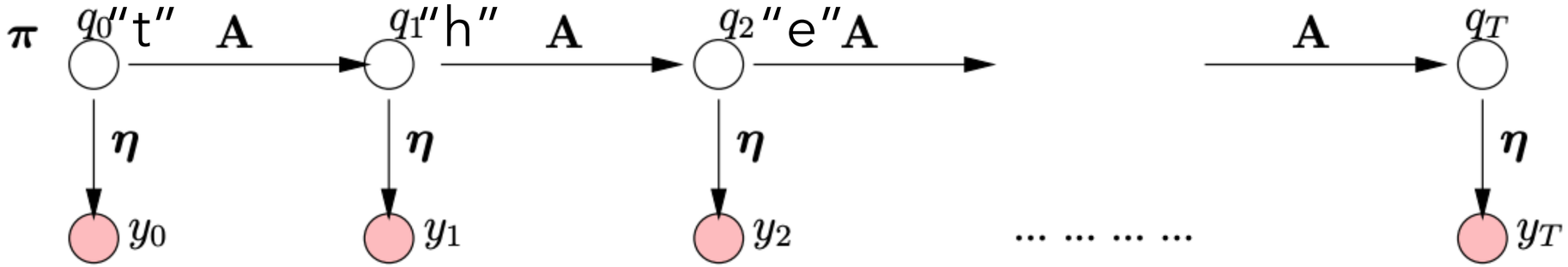
- What do the circles represent in the model? What are shaded and unshaded?
- What is *q*?

Bi-grams of characters



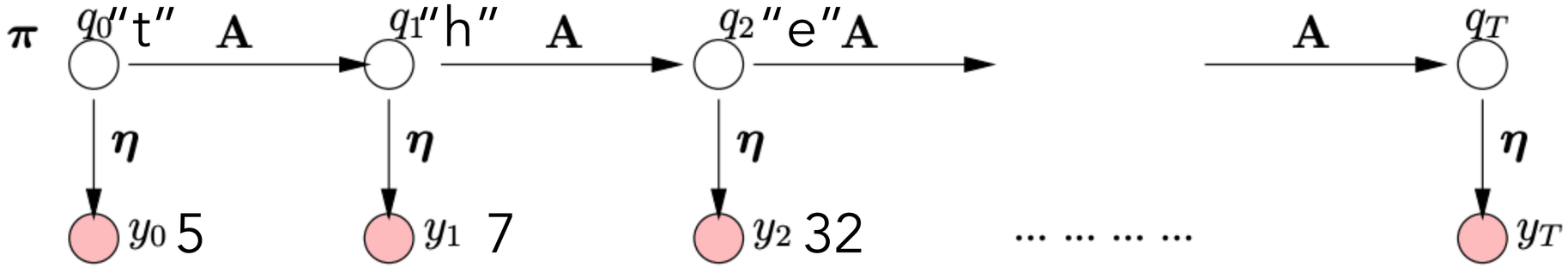
- What do the circles represent in the model? What are shaded and unshaded?
- What is *q*? ***Characters pressed***

Bi-grams of characters



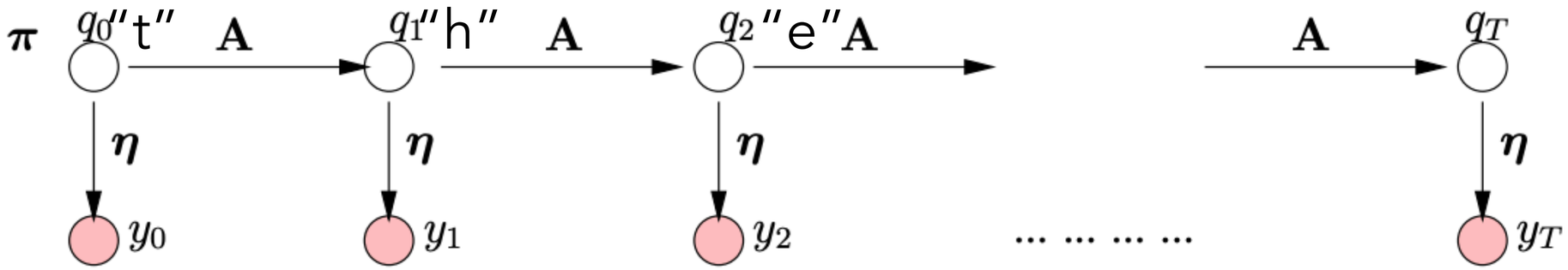
- What do the circles represent in the model? What are shaded and unshaded?
- What is *q*? ***Characters pressed***
- What is *y*?

Bi-grams of characters



- What do the circles represent in the model? What are shaded and unshaded?
- What is *q*? ***Characters pressed***
- What is *y*? ***Cluster labels***

Bi-grams of characters



- What is **A**, otherwise known as the *transition matrix?*
- How do we populate **A?**
- What is *eta*?
- How do the authors populate *eta*?

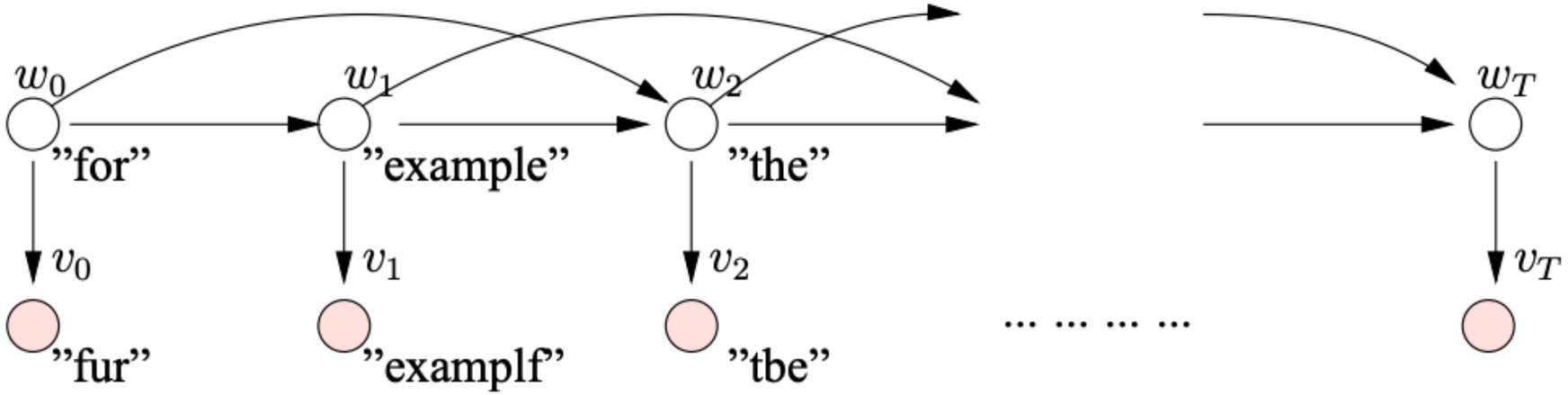
So now we have characters…. are they right?

- Authors used *spellcheck* to try and make the text more readable, but it still made mistakes
  - e.g., “fur example” vs. “for example”
- Can get more readable text using an n-gram language model
  - What’s an n-gram language model?

So now we have characters…. are they right?

- Authors used *spellcheck* to try and make the text more readable, but it still made mistakes
  - e.g., “fur example” vs. “for example”
- Can get more readable text using an n-gram language model
  - What’s an n-gram language model?
- n-gram: sequence of *n* adjacent items in text, speech, genomes, etc.

Tri-grams of words (HMMs to the rescue, again)



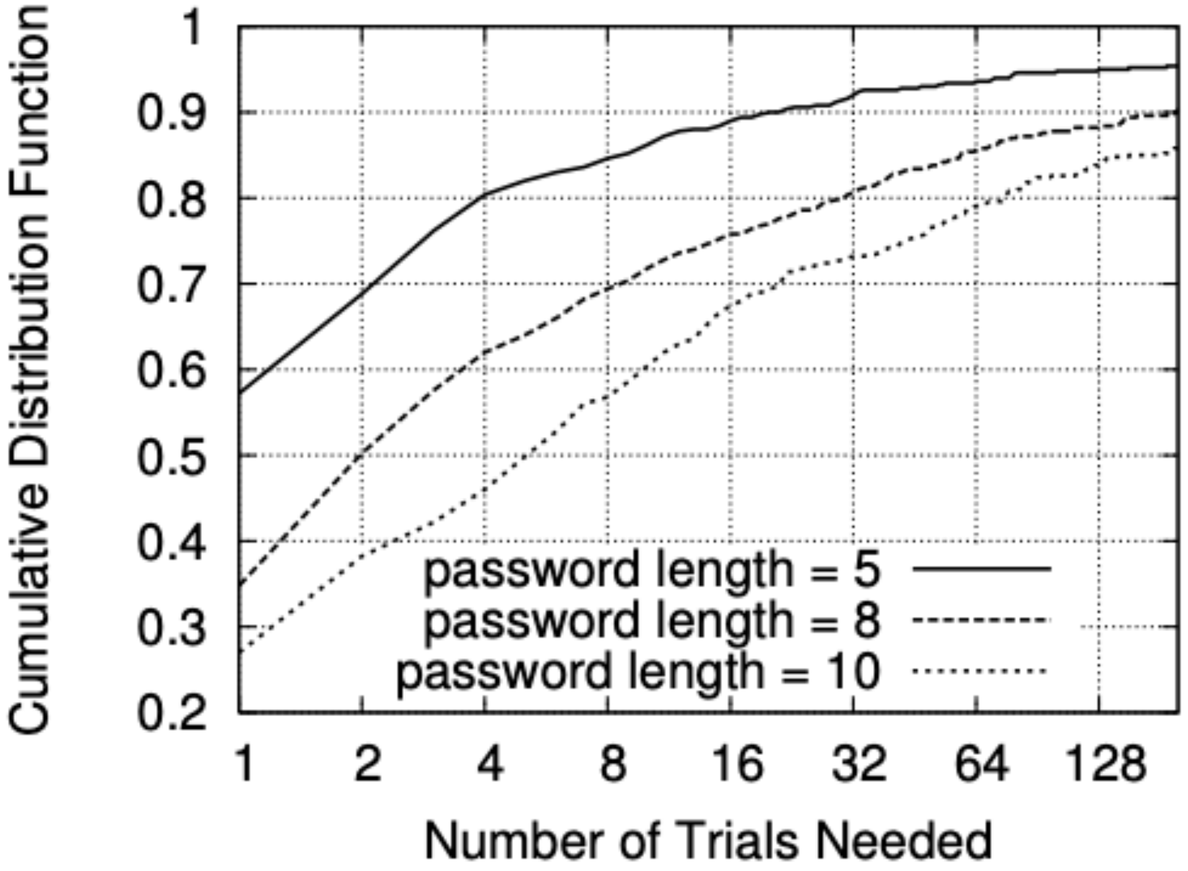
- Hidden variables are the original words
- This HMM depends on *two* layers (previous word **and** previous previous word)
- This is a great strategy if you have **unknowns** you want to predict from a known distribution!

Evaluation

- What was the evaluation setup for the attack?
- How many environments did the authors test their attack in?
- Are all keyboards vulnerable to this kind of attack? How did the authors evaluate this?

So it works on english, does it work on passwords?

- Yes!
- Authors found that they could recover 90% of 5-character passwords, 77% of 8-character passwords, and 69% of 10-character passwords
- Probabilities form a “hit list” of potential passwords to try



5-minute discussion: Meta points

- How feasible is this attack?
- Do you believe this attack will work in practice? Why or why not?
- What do we think about side channel research?

Side channels can be even crazier….

RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis