Side-Channel Attack

CS463/ECE424

University of Illinois

Side Channel Attacks: Two Case Studie

- Keyboard spy via acoustic side channels
- Information leakage via hardware side channels

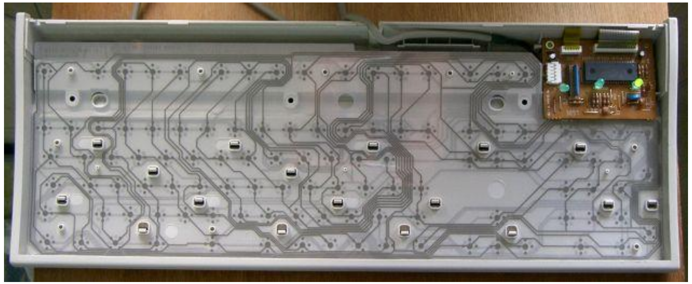
Extracting Information from Side Channels

- Inferring words typed on the keyboard by analyzing the sound

Keyboard Acoustic Emanations Revisited, Li Zhuang, Feng Zhou, J. D. Tygar, CCS 2005

Intuition: Why could this possibly work?

- Different keystrokes make different sounds
  – Locations
  – Underlying hardware



Threat Model and Challenges

- Attacker has a microphone recording the victim's typing
  - **Assumptions**: typing English text, **no labeled input**
  - **Goals:** recovering the English text, **inferring random text** (e.g., password)
- Challenges
  - Hard to obtain labeled training data --- no cooperation from the victim
  - Typing patterns can be keyboard specific
  - Typing patterns can be user specific

Threat Model and Challenges

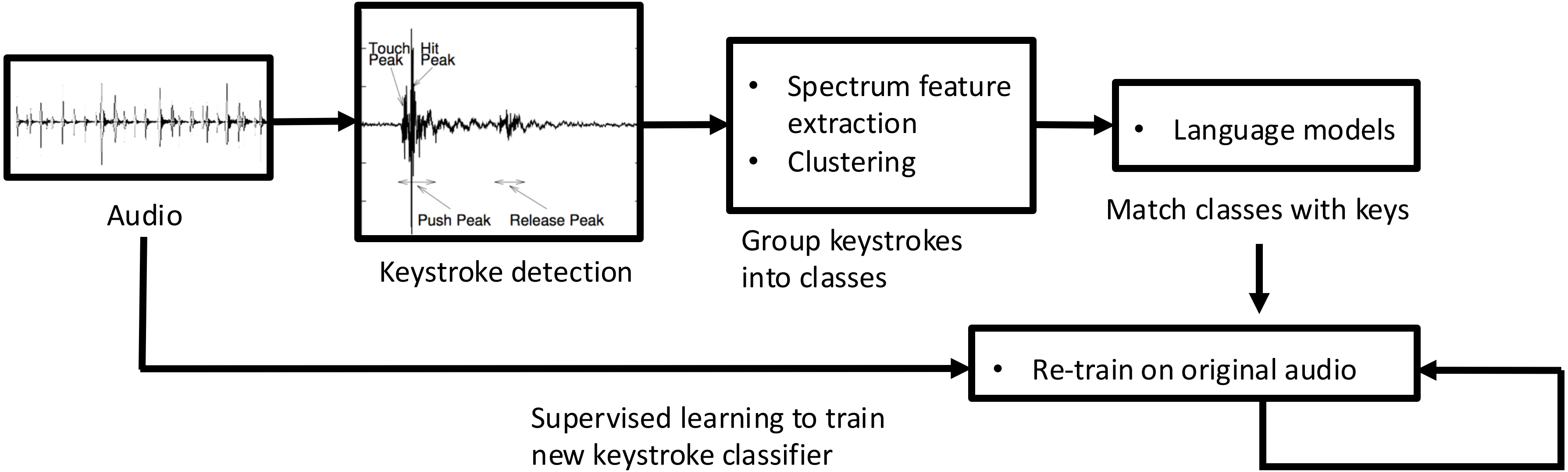
- Attacker has a microphone recording the victim's typing
  - **Assumptions**: typing English text, **no labeled input**
  - **Goals:** recovering the English text, **inferring random text** (e.g., password)
- Challenges

Key Intuition: the typed text is often not random.

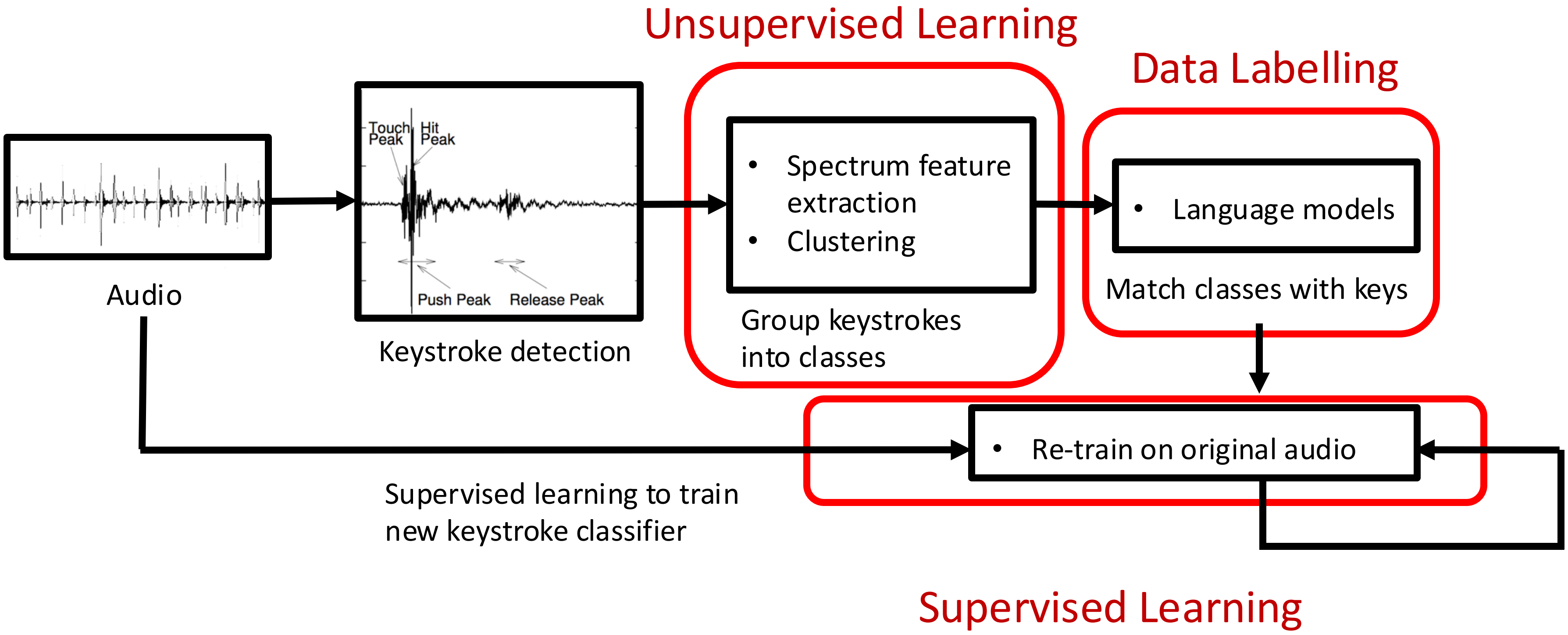
- English words limits the possible temporal combinations of keys
- English grammar limits the word combinations.

How The Attack Works

- Key idea: generating training data automatically
  - Labelling the audio of a key stroke with the actual key



A Combination of Different Learning Methods



Step1: Unsupervised Learning

- Unsupervised clustering
  - Feature generation
    - Cepstrum features
  - Clustering into K classes
    - K > N (actual number of keys used)
- Output
  - K **unlabeled** classes

- Spectrum feature extraction
- Clustering

Group keystrokes into classes

this is the best pizza in town

this is the best pizza in town

Step 2: Context-based Language Model

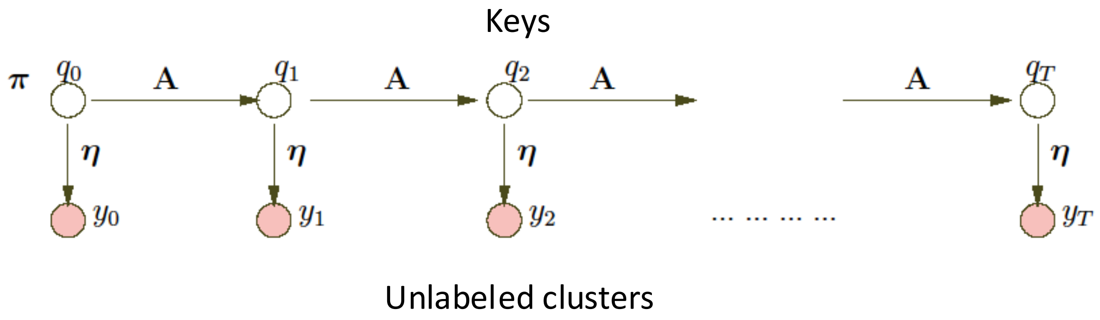
- Need to label the clusters: which key they represent?

- Assume the victim is typing English text
  - Characters follow certain frequency
  - Actual content follows English spelling and grammar

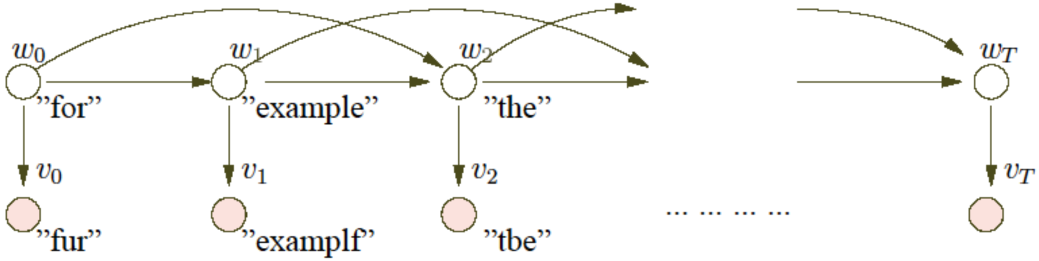
- Advantages:
  - Use 2-character combination frequency to match classes to keys
  - Use language model (spelling, grammar) to correct mistakes

Details: Context-based Language Model

- Character-level mapping:
  - Hidden Markov Model
  - Produce a probability of keys assigned to classes.
  - Example: “th” vs. “tj”



- Word-level correction:
  1. Spell check
  2. Grammar
     - Tri-gram



Details: Context-based Language Model

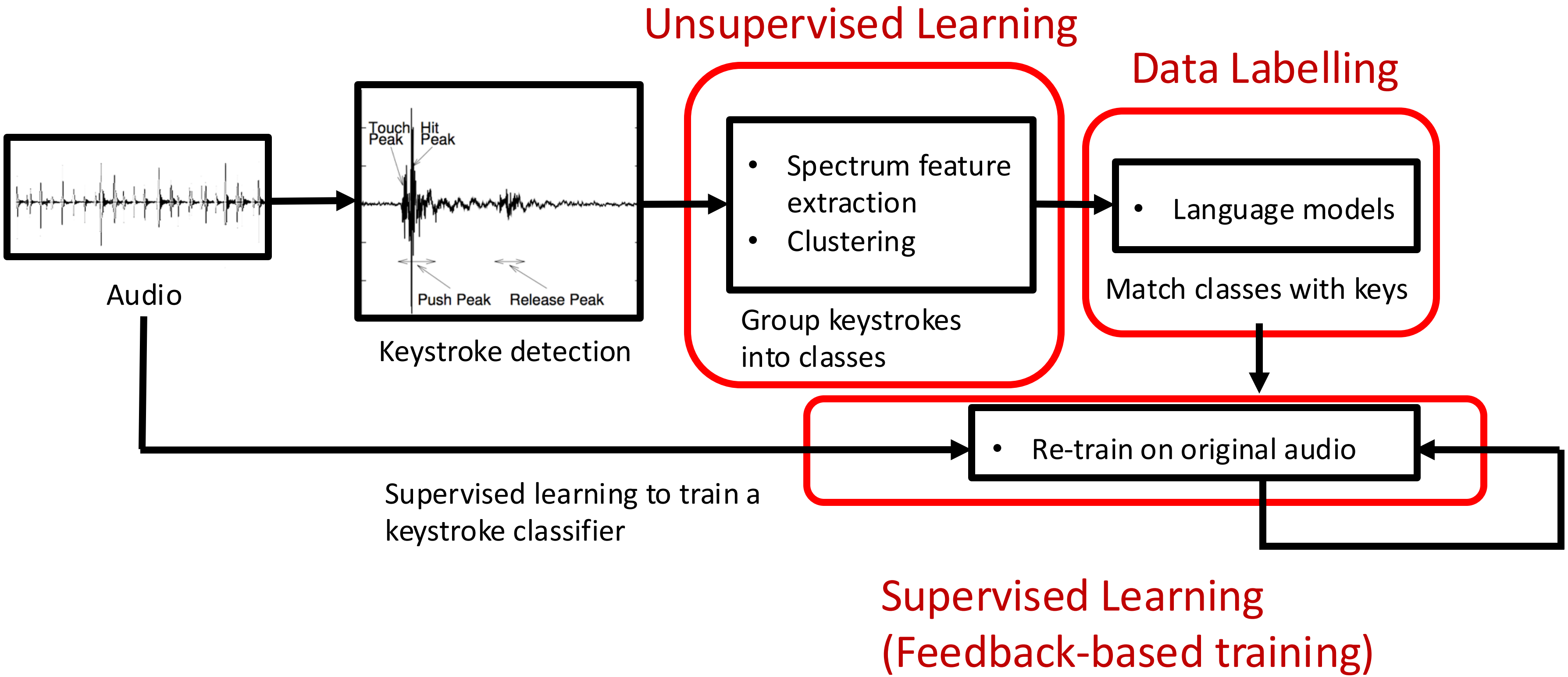
Before spelling and grammar correction

the big money fight has drawn the shoporo
od dosens of companies in the entertainment
industry as well as attorneys gnnerals on
states, who fear the fild shading softwate
will encourage illegal acyivitt, srem the
grosth of small arrists and lead to lost
cobs and dimished sales tas revenue.

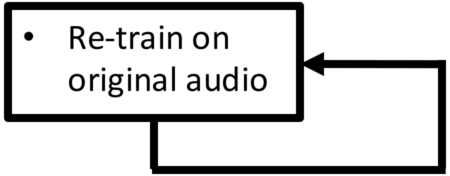
After spelling and grammar correction

the big money fight has drawn the support
of dozens of companies in the entertainment
industry as well as attorneys generals
in states, who fear the film sharing software
will encourage illegal activity, stem the
growth of small artists and lead to lost
jobs and finished sales tax revenue.

A Combination of Different Learning Methods

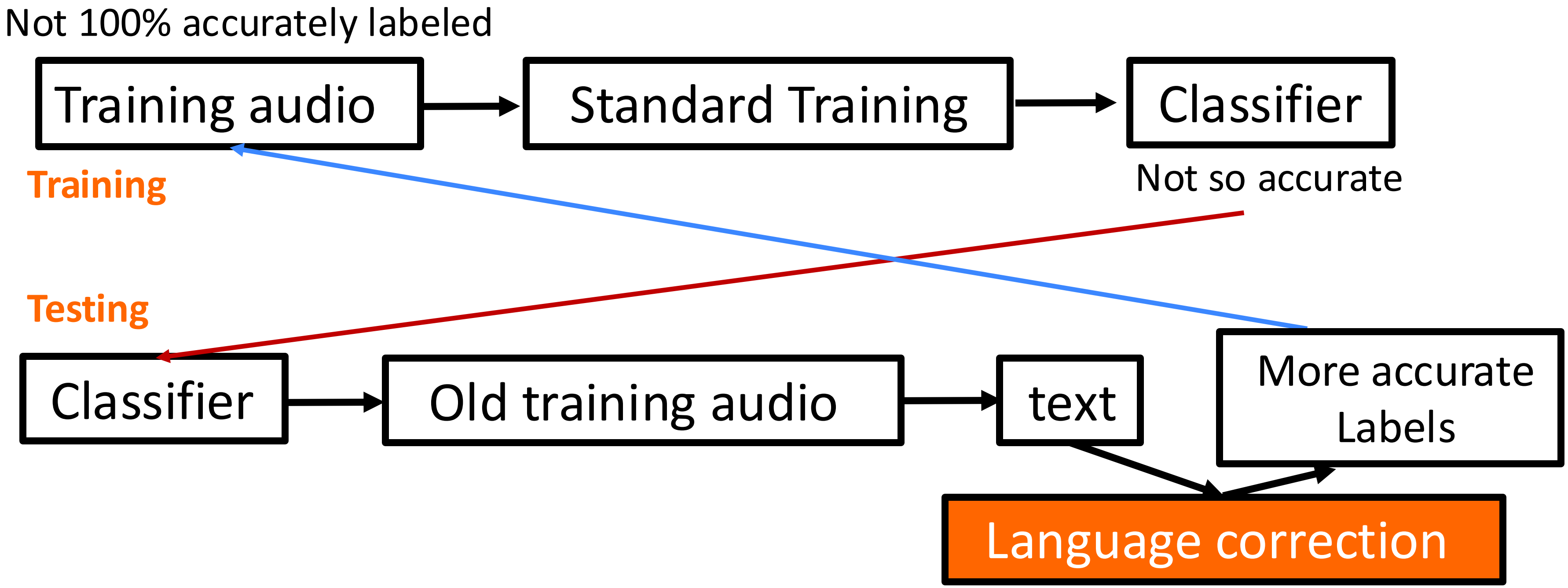


Feedback based Training



- A keystroke classifier (for inferring random text)
  - Given a keystroke, produce the label of the key
- Training
  - Input: noisy training data
    - Only a subset of labeled data from the language models
    - Choose those with fewer corrections by the language model (quality indicator)
  - Output: a not so accurate keystroke classifier
- Testing
  - Use the trained classifier to classify the training data again
  - Use the language model to correct the classification result
  - Use the corrected label for re-training

Feedback based Training (Con't)



Evaluation

		Set 1		Set 2		Set 3		Set 4	
		words	chars	words	chars	words	chars	words	chars
unsupervised learning	keystrokes	34.72	76.17	38.50	79.60	31.61	72.99	23.22	67.67
	language	74.57	87.19	71.30	87.05	56.57	80.37	51.23	75.07
1st supervised feedback	keystrokes	58.19	89.02	58.20	89.86	51.53	87.37	37.84	82.02
	language	89.73	95.94	88.10	95.64	78.75	92.55	73.22	88.60
2nd supervised feedback	keystrokes	65.28	91.81	62.80	91.07	61.75	90.76	45.36	85.98
	language	90.95	96.46	88.70	95.93	82.74	94.48	78.42	91.49
3rd supervised feedback	keystrokes	66.01	**92.04**	62.70	**91.20**	63.35	**91.21**	48.22	**86.58**
	language	**90.46**	**96.34**	**89.30**	**96.09**	**83.13**	**94.72**	**79.51**	**92.49**

Table 2: Text recovery rate at each step. All numbers are percentages.

Other Key Results

- Works for random text
  - Inferring passwords that contain English letters only
  - 90% of 5-character random passwords: < 20 attempts
  - 80% of 10-character random passwords: <75 attempts
- Works for multiple types of keyboards
- Even "low-quality" microphones can do the job

Possible Defenses

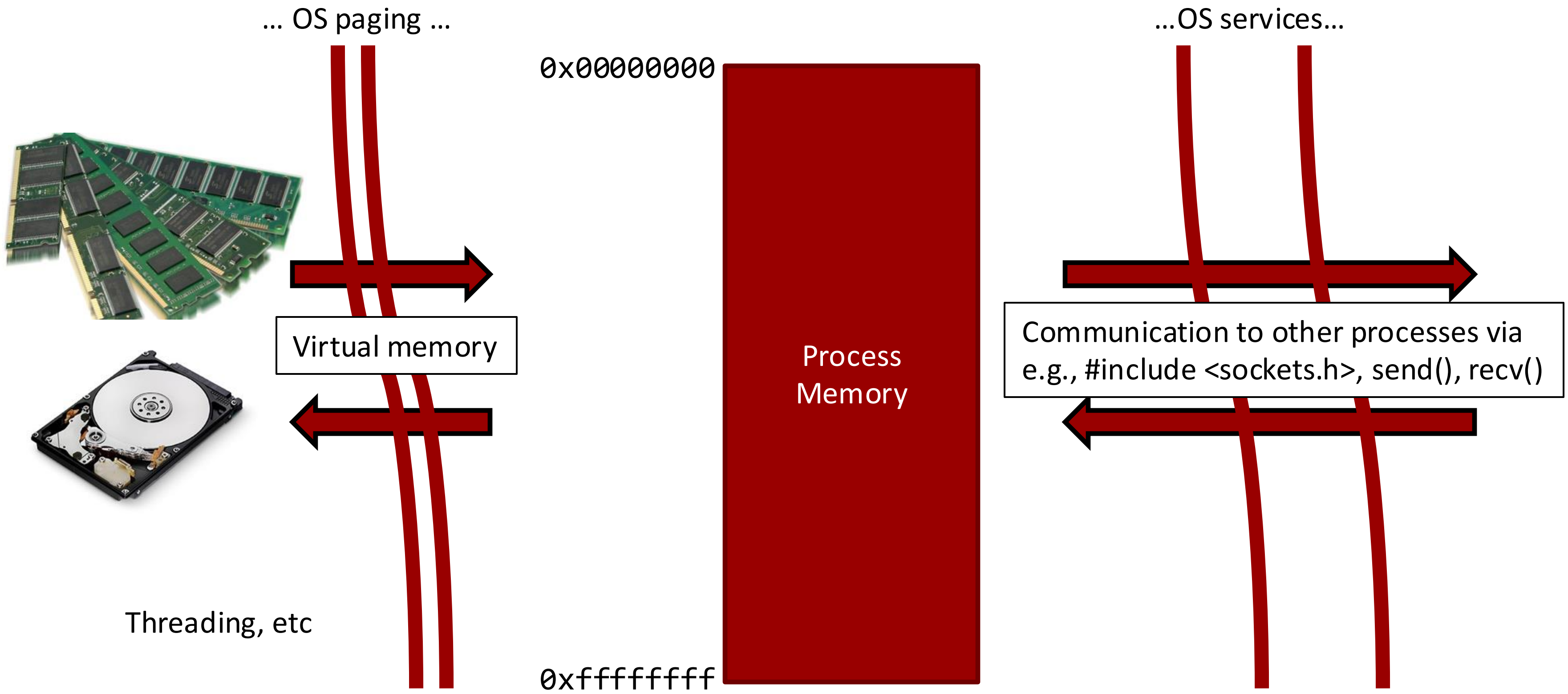
- Introduce noise into the system
  - Add (random) background noise to keystrokes
    - Remove the unique pattern for each key
  - Use quieter keyboards

- Other defenses
  - Two factor authentication (not just typing a password)
  - No microphone in your room?

Microarchitectural covert and side channels (how to share a secret)

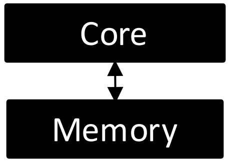
Credit: Chris Fletcher (UIUC)

Process isolation + OS (CS 233)

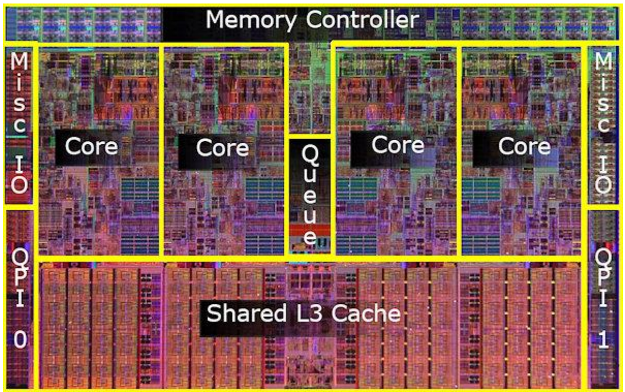


Programs run on processors

- Processor that OS would have you see …

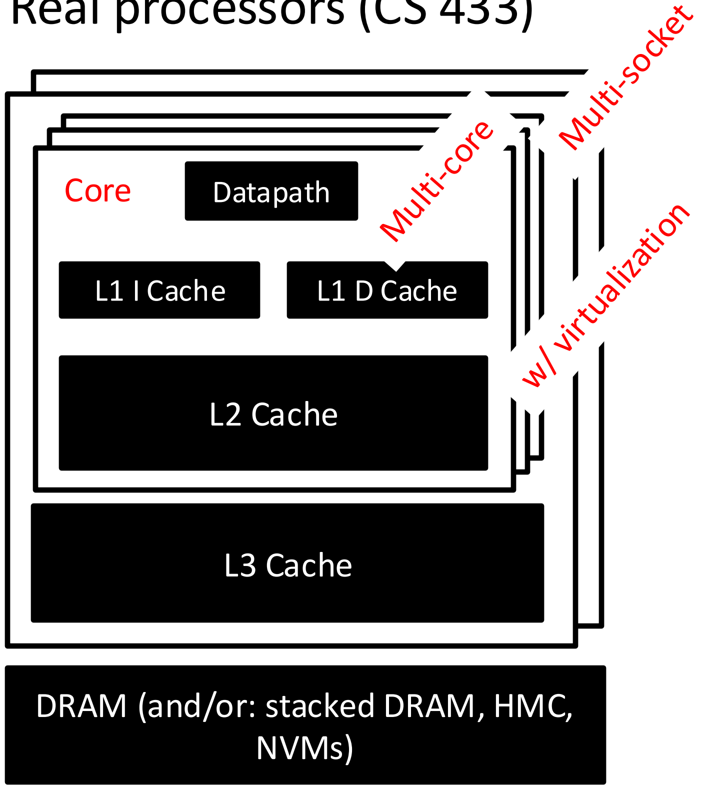


OS swaps work on/off



Cache = on-chip memory, faster to access than DRAM

- Real processors (CS 433)

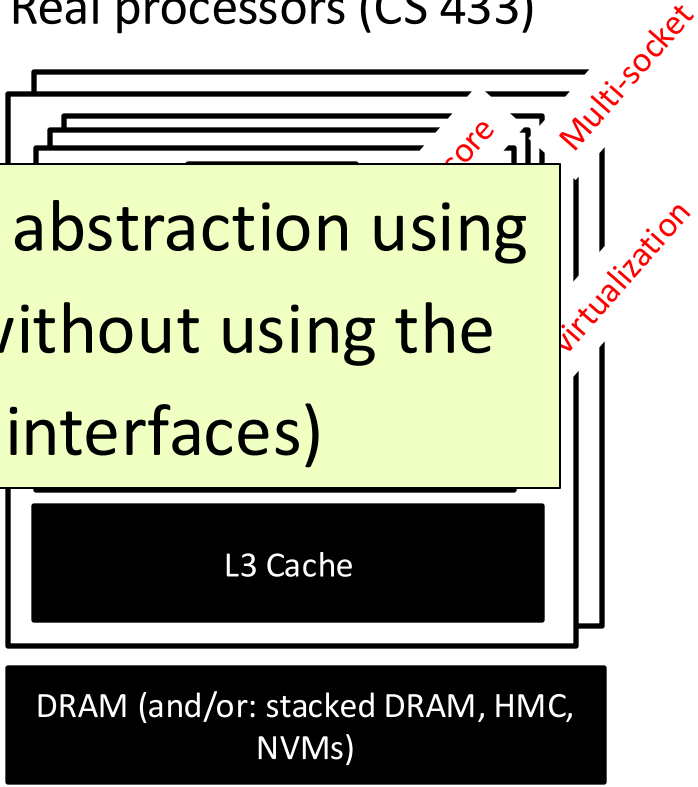


Programs run on processors

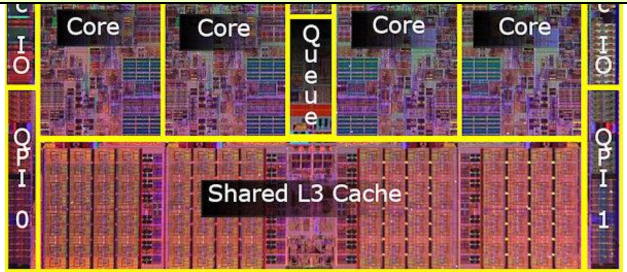
- Processor that OS would have you see …

Cache = on-chip memory, faster to access than DRAM

- Real processors (CS 433)



Goal: create a send(), recv() abstraction using Hardware contention (→ without using the OS/other sanctioned interfaces)



Covert Channels 101: Through the cache

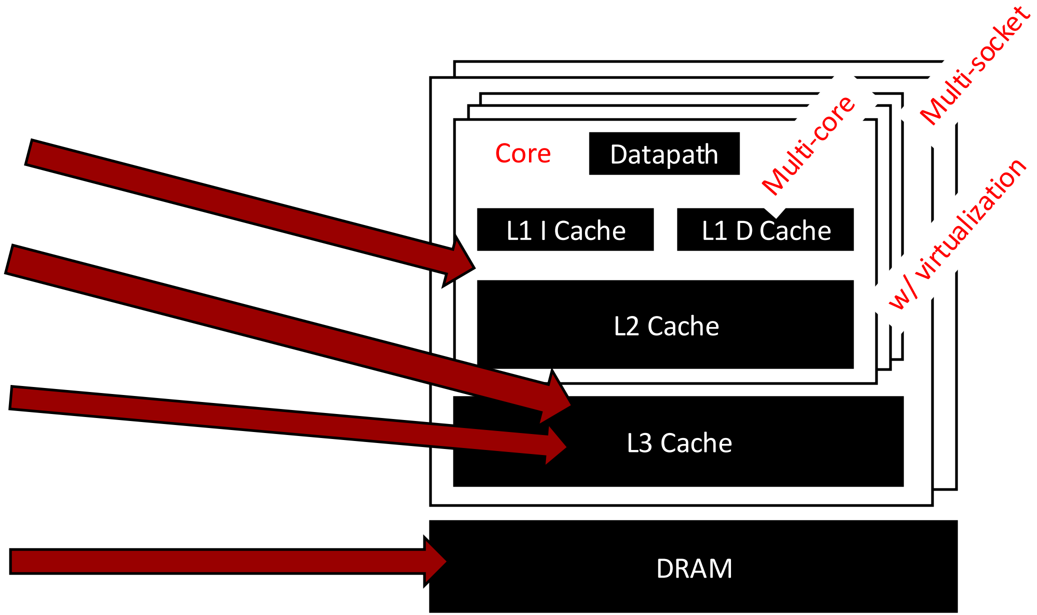
- Cache fill for line A may cause another line B to be evicted
- Various mechanisms for owner of B to detect a hit or miss
- We like the cache: easy to measure, many types of sharing

L1/L2 → Intra-core, inter-thread channels

LLC → inter-core channels

Directory → inter-core/inter-socket channels

DRAM row buffer → “”

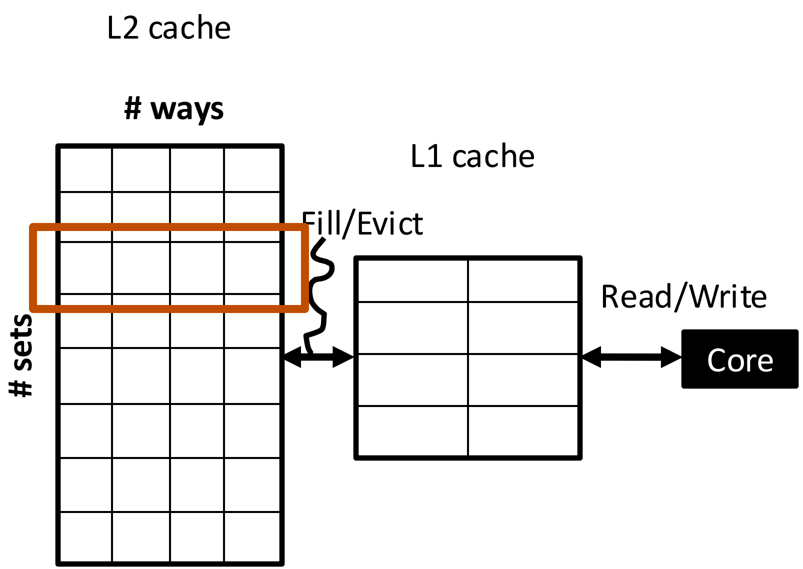


Processor caches

- Motivation
  - Programs have locality
  - Memory access cost ∝ memory size
- Block placement/replacement policies tell us where blocks can live and when

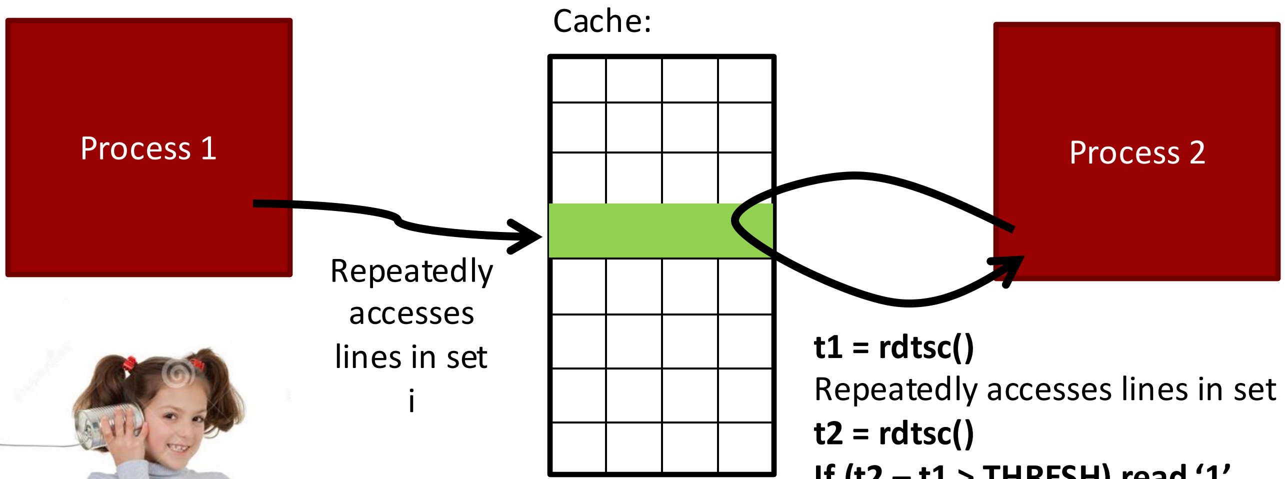
Core-facing API: Read(addr)
Write(addr, word)

Backend API: Evict(addr)
Fill(addr, line)



Why is cache design relevant?

- Two processes can agree on “dead drops” on the processor hardware, to pass information under the OS’s nose



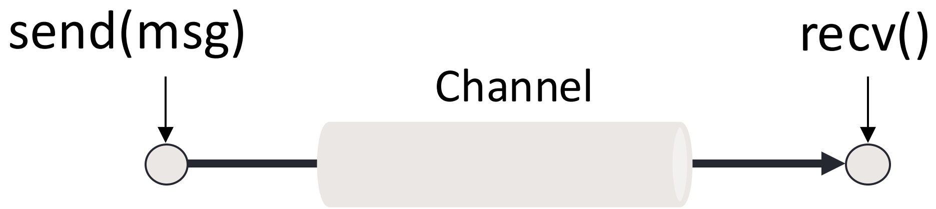
t1 = rdtsc()
Repeatedly accesses lines in set i
t2 = rdtsc()
If (t2 – t1 > THRESH) read ‘1’
Else read ‘0’

weapon.cs.illinois.edu - PuTTY

```
cwfletch@weapon:~/deaddrop$ htop
cwfletch@weapon:~/deaddrop$ ls
lab598clf_v0-2.pdf  Makefile_sender  receiver.c  sender.c      usage   util.o
LICENSE             README.md        receiver.o  sender.o      util.c
Makefile_receiver   receiver         sender      ta_solution   util.h
cwfletch@weapon:~/deaddrop$ ./sender
Please type a message (exit to stop).
<
```

weapon.cs.illinois.edu - PuTTY

```
cwfletch@weapon:~/deaddrop$ ls
lab598clf_v0-2.pdf  Makefile_sender  receiver.c  sender.c     usage   util.o
LICENSE             README.md        receiver.o  sender.o     util.c
Makefile_receiver   receiver         sender      ta_solution  util.h
cwfletch@weapon:~/deaddrop$ ./receiver
Press enter to begin listening
```



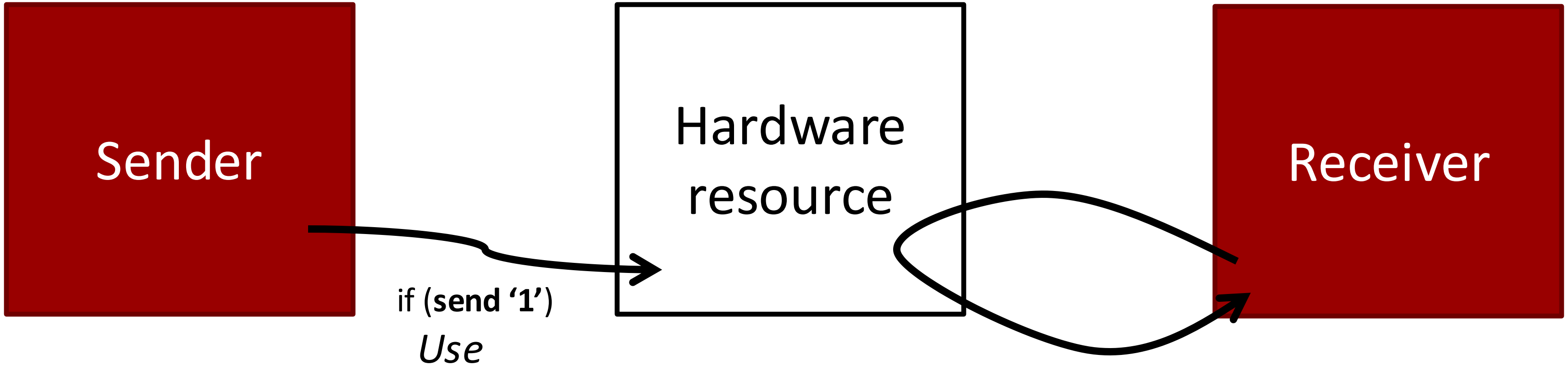
Normal communication

```
include <socket.h>

void send(bit msg) {
  socket.send(msg);
}

bit recv() {
  return socket.recv(msg);
}
```

Covert Channel communication

```
void send(bit msg) {
  // pressure on cache
}

bit recv() {
  st = time();
  // pressure on cache
  return time() - st > THRESH;
}
```

Fun! How else can I do this?

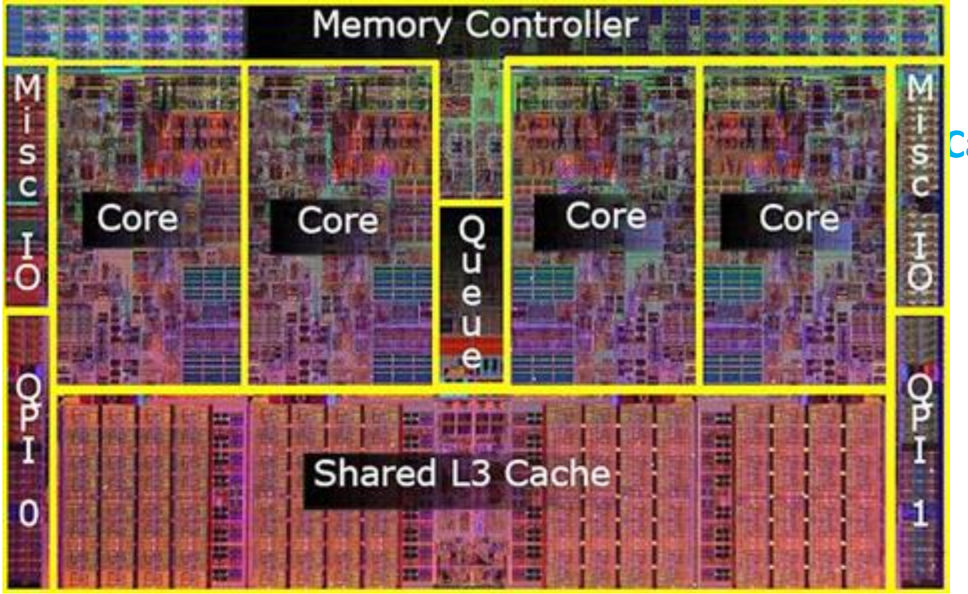


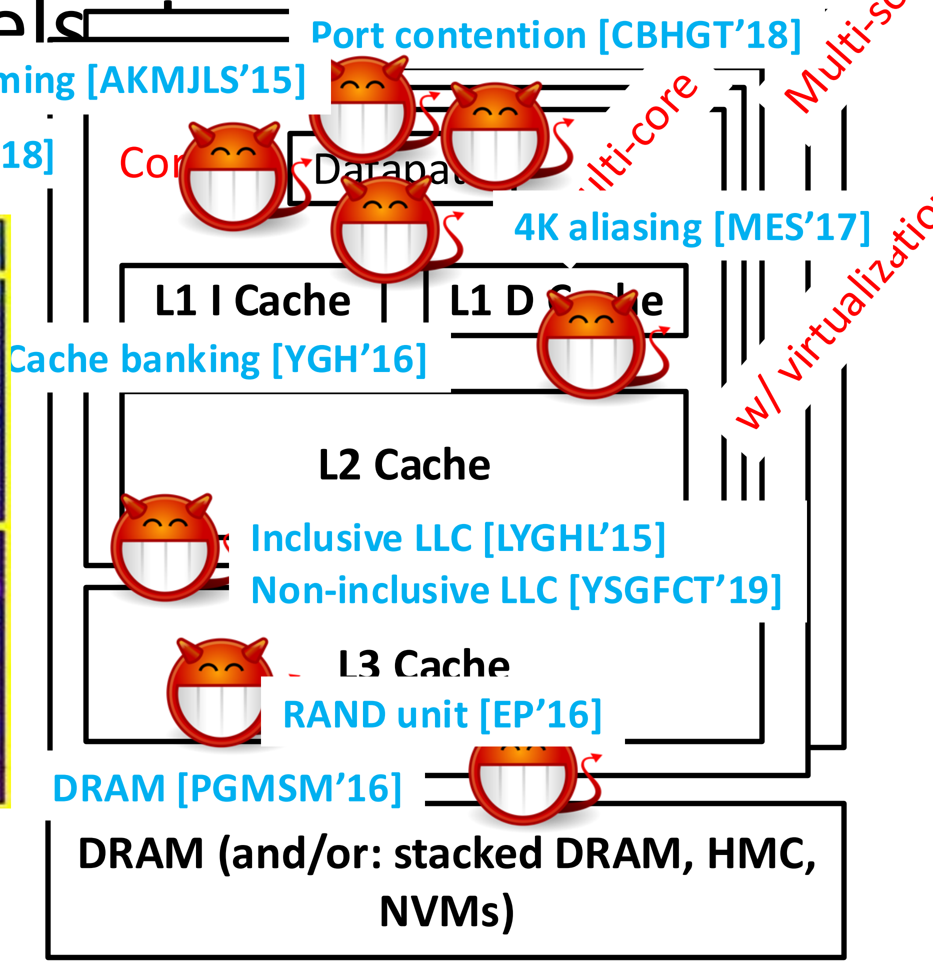
if (**send ‘1’**)
  Use resource
else
  idle

t1 = rdtsc()
Use resource
t2 = rdtsc()

if (t2 – t1 > THRESH) read ‘1’
else read ‘0’

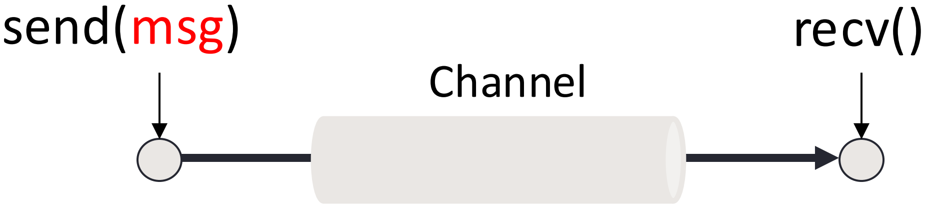
Many potential channels





Bandwidth

Error-free bitrate of send() → recv()



Depends on what hardware structure is used to build the channel.

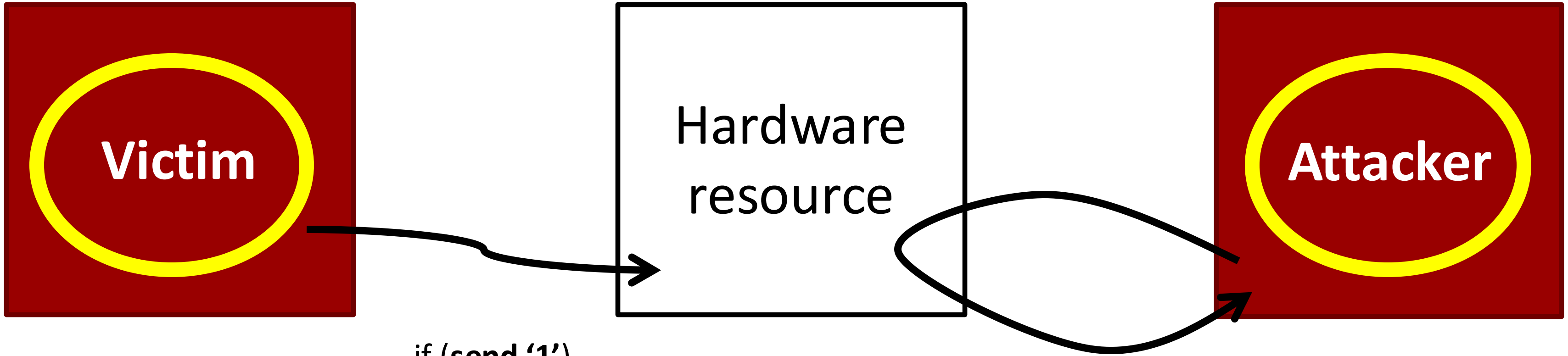
- RDRAND unit: 7-200 Kbps [EP'16]
- Ld/st performance counters: ~75-150 Kbps [HKRVDT'15]
- MemBus/AES-NI contention: ~550-650 Kbps [HKRVDT'15]
- LLC: 1.2 Mbps [MNHF'15]
- Various structures on GPGPU: up to 4 Mbps [NKG'17]

Practical uses

- Talk to your friends for fun
- Malware can inter-communicate w/o OS realizing it
- Different VMs sharing the same box on (e.g.) Amazon AWS can talk

- Side channel attacks
  - Learn private information about co-resident processes

From covert → side channels



Covert channel:

if (**send ‘1’**)
Use resource
else
idle

Side channel:

if (**secret**)
Use resource
else
idle

t1 = rdtsc()
Use resource
t2 = rdtsc()

if (t2 – t1 > THRESH) read ‘1’
else read ‘0’

Side channel attacks

- Shared resource pressure can also lead to side channel attacks
- E.g., RSA encryption   msg = $\text{Decrypt}_{\text{key}}(\text{Encrypt}_{\text{key}}(\text{msg}))$

```
SquareMult(x, e, N):
  let e_n, ..., e_1 be the bits of e
  y ← 1
  for i = n down to 1 {
    y ← Square(y)                 (S)
    y ← ModReduce(y, N)           (R)
    if e_i = 1 then {
      y ← Mult(y, x)              (M)
      y ← ModReduce(y, N)         (R)
    }
  }
  return y
```

Discussion

- Any other examples of side channels you can think of to infer user information / steal data?

- What’s your thoughts on the future development of microarchitecture side channels (try to also think from the defender’s side of view)?