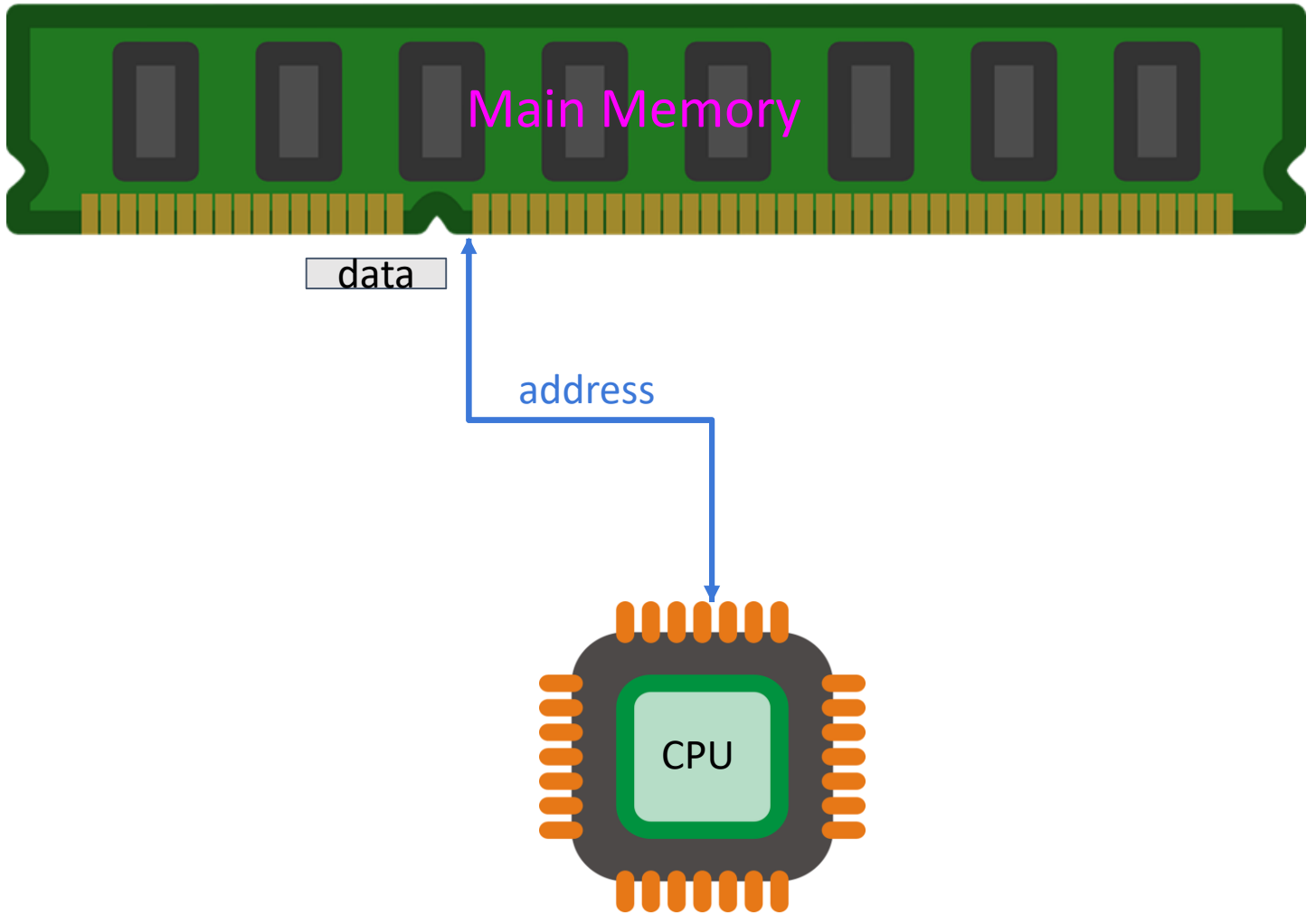
Cache Side-Channel Attacks (Brief Introduction)

Presenter: Aria Shahverdi

9/31/2019

How do we load data from Main Memory?



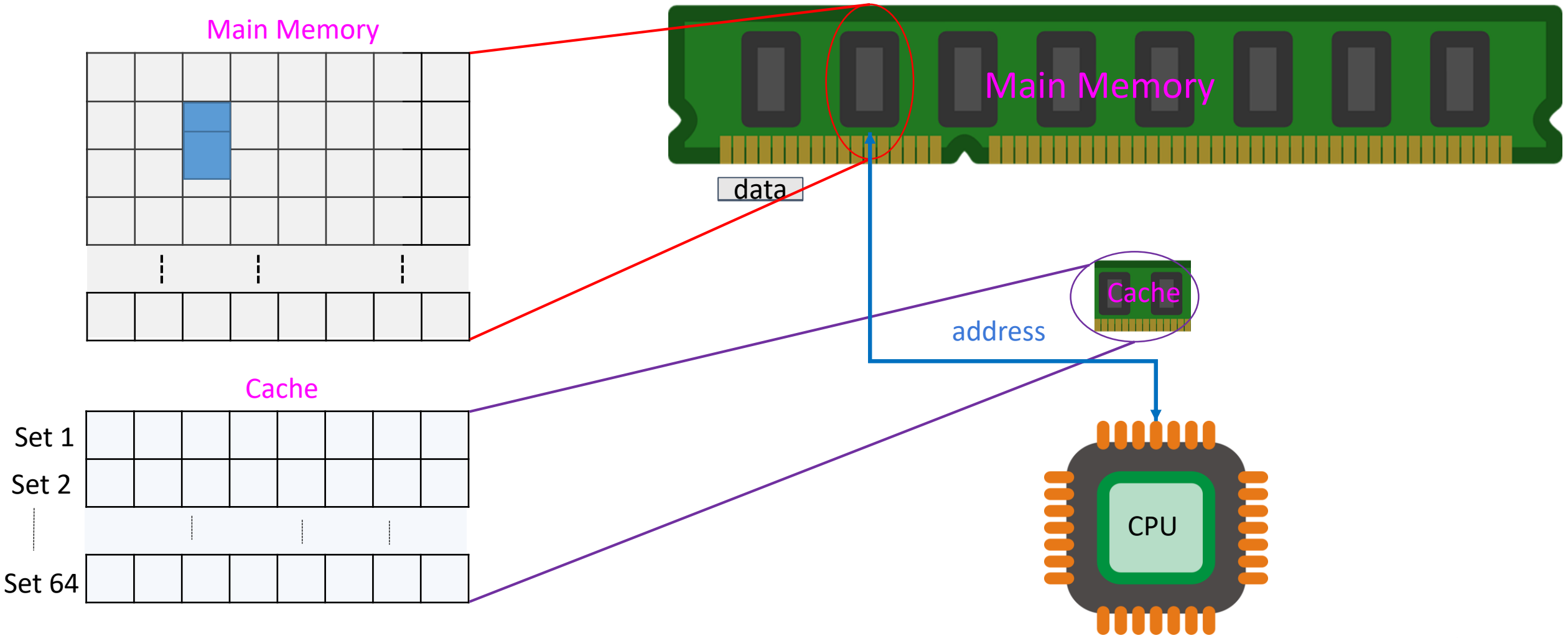
Memory Locality

- Future memory accesses are near past memory accesses
- Memories take advantages of two locality
  - **Temporal Locality**: near in time
    - We will often access the same data again very soon
  - **Spatial Locality**: near in space/distance
    - Our next access is often very close to our last access (or recent accesses)

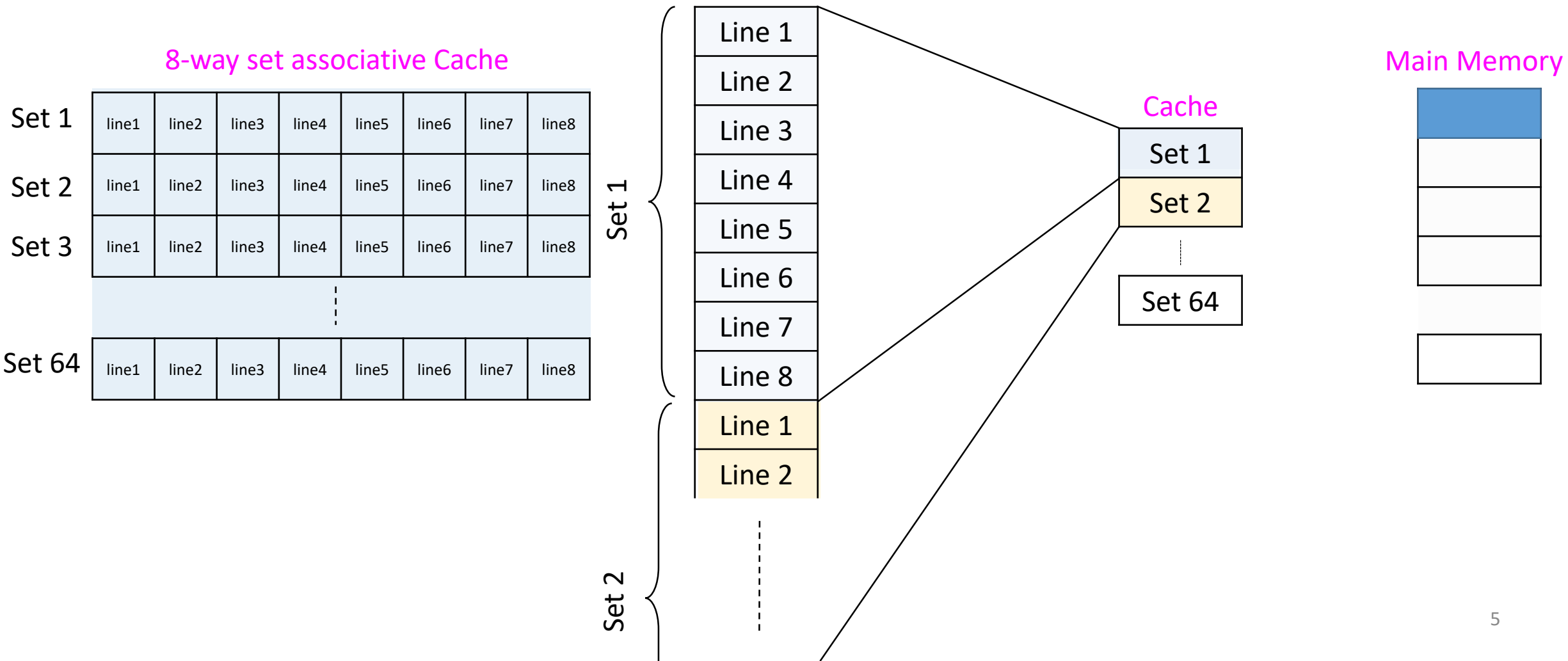
```
for(i = 0; i < 20; i++)
  a[i] = a[i]*2;
```

```
a[0]
a[1]
a[2]
…
```

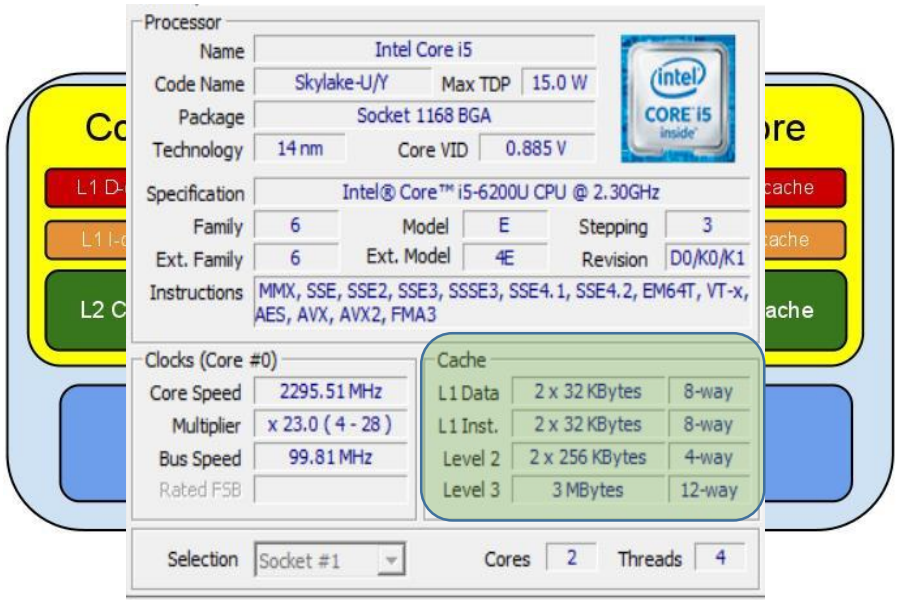
Cache Architecture High Level

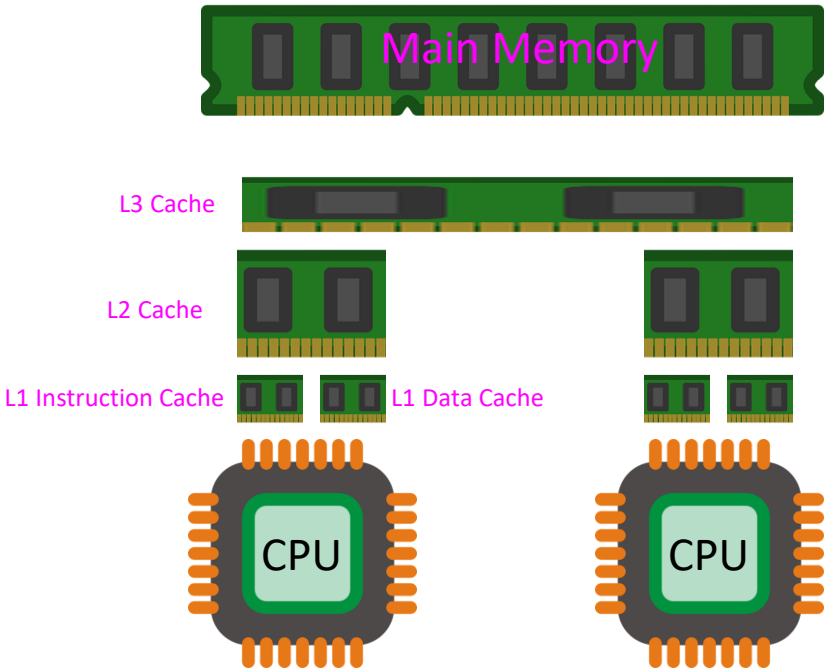


Set Associative Cache Architecture



Introduction to Cache Architecture

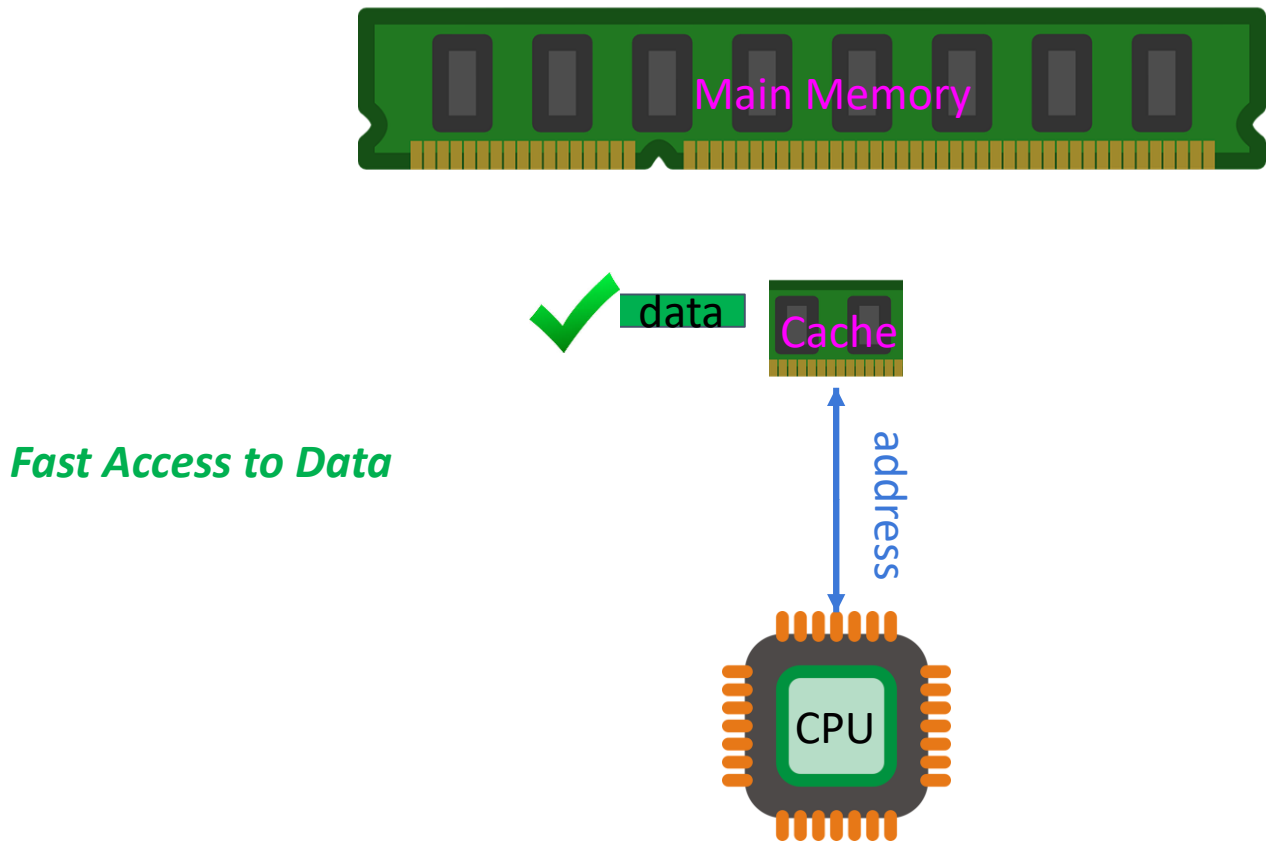




Cache Architecture (Summary)

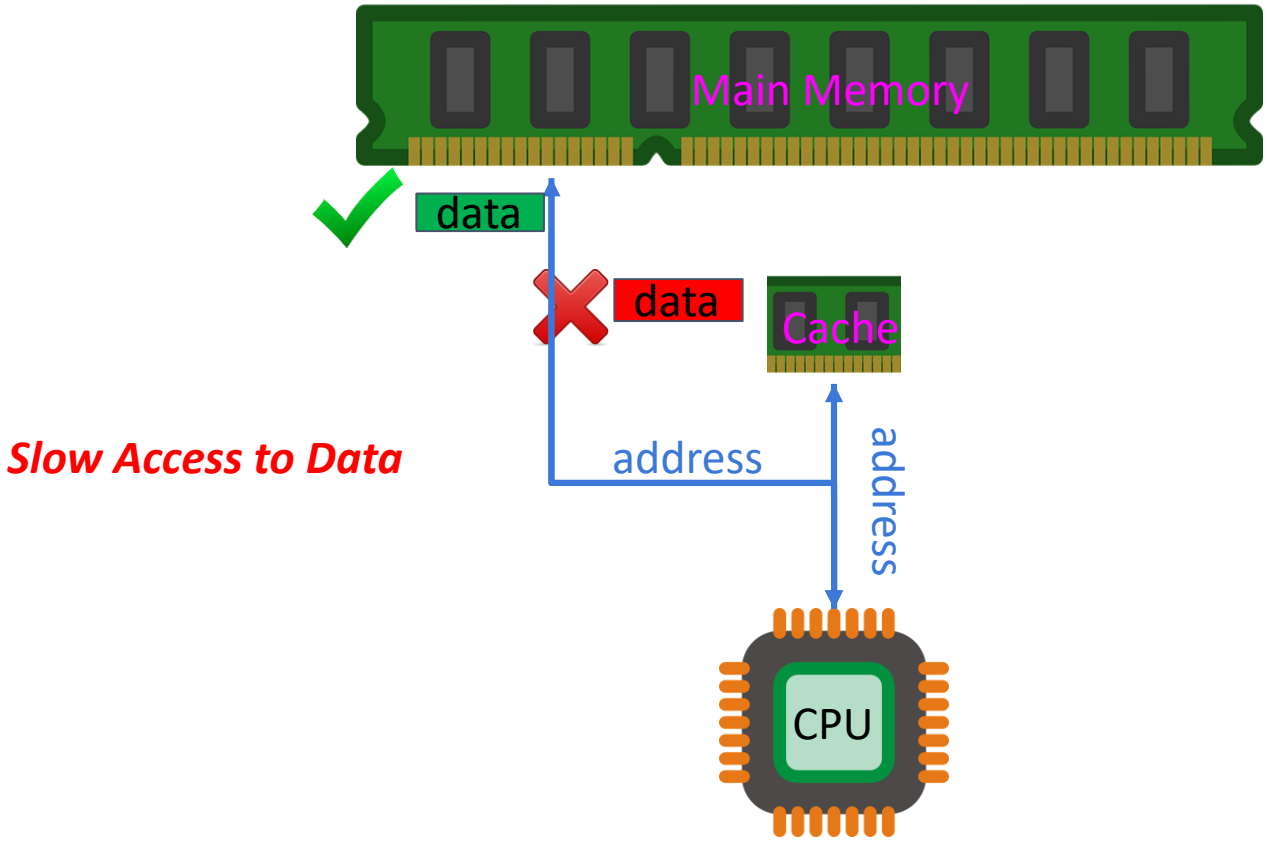
- Unit of Memory in cache is a line
- A cache consists of multiple sets which stores fixed number of lines
- The number of lines in a set is called associativity
  - L1 is 8-way, L2 is 4-way, L3 is 12-way
- Last Level Cache (LLC) is inclusive
  - LLC contains copies of all of the data in the lower cache level
  - Evicting data from LLC remove that data from all other cache levels

Accessing Memory (Cache Hit)



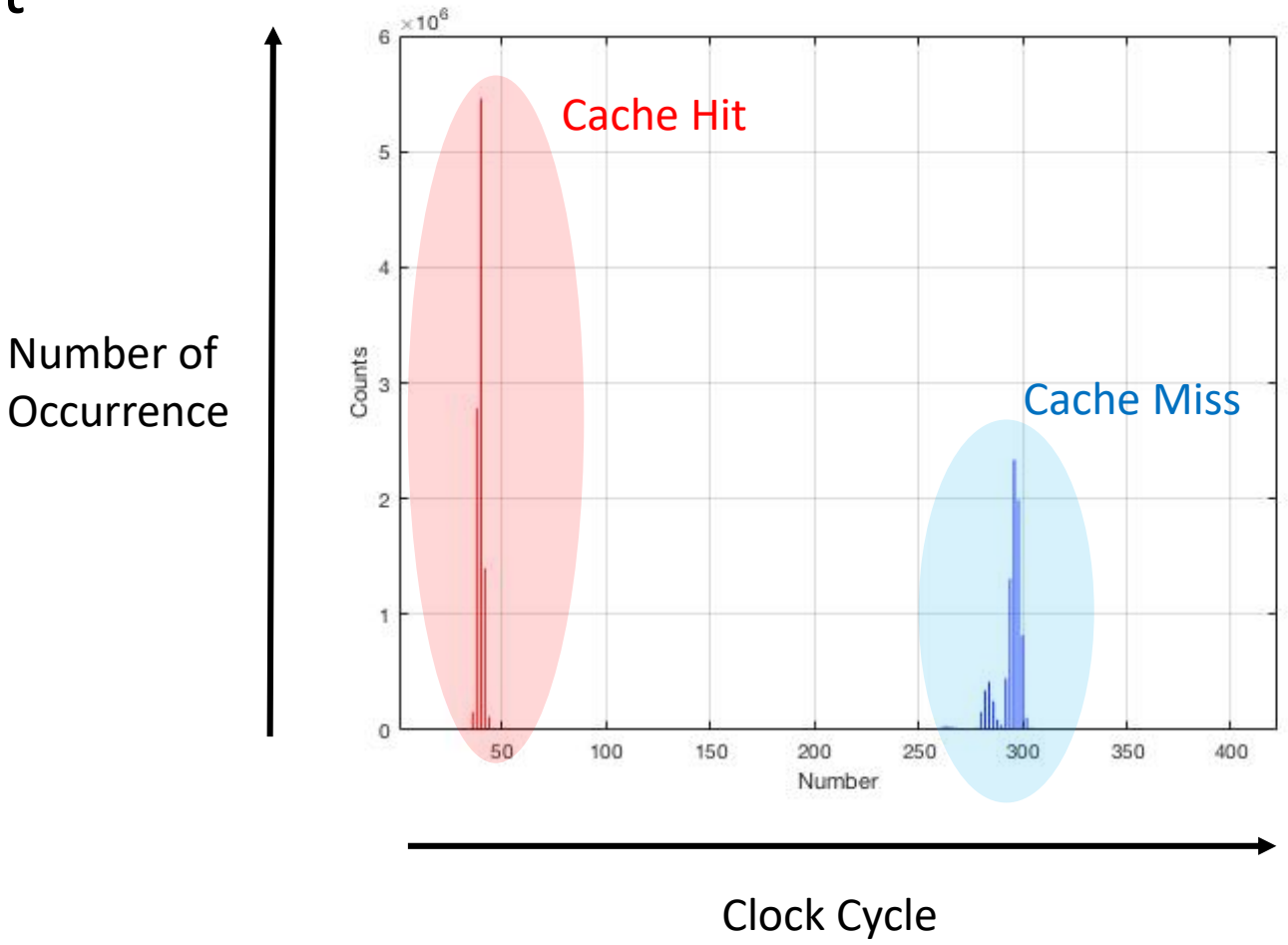
Fast Access to Data

Accessing Memory (Cache Miss)



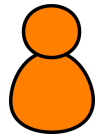
Cache Hit vs. Miss Time Difference

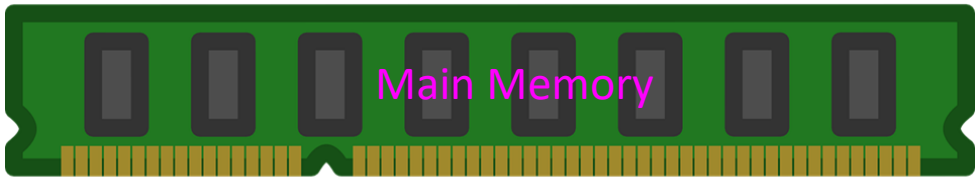
- ≈10 Million measurement

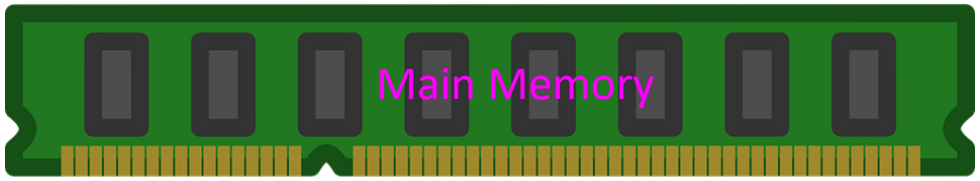


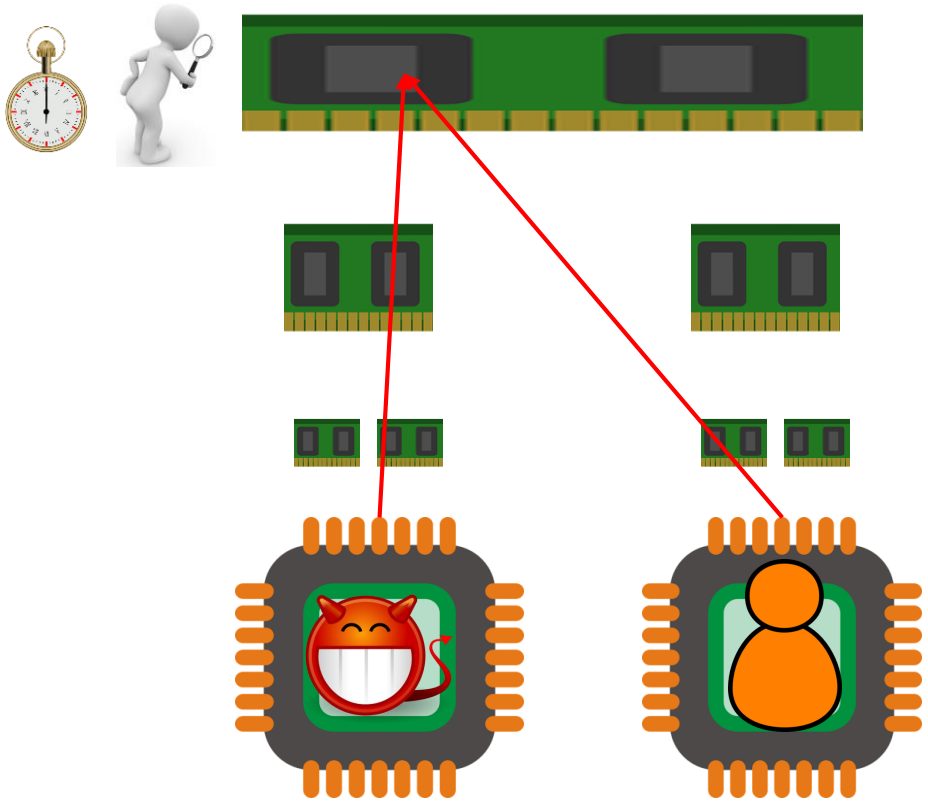
Cache Attack Model

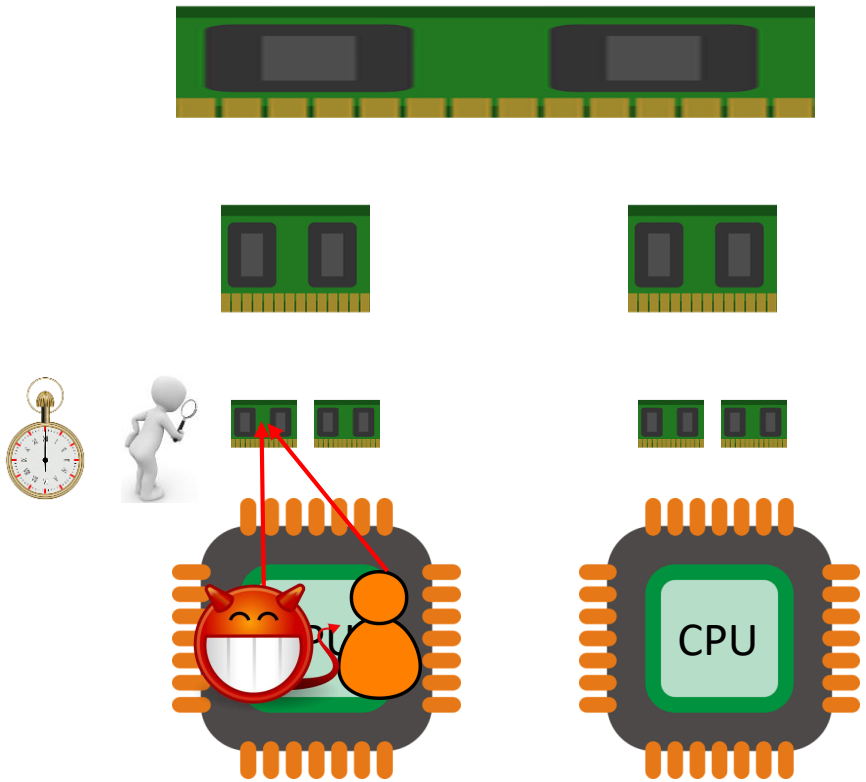

: Attacker


: Victim



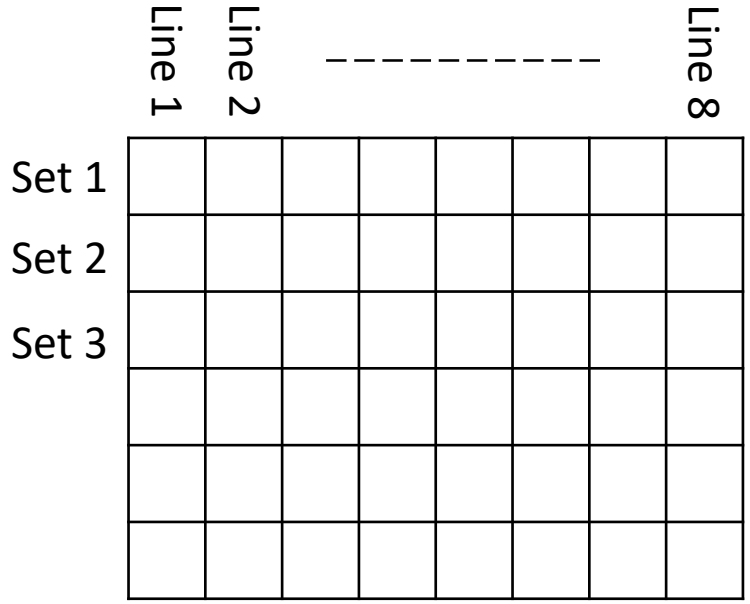






Some Cache Attack Technique

- Evict and Time
- Flush and Reload
- Prime and Probe

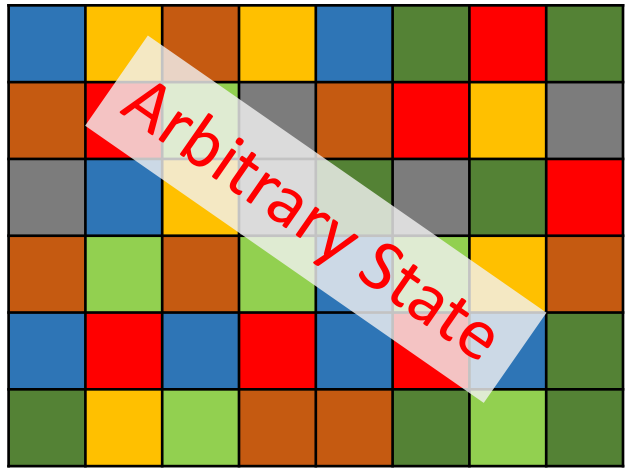


Some Cache Attack Technique

- Evict and Time
- Flush and Reload
- Prime and Probe

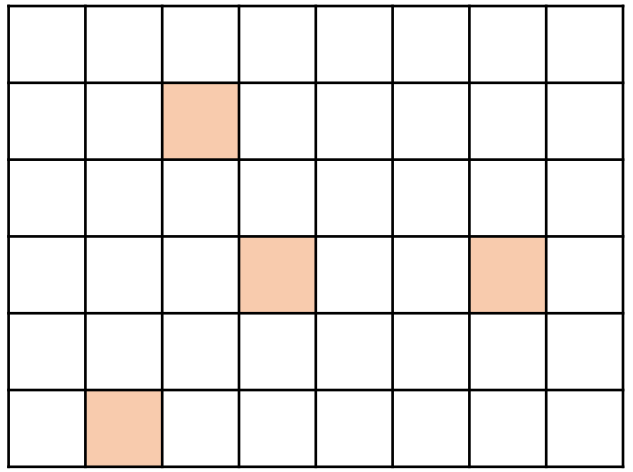
Evict and Time

1. Trigger encryption
2. Selectively manipulate the state of the cache (e.g. evict a full cache set)
3. Trigger encryption
4. Measure how long it took
5. Deduce what cache sets it accessed
6. Repeat step 1-4 to gain information on all the set the encryption accessed



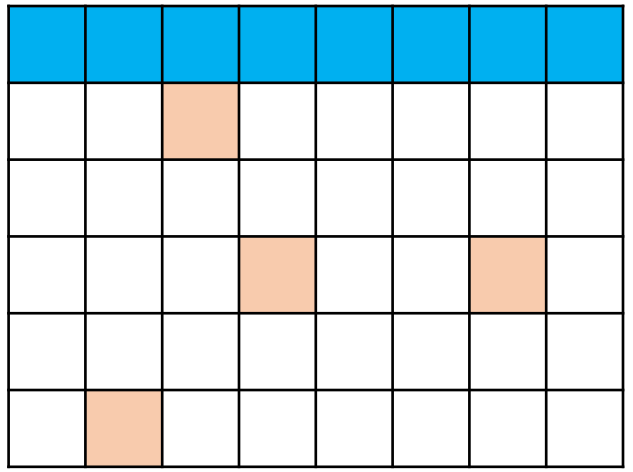
Evict and Time

1. Trigger encryption
2. Selectively manipulate the state of the cache (e.g. evict a full cache set)
3. Trigger encryption
4. Measure how long it took
5. Deduce what cache sets it accessed
6. Repeat step 1-4 to gain information on all the set the encryption accessed



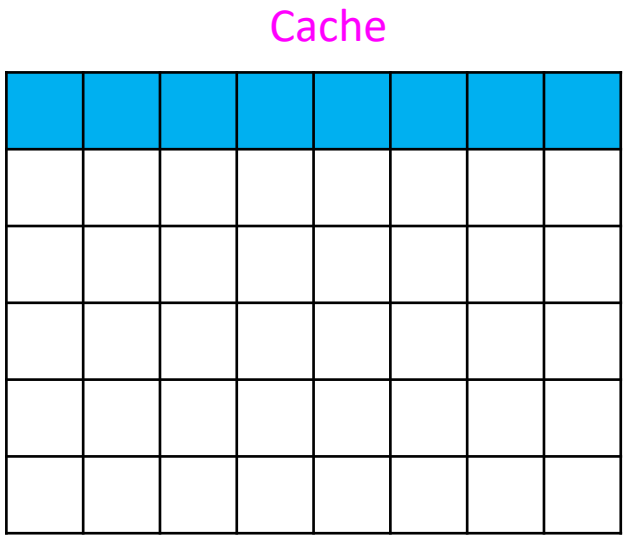
Evict and Time

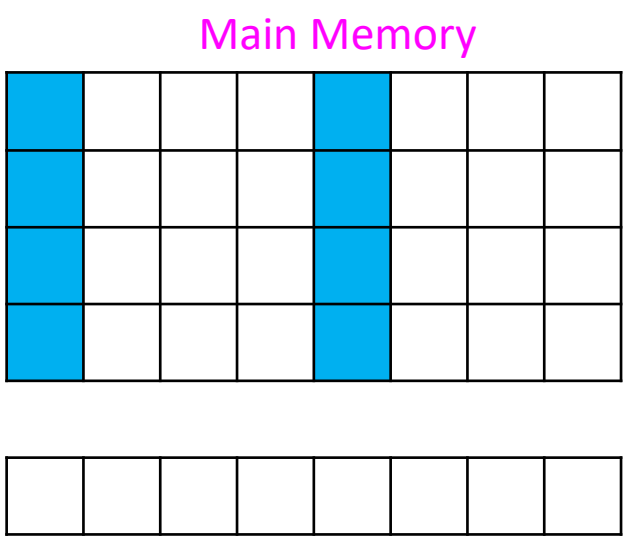
1. Trigger encryption
2. Selectively manipulate the state of the cache (e.g. evict a full cache set)
3. Trigger encryption
4. Measure how long it took
5. Deduce what cache sets it accessed
6. Repeat step 1-4 to gain information on all the set the encryption accessed



How do we fill a cache set?

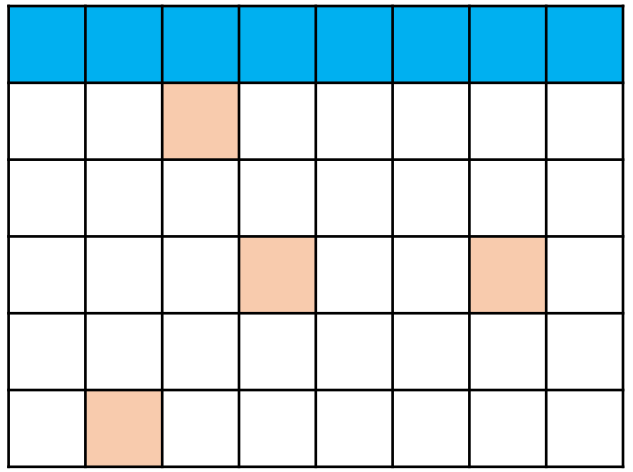
- By Accessing some of the memory locations the corresponding locations in the cache is going to be filled.
- Main Challenge: which lines to access?





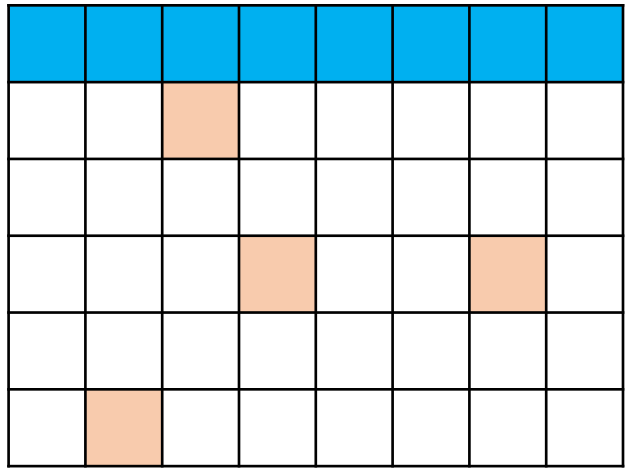
Evict and Time

1. Trigger encryption
2. Selectively manipulate the state of the cache (e.g. evict a full cache set)
3. Trigger encryption
4. Measure how long it took
5. Deduce what cache sets it accessed
6. Repeat step 1-4 to gain information on all the set the encryption accessed



Evict and Time

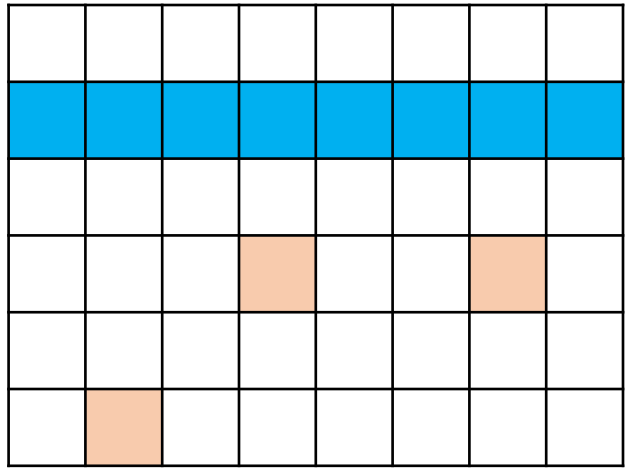
1. Trigger encryption
2. Selectively manipulate the state of the cache (e.g. evict a full cache set)
3. Trigger encryption
4. Measure how long it took
5. Deduce what cache sets it accessed
6. Repeat step 1-4 to gain information on all the set the encryption accessed



It took almost the same amount time

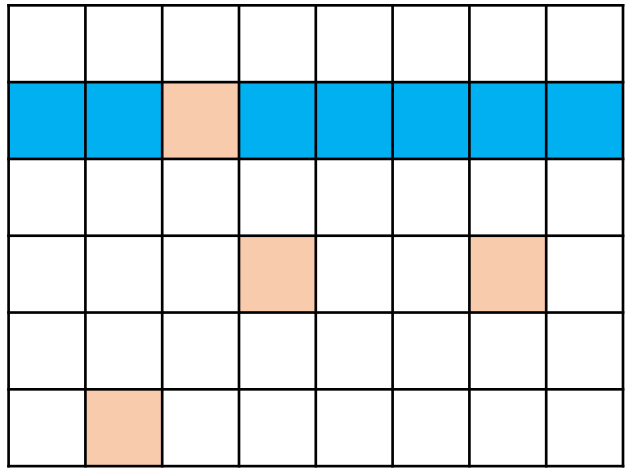
Evict and Time

1. Trigger encryption
2. Selectively manipulate the state of the cache (e.g. evict a full cache set)
3. Trigger encryption
4. Measure how long it took
5. Deduce what cache sets it accessed
6. Repeat step 1-4 to gain information on all the set the encryption accessed



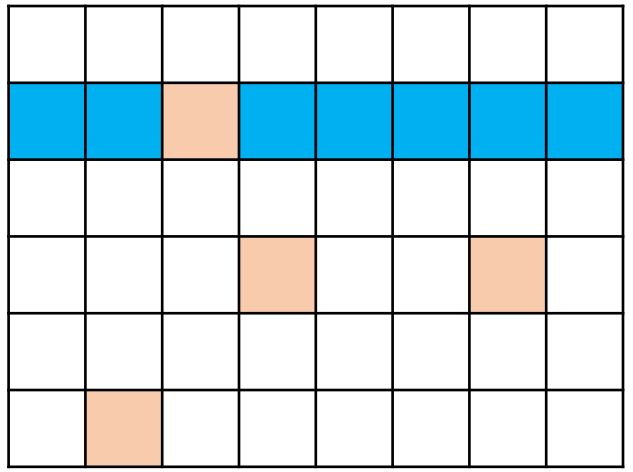
Evict and Time

1. Trigger encryption
2. Selectively manipulate the state of the cache (e.g. evict a full cache set)
3. Trigger encryption
4. Measure how long it took
5. Deduce what cache sets it accessed
6. Repeat step 1-4 to gain information on all the set the encryption accessed



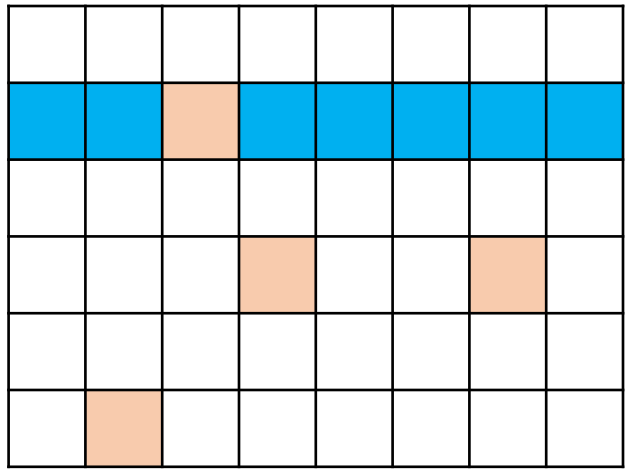
Evict and Time

1. Trigger encryption
2. Selectively manipulate the state of the cache (e.g. evict a full cache set)
3. Trigger encryption
4. Measure how long it took
5. Deduce what cache sets it accessed
6. Repeat step 1-4 to gain information on all the set the encryption accessed



Evict and Time

1. Trigger encryption
2. Selectively manipulate the state of the cache (e.g. evict a full cache set)
3. Trigger encryption
4. Measure how long it took
5. Deduce what cache sets it accessed
6. Repeat step 1-4 to gain information on all the set the encryption accessed



Set 2 was accessed!!

Some Cache Attack Technique

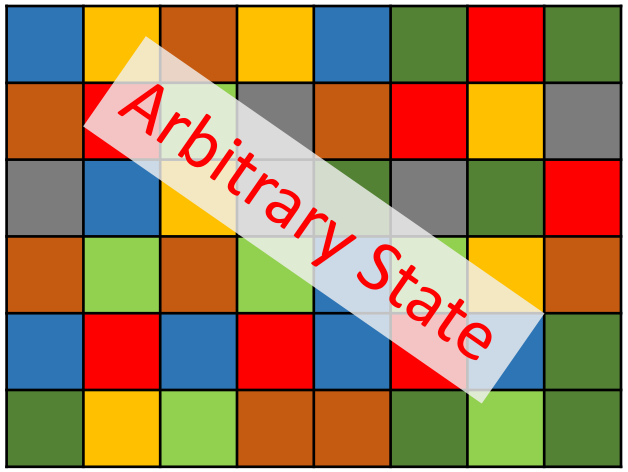
- Evict and Time
- Flush and Reload
- Prime and Probe

Flush and Reload

- Exploits cache behavior to leak information on victim access to shared memory.
  - Shared libraries
  - Memory de-duplication
- Spy monitors victim’s access to shared code
  - Spy can determine what victim does
  - Spy can infer the data the victim operates on

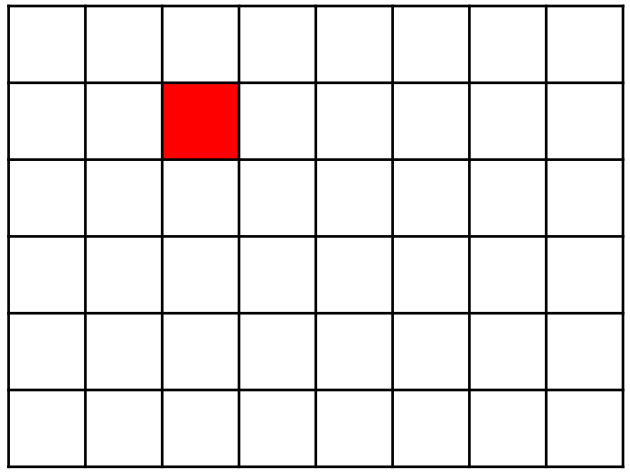
Flush and Reload

1. Flush memory line
2. Wait a bit
3. Measure time to Reload line
4. Repeat

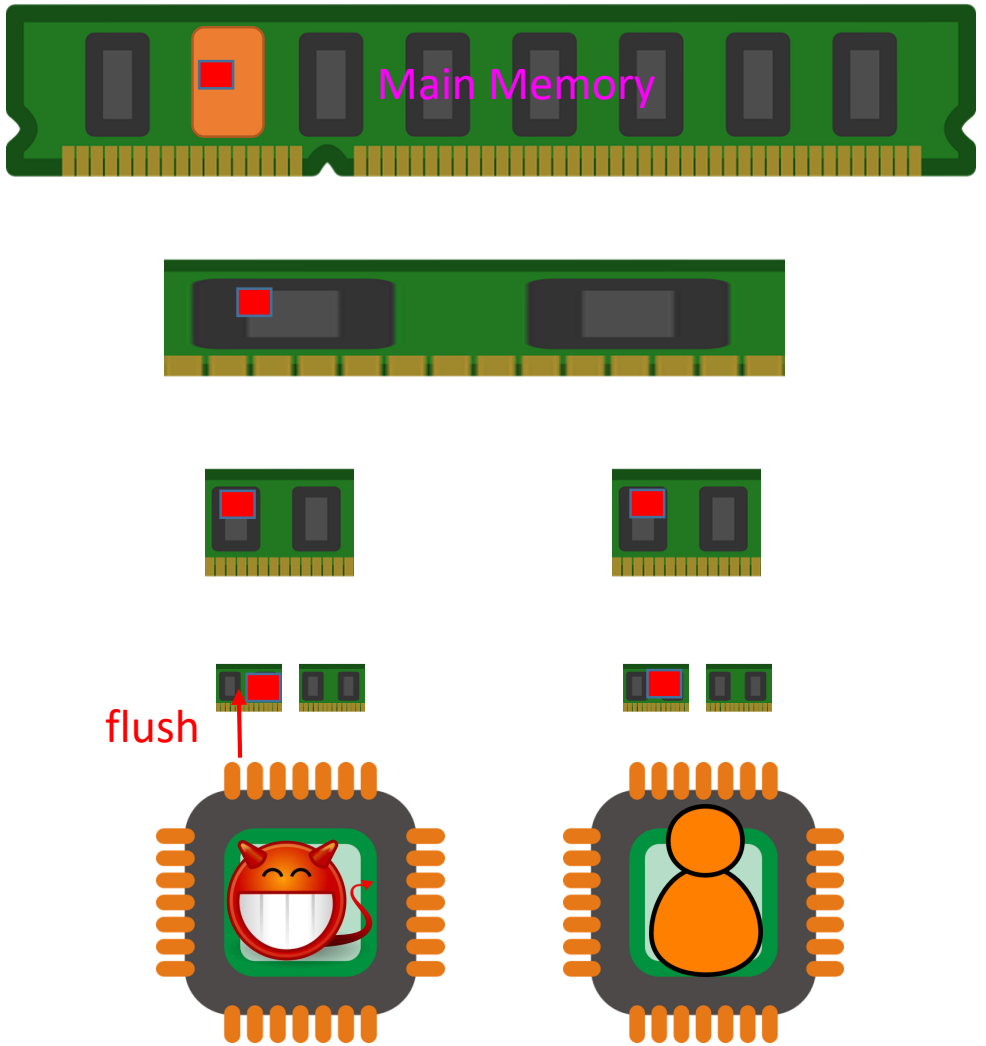


Flush and Reload

1. Flush memory line
2. Wait a bit
3. Measure time to Reload line
4. Repeat

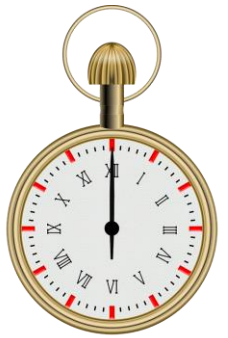


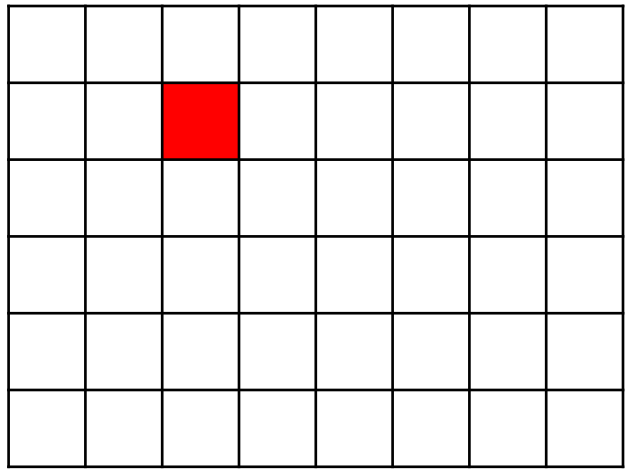
Flush a Line From Cache



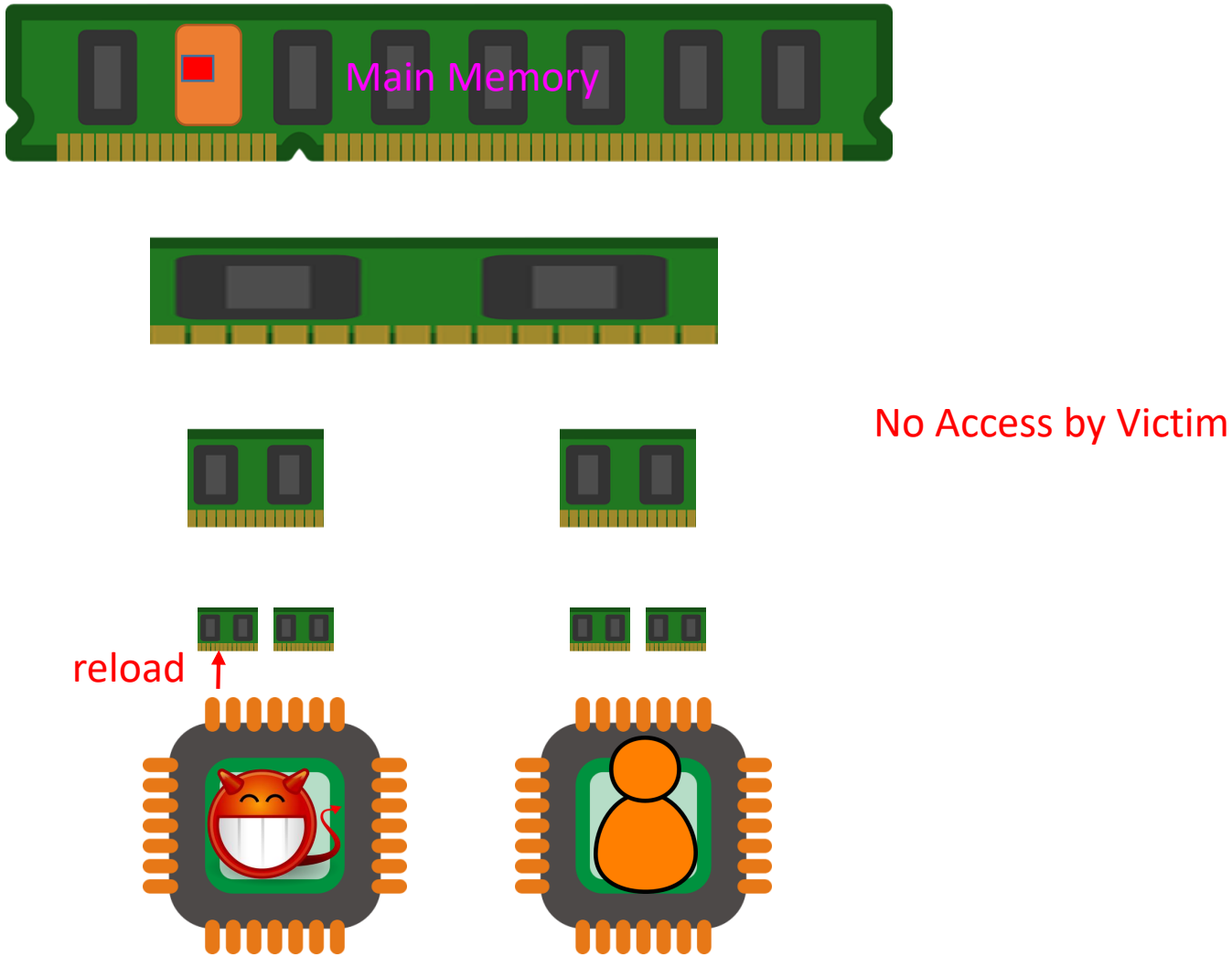
Flush and Reload

1. Flush memory line
2. Wait a bit
3. Measure time to Reload line
4. Repeat



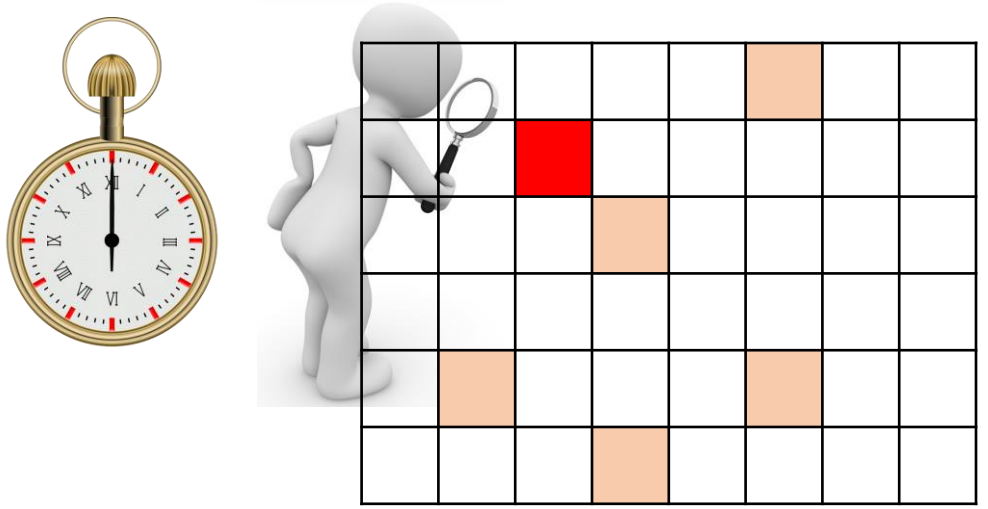


Reload a Line From Cache



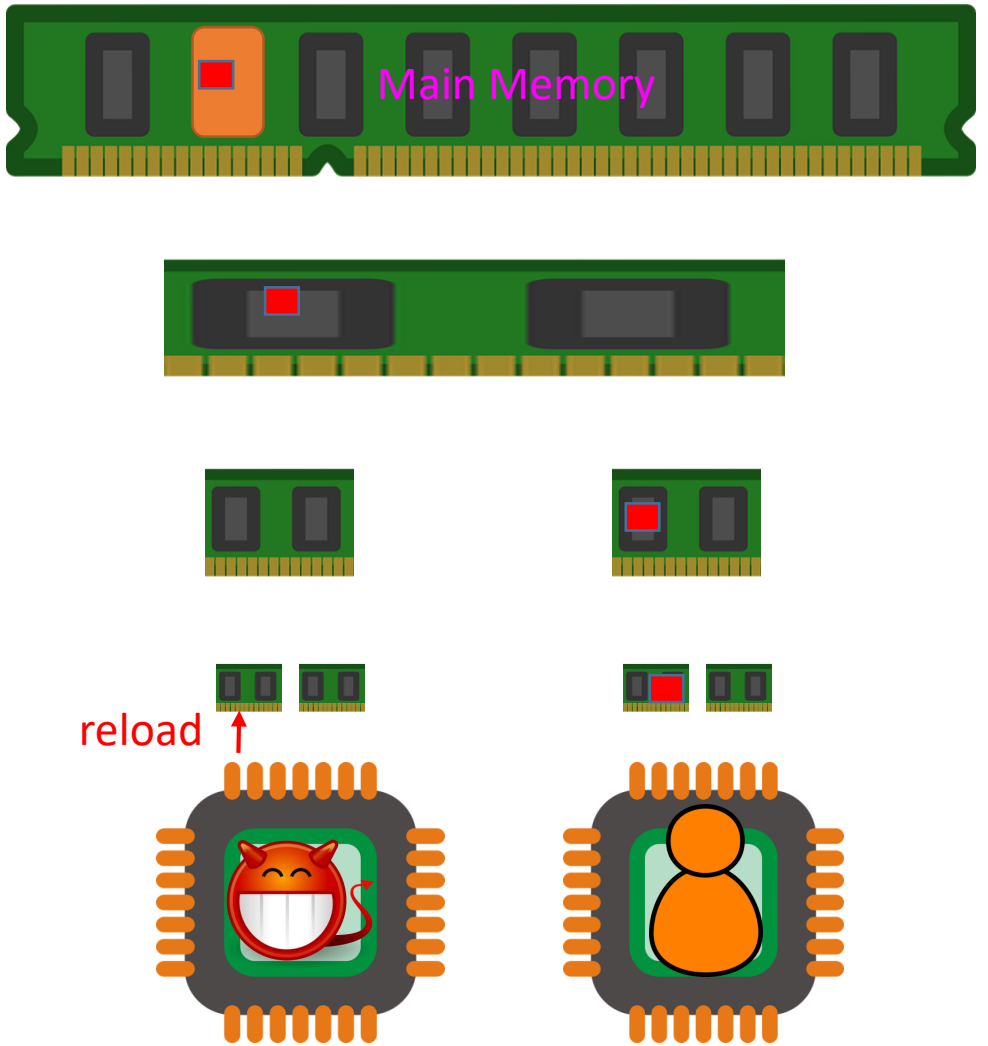
Flush and Reload

1. Flush memory line
2. Wait a bit
3. Measure time to Reload line
4. Repeat



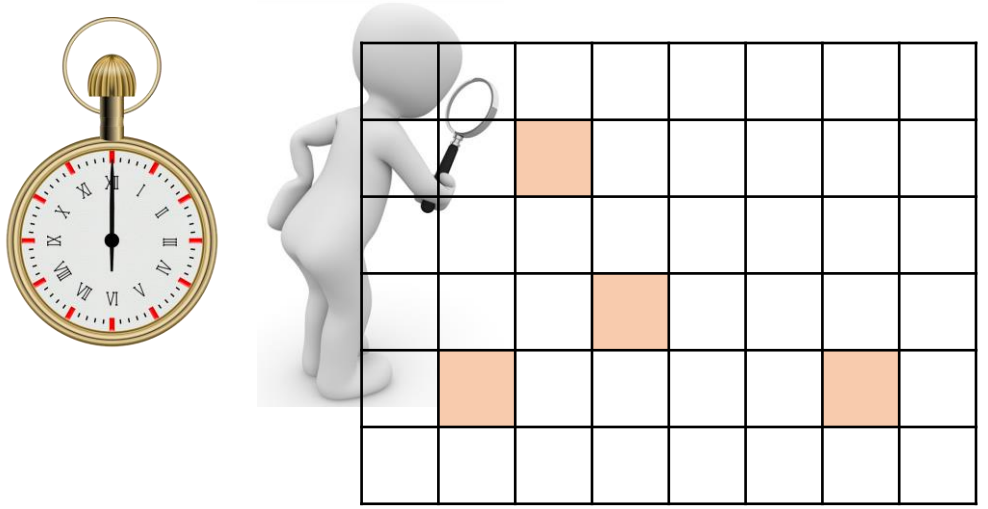
Slow means no access by victim

Reload a Line From Cache



Flush and Reload

1. Flush memory line
2. Wait a bit
3. Measure time to Reload line
4. Repeat



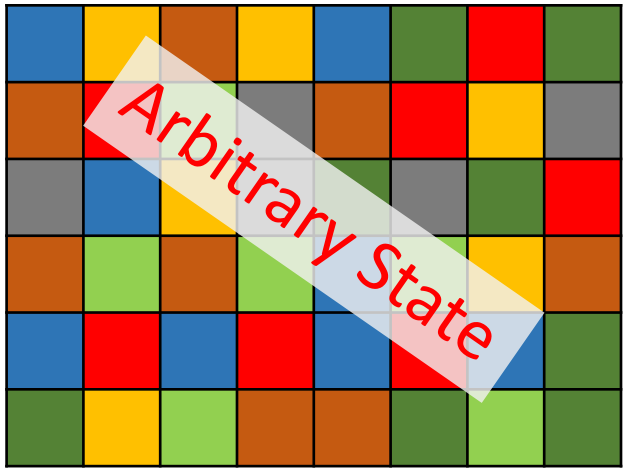
Fast means that victim accessed

Some Cache Attack Technique

- Evict and Time
- Flush and Reload
- Prime and Probe

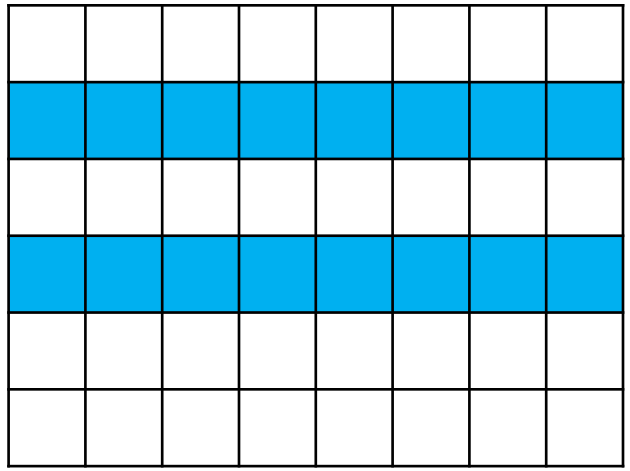
Prime and Probe

1. Attacker fills a set with its own data by accessing some locations in memory
2. Victim Executes and evicts some of the cache lines
3. Attacker accesses those cache line and measure time

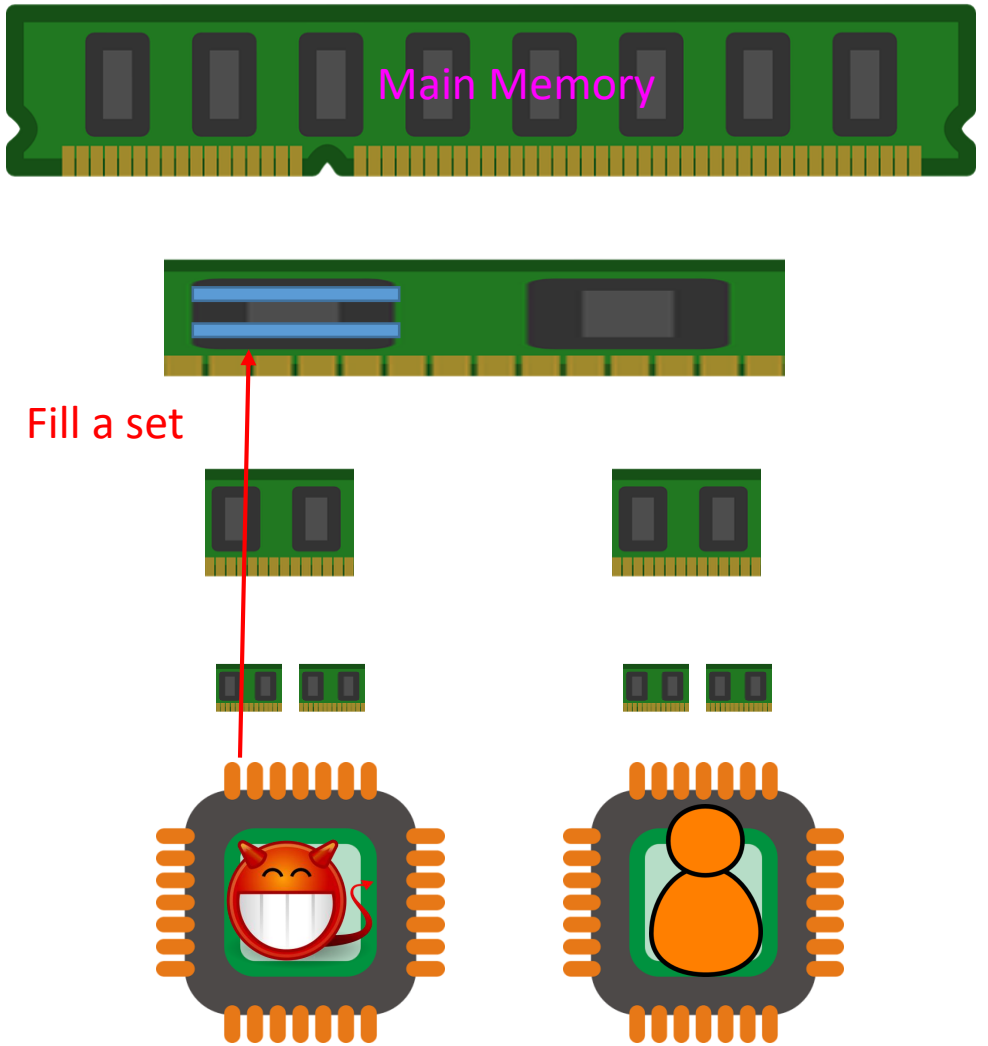


Prime and Probe

1. Attacker fills a set with its own data by accessing some locations in memory
2. Victim Executes and evicts some of the cache lines
3. Attacker accesses those cache line and measure time

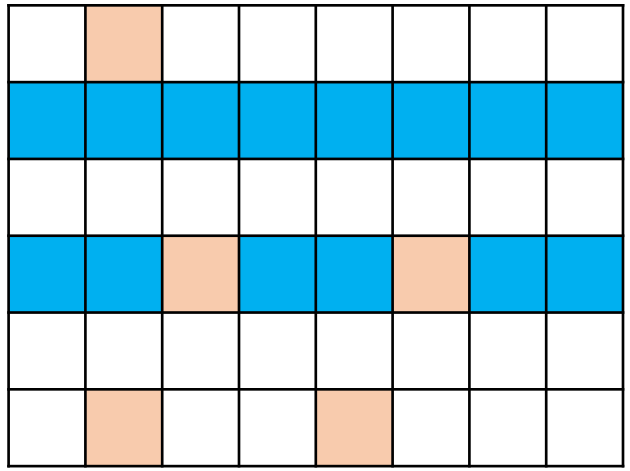


Fill a cache set (In this example 2 sets)

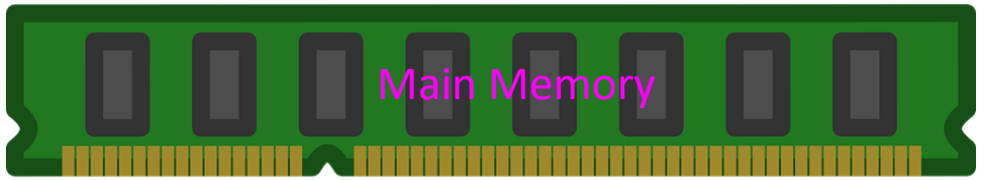


Prime and Probe

1. Attacker fills a set with its own data by accessing some locations in memory
2. Victim Executes and evicts some of the cache lines
3. Attacker accesses those cache line and measure time

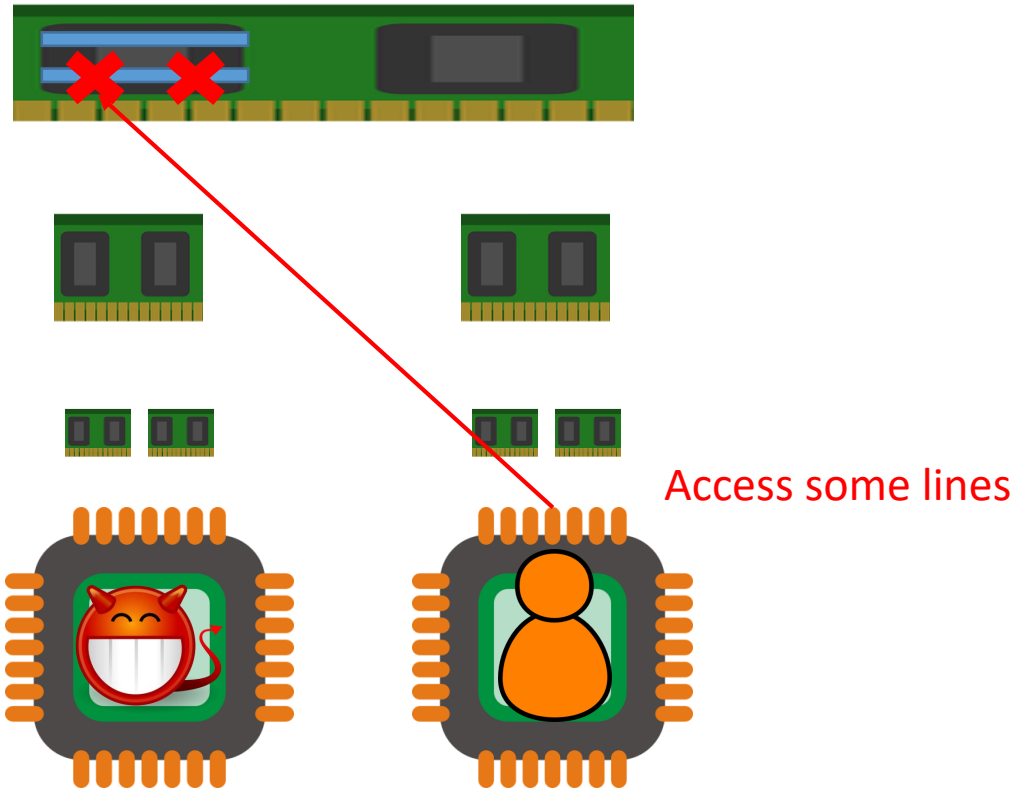


Victim Execution



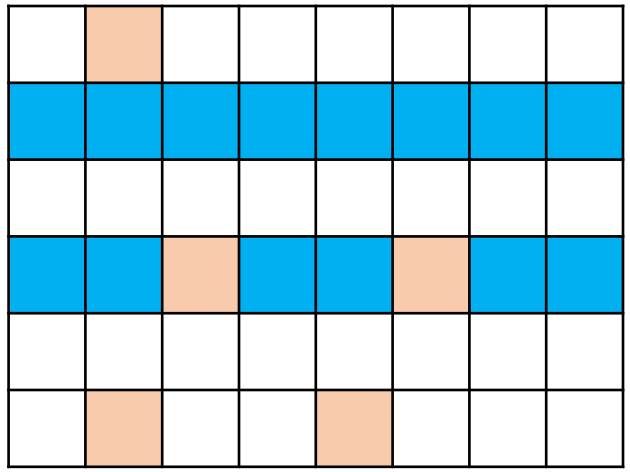
: Attacker's data

: Victim's data

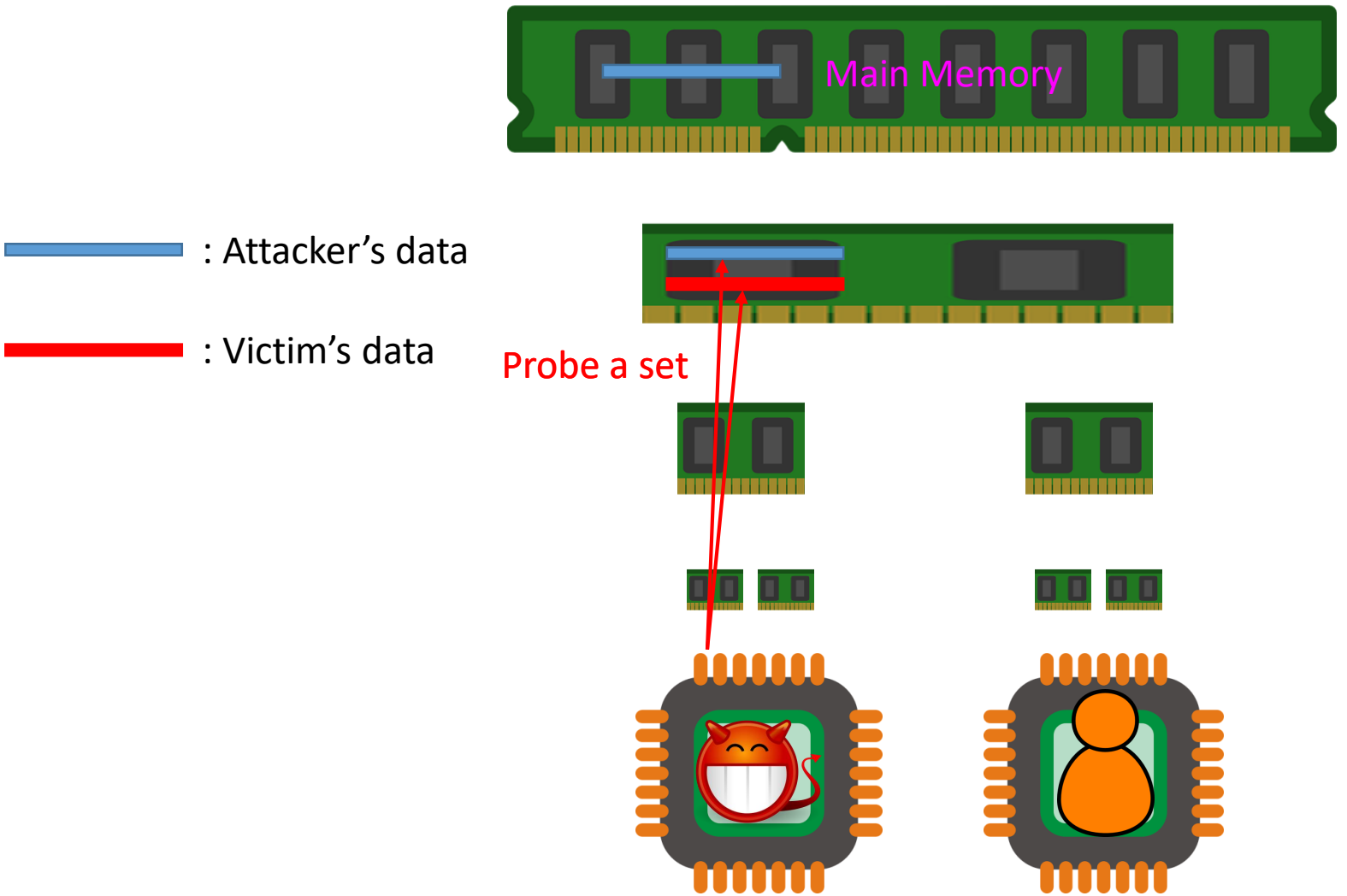


Prime and Probe

1. Attacker fills a set with its own data by accessing some locations in memory
2. Victim Executes and evicts some of the cache lines
3. Attacker accesses those cache line and measure time

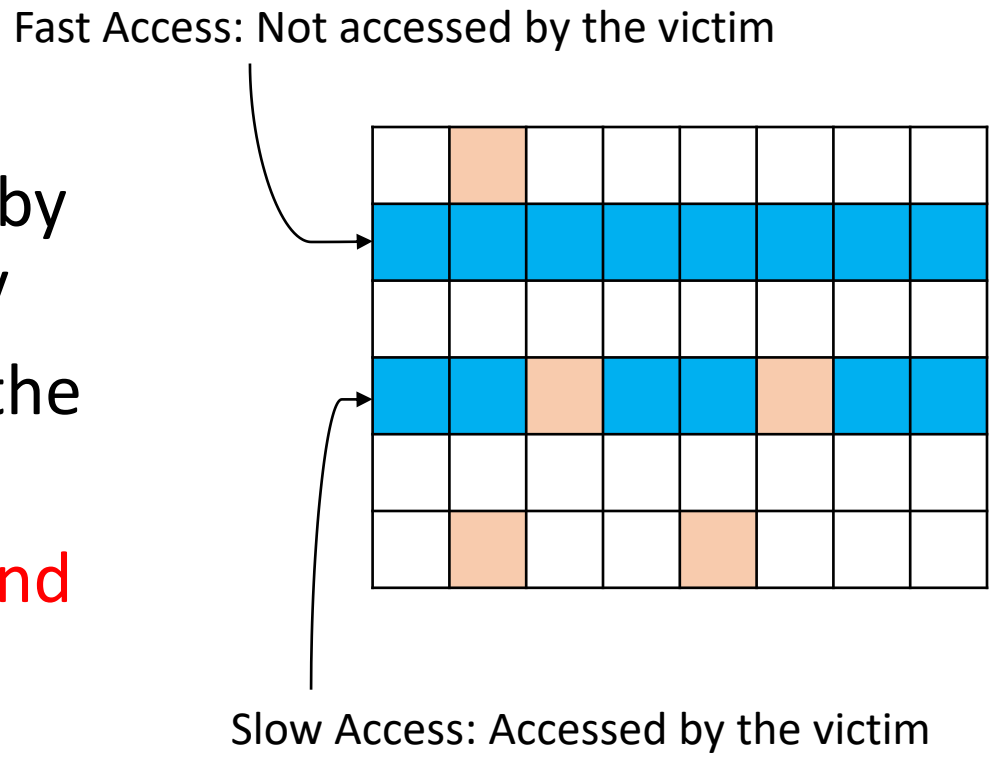


Probe



Prime and Probe

1. Attacker fills a set with its own data by accessing some locations in memory
2. Victim Executes and evicts some of the cache lines
3. Attacker accesses those cache line and measure time



How To Recover Secret Key?

SM S S SM SM S S

1 0 0 1 1 0 0

- How do we compute $b^e mod\ n$?
  - Assume e is secret information we want to recover.
- Bit = 0 : Square
- Bit = 1 : Square + Multiply

✓The Sequence of operation will reveal the secret information.

$x \leftarrow 1$

for $i \leftarrow |e|\text{-}1$ **downto** 0 do

  $x \leftarrow x^2 \bmod n$

  if $(e_i = 1)$ **then**

    $x = xb \bmod n$

  endif

done

return x

A Sample Measurement (Flush and Reload)

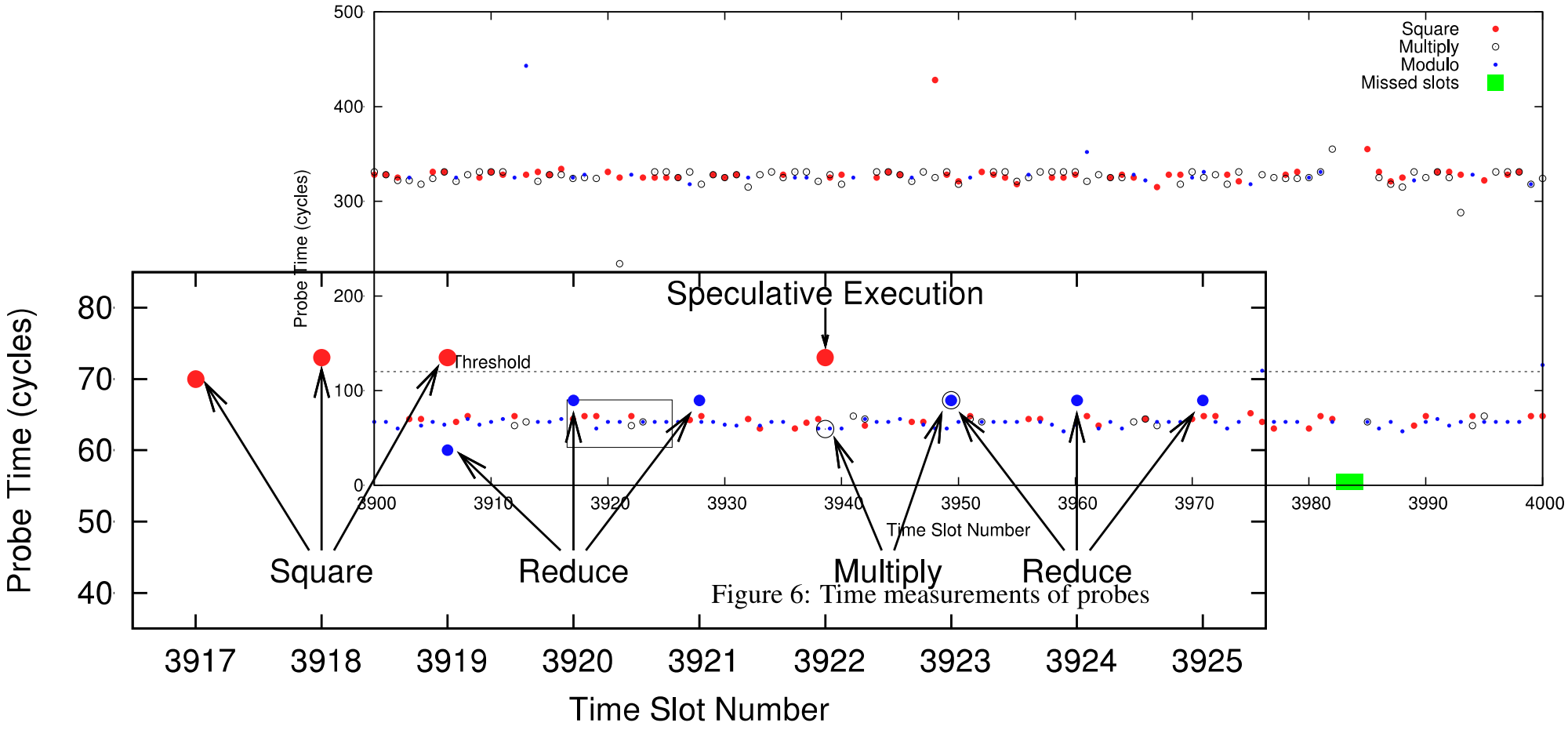
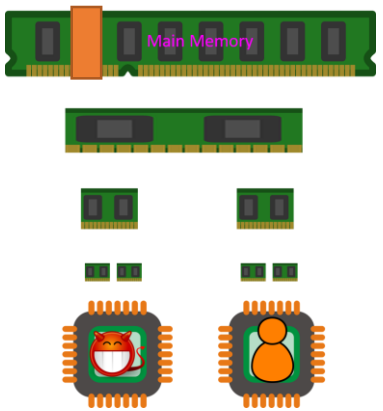
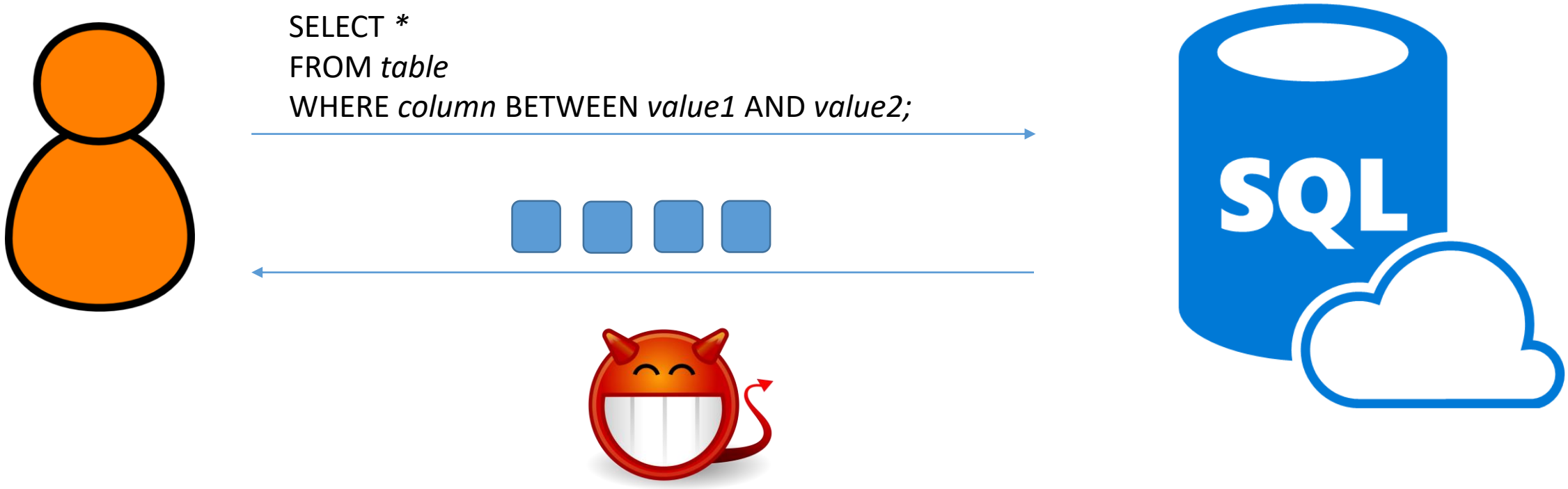


Figure 6: Time measurements of probes



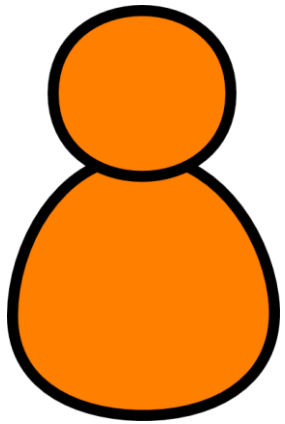
Cache Attack on Database

- User execute queries in the form of range queries
  - Asks for entries with column value between value1 and value 2
- Attacker sees the volume of the responses



Cache Attack on Database

Students	Grade
A	1
B	2
C	2
D	2
E	3
F	3
G	3
H	4
I	4



SELECT * FROM *table* WHERE *Grade* BETWEEN *1* AND *1*

A

SELECT * FROM *table* WHERE *Grade* BETWEEN *1* AND *2*

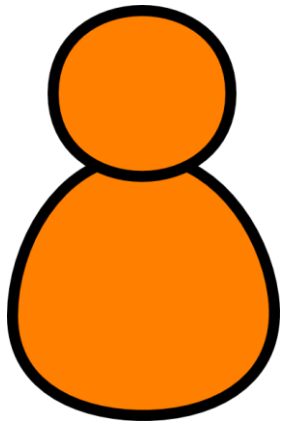
A B C D

SELECT * FROM *table* WHERE *Grade* BETWEEN *3* AND *4*

E F G H I

Cache Attack on Database

Students	Grade
A	1
B	2
C	2
D	2
E	3
F	3
G	3
H	4
I	4

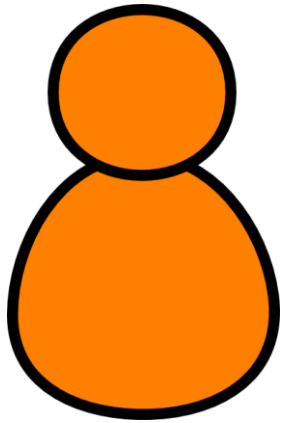


SELECT * FROM *table* WHERE *Grade* BETWEEN *1* AND *1*

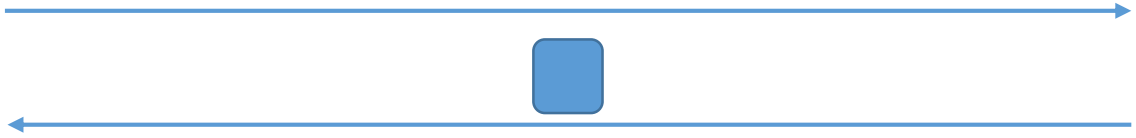
SELECT * FROM *table* WHERE *Grade* BETWEEN *1* AND *2*

SELECT * FROM *table* WHERE *Grade* BETWEEN *3* AND *4*

Cache Attack on Database



SELECT * FROM *table* WHERE *Grade* BETWEEN *1* AND *1*

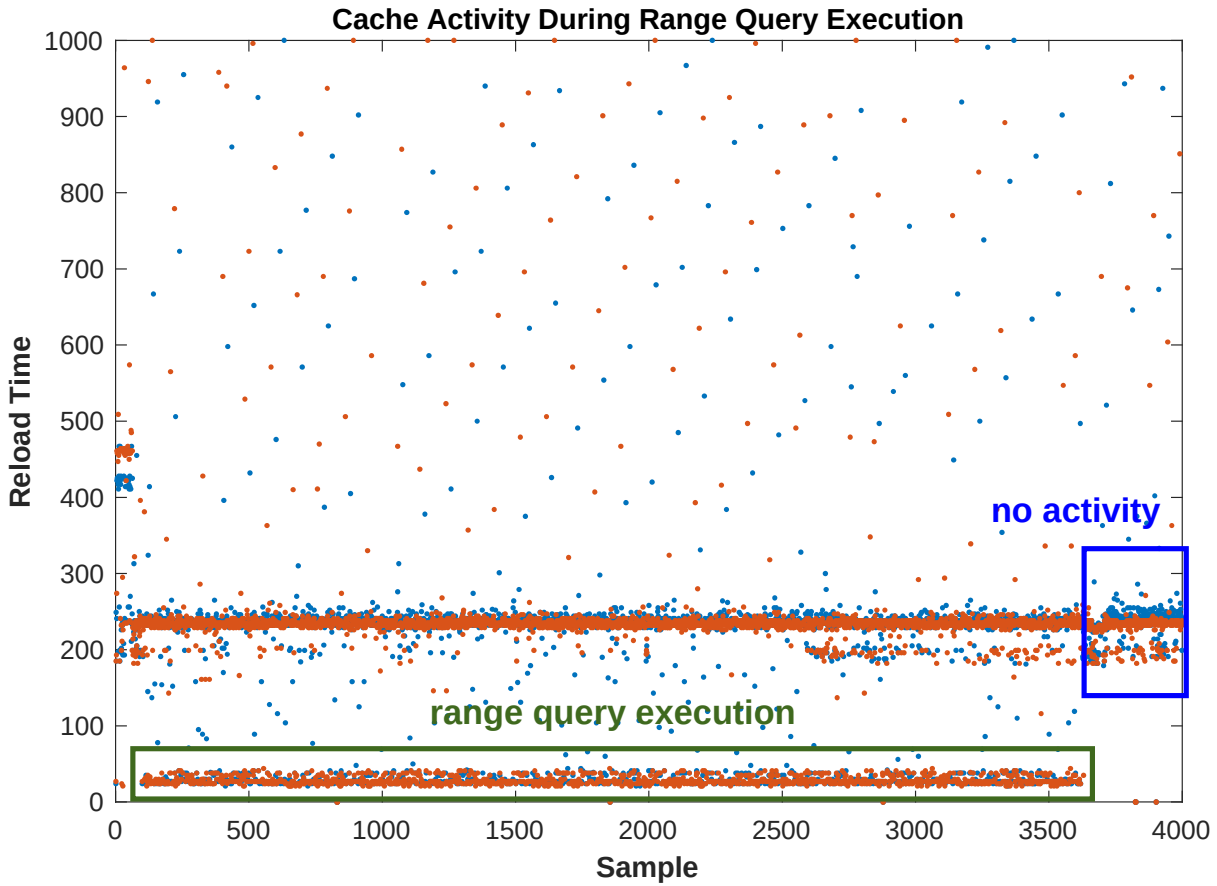


SELECT * FROM *table* WHERE *Grade* BETWEEN *1* AND *2*

SELECT * FROM *table* WHERE *Grade* BETWEEN *3* AND *4*

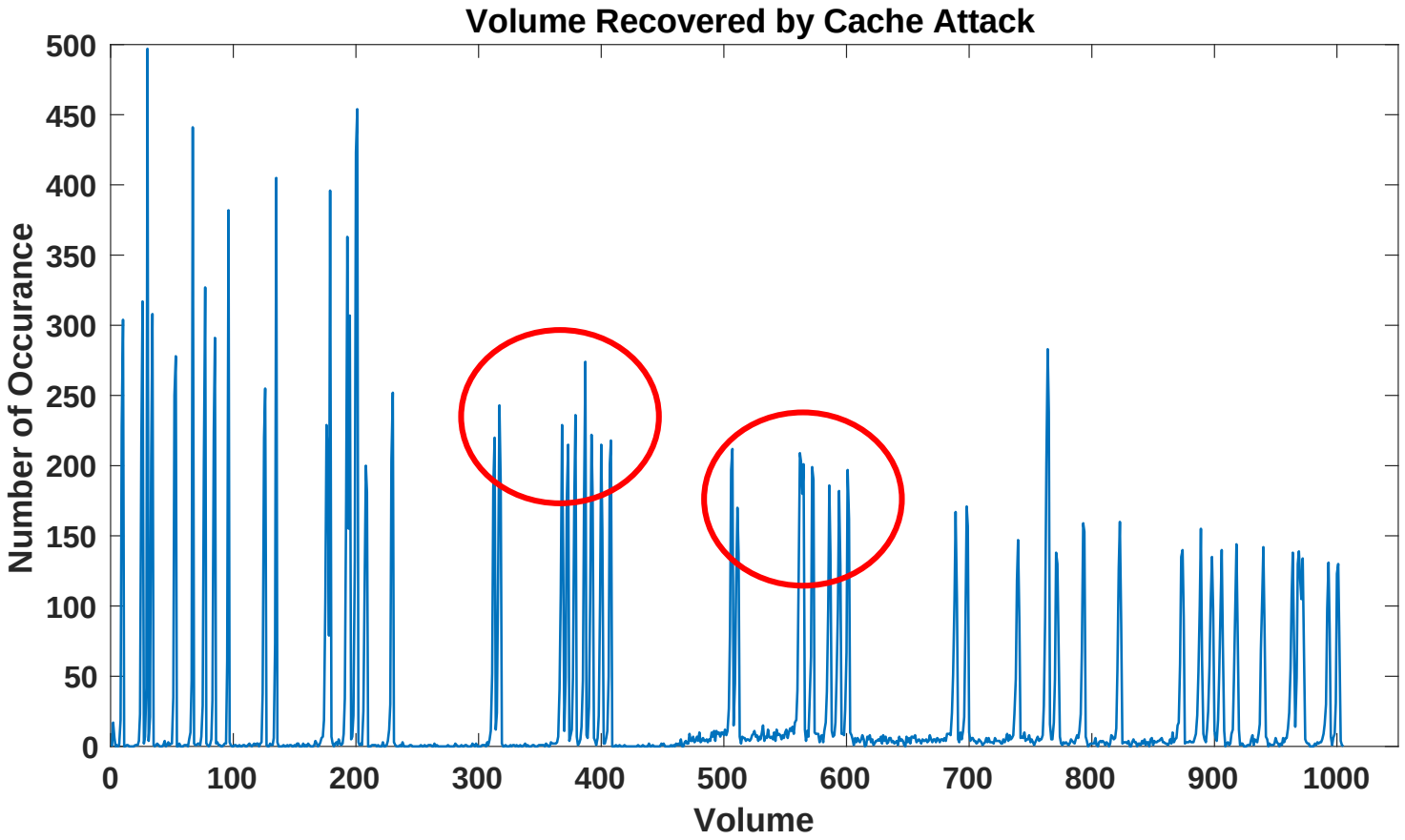
Students	Grade
A	1
B	2
C	2
D	2
E	3
F	3
G	3
H	4
I	4

Attacker View From Cache



- Counts how many time the server execute the line which correspond to returning an entry
- From that, the attacker figures out approximately the value of each volume

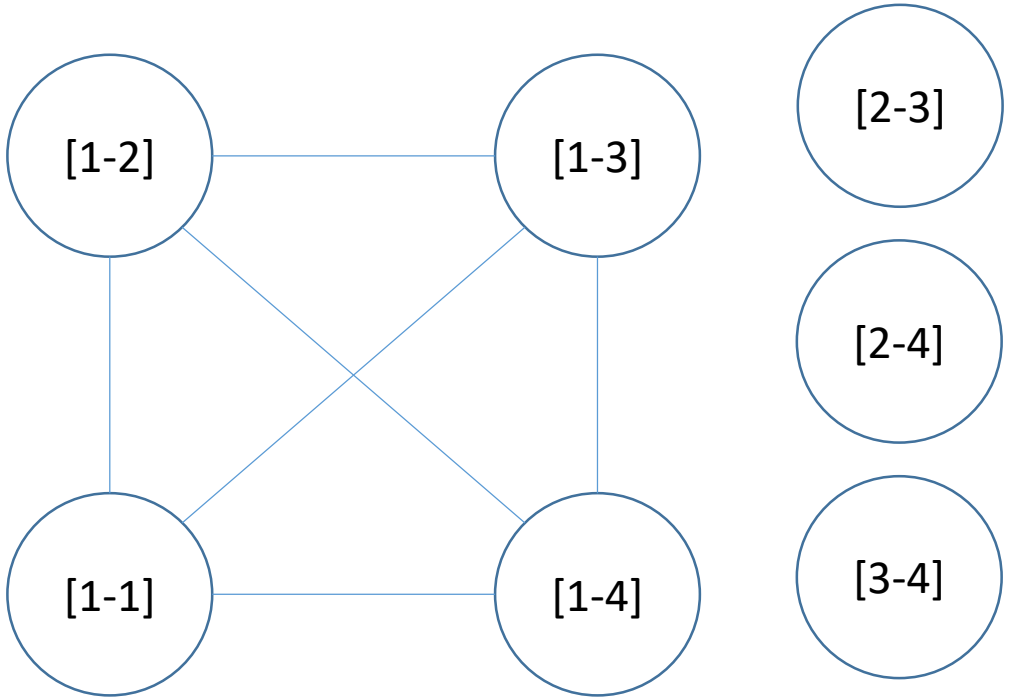
Noisy Volume Recovered

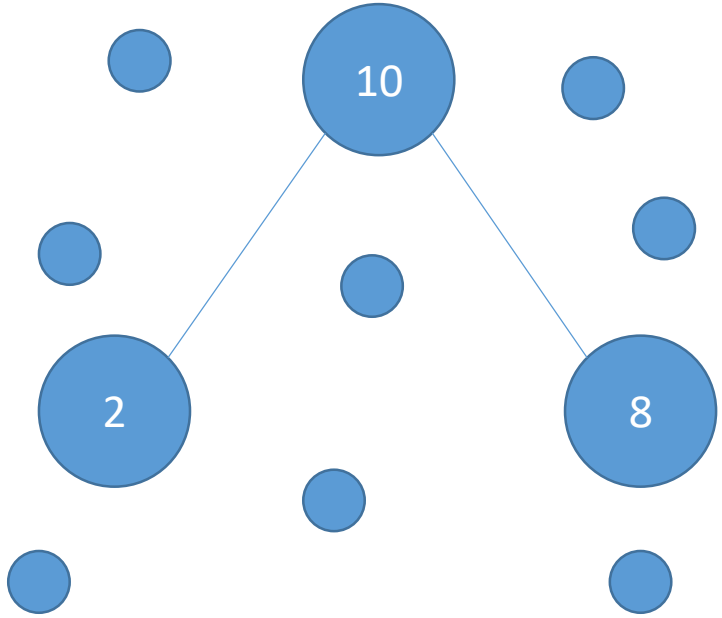


Peaks represent the Volumes

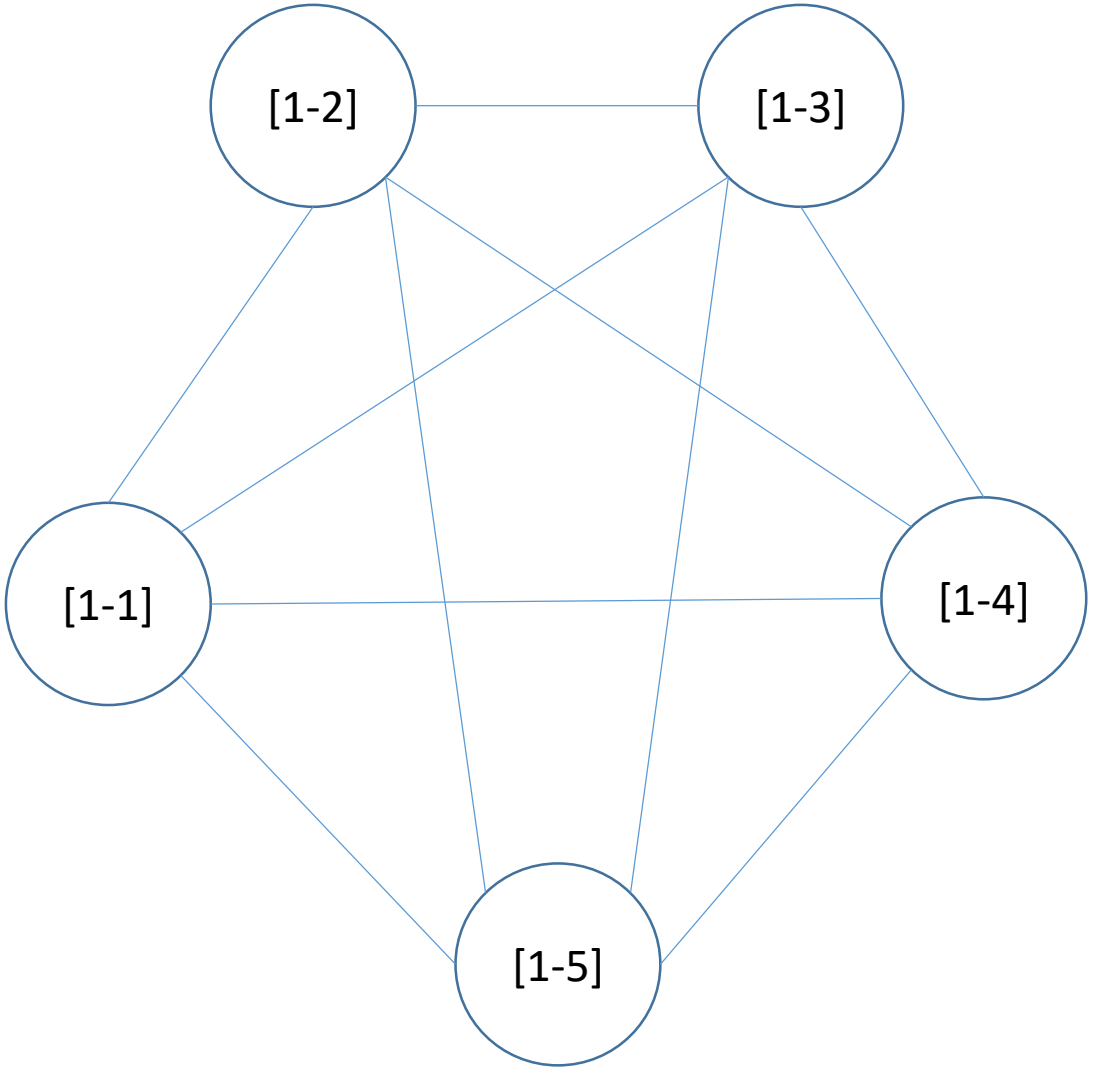
How to reconstruct the database?

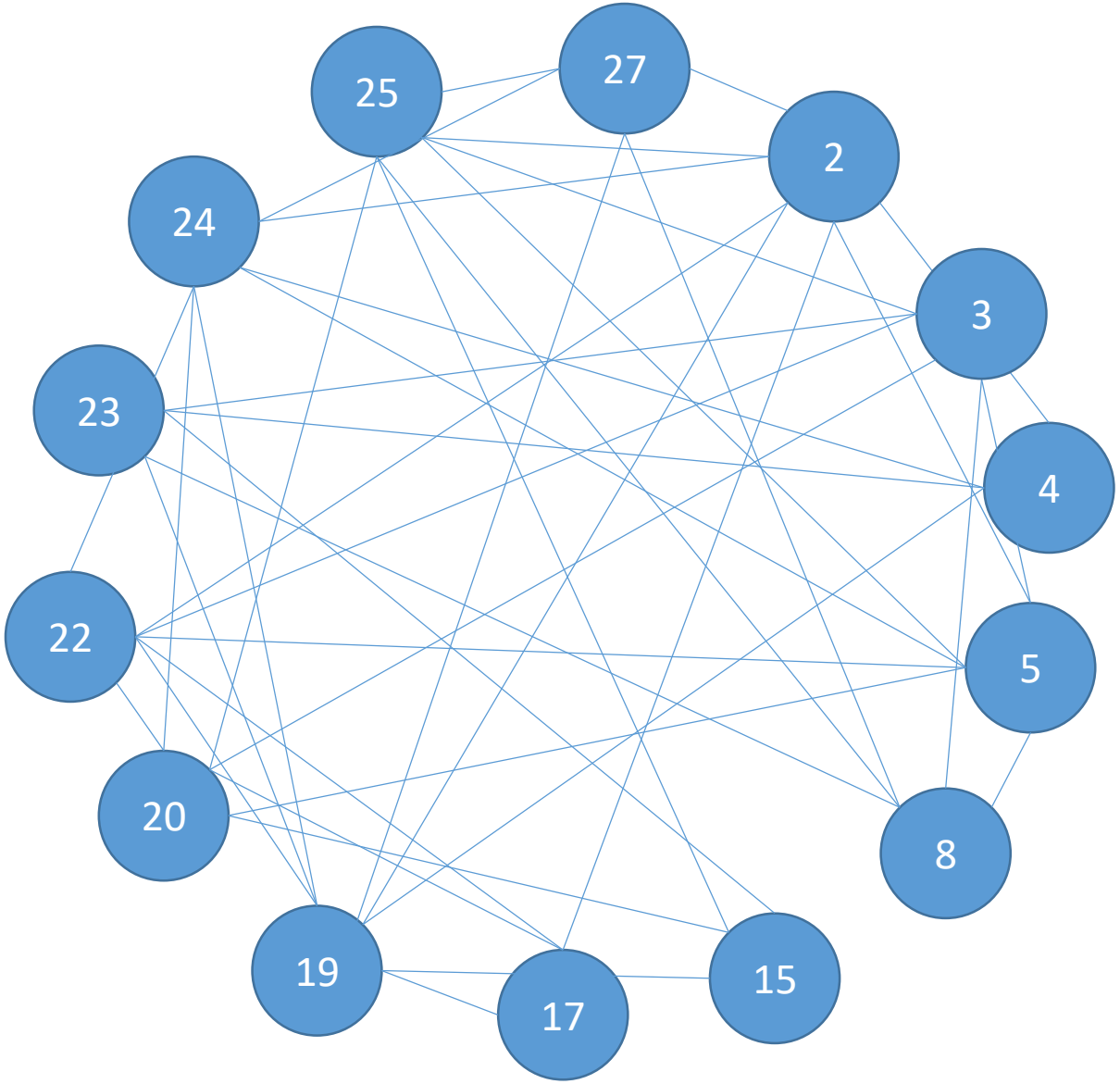
- [1-2] = [1-1] + [2-2]
- [1-3] = [1-2] + [3-3] = [1-1] + [2-3]
- [1-4] = [1-1] + [2-4] = ….

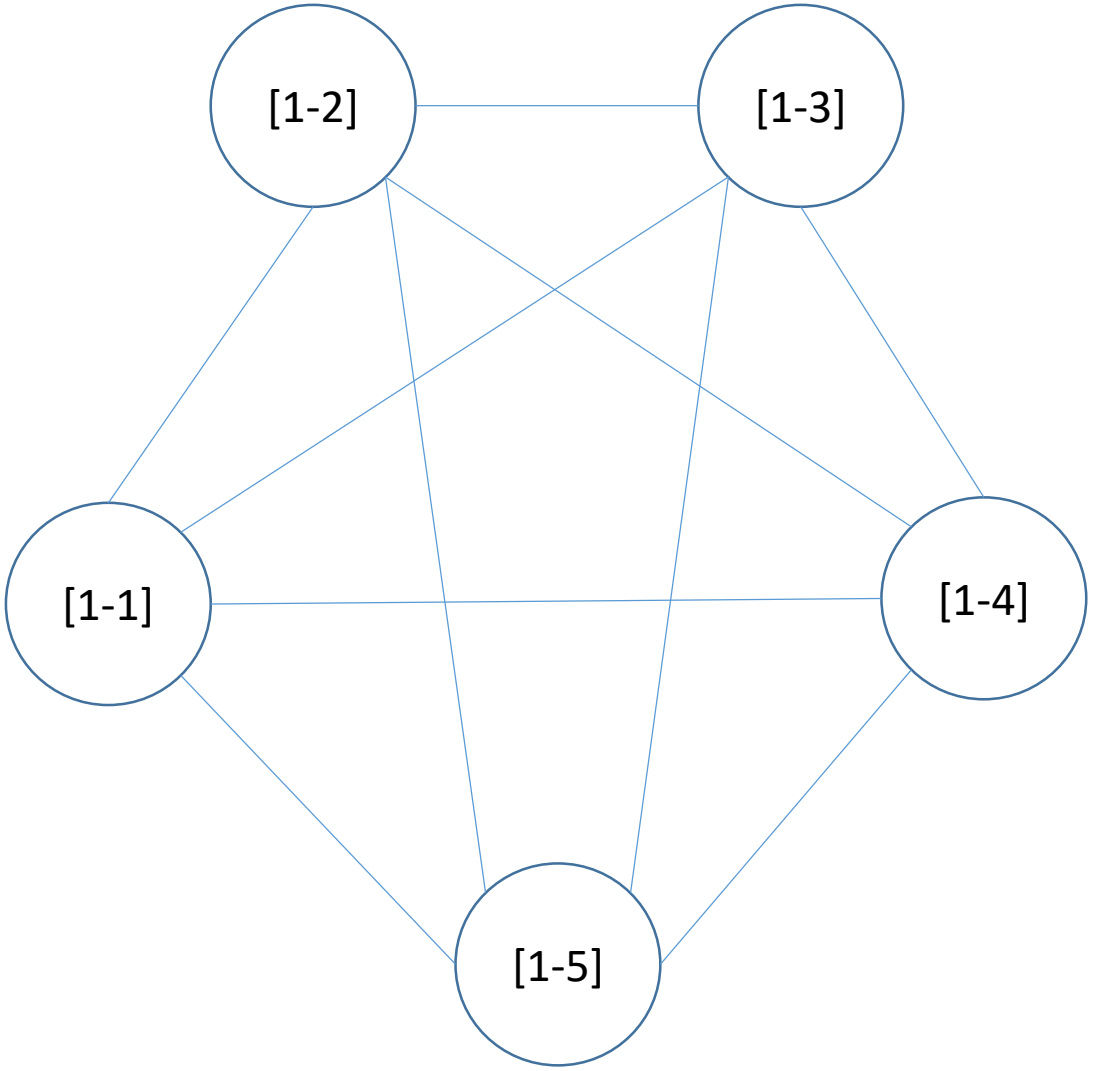


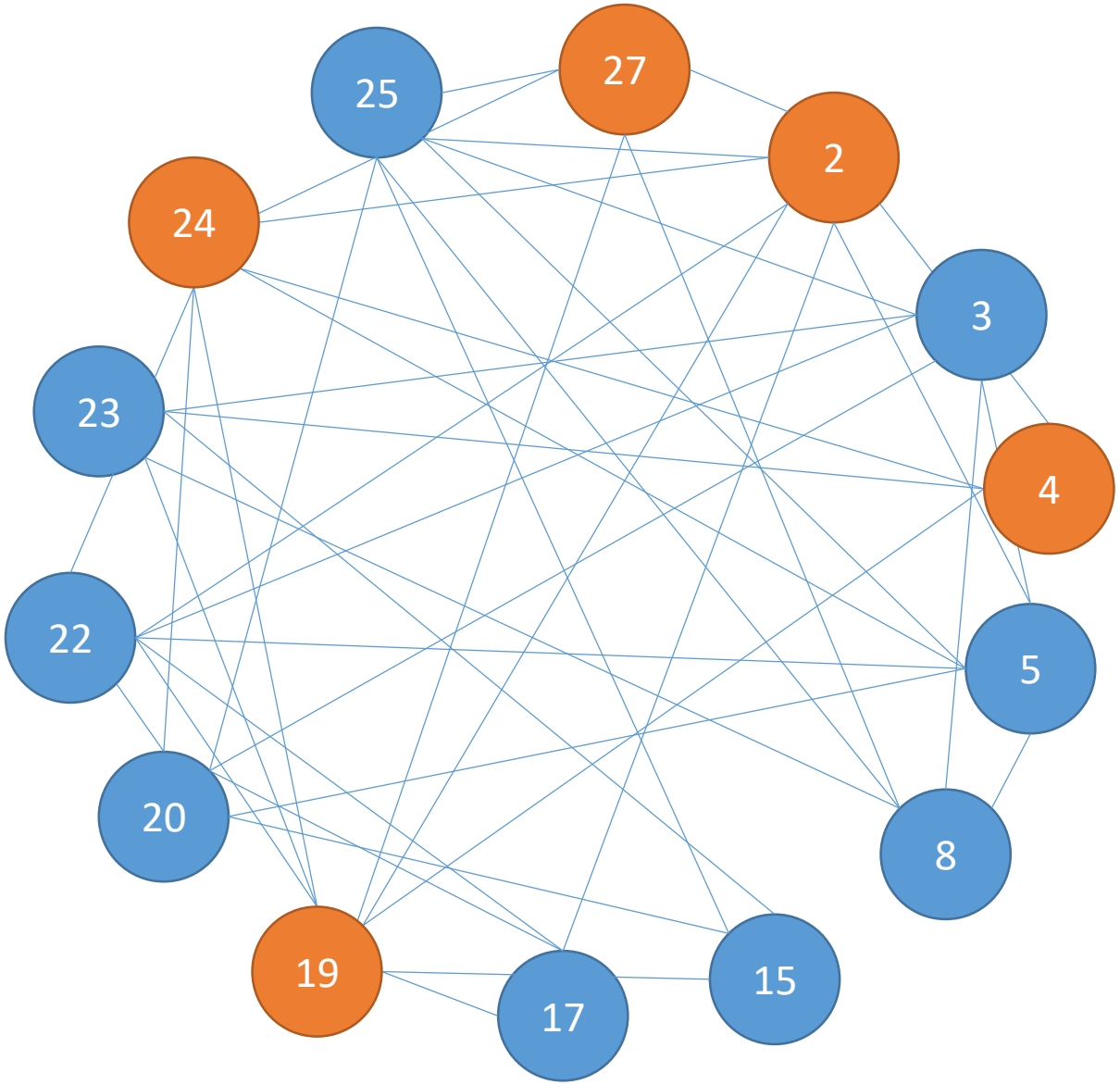


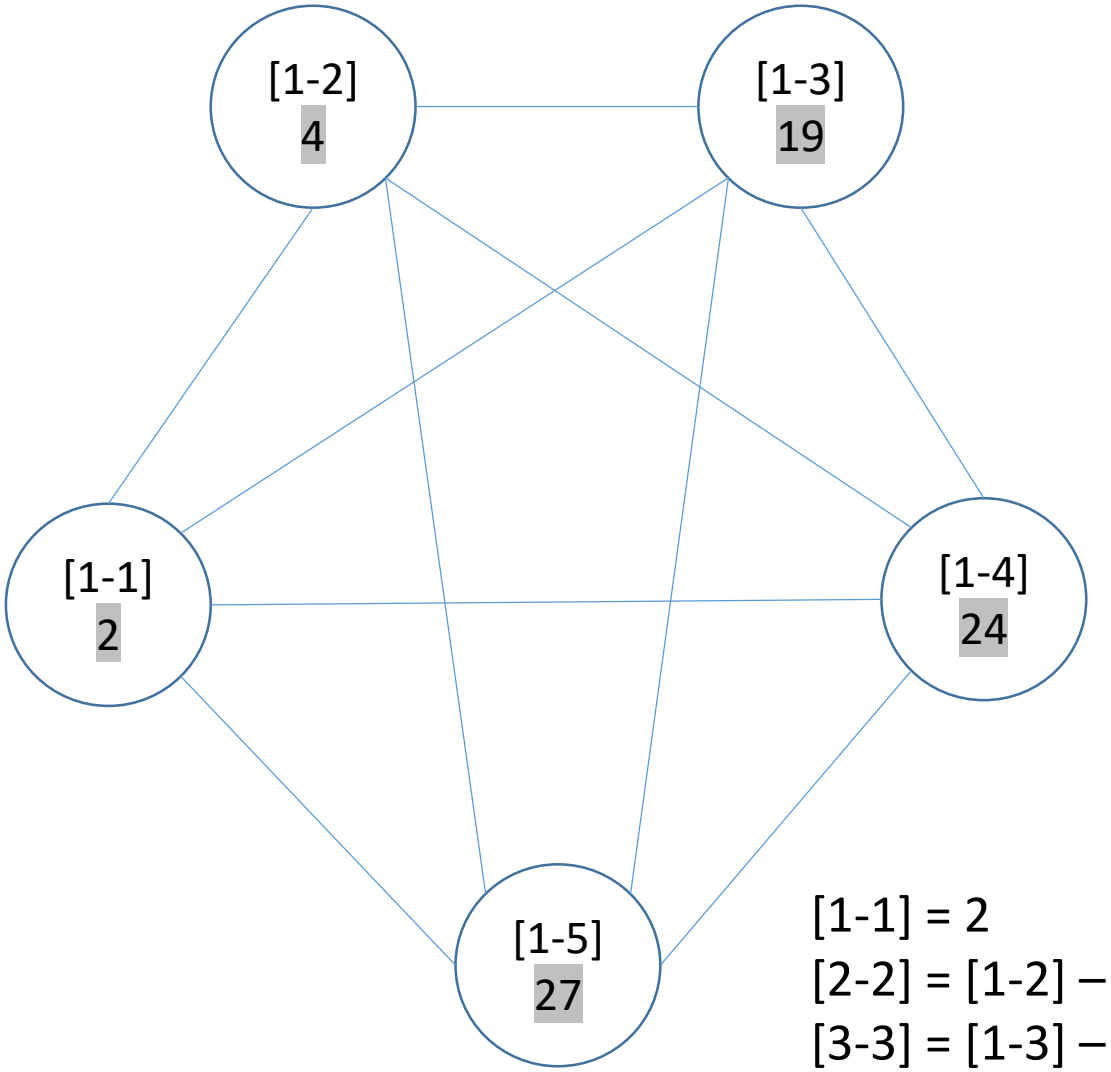
Observed Volumes


[1-2]
[1-3]
[1-1]
[1-4]
[1-5]

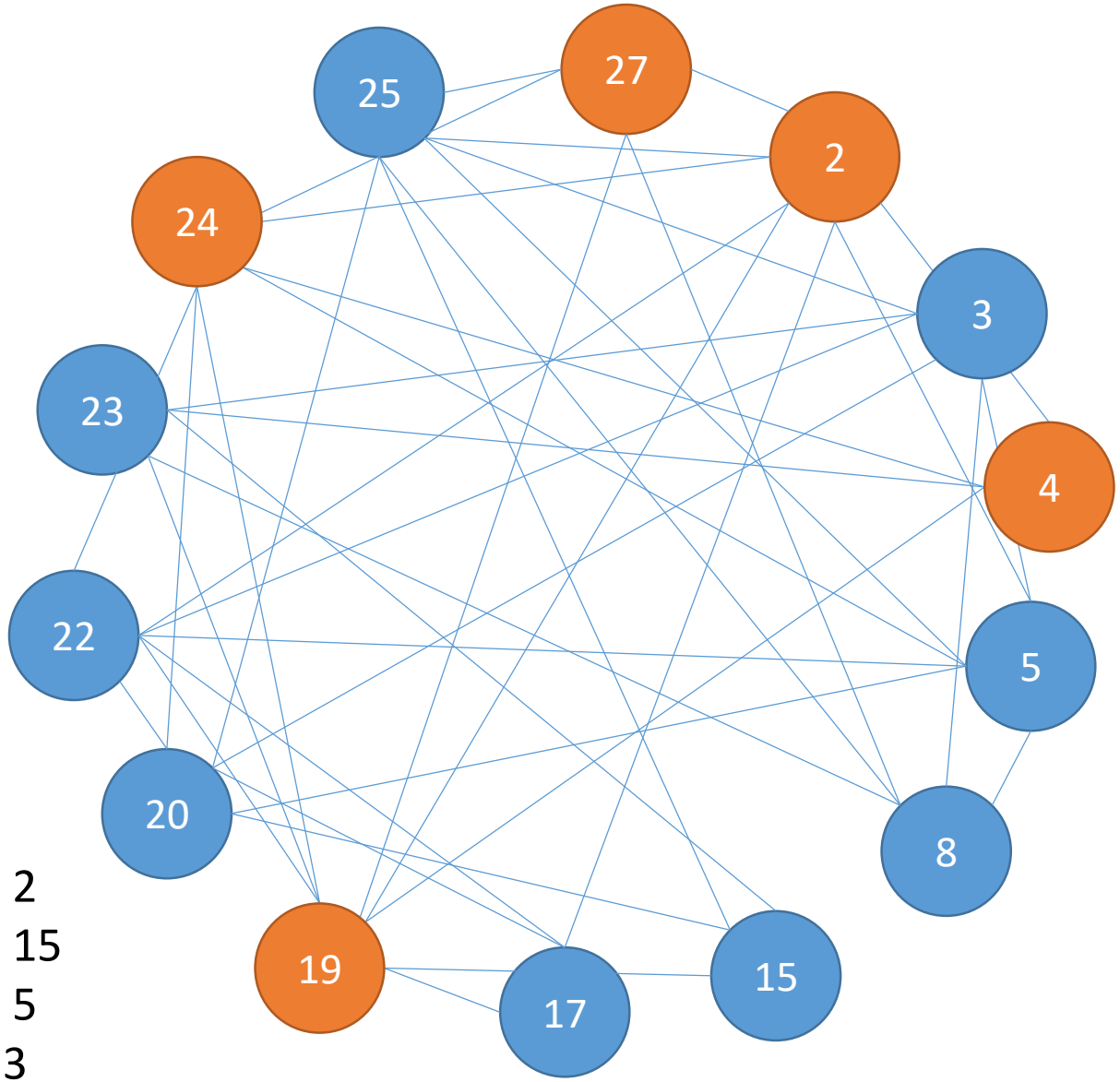

25
27
2
24
3
23
4
22
5
20
8
19
15
17


[1-2]
[1-3]
[1-1]
[1-4]
[1-5]


25
27
2
24
3
23
4
22
5
20
8
19
17
15



[1-1] = 2
[2-2] = [1-2] – [1-1] = 2
[3-3] = [1-3] – [1-2] = 15
[4-4] = [1-4] – [1-3] = 5
[5-5] = [1-5] – [1-4] - 3

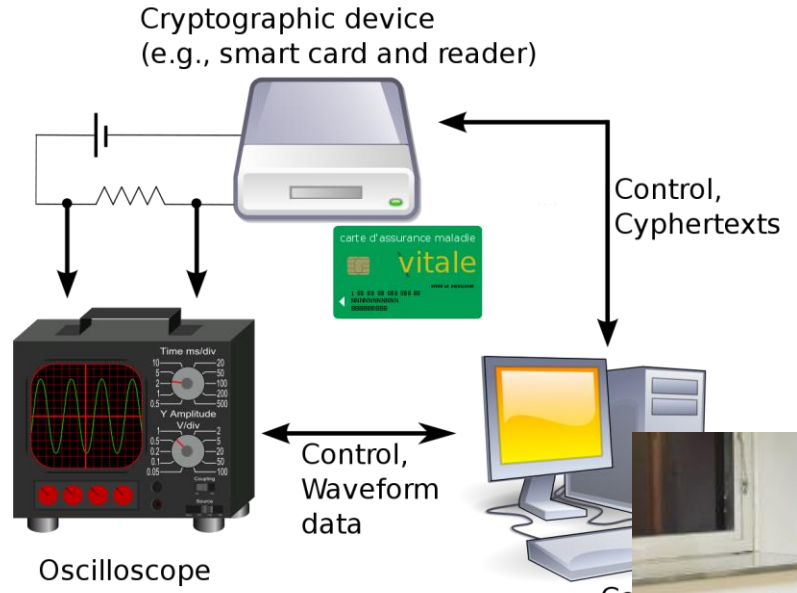


What about the case for measurements from cache

- It is harder because the measurement from cache are noisy
- Some of the volumes might be missing
  - Some of the connections ins graph is missing
- There might be some extra volumes in the graph
  - There are extra nodes in the graph which should not be there
- We still can recover the database in some of the cases

Side Channel Attacks Examples

- Timing Attacks
  - Cache Attack
- Power Analysis Attack
- Electromagnetic Emissions
- Acoustic Emission
- Fault Attacks



Thank You