

Information Security - 5 - Secure Systems Engineering
Professor Chester Rebeiro
Indian Institute of Technology, Madras
Covert Channels

Hello and welcome to this lecture in the course of secure systems engineering in this lecture we will look at a new form of attack known as micro architectural attacks.

(Refer Slide Time: 0:32)



The slide features a yellow smiley face emoji on the left. The title 'Things we thought gave us security!' is written in a reddish-brown font. Below the title is a white box with a black border containing a bulleted list of security measures. A small number '2' is visible in the bottom right corner of the slide.

- Cryptography
- Passwords
- Information Flow Policies
- Privileged Rings
- ASLR
- Virtual Machines and confinement
- Javascript and HTML5 (due to restricted access to system resources)
- Enclaves (SGX and Trustzone)

So the in the course that we have seen so far we had actually studied various things that we thought would actually increase the security of the system for example we had looked at cryptographic algorithms and by means of encryption and decryption these cryptographic algorithms would actually be used to obtain confidentiality and integrity of the system we have known that passwords and information flow policies can be used to restrict how information flows in the system and we have also seen that hardware aspects such as the privileged rings present in modern processors and enclaves such as the SGX and trust zone which are present in Intel and ARM processors respectively have been used at the hardware level to increase security of the system.

We have also looked at various techniques by which we could have confinement wherein a malicious program could be executed in a system and the confined environment Keep makes sure that there is a sandbox and the confined environment creates a sandbox so that the malicious program does not influence any aspect of the system outside of that sandbox we had also seen

that web browsers and web servers make use of JavaScript as the programming language essentially because JavaScript is a programming language which is highly restrictive and therefore access to the system in which a JavaScript program executes is very limited.

(Refer Slide Time: 2:25)

Micro-Architectural Attacks
(can break all of this)

- Cryptography
- Passwords
- Information Flow Policies
- Privileged Rings
- ASLR
- Virtual Machines and confinement
- Javascript and HTML5 (due to restricted access to system resources)
- Enclaves (SGX and Trustzone)

- Cache timing attack
- Branch prediction attack
- Speculation Attacks
- Row hammer
- Fault Injection Attacks
- cold boot attacks
- DRAM Row buffer (DRAMA)
- and many more

3

In this particular lecture we will be looking at a new form of attack known as micro architectural attacks now the interesting thing about micro architectural Attacks is that they can break all of these things that we have actually talked about so for example micro architectural attacks have been used to break cryptographic schemes where in the secret key of the crypto scheme has been retrieved by means of a micro architectural attack similarly passwords and the information flow policies such as the Bell la Padula models have been broken by these attacks similarly privileged links enclaves ASLR and so on have all been broken by macro micro architectural attacks.

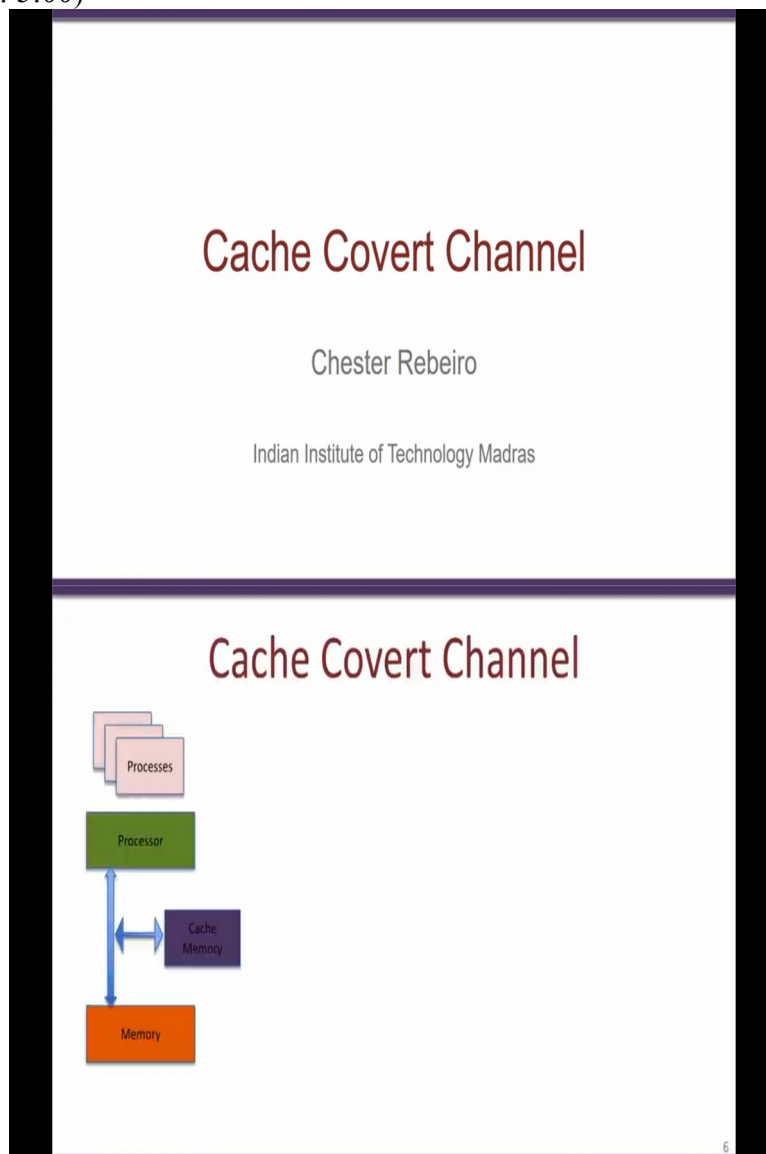
So a micro architectural attack is essentially a class of different types of attacks for example this list away here provides some of the famous micro architectural attacks so the cache timing branch prediction row hammer types of attacks are all micro architectural attacks.

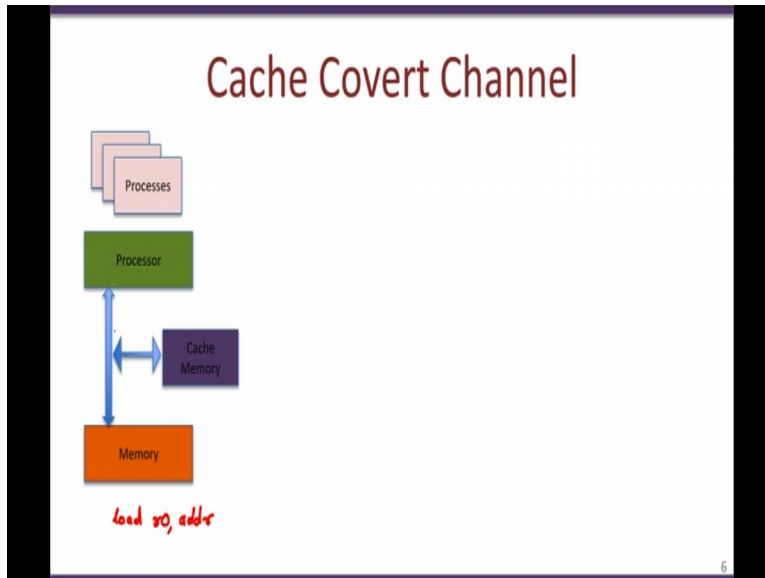
So in this particular course we'll be first looking at the cache timing attacks and we'll have also have a brief overview of speculation attacks but in this but specific lecture we would have an introduction to what a micro architectural cache timing attack can do with respect to the information flow policies so in particular we have seen the various information flow policies like the Bell la Padula and PIBA model and what we had actually studied in the previous lecture was that each of these models could respect how information can flow for example in the Bell la

Padula model we had different levels ranging from top secret to confidential and so on and the rules provided by the Bell la Padula model decided how information should be flowing from a one level to another level for example the model defined that I users present in a lower level such as an unprivileged or unclassified user would not be able to read a top secret document.

So we would start with something known as a covered we look at something known as a cache covert channel and we would see how this covered channel could be used to break information and we would see how schemes such as the Bell la Padula model can be broken using cache timing attacks.

(Refer Slide Time: 5:00)





So let us start with a cache covert channel so before we go into water cache covert a channel is we will have a brief introduction to what a cache memory is and why it is used so a cache memory sits between the processor and the main memory so it caches recently used data and instructions so the reason for the cache memory is that loads and stores from the main memory is extremely slow therefore or the cache memory which can be accessed at a much faster rate than the main memory would be able to service loads and store requests from the processor at a much faster rate so for example we have a load instruction say load to register are not from some address.

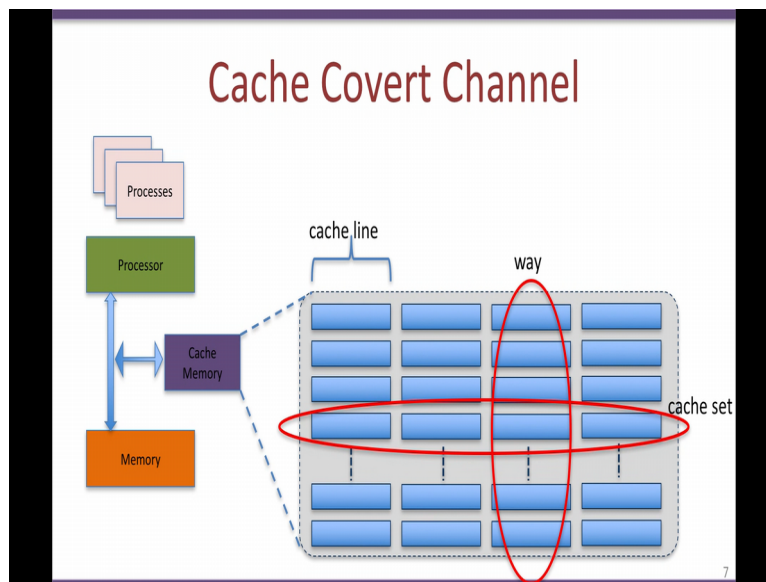
And let us say that this instruction is executed for the first time so and then what would happen is the data corresponding to that address which is massive present in the main memory is loaded into the processor side by side also that same data and data which is adjacent to this particular address is stored in the cache main memory for a subsequent load instruction which is going to an editor to the same address or an address very close to this particular address which is stored in the cache the processor would directly fetch that data from the cache so this fetching from the cache memory is considerably faster and we call it a cache hit now the problem arises.

Because the cache memory is considerably smaller than main memory a typical L1 cache for example is 32 kilobytes while the main memory could be as large as several megabytes so therefore the cache memory stores a small subset of the main memory and there is a replacement policy by which data from the cache is evicted and new data from the memory which is recently being accessed is stored in the cache memory and therefore with every load instruction that is

executed by this processor we could either have a cache hit which would imply that the data is present in the cache memory and the data can be read directly from the cache memory to the processor or we can have a cache miss in which case the data is not present in the cache memory and therefore has to be fetched from the main memory now the critical part over here are two aspects first the cache memory is shared by various processes present which is executing on the system and secondly the cache memory is considerably smaller than the main memory and therefore there is a notion of eviction wherein the data from present in the cache is evicted a new data which is recently accessed is stored in the cache memory so as a result we have something known as cache hits and cache misses.

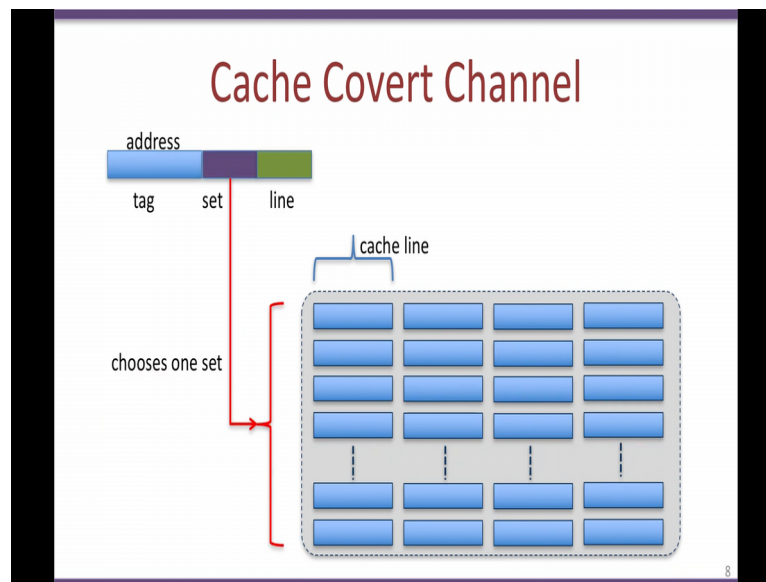
So a cache hit is considerably faster than a cache miss and the reason being that the cache hit fetches data straight from the cache memory while a cache miss would have to fetch data from the main memory or any other lower cache that may be present now we would have to look at some more internal organization about the cache memory so a typical cache memory is organized into several cache sets.

(Refer Slide Time: 8:50)



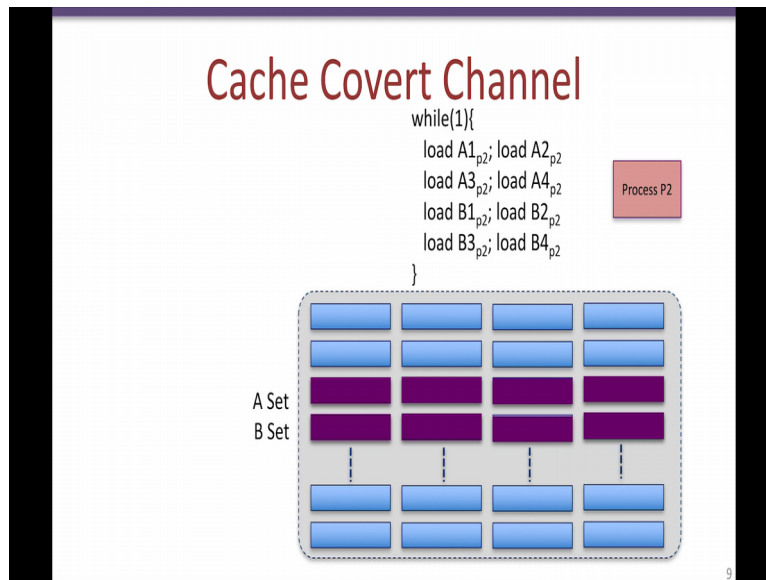
And each cache set or as we see over here each row over here corresponds to a cache set so each cache set could hold multiple cache lines so each of these blocks over here is a cache line a typical cache line size in an Intel processor for instance is around is 64 bytes this particular diagram over here shows that there are 4 cache lines present in a cache set therefore the associativity of this particular cache is full.

(Refer Slide Time: 9:30)



Now whenever the processor executes an load or a store instruction it puts out an address on the address bus so this could in an L1 cache this for example would be an a would be a virtual address and the cache memory the L1 cache memory interprets this virtual address in this particular way there are a few bits are the most least least significant bits which are used to address a cache line there are bits which are known as the set address bits which are used to pick one of the cache sets from this particular cache line and there are the tag bits so what happens is that every time an address is put on the address bus by the processor the set address bits that is this range over here chooses one of these cache sets then the tag bits in the address that is these bits over here is used to determine if the data corresponding to this address is present in this set now if indeed is present we get what is known as a cache hit and the data corresponding to that address is fetched from that corresponding cache line on the other hand if you do not find that tag present in any of these cache lines corresponding to that set then we say that there is a cache miss the lower levels of memory either the DRAM or lower levels of the cache like the L2 or the L3 cache would require to service that particular load request.

(Refer Slide Time: 11:13)

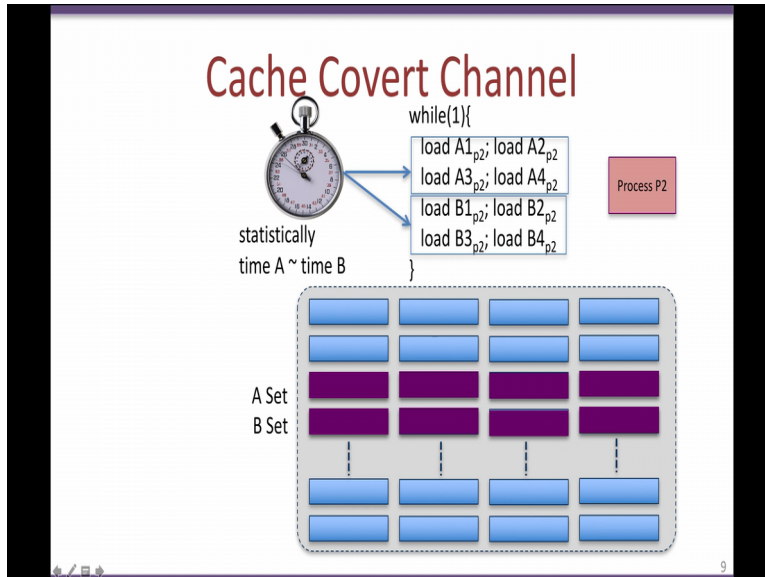


Now let's look at what would happen when we run a program like this so let us say we have process P2 and this process P2 comprises of multiple load instructions so we have impact eight load instructions load A1 A2 A3 and A4 and load B1 B2 B3 and B4 further let us assume that the set address bits corresponding to the addresses A1 A2 and A3 and A4 corresponding to this set known as the A set so thus when for the first time you execute this particular while loop the first iteration of this particular while loop the data corresponding to address A1 gets loaded into the set into one of these cache lines present in the A set similarly the first execution of load A2 load A3 and blade load A4 would fetch data and store it in this A set thus the first execution would all the cache misses right similarly we have another for load instructions load B1 B2 B3 and B4 which get mapped to this B set so at the end of the first iteration of this particular while loop we have all the 4 cache lines in the be set filled with this data corresponding to these four loads now the critical aspect over here is the timing of this particular while so you notice that during the first iteration you would get all cache misses for A as well as B so in total for this first iteration corresponding to these 8 load instructions there will be a total of 8 cache misses.

Now for the second iteration onwards assuming that there is no other process running in the system all of these load operations would result in cache hits the reason being that every subsequent load to say A1 A2 A3 or A4 would directly read the data from the corresponding

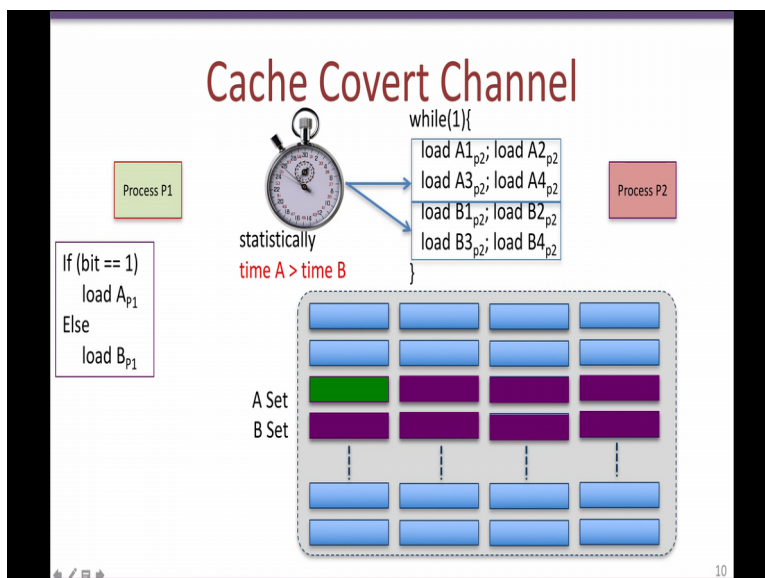
cache line present in the A set similarly every subsequent load to B1 B2 B3 or B4 would directly read the data from this B set.

(Refer Slide Time: 13:43)



The second thing you would notice is that the time taken for these loads versus these loads is approximately the same note that we are using the word statistically over here which would mean this particular loop is run several thousands of times and the time taken for these four loads and these four loads are compared statistically right for example taking the average variance and so on.

(Refer Slide Time: 14:08)

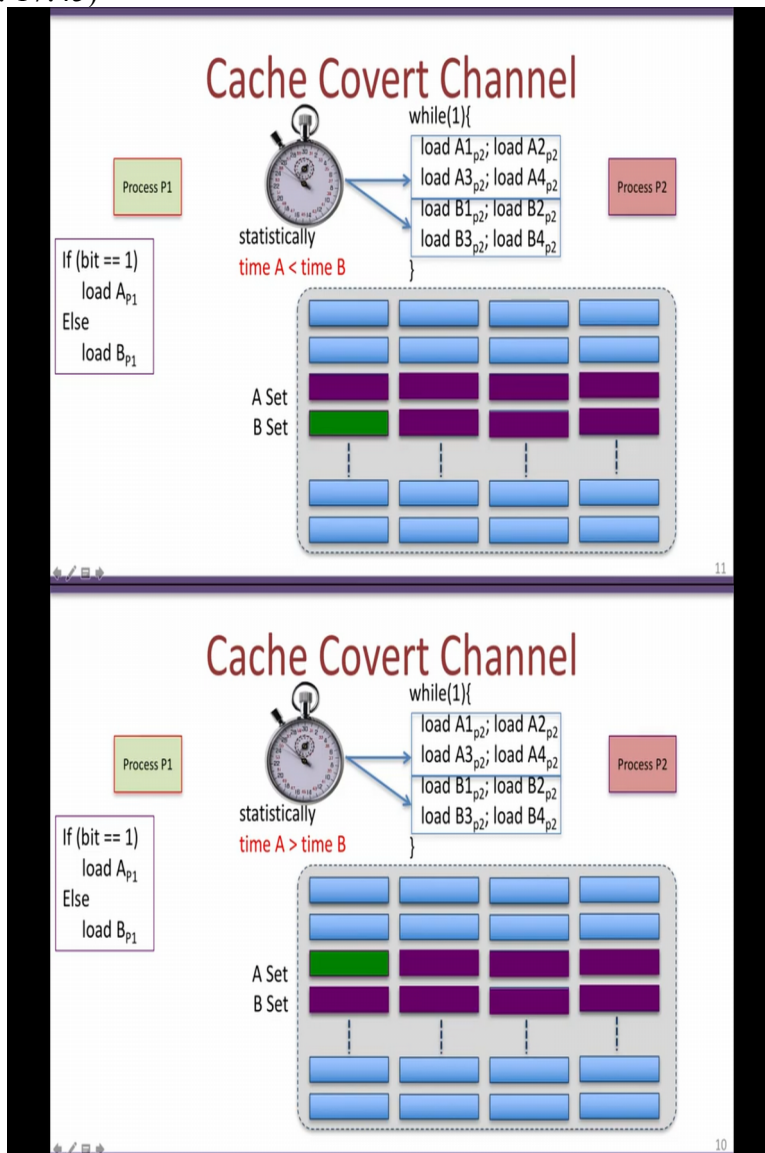


Now consider another process over here process P1 now process P1 has a small loop which looks something like this in this particular loop we have a bit if the bit equal to 1 then you load A P1 else load B P1 so if the bit equal to 1 then load the data corresponding to A in the process P1 address space gets loaded similarly over here the data corresponding to this address B in the P1 address space is loaded note that process P2 and process P1 are totally independent of each other so relating this to the Bell la Padula model that we have studied process P1 may be a top secret process while process P2 may be an unclassified process.

Now you note that we can transmit one bit from process P1 to process P2 using this particular mechanism and the way we go about it is as follows let us say that a process P wants to transmit a bit equal to 0 so what he would do is that he would set the bit to be 0 in which case there would be a load to this particular location which gets executed now these particular addresses in the process P1 address space is selected in such a way that they fall in the A set and the B set respect thus for example if process P1 wants to transmit say a value of 1 to process P2 it would set the bit to be equal to 1 and thus this particular load instruction gets executed that is the load A P1 now what would happen is that the main memory gets accessed there would be a cache miss and one cache line gets loaded from the main memory into this A set so the process P2 data gets evicted and process P1 data would then be filled in that corresponding cache line.

Now if we look at the execution time of these 2 sets of loads what we would see is the time taken for executing these A loads would be greater than the time taken for these B loads statistically again the reason be achieved this is due to the fact that we now have process P1 data present away here as a result when these load A1 A2 A3 and A4 get executed there would be some additional cache misses so that additional cache misses would result in increased execution time for this part of the code on the other hand when the loads to B1 B2 B3 or B4 are executed we similarly we as you shall get cache hits and therefore the time taken would be considerably lesser thus to transmit a value of 1 process P1 which is which for example is a top secret process would set the bit value to be equal to 1 which would then evict one particular cache line from the A set and as a result would influence the execution time of process P2 and therefore execution time for this part of the process P2 would be higher compared to the compared to the other part.

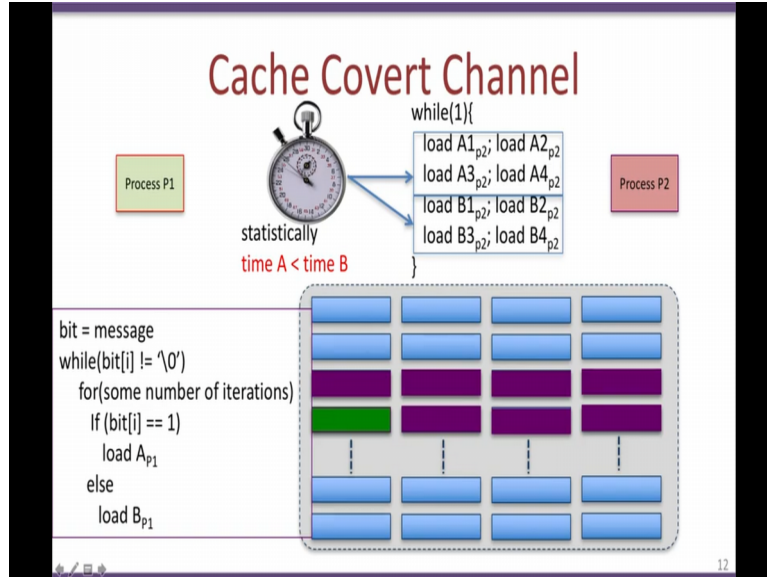
(Refer Slide Time: 17:45)



In a similar way if process P1 wants to transmit a 0 or two process P2 it would set the value of bit to be equal to 0 thus the load P1 gets executed now B P1 as we've discussed would get mapped to the B set and therefore it would evict one of the process P2 data present over here thus in this particular case when a P2 continues its execution in this infinitely while loop the time taken for the loads of B1 B2 B3 and b4 would be higher compared to the low time taken to load A1 A2 A3 and A4 and this is due to the additional cache miss that is present in the set B in this way process P1 can transmit in terms of bits to process P2 by just deciding which address to load from either to load from the address A or to load from the address B and if we just extend this

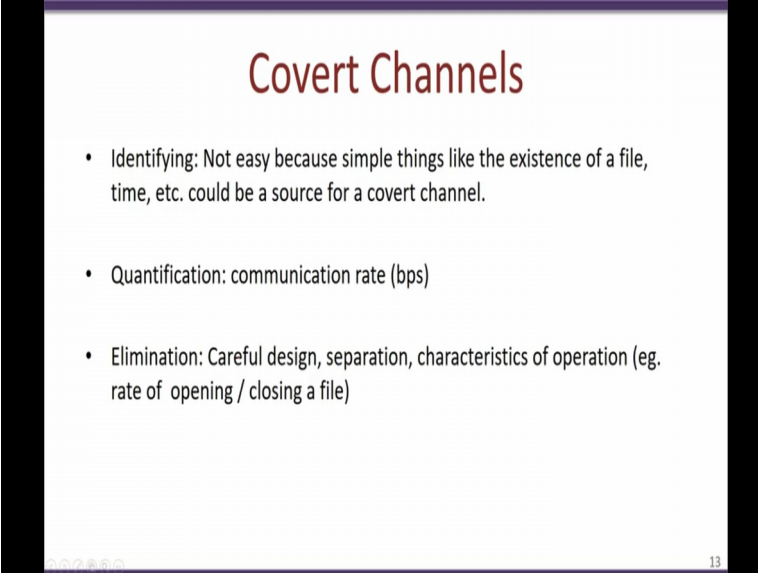
particular idea we can see how a process P1 could send a large message to process P2 and the code for the process P1 would do something like this.

(Refer Slide Time: 19:01)



So for each let us say for example process P1 wants to send a message so what it would do is that it would have an in have a while loop and for each bit in that message it would either load from the A location or the B location so thus we see that it is possible even though there is huge amounts of protection or between process 1 and process 2 both by the hardware as well as the operating system due to the shared cache memory that is present in the processor it would be possible for one process to transfer information to another process however what is actually required over here is that process P1 and process P2 have an agreement about the format in which the bits are transmitted crosses P1 and P2 for example should agree upon the sets which are used for the communication and also they would have to agree upon a particular protocol for communicating between the two processes.

(Refer Slide Time: 20:09)



Covert Channels

- Identifying: Not easy because simple things like the existence of a file, time, etc. could be a source for a covert channel.
- Quantification: communication rate (bps)
- Elimination: Careful design, separation, characteristics of operation (eg. rate of opening / closing a file)

13

Now cache covert channels are a big problem it is very difficult to actually identify such problems cover channels in general are a big problem the cache covert channels are just one example in addition to this there could be several other different types of cover channels and thus identifying all the cover channels present in the system is quite a difficult task.

Another important aspect when we talk about coming about covert channels is the communication rate or to quantify that covert channel here what is important for us is to measure the rate at which data can be communicated through that particular covert channel.

(Refer Slide Time: 19:01)

Cache Covert Channel

The diagram illustrates a cache covert channel between Process P1 and Process P2. Process P1 is on the left, and Process P2 is on the right. A stopwatch icon is positioned between them, with the text "statistically time A < time B" below it. Process P2 has a code block:

```
while(1){
  load A1_p2; load A2_p2
  load A3_p2; load A4_p2
  load B1_p2; load B2_p2
  load B3_p2; load B4_p2
}
```

Process P1 has a code block:

```
bit = message
while(bit[i] != '0')
for(some number of iterations)
  if (bit[i] == 1)
    load A_p1
  else
    load B_p1
```

Below the code blocks is a cache diagram with 4 columns and 4 rows. The top row is blue. The second row has a green cell in the first column and purple cells in the other three. The third row is purple. The bottom row is blue. Vertical dashed lines connect the cells in each column.

Covert Channels

- Identifying: Not easy because simple things like the existence of a file, time, etc. could be a source for a covert channel.
- Quantification: communication rate (bps)
- Elimination: Careful design, separation, characteristics of operation (eg. rate of opening / closing a file)

For example if we go back to this particular slide communication rate of this particular channel would be the number of bits transmitted from process one to process two per second so this typically is very slow so for example a good number would be say one or two bits per second is actually a very good cache covert channel so in order to eliminate such covert channels in other design one should be able to design the system extremely careful carefully and ensure that there is a proper separation between processes not just at the operating system level but also at the hardware level for example to prevent the cache covert channel one should ensure that at no time process P1 as well as process P2 are actually sharing the same cache memory in the lectures that follow we'll be looking at other forms of cache timing attacks where algorithms such as

cryptographic algorithms can be broken and secret keys from that cryptographic algorithm can be stolen by means of measuring the memory access times, thank you.