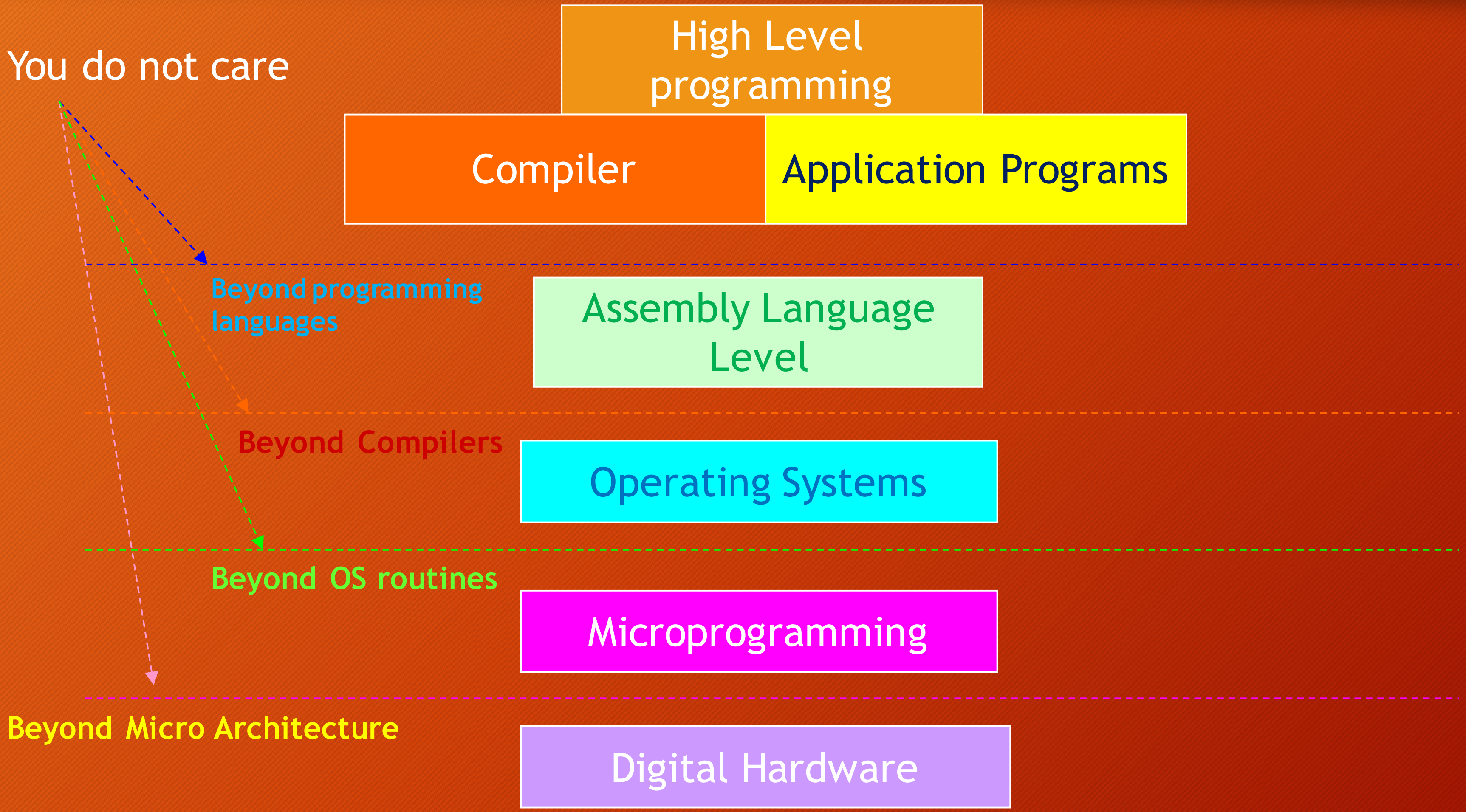
Introduction to Meltdown Attack

Kamakoti V

Computing system is nothing but layers of *virtual machines..*



You do not care

High Level programming

Compiler

Application Programs

Beyond programming languages

Assembly Language Level

Beyond Compilers

Operating Systems

Beyond OS routines

Microprogramming

Beyond Micro Architecture

Digital Hardware

Meltdown attack

- Discovered in August 2017. Published in Jan 2018.
- Vulnerability in all the processors manufactured since 1995.
- Almost every system is affected by Meltdown: Desktops, Laptops, Cloud Servers, as well as Smartphones.
- Almost all manufacturers affected by Meltdown: Intel, AMD, ARM
- High Level Idea: The vulnerability basically melts security boundaries which are normally enforced by the hardware.
- Typical example of "an optimization gone wrong".

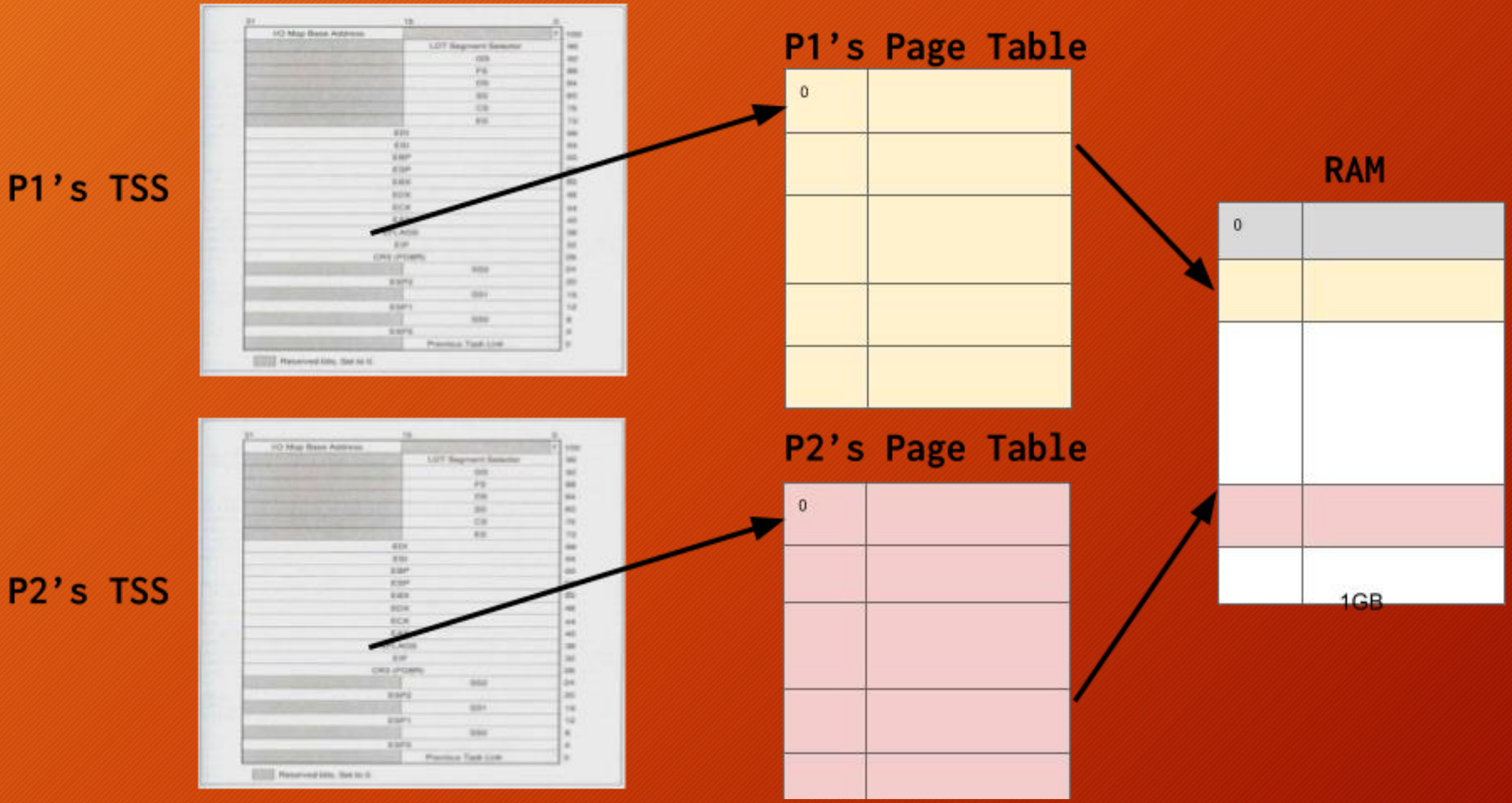
Companies that issued Advisories

- Intel
- ARM
- AMD
- RISC-V
- NVIDIA
- Microsoft
- Amazon
- Google
- Android
- Apple
- Lenovo
- IBM
- Dell
- Mozilla
- Red Hat
- Debian
- Ubuntu
- SUSE
- Fedora
- Qubes
- Fortinet
- NetApp
- LLVM
- CERT
- MITRE
- VMWare
- Citrix
- Xen
- Hewlett Packard Enterprise
- HP Inc.
- Huawei
- Synology
- Cisco
- F5

Meltdown in 2 sentences

- Meltdown breaks the most fundamental isolation between user applications and the operating system.
- This attack allows a program to access the memory, and thus also the secrets, of other programs and the operating system.

Fundamental Isolation: TSS and Paging



Interactions Between Process P_i and OS

- Hardware devices are accessible only by the OS
- Dynamic memory can be allocated only by the OS
- Pi may have a need to read inputs from Keyboard, write to a printer, or allocate dynamic memory
- **System Calls and Interrupts** can be leveraged by Pi to take the assistance of OS to perform these privileged tasks.

Meltdown: A 3 step attack

1. **(OS) Address Space basics**: How address is allocated to a process for its execution? How one process is protected from another? Is it fool-proof?
2. **(HW) Out-of-Order Execution**: Instructions are executed by the processor in an order different from the actual sequence. Why? How attackers can take advantage of this?
3. **(Micro-Architecture) Cache Memory attacks**: A timing side-channel attack on the cache which allows the attacker to read a memory address that he is not supposed to.

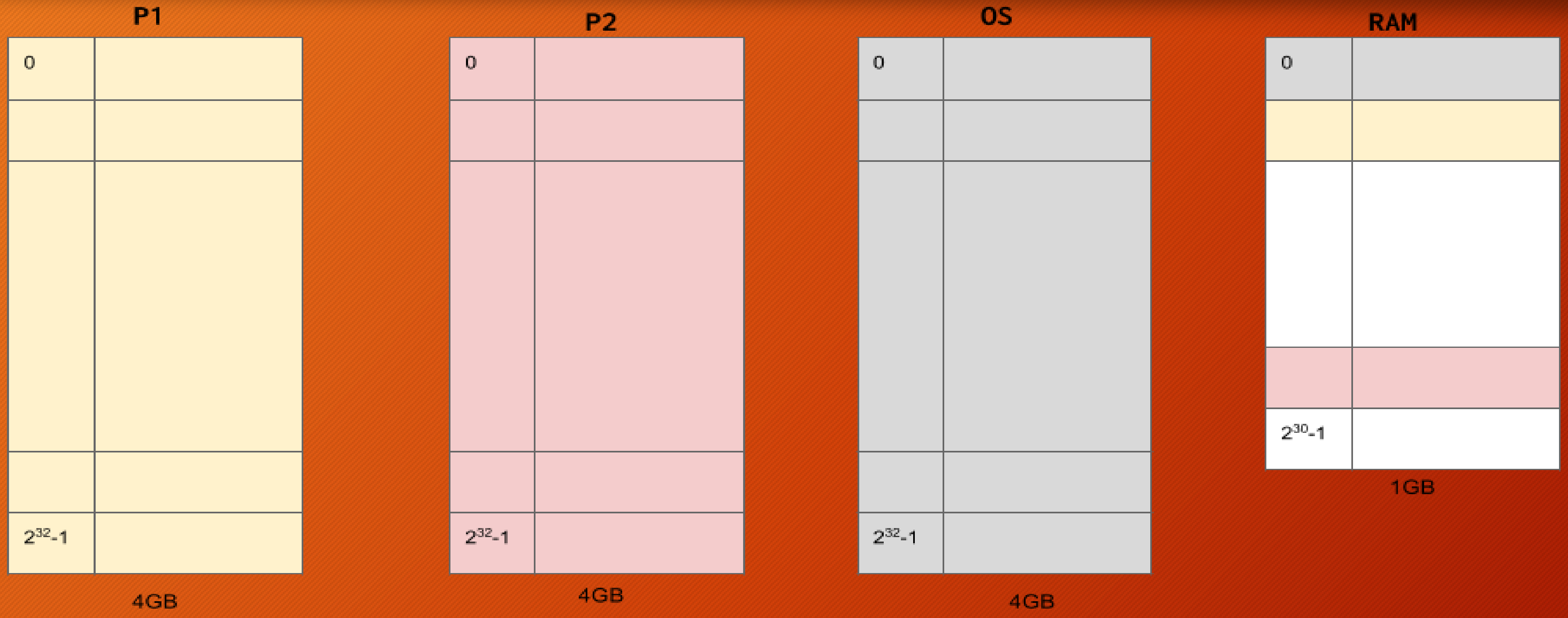
Address Space Basics (Ref: IS-2 and IS-3)

Kamakoti V

Enabling Interactions Between Process P_i and OS

- Hardware devices are accessible only by the OS
- Dynamic memory can be allocated only by the OS
- Pi may have a need to read inputs from Keyboard, write to a printer, or allocate dynamic memory
- **System Calls and Interrupts** can be leveraged by Pi to take the assistance of OS to perform these privileged tasks.

P_i and OS: An Example

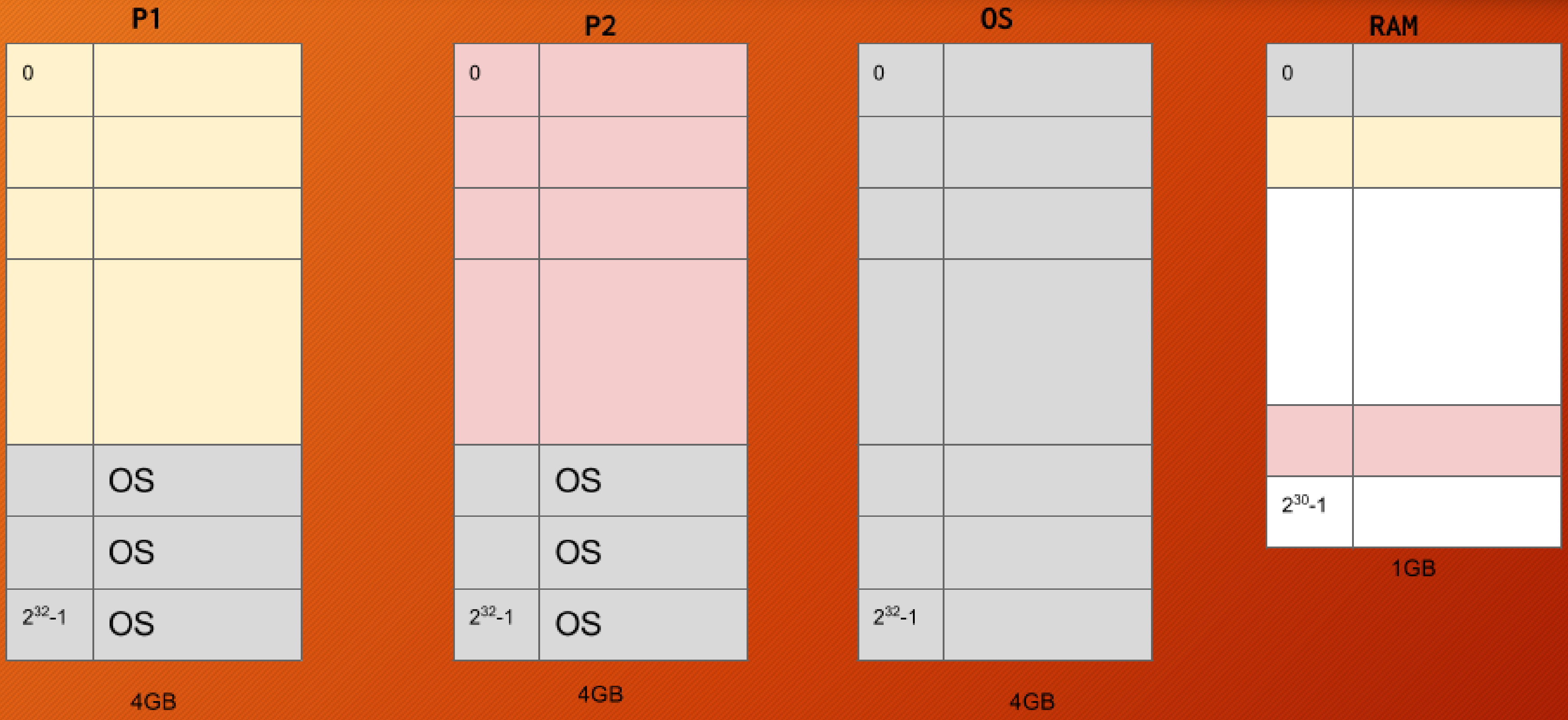


1. Key press Interrupt
2. OS's ISR gets Key Data
3. The Data must be copied from OS's address space to P1's address space
4. Costly Operation!!

What happens when P1 wants to read a keyboard input?

Only OS can access keyboard – How to pass the input to P1

VA Space in Today's OS (Optimization)



1. Key press Interrupt
2. OS's ISR gets Key Data into the pages shared between P1 and itself.
3. Copy complete!!

A safe and sound way to share data between OS and P1

Can P1 access the OS space Today? NO!

- Hardware allows two modes: user mode and supervisor mode.
- Hardware bit that indicates the mode is called "supervisor bit".
- Supervisor mode has extra permissions compared to user mode.

Steps Involved in accessing OS pages

- P1 tries to access the OS pages in its Virtual Address space.
- Hardware checks if the page can be accessed in user mode (for P1).
- OS pages can be accessed only in supervisor mode (as per page access permissions).
- Exception is raised by the hardware to prevent access.

Address space isolation Guarantees

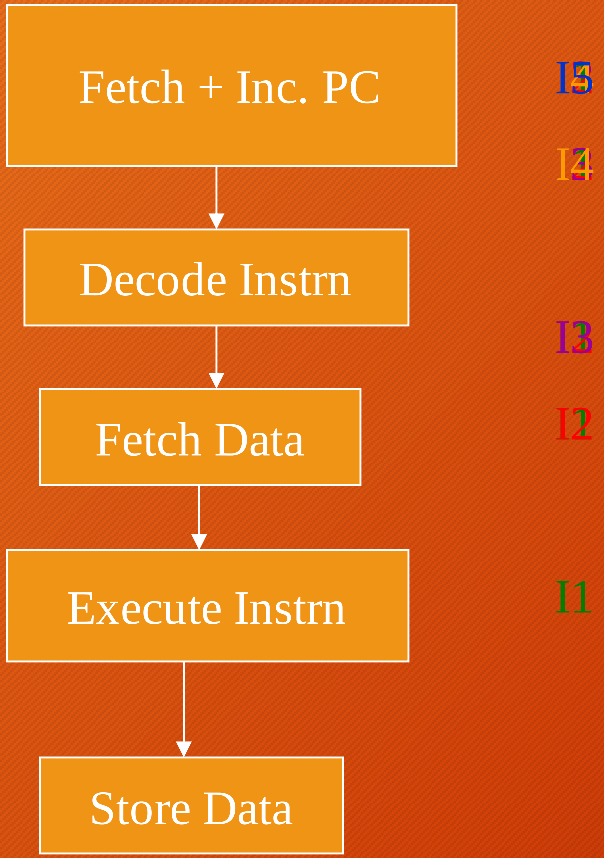
- Physical address space: Every user process is given access to a part of the RAM for its execution.
- A user process cannot access the physical address allocated to another process (including Kernel).
- Kernel has access to physical address space of all processes i.e. the entire RAM.

Takeaway: Somehow enable Pi to read the OS-mapped pages from Pi's own page table

Out-of-order Execution (Ref: IS-2)

Kamakoti V

ILP – In-order Pipelining



First Instruction completes at end of 5th unit

Second instruction at end of 6th unit

10000th instruction at end of 10004 units

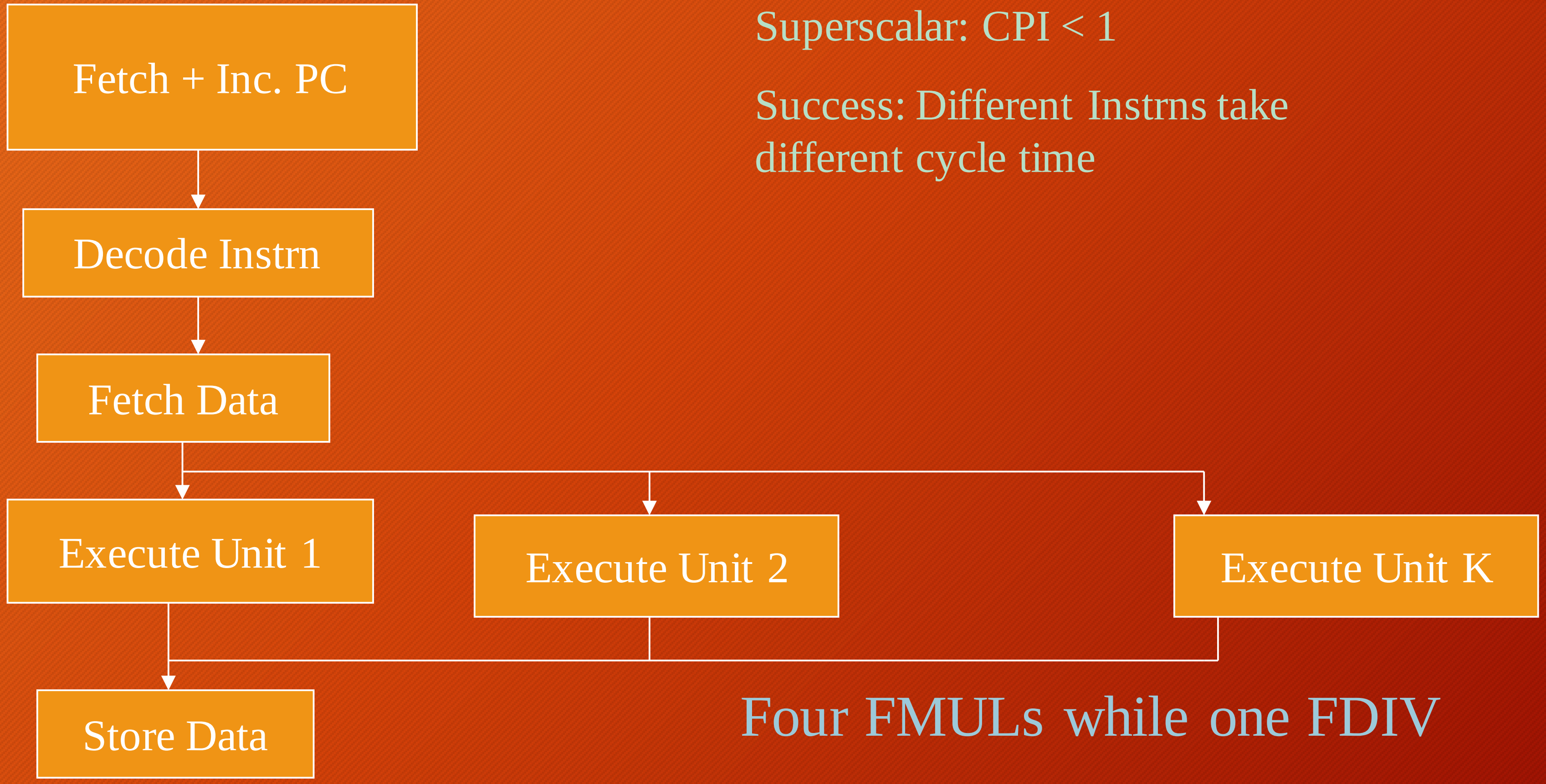
With Pipelining

10000 instructions

No pipelining takes

50000 Units

ILP – Pipelining Advanced



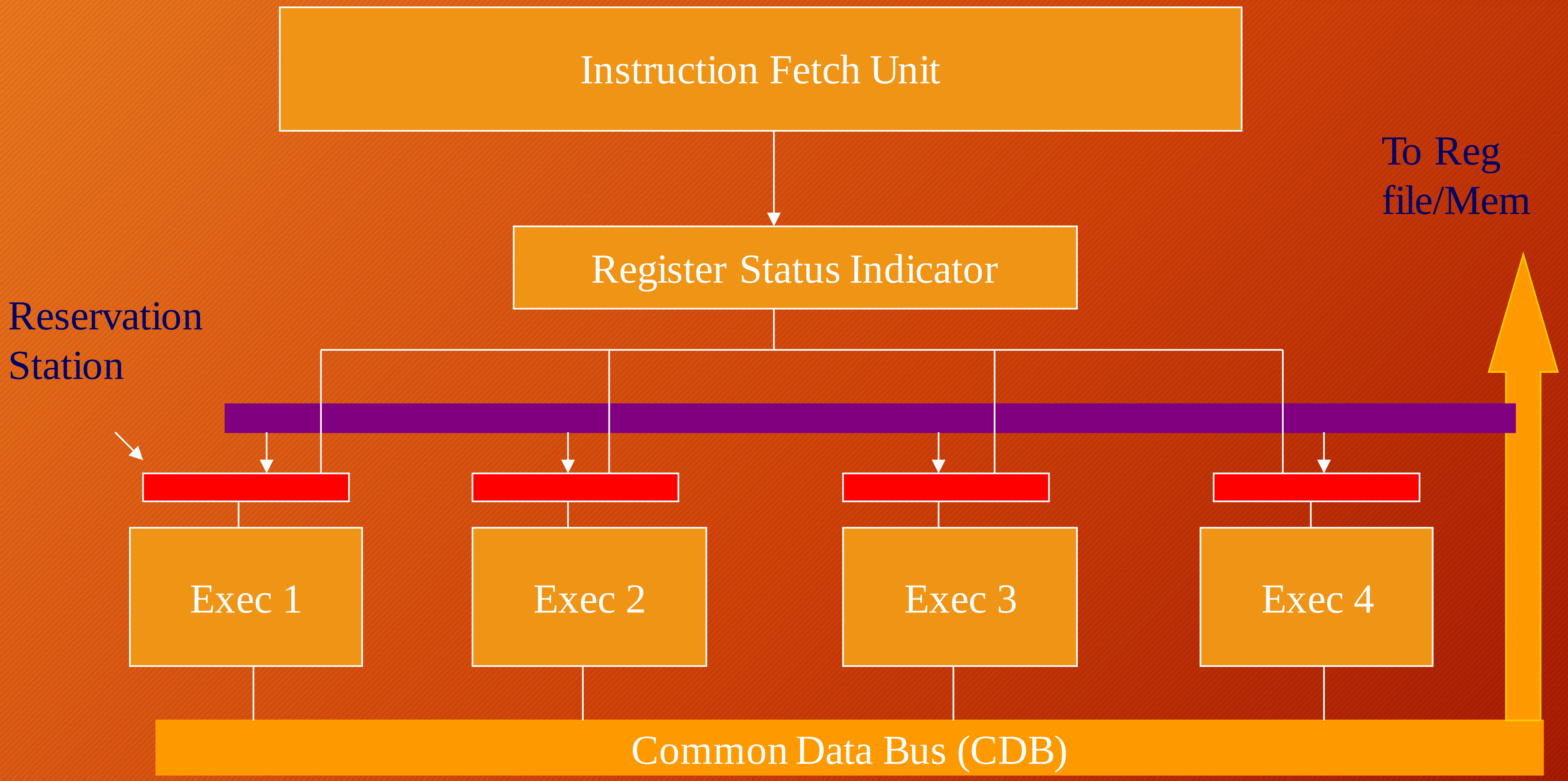
Superscalar: CPI < 1

Success: Different Instrns take different cycle time

Four FMULs while one FDIV

Implies – Out-of-Order Execution

Dynamic Scheduling - Tomasulo



Instructions are fetched one by one and decoded to find the type of operation and the source of operands

Register Status Indicator indicates whether the latest value of the register is in the reg file or currently being computed by some execution unit and if the latter it states the execution unit number

If all operands available then operation proceeds in the allotted execution unit, else, it waits in the reservation station of the allotted execution unit pinging the CDB

Every Execution unit writes the result along with the unit number on to the CDB which is forwarded to all reservation stations, Reg-file and Memory

What happens in this program P1?

Instruction	Process P1's code
i	Mov ("os loc"), ebx
i+1	Transient Instruction - 1
i+2	Transient Instruction - 2
i+3	Transient Instruction - 3
i+4	

1. Instruction i gets executed
2. Instructions i+1, i+2, i+3 are also being executed.
3. i encounters exception (P1 to OS access) and the changes are rolled back.
4. Changes caused by i+1, i+2, i+3 are also rolled back.

Micro-Architectural State: Registers

There is a **small window of time** from the time i writes to the register ebx and the instant at which the exception is triggered.

- The register modified by i can be accessed by i+1, i+2, i+3 in this small window of time.

The Kernel data can now be read by i+1, i+2, … !!

- However, once the exception is raised, all instructions after i are also rolled back .

Recovering from Exception

Kamakoti V

Process Forking

1. Parent Process (P1-p) Starts: Used to spawn a child which launches the attack.
2. Child Process (P1-c) Performs the attack: Accesses the OS region, leaks data to ebx, but encounters an exception.
3. Parent Process (P1-p) Takes Over: The Parent can kill the child on an exception and continue executing.

How to Transfer leaked data from the Child Process to Parent Process?

Covert Channel

Kamakoti V

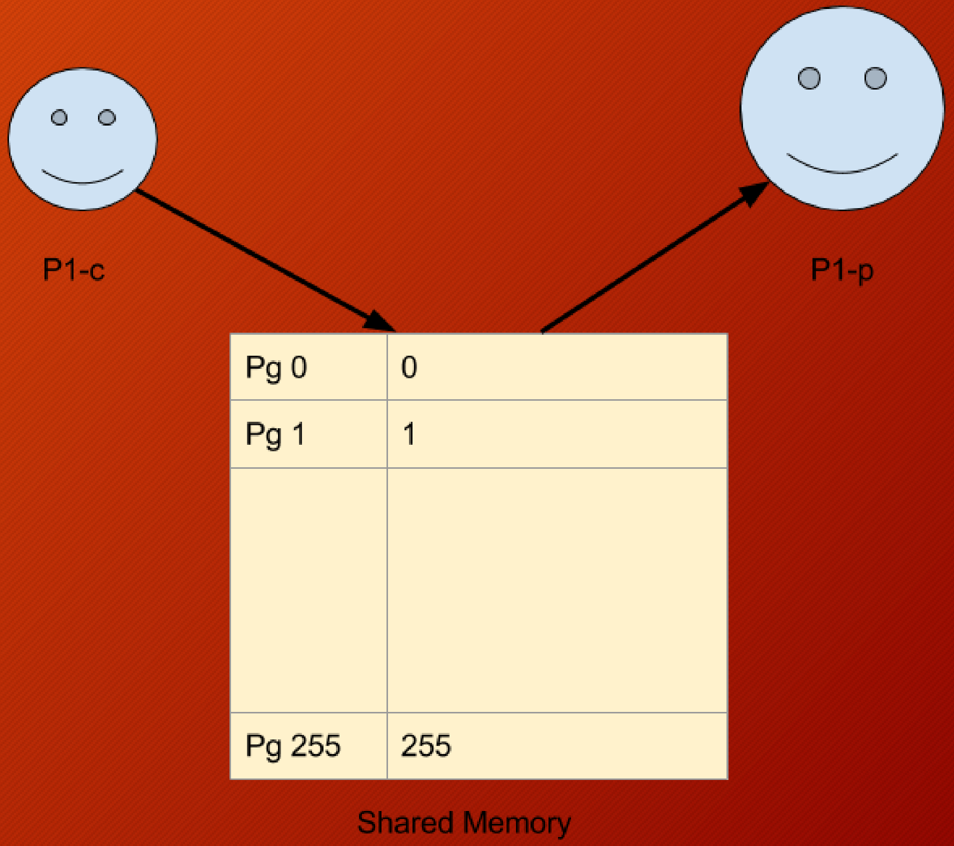
One bit Covert Channel

- P1-c cannot "write" a 1-bit secret data in register ebx to P1-p, since exception will clear it.
- P1-p and P1-c agree that if the secret is "1", P1-c must write to location 0x1000, else if it is "0" then write to location 0x2000.
- Effectively P1-p and P1-p have shared the secret through the covert channel.

8-bit Covert Channel

- P1-c writes to Page-0 (4KB size) if the data in ebx is 0.
- P1-c writes to Page-84 if the data in ebx is 84.
- P1-c can leak 8-bits of data at a time to its parent process P1-p

However, on an exception, all such writes from P1-c to P1-p are discarded !!!



Pg 0	0
Pg 1	1
Pg 255	255

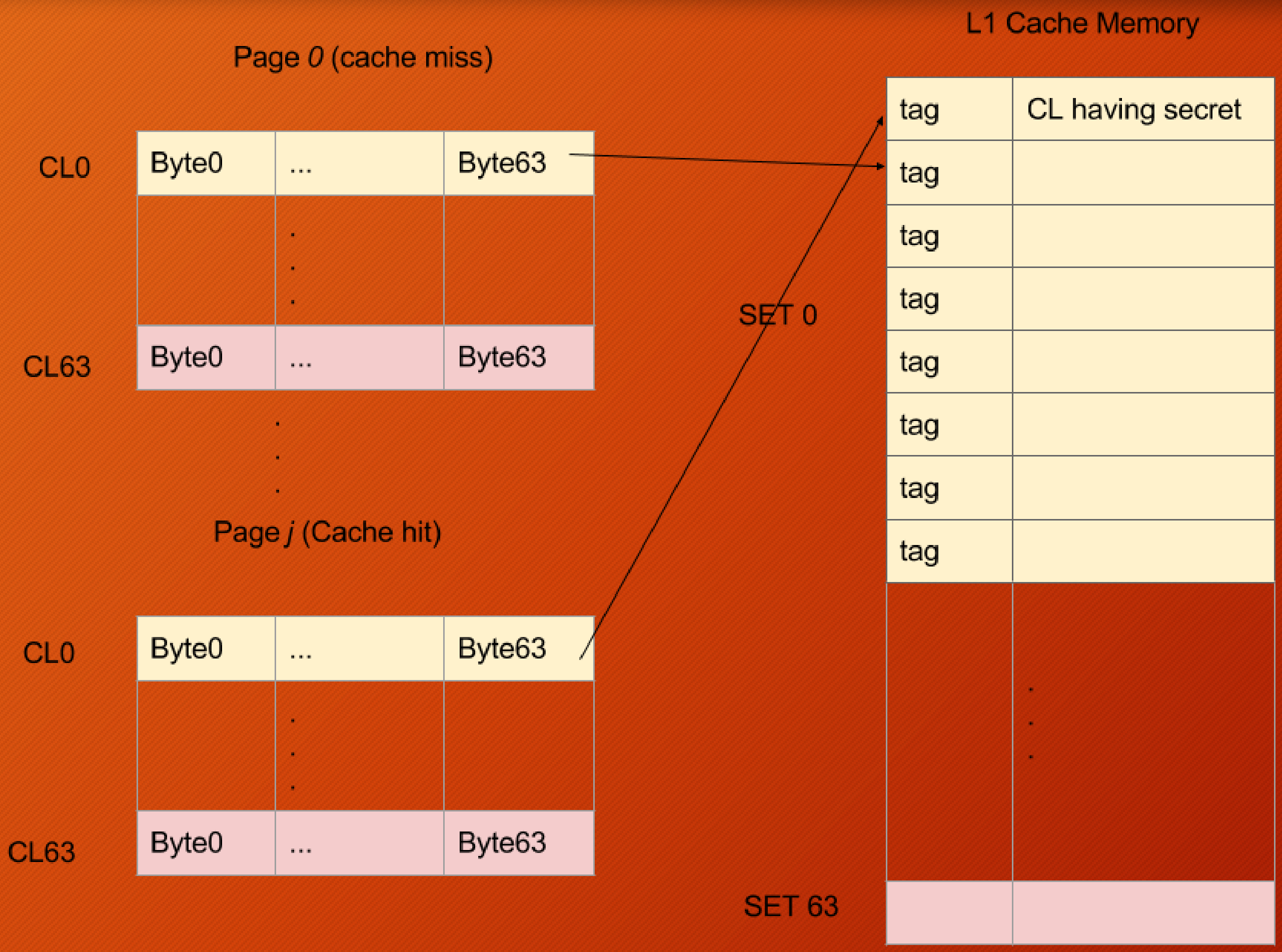
Cache Timing Attacks: Flush+Reload

Kamakoti V

The idea

- Cache hits take less time than cache misses.
- The P1-p can know if the data is in the cache or not, based on the time taken to read a memory address.
- P1-p and P1-c share a memory region. But, P1-p does not know which address P1-c is accessing.
- Flush+Reload Attack:
  - Ensure that the cache does not have any stale data. (Flush part)
  - P1-c writes to a cache-line in the cache while executing.
  - P1-p accesses all cache-lines again.
  - Only one cache-line will result in a cache hit.

The Attack



L1 Cache Memory

Page *0* (cache miss)

CL0	Byte0	...	Byte63
		. . .	
CL63	Byte0	...	Byte63

Page *j* (Cache hit)

CL0	Byte0	...	Byte63
		. . .	
CL63	Byte0	...	Byte63

SET 0

tag	CL having secret
tag	
tag	
tag	
tag	
tag	
tag	
tag	
	. . .

SET 63

Thank You