

# Meltdown for Dummies

---

The road to hell is full of good intentions

The logo for EPSRC (Engineering and Physical Sciences Research Council) features the letters 'EPSRC' in a bold, purple, sans-serif font. The text is framed by two horizontal teal lines, one above and one below.The logo for Cray, 'THE SUPERCOMPUTER COMPANY', features the word 'CRAY' in a blue, stylized, sans-serif font. Below it, the tagline 'THE SUPERCOMPUTER COMPANY' is written in a smaller, blue, sans-serif font.The logo for NERC (Natural Environment Research Council) consists of the letters 'NERC' in white, bold, sans-serif font on a dark green rectangular background. To the right, on a light green background, are the words 'SCIENCE OF THE ENVIRONMENT' in white, sans-serif font.The logo for epcc (Energy Policy Centre) features the lowercase letters 'epcc' in a blue, sans-serif font. The text is flanked by two vertical red lines on either side.The logo for Archer features a red and white bullseye target icon on the left. To its right, the word 'archer' is written in a white, lowercase, sans-serif font on a black rectangular background.

# Reusing this material



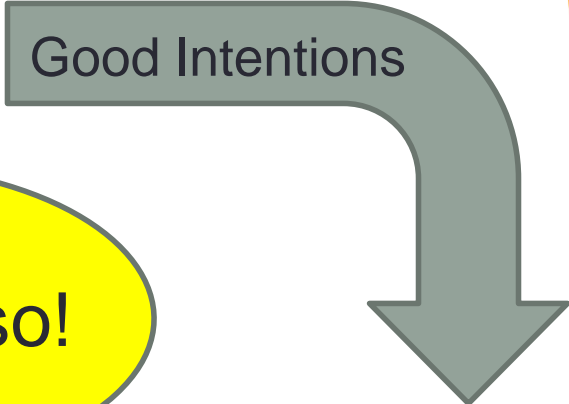
This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

[http://creativecommons.org/licenses/by-nc-sa/4.0/deed.en\\_US](http://creativecommons.org/licenses/by-nc-sa/4.0/deed.en_US)

This means you are free to copy and redistribute the material and adapt and build on the material under the following terms: You must give appropriate credit, provide a link to the license and indicate if changes were made. If you adapt or build on the material you must distribute your work under the same license as the original.

Note that this presentation contains images owned by others. Please seek their permission before reusing these images.

# Saint Bernard of Clairvaux



# Outline

- Introduction
- Analogy
- Anti-meltdown patches
- Performance implications
- Summary

# What is Meltdown?



Meltdown exploits a **race condition**, inherent in the **design of many modern CPUs**. This occurs between **memory access** and **privilege checking** during instruction processing. Additionally, combined with a **cache side-channel attack**, this vulnerability allows a process to bypass the normal privilege checks that isolate the exploit process from accessing data belonging to the operating system and other running processes. The vulnerability allows an unauthorized process to read data from **any address that is mapped to the current process's memory space**.

# Analogy

- Computers are analogies!
  - Surely there must be an everyday situation that exhibits same behaviour

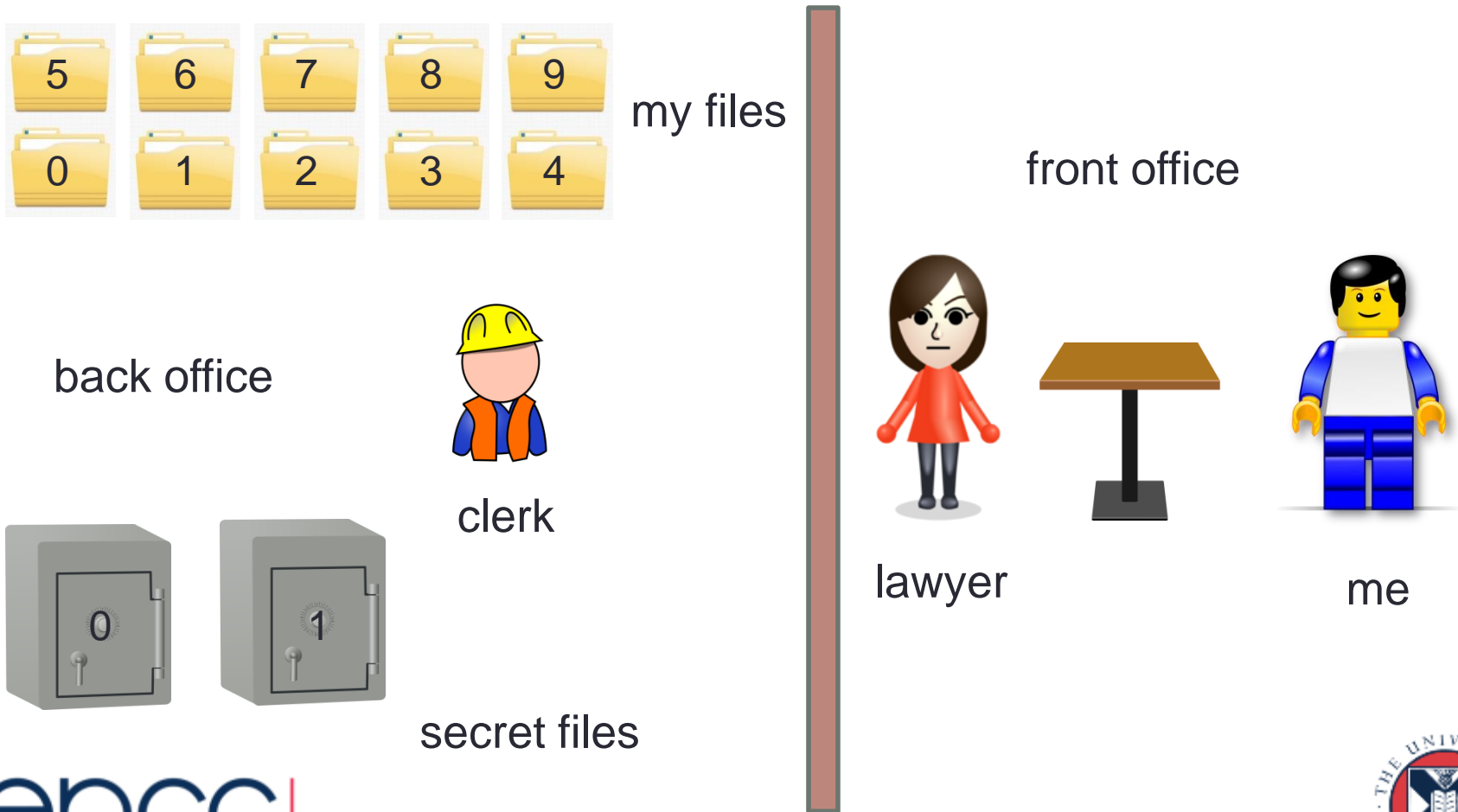


Computer	Analogy
Machine	Lawyer's Office
Operating System	Lawyer
Memory System	Legal Clerk
RAM	Filing Cabinets
Accessible Data	My Legal Documents
Secret Data	Other People's Docs



# Lawyer's Office

- How can I trick them to give me access to sensitive info?



# Computer equivalents



user  
memory

software

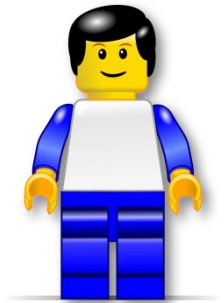
hardware



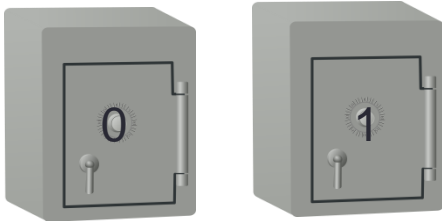
memory  
system



operating  
system



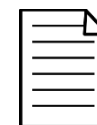
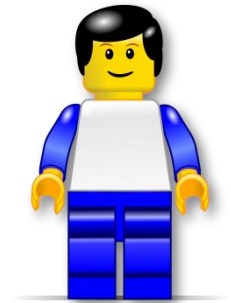
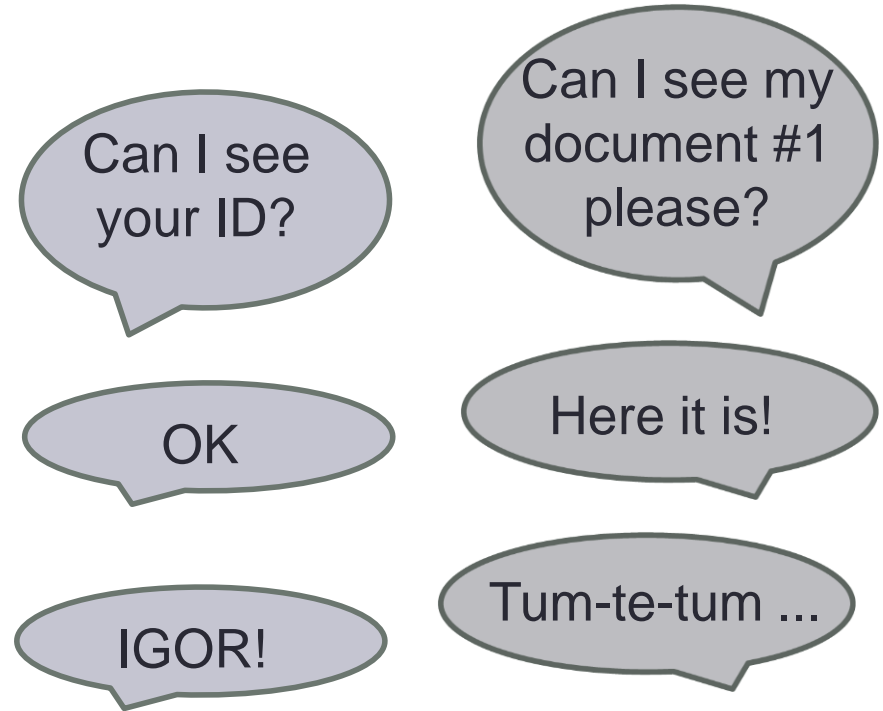
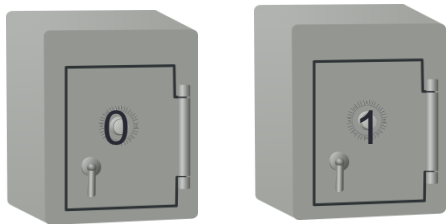
user  
program



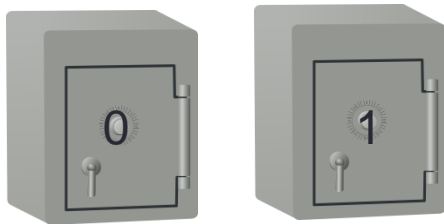
OS kernel  
memory



# Memory request



# Illegal request



Can I see your ID?

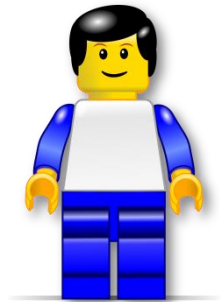
NO!

GO AWAY!

Can I see the contents of safe #0 please

Here it is!

Sorry ...



# G1: office should operate quickly

- It would be really slow to go to a different office every time the clerk wanted to access secret documents
  - store them all in the same office
  - just deny access to the safes if people aren't authorised
- Computer
  - all RAM is mapped into virtual address space of all user programs
  - operating system checks privileges of the accessing process
  - only kernel processes are allowed access outside of user memory
- ... will assume all accesses are valid for next sections

# Unintended Consequence: UC1

- I can see the secret documents
  - so I can ask for them
- If they were in a different office
  - I wouldn't even be able to even ask for them

# Memory request

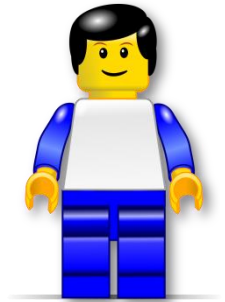
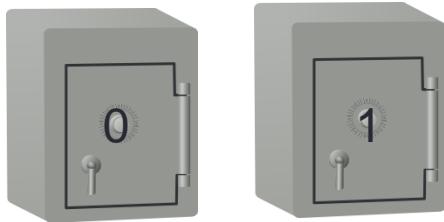


IGOR!

you're welcome

Can I see my document #1 please?

.... thanks!



# Memory request



IGOR!

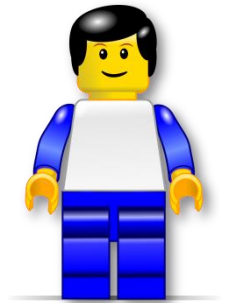
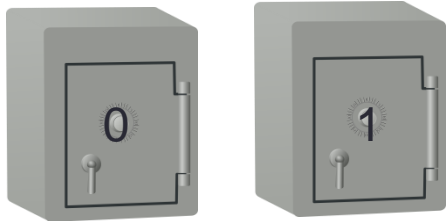
you're welcome

IGOR!

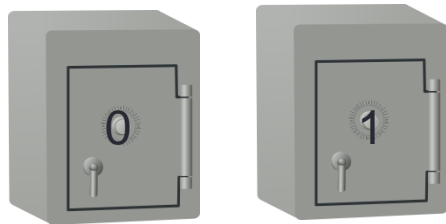
Can I see my document #1 please?

.... thanks!

Again please?



# Memory request

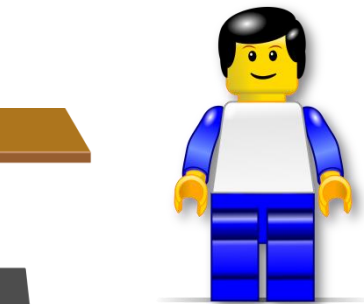


IGOR!

you're welcome

IGOR!

IGOR!



Can I see my document #1 please?

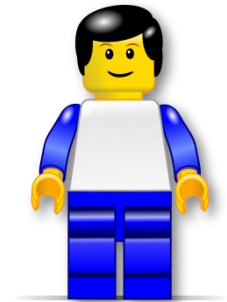
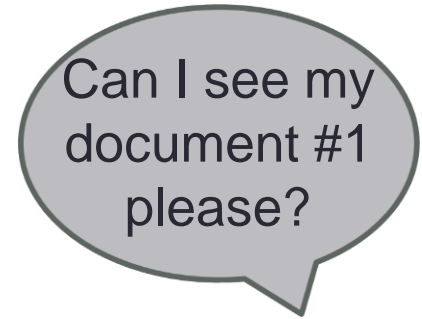
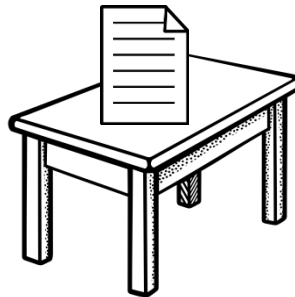
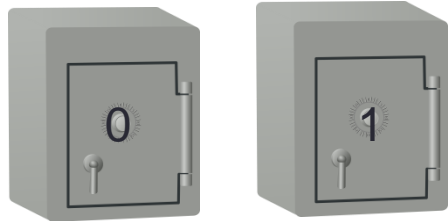
.... thanks!

Again please?

And again?

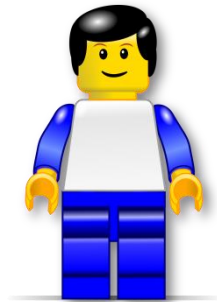
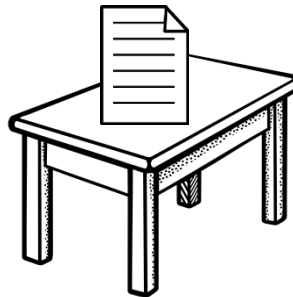
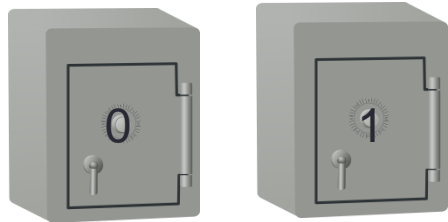


# Memory request





# Memory request



IGOR!

IGOR!

Can I see my document #1 please?

Again please?

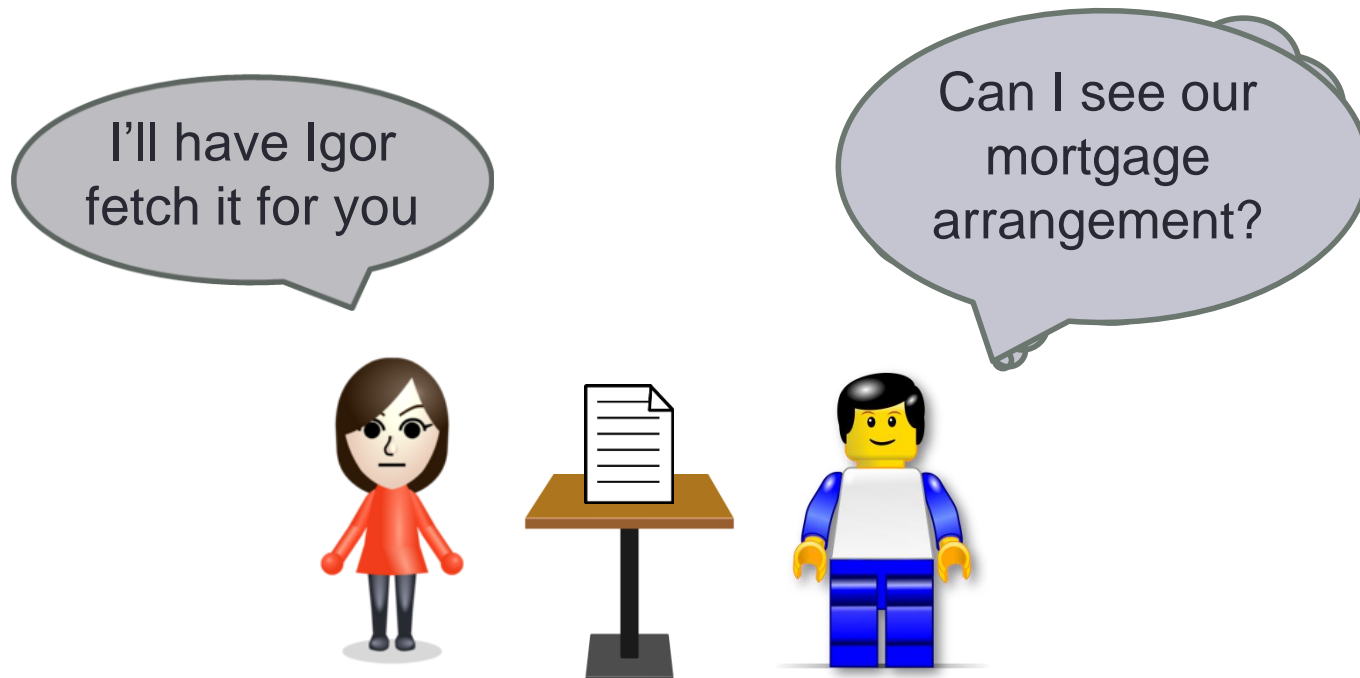
That was quick!

# GI2 : memory access should be fast

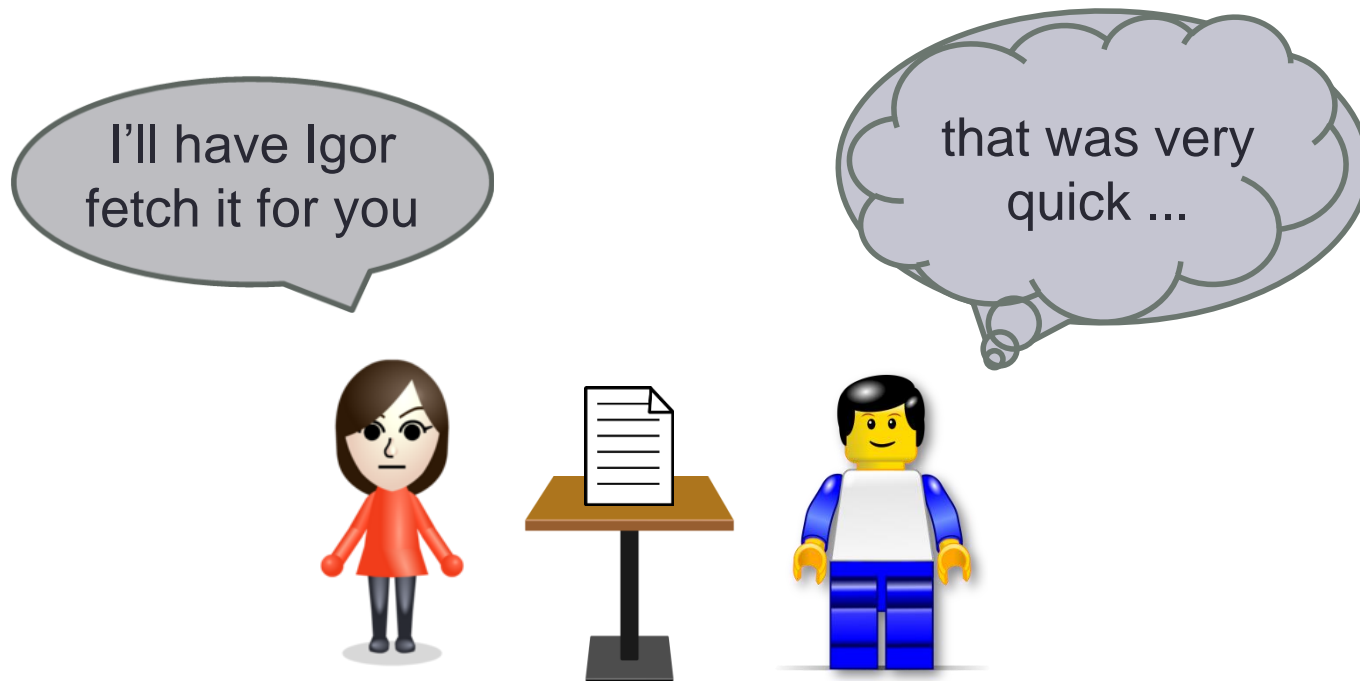
- Why keep going back to document store all the time
  - keep recent documents close to hand ...
  - ... but still secure in the back office
- Computer
  - memory system keeps copies of recent data in fast cache memory
  - reads and writes apply to the cached copy
  - done automatically in hardware – user just accesses memory

# Unintended Consequence

- On bus home, I saw my partner's car at lawyer's office
  - they say they were checking the mortgage arrangements



# Unintended Consequence



# UC2: access to privileged information

- Accessing documents that I am **allowed** to read can give me information that I am **not allowed** to have
  - through differences in timing
- Computer
  - time taken to load data from memory tells you if it was cached
  - you can deduce whether or not it has recently been accessed
- Now let's return to authorisation step

# Authorisation



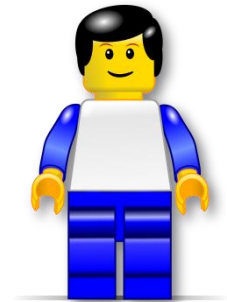
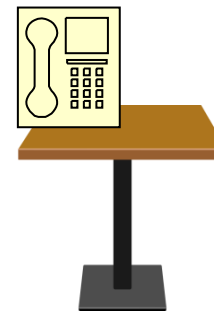
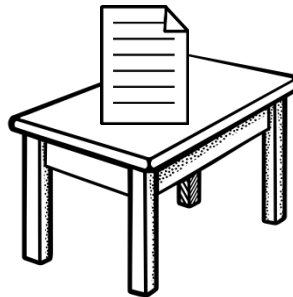
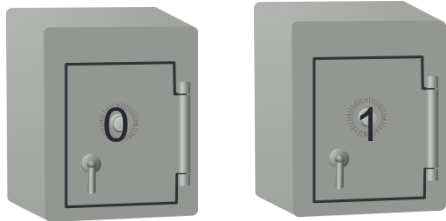
Can I see your ID?

Can I see my document #1 please?

Let me check

Here it is!

IGOR!



# GI3: authorisation should be fast

- Why wait to tell Igor to go for the document?
  - ask for the document **at the same time** as checking authorisation
  - don't deliver the document until authorisation confirmed
- Computer
  - execute instructions **out of order**
    - don't wait for instruction #1 to complete before issuing instruction #2
  - absolutely essential technique to keep modern CPUs busy
  - only deliver data to user program once all checks are passed
  - otherwise **roll back** to restore previous state
    - optimise for the usual case where everything is OK
    - don't care if the unusual case takes more time

# Out-of-order (legal)



Can I see your ID?

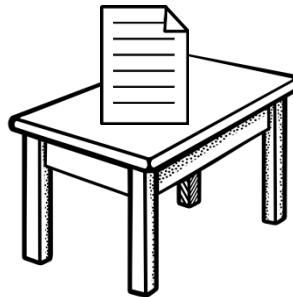
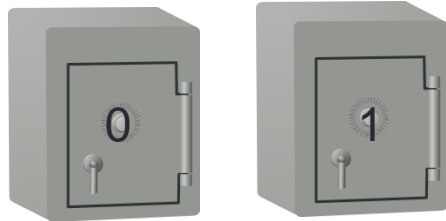
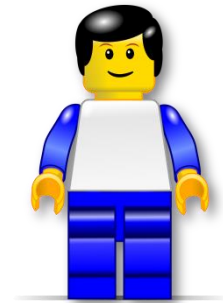
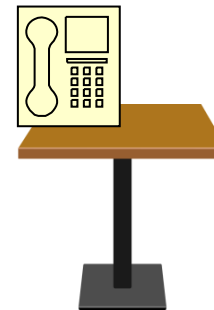
Can I see my document #1 please?

IGOR!

Here it is!

Let me check

It's OK Igor





# Out-of-order (illegal)



Can I see your ID?

Can I see safe #0 please?

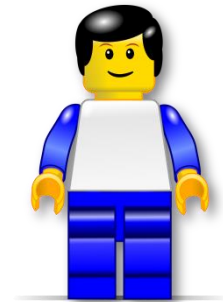
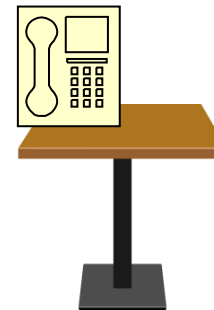
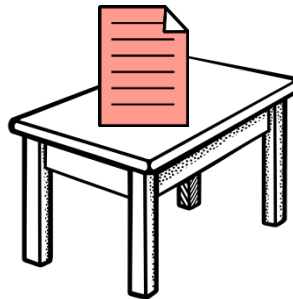
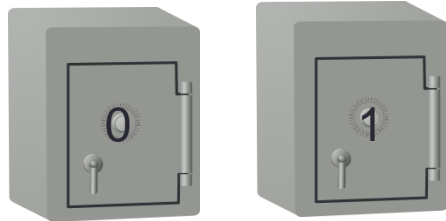
IGOR!

Here it is!

Let me check

Sorry!

Rollback!



# UC3: digital fingerprints

- The rollback is **incomplete**
  - secret data is in the cache
- But I still can't read it
  - so it's all OK?
- Do I detect seeds of doubt?

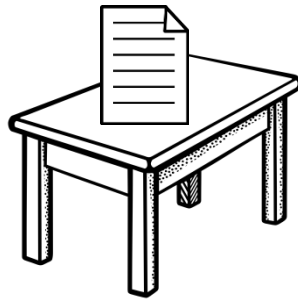
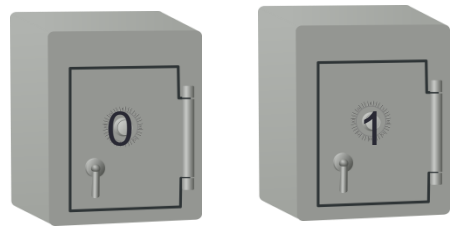
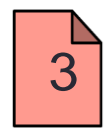
# GI4: Indirection

- Many programs have constructs akin to

```
loop: i = 1, N
      index = lookuptable(i)
      x(i) = x(i) + y(index)
```

- CPUs have machine-code instructions to do this
  - not “load the data from location i”
  - but “load the data from location stored in location i”
- This is the final piece of the puzzle

# Indirection



Can I see your ID?

Can I see my document numbered the same as contents of safe #0 please?

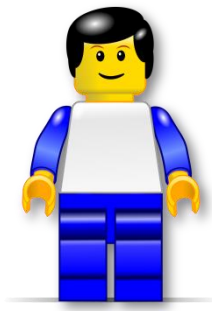
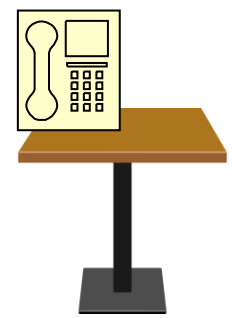
IGOR!

Let me check

Here it is!

Rollback!

Sorry!



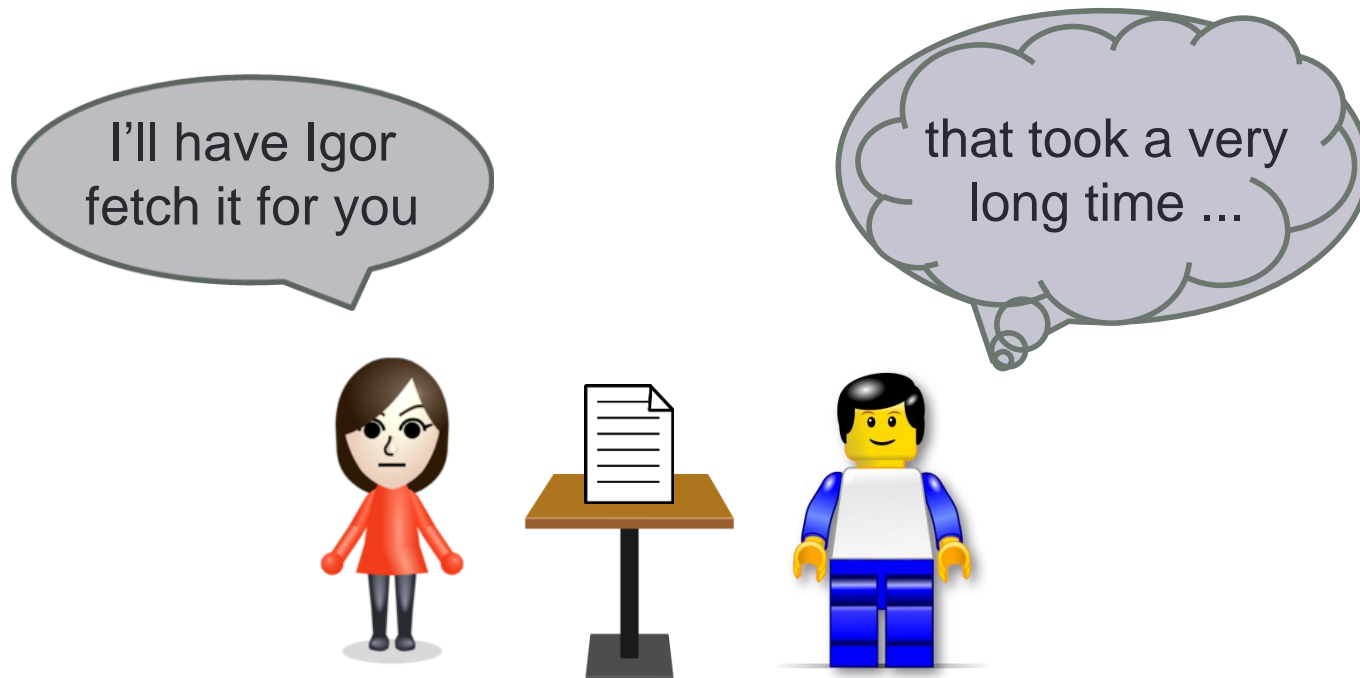
# So what?

- What did we **want** to achieve?
  - a way of being allowed to read data we should not have access to
- What **have** we achieved?
  - a way of not being allowed to read data we do have access to!
- But there are side-effects
  - I was not able to read the value “3” from the safe
  - but my document #3 is in the cache!

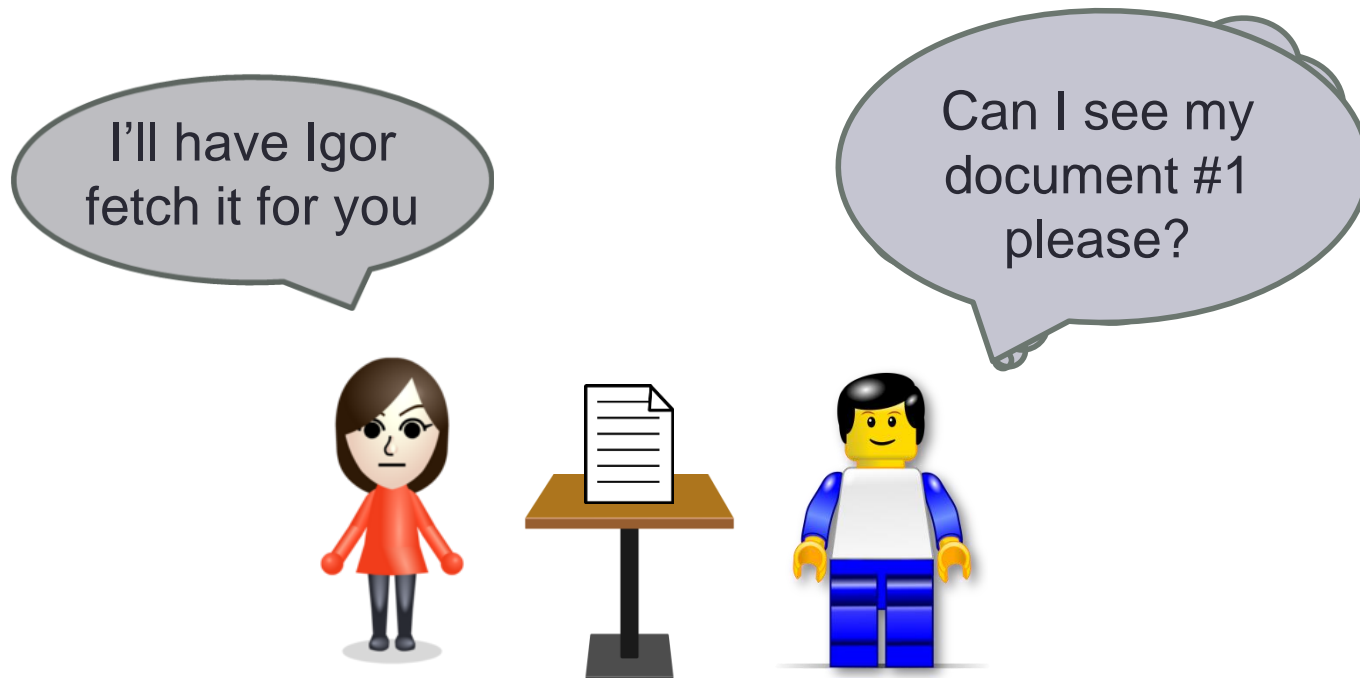
# UC4: side effects of indirection

- The rollback is **incomplete**
  - there is a fingerprint in the cache
  - produced as a side-effect of accessing a secret document
- I wasn't able to read the secret document
- But I can now read my own documents
  - and see how quickly they come back

# Inferring the secret data

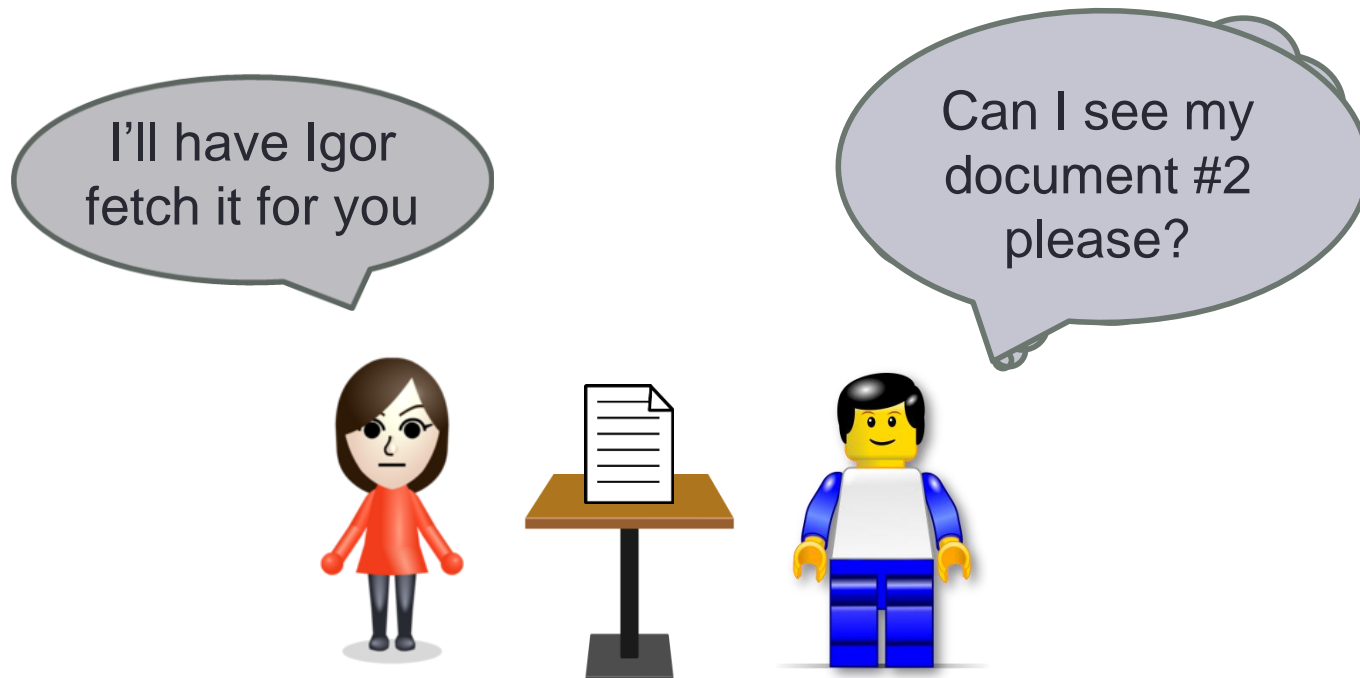


# Inferring the secret data



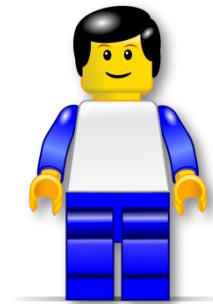


# Inferring the secret data



# Inferring the secret data

I'll have Igor  
fetch it for you



BINGO!

# Meltdown in a nutshell

- Try to read an array element from one of your own arrays
  - with an index taken from the value of secret data
- The read will fail but the array element will be cached
- Scan through your array element by element
  - measure the time taken to read each element to see if it's cached
  - if reading element “i” is fast, then the secret value was “i”
- Simple!

# Making it fast

- Analogy lets me read a single digit 0-9 from any safe
  - since I have 10 document holders
- How best to read 0-99 ?
  - a) two reads, one for each digit?
  - b) one read but with 100 document holders?
- Time to access my documents dominates the read
  - a) up to 20 document accesses (average of 10 before a hit)
  - b) up to 100 document accesses (average of 50 before a hit)

# Technicalities: <https://meltdownattack.com/>

- Read a single bit at a time
  - i.e. target array only has two entries
- Need to space out elements in target array by 4K bytes
  - to work round large cache blocks and memory prefetching
- Illegal access throws an exception
  - need to deal with this in some way
- Can read data at up to 0.5 MB/s (one bit at a time!)
  - with an error rate of 0.02%

# Why is Meltdown?

out-of-order execution

authorise & access simultaneously

hardware

Meltdown exploits a **race condition**, inherent in the **design of many modern CPUs**. This occurs between **memory access** and **privilege checking** during instruction processing. Additionally, combined with a **cache side-channel attack**, this vulnerability allows a process to bypass the normal privilege checks and read data from the operating system and other running processes. This vulnerability allows an unauthorized process to read data from **any address that is mapped to the current process's memory space**.

time taken to read data from user array

operating system

all RAM

# Mitigation

- Give up on GI1: “office should operate quickly”
  - don't store all your data in the same office
  - need to send Igor to a different office to access the safes
- Operating System
  - keep the program memory and system memory separate
  - OS has to actively switch between user and kernel memory
  - introduces additional context-switching overhead
- Effect
  - making OS calls will be slower
  - could adversely impact IO performance

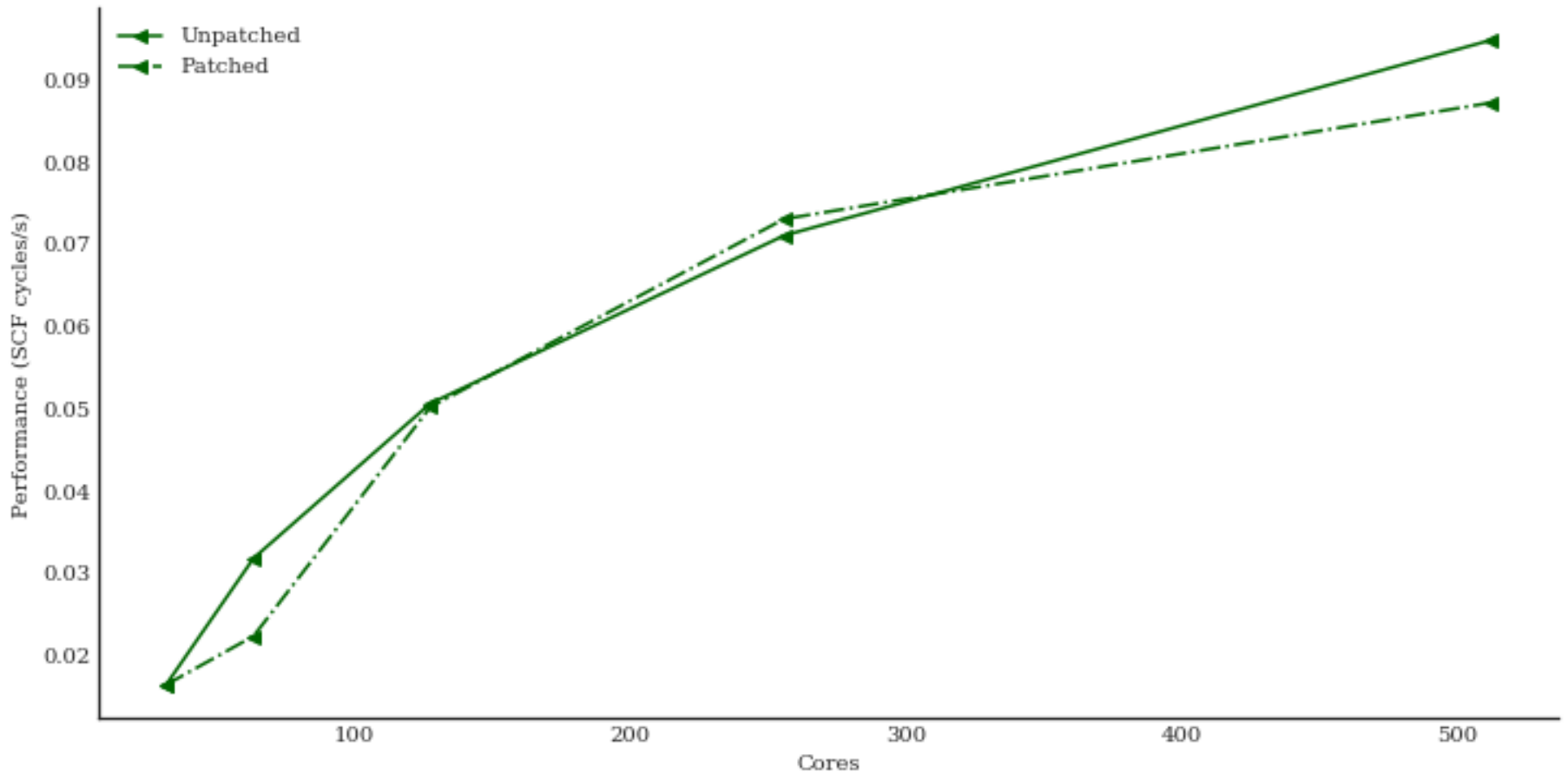
# CSD3 Skylake at University of Cambridge

[https://github.com/ARCHER-CSE/archer-benchmarks/blob/master/analysis/Spectre\\_Meltdown\\_Patch\\_Impact.ipynb](https://github.com/ARCHER-CSE/archer-benchmarks/blob/master/analysis/Spectre_Meltdown_Patch_Impact.ipynb)

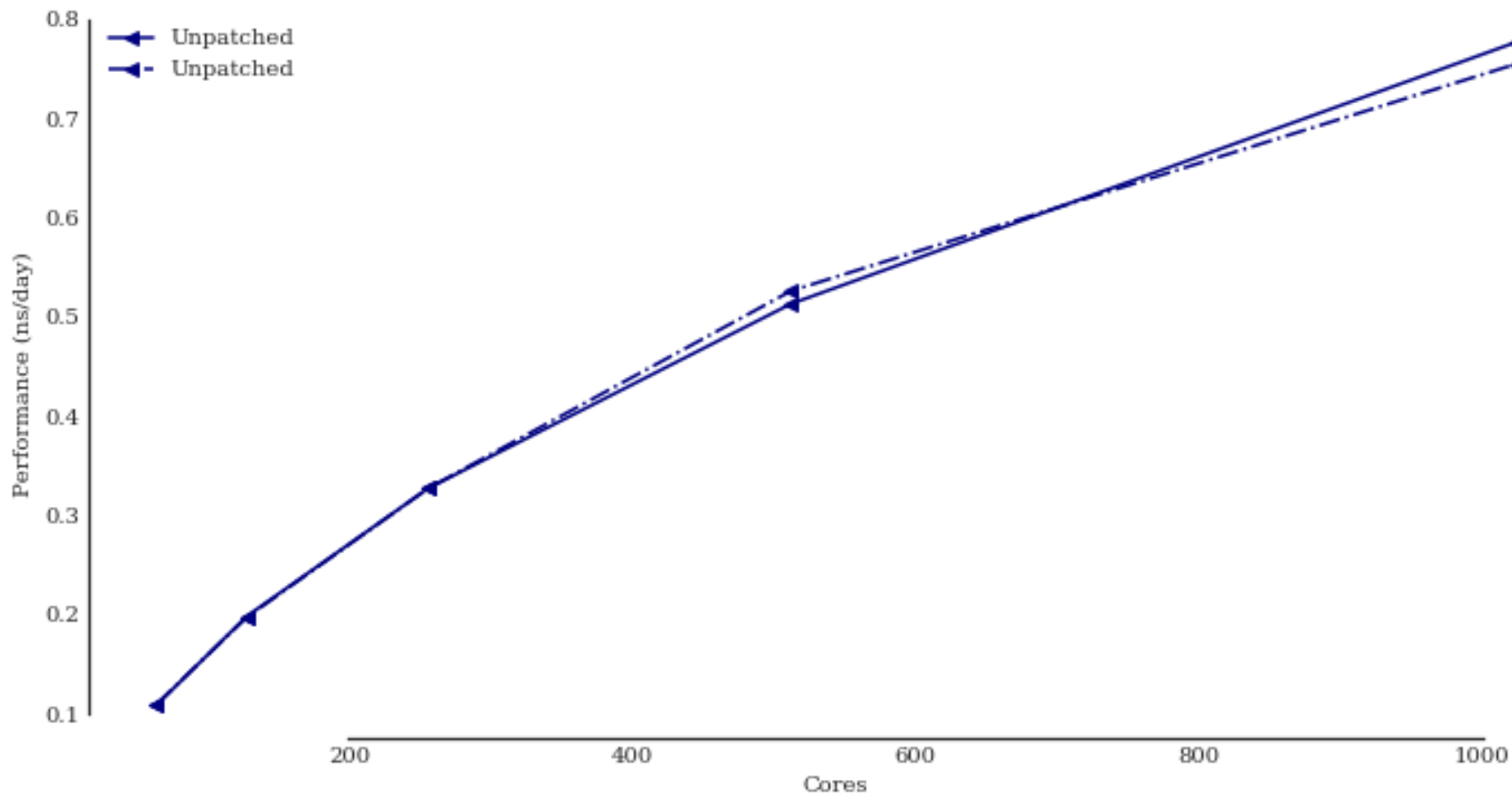
- “No significant performance differences ... apart from the synthetic test of parallel write performance (benchio) where we see a 10-15% performance drop”
  - “This variation is within the variation we would expect from a parallel file system during normal operation so may not be associated with the patching process.”
  - “The results for the random ring latency in the HPCC b\_eff benchmark show some odd features that require further investigation.”



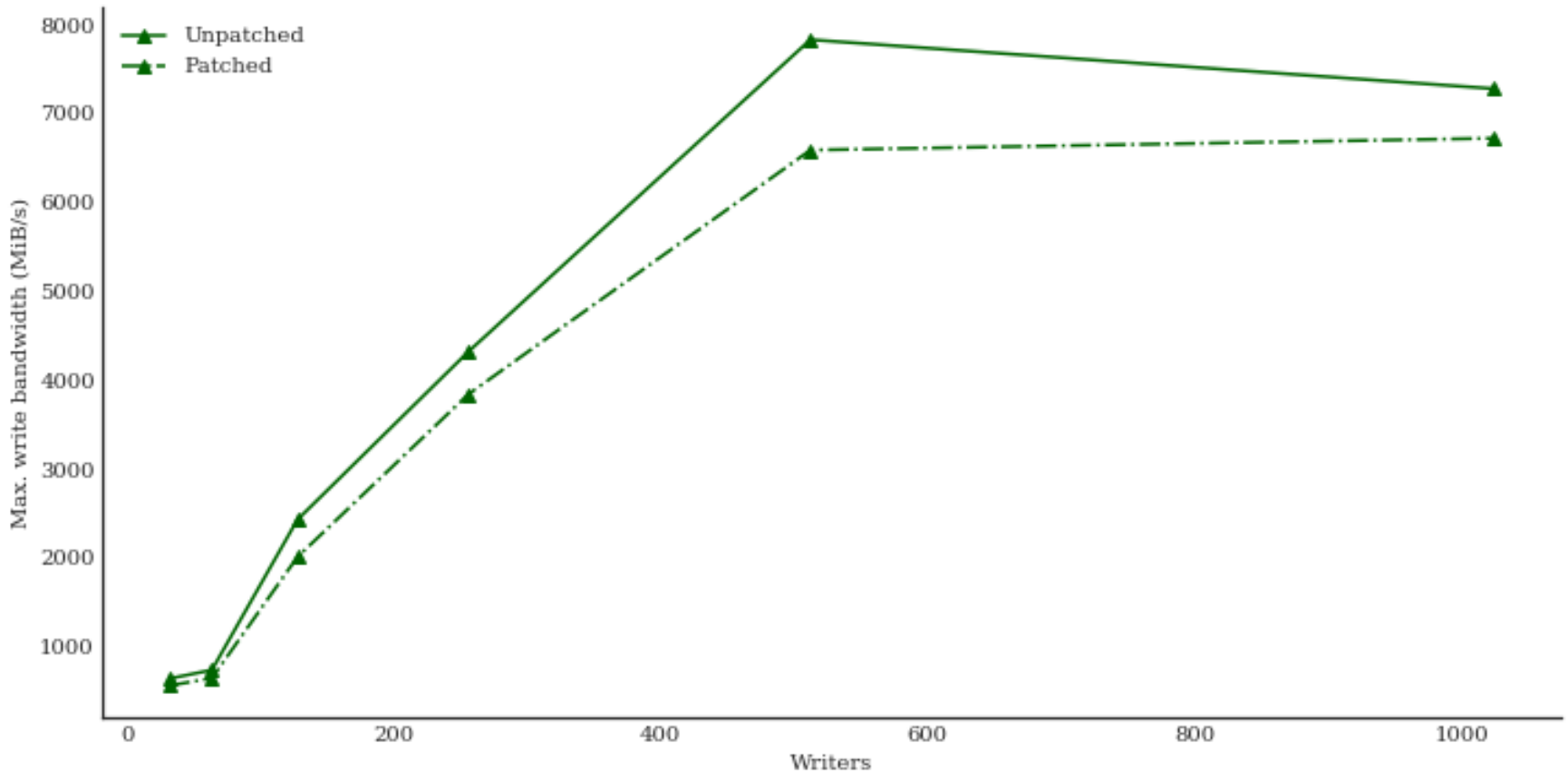
# CASTEP AI Slab (al3x3)



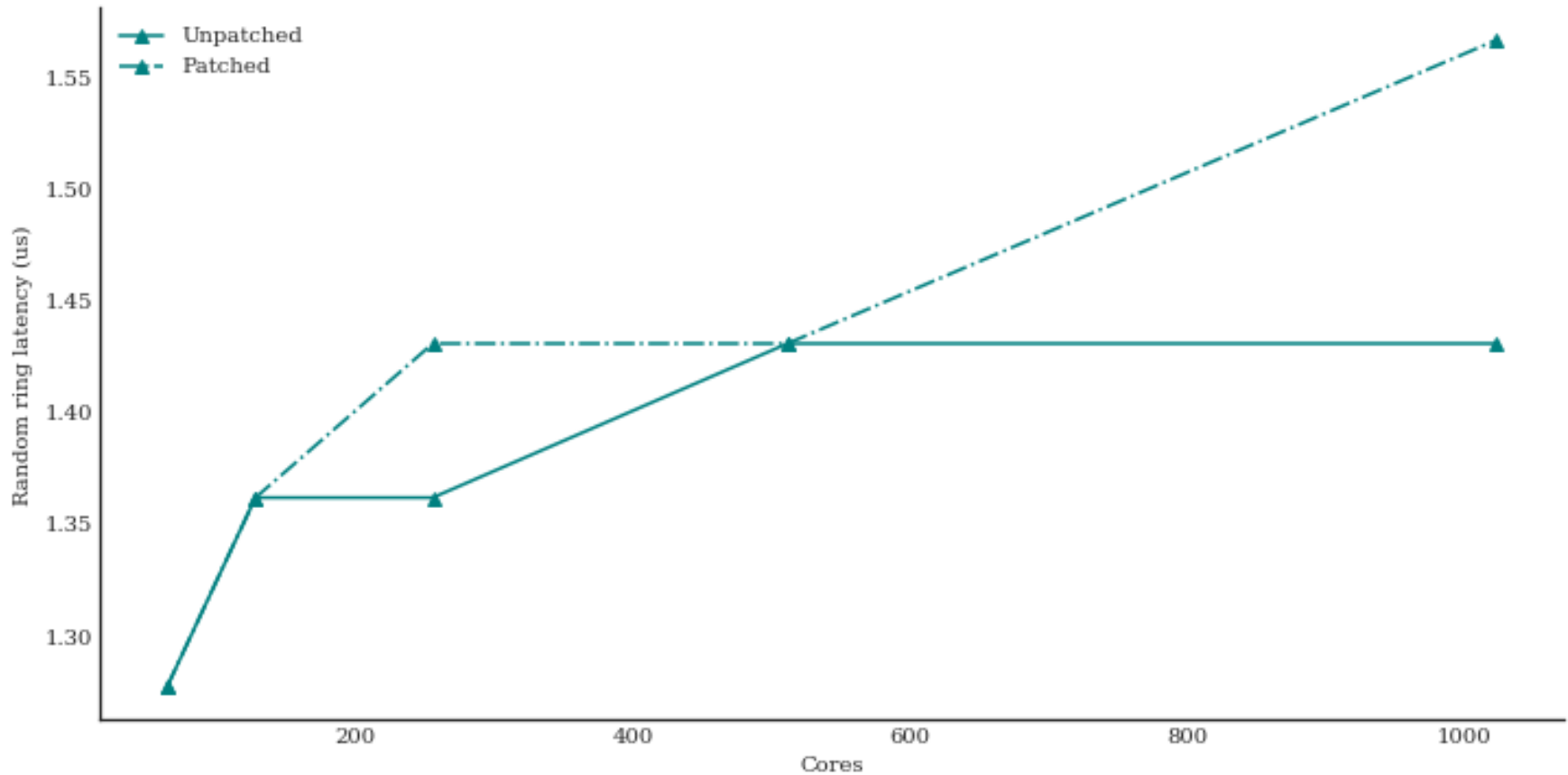
# GROMACS



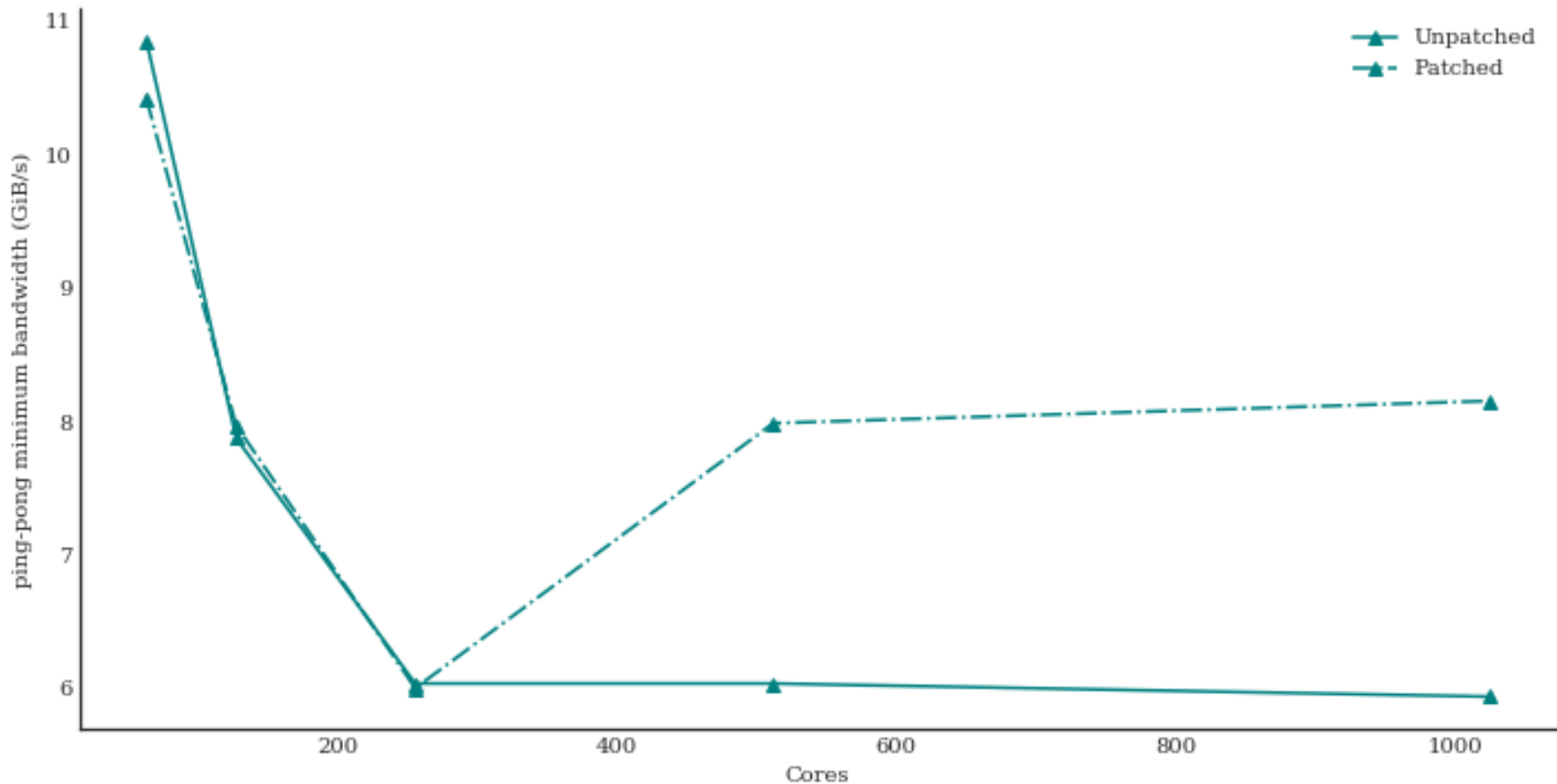
# Benchio – parallel MPI-IO to Lustre



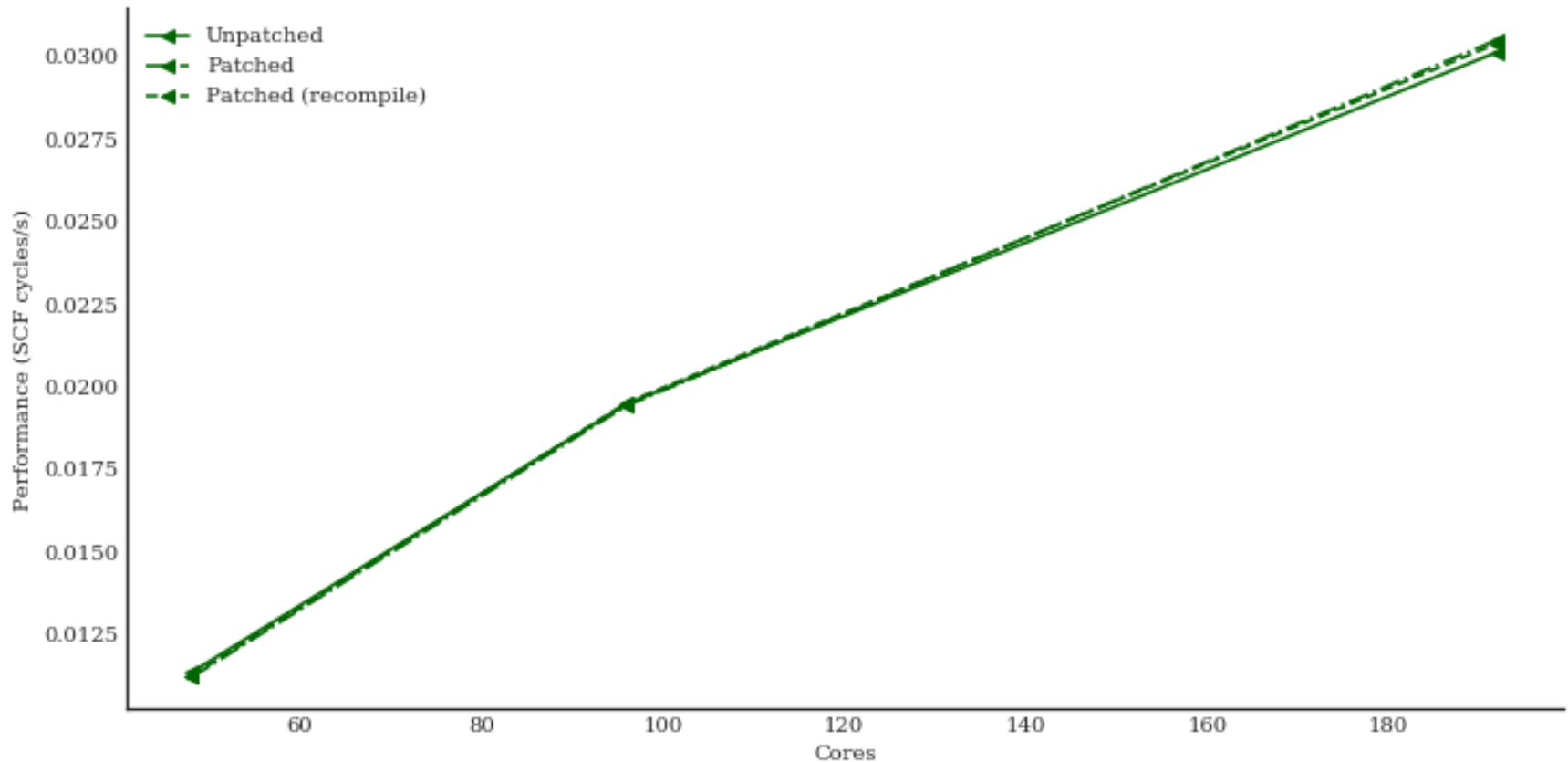
# Random Ring Latency



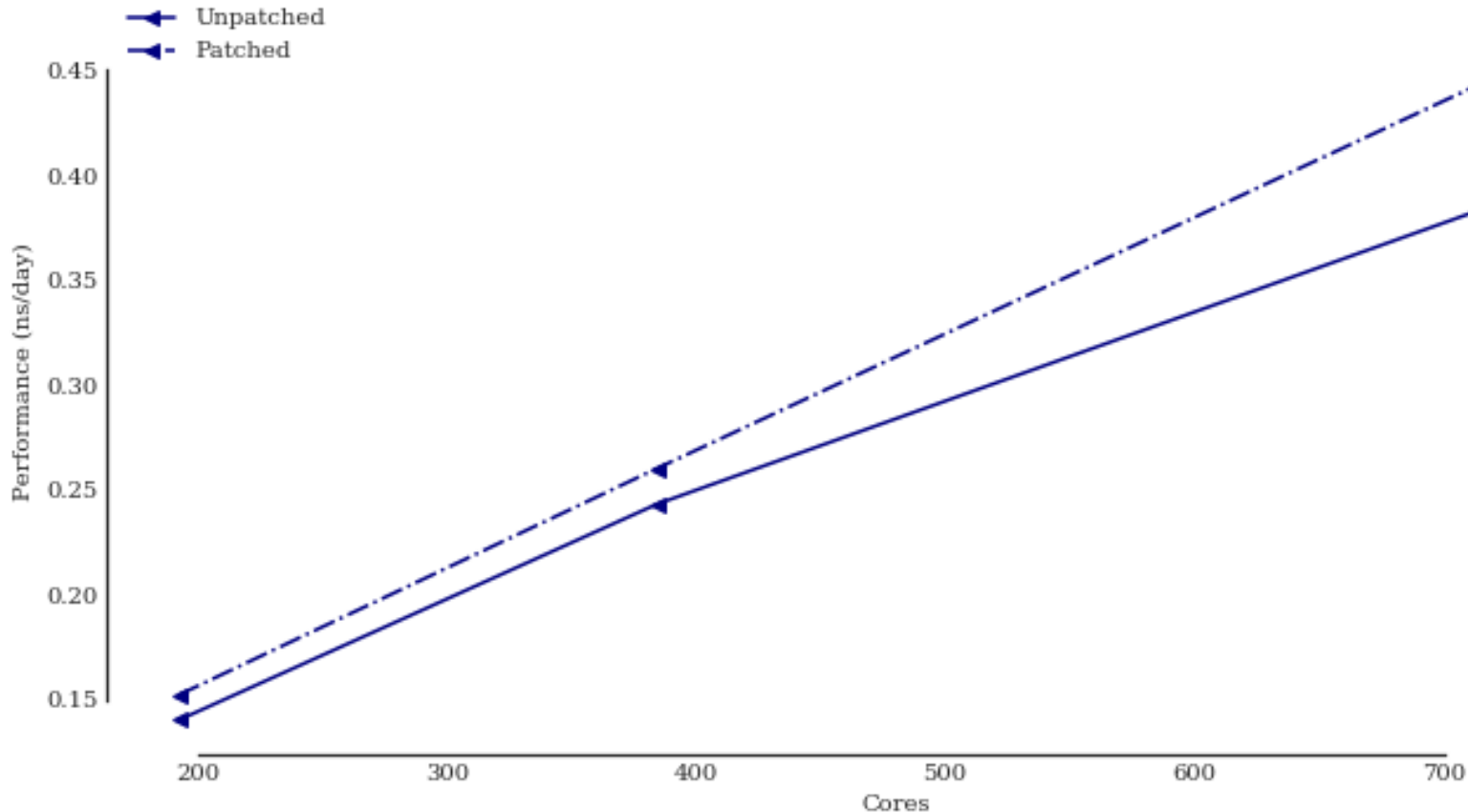
# Minimum Ping-Pong Bandwidth



# ARCHER (TDS): CASTEP



# ARCHER (TDS): GROMACS



# Performance

- Little or no impact seen so far
  - what effects would a normal OS update have?
- IO may suffer
  - benchio optimised for small number of OS write calls
    - little effect from patch
  - IO in many real applications may not be so simple
    - could result in many OS write calls and potential for greater impact
  - needs further investigation
- Are there real security implications on single-user system?
  - On ARCHER compute nodes, OS unlikely to have any info relevant to other users



# Summary

- At its core, Meltdown is remarkably simple
- Completely analogous to everyday office situation
- For many years, CPU + OS design focused on speed ...



# Acknowledgements

- Thanks to the following EPCC staff for useful conversations:
  - Stephen Booth
  - Rupert Nash
  - Nick Johnson
  - Ally Hume
  - Andy Turner
  - Adrian Jackson