Meltdown for Dummies

The road to hell is full of good intentions

Reusing this material

Saint Bernard of Clairvaux

Good Intentions

Told you so!

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**.

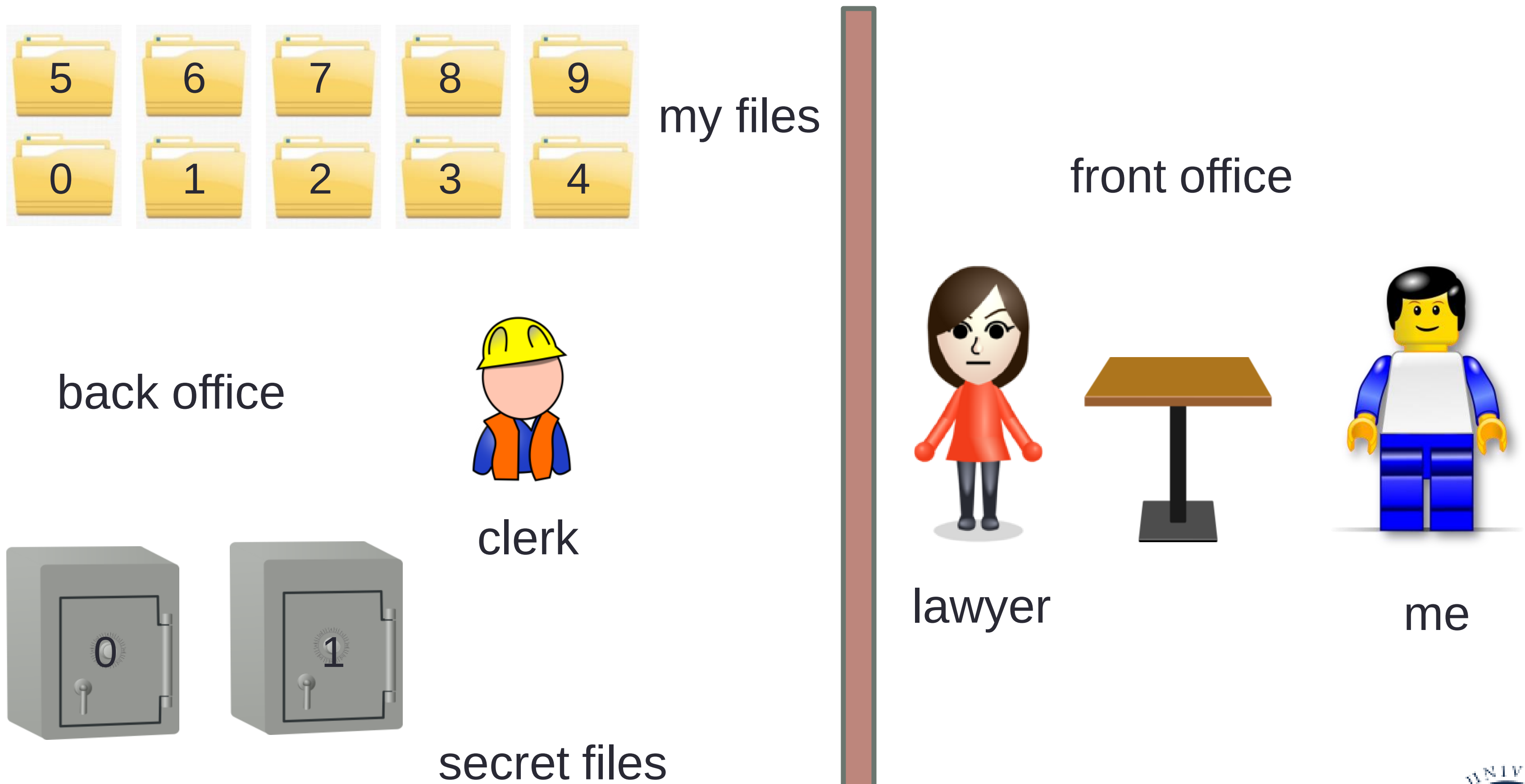
Analogy

- 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?

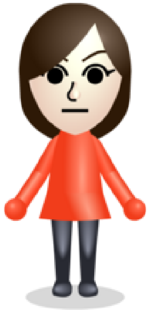


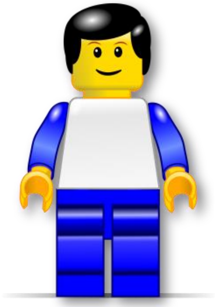
Computer equivalents

user memory

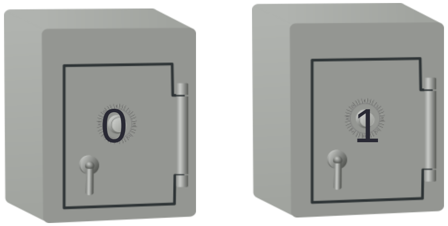
software

hardware





memory system

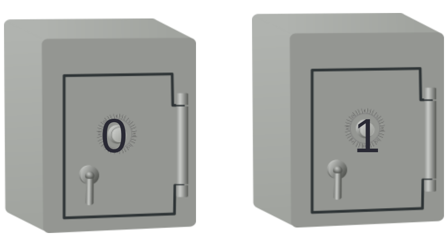


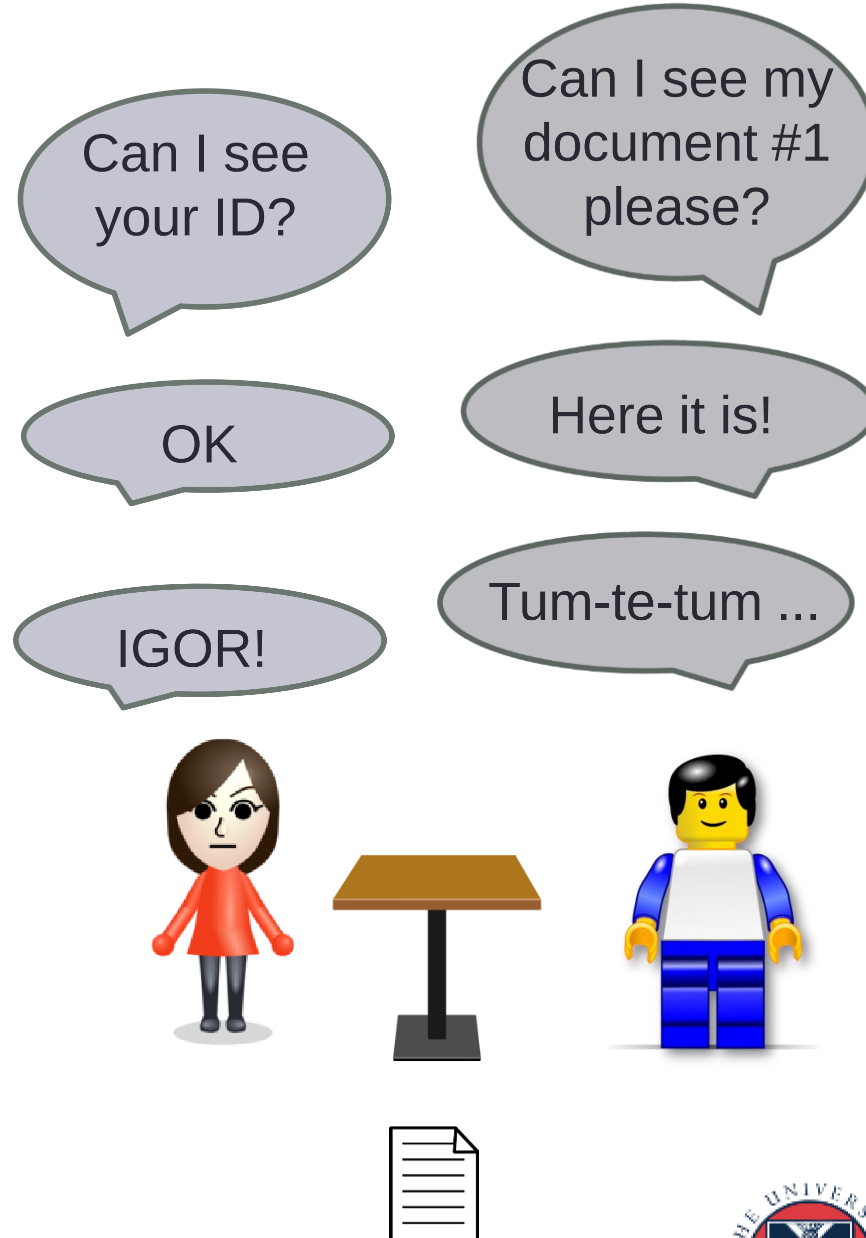
operating system

user program

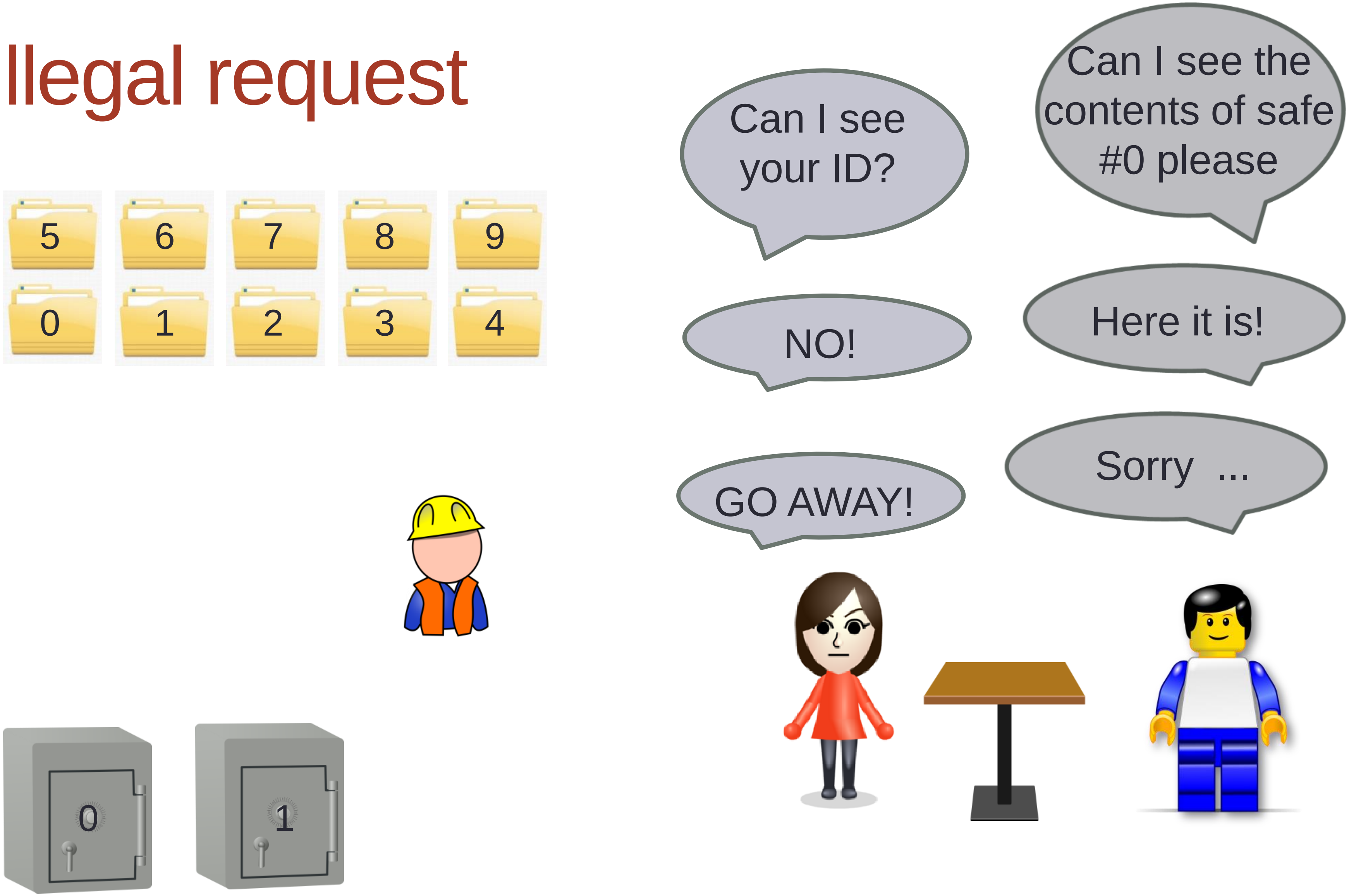
OS kernel memory

Memory request





Illegal request



GI1: 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

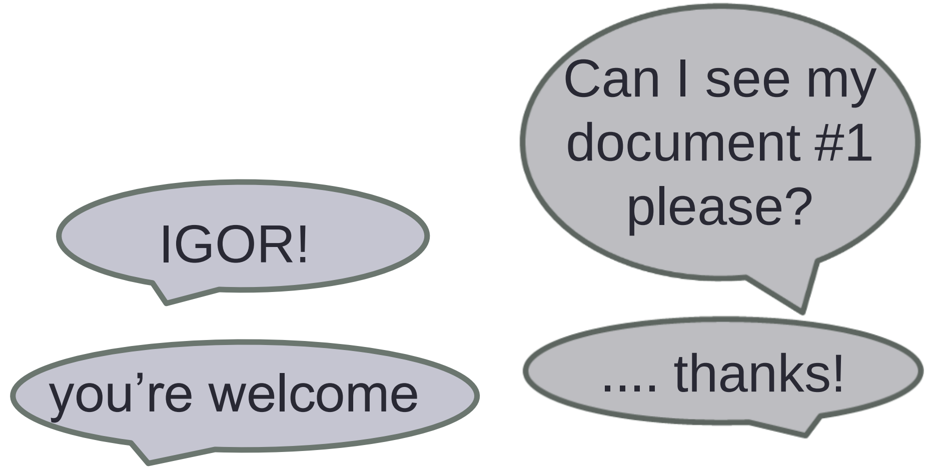
- ... will assume all accesses are valid for next sections

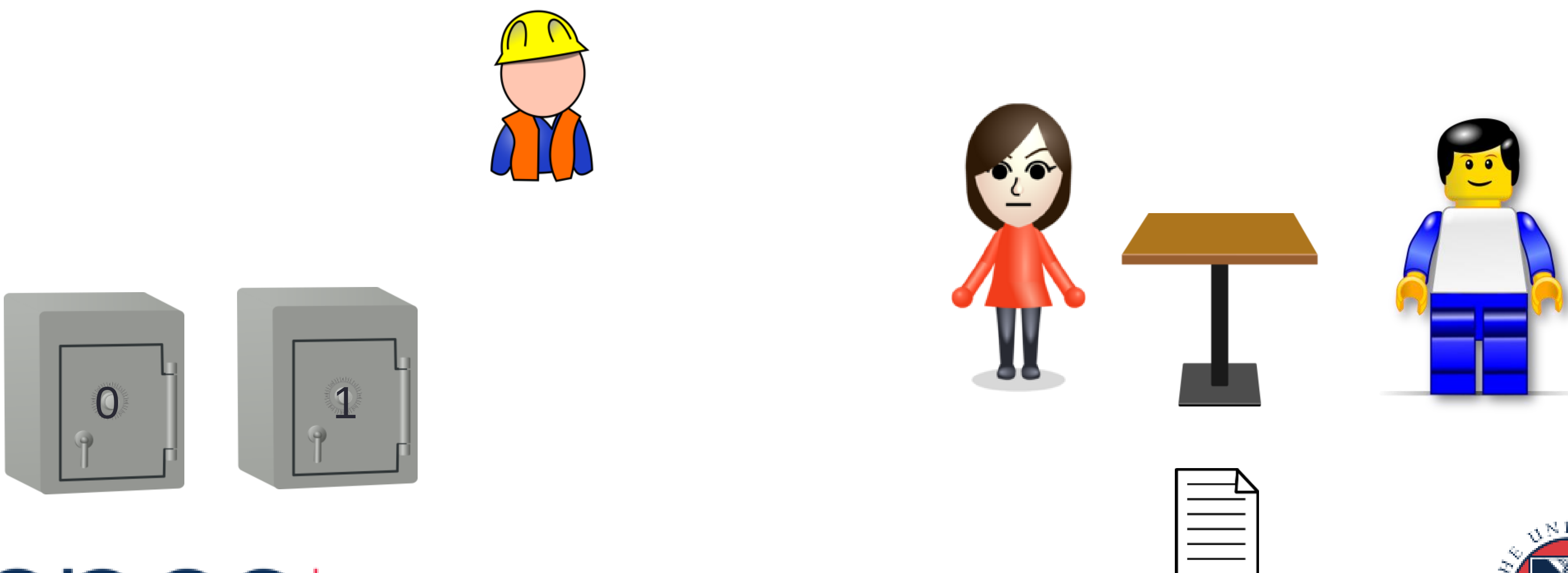
Unintended Consequence: UC1

- I can see the secret documents
  - so I can ask for them

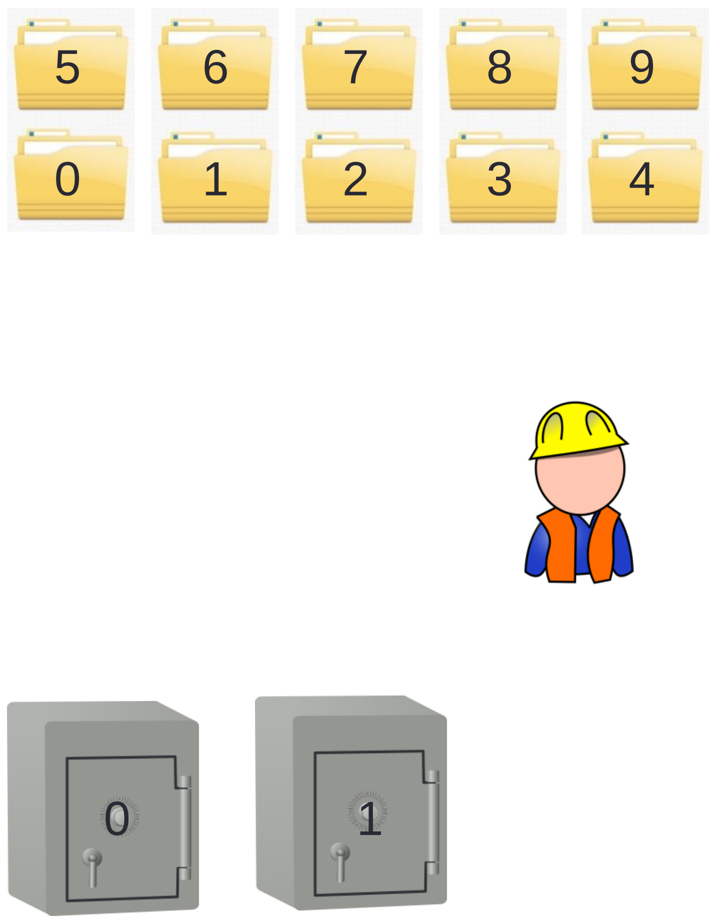
- If they were in a different office
  - I wouldn’t even be able to even ask for them

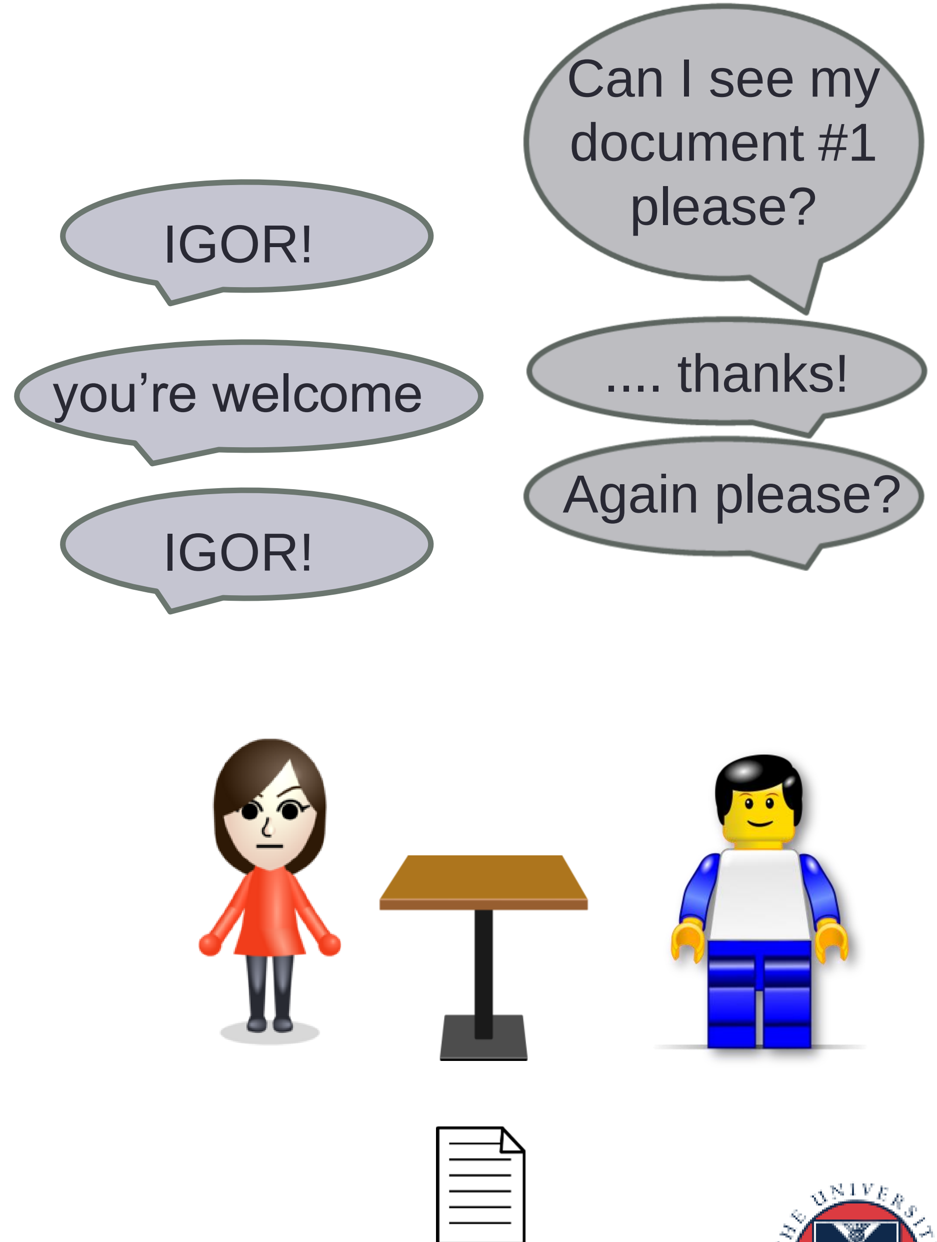
Memory request



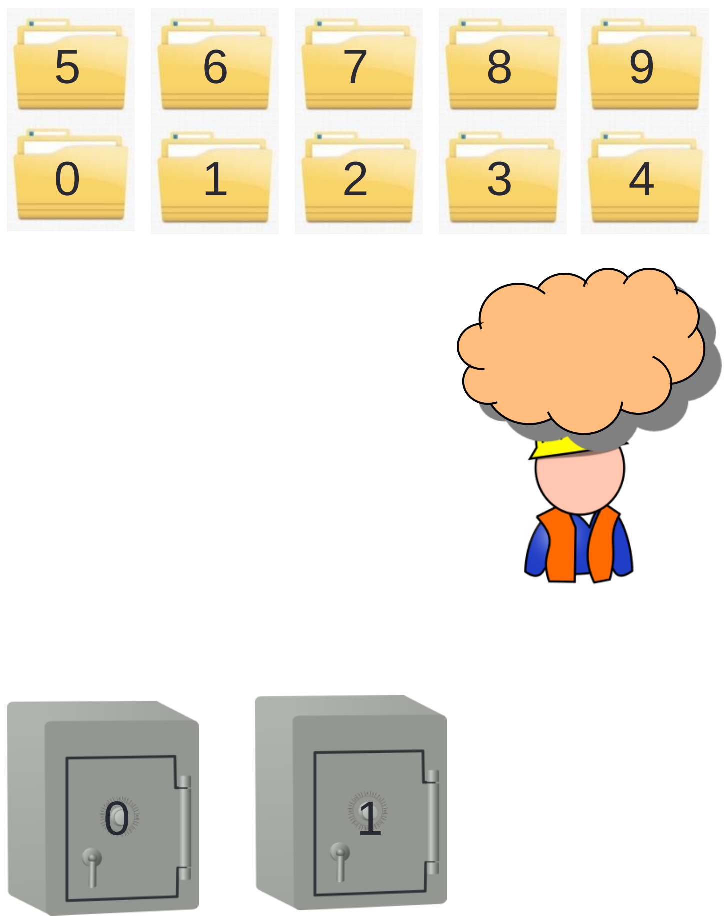


Memory request

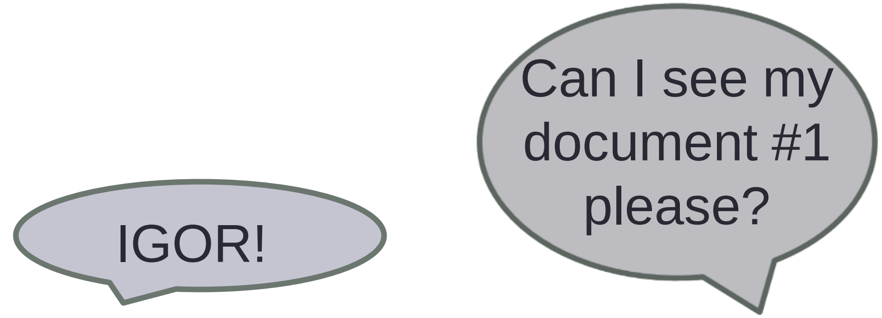


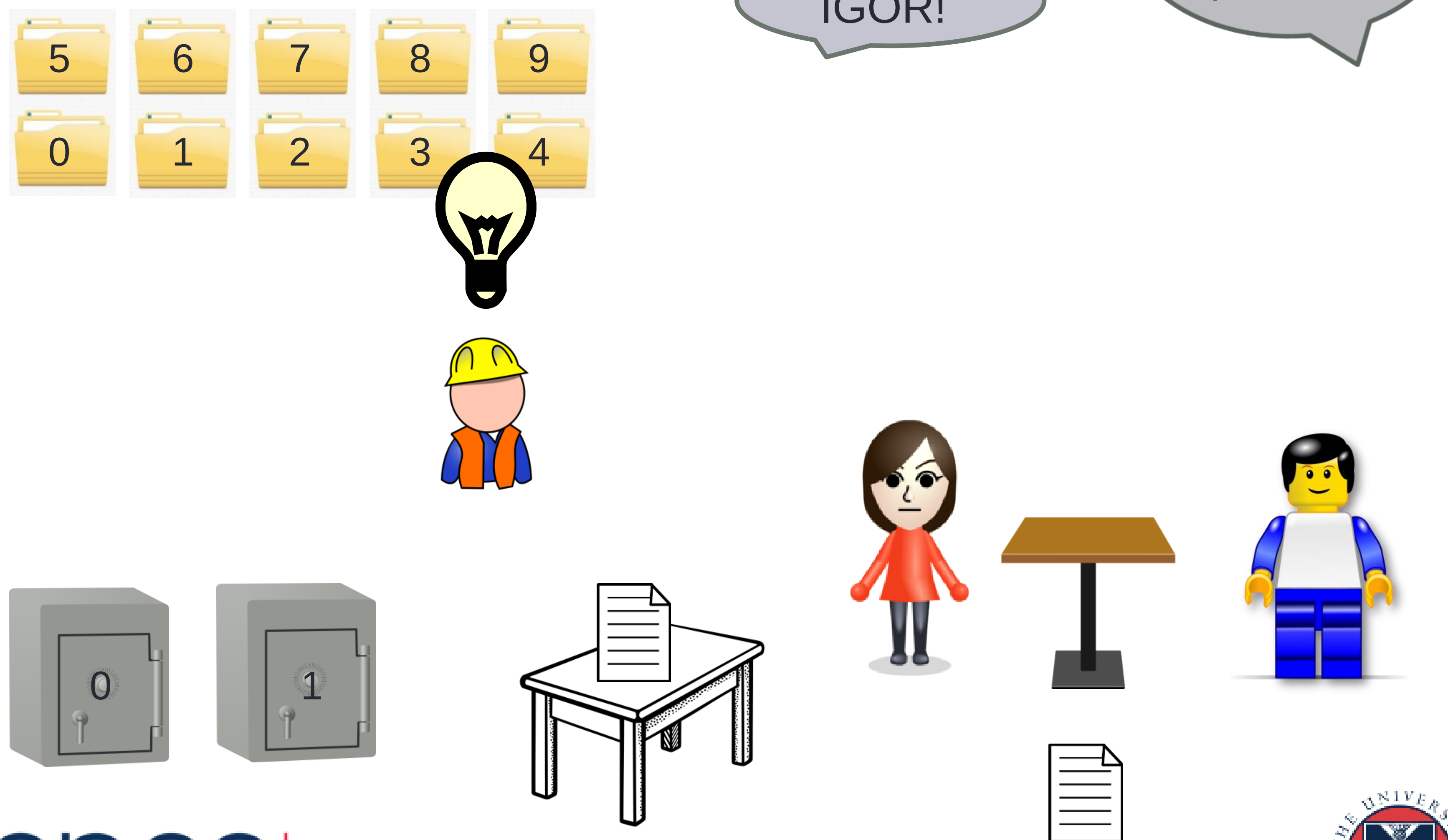


Memory request



Memory request





Memory request

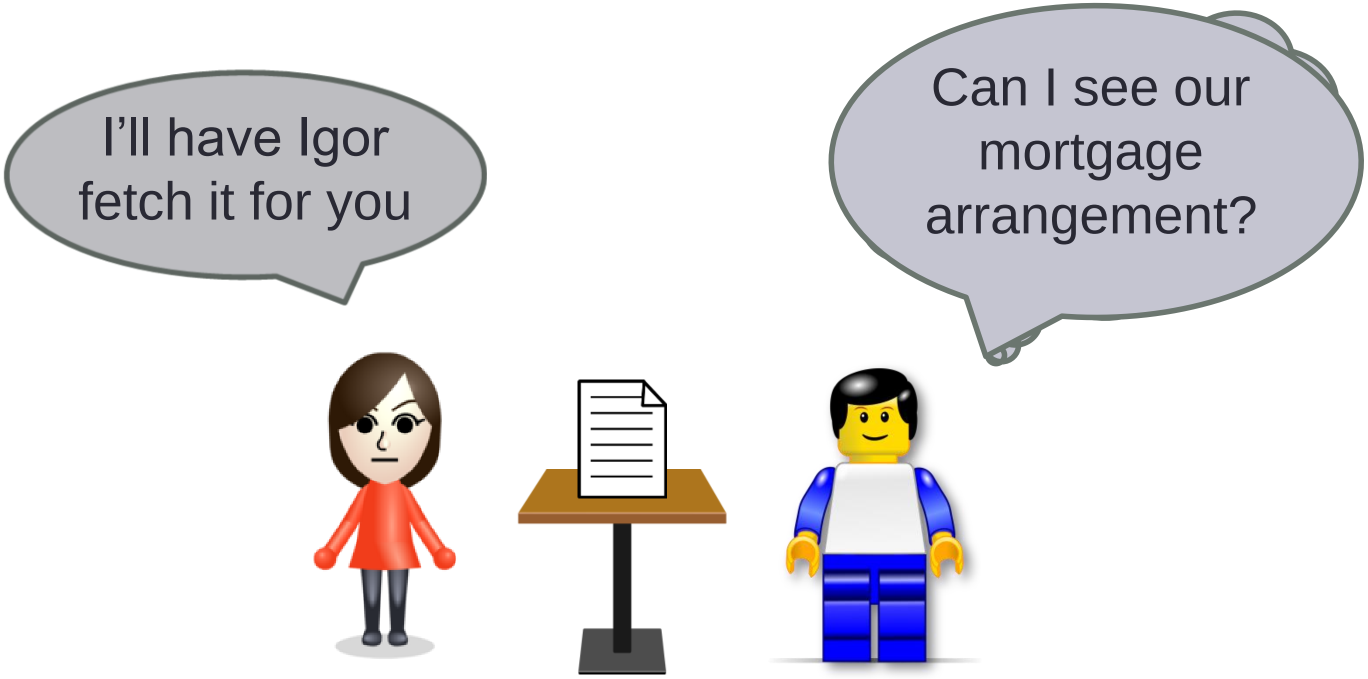
GI2 : memory access should be fast

- Why keep going back to document store all the time
  - keep recent documents close to hand ...
  - ... but still secure in the back office

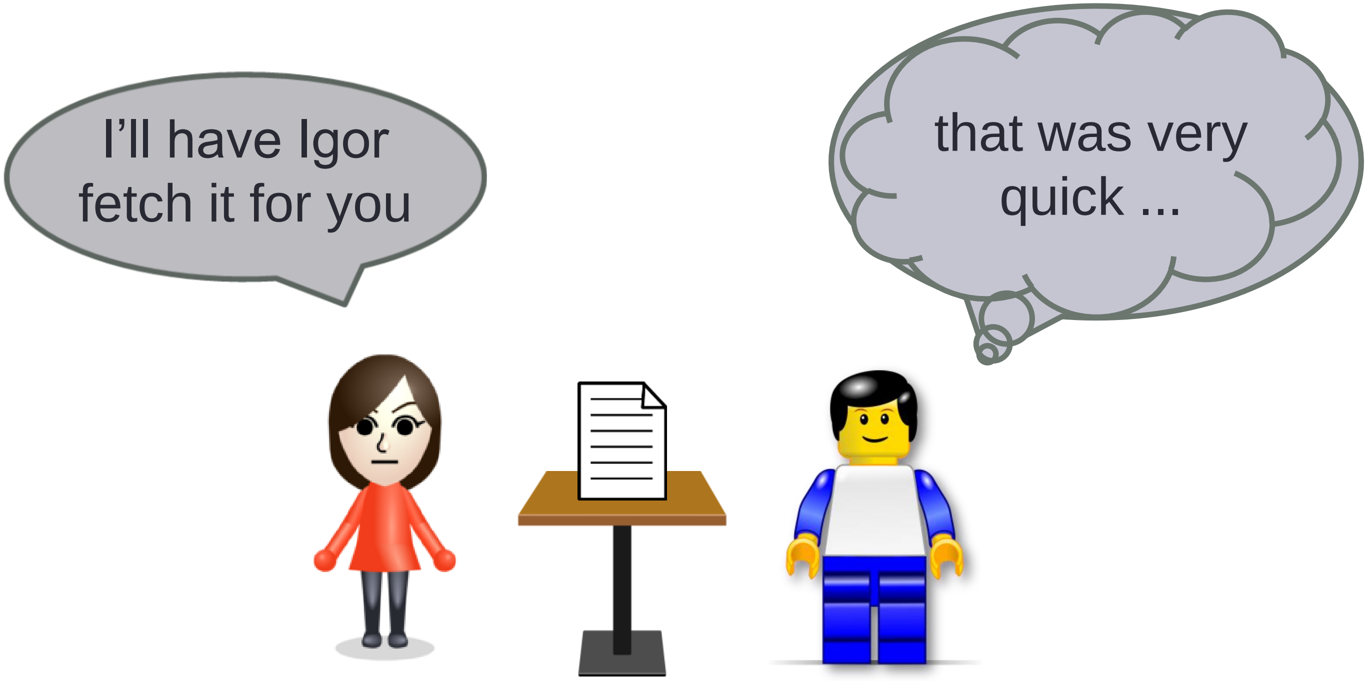
- Computer
  - memory system keeps copies of recent data in fast cache memory
  - reads and writes apply to the cached copy
  - done automatically in hardware – user just accesses memory

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



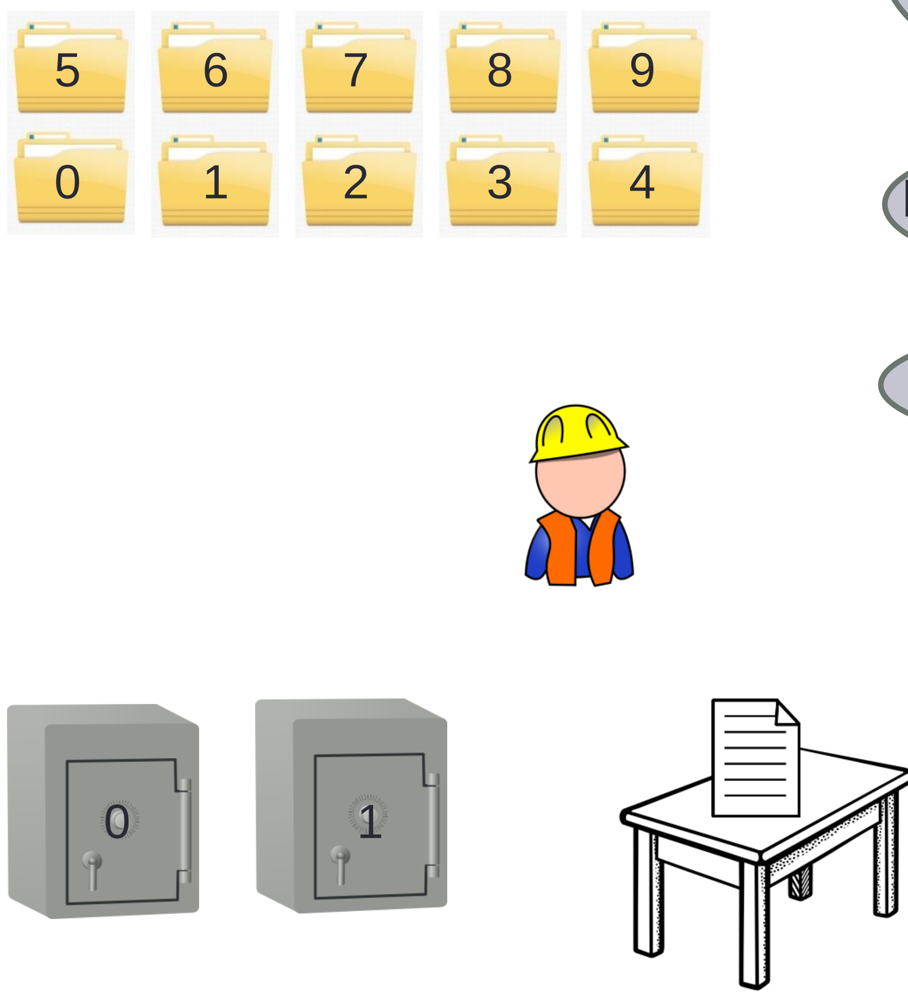
UC2: access to privileged information

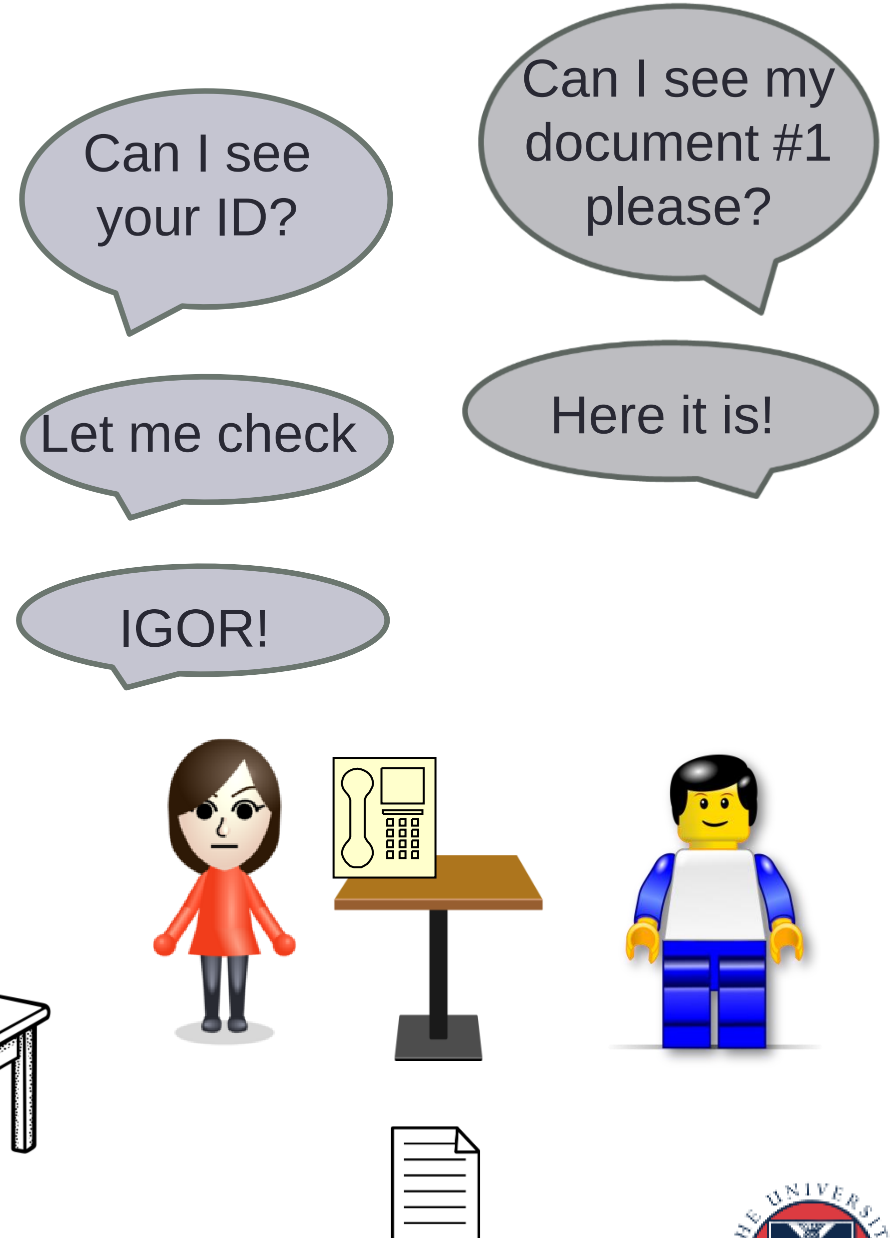
- Accessing documents that I am **allowed** to read can give me information that I am **not allowed** to have
  - through differences in timing

- 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



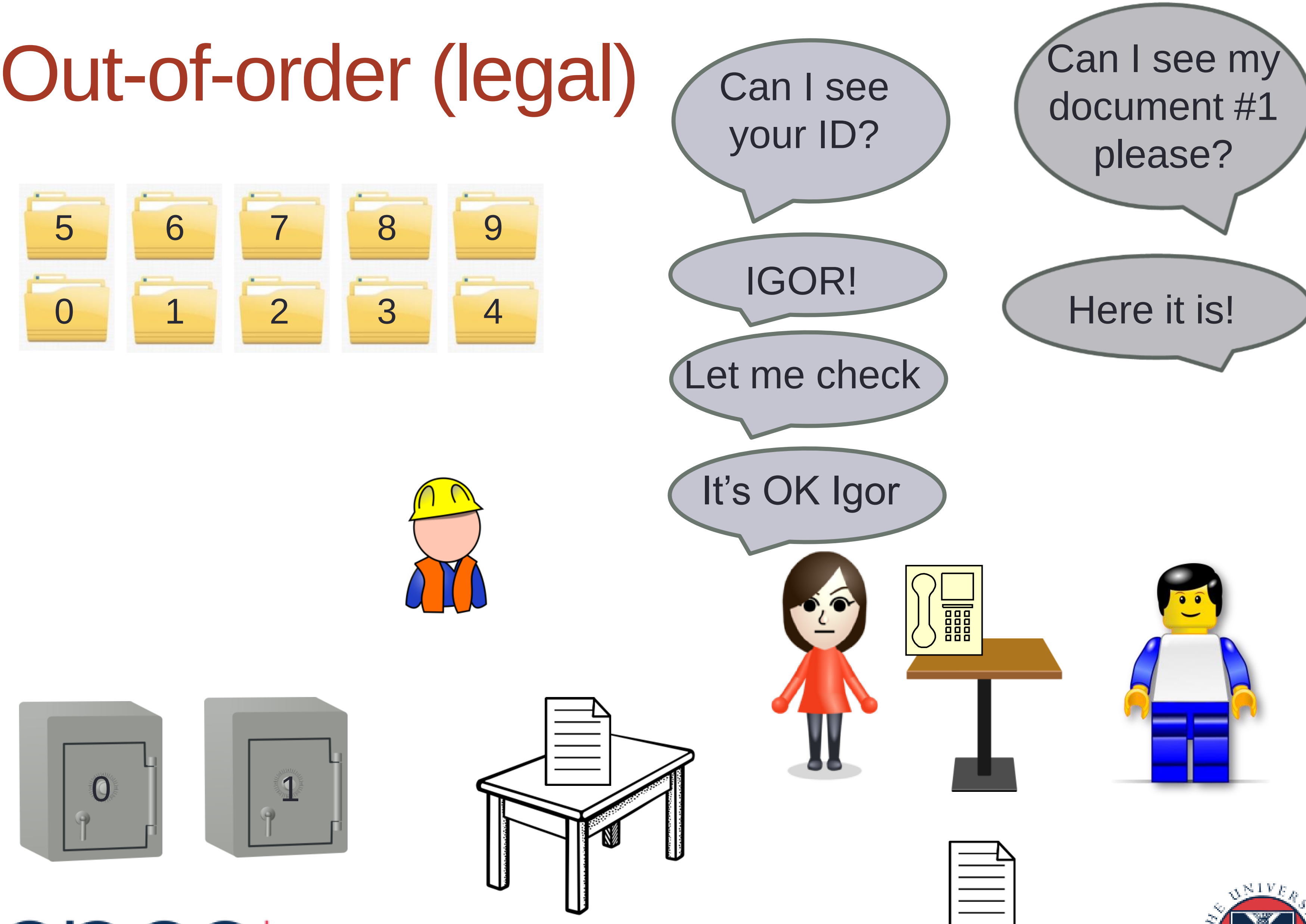


GI3: authorisation should be fast

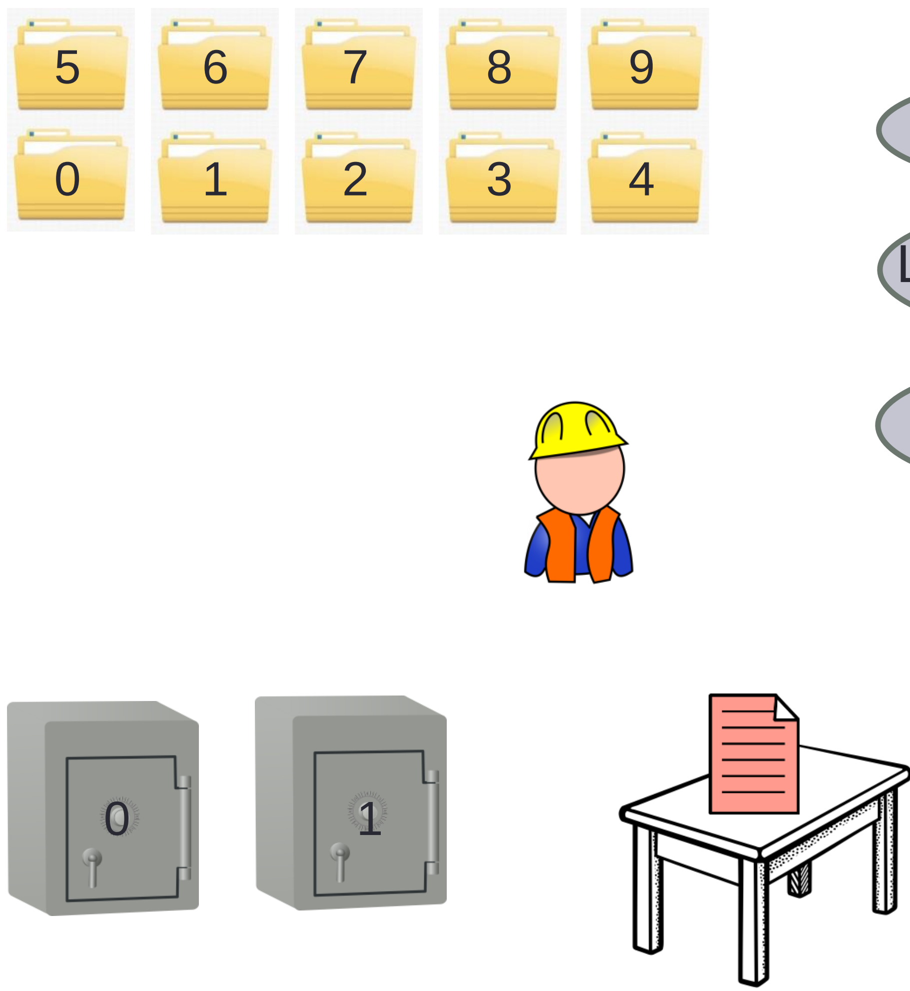
- Why wait to tell Igor to go for the document?
  - ask for the document **at the same time** as checking authorisation
  - don't deliver the document until authorisation confirmed

- 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)



Out-of-order (illegal)



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

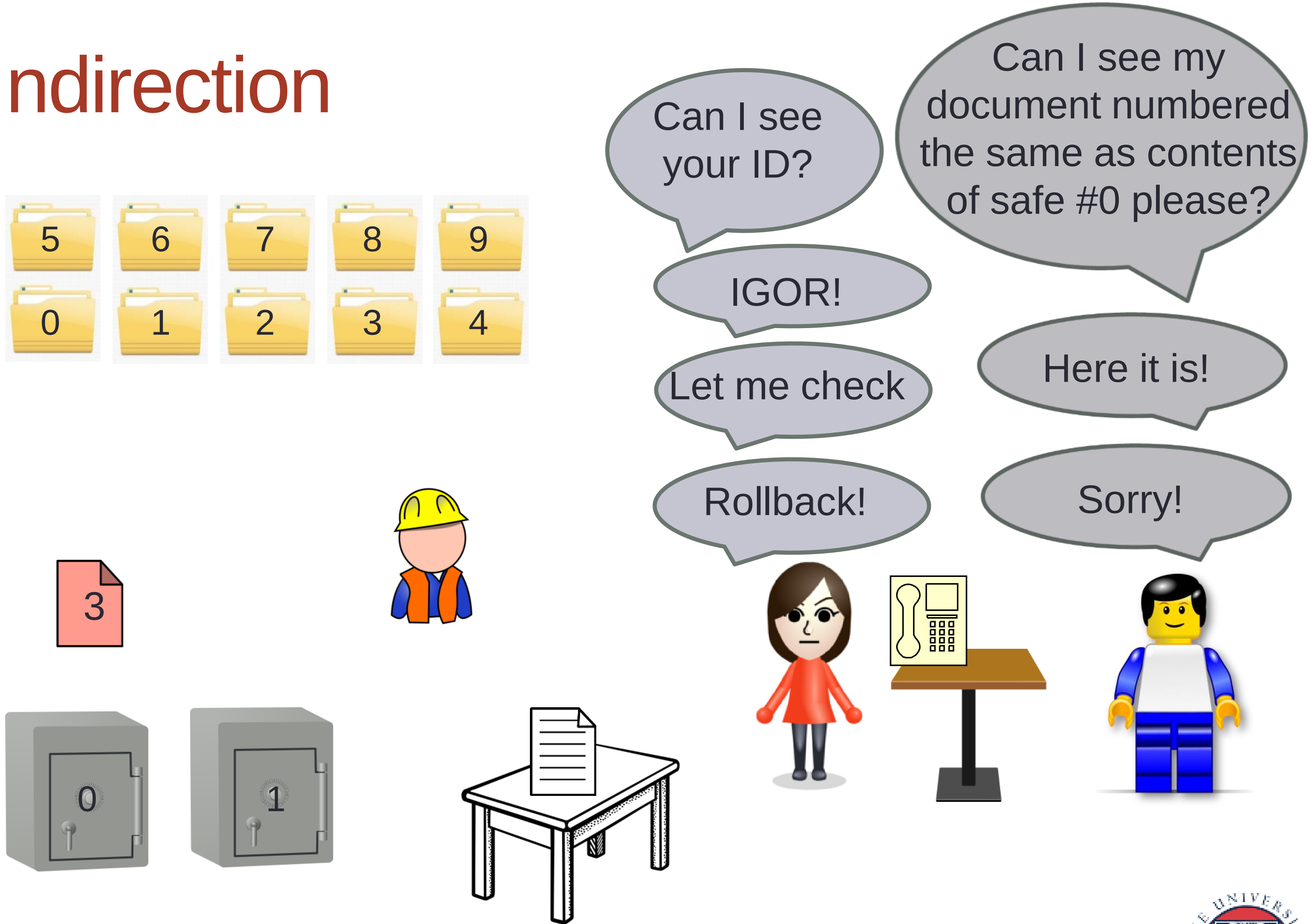
- Many programs have constructs akin to

```
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



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!

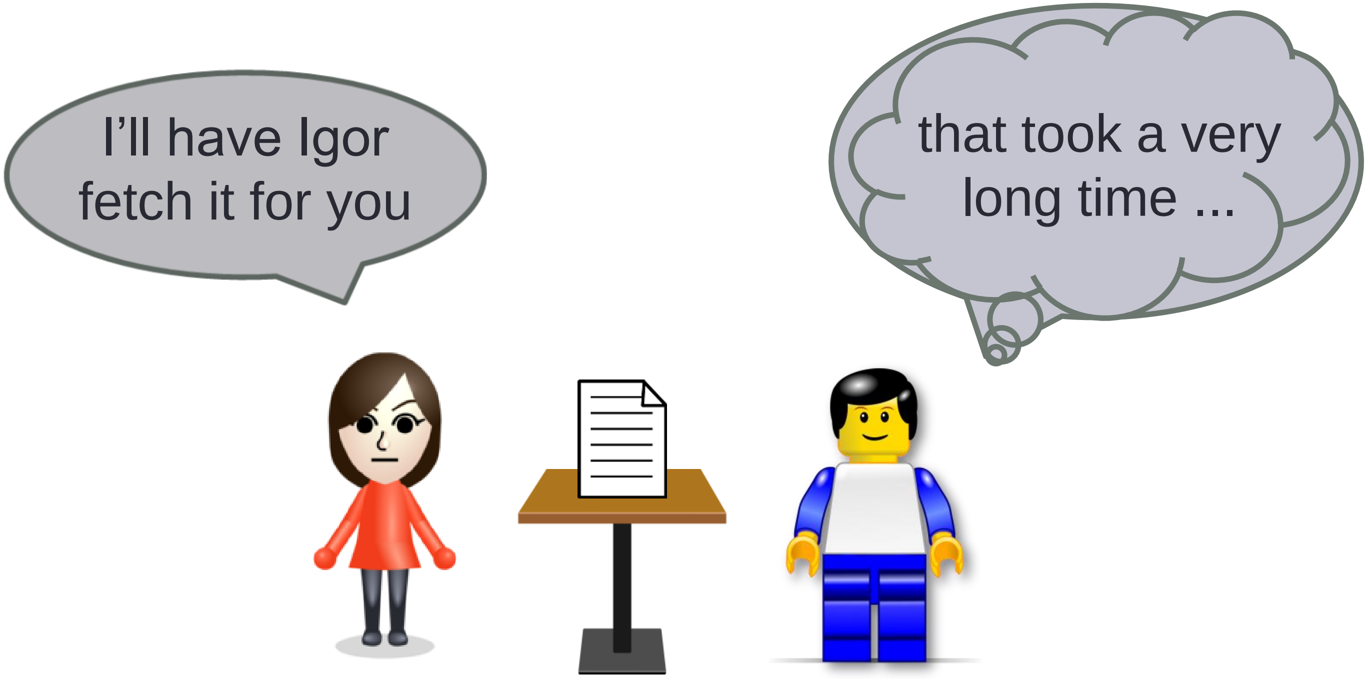
UC4: side effects of indirection

- The rollback is **incomplete**
  - there is a fingerprint in the cache
  - produced as a side-effect of accessing a secret document

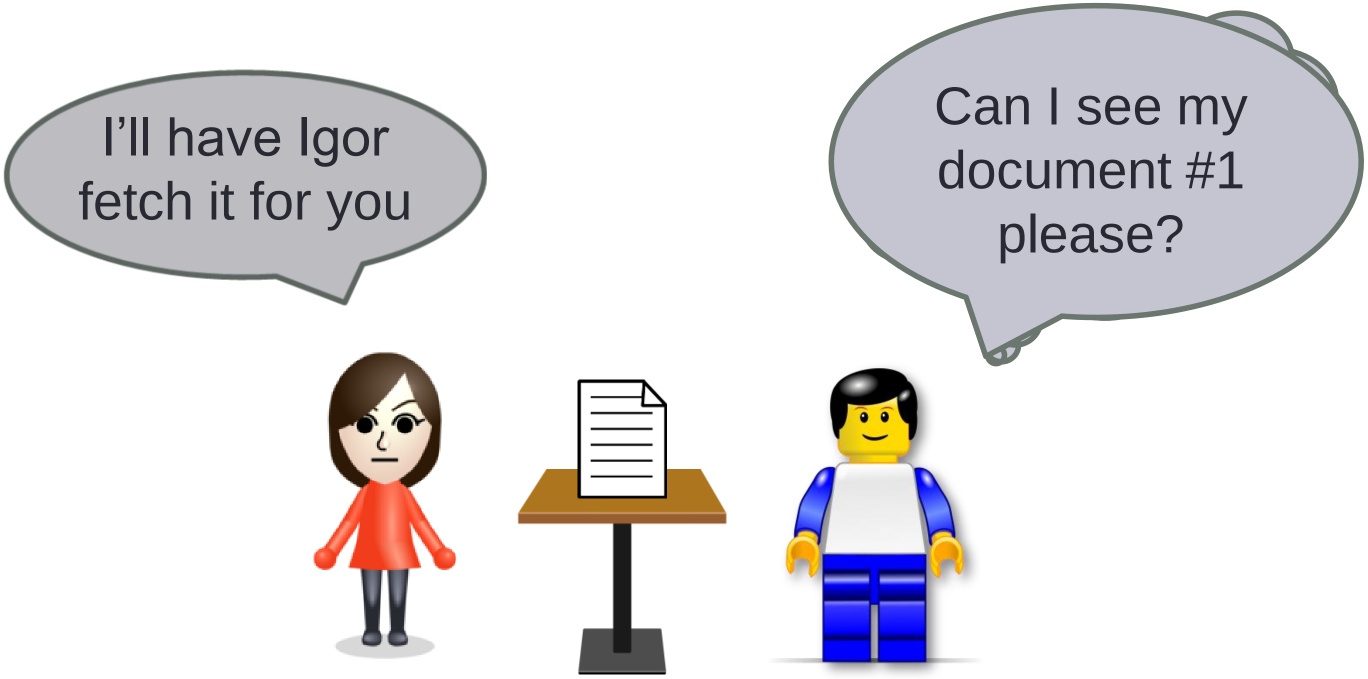
- I wasn't able to read the secret document

- But I can now read my own documents
  - and see how quickly they come back

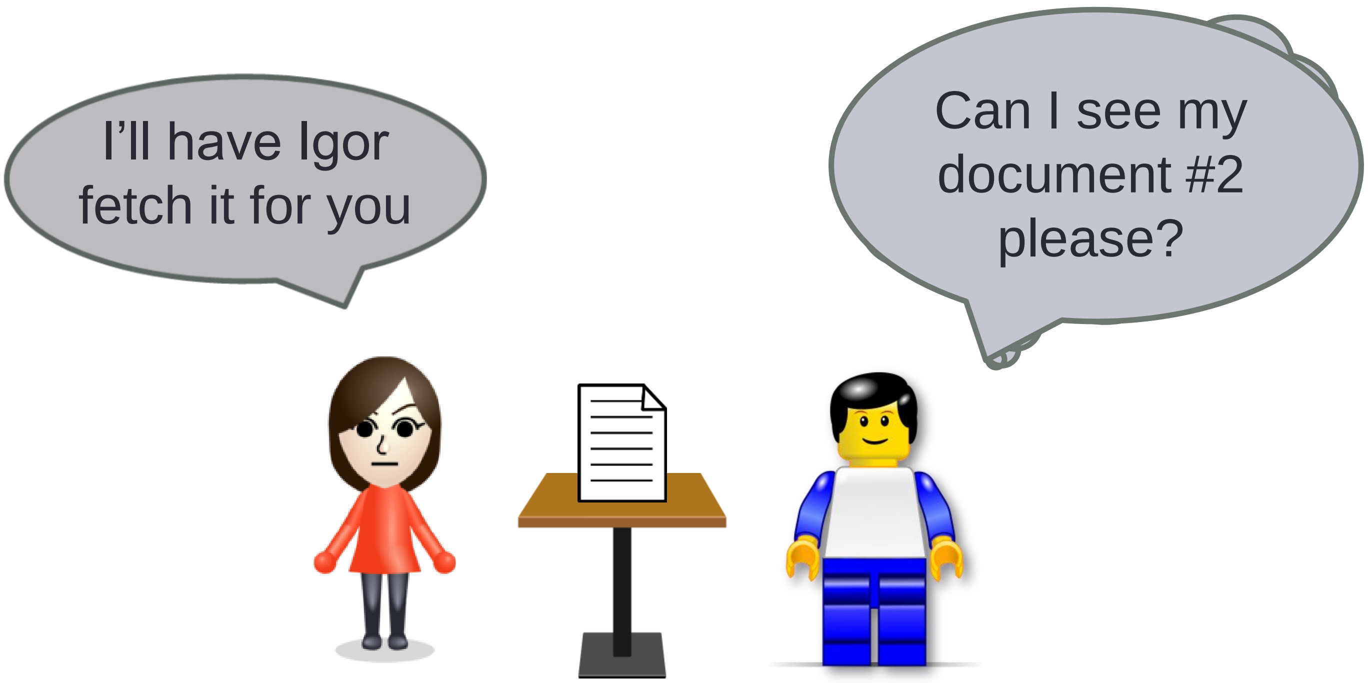
Inferring the secret data



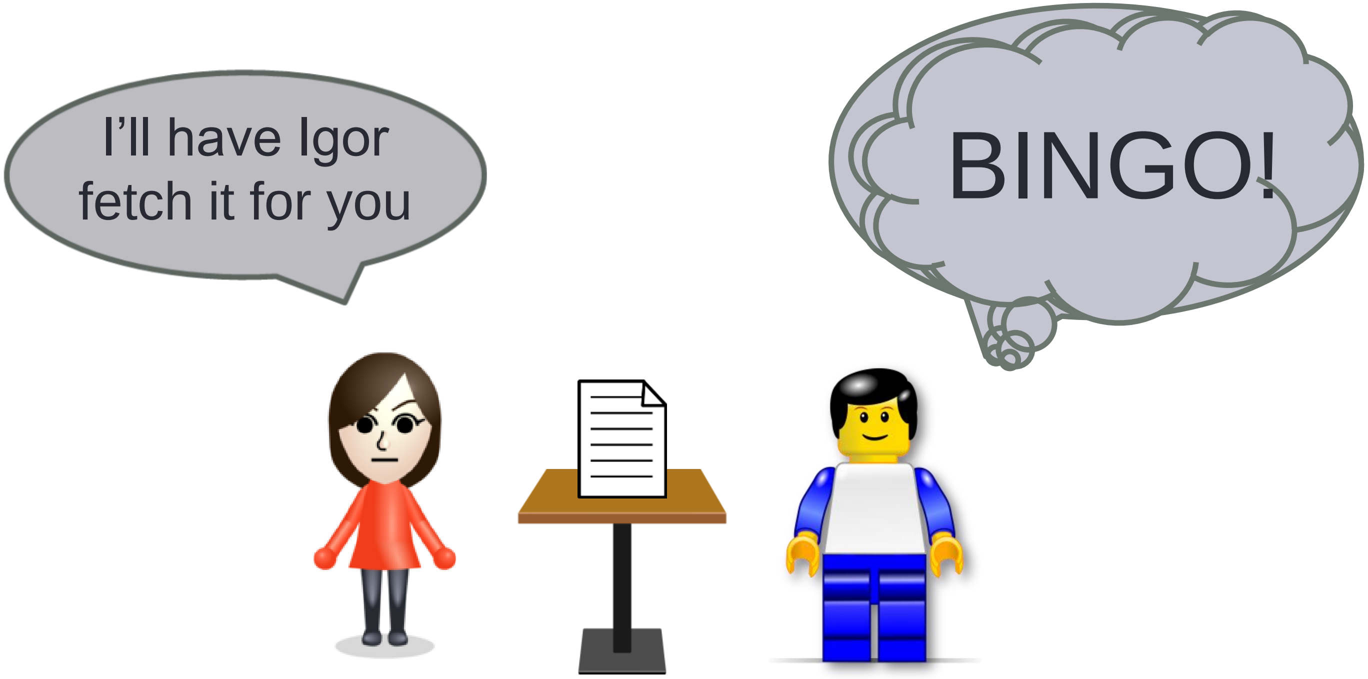
Inferring the secret data



Inferring the secret data



Inferring the secret data



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"
- Simples!

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%

What is Meltdown?

out-of-order execution

authorise & access simultaneously

Meltdown exploits a **race condition**, inheren
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
oit process from accessing data
e operating system and other ru
vulnerability allows an unauthorized
process to read data from **any address that is**
mapped to the current process's memory space.

hardware

operating system

time taken to read data from user array

all RAM

Mitigation

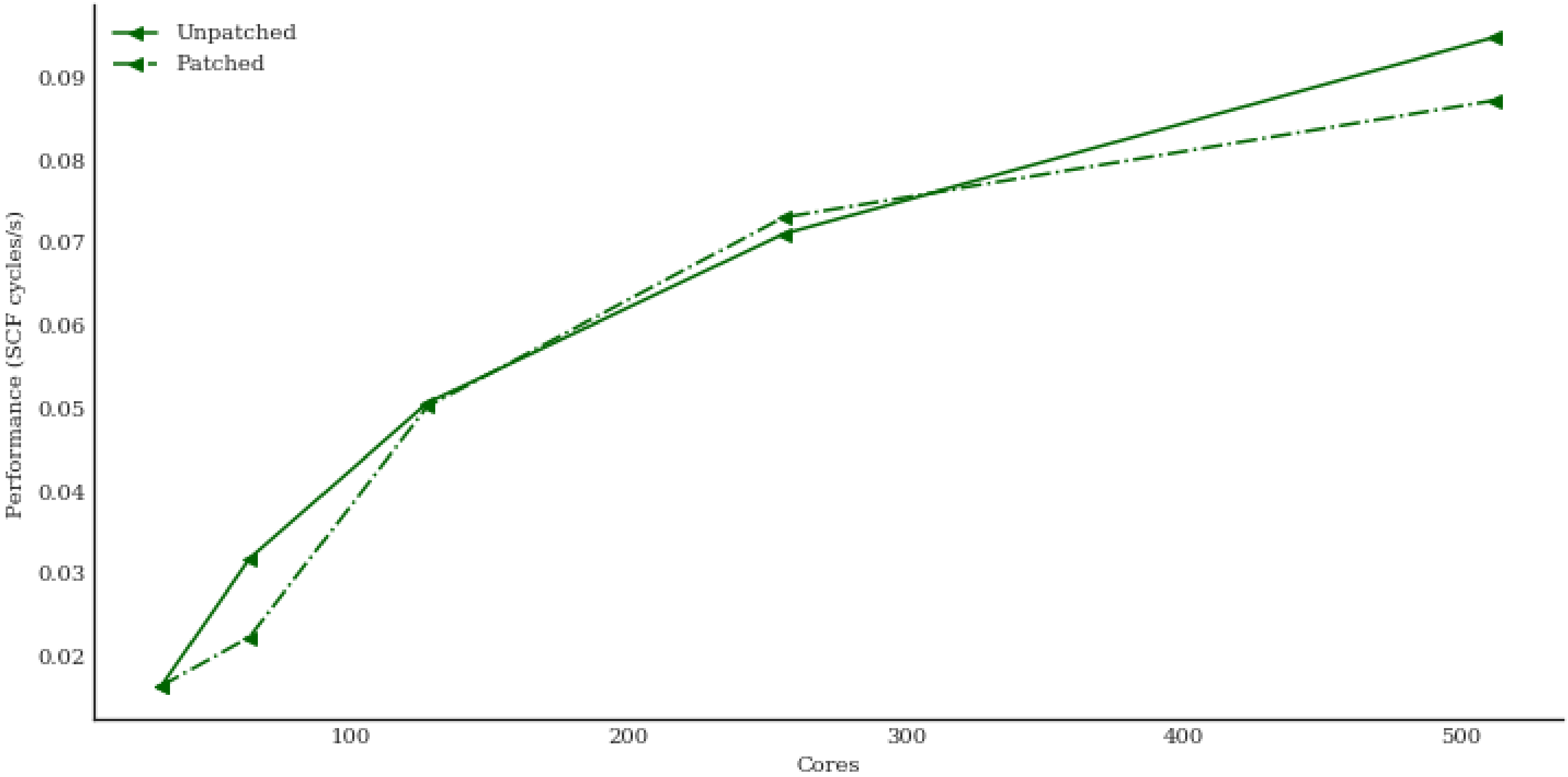
- Give up on GI1: “office should operate quickly”
  - don’t store all your data in the same office
  - need to send Igor to a different office to access the safes
- Operating System
  - keep the program memory and system memory separate
  - OS has to actively switch between user and kernel memory
  -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

CSD3 Skylake at University of Cambridge

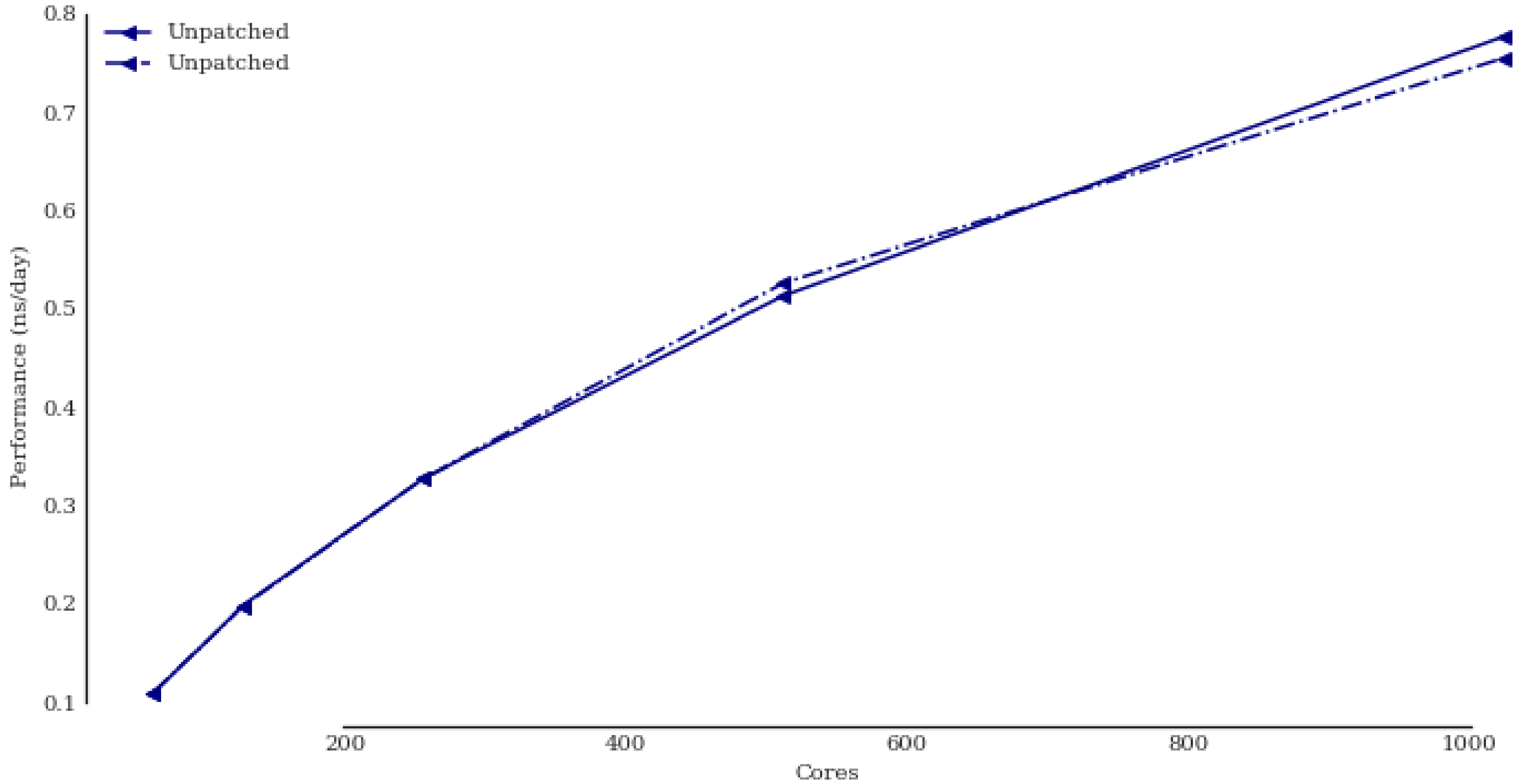
https://github.com/ARCHER-CSE/archer-benchmarks/blob/master/analysis/Spectre_Meltdown_Patch_Impact.ipynb

- "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."

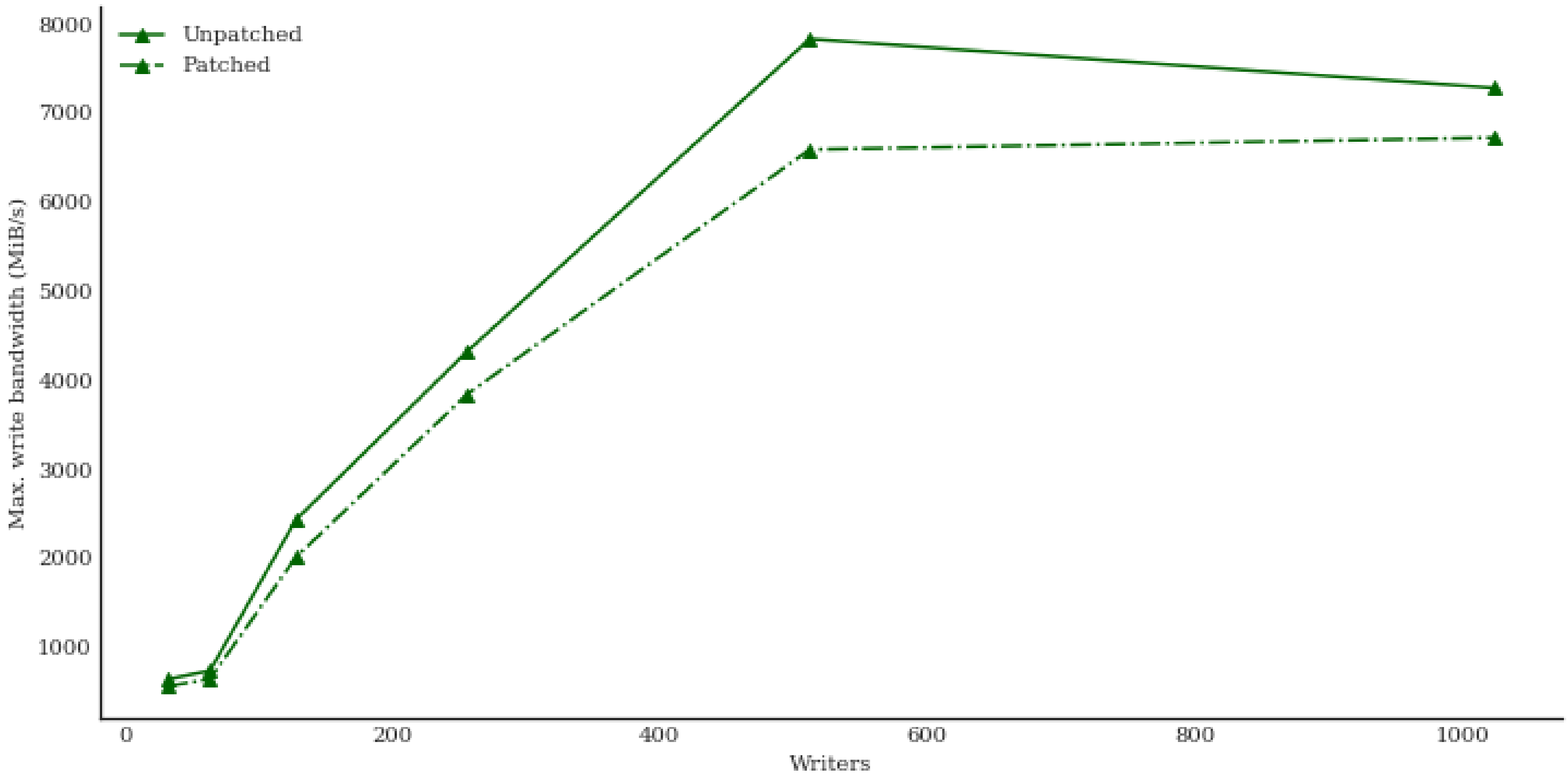
CASTEP Al Slab (al3x3)



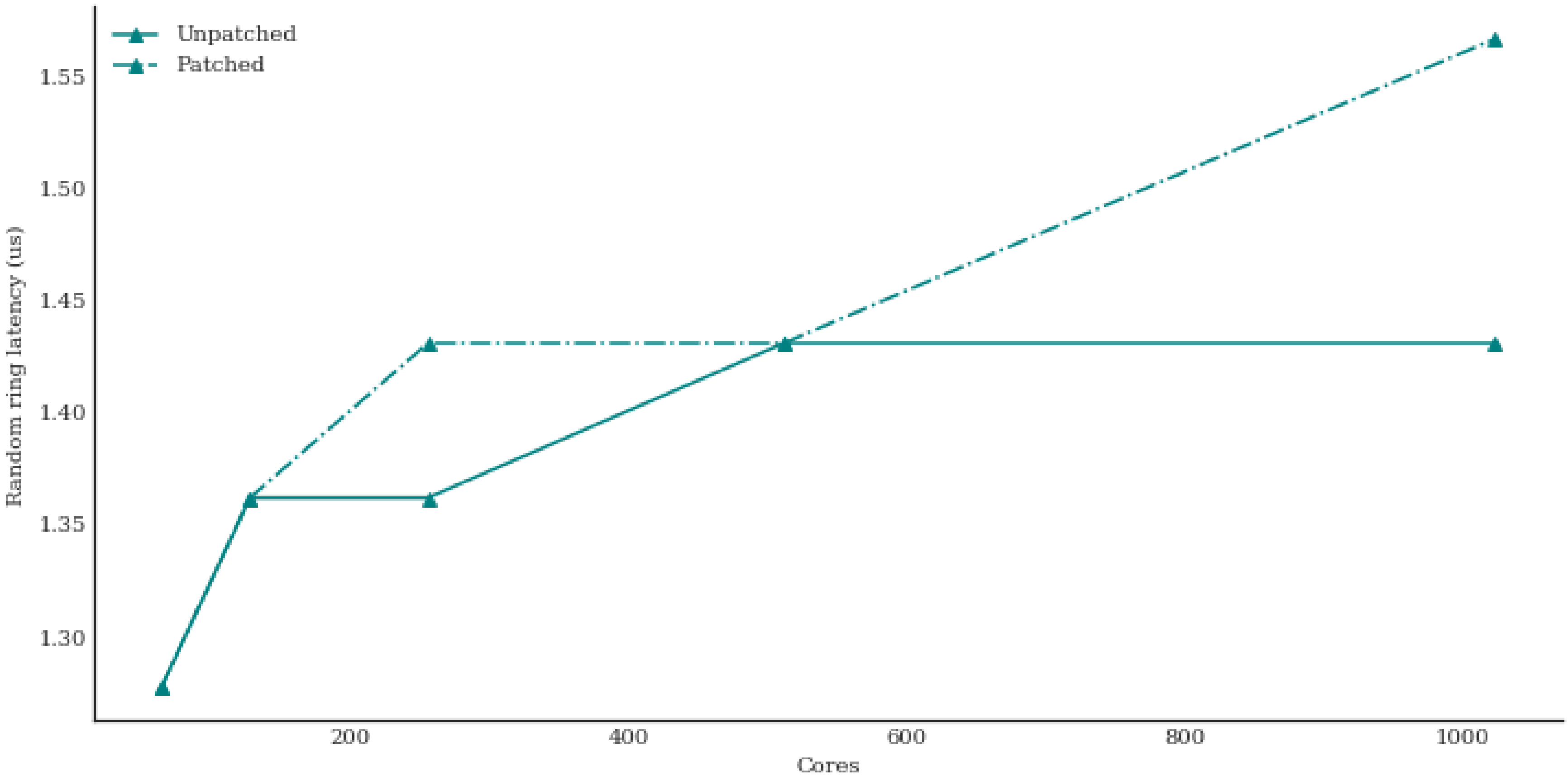
GROMACS


Unpatched
Unpatched
Performance (ns/day)
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
200
400
600
800
1000
Cores

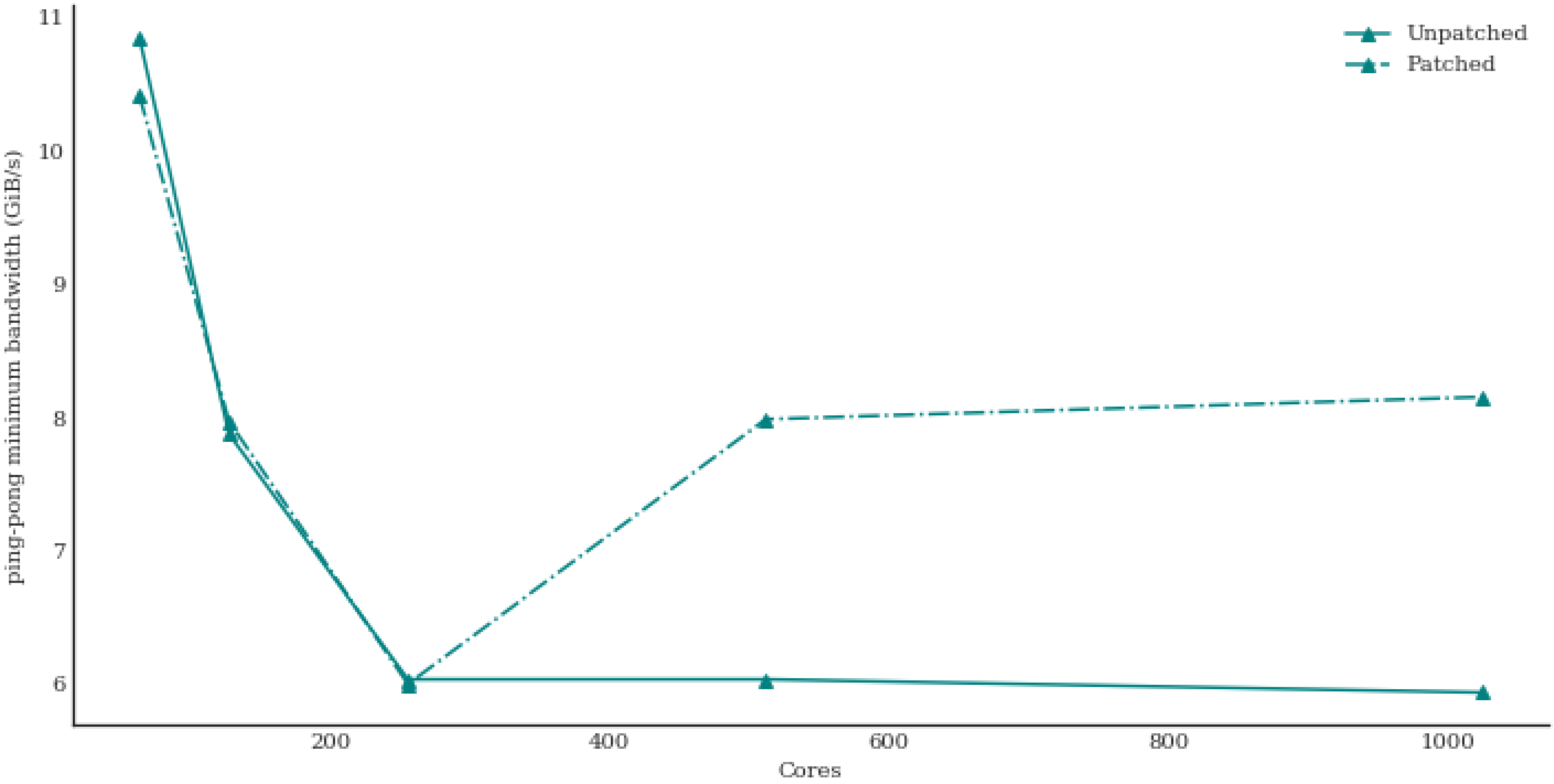
Benchio – parallel MPI-IO to Lustre



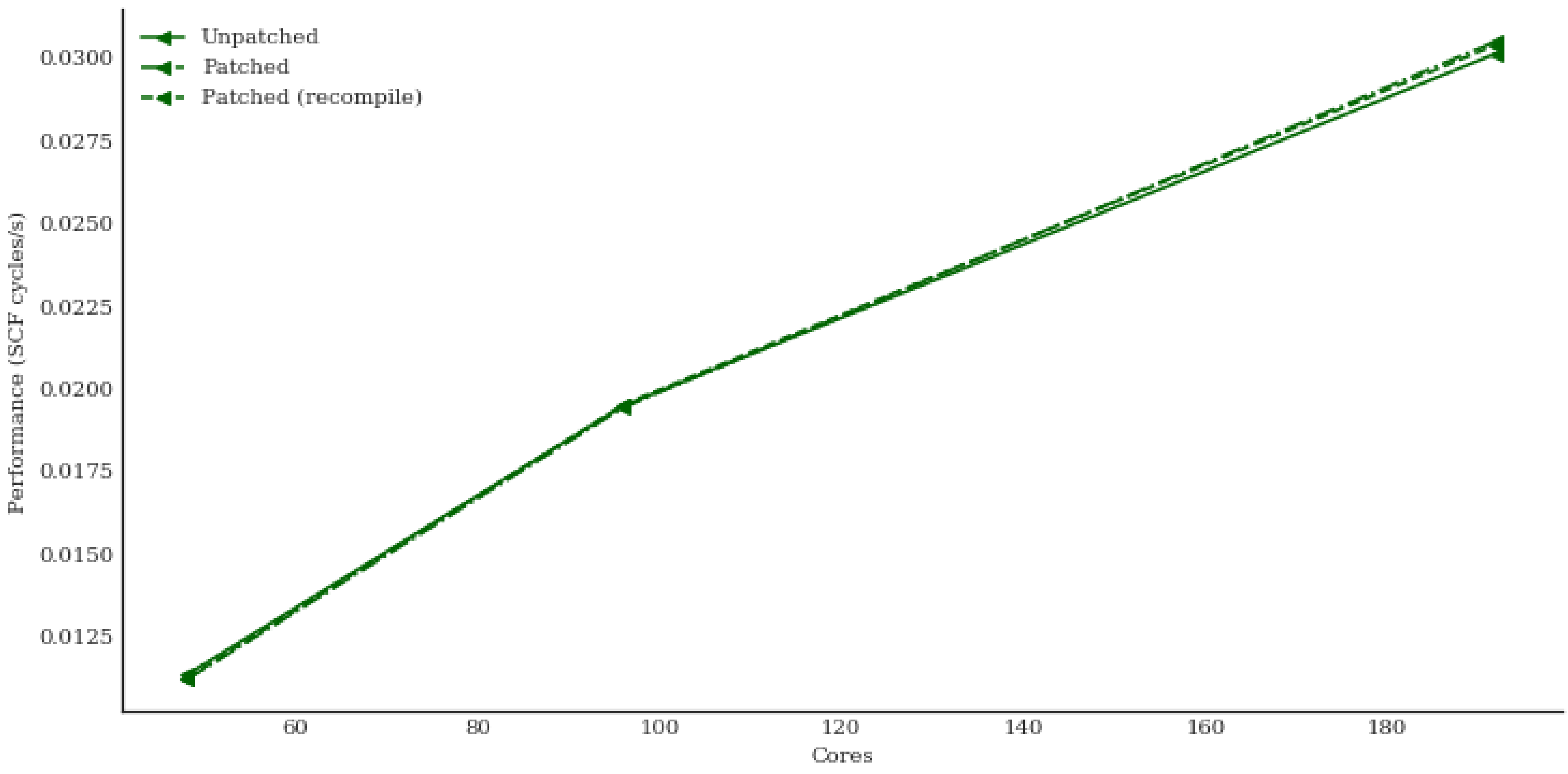
Random Ring Latency



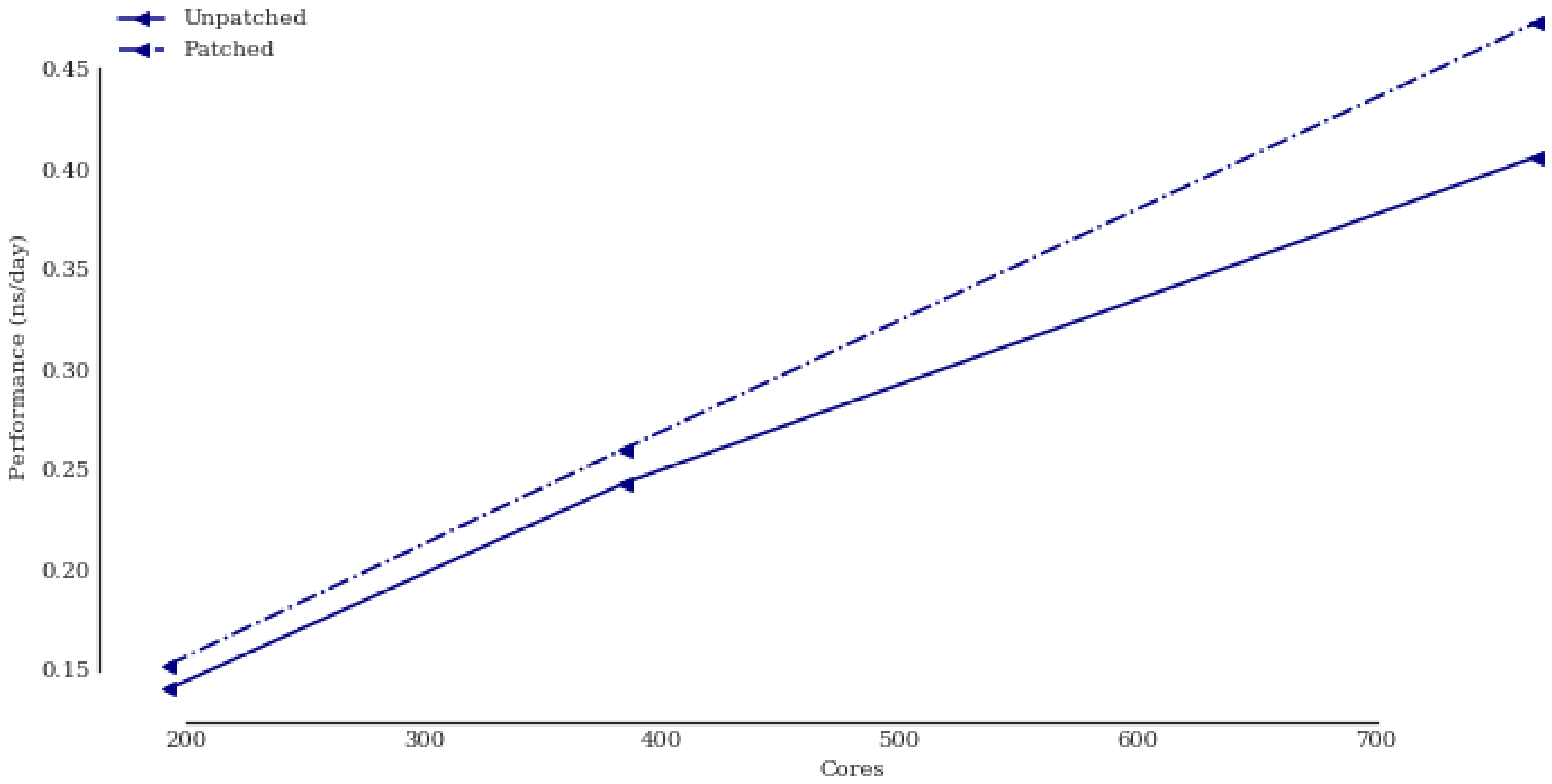
Minimum Ping-Pong Bandwidth



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