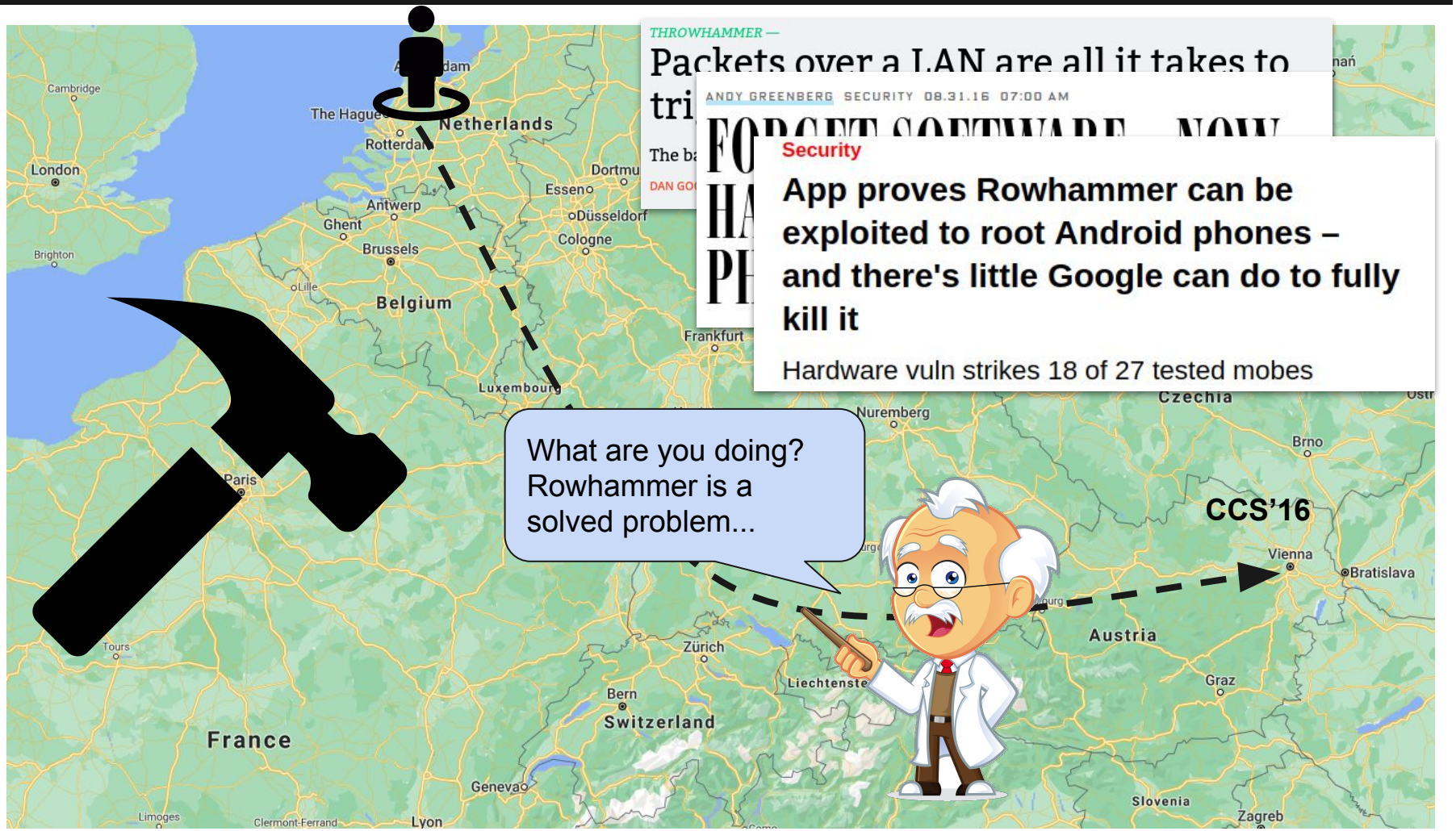
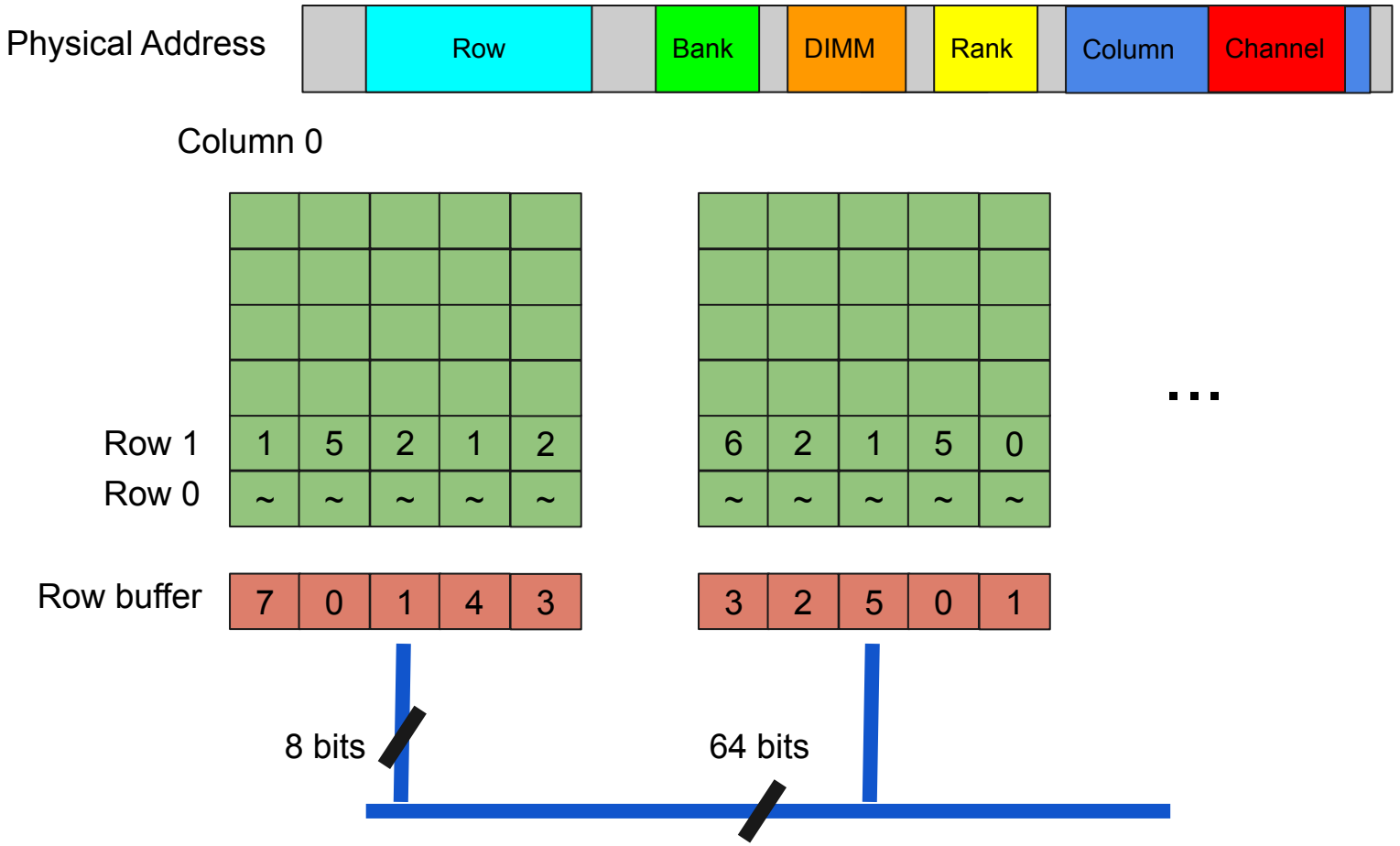
Assignment 1: DRAMA

Hardware Security

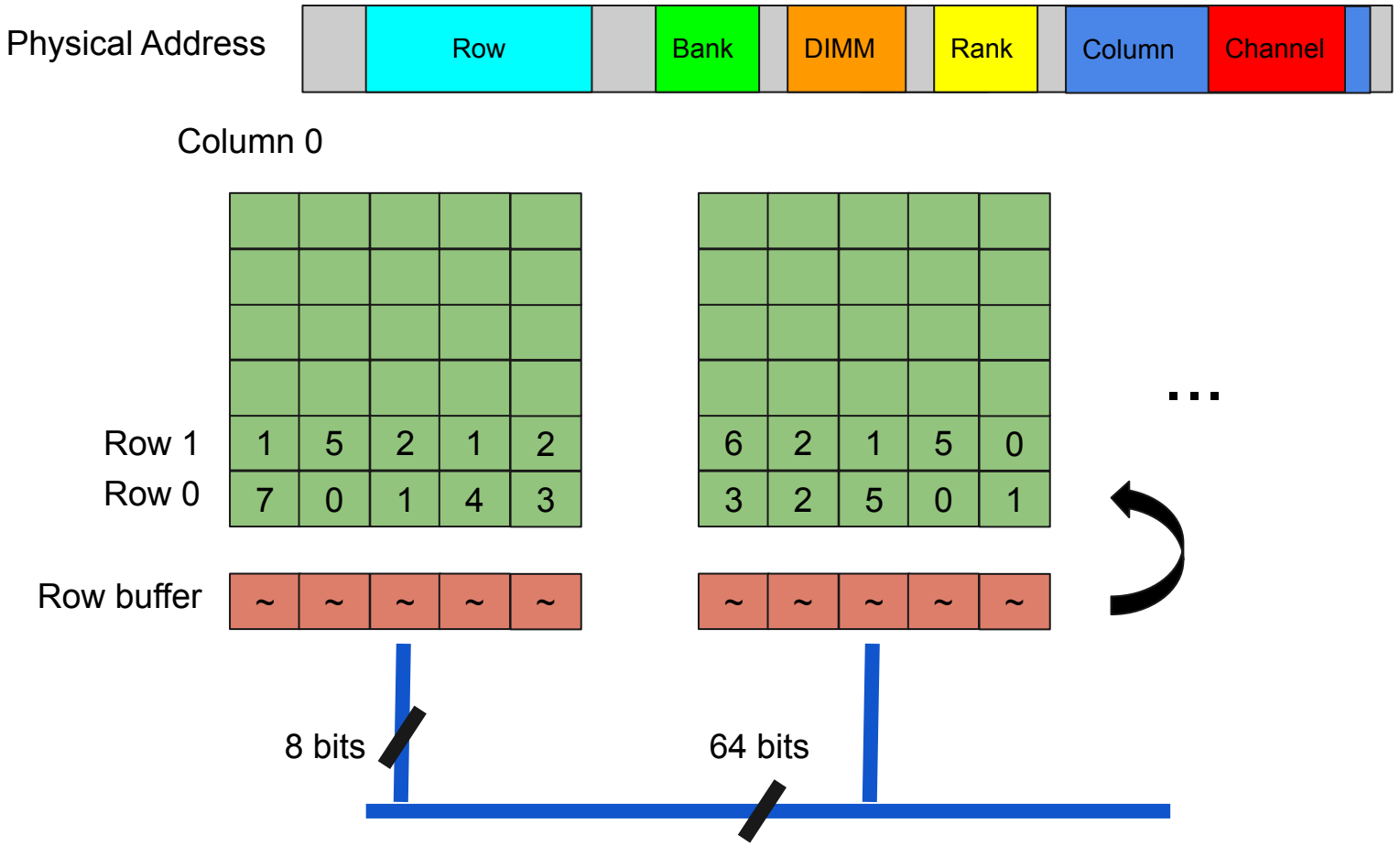
Why?


THROWHAMMER —
Packets over a LAN are all it takes to
ANDY GREENBERG SECURITY 08.31.16 07:00 AM
Security
App proves Rowhammer can be exploited to root Android phones – and there's little Google can do to fully kill it
Hardware vuln strikes 18 of 27 tested mobes
What are you doing? Rowhammer is a solved problem...
CCS'16
Netherlands
Belgium
France
Switzerland
Austria
Czechia
Slovenia
London
Paris
Frankfurt
Cologne
Nuremberg
Vienna
Bratislava
Zürich
Geneva
Lyon
Brussels
Antwerp
Ghent
Rotterdam
Luxembourg
Brno
Graz
Zagreb

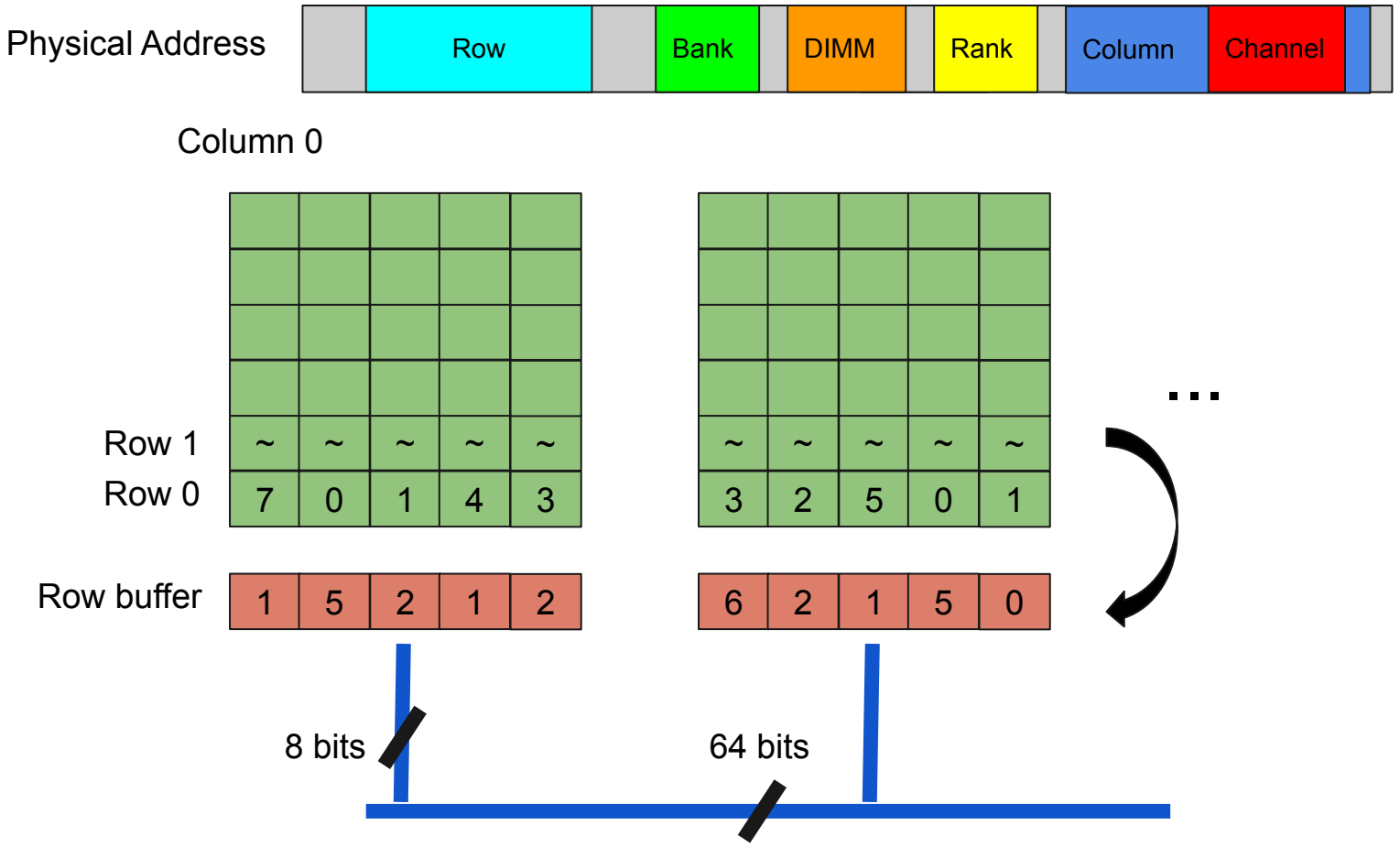
Recap


Physical Address
Row
Bank
DIMM
Rank
Column
Channel
Column 0
Row 1
Row 0
1 5 2 1 2
~ ~ ~ ~ ~
6 2 1 5 0
~ ~ ~ ~ ~
...
Row buffer
7 0 1 4 3
3 2 5 0 1
8 bits
64 bits

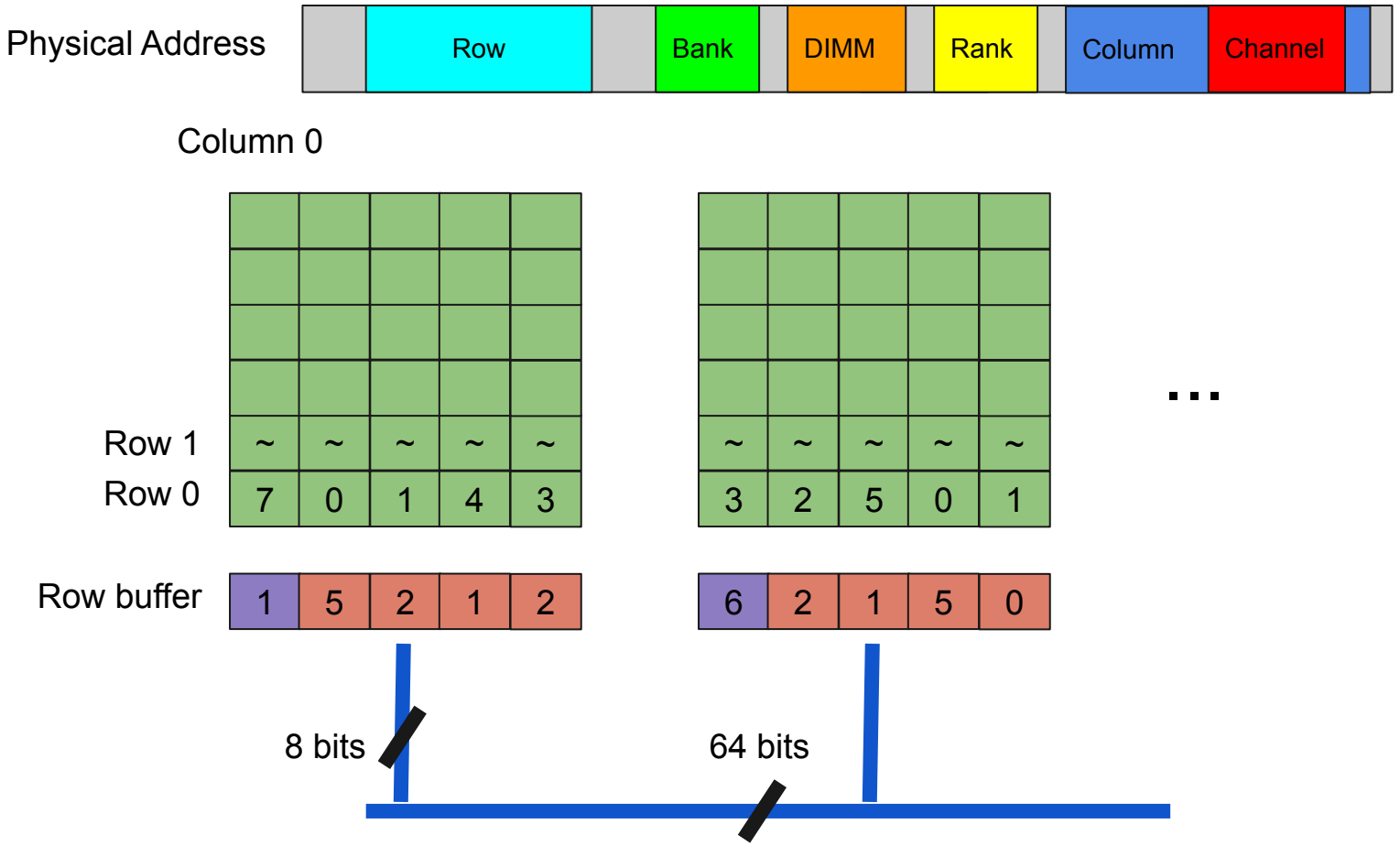
Recap


Physical Address
Row
Bank
DIMM
Rank
Column
Channel
Column 0
Row 1
Row 0
1 5 2 1 2
7 0 1 4 3
6 2 1 5 0
3 2 5 0 1
...
Row buffer
~ ~ ~ ~ ~
~ ~ ~ ~ ~
8 bits
64 bits

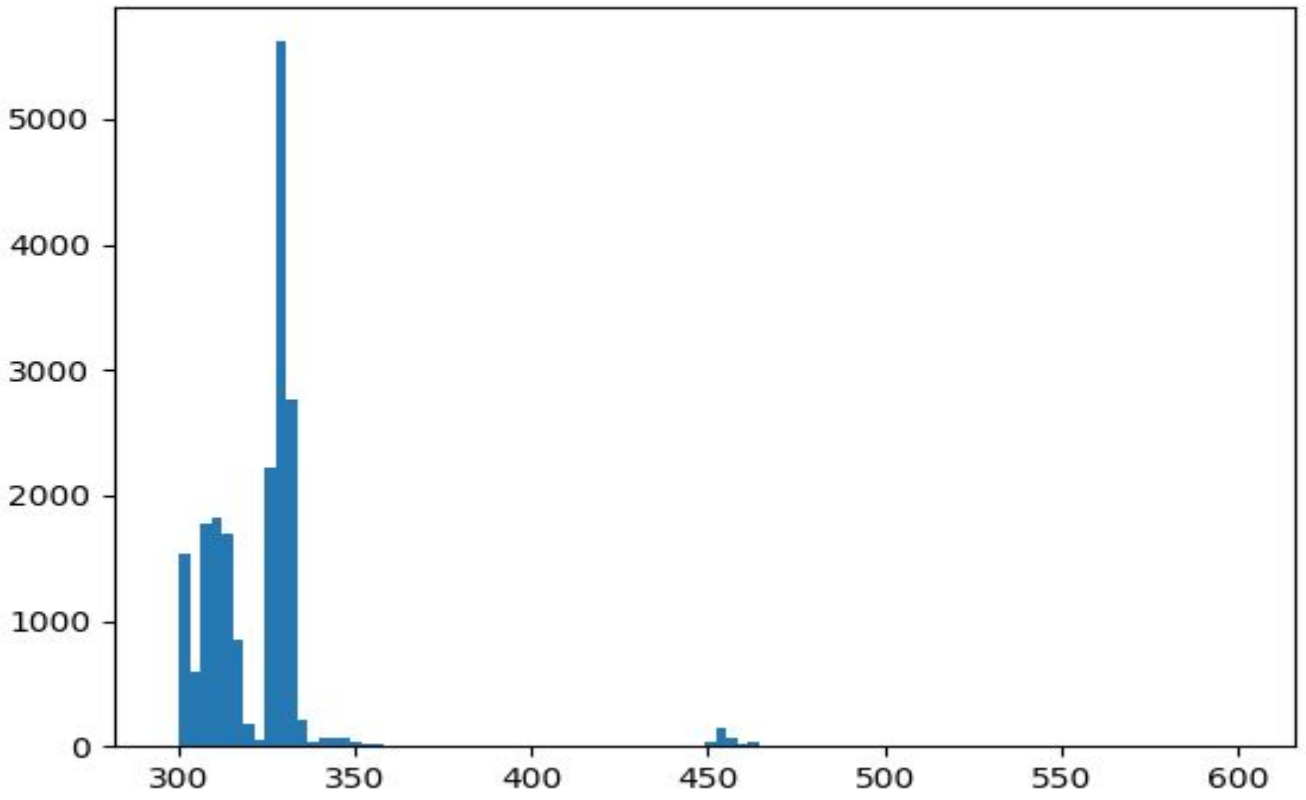
Recap


Physical Address
Row
Bank
DIMM
Rank
Column
Channel
Column 0
Row 1
Row 0
~ ~ ~ ~ ~
7 0 1 4 3
~ ~ ~ ~ ~
3 2 5 0 1
...
Row buffer
1 5 2 1 2
6 2 1 5 0
8 bits
64 bits

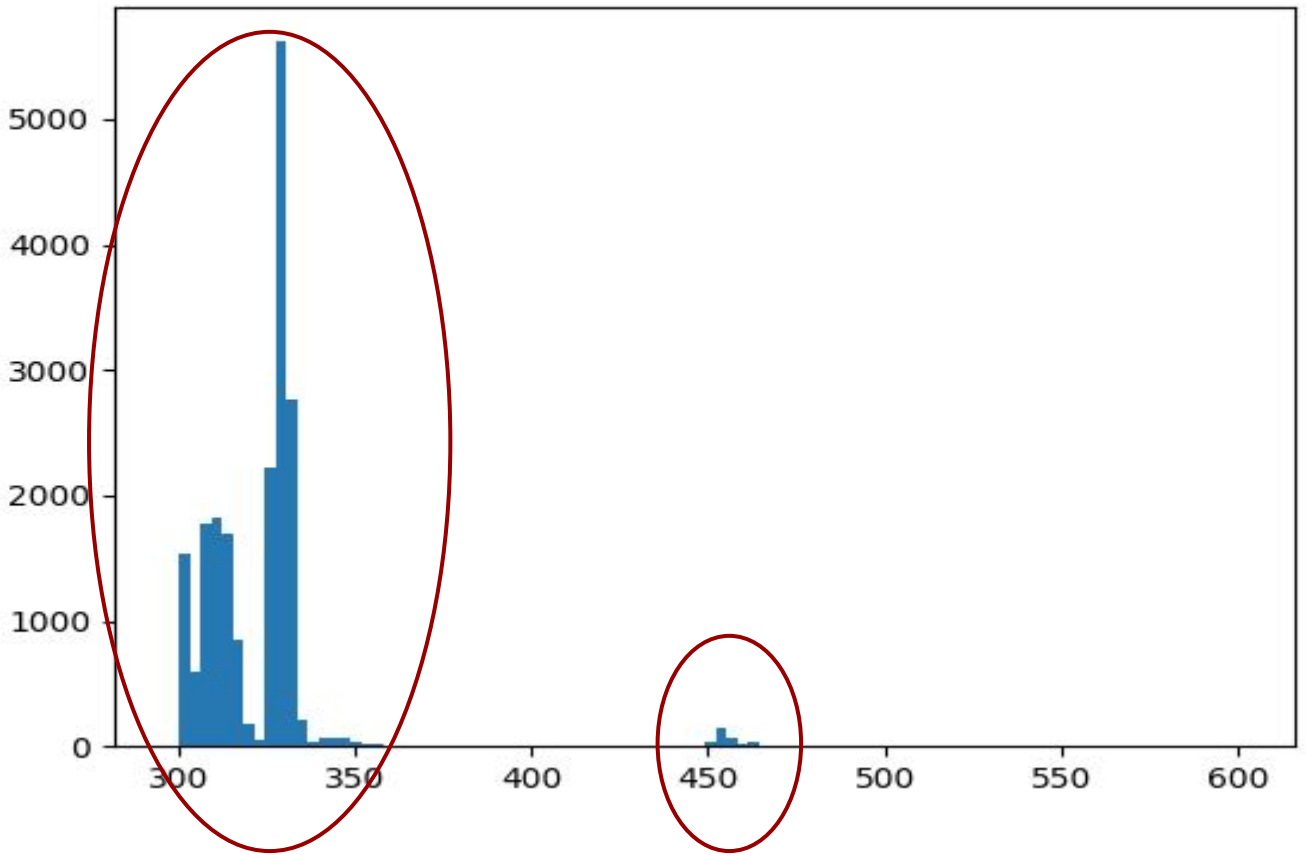
Recap


Physical Address
Row
Bank
DIMM
Rank
Column
Channel
Column 0
Row 1
Row 0
~ ~ ~ ~ ~
7 0 1 4 3
~ ~ ~ ~ ~
3 2 5 0 1
...
Row buffer
1 5 2 1 2
6 2 1 5 0
8 bits
64 bits

Recap



Recap



Recap


usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION
DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks
Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard,
Graz University of Technology
https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/pessl
This paper is included in the Proceedings of the
25th USENIX Security Symposium
August 10–12, 2016 • Austin, TX
ISBN 978-1-931971-32-4
Open access to the Proceedings of the
25th USENIX Security Symposium
is sponsored by USENIX
ware-only Reverse Engineering of Physical
AM Mappings for Rowhammer Attacks
Luca Breveglieri
Politecnico di Milano - DEIB
Piazza Leonardo da Vinci, 32
20133, Milan, Italy
luca.breveglieri@polimi.it
Niccolò Izzo
Politecnico di Milano - DEIB
Piazza Leonardo da Vinci, 32
20133, Milan, Italy
niccolo.izzo@mail.polimi.it
Gerardo Pelosi
Politecnico di Milano - DEIB
Piazza Leonardo da Vinci, 32
20133, Milan, Italy
gerardo.pelosi@polimi.it
implementation on silicon in 1966 by Robert Dennard, DRAM
modules have been increasing their capacity and read/write
speed as the result of a steady shrinking in the etching
technology. One key characteristic of DRAM memories is
that data memorized in them fade over time naturally, as they
are represented by charge levels in non-ideal capacitors. Thus
every DRAM must be periodically read and rewritten, an op-
eration known as refresh, to avoid data corruption. Despite the
fact that a periodic refresh is performed, electrical phenomena
such as sub-threshold leakage and gate-induced drain leakage,
may still cause data alteration. Memory manufacturers have
long known these phenomena, and always considered them
as reliability issues, introducing mechanisms relying on error
correcting codes, to allow the correction of a single-bit error
and the detection of a two-bit error in the module.
While accidental memory corruptions due to fading are
chiefly a reliability concern, the possibility of inducing such
value changes intentionally, via software-only stimuli has
proven to be a significant security problem. Indeed, Kim
Yoongu et al. [5] identified and practically validated the
possibility of circumventing memory protection mechanisms
exploiting software-induced bit-flips in DRAM modules to
change the protection map of user-accessible memory pages.
This attack, known as Rowhammer, relies on repeatedly per-
forming read accesses to a row of a DRAM block, causing
charge depletion in the neighboring rows. The charge depletion
results in flip downs in the stored values, which escape
the common memory protection mechanisms enacted by the
operating system and the CPU. Since its first description, the
ility of Rowhammer to circumvent the access control bar-
between different process domains has been extensively
tigated to highlight and exploit security issues on various
s. Open literature reports attacks aimed at performing
lege escalation of the hosting operating system from
nvironments on desktop [6] and mobile [7] platforms
er sandboxes [8] and virtual machine environments
third-party compute clouds [9]. In addition to priv-
ion, the faults induced in the stored data have also
thwart the security of cryptographic primitives
A strict requirement for a the effectiveness of a
ack, is the knowledge of the mapping between
esses and the actual data location within the
itecture. Such a mapping depends on the memory
19

Assignment

3 Tasks:

- #1 Detecting bank conflicts
  (Timing side channel)
- #2 Detecting number of banks
- #3 Automated threshold detection

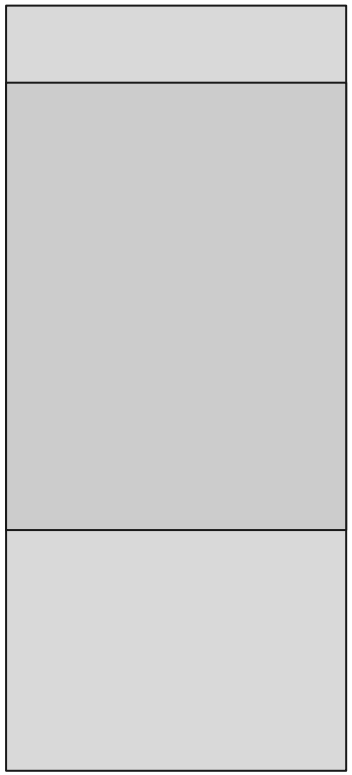
Task 1

Detecting bank conflicts

Task 1

Detecting bank conflicts

#1 HugePages

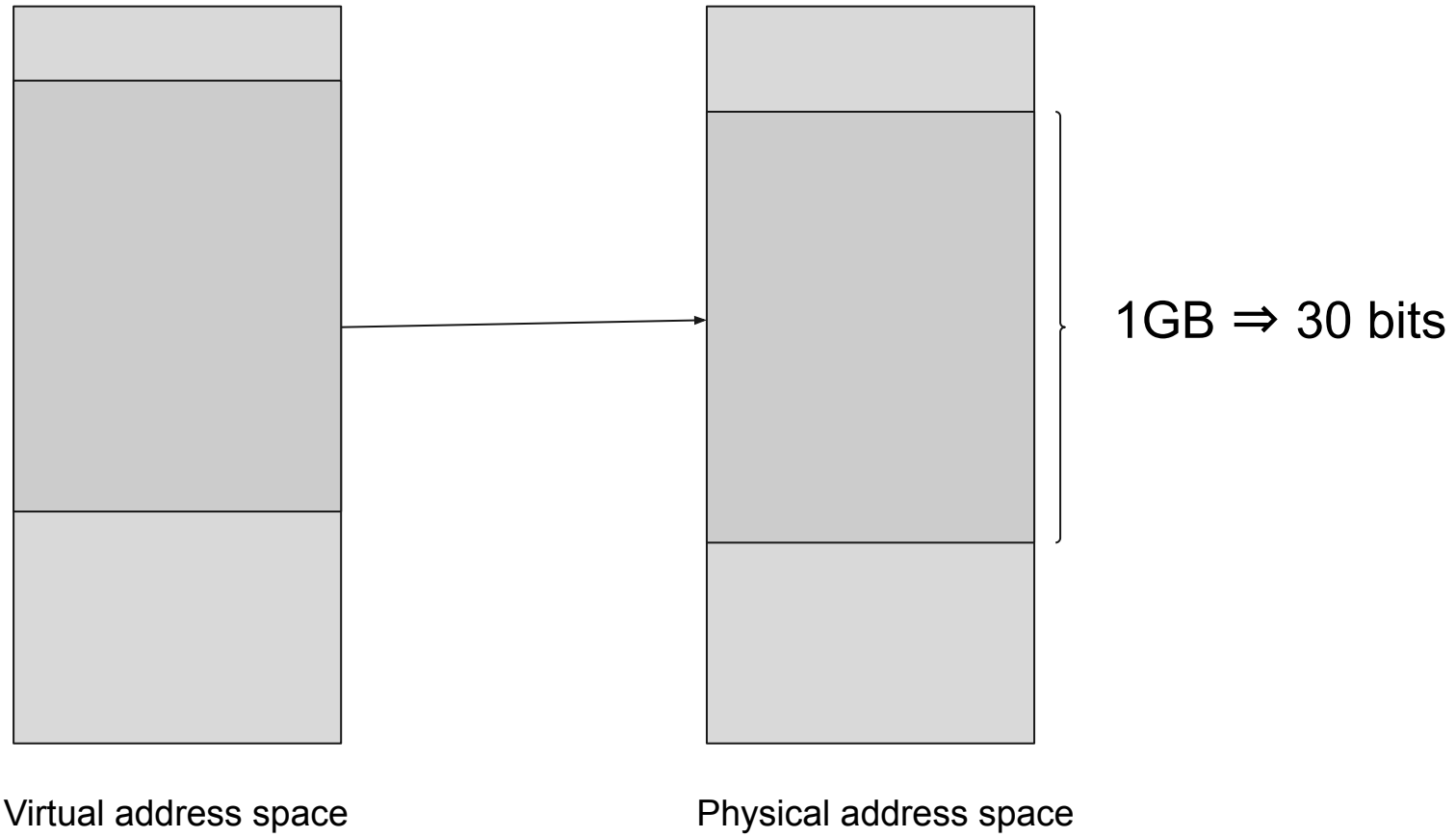


Virtual address space

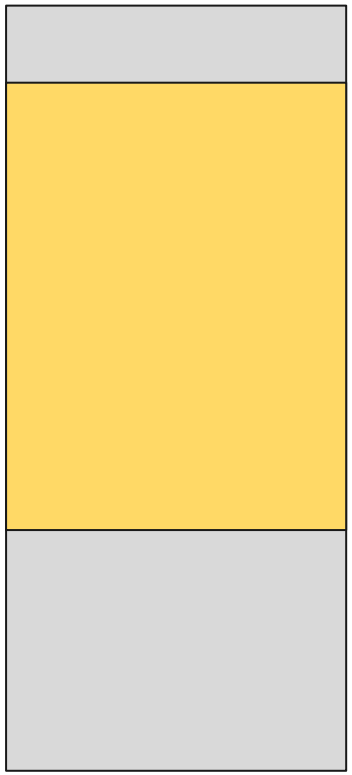
```
#define SUPERPAGE GB(1)


char* buffer =
mmap(1GB_aligned_addr,
     SUPERPAGE,
     PROT_READ|PROT_WRITE,
     MAP_SHARED|MAP_ANONYMOUS|MAP_HUGETLB,
     -1,
     0);
```

Note: we have configured the cluster so that this gives 1 superpage

#1 HugePages

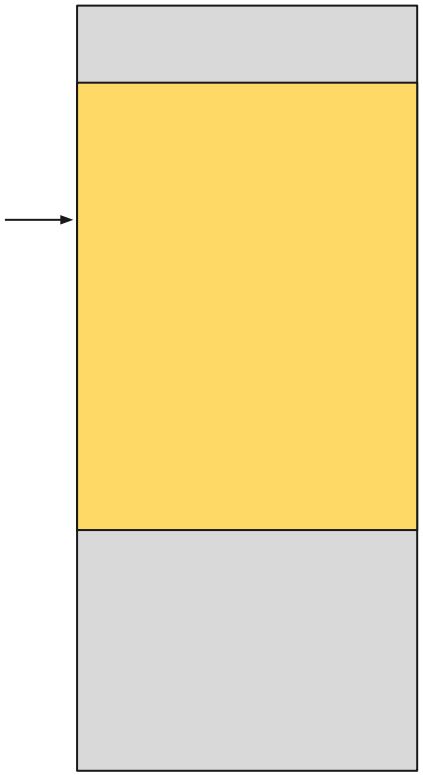


#1 Bank conflict side channel



Virtual address space

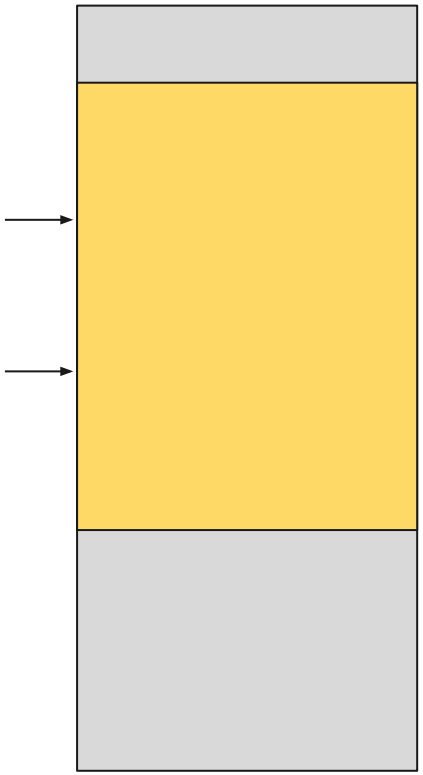
#1 Bank conflict side channel



Virtual address space

```
char** pool = [*addr1,
```

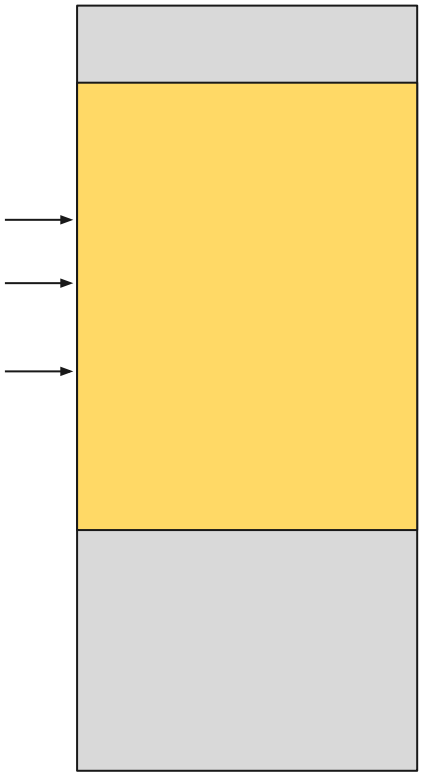
#1 Bank conflict side channel



Virtual address space

```
char** pool = [*addr1, *addr2,
```

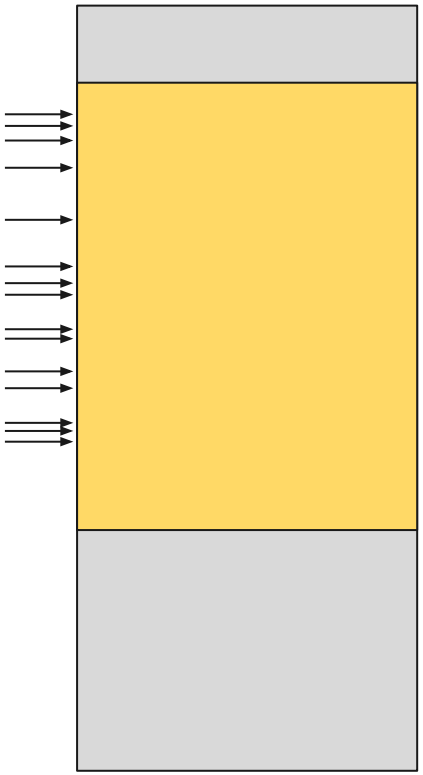
#1 Bank conflict side channel



Virtual address space

```
char** pool = [*addr1, *addr2, *addr3,
```

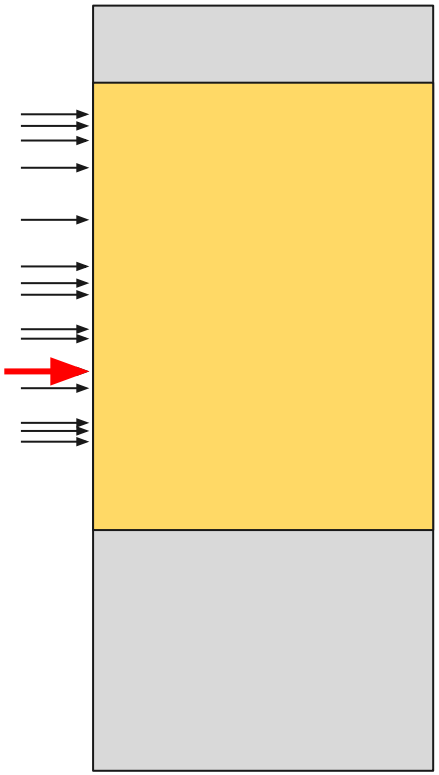
#1 Bank conflict side channel



Virtual address space

```
char** pool = [*addr1, *addr2, *addr3, ..., *addrN];
```

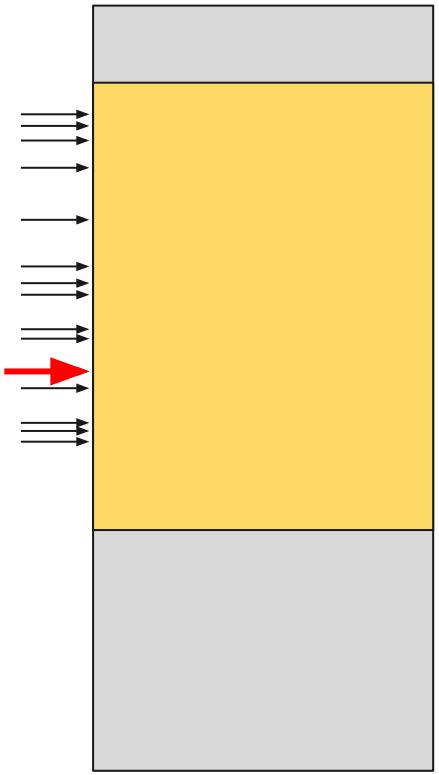
#1 Bank conflict side channel



Virtual address space

```
char** pool = [*addr1, *addr2, *addr3, ..., *addrN];
char* base = pool[rand()%pool_size];
```

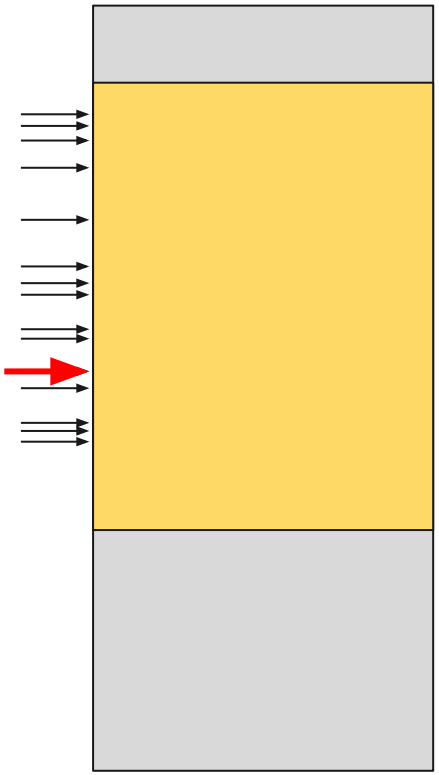
#1 Bank conflict side channel



Virtual address space

```
char** pool = [*addr1, *addr2, *addr3, ..., *addrN];
char* base = pool[rand()%pool_size];
for (addr in pool)
    time_access(base, addr);
```

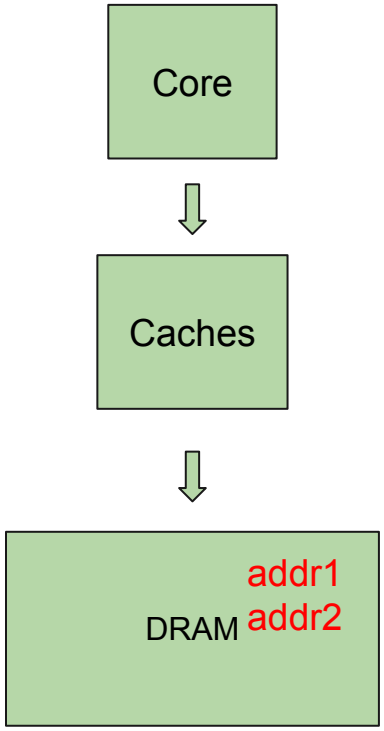
#1 Bank conflict side channel



Virtual address space

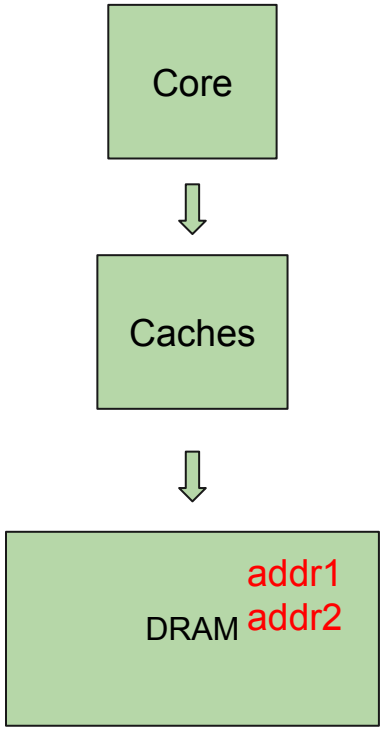
```
char** pool = [*addr1, *addr2, *addr3, ..., *addrN];
char* base = pool[rand()%pool_size];
for (addr in pool)
    time_access(base, addr);

int time_access(char* a, char* b) {
    t0 = rdtscp();
    while (round--) { *addr1; *addr2; }
    t1 = rdtscp();
    return (t1-t0)/ROUNDS;
}
```

#1 Bank conflict side channel



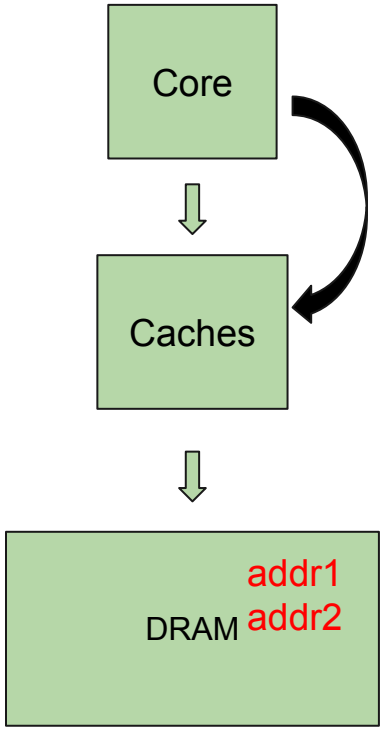

```
while (round--) {
    *addr1;
    *addr2;
}
```

#1 Bank conflict side channel



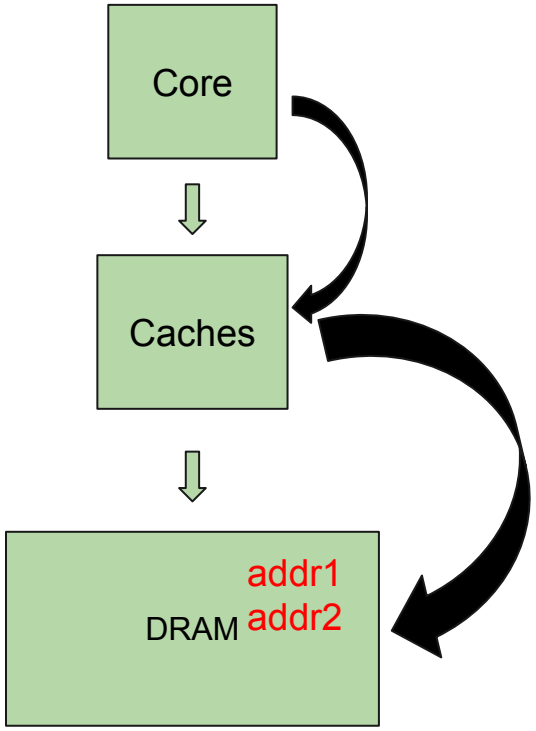

```
while (round--) {
    *addr1;
    *addr2;
}
```

#1 Bank conflict side channel



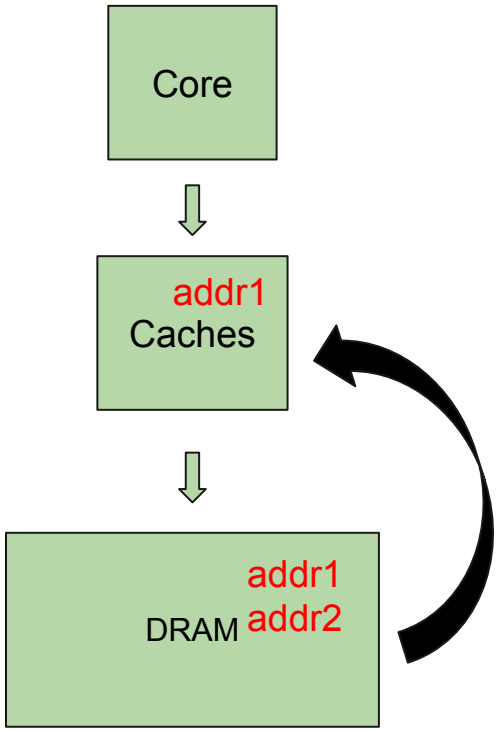

```
while (round--) {
    *addr1;
    *addr2;
}
```

#1 Bank conflict side channel



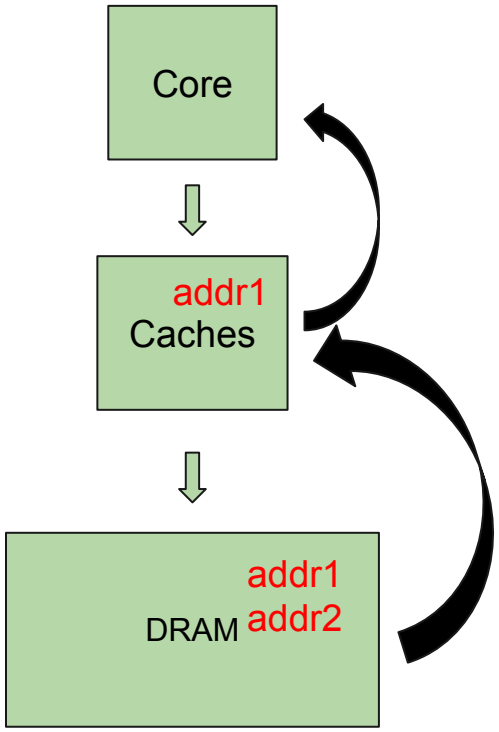

```
while (round--) {
    *addr1;
    *addr2;
}
```

#1 Bank conflict side channel



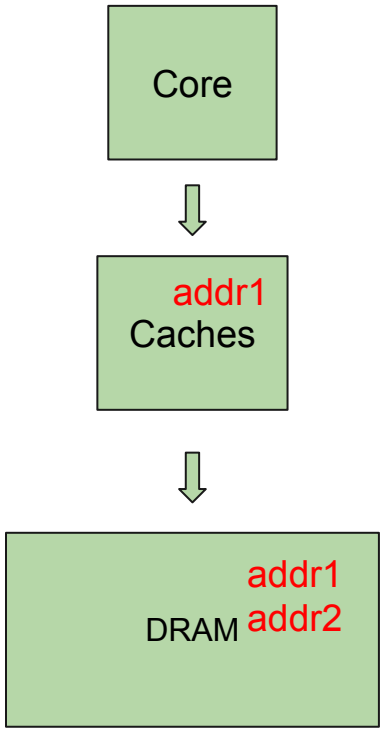

```
while (round--) {
    *addr1;
    *addr2;
}
```

#1 Bank conflict side channel



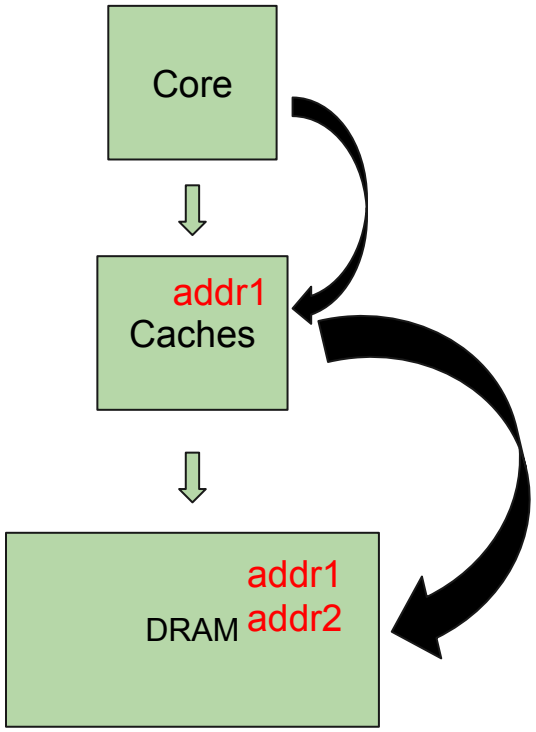

```
while (round--) {
    *addr1;
    *addr2;
}
```

#1 Bank conflict side channel



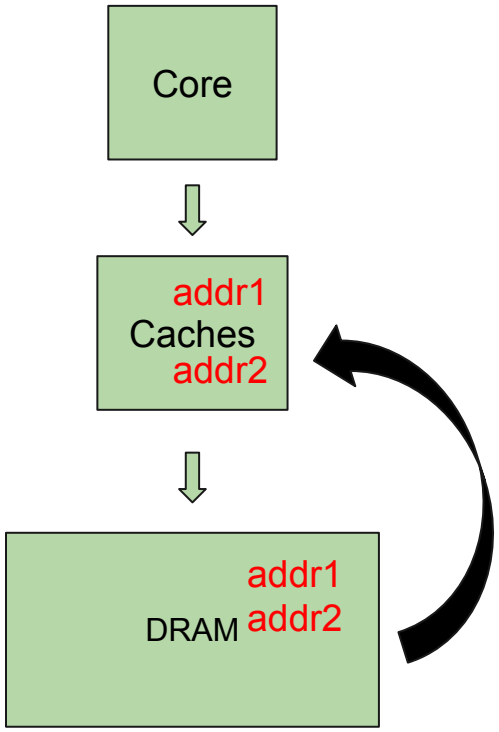

```
while (round--) {
    *addr1;
    *addr2;
}
```

#1 Bank conflict side channel



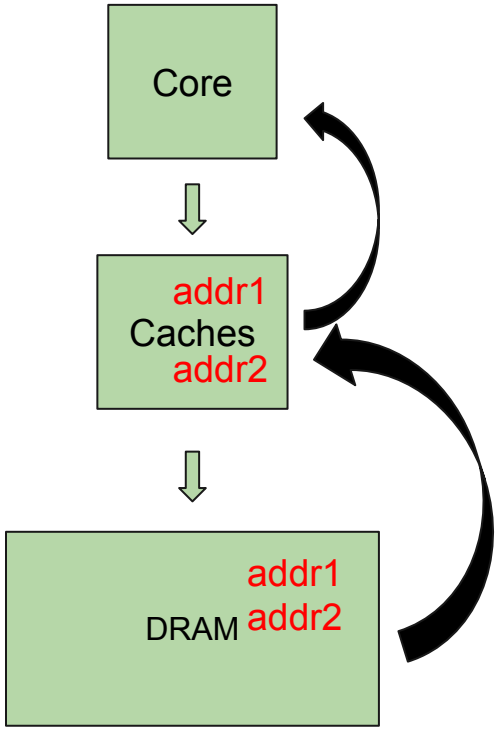

```
while (round--) {
    *addr1;
    *addr2;
}
```

#1 Bank conflict side channel



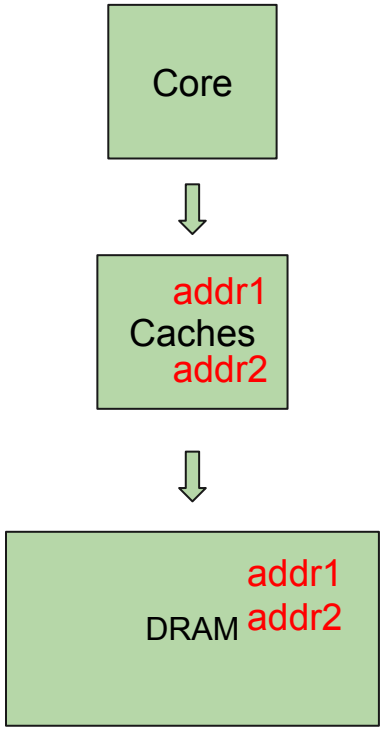

```
while (round--) {
    *addr1;
    *addr2;
}
```

#1 Bank conflict side channel



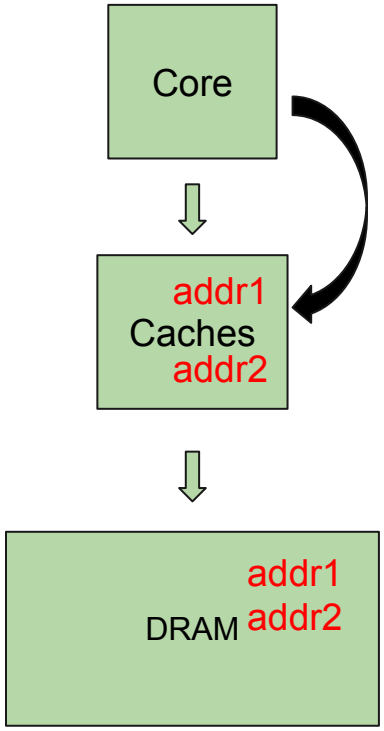

```
while (round--) {
    *addr1;
    *addr2;
}
```

#1 Bank conflict side channel



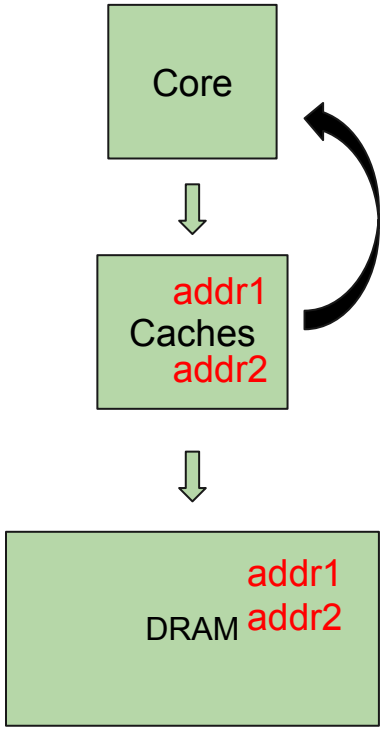

```
while (round--) {
    *addr1;
    *addr2;
}
```

#1 Bank conflict side channel



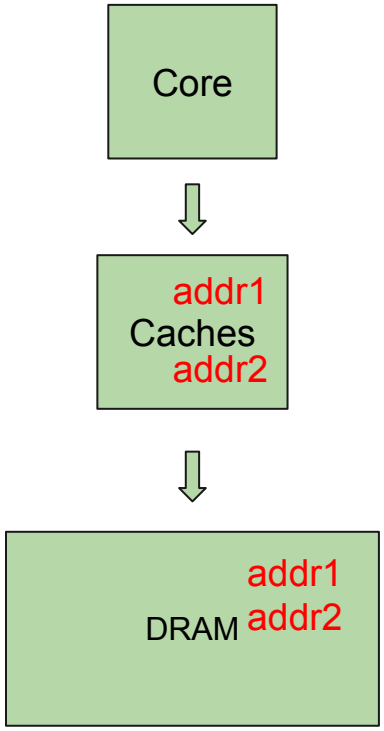

```
while (round--) {
    *addr1;
    *addr2;
}
```

#1 Bank conflict side channel



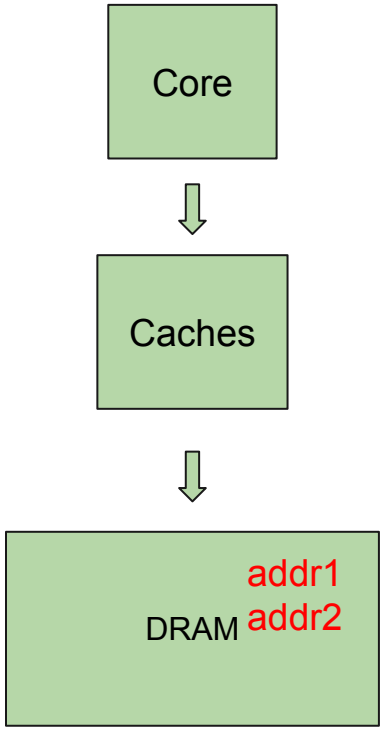

```
while (round--) {
    *addr1;
    *addr2;
}
```

#1 Bank conflict side channel



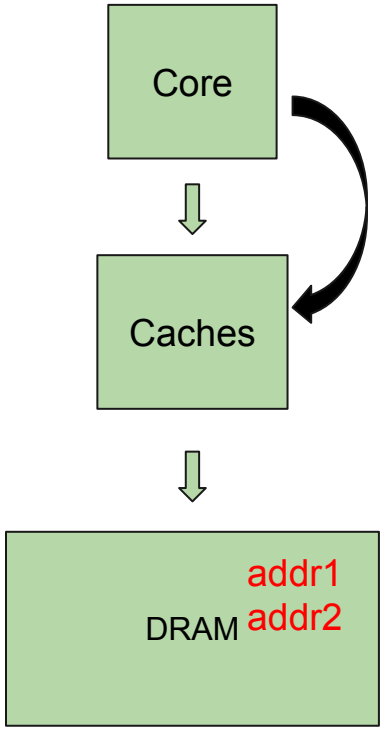

```
while (round--) {
    *addr1;
    *addr2;
    clflushopt(addr1);
    clflushopt(addr2);
}
```

#1 Bank conflict side channel



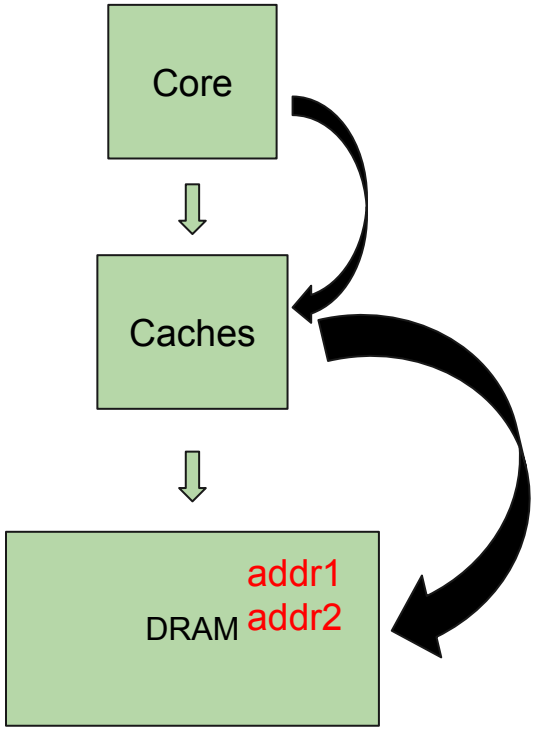

```
while (round--) {
    *addr1;
    *addr2;
    clflushopt(addr1);
    clflushopt(addr2);
}
```

#1 Bank conflict side channel



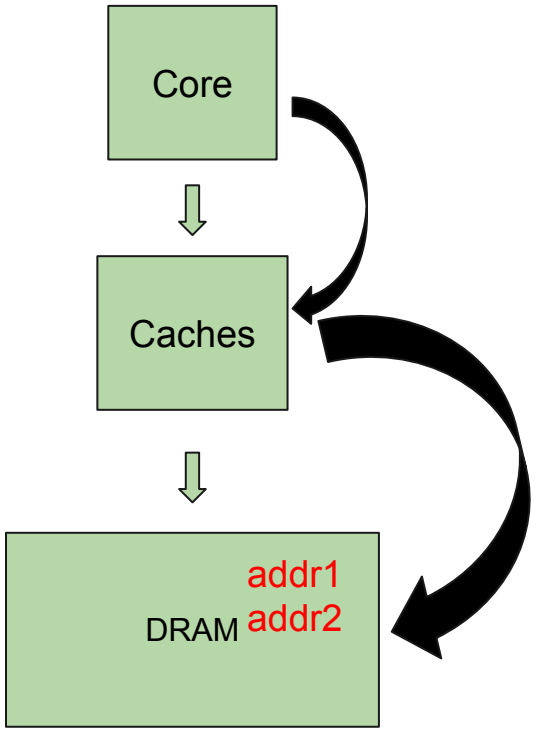

```
while (round--) {
    *addr1;
    *addr2;
    clflushopt(addr1);
    clflushopt(addr2);
}
```

#1 Bank conflict side channel




```
while (round--) {
    *addr1;
    *addr2;
    clflushopt(addr1);
    clflushopt(addr2);
}
```

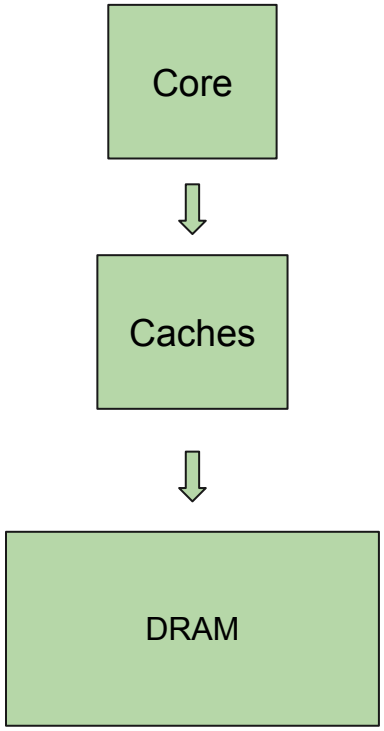
#1 Bank conflict side channel




```
while (round--) {
    *addr1;
    *addr2;
    clflushopt(addr1);
    clflushopt(addr2);
}
```

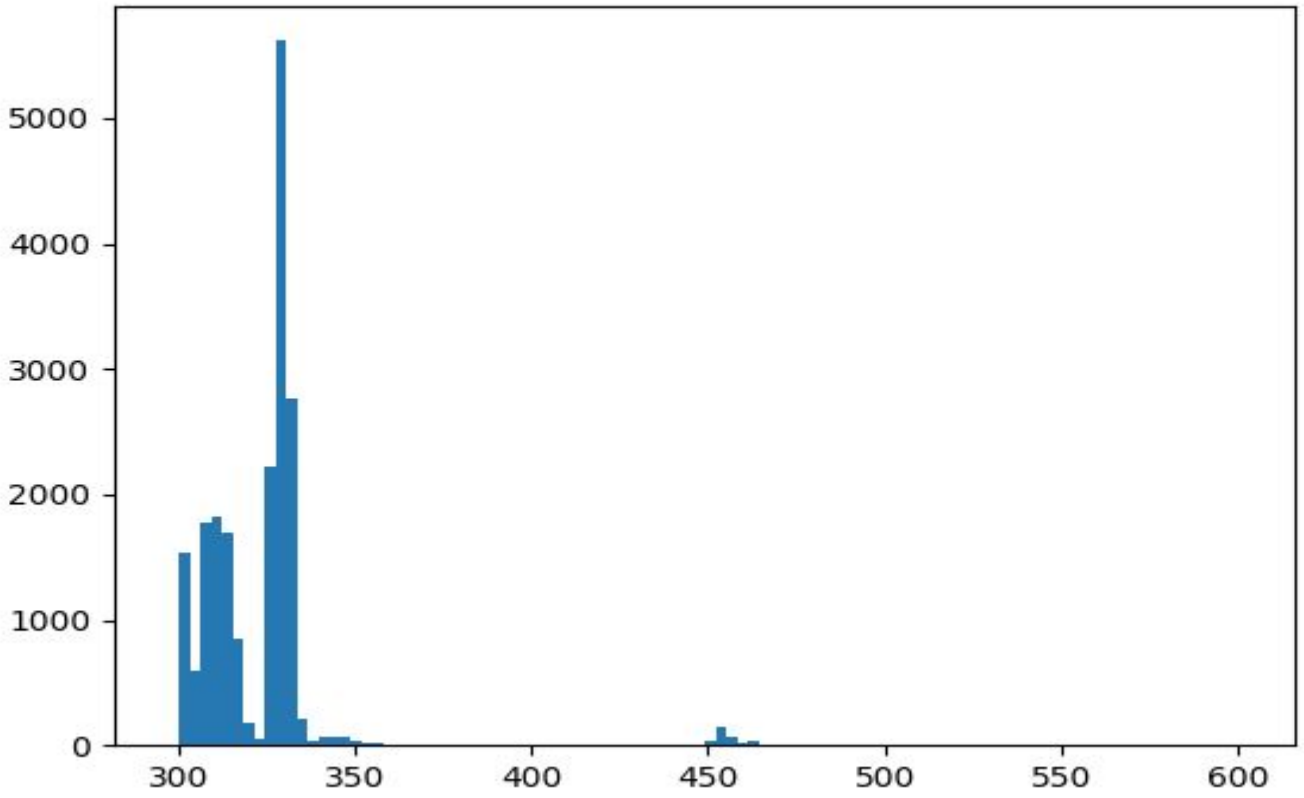
What about the order of the requests?

#1 Tips




```
while (--r) {
    t0 = rdtscp();
    *addr1; *addr2;
    times[r] = rdtscp() - t0;
    fence();

    clflushopt(addr1);
    clflushopt(addr2);
    fence();
}
time = median(times);
```

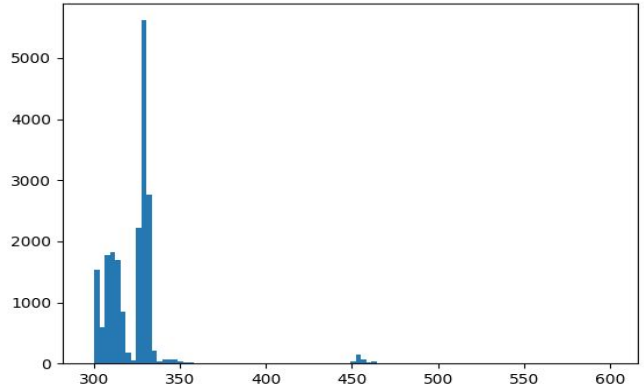
#1 Bank conflict side channel



#1 Bank conflict side channel

Deliverable:

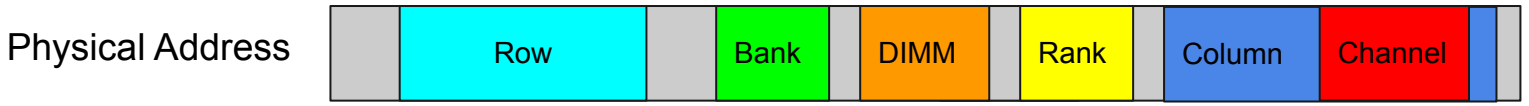
- 5 csv files with raw data in this format <*base, addr, time*>. Files named after five cluster nodes
- Your code that prints the contents of the csv file to stdout on a given node
- Plot showing conflicts on five nodes in the cluster



Task 2

Detecting the number of banks

#2 Number of banks

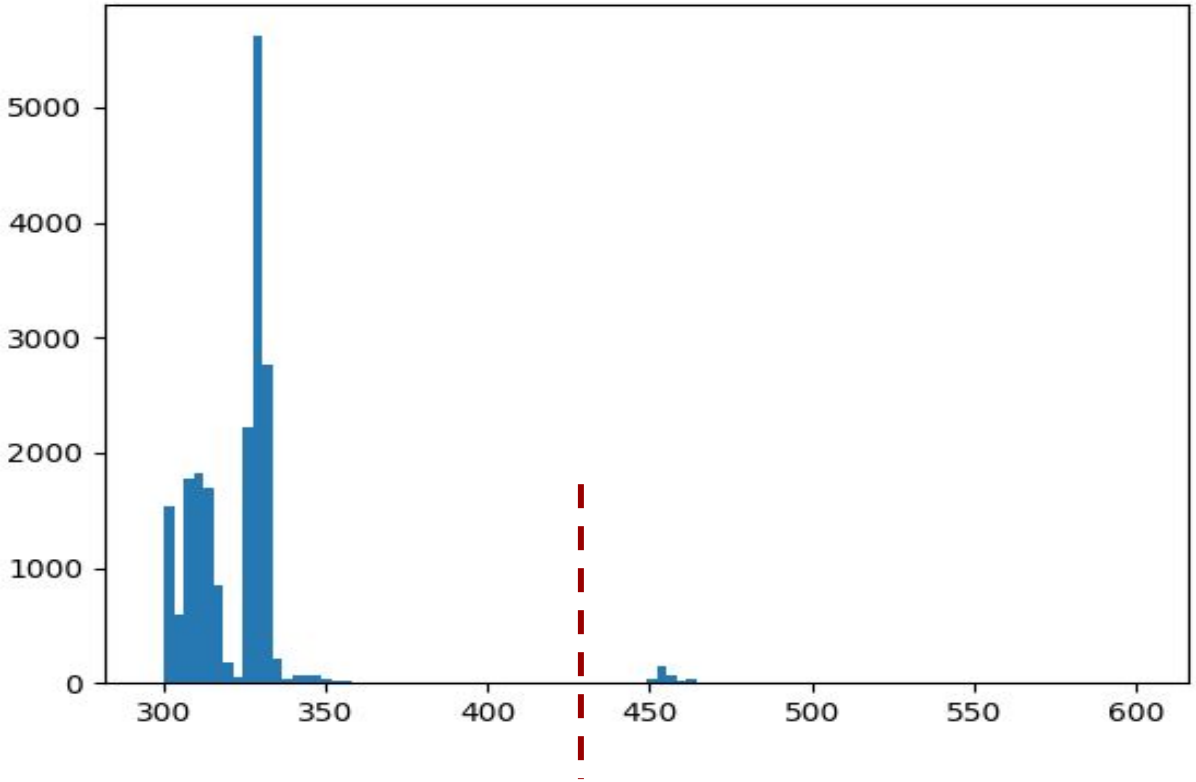


<channel, dimm, rank, bank>

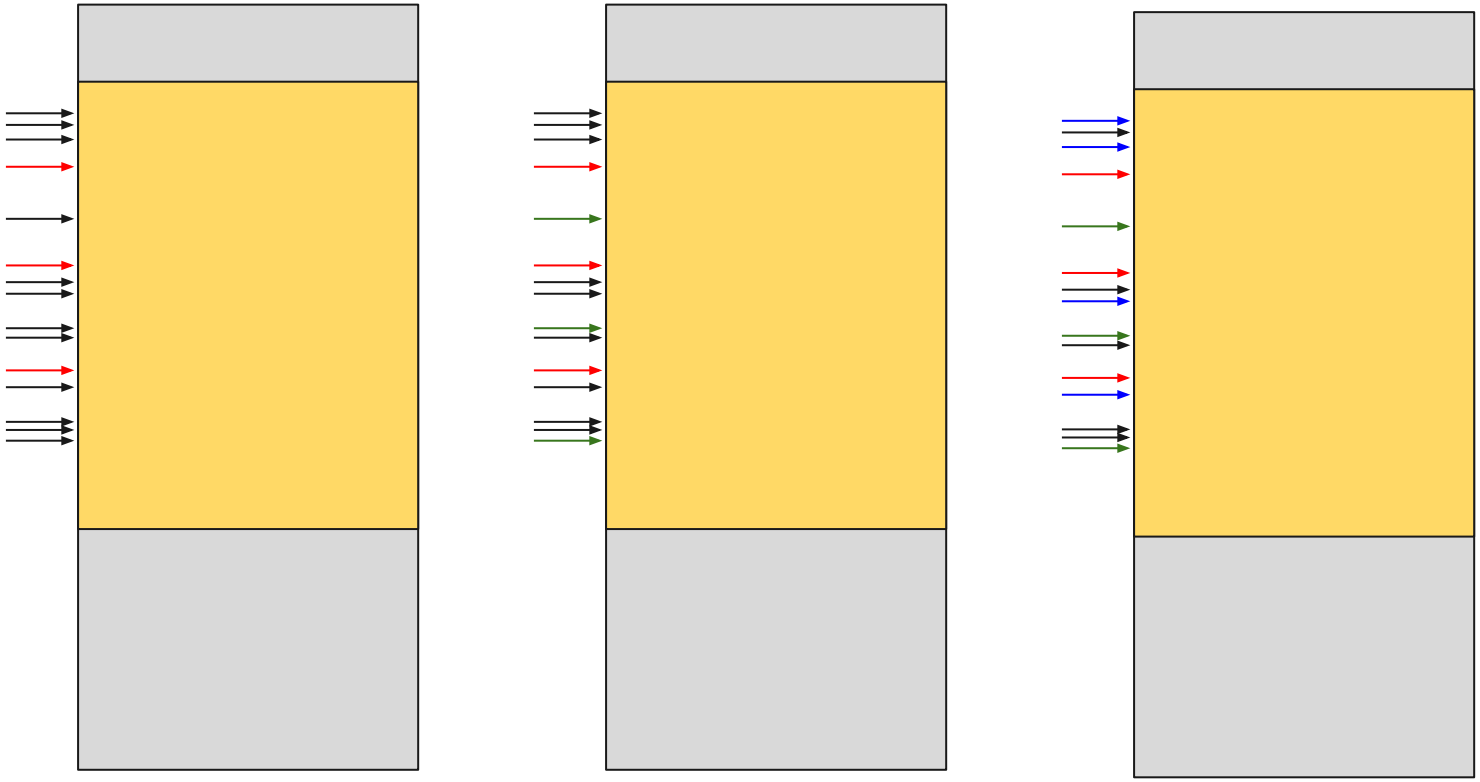
num_banks = channel x dimm x rank x banks_per_chip

On the testbed: dimm = 1, channel = 1

#2 Detecting the first bank



#2 Bank clustering

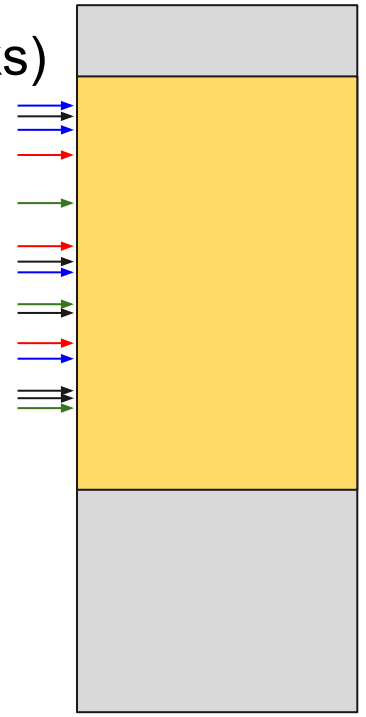


num_colors == num_banks

#2 Detecting the number of banks

Deliverable:

- Your code that performs a printf(“%d\n”, num_banks) at the end
- Number of banks on five nodes in the cluster
  node_name: num_banks

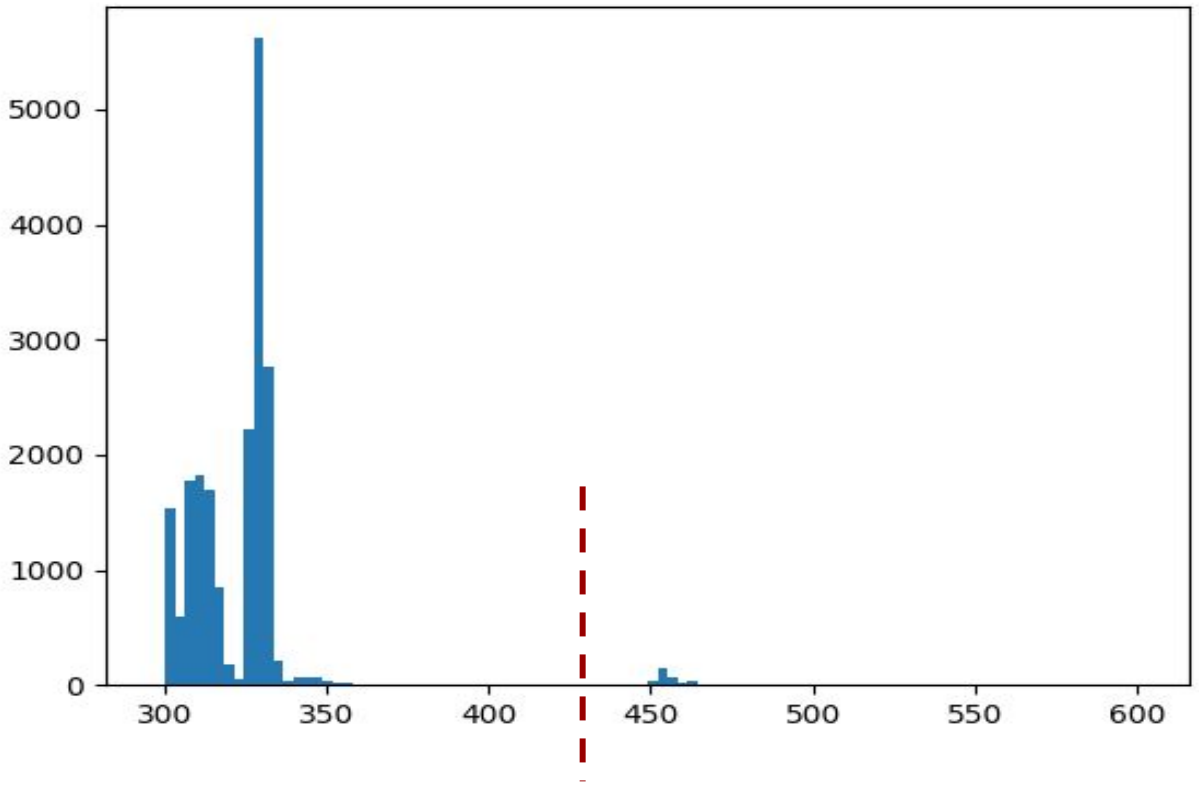


Task 3

Automated threshold detection

#3 Automated threshold detection

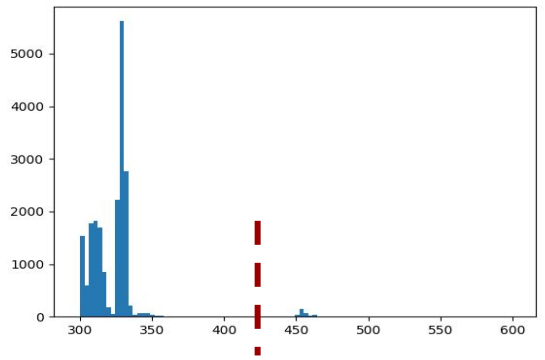
Open: do something that works!



#3 Automated threshold detection

Deliverable:

- Your code that performs a printf("%d %d\n", threshold, num_banks) at the end
- A 1-paragraph description of how your algorithm works
- The thresholds detected on five nodes in the cluster
  node_name: threshold (in cycles) num_banks



Notes on the Deliverable

- make should build ./{student_number}
- Task 1:
  - ./{student_number} should print the contents of the csv file to stdout on the current node
- Task 2:
  - ./{student_number} -t THRESHOLD should print the number of banks to stdout given a THRESHOLD
- Task 3:
  - ./{student_number} -b should print the calculated threshold, and the number of banks to stdout using the calculated threshold
- Each should finish execution < 60s

Grading & Deadline

- Deadline:
  - Deadline **Tuesday Sep 22, 23:59**
    Delays: -0.5pt per late day, max 3 late days.
- Grading:

4 ⇒ Task #1

5 ⇒ Task #2

6 ⇒ Task #3

Questions?

- Forum on Moodle
  - Help each other
  - Don’t give away your solution