Assignment 2: DRAM Functions

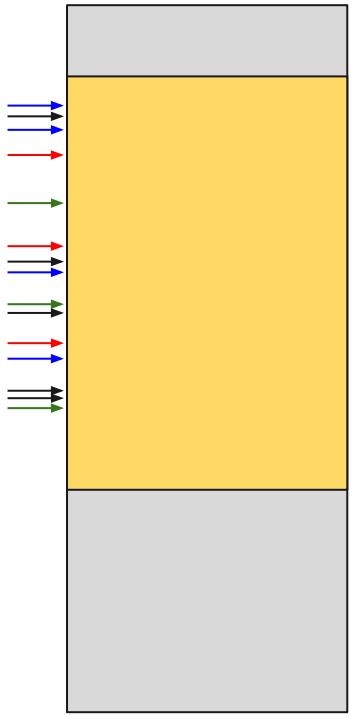
Hardware Security

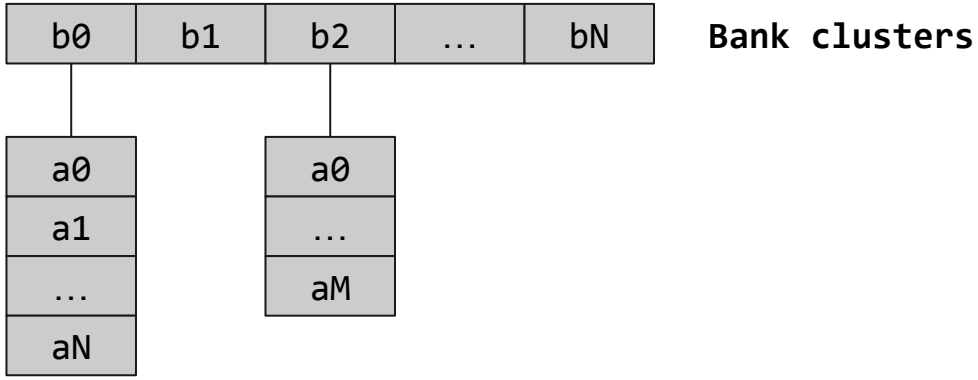
Assignment

3 Tasks:

- #1 Recover DRAM addressing bits
- #2 Recover DRAM bank addressing functions
- #3 Recover DRAM row selection mask

Where We Are...

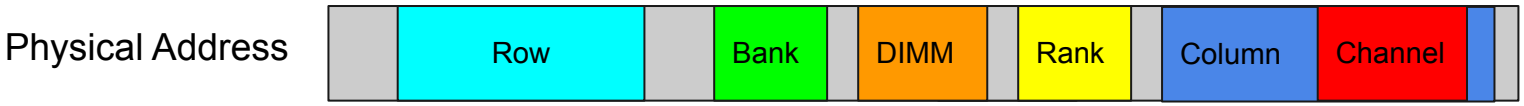




Task 1

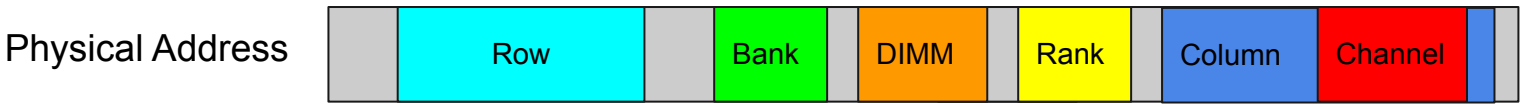
Bit recovery

#1 Bits Recovery



<channel, dimm, rank, bank>

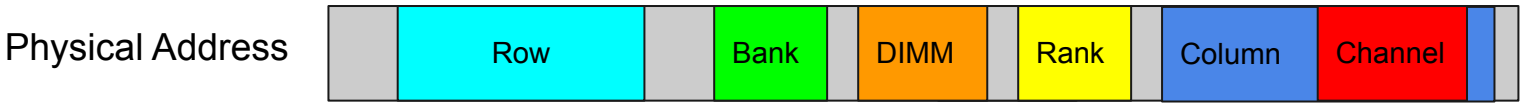
#1 Bits Recovery



`<channel, dimm, rank, bank>`

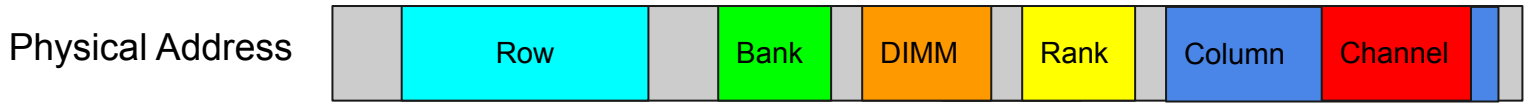
WHICH BITS ??

#1 Bits Recovery



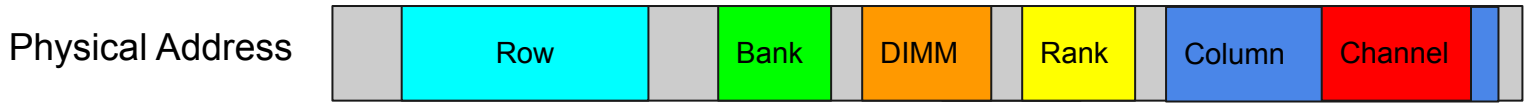

```
char** bank_cluster; // all addresses conflict with each other
```

#1 Bits Recovery




```
char** bank_cluster; // all addresses conflict with each other


for (addr in bank_cluster) {
    for (bit in 0..30) {


    }
}
```

#1 Bits Recovery




```
char** bank_cluster; // all addresses conflict with each other


for (addr in bank_cluster) {
    for (bit in 0..30) {
        new_addr = change_bit(addr, bit);
        time =  time(addr,new_addr);

    }
}
```

#1 Bits Recovery


```
char** bank_cluster; // all addresses conflict with each other


for (addr in bank_cluster) {
    for (bit in 0..30) {
        new_addr = change_bit(addr, bit);
        time =  time(addr,new_addr);
        if (time < threshold) {
            !!!bit is significant for the addressing!!!!
        }
    }
}
```

#1 Bit Recovery

Deliverable:

- Your code that performs a printf(“%llx\n”, significant_bits) at the end
- Significant bits on the five nodes of the cluster
  node_name: significant_bits

#1 Bits Recovery


```
char** bank_cluster; // all addresses conflict with

for (addr in bank_cluster) {
    for (bit in 0..30) {
        new_addr = change_bit(addr, bi
        time =  time(addr,new_addr);
        if (time < threshold) {
            !!!bit is significant for the
        }
    }
}
```

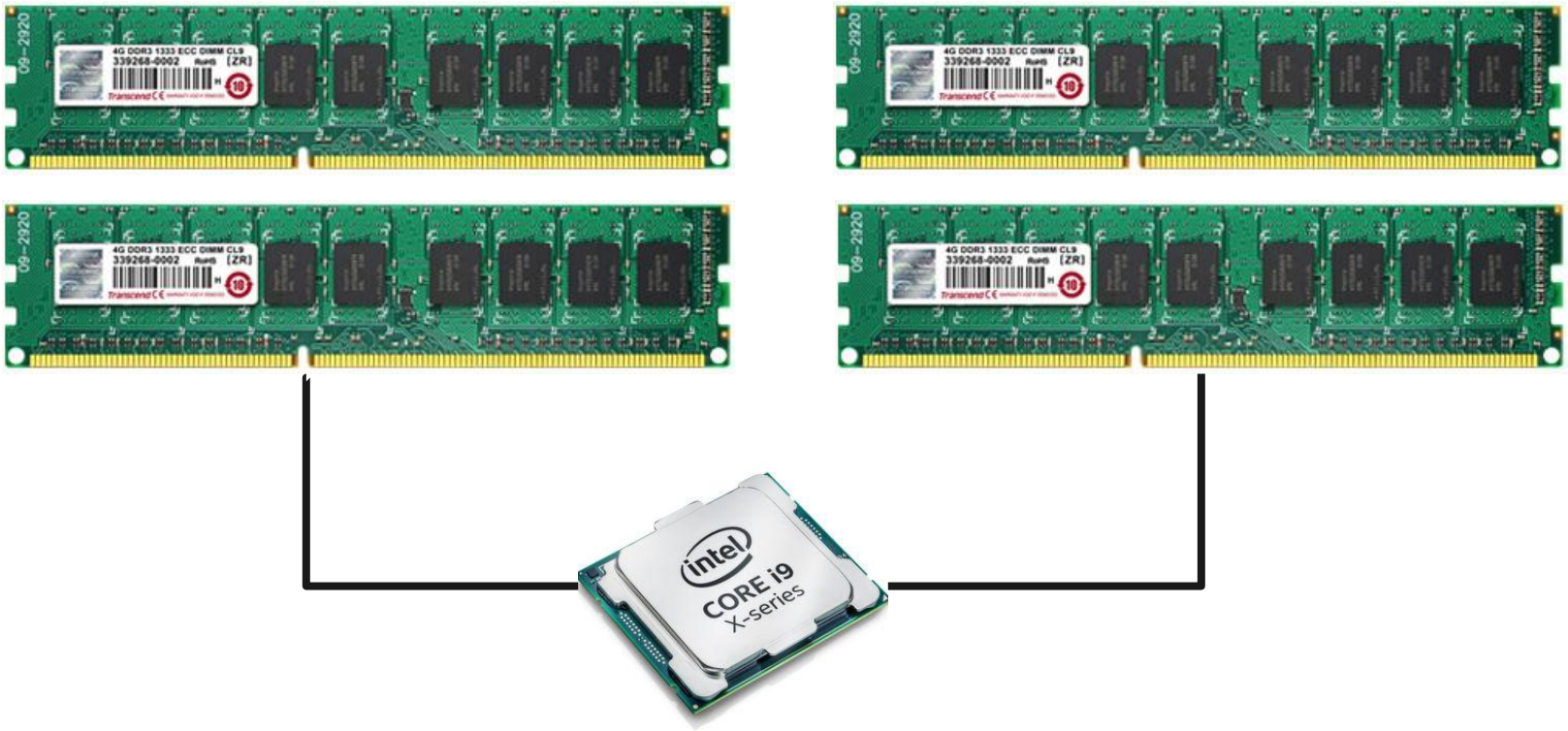
Ok… but what are the actual functions?

Task 2

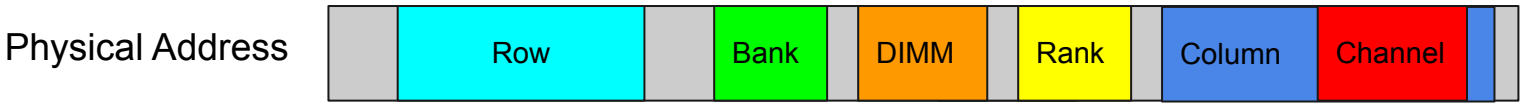
Functions recovery

DIMMs/Channel

Physical Address

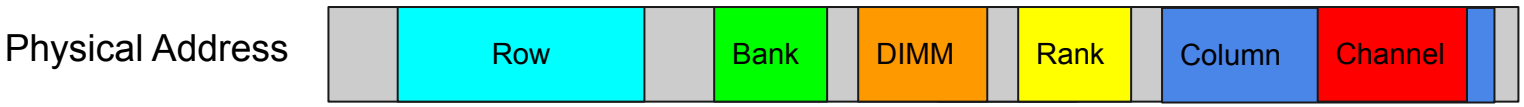


#2 Function Recovery



TOTAL_BANKS = #channels * #dimms * #ranks * #banks

#2 Function Recovery



TOTAL_BANKS = #channels * #dimms * #ranks * #banks

On our cluster only a single DIMM is installed on each compute node → channels = 1, dimms = 1

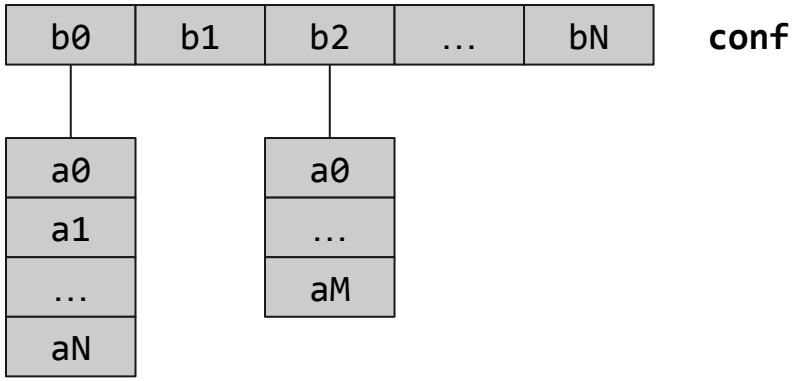
#2 Function Recovery

Bruteforce functions

2 bits	3 bits	N bits
0b000...00011	0b000...00111	0b110...00111
0b000...00101	0b000...01011	0b101...01011
0b000...00110	0b000...01101	0b001...01111
0b010...00100	0b110...00100	0b110...10101

Our CPU's memory controller uses up to only 2 bits for rank/bank functions!

#2 Function Recovery



2 bits

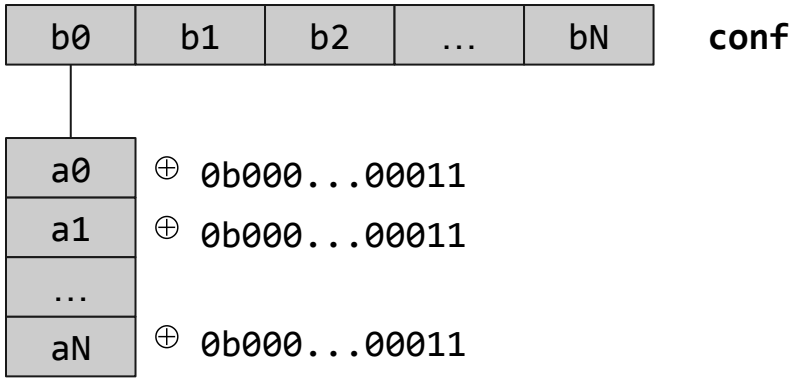
0b000...00011

0b000...00101

0b000...00110

0b010...00100

#2 Function Recovery



2 bits

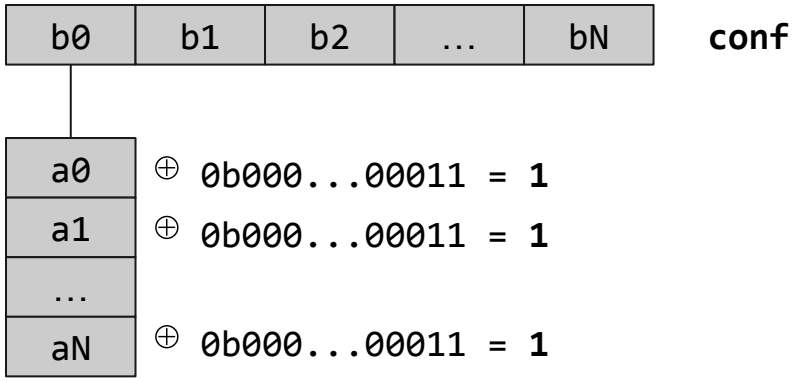
0b000...00011

0b000...00101

0b000...00110

0b010...00100

#2 Function Recovery



2 bits

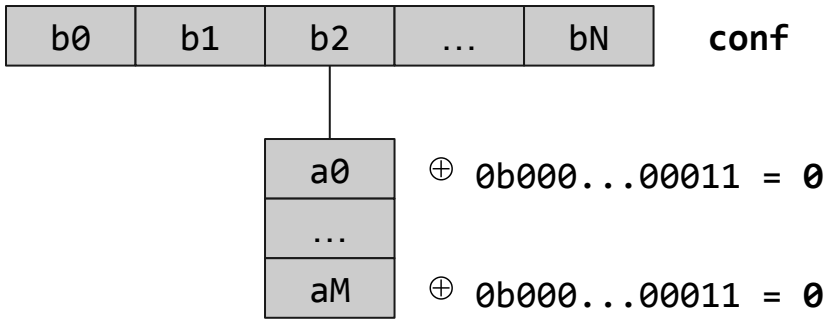
0b000...00011

0b000...00101

0b000...00110

0b010...00100

#2 Function Recovery



2 bits

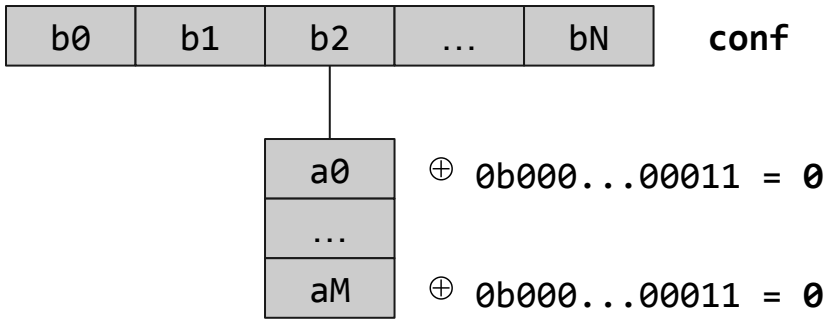
0b000...00011

0b000...00101

0b000...00110

0b010...00100

#2 Function Recovery



#FNs = $\log_2$(#TOTAL_BANKS)

All **(ai ⊕ FN)** equal
==
valid function

2 bits

0b000...00011

0b000...00101

0b000...00110

0b010...00100

#2 Function Recovery

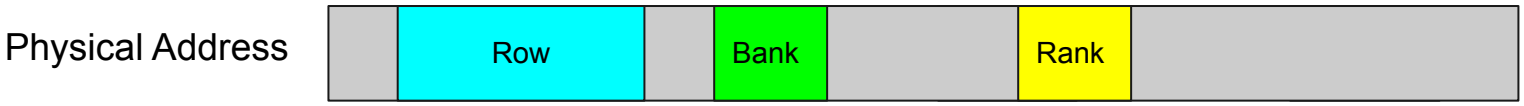
Deliverable:

- Your code that performs a printf(“%d\n”, num_funcs), followed by num_funcs printf(“%llx\n”, funcs[i]) at the end
- DRAM functions on the 5 nodes in our cluster
  node_name: funcs[0] funcs[1] … funcs[n]

Task 3

Row mask recovery

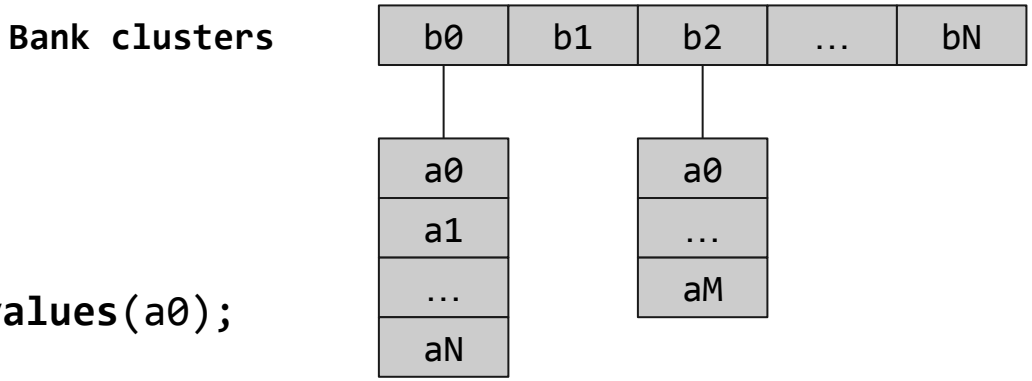
#3 Row mask recovery



Which bits select the row?

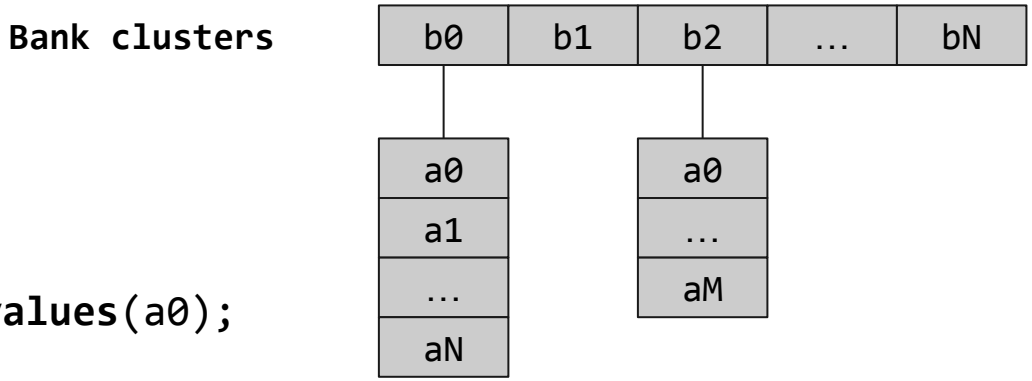
Row mask: bits that when flipped cause conflicts in the same bank

#3 Row mask recovery



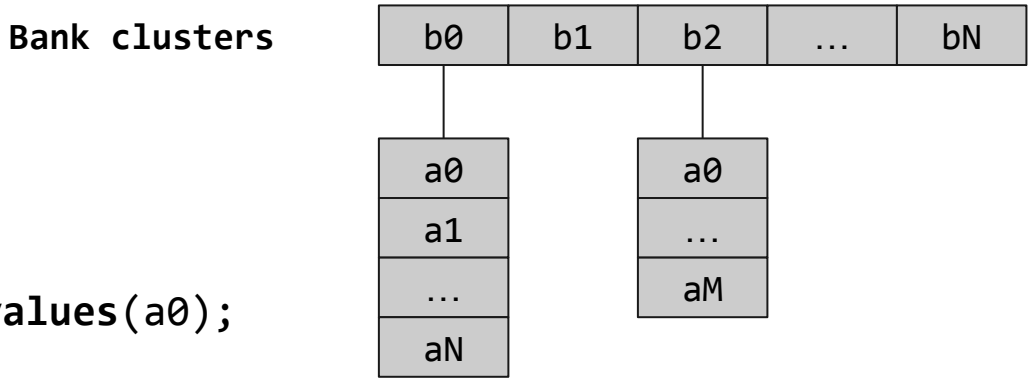

```
char* a0;
int a0_bank[#FNs] = get_funcs_values(a0);
```

#3 Row mask recovery



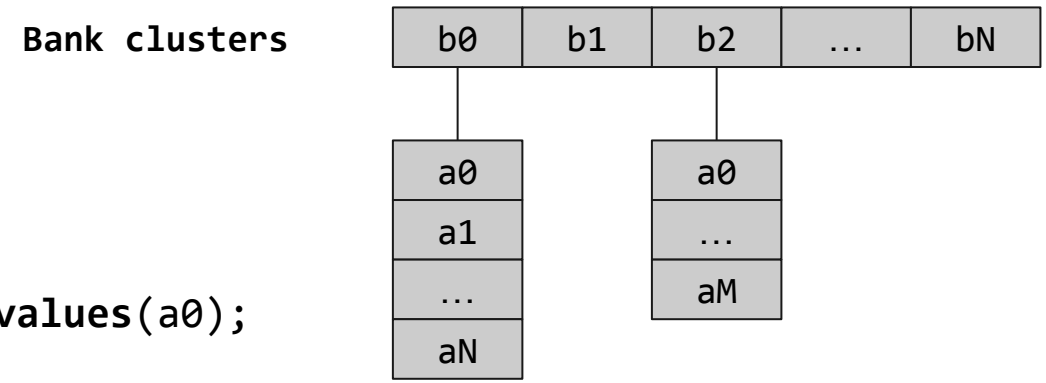

```
char* a0;
int a0_bank[#FNs] = get_funcs_values(a0);

for (bit in 0..30) {





}
```

#3 Row mask recovery




```
char* a0;
int a0_bank[#FNs] = get_funcs_values(a0);

for (bit in 0..30) {
    a0’ = change_bit(a0, bit);
    a0’’ = switch_bank(new_addr, a0_banks);




}
```

#3 Row mask recovery




```
char* a0;
int a0_bank[#FNs] = get_funcs_values(a0);

for (bit in 13..30) {
    a0’ = change_bit(a0, bit);
    a0’’ = switch_bank(new_addr, a0_banks);

    time =  time(a0,a0’’);
    if (time > threshold) {
        !!!highest set bit of a0⊕a0’’ is part of row mask!!!!
    }
}
```

#2 Function Recovery

Deliverable:

- Your code that performs a printf(“%llx\n”, row_mask) at the end
- Row masks on the 5 nodes in our cluster
  node_name: row_mask

Deliverable

- make should build ./{student_number}
- Task 1:
  - ./{student_number} -b should print the bits in hexadecimal (cover all function bits)
- Task 2:
  - ./{student_number} -f should print the number and functions’ masks in hexadecimal
- Task 3:
  - ./{student_number} -m should print the row mask in hexadecimal
- Each should finish execution < 60s

Grading & Deadline

- Deadline:
  - Deadline **Tuesday Sep 29, 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