Datapath Components (3) — Those Who "Remember" Things

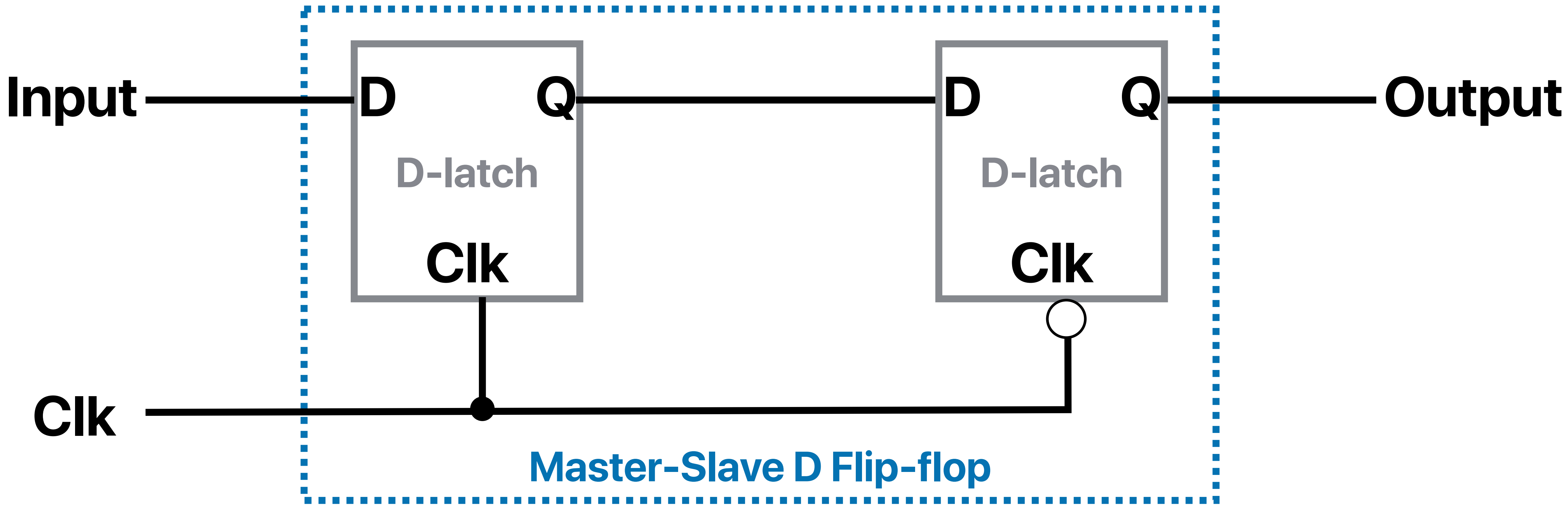
Prof. Usagi

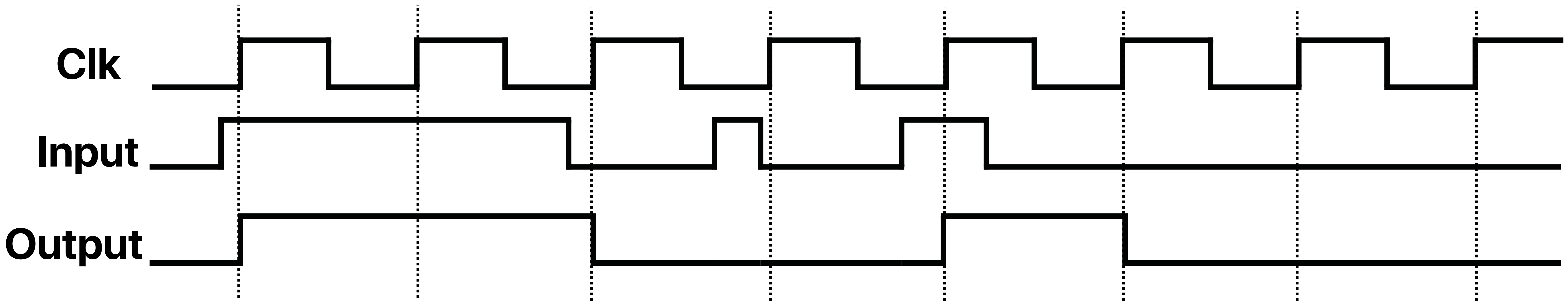
Recap: Combinational v.s. sequential logic

- Combinational logic
  - The output is a pure function of its current inputs
  - The output doesn't change regardless how many times the logic is triggered — Idempotent
- Sequential logic
  - The output depends on current inputs, previous inputs, their history

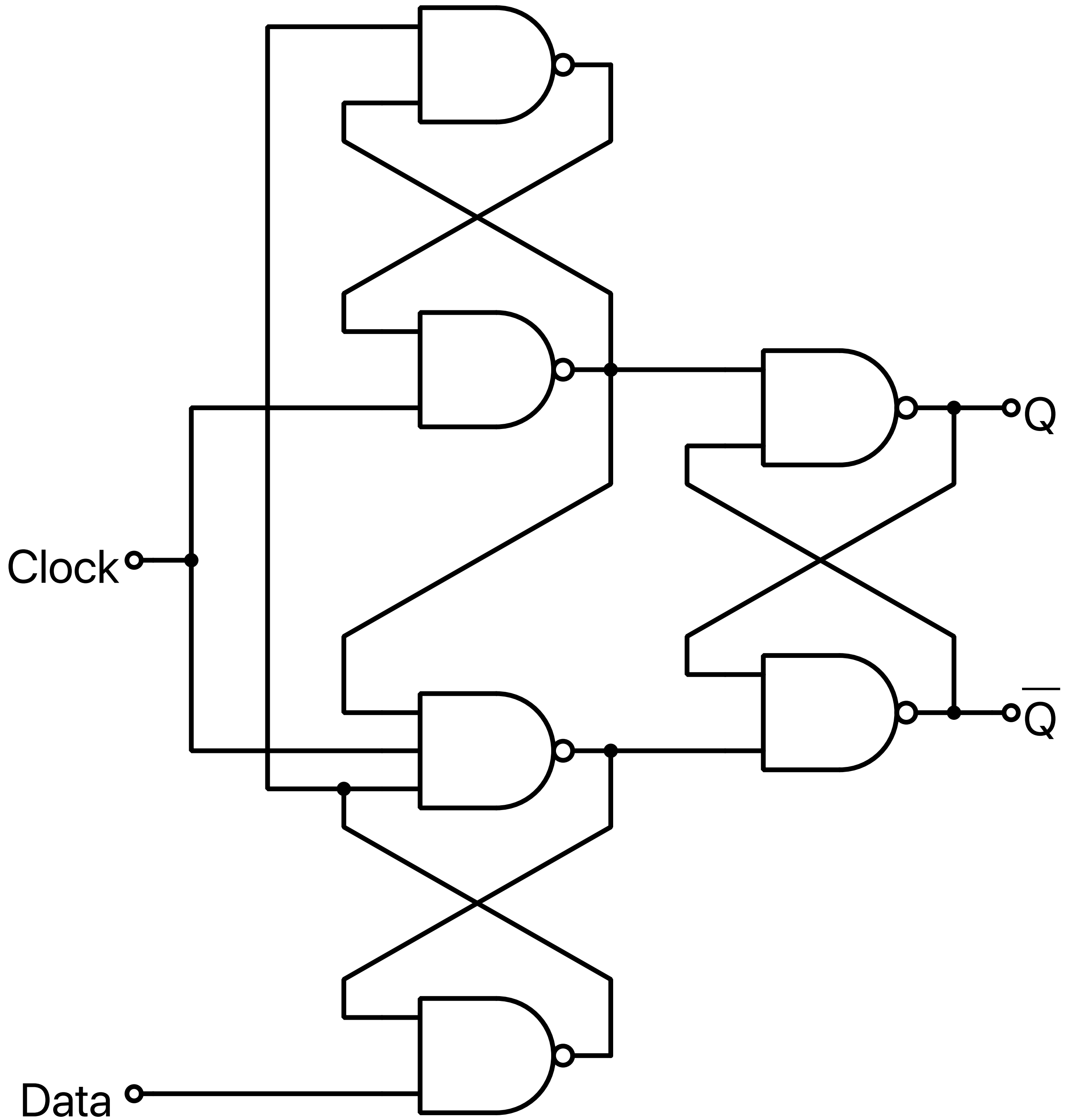
Sequential circuit has memory!

Recap: D flip-flop





Recap: Positive-edge-triggered D flip-flop



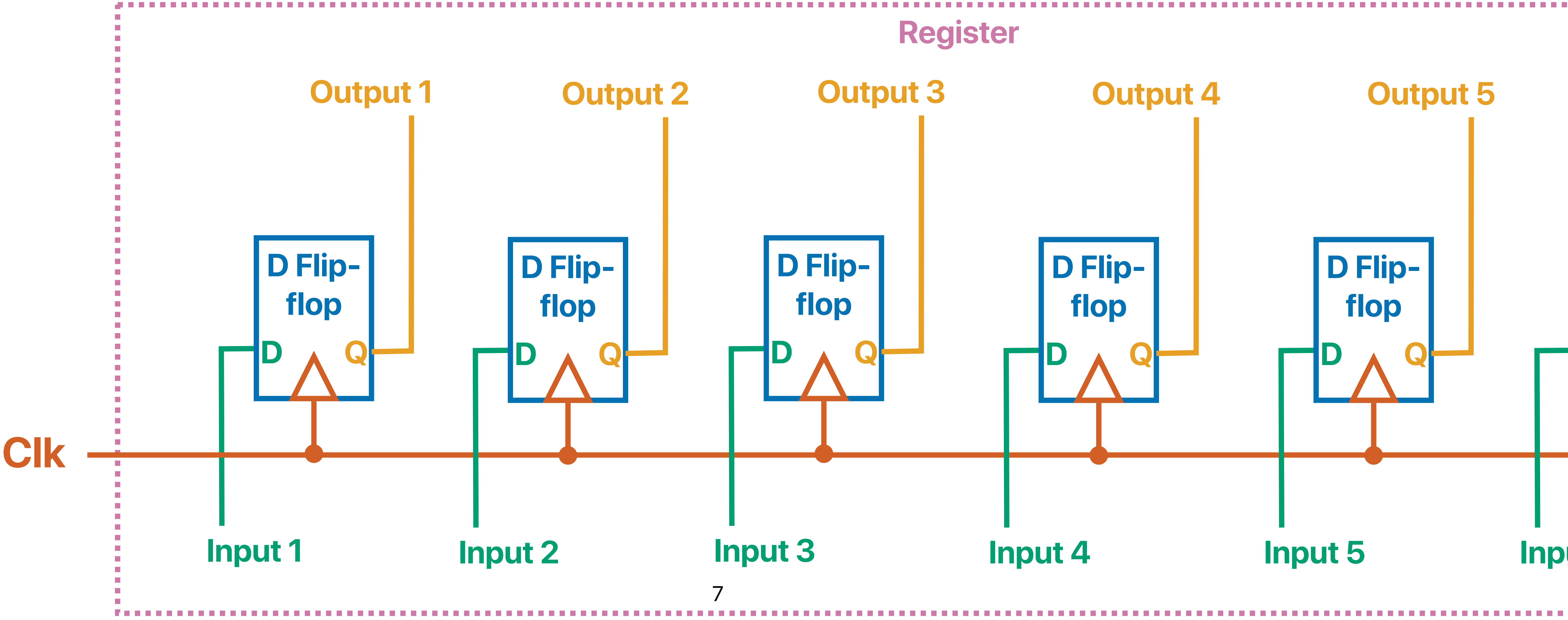
Outline

- Volatile Memory
  - Registers
  - SRAM
  - DRAM
- Programming and memory
- Non-volatile Memory

Registers

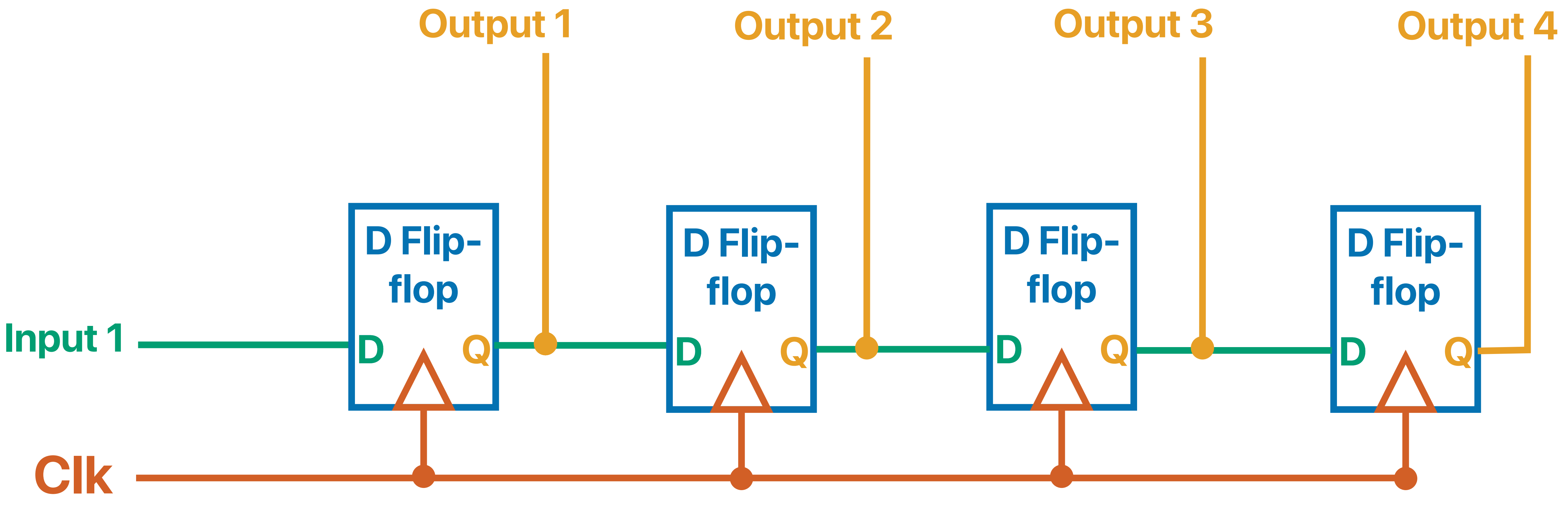
Registers

- Register: a sequential component that can store multiple bits
- A basic register can be built simply by using multiple D-FFs



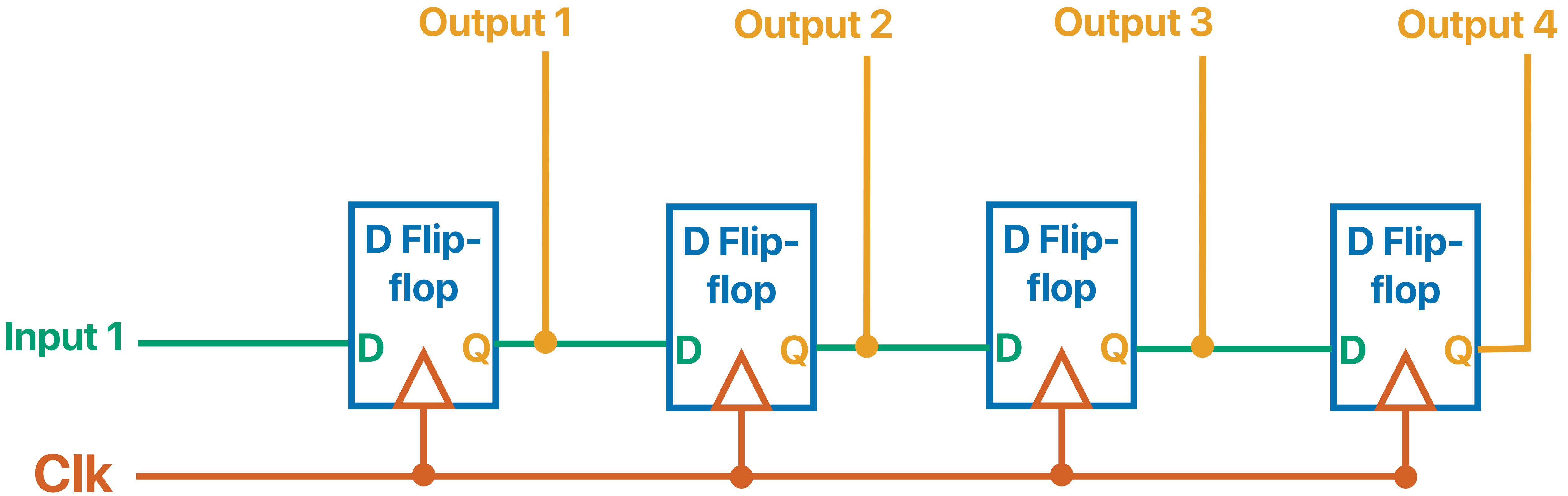
Poll close in 1:30

What will we output 4 cycles later?



- For the above D-FF organization, what are we expecting to see in (O1,O2,O3,O4) in the beginning of the 5th cycle after receiving (1,0,1,1)?
  A. (1,1,1,1)
  B. (1,0,1,1)
  C. (1,1,0,1)
  D. (0,0,1,0)
  E. (0,1,0,0)

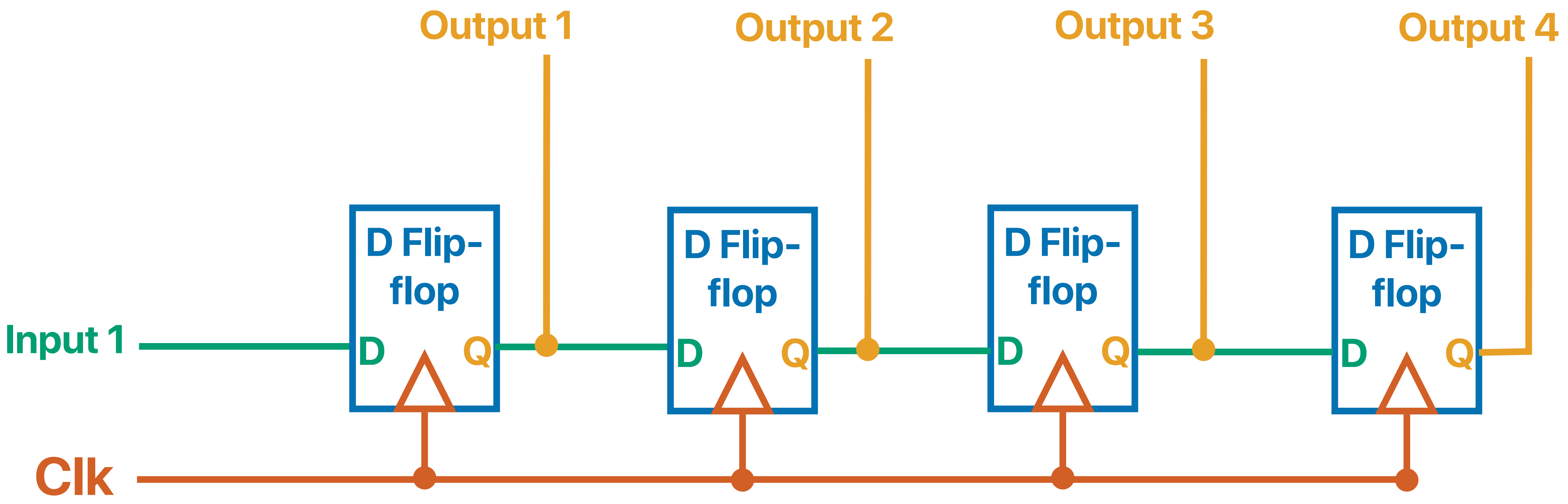
What will we output 4 cycles later?



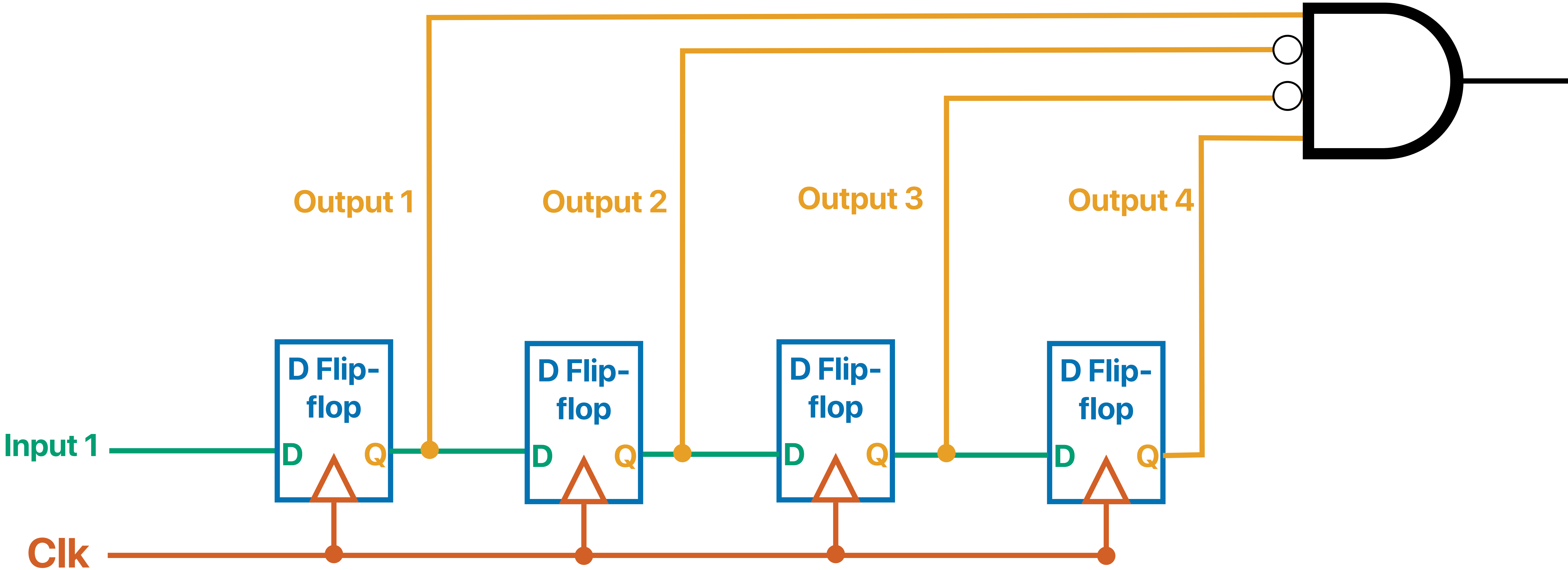
- For the above D-FF organization, what are we expecting to see in (O1,O2,O3,O4) in the beginning of the 5th cycle after receiving (1,0,1,1)?
  A. (1,1,1,1)
  B. (1,0,1,1)
  C. (1,1,0,1)
  D. (0,0,1,0)
  E. (0,1,0,0)

Shift register

- Holds & shifts samples of input



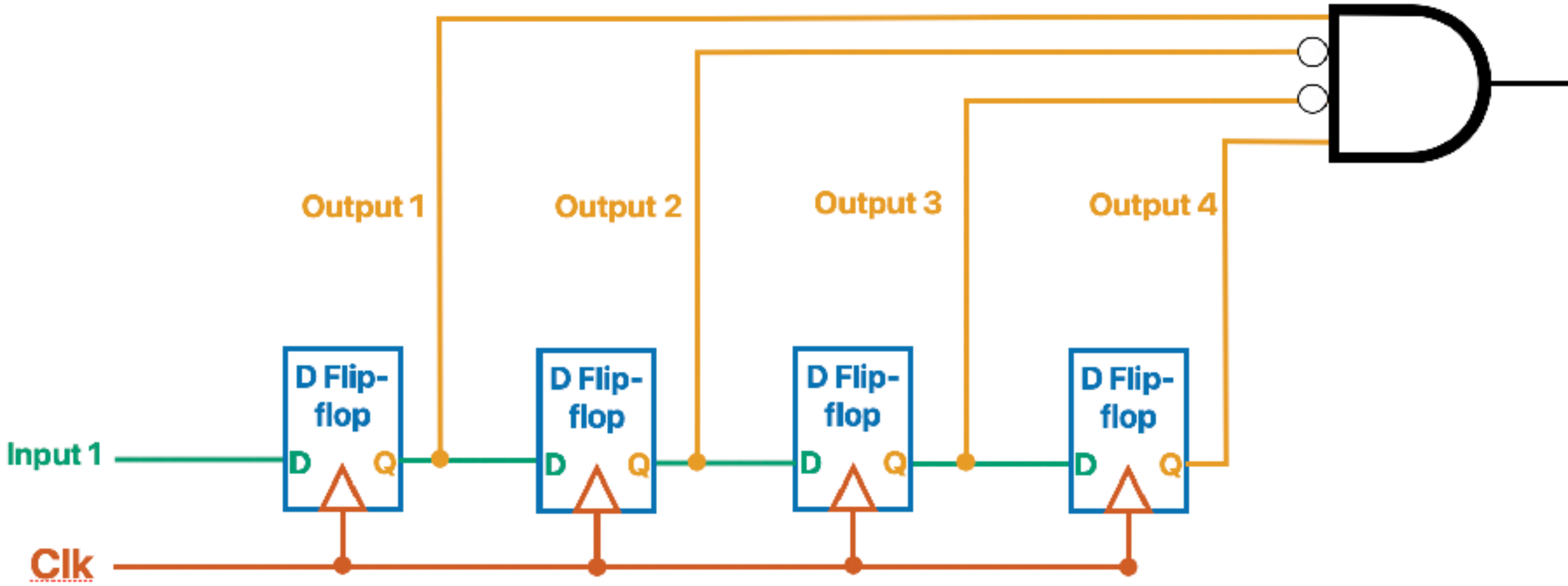
Let's play with the shift register more...



Poll close in 1:30

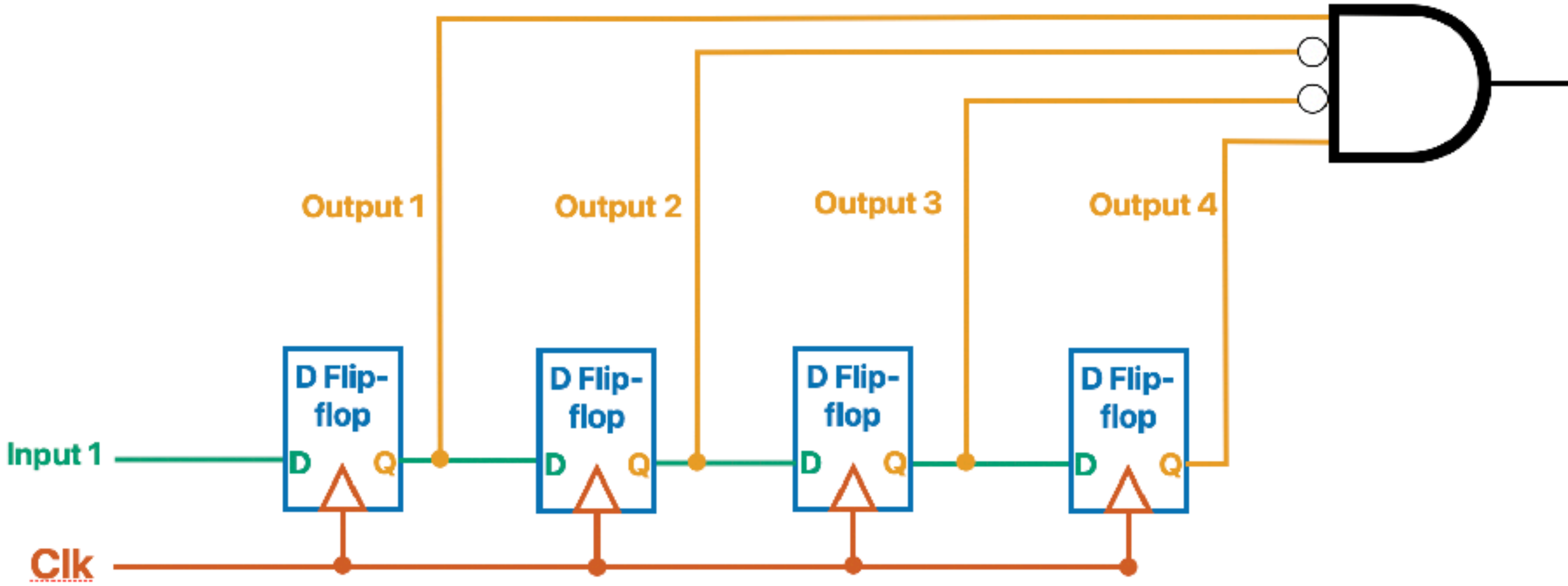
Let's play with the shift register more…

- For the extended shift register, what sequence of input will the let the circuit output "1"?
  A. (1, 1, 1, 1)
  B. (0, 1, 0, 1)
  C. (1, 0, 1, 0)
  D. (0, 1, 1, 0)
  E. (1, 0, 0, 1)



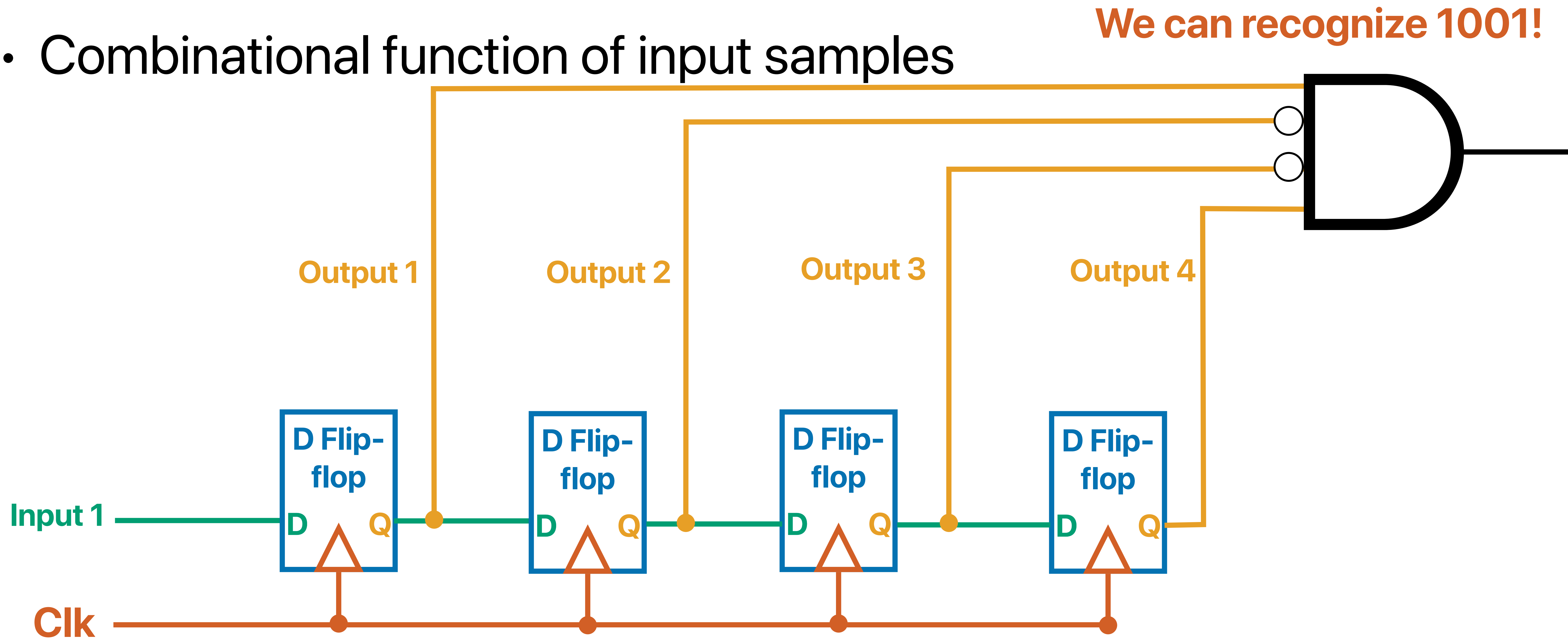
Let's play with the shift register more…

- For the extended shift register, what sequence of input will the let the circuit output "1"?
  A. (1, 1, 1, 1)
  B. (0, 1, 0, 1)
  C. (1, 0, 1, 0)
  D. (0, 1, 1, 0)
  E. (1, 0, 0, 1)



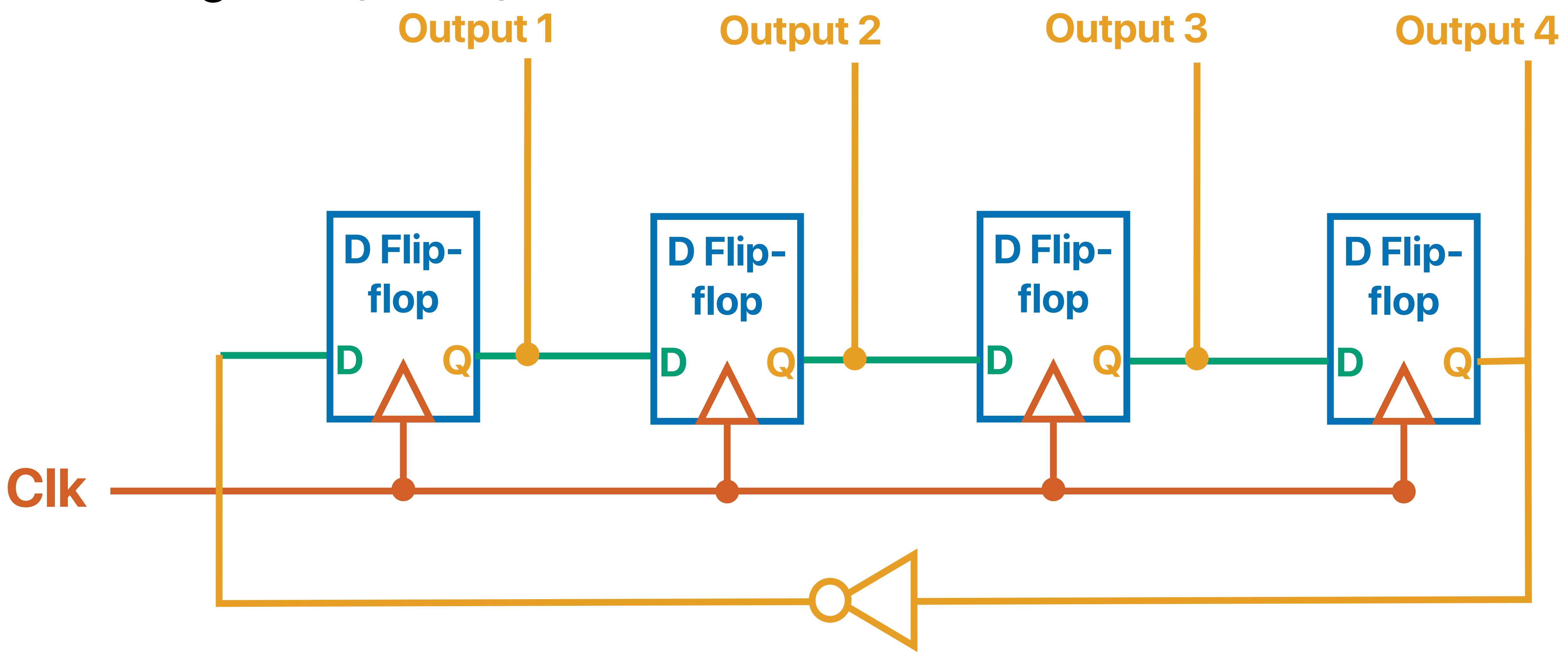
Pattern Recognizer

- Combinational function of input samples



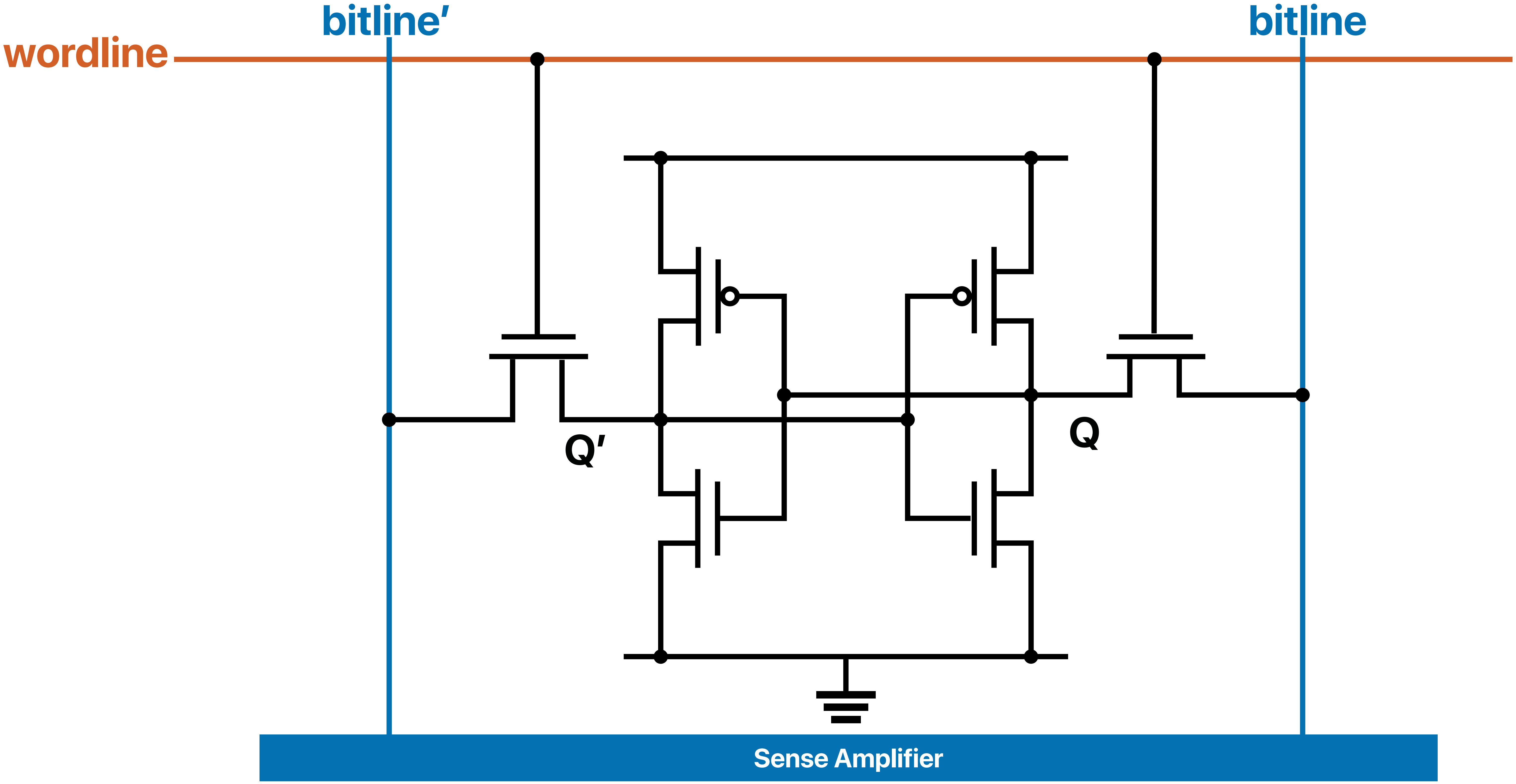
Counters

- Sequences through a fixed set of patterns
- Note: definition is general
- For example, the one in the figure is a type of counter called Linear Feedback Shift Register (LFSR)

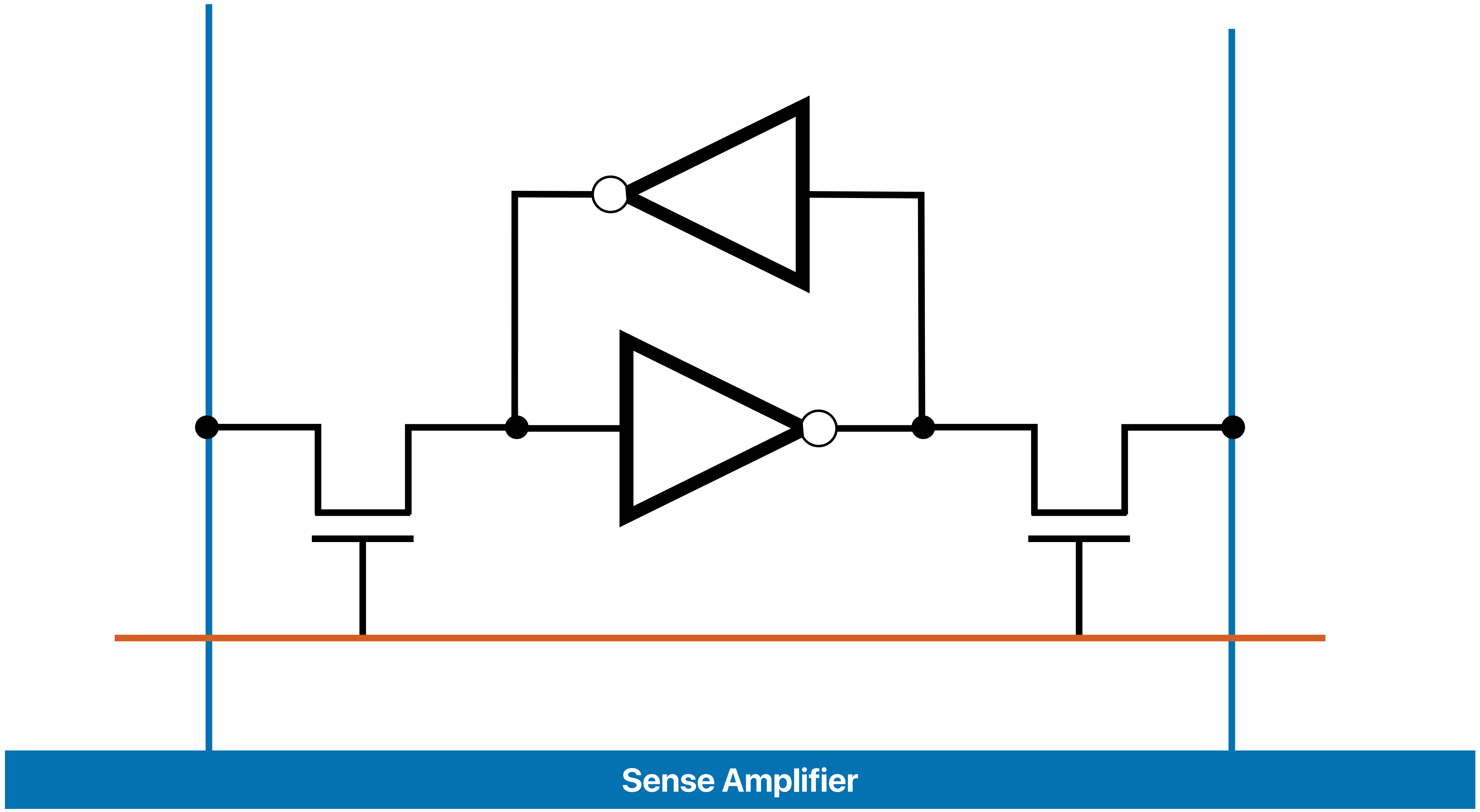


Static Random Access Memory (SRAM)

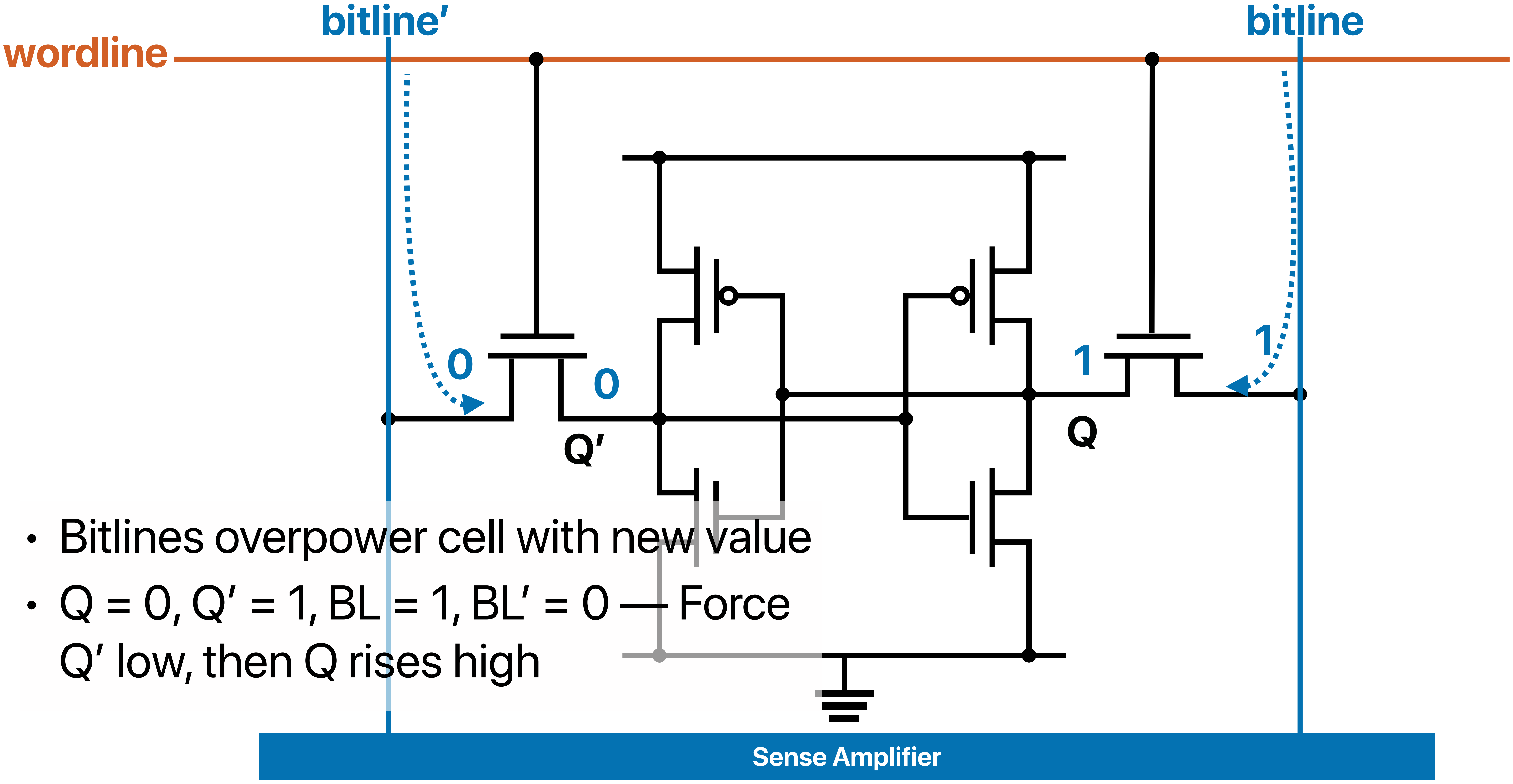
A Classical 6-T SRAM Cell



A Classical 6-T SRAM Cell

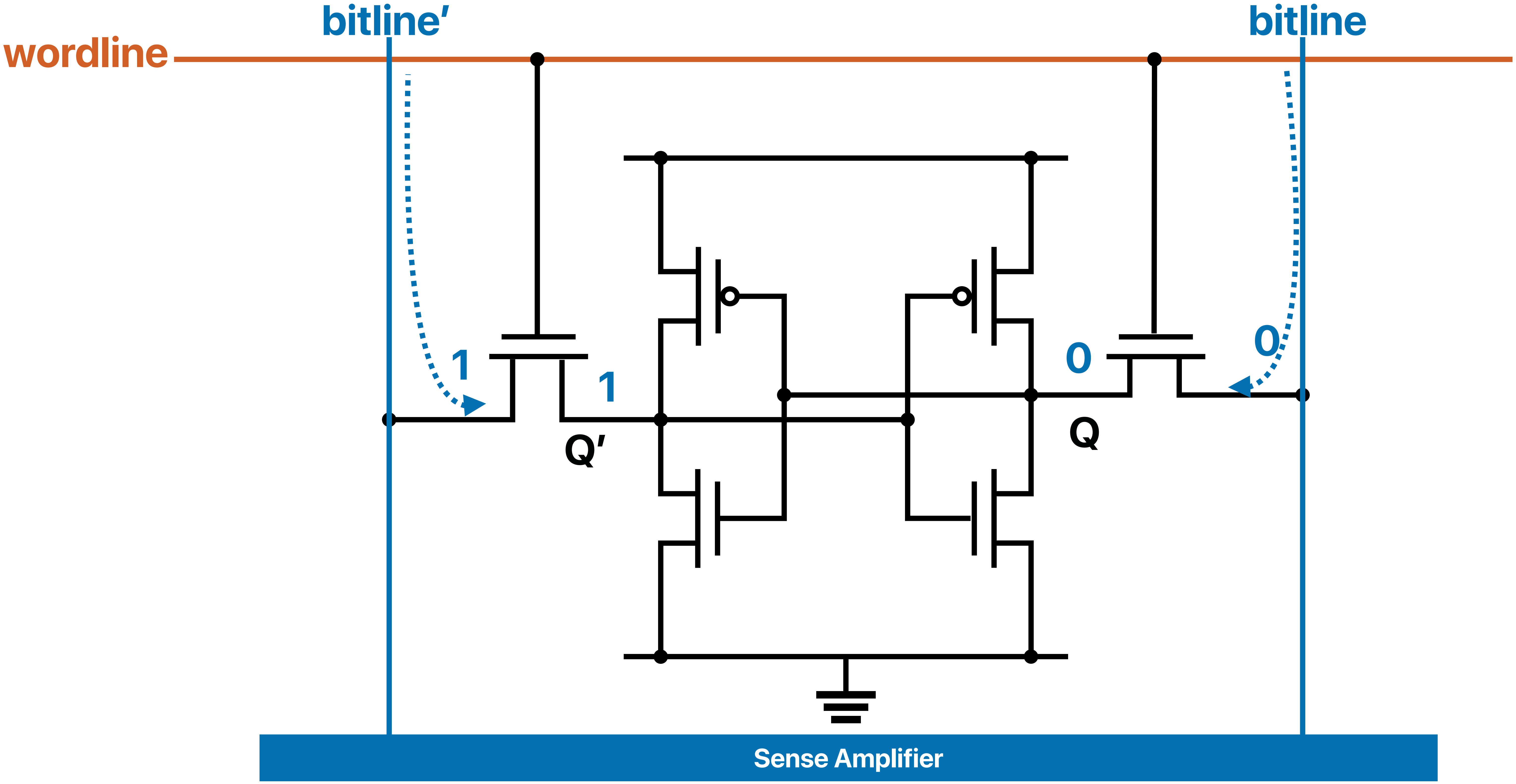


Write "1" to an SRAM Cell

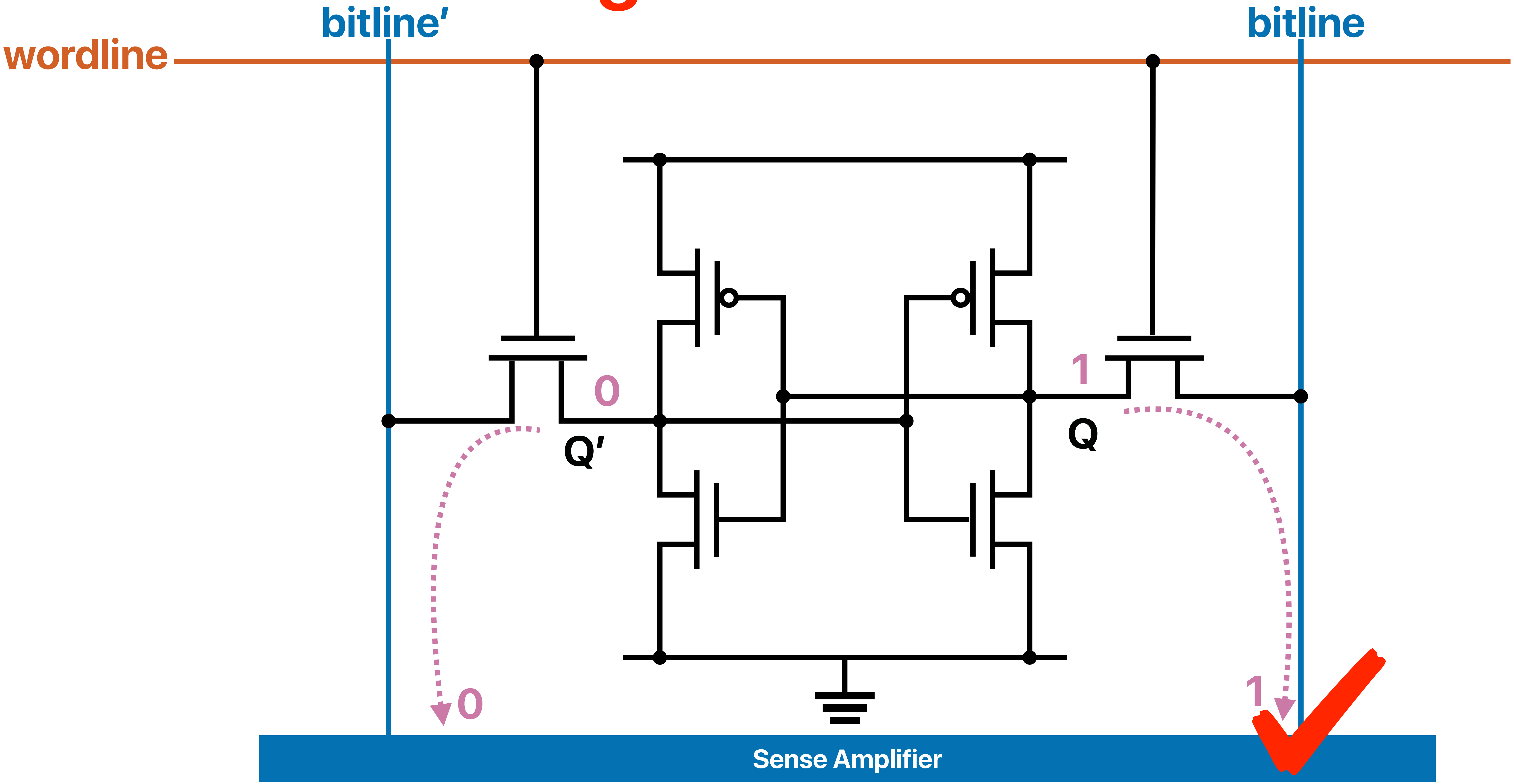


- Bitlines overpower cell with new value
- Q = 0, Q′ = 1, BL = 1, BL′ = 0 — Force Q′ low, then Q rises high

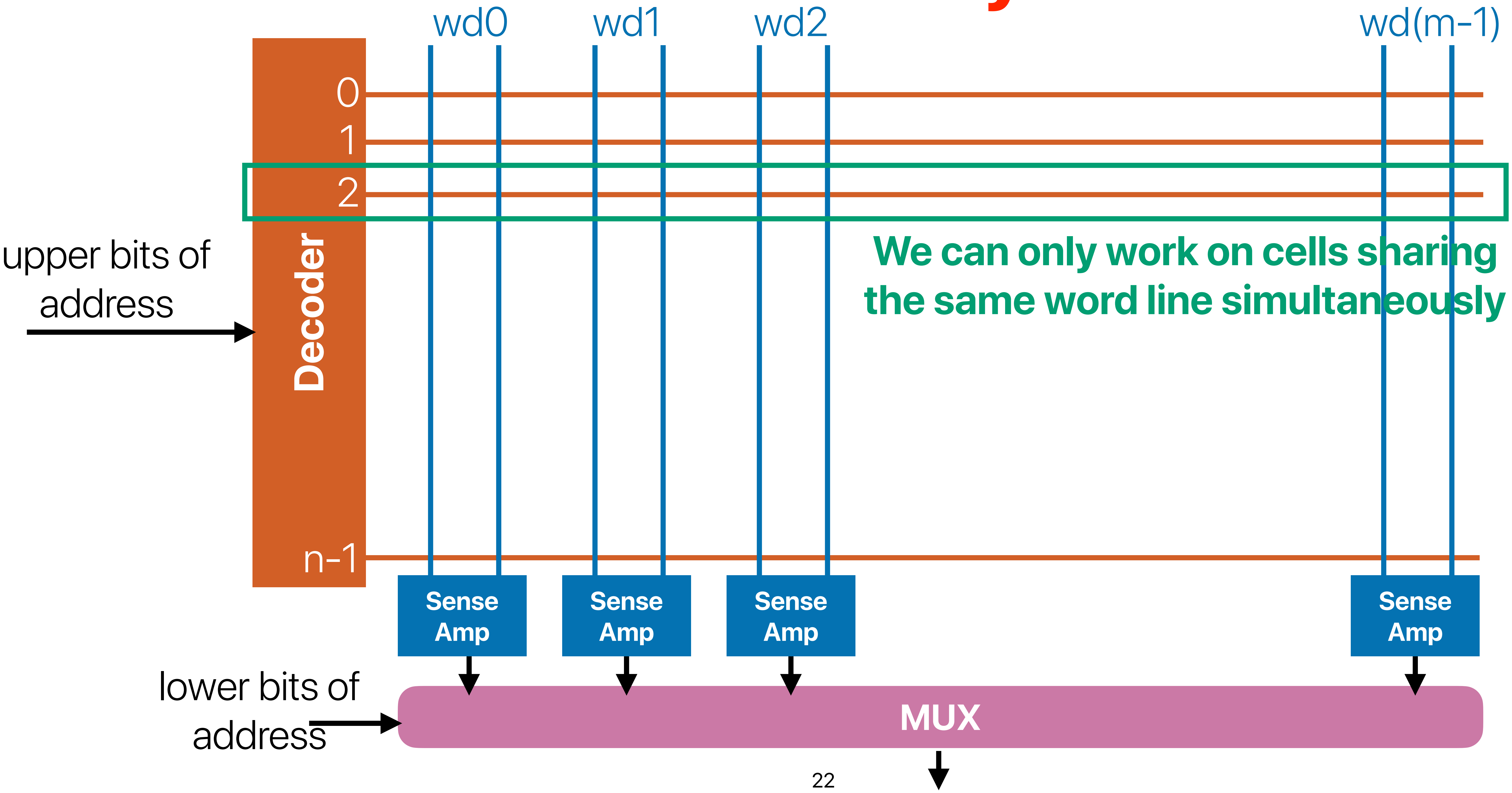
Write "0" to an SRAM Cell



Reading from an SRAM Cell



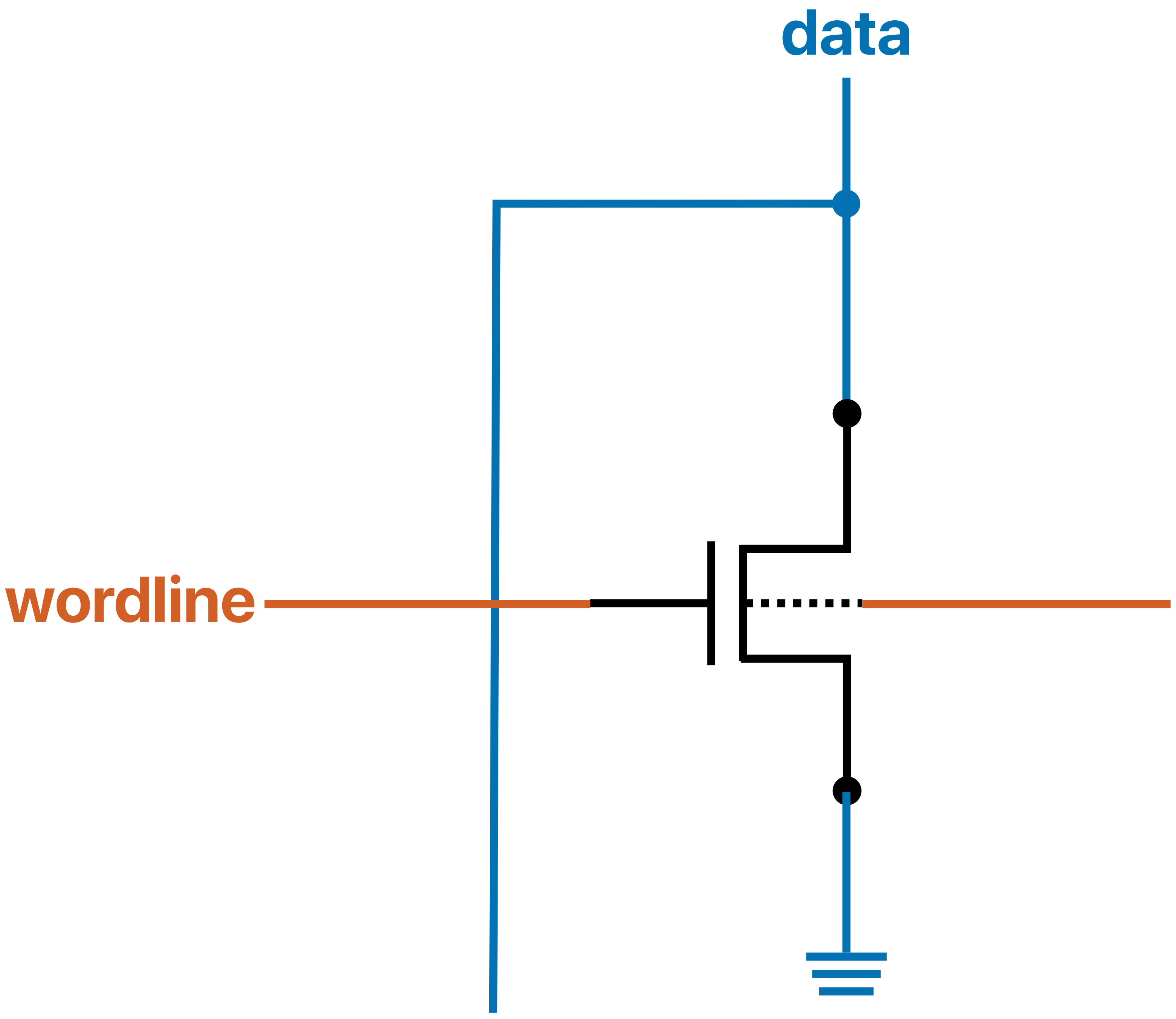
SRAM array



wd0
wd1
wd2
wd(m-1)
0
1
2
n-1
Decoder
upper bits of address
We can only work on cells sharing the same word line simultaneously
Sense Amp
Sense Amp
Sense Amp
Sense Amp
lower bits of address
MUX

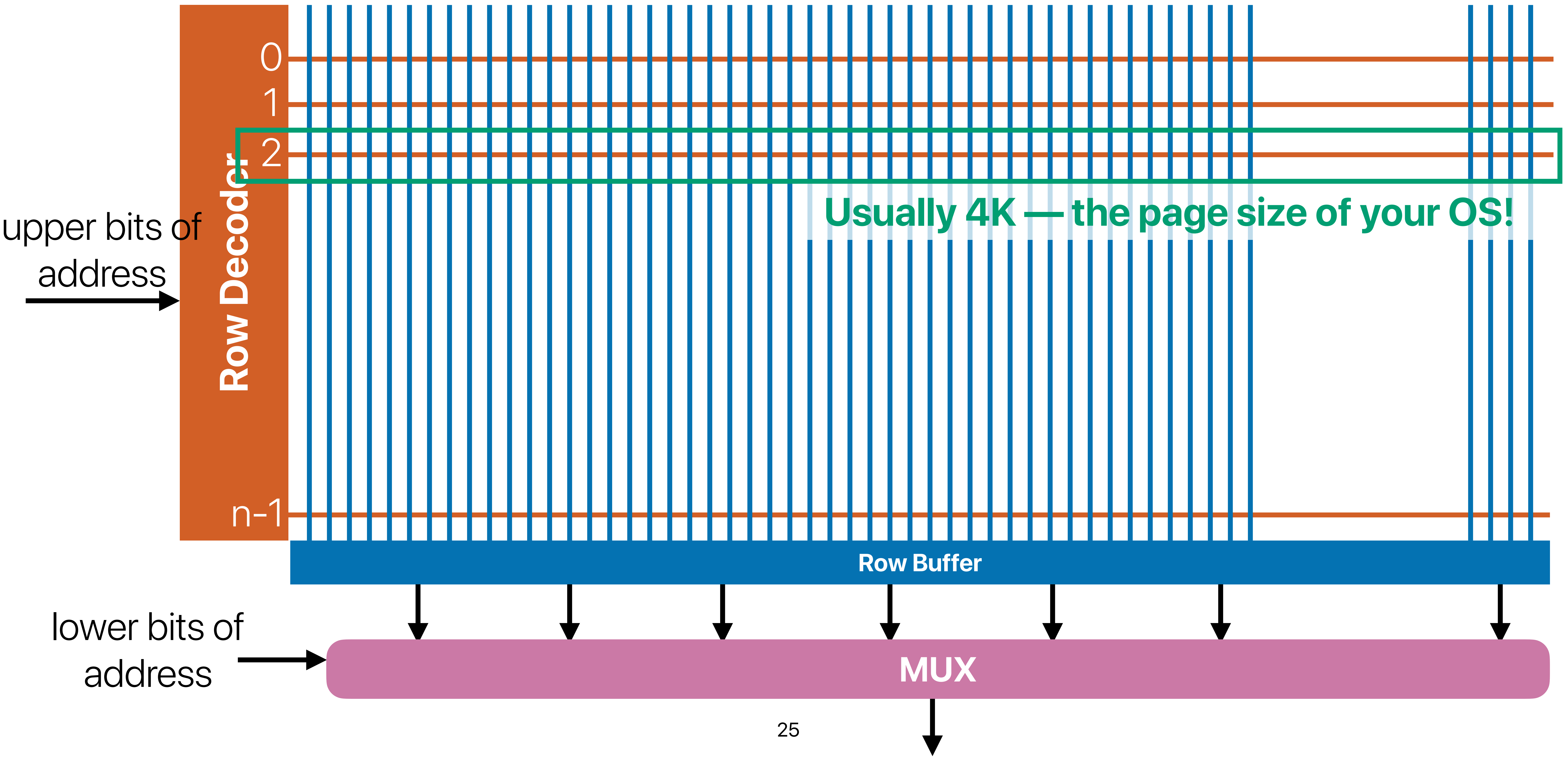
Dynamic Random Access Memory (DRAM)

An DRAM cell



- 1 transistor (rather than 6)
- Relies on large capacitor to store bit
  - Write: transistor conducts, data voltage level gets stored on top plate of capacitor
  - Read: look at the value of d
- Problem: Capacitor discharges over time
  - Must "refresh" regularly, by reading d and then writing it right back

DRAM array



0
1
2
n-1
Row Decoder
upper bits of address
Usually 4K — the page size of your OS!
Row Buffer
lower bits of address
MUX

Poll close in 1:30

Register v.s. DRAM v.s. SRAM

- Consider the following memory elements
  ① 64*64-bit Registers
  ② 512B SRAM
  ③ 512B DRAM

A. Area: (1) > (2) > (3) Delay: (1) < (2) < (3)
B. Area: (1) > (3) > (2) Delay: (1) < (3) < (2)
C. Area: (3) > (1) > (2) Delay: (1) < (3) < (2)
D. Area: (3) > (2) > (1) Delay: (3) < (2) < (1)
E. Area: (2) > (3) > (1) Delay: (2) < (3) < (1)

Register v.s. DRAM v.s. SRAM

- Consider the following memory elements
  1. 64*64-bit Registers
  2. 512B SRAM
  3. 512B DRAM

A. Area: (1) > (2) > (3) Delay: (1) < (2) < (3)

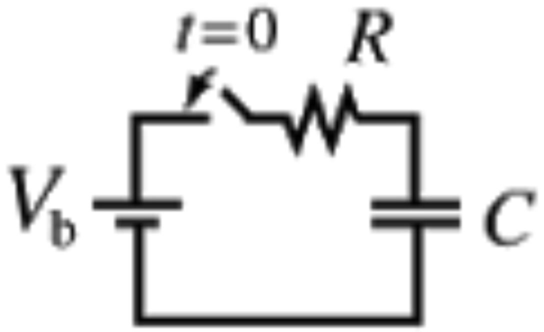
B. Area: (1) > (3) > (2) Delay: (1) < (3) < (2)

C. Area: (3) > (1) > (2) Delay: (1) < (3) < (2)

D. Area: (3) > (2) > (1) Delay: (3) < (2) < (1)

E. Area: (2) > (3) > (1) Delay: (2) < (3) < (1)

RC charging



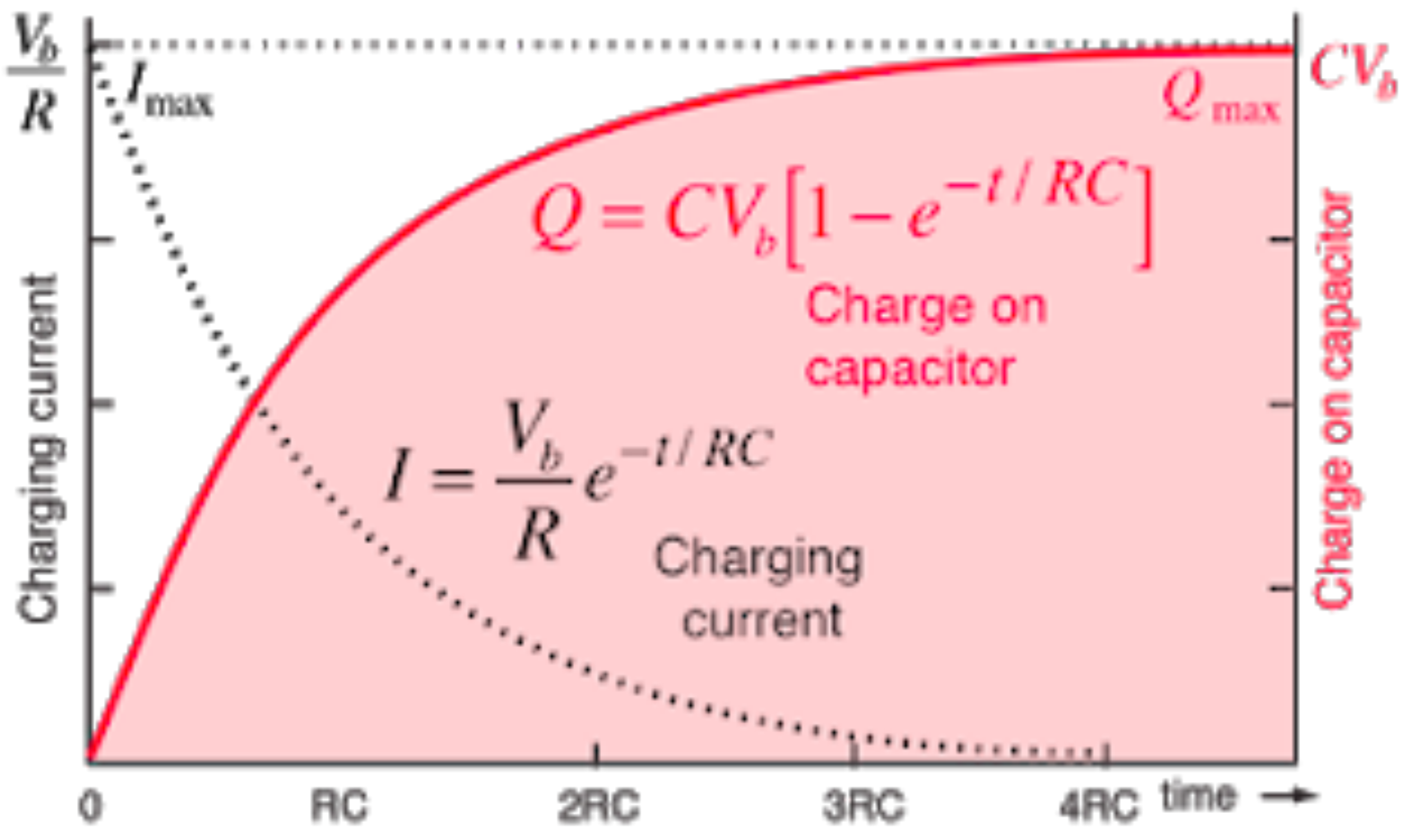
$$V_b = V_R + V_C$$

$$V_b = IR + \frac{Q}{C}$$

As charging progresses,

$$V_b = IR + \frac{Q}{C}$$

current decreases and charge increases.



At $t = 0$
$Q = 0$
$V_C = 0$
$I = \frac{V_b}{R}$

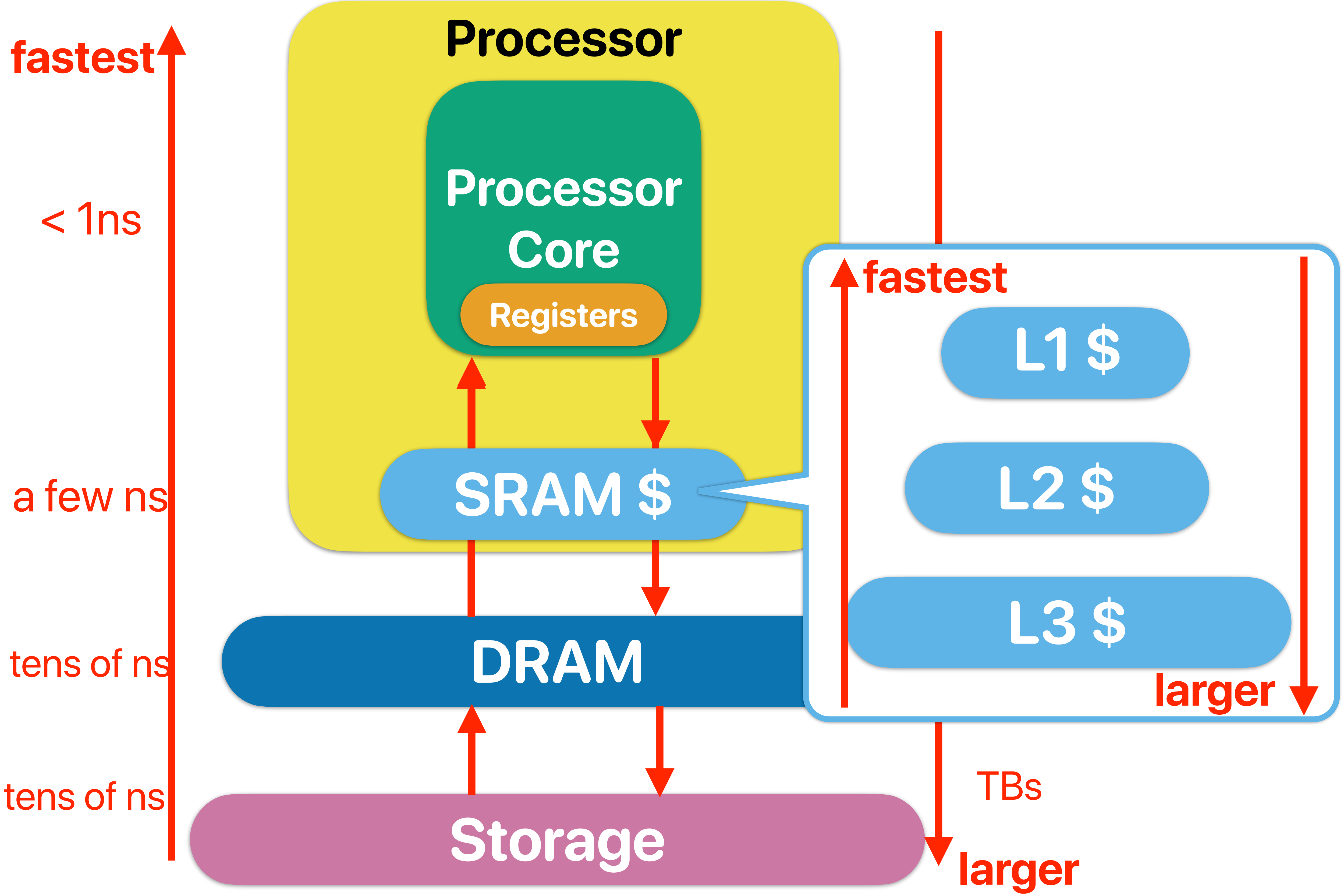
As $t \to \infty$
$Q \to CV_b$
$V_C \to V_b$
$I \to 0$

Latency of volatile memory

	Size (Transistors per bit)	Latency (ns)
Register	18T	~ 0.1 ns
SRAM	6T	~ 0.5 ns
DRAM	1T	50-100 ns

Programming and memory

Memory “hierarchy” in modern processor architectures



Poll close in 1:30

Thinking about programming

```
struct student_record
{
    int id;
    double homework;
    double midterm;
    double final;
};

int main(int argc, char **argv)
{
    int i,j;
    double midterm_average=0.0;
    int number_of_records = 10000000;
    struct timeval time_start, time_end;
    struct student_record *records;
    records = (struct
student_record*)malloc(sizeof(struct
student_record)*number_of_records);
    init(number_of_records,records);


    for (j = 0; j < 100; j++)
        for (i = 0; i < number_of_records; i++)
            midterm_average+=records[i].midterm;

    printf("average: %lf\n",midterm_average/
number_of_records);
   free(records);
    return 0;
}
```

```
int main(int argc, char **argv)
{
    int i,j;
    double midterm_average=0.0;
    int number_of_records = 10000000;
    struct timeval time_start, time_end;
    id = (int*)malloc(sizeof(int)*number_of_records);
    midterm = (double*)malloc(sizeof(double)*number_of_records);
    final = (double*)malloc(sizeof(double)*number_of_records);
    homework = (double*)malloc(sizeof(double)*number_of_records);
    init(number_of_records);

    for (j = 0; j < 100; j++)
        for (i = 0; i < number_of_records; i++)
            midterm_average+=midterm[i];

    free(id);
    free(midterm);
    free(final);
    free(homework);
    return 0;
}
```

- Which side is faster in executing the for-loop?
  - A. Left
  - B. Right
  - C. About the same

Thinking about programming

```
struct student_record
{
    int id;
    double homework;
    double midterm;
    double final;
};

int main(int argc, char **argv)
{
    int i,j;
    double midterm_average=0.0;
    int number_of_records = 10000000;
    struct timeval time_start, time_end;
    struct student_record *records;
    records = (struct
student_record*)malloc(sizeof(struct
student_record)*number_of_records);
    init(number_of_records,records);


    for (j = 0; j < 100; j++)
        for (i = 0; i < number_of_records; i++)
            midterm_average+=records[i].midterm;

    printf("average: %lf\n",midterm_average/
number_of_records);
   free(records);
    return 0;
}
```

```
int main(int argc, char **argv)
{
    int i,j;
    double midterm_average=0.0;
    int number_of_records = 10000000;
    struct timeval time_start, time_end;
    id = (int*)malloc(sizeof(int)*number_of_records);
    midterm = (double*)malloc(sizeof(double)*number_of_records);
    final = (double*)malloc(sizeof(double)*number_of_records);
    homework = (double*)malloc(sizeof(double)*number_of_records);
    init(number_of_records);

    for (j = 0; j < 100; j++)
        for (i = 0; i < number_of_records; i++)
            midterm_average+=midterm[i];

    free(id);
    free(midterm);
    free(final);
    free(homework);
    return 0;
}
```

More row buffer hits in the DRAM, more SRAM hits

- Which side is faster in executing the for-loop?
  - A. Left
  - B. Right
  - C. About the same

Poll close in 1:30

Thinking about programming (2)

```
struct student_record
{
    int id;
    double homework;
    double midterm;
    double final;
};

int main(int argc, char **argv)
{
    int i,j;
    double midterm_average=0.0;
    int number_of_records = 10000000;
    struct timeval time_start, time_end;
    struct student_record *records;
    records = (struct
student_record*)malloc(sizeof(struct
student_record)*number_of_records);
    init(number_of_records,records);


    for (j = 0; j < 100; j++)
        for (i = 0; i < number_of_records; i++)
            midterm_average+=records[i].midterm;

    printf("average: %lf\n",midterm_average/
number_of_records);
   free(records);
    return 0;
}
```

```
int main(int argc, char **argv)
{
    int i,j;
    double midterm_average=0.0;
    int number_of_records = 10000000;
    struct timeval time_start, time_end;
    id = (int*)malloc(sizeof(int)*number_of_records);
    midterm = (double*)malloc(sizeof(double)*number_of_records);
    final = (double*)malloc(sizeof(double)*number_of_records);
    homework = (double*)malloc(sizeof(double)*number_of_records);
    init(number_of_records);

    for (j = 0; j < 100; j++)
        for (i = 0; i < number_of_records; i++)
            midterm_average+=midterm[i];

    free(id);
    free(midterm);
    free(final);
    free(homework);
    return 0;
}
```

- Which side is consuming less memory?
  - A. Left
  - B. Right
  - C. About the same

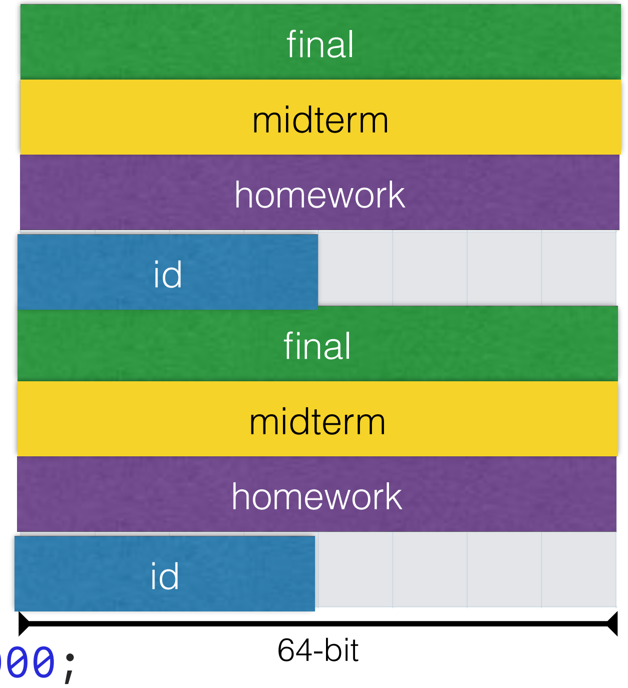
Thinking about programming (2)

```
struct student_record
{
    int id;
    double homework;
    double midterm;
    double final;
};

int main(int argc, char **argv)
{
    int i,j;
    double midterm_average=0.0;
    int number_of_records = 10000000;
    struct timeval time_start, time_end;
    struct student_record *records;
    records = (struct
student_record*)malloc(sizeof(struct
student_record)*number_of_records);
    init(number_of_records,records);


    for (j = 0; j < 100; j++)
        for (i = 0; i < number_of_records; i++)
            midterm_average+=records[i].midterm;

    printf("average: %lf\n",midterm_average/
number_of_records);
   free(records);
    return 0;
}
```




```
int main(int argc, char **argv)
{
    int i,j;
    double midterm_average=0.0;
    int number_of_records = 10000000;
    struct timeval time_start, time_end;
    id = (int*)malloc(sizeof(int)*number_of_records);
    midterm = (double*)malloc(sizeof(double)*number_of_records);
    final = (double*)malloc(sizeof(double)*number_of_records);
    homework = (double*)malloc(sizeof(double)*number_of_records);
    init(number_of_records);

    for (j = 0; j < 100; j++)
        for (i = 0; i < number_of_records; i++)
            midterm_average+=midterm[i];

    free(id);
    free(midterm);
    free(final);
    free(homework);
    return 0;
}
```

- Which side is consuming less memory?
  - A. Left
  - B. Right
  - C. About the same

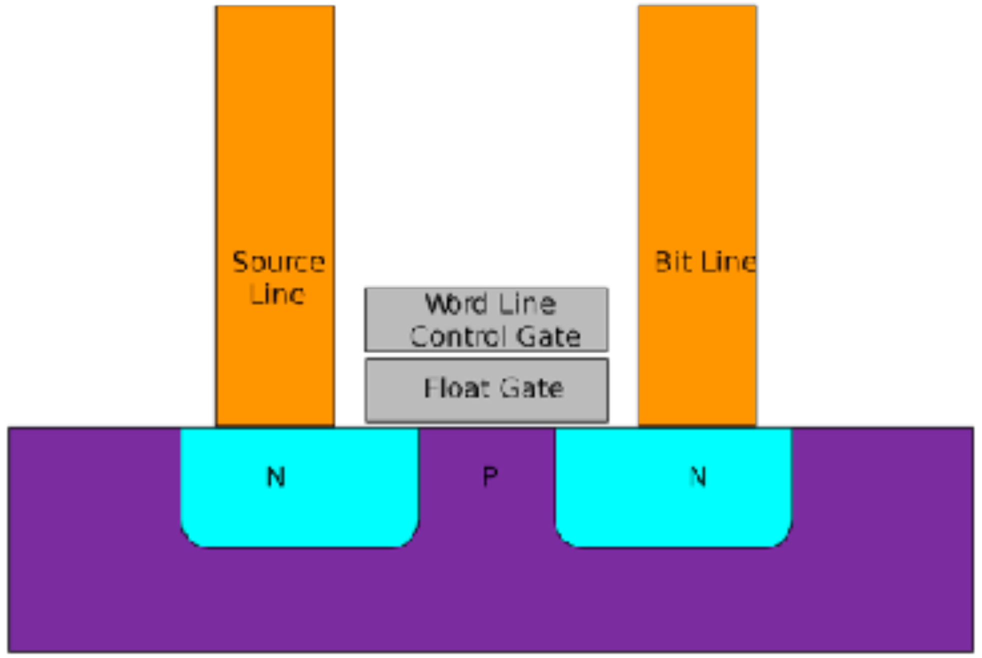
Non-volatile memory

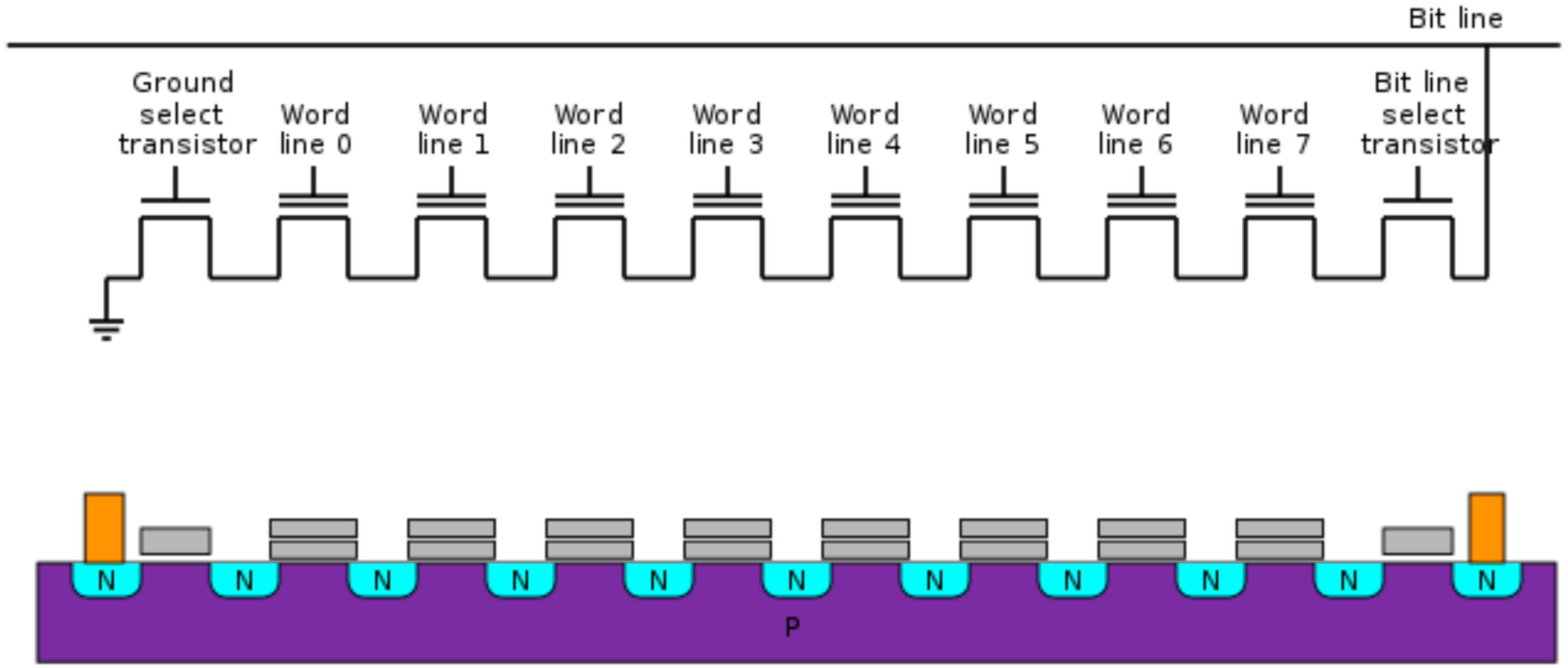
Volatile v.s. Non-volatile

- Volatile memory
  - The stored bits will vanish if the cell is not supplied with eletricity
  - Register, SRAM, DRAM
- Non-volatile memory
  - The stored bits will not vanish “immediately” when it’s out of electricity — usually can last years
  - Flash memory, PCM, MRAM, STTRAM

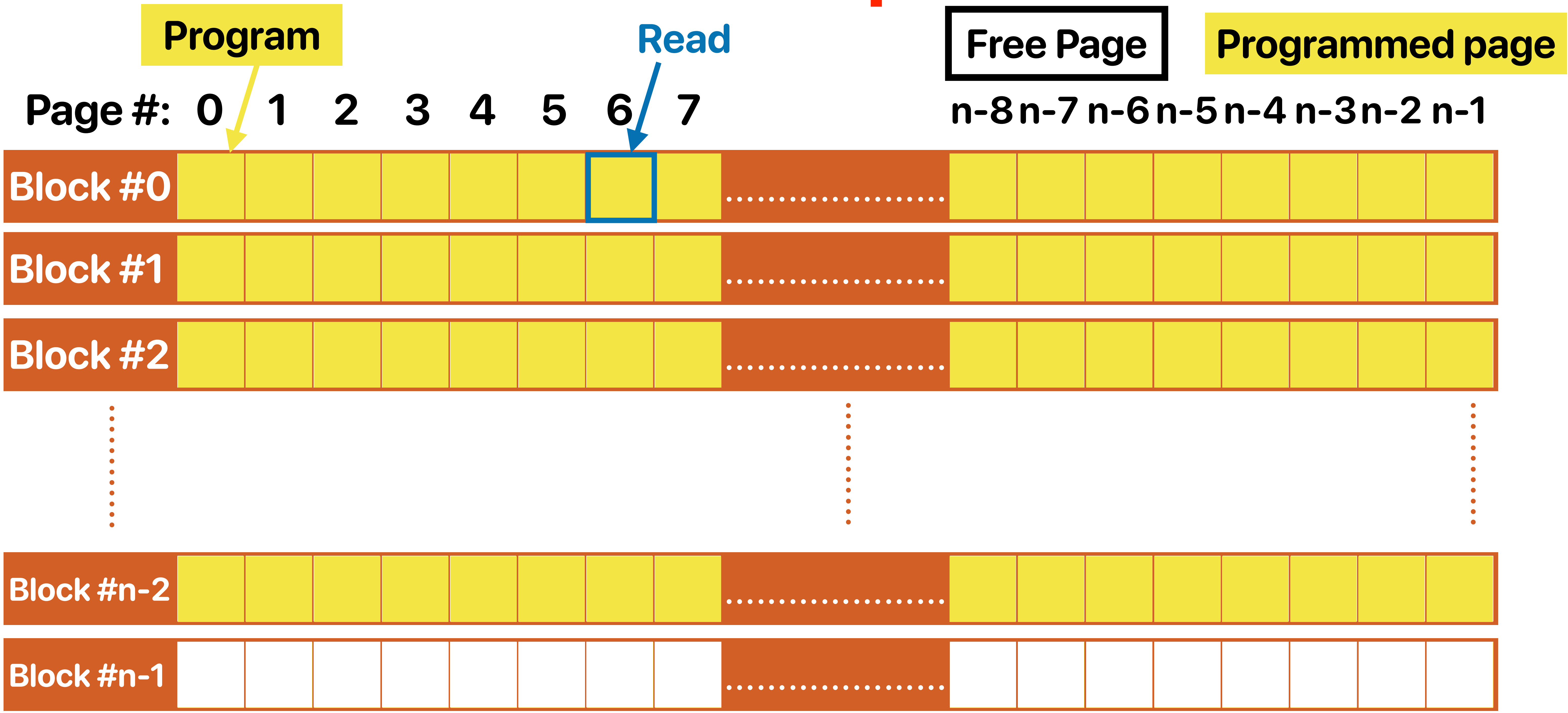
Flash memory

- Floating gate made by polycrystalline silicon trap electrons
- The voltage level within the floating gate determines the value of the cell
- The floating gates will wear out eventually





Basic flash operations



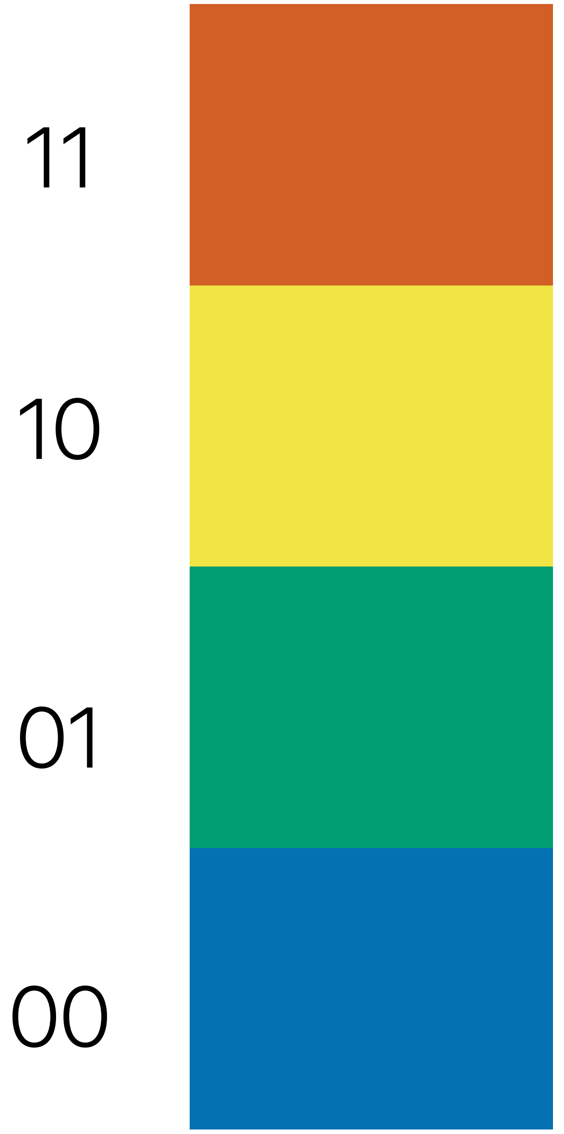
Types of Flash Chips

2 voltage levels, 1-bit | 4 voltage levels, 2-bit | 8 voltage levels, 3-bit | 16 voltage levels, 4-bit

Single-Level Cell (SLC) | **Multi-Level Cell (MLC)** | **Triple-Level Cell (TLC)** | **Quad-Level Cell (QLC)**

Programming in MLC

4 voltage levels,
2-bit

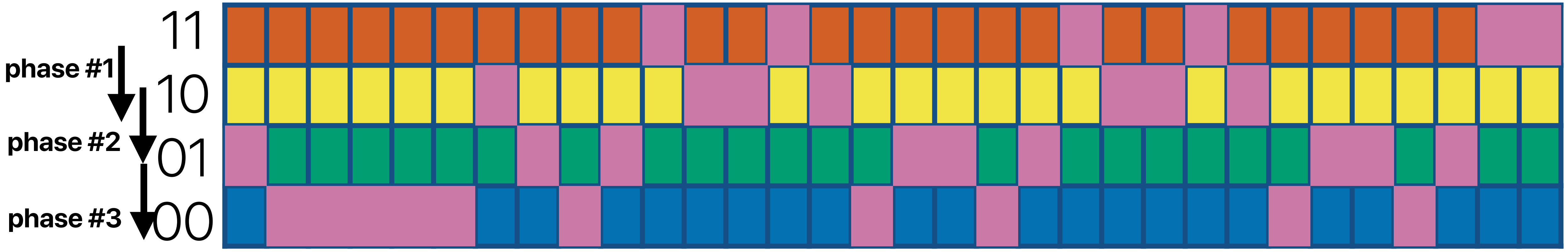


Multi-Level Cell (MLC)

3.14000000000000012434497875802

= 0x40091EB851EB851F

= 01000000 00001001 00011110 10111000 01010001 11101011 10000101 00011111



3 Cycles/Phases to finish programming

Announcement

- Assignment #4 due next Tuesday — **Chapter 4.8-4.9 & 5.2-5.4**
- Lab 5 is up — due next Thursday
  - Start early & plan your time carefully
  - Watch the video and read the instruction BEFORE your session
  - There are links on both course webpage and iLearn lab section
  - Submit through iLearn > Labs
- Check your grades in iLearn

Electrical
Computer Science
Engineering

120A

つづく