Digital Integrated Circuits (83-313)

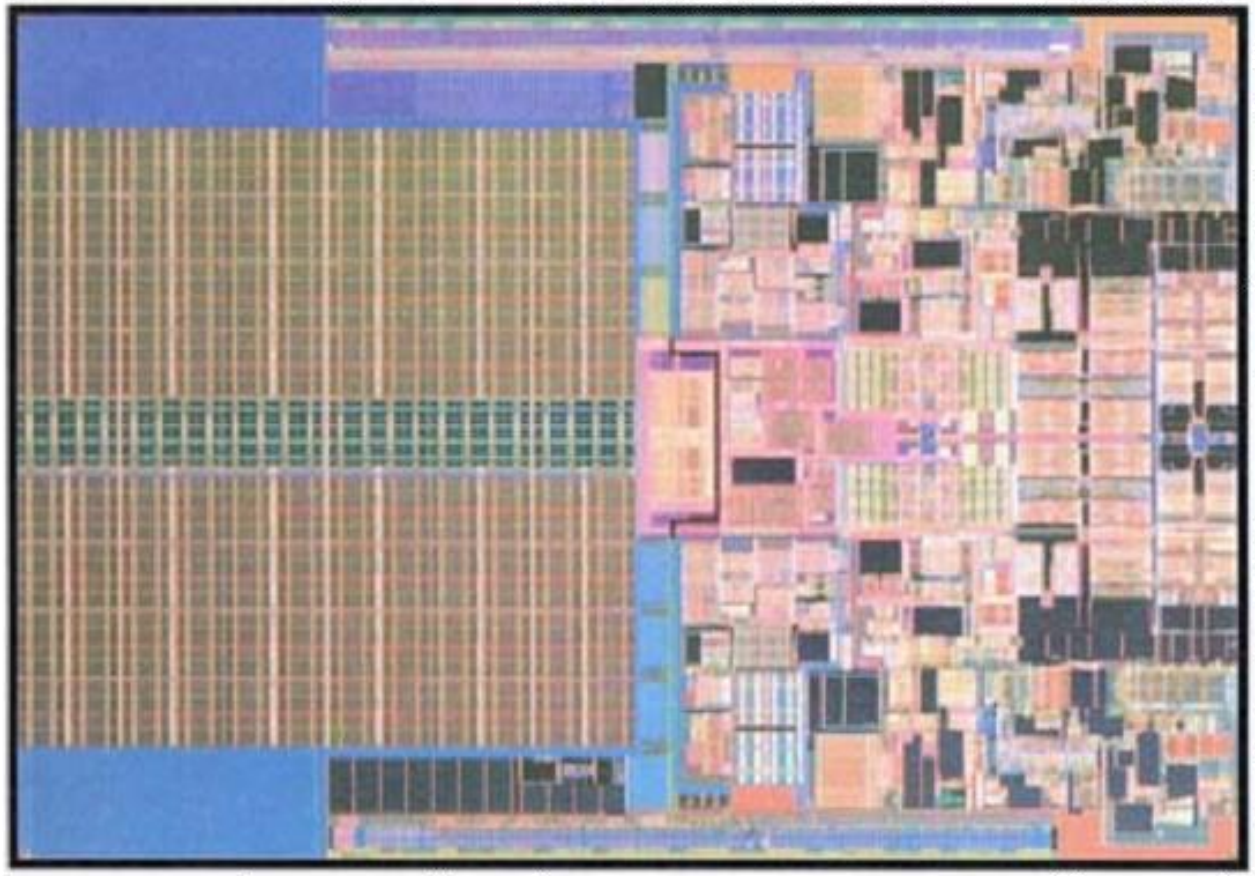
Lecture 6:

SRAM

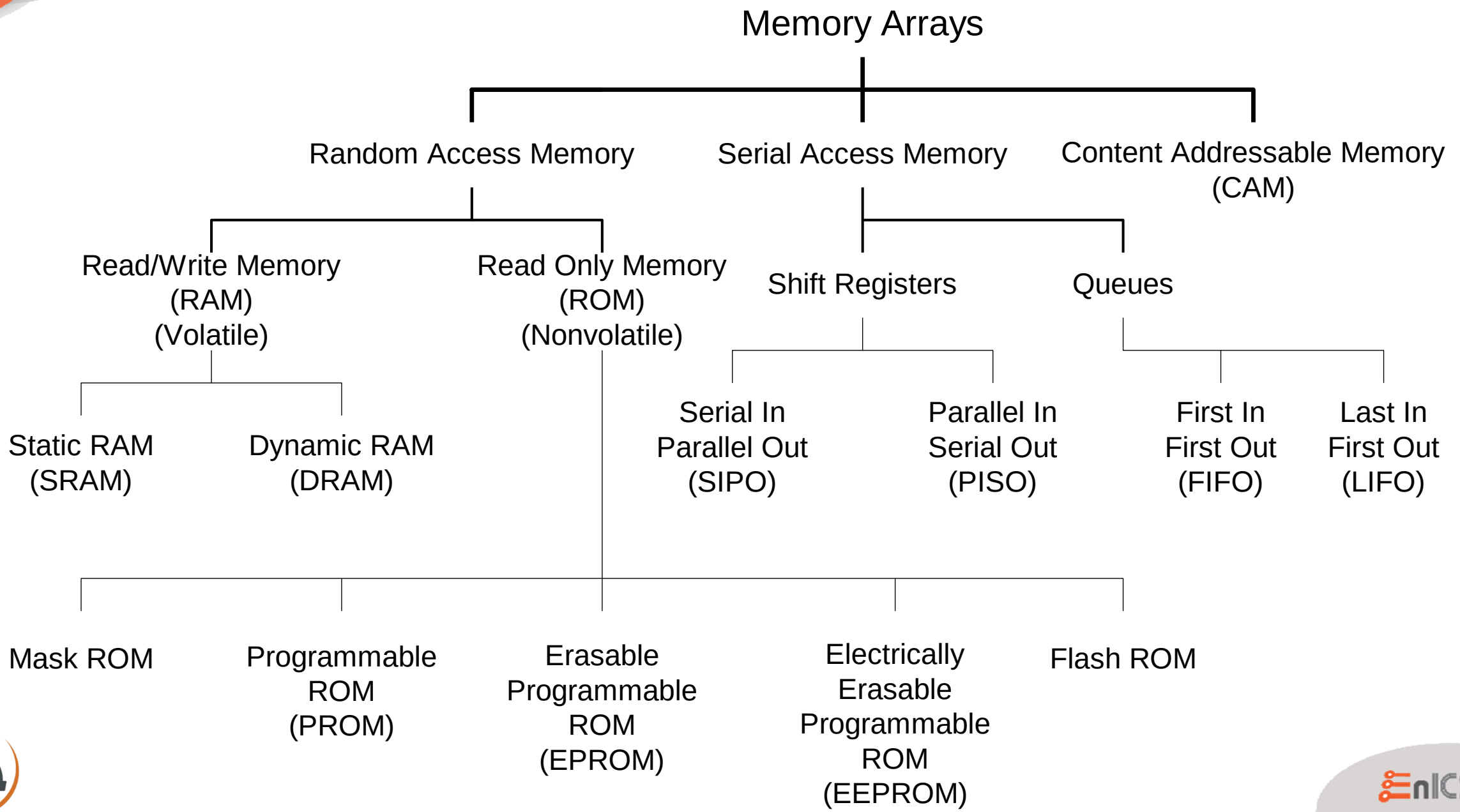
Semester B, 2015-16
Lecturer: Adam Teman
TAs: Itamar Levi, Robert Giterman

Why Memory?

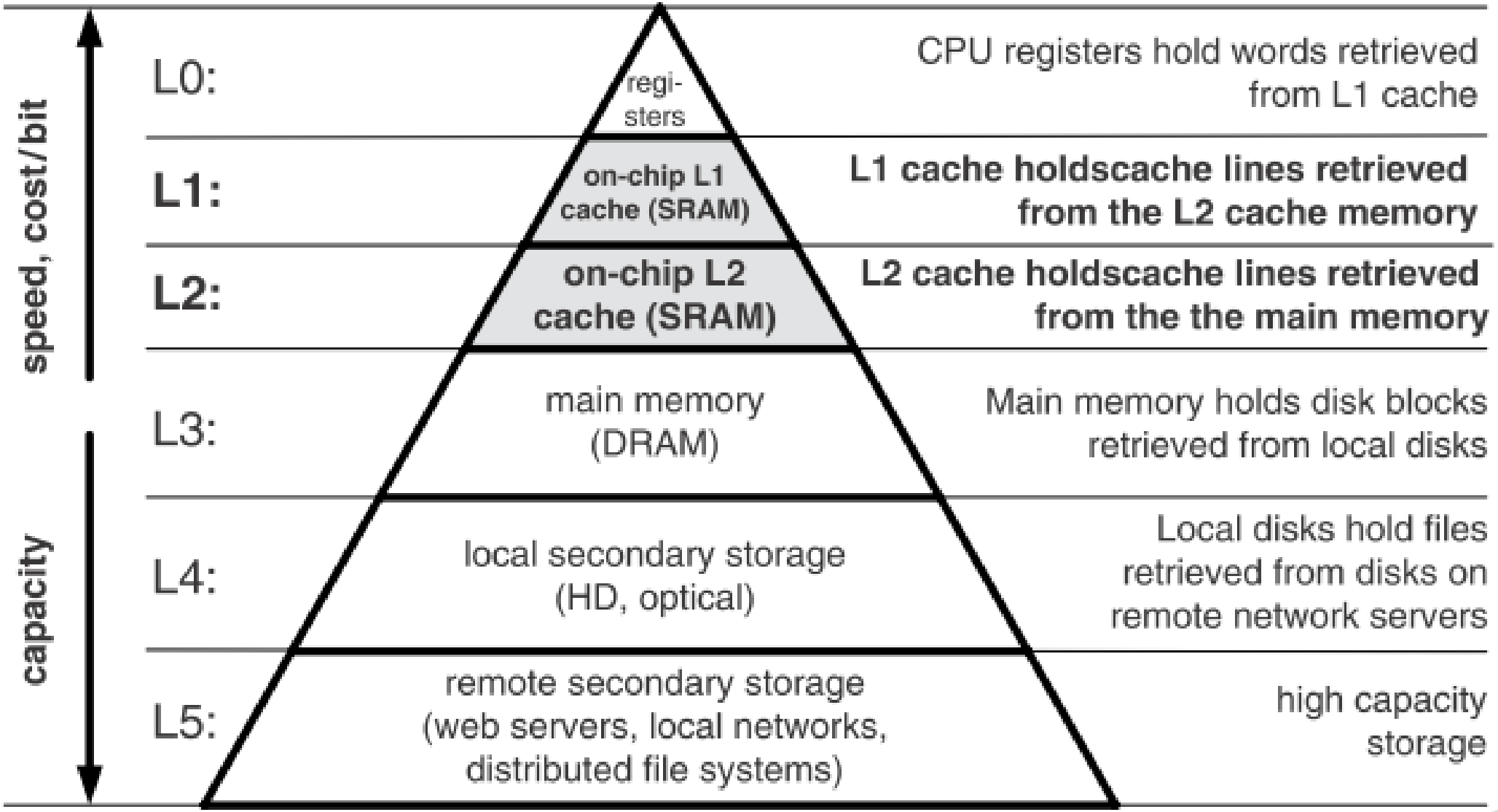
Intel 45nm Core 2



Semiconductor Memory Classification



Memory Hierarchy of a Personal Computer



From Pavlov, Sachdev, 2008

Memory Classification

- Size:
  - Bits, Bytes, Words
- Timing Parameters:
  - read access, write access, cycle time
- Function:
  - Read Only (ROM) – non-volatile
  - Read-Write (RWM) – volatile (SRAM, DRAM)
  - NVRWM – Non-volatile Read Write (EPROM, Flash)
- Access Pattern:
  - Random Access, FIFO, LIFO, Shift Register, CAM
- I/O Architechture:
  - Single Port, Multiport
- Application:
  - Embedded, External, Secondary

1 SRAM Memory Architecture

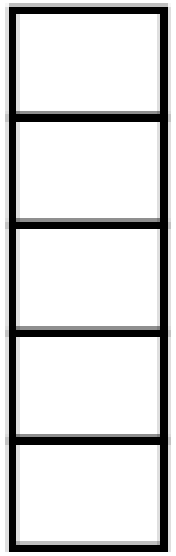
2 The 6T SRAM Bitcell

3 SRAM Stability

4 SNM Calculation

5 Peripheral Circuits

Random Access Chip Architecture

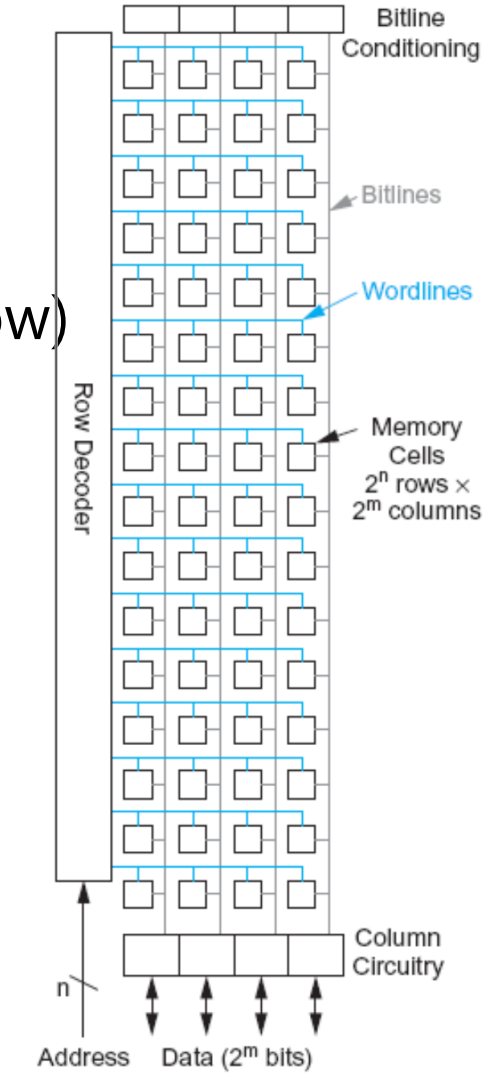


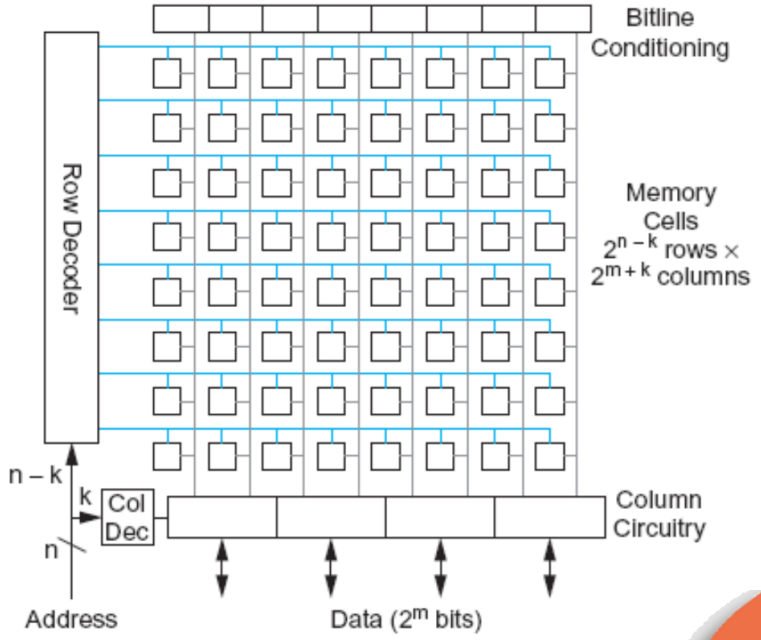
- Conceptual: linear array
  - Each box holds some data
  - But this leads to a long and skinny shape

- Let's say we want to make a 1MB memory:
  - 1MB=2^{20} words X 8 bits=2^{23} bits, each word in a separate row
  - A decoder would reduce the number of access pins from 2^{20} access pins to 20 *address* lines.
  - We'd fit the pitch of the decoder to the word cells, so we'd have Word Lines with no area overhead.
  - The output lines (=bit lines) would be extremely long, as would the delay of the huge decoder.
  - The array's height is about 128,000 times larger than its width ($2^{20}/2^3$).

Square Ratio

- Instead, let's make the array square:
  - 1MB=2^{23} bits=2^{12} rows X 2^{11} columns.
  - There are 4000 rows, so we need a 12-bit row address decoder (to select a single row)
  - There are 2000 columns, representing 256 8-bit words.
  - We need to select only one of the 256 words through a column address decoder (or multiplexer).
  - We call the row lines "*Word Lines*" and the column lines "*Bit Lines*".

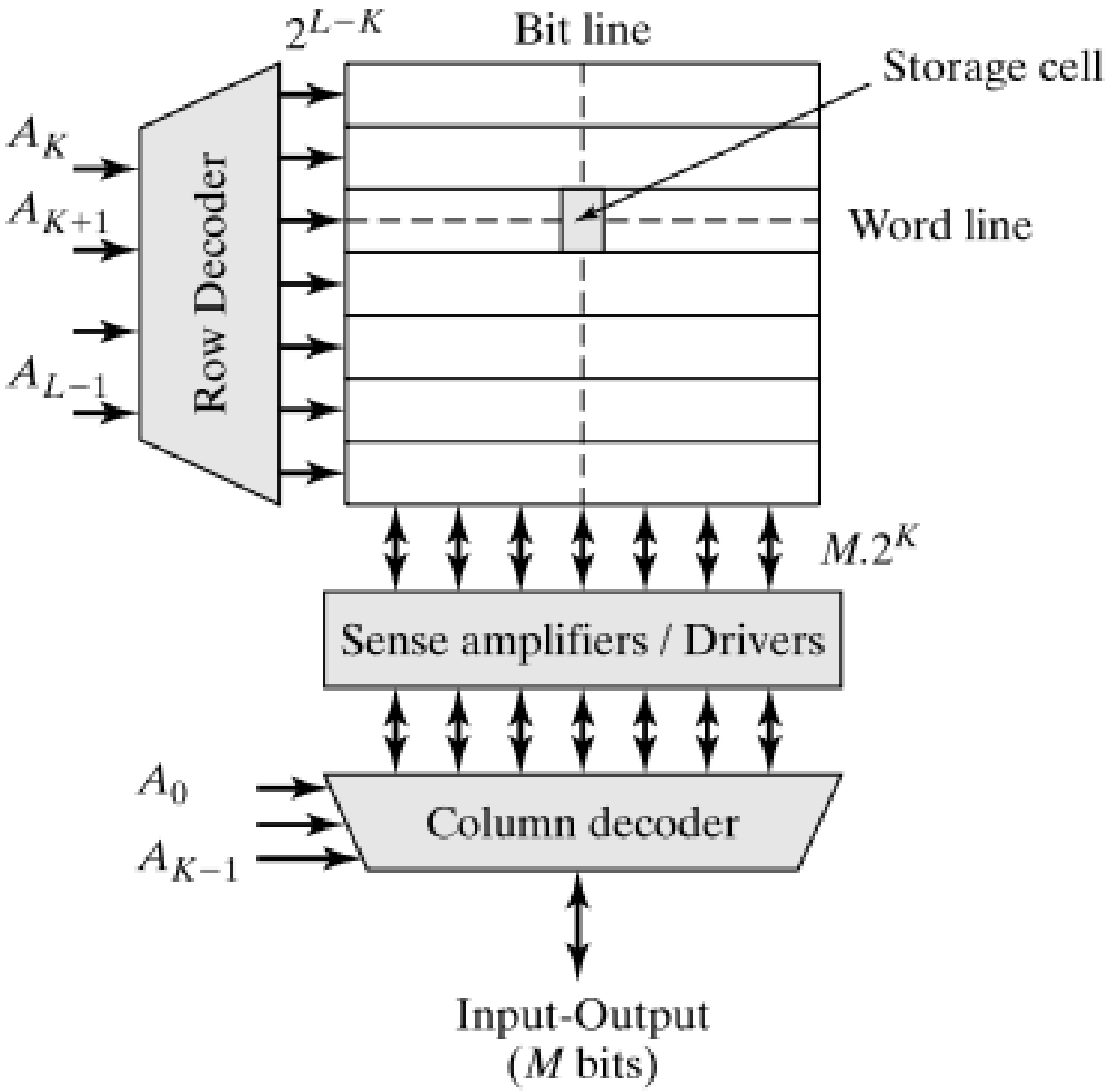




Special Considerations

- The “core” of the memory array is huge.
  It can sometimes take up most of the chip area.
  - For this reason, we will try to
    make the “bitcell” as small as possible.
  - A standard Flip Flop uses at least
    10 transistors per bit. This is very area consuming.
- We will trade-off
  area for other circuit properties:
  - Noise Margins
  - Logic Swing
  - Speed
  - Design Rules
- This requires special peripheral circuitry.

Memory Architecture



Memory Size: *L* Words of *M* bits
$=2^M x L$ bits

Multiplexing Factor: *K*

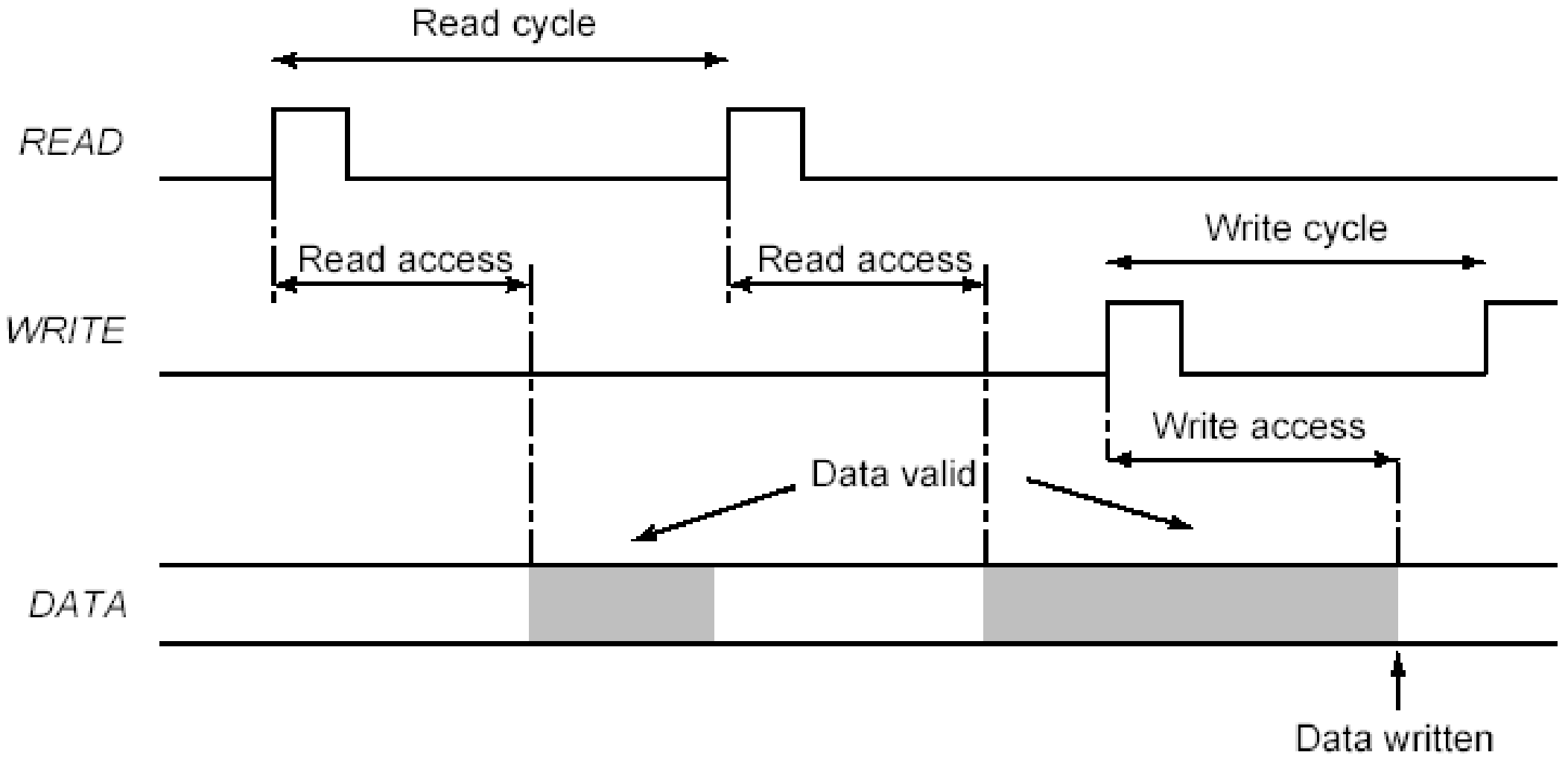
Number of Rows: 2^{L-K}

Number of Columns: $Mx2^K$

Row Decoder: $L-K \rightarrow 2^{L-K}$

Column Decoder: $K \rightarrow 2^K$

Memory Timing: Definitions



1 SRAM Memory Architecture

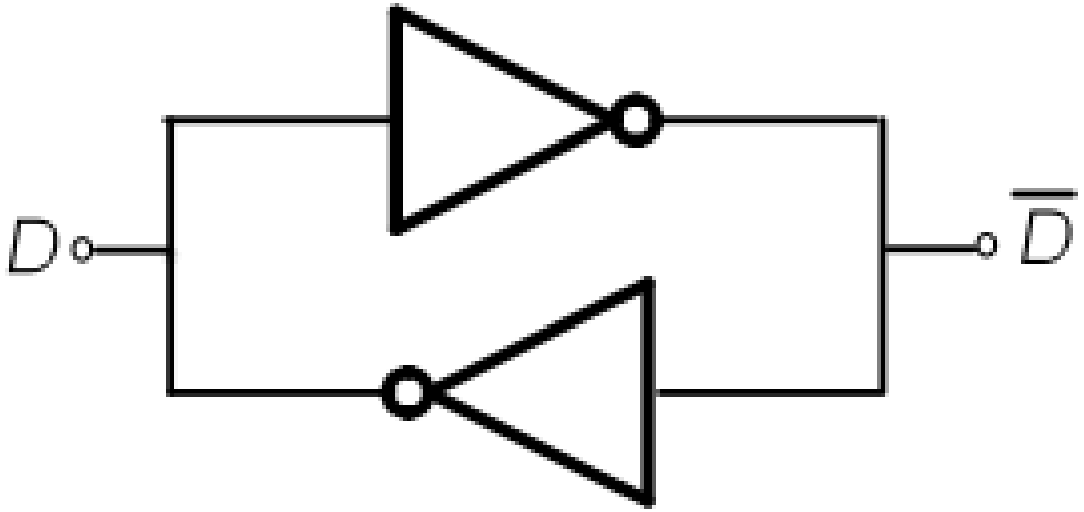
2 **The 6T SRAM Bitcell**

3 SRAM Stability

4 SNM Calculation

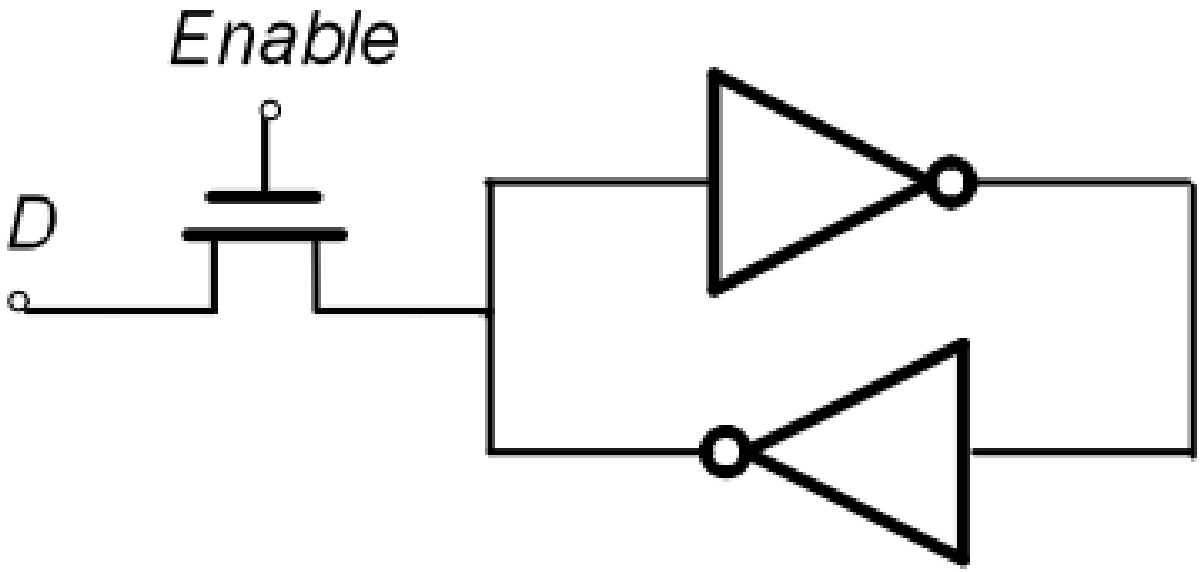
5 Peripheral Circuits

Basic Static Memory Element



Positive Feedback: Bi-Stability

Writing into a Cross-Coupled Pair



Access transistor must be able to overpower the feedback

How should we write a ‘1’

Option 1: nMOS Access Transistor

Passes a “weak ‘1’”, bad at pulling up against the feedback

Option 2: pMOS Access Transistor

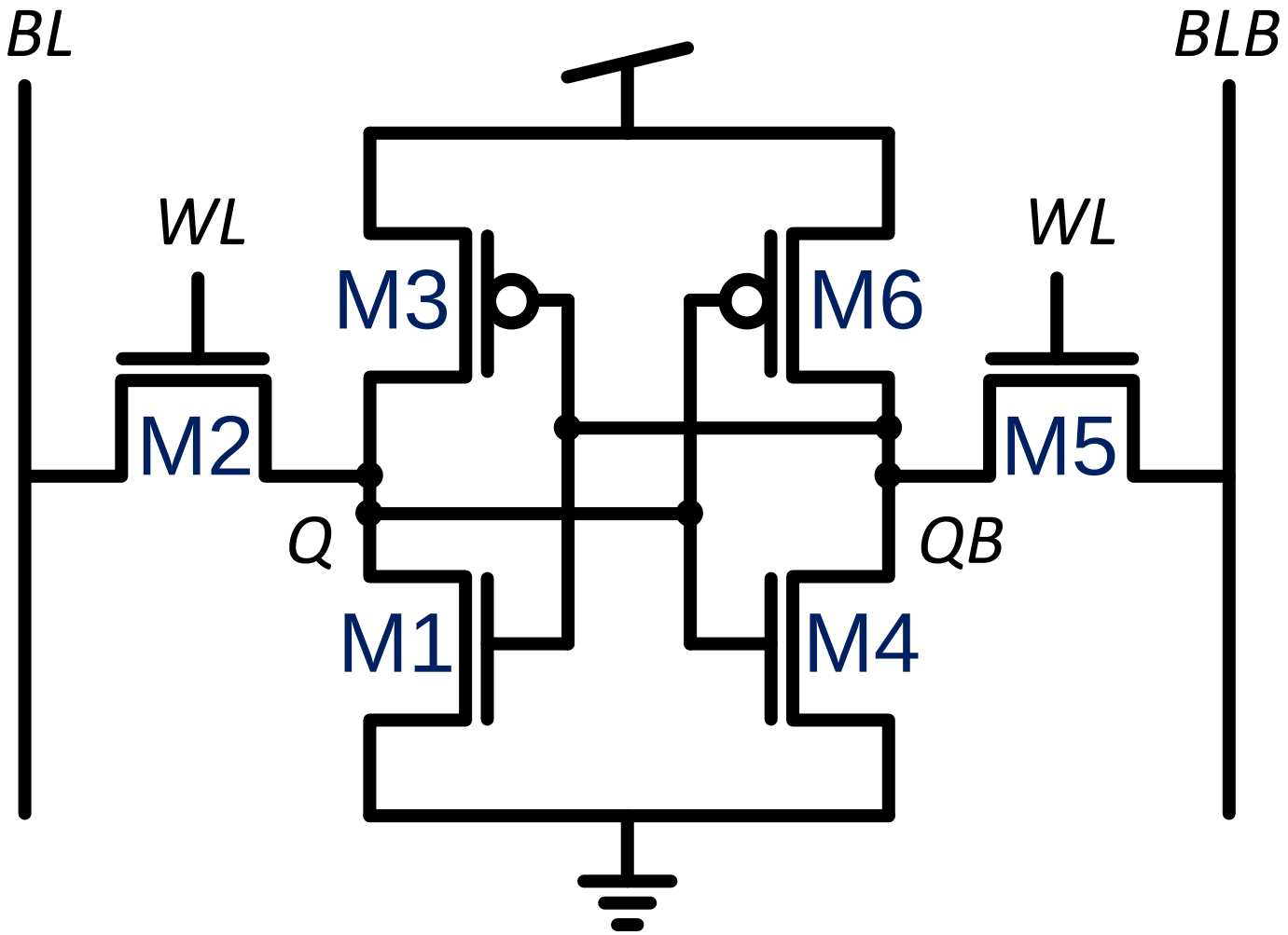
Passes a “weak ‘0’”, bad at pulling down against the feedback

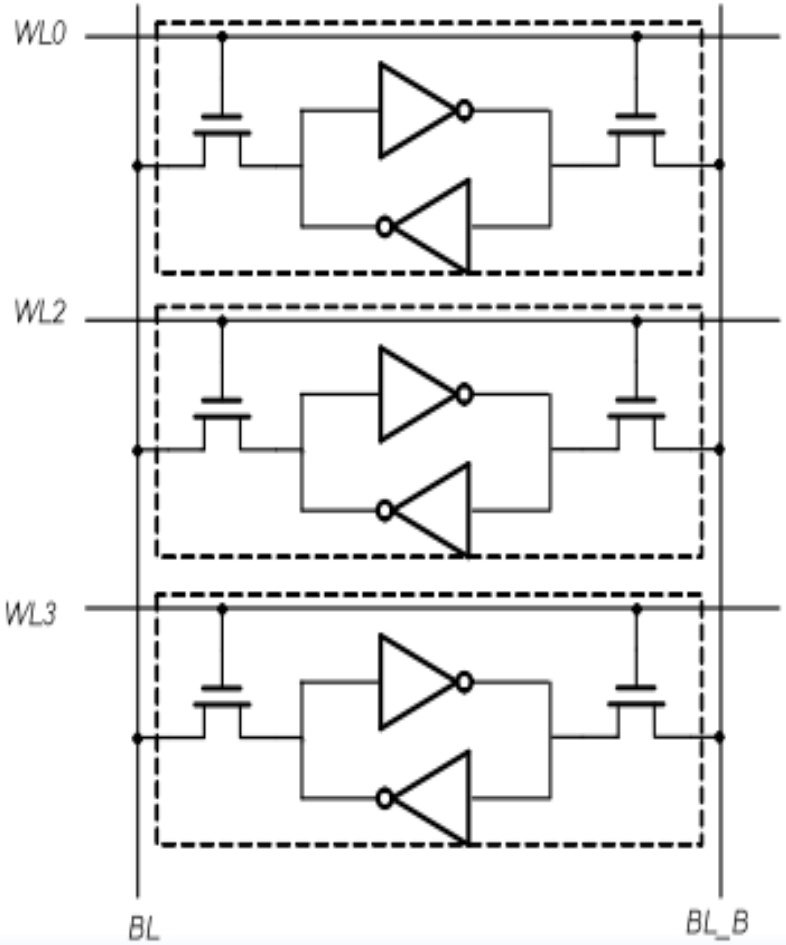
Option 3: Transmission Gate

Writes well, but how do we read?

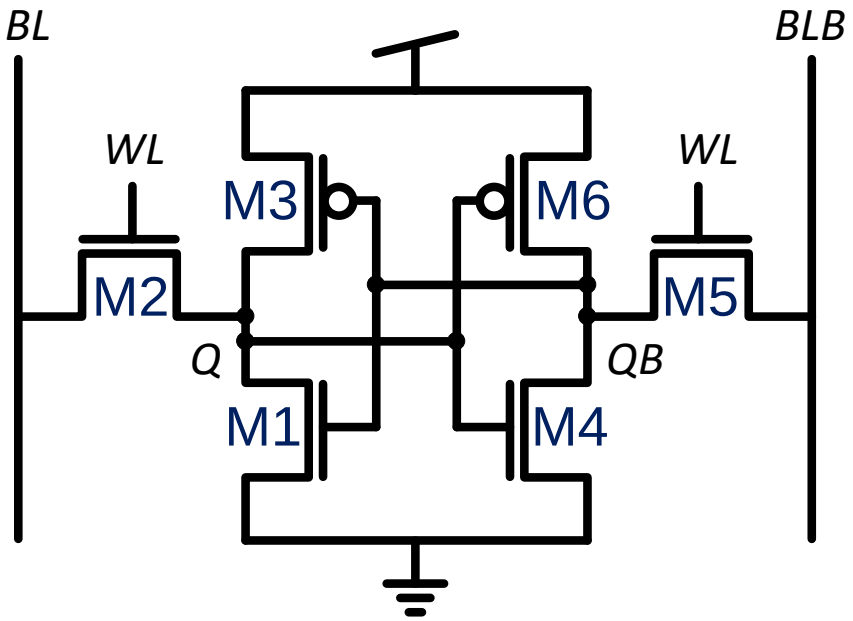
Solution: Differential nMOS Write

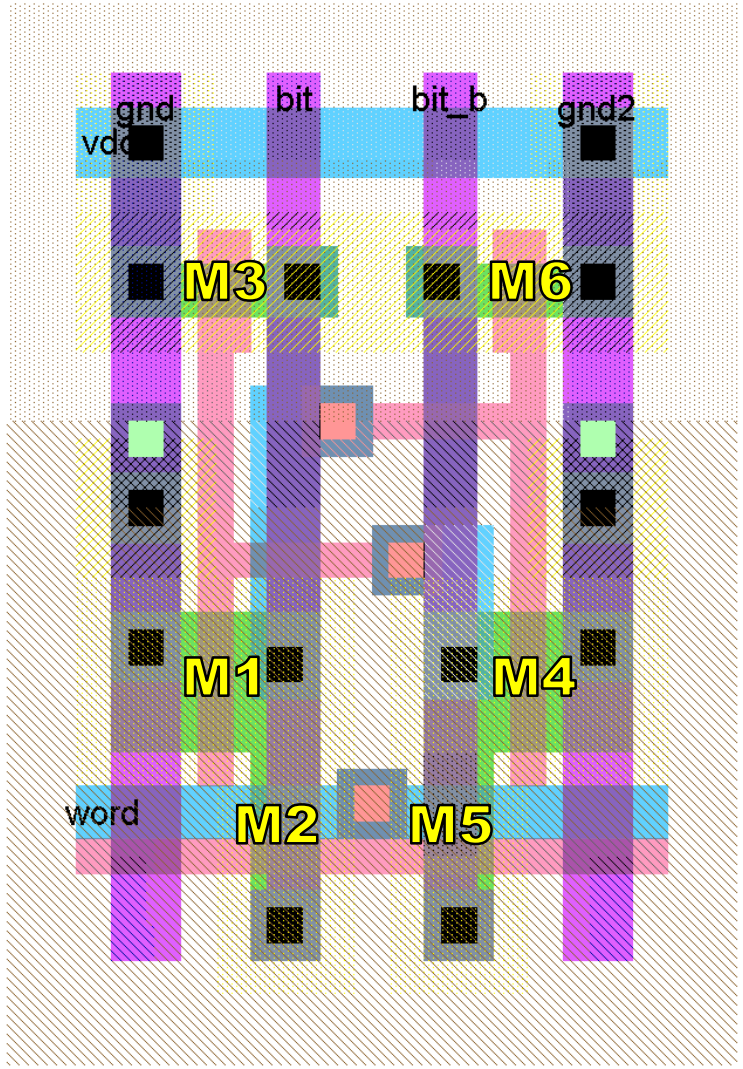
6-transistor CMOS SRAM Cell





SRAM Layout - Traditional

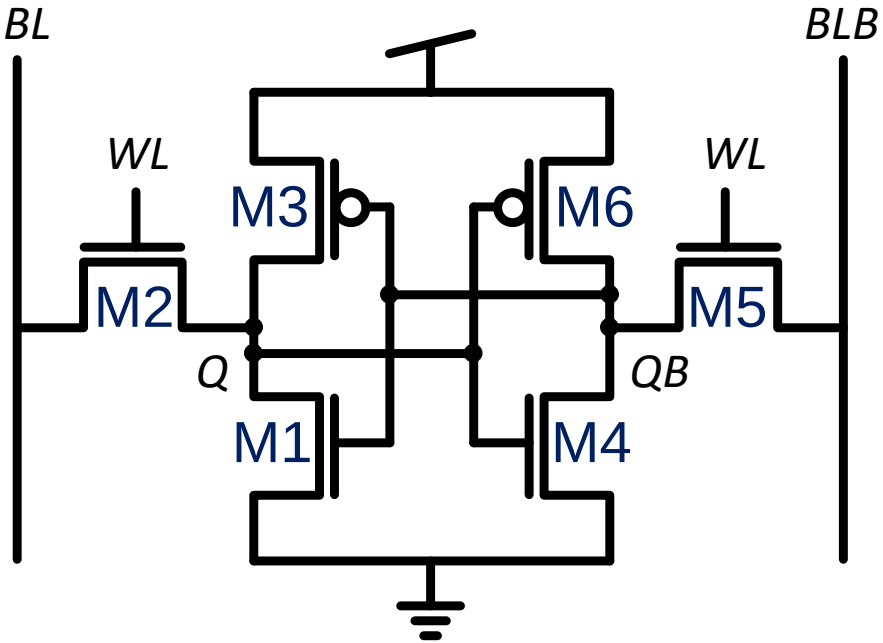


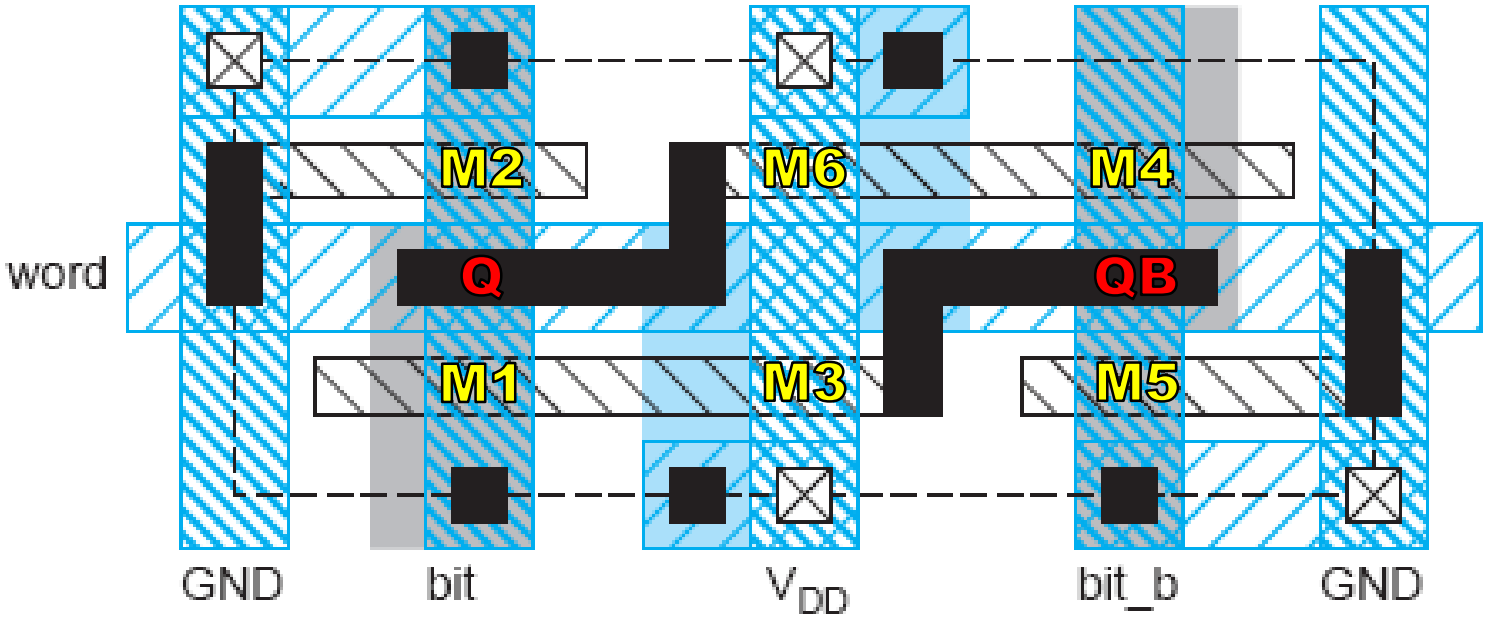


- Share Horizontal Routing (WWL).
- Share Vertical Routing (BL, BLB).
- Share Power and Ground.

SRAM Layout – Thin Cell

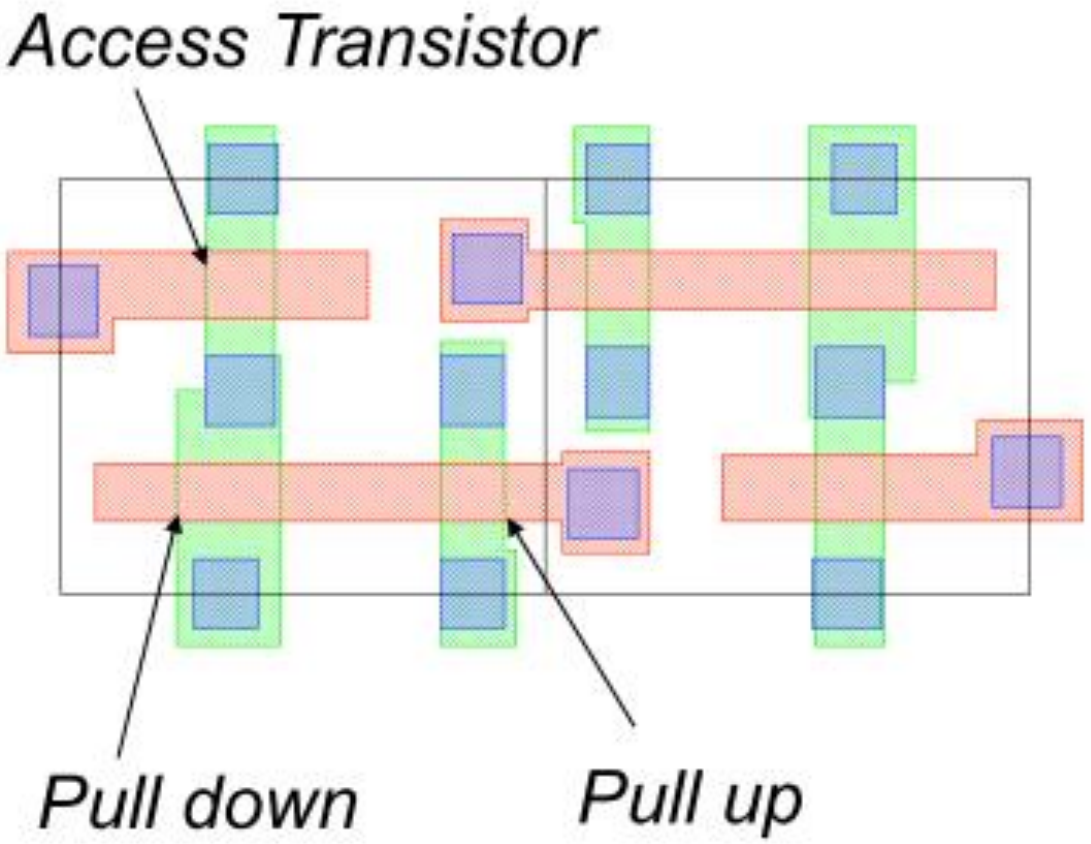
- Avoid Bends in Polysilicon and Diffusion
- Orient all transistors in one direction.
- Minimize Bitline Capacitance.

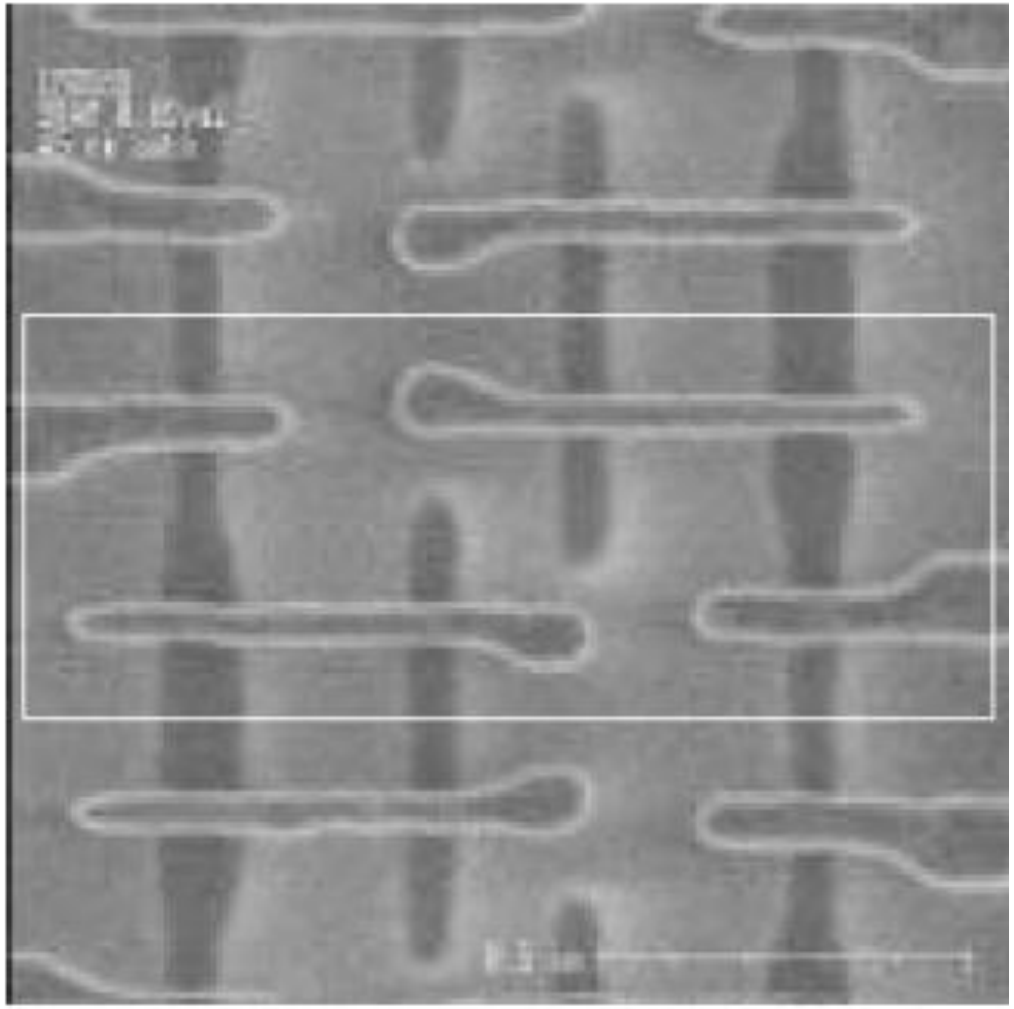




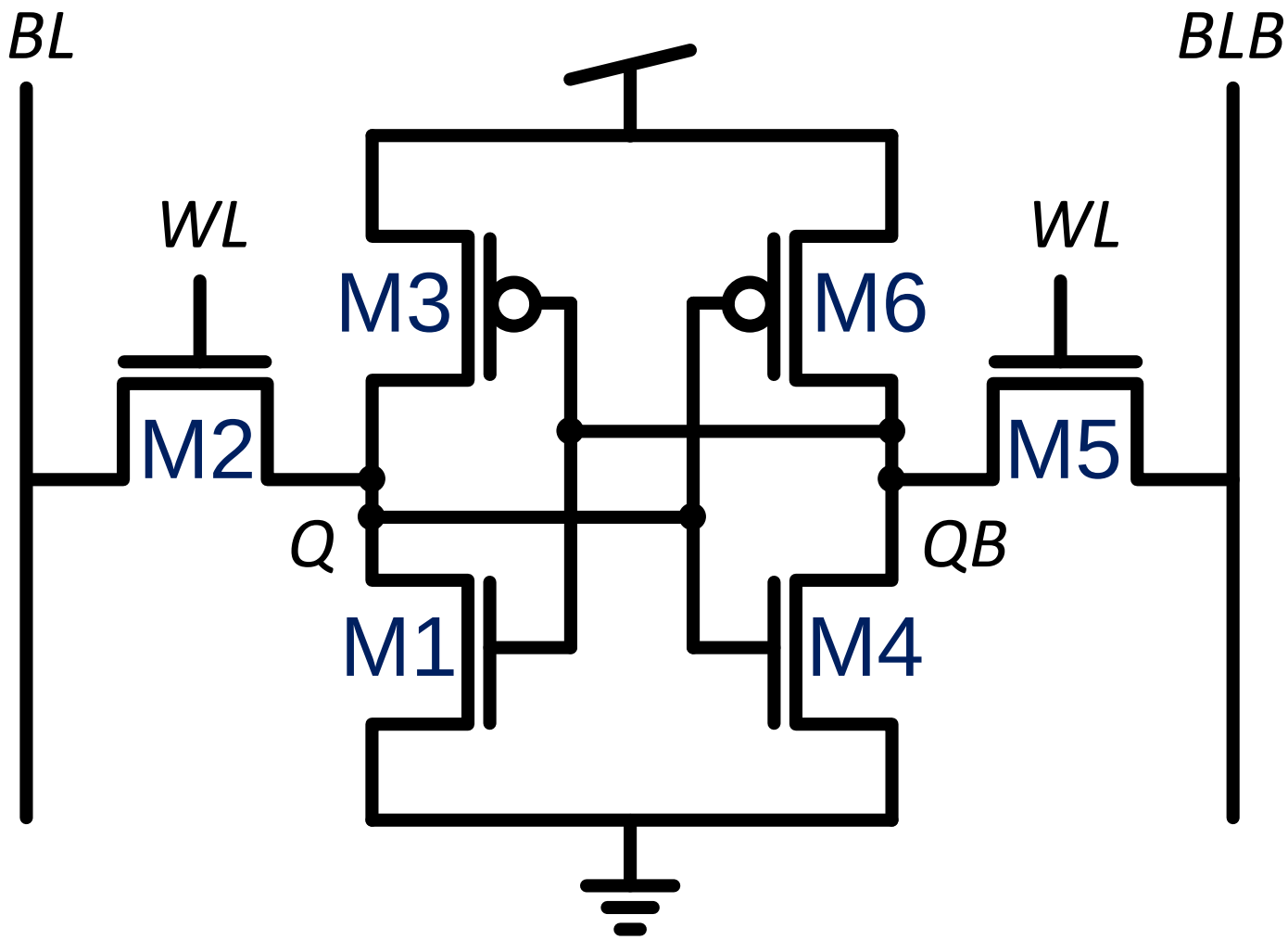
65nm SRAM

- ST/Philips/Motorola

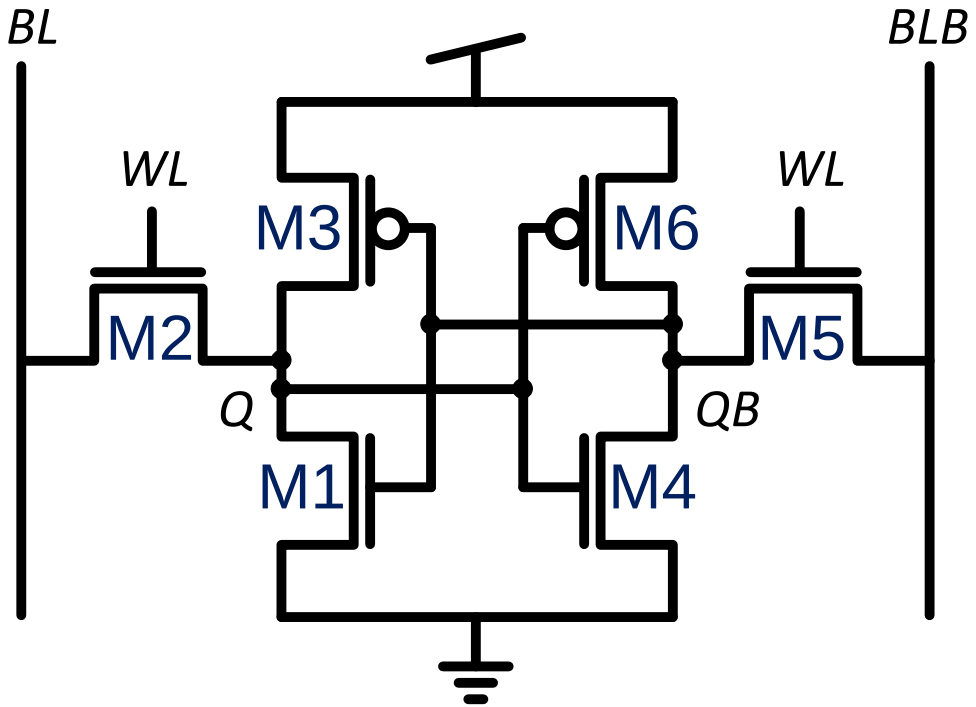


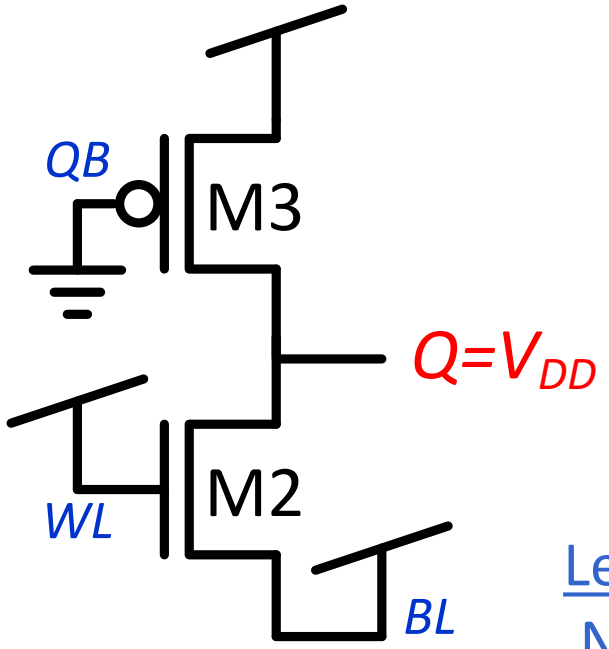


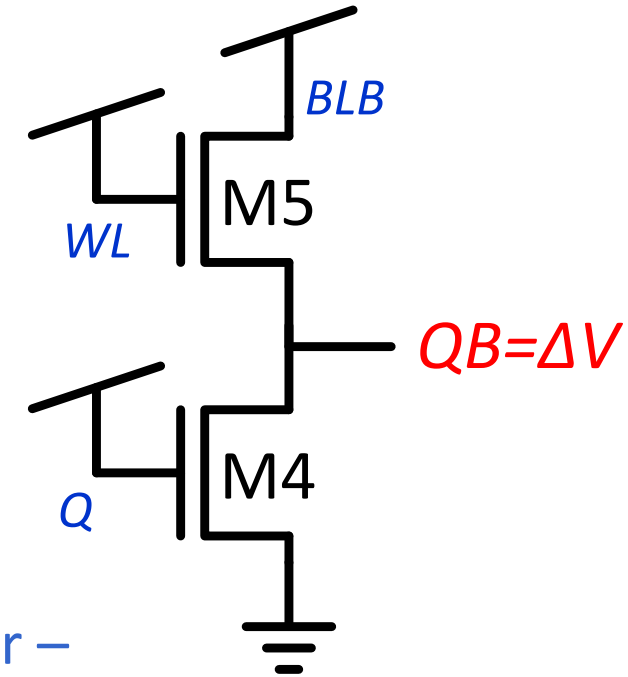
SRAM Operation: HOLD



SRAM Operation: READ



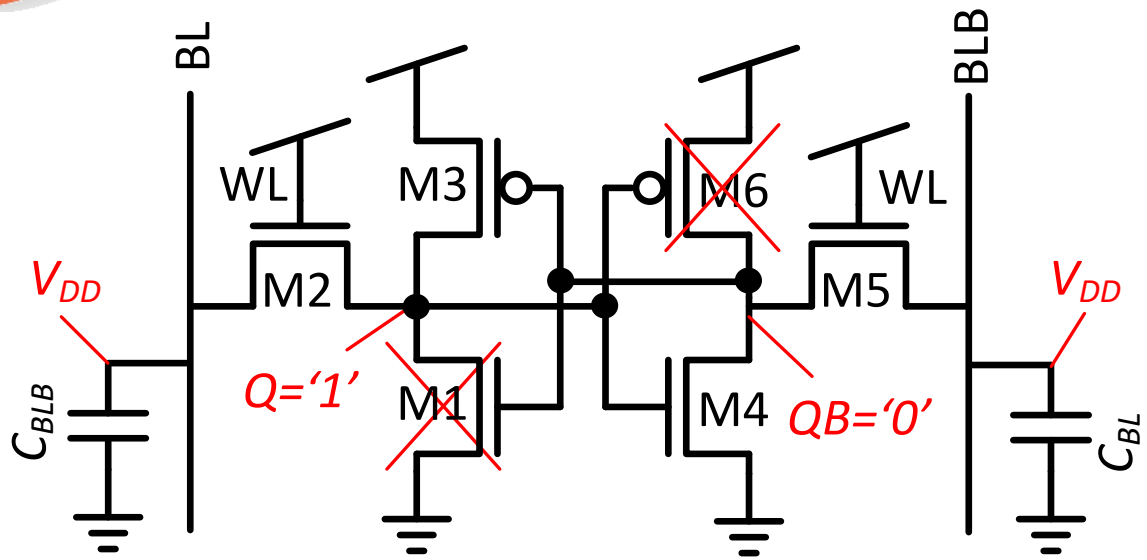


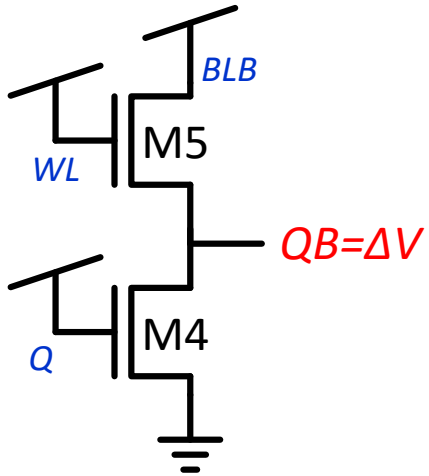


Left Side:
Nothing
Changes…

Right Side:
"nMOS" inverter –
QB voltage rises

SRAM Operation - Read





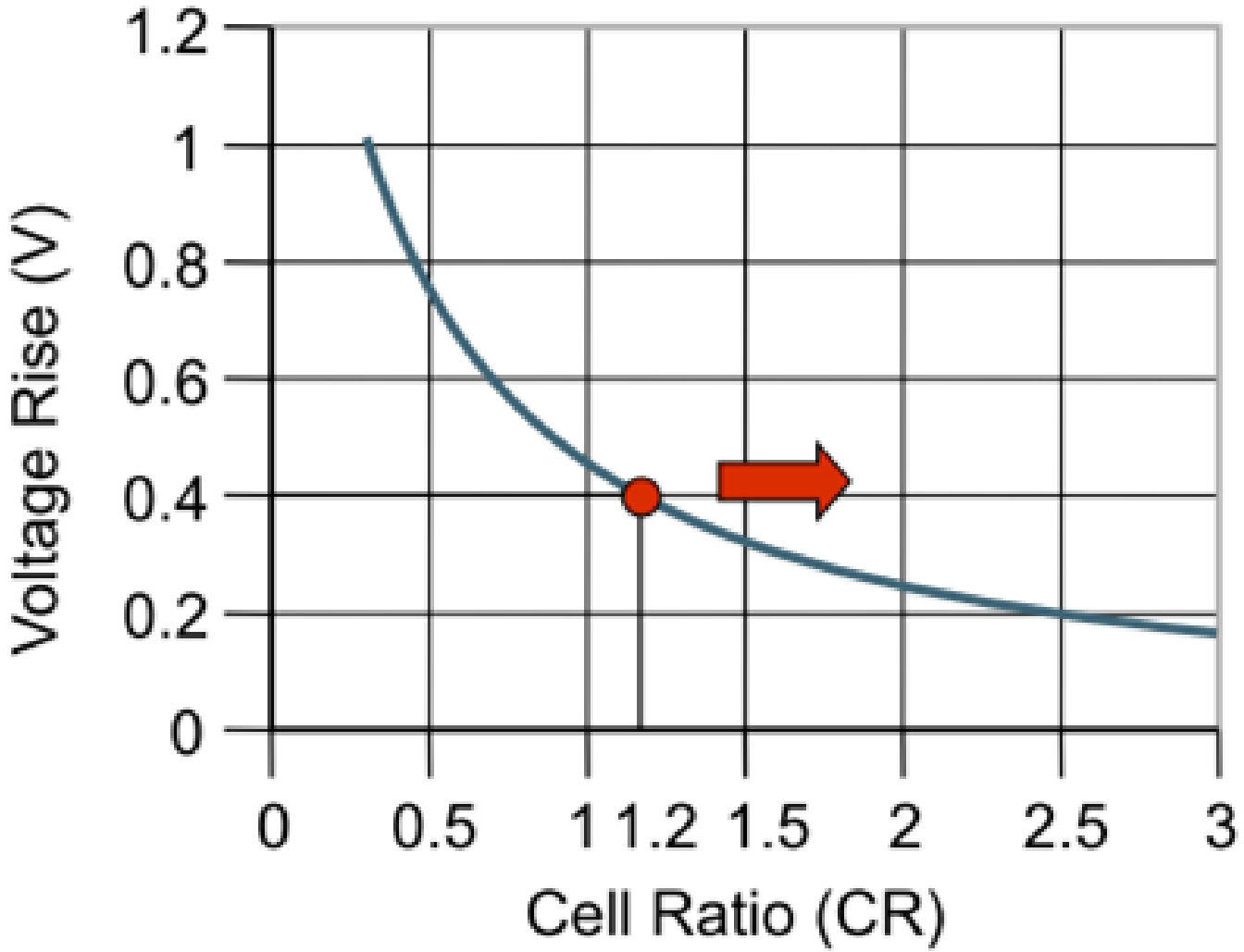
Cell Ratio:

$$CR \equiv \frac{W_4/L_4}{W_5/L_5}$$

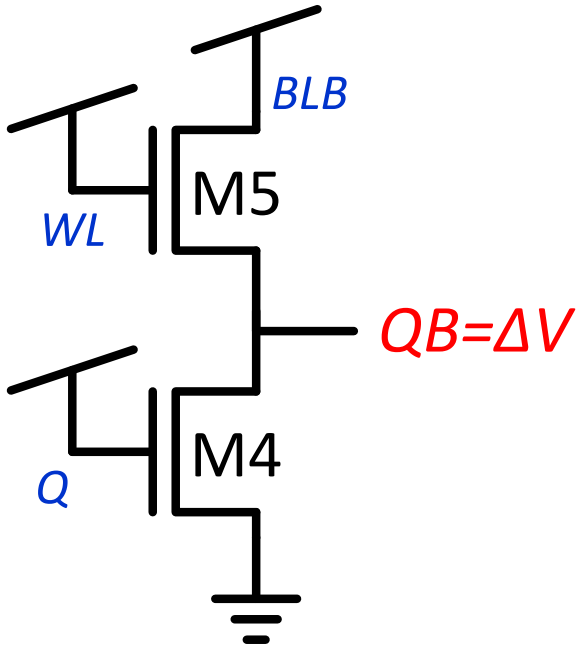
$$k_{M5}\left[\left(V_{DD}-\Delta V-V_{T,n}\right)V_{DSat,n}-\frac{V_{DSat,n}^2}{2}\right]=k_{M4}\left[\left(V_{DD}-V_{T,n}\right)\Delta V-\frac{\Delta V^2}{2}\right]$$

$$\Delta V=\frac{V_{DSat,n}+CR\left(V_{DD}-V_{T,n}\right)-\sqrt{V_{DSat,n}^2\left(1+CR\right)+CR^2\left(V_{DD}-V_{T,n}\right)^2}}{CR}$$

Cell Ratio (Read Constraint)

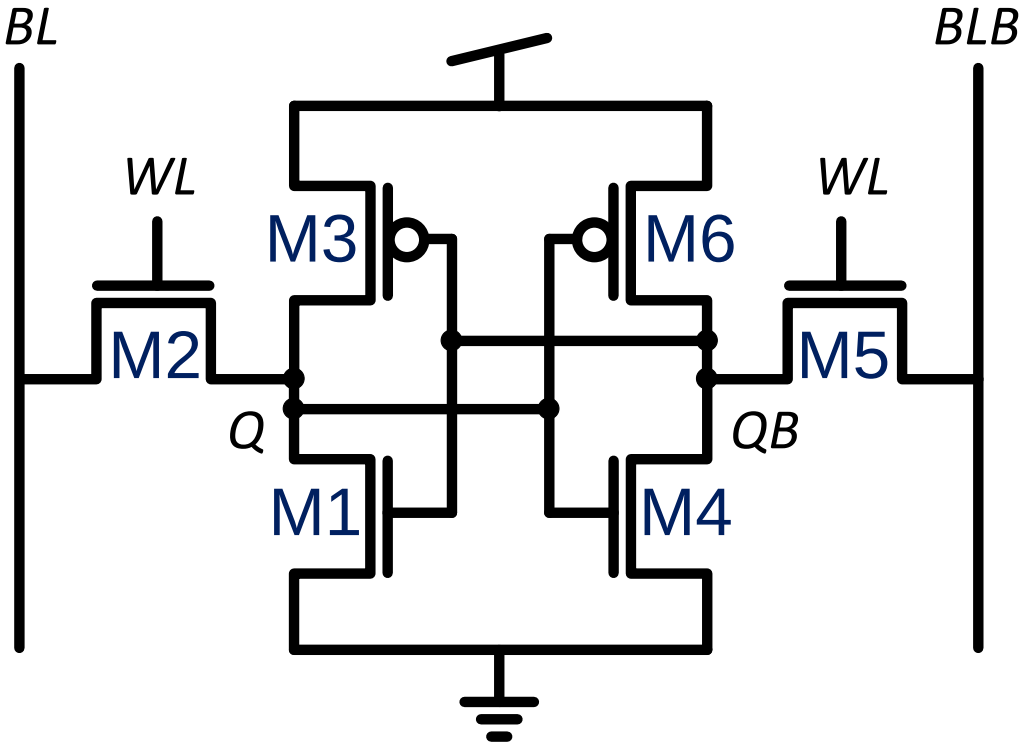


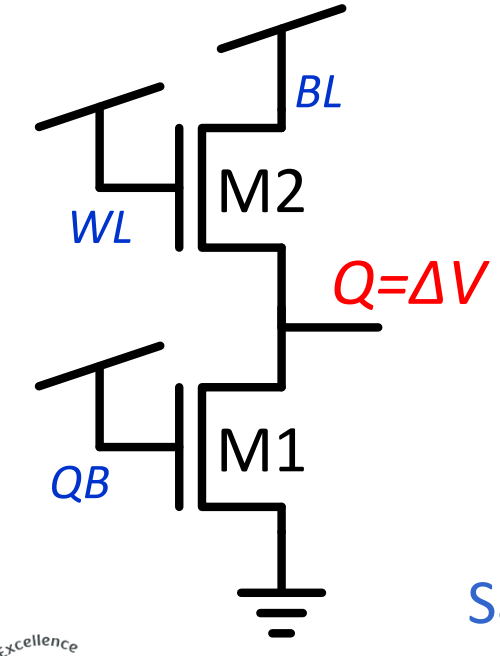
$$CR \equiv \frac{W_4/L_4}{W_5/L_5}$$

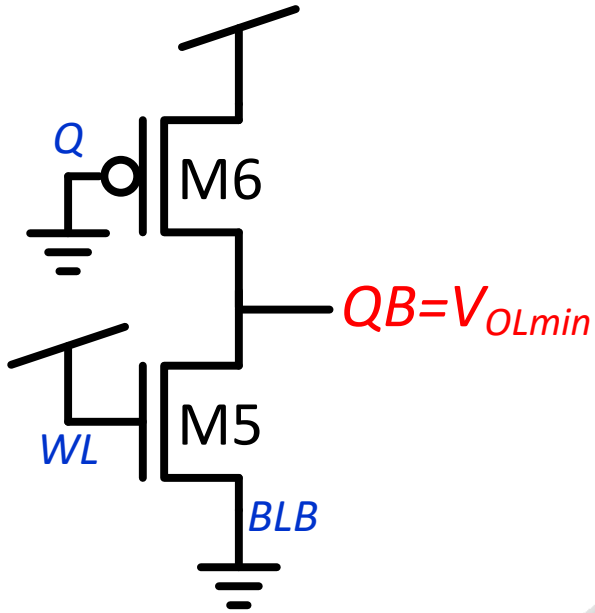


So we need the pull down transistor to be much stronger than the access transistor…

SRAM Operation: WRITE



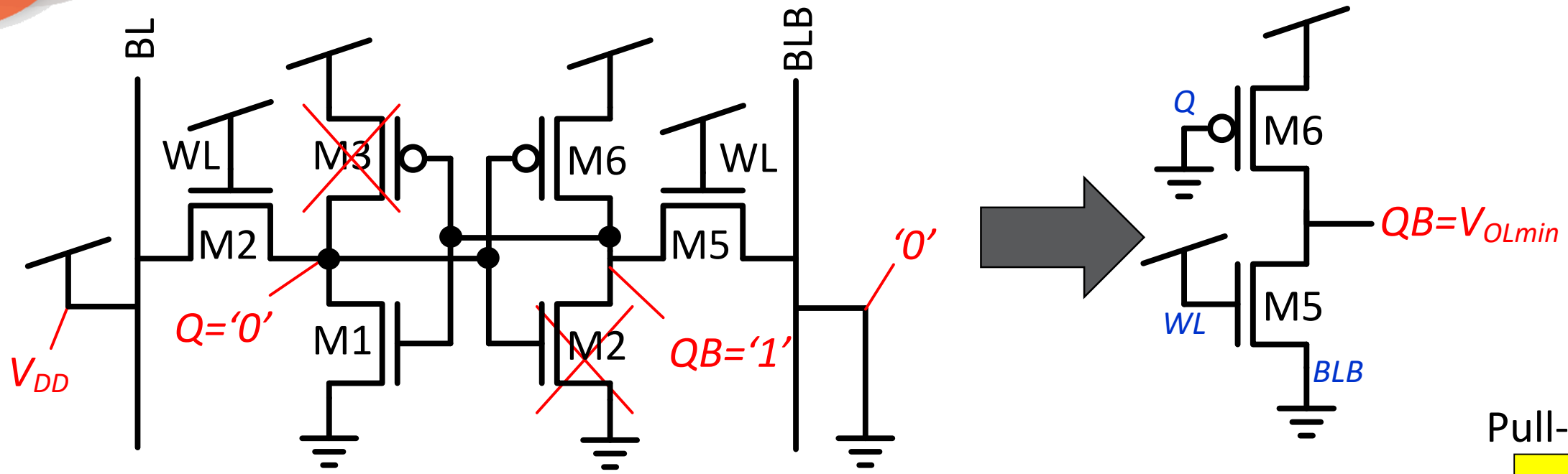




Left Side:
Same as during read – designed so $\Delta V < V_M$

Right Side:
Pseudo nMOS inverter!

SRAM Operation - Write



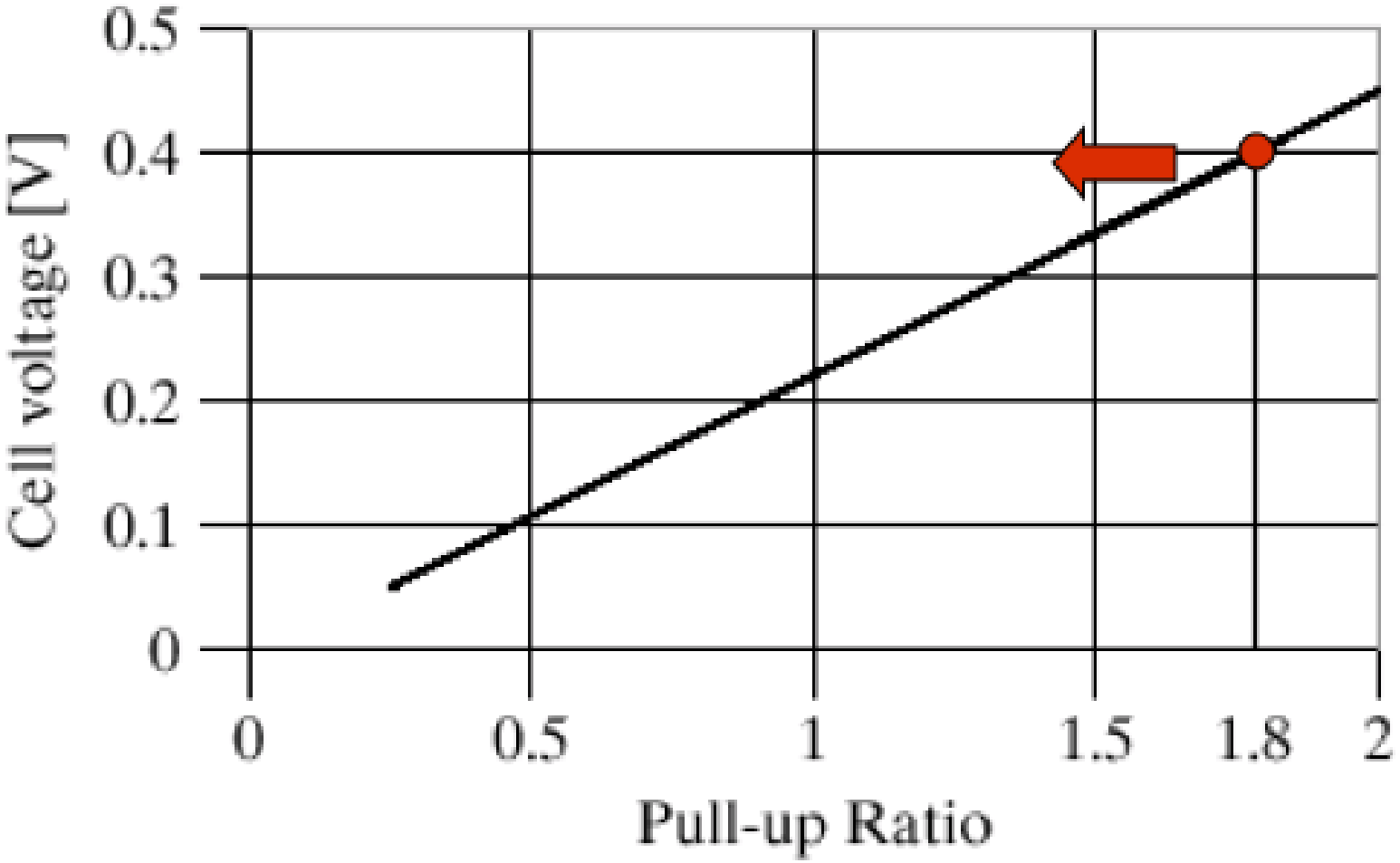
Pull-Up Ratio

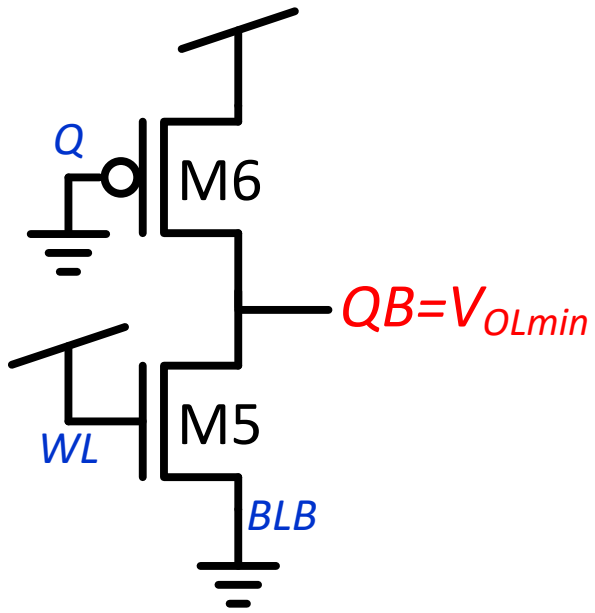
$$PR \equiv \frac{W_6/L_6}{W_5/L_5}$$

$$k_{M6}\left[\left(V_{DD}-\left|V_{T,p}\right|\right)V_{DSat,p}-\frac{V_{DSat,p}^2}{2}\right]=k_{M5}\left[\left(V_{DD}-V_{T,n}\right)V_{QB}-\frac{\Delta V_{QB}^2}{2}\right]$$

$$V_{QB}=V_{DD}-V_{T,n}-\sqrt{\left(V_{DD}-V_{T,n}\right)^2-2\frac{\mu_p}{\mu_n}PR\left[\left(V_{DD}-\left|V_{T,p}\right|\right)V_{DSat,p}-\frac{V_{DSat,p}^2}{2}\right]}$$

Pull Up Ratio – Write Constraint



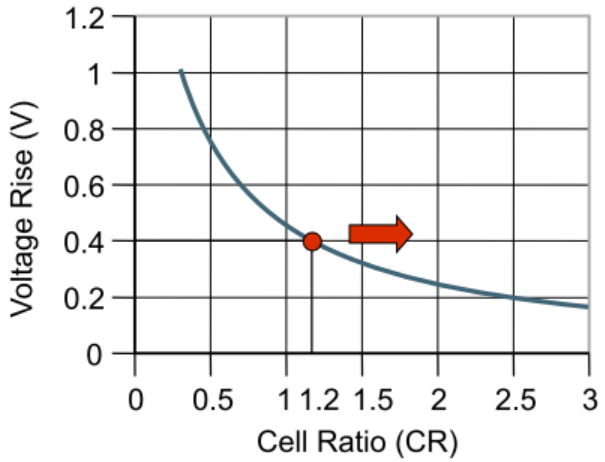


So we need the access transistor to be much stronger than the pull up transistor…

$$PR \equiv \frac{W_6/L_6}{W_5/L_5}$$

Summary – SRAM Sizing Constraints

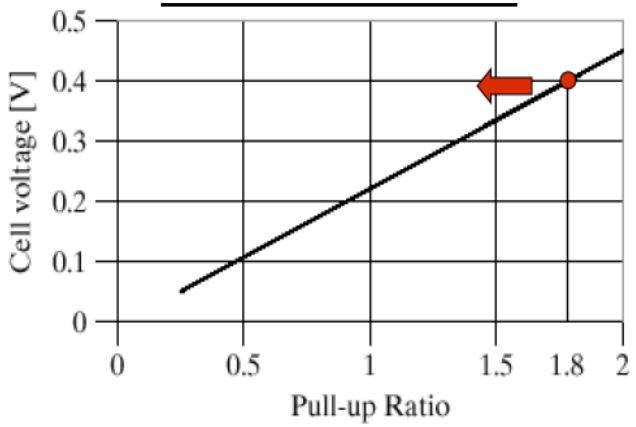
Read Constraint



$$CR \equiv \frac{W_1/L_1}{W_2/L_2} = \frac{W_4/L_4}{W_5/L_5} = \frac{PDN}{access}$$

$$K_{PDN} > K_{access}$$

Write Constraint

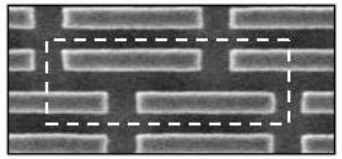


$$PR \equiv \frac{W_3/L_3}{W_2/L_2} = \frac{W_6/L_6}{W_5/L_5} = \frac{PUN}{access}$$

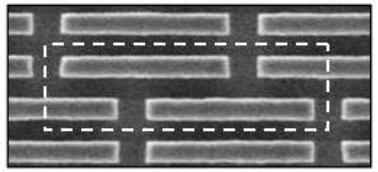
$$K_{access} > K_{PUN}$$

$$K_{PDN} > K_{access} > K_{PUN}$$

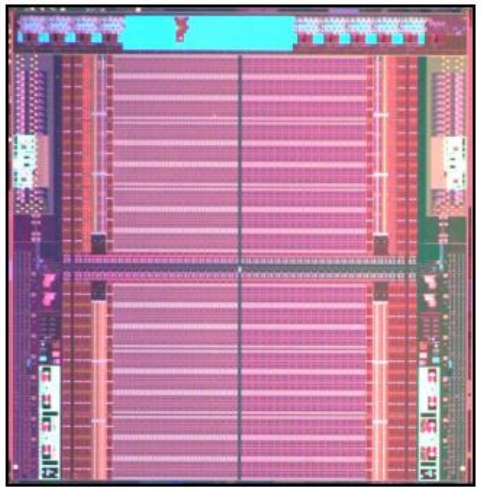
Commercial SRAMs



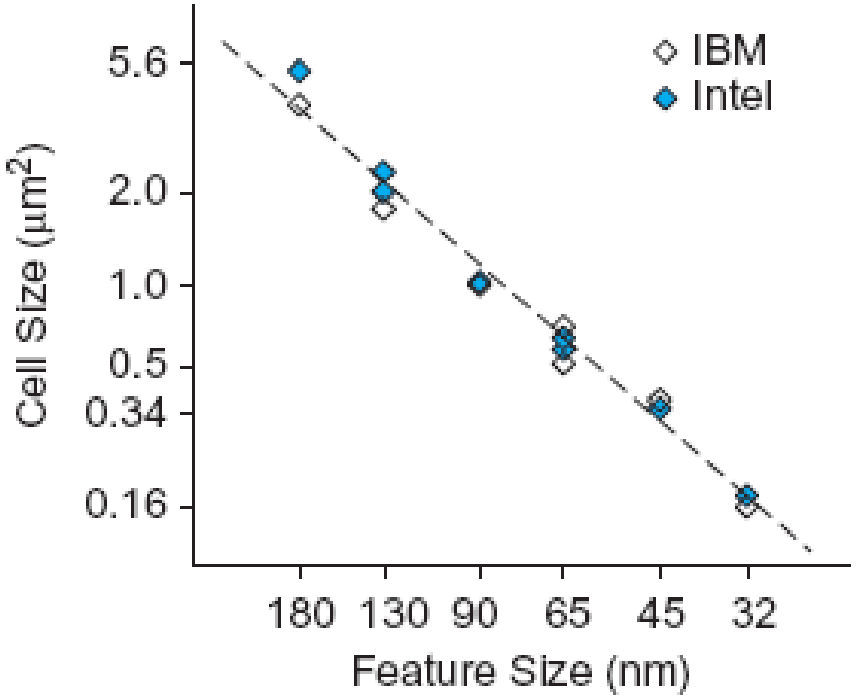
0.092 um^2 SRAM cell
for high density applications

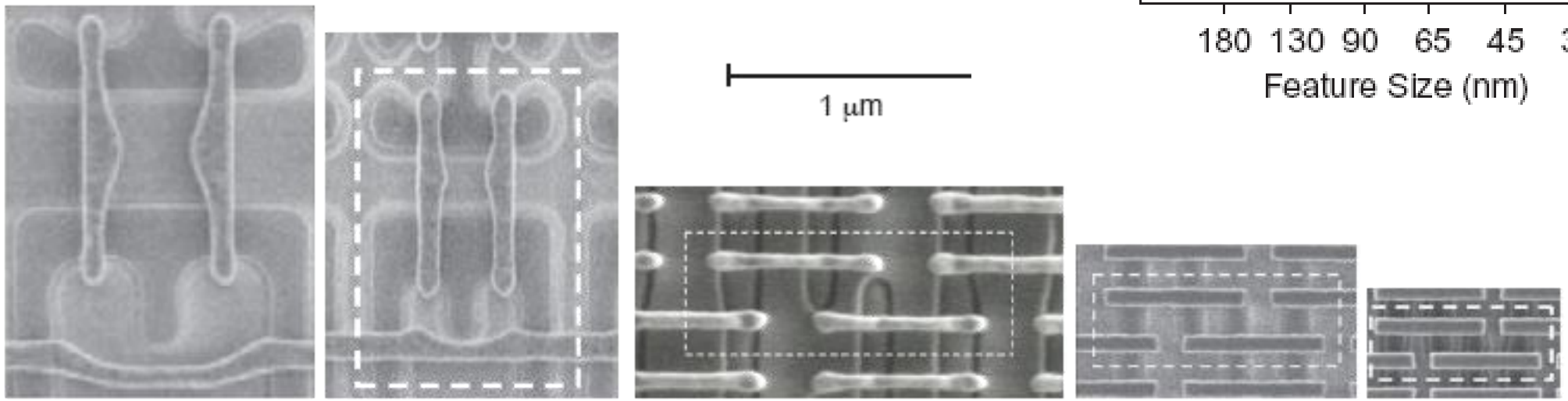


0.108 um^2 SRAM cell
for low voltage applications



Intel Design Forum 2009

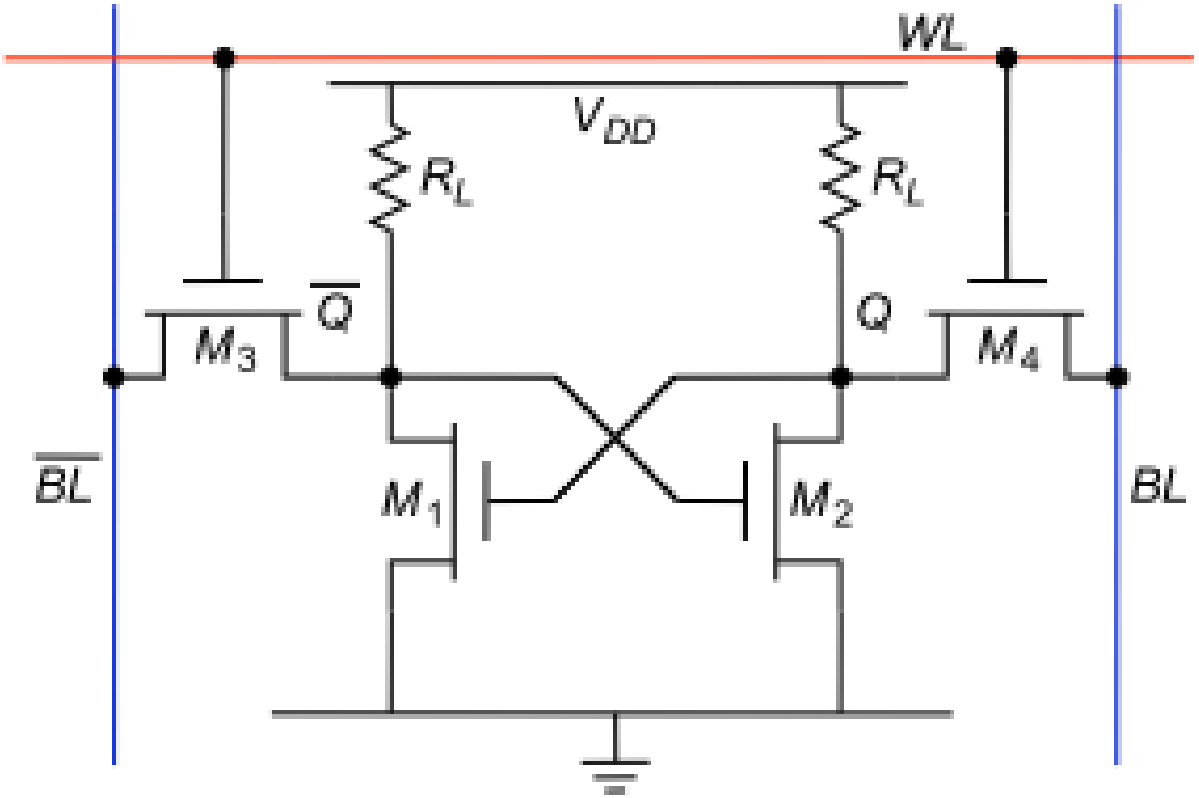




130 nm [Tyagi00] 90 nm [Thompson02] 65 nm [Bai04] 45 nm [Mistry07] 32 nm [Natarajan08]

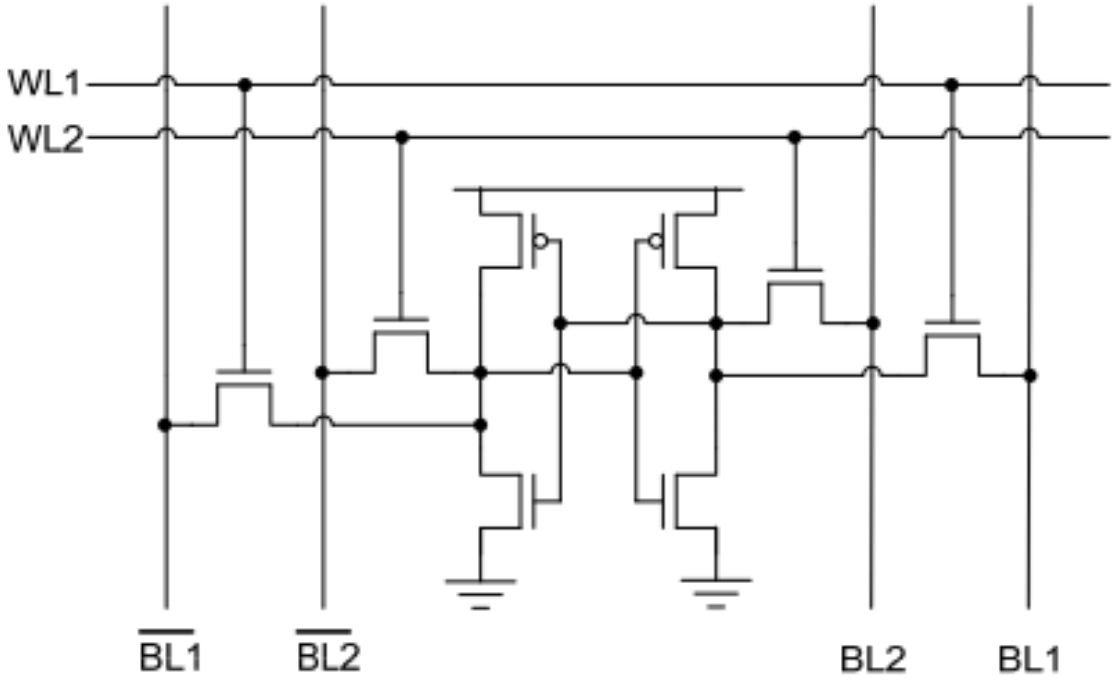
4T Memory Cell

- Achieve density by removing the PMOS pull-up.
- However, this results in static power dissipation.

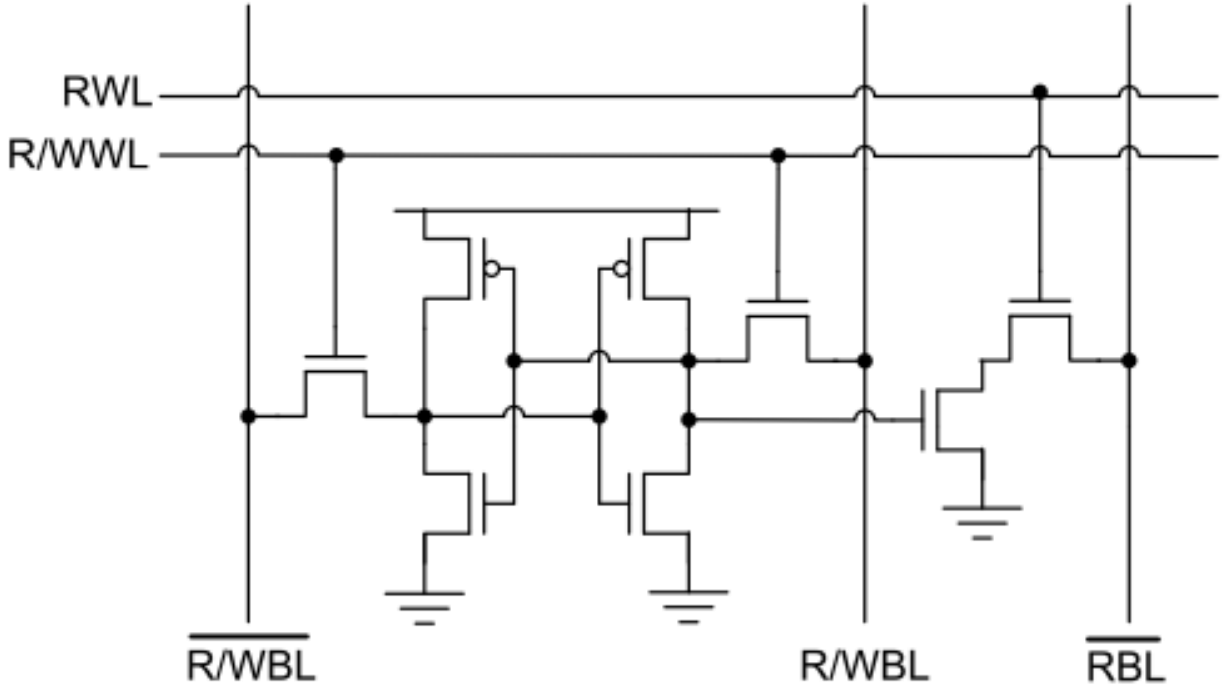


Multi-Port SRAM

Dual Port SRAM



Two Port SRAM



1 SRAM Memory Architecture

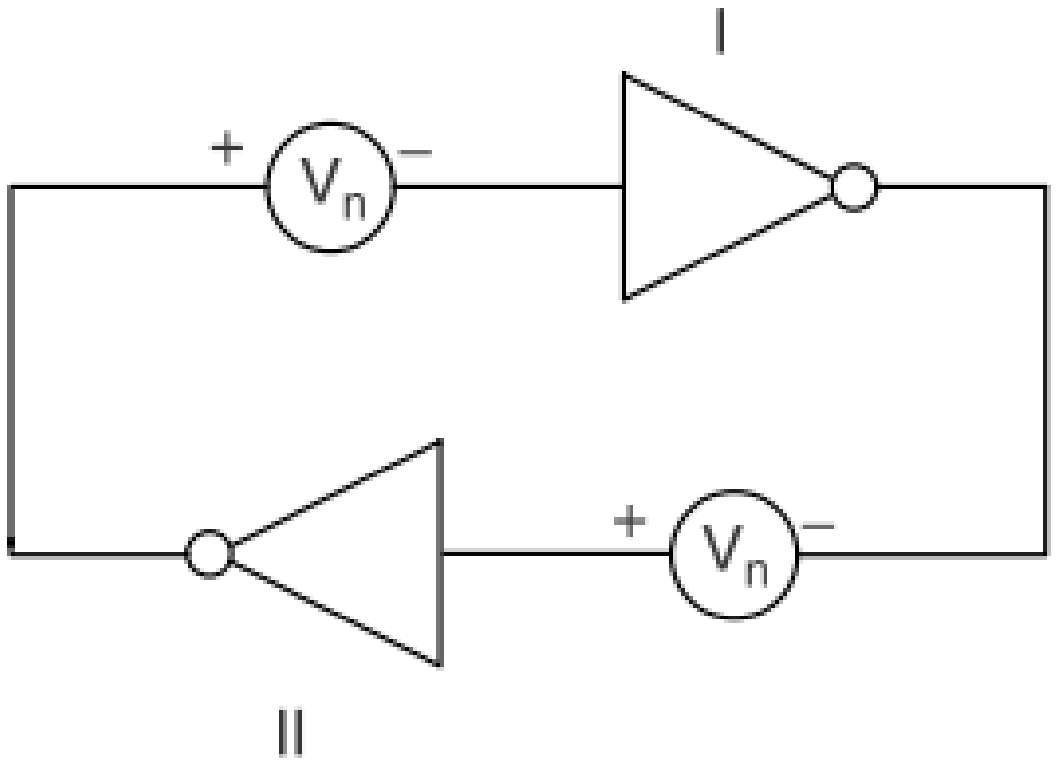
2 The 6T SRAM Bitcell

3 **SRAM Stability**

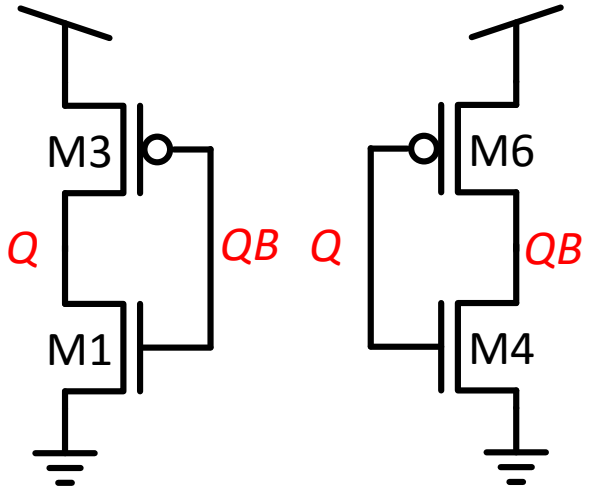
4 SNM Calculation

5 Peripheral Circuits

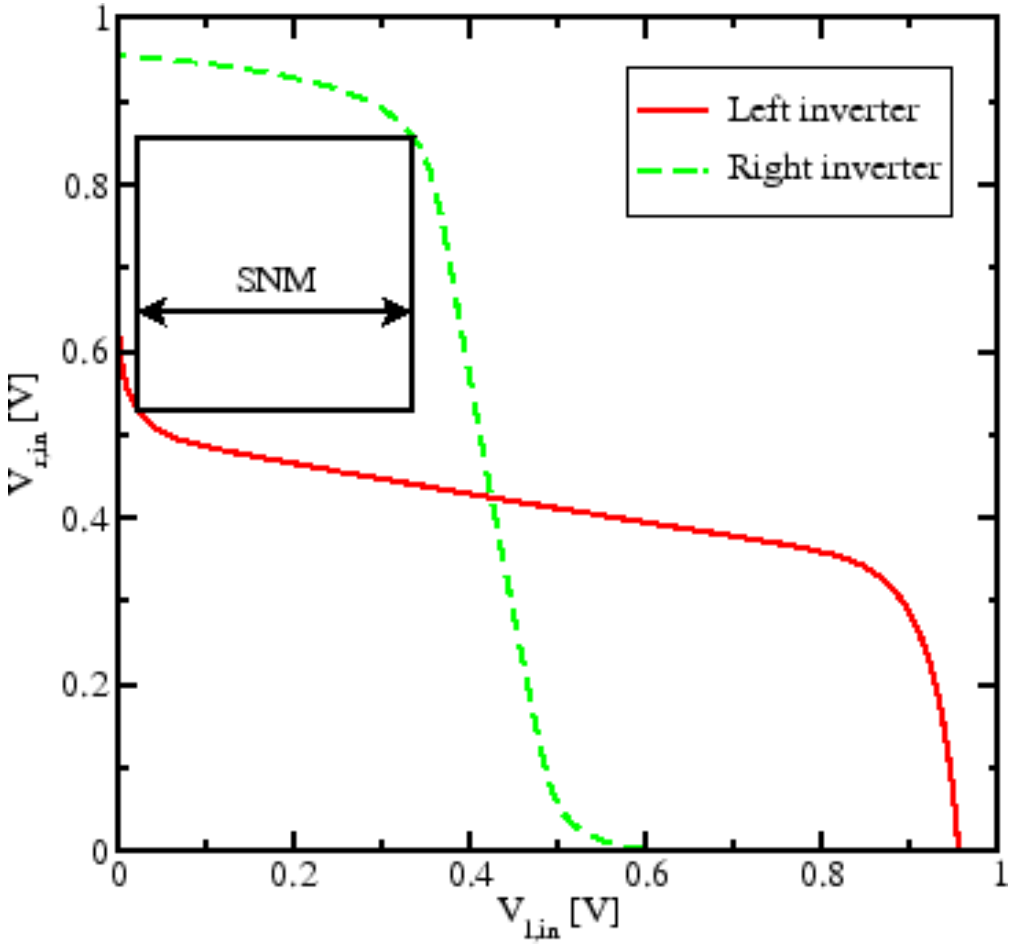
Static Noise Margin - Hold



Static Noise Margin - Hold

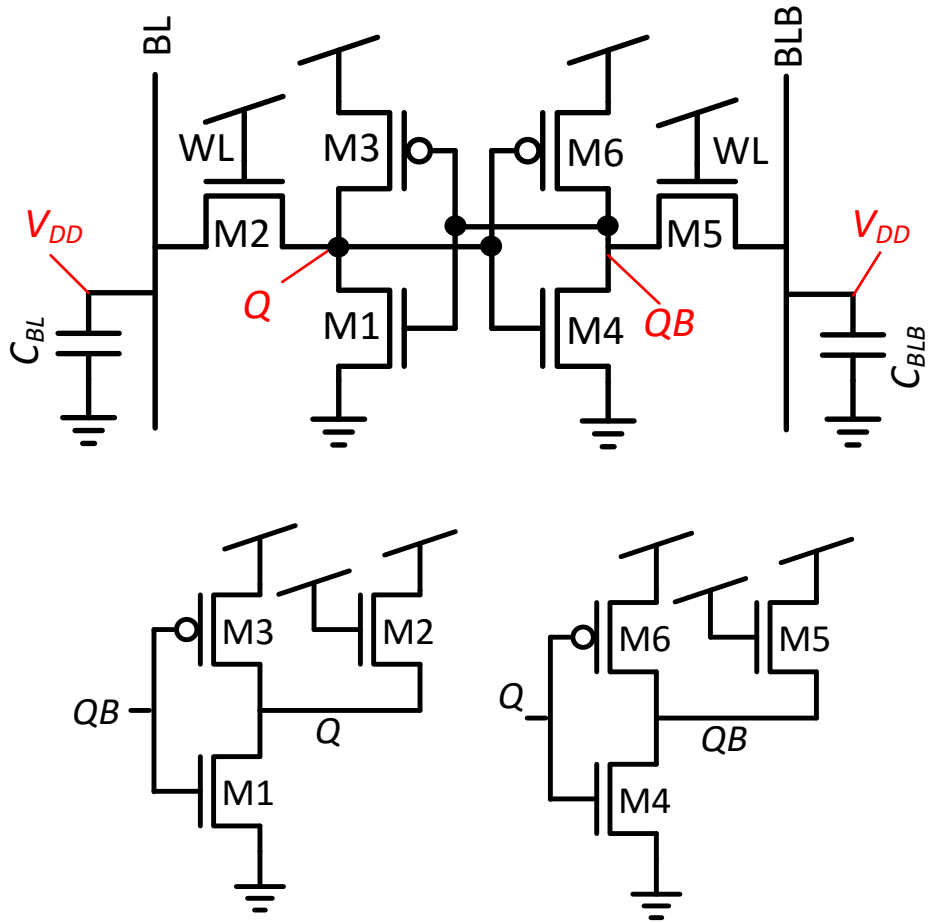


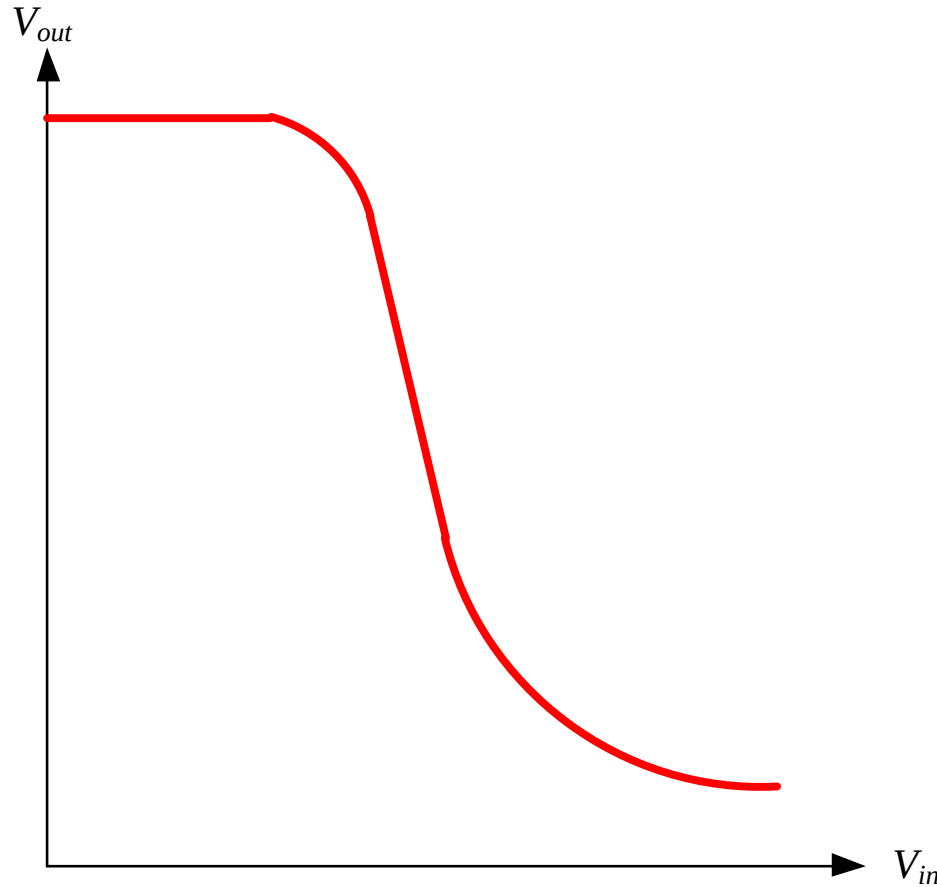
1. Plot both VTCs on the same graph
2. Find the maximum square that fits in the VTC.
3. The SNM is defined as the side of the maximum square.



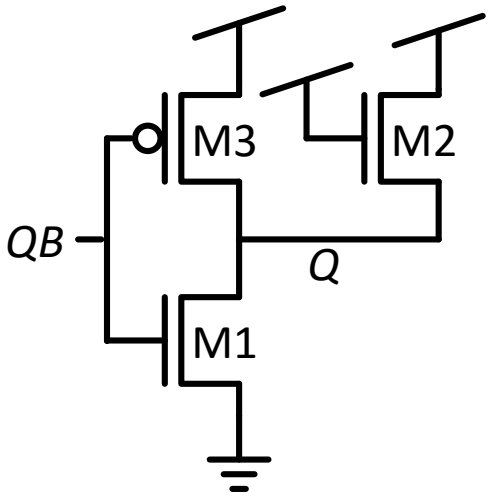
Static Noise Margin - Read

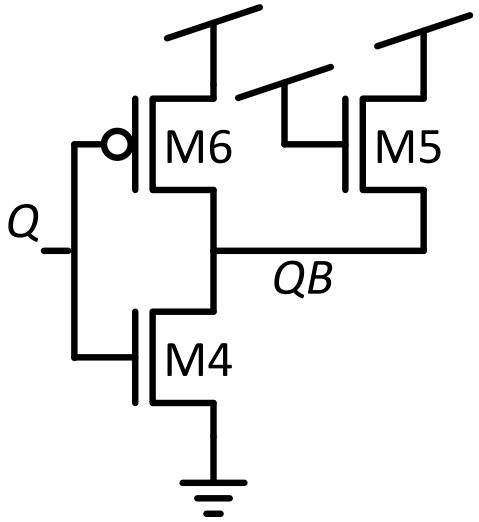
- What happens during Read?
  - We can't ignore the access transistors anymore…

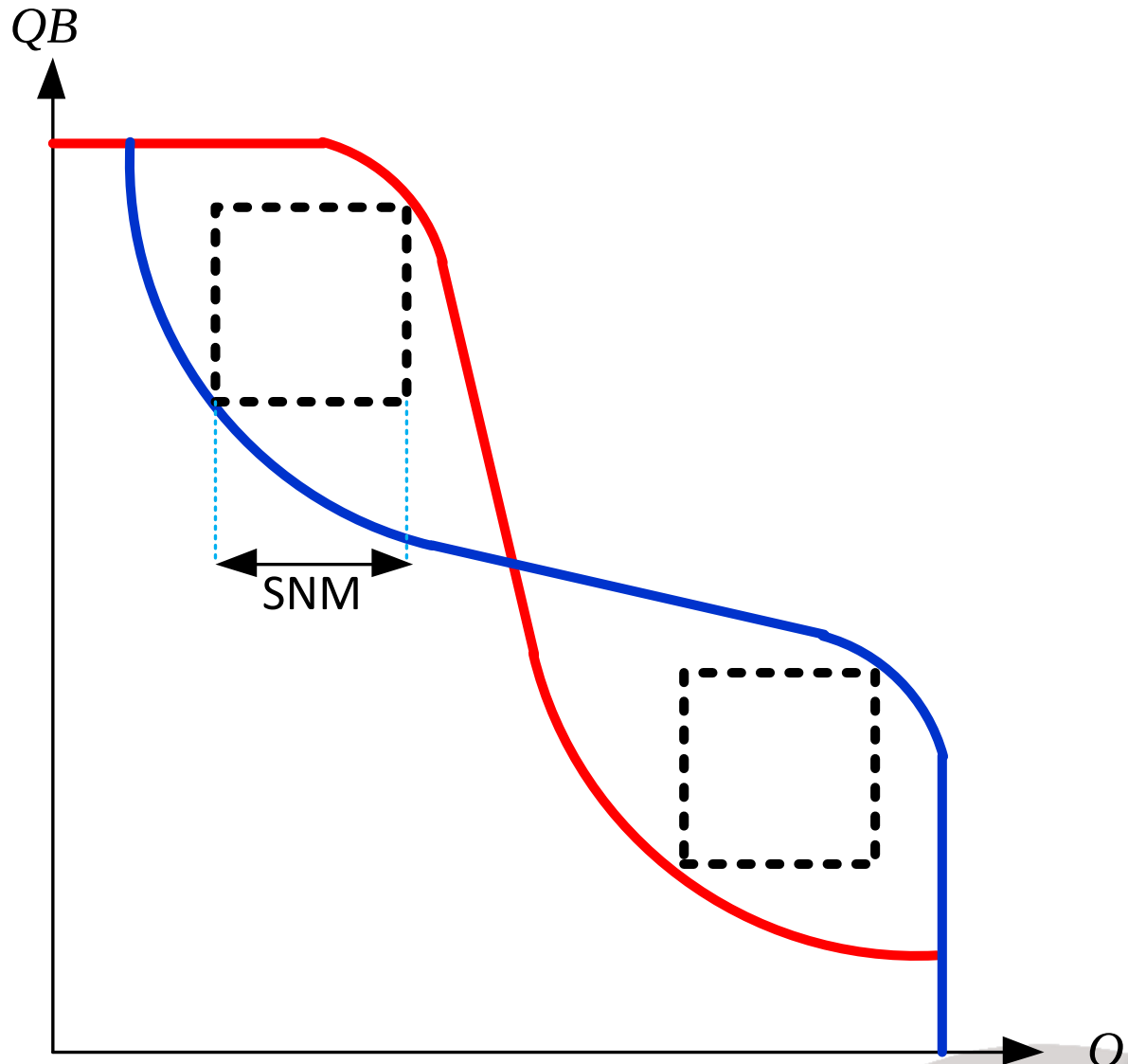




Static Noise Margin - Read

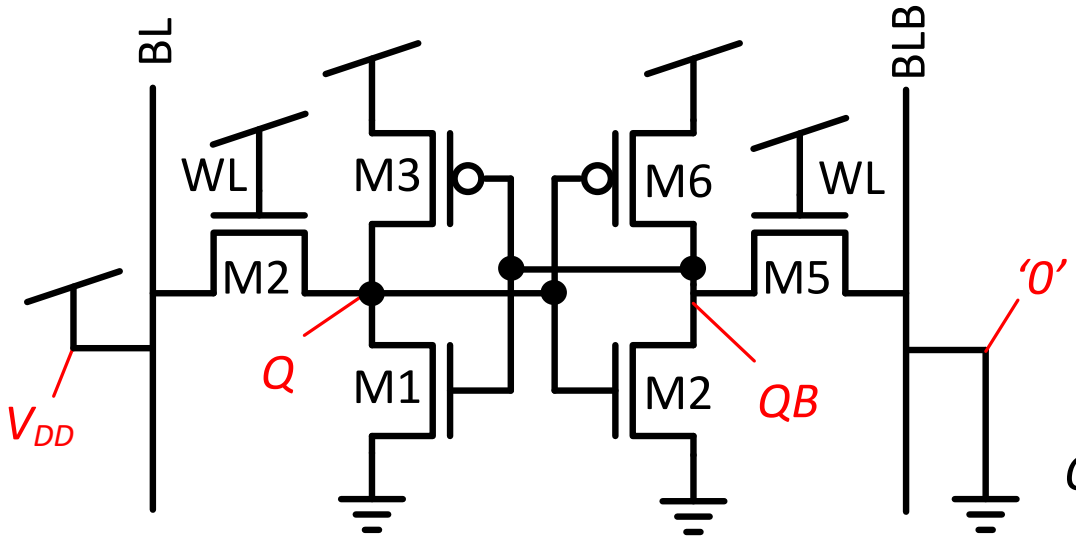


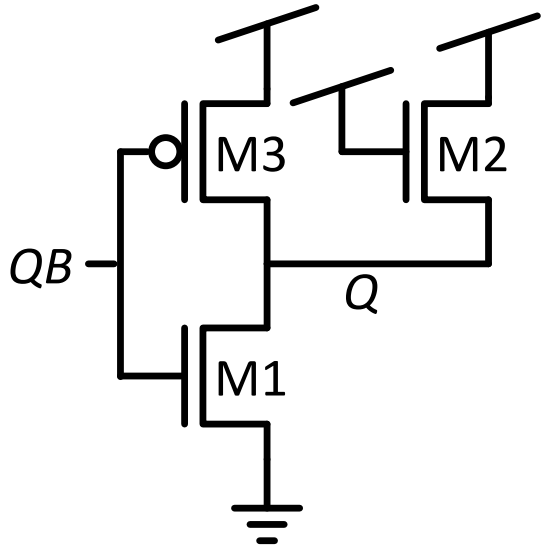


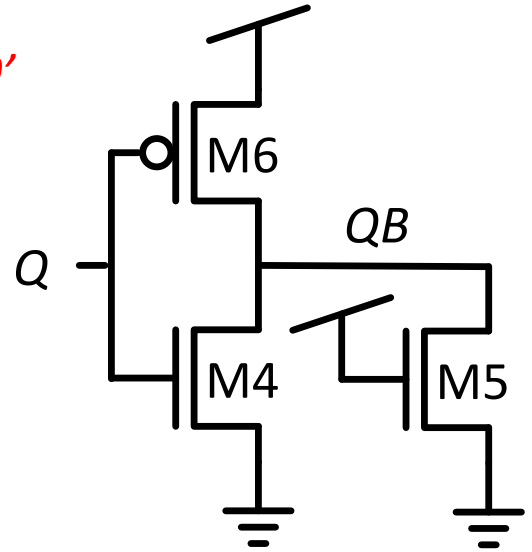


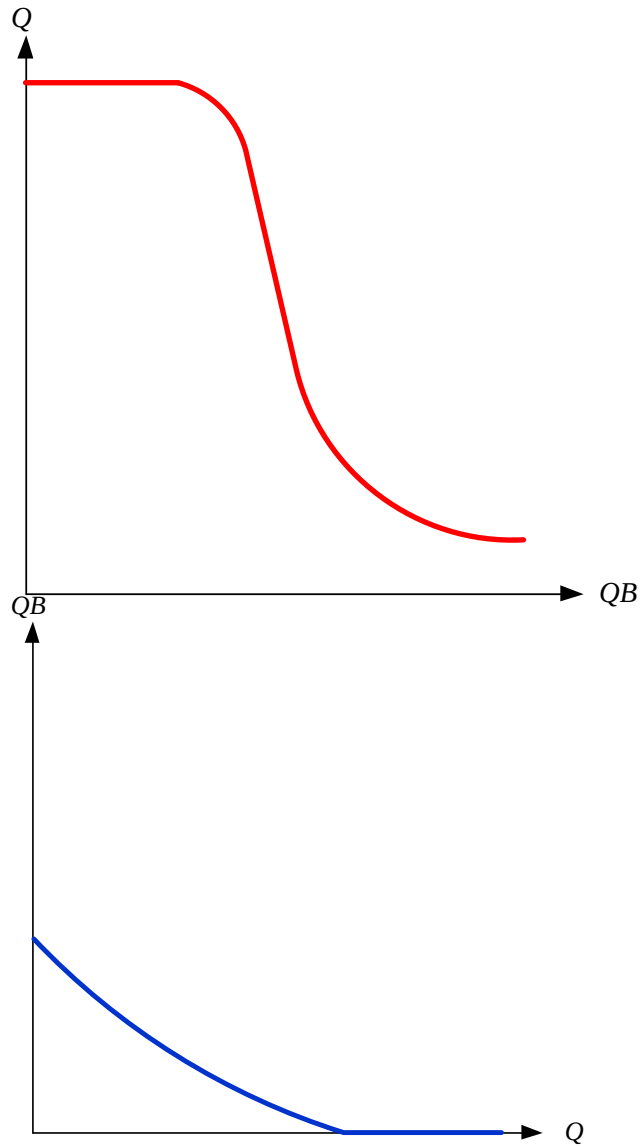
Static Noise Margin - Write

- What happens during Write?
  - The two sides are now different.

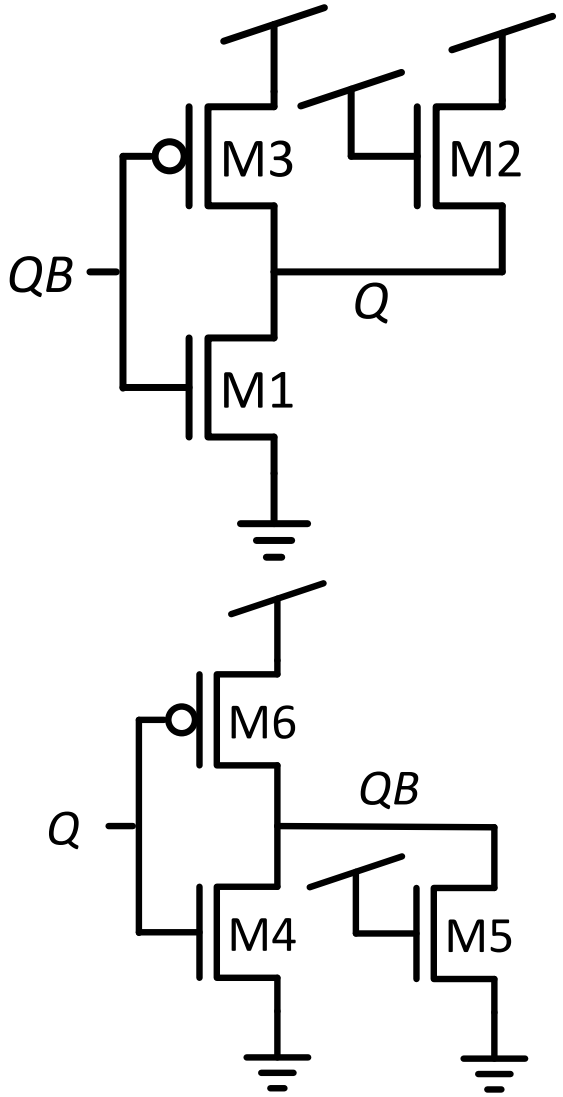


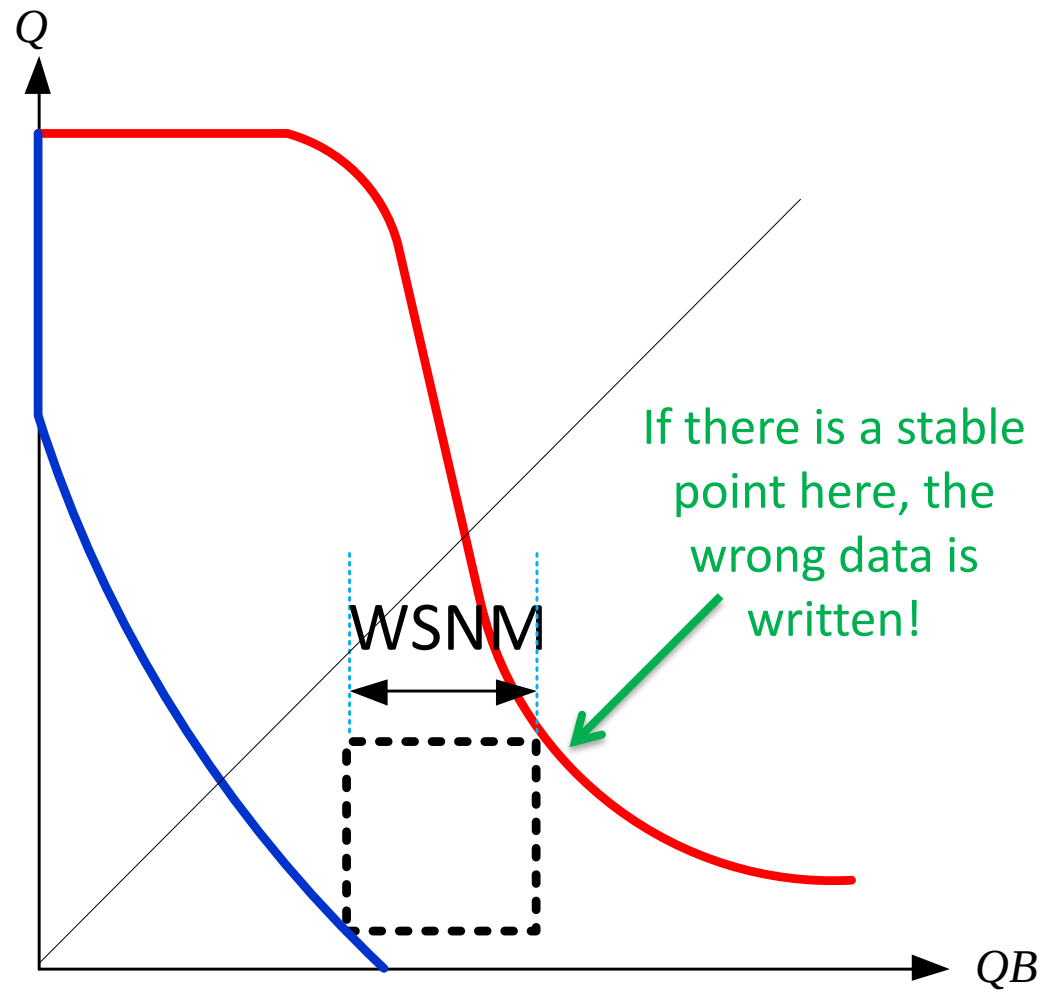






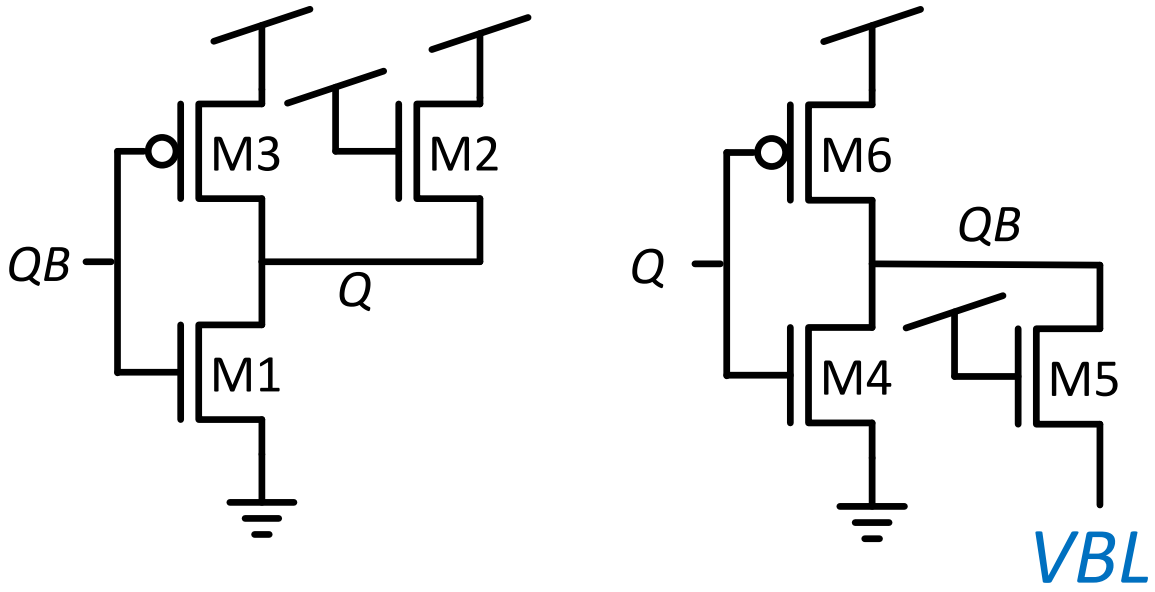
Static Noise Margin - Write

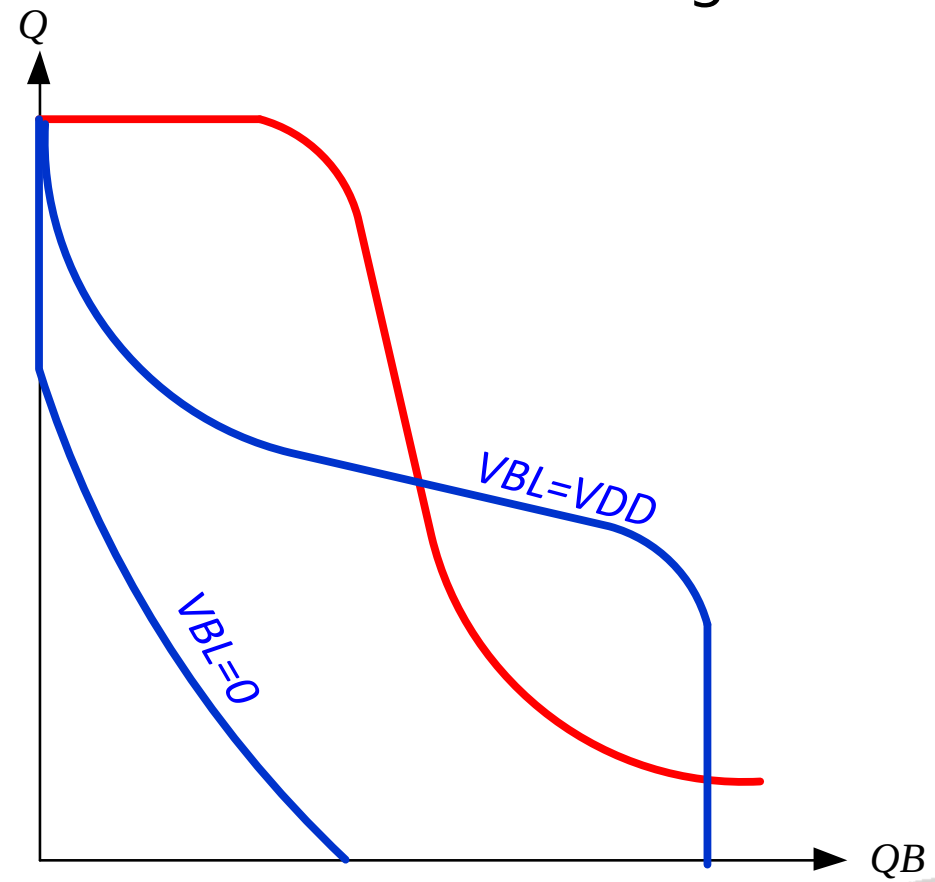




Alternative Write SNM Definition

- Write SNM depends on the cell's separatrix, therefore alternative definitions have been proposed.
- For example, add a DC Voltage (V_{BL}) to the 0 bitline and see how high it can be and still flip the cell.





Dynamic Stability

1 SRAM Memory Architecture

2 The 6T SRAM Bitcell

3 SRAM Stability

4 **SNM Calculation**

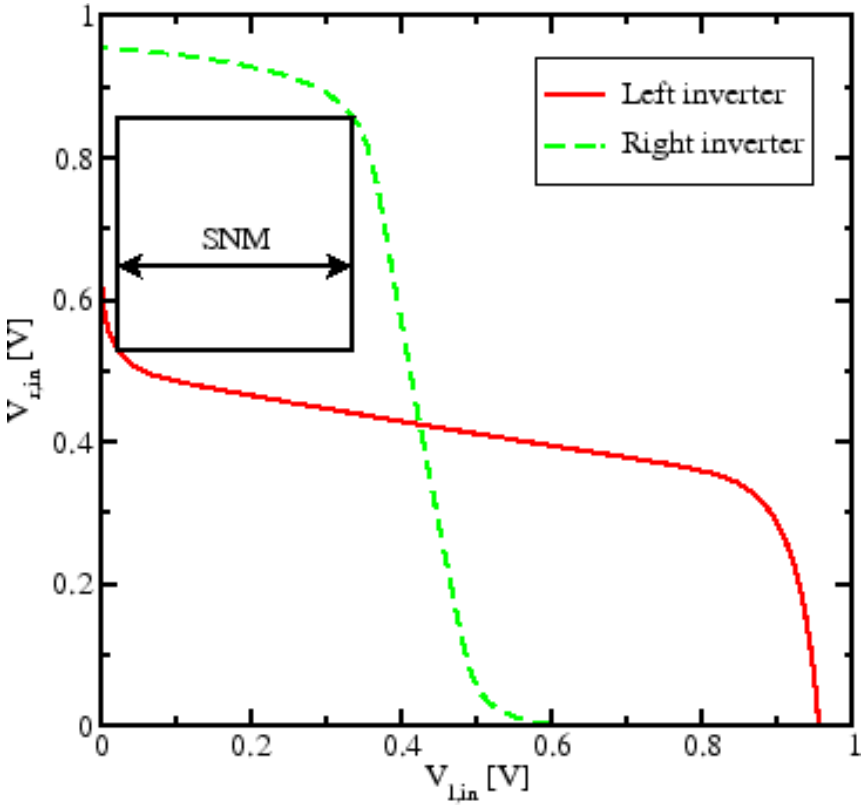
5 Peripheral Circuits

Simulating SNM

- Problem:
  - How can we calculate *SNM* with *SPICE*?
- Some options:
  - Insert *DC sources* at *Q* and *QB*
    - But where exactly do we connect them?
  - Draw *Butterfly Curves*
    - But how do we find the largest squares?
- To run Monte Carlo Simulations we should have an easy way of calculation.

Simulating SNM

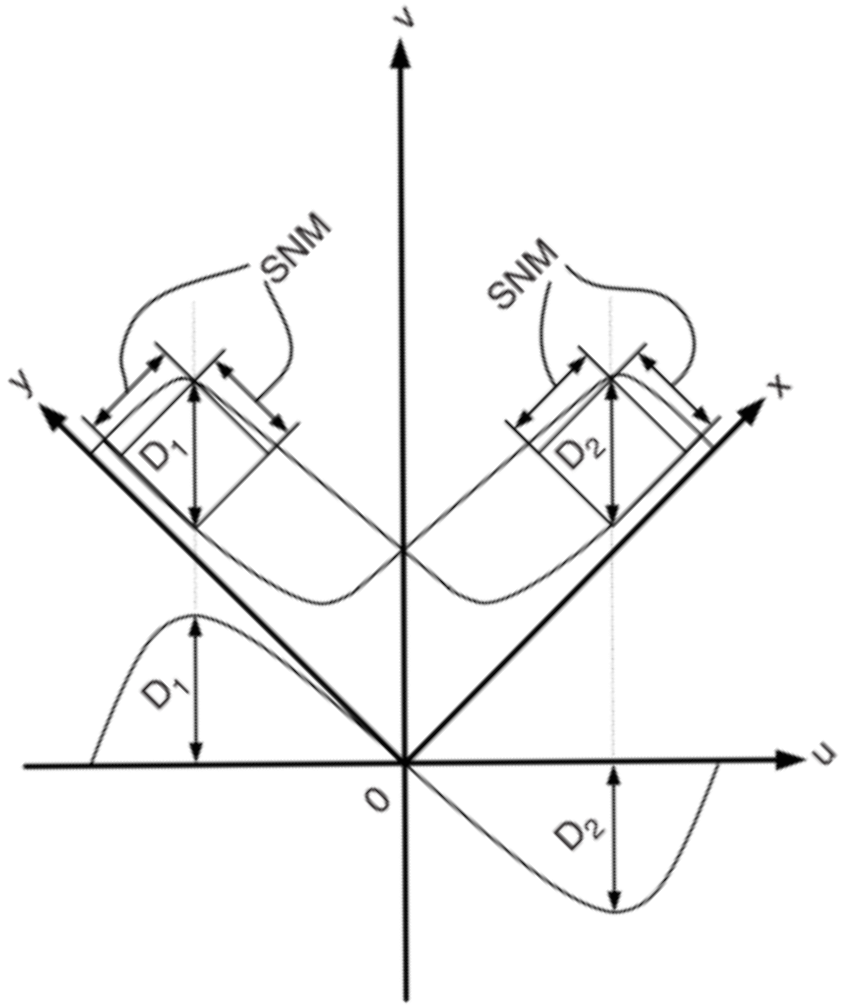
- First let's define the graphical solution:
  - The diagonals of all the squares are on lines parallel to $Q=QB$.
  - We need to find the distance between the points where these intersect the butterfly plot.
  - The largest of these distances is the diagonal of the maximum square in each lobe.
  - Multiply this by *cos45* and we get the SNM.



- Easy, right?

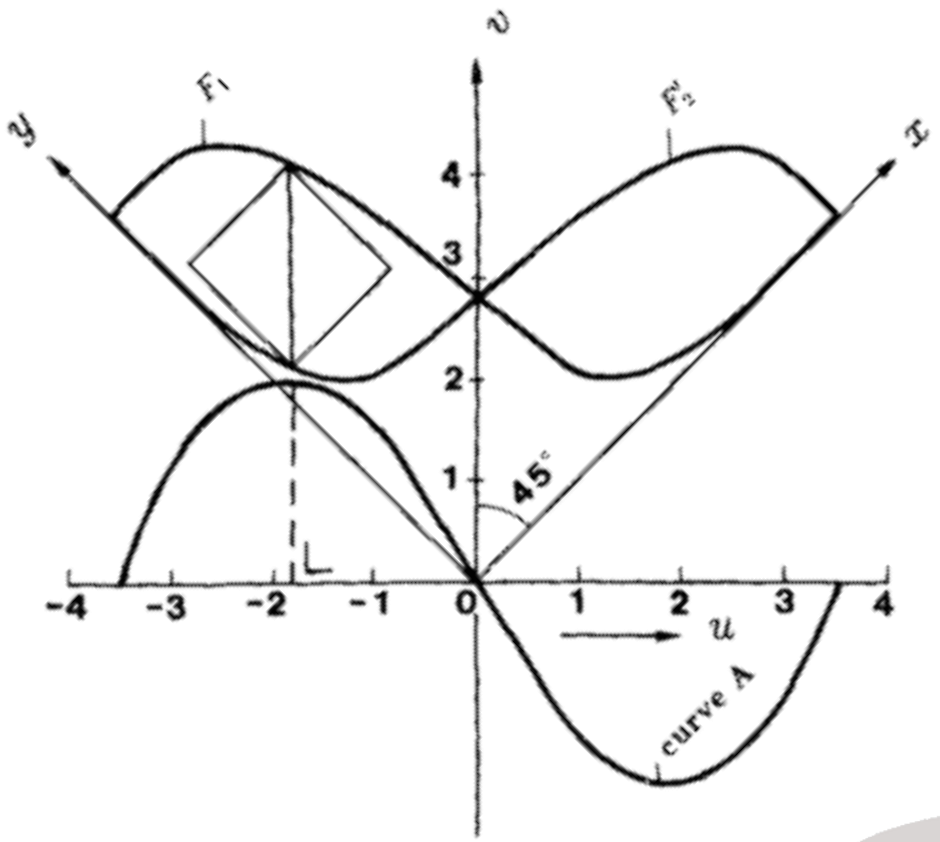
Changing Coordinates

- What if we were to turn the graph?



Changing Coordinates

- If we were to use new axes, we could just subtract the graphs.
  - This gives us the distances between the intersections with the $Q=QB$ parallels.
  - Now all we have to do is find the maximum of the subtraction.
  - (Don't forget to multiply by $cos45$)



Changing Coordinates

- The required transformation is:

$$x = \frac{1}{\sqrt{2}}u + \frac{1}{\sqrt{2}}v$$

$$y = -\frac{1}{\sqrt{2}}u + \frac{1}{\sqrt{2}}v$$

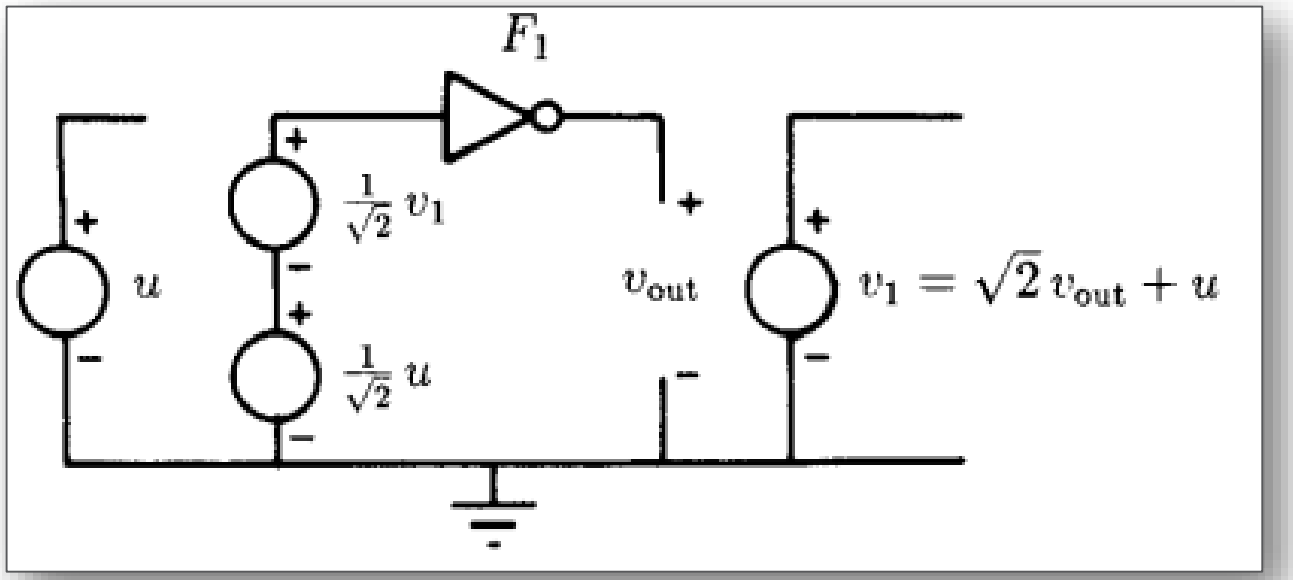
- Now let's define some function as F_1
- Substituting $y=F_1(x)$ gives:

$$y = F_1\left(x\right)$$

$$v = u + \sqrt{2}F_1\left(\frac{1}{\sqrt{2}}u + \frac{1}{\sqrt{2}}v\right)$$

Changing Coordinates

- What we did is turn some function ($F1$) 45 *degrees* counter clockwise.
- This can easily be implemented with the following circuit:

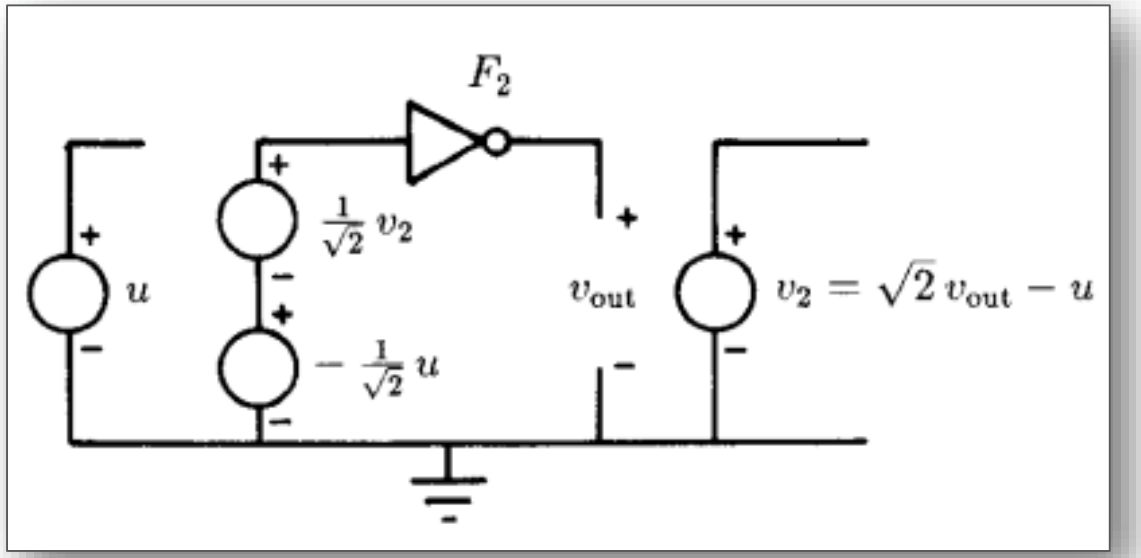


- What is $F1$?
  - It could be the VTC of $Vin=Q$, $Vout=QB$...

Changing Coordinates

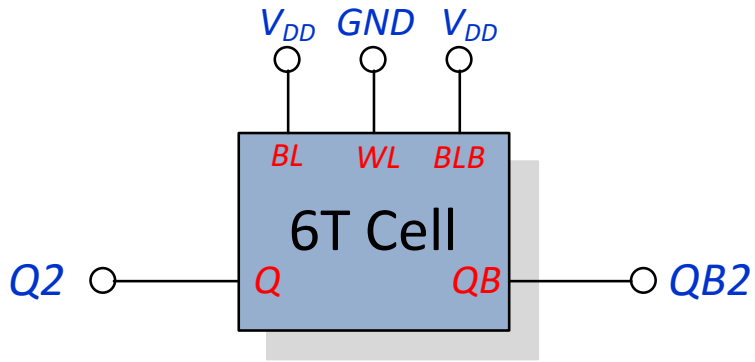
- But what about the "mirrored" VTC?
  - This needs to first be mirrored with respect to the v axis and then transformed to the *(u,v)* system.
  - If we call the second *VTC F2*, then the operation we need is:

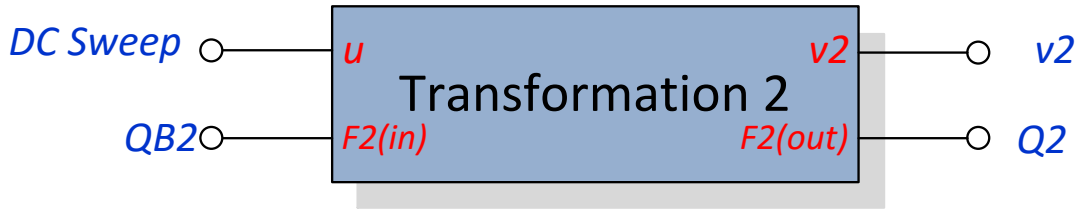
$$v = -u + \sqrt{2} F_2 \left(\frac{v}{\sqrt{2}} - \frac{u}{\sqrt{2}} \right)$$



Final SNM Calculation

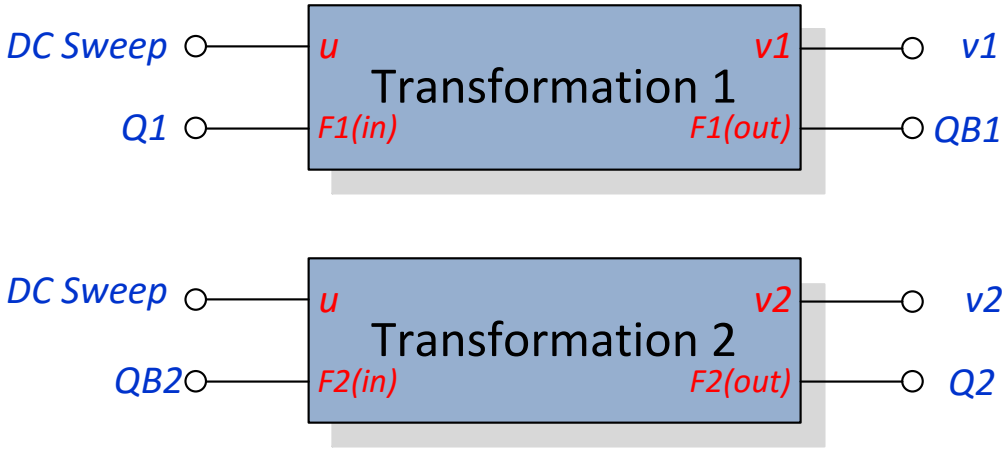
- Now we need to:
  - Make a schematic of our SRAM cell with two pins: Q and QB.
  - Create a coordinate changing circuit for each of the transformations.

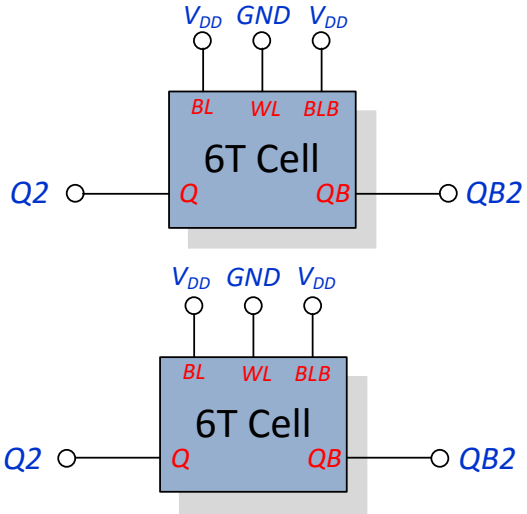




Final SNM Calculation

- Now, connect $F1$ to $Q \rightarrow QB$, and $F2$ to $QB \rightarrow Q$.





- Run a *DC Sweep* on u from $-V_{DD}/\sqrt{2}$ to $V_{DD}/\sqrt{2}$

- This will present the butterfly curves turned 45 degrees.

Final SNM Calculation

- Now just:
  - Subtract the bottom graph from the top one.
  - Find the local maxima for each lobe.
  - The smaller of the local maxima is the diagonal of the largest square.
  - Multiply this by *cos45* for the *SNM*

$$SNM = \frac{1}{\sqrt{2}} \cdot \min\left[\max\left(|v1 - v2|\right)\Big|_{-\sqrt{2}<u<0}, \max\left(|v1 - v2|\right)\Big|_{0<u<\sqrt{2}} \right]$$

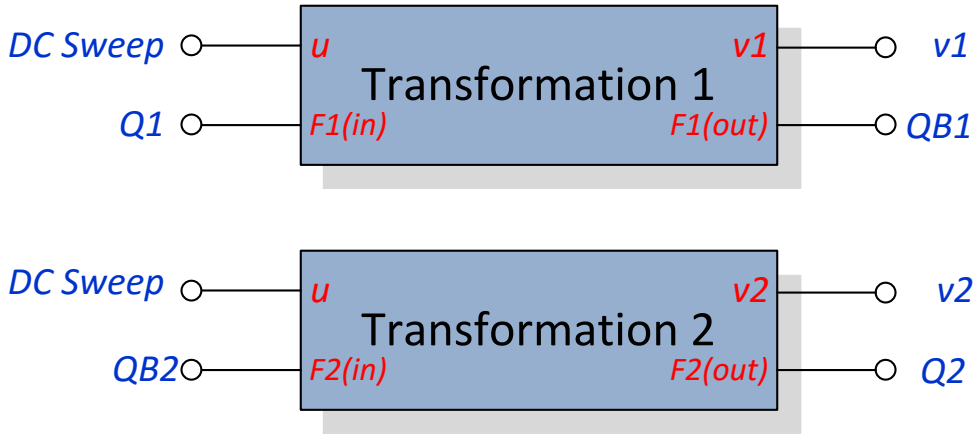
Read/Write SNM

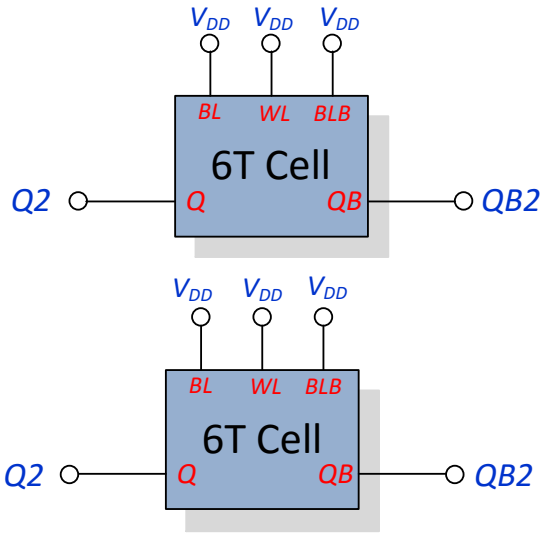
- How about Read SNM:
  - Use the exact same setup.
  - Connect BL and BLB to VDD.
  - Connect WL to VDD.
  - Run the same calculation.

- And Write SNM.
  - Now connect one BL to GND.
  - This is trickier, so you'll have to play around with the calculation.
  - There are other options for WM calculation.

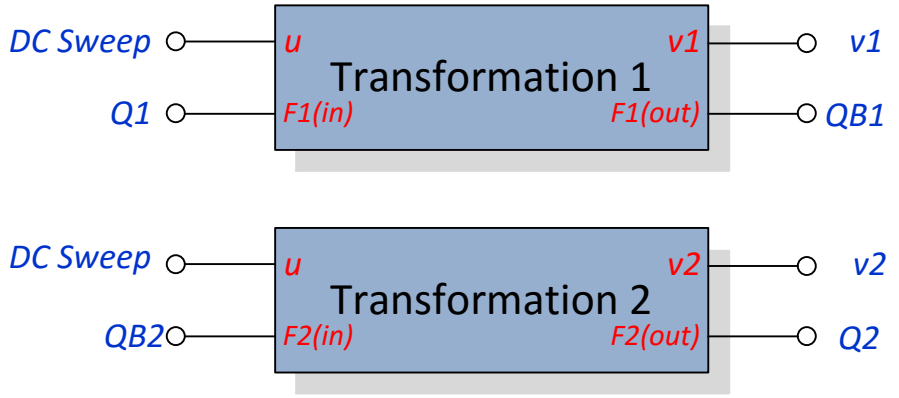
Testbench Setup – Read/Write

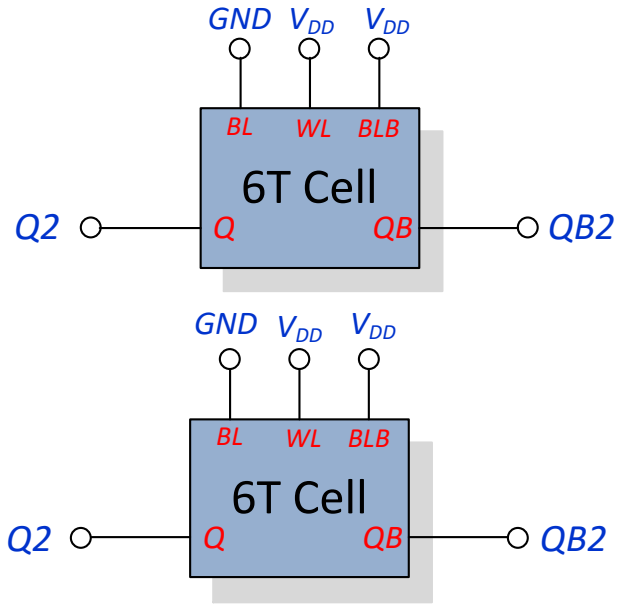
Read Testbench:





Write Testbench:





1 SRAM Memory Architecture

2 The 6T SRAM Bitcell

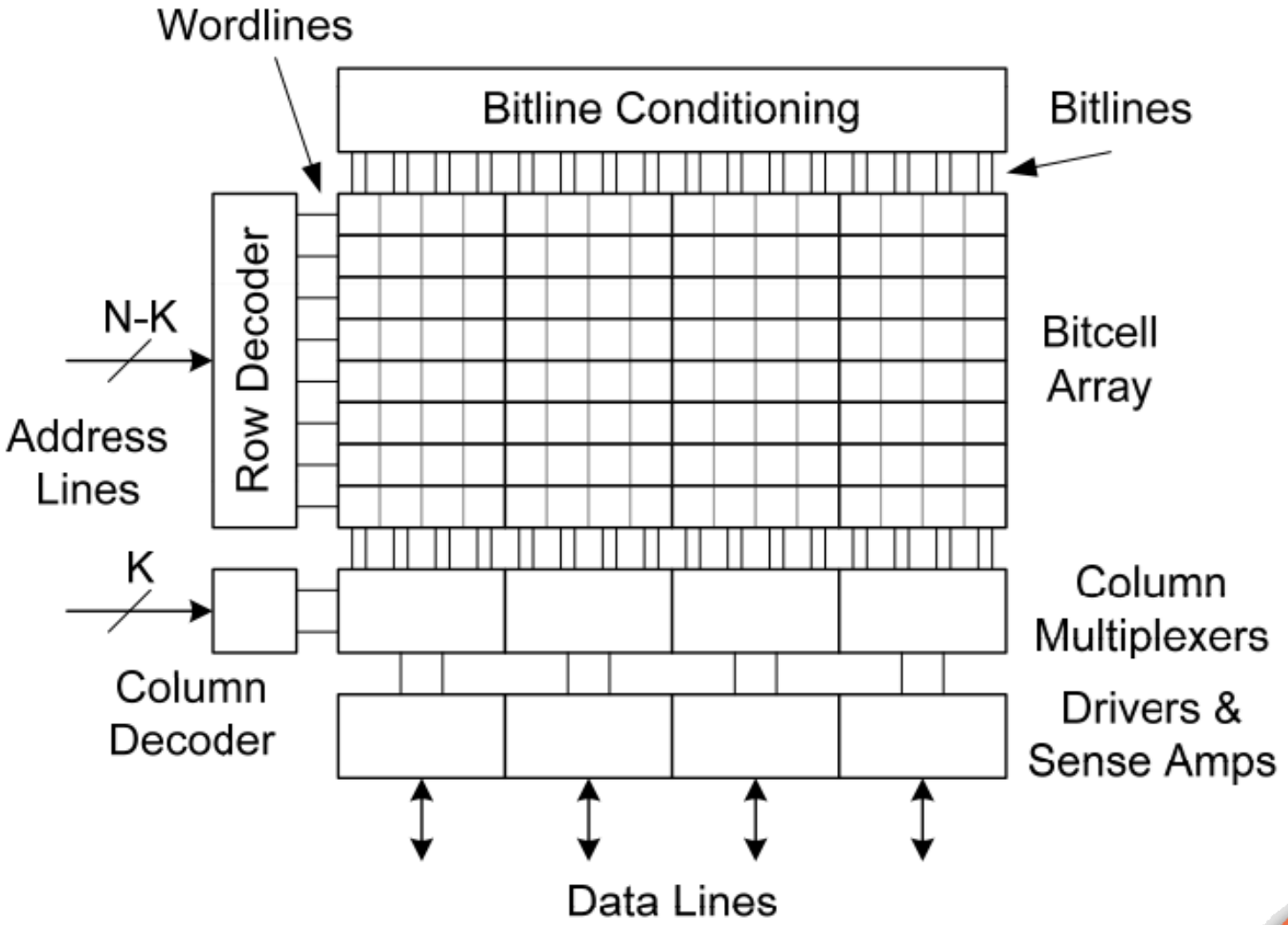
3 SRAM Stability

4 SNM Calculation

5 **Peripheral Circuits**

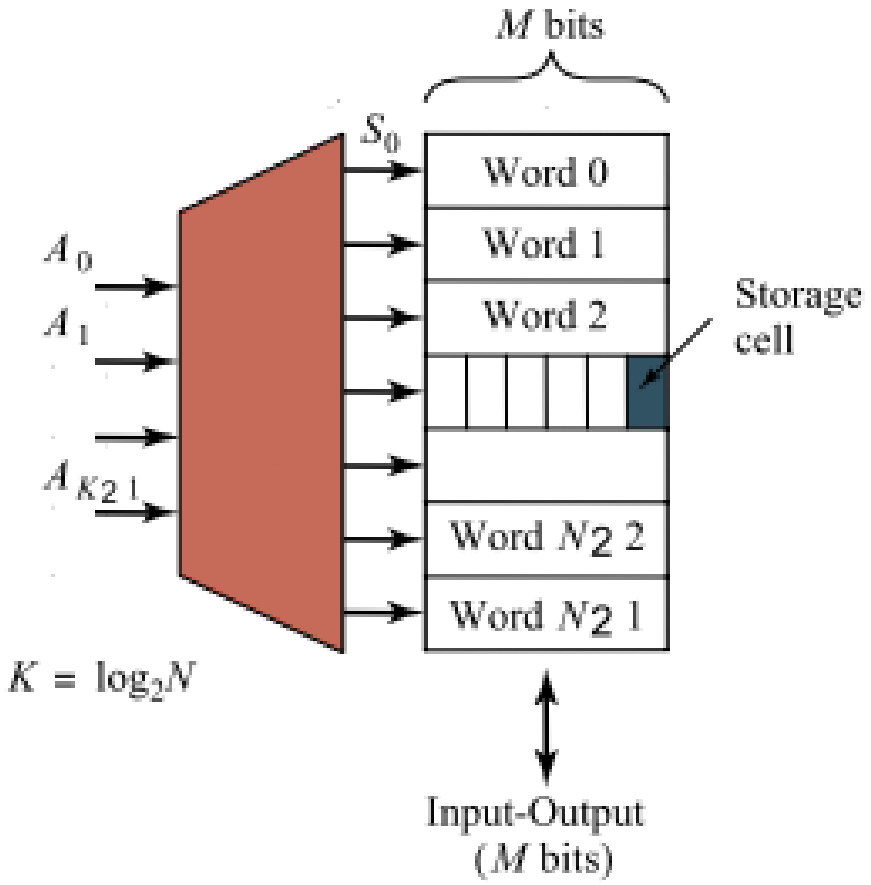
Major Peripheral Circuits

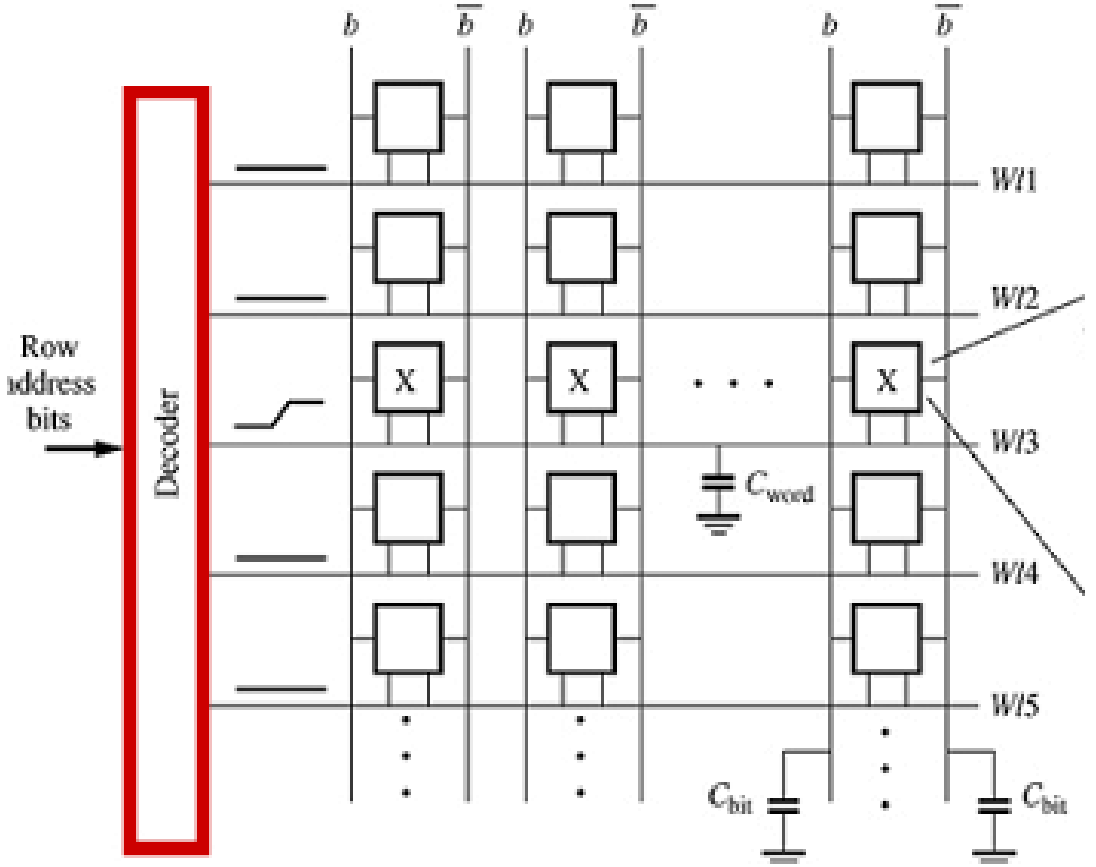
- Row Decoder
- Column Multiplexer
- Sense Amplifier
- Write Driver
- Precharge Circuit



Row Decoders

- A Decoder reduces the number of select signals by $\log_2$.
  - Number of Rows: N
  - Number of Row Address Bits: $\log_2 N$





Row Decoders

- Standard Decoder Design:
  - Each output row is driven by an AND gate with $k=\log_2 N$ inputs.
  - Each gate has a unique combination of address inputs (or their inverted values).
  - For example, an 8 bit row address has 256 8-input AND gates, such as:

$$WL_0 = \overline{A}_7 \overline{A}_6 \overline{A}_5 \overline{A}_4 \overline{A}_3 \overline{A}_2 \overline{A}_1 \overline{A}_0$$

$$WL_{255} = A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0$$

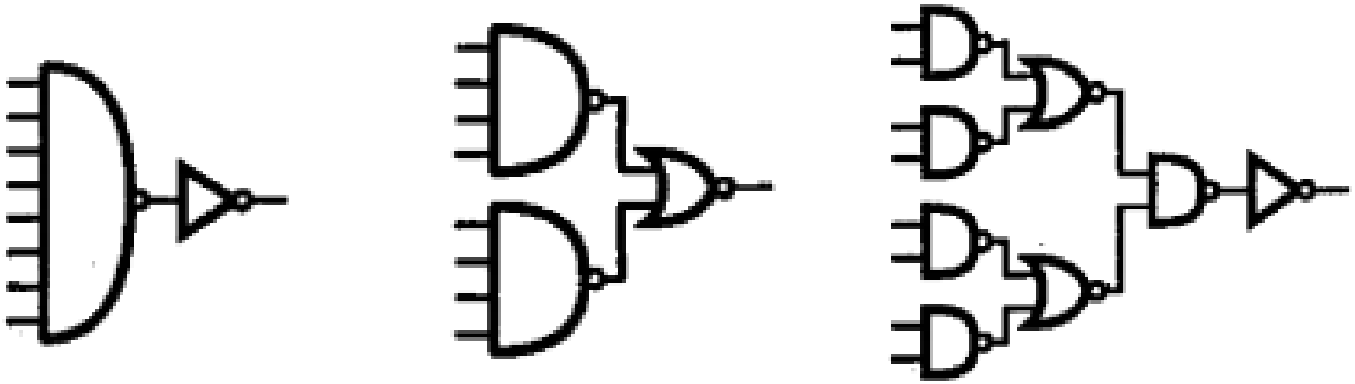
- NOR Decoder:
  - DeMorgan will provide us with a NOR Decoder.
  - In the previous example, we'll get 256 8-input NOR gates:

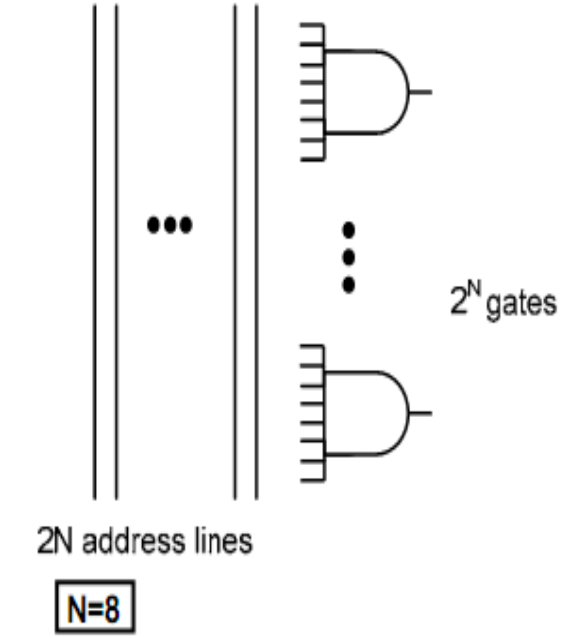
$$WL_0 = \overline{A_7 + A_6 + A_5 + A_4 + A_3 + A_2 + A_1 + A_0}$$

$$WL_{255} = \overline{\overline{A}_7 + \overline{A}_6 + \overline{A}_5 + \overline{A}_4 + \overline{A}_3 + \overline{A}_2 + \overline{A}_1 + \overline{A}_0}$$

How should we build it?

- Let's build a row decoder for a 256x256 SRAM Array.
- We need 256 8-input AND Gates.
- Each gate drives 256 bitcells
- We have various options:





- Which one is best?

1 SRAM Memory Architecture

2 The 6T SRAM Bitcell

3 SRAM Stability

4 SNM Calculation

5 **Peripheral Circuits**

Logical Effort of a Decoder

Reminder: Logical Effort

$$t_{pd,i} = t_{pINV}\left(p_i\gamma + EF_i\right) \qquad EF_i \triangleq LE_i \cdot f_i = LE_i \cdot \frac{b_i \cdot C_{in,i+1}}{C_{in,i}}$$

$$PE = F \cdot \prod LE \cdot B = {C_L}/{C_{in,1}} \cdot \prod LE_i \prod b_i$$

$$EF_{opt} = \sqrt[N]{PE} = \sqrt[N]{F \cdot \prod LE_i \prod b_i}$$

$$N_{opt} = \log_{EF_{opt}} PE = \log_{EF_{opt}} F \cdot LE \cdot B$$

$$t_{pd} = t_{pINV} \sum\left(p_i\gamma + EF_i\right) = t_{pINV}\left(\gamma \sum p_i + N \cdot \sqrt[N]{PE}\right)$$

Problem Setup

- For LE calculation we need to start with:
  - Output Load
  - Input Capacitance
  - Branching
- What is the Load Capacitance?
  - 256 bitcells on each Word Line

$$C_{WL} = 256 \cdot C_{Cell} + C_{Wire}$$

  - Let's ignore the wire for now…
- What is the Input Capacitance?
  - Let's assume our address drivers can drive a bit more than a bitcell, so:

$$C_{in,addr\_driver} = 4 \cdot C_{Cell}$$

Problem Setup

- What is the Branching Effort?
  - Lets take another look at the Boolean expressions:

$$WL_0 = \bar{A}_7 \bar{A}_6 \bar{A}_5 \bar{A}_4 \bar{A}_3 \bar{A}_2 \bar{A}_1 \bar{A}_0$$

$$WL_{255} = A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0$$

  - We see that half of the signals use A_i and half use A_i!
  - So each address driver drives 128 8-input AND gates, but only one is on the selected WL path.

$$C_{on\ path} = C_{nand}; \quad C_{off\ path} = 127 \cdot C_{nand}$$

$$B_{add\_driver} = \frac{C_{on\ path} + C_{off\ path}}{C_{on\ path}} = \frac{C_{nand} + 127 \cdot C_{nand}}{C_{nand}} = 128$$

Number of Stages

- Altogether the path effort is:

$$LE \cdot B \cdot F = LE \cdot \Pi b_i \frac{C_{WL}}{C_{address}} = LE \cdot 128 \cdot \frac{256 C_{Cell}}{4 C_{Cell}}$$

$$= LE \cdot 8k = 2^{13} \cdot LE$$

- The best case logical effort is

$$\Pi LE = 1$$

- So the ***minimum*** number of stages for optimal delay is:

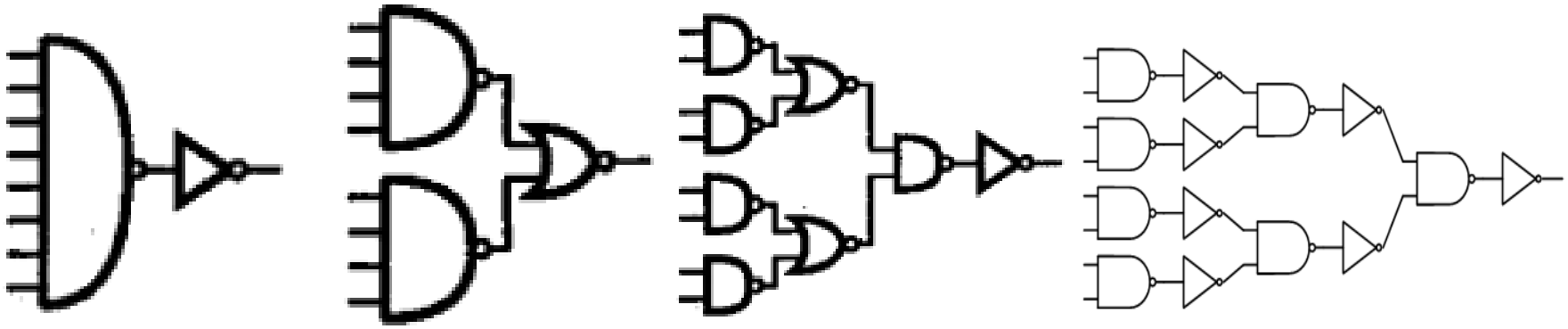
$$PE = 2^{13}$$

$$N_{opt} = \log_{3.6} 2^{13} = 7$$

- That's a lot of stages!

So which implementation should we use?

- The one with the minimum Logical Effort:



$\Pi LE = (10/3)\cdot 1 = 10/3;$
$p = 8+1 = 9$

$\Pi LE = 2\cdot(5/3) = 10/3$
$p = 4+2 = 6$

$\Pi LE = (4/3)\cdot(5/3)\cdot(4/3)\cdot 1 = 80/27;$
$p = 2+2+2+1 = 7$

$\Pi LE = (4/3)^3 = 2.37;$
$p = 2\cdot 3+1\cdot 3 = 9$

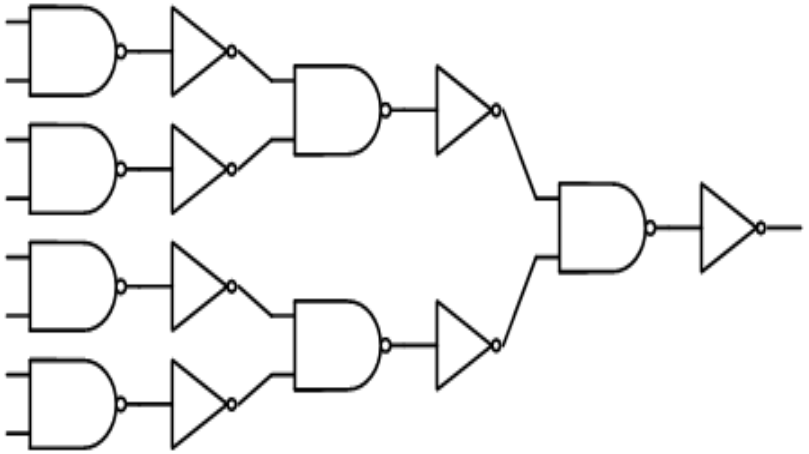
New optimal number of Stages

- So now we can calculate the actual path effort:

$$PE = F \cdot \Pi b_i \cdot \Pi LE_i =$$

$$= 2.37 \cdot 2^{13} = 19.418k$$

$$N_{opt} = \log_{3.6} PE = 7.7$$



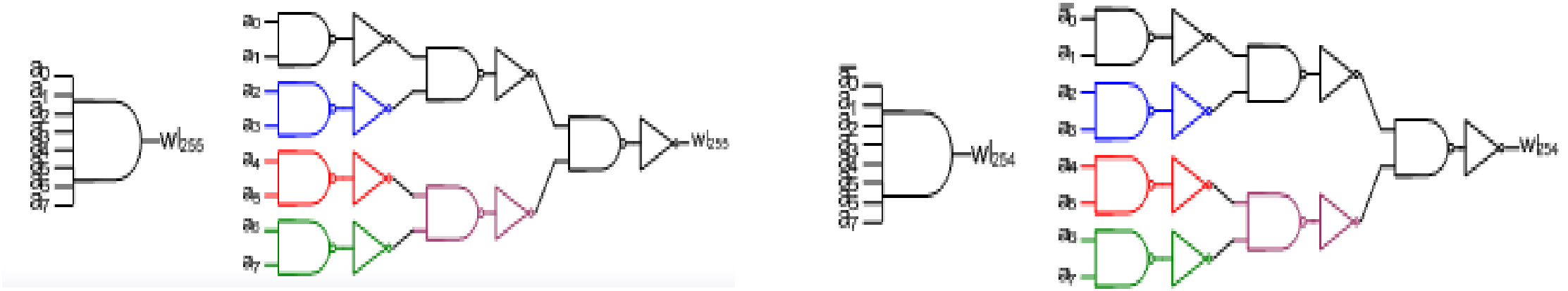
- We could add another inverter or two to get closer to the optimal number of stages…

Implementation Problems

- Address Line Capacitance:
  - Our assumption was that $C_{in,addr\_driver}=4C_{cell}$.
  - But each address drives 128 gates, resulting in a really long wire with high capacitance.
  - This means that we will need to buffer the address lines, and probably ruin our whole analysis…
- Bit-cell Pitch:
  - Each signal drives one row of bitcells.
  - How will we fit 8 address signals into this pitch?

Predecoding - Concept

- Solution:
  - Let's look at two decoder paths: W_{254}, W_{255}



  - We see that there are many "shared" gates.
    - So why not share them?
  - For instance, we can use the purple output for both gates…

Predecoding - Method

- How do we do this?
  - If we look at the final Boolean expression, it has combinations of groups of inputs.
  - By grouping together a few inputs, we actually create a small decoder.
  - Then we just AND the outputs of all the “pre” decoders.

- For example: Two 4:16 predecoders

$$D = dec(A_0, A_1, A_3, A_4);\ \ E = dec(A_5, A_6, A_7, A_8);$$

$$W_0 = D_0 \cdot E_0;\ \ W_{255} = D_{15} \cdot E_{15};\ W_{254} = D_{15} \cdot E_{14};$$

Predecoding - Example

- Let's look at our example:

$$D = dec(A_0, A_1, A_3, A_4) \qquad W_0 = D_0 \cdot E_0$$
$$E = dec(A_5, A_6, A_7, A_8) \qquad W_{255} = D_{15} \cdot E_{15}$$
$$W_{254} = D_{15} \cdot E_{14}$$

- What is our new branching effort?
  - As before, each address drives half the lines of the small decoder.
  - Each predecoder output drives 256/16 post-decoder gates.
  - Altogether, the branching effort is:

- Same as before!

$$B = b_{addr\_driver} \cdot b_{predecoder} = 16/2 \cdot 256/16 = 128$$

Predecoding - Solution

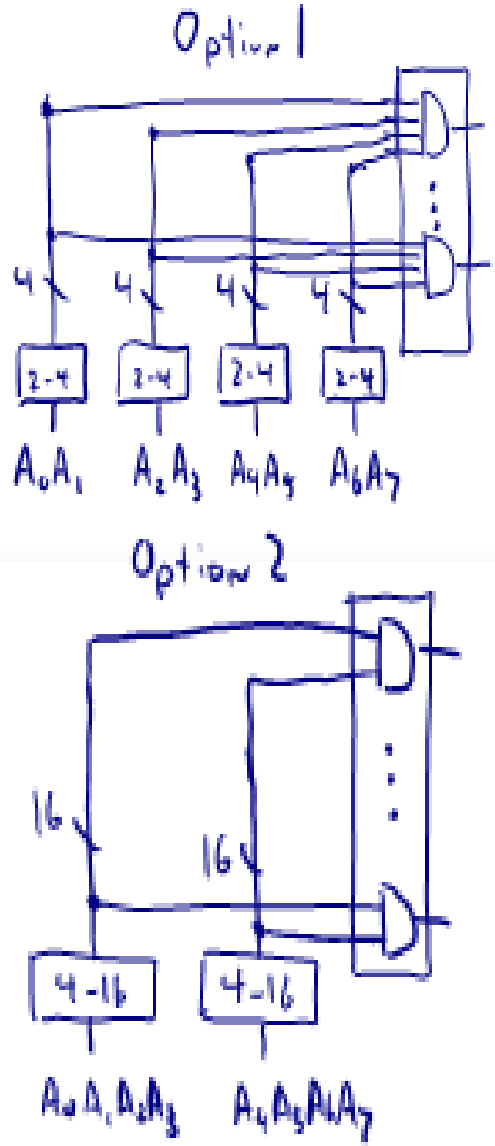
- Why is this a better solution?
  - Each Address driver is only driving four gates
    - less capacitance.
  - We saved a ton of area by “sharing” gates.
  - We can “Pitch Fit” 2-input NAND gates.

Another Predecoding Example

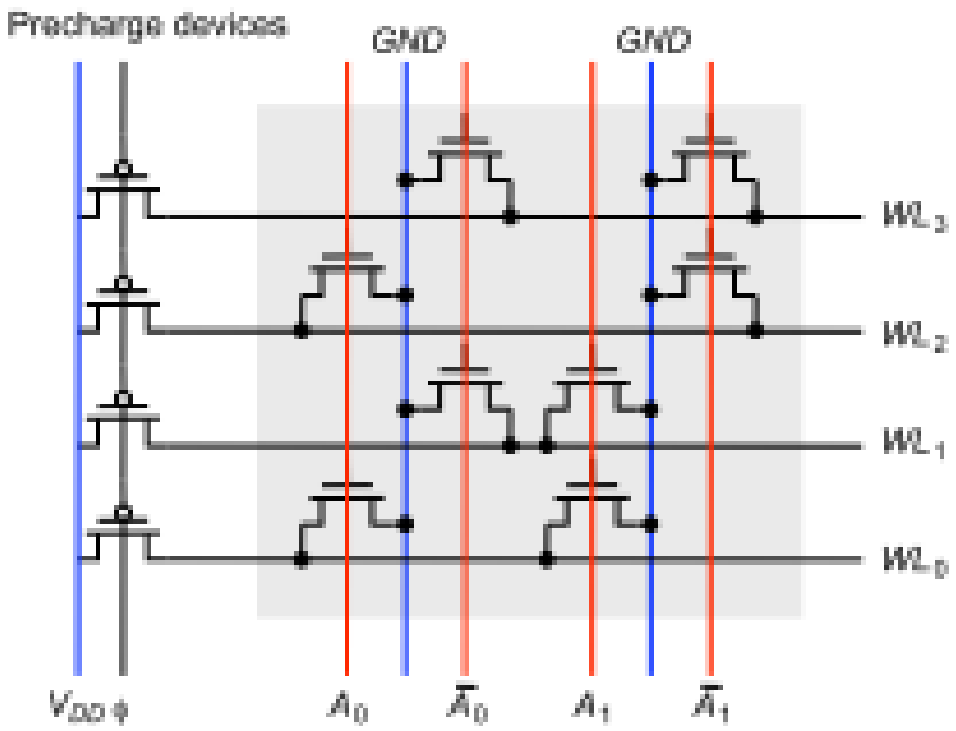
- We can try using four 2 input predecoders:
  - This will require us to use 256 4-input NAND gates.

How do we choose a configuration?

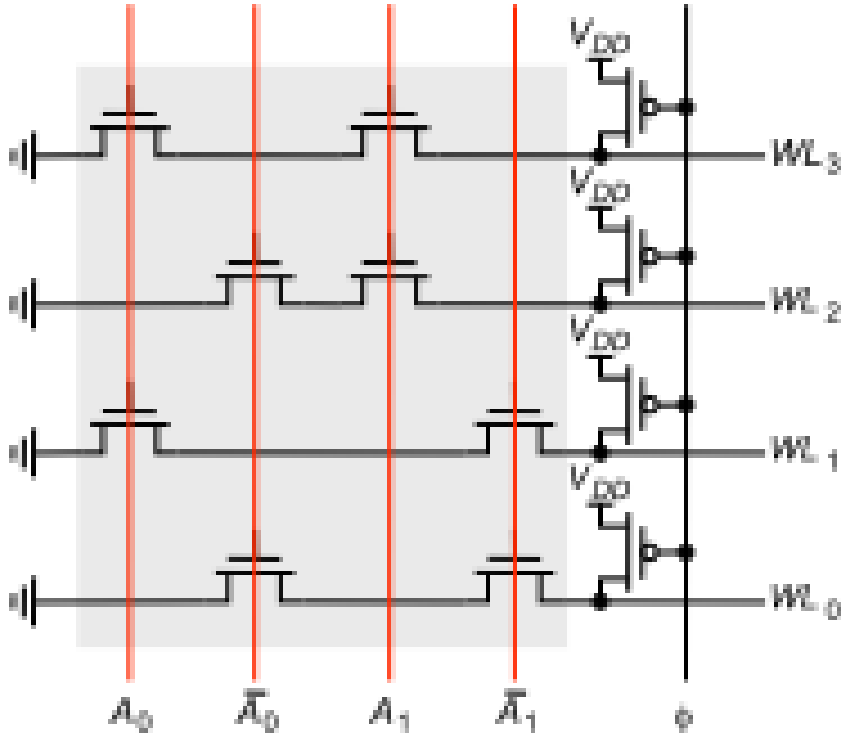
- Pitch Fitting:
  - 2-input NANDs vs. 4-input NAND.
- Switching Capacitance:
  - How many wires switch at each transition?
  - Design for power!
- Stages Before the large cap.
  - Distribution of the load along the delay.
- Conclusion:
  - Usually do as much predecoding as possible!



Alternative Solution: Dynamic Decoders



2-input NOR decoder



2-input NAND decoder

1 SRAM Memory Architecture

2 The 6T SRAM Bitcell

3 SRAM Stability

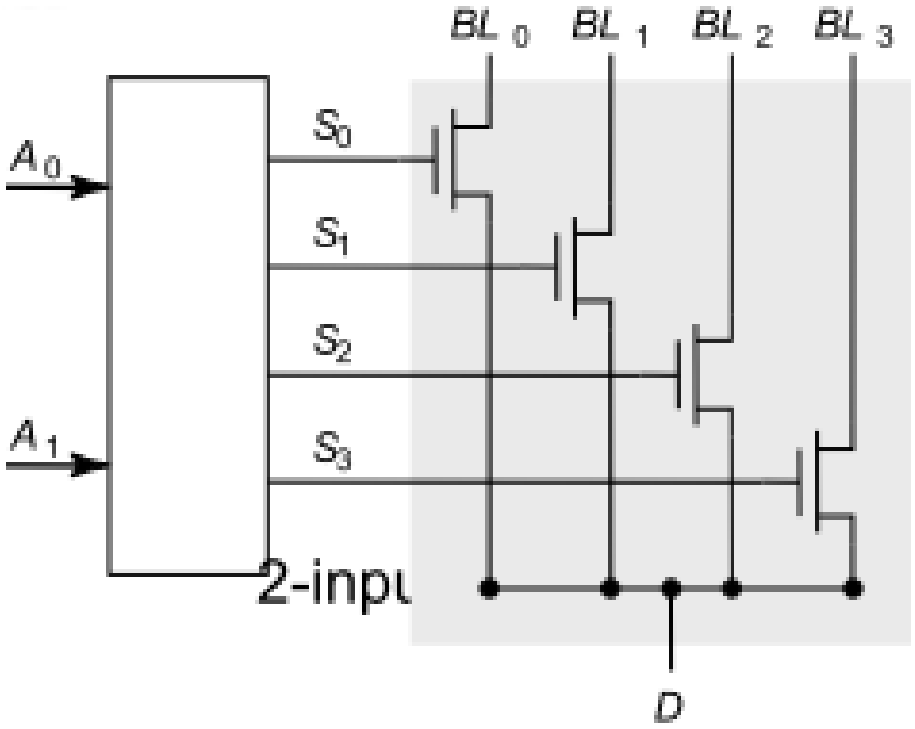
4 SNM Calculation

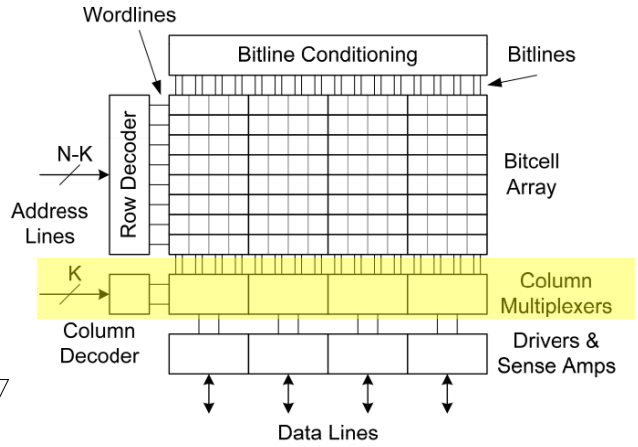
5 **Peripheral Circuits**

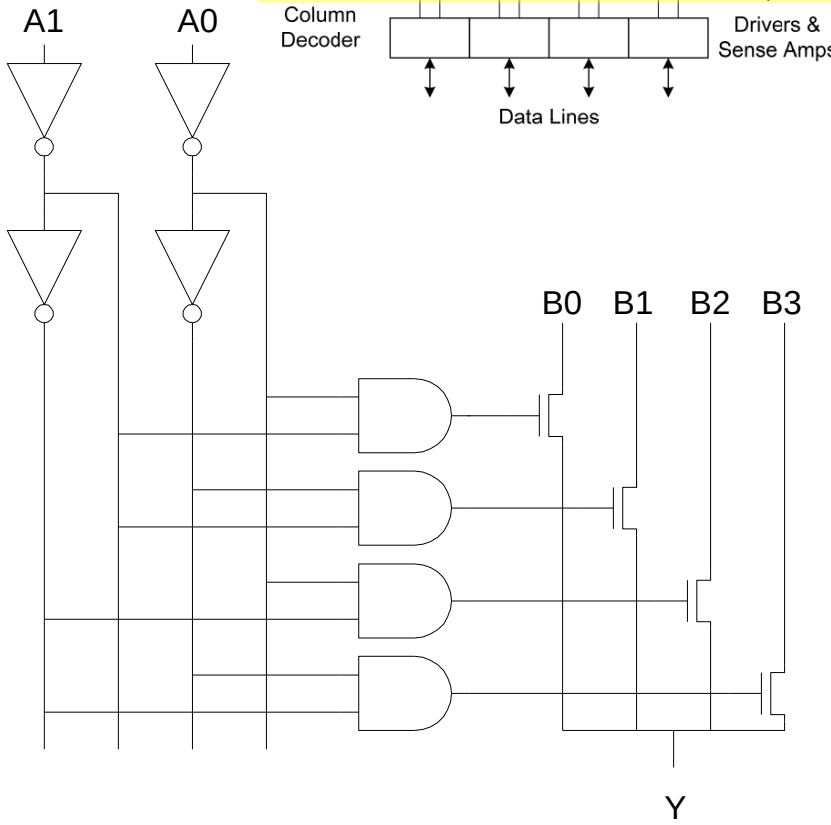
Other Peripherals

Column Multiplexer

- First option – PTL Mux with decoder
  - Fast – only 1 transistor in signal path.
  - Large transistor Count

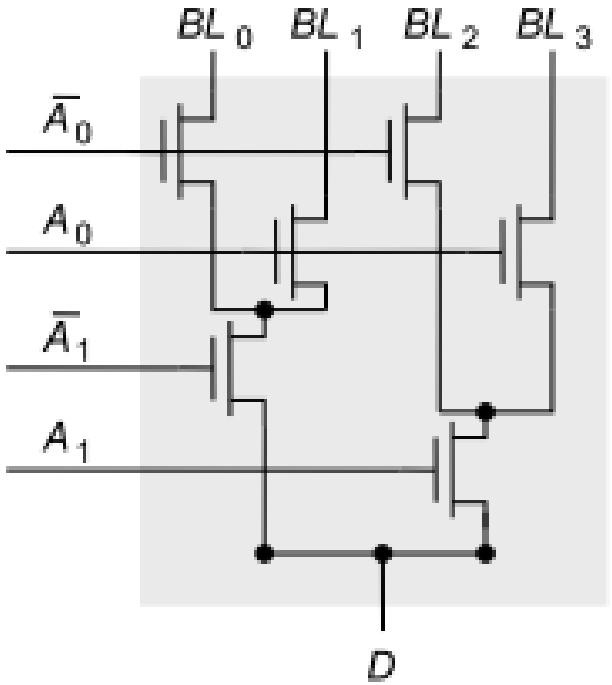


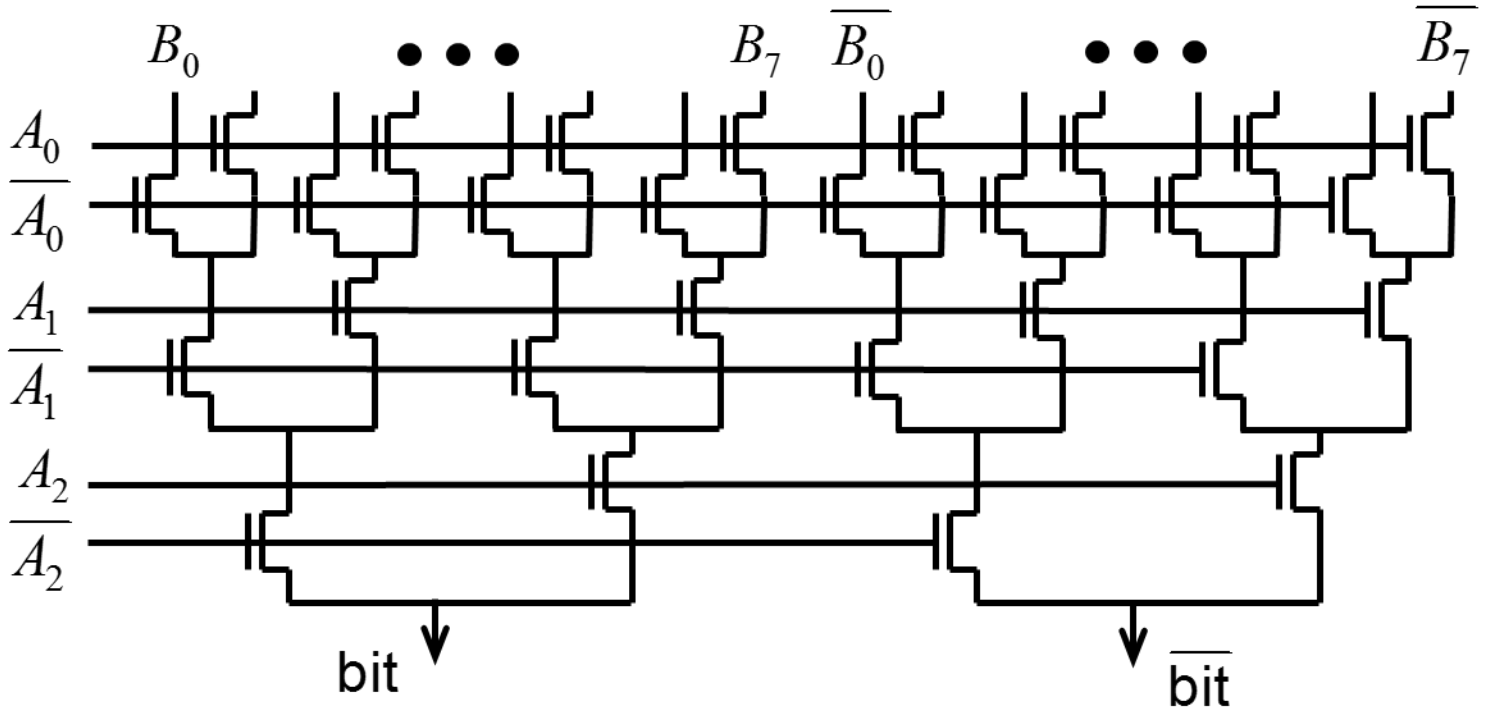




4 to 1 tree based column decoder

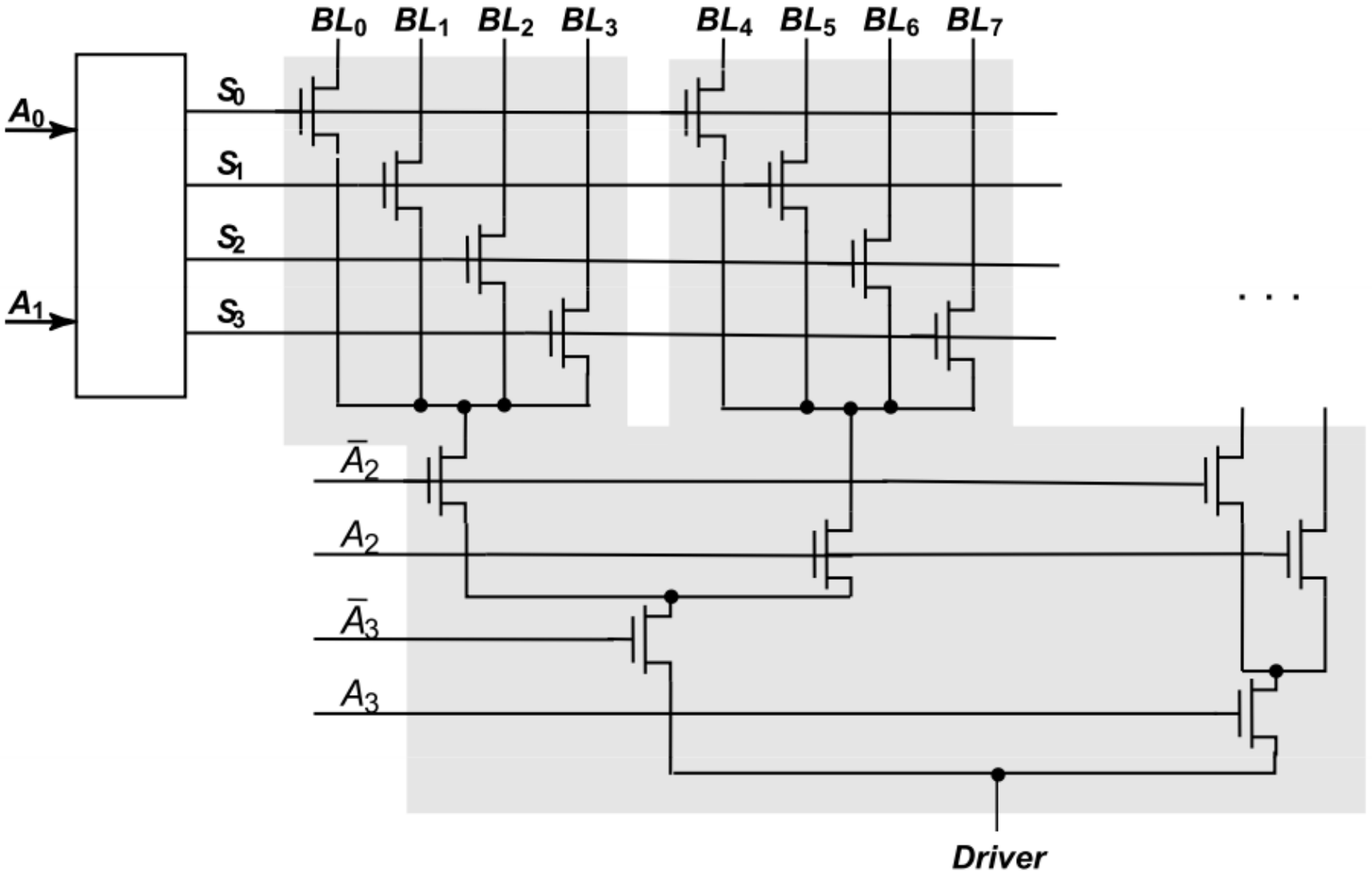
- Second option – Tree Decoder
  - For 2k:1 Mux, it uses k series transistors.
  - Delay increases quadradically
  - No external decode logic → big area reduction.





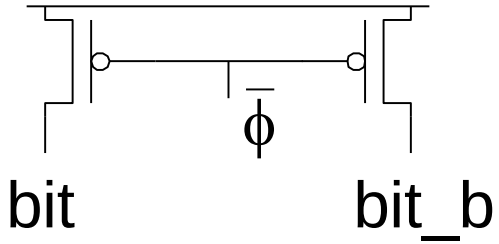
To sense Amps and Write Circuits

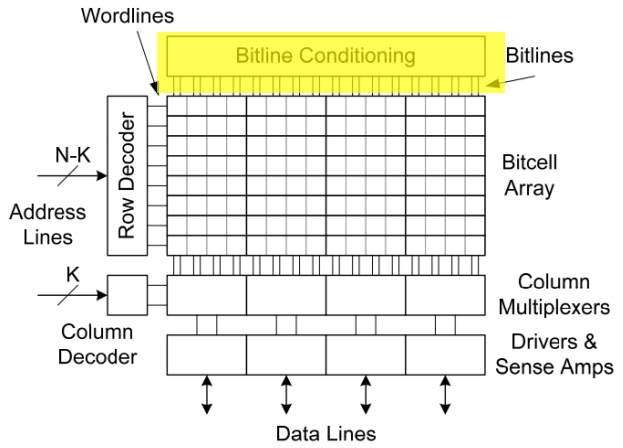
Combining the Two



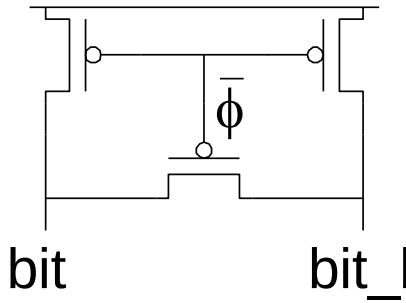
Precharge Circuitry

- Precharge bitlines high before reads





- Equalize bitlines to minimize voltage difference when using sense amplifiers



Sense Amplifiers

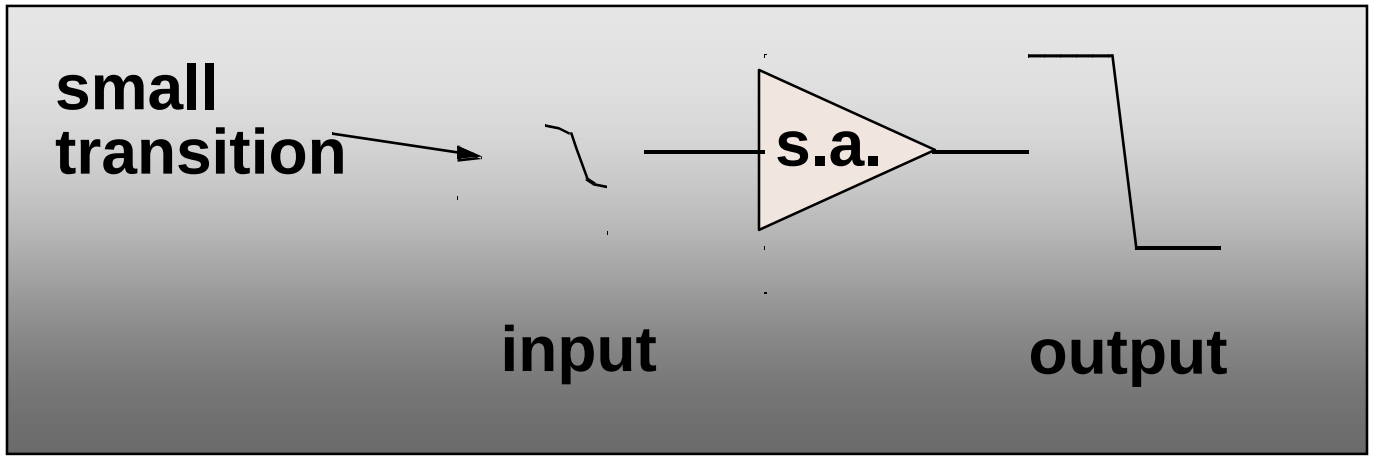
$$t_p = \frac{C \cdot \Delta V}{I_{av}}$$

make ΔV as small as possible

large

small

Idea: Use Sense Amplifer



Differential Sense Amplifier

- Non-clocked Sense Amp has high static power.
- Clocked sense amp saves power
- Requires sense_clk after enough bitline swing
- Isolation transistors cut off large bitline capacitance