

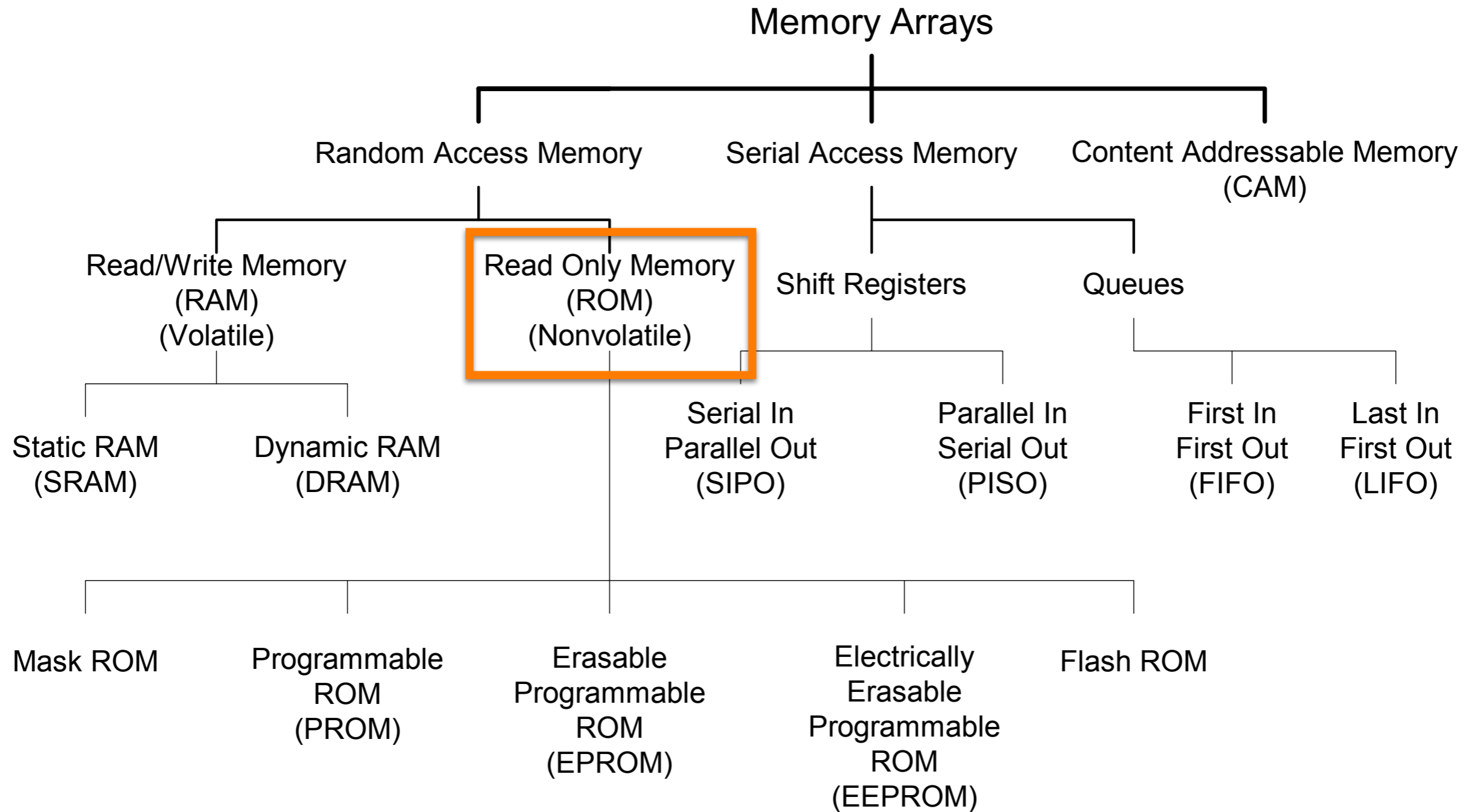
# B4. Memory—2

**Bei Yu**

**Reference:**

- **Chapter 11 Memories**
- **CMOS VLSI Design—A Circuits and Systems Perspective**
- **by H.E.Weste and D.M.Harris**

# Memory Arrays



# Read-Only Memories

- Read-Only Memories are nonvolatile
  - Retain their contents when power is removed
- Mask-programmed ROMs use one transistor per bit
  - Presence or absence determines 1 or 0

# NOR ROM

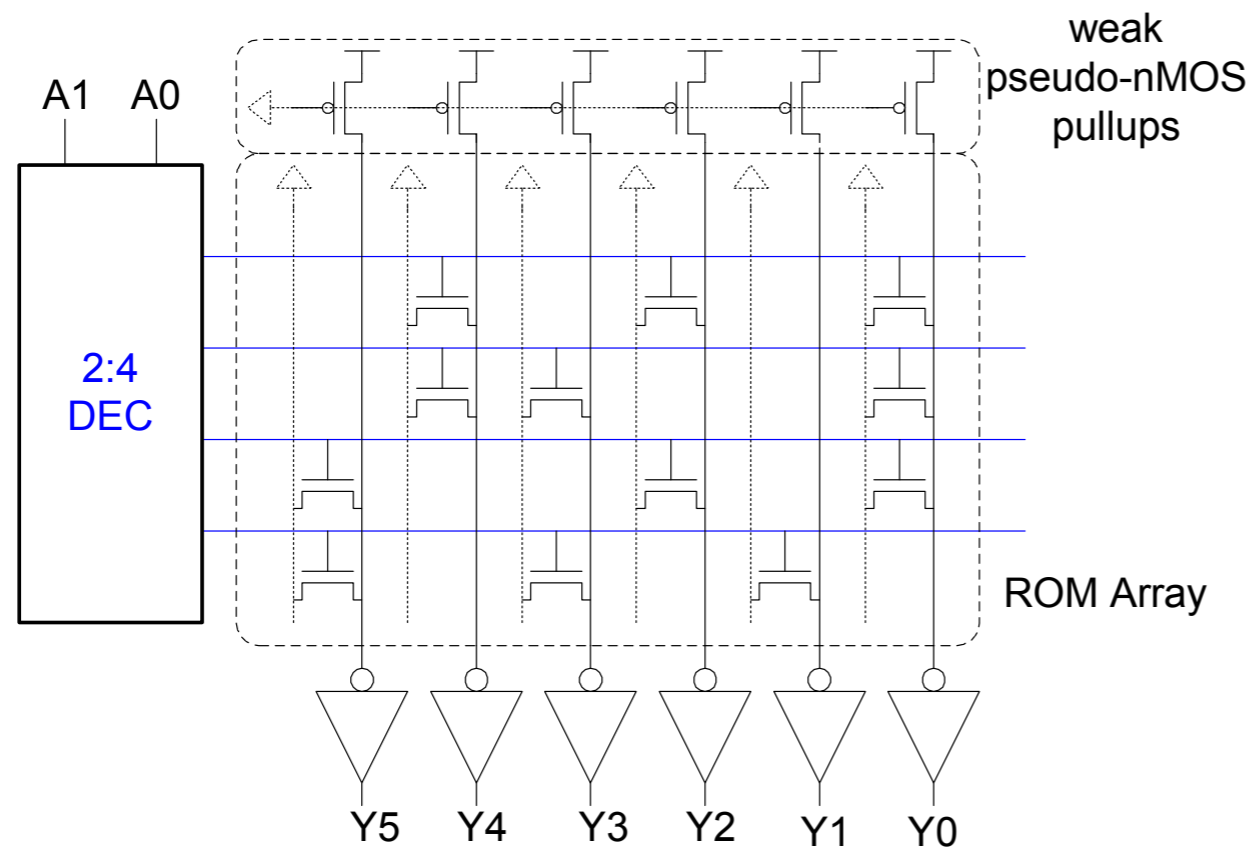
- 4-word x 6-bit NOR-ROM
  - Selected word-line high
  - Represented with dot diagram

Word 0: 010101

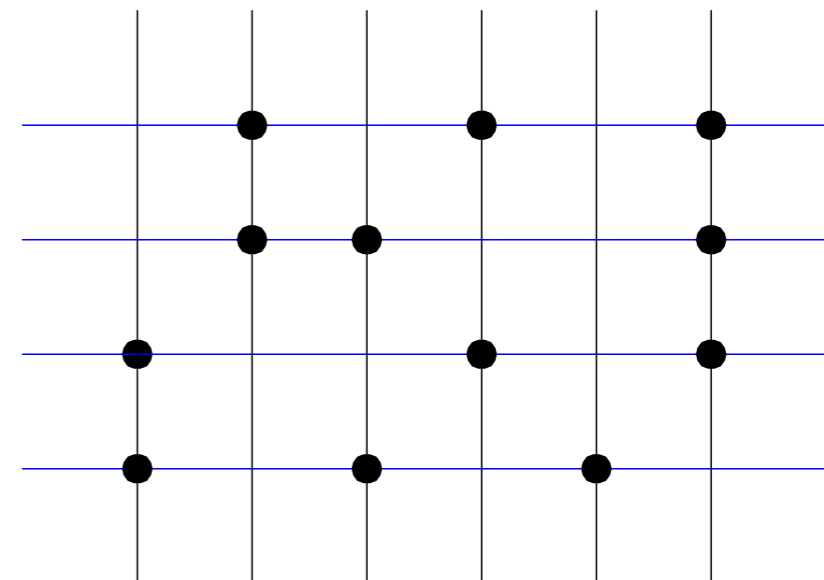
Word 1: 011001

Word 2: 100101

Word 3: 101010



Looks like 6 4-input pseudo-nMOS NORs



# EX: NOR ROM

- Draw 4-word 4-bit NOR-ROM structure and dot diagram

Word 0: **0100**

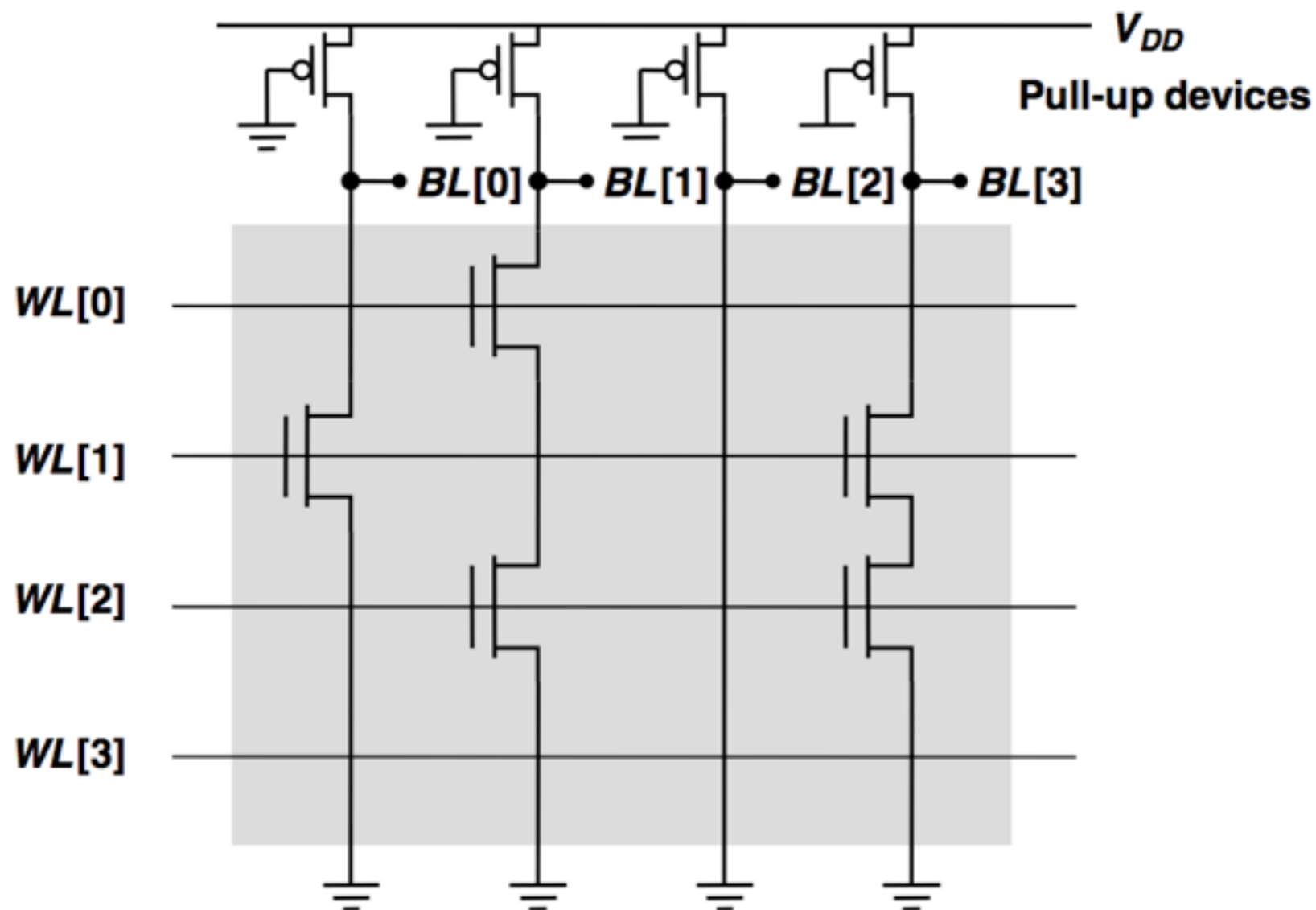
Word 1: **1001**

Word 2: **0101**

Word 3: **0000**

# NAND ROM

- 4-word x 4-bit NAND-ROM
  - All word-lines high with exception of selected row



# EX. NAND ROM

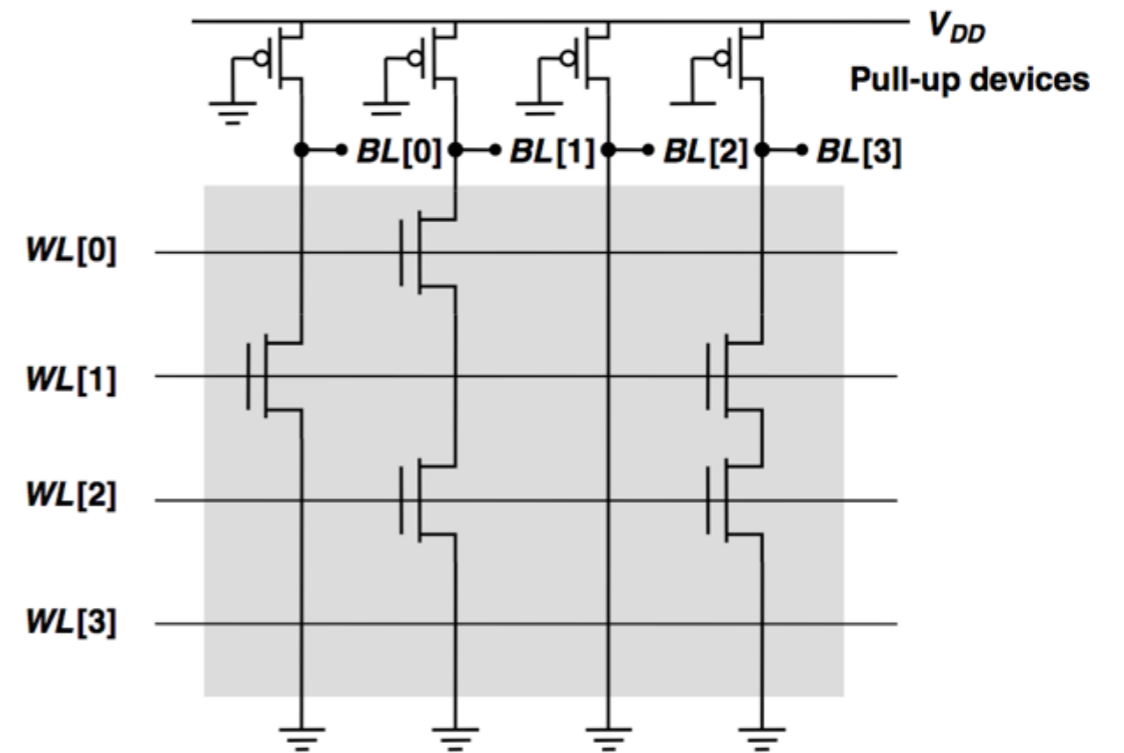
- What's its function?

$WL[0]=0$ :

$WL[1]=0$ :

$WL[2]=0$ :

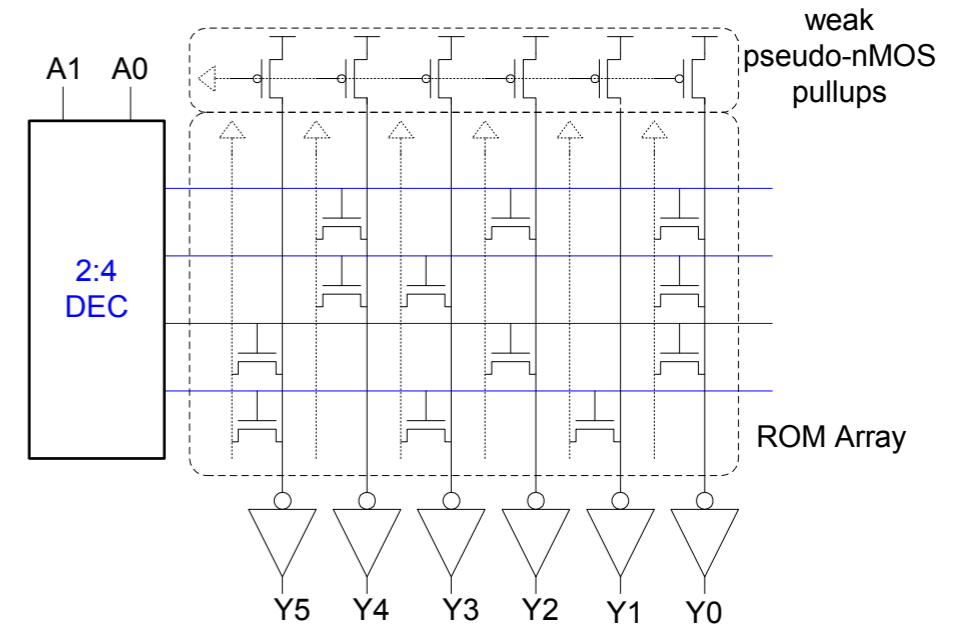
$WL[3]=0$ :



# NOR ROM v.s. NAND ROM

- **NOR ROM:**

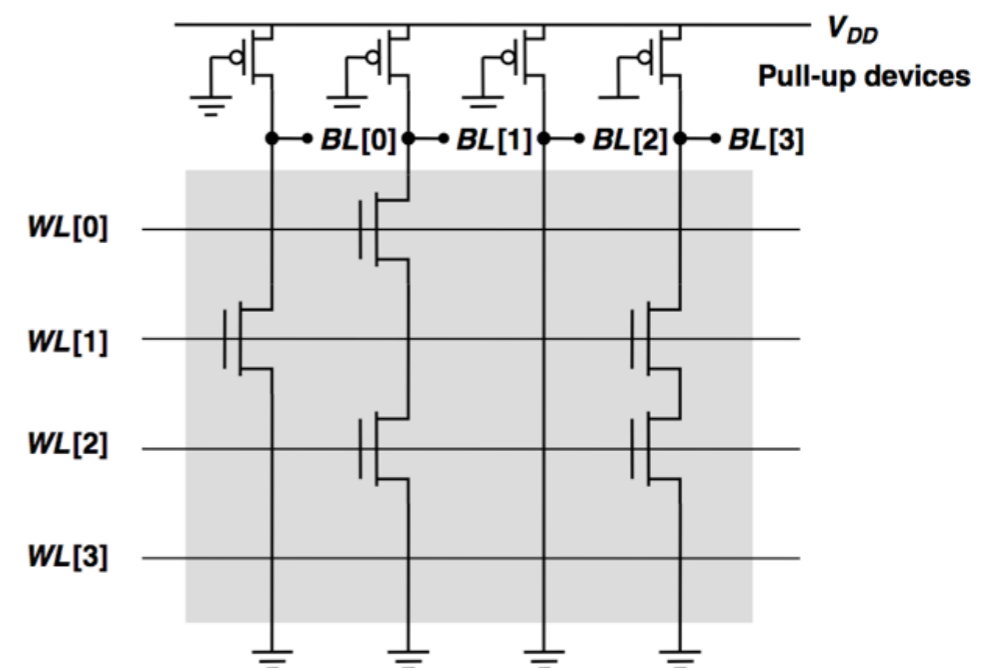
- (+) Faster
- (-) Larger Area (VDD lines)



- **NAND ROM:**

- (+) High density, small area
- (-) Slower

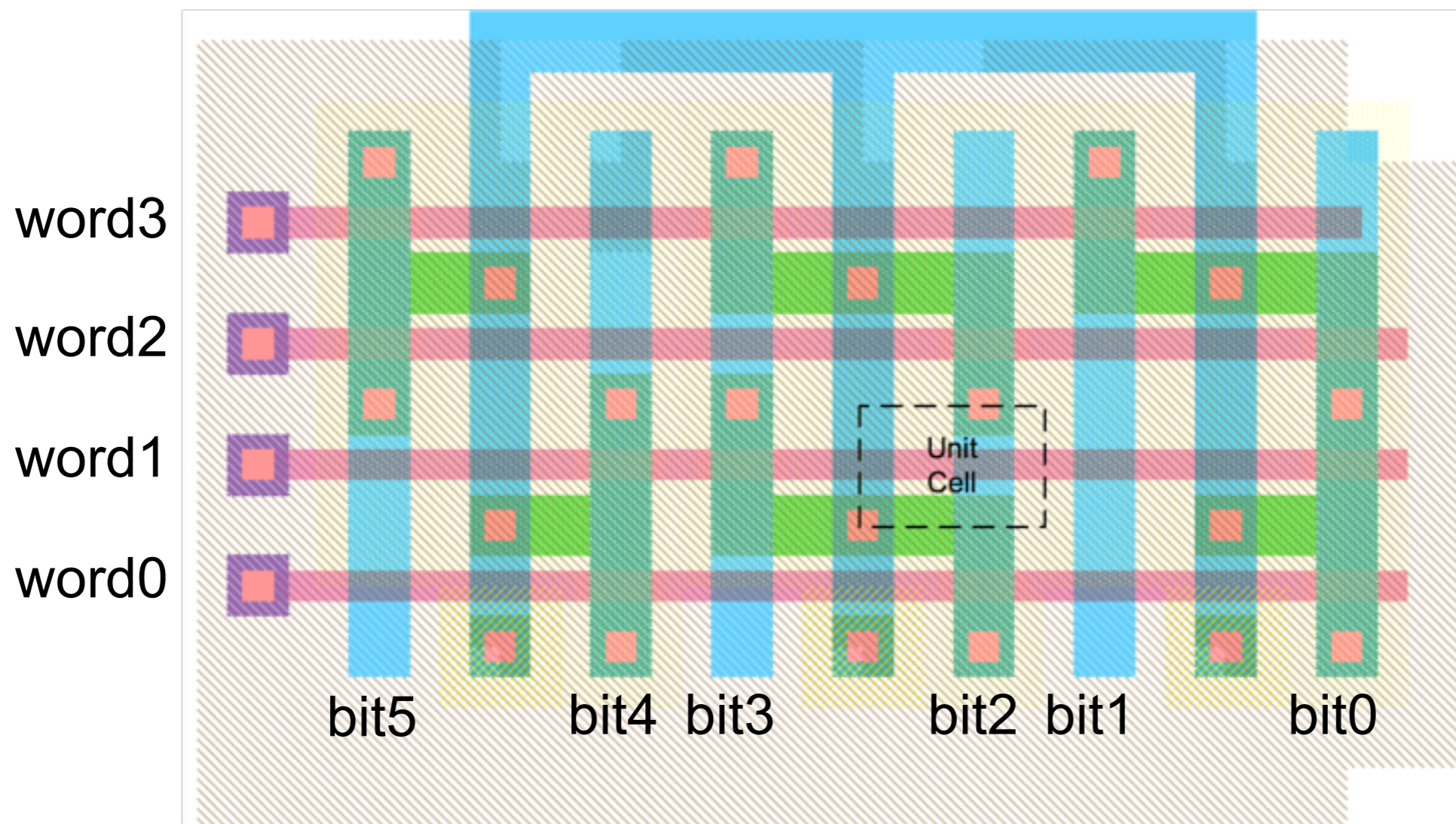
delay grows quadratically with the number of series transistors discharging the bitline.





# NOR ROM Array Layout\*

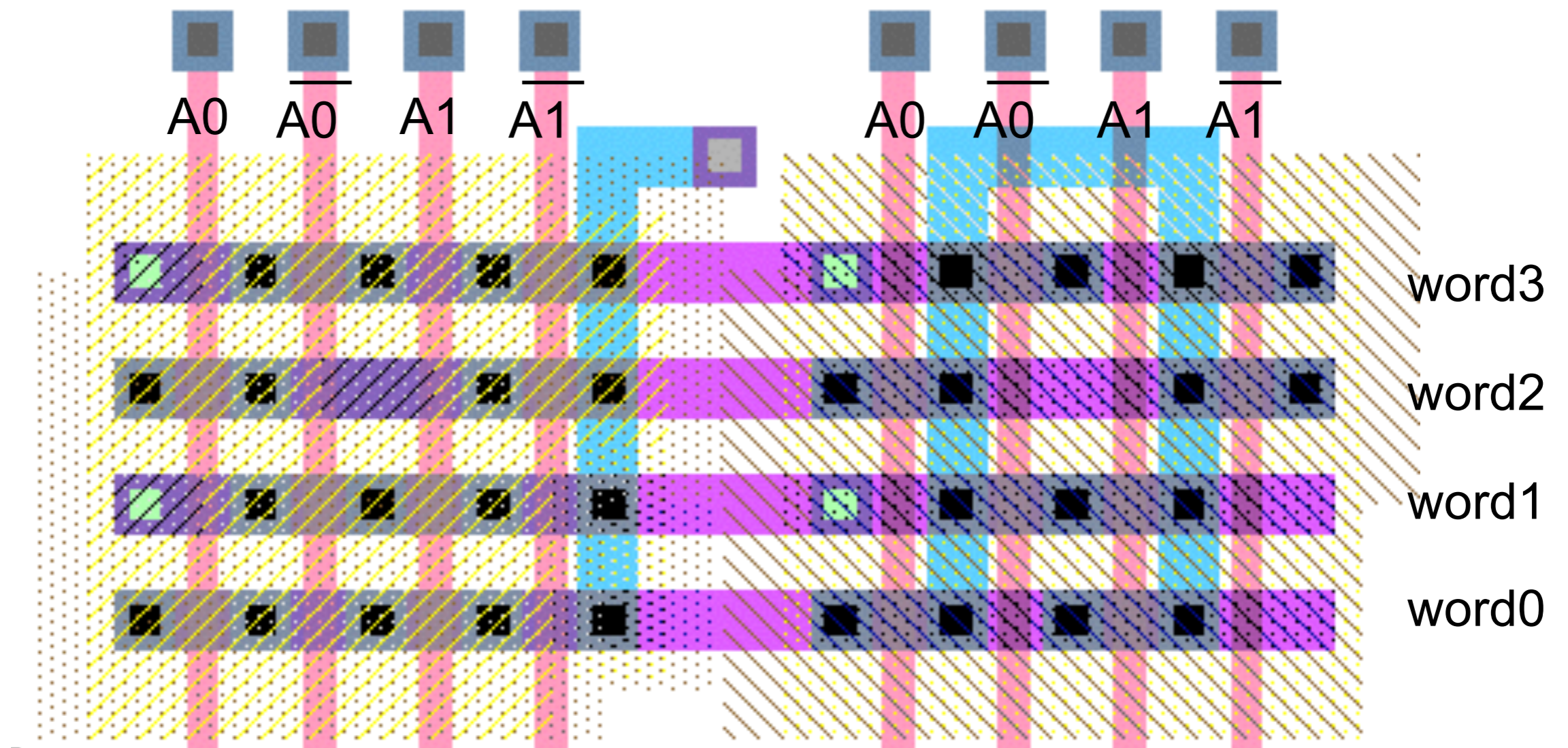
- Unit cell is  $12 \times 8 \lambda$  (about 1/10 size of SRAM)





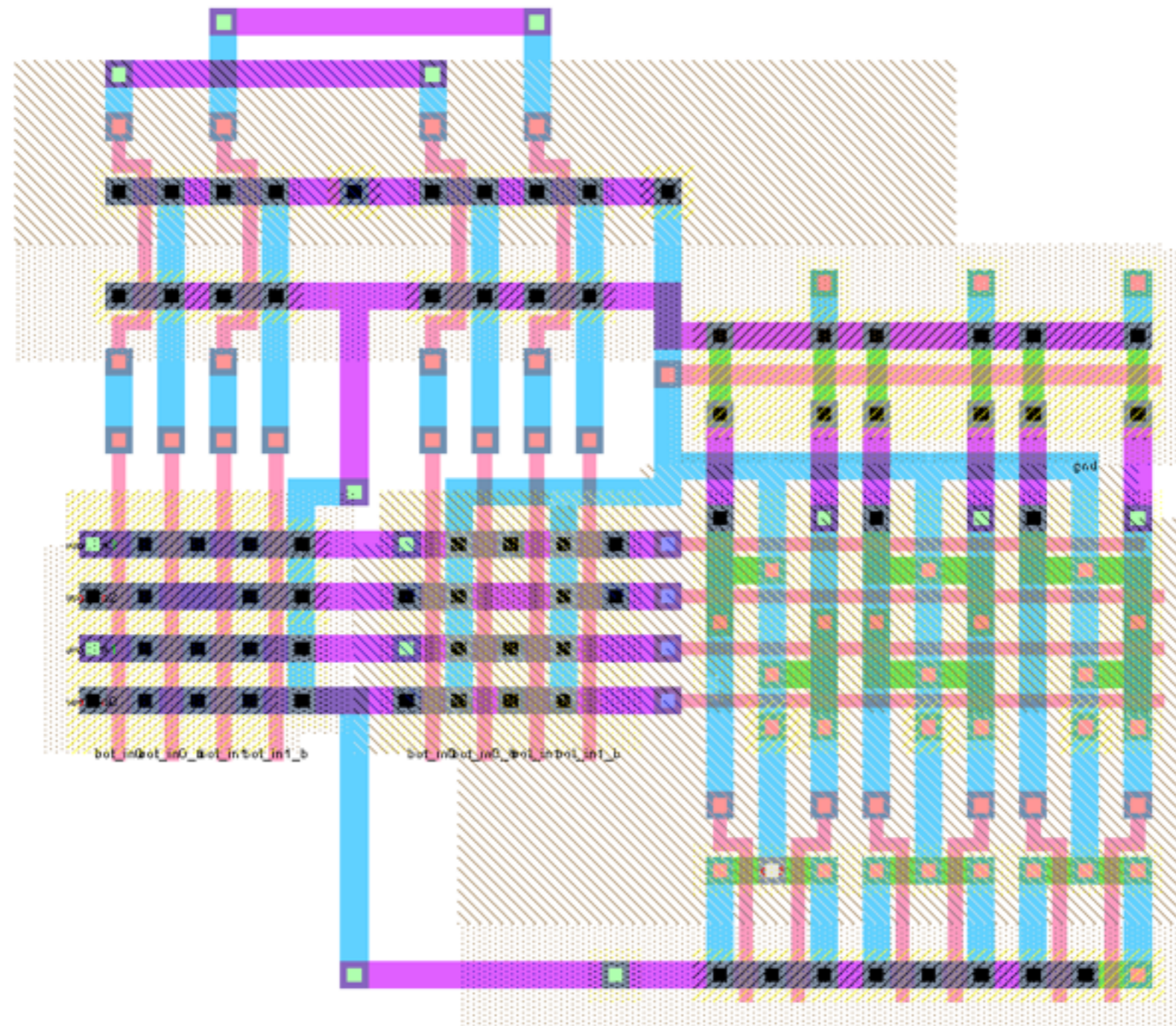
# Row Decoders\*

- ROM row decoders must pitch-match with ROM
  - Only a single track per word!



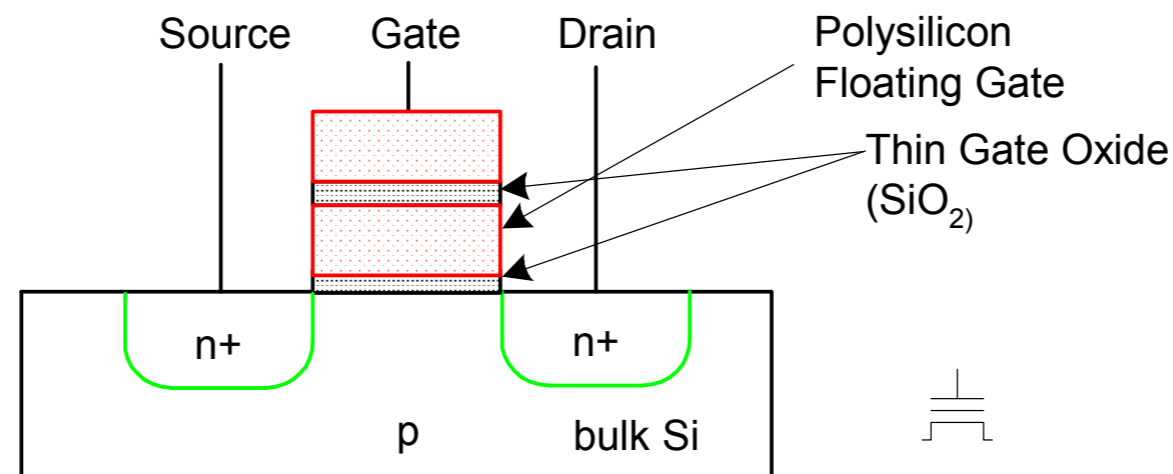


# Complete ROM Layout\*



# PROMs and EPROMs\*

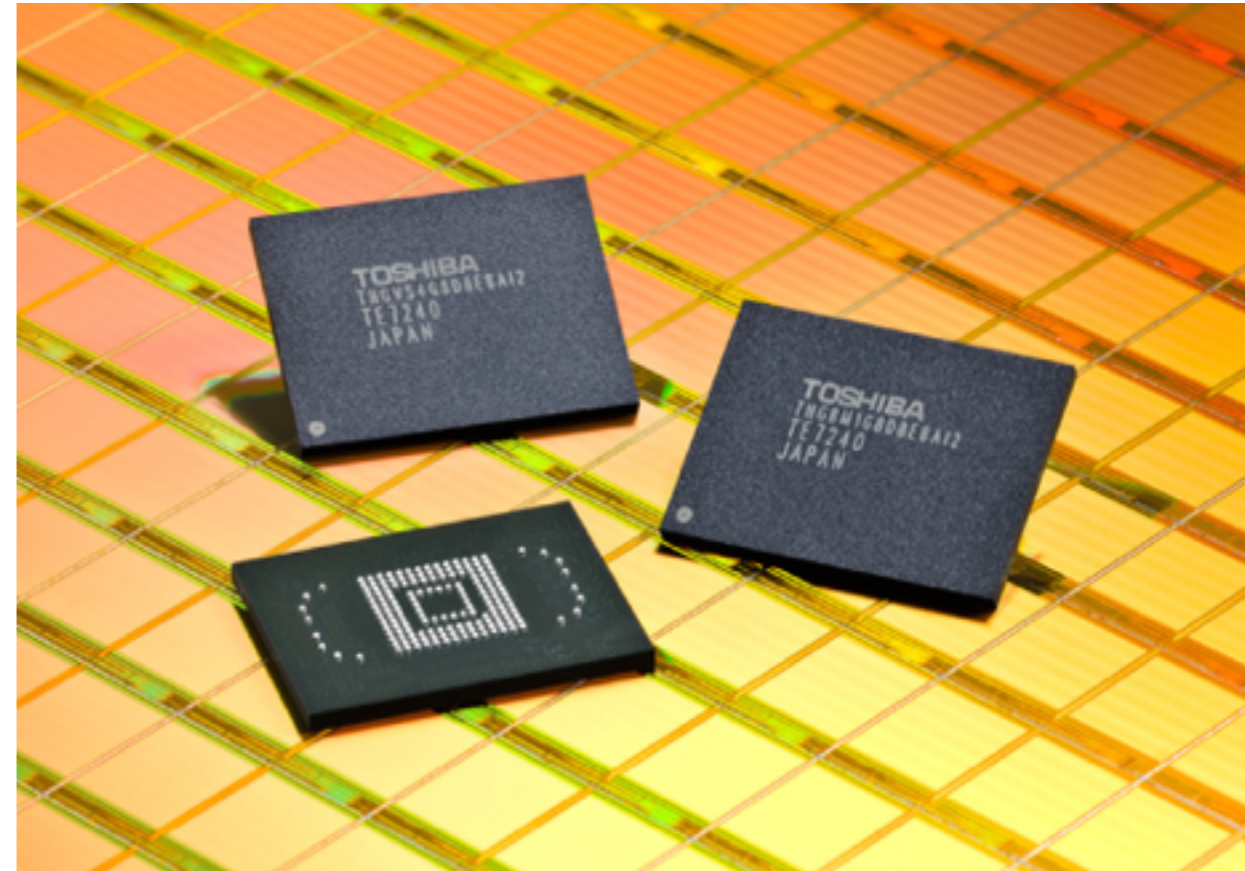
- Programmable ROMs
  - Build array with transistors at every site
  - Burn out fuses to disable unwanted transistors
- Electrically Programmable ROMs
  - Use floating gate to turn off unwanted transistors
  - EPROM, EEPROM, Flash





# NOR / NAND Flash Memory\*

- NOR flash: Intel 1988
- NAND flash: Toshiba 1989

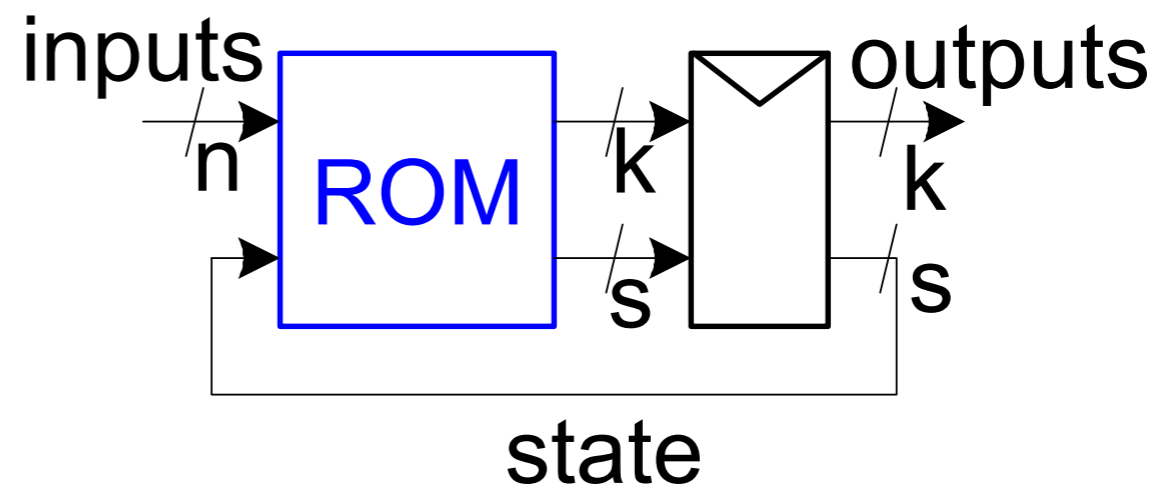
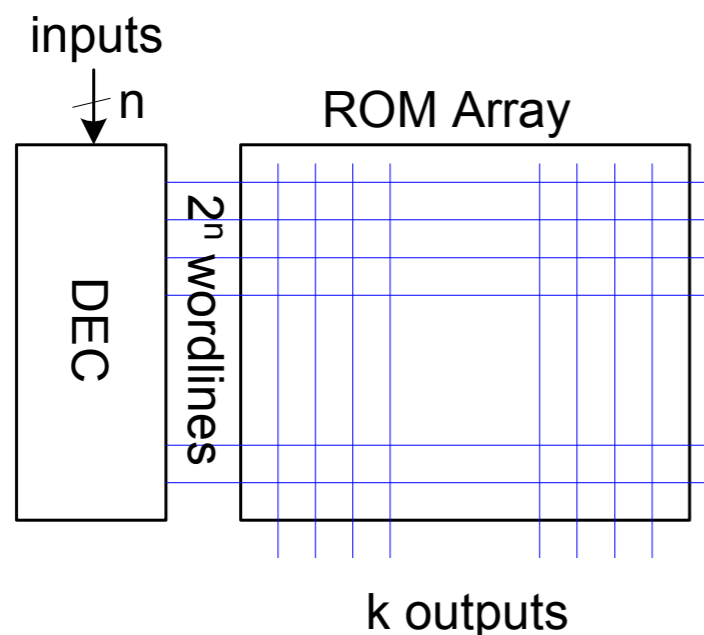


[Toshiba'08]

- NOR: faster, more expensive
- NAND: higher density

# Building Logic with ROMs

- ROM as lookup table containing truth table
  - n inputs, k outputs requires  $2^n$  words x k bits
  - Changing function is easy – reprogram ROM
- **Finite State Machine**
  - n inputs, k outputs, s bits of state
  - Build with  $2^{n+s}$  x (k+s) bit ROM and (k+s) bit reg



# Example: RoboAnt

Let's build an Ant

**Sensors:** Antennae

(L,R) – 1 when in contact

**Actuators:** Legs

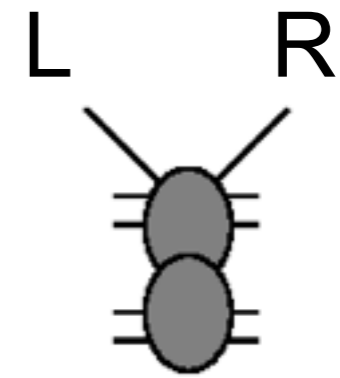
Forward step F

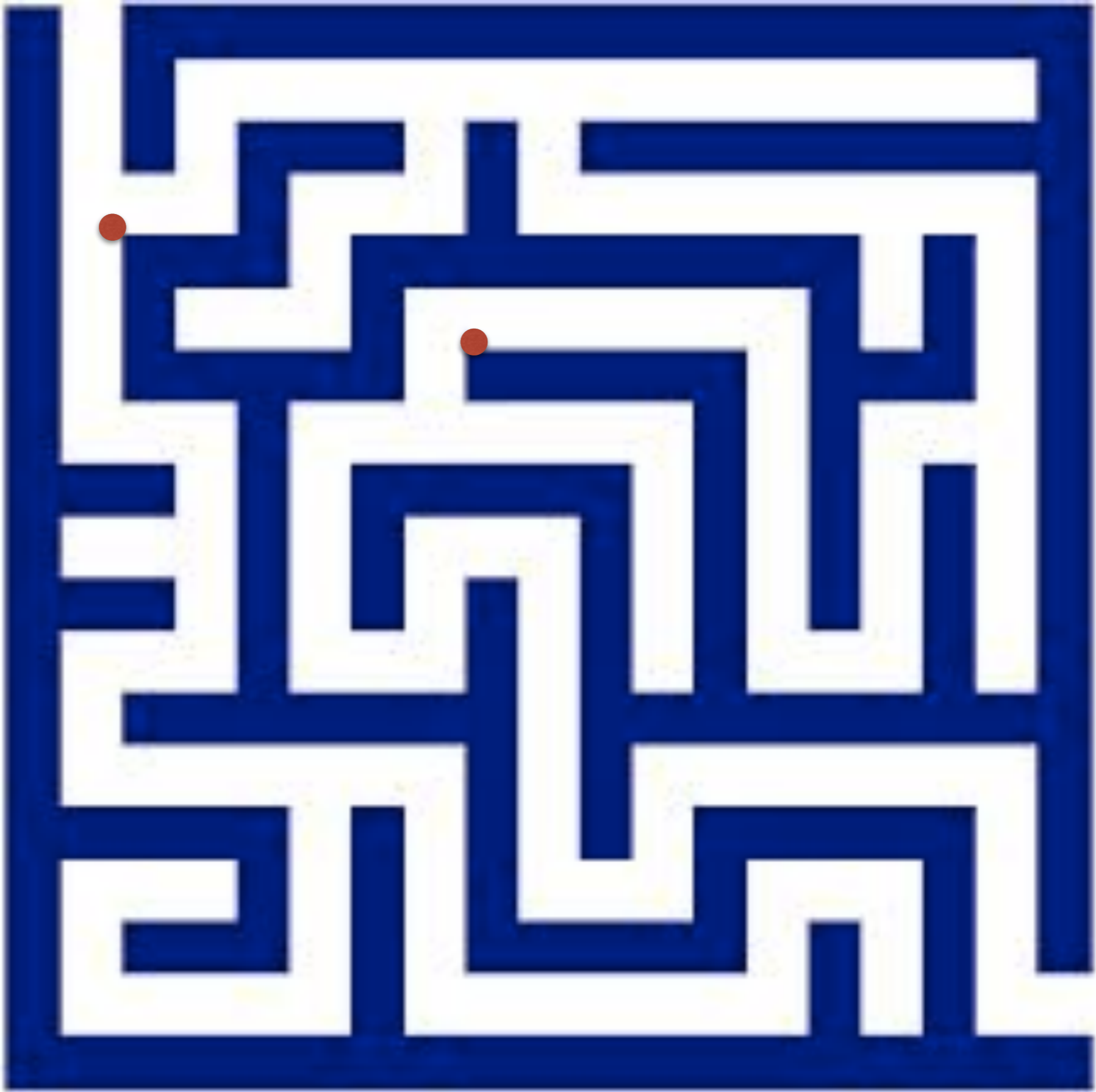
Ten degree turns TL, TR

**Goal:** make our ant smart enough to get out of a maze

**Strategy:** keep right antenna on wall

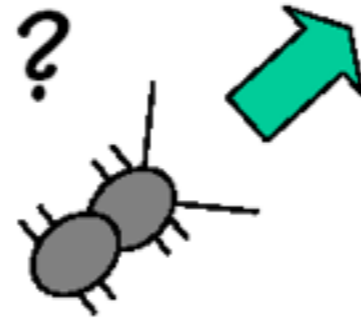
*(RoboAnt adapted from MIT 6.004 2002 OpenCourseWare by Ward and Terman)*



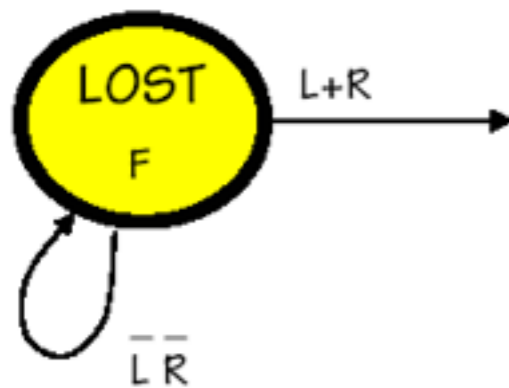




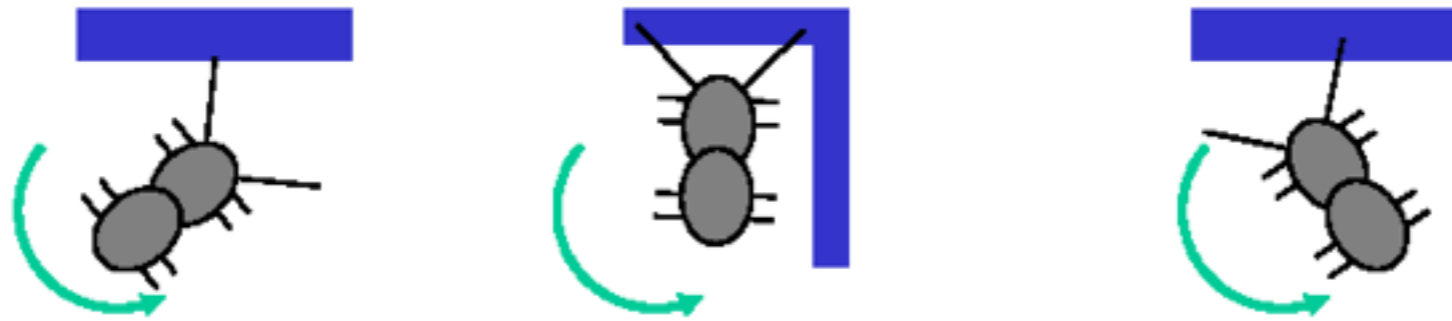
# Lost in space



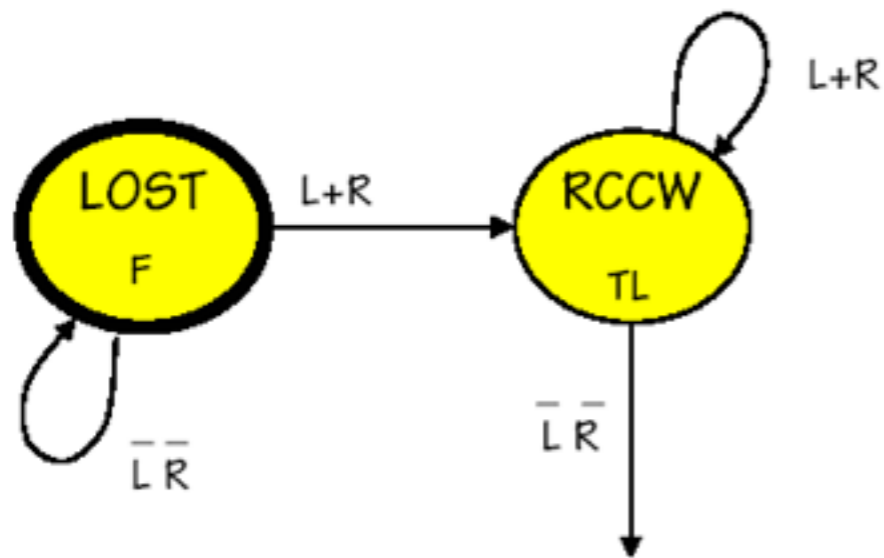
- Action: go forward until we hit something
  - Initial state



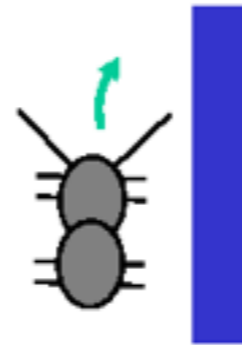
# Bonk!!!



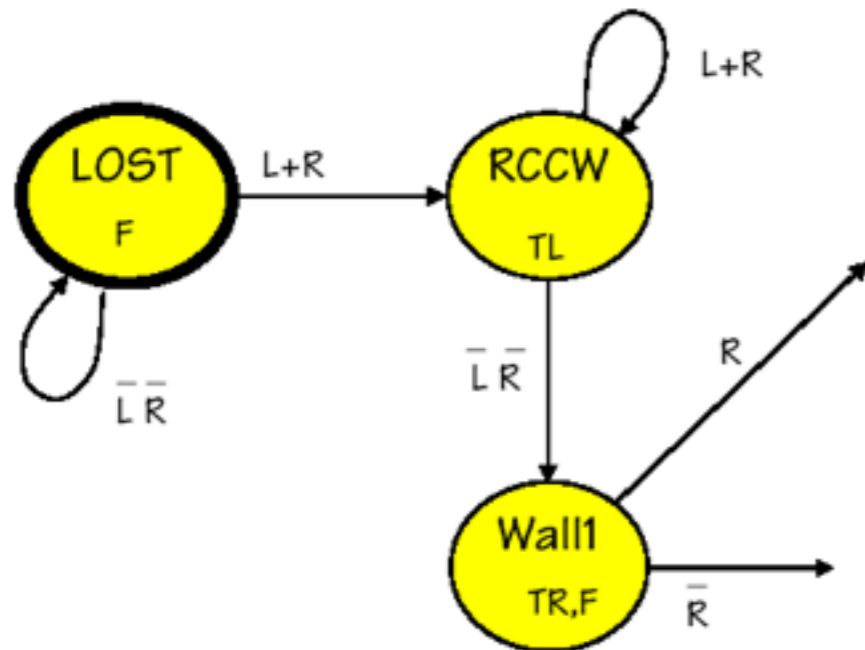
- Action: turn left (rotate counterclockwise)
  - Until we don't touch anymore



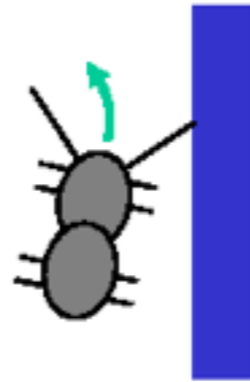
# A little to the right



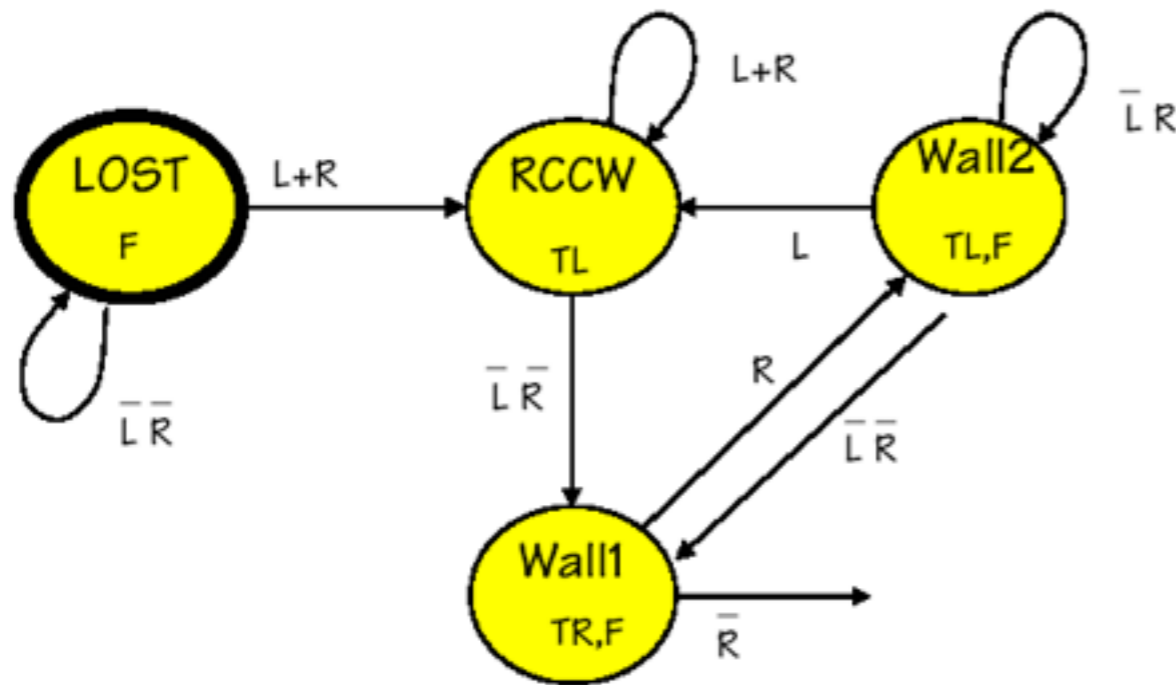
- Action: step forward and turn right a little
  - Looking for wall



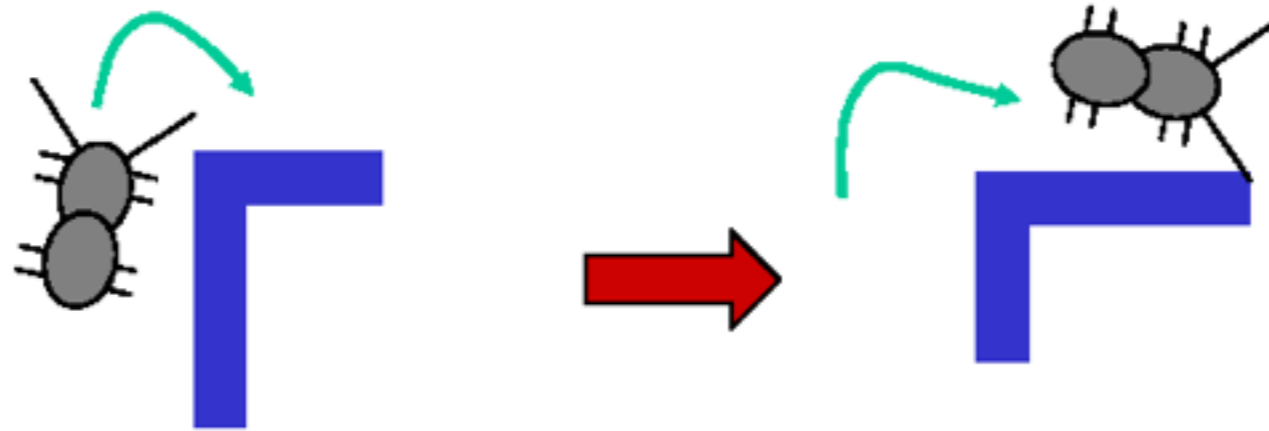
# Then a little to the right



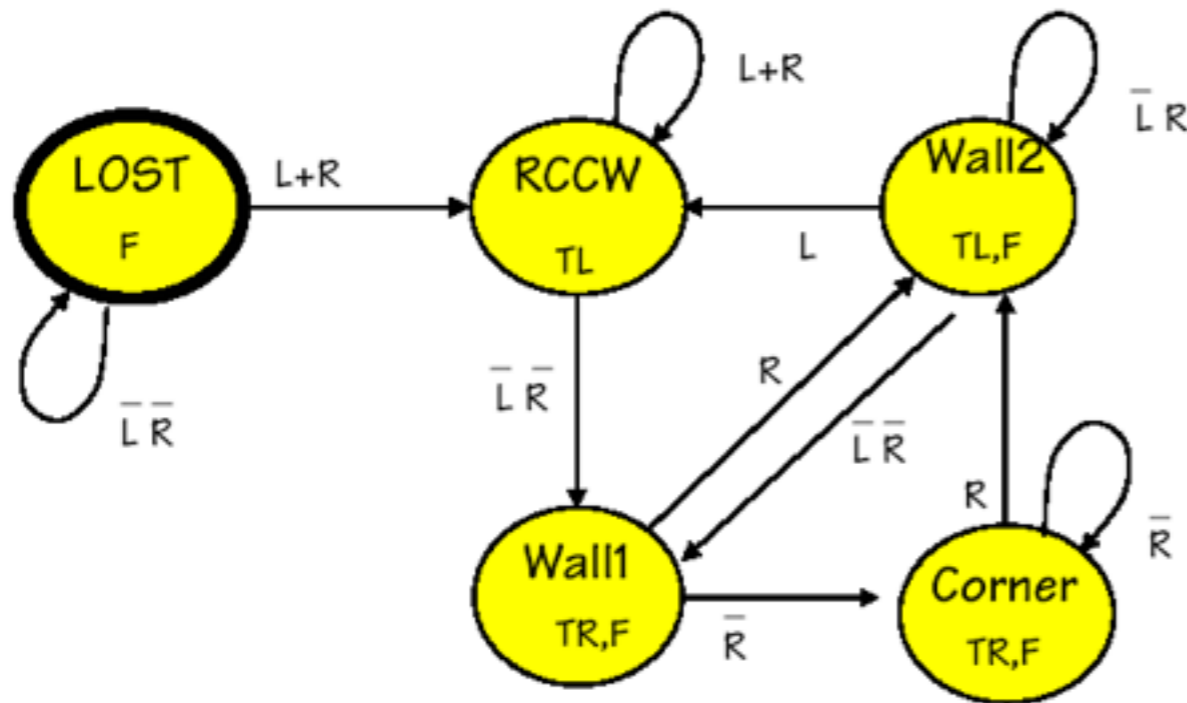
- Action: step and turn left a little, until not touching



# Whoops – a corner!

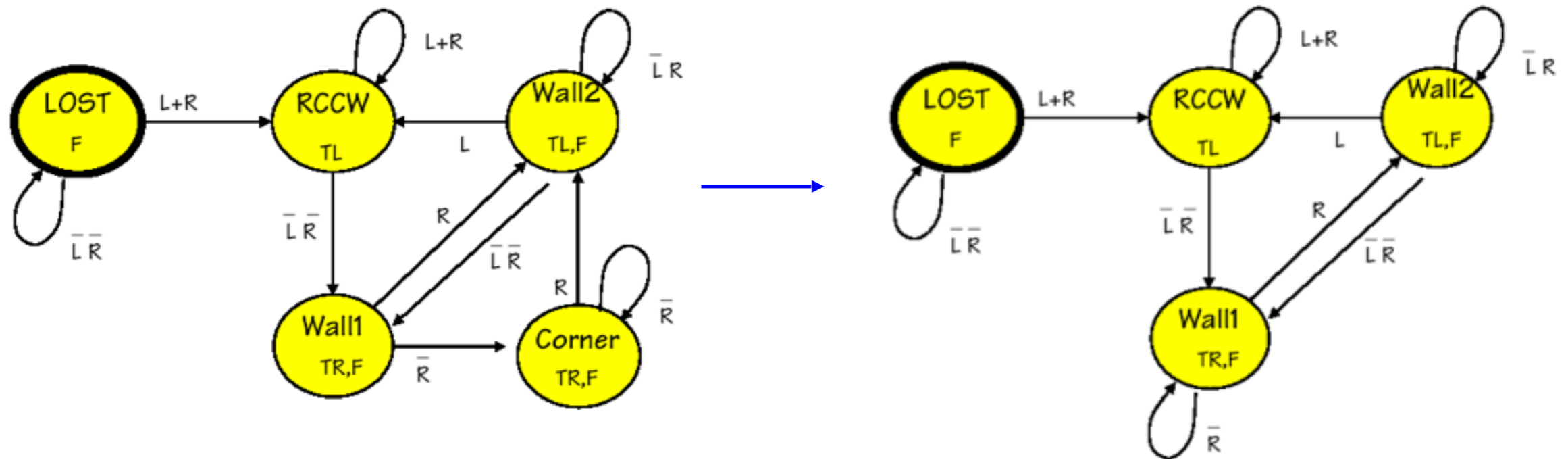


- Action: step and turn right until hitting next wall



# Simplification

- Merge equivalent states where possible



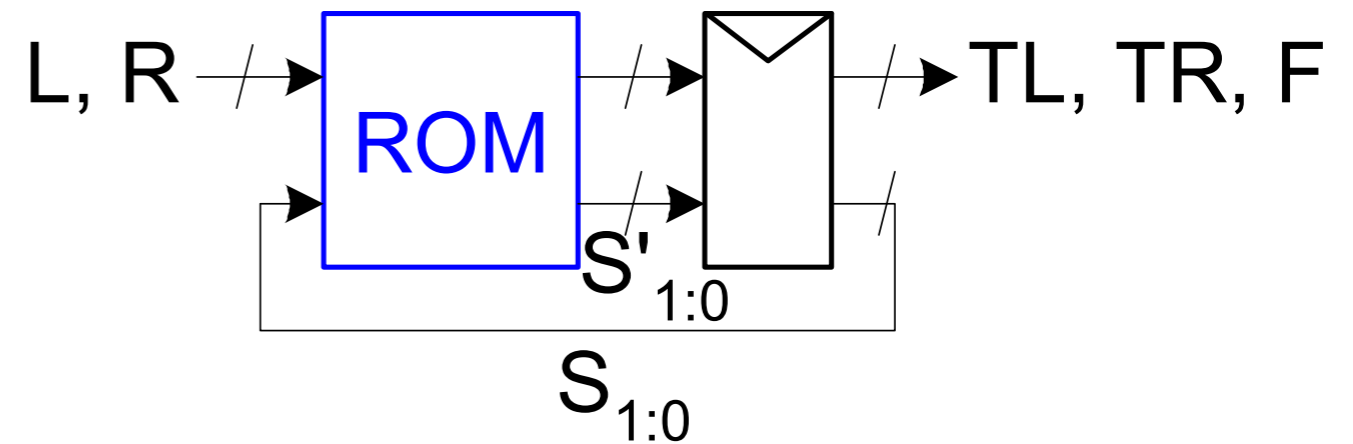
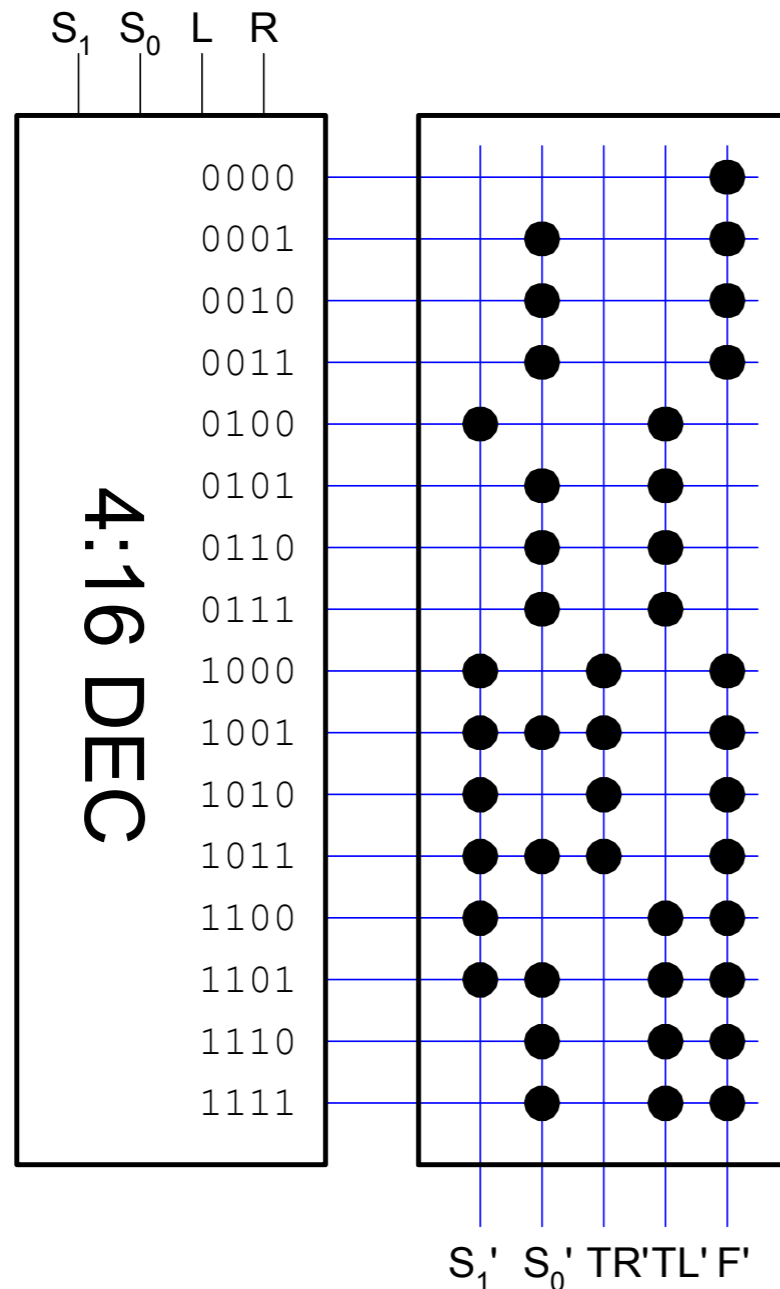
# State Transition Table

Current state      Inputs      Next state      Output values

	$S_{1:0}$	L	R	$S_{1:0}'$	TR	TL	F
Lost	00	0	0	00	0	0	1
	00	1	X	01	0	0	1
	00	0	1	01	0	0	1
RCCW	01	1	X	01	0	1	0
	01	0	1	01	0	1	0
	01	0	0	10	0	1	0
Wall1	10	X	0	10	1	0	1
	10	X	1	11	1	0	1
Wall2	11	1	X	01	0	1	1
	11	0	0	10	0	1	1
	11	0	1	11	0	1	1

# ROM Implementation

- 16-word x 5 bit ROM



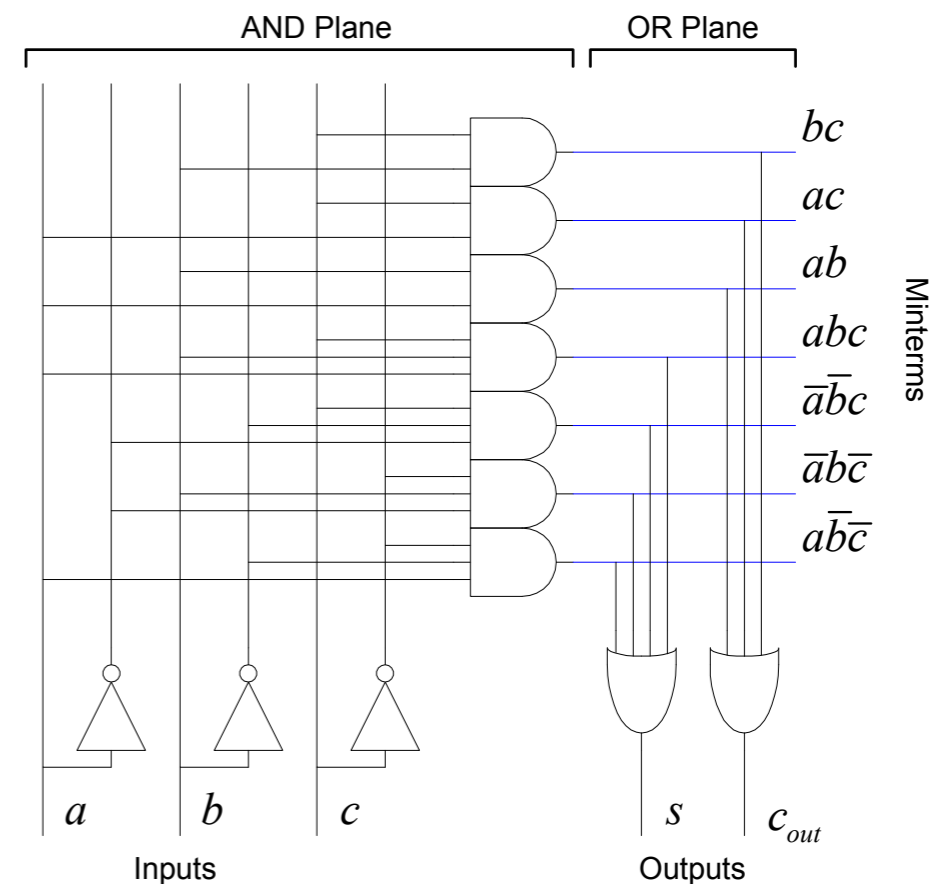


# PLAs

- A *Programmable Logic Array* performs any function in sum-of-products form.
- *Literals*: inputs & complements
- *Products / Minterms*: AND of literals
- *Outputs*: OR of Minterms
- Example: Full Adder

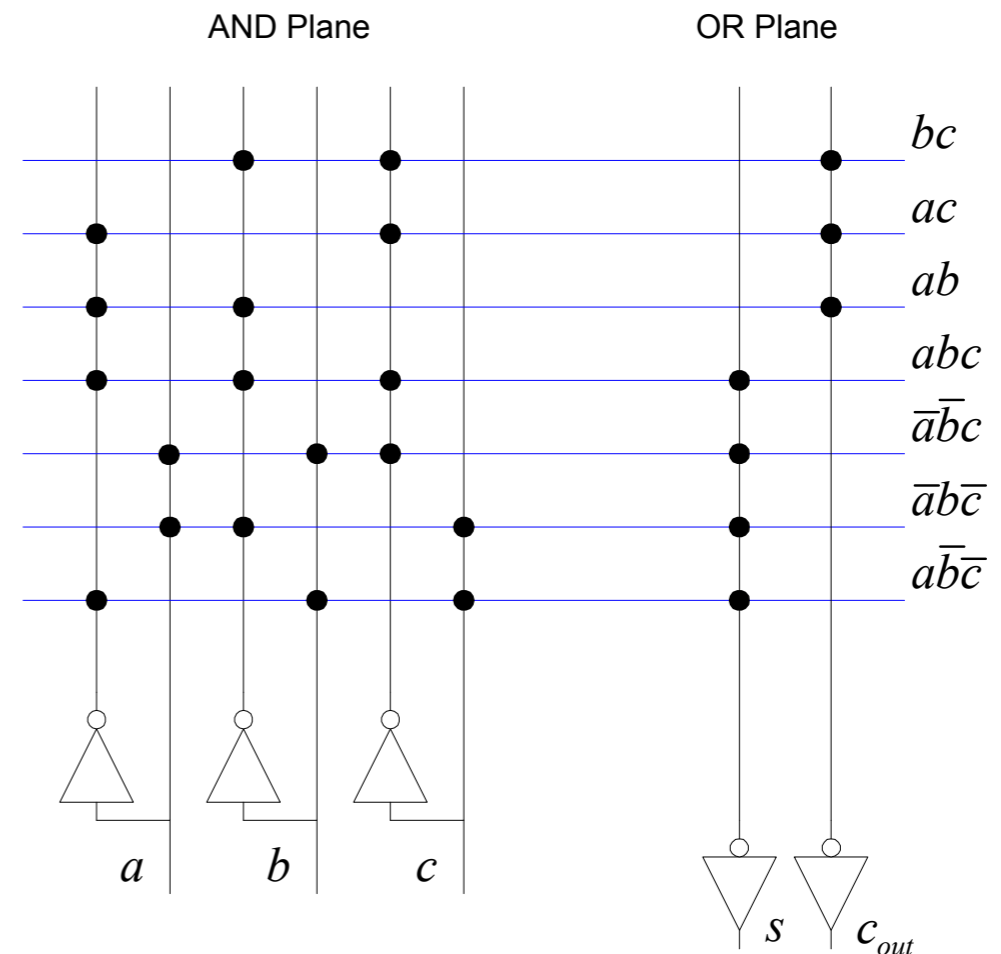
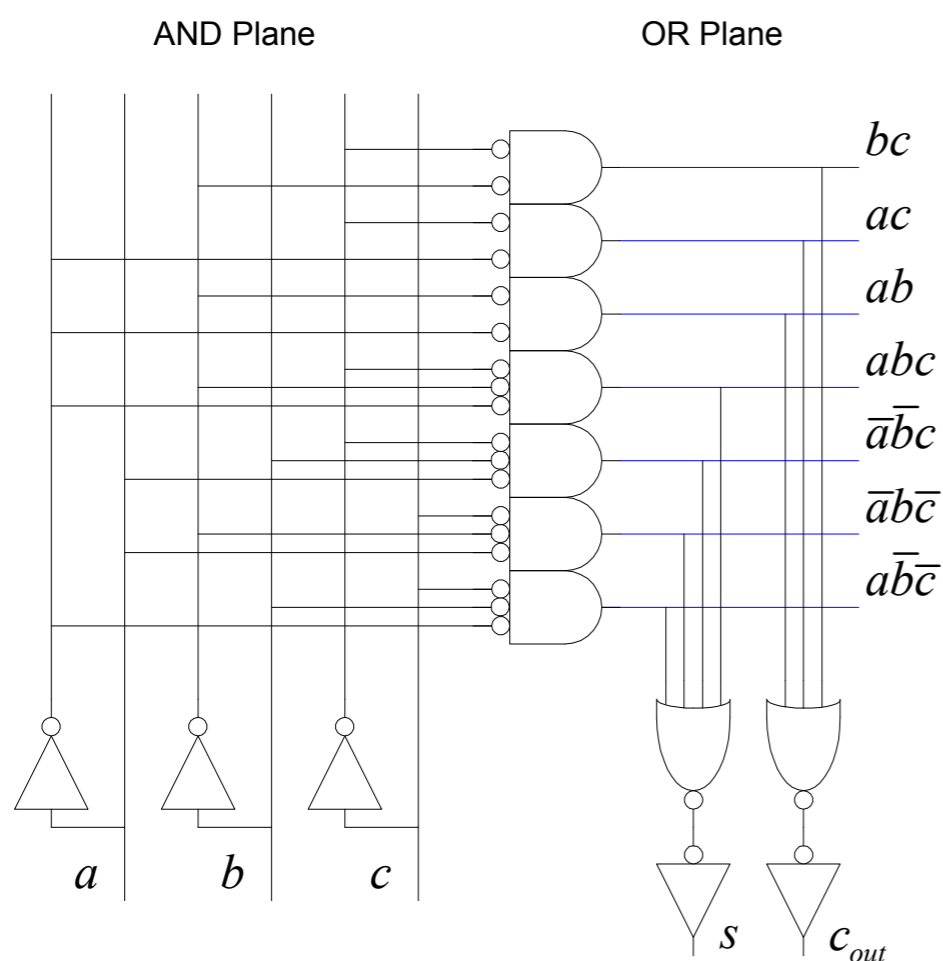
$$s = a\bar{b}\bar{c} + \bar{a}b\bar{c} + \bar{a}\bar{b}c + abc$$

$$c_{out} = ab + bc + ac$$

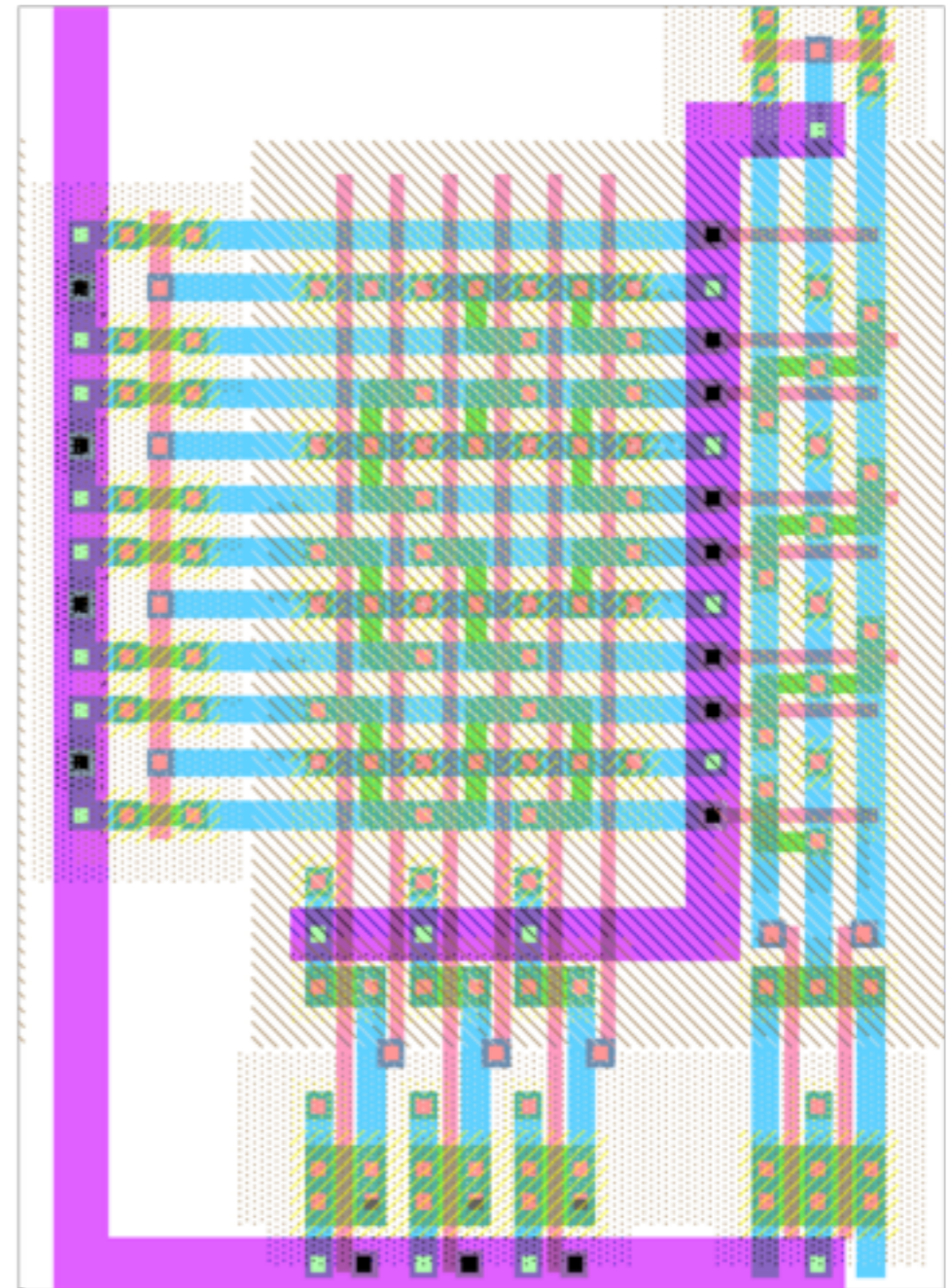
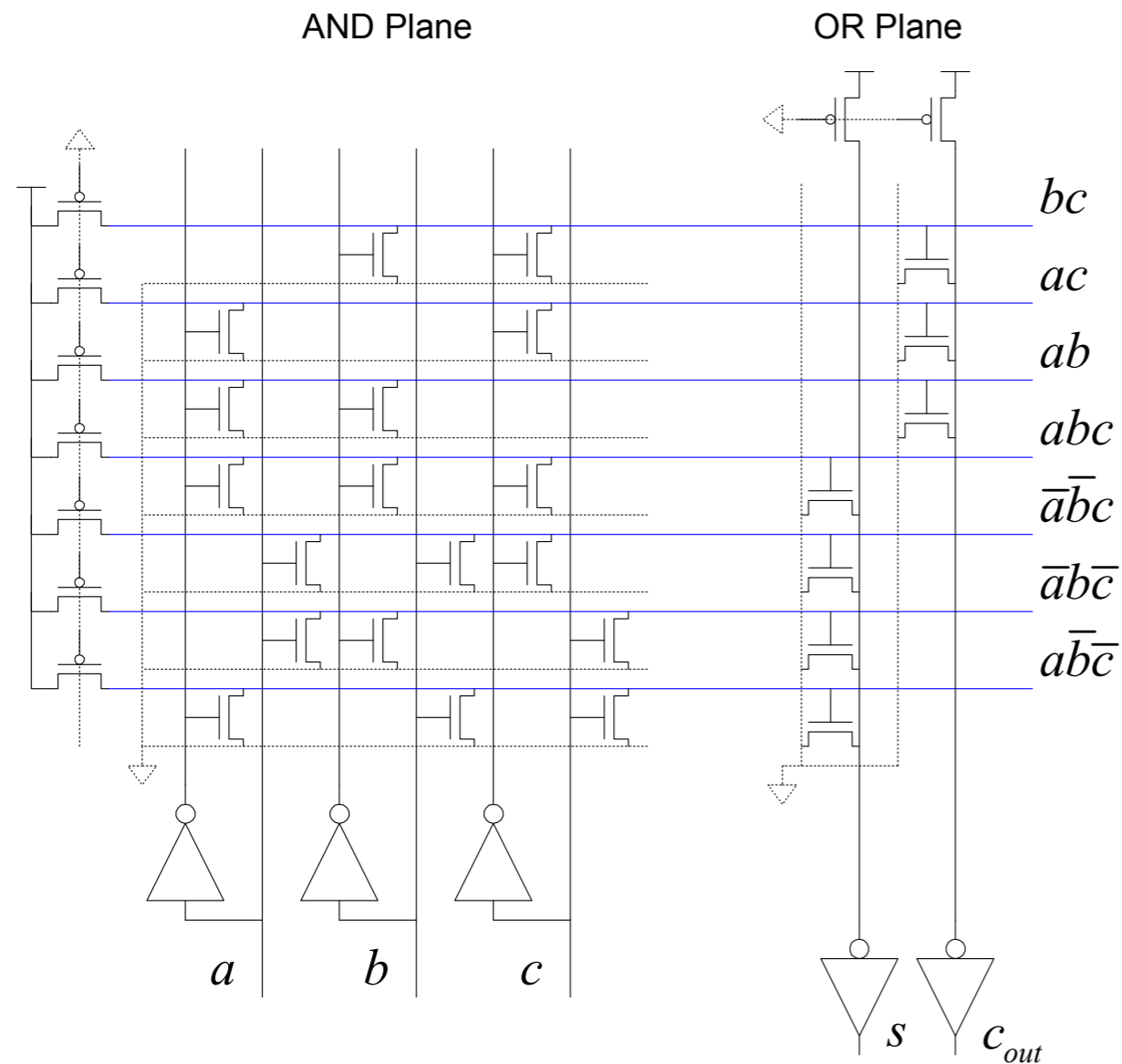


# NOR-NOR PLAs

- ANDs and ORs not very efficient in CMOS
- Dynamic or Pseudo-nMOS NORs very efficient
- Use **DeMorgan's Law** to convert to all NORs



# PLA Schematic & Layout



# PLAs vs. ROMs

- The OR plane of the PLA is like the ROM array
- The AND plane of the PLA is like the ROM decoder
- PLAs are more flexible than ROMs
  - No need to have  $2^n$  rows for  $n$  inputs
  - Only generate the minterms that are needed
  - Take advantage of logic simplification

# RoboAnt PLA

- Convert state transition table to logic
- Karnaugh map

$S_{1:0}$	L	R	$S_{1:0}'$	TR	TL	F
00	0	0	00	0	0	1
00	1	X	01	0	0	1
00	0	1	01	0	0	1
01	1		01	0	1	0
01	0	1	01	0	1	0
01	0	0	10	0	1	0
10	X	0	10	1	0	1
10	X	1	11	1	0	1
11	1	X	01	0	1	1
11	0	0	10	0	1	1
11	0	1	11	0	1	1

$S_1'$

	$S_1 S_0$			
	00	01	11	10
00	0	1	1	1
LR 01	0	0	1	1
11	0	0	0	1
10	0	0	0	1

$S_1' =$

$S_0'$

	$S_1 S_0$			
	00	01	11	10
00	0	0	0	0
LR 01	1	1	1	1
11	1	1	1	1
10	1	1	1	0

$S_0' =$

$TR =$

$TL =$

$F =$

# EX. RoboAnt Dot Diagram

$$S1' = S_1 \overline{S_0} + \overline{L}S_1 + \overline{L}\overline{R}S_0$$

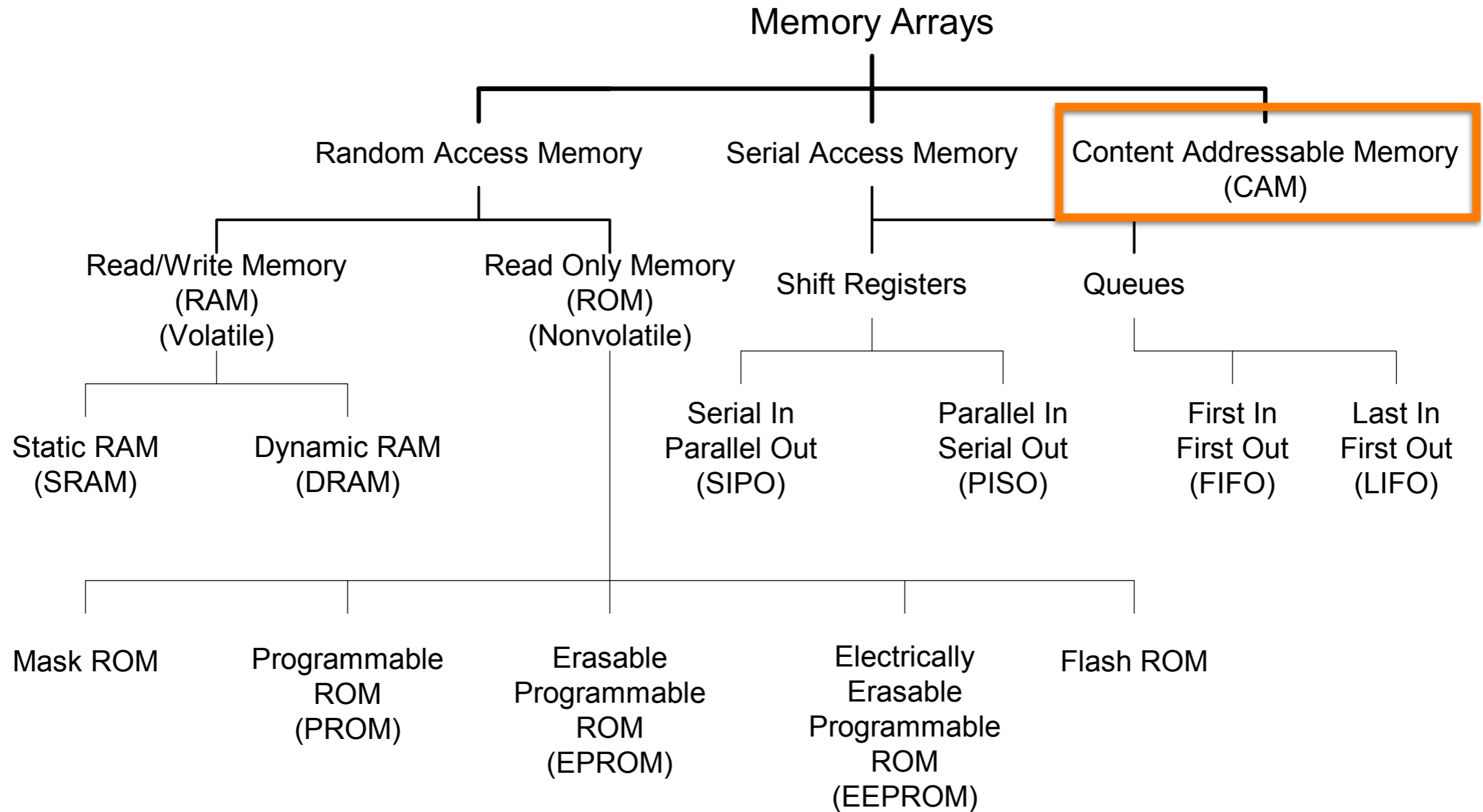
$$S0' = R + L\overline{S_1} + LS_0$$

$$TR = S_1 \overline{S_0}$$

$$TL = S_0$$

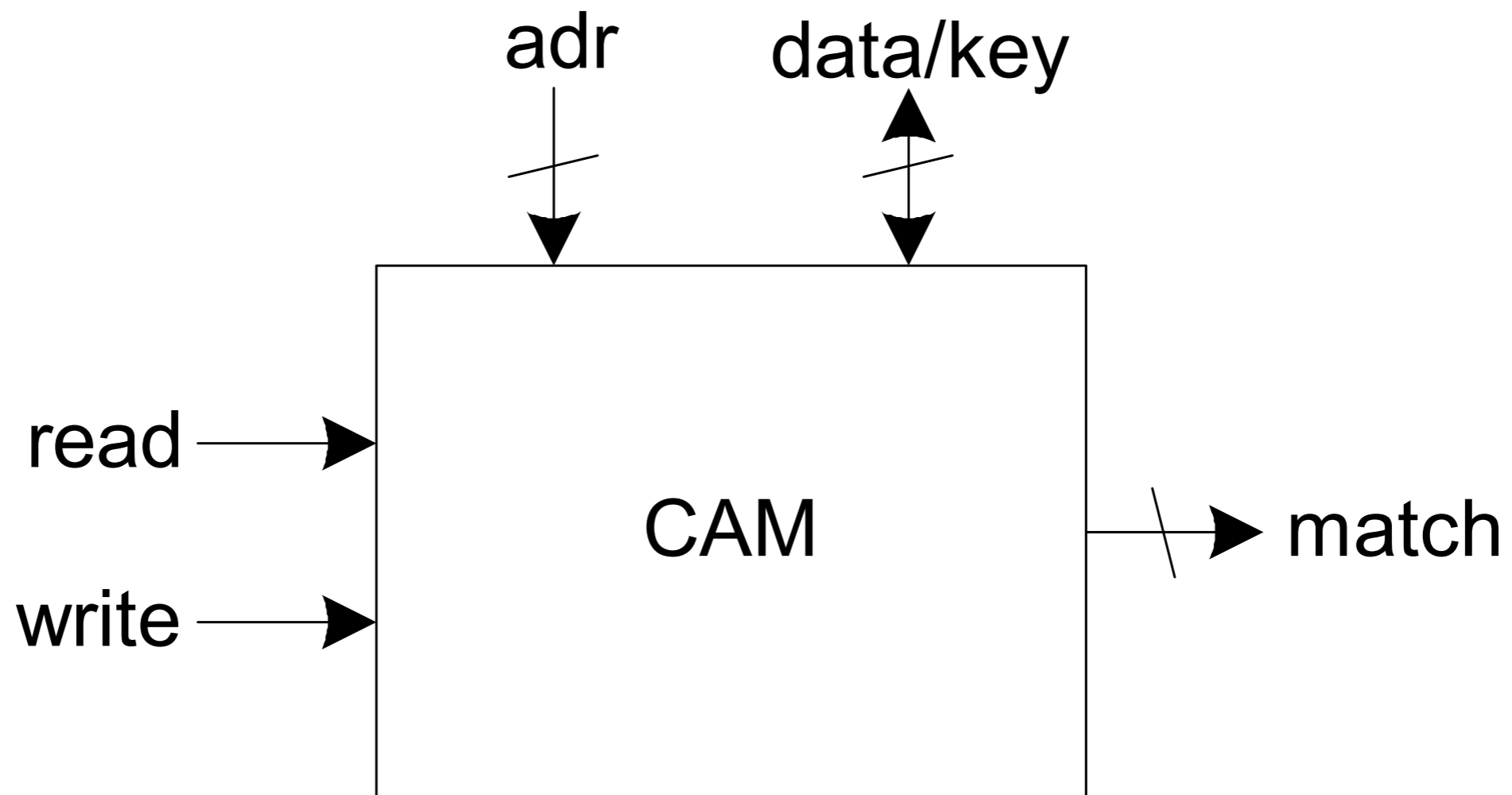
$$F = S_1 + \overline{S_0}$$

# Memory Arrays\*



# CAMs\*

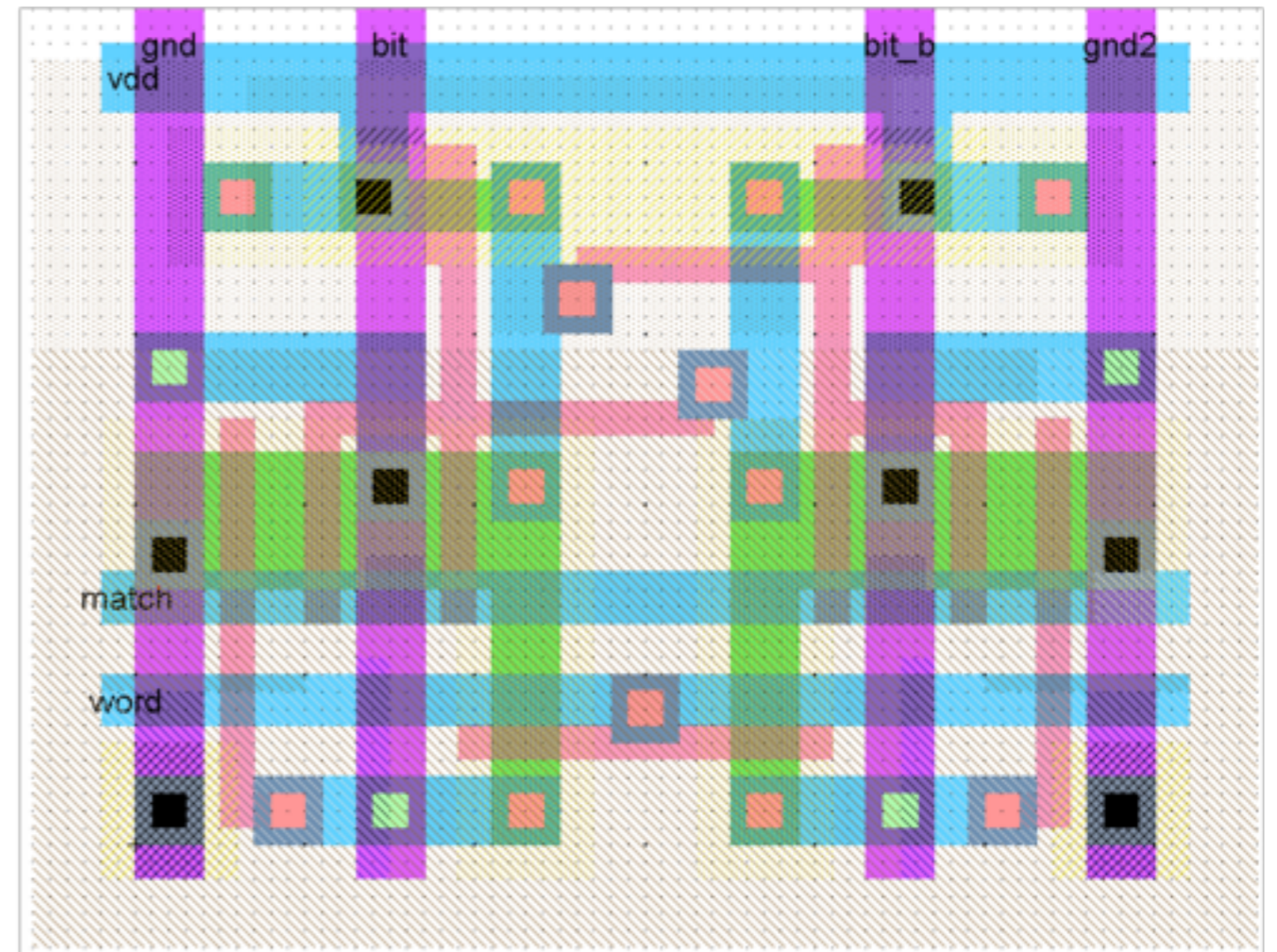
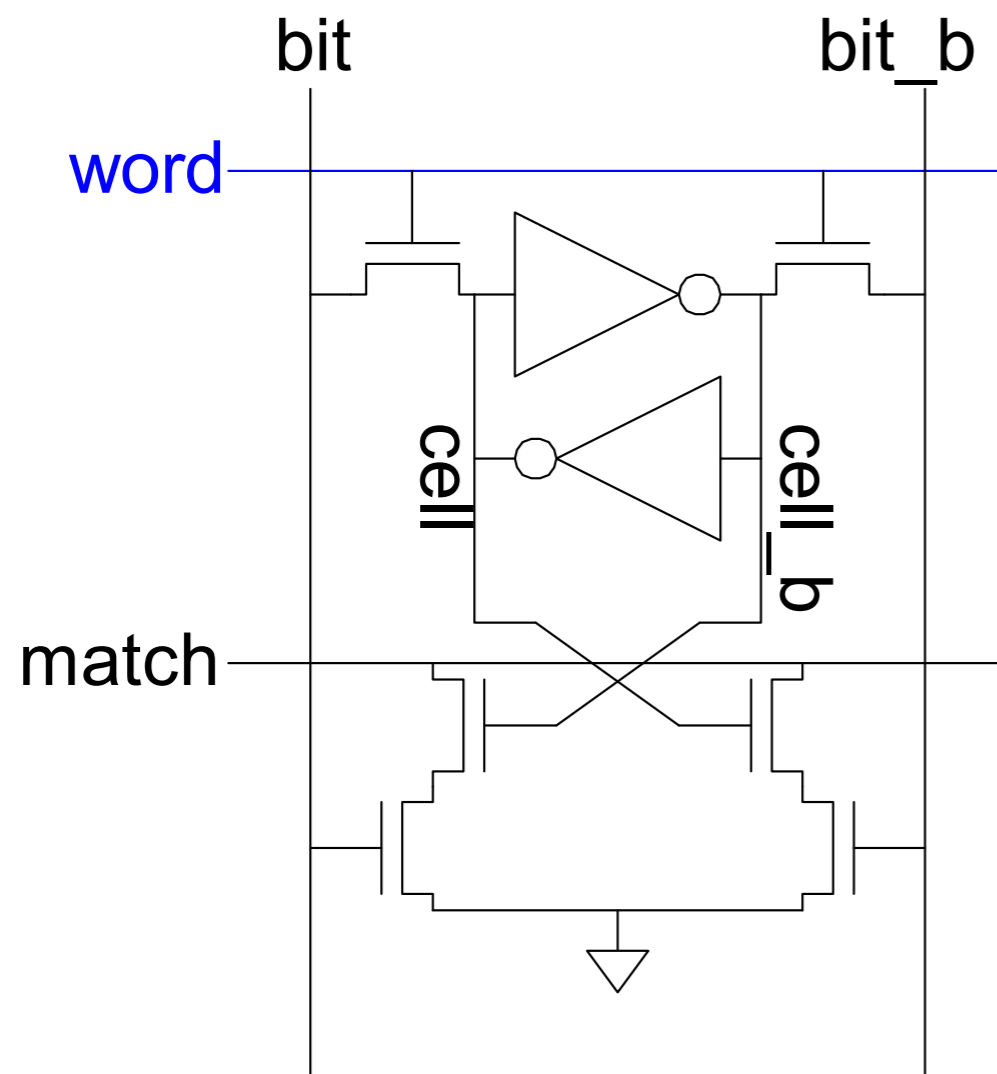
- Extension of ordinary memory (e.g. SRAM)
  - Read and write memory as usual
  - Also *match* to see which words contain a *key*





# 10T CAM Cell\*

- Add four match transistors to 6T SRAM
  - 56 x 43  $\lambda$  unit cell



# CAM Cell Operation\*

- Read and write like ordinary SRAM
- For matching:
  - Leave wordline low
  - Precharge matchlines
  - Place key on bitlines
  - Matchlines evaluate
- Miss line
  - Pseudo-nMOS NOR of match lines
  - Goes high if no words match

