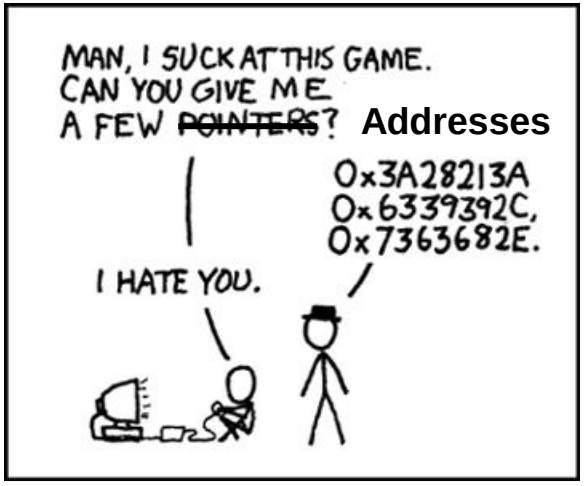
CS/ECE 752: Advanced Computer Architecture I



Professor Matthew D. Sinclair

Memories

Slide History/Attribution Diagram:

Announcements 10/7/20

- Midterm next Wednesday
- Midterm review on Monday
  - Bring questions
- No class Friday
- Project Preliminary Ideas due Friday

Midterm

- Will be similar in format to past exams
  - Samples available on course website
  - Because electronic: a bit more focus on advanced problems
  - Overall: combination of math and essay problems
- Logistics
  - 24 hours to complete (due 215 PM on 10/14)
  - Post questions on Piazza
  - I will be available during lecture time (BBC Ultra)
  - Will run all solutions through software to detect academic misconduct
  - Can be hand-written or typed
- Likely will use Gradescope to grade

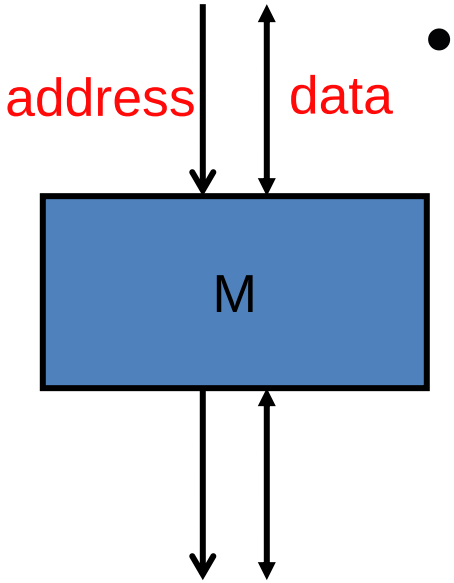
SRAM Technology

- **SRAM**: static RAM
  - **Static**: bits directly connected to power/ground
    - Naturally/continuously "refreshed", never decay (contrast DRAM)
  - Designed for speed

  - Implements all storage arrays in real processors
    - Register file, caches, branch predictor, etc.
    - Everything except pipeline latches

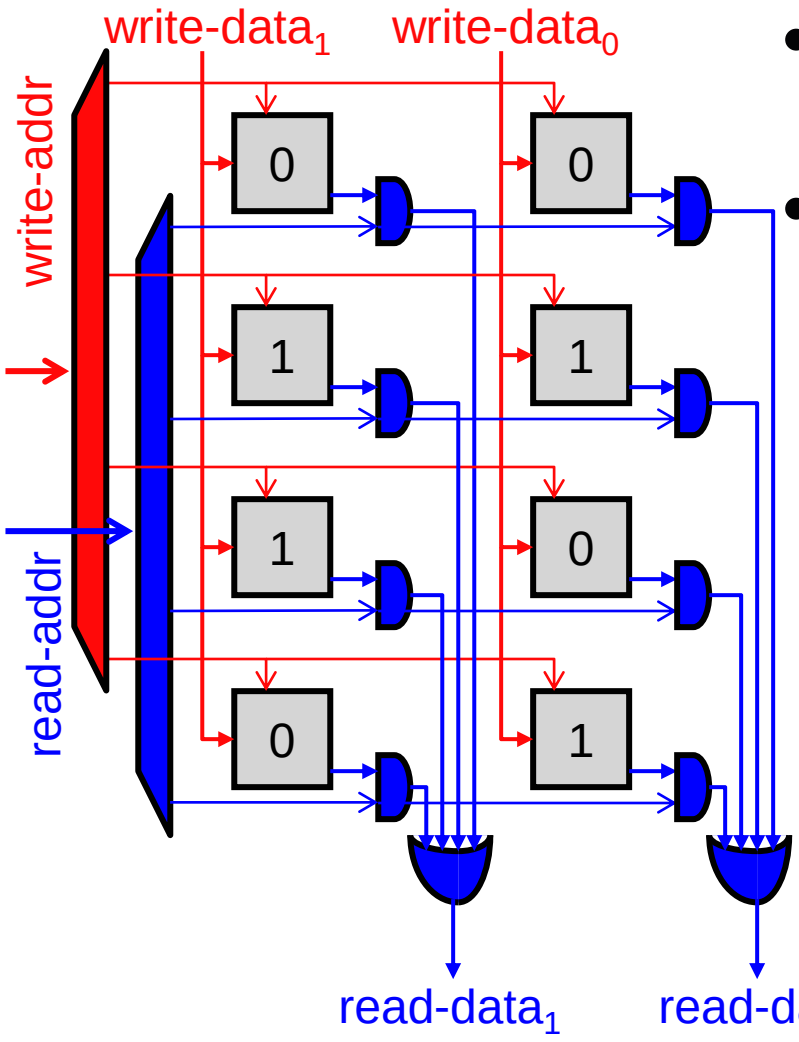
- Latches vs. SRAM
  - Latches: singleton word, always read/write same one
  - SRAM: array of words, can read/write different ones
    - Address indicates which one

(CMOS) Memory Components



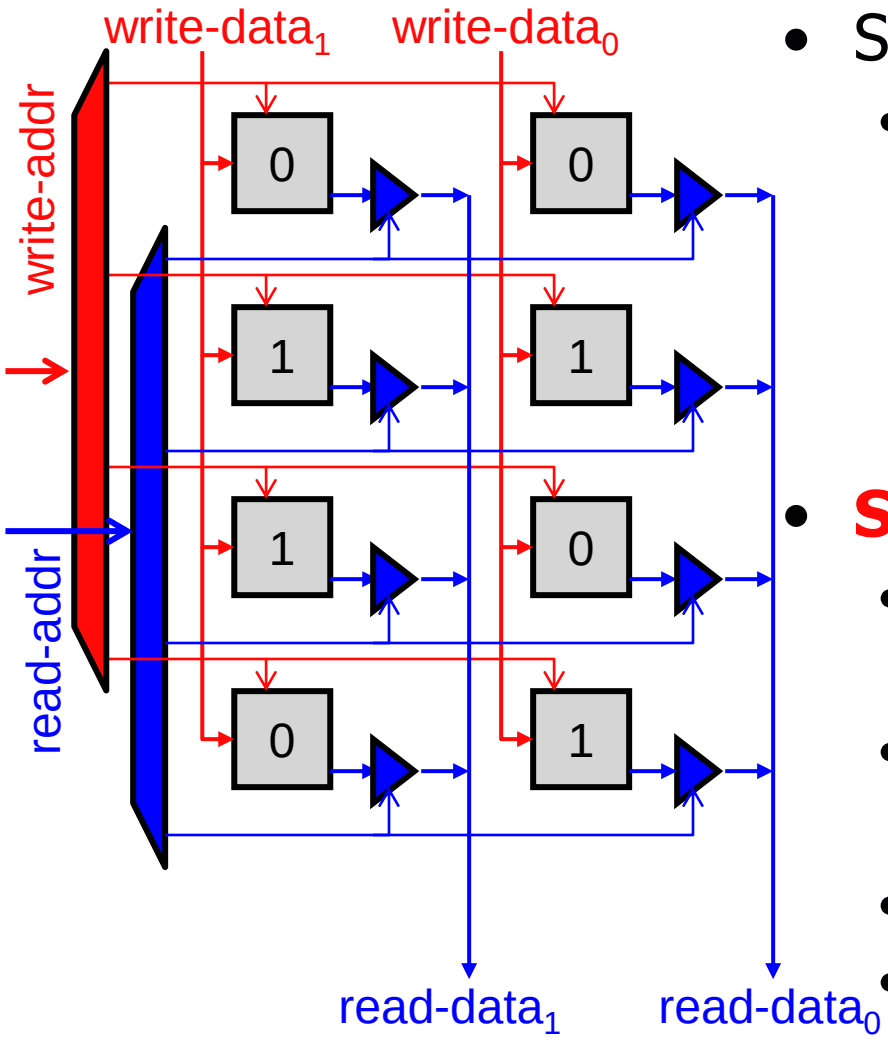
- Interface
  - N-bit **address** bus (on N-bit machine)
  - **Data** bus
    - Can have read/write on same data bus
    - Or, can have dedicated read/write buses
  - Can have multiple **ports**: address/data bus pairs

SRAM: First Cut



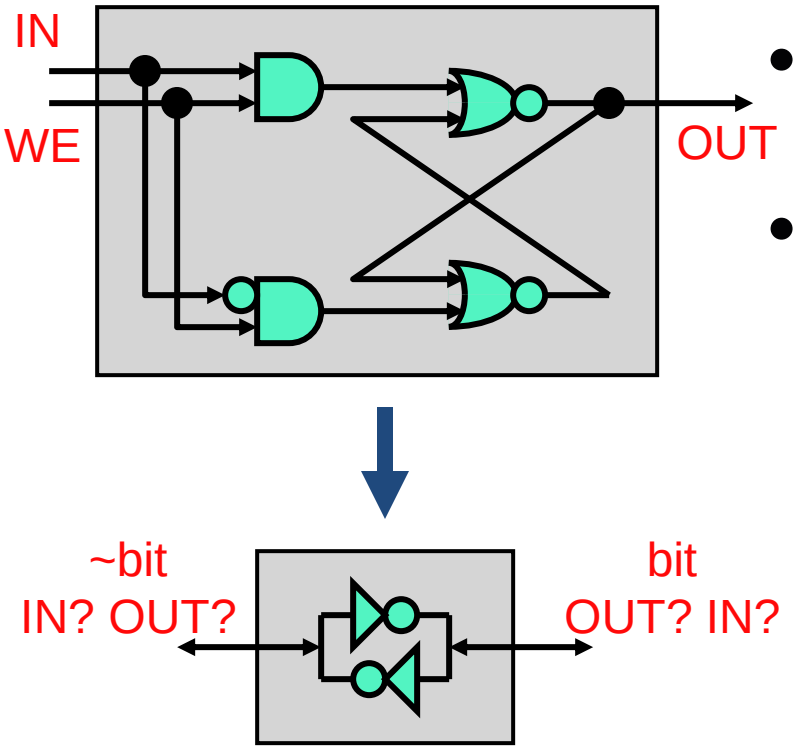
- 4x2 (4 2-bit words) RAM
  - 2-bit addr
- First cut: bits are D-Latches
  - **Write port**
    - Addr **decodes** to enable signals
  - **Read port**
    - Addr **decodes** to mux selectors
    - 1024 input OR gate?
    - Physical layout of output wires
      - RAM width $\propto$ M
      - Wire delay $\propto$ wire length

SRAM: Second Cut



- Second cut: tri-state wired-OR
  - Read mux using **tri-states**
    - + Scalable, distributed "muxes"
    - + Better layout of output wires
      - RAM width independent of M
- **Standard RAM**
  - Bits in word connected by **wordline**
    - 1-hot decode address
  - Bits in position connected by **bitline**
    - Shared input/output wires
  - **Port**: one set of wordlines/bitlines
  - Grid-like design

SRAM: Third Cut

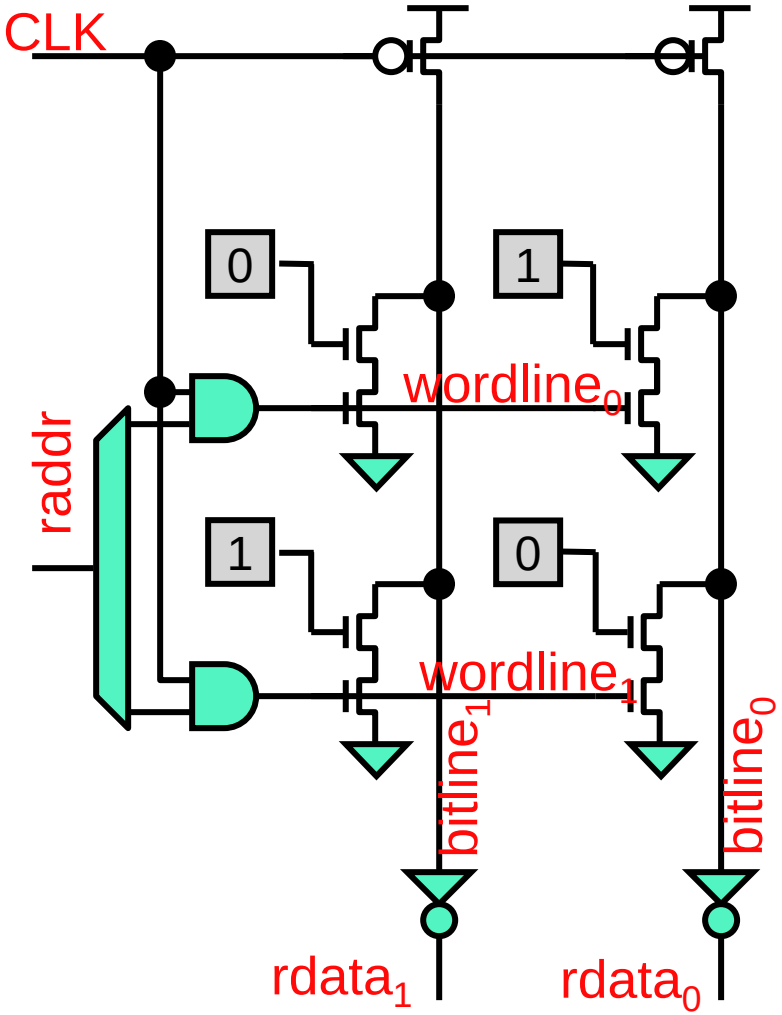


- Third cut: replace latches with…
  - 28 transistors per bit
- **Cross-coupled inverters (CCI)**
  - \+ 4 transistors
  - Convention
    - Right node is **bit**, left is **~bit**
  - Non-digital interface
    - What is the input and output?
    - Where is write enable?
  - Implement ports in "analog" way
    - Transistors, not full gates

SRAM: Register Files and Caches

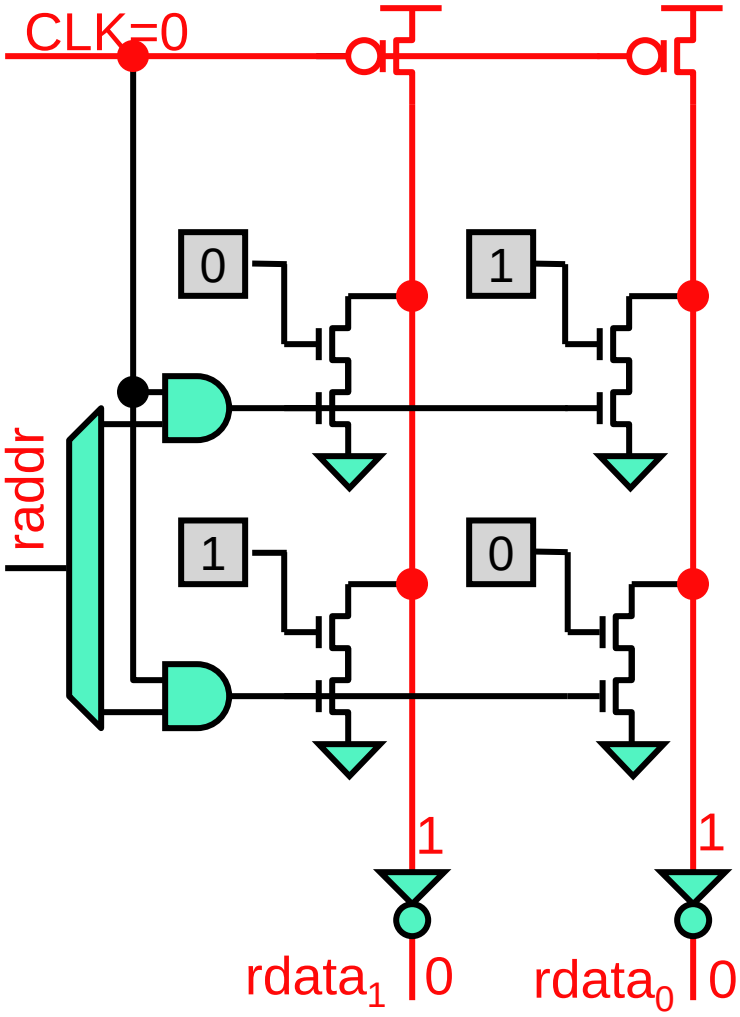
- Two different SRAM port styles
  - **Regfile style**
    - Modest size: <4KB
    - Many ports: some read-only, some write-only
    - Write and read both take half a cycle (write first, read second)
  - **Cache style**
    - Larger size: >8KB
    - Few ports: read/write in a single port
    - Write and read can both take full cycle

Regfile-Style Read Port



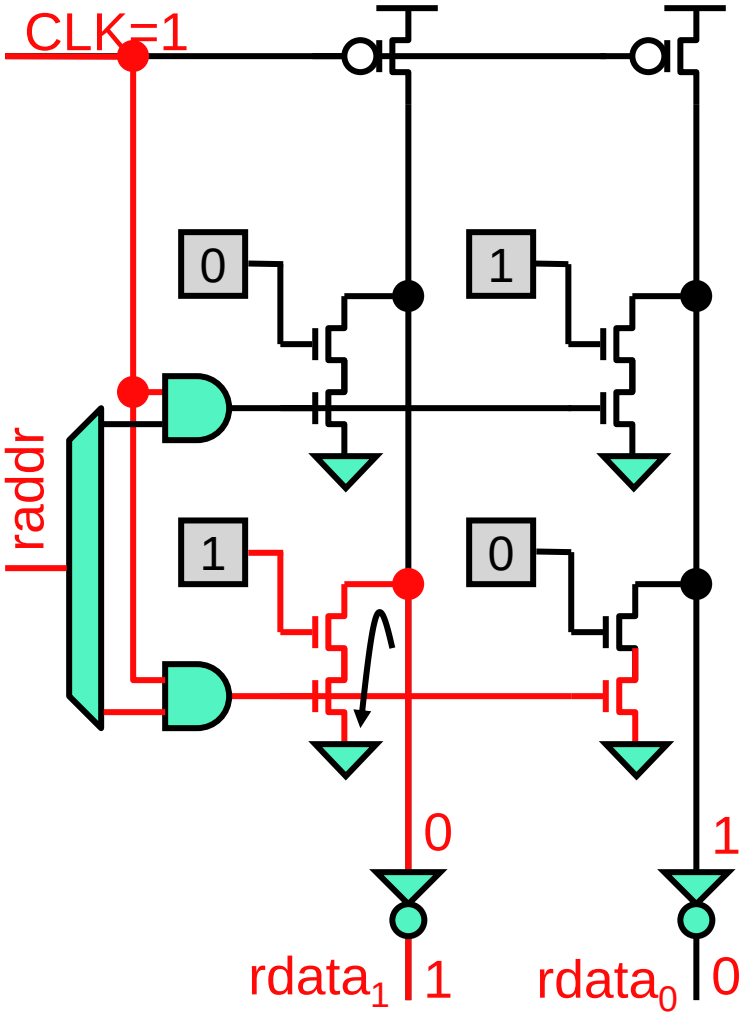
- Two phase read
  - Phase I: clk = 0
    - Pre-charge bitlines to 1
    - Negated bitlines are 0
  - Phase II: clk = 1
    - One wordline goes high
    - All "1" bits in that row discharge their bitlines to 0
    - Negated bitlines go to 1

Read Port In Action: Phase I



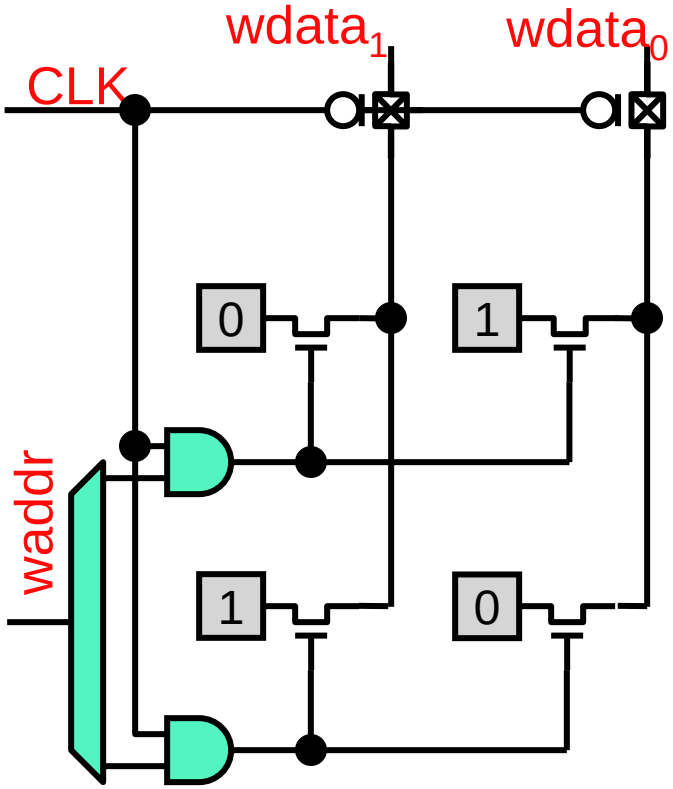
- CLK = 0
  - p-transistors conduct
  - Bitlines "pre-charge" to 1
  - $rdata_{1-0}$ are 0

Read Port In Action: Phase II

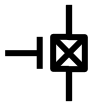


- raddr = 1
- CLK = 1
  - p-transistors close
  - $wordline_1 = 1$
  - "1" bits on $wordline_1$ create path from bitline to ground
    - SRAM[1]
  - Corresponding bitlines discharge
    - $bitline_1$
  - Corresponding rdata bits go to 1
    - $rdata_1$
- That's a read

Regfile-Style Write Port

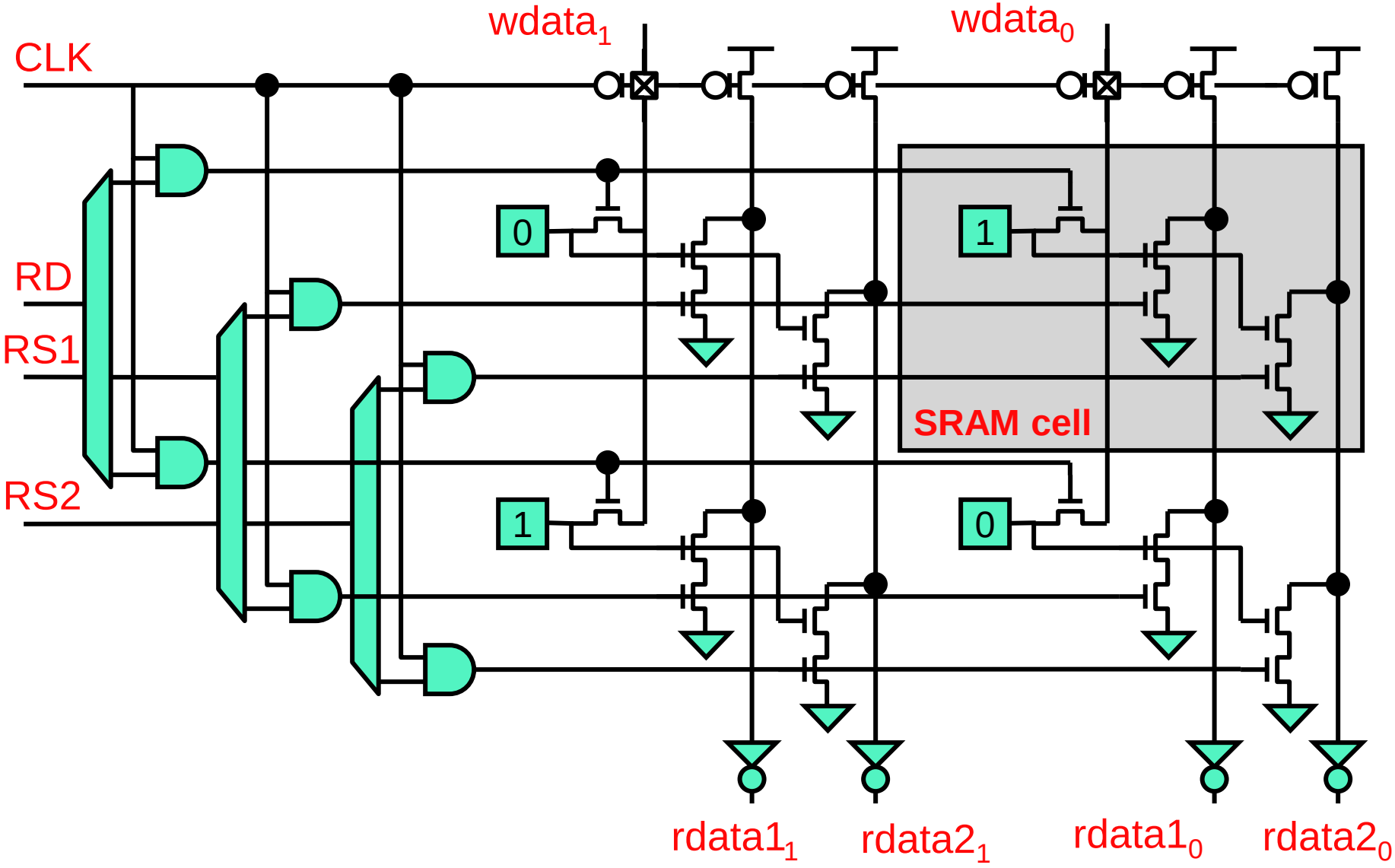


- Two phase write
  - Phase I: clk = 1
    - Stabilize one wordline high
  - Phase II: clk = 0
    - Open pass-transistors
    - "Overwhelm" bits in selected word
  - Actually: two clocks here
    - Both phases in first half

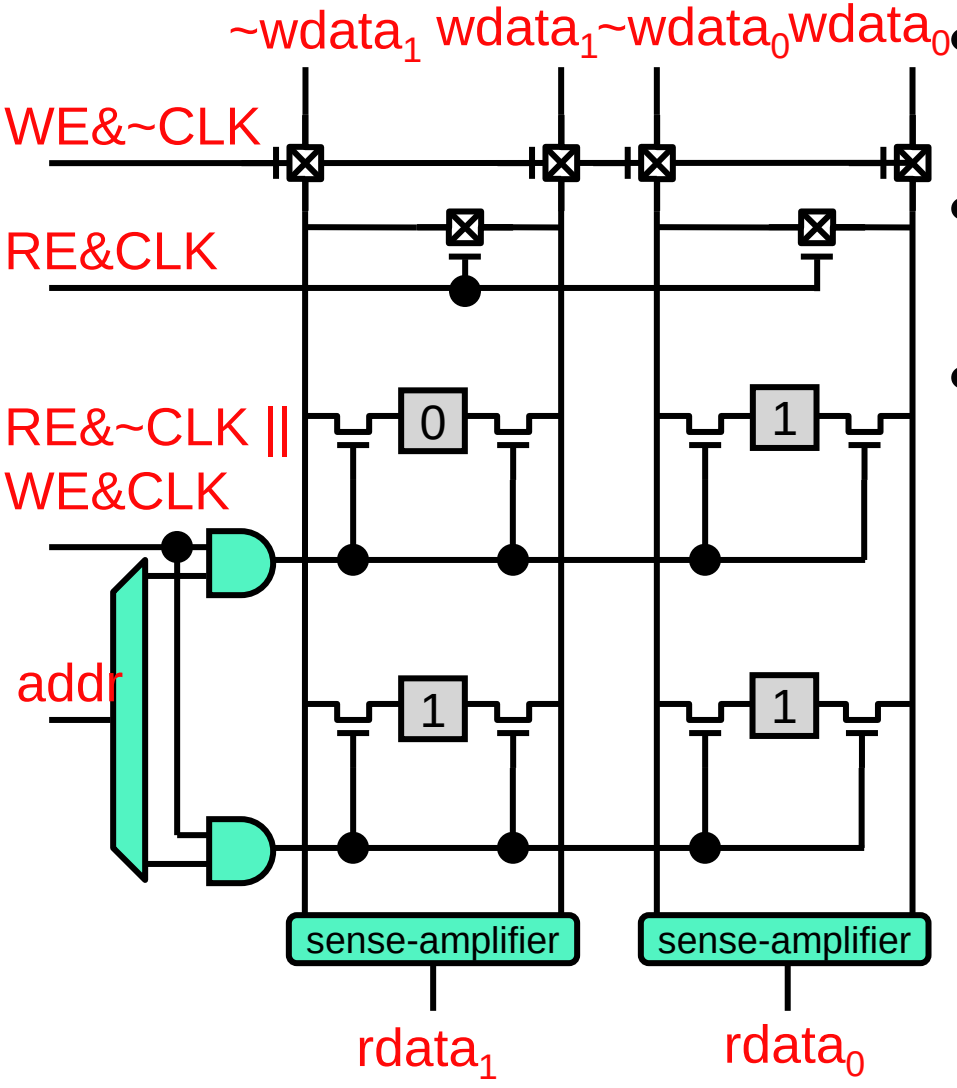


pass transistor: like a tri-state buffer

A 2-Read Port 1-Write Port Regfile

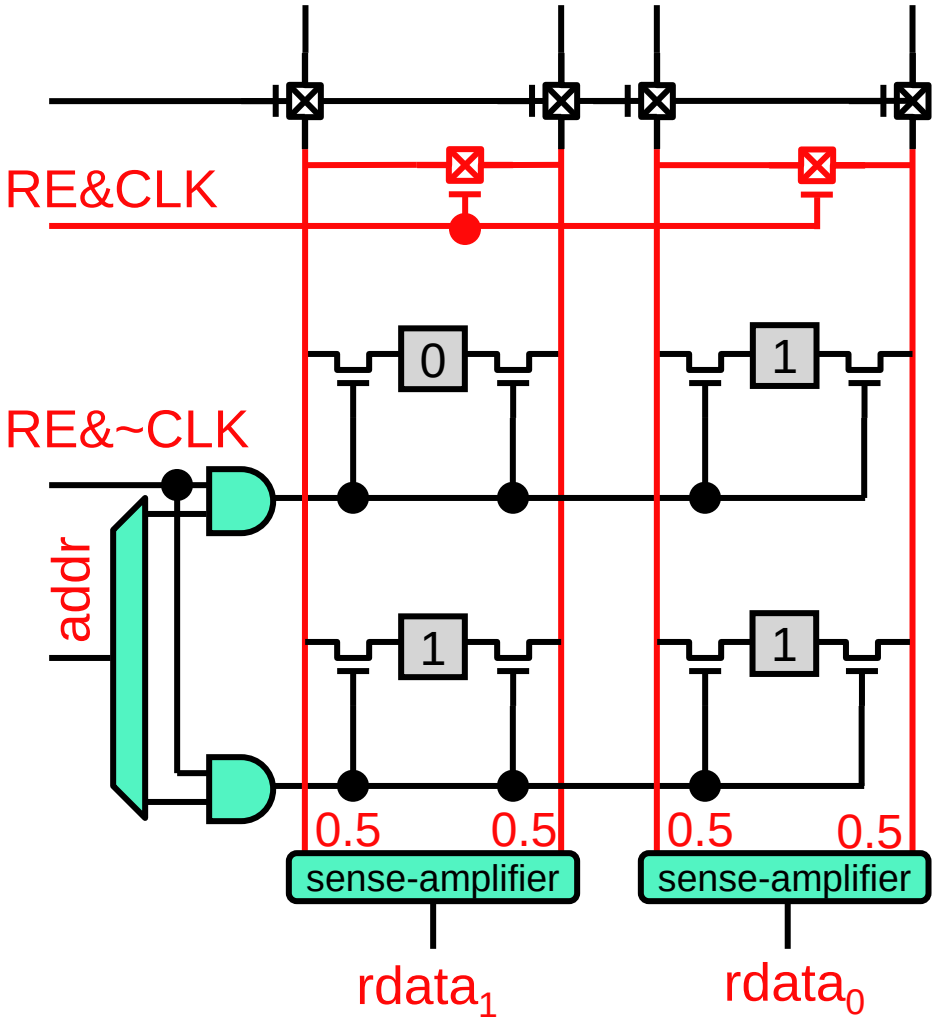


Cache-Style Read/Write Port

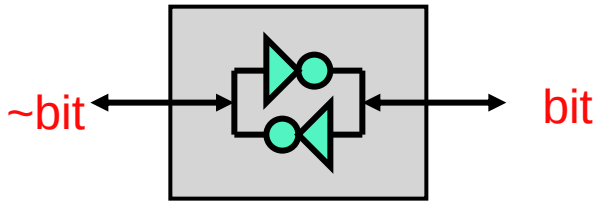


- **Double-ended bitlines**
  - Connect to both sides of bit
- Two-phase write
  - Just like a register file
- Two phase read
  - Phase I: clk = 1
    - Equalize bitline pair voltage
  - Phase II: clk = 0
    - One wordline high
    - "1 side" bitline swings up
    - "0 side" bitline swings down
    - Sens-amp translates swing

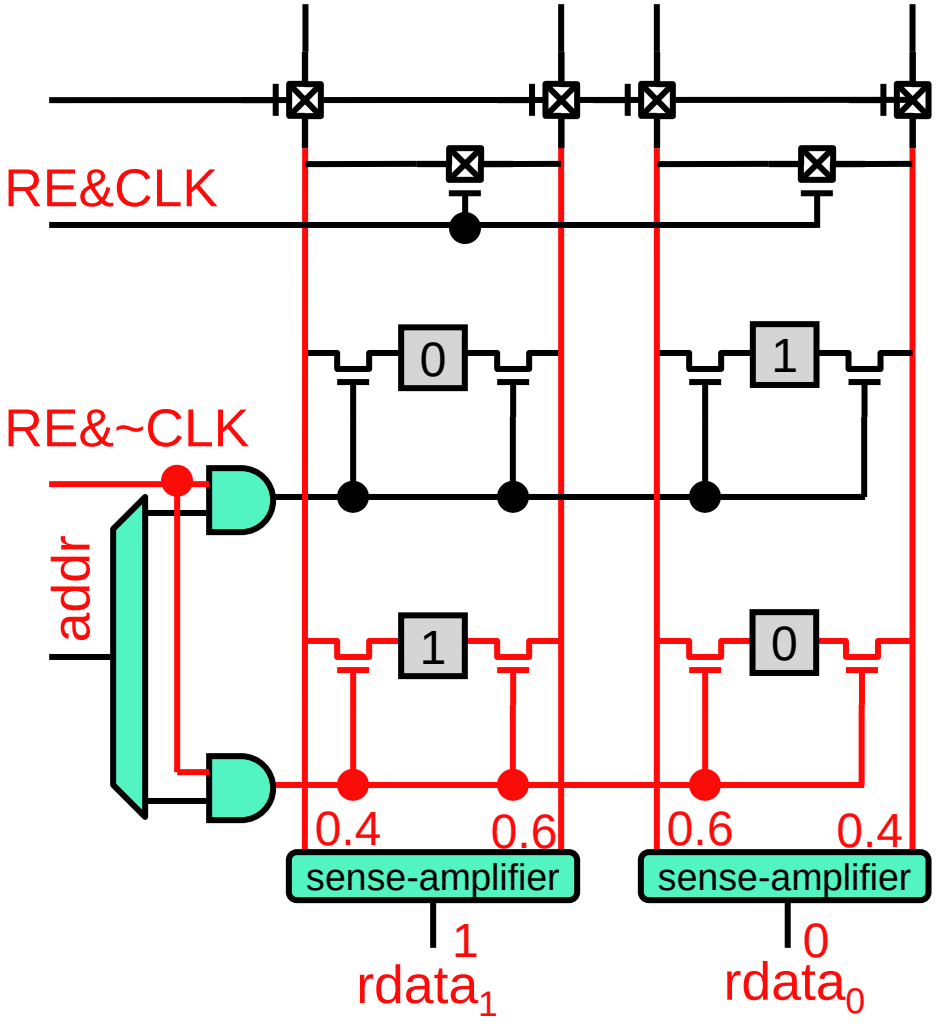
Read/Write Port in Read Action: Phase I



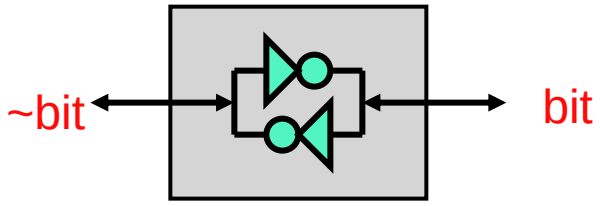
- Phase I: clk = 1
  - Equalize voltage on bitline pairs
  - To (nominally) 0.5



Read/Write Port in Read Action: Phase II

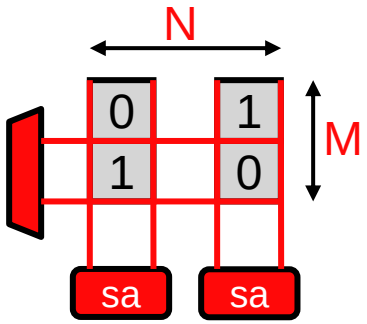


- Phase II: clk = 0
  - $wordline_1$ goes high
  - "1 side" bitlines swing high 0.6
  - "0 side" bitlines swing low 0.4
  - Sens-amps interpret swing



Cache-Style SRAM Latency

- Assume
  - M N-bit words
  - Some minimum wire spacing L
  - CCIs occupy no space

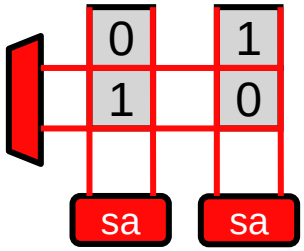


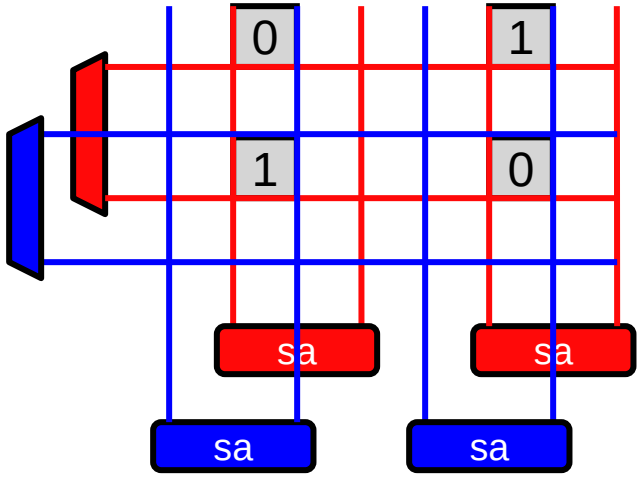
- 4 major latency components: taken in series
  - **Decoder**: $\propto \log_2 M$
  - **Wordlines**: $\propto$ 2NL (cross 2N bitlines)
  - Bitlines: $\propto$ ML (cross M wordlines)
  - **Muxes** + **sense-amps**: constant
  - 32KB SRAM: red components contribute about equally

  - Latency: $\propto$ (2N+M)L
    - Maximize storage for some max latency: make SRAMs as square as possible: minimize 2N+M
  - Latency: $\propto \sqrt{}$#total bits

Multi-Ported Cache-Style SRAM Latency

- Previous calculation had hidden constant
  - Number of ports **P**
- Recalculate latency components
  - Decoder: $\propto \log_2 M$ (unchanged)
  - Wordlines: $\propto 2NL\mathbf{P}$ (cross $2N\mathbf{P}$ bitlines)
  - Bitlines: $\propto ML\mathbf{P}$ (cross $M\mathbf{P}$ wordlines)
  - Muxes + sense-amps: constant (unchanged)
  - Latency: $\propto (2N+M)L\mathbf{P}$
  - **Latency: $\propto \sqrt{\#bits * \#ports}$**
- How does latency scale? (P)
- How does power scale? (P^2)
  - Both length and number active increase

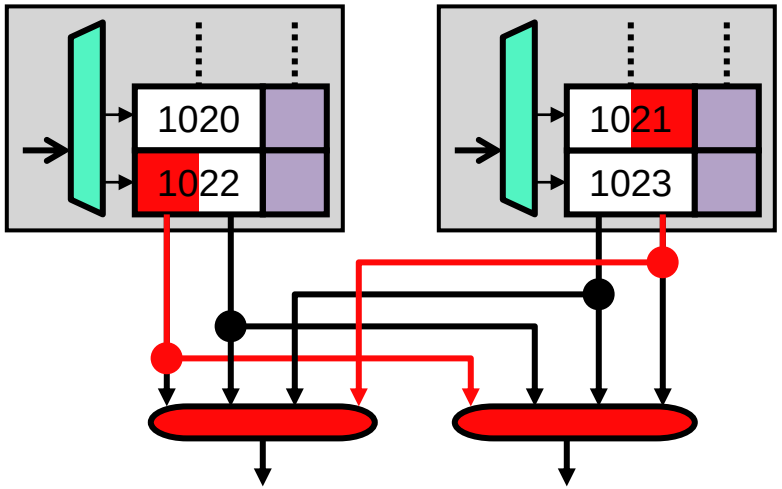




Multi-Porting an SRAM

- Why multi-porting?
  - Multiple accesses per cycle
- **True multi-porting** (physically adding a port) not good
  - \+ Any combination of accesses will work
  - – Increases access latency, energy $\propto$ P, area $\propto P^2$
- Another option: **pipelining**
  - Timeshare single port on clock edges (wave pipelining: no latches)
  - \+ Negligible area, latency, energy increase
  - – Not scalable beyond 2 ports
- Yet another option: **replication**
  - Don't laugh: used for register files, even caches (Alpha 21164)
  - Smaller and faster than true multi-porting $2*P^2 < (2*P)^2$
  - \+ Adds read bandwidth, any combination of reads will work
  - – Doesn't add write bandwidth, not really scalable beyond 2 ports

Banking an SRAM

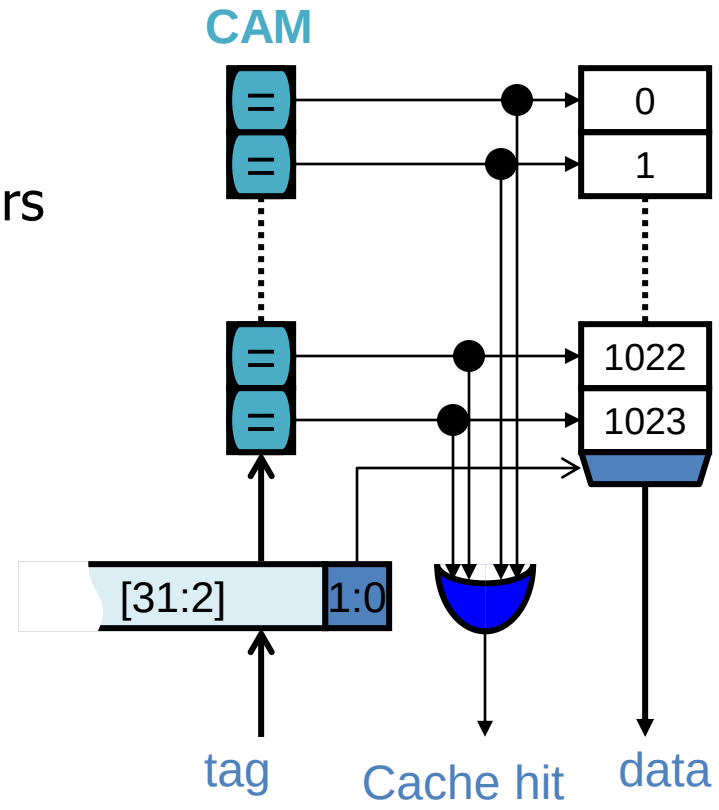


- Divide SRAM into banks, interleave the addresses
- Allow parallel access to different banks
- Two accesses to same bank? **bank-conflict**, one waits
- Low area, latency overhead for routing requests to banks
- Few bank conflicts given sufficient number of banks
  - Rule of thumb: N simultaneous accesses $\rightarrow$ 2N banks

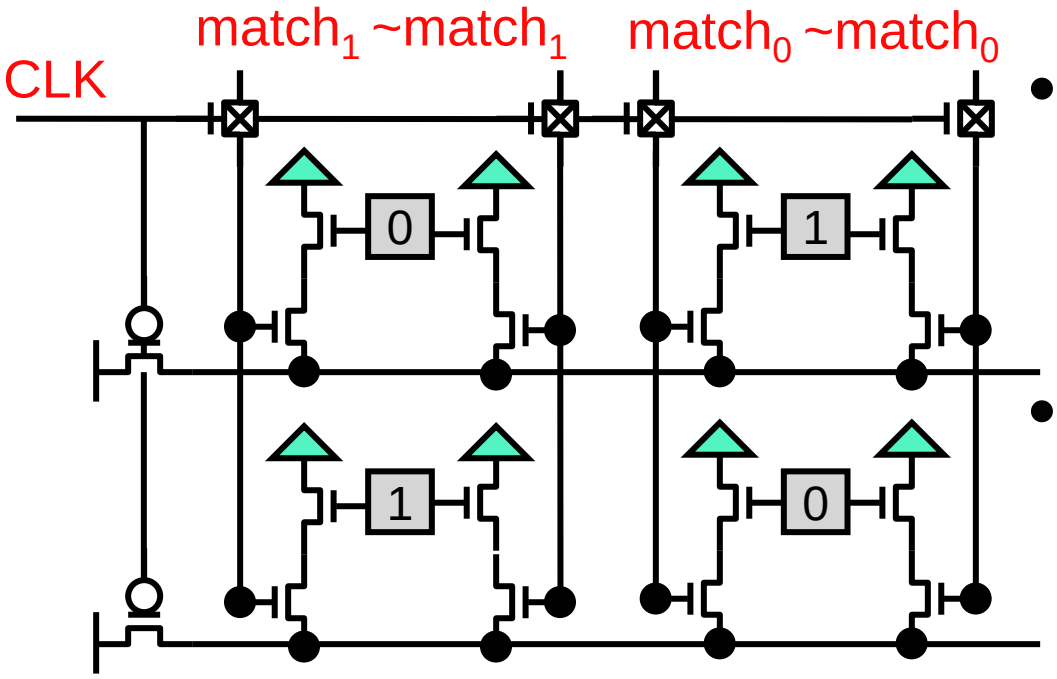
- How to divide words among banks?
  - **Round robin**: using address LSB (least significant bits)
  - Example: 16 word RAM divided into 4 banks
  - **b0**: 0,4,8,12; **b1**: 1,5,9,13; **b2**: 2,6,10,14; **b3**: 3,7,11,15
  - Why? Spatial locality

Full-Associativity with CAMs

- **CAM**: content associative memory
  - Array of words with built-in comparators
  - Matchlines instead of bitlines
  - Output is "one-hot" encoding of match
- FA cache?
  - Tags as CAM
  - Data as RAM
- **Hardware is not software**
  - No such thing as software CAM

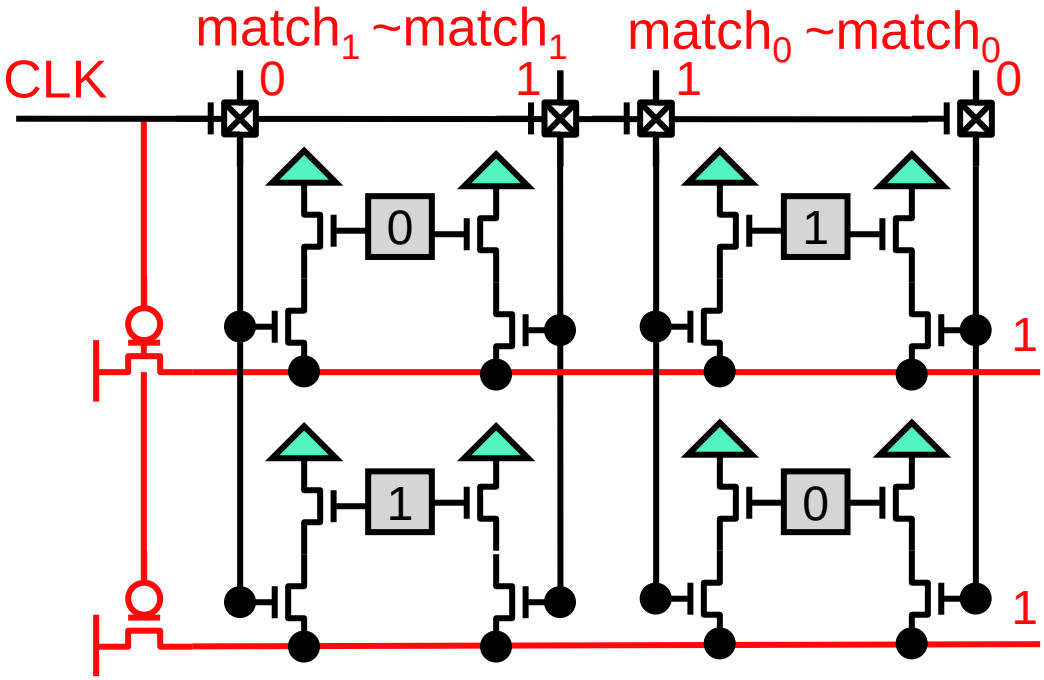


CAM Circuit



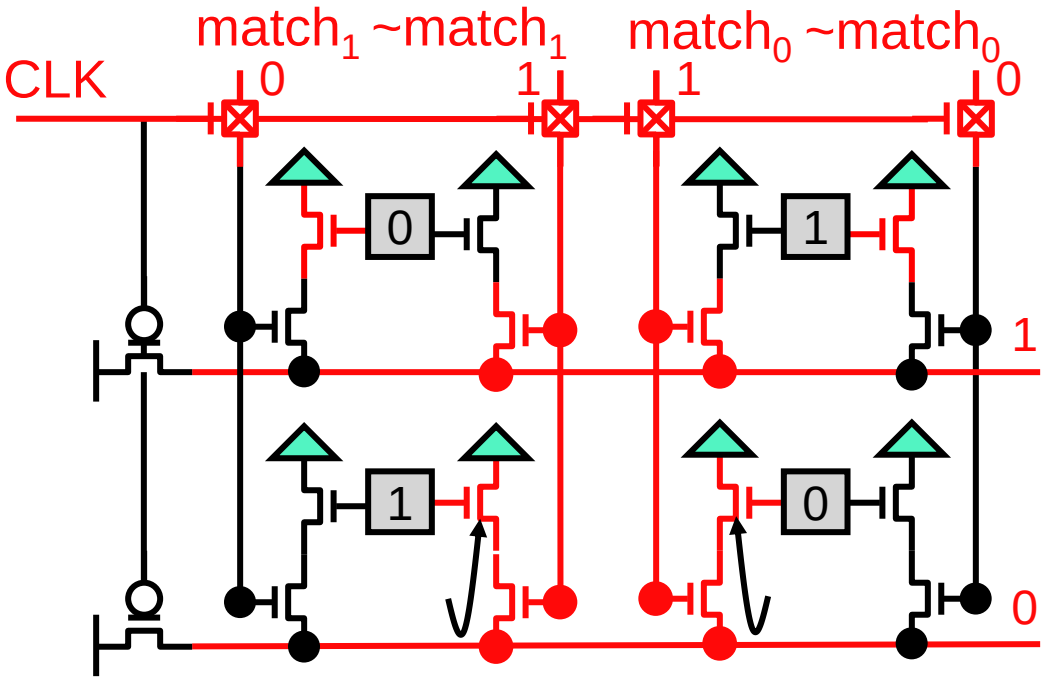
- CAM: reverse RAM
  - Bitlines are inputs
    - Called **matchlines**
  - Wordlines are outputs
- Two phase match
  - Phase I: clk=0
    - Pre-charge wordlines
  - Phase II: clk=1
    - Enable matchlines
    - Non-matching bits dis-charge wordlines

CAM Circuit In Action: Phase I

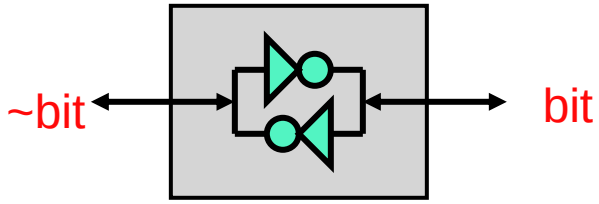


- Phase I: clk=0
  - Pre-charge wordlines

CAM Circuit In Action: Phase II



- Phase II: clk=1
  - Enable matchlines
    - Note: bits flipped
  - Non-matching bit discharges wordline
    - ANDs matches
    - NORs non-matches
  - Similar technique for doing a fast OR for hit detection



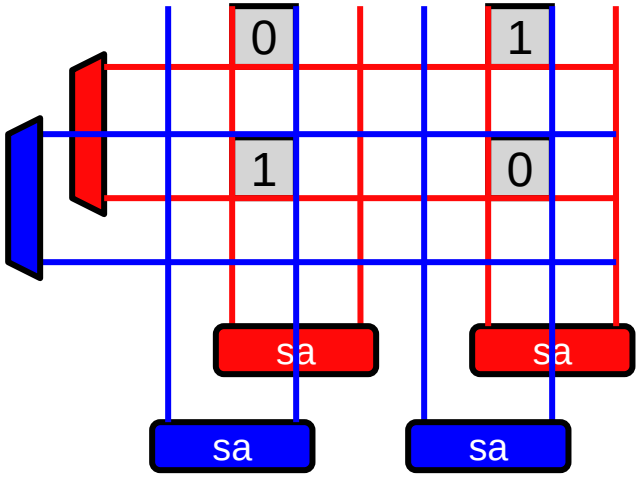
CAM Upshot

- CAMs: effective but expensive
  - Matchlines are very expensive (for nasty EE reasons)
  - Used but only for 16 or 32 way (max) associativity
  - Not for 1024-way associativity

Bonus

Multi-Ported Cache-Style SRAM Power

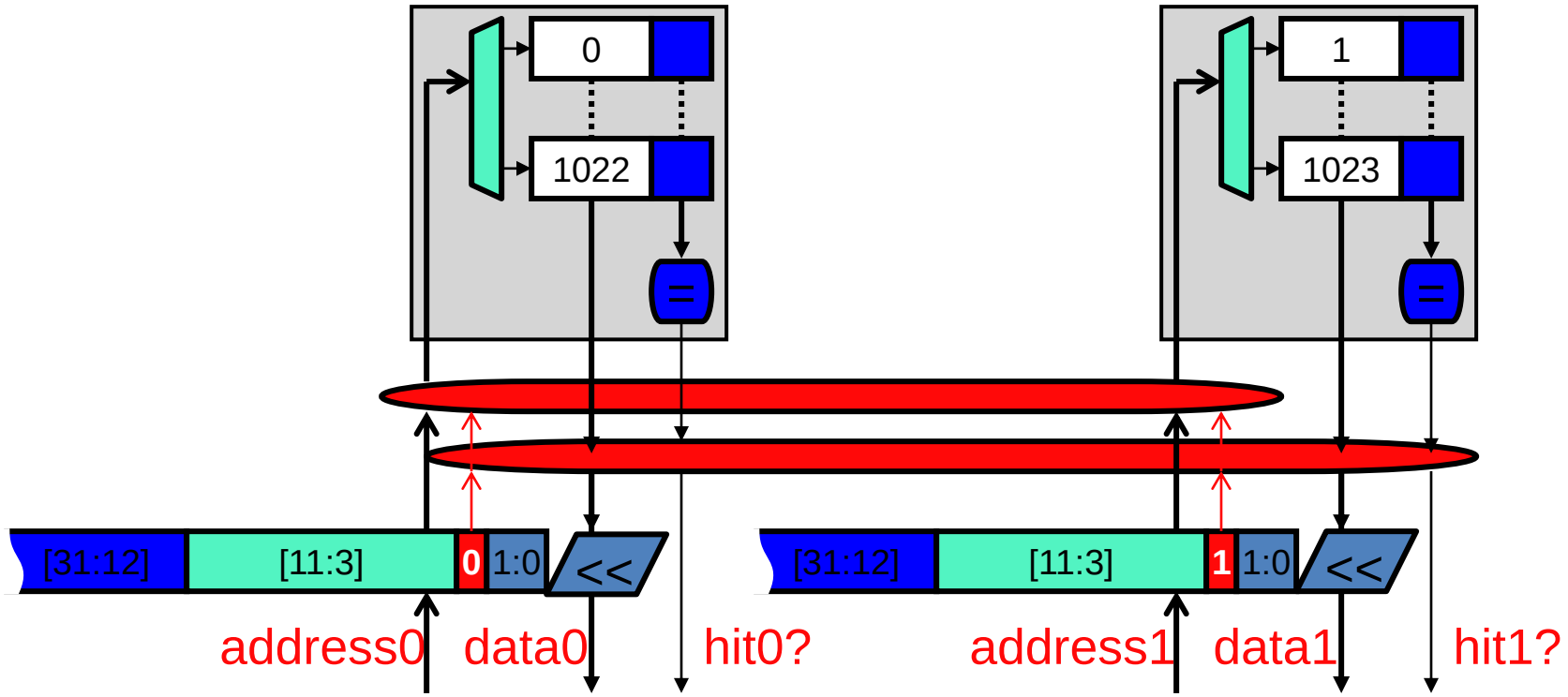
- Same four components for power
  - Decoder: $\propto \log_2 M$
  - Wordlines: $\propto$ 2NLP
    - – Huge Capacitance(C) per wordline (drives 2N gates)
    - + But only one ever high at any time (overall consumption low)
  - Bitlines: $\propto$ MLP
    - – C lower than wordlines, but large
    - + $V_{swing} << V_{DD}$ ($C * V_{swing}^2 * f$)
  - Muxes + **sense-amps**: constant
  - 32KB SRAM: sense-amps are 60–70%



- How does power scale?

A Banked Cache

- **Banking** a cache
  - Simple: bank SRAMs
  - Which address bits determine bank? LSB of index
  - **Bank network** assigns accesses to banks, resolves conflicts
    - Adds some latency too

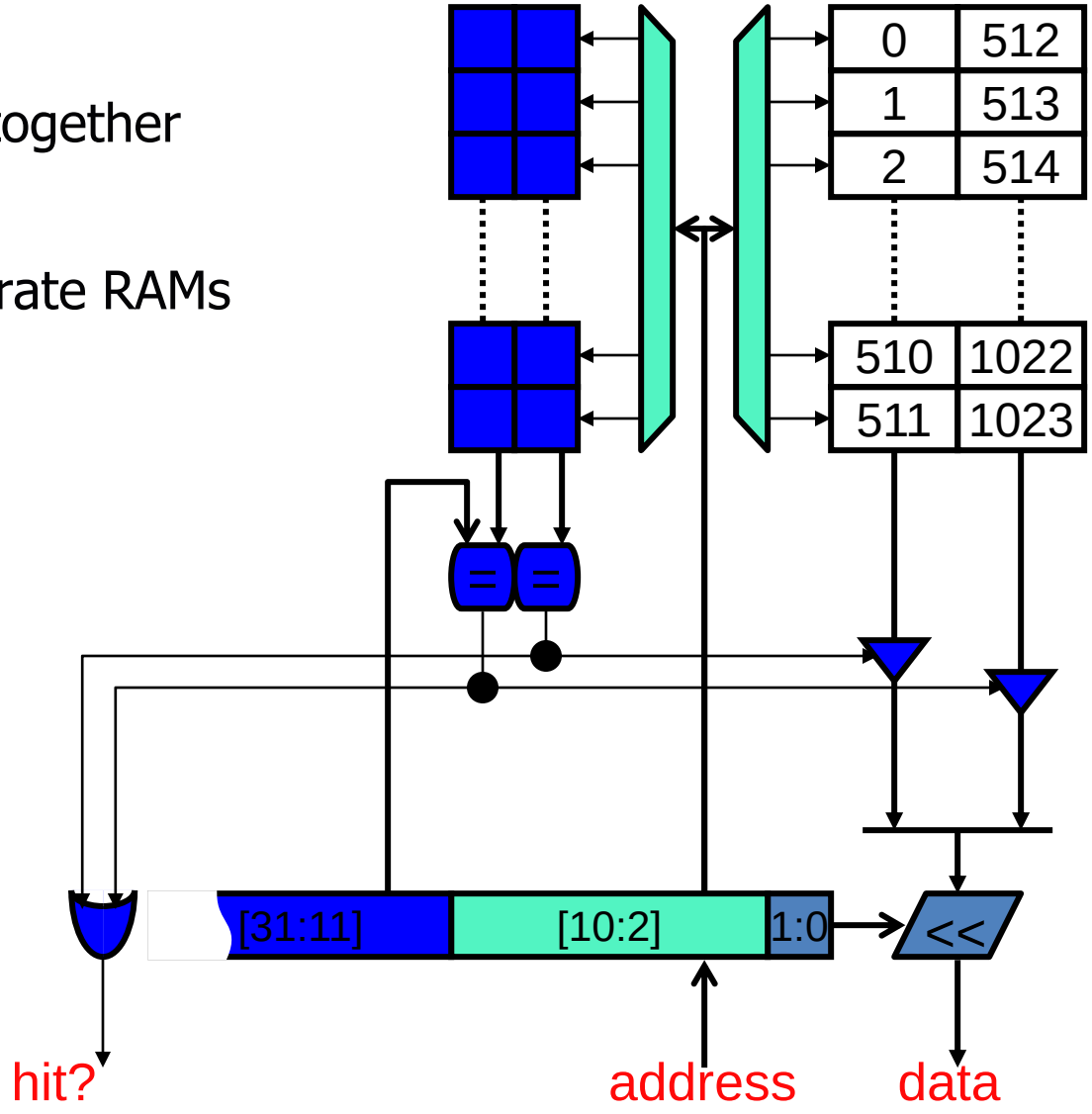


SRAM Summary

- Large storage arrays are not implemented "digitally"
- SRAM implementation exploits analog transistor properties
  - Inverter pair bits much smaller than latch/flip-flop bits
  - Wordline/bitline arrangement gives simple "grid-like" routing
  - Basic understanding of read, write, read/write ports
    - Wordlines select words
    - Overwhelm inverter-pair to write
    - Drain pre-charged line or swing voltage to read
  - Latency proportional to √#bits * #ports

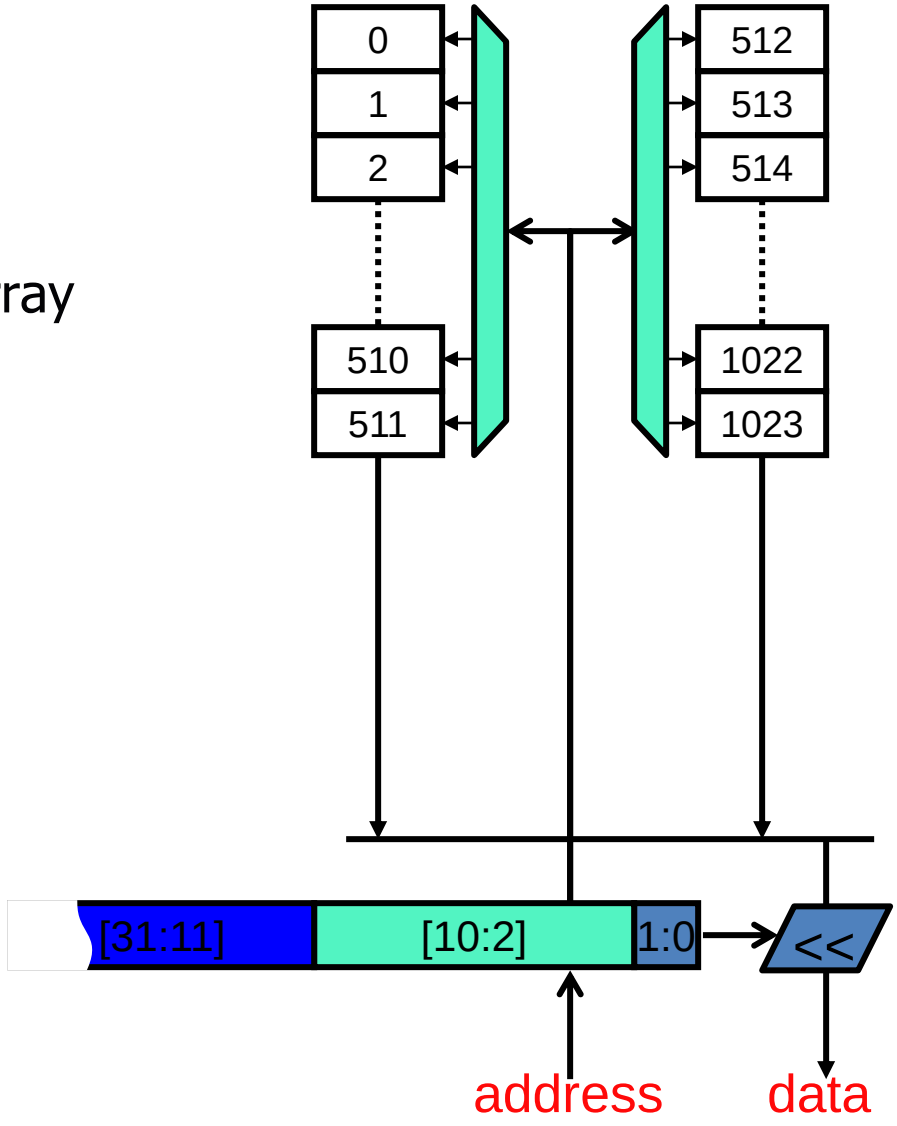
Aside: Physical Cache Layout I

- Logical layout
  - Data and tags mixed together
- Physical layout
  - Data and tags in separate RAMs



Physical Cache Layout II

- Logical layout
  - Data array is monolithic
- Physical layout
  - Each data "way" in separate array

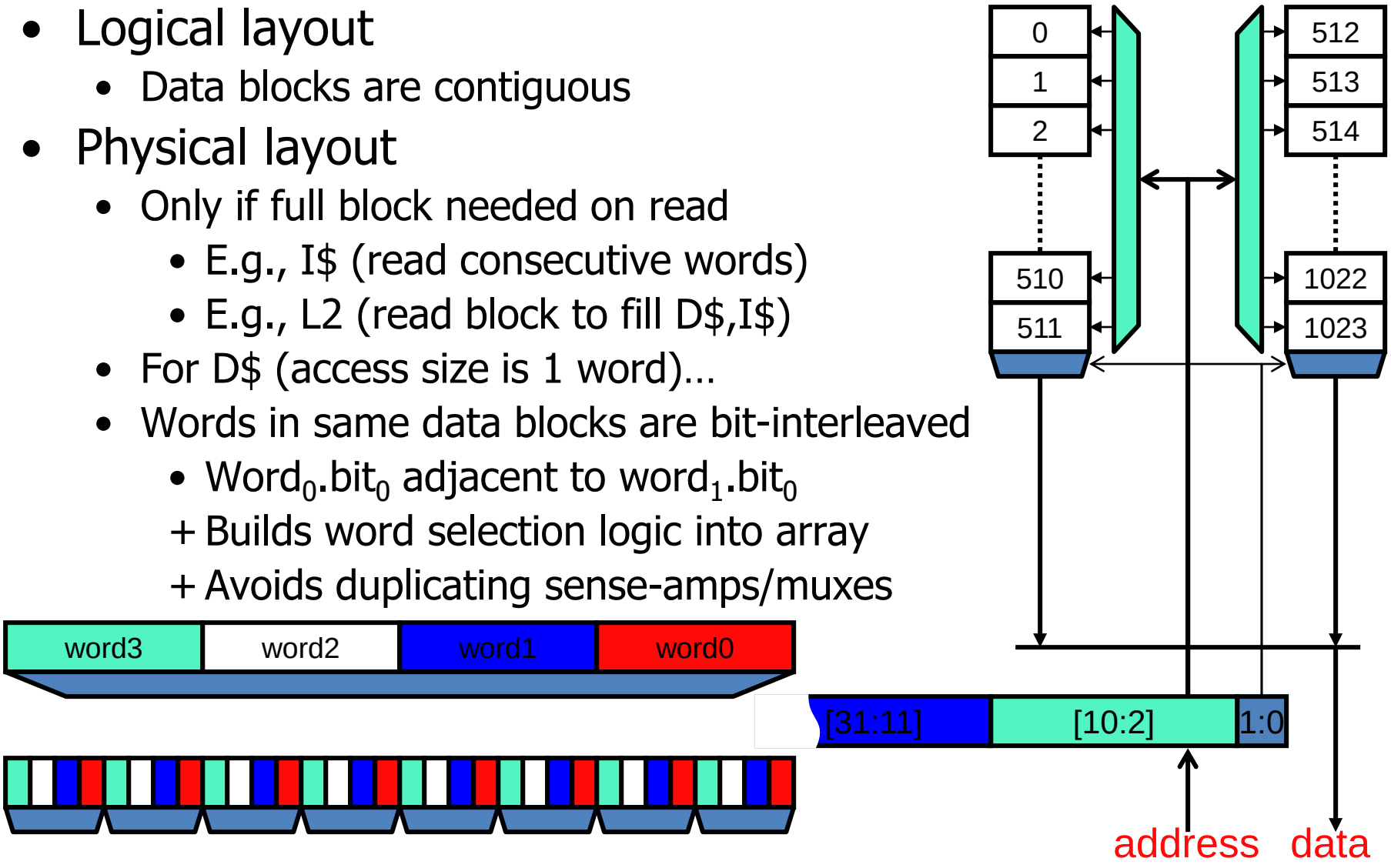


Physical Cache Layout III

- Logical layout
  - Data blocks are contiguous
- Physical layout
  - Only if full block needed on read
    - E.g., I$ (read consecutive words)
    - E.g., L2 (read block to fill D$,I$)
  - For D$ (access size is 1 word)…
  - Words in same data blocks are bit-interleaved
    - $Word_0.bit_0$ adjacent to $word_1.bit_0$

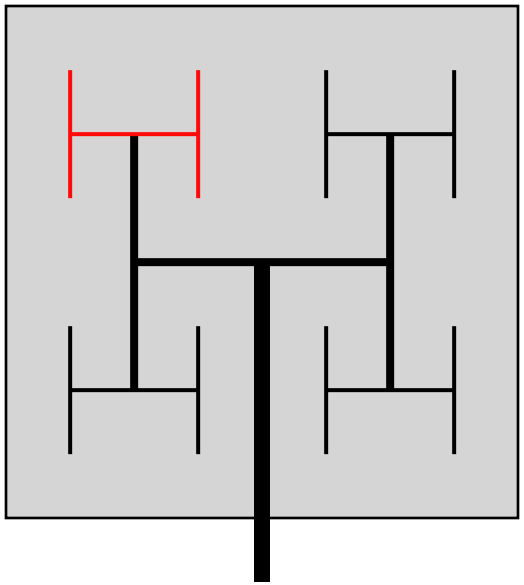
    + Builds word selection logic into array

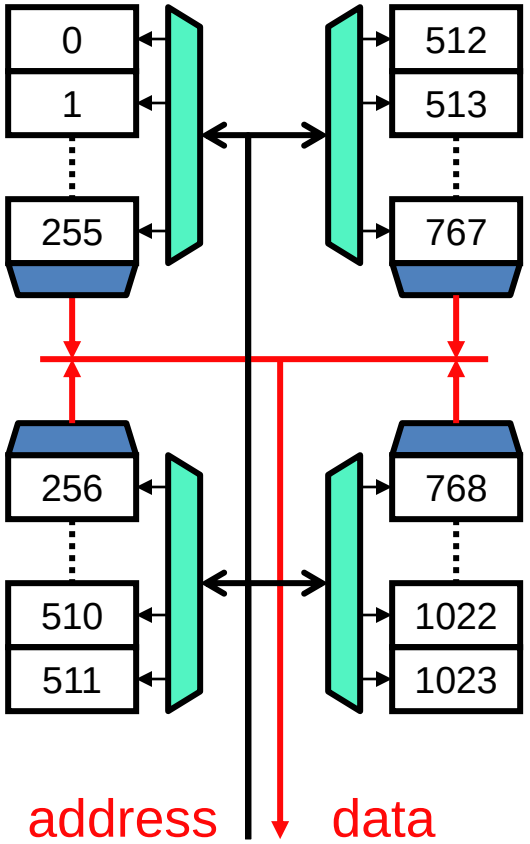
    + Avoids duplicating sense-amps/muxes



Physical Cache Layout IV

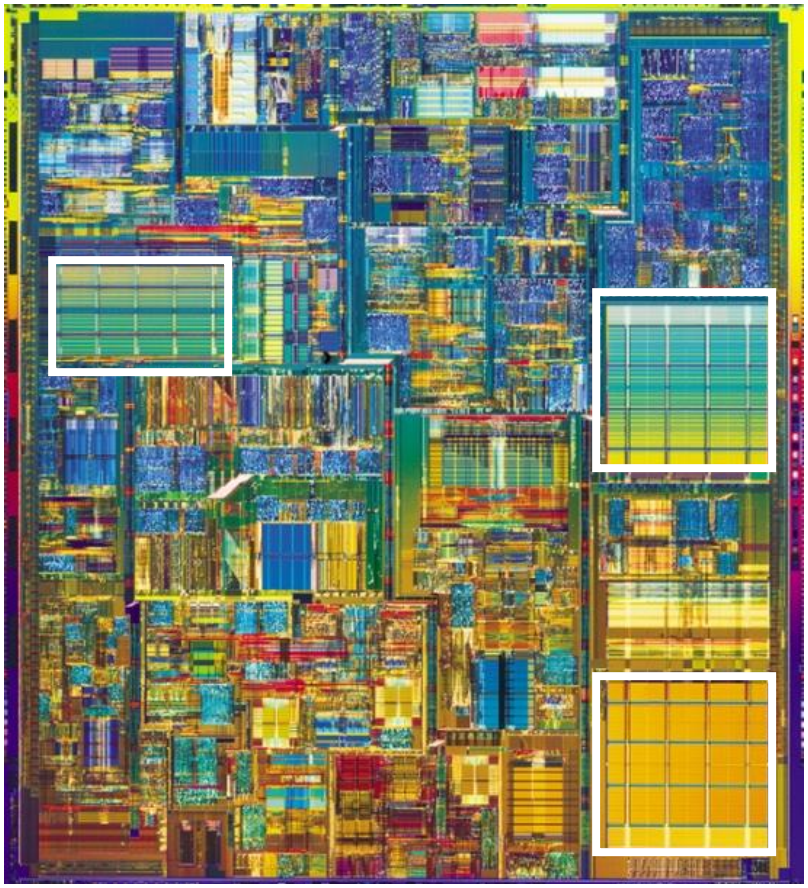
- Logical layout
  - Arrays are vertically contiguous
- Physical layout
  - Vertical partitioning to minimize wire lengths
  - **H-tree**: horizontal/vertical partitioning layout
    - Applied recursively
    - Each node looks like an H



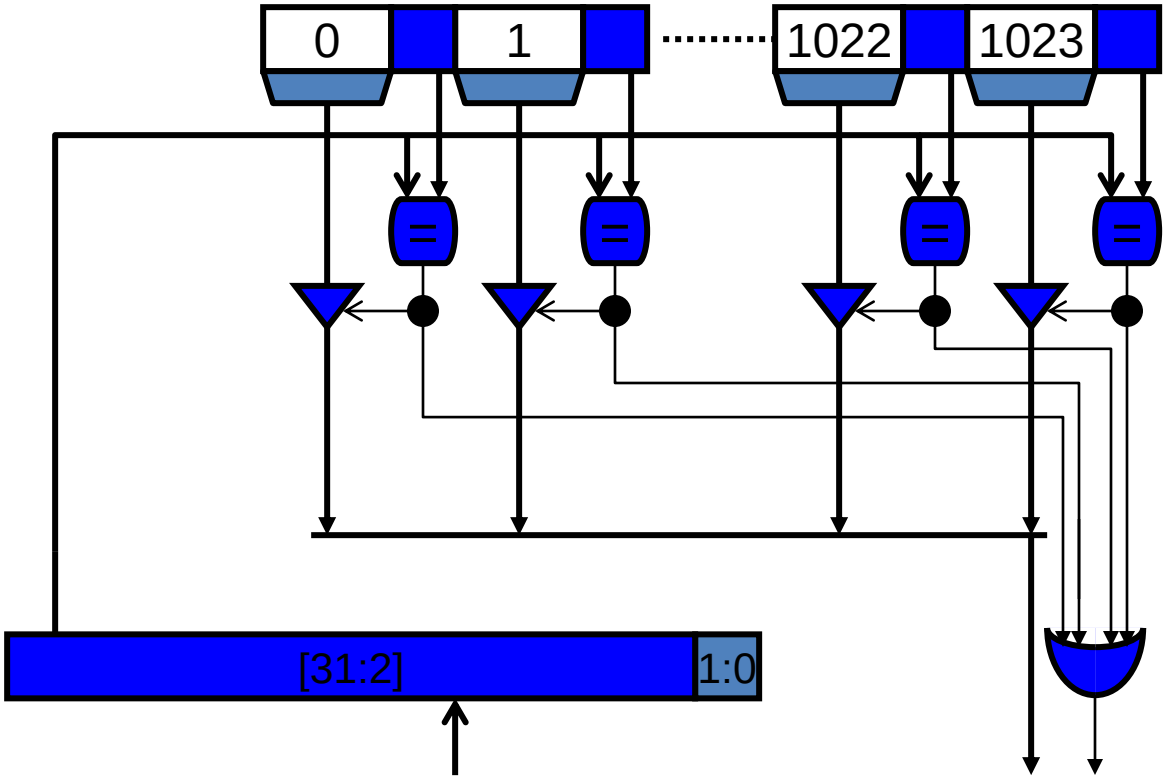


Physical Cache Layout

- Arrays and h-trees make caches easy to spot in μgraphs



Full-Associativity



- How to implement full (or at least high) associativity?
  - 1K tag matches? unavoidable, but at least tags are small
  - 1K data reads? Terribly inefficient