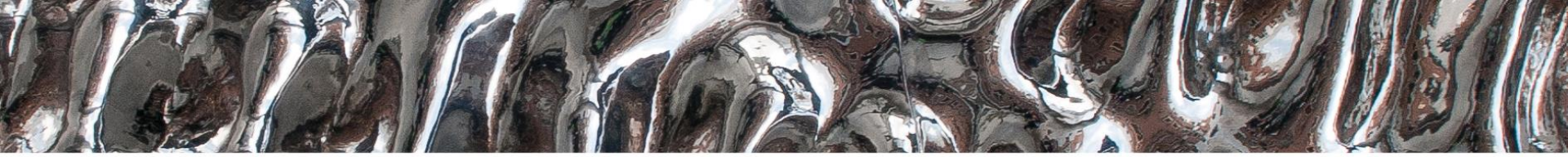


UiO : Department of Informatics
University of Oslo

IN 3160, IN4160

# Interconnect, Memory

Yngve Hafting

# Course Goals and Learning Outcome

https://www.uio.no/studier/emner/matnat/ifi/IN3160/index-eng.html

In this course you will learn about the **design of advanced digital systems**. This includes programmable logic circuits, a hardware design language and system-on-chip design (processor, memory and logic on a chip). Lab assignments provide practical experience in how real design can be made.

*After completion of the course you will*:

- understand important **principles for design** and testing of digital systems
- understand the relationship between behaviour and different construction criteria
- be able to describe advanced digital systems at different levels of detail
- be able to perform simulation and synthesis of digital systems.

***Goals for this lesson:***

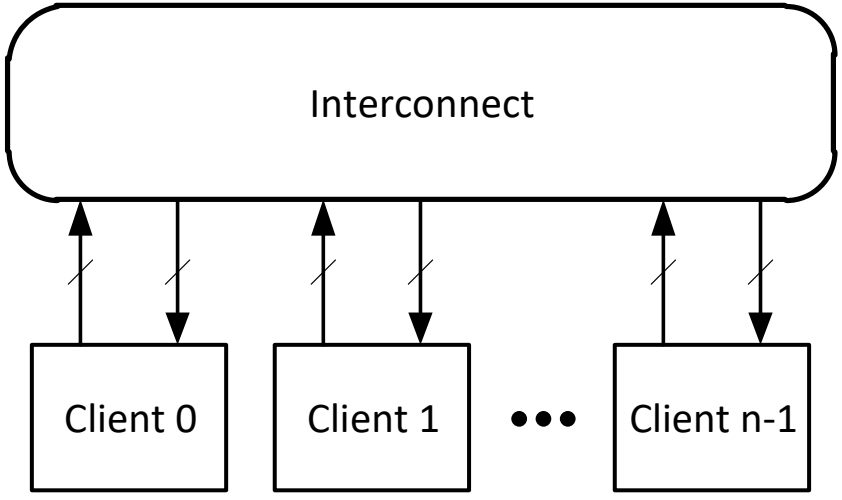
- Know terms and structure
  - Interconnect types
    - buses
    - crossbar switches
    - interconnect networks
- Know typical structures for memory address decoding
  - Bit slicing
  - Banking
  - Tiling

# Overview

- Interconnect
  - bus
  - crossbar
  - network
- Memory
  - SRAM
  - DRAM

- Memory organization
  - Banking
  - Slicing
  - Tiling

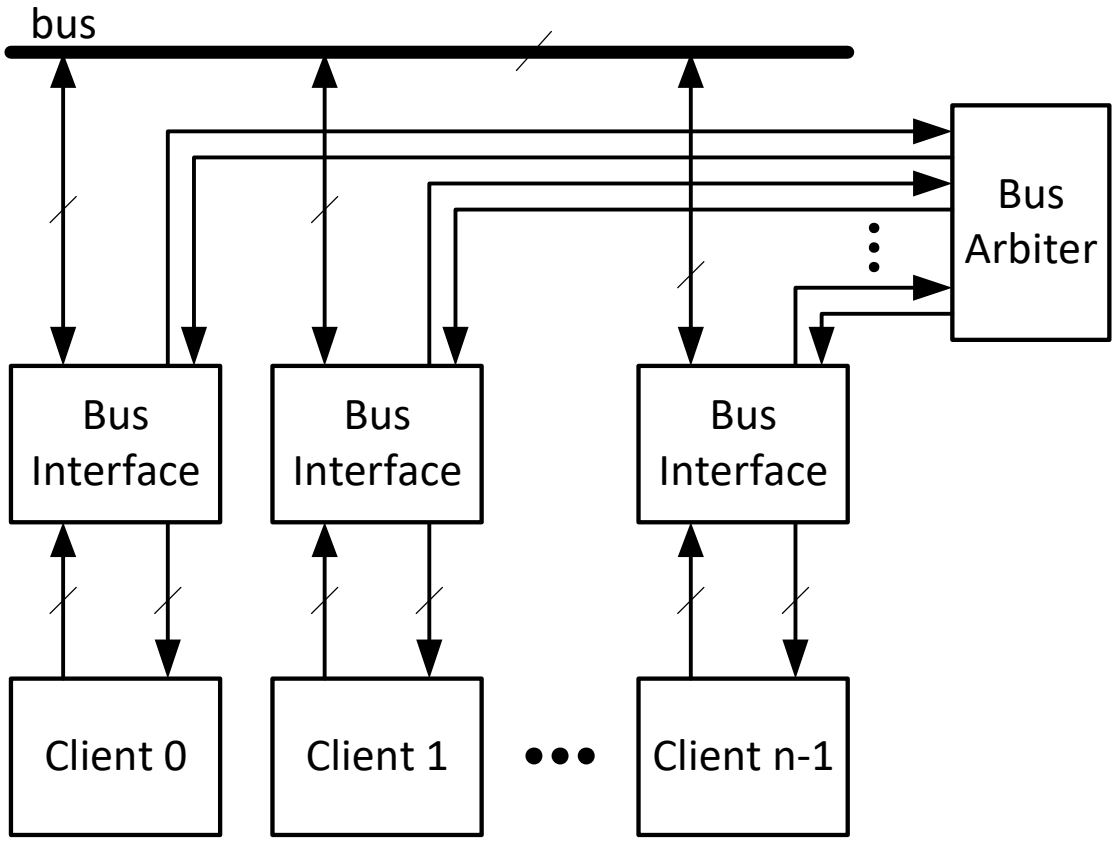
# Interconnect




- Many clients need to communicate
- Ad-hoc point-to-point wiring or shared interconnect
- Like a telephone exchange

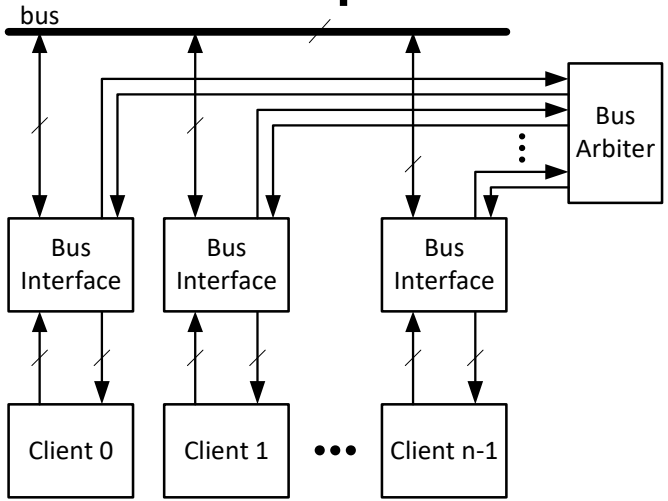
# Bus

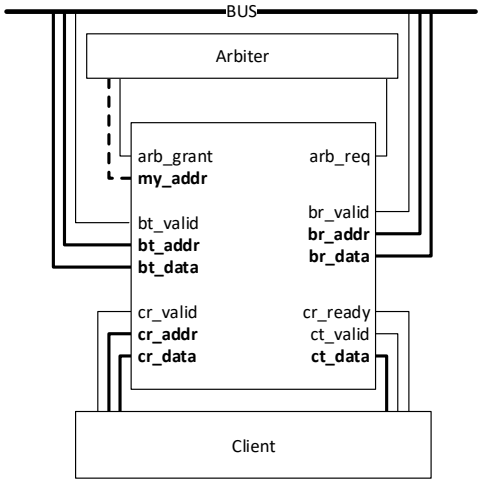
- interconnect with (multiple) clients connected to one (often multiple bit) data line.
- Only one client can send data at a time, but there can be multiple receivers.
- The bus arbiter selects which client is allowed to use the bus.
- External buses:
  - tristate
- Internal buses
  - or'ing of client signals



# VHDL for a simple bus interface



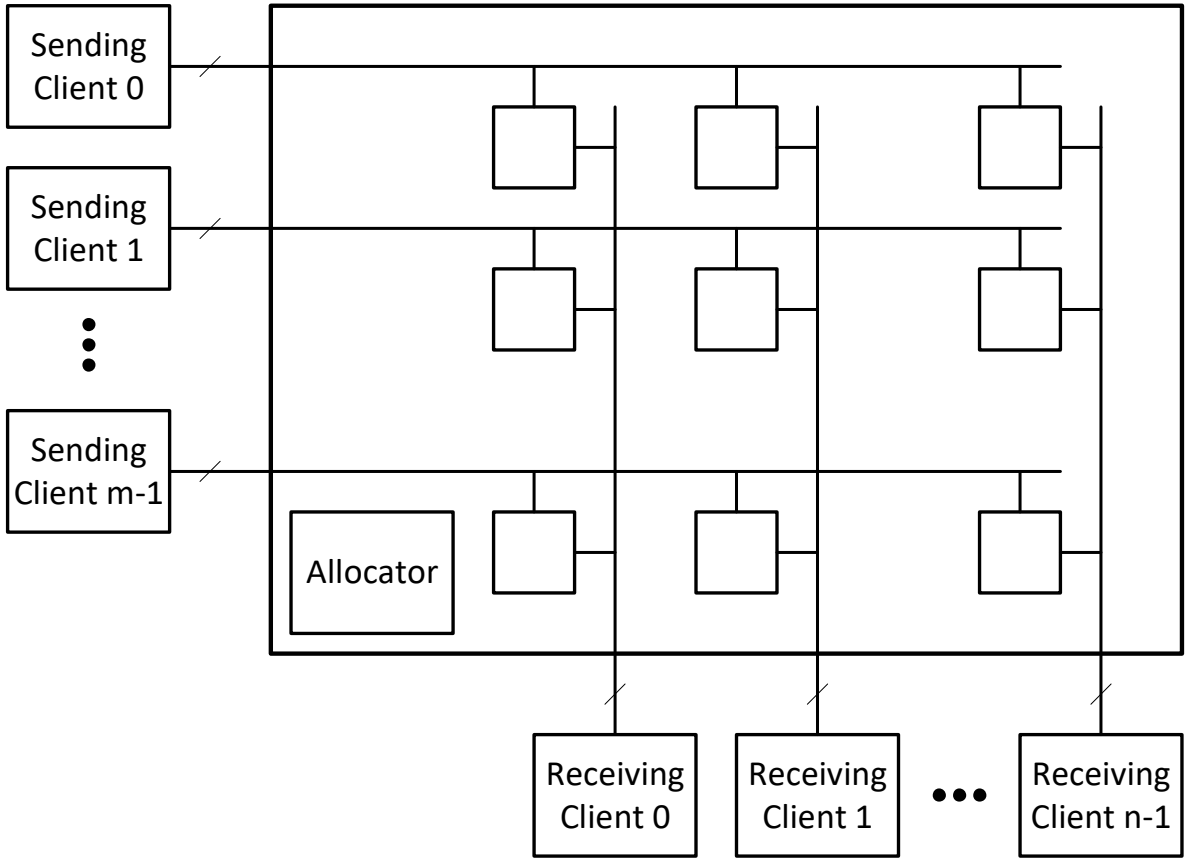


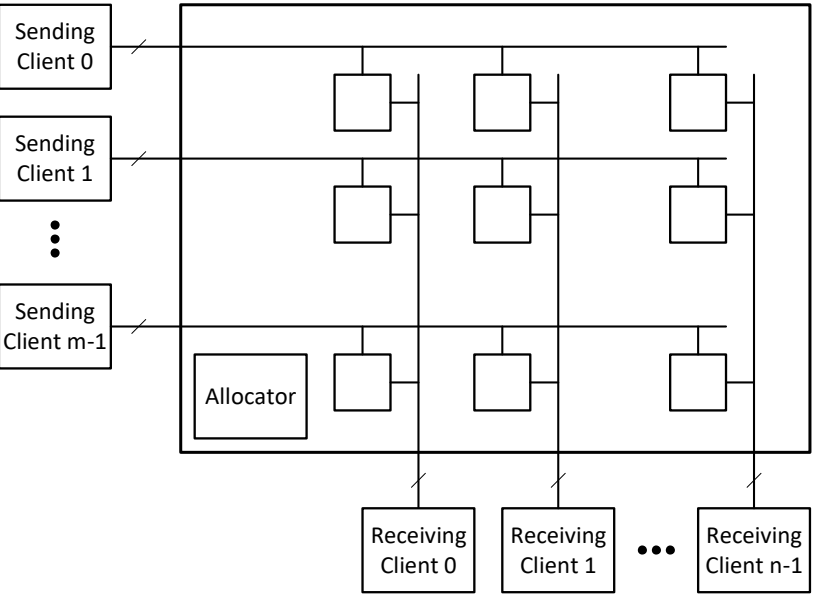

```vhdl
-- Combinational Bus Interface
-- t (transmit) and r (receive) in signal names are from the
-- perspective of the bus
library ieee;
use ieee.std_logic_1164.all;

entity BusInt is
  generic( aw: integer := 2;   -- address width
           dw: integer := 4 ); -- data width
  port( cr_valid, arb_grant, bt_valid: in std_logic;
        cr_ready, ct_valid, arb_req, br_valid: out std_logic;
        cr_addr, bt_addr, my_addr: in std_logic_vector(aw-1 downto 0);
        br_addr: out std_logic_vector(aw-1 downto 0);
        cr_data, bt_data: in std_logic_vector(dw-1 downto 0);
        br_data, ct_data: out std_logic_vector(dw-1 downto 0) );
end BusInt;

architecture impl of BusInt is
begin
  -- arbitration
  arb_req <= cr_valid;
  cr_ready <= arb_grant;

  -- bus drive
  br_valid <= arb_grant;
  br_addr <= cr_addr when arb_grant else (others => '0');
  br_data <= cr_data when arb_grant else (others => '0');

  -- bus receive
  ct_valid <= '1' when (bt_valid = '1') and (bt_addr = my_addr) else '0';
  ct_data <= bt_data ;
end impl;
```

# Crossbar Switch

- interconnect with multiple senders and receivers.
- Several senders may be active at a time, as long as a unique path can be allocated to each reciver.
- Throughput *can* be increased by allowing buffering at each crosspoint..



# Crossbar Switch 2x2 example

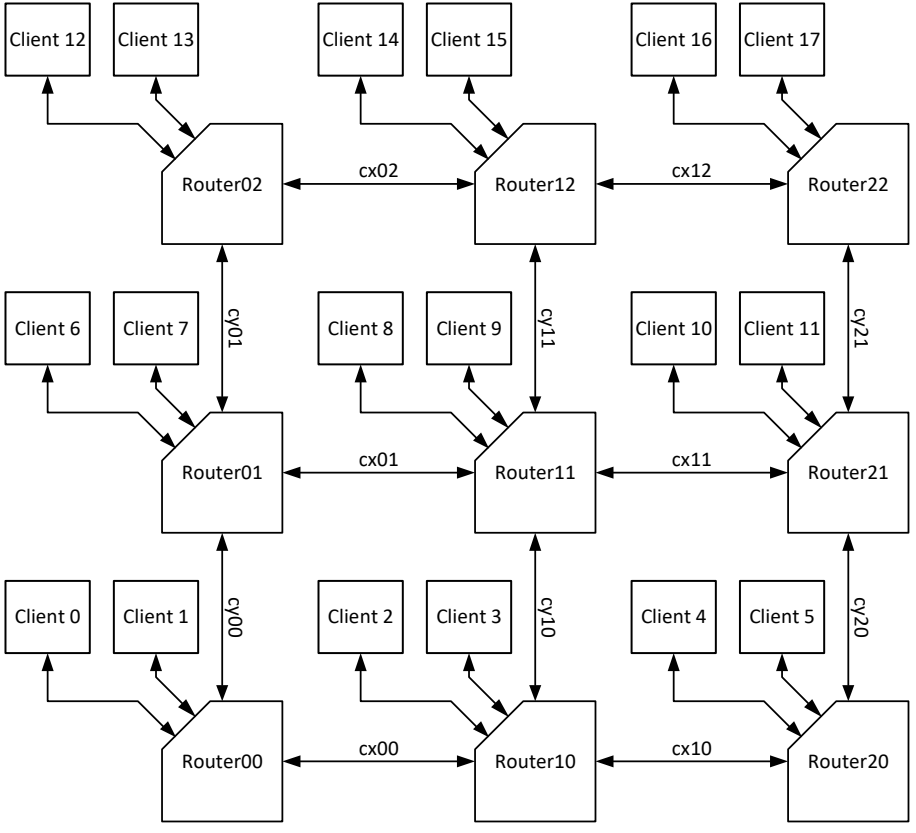



```
entity Xbar22 is
  generic( dw: integer := 4 ); -- data width
  port(
    c0r_valid, c0t_ready, c1r_valid, c1t_ready: in std_logic;
    c0r_ready, c0t_valid, c1r_ready, c1t_valid: out std_logic;
    c0r_addr, c1r_addr: in std_logic;
    c0r_data, c1r_data: in std_logic_vector(dw-1 downto 0);
    c0t_data, c1t_data: out std_logic_vector(dw-1 downto 0) );
end Xbar22;
```

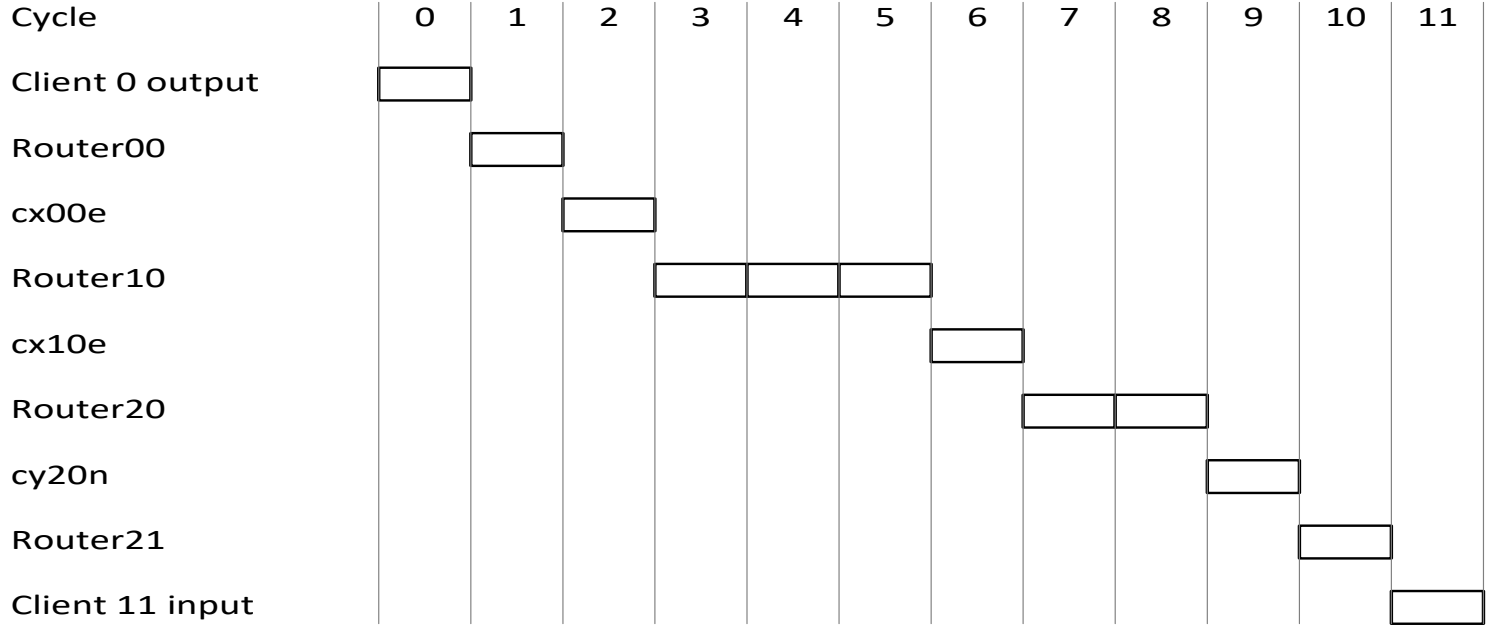
```
architecture impl of Xbar22 is
  signal req00, req01, req10, req11: std_logic;
  signal grant00, grant01, grant10, grant11: std_logic;
begin
  -- request matrix
  req00 <= '1' when not c0r_addr and c0r_valid else '0';
  req01 <= '1' when     c0r_addr and c0r_valid else '0';
  req10 <= '1' when not c1r_addr and c1r_valid else '0';
  req11 <= '1' when     c1r_addr and c1r_valid else '0';

  -- arbitration 0 wins
  grant00 <= req00;
  grant01 <= req01;
  grant10 <= req10 and not req00 ;
  grant11 <= req11 and not req01 ;

  -- connections
  c0t_valid <= (grant00 and c0r_valid) or (grant10 and c1r_valid);
  c0t_data <=  (c0r_data and (dw-1 downto 0 => grant00))  or
               (c1r_data and (dw-1 downto 0 => grant10));
  c1t_valid <= (grant01 and c0r_valid) or (grant11 and c1r_valid);
  c1t_data <=  (c0r_data and (dw-1 downto 0 => grant01)) or
               (c1r_data and (dw-1 downto 0 => grant11));

  -- ready
  c0r_ready <= (grant00 and c0t_ready) or (grant01 and c1t_ready);
  c1r_ready <= (grant10 and c0t_ready) or (grant11 and c1t_ready);
end impl;
```

# Interconnection Networks

- multiple senders and receivers.
- Packets are buffered by routers that transfer the packet in the direction of the recipient.
- Routing algorithm will be depending on the network topology.
  - star/tree topology (LAN)
  - ring topology
  - mesh topology

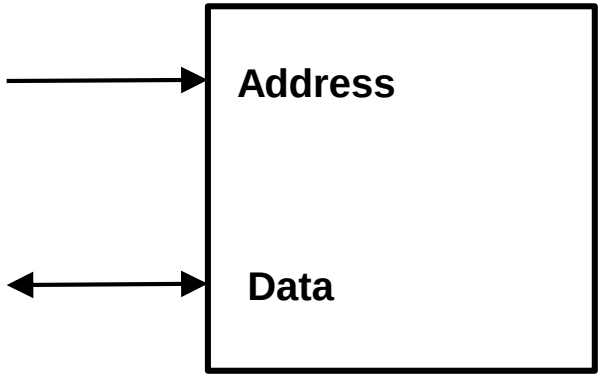


# Interconnection Networks



# Memory

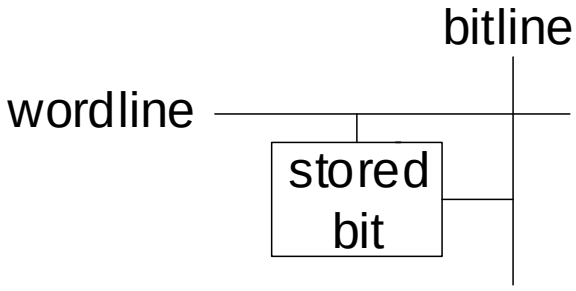
- Row of decoders, column multiplexers
- Typical restriction of array size
  - 256 x 256 = 65536 = 64k bit = 8k Byte
  - electrical restriction
- Larger memory need to be multiples of arrays at the maximum size



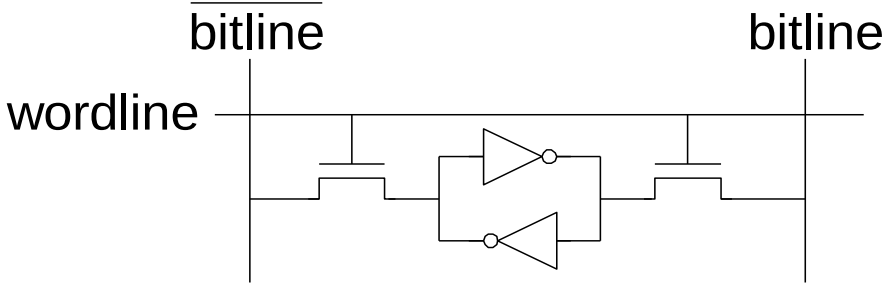

**Capacity**
**Bandwidth**
**Latency**
**Granularity**

# SRAM

- Both asynchronous and synchronous versions exist
- Stored bit is (weakly) upheld by the inverters.
- When wordline is activated
  - bitline either
    - propagates the weak value or
    - sets the bit value if it is driven (strong)

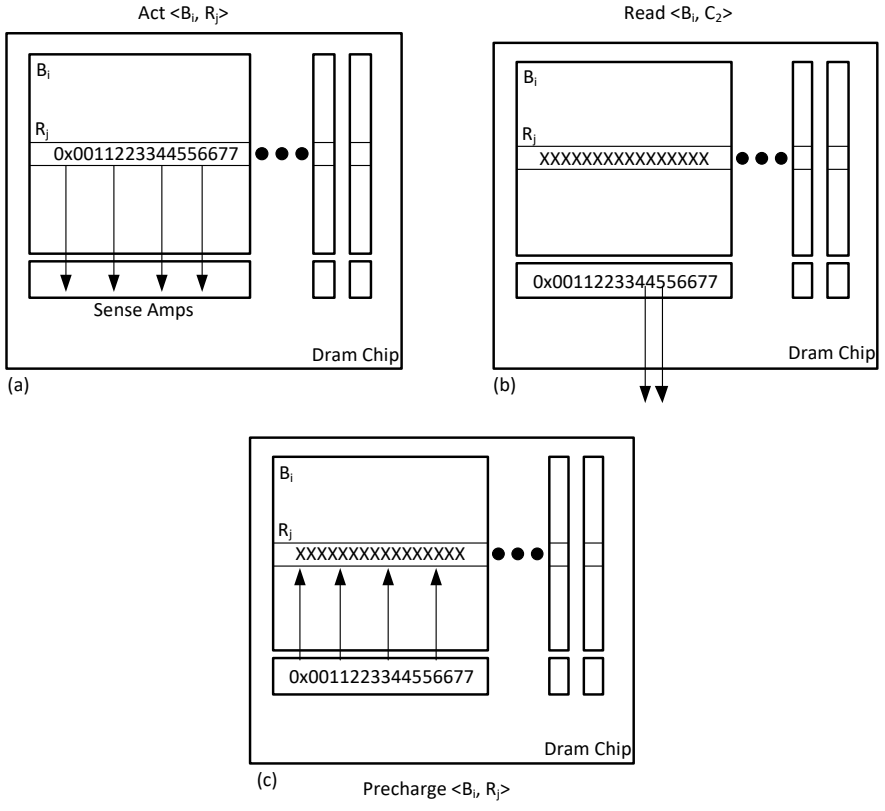



- The physical layout can be more complex, including transistors for precharging bitlines etc.



# DRAM Operation

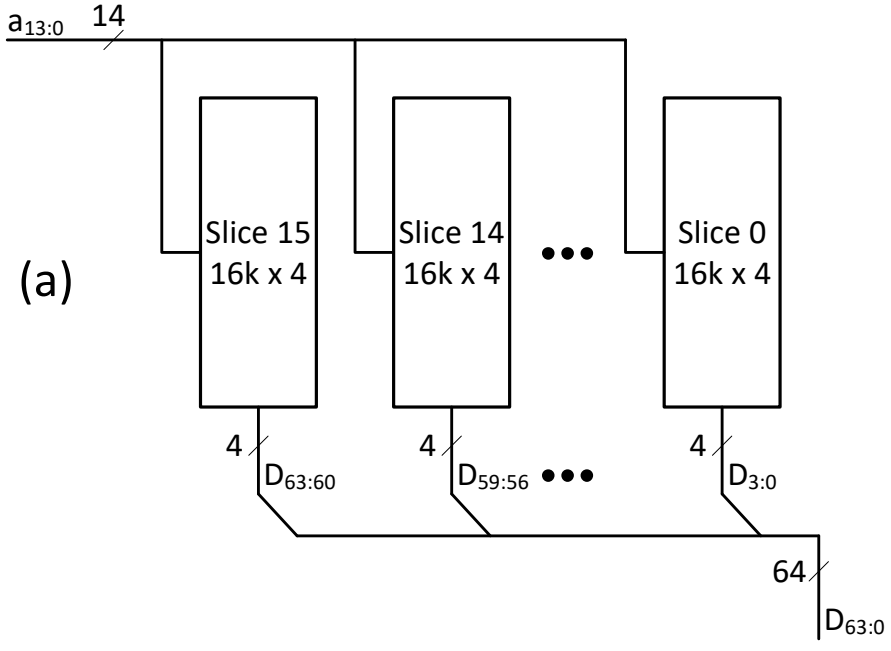
- For each read, the stored capacitor charge is removed, and thus need to be re-written to remain stored.
- Several columns in one row can be read before rewrite.
- Rewrite (D&H: precharge operation) takes a certain time
  - DRAM stores bit as charge on capacitors.
- RAM cells need periodically refresh that performs read/write



**What if you need more memory or more bandwidth than one primitive?**

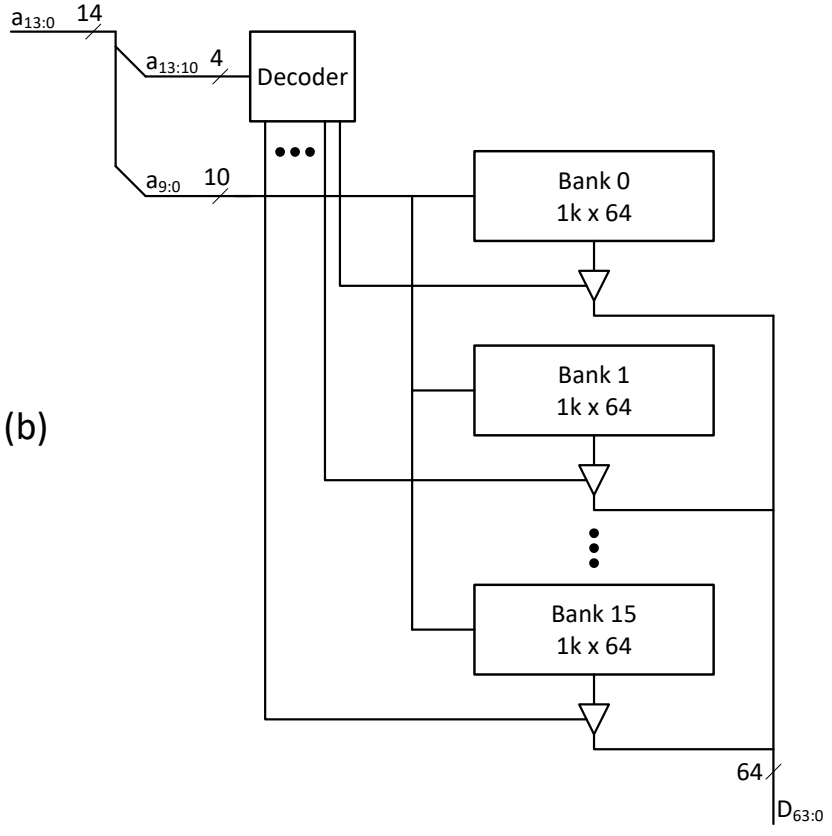
# Bit-Slicing

- Several slices shares the same address
- Data is read from a row of slices in parallel.



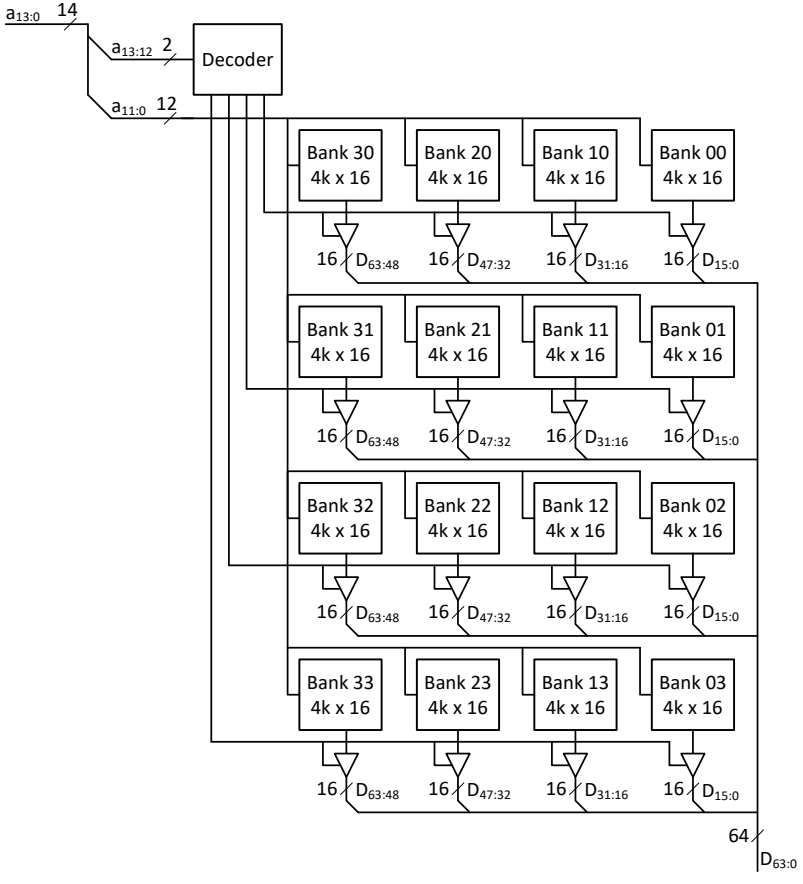
# Banking

- Banked memory is organized in columns of array banks
- Parts of the address is decoded to enable output from the selected bank only.
  - May be used to have the other banks idle and save power.



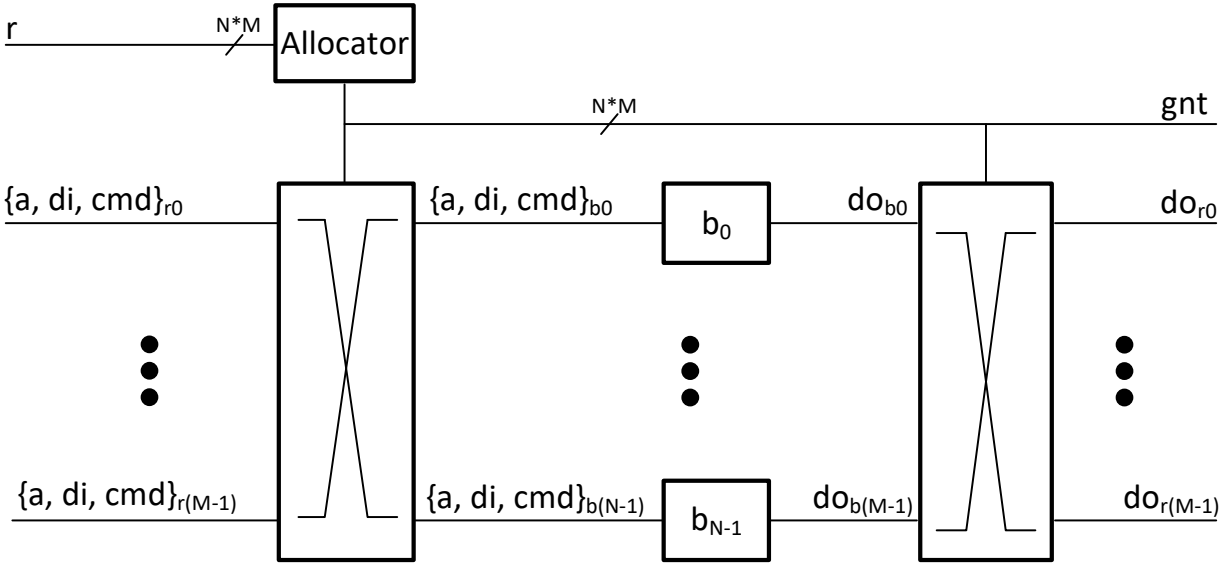

(b)

# Tiling =
# Bit slicing & banking

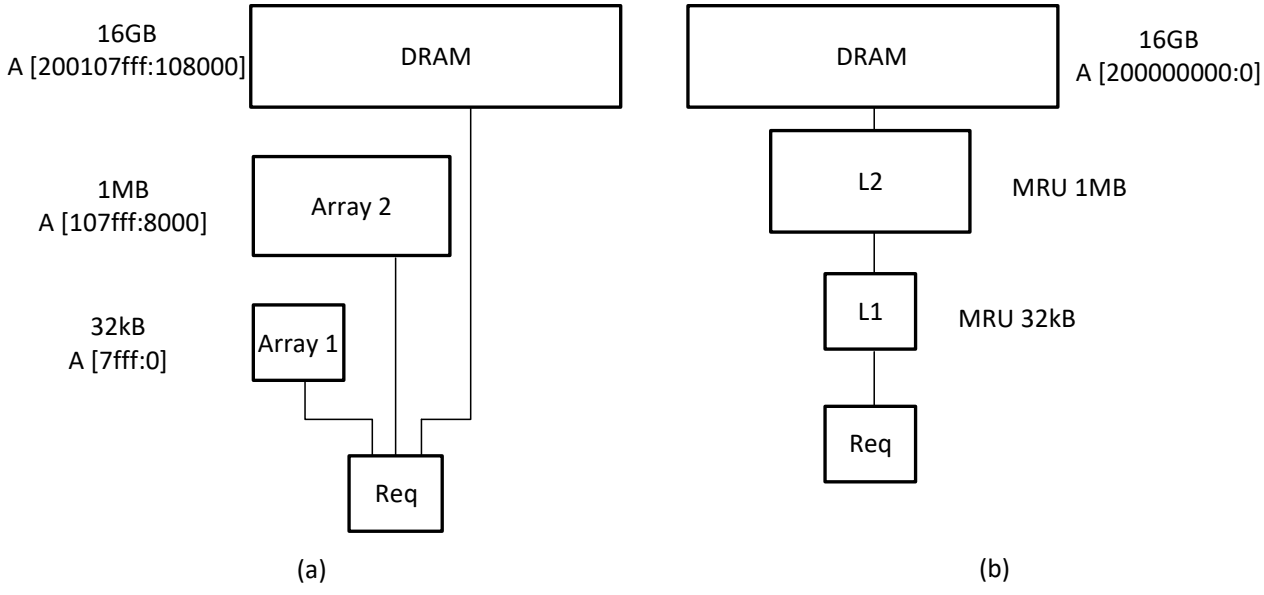


# Interleaving

- Using a crossbar switch for a tiled set of banks
- Allow for multiple read or writes during the same cycle
- Ex: Quad data rate (QDR) RAM



# Hierarchy (->IN2060)



# Summary

- DHA
  - 24-25.3, p 521-540
  - (25.4 can be read to connect the dots from IN2060)