1. Fully associative caches
   a. Any block of main memory can go into any line
   b. W = LIC, S = 1
      i. Entire cache is a whole set
      ii. Any line of cache can go into any way of that set
      iii. Number of set bits are always 0
   c. Won't have conflicts like with a DM cache
      i. Any line can go anywhere
   d. However, extremely expensive to implement in terms of both power and money
      i. Must check all tags at once to see if data is in cache
   e. Example
      i. Same cache parameters as before, except now a FA cache
      ii. 8-byte FA cache with line size of 2, and 4-bit address
      iii. LIC = C / LS = 8 / 2 = 4
      iv. For a FA cache, S = 1

| **Tag** | **Set** | **Offset** | | **Address Bits** |
|---|---|---|---|---|
| 4 – 1 – 0 = 3 bits | $\log_2 S = \log_2 1 = 0$ bits | $\log_2 LS = \log_2 2 = 1$ bit | = | 4 bits |

2. Cache mapping example
   a. Use same addresses and data from DM cache example previously

| Address | Data |
|---|---|
| 0110 | 0x1B |
| 0111 | 0x59 |
| 1000 | 0xFE |
| 1001 | 0x3D |
| 1110 | 0x0C |
| 1111 | 0x3A |
| 1010 | 0x25 |
| 1011 | 0x98 |

   b. Load addresses into cache in same fashion
      i. However, this time there's no set bits
      ii. Any line of cache can go anywhere, as we see in the next table

| Memory Access | Tag | Offset | Access Type | Data | Hit or Miss |
|---|---|---|---|---|---|
| 0110 | 011 | 0 | Read | | Miss |
| 1000 | 100 | 0 | Read | | Miss |
| 1110 | 111 | 0 | Read | | Miss |
| 1010 | 101 | 0 | Write | 0xBE | Miss |
| 1011 | 101 | 1 | Write | 0xEF | Hit |

c. Place values from first three accesses into table below
    i. First three accesses fill up first three lines of cache
    ii. Each one is a miss, but they bring in the entire line
    iii. No conflict between 0110 and 1110 this time
    iv. Dirty bit indicates that line has been modified since brought in from memory
        1. Since we only read for the first three accesses, dirty bits are 0 for those

| **Line in Cache** | **Tag** | **Byte 0** | **Byte 1** | **Dirty** |
|---|---|---|---|---|
| 00 | 011 | 1B | 59 | 0 |
| 01 | 100 | FE | 3D | 0 |
| 10 | 111 | 0C | 3A | 0 |
| 11 | | | | |

d. Now we introduce writes
    i. Write changes the value in the cache
    ii. The first write misses, so we must bring the original values in from memory
        1. Known as *write-allocate* policy
        2. Once we bring it in, *then* we modify the 0th byte
        3. Set dirty bit for this line since we modified the value from memory
    iii. Second write hits, so we don't need to bring the line in
        1. Change the 1st byte to the new value
        2. Dirty bit stays 1 here

| **Line in Cache** | **Tag** | **Byte 0** | **Byte 1** | **Dirty** |
|---|---|---|---|---|
| 00 | 011 | 1B | 59 | 0 |
| 01 | 100 | FE | 3D | 0 |
| 10 | 111 | 0C | 3A | 0 |
| 11 | 101 | ~~25~~ BE | ~~98~~ EF | 1 |

3. Set associative caches
    a. Compromise between DM and FA
    b. N-way SA cache means there's N lines in each set
        i. Thus, there's N different places a line can go into
        ii. W = N(-way)
    c. Advantages and disadvantages of both DM and FA
        i. More options to place lines, so less conflicts than a DM cache
        ii. Will still have conflicts compared to a FA cache
        iii. More expensive in power and cost than a DM cache
        iv. Cheaper and less power than a FA cache
    d. Example
        i. Same cache parameters as before, except now a 2-way SA cache
        ii. 8-byte 2-way SA cache with line size of 2, and 4-bit address
        iii. LIC = C / LS = 8 / 2 = 4
        iv. For a SA cache, S = LIC / W = 4 / 2 = 2

| **Tag** | **Set** | **Offset** | | **Address Bits** |
|---|---|---|---|---|
| 4 – 2 – 0 = 2 bits | $\log_2 S = \log_2 2 = 1$ bit | $\log_2 LS = \log_2 2 = 1$ bit | = | 4 bits |