Review: Explicit Renaming/Limits to ILP

° **Explicit Renaming:** more physical registers than ISA.
  - Separates *renaming* from *scheduling*
    - Opens up lots of options for resolving RAW hazards
  - Rename table: tracks current association between architectural registers and physical registers
  - Potentially complicated rename table management

° **Multi-issue:** simple matter of accounting
  - Must do dataflow analysis across multiple instructions simultaneously
  - Rename table updated as if instructions happened serially!

° **To sustain:** need execution bandwidth+commit bandwidth
  - To sustain ILP of X need at least
    - X-way issue, > X execution bandwidth (for mix), X way commit

° **Limits to ILP**
  - Inherent parallelism of applications as high as 150 IPC
  - Realistic limits rapidly reduce this to < 4 IPC for most applications

Recall: Upper Limit to ILP: Ideal Machine

Recall: More Realistic HW: Branch Impact

Change from Infinite window to examine to 2000 and maximum issue of 64 instructions per clock cycle

- Integer: 18 - 60
  - Integer: 6 - 12

- FP: 75 - 150
  - FP: 15 - 45

Perfect  Pick Cor. or BHT  BHT (512)  Profile  No prediction
The Big Picture: Where are We Now?

° The Five Classic Components of a Computer

- Processor
  - Control
  - Datapath
- Memory
  - Input
  - Output

° Today’s Topics:
  - Recap last lecture
  - Locality and Memory Hierarchy
  - Administrivia
  - SRAM Memory Technology
  - DRAM Memory Technology
  - Memory Organization

Recall: Who Cares About the Memory Hierarchy?

Processor-DRAM Memory Gap (latency)

Recall: Memory Hierarchy of a Modern Computer System

° By taking advantage of the principle of locality:
  - Present the user with as much memory as is available in the cheapest technology.
  - Provide access at the speed offered by the fastest technology.

Impact of Memory Hierarchy on Algorithms

° Today CPU time is a function of (ops, cache misses)
° What does this mean to Compilers, Data structures, Algorithms?
  - Quicksort: fastest comparison based sorting algorithm when keys fit in memory
  - Radix sort: also called “linear time” sort
    - For keys of fixed length and fixed radix a constant number of passes over the data is sufficient independent of the number of keys
  - For Alphastation 250, 32 byte blocks, direct mapped L2 2MB cache, 8 byte keys, from 4000 to 4000000
Quicksort vs. Radix as vary number keys: Instructions

<table>
<thead>
<tr>
<th>Job size in keys</th>
<th>Quick (Instr/key)</th>
<th>Radix (Instr/key)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1E+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1E+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1E+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1E+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1E+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1E+</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Quicksort vs. Radix as vary number keys: Instrs & Time

<table>
<thead>
<tr>
<th>Job size in keys</th>
<th>Quick (Instr/key)</th>
<th>Radix (Instr/key)</th>
<th>Quick (Clocks/key)</th>
<th>Radix (Clocks/key)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1E+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1E+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1E+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1E+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1E+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1E+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Quicksort vs. Radix as vary number keys: Cache misses

<table>
<thead>
<tr>
<th>Job size in keys</th>
<th>Quick (miss/key)</th>
<th>Radix (miss/key)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1E+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1E+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1E+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1E+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1E+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1E+</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What is proper approach to fast algorithms?

Main Memory Background

- **Performance of Main Memory:**
  - Latency: Cache Miss Penalty
    - Access Time: time between request and word arrives
    - Cycle Time: time between requests
  - Bandwidth: I/O & Large Block Miss Penalty (L2)

- **Cache uses SRAM**: Static Random Access Memory
  - No refresh (6 transistors/bit vs. 1 transistor)
    - Size: DRAM/SRAM - 4-8
    - Cost/Cycle time: SRAM/DRAM - 8-16

- **Main Memory is DRAM**: Dynamic Random Access Memory
  - Dynamic since needs to be refreshed periodically (8 ms)
  - Addresses divided into 2 halves (Memory as a 2D matrix):
    - RAS or Row Address Strobe
    - CAS or Column Address Strobe
Why do computer designers need to know about RAM technology?
- Processor performance is usually limited by memory bandwidth
- As IC densities increase, lots of memory will fit on processor chip
  - Tailor on-chip memory to specific needs
    - Instruction cache
    - Data cache
    - Write buffer

What makes RAM different from a bunch of flip-flops?
- Density: RAM is much denser

Static RAM Cell

Write:
1. Drive bit lines (bit=1, bit=0)
2. Select row

Read:
1. Precharge bit and bit to Vdd or Vdd/2 ⇒ make sure equal!
2. Select row
3. Cell pulls one line low
4. Sense amp on column detects difference between bit and bit

Word 0
Word 1
Word 15

Address Decoder

2^N words x M bit SRAM

Logic Diagram of a Typical SRAM

Write Enable is usually active low (WE_L)

Din and Dout are combined to save pins:
- A new control signal, output enable (OE_L) is needed
- WE_L is asserted (Low), OE_L is disasserted (High)
  - D serves as the data input pin
- WE_L is disasserted (High), OE_L is asserted (Low)
  - D is the data output pin
- Both WE_L and OE_L are asserted:
  - Result is unknown. Don’t do that!!!

Although could change VHDL to do what desire, must do the best with what you’ve got (vs. what you need)
Typical SRAM Timing

- **Write Timing:**
  - \(2^N\) words
  - \(x\) \(M\) bit SRAM

- **Read Timing:**
  - \(WE_L\)
  - \(A\)
  - \(OE_L\)

Write Timing:

- \(D\)
- \(D\)
- \(A\)
- \(WE_L\)
- \(OE_L\)

Read Timing:

- \(D\)
- \(D\)
- \(A\)
- \(WE_L\)
- \(OE_L\)

Problems with SRAM

- **Six transistors use up a lot of area**
- **Consider a “Zero” is stored in the cell:**
  - Transistor \(N1\) will try to pull “bit” to 0
  - Transistor \(P2\) will try to pull “bit bar” to 1
- **But bit lines are precharged to high: Are \(P1\) and \(P2\) necessary?**

Administrative Issues

- **Should be reading Chapter 7 of your book**
  - Some of the advanced things in the graduate text
- **Second midterm: Monday May 5th**
  - Pipelining
    - Hazards, branches, forwarding, CPI calculations
    - (may include something on dynamic scheduling)
  - Memory Hierarchy
  - Possibly something on I/O (see where we get in lectures)
  - Possibly something on power (Broderson Lecture)
- **Lab 5: Hopefully all is well!**
  - There are 2 DRAMs on the board. They are 2Mx16x4bank memories
    - Simulate them this way. Use 2 DRAMs
  - Consider “fake processor” written in Verilog to debug memory system?
    - Make sure to try writes to individual words and reads from adjacent words in the cache
  - Careful about 2-clocks for DRAM module
    - Note: you should be able to single-step your processor while DRAM continues to run at full speed....
- **No 2-port caches!**
  - Must build cache controllers that stall access to cache during fill

Main Memory Deep Background

- **“Out-of-Core”, “In-Core,” “Core Dump”?**
- **“Core memory”?**
- **Non-volatile, magnetic**
- **Lost to 4 Kbit DRAM (today using 64Mbit DRAM)**
- **Access time 750 ns, cycle time 1500-3000 ns**
1-Transistor Memory Cell (DRAM)

° Write:
  • 1. Drive bit line
  • 2. Select row

° Read:
  • 1. Precharge bit line to Vdd/2
  • 2. Select row
  • 3. Cell and bit line share charges
    - Very small voltage changes on the bit line
  • 4. Sense (fancy sense amp)
    - Can detect changes of ~1 million electrons
  • 5. Write: restore the value

° Refresh
  • 1. Just do a dummy read to every cell.

DRAM Capacitors: more capacitance in a small area

° Trench capacitors:
  • Logic ABOVE capacitor
  • Gain in surface area of capacitor
  • Better Scaling properties
  • Better Planarization

° Stacked capacitors
  • Logic BELOW capacitor
  • Gain in surface area of capacitor
  • 2-dim cross-section quite small

Classical DRAM Organization (square)

° Row and Column Address together:
  • Select 1 bit a time

DRAM logical organization (4 Mbit)

° Square root of bits per RAS/CAS
**DRAM physical organization (4 Mbit)**

<table>
<thead>
<tr>
<th>Column Address</th>
<th>I/O</th>
<th>I/O</th>
<th>I/O</th>
<th>8 I/Os</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row Address</td>
<td>I/O</td>
<td>I/O</td>
<td>I/O</td>
<td>8 I/Os</td>
</tr>
</tbody>
</table>

Block 0

Block 3

---

**Logic Diagram of a Typical (Asynchronous) DRAM**

- Control Signals (RAS_L, CAS_L, WE_L, OE_L) are all active low
- Din and Dout are combined (D):
  - WE_L is asserted (Low), OE_L is disasserted (High)
    - D serves as the data input pin
  - WE_L is disasserted (High), OE_L is asserted (Low)
    - D is the data output pin
- Row and column addresses share the same pins (A):
  - RAS_L goes low: Pins A are latched in as row address
  - CAS_L goes low: Pins A are latched in as column address
  - RAS/CAS edge-sensitive

---

**DRAM Read Timing**

- Every DRAM access begins at:
  - The assertion of the RAS_L
  - 2 ways to read: early or late v. CAS

<table>
<thead>
<tr>
<th>RAS_L</th>
<th>CAS_L</th>
<th>WE_L</th>
<th>OE_L</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>256K x 8 DRAM D</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Early Read Cycle: OE_L asserted before CAS_L Late Read Cycle: OE_L asserted after CAS_L

---

**DRAM Write Timing**

- Every DRAM access begins at:
  - The assertion of the RAS_L
  - 2 ways to write: early or late v. CAS

<table>
<thead>
<tr>
<th>RAS_L</th>
<th>CAS_L</th>
<th>WE_L</th>
<th>OE_L</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>256K x 8 DRAM D</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Early Wr Cycle: WE_L asserted before CAS_L Late Wr Cycle: WE_L asserted after CAS_L
Key DRAM Timing Parameters

- $t_{RAC}$: minimum time from RAS line falling to the valid data output.
  - Quoted as the speed of a DRAM
  - A fast 4Mb DRAM $t_{RAC} = 60$ ns
- $t_{RC}$: minimum time from the start of one row access to the start of the next.
  - $t_{RC} = 110$ ns for a 4Mbit DRAM with a $t_{RAC}$ of 60 ns
- $t_{CAC}$: minimum time from CAS line falling to valid data output.
  - 15 ns for a 4Mbit DRAM with a $t_{RAC}$ of 60 ns

DRAM Performance

- A 60 ns ($t_{RAC}$) DRAM can
  - perform a row access only every 110 ns ($t_{RC}$)
  - perform column access ($t_{CAC}$) in 15 ns, but time between column accesses is at least 35 ns ($t_{PC}$).
    - In practice, external address delays and turning around buses make it 40 to 50 ns
- These times do not include the time to drive the addresses off the microprocessor nor the memory controller overhead.
  - Drive parallel DRAMs, external memory controller, bus to turn around, SIMM module, pins...
  - 180 ns to 250 ns latency from processor to memory is good for a “60 ns” ($t_{RAC}$) DRAM

Something new: Structure of Tunneling Magnetic Junction

- Tunneling Magnetic Junction RAM (TMJ-RAM)
  - Speed of SRAM, density of DRAM, non-volatile (no refresh)
  - “Spintronics”: combination quantum spin and electronics
  - Same technology used in high-density disk-drives

Main Memory Performance

- Simple:
  - CPU, Cache, Bus, Memory same width (32 bits)
- Interleaved:
  - CPU, Cache, Bus 1 word: Memory N Modules (4 Modules); example is word interleaved
- Wide:
  - CPU/Mux 1 word; Mux/Cache, Bus, Memory N words (Alpha: 64 bits & 256 bits)
Main Memory Performance

- **DRAM (Read/Write) Cycle Time** >> DRAM (Read/Write) Access Time
  - 2:1; why?

- **DRAM (Read/Write) Cycle Time**: How frequent can you initiate an access?
  - Analogy: A little kid can only ask his father for money on Saturday

- **DRAM (Read/Write) Access Time**: How quickly will you get what you want once you initiate an access?
  - Analogy: As soon as he asks, his father will give him the money

- **DRAM Bandwidth Limitation analogy**: What happens if he runs out of money on Wednesday?

Increasing Bandwidth - Interleaving

Access Pattern without Interleaving:

- Start Access for D1
- D1 available
- Start Access for D2

Access Pattern with 4-way Interleaving:

- Access Bank 0
- Access Bank 1
- Access Bank 2
- Access Bank 3

We can Access Bank 0 again

Increasing Bandwidth - Interleaving

- **Timing model**
  - 1 to send address,
  - 4 for access time, 10 cycle time, 1 to send data
  - Cache Block is 4 words
  - **Simple M.P.** = 4 \times (1+10+1) = 48
  - **Wide M.P.** = 1 + 10 + 1 = 12
  - **Interleaved M.P.** = 1+10+1 + 3 =15

Independent Memory Banks

- **How many banks?**
  - number banks ≥ number clocks to access word in bank
    - For sequential accesses, otherwise will return to original bank before it has next word ready
    - **Prime number of banks: good for a variety of access patterns**

- **Increasing DRAM => fewer chips => harder to have banks**
  - Growth bits/chip DRAM : 50%-60%/yr
  - Nathan Myrvold M/S: mature software growth (33%/yr for NT) - growth MB/$ of DRAM (25%-30%/yr)
Fewer DRAMs/System over Time

- DRAM Generation:
  - '86
  - '89
  - '92
  - '96
  - '99
  - '02
  - Memory per System growth:
    - 1 Mb
    - 4 Mb
    - 16 Mb
    - 64 Mb
    - 256 Mb
    - 1 Gb
  - Memory per DRAM growth:
    - 4 MB
    - 8 MB
    - 16 MB
    - 32 MB
    - 64 MB
    - 128 MB
    - 256 MB

- Minimum PC Memory Size:
  - 4 MB
  - 8 MB
  - 16 MB
  - 32 MB
  - 64 MB
  - 128 MB
  - 256 MB
  - @ 25%-30% / year

Fast Page Mode Operation

- Regular DRAM Organization:
  - N rows x N column x M-bit
  - Each M-bit access requires a RAS / CAS cycle

- Fast Page Mode DRAM:
  - N x M “SRAM” to save a row
  - After a row is read into the register:
    - Only CAS is needed to access other M-bit blocks on that row
    - RAS_L remains asserted while CAS_L is toggled

Key DRAM Timing Parameters

- \( t_{RAC} \): minimum time from RAS line falling to the valid data output.
  - Quoted as the speed of a DRAM
  - A fast 4Mb DRAM \( t_{RAC} = 60 \text{ ns} \)

- \( t_{RC} \): minimum time from the start of one row access to the start of the next.
  - \( t_{RC} = 110 \text{ ns} \) for a 4Mbit DRAM with a \( t_{RAC} \) of 60 ns

- \( t_{CAC} \): minimum time from CAS line falling to valid data output.
  - 15 ns for a 4Mbit DRAM with a \( t_{RAC} \) of 60 ns

- \( t_{PC} \): minimum time from the start of one column access to the start of the next.
  - 35 ns for a 4Mbit DRAM with a \( t_{RAC} \) of 60 ns

What does “Synchronous” RAM mean?

- Take basic RAMs (SRAM and DRAM) and add clock:
  - Gives SSRAM or SDRAM (Synchronous SRAM/DRAM)

- More complicated, on-chip controller
  - Operations synchronized to clock
    - So, give row address one cycle
    - Column address some number of cycles later (say 2)
    - Data comes out later (say 2 cycles later)

- Burst modes
  - Typical might be 1,2,4,8, or 256 length burst
  - Thus, only give RAS and CAS once for all of these accesses

- Multi-bank operation (on-chip interleaving)
  - Lets you overlap startup latency (5 cycles above) of two banks

- Careful of timing specs!
  - 10ns SDRAM may still require 50ns to get first data!
  - 50ns DRAM means first data out in 50ns
**Example: SDRAM timing for Lab6**

- Micron 128M-bit dram (using 2Megx16bitx4bank ver)
  - Row (12 bits), bank (2 bits), column (9 bits)

**DRAMs over Time**

<table>
<thead>
<tr>
<th>DRAM Generation</th>
<th>DRAM Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st Gen. Sample</td>
<td>‘84 ‘87 ‘90 ‘93 ‘96 ‘99</td>
</tr>
<tr>
<td>Memory Size</td>
<td>1 Mb 4 Mb 16 Mb 64 Mb 256 Mb 1 Gb</td>
</tr>
<tr>
<td>Die Size (mm²)</td>
<td>55 85 130 200 300 450</td>
</tr>
<tr>
<td>Memory Area</td>
<td>30 47 72 110 165 250</td>
</tr>
<tr>
<td>Memory Cell Area (µm²)</td>
<td>28.84 11.1 4.26 1.64 0.61 0.23</td>
</tr>
</tbody>
</table>

(from Kazuhiro Sakashita, Mitsubishi)

**DRAM History**

- DRAMs: capacity +60%/yr, cost -30%/yr
  - 2.5X cells/area, 1.5X die size in -3 years
- ‘97 DRAM fab line costs $1B to $2B
  - DRAM only: density, leakage v. speed
- Rely on increasing no. of computers & memory per computer (60% market)
  - SIMM or DIMM is replaceable unit => computers use any generation DRAM
- Commodity, second source industry => high volume, low profit, conservative
  - Little organization innovation in 20 years page mode, EDO, Synch DRAM
- Order of importance: 1) Cost/bit 1a) Capacity
  - RAMBUS: 10X BW, +30% cost => little impact

**DRAM Design Goals**

- Reduce cell size 2.5, increase die size 1.5
- Sell 10% of a single DRAM generation
  - 6.25 billion DRAMs sold in 1996
- 3 phases: engineering samples, first customer ship(FCS), mass production
  - Fastest to FCS, mass production wins share
- Die size, testing time, yield => profit
  - Yield >> 60%
  (redundant rows/columns to repair flaws)
Today’s Situation: DRAM

- **Commodity, second source industry**
  - high volume, low profit, conservative
    - Little organization innovation (vs. processors) in 20 years: page mode, EDO, Synch DRAM

- **DRAM industry at a crossroads:**
  - Fewer DRAMs per computer over time
    - Growth bits/chip DRAM: 50%-60%/yr
    - Nathan Myrvold M/S: mature software growth (33%/yr for NT) - growth MB/$ of DRAM (25%-30%/yr)
  - Starting to question buying larger DRAMs?

Today’s Situation: DRAM

- Intel: 30%/year since 1987; 1/3 income profit

**The Art of Memory System Design**

- **Workload or Benchmark programs**
- **Processor**
- **Memory**
- **Optimize the memory system organization**
  - to minimize the average memory access time for typical workloads

**A Summary on Sources of Cache Misses**

- **Compulsory** (cold start or process migration, first reference): first access to a block
  - “Cold” fact of life: not a whole lot you can do about it
  - Note: If you are going to run “billions” of instruction, Compulsory Misses are insignificant

- **Capacity**:
  - Cache cannot contain all blocks access by the program
  - Solution: increase cache size

- **Conflict** (collision):
  - Multiple memory locations mapped to the same cache location
  - Solution 1: increase cache size
  - Solution 2: increase associativity

- **Coherence** (Invalidation): other process (e.g., I/O) updates memory
**Example: 1 KB Direct Mapped Cache with 32 B Blocks**

- For a $2^N$ byte cache:
  - The uppermost $(32 - N)$ bits are always the Cache Tag
  - The lowest $M$ bits are the Byte Select (Block Size = $2^M$)
  - One cache miss, pull in complete “Cache Block” (or “Cache Line”)

```
+----------------+---+---+---+
|    Block addr  | 9 | 4 | 0 |
+----------------+---+---+---+
| 31: Cache Tag  |   |   |   |
| Example: 0x50  |   |   |   |
+----------------+---+---+---+
| 9: Cache Index | Ex: 0x01 | Ex: 0x00 |
+----------------+---+---+---+
| 4: Cache Data  |   |   |   |
| Byte 31 ** Byte 1 Byte 0 0 |   |   |   |
| Byte 63 ** Byte 33 Byte 32 1 |   |   |   |
| 3 | 2 | 1 | 0 |
| Byte 1023 ** Byte 992 31 |
```

Valid Bit Cache Tag

- Stored as part of the cache “state”
- 0x50

**Set Associative Cache**

- **N-way set associative**: $N$ entries for each Cache Index
  - $N$ direct mapped caches operate in parallel

**Example: Two-way set associative cache**

- Cache Index selects a “set” from the cache
- The two tags in the set are compared to the input in parallel
- Data is selected based on the tag result

```
Valid Cache Tag Cache Data Cache Data Cache Tag Valid
```

Cache Index

```
adr Tag Compare Mux 0 Sel 1 0 Sel 1
OR Hit Cache Block
```

**Disadvantage of Set Associative Cache**

- **N-way Set Associative Cache versus Direct Mapped Cache**:
  - $N$ comparators vs. 1
  - Extra MUX delay for the data
  - Data comes AFTER Hit/Miss decision and set selection

- In a direct mapped cache, Cache Block is available BEFORE Hit/Miss:
  - Possible to assume a hit and continue. Recover later if miss.

```
Valid Cache Tag Cache Data Cache Data Cache Tag Valid
```

Cache Index

```
adr Tag Compare Mux 0 Sel 1 0 Sel 1
OR Hit Cache Block
```

**Example: Fully Associative**

- **Fully Associative Cache**
  - Forget about the Cache Index
  - Compare the Cache Tags of all cache entries in parallel
  - Example: Block Size = 32 B blocks, we need $N$ 27-bit comparators

- **By definition**: Conflict Miss = 0 for a fully associative cache
Two Different Types of Locality:
- Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon.
- Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon.

SRAM is fast but expensive and not very dense:
- 6-Transistor cell (no static current) or 4-Transistor cell (static current)
- Does not need to be refreshed
- Good choice for providing the user FAST access time.
- Typically used for CACHE

DRAM is slow but cheap and dense:
- 1-Transistor cell (+ trench capacitor)
- Must be refreshed
- Good choice for presenting the user with a BIG memory system
- Both asynchronous and synchronous versions
- Limited signal requires “sense-amplifiers” to recover