CprE 381: Computer Organization and Assembly Level Programming

Caches

Henry Duwe
Electrical and Computer Engineering
Iowa State University

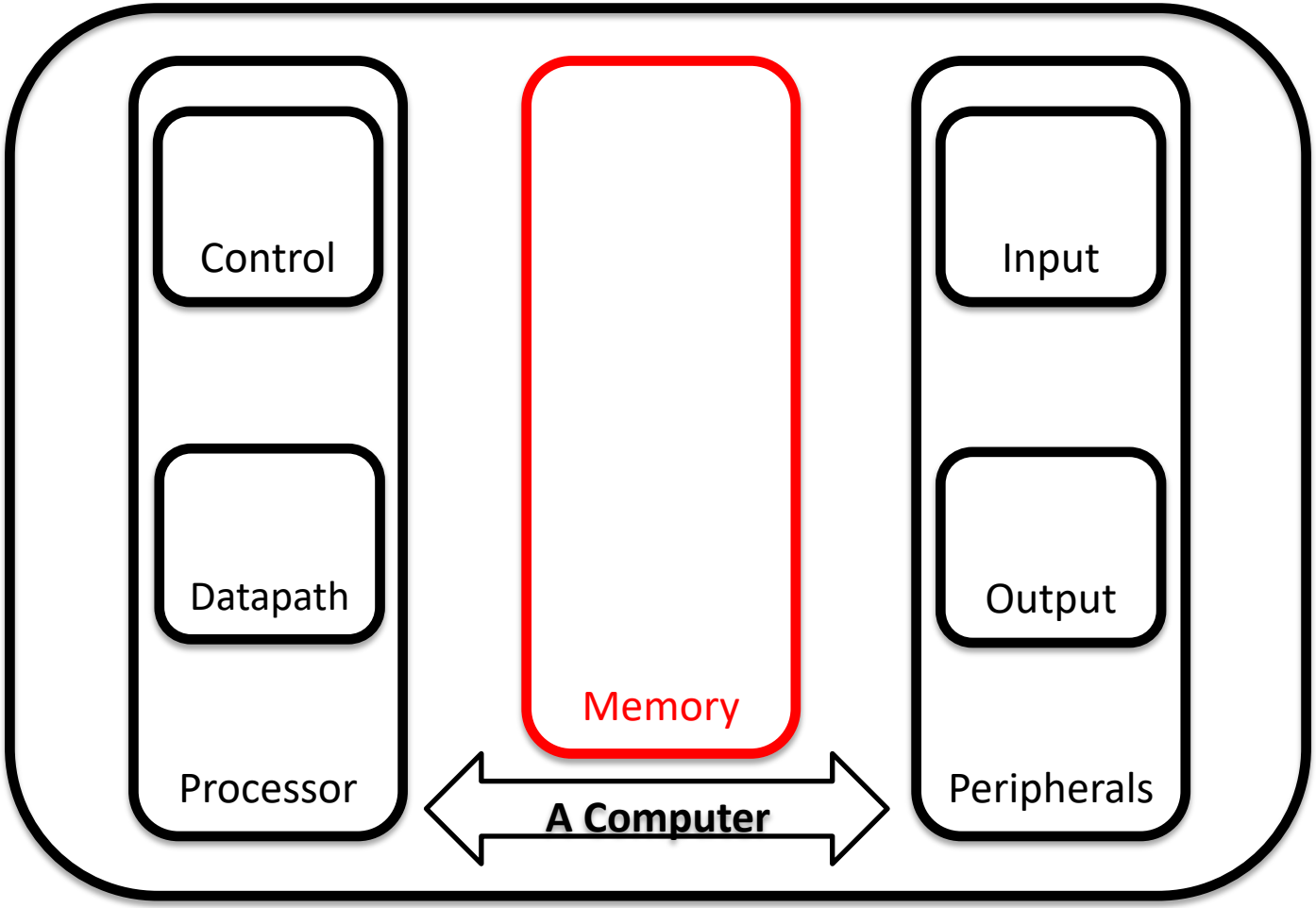
Administrative

- Exam 2
  - Average: 59 High: 97 Low: 13
  - HW8 Q1
    - Rework problem you lost the largest number of total points on
    - Use blank exam (with correction) posted on Canvas
    - Redo only the subparts you lost points on
    - Make perfect

Administrative

- Lec Quiz 11.1
  - In 2016 best flash density passed hard disk density
- Project Part 3a
  - Due next week
  - **WARNING**: Much easier that Part 3b!

Remember the System View!



Review: Small or Slow

- Unfortunately there is a tradeoff between speed, cost and capacity

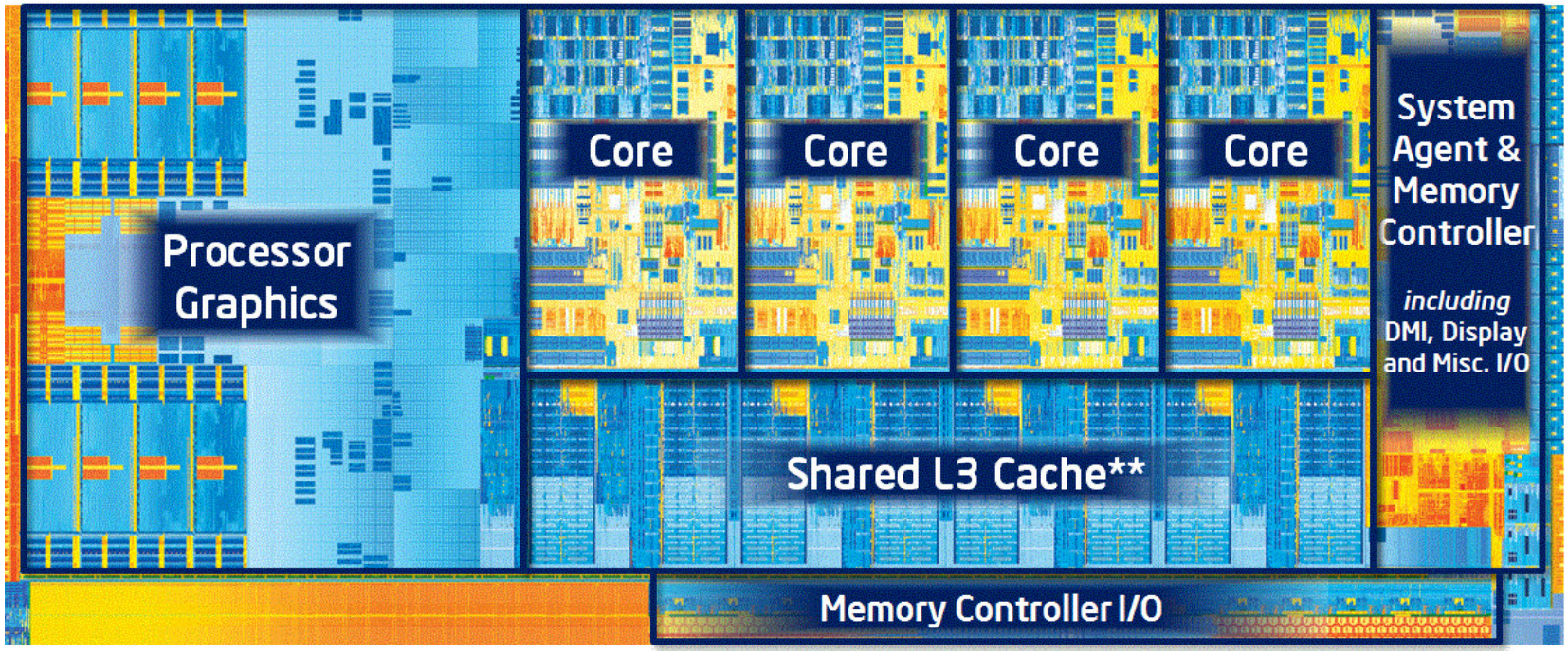
Storage	Speed	Cost	Capacity
Static RAM	Fastest	Expensive	Smallest
Dynamic RAM	Slow	Cheap	Large
Hard disks	Slowest	Cheapest	Largest

- Fast memory is too expensive for most people to buy a lot of
- But dynamic memory has a much longer delay than other functional units in a datapath. If every `lw` or `sw` accessed dynamic memory, we'd have to either increase the cycle time or stall frequently
- Here are ***rough*** estimates of some current storage parameters

Storage	Delay	Cost/MB	Capacity
Static RAM	1-10 cycles	~$1	128KB-128MB
Dynamic RAM	100-200 cycles	~$0.005	256MB-512GB
Hard disks	10,000,000 cycles	~$0.00005	512GB-10TB

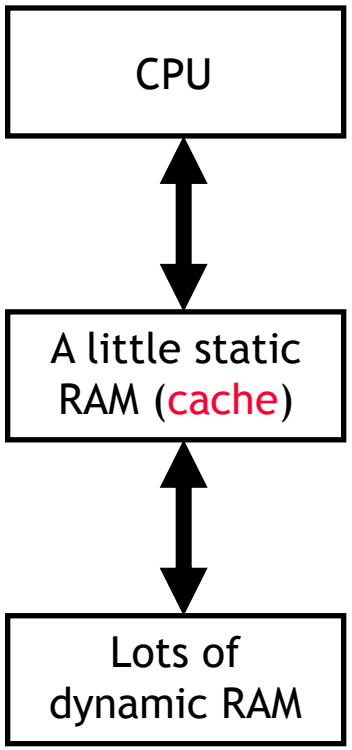
Cache Introduction

- What is a cache?
  - What problem does a cache solve?
- Why caches work? (answer: locality)
- How are caches organized?
  - Where do we put things--and--how do we find them?



Introducing Caches

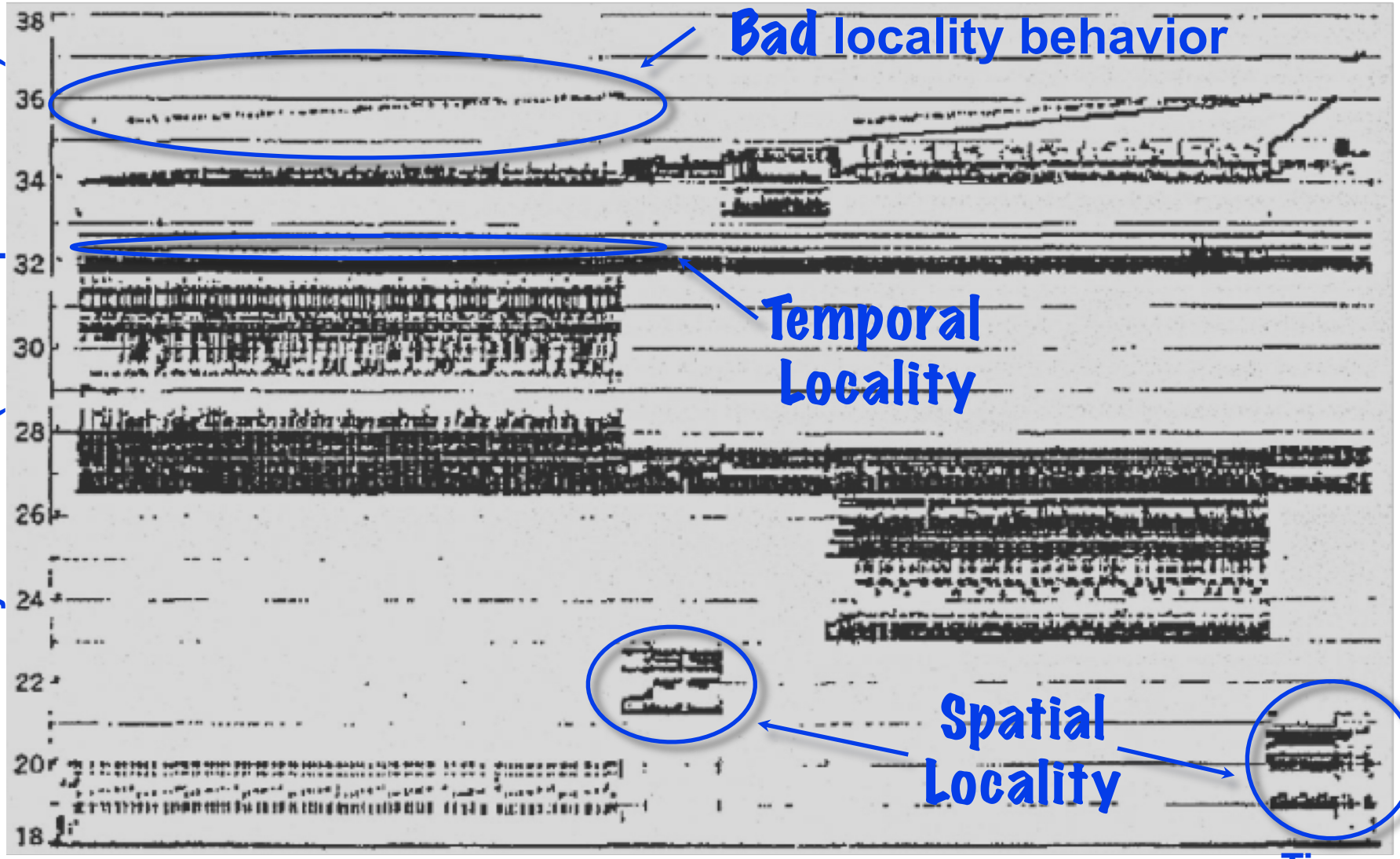
- Wouldn't it be nice if we could find a balance between fast and cheap memory?
- We do this by introducing a cache, which is a small amount of fast, expensive memory
  - The cache goes between the processor and the slower, dynamic main memory
  - It keeps a copy of the most frequently used data from the main memory
- Memory access speed increases overall, because we've made the common case faster
  - Reads and writes to the most frequently used addresses will be serviced by the cache
  - We only need to access the slower main memory for less frequently used data



The Principle of Locality

- It's usually difficult or impossible to figure out what data will be "most frequently accessed" before a program actually runs, which makes it hard to know what to store into the small, precious cache memory.
- But in practice, most programs exhibit *locality*, which the cache can take advantage of.
  - The principle of temporal locality says that if a program accesses one memory address, there is a good chance that it will access the same address again.
  - The principle of spatial locality says that if a program accesses one memory address, there is a good chance that it will also access other nearby addresses.

Programs with locality cache well ...



[Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192 (1971)]

Temporal Locality in Programs

- The principle of temporal locality says that if a program accesses one memory address, there is a good chance that it will access the same address again.
- Loops are excellent examples of temporal locality in programs.
  - The loop body will be executed many times.
  - The computer will need to access those same few locations of the instruction memory repeatedly.

- For example:

```
Loop: lw   $t0, 0($s1)
      add  $t0, $t0, $s2
      sw   $t0, 0($s1)
      addi $s1, $s1, -4
      bne  $s1, $0, Loop
```

  - Each instruction will be fetched over and over again, once on every loop iteration.

Temporal Locality in Data

- Programs often access the same variables over and over, especially within loops. Below, sum and i are repeatedly read and written.

```
sum = 0;
for (i = 0; i < MAX; i++)
    sum = sum + f(i);
```

- Commonly-accessed variables can sometimes be kept in registers, but this is not always possible.
  - There are a limited number of registers.
  - You can't take the address of a register (i.e., create a pointer to it)
  - There are situations where the data must be kept in memory, as is the case with shared or dynamically-allocated memory.

Spatial Locality in Programs

- The principle of spatial locality says that if a program accesses one memory address, there is a good chance that it will also access other nearby addresses.

```
sub $sp, $sp, 16
sw  $ra, 0($sp)
sw  $s0, 4($sp)
sw  $a0, 8($sp)
sw  $a1, 12($sp)
```

- Nearly every program exhibits spatial locality, because instructions are usually executed in sequence—if we execute an instruction at memory location *i*, then we will probably also execute the next instruction, at memory location *i+4*.

Spatial Locality in Data

- Programs often access data that is stored contiguously.
  - Arrays, like a in the code on the top, are stored in memory contiguously.
  - The individual fields of a record or object like employee are also kept contiguously in memory.
- Can data have both spatial and temporal locality?

```
sum = 0;
for (i = 0; i < MAX; i++)
    sum = sum + a[i];
```

```
employee.name = "Homer Simpson";
employee.boss = "Mr. Burns";
employee.age = 45;
```

Spatial Locality in Data

- Programs often access data that is stored contiguously.
  - Arrays, like a in the code

```
sum = 0;
for (i = 0; i < MAX; i++)
    sum = sum + a[i];
```

In-class Assessment!

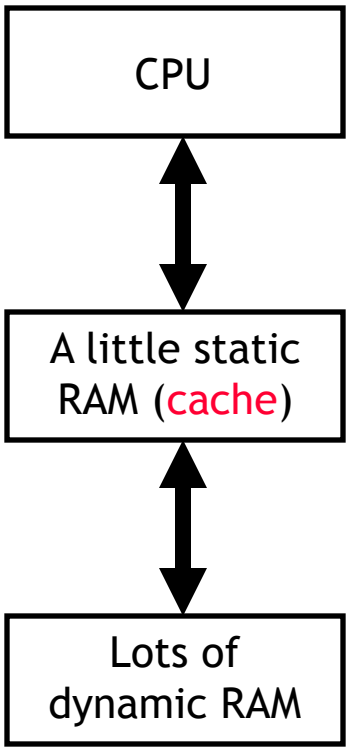
Access Code: NextDoor

Note: sharing access code to those outside of classroom or using access while outside of classroom is considered cheating

- Can data have both spatial and temporal locality?

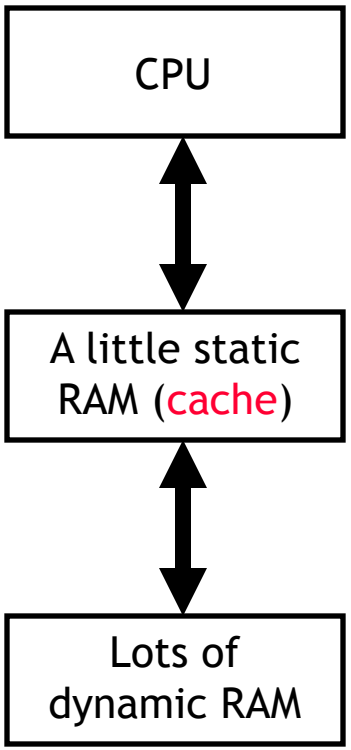
Taking Advantage of Temporal Locality

- The first time the processor reads from an address in main memory, a copy of that data is also stored in the cache.
  - The next time that same address is read, we can use the copy of the data in the cache *instead* of accessing the slower dynamic memory.
  - So the first read is a little slower than before since it goes through both main memory and the cache, but subsequent reads are much faster.
- This takes advantage of temporal locality—commonly accessed data is stored in the faster cache memory.



Taking Advantage of Spatial Locality

- When the CPU reads location *i* from main memory, a copy of that data is placed in the cache.
- But instead of just copying the contents of location *i*, we can copy *several* values into the cache at once, such as the four bytes from locations *i* through *i* + 3.
  - If the CPU later does need to read from locations *i* + 1, *i* + 2 or *i* + 3, it can access that data from the cache and not the slower main memory.
  - For example, instead of reading just one array element at a time, the cache might actually be loading four array elements at once.
- Again, the initial load incurs a performance penalty, but we're gambling on spatial locality and the chance that the CPU will need the extra data.



Other Kinds of Caches

- The general idea behind caches is used in many other situations.
- Networks are probably the best example.
  - Networks have relatively high “latency” and low “bandwidth,” so repeated data transfers are undesirable.
  - Browsers like Firefox and Chrome store your most recently accessed web pages on your hard disk.
  - Administrators can set up a network-wide cache, and companies like Akamai also provide caching services.
- A few other examples:
  - Many processors have a “translation lookaside buffer,” which is a cache dedicated to virtual memory support.
  - Operating systems may store frequently-accessed disk blocks, like directories, in main memory... and that data may then in turn be stored in the CPU cache!

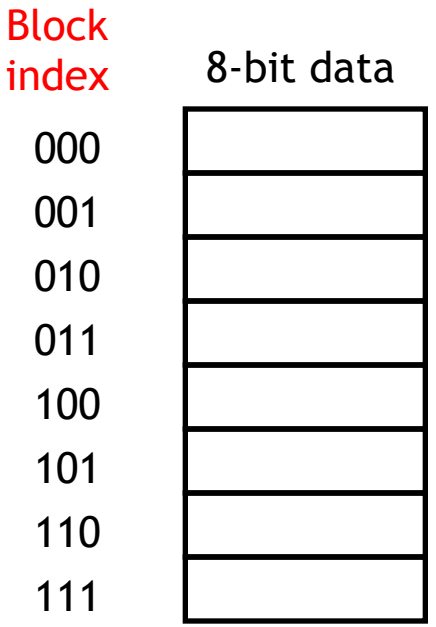
Definitions: Hits and Misses

- A cache hit occurs if the cache contains the data that we're looking for. Hits are good, because the cache can return the data much faster than main memory.
- A cache miss occurs if the cache does not contain the requested data. This is bad, since the CPU must then wait for the slower main memory.

- There are two basic measurements of cache performance.
  - The hit rate is the percentage of memory accesses that are handled by the cache.
  - The miss rate (1 – hit rate) is the percentage of accesses that must be handled by the slower main RAM.
- Typical caches have a hit rate of 95% or higher, so in fact most memory accesses will be handled by the cache and will be dramatically faster.

Preview: A Simple Cache Design

- Caches are divided into blocks, which may be of various sizes
  - The number of blocks in a cache is usually a power of 2
  - For now we'll say that each block contains one byte. This won't take advantage of spatial locality, but we'll do that next time
- Here is an example cache with eight blocks, each holding one byte



Summary

- Today we studied the basic ideas of caches
  - By taking advantage of spatial and temporal locality, we can use a small amount of fast but expensive memory to dramatically speed up the average memory access time
  - A cache is divided into many blocks, each of which contains a valid bit, a tag for matching memory addresses to cache contents, and the data itself
- In the next lectures, we'll look at some more advanced cache organizations and see how to measure the performance of memory systems

Acknowledgments

- These slides contain material developed and copyright by:
  - Joe Zambreno (Iowa State)
  - Akhilesh Tyagi (Iowa State)
  - David Patterson (UC Berkeley)
  - Mary Jane Irwin (Penn State)
  - Christos Kozyrakis (Stanford)
  - Onur Mutlu (Carnegie Mellon)
  - Krste Asanović (UC Berkeley)
  - Morgan Kaufmann