

Memory Hierarchy

CSCI-0310

Pascal Van Hentenryck



Memory Hierarchy

- Memory hierarchy
- Caches
- Buses

Memory Hierarchy

We would like to create an illusion that the processor has large amounts of very fast memory connected to it.

It needs to be an illusion because fast memory is expensive

<u>Technology</u>	<u>Access Time</u>	<u>\$ per Mbyte</u>
SRAM	8 - 35 ns	3.00
DRAM	90 - 120 ns	1.00
Magnetic Disk	10,000,000 ns	0.25

Principle of Locality

Memory references in programs tend to obey these principles:

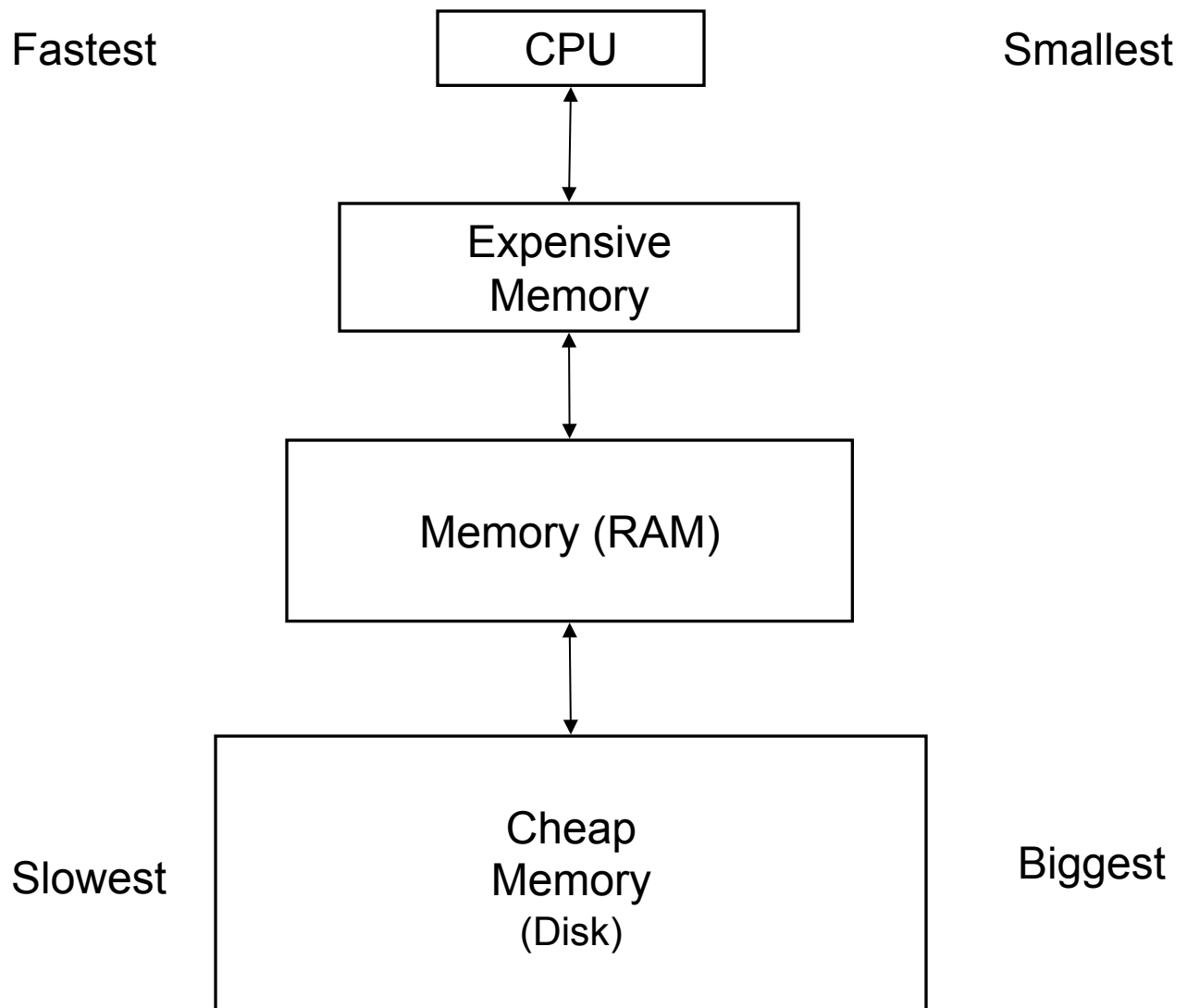
Temporal locality: If an item is referenced, it will tend to be referenced again soon.

Spatial locality: If an item is referenced, items whose addresses are close by will tend to be referenced soon.

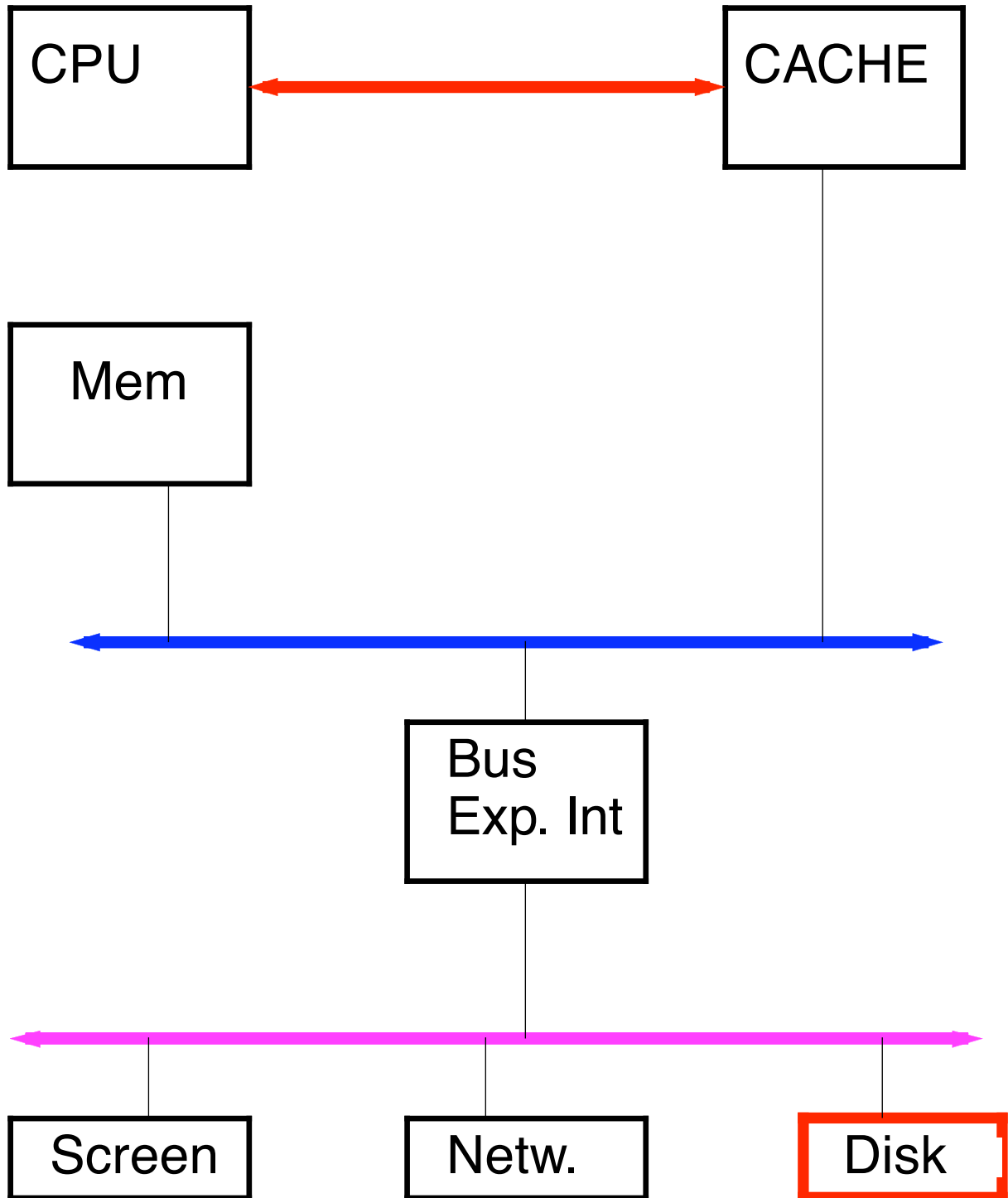
Analogy: your use of web pages.

Memory Hierarchy

We'll use a hierarchy of memory modules to maintain the illusion



Hierarchical Organization



Memory Hierarchy

Today's lecture

- Two levels: the RAM + the cache

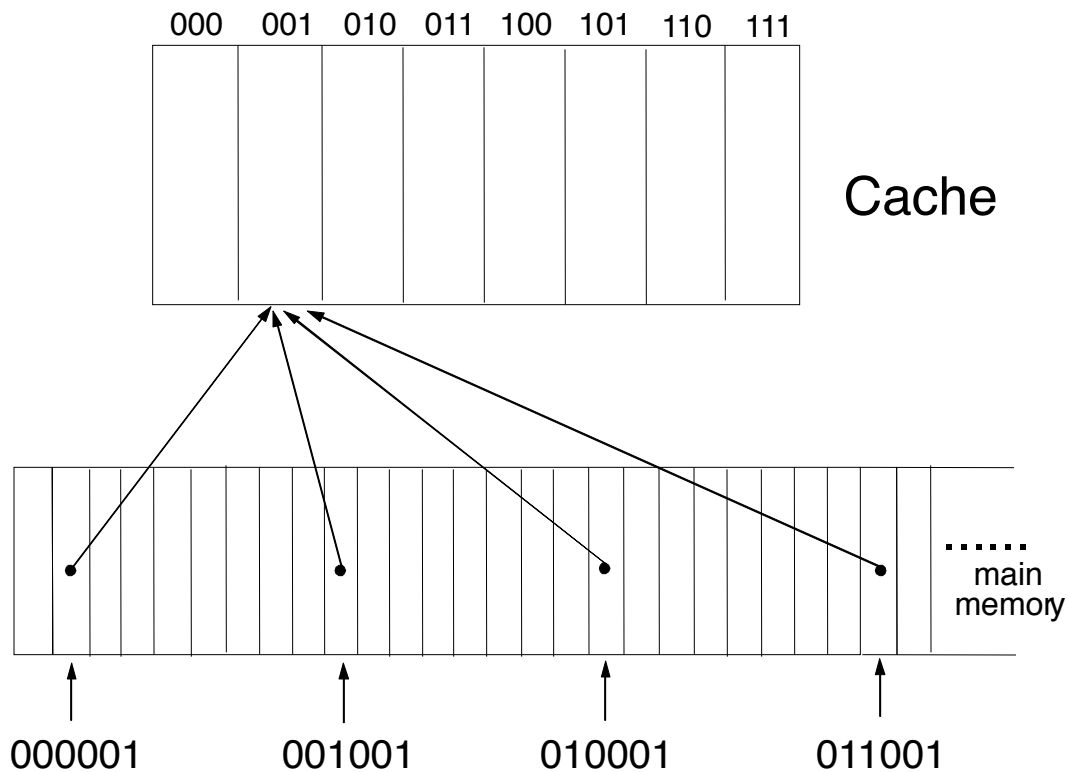
Basic Principles

- Store all the data in the lower level (RAM)
- Store some of the “good stuff” in the higher level (Cache)
- The *hit rate* is the proportion of memory references that are in the higher level
- The *miss penalty* is the time to replace a block in the upper level with the corresponding block from the lower level
- If we can make the hit rate high, most memory accesses will be fast

Cache

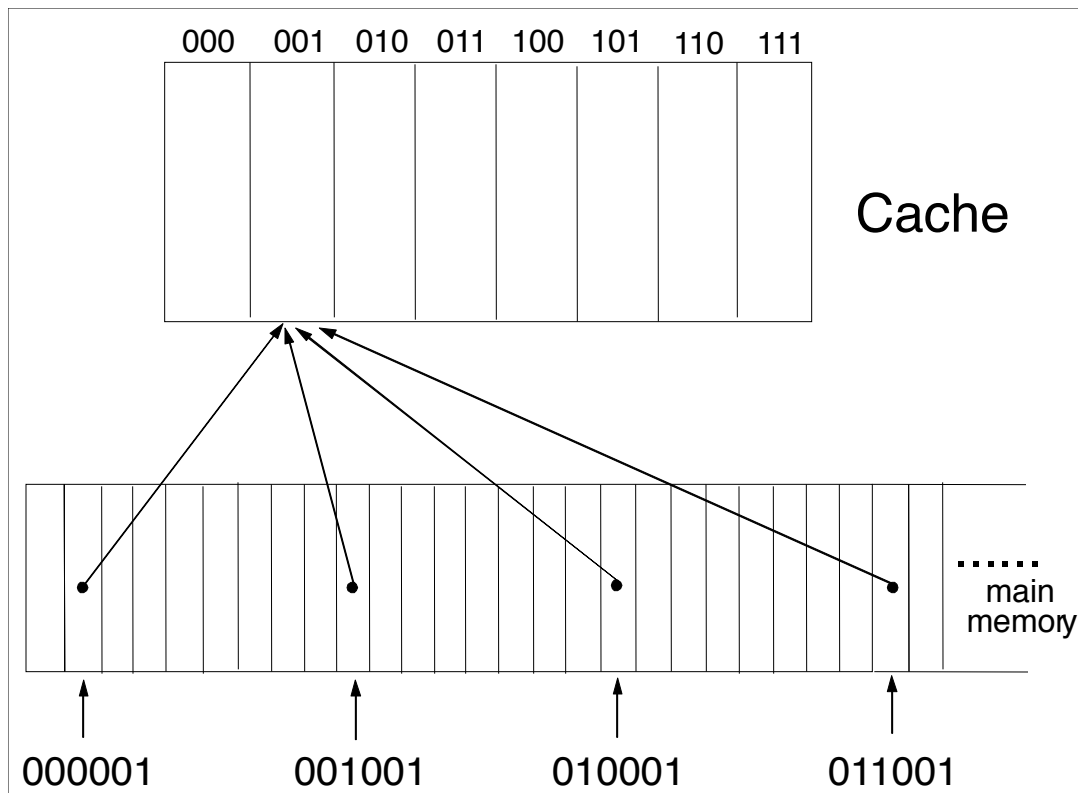
A small, fast memory where we keep relevant data is called a *cache*

- Simplest kind is a *direct-mapped* cache
- Each memory location is mapped to a cache location according to its **low order bits**



Cache

Why not indexing on the higher-order bits?



Cache

Each cache block contains

- a bit saying whether the data is valid
- the high-order bits of the address of the data that is currently there
- the data

Index	V	Tag	Data
000			
001			
010			
011			
100			
101			
110			
111			

Step by Step

Given an address to read from:

- use the low-order bits of the address to index into the cache
- if that cache line is invalid, we *miss*
- if the high-order bits of the address match the tag in the selected cache line, we *hit*
- if they don't match, we *miss*

If we hit (fast!)

- return the data from the selected cache line

If we miss (slow...)

- set the valid bit in this cache line
- set the tag bits to be the high-order bits of the address
- copy the data at this address from main memory into this cache line
- return the data

Cache

000001

Index	V	Tag	Data
000			
001	1	000	
010			
011			
100			
101			
110			
111			

001010

Index	V	Tag	Data
000			
001	1	000	
010	1	001	
011			
100			
101			
110			
111			

Cache

000001 HIT!

Index	V	Tag	Data
000			
001	1	000	
010	1	001	
011			
100			
101			
110			
111			

001001 MISS!

Index	V	Tag	Data
000			
001	1	001	
010	1	001	
011			
100			
101			
110			
111			

Cache

Index	V	Tag	Data
000			
001	1	001	
010	1	001	
011			
100			
101			
110			
111			

000001

Writing

The simplest scheme is *write-through*

Whether we hit or miss

- update the data in the cache
- update the data in the memory

This ensures that the cache is always *consistent* with the main memory

We can use a *write buffer* to decrease waiting time for the processor

Spatial Locality

In the previous example, each cache line held a word of data. What if we store more data per cache line?

- we exploit spatial locality for reads
- we have to do write misses differently:
when we write one word of block x
where block y used to be, we have to
read in the rest of block x

How big should blocks be?

More Caches

Direct-Mapped: every low-level block can be put in exactly one high-level location

Fully Associative: any low-level block can be put in any high-level location

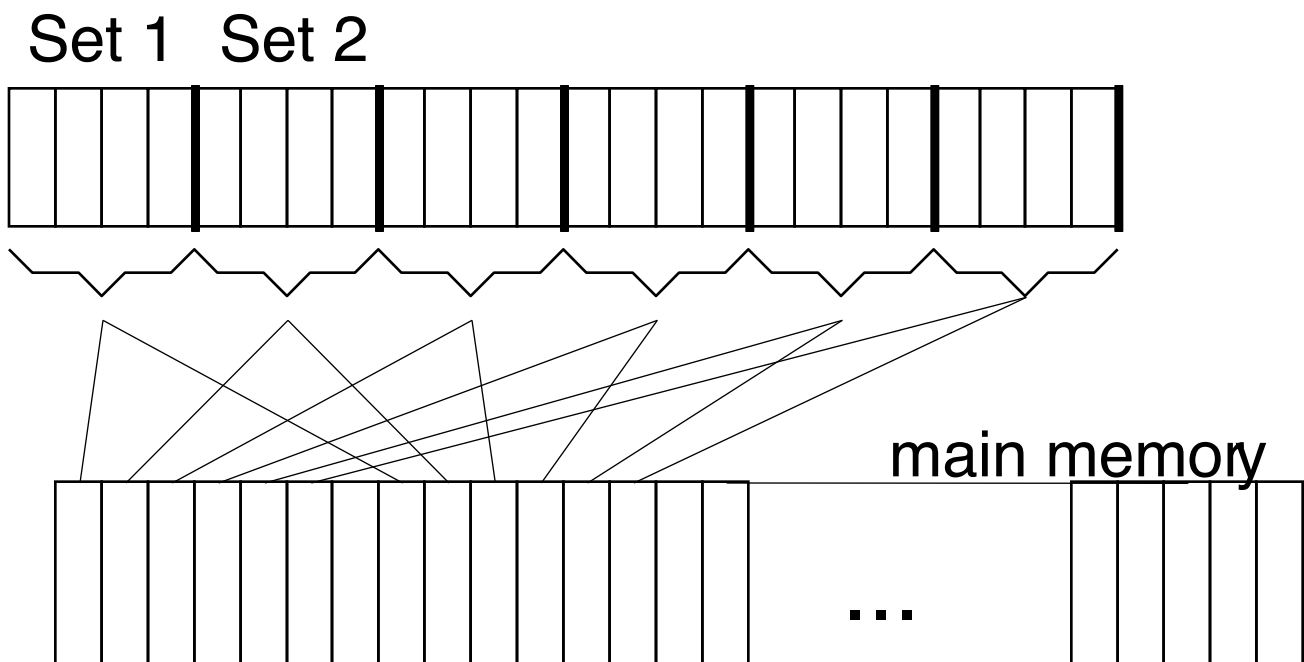
Set Associative: every low-level block can be put in one of a small number of high-level locations

Most caches are

- direct-mapped or set-associative

Set-Associative Caches

4-way set associative cache



Every low-level location is mapped directly to a set of high-level locations; within the high-level set, the mapping is associative.

How is a Block Found?

To look up an address:

- Determine which set it maps to, using low-order bits
- Match high-order bits against elements of the set

For a given set

Tag Bits	Data

- If the high-order bits match one of the tags, you hit! Return associated data
- On a miss, delete an element from this set and write in new data

Bus Design

A variety of issues

- structure
- arbitration
- data transfer

Bus Design

Structure

- 50 to 100 lines

Data lines

- Data to be read or to be written

Address lines

- Where to write or read the data

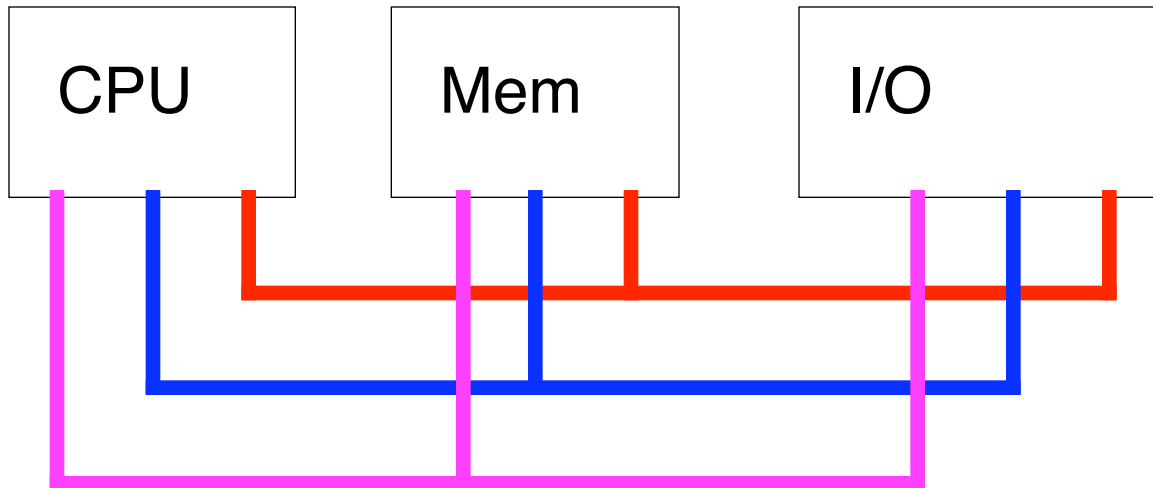
Control lines

- What to do with the lines
- MemoryRead, MemoryWrite, IOreadBus
- Bus request, bus granted

Dedicated versus Multiplexed lines

- Same lines used for data and addresses
- Dedicated lines

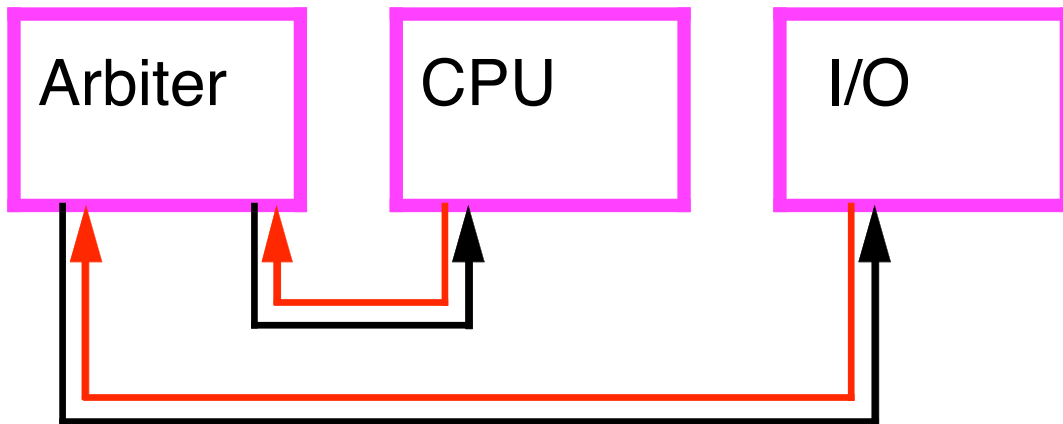
Bus Design



Bus Arbitration

- Different devices may want to use the bus at the same time
- I/O may write some memory while CPU may want to read/write some memory as well
- Need a way to coordinate them

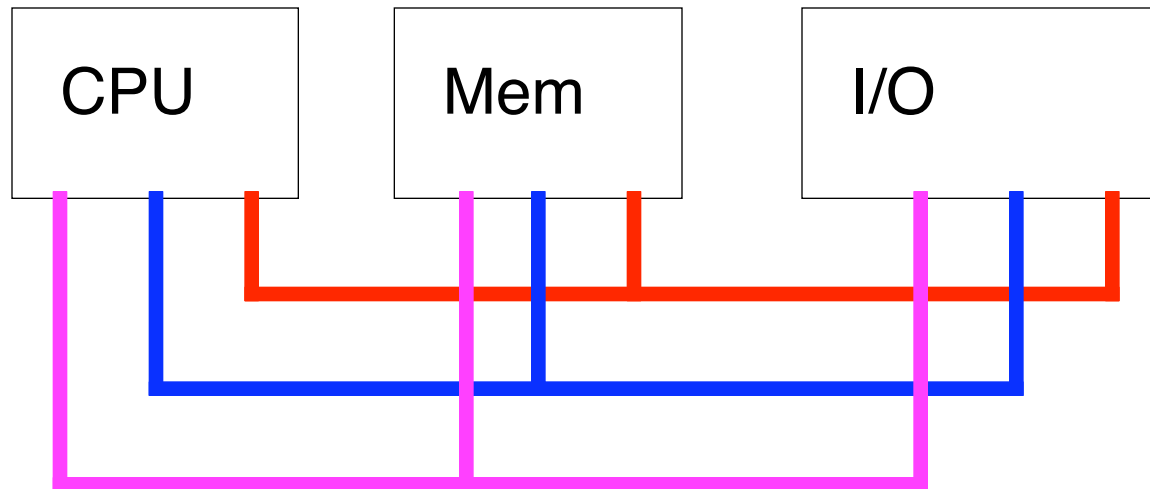
Central Arbitration



Main Problem of Buses

- Contention
- Everybody wants to use the bus

Distributed Arbitration

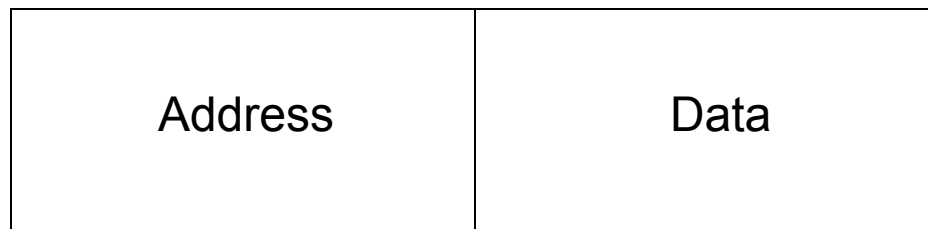


The devices talk to each other to elect a master

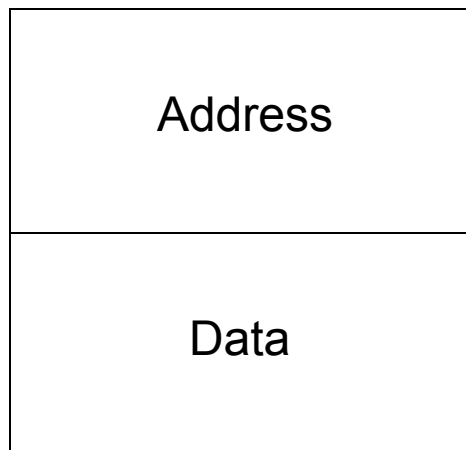
The master initiates the data transfer

Data Transfer

Write: multiplexed

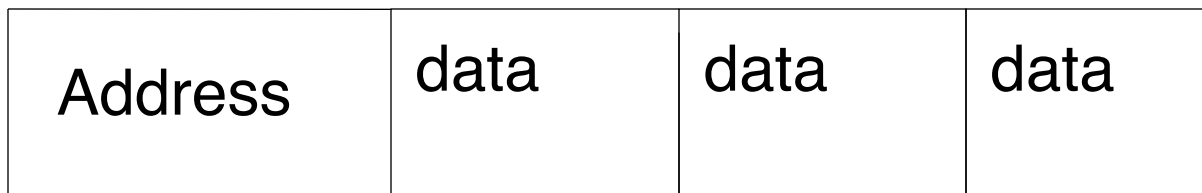


Write: dedicated



Data Transfer

Block write



Read-Modify-Write



PCI Bus

Structure (Intel, patents are public domain)

- 32- or 64 bit bus
- multiplexed data and address lines
- centralized arbitration
- interface control lines
- ...

Command (by the master at address time)

- I/O read
- I/O write
- Ram read / Ram multiple read
- Ram write / Ram multiple write
-

FutureBus+

Structure (IEEE Standard)

- address and data lines (many)

Arbitration

- decentralized or centralized

Data transfer

- many ...

FutureBus+

Basics

- Each module has a geographical address (hardwired)
- This makes sure that the system is open without having to reconfigure it.

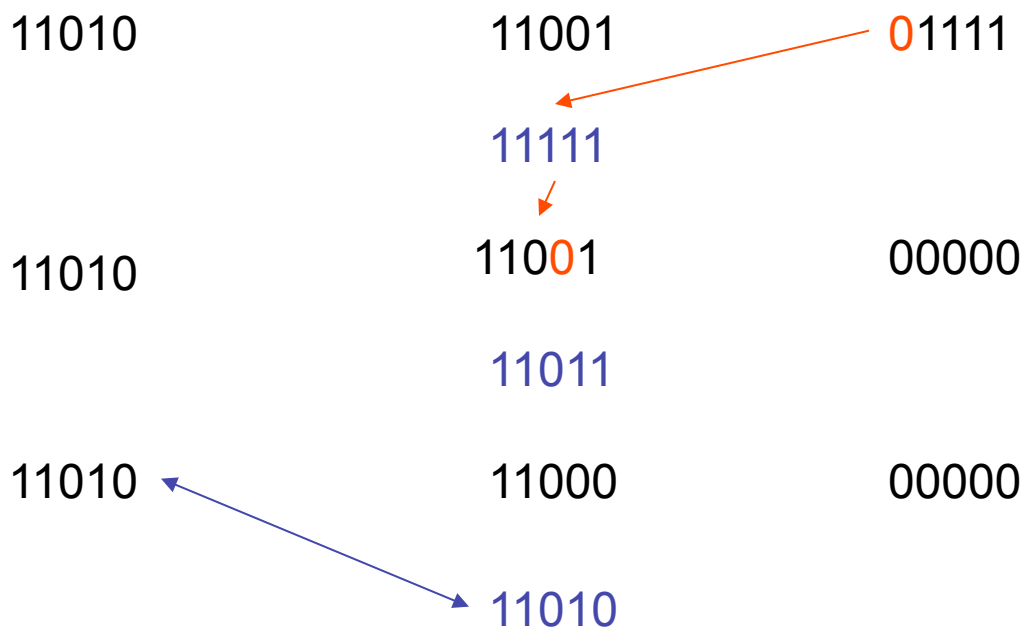
Arbitration Process

- Every module submit a competition number
- The module with the highest competition number wins

Competition

Basic ideas

- The bus contains the or of the competition numbers
- If a device notices that it has a zero while the bus has a one, it removes its lower bits
- At the end, the largest number will win



Competition Numbers

Structure

- Most significant: 8 bit of priorities
- A round robin bit
- Least significant: geographic location

Fairness

- the RR bit is set when tenure is given to a module at the same priority level but with a higher geographical address
- the RR bit is reset when tenure is given to a module at the same priority level with a lower geographical address
- This ensures round robin under heavy load