Announcements

- Final exam on Thursday, Dec. 14th from 1:30pm to 4:30pm
  - email me if you need a conflict

- Midterm 3 syllabus:
  - Caches : lectures 15, 16, 17, 18
  - Virtual Memory : lecture 19
  - IO : lectures 21, 22
  - Sections 8 and 9

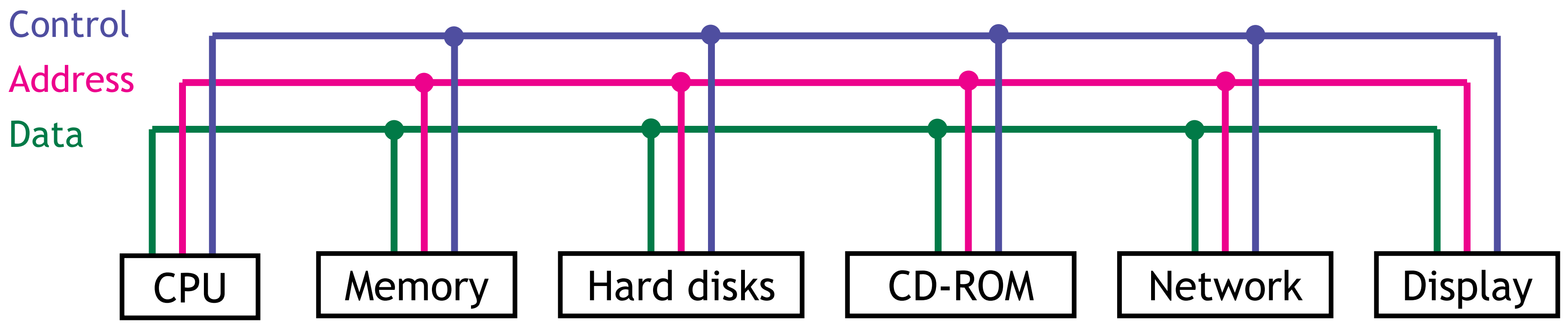
  - Lecture 20 (parallelism) and Section 10 (VTune) not on Exam 3

- Practice exam will be released middle of next week

What is the bus anyway?

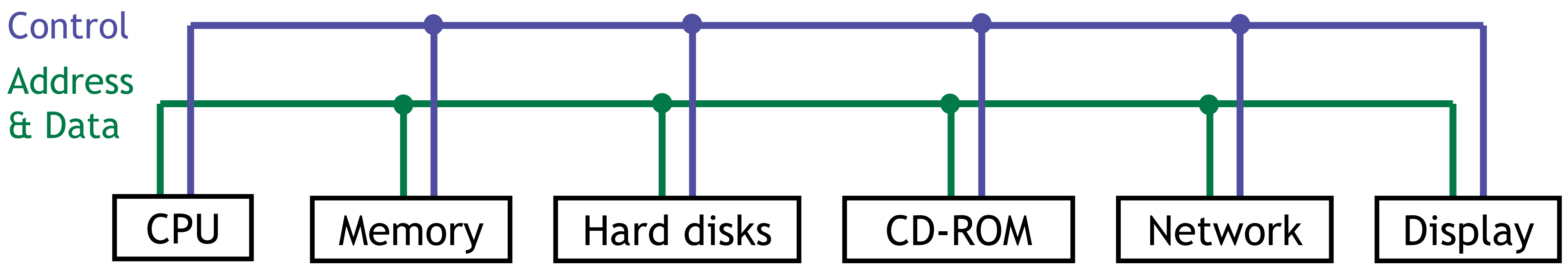
- A bus is just a bunch of wires which transmits three kinds of information
  - control signals (“read” or “write”)
  - the address on the device to read or write
  - the actual data being transferred

- Some buses have separate control, address and data lines
  - all takes one clock cycle



Multiplexed bus lines

- This is expensive
  - buses transfer 32 to 64 bits of data at a time
  - addresses are usually at least 32-bits long
- Another approach: multiplex some lines
  - same lines for address and the data (one after the other)
  - it takes *two* cycles to transmit both pieces of information

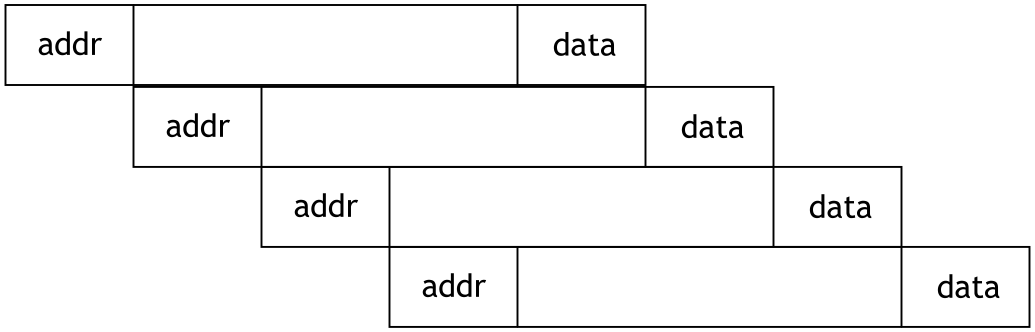


Example problem from HW 2, part C

- Interleaved memory, 4 banks (each has latency = 6 cycles)
- Cache block-size = 16-bytes
- CPU-memory bus has latency = 2 cycles and width = 4 bytes

Clean block

Multiplexed address/data bus



A fifth transfer would have stalled since addr and data share the bus

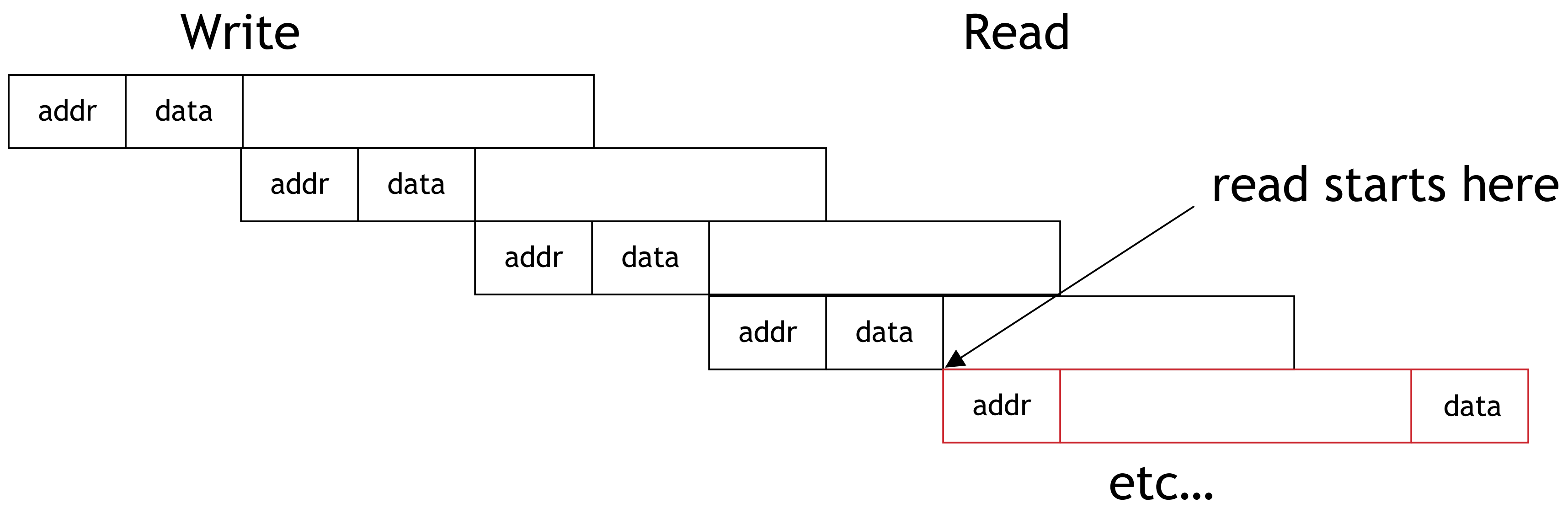
Separate address/data buses

Same picture, but a fifth transfer would not stall since addr and data use different buses

Example problem from HW 2, part C

- Interleaved memory, 4 banks (each has latency = 6 cycles)
- Cache block-size = 16-bytes
- CPU-memory bus has latency = 2 cycles and width = 4 bytes

Dirty block (multiplexed bus)



Frequencies

- CPUs actually operate at two frequencies:
  - The internal frequency is the clock rate inside the CPU, which is what we've been talking about so far
  - The external frequency is the speed of the processor bus, which limits how fast the CPU can transfer data
- The internal frequency is usually a multiple of the external bus speed
  - A 2.167 GHz Athlon XP sits on a 166 MHz bus (166 x 13)
  - A 2.66 GHz Pentium 4 might use a 133 MHz bus (133 x 20)

Synchronous and asynchronous buses

- A synchronous bus operates with a central clock signal
  - transactions can be handled easily with finite state machines
  - all devices on the bus run at the same speed (even if they can run faster)
  - limited number of devices (more devices means longer bus)

- An asynchronous bus does not rely on clock signals
  - transactions rely on complicated protocols
  - the bus can be longer, devices can operate at different speeds
  - external buses like USB and Firewire are asynchronous

External buses

- External buses support the frequent plugging and un-plugging of devices

- Universal Serial Bus (USB) and FireWire:
  - Plug-and-play standards devices configured by software, instead of flipping switches or setting jumpers
  - Hot plugging add/remove peripheral without turning machine off
  - The cable transmits power – fewer power cables
  - Serial links are used, so the cable and connectors are small

FireWire

The Serial/Parallel conundrum

- Generally, more wires means more bandwidth – quicker transfer times
  Recall the equation:
  Time = latency + transfer_size / bandwidth
  - Other things being equal, bigger bandwidth means less time
- But bigger bandwidth (parallel) also means bigger latency…
  Serial buses can be clocked at higher frequencies:
  - parallel interconnects have interference between signal wires
  - skew is also a problem: bits arrive at slightly different times
- Serial links are being increasingly considered for internal buses:
  - Serial ATA is a new standard for hard drive interconnects
  - PCI-Express (aka 3GI/O) is a PCI bus replacement that uses serial links

Error Correcting Codes

- Physical representations of bits can get corrupted
  - disks can get scratched
  - interference
  - cosmic rays!!
  - …

- Detect and correct errors by introducing redundancy
  - store data plus check-bits

- Example 1: along with every byte, store its parity

  Instead of storing just 1001 1101, store 1001 1101 1 ← parity bit

  If *any* bit is flipped (even the parity bit), error can be detected
- Example 2: store every bit multiple times

  Instead of storing just 1, store 1 1 1

  Use “majority” to detect and correct upto one error

Redundant bits

- Data range is 0x00, …, 0xFF (8 bits)
- Instead of storing 8-bit values, we store 8 + *r* bit values
- Can represent 2^{8+r} things, but only 2^8 “real” values
  - codewords (pre-defined list)
- All non-codewords indicate errors
  - correct these to “nearest” codeword
  - Hamming distance between two binary numbers *A* and *B* is the number of bits of *A* to flip until you get *B*

How much redundancy?

- We want to detect upto *d* errors and correct upto *c* errors

- Codewords must be "spaced out" with a minimum distance of:
  - at least $d + 1$ for "detect" condition

    If they were only *d* apart, one codeword could suffer *d* errors and be received as another codeword – impossible to detect error!
  - at least $2c + 1$ for "correct" condition

    If the closest codeword is at distance *c*, all other codewords must be at distance at least $c + 1$ in order to determine the *unique* corrected codeword. So, no two codewords are less than $2c + 1$ apart in Hamming distance.
- Take the *maximum* of these two for *d* bit detection *and c* bit correction

- Common scheme: SECDED (single error correction, double error detection).