CS 350

Visual Aids for Lectures

Part I

Eric G. Manning

A Theme

- Computer Organization (HARDWARE AND SOFTWARE) as a series of

- LEVELS

Tanenbaum's Levels *

- LEVEL 1 **-- microprogramming level**

  – *primitives available are*
    - microinstructions
    - e.g. GPR1 -> GPR2
    - START MEMORY_READ

  – *provided by*
    - hardware

Tanenbaum's Levels

- LEVEL 2 **the conventional machine level**
  - *primitives available are*
    - instructions, e.g.
    - LDA R1, 1000
    - ADA R1, 1001
    - STA R1, 1000
  - *provided by*
    - **microprograms** of Level 1, executed interpretively
  - *used to implement*
    - Operating System functions (and other things)

Tanenbaum's Levels

- LEVEL 3 **the Operating System Machine Level (OSML)**

  – *primitives provided are*

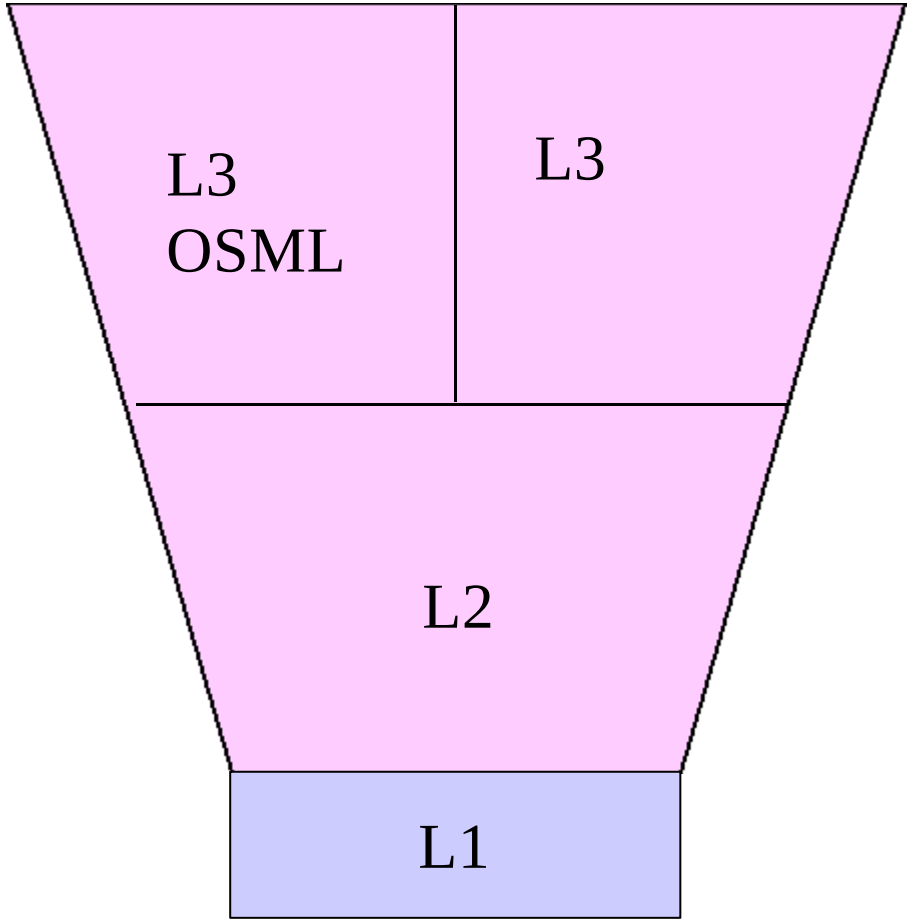
    - virtual machines **(enhanced hardware)** consisting of . . .

LEVEL 3 **the Operating System Machine Level (OSML)**

- Consisting of . . .
  - most L2 primitives (except HALT, etc)
  - OSML primitives such as
    - » virtual I/O to files (but not real I/O to discs etc)
    - » process creation & destruction
    - » process synchronization
    - » inter-process communication
  - plus facilities such as
    - » virtual memory
    - » virtual cpus
    - » shells

Tanenbaum's Levels

- LEVEL 3 **the Operating System Machine Level (OSML)**
  - *primitives are provided by*
    - programs composed of Level 2 primitives, executed interpretively
      - for OSML primitives
    - L2 primitives themselves
  - *used to implement*
    - Assembly Language functions



OS calls available. Many L2 primitives visible too.

Machine instructions are provided. NO L1 primitives visible.

Microinstructions provided by the hardware

Layers of Structure (so far)

Tanenbaum's Levels

- LEVEL 4 **the Assembly Language Level**
  - *primitives available are A/L statements corresponding to*
    - single L3 (L2) instructions
      - LOAD R1, GEORGE
    - single OSML (L3) instructions
      - OPEN /USR/FILENAME
    - sequences of L3 instructions (macros)

Tanenbaum's Levels

– *primitives are provided by*
  - a translator called an *assembler*

– *primitives are used by*
  - the next level up - as usual - the POL level

Tanenbaum's Levels

- LEVEL 5 **the Procedure-Oriented Language (POL) Level**

  – *primitives available are POL statements corresponding to*

    - sequences of L4 instructions , e.g.

A <-- B + C; generates

```
LOAD   R1, C
ADD            R1, B
STORE  R1, A
```

– *primitives are provided by*

- various translators called **Compilers or Interpreters**

Observations

- LEVEL J's primitives
  - are available to users of LEVEL J
  - are implemented in terms of LEVEL J-1s primitives

Observations

- Normally, primitives of L_{j-1} are invisible to users of L_j (exception : L3)
  - (machine language instructions available at OSML level)

- Primitives of L_j are collectively called the **Virtual Machine of Level j** , VM(j)

Who Implements What?

- L1: microprograms
  - usually written by the cpu manufacturer
  - sometimes users can write them too
    - (Nanodata QM-1, DEC pdp-11 Model 40)
  - sometimes L1 is implemented in hardware for speed (modern microprocessors, where Mp is as fast as microprogram memory)

Who Implements What?

- L2: assembler
  - usually cpu manufacturer

Who Implements What?

- L3: OSML routines

  usually manufacturer
  - (MacOS, IBM MVS / 370) or

- software house
  - (PC DOS) or

- consortium
  - (unix)

Who Implements What?

- L4: Compilers & Interpreters
- hardware manufacturer
  - (IBM PL/1 compilers)
- software house
  - Borland C++ compilers
- everybody but Microsoft
  - Java!

Some History

- L2: the machine instruction level or Instruction Set Processor (ISP) Level
  - von Neumann (1950), Babbage (1860), RISC researchers (1980s)

Some History

- L3: OSML
  - resident I/O routines 1950s
  - user interface languages
    - OS 360 Job Control Language 1965
    - unix Shell (1968)
  - virtual cpu (by time-multiplexing the real cpu)
    - IBM Control Program (1965)

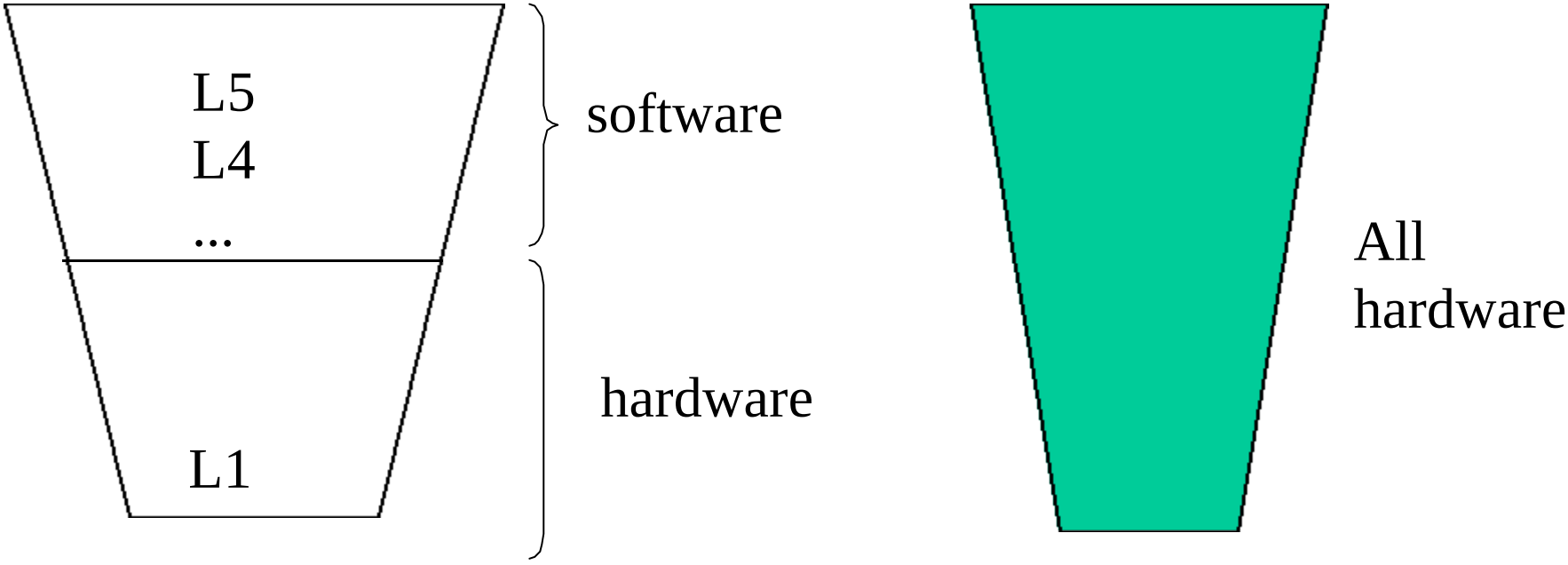
– virtual memory
  - ATLAS, CTSS (1963)
– processes
  - Multics (1965), unix (1968)

– timesharing the cpu but not full Virtual Machines:
  - Compatible Time Sharing System (MIT, 1962)

Some History

- L4: assemblers
  - various folks (1950s)

Hardware & Software Equivalence



They are equivalent

Hardware & Software Equivalence

- How to decide on the tradeoffs??
  - Do it in hardware for
    - higher performance
    - orderly, predictable design process
  - Do it in software for
    - lower replication cost
    - possibility of modifying the design

Things which moved to hardware implementation ...

- Integer Multiply / Divide [1960]
- Floating Multiply / Divide [1960]
- Loop control
  - [IBM 360 ISP - BXLE - 1965]
- character-string processing
  - [IBM 1401 -1960]
- relocation of code
  - [GE 635 - 1965]

Things which moved to hardware implementation ...

- Context switching
  - [DEC VAX single instruction - 1975]
- I/O operations
  - [IBM 360 channel - 1965]
  - [HIS 6050 Front End Processor - 1972]
  - [DEC pdp-11 Direct Memory Access - 1972]

Things which moved to software implementation ...

- Use of general-purpose microprocessors plus software to implement the functionality of
  - thermostats
  - digital filters
  - automobile engine fuel injection
  - video games
  - **watches**

Things which moved to software implementation ...

- Use of general-purpose microprocessors plus software to implement the functionality of
  - one-armed bandits
  - automatic transmission shift control
  - graphical user interfaces
  - *embedded systems*

Things which moved to software implementation ...

- The cpu instruction set (Crusoe chip)
  - reduces gate count hence power consumption
  - optimized for Unix execution
  - just provides microinstructions in hardware
    - a “micro-cpu”

Review : Processes

- Definition: an instance of a program, executing on a particular set of data.

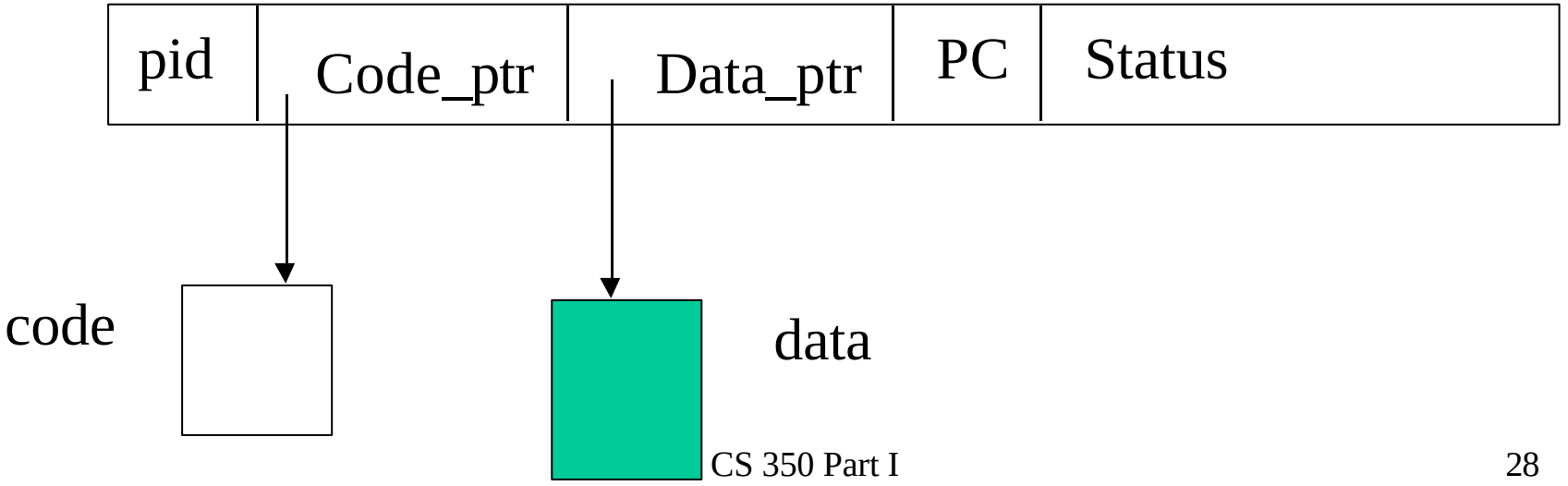
  – Abstractly: a 2-tuple

    – process_id = (program, data)

Review : Processes

- An implementation (one of many !)

- Process queue entry

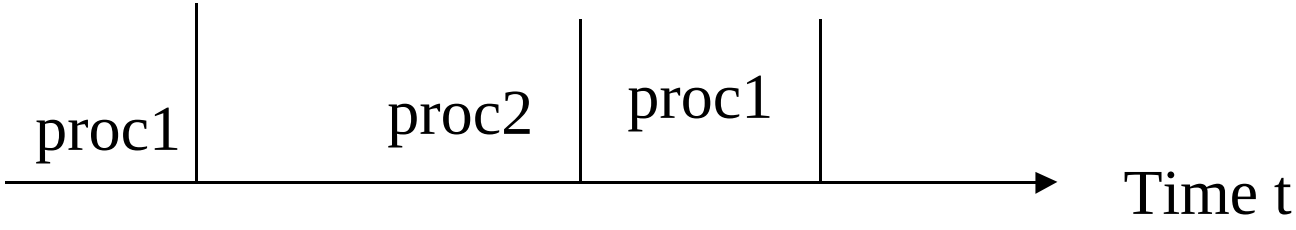


Review : Processes

- Why bother?
  - Imagine 500 CS students compiling C programs on CSC
  - without the process construct, there would be 50 identical copies of the C compiler resident in core
  - a necessary property of the compiler code?

- *Process state* includes
  - program
  - virtual PC
  - values of all variables
  - I/O status of all devices
  - ready / running /blocked flag

– everything needed to resume execution correctly after the cpu has been taken away and given back
– time multiplexing of the cpu



Summary - this lecture

- key ideas:
  - hierarchical layers of hardware & software
  - abstraction
  - process structure

Paterson -Hennessey Break

- You’re responsible for :
  - Chapter 1 (please read it)

PH Break

- Chapter 2
- key ideas:
  - performance metrics
  - how to calculate them
  - how to abuse them
  - relationships among them

Assignment 1

- See the webpage