EE108B

Digital Systems II

Christos Kozyrakis
Stanford University
christos@ee.stanford.edu

EE108B: Digital Systems II

- Part of the Digital Systems sequence of the new ugrad EE curriculum
  - Revision of EE182 and EE183
    - Follow on to EE108A (Digital Systems I)
    - Prerequisite for courses like EE109, EE282, …
  - Should better match your interests
    - So give us feedback on what works and what doesn't

- Class has TWO flavors
  - Take class with labs (for EE undergrads)
    - Prereq is EE108A
    - Lab enrollment is limited, so signup for lab section ASAP
  - Take lectures only (for CS undergrads, SCPD, grad students)
    - Do programming assignments instead of labs

What EE108b is About

- Many different views:
  - How to build programmable digital systems
  - Introduction to processor architecture
  - Understanding why your programs sometimes run much more slowly then you expect or don’t run at all

- Bottom line:
  - Digital systems are ubiquitous
  - Processors are one of the common idioms in digital design
    - Can’t avoid them these days
    - They are in your computers, TV, car, phones, door locks, …
  - It pays to understand how they work
    - To understand what they can and can’t do

Major Topics

- Hardware-software interface
  - Machine language and assembly language programming
- Compiler optimizations and performance
- Processor design
  - Pipelined processor design
- Memory hierarchy
- Virtual memory & operating systems support
- I/O devices

Syllabus

- Please see Handout #1
  - Contact class staff with any questions
- Includes tentative schedule
- Assignment due dates & quiz dates listed
  - Make sure you have time for everything before you sign up…
    - Especially if you are a SCPD student
  - If not, you should not take EE108b
    - Class offered again in Fall and Winter of 07-08 academic year

Course Information

- Instructor: Christos Kozyrakis
  - E-mail: christos@stanford.edu
  - Office: Gates 304
  - Office Hours: TBD or by appointment

- TAs
  - Yi Gu and Drew Hall
  - E-mail: ee108b-win0607-tas@lists.stanford.edu

- Course Support: Teresa Lynn
  - Office: Gates 310
  - Email: tlynn@csl.stanford.edu

Lectures & Discussion Sessions

- Lectures
  - Tuesday/Thursday 11am – 12.15pm, Skilling 191
    - We are trying to get a larger room (apologies)
    - Also live on SCPD channel E3, but it's best to come to class
- Discussion Session
  - Fridays TBD
  - Look at on-line schedule for specific sessions
- You should actively participate in lectures
  - Feel free to interrupt for Q&A, further thoughts on material, etc.
  - This is your way of setting the pace & quality of the class
  - Best way to get me to learn your name…

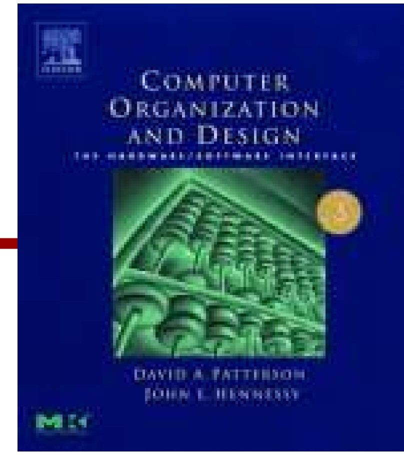
Prerequisites

- Prerequisites
  - EE undergrads: EE108A and CS106B
  - CS undergrads: E40 and CS106B

- The problem with separate prerequisites
  - EE students: know logic design but little about low level software
  - CS students: know software but no logic design
    - Many of you have taken CS107 and (perhaps) CS140

- I'll try to satisfy both sides but
  - Need your feedback to make the best of this situation
    - Questions, answers, insightful comments
  - Help me teach your classmates for topics you know well

Other Course Info



- Course Text
  - Computer Organization & Design, 3rd Edition
    - By D. Patterson & J. Hennessy
    - CD includes manuals, appendices, simulators, CAD tools, …
  - “Green card” summarizes MIPS ISA
- Website
  - http://eeclass.stanford.edu/ee108b
  - Check frequently
  - Will need to sign up with webpage once we enable it
    - Towards the end of 2nd week of lectures
- Handouts
  - Extras placed in cabinet on Gates 3rd floor
  - Print from class website

Grading

• Homework Sets	15%
• Programming Projects/Labs	25%
• Midterm	25%
• Final	35%

Workload

- 4-5 Homework Sets
  - Work in groups of up to 2 (no more!)
  - Due at 5 PM
  - Submit in class or to outside Gates 310
  - No late days
- 2 Programming Projects (for 3-unit option)
  - Complete individually
  - Use spim simulator on Linux & Windows machines
  - Electronic submission
  - No late days
- 4 Laboratory Projects (for 4-unit option)
  - Work in groups of up to 2 (no more!)
  - Use Xilinx FPGA boards in Packard 129
  - Demo and short electronic report submission
  - No late days

Quizzes

- No exams, just two quizzes
- Quiz 1: February 6th, 7pm – 9pm
  - Room TBD
  - No lecture on that day
- Final: March 15th, 11am – 12.30pm (in-class)
  - Room TBD
- The rules
  - Closed book, 1 page of notes, calculator
  - Local SCPD students expected to come to campus
  - Remote SCPD students must take quiz on same date

Lecture 1

Introduction to Programmable Digital Systems

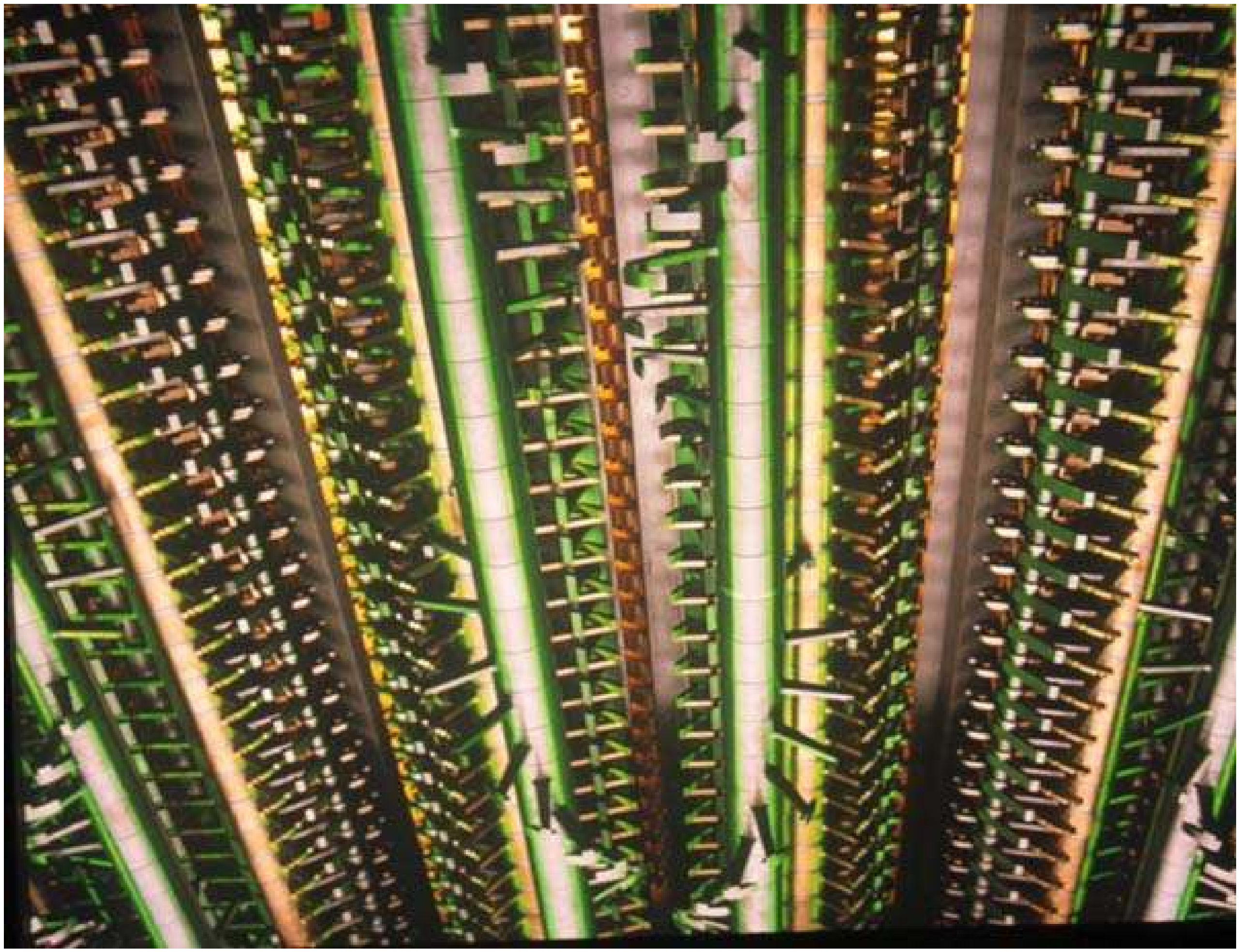
EE108b – Spring 2006
Stanford University
http://eeclass.stanford.edu/ee108b

Current State of the World

- Electronic systems dominate almost everything
  - And most of these systems use processors and memory

- Why?
  - Break this question into three questions
    - Why electronics
    - Why use ICs to build electronics
    - Why use processors in ICs
- Why use electronics
  - Electrons are easy to move / control
  - Easier than the current alternatives
- Result is that we move information / not real physical stuff
  - Think phone, email, fax, TV, WWW, etc.

Mechanical Alternative to Electronics



Picture of a version of the Babbage difference engine built by the Museum of Science UK

"The calculating section of Difference Engine No. 2, has 4,000 moving parts (excluding the printing mechanism) and weighs 2.6 tons. It is seven feet high, eleven feet long and eighteen inches in depth"

Electronics

- Building electronics:
  - Started with tubes, then miniature tubes
  - Transistors, then miniature transistors
  - Components were getting cheaper, more reliable but
    - There is a minimum cost of a component (storage, handling …)
    - Total system cost was proportional to complexity

- Integrated circuits changed that
  - Devices that integrate multiple transistors
  - Printed a circuit, like you print a picture,
    - Create components in parallel
    - Cost no longer depended on # of devices
  - What happens as resolution goes up?

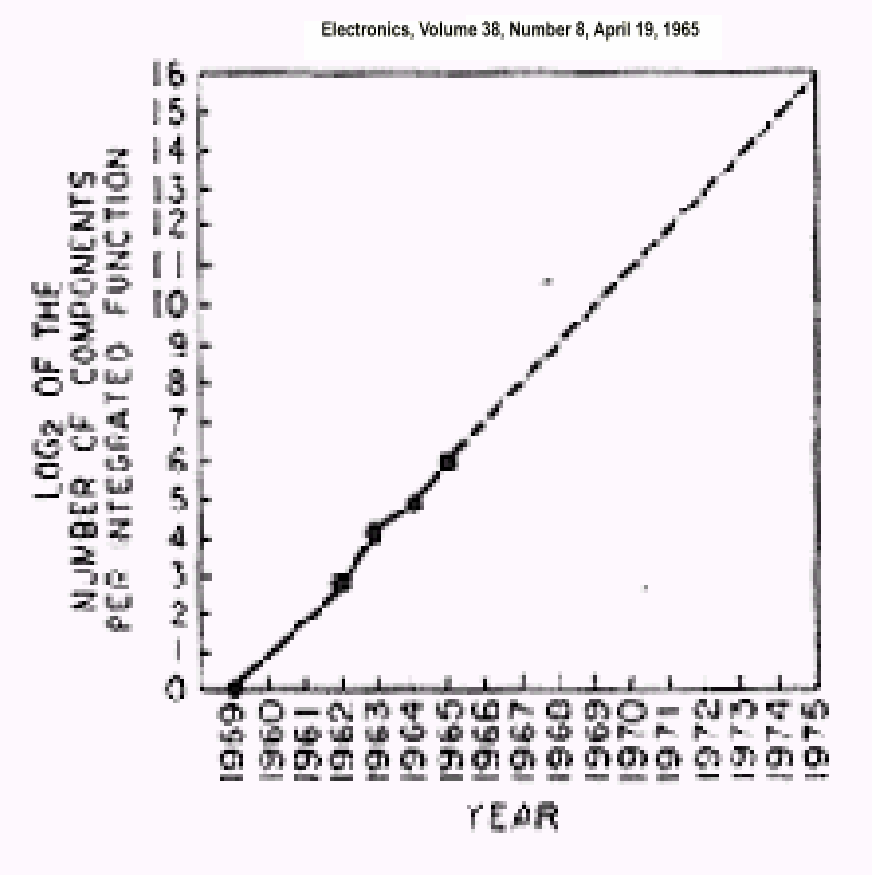
The Famous Moore's Law

- Devices get smaller
  - Get more devices on a chip
  - Devices get faster

- Initial graph from 1965 paper
  - Prediction: 2x per year
  - Not too many data points

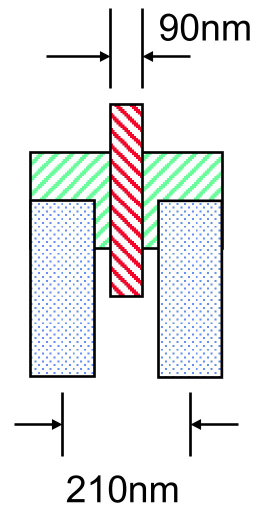
- Slowed down to 2x
  - Every 1.5 to 2 years?

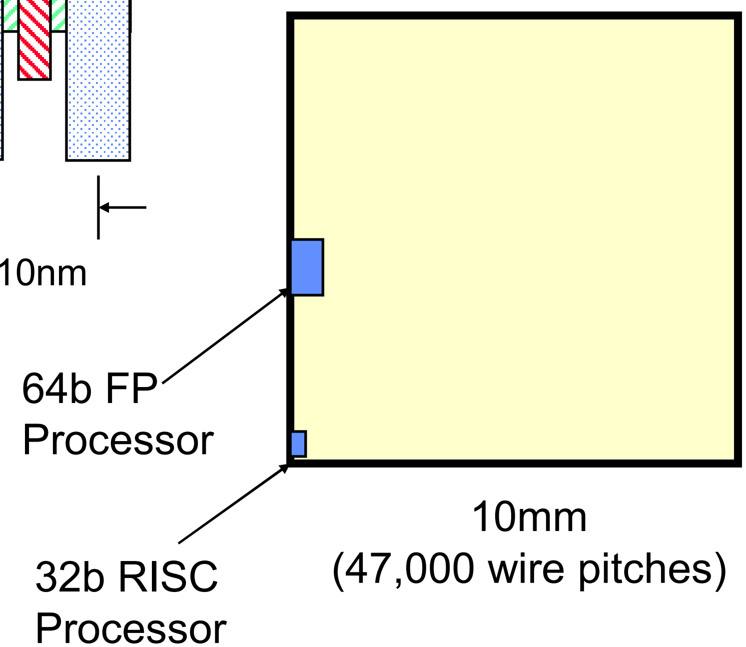
- Is Moore's Law really a Law?
- What does it say about performance?



Sense of Scale

- What fits on a chip today?
- Mainstream logic chip
  - 10mm on a side ($100mm^2$)
  - 90nm drawn gate length
  - 210nm wire pitch
  - 10 wires levels
- For comparison
  - 32b RISC integer processor
    - 1K x 2K wire grids
    - 1100 processors
  - SRAM
    - About 4 x 4 grids / bit
    - 138 M SRAM cells
  - DRAM
    - 1 x 2 grids / bit
    - 1.1 B cells

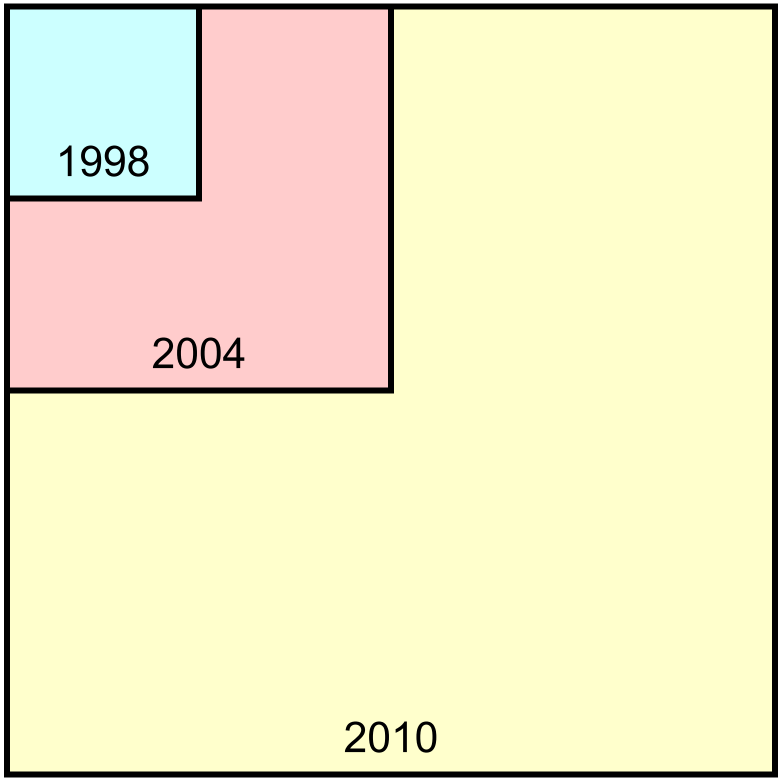




Technology Scaling

- Chip density doubles every 3 years
  - What can you do with this?

– More devices ⇒ harder to design



The Complexity Problem

- Complexity is the limiting factor in modern chip design
  - Two problems

1. How do you make use of all that space?
   - Uberappliance
     - Cellphone, PDA, iPod, mobile TV, video camera
   - Too many applications to cast all into hardware logic
   - Takes too long to finish the design
2. How do you make sure it works?
   - Verification problem

- Only way to survive complexity:
  - Hide complexity in “general-purpose” components
  - Reuse components

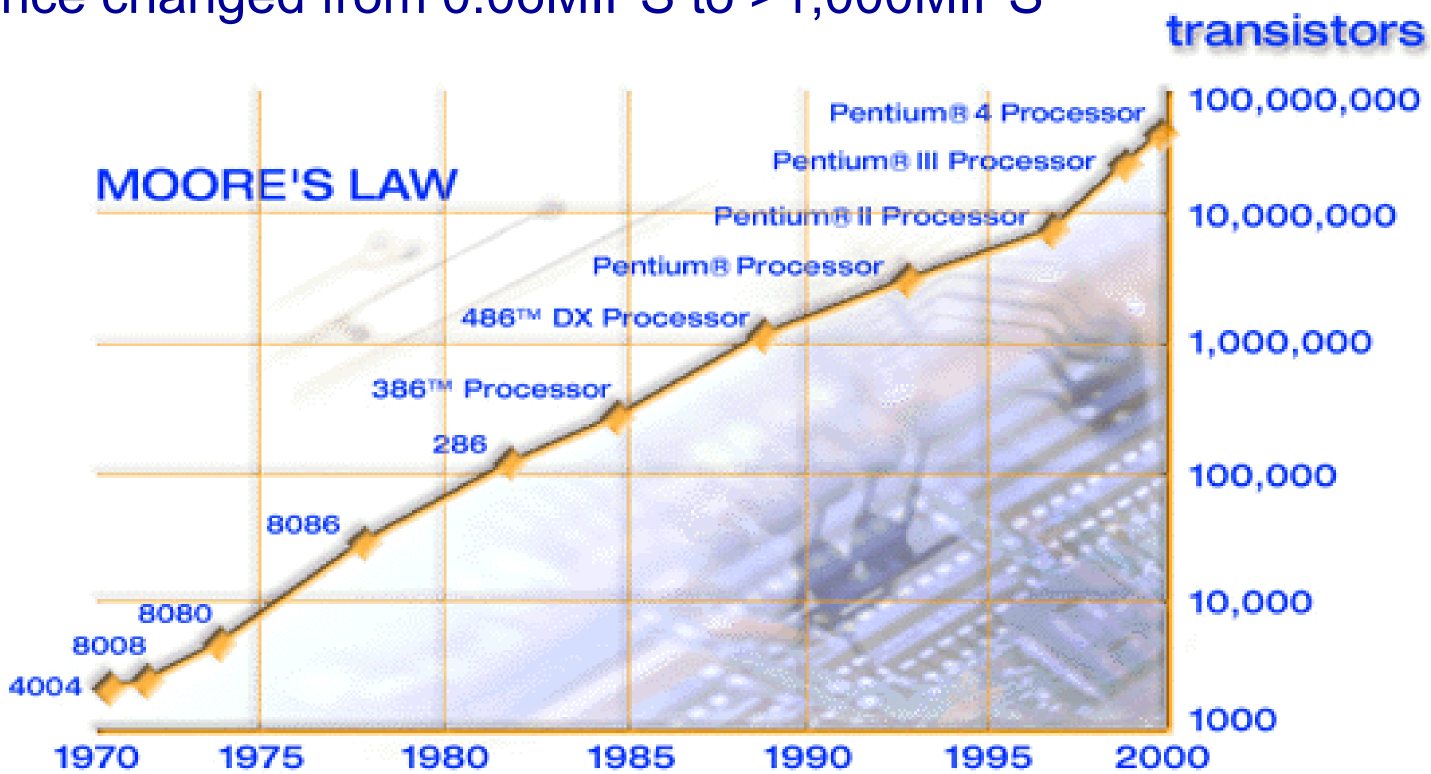
Programmable Components aka Processors

- An old approach to “solve” complexity problem
  - Build a generic device and customize with memory (program)
  - Best way to do this is with a general purpose processor

- Processor complexity grows with technology
  - But software model stays roughly the same
    - C, C++, Java, … run on Pentium 2, 3, and 4
    - True for sequential programs
  - This is getting much tougher to do
    - Recent hardware developments require software model changes
    - Multi-core processors

Microprocessor Complexity

- Model has hidden the scaling of technology
  - Efficiently transformed transistors to performance
  - 8080 – 3,500 transistors, and ran at 200kHz (1975)
  - Pentium4 – 42M transistors, runs at 3+GHz (2003)
  - Performance changed from 0.06MIPS to >1,000MIPS

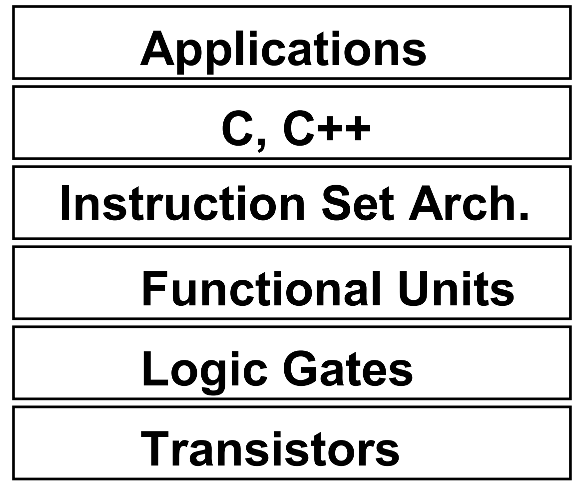


Key to Complexity: Nice Interfaces

- Use abstraction to hide complexity
  - Define an interface to allow people to use features without needing to understand all the implementation details

- Works for hardware and software

- Stable interfaces allows people to optimize below and above it



But I Never Want to Build Hardware

- Why should I care about how a computer works?
- And why should I have to learn about assembly code?
  - No one codes in assembly any more, right?
    - Unfortunately that is not correct
    - E.g. compilers, operating systems kernel
    - E.g. Embedded systems, video games

- It is still useful to look inside the box
  - Understand limitations of the programmers model
  - Understand strange performance issues
    - Efficiency and performance issues will become more important
  - Help you when things go wrong

Reality #1
Int's are not Integers, Float's are not Reals

- Examples
  - Is $x^2 \geq 0$?
    - Float's: Yes!
    - 32b Int's:
      - 40,000 * 40,000 --> 1,600,000,000
      - 50,000 * 50,000 --> ??
  - Is $(x + y) + z = x + (y + z)$?
    - Unsigned & Signed Int's: Yes!
    - Float's:
      - (1e20 + -1e20) + 3.14 --> 3.14
      - 1e20 + (-1e20 + 3.14) --> ??

Reality #2
You've got to know assembly

- Chances are, you'll never write program in assembly
  - Compilers are much better & more patient than you are

- Understanding assembly key to machine-level execution model
  - Behavior of programs in presence of bugs
    - High-level language model breaks down
  - Tuning program performance
    - Understanding sources of program inefficiency
  - Implementing system software
    - Compiler has machine code as target
    - Operating systems must manage process state

Reality #3
Memory Matters

- Memory is not unbounded
  - It must be allocated and managed
  - Many applications are memory dominated
- Memory referencing bugs especially pernicious
  - Effects are distant in both time and space
- Memory performance is not uniform
  - Cache and virtual memory can greatly affect program performance
  - Adapting program to characteristics of memory system can lead to major speed improvements
    - 10x to 100x in several cases

Class Goal

- Provide a better understanding of modern digital systems design
  - These systems almost always have a programmable processor
  - Processors are a good example of a complex system
    - Pipelining and caches

- Tie the hardware with the software
  - Most people use processors and don't build them
  - Interaction of HW and SW is fundamental to computer systems
  - Write better software

- Provide a foundation for other classes in systems
  - Networking, OS, Compilers, Embedded Systems, etc.
  - Understand capabilities of Compilers, OS

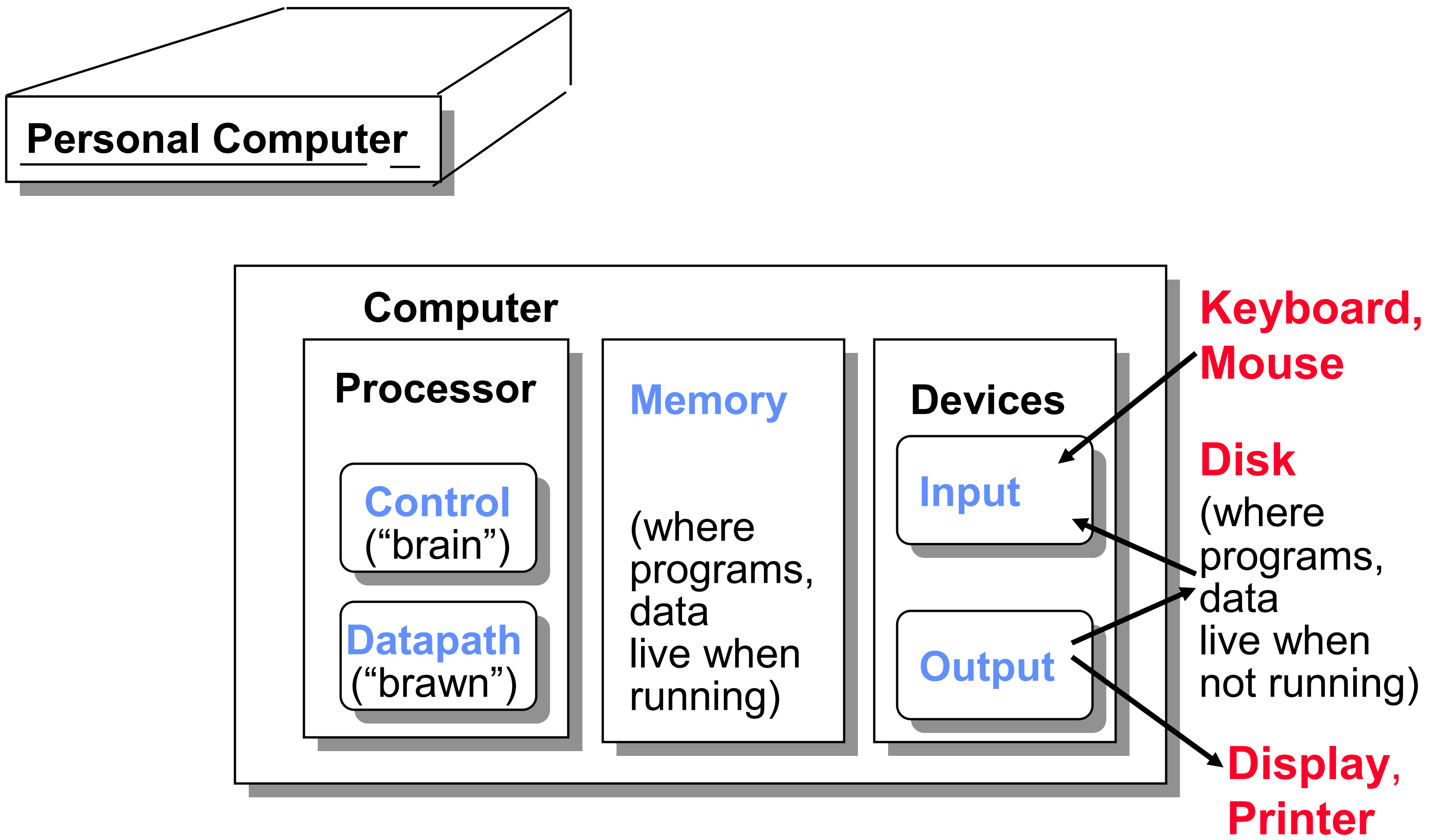
What is a Computer System?

- Depends (a little) on what type of computer system
- We probably mostly think about PC systems

What is a Computer System?

- Actually most computers look like this…

5 components of any Computer



What is in a Computer System?

- Each system is different, but generally have similar parts:

- Must have:
  - Processor, Memory
  - Interface to outside world (I/O)

- Generally have:
  - Cache memory
  - System bus
  - Memory controller
  - I/O bus

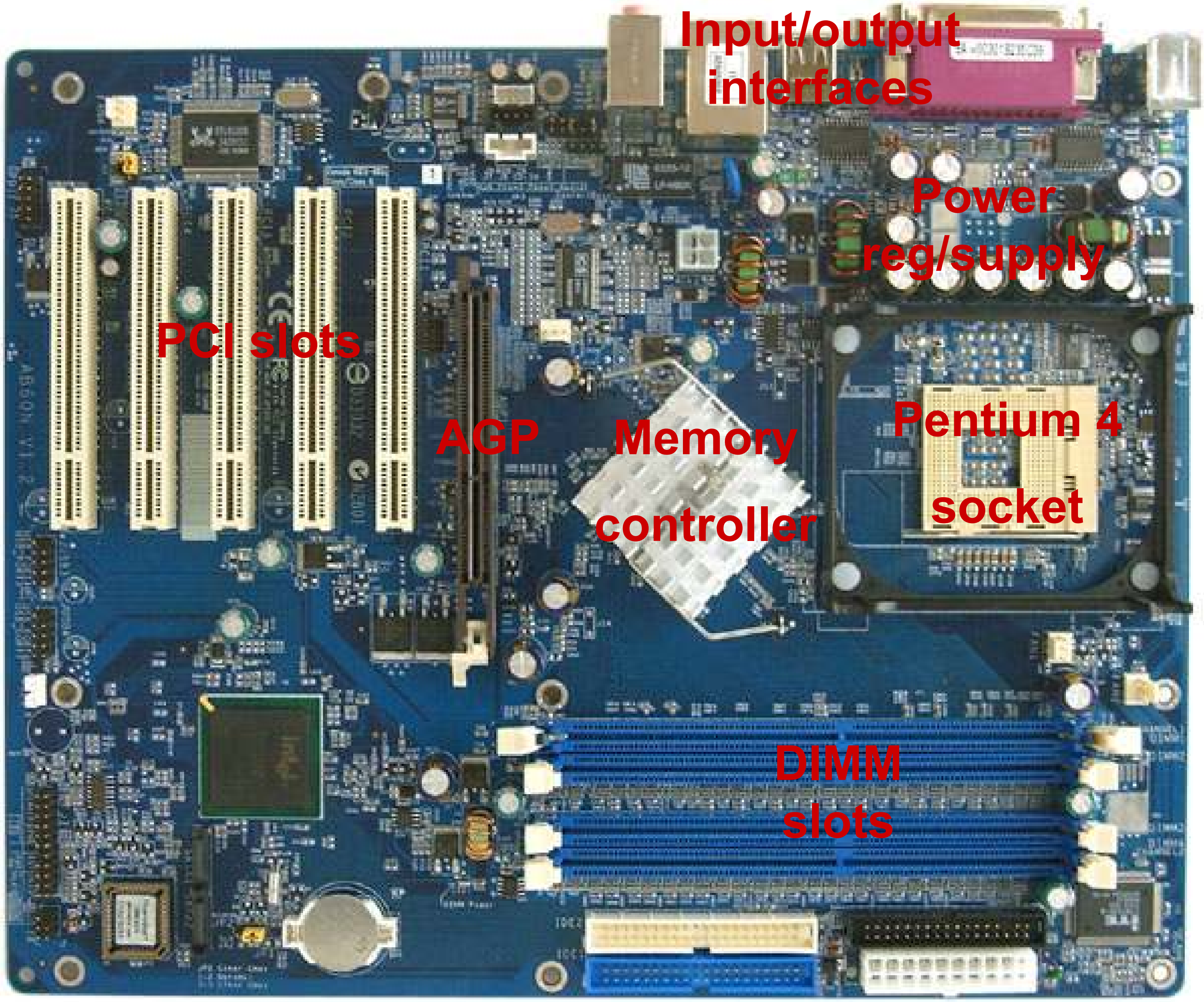
Example Processor Based Systems

- MIPS processor board
- PC Board
- Digital cell phone
- Game console

MIPS Processor Board

- R3000 CPU (120K transistors)
- R3010 FPU
- 32 KB Instruction cache
- 32 KB Data cache
- 256 KB secondary cache
- Memory controller chips

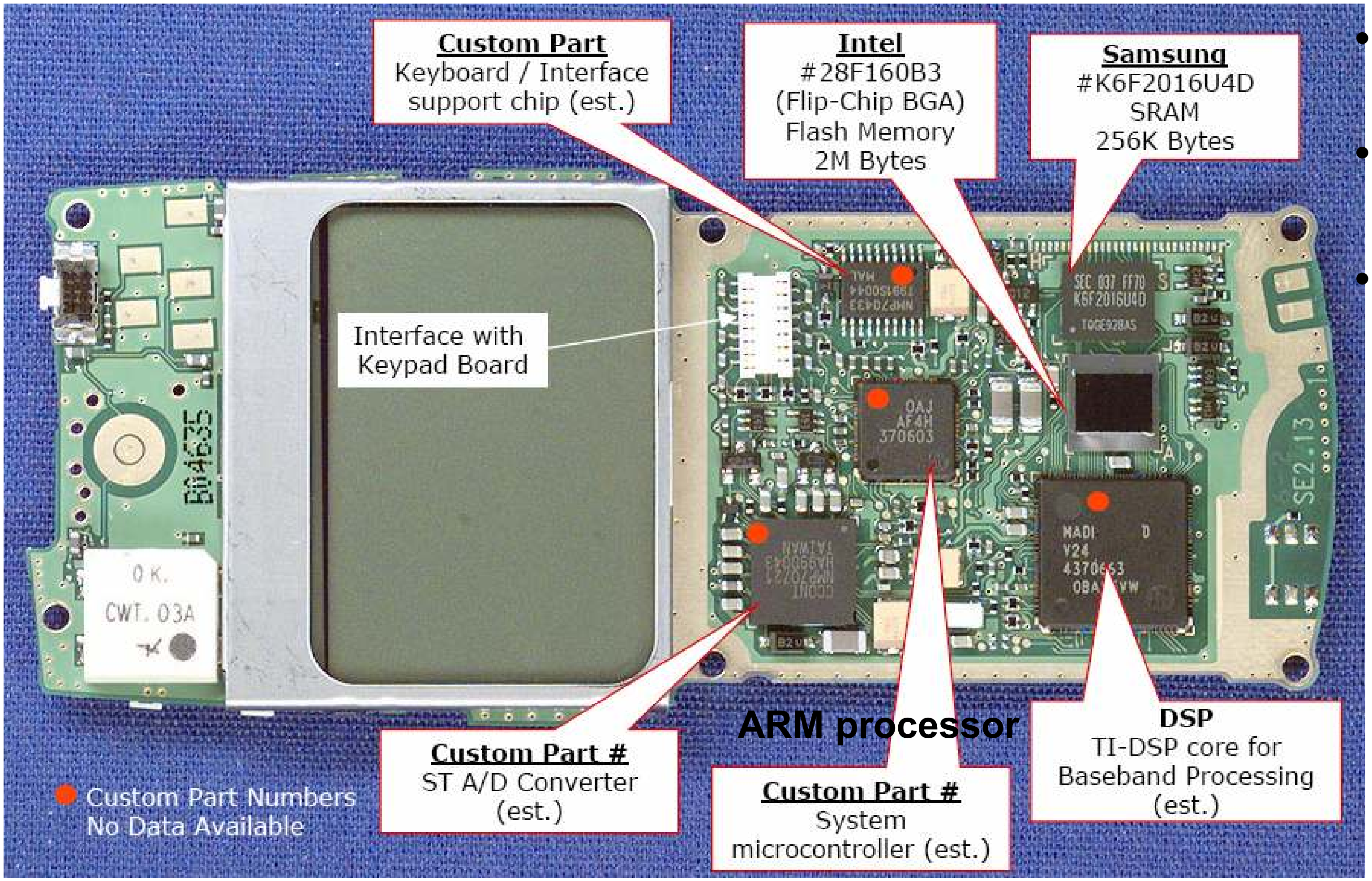
PC Motherboard



PC System

- Pentium 4 2.66 GHz
  - 8KB Data cache, 12 KB Instruction cache
  - 512 KB L2 Cache
  - 533 MHz System Bus
  - 68 Watts
- Memory system
  - 4 DDR DIMM slots
  - Up to 4 GB
- I/O interfaces
  - Ethernet
  - USB
  - Serial ATA (disk)
  - Serial port
  - Parallel port

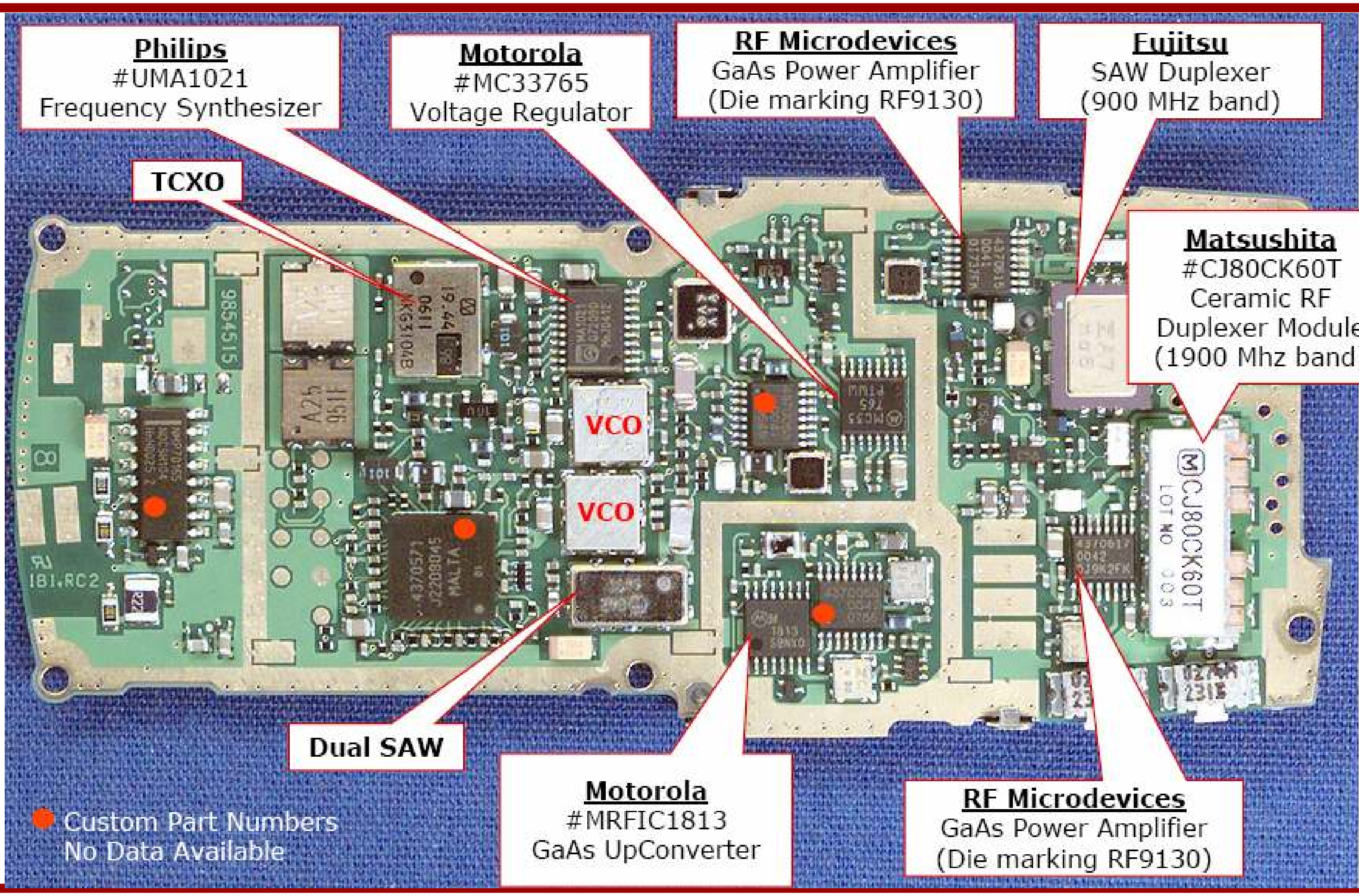
Digital Cell Phone (Nokia 8260) Front Side



- Battery 900 mAhr
- 3.5 hr talk
  - ~ 1 W
- 8 days standby
  - ~ 1mW

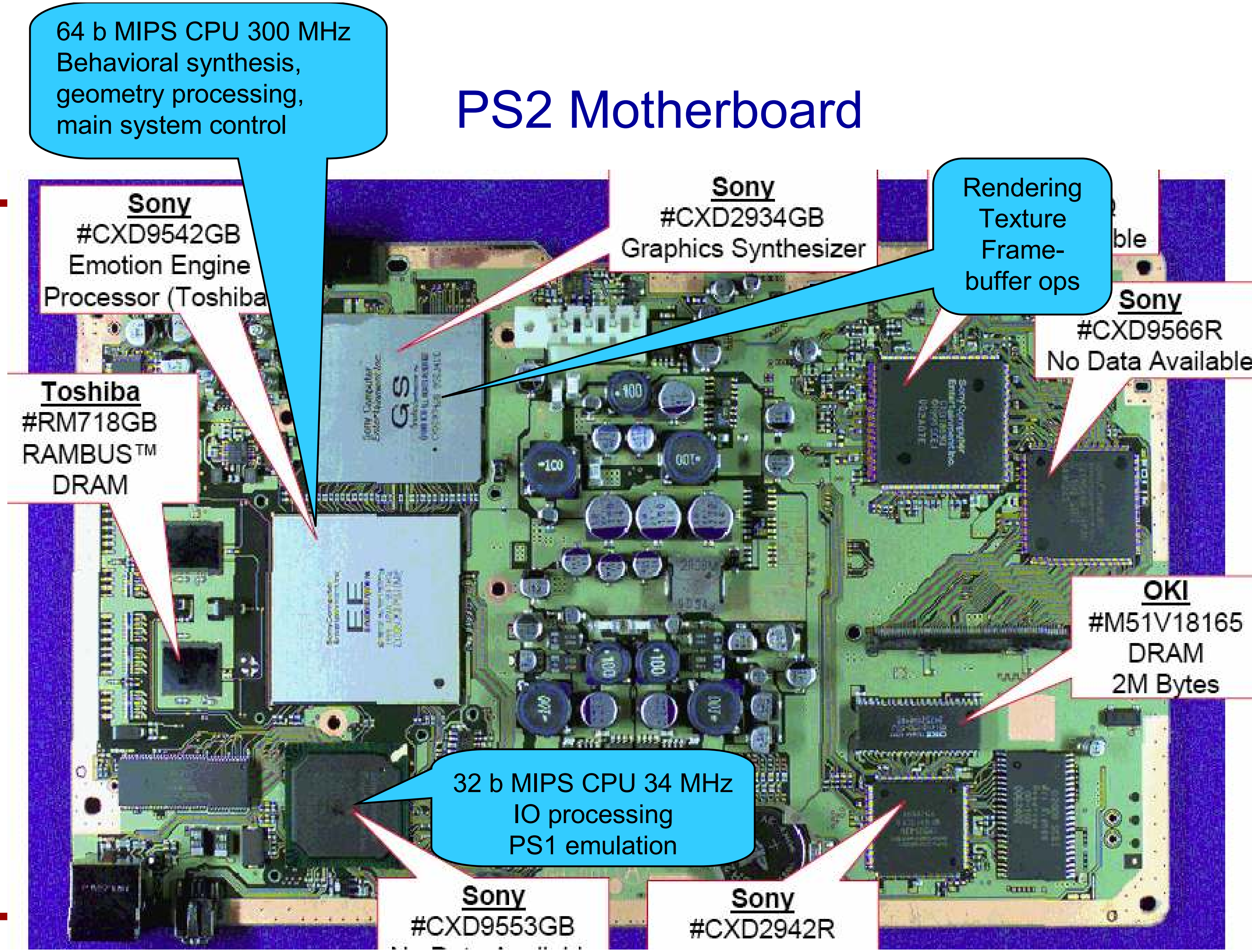
 Courtesy of Portelligent

Digital Cell Phone (Nokia 8260)
Back Side



Courtesy of Portelligent

PS2 Motherboard



Courtesy of Portelligent

Looking Forward

- EE108b is about understanding how to build & use such systems
  - From PCs to hi-tech refrigerators
- Starting in the next lecture
  - What is the hardware/software interface?
  - What is the functionality that the hardware must implement?