Lecture 21:

Goodbye to Computer Arhcitecture and How to Have a Bad Career in Research/Academia

Professor David A. Patterson

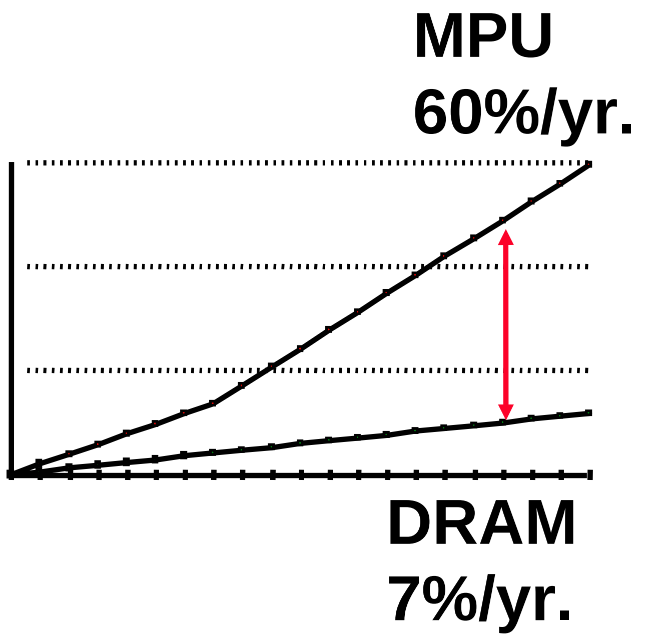
Computer Science 252

Spring 2001

Review of Course

- Review and Goodbye to Computer Architecture, topic by topic + follow-on courses
- How to have a bad career
- HKN teaching evaluation last 15 minutes

Chapter 5: Memory Hierarchy



- Processor-DRAM Performance gap
- 1/3 to 2/3 die area for caches, TLB
- Alpha 21264: 108 clock to memory
  $\Rightarrow$ 648 instruction issues during miss
- 3 Cs: Compulsory, Capacity, Conflict
- 4 Questions: where, who, which, write
- Applied recursively to create multilevel caches
- Performance = *f*(hit time, miss rate, miss penalty)
  – danger of concentrating on just one when evaluating performance

Cache Optimization Summary

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{Memory\ \ accesses}{Instruction} \times \textbf{Miss rate} \times Miss\ \ penalty \right) \times Clock\ \ cycle\ \ time$$

	Technique	*MR*	*MP*	*HT*	*Complexity*
miss rate	Larger Block Size	+	–		0
	Higher Associativity	+		–	1
	Victim Caches	+			2
	Pseudo-Associative Caches	+			2
	HW Prefetching of Instr/Data	+			2
	Compiler Controlled Prefetching	+			3
	Compiler Reduce Misses	+			0
miss penalty	Priority to Read Misses		+		1
	Subblock Placement		+	+	1
	Early Restart & Critical Word 1st		+		2
	Non-Blocking Caches		+		3
	Second Level Caches		+		2
hit time	Small & Simple Caches	–		+	0
	Avoiding Address Translation			+	2
	Pipelining Writes			+	1

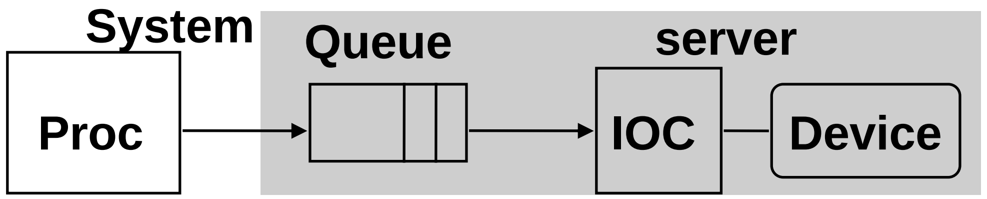
memory hierarchy art: taste in selecting between alternatives to find combination that fits well together

Goodbye to Memory Hierarchy

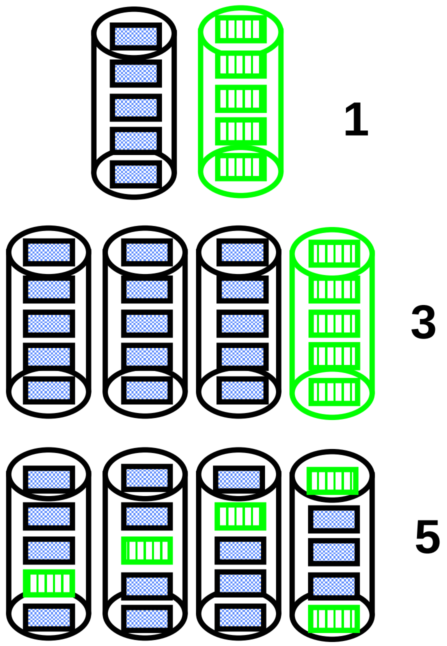
- **Will L2 cache keep growing? (e.g, 64 MB L2 cache?)**
- **Will multilevel hierarchy get deeper? (L4 cache?)**
- **Will DRAM capacity/chip keep going at 4X / 4 years? (e.g., 16 Gbit chip?)**
- **Will processor and DRAM/Disk be unified? For which apps?**
- **Out-of-order CPU hides L1 data cache miss (3–5 clocks), but hide L2 miss? (>100 clocks)**
- **Memory hierarchy likely overriding issue in algorithm performance: do algorithms and data structures of 1960s work with machines of 2000s?**

Chapter 6: Storage I/O

- Disk BW 40%/yr, areal density 60%/ yr, $/MB faster?
- Little's Law: $Length_{system} = rate \times Time_{system}$
  (Mean number customers = arrival rate x mean service time)



  - Througput vs. response time
  - Value of faster response time on productivity
- Benchmarks: scaling, cost, auditing, response time limits
- RAID: performance and reliability
- Queueing theory? IEOR 161, 267, 268
- CS262-A: Advanced Topics in Computer Systems



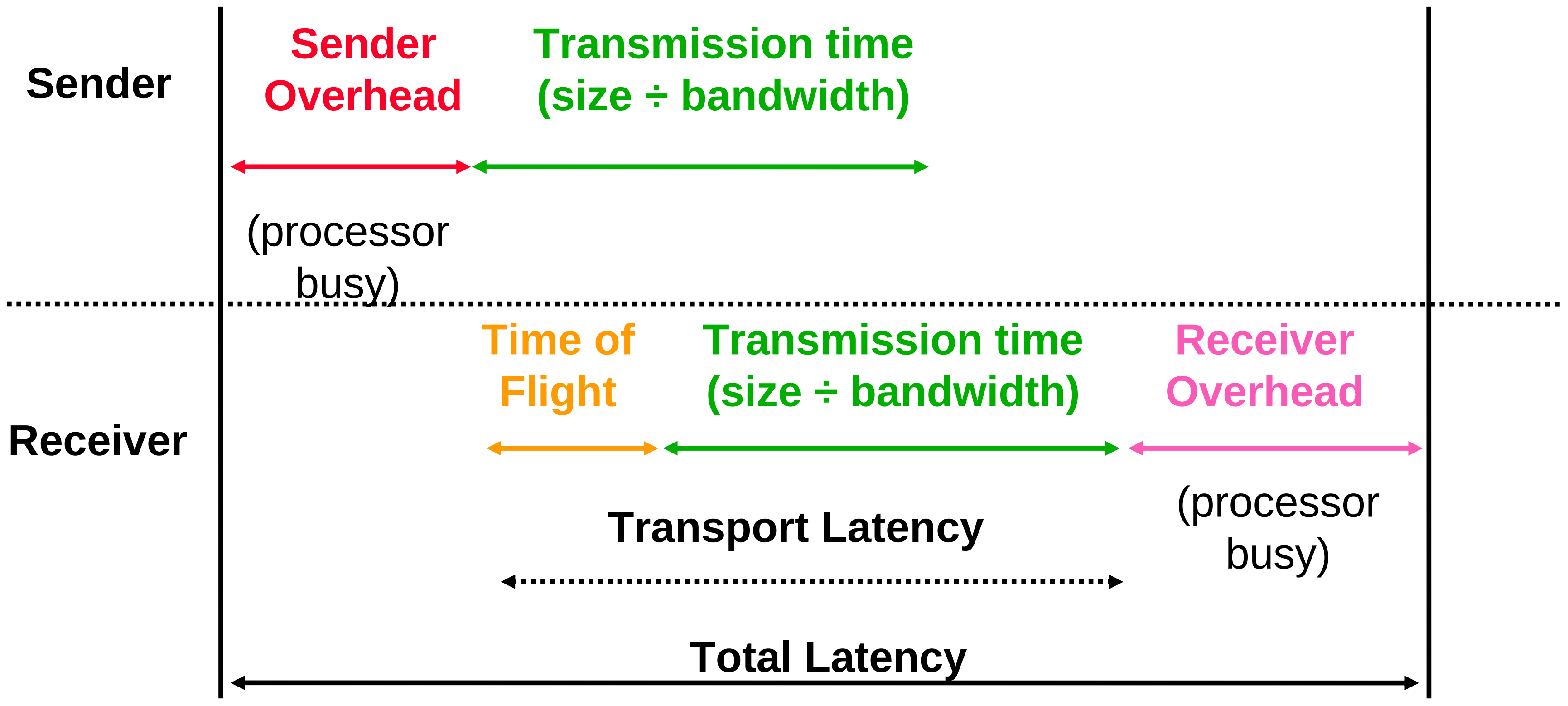
Goodbye to Storage I/O

- Disks attached directly to networks, avoiding the file server? ("Network Attached Storage Devices")
- Disks:
  - Extraodinary advance in capacity/drive, $/GB
  - Currently 17 Gbit/sq. in. ; can continue past 100 Gbit/sq. in.?
  - Bandwidth, seek time not keeping up: 3.5 inch form factor makes sense? 2.5 inch form factor in near future? 1.0 inch form factor in long term?
  - Heading towards a personal terabyte:
    hierarchical file systems vs. database to organize personal storage?
  - What going to do when can have video record of entire life on line?
  - Will Patterson continue get email messages to reduce file storage for the rest of his career?
- Tapes
  - No investment, must be backwards compatible
  - Are they already dead?
  - What is a tapeless backup system?

Goodbye to Storage I/O

- I/O Benchmarks: Must scale to track technological change
- TPC: price performance as normalizing configuration feature
  - Auditing to ensure no foul play
  - Throughput with restricted response time is normal measure
- Terminology of Fault/Error/Failure
- Is Availability the killer metric for Service oriented world?
- Can we construct systems that will actually achieve 99.999% availability, including software and people?
- Benchmarks to measure Availability, Maintainability thereby creating a competitive environment?
- Interested in learning more on HW/SW Availability/Maintability? CS 294-4 “High Confidence Computing” Fall 01 (Patterson, Armando Fox, Stanford): 10 students from Berkeley, 10 from Stanford; at Berkeley Thursdays 3-6

Chapter 7: Networks



Total Latency = Sender Overhead + Time of Flight + Message Size ÷ BW + Receiver Overhead

High BW networks + high overheads violate of Amdahl’s Law

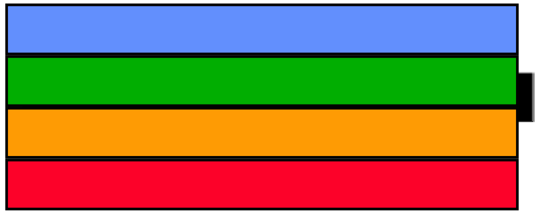
Chapter 7: Networks

- Similarities of SANs, LANs, WANs
- Integrated circuit revolutionizing networks as well as processors
- Switch is a specialized computer
- Protocols allow hetereogeneous networking ,
  handle normal and abnormal events
- Interested in learning more on networks?
  EE 122 “Introduction to Computer Networks” (Fa 01Stoika)
  CS 268 “Computer Networks” (Spr 02 Stoika)

Goodbye to Networking

- Clusters +: fault isolation and repair, scaling, cost
- Clusters -: maintenance, network interface performance,
- Google as cluster example:
  - scaling (8000 PCs, 1 petabyte storage)
  - fault isolation (2 failures per day yet available)
  - repair (replace failures weekly/repair offline)
  - Maintenance: 8 people for 8000 PCs
- Cell phone as portable network device
  - # Handsets >> # PCs
  - Univerisal mobile interface?
- Is future services built on Google-like clusters delivered to gadgets like cell phone handset?
- Will network interfaces follow example of graphics interfaces and become first class citizens in microprocessors, thereby avoiding the I/O bus?
- Will Ethernet standard keep winning the LAN wars? e.g., 1 Gbit/sec, 10 Gbit/sec, wireless (802.11B)...

Chapter 8: Multiprocessors



- Layers:
  - Programming Model:
    - » Multiprogramming : lots of jobs, no communication
    - » Shared address space: communicate via memory
    - » Message passing: send and recieve messages
    - » Data Parallel: several agents operate on several data sets simultaneously and then exchange information globally and simultaneously (shared or message passing)
  - Communication Abstraction:
    - » Shared address space: e.g., load, store, atomic swap
    - » Message passing: e.g., send, recieve library calls
    - » Debate over this topic (ease of programming, scaling) => many hardware designs 1:1 programming model
- Interested in learning more on multiprocessors: CS 258 “Parallel Computer Architecture” (Kubiatowicz, Spr 02)
- E 267 “Programming Parallel Computers” (Yelick, Fall 01)
  - I hear and I forget; I see and I remember; I do and I understand

Goodbye to Multiprocessors

- Successful today for file servers, time sharing, databases, graphics; will parallel programming become standard for production programs?
  If so, what enabled it: new programming languauges, new data structures, new hardware, new coures, ...?
- Which won large scale number crunching, databases: Clusters of independent computers connected via switched LAN vs. large shared NUMA machines? Why?
- Is explicit parallelism only path to performance later in decade?
- Will we have 10 processors per chip? Will we have 100 processors per chip? Will extra processors help with availability?
- Was the popular parallel programming model thread-based vs. message based?

Chapter 2: Instruction Set Architecture

- **What ISA looks like to pipeline?**
  - **Cray: load/store machine; registers; simple instr. format**
- **RISC: Making an ISA that supports pipelined execution**
- **80x86: importance of being their first**
- **VLIW/EPIC: compiler controls Instruction Level Parallelism (ILP)**
- **Interested in learning more on compilers and ISA? CS 264/5 “Advanced Programming Language Design and Optimization”**

Goodbye to Instruction Set Architecture

- What did IA-64/EPIC do well besides floating point programs?
  - Was the only difference the 64-bit address v. 32-bit address?
  - What happened to the AMD 64-bit address 80x86 proposal?
- What happened on EPIC code size vs. x86?
- Did Intel Oregon keep increasing x86 performance so as to keep Intel Santa Clara EPIC performance embarrassing?
- Did anybody propose anything at ISA to help with software quality? With availability?

Goodbye to Dynamic Execution (Chapter 3)

- **Did Transmeta-like compiler-oriented translation survive vs. hardware translation into more efficient internal instruction set?**
- **Did ILP limits really restrict practical machines to 4-issue, 4-commit?**
- **Did we ever really get CPI below 1.0?**
- **Did value prediction become practical?**
- **Branch prediction: How accurate did it become?**
  - **For real programs, how much better than 2 bit table?**
- **Did Simultaneous Multithreading (SMT) exploit underutilized Dynamic Execution HW to get higher throughput at low extra cost?**
  - **For multiprogrammed workload (servers) or for parallelized single program?**

Goodbye to Static, Embedded

- Did VLIW become popular in embedded? What happened on code size?
- Did vector become popular for media applications, or simply keep evolving SIMD?
- Did DSP and general purpose microprocessors remain separate cultures, or did ISAs and cultures merge?
  - Compiler oriented?
  - Benchmark oriented?
  - Library oriented?
  - Saturation vs. 2's complement?

Goodbye to Computer Architecture

- **1985-2000: 1000X performance**
  - **Moore's Law transistors/chip => Moore's Law for Performance/MPU**
- **Hennessy: industry been following a roadmap of ideas known in 1985 to exploit Instruction Level Parallelism to get 1.55X/year**
  - **Caches, Pipelining, Superscalar, Branch Prediction, Out-of-order execution, …**
- **ILP limits: To make performance progress in future need to have explicit parallelism from programmer vs. implicit parallelism of ILP exploited by compiler, HW?**
- **Did Moore's Law in transistors stop predicting microprocessor performance? Did it drop to old rate of 1.3X per year?**
  - **Less because of processor-memory performance gap?**
- **Did emphasis switch from cost-performance to cost-performance-availability?**
- **What support for improving software reliability? Security?**

Administratrivia

- Quiz #2 310 Soda at 5:30
  - Did great on Question 2 (Tomasulo)
  - Did good on static pipelining
  - Crusoe vs. Pentium answers more variable
  - Will return quizes at Oral Presentations
- What's left:
  - signup for talks
  - 4/25 Wed, Oral Presentations (8AM to 2 PM) 611 Soda (no lecture)
    - Signup for talks (possible for 3 Eric Anderson suggested topics to be together)
    - Projection from laptop (I'll bring one) or overhead transparencies
    - 22-24 minutes for presentation + 4-5 minutes for questions
  - 4/27 Fri (no lecture)
  - 5/2 Wed Poster session (noon - 2); end of course meetings
  - 5/7 Mon URLs of written projects due by 4:30

Outline

- Part I: Key Advice for a Bad Career while a Grad Student
- Part II: Key Advice on Alternatives to a Bad Graduate Career
- Part III: Key Advice for a Bad Career, Post PhD
- Part IV: Key Advice on Alternatives to a Bad Career, Post PhD
- Topics covered in parts III and IV
  - Selecting a Problem
  - Picking a Solution
  - Performing the Research
  - Evaluating the Results
  - Communicating Results
  - Transferringlogy

Part I: How to Have a Bad Graduate Career

- **Concentrate on getting good grades:**
  - postpone research involvement: might lower GPA
- **Minimize number and flavors of courses**
  - Why take advantage of 1 of the top departments with an emphasis on excellent grad courses?
  - Why take advantage of a campus with 35/36 courses in the top 10?
  - May affect GPA
- **Don’t trust your advisor**
  - Advisor is only interested in his or her own career, not your’s
  - Advisor may try to mentor you, use up time, interfering with GPA
- **Only work the number of hours per week you are paid!**
  - Don’t let master class exploit the workers!

Part I: How to Have a Bad Graduate Career

- Concentrate on graduating as fast as possible
  - Winner is first in class to PhD
  - People only care about that you have a PhD and your GPA, not on what you know
    - » Nirvana: graduating in 3.5 years with a 4.0 GPA!
  - Don't spend a summer in industry: takes longer
    - » How could industry experience help with selecting PhD topic?
  - Don't work on large projects: takes longer
    - » Have to talk to others, have to learn different areas
    - » Synchronization overhead of multiple people
  - Don't do a systems PhD: takes longer
- Don't go to conferences
  - It costs money and takes time; you'll have plenty of time to learn the field after graduating
- Don't waste time polishing writing or talks
  - Again, that takes time

Part II: Alternatives to a Bad Graduate Career

- Concentrate on getting good grades?
  - Reality: need to maintain reasonable grades
    - » Only once gave a below B in CS 252
    - » 3 prelim courses only real grades that count
  - What matters on graduation is letters of recommendation from 3-4 faculty/PhDs who have known you for 5+ years (including 1 outside of Berkeley: see summer jobs)
- Minimize number and flavors of courses?
  - Your last chance to be exposed to new ideas before have to learn them on your own (re: queueing theory and me)
  - Get a real outside minor from a campus with great departments in all fields; e.g., Management of Technology certificate, Copyright Law
- Don't trust your advisor?
  - Primary attraction of campus vs. research lab is getting to work with grad students
  - Faculty career is judged in large part by success of his or her students

Part II: Alternatives to a Bad Graduate Career

- Concentrate on graduating as fast as possible?
  - Your last chance to learn; most learning will be outside the classroom
  - Considered newly "minted" when finish PhD
    - » Judged on year of PhD vs. year of birth
    - » To a person in their 40s or 50s, 1 or 2 more years is roundoff error (27 = 29)
- Don't go to conferences?
  - Chance to see firsthand what the field is like, where its going
  - There are student rates, you can share a room
  - Talk to people in the field in the halls
  - If your faculty advisor won't pay, then pay it yourself; almost always offer student rates, can often share rooms
    - » Prof. Landay paid his own way to conferences while grad student
- Don't waste time polishing writing or talks?
  - In the marketplace of ideas, the more polish the more likely people will pay attention to your ideas
  - Practice presentation AND answering tough questions

Part II: Alternatives to a Bad Graduate Career

- Only work the number of hours per week you are paid?
  - Campus Faculty average is 65-70 hours/work; EECS higher
  - Students should be in that range
  - Organize each day: when most alert? nap? exercise? sleep?
  - When/how often/how long: write, read, program, email?
  - To do lists: daily, weekly, semester
- Industrial Experience?
  - 1st or 2nd summer get work experience, or 1 semester off
- Sutherland's advice (Father of Computer Graphics)
  - Be bold; Take chances on hard topics
  - *Technology and Courage* see pdf Spr 98 CS252 home page
- Advice from a very successful recent student; Remzi Arpaci
  - Great ideas, did lots of papers, well thought of
  - I asked: Why do you think you did so well?
  - He said I gave him advice the first week he arrived

Part II: How to be a Success in Graduate School

- 1) **“Swim or Sink”**
  - “Success is determined by me (student) primarily”
  - Faculty will set up the opportunity, but its up to me leverage it
- 2) **“Read/learn on your own”**
  - “Related to 1), I think you told me this as you handed me a stack of about 20 papers”
- 3) **“Teach your advisor”**
  - “I really liked this concept; go out and learn about something and then teach the professor”
  - Fast moving field, don’t expect prof to be at forefront everywhere

Outline

- Part I: Key Advice for a Bad Career while a Grad Student
- Part II: Key Advice on Alternatives to a Bad Graduate Career
- Part III: Key Advice for a Bad Career, Post PhD
- Part IV: Key Advice on Alternatives to a Bad Career, Post PhD
- Topics covered in parts III and IV
  - Selecting a Problem
  - Picking a Solution
  - Performing the Research
  - Evaluating the Results
  - Communicating Results
  - Transferring Technology

Bad Career Move #1: Be THE leading expert

- Invent a new field!
  - Make sure its slightly different
- Be the real Lone Ranger: Don’t work with others
  - No ambiguity in credit
  - Adopt the Prima Donna personalityResearch Horizons
  - Never define success
  - Avoid Payoffs of less than 20 years
  - Stick to one topic for whole career
  - Even if technology appears to leave you behind, stand by your problem

Announcing a New Operating System Field: "Disability Based Systems"

- Computer SecurityInsight: capability based addressing almost right
- Idea: Create list of things that process CANNOT do!
- Key Question:
  should you store disabilities with each user
  or with the objects they can't access?
- Other topics: encrypted disabilities, disability-based addressing
- Start a new sequence of courses

Announcing yet another New O.S. Field: “Omni-Femtokernels”

- “Femto” – microkernels, only more so
- “Omni” – omnipresent, run femtokernels everywhere:
  - Operating System
  - Applications
  - VCRs
  - automobiles
- Key contribution: employment

Bad Career Move #2: Let Complexity Be Your Guide (Confuse Thine Enemies)

- Best compliment:
  "Its so complicated, I can't understand the ideas"
  - If no one understands, how can they contradict your claim?
- It's easier to be complicated
  - Also: to publish it must be different; N+1st incremental change
- If it were not unsimple then how could distinguished colleagues in departments around the world be positively appreciative of both your extraordinary skills and talents

Bad Career Move #3: Never be Proven Wrong

- Avoid Implementing
- Avoid Quantitative Experiments
  - If you’ve got good intuition, who needs experiments?
  - Why give grist for critics’ mill?
  - Takes too long to measure
- Avoid Benchmarks
- Projects whose payoff is >= 20 years
  gives you 19 safe years

Bad Career Move #4: Use the <u>Computer</u> Scientific Method

Obsolete Scientific Method

- Hypothesis
- Sequence of experiments
- Change 1 parameter/exp.
- Prove/Disprove Hypothesis
- Document for others to reproduce results

Computer Scientific Method

- Hunch
- 1 experiment & change all parameters
- Discard if doesn't support hunch
- Why waste time? We know this

Bad Career Move #5:
Don’t be Distracted by Others (Avoid Feedback)

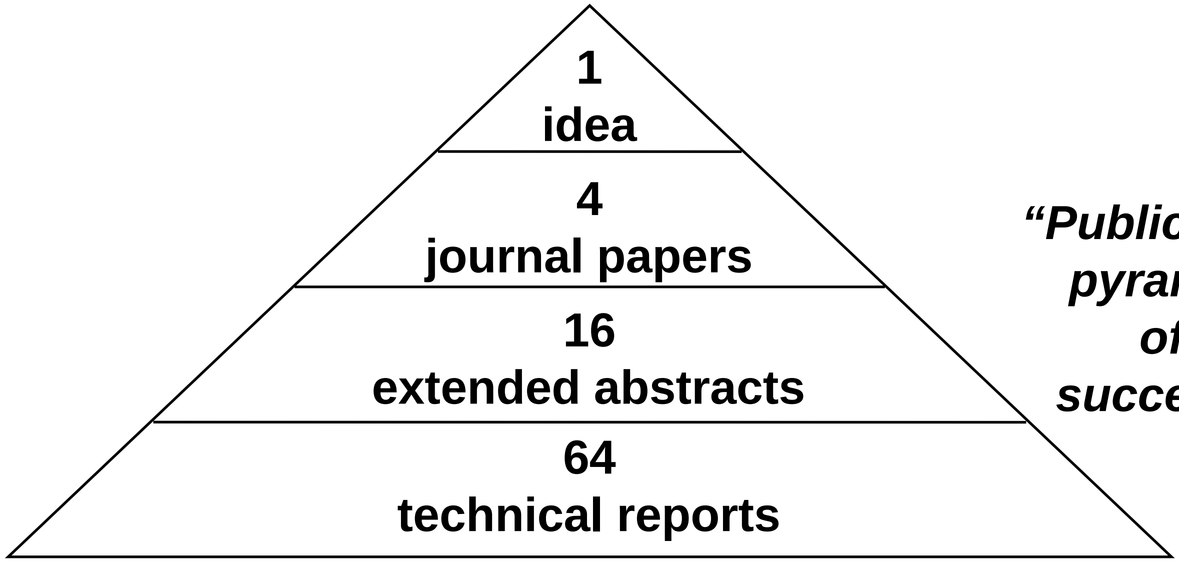
- Always dominate conversations: Silence is ignorance
  - Corollary: Loud is smart
- Don’t read
- Don’t be tainted by interaction with users, industry
- Reviews
  - If it's simple and obvious in retrospect => Reject
  - Quantitative results don't matter if they just show you what you already know => Reject
  - Everything else => Reject

Bad Career Move #6: Publishing Journal Papers IS Technology Transfer

- **Target Archival Journals: the Coin of the Academic Realm**
  - **It takes 2 to 3 years from submission to publication=>timeless**
- **As the leading scientist, your job is to publish in journals*not* your job to make you the ideas palatable; wastes valuable research time**
  - **Travel time, having to interact with others, serve on program committees, …**

Bad Career Move #7: Writing Tactics for a Bad Career

- Papers: It's Quantity, not Quality
  - Personal Success = Length of Publication List
  - "The LPU (Least Publishable Unit) is Good for You"



"Publication pyramid of success"

- Student productivity = number of papers
  - Number of students: big is beautiful
  - Never ask students to implement: reduces papers
- Legally change your name to Aaaanderson

5 Writing Commandments for a Bad Career

- **I. Thou shalt not define terms, nor explain anything.**
- **II. Thou shalt replace “will do” with “have done”.**
- **III. Thou shalt not mention drawbacks to your approach.**
- **IV. Thou shalt not reference any papers.**
- **V. Thou shalt publish before implementing.**

7 Talk Commandments for a Bad Career

I. Thou shalt not illustrate.

II. Thou shalt not covet brevity.

III. Thou shalt not print large.

IV. Thou shalt not use color.

V. Thou shalt not skip slides in a long talk.

VI. Thou shalt not practice.

Following all the commandments

- We describe the philosophy and design of the control flow machine, and present the results of detailed simulations of the performance of a single processing element. Each factor is compared with the measured performance of an advanced von Neumann computer running equivalent code. It is shown that the control flow processor compares favorablylism in the program.
- We present a denotational semantics for a logic program to construct a control flow for the logic program. The control flow is defined as an algebraic manipulator of idempotent substitutions and it virtually reflects the resolution deductions. We also present a bottom-up compilation of medium grain clusters from a fine grain control flow graph. We compare the basic block and the dependence sets algorithms that partition control flow graphs into clusters.
- Our compiling strategy is to exploit coarse-grain parallelism at function application level: and the function application level parallelism is implemented by fork-join mechanism. The compiler translates source programs into control flow graphs based on analyzing flow of control, and then serializes instructions within graphs according to flow arcs such that function applications, which have no control dependency, are executed in parallel.
- A hierarchical macro-control-flow computation allows them to exploit the coarse grain parallelism inside a macrotask, such as a subroutine or a loop, hierarchically. We use a hierarchical definition of macrotasks, a parallelism extraction scheme among macrotasks defined inside an upper layer macrotask, and a scheduling scheme which assigns hierarchical macrotasks on hierarchical clusters.
- We apply a parallel simulation scheme to a real problem: the simulation of a control flow architecture, and we compare the performance of this simulator with that of a sequential one. Moreover, we investigate the effect of modelling the application on the performance of the simulator. Our study indicates that parallel simulation can reduce the execution time significantly if appropriate modelling is used.
- We have demonstrated that to achieve the best execution time for a control flow program, the number of nodes within the system and the type of mapping scheme used are particularly important. In addition, we observe that a large number of subsystem nodes allows more actors to be fired concurrently, but the communication overhead in passing control tokens to their destination nodes causes the overall execution time to increase substantially.
- The relationship between the mapping scheme employed and locality effect in a program are discussed. The mapping scheme employed has to exhibit a strong locality effect in order to allow efficient execution. We assess the average number of instructions in a cluster and the reduction in matching operations compared with fine grain control flow execution.
- Medium grain execution can benefit from a higher output bandwidth of a processor and finally, a simple superscalar processor with an issue rate of ten is sufficient to exploit the internal parallelism of a cluster. Although the technique does not exhaustively detect all possible errors, it detects nontrivial errors with a worst-case complexity quadratic to the system size. It can be automated and applied to systems with arbitrary loops and nondeterminism.

Outline

- Part I: Key Advice for a Bad Career
  - Invent a field and Stick to it
  - Let Complexity be Your Guide (Confuse Thine Enemies)
  - Never be Proven Wrong
  - Use the Computer Scientific Method
  - Avoid Feedback
  - Publishing Journal Papers is Technology Transfer
  - Write Many (Bad) Papers by following 5 writing commandments
  - Give Bad Talks by following 6 talk commandments
- Part II: Alternatives to a Bad Career

One Alternative Strategy to a Bad Career

- Caveats:
  - From a project leader's point of view
  - Works for me; not the only way
  - Primarily from academic, computer systesm perspective
- Goal is to have impact:
  Change way people do Computer Science & Engineering
  - Academics have bad benchmarks: published papers
- 6 Steps

  1) Selecting a problem

  2) Picking a solution

  3) Running a project

  4) Finishing a project

  5) Quantitative Evaluation

  6) Transferring Technology

1) Selecting a Problem

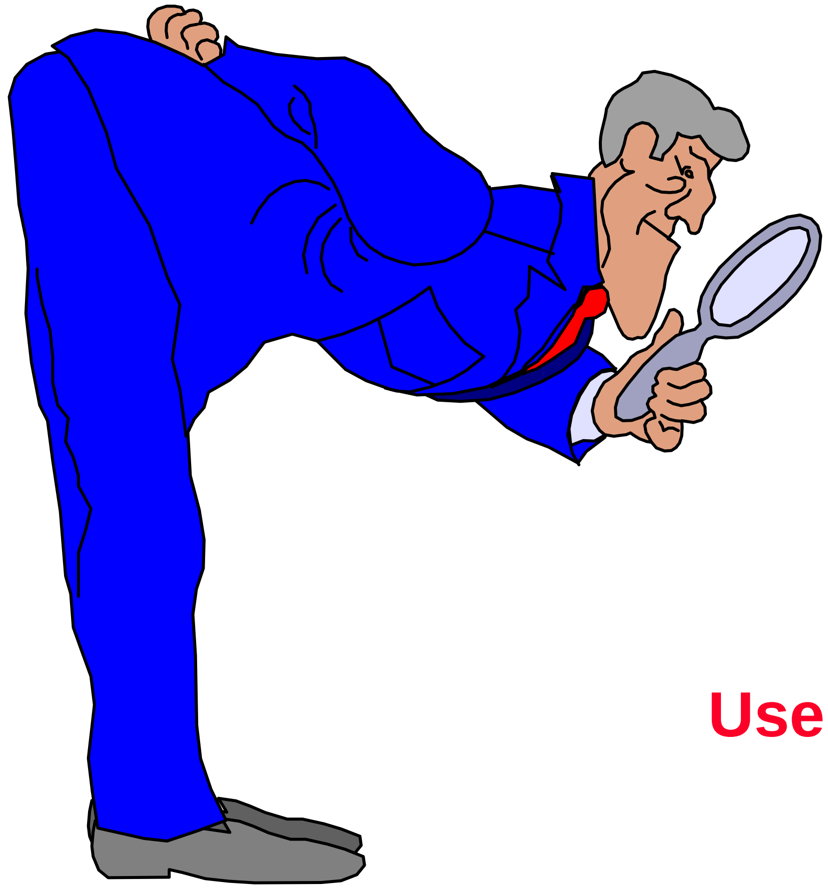
Invent a new field & stick to it?

- No! Do “Real Stuff”: solve problem that someone cares about
- No! Use separate, short projects
  - Always takes longer than expected
  - Matches student lifetimes
  - Long effort in fast changing field???
  - Learning: Number of projects vs. calendar time; I’m on 9th project?
  - If going to fail, better to know soon
- Strive for multi-disciplinary, multiple investigator projects
  - 1 expert/area is ideal (no arguments)
- Match the strengths and weaknesses of local environment

My first project (“Xtree”)

- Multiprocessor project with 3 hardware faculty
- 1977: Design our own instruction set, microprocessor, interconnection topology, routing, boards, systems, operating system
- Unblemished Experience:
  - none in VLSI
  - none in microprocessors
  - none in networking
  - none in operating systems
- Unblemished Resources:
  - No staff
  - No dedicated computer   (used department PDP-11/70)
  - No CAD tools
  - No applications
  - No funding
- Results: 2 journal papers, 12 conference papers, 20 TRs
- Impact?

2) Picking a solution



Let Complexity Be Your Guide?

- No! Keep things simple unless a very good reason not to
  - Pick innovation points carefully, and be compatible everywhere else
  - Best results are obvious in retrospect
    "Anyone could have thought of that"
- Complexity cost is in longer design, construction, test, and debug
  - Fast changing field + delays => less impressive results

Use the Computer Scientific Method?

- No! Run experiments to discover real problems
- Use intuition to ask questions, not answer them

(And Pick A Good Name!)

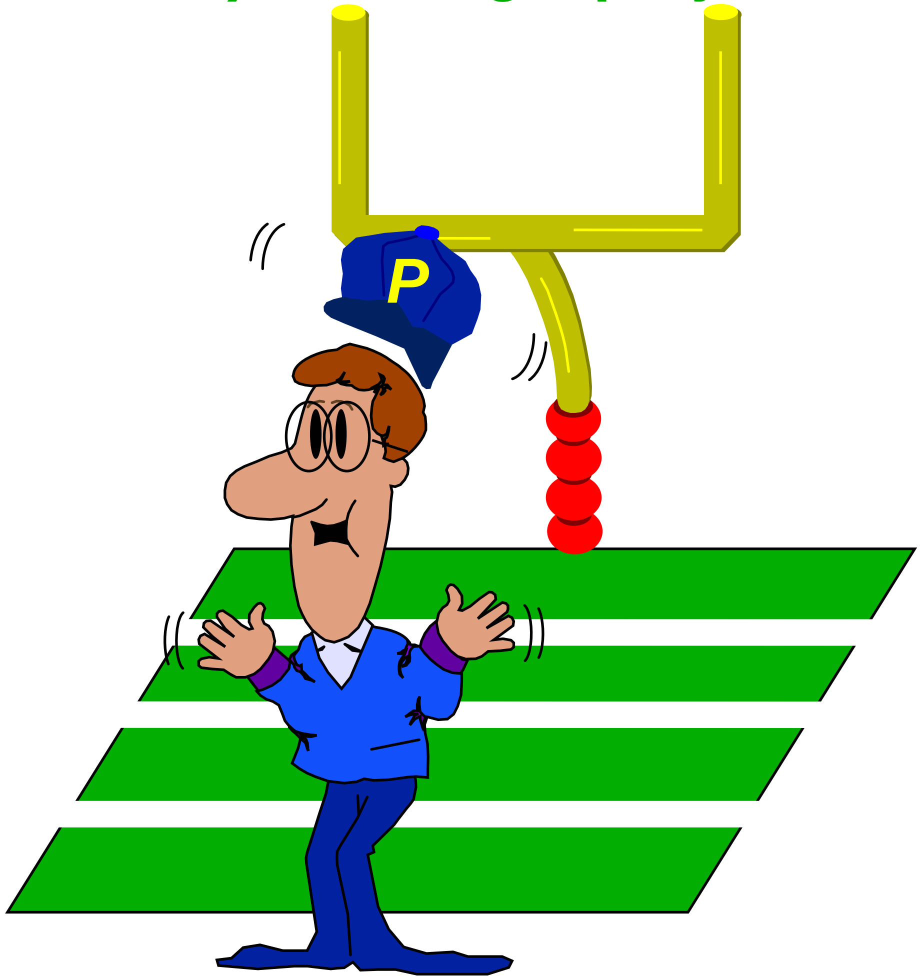
Reduced
I nstruction
Set
Computers

Redundant
Array of
I nexpensive
Disks

I ntelligent
Random
Access
Memory

...

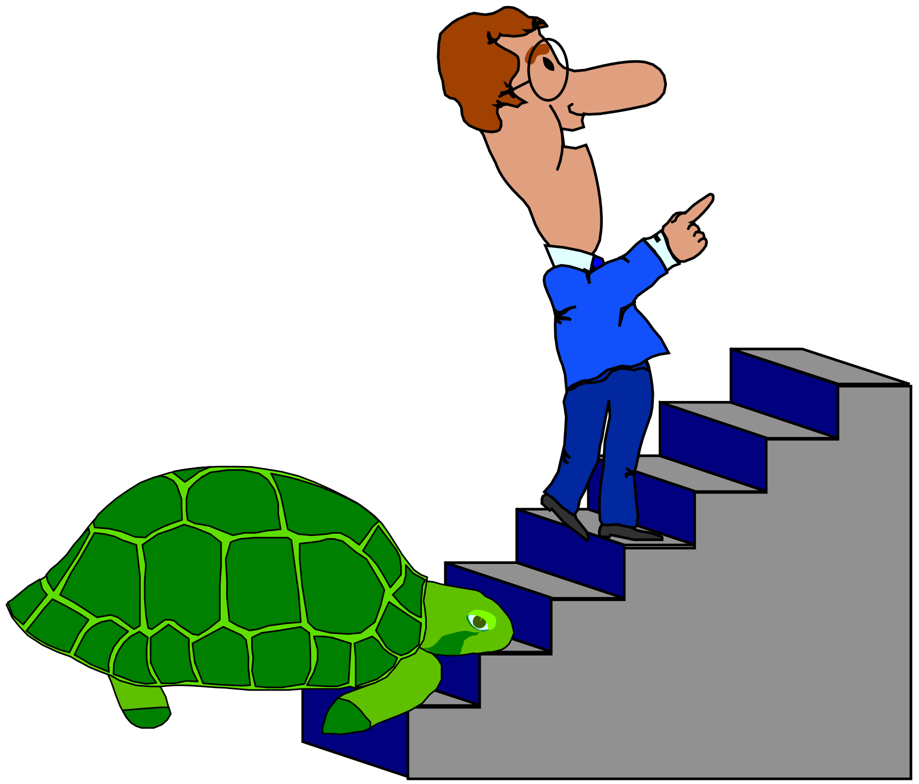
3) Running a project



Avoid Feedback?

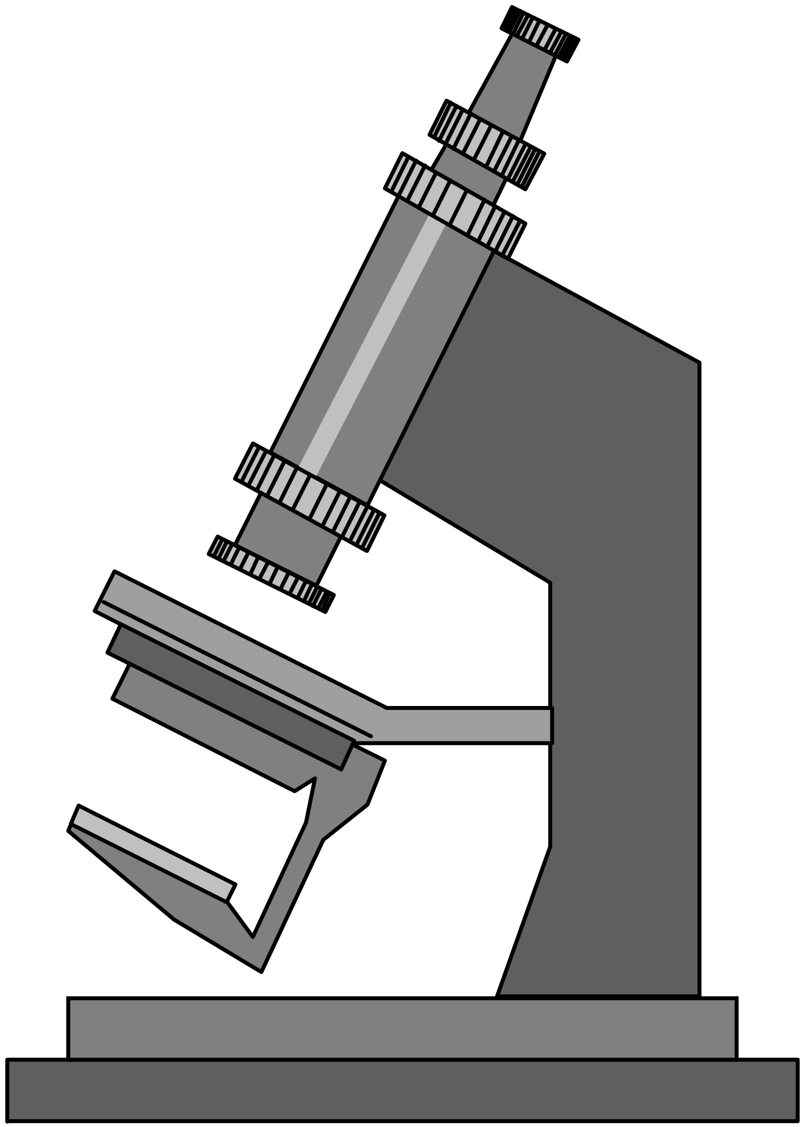
- No! Periodic Project Reviews with Outsiders
  - Twice a year: 3-day retreat
  - faculty, students, staff + guests
  - Key piece is feedback at end
  - Helps create deadlines
  - Give students chance to give many talks, interact with others industry
- Consider mid-course correction
  - Fast changing field & 3-5 year projects => assumptions changed
- Pick size and members of team carefully
  - Tough personalities are hard for everyone
  - Again, 1 faculty per area reduces chance of disagreement

4) Finishing a project



- ***People count projects you finish, not the ones you start***
- ***Successful projects*** go thtrough an unglamorous, hard phase
  - Design is more fun than making it work
  - "No winners on a losing team; no losers on a winning team."
  - "You can quickly tell whether or not the authors have ever built something and made it work."
- Reduce the project if its late
  - "Adding people to a late project makes it later."
- Finishing a project is how people acquire taste in selecting good problems, finding simple solutions

5) Evaluating Quantitatively



Never be Proven Wrong?

- If you can’t be proven wrong, then you can’t prove you’re right
- Report in sufficient detail for others to reproduce results
  - can’t convince others if they can’t get same results
- For better or for worse, benchmarks shape a field
- Good ones accelerate progress
  - good target for development
- Bad benchmarks hurt progress
  - help real users v. help sales?

6) Transferring

Publishing Journal Papers IS Technology Transfer?

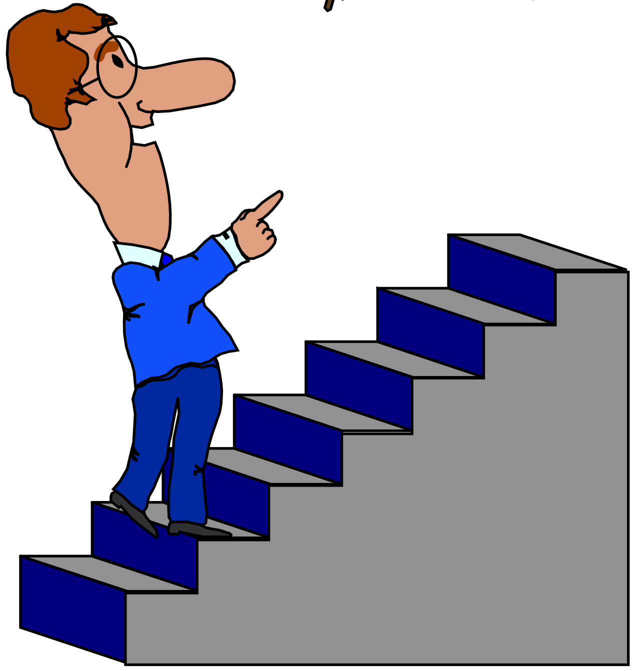
- No! Missionary work: "Sermons" first, then they read papers
  - Selecting problem is key: "Real stuff"
    - » Ideally, more interest as time passes
    - » Change minds with believable results
    - » Prima Donnas interfere with transfer
- My experience: industry is reluctant to embrace change
  - Howard Aiken, circa 1950:

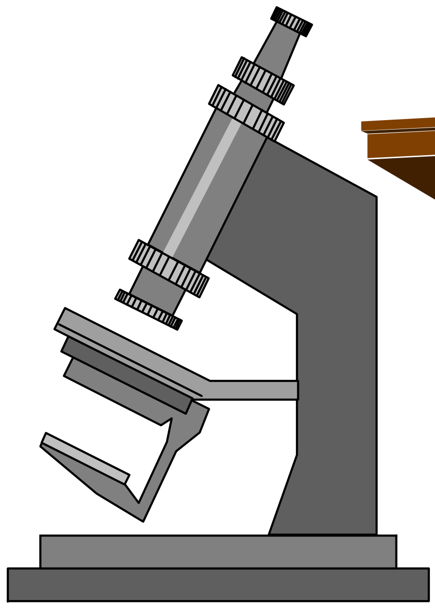
    "The problem in this business isn't to keep people from stealing your ideas; its ***making*** *them steal your ideas!"*
  - Need 1 bold company (often not no. 1) to take chance ***and*** be successful RISC with Sun, RAID with (Compaq, EMC, ...)

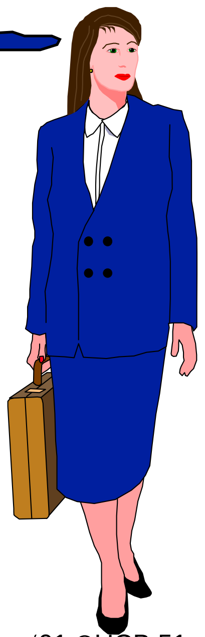
6) Transferring Technology

- Pros
  - Personal satisfaction: seeing your product used by others
  - Personal $$$ (potentially)
  - Fame
- Cons
  - Learn about business plans, sales vs. marketing, financing, personnel benefits, hiring, …
  - Spend time doing above vs. research/development
  - Fame also if company not always successful

Summary: Leader’s Role Changes during Project







Acknowledgments

- Many of these ideas were borrowed from (inspired by?) Tom Anderson, David Culler, Al Davis, John Hennessy, Steve Johnson, John Ousterhout, Bob Sproull, Carlo Séquin and many others

Conclusion: Alternatives to a Bad Career

- **Goal is to have impact:**
  Change way people do Computer Science & Engineering
  - **Many 3 - 5 year projects gives more chances for impact**
- **Feedback is key: seek out & value critics**
- **Do "Real Stuff": make sure you are solving some problem that someone cares about**
- **Taste is critical in selecting research problems, solutions, experiments, & communicating results; acquired by feedback**
- **Your real legacy is people, not paper:**
  create environments that develop professionals of whom you are proud
- ***Students*** **are the coin of the academic realm**

Backup Slides to Help Answer Questions

Applying the Computer Scientific Method to OS

- Create private, highly tuned version for testing
  - take out all special checks: who cares about crashes during benchmarks?
- Never give out code of private version
  - might be embarrassing, no one expects it
- Run experiments repeatedly, discarding runs that don't confirm the generic OS hypothesis
  - Corollary