CS162
Operating Systems and Systems Programming
Lecture 28

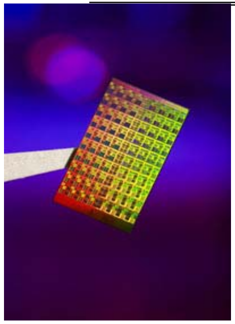
ManyCore, Quantum Computing and Other Topics

December 10, 2008
Prof. John Kubiatowicz
http://inst.eecs.berkeley.edu/~cs162

Requests for Final Topics

- Some topics people requested:
  - Dragons: too big of a topic for today
  - ManyCore Operating Systems
  - Quantum Computers (and factoring)
  - Mobile Operating Systems
  - User Sessions
  - Power Management
  - Data Privacy
  - Berkeley OS History
- Today:
  - ManyCore/Parallel OS
  - Realtime OS
  - Quantum Computing and Quantum factoring
- Other Topics:
  - Come look for me at office hours (Or any other time)

ManyCore Chips: The future is on the way



- Intel 80-core multicore chip (Feb 2007)
  - 80 simple cores
  - Two floating point engines /core
  - Mesh-like "network-on-a-chip"
  - 100 million transistors
  - 65nm feature size

Frequency	Voltage	Power	Bandwidth	Performance
3.16 GHz	0.95 V	62W	1.62 Terabits/s	1.01 Teraflops
5.1 GHz	1.2 V	175W	2.61 Terabits/s	1.63 Teraflops
5.7 GHz	1.35 V	265W	2.92 Terabits/s	1.81 Teraflops

- "ManyCore" refers to many processors/chip
  - 64? 128? Hard to say exact boundary
- How to program these?
  - Use 2 CPUs for video/audio
  - Use 1 for word processor, 1 for browser
  - 76 for virus checking???
- Something new is clearly needed here…

Traditional Parallel OS

- Job of OS is support and protect
  - Need to stay out of way of application
- Traditional single-threaded OS
  - Only one thread active inside kernel at a time
    - » One exception – interrupt handlers
    - » Does not mean that that there aren't many threads – just that all but one of them are asleep or in user-space
    - » Easiest to think about – no problems introduced by sharing
  - Easy to enforce if only one processor (with single core)
    - » Never context switch when thread is in middle of system call
    - » Always disable interrupts when dangerous
  - Didn't get in way of performance, since only one task could actually happen simultaneously anyway
- Problem with Parallel OSs: code base already very large by time that parallel processing hit mainstream
  - Lots of code that couldn't deal with multiple simultaneous threads ⇒One or two locks for whole system

Some Tricky Things about Parallel OSs

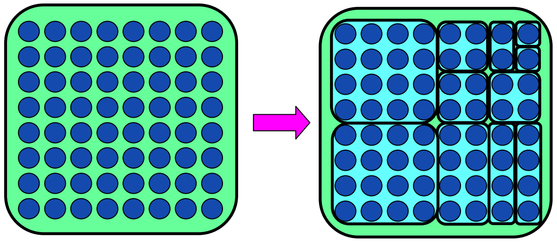
- How to get truly multithreaded kernel?
  - More things happening simultaneously⇒need for:
    - » Synchronization: thread-safe queues, critical sections, …
    - » Reentrant Code – code that can have multiple threads executing in it at the same time
    - » Removal of global variables – since multiple threads may need a variable at the same time
  - Potential for greater performance⇒need for:
    - » Splitting kernel tasks into pieces
- Very labor intensive process of parallelizing kernel
  - Starting from pre-existing code base: very hard
  - Needed to rewrite major portions of kernel with finer-grained locks
    - » Shared among multiple threads on multiple processors⇒ Must satisfy multiple parallel requests
    - » Bottlenecks (coarse-grained locks) in resource allocation can kill all performance
- Truly multithreaded mainstream kernels are recent:
  - Linux 2.6, Windows XP, …

How Should OSs Change for ManyCore?

ManyCore opportunities: Rethink the Sink

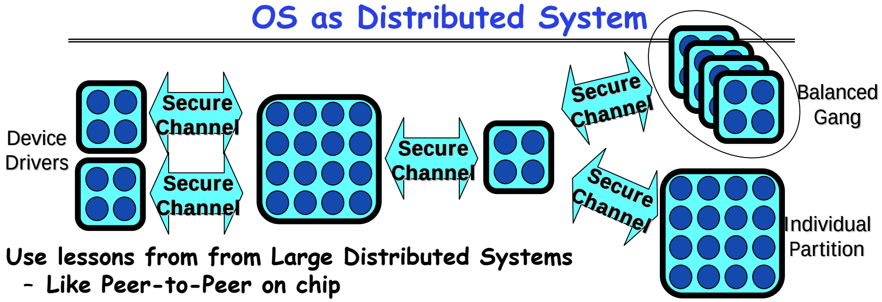
- Computing Resources are *not* Limited
  - High Utilization of *every* core unnecessary
  - Partition *Spatially* rather than *Temporally*
- Protection domains not necessarily heavyweight
  - Spatial Partitioning⇒ protection crossing as simple as sending a message from one part of chip to another
- I/O devices *not* necessarily limited and *do not need to be* heavily multiplexed
  - High bandwidth devices available through network
  - FLASH or other persistent storage yields fast, flat hierarchy (not necessarily disk as bottleneck)
- New constraints
  - Power/Energy major concern
  - Security extremely important
  - Parallelism *must* be exploited in applications
    - » Extremely important for OS to get out of the way

Important New Mechanism: Spatial Partitioning



- Spatial Partition: group of processors acting within hardware boundary
  - Boundaries are "hard", communication between partitions controlled
  - Anything goes within partition
- Each Partition receives a *vector* of resources
  - Some number of dedicated processors
  - Some set of dedicated resources (exclusive access)
    - » Complete access to certain hardware devices
    - » Dedicated raw storage partition
  - Some guaranteed fraction of other resources (QoS guarantee):
    - » Memory bandwidth, Network bandwidth
    - » fractional services from other partitions
- Key Idea: Resource Isolation Between Partitions

OS as Distributed System

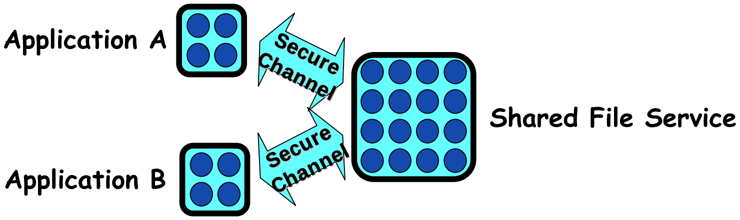


- Use lessons from from Large Distributed Systems
  - Like Peer-to-Peer on chip
  - OS is a set of independent interacting components
  - Shared state across components minimized
- Component-based design:
  - All applications designed with pieces from many sources
  - Requires composition: Performance, Interfaces, Security
- Spatial Partitioning Advantages:
  - Protection of computing resources *not required* within partition
    - » High walls between partitions ⇒ anything goes within partition
    - » "Bare Metal" access to hardware resources
  - Partitions exist simultaneously ⇒ fast communication between domains
    - » Applications split into distrusting partitions w/ controlled communication
    - » Hardware acceleration/tagging for fast secure messaging

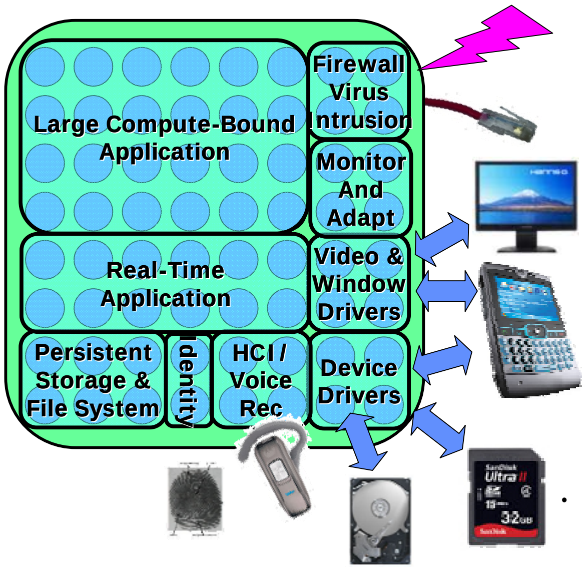
---

It's all about the communication

- We are interested in communication for many reasons:
  - Communication represents a security vulnerability
  - Quality of Service (QoS) boils down message tracking
  - Communication efficiency impacts decomposability
- Shared components complicate resource isolation:
  - Need distributed mechanism for tracking and accounting of resource usage
    - » E.g.: How do we guarantee that each partition gets a guaranteed fraction of the service:

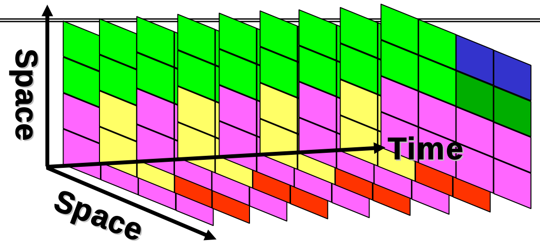


Tessellation: The Exploded OS



- Normal Components split into pieces
  - Device drivers (Security/Reliability)
  - Network Services (Performance)
    - » TCP/IP stack
    - » Firewall
    - » Virus Checking
    - » Intrusion Detection
  - Persistent Storage (Performance, Security, Reliability)
  - Monitoring services
    - » Performance counters
    - » Introspection
  - Identity/Environment services (Security)
    - » Biometric, GPS, Possession Tracking
- Applications Given Larger Partitions
  - Freedom to use resources arbitrarily

Space-Time Partitioning

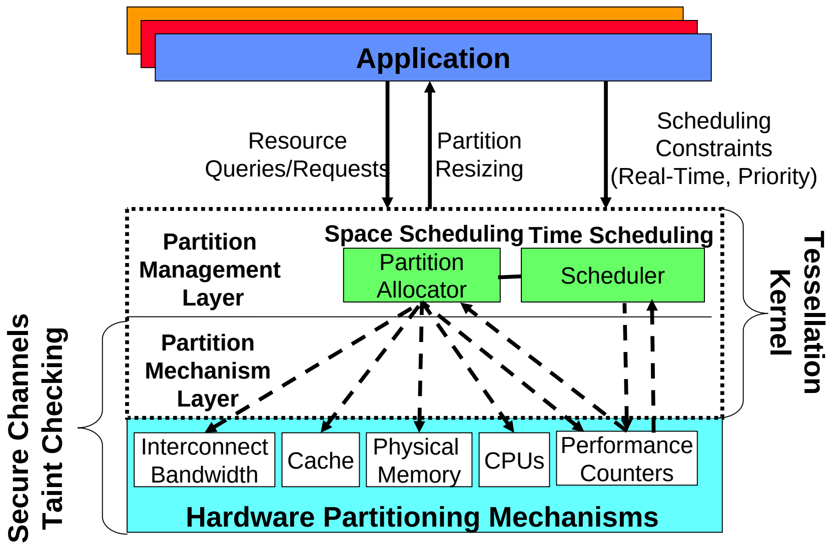


- Spatial Partitioning Varies over Time
  - Partitioning adapts to needs of the system
  - Some partitions persist, others change with time
  - Further, Partititions can be Time Multiplexed
    - » Services (i.e. file system), device drivers, hard realtime partitions
    - » Some user-level schedulers will time-multiplex threads within a partition
- Global Partitioning Goals:
  - Power-performance tradeoffs
  - Setup to achieve QoS and/or Responsiveness guarantees
  - Isolation of real-time partitions for better guarantees
- Monitoring and Adaptation
  - Integration of performance/power/efficiency counters

Another Look: Two-Level Scheduling

- First Level: Gross partitioning of resources
  - Goals: Power Budget, Overall Responsiveness/QoS, Security
  - Partitioning of CPUs, Memory, Interrupts, Devices, other resources
  - Constant for sufficient period of time to:
    - » Amortize cost of global decision making
    - » Allow time for partition-level scheduling to be effective
  - Hard boundaries ⇒ interference-free use of resources
- Second Level: Application-Specific Scheduling
  - Goals: Performance, Real-time Behavior, Responsiveness, Predictability
  - CPU scheduling tuned to specific applications
  - Resources distributed in application-specific fashion
  - External events (I/O, active messages, etc) deferrable as appropriate
- Justifications for two-level scheduling?
  - Global/cross-app decisions made by 1st level
    - » E.g. Save power by focusing I/O handling to smaller # of cores
  - App-scheduler (2nd level) better tuned to application
    - » Lower overhead/better match to app than global scheduler
    - » No global scheduler could handle all applications

Tessellation Partition Manager



Administrivia

- Midterm II
  - Grading is done!
    - » Mean=66.2, Std=14
  - I put up solutions already
- Status of Project 3 grading – hopefully very soon.
- Final Exam
  - 8:00-11:00AM, December 18th
  - Bechtel Auditorium
  - Bring 2 sheets of notes, double-sided
  - All lectures – except today (this is a freebie!)

Realtime OS/Embedded Applications

- Embedded applications:
  - Limited Hardware
  - Dedicated to some particular task
  - Examples: 50-100 CPUs in modern car!
- What does it mean to be “Realtime”?
  - Meeting time-related goals in the real world
    - » For instance: to show video, need to display X frames/sec
  - Hard real-time task:
    - » one which we must meet its deadline
    - » otherwise, fatal damage or error will occur.
  - Soft real-time task:
    - » one which we should meet its deadline, but not mandatory.
    - » We should schedule it even if the deadline has passed
  - Determinism:
    - » Sometimes, deterministic behavior is more important than high performance

ManyCore and Realtime

- Realtime OS Details
  - Realtime scheduler looks at deadlines to decide who to schedule next
    - » Example: schedule the thread whose deadline is next
  - What makes it hard to perform realtime scheduling:
    - » Too many background tasks
    - » Optimizing for overall responsiveness or throughput is different from meeting explicit deadlines
- Why are Realtime apps often handled by embedded processors?
  - Because they are dedicated and more predictable
  - Idea: Only need to meet throughput requirements
    - » Might as well slow down processor (via lower voltage) as long as performance criteria met
    - » Power reduces as V^2!
- ManyCore
  - Opportunity to devote cores to realtime activities
  - "Bare metal" partitions: best of realtime and general OSs in one chip…!

Achieving Responsiveness & Agility in Tessellation

- Place time-critical components in their own partition
  - E.g.: User Interface Components, Jitter-critical applications
  - User-level scheduler tuned for deadline scheduling
- Grouping of external events to handle in next partition time slice
  - Achieving regularity (low standard deviation of behavior) more important than lowest latency for many types of real-time scheduling
  - Removes interrupt overhead (replaces it with polling)
- Pre-compose partition configurations
  - Quick start of partitions in response to I/O events or real-time triggers
- Judicious use of Speculation
  - Basic variant of the checkpointing mechanism to fork execution
  - When long-latency operations intervene, generate speculative partition
    - » Can track speculative state through different partitions/processes/etc
    - » Can be use to improve I/O speed, interaction with services, etc

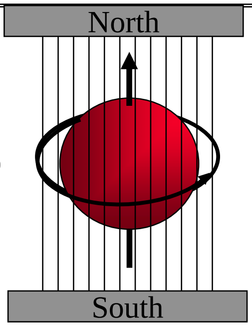
---

Quantum Computing

Can we Use Quantum Mechanics to Compute?

- Weird properties of quantum mechanics?
  - Quantization: Only certain values or orbits are good
    - » Remember orbitals from chemistry???
  - Superposition: Schizophrenic physical elements don't quite know whether they are one thing or another
- All existing digital abstractions try to eliminate QM
  - Transistors/Gates designed with classical behavior
  - Binary abstraction: a "1" is a "1" and a "0" is a "0"
- Quantum Computing:
  Use of Quantization and Superposition to compute.
- Interesting results:
  - Shor's algorithm: factors in polynomial time!
  - Grover's algorithm: Finds items in unsorted database in time proportional to square-root of n.

Quantization: Use of "Spin"



Spin ½ particle:
(Proton/Electron)

Representation:
|0> or |1>

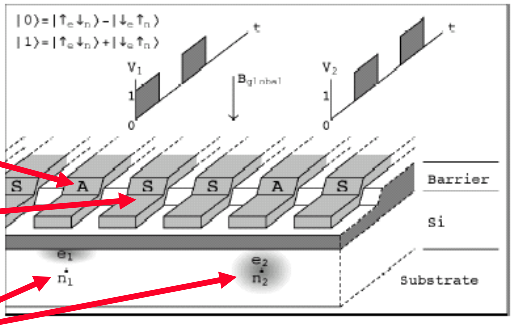
- Particles like Protons have an intrinsic "Spin" when defined with respect to an external magnetic field
- Quantum effect gives "1" and "0":
  - Either spin is "UP" or "DOWN" nothing between

Kane Proposal II (First one didn't quite work)

Single Spin Control Gates

Inter-bit Control Gates

Phosphorus Impurity Atoms



- Bits Represented by combination of proton/electron spin
- Operations performed by manipulating control gates
  - Complex sequences of pulses perform NMR-like operations
- Temperature < 1° Kelvin!

Now add Superposition!

- The bit can be in a combination of "1" and "0":
  - Written as: $\Psi = C_0|0\rangle + C_1|1\rangle$
  - The C's are *complex numbers!*
  - Important Constraint: $|C_0|^2 + |C_1|^2 = 1$
- If *measure* bit to see what looks like,
  - With probability $|C_0|^2$ we will find $|0\rangle$ (say "UP")
  - With probability $|C_1|^2$ we will find $|1\rangle$ (say "DOWN")
- Is this a real effect? Options:
  - This is just statistical – given a large number of protons, a fraction of them ($|C_0|^2$) are "UP" and the rest are down.
  - This is a real effect, and the proton is really both things until you try to look at it
- Reality: second choice!
  - There are experiments to prove it!

Implications: A register can have many values

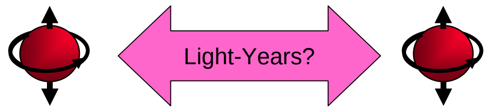
- Implications of superposition:
  - An *n*-bit register can have 2^n values simultaneously!
  - 3-bit example:

$$\Psi = C_{000}|000\rangle + C_{001}|001\rangle + C_{010}|010\rangle + C_{011}|011\rangle + C_{100}|100\rangle + C_{101}|101\rangle + C_{110}|110\rangle + C_{111}|111\rangle$$

- Probabilities of measuring all bits are set by coefficients:
  - So, prob of getting $|000\rangle$ is $|C_{000}|^2$, etc.
  - Suppose we measure only one bit (first):
    - We get "0" with probability: $P_0 = |C_{000}|^2 + |C_{001}|^2 + |C_{010}|^2 + |C_{011}|^2$
      Result: $\Psi = (C_{000}|000\rangle + C_{001}|001\rangle + C_{010}|010\rangle + C_{011}|011\rangle)$
    - We get "1" with probability: $P_1 = |C_{100}|^2 + |C_{101}|^2 + |C_{110}|^2 + |C_{111}|^2$
      Result: $\Psi = (C_{100}|100\rangle + C_{101}|101\rangle + C_{110}|110\rangle + C_{111}|111\rangle)$
- Problem: Don't want environment to *measure* before ready!
  - Solution: Quantum Error Correction Codes!

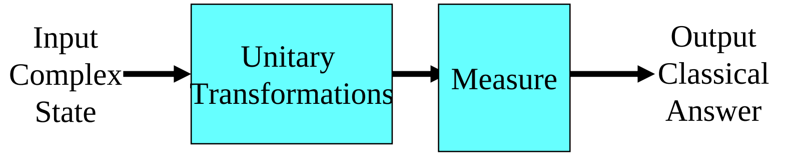
Spooky action at a distance

- Consider the following simple 2-bit state:
  $\Psi = C_{00}|00\rangle + C_{11}|11\rangle$
  - Called an "EPR" pair for "Einstein, Podolsky, Rosen"
- Now, separate the two bits:



- If we measure one of them, it instantaneously sets other one!
  - Einstein called this a "spooky action at a distance"
  - In particular, if we measure a |0> at one side, we get a |0> at the other (and vice versa)
- Teleportation
  - Can "pre-transport" an EPR pair (say bits X and Y)
  - Later to transport bit A from one side to the other we:
    - » Perform operation between A and X, yielding two classical bits
    - » Send the two bits to the other side
    - » Use the two bits to operate on Y
    - » Poof! State of bit A appears in place of Y

Model? Operations on coefficients + measurements



- Basic Computing Paradigm:
  - Input is a register with superposition of many values
    - » Possibly all 2^n inputs equally probable!
  - Unitary transformations compute on coefficients
    - » Must maintain probability property (sum of squares = 1)
    - » Looks like doing computation on all 2^n inputs simultaneously!
  - Output is one result attained by measurement
- If do this poorly, just like probabilistic computation:
  - If 2^n inputs equally probable, may be 2^n outputs equally probable.
  - After measure, like picked random input to classical function!
  - All interesting results have some form of "fourier transform" computation being done in unitary transformation

---

Security of Factoring

- The Security of RSA Public-key cryptosystems depends on the difficult of factoring a number N=pq (product of two primes)
  - Classical computer: sub-exponential time factoring
  - Quantum computer: polynomial time factoring
- Shor's Factoring Algorithm (for a quantum computer)

Easy 1) Choose random x : $2 \leq x \leq N-1$.

Easy 2) If gcd(x,N) ≠ 1, Bingo!

Hard 3) Find smallest integer r : $x^r \equiv 1 \pmod{N}$

Easy 4) If r is odd, GOTO 1

Easy 5) If r is even, $a = x^{r/2} \pmod{N} \Rightarrow (a-1)\times(a+1) = kN$

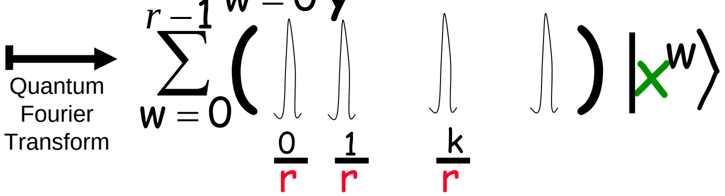
Easy 6) If $a = N-1$ GOTO 1

Easy 7) ELSE gcd($a \pm 1$,N) is a non trivial factor of N.

Shor's Factoring Algorithm

$$\sum_{k}|k\rangle|1\rangle \rightarrow \sum_{k}|k\rangle|x^{k}\rangle$$

$$= \sum_{w=0}^{r-1}\sum_{y}|w + ry\rangle|x^{w}\rangle$$

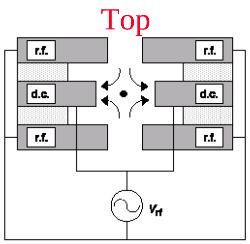


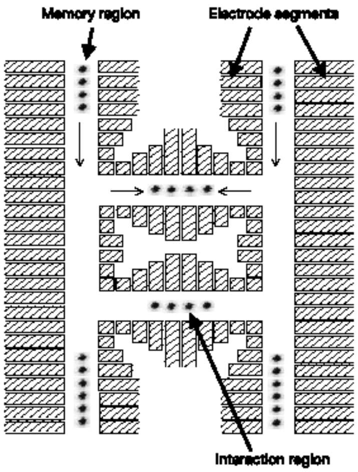
- Finally: Perform measurement
  - Find out r with high probability
  - Get $|y\rangle|a^{w'}\rangle$ where y is of form k/r and w' is related

Some Issues in building quantum computer

- What are the bits and how do we manipulate them?
  - NMR computation: use "cup of liquid".
    - Use nuclear spins (special protons on complex molecules).
    - Manipulate with radio-frequencies
    - IBM Has produced a 7-bit computer
  - Silicon options (more scalable)
    - Impurity Phosphorus in silicon
    - Manipulate through electrons (including measurement)
    - Still serious noise/fabrication issues
  - Other options:
    - Optical (Phases of photons represent bits)
    - Single ions trapped in magnetic fields
- How do we prevent the environment from "Measuring"?
  - Make spins as insulated from environment as possible
  - Quantum Error Correction!
- Where get "clean" bits (I.e. unsuperposed |0> or |1>)?
  - Entropy exchange unit:
    - Radiates heat to environment (entropy)
    - Produces clean bits (COLD) to enter into device

ION Trap Quantum Computer: Promising technology





Top View

Proposal: NIST Group

- IONS of Be+ trapped in oscillating quadrature field
  - Internal electronic modes of IONS used for quantum bits
  - MEMs technology
  - Target? 50,000 ions
  - ROOM Temperature!
- Ions moved to interaction regions
  - Ions interactions with one another moderated by lasers

Conclusions

- Spatial Partitioning: grouping processors and resources behind hardware boundary
  - Two-level scheduling
    1)Global Distribution of resources
    2)Application-Specific scheduling of resources
  - Bare Metal Execution within partition
  - Distributed systems view of OS design
- Tessellation OS: ParLAB's new OS
  - Exploded, spatially partitioned, interacting services
- Quantum Computing
  - Using interesting properties of physics to compute
- Berkely PARLAb
  - Check out: view.eecs.berkeley.edu
    parlab.eecs.berkeley.edu
- Let's give a hand to the TAs!

Good Bye!