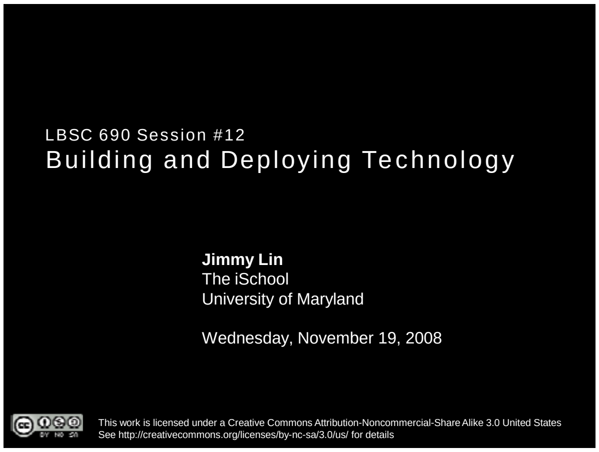
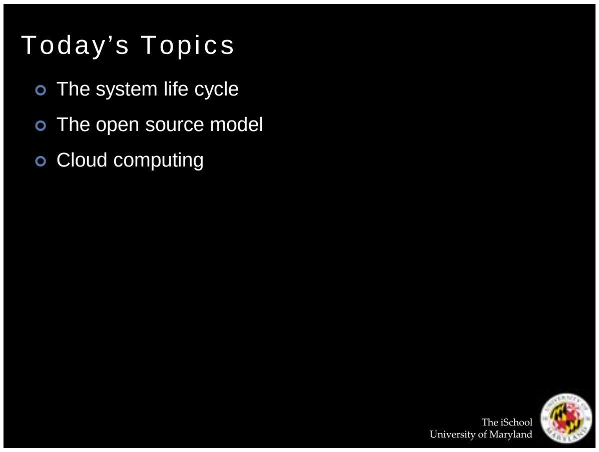

LBSC 690 Session #12

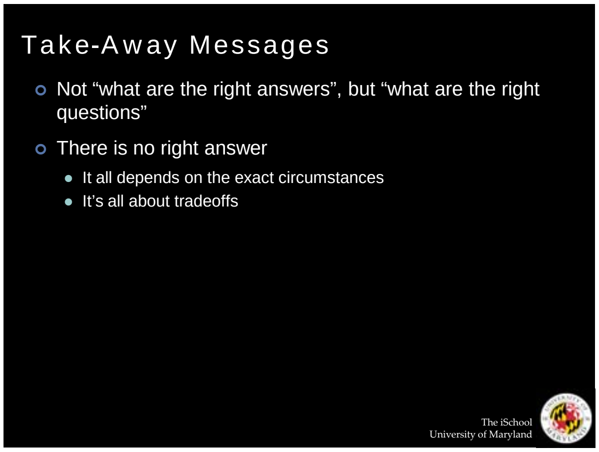
# Building and Deploying Technology

**Jimmy Lin**
The iSchool
University of Maryland

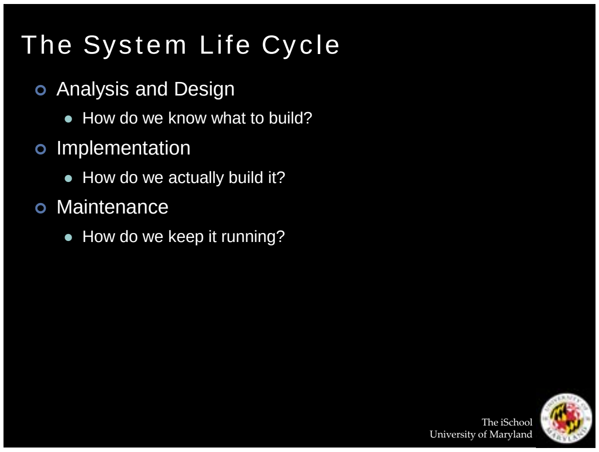
Wednesday, November 19, 2008


## Today's Topics

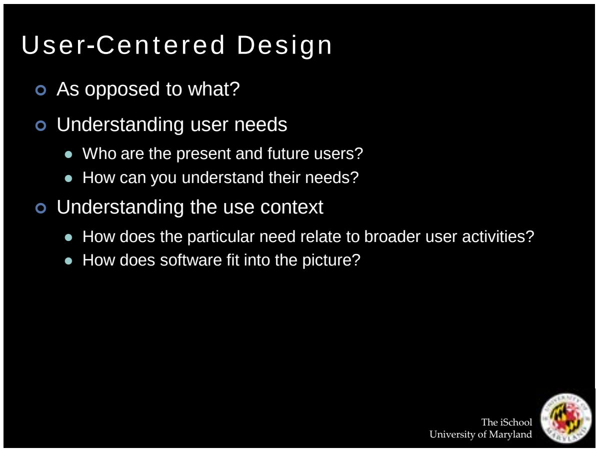
- The system life cycle
- The open source model
- Cloud computing


## Take-Away Messages

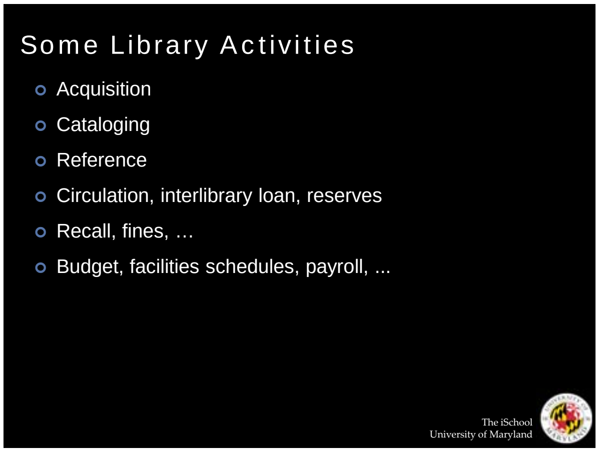
- Not "what are the right answers", but "what are the right questions"
- There is no right answer
  - It all depends on the exact circumstances
  - It's all about tradeoffs


## The System Life Cycle

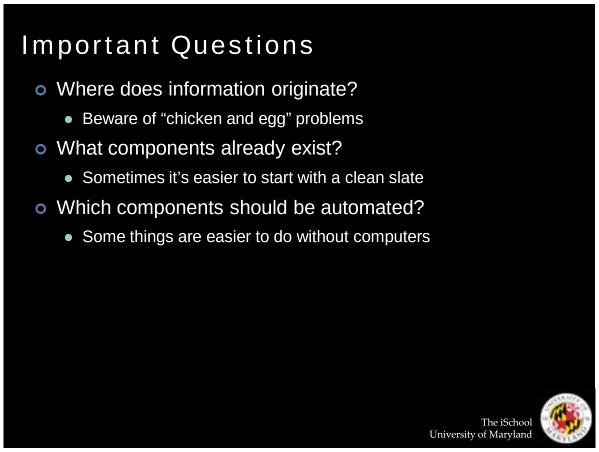
- Analysis and Design
  - How do we know what to build?
- Implementation
  - How do we actually build it?
- Maintenance
  - How do we keep it running?


## User-Centered Design

- As opposed to what?
- Understanding user needs
  - Who are the present and future users?
  - How can you understand their needs?
- Understanding the use context
  - How does the particular need relate to broader user activities?
  - How does software fit into the picture?


## Some Library Activities

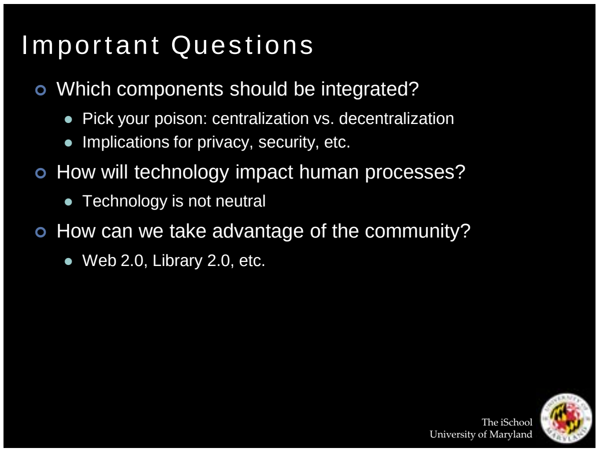
- Acquisition
- Cataloging
- Reference
- Circulation, interlibrary loan, reserves
- Recall, fines, …
- Budget, facilities schedules, payroll, ...



## Important Questions

- Where does information originate?
  - Beware of “chicken and egg” problems
- What components already exist?
  - Sometimes it’s easier to start with a clean slate
- Which components should be automated?
  - Some things are easier to do without computers

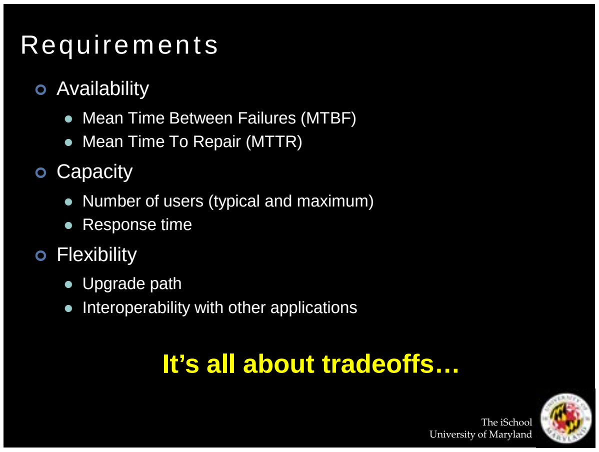
The iSchool
University of Maryland



## Important Questions

- Which components should be integrated?
  - Pick your poison: centralization vs. decentralization
  - Implications for privacy, security, etc.
- How will technology impact human processes?
  - Technology is not neutral
- How can we take advantage of the community?
  - Web 2.0, Library 2.0, etc.

The iSchool
University of Maryland

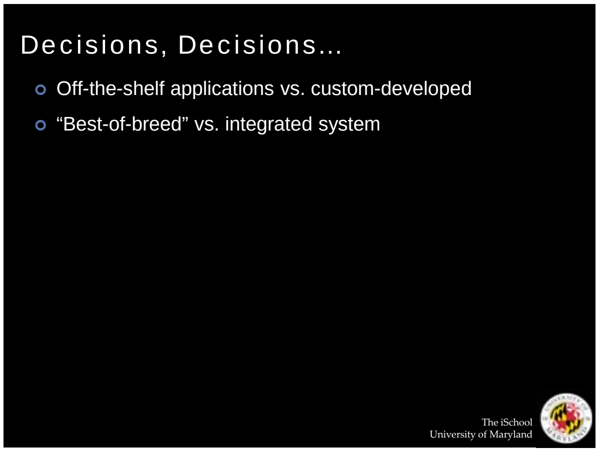


## Requirements

- Availability
  - Mean Time Between Failures (MTBF)
  - Mean Time To Repair (MTTR)
- Capacity
  - Number of users (typical and maximum)
  - Response time
- Flexibility
  - Upgrade path
  - Interoperability with other applications

**It’s all about tradeoffs…**

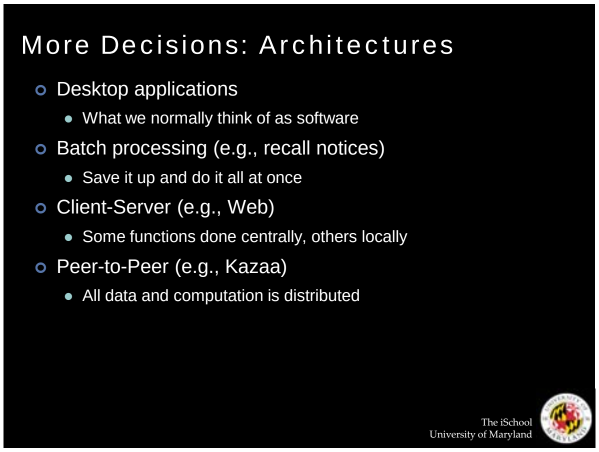
The iSchool
University of Maryland



## Decisions, Decisions…

- Off-the-shelf applications vs. custom-developed
- “Best-of-breed” vs. integrated system

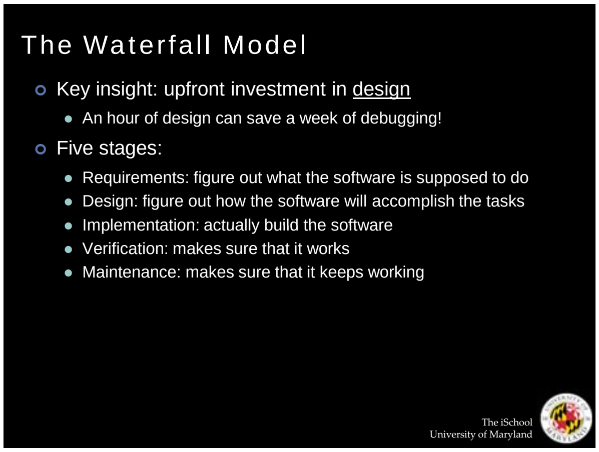
The iSchool
University of Maryland



## More Decisions: Architectures

- Desktop applications
  - What we normally think of as software
- Batch processing (e.g., recall notices)
  - Save it up and do it all at once
- Client-Server (e.g., Web)
  - Some functions done centrally, others locally
- Peer-to-Peer (e.g., Kazaa)
  - All data and computation is distributed

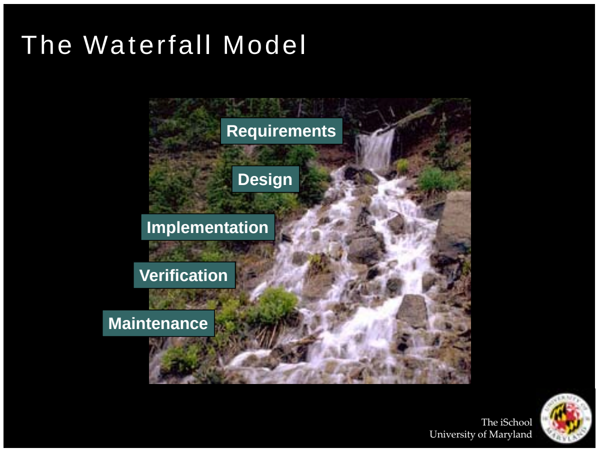
The iSchool
University of Maryland



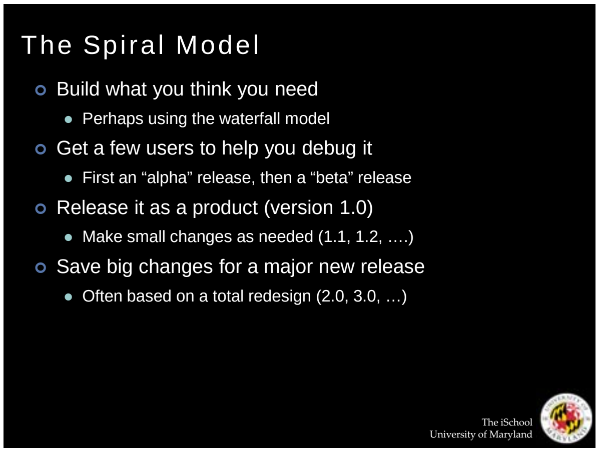
## The Waterfall Model

- Key insight: upfront investment in design
  - An hour of design can save a week of debugging!
- Five stages:
  - Requirements: figure out what the software is supposed to do
  - Design: figure out how the software will accomplish the tasks
  - Implementation: actually build the software
  - Verification: makes sure that it works
  - Maintenance: makes sure that it keeps working

The iSchool
University of Maryland

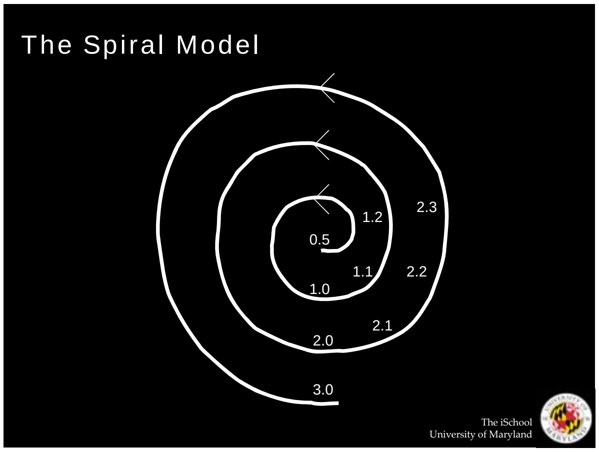


## The Waterfall Model

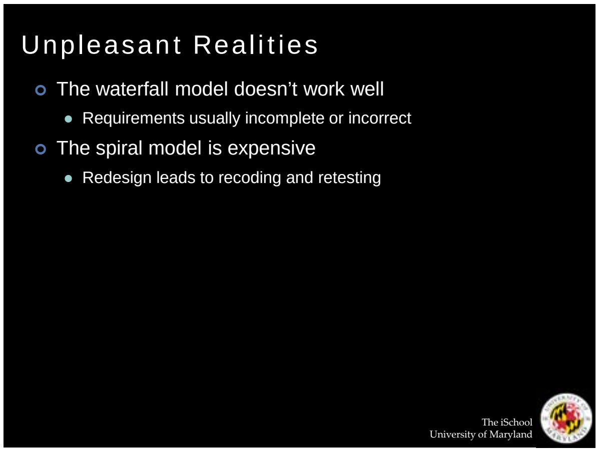


## The Spiral Model

- Build what you think you need
  - Perhaps using the waterfall model
- Get a few users to help you debug it
  - First an "alpha" release, then a "beta" release
- Release it as a product (version 1.0)
  - Make small changes as needed (1.1, 1.2, ….)
- Save big changes for a major new release
  - Often based on a total redesign (2.0, 3.0, …)

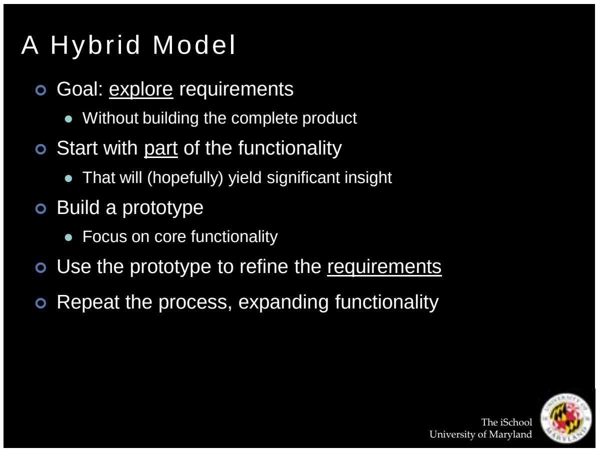


## The Spiral Model



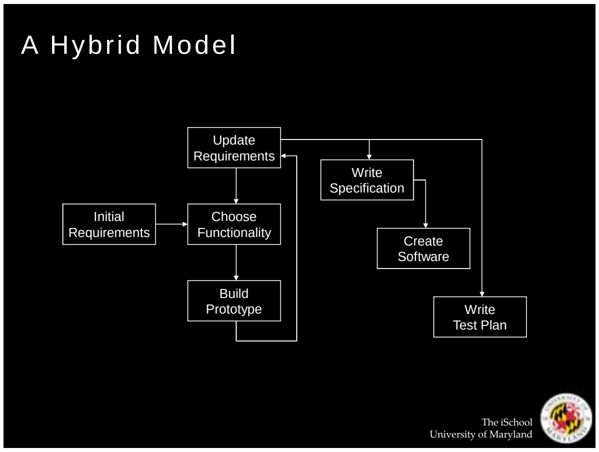
## Unpleasant Realities

- The waterfall model doesn't work well
  - Requirements usually incomplete or incorrect
- The spiral model is expensive
  - Redesign leads to recoding and retesting



## A Hybrid Model

- Goal: <u>explore</u> requirements
  - Without building the complete product
- Start with <u>part</u> of the functionality
  - That will (hopefully) yield significant insight
- Build a prototype
  - Focus on core functionality
- Use the prototype to refine the <u>requirements</u>
- Repeat the process, expanding functionality



## A Hybrid Model

Update Requirements

Write Specification

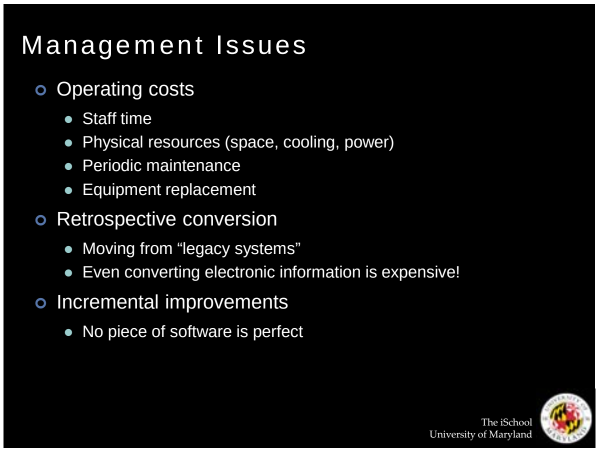
Initial Requirements

Choose Functionality

Create Software

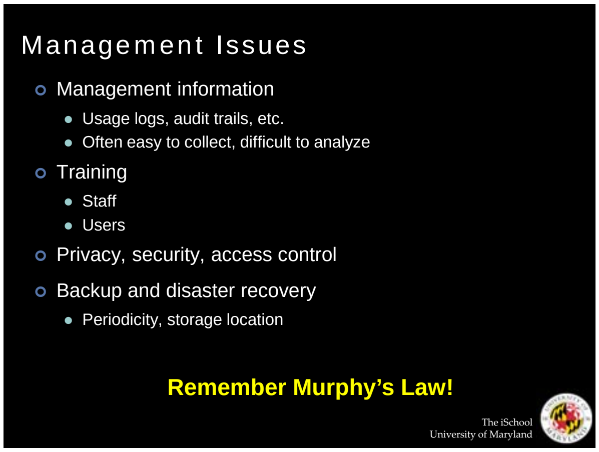
Build Prototype

Write Test Plan



## Management Issues

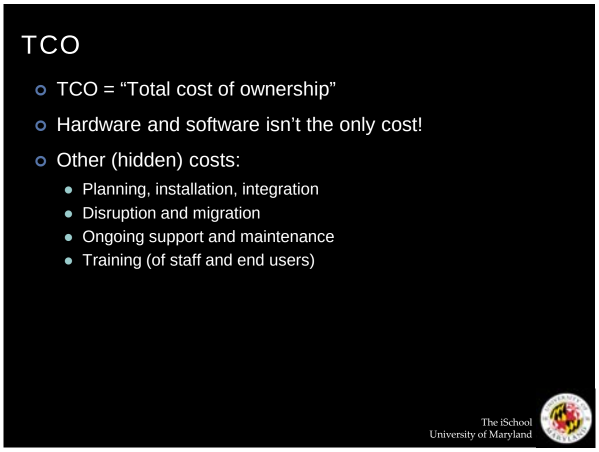
- Operating costs
  - Staff time
  - Physical resources (space, cooling, power)
  - Periodic maintenance
  - Equipment replacement
- Retrospective conversion
  - Moving from “legacy systems”
  - Even converting electronic information is expensive!
- Incremental improvements
  - No piece of software is perfect



## Management Issues

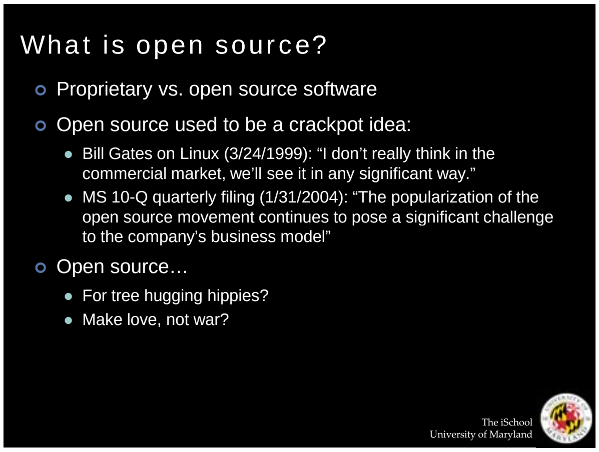
- Management information
  - Usage logs, audit trails, etc.
  - Often easy to collect, difficult to analyze
- Training
  - Staff
  - Users
- Privacy, security, access control
- Backup and disaster recovery
  - Periodicity, storage location

**Remember Murphy’s Law!**



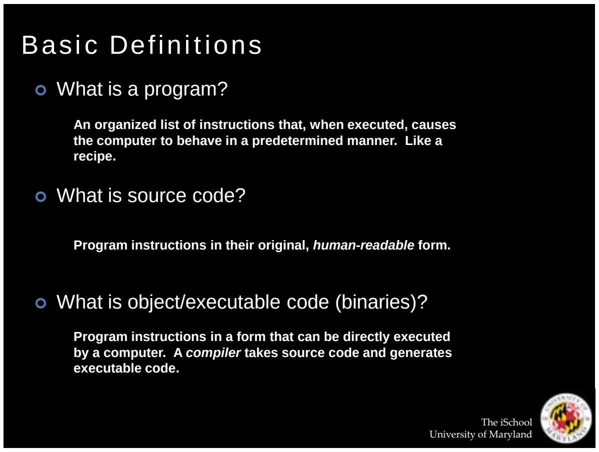
## TCO

- TCO = “Total cost of ownership”
- Hardware and software isn’t the only cost!
- Other (hidden) costs:
  - Planning, installation, integration
  - Disruption and migration
  - Ongoing support and maintenance
  - Training (of staff and end users)



## What is open source?

- Proprietary vs. open source software
- Open source used to be a crackpot idea:
  - Bill Gates on Linux (3/24/1999): “I don’t really think in the commercial market, we’ll see it in any significant way.”
  - MS 10-Q quarterly filing (1/31/2004): “The popularization of the open source movement continues to pose a significant challenge to the company’s business model”
- Open source…
  - For tree hugging hippies?
  - Make love, not war?



## Basic Definitions

- What is a program?

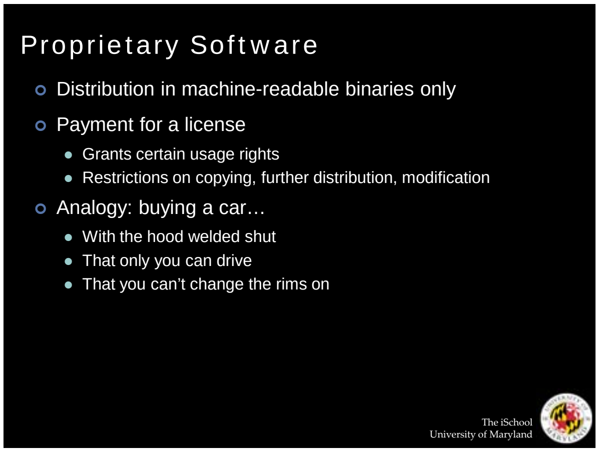
  **An organized list of instructions that, when executed, causes the computer to behave in a predetermined manner. Like a recipe.**

- What is source code?

  **Program instructions in their original, *human-readable* form.**

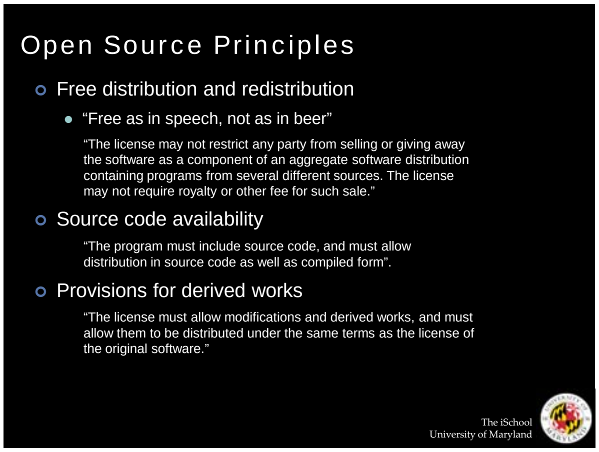
- What is object/executable code (binaries)?

  **Program instructions in a form that can be directly executed by a computer. A *compiler* takes source code and generates executable code.**



## Proprietary Software

- Distribution in machine-readable binaries only
- Payment for a license
  - Grants certain usage rights
  - Restrictions on copying, further distribution, modification
- Analogy: buying a car…
  - With the hood welded shut
  - That only you can drive
  - That you can’t change the rims on



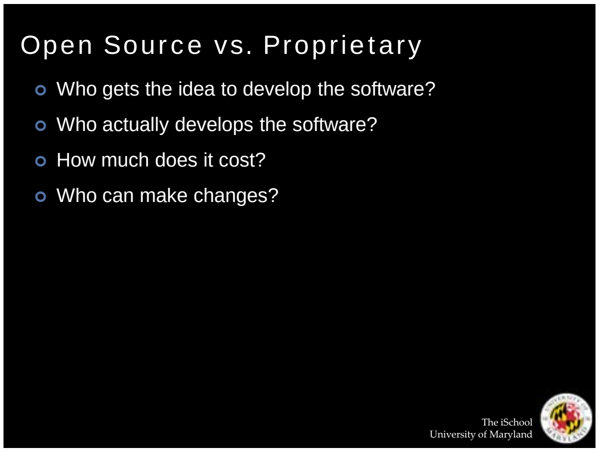
## Open Source Principles

- Free distribution and redistribution
  - "Free as in speech, not as in beer"

    "The license may not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license may not require royalty or other fee for such sale."
- Source code availability

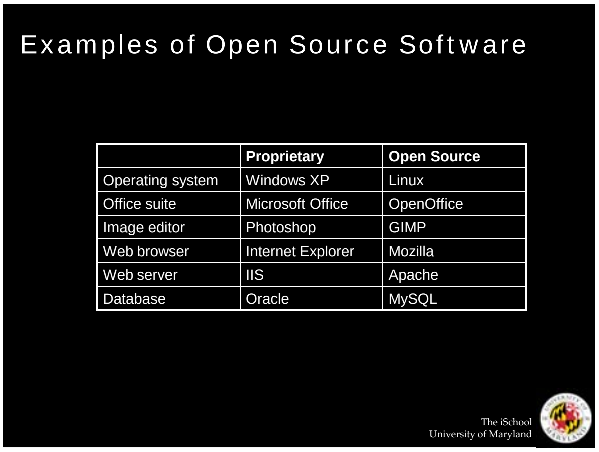
  "The program must include source code, and must allow distribution in source code as well as compiled form".
- Provisions for derived works

  "The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software."



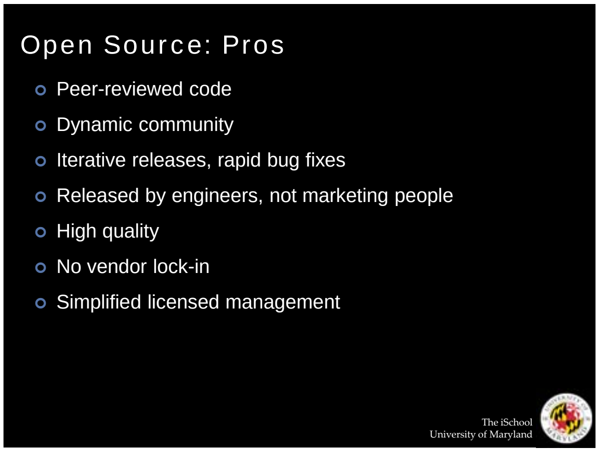
## Open Source vs. Proprietary

- Who gets the idea to develop the software?
- Who actually develops the software?
- How much does it cost?
- Who can make changes?



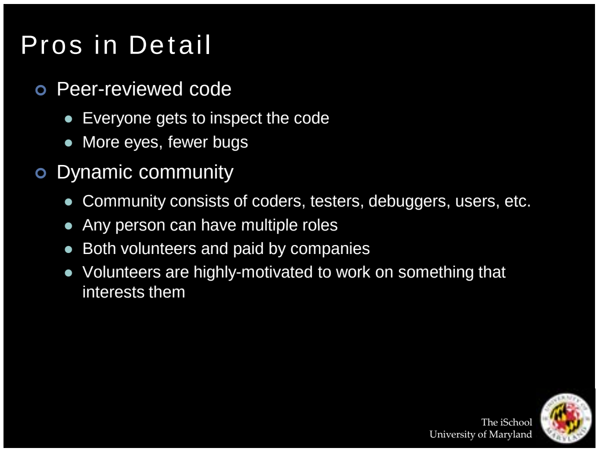
## Examples of Open Source Software

| | Proprietary | Open Source |
|---|---|---|
| Operating system | Windows XP | Linux |
| Office suite | Microsoft Office | OpenOffice |
| Image editor | Photoshop | GIMP |
| Web browser | Internet Explorer | Mozilla |
| Web server | IIS | Apache |
| Database | Oracle | MySQL |



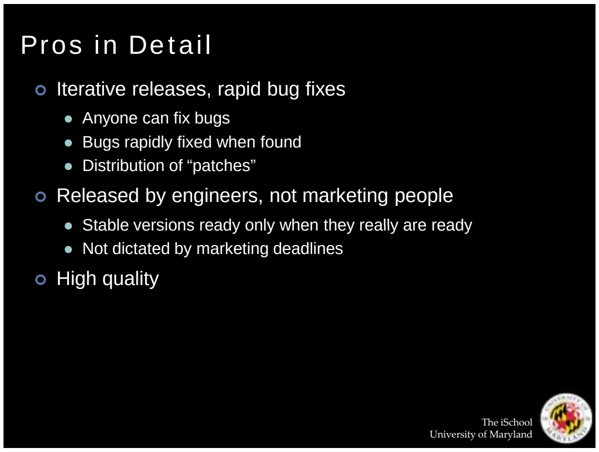
## Open Source: Pros

- Peer-reviewed code
- Dynamic community
- Iterative releases, rapid bug fixes
- Released by engineers, not marketing people
- High quality
- No vendor lock-in
- Simplified licensed management



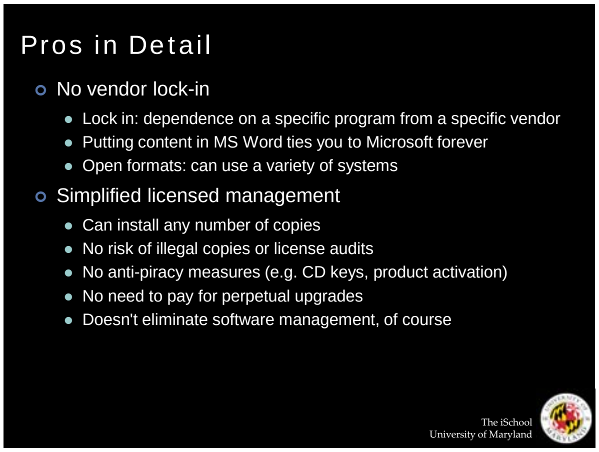
## Pros in Detail

- Peer-reviewed code
  - Everyone gets to inspect the code
  - More eyes, fewer bugs
- Dynamic community
  - Community consists of coders, testers, debuggers, users, etc.
  - Any person can have multiple roles
  - Both volunteers and paid by companies
  - Volunteers are highly-motivated to work on something that interests them



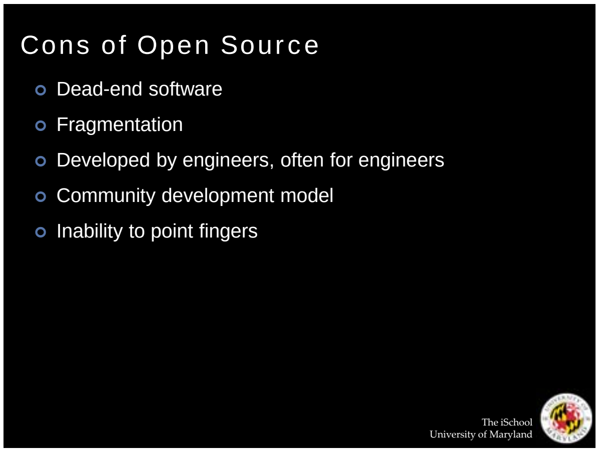
## Pros in Detail

- Iterative releases, rapid bug fixes
  - Anyone can fix bugs
  - Bugs rapidly fixed when found
  - Distribution of "patches"
- Released by engineers, not marketing people
  - Stable versions ready only when they really are ready
  - Not dictated by marketing deadlines
- High quality



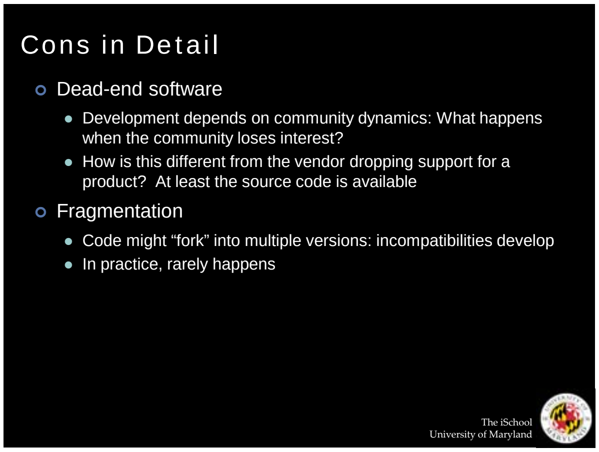
## Pros in Detail

- No vendor lock-in
  - Lock in: dependence on a specific program from a specific vendor
  - Putting content in MS Word ties you to Microsoft forever
  - Open formats: can use a variety of systems
- Simplified licensed management
  - Can install any number of copies
  - No risk of illegal copies or license audits
  - No anti-piracy measures (e.g. CD keys, product activation)
  - No need to pay for perpetual upgrades
  - Doesn't eliminate software management, of course



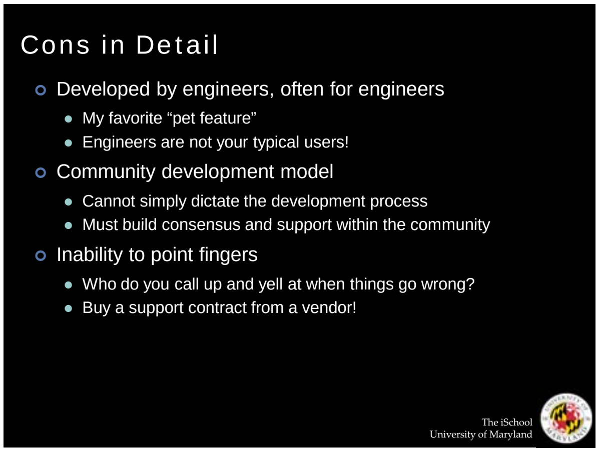
## Cons of Open Source

- Dead-end software
- Fragmentation
- Developed by engineers, often for engineers
- Community development model
- Inability to point fingers



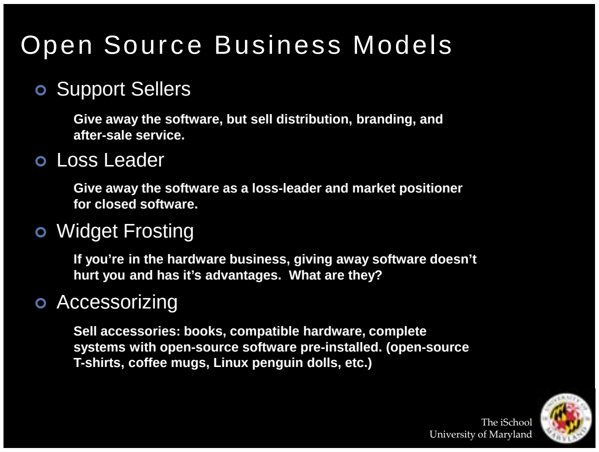
## Cons in Detail

- Dead-end software
  - Development depends on community dynamics: What happens when the community loses interest?
  - How is this different from the vendor dropping support for a product? At least the source code is available
- Fragmentation
  - Code might “fork” into multiple versions: incompatibilities develop
  - In practice, rarely happens



## Cons in Detail

- Developed by engineers, often for engineers
  - My favorite “pet feature”
  - Engineers are not your typical users!
- Community development model
  - Cannot simply dictate the development process
  - Must build consensus and support within the community
- Inability to point fingers
  - Who do you call up and yell at when things go wrong?
  - Buy a support contract from a vendor!



## Open Source Business Models

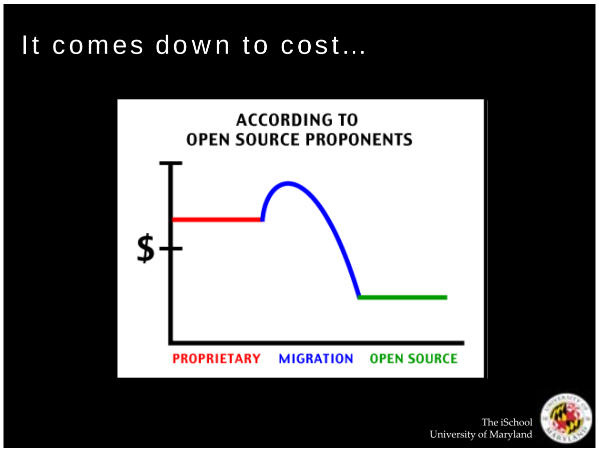
- Support Sellers

  **Give away the software, but sell distribution, branding, and after-sale service.**
- Loss Leader

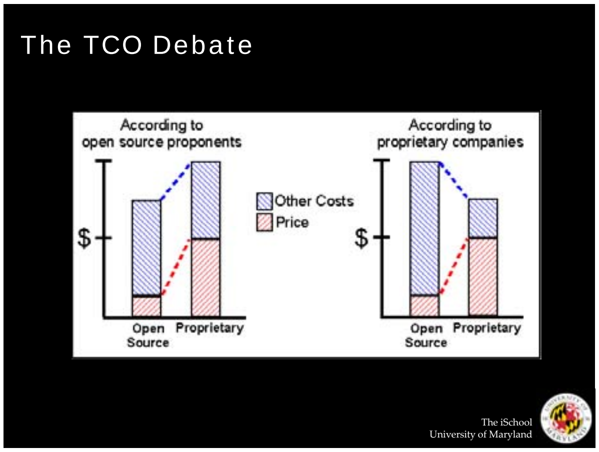
  **Give away the software as a loss-leader and market positioner for closed software.**
- Widget Frosting

  **If you’re in the hardware business, giving away software doesn’t hurt you and has it’s advantages. What are they?**
- Accessorizing

  **Sell accessories: books, compatible hardware, complete systems with open-source software pre-installed. (open-source T-shirts, coffee mugs, Linux penguin dolls, etc.)**

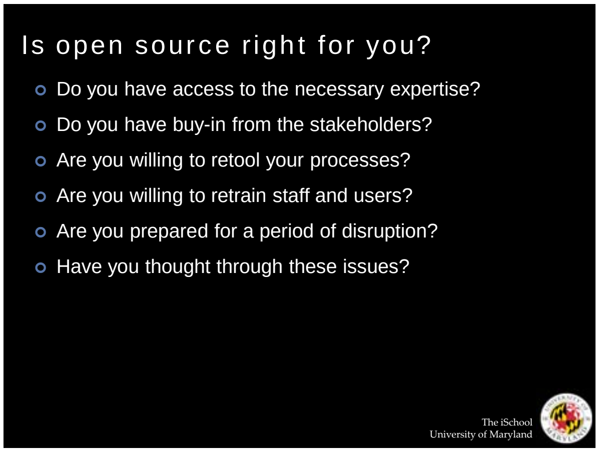


## It comes down to cost…



## The TCO Debate

The iSchool
University of Maryland

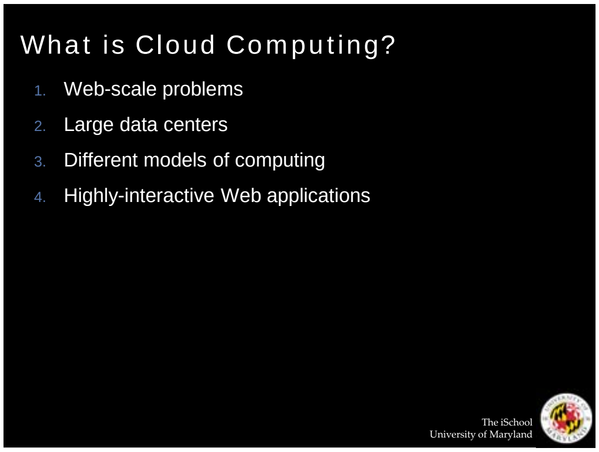


## Is open source right for you?

- Do you have access to the necessary expertise?
- Do you have buy-in from the stakeholders?
- Are you willing to retool your processes?
- Are you willing to retrain staff and users?
- Are you prepared for a period of disruption?
- Have you thought through these issues?

The iSchool
University of Maryland

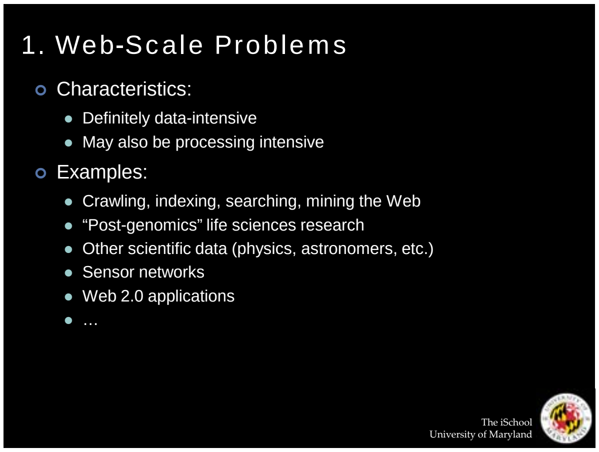
Source: http://www.free-pictures-photos.com/



## What is Cloud Computing?

1. Web-scale problems
2. Large data centers
3. Different models of computing
4. Highly-interactive Web applications

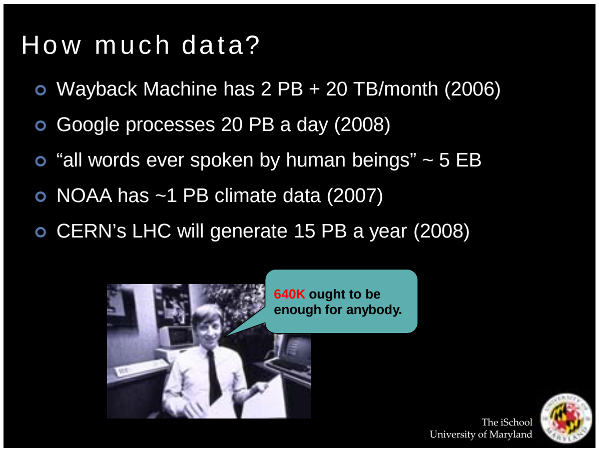
The iSchool
University of Maryland



## 1. Web-Scale Problems

- Characteristics:
  - Definitely data-intensive
  - May also be processing intensive
- Examples:
  - Crawling, indexing, searching, mining the Web
  - “Post-genomics” life sciences research
  - Other scientific data (physics, astronomers, etc.)
  - Sensor networks
  - Web 2.0 applications
  - …

The iSchool
University of Maryland

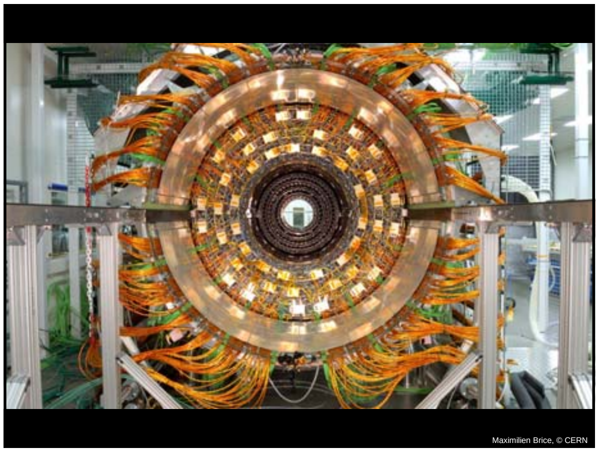


## How much data?

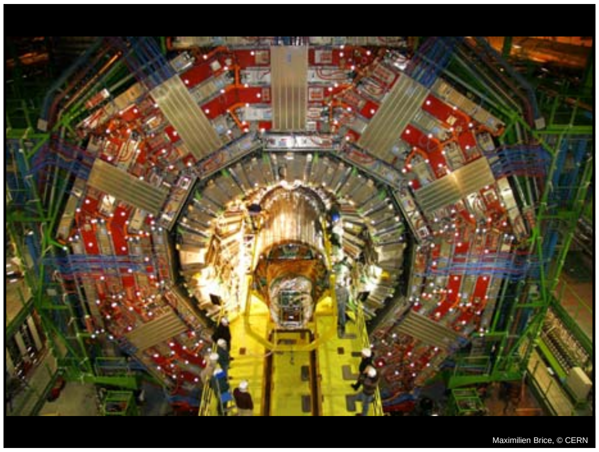
- Wayback Machine has 2 PB + 20 TB/month (2006)
- Google processes 20 PB a day (2008)
- “all words ever spoken by human beings” ~ 5 EB
- NOAA has ~1 PB climate data (2007)
- CERN’s LHC will generate 15 PB a year (2008)

640K ought to be enough for anybody.

The iSchool
University of Maryland



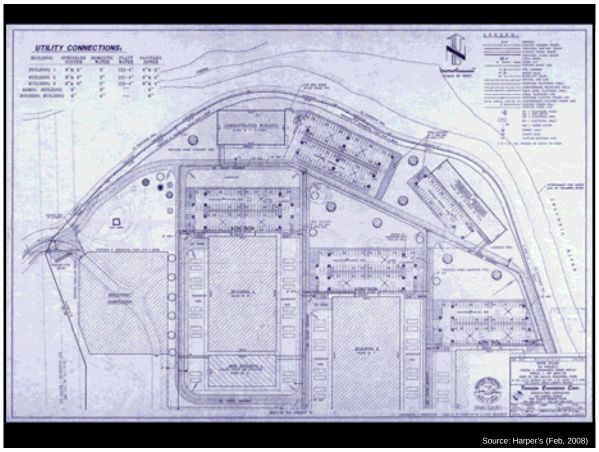
Maximilien Brice, © CERN



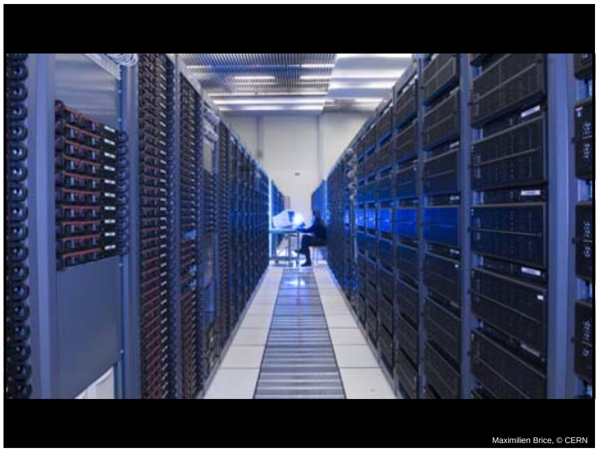
Maximilien Brice, © CERN

## 2. Large Data Centers

- Web-scale problems? Throw more machines at it!
- Clear trend: centralization of computing resources in large data centers
  - Necessary ingredients: fiber, juice, and space
  - What do Oregon, Iceland, and abandoned mines have in common?
- Important Issues:
  - Redundancy
  - Efficiency
  - Utilization
  - Management

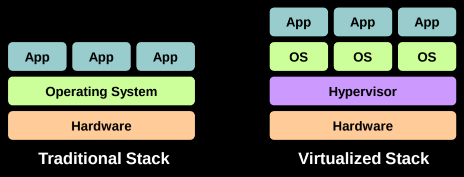


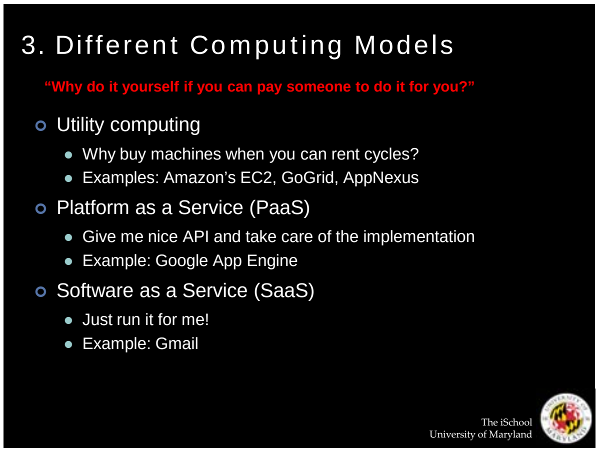
Source: Harper's (Feb, 2008)



Maximilien Brice, © CERN

## Key Technology: Virtualization



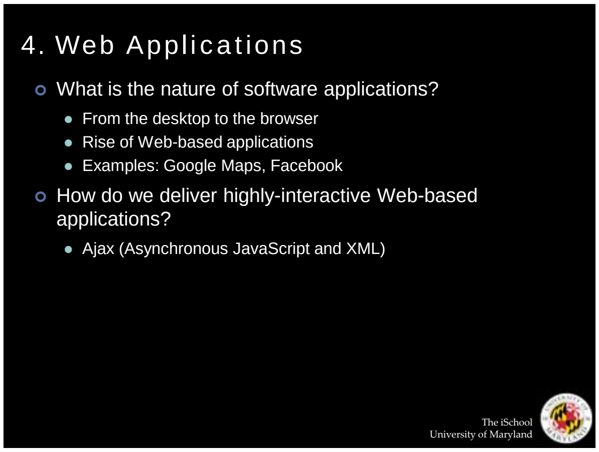


## 3. Different Computing Models

"Why do it yourself if you can pay someone to do it for you?"

- Utility computing
  - Why buy machines when you can rent cycles?
  - Examples: Amazon's EC2, GoGrid, AppNexus
- Platform as a Service (PaaS)
  - Give me nice API and take care of the implementation
  - Example: Google App Engine
- Software as a Service (SaaS)
  - Just run it for me!
  - Example: Gmail

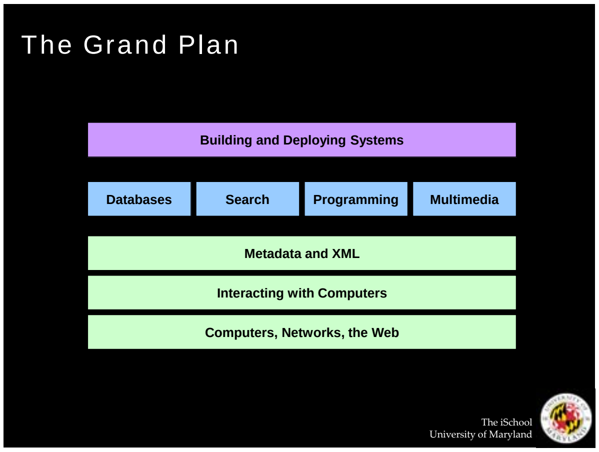
The iSchool
University of Maryland



## 4. Web Applications

- What is the nature of software applications?
  - From the desktop to the browser
  - Rise of Web-based applications
  - Examples: Google Maps, Facebook
- How do we deliver highly-interactive Web-based applications?
  - Ajax (Asynchronous JavaScript and XML)

The iSchool
University of Maryland



## The Grand Plan

Building and Deploying Systems

Databases | Search | Programming | Multimedia

Metadata and XML

Interacting with Computers

Computers, Networks, the Web

The iSchool
University of Maryland