CS 6410: ADVANCED SYSTEMS
KEN BIRMAN

Fall 2015

A PhD-oriented course about research in systems

About me…

- My research is focused on "high assurance"
  - In fact as a graduate student I was torn between machine learning in medicine and distributed systems
  - I've ended up working mostly in systems, on topics involving fault-tolerance, consistency, coordination, security and other kinds of high-assurance
- My current hot topics?
  - Cloud-scale high assurance via platform and language support (often using some form of machine learning)
  - Using the cloud to monitor/control the smart power grid
- … but CS6410 is much broader than just "Ken stuff"

Goals for Today

- What is CS6410 "about"?
  - What will be covered, and what background is assumed?
  - Why take this course?
  - How does this class operate?
  - Class details

- Non-goal: We won't have a real lecture today
  - This is because our lectures are always tied to readings

Coverage

- The course is about the cutting edge in computer systems – the topics that people at conferences like ACM Symposium on Operating Systems Principles (SOSP) and the Usenix Conference on Operating Systems Design and Implementation (OSDI) love
- We look at a mix of topics:
  - Classic insights and classic systems that taught us a great deal or that distilled key findings into useable platform technologies
  - Fundamental (applied theory) side of these questions
  - New topics that have people excited right now

Lots of work required

- First and foremost: Attend every class, participate
  - You'll need to do a lot of reading.
  - You'll write a short (1-2 page) summary of the papers each time
  - Whoever presents the paper that day grades these ($\surd$-, $\surd$, $\surd$+)
  - You can skip up to 5 of them, whenever you like. Hand in "I'm skipping this one" and the grader will record that. But not more than 5.
- You'll have two "homework assignments" during first six weeks
  - Build (from scratch) a parallel version of the game of life designed to extract maximum speed from a multicore processor (2 is fine, 12 would be awesome)
  - Distributed coordination service running on EC2 (use a preexisting version of Paxos, and access it via Elastic Beanstalk). Study to identify bottlenecks, but no need to change the version of Paxos we provide
- Then will do a more substantial semester-long independent project
- Most students volunteer to present a paper. Not required but useful

Takeway?

- You could probably take one other class too
- But if you have any desire to have any kind of life at all, plus to begin to explore a research area, you can't take more than two classes like this!
- Not so much that it is "hard" (by and large, systems isn't about hard ideas so much as challenging engineering), but it definitely takes time

Systems: Three "arcs

- In the e

Risk: Cool theory but impractical result
…eployed . Sometimes
…odel is unrealistic!

Advantage: At massive scale your intuition breaks down. Just doing …!

…and proofs

Risk: Totally unprincipled spaghetti

…ly how

SOSP

Build/evaluate a research prototype

Prove stuff abou… something

SOC…

Report on amazing industry successes

- Today, these lines are more and more separated
- Some people get emotional over which is best!

Background: Ken's stuff

I'm obsessed with reliable, super-fast data replication and applications that use that model.

But I try not to let it show…

My work blends theory and building

- This isn't unusual, many projects overlap lines
- But it also moves me out of the mainstream SOSP community: I'm more of a "distributed systems" researcher than a "core systems" researcher
- My main interest: *How should theories of consistency and fault-tolerance inform the design of high-assurance applications and platforms?*

Questions this poses

- Which theory to use? We have more than one theoretical network model (synchronous, asynchronous, stochastic) and they differ in their "power"
- How to translate this to a provably sound systems construct and to embed that into a platform (we use a model shared with Lamport's Paxos system)
- Having done all that, how to make the resulting system scale to run on the cloud, perform absolutely as fast as possible, exhibit stability... how to make it "natural" to use and easy to work with...

Current passion: my new Isis^2 System

- C# library (but callable from any .NET language) offering replication techniques for cloud computing developers
- Based on a model that fuses virtual synchrony and state machine replication models
- Research challenges center on creating protocols that function well despite cloud "events"

- **Elasticity (sudden scale changes)**
- **Potentially heavily loads**
- **High node failure rates**
- **Concurrent (multithreaded) apps**

- **Long scheduling delays, resource contention**
- **Bursts of message loss**
- **Need for very rapid response times**
- **Community skeptical of "assurance properties"**

Isis^2 makes developer's life easier

Benefits of Using Formal model

- Formal model permits us to achieve correctness
- Isis^2 is too complex to use formal methods as a development too, but does facilitate debugging (model checking)
- Think of Isis^2 as a collection of modules, each with rigorously stated properties

Importance of Sound Engineering

- Isis^2 implementation needs to be fast, lean, easy to use
- Developer must see it as easier to use Isis^2 than to build from scratch
- Seek great performance under "cloudy conditions"
- Forced to anticipate many styles of use

Isis^2 makes developer's life easier


```
Group g = new Group("myGroup");
g.ViewHandlers += delegate(View v) {
      Console.Title = "myGroup members: "+v.members;
};
g.Handlers[UPDATE] += delegate(string s, double v) {
      Values[s] = v;
};
g.Handlers[LOOKUP] += delegate(string s) {
       Reply(Values[s]);
};
g.Join();

g.Send(UPDATE, "Harry", 20.75);

List<double> resultlist = new List<double>;
nr = g.Query(LOOKUP, ALL, "Harry", EOL, resultlist);
```

- First sets up group
- Join makes this entity a member. State transfer isn't shown
- Then can multicast, query. Runtime callbacks to the "delegates" as events arrive
- Easy to request security (g.SetSecure), persistence
- "Consistency" model dictates the ordering aseen for event upcalls and the assumptions user can make

Isis^2 makes developer's life easier


```
Group g = new Group("myGroup");
g.ViewHandlers += delegate(View v) {
      Console.Title = "myGroup members: "+v.members;
};
g.Handlers[UPDATE] += delegate(string s, double v) {
      Values[s] = v;
};
g.Handlers[LOOKUP] += delegate(string s) {
       Reply(Values[s]);
};
g.Join();

g.Send(UPDATE, "Harry", 20.75);

List<double> resultlist = new List<double>;
nr = g.Query(LOOKUP, ALL, "Harry", EOL, resultlist);
```

- **First sets up group**
- Join makes this entity a member. State transfer isn't shown
- Then can multicast, query. Runtime callbacks to the "delegates" as events arrive
- Easy to request security (g.SetSecure), persistence
- "Consistency" model dictates the ordering seen for event upcalls and the assumptions user can make

Isis^2 makes developer's life easier


```
Group g = new Group("myGroup");
g.ViewHandlers += delegate(View v) {
      Console.Title = "myGroup members: "+v.members;
};
g.Handlers[UPDATE] += delegate(string s, double v) {
      Values[s] = v;
};
g.Handlers[LOOKUP] += delegate(string s) {
       Reply(Values[s]);
};
g.Join();

g.Send(UPDATE, "Harry", 20.75);

List<double> resultlist = new List<double>;
nr = g.Query(LOOKUP, ALL, "Harry", EOL, resultlist);
```

- First sets up group
- **Join makes this entity a member. State transfer isn't shown**
- Then can multicast, query. Runtime callbacks to the "delegates" as events arrive
- Easy to request security (g.SetSecure), persistence
- "Consistency" model dictates the ordering seen for event upcalls and the assumptions user can make

Isis^2 makes developer's life easier


```
Group g = new Group("myGroup");
g.ViewHandlers += delegate(View v) {
      Console.Title = "myGroup members: "+v.members;
};
g.Handlers[UPDATE] += delegate(string s, double v) {
      Values[s] = v;
};
g.Handlers[LOOKUP] += delegate(string s) {
       Reply(Values[s]);
};
g.Join();

g.Send(UPDATE, "Harry", 20.75);

List<double> resultlist = new List<double>;
nr = g.Query(LOOKUP, ALL, "Harry", EOL, resultlist);
```

- First sets up group
- Join makes this entity a member. State transfer isn't shown
- **Then can multicast, query. Runtime callbacks to the "delegates" as events arrive**
- Easy to request security (g.SetSecure), persistence
- "Consistency" model dictates the ordering seen for event upcalls and the assumptions user can make

Isis^2 makes developer's life easier


```
Group g = new Group("myGroup");
g.ViewHandlers += delegate(View v) {
      Console.Title = "myGroup members: "+v.members;
};
g.Handlers[UPDATE] += delegate(string s, double v) {
      Values[s] = v;
};
g.Handlers[LOOKUP] += delegate(string s) {
       Reply(Values[s]);
};
g.Join();

g.Send(UPDATE, "Harry", 20.75);

List<double> resultlist = new List<double>;
nr = g.Query(LOOKUP, ALL, "Harry", EOL, resultlist);
```

- First sets up group
- Join makes this entity a member. State transfer isn't shown
- **Then can multicast, query. Runtime callbacks to the "delegates" as events arrive**
- Easy to request security (g.SetSecure), persistence
- "Consistency" model dictates the ordering seen for event upcalls and the assumptions user can make

Isis^2 makes developer's life easier


```
Group g = new Group("myGroup");
g.ViewHandlers += delegate(View v) {
      Console.Title = "myGroup members: "+v.members;
};
g.Handlers[UPDATE] += delegate(string s, double v) {
      Values[s] = v;
};
g.Handlers[LOOKUP] += delegate(string s) {
       Reply(Values[s]);
};
g.Join();

g.Send(UPDATE, "Harry", 20.75);

List<double> resultlist = new List<double>;
nr = g.Query(LOOKUP, ALL, "Harry", EOL, resultlist);
```

- First sets up group
- Join makes this entity a member. State transfer isn't shown
- Then can multicast, query. Runtime callbacks to the "delegates" as events arrive
- **Easy to request security (g.SetSecure), persistence**
- **"Consistency" model dictates the ordering seen for event upcalls and the assumptions user can make**

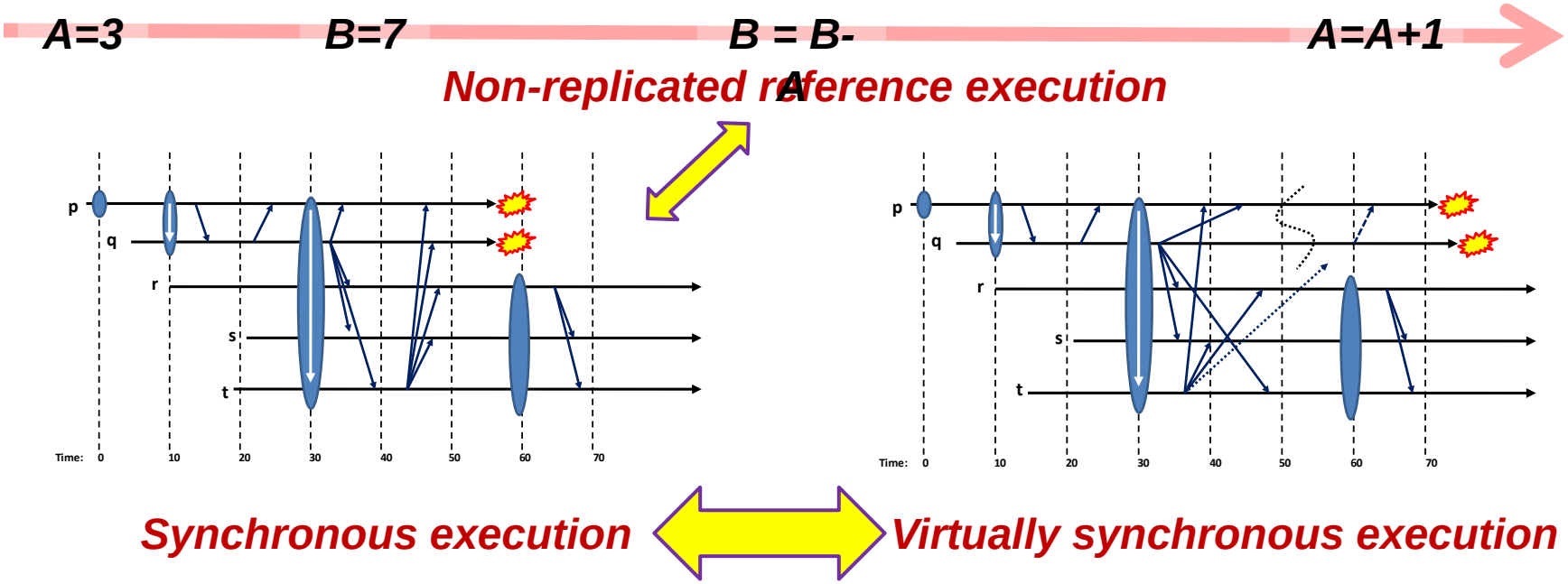
Isis^2 makes developer's life easier


```
Group g = new Group("myGroup");
g.ViewHandlers += delegate(View v) {
      Console.Title = "myGroup members: "+v.members;
};
g.Handlers[UPDATE] += delegate(string s, double v) {
      Values[s] = v;
};
g.Handlers[LOOKUP] += delegate(string s) {
       Reply(Values[s]);
};
g.SetSecure();
g.Join();

g.Send(UPDATE, "Harry", 20.75);

List<double> resultlist = new List<double>;
nr = g.Query(LOOKUP, ALL, "Harry", EOL, resultlist);
```

- First sets up group
- Join makes this entity a member. State transfer isn't shown
- Then can multicast, query. Runtime callbacks to the "delegates" as events arrive
- **Easy to request security** (g.SetSecure), persistence
- "Consistency" model dictates the ordering seen for event upcalls and the assumptions user can make

Consitency model: Virtual synchrony meets Paxos (and they live happily ever after…)

- **Virtual synchrony is a "consistency" model:**
  - ***Membership epochs:*** *begin when a new configuration is installed and reported by delivery of a new "view" and associated state*
  - ***Protocols run "during" a single epoch:*** *rather than overcome failure, we reconfigure when a failure occurs*



How wo

We build the group as the system runs. Each participant just adds itself.

The leader monitors membership. This particular version doesn't handle failures but the "full" version is easy.

We can trust the membership. Even failure notifications reflect a system-wide consensus.

```
Group g = new G
g.ViewHandle  += dele
    IMPORT "db-replica:"+v.GetMyRank();
};
g.Handlers[UPDATE] += delegate(string s, double v)
{
    START TRANSACTION;
    UPDATE salary = v WHERE SET
    COMMIT;
};
...

g.SafeSend(UPDATE, "Harry", "85
```

1. **Modify the view handler to bind to the appropriate replicate (db-replica:0)**

Paxos guarantees agreement on message set, the order in which to perform actions and durability if any.

This code requires that mySQL is deterministic and that the serialization order won't be changed by QUERY operations (read-only, but they might get locks). As it happens, those assumptions are valid.

Example Application: Smart Power Grid

- Today's electric power grid is aging, inefficient
  - Some studies suggest that as much as 65% is wasted
  - Poor job of integrating renewable energy (wind, solar)

- Two distinct forms of smart grid (both matter):
  - Deploy sensors in the power grid itself (PMUs), plus switchable lines and other new technologies
    - Some like to think of this as a new power grid "network"
    - But keep in mind that power doesn't flow in packets
  - Smart meters in the home: controlled demand/response

No time for all of it today…

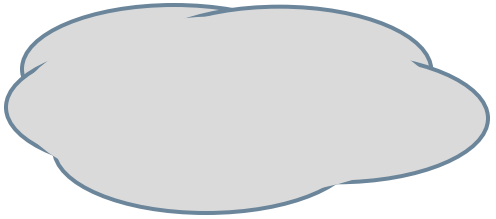
- Focus will be on the "bulk" power network
  - And within this, on capturing and computing on PMU data in real-time
- Assumptions
  - Human operators will be part of the equation for a long time into the future
  - Later could extend into the power distribution network and explore ways of automating certain control tasks

Cloud Computing for the Smart Grid

- Real-time collection of data from widely deployed PMU and other SCADA data sources
  - PMU = Synchronized **P**hasor **M**easurement **U**nit
    - Each PMU device captures 44 byte records at 30Hz
    - One per "bus": Data rates within large regions high
- Robust real-time tracking enables shared, consistent situational awareness and coordination

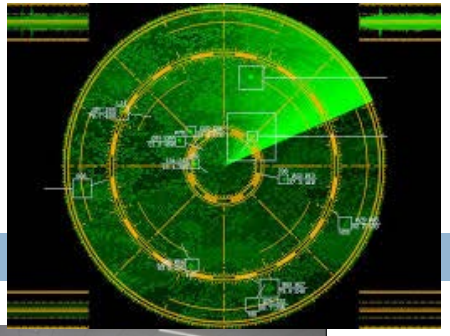
Cloud Computing for the Smart Grid

- Core premise: Use the cloud!



- By reusing today's scalable cloud infrastructure, we:
  - Benefit from a low-cost solution
  - Leverage a proven, universally accessible technology
  - The cloud is hosted at geographically diverse places
- But our need is for stronger assurance than the cloud can normally offer

Killer Applications?



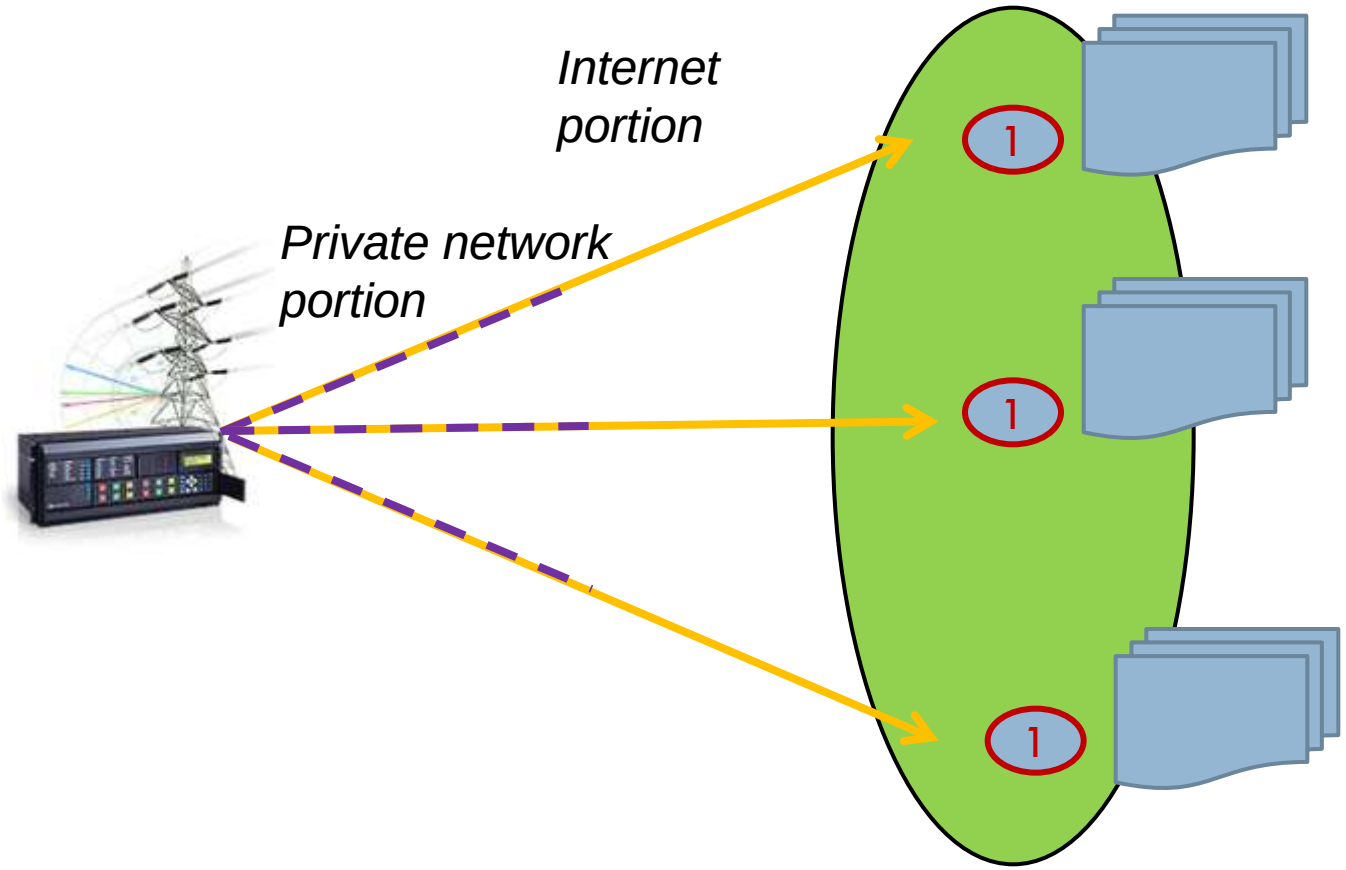
- Over the horizon "grid radar" helps operators understand wide-area grid stress, disturbances

- Tools ("apps for the smart grid") help operators cooperate to solve problems, search knowledge base for past situations with similar fingerprint, explore what-if scenarios

Does the Cloud Have an Achilles Heel?

- Today's cloud is optimized for applications with weak security needs. It offers scalable snappy response, but lacks robust guarantees. Lacks:
  - Hardened network protocols aimed at consistent but tightly controlled sharing for collaboration
  - A new distributed security model supporting total control by regional operator, controlled data flows
- To leverage the cloud, we need a new smart-grid technology built within today's cloud technology!

From the sensor to the shard

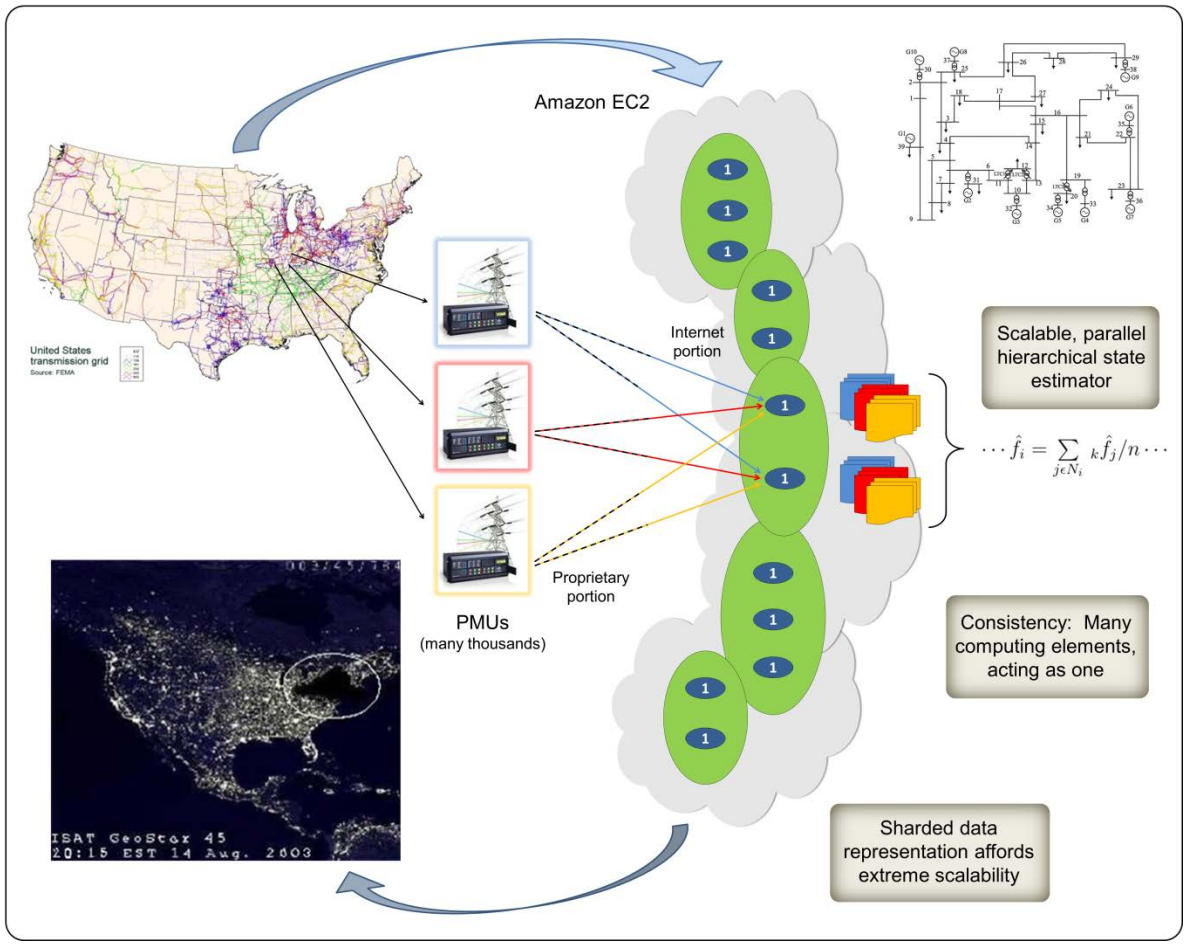


The shard members keep logs of values received indexed by time.

Due to network delay, not all have the same data at the same time.

Transport could be via GridStat, IP multicast, Isis^2 multicast (which runs on IP multicast) or even TCP connections

GridCloud: Mile high perspective



Definitions

PMU

A sensor (synchrophasor) used to measure voltage and phase angle of a power bus

State Estimator (SE)

Code that computes the state of a regional grid using PMU data as input

Cornell University


Ken Birman


Robbert van Renesse


Ketan Maheshwari

Washington State University


Dave Bakken


Anjan Bose


Carl Hauser

www.cs.cornell.edu/Projects/Gridcontrol/

Prepared for the 2013 ARPA-E Energy Innovation Summit


Cornell University

WASHINGTON STATE UNIVERSITY
World Class. Face to Face.

Deployed for Collaboration

GridCloud
Core Technologies

Highly Assured Cloud Computing Technology

sponsored by the Department of Energy ARPA-E program

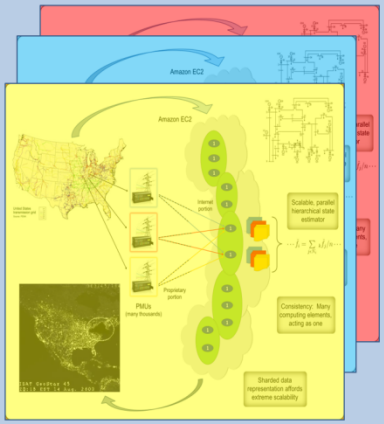
Cornell University

Why Clouds?

- Cost effective: pay only for resources you are using, amortize infrastructure over many users
- Geographic scale: multiple data centers at widely separated locations gives physical reliability
- Scalable capacity: potential to do real-time tracking of PMU data at national scale

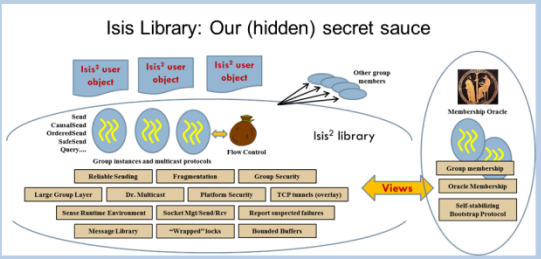
What Makes it Hard?

- Today's cloud is inadequately secure and has poor real-time guarantees
- At scale with many moving parts, transient and permanent faults are common, and rare events occur surprisingly often
- We need a computing model that matches the reality: multiple operators
- We need to find scalable ways to compute state estimates rapidly and robustly
- Even if power industry runs the cloud, demands new trust and auditing approaches



Application Layer

Real-Time State Estimation enabling a wide range of new operator-oriented functionality and the potential for direct control of sensitive tasks

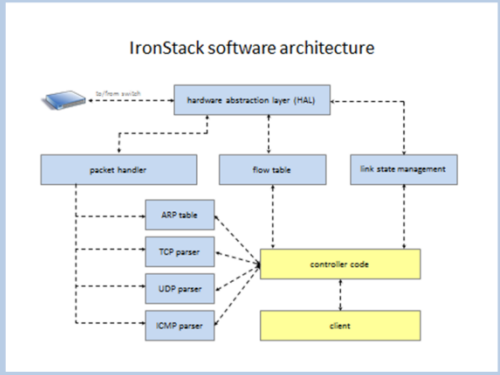


Performance targets?

- **15,000 or more PMUs or other sensor devices monitored at 30Hz**
- **Nationwide physical scale**
- **30 State estimates per second with 250ms delay**
- **Delays 10x smaller in smaller regional setups**
- **Instant and automated recovery from faults. Geographic replication to handle major outages.**

Status?

- **GridCloud is working! Demos at steadily increasing scale (but using simulated data, and Amazon EC2).**



Three key techniques

- Redundancy / Replication
- Consistent monitoring and management
- Software defined network with real-time guarantees

Tools

- **Isis2:** A DARPA funded Cornell-developed toolkit for building highly assured cloud computing solutions. Aims at programmers.
- **DMake:** Based on Isis2, monitors and manages a large, complex system. Aims at a higher level system operator.
- **IronStack:** A new networking package that transforms private networks into highly secure, highly assured real-time network solutions

Future:

- **Powerful operator-oriented visualization and collaboration tools**
- Think of a table-sized tablet with a wide range of "smart" computational elements you can touch/drag/drop

Main Components: GC-FS

- **GridCloud File System:** A file system for secure, strongly consistent real-time mirrored data sharing
  - It spreads data over multiple servers keeping data in memory for fast performance and scalability.
  - The data mirrored can be updated in real-time. (For example files of PMU data).
  - Really cool feature: it can pull up snapshots of past file states – huge numbers of them at very low cost. We plan to use this with Hadoop (MapReduce) applications

Leader developer: Weijia Song. Using .NET FileSystem class + Isis² (Birman)

Main Components: GC-Collab

- **GridCloud Collaboration Tool**: A tool for creating a kind of sharable virtual iPad
  - It graphs the current power network and can show you the status of any line at a click
  - Various "apps" can be dragged onto the network and this triggers actions, like a transient stability analysis or listing "similar network states seen in the past" (we're the framework. Other people build these apps)
  - Shared with real-time consistency as needed

Based on: Live Distributed Objects + Isis^2 (Ostrowski, Birman)

Main Components: IronStack

- **IronStack**: A *software defined network* manager
  - You run normal TCP/IP and UDP protocols over it
  - Guarantees secure, real-time fault-tolerance
  - Only allows data flow according to security policies
  - It encrypts data, sends it redundantly for robustness. Can tolerate multiple failures
  - Operator console optimally schedules repairs after major damage, warns if failures threaten connectivity

Lead developer: Z Teo

Elegant…

Rock Solid

Main Components: DMake

- **DMake**: Manages your GridCloud applications
  - Based on the popular Unix "makefile" concept
  - But generalized to support distributed programs where their operating parameters can be modified at runtime
  - It handles system repair after failures, load balancing, mapping of your computation to the cloud computing nodes, etc
  - Incredibly easy to use.

Lead developer: Theo Gkountouvas, uses Isis²

GridCloud

Cloud-hosted high-assurance system to monitor the electric power grid

sponsored by the Department of Energy ARPA-E program

Goal

Demonstrate a cloud-scale monitoring infrastructure able to host "smart grid" applications: the code that will make the power grid "smart"

Use cases

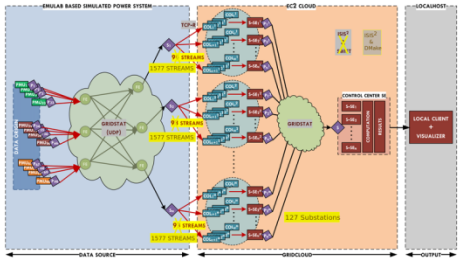
- Routine balancing of loads and generation
- Grid protection
- Analysis and adaptation after topology changes
- Integration of renewable energy

Challenges

Cloud lacks consistency, assurance, and timing guarantees. Industry demands very strong control over data flow with provable security.

Status

We're using Isis^2 to manage a structure in which replicated data permits high assurance reactive smart-grid monitoring and control. GridCloud features state estimation and GridStat software from Washington State University.



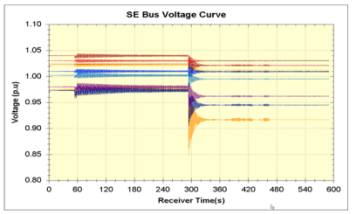
GridCloud enables

Large-scale, distributed power system applications in the cloud

Low-latency and high-reliability using replicated data streams and application processes

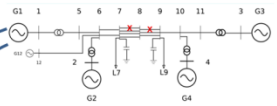
Key GridCloud technologies

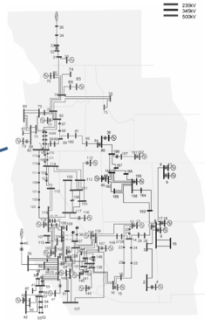
- ISIS and Dmake to manage a large number of processes running in the cloud
- GridStat to provide multi-cast and rate filtering
- Hierarchical Linear State Estimator (HLSE) as example application



The HLSE uses EC2 Cloud resources to quickly and reliably do full-system SE 5 times per second with thousands of PMU data streams as input

	Aggregate Data Rate (Mb/s)	Total Connections	Substation SEs
Mar '13 12-bus (1 replica)	2.92	98	12
12-bus (3 replicas)	**8.75**	**294**	**36**
Oct '13 179-bus (1 replica)	46.93	1,577	127
179-bus (3 replicas)	**140.79**	**4,731**	**381**
Spring '14 6K-bus (3 replicas)	~570	~18,000	~3,000





Upcoming challenge: sourcing the total volume of data: let's discuss whether you can help!

Definitions

PMU

A sensor (synchrophasor) used to measure voltage and phase angle of a power bus

Linear State Estimator (LSE)

Code that computes the state of a power grid using PMU data as input

Cornell University


Ken Birman


Robbert van Renesse


Zhiyuan Teo

Washington State University


Dave Bakken


Anjan Bose


Carl Hauser

www.cs.cornell.edu/Projects/Gridcontrol/

Cornell University

WASHINGTON STATE UNIVERSITY

World Class. Face to Face.

Demonstration Application

- **GridStat State Estimator:**
  - Linear hierarchical state estimator
  - Based on work by Anjan Bose and his team at WSU
  - Runs today on GridCloud prototype, we've scaled it to handle 1900 PMUs distributed nationally
  - Can operate on a private cloud or Amazon EC2
  - Designed to "conceal" failures

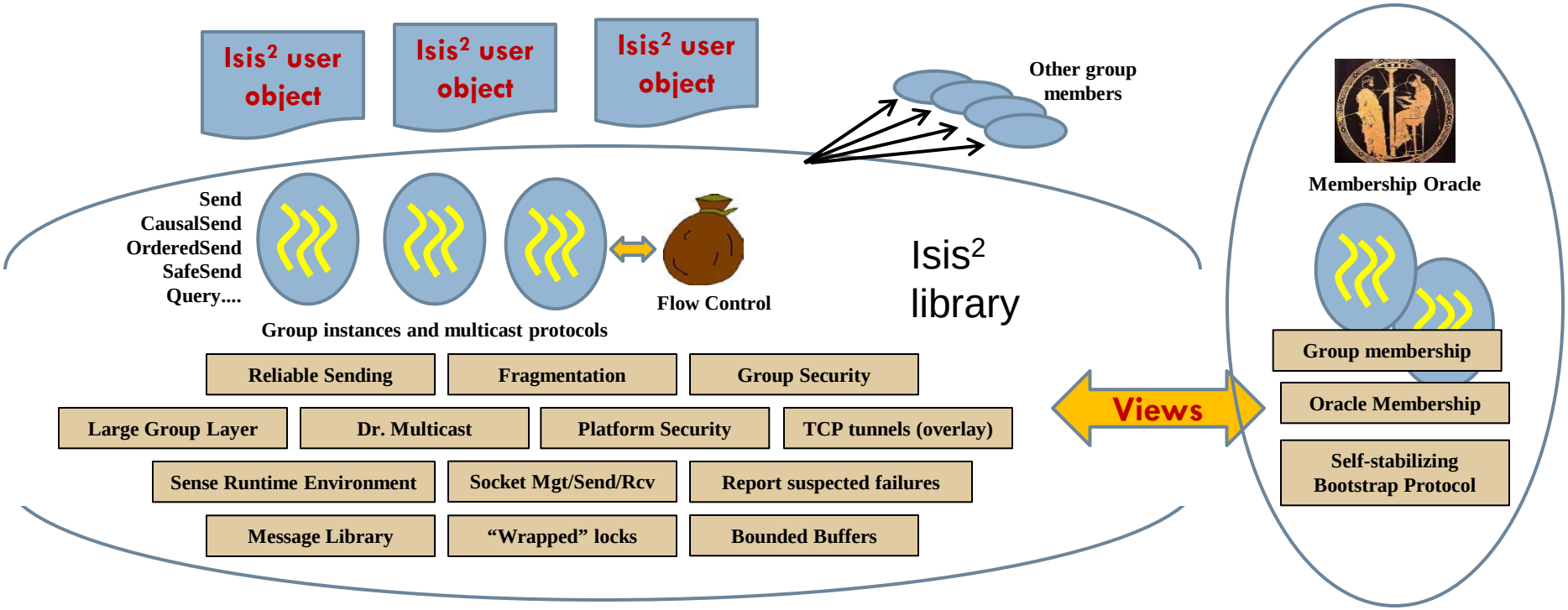
Lead developers: Dave Anderson, Carl Hauser (WSU)

Under the Covers: Powered by Isis^2

- Used internally by these other tools
  - Provides secure, fault-tolerant data replication, coordination and self-repair. Lead: Birman
  - Employs cutting edge "virtual synchrony" programming model (basis of CORBA FT standard)
  - Open source, more than 4000 downloads to date from isis2.codeplex.com

Egyptian myth: After her brother Osiris was torn apart by Seth, Isis restored him to life

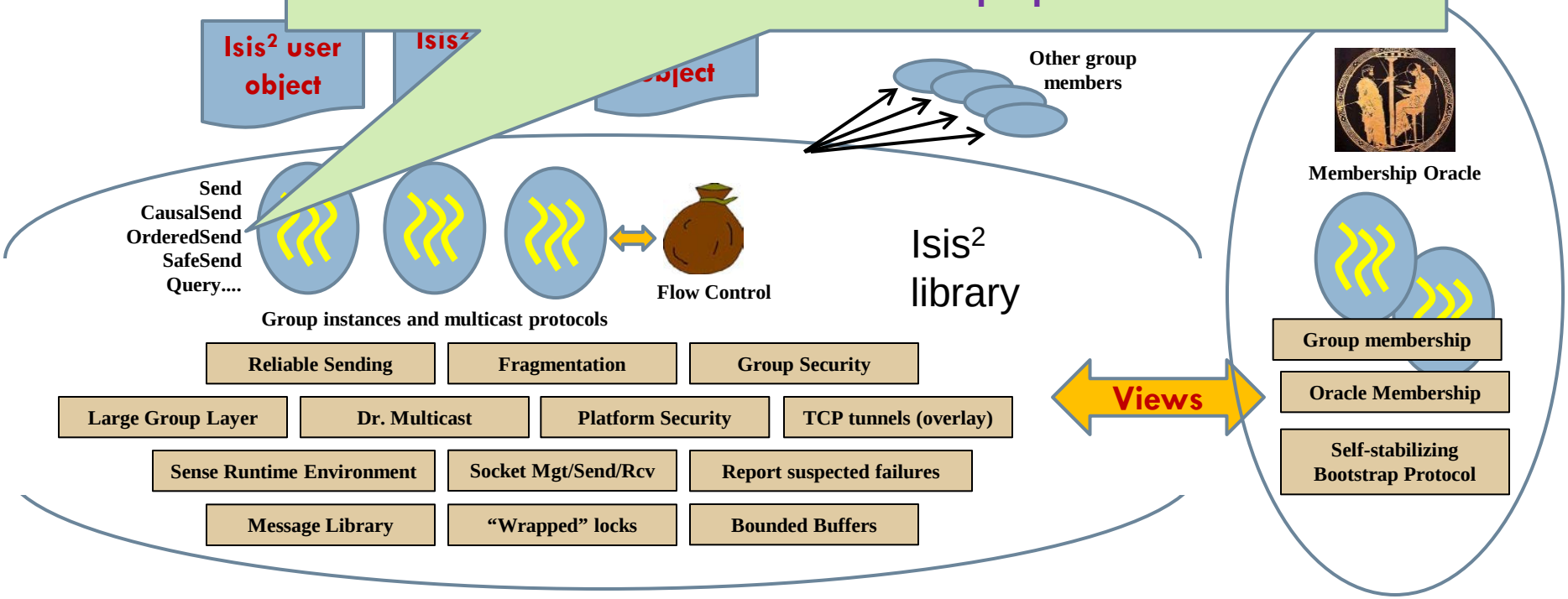
Why not just UDP multicast?



- **These systems are complex, especially if you want to run on platforms like EC2**

Why not just UDP multicast?

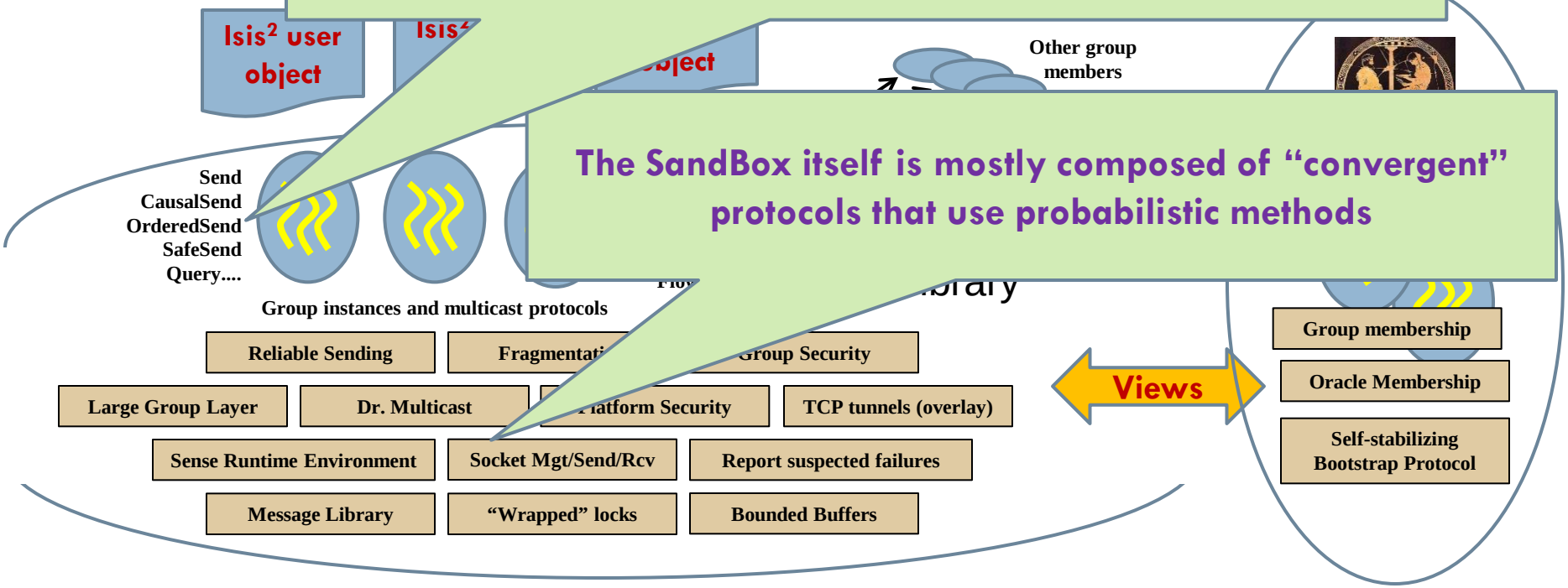
SafeSend and Send are two of the protocol components hosted over what we call the *large-scale properties sandbox*. The sandbox addresses issues like flow control, security, etc. All protocols share and benefit from those properties



- These systems are complex, especially if you want to run on platforms like EC2

Why not just UDP multicast?

SafeSend and Send are two of the protocol components hosted over what we call the *large-scale properties sandbox*. The sandbox addresses issues like flow control, security, etc. All protocols share and benefit from those properties



- These systems are complex, especially if you want to run on platforms like EC2

Other work: Differential Privacy for the Smart Grid

- We are also working on a new approach to achieve differential privacy for smart meters in the home (joint with Edward Tremel, Mark Jelasity, Bobby Kleinberg)

- In our scheme the meters run a cooperative protocol to jointly build the optimization models needed by the system operator to match supply and demand
  - Utility operates the solution, yet only learns the model!
  - But we don't have time to discuss this today

Back to CS6410 stuff

Pinning down the plan

Why take this course

- Learn about systems abstractions, principles, and artifacts that have had lasting value,
- Understand attributes of systems research that is likely to have impact,
- Become comfortable navigating the literature in this field,
- Learn to present papers in a classroom setting
- Gain experience in thinking critically and analytically about systems research, and
- Acquire the background needed to work on research problems currently under study at Cornell and elsewhere.

Who is the course "for"?

- Most of our CS6410 students are either
  - PhD students (but many are from non-CS fields, such as ECE, CAM, IS, etc)
  - Two year MS students who might switch into PhD
  - Undergraduates seriously considering a PhD

- Fall 2015: Too big to allow MEng students.
  - MEng program offers lots of other options;
  - CS6410 has a unique role for the core CS PhD group

CS6410 versus just-read-papers

- A paper on Isis^2 might just brag about how great it is, how well it scales, etc
- Reality is often complex and reflects complex tensions and decisions that force compromises
- In CS6410 our goal is to be honest about systems: see what the authors had to say, but think outside of the box they were in when they wrote the papers

Details

- Instructor: Ken Birman
  - ken@cs.cornell.edu
  - Office Location: 114 Gates
- TA: Theo Gkountouvas
- Lectures:
  - CS 6410: Tu, Th: 10:10 – 11:25 PM, 114 Gates

Course Help

- Course staff, office hours, announcements, etc:
  - http://www.cs.cornell.edu/courses/cs6410/2015fa
- Please look at the course syllabus: the list of papers is central to the whole concept of this class
- Research project ideas are also listed there

CS 6410: Overview

- Prerequisite:
  - Mastery of CS3410, CS 4410 material
    - Fundamentals of computer architecture and OS design
    - How parts of the OS are structured
    - What algorithms are commonly used
    - What are the mechanisms and policies used
  - Some insights into storage systems, database systems "helpful"
  - Some exposure to networks, web, basic security ideas like public keys

CS 6410: Topics:

- Operating Systems
  - Core concepts, multicore, virtualization, uses of VMs, other kinds of "containment", fighting worms/viruses.
- Cloud-scale stuff
  - Storage systems for big data, Internet trends, OpenFlow
- Foundational theory
  - Models of distributed computing, state machine replication and atomicity, Byzantine Agreement.
  - Impact of social networks, P2P models, Self-Stabilization
- A few lectures will focus on new trends: RDMA, BitCoin (a distributed protocol!), etc

CS 6410: Readings

- Required reading for each lecture: 2 or 3 papers
  - Reflecting contrasting approaches, competition, criticism,…
  - Papers pulled from, best journals and conferences
    - TOCS, SOSP, OSDI, …
  - 26 lectures, 54 (required) papers + 50 or so "recommended"!

- Read papers before each class and bring notes
  - takes ~1 to 2 hrs per paper, write notes and questions
  - At the most, one or two papers may take 4 hours to understand

- Write a review and turn in at least one hour before class
  - Turn on online via Course Management System (CMS)
  - No late reviews will be accepted, but you can skip 5 of them
  - Graded by the person doing that lecture on a simple √-,√,√+ basis plus written comments.

Mini-Projects

- New, early part of semester

- Two of them
  - Hands on experience with multicore parallelism in C or C++

  - Hands on experience with cloud computing on EC2

CS 6410: Two small projects

- Goal: Get the rust off your systems skills!
- Mini-project one: Build a multi-threaded, multicore version of the game of life, in C or C++ unless you absolutely cannot use those languages. Make it really, really fast.
- Mini-project two: Take a standard Paxos and run it on Amazon's EC2 using Elastic Beanstalk. Identify bottlenecks (we aren't asking you to fix them)

CS 6410: Writing Reviews

- Each student is required to prepare notes on each paper before class and to bring them to class for use in discussion.
- Your notes should list assumptions, innovative contributions and criticisms.
  - Every paper in the reading list has at least one major weakness.
  - Don't channel the authors: your job is to see the bigger questions!

- Turn paper reviews in online before class via CMS
  - Be succinct—One paragraph per paper
    - Short summary of paper (two or three sentences)
    - Two to three strengths/contributions
    - and at least one weaknesses
  - One paragraph to compare/contrast papers
  - In all, turn in two to three paragraphs

CS 6410: Paper Presentations

- Ideally, each person will present a paper, depending on the stable class size
  - Read and understand both required and suggested papers
  - Learning to present a paper is a big part of the job!
  - The presenting person also grades the essays for that topic
- Two and a half weeks ahead of time
  - Meet with professor to agree on ideas to focus on
- One and a half weeks ahead of time
  - Have presentation prepared and show slides or “chalk talk” to professor
- One week ahead of time
  - Final review / do a number of dry-runs

CS 6410: Class Format

- 45-50 minutes presentation,
- 30 minutes discussion/brainstorming.
  - In that order, or mixed.
- All students are required to participate!
- Counts in final grading.

CS 6410: Research Project

- One major project per person
  - Or two persons for a very major project
- Initial proposal of project topic – due mid-September
- Survey of area (related works)–due begin of October

- Midterm draft paper – due begin of November
- Peer reviews—due a week later

- Final demo/presentation–due begin of December
- Final project report – due a week later

CS 6410: Project Suggestions

- Operating system features to better leverage RDMA
- New cloud-scale computing services, perhaps focused on applications such as the smart power grid, smart self-driving cars, internet of things, smart homes
- Study the security and distributed systems properties of BitCoin
- New systems concepts aimed at better supporting "self aware" applications in cloud computing settings (or even in other settings)
- Building better memory-mapped file systems: current model has become outmoded and awkward
- Tools for improving development of super fast multicore applications like the one in mini-project one.
- Software defined network infrastructure on the systems or network side (as distinct from Nate's focus on the PL side)
- … and you can invent more of your own!

Important Project Deadlines

9/11	Submit your topic of interest proposal
9/25	Submit 2-3 pages survey on topic
(Oct)	Discuss project topic with Matt/me
11/4	Midterm draft paper of project
12/4	Final demo/presentation of project
	Final paper on project

CS 6410: Grading

- Class Participation ~ 40%
  - lead presentation, reading papers, write reviews, participation in class discussion
- Projects ~ 50%
  - Probably 20% will be the two mini-projects, 30% the big term one
  - Proposal, survey, draft, peer review, final demo/paper
- Subjective ~ 10%
- This is a rough guide

Academic Integrity

- Submitted work should be your own
- Acceptable collaboration:
  - Clarify problem, C syntax doubts, debugging strategy
  - You may use any idea from any other person or group in the class or out, provided you clearly state what you have borrowed and from whom.
  - If you do not provide a citation (i.e. you turn other people's work in as your own) that is cheating.
- Dishonesty has no place in any community
  - May NOT be in possession of someone else's homework/project
  - May NOT copy code from another group
  - May NOT copy, collaborate or share homework/assignments
  - University Academic Integrity rules are the general guidelines
- Penalty can be as severe as an 'F' in CS 6410

Stress, Health and Wellness

- Need to pace yourself to manage stress
  - Need regular sleep, eating, and exercising
- Don't miss class… but….
- Do not come to class sick (with the flu)!
  - Email me ahead of time that you are not feeling well
  - People not usually sick more than once in a semester

Before Next time

- Rank-order 2 papers to present (first and second half)
- Read first papers below and write review
  - End-to-end arguments in system design, J.H. Saltzer, D.P. Reed, D.D. Clark. ACM Transactions on Computer Systems Volume 2, Issue 4 (November 1984), pages 277--288.

    http://portal.acm.org/citation.cfm?id=357402
  - Hints for computer system design, B. Lampson. Proceedings of the Ninth ACM Symposium on Operating Systems Principles (Bretton Woods, New Hampshire, United States) 1983, pages 33--48.

    http://portal.acm.org/citation.cfm?id=806614
- Check website for updated schedule