CS514: Intermediate Course in Operating Systems

Professor Ken Birman
Vivek Vishnumurthy: TA

Quicksilver: Multicast for modern settings

- Developed by Krzys Ostrowski
- Goal is to reinvent multicast with modern datacenter and web systems in mind

Talk outline

- Objective
- Two motivating examples
- Our idea and how it “looks” in Windows
- How Quicksilver works and why it scales
- What next? (perhaps, gossip solutions)
- Summary

Our Objective

- Make it easier for people to build scalable distributed systems
- Do this by
  - Building better technology
  - Making it easier to use
  - Matching solutions to problems people really are facing

Motivating examples

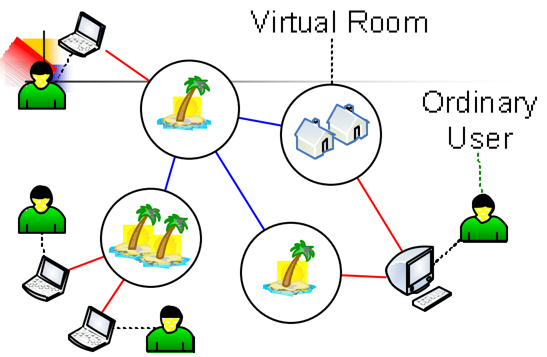
- Before we continue, look at some examples of challenging problems
- Today these are hard to solve
- Our work needs to make them easier
- Motivating examples:

  (1) Web 3.0 – “active content”

  (2) Data center with clustered services

Web 1.0... 2.0... 3.0...

- Web 1.0: browsers and web sites
- Web 2.0: Google mashups and web services that let programs interact with services using Web 1.0 protocols. Support for social networks.
- Web 3.0: A world of “live content”

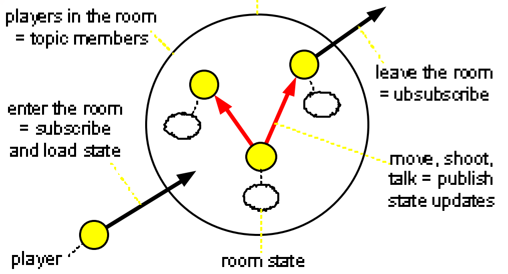


Virtual Room

Ordinary User

"Publish-Subscribe" Services (I)

room in a game = publish-subscribe topic



players in the room = topic members

enter the room = subscribe and load state

leave the room = ubsubscribe

move, shoot, talk = publish state updates

player

room state

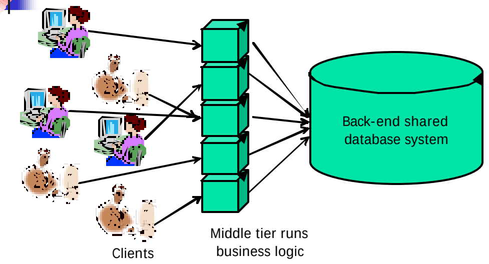
Observations?

- Web 3.0 could be a world of highly dynamic, high-data rate pub-sub
- But we would need a very different kind of pub-sub infrastructure
  - Existing solutions can't scale this way...
  - ... and aren't stable at high data rates
  - ... and can't guarantee "consistency"

Motivating example (2)

- Goal: Make it easy to build a datacenter
  - For Google, Amazon, Fnac, eBay, etc
- Assume each center
  - Has many computers (perhaps 10,000)
  - Runs lots of "services" (hundreds or more)
  - Replicates services & data to handle load
- Must also interconnect centers

Today's prevailing solution



Back-end shared database system

Middle tier runs business logic

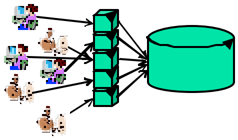
Clients

Concerns?

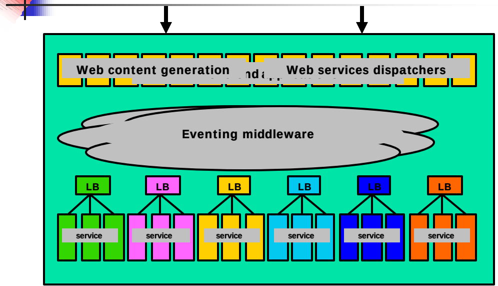
- Potentially slow (especially after crashes)
- Many applications find it hard to keep all their data in databases
  - Otherwise, we wouldn't need general purpose operating systems!
- Can we eliminate the database?
  - We'll need to replicate the "state" of the service in order to scale up

Response?

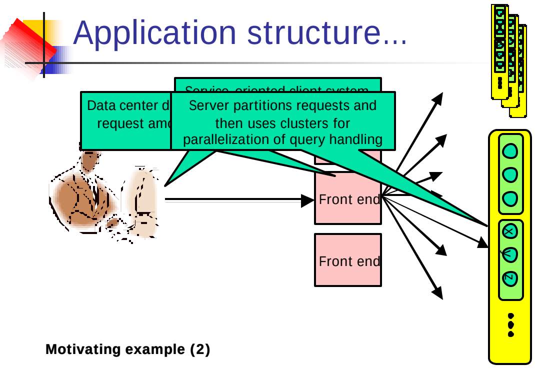
- Industry is exploring various kinds of in-memory database solutions
- These eliminate the third tier



A glimpse inside eStuff.com



Application structure...



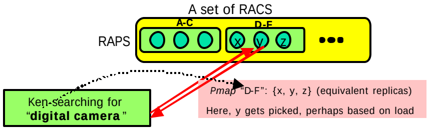
Service-oriented client system

Data center d
request amo

Server partitions requests and then uses clusters for parallelization of query handling

A RAPS of RACS (Jim Gray)

- RAPS: A reliable array of partitioned subservices
- RACS: A reliable array of cloned server processes



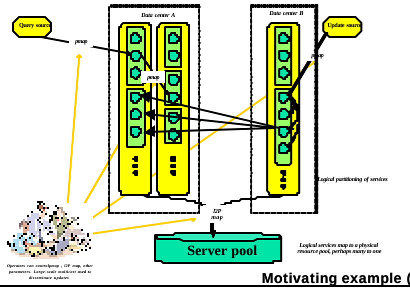
Ken searching for "digital camera"

Pmap "D-F": {x, y, z} (equivalent replicas)

Here, y gets picked, perhaps based on load

"RAPS of RACS" in Data Centers

Services are hosted at data centers but accessible systemwide



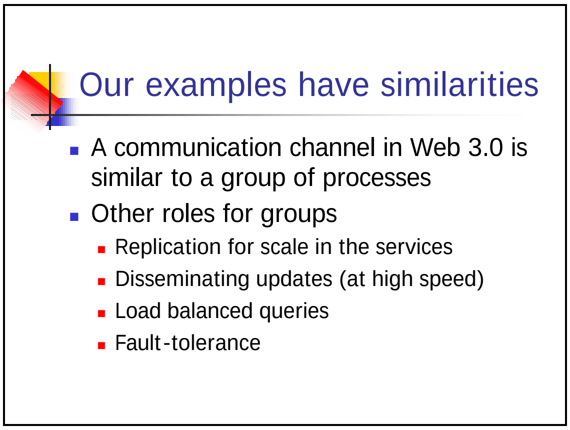
Logical partitioning of services

Logical services map to a physical resource pool, perhaps many to one

Operators can control pmap , l2P map, other parameters. Large-scale multicast used to disseminate updates

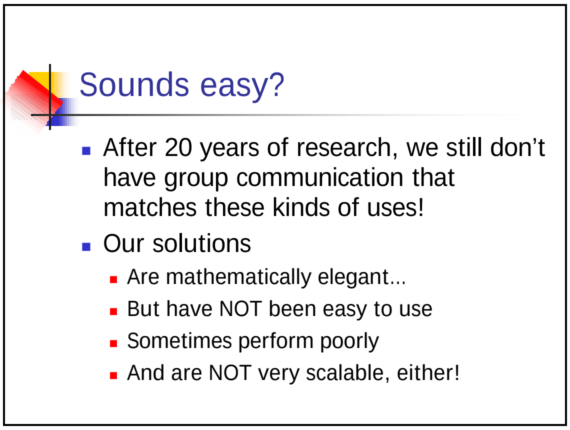
Our examples have similarities

- Both replicate data in groups
  - ... that have a **state** (evolved over time)
  - ... and a **name** (or "topic", like a file name)
  - ... updates are done by **multicasts**
  - ... queries can be handled by any member
- There will be a *lot* of groups
- Reliability need depends on application



Our examples have similarities

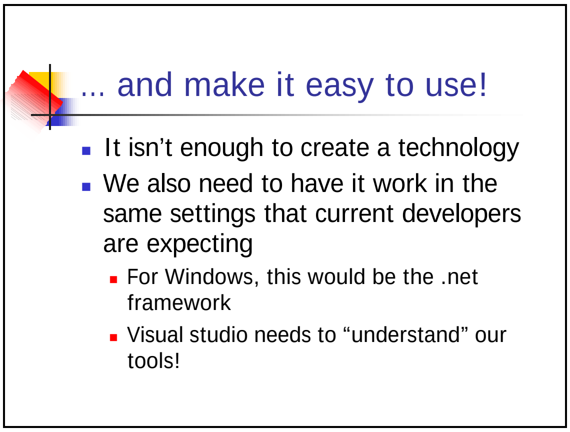
- A communication channel in Web 3.0 is similar to a group of processes
- Other roles for groups
  - Replication for scale in the services
  - Disseminating updates (at high speed)
  - Load balanced queries
  - Fault-tolerance



Sounds easy?

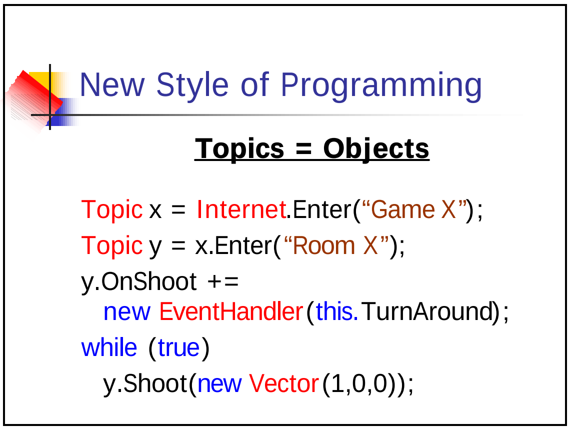
- After 20 years of research, we still don't have group communication that matches these kinds of uses!
- Our solutions
  - Are mathematically elegant…
  - But have NOT been easy to use
  - Sometimes perform poorly
  - And are NOT very scalable, either!

Integrating groups with modern platforms



… and make it easy to use!

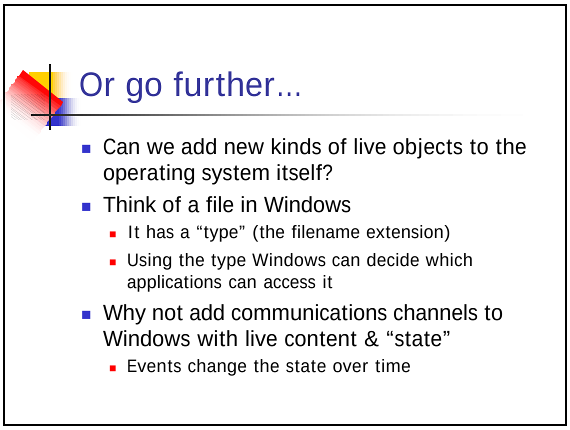
- It isn't enough to create a technology
- We also need to have it work in the same settings that current developers are expecting
  - For Windows, this would be the .net framework
  - Visual studio needs to "understand" our tools!



New Style of Programming

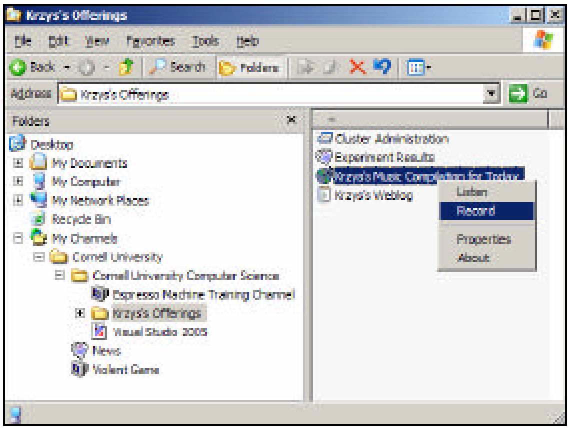
Topics = Objects

```
Topic x = Internet.Enter("Game X");
Topic y = x.Enter("Room X");
y.OnShoot +=
  new EventHandler(this.TurnAround);
while (true)
  y.Shoot(new Vector(1,0,0));
```



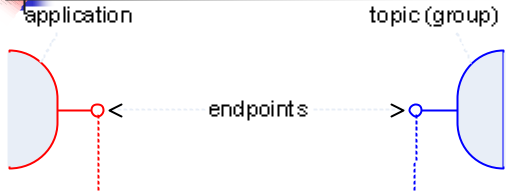
Or go further…

- Can we add new kinds of live objects to the operating system itself?
- Think of a file in Windows
  - It has a "type" (the filename extension)
  - Using the type Windows can decide which applications can access it
- Why not add communications channels to Windows with live content & "state"
  - Events change the state over time



Exploiting the Type System

Typed Publish-Subscribe



need virtual synchrony, RSA encryption, msgs signed by K-directorate

must buffer all msgs, participate in peer-to-peer recovery etc.

Vision: A new style of computing

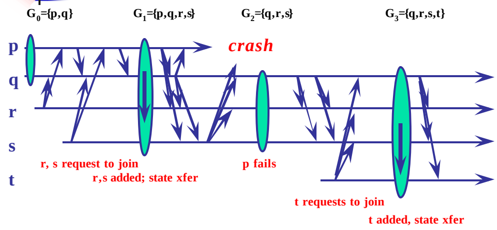
- With groups that could represent…
  - A distributed service replicated for fault-tolerance or availability or performance
  - An abstract data type or shared object
  - A sharable mapped file
  - A “place” where things happen

The “Type” of a Group means “The properties it supports”

Examples of properties

- Best effort
- Virtual synchrony
- State machine replication (consensus)
- Byzantine replication (PRACTI)
- Transactional 1-copy serializability

Virtual Synchrony Model



... to date, the *only* widely adopted model for consistency and fault-tolerance in highly available networked applications

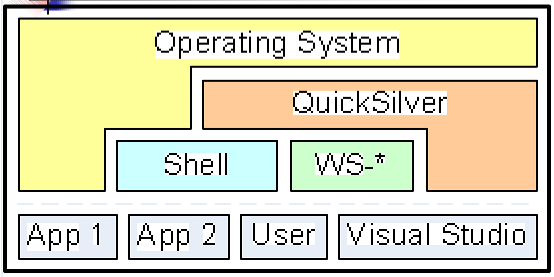
Quicksilver system

- Quicksilver: Incredibly scalable infrastructure for publish-subscribe
  - Each topic is a group
  - Tightly integrated with Windows .net
  - Tremendous performance and robustness
- Being developed step by step
  - Currently: QSM (scalability and speed)
  - Next: QS/2 (QSM + reliability models)

QS/2 Properties Framework

- In QS/2, the type of a group is
  - Understood by the operating system
  - But implemented by our “properties framework”
- Each type corresponds to a small code fragment in a new high-level language
  - It looks a bit like SETL (set-valued logic)
  - Joint work with Danny Dolev

Operating System Embedding



Technology Needs

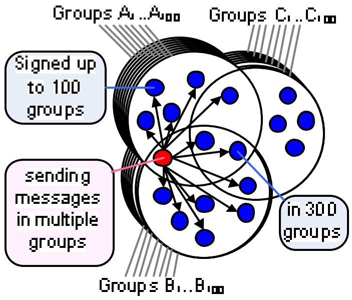
- **Scalability** → in multiple dimensions: #nodes, #groups, churn, failure rates etc.
- **Performance** → full power of the platform
- **Reliability** → consistent views of the state
- **Embeddings** → easy and natural to use
- **Interoperability** → integrating different systems, modularity, local optimization

QuickSilver Scalable Multicast

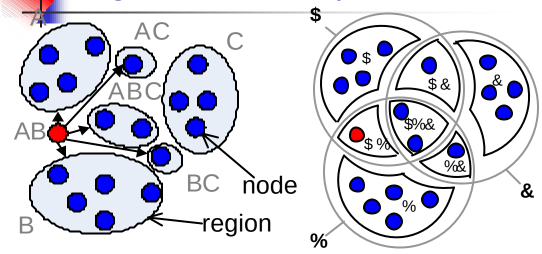
- Simple ACK-based reliability property
- Managed code (.NET, 95%C#, 5%MC++)
- Entire QuickSilver platform: ~250 KLOC
- Throughputs close to network speeds
- Scalable in multiple dimensions
- Tested with up to ~200 nodes, 8K groups
- Robust against a range of perturbances
- Free: www.cs.cornell.edu/projects/QuickSilver/QSM

Making It Scalable

Scalable Dissemination

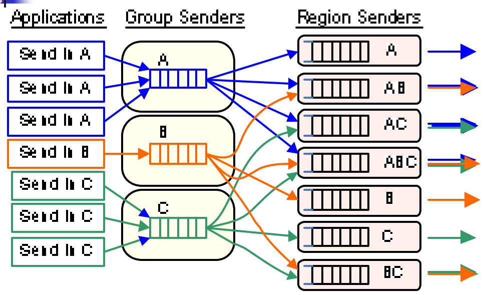


Regions of Overlap

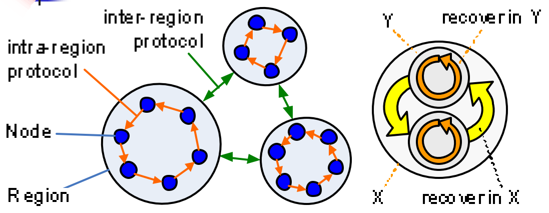


“region” = set of nodes with“similar” membership

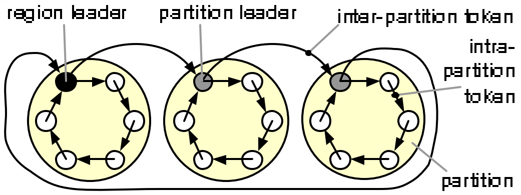
Mapping Groups to Regions (I)

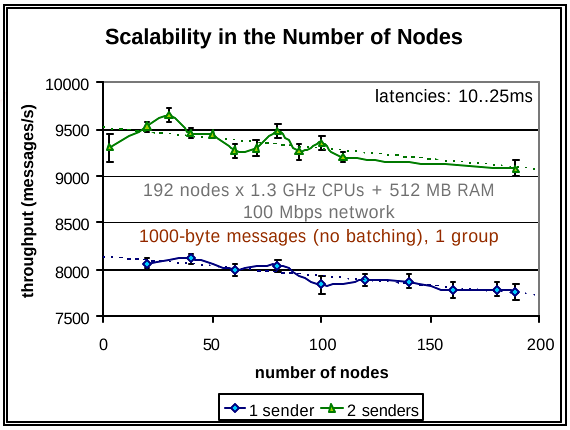


Hierarchy of Protocols (I)



Hierarchy of Protocols (II)



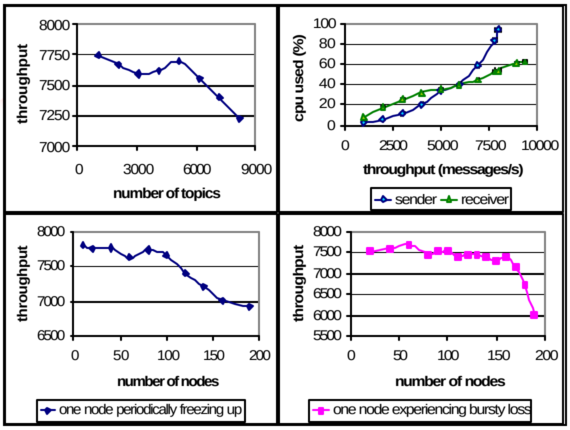


Scalability in the Number of Nodes

latencies: 10..25ms

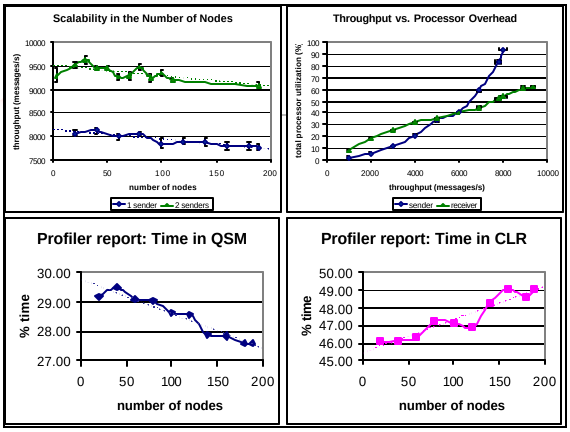
192 nodes x 1.3 GHz CPUs + 512 MB RAM
100 Mbps network

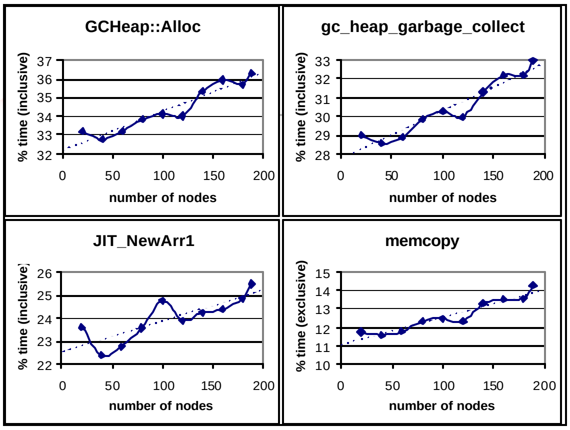
1000-byte messages (no batching), 1 group

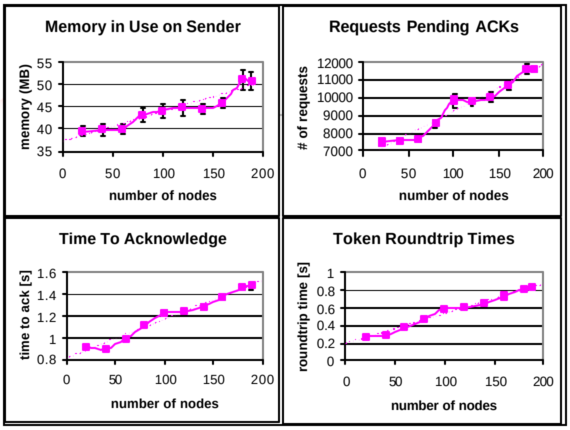


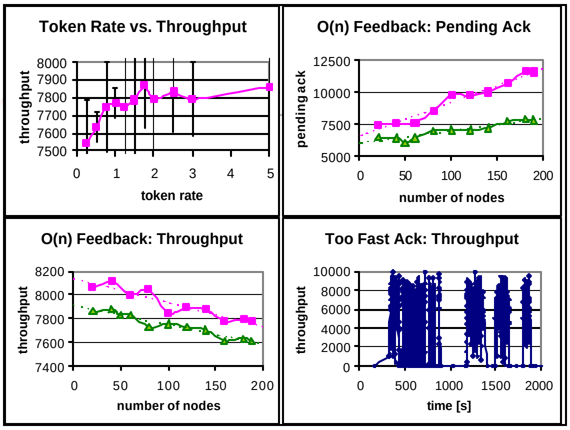
Is a Scalable Protocol Enough?

- So we know how to design a protocol…
- …but building a high-performance pub-sub engine is much more than that:
  - System resources are limited
  - Scheduling behaviors matter
  - Running in managed environment
  - Must tolerate other processes, GC, etc.





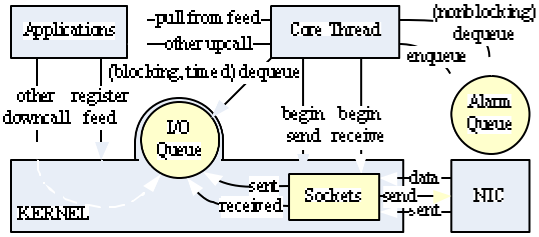




Observations

- In managed environment memory is costly
  - Buffering, complex data structures etc. matter
  - ...and garbage collection can be disruptive
- Low latency is the key
  - Allows to limit resource usage
  - Depends on the protocol…
  - ...but is also affected by GC, applications etc.
  - Can't be easily substituted

Threads Considered Harmful



Applications

pull from feed

other upcall

Core Thread

(nonblocking) dequeue

enqueue

(blocking, timed) dequeue

other downcall

register feed

I/O Queue

begin send

begin receive

Alarm Queue

KERNEL

sent

received

Sockets

data

send

sent

NIC

Looking beyond Quicksilver

- Quicksilver is really two ideas
  - One idea is concerned with how to embed live content into systems like Windows
    - As typed channels with file-system names
    - Or as pub-sub event topics
  - The other concerns scalable support for group communication in managed settings
    - The protocol tricks we've just seen

Looking beyond Quicksilver

- Quicksilver supports virtual synchrony
  - Hence is incredibly powerful for coordinated, consistent behavior
  - And fast too
- But not everything is ideally matched to this model of system
  - Could gossip mechanisms bring something of value?

Gossip versus other "models"

- Gossip is good for:
  - Emergent structure
  - Steady background tracking of state
  - Finding things in systems that are big and unstructured
- ... but is
  - Slow, perhaps costly in messages

- Vsync is good for:
  - Replicating data
  - Notifying processes when events occur
  - 2-phase interactions within groups
- ... but needs
  - "Configuration"
  - Costly setup

Emergent structure

- For example, building an overlay
  - We might want to overlay a tree on some set of nodes
  - Gossip algorithms for this sort of thing work incredibly well and need very little configuration help
  - And are extremely robust – they usually converge in log(N) time using bounded size messages…

Background state

- Suppose we want to continuously track status of some kind
  - Average load on a system, or average rate of timeout events
  - Closest server of some kind
- Gossip is very good at this kind of continuous monitoring – we pay a small overhead and the answer is always at hand.

Finding things

- The problem arises in settings where
  - There are many "things"
  - State is rather dynamic and we prefer to keep information close to the owner
  - Now and then (rarely) someone does a search, and we want snappy response
- Gossip-based lookup structures work really well for these sorts of purposes

Gossip versus other "models"

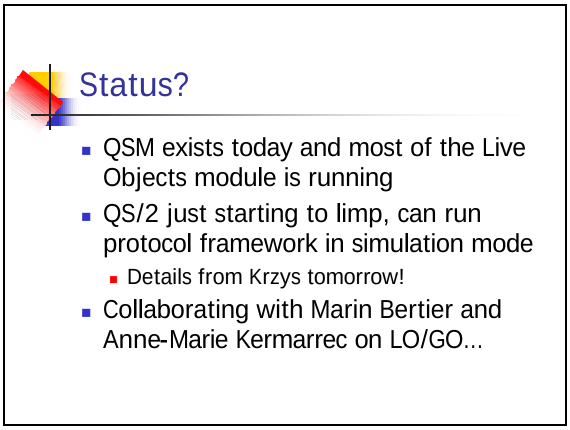
- Gossip is good for:
  - Emergent structure
  - Steady background tracking of state
  - Finding things in systems that are big and unstructured
- Vsync is good for:
  - Replicating data
  - Notifying processes when events occur
  - 2-phase interactions within groups

Unifying the models

- Could we imagine a system that
  - Would "look like" Quicksilver within Windows (an elegant, clean fit)…
  - Would offer gossip mechanisms to support what gossip is best at…
  - And would offer group communication with a range of strong consistency models for what "they" are best at?

Building QS/3 for Web 3.0…

- Break QS/2 into two modules
  - A "framework" that supports plug-in communication modules
  - A module for scalable group communication
- Then design a gossip-based subsystem that focuses on what gossip does best
  - And run it as a second module under the "Live Objects" layer of QS/2: LO/GO



Status?

- QSM exists today and most of the Live Objects module is running
- QS/2 just starting to limp, can run protocol framework in simulation mode
  - Details from Krzys tomorrow!
- Collaborating with Marin Bertier and Anne-Marie Kermarrec on LO/GO…