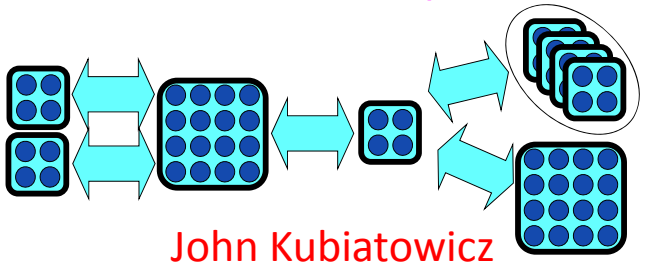
Tessellation: Refactoring the OS around Explicit Resource Containers with Continuous Adaptation

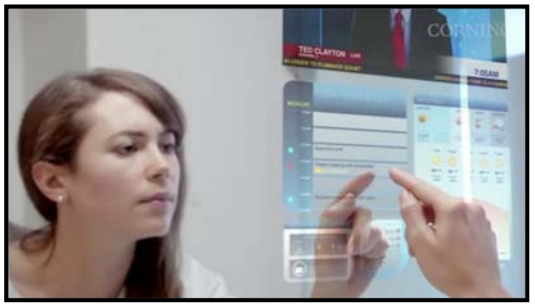


John Kubiatowicz
UC Berkeley
June 5th, 2013

Juan Colmenares, Gage Eads, Steven Hofmeyr, Sarah Bird, Miquel Moreto, David Chou, Brian Gluzman, Eric Roman, Davide Bartolini, Nitesh Mor, Krste Asanovic

Resources in a Smart Space

- Potential Displays Everywhere
  - Walls, Tables, Appliances, Smart Phones, Google Glasses….
- Audio Output Everywhere
- Inputs Everywhere
  - Touch Surfaces
  - Cameras/ Gesture Tracking
  - Voice
- Context Tracking
  - Who is Where
  - What do they want
  - Which Inputs map to which applications



Support for Applications

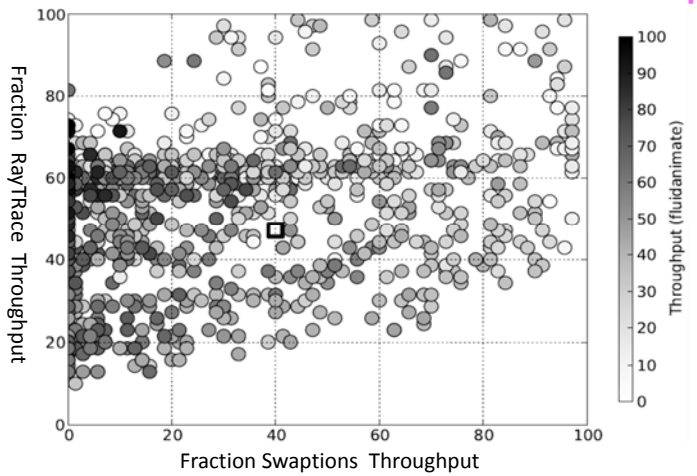
- Real Time Requirements
  - Sophisticated multimedia interactions
  - Control of/interaction with health-related devices
- Responsiveness Requirements
  - Provide a good interactive experience to users
- Explicitly Parallel Components
  - However, parallelism may be “hard won” (not embarrassingly parallel)
  - Must not interfere with this parallelism
- Direct Interaction with Cloud services (including storage and compute)
  - Potentially extensive use of remote services
  - Serious security/data vulnerability concerns
- No existing OS handles all of these well….

Changing the Structure of Operating Systems (and Applications that run on them)

Guaranteeing Resources

- What might we want to guarantee?
  - Examples:
    - Guarantees of BW (say data committed to Cloud Storage)
    - Guarantees of Requests/Unit time (DB service)
    - Guarantees of Latency to Response (Deadline scheduling)
    - Guarantees of maximum time to Durability in cloud
    - Guarantees of total energy/battery power available to Cell
- What level of guarantee?
  - Firm Guarantee (Better than existing systems)
    - With high confidence (specified), Maximum deviation, etc.
- What does it mean to have guaranteed resources?
  - A Service Level Agreement (SLA)?
  - Something else?
- "Impedance-mismatch" problem
  - The SLA guarantees properties that programmer/user wants
  - The *resources* required to satisfy SLA are not things that programmer/user really understands

Resource Allocation must be Adaptive

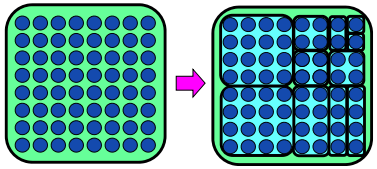


- Three applications:
  - 2 Guaranteed Performance apps (RayTrace, Swaptions)
  - 1 Best-Effort App (Fluidanimate)

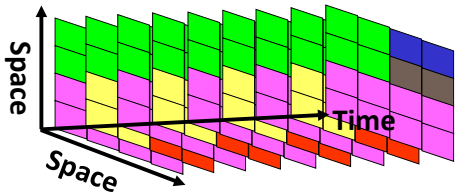
New Abstraction: the Cell

- Properties of a Cell
  - A user-level software component with guaranteed resources
  - Has full control over resources it owns ("Bare Metal")
  - Contains at least one memory protection domain (possibly more)
  - Contains a set of secured channel endpoints to other Cells
  - Contains a security context which may protect and decrypt information
- When mapped to the hardware, a cell gets:
  - Gang-schedule hardware thread resources ("Harts")
  - Guaranteed fractions of other physical resources
    - Physical Pages (DRAM), Cache partitions, memory bandwidth, power
  - Guaranteed fractions of system services
- Predictability of performance ⇒
  - Ability to model performance vs resources
  - Ability for user-level schedulers to better provide QoS

Implementing Cells: Space-Time Partitioning

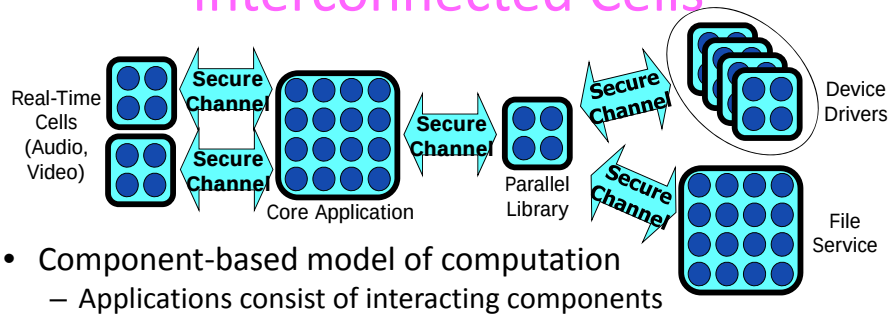


- Spatial Partition: Performance isolation
  - Each partition receives a vector of basic resources
    - A number HW threads
    - Chunk of physical memory
    - A portion of shared cache
    - A fraction of memory BW
    - Shared fractions of services



- Partitioning varies over time
  - Fine-grained multiplexing and guarantee of resources
    - Resources are gang-scheduled
- Controlled multiplexing, not uncontrolled virtualization
- Partitioning adapted to the system's needs

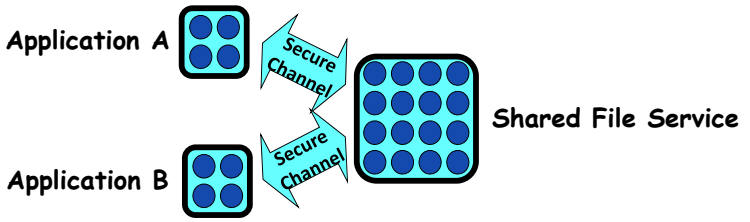
Applications Composed of Interconnected Cells



- Component-based model of computation
  - Applications consist of interacting components
  - Components may be local or remote
- Communication impacts Security and Performance
  - Channels are points at which data may be compromised
  - Channels define points for QoS constraints
- Naming process for initiating endpoints
  - Need to find compatible remote services
  - Continuous adaptation: links changing over time!

Impact on the Programmer

- Connected graph of Cells ⇔ Object-Oriented Programming
  - Lowest-Impact: Wrap a functional interface around channel
    - Cells hold "Objects", Channels carry RPCs for "method calls"
  - Greater Parallelism: Event triggered programming
- Shared services complicate resource isolation:
  - How to provide each client with guaranteed fraction of service?



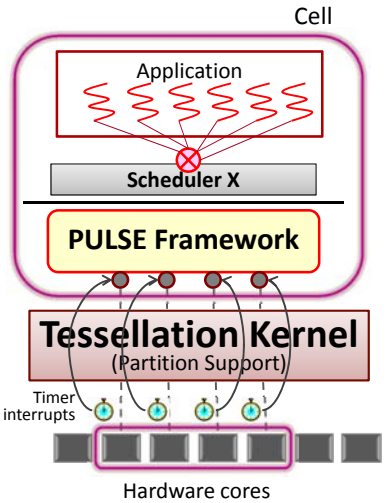
- Must somehow request the right number of resources
  - Analytically? AdHoc Profiling? Over commitment of resources?
  - Clearly doesn't make it easy to adapt to changes in environment

Custom Adaptive User-Level Scheduling

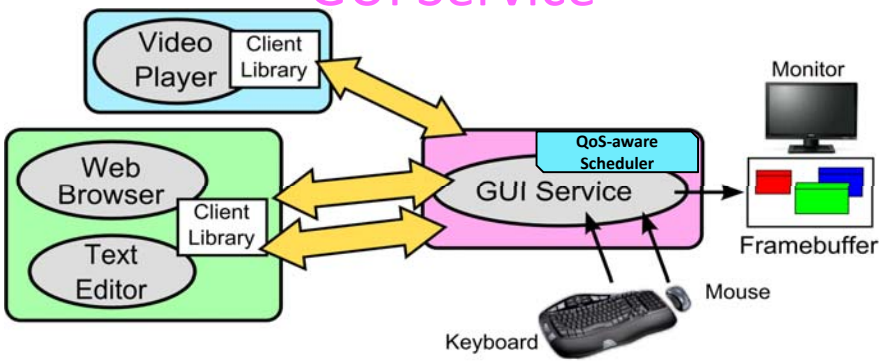
Frameworks for User-Level Runtimes

PULSE: Preemptive User-Level SchEdulers

- Framework Components
  - Hardware threads (harts)
  - Timer Interrupts for preemption
- Able to handle cell resizing
  - Automatic simulation of suspended scheduler segments
- Available schedulers
  - Round-robin (and pthreads)
  - EDF and Fixed Priority
  - Constant Bandwidth Server (M-CBS)
  - Juggle: load balancing SPMD apps
- Other framework: Lithe
  - Non-preemptive, Cooperative
  - Allows construction of schedulers that cooperate with libraries in handling processor resources.

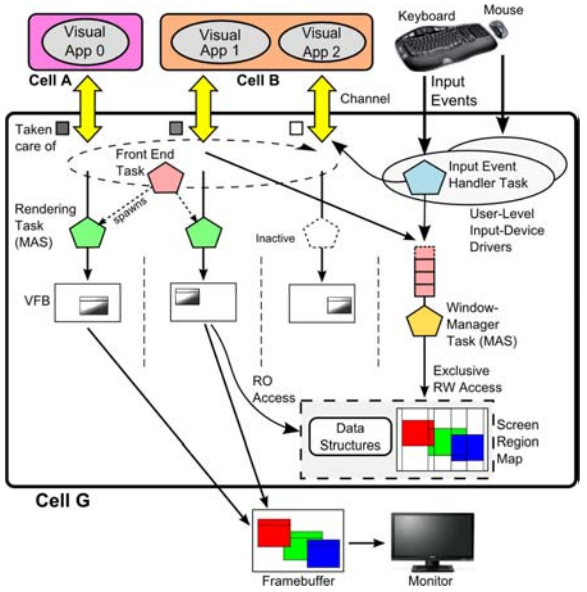


GUI Service

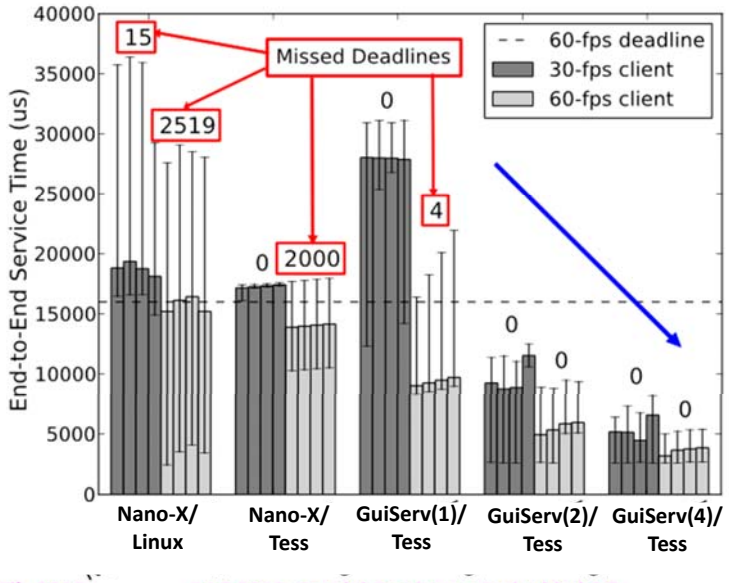


- Provides differentiated service to applications
  - Soft service-time guarantees
  - Performance isolation from other applications
- Operates on user-meaningful "actions"
  - E.g. "draw frame", "move window"
- Exploits task parallelism for improved service times

GUI Service Architecture



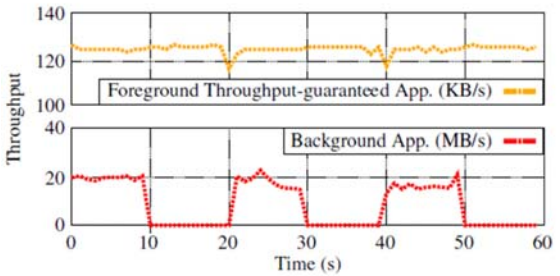
Performance of GUI Service



Out of 4000 frames

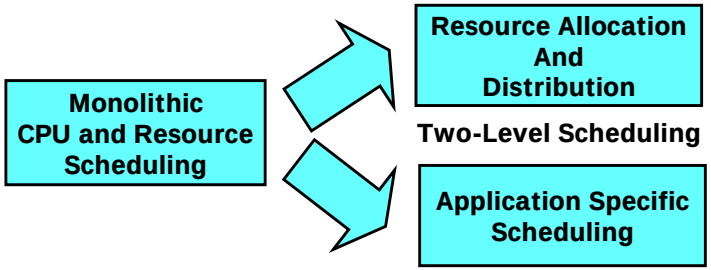
Network Service

- Supports reservations and proportional share of bandwidth
  - Using mClock scheduling algorithm (on top of PULSE)
- NIC driver is entirely contained in user-space
  - No system calls to transmit and receive buffers



Resource Distribution and Adaptation

Two Level Scheduling: Control vs Data Plane



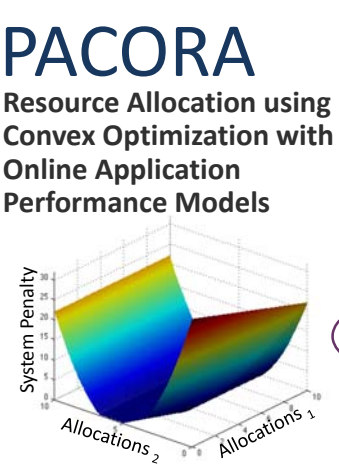
- Split monolithic scheduling into two pieces:
  - Course-Grained Resource Allocation and Distribution to Cells
    - Chunks of resources (CPUs, Memory Bandwidth, QoS to Services)
    - Ultimately a hierarchical process negotiated with service providers
  - Fine-Grained (User-Level) Application-Specific Scheduling
    - Applications allowed to utilize their resources in any way they see fit
    - Performance Isolation: Other components of the system cannot interfere with Cells use of resources

Performance-Aware Convex Optimization for Resource Allocation

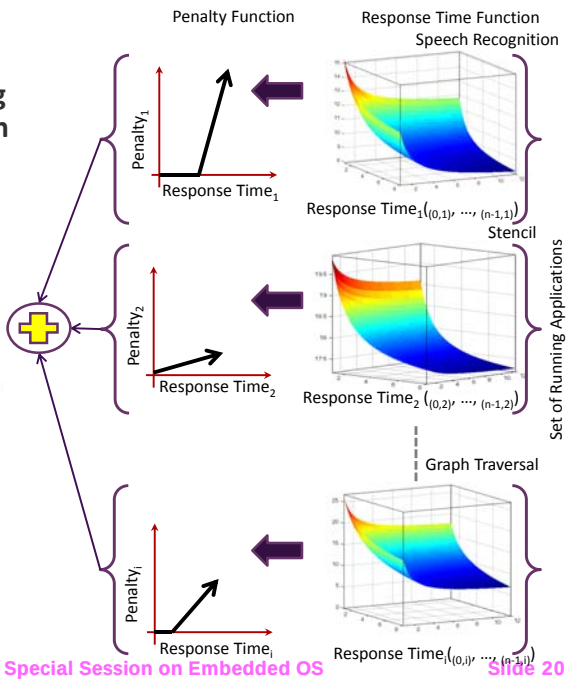
- Express problem as an optimization with constraints
  - Maximize for throughput, performance, battery life, etc.
- Measure performance as a function of resources for each application
  - Create performance model
  - Refine performance function from application history
- Combine resource-value functions and QoS requirements to make resource decisions
  - Application is given enough resources for QoS
  - Turn off resources that don't improve the system
  - Developers don't need to understand resources
- Basic Solution to Impedance Mismatch problem

PACORA

Resource Allocation using Convex Optimization with Online Application Performance Models

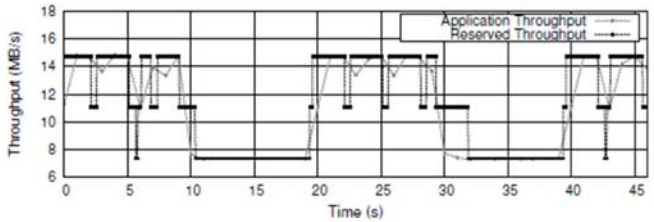


Continuously minimize the penalty of the system (subject to restrictions on the total amount of resources)



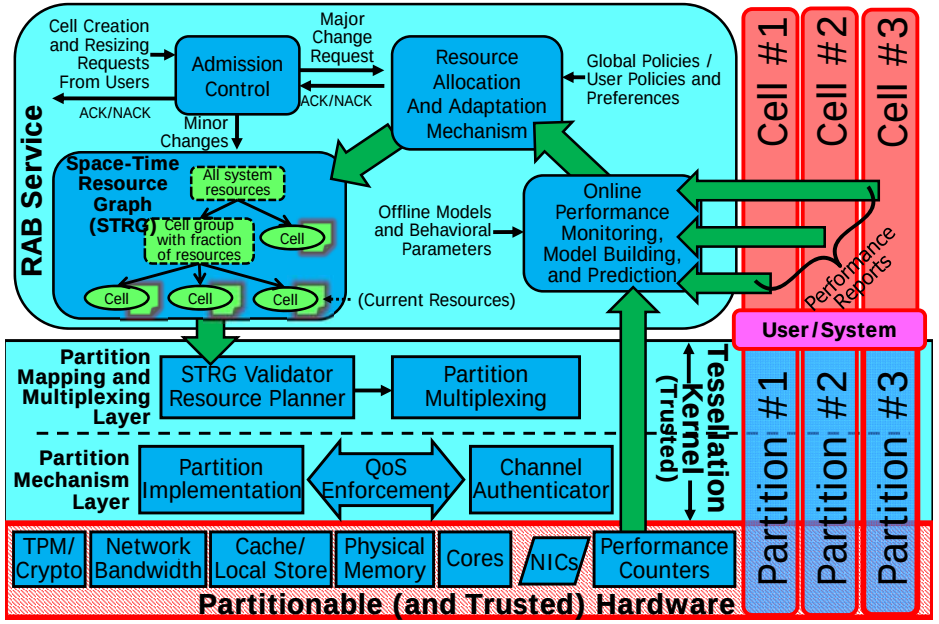
Feedback Driven Policies

- Generic Policies do well but online exploration can cause oscillations in performance
- Example: Video Player interaction with Network
  - Server or GUI changes between high and low bit rate
  - Goal: set guaranteed network rate:

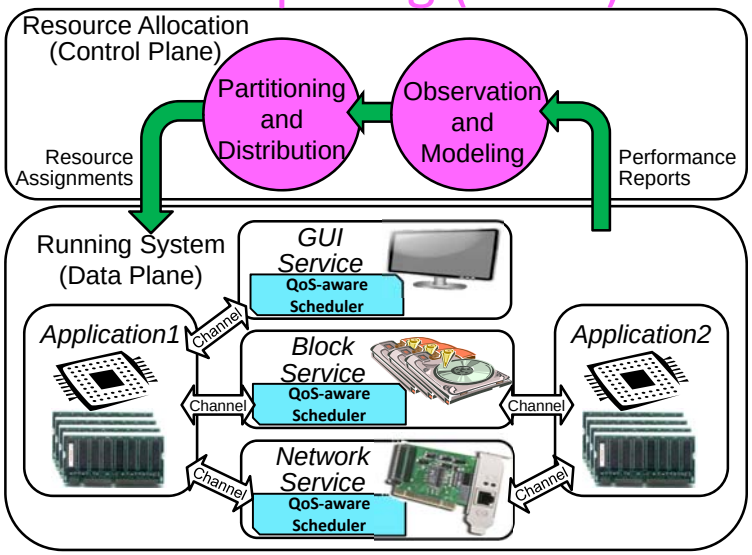


- Alternative: Application Driven Policy
  - Static models
  - Let network choose when to decrease allocation
  - Application-informed metrics such as needed BW

Architecture of Tessellation OS

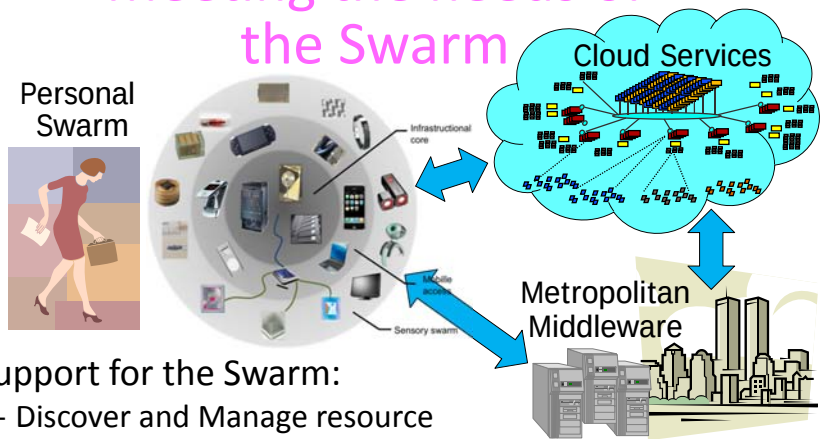


Adaptive Resource-Centric Computing (ARCC)



On Toward the Swarm

Meeting the needs of the Swarm

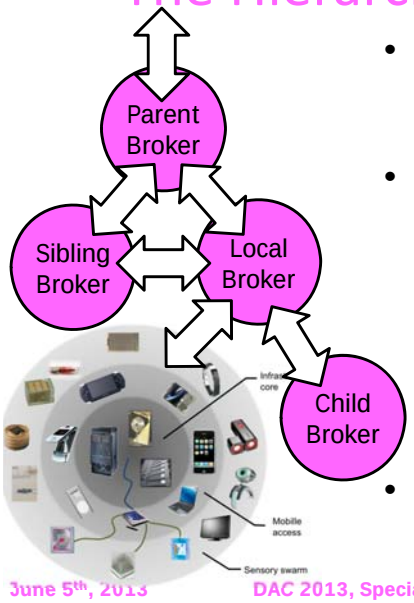


- Support for the Swarm:
  - Discover and Manage resource
  - Integrate sensors, portable devices, cloud components
  - Guarantee responsiveness, real-time behavior, throughput
  - Self-adapt to adjust for failure and provide performance predictability
  - Secure, high-performance, durable, available data

Cell as Ubiquitous Swarm Primitive

- Should every component in system host Cells?
  - Even sensors!?
  - What is minimal support?
    - Security Primitives
    - Communication support
  - Alternative: Bare sensors do not host Cells
- Mobile Environment
  - Constant changes in resource availability
  - Adaptation in resource requirements
  - Change connected graph continuously ⇒ Dynamic SLAs?
- Hierarchical Resource Broker Architecture
  - Many overlapping broker domains
  - Resource Ontologies?
- Cell is a natural way to handle heterogeneity
  - From the outside: export services to other Cells

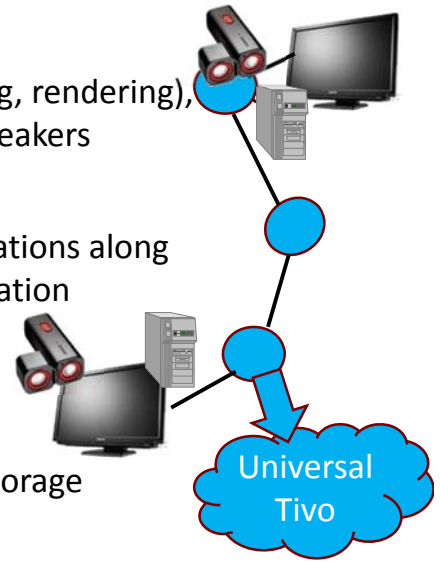
Brokering Service: The Hierarchy of Ownership



- Discover Resources in "Domain"
  - Devices, Services, Other Brokers
  - Resources self-describing?
- Allocate and Distribute Resources to Cells that need them
  - Solve Impedance-mismatch problem
  - Dynamically optimize execution
  - Hand out Service-Level Agreements (SLAs) to Cells
  - Deny admission to Cells which violate existing agreements
- Complete hierarchy
  - World graph of applications

On Demand Video Conference

- Local Resources
  - Display, Compute (encoding, rendering), Cameras, Microphones, Speakers
- Brokered resources
  - Request Bandwidth reservations along path from source to destination
    - Openflow? QoS API
  - Request service from compute for rendering
  - Guaranteed path to data storage



DataCentric Vision

- Hardware resources are a commodity
  - Computation resource fails? Get another
  - Sensor fails? Find another
  - Change your location? Find new resources
- All that really matters is the data!
  - Integrity, Privacy, Availability, Durability
  - Hardware to prevent accidental information leakage
- Permanent state handled by Universal Data Storage, Distribution, and Archiving
- Oceanstore Metric of Interest:
  - 1Mole of Bytes 6×10^{23} bytes (i.e. a YottaByte)

Universal Data: The Great Integrator

Conclusion

- Adaptive Resource-Centric Computing
  - Use of Resources negotiated hierarchically
  - Underlying Execution environment guarantees QoS
  - New Resources constructed from Old ones:
    - Aggregate resources in combination with QoS-Aware Scheduler
    - Result is a *new* resource that can be negotiated for
  - Continual adaptation and optimization
- Important components of future OS environment
  - Cells as Basic Unit of Resource and Security
    - User-Level Software Component with Guaranteed Resources
    - Secure Channels to other Cells
  - Observation, Monitoring, and Adaptation layers
    - Machine learning, Convex Optimization
  - Portable Secure Data infrastructure
    - If you can name it, you can use it
- Tessellation OS: http://tessellation.cs.berkeley.edu
  SwarmLab: http://swarmlab.eecs.berkeley.edu