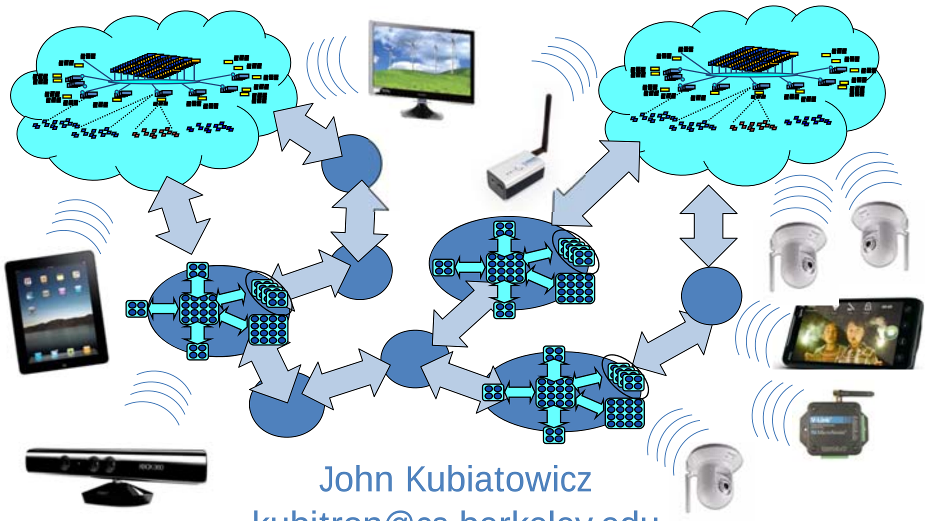
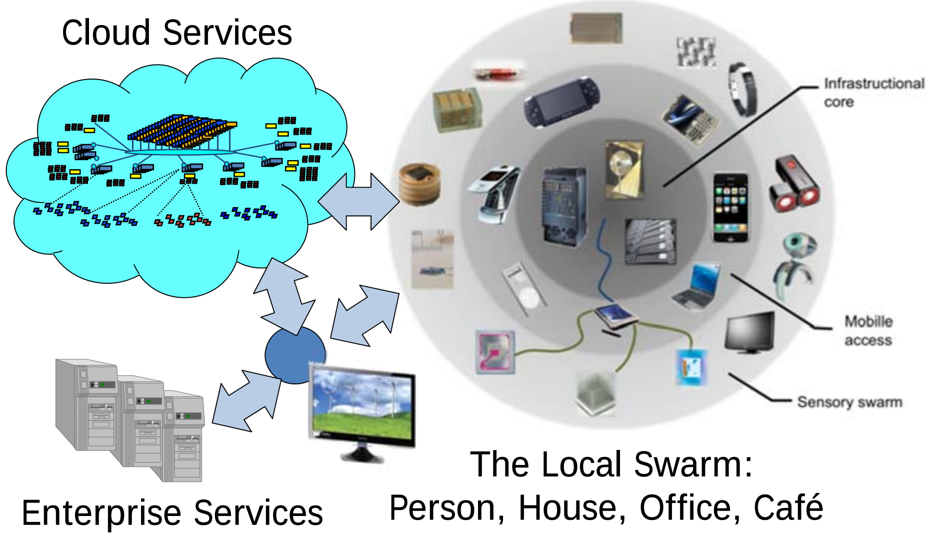
Reinventing Operating Systems for the Manycore Ubiquitous Swarm



John Kubiatowicz
kubitron@cs.berkeley.edu

The Swarm of Resources



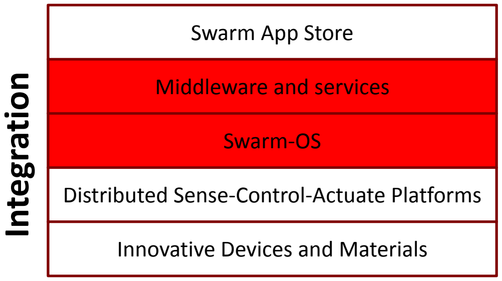
Cloud Services

Enterprise Services

The Local Swarm:
Person, House, Office, Café

Requirements for Swarm OS?

- What system structure required to support Swarm?
  - Integrate sensors, portable devices, cloud components
  - Guarantee responsiveness, real-time behavior, throughput
  - Services with guaranteed behavior, self-adapting to adjust for failure and performance predictability
  - Uniformly secure, durable, available data



Integration

Swarm App Store

Middleware and services

Swarm-OS

Distributed Sense-Control-Actuate Platforms

Innovative Devices and Materials

Infrastructional core
Mobile access
Sensory swarm

Today’s Software Reality

- Resources not well managed: QoS hard to achieve
  - 20th-century notions of utilization and resource virtualization
  - Despite a cornucopia of resources – we still cannot get the ones we need when we need them!
- Services not easily interconnected
  - Every service has a unique API
  - Highly-specialized “stovepipes” often do not provide exactly what users are looking for ⇒ they end up integrating “by hand”
  - Tradeoffs between client and cloud not easy to achieve
- Too many things explicitly depend on location:
  - *Where:* is my data stored? (oops – it *was* there!)
  - *Where:* can I execute this piece of functionality?
  - *Where:* can I display this information?
  - *Where:* did I *start* this job (because I have to finish it there)
- And others don’t properly depend on location:
  - *Here I am:* do something about it!

Support for Applications

- What support do we need for new Swarm applications?
  - Should we just port Linux, Android, or Windows 7?
  - A lot of functionality, hard to experiment with, possibly fragile, …
- Clearly, these applications will contain:
  - Direct interaction with Swarm and Cloud services
    - Potentially extensive use of remote services
    - Serious security/data vulnerability concerns
  - 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
- No existing OS handles all of these well….

The Swarm OS in a Nutshell

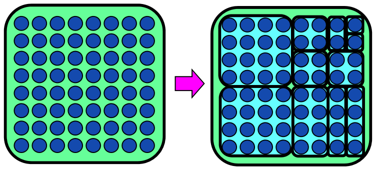
- Treat the world as a single system…
  - Set of all applications forms a connected graph of services
    - New Buzzword: Service Oriented Architecture?
  - Important aspects:
    - How to discover and assign resources to components
    - How to provide guaranteed access (QoS) to components
    - How to startup (boot?) applications
  - Data as a First-Class Citizen
- Primary software component: Cell
  - Software component with QoS-guaranteed access to resources
  - Exports service(s) with QoS guarantees to other Cells
    - Service provided by a Cell becomes resource for other Cells
- Hierarchical Resource Discovery and Awareness
  - Resources owned by broker agents capable of discovering, configuring, and assigning them
- Guaranteed Access to Assigned Resources
  - Cell and Networking implementation designed to guarantee access to resources and services

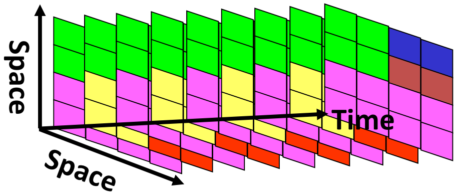
Changing the Structure of Operating Systems (and the Application that run on them)

Guaranteed 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

Space-Time Partitioning



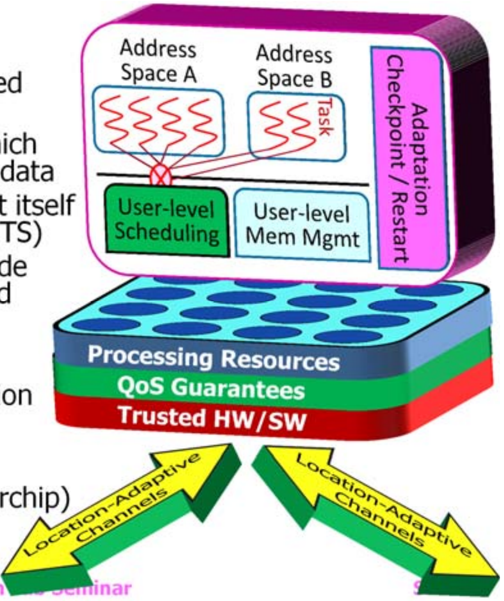


- Spatial Partition: Performance isolation
  - Each partition receives a vector of basic resources
    - A number HW threads
    - A portion of physical memory
    - A portion of shared cache
    - A fraction of memory bandwidth

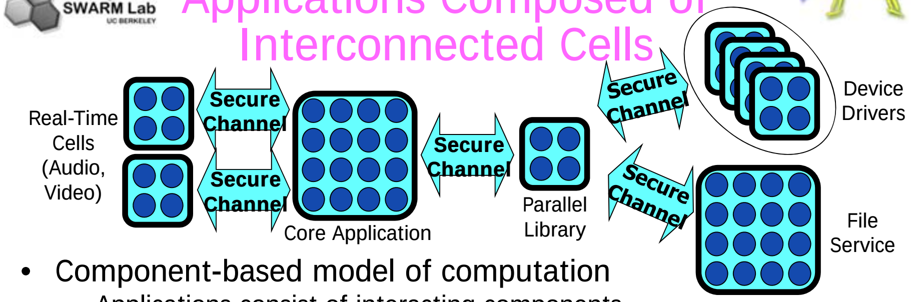
- Partitioning varies over time
  - Fine-grained multiplexing and guarantee of resources
    - Resources are gang-scheduled
- Controlled multiplexing, not uncontrolled virtualization
- Partitioning adapted to the system's needs

New OS Primitive: the Cell

- Cell Properties:
  - A user-level software component, with guaranteed resources
  - Explicit security context which allows access to protected data
  - Knowledge of how to adapt itself to new environments (SEJITS)
  - Checkpoint/restart to provide fault tolerance, mobility and adaptation
- Execution Environment:
  - Explicitly parallel computation
  - Resource Guarantees
  - Trusted computing base
  - Secure channels (intra/interchip) with ability to suspend and restart during migration



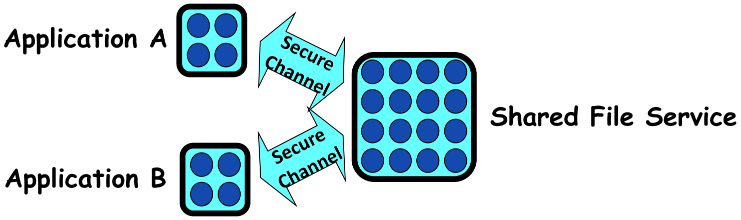
Applications Composed of Interconnected Cells



- Component-based model of computation
  - Applications consist of interacting components
  - Components may be local or remote
- Communication defines Security Model
  - Channels are points at which data may be compromised
  - Channels define points for QoS constraints
  - Question: Can we provide proofs of correctness on inter-cell protocols?
- Naming process for initiating endpoints
  - Need to find consistent version of library code (within cell)
  - Need to find compatible remote services
  - Solution of version constraint problem for running application

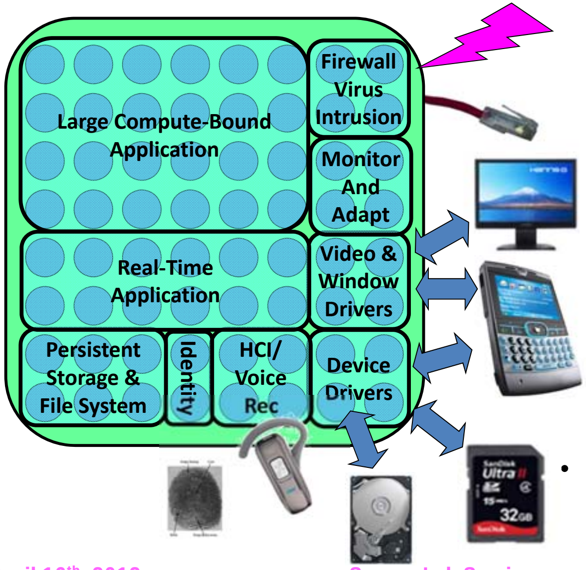
Impact on the Programmer

- Connected graph of Cells ⇔ Object-Oriented Programming
  - Lowest-Impact: Wrap a functional interface around channel
    - Cells hold "Objects", Secure channels carry RPCs for "method calls"
    - Example: POSIX shim library calling shared service Cells
  - Greater Parallelism: Event triggered programming
- Shared services complicate resource isolation:
  - How to guarantee that each client gets guaranteed fraction of service?
  - Distributed resource attribution (application as distributed graph)



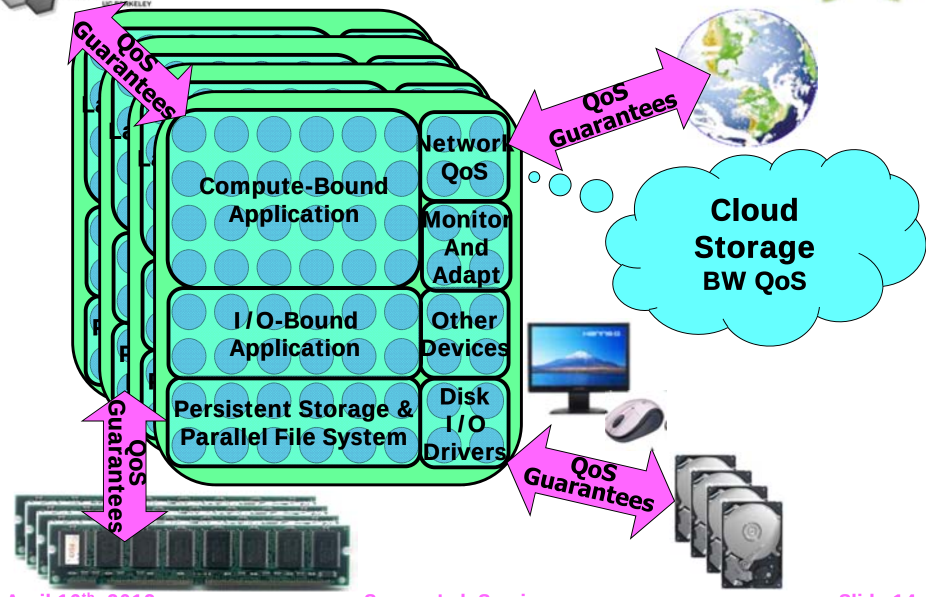
- Communication defines Security Model
  - SecureCell: Keys as resource – Outside entity handles privacy concerns
  - Mandatory Access Control Tagging (levels of information confidentiality)

Tessellation: The Exploded OS



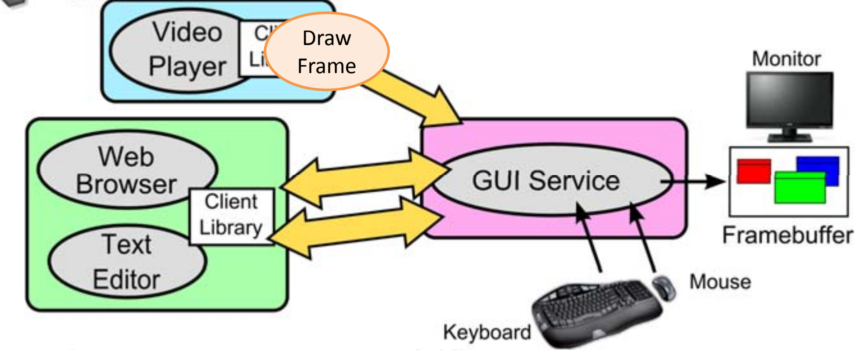
- Normal Components split into pieces
  - Device drivers (Security/Reliability)
  - Network Services (Performance)
    - TCP/IP stack
    - Firewall
    - Virus Checking
    - Intrusion Detection
  - Persistent Storage (Performance, Security, Reliability)
  - Monitoring services
    - Performance counters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

Tessellation on Cloud Servers



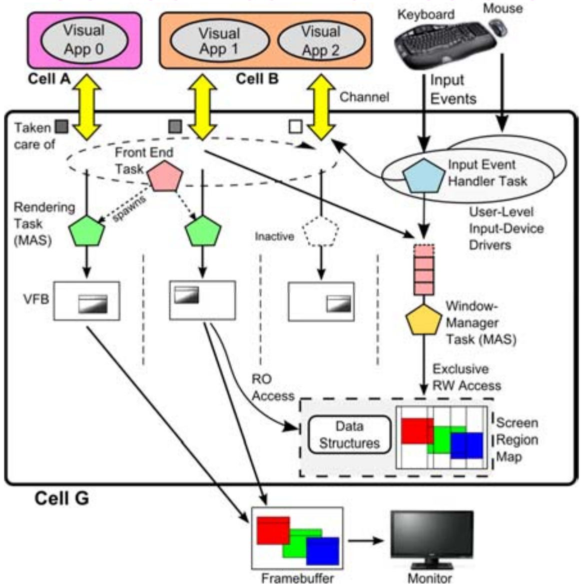
Example of Wrapping Resource In Service-providing Cell

The Tesselation GUI Service



- Operate on user-meaningful "actions"
  - E.g. "draw frame", "move window"
- Service time guarantees (soft real-time)
  - Differentiated service per application
  - E.g. text editor vs video
- Performance isolation from other applications

GUI Service Architecture

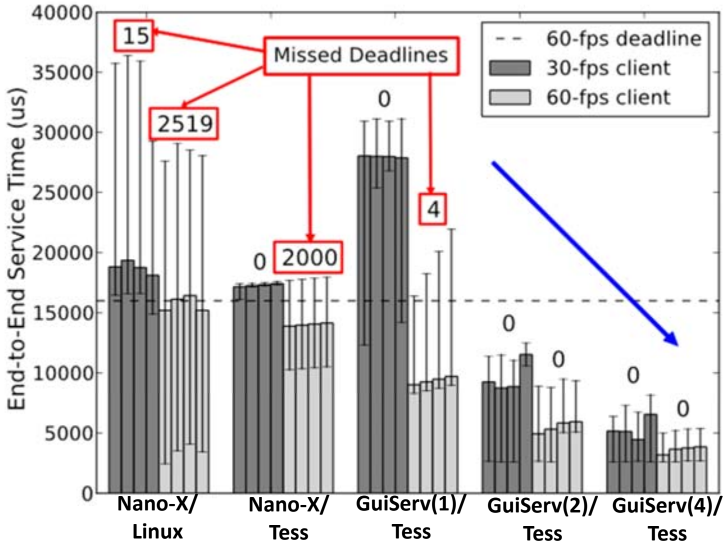


Experiment Setup

- Capture end-to-end service times (less is better)
- 8 video clients, each sending 4000 video frame requests
  - 4 are 30-fps videos (352 x 288)
  - 4 are 60-fps videos (352 x 288)
- 5 different GUI system setups
  - Traditional window system running on Linux (Nano-X/Linux)
  - Traditional window system running on Tessellation (Nano-X/Tess)
  - GUI Service running on Tessellation
    - With 1 core (GuiServ(1)/Tess)
    - With 2 cores (GuiServ(2)/Tess)
    - With 4 cores (GuiServ(4)/Tess)
- Running on machine with 8 hyperthreads
- Window system and video clients running on separate cores

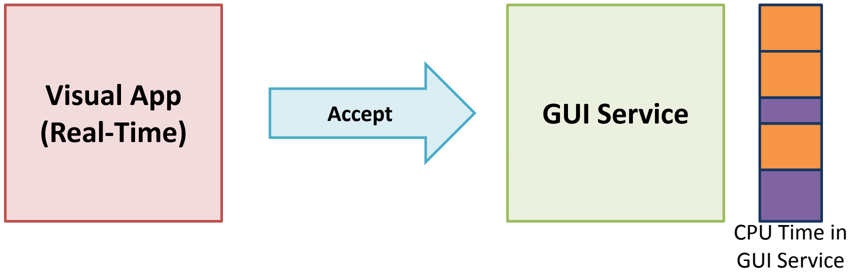
Experimental Data

Out of 4000 frames



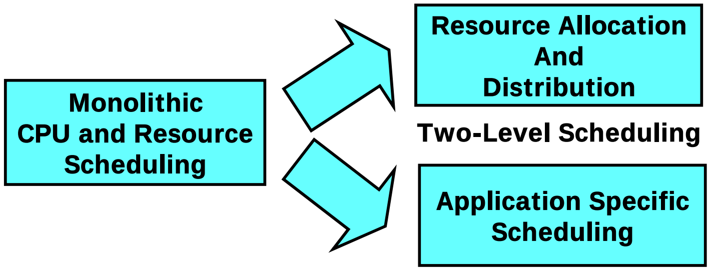
Establishing the Service Level Agreement

- Conventional Wisdom on achieving QoS
  - Painful and takes a lot of effort
  - Reason why people don't like doing real-time
- Instead: Automatic profiling of User Meaningful Actions
  - Submit complete frames for profiling (Exploration)
  - Followed by offer of SLA



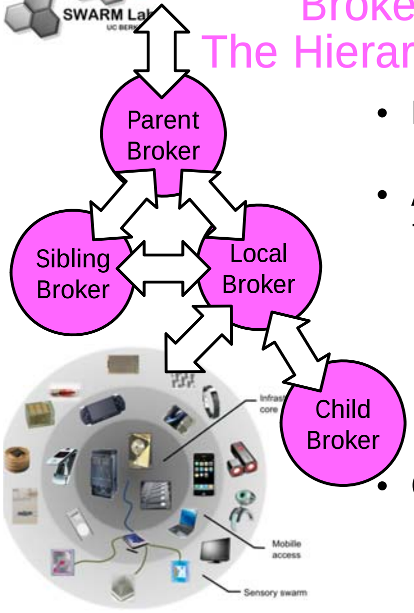
Allocation of Resources
Discovery, Distribution, and Adaptation

Two Level Scheduling: Control vs Data Plane



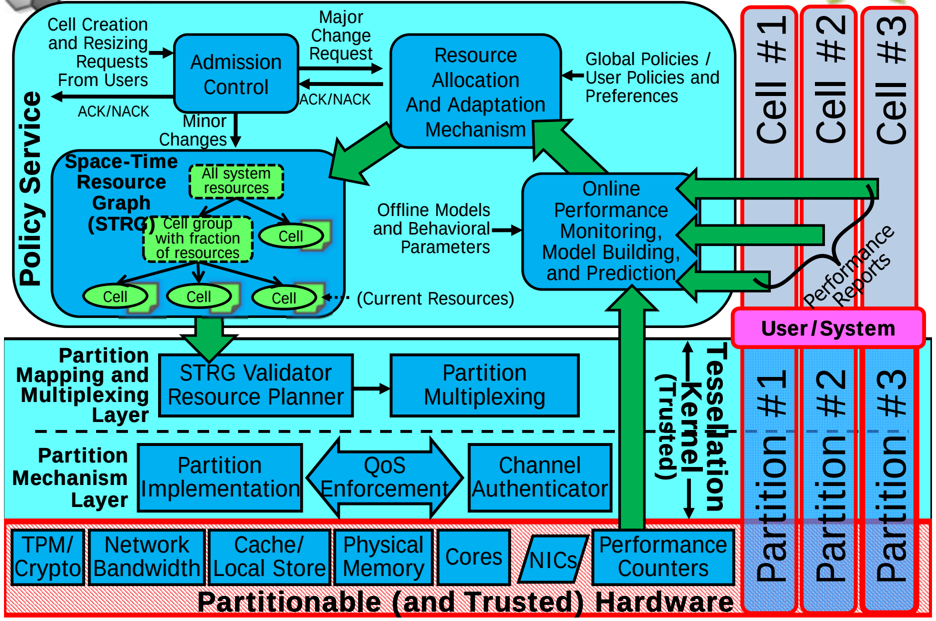
- Split monolithic scheduling into two pieces:
  - Course-Grained Resource Allocation and Distribution to Cells
    - Chunks of resources (CPUs, Memory Bandwidth, QoS to Services)
    - Ultimately a hierarchical process negotiated with service providers
  - Fine-Grained (User-Level) Application-Specific 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

Brokering Service: The Hierarchy of Ownership



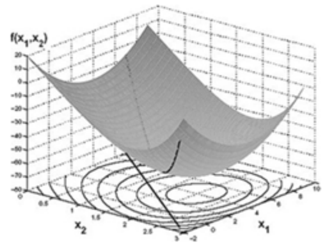
- Discover Resources in “Domain”
  - Devices, Services, Other Brokers
- Allocate and Distribute Resources to Cells that need them
  - Solve Impedance-mismatch problem
  - Dynamically optimize execution
  - Hand out Service-Level Agreements (SLAs) to Cells
  - Deny admission to Cells when violates existing agreements
- Complete hierarchy
  - Throughout world graph of applications

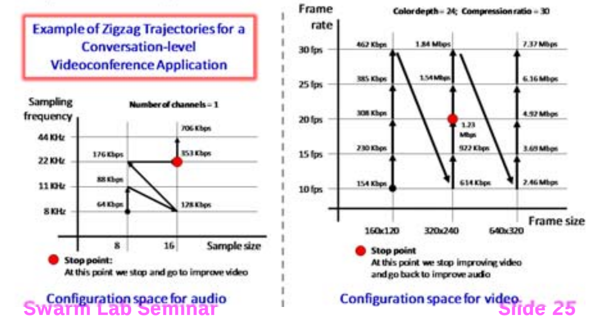
Architecture of Tessellation OS



Modeling and Adaptation

- Modeling of Applications
  - Static Profiling: may be useful with Cell guarantees
  - Multi-variable model building: Get performance as function of resources
- Adaptation according to User and System Policies
  - Convex optimization
    - Relative importance of different Cells expressed via scaling functions
  - Walk through Configuration space
    - Meet minimum QoS properties first, enhancement with excess resources





Configuration space for audio

Configuration space for video

On Toward the Swarm

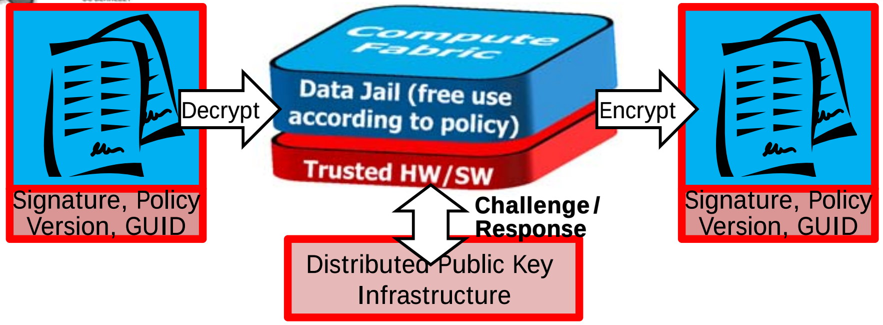
Swarm Data

- Information (Data) as a First Class Citizen:
  - Current Viewpoint: Data is byproduct of computation
  - Much Better: Data independent of computation, outlasts computation, transformed by computation
  - Computation should be the ephemeral thing!
- Fallacy: Data Resides in a Particular Location
  - A breach of the system results in loss of privacy
  - Incorrect security configuration results in loss of integrity
  - A crash results in loss of updates or new information
  - Transient routing failure results in inaccessbility

⇒Integrated, Secure, Deep Archival Storage

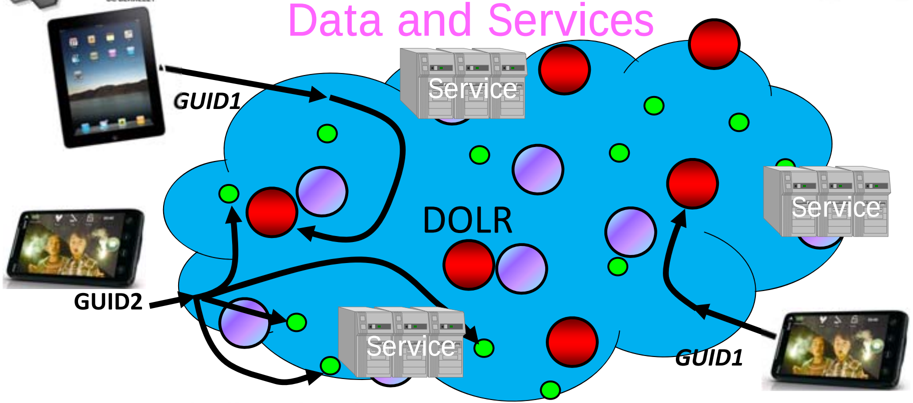
- Data available from anywhere, anytime
- Data encrypted all the time (except in authorized cells)
- Data durable by default (coding, widespread replication)

Secure Cell: Portable Secure Data



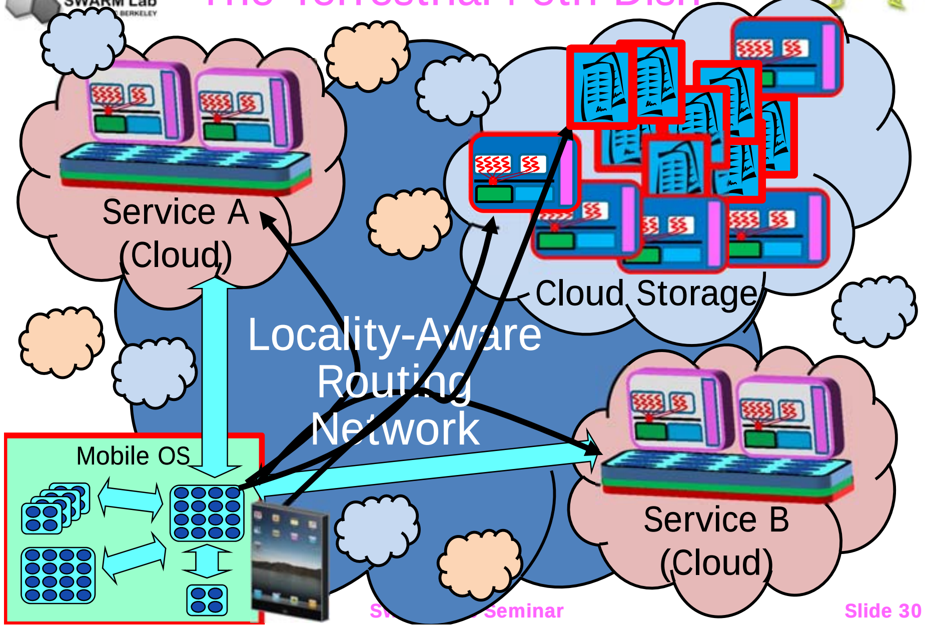
- Data divided into globally-addressable capsules
  - Addressable by unique GUID and/or metadata search
  - Conceptually stored in THE Storage Cloud (cyberspace?)
  - ⇒ If you can name it, you can use it!
- Secure Cell: Security Context as a resource
  - Data is signed, has attached policy, Optionally encrypted
  - Unwrappable only in correct trusted environment
- Key Distribution ⇒ resource management

Location Independent Data and Services



- Level of indirection in network
  - "Decentralized Object Location and Routing" (DOLR)
  - All data and services explicitly named by secure hash (Sha256?)
- Deep Archival Storage in Cloud
  - Integrated use of coding for maximum durability

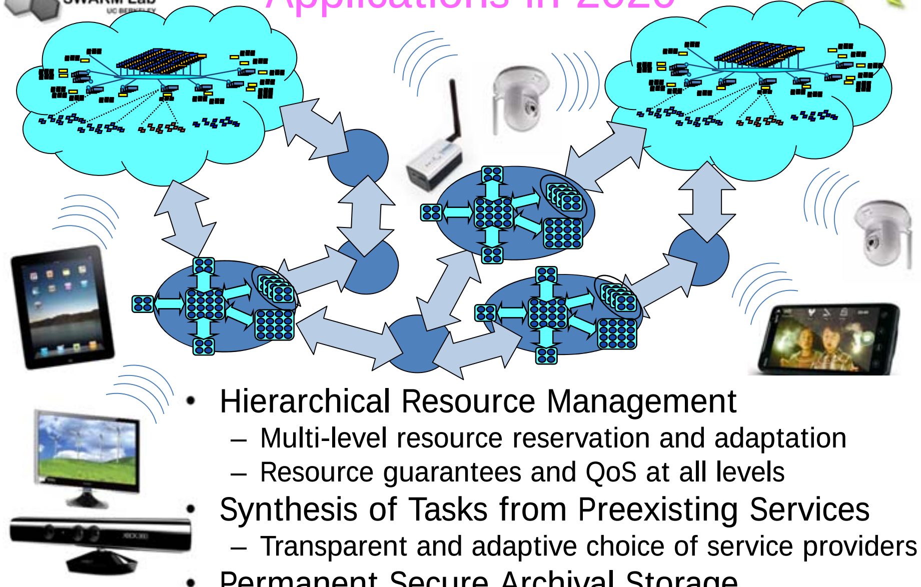
The Terrestrial Petri Dish



Cell as a Ubiquitous Primitive for the Swarm

- Cell with network interconnect is an ideal way to handle heterogeneity
  - From the outside: export services to other Cells
  - From the inside: naturally partition components along heterogeneous boundaries
- Hierarchical Resource Broker Architecture
  - Separate allocation of resources from use of resources
- Every component in system should host Cells?
  - Even sensors!?
  - What is minimal support?
    - Security Primitives
    - Communication support
- Alternative: Bare sensors do not host Cells
  - Requires minimal computational capability
- Legacy components????

Applications in 2020



- Hierarchical Resource Management
  - Multi-level resource reservation and adaptation
  - Resource guarantees and QoS at all levels
- Synthesis of Tasks from Preexisting Services
  - Transparent and adaptive choice of service providers
- Permanent Secure Archival Storage

Conclusion

- Essential ideas:
  - Resource guarantees negotiated hierarchically
  - Continual adaptation and optimization
  - Deep Archival Storage available from anywhere, anytime
  - Mobility of secure data, computation (is there a difference?)
- 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