Swarming toward the Internet of Things (via the direct route)

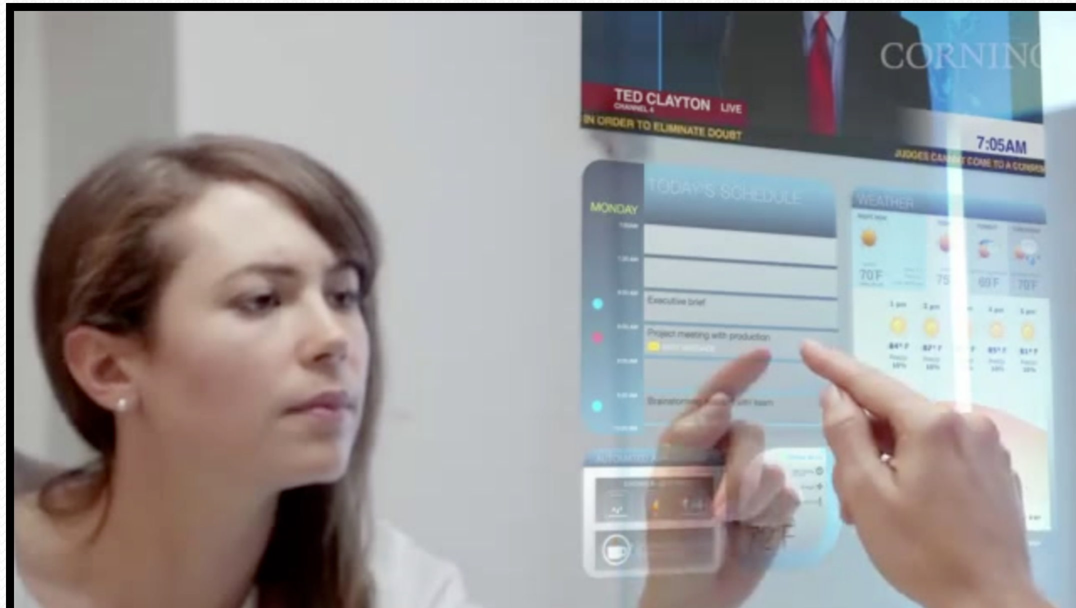
John Kubiatowicz
UC Berkeley Swarm Lab
September 29th, 2013

Disclaimer: I'm not talking about the run-of-the-mill Internet of Things

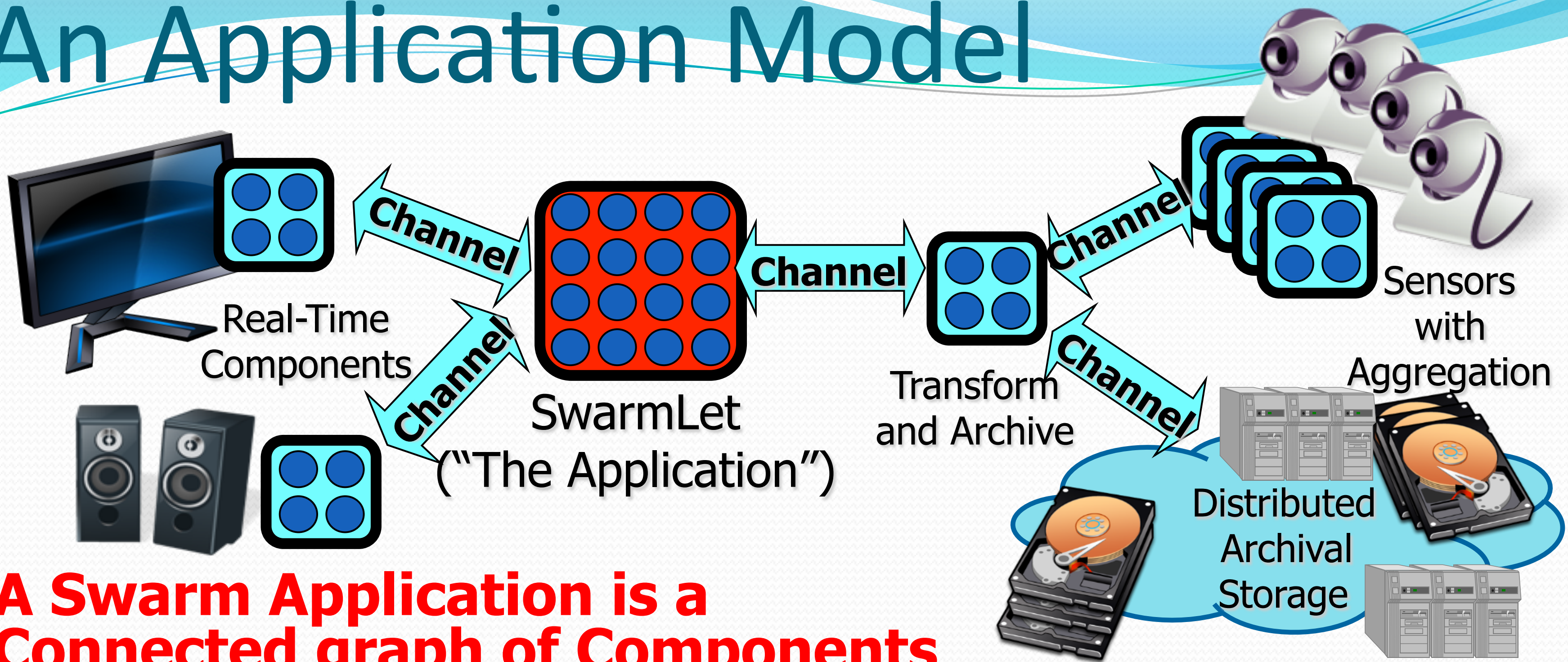
- When people talk about the IoT, they often seem to be talking about a two-level system: sensor + cloud
- We are going to talk about a locality, bandwidth, and energy-aware system for constructing globally distributed applications
  - Locality *REALLY matters*
  - At 10^{12} components (for instance), transmitting all information to and from the cloud would be impossible

Example: a Smart Space

- 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
- How do we hope to organize this complexity?
  - Not via Stovepipe solutions! Today's typical solution!
  - Need something more global!



An Application Model

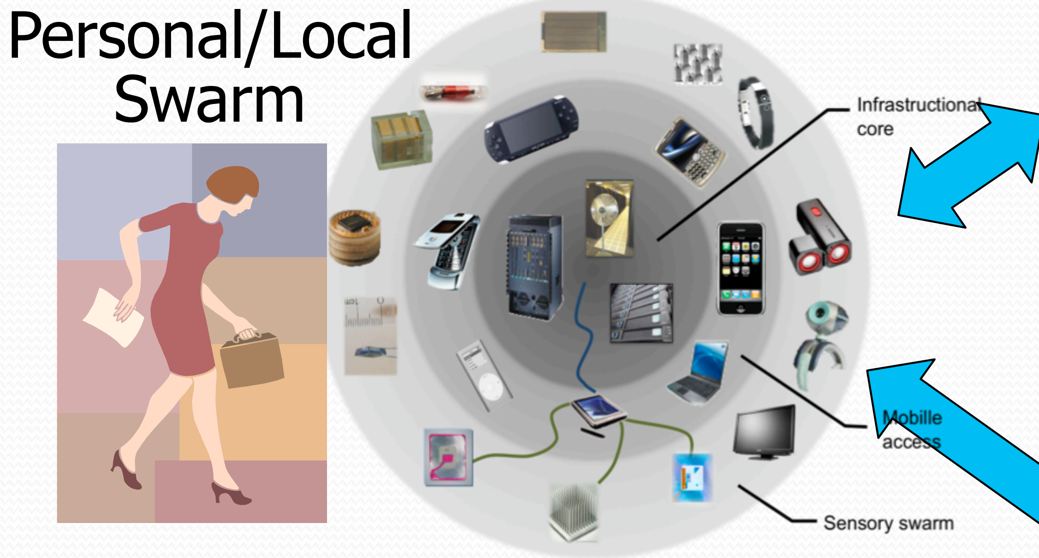


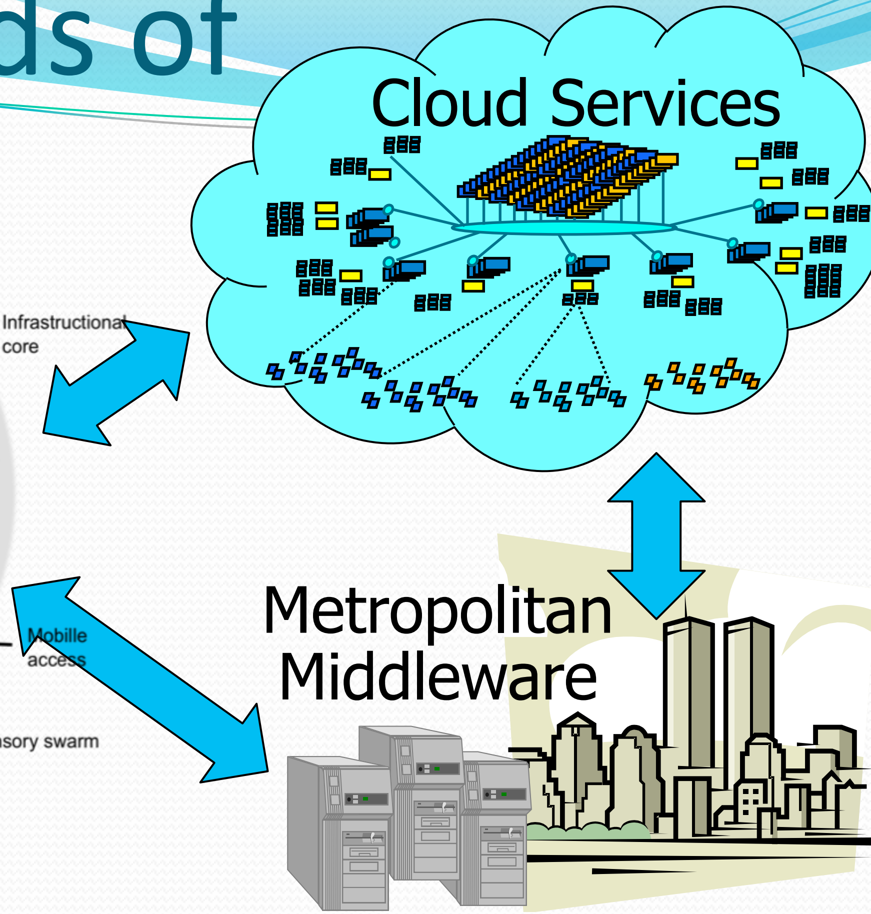
- **A Swarm Application is a Connected graph of Components**
  - **Globally distributed, but locality and QoS aware**
  - **Avoid Stovepipe solutions through reusability**
- Many components are *Shared Services* written by programmers with a variety of skill-sets and motivations
  - Well-defined semantics and a managed software version scheme
  - Service Level Agreements (SLA) with micropayments
- Many are "Swarmlets" written by *domain programmers*
  - They care *what* application does, not *how* it does it

SWARMLETs

- SWARMLET: a software component written by domain programmer that is easy to write but exhibits sophisticated behavior by exploiting services distributed within the infrastructure
- Swarmlets specify their needs in terms of human-understandable requirements
  - Necessary Services, Frame rates, Minimum Bandwidths
  - Locality, Ownership, and Micropayment parameters for sensors and/or data
- Swarmlets may evolve into Shared Services
- Programmers of Services used by Swarmlets think in terms of contracts provided to swarmlets

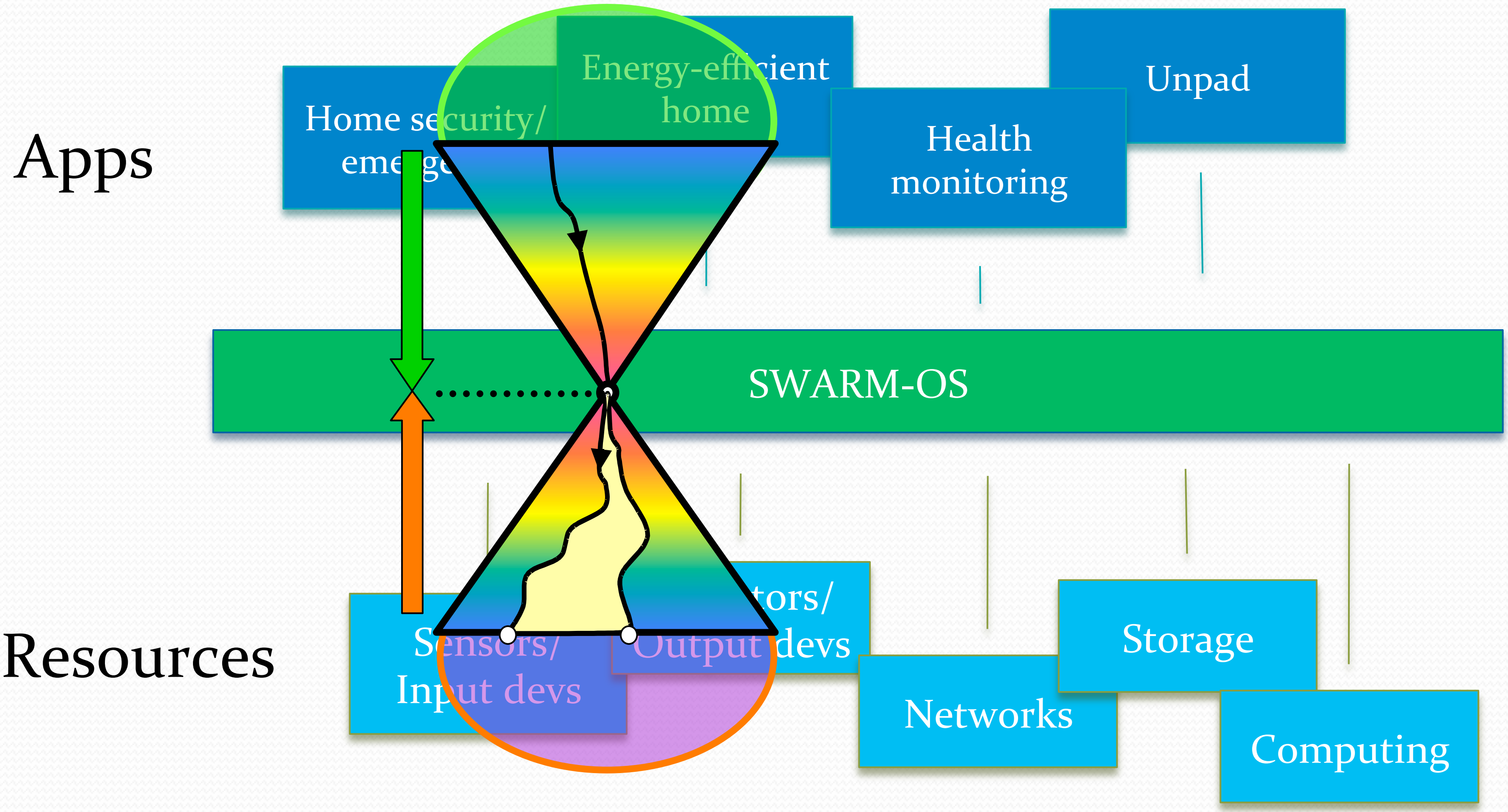
Meeting the needs of the Swarm



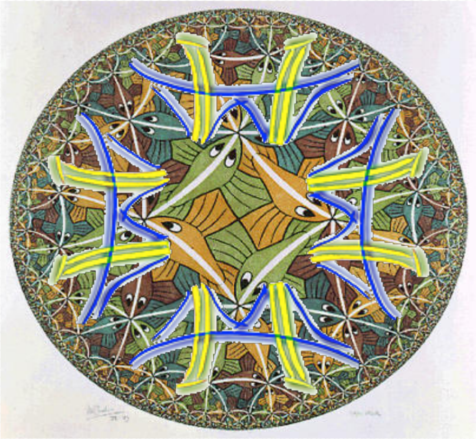


- Discover and Manage resource
- Integrate sensors, portable devices, cloud components
- Guarantee responsiveness, real-time behavior, throughput
- Self-adapt to failure and provide performance predictability
- Secure, high-performance, durable, available information
- Monetize resources when necessary: micropayments

The Missing Link?



SWARM-OS: A mediation layer that discovers resources and connects them with applications



The Cell Model for Swarm Components

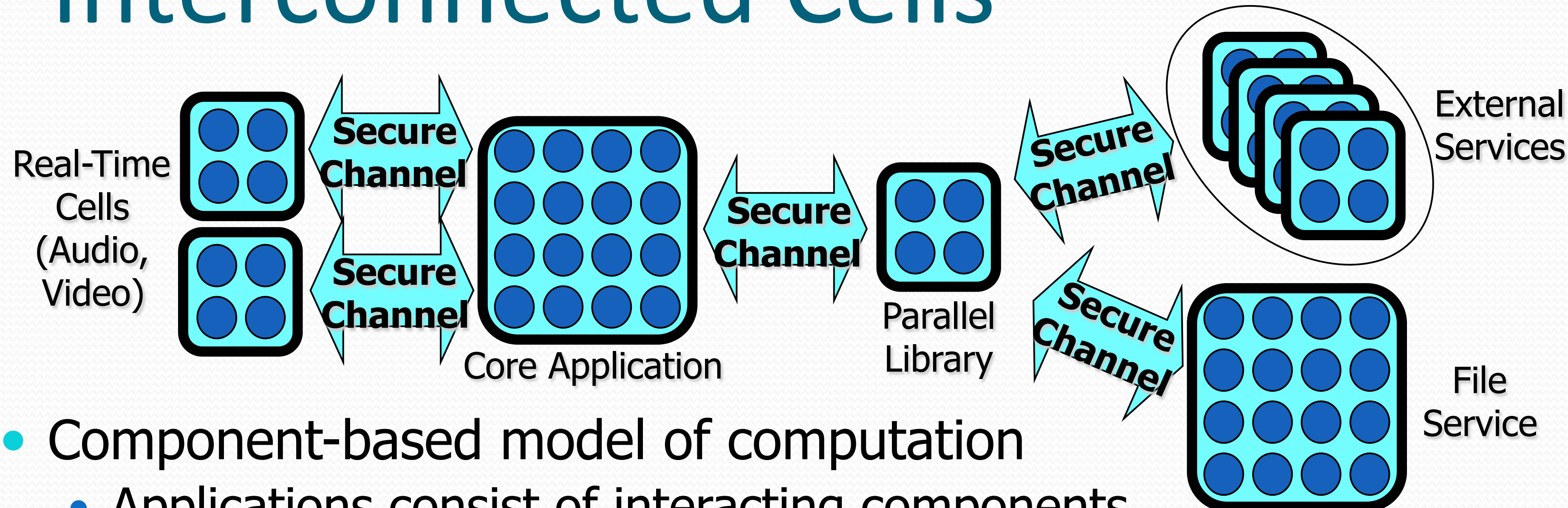
A Resource-Centric Approach: Guaranteeing Resources

- What might we want to guarantee?
  - 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)
- "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

New Abstraction: the Cell

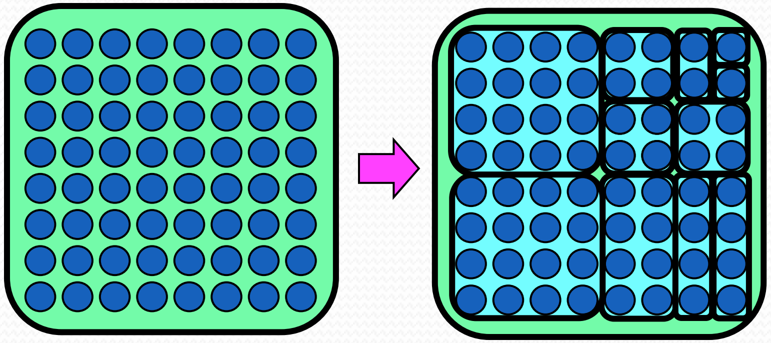
- Properties of a Cell
  - A user-level software component with guaranteed resources
  - Has full control over resources it owns ("Bare Metal")
  - Contains a set of secured channel endpoints to other Cells
  - Contains a security context which may protect and decrypt information
- When mapped to the hardware, a cell gets:
  - Gang-schedule hardware thread resources ("Harts")
  - Guaranteed fractions of other physical resources
    - 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

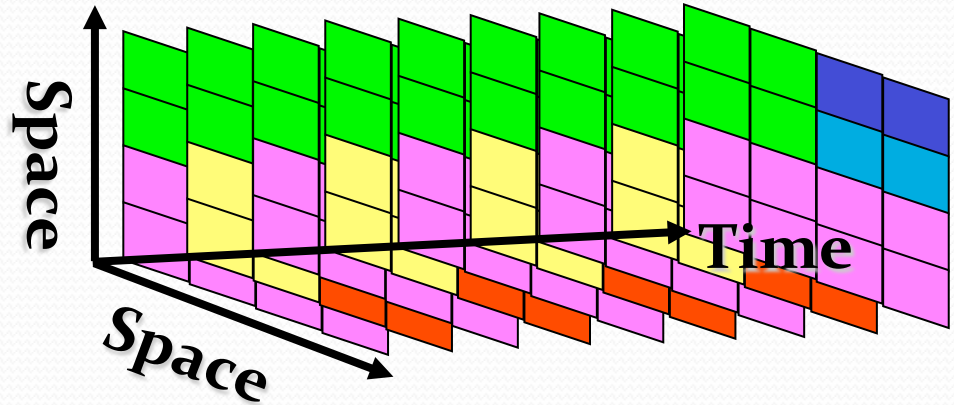
Applications Composed of Interconnected Cells



- 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!

Example: Cells on Multicore via 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

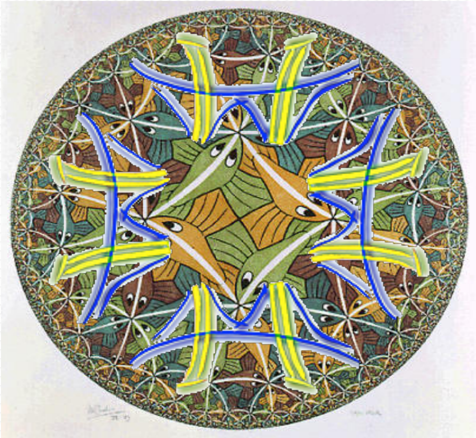
- Partitioning varies over time
  - Fine-grained multiplexing and guarantee of resources
    - Resources are gang-scheduled
- Controlled multiplexing, not uncontrolled virtualization
- Partitioning adapted to the system's needs

Secure Cell: Trusted Swarm Platform

- **External Data Encrypted All The Time**
- Only decrypted in "Data Jails" (trusted platform)
  - Build in hardware or in software with secure attestation
  - Data leaving cell automatically reencrypted
- Trusted Platform given keys to do its work
  - Keys never given out to application software
- Similar idea: Hardware micropayment support

What would this mean for the Swarm?

- Where are Cells in the swarm?
  - In the Cloud and Fog at the edges of the cloud
  - Mobile devices with significant processing
- What about Sensors or Actuators?
  - Very little processing, but ability to provide guarantees could be quite important
  - QoS in form of probabilistic guarantees
  - Resistance to denial of service
- What about the network? Is it a Service?
  - Probabilistic guarantees?
  - Wireless channel reservation?
  - Flow-level guarantees (AVB)?
- Is there a minimal hardware base for swarm integration?
  - QoS enforcement, Secure Cell, Network guarantees

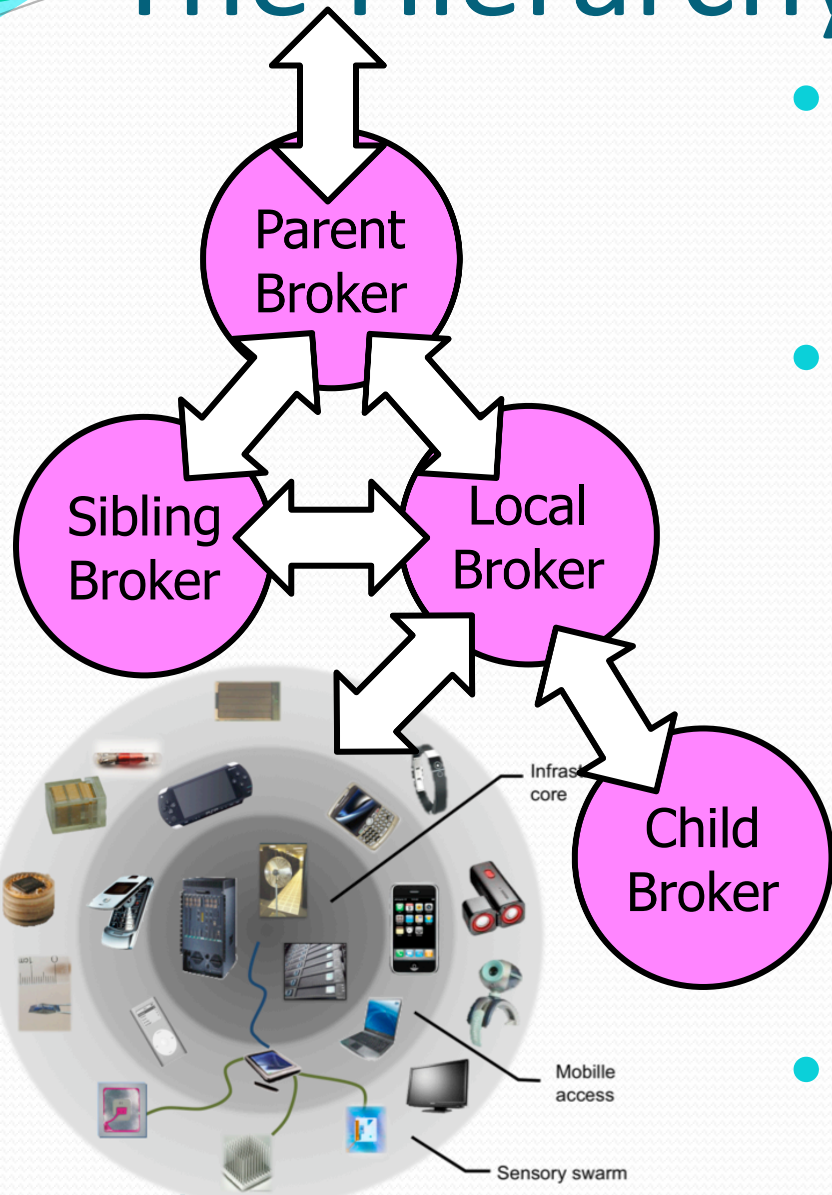


Resource Distribution and Adaptation

Resource Discovery and Ontology

- Dynamically discover resources, services, and cyber-physical components (sensors/actuators) that meet application requirements
  - Find *local* components that meet some specification
  - Use ontology to describe exactly what component do
  - Distribute these resources (or fractions of services) to application cells in order to meet QoS requirements
- Many partial solutions out there, no complete solutions
  - Must deal with locality (discover local items) while at same time dealing with remote (global) services
  - Must gracefully handle failover of components
- One important aspect is that resources must be handed out only to authorized users
  - Authorization can involve ownership, micropayments, etc..

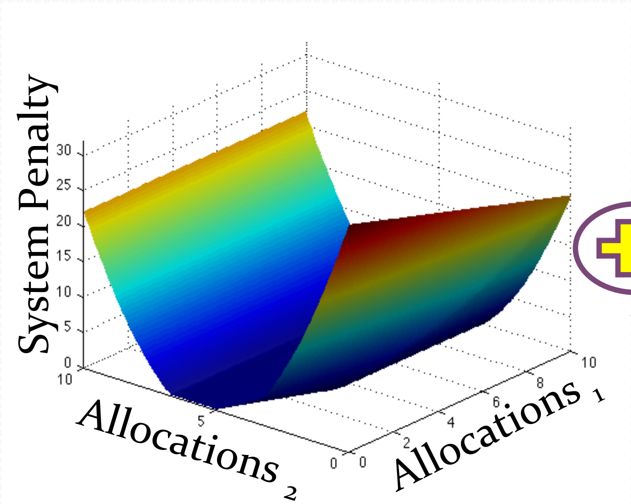
Brokering Service: The Hierarchy of Ownership

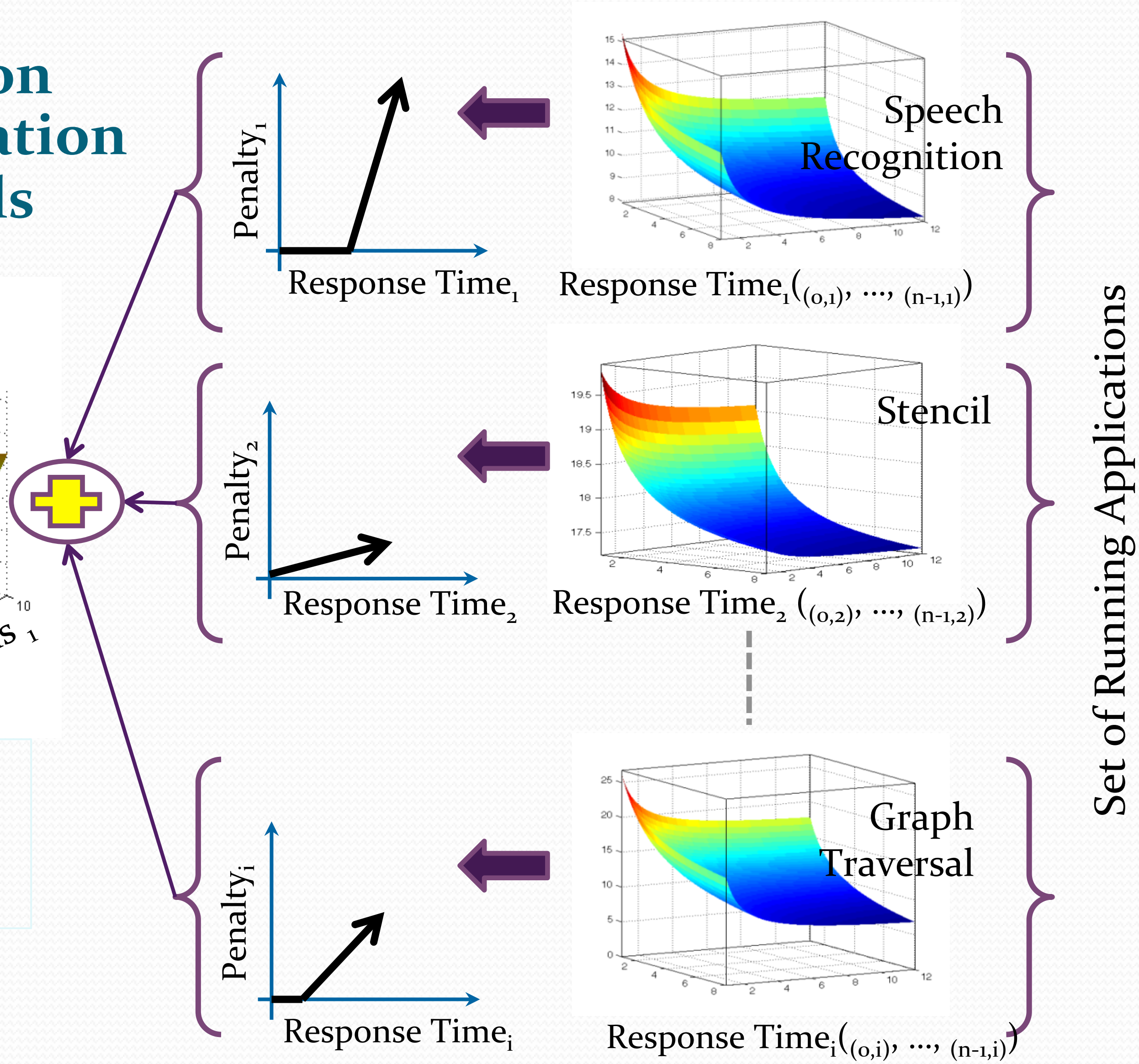


- Discover Resources in "Domain"
  - Devices, Services, Other Brokers
  - Resources self-describing?
- Allocate and Distribute Resources to Cells that need them
  - Solve Impedance-mismatch problem
  - Dynamically optimize execution
  - Hand out Service-Level Agreements (SLAs) to Cells
  - Deny admission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

Example: Convex allocation (PACORA)

Convex Optimization with Online Application Performance Models

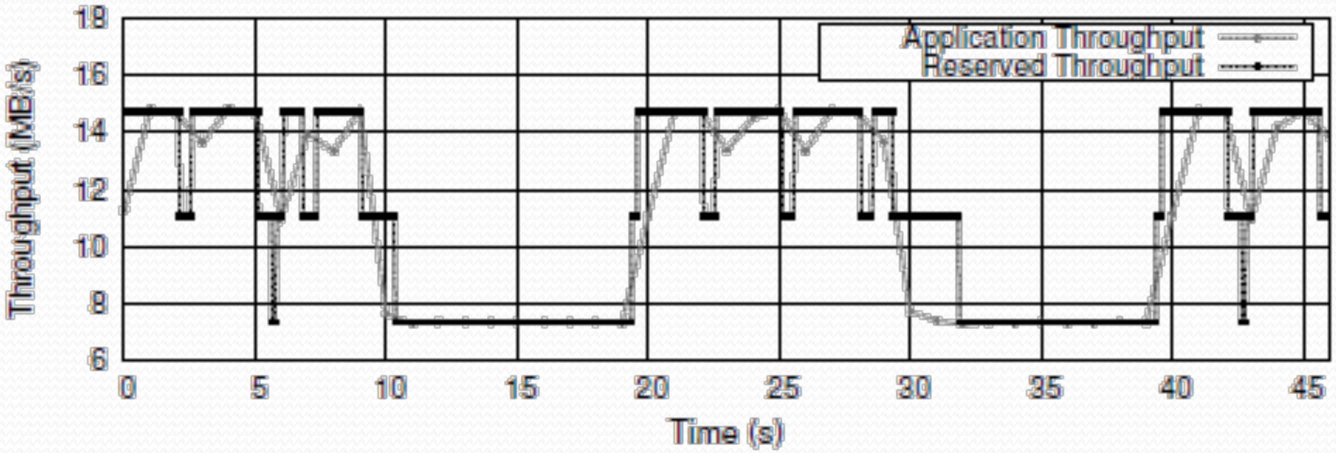




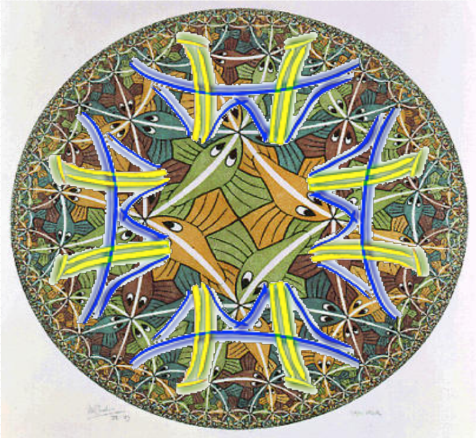
Continuously minimize the penalty of the system
(subject to restrictions on the total amount of resources)

Example: Feedback allocation

- Utilize dynamic control loops to fine-tune resources
- 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

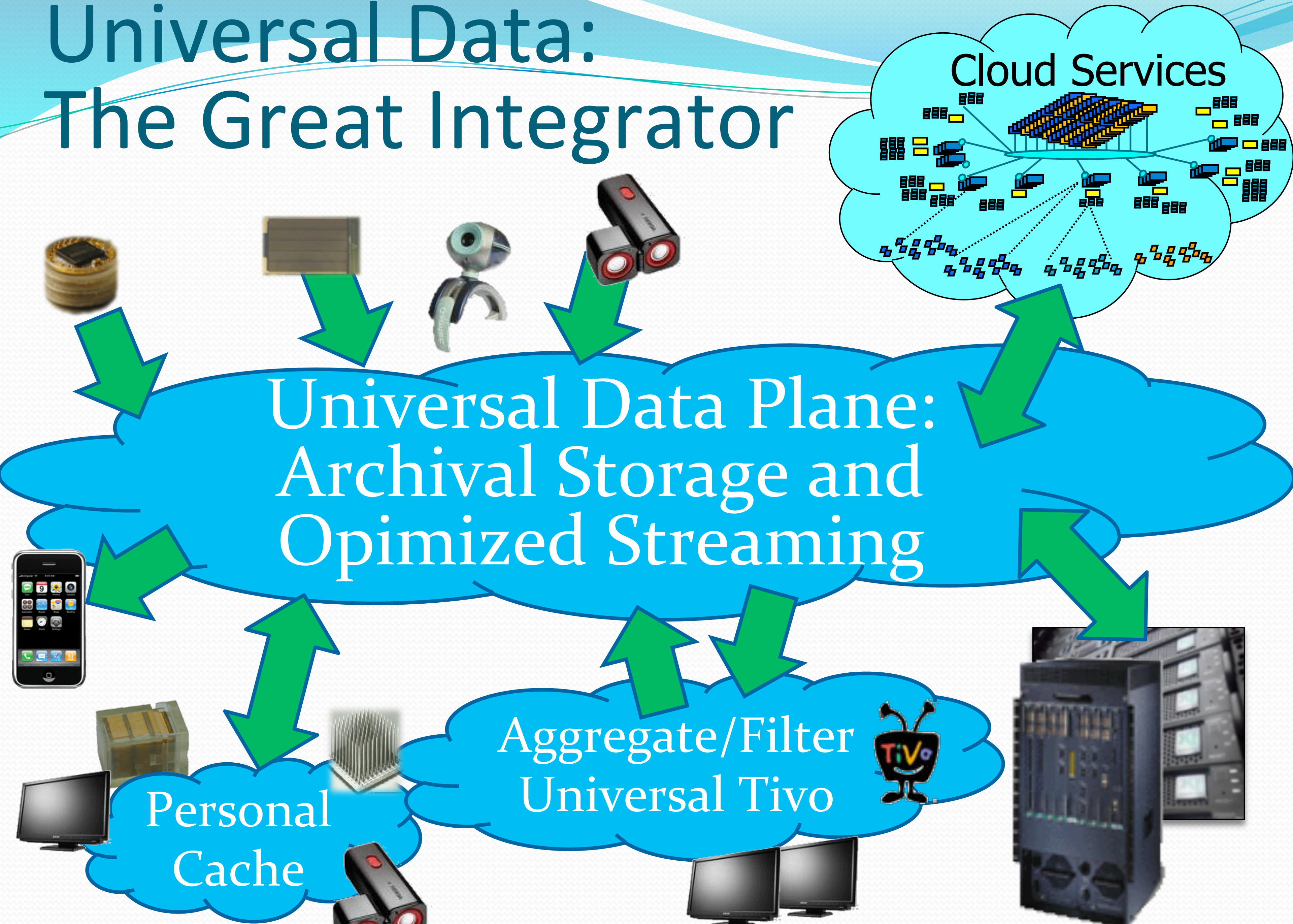


The Internet for the Internet of Things

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 information
  - Integrity, Privacy, Availability, Durability
  - Hardware to prevent accidental information leakage
- Permanent state handled by Universal Data Storage, Distribution, and Archiving
- We need a new Internet for the Internet of Things
  - Communication and Storage are really duals
  - Why separate them?

Universal Data:
The Great Integrator



Internet for the Internet of Things

- Duality between communication and storage
  - Why explicitly distinguish them?
  - The "Data Grid" equivalent to the "Power Grid"
  - All data is is *read-only* and time stamped at the time that it enters the grid and preserved as long as it stays in the grid
- Provide a large flat namespace for routing to endpoints independent of their location
  - Endpoints can be services, sensors, or archival objects
  - Automatically locate close objects with given endpoint (when there are multiple of them such as cached read-only data)
- Dynamic Optimization: Gain advantages normally available only to large internet providers
  - Generate optimized multicast networks when necessary
  - Construct content distribution networks (CDNs) on the fly
- Security, authentication, privacy, micropayments

Conclusion

- Advance the Swarm by making it easy for programmers to construct applications
  - Distributed application model focused on QoS, micropayments, stable services
  - Sophisticated applications built with Swarmlets
- Cell Model
  - User-Level Resource Container with guaranteed resources
  - Hardware-Enforced Security Context
- Dynamic Resource Discovery, Brokerage, Optimization
- Universal Data Plane
  - Provide a better Internet for the Internet of Things
  - Security, dynamic optimization, caching, archival storage
- Tessellation OS: http://tessellation.cs.berkeley.edu
  SwarmLab: http://swarmlab.eecs.berkeley.edu