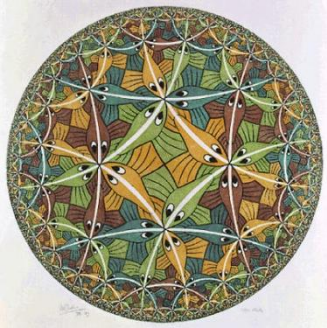


Resource Management in Tessellation OS

Juan A. Colmenares, Sarah Bird, Henry Cook, Paul Pearce, David Zhu, Krste Asanovic, and John Kubiatowicz

[ParLab]

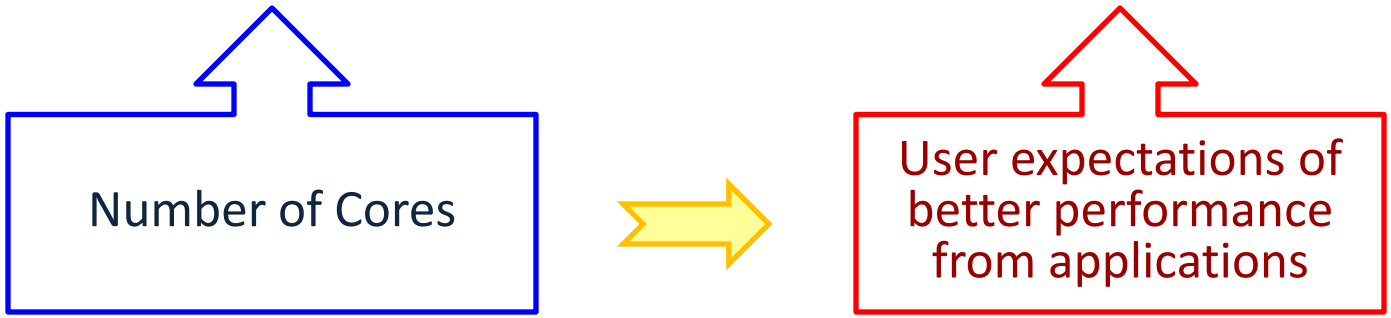
John Shalf and Steven Hofmeyr

[Lawrence Berkeley National Laboratory]

Presented by Juan A. Colmenares at the ParLab Summer Retreat
May 25th, 2010
Chaminade Resort & Spa , Santa Cruz, CA

Challenges for Client Devices in the Many-core Era

Just Some of Them



- *Natural Response*: Parallelization of client applications to improve performance
- Increasing appetite for
  - Responsive user interfaces
  - High-quality multimedia applications with real-time requirements
- Applications will consist of a variety of components with different performance requirements
  - e.g., Web browser + multimedia plug-ins

What's the Problem with Current OSs?

- They (often?) do not allow expression of performance requirements
  - Minimal frame rate, minimal memory bandwidth, minimal QoS from system services, real-time constraints, …
  - No clean interfaces for reflecting these requirements
- They (often?) do not provide guarantees that applications can use
  - They do not provide performance isolation
  - Resources can be removed or decreased without permission
  - Maximum response time to events cannot be characterized
- They (often?) do not provide fully custom scheduling
  - In a parallel programming environment, ideal scheduling can depend crucially on the programming model
- They (often?) do not …

The advent of many-cores both:

- Exacerbates these deficiencies with a greater number of shared resources
- Provides an opportunity to fundamentally restructure OSs

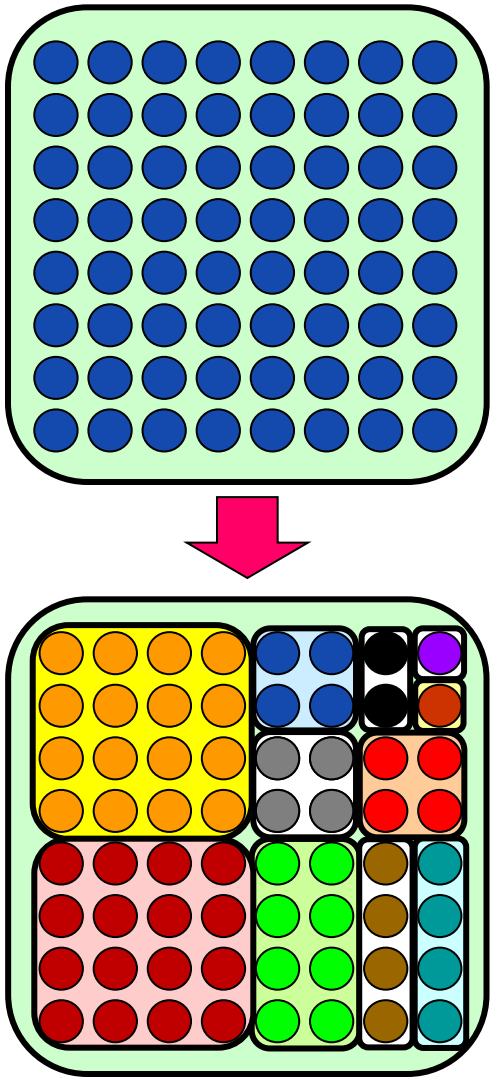
Some of Our Basic Goals in Tessellation OS

- Support a simultaneous mix of high-throughput parallel, interactive, and real-time applications
  - Improve performance by exploiting many-core hardware
  - Enable high responsiveness
  - Facilitate providing timing guarantees
- Allow applications to consistently deliver performance improvements
  - In presence of other applications with conflicting requirements
- Rapid adaptation
  - To changes in the application mix and resource availability
- Scalability

Spatial Partitioning

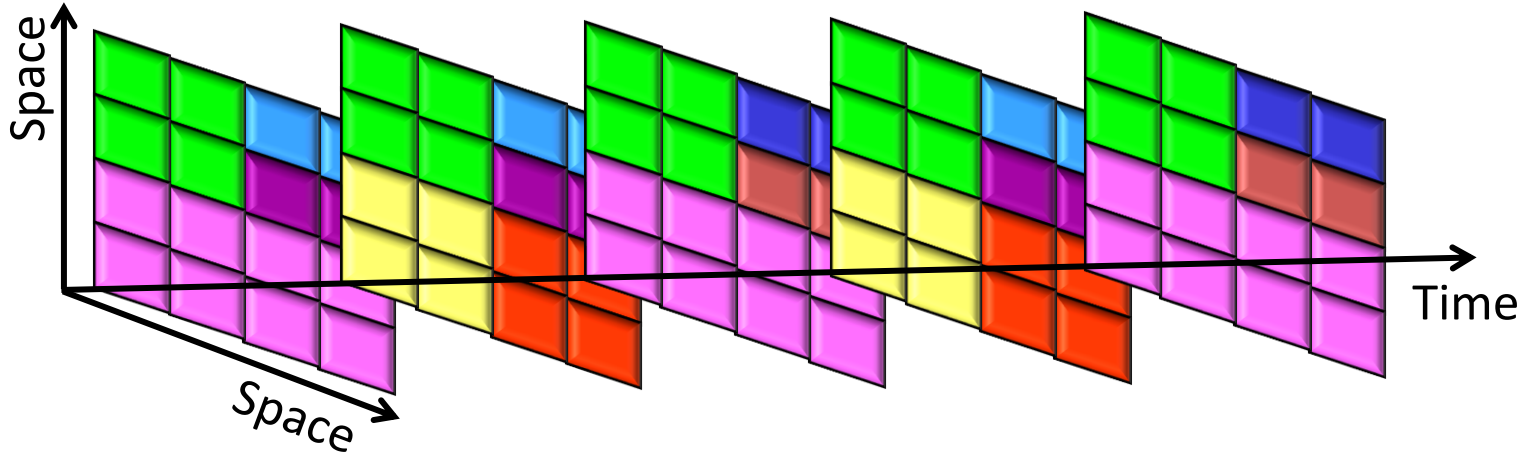
Key for Performance Isolation

- Spatial Partition: Group of processors acting within a *firm* hardware boundary
  - Communication between partitions is controlled
- Each partition receives a vector of dedicated basic resources
  - A number of processing elements (e.g., cores)
  - A portion of physical memory
  - A portion of shared cache memory
  - A fraction of memory bandwidth
- A partition may also receive
  - Exclusive access to other resources
    - e.g., certain hardware devices
  - Guaranteed fractional services from other partitions
    - e.g., network service and file service



Space-Time Partitioning

Spatial Partitioning Varies over Time



- Some partitions persist while others change with time
- Partitions can be time multiplexed
  - Resources are gang-scheduled
  - Multiplexing at a coarse granularity
- Partitioning adapts to the needs of the system

Two-level Scheduling

Monolithic
CPU and Resource Scheduling

Coarse-grained
Resource Allocation and Distribution
(Level 1)

Global decisions about allocating chunks of resources to partitions

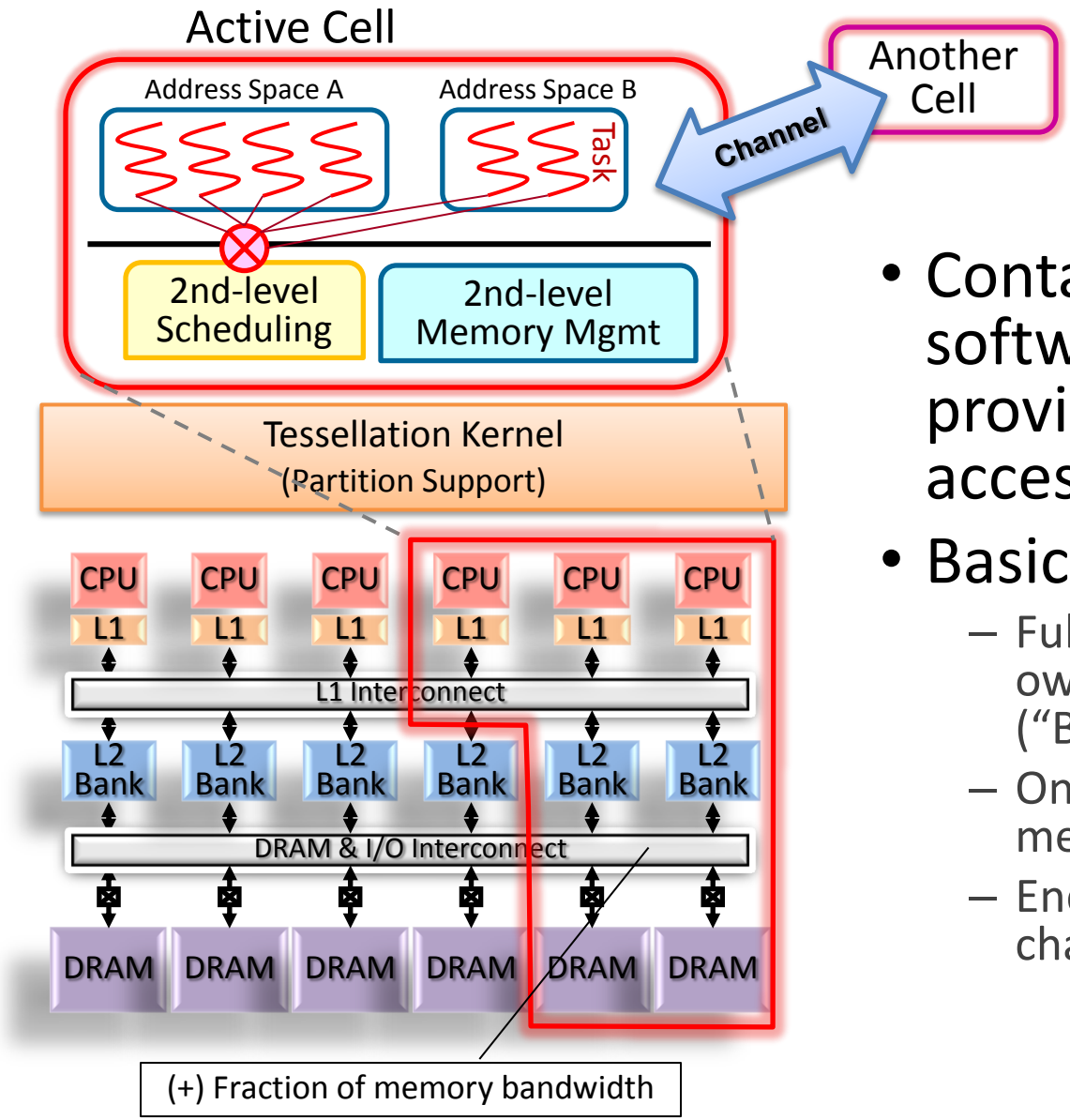
Fine-grained
Application-specific Scheduling
(Level 2)

Local decisions about resource usage within the partition

No interference from components in other partitions

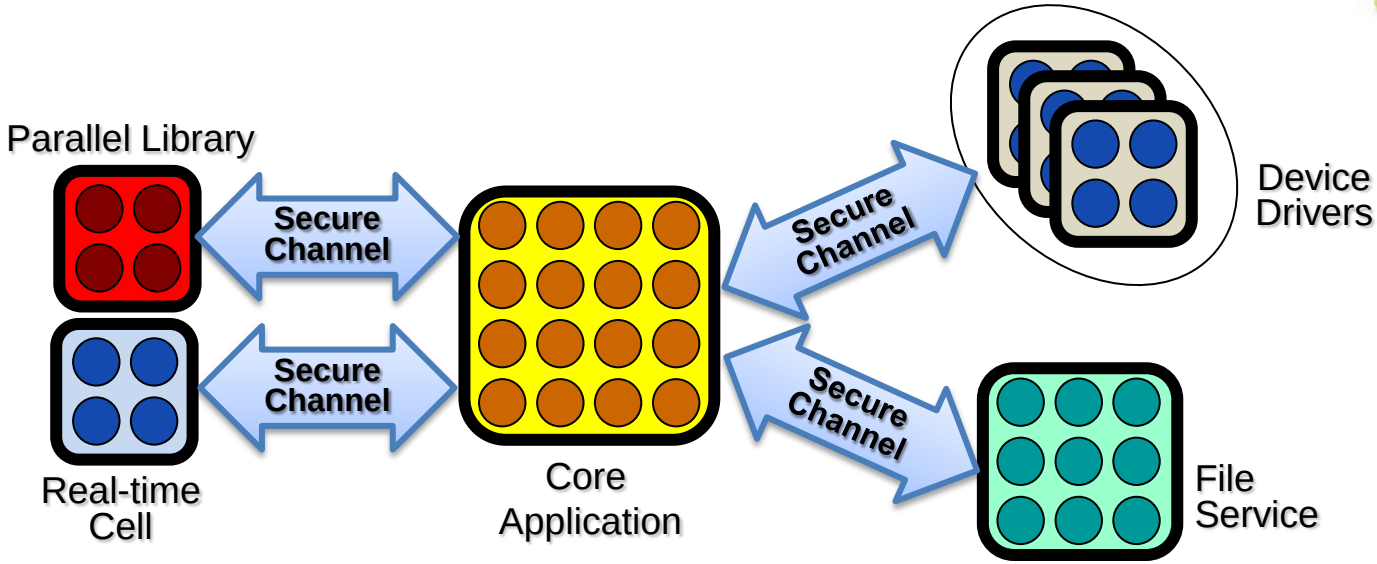
The Cell: Our Partitioning Abstraction

Defining the Partitioned Environment



- Container for parallel software components providing guaranteed access to resources
- Basic properties of a cell
  - Full control over resources it owns when mapped to hardware ("Bare Metal")
  - One or more address spaces (or memory protection domains)
  - Endpoints of communication channels to other cells

Inter-Cell Channels

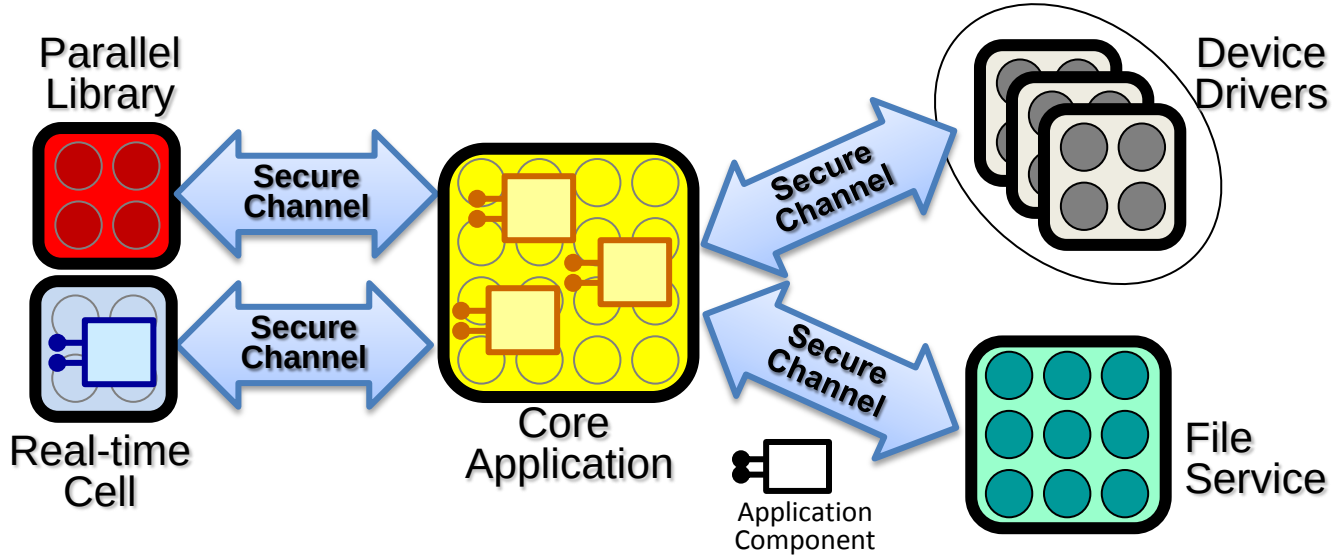


- Provide efficient and non-blocking message passing between different cells
  - Application Component ↔ Application Component
  - Application Component ↔ Service
- Communication controlled for QoS and security enforcement

Basis of a Component-based Model
with Composable Performance

Deployment Descriptor

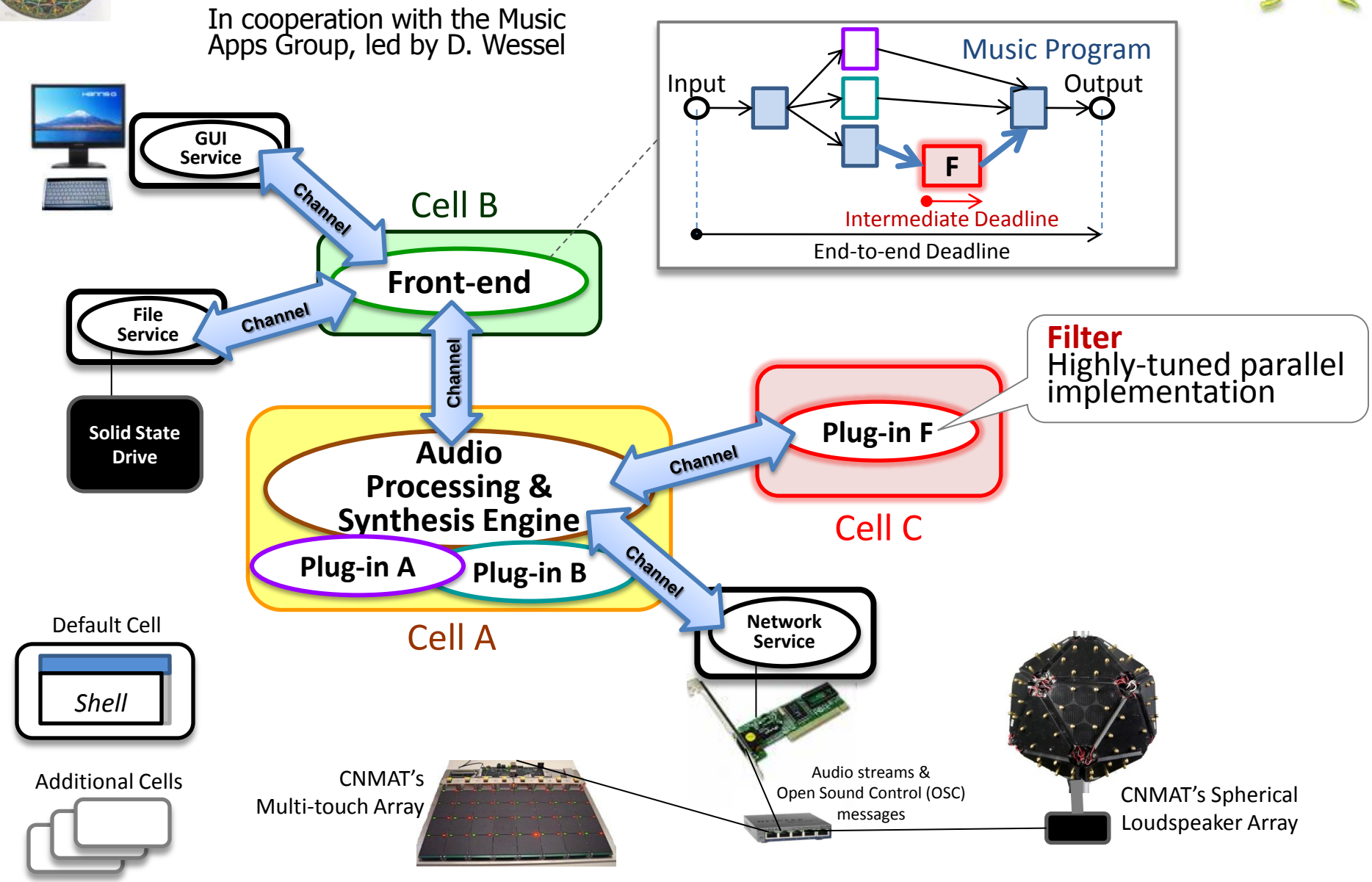
```
Application = MyApp
Comp 1 {
- host-cell = "CellA"
- …
}
…
Comp 4 {
- host-cell= "CellB"
-…
}
```



- Applications = Set of interacting components deployed on different cells
  - A Deployment Declarative Language helps hide the complexity involved
- Applications split into performance-incompatible and mutually distrusting cells with controlled communication
- OS services are independent servers

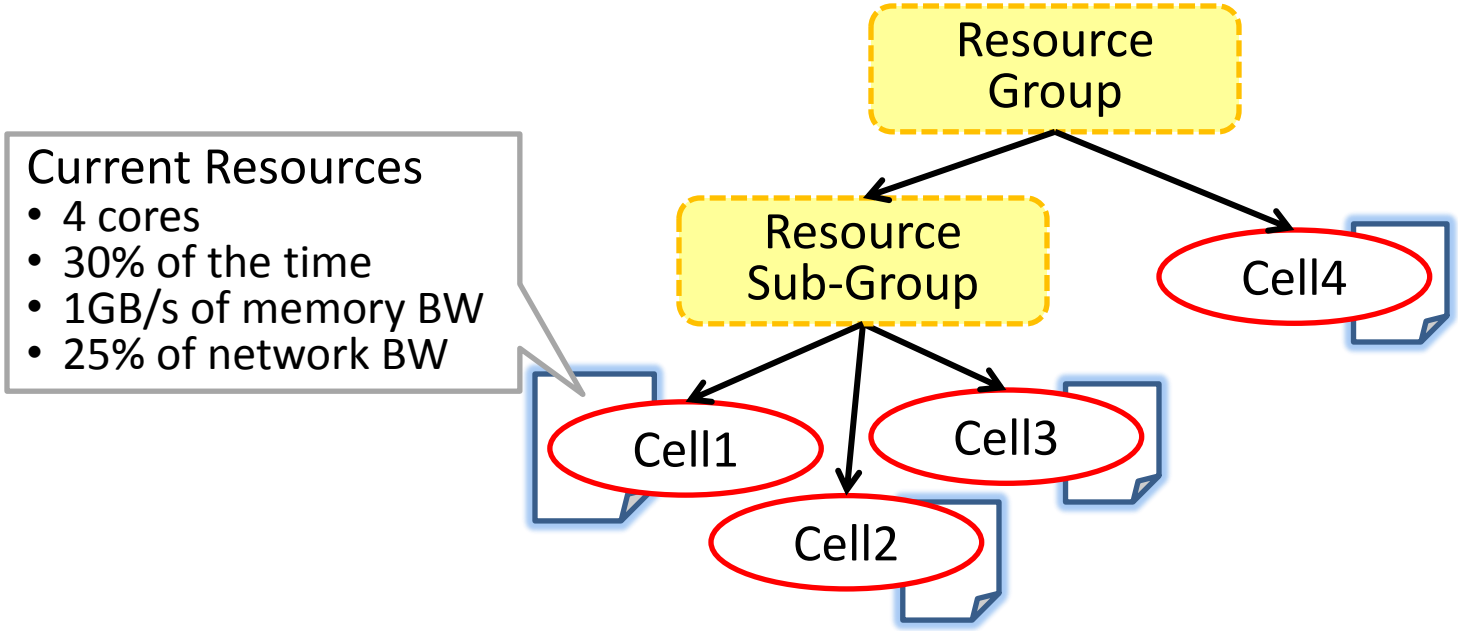
Example of a Music Application

In cooperation with the Music Apps Group, led by D. Wessel



Space-Time Resource Graph (STRG)

Contains the Current Distribution of System Resources to Cells



- Leaves hold the resource assignments of cells
- Interior Nodes = Resource Groups
  - Hold resources to be shared by children
  - “Pre-allocated” resources can be shared as excess until needed

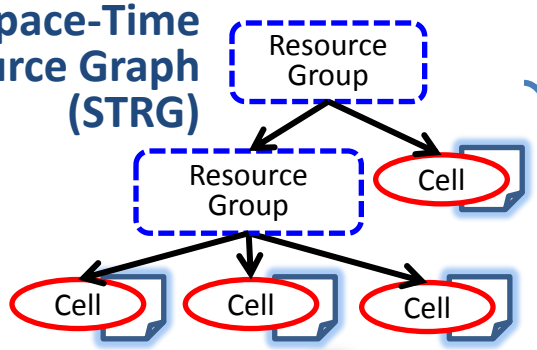
Cell-Management Software Architecture

A Simplified View

Policy Service

Performs Resource Allocation, Adaptation, and Admission Control for Cells

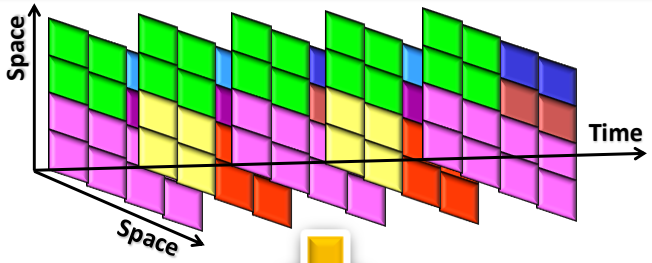
Space-Time Resource Graph (STRG)



Contains the current distribution of system resources to Cells

Mapping & Time-Multiplexing Layer

Implements the STRG

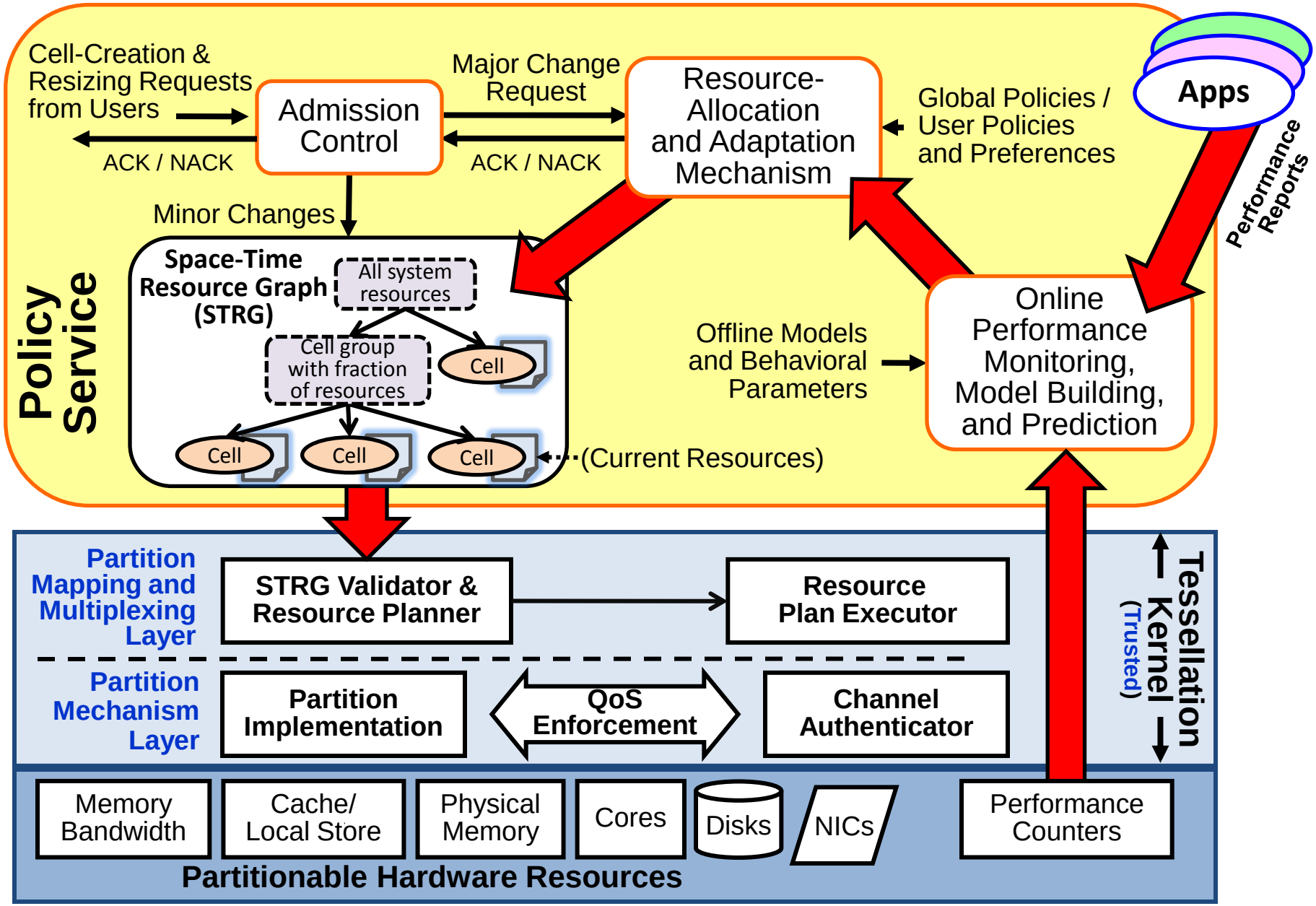


Partition Mechanism Layer

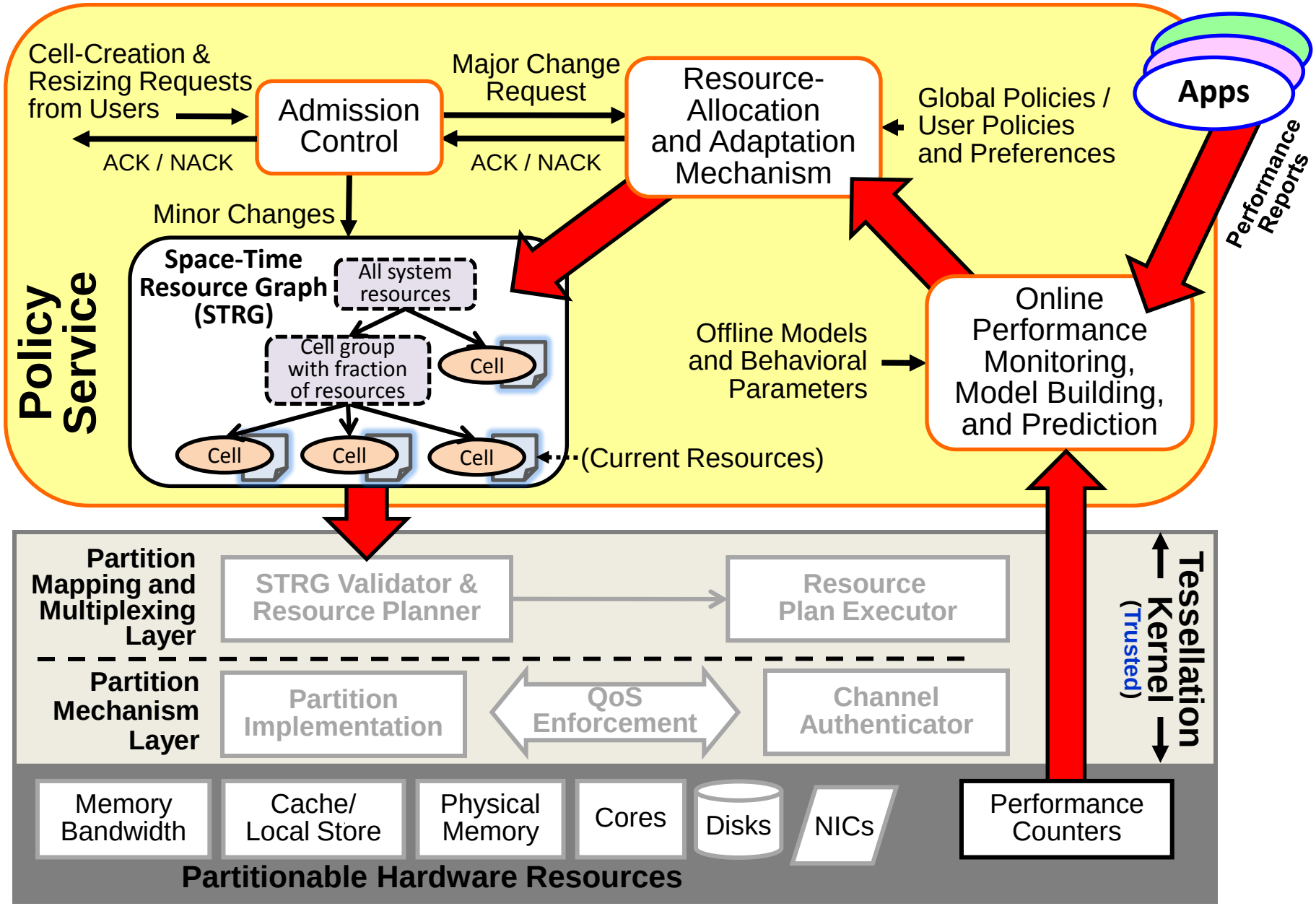
Implementation of Partitions and Channels

Tessellation Kernel (Trusted)

System-wide Adaptation Loop

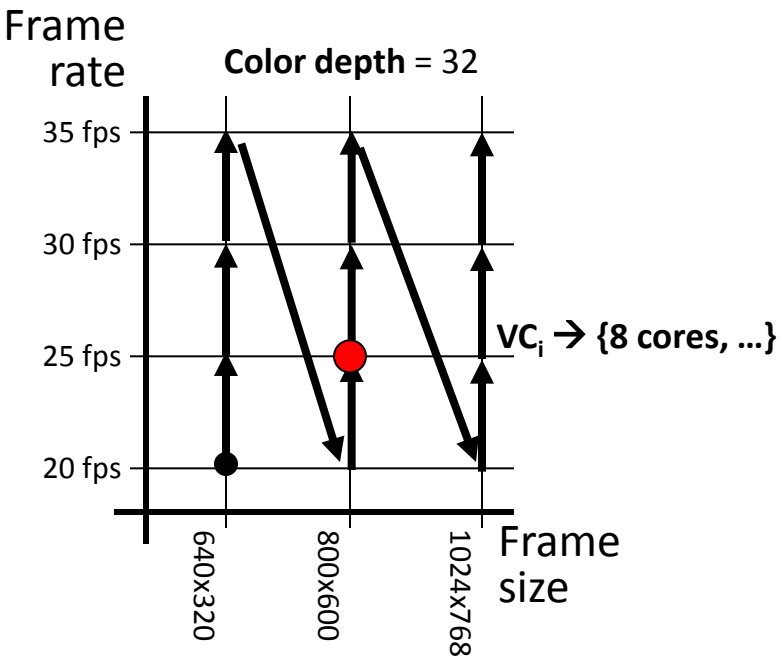


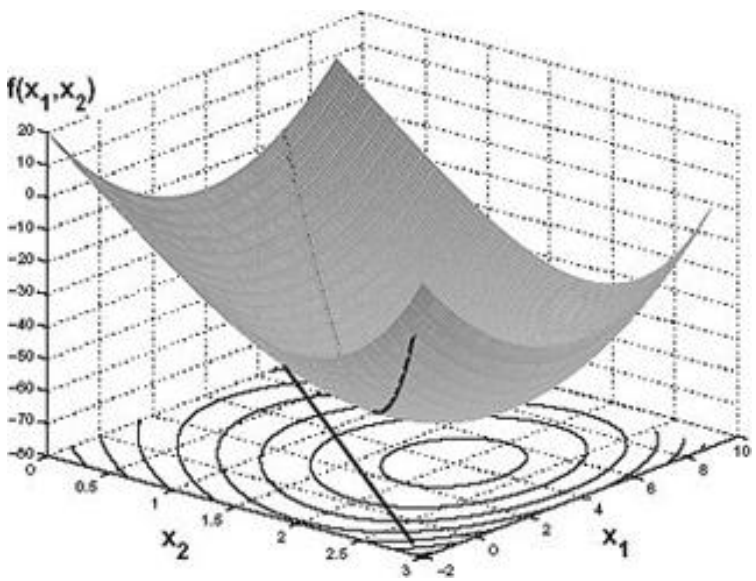
System-wide Adaptation Loop



Adaptation Policies

From a Wide Variety, Several Alternatives Are Being Investigated





- Walk through a predefined set of configurations

Application-specific

[J.A. Colmenares et al. ISORC 2010]

- Minimize the "urgency" of the system via convex optimization

System-wide

[Burton Smith (MSR), Keynote at IPDPS 2010]

An Example of

A Simple Adaptation Policy

[J.A. Colmenares et al. ISORC 2010]

Configuration Space for Video

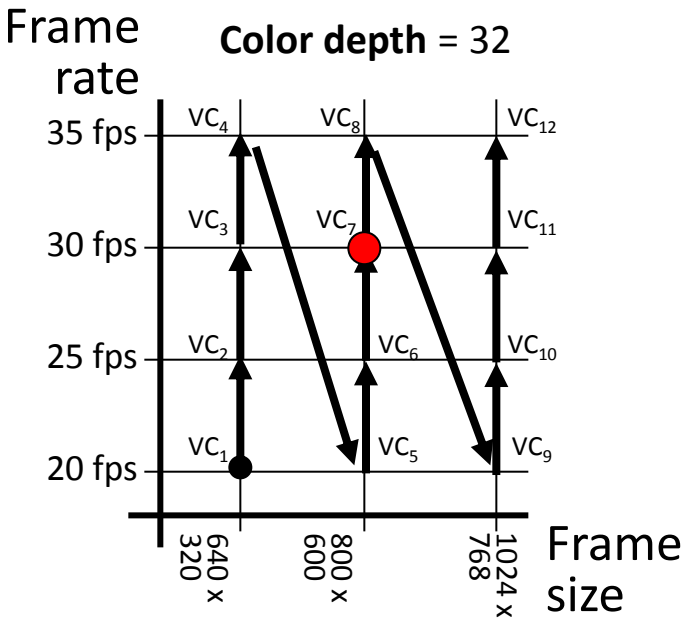


Table of Predefined Configurations

(in the increasing order of preference)

Conf.	Params	Requirement Vectors
…	…	…
VC_6	{25 fps, 800x600, …}	$RV_{(6,1)}$ = {8 cores, … , …} $RV_{(6,2)}$ = { …, …., ….}
VC_7	…	…
…	…	…

Requirement vectors can be derived via static profiling

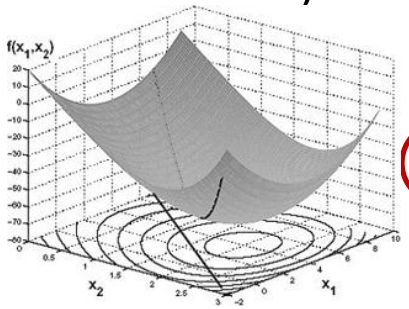
- Adaptation Strategy
  - Select the best configuration that can be sustained under the current conditions
- Application-specific, very simple, minimal overhead, and high effort in application profiling

Minimizing the Urgency of the System

A More Sophisticated Adaptation Policy

[Burton Smith (MSR), Operating System Resource Management (Keynote), IPDPS 2010]

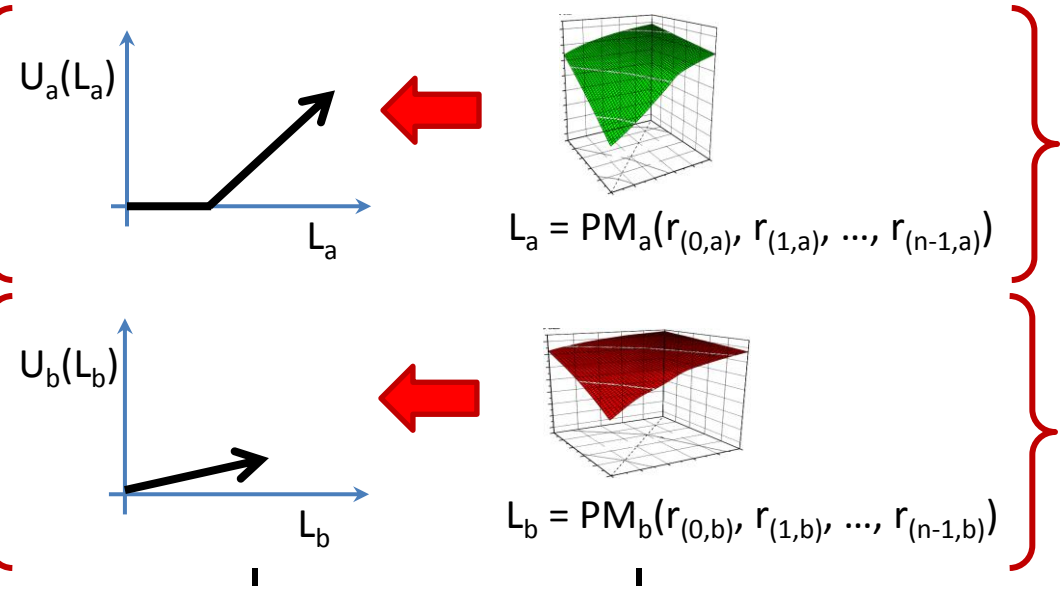
Continuously Minimize
(subject to restrictions on the total amount of resources)



Convex Surface

Urgency Function
Reflects the cell's importance

Performance Model (PM)
Output improves with resources



Performance Metric (L), e.g., latency

- System-wide, some overhead, and performance modeling is the key challenge

Implementation Status

- First prototype (ROS, poster by Barret, Kevin, et al.)
  - Supports partitioning, channels, among other features
  - Network driver and TCP/IP stack running in a partition
  - Currently two ports
    - 32-core Nehalem system (Intel)
    - 64-core RAMP emulation of a many-core processor (SPARC)
- A new version featuring the Mapping & Multiplexing Layer, Policy Service and Adaptation Policies is under development and to come soon
  - Sarah's poster & Juan's poster

Final Remarks

- Design principles in Tessellation OS
  - Space-time partitioning + Two-level scheduling
  - Key for performance isolation and composable performance
- Cell model
  - Cells + Channels
  - Cells for applications and OS services
- Adaptive resource-management framework
  - Implemented in the Policy Service
  - Will allow us experiment with different adaptation policies

Questions?

THANKS