Microsoft

EXTREME COMPUTING GROUP

Defining the future.

Cloud Programming

James Larus
Microsoft Research

July 13, 2010

New Programming Model, New Problems (and some old, unsolved ones)

Concurrency

Parallelism

Message passing

Distribution

High availability

Performance

Application partitioning

Defect detection

High-level abstractions

Concurrency

Inherently concurrent programming

- Asynchronous, message-driven model
- Multiple requests streams

Threads or events??

- Threads offer familiar sequential programming model
  - But, state can change when thread is preempted (synchronization)
  - Cost of thread and context switch limits concurrency
- Handlers fracture program control flow
  - Logic split across event handlers
  - Explicit manipulation of local state (no stack frames)

Higher-level (state machine, Actor, …) models?

Lack of consensus inhibits research, development, reuse, interoperability

- Parallel programming, redux

Parallelism

Computers are parallel
- Increased performance + power efficiency

Computers will be heterogeneous
- Multiple, non-isomorphic functional units

Data centers are vast message-passing clusters
- Availability and throughput

Parallel programming is long-standing sore point for computer science
- State of the art: threads and synchronization (assembly language)
- No consensus on shared memory semantics

New research on higher-level models is not panaceas
- Transactional memory
- Deterministic parallelism

Radical proposal: abolish shared memory
- Message passing is inherent in distributed systems, so why 2 models?
- Shared memory is difficult and error prone

Message Passing

Fundamental in distributed systems and better programming model

- Performance / correctness isolation
- Well-defined points of interaction
- Scalable

More difficult to use

- Little language support
  - Erlang integrates message with pattern matching
  - Sing# channel contracts
  - Sing# postbox semantics
- Message passing libraries
  - Fundamental mismatch: asynchronous strange in a synchronous world

Open problems

- Control structures for asynchronous messages
- Communications contracts
- Integration of messages in type system and memory model

Distribution

Distributed systems are rich source of difficult problems

- Replication
- Consistency
- Quorum

Well-studied field with good solutions

- Outsider's perspective: research has focused on fundamental problems and techniques used in real systems

Common abstractions

- Replication
- Relaxed consistency
- Persistence

How can these techniques be incorporated into programming model?

- Libraries
- Language integration
- New models

Availability


Democrat and Chronicle

BLACKOUT

Services must be highly available

- Blackberry/Google/… outage gets national media attention
- Affect millions of people simultaneously
- Service becomes part of national infrastructure

High availability is challenge

- Starts with design and engineering
- Hard to eliminate all "single points of failure"
- Murphy's law rules
- Antithetical to rapid software evolution

Programming models provide little support for systematic error handling

- Disproportionate software defects in error-handling code
  - Afterthought
  - Run in inconsistent state
  - Difficult to test
- Erlang has systematic philosophy of fail and notify (but stateless)
- Could lightweight transactions simplify rollback for stateful languages?

Performance

Performance is system-level concern
- Goes far beyond the code running on a machine
- Most performance tools focus on low-level details

Current approach is wasteful and uncertain
- Build, observe, tweak, overprovision, pray

Performance should be specified as part of behavior
- SLAs as well as pre-/post-conditions

Need scalability
- Grow by adding machines, not rewriting software

Architecture should be the starting point
- Model and simulate before building a system
- What is equivalent of Big-O notation for scalability?

Adaptivity
- Systems need to be introspective and capable of adapting behavior to load
- e.g., simplify home page when load spikes, defer low-priority tasks, provision more machines, …

Power


Fig. 600. Kolben & Co.

New challenge is power consumption

- Processor performance limited by power
  - Multicore only a temporary solution – "dark silicon"
- Data center locally and globally power constrained
  - Significant cost (CAPEX + OPEX)

Power-efficient software design and construction

- Equivalent of asymptotic analysis for power?
- How to measure power consumption?
- How to compare alternatives?
- Design patterns

Application Partitioning

Static partition of functionality between client and server

- Clients have different architectures and capabilities
- Adapt to changing constraints (e.g., battery)
- Move computation to data, particularly when communications constrained
- Code mobility
  - Exists in data center (VMs), why not across data center boundary?

Currently, client and server are two fundamentally different applications

- Evolution around interfaces
- Volta (Microsoft)
  - Single program model, compiled for server and client

Defect Detection

Considerable progress in past decade on defect detection tools

- Tools focused on local properties (e.g., buffer overruns, test coverage, races, etc.)
- Little work on system-wide properties

Modular checking

- Whole program analysis expensive and difficult
- Not practical for services
- Assertions and annotations at module boundaries
- Can check global properties locally
- e.g., Rajamani & Rehof's Conformance Checking

New domain of defects

- Message passing
- Code robustness
- Potential performance bottlenecks

High-Level Abstractions

Map-reduce and dataflow abstractions simplify large-scale data analysis in data centers

- Convenient way to express problems
- Hide complex details (distribution, failure, restart)
- Allow optimization (speculation)
- Not appropriate for services

Need abstractions for wider range of problems

- Interactive applications

Orleans

Goals

- Simple, widely accessible programming model
- Encourage use of scalable, resilient software architectures
- Raise level of abstraction (CLR – Windows ≈ Orleans – Azure)

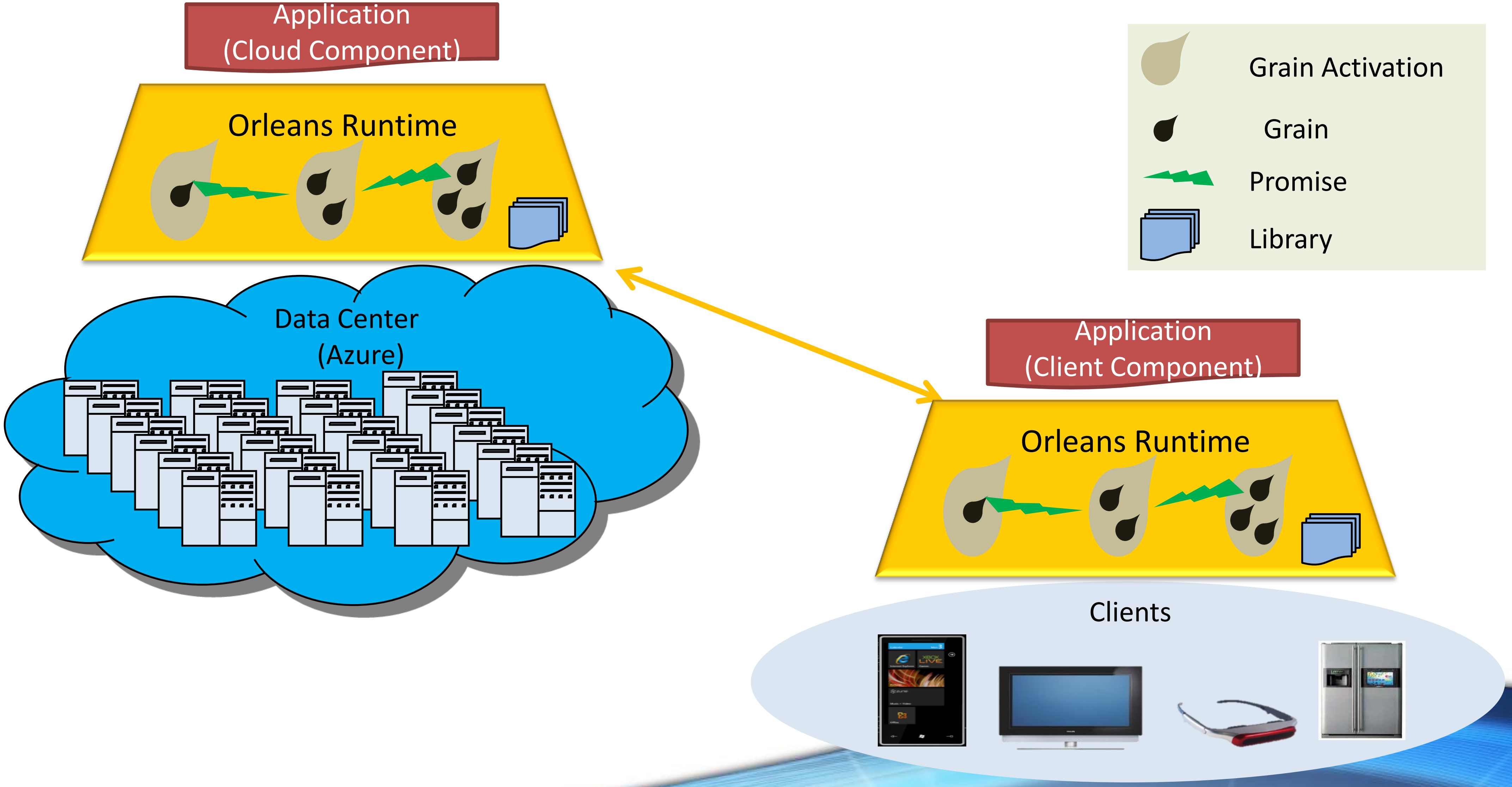
Grains are unit of computation and data storage (Actors)

- Can migrate between data centers
- Replication, consistency, persistence handled by runtime system

One programming model for client and server

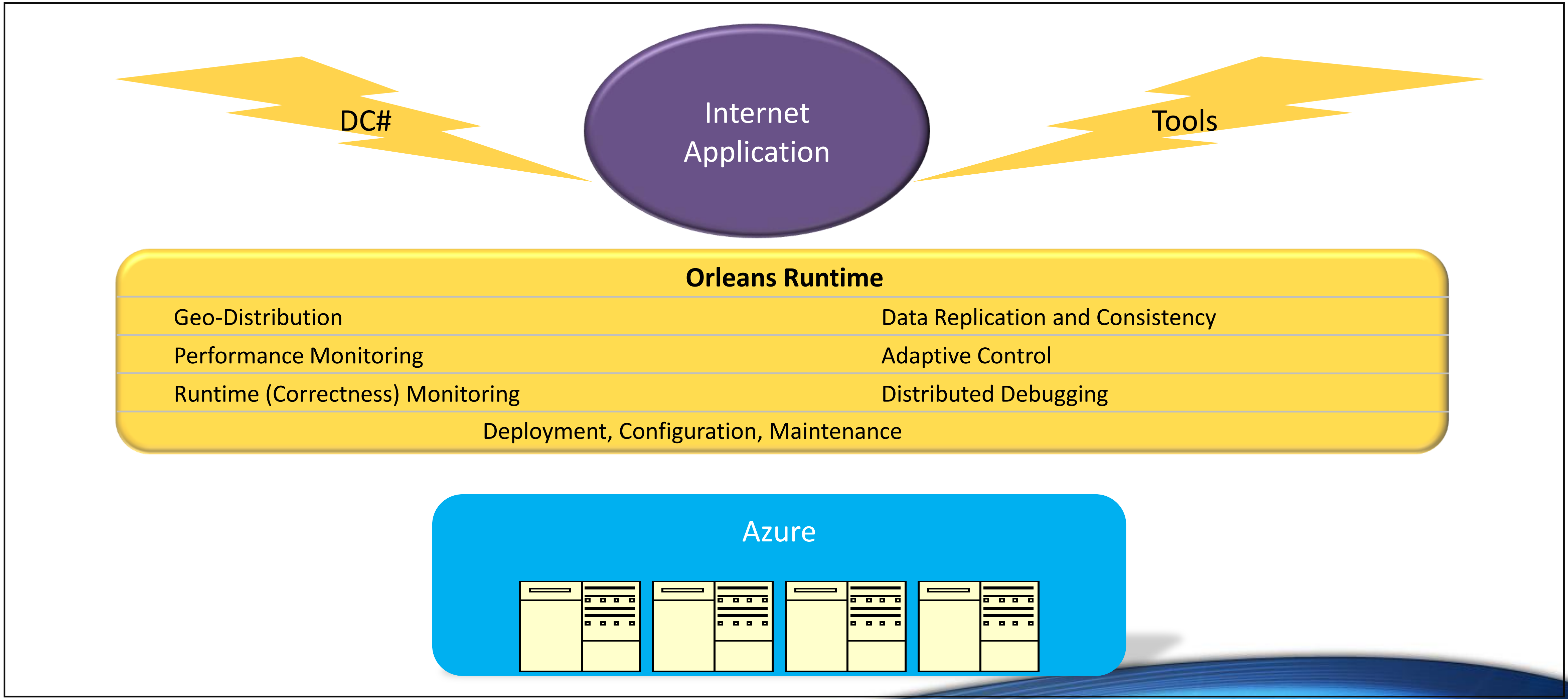
- Simplify development, debugging, performance tuning, etc.
- Single-source distributed programs (eg Volta)
- Enable code mobility

Orleans



Application
(Cloud Component)
Orleans Runtime
Data Center
(Azure)
Grain Activation
Grain
Promise
Library
Application
(Client Component)
Orleans Runtime
Clients

Orleans Architecture



Microsoft®