Many Core OS

Haoliang & Quanzeng

Background

- Many core processor
  - Intel Many Integrated Core (MIC) Architecture
  - NVidia Kepler Compute Architecture
- Why many core?
  - Performance

Challenges

- Scalability
- Elasticity
- Availability
- Diversity

Scalability

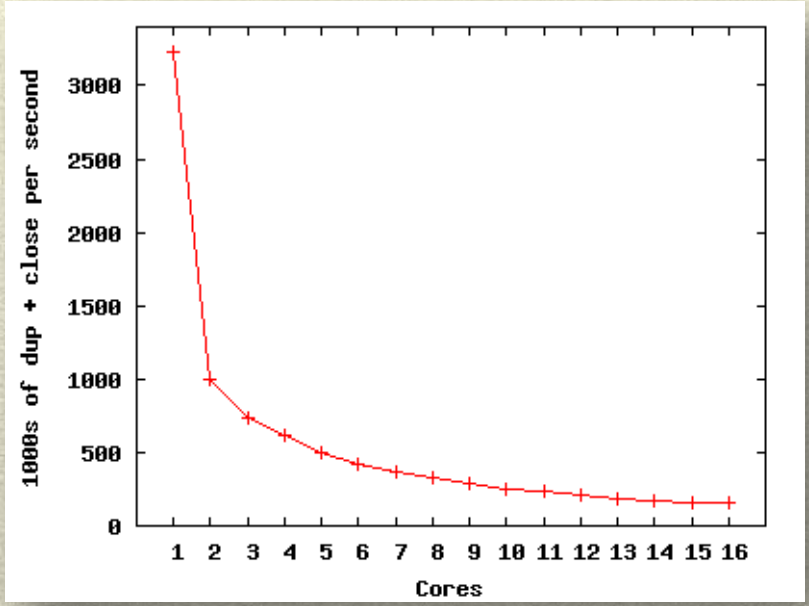


Figure 1: Throughput of the file descriptor dup and close on Linux.[1]

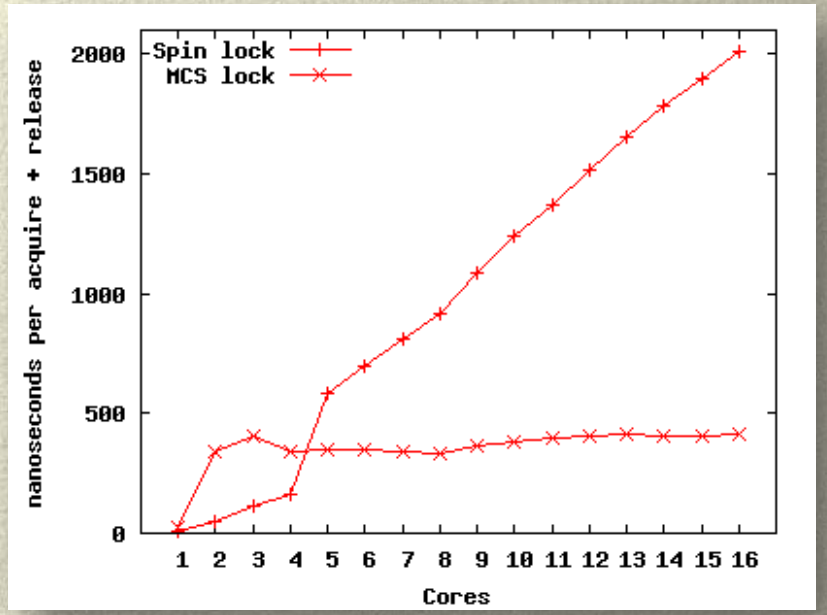


Figure 2: Time required to acquire and release a lock. [1]

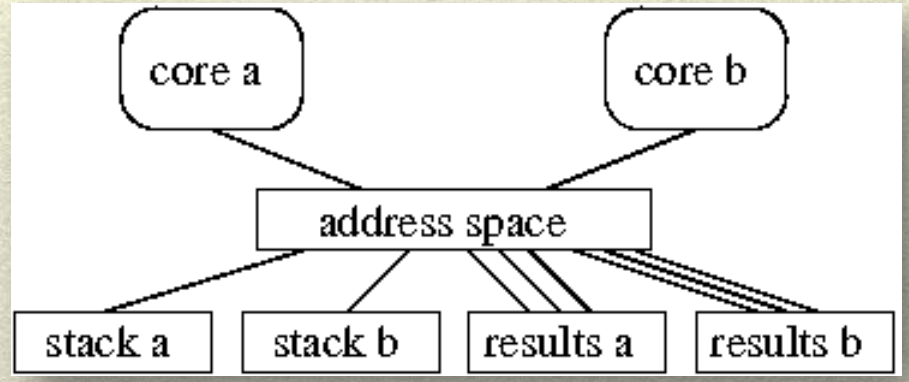
- Poor scalability of system services becomes the bottleneck of the performance.

Corey

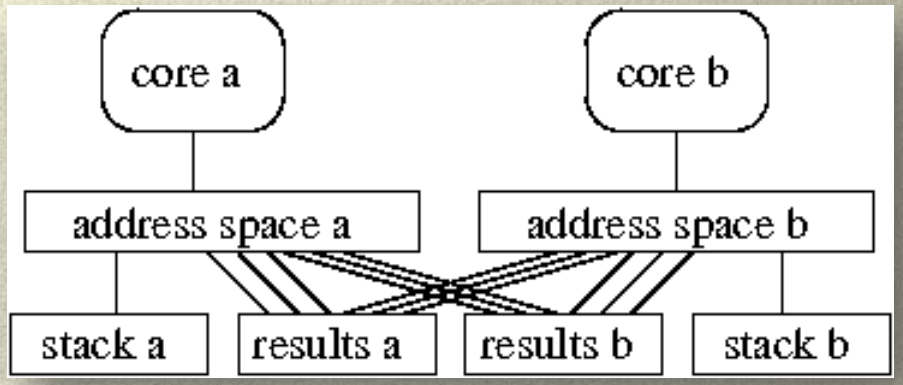
- Ideas

  Avoid contentions over unnecessary sharing.
- Solutions
  - Application controlled sharing
  - Dedicated kernel core
- Target applications
  - Web server
  - MapReduce

Memory address space



(a) A single address space. [1]

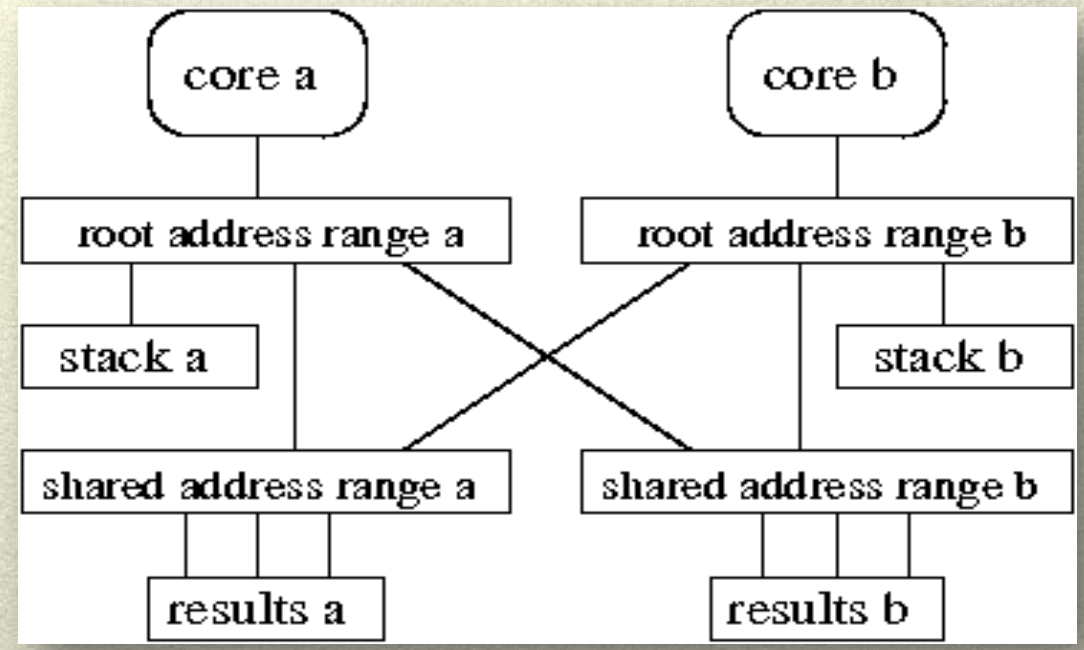


(b) Separate address spaces. [1]

- Advantages:
  - No soft page fault
- Disadvantages:
  - Contention
  - TLB invalidation

- Advantages:
  - No contention
  - No TLB invalidation
- Disadvantages:
  - Soft page fault

Memory address space



(c) Two address spaces with shared result mappings. [1]

- No contention in private address space
- No TLB invalidations
- No soft page fault

Kernel data structure

- Cache coherence cost

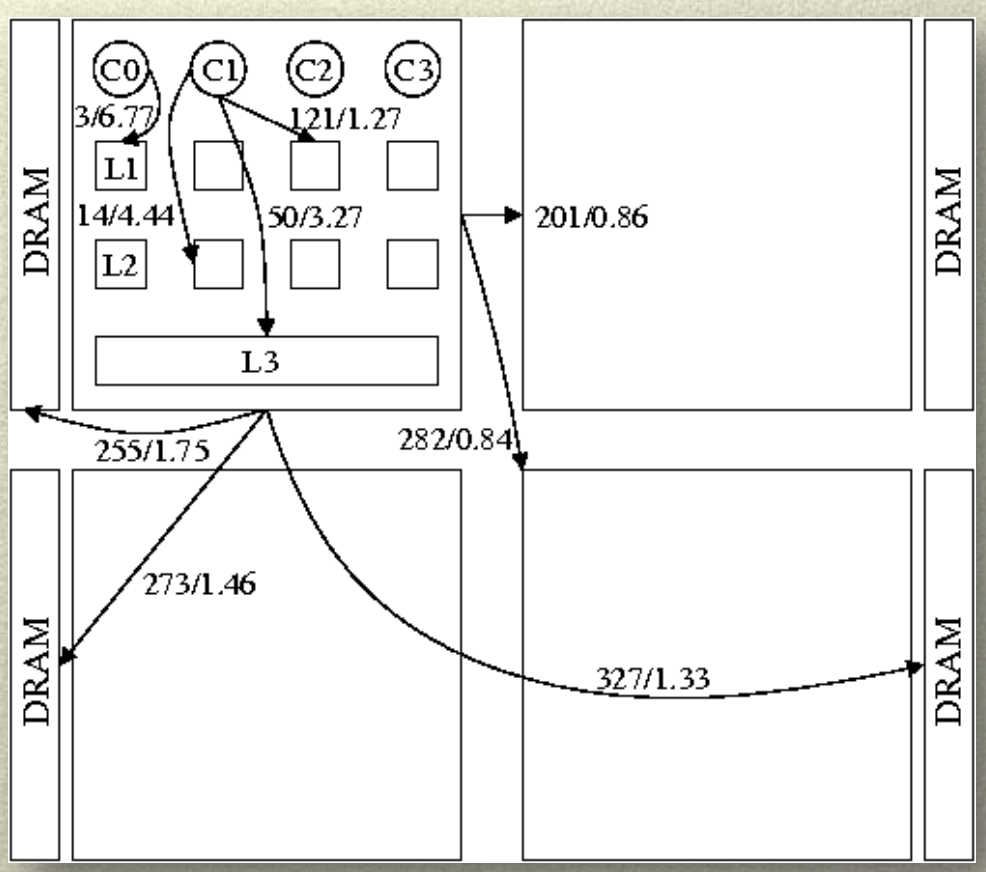


Figure 4: The AMD 16-core system topology. [1]

Application control sharing

- To eliminate the unnecessary sharing, the memory address space and kernel data structures will be divide into two parts – private and shared part.
- Applications decide what to share and who they share with.
- Increase the performance and also increase the complexity of programming.

Kernel cores

- Not all kernel structure can be privatized. Kernel data structures used by certain system call need to be shared.
- To solve this, dedicated kernel core is introduced.
- Dedicated to certain system service like network service.
- Communication between cores is accomplished by shared-memory IPC.

Kernel cores

- Two kernel core configuration in Corey

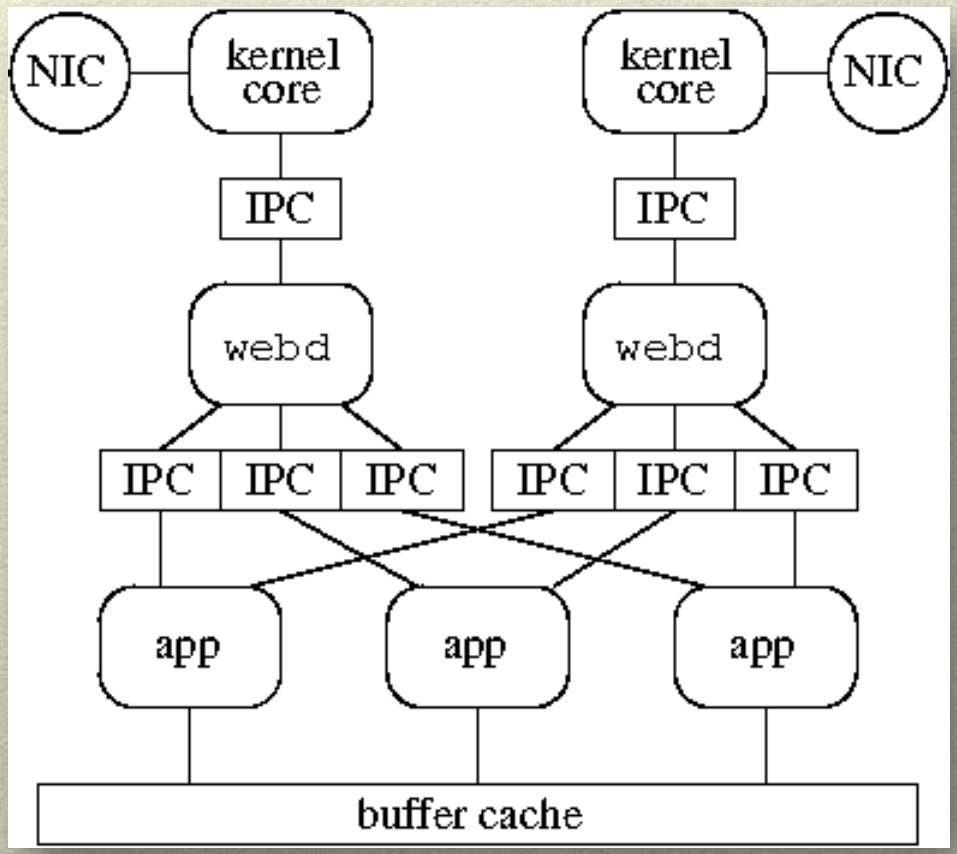


Figure 4: A Corey Web server configuration with two kernel cores, two webd cores and three application cores. Rectangles represent segments, rounded rectangles represents pcores, and circles represent devices.. [1]

Kernel cores

- Certain system services will be greatly accelerated. However, we lose some cores.
- Need to take locality into consideration.
- Also decided by the application which requires programmer to balance carefully.

fOS

Challenges of Cloud Systems

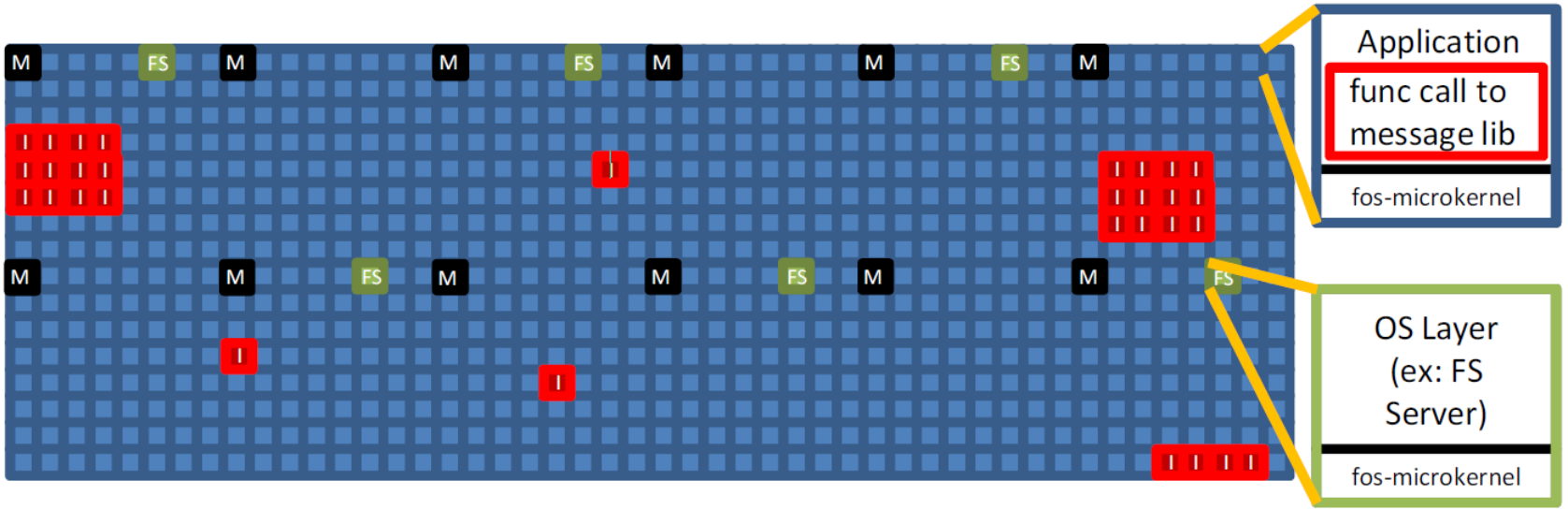
- Variability of Demand
  - Number of live cores
  - Peak load(unknown)
- Faults
  - More common(more cores, more smaller transisters)
  - Software faults(massive systems)
- Programming Challenges
  - Communication
  - Debug

Design Principles

- Space multiplexing replaces time multiplexing
  - Scheduling becomes a layout problem
  - OS runs on distinct cores
  - OS does not interfere with application cache
- OS is factored into function specific services
  - Spatially distributed servers
  - Message passing
- Faults detection and handling

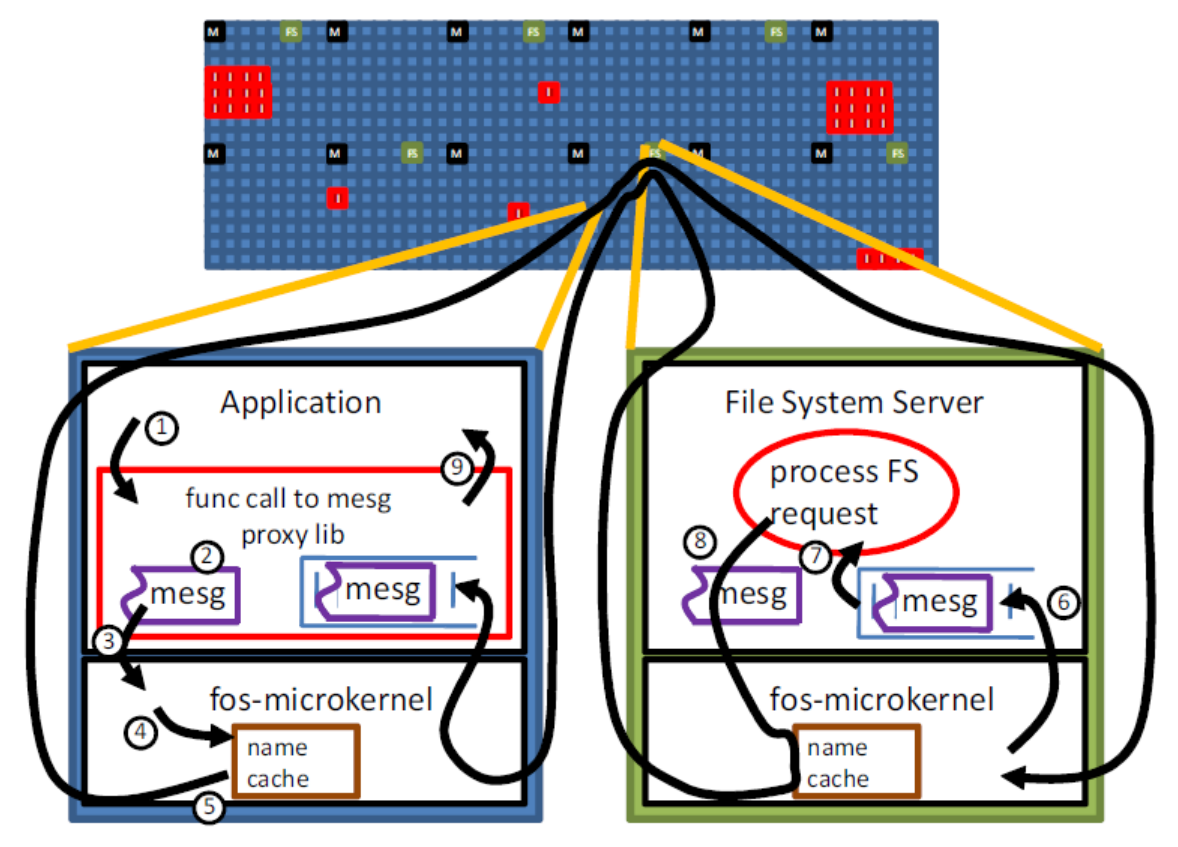
Structure of fos

A microkernel; OS layer; Application



Microkernel Messaging

- Mailbox
- Name server



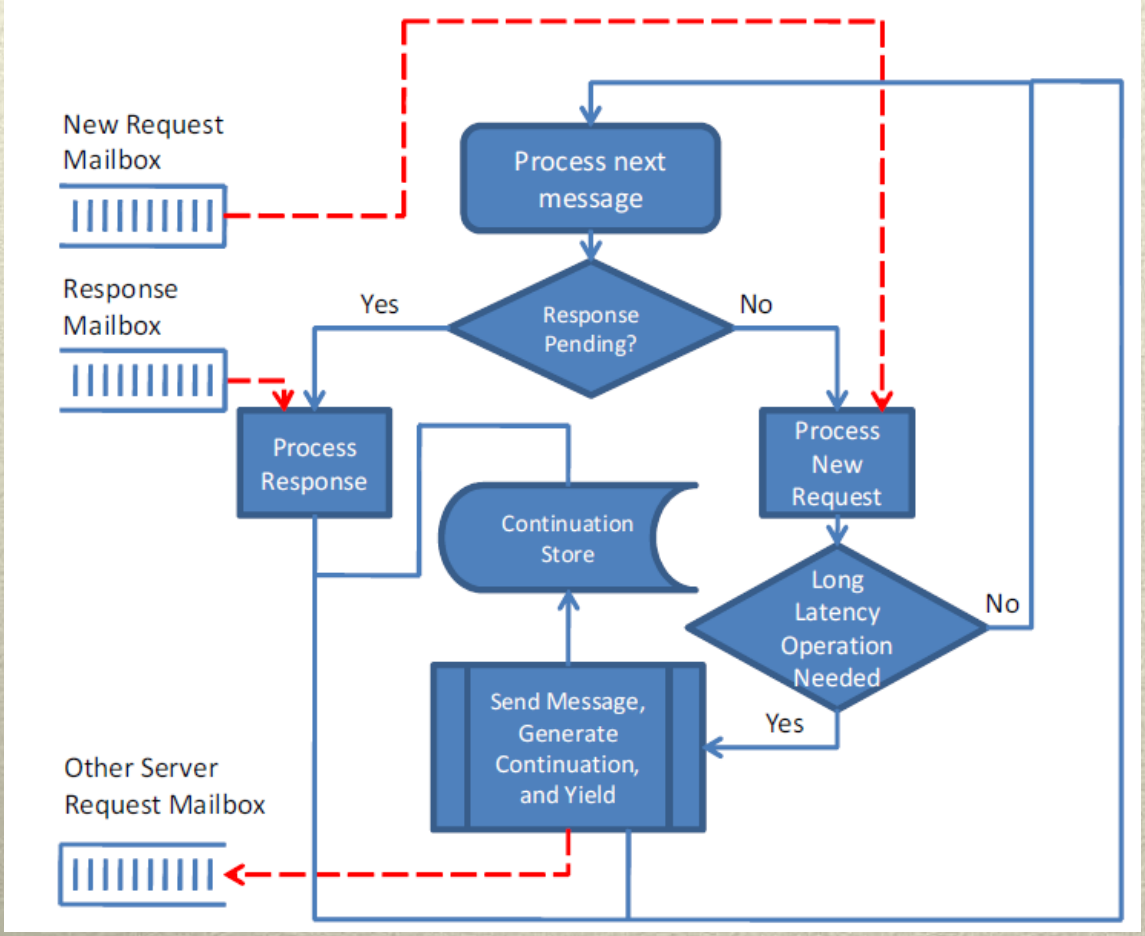
Example of an access to file system

1. System call *read*
2. Msg proxy lib
3. Call kernel to send msg
4. Microkernel looks up physical destination
5. Transport
6. Deposit msg to FS server's mailbox
7. Process request

8-9 Return data

Structure of a server

Transaction-oriented **stateless** process



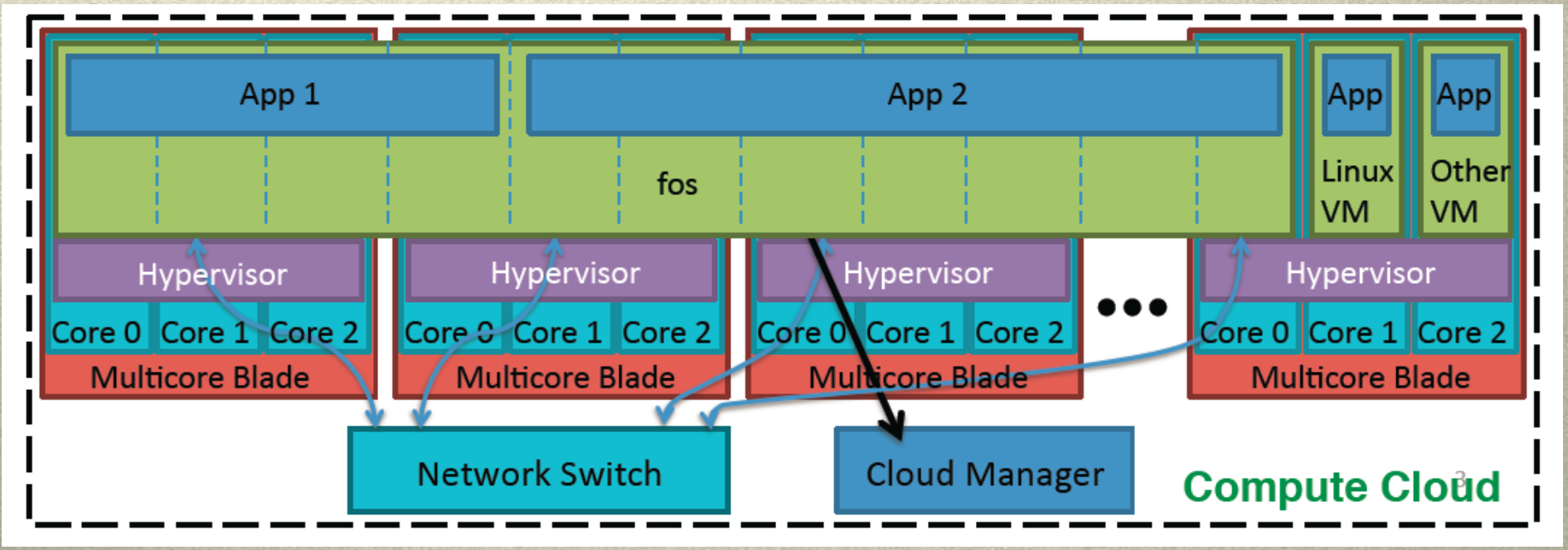
Transaction:

1. A request sent to a server
2. Server process it
3. Server Reply

Priorities response mailbox over request mailbox

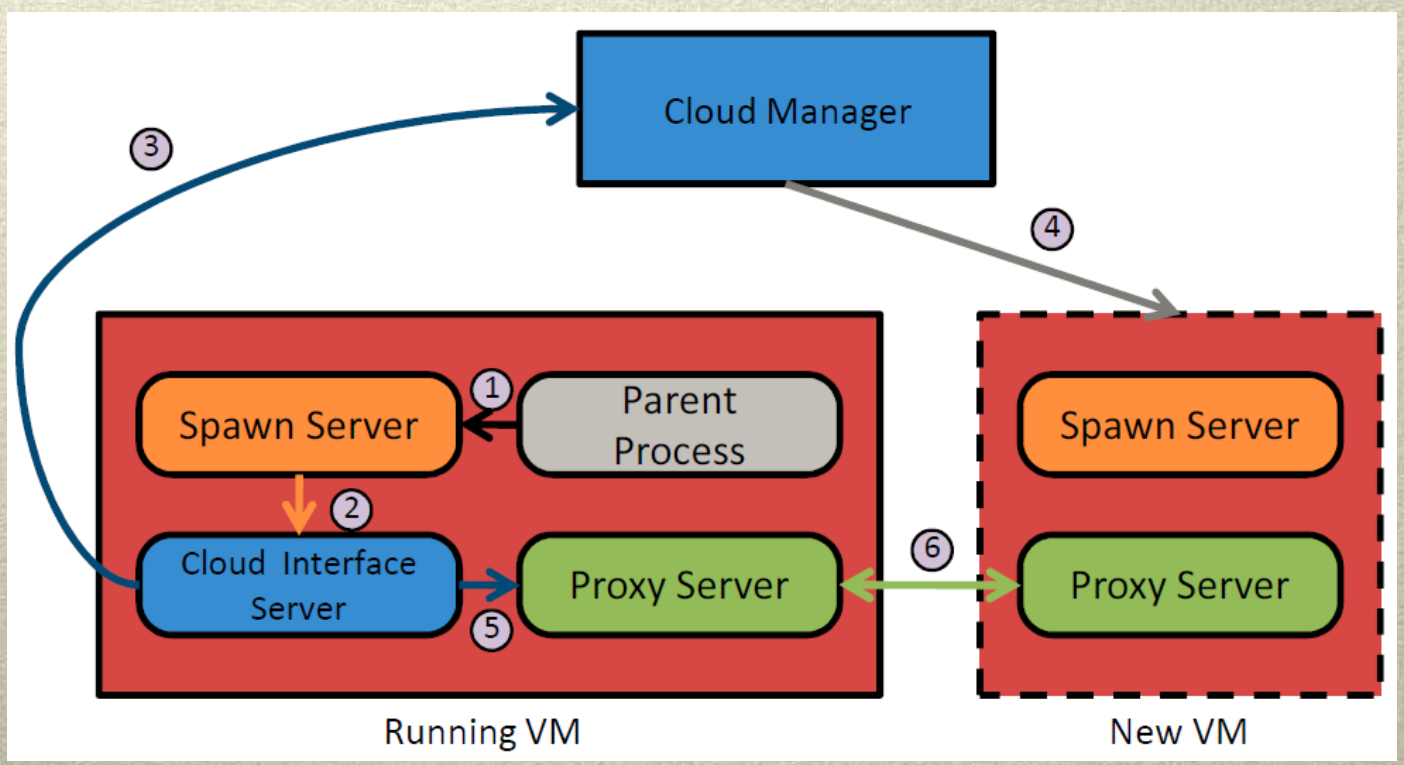
Structure of fos on cloud

Build on top of VMs



Spawning Servers

- SpawnProcess()
  - Where to deploy?
  - Need new VM?



Elastic Fleet

- Watchdog process
  - Watch the load
- Choose decision and policy
  - Resources available in a global scale
  - Load and location of existing servers
  - Monetary concerns
  - …

Design Evaluation

- Inherently message passing
  - No shared memory locks between cores
- Non-preemptive servers
- No dependent on hardware shared memory

- Challenges
  - Introduces core-to-core communication latency

Open Questions

- Accessing performance of traditionally shared structures
  - E.g. fs buffer caches
  - Lock servers(notional lock or transaction server)
- Multi-sequence without local locks on a server?

Summary

- Application controlled sharing
- Dedicated kernel core
- Space multiplexing
- Less shared data
  - New way of communication

Thank you

Reference

1) Corey: An Operating System for Many Cores
2) Tessellation Project
3) A Unified Operating System for Clouds and Manycore: fos
4) The Barrel Fish Project
5) http://groups.csail.mit.edu/carbon/?page_id=39
6) An Operating System for Multicore and Clouds: Mechanisms and Implementation, by David Wentzlaff, Charles Gruenwald III, Nathan Beckmann, Kevin Modzelewski, Adam Belay, Lamia Youseff, Jason Miller, and Anant Agarwal ACM Symposium on Cloud Computing (SOCC), June 2010.
7) Factored Operating Systems (fos): The Case for a Scalable Operating System for Multicores, by David Wentzlaff and Anant Agarwal. ACM SIGOPS Operating System Review (OSR), April 2009. (pdf)