Extreme High Performance Computing or Why Microkernels Suck

Ottawa Linux Symposium

2007-06-30

Christoph Lameter, Ph.D.
christoph@lameter.com
Technical Lead Linux Kernel Software
Silicon Graphics, Inc.

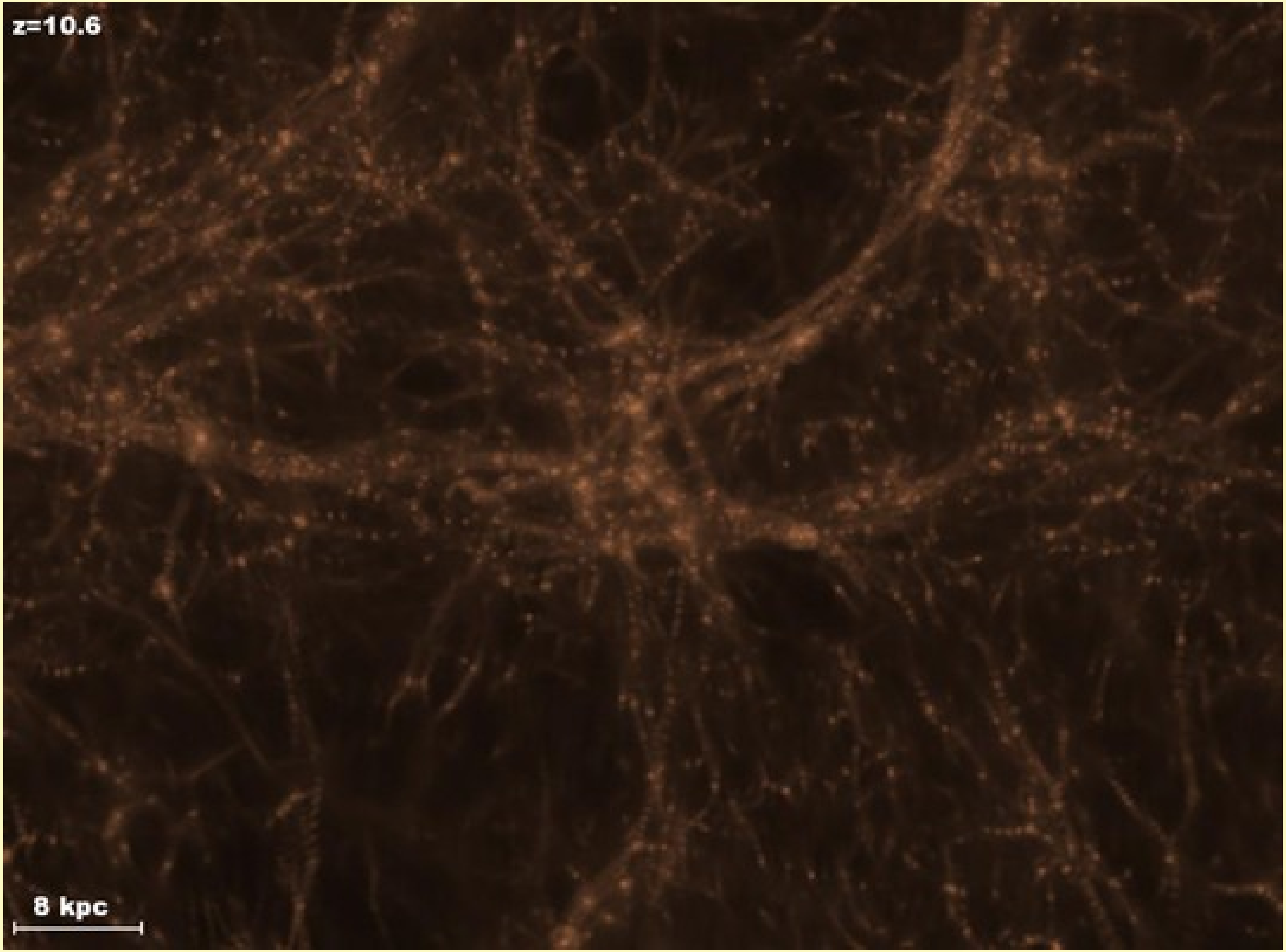
Introduction

- Short intro to High Performance Computing
- How high does Linux currently scale
- Conceptual comparison: microkernel and monolithic OS (Linux)
- Fundamental scaling problems of a microkernel based architecture
- Monolithic kernel are also modular
- Why does Linux scale so well and adapt to ever larger and more complex machines
- Current issues
- Conclusion: Microkernel is an idea taken to unhealthy extremes.

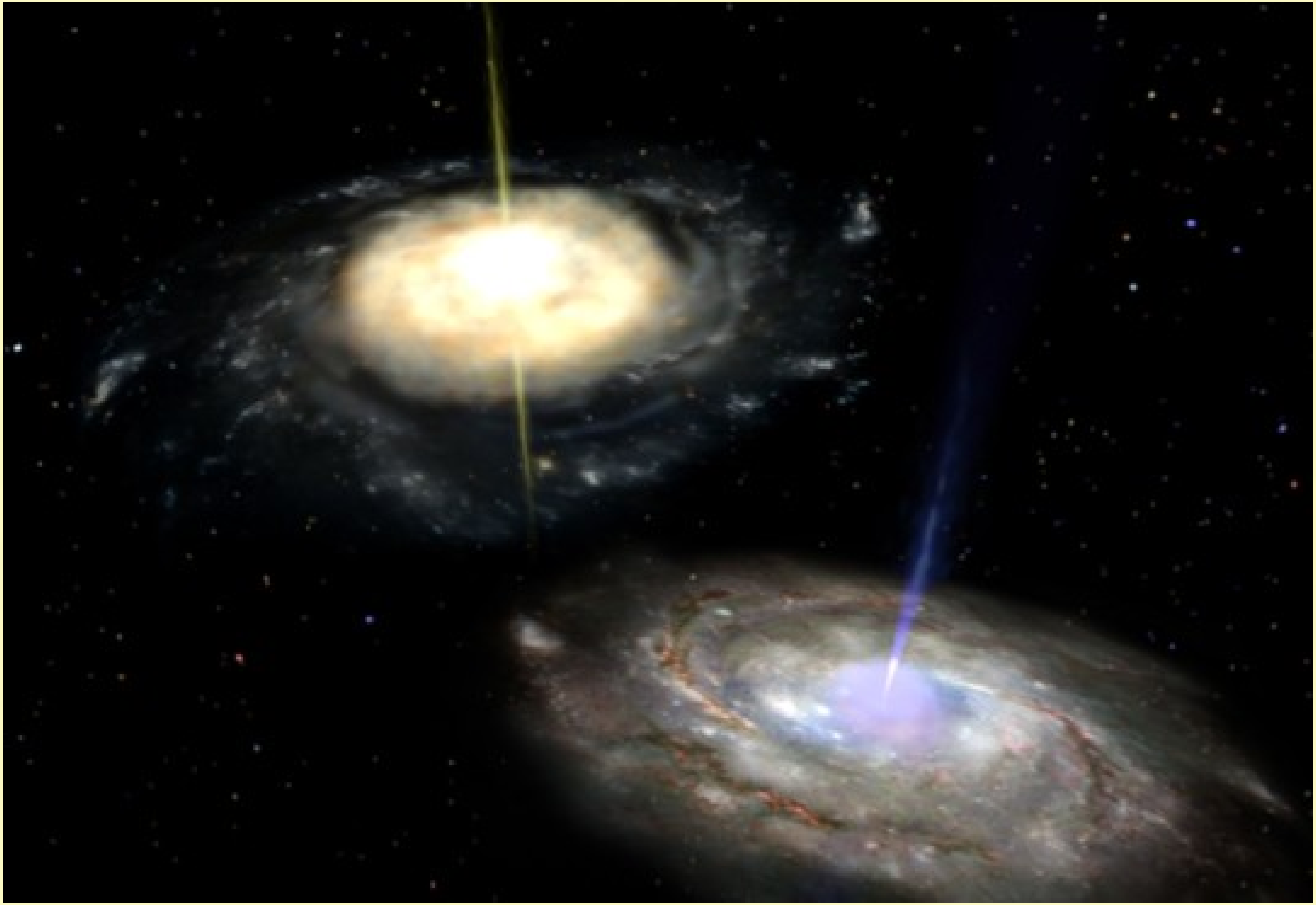
Applications of High Performance Computing

- Solve complex computationally expensive problems
- Scientific Research
  - Physics (quantum mechanics, nuclear phenomena)
  - Cosmology
  - Space
  - Biology (gene analysis, virus, bacteria etc)
- Simulations
  - Weather (Hurricanes)
  - Study of molecules and new substances
- Complex data analysis
- 3D design
  - Interactive modeling (f.e. car design, aircraft design)
  - Structural analysis.

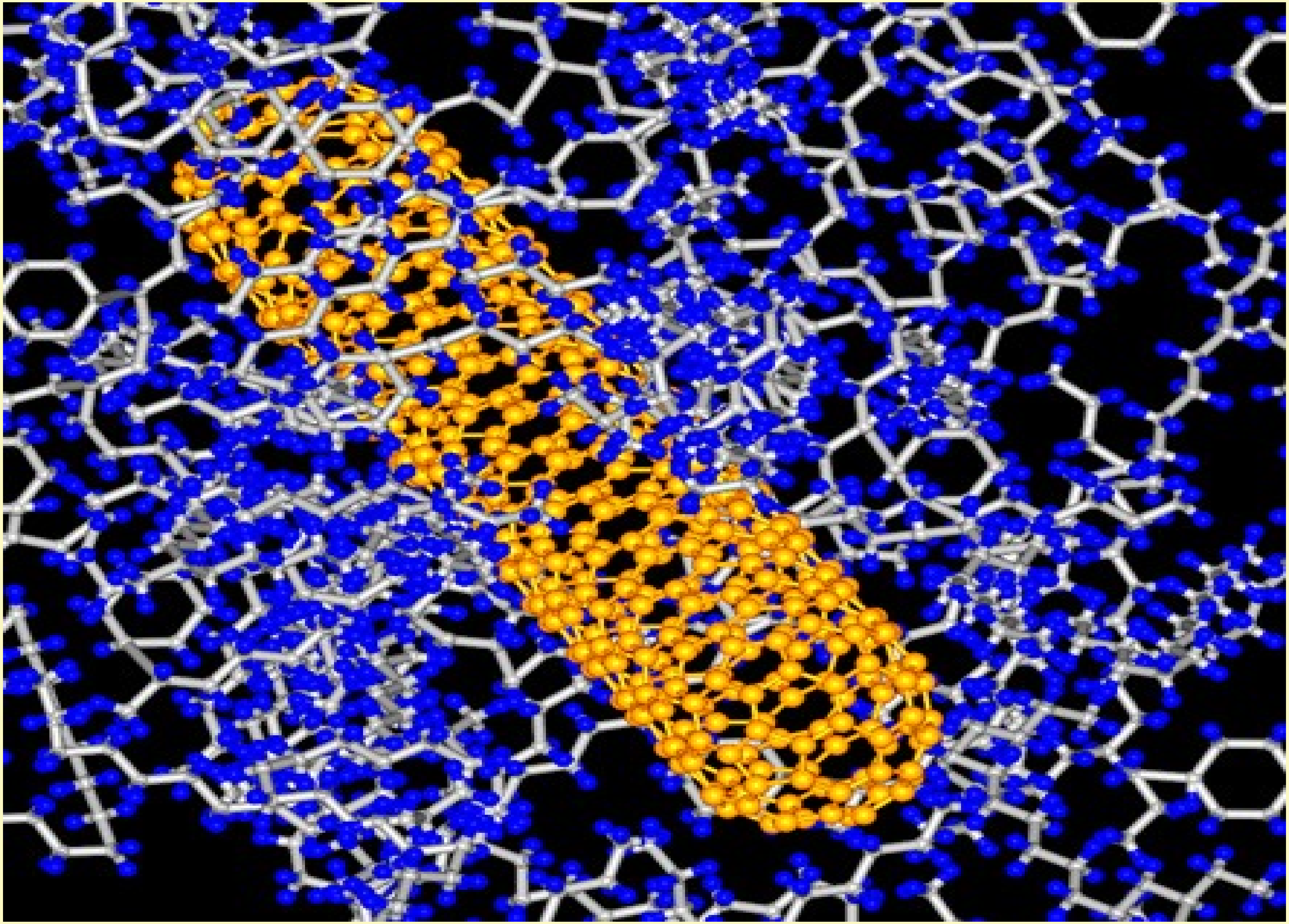
Dark Matter Halo Simulation for the Milky Way



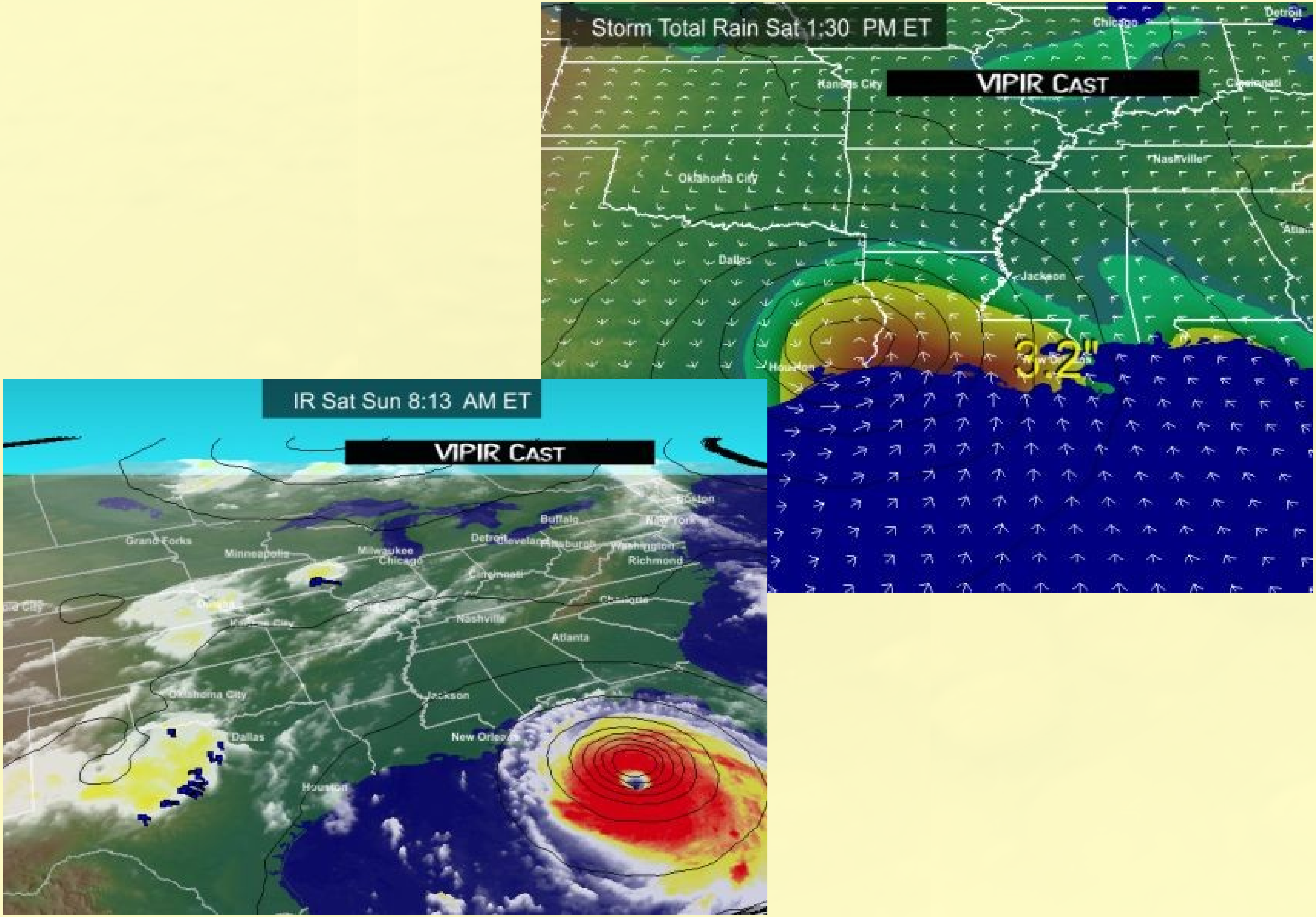
Black Hole Simulation



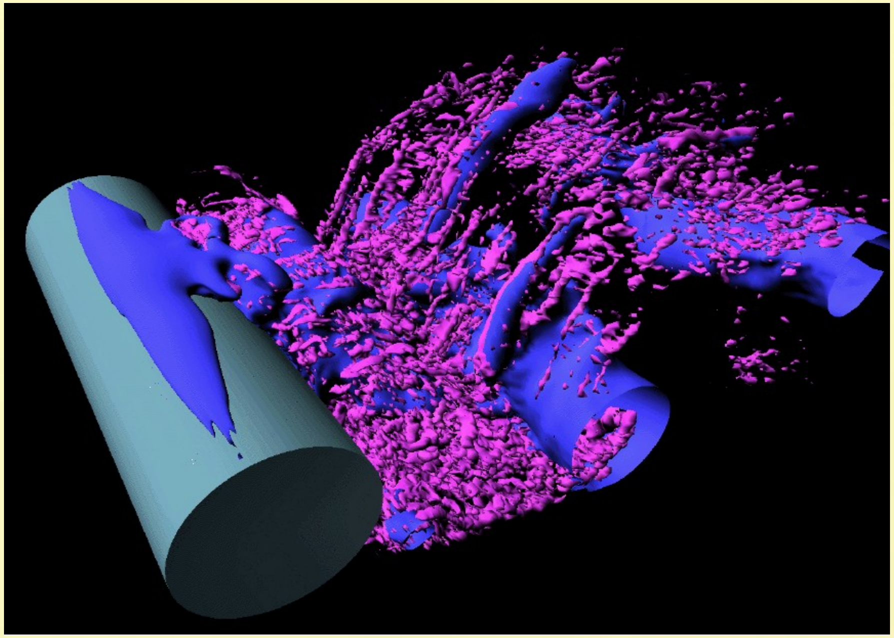
Carbon Nanotube-polymer composite material

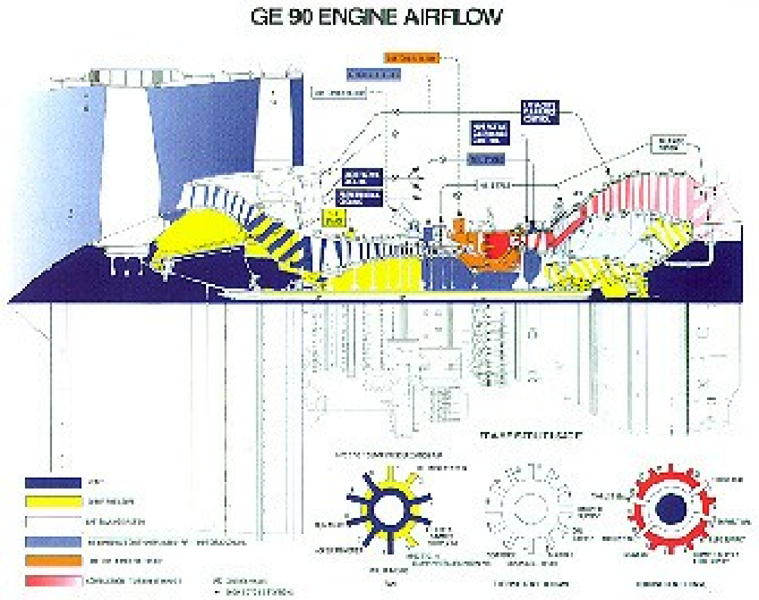


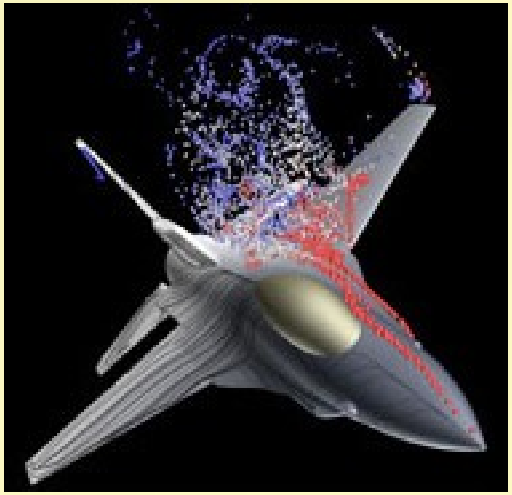
Forecast of Hurricane Katrina



Airflow Simulations







High Performance Computer Architectures

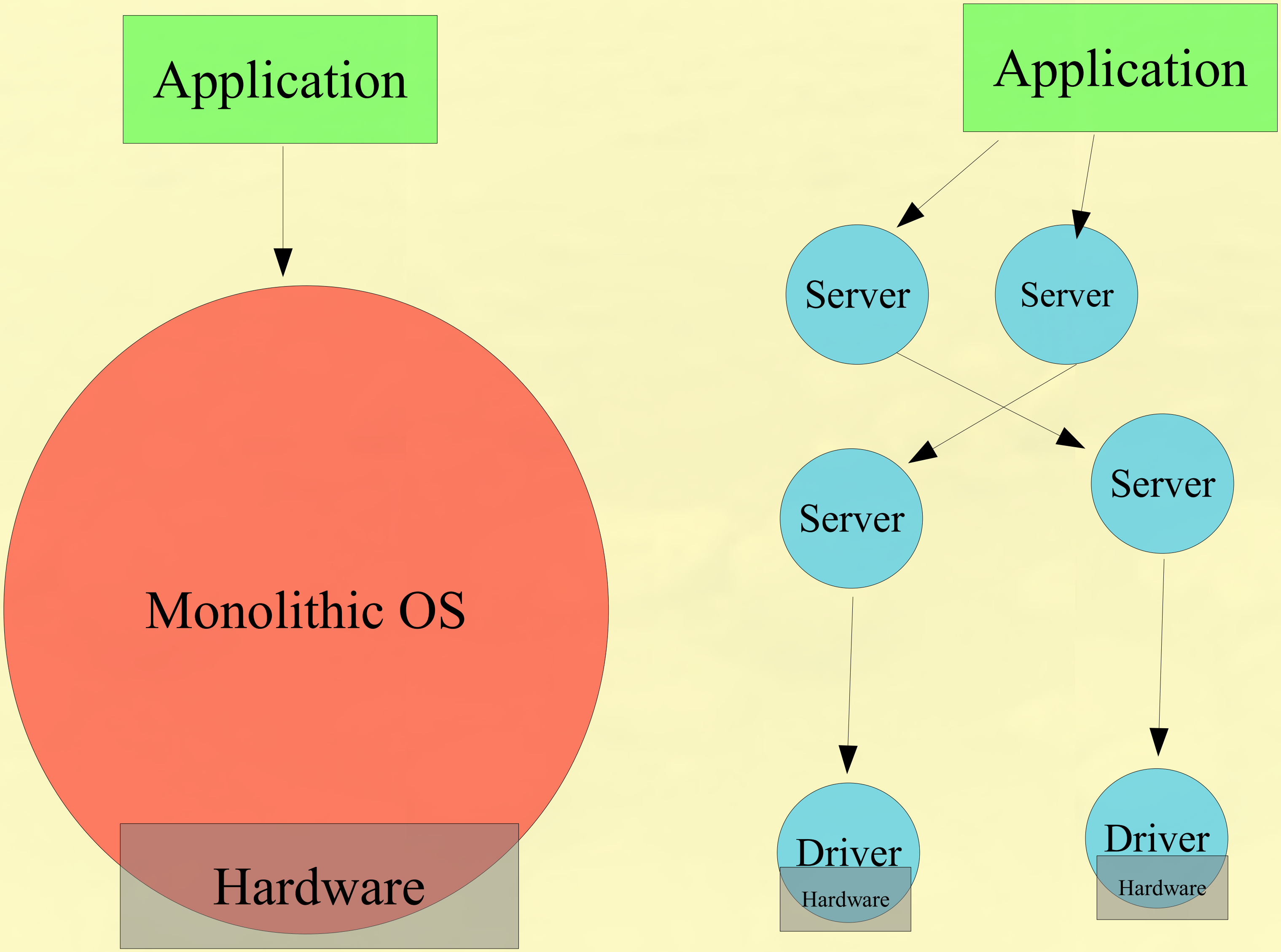
- Supercomputer
  - Single memory space
  - NUMA architecture. Memory nodes / Distant memory.
  - Challenge to scale the Operating System
- Cluster
  - Multiple memory spaces
  - Networked commodity servers
  - Network communication critical for performance
  - Challenge to redesign applications for a cluster
- Mainframe
  - Singe uniform memory space with multiple processors
  - Scalable I/O subsystem
  - Mainly targeted to I/O transactions
  - Reliable and maintainable (24 by 7 availability)

NASA Columbia Supercomputer with 10240 processors

Current Maximum Scaling of a single Linux Kernel

- This is no cluster
  - Single address space
  - Processes communicate using shared memory
- Currently deployed configurations
  - Single kernel boots 1024 processors
  - 8 Terabyte of main memory
  - 10GB/sec I/O throughput
- Known working configurations
  - 4096 processors
  - 256TB memory
- Next generation platform
  - 16384 processors
  - 4-8 Petabyte (2^50 bytes) Memory

Monolithic kernel vs micro kernel



Microkernels vs. Monolithic

- Microkernel claims
  - Essential to deal with scalability issues.
  - Allow a better designed system
  - Essential to deal with complexity of large Operating systems
  - Make the system work reliable
- However
  - Large scale microkernel systems do not exist
  - Research systems exist up to 24p (an unconfirmed rumors about 64p).
- IPC overhead vs. Monolithic kernels function calls
  - Need for context switches within the kernel
  - Transfer issues of messages.
  - Significant effort is spend on optimizing around these.

Isolation vs. Integration

- Microkernel isolates kernel components
  - More secure from failure
  - Defined API to between components of a kernel
- Monolithic OS
  - Large potentially complex code
  - Universal access to data
  - API implicitly established by function call convention
- Difficulty of keeping application state in Microkernels
- Performance issues by not having direct access to relevant data from other subsystems.
- Monolithic OS like Linux also have isolation methods
  - Source code modularization
  - Binary modules

APIs

- Monolithic kernel has flexible APIs if no binary APIs are supported like in Linux
- Microkernel must attempt to standardize on APIs to ensure that operating system components can be replaced.
- Thus a monolithic kernel can evolve faster than microkernel.

Competing technologies within a Monolithic Kernel

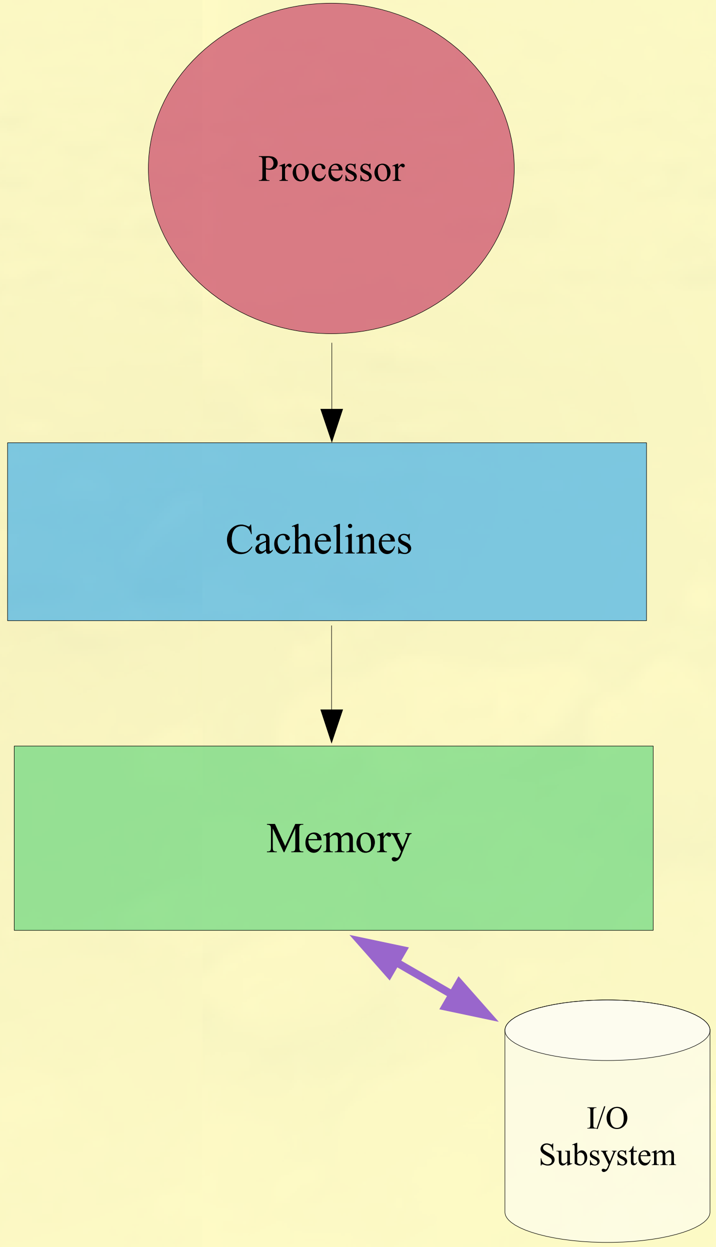
- Variety of locks that can be used to architect synchronization methods
  - Atomic operations
  - Reference counts
  - Read Copy Update
  - Spinlocks
  - Semaphores
- New Approaches to locking are frequently introduces to solve particular hard issues.

Scaling up Linux

- Per cpu areas
- Per node structures
- Memory allocators aware of distance to memory
- Lock splitting
- Cache line optimization
- Memory allocation control from user space
- Sharing is a problem
- Local Memory is the best
- Larger distances mean larger systems are possible
- The bigger the system the smaller the portion of local memory.

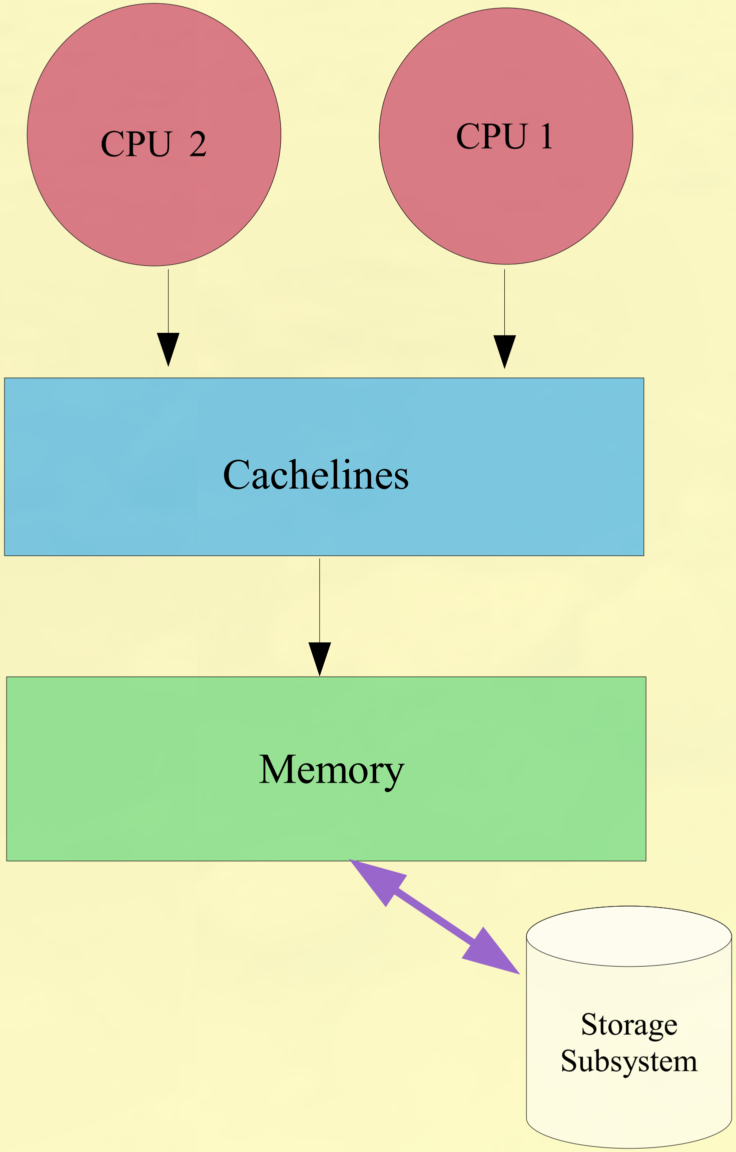
Single Processor System

- All computation on a single processor
- Only parallelism that needs to be managed is with the I/O subsystem
- Memory is slow compared to the processor.
- Speed of the system depends on the effectiveness of the cache
- Memory accesses have the same performance.



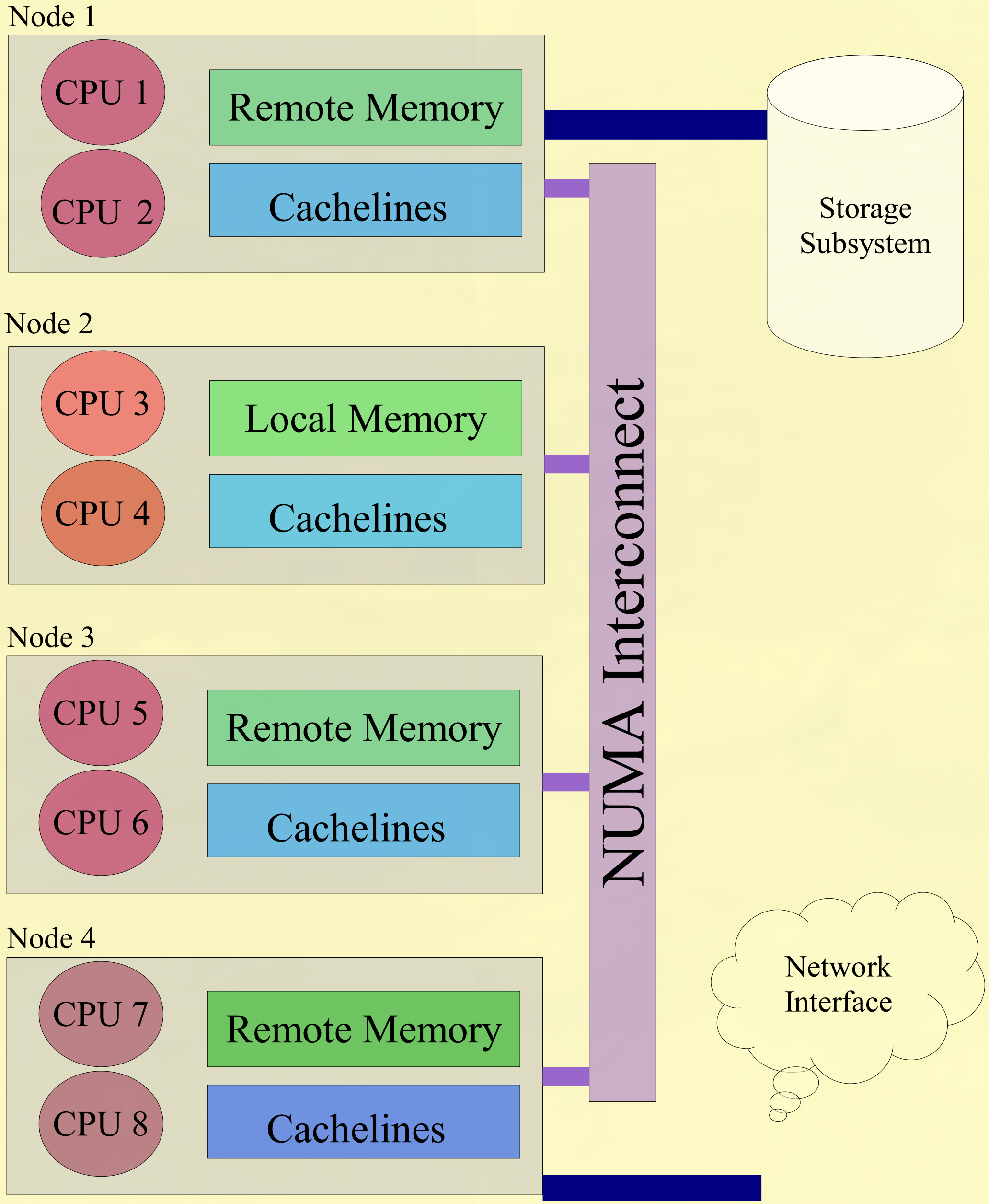
Symmetric Multi Processing (SMP)

- Multiple processors
- New need for synchronization between processors
- Cache control issues
- Performance enhancement through multiple processors working independently
- Cacheline contention
- Data layout challenges: shared vs. processor local
- All memory access have the same performance



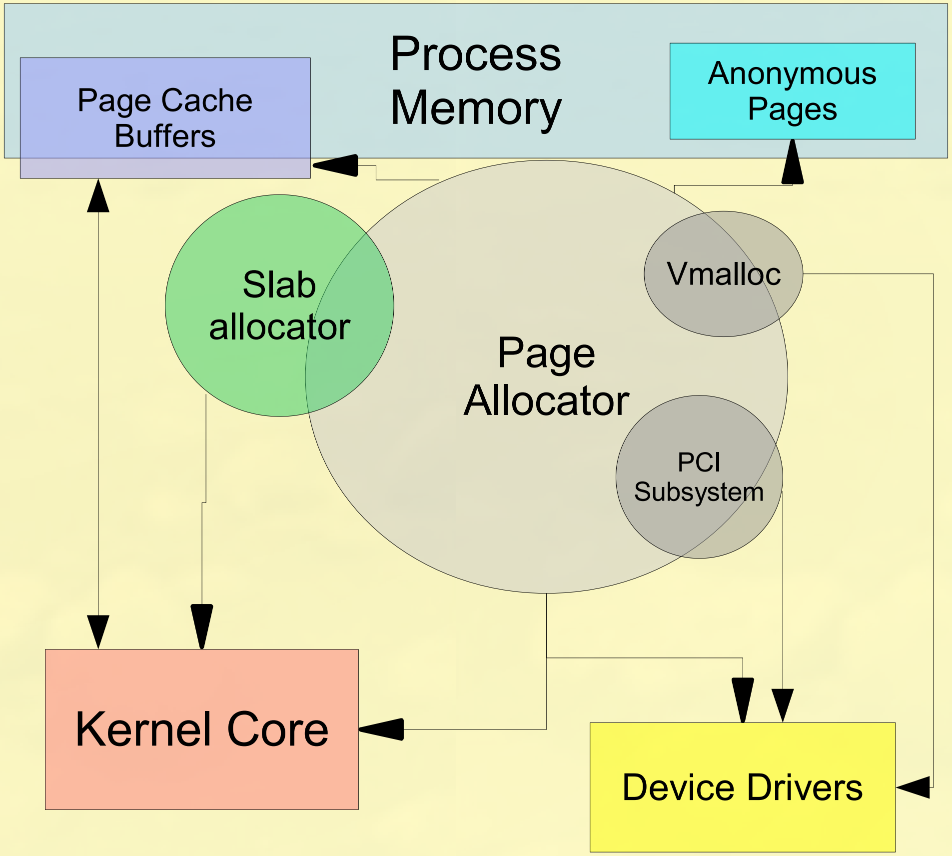
Non Uniform Memory Architecture (NUMA)

- Multiple SMP like systems called “nodes”
- Memory at various distances (NUMA)
- Interconnect
- MESI type cache coherency protocols
- SLIT tables
- Memory Placement
- Node Local from node 2 processor 3
- Device Local



Allocators for a Uniform Memory Architecture

- Page Chunks
- Page allocator
- Anonymous memory
- File backed memory
- Swapping
- Slab allocator
- Device DMA allocator
- Page Cache
- read() / write()
- Mmapped I/O.



NUMA Allocators

- Memory management per node
- Memory state and possibilities of allocation
- Traversal of the zonelist (or nodelist)
- Process location vs. memory allocation
- Scheduler interactions
- Predicting memory use?
- Memory load balancing
- Support to shift the memory load