Multiprocessors

CS 6410

Ashik Ratnani, Cornell University

Need for Faster Processors

- End of Moore's Law?
- Unable to improve the processing speed of Uniprocessors.

Flynn's Classification of Multiple processor machines

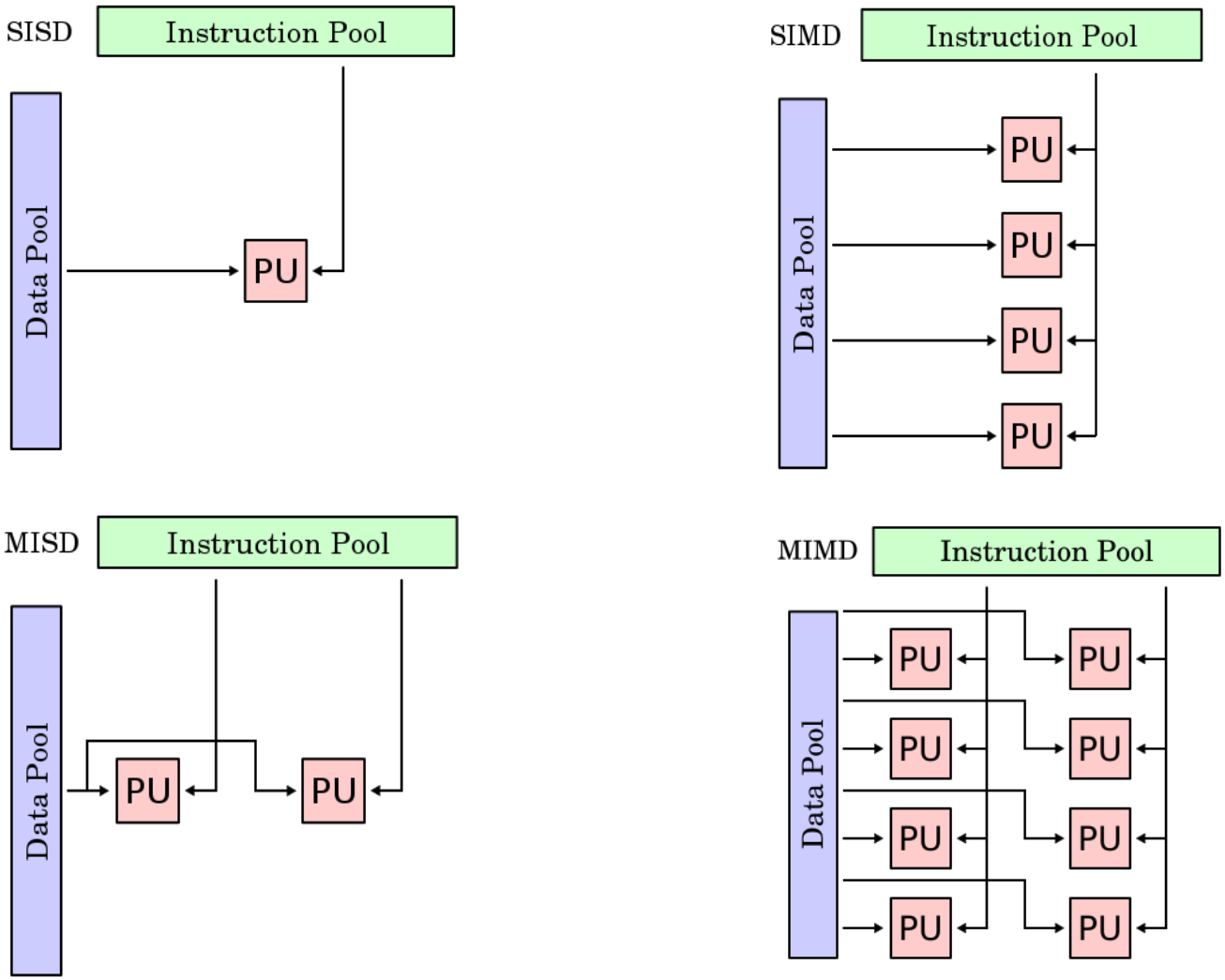
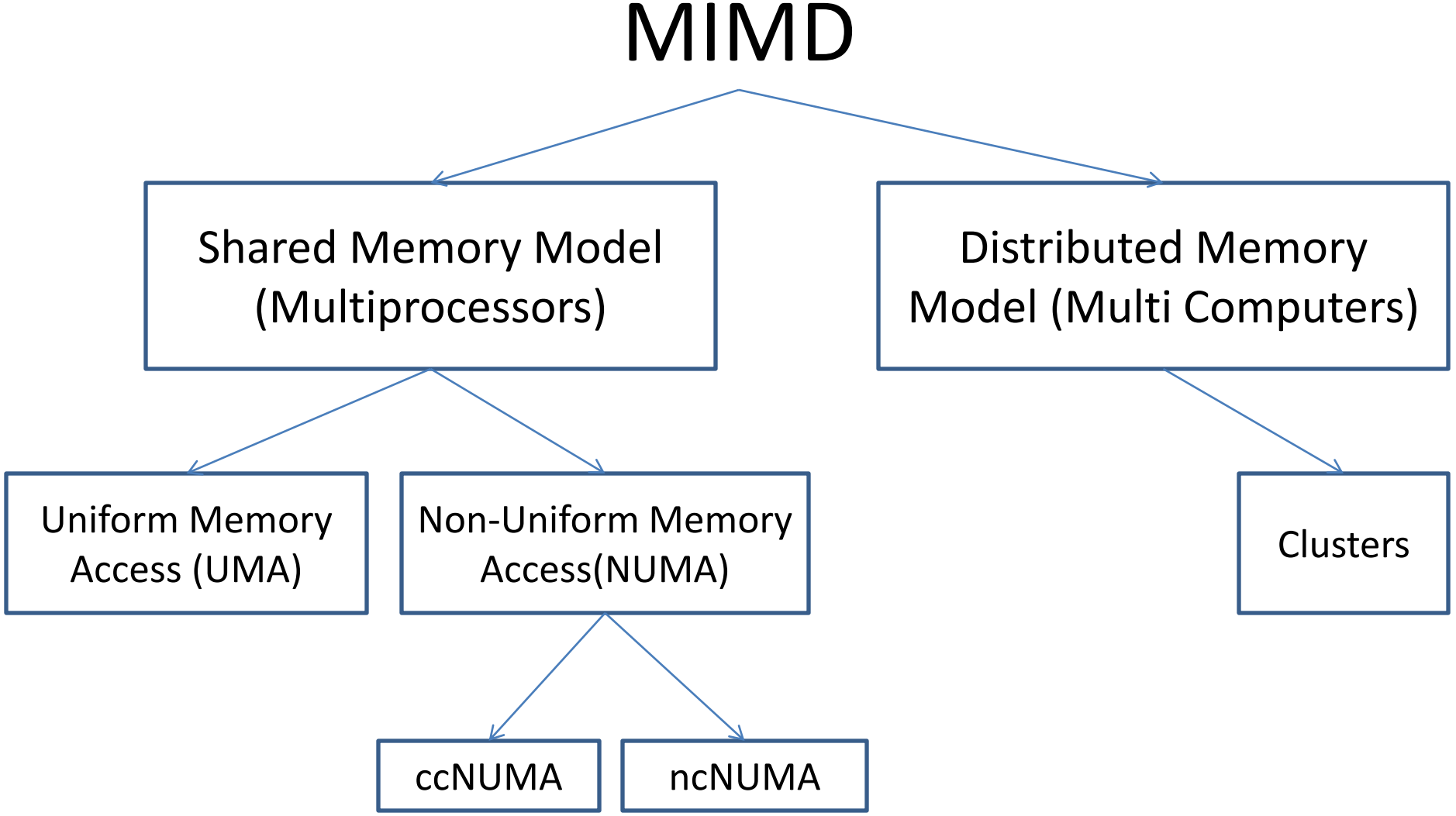


*Image Source: Wikipedia

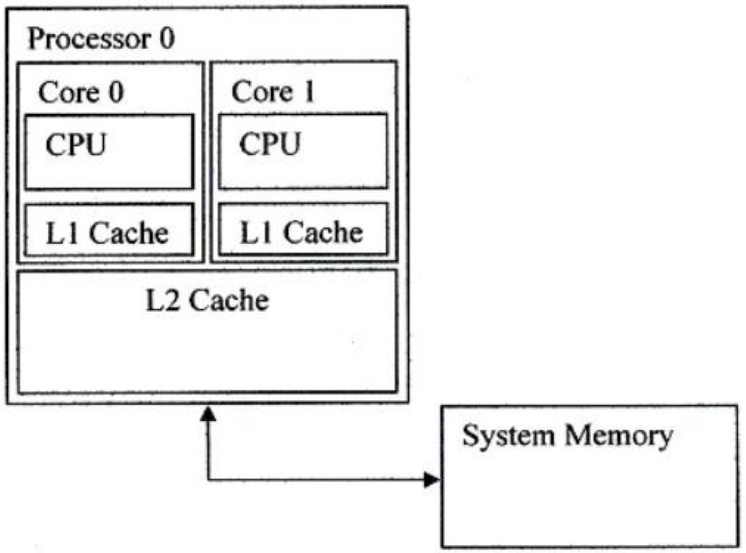

MIMD
Shared Memory Model (Multiprocessors)
Distributed Memory Model (Multi Computers)
Uniform Memory Access (UMA)
Non-Uniform Memory Access(NUMA)
Clusters
ccNUMA
ncNUMA

Multicore

- Multiple Cores /Chip & Single PU
- Independent L1 cache and Shared L2 cache.

Multiprocessors

- Single or Multiple Cores /Chip & Multiple PUs
- Independent L1 cache and Independent L2 cache.



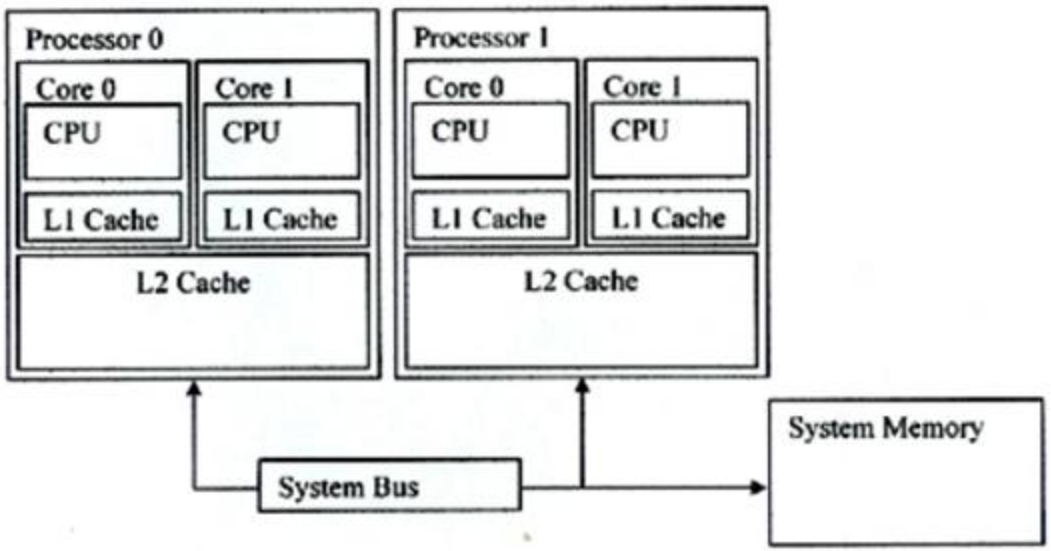
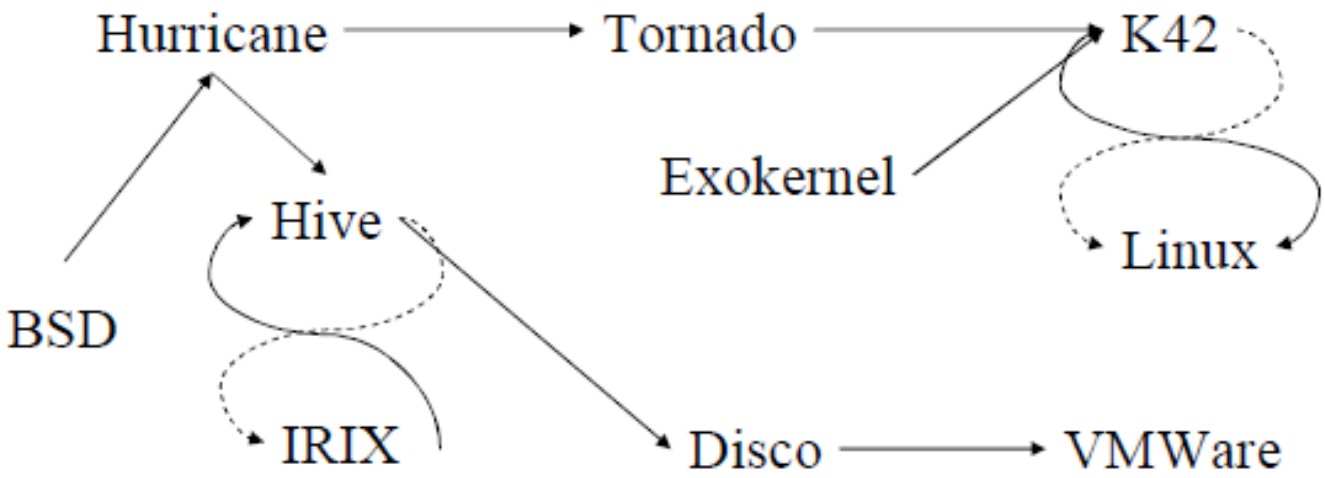


Image Source: Slides of Deniz Altinbuken

Challenges for System Software

- Scalability
- Flexibility
- Reliability
- Performance

History


Hurricane
Tornado
K42
Exokernel
Linux
Hive
BSD
IRIX
Disco
VMWare

Approaches

- Disco
  - Uses a Virtual Machine Monitor
  - Page Replication & Page Migration
  - Copy on write & Virtual subnet
- Tornado
  - Designs an multiprocessor OS
  - Uses Object Oriented Design
  - Clustered Objects
  - A New locking strategy
  - Protected procedure call

DISCO: Running Commodity Operating system on Scalable Multiprocessor

- Edouard Bugnion, VP, Cisco
  - Phd from Stanford, Co-Founder of VMware, key member of Sim OS, Co-Founder of “Nuova Systems”
- Scott Devine, Principal Engineer, VMware
  - Phd from Stanford, Co-Founder of VMware, key member of Sim OS
- Mendel Rosenblum, Associate Prof, Stanford
  - Phd from UC Berkley, Co-Founder of VMware, key member of Sim OS

Virtual Machine Monitors (VMM)

- Definition: *VMM is a hardware virtualization techniques that allow multiple operating systems, termed guests, to run concurrently on a host computer.*
- Classification
  - Type 1: run directly on the host's hardware
  - Type 2: run within a conventional operating system environment

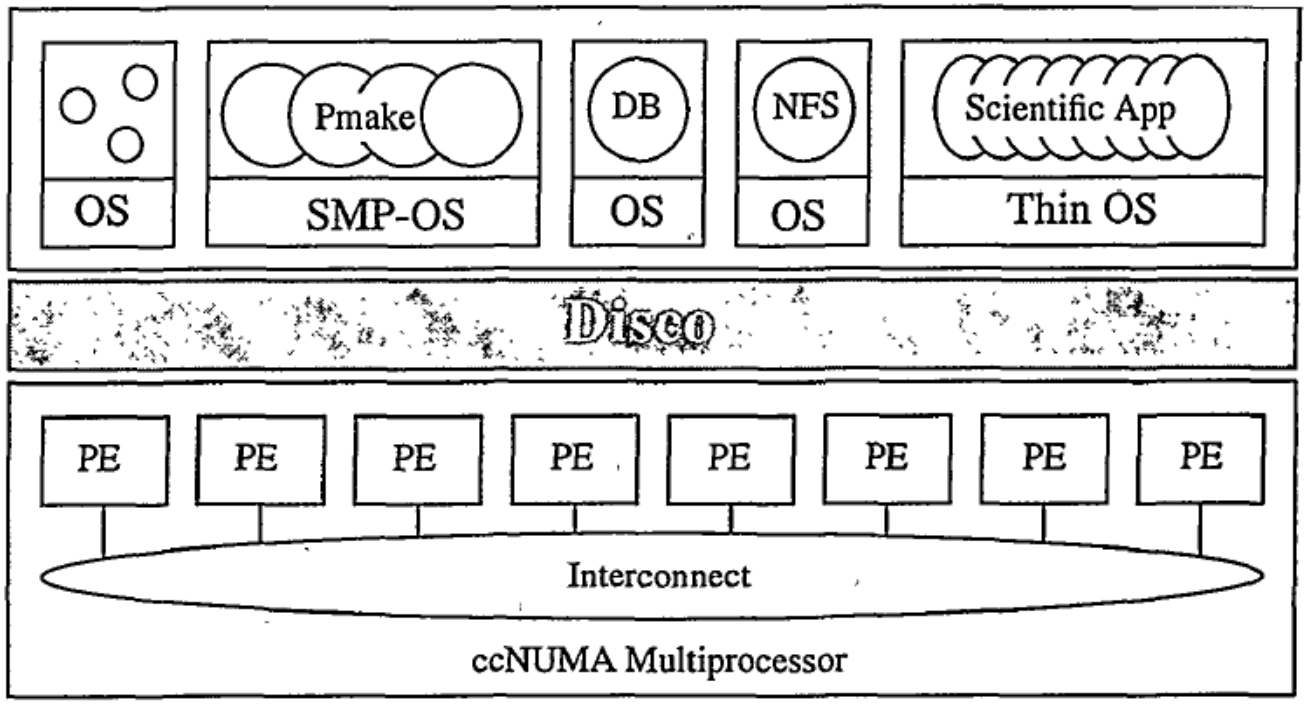
Return of Virtual Machines Monitors

- Cost of development is less
- Less risk of introducing software bugs
- Flexibility to support wide variety of workloads
- NUMA memory management is hidden from guest OS.

Disco: Challenges

- Overheads
  - Additional Execution
  - Virtualization I/O
  - Memory management for multiple VMs
- Resource Management
  - Lack of information to make good policy decisions
- Communication & Sharing
  - Interface to share memory between multiple VMs

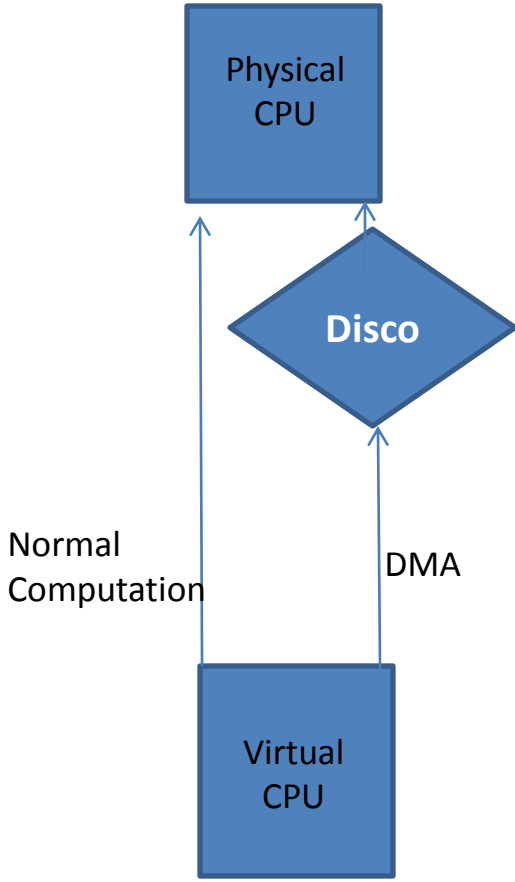
Disco: Interface



- Processors – Virtual CPU
- Memory
- I/O Devices

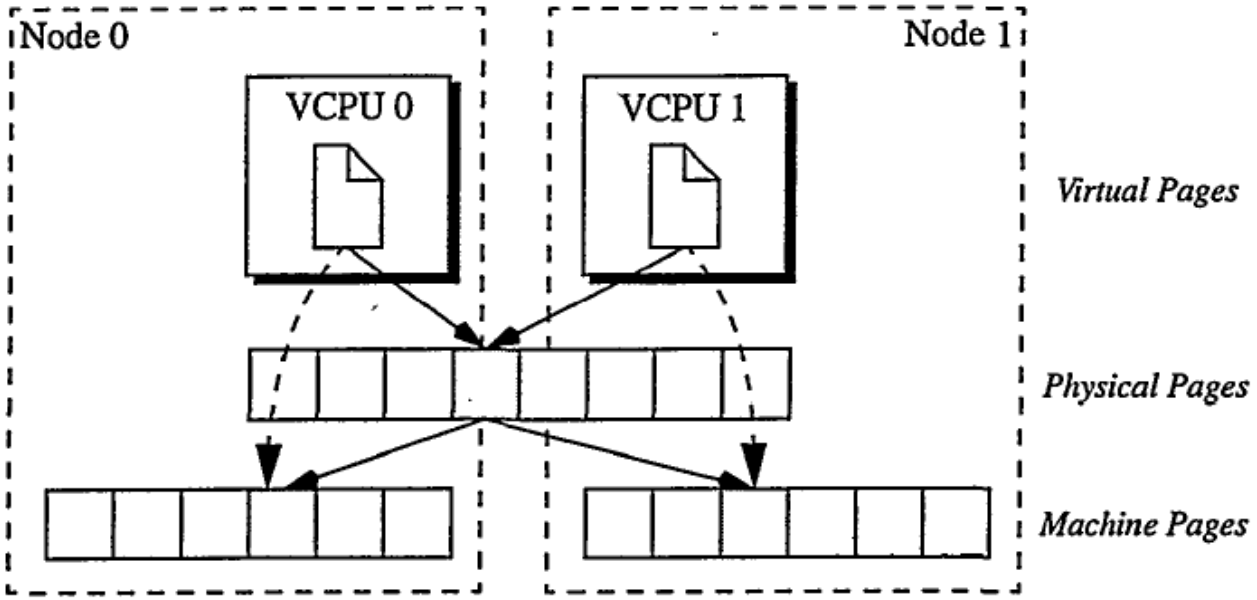
Disco: Virtual CPUs

- Direct Execution on the real CPU
- Intercept Privileged Instructions
- Different Modes:
  - Kernel Mode: Disco
  - Supervisor Mode: Virtual Machines
  - User Mode: Applications



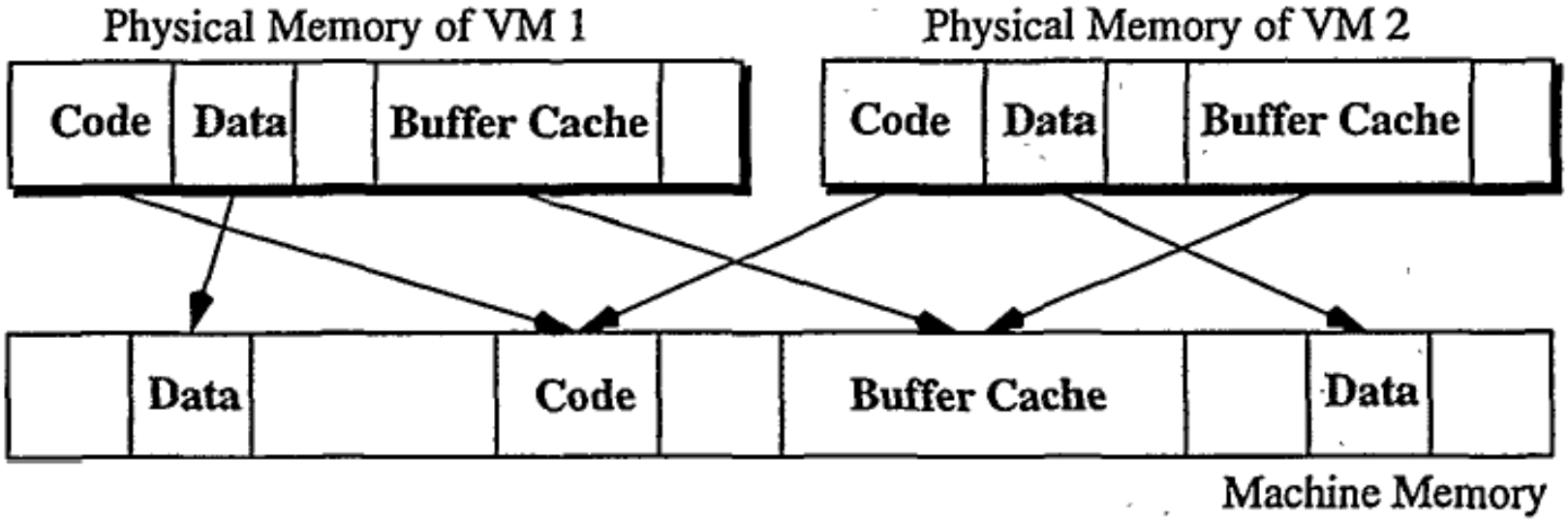
Disco: Memory Virtualization

- Adds a level of address translation
- Uses Software reloaded TLB and pmap
- Flushes TLB on VCPU Switch
- Uses second level Software TLB



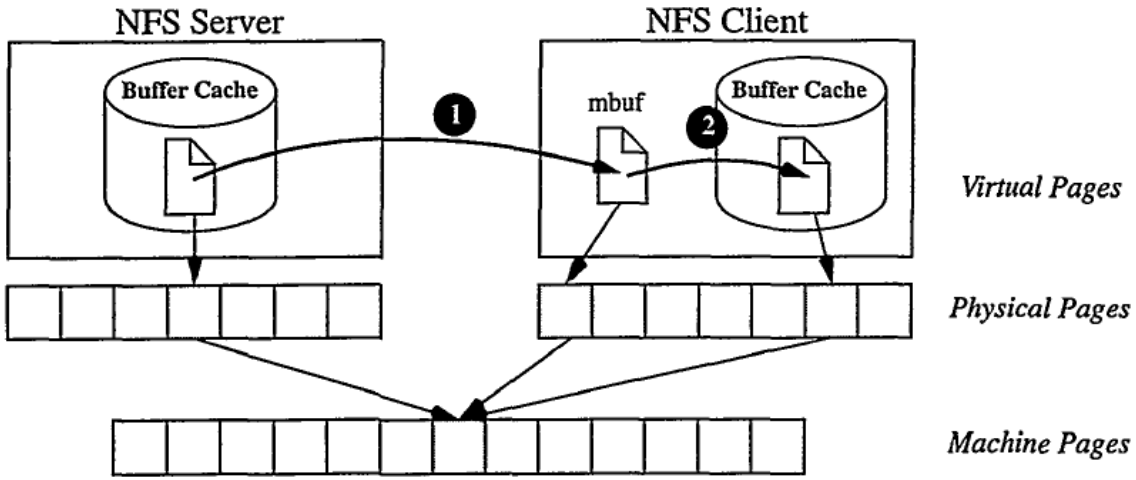
Disco: Memory Management

- Affinity Scheduling
- Page Migration
- Page Replication
- memmap



Disco: I/O Virtualization

- Virtualizes access to I/O devices and intercepts all device access
- Adds device drivers in to OS
- Special support for Disk and Network access
  - Copy-on-write
  - Virtual Subnet
- Allows memory sharing between VMs agnostic of each other



Running Commodity OSes

- Changes for MIPS Architecture
  - Required to relocate the unmapped segment
- Device Drivers
  - Added device drivers for I/O devices.
- Changes to the HAL
  - Inserted some monitor calls in the OS

Experimental Results

- Uses Sim OS Simulator for Evaluations

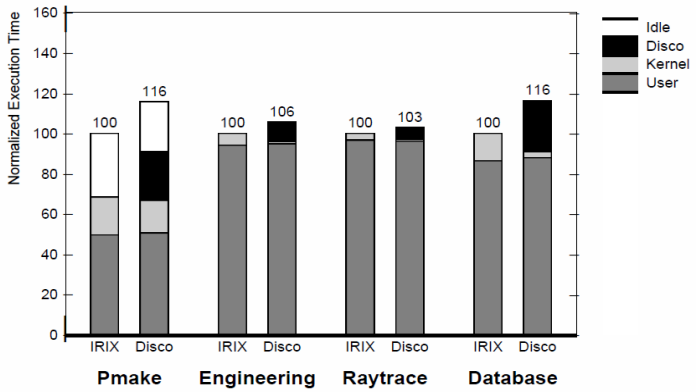


Fig. 6. Overhead of virtualization

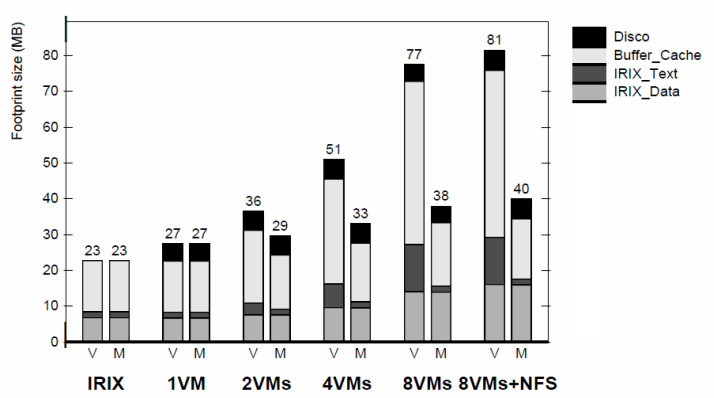


Fig. 7. Data sharing in Disco between virtual machines

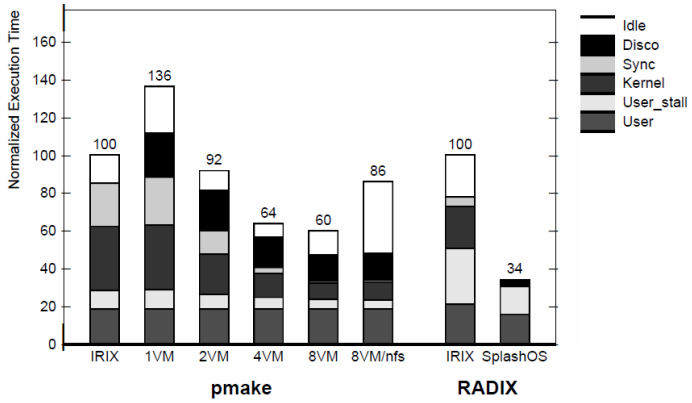


Fig. 8. Workload scalability under Disco

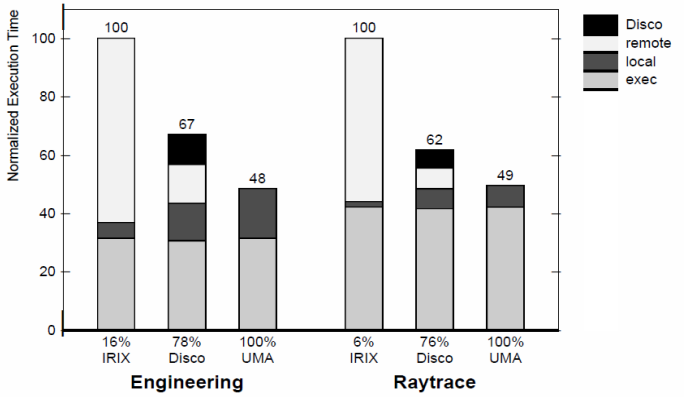


Fig. 9. Performance benefits of page migration and replication

Disco: Takeaways

- Develop system s/w with less effort
- Low/Modest overhead
- Simple solution for Scalable Hardware
- Future Scope
  - Provide Resource management mechanisms comparable to Customized Operating Systems

Tornado Maximizing Locality and Concurrency in a SMMP OS

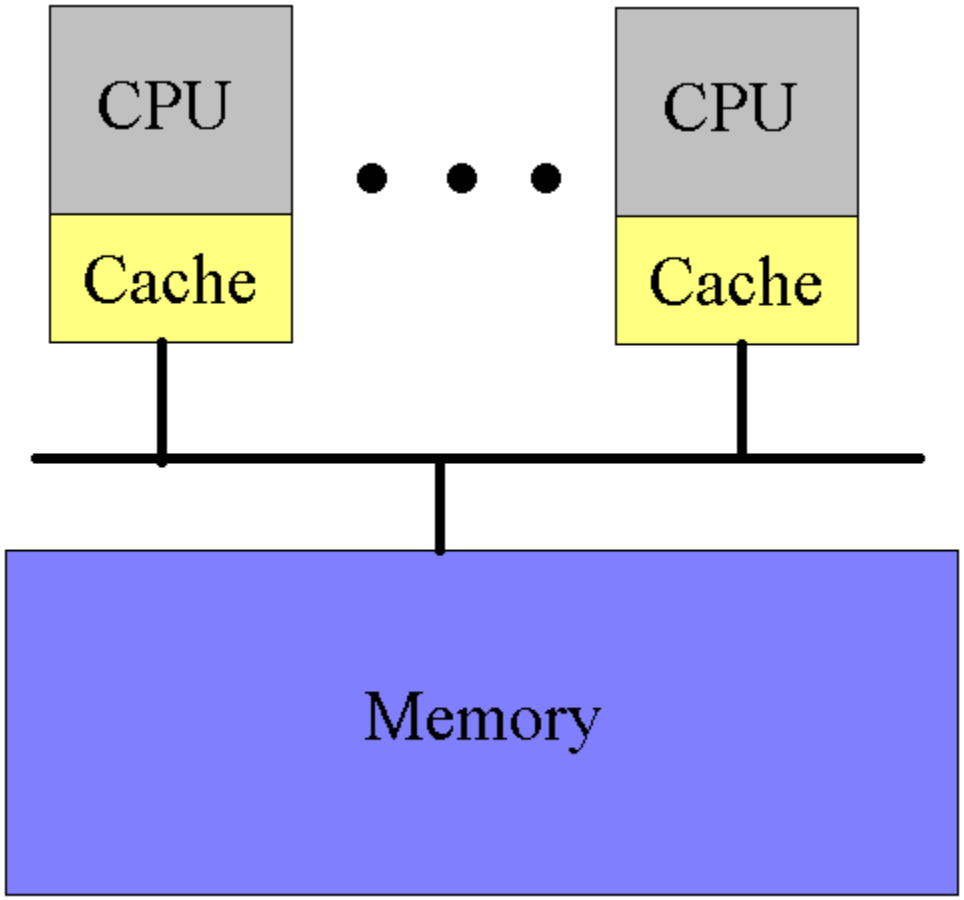
- Ben Gamsa, Orran Krieger
  - Phd from University of Toronto
- Jonathan Appavoo, Asistant Prof., Boston University
  - Phd from Uni of Toronto, worked as researcher in IBM
- Michael Stumm, Prof, University of Toronto
  - Works with IBM research to develop K42 OS

Locality

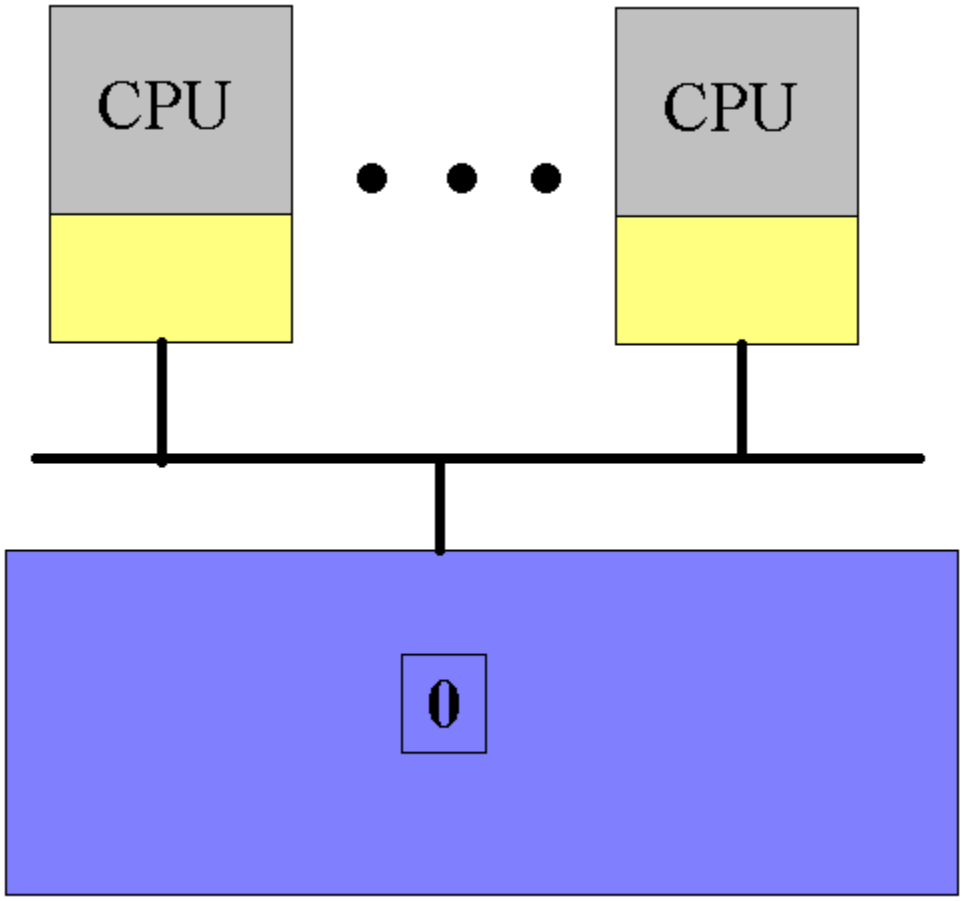
Types of Locality

- Temporal Locality
  - If at one point in time a particular memory location is referenced, then it is likely that the same location will be referenced again in the near future.
- Spatial Locality
  - If a particular memory location is referenced at a particular time, then it is likely that nearby memory locations will be referenced in the near future.

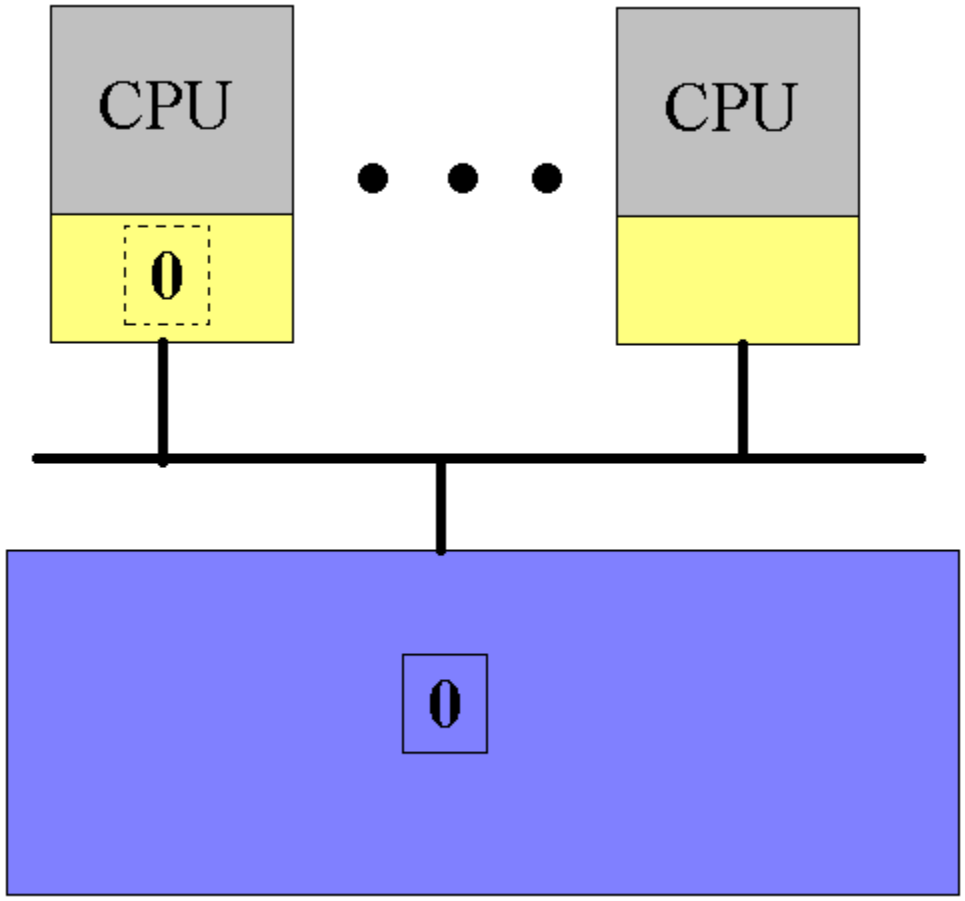
Example: Counter



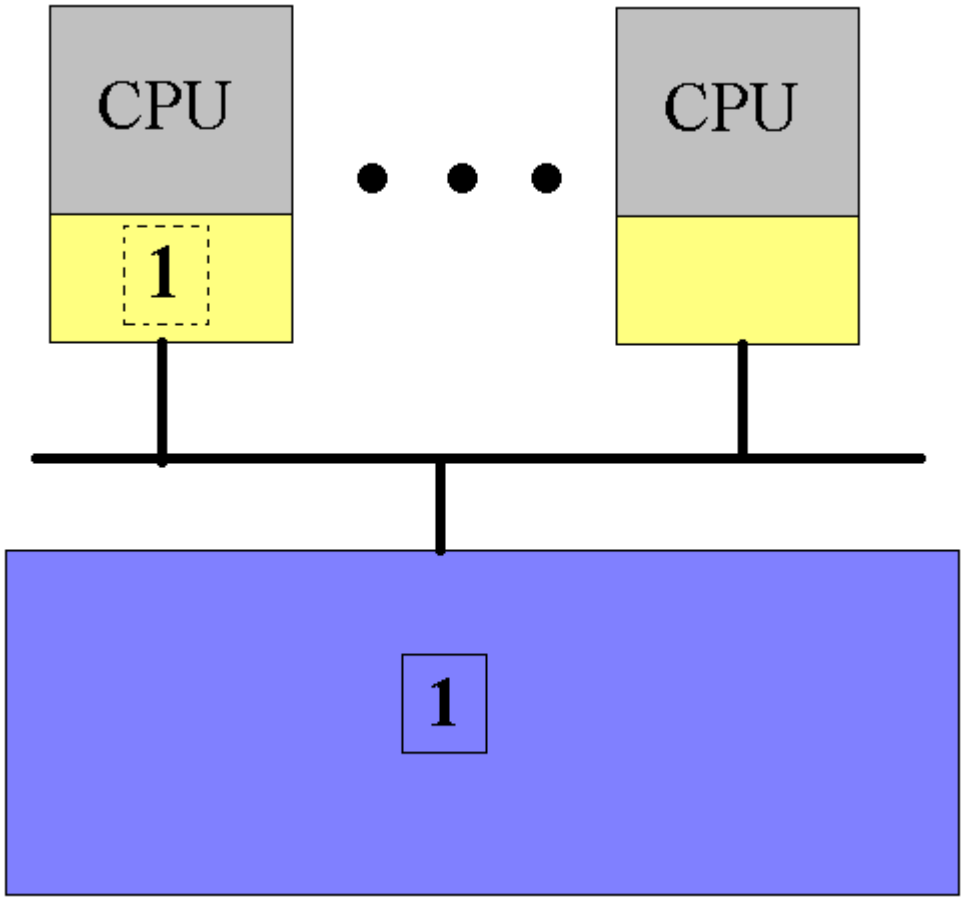
Example: Counter



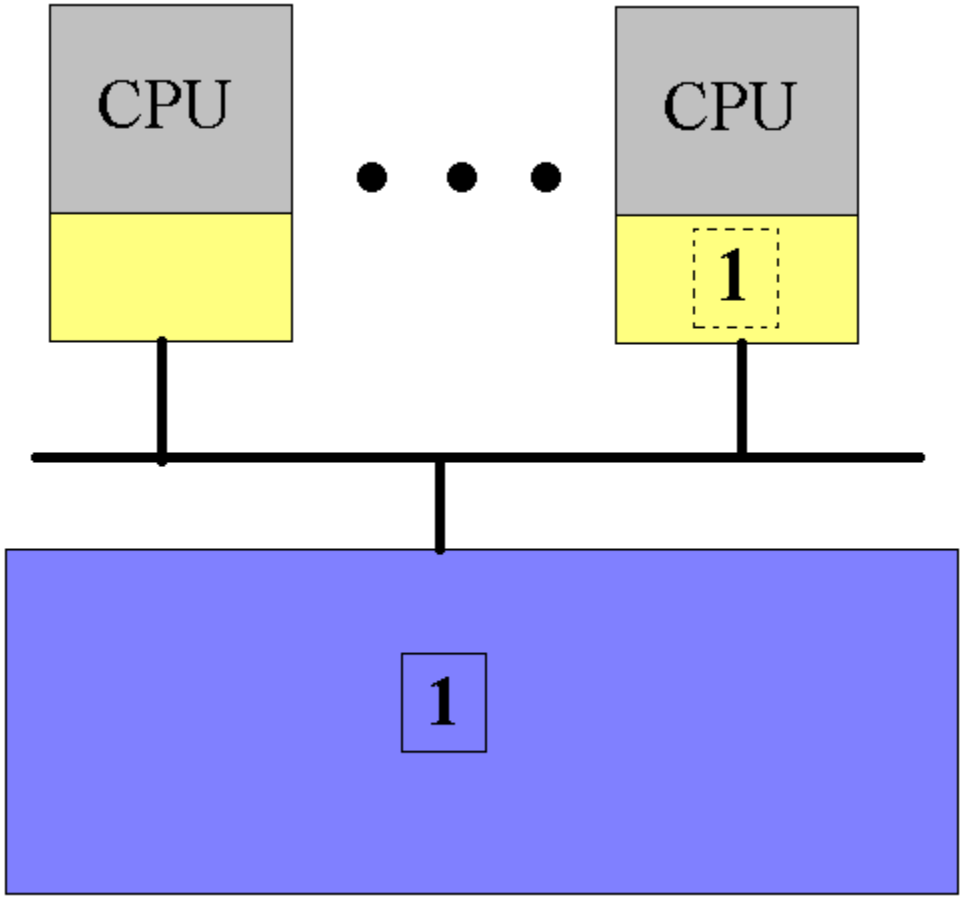
Example: Counter



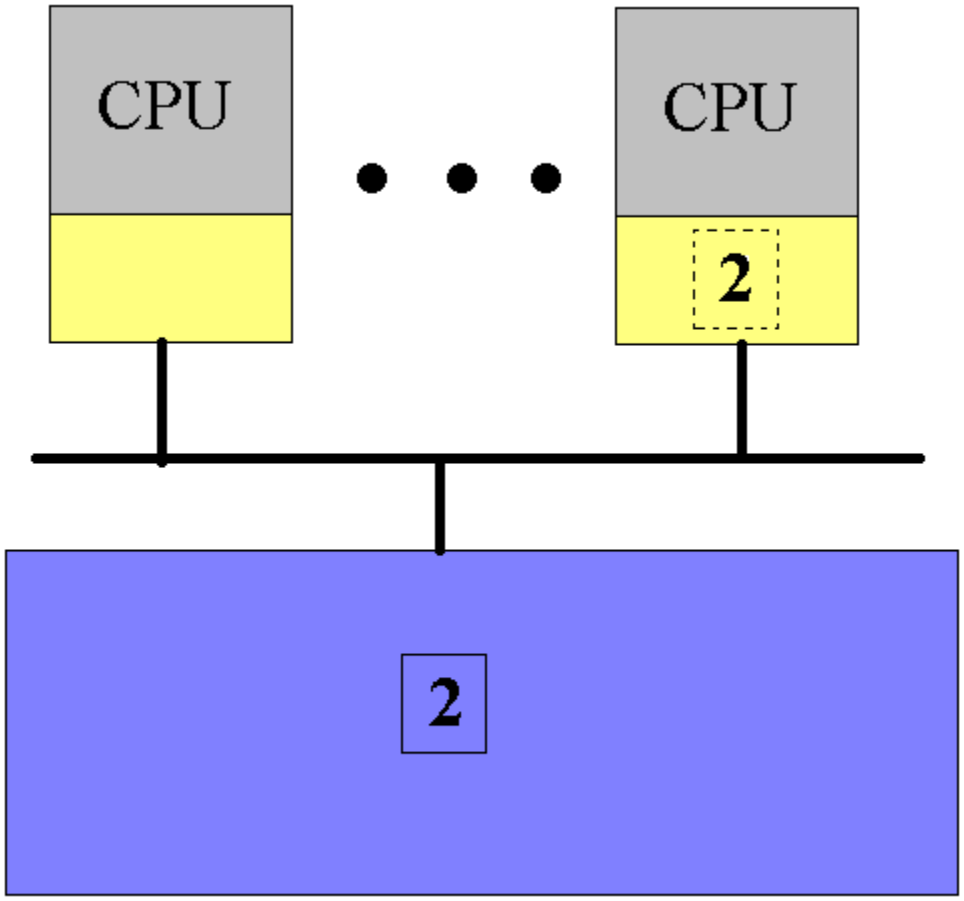
Example: Counter



Example: Counter



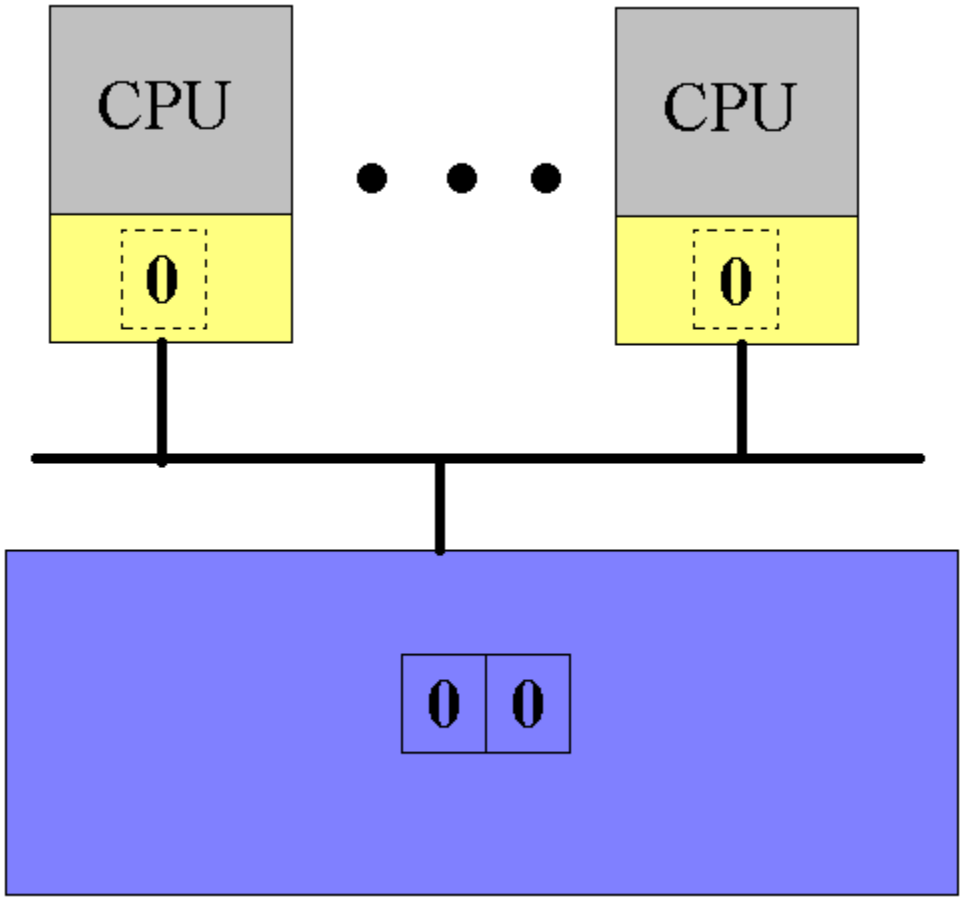
Example: Counter



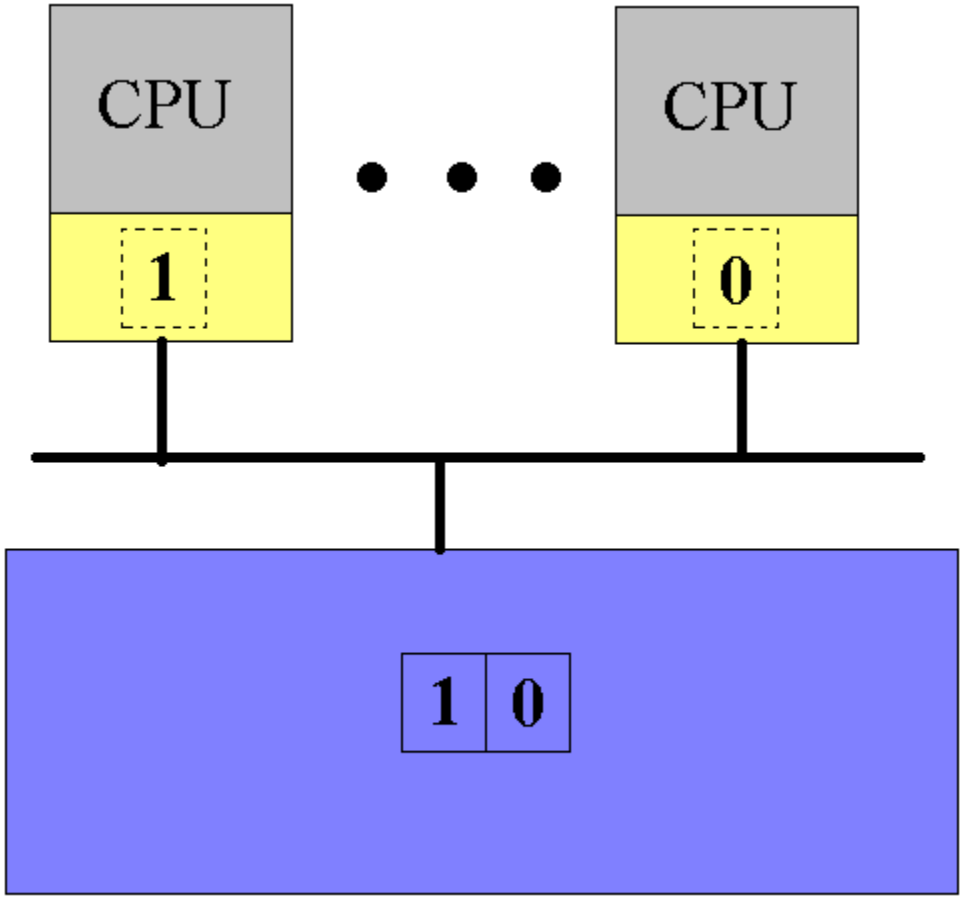
Example: Counter

- Sharing the counter requires moving it back and forth between the CPU caches
- Solution??
  - Split the counter into an array of integers
  - Each CPU gets its own counter

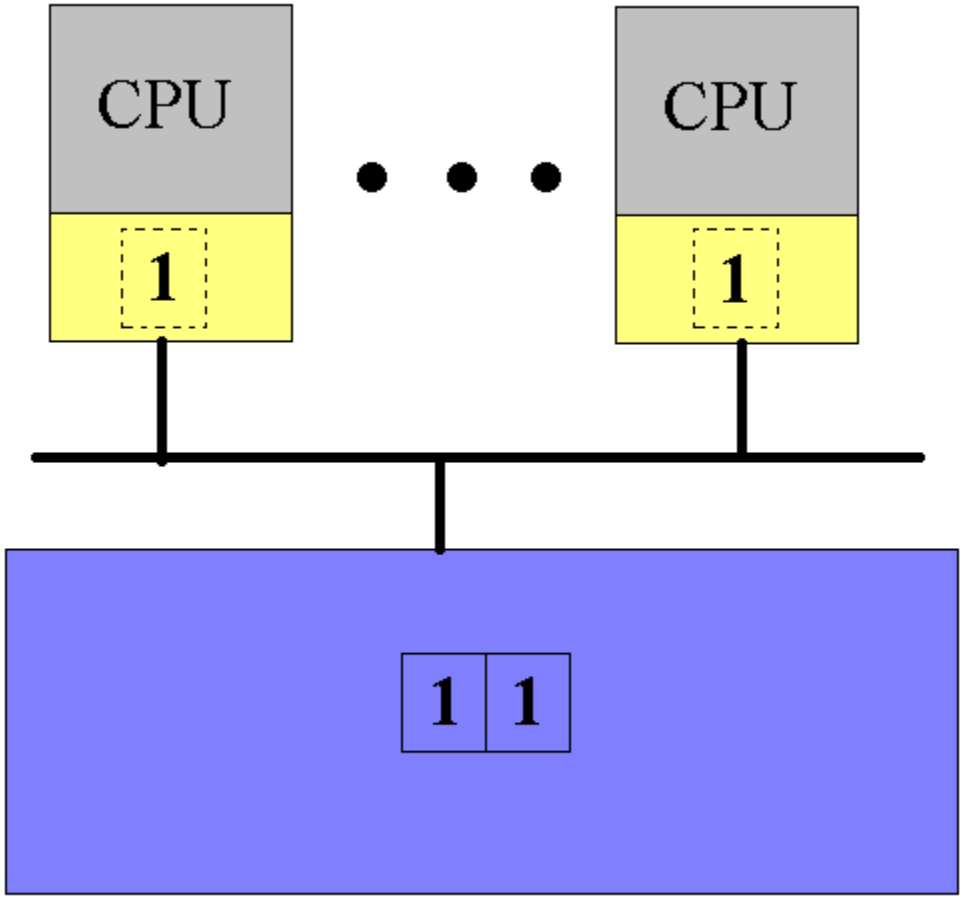
Example: Counter



Example: Counter



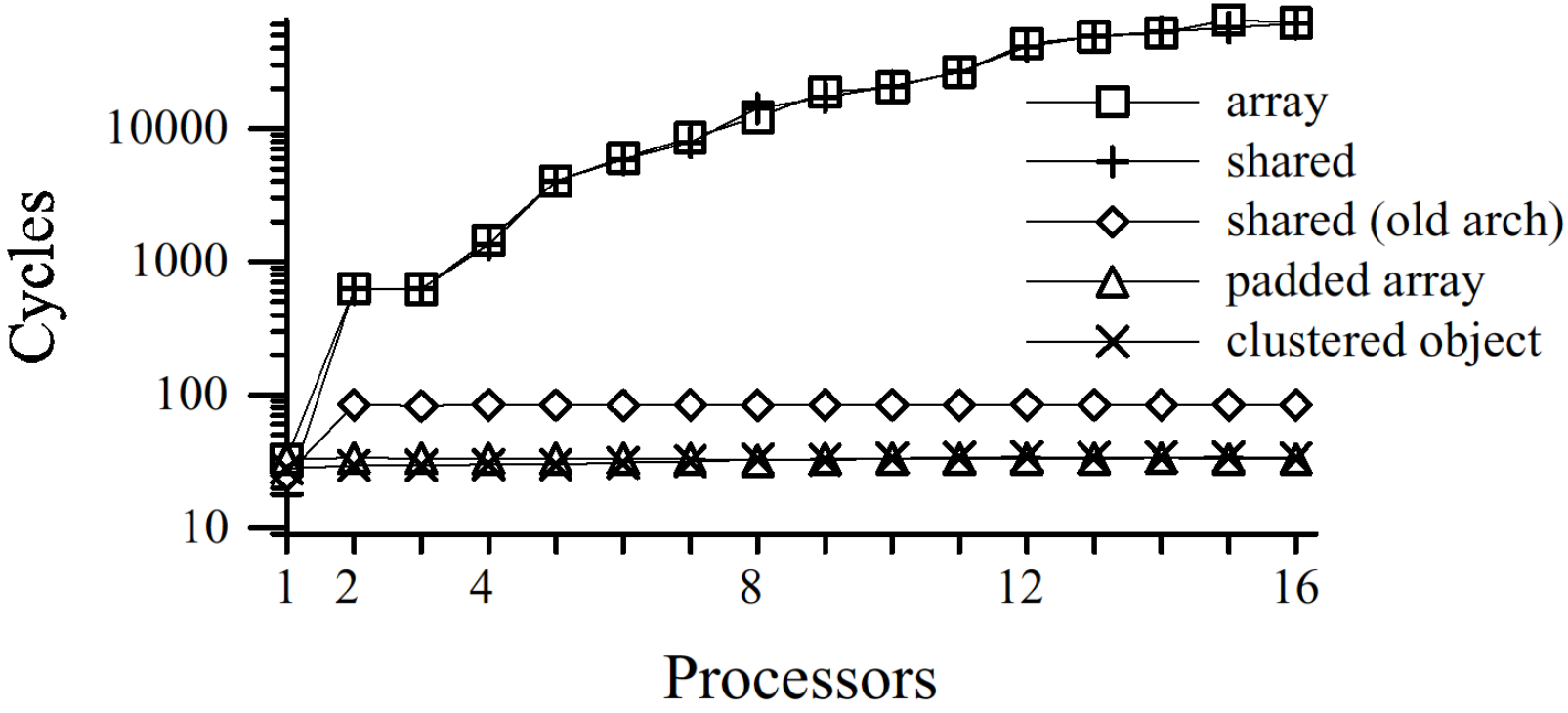
Example: Counter



Example: Counter

- Using an array of counters seems to solve our problem…
- But what happens if both array elements map to the same cache line?
  - False sharing
- Solution:
  - Pad each array element
    - Different elements map to different cache lines

Example: Counter



Challenges

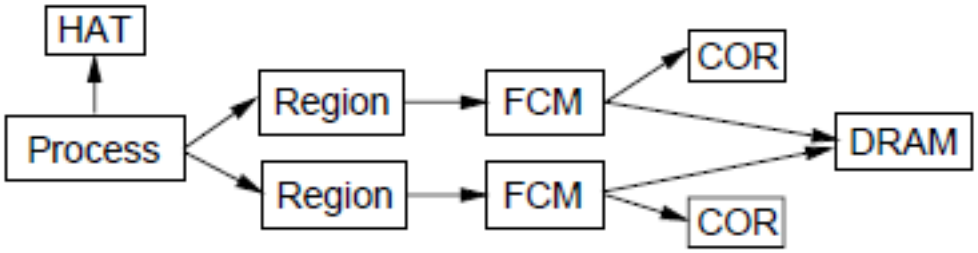
- Minimize read/write and write sharing to minimize cache coherence
- Minimize false sharing
- Minimize distance between the accessing processor and target memory

Tornado's Approach to Maximize Locality

- Object Oriented Structure
- Clustered Objects
- Synchronization
- Protected Procedure call

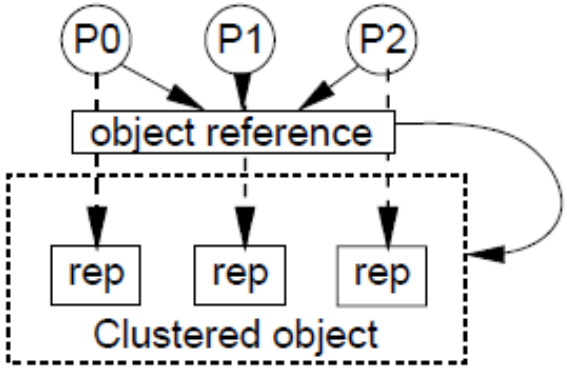
Object Oriented Structure

- Each OS resource is represented as a separate object
- All locks and data structures are internal to the objects
- This structure allows different resources to be managed
  - without accessing shared data structures
  - without acquiring shared locks



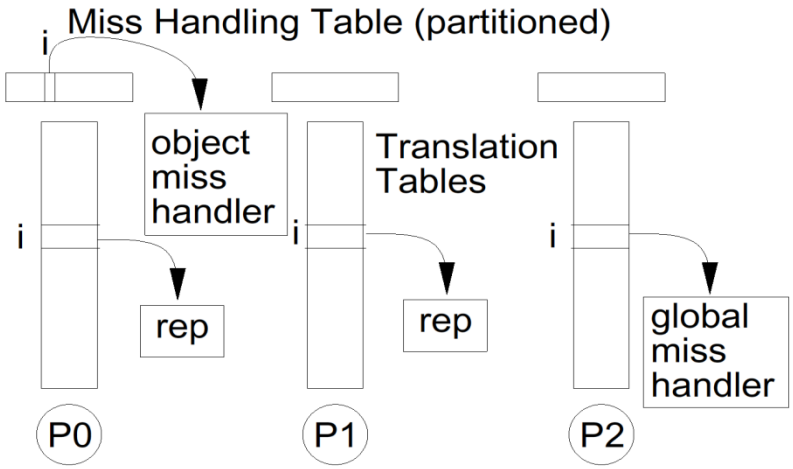
Clustered Objects

- Appear as a single object.
- Multiple “reps” to process object references.
- Uses Replication, Distribution and partitioning
- Benefits
  - Allows multiprocessor optimizations
  - Abstracts the object reference for programmers
  - Enables incremental optimization



Clustered Objects: Implementation

- Uses Translation tables
  - Per processor access to local object
  - Global partitioned table to find rep for given object
  - Default “miss” handler which is called by global miss handler



Synchronization

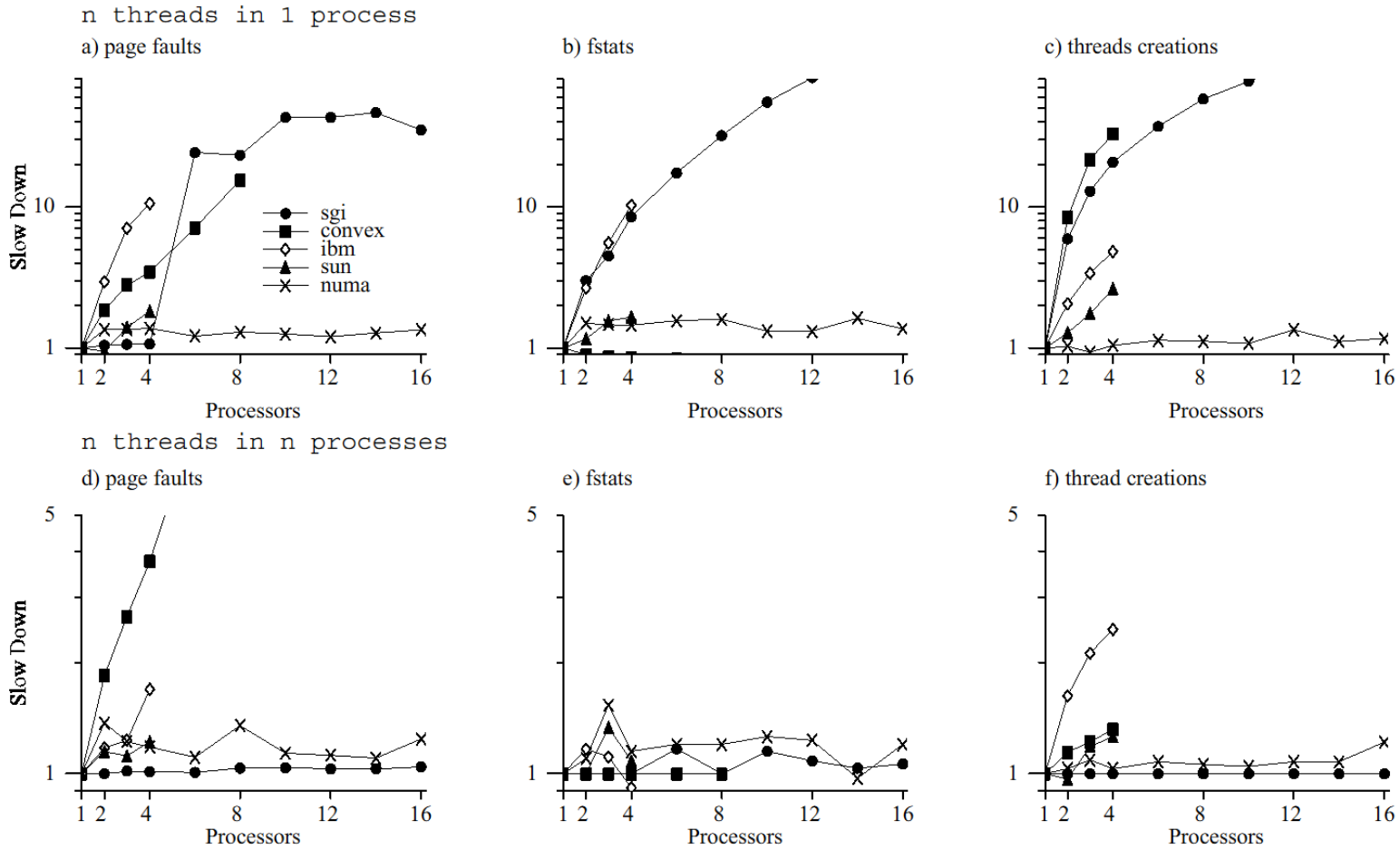
- Separates Locking & Existence Guarantees
  - Locking:
    - Encapsulate lock with in the object
    - Uses Spin then block locks
  - Semi-Automatic Garbage collection
    - Ensures all persistent and temporary object reference removal
    - Maintains list of processor having object refs

Protected Procedure call

- Brings locality and concurrency to IPC
- A PPC is a call from a client object to a server object
- Benefits
  - Client requests are always serviced on their local processor
  - Clients and servers share the CPU in a manner similar to handoff scheduling
  - The server has one thread of control for each client request

Experimental Results

- Uses 16 processor NUMAchine prototype and SIM OS Simulator



Takeaways

- Tornado's increased locality and concurrency has produced a scalable OS design for multiprocessors
- This locality was provided by several key system features
  - An object-oriented design
  - Clustered objects
  - A new locking strategy
  - Protected procedure calls

Perspective

- Disco
  - Uses a Virtual Machine
  - Page Replication & Page Migration
  - Copy on write & Virtual subnet
- Tornado
  - Designs an multiprocessor OS
  - Uses Object Oriented Design
  - Clustered Objects
  - A New locking strategy
  - Protected procedure call

Discussion