- **The impact of operating system structure on memory system performance, Chen, J. B. and Bershad, B. N. SOSP '93**

  - Compares the memory system performance of Mach (Microkernel based OS) and Ultrix (monolithic kernel) on the same hardware. It shows that a number of popular memory system assumptions were true but the impacts might be negligible
  - What did you think of the experiment setup and quality?

OS structure

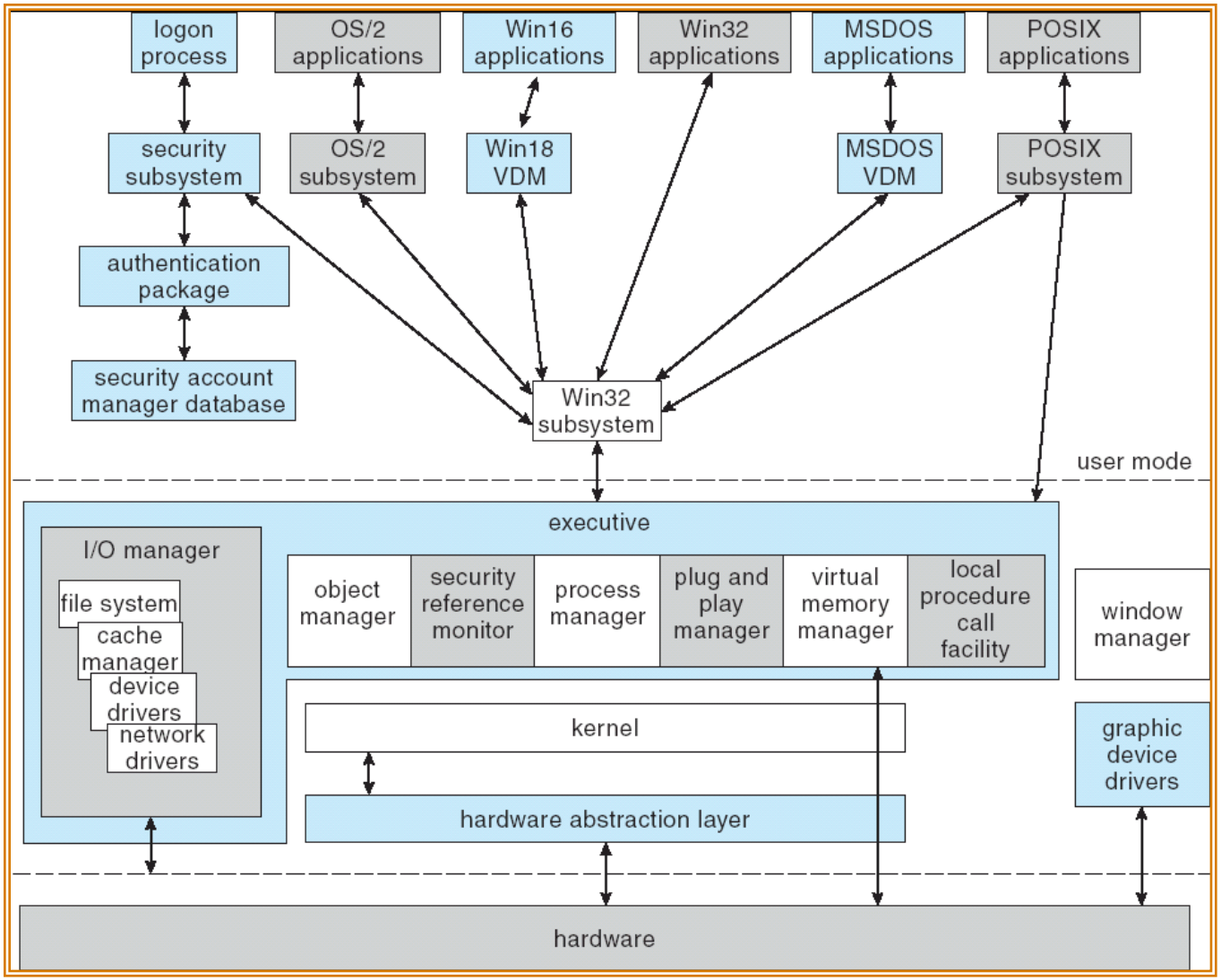
- Monolithic kernel
  - Spaghetti code, Layered
- Microkernel
- Nano kernel

- Question: What is the problem being solved here?

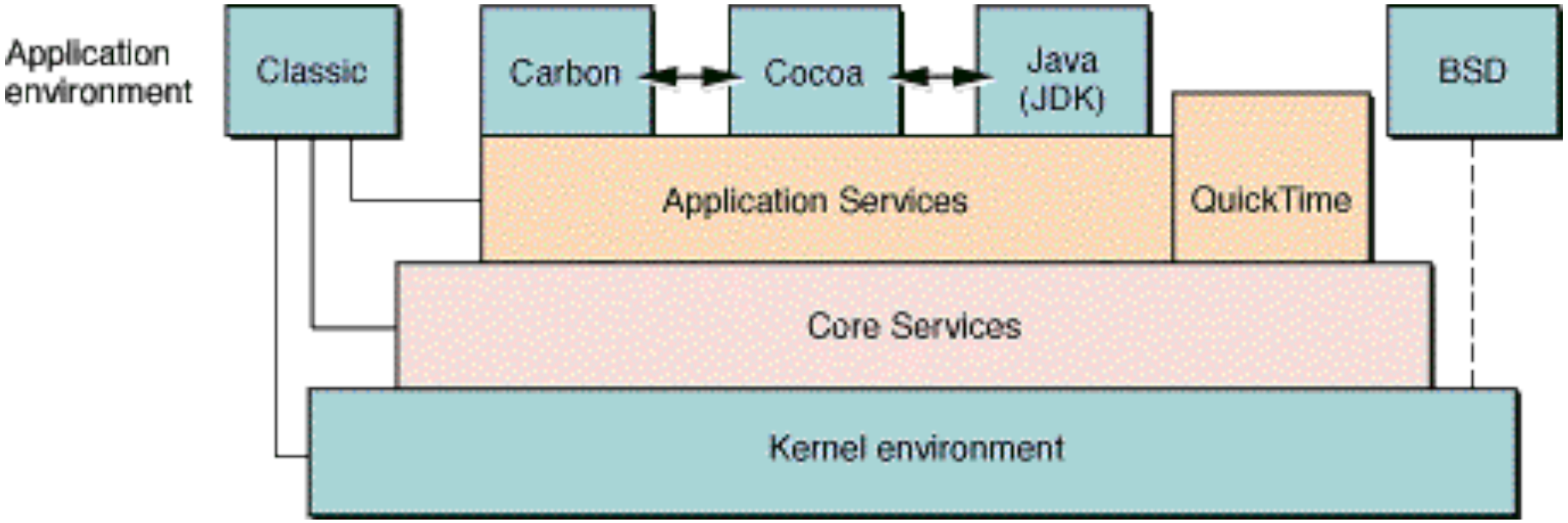
Problem being solved

- Monolithic kernels tend to get big and bloated
  - Question: Why?
- Microkernels can help. Microkernel will have a well thought out and optimized component that implements mechanisms. Application level servers implement policy.
  - Side benefits: You can implement multiple policies (Win32, POSIX) over the same microkernel. Requires: microkernel is well thought out so that the overlap in mechanisms is limited
  - Question: Is it important for Microkernels to be small?

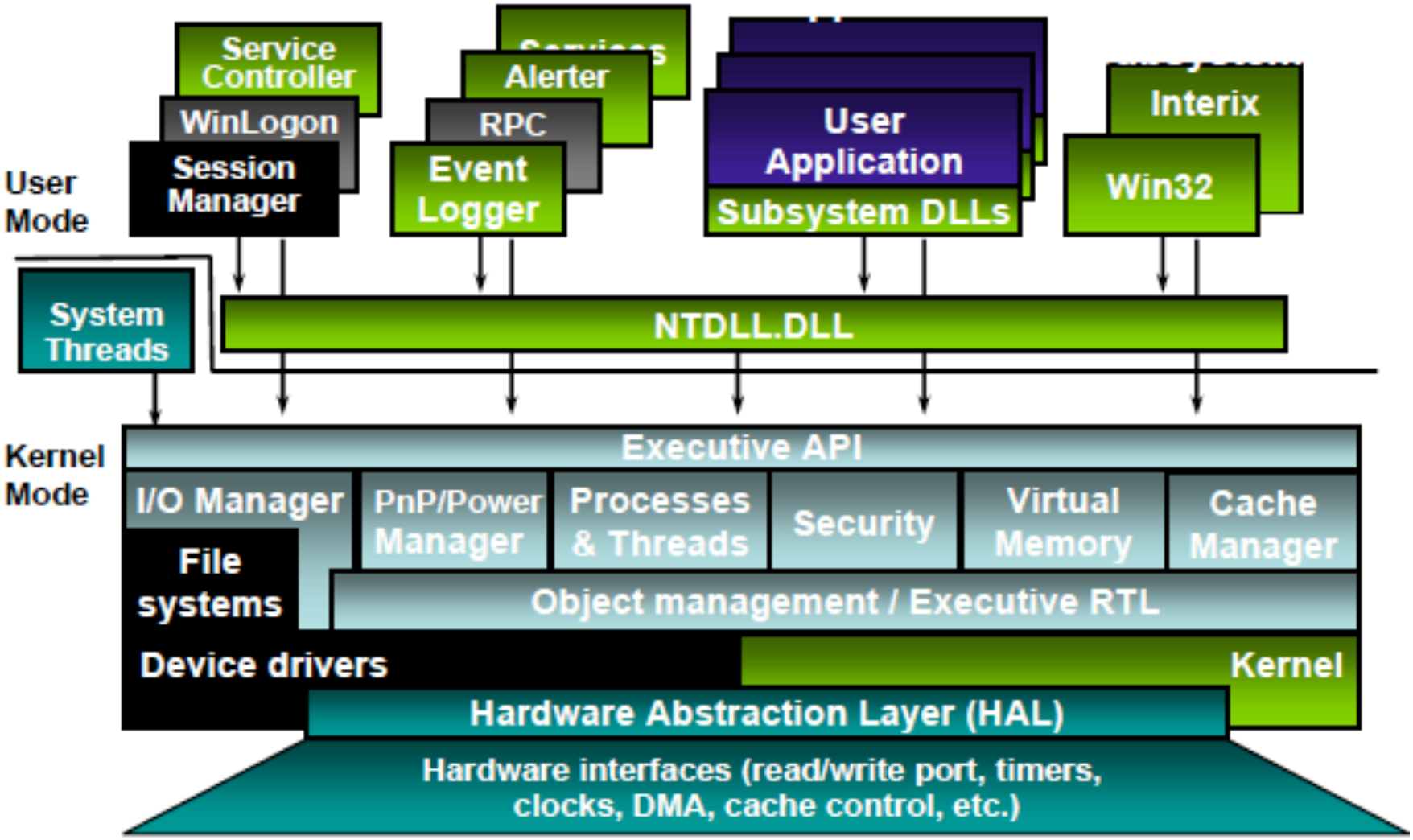
Windows XP architecture



Mac OS X


Application environment
Classic
Carbon
Cocoa
Java (JDK)
BSD
Application Services
QuickTime
Core Services
Kernel environment

Windows 2003 server



Microkernel

- Kernels operate in privileged mode
  - Question: why?
- In monolithic kernel, one needs to enter supervisor mode for all kernel level functionality (e.g. read system time)
  - Question: How?
- In microkernel, kernel operates in supervisor mode. Application level servers, operate in user mode. Communication between application level servers can operate using kernel (performance hit). Hence, communications is an important OS functionality in Mach. Application to application communications

Consider a certain operation

- Say, read current time
  - Monolithic kernel: system call to kernel
  - Microkernel: application->library->microkernel
- Say, write a block
  - Monolithic kernel: system call to kernel
    - Performance can be slow because of code bloat
  - Microkernel: application->library->microkernel->application servers and microkernel
    - Depending on what is being serviced by the microkernel, the rest might be in a application server
- How do you optimize to make microkernel faster?
  - On to the assigned paper

Assertions

- OS has less instruction and data locality than user programs
  - Question: What is instruction and data locality?
  - Implication: The operating system isn't getting faster as fast as user programs.
  - Question: Should it? Does it matter when we are talking about monolithic vs micro kernel?
  - Question: Some applications (e.g.) are entirely kernel bound (perhaps IO). Performance perspective, do you prefer monolithic or microkernel?
  - Observation: Instruction locality is worse than data locality during system execution (Mach worse than Ultrix)
    - Question: How does locality work? Why is this happening?

Assertion 2

- System execution is more dependent on instruction cache behavior than is user execution
  - Question: how is locality related to caching behavior?
  - System instructions exhibit more I-cache misses
  - Implications: A balanced cache system for user programs may not be balanced for the system

Assertion 3

- Collision between user and system references lead to significant performance degradation in the memory system (cache and TLB)
  - Question: What is a TLB?
  - Implication: A split user/system cache could improve performance
  - Observation: not true. Impact of Mach's MK structure is not significant
    - Cache miss penalty following a voluntary context switch (client->server) is not significant
      - Client side: I-cache miss is low, D-cache is high
      - Server side: I-cache and D-cache are poor

Assertion 4

- Self-interferences is a problem in system instruction reference streams
  - Insufficient cache associativity
  - Implication: Increased cache associativity and/or the use of text placement tools could improve performance
  - Observation: Compared to Ultrix, associativity eliminates a lower percentage of Machs cache misses because of its greater demand for cache resources

Assertion 5

- System block memory operations are responsible for a large percentage of memory system reference costs
  - Question: What is a block memory operation?
  - Implications: Programs that incur many block memory operations will run more slowly than expected
  - Observation: Mach references more block data than Ultrix. In Mach, block operations occur within kernel as part of VM and IPC system and within the UNIX server as part of the file system. In Ultrix, block operations occur entirely within the kernel and mostly due to VM and file system operation

Assertion 6

- Write buffers are less effective for system reference streams
  - Question: What is a streaming write, what is a write buffer?
  - Implications: A write buffer adequate for user code may not be adequate for system code
  - Observation write buffers are less effective

Assertion 7

- Virtual page mapping strategies can have significant impact on cache performance
  - Question: What is a virtual page mapping strategy? What is it trying to solve?
  - Implications: Systems should support a flexible page mapping interface, and should avoid default strategies that are prone to pathological behavior
  - Observation: True. Random strategy is less likely to induce consistent poor behavior. Strategies are OS architecture independent

Discussion

- How relevant are these observations?
  - Modern hardware (PDA, desktop, servers)
  - Modern CPU's (multicore)