Outline

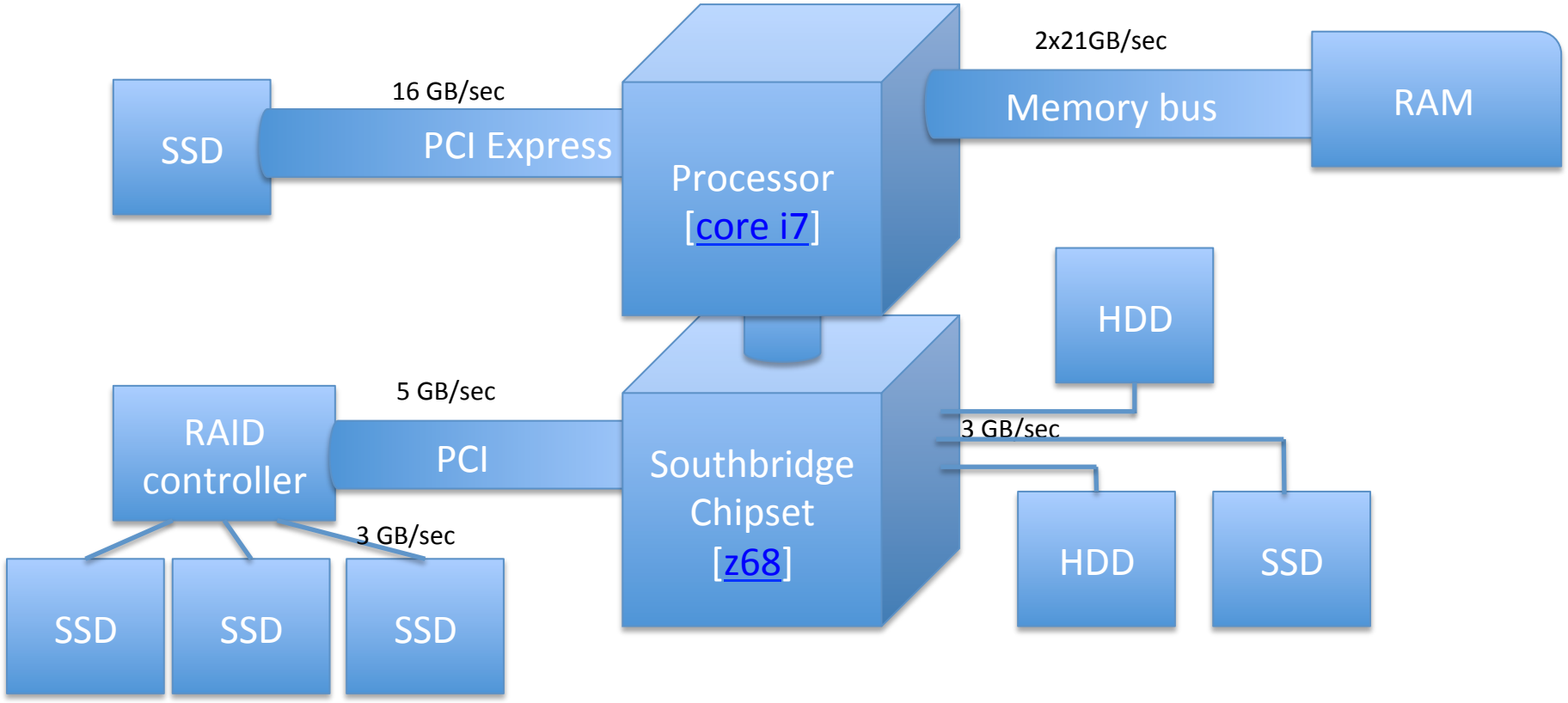
Configuration

- IO stack
  - SSDs and HDDs
  - RAID controller
  - Storage Area Network
  - block layer and file system
  - virtual storage
- Multi-core
- Network stack

Tuning

- Tuning IO priorities
- Tuning Virtual Storage
- Tuning for maximum concurrency
- Tuning for RAM locality
- The priority inversion problem
- Transferring large files
- How to throw hardware at a problem?

IO Architecture



LOOK UP: Smart Response Technology (SSD caching managed by z68)

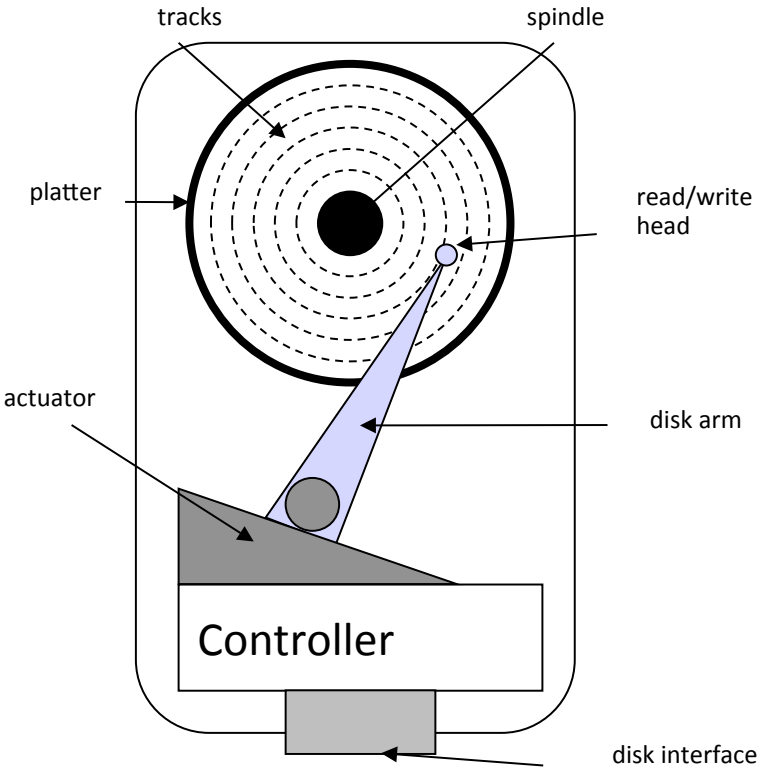
IO Architecture

Exercise 3.1:
How many IO per second can a core i7 processor issue (assume that the core i7 performs at 180 GIPS and that it takes 500000 instructions per IO).

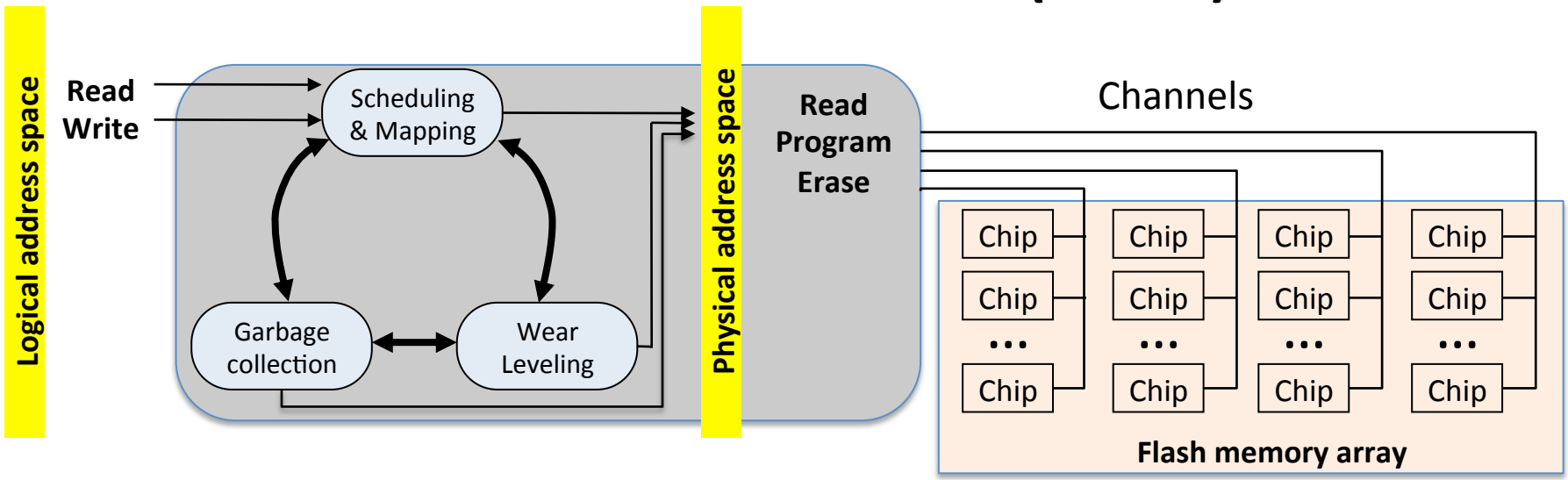
Exercise 3.2:
How many IO per second can your laptop CPU issue (look up the MIPS number associated to your processor).

Exercise 3.3:
Define the IO architecture for your laptop/ server.

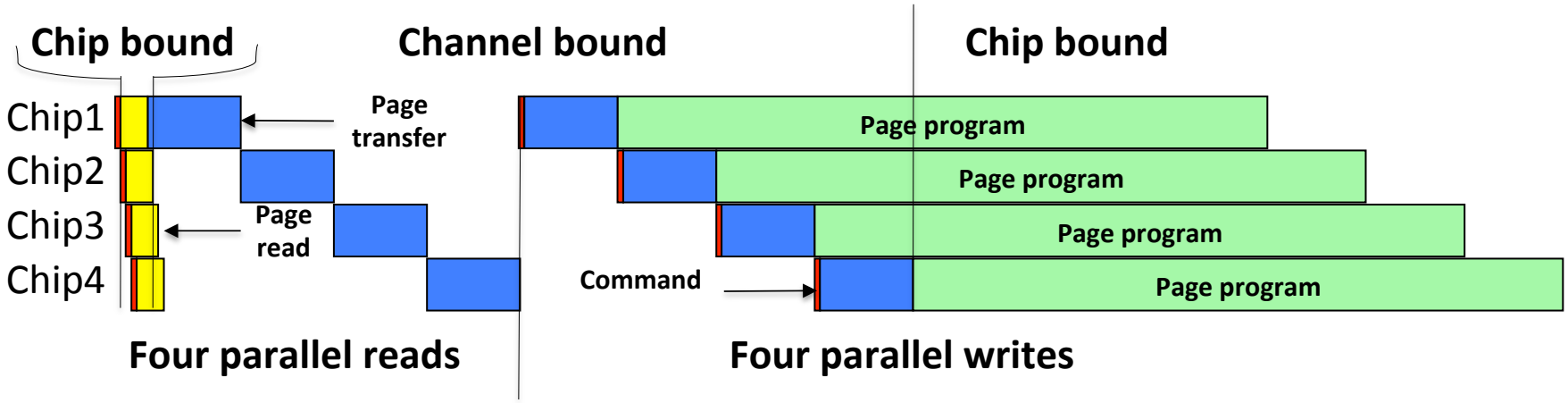
Hard Drive (HDD)



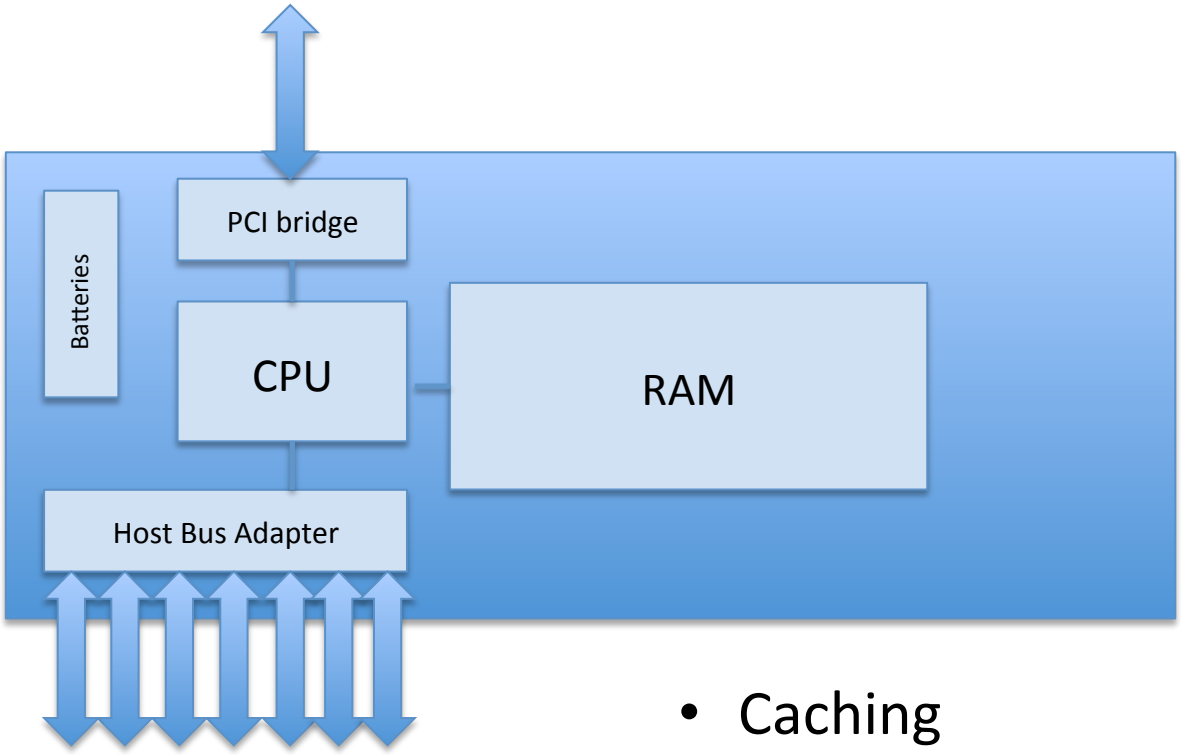
Solid State Drive (SSD)



Example on a disk with 1 channel and 4 chips



RAID Controller



- Caching
  - Write-back / write-through
- Logical disk organization
  - JBOD
  - RAID

RAID

Redundant Array of Inexpensive Disks

- RAID 0: Striping *[n disks]*
- RAID 1: Mirroring *[2 disks]*
- RAID 10: Each stripe is mirrored *[2n disks]*
- RAID 5: Floating parity [3+ disks]

Exercise 3.4:
A – What is the advantage of striping over magnetic disks?
B- what is the advantage of striping over SSDs?

Storage Area Network (SAN)

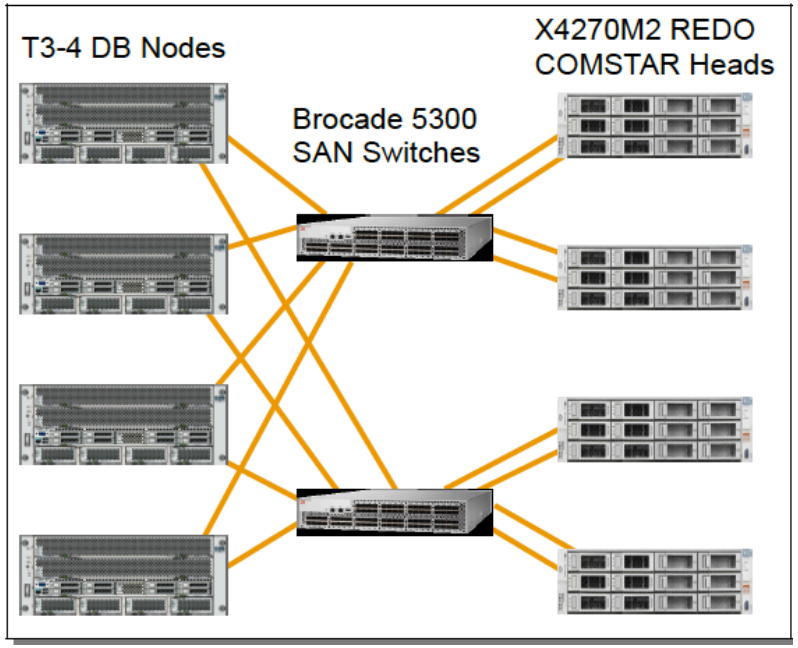
"A storage area network is one or more devices communicating via a serial SCSI protocol (such as FC, SAS or iSCSI)."

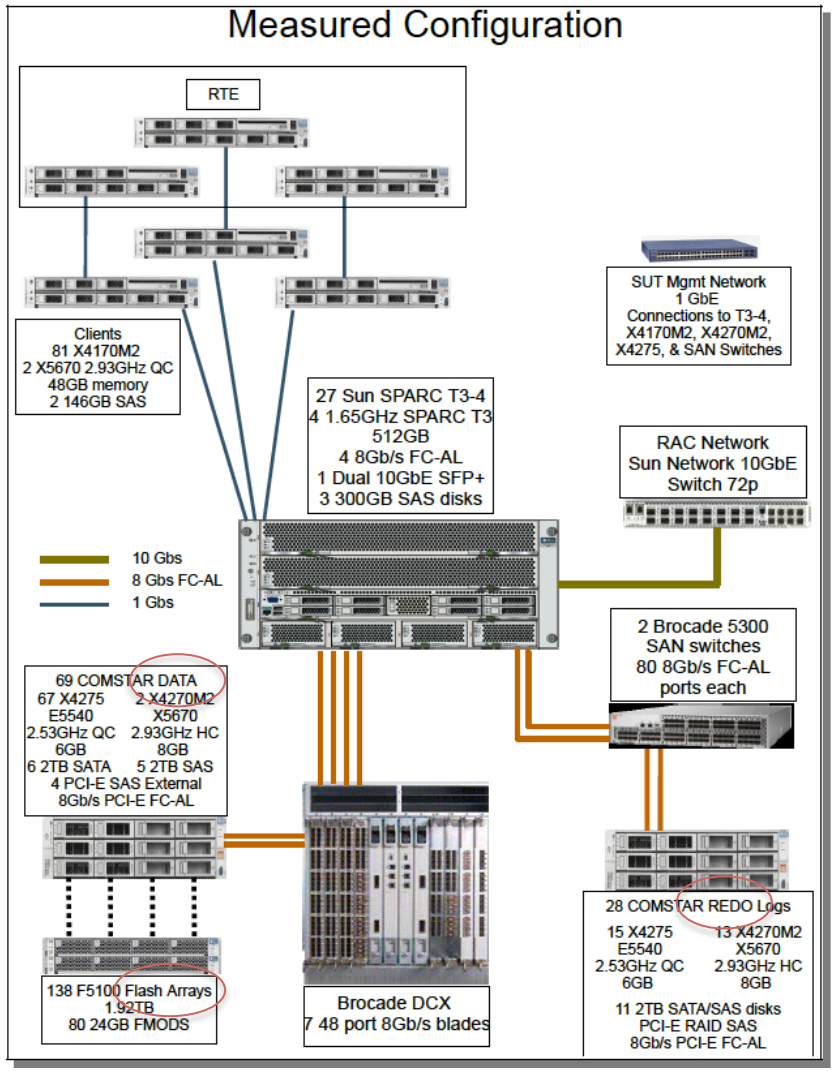
Using SANs and NAS, W. Preston, O' Reilly

SAN Topologies

- Point-to-point
- Bus
  - Synchronous (Parallel SCSI, ATA)
  - CSMA (Gb Ethernet)
- Arbitrated Loop (FC)
- Fabric (FC)

Case: TPC-C Top Performer (01/13)





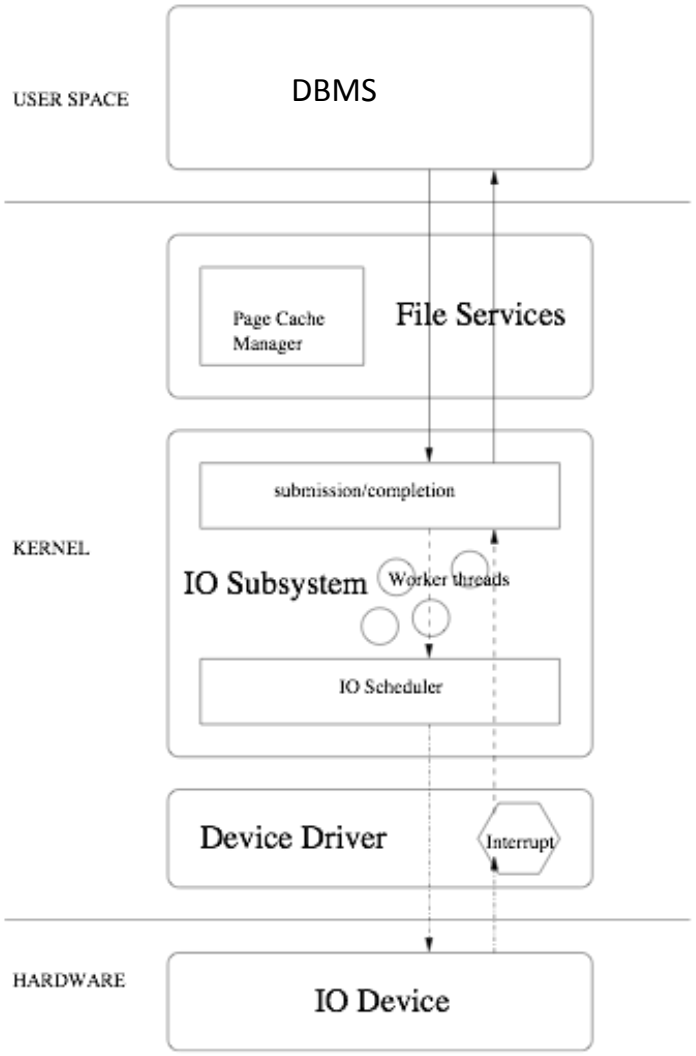
Redo Log Configuration

Total system cost	30,528,863 USD
Performance	30,249,688 tpmC
Total #processors	108
Total #cores	1728
Total storage	1,76 PB
Total #users	24,300,000

LOOK UP: [TPC-C OLTP Benchmark]

Source: http://www.tpc.org/tpcc/results/tpcc_result_detail.asp?id=110120201

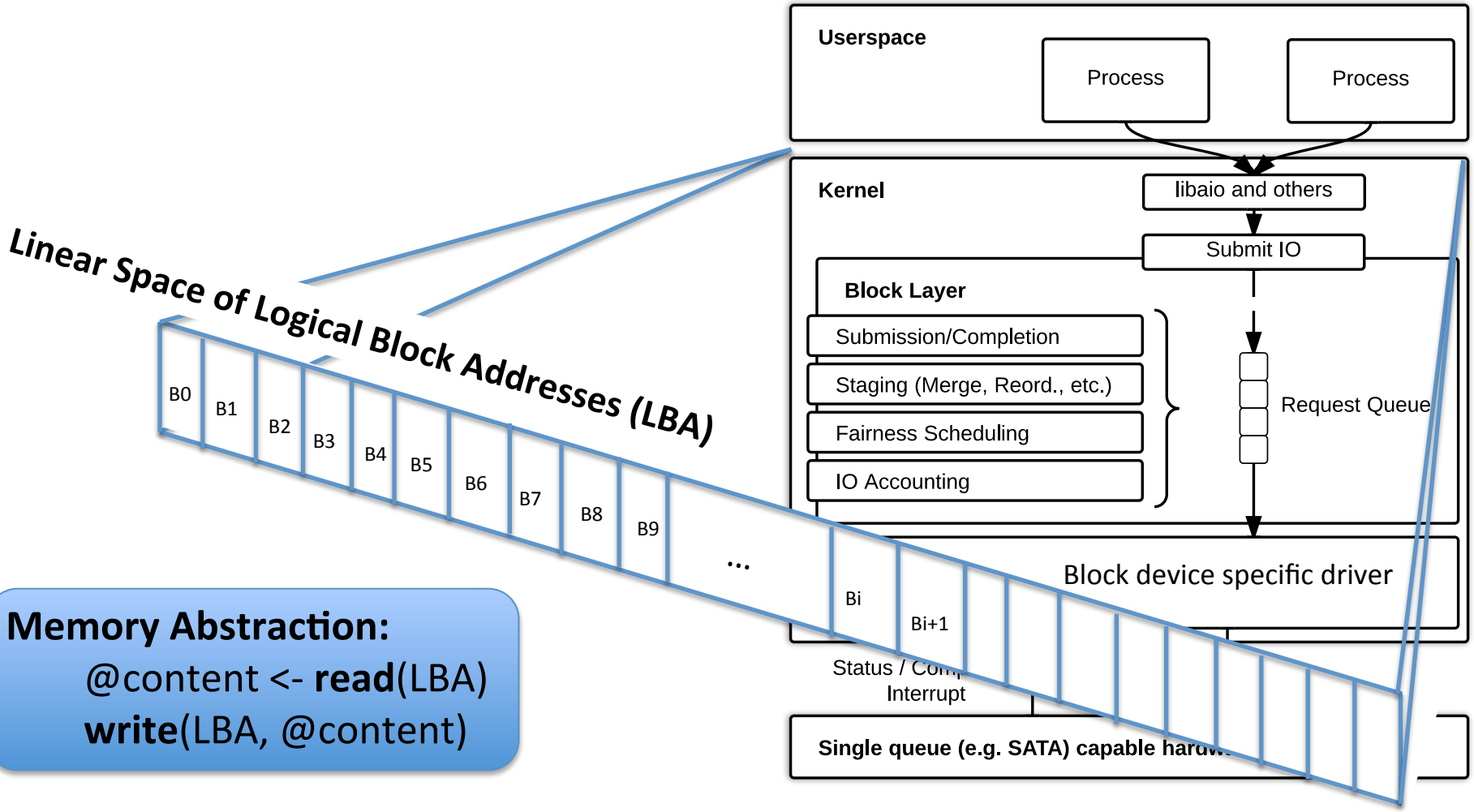
IO Stack



DBMS IOs:

- Asynchronous IO
- Direct IO

Block Device Interface



Memory Abstraction:

@content <- **read**(LBA)

write(LBA, @content)

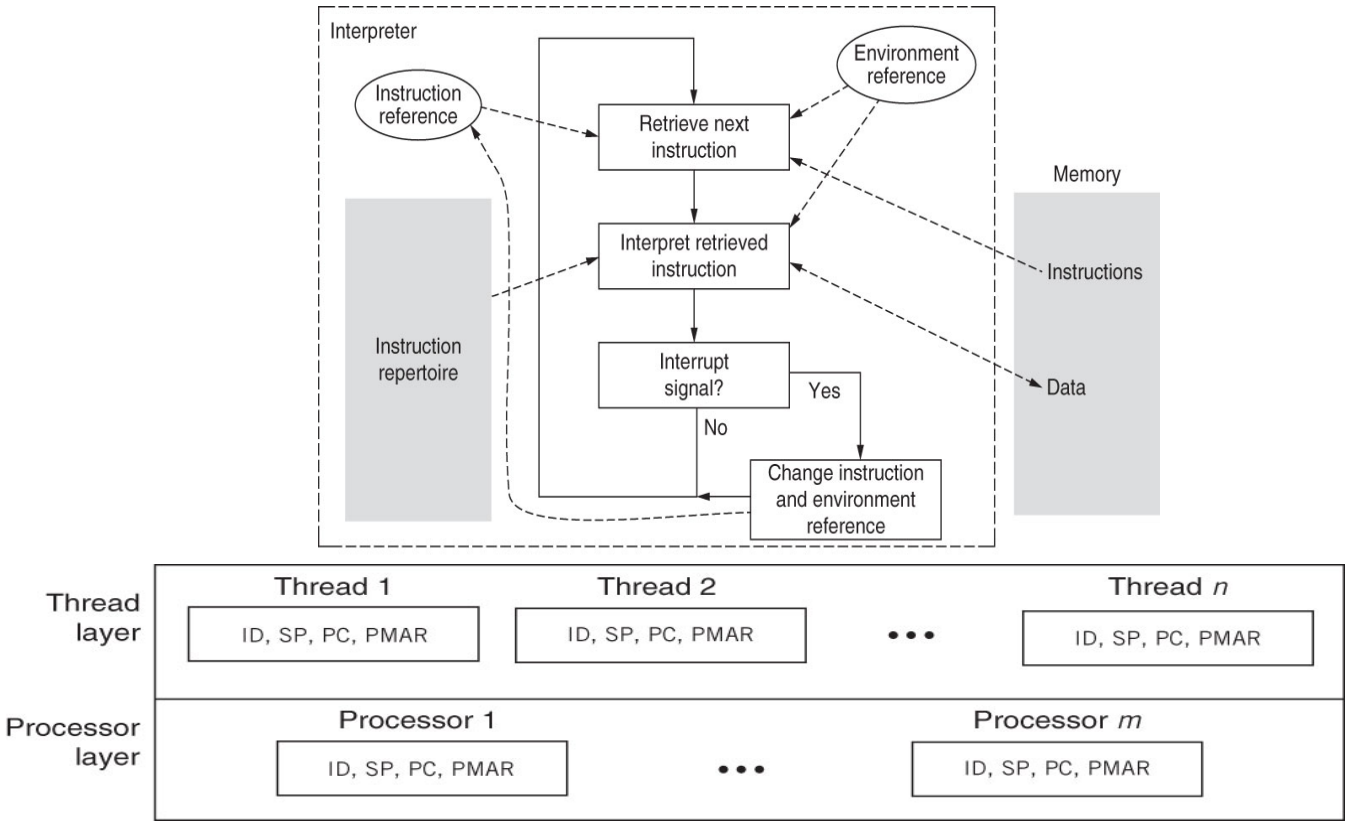
Performance Contract

- HDD
  - The block device abstraction hides a lot of complexity while providing a simple performance contract:
    - Sequential IOs are orders of magnitude faster than random IOs
    - Contiguity in the logical space favors sequential Ios
- SSD
  - No intrinsic performance contract
  - A few invariants:
    - No need to avoid random IOs
    - Applications should avoid writes smaller than a flash page
    - Applications should fill up the device IO queues (but not overflow them) so that the SSD can leverage its internal parallelism

Virtual Storage

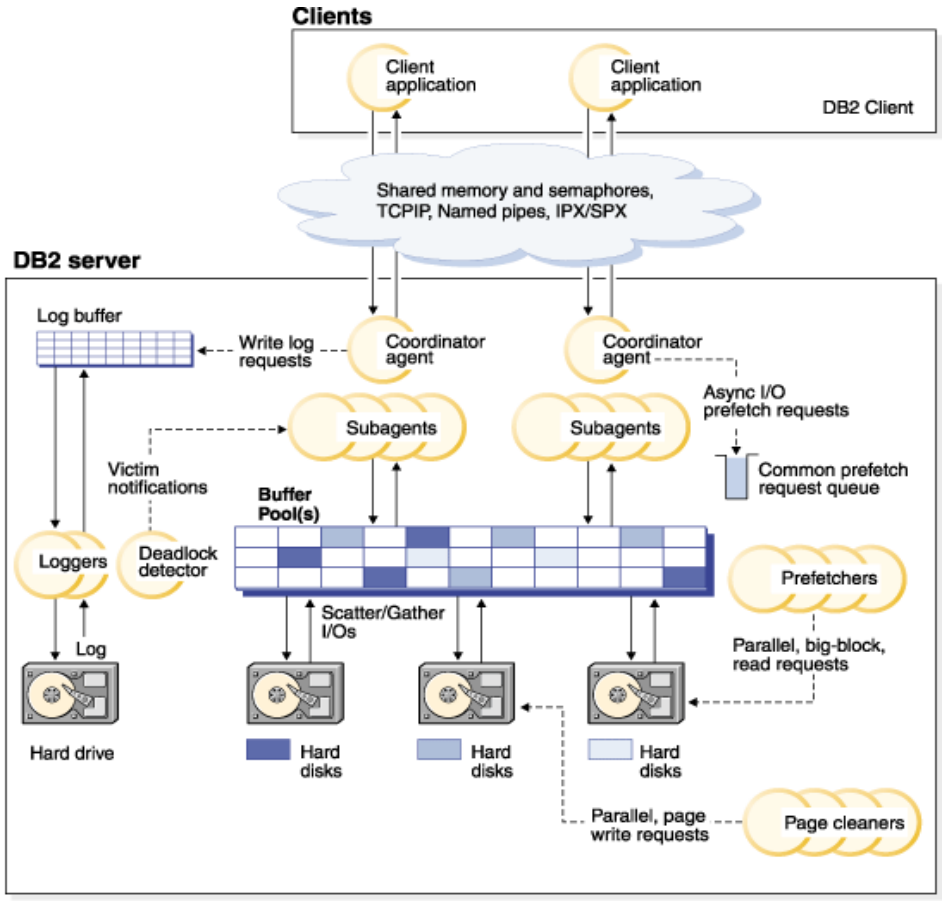
- No Host Caching.
  - The database system expects that the IO it submits are transferred to disk as directly as possible. It is thus critical to disable file system caching on the host for the IOs submitted by the virtual machine.
- Hardware supported IO Virtualization.
  - A range of modern CPUs incorporate components that speed up access to IO devices through direct memory access and interrupt remapping (i.e., Intel's VT-d and AMD's AMD-vi)). This should be activated.
- Statically allocated disk space
  - Static allocation avoids overhead when the database grows and favors contiguity in the logical address space (good for HDD).

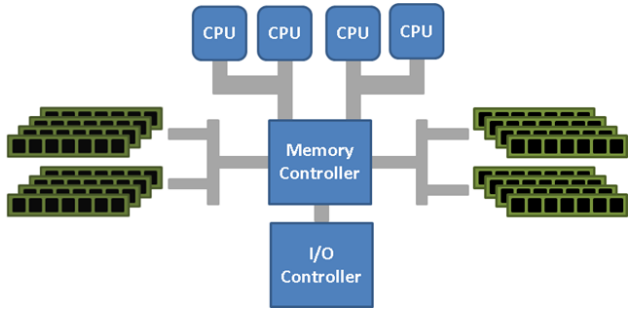
Processor Virtualization



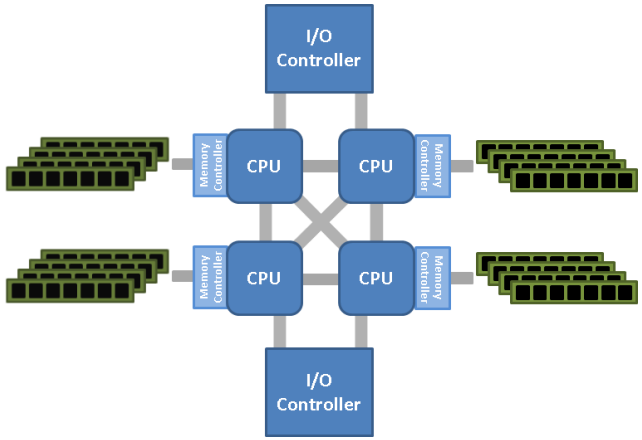
Source: Principles of Computer System Design, Kaashoek and Saltzer.

Dealing with Multi-Core





SMP: Symmetric Multiprocessor

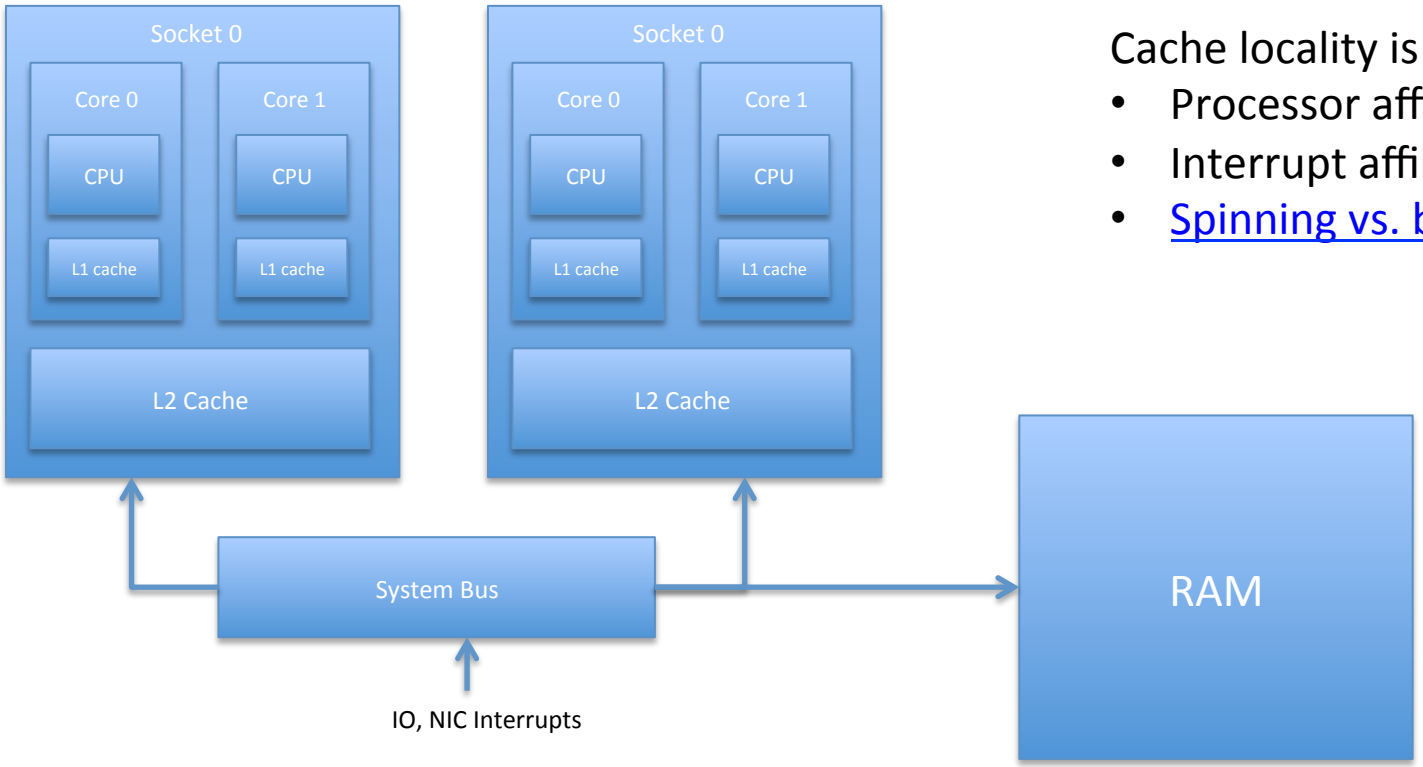


NUMA: Non-Uniform Memory Access

LOOK UP: Understanding NUMA

Source: http://pic.dhe.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.admin.perf.doc/doc/00003525.gif

Dealing with Multi-Core

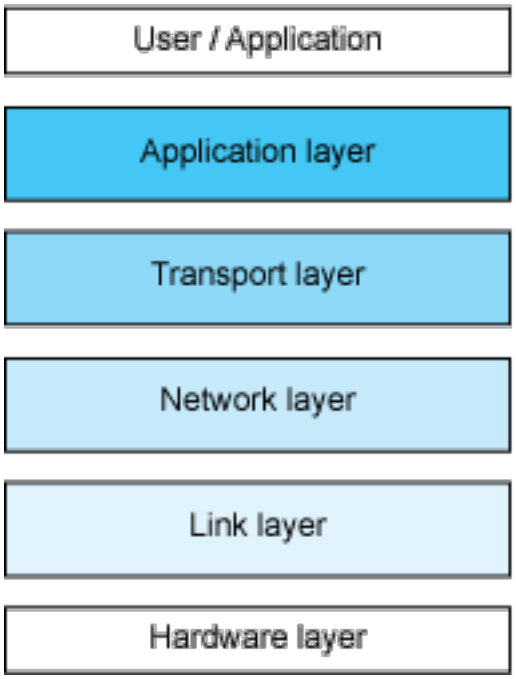


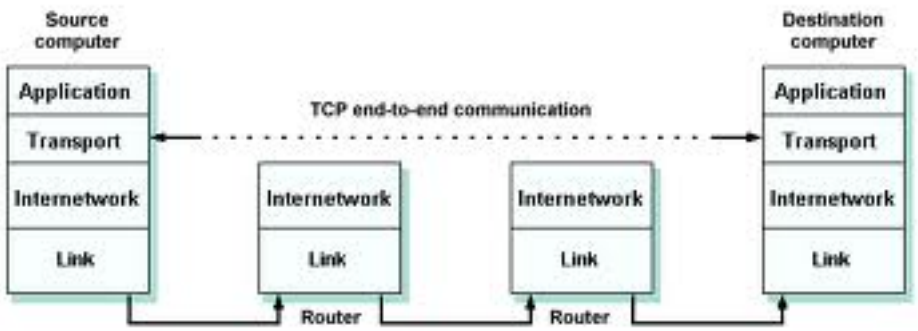
Cache locality is king:

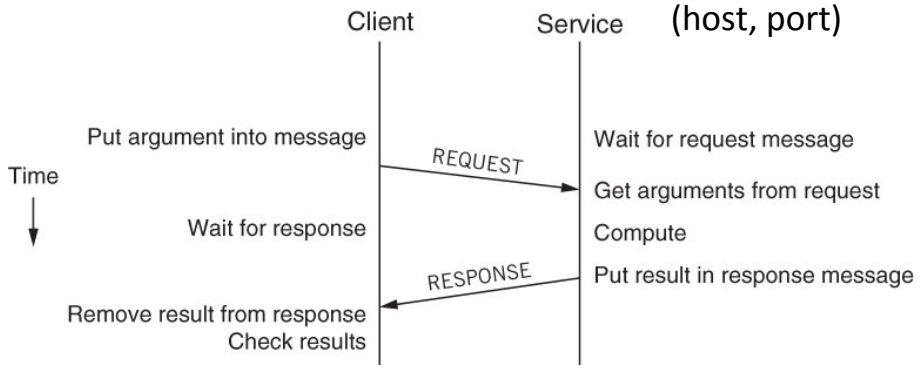
- Processor affinity
- Interrupt affinity
- Spinning vs. blocking

LOOK UP: SW for shared multi-core, Interrupts and IRQ tuning

Network Stack

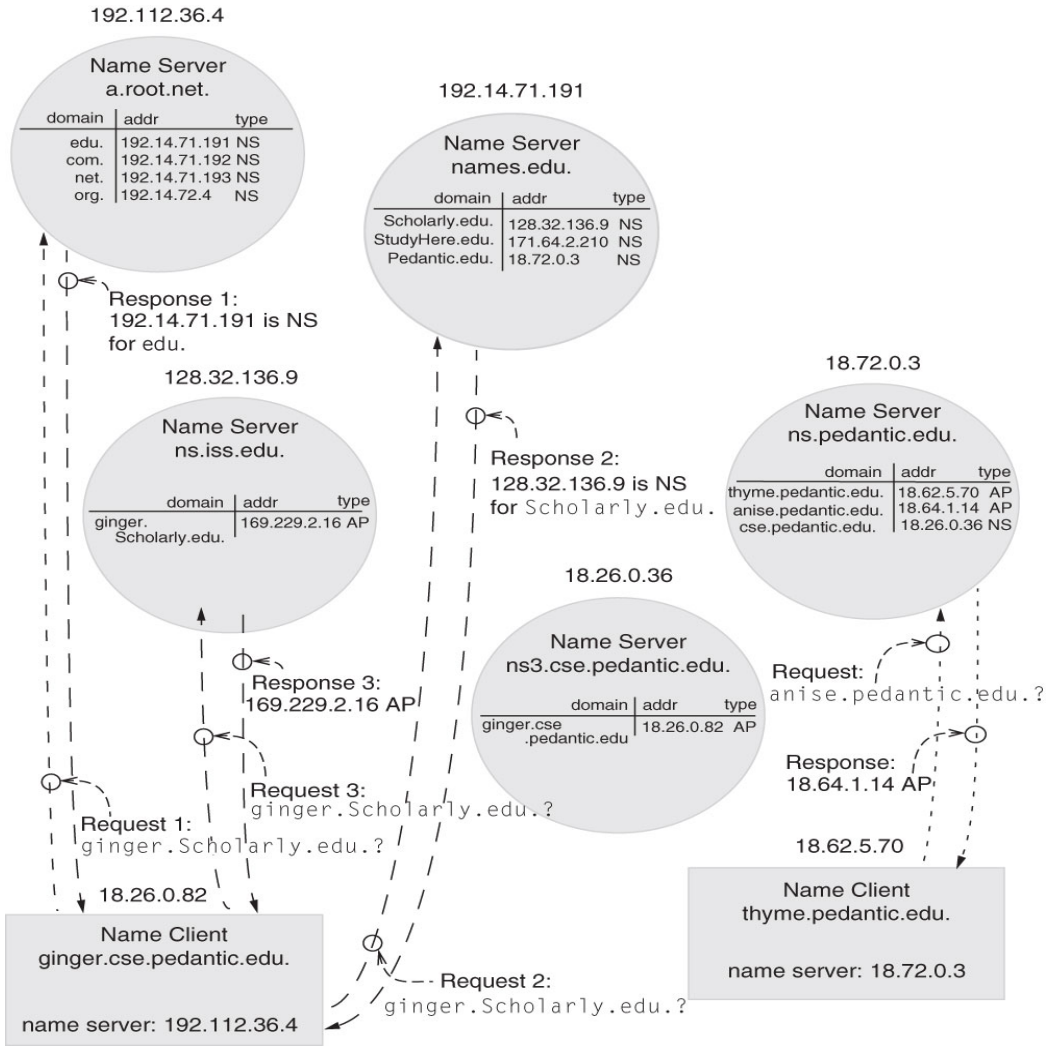






LOOK UP: [Linux Network Stack]

DNS Servers



Source: Principles of Computer System Design, Kaashoek and Saltzer.

Tuning the Guts

- RAID levels
- Controller cache
- Partitioning
- Priorities
- Number of DBMS threads
- Processor/interrupt affinity
- Throwing hardware at a problem

RAID Levels

- Log File
  - RAID 1 is appropriate
    - Fault tolerance with high write throughput. Writes are synchronous and sequential. No benefits in striping.
- Temporary Files
  - RAID 0 is appropriate
  - No fault tolerance. High throughput.
- Data and Index Files
  - RAID 5 is best suited for read intensive apps.
  - RAID 10 is best suited for write intensive apps.

Controller Cache

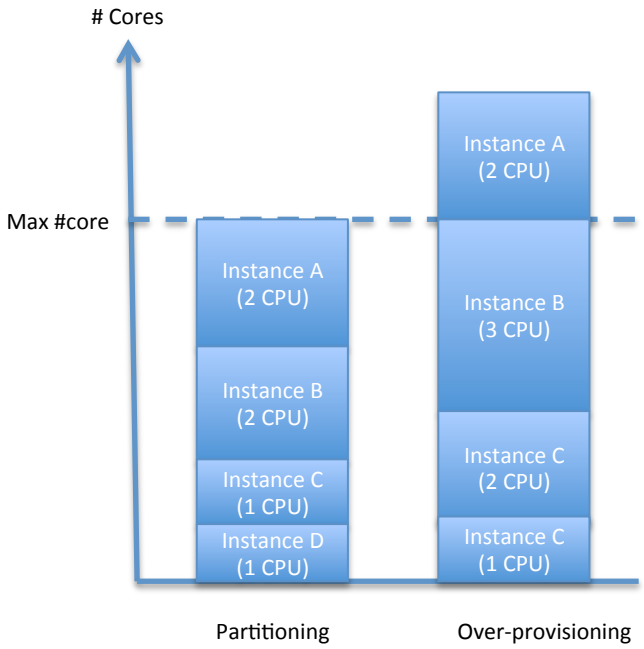
- Read-ahead:
  - Prefetching at the disk controller level.
  - No information on access pattern.
  - Not recommended.
- Write-back vs. write through:
  - Write back: transfer terminated as soon as data is written to cache.
    - Batteries to guarantee write back in case of power failure
    - Fast cache flushing is a priority
  - Write through: transfer terminated as soon as data is written to disk.

Partitioning

- There is parallelism (i) across servers, and (ii) within a server both at the CPU level and throughout the IO stack.
- To leverage this parallelism
  - Rely on multiple instances/multiple partitions per instance
    - A single database is split across several instances. Different partitions can be allocated to different CPUs (partition servers) / Disks (partition).
    - Problem#1: How to control overall resource usage across instances/ partitions?
      - Fix: static mapping vs. Instance caging
  - Control the number/priority of threads spawned by a DBMS instance
    - Problem#2: How to manage priorities?
    - Problem#3: How to map threads onto the available cores
      - Fix: processor/interrupt affinity

Instance Caging

- Allocating a number of CPU (core) or a percentage of the available IO bandwidth to a given DBMS Instance
- Two policies:
  - Partitioning: the total number of CPUs is partitioned across all instances
  - Over-provisioning: more than the total number of CPUs is allocated to all instances



LOOK UP: Instance Caging

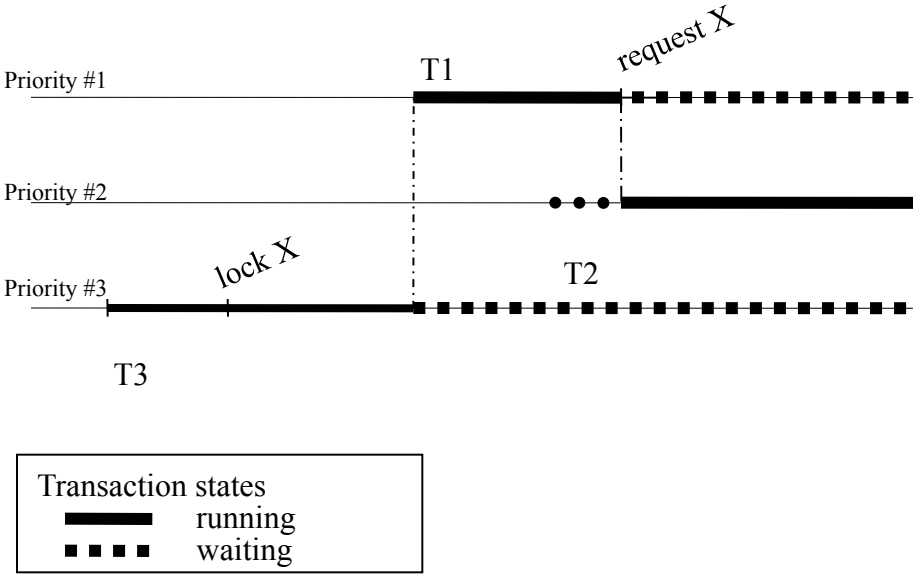
Number of DBMS Threads

- Given the DBMS process architecture
  - How many threads should be defined for
    - Query agents (max per query, max per instance)
      - Multiprogramming level (see tuning the writes and index tuning)
    - Log flusher
      - See tuning the writes
    - Page cleaners
      - See tuning the writes
    - Prefetcher
      - See index tuning
    - Deadlock detection
      - See lock tuning
- Fix the number of DBMS threads based on the number of cores available at HW/VM level
    - Partitioning vs. Over-provisioning
    - Provisioning for monitoring, back-up, expensive stored procedures/UDF

Priorities

- Mainframe DBMS allow to configure thread priority as well as IO priority
  - Threads associated to synchronous IOs (writes to the log, page cleaning under memory pressure, query agent reads) should have higher priorities than threads associated to asynchronous IOs (prefetching, page cleaner with no memory pressure) – see tuning the writes and index tuning
  - Synchronous IOs should have higher priority than asynchronous IOs.

The Priority Inversion Problem



Three transactions:
T1, T2, T3 in priority order (high to low)

- T3 obtains lock on x and is preempted
- T1 blocks on x lock, so is descheduled
- T2 does not access x and runs for a long time

Net effect: T1 waits for T2

Solution:

- No thread priority
- Priority inheritance

Processor/Interrupt Affinity

- Mapping of thread context or interrupt to a given core
  - Allows cache line sharing between application threads or between application thread and interrupt (or even RAM sharing in NUMA)
  - Avoid dispatch of all IO interrupts to core 0 (which then dispatches software interrupts to the other cores)
  - Should be combined with VM options
  - Specially important in NUMA context
  - Affinity policy set at OS level or DBMS level?

LOOK UP: Linux CPU tuning

Processor/Interrupt Affinity

- IOs should complete on the core that issued them
  - I/O affinity in SQL server
  - Log writers distributed across all NUMA nodes
- Locking of a shared data structure across cores, and specially across NUMA nodes
  - Avoid multiprogramming for query agents that modify data
  - Query agents should be on the same NUMA node
- DBMS have pre-set NUMA affinity policies

LOOK UP: Oracle and NUMA, SQL Server and NUMA

Transferring large files (with TCP)

- With the advent of compute cloud, it is often necessary to transfer large files over the network when loading a database.
- To speed up the transfer of large files:
  - Increase the size of the TCP buffers
  - Increase the socket buffer size (Linux)
  - Set up TCP large windows (and timestamp)
  - Rely on selective acks

LOOK UP: TCP Tuning

Throwing hardware at a problem

- More Memory
  - Increase buffer size without increasing paging
- More Disks
  - Log on separate disk
  - Mirror frequently read file
  - Partition large files
- More Processors
  - Off-load non-database applications onto other CPUs
  - Off-load data mining applications to old database copy
  - Increase throughput to shared data
    - Shared memory or shared disk architecture