ENTERPRISE APPLICATIONS THAT SCALE AND PERFORM

Enterprise Architectue Conference
Boston
August 30, 1998
1:00 P.M. - 5:00 P.M.

David McGoveran
Alternative Technologies
13150 Highway 9, Suite 123
Boulder Creek, CA 95006
Telephone: 831/338-4621
www.AlternativeTech.com

PLEASE FILL OUT YOUR EVALUATIONS...

Thank you!

OVERVIEW

- Distributed Enterprise Applications
  - WHAT ARE DISTRIBUTED ENTERPRISE APPLICATIONS?
  - HOW DO THE KEY ARCHITECTURES DIFFER?
- Distributed Architectures
  - WHAT ARE THEY?
  - KEY ARCHITECTURAL ISSUES
  - 2-TIER VS. 3-TIER
  - APPLICATION SERVERS AND TP MONITORS
- Scalability
  - WHAT IS IT?
- Performance

OVERVIEW

- Transactions: Concepts, Design, and Management
  - WHY TRANSACTIONS MATTER
- Principles of Scalable Design
  - WHAT MAKES ENTERPRISE APPLICATIONS FAIL?
  - WHY ARCHITECTURES FAIL
  - KEY CLIENT DESIGN PRINCIPLES
  - KEY DATABASE DESIGN PRINCIPLES
- Challenge the speaker
  - Q & A
  - AUDIENCE CONCERNS

DISTRIBUTED ENTERPRISE APPLICATIONS

DISTRIBUTED ENTERPRISE APPLICATIONS

- **Definition**

  DISTRIBUTED: DIVIDED AND SHARED, PLACED AT DIFFERENT POINTS

  ENTERPRISE: A BUSINESS ACTIVITY OR INITIATIVE

  APPLICATION: A PROGRAM APPLIED TO SOLVE A PARTICULAR PROBLEM

  or "A DIVIDED AND SHARED PROGRAM, PLACED AT DIFFERENT POINTS AND APPLIED TO SOLVE A PARTICULAR PROBLEM ASSOCIATED WITH THE BUSINESS ACTIVITY"

- **Enterprise is Understood to Imply:**
  - ASSOCIATED WITH THE MISSION (PERHAPS MISSION CRITICAL)
  - ROBUST
  - AVAILABLE
  - MANAGEABLE

DISTRIBUTED ENTERPRISE APPLICATIONS
WHY?

- **Why Enterprise?**
  - I.T. MUST JUSTIFY THE BUSINESS VALUE OF PROJECTS
  - ENTERPRISE APPLICATIONS HAVE BUSINESS VALUE (BY DEF.)
  - ENTERPRISE APPLICATIONS MUST PERFORM AND SCALE
  - ACCESSIBILTY HAS BECOME CRUCIAL
- **Why Distributed?**
  - BUSINESS REQUIRMENTS ARE CHANGING RAPIDLY
  - TECHNOLOGY IS CHANGING RAPIDLY
  - ENTERPRISE APPLICATIONS OFTEN HAVE HIGHLY VARIABLE LOAD
  - DISTRIBUTED APPLICATIONS ARE FLEXIBLE AND SCALABLE

DISTRIBUTED ENTERPRISE APPLICATIONS

WHY?

- Distribution of Processing Load
- Distribution of Access
- Better Off-the-shelf Tools
  - DESIGN
  - DEVELOPMENT
  - END-USER REPORTING AND QUERY
- Removable of I.T. Bottlenecks
- Independent Hardware Upgrades
- Better Load Balancing

DISTRIBUTED ENTERPRISE APPLICATIONS

A LITTLE HISTORICAL PERSPECTIVE

- **Mainframe Applications**
  - MONOLITHIC WITH TERMINAL ACCESS
  - ROBUST, BUT SENSITIVE ENVIRONMENT
  - UNRESPONSIVE TO BUSINESS CHANGE
  - APPLICATION BACKLOG
  - GOOD PERFORMANCE BUT DID NOT SCALE
  - INTRODUCED SYSTEM SERVICES
- **Remote Access**
  - SLOW DIAL UP, REMOTE JOB ENTRY
  - TERMINAL SERVERS IMPROVED CONNECTION MULTIPLEXING AND POOLING

DISTRIBUTED ENTERPRISE APPLICATIONS
A LITTLE HISTORICAL PERSPECTIVE

- **Minicomputers and (D)ARPANET**
  - GREATER EMPHASIS ON SHARED SERVICES
  - DEDICATED MINICOMPUTERS BECAME “SERVERS”
  - EARLY MESSAGE-BASED COMPUTING (ETHERNET)
- **Early Clusters**
  - INTRODUCED DISTRIBUTED LOCK MANAGEMENT
  - ADDED AVAILABILITY, SIMPLY FAULT TOLERANCE, AND SOME SCALABILITY
  - NETWORK BASED TERMINAL ACCESS

DISTRIBUTED ENTERPRISE APPLICATIONS

A LITTLE HISTORICAL PERSPECTIVE

- **Client/Server**
  - **SIMPLE PARTITIONED FUNCTIONAL LOAD MODEL**
  - **MAINTAINED CENTRALIZED CONTROL**
  - **INITIALLY SERIAL / PARALLEL DIRECT ACCESS, NETWORK**
  - **FOCUS ON DBMS SERVER, PRINT AND NETWORK SERVERS CAME LATER**
  - **IMPROVED SCALABILITY AND PERFORMANCE**
  - **MOST IMPLEMENTATIONS FAILED TO MEET EXPECTATIONS**
  - **WIDESPREAD EXPERIENCE WITH DISTRIBUTED DESIGN**
  - **SERVER OFTEN BECAME A BOTTLENECK**

DISTRIBUTED ENTERPRISE APPLICATIONS

A LITTLE HISTORICAL PERSPECTIVE

- **Cooperative Processing and Peer-to-Peer**
  - FULL DISTRIBUTION AND FUNCTION SHARING
  - REQUIRED DISTRIBUTED CONTROL
  - TOO COMPLICATED TO DESIGN, DEVELOP, AND MANAGE
  - PEER-TO-PEER APPLICATIONS RARELY SUCCEEDED
- **Multi-tier Client/Server**
  - INTRODUCED TP MONITORS
    - » *CONNECTION OVERHEAD, DISTRIBUTED TRANSACTIONS*
  - INTRODUCED APPLICATION SERVERS
    - » *IMPROVED DEPLOYMENT PROBLEM*
  - MORE COMPLEX APPLICATION PARTITIONING

DISTRIBUTED ENTERPRISE APPLICATIONS
A LITTLE HISTORICAL PERSPECTIVE

- **Network Computing and "Thin Client"**
  - EVOLUTION OF DISTRIBUTED PRESENTATION AND APPLICATION SERVERS
  - INTEGRATION WITH OBJECT ORIENTED PROGRAMMING
  - REQUIRES INTEROPERABILITY STANDARDS
- **The Web and The Emergence of the Extraprise**
  - DISTRIBUTION MOVES BEYOND THE ENTERPRISE
  - DRIVEN BY BUSINESS RAPID CHANGE
  - ENABLED BY PORTABILITY STANDARDS
    - » *HTML AND JAVA*
  - SCALABILITY AND PERFORMANCE PROBLEMS ABOUND

DISTRIBUTION ARCHITECTURES

DISTRIBUTED ARCHITECTURES

- Distributed Architectures *Permit* Distributed Deployment
- Distribution Requires:
  - EFFICIENCY OF COMMUNICATIONS
  - MODULARITY OF COMPONENTS
  - PROPER FUNCTIONAL PARTITIONING
- Key Decisions
  - FAT VS. THIN CLIENT
  - APPLICATION AND MIDDLEWARE SERVERS
  - TP MONITORS / TRANSACTION SERVERS
  - APPLICATION PARTITIONING
  - NUMBER OF TIERS

THE PURPOSE OF ARCHITECTURE

(Technical) Architecture Is A Set of Rules and Protocols

- **Rules for Functional Partitioning**
  - **WHAT GENERATES REQUESTS**
  - **WHAT SERVICES REQUESTS**
  - **DISTRIBUTABLE COMPONENT GRANULARITY**
- **Rules Mandating Uniform Component Properties**
- **Interoperation Protocols**
  - **COMPONENT INTERFACES**
  - **COMMUNICATION**
- **Hardware Utilization**

ARCHITECTURE ISSUES

- **Synchronization:**
  - BLOCKING VS. NON-BLOCKING
- **Request Granularity:**
  - INTERFACE-DRIVEN VS. BUSINESS FUNCTION DRIVEN
- **Event Management**
  - TIGHT VS. WEAK COUPLING TO THE USER INTERFACE
- **Processing:**
  - PROCEDURAL VS. NON-PROCEDURAL
- **Distribution:**
  - SINGLE PLATFORM VS. MULTI-PLATFORM DEPLOYMENT

Architecture determines distributed functional performance and scalability!

SERVER ARCHITECTURE

- **Task Granularity**
  - PROCESS VS. THREADS
  - SINGLE VS. MULTI-THREADED
- **Scheduling and Optimization**
  - PREEMPTIVE VS. NON-PREEMPTIVE
  - TASK PRIORITIZATION
  - LOAD BALANCING
- **State Management**

Server architecture determines distributed request performance and scalability!

PLATFORM ARCHITECTURE

- **Operating System Characteristics**
  - **TASK MANAGEMENT**
  - **RESOURCE MANAGEMENT**
- **Hardware Characteristics**
  - **UNIPROCESSER, SMP, CLUSTER, SHARED NOTHING**
    - » ***SPEED***
  - **RESOURCES (MEMORY, DISK SPACE, ETC.)**
- **Single vs. Multiple Platforms**

Platform architecture determines distributed system performance and scalability!

SINGLE PLATFORM ARCHITECTURES

- **Presentation Logic and Application Software Reside on the Same Hardware**
- **Communicate Through:**
  - NETWORK SERVICES (LOOP-BACK)
  - OPERATING SYSTEM FACILITIES: SHARED MEMORY, PIPES, MAILBOXES, ETC.
- **Presentation Can Be Thin Client**
  - CHEAP

SINGLE PLATFORM ARCHITECTURES

KEY STRENGTHS

- **Faster Response Time Due to Decreased Network Costs**
- **Simplified System Management**
- **Scalable to Multiple Platform Architectures**
  - **IF GOOD DESIGN PRACTICES ARE FOLLOWED!**
- **Faster Debugging**
  - **A GOOD WAY TO DEVELOP, PROTOTYPE, AND TEST**

SINGLE PLATFORM ARCHITECTURES
KEY WEAKNESSES

- May Encourage Non-distributed Design
- Platform May Have to Be Very Powerful
- User Interface Management Not Distributed
- User Context Management Not Distributed
- Difficult to Tune
  - DIFFERENT GOALS FOR SERVER PORTION AND CLIENT PORTION INTERFERE WITH EACH OTHER

MULTIPLE PLATFORM ARCHITECTURES

- **Client and server software *can* reside on different hardware**
- **Network Communication**
  - **LAN, WAN, DEDICATED LINE, SATELLITE, RF, ETC.**
  - **ASYNC**
- **Distribution Protocols**
  - **COM**
  - **CORBA**
- **Can be multiple clients, multiple servers, and multi-tier**

MULTIPLE PLATFORM ARCHITECTURES

KEY STRENGTHS

- **If You Don't Do It Right, It Doesn't Work!**
  - HIGHLY VISIBLE ERRORS ENCOURAGE BETTER DESIGN THAN SINGLE PLATFORM
- **Load Balancing Is Possible**
  - BETWEEN CLIENT AND SERVER
  - ACROSS MULTIPLE SERVERS
- **Better Server Environment Tuning Possible**
  - ASSUMES DEDICATED TASK SERVER

MULTIPLE PLATFORM ARCHITECTURES
KEY WEAKNESSES

- IF YOU DON'T DO IT RIGHT, IT DOESN'T WORK!
  - DESIGN ERRORS CAN BE COSTLY
- Higher Communications Overhead
- State Management Is Required Across Platforms
- Distributed System Management Is Required

TWO-TIER

- **Draw Your Architecture in Tiers**
- **"Classic" Client/Server Is Physical Two-tier**
  - **SIMPLIFIED SYSTEM MANAGEMENT**
  - **SIMPLIFIED APPLICATION DESIGN**
  - **SERVER *MIGHT* BECOME A BOTTLENECK**
    - » ***SINGLE SERVER SUPPORTS VERTICAL SCALABILITY ONLY***
    - » ***MULTIPLE SERVERS SUPPORT BOTH HORIZONTAL AND VERTICAL SCALABILITY***
- **Viewed Logically, Two-tier Can Be M:M**
  - **TODAY'S SYSTEMS DON'T SUPPORT TRANSPARENT HORIZONTAL SERVER SCALABILITY**

MULTI-TIER

- Middle Tier Can Be TP Monitors or Application Servers
- DBMS Servers Can Be Multi-Tier Hierarchies
  - MAY USE DISTRIBUTED DBMS OR REPLICATION
- Application Servers
  - CAN BE ANY APPLICATION OR FUNCTIONAL CODE
  - NEED NOT BE COMPLEX
  - NEED NOT BE SPECIFICALLY DESIGNED AS A SERVICE
  - CAN BE SINGLE OR MULTI-THREADED
  - CAN BE SINGLE OR MULTIPLE INSTANCE

TP MONITORS
ADVANTAGES

- Stable Queues (Tasks vs. Messages)
- Both Database and Non-database Transactions
- Task Scheduling, Dispatch, and Distribution
- Prioritization
- Resource Sharing
- Potentially High Levels of Recovery/Availability
  - INFLIGHT RECOVERY

TP MONITORS
DISADVANTAGES

- Requires Programmatic Control
- Complex Environment
- Not Database Integrated
  - DATABASE SCHEDULING
  - OPTIMIZATION
  - 2PC WHEN YOU DON'T NEED IT
  - SUBTRANSACTIONS CAN LIVELOCK
- Does Not Preserve Database User Identity

SERVER ARCHITECTURES

Server Usage

- **Multi-user vs. single user clients**
- **Multi-transaction clients**
- **Multi-session clients**
- **Multi-connection clients**
- **Multi-server clients**
  - **SERIAL**
  - **PARALLEL (SYNCHRONOUS SERVER USE)**
  - **CONCURRENT (ASYNCHRONOUS SERVER USE)**

SERVER ARCHITECTURES

Application Architecture

- Stateless vs. state-dependent
- Serial client/server
- Synchronous client/server multi-tasking
- Asynchronous client/server multi-processing
- Single tasking vs. multi-tasking clients
  - MULTI-THREADING

TYPES OF SERVER ARCHITECTURES

- **Local Server**
  - SINGLE-USER ON THE CLIENT
  - CACHING RELATIVELY STATIC OBJECTS
  - EASY DEVELOPMENT AND ADMINISTRATION AT THE EXPENSE OF LIMITED SCALABILITY
- **Remote Server**
  - SINGLE SITE TRANSACTIONS BY DEFINITION
  - LIMITED APPLICATION MIX
  - IMPROVED SYSTEM SCALABILITY FOR THE PRICE OF DISTRIBUTED DESIGN

TYPES OF SERVER ARCHITECTURES

- **Multiple Remote Servers**
  - SINGLE-SITE READ AND WRITE TRANSACTIONS
  - SEGMENTABLE BY TRANSACTION OR APPLICATION OR USER REQUIRED
  - MODERATE SCALABILITY AT DEVELOPMENT, MAINTENANCE, AND ADMINISTRATION EXPENSE
- **Distributed Transaction Server**
  - MULTI-SITE READ AND WRITE TRANSACTIONS
  - SEGMENTABLE BY TRANSACTION OR APPLICATION OR USER DESIRABLE TO MINIMIZE OVERHEAD
  - GOOD SCALABILITY AT DEVELOPMENT, MAINTENANCE, AND ADMINISTRATION EXPENSE

TYPES OF SERVER ARCHITECTURES

- **Distributed Servers**
  - **COMPLEX TRANSACTIONS**
  - **SHARED-NOTHING (LARGE DATABASES)**
    - ***FUNCTION SHIPPING AMONG SERVER PEERS***
  - **TWO-PHASE COMMIT OVERHEAD (*OR ITS EQUIVALENT)* REQUIRED**
  - **HIGH SCALABILITY AT THE EXPENSE OF ADDITIONAL RESOURCES AND DESIGN SOPHISTICATION**
  - **PROVIDES THE BEST INDEPENDENCE BETWEEN APPLICATION CODE AND SERVICE LOCATION**

SCALABILITY

SCALABILITY

- **Formal Definition**
  - SCALEUP VS. SPEEDUP
  - OVER A RANGE
  - WITH RESPECT TO A RESOURCE
  - FOR A PARTICULAR WORKLOAD
    - » *NUMBER OF USERS, DB SIZE, TRANSACTION RATE, TRANSACTION COMPLEXITY*
- **Scale up**

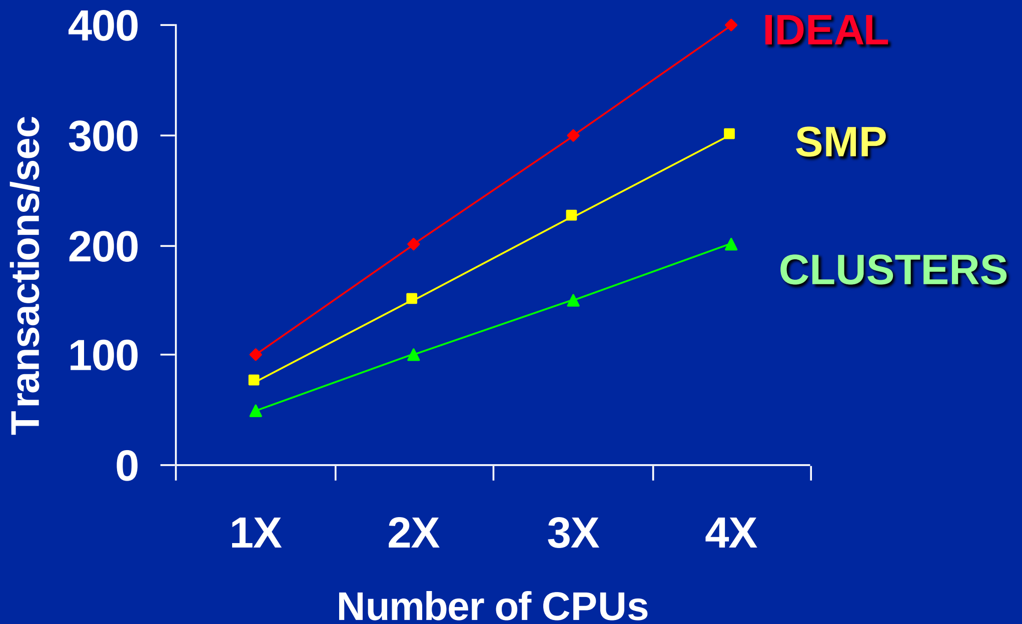
  MORE RESOURCES = SAME PERFORMANCE FOR BIGGER WORKLOAD
- **Speed up**

  MORE RESOURCES = BETTER PERFORMANCE FOR SAME WORKLOAD

SCALEUP OR SPEEDUP
NOT PROVABLE BY EXAMPLE

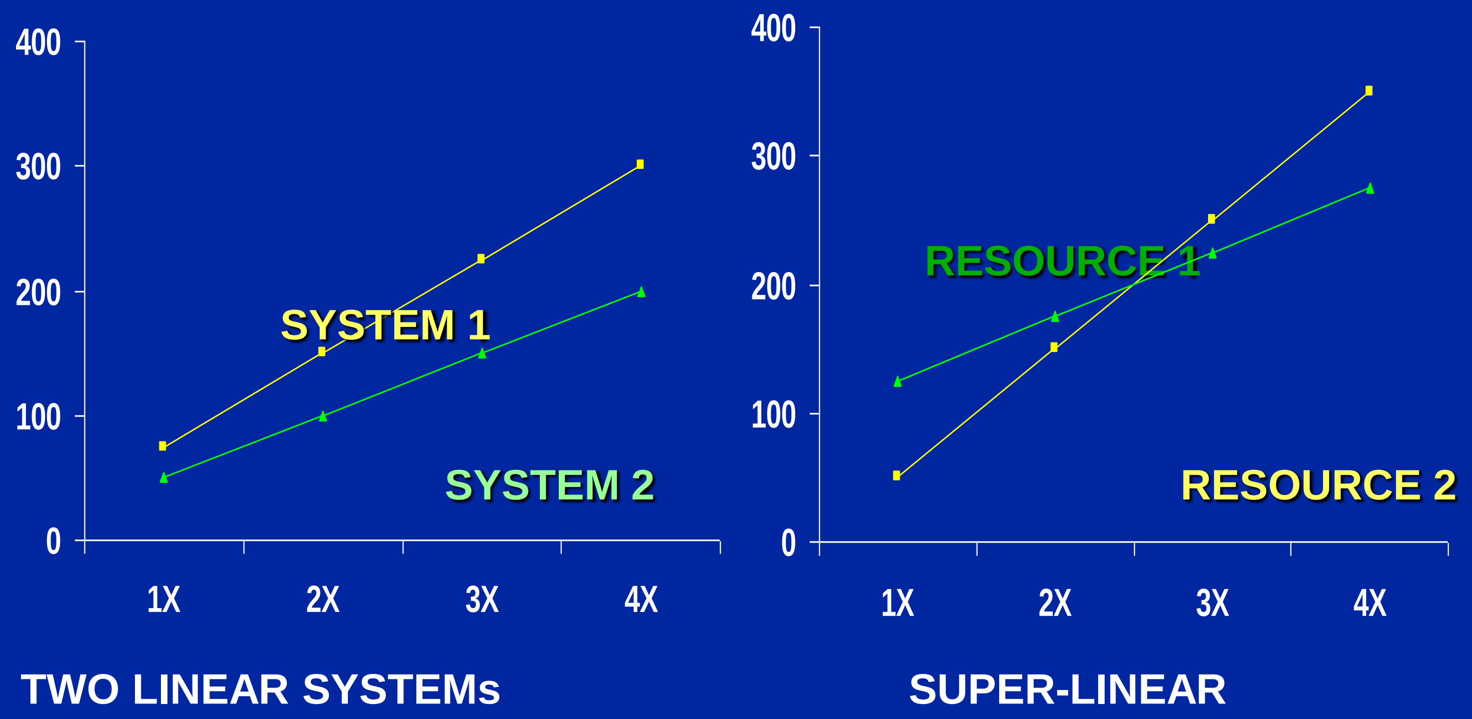
SCALEUP AND SPEEDUP ARE:

- PLATFORM AND APPLICATION SPECIFIC
- STRONGLY AFFECTED BY TRANSACTION AND DB DESIGN



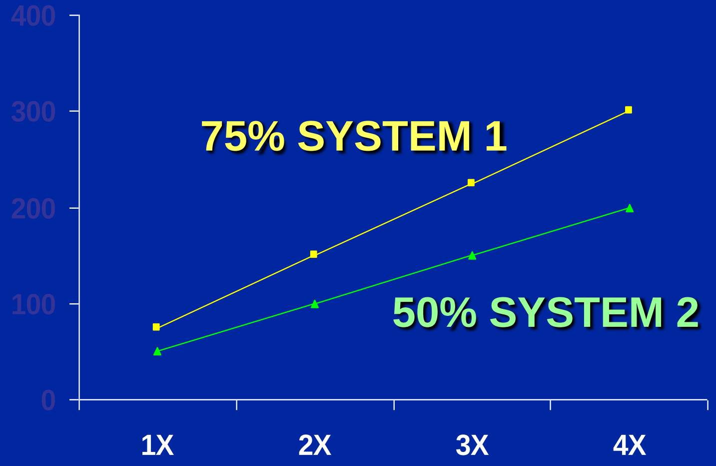
Transaction rate versus CPUs

SCALEUP AND SPEEDUP
LINEARITY AND SUPER-LINEAR

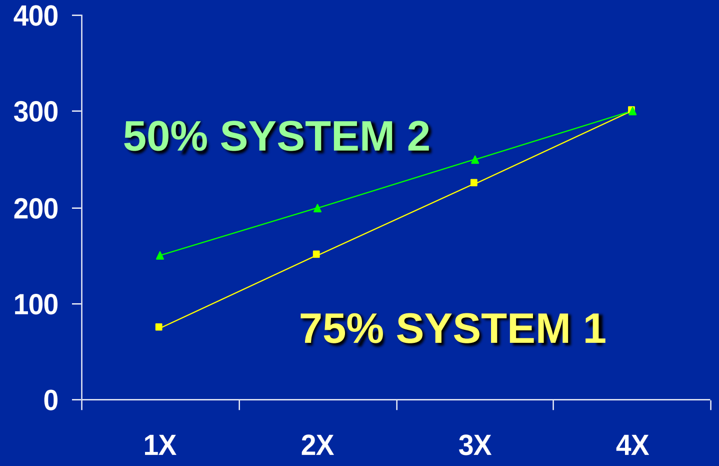


SCALEUP AND SPEEDUP
PERCENT NOT A METRIC OF VALUE

WHAT DOES PERCENT SCALABILITY MEAN?



UNLABELED PERFORMANCE



LABELED PERFORMANCE

SOME TYPES OF SCALABILITY

- **Administrative scalability**
- **Platform scalability**
- **Processor scalability**
- **Horizontal scalability**

  MORE BOXES APPROACH
- **Vertical scalability**

  BIGGER BOXES APPROACH
- **Functional scalability - extensibility**
- **Hardware vs. software**

WHAT AFFECTS SCALABILITY?

- **Efficiency of Resource Usage**
  - DETERMINES BASELINE AND INCREMENTAL PERFORMANCE
  - DYNAMIC OPTIMIZATION
- **Parallelism**
  - IMPROVES RESOURCE USAGE
- **State Management**
  - CLIENT (COOKIE)
  - MIDDLEWARE
  - APPLICATION SERVER
  - STATE SERVER
- **Load Balancing and Scheduling**
  - ROUND ROBIN, FIFO, LEAST LOAD

WHAT ENABLES SCALABILITY?

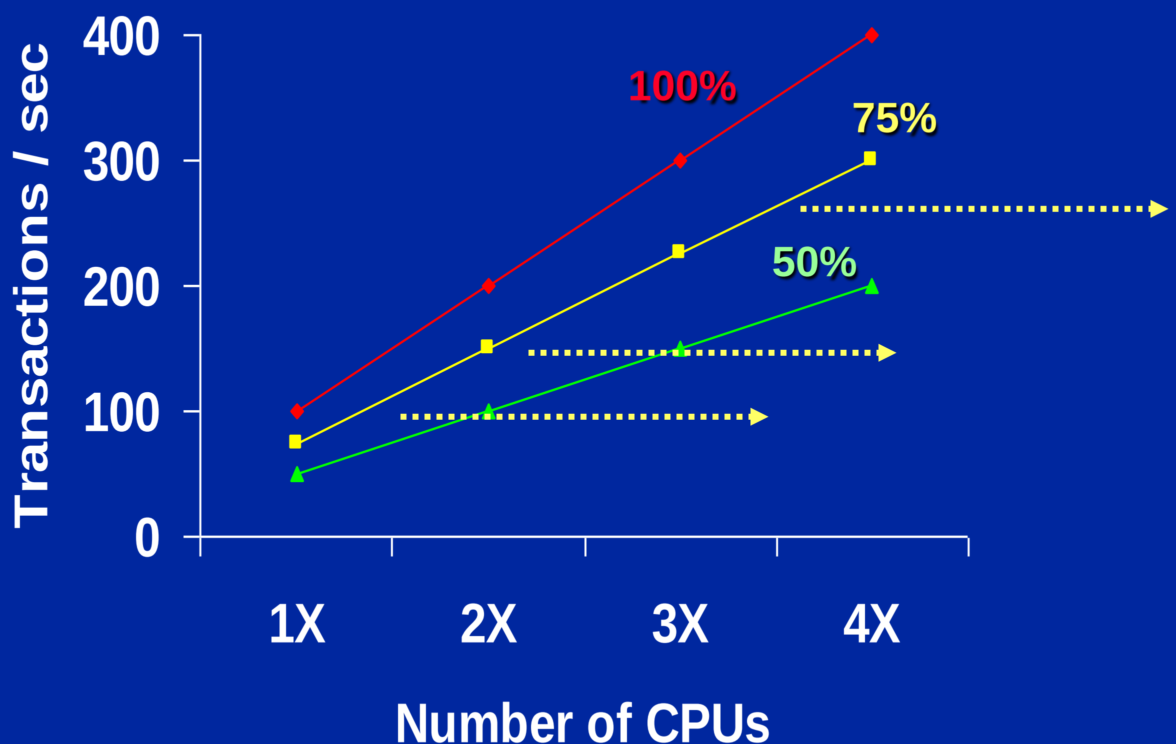
- **Application Tool Flexibility**
- **Designing for Multi-user Systems**
- **Context-free Applications and Transactions**
  - **NON-CONVERSATIONAL**
  - **STATELESS SESSIONS**
- **Capacity**
- **Configuration Control**

Choosing the right architecture(s) for the job!

PLATFORM SCALABILITY
CLUSTERING

- **Clustering Primarily Provides, and Is Used For, High Availability**
  - GENERALLY NOT A SCALABILITY SOLUTION
- **Great Care Is Required to Obtain Even Moderate Scaleup or Speedup**
  - CROSS-NODE CLUSTER RESOURCE USAGES IS NON-LINEAR
- **Designed More Like a Federation of Loosely Coupled Systems**
- **Costs Can Be High**
  - DESIGN TIME, ADDITIONAL ADMINISTRATION, POSSIBLY CODING, AND LOCK OR CACHE COHERENCE MANAGEMENT

PROCESSOR SCALABILITY
NOT AN ABSOLUTE ATTRIBUTE



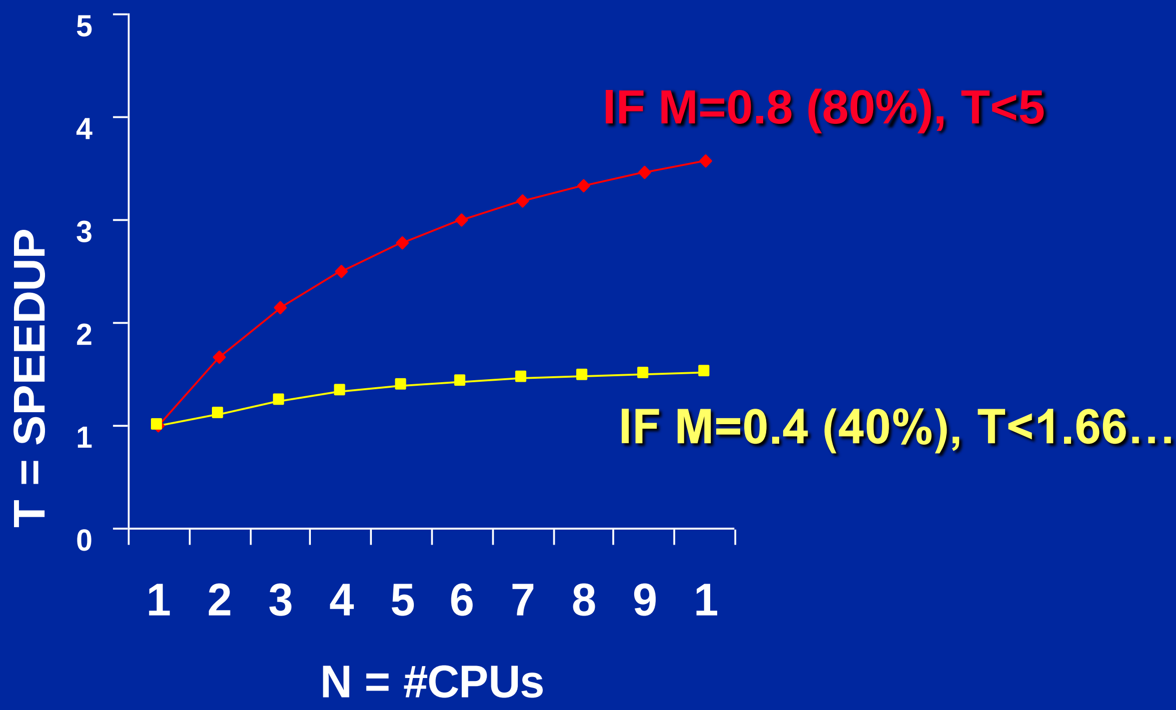
DOES "X" EQUAL 1 OR 10? RANGE MATTERS!

PROCESSOR SCALABILITY

ARBITRARY SPEEDUP IS NOT POSSIBLE

PROCESSOR SPEEDUP (T) FOLLOWS AMDAHL'S LAW:

$$T = 1 / ((1 - M) + (M / N))$$



PERFORMANCE

PERFORMANCE
DEFINITION

- **(MINIMUM) RESPONSE TIME**
  - **TIME TO FIRST RESPONSE**
- **ELAPSED TIME**
  - **AMOUNT OF TIME TO COMPLETE A UNIT OF WORK**
- **THROUGHPUT**
  - **AMOUNT OF WORK COMPLETED IN A TIME PERIOD**
  - **FOR A SINGLE TYPE OF REQUEST**
  - **FOR A SPECIFIC WORKLOAD MIX**
- **CONCURRENCY**
  - **NUMBERS OF USERS ACTIVE**
  - **CONNECTED USERS AFFECT SYSTEM LOAD**

WHAT IS PERFORMANCE?

- COMPARING PERFORMANCE
  - WITH RESPECT TO FIXED RESOURCES
  - FOR A PARTICULAR WORKLOAD
    - » *NUMBER OF USERS, TRANSACTION RATE*
    - » *TRANSACTION COMPLEXITY, DB SIZE, ETC.*
- PERFORMANCE BENCHMARKS
  - RESOURCES AREN'T FIXED
  - WORKLOADS AREN'T WELL-DEFINED
  - RESULTS AREN'T REPEATABLE

Transaction design is crucial!

PERFORMANCE
MINIMUM RESPONSE TIME

MIINIMUM RESPONSE TIME IS PERCEIVED!

- **Defer Confirming Request Send**
- **Confirm Request Receipt Immediately**
- **Give the User More to Do by Not Blocking**
- **Minimize Request Responses**
  - **AVOID UNNECESSARY REPORTS AND BROWSING UPDATES**

PERFORMANCE
ELAPSED TIME
(aka COMPLETE RESPONSE TIME)

- **Minimize Inter-Component Communication**
  - **WITHIN A BUSINESS TRANSACTION**
- **Minimize State Management**
- **Avoid Inter-component Synchronization**
  - **STATE SHOULD NOT BE DISTRIBUTED**
  - **IMPLIES REQUEST CANNOT BE CONVERSATIONAL**
- **Add Resources As Required**
  - **ONLY WORKS IF REQUEST IS NON-PROCEDURAL**

PERFORMANCE
THROUGHPUT

- **Set Task Priorities by Request Class**
- **Balance Load Across Platform Resources**
- **Tune Servers for the Entire Workload**
  - **AVOID TUNING FOR A SINGLE REQUEST**
- **Add Resources to Achieve Desired Throughput**
- **Balance Load Within Each Platform**
  - **PARALLEL SUB-TASKS SHOULD COMPLETE TOGETHER**

PERFORMANCE
CONCURRENCY

RESOURCE CONFLICTS ARE THE PRIMARY ENEMY

- **Minimize Resource Usage**
- **Localize Each Resource Use in Time**
- **Avoid Resource Waits Through Transaction Design**
  - **CONFLICT ANALYSIS CAN HELP WITH SCHEDULING**
- **Use Connection Multi-plexing and Pooling to Minimize Overhead**
- **Balance User Load**
  - **ACROSS PLATFORM RESOURCES**
  - **WITHIN PLATFORM RESOURCES**

TRANSACTIONS

CONCEPTS, DESIGN, AND MANAGEMENT

TRANSACTIONS
DEFINITION

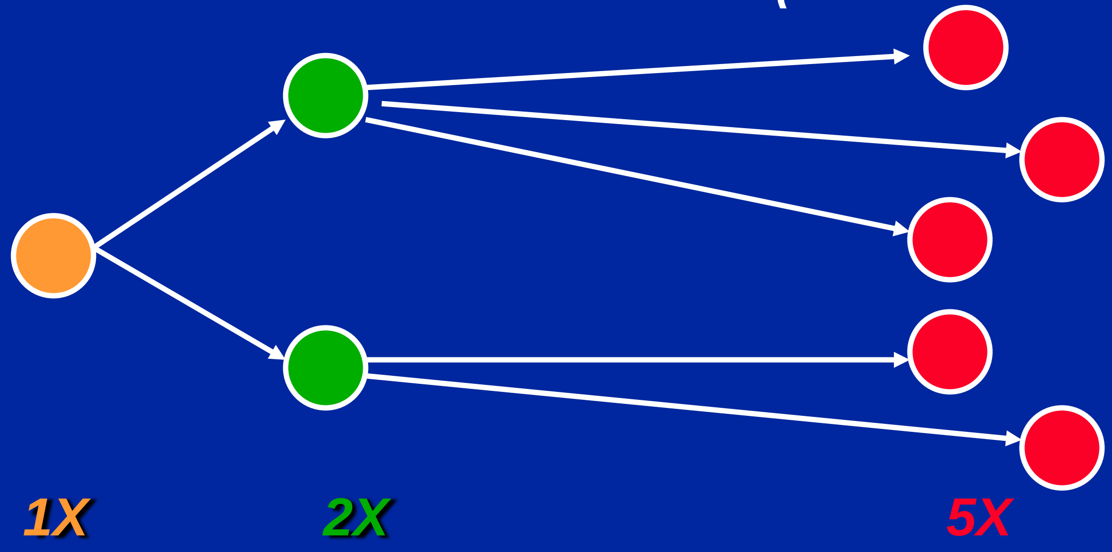
A UNIT OF WORK HAVING WELL-DEFINED BOUNDARIES

- **BUSINESS TRANSACTION**
  - **THE UNIT OF AUDIT**
  - **BOUNDARIES ARE AUDIT POINTS**
- **LOGICAL TRANSACTION**
  - **THE UNIT OF CONSISTENCY**
  - **BOUNDARIES MEET A SET OF CONSISTENCY CONDITIONS**
- **PHYSICAL TRANSACTION**
  - **THE UNIT OF RECOVERY**
  - **BOUNDARIES ARE RECOVERABLE STATES**

UNDERSTANDING TRANSACTIONS
BUSINESS TRANSACTIONS

ONLY *BUSINESS* TRANSACTIONS (UNIT OF AUDIT) ARE IMPLEMENTATION INDEPENDENT

- ***VERSUS LOGICAL TRANSACTIONS (UNIT OF CONSISTENCY)***
- ***VERSUS PHYSICAL TRANSACTIONS (UNIT OF RECOVERY)***



UNDERSTANDING TRANSACTIONS
LOGICAL TRANSACTIONS

- **Maintain Integrity and Consistency**
- **Transition a Database Between Two Consistent States**
- **Requires ACID Properties**
  - **ATOMICITY - ALL OR NOTHING**
    - » ***STATEMENT ATOMICITY IS PART OF RELATIONAL MODEL***
  - **CONSISTENCY**
  - **ISOLATION**
  - **DURABILITY**

UNDERSTANDING TRANSACTIONS

LOGICAL TRANSACTIONS

- **Serializability**
- **Isolation and Anomalies**
  - **LOST UPDATES**
    - » ***ONE TRANSACTION OVERWRITES ANOTHER'S UPDATE***
  - **UNCOMMITTED DEPENDENCIES**
    - » ***ONE TRANSACTION READS/UPDATES ANOTHER'S UNCOMMITTED UPDATE***
    - » ***THE UNCOMMITTED DATA IS SOMETIMES CALLED A "PHANTOM"***

UNDERSTANDING TRANSACTIONS

LOGICAL TRANSACTIONS

- **Isolation and Anomalies (continued)**
  - **INCONSISTENT ANALYSIS**
    - » ***ONE TRANSACTION IS PERMITTED TO READ DATA BOTH BEFORE AND AFTER ANOTHER TRANSACTION UPDATES IT***
    - » ***NON-REPEATABLE READS***
- **Special Types of Transactions**
  - **SAVEPOINTS**
  - **ASYNCHRONOUS TRANSACTIONS**
  - **NESTED TRANSACTIONS**

UNDERSTANDING TRANSACTIONS

LOGICAL TRANSACTIONS

- **Remote Transactions**
- **Distributed Transactions**
  - **TWO-PHASE COMMIT**
- **Explicit Transaction Boundaries**
  - **CRITICAL FOR DISTRIBUTED SYSTEMS!**
  - **NECESSARY FOR TP MONITOR INTERFACES**

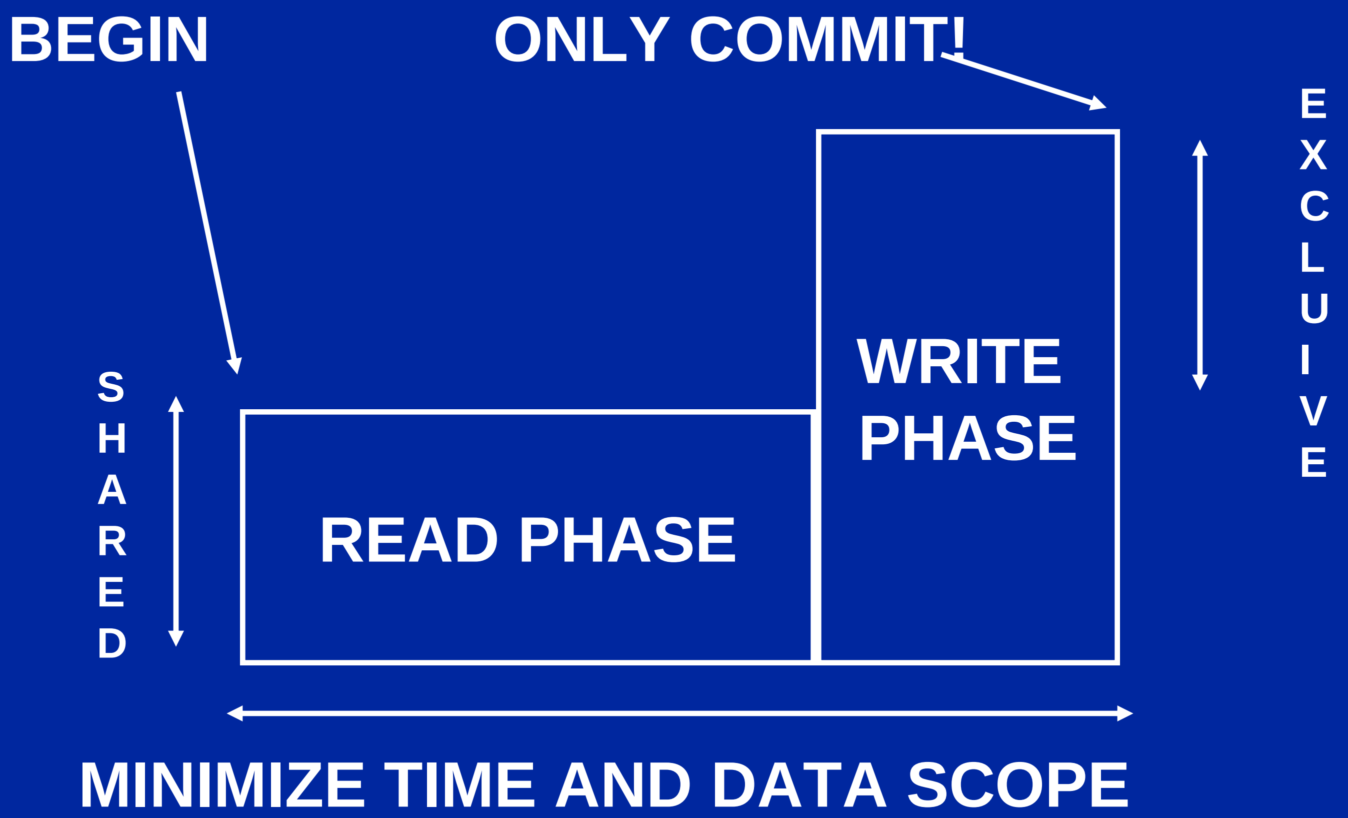
UNDERSTANDING TRANSACTIONS

DESIGN ISSUES

- **Understand transaction structure**
  - AN INITIAL READ PHASE
  - AVOID RE-READING DATA
  - A WRITE PHASE THAT BEGINS WITH THE FIRST INSERT, UPDATE, OR DELETE
- **Minimize the write phase**
  - DATA TOUCHED
  - TIME TO COMMIT
  - CONSIDER PRE-READING DURING THE READ PHASE
- **Minimize transaction scope**
  - MINIMIZE NUMBER OF ACTIONS
- **Non-conversational transactions are best**

UNDERSTANDING TRANSACTIONS

DESIGN ISSUES



TRANSACTION DESIGN
CONFLICT ANALYSIS

- **Identify transactions that can interfere**

- **Why?**
  - **SCHEDULE TRANSACTIONS AND REDUCE CONTENTION**
    - » **Avoid submitting two or more transactions that require locking to guarantee isolation**
    - » **Unfortunately, you must do the scheduling yourself.**
  - **INCREASE RESPONSE TIME AND THROUGHPUT**

TRANSACTION DESIGN
CONFLICT ANALYSIS

Two transactions cannot interfere if:

- **THEY DON'T TOUCH THE SAME DATA**
- **THEY ARE READ ONLY**
- **THEY COMMUTE**

OR

- **<u>THEY DON'T RUN AT THE SAME TIME</u>**

TRANSACTION DESIGN
CONFLICT ANALYSIS

Two Transactions Cannot Interfere If:

– **THEY DON'T TOUCH THE SAME DATA**

– **THEY ARE READ ONLY**

– **THEY COMMUTE**

OR

– **THEY DON'T RUN AT THE SAME TIME**

CONFLICT EXAMPLE

Which pairs of the following can interfere?

- **1: UPDATE SUPPLIERS SET SNAME = ‘NEW_CO_NAME’ WHERE SNAME = ‘OLD_CO_NAME’ AND CITY = ‘NEW YORK’**
- **2: UPDATE SUPPLIERS SET SNAME = ‘OLD_CO_NAME’ WHERE SNAME = ‘NEW_CO_NAME’ AND CITY = ‘NEW YORK’**
- **3: UPDATE SUPPLIERS SET SNAME = ‘NEW_CO_NAME’ WHERE SNAME = ‘OLD_CO_NAME’ AND CITY <> ‘NEW YORK’**
- **4: UPDATE SUPPLIERS SET SNAME = ‘NEW_CO_NAME’ WHERE SNAME = ‘OLD_CO_NAME’ OR CITY <> ‘NEW YORK’**

- **What level of transaction isolation enforcement is required?**
- **What is the effect of existence or non-existence of indexes?**

PRINCIPLES OF SCALABLE DESIGN

WHY DO IMPLEMENTATIONS FAIL?

- **Minimize State Management**
  - BUSINESS FUNCTION REQUESTS
  - MAINTAIN AUDIT POINTS IN A DATABASE
- **Avoid Optimistic Concurrency Control**
  - TOO DIFFICULT TO MAINTAIN CONSISTENCY
- **Implementation and Maintenance Must Be Disciplined**
- **Performance or Scalability Must Be Planned**
- **System Management Must Be Designed-In**
- **Perform a Cost/Benefit Analysis**

WHY DO IMPLEMENTATIONS FAIL?

- **Server Design Should Not Be Too Use Specific**
  - GENERIC SERVER DESIGNS ENSURE FLEXIBILITY
  - DATABASE DESIGNS AND DBMS TUNING AS A SYSTEM
- **Avoid Field-by-field Validation**
  - FROM CLIENT TO SERVER
- **Avoid Excessive Messaging**
  - CACHE DATA WHEN RE-USE IS ANTICIPATED
  - AVOID TRANSACTION ROLLBACK
  - SEND ENTIRE TRANSACTIONS
  - USE SET PROCESSING

APPLICATION DESIGN ISSUES

- Architecture
  - LOCATE PROCESS ACCORDING TO INTEGRITY RULES
  - STATE-FREE VERSUS STATE-DEPENDENT INTEGRITY RULES
- Application type and design
  - USE STATELESS SESSIONS
  - AVOID CONVERSATIONAL SERVER INTERACTIONS
  - CONSIDER MULTIPLE PARALLEL SESSIONS (CHECK OVERHEAD FIRST)
  - USE SET PROCESSING

APPLICATION DESIGN ISSUES

- **Use Proper Transaction Design Techniques**
- **Design for:**
  - **COMPONENT-BASED APPLICATION SERVICES**
    - » ***COARSE GRAINED COMPONENTS RECEIVE FRONT-END REQUESTS***
    - » ***SHOULD SUPPORT BUSINESS TRANSACTIONS***
    - » ***IMPLEMENT VIA FINE GRAINED COMPONENTS***
  - **STORED PROCEDURES**
  - **ASYNCHRONOUS MESSAGES**
    - » ***FRONT-END SHOULD NEVER BLOCK***
  - **TRANSACTION SHIPPING**

DATABASE DESIGN ISSUES

- Normalize the Logical Design
- Avoid Denormalization and Nulls in the Physical Design
- Use Association Tables and Lookup Tables
- Use Surrogate Keys
- Enforce Orthogonality, Completeness, and Minimality Design Principles
- Concurrency and Conflict Analysis

These Provide Implementation Independence!

SUMMARY

- **Good Distributed Application Design Is Different!**
  - DON'T LET OLD HABITS GET IN THE WAY OF SUCCESS
- **Use The Right Architecture for the Job**
  - INVEST IN THE ARCHITECTURE(S) YOU NEED FROM THE BEGINNING
- **Design Your Transactions for Concurrency and Stateless Behavior**
  - SCALABILITY WILL FOLLOW ASSUMING THE ARCHITECTURE IS SCALABLE
  - INSIST THAT YOUR DBMS BECOME MORE AND MORE RELATIONAL

BIOGRAPHY

David McGoveran is a well-known relational database consultant and president of Alternative Technologies (Boulder Creek, CA), specialists in solving difficult relational applications problems since 1981. He published The Database Product Evaluation Report Series; has authored (with Chris Date) A Guide to SYBASE and SQL Server; and is completing Zero Management: Business in the Next Millenium. This seminar is based partially on his workshop: The Client/Server University: Designing Effective Applications.

PLEASE FILL OUT YOUR EVALUATIONS...

Thank you!

