Viewpoint: Concurrency

CSCE 742 - Lecture 14 - 10/25/2018

Software Evolution

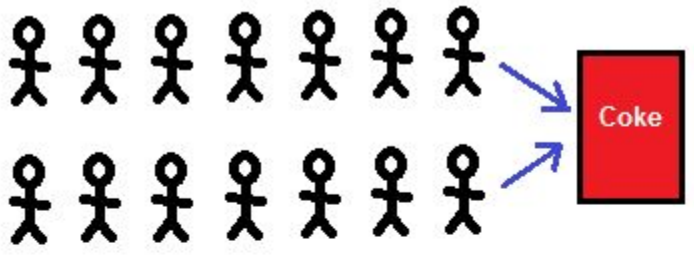
- In the beginning:
  - Information systems designed to run in batch mode on large central computers.
- Now:
  - Moore's law dictates more cores, not faster cores.
  - More focus on real-time response.
  - Information systems are inherently concurrent
- Control systems have always been concurrent.
- **Architects must describe and manage concurrency**

The Concurrency View

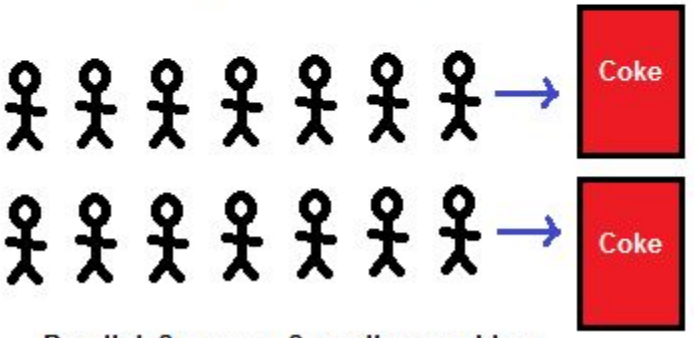
- Describes the concurrent structure of the system and maps functional elements to concurrency units to identify the parts of the system that can execute concurrently.
- Describes how concurrent execution is coordinated and controlled.
  - **Process Model:** Shows processes, threads, and interprocess communication structure.
  - **State Model:** Shows the set of states runtime elements can be in, and how to transition between them

Concurrency Versus Parallelism

- In a parallel environment, operations can be performed simultaneously on duplicated, independent resources.
- In a concurrent environment, multiple processes share resources.
  - Multiple actions can take place concurrently, but only as allowed given resource constraints.
  - Must consider shared resources, multiple consumers/producers, out of order events, deadlocks.



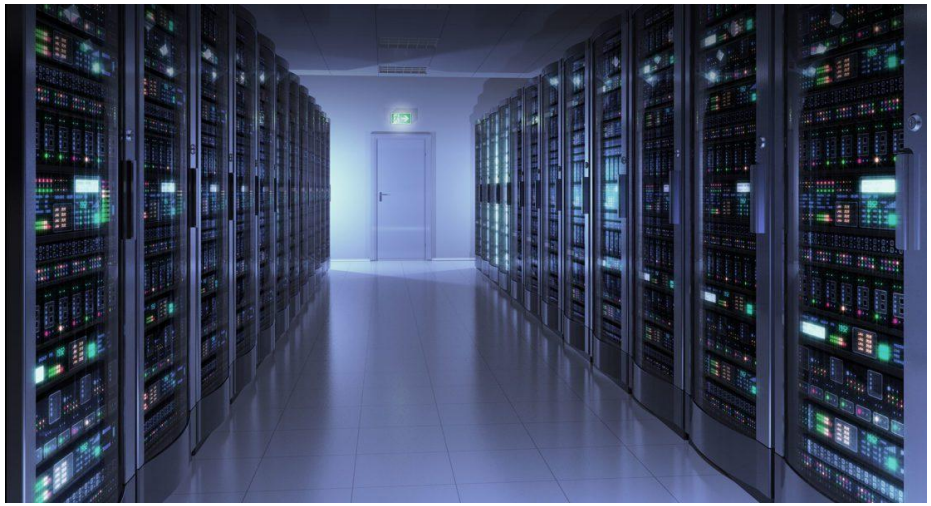
Concurrent: 2 queues, 1 vending machine



Parallel: 2 queues, 2 vending machines

Why Design for Concurrency?

- Scale
- Redundancy
- Security / Isolation
- Better utilization of hardware resources
- Cost: use cheaper “commodity” hardware
- Future flexibility



Concurrency Elements

- **Processes**
  - An operating system process.
    - Address space providing an execution environment for **threads** of execution.
  - Processes are independent, using **interprocess communication mechanisms** to work together.
- **Threads**
  - A thread of execution that can be independently scheduled within a process.
  - Represented through process decomposition.
  - Implementation detail, but affects quality properties, so may need to be discussed.

Concurrency Elements

- **Process Groups**
  - Architecture may group processes into a “single entity” to allow less important concerns to be deferred until later stages of design.
    - DBMS: Will be concurrent, but we don’t need to know how many/what processes it uses.
    - Can be modelled as a single black box, as it is independent and has well-defined interfaces.
  - Complex systems may be modeled in layers, with lower layers represented at higher levels as process groups.

Interprocess Communication

- Processes are isolated. One process cannot directly change another process.
  - Processes must work together through interprocess communication mechanisms.
  - Depicted as connectors in the concurrency model.
- **Procedure Call Mechanisms**
  - Invoke an operation on a process.
  - Remote procedure calls or message passing.
- **Execution Coordination Mechanisms**
  - Allow processes to signal each other when events occur, using semaphores and mutexes.
  - Limited to processes and threads on one machine.

Interprocess Communication

- **Data-Sharing Mechanisms**
  - Allows processes to share data structures and access them concurrently.
  - Shared memory, databases, file storage.
- **Messaging Mechanisms**
  - Transmit data directly from one task to another.
  - Queuing allows consumers to read messages from a queue, then deletes the message (delivered to one consumer).
  - Publisher/Subscriber introduces topics where consumers indicate types of messages of interest. Message consumed by all interested consumers.

Concurrency Concerns

Task Structure

- Task = Generic term for process or thread.
- Task structure: the overall strategy for using concurrency in the system.
  - Partitions the system's workload into tasks.
  - Defines how system functionality is distributed across tasks.
  - May define how tasks are mapped to OS threads.
    - May need to abstract from individual processes and consider groups of processes.

Task Structure

- Details addressed depend on the needs of the system.
  - A complex system with a small footprint may only have 1-2 “tasks”, but those may need to be mapped to a larger number of threads to meet quality goals.
    - Focus of view on thread level.
    - Explain how threads function and communicate, rather than one the core tasks.
  - Large enterprise system with hundreds of processes, each with dozens of threads.
    - Focus on “task” level (groups of related processes), emphasize architectural significance.

Mapping Functionality to Tasks

- How do we map elements to concurrent tasks?
- Affects performance, efficiency, reliability, flexibility of the architecture.
- Which functional elements need to be isolated from each other?
  - Separate tasks/processes.
- Which need to cooperate closely?
  - Same task/process.

Interprocess Communication

- If all elements part of one process, communication and control are simple.
  - Shared memory space, can just transfer control via method calls.
- Communication between processes is more complex, especially across machines.
- Can communicate through remote procedure calls, messaging, shared memory, queues.
  - Each has strengths and weaknesses to consider.
  - Each impacts quality properties.
    - Message queue latency causes scalability issues

State Management

- Concurrent systems often process operations through state machine implementations.
  - Such as locking mechanisms and shared resource management.
- Concurrency view must define set of states each element can be in and how states transition.
  - Part of the runtime behavior of the system.

Synchronization and Integrity

- Concurrent execution often results in corruption of information, if not careful.
  - Shared variables, shared transaction data.
- Concurrency view must address how concurrent activity is coordinated to that data integrity is maintained.

Supporting Scalability

- Task mapping, synchronization strategy, and state management affect scalability.
  - Too many processes or too few can slow down a system.
  - Too much synchronization can cause major performance issues during high workloads.
- Planning for quality is more difficult in a concurrent system.
  - Must address how concurrency strategies will support performance and scalability requirements.
  - System must still be implemented cost-effectively.

Startup and Shutdown

Startup and Shutdown

- Intertask dependencies may require tasks to be started or stopped in specific orders.
  - If some tasks fail, others should not be started.
- Startup and shutdown policies are important part of concurrency design.

Task Failure

- When elements are split into different processes, an element cannot rely on the other element being available.
- Concurrency design needs to account for process failure.
  - Need system-wide strategy for handling and recovering from task failure.

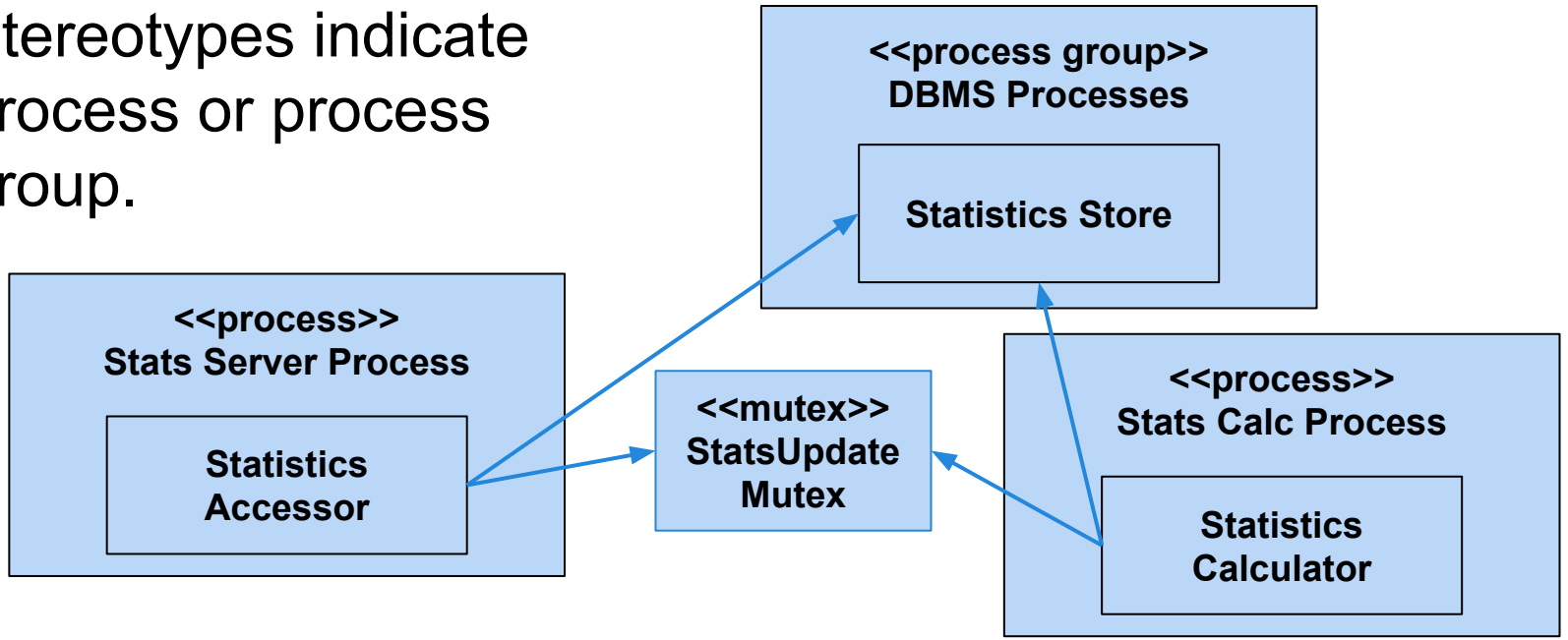
Re-entrancy

- Ability of an element to operate correctly when used by multiple threads.
- Architecture needs to define which elements need to be re-entrant.
- E-mail server implemented as many threads in one process.
  - Any elements related to sending/receiving e-mail need to be re-entrant, as many threads (belonging to many users) will send and receive e-mails at once.
  - Internal state can be corrupted by concurrent access unless re-entrancy is guaranteed.

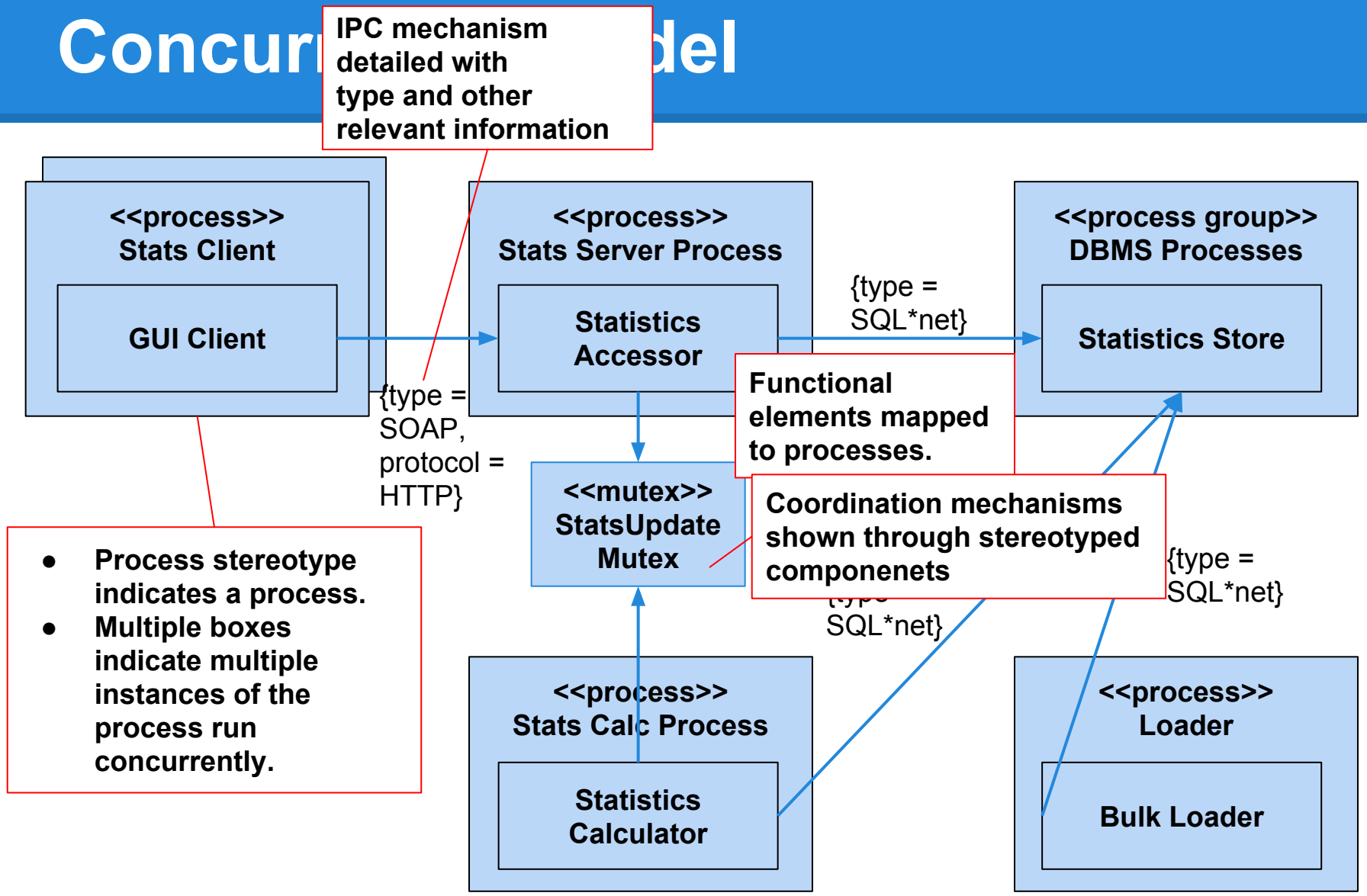
Concurrency Models

Concurrency Models

- Often visualized using UML component models (like Functional View).
- Can be modelled at process or thread level.
- Stereotypes indicate process or process group.
- Arrows on connections indicate direction of communication.
- Can be tagged to make connection clear.

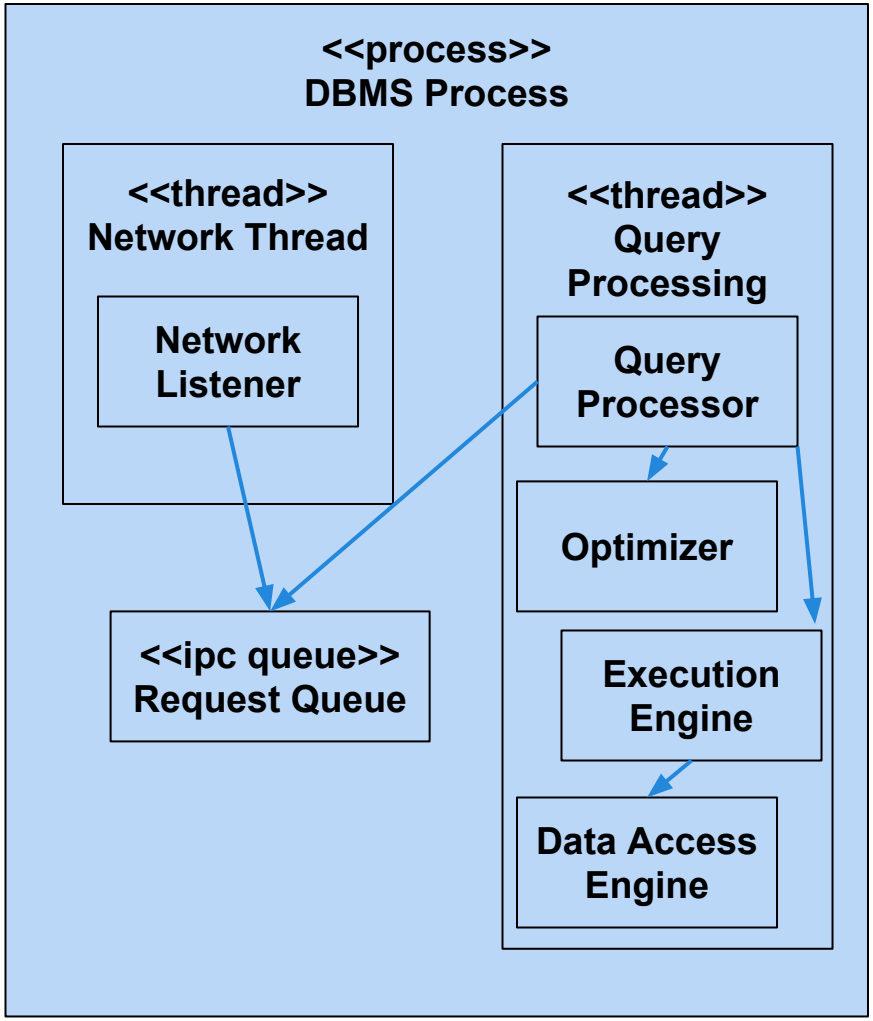


Concurr del

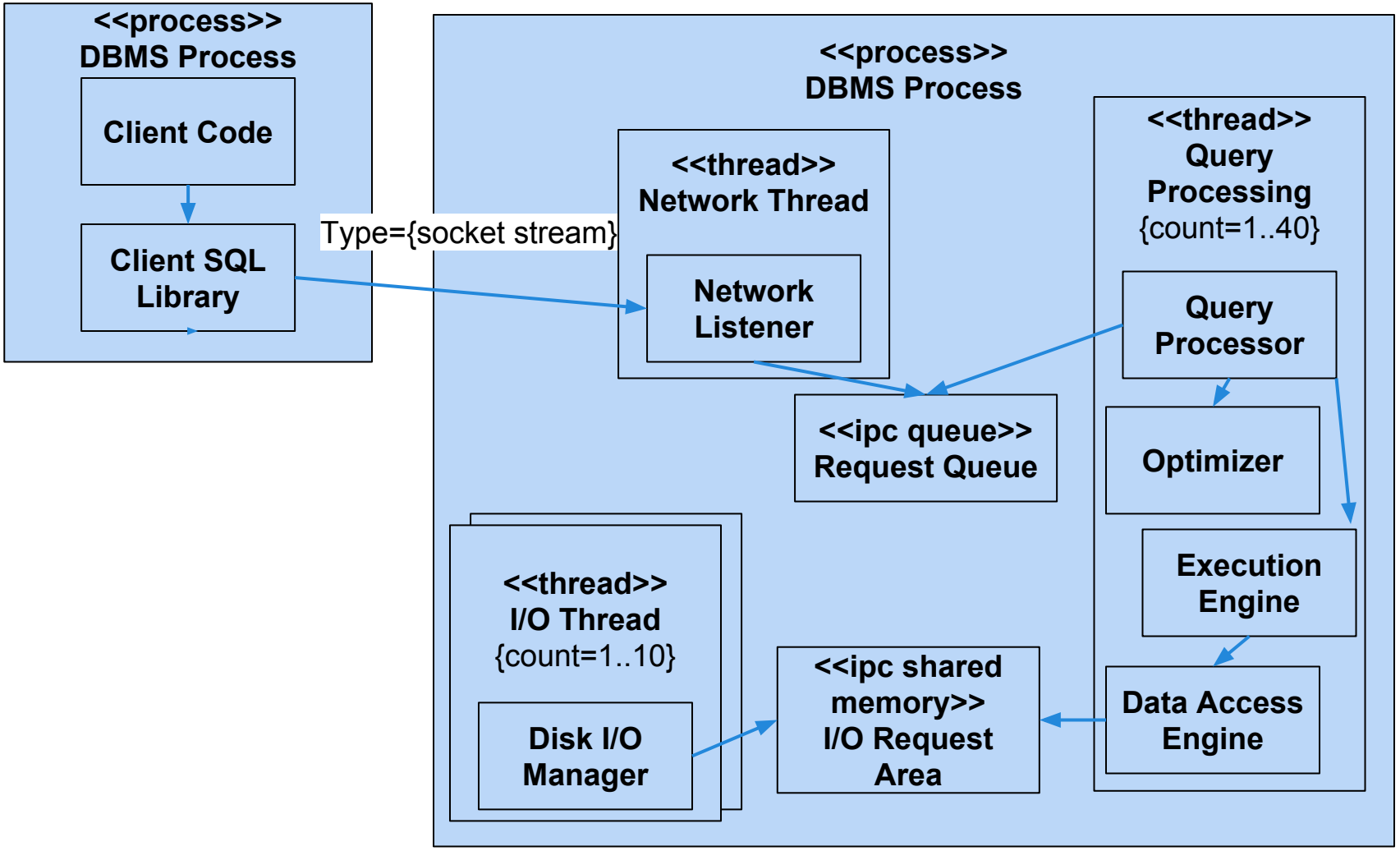

IPC mechanism detailed with type and other relevant information
<<process>>
Stats Client
GUI Client
<<process>>
Stats Server Process
Statistics Accessor
<<process group>>
DBMS Processes
Statistics Store
{type = SQL*net}
{type = SOAP, protocol = HTTP}
Functional elements mapped to processes.
<<mutex>>
StatsUpdate Mutex
Coordination mechanisms shown through stereotyped componenets
{type = SQL*net}
SQL*net}
<<process>>
Stats Calc Process
Statistics Calculator
<<process>>
Loader
Bulk Loader
Process stereotype indicates a process.
Multiple boxes indicate multiple instances of the process run concurrently.

Thread Model

- Threads can be shown within processes using a thread stereotype.



Thread Model



Map the Elements to the Tasks

- Work out how many processes are needed and decide which functional elements will run in each process.
  - Simple case: one element per process, or all elements in one process (not concurrent).
  - More complex: some elements spread over multiple processes.
- Only introduce concurrency when required.
  - Adds overhead to interelement communication.
  - Only add when needed for scalability, distribution, isolation, etc.

Modeling Activities

Determine the Threading Design

- Deciding how many threads per process.
- Often not part of architecture, except to prescribe a general approach or patterns.
  - Important to specify if threading impacts quality properties.

Define the IPC Mechanisms to Use

- Must consider how processes will communicate.
  - Minimize intertask communication.
  - Many libraries available for implementation.
- Define a system-wide approach.

Define Resource Sharing Mechanisms

- Threads share resources (such as memory spaces), and you must control how these resources are used.
  - Usually achieved through a locking protocol.
  - When one process is using a resource, all others are prevented from using it.
- In the architecture, you must define resource sharing in terms of the effect on the system as a whole.

Assign Priorities to Threads and Processes

- Processes on a single machine can be prioritized so more important work is completed before less important work.
  - Done within OS. Tasks are given runtime priorities and controlled by OS thread scheduler.
- In general, avoid explicitly assigning priorities. Complicates model, and can cause problems.
- If needed, keep assignments simple and regular. Reassess throughout development.

Analyze Deadlock and Contention

- Deadlock occurs when a process fails to release a shared resource, and all others wait forever for it.
  - Analysis techniques can identify likely sources of deadlock (i.e., petri net analysis)
- Contention occurs when multiple tasks want a shared resource concurrently.
  - If contention is high, system will slow dramatically.
  - Estimate how many tasks will need a resource concurrently, establish likely wait times.

Concurrency Pitfalls

Modeling the Wrong Aspects

- Your role is on the system as a whole.
  - Low-level individual thread structure and coordination is an implementation detail.
- Model overall concurrency structure and mapping of elements to that structure.
- Specify overall approaches and patterns.
- Focus on architectural significance.
- Involve developers as early as possible so they can plan detailed aspects of problem.

Resource Contention

- Usually manifested as long wait times for shared resources.
- Often impossible to eliminate entirely.
  - Reduce to an acceptable level.
  - Usage scenarios can be used to predict where there will be high levels of concurrency.
  - Decompose locks on large resources into finer-grained locks (reduce time locks held).
  - Perform optimistic locking.
  - Remove shared resources or make immutable.
  - Reduce concurrency around contention points.
  - Avoid locking by using approximately correct results.

Deadlock

- Manifests when processes are waiting for a locked resource that never unlocks.
- Try to redesign to avoid deadlock points.
- Ensure resources are allocated and locked in a fixed order.
- Isolate parallel tasks and control timing.
- Reduce number or duration of locks.
- Some commercial products (DBMS) provide tools for handling deadlock. Use, but be careful that they integrate into your project.

Race Conditions

- Occurs when two tasks try to perform the same action concurrently.
  - Problem if system not planned to deal with concurrent attempts to perform an action.
  - Can result in data loss or corruption.
- Ensure no unprotected shared resources.
- Use immutable data structures.
- Ensure each element interface states whether the interface is re-entrant.

Concurrency is Hard

- Rather than one sequence of instructions, you must consider sequences in parallel.
- Without assistance, you will get it wrong.
  - Application servers (EJB, CICS) ensure sequential access to transactions (“beans”).
  - Polling-concurrent systems do not deadlock (though they can livelock and have race-conditions).
  - Several design styles can be used to ensure deadlock freedom (but may cause contention).
  - Tools such as model checkers or theorem provers are necessary for reasoning about “tricky” concurrent code.

Fallacies of Distributed Computing

- The network is reliable
- Latency is zero
- Bandwidth is infinite
- The network is secure
- Topology doesn't change
- There is one administrator
- **None of these are true.**

The Network is Not Reliable

- Hardware can fail, power can fail, mistakes happen, and the software can (of course) fail
- More complicated if your system works with external systems. You lack control.
- Security threats can take down networks.
- You must consider hw/sw redundancy.
- In software, consider how messages or calls can be lost.
  - Retry, use acknowledgements, ignore duplicates.
  - Verify message integrity.
  - Be able to reorder messages (or ignore order).

Latency is not Zero

- Time it takes for data to move from one place to another.
  - Good on a LAN, goes down quickly over the internet
  - More problematic than bandwidth.
- Do not treat calls over a network like local calls. Do not assume distributed objects act like local objects.
  - Make fewer calls. Move more data with each call.
  - AJAX: While user reads data, retrieve more data.
    - Latency still matters - background downloads must still complete in time.
  - Opnet Modeler, Shunra allow network simulation.

Bandwidth is not Infinite

- While bandwidth grows, so does the amount of information we download.
- Packet loss is a problem for online applications, and we cannot control it.
  - Solution is usually to use larger packet sizes.
  - More data per packet, but fewer need to get through.
- Server: Do not send data until requested.
- Client: Only ask for data you need.
  - And do so in one call.

Networks are always Insecure

- Even if behind multiple firewalls.
- Even if not connected to the internet.
- Some statistics [Symantec 2012]:
  - Net attacks growing by 42% / year
  - Total identities stolen: ~100 Million
  - Avg. exposed identities per breach: 604k
  - Overall e-mail virus rate: 1 per 291 emails
  - Avg. targeted attacks per day: 116
- Perform threat modeling.
  - Decide how to mitigate worst risks.
- Network traffic should always be encrypted.
  - Even behind a firewall.
- Handle security on network, infrastructure and in the software.

Network Topologies Change

- Ops team adds/removes servers and changes network implementation.
- Server/network faults change routing.
- Do not depend on specific endpoints/routes.
  - Provide local transparency (multicast) or use discovery services (Active Directory)
  - Abstract physical structure of network.
    - **Moving webpages:** DNS routing tables can change IP address a domain points to without problems, as transparency is provided to client.

There are Multiple Administrators

- Most groups have many administrators.
- Your service might interface with services owned by other organizations.
- You have no control over who has admin rights in these organizations (or in your own)
  - Other domains may not trust your app.
  - Restrictions on allows APIs.
  - Must provide tools to admins to diagnose problems.
  - Other domains may upgrade (or not) services. You lack control.

General Concurrency Principles

- When possible, let someone else do it!
  - Middleware
  - Component frameworks
- Manage synchronization between threads using asynchronous queues.
  - Languages have thread-safe queueing primitives
  - Removes most issues with deadlock, race conditions
- Try not to share data between threads
  - No concurrency issues if data is not shared!
  - “Globals” for threads can use thread-local storage
- Communicate in large chunks
  - Use bandwidth efficiently

Food for Thought

- Is there a clear concurrency model?
- Are your models at the right level of abstraction? Have you focused on the architecturally significant aspects?
- Can you simplify your concurrency design?
- Do all interested parties understand the overall concurrency strategy?
- Have you mapped all functional elements to a process (and thread if necessary)?

Food for Thought

- Have you defined interprocess communication mechanisms?
  - Have you minimized intertask communication and synchronization?
- Are all shared resources protected?
  - Do you have any resource hot spots?
  - If so, have you estimated likely throughput?
  - Do you know how you would reduce contention at these points if forced to later?
- Can the system possibly deadlock?
  - Do you have a strategy for recognizing and dealing with this when it occurs?

Next Time

- Development and Deployment Viewpoints
  - Sources: Rozanski & Woods, Ch. 20-21
- Homework:
  - Assignment 2 - Due tonight!
  - Reading Assignment 2 - Due November 1st
    - Arnon Rotem-Gal-Oz, “Fallacies of Distributed Computing Explained”
      - Summarize the fallacies.
      - Do you believe that these problems still stand today? (The paper is from 2014!)
      - Do you believe these problems can be overcome in the future?

Project Part 3

- Due Nov 18 (Get Started!)
- Describe the architecture at multiple layers of abstraction and also from multiple viewpoints.
  - For the system itself and a chosen subsystem, document the functional view and one other view.
  - Discuss how perspectives impact these views.
  - Functional view should include UML sequence diagrams to illustrate scenarios.
  - Propose either a refactoring to the architecture or a concrete extension.