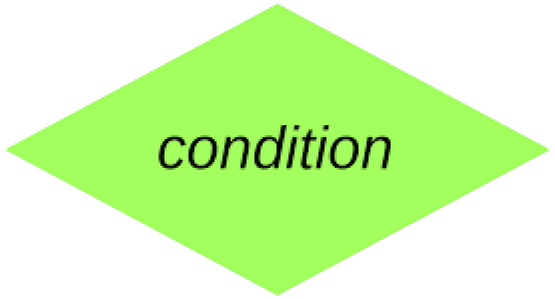
Application Design Using Java

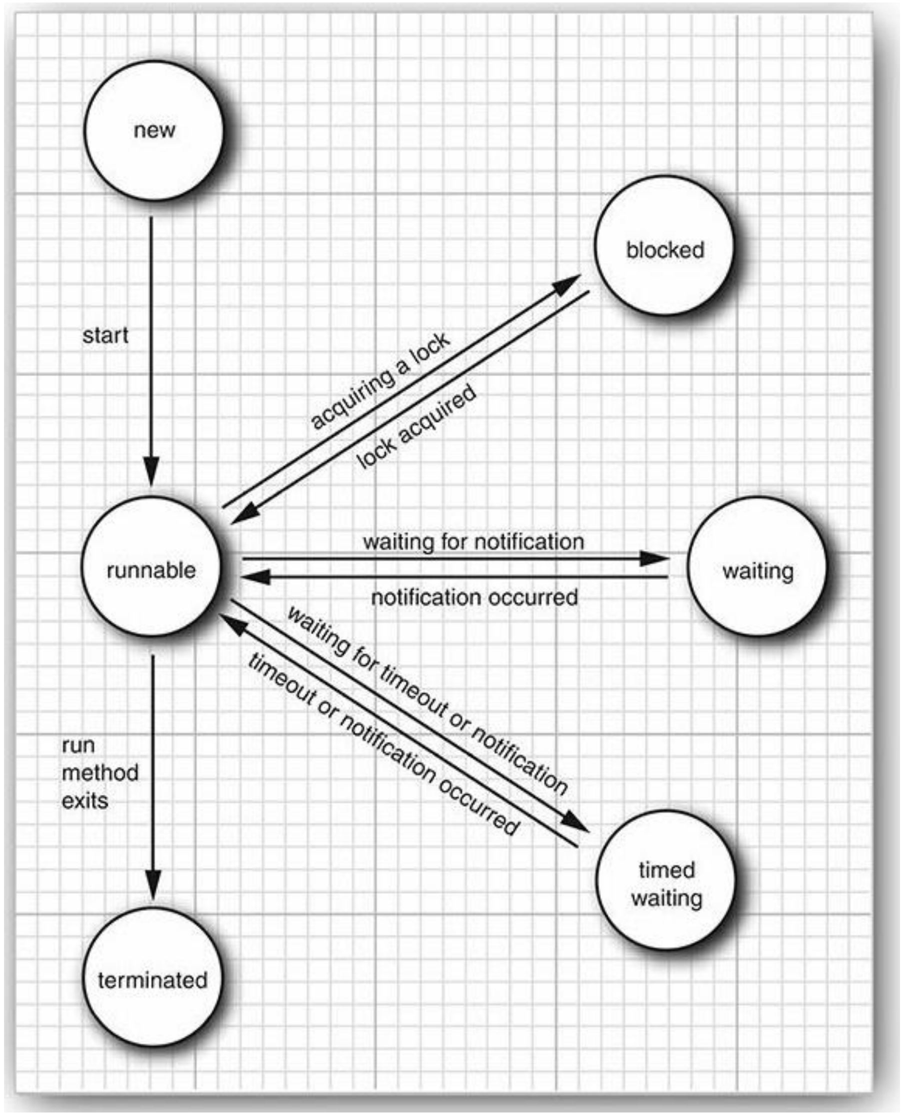
Lecture 14

Conditions



- Condition
- Intrinsic condition (methods *wait()*, *notify()*, *notifyAll()*)

Thread States



Synchronization

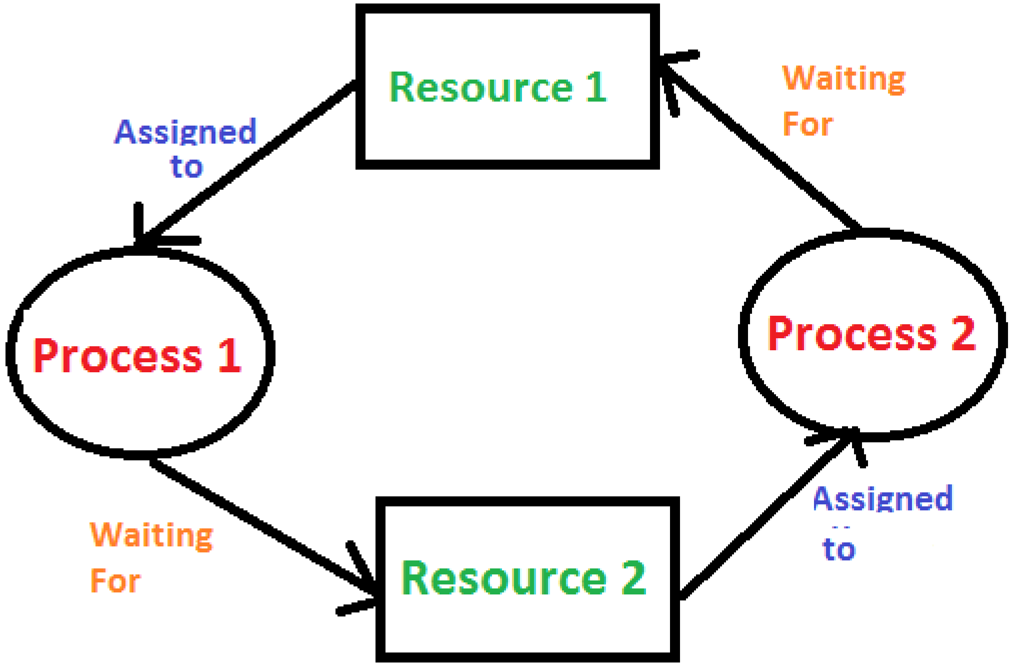
- *volatile* keyword
    - The compiler and the virtual machine take into account that the field may be concurrently updated by another thread
    - A change to a volatile variable in one thread is visible from any other thread that reads the variable
    - No guarantee of atomicity if operations other than assignment are performed
- *final* variables
    - Are safe to be accessed by multiple threads
    - If the object is mutable, synchronization is still required

Atomics

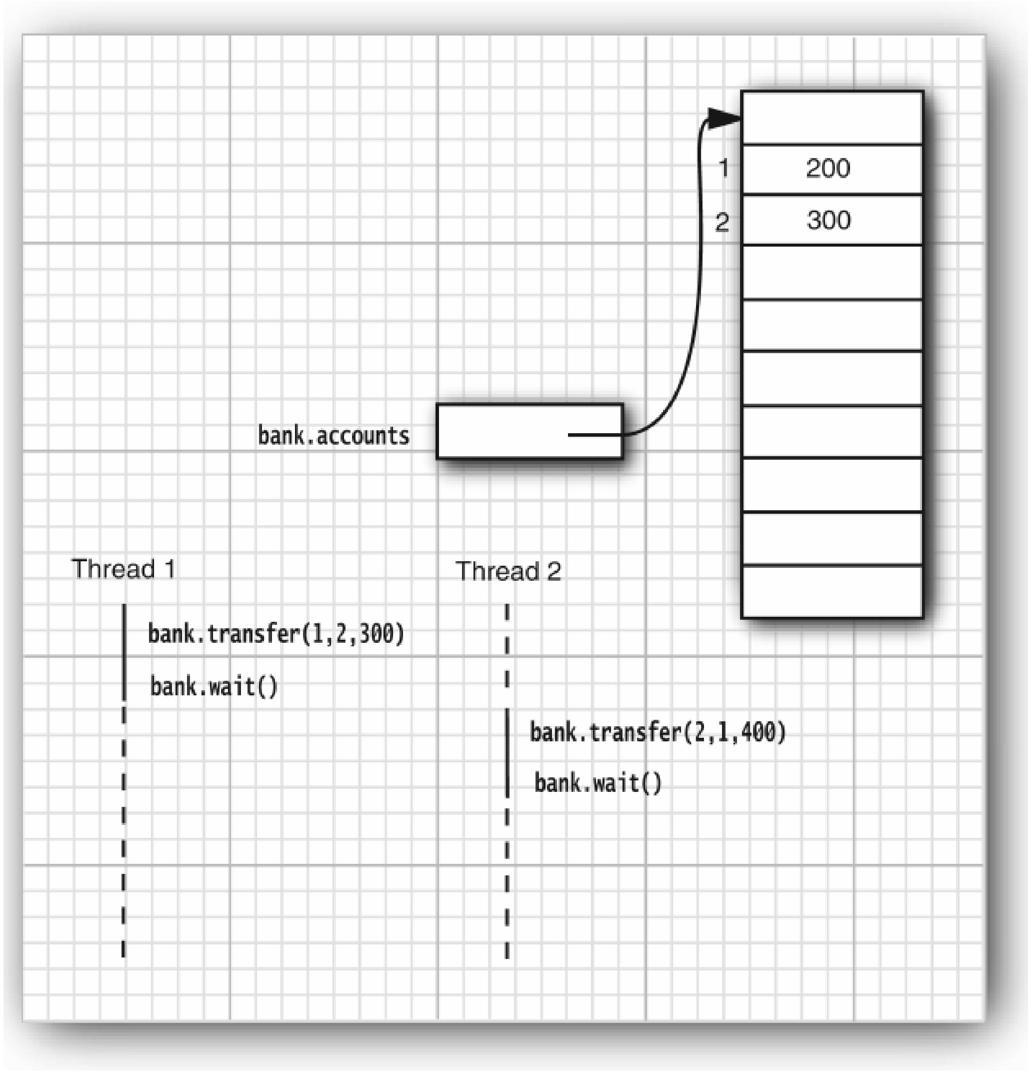
- Classes with atomic operations (java.util.concurrent.atomic package)
- E.g., AtomicInteger, AtomicLong, AtomicReference, AtomicLongArray, etc.
  - Atomic increment
    ```
    public static AtomicLong nextNumber = new AtomicLong();
    // In some thread...
    long id = nextNumber.incrementAndGet();
    ```
  - Keep track of the largest value observed by different threads
    ```
    do {
      oldValue = largest.get();
      newValue = Math.max(oldValue, observed);
    } while (!largest.compareAndSet(oldValue, newValue));
    ```

Deadlock I

- A requested resource is held by a waiting process
- That process in turn is waiting for another resource held by another waiting process



Deadlock II


1
200
2
300
bank.accounts
Thread 1
Thread 2
bank.transfer(1,2,300)
bank.wait()
bank.transfer(2,1,400)
bank.wait()

Blocking Queues

- Many threading problems can be formulated in terms of a queue
- Producer – consumer idea:
    - Producer threads insert items into the queue
    - Consumer threads retrieve them
- Blocking queue
    - Thread blocks when you try to add an element when the queue is currently full
    - Thread blocks when you try to remove an element when the queue is empty

Synchronizers

- Help manage a set of collaborating threads:

Class	What it Does	Notes
CyclicBarrier	Allows a set of threads to wait until a predefined count of them has reached a common barrier, and then optionally executes a barrier action.	Use when a number of threads need to complete before their results can be used. The barrier can be reused after the waiting threads have been released.
Phaser	Like a cyclic barrier, but with a mutable party count.	Introduced in Java SE 7.
CountDownLatch	Allows a set of threads to wait until a count has been decremented to 0.	Use when one or more threads need to wait until a specified number of events have occurred.
Exchanger	Allows two threads to exchange objects when both are ready for the exchange.	Use when two threads work on two instances of the same data structure, with the first thread filling one instance and the second thread emptying the other.
Semaphore	Allows a set of threads to wait until permits are available for proceeding.	Use to restrict the total number of threads that can access a resource. If the permit count is one, use to block threads until another thread gives permission.
SynchronousQueue	Allows a thread to hand off an object to another thread.	Use to send an object from one thread to another when both are ready, without explicit synchronization.

Multithreading in Swing

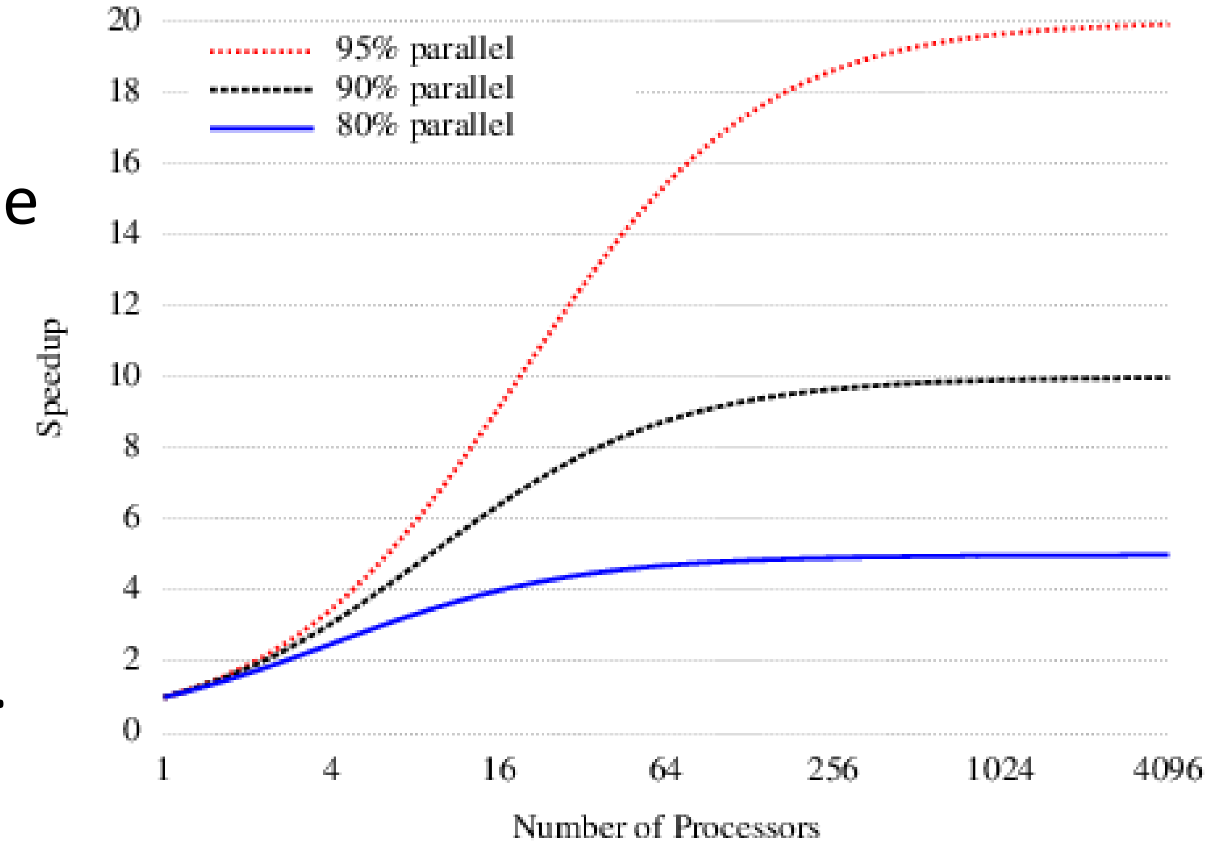
- If an action takes a long time, do it in a separate worker thread and never in the event dispatch thread
- Do not touch Swing components in any thread other than the event dispatch thread

Measuring Performance

- Speedup = t_1 / $t_{N,}$
  where t_1 is the computational time on one processor, and t_N is the computational time running the same program with N processors. Determines how much faster parallel execution is versus serial execution.
- Efficiency = Speedup / N
  Indicates how well software utilizes the computational resources of the system.

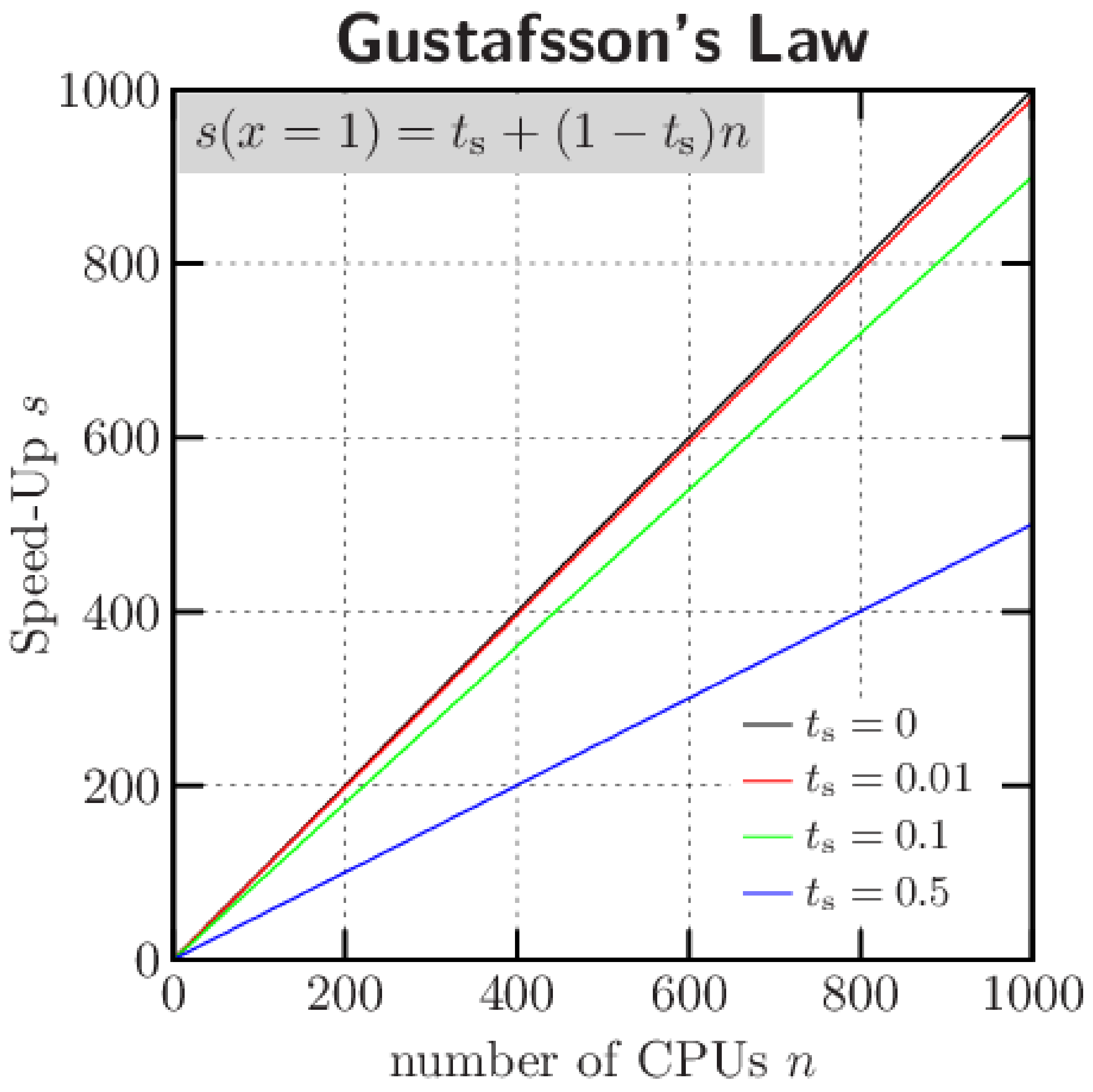
Amdahl's Law

- $Speedup \leq \dfrac{1}{(1-f) + \frac{f}{N}},$
  where *f* is the proportion of the program that can be made parallel, (correspondingly, $(1 - f)$ is the proportion of the program that remains sequential),
  N is the number of processors.



Gustafson's Law

- $Speedup \leq N + (1 - N)(1 - f)$



Types of Scalability

- Strong scaling
  How long it takes to solve a problem with the fixed *total* size as we vary the number of processors.
- Weak scaling
  How long it takes to solve a problem with the fixed size *per processor* as we vary the number of processors.

//TODO before next lecture:

- Peer Grading for Homework 2 due 3/16 at 11:59 pm EDT.
- Final Project team formation due on 3/19 at 11:59 pm EDT. Teams must be declared on Submitty.
- Article Review and Presentation was posted. You need to let me know which article you want to review and get my approval, if you haven't already. This ensures that there are no two students reviewing the same article. Drafts are due on 3/29 at 11:59 pm EDT. Must be submitted on Submitty.
- Java puzzler (posted on Submitty Forum)