Lecture 29
Review
Final Exam

- Mon. 5/16 10:30 am – 12:30 pm
- Cumulative!
- Open book / open notes / open off-line electronics / closed neighbor
Topics Covered: Midterm

- Parallel / concurrent / distributed systems
- Nondeterminism
- Procedural abstraction
- Processes and threads
- Scheduling
- Context switching
- Testing multi-threaded programs
- Types of testing: functional / performance / stress / unit / integration / acceptance
- Interleavings and how to count them
- Forcing interleavings via `Thread.yield()` and `Thread.sleep()`
- Threads as objects in Java
- Thread class
- Runnable interface
- Thread states
- User vs. daemon threads
- Thread safety
- Data races
- Race conditions
- Class specifications, correctness
- Thread safety
- Atomicity
- Locks
- Intrinsic / monitor locks
- Synchronized blocks, methods
- Reentrant locks
- Locks and performance
- Locking protocols
- Deadlock
- Waits-for graphs
- Deadlock prevention
- Built-in atomic memory access in Java
- Synchronization and visibility
- Volatile variables / fields
- Locking and visibility in Java
- Object publishing and escape
- Indirect publishing
- Improper object construction and escape of `this`
- Safe object construction via factory methods
- Thread confinement
- Stack confinement
- ThreadLocal
- Immutable objects and final fields
- Initialization safety
- Safe publication
- Effectively immutable objects
- State-dependent actions
- Balking / guarded suspension / optimistic retry
- `wait()` / `notify()` / `notifyAll()`
- `notify()` and deadlock
- Timed waiting
- Nested monitor lockout
- `Collections.synchronizedXXX()` (XXX is the name of a type of collection)
- Thread safety, compound actions and client-side locking
- `ConcurrentModificationException` and hidden iteration
- Concurrent collections
- `ConcurrentHashMap` and lock striping
- Fail-fast vs. weakly consistent iterators
- `CopyOnWriteArrayList`
Topics Covered Since Midterm

- ConcurrentLinkedQueue
- Blocking queues
- SynchronousQueue
- Producer/Consumer pattern
- InterruptedException
- Synchronizers
- Explicit (reentrant) locks
- Conditions, await() / signal() / signalAll()
- Latches
- Futures / FutureTasks
- Counting semaphores
- Barriers
- Tasks, task boundaries
- Executors, thread pools and execution policies
- Executor shutdown and rejected execution handling
- CompletionService
- Thread-starvation deadlock
- Thread-pool sizing
- Parallelizing algorithms: iterative, recursive
- Tail recursion
- Performance tuning
- Parallelization and dependent tasks
- Fork/Join framework and divide-and-conquer
- Deques and work stealing

- Distributed and multi-process computing
- Actors and message passing
- Asynchronous, at-most-once-delivery, locally FIFO message passing
- Akka actor implementation
- Location transparency in akka
- Akka actor systems
- Creating actors in akka
- tell()
- Patterns.ask()
- getSender()
- getSelf()
- Requirements for classes to be message types in akka
- ActorContext in akka
- Actor supervision and fault-tolerance in akka
- Resume vs. restart vs. stop vs. escalate
- ActorPaths in akka
- Stopping actors in akka
- Dead letters
- Akka and the Java Memory Model
- Communication protocol design for akka actors
- Sequence diagrams and message sequence charts
- Designing message classes
- Testing actor systems
Topics Covered Since Midterm (cont.)

- MapReduce and “big-data” processing
- MapReduce and map, fold operations from functional programming
- Hadoop design principles, components
- Hadoop execution flow
- Hadoop fault-tolerance
- Java Memory Model
- Events
- Program order
- Program executions
- “happens-before” and data races
- Sequential consistency
- Nonblocking algorithms
- Compare and set
- Compare and swap
- AtomicInteger / AtomicBoolean / AtomicLong / AtomicReference
- Nonblocking stacks
- Nonblocking queues