Classification of concurrency

- Definitions: parallelism vs. concurrency (examples of both)
- Reasons for concurrency (examples)
- Classifications: task/data, coarse/fine-grained, high/low level, implicit/explicit, by application areas, by computation model (examples for all)
- why to use language for concurrent computations (+your own view/experience, incl. negative, is welcome)
- Main concurrent features/constructs of languages: Erlang, Clojure, Haskell (yes, learning by hard may sometimes be useful too)

Problems, locks, and monitors

- Accessing a shared variable: problems of synchronization between 2 threads, including memory hierarchy, race condition (with examples)
- shared state in a data structure, thread-safety of a function/Java object, GUI as a shared state, event dispatching thread (EDT)
- Locks, Java synchronized block/methods (details/examples), deadlock
- Monitors: how wait(), notify() work with example (i.e. typical usage pattern included)
- Deterministic execution models: dependencies of actions, interleavings in program execution, single-assignment variables, eager/lazy evaluation (Oz/Haskell), sources of non-determinism

Declarative concurrency

- Declarativeness: definition, how important, declarativeness in Clojure
- Immutable data structures: linked lists, appending lists, (example in Clojure)
- IDT: Immutable trees, changing/mutating trees (graphical example)
- Iterative computation (how to do without mutable variables): Clojure recursion and loop construct, examples, problems with recursion
- Tail call optimization: problem, tail position, TCO in JVM
- Problems with recursion: wasteful or slow implementations, examples, solutions
- Idiomatic functional programming: description of map, filter, reduce with usage examples
- Benefits of IDT, transient counterparts and how/where to use them

More immutable data structures

- Persistent vector and map in Clojure and how they work (partially ex-slide material), time and space complexities
- Immutable (sorted) binary search tree (red-black tree): code of traversal, description of element insertion ideas (no need of very precise algorithm)
- Immutable queue: two lists, description (Clojure or pseudocode) of how it works (empty?/insert/pop)
Ref. types and STM

- Transactions, ACID properties, isolation levels
- Clojure 4 reference types: general description and quick comparison, atoms, example of conflict with `swap!`
- Clojure `ref`, transactions, Clojure STM semantics, ref-history-queue
- Write-skew problem, example with `ensure`

Agents

- General ideas of agents/actors: agents vs. actors, passive/active objects, compare to RPC
- Clojure agents: agents mechanics, thread pools, state transform functions, examples in Clojure
- Handling errors in Clojure agents, side-effects, IO, STM, nested sends
- Agent diagrams: notation, semantics, example
- Futures and promises: Clojure and javascript/jquery (partially ex-slide material), pipes, pipe cascading

Clojure tasks

Write Clojure code for one or two given problems. Includes declarative/functional programming and Clojure refs types (i.e. no protocols or macros). Usage of cheat sheets and similar references are ok, syntax/or other minor mistakes and typos are not important.