Dealing with Nondeterminism
Multithreaded apps are hard to ... 

... write, test and debug

Nondeterministic schedule =>

Write: Reason about multiple possible executions
Test: No guaranteed behavior
Debug: Bugs are hard to reproduce
Deterministic multithreading (DMT)

- same input -> same schedule

- Many proposals:
  - HW: DMP(ASPLOS’09), DTHREADS(SOSP’11), etc.
  - SW: Kendo(ASPLOS’09), CorDet(ASPLOS’10), etc.
Research questions

• How to make DMTs faster?
• Are DMTs enough?
How to make DMTs faster? (One explored idea)

• mem-schedule (strong determinism): truly deterministic, even for programs with data races
  deterministic schedule of shared memory accesses (load/store instructions)

• sync-schedule (weak determinism): efficient
  deterministic order of synchronization operations (lock/unlock)
How to make DMTs faster? (One explored idea)

• Sync-schedule
  • Kendo(ASPLOS’09) - deterministic logical time (independent for each thread); acquire locks in deterministic logical time order (using the turn concept)

• Hybrid-schedule
  • Peregrine(SOSP’11) - first run -> a detailed trace -> relax trace into a sync-schedule + execution order constraints for each data race
Are DMTs enough?

• dOS - framework to help programmers deal with external nondeterminism as well

• Stability (input perturbations, slight changes in execution environment - e.g., shared libraries)
Stable multithreading (SMT)*

- Stateful schedules
- Reuses schedules if input constraints met
  - TERN(OSDI’10) - uses KLEE(OSDI’08) to track input constraints (similar inputs, same schedule)
  - PARROT(SOSP’13) - much smaller set of schedules

*Not to be confused with simultaneous multithreading