The Realm of Shared Variables

- Originates in multiprogramming on mono/multi-processors

Examples

- Threads sharing a common address space
- Processes sharing a common file (data base).
- Distributed objects

Observations

- Execution seen as interleaving of atomic actions
- Atomicity of anything but read/writes must be implemented

Selection of Granularity

- Atomic actions must be large to ensure *consistency*
- Atomic actions should be small to allow for parallelism
- Starting point:

  Operations on abstract data types should be atomic

- A *concurrent object* is a shared data-structure with atomic operations
- *monitor* = concurrent object with atomicity by locking
- Monitors should always be the first choice

Implementation of Concurrent Objects

Implicitly by Syntactic Support

- Monitors + condition queues.
  Concurrent Pascal, Pascal Plus, Mesa:
  monitor, **condition**, $wait(c)$, $signal(c)$
  Ada95: **protected object**, **when** B, **requeue**
- Critical sections
  Java: **synchronized** $(o)\{\ldots\}$, **synchronized**
  C#: **lock** $(o)\{\ldots\}$
- Does not allow for fine-tuning

Explicit use of Synchronization Primitives

- Semaphores (almost any operating system)
- Mutex + conditions (Pthreads, C#/.NET, Java 1.5, **not** Win32)

Monitor Issues

- Nested monitor calls
  - Drop monitor while calling blocking operations
- Deadlocks
  - Recursive locks
  - Locking hierarchy
  - Deadlock detection/retry
  - Combining monitors
- Locking of whole structure — degrades concurrency
  - Apply *R/W locking* of operations
  - Use *individual locking* of parts
  - *Loosen* the locking (!!)
- Synchronization overhead
  - Prevent sharing by outer critical region
  - Use *spin locks* (on multiprocessors)
  - Use *non-blocking synchronization* (at low or high level)