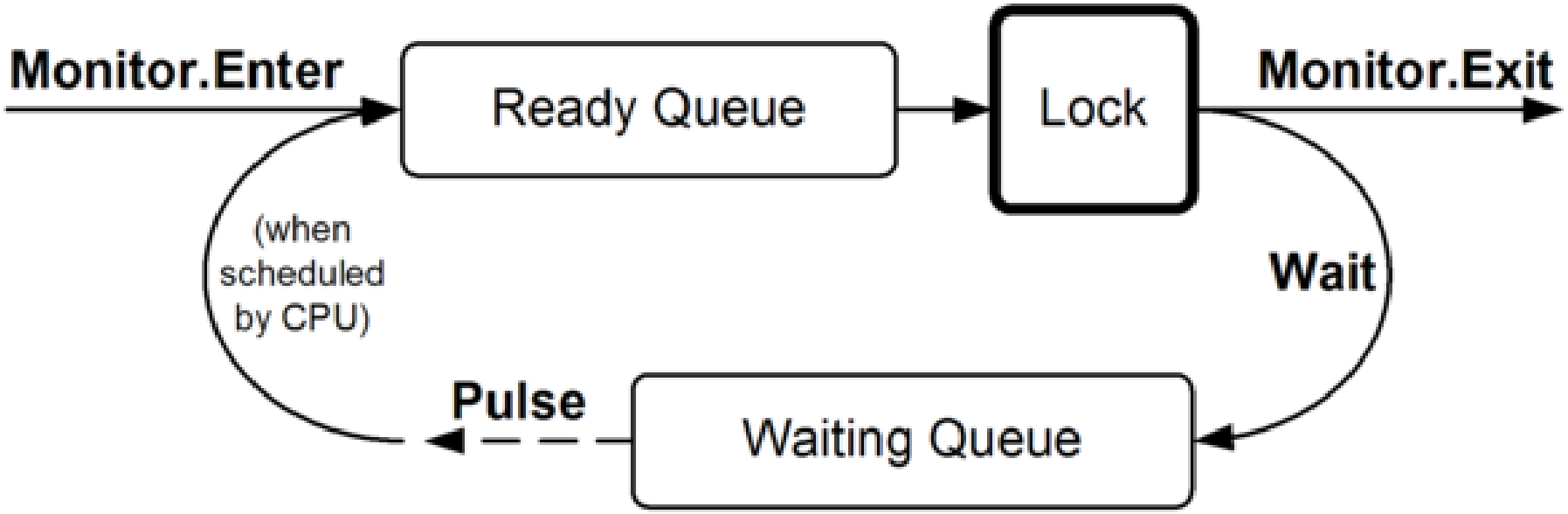
C# AND SQL CONCURRENCY

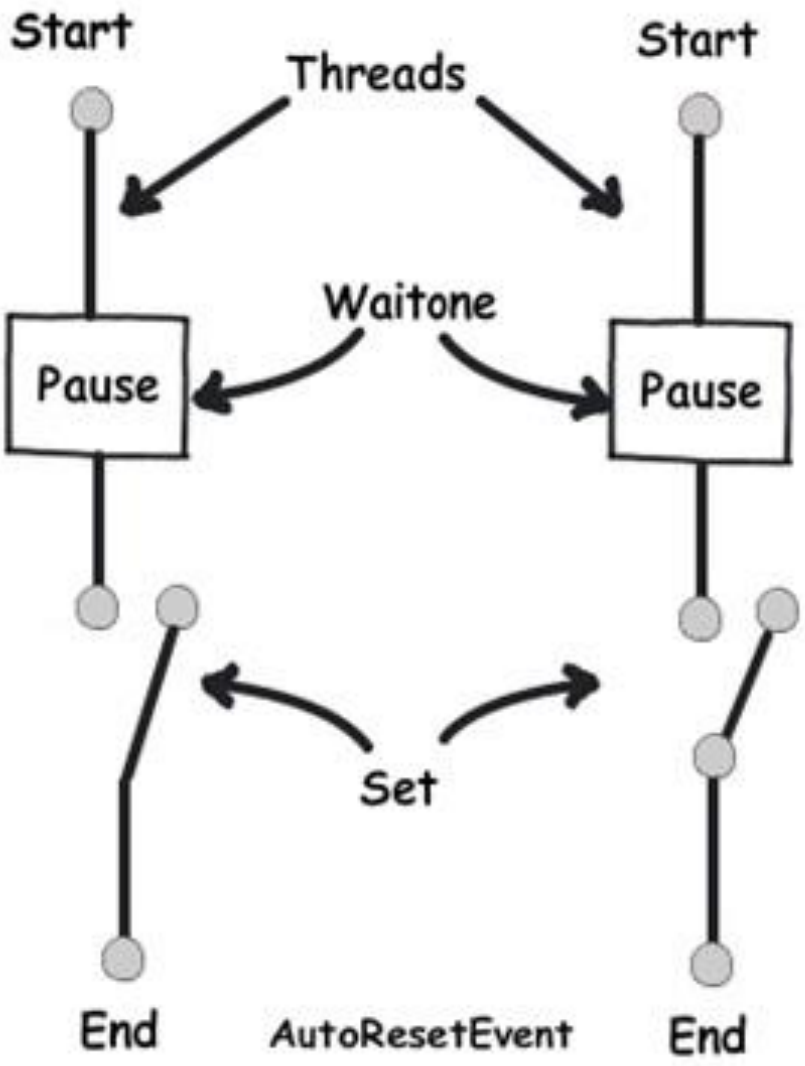
C# SIGNALING MECHANISMS

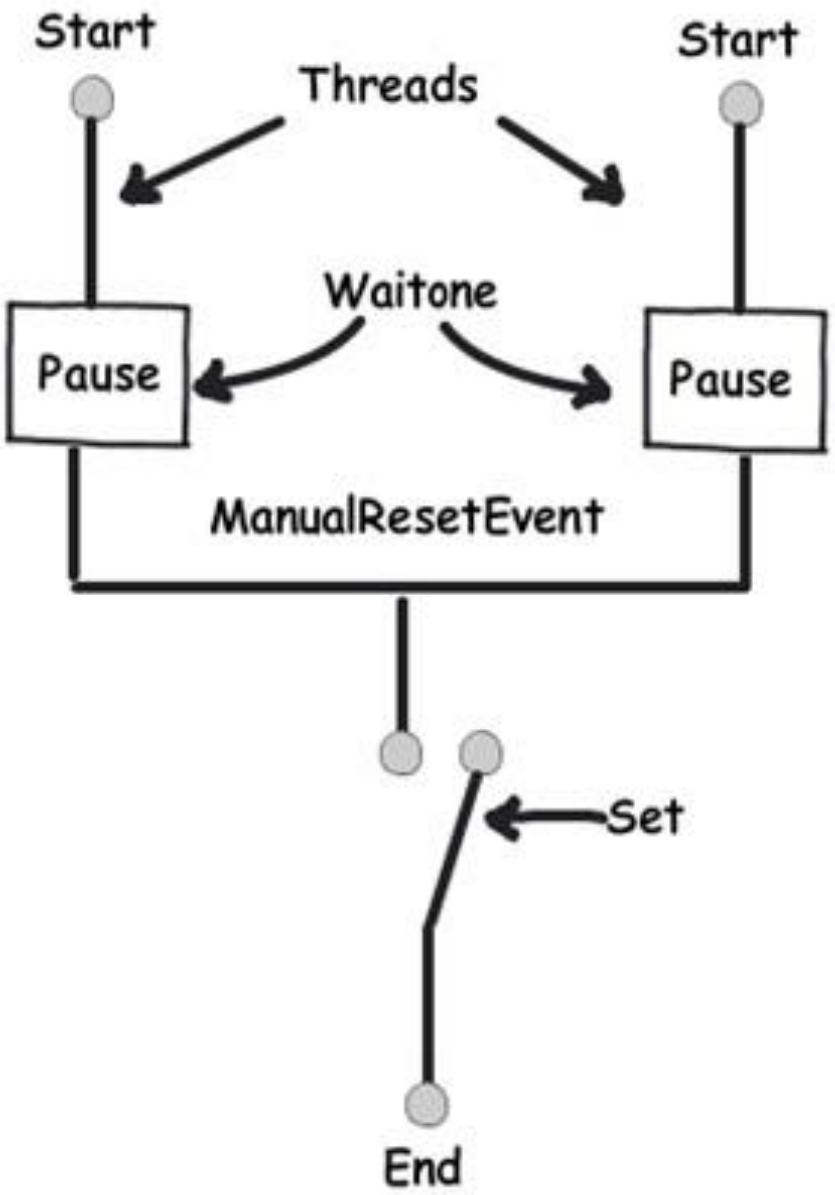
Coordinate Threads, Prevent Chaos

- **`Monitor.Wait/Pulse`**: Mutual exclusion and notification (via `lock` keyword).
- **`AutoResetEvent`**: Releases one waiting thread, then auto-resets.
- **`ManualResetEvent`**: Releases all waiting threads, requires manual reset.

C# SIGNALING MECHANISMS




Start
Threads
Start
Waitone
Pause
Pause
Set
End
AutoResetEvent
End


Start
Threads
Start
Waitone
Pause
Pause
ManualResetEvent
Set
End

CHANNELS - MODERN MESSAGE PASSING

Decouple & Stream Data Asynchronously

- **System.Threading.Channels**: Asynchronous, thread-safe producer-consumer queues.
- **Decoupling**: Producers and consumers don't share direct state.
- **Async-Friendly**: Built for async/await patterns.
- **Bounded/Unbounded**: Control capacity and flow.

CHANNELS - MODERN MESSAGE PASSING

```
var tasks = transactions.Select(t => Task.Run(() => ProcessTransaction(t)));
await Task.WhenAll(tasks); // Result: Memory explosion, thread pool starvation
```

```
var channel = Channel.CreateBounded<Transaction>(1000);

var workers = Enumerable.Range(0,
Environment.ProcessorCount).Select(_ =>

Task.Run(async () => {
   await foreach (var transaction in channel.Reader.ReadAllAsync())
      await ProcessTransaction(transaction); }));

 foreach (var transaction in transactions)
    await channel.Writer.WriteAsync(transaction);
```

C# Concurrency: Shared Memory vs. Channels

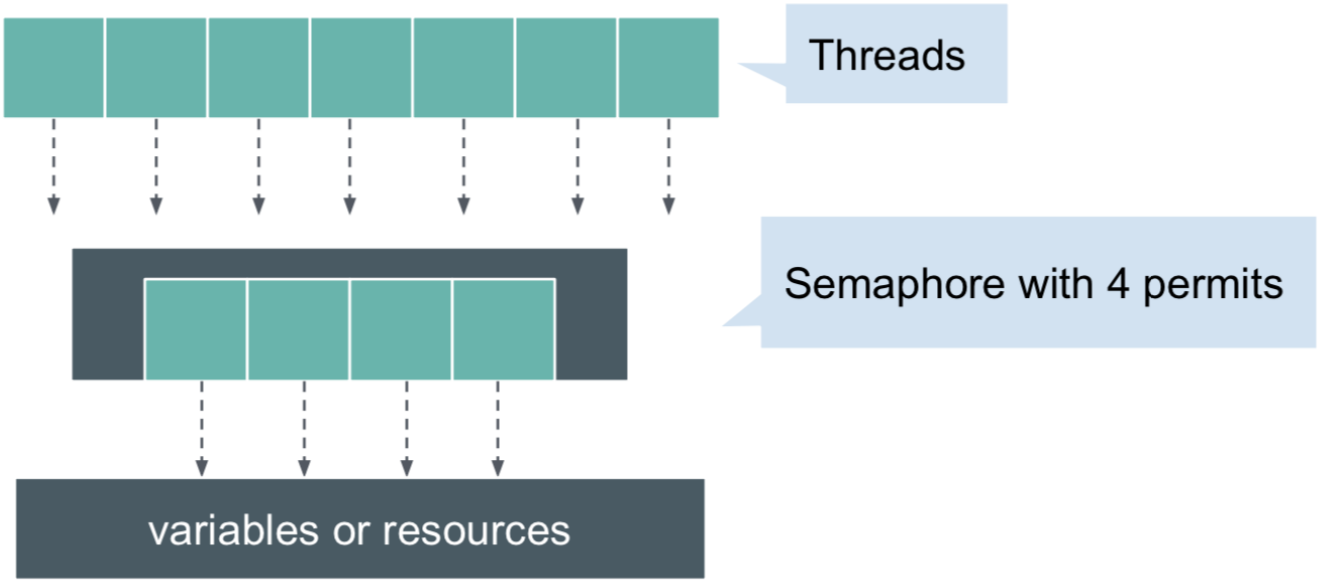
Feature	Shared Memory (e.g., lock, Monitor)	Message Passing (e.g., Channels)
Concept	Threads directly access shared data	Threads communicate via messages
Complexity	High (race conditions, deadlocks)	Lower (decoupled, safer)
Performance	Potentially faster (no data copy)	Overhead for data copying
Best For	Fine-grained control, large data, within process	Decoupling, async, producer-consumer

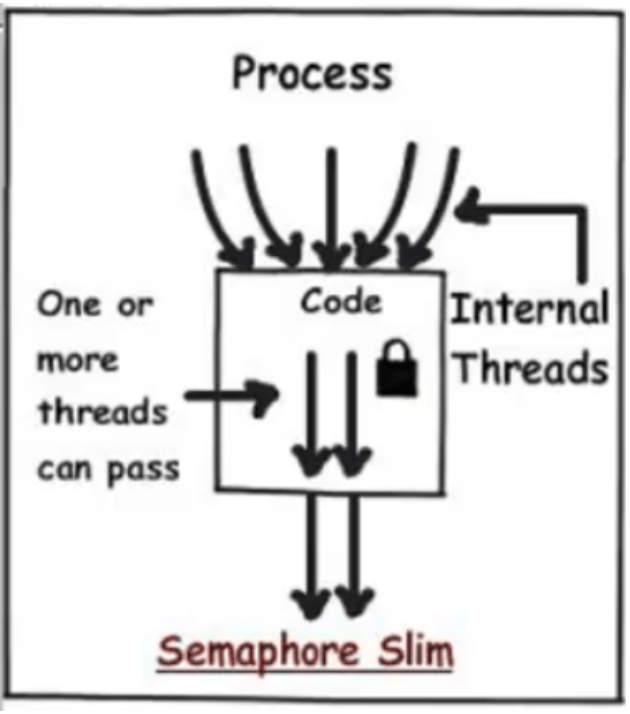
SEMAPHORE - LIMITING ACCESS

Limit Concurrent Access to Resources

- **Semaphore**: System-wide (cross-process) semaphore.
- **SemaphoreSlim**: Lighter, in-process semaphore (supports async).

- **Use Cases**:
- Database connection pooling.
- Limiting concurrent calls to external APIs.
- Controlling access to a fixed number of resources.


Threads
Semaphore with 4 permits
variables or resources


Process
One or more threads can pass
Code
Internal Threads
Semaphore Slim

SQL TRANSACTION ISOLATION

ACID Properties

- Atomicity
- Consistency
- Isolation
- Durability

Isolation Levels:

- ReadUncommitted
- Read Committed
- Repeatable Read
- Serializable

READ PHENOMENA

- -- Dirty Read
- T1: UPDATE Account SET Balance = 100
- T2: SELECT Balance -- Sees 100
- T1: ROLLBACK -- Oops!

- -- Non-Repeatable Read
- T1: SELECT Balance -- 50
- T2: UPDATE Balance = 100
- T1: SELECT Balance -- 100 (different!)

- -- Phantom Read
- T1: SELECT COUNT(*) WHERE Age > 25 -- 10
- T2: INSERT Person (Age = 30)
- T1: SELECT COUNT(*) WHERE Age > 25 -- 11

Lock Types:
- S - Shared (SELECT)
- X- Exclusive (UPDATE/DELETE)
- U - Update (SELECT FORUPDATE)
- I* - Intent (IS, IX, IU)

THANK YOU!