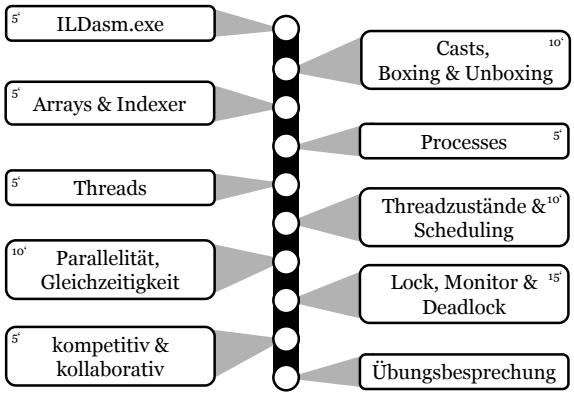
## Heute tun wir viel

23. Mai 2003




## ILDasm.exe

*Intermediate Language Dis-Assembler*

test.cs

```
using System;
class Test {
      static void Main() {
            Console.WriteLine(„Hallo!"); }}
```

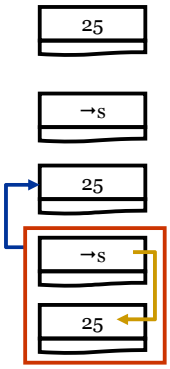
```
csc test.cs
ildasm test.exe
```

```
.maxstack 8
IL_0000: ldarg.0
IL_0001: call instance void ↵
[mscorlib]System.Object::.ctor()
IL_0006: ret
```

```
.maxstack 8
IL_0000: ldarg.0
IL_0005: call instance void ↵
[mscorlib]System.Console::WriteLine(class System.String)
IL_000a: ret
```

## Casts, Boxing & Unboxing

```
// Wertetyp
short s = 25;

// Boxing
// Werttyp in Objektverweis ändern
object objShort = s;

// Unboxing
// Objektverweis zurück in Wertetyp
short t = (short)objShort;

// ???
string str = (string)objShort;

// inkompatibler Cast
}
catch (InvalidCastException e) {
// ...
}
```



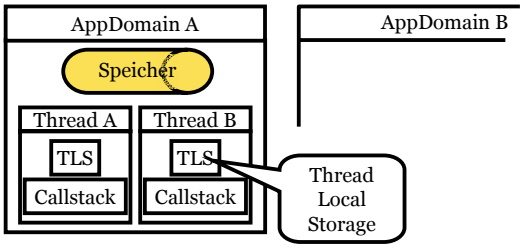
## Arrays & Indexer

```
// normaler Array
Auto[] autos = new Auto[5];
// Array füllen
autos[0] = new Auto(„Mercedes“);
// ...
// Array durchlaufen
foreach ( Auto a in autos ) {
 // ...
}

// Indexer
class Autos {
 private Auto[] _auto;
 public Auto this[ int i ] {
  get { /* ... */ } set { /* ... */ }
 }
 public void add( Auto a ) { /* ... */ }
}
```

## Processes

*Schwergewichtsprozesse* mit eigenem Adressraum

ein .NET-Process besteht aus einem oder mehreren **`System.AppDomain`**



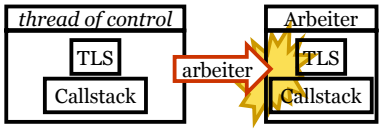

## `System.AppDomain`

```
// alle Assemblies im aktuellen AppDomain
AppDomain ad = AppDomain.CurrentDomain;

Assembly[] as = ad.GetAssemblies();

foreach ( Assembly a in as ) {
 Console.WriteLine( a.FullName );
}

// Assemblies dazuladen, ausführen und entladen
ad.Load(„meinAssembly“);
ad.ExecuteAssembly(„meinAssembly“)
ad.Unload(„meinAssembly“);
```

## System.Thread

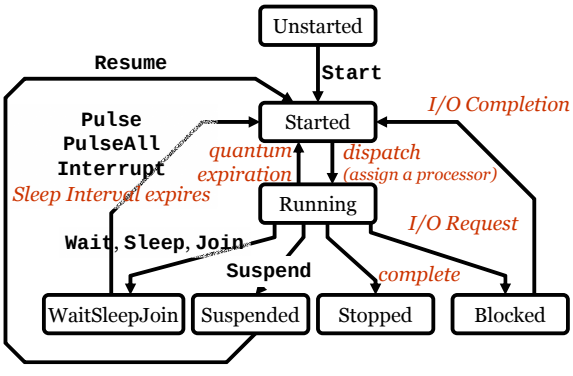
```
// welcher Thread läuft gerade?
Console.WriteLine(
 Thread.CurrentThread.GetHashCode();
);

// wo läuft dieser Thread?
DomainApp da = Thread.getDomain()

// setzt aktuelle Thread aus
Thread.Sleep();
```

## eine Instanz von System.Thread



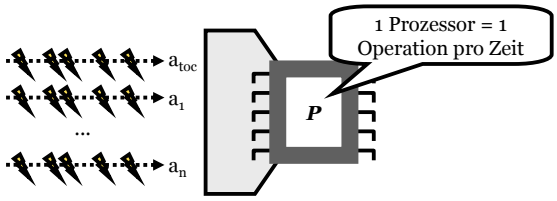

```
// Hashcode des Thread of control
Console.WriteLine(
 Thread.CurrentThread.GetHashCode();
);

// Thread erstellen
Thread arbeiter = new Thread(new ThreadStart(eineMethode))

// Thread starten
Thread.Start();
```

## Zustände eines Threads



## Scheduling

wer entscheidet, wann & wie lange welcher Thread läuft?

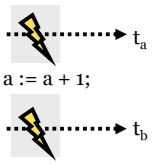


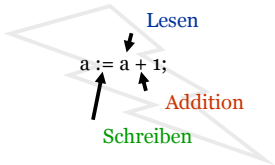

- Prozessor wird in Zeitquantum (time slices) aufgeteilt
- jeder Thread erhält *ab & zu* ein solches Quantum


## Parallelität, Gleichzeitigkeit

was bedeutet „gleichzeitig“ ausgeführt?

2 Threads – 1 Prozessor




→ Lesen, Schreiben, Addition & Subtraktion sind **atomar**, können nicht unterbrochen werden

gleichzeitig!

ein Szenario aus $\binom{6}{3}$ möglichen:


## Beispiel zur Parallelität

```
// Methode, die in Threads abgearbeitet wird
void arbeit() {
 for( int i = 0; i < 5; i++ ) {
  Console.Write ( „hey: {0}“, i );
 }
}
// ...
// 3 Threads laufen lassen
Thread a = new Thread(new ThreadStart(arbeit));
Thread b = new Thread(new ThreadStart(arbeit));
Thread c = new Thread(new ThreadStart(arbeit));
a.Start(); b.Start(); c.Start();
```

```
hey: 0 hey: 1 hey: 2 hey: hey: hey: 0
hey: 0 hey: 1 hey: 2 hey: 2 hey: 3
hey: 3 hey: 4 hey: 4 hey: 3 hey: 4
```

Ausgabe ist ziemlich ungeordnet, weil sich die Threads rivalisieren um gemeinsame Ressource `arbeit` → **kompetitiv**

## Synchronisation mit Lock

```
// die Methode arbeit muss also gesperrt sein, bis
// ein Thread abgearbeitet ist
void arbeit() {
 lock(this);
 for( int i = 0; i < 5; i++ ) {
  Console.Write ( „hey: {0}“, i );
 }
}
```

alle warten hier, bis `this` (Instanz, welche Methode `arbeit` enthält) fertig ist

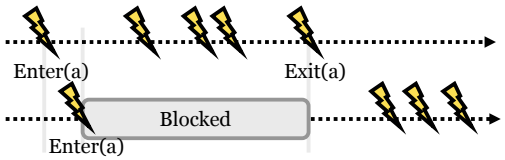
```
// ...
// 3 Threads laufen lassen
Thread a = new Thread(new ThreadStart(arbeit));
Thread b = new Thread(new ThreadStart(arbeit));
Thread c = new Thread(new ThreadStart(arbeit));
a.Start(); b.Start(); c.Start();
```

```
hey: 0 hey: 1 hey: 2 hey: 3 hey: 4
hey: 0 hey: 1 hey: 2 hey: 3 hey: 4
hey: 0 hey: 1 hey: 2 hey: 3 hey: 4
```

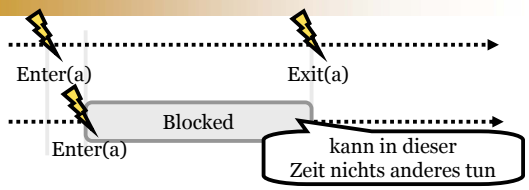
## Was ist lock?

```
// lock ist so etwas wie ein generischer Befehl
// und sieht ausgeschrieben folgendermassen aus
void arbeit() {
 Monitor.Enter(this); try {
  for( int i = 0; i < 5; i++ ) {
   Console.Write ( „hey: {0}“, i );
  }
 } finally { Monitor.Exit(this); }
}
```

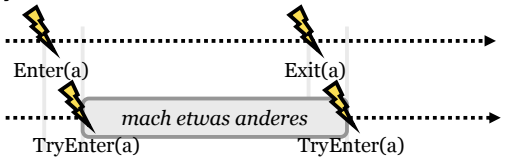
`Monitor.Enter( object ressource )` sperrt die Ressource bis `Monitor.Exit( object ressource )` ausgeführt wird




## Um nicht alles zu blockieren

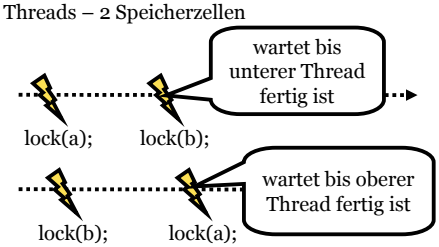



```
if ( !Monitor.TryEnter( a ) ) {
 // mach etwas anderes
}
```



## Deadlock

2 Threads – 2 Speicherzellen




- die beiden Threads können, weil sie egoistisch sind nie weiterfahren
- der Scheduler kann diese Situation nicht voraussehen
- Threads müssen **kollaborieren** / zusammen arbeiten

## Kollaboration

Threads geben Ressource freiwillig ab