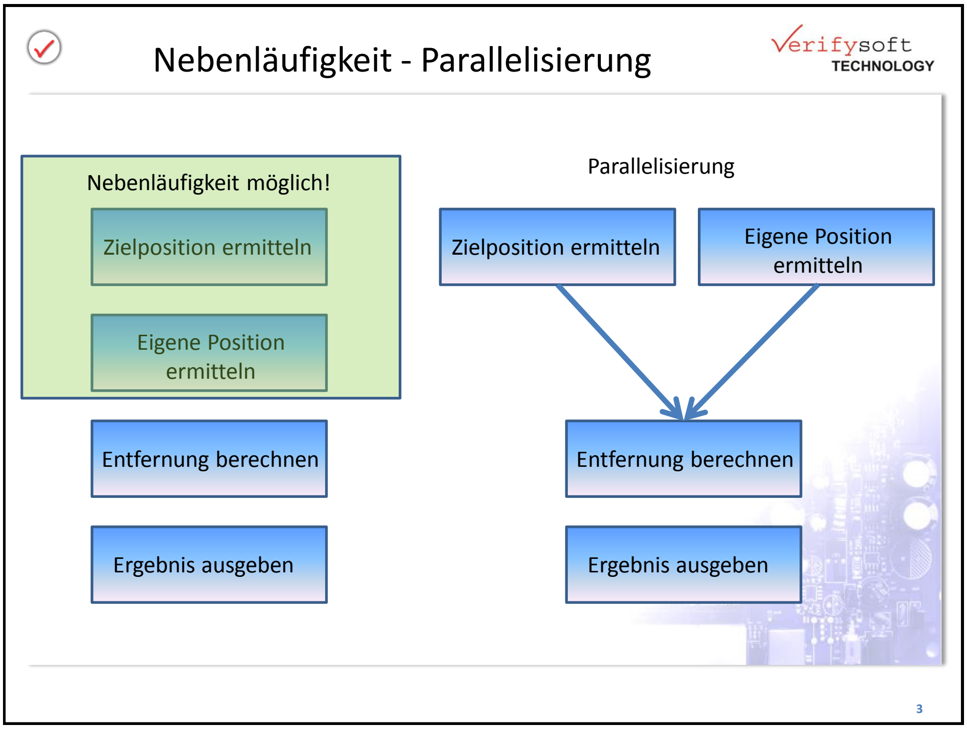
Auffinden von Nebenläufigkeitsfehlern durch statische Codeanalyse

Royd Lüdtke
Verifysoft Technology GmbH
luedtke@verifysoft.com
+49 781 127 8118-8

Agenda

- Was ist Nebenläufigkeit?
- Was ist Parallelisierung?
- Zuweisung von CPU-Zeit durch den Scheduler
- Motivation zur Implementierung von Nebenläufigkeit?
- Was sind Nebenläufigkeitsfehler?
- Warum sind Nebenläufigkeitsfehler so gefürchtet?
- Beispiel Data Race
- Beispiel Dead Lock (Nested Lock)
- Wie kann die statische Codeanalyse helfen?



Nebenläufigkeit - Parallelisierung

Nebenläufigkeit möglich!

Zielposition ermitteln

Eigene Position ermitteln

Entfernung berechnen

Ergebnis ausgeben

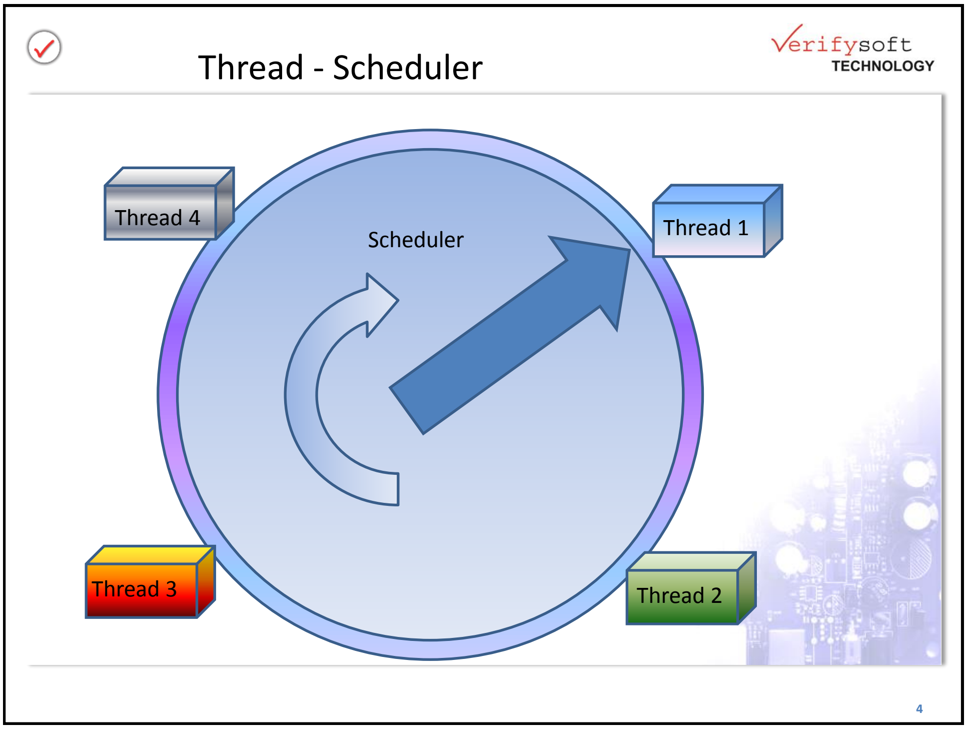
Parallelisierung

Zielposition ermitteln

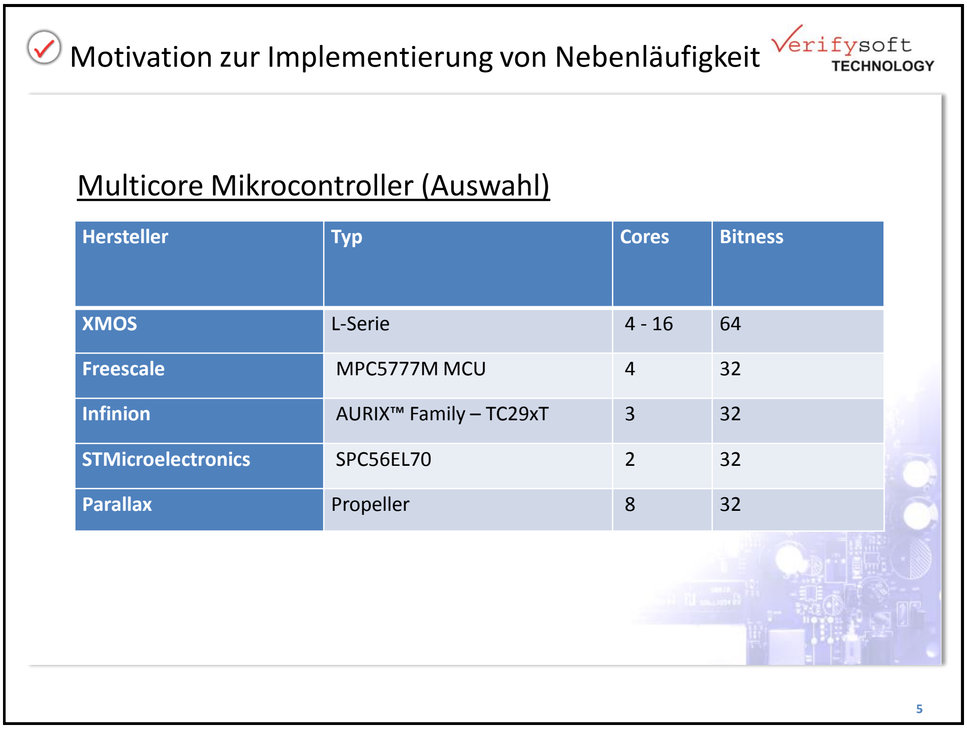
Eigene Position ermitteln

Entfernung berechnen

Ergebnis ausgeben



Thread - Scheduler



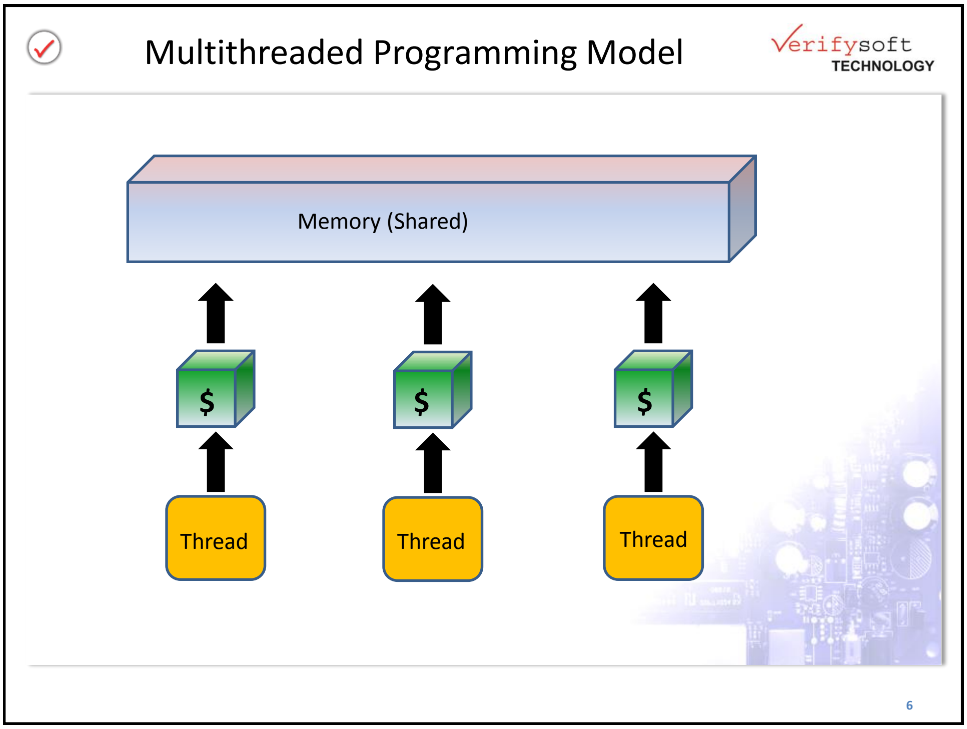
Motivation zur Implementierung von Nebenläufigkeit

Verifysoft TECHNOLOGY

Multicore Mikrocontroller (Auswahl)

Hersteller	Typ	Cores	Bitness
XMOS	L-Serie	4 - 16	64
Freescale	MPC5777M MCU	4	32
Infinion	AURIX™ Family – TC29xT	3	32
STMicroelectronics	SPC56EL70	2	32
Parallax	Propeller	8	32

5



Multithreaded Programming Model

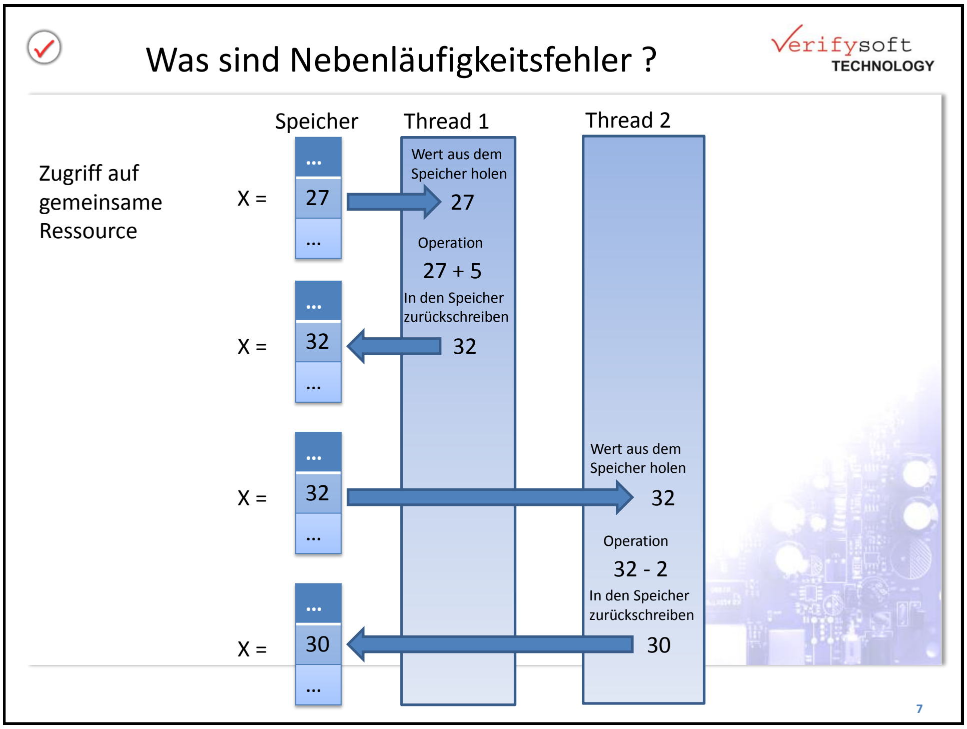
Verifysoft TECHNOLOGY

Memory (Shared)

$ $ $

Thread Thread Thread

6



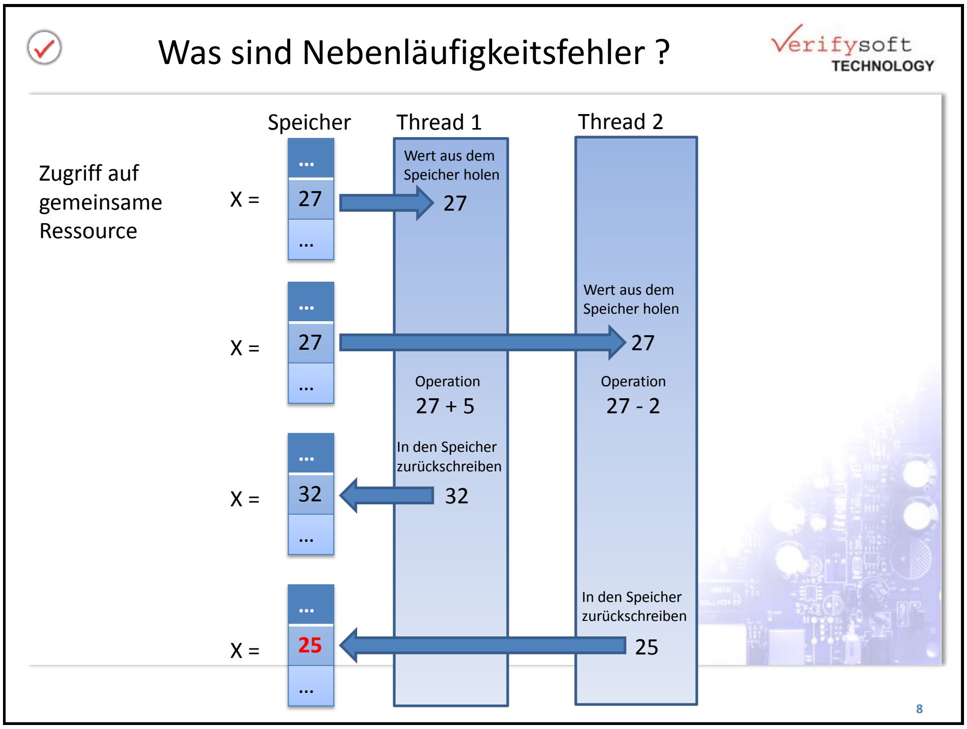
Was sind Nebenläufigkeitsfehler ?

Verifysoft TECHNOLOGY

Zugriff auf gemeinsame Ressource

Speicher	Thread 1	Thread 2
X = … / 27 / …	Wert aus dem Speicher holen 27	
	Operation 27 + 5	
X = … / 32 / …	In den Speicher zurückschreiben 32	
X = … / 32 / …		Wert aus dem Speicher holen 32
		Operation 32 - 2
X = … / 30 / …		In den Speicher zurückschreiben 30

7



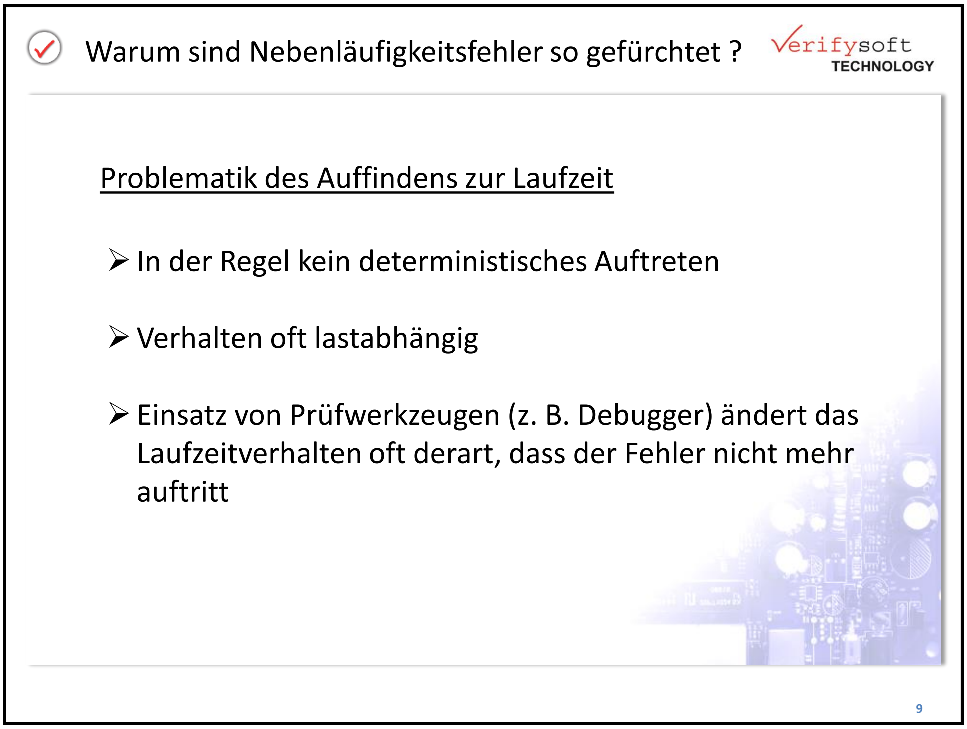
Was sind Nebenläufigkeitsfehler ?

Verifysoft TECHNOLOGY

Zugriff auf gemeinsame Ressource

Speicher	Thread 1	Thread 2
X = … / 27 / …	Wert aus dem Speicher holen 27	
X = … / 27 / …		Wert aus dem Speicher holen 27
	Operation 27 + 5	Operation 27 - 2
X = … / 32 / …	In den Speicher zurückschreiben 32	
X = … / 25 / …		In den Speicher zurückschreiben 25

8



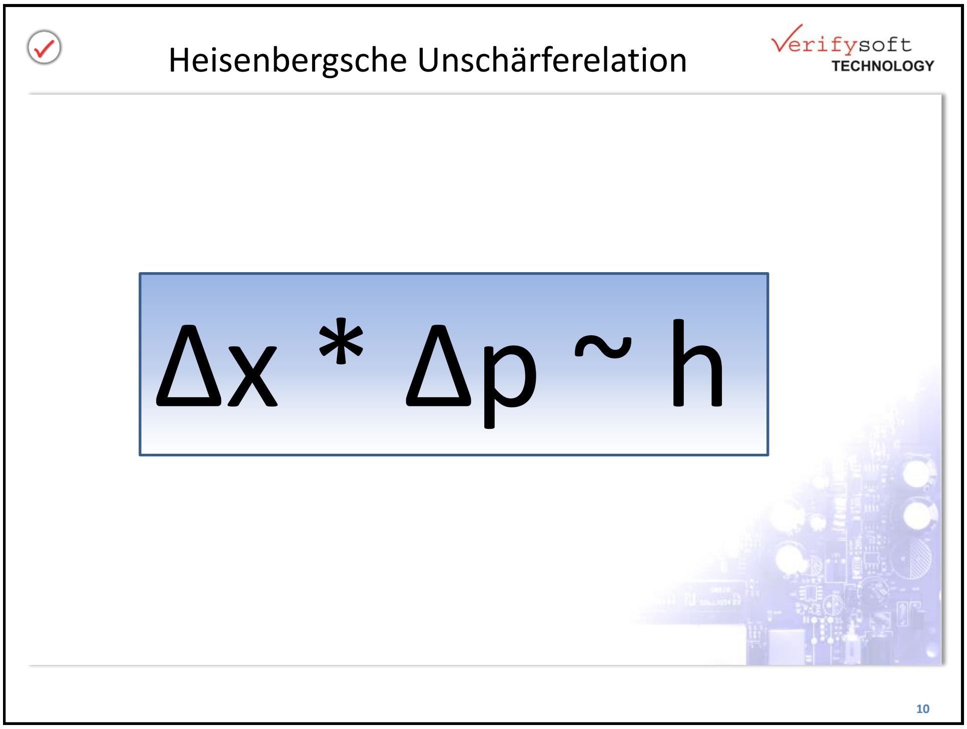
Warum sind Nebenläufigkeitsfehler so gefürchtet ?

Verifysoft
TECHNOLOGY

Problematik des Auffindens zur Laufzeit

- In der Regel kein deterministisches Auftreten
- Verhalten oft lastabhängig
- Einsatz von Prüfwerkzeugen (z. B. Debugger) ändert das Laufzeitverhalten oft derart, dass der Fehler nicht mehr auftritt

9



Heisenbergsche Unschärferelation

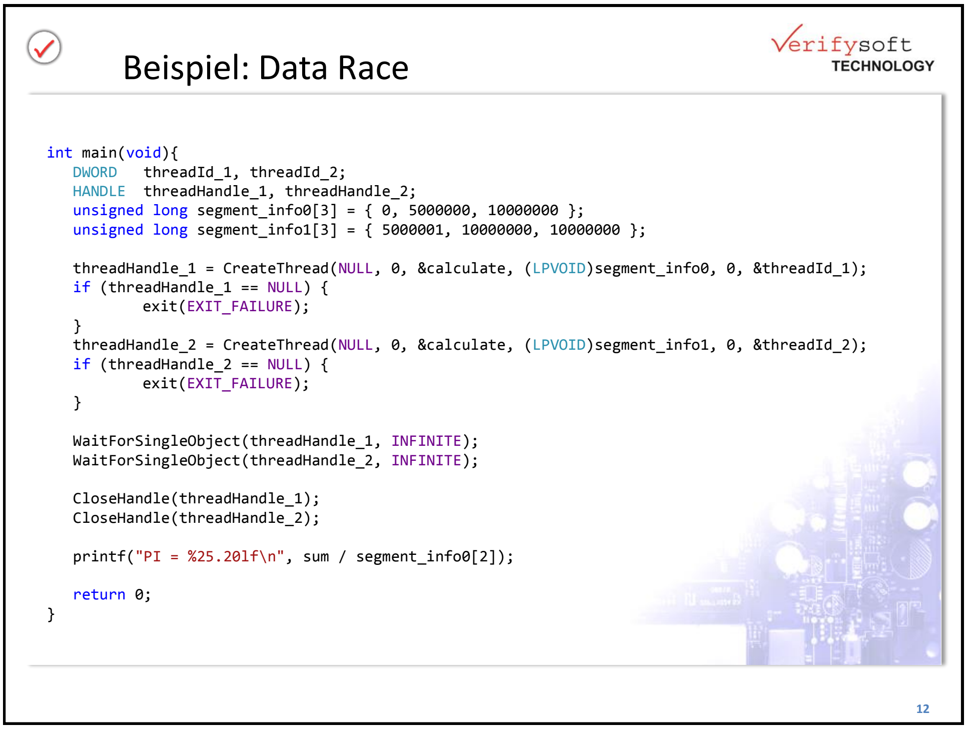
Verifysoft
TECHNOLOGY

$$\Delta x * \Delta p \sim h$$

10

Nebenläufigkeitsfehler
sind hinterlistig
und gemein!

Quelle: www.fotolia.de

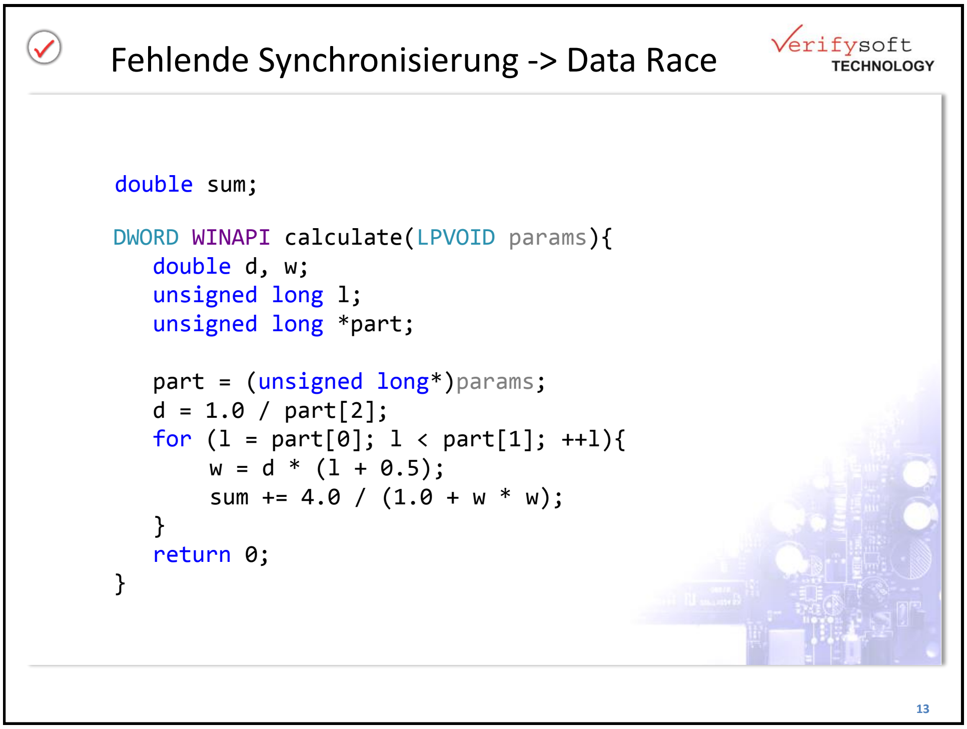


Beispiel: Data Race

Verifysoft TECHNOLOGY

```
int main(void){
   DWORD   threadId_1, threadId_2;
   HANDLE  threadHandle_1, threadHandle_2;
   unsigned long segment_info0[3] = { 0, 5000000, 10000000 };
   unsigned long segment_info1[3] = { 5000001, 10000000, 10000000 };

   threadHandle_1 = CreateThread(NULL, 0, &calculate, (LPVOID)segment_info0, 0, &threadId_1);
   if (threadHandle_1 == NULL) {
           exit(EXIT_FAILURE);
   }
   threadHandle_2 = CreateThread(NULL, 0, &calculate, (LPVOID)segment_info1, 0, &threadId_2);
   if (threadHandle_2 == NULL) {
           exit(EXIT_FAILURE);
   }

   WaitForSingleObject(threadHandle_1, INFINITE);
   WaitForSingleObject(threadHandle_2, INFINITE);

   CloseHandle(threadHandle_1);
   CloseHandle(threadHandle_2);

   printf("PI = %25.20lf\n", sum / segment_info0[2]);

   return 0;
}
```

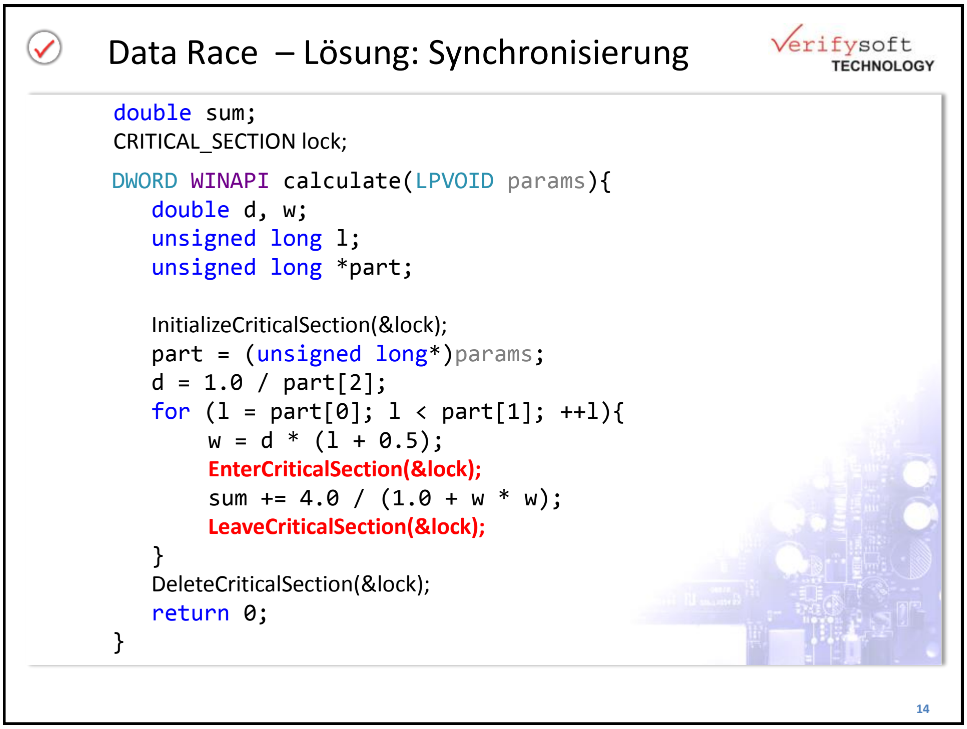
12



Fehlende Synchronisierung -> Data Race

Verifysoft TECHNOLOGY

```
double sum;

DWORD WINAPI calculate(LPVOID params){
   double d, w;
   unsigned long l;
   unsigned long *part;

   part = (unsigned long*)params;
   d = 1.0 / part[2];
   for (l = part[0]; l < part[1]; ++l){
       w = d * (l + 0.5);
       sum += 4.0 / (1.0 + w * w);
   }
   return 0;
}
```

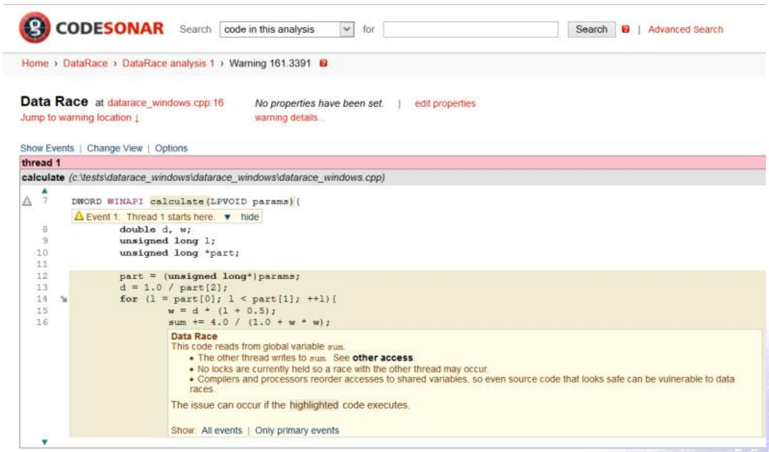


Data Race – Lösung: Synchronisierung

Verifysoft TECHNOLOGY

```
double sum;
CRITICAL_SECTION lock;
DWORD WINAPI calculate(LPVOID params){
   double d, w;
   unsigned long l;
   unsigned long *part;

   InitializeCriticalSection(&lock);
   part = (unsigned long*)params;
   d = 1.0 / part[2];
   for (l = part[0]; l < part[1]; ++l){
       w = d * (l + 0.5);
       EnterCriticalSection(&lock);
       sum += 4.0 / (1.0 + w * w);
       LeaveCriticalSection(&lock);
   }
   DeleteCriticalSection(&lock);
   return 0;
}
```

Aufdeckung durch statische Codeanalyse



CODESONAR Search code in this analysis for Search | Advanced Search

Home › DataRace › DataRace analysis 1 › Warning 161.3391

Data Race at datarace_windows.cpp:16 *No properties have been set.* | edit properties
Jump to warning location ↓ warning details...

Show Events | Change View | Options

thread 1

calculate *(c:\tests\datarace_windows\datarace_windows\datarace_windows.cpp)*

```
 7   DWORD WINAPI calculate(LPVOID params){
     ⚠ Event 1:  Thread 1 starts here.  ▼  hide
 8           double d, w;
 9           unsigned long l;
10           unsigned long *part;
11
12           part = (unsigned long*)params;
13           d = 1.0 / part[2];
14           for (l = part[0]; l < part[1]; ++l){
15                   w = d * (l + 0.5);
16                   sum += 4.0 / (1.0 + w * w);
```

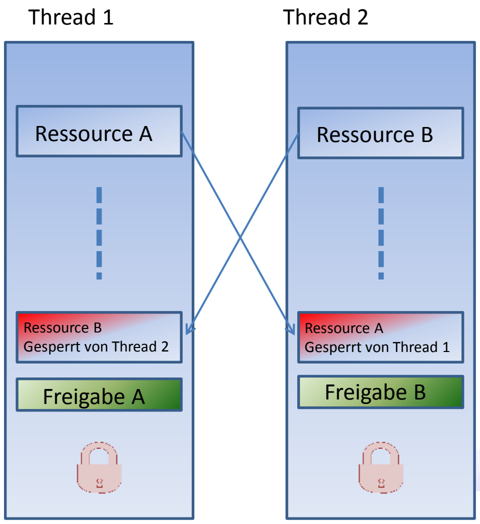
Data Race
This code reads from global variable `sum`.

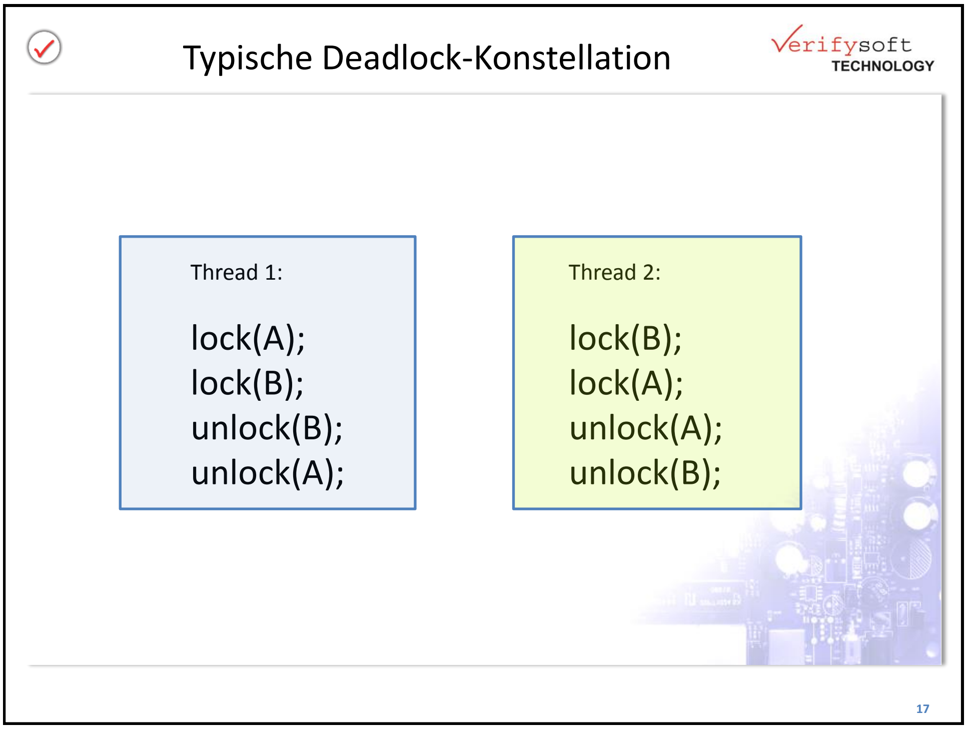
- The other thread writes to `sum`. See **other access**.
- No locks are currently held so a race with the other thread may occur.
- Compilers and processors reorder accesses to shared variables, so even source code that looks safe can be vulnerable to data races.

The issue can occur if the highlighted code executes.

Show: All events | Only primary events

Deadlock-Situation





Typische Deadlock-Konstellation

Verifysoft TECHNOLOGY

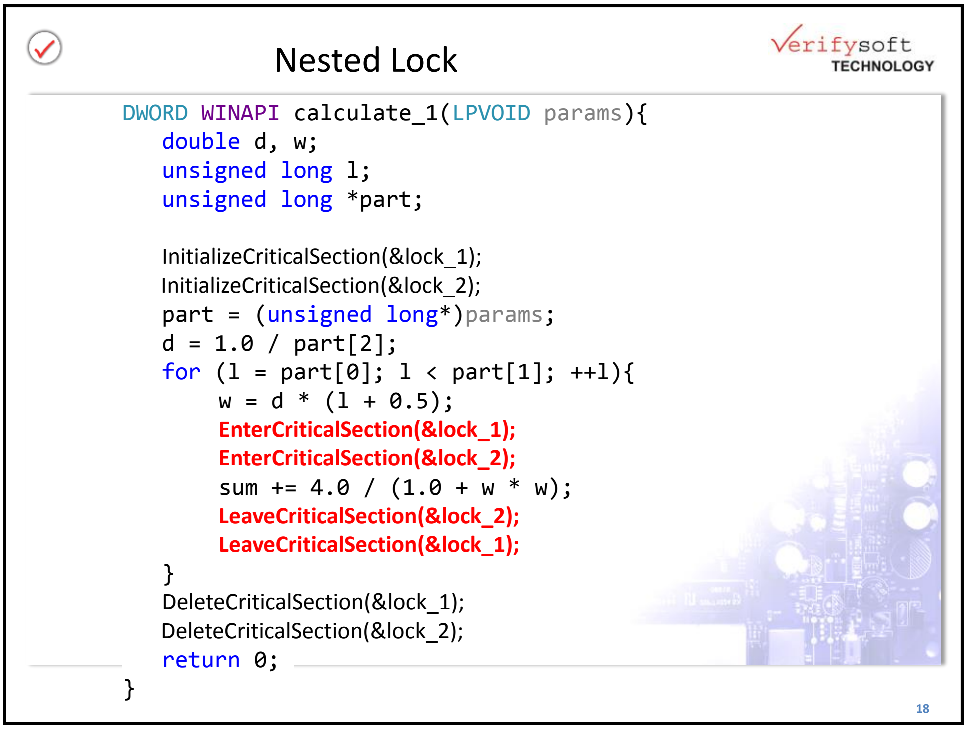
Thread 1:

```
lock(A);
lock(B);
unlock(B);
unlock(A);
```

Thread 2:

```
lock(B);
lock(A);
unlock(A);
unlock(B);
```

17



Nested Lock

Verifysoft TECHNOLOGY

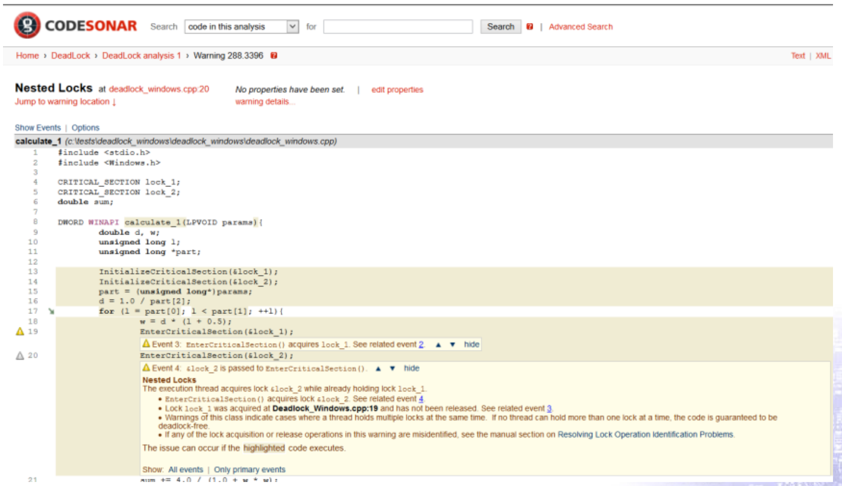
```
DWORD WINAPI calculate_1(LPVOID params){
   double d, w;
   unsigned long l;
   unsigned long *part;

   InitializeCriticalSection(&lock_1);
   InitializeCriticalSection(&lock_2);
   part = (unsigned long*)params;
   d = 1.0 / part[2];
   for (l = part[0]; l < part[1]; ++l){
       w = d * (l + 0.5);
       EnterCriticalSection(&lock_1);
       EnterCriticalSection(&lock_2);
       sum += 4.0 / (1.0 + w * w);
       LeaveCriticalSection(&lock_2);
       LeaveCriticalSection(&lock_1);
   }
   DeleteCriticalSection(&lock_1);
   DeleteCriticalSection(&lock_2);
   return 0;
}
```

18

Nested Lock

```
DWORD WINAPI calculate_2(LPVOID params){
   double d, w;
   unsigned long l;
   unsigned long *part;

   InitializeCriticalSection(&lock_1);
   InitializeCriticalSection(&lock_2);
   part = (unsigned long*)params;
   d = 1.0 / part[2];
   for (l = part[0]; l < part[1]; ++l){
       w = d * (l + 0.5);
       EnterCriticalSection(&lock_2);
       EnterCriticalSection(&lock_1);
       sum += 4.0 / (1.0 + w * w);
       LeaveCriticalSection(&lock_1);
       LeaveCriticalSection(&lock_2);
   }
   DeleteCriticalSection(&lock_1);
   DeleteCriticalSection(&lock_2);
   return 0;
}
```

Aufdeckung durch statische Codeanalyse



Nested Locks at deadlock_windows.cpp:20
Jump to warning location ↓

No properties have been set. | edit properties
warning details...

Show Events | Options

calculate_1 *(c:\tests\deadlock_windows\deadlock_windows\deadlock_windows.cpp)*

```
 1   #include <stdio.h>
 2   #include <Windows.h>
 3
 4   CRITICAL_SECTION lock_1;
 5   CRITICAL_SECTION lock_2;
 6   double sum;
 7
 8   DWORD WINAPI calculate_1(LPVOID params){
 9           double d, w;
10           unsigned long l;
11           unsigned long *part;
12
13           InitializeCriticalSection(&lock_1);
14           InitializeCriticalSection(&lock_2);
15           part = (unsigned long*)params;
16           d = 1.0 / part[2];
17           for (l = part[0]; l < part[1]; ++l){
18                   w = d * (l + 0.5);
⚠ 19                 EnterCriticalSection(&lock_1);
```

⚠ Event 3: EnterCriticalSection() acquires lock_1. See related event 2. ▲ ▼ hide

```
⚠ 20                 EnterCriticalSection(&lock_2);
```

⚠ Event 4: &lock_2 is passed to EnterCriticalSection(). ▲ ▼ hide

Nested Locks
The execution thread acquires lock &lock_2 while already holding lock lock_1.
- EnterCriticalSection() acquires lock &lock_2. See related event 4.
- Lock lock_1 was acquired at **Deadlock_Windows.cpp:19** and has not been released. See related event 3.
- Warnings of this class indicate cases where a thread holds multiple locks at the same time. If no thread can hold more than one lock at a time, the code is guaranteed to be deadlock-free.
- If any of the lock acquisition or release operations in this warning are misidentified, see the manual section on Resolving Lock Operation Identification Problems.

The issue can occur if the highlighted code executes.

Show: All events | Only primary events

```
21                   sum += 4.0 / (1.0 + w * w);
```

Verifysoft
TECHNOLOGY

VIELEN DANK !

Royd Lüdtke
Verifysoft Technology GmbH
luedtke@verifysoft.com
+49 781 127 8118-8