Synchronisation nebenläufiger Threads

Dr. Andreas Schroeder

Überblick

Dieses Video behandelt

- Synchronisation
  - Interferenz und Speicherkonsistenz-Fehler
  - Synchronisation von Zugriffen
  - Geschützte Code-Bereiche

Interferenz

Thread A

A1
A2
A3

Thread B

B1
B2

Interleaving

B1	B1	A1	A1	A1
B2	A1	B1	A2	A2
A1	A2	A2	B1	A3
A2	A3	B2	A3	B1
A3	B2	A3	B2	B2

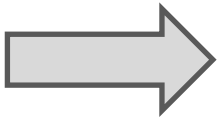
$$\binom{5}{2} = 10$$ Interleavings

Thread A

A1	`int x = v`
A2	`x = x + 1`
A3	`v = x`

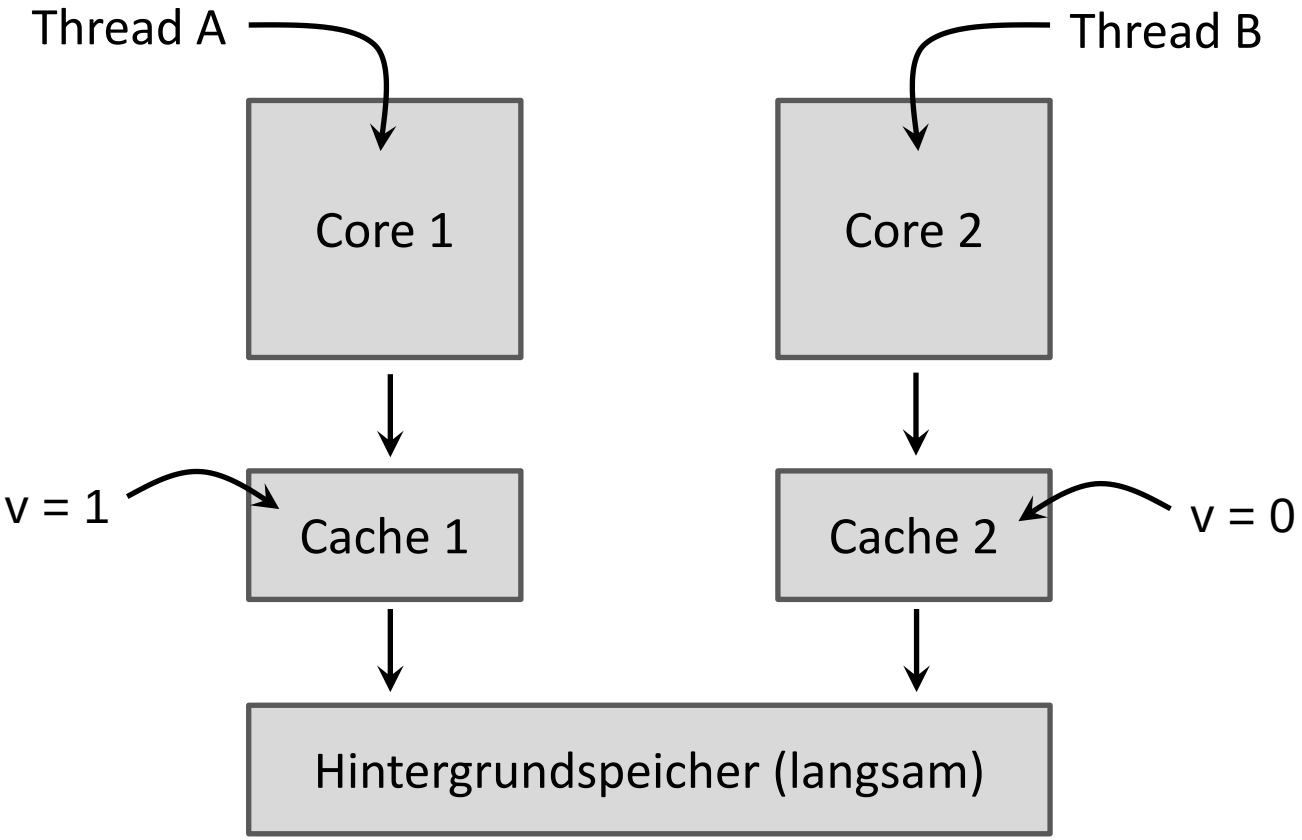
Thread B

B1	`int y = v`
B2	`v = y - 1`



			`int v = 0`
B1	A1	B1	`int y = 0`
B2	A2	A1	`int x = 0`
A1	A3	A2	`x = 0 + 1`
A2	B1	A3	`v = 1`
A3	B2	B2	`v = 0 - 1`
ok	ok	lost update	`v = -1`

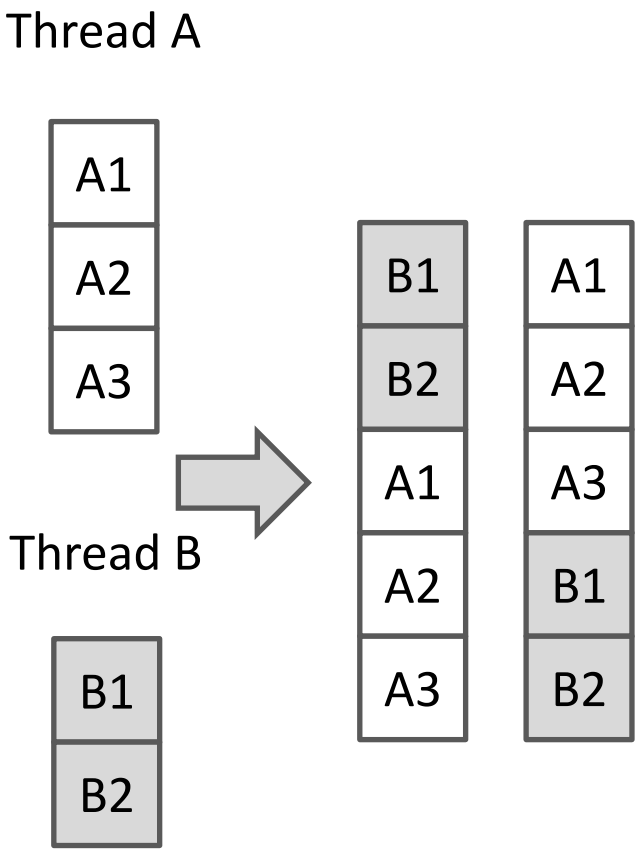
Speicherkonsistenz-Fehler



Synchronisierung

- Synchronized blocks, d.h. Ununterbrechbare Bereiche (engl. Atomic regions)
- Synchronization funktioniert über Lock-Objekte

  Wechselseitiger Ausschluss nur bei Synchronisation über gleiche Locks!



Guarded Blocks

- Oft muss ein Thread auf ein Ereignis oder eine Bedingung zu warten – dafür gibt es das Guarded Block-Muster
- Um zu warten:

```
synchronized(lock) {
        while(!condition) { lock.wait(); }
}
```

- Um zu benachrichtigen:

```
synchronized(lock) {
        condition= true;
        lock.notifyAll();
}
```

http://docs.oracle.com/javase/tutorial/essential/concurrency/guardmeth.html

Überblick

Was in diesem Video gezeigt wurde

- Synchronisation
  - Interferenz und Speicherkonsistenz-Fehler
  - Synchronisation von Zugriffen
  - Geschützte Code-Bereiche