

# **HeapManager oder multi-thread Garbage Collector**

**Vortrag von Thomas Weise**

**27.10.2003 15.30 1/336**

# Gliederung

- > **Ursprung und Idee**
- > **verwendete Datenstrukturen**
- > **Speichergrößeneinteilung mittels Hash**
- > **endgültige Freigabe**
- > **Tests und Interpretation**

**(14 Folien)**

# Ursprung

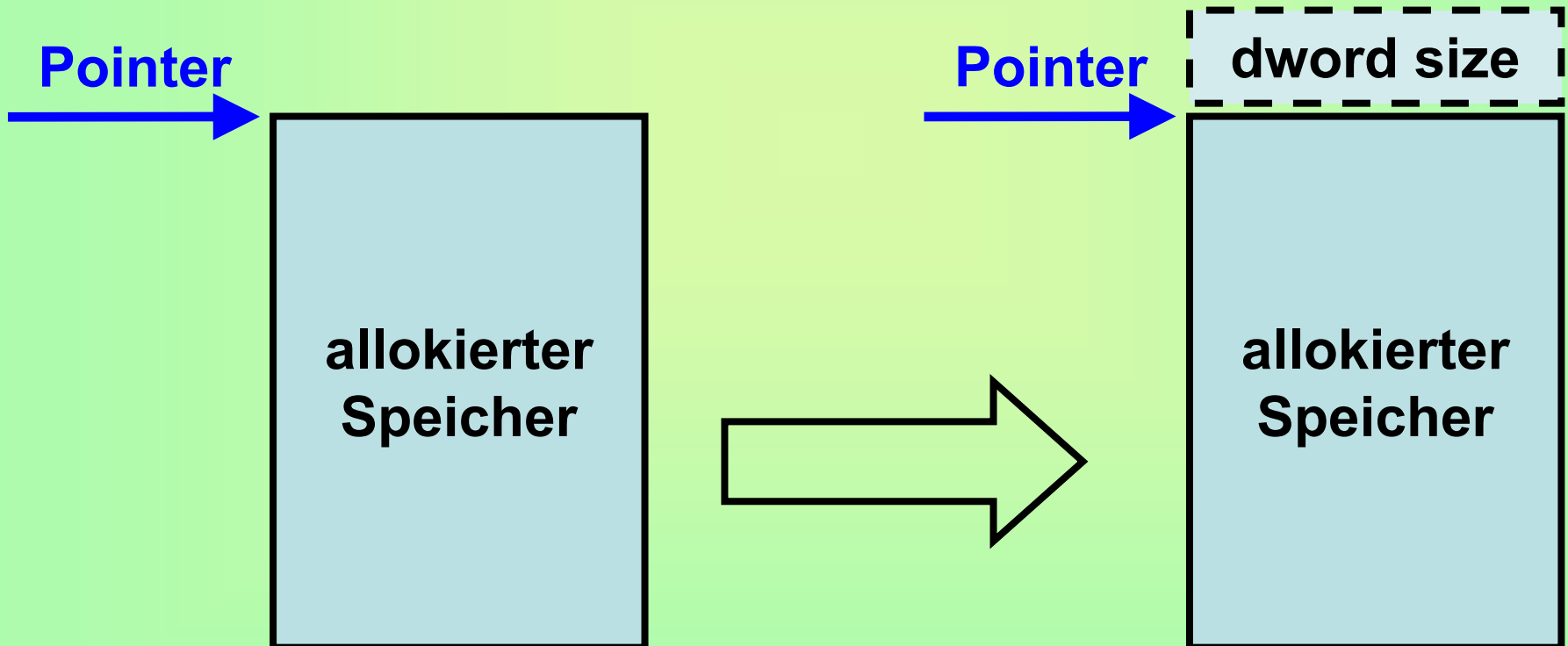
- > Fokus auf multi-thread Applikationen
- > Applikationen erzeugen/zerstören oft viele Speicherobjekte
- > häufig Allokation/Deallokation
- > ist es möglich, die Performance zu erhöhen?

# Idee

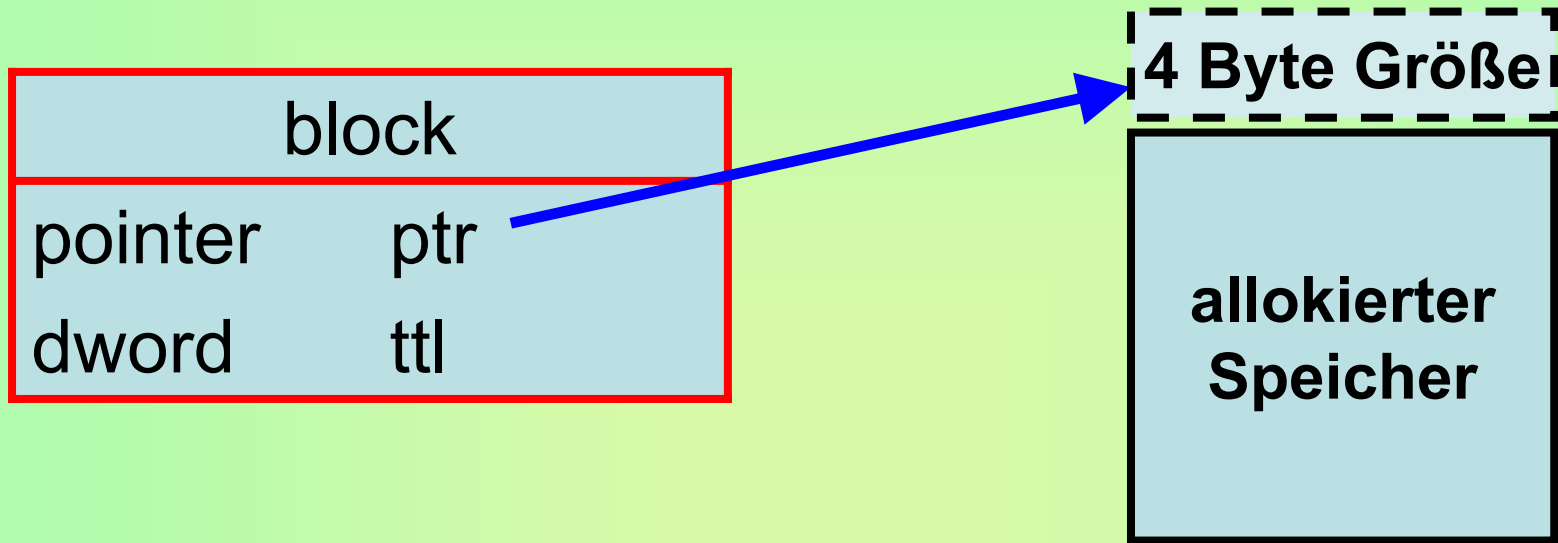
- > Speicherallokation durch „Memory Management Functions“, z.B. HeapAlloc
- > WinAPI zwar thread-sicher, verwendet jedoch „mutual exclusion“-Prinzip [MSDN](#)
- > Verwendung Garbage-Collector-Methode
- > Berücksichtigung von multi-thread-Aspekt, dadurch Beschleunigung
- > Klar: +Performance  $\Rightarrow$  +Speicherbelastung

# Datenstrukturen - pointer

Entwicklung effizienter Datenstrukturen zur Aufbewahrung von Speicher nach „Freigabe“



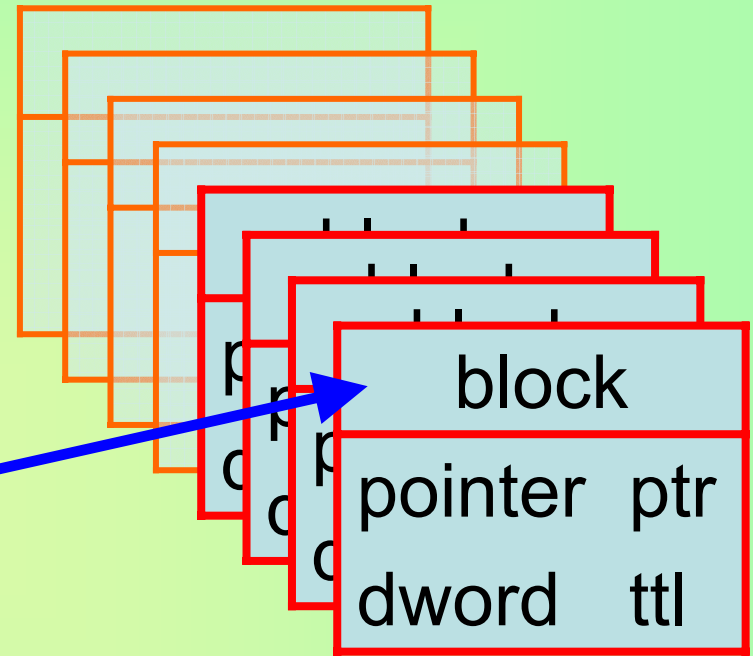
# Datenstrukturen - block



- > **Aufbewahrung einzelner Speicherblöcke**
- > **Lebensdauer bis endgültige Freigabe geregelt mit ttl**

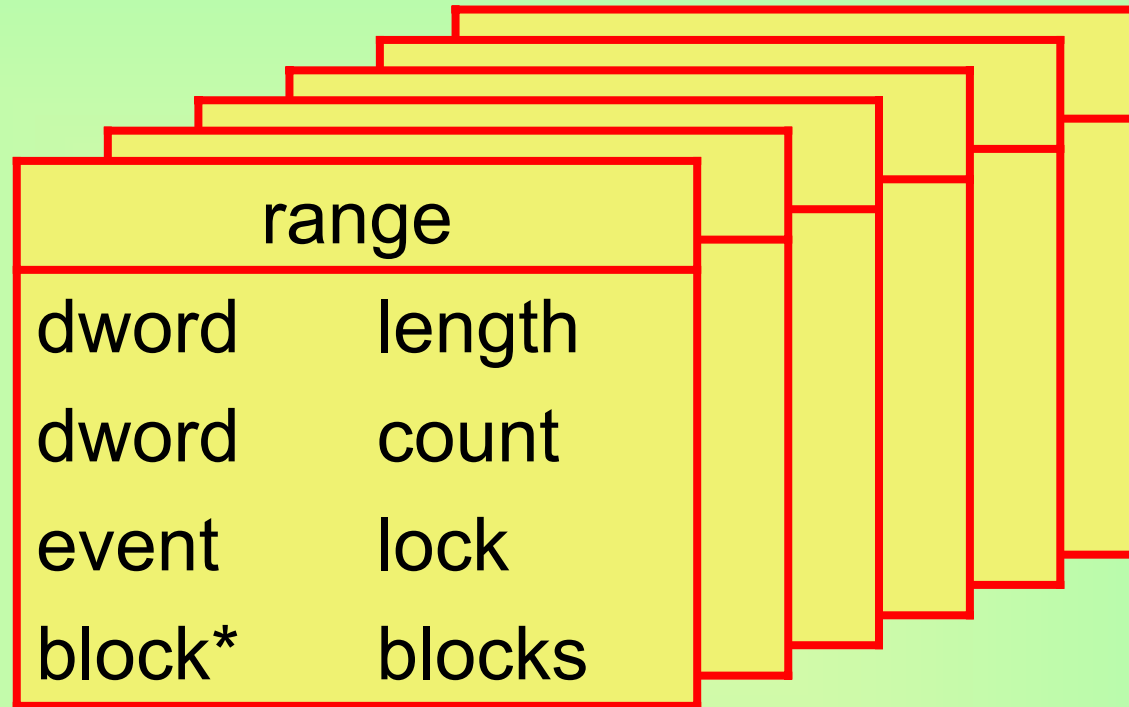
# Datenstrukturen - range

range	
dword	length
dword	count
event	lock
block*	blocks



- > **vakante Blöcke werden in Klassen unterteilt**
- > **jede range-Struktur verwaltet eine Klasse**
- > **wird über event lock synchronisiert, daher pro Struktur ein paralleler Zugriff möglich**

# Datenstruktur – range array



- > feste Anzahl range-Strukturen (array)
- > mit jeweils dynamisch vielen Blöcken
- > aber wie effizient organisieren?

# Hash

(...Buddy Verfahren...)

- > Organisation in einem Hash
- > Hash-Funktion realisiert exponentielles Wachstum
- > eigentliche Hash-Funktion ist nur zwei Instruktionen lang

```
__forceinline c_dword classify(c_dword asize)
{ __asm {
    bsr eax, asize
#if MIN_MEM > 0
    sub eax, MIN_MEM
#endif
} }
```

# Hash - Vertiefung

...
...
64-127
128-255
256-511
512-1023
1024-2047
2047-4095
...
...



- > alle Speicherblöcke mit gleichem MSB sind in einer Klasse
- > bei Allokierung wird *erstbester* Block mit  $\text{Größe} \geq \text{gewünscht}$  zurückgegeben
- > wird kein passender gefunden, wird auf HeapAlloc ausgewichen

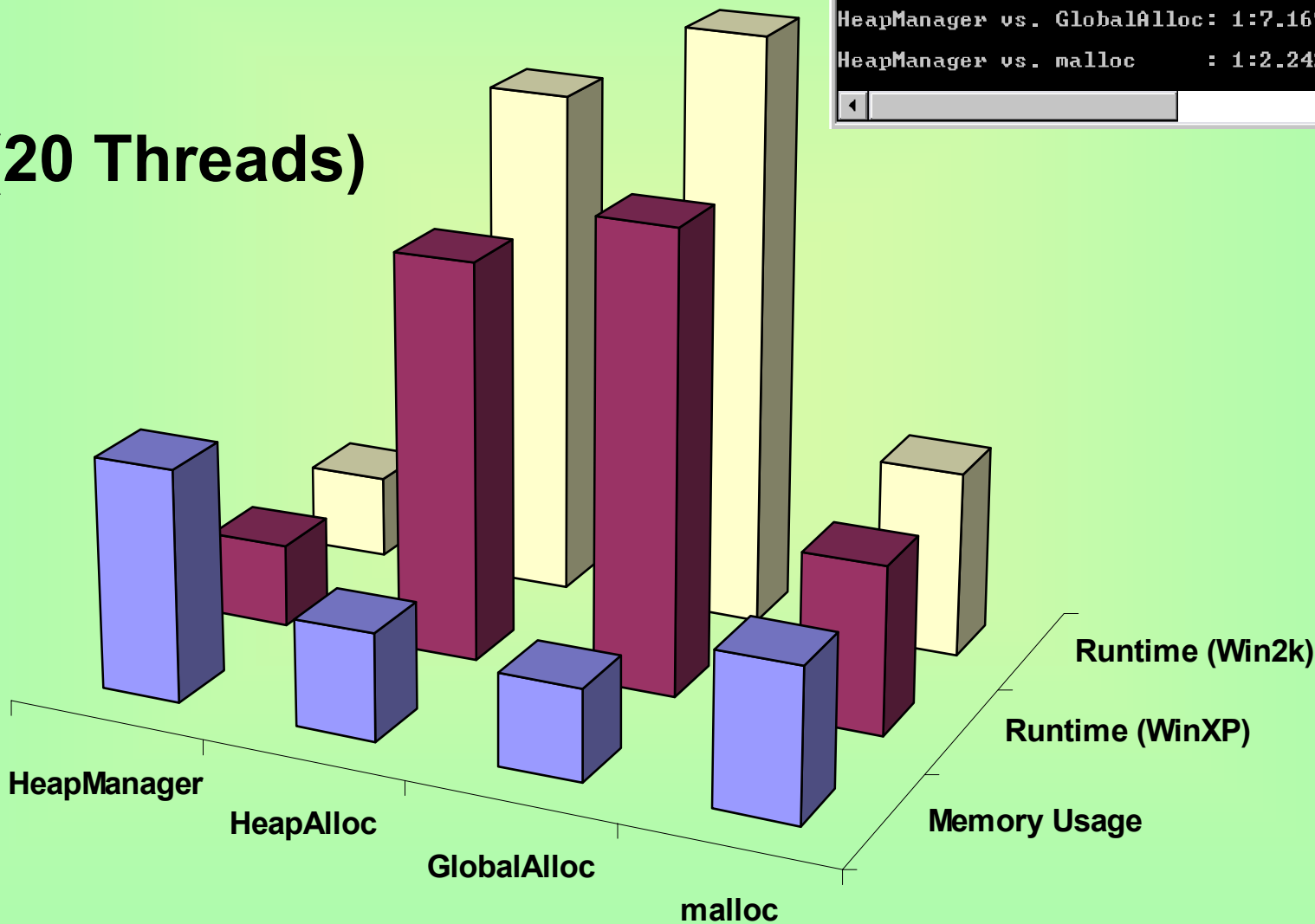
# Aufräumen

- > Cleaner-Thread dekrementiert nach fester Zeiteinheit (2 s) die ttl der vakanten Blöcke
- > wird ttl eines Blockes 0, wird er endgültig freigegeben

[Source](#)

# Tests

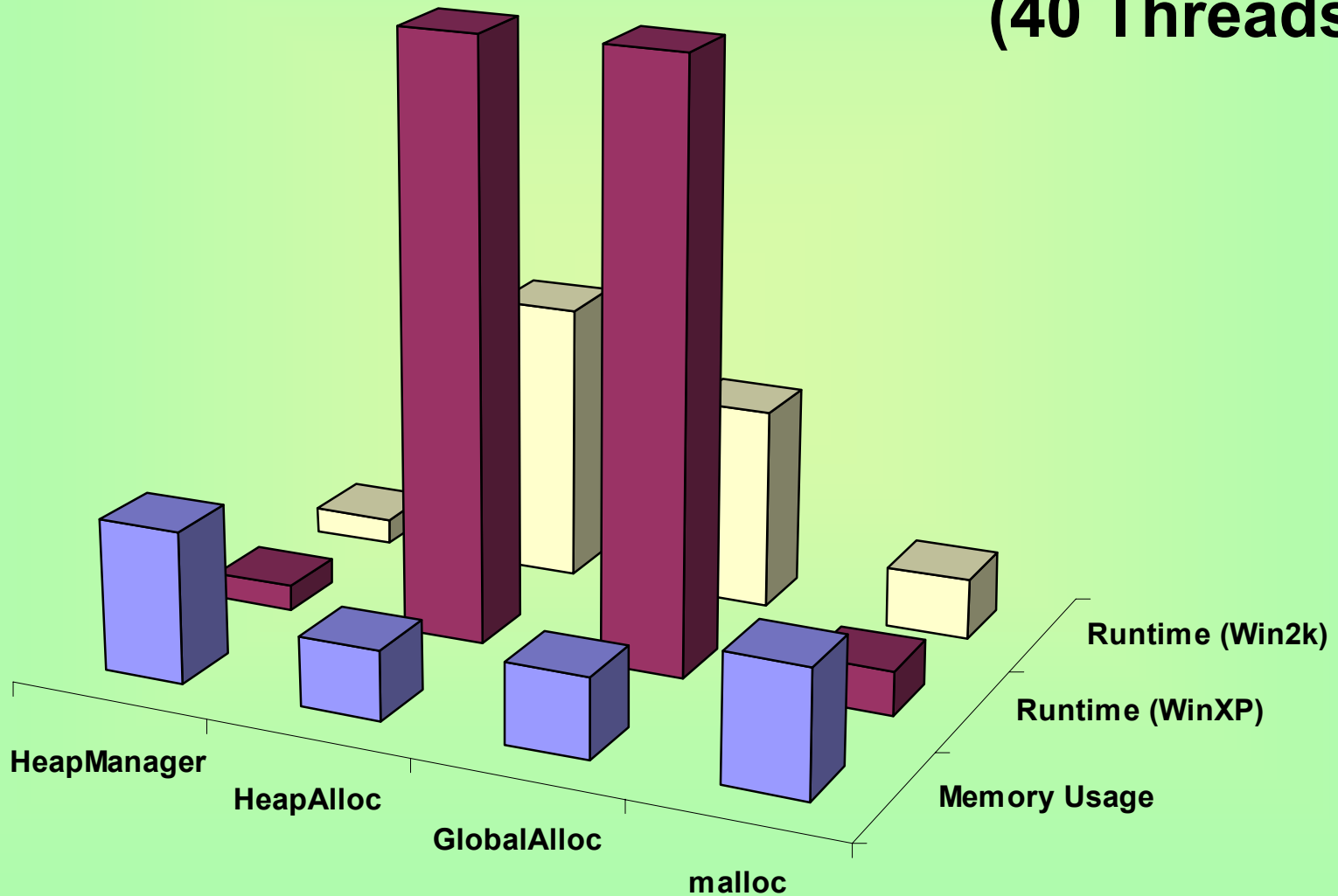
(20 Threads)



```
E:\HeapManager.exe
HeapManager      : 2509.959144 s.
HeapAlloc       : 15524.052491 s.
GlobalAlloc     : 17995.776656 s.
malloc          : 5628.022699 s.
HeapManager vs. HeapAlloc : 1:6.184982
HeapManager vs. GlobalAlloc: 1:7.169749
HeapManager vs. malloc   : 1:2.242277
```

# Tests

(40 Threads)



# Fragen

- > [tweise@gmx.de](mailto:tweise@gmx.de)
- > <http://www.tu-chemnitz.de/~weist>