Betriebssysteme (BS)

VL 9.3 – Fadenverwaltung – Windows

Volkmar Sieh / **Daniel Lohmann**

Lehrstuhl für Informatik 4
Verteilte Systeme und Betriebssysteme

Friedrich-Alexander-Universität
Erlangen Nürnberg

WS 20 – 11. Januar 2021

`https://www4.cs.fau.de/Lehre/WS20/V_BS`

Agenda

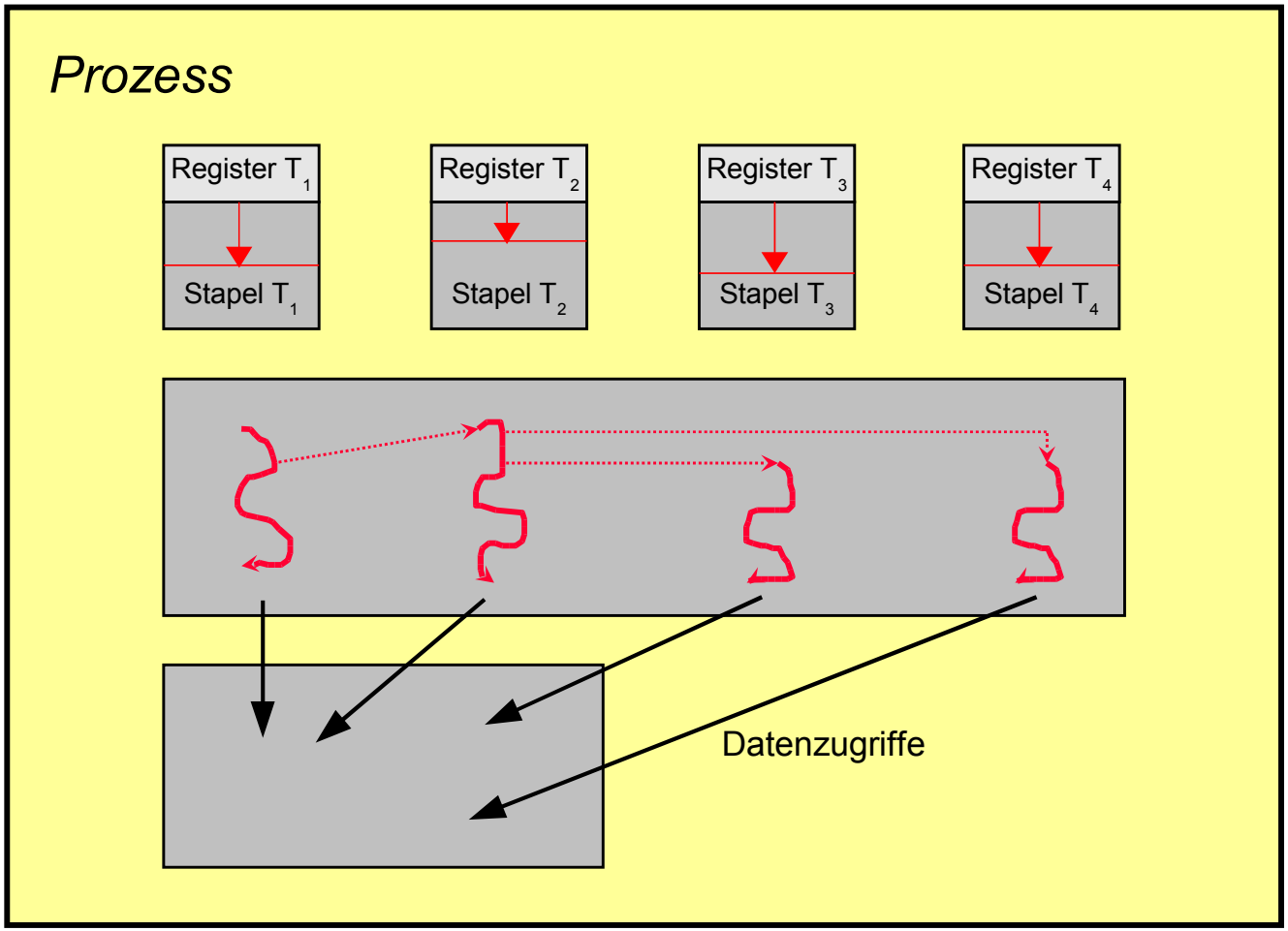
- Einordnung
- Betriebssystemfäden
- Motivation
- Kooperativer Fadenwechsel
- Präemptiver Fadenwechsel
- Ablaufplanung
  - Grundbegriffe und Klassifizierung
  - unter Windows
  - unter Linux
- Zusammenfassung

Prozesse und Fäden in Windows

Stapel + Registersatz (1 je Faden)

Code

Gobale und statische Daten



Prozesse und Fäden in Windows

- Prozess: Umgebung und Adressraum für Fäden
  - Ein Win32 Prozess enthält immer mindestens einen Faden
  - Faden (engl. thread): Code ausführende Einheit
- Fadenimplementierung wird durch den NT Systemkern erbracht
  - *Usermode-Threads* möglich („*Fibers*“), aber unüblich
- „*Threads*“ bekommen vom *Scheduler* Rechenzeit zugeteilt

Der Windows-Scheduler

- Preemptives, prioritätengesteuertes Scheduling:
  - Thread mit höherer Priorität verdrängt Thread niedrigerer Priorität
    - Egal ob Thread sich im User- oder Kernelmode befindet
    - Die meisten Funktionen der Executive („Kernel“) sind ebenfalls als Threads implementiert
  - Round-Robin bei Threads gleicher Priorität
    - Zuteilung erfolgt reihum für eine Zeitscheibe (Quantum)
- Thread-Prioritäten
  - Derzeit 0 bis 31, aufgeteilt in drei Bereiche
    - Variable Priorities: 1 bis 15
    - Realtime Priorities: 16 bis 31
    - Priorität 0 ist reserviert für den Nullseiten-Thread
  - Threads der Executive verwenden maximal Priorität 23

Zeitscheiben (Quantum)

	Kurze Quantumwerte		Lange Quantumwerte	
	Variabel	**Fix**	**Variabel**	**Fix**
Thread in HG-Prozess	6	18	12	36
Thread in VG-Prozess	12	18	24	36
Aktiver Thread in VG-Prozess	18	18	36	36

- Quantum wird vermindert
  - um den Wert 3 bei jedem Clock-Tick (alle 10 bzw. 15 msec)
  - um den Wert 1, falls Thread in den Wartezustand geht
- Länge einer Zeitscheibe: 20 – 180 msec

Prioritätsklassen, relative Threadpriorität

Relative Thread Priority		Process Priority Class: Idle 4	Below Normal 6	Normal 8	Above Normal 10	High 13	Realtime 24
Time Critical	=15	15	15	15	15	15	31
Highest	+2	6	8	10	12	15	26
Above Normal	+1	5	7	9	11	14	25
Normal		4	6	8	10	13	24
Below Normal	-1	3	5	7	9	12	23
Lowest	-2	2	4	6	8	11	22
Idle	=1	1	1	1	1	1	16

Prioritäten: *Variable Priorities*

- *Variable Priorities* (1-15)
  - Scheduler verwendet Strategien, um „wichtige“ Threads zu bevorzugen
    - *Quantum-Stretching* (Bevorzugung des aktiven GUI-Threads)
    - dynamische Anhebung (*Boost*) der Priorität für wenige Zeitscheiben bei Ereignissen
  - Fortschrittsgarantie
    - Alle 3 bis 4 Sekunden bekommen bis zu 10 „benachteiligte“ Threads für zwei Zeitscheiben die Priorität 15
  - Threadpriorität berechnet sich wie folgt (vereinfacht):

Prozessprioritätsklasse + Threadpriorität + Boost

Prioritäten: *Realtime Priorities*

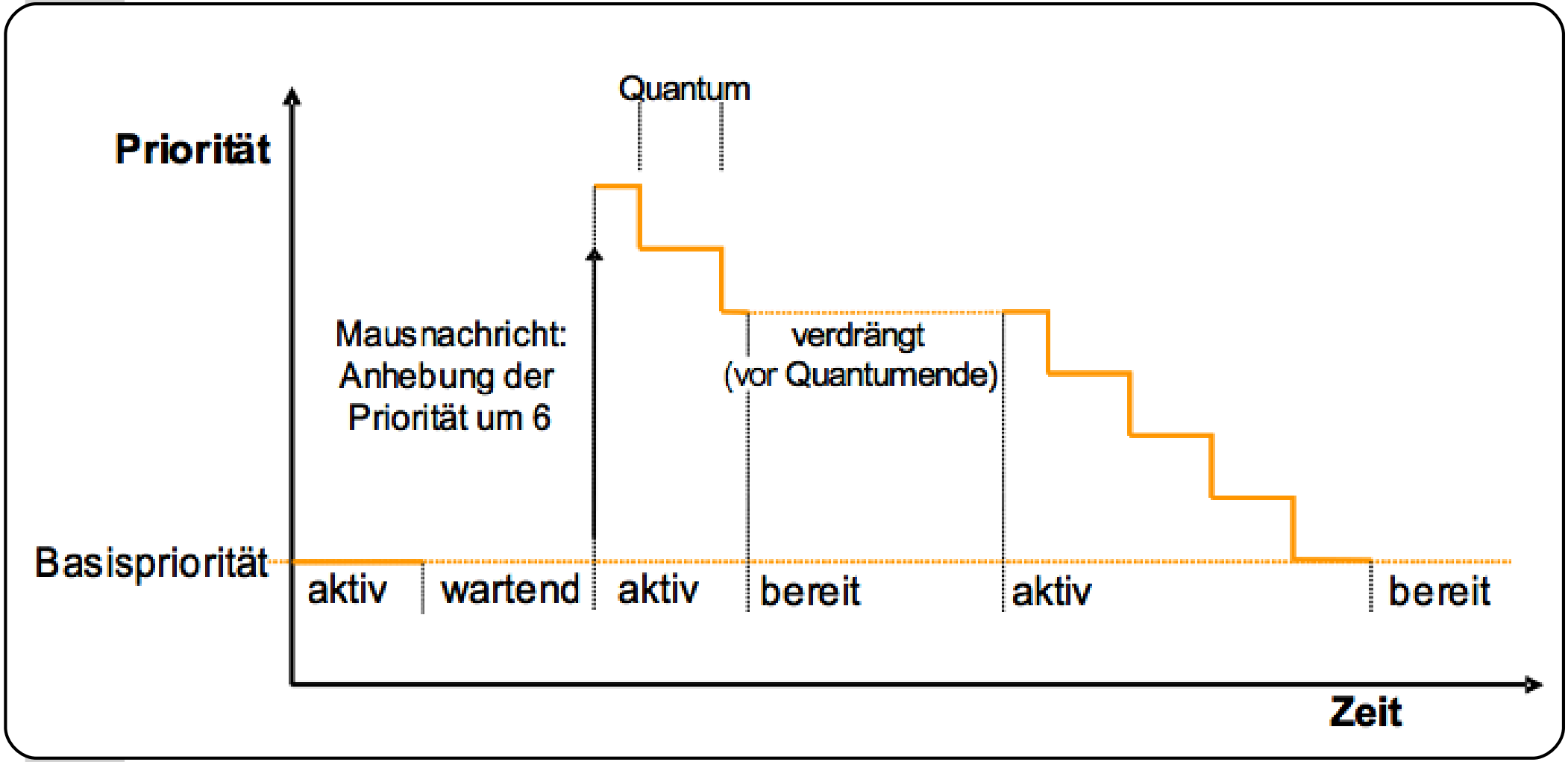
- *Realtime Priorities* (16-31)
  - Reines prioritätengesteuertes Round-Robin
    - Keine Fortschrittsgarantie
    - Keine dynamische Anhebung
    - Betriebssystem kann negativ beeinflusst werden
    - Spezielles Benutzerrecht erforderlich (SeIncreaseBasePriorityPrivilege)
  - Threadpriorität berechnet sich wie folgt:

    REALTIME_PRIORITY_CLASS + Threadpriorität

Dynamische Prioritätsanpassung

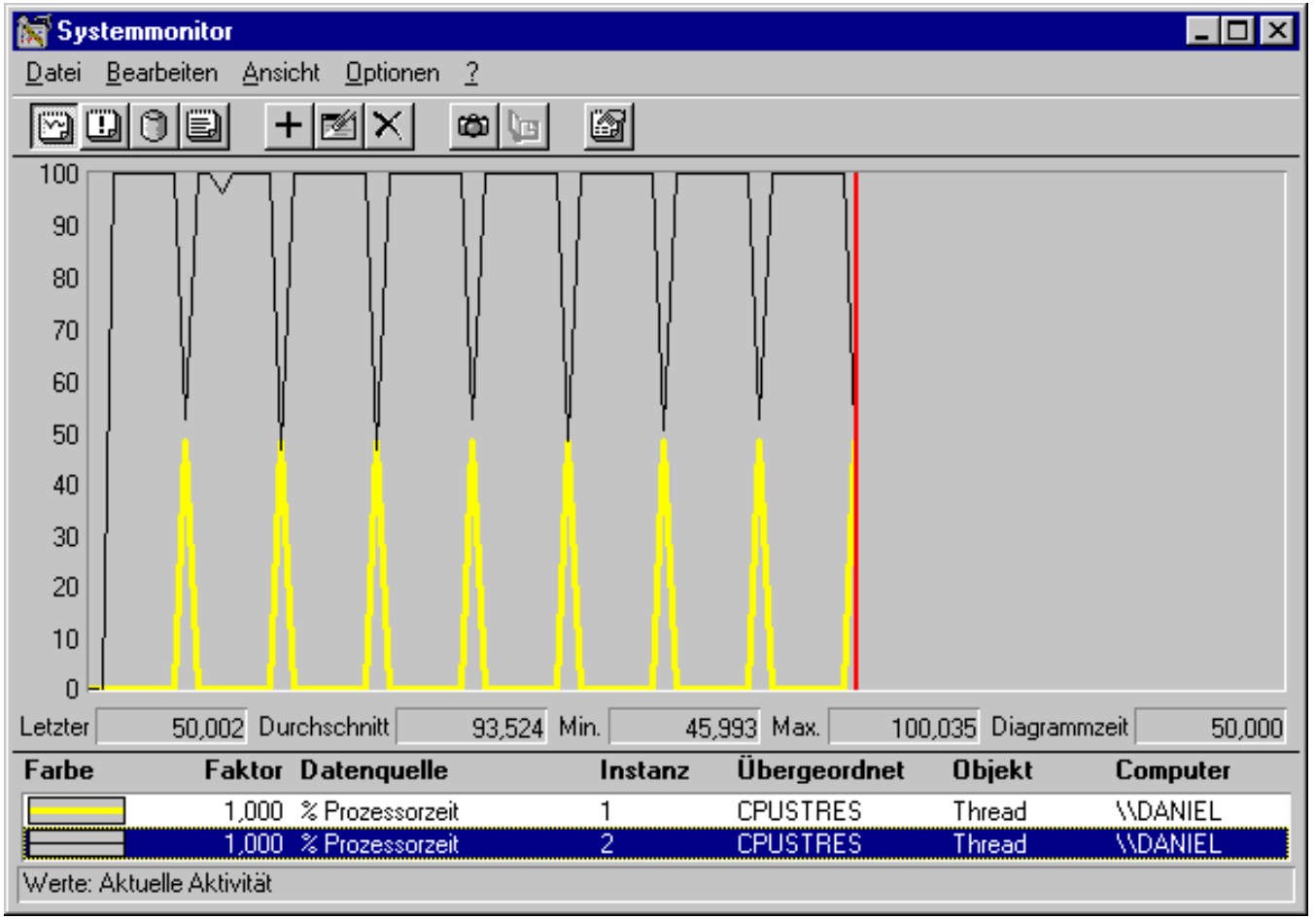
- *Dynamic Boosts*
  - Thread-Prioritäten werden vom System in bestimmten Situationen dynamisch angehoben (nicht bei REALTIME_PRIORITY_CLASS)
    - Plattenein/ausgabe abgeschlossen: +1
    - Maus-, Tastatureingabe: +6
    - Semaphore, Event, Mutex: +1
    - Andere Ereignisse (Netzwerk, Pipe,...) +2
    - Ereignis in Vordergrundapplikation +2
  - Dynamic Boost wird „verbraucht“ (eine Stufe pro Quantum)

Prioritätänderung nach einem Boost



Der *Balance-Set-Manager*

- Etwa alle 3-4 Sekunden erhalten bis zu 10 „benachteiligte“ Threads für zwei Zeitscheiben die Priorität 15
  - Implementierung der Fortschrittsgarantie



Auswahl des nächsten Threads (SMP)

Ziel: „gerechtes“ RoundRobin bei max. Durchsatz

Problem: Cache-Effekte

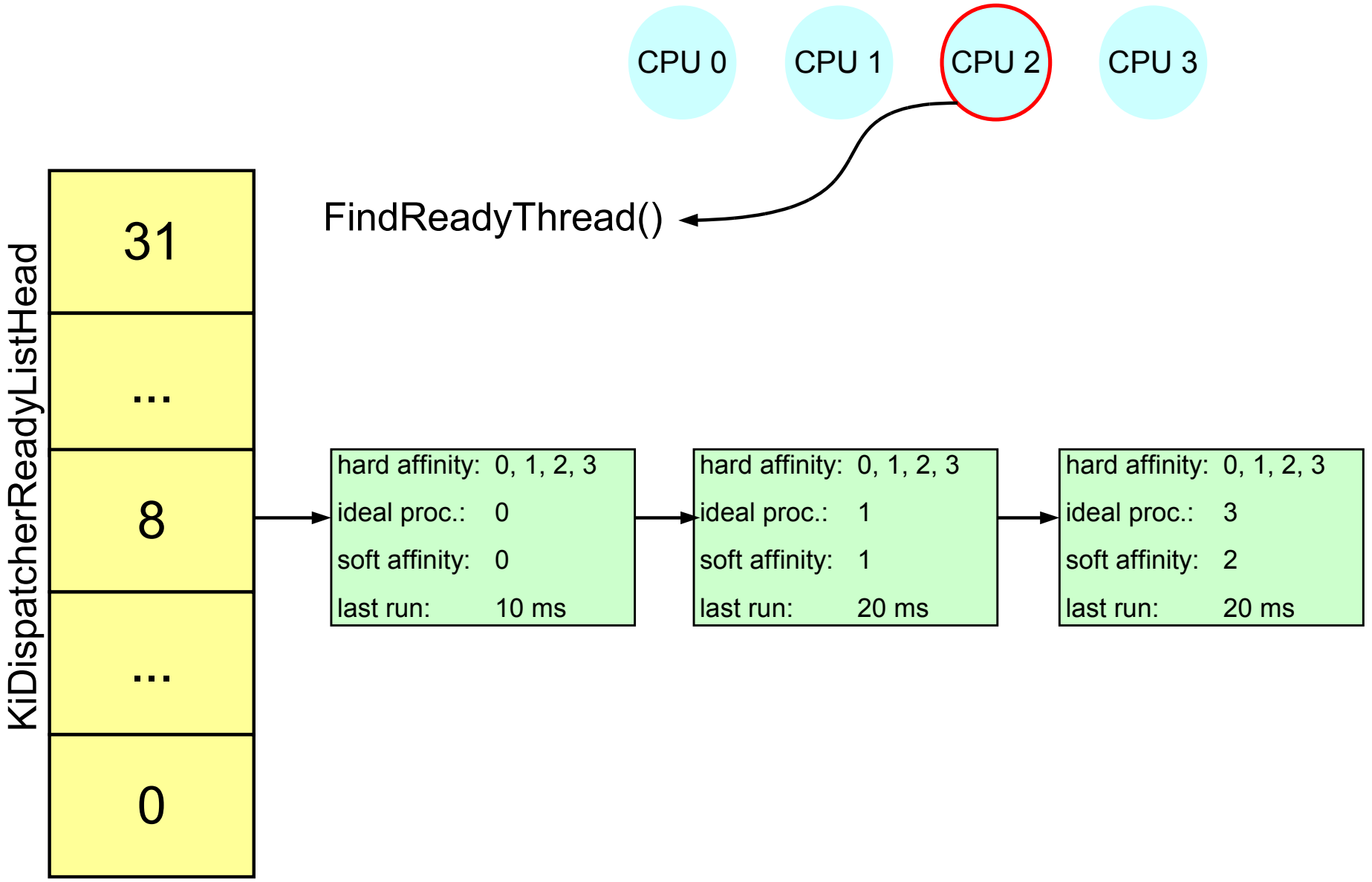
Affinität (Zuordnung von CPUs zu Thread):

- hard_affinity: Feste Zuordnung
  → explizit durch SetThreadAffinity() zugewiesen
- ideal_processor: „Ideale“ Zuordnung
  → implizit bei Erzeugung zugewiesen („zufällig“)
  → änderbar mit SetThreadIdealProcessor()
- soft_affinity: Letzte CPU, auf welcher der Thread lief
  → intern vom Scheduler verwaltet
- last_run: Zeitpunkt der letzten Zuweisung zu einer CPU
  → intern vom Scheduler verwaltet

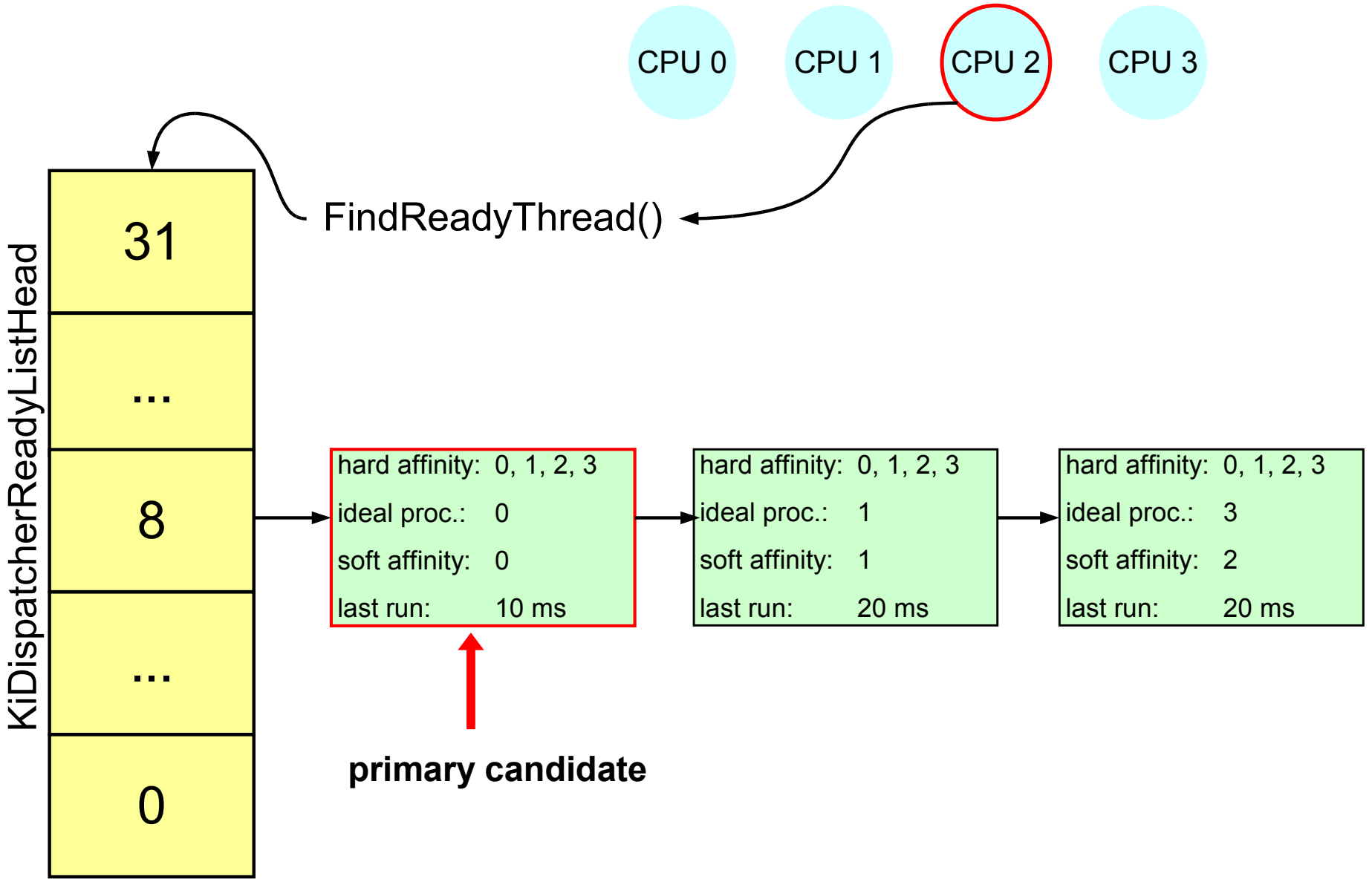
Auswahl des nächsten Threads (SMP)

- Algorithmus: CPU *n* ruft FindReadyThread() auf
  - Wähle höchstpriore, nicht-leere Warteschlange
  - Suche in dieser Warteschlange nach Thread, mit
    - soft_affinity == *n* oder
    - ideal_processor == *n* oder
    - currentTime() – last_run > 2 Quantum oder
    - priority >= 24
  - Sonst wähle Kopf der Warteschlange

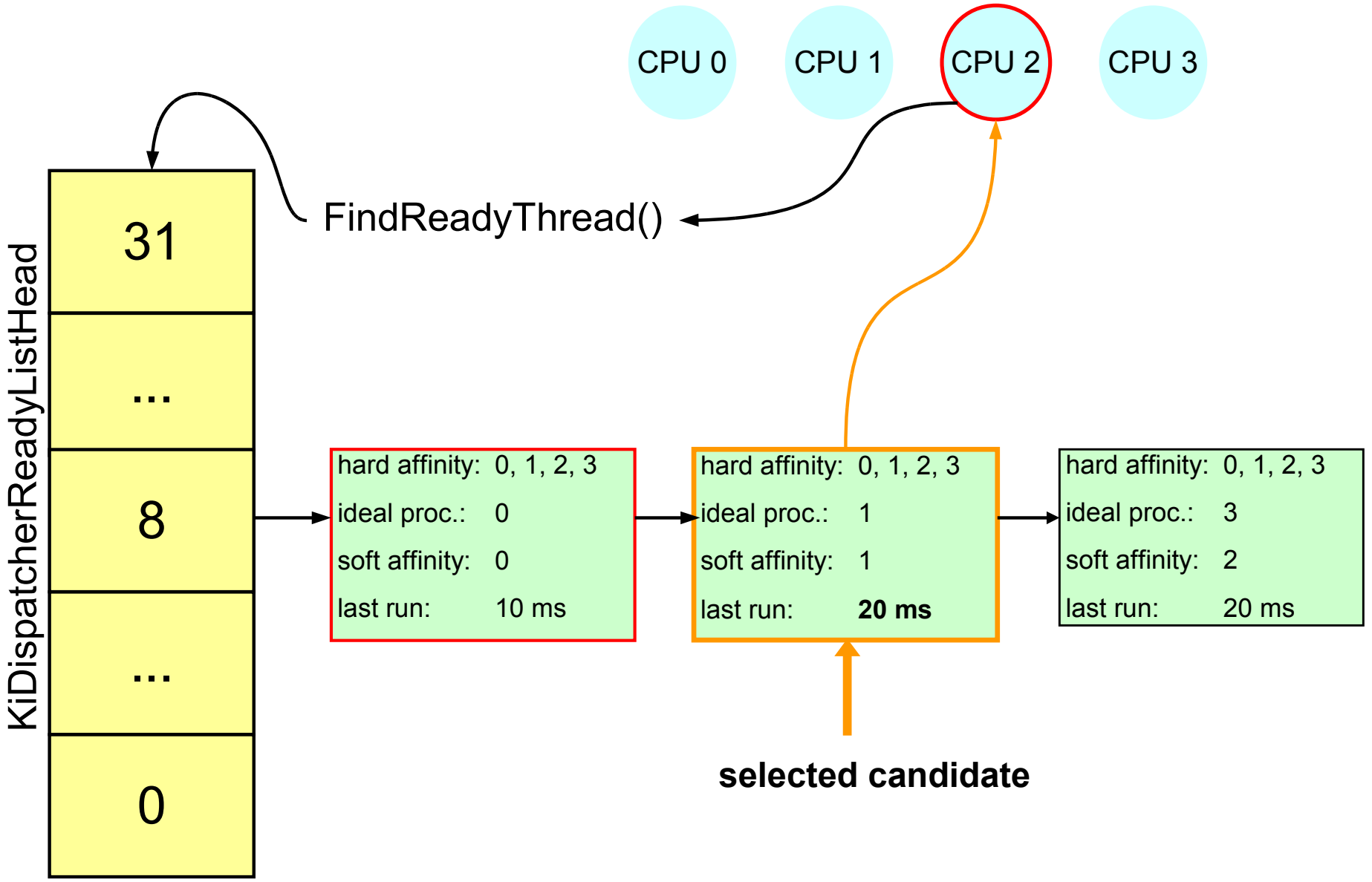
Auswahl des nächsten Threads (SMP)



Auswahl des nächsten Threads (SMP)



Auswahl des nächsten Threads (SMP)



Änderungen ab Windows 2003

- Eine ReadyQueue pro CPU
- Algorithmus: CPU n ruft FindReadyThread() auf
  - Wähle höchstpriore, nicht-leere Warteschlange von CPU n
  - Wähle Kopf dieser Warteschlange
  - Falls ReadyQueue komplett leer ist, aktiviere Idle-Loop
  - Im Idle-Loop: Durchsuche ReadyQueue anderer CPUs

Fazit Windows

- ***„interactive, probabilistic, online, preemptive, multiprocessor CPU scheduling“***
- Prioritätenmodell erlaubt feine Zuteilung der Prozessorzeit
  - Dynamische Anpassungen beachten
  - Usermode-Threads mit hohen Echtzeitprioritäten haben Vorrang vor allen System-Threads!
  - Executive ist im allgemeinen unterbrechbar
- Interaktive Threads können bevorzugt werden
  - Insbesondere GUI/Multimedia-zentrierte Threads
- Weitere Verbesserungen für SMP in Windows 2003