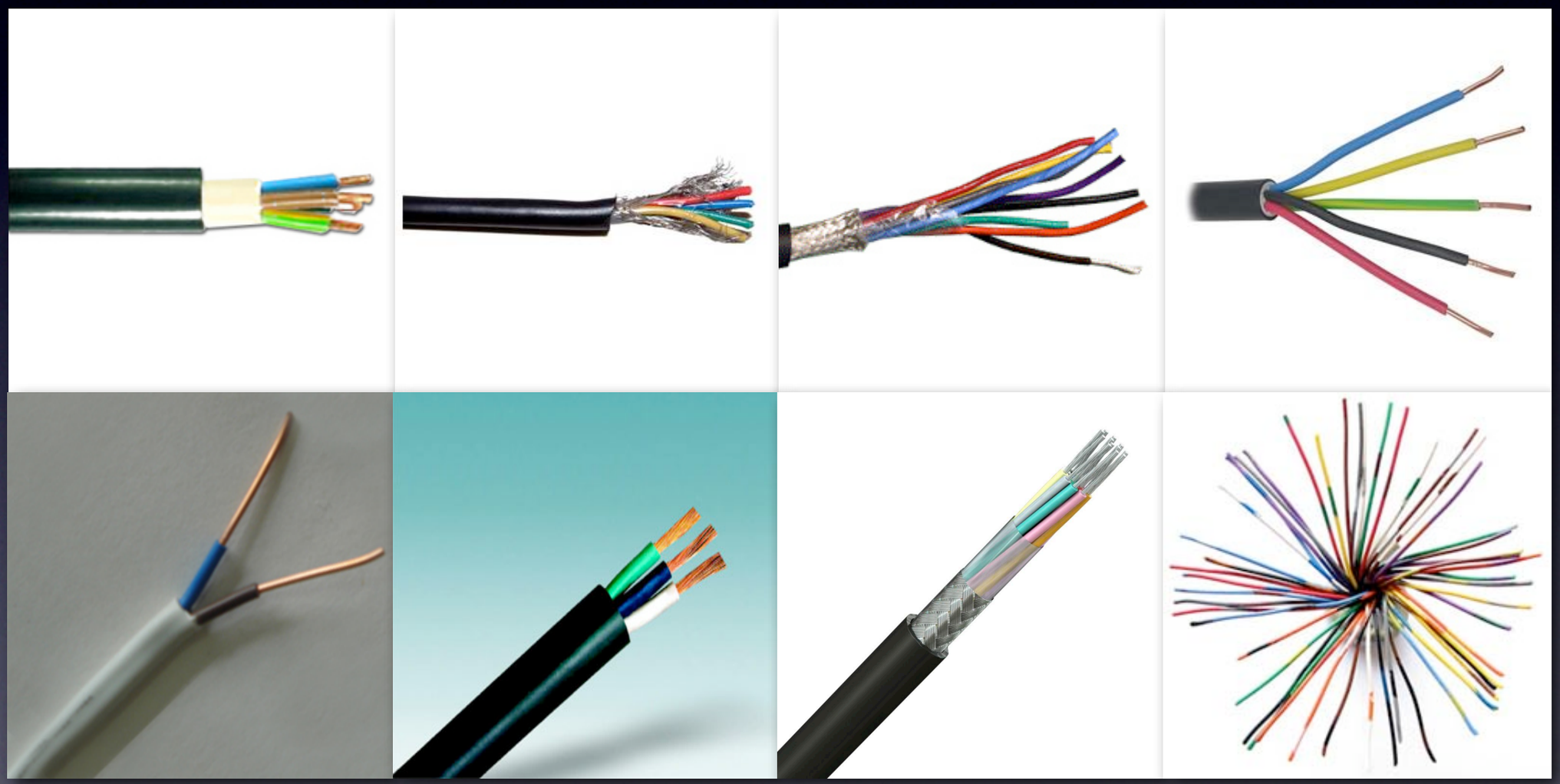
Systemsoftware im Zeitalter mehrkerniger Prozessoren

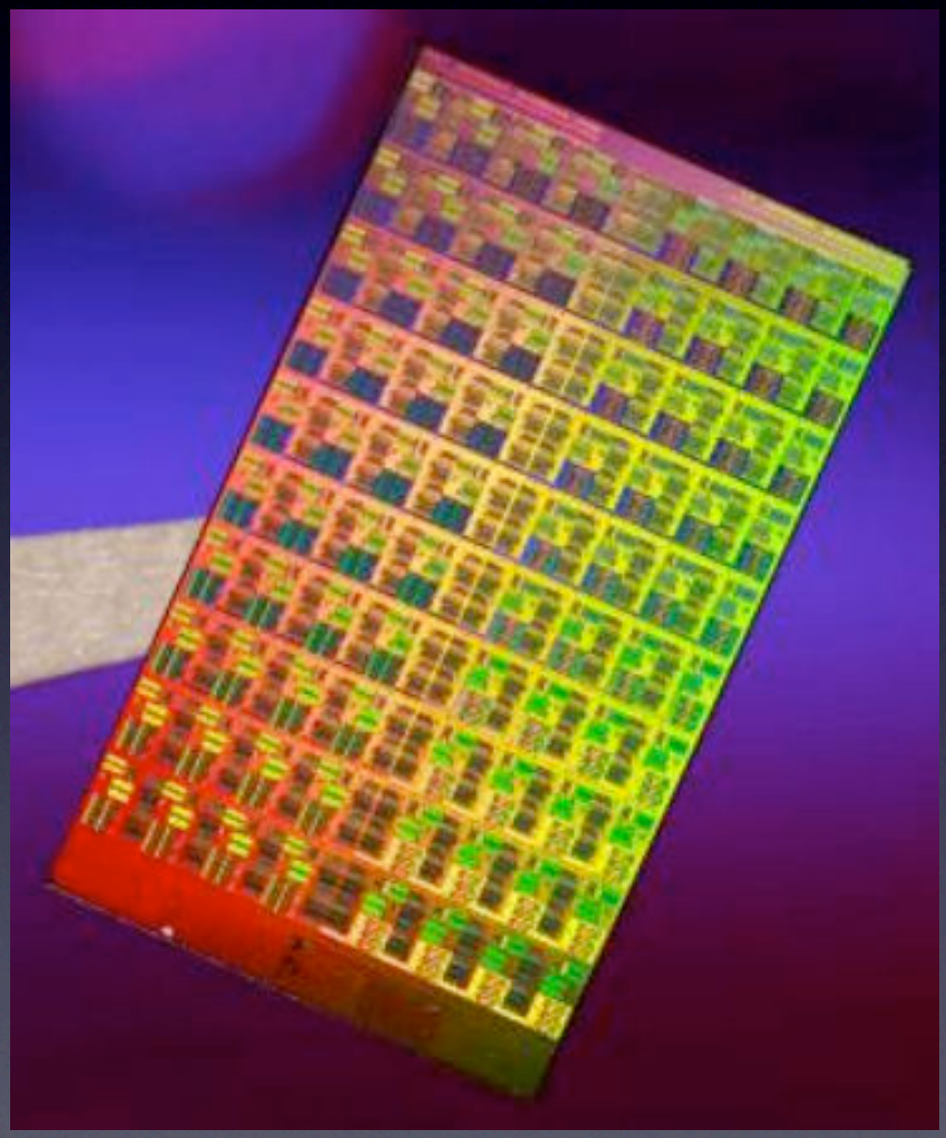
Wolfgang Schröder-Preikschat

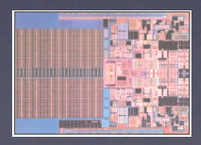
Multi-Core

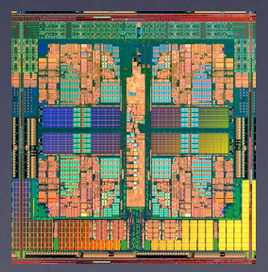


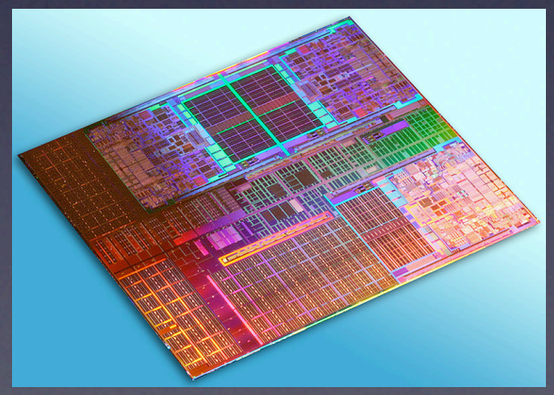
Multi-Core

Mehr- bzw. vielkernige Prozessoren, homogener oder heterogener Art.

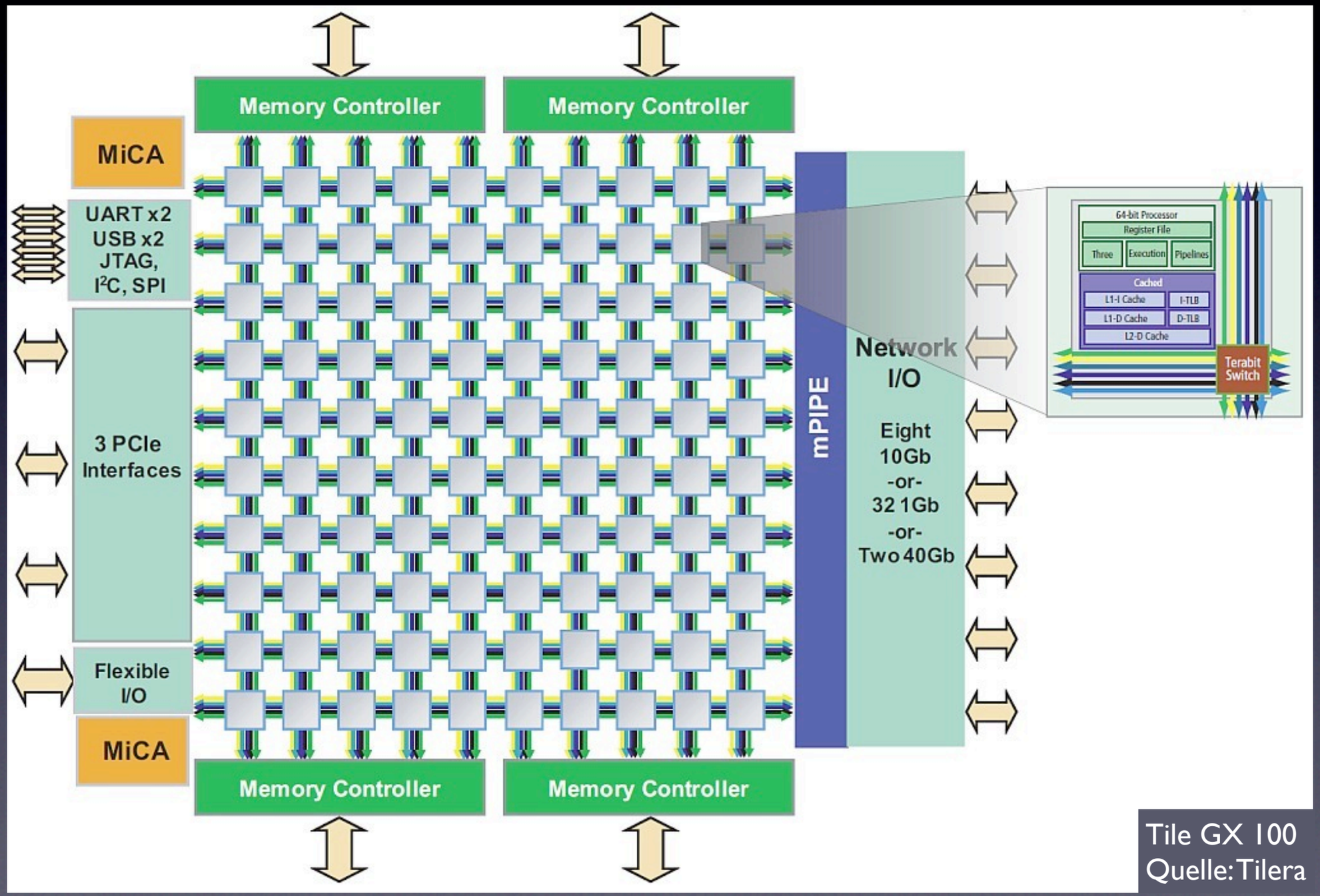








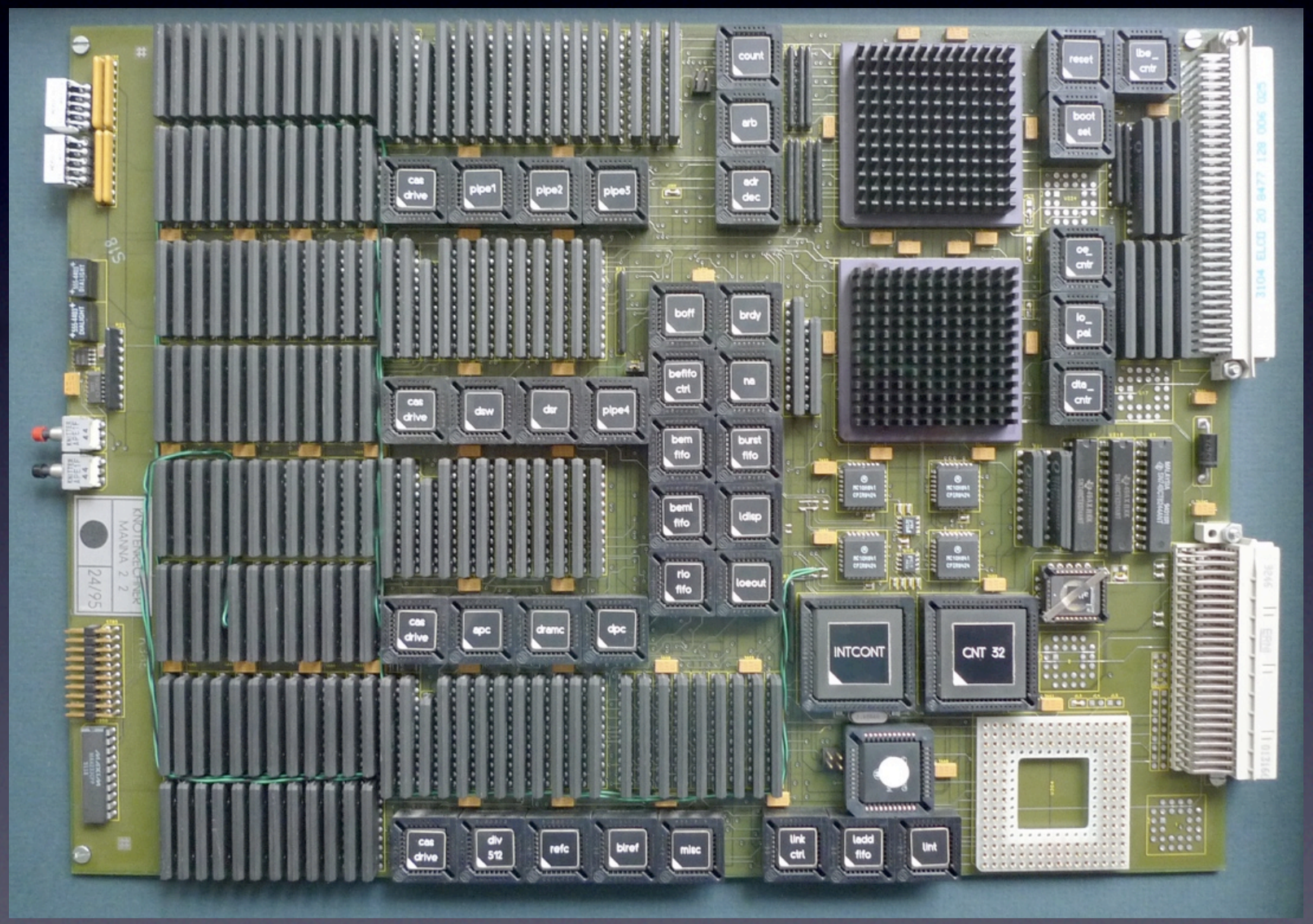
Multi-Core Embedded



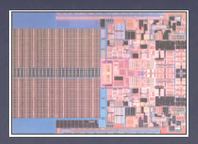
Tile GX 100
Quelle: Tilera

Déjà vu

Dualprozessor



Dualcore



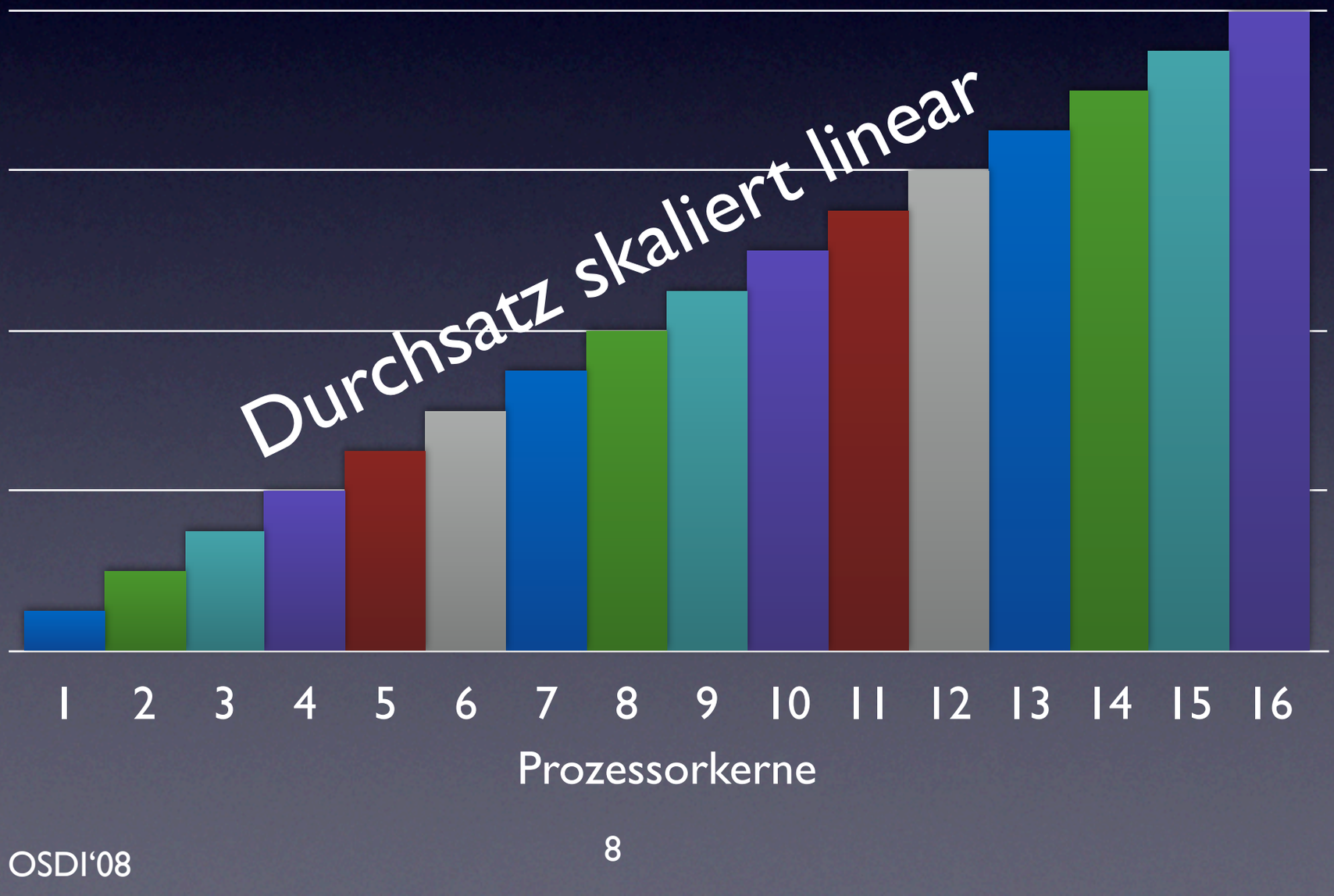
Problemfelder

- Wettstreitigkeit
  - Interferenz gleichzeitiger Prozesse
- Synchronisation
  - Kooperation & Konkurrenz koordinieren
- Latenzen
  - Kaschierung von Systemaktivitäten

Wettstreitigkeit

- Fallstudie: Dateideskriptortabelle
  - gemeinsam benutzte Datenstruktur
- Testlauf: Kosten bei gleichzeitigen Zugriffen
  - `dup`/`close` fadenlokaler Deskriptoren
  - pro Kern genau ein `dup`/`close`-Faden
  - 16-Kern AMD Opteron, Linux 2.6.27

Theorie



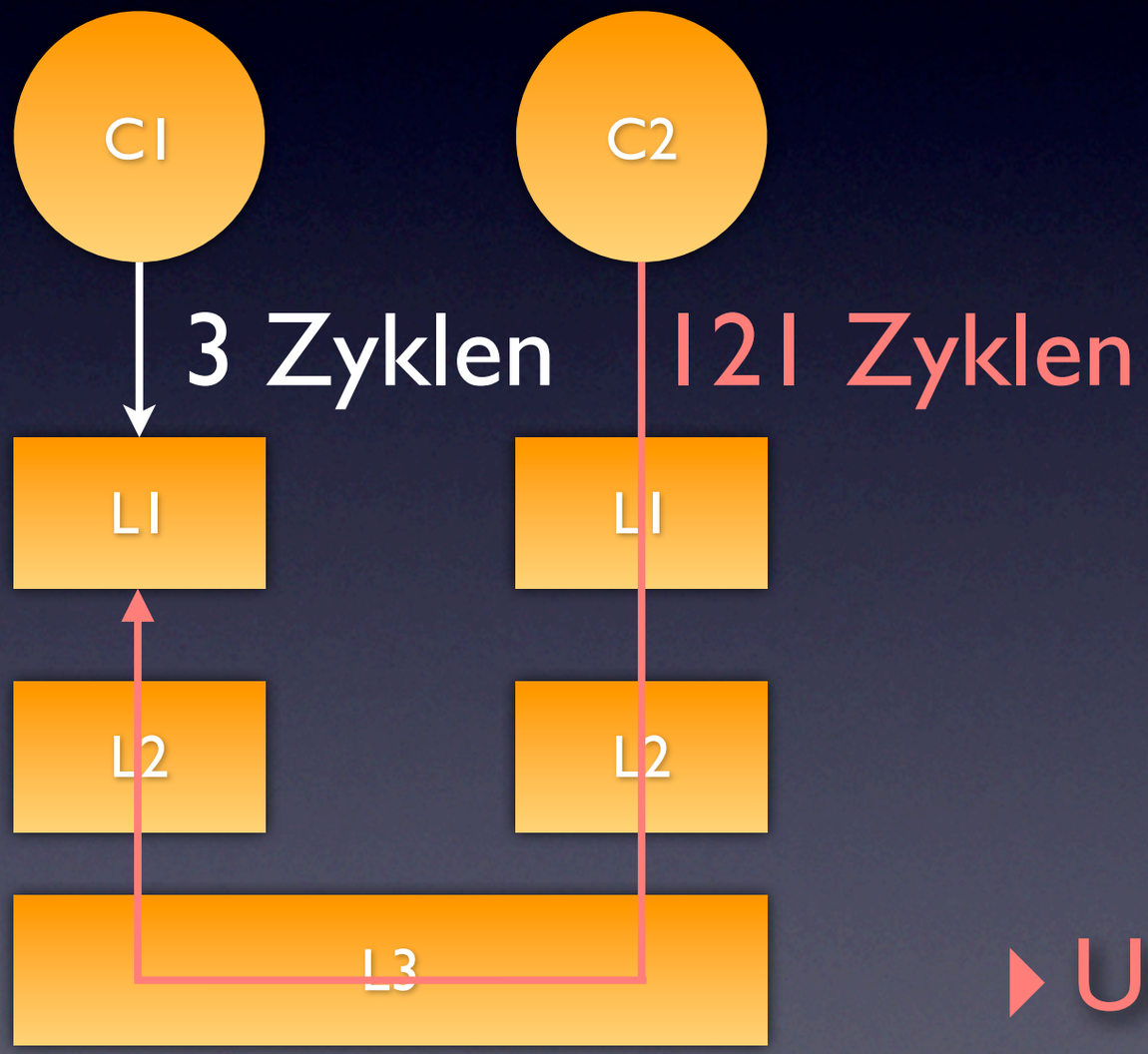
Boyd-Wickizer et al., OSDI'08

Praxis

Dateideskriptortabelle: # dup/close pro Sekunde

Boyd-Wickizer et al., OSDI'08

Zugriffsmuster




```
fd_alloc () {
    lock(fd_table);
    fd = get_free_fd();
    set_fd_used(fd);
    fix_smallest_fd();
    unlock(fd_table);
}
```

- Ursache für Durchsatzabfall

Boyd-Wickizer et al., OSDI'08

Konkurrenz

Die Vorgangssperre sequentialisiert alle Zugriffe auf `fd_table`, obwohl nicht alle Fäden auf alle Tabelleneinträge zugleich zugreifen!

```
fd_alloc () {
    lock(fd_table);
    fd = get_free_fd();
    set_fd_used(fd);
    fix_smallest_fd();
    unlock(fd_table);
}
```

false sharing ▸ Ursache keiner Verbesserung

Boyd-Wickizer et al., OSDI'08

Wo der Schuh drückt...

- Implementierung des Betriebssystems
  - mangelnde Skalierbarkeit
- Implementierung der Anwendung
  - mangelnde Parallelität
- Nutzung von Betriebssystemfunktionen
  - mangelndes Bewusstheit

Boyd-Wickizer et al., OSDI'10

Skalierungsanleitungen für den Linux-Kern

- mehrkernige Paketverarbeitung
- Elimination falscher Mitbenutzung
  - „schludrige“ (*sloppy*) Zähler
  - prozessorkernlokale Datenstrukturen
- sperrfreie Vergleiche
- Vermeidung unnötiger Blockierungen

Boyd-Wickizer et al., OSDI'10

Empfehlungen

- Mitbenutzung kontrollierbar machen
  - Anwendungen entscheiden lassen können
- geschickte Einplanung gleichzeitiger Fäden
  - Anwendungswissen einfließen lassen
- Softwaretechnik (wirklich) praktizieren
  - Entwurfsentscheidungen zurückstellen

Synchronisation

Synchronisation

- altbekannte Techniken dominieren
  - Schlossvariable, Semaphor, Monitor
- naiver Einsatz blockierender Verfahren
  - kritischer Abschnitt ≠ unteilbar
  - CPU ≠ unteilbares Betriebsmittel
- funktionale vs. nichtfunktionale Eigenschaft

Umbau von Altsoftware ist nicht trivial...

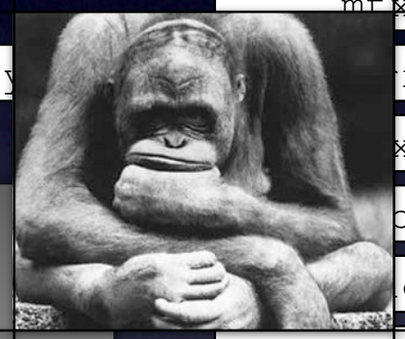
Untiefen...

FreeBSD

lock_mtx
mtx_lock
mtx_lock_flags
_get_sleep_lock
mtx_lock_sleep
nstile_wait
x_lock_spin
ock_spin_flags
ock_spin_flags
_get_spin_lock
_mtx_lock_spin
_obtain_lock
atomic_cmpset_acp_ptr
atomic_cmpset_acp_long
atomic_cmpset_long
cmpxchgq

thread_lock
thread_lock_flags
_thread_lock_flags
spinlock_enter
intr_disable

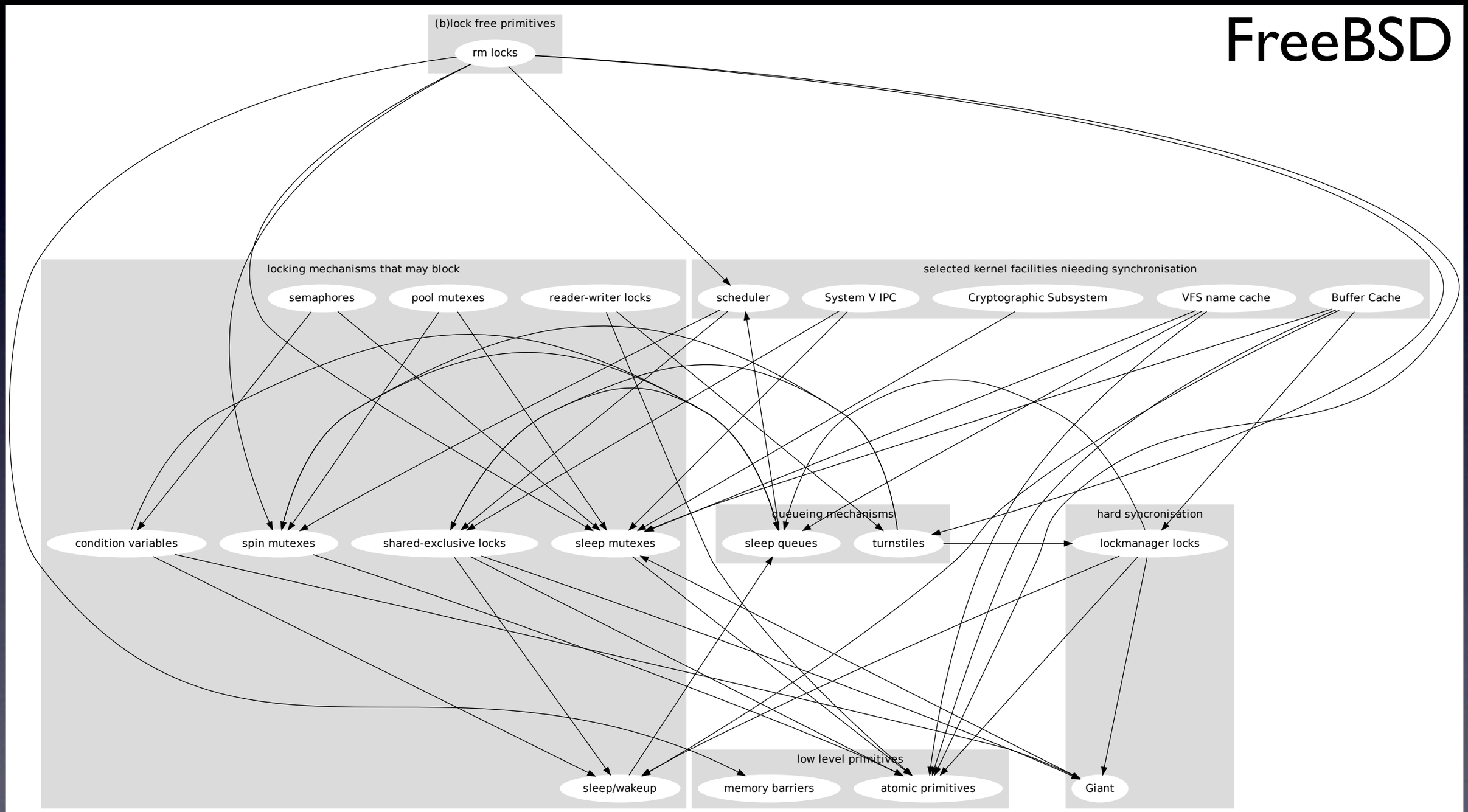
cpu_spinwait
turnstile_try



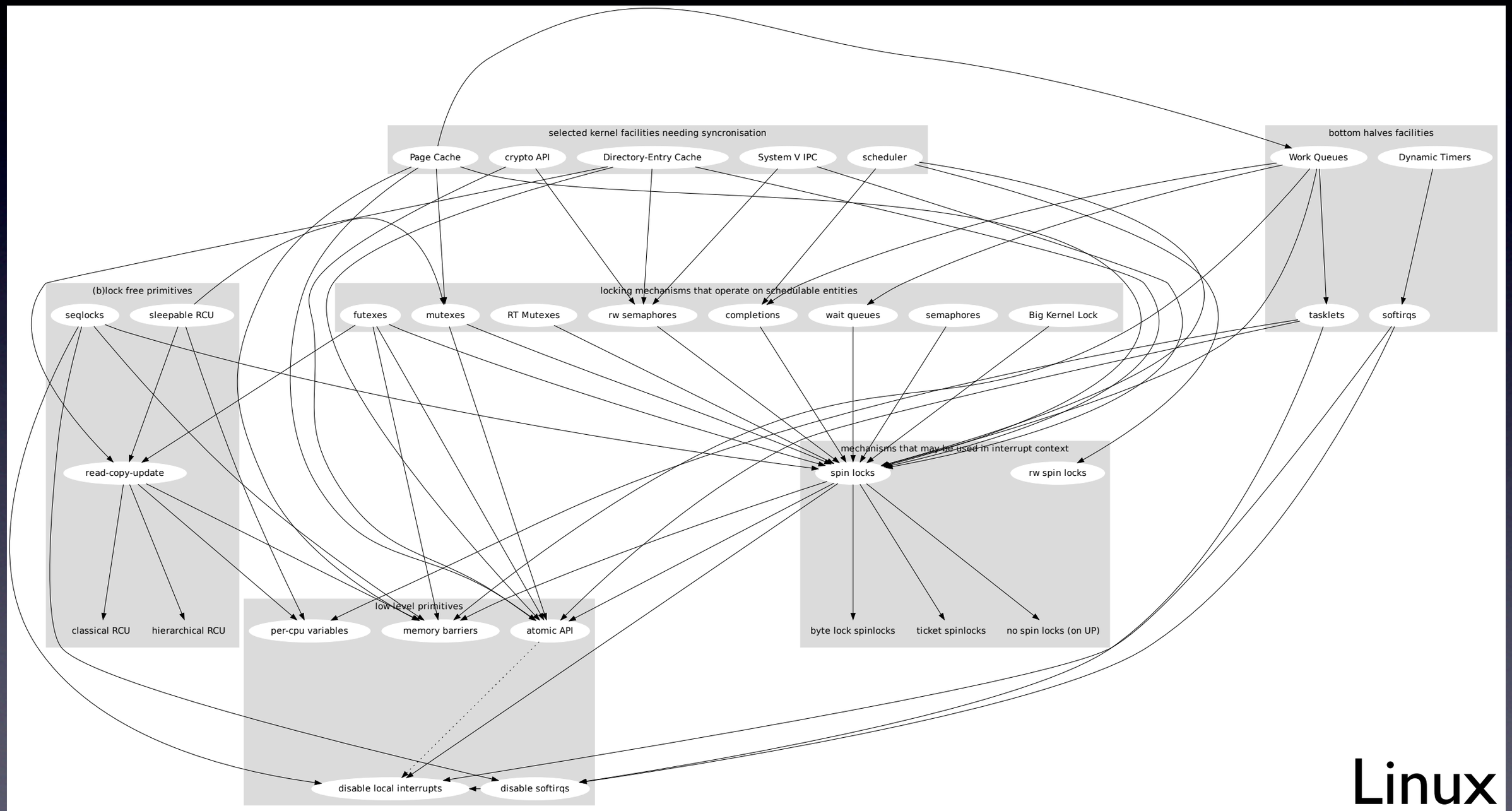
```
void enable_mmiotrace (void) {
   mutex_lock(&mmiotrace_mutex);
   if (is_enabled())
      goto out;
   if (nommiotrace)
      pr_info("MMIO trace disabled\n");
   kmmio_init();
   enter_uniprocessor();
   spin_lock_irq(&trace_lock);
   atomic_inc(&mmiotrace_enabled);
   spin_unlock_irq(&trace_lock);
   pr_info("enabled.\n");
out:
   mutex_unlock(&mmiotrace_mutex);
}
```

Linux

Untiefen...


FreeBSD
(b)lock free primitives
rm locks
locking mechanisms that may block
semaphores
pool mutexes
reader-writer locks
selected kernel facilities nieeding synchronisation
scheduler
System V IPC
Cryptographic Subsystem
VFS name cache
Buffer Cache
queueing mechanisms
sleep queues
turnstiles
hard syncronisation
lockmanager locks
condition variables
spin mutexes
shared-exclusive locks
sleep mutexes
low level primitives
sleep/wakeup
memory barriers
atomic primitives
Giant

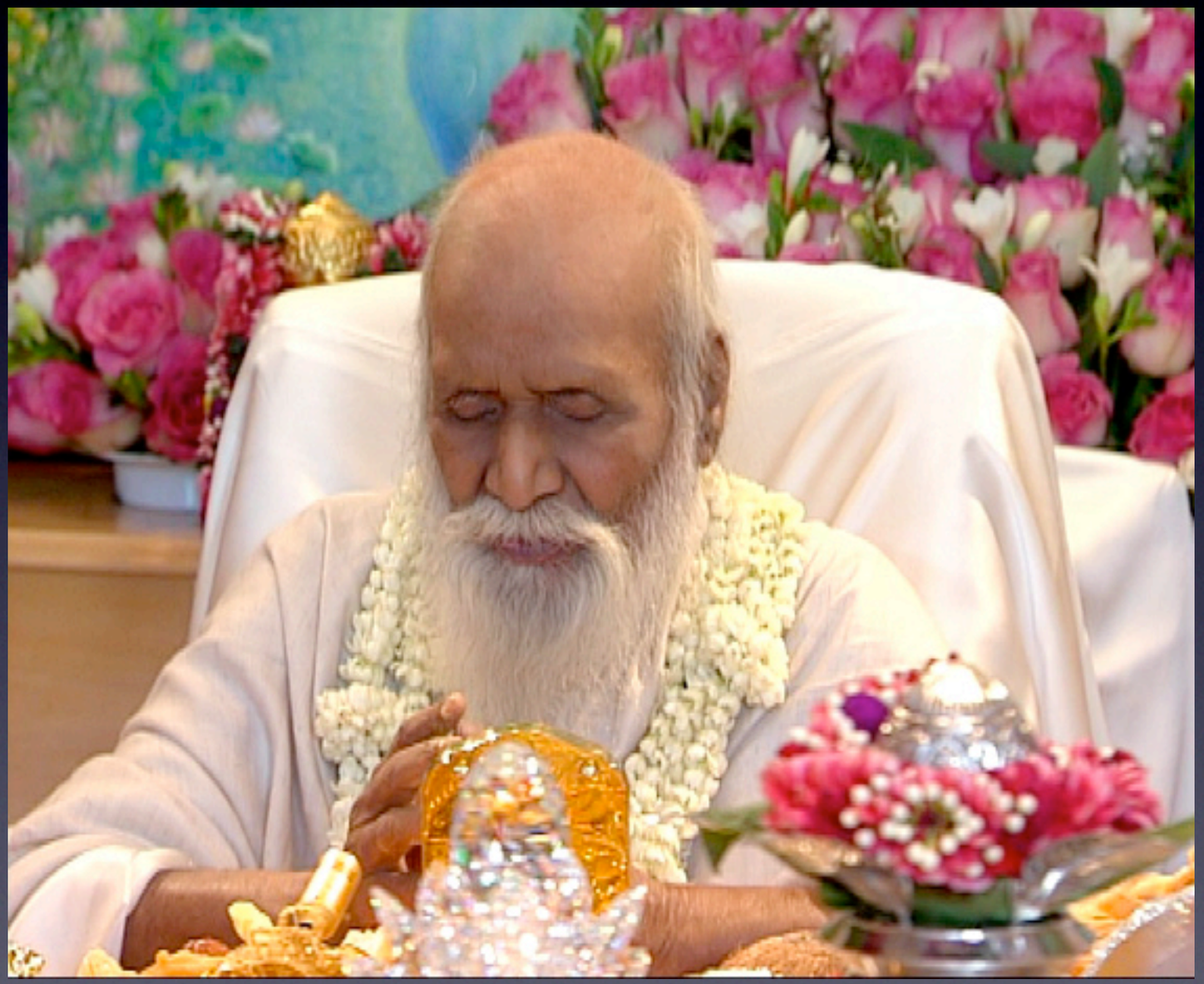
Untiefen...


selected kernel facilities needing syncronisation
Page Cache
crypto API
Directory-Entry Cache
System V IPC
scheduler
bottom halves facilities
Work Queues
Dynamic Timers
(b)lock free primitives
seqlocks
sleepable RCU
locking mechanisms that operate on schedulable entities
futexes
mutexes
RT Mutexes
rw semaphores
completions
wait queues
semaphores
Big Kernel Lock
tasklets
softirqs
mechanisms that may be used in interrupt context
read-copy-update
spin locks
rw spin locks
low level primitives
classical RCU
hierarchical RCU
per-cpu variables
memory barriers
atomic API
byte lock spinlocks
ticket spinlocks
no spin locks (on UP)
disable local interrupts
disable softirqs
Linux

Fortschrittsgarantien

- behinderungsfrei (*obstruction-free*)
  - außer Konkurrenz, Fortschritt garantiert
- sperrfrei (*lock-free*)
  - Fortschritt des Systems garantiert
- wartefrei (*wait-free*)
  - Fortschritt jedes Fadens garantiert

TM



Transactional Memory

- Hardware (1986)
  - Sun Rock (ASPLOS'09) ?
- Software (1995)
  - auf Basis konventioneller Primitiven
- Hybrid (2005)
  - HTM, solange Ressourcen es hergeben
  - STM, sonst...

HTM

STM

(2009)

Transaktionen...

```
void stm_ahead (chain_t *this, chain_t *item) {
   transactional {
      item->link = this->link;
      this->link = item;
   }
}
```

```
chain_t *stm_strip (chain_t *this) {
   chain_t *node;
   transactional {
      node = this->link;
      if (node != 0)
         this->link = node->link;
   }
   return node;
}
```

...considered harmful!

```
void ewd_prolaag (semaphore_t *this) {
    transactional {
        if (this->load-- <= 0)
            sad_sleep(&this->line);
    }
}
```

```
void ewd_verhoog (semaphore_t *this) {
    transactional {
        if (this->load++ < 0)
            sad_awake(&this->line);
    }
}
```

Pro und Contra TM

- geeignet für „einfache“ kritische Abschnitte
- komponierbar und verklemmungsfrei: wie CAS bzw. LL/SC, aber semantisch mächtiger
- Nutzung in Betriebssystemen gestaltet sich nicht einfacher als mit CAS bzw. LL/SC
- erfordert Hardwareunterstützung — und fordert einiges an Hardwareressourcen

Latenzverbergung

- altbekannte Technik im Bereich HPC
  - Minimierung von Startzeiten (IPC, E/A)
  - asymmetrische Mehrprozessorsysteme
- weitgehend ungenutzt für Echtzeitbetrieb
  - deterministische Programmabläufe
- Prozessorkerne für Betriebssystemzwecke

Verwendung der Kerne

Mehrkernprozessor	
— Universalkerne	*Balancierung allgemeiner Systemlast*
— Spezialkerne	*Kaschierung systembedingter Latenzen*
— Wandelkerne	*„horizontale“ Parallelität (Systemaufrufe)*
— Fixkerne	*„vertikale“ Parallelität (Fließbandverfahren)*
— Treiberkerne	*abgesetzte Geräteprogrammierung*
— Gerätekerne	*Steuerung der Peripherie*
— Unterbrecherkerne	*Annahme von (asynchronen) Unterbrechungen*
— Dienstkerne	*abgesetzte Dienstleistung*
— Einplanungskern	*Zustellung von Arbeitsaufträgen*
— Ein-/Ausgabekern	*Operationen auf Dateien*
— Kommunikationskern	*Abwicklung des Protokollstapels*
— Seitenumlagerungskern	*„external pager“ (Ersetzungsstrategie)*

Aktuelle Entwicklungen

- Corey (MIT, Exokern-artig)
- Barrelfish (ETH, Mikrokern-artig)
- Factored OS (MIT, Mikrokern-artig)
- TxLinux (UT, Monolith) — Simulation
- Helios (MSR, Mikrokern-artig)
- TxOS (UT, Monolith)

TM

Resümee

- Multi-Core — des Kaisers neue Kleider!?
- Wettstreitigkeit vorbeugen/vermeiden
- viel stärker optimistische Steuerung von Nebenläufigkeit in Erwägung ziehen
- Systemlatenzen mittels Kernen verbergen
- TM kein Allheilmittel (für Systemsoftware)