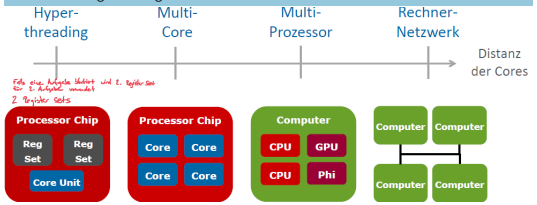
Multithreading Grundlagen



Parallelität: Aufteilung in Teilabläufe, laufen gleichzeitig auf mehreren Prozessoren. **Nebenläufigkeit:** Gleichzeitig oder verzahnt ausführbare Abläufe, greifen au gemeinsame Ressourcen zu. **Prozess:** Parallel laufende Programm-Instanz im System. Eigener Adressraum. **Thread:** Parallele Ablaufsequenz innerhalb eines Programms. Teilen gleichen Adressraum.

Thread-Implementationen

User-Level Threads: Im Prozess implementiert. Keine echte Parallelität durch mehrere Prozessoren. **Kernel-Level Threads:** Im Kernel implementiert (Multi-Core Ausnutzung). Kontextwechsel vom Prozess per SW-Interrupt. **Thread Scheduling:** Processor Sharing - #Threads >#Prozessoren.

Prozessor Multiplexing

Verzahnte Ausführung: Instruktionen von mehreren Threads in Teilsequenzen. Illusion der Parallelität. **Kontextwechsel:** *Synchron* (Freiwillige abgabe), Thread wechselt zu wait. *Asynchron:* (gezwungene Abgabe) Begrenzte Laufzeit für Threads.

Multi-Tasking:

Kooperativ: Threads initiieren Kontextwechsel synchron (freiwillig). Scheduler kann Thread nicht unterbrechen. **Preemtiv:** Scheduler kann Thread mit Timer-Interrupt asynchron unterbrechen (Time-Sliced-Scheduling)(*Standart heute*). **Thread Zustände:** Ready, Waiting, Running.

Multi-Thread Programmierung

JVM Thread Modell

Java ist ein Single Process System. JVM ist ein Prozess im Bsys. **Main-Thread** wird beim Aufstarten der JVM anhand *main()* Methode erzeugt. JVM läuft, solange Threads laufen (Ausnahme Daemon Threads).

```
var myThread = new Thread(() -> { // ... });
myThread.start();
```

Thread wird erst bei *start()* erzeugt. Führt *run()*-Methode des Runnable Interface aus. Thread endet beim Verlassen von *run()*.
Nicht-Determinismus: Threads laufen ohne Vorkehrungen beliebig verzahnt oder parallel.

Explizite Runnable-Implementation:

```
class SimpleLogic implements Runnable {
   @Override
   public void run() { // ... }
}
var myThread = new Thread(new SimpleLogic()).start();
```

Sub-Klasse von Thread

```
class SimpleThread extends Thread {
   @Override
   public void run() { // ... }
}
var myThread = new SimpleThread().start();
```

Thread Join

Warten auf Beendigung eines Threads. *t2.join()* blockiert, solange t2 läuft.

Thread Passivierung

Thraed.sleep(ms): Laufender Thread geht in Wartezustand, dann ready. **Thread.yield():** Gibt Prozesor frei, direkt ready.