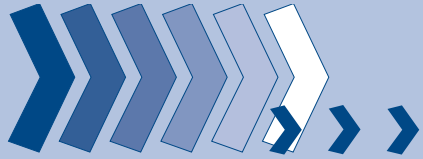
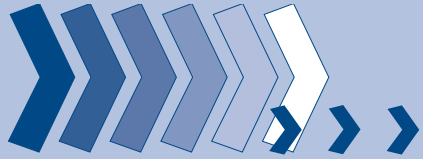


Echtzeitsysteme: Grundlagen



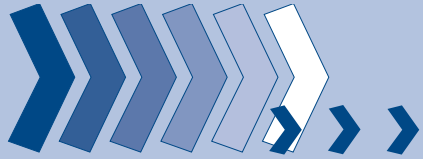
Was ist Echtzeit?





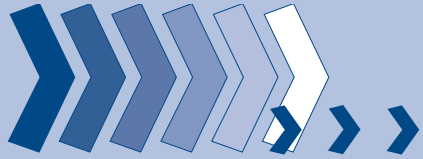
Was ist Echtzeit?

- Es gibt eine Reihe verwirrender Vorstellungen, was ein Echtzeitsystem ist, beziehungsweise, welche Eigenschaften ein solches ausmachen



Was ist Echtzeit?

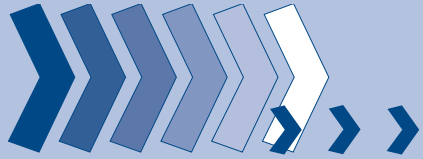
- Es gibt eine Reihe verwirrender Vorstellungen, was ein Echtzeitsystem ist, beziehungsweise, welche Eigenschaften ein solches ausmachen
- Echtzeitsysteme müssen
 - “schnell” sein?
 - vorhersagbar sein?
 - rechtzeitig Ergebnisse liefern?
 - “sicher” sein?
 - fehlerfrei arbeiten?
 - so konstruiert sein, daß sie auch im Fehlerfall noch funktionieren?
- Definition ist erforderlich



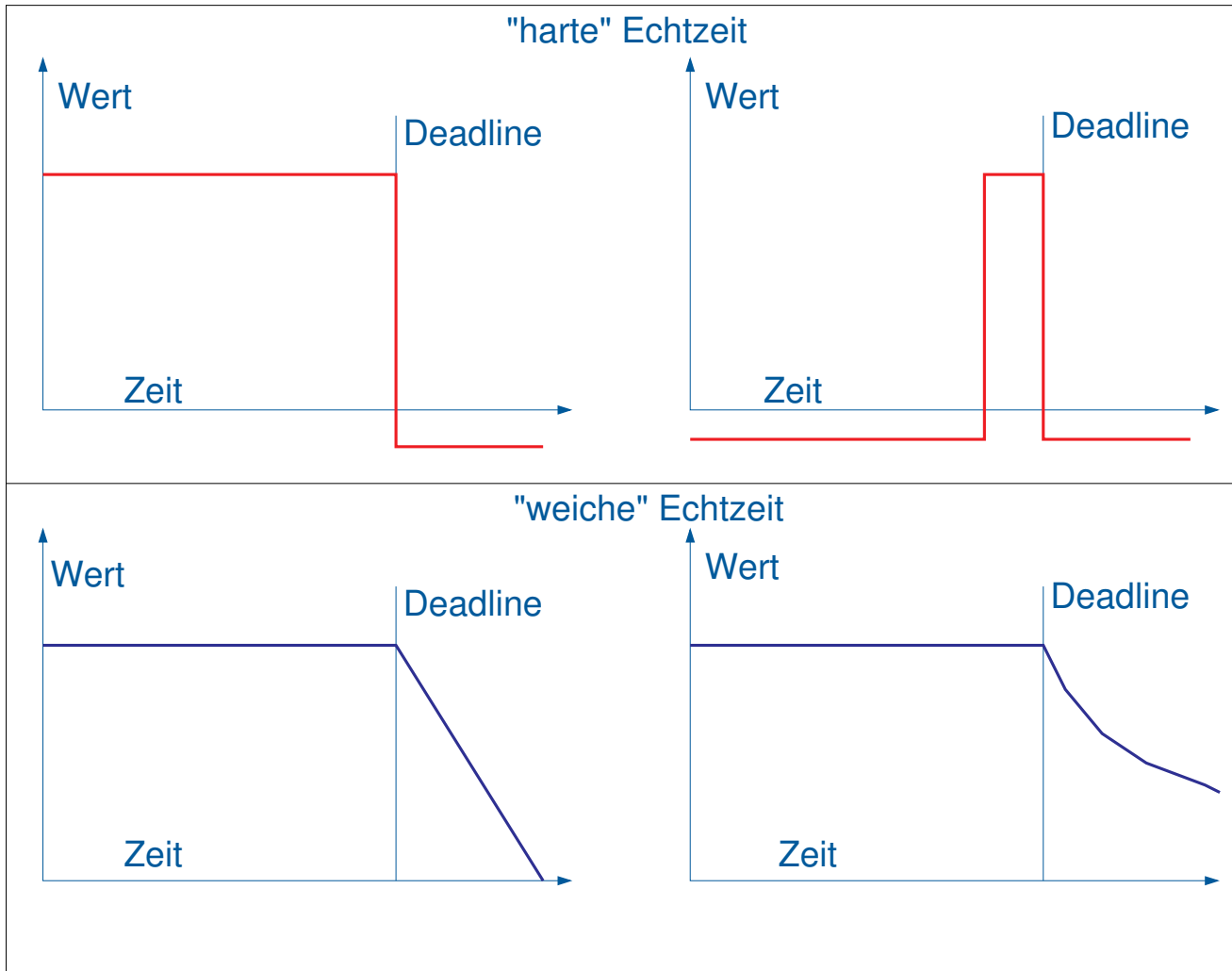
Definition 1 - Krishna und Shin

“Any system where a timely response by the computer to external stimuli is vital is a real-time system”

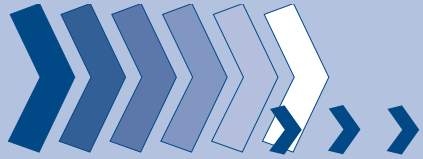
- “Rechtzeitig” — definiert über Deadlines
- Deadlines — spätester Zeitpunkt der Vollendung einer Task
- rechtzeitige Reaktion — immer? manchmal? in den meisten Fällen?
- Ist dann jeder Computer ein Echtzeitsystem?



Definition 2 - Resultat/Wert-Funktion



Echtzeitsysteme sind Computersysteme, bei denen der Nutzen eines Resultates nicht nur vom Resultat abhängt, sondern auch vom Zeitpunkt der Auslieferung des Resultats.



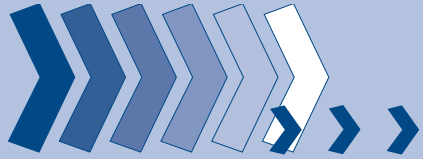
Definition 3 - analog Krishna und Shin

Ein Echtzeitsystem ist alles, was wir in dieser Vorlesung als Echtzeitsystem verstehen. Das umfaßt eingebettete Systeme wie beispielsweise

- Steuerungen in Fahr- und Flugzeugen
- Automatisierungsanlagen in der Industrie und in Kraftwerken
- ... und andere Objekte, wo Unglücke geschehen, wenn das Computersystem nicht rechtzeitig Ergebnisse liefert

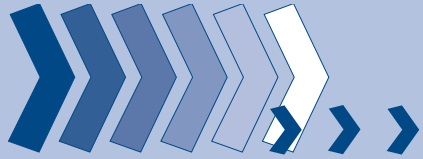
Diese System nennen wir *harte* Echtzeitsysteme.

Im Gegensatz dazu gibt es auch *weiche* Echtzeitsysteme wie Multimedia-Systeme, bei denen verpaßte Deadlines “nur” die Qualität des Ergebnisses beeinträchtigen, aber keine weiteren Auswirkungen hat.



Beispiel für Steuerungen: Auto und Fahrer

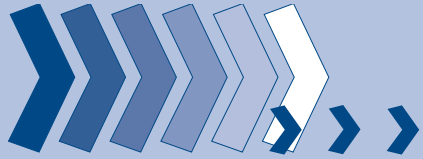
- Fahrer:
Echtzeitsteuerung
- Auto:
Gesteuerter Prozeß
- Bedienelemente:
Aktuatoren
- Mission:
Von A nach B fahren ohne Kollisionen mit anderen Fahrzeugen, Personen, oder Gegenständen und unter Einhaltung der Verkehrsregeln



Wie mißt man die “Performance” des Fahrers? Genügt “Ankommen”?

- Maß: Zeit bis zum Ankommen
 - Mit 15 km/h sicher fahren in einem Schneesturm ist ein akzeptables Ergebnis
 - Mit 15 km/h auf einer leeren Autobahn fahren ist inakzeptabel
- Maß: Unfallfreiheit
 - In den Graben fahren, weil das Auto in einer Kurve ins Schleudern gekommen ist, ist inakzeptabel
 - In den Graben fahren, um damit einen (unverschuldeten) Frontalzusammenstoß zu vermeiden, ist akzeptabel

Performance-Maße hängen von der Umgebung ab und können nur relativ zu dem bestmöglichen Ergebnis für die jeweiligen Parameter der Umgebung gemessen werden.



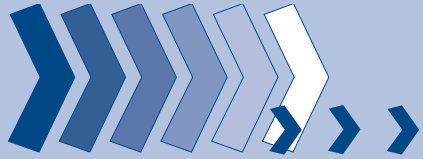
“Der Fahrer war während 99.99% der Fahrzeit wach”

→ Nutzlose Information

“Der Fahrer hat nie länger als 500 ms geschlafen”

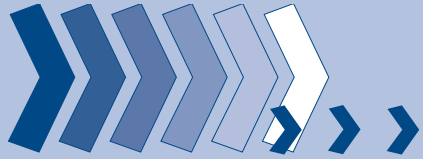
→ Verwertbare Information (unter gewissen Nebenbedingungen)

- Durchschnittswerte sind nicht ausreichend
- Werte sind von Nebenbedingungen abhängig



Tasks und Deadlines

- Was sind zeitkritische Tasks?
- Was sind ihre Deadlines?
- Was sind unkritische Tasks?



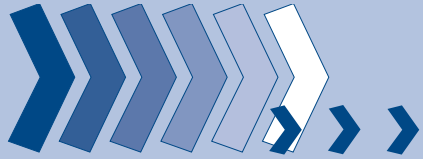
Tasks und Deadlines

- Was sind zeitkritische Tasks?
 - Bremsen
 - Lenken
 - (Beschleunigen)
- Was sind ihre Deadlines?
- Was sind unkritische Tasks?

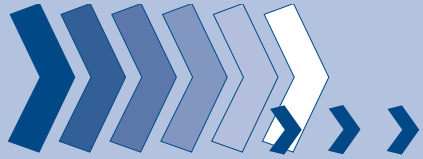


Tasks und Deadlines

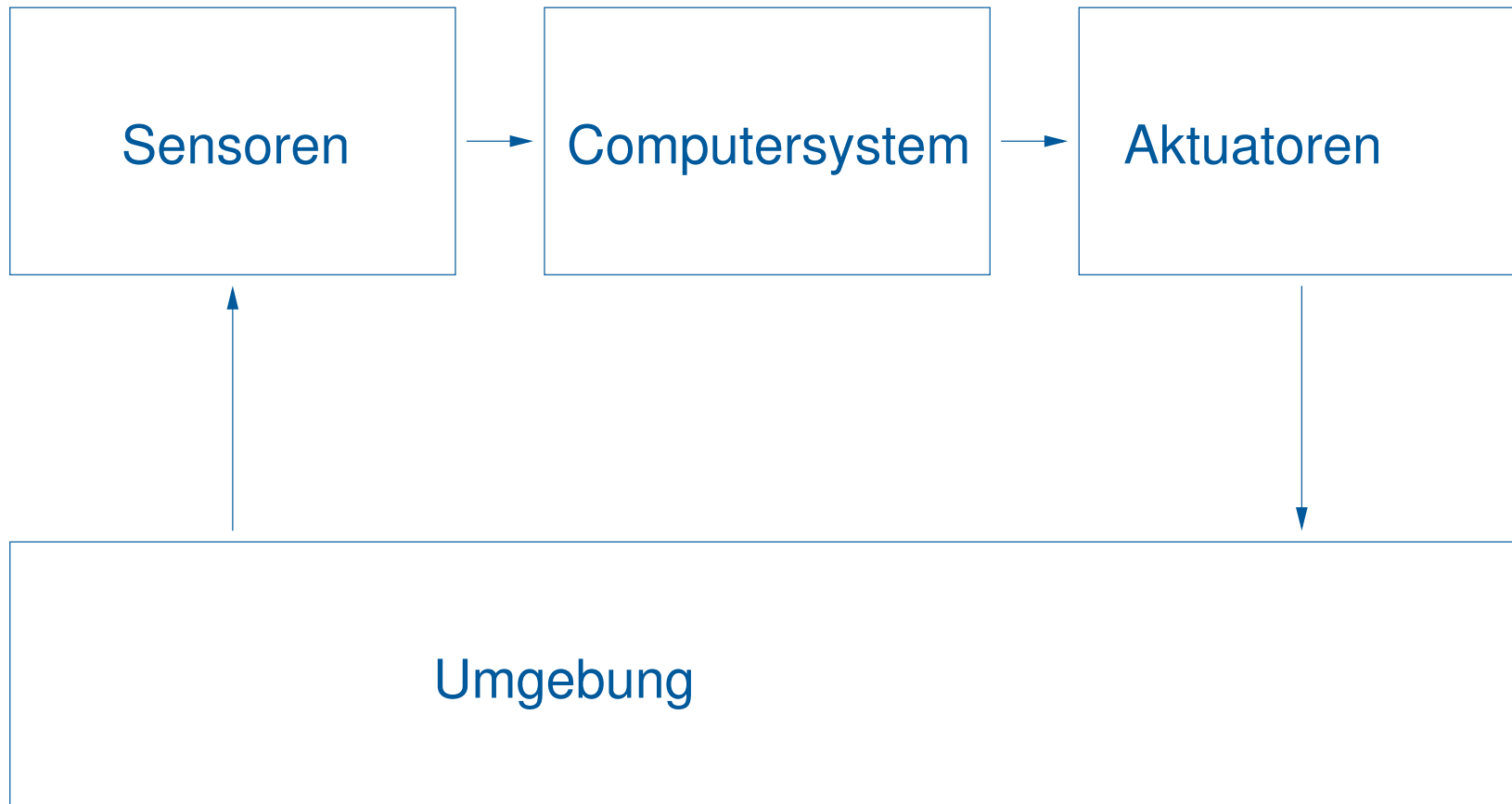
- Was sind zeitkritische Tasks?
 - Bremsen
 - Lenken
 - (Beschleunigen)
- Was sind ihre Deadlines?
 - Hängen von der Situation ab:
 - * Sonntag, 6 Uhr morgens, auf leerer Straße
 - * Werktags, 16 Uhr, Innenstadtverkehr
 - Task-Deadlines in Echtzeitsystemen sind nicht konstant
- Was sind unkritische Tasks?

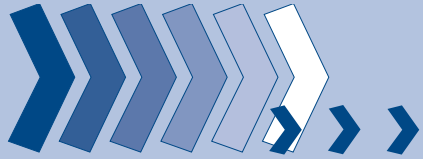


- Was sind zeitkritische Tasks?
 - Bremsen
 - Lenken
 - (Beschleunigen)
- Was sind ihre Deadlines?
 - Hängen von der Situation ab:
 - * Sonntag, 6 Uhr morgens, auf leerer Straße
 - * Werktags, 16 Uhr, Innenstadtverkehr
 - Task-Deadlines in Echtzeitsystemen sind nicht konstant
- Was sind unkritische Tasks?
 - Radio anschalten
 - Heizung bedienen
 - Licht einschalten
 - Aber! Auch diese Tasks können “weiche” Deadlines haben.

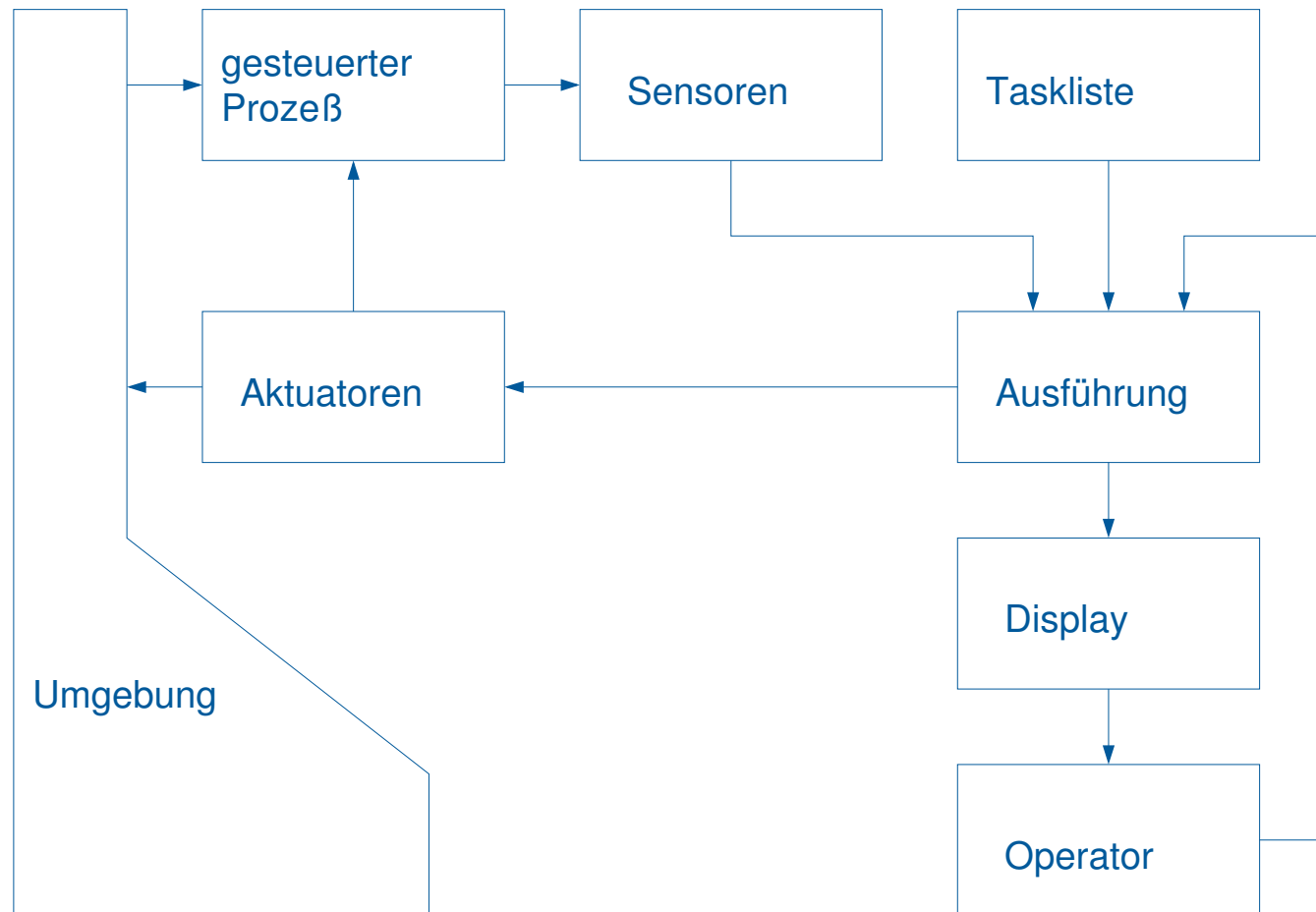


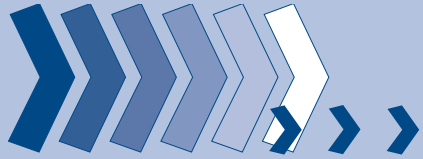
Struktur einer Echtzeitanwendung





Struktur eines Echtzeitsystems

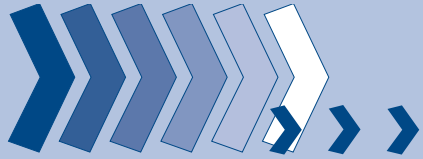




Charakterisierung von Tasks I

Nach ihrer Wichtigkeit

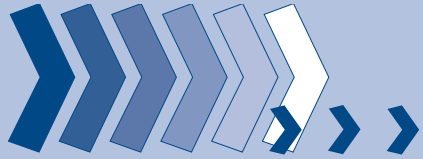
- kritische Tasks
Verpassen einer Deadline hat katastrophale Auswirkungen auf die Echtzeitanwendung
Ziel: Alle Deadlines müssen eingehalten werden
- unkritische Tasks
Verpassen von Deadlines ist unkritisch.
Ziel: Möglichst viele unkritische Tasks innerhalb der Deadline beenden



Charakterisierung von Tasks II

Nach dem Zeitpunkt ihres Auftretens

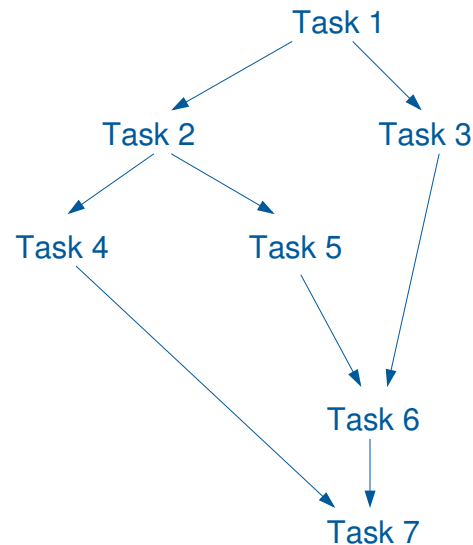
- periodische Tasks
Task wird periodisch alle x Zeiteinheiten (*Periode*) wiederholt ausgeführt
Beispiel: Steuerungstasks
- aperiodische Tasks
Task tritt unregelmäßig und unvorhersagbar auf
Beispiel: Alarmierungstasks
Spezialfall: Sporadische Tasks — zeitlicher Abstand zwischen zwei Auftreten ist begrenzt

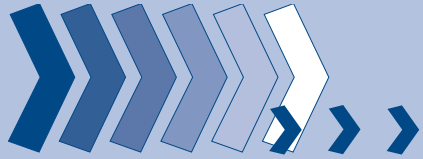


Charakterisierung von Tasks III

Nach Abhängigkeiten von anderen Tasks

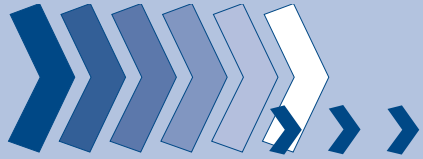
- Unabhängige Tasks
können in jeder beliebigen Reihenfolge ausgeführt werden
- Abhängige Tasks
Abhängigkeiten sind in einem Precedencegraphen darstellbar als “vorher”-Relation





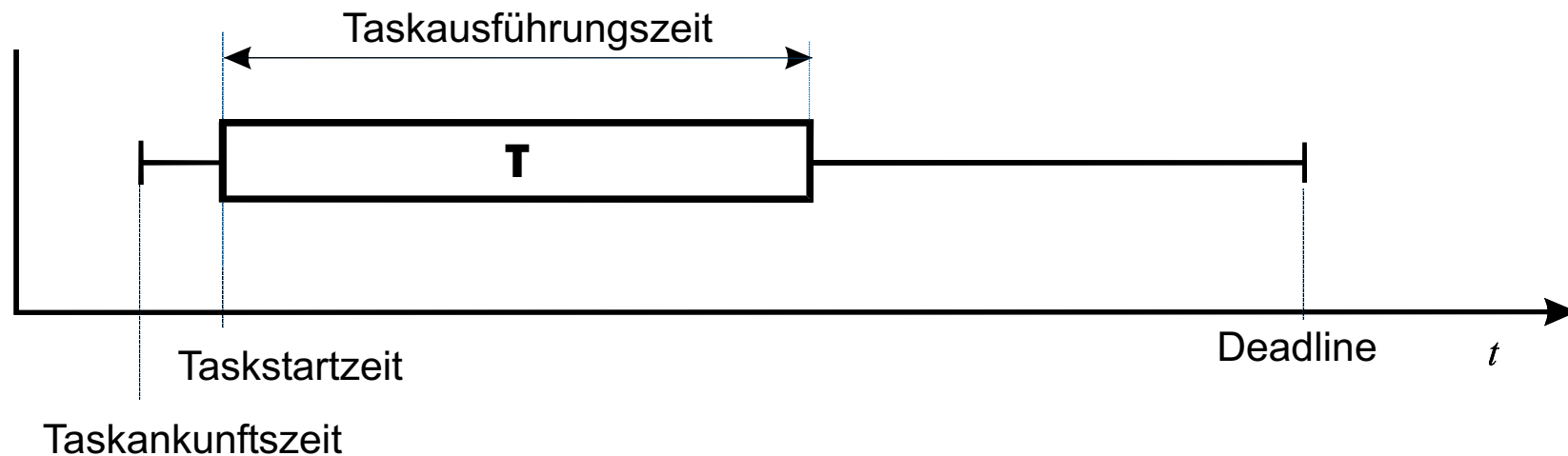
Periodisches Taskmodell

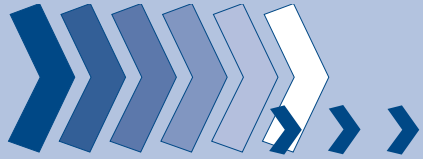
- Ausgangsidee: Viele Echtzeitanwendungen erfordern periodische Tasks
- Annahme: Tasks sind unabhängig
- Vorteil: Annahme von Periodizität und Unabhängigkeit führt zu Wissen über die zukünftige Belastung des Systems
- Problem: Es gibt auch aperiodische Tasks, die behandelt werden müssen!



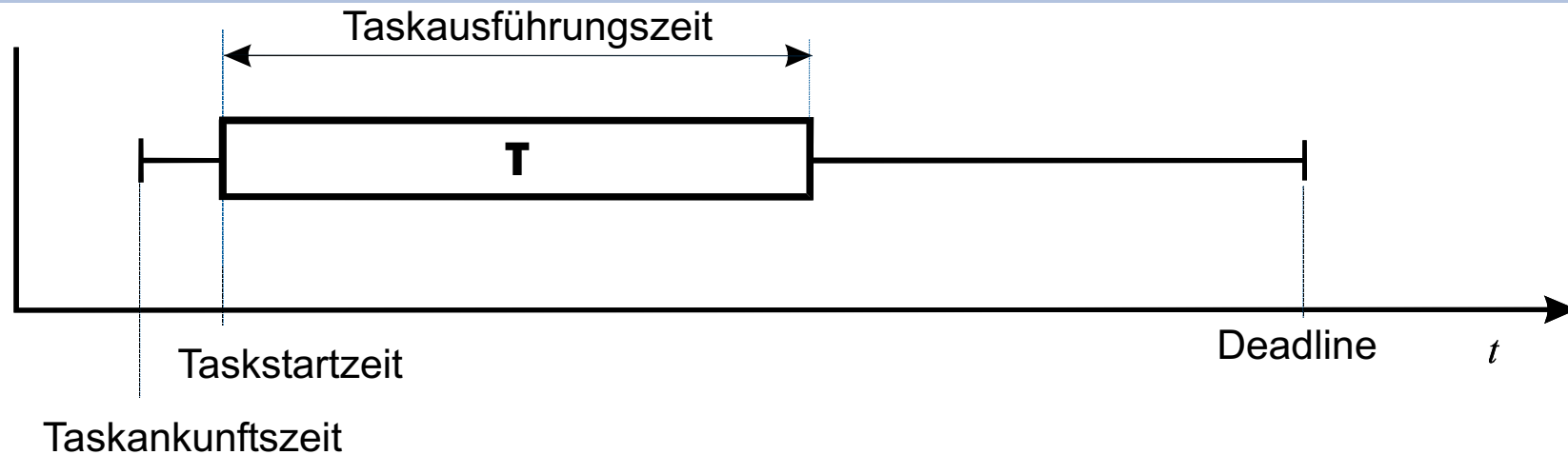
Parameter einer Task I

- Taskankunft r_i
- Taskstartzeit
- Ausführungszeit e_i
- Deadline D_i
- Periode P_i
- Maß für "Wichtigkeit"

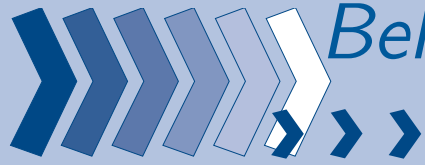




Parameter einer Task II

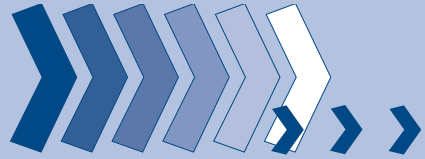


- Auslastung (Utilization): $U_i = \frac{e_i}{P_i}$
Bemerkung: $\forall i : U_i \leq 1$
- Jitter
 - exakte Periodizität ist real nicht immer erreichbar
 - Schwankungen der Ankunftszeit heissen *Jitter*
- Unterbrechbarkeit
Darf die Abarbeitung der Task unterbrochen werden?



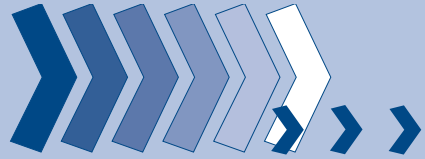
Behandlung von aperiodischen und sporadischen Tasks

- Minimale Zeit zwischen der Ankünften der Task muß bekannt sein (*minimal interarrival time*)
- Task wird im Modell als periodisch mit einer Periode entsprechend der minimal interarrival time angenommen
- Nachteil: Ausführung findet nicht in jeder Periode statt, damit wird Last als höher angenommen, als sie ist
- Vorteil: Periodisches Taskmodell ist anwendbar, geringes Vorwissen über die Task genügt



Generelles Problem: Aufteilung von Ressourcen an konkurrierende “Verbraucher”
Ressourcen bei Echtzeitsystemen (u.a.):

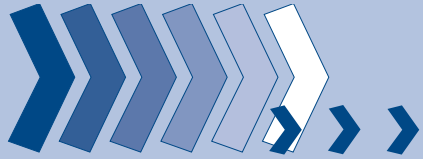
- CPU-Zeit (klassisches Schedulingproblem)
- Speicher
- I/O (Zugriff auf Geräte)
- Bandbreite von Netzwerkanbindungen
- Energie



klassische (nicht Echtzeit) Ziele:

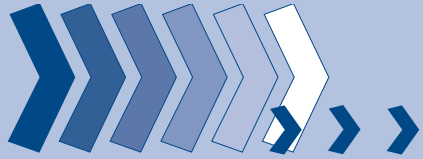
- Hohe Auslastung des Systems
- Fairness
- keine Starvation (lang anhaltende Verdrängung einzelner Tasks)
- Antwortverhalten (kurze Reaktionszeit)
- hoher Durchsatz (abgearbeitete Prozesse pro Zeiteinheit)

→ So nicht anwendbar bei Echtzeitsystemen!



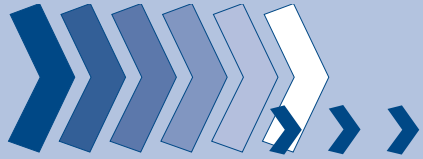
Schedulingstrategien für Nicht-Echtzeit I

- First come first serve (FCFS)
 - Prozeß kommt (arrival, Ankunft) - wird ans Ende der Liste gestellt
 - Aktueller Prozeß fertig - nächster aus Liste wird bis zur Fertigstellung abgearbeitet
 - effizient, aber lange Prozesse blockieren andere
- Shortest job first (SJF)
 - Wissen über Laufzeit des Prozessen
 - Prozeß kommt - entsprechend Laufzeit in Liste eingeordnet
 - preemptive - non preemptive (unterbrechend - nicht unterbrechend)
 - gute response times, aber Wissen notwendig (oft nicht vorhanden)



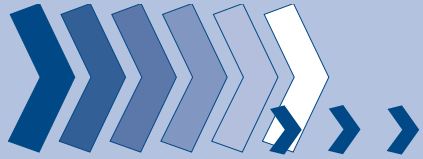
Schedulingstrategien für Nicht-Echtzeit II

- Round robin (RR)
 - wie FCFS mit preemption
 - Prozesse laufen ohne Unterbrechung eine bestimmte Zeitdauer (quantum)
 - eben laufender Prozeß wird ans Ende der ready-Liste eingeordnet
- Prioritätsscheduling
 - ready-Liste wird nach Prioritäten sortiert
 - statisch: Prioritäten werden einmal gesetzt und bleiben gleich
 - dynamisch: Prioritäten werden zur Laufzeit modifiziert
 - häufig: aging (Alterung) → länger laufende Prozesse verlieren hohe Priorität



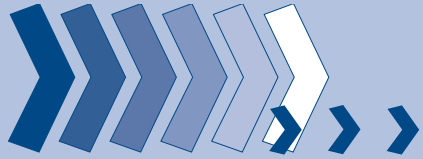
Verfahren des Nicht-Echtzeitscheduling können nicht übernommen werden:

- Deadlines müssen erfüllt werden
- Tasks sind verschieden wichtig → Keine Fairness
- Kurze Reaktionszeiten genügen nicht, Zeiten müssen garantiert sein
- Weitere Parameter sind wichtig:
 - Deadline
 - Periode
 - Abhängigkeiten von anderen Tasks (Taskgraph)



Schedule kann erzeugt werden:

- Vor Laufzeit des Echtzeitsystems: *offline-scheduling*
 - unflexibel gegenüber Änderungen
 - maximale Auslastung stets erreichbar
 - kaum Aufwand zur Laufzeit
- Während der Laufzeit des Systems: *online-scheduling*
 - flexibel gegenüber Änderungen
 - erreichbare Auslastung abhängig vom Verfahren
 - Aufwand zur Laufzeit höher
 - meist prioritätenbasiert



Prioritätenbasiertes Echtzeitscheduling

Zuweisung der Prioritäten erfolgt:

- statisch (keine Änderung während des Ablaufes)
 - Einfache und einmalige Berechnung bei Taskankunft
 - Festlegung anhand statischer Parameter der Task, beispielsweise Periodenlänge
 - Beispiel: RMS (*rate monotonic scheduling*)
- dynamisch
 - dynamische Neuberechnung der Prioritäten
 - Festlegung in Abhängigkeit von dynamischen Parametern der Task wie der aktuellen Deadline
 - Beispiel: EDF (*earliest deadline first*)