Echtzeitsysteme

Struktureller Aufbau von Echtzeitanwendungen

Lehrstuhl Informatik 4

08. November 2012

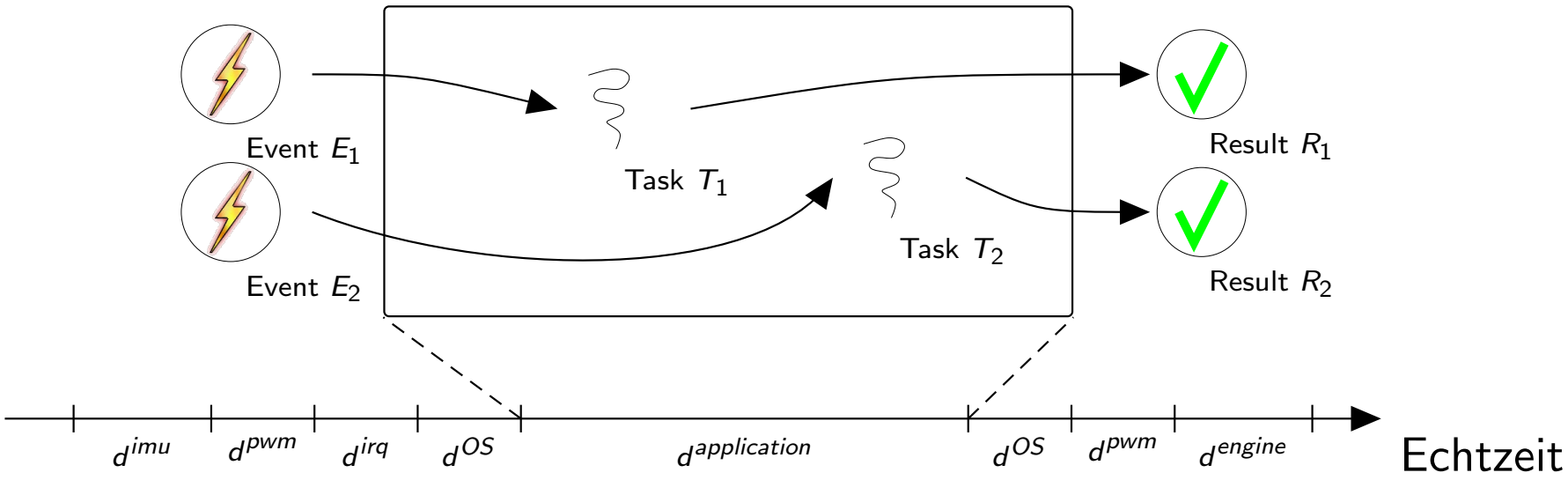
Gliederung

Fragestellungen

- Was sind die grundlegenden Bestandteile einer Echtzeitanwendung?
  - Was ist eine Aufgabe, was ein Ereignis und was ein Arbeitsauftrag?
- Wie bildet man Aufgaben auf das kontrollierende Rechensystem ab?
  - Wie funktioniert die Ablaufsteuerung?
  - Welche Kosten verursacht sie?
- Wie hängen das physikalische Objekt und die Echtzeitanwendung zeitlich zusammen?
  - Welche grundlegenden Zeitparameter gibt es?
  - Wie hängen Auslösezeit, Termin und Ausführungszeit zusammen?
- Was versteht man unter dem Begriff Planbarkeit?
  - Wie stellt man die Rechtzeitigkeit einer Echtzeitanwendung sicher?

Gliederung

Strukturelle Elemente einer Echtzeitanwendung



- Echtzeitanwendungen bestehen aus Aufgaben T_i (engl. *tasks*).
- Aufgaben werden durch Ereignisse E_i (engl. *events*) aktiviert.
- Aufgaben stellen Ergebnisse R_i (engl. *results*) bereit.

Aufgabe (engl. *task*)

Sichtweise: zeitgesteuerte Systeme

> *A task is the execution of a sequential program. It starts with reading of the input data and of the internal state of the task, and terminates with the production of the results and updating the internal state. [1, S. 75]*

Sichtweise: ereignisgesteuerte Systeme

> *We call each unit of work that is scheduled and executed by the system a job and a set of related jobs which jointly provide some system function a task. [2, S. 26]*
>
> *A real-time task is an executable entity of work which, at a minimum, is characterized by a worst case execution time and a time constraint. [4, S. 13]*

Einfache und komplexe Aufgaben

einfache Aufgabe (engl. *simple task*) ohne Synchronisationspunkt

- läuft durch, ohne zu blockieren
- unabhängig vom Fortschritt anderer Aufgaben
- ist von wichtigeren Aufgaben ggf. verdrängbar

☞ einfache Aufgaben können lokal betrachtet werden

- sie wird nicht von andere Aufgaben beeinflusst

komplexe Aufgabe (engl. *complex task*) mit Synchronisationspunkt

- erwartet die Zuteilung von Betriebsmitteln
  - wiederverwendbare und/oder konsumierbare BM
- hängt ab von der Bearbeitung anderer Aufgaben

☞ komplexe Aufgaben müssen global betrachtet werden

- im Verbund mit allen kooperierenden Aufgaben

Ereignis (engl. *event*)

> *An event is a change of state, occuring at an instant. [3, S. 10]*

Ein Ereignis ist ...

bestimmt, falls sein Auftreten als Funktion der physikalischen Zeit beschrieben werden kann, und

ungewiss, falls dies nicht zutrifft.

Ereignisse lösen (engl. *trigger*) Arbeitsaufträge aus:

event trigger Ereignisse werden von Zustandsänderungen in der physikalischen Umwelt oder dem kontrollierenden Rechensystem abgeleitet.

time trigger Ereignisse rühren ausschließlich vom Vorranschreiten der physikalischen Zeit her.[1]

[1] Das Vorranschreiten der Zeit stellt natürlich auch eine Zustandsänderung in der physikalischen Umwelt dar.

Arbeitsauftrag (engl. *job*)

> *We call each unit of work that is scheduled and executed by the system a job and a set of related jobs which jointly provide some system function a task. [2, S. 26, s. Folie III-2/6]*
>
> *A job is an instance of a task. [4, S. 26, s. Folie III-2/6]*

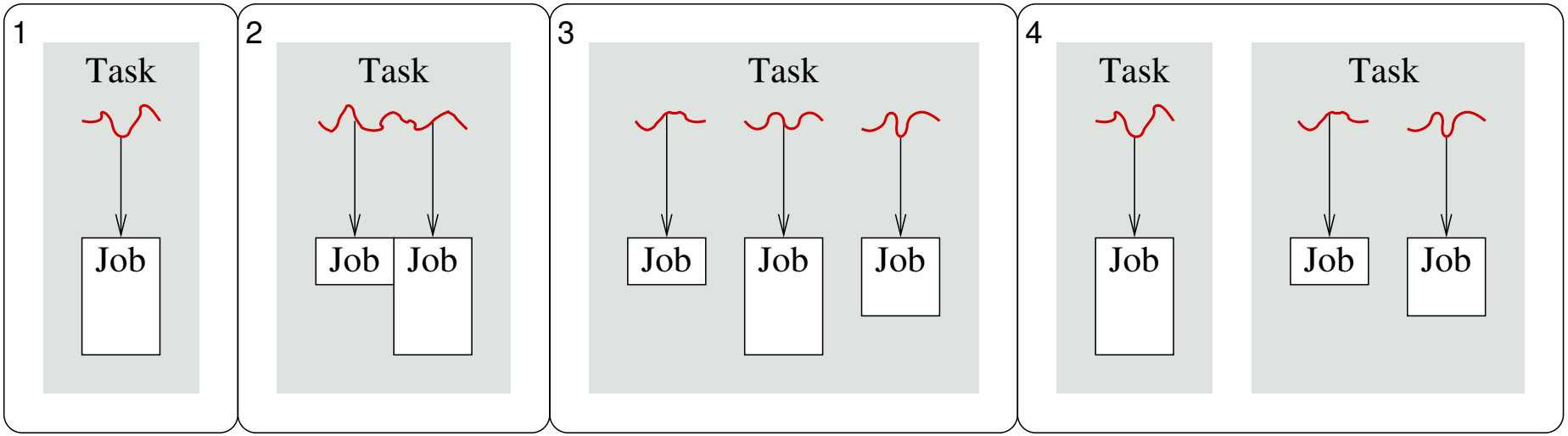
Tritt das Ereignis E_i ein und aktiviert die Aufgabe T_i, werden einer oder mehrere Arbeitsaufträge $J_{i,j}$ dieser Aufgabe ausgelöst.

Einplanungseinheit ist der Arbeitsauftrag und nicht der Faden, weil...

- die rechtzeitige Fertigstellung von Arbeitsaufträgen zu garantieren ist
- alle Arbeitsaufträge von nur einem Faden ausgeführt werden könnten
- bei einfädigen Systemen Fadeneinplanung irrelevant und sinnlos ist

Aufgabe vs. Arbeitsauftrag vs. Faden

Konfigurationsbeispiele für jeweils ein Rechensystem



1. eine einfädige Aufgabe, ein Arbeitsauftrag
   - häufig verwendete Zuordnung (vgl. Folie III-2/9, [4, S. 26])
2. eine einfädige Aufgabe, zwei Arbeitsaufträge
3. eine mehrfädige Aufgabe, drei Arbeitsaufträge
4. zwei Aufgaben (ein- und mehrfädig), drei Arbeitsaufträge

Gliederung

Ablaufsteuerung als zweiphasiger Prozess

Ablaufsteuerung

Wann wird welcher Arbeitsauftrag auf welcher Recheneinheit ausgeführt?

☞ Ziel ist die rechtzeitige Fertigstellung der Arbeitsaufträge

- d. h. die Einhaltung ihrer jeweiligen Termine (s. Folie III-2/26)
- die Ablaufsteuerung hat hier signifikanten Einfluss

Phase 1: Aufgaben/Arbeitsaufträge $\mapsto$ Fäden

- Abbildung erfolgt statisch während der Entwicklung
- impliziert Latenzen durch Sequentialisierung
- entspricht einer Allokation von Betriebsmitteln
  - Fäden können hier als Betriebsmittel aufgefasst werden
- oft wird eine 1:1-Abbildung von Arbeitsaufträgen auf Fäden angenommen

Ablaufsteuerung als zweiphasiger Prozess (Forts.)

Phase 2: Fäden $\mapsto$ Prozessor

- Nutzung des Prozessors im zeitlichen Mehrfachbetrieb
  - engl. *temporal multiplexing*
  - meist gibt es mehr Fäden als Prozessoren
- ist als Einplanung (engl. *scheduling*) bekannt
- Abbildung erfolgt statisch oder dynamisch
  - statisch $\rightsquigarrow$ der komplette Ablauf wird vorab festgelegt
  - dynamisch $\rightsquigarrow$ der Prozessor wird zur Laufzeit zugeteilt

☞ zeitlicher Mehrfachbetrieb impliziert ggf. den dynamischen Wechsel zwischen verschiedenen Fäden $\rightsquigarrow$ zusätzlicher Aufwand

- Mechanismus zum Abfertigen einzelner Fäden
- Strategie zur Auswahl des nächsten laufbereiten Fadens

☞ in Echtzeitsystemen nicht vernachlässigbar!

Ausführungsstrang

Physikalische Einheit der Einplanung (engl. *unit of scheduling*)

Abstraktion (des Betriebssystems) von einem beliebigen Programm in Abarbeitung bzw. Ausführung ist der Prozess

- im Prozess existieren ggf. mehrere Ausführungsstränge gleichzeitig
  - nebenläufiges Programm (engl. *concurrent program*)
  - durchzogen mit mehr als einen Programmfaden (engl. *thread*)
- der Kontext eines Fadens manifestiert sich im Prozessorstatus
  - physisch die Inhalte der Arbeits- und Statusregister der CPU
  - ggf. erweitert um Segment-/Seitendeskriptoren von MMU bzw. TLB
- Einlastung (engl. *dispatching*) eines Fadens bedeutet Kontextwechsel

Prozessinstanz

Ein- oder mehrfädiges Programm, mit oder ohne eigenem Adressraum — oder aber bloß „Prozeduraktivierung“ eines übergeordneten Programms, bei kooperativer Einplanung und Verzicht auf Synchronisationspunkte.

Verwaltungsgemeinkosten (engl. *overhead*)

Eine Frage der Repräsentation von Aufgabe und Arbeitsauftrag

1. einfädige Aufgabe $\rightsquigarrow$ fliegengewichtig
   - $O(\mathit{Prozeduraufruf})$
   - Auf- und Abbau vom Aktivierungsblock (engl. *activation record*)
2. mehrfädige Aufgabe
   1. gemeinsamer Adressraum $\rightsquigarrow$ federgewichtig
      - $O(\mathit{Fadenwechsel}) + O(1.)$
      - Austausch der Inhalte von Arbeits-/Statusregister der CPU
   2. separierter Betriebssystemkern $\rightsquigarrow$ leichtgewichtig
      - $O(\mathit{Systemaufruf}) + O(2.1.)$
      - Behandlung der synchronen Programmunterbrechung (engl. *trap*)
   3. getrennte Adressräume $\rightsquigarrow$ schwergewichtig
      - $O(\mathit{Adressraumwechsel}) + O(2.2.)$
      - Löschen/Laden vom Zwischenspeicher (engl. *cache*) der MMU

- bis auf $O(2.3)$ haben alle anderen Fälle konstanten Aufwand

Verdrängbare Aufgabe

Unterbrechung und Wiederaufnahme der Bearbeitung eines Arbeitsauftrags

Jobs, die ablaufen, können Verdrängung (engl. *preemption*) erleiden

- dem Faden des laufenden Arbeitsauftrags wird die CPU entzogen:
  1. eine asynchrone Programmunterbrechung (engl. *interrupt*) tritt auf
  2. der unterbrochene Faden wird als laufbereit (erneut) eingeplant
  3. ein anderer laufbereiter Faden wird ausgewählt und eingelastet
- eine Systemfunktion, die Bedingungen zum korrekten Ablauf stellt:
  - asynchrone Programmunterbrechungen müssen möglich sein
  - die Behandlungsroutine muss den Planer (engl. *scheduler*) aktivieren
  - der Planer muss verdrängend arbeiten (engl. *preemptive scheduling*)
  - mindestens ein anderer laufbereiter Faden muss zur Verfügung stehen

Verdrängung ist als nicht-funktionale Systemeigenschaft anzusehen

- die an anderen Programmstellen Synchronisationsbedarf impliziert
- die transparent für die betroffene Aufgabe sein muss

Verdrängbare Aufgabe (Forts.)

Verarbeitung der Ausführungsstränge frei von Seiteneffekten

Transparenz (engl. *transparency*) von Verdrängung meint zweierlei:

1. der Zustand eines unterbrochenen Fadens ist invariant
   - erfordert Sicherung und Wiederherstellung des Prozessorstatus
   - Maßnahmen zur Einlastung (engl. *dispatching*) von Fäden
2. die verzögerte Ausführung des Fadens verletzt keine Fristen
   - fordert Vergabe, Überwachung und Einhaltung von Dringlichkeiten
   - jeder Faden ist von statischer/dynamischer Priorität (engl. *priority*)
   - Maßnahmen zur Einplanung (engl. *scheduling*) von Fäden

- einfache nicht-verdrängbare Aufgaben können auf 1. verzichten
  - einfache verdrängbare oder komplexe jedoch nicht
- für Echtzeitrechensysteme ist 2. ggf. sogar verzichtbar
  - sofern die Umgebung keine harten Echtzeitbedingungen vorgibt

Trennung von Belangen

Planung des zeitlichen Ablaufs und Abfertigung

Einplanung (engl. *scheduling*) $\mapsto$ Strategie

- Erstellung des Ablaufplans von Arbeitsaufträgen
  - Festlegung einer Einlastungsreihenfolge
- in Bezug auf die Aufgabenbearbeitung geschieht dies:

  entkoppelt (engl. *off-line*) $\rightsquigarrow$ statisch, vor Laufzeit

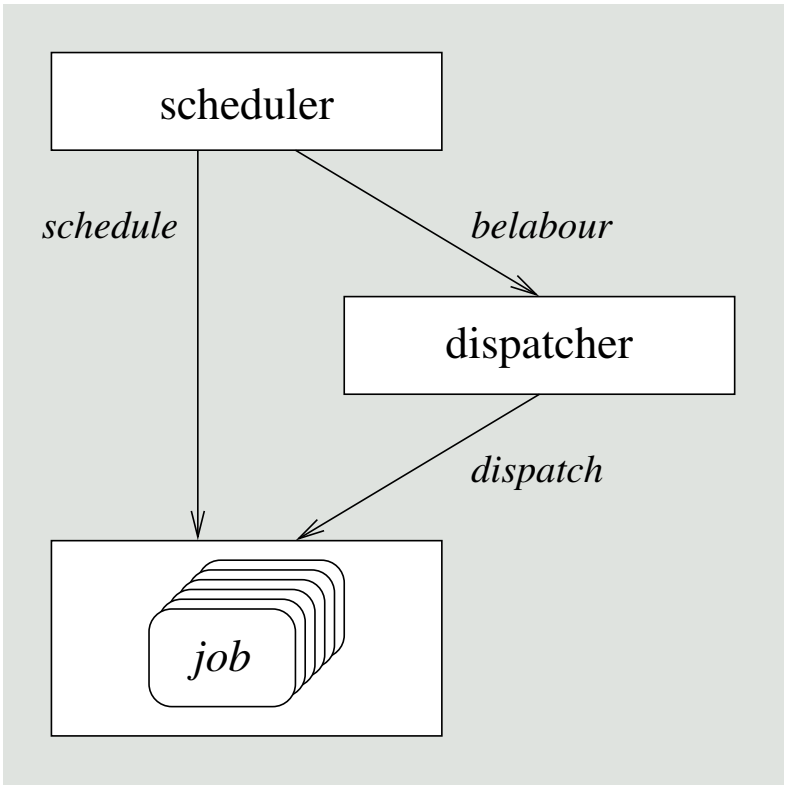
  gekoppelt (engl. *on-line*) $\rightsquigarrow$ dynamisch, zur Laufzeit[2]

Einlastung (engl. *dispatching*) $\mapsto$ Mechanismus

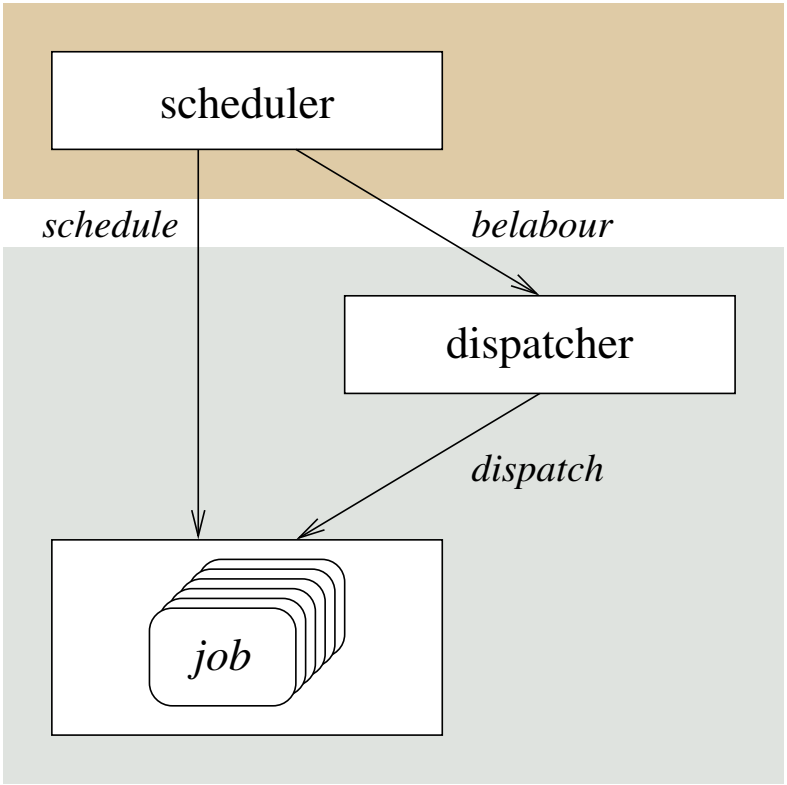
- Abarbeitung des Ablaufplans von Arbeitsaufträgen
  - Umsetzung der Einplanungsentscheidungen
- ist immer gekoppelt mit der Aufgabenbearbeitung
  - Ablaufpläne können nur *online* befolgt werden

[2] Vorlage ist ggf. ein vor Beginn der Aufgabenbearbeitung statisch erstellter Ablaufplan, der während der Aufgabenbearbeitung dynamisch fortgeschrieben wird.

Einplanung und Einlastung



gekoppeltes System



entkoppeltes System

- zeit- und örtlich gekoppelt
  - zur Laufzeit
  - integriert in einem System
    - auf einem Rechner
- zeit- oder örtlich entkoppelt
  - vor und zur Laufzeit
  - separiert in zwei Systeme
    - ggf. auf zwei Rechner

Einplanungszeitpunkte

Adaptierbarkeit (engl. *adaptability*) vs. Vorhersagbarkeit (engl. *predictability*)

on-line scheduling (zur Laufzeit), kommt ohne *à priori* Wissen aus

- einzige Option bei unbekannter zukünftiger Auslastung
  - Lastparameter sind erst zur Joblaufzeit bekannt
- die getroffenen Entscheidungen sind häufig nur suboptimal
  - eingeschränkte Fähigkeit, Betriebsmittel maximal zu nutzen
- ermöglicht/unterstützt jedoch ein flexibles System

off-line scheduling (vor Laufzeit), benötigt *à priori* Wissen

- Voraussetzung ist ein deterministisches System, d.h.:
  - alle Lastparameter sind vor Joblaufzeit bekannt
  - ein fester Satz von Systemfunktionen ist gegeben
- zur Laufzeit ist kein $\mathcal{NP}$-schweres Problem mehr zu lösen
  - d.h, einen Ablaufplan zu finden, der alle Task/Job-Fristen einhält
- Änderungen am System bedeuten Neuberechnung des Ablaufplans
  - dies gilt für alle Änderungen an Software *und* Hardware

Grundsätzliche Verfahren

Vorangetrieben durch interne oder externe Ereignisse

taktgesteuert (engl. *clock-driven*, auch *time-driven*) ✓

- Einlastung nur zu festen Zeitpunkten
  - vorgegeben durch das Echtzeitrechensystem
- statische (entkoppelte) Einplanung

reihum gewichtet (engl. *weighted round-robin*)

- Echtzeitverkehr in Hochgeschwindigkeitsnetzen
  - im Koppelnetz (engl. *switched network*)
- untypisch für die Einplanung von CPU-Jobs

vorranggesteuert (engl. *priority-driven*, auch *event-driven*) ✓

- Einlastung zu Ereigniszeitpunkten
  - vorgegeben durch das kontrollierte Objekt
- dynamische (gekoppelte) Einplanung

Taktsteuerung

Zeitgesteuertes (engl. *time-triggered*) System

Einlastungszeitpunkte von Arbeitsaufträgen wurden *à priori* bestimmt

- alle Parameter aller Arbeitsaufträge sind *off-line* bekannt
  - WCET, Betriebsmittelbedarf (z.B. Speicher, Fäden, Energie), ...
- zur Laufzeit anfallende Verwaltungsgemeinkosten sind minimal

Einlastung der Arbeitsaufträge erfolgt in variablen oder festen Intervallen

- im variablen Fall wird ein Zeitgeber (engl. *timer*) mit der Länge des jeweils einzulastenden Arbeitsauftrags programmiert $\mapsto$ WCET
  - jeder Zeitablauf bewirkt eine asynchrone Programmunterbrechung
  - als Folge findet die Einlastung des nächsten Arbeitsauftrags statt
- im festen Fall liefert der Zeitgeber regelmäßige Unterbrechungen
  - ein festes Zeitraster liegt über die Ausführung der Arbeitsaufträge
  - dient z.B. dem Abfragen (engl. *polling*) von Sensoren/Geräten

Vorrangsteuerung

Ereignisgesteuertes (engl. *event-triggered*) System

Einplanung und Einlastung laufen gekoppelt ab $\mapsto$ Ereigniszeitpunkte

- asynchrone Programmunterbrechungen: Hardwareereignisse
  - Zeitsignal, Bereitstellung von Sensordaten, Beendigung von E/A
- Synchronisationspunkte: ein-/mehrseitige Synchronisation
  - Schlossvariable, Semaphor, Monitor

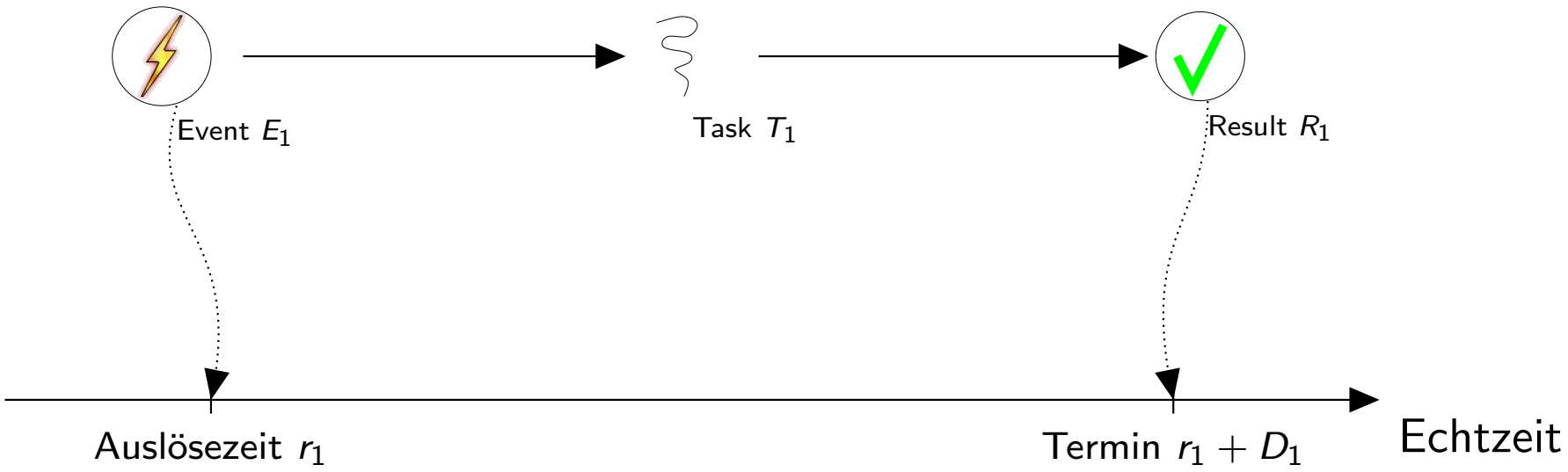
Ereignisse haben Prioritäten, die Dringlichkeiten zum Ausdruck bringen

- Prioritäten werden *off-line* vergeben und ggf. *on-line* fortgeschrieben
  - Arbeitsaufträge haben eine statische oder dynamische Priorität
- die Zuteilung von Betriebsmitteln erfolgt prioritätsorientiert
  - Arbeitsaufträge höherer Priorität haben Vorrang
- Betriebsmittel (insb. CPU) bleiben niemals absichtlich ungenutzt
  - im Gegensatz zur Taktsteuerung, die Betriebsmittel brach liegen lässt

Gliederung

Punkte auf der Echtzeitachse

Bereitstellung und Erfüllung



- Das Ereignis E_i löst zum Auslösezeitpunkt r_i den Arbeitsauftrag $J_{i,j}$ der Aufgabe T_i aus.
- Das Ergebnis R_i muss bis zum Termin D_i vorliegen.

Punkte auf der Echtzeitachse (Forts.)

Bereitstellung und Erfüllung

Auslösezeit (engl. *release time*) Zeitpunkt, zu dem ein Arbeitsauftrag zur Ausführung bereitgestellt wird

- ab dem Zeitpunkt ist Einlastung des betreffenden Jobs möglich
  - vorausgesetzt, gewisse Abhängigkeitsbedingungen[3] sind erfüllt
- ggf. verzögern Einplanung/Koordinierung die Einlastung des Jobs

Termin (engl. *deadline*) Zeitpunkt, zu dem ein Arbeitsauftrag seine Ausführung beendet haben soll bzw. muss

- wird differenziert nach der Art seines Bezugszeitpunktes:

  relativ (engl. *relative deadline*) zur Auslösezeit oder
  absolut (engl. *absolute deadline*) als Echtzeit
  - absoluter Termin = Auslösezeit r_i + relativer Termin D_i
- ist je nach Anforderung weich, fest oder hart
  - gibt mit dem Wert ∞ keine Frist für den betreffenden Job vor

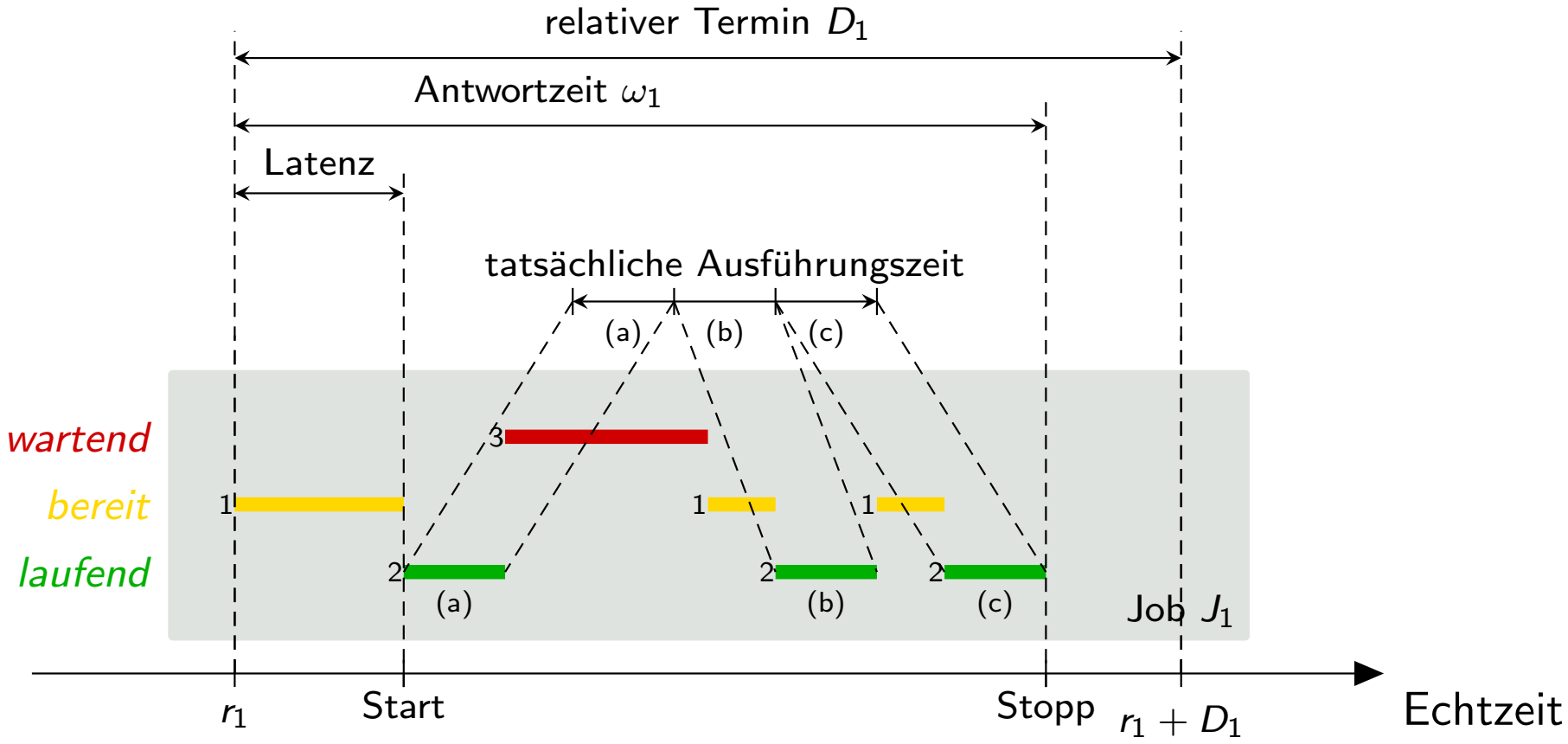
[3] Daten-/Kontrollfluss bzgl. kontrolliertem Objekt/anderer Arbeitsaufträge.

Jobphasen auf der Echtzeitachse

Ablaufzustände eines Fadens

Arbeitsauftrag einer komplexen Aufgabe

- (1) Einplanung, (2) Einlastung, (3) Synchronisation



Intervalle auf der Echtzeitachse

Ausführung und Freiraum

Latenz Zeitdauer zwischen Auslösezeit und dem Beginn der Abarbeitung

tatsächliche Ausführungszeit (engl. *elapsed time*) durch den ausgeführten Arbeitsauftrag beanspruchte Rechenzeit

- wird durch die maximale Ausführungszeit e_i der Aufgabe T_i beschränkt (engl. **w***orst* **c***ase* **e***xecution* **t***ime*, WCET)
- die WCET ist prinzipiell unabhängig von anderen Arbeitsaufträgen

Antwortzeit ω_i (engl. *response time*) Zeitdauer zwischen Auslösung und Terminierung des Arbeitsauftrags (genauer: bis das Ergebnis bereitgestellt wurde)

- die maximal erlaubte Antwortzeit wird durch einen relativen Termin beschränkt: Antwortzeit $\omega_i \leq$ relativer Termin D_i

Schlupfzeit

Der zeitliche Spielraum eines Arbeitsauftrags

Schlupfzeit $\sigma_{J_i}(t)$ (engl. *slack time*) Zeitdauer zwischen dem voraussichtlichen Terminationszeitpunkt und dem Fristablauf eines sich in Bearbeitung befindlichen Arbeitsauftrags

- unter der Annahme, dass der Arbeitsauftrag nicht mehr blockiert oder unterbrochen wird

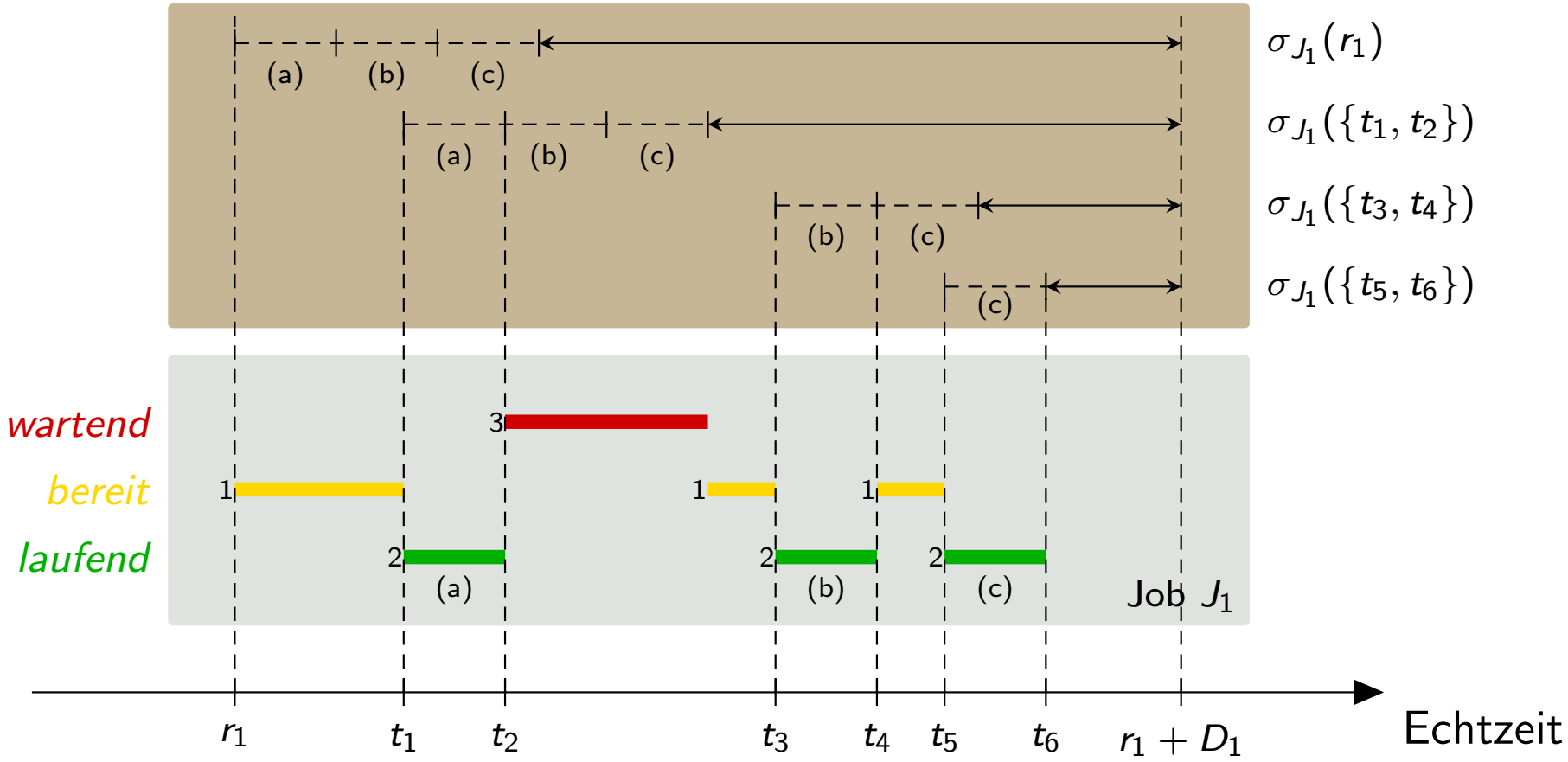
$$\begin{aligned} \sigma_{J_i}(t) &= r_i + D_i - t - \mathit{maturity}(J_i, t) \\ \mathit{maturity}(J_i, t) &= e_i - \mathit{elapsed\ time}(J_i, t) \end{aligned}$$

- Ziel: rechtzeitige, nicht möglichst schnelle Fertigstellung von Jobs
- gibt der Einplanung Spielraum zur Einlastung eines Jobs

Schlupfzeit (Forts.)

Der zeitliche Spielraum eines Arbeitsauftrags

- in Phasen der Untätigkeit verringert sich die Schlupfzeit
- während der Ausführung bleibt die Schlupfzeit konstant



Gliederung

Aufgabenstellung

Gegeben sei eine Menge Aufgaben T_i einer Echtzeitanwendung mit

- D_i dem relativen Termin (engl. *deadline*)
- e_i der maximalen Ausführungszeit (WCET)

der jeweiligen Aufgabe.

Fragestellung:

Ist diese Menge von Aufgaben zulässig (engl. *feasible* oder *schedulable*)?

Zulässigkeit

(engl. *feasibility* oder *schedulability*)

Ein Ablaufplan ist gültig (engl. *valid*), falls gewisse strukturelle Vorgaben eingehalten werden:

- jeder CPU gleichzeitig max. ein Arbeitsauftrag zugeteilt wird
- jeder Arbeitsauftrag gleichzeitig an max. eine CPU zugeteilt wird
- kein Arbeitsauftrag vor seinem Auslösezeitpunkt eingeplant wird
- einem Arbeitsauftrag entweder seine tatsächliche oder seine maximale Ausführungszeit zugeteilt wird
- alle (un)gerichteten Abhängigkeiten eingehalten werden

Ein Ablaufplan ist zulässig, falls

- der Ablaufplan gültig ist und
- alle Arbeitsaufträge termingerecht eingeplant werden.

$$\forall i : \text{maximale Antwortzeit } \omega_i \leq \text{relativer Termin } D_i$$

Zulässigkeit (Forts.)

Eine Menge von Aufgaben ist zulässig (engl. *feasible* oder *schedulable*)

- hinsichtlich eines Einplanungsalgorithmus,
- falls dieser Algorithmus einen zulässigen Ablaufplan erzeugt.

Die Entscheidung, ob eine Aufgabenmenge planbar ist, hängt somit

- vom verwendeten Einplanungsalgorithmus
- sowie von den Eigenschaften der Aufgaben ab.
  - z.B. periodisch, verdrängbar, frei von Abhängigkeiten, ...

☞ häufig schränken Algorithmen die Eigenschaften von Aufgaben ein

- dies vereinfacht die Frage der Zulässigkeit oft beträchtlich
- aufwändige Analysen können Einschränkungen lockern/aufheben

Klassifikation von Einplanungsalgorithmen

Optimalität (engl. *optimality*)

Ein Einplanungsalgorithmus ist optimal (engl. *optimal*) für eine gewisse Klasse von Aufgaben, falls er für eine Menge solcher Aufgaben einen zulässigen Ablaufplan findet, sofern ein zulässiger Ablaufplan existiert.

- Solch ein Algorithmus stellt eine Referenz dar.
  - Schafft es dieser Algorithmus nicht, schafft es keiner!
- Die (generelle) Zulässigkeit einer Menge von Aufgaben
  - kann auf die Zulässigkeit für diesen Algorithmus reduziert werden,
  - sofern ein entsprechendes Kriterium existiert.

Einhaltung von Terminen

Taktgesteuerte Systeme $\leadsto$ konstruktiv

- alle Lastparameter sind à priori bekannt
- die Konstruktion einer Ablauftabelle trägt ihnen Rechnung
- Abhängigkeiten können berücksichtigt werden

☞ alle Termine werden eingehalten

- wenn eine zulässige Ablauftabelle erzeugt werden kann

Vorranggesteuerte Systeme $\leadsto$ analytisch

- Lastparameter sind nicht vollständig bekannt
- Ablauf wird erst zur Laufzeit berechnet
- Abhängigkeiten müssen explizit gesichert werden

☞ Einhaltung von Terminen muss explizit überprüft werden

Gliederung

Resümee

Einplanungseinheit $\mapsto$ Prozedur, Faden und/oder Fadengruppe

- Aufgaben (*Tasks*) und Arbeitsaufträgen (*Jobs*)
- Verwaltungsgemeinkosten ein- und mehrfädiger Aufgaben
- Einplanung als zweiphasiger Prozess

Ablaufsteuerung Trennung von Belangen $\mapsto$ Strategie & Mechanismus

- Einplanung ist die Strategie, Einlastung ist der Mechanismus
- entkoppelt vs. gekoppelt, Taktsteuerung vs. Vorrangsteuerung

Zeitparameter sind Punkte und Intervalle auf der Echtzeitachse

- Auslösezeit, (absoluter) Termin
- Antwortzeit, relativer Termin, Schlupfzeit, Ausführungszeit

Planbarkeit sichert Rechtzeitigkeit der Echtzeitanwendung

- gültige und zulässige Ablaufpläne
- optimale Einplanungsalgorithmen
- konstruktive vs. analytische Überprüfung der Planbarkeit

Literaturverzeichnis

[1] Kopetz, H. :
Real-Time Systems: Design Principles for Distributed Embedded Applications.
Kluwer Academic Publishers, 1997. –
ISBN 0–7923–9894–7

[2] Liu, J. W. S.:
Real-Time Systems.
Prentice-Hall, Inc., 2000. –
ISBN 0–13–099651–3

[3] Obermaisser, R. :
Event-Triggered and Time-Triggered Control Paradigms.
Springer-Verlag, 2005. –
ISBN 0–387–23043–2

[4] Stankovic, J. A. ; Spuri, M. ; Ramamritham, K. ; Buttazzo, G. C.:
Deadline Scheduling for Real-Time Systems.
Kluwer Academic Publishers, 1998. –
ISBN 0–7923–8269–2