

# Verlässliche Echtzeitsysteme

## Verteilte Echtzeitsysteme

**Peter Ulbrich**

Friedrich-Alexander-Universität Erlangen-Nürnberg  
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)

[www4.informatik.uni-erlangen.de](http://www4.informatik.uni-erlangen.de)

30. Juni 2015



- Eure Meinung ist uns wichtig! (Lob und/oder Kritik)  
→ Bitte evaluiert Vorlesung und Übung

## Motivationsanreiz zur Evaluation



- **Traditionell:** Kaffee und Kekse in der letzten Vorlesung
- **Bedingung:**  $\geq 60\%$  der ausgegebenen TANs werden evaluiert!

- Eure Meinung ist uns wichtig! (Lob und/oder Kritik)  
→ Bitte evaluiert Vorlesung und Übung



Nach aktuellem Stand erst 2/13 TANs genutzt.

## Motivationsanreiz zur Evaluation



- **Traditionell:** Kaffee und Kekse in der letzten Vorlesung
- **Bedingung:**  $\geq 60\%$  der ausgegebenen TANs werden evaluiert!

- Wie lässt sich die Verlässlichkeit weiter steigern?
  - Verteilung von Funktionen?
  - Was bringt uns hier die Replikation?
  
- Wie muss die hierfür notwendige Kommunikationsinfrastruktur aussehen?
  - Wie lässt sich Fehlerausbreitung verhindern?
  - Zeit- vs. ereignisgesteuerte Kommunikation?
  
- Was passiert nach dem Auftreten eines (transienten) Fehlers?
  - Wie kann sich ein Replikat von einem Fehler erholen?
  - Wie lassen sich Knoten reintegrieren?

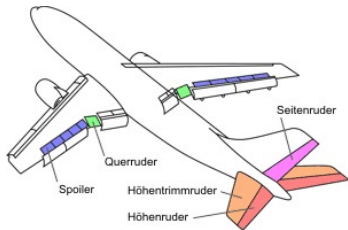


- 1 Überblick
- 2 **Verlässlichkeit durch Verteilung**
  - Beispiel: Flugsteuerung
  - Verteiltes Fly-By-Wire
- 3 Kommunikation
  - Problemstellung und Grundlagen
  - Zeitgesteuerte Kommunikationssysteme
- 4 Zustandswiederherstellung
  - Vorgehen bei der Reintegration
  - Fehlererholung
  - Interner Zustand
- 5 Zusammenfassung



- **Steuerflächen** (engl. *control surfaces*) und ihre Ansteuerung  
→ Elemente um die Lage des Flugzeugs im Raum zu ändern

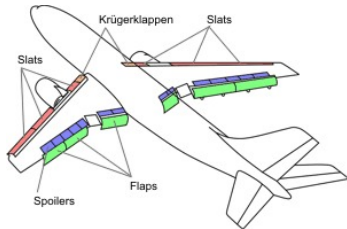
- **Primäre Steuerflächen** eines Flugzeugs:



- **Querruder** (engl. *ailerons*)  
→ Rollen um die Längsachse
- **Seitenruder** (engl. *rudder*)  
→ Gieren um die Hochachse
- **Höhenruder** (engl. *elevator*)  
→ Kippen um die Querachse
- **Störklappe** (engl. *spoiler*)  
→ Verlangsamen, unterstützt Gieren

- **Höhentrimmruder** (engl. *trimmable horizontal stabilizer, (THS)*)  
→ Höhensteuerung und Höhentrimmung

## ■ Sekundäre Steuerflächen und Auftriebshilfen:



### ■ Vorflügel (engl. *slats*)

→ Höherer Auftrieb durch mehr Wölbung

### ■ Krügerklappen (engl. *Kruger slats*)

→ Ähnlich dem Vorflügel, aber einfacher

### ■ Landeklappen (engl. *flaps*)

→ Auftriebshilfen für den Landeanflug

→ Ermöglichen verminderte Geschwindigkeit beim Landen

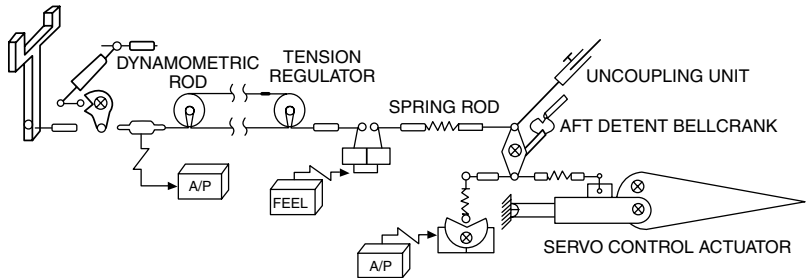
## ■ Vielzahl weiterer Typen (für Technikbegeisterte):

- Kippnasen, Spalt, Fowler, Spreiz-Klappen, Junkers-Doppelflügel, ...



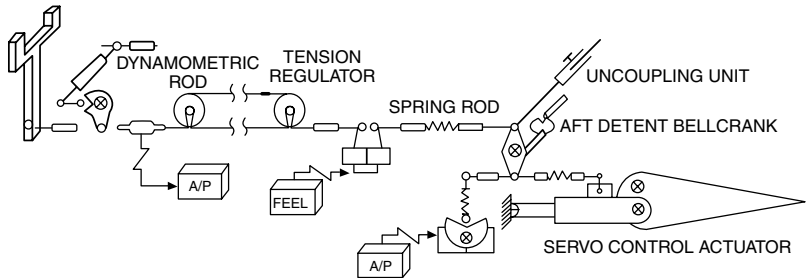
Flugzeuge haben i. d. R. nur eine Auswahl der genannten Steuerflächen



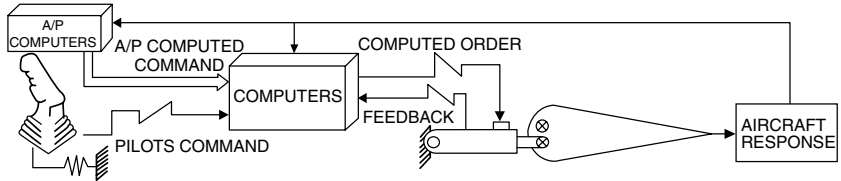


- **Mechanische Systeme**  $\leadsto$  direkte Verbindung zur Steuerfläche
  - Ein Lenkgestänge verbindet Steuerknüppel und Steuerflächen direkt
    - Am Steuerknüppel ist der Luftwiderstand an der Steuerfläche spürbar
    - $\rightarrow$  Problematisch bei großen Flugzeugen/hohen Geschwindigkeiten



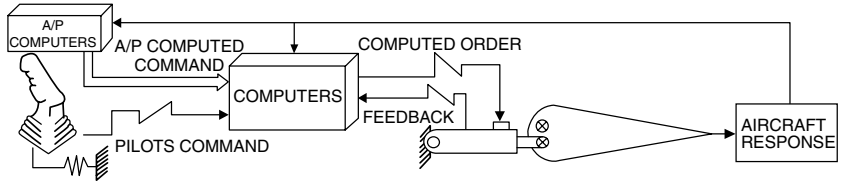


- **Mechanische Systeme**  $\leadsto$  direkte Verbindung zur Steuerfläche
  - Ein Lenkgestänge verbindet Steuerknüppel und Steuerflächen direkt
    - Am Steuerknüppel ist der Luftwiderstand an der Steuerfläche spürbar
    - $\rightarrow$  Problematisch bei großen Flugzeugen/hohen Geschwindigkeiten
- **Elektrische/hydraulische Systeme**  $\leadsto$  motorisierte Aktoren
  - Der Pilot steuert die Steuerfläche mithilfe von Aktoren an
    - Lenkbefehle öffnen Ventile oder setzen Servomotoren in Gang
    - Mechanische Ansteuerung erhält Verbindung
    - $\rightarrow$  **Servolenkung** welche den Piloten unterstützt



## ■ Elektrische/elektronische Ansteuerung $\leadsto$ Fly-by-Wire

- **Analog**  $\leadsto$  Steuerbefehle werden direkt auf elektrische Signale umgesetzt
- **Digital**  $\leadsto$  computergesteuerte Umsetzung der Steuerbefehle
- **Force Feedback** zur Rückmeldung wird notwendig!



- Elektrische/elektronische Ansteuerung  $\leadsto$  Fly-by-Wire
  - Analog  $\leadsto$  Steuerbefehle werden direkt auf elektrische Signale umgesetzt
  - Digital  $\leadsto$  computergesteuerte Umsetzung der Steuerbefehle
  - $\rightarrow$  Force Feedback zur Rückmeldung wird notwendig!

☞ Digitale Fly-by-Wire-Systeme haben diverse Vorteile

- Gewichtsersparnis im Vergleich zur Mechanik und zur Analogtechnik
- Computerunterstützung: Autopilot, Erhöhung der Flugzeugstabilität, Absicherung des sicheren Flugbereichs (engl. *flight envelope*)



## 1930er **Tupolev ANT-20 Maxim Gorky**

- Lange mechanische Steuerstrecken wurden elektrisch überbrückt



## 1930er **Tupolev ANT-20 Maxim Gorky**

- Lange mechanische Steuerstrecken wurden elektrisch überbrückt

## 1958 **Avro Canada CF-105 Arrow**

- Erstes militärische Serienflugzeug mit analogem Fly-by-Wire
- Auch Force Feedback (engl. *artificial feel*) wurde integriert



## 1930er **Tupolev ANT-20 Maxim Gorky**

- Lange mechanische Steuerstrecken wurden elektrisch überbrückt

## 1958 **Avro Canada CF-105 Arrow**

- Erstes militärische Serienflugzeug mit analogem Fly-by-Wire
- Auch Force Feedback (engl. *artificial feel*) wurde integriert

## 1969 **Concorde**

- Erstes ziviles Verkehrsflugzeug mit analogem Fly-by-Wire



## 1930er **Tupolev ANT-20 Maxim Gorky**

- Lange mechanische Steuerstrecken wurden elektrisch überbrückt

## 1958 **Avro Canada CF-105 Arrow**

- Erstes militärische Serienflugzeug mit analogem Fly-by-Wire
- Auch Force Feedback (engl. *artificial feel*) wurde integriert

## 1969 **Concorde**

- Erstes ziviles Verkehrsflugzeug mit analogem Fly-by-Wire

## 1972 **F-8 Crusader**

- Forschungsprojekt für digitales Fly-by-Wire
  - Eine digitale Flugsteuerung, drei analoge Backup-Systeme
- Etwa zeitgleich: Sukhoi T-4 (UdSSR), Hawker Hunter (GBR)



## 1930er **Tupolev ANT-20 Maxim Gorky**

- Lange mechanische Steuerstrecken wurden elektrisch überbrückt

## 1958 **Avro Canada CF-105 Arrow**

- Erstes militärische Serienflugzeug mit analogem Fly-by-Wire
- Auch Force Feedback (engl. *artificial feel*) wurde integriert

## 1969 **Concorde**

- Erstes ziviles Verkehrsflugzeug mit analogem Fly-by-Wire

## 1972 **F-8 Crusader**

- Forschungsprojekt für digitales Fly-by-Wire
  - Eine digitale Flugsteuerung, drei analoge Backup-Systeme
- Etwa zeitgleich: Sukhoi T-4 (UdSSR), Hawker Hunter (GBR)

## 1977 **Space Shuttle Orbiter** $\leadsto$ komplett digitale Flugsteuerung





## 1930er **Tupolev ANT-20 Maxim Gorky**

- Lange mechanische Steuerstrecken wurden elektrisch überbrückt

## 1958 **Avro Canada CF-105 Arrow**

- Erstes militärische Serienflugzeug mit analogem Fly-by-Wire
- Auch Force Feedback (engl. *artificial feel*) wurde integriert

## 1969 **Concorde**

- Erstes ziviles Verkehrsflugzeug mit analogem Fly-by-Wire

## 1972 **F-8 Crusader**

- Forschungsprojekt für digitales Fly-by-Wire
  - Eine digitale Flugsteuerung, drei analoge Backup-Systeme
- Etwa zeitgleich: Sukhoi T-4 (UdSSR), Hawker Hunter (GBR)

## 1977 **Space Shuttle Orbiter** $\leadsto$ komplett digitale Flugsteuerung

## 1984 **Airbus A320**

- Erstes Verkehrsflugzeug mit vollständig digitaler Flugsteuerung
  - Es existieren aber weiterhin elektrische Backup-Systeme





## redundanter Aufbau der Flugsteuerung

- 7 (A320/321) bzw. 5 (A330/340) Computer zur Flugsteuerung
  - Hardware Replikation und Implementierung von fail-silent-Verhalten
- 3 Hydraulikkreisläufe zur Versorgung der Aktoren
- Stromversorgungen durch unabhängige Generatoren an den Triebwerken
  - $\geq 2$  Stromkreise + Notstromversorgung (Batterien, Windrad)





## redundanter Aufbau der Flugsteuerung

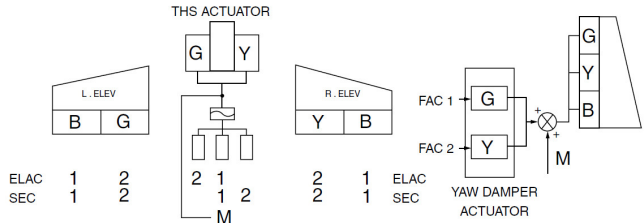
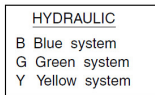
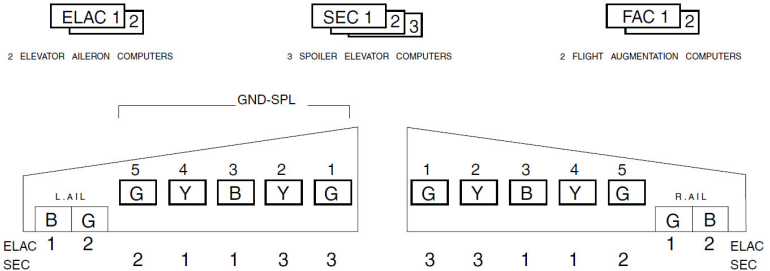
- 7 (A320/321) bzw. 5 (A330/340) Computer zur Flugsteuerung
  - Hardware Replikation und Implementierung von fail-silent-Verhalten
- 3 Hydraulikkreisläufe zur Versorgung der Aktoren
- Stromversorgungen durch unabhängige Generatoren an den Triebwerken
  - $\geq 2$  Stromkreise + Notstromversorgung (Batterien, Windrad)

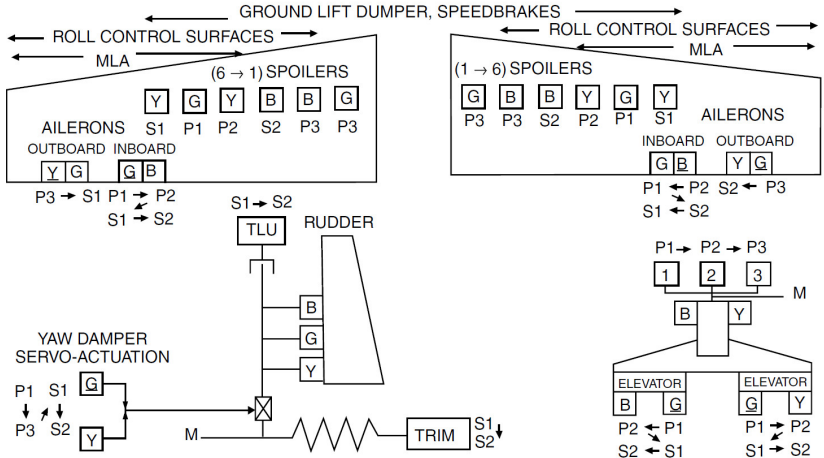


## diversitärer Aufbau der Flugsteuerung

- Verschiedene Steuercomputer im A320/A321 [3]
  - 2 Elevator Aileron Computer (ELAC)  $\leadsto$  Motorola 68010 (Thomson-CSF)
  - 3 Spoiler Elevator Computer (SEC)  $\leadsto$  Intel 80186 (SFENA/Aerospatiale)
  - 2 Flight Augmentation Computer (FAC)
- Verschiedene Steuercomputer ab A330/A340 [12]
  - 3 Flight Control Primary Computers (PRIM)  $\leadsto$  Power PC
  - 2 Flight Control Secondary Computers (SEC)  $\leadsto$  Sharc
- Jeder Computer besteht aus einem Kontroll- und einem Monitor-Kanal
  - Verwendung unterschiedlicher Softwarepakete (z. B. Codegeneratoren)
  - 4 Softwarepakete: 2 x ELAC, 2 x SEC im A320/321

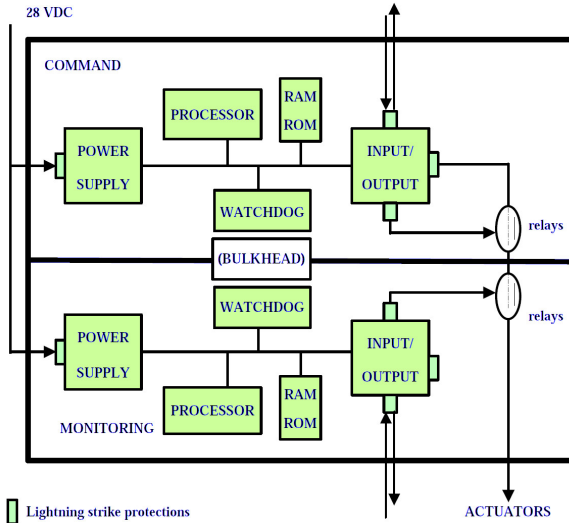






# Aufbau eines A330/A340-Steuercomputers

Quelle Grafik: [12]



- Selbstüberwachung einzelner Steuercomputer
  - Stetige gegenseitige Überwachung von Kontroll- und Monitor-Kanal
    - Vergleich von Ausgaben und ausgewählten Zustandsvariablen
    - Übersteigt die Differenz einen gewissen Schwellwert  $\leadsto$  Fehler
  - Selbsttests werden zumindest bei Systemstart durchgeführt
    - Flugzeuge werden anders als Kernkraftwerke öfters neu gestartet



- Selbstüberwachung einzelner Steuercomputer
  - Stetige gegenseitige Überwachung von Kontroll- und Monitor-Kanal
    - Vergleich von Ausgaben und ausgewählten Zustandsvariablen
    - Übersteigt die Differenz einen gewissen Schwellwert  $\leadsto$  Fehler
  - Selbsttests werden zumindest bei Systemstart durchgeführt
    - Flugzeuge werden anders als Kernkraftwerke öfters neu gestartet
- Rekonfiguration  $\leadsto$  Graceful Degradation
  - Regelkreise zur Fluglageregelung sind auf Sensoren angewiesen
    - Andernfalls kann der Zustand des Flugzeugs nicht mehr erfasst werden
  - Im Fokus: Air Data and Inertial Reference Units (ADIRUs)
    - Nicken  $\leadsto$  Unterstützung durch Beschleunigungssensoren und Gyroskope
    - $\rightarrow$  Funktionelle Redundanz
  - $\rightarrow$  Graceful Degradation  $\leadsto$  Rückzug auf eine direkte Flugsteuerung
    - z. B. wenn alle ADIRUs ausgefallen sind
    - $\rightarrow$  Keine computergestützte Fluglageregelung, kein gesicherter Flugbereich





- 1 Überblick
- 2 Verlässlichkeit durch Verteilung
  - Beispiel: Flugsteuerung
  - Verteiltes Fly-By-Wire
- 3 Kommunikation
  - Problemstellung und Grundlagen
  - Zeitgesteuerte Kommunikationssysteme
- 4 Zustandswiederherstellung
  - Vorgehen bei der Reintegration
  - Fehlererholung
  - Interner Zustand
- 5 Zusammenfassung





Zuschnitt und Platzierung von Funktionen und die Auslegung ihrer Schnittstelle bilden wesentliche Aktivitäten des Entwurfs von **verlässlichen Echtzeitsystemen**

- Bestimmt die Eigenschaften des Systems als Ganzes
- Zentralisierung erschwert Fehlerdiagnose und Analyse von Fehlereffekten





Zuschnitt und Platzierung von Funktionen und die Auslegung ihrer Schnittstelle bilden wesentliche Aktivitäten des Entwurfs von **verlässlichen Echtzeitsystemen**

- Bestimmt die Eigenschaften des Systems als Ganzes
- Zentralisierung erschwert Fehlerdiagnose und Analyse von Fehlereffekten



Herausforderung dabei ist, dass die Abstraktionen an den Schnittstellen auch im **Fehlerfall** ihre Gültigkeit haben sollten





Zuschnitt und Platzierung von Funktionen und die Auslegung ihrer Schnittstelle bilden wesentliche Aktivitäten des Entwurfs von **verlässlichen Echtzeitrechensystemen**

- Bestimmt die Eigenschaften des Systems als Ganzes
- Zentralisierung erschwert Fehlerdiagnose und Analyse von Fehlereffekten



Herausforderung dabei ist, dass die Abstraktionen an den Schnittstellen auch im **Fehlerfall** ihre Gültigkeit haben sollten

- Ideal ist ein **verteiltes Rechensystem** mit einer 1-zu-1-Beziehung zwischen Funktion und Rechenknoten
  - Ursache bzw. Knoten einer Fehlfunktion ist **leicht identifizierbar**
  - Auswirkungen eines fehlerhaften Knotens sind **gut vorhersehbar**



# Reaktionsfähiges System (engl. *responsive system*)

⇒ Verteilung + Echtzeitperformanz + Fehlertoleranz [10]



Voraussetzung: Fehlererkennung auch im verteilten Fall!



# Reaktionsfähiges System (engl. *responsive system*)

⇒ Verteilung + Echtzeitperformanz + Fehlertoleranz [10]



Voraussetzung: Fehlererkennung auch im verteilten Fall!



**Replikation** (der Funktionen) von Knoten (vgl. Folie VIII/9 ff.)

- Erfordert **Replikdeterminismus** (engl. *replica determinism* [11])
  - Im Wertebereich: Gleiche Eingaben führen zu gleichen Ergebnissen!
  - Im Zeitbereich:
    - Aktiv replizierte Knoten sehen denselben Zustand *zur selben Zeit*
    - Mit Zugeständnis zur endl. Genauigkeit der *Uhrensynchronisation*
- Ohne diese Voraussetzungen ist eine Mehrheitsentscheid sinnlos!



# Reaktionsfähiges System (engl. *responsive system*)

⇒ Verteilung + Echtzeitperformanz + Fehlertoleranz [10]



Voraussetzung: Fehlererkennung auch im verteilten Fall!



**Replikation** (der Funktionen) von Knoten (vgl. Folie VIII/9 ff.)

- Erfordert **Replikdeterminismus** (engl. *replica determinism* [11])
- Im Wertebereich: Gleiche Eingaben führen zu gleichen Ergebnissen!
- Im Zeitbereich:
  - Aktiv replizierte Knoten sehen denselben Zustand *zur selben Zeit*
  - Mit Zugeständnis zur endl. Genauigkeit der *Uhrensynchronisation*

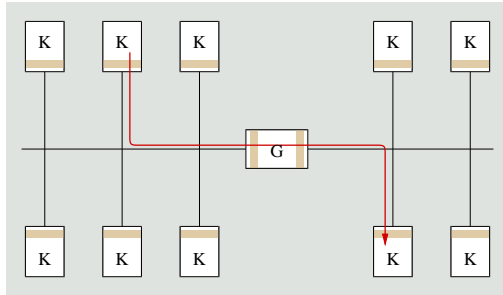
→ Ohne diese Voraussetzungen ist ein Mehrheitsentscheid sinnlos!



**Fehlereingrenzung** durch eine **Sicherheitshülle** (engl. *containment*)

- Fehlertolerante Systeme sind in Partitionen strukturiert
  - In **fehlereingrenzende Regionen** (engl. *error-containment regions*)
- Fehler, die in einer Partition auftreten, bleiben isoliert
  - Sie werden lokal erkannt und korrigiert oder maskiert
  - Sie können das restliche System nicht beschädigen





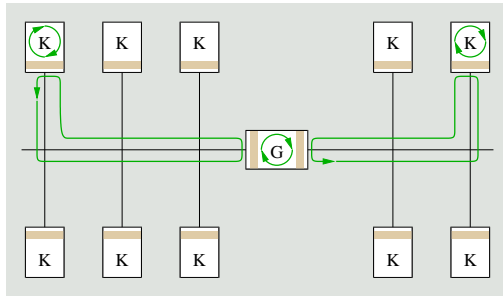
## Ereignisgesteuertes Kommunikationssystem

- Kommunikation basiert auf Ereignisnachrichten
  - Auslösezeitpunkt und Übertragung **abhängig von Lastsituation**
  - Einfluss durch Ausfall/Integration von Knoten
- **Nicht für die Kapselung des Zeitverhaltens geeignet!**





# Zeitgesteuerte vs. Ereignisgesteuerte Kommunikation



## Zeitgesteuertes Kommunikationssystem

- Kommunikation basiert auf Zustandsnachrichten
- Kein unkoordinierter Nachrichtenversand, **Auslastung vorab bestimmbar**  
→ **Kapselung des zeitlichen Verhaltens (temporale Brandmauer)!**
- ⚠ Uhrensynchronisation erforderlich



- Aeronautical Radio Incorporated 629 Standard [1]
  - Entwickelt für die Boeing 777
  - Erstmaliger Einsatz eines standardisierten Bussystems
  - Teilnehmer können ihre Daten in festgelegten Zeitintervallen versenden
  - Kein (böswilliger) Knoten kann den Bus monopolisieren (**Busschutz**)



- Aeronautical Radio Incorporated 629 Standard [1]
  - Entwickelt für die Boeing 777
  - Erstmaliger Einsatz eines standardisierten Bussystems
  - Teilnehmer können ihre Daten in festgelegten Zeitintervallen versenden
  - Kein (böswilliger) Knoten kann den Bus monopolisieren (**Busschutz**)
- Aufteilung der Zeit in **Minischlitze** (engl. *mini slots*)
  - Kommunikation wird in 3 Zeitintervalle unterteilt
    - Sendeintervall, Synchronisation, Geräteerkennung
    - Länger als die Ausbreitungsverzögerung des Kanals
    - Jedem Knoten ist eine eindeutige Anzahl von *Minislots* zugeordnet
  - Ein **Warteraumprotokoll** ähnlich zum Bäckereialgorithmus [9]
    - Zunächst finden sich alle Sender in einem (verteilten) Warteraum ein
    - Im nachfolgenden Zeitintervall (Epoche) findet die Kommunikation statt



- Aeronautical Radio Incorporated 629 Standard [1]
  - Entwickelt für die Boeing 777
  - Erstmaliger Einsatz eines standardisierten Bussystems
  - Teilnehmer können ihre Daten in festgelegten Zeitintervallen versenden
  - Kein (böswilliger) Knoten kann den Bus monopolisieren (**Busschutz**)
- Aufteilung der Zeit in **Minischlitze** (engl. *mini slots*)
  - Kommunikation wird in 3 Zeitintervalle unterteilt
    - Sendeintervall, Synchronisation, Geräteerkennung
    - Länger als die Ausbreitungsverzögerung des Kanals
    - Jedem Knoten ist eine eindeutige Anzahl von *Minislots* zugeordnet
  - Ein **Warteraumprotokoll** ähnlich zum Bäckereialgorithmus [9]
    - Zunächst finden sich alle Sender in einem (verteilten) Warteraum ein
    - Im nachfolgenden Zeitintervall (Epoche) findet die Kommunikation statt



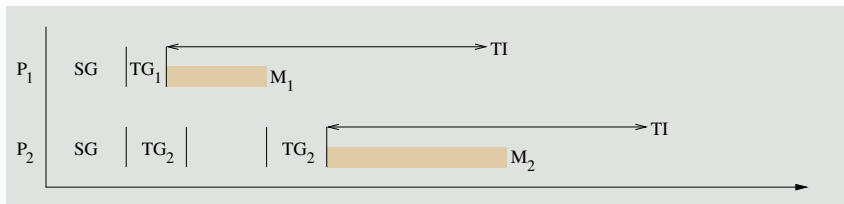
Achtung: Medienzugriff basiert auf CSMA/CD  $\leadsto$  Mischform!



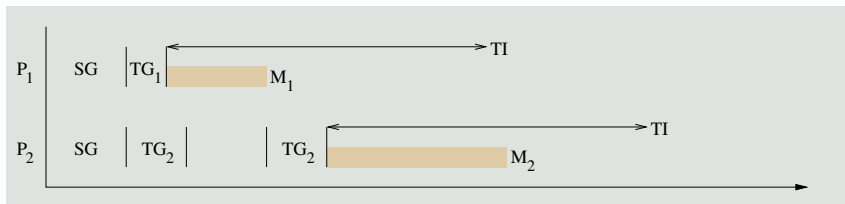
- **TI** (engl. *transmit interval*)  $\leadsto$  Sendeintervall
  - Verhindert Monopolisierung des Busses: identisch für alle Knoten
- **SG** (engl. *synchronization gap*)  $\leadsto$  Synchronisationsintervall
  - Kontrolliert Zutritt zum Warteraum: identisch für alle Knoten
- **TG** (engl. *termination gap*)  $\leadsto$  Gerätepriorisierung
  - Kontrolliert den Buszugriff: unterschiedlich für jeden Knoten
- Relationen zwischen den Zeitparametern (*Timeouts*):
  - $SG > \max(TG_i)$ , für alle Knoten bzw. Prozesse  $i$
  - $TI > SG$



Prozesse  $P_1$  und  $P_2$  wollen gleichzeitig senden,  $TG_1 < TG_2$ :

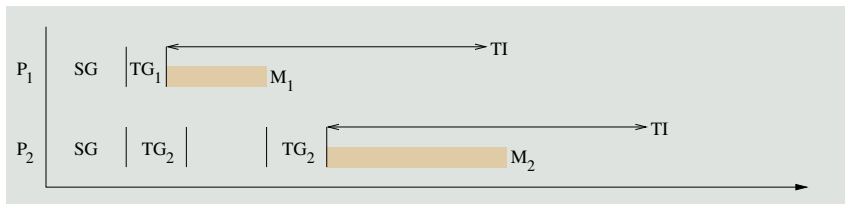


Prozesse  $P_1$  und  $P_2$  wollen gleichzeitig senden,  $TG_1 < TG_2$ :



- 1  $P_1$  und  $P_2$  warten SG Zeiten Busruhe ab, betreten den Warteraum

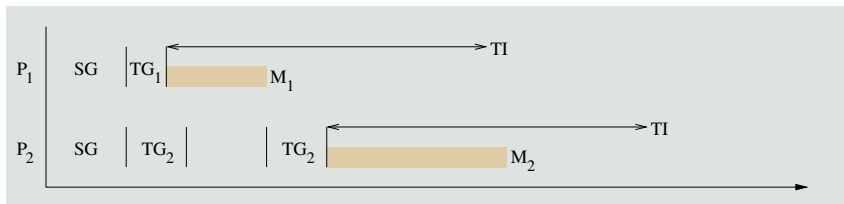
Prozesse  $P_1$  und  $P_2$  wollen gleichzeitig senden,  $TG_1 < TG_2$ :



- 1  $P_1$  und  $P_2$  warten  $SG$  Zeiten Busruhe ab, betreten den Warteraum
- 2 jeder Prozess  $i$  wartet zusätzlich noch seine  $TG_i$  Zeiten Busruhe ab

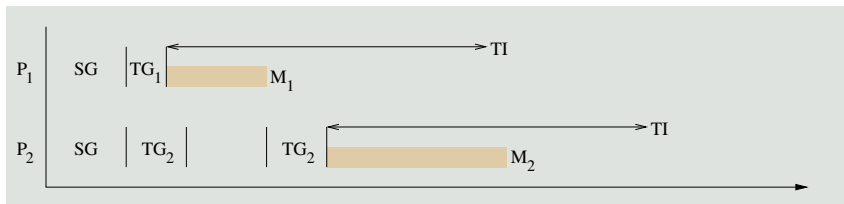


Prozesse  $P_1$  und  $P_2$  wollen gleichzeitig senden,  $TG_1 < TG_2$ :



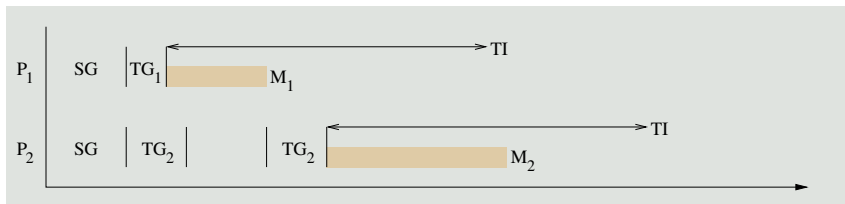
- 1  $P_1$  und  $P_2$  warten SG Zeiten Busruhe ab, betreten den Warteraum
- 2 jeder Prozess  $i$  wartet zusätzlich noch seine  $TG_i$  Zeiten Busruhe ab
- 3 wegen  $TG_1 < TG_2$  sendet  $P_1$  zuerst seine Nachricht  $M_1$

Prozesse  $P_1$  und  $P_2$  wollen gleichzeitig senden,  $TG_1 < TG_2$ :



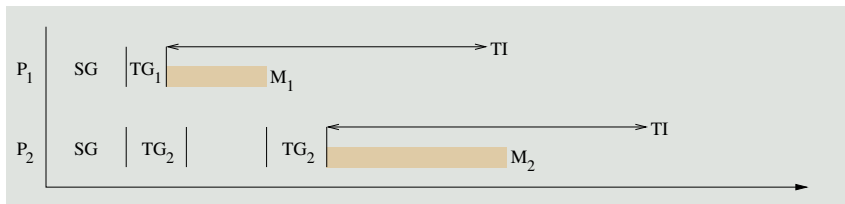
- 1  $P_1$  und  $P_2$  warten  $SG$  Zeiten Busruhe ab, betreten den Warteraum
- 2 jeder Prozess  $i$  wartet zusätzlich noch seine  $TG_i$  Zeiten Busruhe ab
- 3 wegen  $TG_1 < TG_2$  sendet  $P_1$  zuerst seine Nachricht  $M_1$
- 4  $P_2$  erkennt Verkehr auf den Bus; wartet, bis  $M_1$  übertragen wurde

Prozesse  $P_1$  und  $P_2$  wollen gleichzeitig senden,  $TG_1 < TG_2$ :



- 1  $P_1$  und  $P_2$  warten SG Zeiten Busruhe ab, betreten den Warteraum
- 2 jeder Prozess  $i$  wartet zusätzlich noch seine  $TG_i$  Zeiten Busruhe ab
- 3 wegen  $TG_1 < TG_2$  sendet  $P_1$  zuerst seine Nachricht  $M_1$
- 4  $P_2$  erkennt Verkehr auf den Bus; wartet, bis  $M_1$  übertragen wurde
- 5  $P_2$  wartet  $TG_2$  Zeiten Busruhe ab und sendet seine Nachricht  $M_2$

Prozesse  $P_1$  und  $P_2$  wollen gleichzeitig senden,  $TG_1 < TG_2$ :



- 1  $P_1$  und  $P_2$  warten  $SG$  Zeiten Busruhe ab, betreten den Warteraum
- 2 jeder Prozess  $i$  wartet zusätzlich noch seine  $TG_i$  Zeiten Busruhe ab
- 3 wegen  $TG_1 < TG_2$  sendet  $P_1$  zuerst seine Nachricht  $M_1$
- 4  $P_2$  erkennt Verkehr auf den Bus; wartet, bis  $M_1$  übertragen wurde
- 5  $P_2$  wartet  $TG_2$  Zeiten Busruhe ab und sendet seine Nachricht  $M_2$
- 6  $P_1$  und  $P_2$  können frühestens nach  $TI$  Zeiten erneut senden



# TDMA (engl. *time division multiple access*)

a.k.a. Avionics Full-Duplex Ethernet Switching (AFDX)

- Übertragungsrechte durch Voranschreiten der Echtzeit:
  - Voraussetzung: fehlertolerante globale Zeitbasis in allen Knoten



# TDMA (engl. *time division multiple access*)

a.k.a. Avionics Full-Duplex Ethernet Switching (AFDX)

- Übertragungsrechte durch Voranschreiten der Echtzeit:
  - Voraussetzung: **fehlertolerante globale Zeitbasis** in allen Knoten
  - statische Aufteilung der gesamten Kanalkapazität in **Zeitschlitz**
    - Jeder Knoten (Busteilnehmer) hat einen eindeutigen Sendeschlitz
    - **TDMA Runde**  $\leadsto$  Sequenz von Sendeschlitzen einer Knotengruppe
    - In jeder Runde kann ein Knoten eine Nachricht übertragen
    - Ist nichts zu versenden, bleibt ein **Rahmen** (engl. *frame*) leer



- Übertragungsrechte durch Voranschreiten der Echtzeit:
  - Voraussetzung: **fehlertolerante globale Zeitbasis** in allen Knoten
  - statische Aufteilung der gesamten Kanalkapazität in **Zeitschlitz**
    - Jeder Knoten (Busteilnehmer) hat einen eindeutigen Sendeschlitz
    - **TDMA Runde**  $\leadsto$  Sequenz von Sendeschlitzen einer Knotengruppe
    - In jeder Runde kann ein Knoten eine Nachricht übertragen
    - Ist nichts zu versenden, bleibt ein **Rahmen** (engl. *frame*) leer
  - Runden wiederholen sich  $\leadsto$  **Gruppentakt** (engl. *cluster cycle*)
    - Sequenz verschiedener TDMA-Runden
    - Die Gruppentaktlänge bestimmt die Periodizität des TDMA-Systems



# TDMA (engl. *time division multiple access*)

a.k.a. Avionics Full-Duplex Ethernet Switching (AFDX)

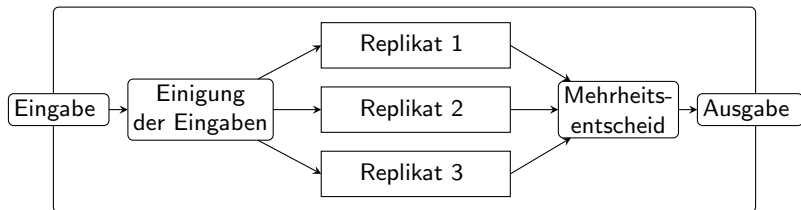
- Übertragungsrechte durch Voranschreiten der Echtzeit:
  - Voraussetzung: **fehlertolerante globale Zeitbasis** in allen Knoten
  - statische Aufteilung der gesamten Kanalkapazität in **Zeitschlitz**
    - Jeder Knoten (Busteilnehmer) hat einen eindeutigen Sendeschlitz
    - **TDMA Runde**  $\leadsto$  Sequenz von Sendeschlitzen einer Knotengruppe
    - In jeder Runde kann ein Knoten eine Nachricht übertragen
    - Ist nichts zu versenden, bleibt ein **Rahmen** (engl. *frame*) leer
  - Runden wiederholen sich  $\leadsto$  **Gruppentakt** (engl. *cluster cycle*)
    - Sequenz verschiedener TDMA-Runden
    - Die Gruppentaktlänge bestimmt die Periodizität des TDMA-Systems
- Grundlagenforschung: TTP (engl. *time-triggered protocol* [8])
  - Kommerzielle Varianten:
    - byteflight** ([2], Sicherheits- und Informationsbussystem: **SI-BUS**)
    - FlexRay** ([6], zeit- und ereignisgesteuerter Bus)
    - AFDX** ([5], Airbus, Boeing, NASA)



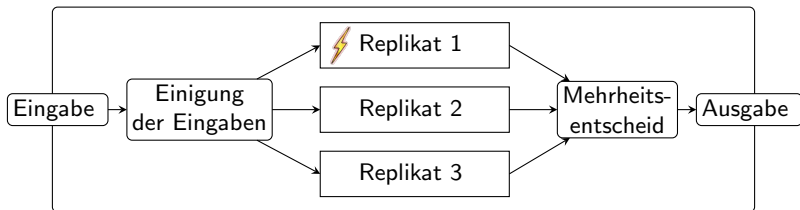


- 1 Überblick
- 2 Verlässlichkeit durch Verteilung
  - Beispiel: Flugsteuerung
  - Verteiltes Fly-By-Wire
- 3 Kommunikation
  - Problemstellung und Grundlagen
  - Zeitgesteuerte Kommunikationssysteme
- 4 Zustandswiederherstellung
  - Vorgehen bei der Reintegration
  - Fehlererholung
  - Interner Zustand
- 5 Zusammenfassung





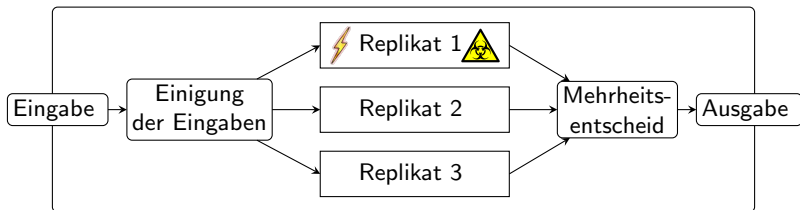
- **Normalbetrieb:** Alle Replikate arbeiten
  - Redundanz und Fehlertoleranz sind in vollem Umfang gegeben



- **Normalbetrieb:** Alle Replikate arbeiten
  - Redundanz und Fehlertoleranz sind in vollem Umfang gegeben

⚠ **Fehlerfall:** Ein Replikat erleidet einen **transienten Fehler**

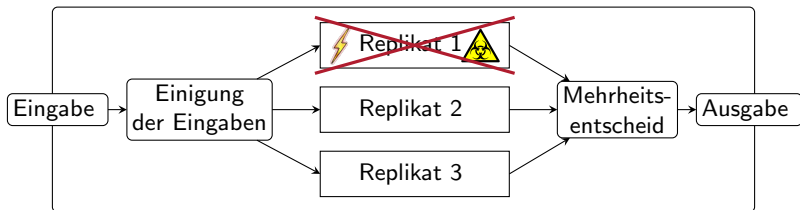
→ Der interne Zustand wird hierdurch korrumpiert



- **Normalbetrieb:** Alle Replikate arbeiten
  - Redundanz und Fehlertoleranz sind in vollem Umfang gegeben

⚠ **Fehlerfall:** Ein Replikat erleidet einen **transienten Fehler**

- Der interne Zustand wird hierdurch korrumpiert
- Das Replikat **fällt aus** ...



- **Normalbetrieb:** Alle Replikate arbeiten
  - Redundanz und Fehlertoleranz sind in vollem Umfang gegeben

⚠ **Fehlerfall:** Ein Replikat erleidet einen **transienten Fehler**

- Der interne Zustand wird hierdurch korruptiert
- Das Replikat **fällt aus** ...

👉 Dies bedeutet eine **Reduktion der Fähigkeit Fehler zu tolerieren**

- Nur noch zwei funktionsfähige Replikate  $\leadsto$  das System ist **verwundbar**
- Replikat 1 muss sich **erholen** und **reintegriert** werden



# Reintegration fehlgeschlagener Knoten

Wie geht man grundsätzlich mit dem Ausfall eines Knotens um?

## 1 Fehlererkennung

⇒ Ist ein Fehler aufgetreten?

- Zu Beginn steht die Erkennung, dass ein Replikat fehlerhaft ist
  - Dies kann durch den Mehrheitsentscheid erfolgen, oder
  - Das fehlerhafte Replikat teilt dies selbst mit (s. Folie VIII/11, *crash failure*)



# Reintegration fehlgeschlagener Knoten

Wie geht man grundsätzlich mit dem Ausfall eines Knotens um?

## 1 Fehlererkennung

→ Ist ein Fehler aufgetreten?

- Zu Beginn steht die Erkennung, dass ein Replikat fehlerhaft ist
  - Dies kann durch den Mehrheitsentscheid erfolgen, oder
  - Das fehlerhafte Replikat teilt dies selbst mit (s. Folie VIII/11, [crash failure](#))

## 2 Fehlerdiagnose

→ Wo ist der Fehler aufgetreten?

- Welches Replikat ist ausgefallen?



# Reintegration fehlgeschlagener Knoten

Wie geht man grundsätzlich mit dem Ausfall eines Knotens um?

- 1 Fehlererkennung  $\mapsto$  Ist ein Fehler aufgetreten?
  - Zu Beginn steht die Erkennung, dass ein Replikat fehlerhaft ist
    - Dies kann durch den Mehrheitsentscheid erfolgen, oder
    - Das fehlerhafte Replikat teilt dies selbst mit (s. Folie VIII/11, *crash failure*)
- 2 Fehlerdiagnose  $\mapsto$  Wo ist der Fehler aufgetreten?
  - Welches Replikat ist ausgefallen?
- 3 Rekonfiguration  $\mapsto$  Das fehlerhafte Replikat aussperren.
  - Das fehlerhafte Replikat darf nicht mehr am Betrieb teilnehmen
  - Ggf. wird das Replikat durch ein Backup-System ersetzt
    - So wird die volle Fähigkeit Fehler zu tolerieren schnell wiederhergestellt





# Reintegration fehlgeschlagener Knoten

Wie geht man grundsätzlich mit dem Ausfall eines Knotens um?

- 1 Fehlererkennung  $\mapsto$  Ist ein Fehler aufgetreten?
  - Zu Beginn steht die Erkennung, dass ein Replikat fehlerhaft ist
    - Dies kann durch den Mehrheitsentscheid erfolgen, oder
    - Das fehlerhafte Replikat teilt dies selbst mit (s. Folie VIII/11, **crash failure**)
- 2 Fehlerdiagnose  $\mapsto$  Wo ist der Fehler aufgetreten?
  - Welches Replikat ist ausgefallen?
- 3 Rekonfiguration  $\mapsto$  Das fehlerhafte Replikat aussperren.
  - Das fehlerhafte Replikat darf nicht mehr am Betrieb teilnehmen
  - Ggf. wird das Replikat durch ein Backup-System ersetzt
    - So wird die volle Fähigkeit Fehler zu tolerieren schnell wiederhergestellt
- 4 Fehlererholung  $\mapsto$  Den Fehler beseitigen.
  - Kann der Fehler behoben werden, oder liegt ein permanenter Fehler vor?
    - Umsetzung durch einen Selbsttest, den das Replikat durchführt.

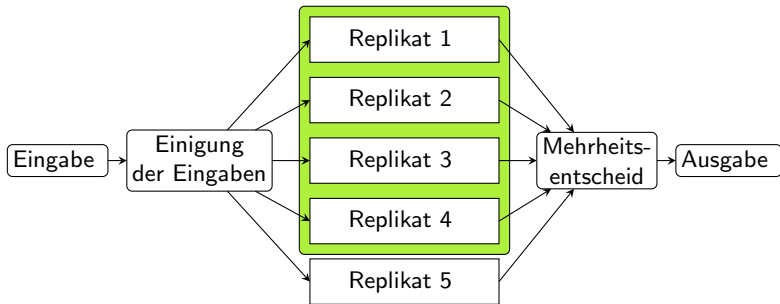


# Reintegration fehlgeschlagener Knoten

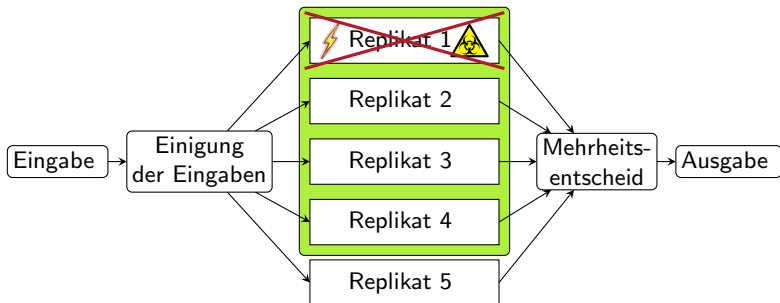
Wie geht man grundsätzlich mit dem Ausfall eines Knotens um?

- 1 Fehlererkennung  $\mapsto$  Ist ein Fehler aufgetreten?
  - Zu Beginn steht die Erkennung, dass ein Replikat fehlerhaft ist
    - Dies kann durch den Mehrheitsentscheid erfolgen, oder
    - Das fehlerhafte Replikat teilt dies selbst mit (s. Folie VIII/11, [crash failure](#))
- 2 Fehlerdiagnose  $\mapsto$  Wo ist der Fehler aufgetreten?
  - Welches Replikat ist ausgefallen?
- 3 Rekonfiguration  $\mapsto$  Das fehlerhafte Replikat aussperren.
  - Das fehlerhafte Replikat darf nicht mehr am Betrieb teilnehmen
  - Ggf. wird das Replikat durch ein Backup-System ersetzt
    - So wird die volle Fähigkeit Fehler zu tolerieren schnell wiederhergestellt
- 4 Fehlererholung  $\mapsto$  Den Fehler beseitigen.
  - Kann der Fehler behoben werden, oder liegt ein permanenter Fehler vor?
    - Umsetzung durch einen Selbsttest, den das Replikat durchführt.
- 5 Reintegration  $\mapsto$  Den Ursprungszustand wiederherstellen.
  - Das Replikat wird erneut in den Verbund aufgenommen

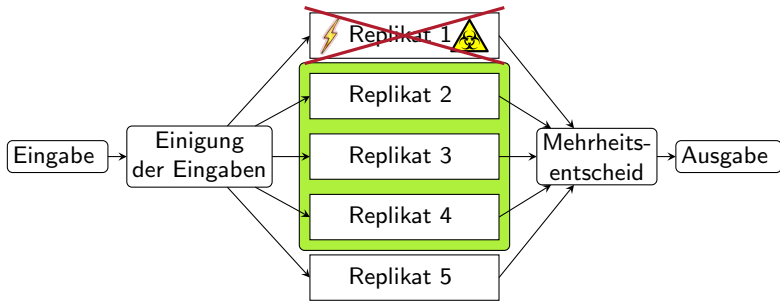




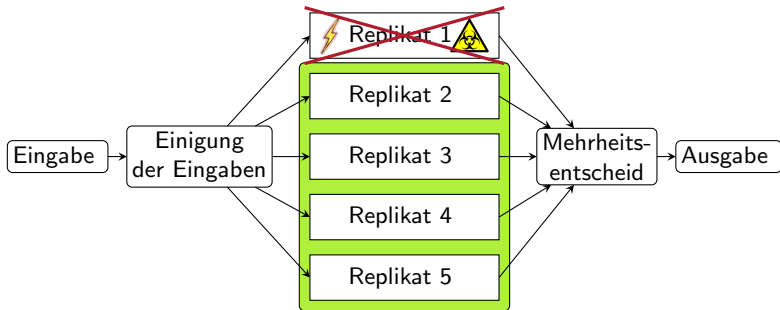
- Replikate  $\{1,2,3,4\}$  arbeiten in einem 4-fach redundanten Verbund
- Replik 5 ist das Backup-System



- Replikate {1,2,3,4} arbeiten in einem 4-fach redundanten Verbund
  - Replikat 5 ist das Backup-System
- ☞ Replikat 1 fällt nun aufgrund eines transienten Fehlers aus

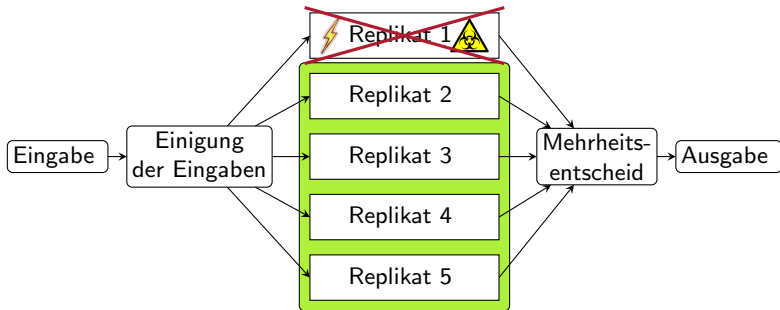


- Replikate  $\{1,2,3,4\}$  arbeiten in einem 4-fach redundanten Verbund
  - Replik 5 ist das Backup-System
- ☞ Replik 1 fällt nun aufgrund eines transienten Fehlers aus
  - Zunächst wird Replik 1 aus dem Verbund ausgeschlossen



- Replikate  $\{1,2,3,4\}$  arbeiten in einem 4-fach redundanten Verbund
  - Replikat 5 ist das Backup-System
- ☞ Replikat 1 fällt nun aufgrund eines transienten Fehlers aus
  - Zunächst wird Replikat 1 aus dem Verbund ausgeschlossen
  - Dann wird Replikat 5 in den Verbund aufgenommen
    - Um die Fähigkeit zur Fehlertoleranz wieder vollkommen herzustellen





- Replikate  $\{1,2,3,4\}$  arbeiten in einem 4-fach redundanten Verbund
  - Replikat 5 ist das Backup-System
- ☞ Replikat 1 fällt nun aufgrund eines transienten Fehlers aus
  - Zunächst wird Replikat 1 aus dem Verbund ausgeschlossen
  - Dann wird Replikat 5 in den Verbund aufgenommen
    - Um die Fähigkeit zur Fehlertoleranz wieder vollkommen herzustellen
  - Schließlich bleibt noch Replikat 1 zu erholen und neu zu integrieren

**Reaktive Fehlererholung**  $\leadsto$  nachdem der Fehler aufgetreten ist

- Setzt die Erkennung des Fehlers voraus
- Anschließend wird die Fehlersituation repariert
  - Ein **Wiederholungsversuch** (engl. *retry*) wird unternommen
  - Es wird zu einem **Sicherungspunkt** (engl. *checkpoint*) zurückgekehrt





**Reaktive Fehlererholung**  $\leadsto$  nachdem der Fehler aufgetreten ist

- Setzt die Erkennung des Fehlers voraus
- Anschließend wird die Fehlersituation repariert
  - Ein **Wiederholungsversuch** (engl. *retry*) wird unternommen
  - Es wird zu einem **Sicherungspunkt** (engl. *checkpoint*) zurückgekehrt

**Proaktive Fehlererholung**  $\leadsto$  Vermeidung von Fehlersituationen

- Vorsorglich durchgeführte Maßnahmen, die
  - Regelmäßig zu festen Zeitpunkten, oder
  - Als Reaktion auf die Veränderung bestimmter Indikatoren stattfinden
    - Wenn ein Leistungsabfall (z. B. mittleren Antwortzeit) beobachtbar ist
- Sie umfassen z. B. regelmäßige Neustarts, leeren von Puffern, ...



**Reaktive Fehlererholung**  $\leadsto$  nachdem der Fehler aufgetreten ist

- Setzt die Erkennung des Fehlers voraus
- Anschließend wird die Fehlersituation repariert
  - Ein **Wiederholungsversuch** (engl. *retry*) wird unternommen
  - Es wird zu einem **Sicherungspunkt** (engl. *checkpoint*) zurückgekehrt

**Proaktive/reaktive Fehlererholung**  $\leadsto$  Vorbereitungen treffen

- Proaktive Maßnahmen bereiten eine reaktive Fehlererholung vor
  - Das regelmäßige Erstellen von Sicherungspunkten ist proaktiv
  - Deren Wiederherstellung im Fehlerfall hingegen reaktiv

**Proaktive Fehlererholung**  $\leadsto$  Vermeidung von Fehlersituationen

- Vorsorglich durchgeführte Maßnahmen, die
  - Regelmäßig zu festen Zeitpunkten, oder
  - Als Reaktion auf die Veränderung bestimmter Indikatoren stattfinden
    - Wenn ein Leistungsabfall (z. B. mittleren Antwortzeit) beobachtbar ist
- Sie umfassen z. B. regelmäßige Neustarts, leeren von Puffern, ...



## Rückwärtskorrektur (engl. *roll backward, backward error recovery*)

- Rückkehr zu einem **Sicherungspunkt** im Fehlerfall
    - Der Sicherungspunkt muss entsprechend vertrauenswürdig sein
  - Bearbeitung muss komplett **umkehrbar** sein
    - Keine **nicht-zurücknehmbaren Aktionen** (engl. *irrevocable actions*)
    - Ansonsten wäre eine Rückkehr zum Sicherungspunkt unmöglich
  - **Dauer der Fehlererholung** hängt von folgenden Faktoren ab
    - **Größe bzw. Umfang** des Sicherungspunkts
      - Wie viele Daten müssen kopiert werden? Wie lange dauert dies?
    - **Alter** des Sicherungspunkts
      - Evtl. müssen verpasste Anfragen nachgeholt werden (engl. *audit trail*)
- Für Echtzeitsysteme häufig **zu langwierig**



## Rückwärtskorrektur (engl. *roll backward, backward error recovery*)

- Rückkehr zu einem **Sicherungspunkt** im Fehlerfall
    - Der Sicherungspunkt muss entsprechend vertrauenswürdig sein
  - Bearbeitung muss komplett **umkehrbar** sein
    - Keine **nicht-zurücknehmbaren Aktionen** (engl. *irrevocable actions*)
    - Ansonsten wäre eine Rückkehr zum Sicherungspunkt unmöglich
  - **Dauer der Fehlererholung** hängt von folgenden Faktoren ab
    - **Größe bzw. Umfang** des Sicherungspunkts
      - Wie viele Daten müssen kopiert werden? Wie lange dauert dies?
    - **Alter** des Sicherungspunkts
      - Evtl. müssen verpasste Anfragen nachgeholt werden (engl. *audit trail*)
- Für Echtzeitsysteme häufig **zu langwierig**

## Vorwärtskorrektur (engl. *roll forward, forward error recovery*)

- Kommt **ohne Rückgriff auf vorher gesicherten Zustand** aus
  - Die **bevorzugte Variante für Echtzeitsysteme**
    - Zustand wird in jedem Zyklus neu erfasst  $\leadsto$  Maskierung alter Werte
    - **Kurze Fehlererholung**  $\leadsto$  im Idealfall durch den Normalbetrieb





- Zwei Teile sind maßgeblich für den Zustand eines Replikats



- Zwei Teile sind maßgeblich für den Zustand eines Replikats  
**Initialzustand** (engl. *initialization state*, *i-state*)
  - Statische, zur Laufzeit unveränderliche Datenstruktur
    - Umfasst z. B. den Programmcode und Initialisierungsdaten
    - Auch zur Laufzeit konstante Daten zählen hierzu
  - Ablage im **nicht-schreibbaren Speicher** (engl. *read-only memory*, *ROM*)



- Zwei Teile sind maßgeblich für den Zustand eines Replikats  
**Initialzustand** (engl. *initialization state*, *i-state*)

- Statische, zur Laufzeit unveränderliche Datenstruktur
  - Umfasst z. B. den Programmcode und Initialisierungsdaten
  - Auch zur Laufzeit konstante Daten zählen hierzu
- Ablage im **nicht-schreibbaren Speicher** (engl. *read-only memory*, *ROM*)

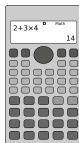
**Dynamischer Zustand** (engl. *history state*, *h-state*)

- Zur Laufzeit veränderliche Daten
  - Beinhalten Informationen zum Fortschritt einer Berechnung
  - Ist bei der Reintegration ausgefallener Replikate zu restaurieren
- Ablage im **schreibbaren Speicher** (engl. *random-access memory*, *RAM*)



# Beispiel: Dynamischer Zustand eines Taschenrechners

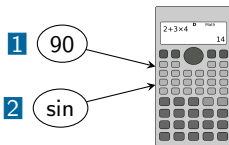
- Verwendung eines Taschenrechners





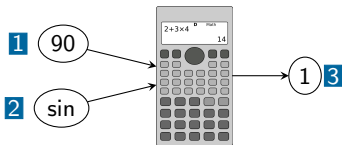
# Beispiel: Dynamischer Zustand eines Taschenrechners

- Verwendung eines Taschenrechners
  - Eingabe der Operanden und Operatoren



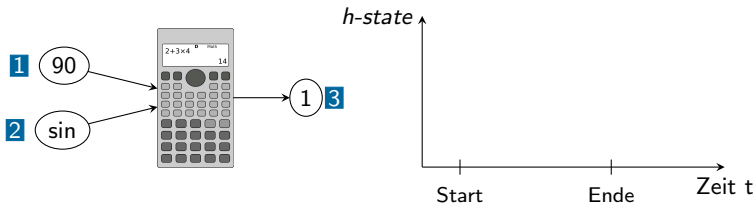
# Beispiel: Dynamischer Zustand eines Taschenrechners

- Verwendung eines Taschenrechners
  - Eingabe der Operanden und Operatoren  $\leadsto$  Ergebnis



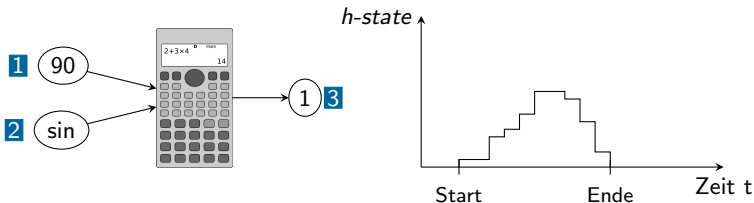
# Beispiel: Dynamischer Zustand eines Taschenrechners

- Verwendung eines Taschenrechners
  - Eingabe der Operanden und Operatoren  $\leadsto$  Ergebnis
  - Der dynamische Zustand ist zu Beginn und am Ende der Berechnung leer
    - Immer, wenn man Berechnungen als **unteilbar** (engl. *atomic*) betrachtet



# Beispiel: Dynamischer Zustand eines Taschenrechners

- Verwendung eines Taschenrechners
  - Eingabe der Operanden und Operatoren  $\leadsto$  Ergebnis
  - Der dynamische Zustand ist zu Beginn und am Ende der Berechnung leer
    - Immer, wenn man Berechnungen als **unteilbar** (engl. *atomic*) betrachtet



- Angenommen die Sinus-Funktion wird iterativ approximiert
  - Der Taschenrechner verwaltet Zustandsvariablen für diese Berechnung
  - Diese Zustandsvariablen sind für die Berechnung unersetzlich
    - Ihr Transfer ist erforderlich, soll die Berechnung anderswo fortgesetzt werden



- Der dynamische Zustand lässt sich weiter aufteilen:



- Der dynamische Zustand lässt sich weiter aufteilen:

## Zustand des physikalischen Objekts

- Kann durch Sensoren erneut erfasst werden
  - Das Replikat wird hier mit der physikalischen Umwelt synchronisiert
- Zur Restauration **kein Transfer erforderlich**



- Der dynamische Zustand lässt sich weiter aufteilen:

## Zustand des physikalischen Objekts

- Kann durch Sensoren erneut erfasst werden
  - Das Replikat wird hier mit der physikalischen Umwelt synchronisiert
- Zur Restauration **kein Transfer erforderlich**

## Standardausgaben (engl. *restart vector*)

- **Sichere Ausgabewerte** für das kontrollierte Objekt
  - Sie werden für die Initialisierung von Aktoren etc. verwendet
  - Sind i. d. R. bereits zum Entwurfszeitpunkt festlegbar
- Zur Restauration **kein Transfer erforderlich**



- Der dynamische Zustand lässt sich weiter aufteilen:

## Zustand des physikalischen Objekts

- Kann durch Sensoren erneut erfasst werden
  - Das Replikat wird hier mit der physikalischen Umwelt synchronisiert
- Zur Restauration **kein Transfer erforderlich**

## Standardausgaben (engl. *restart vector*)

- **Sichere Ausgabewerte** für das kontrollierte Objekt
  - Sie werden für die Initialisierung von Aktoren etc. verwendet
  - Sind i. d. R. bereits zum Entwurfszeitpunkt festlegbar
- Zur Restauration **kein Transfer erforderlich**

## Sonstiger dynamischer Zustand

- der Rest, gehört entsprechend nicht zu den obigen Kategorien
- Zur Restauration ist ein **Transfer notwendig**
  - Kann durch eine Überarbeitung des Systementwurfs evtl. reduziert werden
  - z. B. indem ein zusätzlicher Sensor zum Einsatz kommt





- Der dynamische Zustand einer Berechnung ist für ihre Fortführung auf einem anderen Replikat zwingend erforderlich
  - Die Reintegration eines Replikats erfordert seine Restauration
    - Was letztendlich zu einem Zustandstransfer zwischen den Replikaten führt



Für eine schnelle Fehlererholung sollte er so klein wie möglich sein



- Der dynamische Zustand einer Berechnung ist für ihre Fortführung auf einem anderen Replikat zwingend erforderlich
  - Die Reintegration eines Replikats erfordert seine Restauration
    - Was letztendlich zu einem Zustandstransfer zwischen den Replikaten führt



Für eine schnelle Fehlererholung sollte er so klein wie möglich sein

→ Aus dem Taschenrechner-Beispiel wird klar:

Dies ist der Fall, wenn keine Berechnung aktiv ist!



# Minimierung des dynamischen Zustands

- Der dynamische Zustand einer Berechnung ist für ihre Fortführung auf einem anderen Replikat zwingend erforderlich
  - Die Reintegration eines Replikats erfordert seine Restauration
    - Was letztendlich zu einem Zustandstransfer zwischen den Replikaten führt
- ⚠ Für eine **schnelle Fehlererholung** sollte er **so klein wie möglich** sein
  - Aus dem Taschenrechner-Beispiel wird klar:  
Dies ist der Fall, wenn keine Berechnung aktiv ist!
- 👉 Zu bevorzugen ist der **Grundzustand** (engl. *ground state*)
  - Systemweit ist keine Berechnung aktiv, alle Nachrichtenkanäle sind leer
    - Der dynamische Zustand ist im Grundzustand demnach am kleinsten



# Minimierung des dynamischen Zustands

- Der dynamische Zustand einer Berechnung ist für ihre Fortführung auf einem anderen Replikat zwingend erforderlich
  - Die Reintegration eines Replikats erfordert seine Restauration
    - Was letztendlich zu einem Zustandstransfer zwischen den Replikaten führt



Für eine **schnelle Fehlererholung** sollte er **so klein wie möglich** sein

→ Aus dem Taschenrechner-Beispiel wird klar:

Dies ist der Fall, wenn keine Berechnung aktiv ist!



Zu bevorzugen ist der **Grundzustand** (engl. *ground state*)

- Systemweit ist keine Berechnung aktiv, alle Nachrichtenkanäle sind leer
  - Der dynamische Zustand ist im Grundzustand demnach am kleinsten

→ der Grundzustand ist **für die Reintegration besonders geeignet**



# Minimierung des dynamischen Zustands

- Der dynamische Zustand einer Berechnung ist für ihre Fortführung auf einem anderen Replikat zwingend erforderlich
  - Die Reintegration eines Replikats erfordert seine Restauration
    - Was letztendlich zu einem Zustandstransfer zwischen den Replikaten führt



Für eine **schnelle Fehlererholung** sollte er **so klein wie möglich** sein

→ Aus dem Taschenrechner-Beispiel wird klar:

Dies ist der Fall, wenn keine Berechnung aktiv ist!



Zu bevorzugen ist der **Grundzustand** (engl. *ground state*)

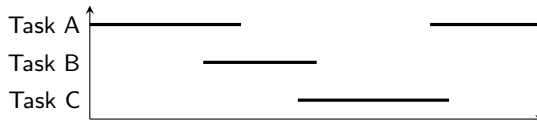
- Systemweit ist keine Berechnung aktiv, alle Nachrichtenkanäle sind leer
  - Der dynamische Zustand ist im Grundzustand demnach am kleinsten
- der Grundzustand ist **für die Reintegration besonders geeignet**
- der Grundzustand sollte **regelmäßig eingenommen** werden
  - Eine zyklische Ablaufstruktur (s. Folie VIII/14) unterstützt dies
  - Nach jedem Zyklus lesen, rechnen, schreiben wäre dies möglich



- Die Einplanungsstrategie hat hier einen signifikanten Einfluss



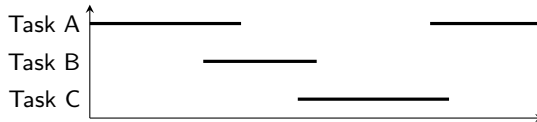
- Die Einplanungsstrategie hat hier einen signifikanten Einfluss
  - Ereignisgesteuerte Einplanung:



- Zeitlicher Ablauf hängt vom Eintreffen der Ereignisse ab
- Das Erreichen des Grundzustands kann **nicht garantiert** werden

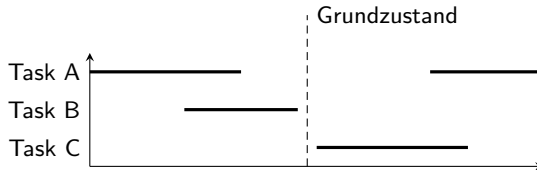
# Erreichbarkeit des Grundzustands

- Die Einplanungsstrategie hat hier einen signifikanten Einfluss
  - Ereignisgesteuerte Einplanung:



- Zeitlicher Ablauf hängt vom Eintreffen der Ereignisse ab
- Das Erreichen des Grundzustands kann **nicht garantiert** werden

- Zeitgesteuerte Einplanung:



- Zyklensynchrone Ausführung erlaubt die **Einplanung des Grundzustands**
- Für Replikdeterminismus (s. Folie VIII/15) ist dies ohnehin sinnvoll





- 1 Überblick
- 2 Verlässlichkeit durch Verteilung
  - Beispiel: Flugsteuerung
  - Verteiltes Fly-By-Wire
- 3 Kommunikation
  - Problemstellung und Grundlagen
  - Zeitgesteuerte Kommunikationssysteme
- 4 Zustandswiederherstellung
  - Vorgehen bei der Reintegration
  - Fehlererholung
  - Interner Zustand
- 5 Zusammenfassung



Verlässlichkeit durch Verteilung  $\mapsto$  Funktionen trennen und replizieren

- Am Beispiel von Fly-By-Wire

Kommunikationssysteme  $\mapsto$  Fehlereingrenzung

- Ereignis- vs. Zustandsnachricht
- Zeitgesteuerte Kommunikation (ARINC 629, TDMA)

Grundlagen der Reintegration

- Reaktiv, proaktiv und reaktiv-proaktiv
- Vorwärts- und Rückwärtskorrektur
- Initialzustand und dynamischer Zustand
- Bestandteile und Minimierung des dynamischen Zustands



- [1] AUDSLEY, N. C. ; GRIGG, A. :  
Timing Analysis of the ARINC 629 Databus for Real-Time Applications.  
In: *Proceedings of the ERA Avionics Conference and Exhibition*.  
Heathrow, UK : ERA Technology Ltd, Nov. 20–21, 1996, S. 10.1.1–10.1.11
  
- [2] BERWANGER, J. ; PELLER, M. ; GRIESSBACH, R. :  
*byteflight* — A New Protocol for Safety Critical Applications.  
In: *Proceedings of the 28th FISITA World Automotive Congress*.  
Seoul, Korea : FISITA, Jun. 12–15, 2000
  
- [3] *Kapitel 12.*  
In: BRIERE, D. ; FAVRE, C. ; TRAVERSE, P. :  
*Electrical Flight Controls, From Airbus A320/A330/A340 to Future Military Transport Aircraft: A Family of Fault-Tolerant Systems*.  
CRC Press LLC, 2001 (The Electrical Engineering Handbook Series). –  
ISBN 978–0849383489
  
- [4] CARLOW, G. D.:  
Architecture of the space shuttle primary avionics software system.  
In: *Communications of the ACM* 27 (1984), Nr. 9, S. 926–936.  
<http://dx.doi.org/10.1145/358234.358258>. –  
DOI 10.1145/358234.358258. –  
ISSN 0001–0782



- [5] CHARARA, H. ; SCHARBARG, J.-L. ; ERMONT, J. ; FRABOUL, C. :  
Methods for bounding end-to-end delays on an AFDX network.  
In: *Real-Time Systems, 2006. 18th Euromicro Conference on IEEE*, 2006, S. 10–pp
- [6] FLEXRAY CONSORTIUM:  
*FlexRay protocol specification 2.1 Revision A*.  
FlexRay Consortium, 2005. –  
<http://www.flexray.com>
- [7] KOPETZ, H. :  
*Real-Time Systems: Design Principles for Distributed Embedded Applications*.  
Kluwer Academic Publishers, 1997. –  
ISBN 0-7923-9894-7
- [8] KOPETZ, H. ; GRÜNSTEIDL, G. :  
TTP—A Time-Triggered Protocol for Fault-Tolerant Real-Time Systems.  
In: *Proceedings of the Twenty-Third Annual International Symposium on Fault-Tolerant Computing (FTCS-23)*.  
Toulouse, France : IEEE, Jun. 22–24, 1993, S. 524–533
- [9] LAMPORT, L. :  
A New Solution of Dijkstra's Concurrent Programming Problem.  
In: *Communications of the ACM* 8 (1974), Nr. 7, S. 453–455



- [10] MALEK, M. :  
Responsive Computer Systems.  
In: *Real-Time Systems 7* (1994), Nr. 3. –  
Special Issue
  
- [11] POLEDNA, S. :  
*Replica Determinism in Fault-Tolerant Distributed Real-Time Systems.*  
Vienna, Austria, Technical University of Vienna, Diss., 1995. –  
Research Report 28/95
  
- [12] TRAVERSE, P. ; LACAZE, I. ; SOUYRIS, J. :  
Airbus Fly-By-Wire: A Process Toward Total Dependability.  
In: *Proceedings of the 25th Congress of International Council of the Aeronautical Sciences (ICAS '06)*, 2006

