Diplomverteidigung

Automatisches Entwurfs- und Entwicklungssystem für harte Echtzeitsysteme

Jan Blumenthal

17.07.2002

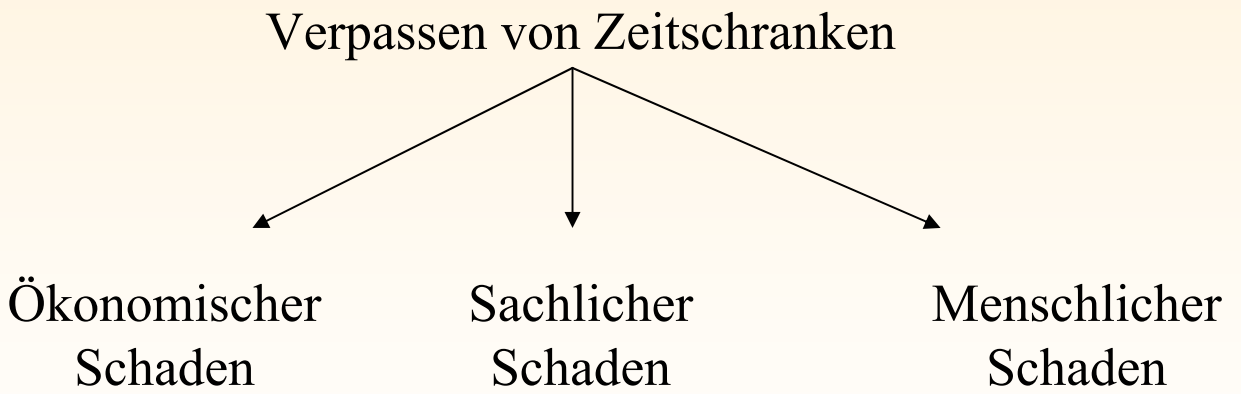
Universität Rostock, Fachbereich für Elektrotechnik und Informationstechnik
Institut für Angewandte Mikroelektronik und Datentechnik

Gliederung

- Vortrag
  - Einführung
  - Bestandsaufnahme
  - Konzept des Frameworks YASA 2
  - Vorstellung einzelner Komponenten
    - Executives
    - Scheduler
  - Zusammenfassung
- Vorführung

Allgemeine Ziele

- Echtzeitsysteme:
  - funktionaler Nachweis
  - nicht-funktionaler (zeitlicher) Nachweis

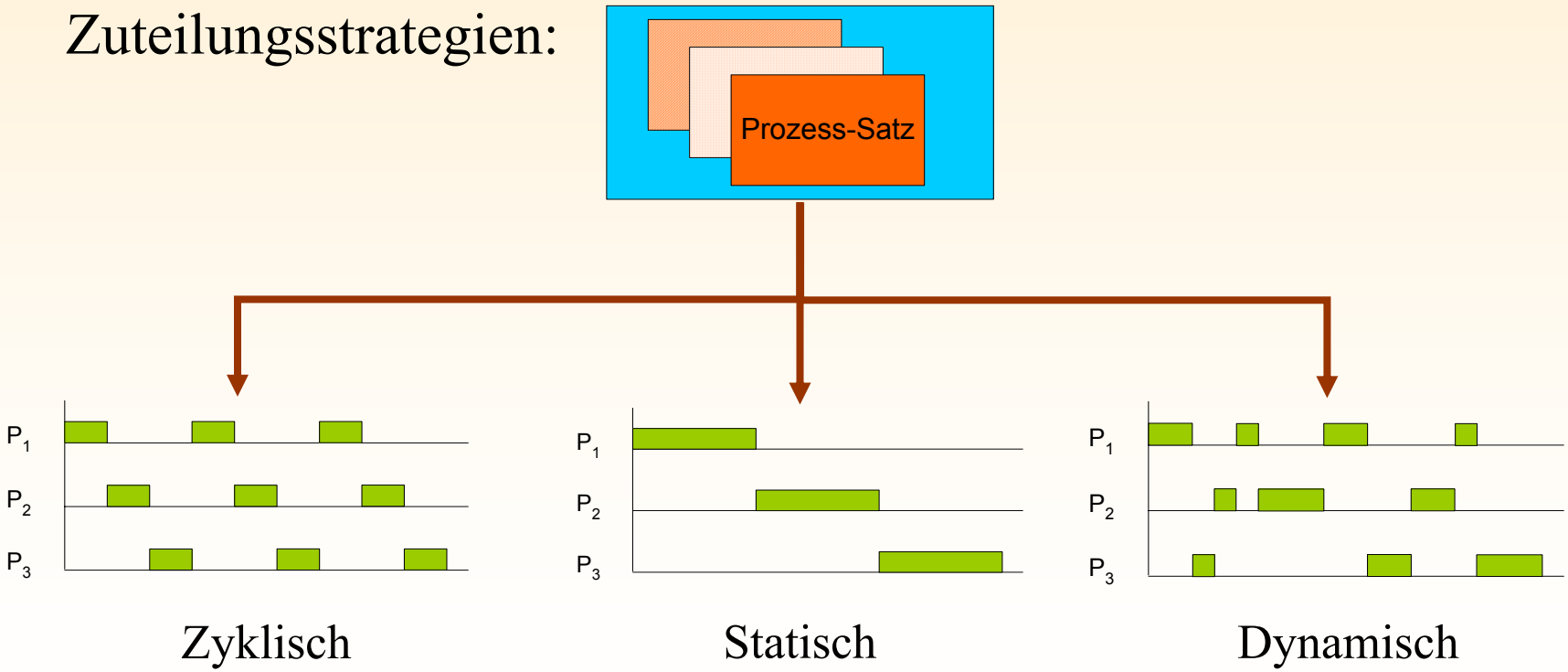


Konzentration auf Optimierung des Schedulingverhaltens

Scheduler

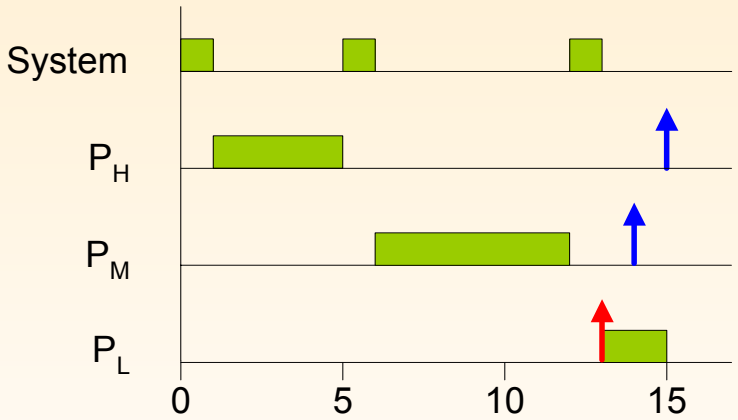
Aufgabe: Zuteilung der Prozesse auf die CPU

Zuteilungsstrategien:

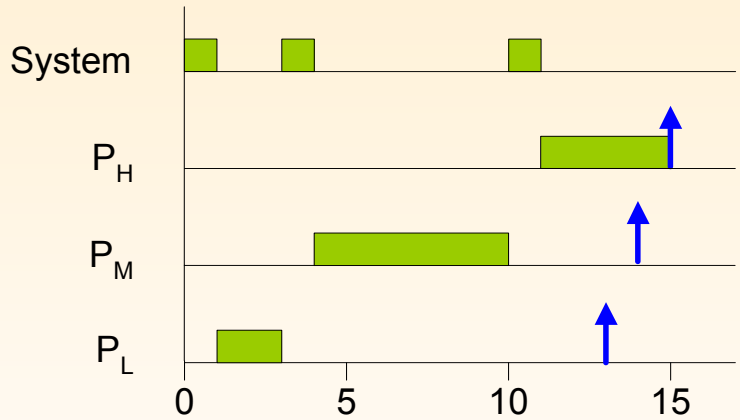


Optimierung des Schedulings

Prioritätsbasiert:



Earliest Deadline First:



- Analyse des Verhaltens nötig
- Austausch des Schedulers erforderlich

Synchronisationsprotokolle

Semaphoren:

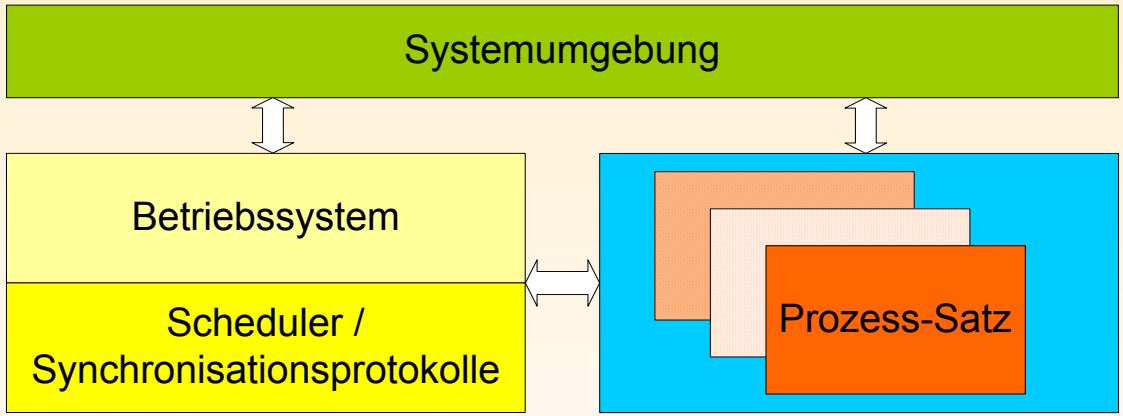
- Datenaustausch von Prozessen
- Absicherung von kritischen Abschnitten
- Betriebsmittelanforderungen

Synchronisationsprotokolle:

- Vermeidung von Effekten bei Belegung von Semaphoren
  - Prioritäteninversion
  - Deadlocks
- Enge Kopplung an Scheduler

Austausch der Synchronisationsprotokolle erforderlich

Aktuelle Systeme



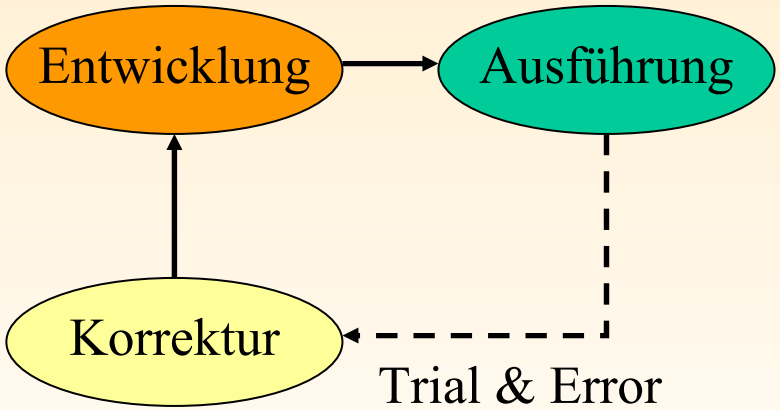
Nachteile:
- Scheduler fest mit Betriebssystem verbunden
- Keine dynamischen Scheduler bzw. Synchronisationsprotokolle
- Schedulingverlauf nicht überprüfbar

Ziele von YASA

- Optimierung des Schedulingverhaltens in Echtzeitbetriebssystemen:
  - Umfangreiche Analyse des Schedulingverhaltens
  - Austausch der Scheduler
  - Nutzung weiterer Synchronisationsprotokolle

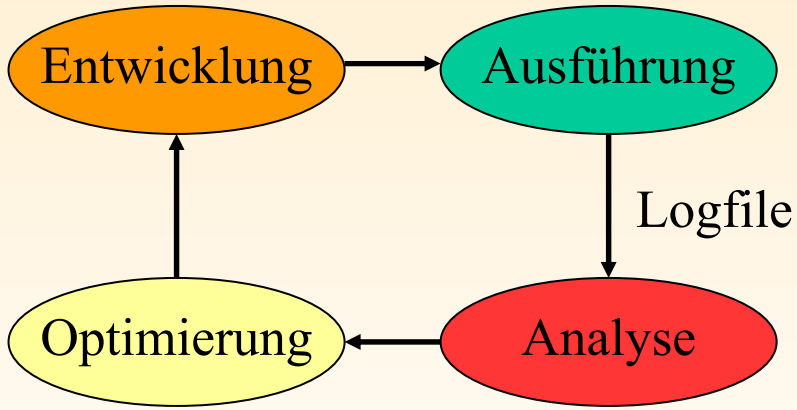
Anwendungsentwicklung

Aktueller Stand:



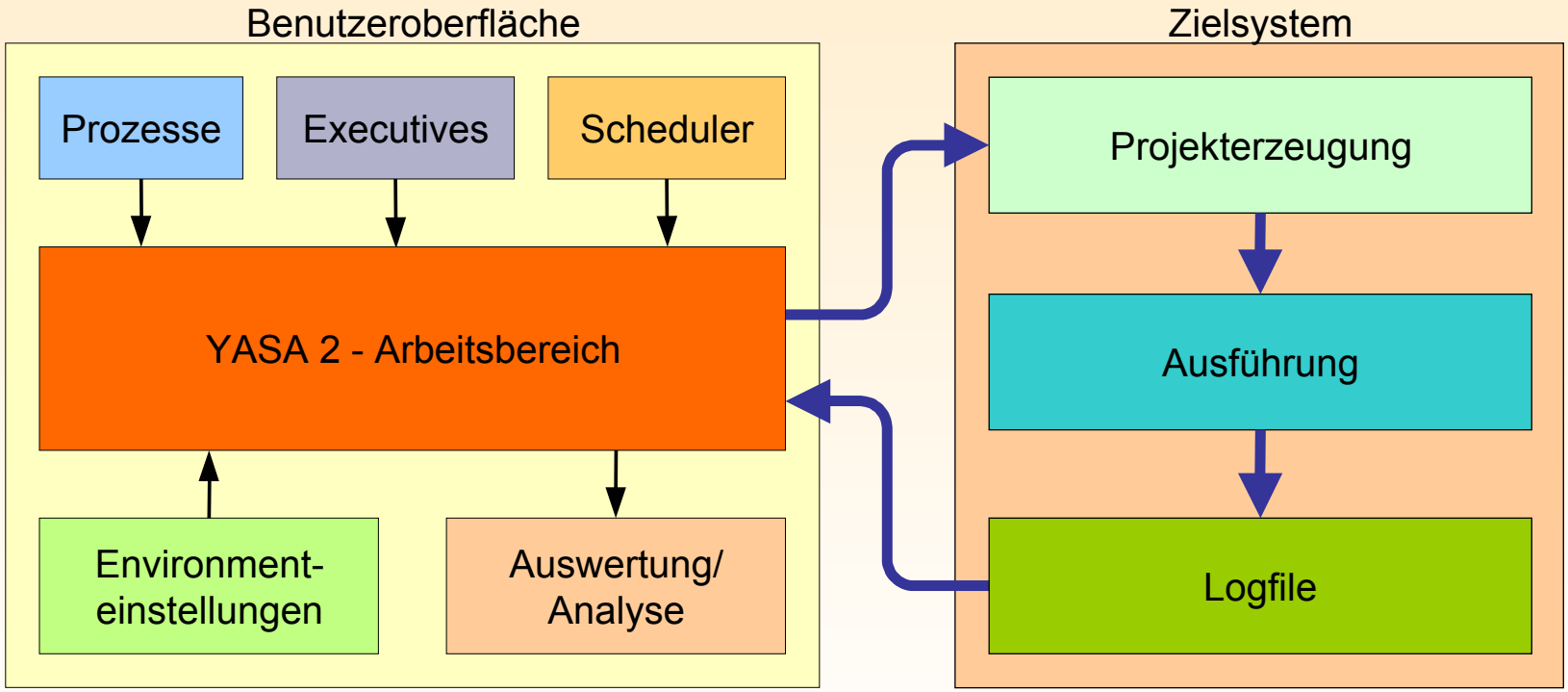
- Unbekannter Schedulingverlauf
- Keine Analyse
- Keine bzw. schlechte Fehlererkennung

YASA 2:



- Protokollierter Schedulingverlauf
- Analyse des Schedulingverlaufs möglich

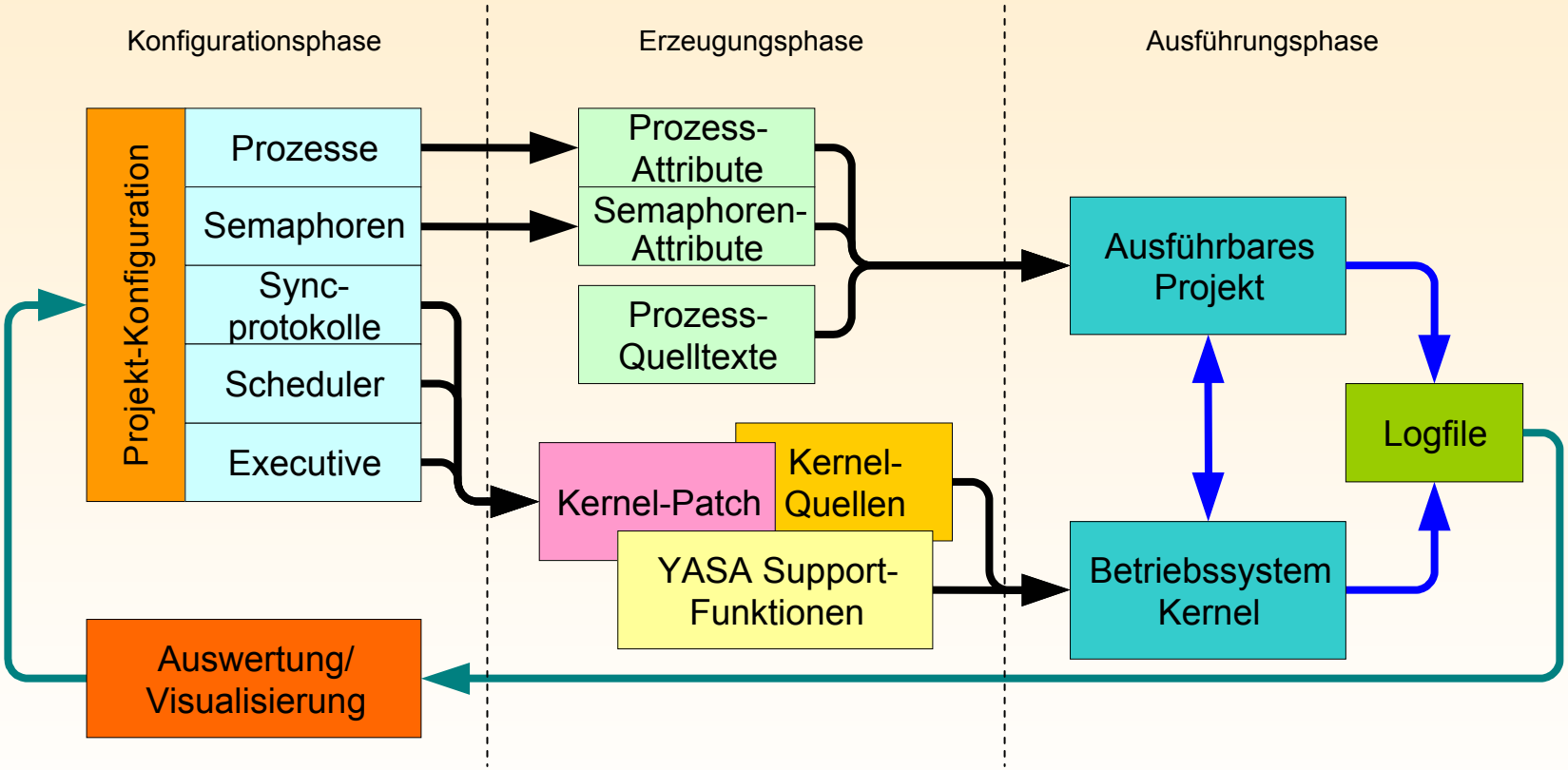
Gliederung von YASA



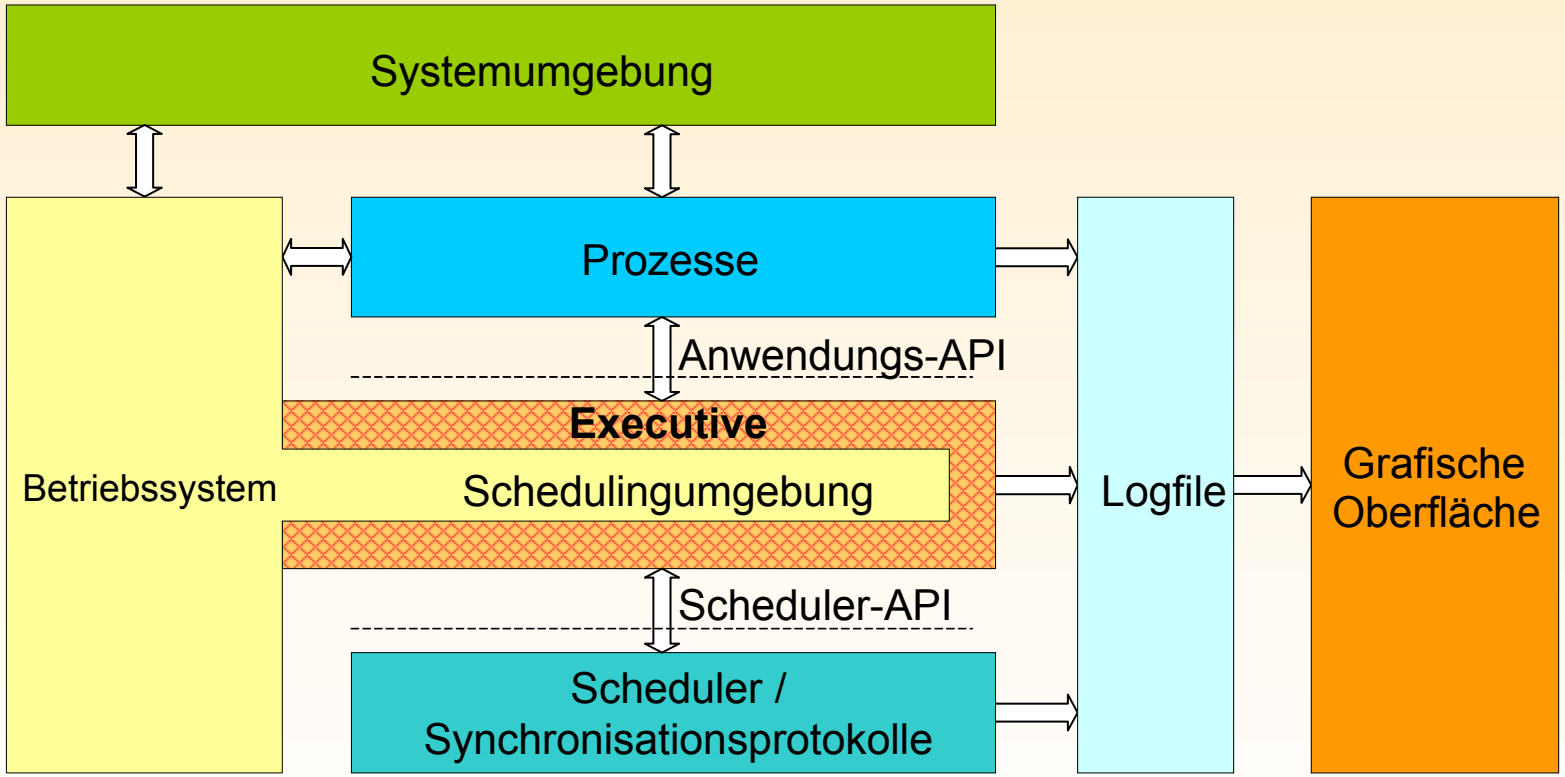
- Projektverwaltung & Konfiguration
- Quelltextgenerierung
- Auswertung

- Kompilierung
- Ausführung
- Logfile-Erstellung

Zyklus eines Projekts



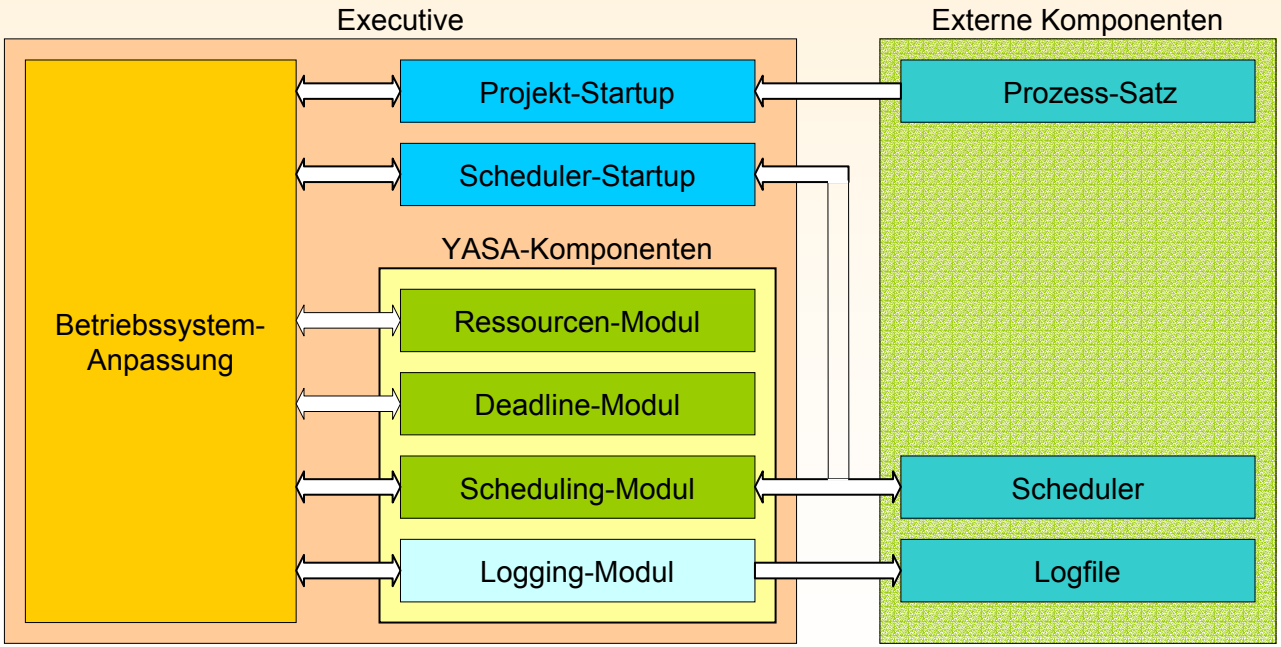
Arbeitsweise von YASA



➡ Kapselung der Schedulingumgebung durch Executives

Aufbau eines Executives

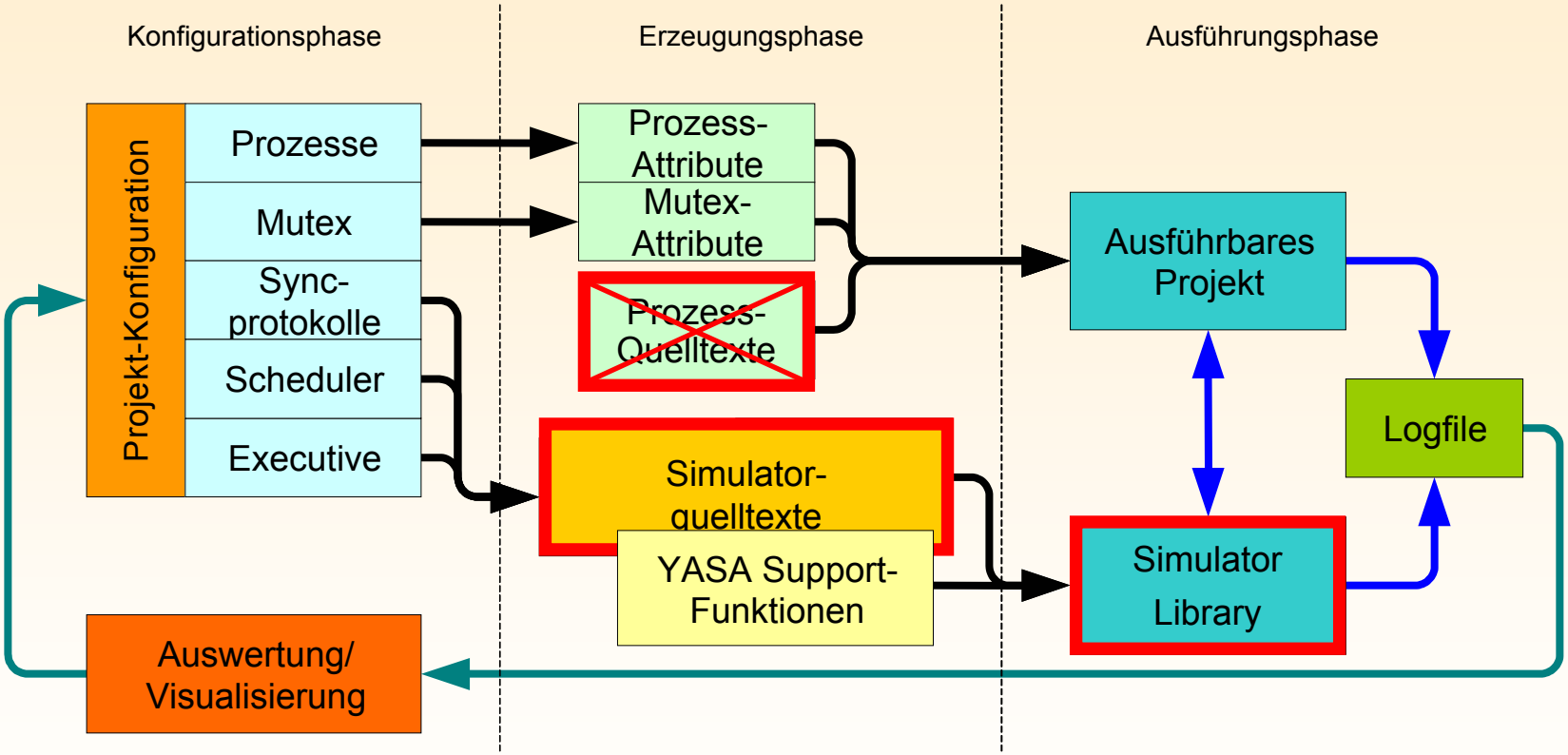
- Virtuelle Schedulingumgebung
- Schnittstellen für Scheduler und Prozesse
- Unterstützung portabler Scheduler



Simulator-Executive

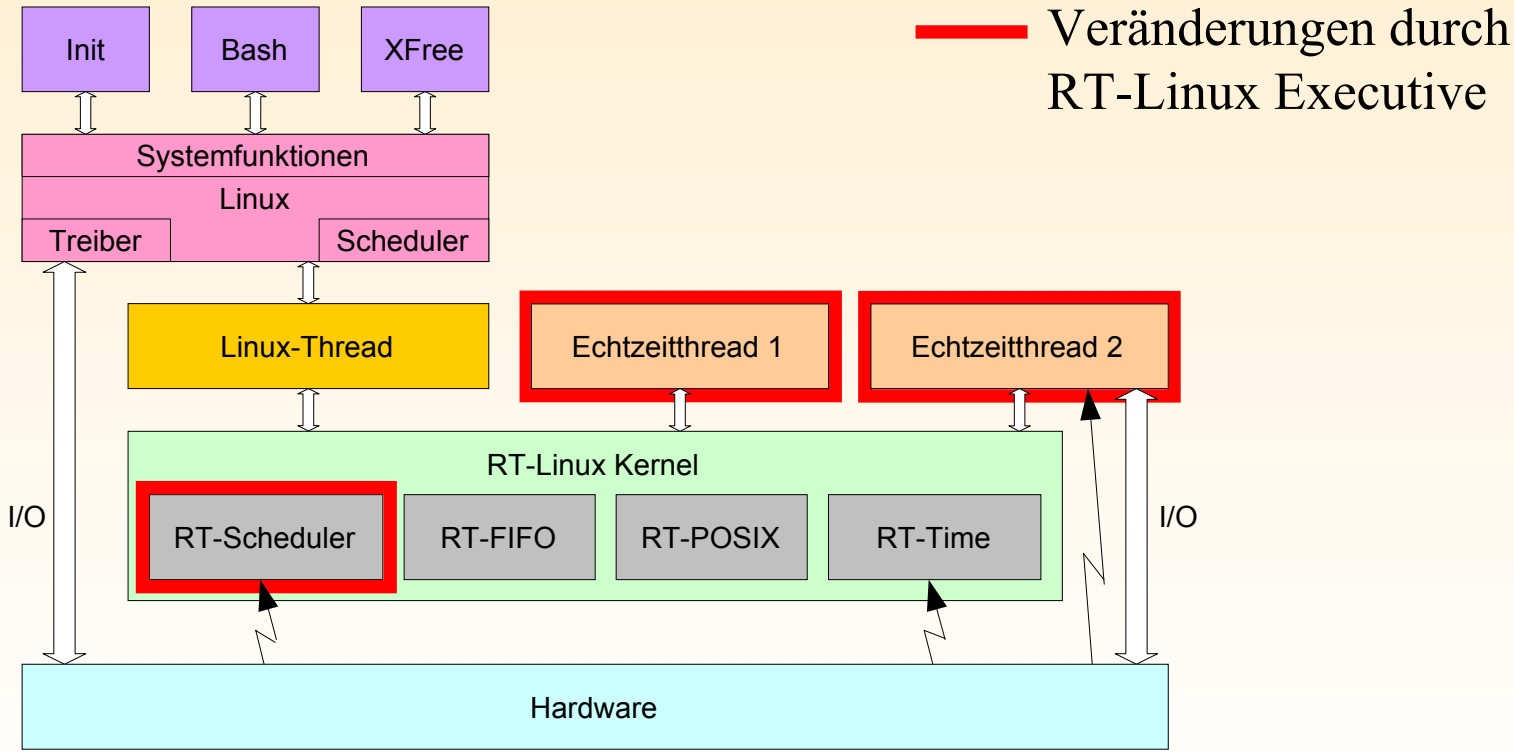
- Ausführung der Anwendung mit synthetischer Last
- Virtueller Programmlauf
  - Prozesszustände
  - Ressourcenanforderungen
- Multiprozessorfähig
- Plattformunabhängig
- Anwendungsgebiete:
  - Nachweis der theoretischer Analysen
  - Entwicklung von Schedulern

Projektzyklus im Simulator



Besonderheiten des Simulator-Executives

RT-Linux Executive



Scheduler

- Wechsel zur Laufzeit
- Implementierung
  - Executive-unabhängig
  - Plattformunabhängig
  - Keine Limitierung der Zeitauflösung, Prozessanzahl

Verfügbare Scheduler:

Statische Scheduler	Dynamische Scheduler	Zyklische Scheduler
Prioritätsbasiert	Earliest Deadline First	Round Robin
Rate Monotonic	Least Laxity First	First Come – First Serve
Deadline Monotonic	Enhanced Least Laxity First	

Synchronisationsprotokolle

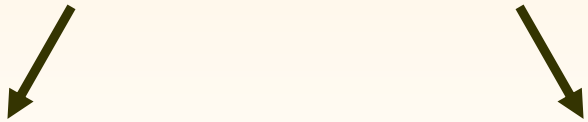
- Synchronisation der Betriebsmittelanforderungen
  - Verhinderung der Prioritäteninversion
  - Deadlockvermeidung
- Einführung dynamischer Prioritäten
  - Entkopplung von Synchronisationsprotokollen und Schedulern

Verfügbare Synchronisationsprotokolle:

Statisch	Dynamisch
Priority Inheritance Protocol	Dynamic Priority Ceiling Protocol
Priority Ceiling Protocol	Stack Resource Policy
Ceiling Semaphore Protocol	

Benutzeroberfläche YASA 2

- Plattformunabhängig durch Klassenbibliothek Qt
- Objekt-orientierte Programmierung (261 Klassen)
- Nutzung von Entwurfsmustern
- Multilingual (UNICODE)
- Ausgaben in grafischer und tabellarischer Form



Taskdiagramm
Mutexdiagramm
Prozessordiagramm

Tasktabelle
Mutextabelle
Prozessortabelle

Einsatzgebiete

Zusammenfassung YASA 2

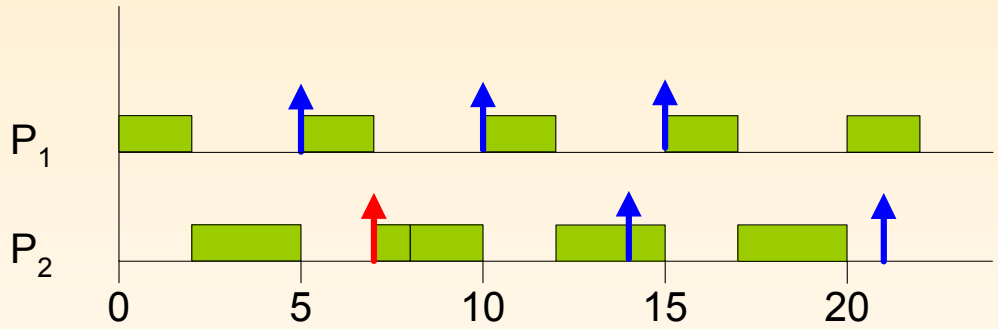
- Entwicklungsframework für bestehende Echtzeitbetriebssysteme
  - Optimierung des Schedulingverhaltens von Echtzeitanwendungen
  - Virtuelle Schedulingumgebung
  - Plattformunabhängige Scheduler & Synchronisationsprotokolle
  - Komponenten zur Protokollierung & Deadlinebehandlung
  - Umfangreiche Auswertungen in der Benutzeroberfläche

- Ergebnisse
  - Veröffentlichung: 13th IEEE International RSP Workshop, Darmstadt, 2002
  - Anfragen der Industrie (ABB)

Vielen Dank

Anhang

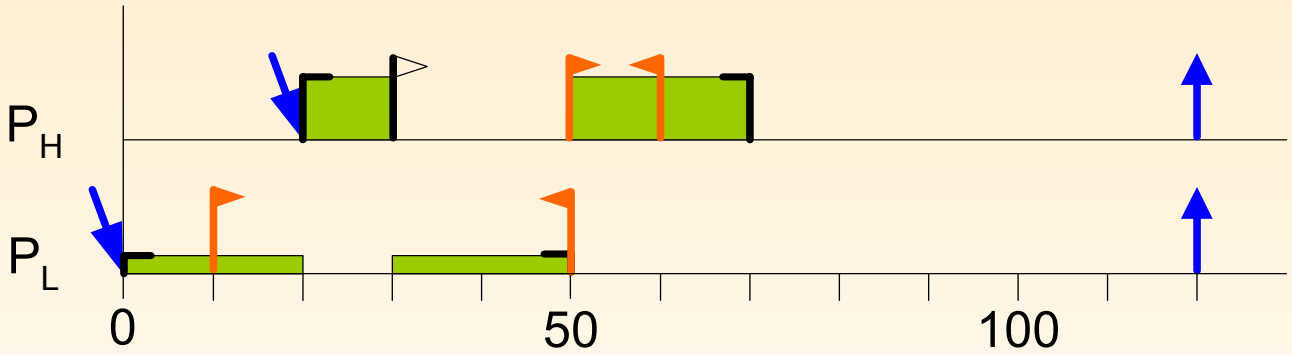
Fehlerhaftes RMS-Scheduling



$$Auslastung_U = \frac{2}{5} + \frac{4}{7} = \frac{34}{35} \approx 0{,}97$$

Prozess	Priorität	Periode	Deadline	Rechenzeit	Auslastung
1	Hoch	5	5	2	0,4
2	niedrig	7	7	4	0,57

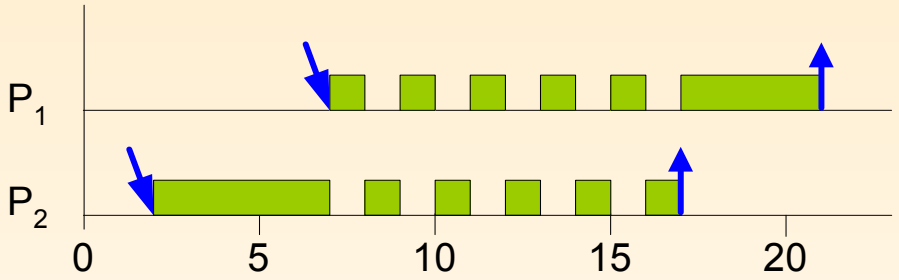
Prioritäteninversion



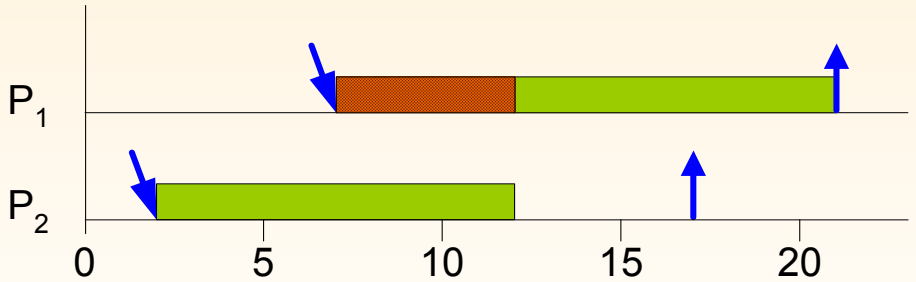
- P_L setzt bei t=10 Semaphore S
- P_H startet bei t=20 aufgrund höherer Priorität
- P_H versucht S bei t=30 zu setzen und blockiert
- P_L arbeitet bis zur Freigabe von S (t=50)
  - P_L erhält ‚faktisch' höhere Priorität
  - Blockiert und verzögert höher priorisierten Prozess P_H

Beispiel: Trashing-Effekt

Least Laxity First (LLF)

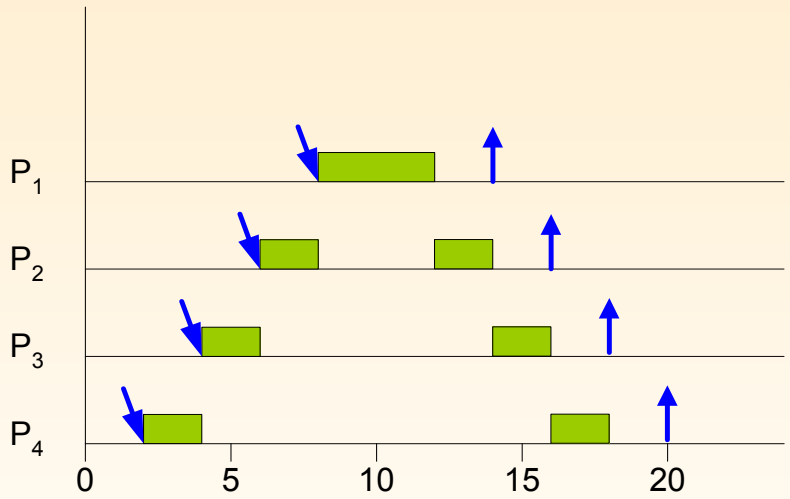


Enhanced Least Laxity First (ELLF)

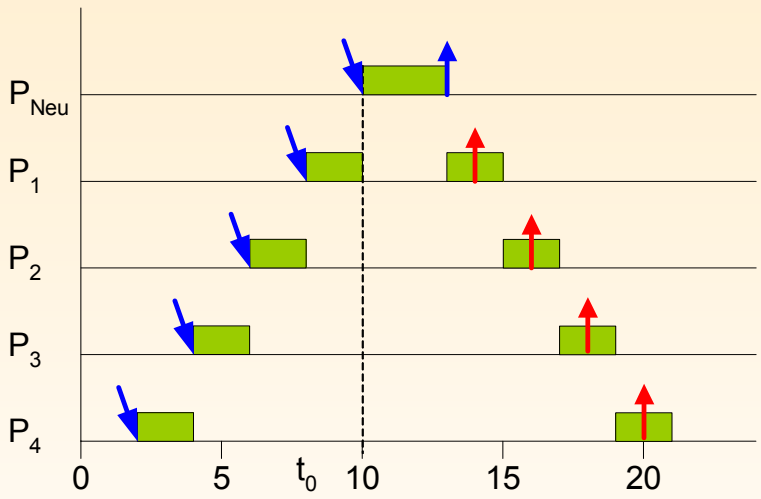


	L(2)	L(7)	L(8)	L(9)	L(15)	L(16)
P1	-	5	5	4	1	1
P2	5	5	4	4	1	0

Beispiel: Domino-Effekt

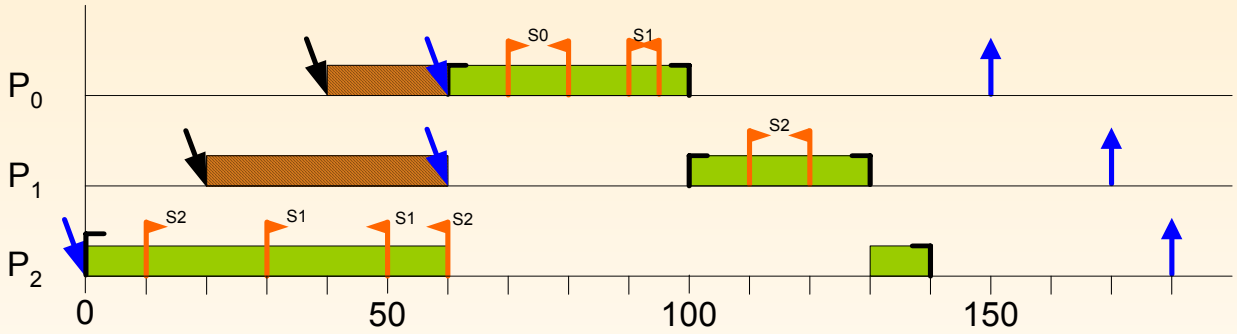


Prozess-Satz hält alle Zeitschranken ein.



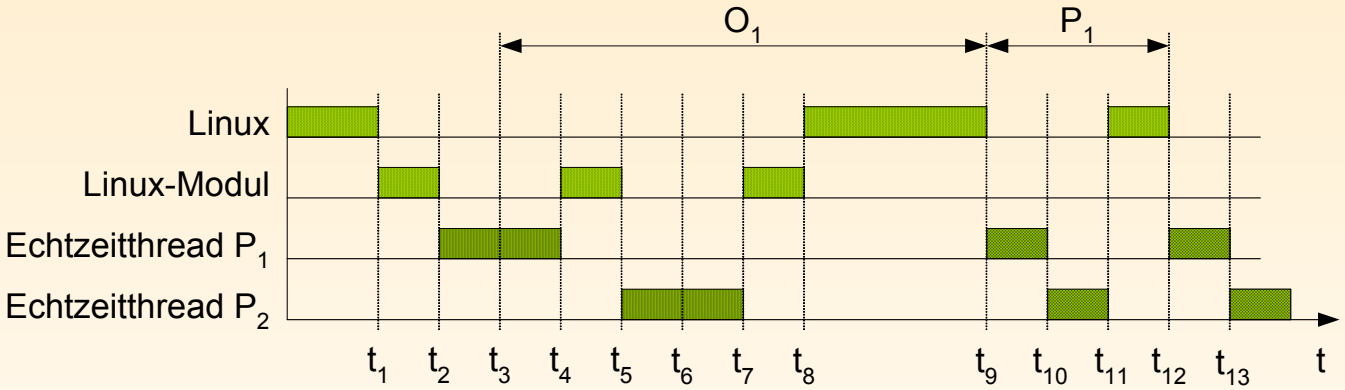
Hinzufügen des Prozesses P_{Neu} bewirkt ein Verpassen aller weiteren Deadlines.

Beispiel: Kernel Priority Protocol



- Aktivierung von P_0 und P_1 wird durch verwendetes Synchronisationsprotokoll verzögert.

Starten von RT-Linux Prozessen



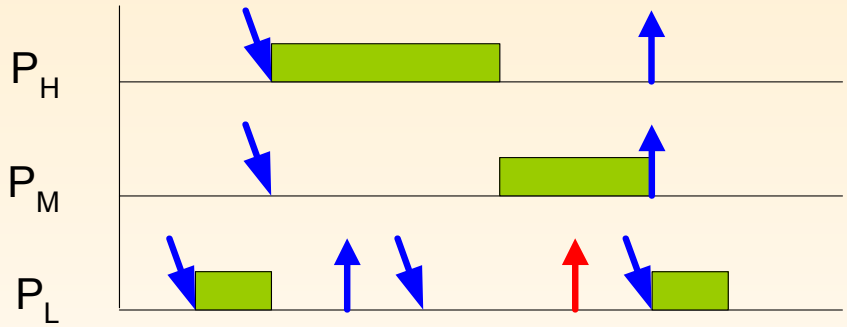
O_i: Offset des ersten Starts von $Thread_i$

P_i: Periode

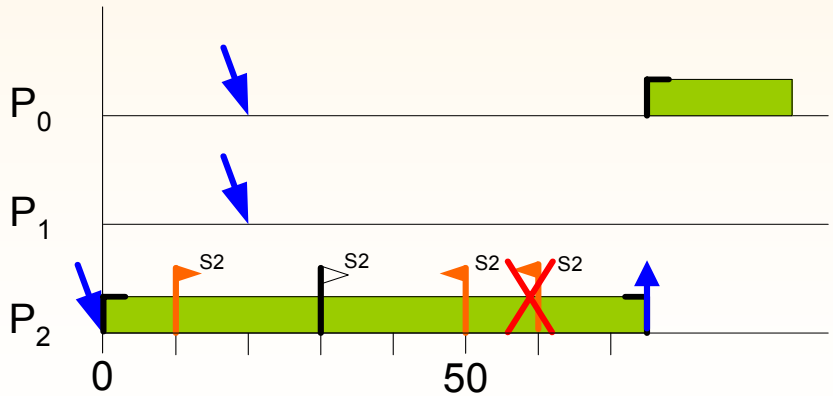
- Erzeugung, Start und Initialisierung von P_1 (t_2-t_3)
- Übergang in periodischen Zustand (t_3) und Suspendierung (t_4)
- Äquivalentes Verhalten für P_2 (t_5-t_7)
- Start der 1. Periode bei t_9
- Problem: Initialisierung von P_2 kann nicht rechtzeitig abgeschlossen werden,wenn $t_3+O_1<t_7$

Fehlererkennung

- Periode übersprungen



- Fehler bei Nutzung von Semaphoren
  - Erneutes Setzen
  - Unerlaubtes Freigeben



YASA-Voraussetzungen

Framework:

- UNIX-Entwicklungsumgebung (GNU-Make, gcc)
- Executive und Betriebssystem-Quelltext

Benutzeroberfläche:

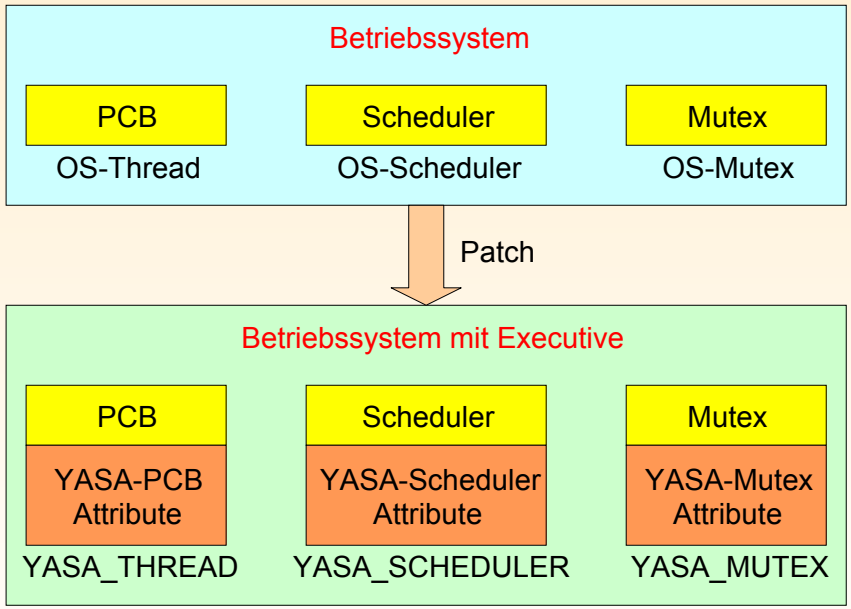
- UNIX-Entwicklungsumgebung (GNU-Make, gcc)
- Qt Klassenbibliothek

Vergleich YASA 1 vs. YASA 2

YASA 1	YASA 2

• Simulation
• Windows (MFC)
• Theoretische Analyse
• Eingeschränkter Plugin-Port für Scheduler
 	
• Serverprozesse
	• Simulation & Ausführung
	• Portabel
	• Praktische Auswertung
	• Plugin-Ports für
	  - Betriebssysteme
	  - Scheduler
	- Keine Serverprozesse

Erweiterung eines Betriebssystems

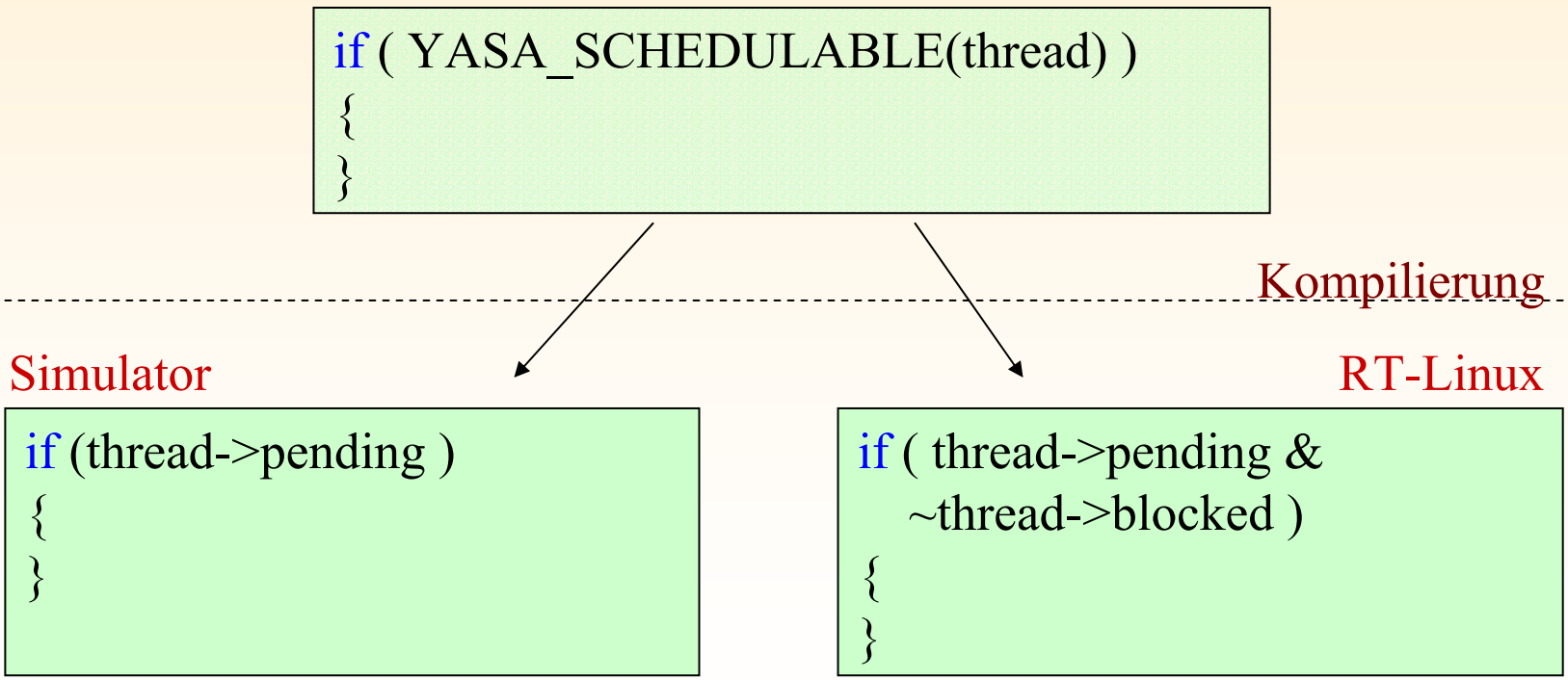


Erweiterung durch Präprozessormakros

- Geschwindigkeitsvorteile
- Verbesserung von Wartung & Pflege

Portabilität von Schedulern

- Einsatz von Präprozessormakros
- Definition der Makros in Executives

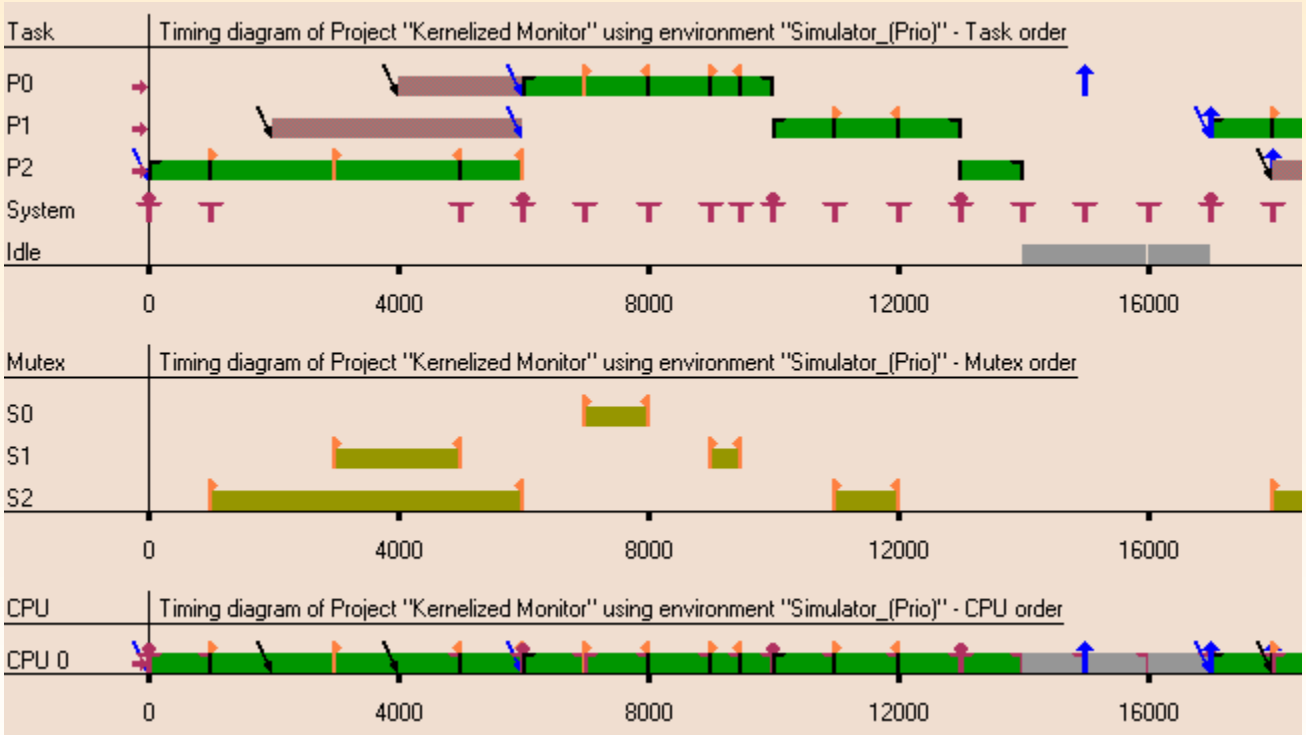


Dynamische Prioritäten

- Prioritätstyp von Prozess P_x abhängig vom Scheduler
- Zugriff auf Variablen durch Mappingmakros während der Kompilierung

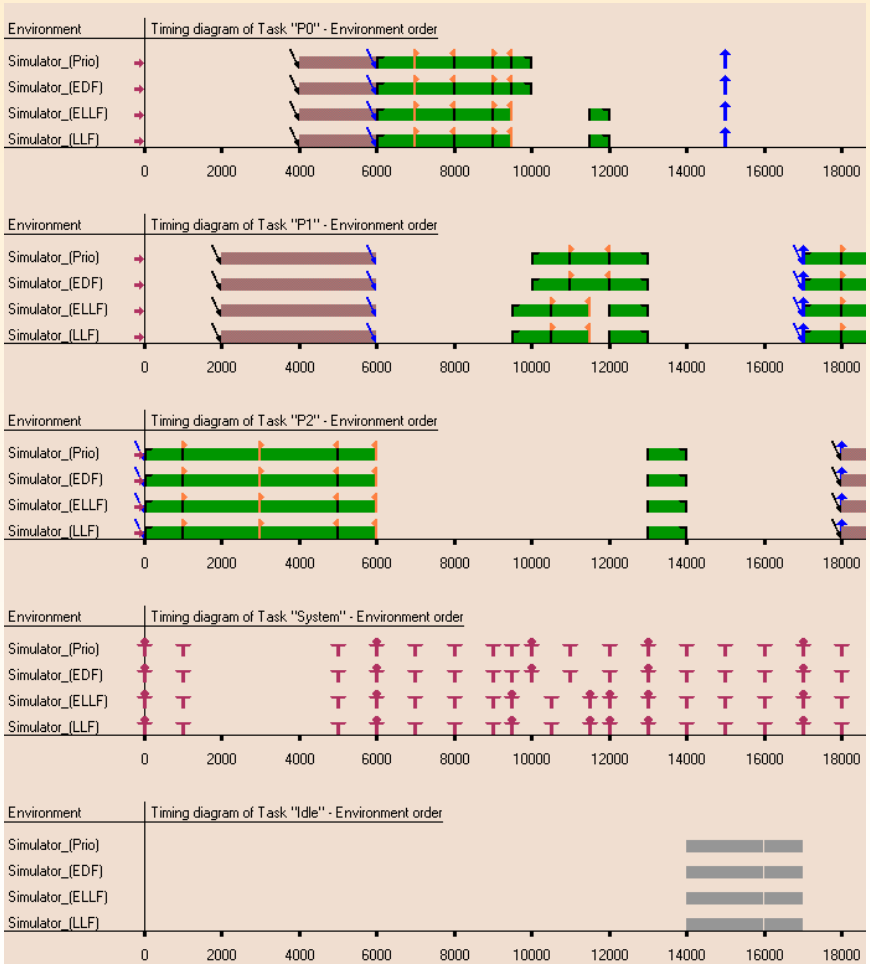
Name	Typ	Datentyp	Vergleichsoperation
Priorität	Statisch	int	$P_1 > P_2$
Schlupf	Dynamisch	YASA_TIME	$P_1 < P_2$
Deadline	Dynamisch	YASA_TIME	$P_1 < P_2$
Benötigte Zeit	Dynamisch	YASA_TIME	$P_1 > P_2$
Kürzeste Zeit	Dynamisch	YASA_TIME	$P_1 < P_2$

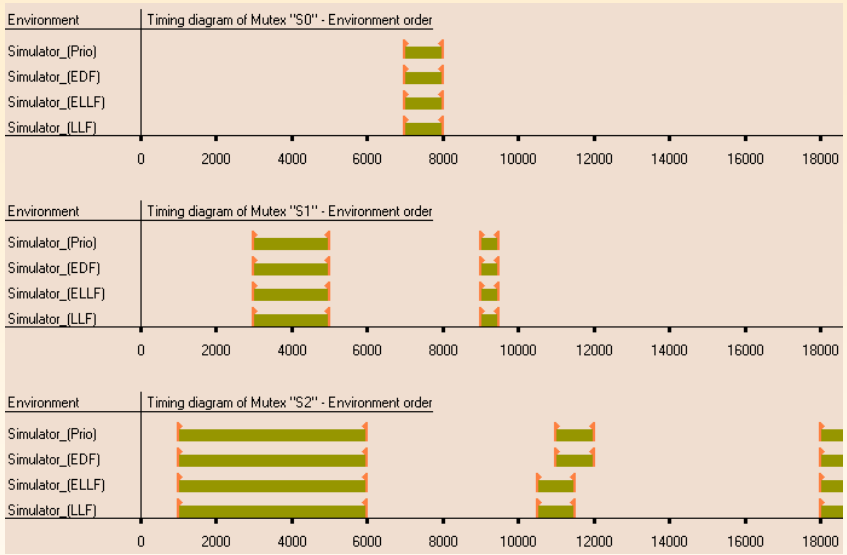
Grafische Auswertung I

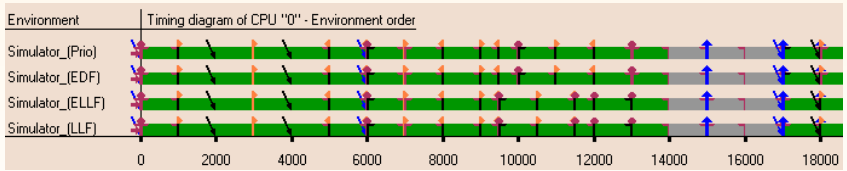


Darstellung des Zeitverhaltens einer Projektausführung

Grafische Auswertung II

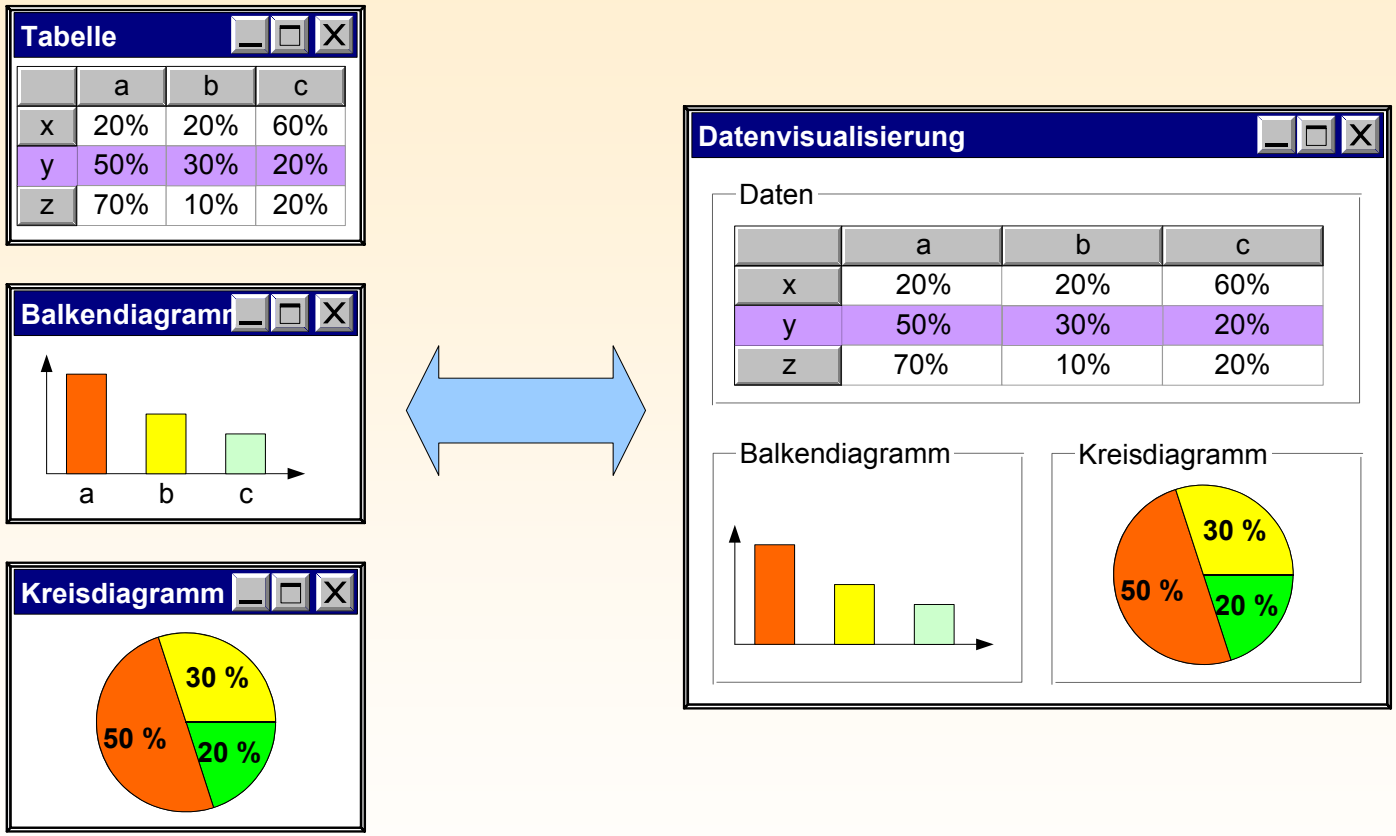






Darstellung des Zeitverhaltens mehrerer Projektausführungen

Signal-Slot Mechanismus



→ Anwendung des Beobachter-Musters