ROBOTS, SOFTWARE, MAYHAM?

Dietmar Schreiner
TU Wien
Institut für Computersprachen

ÜBERBLICK

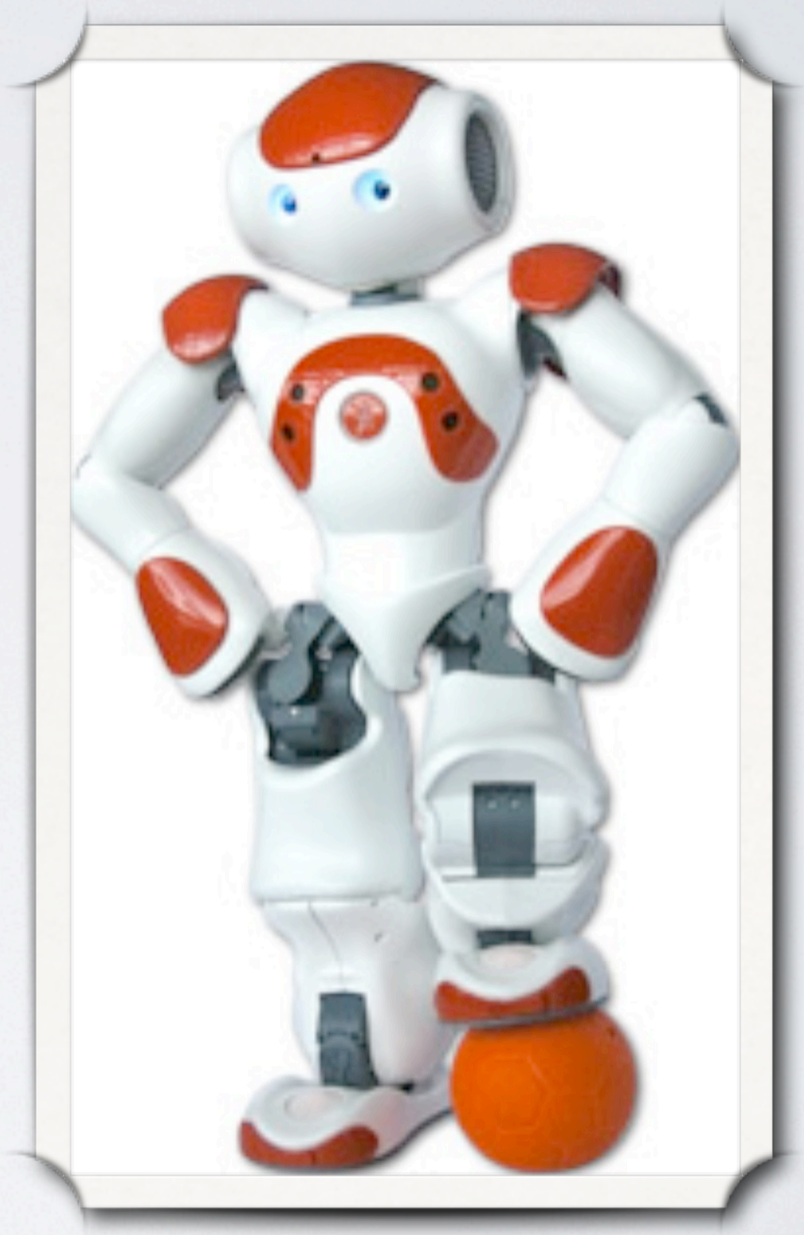
- **Domainüberblick**
  - System Eigenschaften
  - State-of-the-Art Architekturen
- **Probleme Statement**
  - Beispiel Austrian-Kangaroos
- **Also welche Methodik?**
- **Beispiel**

MOBILE AUTONOME ROBOTER

- **Interaktion von HW/SW mit echtem Environment**
- **Unterstützung und Ergänzung menschlicher Tätigkeiten**
  - „Gefährliche“ Einsatzgebiete
    - Kernkraftwerke, Katastrophenszenarien, Unterwasserarbeiten
  - Schwere körperliche Arbeiten
    - Bergbau, z.B. TBMs
  - Haushalt und Altenpflege
  - UVs

FUNKTIONALITÄT EINES MARS

- **Sensorik**
  - Wahrnehmung von Umwelt und Ego
- **Lokalisierung**
  - Bestimmung der eigenen Position innerhalb der Welt
- **Kognition**
  - Planen, Schließen, Lernen
- **Motorik**
  - Reaktive Regelung



SYSTEM EIGENSCHAFTEN (1)

- **Reaktiv**
  - System reagiert auf Änderungen im Environment (oder auf interne Veränderungen)
  - System wird kontinuierlich von einem diskreten Systemzustand in einen weiteren übergeführt
- **Zeitgesteuert**
  - Ansteuerung von Sensoren und Aktuatoren ist zeitdiskret (real-time)
  - Zustandsübergänge können durch Timer ausgelöst werden

SYSTEM EIGENSCHAFTEN (2)

- **Nebenläufig / Parallel**
  - Autonome Teilsysteme werden nebenläufig ausgeführt
  - Teilsysteme verändern Weltmodell und haben daher indirekt Einfluss auf andere
- **Probabilistisch**
  - Wahrnehmung und Weltmodell ist eine diskrete und

SYSTEM EIGENSCHAFTEN (3)

- **Verhalten des Roboters ergibt sich aus der Menge der vorhandenen „Behaviors“**

- Ereignisgesteuerte „Verhalten“ (reaktiv)
- Typischerweise als DEAs modelliert
  - z.B. Matlab/Simulink-Stateflow, rDNA

SOFTWARE-ARCHITEKTUR (1)

- **Subsumption Architektur**
  - Augmented Finite State Machines (AFSM)
    - DEA mit Zeitkomponente
      - Konditionale Transitionen
        - Inhibitorische Knoten (Message Filter)
  - Komposition von DEAs
- **Problem**: Komplexe Systeme
  - nur schlecht wartbar
  - Austausch von Teilsystemen schwierig

SOFTWARE-ARCHITEKTUR (2)

- **Schichten Architektur & Komponenten/Services**
  - 3-Schichten Architektur in Top-Level View
    - Nachdenkend/Ausführend/Reaktiv
- Jede dieser Schichten ist Komponenten- bzw. Service-orientiert zusammengesetzt
  - Erhöhung der Wiederverwendbarkeit und Wartbarkeit
  - z.B. LWCCM, NaoQi

SOFTWARE-ARCHITEKTUR (3)

- **Middleware**
  - **Abstraktion von Hardware**
    - z.B. Player Stage
  - **Abstraktion von Kommunikation und Koordination**
    - z.B. MS CCR & DSS
  - **Abstraktion von Nebenläufigkeit und Zeit**
    - z.B. URBI Execution Kernel
  - **Problem**: Zahlreiche spezialisierte Arten von Middleware auf einem Roboter.

PROBLEMSTATEMENT

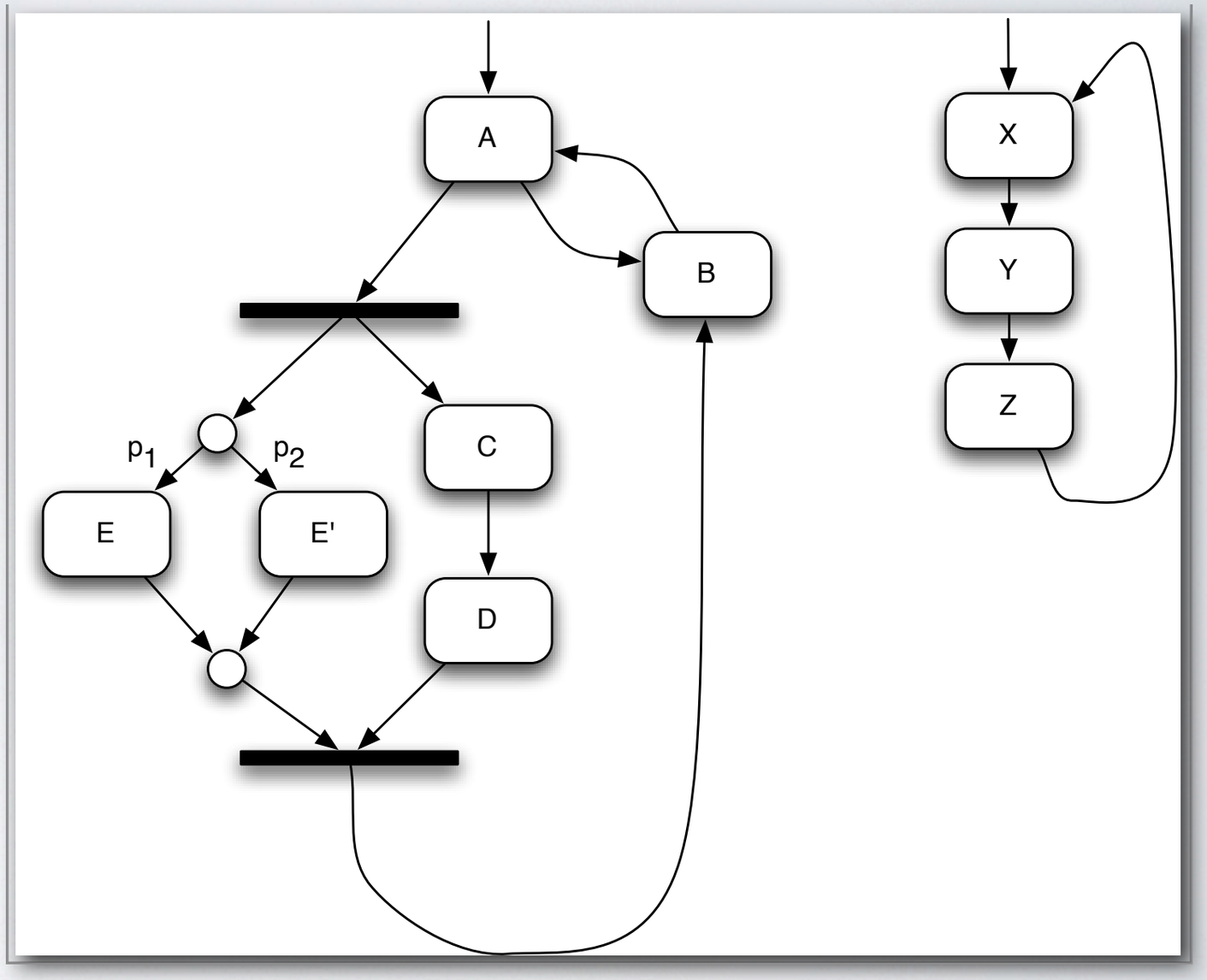
- Entwickler sollen einzelne Behaviors konstruieren
  - dabei möglichst wenig belastet werden mit
    - Synchronisation zwischen abhängigen Behaviors
    - Synchronisation zwischen Behaviors und Ressourcen
    - Zugriffskontrolle auf Ressourcen
- Gesamtsystem soll (automatisch) überprüfbar sein

WELCHE METHODIK ALSO?

- **Model Driven Development**
  - Modelle bieten spezialisierte Sichten
  - Entwickler kümmern sich nur um „ihr“ Problem
- **CBSE**
  - Baukastenprinzip für Wiederverwendbarkeit und Wartbarkeit
  - Timing-aware / Ressource-aware Software Composition
- **Datenfluß / Ressourcen getriebenes Design**
  - Explizite Modellierung von Ressourcen ermöglicht automatische Synthese von Synchronisationscode

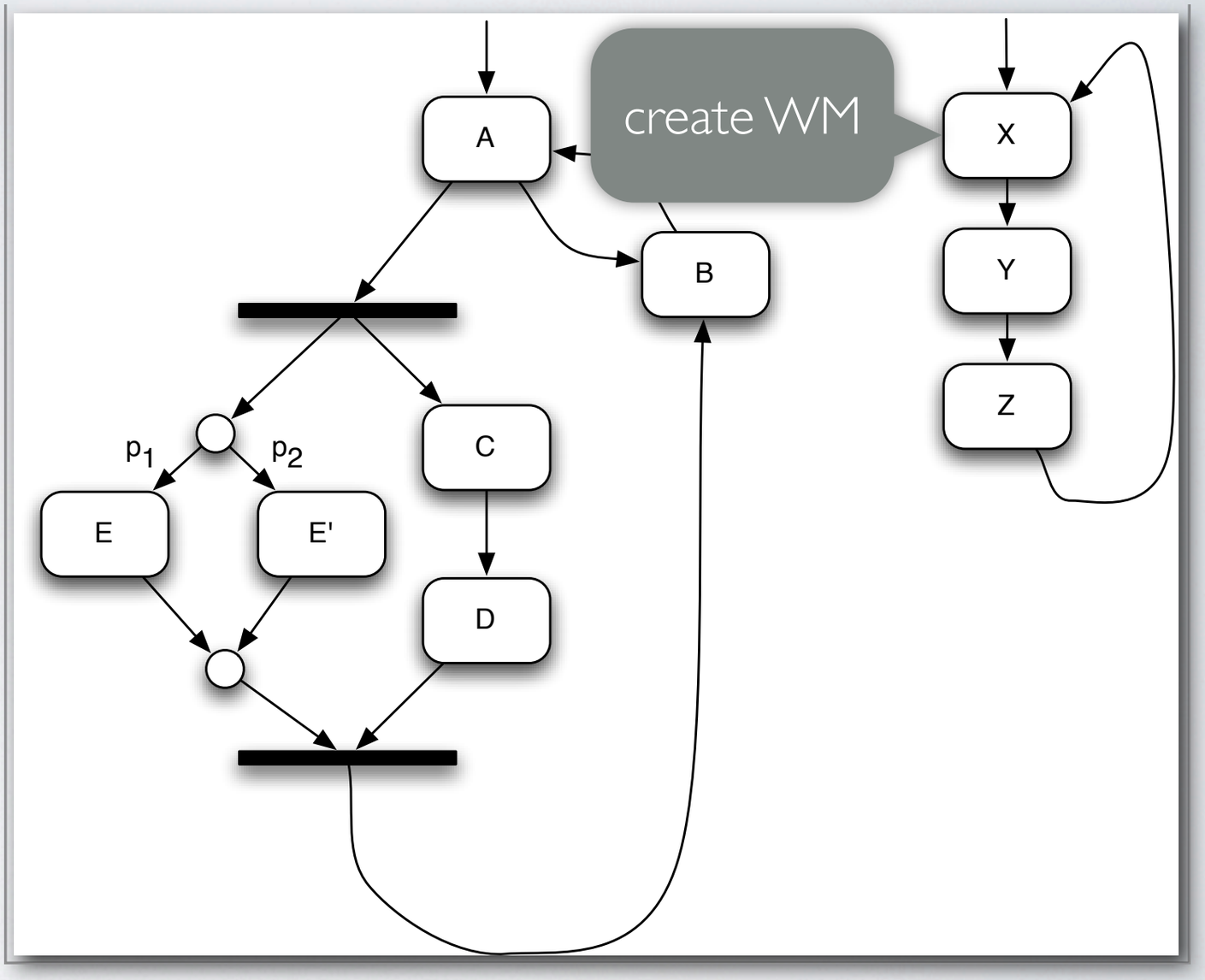
BEISPIEL (1)
TRADITIONELLER ANSATZ

- „Klassische“ Modellierung als Zustandsautomaten
- Ressourcen-Konflikte nicht sichtbar



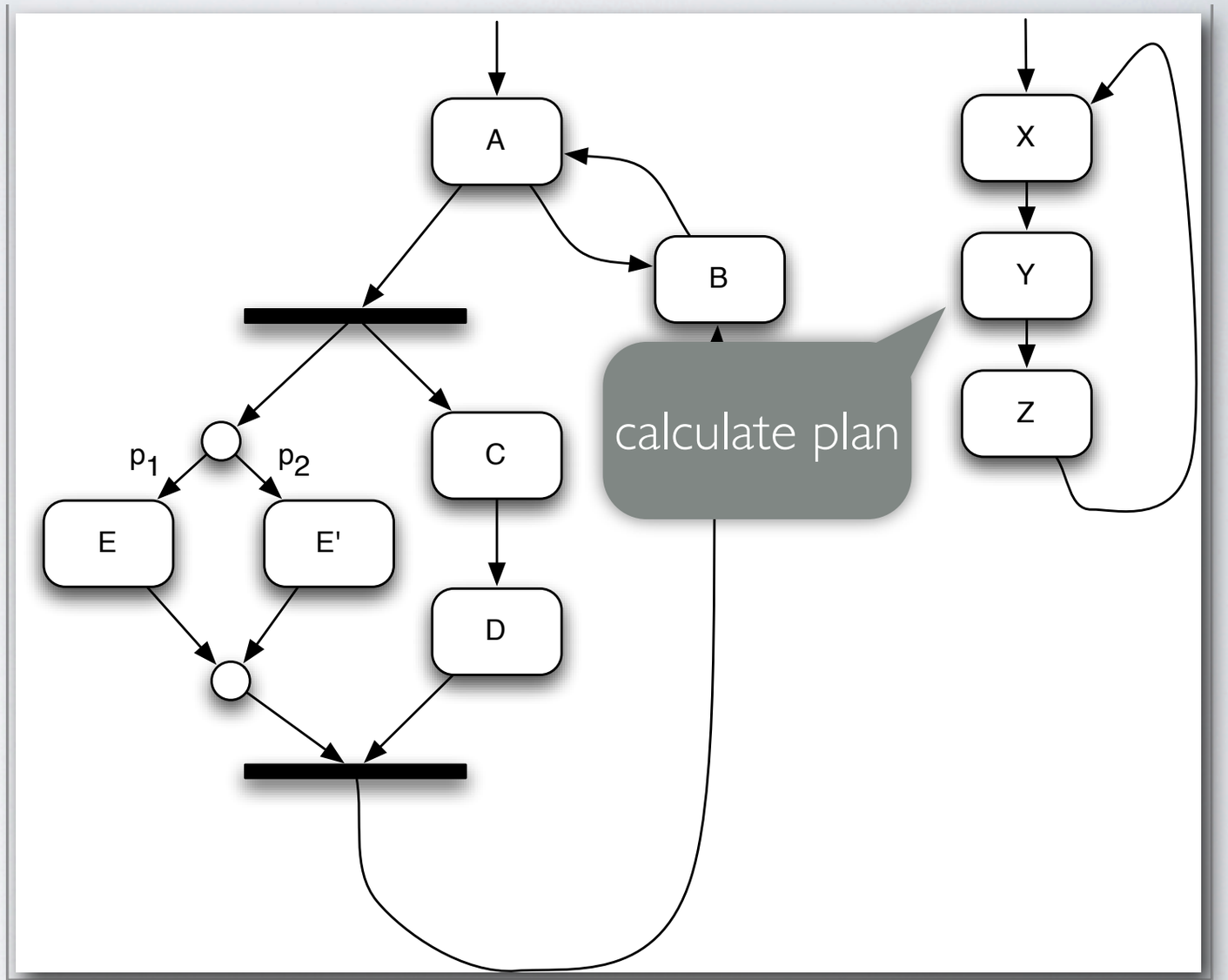
BEISPIEL (1)
TRADITIONELLER ANSATZ

- „Klassische“ Modellierung als Zustandsautomaten
- Ressourcen-Konflikte nicht sichtbar



BEISPIEL (I)
TRADITIONELLER ANSATZ

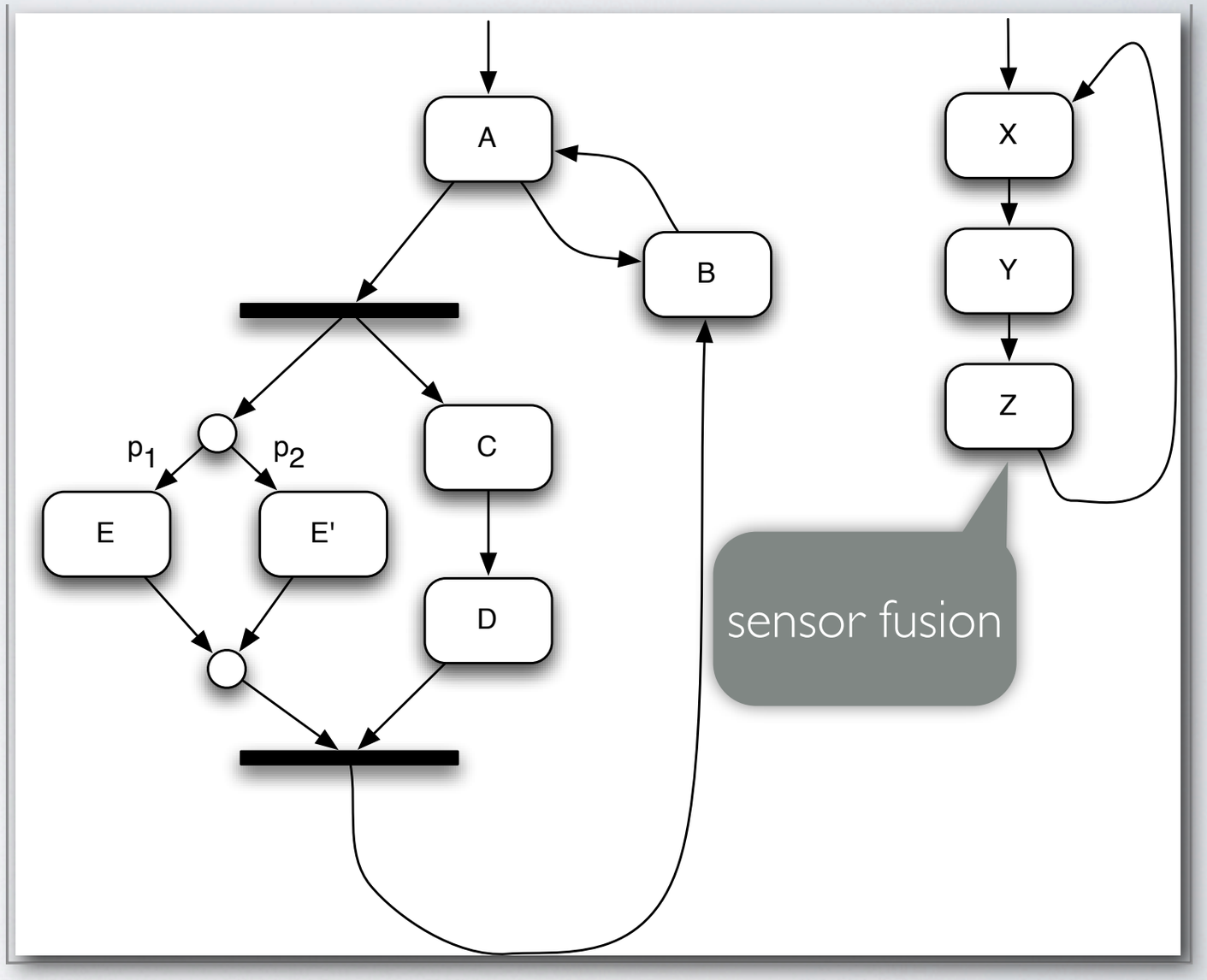
- „Klassische“ Modellierung als Zustandsautomaten
- Ressourcen-Konflikte nicht sichtbar



BEISPIEL (I)
TRADITIONELLER ANSATZ

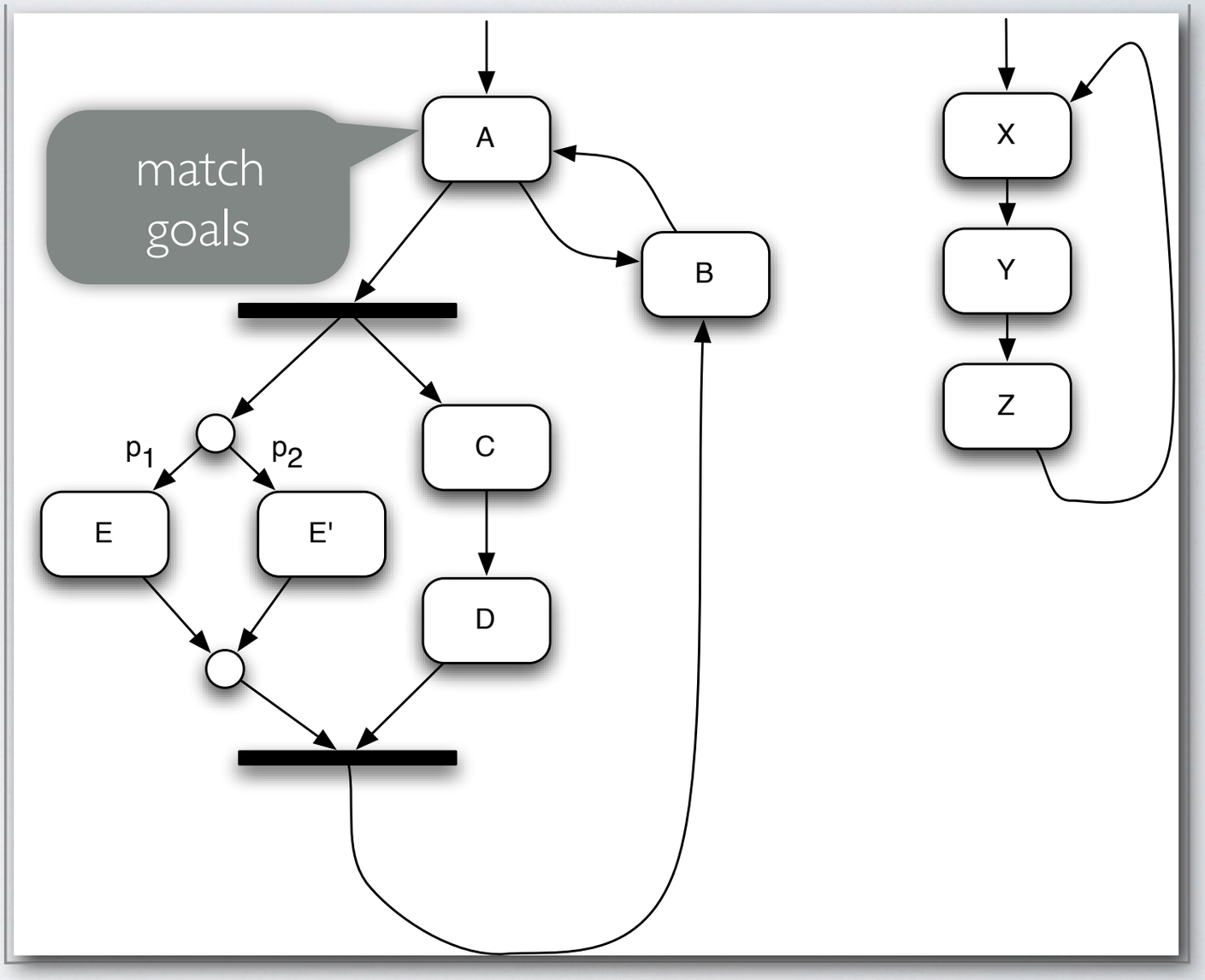
- „Klassische“ Modellierung als Zustandsautomaten

- Ressourcen-Konflikte nicht sichtbar



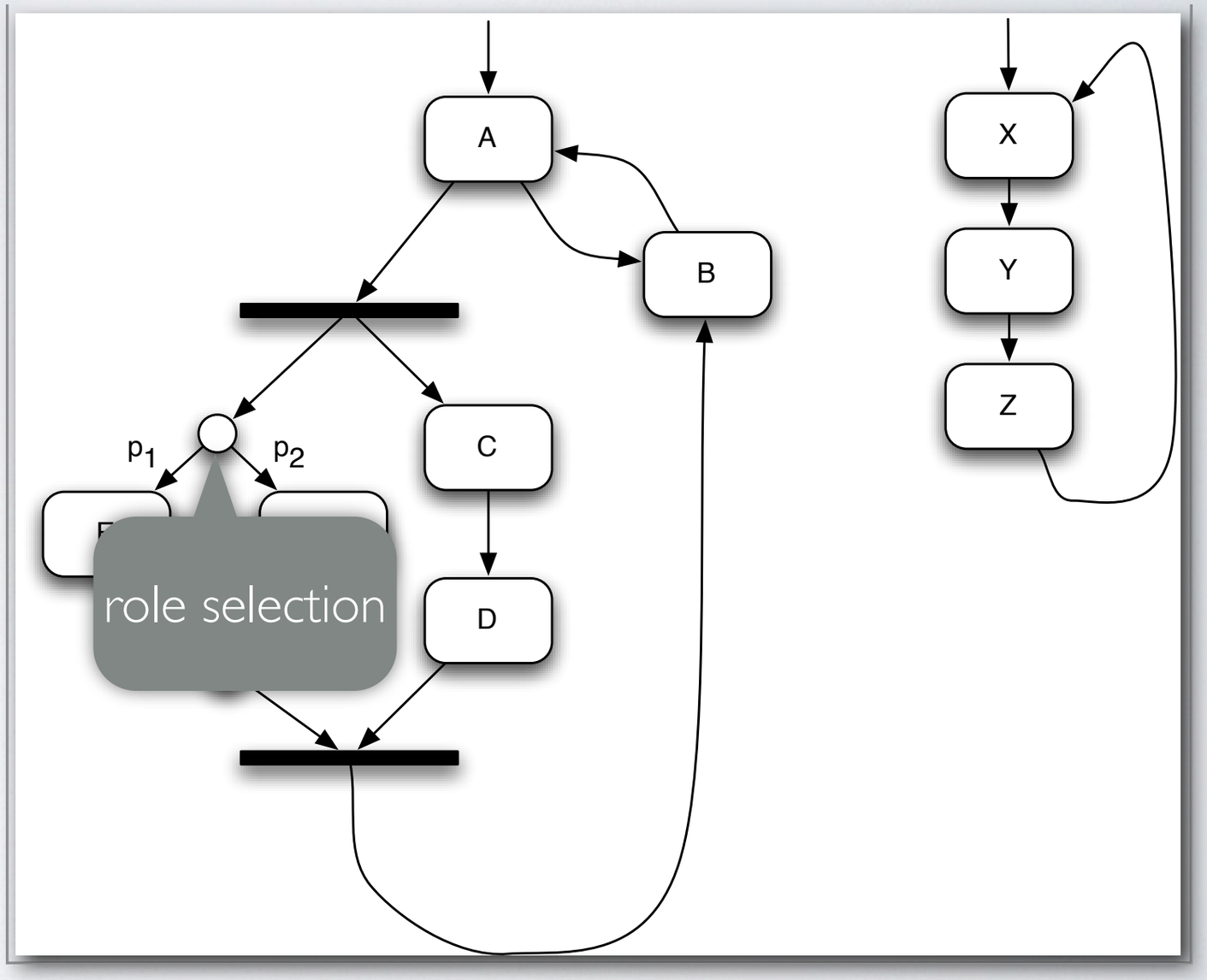
BEISPIEL (1)
TRADITIONELLER ANSATZ

- „Klassische“ Modellierung als Zustandsautomaten
- Ressourcen-Konflikte nicht sichtbar



BEISPIEL (I)
TRADITIONELLER ANSATZ

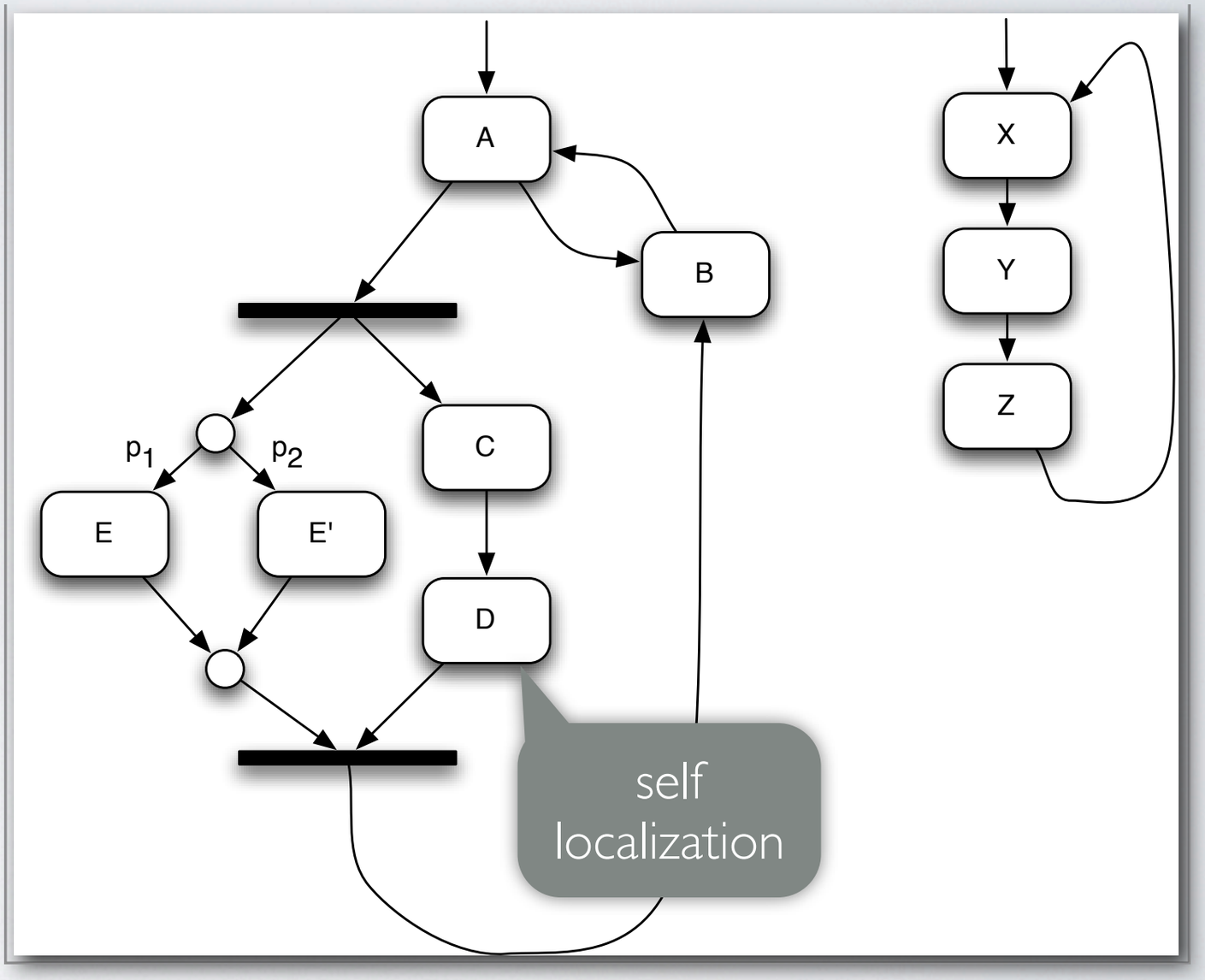
- „Klassische“ Modellierung als Zustandsautomaten
- Ressourcen-Konflikte nicht sichtbar



BEISPIEL (1)
TRADITIONELLER ANSATZ

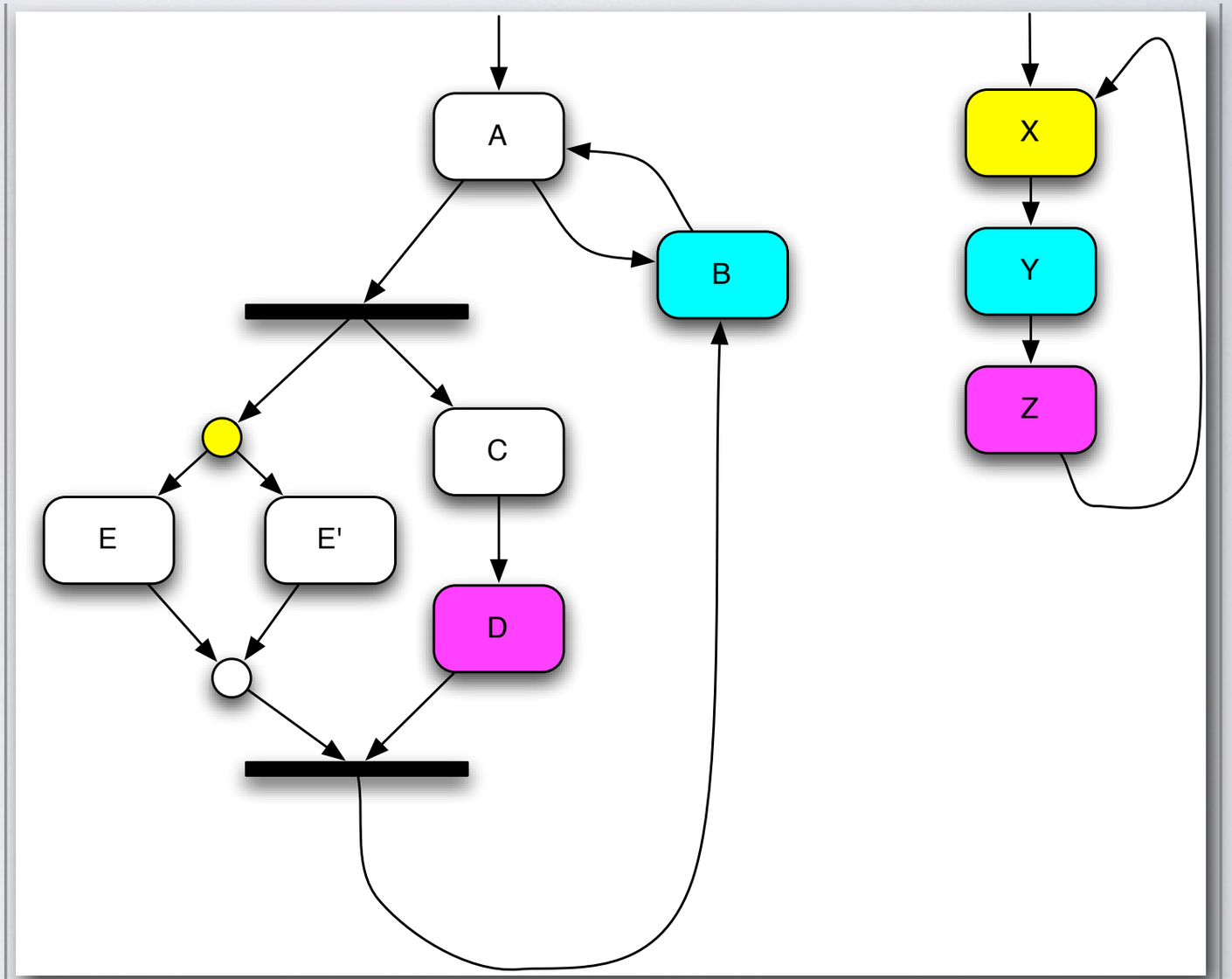
- „Klassische“ Modellierung als Zustandsautomaten

- Ressourcen-Konflikte nicht sichtbar



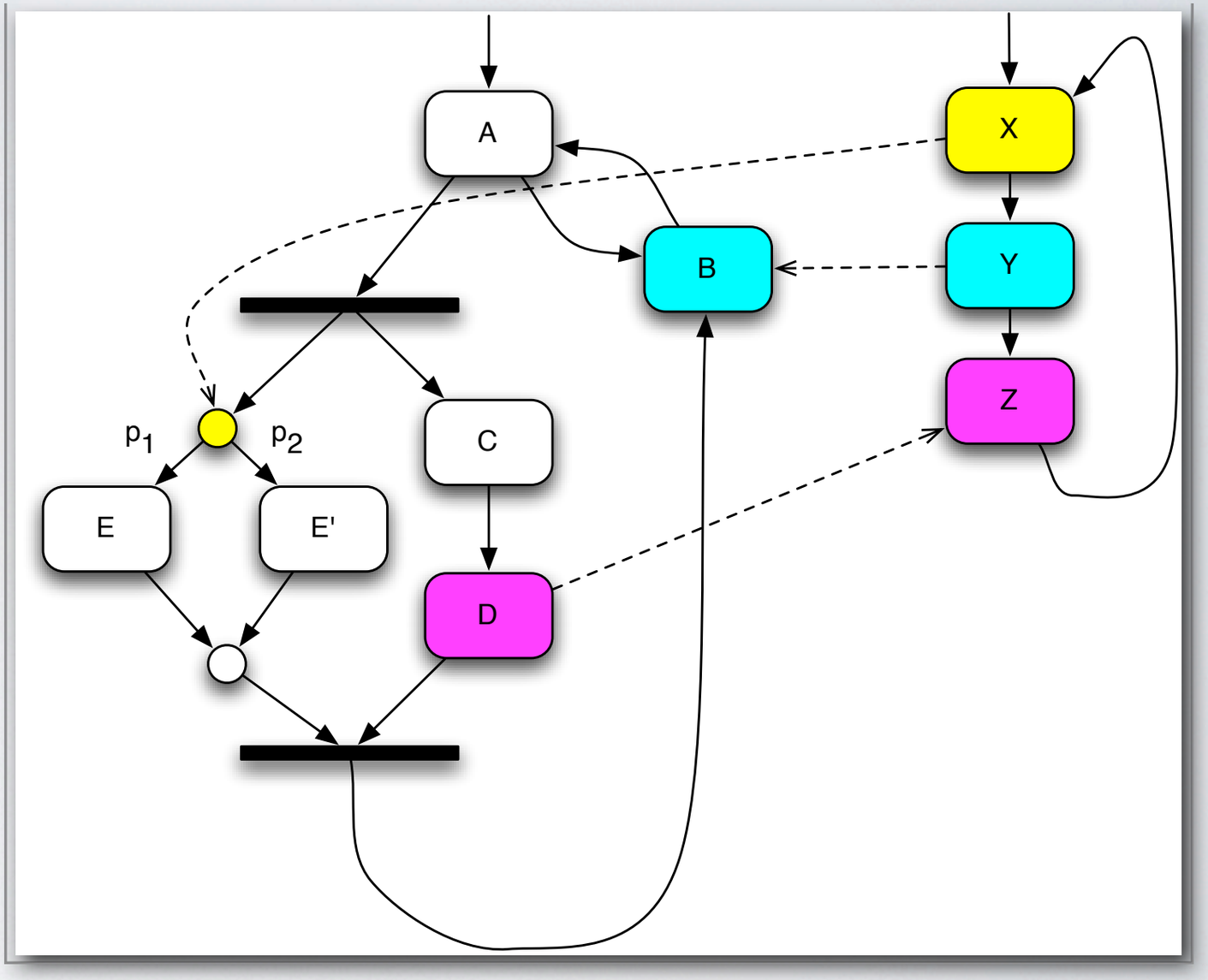
BEISPIEL (2)
DARSTELLUNG VON ABHÄNGIGKEITEN

- Abhängigkeiten von Ressourcen in B, pre E, D, X, Y, Z
- Auflösung manuell im Code der einzelnen Zustände
- Fehleranfällig



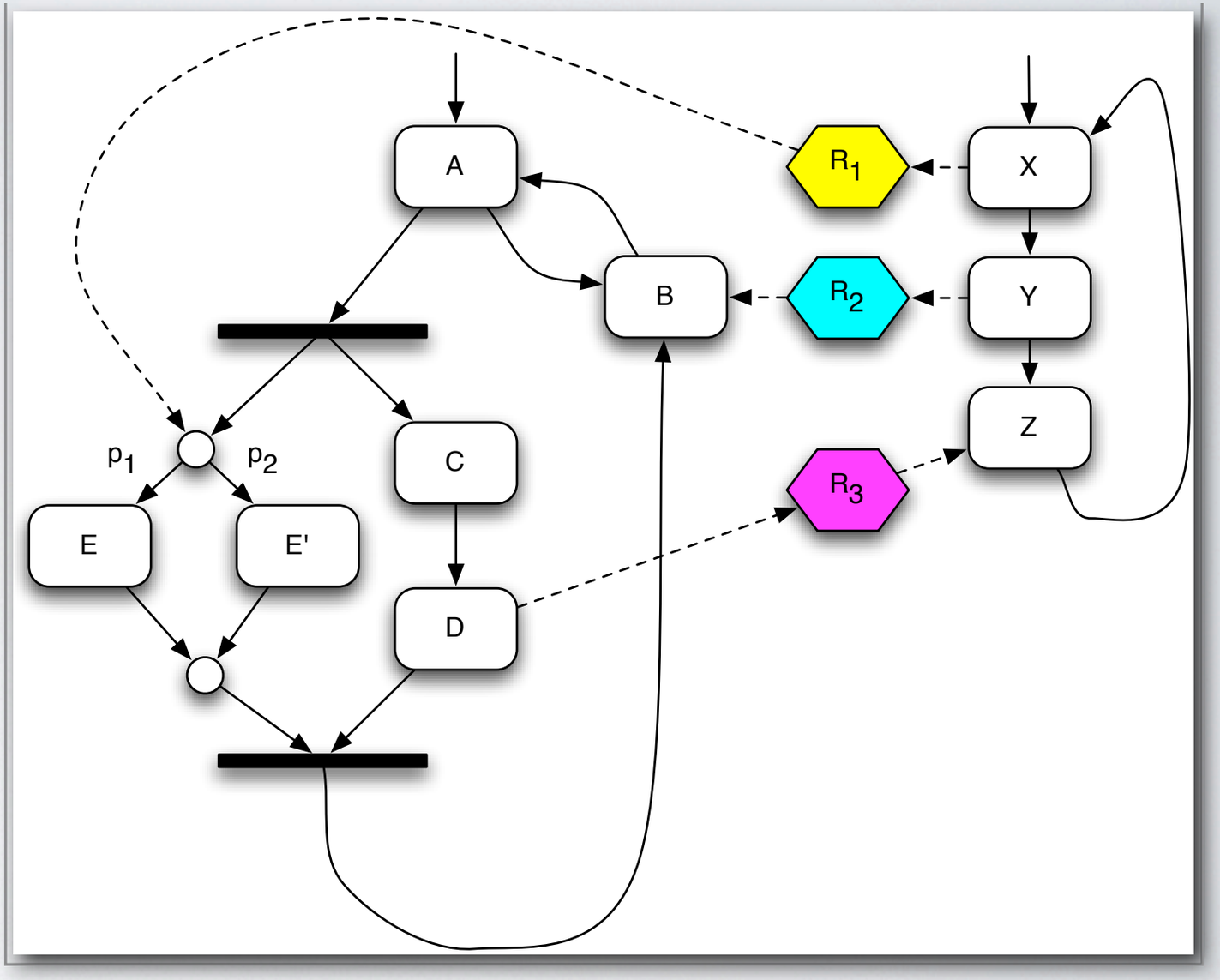
BEISPIEL (3)
DARSTELLUNG VON ABHÄNGIGKEITEN

- Abhängigkeiten haben „Richtung“
  - out = produzierend = write
  - in = konsumierend = read

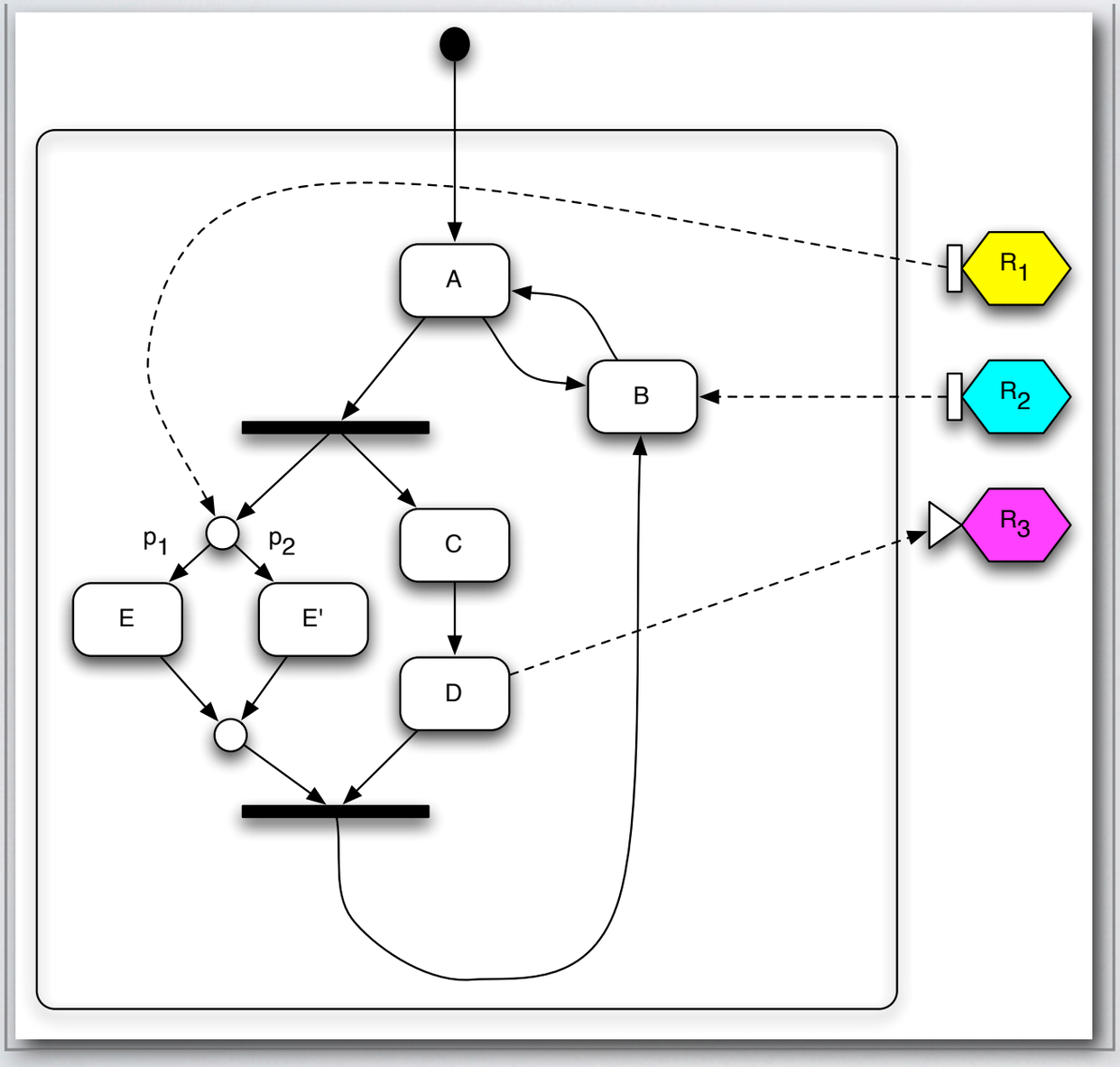


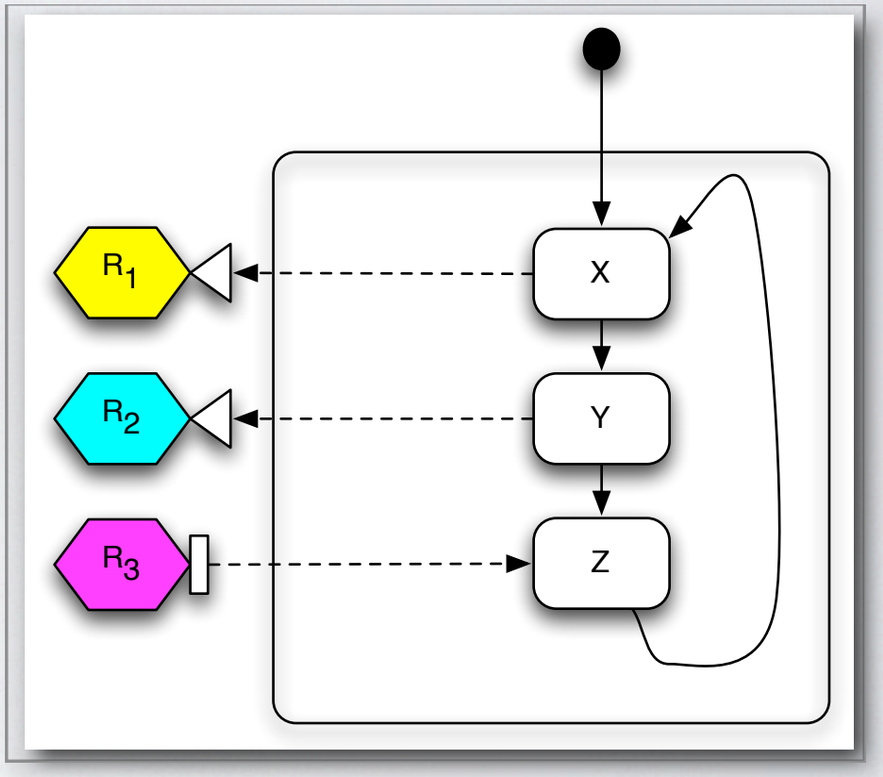
BEISPIEL (4)
DARSTELLUNG VON ABHÄNGIGKEITEN

- Explizite Ressourcen mit gerichteten Abhängigkeiten
- Modellierung
- Extraktion aus Source Codes mittels statischer Analyse



BEISPIEL (5) KOMPONENTENSICHT





GEWONNENER MEHRWERT?

- Explizit gemachte Ressourcen-Abhängigkeiten erlauben
  - Statische Erkennung von Konflikten
  - Synthese von Synchronisations-Code
    - Franz Puntigam: Synchronisation auf Token-Basis
- Parallelisierung / Out-of-order Execution von unabhängigen Teilautomaten
- Verbesserte Zusammensetzbarkeit von modularen Behaviors.