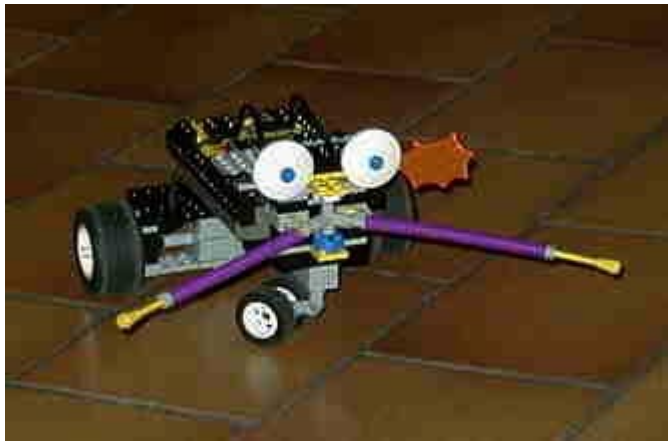
Ursprung: Roboterwettbewerb

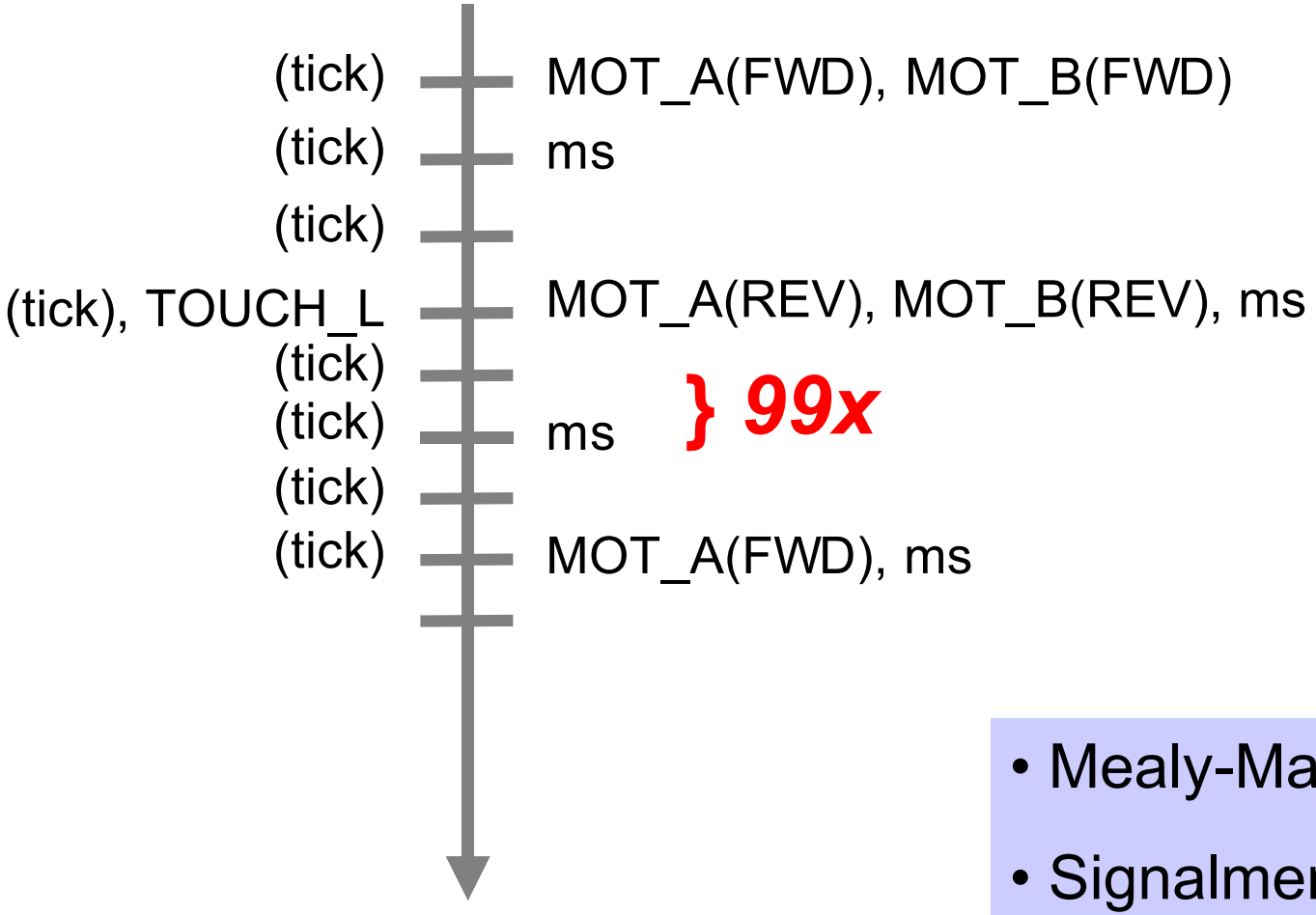


- Zyklische Reaktionen auf Eingaben
- Gleichbehandlung von Zeit und Sensorwerten
- Ereigniszähler
- Feingranulare Parallelität

```
input  TOUCH_L, TOUCH_R;
output MOT_A(integer), MOT_B(integer),
constant FWD, REV : integer;
var t : integer;

signal ms in
 loop
  emit MOT_A(FWD); emit MOT_B(FWD);
  await [TOUCH_L or TOUCH_R];
  present TOUCH_L then t:=1; else t:=2;  end present;
  emit MOT_A(REV);  emit MOT_B(REV);
  await 100 ms
  if t=1 then emit MOT_A(FWD);
         else emit MOT_B(FWD);
  end if;
  await 100 ms
end loop
||
loop await 2 tick; emit ms; end loop
end signal;
```

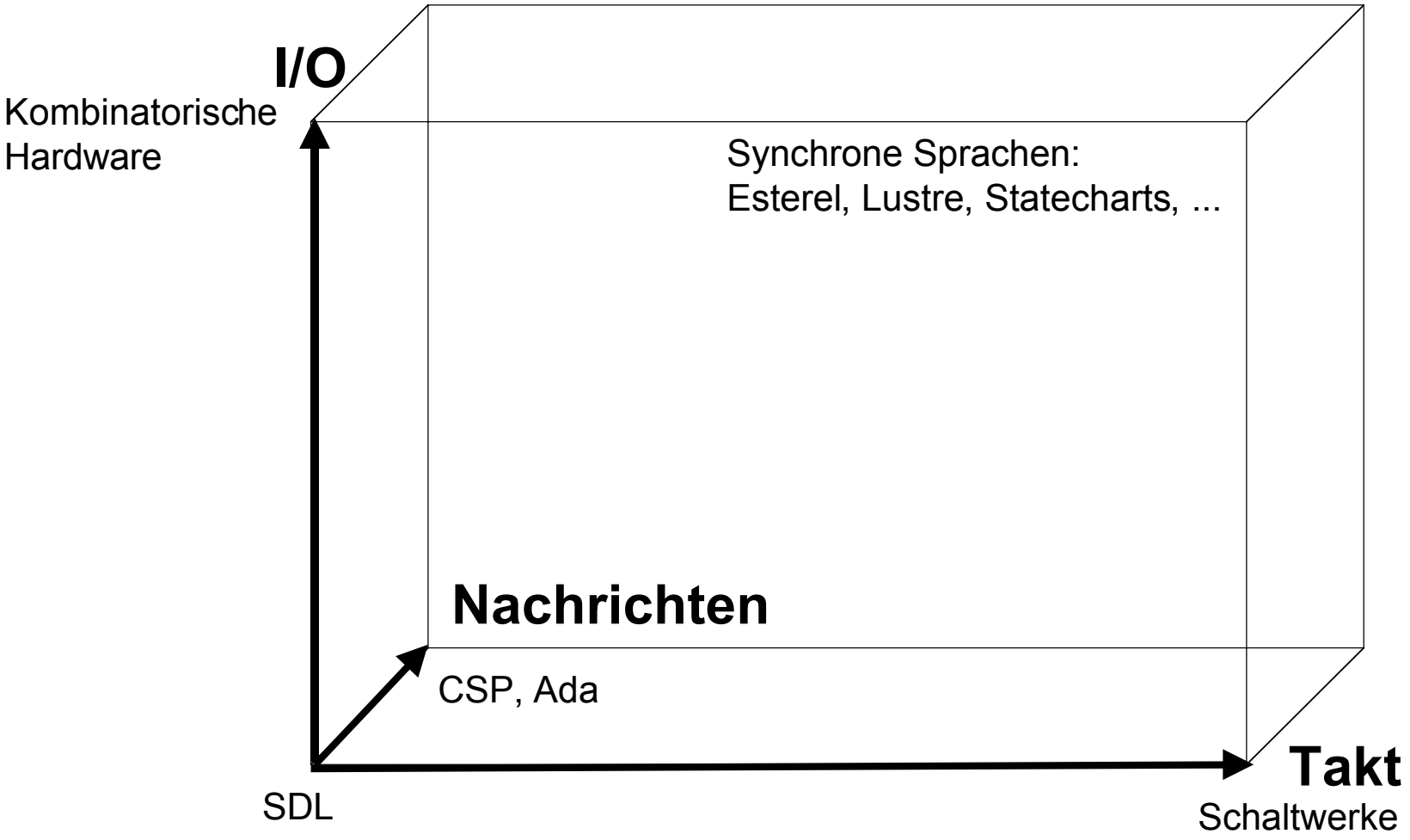
Reaktive Systeme



- Mealy-Maschinen
- Signalmengen als Alphabet

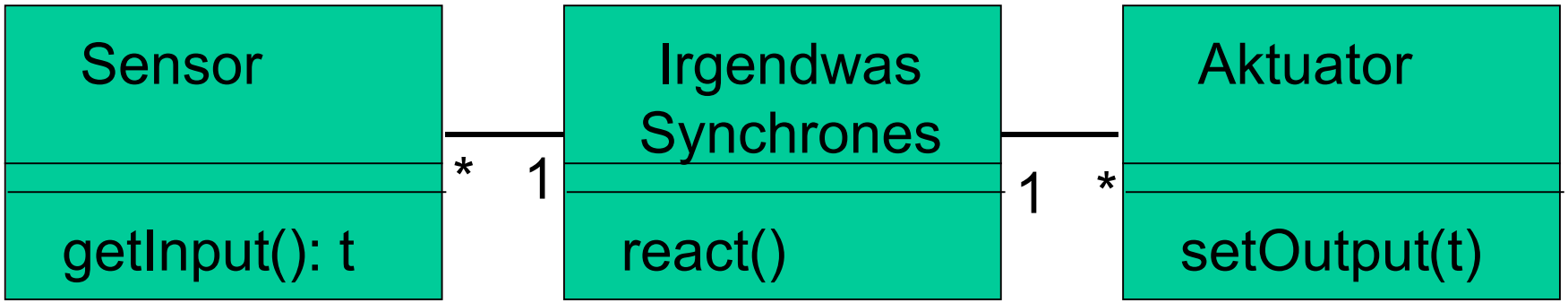
Synchron?

Synchron?


I/O
Kombinatorische
Hardware
Synchrone Sprachen:
Esterel, Lustre, Statecharts, ...
Nachrichten
CSP, Ada
SDL
Takt
Schaltwerke

Programmiersprache?

- Einbettung in Host-Sprache für
  - Datentypen, Konstante, Funktionen
  - Lesen von Eingaben und Schreiben von Ausgaben

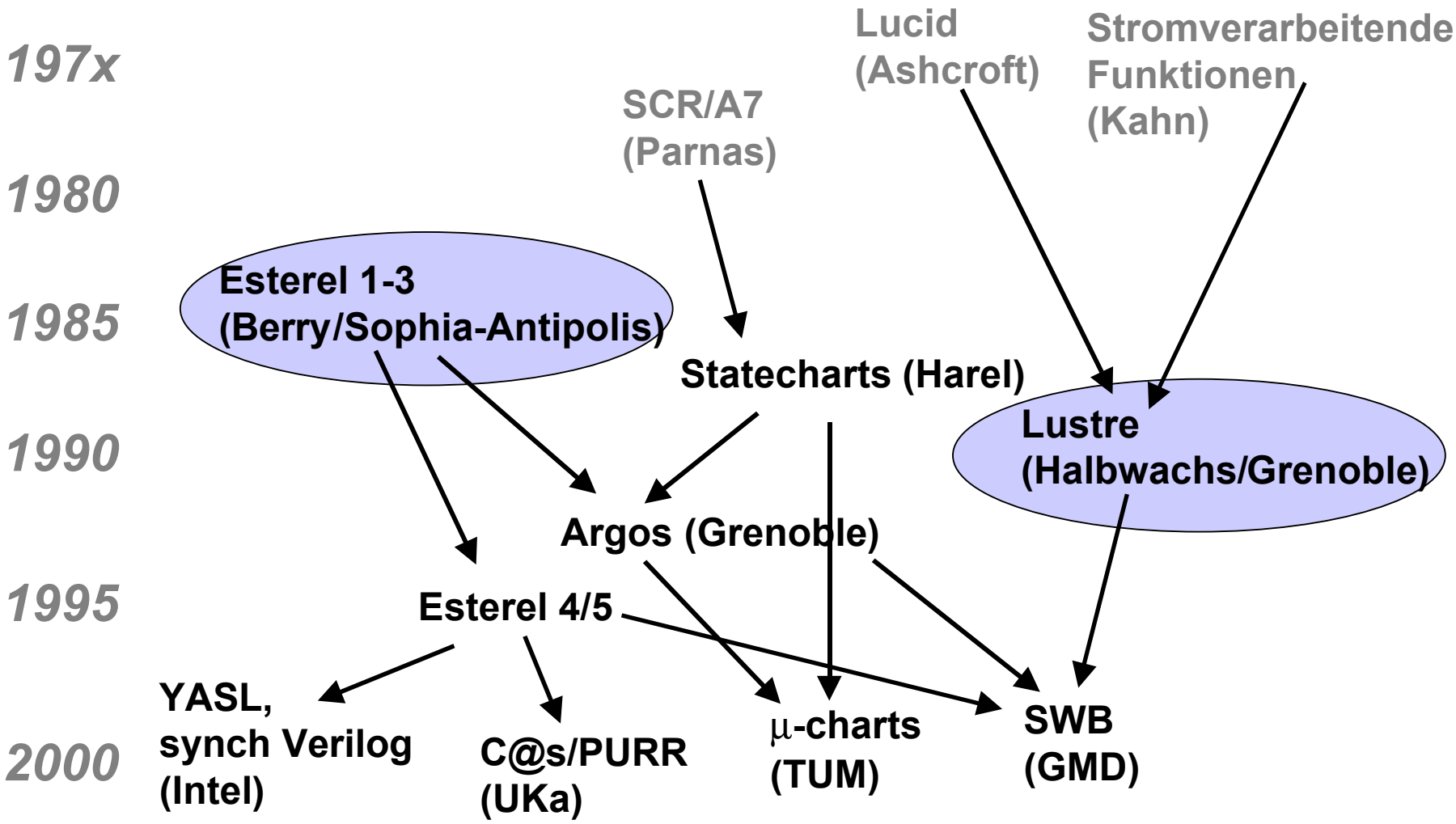


Worum geht es also?

- Übersetzer: Mealy-Maschinen-Darstellung ➔ C, Ada, Java, ...
- Präzise Semantik
- Kurze Reaktionszeiten
- Geringe Code-Größe, statische Speicherbelegung
- Parallelität führt nicht zu Nichtdeterminismus

- Geschichte
- Kontrollfluß: Esterel
- Datenfluß: Lustre
- Verifikation
- Anwendungen
- Steinheil

Historie


197x
1980
1985
1990
1995
2000
Lucid
(Ashcroft)
Stromverarbeitende
Funktionen
(Kahn)
SCR/A7
(Parnas)
Esterel 1-3
(Berry/Sophia-Antipolis)
Statecharts (Harel)
Lustre
(Halbwachs/Grenoble)
Argos (Grenoble)
Esterel 4/5
YASL,
synch Verilog
(Intel)
C@s/PURR
(UKa)
μ-charts
(TUM)
SWB
(GMD)

Esterel

- Kontrollfluß
  - if-then-else, Sequenz, loop-end
- Ein-/Ausgabe
  - emit S
  - present S then ... else ... End
- Zustände
  - pause
  - halt = loop pause end
- Unterbrechungen
  - abort ... when S
  - abort ... when immediate S
  - await S = abort halt when S
- Zähler für Signale
  - await 2 S = await S; await S
- Halbherzige Unterstützung klassischer imperativer Konstrukte
  - await S; X := S?; X := X + 1; ...

```
loop
  abort
    await 100 ROUNDS; emit DANGER
  when 1000 MSEC
end
```

```
await click;
abort
      [ await 5 tick;
        emit single;
      ||
        await click;
        emit double;
      ]
when [single or double];
```

Problem: Eindeutigkeit & Kausalität

```
present X
    then emit X
    else nothing;
end
```

```
present X
    then nothing;
    else emit X
end
```

```
present Z then
    present X then emit Y else nothing end
else
    present Y then emit X else nothing end
end
```

- In jedem Schritt müssen sich die Signale kausal ordnen lassen
- Compiler führt umfangreiche statische & dynamische Analysen durch
- Querbezüge zur Logikprogrammierung
- Kein vernünftiges Modulkonzept

Code-Erzeugung

- Interpretation von Esterel als reguläre Ausdrücke für MMs
- Generierung von Maschinen durch „Ableitung“ von Programmen nach Eingabesignalmengen (Brzozowski, 1964)

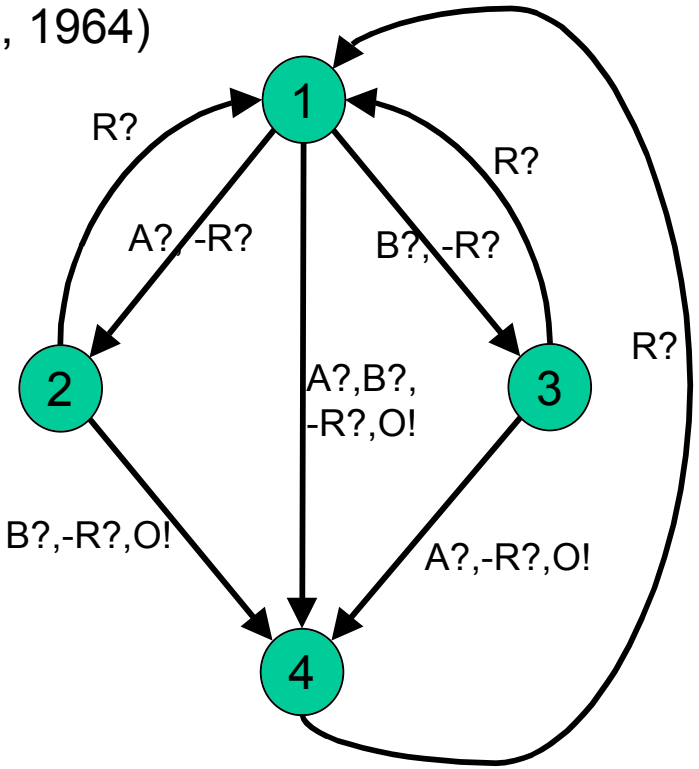
```
input A,B,R;  output O;
loop
    [ await A || await B ]
    emit O;
each R;
```

„1“: loop ... each R

„2“: await [B or R];
present R then „1“ else emit O; „4“ end;

„3“: ...

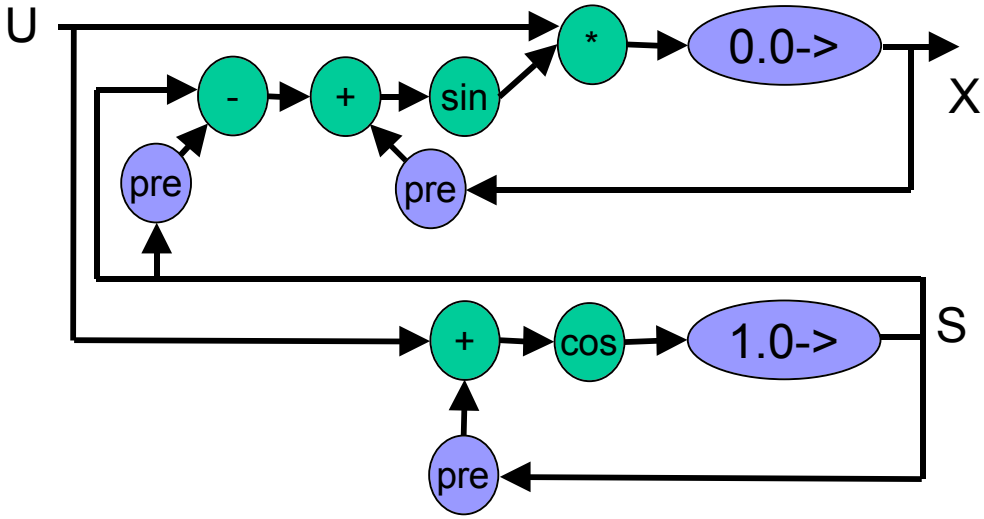
„4“: await R; „1“



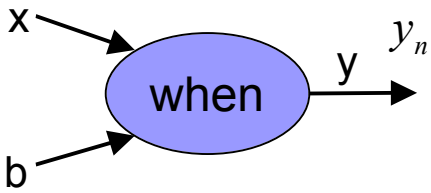
- Parallelität verschwindet
- Übersetzung nicht kompositional
- Ab Esterel 4: Übersetzung in (selbsttaktende) Schaltungen

Lustre

$$X_0 = 0 \qquad S_0 = 1$$
$$X_{t+1} = U_{t+1} \leftarrow \sin(X_t + S_{t+1} - S_t)$$
$$S_{t+1} = \cos(S_t + U_{t+1})$$



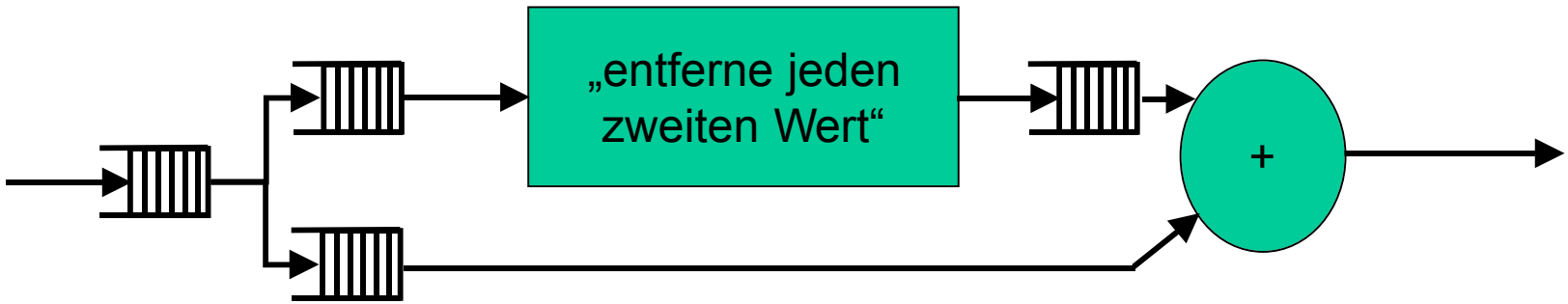
- Datenflußsprache für Signalverarbeitung
- Atomare Knoten
  - Register „pre“
  - Initialisierung „init -> rest“
  - Arithmetik
- Zyklen im Abhängigkeitsgraph müssen „pre“ enthalten
- Nicht jeder Strom muß stets einen Wert enthalten:



$$y_n = \begin{cases} x_n & \text{falls } b_n = \text{tt} \\ - & \text{falls } b_n = \text{ff} \end{cases}$$

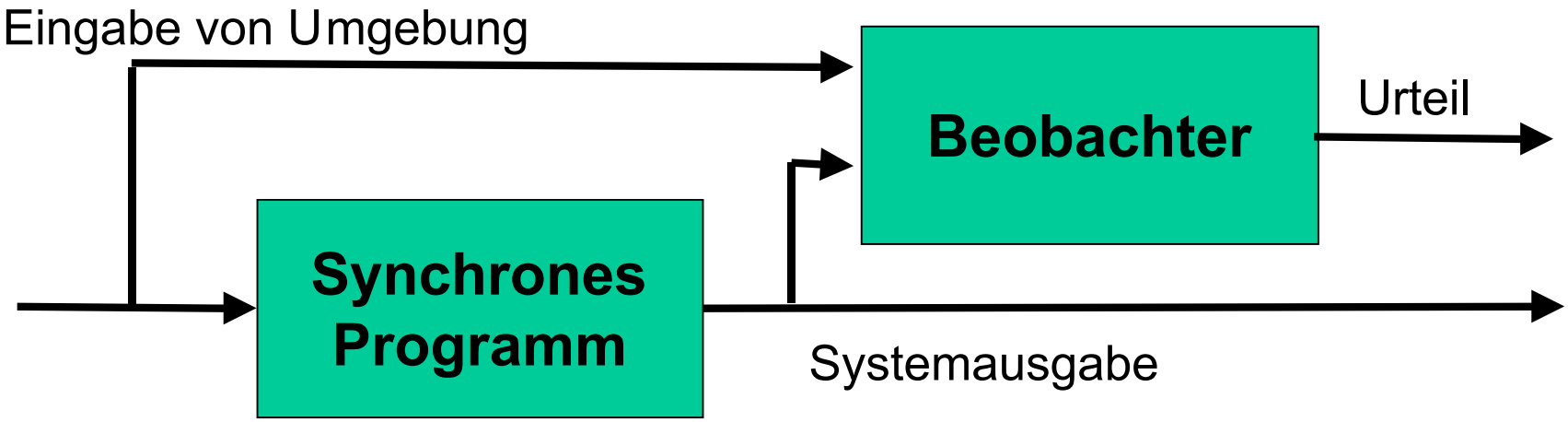
```
node Control(U: float) returns (X: float)
var S: float;
let
  X = 0.0 -> (U*sin(pre(X) + S - pre(S);
  S = 1.0 -> cos(pre(S) + U);
tel
```

Problem: Speicherbegrenzung



- Lösung: Takt-Hierarchie
  - Jeder Strom hat einen Teil-Takt des Systemtakts
  - Ein- und Ausgänge arithmetischer Knoten haben gleichen Takt
  - Heruntertakten: „x when b“
  - Herauftakten: „current x“
- Compiler prüft Kompatibilität der Takte
- Separate Übersetzung schwierig

Verifikation mit Beobachtern



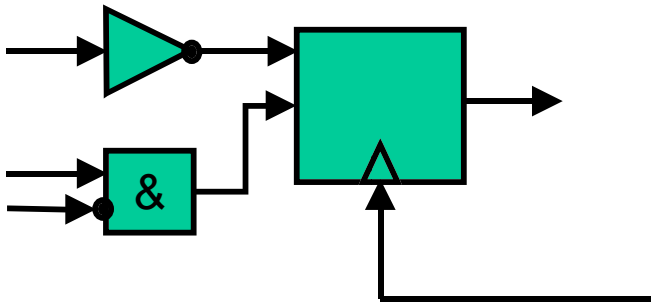
- Auch der Beobachter ist synchrones Programm
- Beobachter gibt bei unzulässiger I/O-Kombination Signal ERROR aus
- Gegenbeispiele direkt zum Tracen des Objektcodes verwendbar
- Assumption/Guarantee durch vorgeschaltete Umgebungskomponente

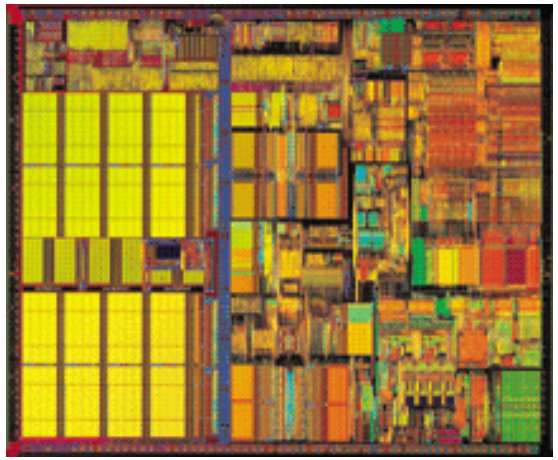
Industrielle Anwendungen

- **Dassault**: Esterel für Landungs- & Wartungs-Computer, Display-Management
- **Airbus:** Lustre für Fahrwerksteuerung?
- **Synopsys:** Esterel?
- **Schneider Electric:** Lustre für Sicherheitsfunktionen und Energiemanagement von KKWs
- **Aerospatiale, Eurocopter:** Lustre/SCADE

Hardware-Design

```
ARCHITECTURE blabla IS
BEGIN
  write_data_alg: PROCESS
  BEGIN
    WAIT UNTIL adr_request = '1';
    WAIT FOR 10 ns;
    bus_adr <= int_adr;
  END PROCESS write_data_alg;
END blabla;
```





Sonstige Arbeiten

- Uni Karlsruhe / Detlef Schmid:
  - Esterel-Erweiterung um Nichtdeterminismus
  - Starke Ausrichtung auf Verifikation
- GMD Bonn:
  - Synchronie Workbench
  - Integration von Esterel, Lustre, Argos (Statecharts)
- Intel, Cadence, Synopsys
  - Experimentelle Hardwarebeschreibungssprachen
- U Berkeley
  - Ptolemy: Integration verschiedener Sprachen zur Modellierung & Simulation

- ...und etliche Arbeiten zu Statecharts-Formalisierungen

Resümee

- Kernidee: Sprachen zur Beschreibung von Mealy-Maschinen mit mengenwertigem Alphabet
- Enormer Aufwand in die Compiler gesteckt (Lex für MMs)
  - Parallelität wird wegübersetzt
- Natürliche Grenzen durch
  - unzureichendes Modulkonzept
  - teilweise obskure Syntax
- Keine Kombination von Kontroll- und Datenfluß
- Verwendung heute als Backend für graphische Notationen
  - Esterel: Synchcharts (Esterel Studio, Simulog)
  - Lustre: Blockdiagramme (SCADE, CS Verilog/Telelogic)