On Automatic Code Generation for Control Applications

Karl-Erik Årzén
Dept of Automatic Control, LTH
Lund University
Sweden

LUND INSTITUTE OF TECHNOLOGY
Lund University

Outline

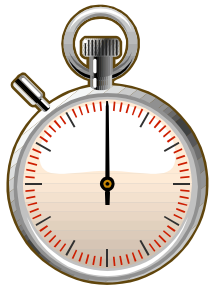
- **Background**
- Discretization
- Realization
- Execution order
- Multi-Core

Current Situation

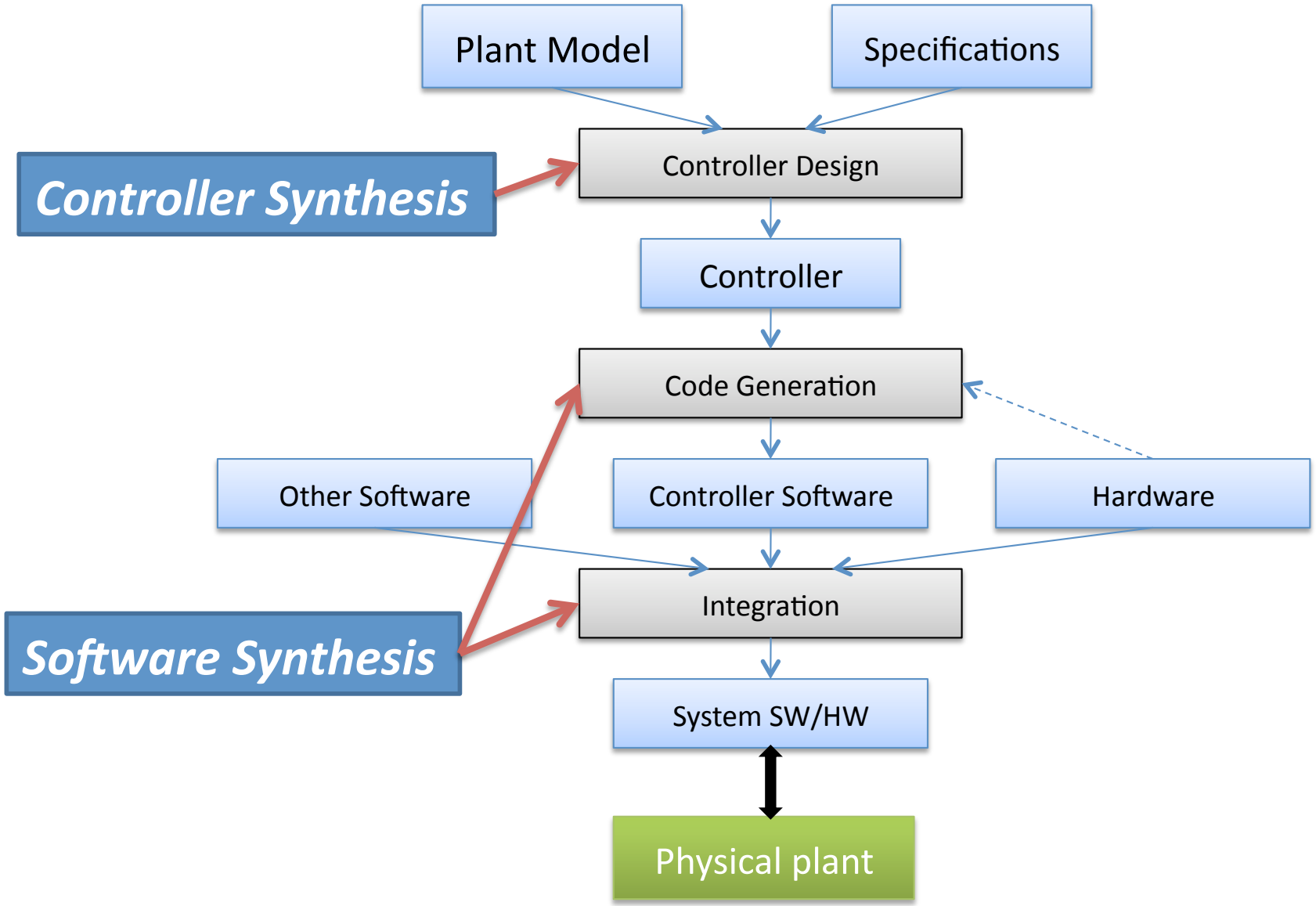
- Automatic code generation used a lot in industry for control applications
  - Automotive, aerospace, …..
- Generate, typically C, code from high-level abstractions used for control design and analysis
  - Matlab/Simulink, Scilab/Scicos, SCADE/Lustre, Esterel, …
- Growing interest for data-flow programming models (SDF, DDF, Kahn …) and associated code generation tools

Why Automatic Code Generation

- Shorten lead times
- Minimize # of software errors
  - "Correct-by-construction"
- Save money

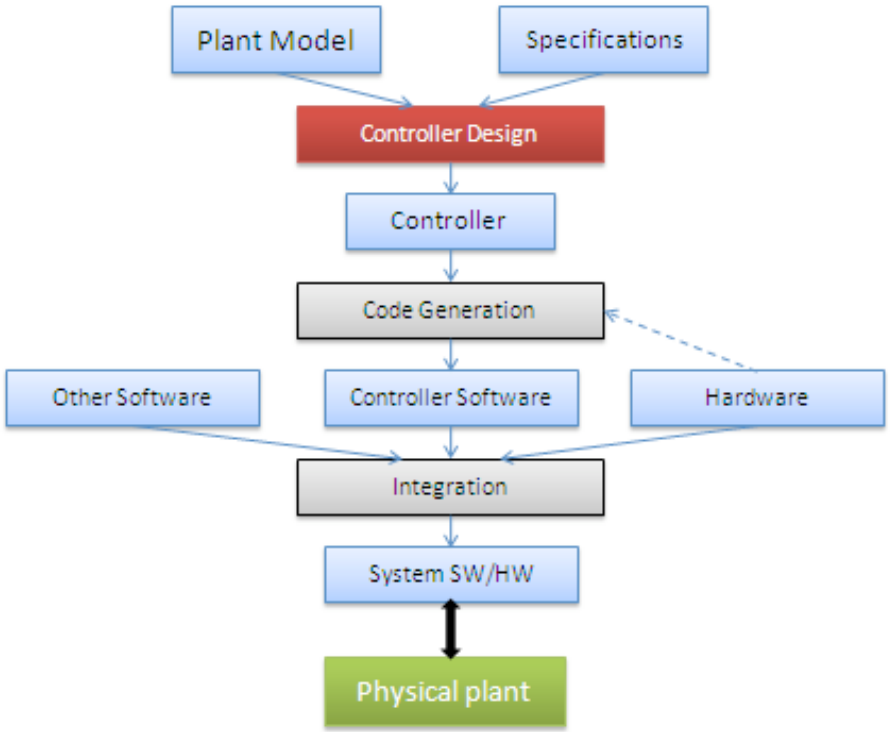


Control Design Flow



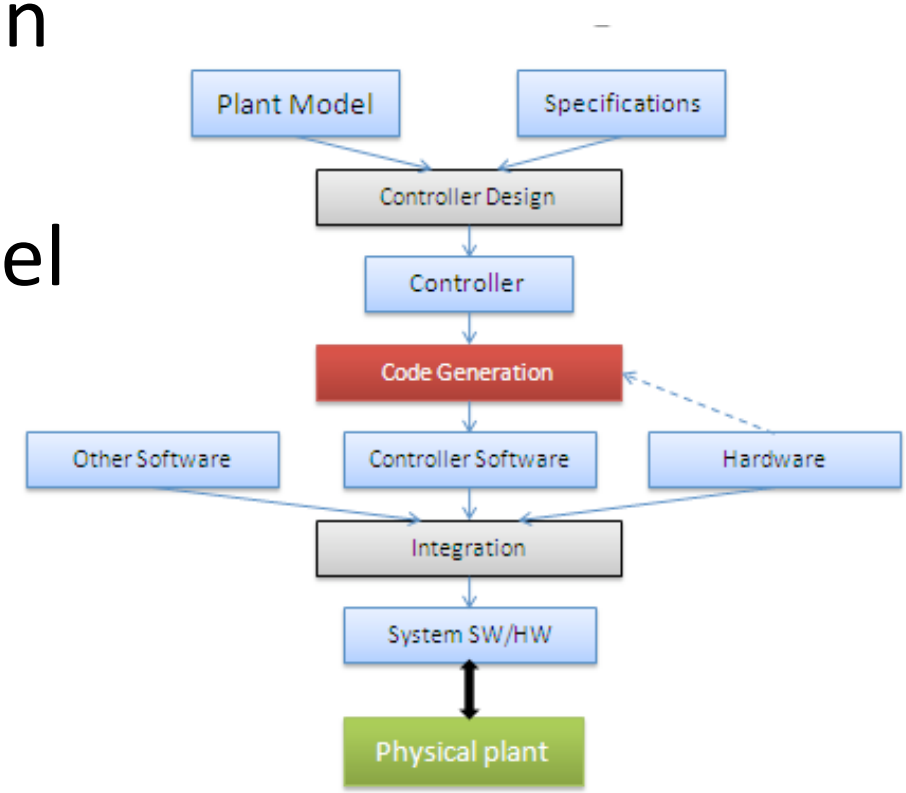
Controller Design

- Different control design methods
  - Model-based or empirical
  - More or less formal
  - More or less automated
- Verification by simulation in e.g. Matlab/Simulink



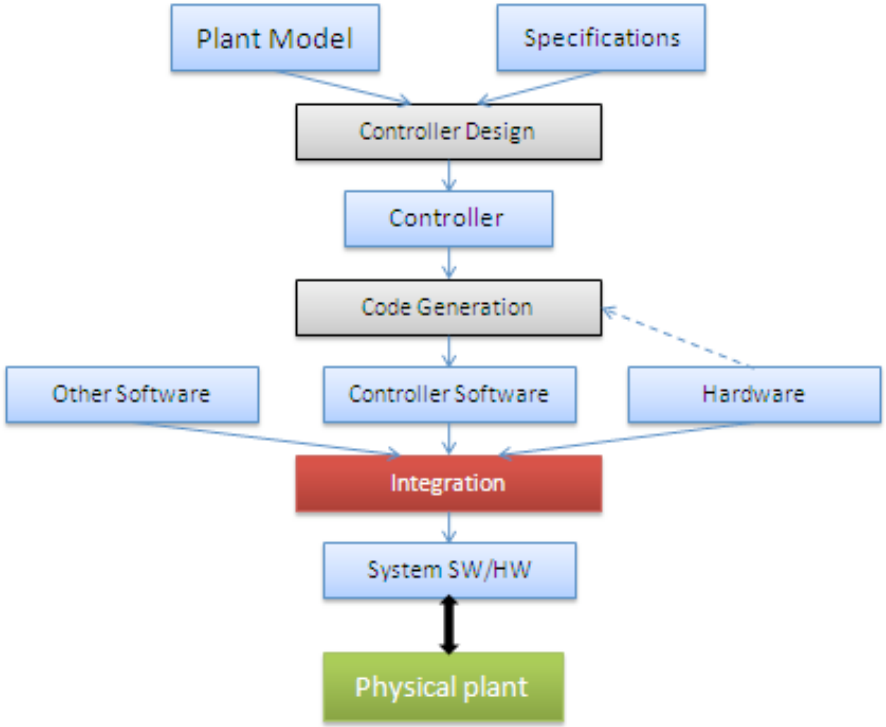
Code Generation

- Automatic code generation tools
- Threads as the main model of computation
- Single-threaded or multi-threaded
- Verification by simulation against hi-fidelity model, HW in the loop techniques, etc

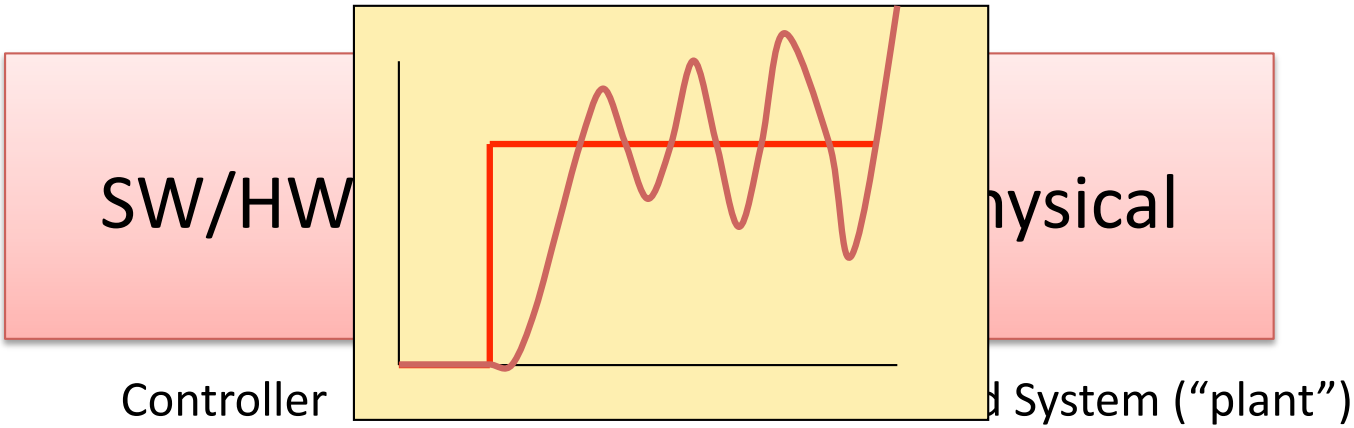


Integration

- Shared computing and communication resources in the implementation platform cause non-determinism
- Schedulability-based verification increasingly difficult
- Limited tool support



Problem with Feedback



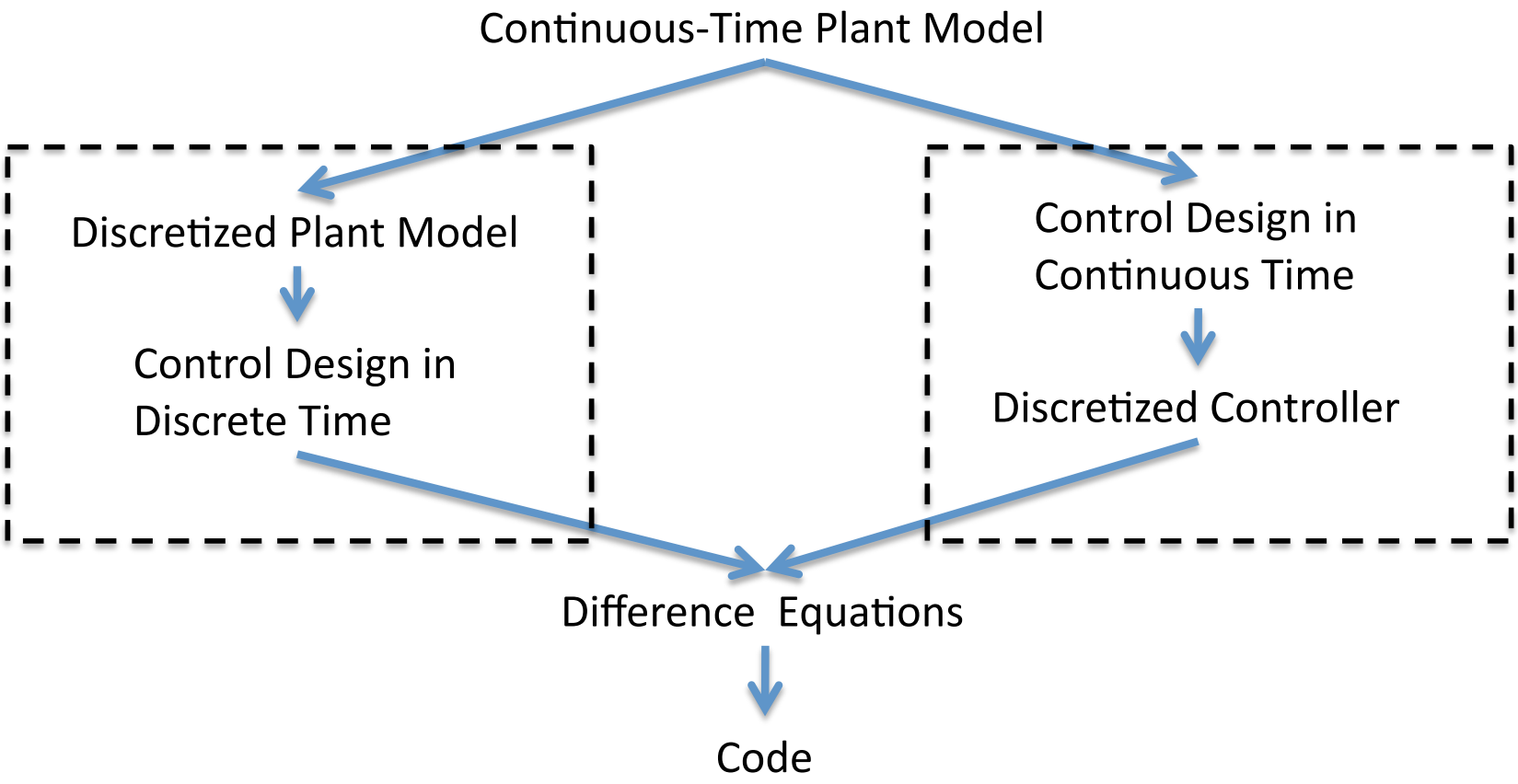
- Errors can be difficult to isolate
- Similar symptoms can be the cause of errors either in the **controller design, the code generation**, or the **software integration** + errors in the **information transfer** between the phases
-

Outline

- Background
- **Discretization**
- Realization
- Execution order
- Multi-Core

Discrete or Continuous Time?

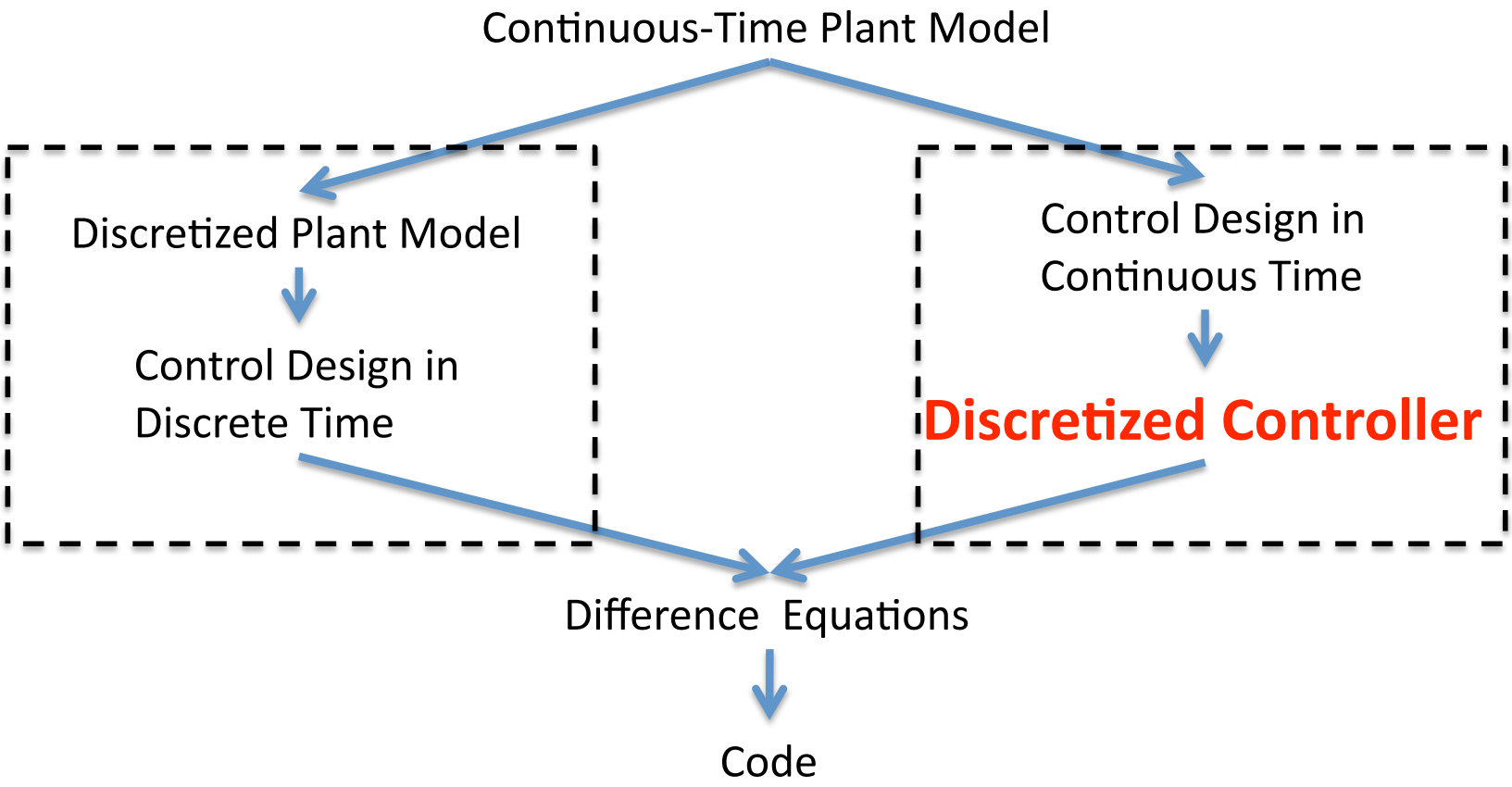
- When designing a controller you often have the choice to either perform the design in the continuous-time framework or in the discrete-time framework



Discrete or Continuous Time?

- Discrete-Time when
  - Plant model on discrete-time form
  - Control design method assumes a discrete-time controller, e.g. MPC
  - Fast sampling not possible
- Continuous-Time when
  - Empirical control design
  - Nonlinear continuous-time model
- However, in most cases the choice does not matter from a control design point of view

Continuous-Time Approximations



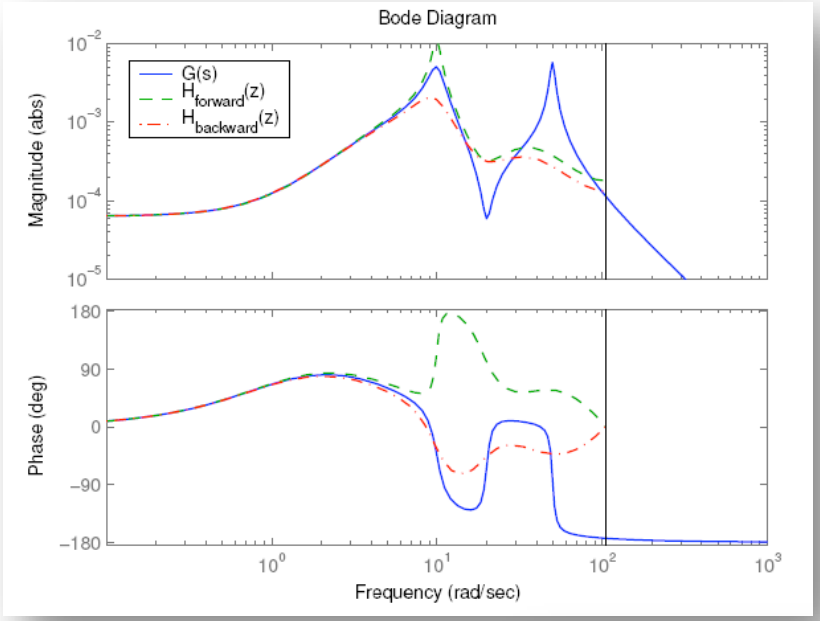
- A lot of alternative methods
- No clear consensus about which method to use

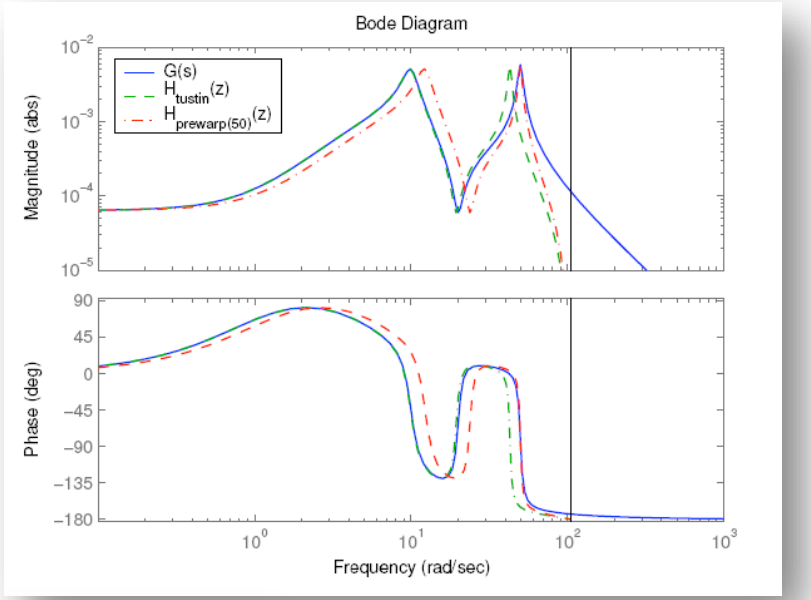
Approximation Alternatives

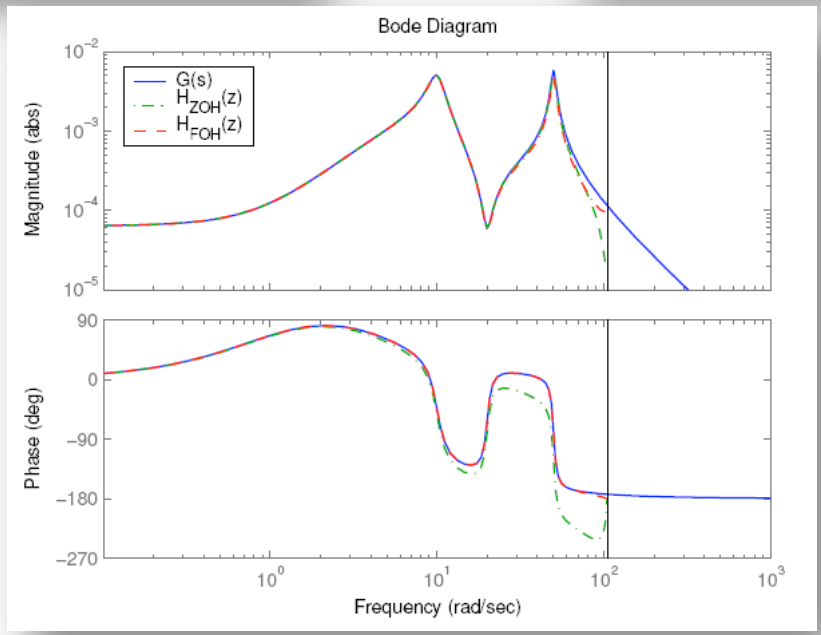
- Forward Euler
- Backward Euler
- Tustin w/wo frequency prewarping
- Step invariance (ZOH-sampling)
- Ramp invariance (FOH-sampling)
- Pole-zero matching
- …..

➧ May get quite different results

Example


Bode Diagram
G(s)
H_forward(z)
H_backward(z)
Magnitude (abs)
Phase (deg)
Frequency (rad/sec)


Bode Diagram
G(s)
H_tustin(z)
H_prewarp(50)(z)
Magnitude (abs)
Phase (deg)
Frequency (rad/sec)


Bode Diagram
G(s)
H_ZOH(z)
H_FOH(z)
Magnitude (abs)
Phase (deg)
Frequency (rad/sec)

Code Generation Consequences

- Several code generation tools perform approximation of continuous-time controllers automatically
- Potentially quite dangerous
- The control designer need to be aware of this and to be in full control
- In my opinion it is the responsibility of the control designer to perform the discretization

Outline

- Background
- Discretization
- **Realization**
- Execution order
- Multi-Core

Transfer Functions

- Several control design methods assume input-output models
- Resulting controller is a transfer function
  - Laplace transfer function
  - Z transfer function

$$H(z) = \frac{b_0 + b_1 z^{-1} + \ldots + b_n z^{-n}}{1 + a_1 z^{-1} + \ldots + a_n z^{-n}}$$

- Can be realized in a number of different ways with the same input-output behavior
  - Assuming infinite precision arithmetic
  - **Not the case for fixed point arithmetic**

Realization Forms

- ~~Direct form~~

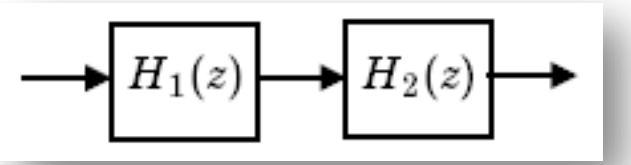
$$u(k) = \sum_{i=0}^{n} b_i y(k-i) - \sum_{i=1}^{n} a_i u(k-i)$$

- ~~Companion forms~~
  - Controllable
  - Observable

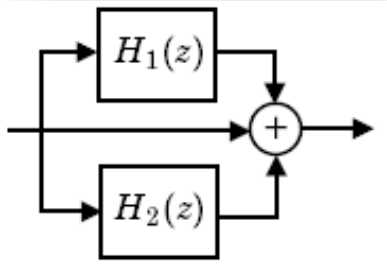
$$x(k+1) = \begin{pmatrix} -a_1 & -a_2 & \cdots & -a_{n-1} & -a_n \\ 1 & 0 & & 0 & 0 \\ 0 & 1 & & 0 & 0 \\ \vdots & & & & \\ 0 & 0 & & 1 & 0 \end{pmatrix} x(k) + \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} y(k)$$

$$u(k) = \begin{pmatrix} b_1 & b_2 & \cdots & b_n \end{pmatrix} x(k)$$

- Series form



- Parallel form

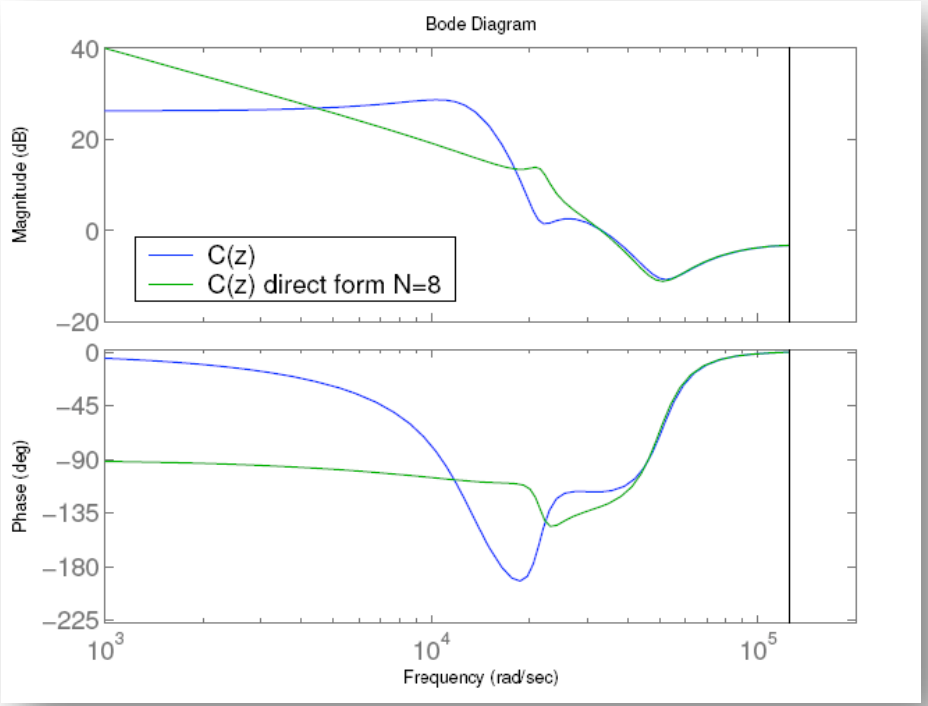


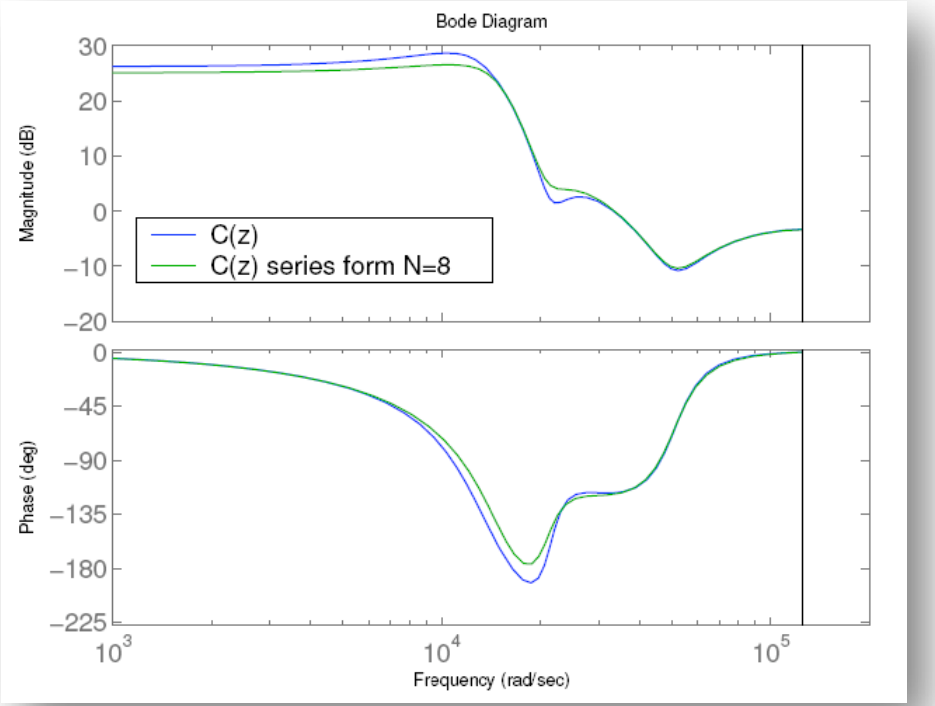
Example

$$C(z) = \frac{z^4 - 2.13z^3 + 2.351z^2 - 1.493z + 0.5776}{z^4 - 3.2z^3 + 3.997z^2 - 2.301z + 0.5184} \qquad \text{(Direct)}$$

$$= \left(\frac{z^2 - 1.635z + 0.9025}{z^2 - 1.712z + 0.81}\right)\left(\frac{z^2 - 0.4944z + 0.64}{z^2 - 1.488z + 0.64}\right) \qquad \text{(Series)}$$

$$= 1 + \frac{-5.396z + 6.302}{z^2 - 1.712z + 0.81} + \frac{6.466z - 4.907}{z^2 - 1.488z + 0.64} \qquad \text{(Parallel)}$$





- Fixed point arithmetic with 8 bit word length
- Q3.4 format
- Left - direct form
- Right - series form
- Large qualitative differences

Code Generation Consequences

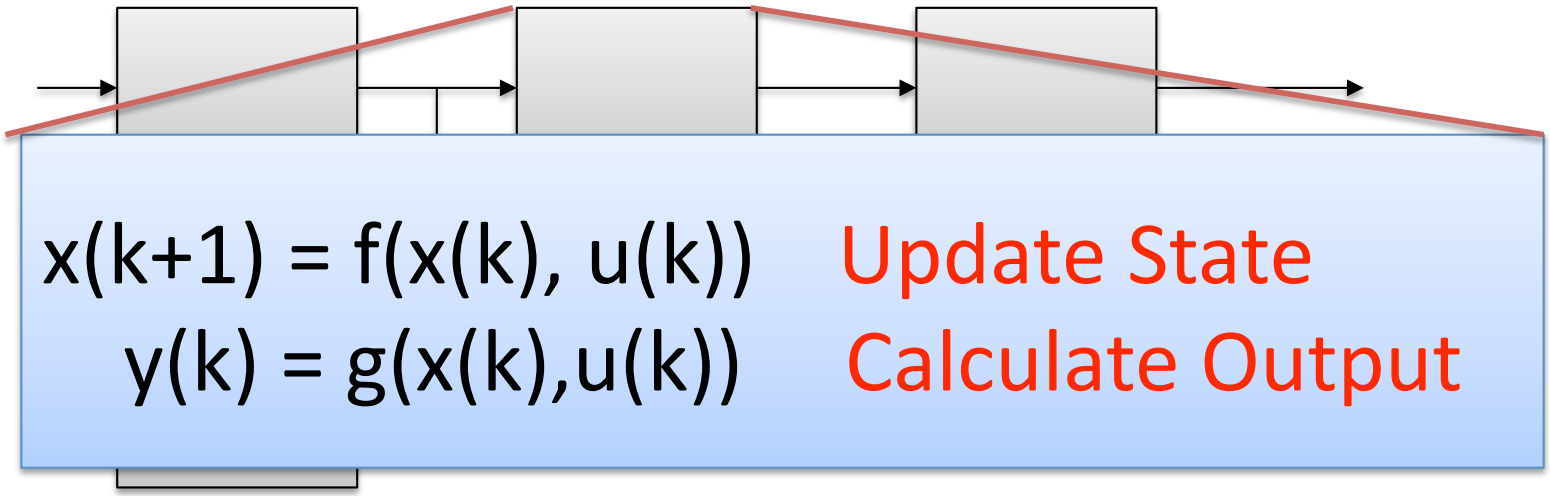
- If the code generation tool at all should be responsible for choosing realizations then the user must be able to trust that a numerically well-conditioned method is used

Outline

- Background
- Discretization
- Realization
- **Execution order**
- Multi-Core

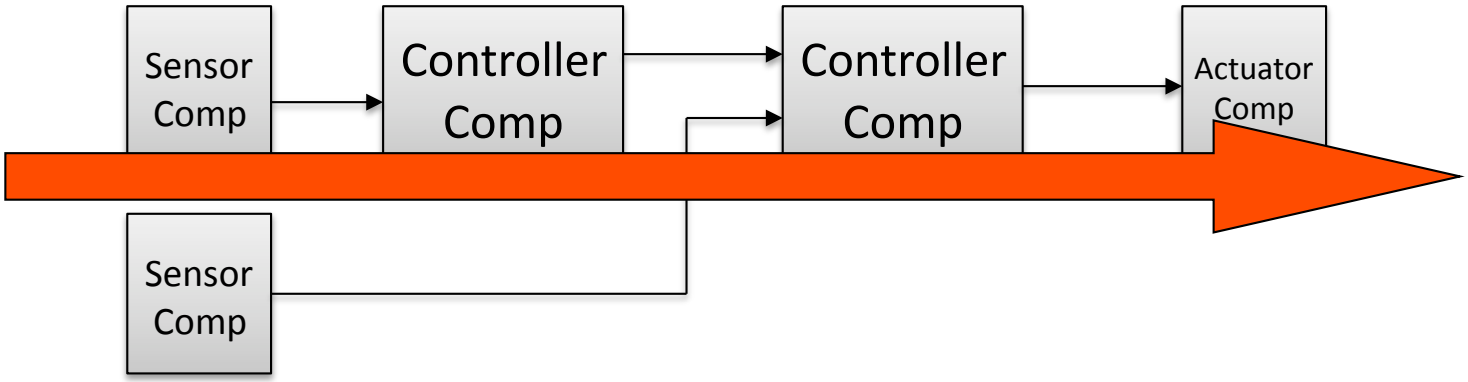
Controller Components

- Component models for embedded systems are often based on the ”pipe and filter” model



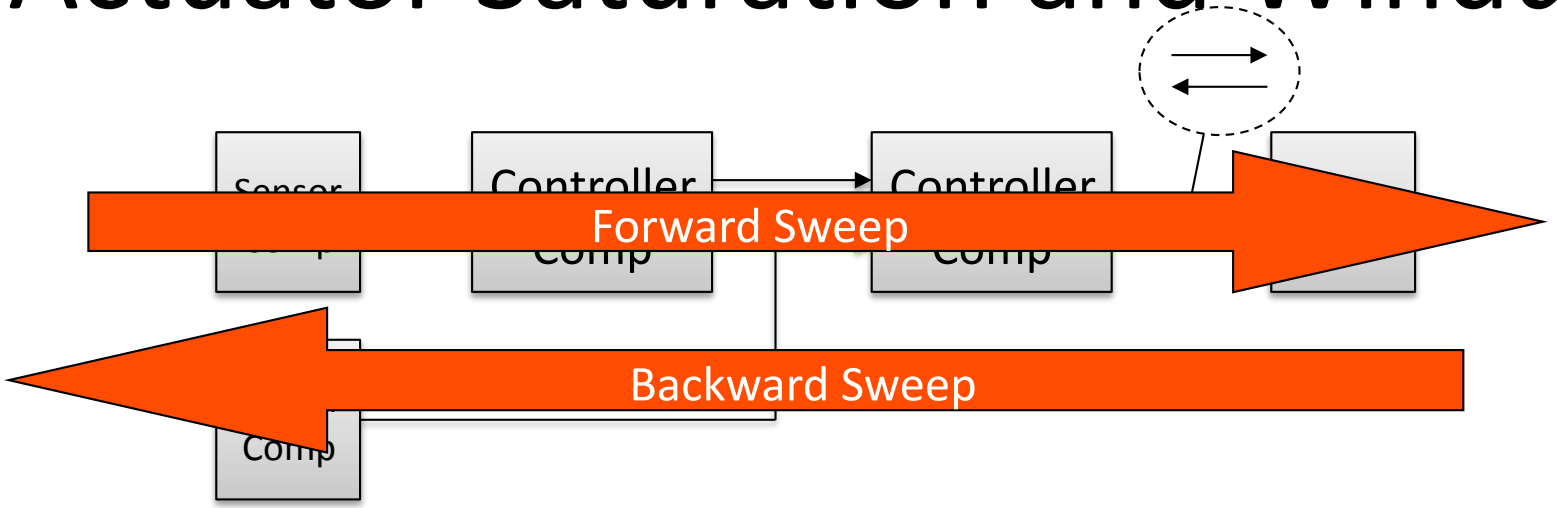
- Components (cp Simulink blocks)
- Logical signal flow
- However, not enough for controller components

Minimize Latency



- From sensor input blocks to actuator output blocks
- Solution:
  1. Execute the CalculateOutput part of all the components according to the logical signal flow
  2. Afterwards execute the UpdateState part of the components
- Multi-layered cascade control structures where common in industrial practice

Actuator Saturation and Windup



- All actuators have a limited range
- Controllers with integral action must take special precautions to avoid windup
- Information flows backwards from actuator blocks to sensor blocks
- Solution:
  1. Execute the CalculateOutput part of all the components according to the logical signal flow - Forward Sweep
  2. Afterwards execute the UpdateState part of the components in the opposite oreder - Backward Sweep

Code Generation Consequences

- Code generation tools and simulation frameworks should support this
- Most do
- However, there it is still quite difficult to convince computer science persons about how important this really is - especially people in the components community
- Well-known and supported by tools since almost 30 years in the industrial automation community
  - DCS, PLC vendors

Outline

- Background
- Discretization
- Realization
- Execution order
- **Data flow and Multi-Core**

Data-Flow Models and Multi-Core

- Large renewed interest in data-flow models for media streaming and signal processing
- Model parallelism explicitly
- Potentially a good well suited for multi and many-core platform
- However,
  - Not as easy as it may seem
  - Good tools strongly needed

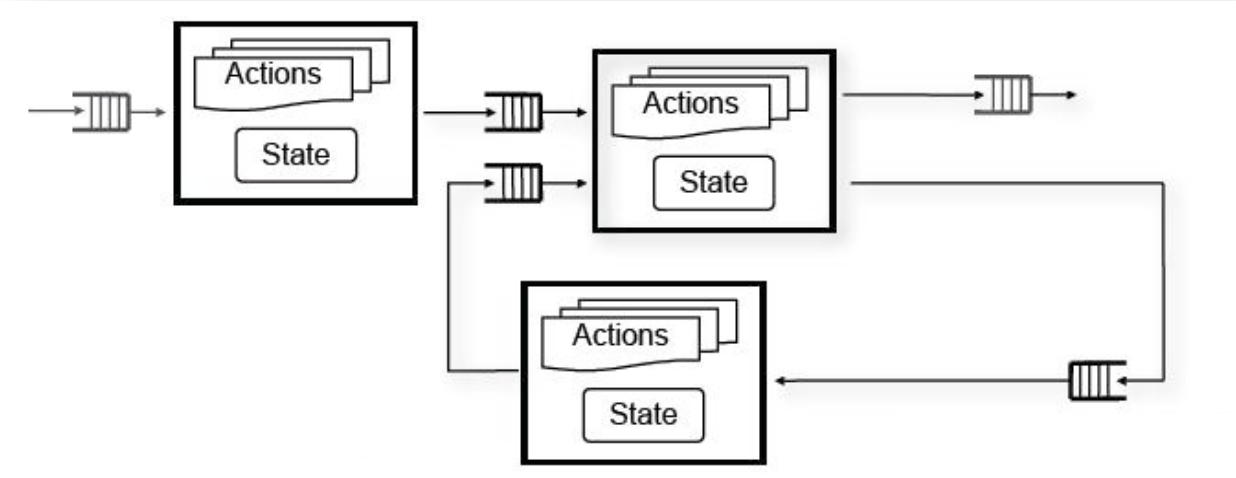
ACTORS

- ACTORS - Adaptivity and Control of Resources in Embedded Systems
  - FP7 STREP coordinated by Ericsson
  - SSSA, TUKL, EPFL, ULUND, Evidence, Akatech
  - Elements:
    - Data-flow programming
    - Adaptive resource management
    - Bandwidth reservation scheduling techniques
    - Linux
    - ARM 11 and x86 multi-core platforms

ACTORS

ACTORS: Dataflow Modeling

- Data flow programming with actors
  - Associate resources with streams
  - Clean cut between execution specifics and algorithm design
  - Strict semantics with explicit parallelism provides foundation for analysis and model transformation
- CAL Actor Language (UC Berkeley, Xilinx) http://opendf.org
  - Part of MPEG/RVC standardization
  - SDF, DDF, …..



CAL Language

```
actor PingPongMerge ()
      Input1, Input2 ==>
Output:

    s := 0;

    action Input1: [x] ==> [x]
    guard s = 0
    do
        s := 1;
    end

    action Input2: [x] ==> [x]
    guard s = 1
    do
        s := 0;
    end
end
```

Input and output ports

State

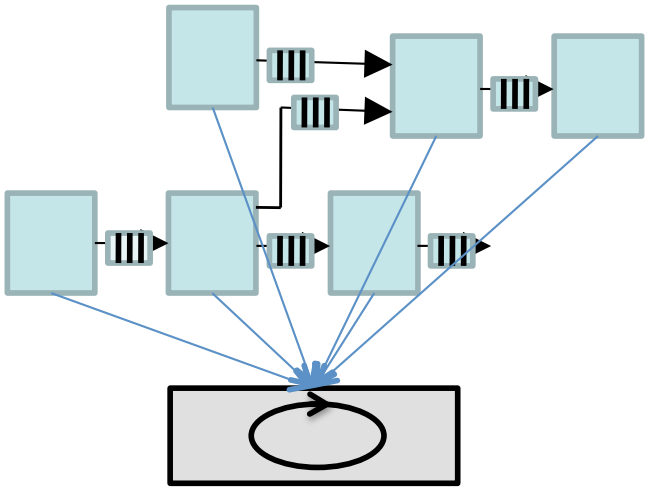
Action guards

Action body

+ priority relations among actions, FSM for represent action activations, …..

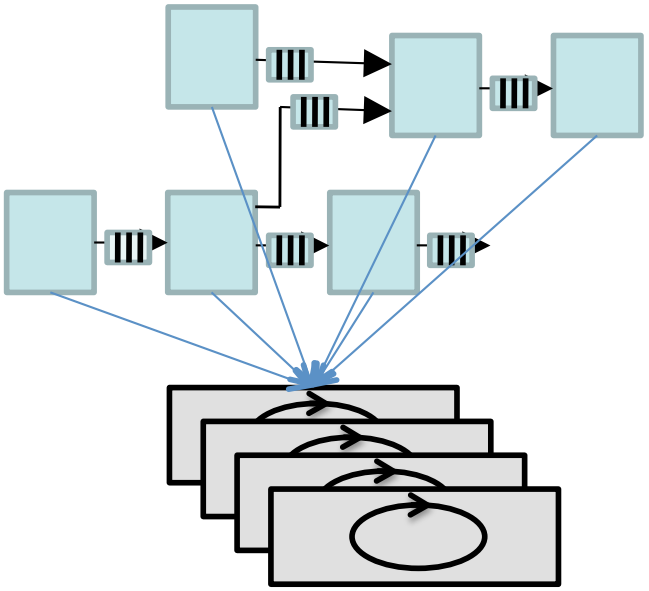
Run-Time System and Code Generation

- Code generation from actors code to C (ARM) code has been implemented
- Run-time system for CAL networks
- ARM 11 and x86 platforms
- Initially:
  - Single-threaded run-time
  - “System actors” for I/O and communication



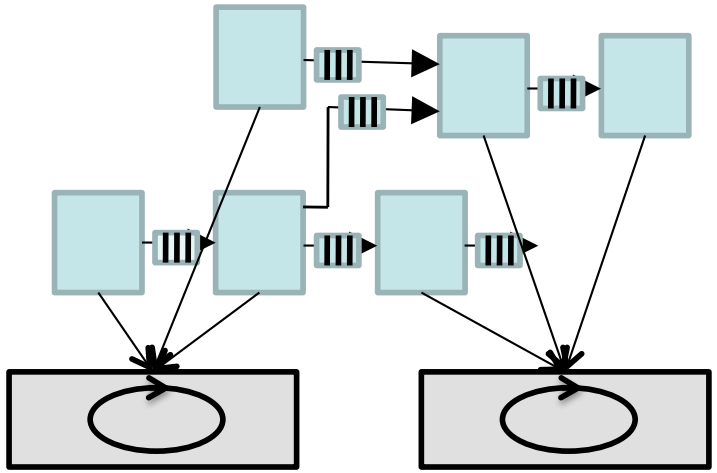
Run-Time System and Code Generation

- Second attempt:
  - Dynamically partitioned multi-threaded run-time
  - One thread per core
  - Extremely inefficient
    - Locking and synchronization
    - Cache effects from data shuffling between cores



Run-Time System and Code Generation

- Current attempt:
  - Statically partitioned multi-threaded run-time
  - One thread per core
  - Performance gain compared to single-thread
  - But extremely cautious programming
  - I would guess that there currently are a lot of programmers trying in vain to get the “promised” performance gains from multi-cores
  - Tool support necessary



Summary

- Automatic code generation for control system requires a good understanding of control
- Issues to consider includes discretization, realization, and execution ordering
- It would be useful to have the same level of code generation support for dataflow models as we today have for “Simulink”-type models
- Software synthesis tools targeting multi-core platforms