TTK4130 Modeling and Simulation

Lecture 2

Information – guidance hours

- Sign up for the guidance hours – deadline today

https://docs.google.com/forms/d/e/1FAIpQLSdU9WtDpb8a3a1YBpPRUAtwbdaSOmDSWzuV602O4TGruoyQjQ/viewform?usp=sf_link
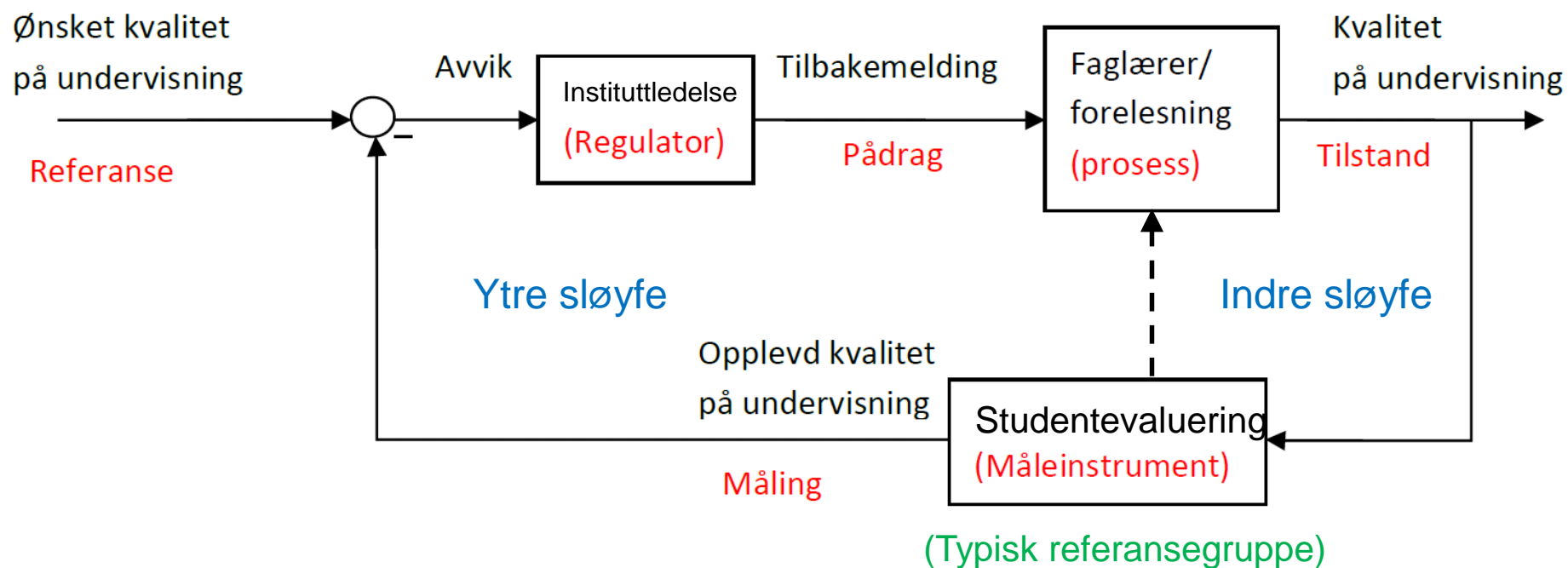
Kvalitetssikring

- Fra http://snl.no/kvalitetssikring: Kvalitetssikring, planlagte og systematiske aktiviteter som gjøres for å oppnå at et produkt eller en tjeneste vil oppfylle kravene til kvalitet

- På samme måte som industrien tilbyr produkter til sine kunder, så tilbyr NTNU emner til sine studenter

- I så måte ønsker NTNU å tilby best mulig kvalitet på sine emner til studentene:
  - Sikre at emnets læringsmål er oppdaterte og relevante
  - Sikre at læringsaktivitetene i emnet bidrar til at studentene oppnår læringsutbyttet
  - Sikre at det er sammenheng mellom læringsmålene, læringsaktivitetene og vurderingsformene

- NTNU har egne wiki-sider med mye nyttig informasjon for både studenter og ansatte: https://innsida.ntnu.no/wiki (her kan man søke seg frem til det meste)

- Direkte lenke til NTNUs sider om kvalitetssikring av utdanning: https://innsida.ntnu.no/wiki/-/wiki/Norsk/Kvalitetssikring+av+utdanning

- Hvis ingenting blir gjort kan dere melde inn via NTNUs avvikssystem: https://innsida.ntnu.no/avvik
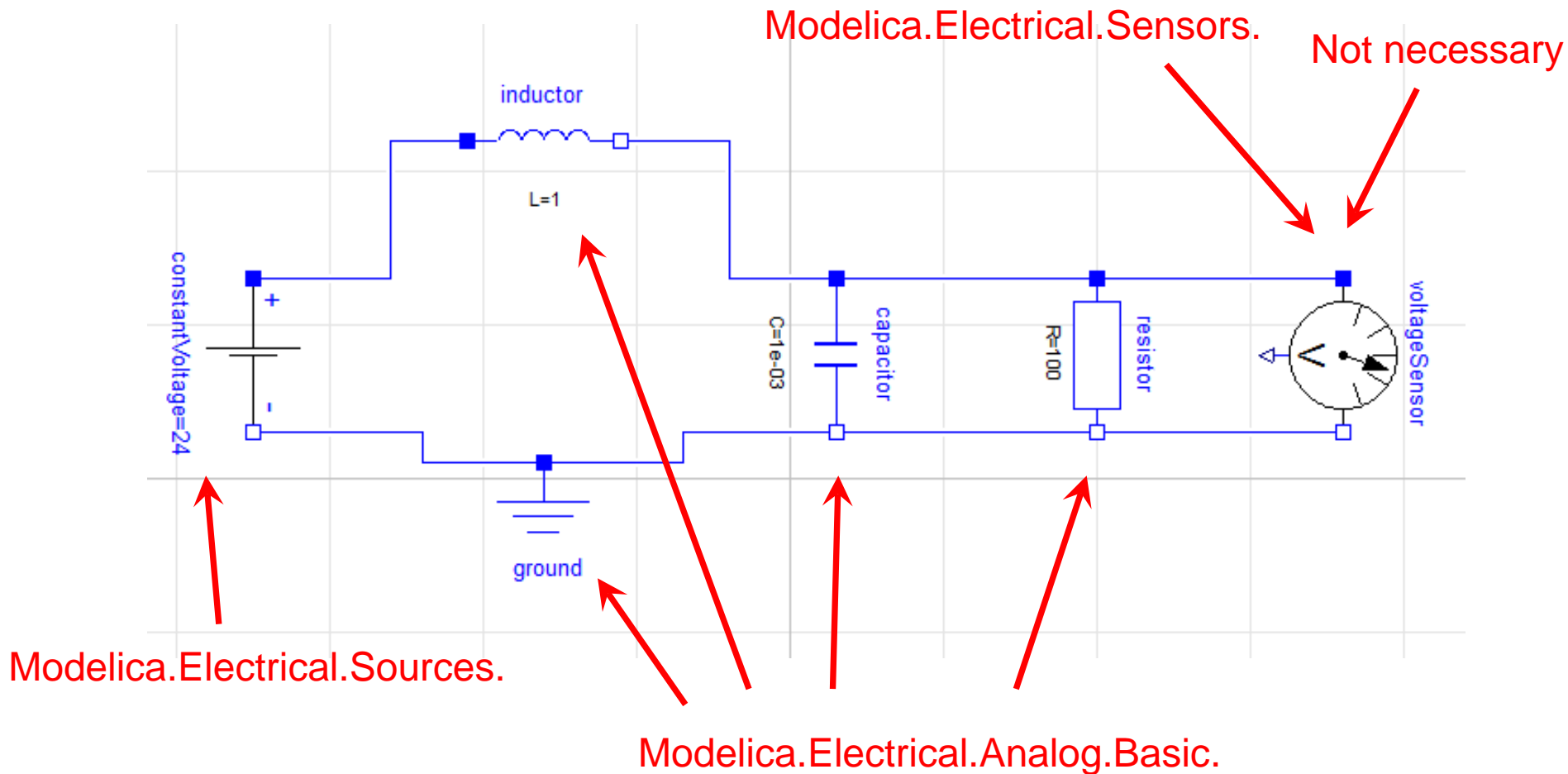
Referansegruppe

- Hver gang et emne har blitt undervist ved NTNU skal faglærer lage en emnerapport som beskriver tilstanden til emnet, inkludert tilbakemelding fra studenter og handlingsplan med tiltak for forbedringer til neste gang

- Tilbakemelding fra studenter innhentes vanligvis gjennom en referansegruppe, som skal gi faglærer en referanse på hva studentene mener om faget

- Referansegruppe:
  - Skal bestå av et representativt utvalg av emnets studenter (kjønn, studieprogram)
  - Minimum tre studenter
  - Skal løpende ta imot innspill fra alle studentene som følger emnet
  - Gjennomfører tre møter i løpet av semesteret: Oppstart, midtveis og ved avslutning av emnet
  - Skriver referansegrupperapport som oppsummerer studentenes synspunkter og forbedringsforslag
  - Alle medlemmene får tilbud om de ønsker bekreftelse på at de har deltatt i referansegruppen
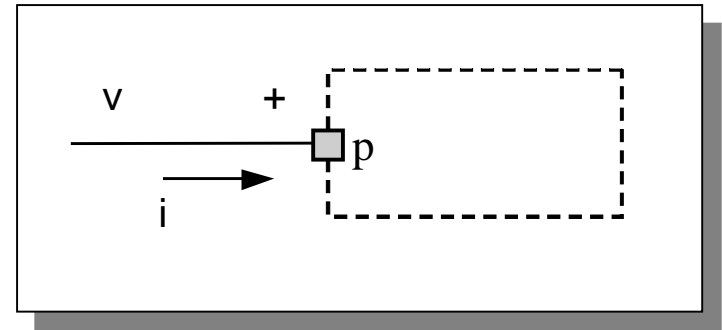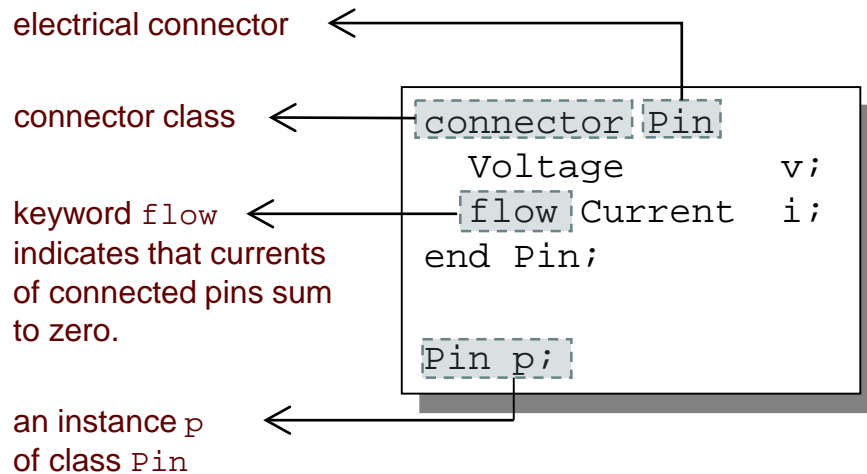
Tilbakekobling for undervisning

Ønsket kvalitet på undervisning

Referanse

Avvik

Instituttledelse (Regulator)

Tilbakemelding

Pådrag

Faglærer/ forelesning (prosess)

Kvalitet på undervisning

Tilstand

Ytre sløyfe

Indre sløyfe

Opplevd kvalitet på undervisning

Studentevaluering (Måleinstrument)

Måling

(Typisk referansegruppe)

2nd Example – Graphical solution with the help of Dymola standard packages

Modelica.Electrical.Sensors.

Not necessary

inductor

L=1

constantVoltage=24

+

-

C=1e-03

capacitor

R=100

resistor

voltageSensor

ground

Modelica.Electrical.Sources.

Modelica.Electrical.Analog.Basic.

Connectors and Connector Classes

Connectors are instances of *connector classes*

electrical connector

connector class

keyword `flow`
indicates that currents
of connected pins sum
to zero.

an instance `p`
of class `Pin`

```
connector Pin
  Voltage        v;
  flow Current   i;
end Pin;


Pin p;
```

v   +

i
p

The `flow` prefix

Two kinds of variables in connectors:

– *Non-flow variables* **potential** or energy level
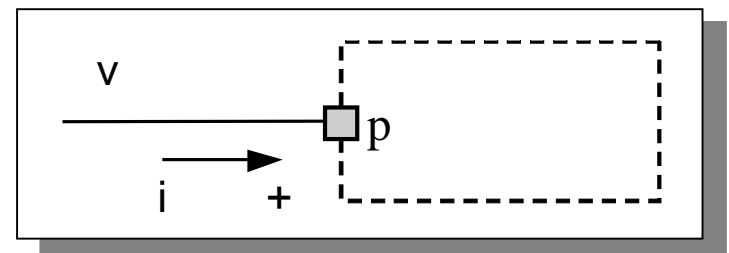
– *Flow variables* represent some kind of **flow**

The `flow` prefix

Coupling

– *Equality coupling*, for non-`flow` variables

• In electrics: $v_1 = v_2 = \ldots = v_n$ (Kirchhoff's 2nd law)

– *Sum-to-zero coupling*, for `flow` variables

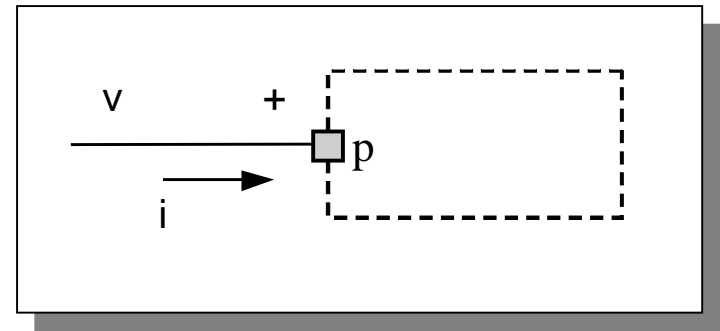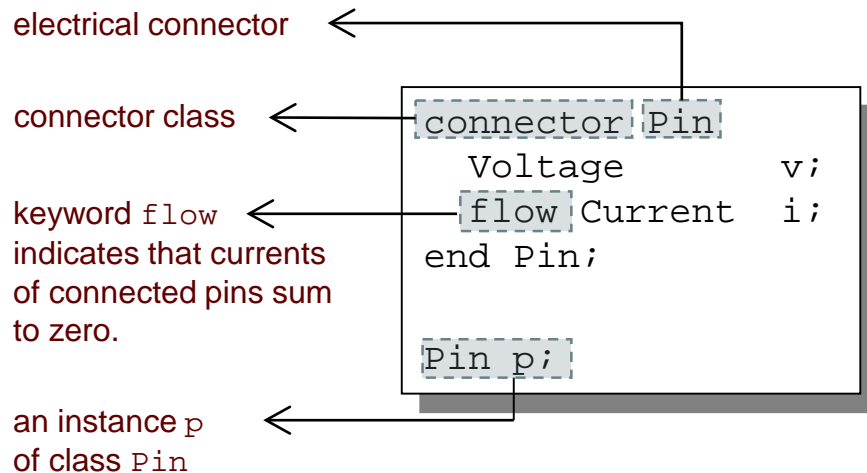• In electrics: $i_1 + i_2 + \ldots + i_n = 0$ (Kirchhoff's 1st law)

The value of a `flow` variable is *positive* when the current or the flow is *into* the component

positive flow direction:
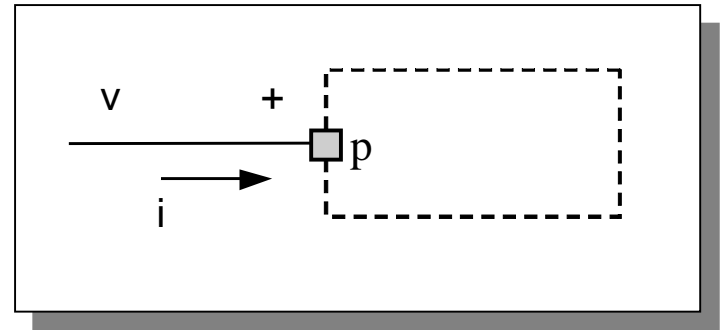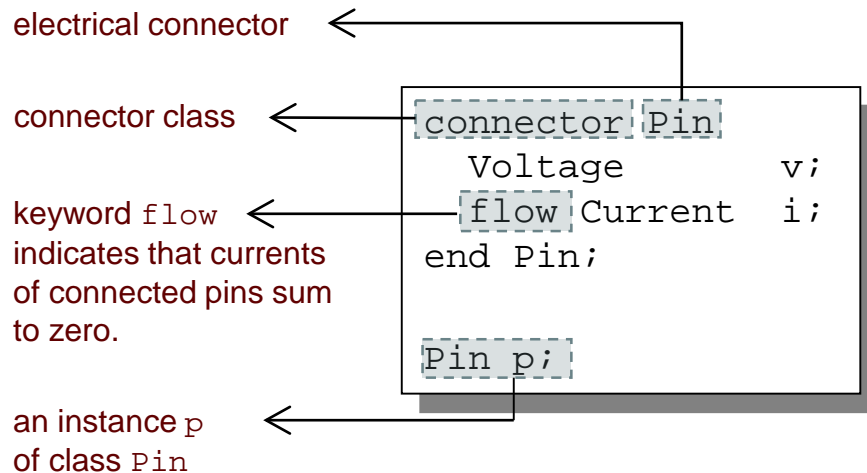
Connectors and Connector Classes

Connectors are instances of *connector classes*

electrical connector

connector class

keyword `flow`
indicates that currents
of connected pins sum
to zero.

```
connector Pin
   Voltage       v;
   flow Current  i;
end Pin;


Pin p;
```

an instance `p`
of class `Pin`

What does the product of the electric pair of connector variables represent?

Connectors and Connector Classes

Connectors are instances of *connector classes*

electrical connector

connector class

```
connector Pin
   Voltage        v;
   flow Current   i;
end Pin;


Pin p;
```

keyword `flow`
indicates that currents
of connected pins sum
to zero.

an instance `p`
of class `Pin`

v   +

i

p

What does the product of the electric pair of connector variables represent?

- $v \left[\frac{Nm}{C}\right] \cdot i \left[\frac{C}{s}\right] = p \left[\frac{Nm}{s} = W\right]$
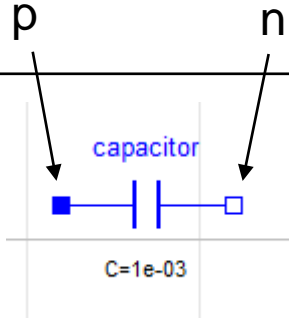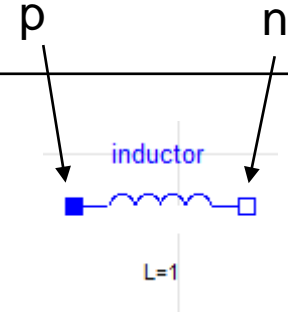
→ Flow of energy
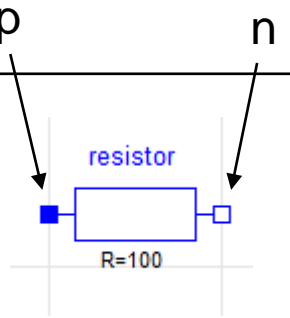
Software Component Model
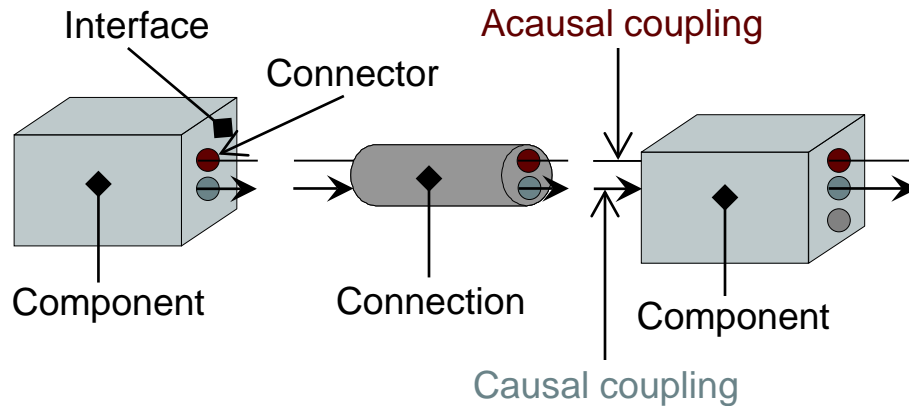
A component class should be defined *independently of the environment,* very essential for *reusability*

A component may internally consist of other components, i.e. *hierarchical* modeling

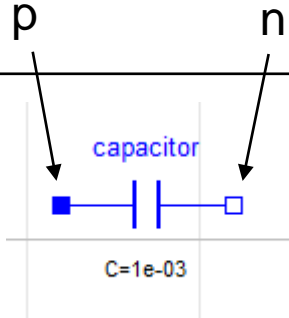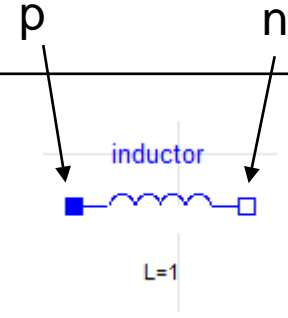Complex systems usually consist of large numbers of *connected* components
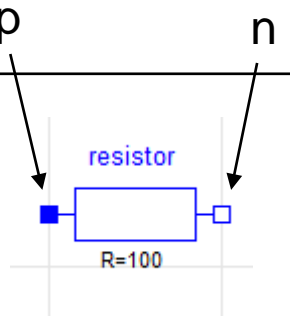
Use of Inheritance & Connectors

- Connector-Name (Pin): p, n

	capacitor	inductor	resistor
	p n	p n	p n
	$C=1e\text{-}03$	$L=1$	$R=100$
Equations:	$0 = p.i + n.i$ $v = p.v - n.v$ $i = p.i$ $i = C * der(v)$	$0 = p.i + n.i$ $v = p.v - n.v$ $i = p.i$ $v = L * der(i)$	$0 = p.i + n.i$ $v = p.v - n.v$ $i = p.i$ $v = R * i$

Use of Inheritance & Connectors

- Connector-Name (Pin): p,n

	p ↘ capacitor ↙ n	p ↘ inductor ↙ n	p ↘ resistor ↙ n
	C=1e-03	L=1	R=100
Equations:	$0 = p.i + n.i$ $v = p.v - n.v$ $i = p.i$ $i = C * der(v)$	$0 = p.i + n.i$ $v = p.v - n.v$ $i = p.i$ $v = L * der(i)$	$0 = p.i + n.i$ $v = p.v - n.v$ $i = p.i$ $v = R * i$

→ Only <u>one</u> equation different

Reuse same components

1) Create connector: ▪

```
connector Pin
  Modelica.SIunits.Voltage v;//identical at connection
  flow Modelica.SIunits.Current i;//sum-to-0 at connection
end Pin;
```

Reuse same components

1) Create connector: ■

```
connector Pin
  Modelica.SIunits.Voltage v;//identical at connection
  flow Modelica.SIunits.Current i;//sum-to-0 at connection
end Pin;
```

2) Create "blueprint" model class TwoPin: ■   □

```
partial model TwoPin "Superclass of elements
with two electrical pins"
  Pin p, n;
  Modelica.SIunits.Voltage v;
  Modelica.SIunits.Current i;
equation
  v = p.v - n.v;
  0 = p.i + n.i;
  i = p.i;
end TwoPin;
```

Reuse same components

1) Create connector: ■

```
connector Pin
  Modelica.SIunits.Voltage v;//identical at connection
  flow Modelica.SIunits.Current i;//sum-to-0 at connection
end Pin;
```

2) Create "blueprint" model class TwoPin: ■   □

```
partial model TwoPin "Superclass of elements
with two electrical pins"
  Pin p, n;
  Modelica.SIunits.Voltage v;
  Modelica.SIunits.Current i;
equation
  v = p.v – n.v;
  0 = p.i + n.i;
  i = p.i;
end TwoPin;
```

partial because:
- Problem is structurally singular
- 6 variables & only 5 equations

Reuse same components with extends

3) Create model with previous components

```
model Resistor "Ideal electrical resistor"
  extends TwoPin;
  parameter Modelica.SIunits.Resistance R;

equation
  R*i = v;
end Resistor;
```

```
model Capacitor "Ideal electrical capacitor"
  extends TwoPin;
  parameter Modelica.SIunits.Capacitance C;

equation
  C*der(v) = i;
end Capacitor;
```
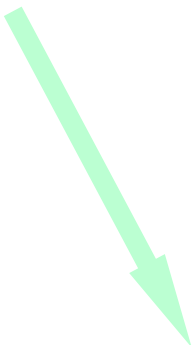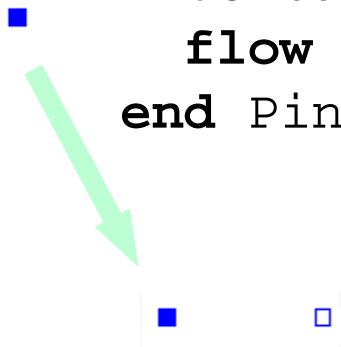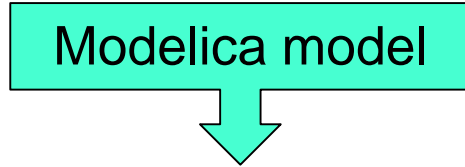
Use of Modelica connectors

```modelica
connector Pin
  Voltage      v;    // identical at connection
  flow Current i;    // sums to zero at connection
end Pin;

        partial model TwoPin
          Pin p, n;    Voltage v; Current i;
        equation
          v = p.v – n.v;
          0 = p.i + n.i;
          i = p.i;
        end TwoPin;

                model Capacitor
                  extends TwoPin;
                  parameter Capacitance C;
                equation
                  C*der(v) = i;
                end Capacitor;
```
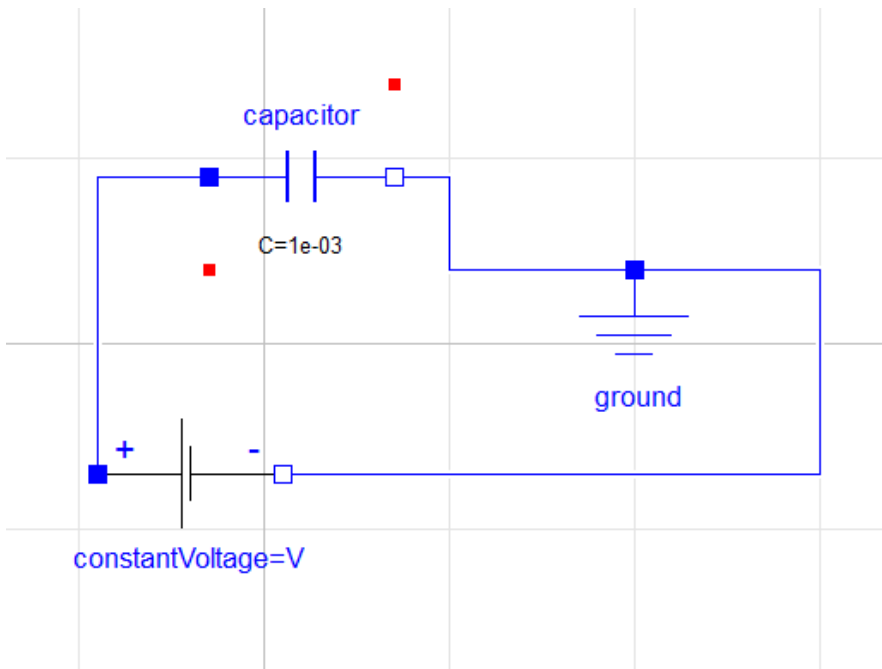
Implementation and Execution of Modelica

Modelica model

Modelica model


```
model Test_Capacitor

//   parameter Real i=1;
//   parameter Real v=20;
//
//   Pin p;

  Modelica.Electrical.Analog.Basic.Capacitor capacitor(C=1e-03);
  Modelica.Electrical.Analog.Sources.ConstantVoltage constantVoltage;
  Modelica.Electrical.Analog.Basic.Ground ground;
equation
  connect(constantVoltage.p, capacitor.p);
  connect(capacitor.n, ground.p);
  connect(ground.p, constantVoltage.n);

end Test_Capacitor;

partial package Modelica.Icons.Package "Icon for standard packages"

end Package;

model Modelica.Electrical.Analog.Basic.Capacitor
  "Ideal linear electrical capacitor"
  extends Interfaces.OnePort;
  parameter SI.Capacitance C(start=1) "Capacitance";

equation
  i = C*der(v);
end Capacitor;

partial package Modelica.Icons.InterfacesPackage
  "Icon for packages containing interfaces"
//extends Modelica.Icons.Package;
end InterfacesPackage;

partial model Modelica.Electrical.Analog.Interfaces.OnePort
  "Component with two electrical pins p and n and current i from p to n"

  SI.Voltage v "Voltage drop between the two pins (= p.v - n.v)";
  SI.Current i "Current flowing from pin p to pin n";
  PositivePin p
    "Positive pin (potential p.v > n.v for positive voltage drop v)";
  NegativePin n "Negative pin";
equation
  v = p.v - n.v;
  0 = p.i + n.i;
  i = p.i;
end OnePort;

connector Modelica.Electrical.Analog.Interfaces.PositivePin
  "Positive pin of an electric component"
  Modelica.SIunits.Voltage v "Potential at the pin";
  flow Modelica.SIunits.Current i "Current flowing into the pin";
```
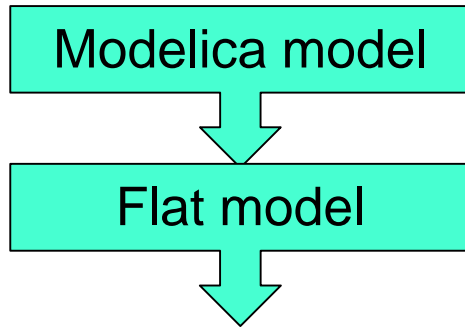
Implementation and Execution of Modelica

Modelica model

Flat model

Parsed, preprocessing, flattening

Flat model

```
model Test_Capacitor
parameter Modelica.SIunits.Capacitance capacitor.C(start = 1) = 0.001
  "Capacitance";
parameter Modelica.SIunits.Voltage constantVoltage.V = 1 "Value of constant voltage";

Modelica.SIunits.Voltage capacitor.v "Voltage drop between the two pins (= p.v - n.v)";
Modelica.SIunits.Current capacitor.i "Current flowing from pin p to pin n";
Modelica.SIunits.Voltage capacitor.p.v "Potential at the pin";
Modelica.SIunits.Current capacitor.p.i "Current flowing into the pin";
Modelica.SIunits.Voltage capacitor.n.v "Potential at the pin";
Modelica.SIunits.Current capacitor.n.i "Current flowing into the pin";
Modelica.SIunits.Voltage constantVoltage.v "Voltage drop between the two pins (= p.v - n.v)";
Modelica.SIunits.Current constantVoltage.i "Current flowing from pin p to pin n";
Modelica.SIunits.Voltage constantVoltage.p.v "Potential at the pin";
Modelica.SIunits.Current constantVoltage.p.i "Current flowing into the pin";
Modelica.SIunits.Voltage constantVoltage.n.v "Potential at the pin";
Modelica.SIunits.Current constantVoltage.n.i "Current flowing into the pin";
Modelica.SIunits.Voltage ground.p.v "Potential at the pin";
Modelica.SIunits.Current ground.p.i "Current flowing into the pin";

// Equations and algorithms

  // Component capacitor
  // class Modelica.Electrical.Analog.Basic.Capacitor
    // extends Modelica.Electrical.Analog.Interfaces.OnePort
    equation
      capacitor.v = capacitor.p.v-capacitor.n.v;
      0 = capacitor.p.i+capacitor.n.i;
      capacitor.i = capacitor.p.i;
    // end of extends
  equation
    capacitor.i = capacitor.C*der(capacitor.v);

  // Component constantVoltage
  // class Modelica.Electrical.Analog.Sources.ConstantVoltage
    // extends Modelica.Electrical.Analog.Interfaces.OnePort
    equation
      constantVoltage.v = constantVoltage.p.v-constantVoltage.n.v;
      0 = constantVoltage.p.i+constantVoltage.n.i;
      constantVoltage.i = constantVoltage.p.i;
    // end of extends
  equation
    constantVoltage.v = constantVoltage.V;

  // Component ground
  // class Modelica.Electrical.Analog.Basic.Ground
  equation
    ground.p.v = 0;

  // Component
  // class Test_Capacitor
  equation
    capacitor.n.i+constantVoltage.n.i+ground.p.i = 0.0;
    constantVoltage.n.v = capacitor.n.v;
    ground.p.v = capacitor.n.v;
    capacitor.p.i+constantVoltage.p.i = 0.0;
    constantVoltage.p.v = capacitor.p.v;

end Test_Capacitor;
```
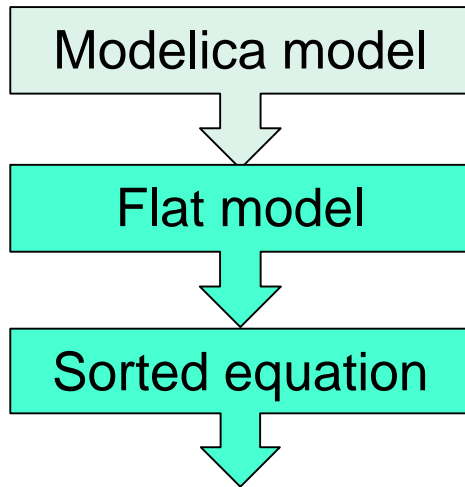
Implementation and Execution of Modelica

Modelica model

Flat model

Sorted equation

Parsed, preprocessing, flattening

"make equations causal", perform BLT transformation

Lower Triangular Matrix (Example)

	v_G	v_{C1}	v_{R1}	u_R	i_{R1}	i_{S1}	i_{C1}	du_c	i_G
1)	x								
6)	x	x							
2)	x		x						
4)		x	x	x					
3)				x	x				
7)					x	x			
8)					x		x		
5)							x	x	
9)						x	x		x

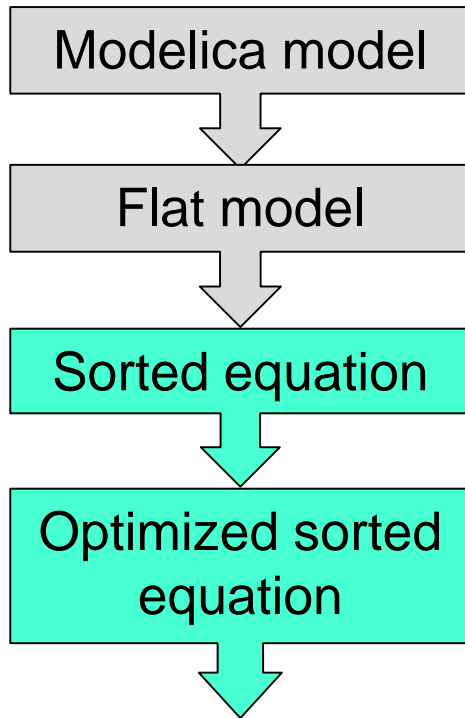**Causal List:**

1) $v_G := 0$
6) $v_{C1} := -u_c + v_G$
2) $v_{R1} := v_G + 10V$
4) $u_R := v_{C1} - v_{R1}$
3) $i_{R1} := u_R/R$
7) $i_{S1} := i_{R1}$
8) $i_{C1} := i_{R1}$
5) $\frac{du_c}{dt} := i_{C1}/C$
9) $i_G := i_{C1} - i_{S1}$

Block Lower Triangular Matrix (Example)

Implementation and Execution of Modelica

Modelica model

Parsed, preprocessing, flattening

Flat model

"make equations causal", perform BLT transformation

Sorted equation

Optimize and eliminate equation

Optimized sorted equation

Implementation and Execution of Modelica

Modelica model	Parsed, preprocessing, flattening
Flat model	"make equations causal", perform BLT transformation
Sorted equation	Optimize and eliminate equation
Optimized sorted equation	
C Code	Generate C code
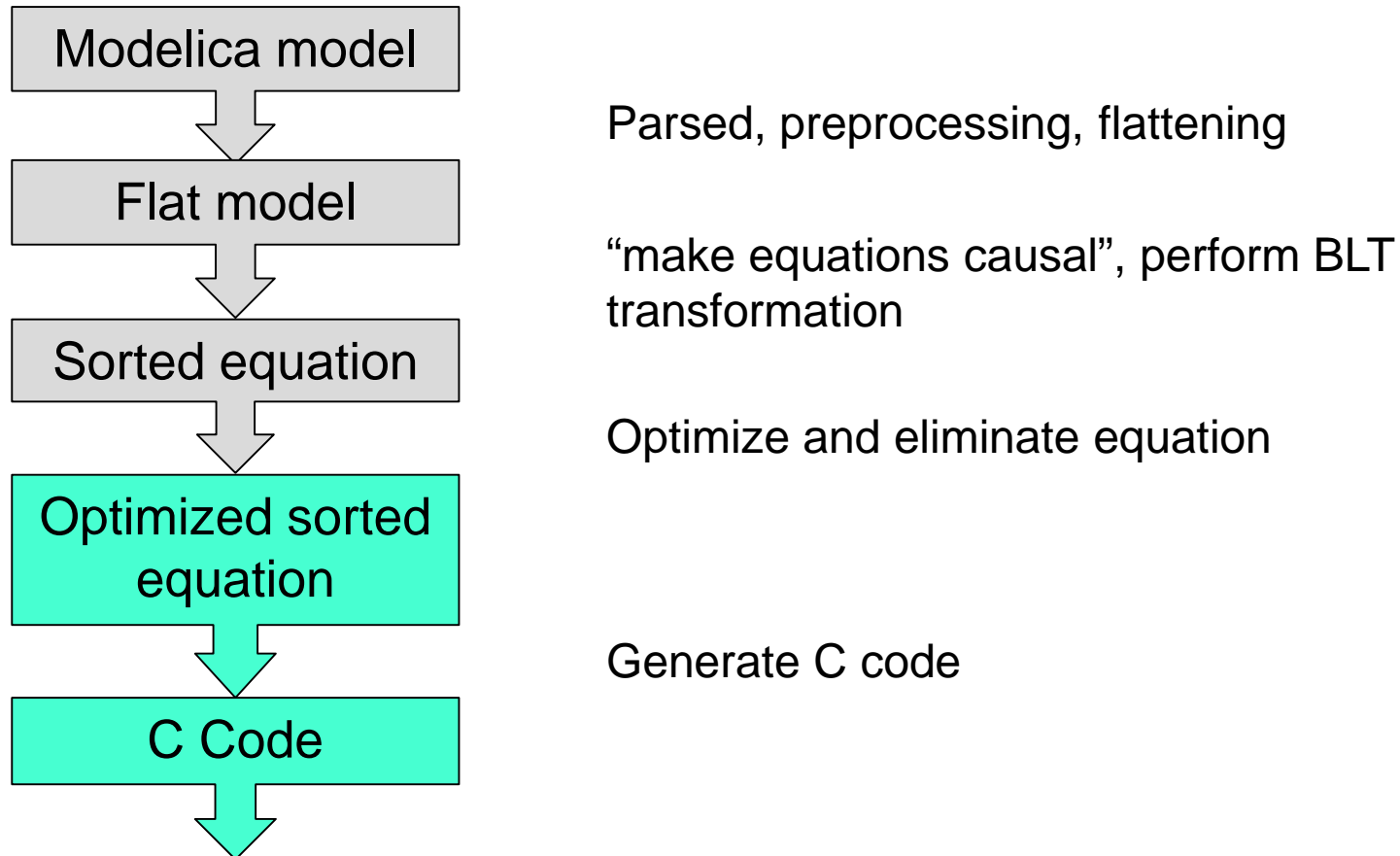
C Code

```c
/* DSblock model generated by Dymola from Modelica model Test_Capacitor
 Dymola Version 2014 FD01 (32-bit), 2013-10-17 translated this at Mon Jan 11 08:49:59 2016

    */

#include <matrixop.h>
/* Declaration of C-structs */
/* Prototypes for functions used in model */
/* Codes used in model */
/* DSblock C-code: */

#include <moutil.c>
PreNonAliasDef(0)
PreNonAliasDef(1)
DYMOLA_STATIC const char*modelName="Test_Capacitor";
DYMOLA_STATIC const char*usedLibraries[]={0};
DYMOLA_STATIC const char*dllLibraryPath[]={0};
DYMOLA_STATIC const char*default_dymosim_license_filename=
 "c:/users/leifea/appdata/roaming/dynasim/dymola.lic";
#include <dsblock1.c>

/* Define variable names. */

#define Sections_

TranslatedEquations

InitialSection
W_[3] = 0;
W_[0] = 0.0;
W_[1] = 0.0;
W_[10] = 0.0;
W_[4] = 0.0;
W_[2] = 0.0;
W_[5] = 0.0;
W_[8] = 0.0;
W_[7] = 0.0;
W_[6] = 0.0;
W_[9] = 0.0;
BoundParameterSection
InitialSection
InitialSection
InitialStartSection
InitialSection
DefaultSection
InitializeData(0)
InitialSection
InitialSection
Init=false;InitializeData(2);Init=true;
EndInitialSection

OutputSection

DynamicsSection

AcceptedSection1

AcceptedSection2

DefaultSection
InitializeData(1)
EndTranslatedEquations
```
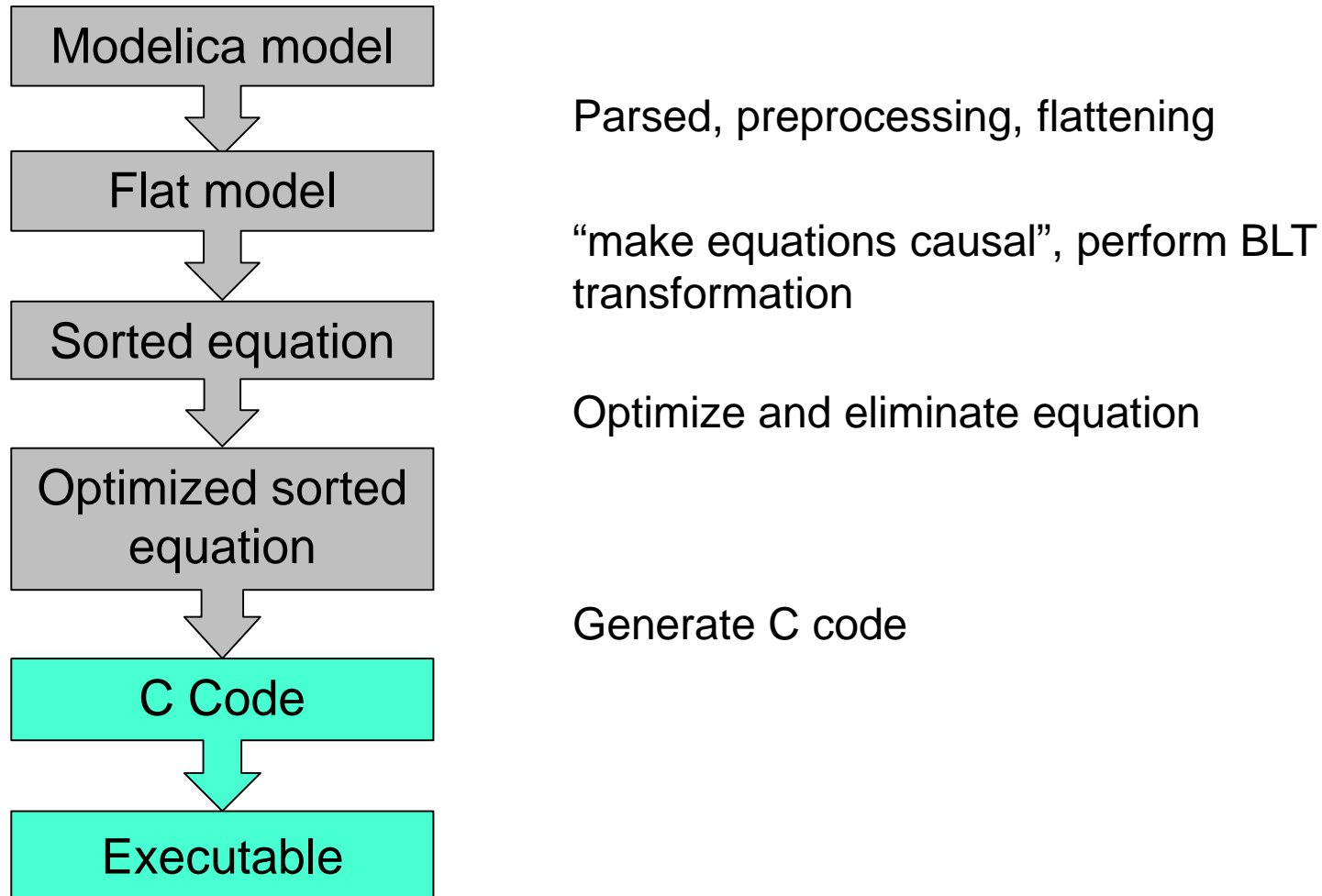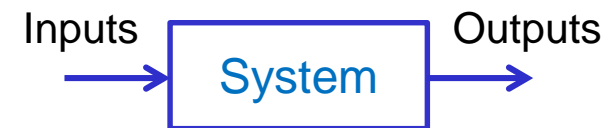
Implementation and Execution of Modelica

```
┌─────────────────────┐
│   Modelica model    │        Parsed, preprocessing, flattening
└─────────────────────┘
          ↓
┌─────────────────────┐
│     Flat model      │        "make equations causal", perform BLT
└─────────────────────┘        transformation
          ↓
┌─────────────────────┐
│   Sorted equation   │        Optimize and eliminate equation
└─────────────────────┘
          ↓
┌─────────────────────┐
│  Optimized sorted   │
│     equation        │
└─────────────────────┘
          ↓                    Generate C code
┌─────────────────────┐
│       C Code        │
└─────────────────────┘
          ↓
┌─────────────────────┐
│     Executable      │
└─────────────────────┘
```

Lecture 2:

- About modeling and simulation (F1)
- Why&what do control engineers need to model?
  - Examples (mostly examples I have worked with)
- Model types (E1.1-1.3,E2.1-2.2)
  - State space models, transfer functions
  - Linear models, nonlinear models

We will *model* and *simulate systems*

What is a system, what is a model and why simulate?

- "A system is an object or collection of objects whose properties we want to study."

"A system is what we distinguish as a system"

- Note that we are usually interested in *specific properties* of these objects
  – that is, there is usually an *abstraction* implied

  – Ex.: Our system is a pendulum, but we are only interested in (say) pendulum motion, not pendumlum temperature, electrical conductivity, bacterial contamination, etc.

- <u>Our</u> systems have inputs and outputs

  – Inputs: Variables of the environment that influences the behavior of the system

  – Outputs: Variables determined by the system that may influence the environment

Inputs → System → Outputs

Systems and experiments

Inputs → | System | → Outputs

- "An *experiment* is the process of extracting information from a system by exercising its inputs."

- Challenges:
  - Not all inputs may be manipulated
    - Manipulated vs. disturbance inputs
  - Not all outputs may be measured

  - Experiments may be
    - too expensive
    - too dangerous
    - too slow

  - Our system may not exist (yet)

The Model concept

- "A *model* of a system is anything an *experiment* can be applied to in order to answer questions about that *system*."

    – Mental model

    – Verbal model

    – Physical model

    – Mathematical model

- Mathematical models are often implemented in computers (virtual prototypes)

Simulation

- "A *simulation* is an experiment performed on a model."
  - For us: Mathematical model implemented in computer

- Why simulate instead of experiments on system?
  - Experiments are too *expensive*, too *dangerous*, too *slow*, or the system to be investigated does *not yet exist*.
  - Variables may be *inaccessible.*
  - Easy *manipulation of models.*
  - Suppression of *disturbances* and *second-order effects.*

- Our (main) purpose for modeling and simulation: Design, testing and validation of control systems

Dangers of modeling and simulation

(from Ljung&Glad, "Modelbygge og simulering")

Models are based on assumptions with limited validity
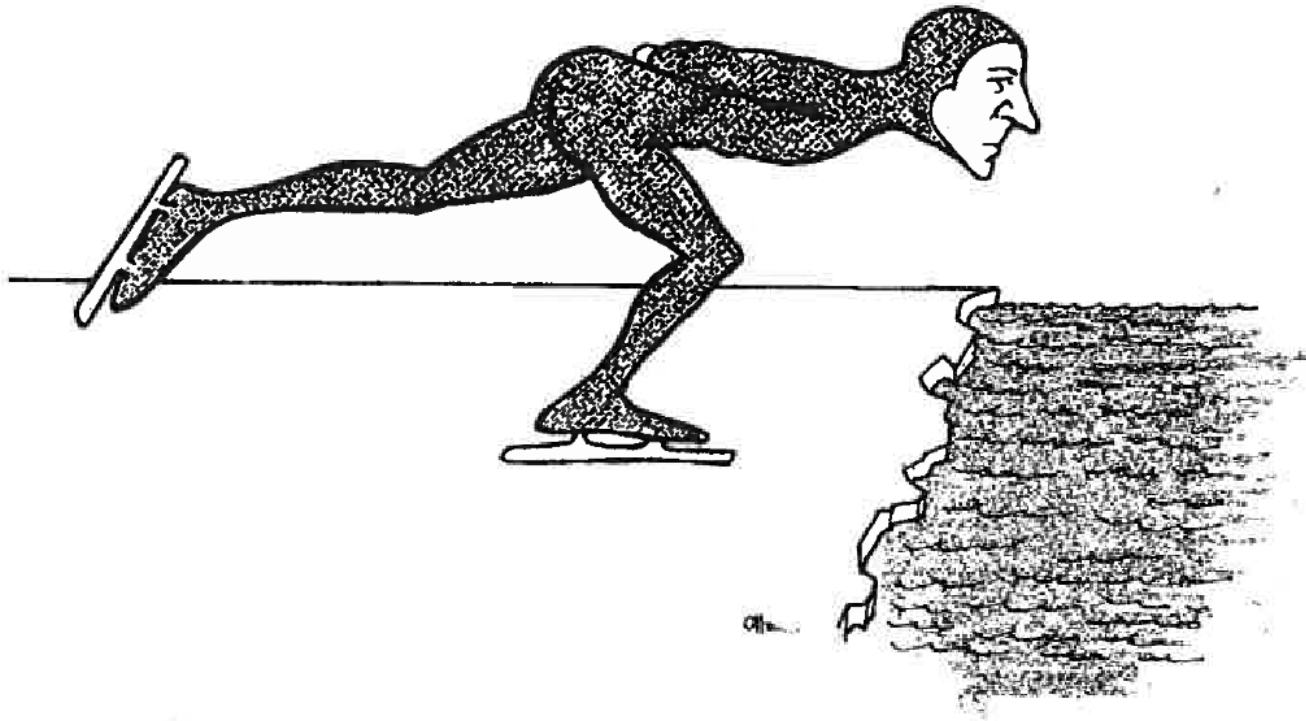
Figur 16.4: Faran att extrapolera modeller.

Models are not the 'real world' (The Pygmalion-effect)
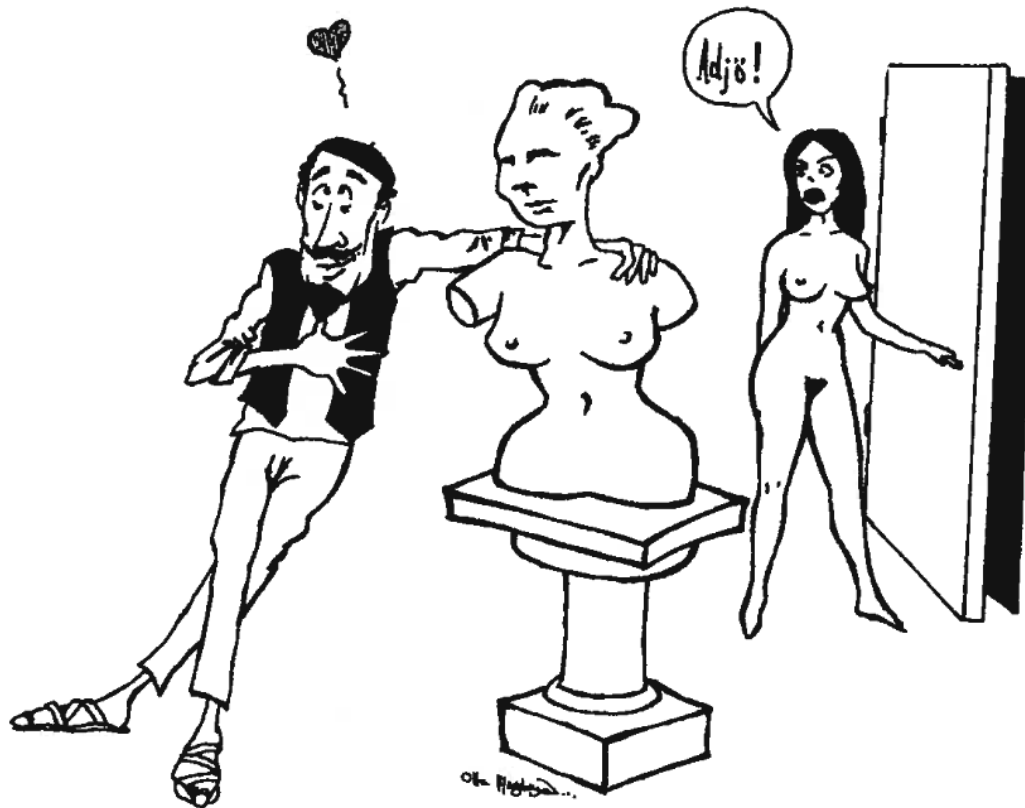
Figur 16.5: Bli inte förälskad i modellen.

Do not force reality into the constraints of a model (The Procrustes effect)
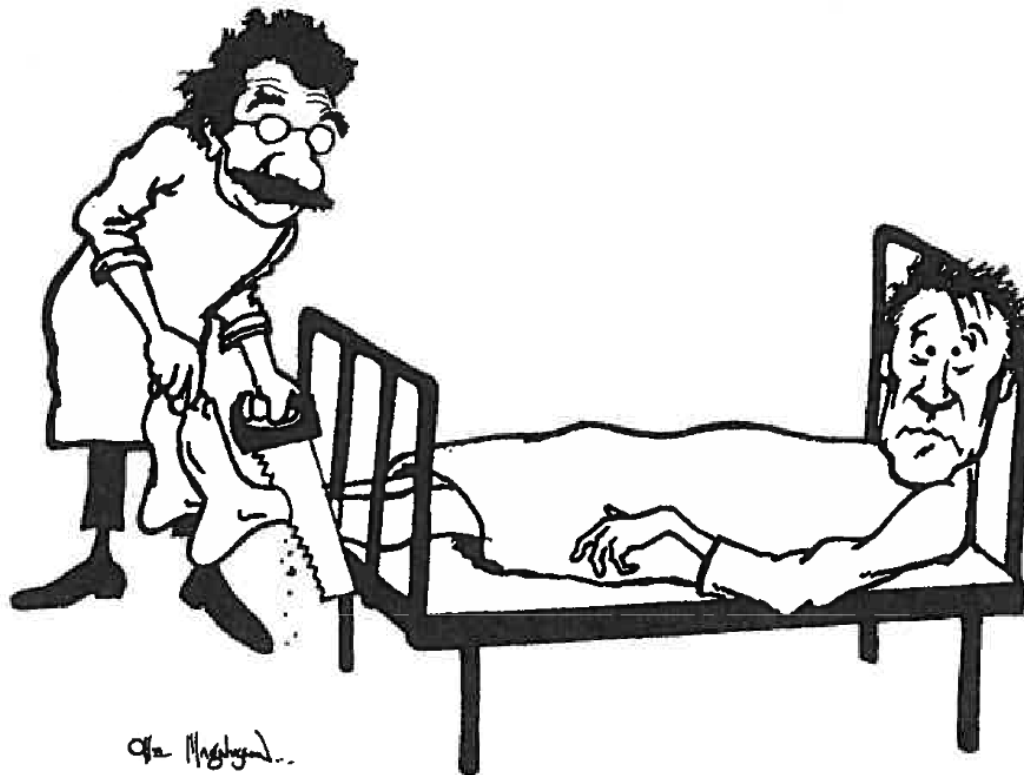
Figur 16.6: Försök inte anpassa verkligheten till modellen.

Keep an engineering sense about your modeling process

Figur 16.7: Förkasta inte fakta som är i strid med modellen.

Figur 16.8: Använd gärna flera olika modeller.

Kinds of mathematical models

- Static vs <u>dynamic</u>
  - Does the model involve time?

- <u>Continuous</u> vs discrete
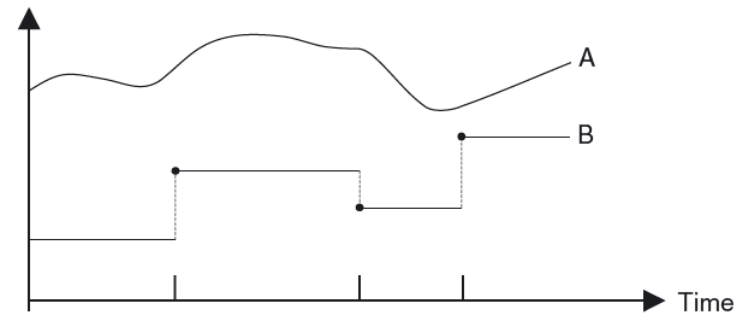
Figure 1.4 Discrete-time system B changes values only at certain points in time, whereas continuous-time systems like A evolve values continuously.

- <u>Lumped</u> vs distributed

$$\frac{\mathrm{d}}{\mathrm{d}t} T(t) = -aT(t)$$

$$\frac{\partial}{\partial t} T(t, x) = -\frac{\partial^2}{\partial x^2} T(t, x)$$

- Stochastic vs <u>deterministic</u>

- Empirical vs '<u>first principles</u>'
  - Build models from measurement data, or derive from laws of physics?
  - Or both: "Grey box"

Examples of modeling and simulation

Power

- Power productions
  - Gas turbines, hydro-electric power, bio, wind turbines, fuel cells, etc.
- Pollution&emissions
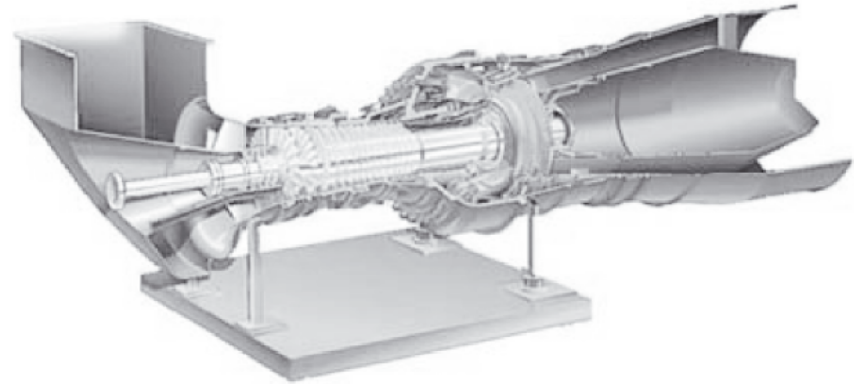 - CO2 capture
- Smart grids
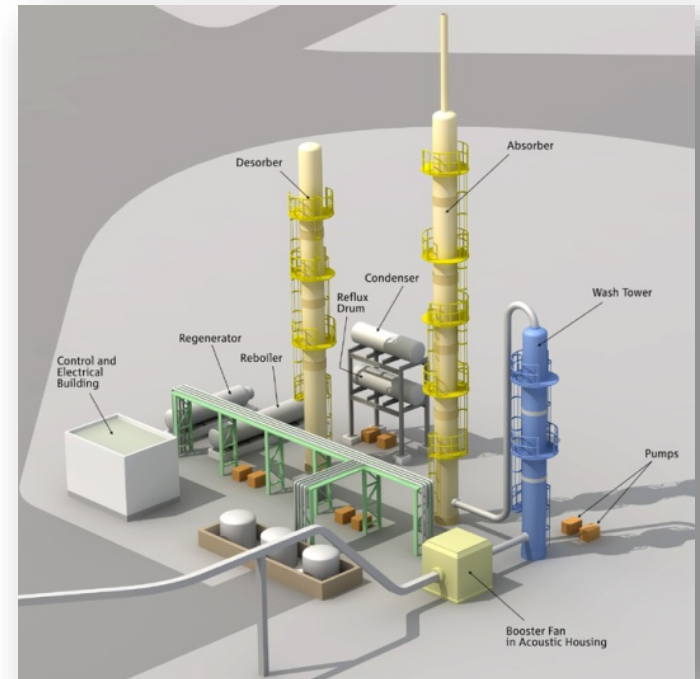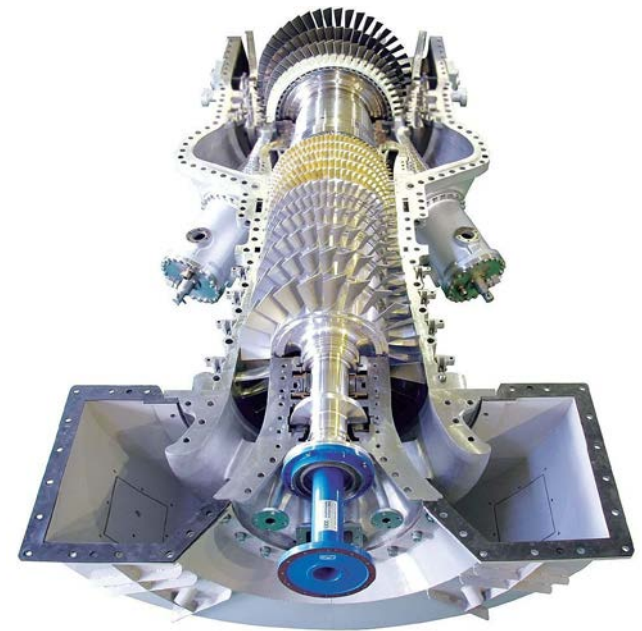
Gas turbine

Figure 1.7 Schematic picture of the gas turbine GTX100. (Courtesy Siemens Industrial Turbomachinery AB, Finspång, Sweden.)
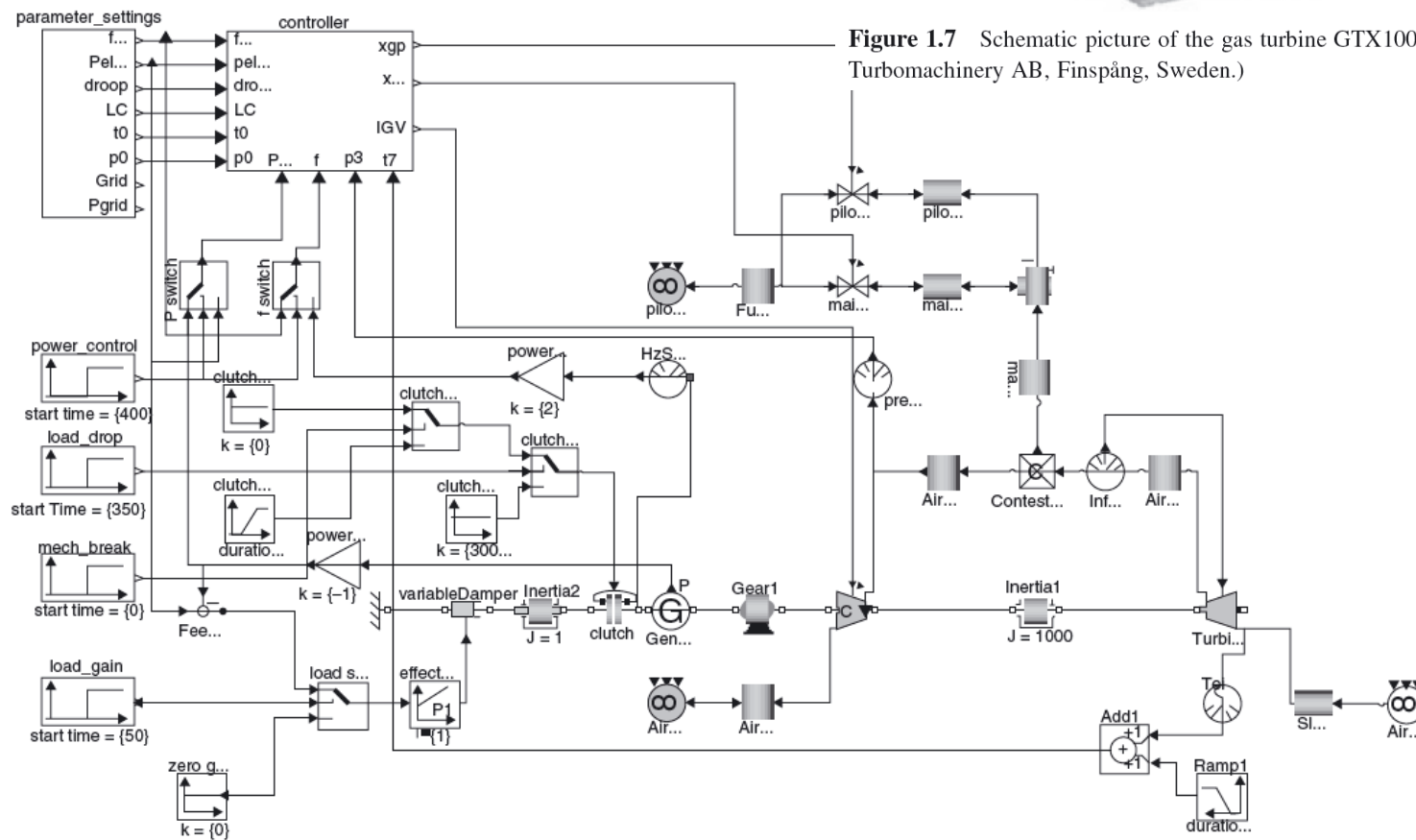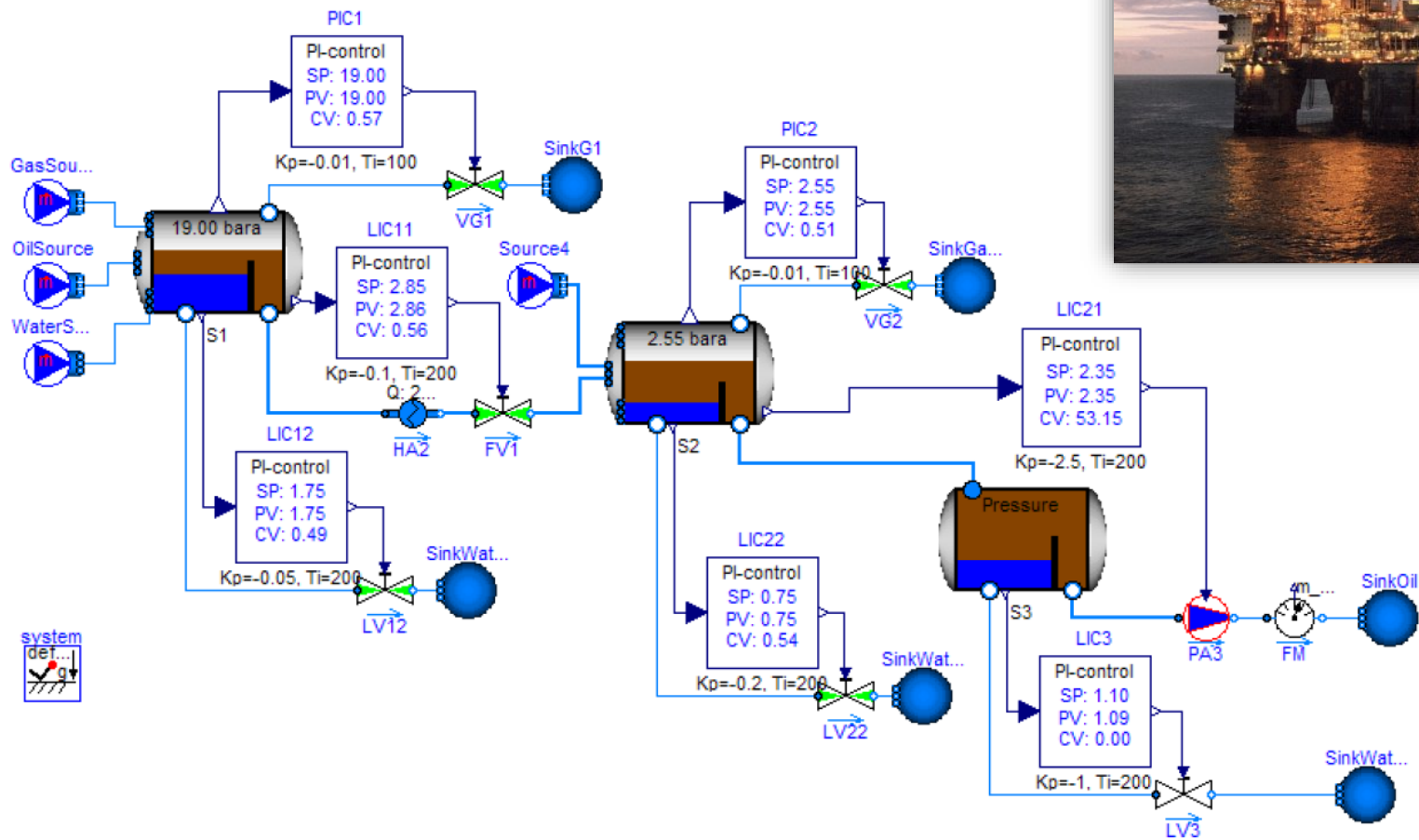

Figure 1.8 Detail of power cutoff mechanism in 40 MW GTX100 gas turbine model. (Courtesy Siemens Industrial Turbomachinery AB, Finspång, Sweden.)

Offshore production of oil and gas

- Production facilities
  - Separation, compression, routing, ...
- Subsea production, remote&arctic production
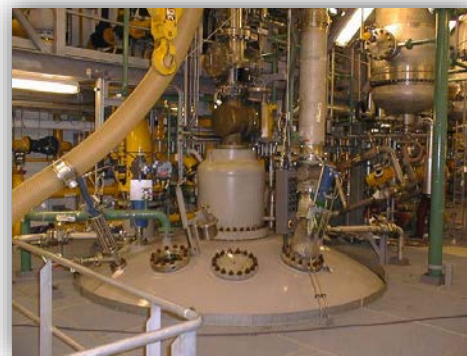- Automation in offshore drilling

Offshore oil and gas processing (topside)

Courtesy: Cybernetica AS

Petrochemical industry

- Refineries, oil&gas processing
- Polymer production
- Methanol production

Production of metals

- Aluminium, ferrosilicon, ferromanganese, silicon, ...
- Si-wafers for solar cells

Navigation and control

- Underwater, ships, cars, air
- Unmanned vehicles
- Robots
- Mechatronics

Courtesy: Maritime Robotics

Courtesy: Prox Dynamics

Active safety in cars

- Control systems that prevent accidents
- Examples:
  - Anti-lock braking systems (ABS)
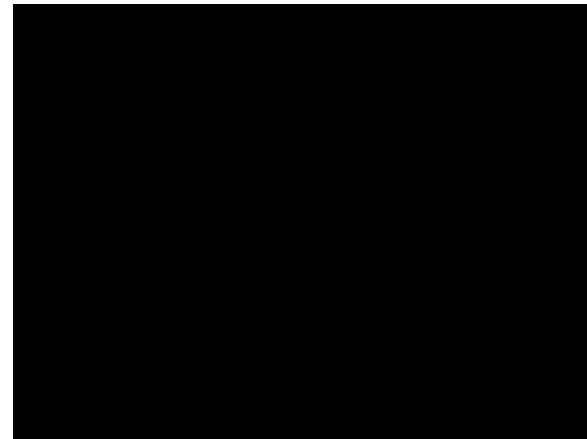 - Anti-skid (ESC)
  - Collision avoidance
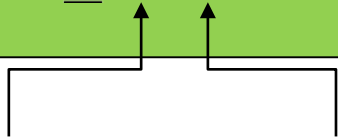

Uten ESC:
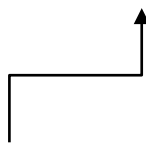
Med ESC:

State space models

$$\dot{x}_1 = f_1(x_1, x_2, \ldots, x_n, u_1, \ldots, u_m, t)$$

$$\vdots$$

$$\dot{x}_n = f_n(x_1, x_2, \ldots, x_n, u_1, \ldots, u_m, t)$$

$$\underline{\dot{x}} = \underline{f}(\underline{x}, \underline{u}, t)$$

State vector   Input vector

$$\underline{y} = \underline{h}(\underline{x}, \underline{u}, t)$$

Output vector

Linear time invariant model (LTI)

$$\underline{\dot{x}} = \mathbf{A}\underline{x} + \mathbf{B}\underline{u}$$

System matrix   Control matrix

$$\underline{y} = \mathbf{C}\underline{x} + \mathbf{D}\underline{u}$$

Output matrix   Feed-Forward matrix

Second order models of mechanical systems (I)

$$M(q)\ddot{q} + f(q, \dot{q}) = u$$

q: generalised coordinates

u: generalised force

Mass - Spring - Damper system

$$m\ddot{x} + d\dot{x} + kx = u$$

Second order models of mechanical systems (II)

$$M(\underline{q}) = M^T(\underline{q}) \quad : \quad \text{mass matrix}$$

$$\text{positive definite} \quad z^T M z > 0$$

all eigenvalues λi of M are positive

State - space form :

$$\underline{x}_1 = q \qquad \underline{x}_2 = \dot{q}$$

$$\underline{\dot{x}}_1 = x_2$$

$$\underline{\dot{x}}_2 = M^{-1}(\underline{x}_1) \left[u - f(\underline{q}, \underline{\dot{q}}) \right]$$

Linearization of state space models

$$\dot{\underline{x}} = \underline{f}(\underline{x}, \underline{u}, t)$$
$$\underline{y} = \underline{h}(\underline{x}, \underline{u}, t)$$

- We linearize around a solution of the system
  →Control point

$$\dot{\underline{x}}_0 = \underline{f}(\underline{x}_0(t), \underline{u}_0(t), t)$$
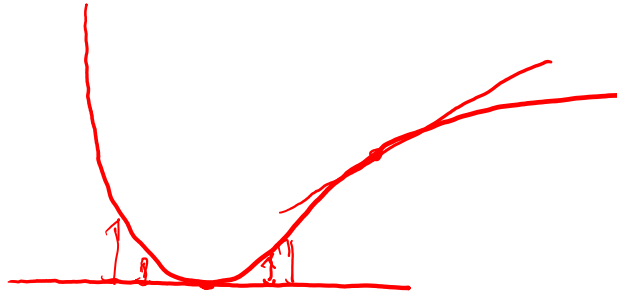
→Typically (steady-state):

$$0 = \underline{f}(\underline{x}_0(t), \underline{u}_0(t), t)$$

Principle example – linearisation

Define   purturbation Δx , Δu , Δy

$(x_1 - x_0)$ $(u_1 - u_0)$

$$x(t) = x_0(t) + \Delta \underline{x}(t)$$

$$\underline{u}(t) = u_0(t) + \Delta u(t)$$

$$\underline{y}(t) = h(x_0(t), u_0(t), t) + \Delta y(t)$$

Taylor series expansion about (x_0, u_0)

$$\dot{\underline{x}} = f(\underline{x}_0, u_0, t) + \underbrace{\frac{\partial f}{\partial x}\Big|_{x_0, u_0}}_{A} \cdot \Delta\underline{x} + \underbrace{\frac{\partial f}{\partial u}\Big|_{x_0, u_0}}_{B} \Delta u \quad + h.o.t.$$

$$\uparrow$$
$$\dot{x}_0$$

$$\Delta\dot{\underline{x}}(t) = A(t) \cdot \Delta\underline{x}(t) + B(t) \cdot \Delta\underline{u}(t)$$

$$\Delta y(t) = C(t) \cdot \Delta\underline{x}(t) + D(t) \cdot \Delta\underline{u}(t)$$

Book: Example 2

drag coefficient   area   throttle constant

$$m\,\dot{v} = \frac{1}{2} C_D \rho A v^2 + K_t u$$

mass   acceleration   air density   velocity   throttle input

$$f_1 = \dot{v} = \frac{1}{2m} C_D \rho A v^2 + \frac{1}{m} K_t u$$

$$A = \left. \frac{\partial f_1}{\partial v} \right|_{\substack{v=v_0 \\ u=u_0}} = \frac{1}{m} C_D \rho A v_0$$

$$B = \left. \frac{\partial f_1}{\partial u} \right|_{\substack{v=v_0 \\ u=u_0}} = \frac{1}{m} K_t$$

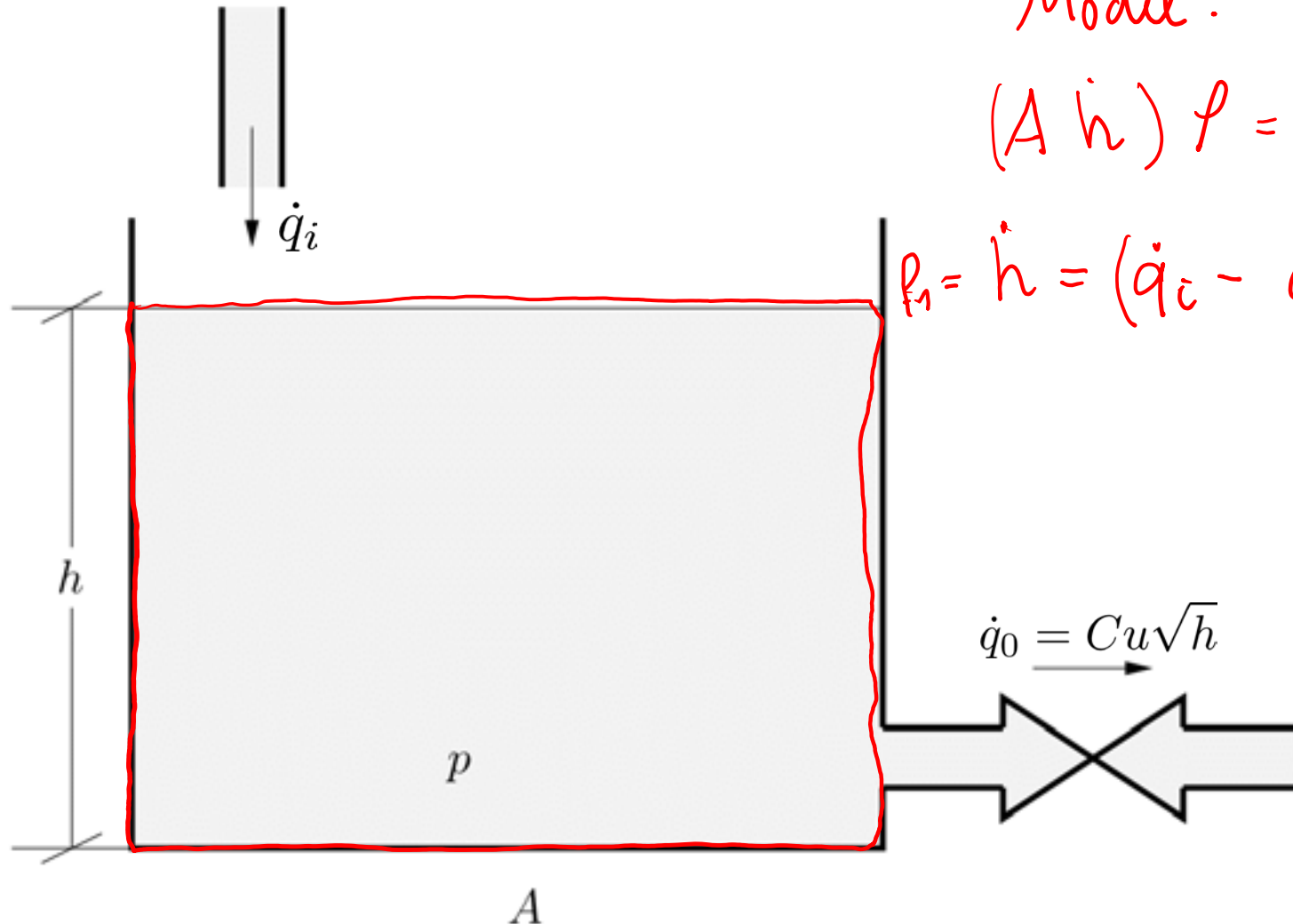$$\Delta \dot{v} = \frac{1}{m} C_D \rho A v_0 \cdot \Delta v + \frac{1}{m} K_t \Delta u$$

Book: Example 2 – steady state

$$\text{Linearise} \quad \text{around} \quad m\dot{v} = 0$$

$$0 = \frac{1}{2} C_{D} \rho A v_{0}^{2} + K_{t} u_{0}$$

$$u_{0} = -\frac{1}{2K_{t}} C_{D} \rho A v_{0}^{2}$$

Example: Linearization tank

Model:

$$(A\dot{h})\rho = \rho(\dot{q}_i - \dot{q}_0)$$

$$\Rightarrow \dot{h} = (\dot{q}_i - Cu\sqrt{h})\frac{1}{A}$$

$$\dot{q}_i$$

$$\dot{q}_0 = Cu\sqrt{h}$$

h

p

A

Example: Linearization tank (I) $f_1 = \dot{h} = \frac{1}{A}(\dot{q}_i - Cu\sqrt{h})$

Linearise around $\dot{h}_0 = 0$ $h(t) = h_0$

steady-state: $\dot{q}_i = C_{u_0}\sqrt{h_0}$

$$u_0 = \frac{\dot{q}_i}{C\sqrt{h_0}}$$

$$A = \frac{\partial f_1}{\partial h}\bigg|_{\substack{h=h_0 \\ u=u_0}} = -\frac{1}{A}Cu\frac{1}{2\sqrt{h}}\bigg|_{\substack{h=h_0 \\ u=u_0}}$$

$$= -\frac{1}{A}Cu_0\frac{1}{2\sqrt{h_0}}$$

$$B = \frac{\partial f_1}{\partial u}\bigg|_{\substack{h=h_0 \\ u=u_0}} = -\frac{1}{A}C\sqrt{h_0}\bigg|_{\substack{h=h_0 \\ u=u_0}} = -\frac{1}{A}C\sqrt{h_0}$$

Example: Linearization tank (II)

$$\Delta \dot{h} = - \frac{C u_0}{2 A \sqrt{h_0}} \Delta h - \frac{C \sqrt{h_0}}{A} \Delta u$$

$$= - \frac{C \dot{q_i}}{2 A \sqrt{h_0} \, C \sqrt{h_0}} \Delta h - \frac{C \sqrt{h_0}}{A} \Delta u$$

$$= - \frac{\dot{q_0}}{2 A h_0} \Delta h - \frac{C \sqrt{h_0}}{A} \Delta u$$

Homework (recommended)

- Linearise the following model around $\theta = \dot{\theta} = u = 0$:

$$\ddot{\theta} = -\frac{u}{l}\cos\theta + \frac{g}{l}\sin\theta$$

  – If you need help use Example 4 in the book for guidance

- Read section 2.3. in the book
  – Try to understand the Mass-spring-damper example in Section 2.3.4

TTK4130 Modeling and Simulation