Data flow models

Peter Marwedel
Informatik 12
TU Dortmund
Germany

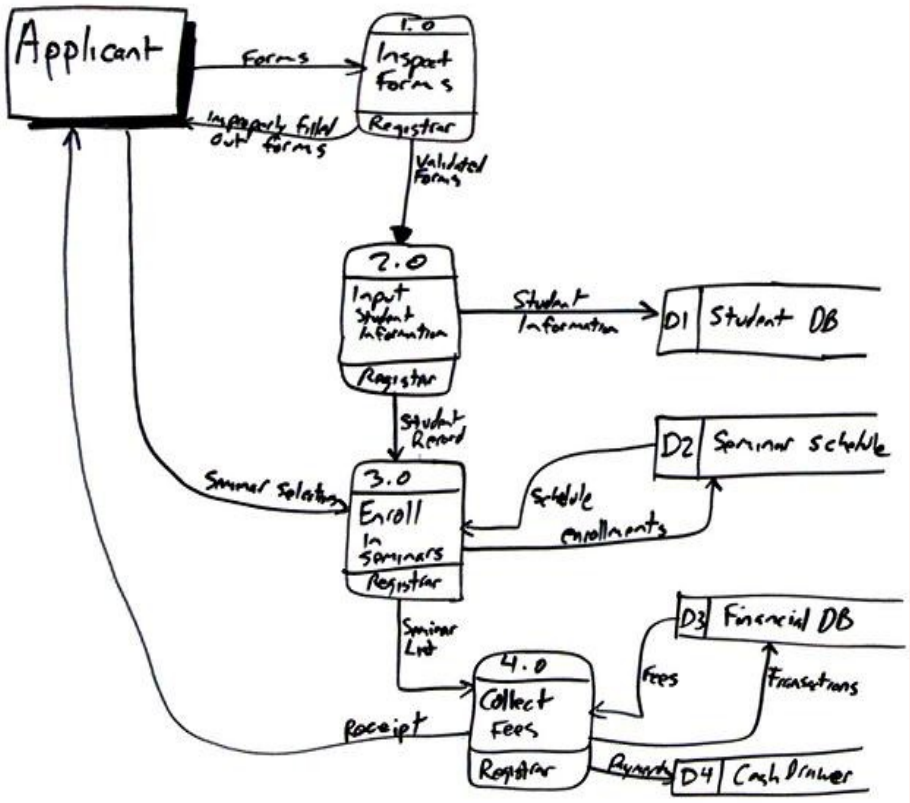
Models of computation considered in this course

Communication/ local computations	Shared memory	Message passing Synchronous | Asynchronous	
Communicating finite state machines	StateCharts		SDL
Data flow model	Not useful	(Simulink, LabView)	Kahn process networks, SDF
Von Neumann model	C, C++, Java	C, C++, Java with libraries CSP, ADA |	
Discrete event (DE) model	VHDL, Verilog, SystemC	Only experimental systems, e.g. distributed DE in Ptolemy	

Data flow as a “natural” model of applications

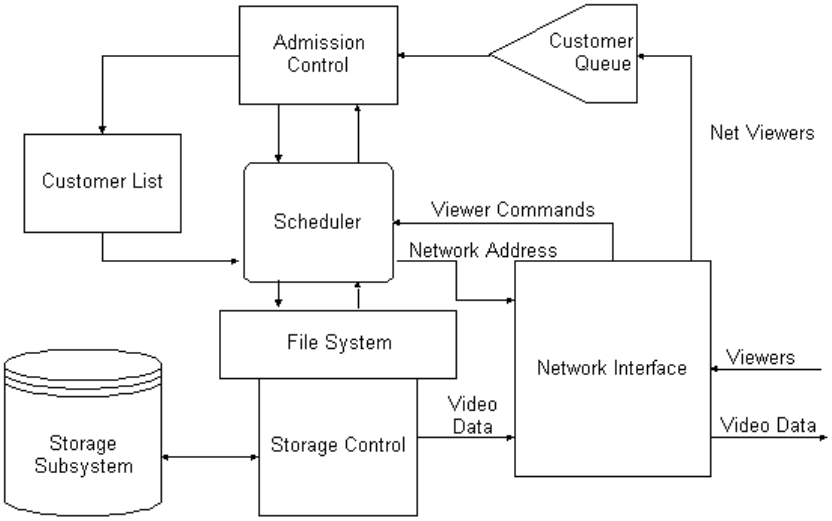
Examples

Registering for courses



http://www.agilemodeling.com/artifacts/dataFlowDiagram.htm

Video on demand system



www.ece.ubc.ca/~irenek/techpaps/vod/vod.html

Data flow modeling

Def.: The process of identifying, modeling and documenting how data moves around an information system.

Data flow modeling examines

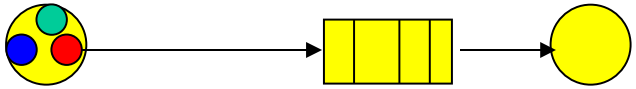
- *processes* (activities that transform data from one form to another),
- *data stores* (the holding areas for data),
- *external entities* (what sends data into a system or receives data from a system, and
- *data flows* (routes by which data can flow).

http://www.webopedia.com/TERM/D/data_flow_modeling.html

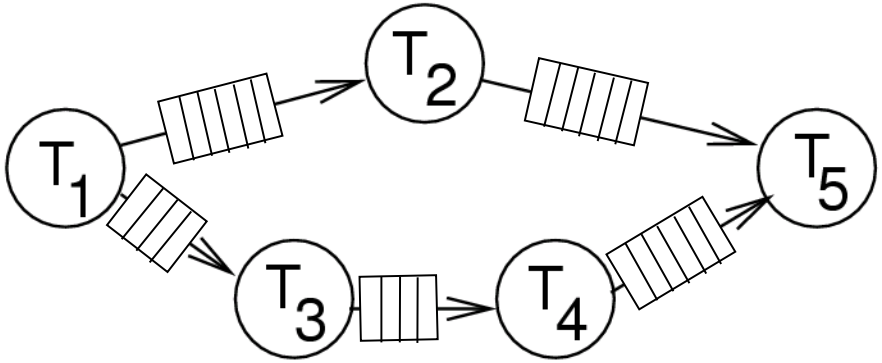
See also S. Edwards: http://www.cs.columbia.edu/~sedwards/classes/2001/w4995-02/presentations/dataflow.ppt

Reference model for data flow:
Kahn process networks

For asynchronous message passing:
communication between tasks is buffered



Special case: Kahn process networks:
executable task graphs;
Communication via infinitely large FIFOs



Properties of Kahn process networks (1)

- Each node corresponds to one program/task;
- Communication is only via channels;
- Channels include FIFOs as large as needed;
- Channels transmit information within an unpredictable but finite amount of time;
- Mapping from ≥1 input seq. to ≥1 output sequence;
- In general, execution times are unknown;
- Send operations are non-blocking, reads are blocking.
- One producer and one consumer;
  i.e. there is only one sender per channel;

Properties of Kahn process networks (2)

- There is only one sender per channel.
- A process cannot check whether data is available before attempting a read.
- A process cannot wait for data for more than one port at a time.
- Therefore, the order of reads depends only on data, not on the arrival time.
- Therefore, Kahn process networks are deterministic (!); for a given input, the result will always the same, regardless of the speed of the nodes. SDL-like conflicts at FIFOs do not exist.

This is the key beauty of KPNs!

Example

```
Process f (in int u, in int v, out int w) {
    int i; bool b = true;
    for (;;) {
        i = b ? wait (u) : wait (v); // wait returns next token in FIFO, blocks if empty
        printf("%i\n", i);
        send (i, w); // writes a token into a FIFO w/o blocking
        b = !b;
    }
}
```

© R. Gupta (UCSD), W. Wolf (Princeton), 2003

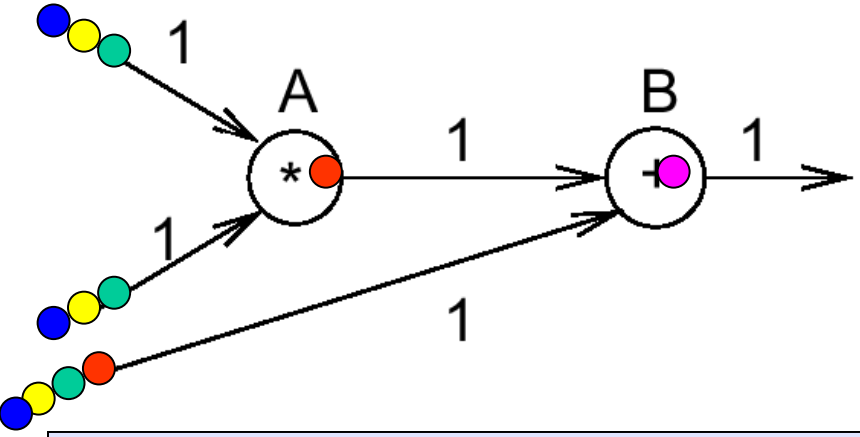
- Model of parallel computations used in practice (e.g. at Philips/NXP).
- It is a challenge to schedule KPNs without accumulating tokens

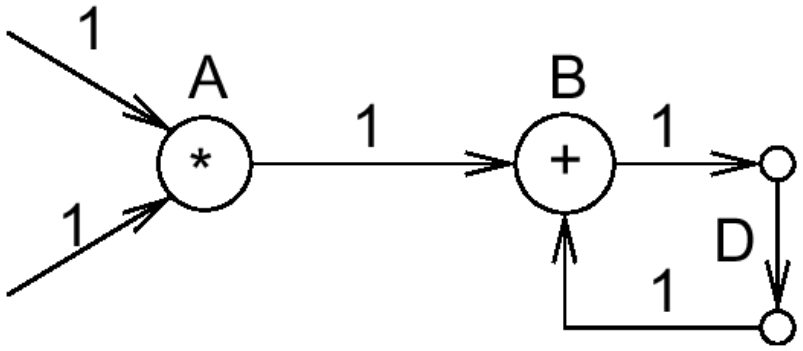
☞http://en.wikipedia.org/wiki/Kahn_process_networks

☞http://ls12-www.cs.tu-dortmund.de/edu/ES/leviKPN.zip: Animation

Asynchronous message passing: Synchronous data flow (SDF)

Asynchronous message passing=
tasks do not have to wait until output is accepted.
Synchronous data flow =
all tokens are consumed at the same time.





SDF model allows static scheduling of token production and consumption.
In the general case, buffers may be needed at edges.

Synchronous Dataflow (SDF)

Fixed Production/Consumption Rates

Balance equations (one for each channel):

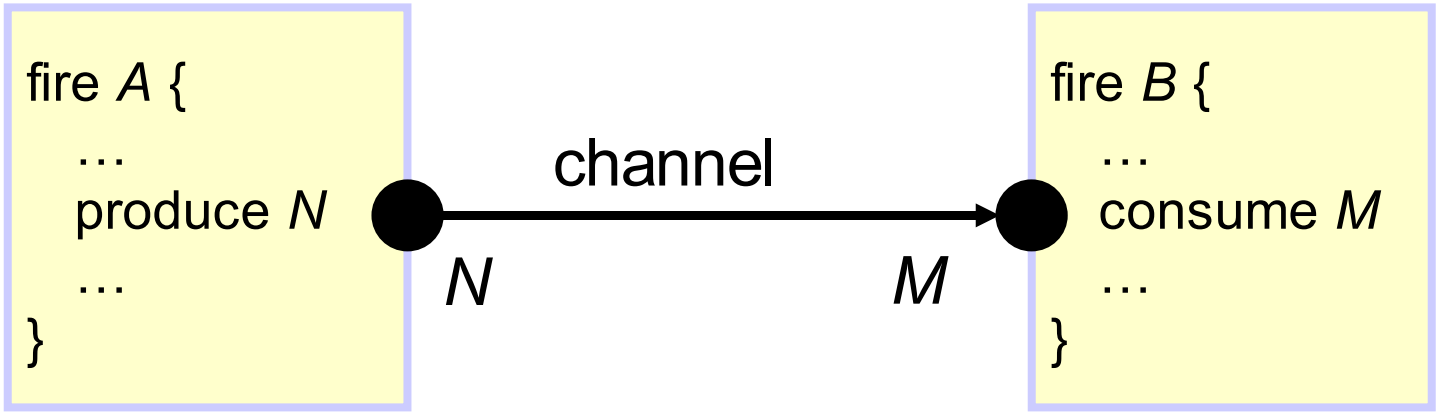
$$f_A N = f_B M$$

- f_A: number of tokens consumed
- N: number of firings per "iteration"
- f_B: number of tokens produced

Schedulable statically

Decidable:

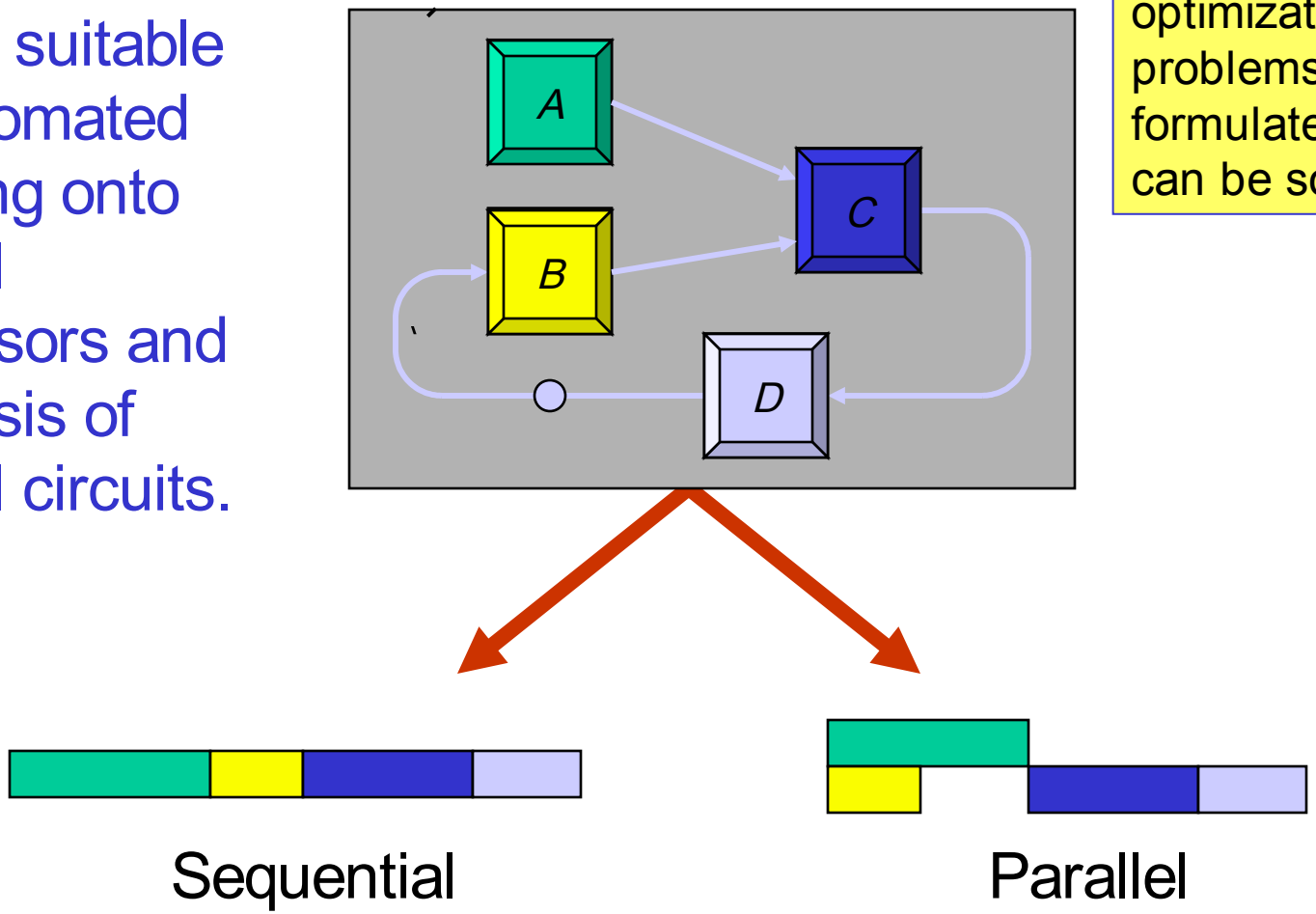
- buffer memory requirements
- deadlock



Source: ptolemy.eecs.berkeley.edu/presentations/03/streamingEAL.ppt

Parallel Scheduling of SDF Models

SDF is suitable for automated mapping onto parallel processors and synthesis of parallel circuits.

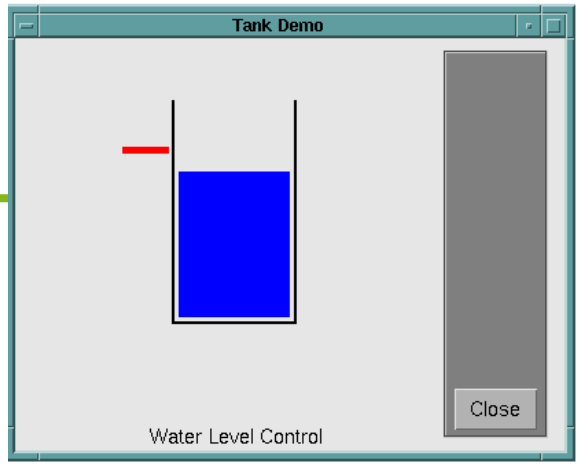


Many scheduling optimization problems can be formulated. Some can be solved, too!

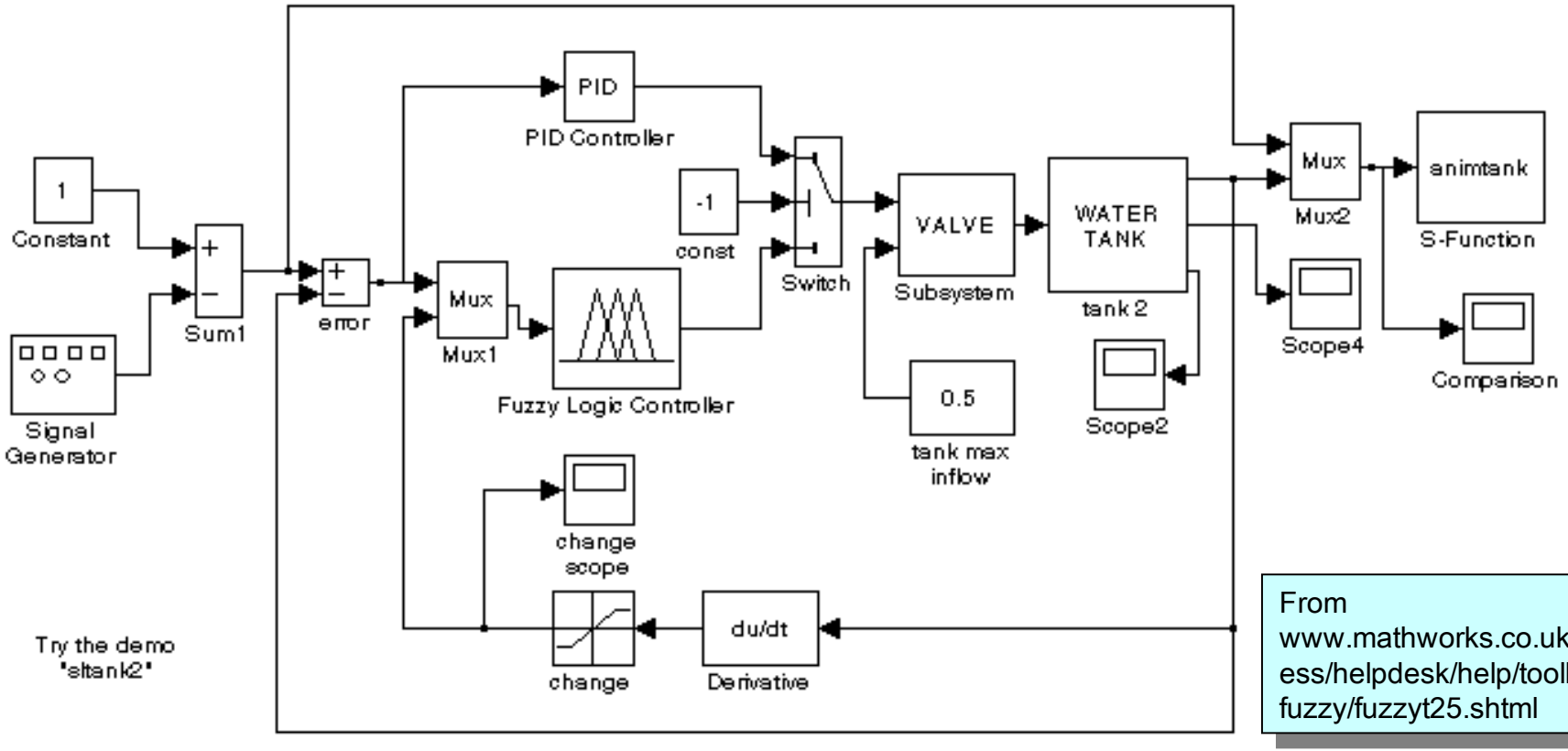
Source: ptolemy.eecs.berkeley.edu/presentations/03/streamingEAL.ppt

Similar MoC: Simulink

- example -



Simulink uses an idealized timing model for block execution and communication. Both happen infinitely fast at exact points in simulated time. Thereafter, simulated time is advanced by exact time steps. All values on edges are constant in between time steps. [Nicolae Marian, Yue Ma]

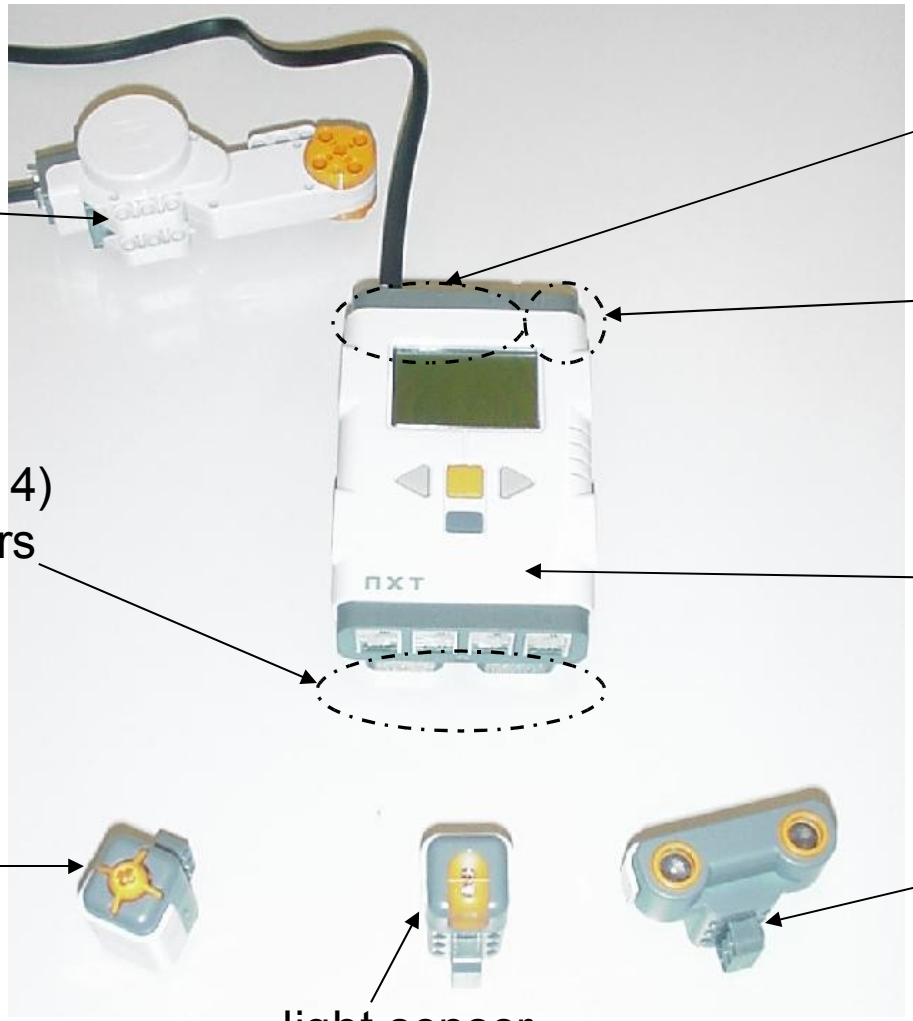


From www.mathworks.co.uk/access/helpdesk/help/toolbox/fuzzy/fuzzyt25.shtml

Data flow programming of LEGO mindstorms robots

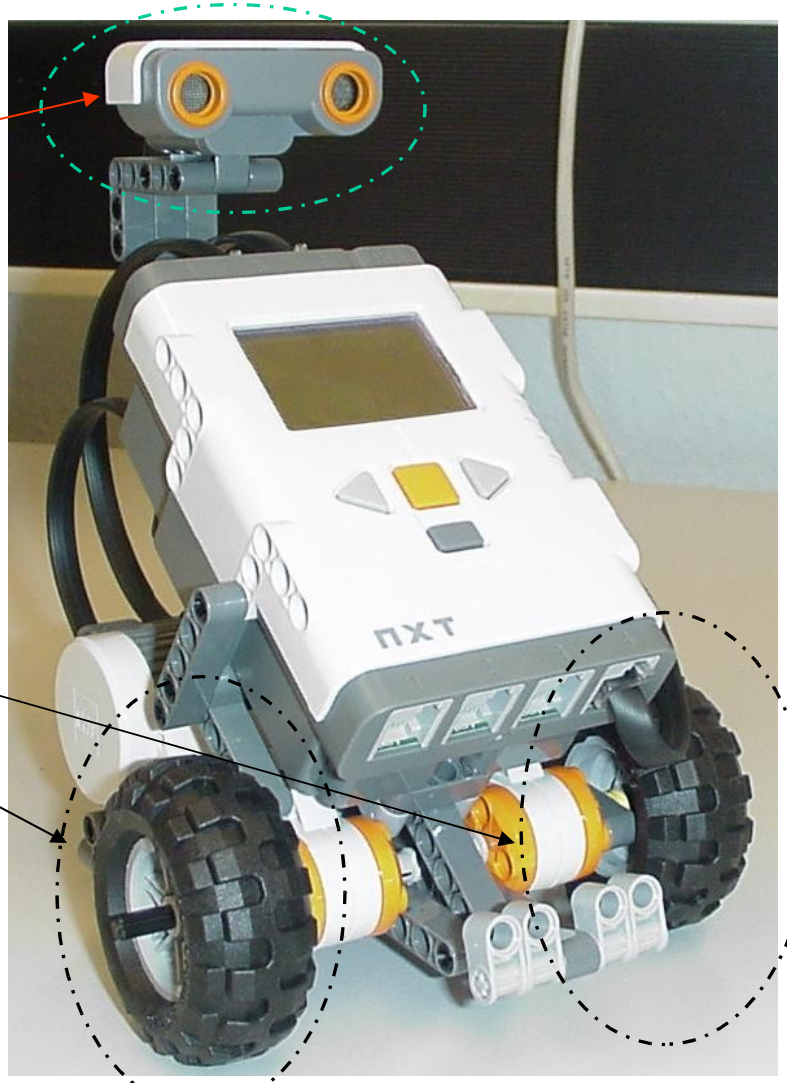
Peter Marwedel
Informatik 12
TU Dortmund
Germany

Lego Mindstorm® components



Basic robot for lab

1 ultra sonic sensor



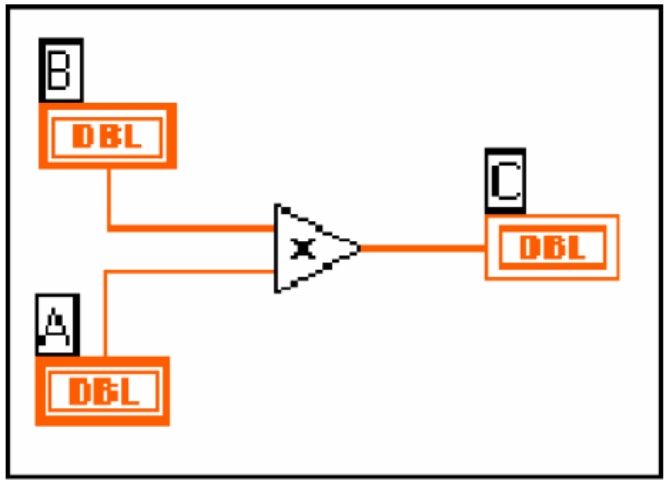
2 indepently controlled wheels

The basic robot will be extended by additional sensors and actuators during the labs

Data flow programming using LabVIEW

- LabVIEW programs = graphs
- Specification of operations and dependences
- can be executed in arbitrary sequence as long as data dependences are met
- we don‘t care about the precise sequential code needed for each of the nodes.

Example:

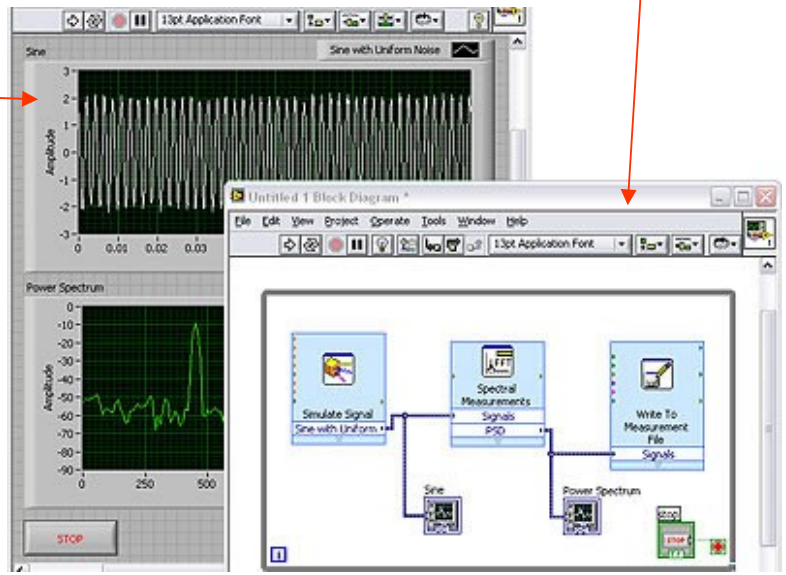


Virtual instruments

VI = virtual instrument

VIs represented in 2 windows:

- Front panel: user interface
- Block diagram: functionality of the system



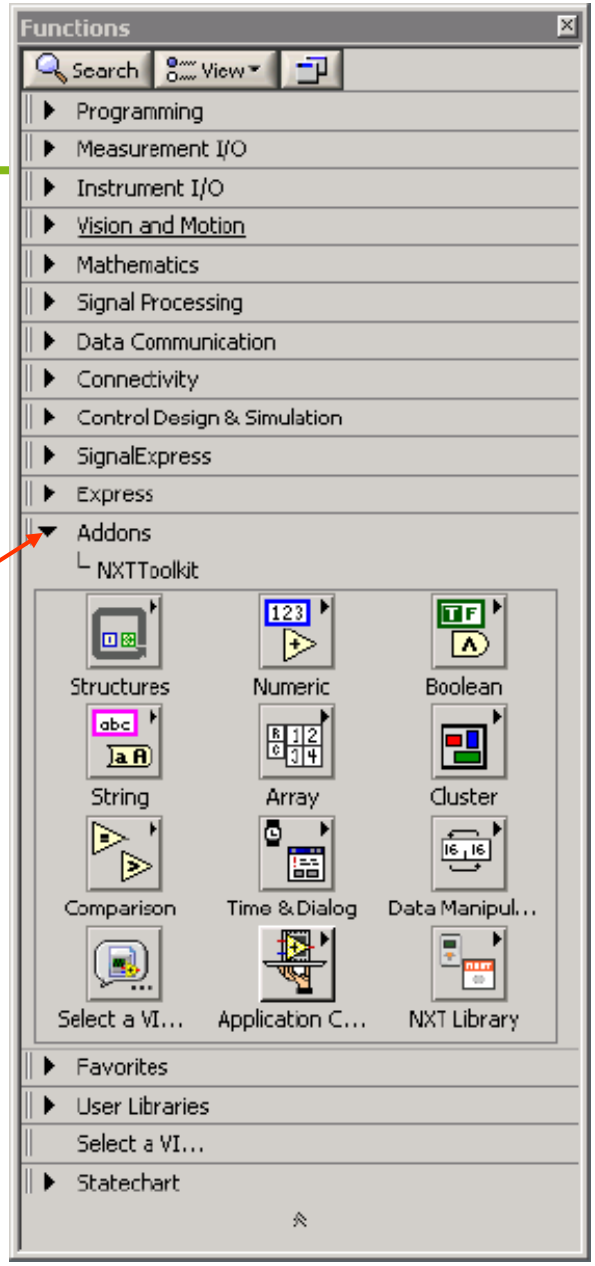
LabVIEW NXT tool kit

Front panel irrelevant, since robots have no “user interface“

Mindstorm programming:

- functions menu contains all required features as addons
- Use **only** these addons!
  The other features are not available for the mindstorms

☞ Introduction of most relevant NXT features

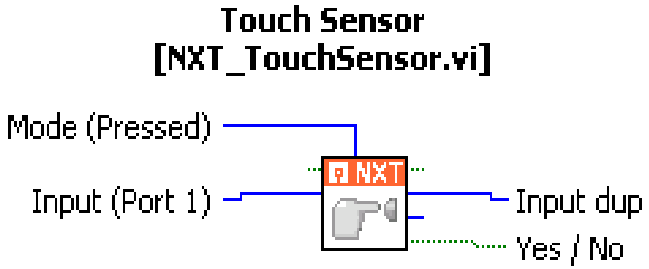


Input (1)

Menu: NXTToolkit => NXT Library => Input => Touch Sensor / Sound Sensor

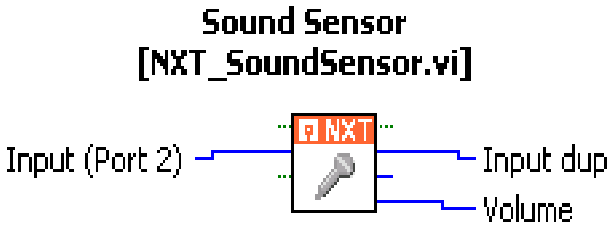
Touch sensor

- reads in sensor from designated input port
- Mode: pressed/released
- Output: yes/no



Sound sensor

- reads in sensor from designated input port
- Output: volume

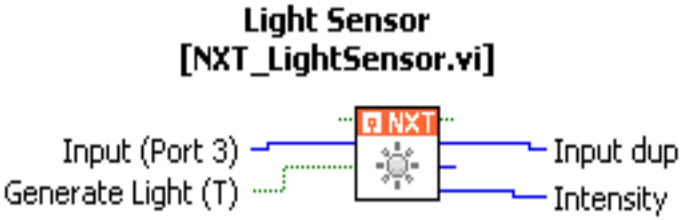


Input (2)

Menu: NXTToolkit => NXT Library => Input => Light Sensor / Ultrasonic Sensor

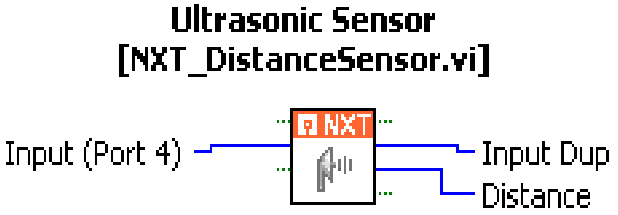
Light sensor

- reads in sensor from designated input port
- sensor must be switched on (Generate Light => True)
- Output: intensity



Ultra sonic sensor:

- reads sensor from designated input port
- Output: distance

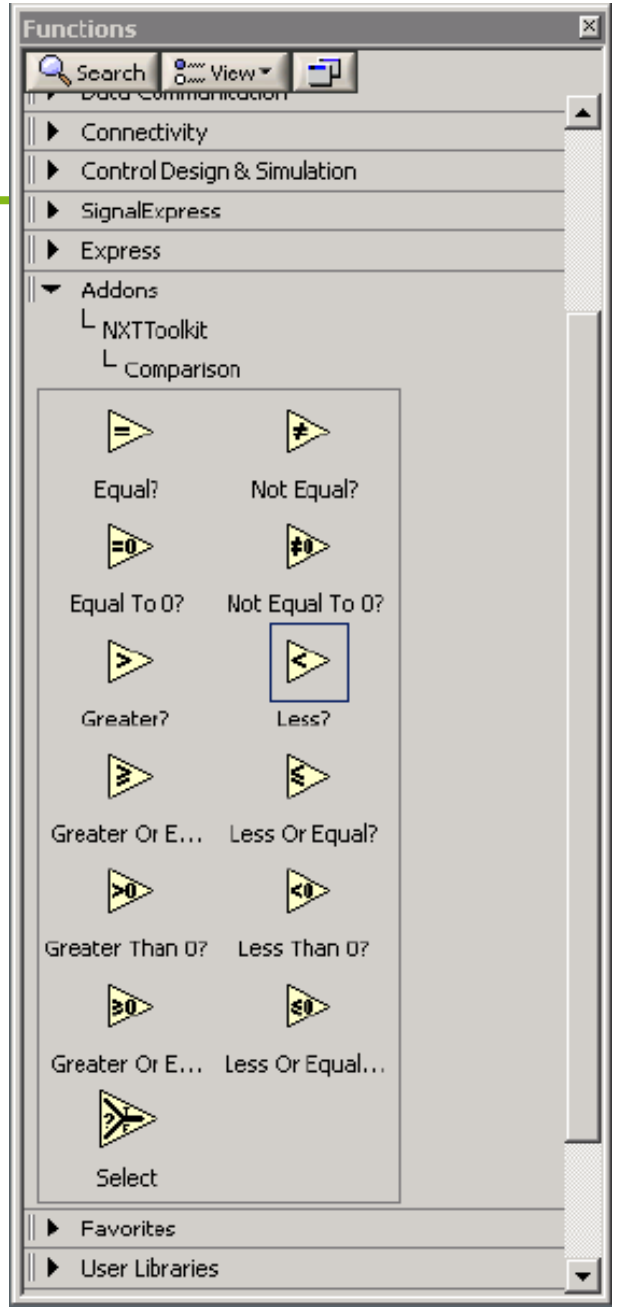


Comparison

Menu: NXTToolkit => Comparison

- Essentially self-explaining
- Result: Boolean
- Exception: Select

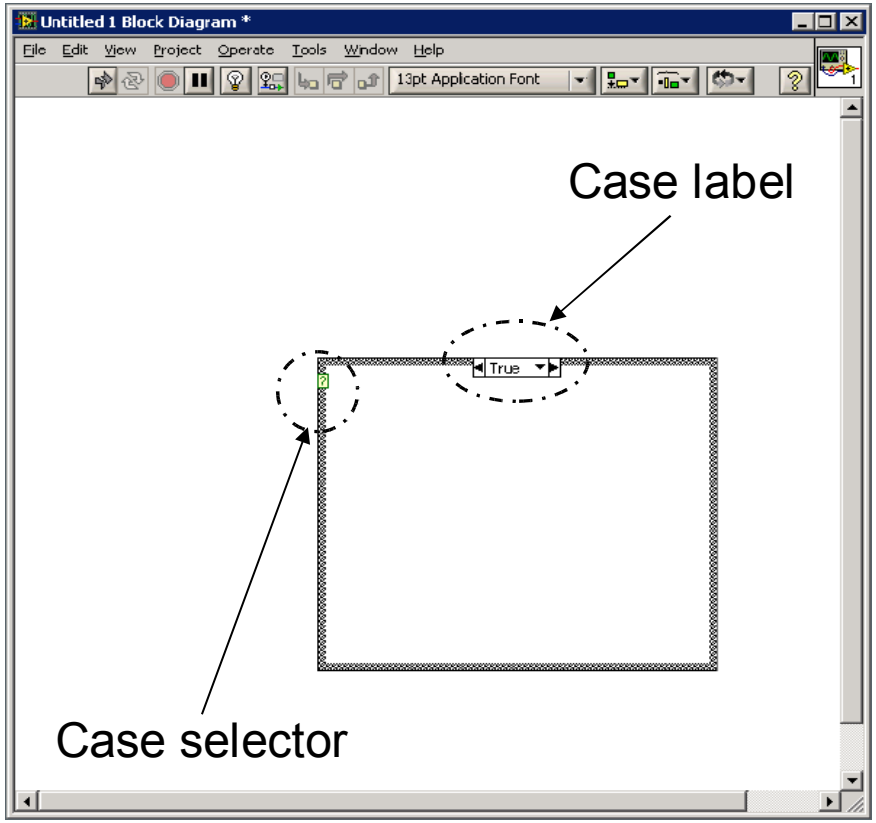
  ≈ 2 → 1 Mux



Case-dependent data flow

Menu: NXTToolkit => Structures => Case Structure
Move from Functions-Palette into editing area using drag & drop

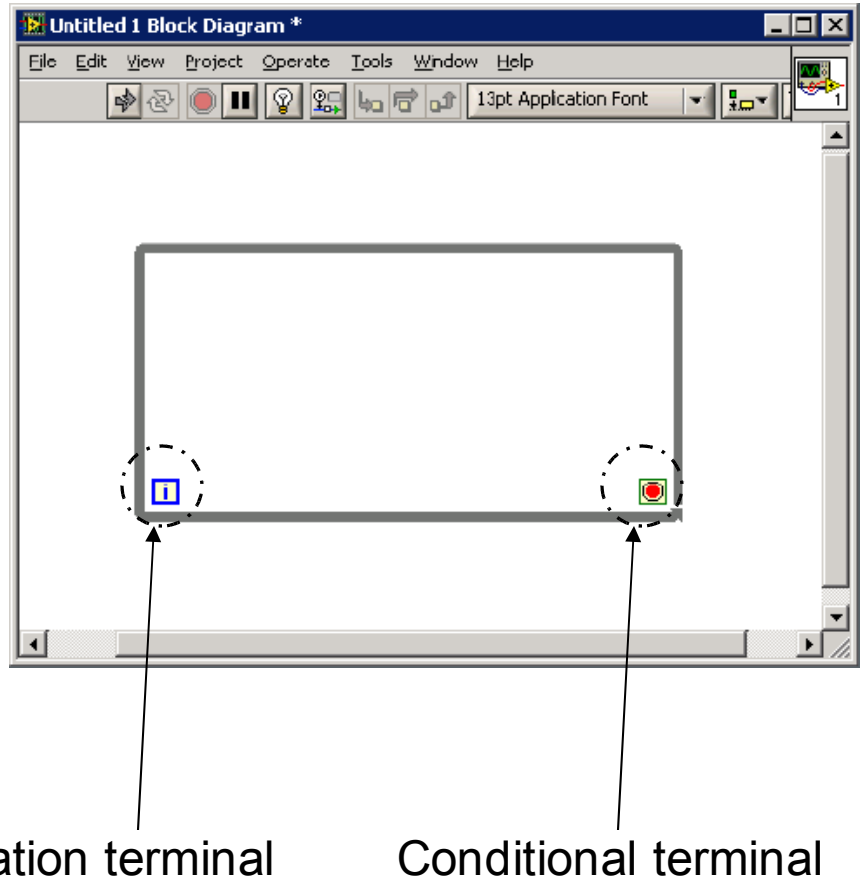
- Consists of several sub diagrams, only one of which can be active
- Click on the arrows next to the case label to display a particular sub diagram.
- The case selector serves as the input to the case structure;
  Possible data types: Bool, String, Integer.
- Action to be performed designated by additional elements within the case structure
- Right click opens context menu



Data flow loops

Menu: NXTToolkit => Structures => While Loop

- Sub diagram will be repeated until Boolean condition is true
- Condition is represented by conditional terminal
- Right allows selecting whether iterations will stop or continue if condition is true
- The iteration terminal includes the number of the actually executed iteration

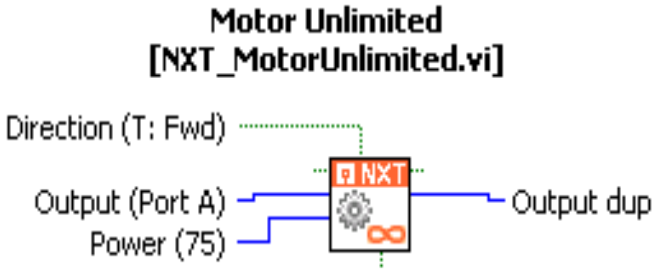


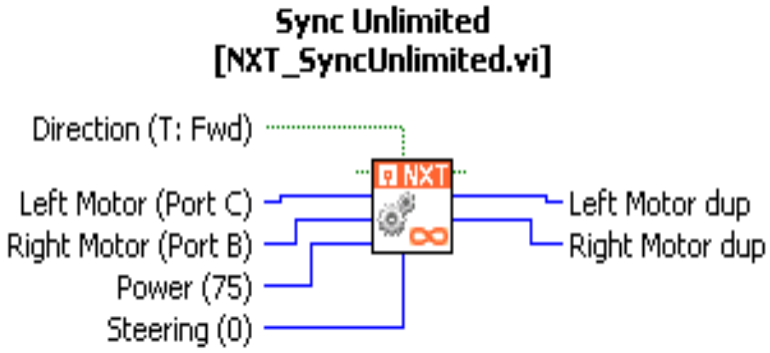
Output (1)

Menu: NXTToolkit => NXT Library => Output => Motor Unlimited / Sync Unlimited

Motor control (unlimited)

- controlling one motor
  - Designate output port
  - Direction (true = forward)
  - Velocity
- controlling both motors
  - Designate output ports
  - Direction
  - Velocity
  - Relative speed (direction): -100 (left) … 100 (right)





Output (2)

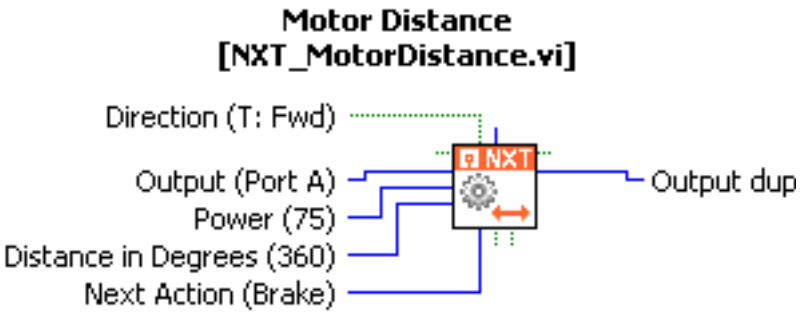
Menu: NXTToolkit => NXT Library =>
Output => Motor Distance / Motor Time

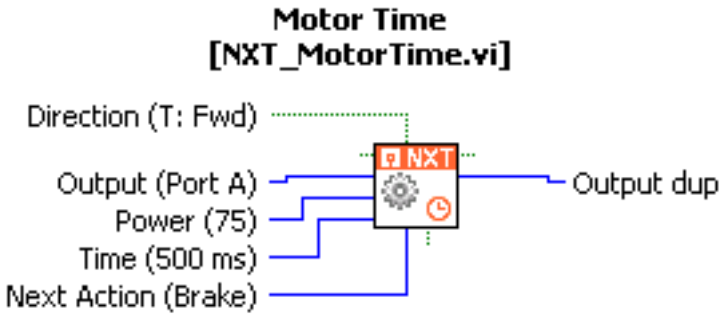
Motor control (distance)

- controlling one motor
  - Designate output port
  - Direction
  - Velocity
  - Distance in degrees
  - Follow-up action (braking, free running)

Motor control (time)

- controlling one motor
  - Distance indicated via time
  - Otherwise, same as above



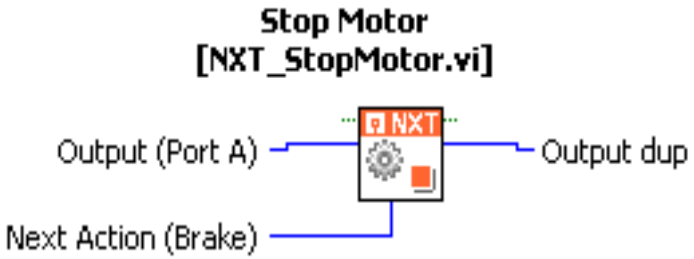


Output (3)

Menu: NXTToolkit => NXT Library => Output => Stop Motor / Legacy Lamp

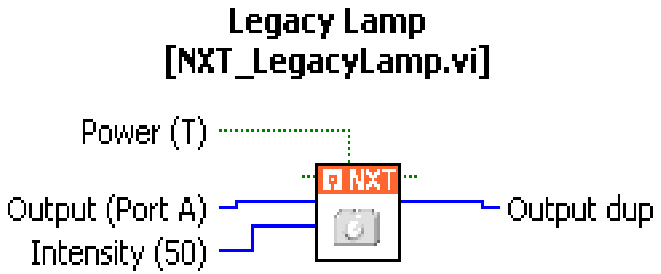
Motor control (stop)

- controlling one motor
  - Designate port
  - Stop motor thrust
- there is a variant controlling both motors



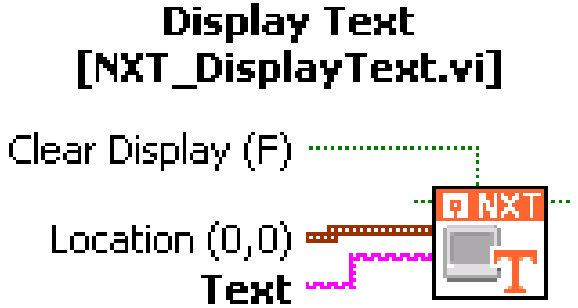
Lamp

- Designate output port
- indicate intensity
- switch on explicitly



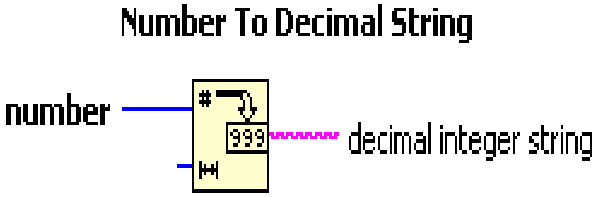
Display

Menu: NXTToolkit => NXT Library => Display => Display Text



Display

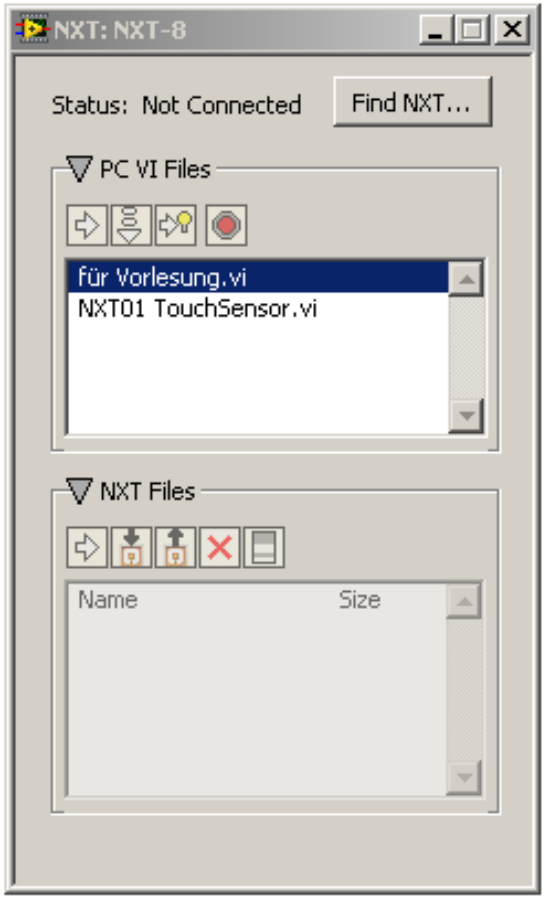
- display line of text on robot‘s display
- displays only strings
- numbers must be converted into strings:
  N*XTToolkit* => *String* => *String/Number Conversion* => *Number to Decimal*



Downloading software

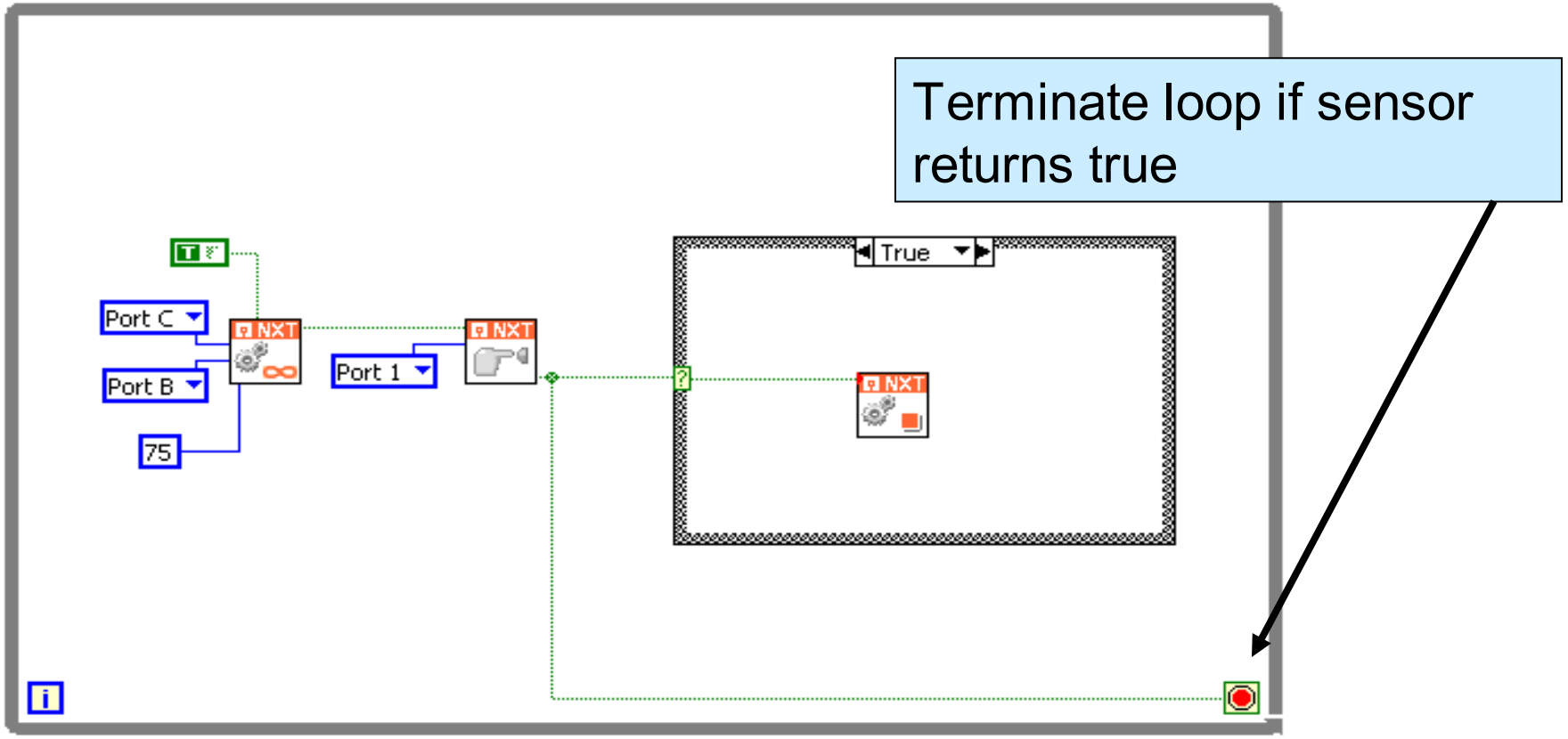
Menu: Tools => NXT Module => NXT Terminal

- Download either via USB or Bluetooth
- Terminal window
  - Find robot
  - compile + download
  - upper window: files on PC
  - lower window: files on robot



Small example

Goal: robot moves forward until collision sensor detects a collision

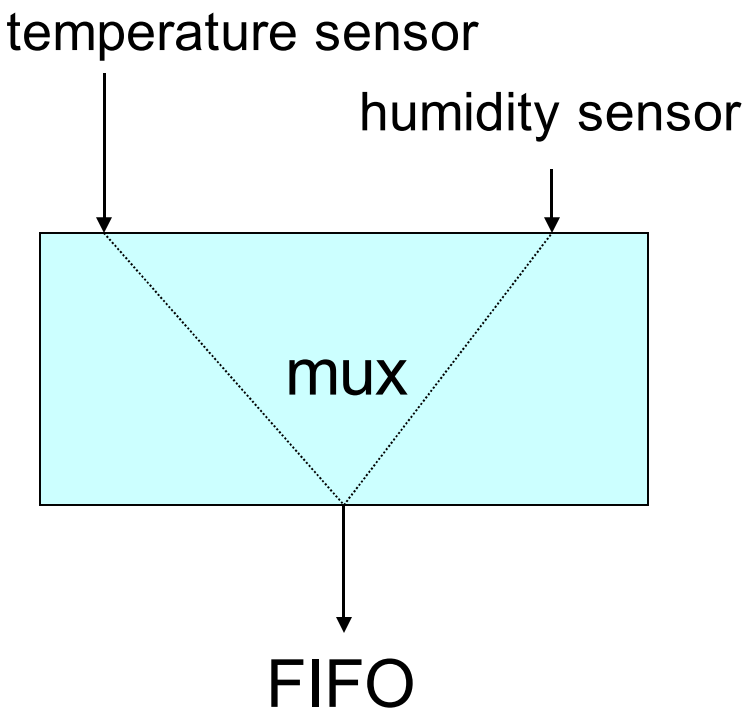


From data flow to task graphs

- "Lunatics" (S. Edwards) tried to build general purpose data flow computers, but failed
- However, modern computers contain subsystems build on the data flow paradigm (scoreboard, Tomasulo's algorithm)
- Pure data flow frequently too restrictive: lack of modeling control, resources etc.
- Data flow graphs are a special case of the somewhat more general task graphs.
- ☞

Task graphs

Many applications can be specified in the form of a set of communicating processes.
Example: system with two sensors:



Alternating read

```
loop
 read_temp; read_humi
until false;
```

of the two sensors no the right approach.

The case for multi-process modeling

```
MODULE main;
 TYPE some_channel =
         (temperature, humidity);
      some_sample : RECORD
                     value : integer;
                     line  : some_channel
                    END;
 PROCESS get_temperature;
 VAR sample : some_sample;
 BEGIN
  LOOP
   sample.value := new_temperature;
   IF sample.value > 30 THEN ....
   sample.line := temperature;
   to_fifo(sample);
  END
 END get_temperature;
```

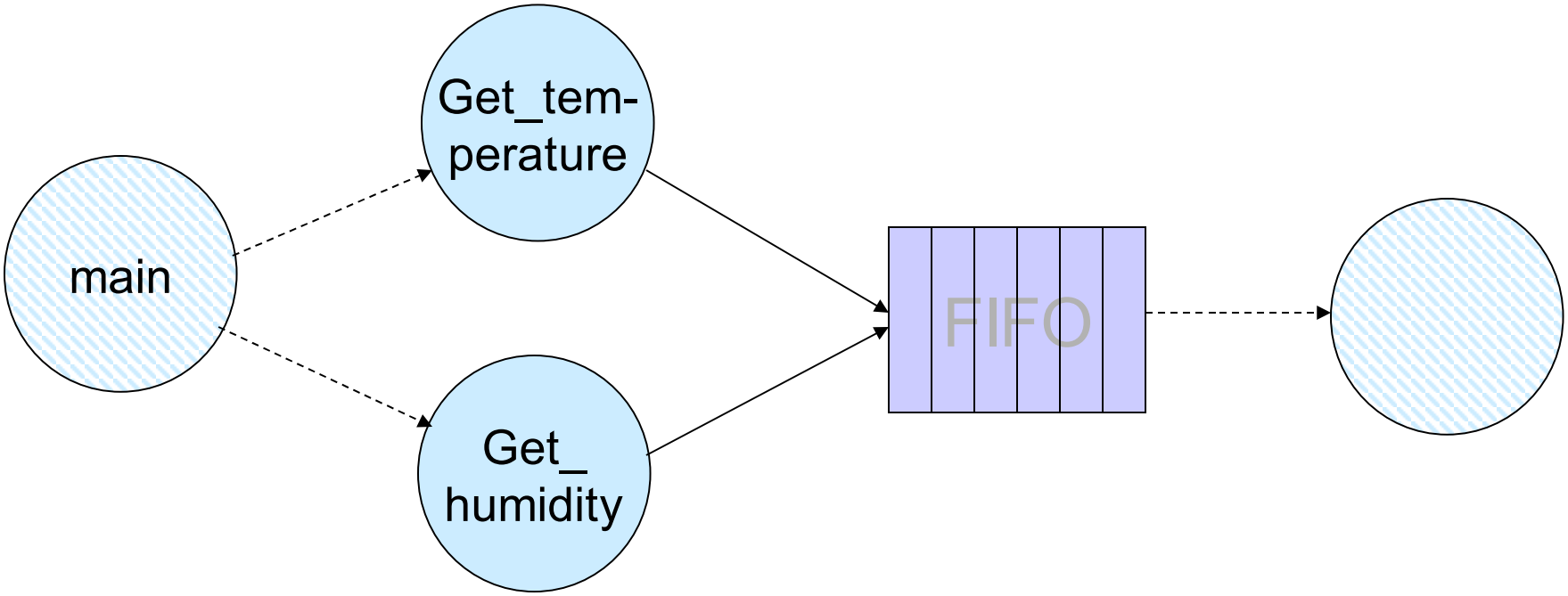
```
PROCESS get_humidity;
  VAR sample : some_sample;
  BEGIN
   LOOP
    sample.value := new_humidity;
    sample.line := humidity;
    to_fifo(sample);
   END
  END  get_humidity;

 BEGIN
   get_temperature; get_humidity;
 END;
```

- Blocking calls new_temperature, new_humidity
- **Structure clearer than alternating checks for new values in a single process**

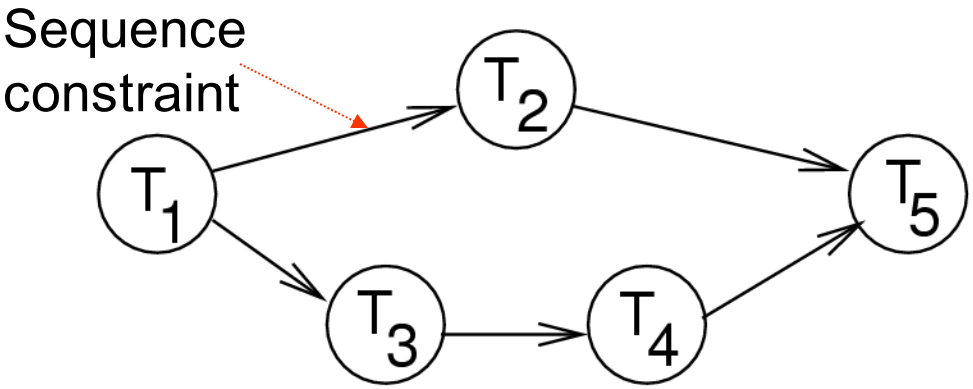
How to model dependencies between tasks/processes?

Dependences between processes/tasks



☞General discussion of process networks

Task graphs

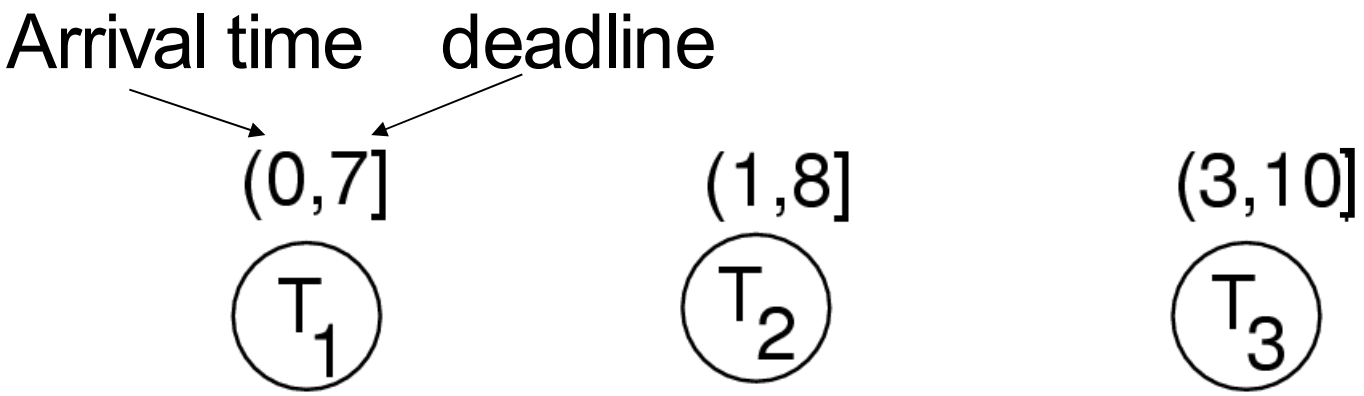


Nodes are assumed to be a “program“ described in some programming language, e.g. C or Java.

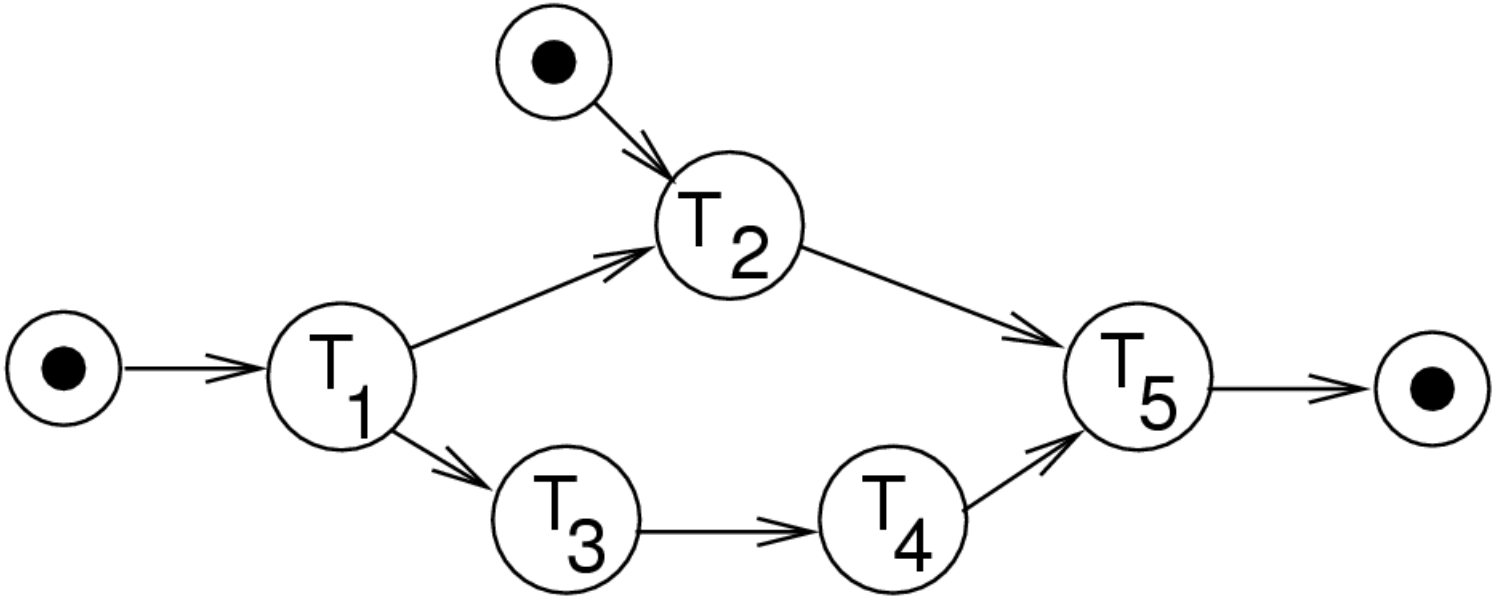
Def.: A **dependence graph** is a directed graph $G=(V,E)$ in which $E \subseteq V \times V$ is a partial order.
If $(v1, v2) \in E$, then *v1* is called an **immediate predecessor** of *v2* and *v2* is called an **immediate successor** of *v1*.
Suppose E^* is the transitive closure of E.
If $(v1, v2) \in E^*$, then *v1* is called a **predecessor** of *v2* and *v2* is called a **successor** of *v1*.

Task graphs
1. Timing information -

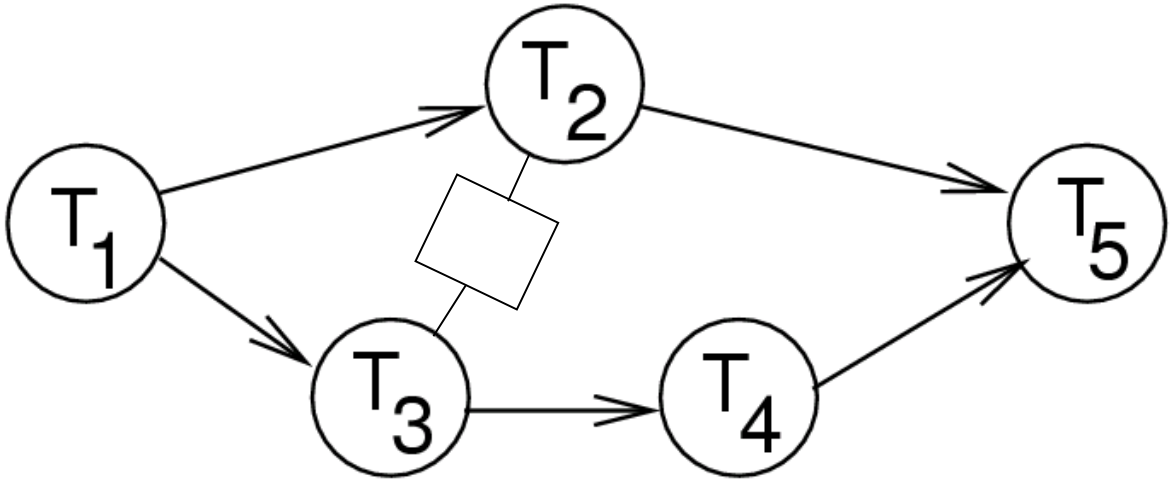
Task graphs may contain additional information,
for example: Timing information

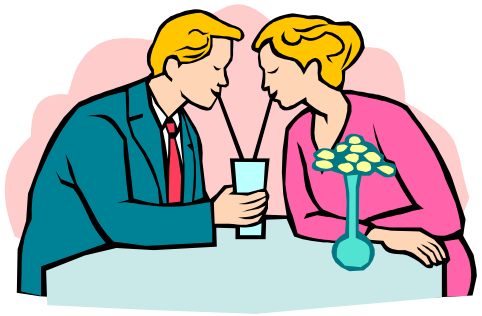


Task graphs
2. I/O-information

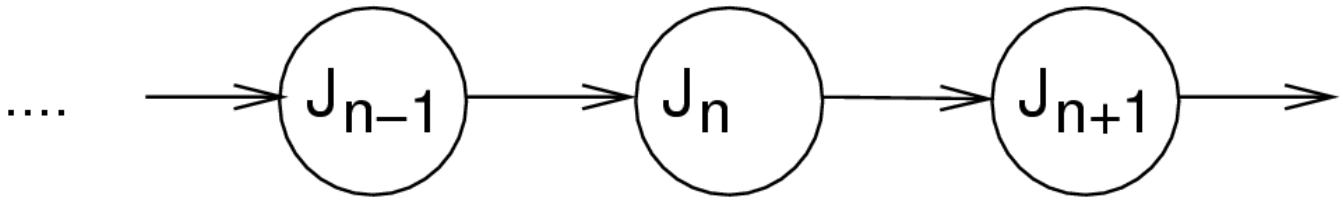


Task graphs
3. Shared resources





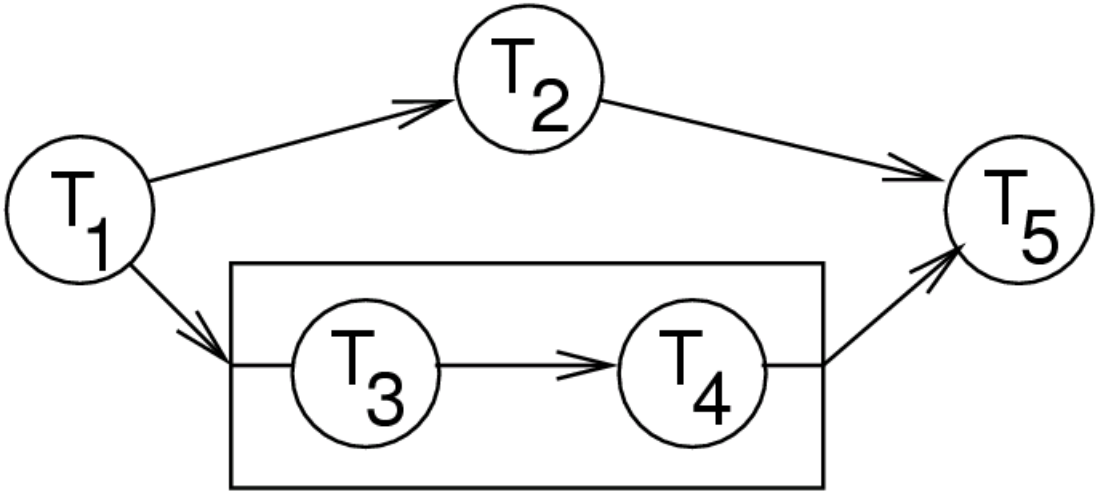
Task graphs
4. Periodic schedules



.. infinite task graphs

Each instance called a “job”

Task graphs
5. Hierarchical task graphs -

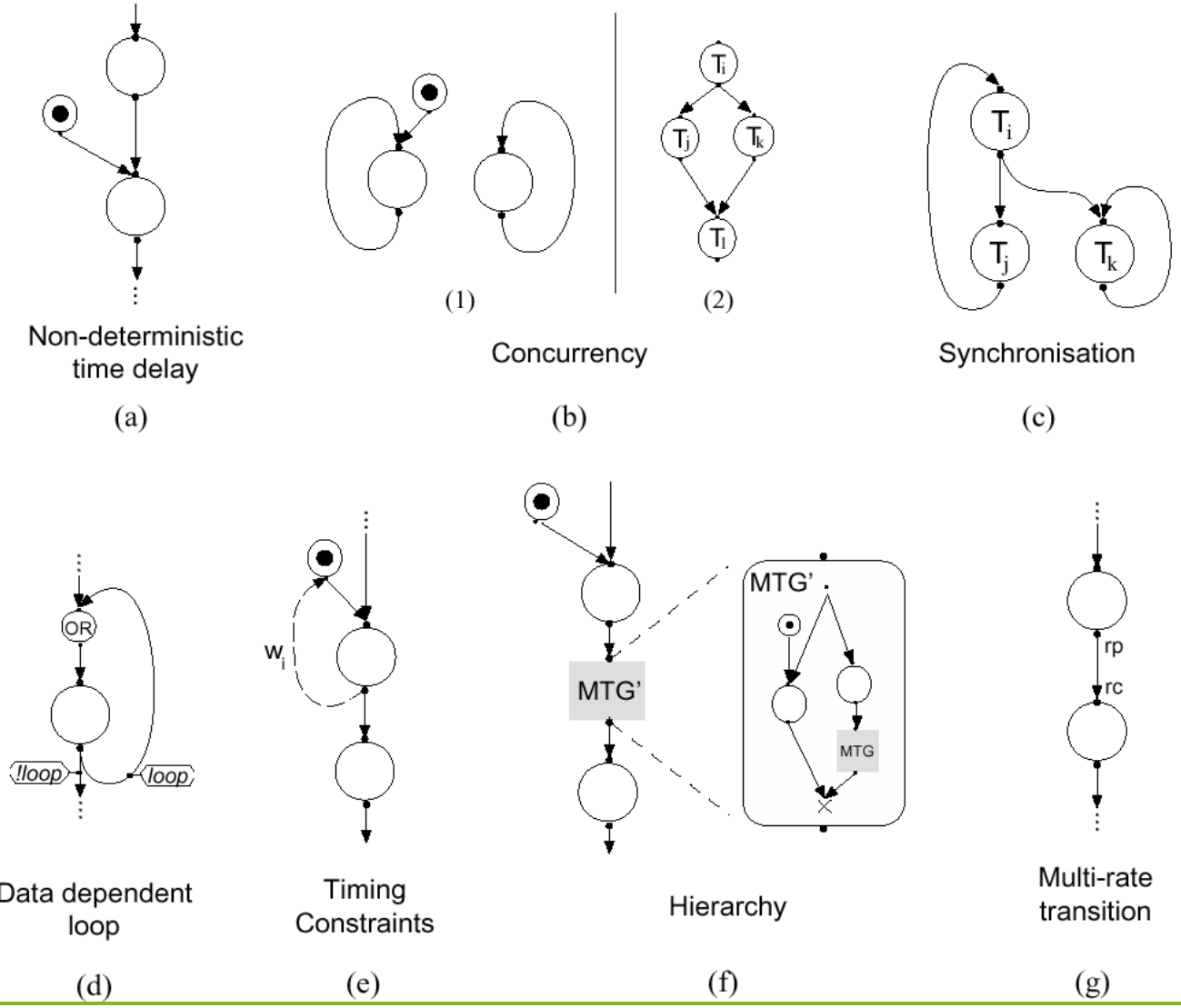


Multi-thread graphs (IMEC)

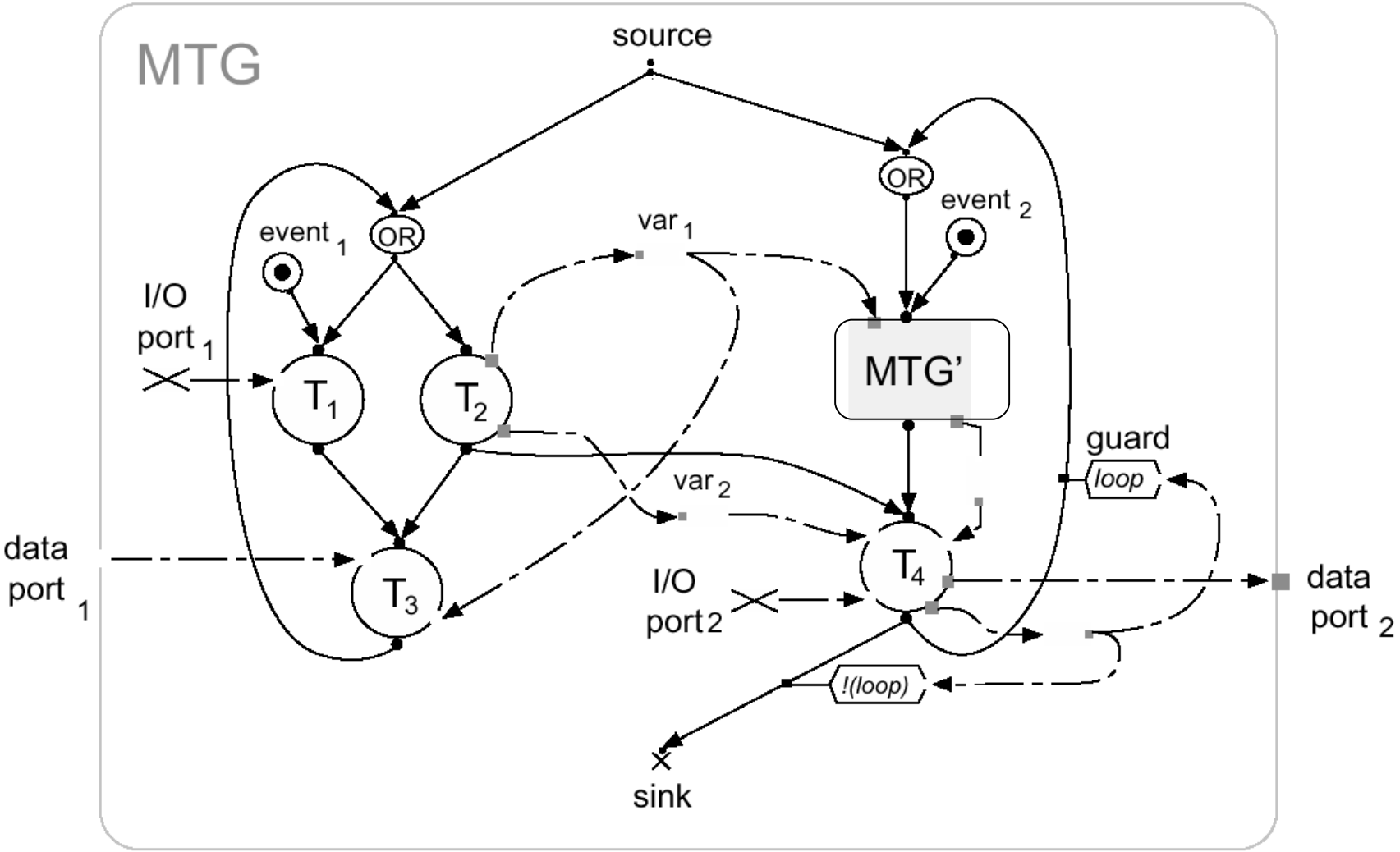
Def.: A multi-thread graph *M* is defined as an 11-tuple ($O, E, V, D, \vartheta, \iota, \Lambda, E^{lat}, E^{resp}, \nabla^{i}, \nabla^{av}$) with:

- O: set of operation nodes,
- E: set of control edges,
- V, D : refer to the access of variables,
- ι: is the set of input/output nodes,
- Λ: associates execution latency intervals with all threads,
- E^{lat}, E^{resp}, ∇^{i}, ∇^{av} are timing constraints.

MTG graphs: graphical notation



Multi-thread graph (Example)



Summary

Data flow model of computation

- Motivation
- Kahn process networks
- SDF
- Visual programming languages
  - Simulink
  - Labview programming of mindstorm NXT robots
- Task graphs
  - Multi thread graph (IMEC)