Norges miljø- og biovitenskapelige universitet

Institutt for datavitenskap

Digitalisering på Ås

INF205
Resource-efficient programming

4 Concurrency

4.1 Parallel programming
4.2 Message passing interface
4.3 Domain decomposition

4.4 Robotics middleware
4.5 Concurrency theory
4.6 Parallel process models

Weekly glossary concepts

Norwegian University of Life Sciences

What are essential concepts from the previous lecture?

Let us include them in the **INF205 glossary**.[1]

embarrassing parallelism

???

???

domain decomposition

state space

???

[1]https://home.bawue.de/~horsch/teaching/inf205/glossary-en.html

Noregs miljø- og biovitskaplege universitet

Institutt for datavitskap

Digitalisering på Ås

4 Concurrency

4.1 Parallel programming

4.2 Message passing

4.3 Domain decomposition

4.4 Robotics middleware

Norwegian University of Life Sciences

ROS message passing paradigm

MPI follows the **SPMD** approach ("single program, multiple data"), whether it is SIMD ("single instruction") or MIMD ("multiple instruction") parallelization.

In **ROS**, it is **MPMD** and therefore MIMD:
Different processes (nodes) have their own codes and binary executables.

Communication in ROS can be categorized as follows:
Topic:
- **Asynchronous *n*-to-*n*** communication channel
- Publisher nodes can **publish** to the topic, all **subscriber** nodes can read

Service:
- **Synchronous one-to-one** communication
- One node **requests** another node and waits until the response comes

Action:
- **Asynchronous request** from one node to another node

ROS message passing paradigm

ROS calls its parallel processes **nodes** (do not need to be separate machines).
Communication scheme as summarized in the ROS 2 paper:[1]

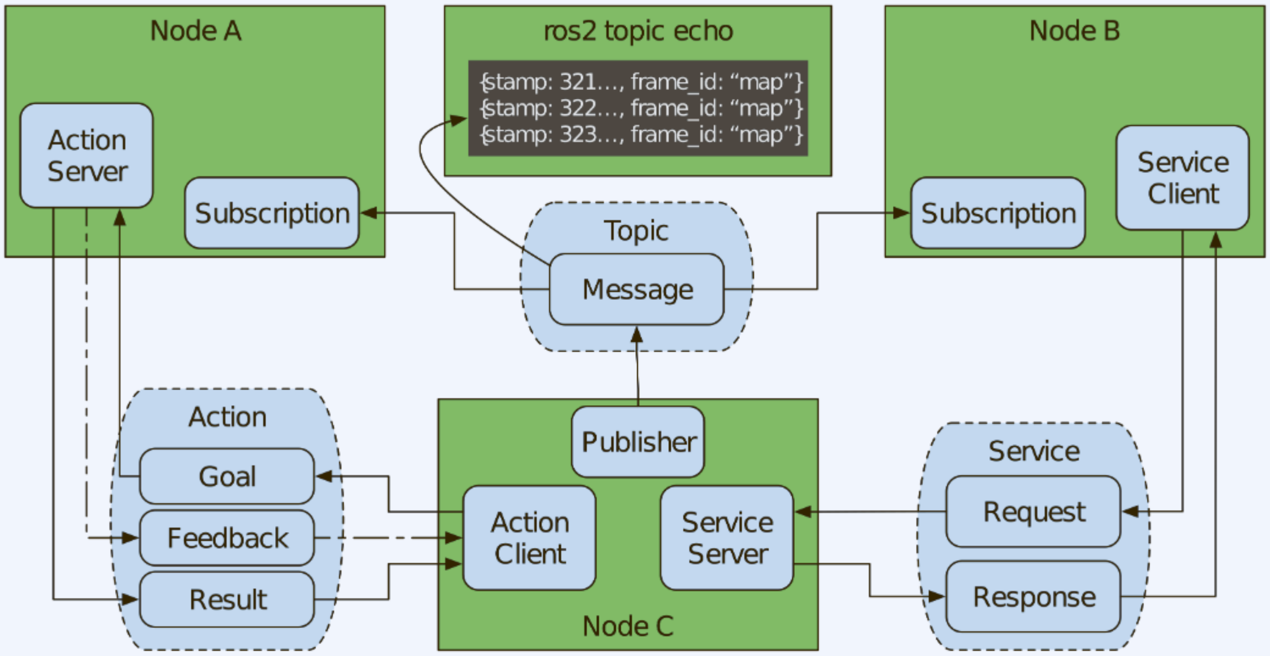


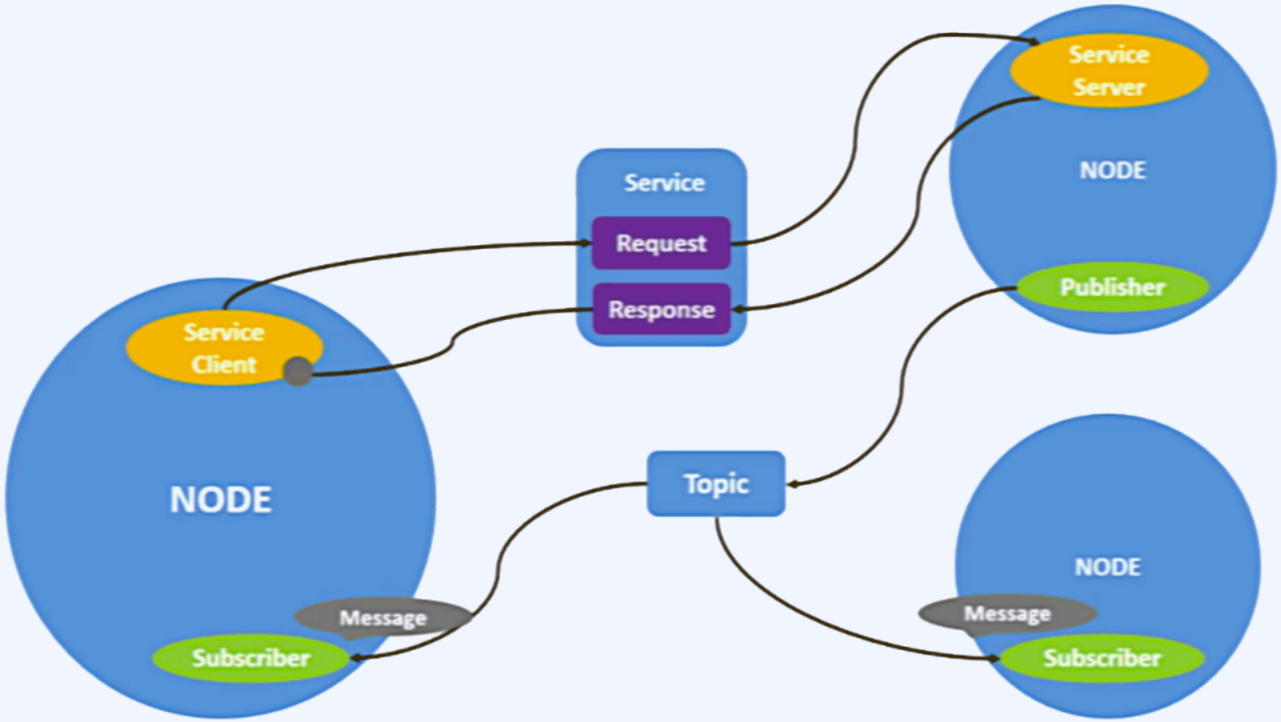
Fig. 1: ROS 2 node interfaces: topics, services, and actions.

[1]S. Macenski *et al.*, *Science Robotics* **7**(66): 2, **2022**. Open access preprint: arXiv:2211.07752 [cs.RO]

ROS message passing paradigm

ROS calls its parallel processes **nodes** (do not need to be separate machines).

In a **ROS 2 communication graph**, nodes and communication patterns are connected by edges that describe the direction of the data flow:



Source: https://docs.ros.org/en/rolling/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Nodes/Understanding-ROS2-Nodes.html

What has ROS been designed for?

Despite the name, ROS (“robot operating system”) is *not an operating system*. It is a library that provides a **middleware**, algorithms, and developer tools.

Requirements addressed by ROS 2 (see the ROS 2 paper[1] for more detail):

Distribution: In a distributed system, there are **no single points of failure**. With ROS 2, you *can* do distributed programming, with MPI you *cannot*.

Asynchrony: “messages [...] are communicated asynchronously, creating an **event-based system**” (as discussed in Sections “4.5” and “4.6” of this lecture).

Embedded systems: For robotics applications that include “small embedded devices,” there is a special implementation called **micro-ROS**: ROS 2 for microcontrollers.[2]

[1]S. Macenski *et al.*, *Science Robotics* **7**(66): 2, **2022**. Open access preprint: arXiv:2211.07752 [cs.RO].

[2]https://micro.ros.org/

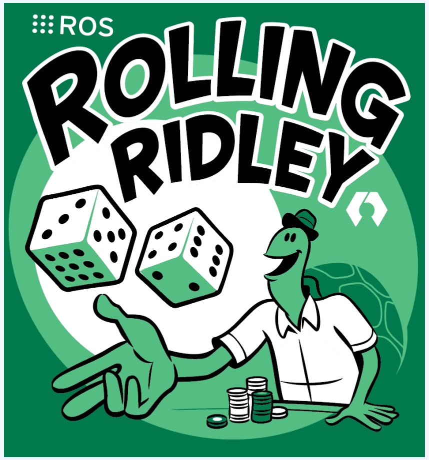
ROS 2 installation

Documentation: http://docs.ros.org/

Active ROS 2 distributions:

recommended for **robot development**

latest stable version



development version (“will at times include breaking changes”)

Installation by adding http://packages.ros.org/ros2/ubuntu repository to apt.

The standard procedure for compiling code that uses ROS 2 requires **cmake**.

ROS 2 installation (Ubuntu-like system)

http://docs.ros.org/en/rolling/Installation/Ubuntu-Install-Debians.html

Packages to be installed:

- ros-base
- ros-desktop

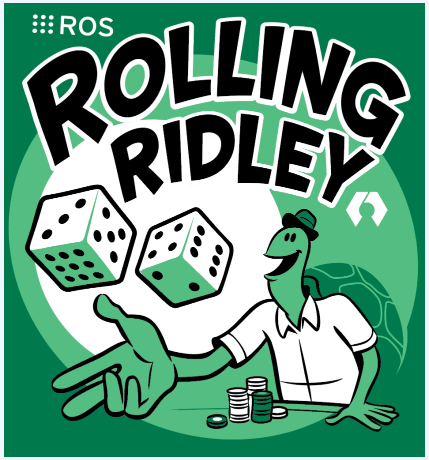
Bash script to be loaded at each use:

source /opt/ros/rolling/setup.bash

Simple test using two default nodes:

ros2 run demo_nodes_cpp talker

ros2 run demo_nodes_cpp listener



development version (“will at times include breaking changes”)

Installation by adding http://packages.ros.org/ros2/ubuntu repository to apt.

The standard procedure for compiling code that uses ROS 2 requires **cmake**.

ROS 2 package creation

A ROS2 C++ **package** for compilation supported by **cmake** can be created by

e.g. **cpp_srvcli**

ros2 pkg create --build-type ament_cmake *prjname* --dependencies rclcpp …

… for the example,[1] add **example_interfaces** here

This creates a **package XML file** and an input file for cmake.

XSD metadata schema http://download.ros.org/schema/package_format3.xsd

```
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd"
                              schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>prjname</name>
  …
  <license>CC BY-NC-SA</license>
  <buildtool_depend>ament_cmake</buildtool_depend>
  <depend>rclcpp</depend>
   …     example:1 <depend>example_interfaces</depend>
</package>
```

package.xml

[1]http://docs.ros.org/en/rolling/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Cpp-Service-And-Client.html

Material: **ros-nodes-howto.zip**.

Service example[1-3]

Node acting as a server

```
shared_ptr<Node> node
  = Node::make_shared("server_name");
node->create_service<...>(
  "service_name", &fct
);
```

Node acting as a client

```
shared_ptr<Node> node
  = Node::make_shared("client_name");
auto client
  = node->create_client<...>("service_name");
// ... create request ...
auto result = client->async_send_request(request);
```

CMakeLists.txt

```
add_executable(
  server src/add_two_ints_server.cpp
)
ament_target_dependencies(
  server rclcpp example_interfaces
)

add_executable(
  client src/add_two_ints_client.cpp
)
ament_target_dependencies(
  client rclcpp example_interfaces
)

install(
  TARGETS server client
  DESTINATION lib/${PROJECT_NAME}
)
```

[1]http://docs.ros.org/en/rolling/p/rclcpp/generated/

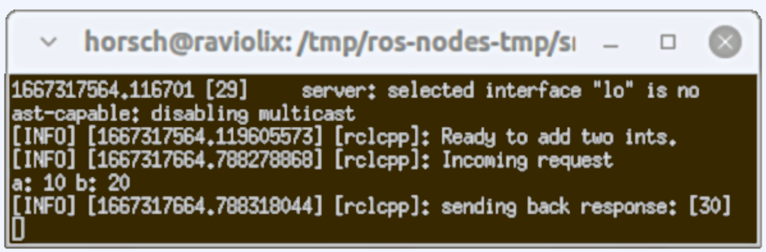
[2]https://docs.ros.org/en/foxy/Tutorials/Intermediate/Writing-an-Action-Server-Client/Cpp.html

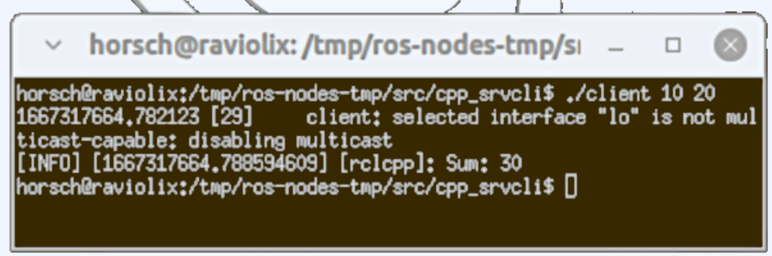
[3]http://docs.ros.org/en/rolling/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Cpp-Service-And-Client.html

Example[1]

How to test the **ros-nodes-example**:

- Compile the client and server codes using "colcon" (which calls cmake).
  - You may need to install cmake first.
- Run "server" on one terminal (or one computer in the network).
- Run "client x y" on another.
- They should interact, and the addition x+y should be performed.





Disclaimer: If you use ROS 2 for your work and it leads to a publication (or master thesis), include a citation to the reference S. Macenski *et al.*, *Science Robotics* **7**(66): eabm6074, doi:10.1126/scirobotics.abm6074, **2022**.

[1] http://docs.ros.org/en/rolling/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Cpp-Service-And-Client.html

Noregs miljø- og biovitskaplege universitet

Institutt for datavitskap

Digitalisering på Ås

Presentation scheduling:

Fifth worksheet and programming projects

INF205

15. april 2024

Tutorial schedule: Second half of the semester

Week 12
Tutorial session attended by two people

Week 13
Easter break – no tutorial

Week 14
Tuesday: Deadline for worksheet 4 – Wednesday: Presentations on worksheet 4

Week 15
Tutorial session attended by zero people

Week 16
No tutorial session this Wednesday

Week 17
Tuesday: Deadline for worksheet 5 – Wednesday: Presentations on worksheet 5

Week 18
Wednesday falls on Labour Day – no tutorial

Week 19
Presentations of the programming projects: Both on Monday and Wednesday

Programming project groups

Listing according to https://nmbu.instructure.com/courses/10489/groups:

Monday, 6th May 2024

Group 1
Hallvard H. Lavik, Kim Son Ly

Group 3
Trygve B. Nomeland, Esther M. Zijerveld

Group 5
William F. B. Dahl, Natnael K. Habte
Amanda S. Halvorsen, Kristoffer Romsaas
Nicolai S. Terland

Group 7
Mina Therese Gjefle

Group 9
Isak Vartdal-Gjerde

Group 11
Agnes Agersborg

Group 15
Henrik Røiseland, Yngve R. Skaug

Wednesday, 8th May 2024

Group 2
Oliver F. Aunan, Håkon Bråten
Mathias J. Dyrén, Brage H. Ringheim

Group 4
Karan Kumar, Liibaan H. Osman

Group 6
Ragne Wiklund

Group 8
Bjørn-Eirik Roald

Group 10
Vilde R. Dale, Vishnupriya Jayachandran
Jon Kastdalen, Nada S. Mahamed

Group 12
Endre M. Åsgard

Plan: Each group to present 6 minutes, followed by 3 minutes for questions.

Noregs miljø- og biovitskaplege universitet

Institutt for datavitskap

Digitalisering på Ås

E-R diagrams on draw.io

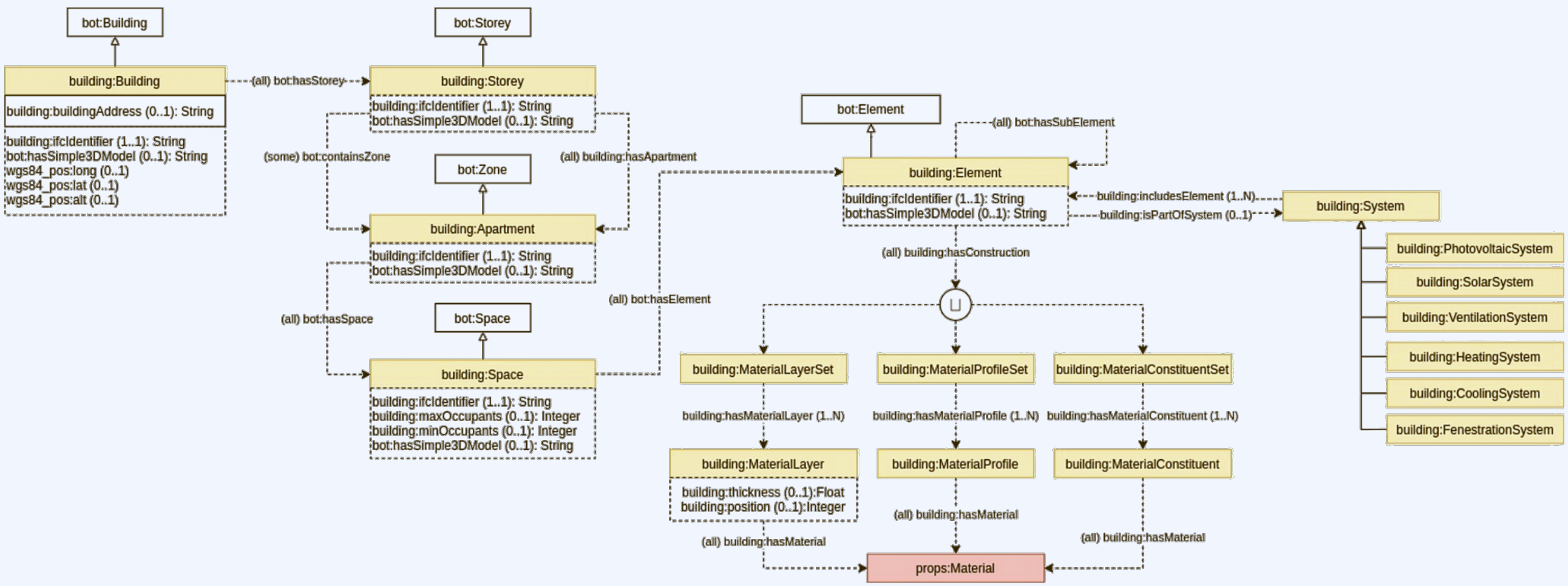
INF205

15. april 2024

E-R diagrams on draw.io and Chowlk[1,2]

The draw.io tool can be used for E-R diagrams using a variety of conventions.

With Chowlk by Poveda Villalón *et al.*,[1,2] these can be converted to ontologies.



[1] M. Poveda Villalón *et al.*, in *Proc. VOILA23*, *CEUR Works. Proc.* **3508**: 2 (link to paper), **2023**.

[2] Chowlk template: https://chowlk.linkeddata.es/static/resources/chowlk-library-complete.xml
Lightweight version: https://chowlk.linkeddata.es/static/resources/chowlk-library-lightweight.xml

Noregs miljø- og biovitskaplege universitet

Institutt for datavitskap

Digitalisering på Ås

4 Concurrency

4.1 Parallel programming

4.2 Message passing

4.3 Domain decomposition

4.4 Robotics middleware

<u>4.5 Concurrency theory</u>

States and transitions (events)

Terminology related to concurrency is often taken from the domain of **discrete event systems** (for example, *finite automata*). Adopting such an approach:

- A system can be in any of a finite number of **states**.
- Events, or **transitions** between states, are thought of as instantaneous.
- A concurrent **process** is a (partially) temporally ordered set of events.

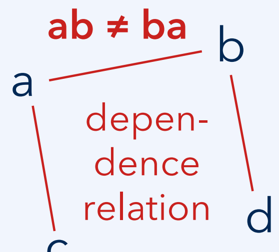
- Two events or transitions t and t' can be …
  - … **concurrent** whenever they are both enabled (*i.e.*, both can occur), one does not inhibit the other, and $t \cdot t'$ has the same outcome as $t' \cdot t$; in other words, they are concurrent if we don't say which comes first.
  - … **causally dependent** if they both occur, and it is important to say which comes first, either because only one order is possible or because it will have an impact on the outcome.

- **Limitation:** This model cannot make two transitions strictly synchronous.

Traces:[1] Partially ordered sets of events

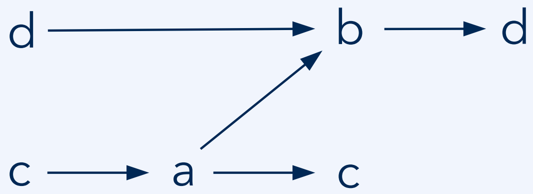
Dependence/independence between actions & events in an enterprise system:

a) Updated raw sensory data ingested into knowledge base
b) Data analysis on raw sensory data, creating aggregated data
c) Read access to raw sensory data by a user
d) Read access to aggregated data by a user



Events that are **dependent** can *never* occur *concurrently*.
Events are independent if they are **commutative**: bc = cb.

In a particular execution or process, if it is unsubstantial in what order two events occur, they are **concurrent**: Below, *e.g.*, the first and second c-d pairs:



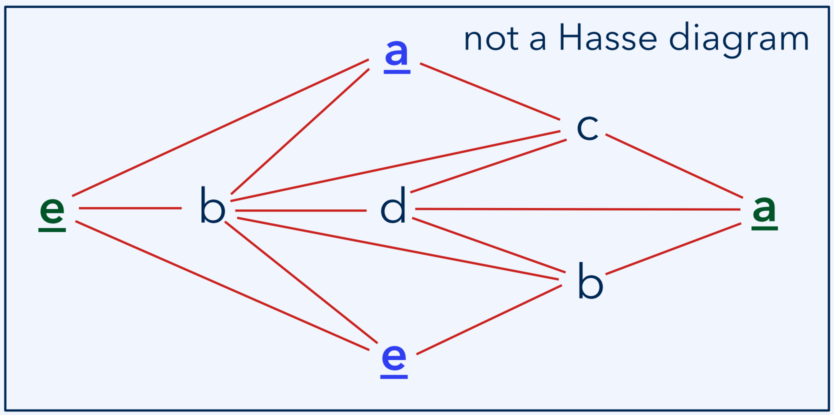
Hasse diagram for the *trace*[1]

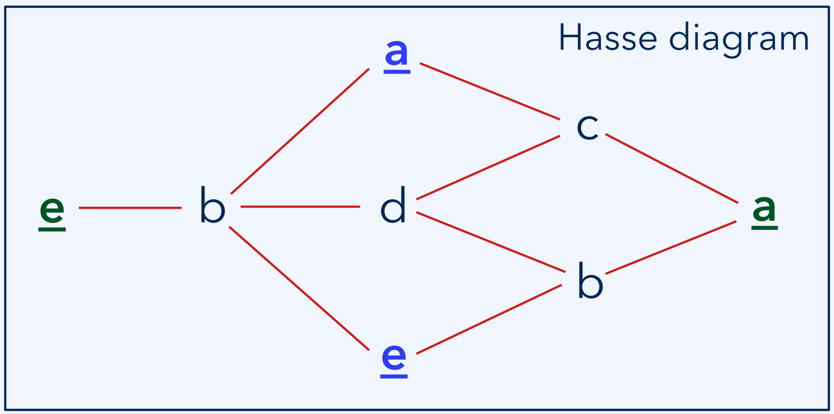
cacdbd = cdacbd = dcabdc = …

[3]Also called **Mazurkiewicz traces** after Polish mathematician Antoni Mazurkiewicz.

Diagrams for partially ordered sets

By convention, **Hasse diagrams** are often used to denote causal dependency of events. These diagrams remove *any **indirect** or **redundant** dependencies*:





Two events are **directly or indirectly causally dependent** if one is specified to occur (conclude) before the other occurs (begins). Above: e and a are indirectly dependent.

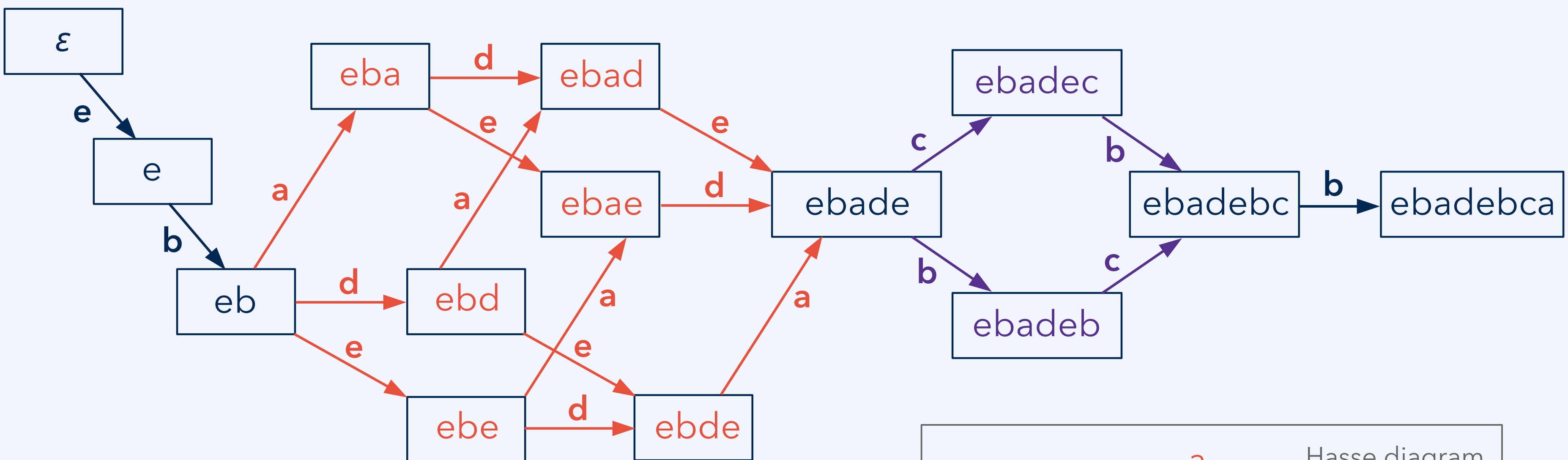
Events are **concurrent** if they are not directly or indirectly causally dependent – it does not matter which occurs first. Above: e and a are concurrent.

Attention

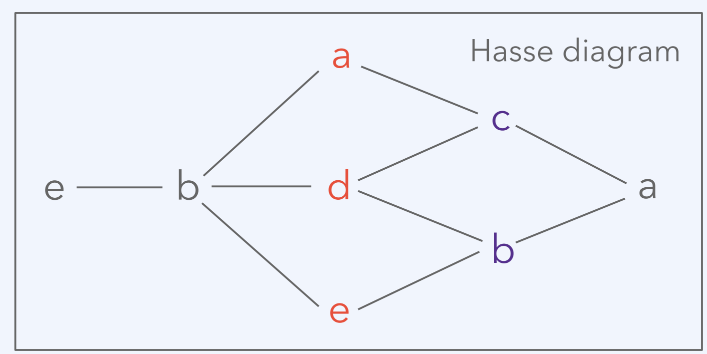
This notation only shows the **transitions** (events). The **states** (configurations) of the system are not shown.

State-transition diagrams

In a **state-transition diagram**, *two concurrent transitions* give rise to *"diamond" patterns*. *More than two concurrent transitions* lead to *(hyper-)cube patterns*:



Observation: With n concurrent events, we obtain 2^n states, making it prohibitively expensive to explore the whole state space. ("**State explosion problem**".)



Noregs miljø- og biovitskaplege universitet

Institutt for datavitskap

Digitalisering på Ås

4 Concurrency

4.1 Parallel programming

4.2 Message passing

4.3 Domain decomposition

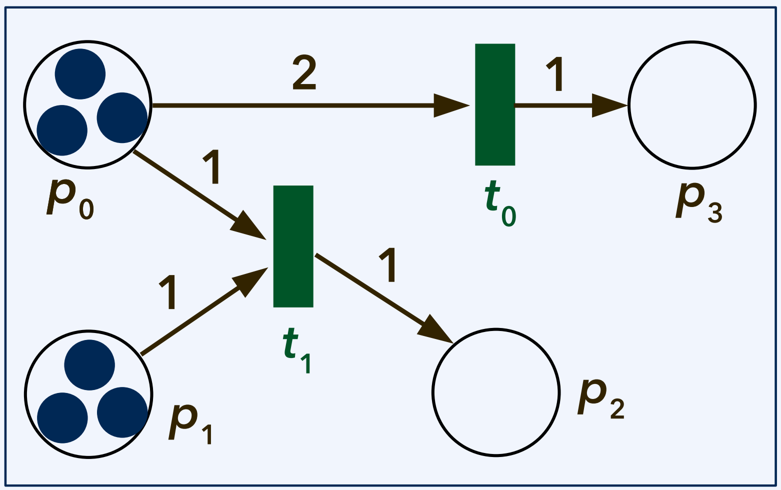
4.4 Robotics middleware

4.5 Concurrency theory

4.6 Parallel process models

Petri nets

Components of a Petri net: **places** **transitions** **tokens** **arc**

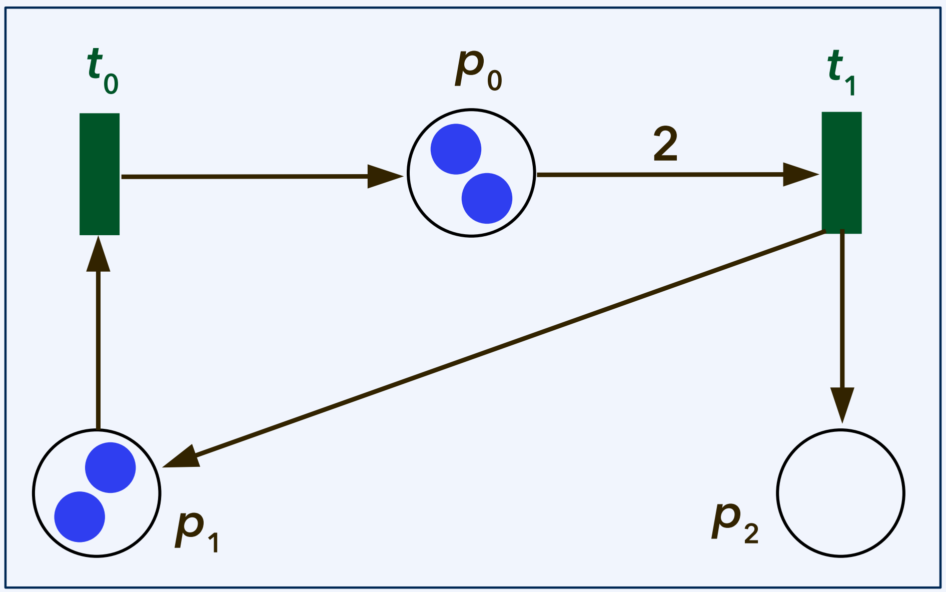


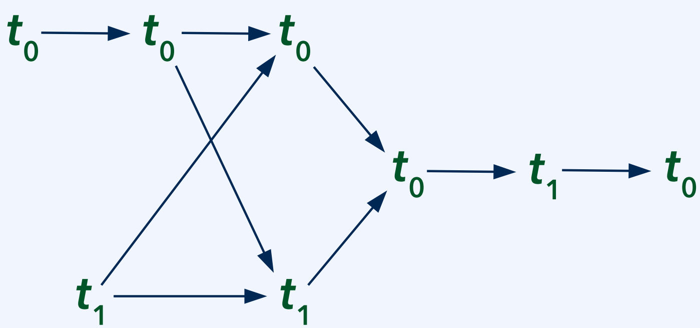
Semantics of this net:

Transition t_0 can only be **fired** if place p_0 contains at least two tokens. Firing t_0 will take away two tokens from p_0 and add one token to p_3.

Transition t_1 can only be fired if both p_0 and p_1 each contain at least one token. It removes one token from each, and adds one token to place p_2.

Petri nets: Example



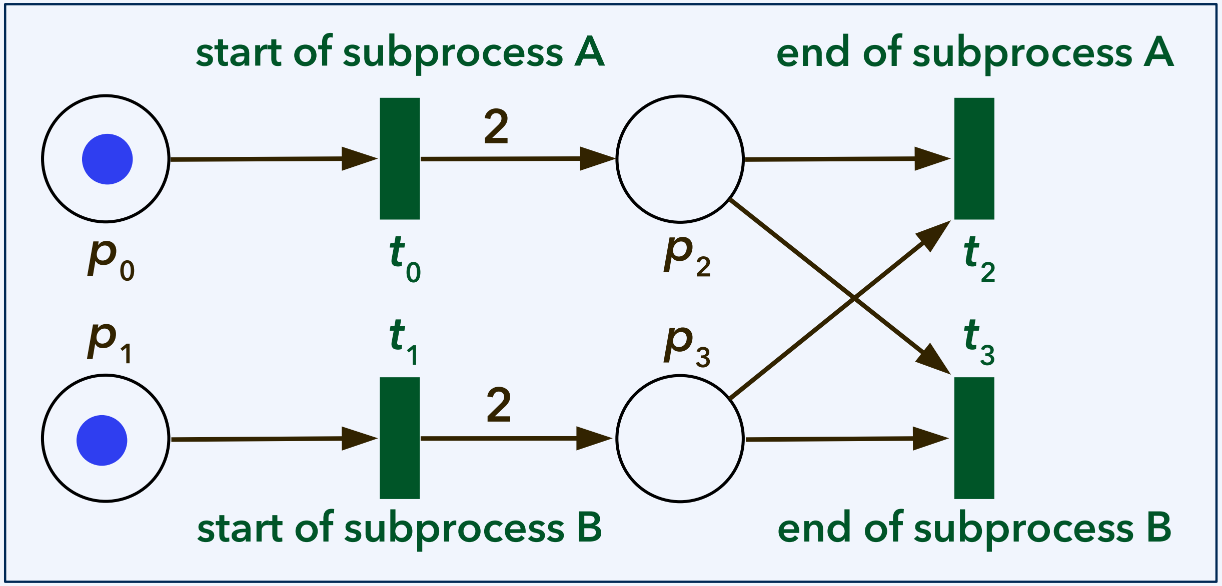


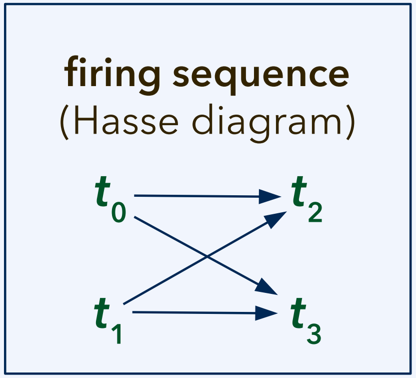
- Transitions can be fired in the following order: $t_0t_0t1t_0t1t_0t1t_0$, $t_0t_0t1t1t_0t_0t1t_0$, $t_0t1t_0t_0t1t_0t1t_0$, $t_0t1t_0t1t_0t_0t1t_0$, $t1t_0t_0t_0t1t_0t1t_0$, and $t1t_0t_0t1t_0t_0t1t_0$. At that point, respectively, a deadlock is reached.
- The net is bounded: There is a limit to the number of tokens per place.

Petri nets and synchronous processes

Two subprocesses are synchronous (also, "coupled") if it is specified that they must overlap temporally, *i.e.*, they must at least in part run at the same time.

Petri net representing two synchronous subprocesses A and B

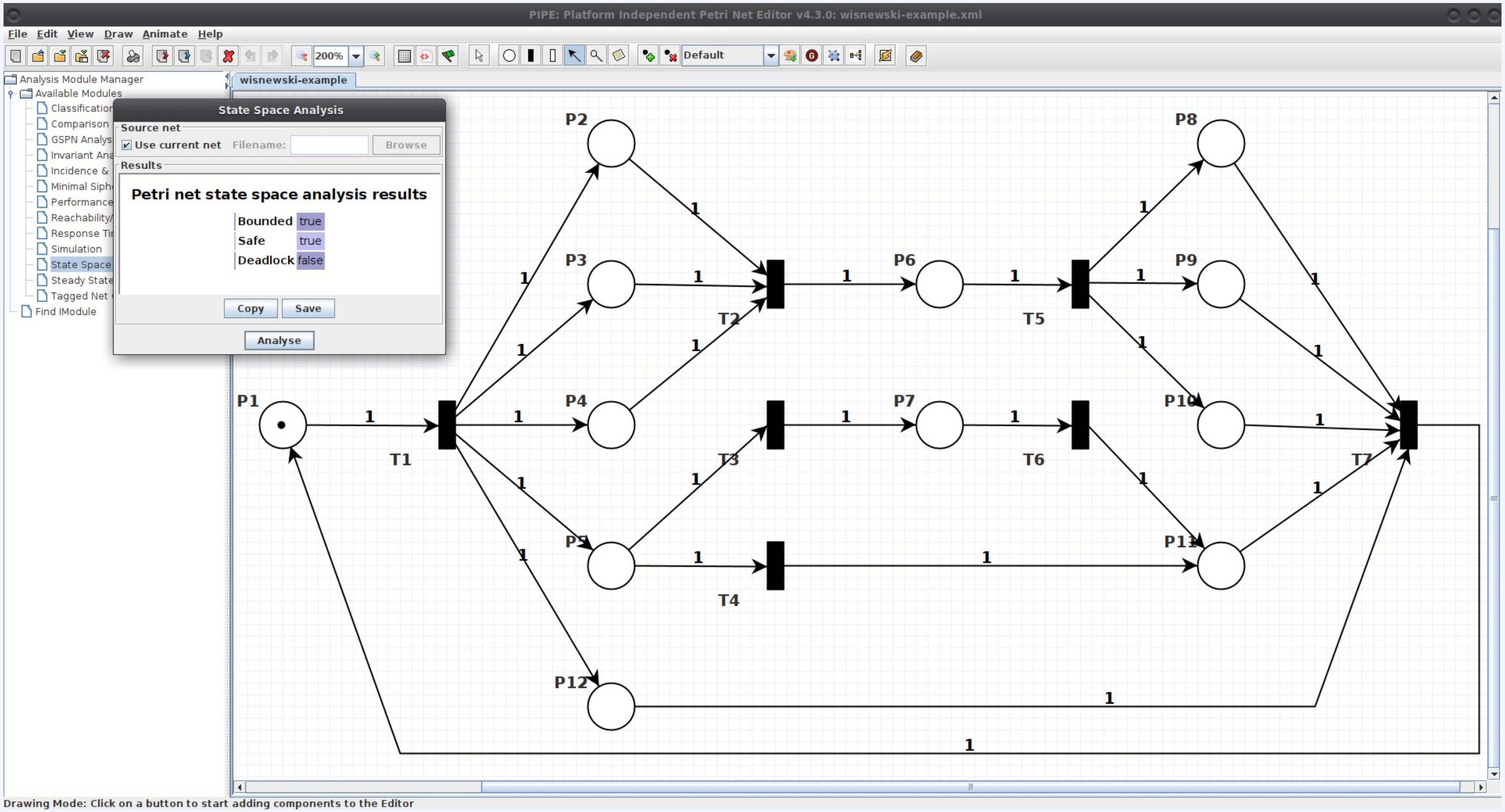




Note: **Synchronicity** ("**coupling**" – subprocesses must overlap) vs. **direct causal dependency** ("**linking**" – may not overlap) vs. **concurrency** (order unspecified).

Petri net editor

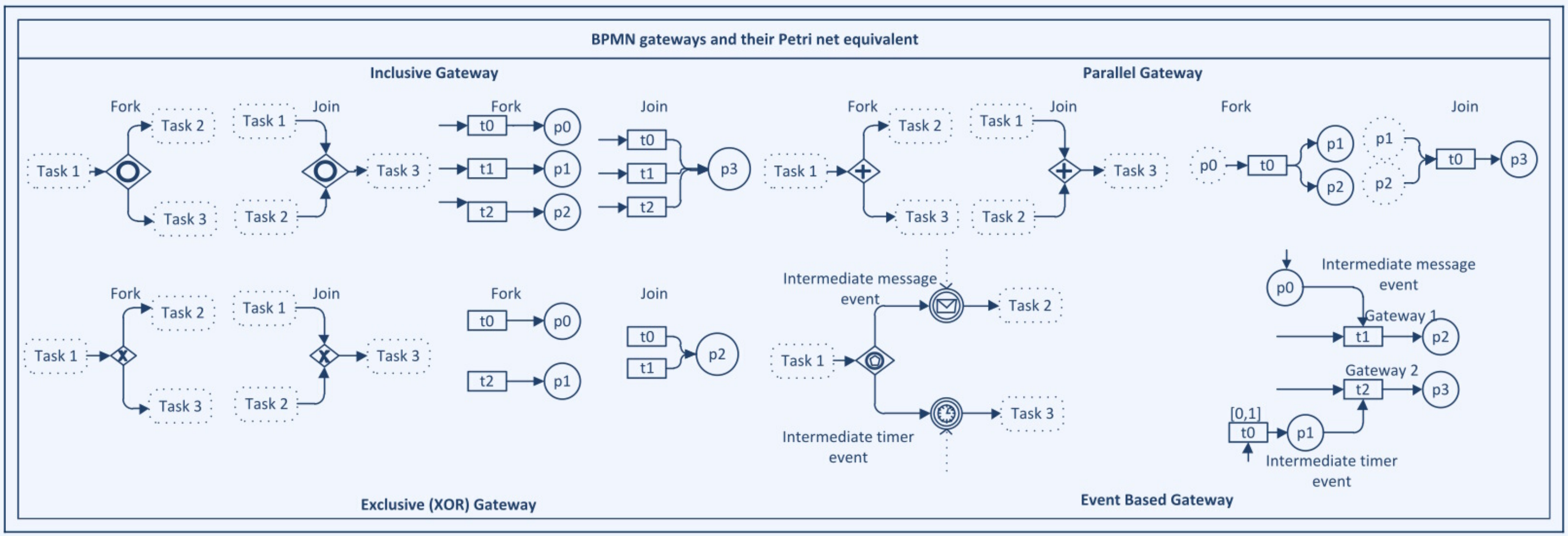
PIPE tool for editing/simulating Petri nets: http://pipe2.sourceforge.net/



Petri nets in relation to BPMN

BPMN: Business Process Model and Notation

- XML input/output of workflows[1] based on an XML schema (XSD)
- Hierarchical inclusion of a subworkflow within an overarching workflow
- Orchestration via process automation systems[2] (*e.g.*, Camunda)
- … and there are algorithms that translate BPMN into Petri nets:[3]



[1]https://www.omg.org/spec/BPMN/2.0.2/PDF. [2]Ruecker, *Practical Process Automation*, O'Reilly, **2021**.
[3]U. Mutarraf *et al.*, *Adv. Mech. Eng.* 10(12), doi:10.1177/1687814018808170, **2018**.

Noregs miljø- og biovitskaplege universitet

Institutt for datavitskap

Digitalisering på Ås

Conclusion

Weekly glossary concepts

What are essential concepts from this lecture?

Let us include them in the **INF205 glossary**.[1]

???

trace

distributed system

???

Petri net

???

[1]https://home.bawue.de/~horsch/teaching/inf205/glossary-en.html

Norges miljø- og biovitenskapelige universitet

Institutt for datavitenskap

Digitalisering på Ås

INF205

Resource-efficient programming

4 Concurrency

4.1 Parallel programming
4.2 Message passing interface
4.3 Domain decomposition
4.4 Robotics middleware
4.5 Concurrency theory
4.6 Parallel process models