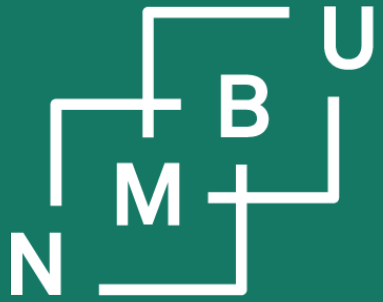
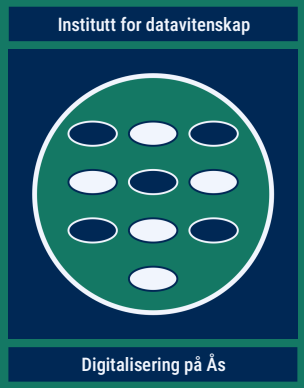


Norges miljø- og biovitenskapelige universitet



INF205
Resource-efficient programming

3 Concurrency

3.5 Message passing with ROS
3.6 Shared memory (OpenMP)
3.7 Concurrent process models

Orion: A heterogeneous architecture

Number of nodes	RAM(GB)*	CPU type	Clock rate (GHz)	Cores **	$TMPDIR(TB)***	Nodes	Remark
1	3000	Xeon(R) Gold 6140	2.30	144	2	cn-1	
7	192	Xeon(R) CPU E5-2650 v2	2.60	32	4.5	cn[4-11]	
1	256	Xeon(R) CPU E5-2683 v4	2.10	64	4.5	cn-13	
2	1000	Xeon(R) CPU E7-4870	2.40	80	18/8	cn-[2-3]	
4	256	EPYC 7302 16-Core	3.0	64	16	gn-[0-3]	3 x NVIDIA Quadro RTX 8000
2	256	Xeon(R) CPU E5-2650 v2	2.60	32	11	cn-12,15	
1	2000	EPYC 7742 64-Core	2.25 - 3.40	256	8	cn-14	
2	1000	EPYC 7702 64-Core	2 - 3.35	256	2	cn-[16-17]	

> * Total memory for the number of nodes in this category.
> ** Total for number of nodes in this category. The actual physical core per server is Cores / 2 because all Orion HPC servers cores have 2 threads.
>
> *** is fast temporary storage at $TMPDIR per server.

https://orion.nmbu.no/en/OrionHPC

Orion accounts

Groups that submitted a week 43 status report should have received an account (inf205-22-xx; "xx" is the group *no.*) and password for Orion access.

Documentation:

- https://orion.nmbu.no/en/connectingtoorion

Verify that you can login:

- ssh inf205-22-xx@login.orion.nmbu.no

To get ssh/scp access without needing to type the password:

- ssh-keygen (on Orion)
- scp ~/.ssh/id_rsa.pub inf205-22-xx@login.orion.nmbu.no:~/.ssh/authorized_keys

Home directory (~) for executables, *etc.*, $SCRATCH for large temporary data.

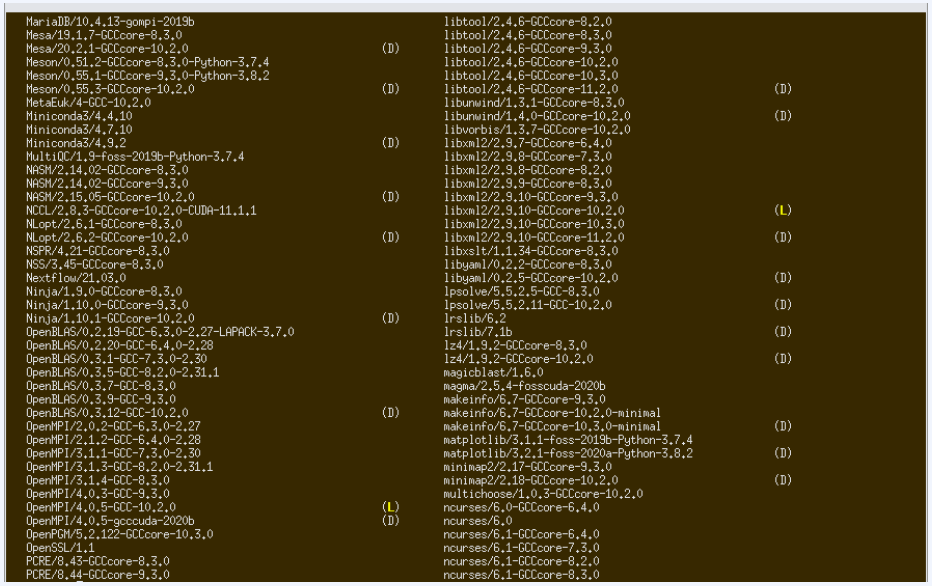
You need to be connected to the VPN (https://na.nmbu.no/) to obtain access.

Load modules and compile the code

Whenever possible, compile code on the target platform.
(Otherwise, cross-compilation has to be done …). So let us compile on Orion.

Example

```
scp ... inf205-22-xx@login.orion.nmbu.no:~/src/primes/

ssh inf205-22-xx@login.orion.nmbu.no

module avail
module load OpenMPI/4.0.5-GCC-10.2.0

...

make

mv count-primes-mpi ~/bin/
```



Submit your job to the SLURM queue

Norwegian University of Life Sciences

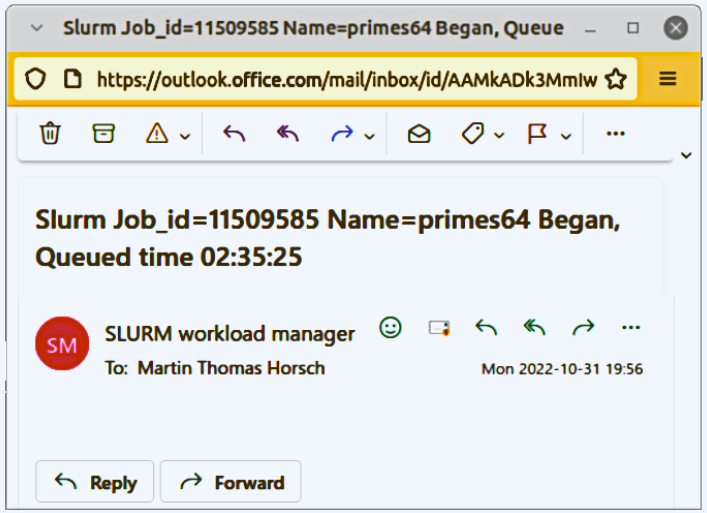
Example **primes24.run**

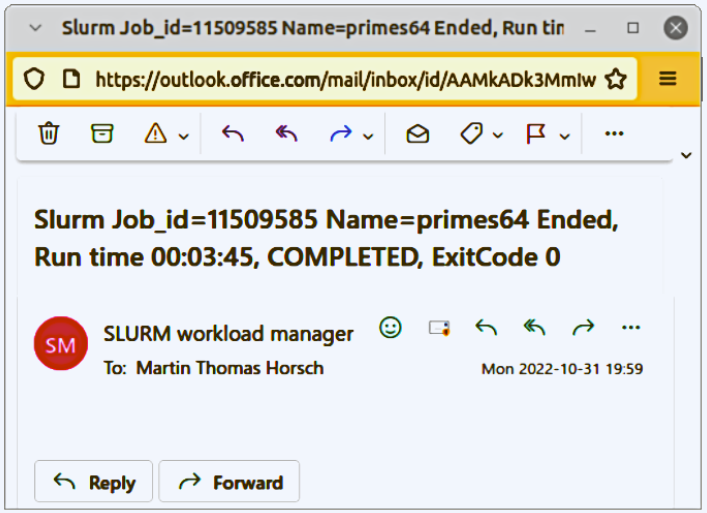
```
#!/bin/bash
#SBATCH --tasks-per-node=24          # 24 cores
#SBATCH --nodes=1                    # use 1 node
#SBATCH --time=00:30:00              # half an hour walltime
#SBATCH --job-name=primes24          # sensible name for the job
#SBATCH --partition=smallmem         # use smallmem when requiring <10 GB RAM
#SBATCH --mail-user=XXX@nmbu.no      # email me when job is done.
#SBATCH --mail-type=ALL

cd /mnt/SCRATCH/inf205-22-xx

module load OpenMPI/4.0.5-GCC-10.2.0

mpirun -np 24 /mnt/users/inf205-22-xx/bin/count-primes-mpi 1000000000 > primes24.out
```

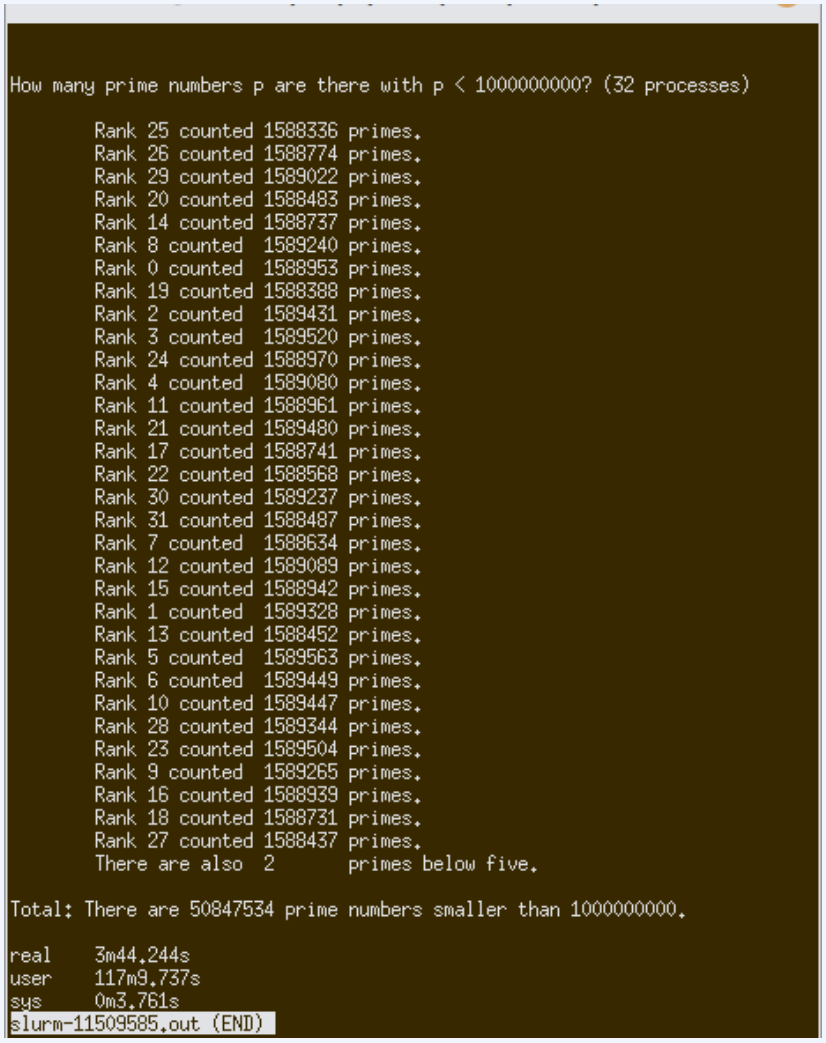
sbatch primes24.run

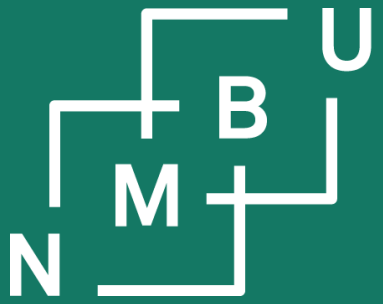
squeue

Submit your job to the SLURM queue

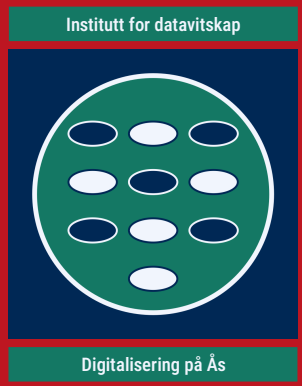








Noregs miljø- og biovitskaplege universitet



3 Concurrency

3.5 Robot operating system

INF205

2nd November 2022

ROS2 installation

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

Active ROS 2 distributions

		Recommended	Development
			
Release: June 2020 **End of life:** May 2023	**Release:** May 2021 **End of life:** Nov. 2022	**Release:** May 2022 **End of life:** May 2027	

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

The standard procedure for compiling code that uses ROS2 requires **cmake**.

ROS2 message passing paradigm

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

In a **ROS2 communication graph**, nodes and communication patterns are connected by edges that describe the direction of the data flow:

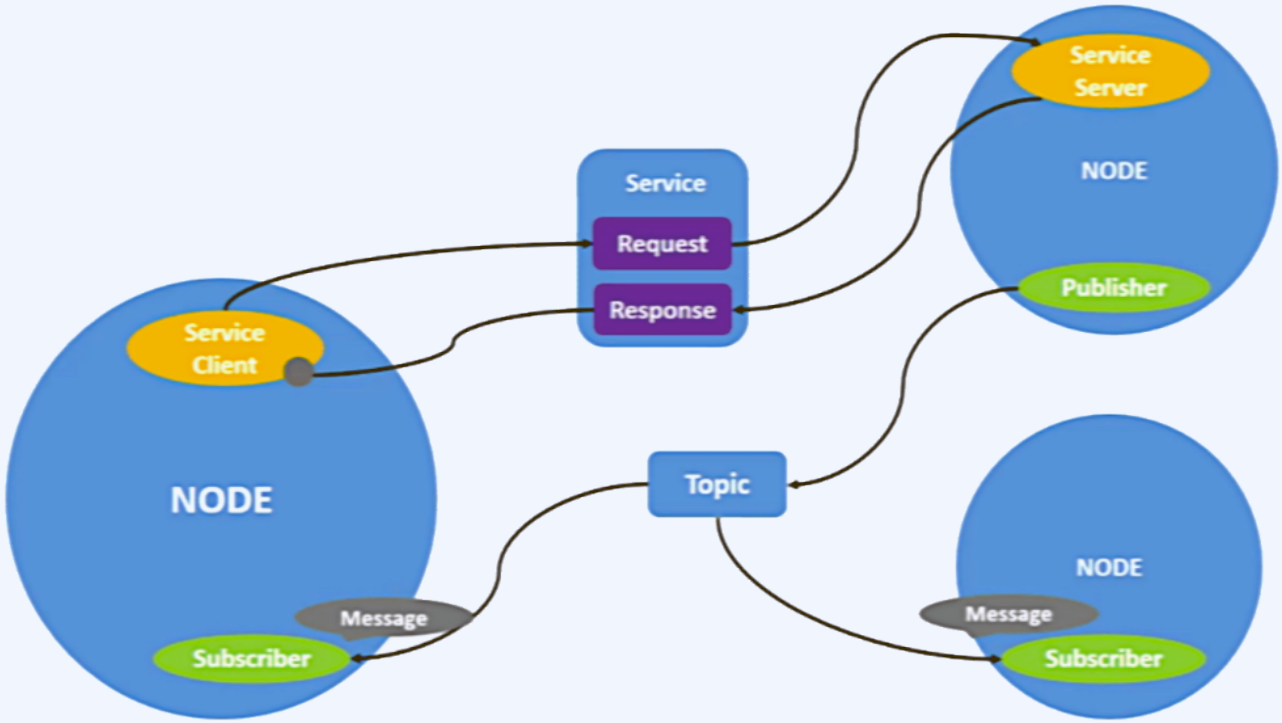


Figure: https://docs.ros.org/en/rolling/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Nodes/Understanding-ROS2-Nodes.html

ROS message passing paradigm

Where MPI and OpenMP both build on SPMD ("single program, multiple data" whether they are SIMD or MIMD), in ROS it would be MPMD. **Each node/process in ROS has its own code** and its own binary executable.

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

ROS2 package creation

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

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

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>
   …
</package>
```

package.xml

Action example[1]

Norwegian University of Life Sciences

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

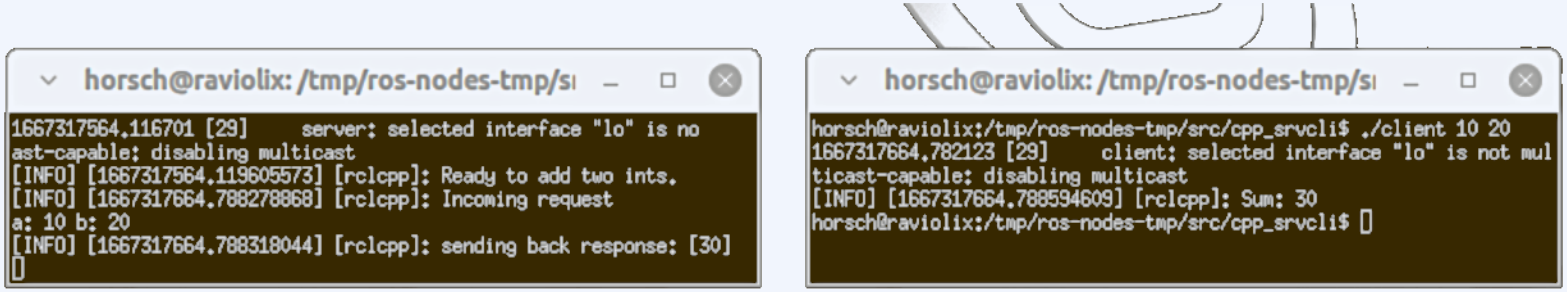
```
add_executable(
  server src/add_two_ints_server.cpp
)
ament_target_dependencies(
  server rclcpp example_interfaces
)

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/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 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.

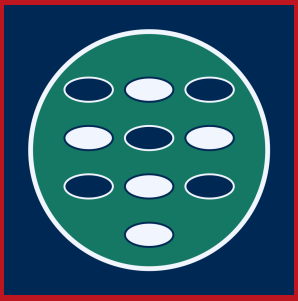


Disclaimer: If you use ROS2 for your work, include a citation to the reference S. Macenski *et al.*, *Science Robotics* **7**(66): eabm6074, doi:10.1126/scirobotics.abm6074, **2022**.

[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

3 Concurrency

3.5 Robot operating system

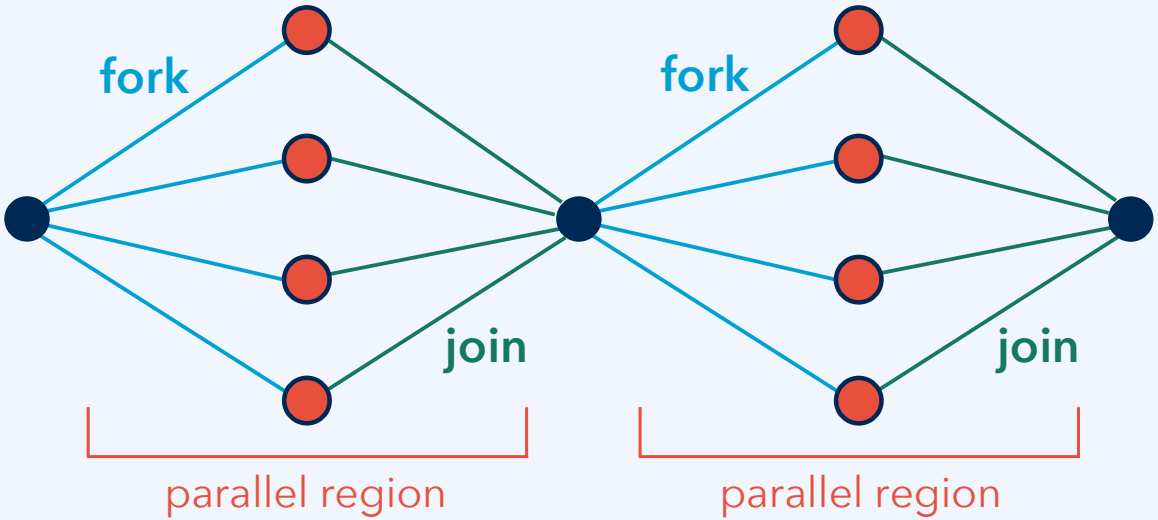
3.6 OpenMP parallelization

INF205

2nd November 2022

SIMD parallelism with OpenMP

Fork-join programming model: Alternation of parallel and sequential code.



Compiler directives are used to specify that there should be a parallel region.

- #include <**omp.h**>
- compile with **-fopenmp**
- before execution, export **OMP_NUM_THREADS=…**

```
#pragma omp parallel
{
  …
}
```

Example: OpenMP compared to MPI

Norwegian University of Life Sciences

Compare the **omp-primes** example code to the **mpi-primes** example code.

```
int main(int argc, char** argv){
  ...
  int64_t* counted_primes = new int64_t[num_threads];  // shared memory!
  omp_set_num_threads(num_threads);  // default would be to create $OMP_NUM_THREADS threads
  #pragma omp parallel {
    int thread_id = omp_get_thread_num();  // corresponds to MPI_Comm_rank in the MPI code

    counted_primes[thread_id] = 0;
    for(int64_t n = 6*(thread_id+1) - 1; n < limit; n += 6*num_threads)
      if(is_prime(n)) counted_primes[thread_id]++;
    for(int64_t n = 6*(thread_id+1) + 1; n < limit; n += 6*num_threads)
      if(is_prime(n)) counted_primes[thread_id]++;
  }
  ...
  int64_t overall_primes = 0;
  for(int i = 0; i < num_threads; i++) overall_primes += counted_primes[i];  // shared memory!
  ...
}
```

Attention: Risk of "**false sharing**" due to L1 cache line overlap.

(Compare code **omp-primes-padding**.)

Discussion

Idea: What if we simply use a single int64_t variable to count all the primes?

```
int main(int argc, char** argv){
  ...
  int64_t overall_primes = 0;  // shared memory!
  omp_set_num_threads(num_threads);  // default would be to create $OMP_NUM_THREADS threads
  #pragma omp parallel
  {
    int thread_id = omp_get_thread_num();  // corresponds to MPI_Comm_rank in the MPI code

    for(int64_t n = 6*(thread_id+1) - 1; n < limit; n += 6*num_threads)
      if(is_prime(n)) overall_primes++;
    for(int64_t n = 6*(thread_id+1) + 1; n < limit; n += 6*num_threads)
      if(is_prime(n)) overall_primes++;
  }
  ...
}
```

What do you expect from a code like this?

Synchronization

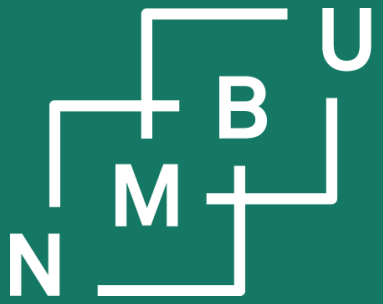
- #pragma omp **barrier**:
  Wait until all the processes have reached this point (same as in MPI)

- #pragma omp **ordered**:
  Executed by the parallel processes *in order*, 0, 1, 2, …, sequentally

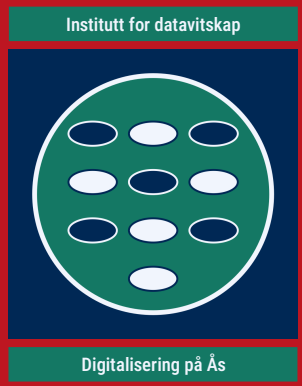
- #pragma omp **atomic**:
  Mutually exclusive access to a statement where a data item is updated

- #pragma omp **critical**:
  Mutually exclusive access to a block of code

- #pragma omp **single**:
  Block of code is only executed by *one* of the concurrent processes



Noregs miljø- og biovitskaplege universitet



Programming projects: Where do we stand?

INF205

2nd November 2022

Programming projects

19 project groups with 51 members are confirmed.
(Out of 26 groups with 72 members theoretically existing on Canvas.)

pixels to circles

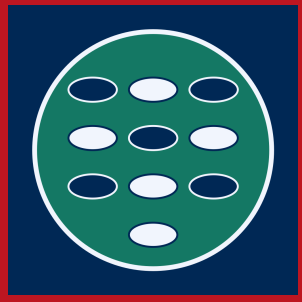
paths in labelled graphs

minimize overlap between spheres

others

Noregs miljø- og biovitskaplege universitet

Institutt for datavitskap



Digitalisering på Ås

3 Concurrency

3.5 Robot operating system

3.6 OpenMP parallelization

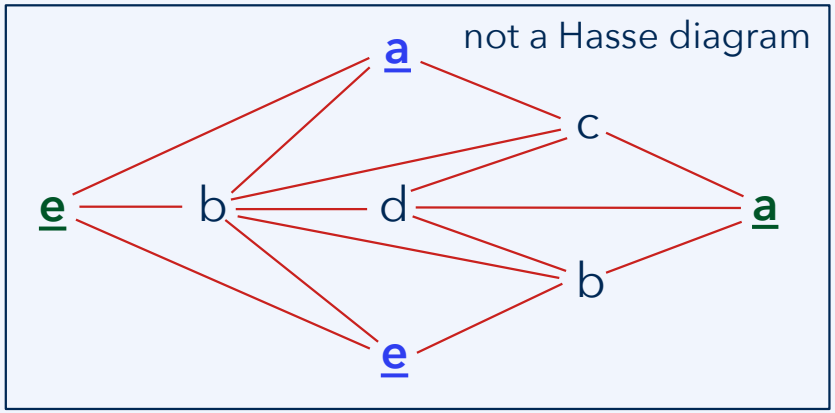
3.7 Concurrent process models

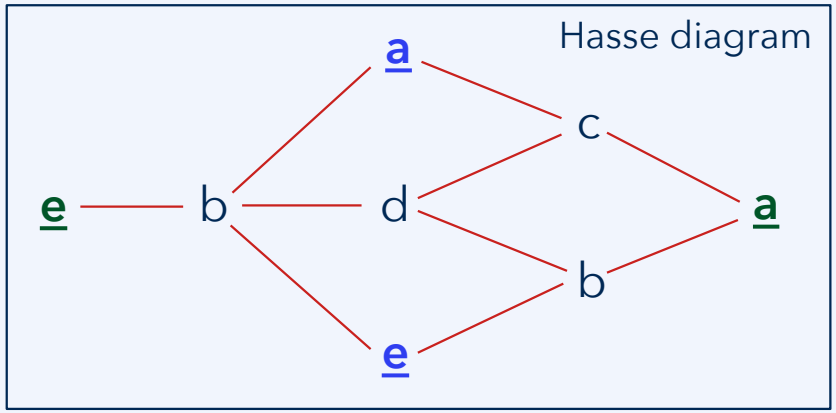
INF205

2nd November 2022

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.

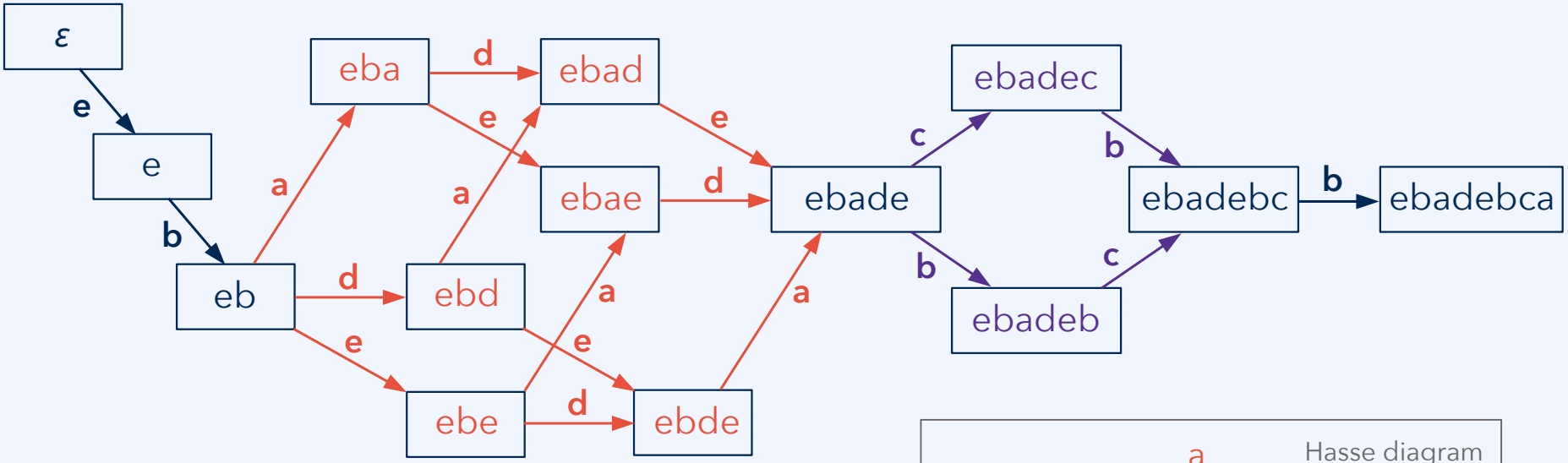
Events are **concurrent** if they are not directly or indirectly causally dependent – it does not matter which occurs first. Above: **e** and **a** are concurrent.

Attention

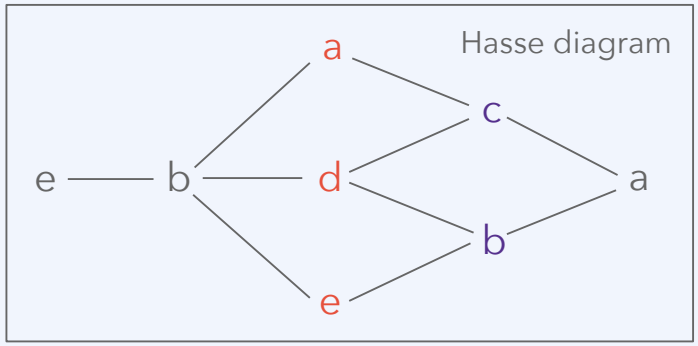
This notation only shows the **transitions** (events). The **states** (configurations) of the system are not shown.

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*:

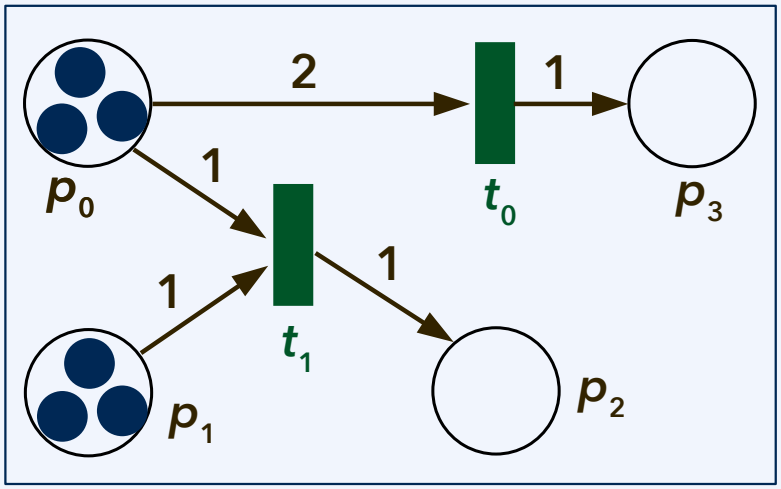


Observation: With n concurrent events, we obtain 2^n states, making it prohibitively expensive to explore the whole state space. ("**State explosion problem**".)



Petri nets

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

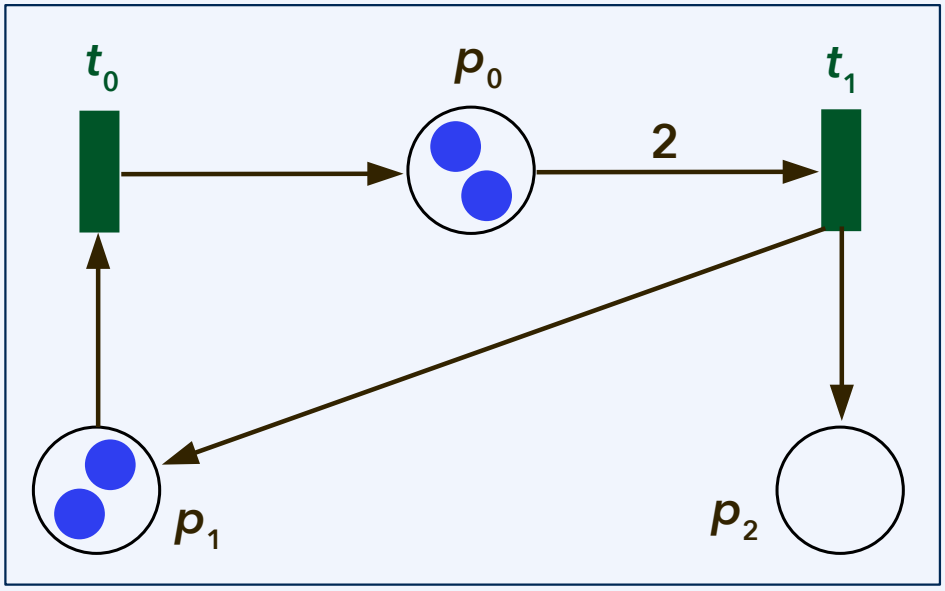


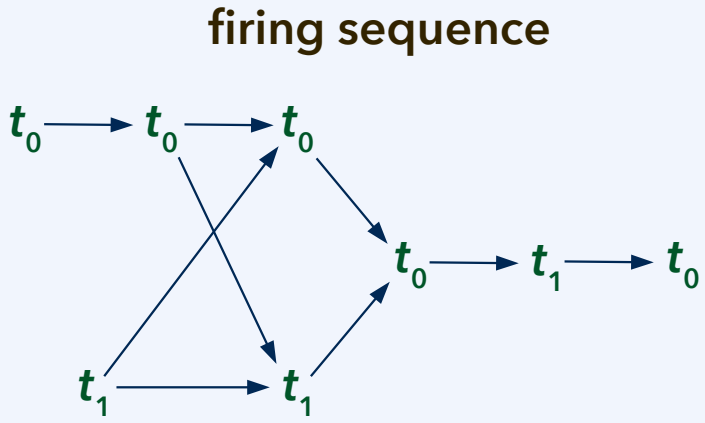
Semantics of this net:

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



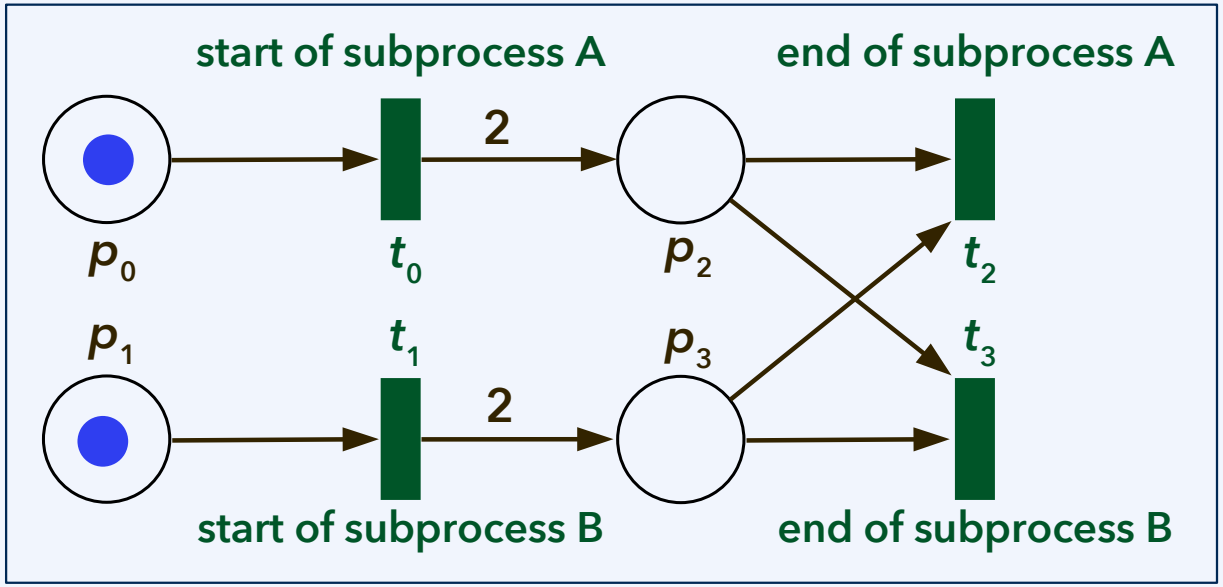


- Transitions can be fired in the following order: $t_0t_0t1t_0t1t_0t1t_0$, $t_0t_0t1t1t_0t_0t1t_0$, $t_0t1t_0t_0t1t_0t1t_0$, $t_0t1t_0t1t_0t_0t1t_0$, $t1t_0t_0t_0t1t_0t1t_0$, and $t1t_0t_0t1t_0t_0t1t_0$. At that point, respectively, a deadlock is reached.
- The net is bounded: There is a limit to the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

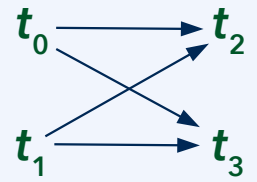
Petri nets and synchronous processes

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



firing sequence
(Hasse diagram)

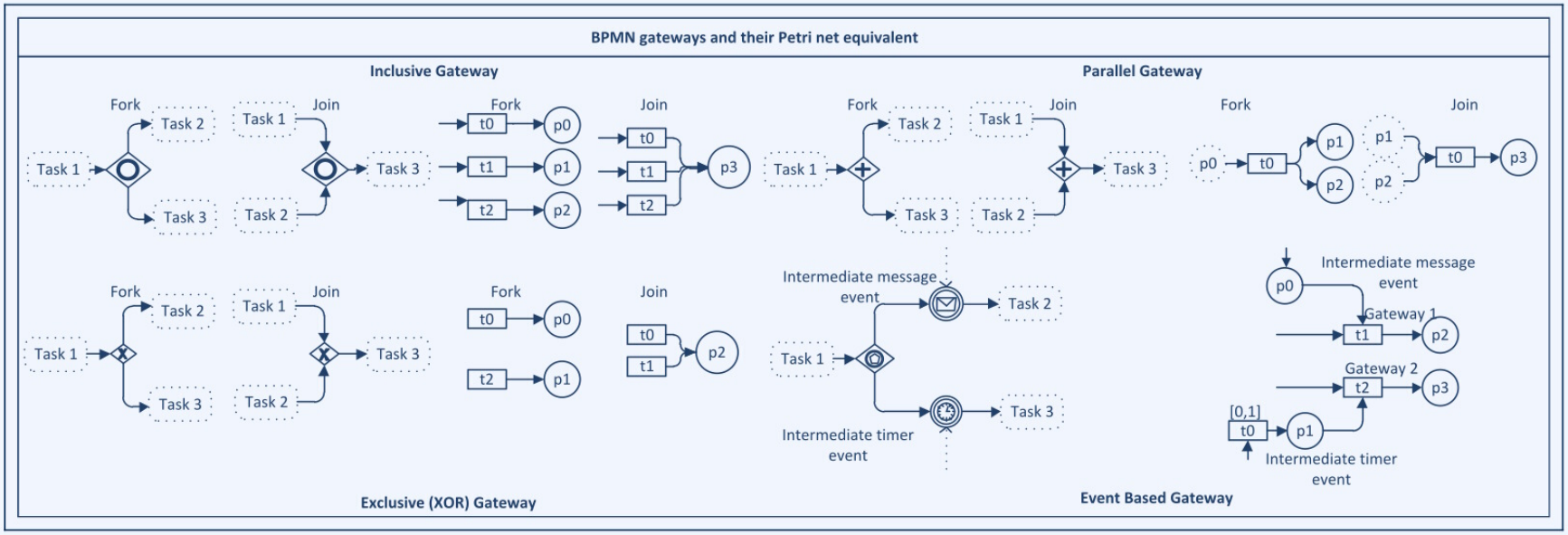


Note: **Synchronicity** ("**coupling**" – subprocesses must overlap) vs. **direct causal dependency** ("**linking**" – may not overlap) vs. **concurrency** (order unspecified).

BPMN workflows

BPMN: Business Process Model and Notation

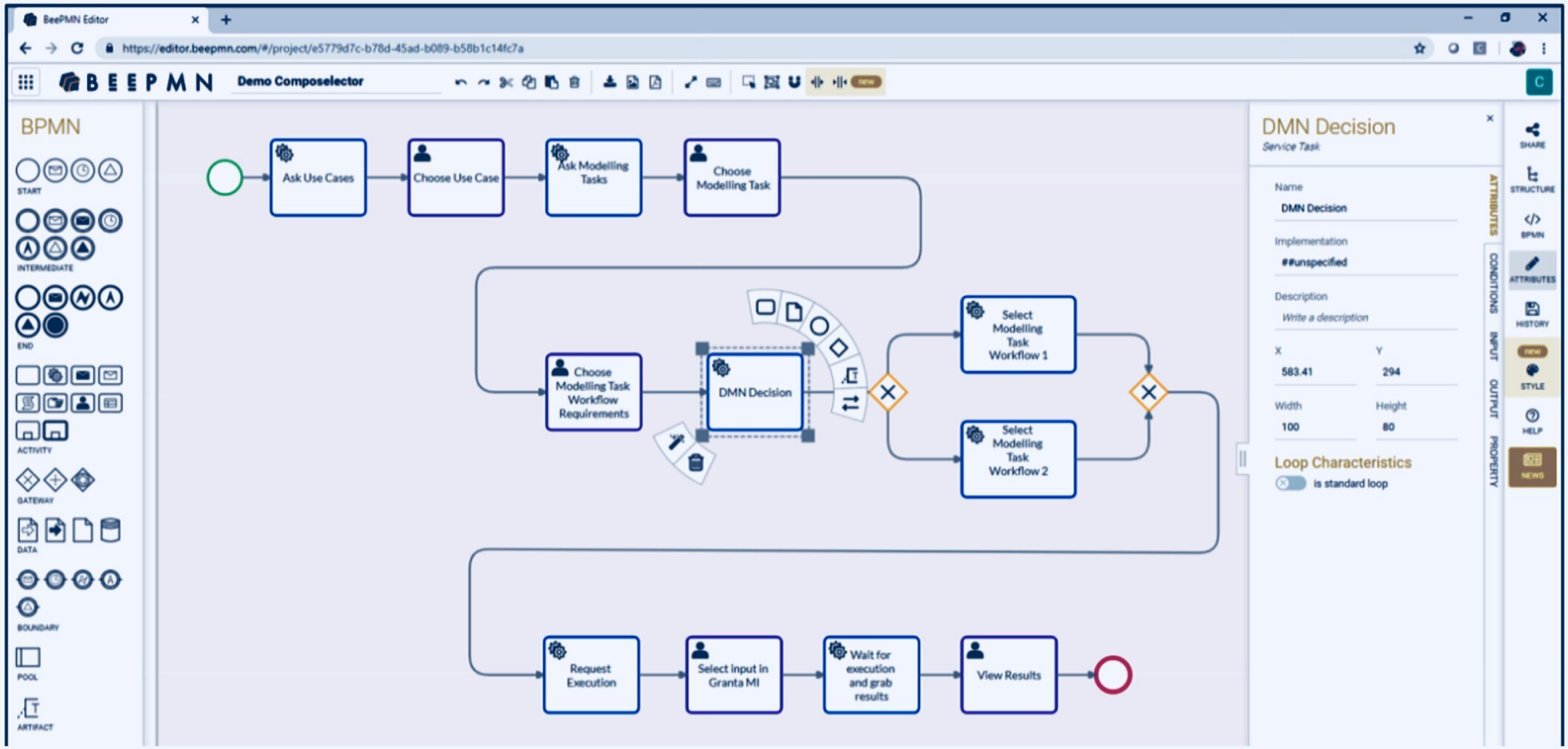
- XML input/output of workflows[1] based on an XML schema (XSD)
- Hierarchical inclusion of a subworkflow within an overarching workflow
- Orchestration via process automation systems[2] (*e.g.*, Camunda)
- … and there are algorithms that translate BPMN into Petri nets:[3]



[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**.

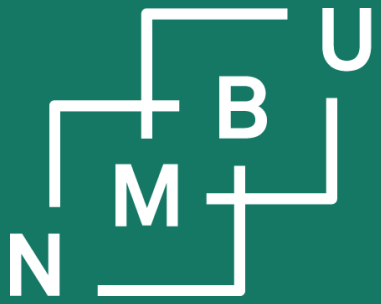
BPMN workflows

Business Process Model and Notation is standardized[1] as ISO/IEC 19510:2013.

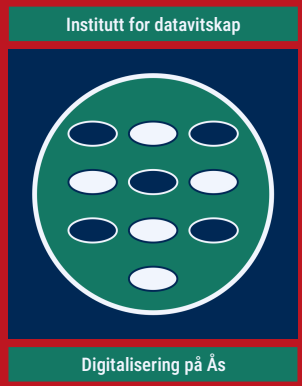


Example from A. Segatto, M. Milleri, C. Kavka, COMPOSELECTOR project deliverable 3.4, **2018**.

[1]See also the specification at https://www.omg.org/spec/BPMN/2.0.2/PDF.



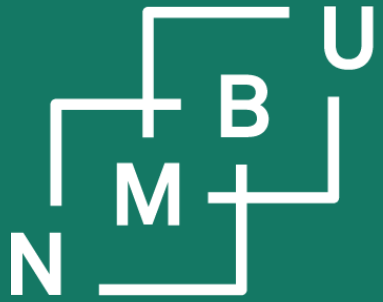
Noregs miljø- og biovitskaplege universitet



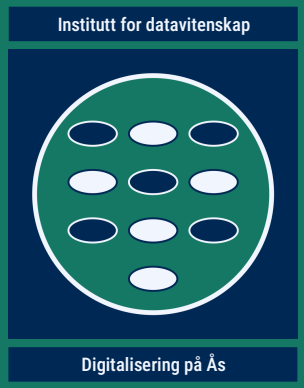
Conclusion

INF205

2nd November 2022



Norges miljø- og biovitenskapelige universitet



INF205
Resource-efficient programming

3 Concurrency

3.5 Message passing with ROS
3.6 Shared memory (OpenMP)
3.7 Concurrent process models