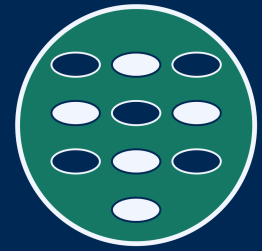


Norges miljø- og  
biovitenskapelige  
universitet

Institutt for datavitenskap



Digitalisering på Ås

# 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/
```

MariaDB/10.4.13-gompi-2019b		libtool/2.4.6-GCCcore-8.2.0	
Mesa/19.1.1-GCCcore-8.3.0		libtool/2.4.6-GCCcore-8.3.0	
Mesa/20.2.1-GCCcore-10.2.0		libtool/2.4.6-GCCcore-8.3.0	
Meson/0.51.2-GCCcore-8.3.0-Python-3.7.4	(D)	libtool/2.4.6-GCCcore-10.2.0	
Meson/0.55.1-GCCcore-9.3.0-Python-3.8.2		libtool/2.4.6-GCCcore-10.3.0	
Meson/0.55.3-GCCcore-10.2.0		libtool/2.4.6-GCCcore-11.2.0	(D)
MetaEuk/4-GCC-10.2.0		libunwind/1.3.1-GCCcore-8.3.0	
Miniconda3/4.4.10		libunwind/1.4.0-GCCcore-10.2.0	(D)
Miniconda3/4.7.10		libxkbcommon/1.3.7-GCCcore-10.2.0	
Miniconda3/4.9.2	(D)	libxml2/2.9.7-GCCcore-6.4.0	
MultiQC/1.9-Foss-2019b-Python-3.7.4		libxml2/2.9.8-GCCcore-7.3.0	
NASM/2.14.02-GCCcore-8.3.0		libxml2/2.9.8-GCCcore-8.2.0	
NASM/2.14.02-GCCcore-9.3.0		libxml2/2.9.9-GCCcore-8.3.0	
NASM/2.15.05-GCCcore-10.2.0		libxml2/2.9.10-GCCcore-9.3.0	(D)
NDGL/2.8.3-GCCcore-10.2.0-CUDA-11.1.1		libxml2/2.9.10-GCCcore-10.2.0	(L)
NLopt/2.6.1-GCCcore-8.3.0		libxml2/2.9.10-GCCcore-10.3.0	
NLopt/2.6.2-GCCcore-10.2.0	(D)	libxml2/2.9.10-GCCcore-11.2.0	(D)
NSPR/4.21-GCCcore-8.3.0		libxslt/1.1.34-GCCcore-8.3.0	
NSS/3.45-GCCcore-8.3.0		libyaml/0.2.2-GCCcore-8.3.0	
NextFlow/21.03.0		libyaml/0.2.5-GCCcore-10.2.0	(D)
Ninja/1.9.0-GCCcore-8.3.0		lpsolve/5.5.2.9-GCC-8.3.0	
Ninja/1.10.0-GCCcore-9.3.0		lpsolve/5.5.2.11-GCC-10.2.0	(D)
Ninja/1.10.1-GCCcore-10.2.0		lrslib/6.2	
OpenBLAS/0.2.19-GCC-6.3.0-2.27-LAPACK-3.7.0	(D)	lrslib/7.1b	(D)
OpenBLAS/0.2.20-GCC-6.4.0-2.28		l24/1.9.2-GCCcore-8.3.0	
OpenBLAS/0.3.1-GCC-7.3.0-2.30		l24/1.9.2-GCCcore-10.2.0	(D)
OpenBLAS/0.3.5-GCC-8.2.0-2.31.1		magicblast/1.6.0	
OpenBLAS/0.3.7-GCC-8.3.0		magma/2.5.4-Fosscore-2020b	
OpenBLAS/0.3.9-GCC-9.3.0		makeinfo/6.7-GCCcore-9.3.0	
OpenBLAS/0.3.12-GCC-10.2.0		makeinfo/6.7-GCCcore-10.2.0-minimal	
OpenMPI/2.0.2-GCC-6.3.0-2.27	(D)	makeinfo/6.7-GCCcore-10.3.0-minimal	(D)
OpenMPI/2.1.2-GCC-6.4.0-2.28		matplotlib/3.1.1-Foss-2019b-Python-3.7.4	
OpenMPI/3.1.1-GCC-7.3.0-2.30		matplotlib/3.2.1-Foss-2020a-Python-3.8.2	(D)
OpenMPI/3.1.3-GCC-8.2.0-2.31.1		minimap2/2.17-GCCcore-8.3.0	
OpenMPI/3.1.4-GCC-8.3.0		minimap2/2.18-GCCcore-10.2.0	(D)
OpenMPI/4.0.3-GCC-9.3.0		multichoose/1.0.3-GCCcore-10.2.0	
OpenMPI/4.0.5-GCC-10.2.0	(L)	ncores/6.0-GCCcore-6.4.0	
OpenMPI/4.0.5-gcccore-2020b	(D)	ncores/6.0	
OpenPGM/5.2.122-GCCcore-10.3.0		ncores/6.1-GCCcore-6.4.0	
OpenSSL/1.1		ncores/6.1-GCCcore-7.3.0	
PCRE/8.43-GCCcore-8.3.0		ncores/6.1-GCCcore-8.2.0	
PCRE/8.44-GCCcore-9.3.0		ncores/6.1-GCCcore-8.3.0	

# Submit your job to the SLURM queue

## 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

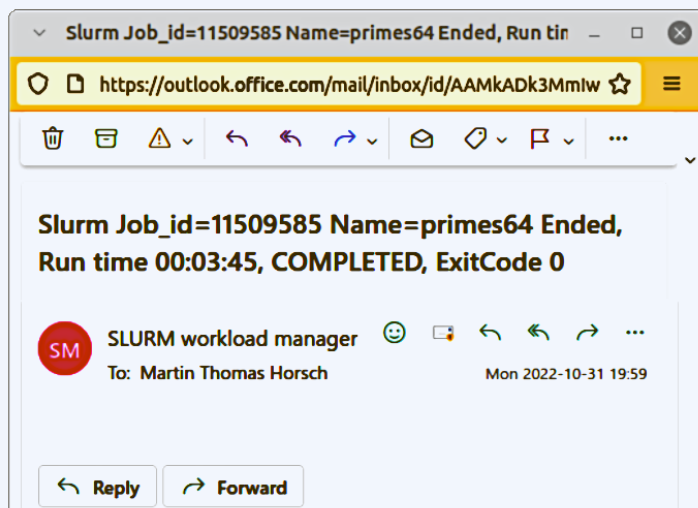
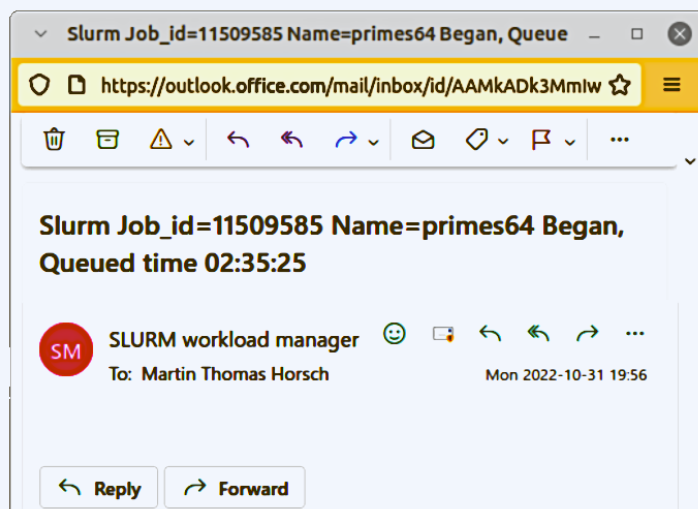
module load OpenMPI/4.0.5-GCC-10.2.0

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

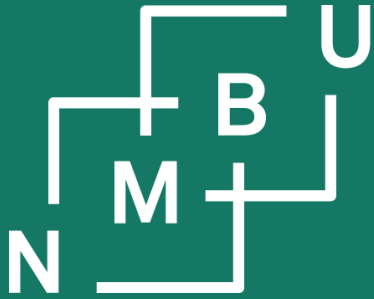


```
How many prime numbers p are there with p < 1000000000? (32 processes)

Rank 25 counted 1588336 primes.
Rank 26 counted 1588774 primes.
Rank 29 counted 1589022 primes.
Rank 20 counted 1588483 primes.
Rank 14 counted 1588737 primes.
Rank 8 counted 1589240 primes.
Rank 0 counted 1588953 primes.
Rank 19 counted 1588388 primes.
Rank 2 counted 1589431 primes.
Rank 3 counted 1589520 primes.
Rank 24 counted 1588970 primes.
Rank 4 counted 1589080 primes.
Rank 11 counted 1588961 primes.
Rank 21 counted 1589480 primes.
Rank 17 counted 1588741 primes.
Rank 22 counted 1588568 primes.
Rank 30 counted 1589237 primes.
Rank 31 counted 1588487 primes.
Rank 7 counted 1588634 primes.
Rank 12 counted 1589089 primes.
Rank 15 counted 1588942 primes.
Rank 1 counted 1589328 primes.
Rank 13 counted 1588452 primes.
Rank 5 counted 1589563 primes.
Rank 6 counted 1589449 primes.
Rank 10 counted 1589447 primes.
Rank 28 counted 1589344 primes.
Rank 23 counted 1589504 primes.
Rank 9 counted 1589265 primes.
Rank 16 counted 1588939 primes.
Rank 18 counted 1588731 primes.
Rank 27 counted 1588437 primes.
There are also 2 primes below five.

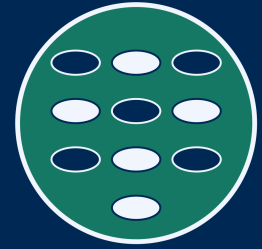
Total: There are 50847534 prime numbers smaller than 1000000000.

real    3m44.244s
user    117m9.737s
sys      0m3.761s
slurm-11509585.out (END)
```



Noregs miljø- og  
biovitenskapelige  
universitet

Institutt for datavitenskap



Digitalisering på Ås

## 3 Concurrency

### 3.5 Robot operating system

# 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

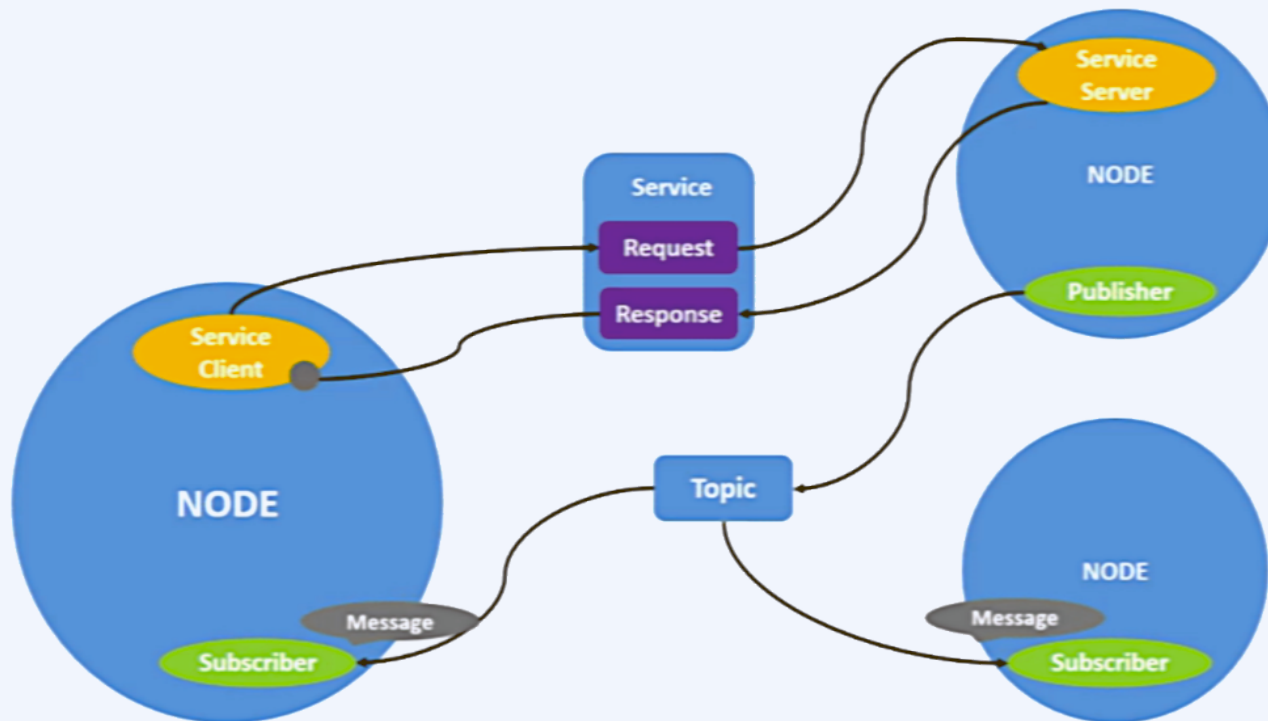


Installation by adding <http://packages.ros.org/ros2/ubuntu> repository to apt.  
The standard procedure for compiling code that uses ROS2 requires **cmake**.

# 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](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<sup>1</sup>

## 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);
```

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

## 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}
)
```

# Example<sup>1</sup>

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

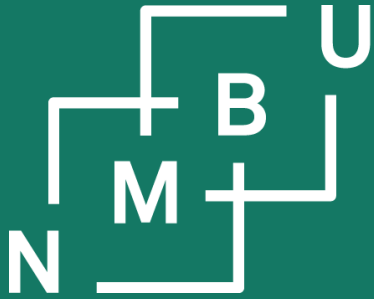
- Compile the client and server codes using cmake.
  - You may need to install cmake first.
- Run “server” on one terminal (or one computer in the network).
- Run “client x y” on another.
- They should interact, and the addition  $x+y$  should be performed.

```
horsch@raviolix: /tmp/ros-nodes-tmp/si
1667317564,116701 [29] server: selected interface "lo" is not
multicast-capable; disabling multicast
[INFO] [1667317564,119605573] [roscpp]: Ready to add two ints.
[INFO] [1667317664,788278868] [roscpp]: Incoming request
a: 10 b: 20
[INFO] [1667317664,788318044] [roscpp]: sending back response: [30]
```

```
horsch@raviolix: /tmp/ros-nodes-tmp/si
horsch@raviolix:/tmp/ros-nodes-tmp/src/cpp_srvcli$ ./client 10 20
1667317664,782123 [29] client: selected interface "lo" is not mul
ticast-capable; disabling multicast
[INFO] [1667317664,788594609] [roscpp]: Sum: 30
horsch@raviolix:/tmp/ros-nodes-tmp/src/cpp_srvcli$
```

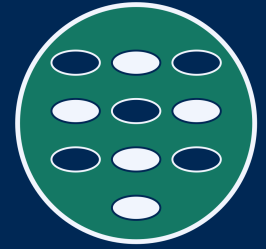
**Disclaimer:** If you use ROS2 for your work, include a citation to the reference S. Macenski et al., *Science Robotics* **7**(66): eabm6074, doi:10.1126/scirobotics.abm6074, **2022**.

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



Noregs miljø- og  
biovitenskaplege  
universitet

Institutt for datavitenskap



Digitalisering på Ås

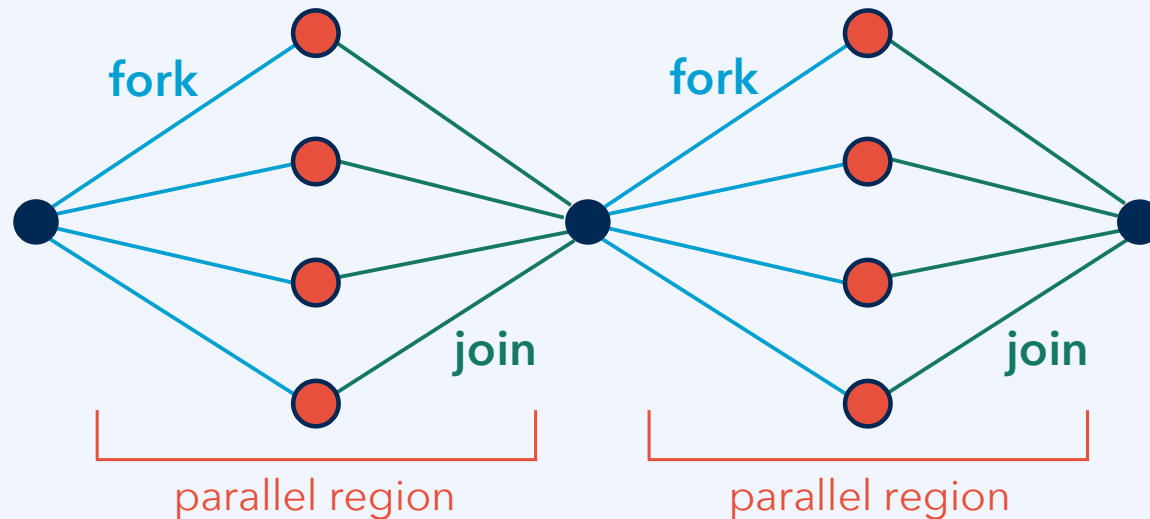
## 3 Concurrency

3.5 Robot operating system

3.6 OpenMP parallelization

# 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

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?

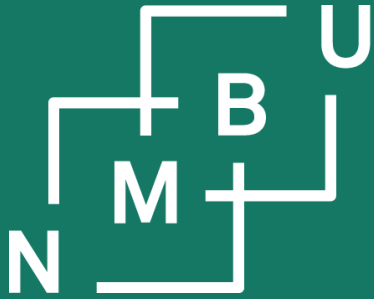
```
int main(int argc, char** argv){
    ...
    int64_t overall_primes = 0; // shared memory!
    omp_set_num_threads(num_threads); // default would be to create $OMP_NUM_THREADS threads
    #pragma omp parallel
    {
        int thread_id = omp_get_thread_num(); // corresponds to MPI_Comm_rank in the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

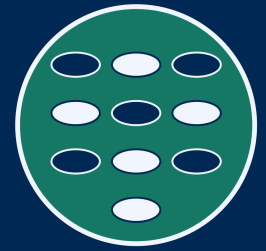
# Synchronization

- `#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, ..., sequentially
- `#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  
biovitenskapelige  
universitet

# Programming projects: Where do we stand?



# Programming projects

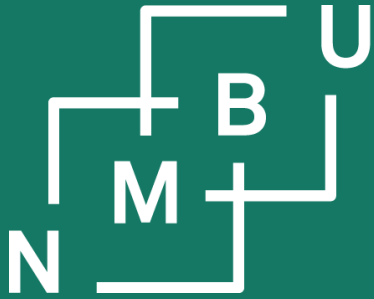
19 project groups with 51 members are confirmed.  
(Out of 26 groups with 72 members theoretically existing on Canvas.)

**pixels to circles**

**paths in  
labelled graphs**

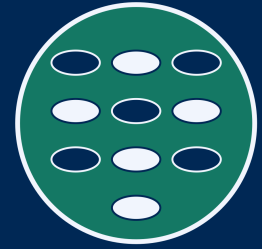
**minimize overlap  
between spheres**

**others**



Noregs miljø- og  
biovitenskaplege  
universitet

Institutt for datavitenskap



Digitalisering på Ås

## 3 Concurrency

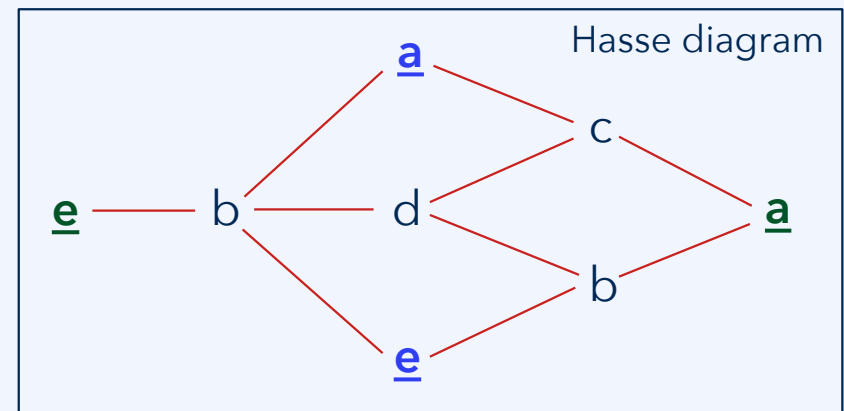
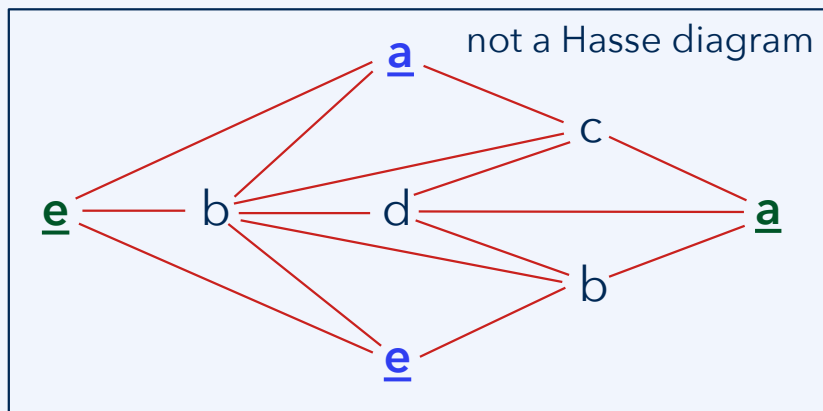
3.5 Robot operating system

3.6 OpenMP parallelization

3.7 Concurrent process models

# 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*:



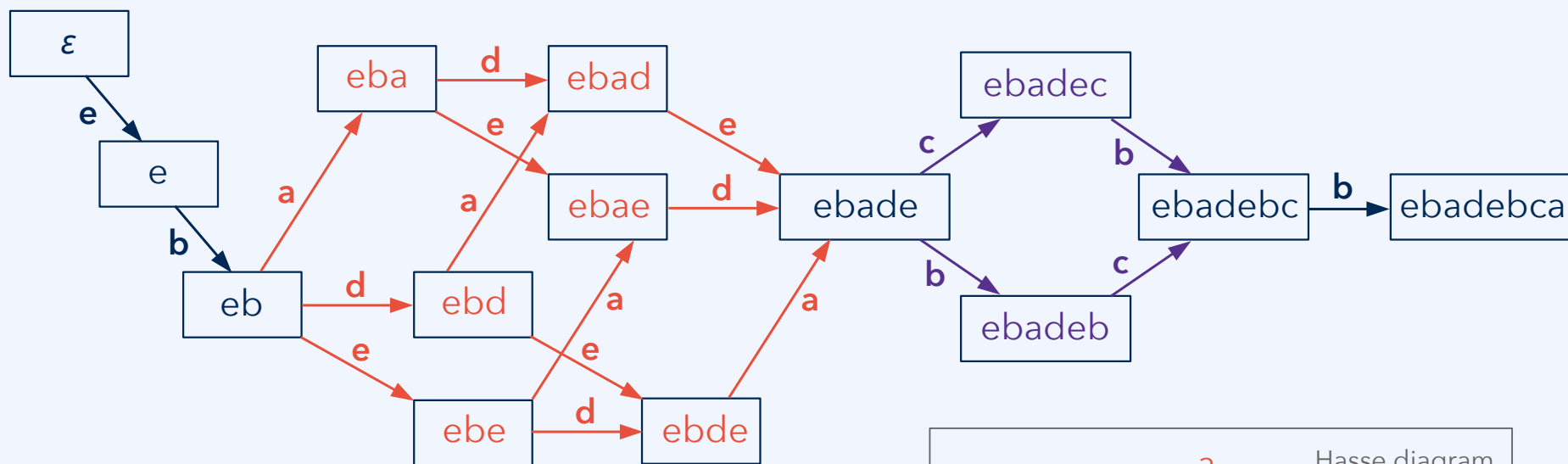
Two events are **directly or indirectly causally dependent** if one is specified to occur (conclude) before the other occurs (begins). Above: e and a are indirectly dependent. Events are **concurrent** if they are not directly or indirectly causally dependent – it does not matter which occurs first. Above: e and a are concurrent.

## Attention

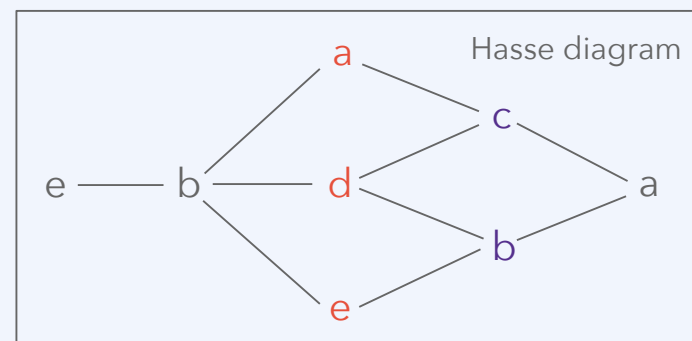
This notation only shows the **transitions** (events). The **states** (configurations) of the system are not shown.

# 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

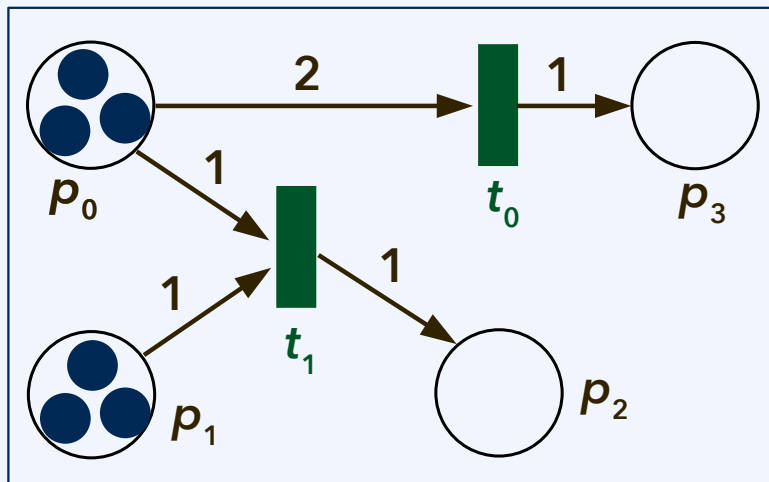
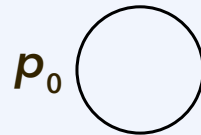
Components of a Petri net:

places

transitions

tokens

arc

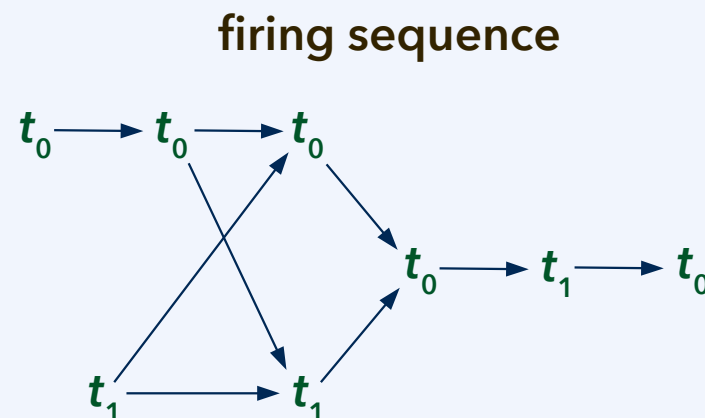
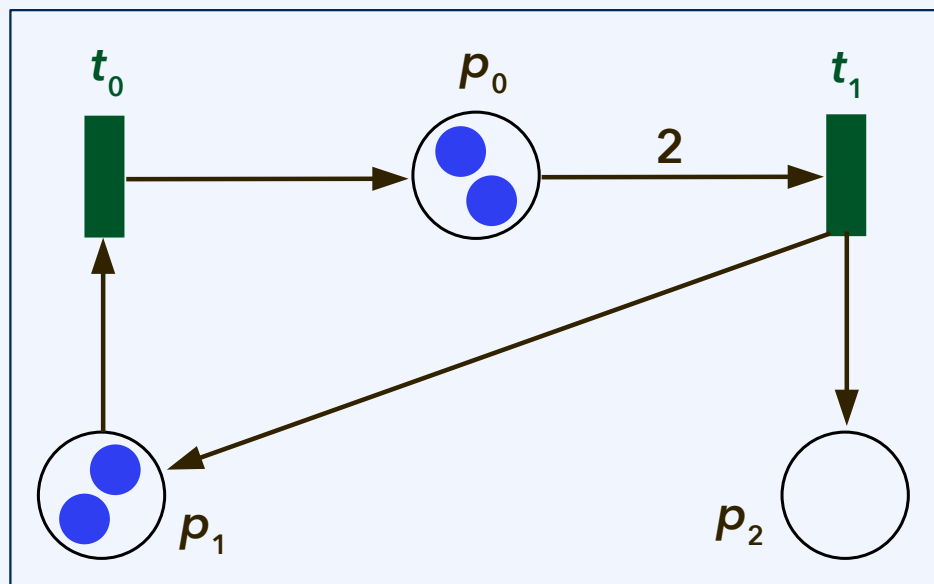


Semantics of this net:

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

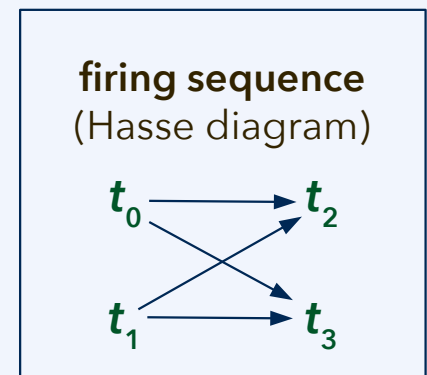
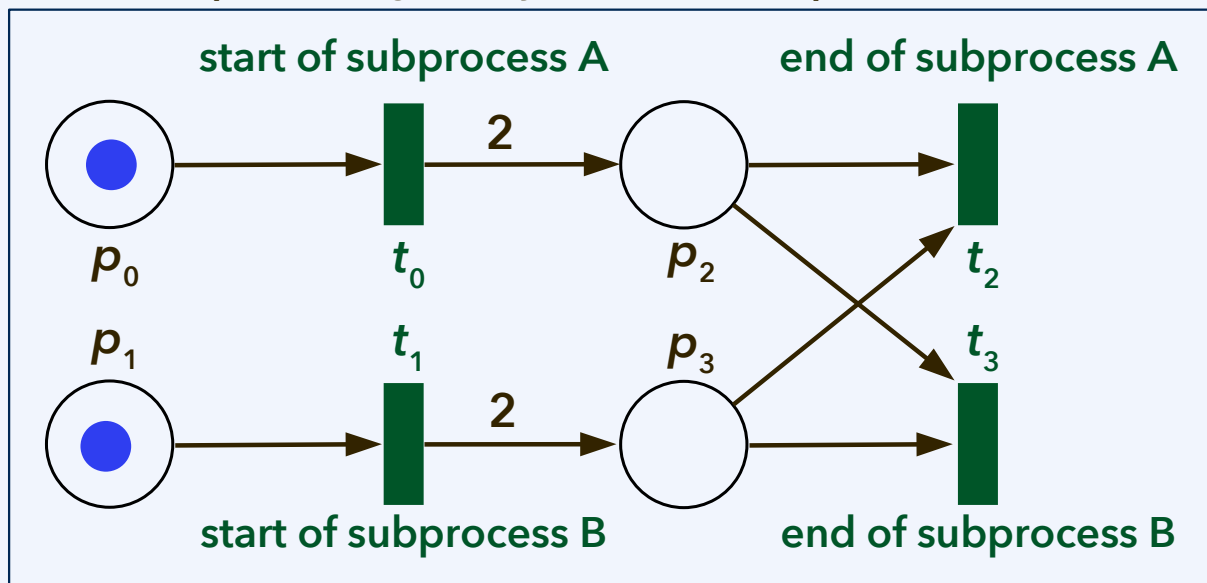


- Transitions can be fired in the following order:  $t_0 t_0 t_1 t_0 t_1 t_0 t_1 t_0$ ,  $t_0 t_0 t_1 t_1 t_0 t_0 t_1 t_0$ ,  $t_0 t_1 t_0 t_0 t_1 t_0 t_1 t_0$ ,  $t_0 t_1 t_0 t_1 t_0 t_0 t_1 t_0$ ,  $t_1 t_0 t_0 t_0 t_1 t_0 t_1 t_0$ , and  $t_1 t_0 t_0 t_1 t_0 t_0 t_1 t_0$ . At that point, respectively, 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

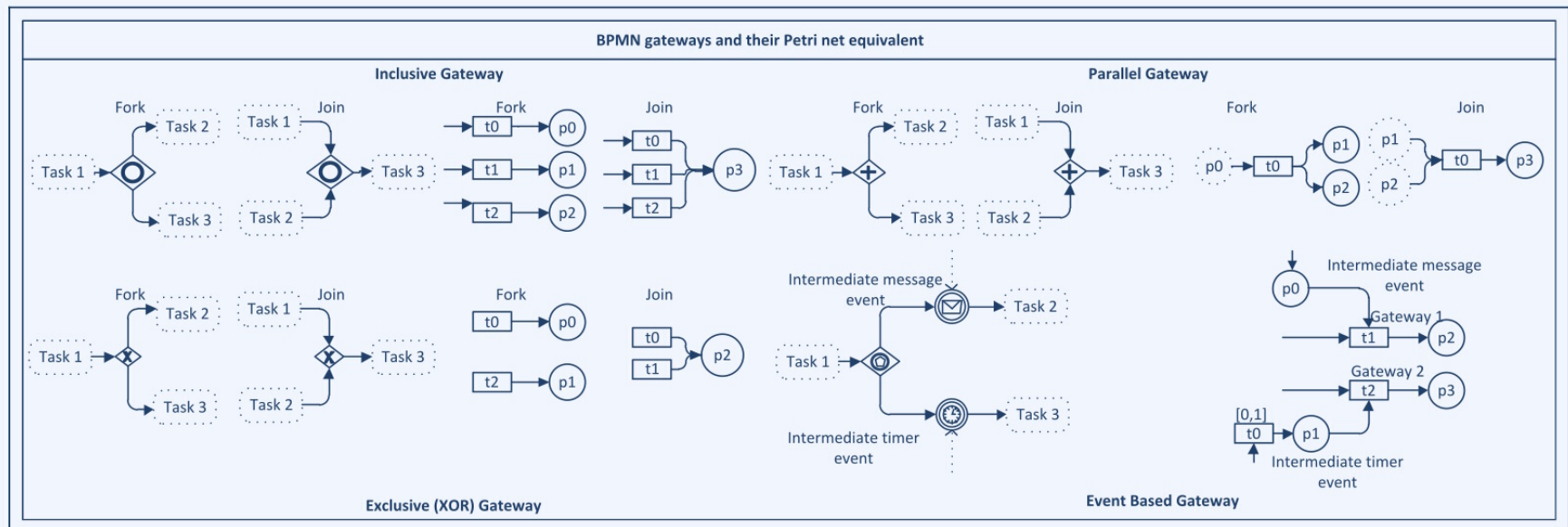


Note: **Synchronicity** (“coupling” – subprocesses must overlap) vs. **direct causal dependency** (“linking” – may not overlap) vs. **concurrency** (order unspecified).

# BPMN workflows

## BPMN: Business Process Model and Notation

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

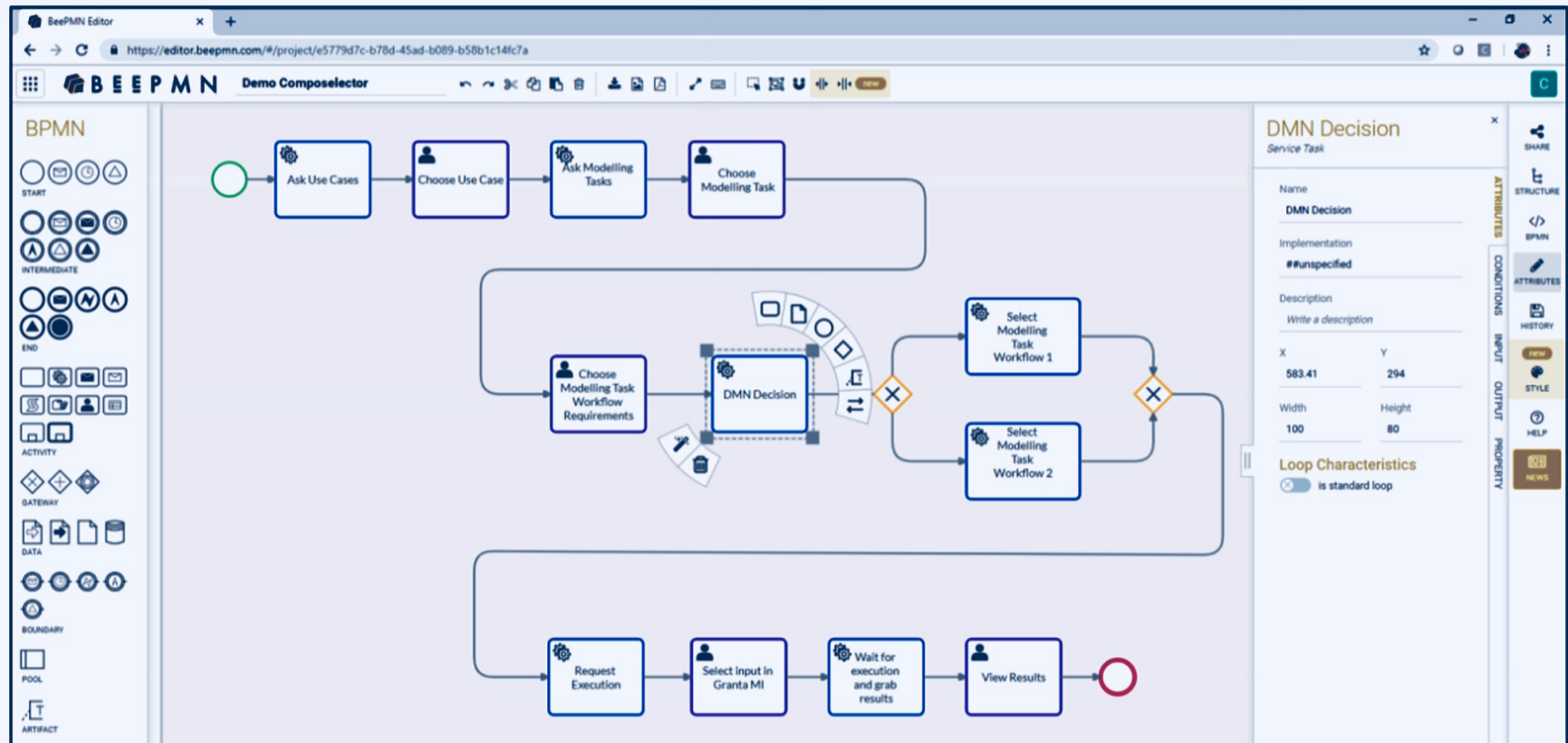


<sup>1</sup><https://www.omg.org/spec/BPMN/2.0.2/PDF>. <sup>2</sup>Ruecker, *Practical Process Automation*, O'Reilly, **2021**.

<sup>3</sup>U. Mutarraf et al., *Adv. Mech. Eng.* 10(12), doi:10.1177/1687814018808170, **2018**.

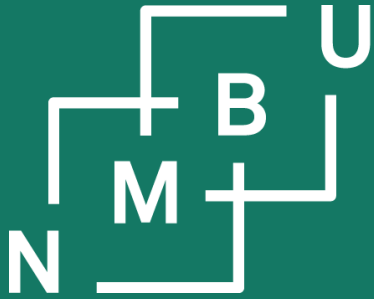
# BPMN workflows

Business Process Model and Notation is standardized<sup>1</sup> as ISO/IEC 19510:2013.



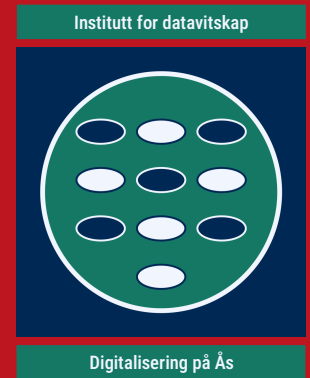
Example from A. Segatto, M. Milleri, C. Kavka, COMPOSELECTOR project deliverable 3.4, **2018**.

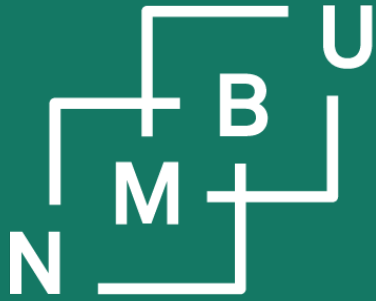
<sup>1</sup>See also the specification at <https://www.omg.org/spec/BPMN/2.0.2/PDF>.



Noregs miljø- og  
biovitenskaplege  
universitet

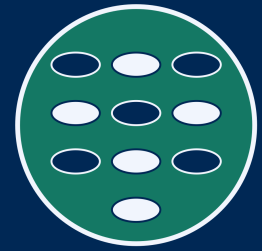
# Conclusion





Norges miljø- og  
biovitenskapelige  
universitet

Institutt for datavitenskap



Digitalisering på Ås

# INF205

## Resource-efficient programming

### 3 Concurrency

#### 3.5 Message passing with ROS

#### 3.6 Shared memory (OpenMP)

#### 3.7 Concurrent process models