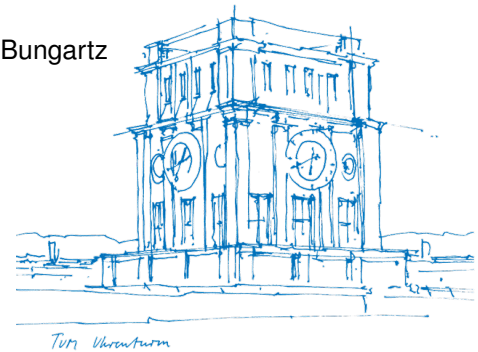
# Adaptive Dynamic Meshes for Fully-Parallel Partitioned Multi-Physics in preCICE

WCCM-ECCOMAS 2020

**Frédéric Simonis**, Benjamin Uekermann, Hans-Joachim Bungartz
Technical University of Munich

Recorded on 28. November 2020



Please watch first:

**Gerasimos Chourdakis**

**An introduction to the preCICE coupling library.**

Part I

# Meshes in preCICE

# Static interfaces in preCICE

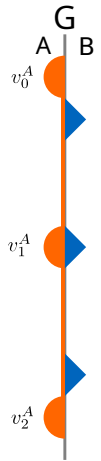
- Common geometry

# Static interfaces in preCICE

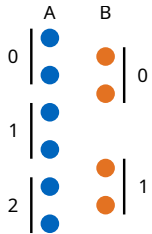
- Common geometry
- Various meshes / discretizations
- Optional connectivity information

# Static interfaces in preCICE

- Common geometry
- Various meshes / discretizations
- Optional connectivity information
- Data on vertices

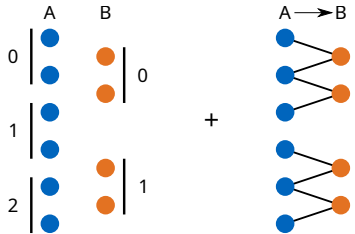


# Why static meshes?



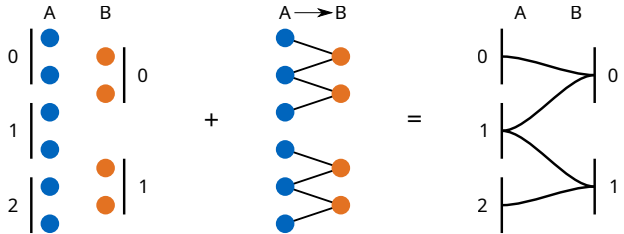

- 2 static partitioned meshes

# Why static meshes?



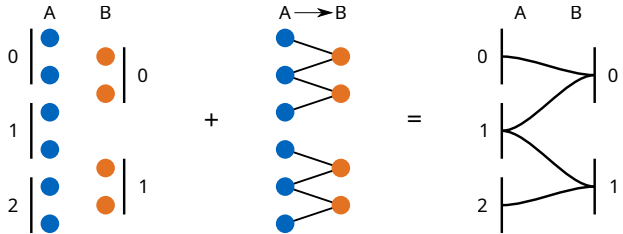

- 2 static partitioned meshes + mapping method

# Why static meshes?



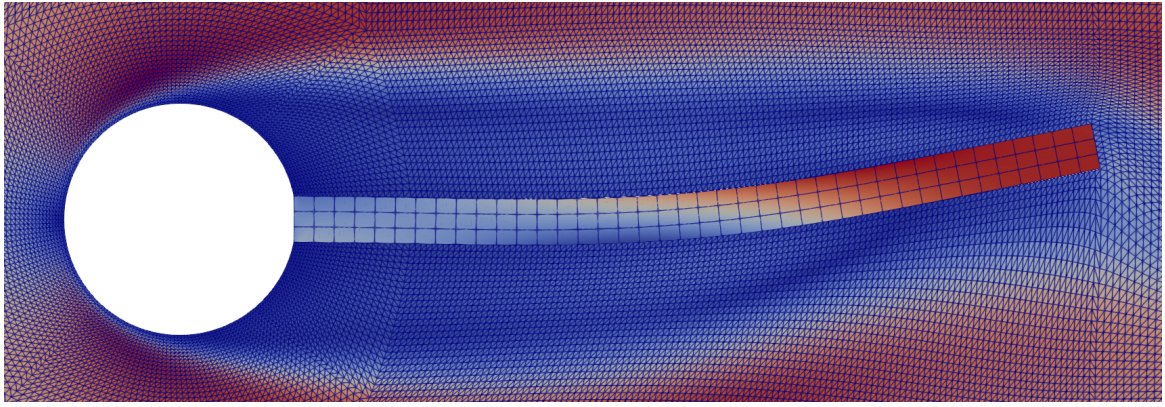

- 2 static partitioned meshes + mapping method = communication pattern

# Why static meshes?



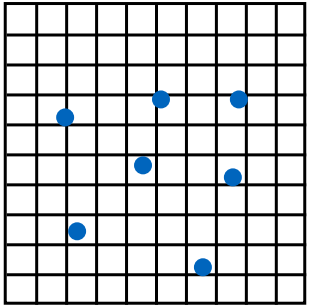

- 2 static partitioned meshes + mapping method = communication pattern
- Expensive logic runs during initialisation only

# Moving meshes to date



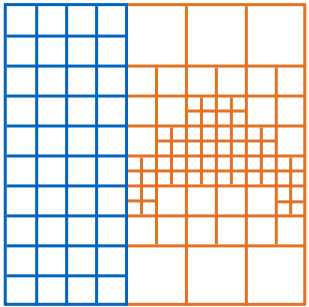
- Arbitrary Lagrangian-Eulerian method
- Vertex displacements and data on static mesh
- preCICE maps in reference domain

# Why dynamic-adaptive meshes?



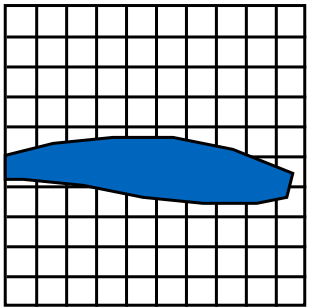
Volume coupling of particle codes.
Moving particles as a dynamic mesh.

# Why dynamic-adaptive meshes?



Interface coupling with dynamic-adaptive codes.
Profit from mesh refinement directly at the interface.

# Why dynamic-adaptive meshes?



Immersed boundary methods.
Freely moving interface with partial mapping.

# User perspective

Current API does not allow changes to the mesh after initialization.

```
precice::SolverInterface si;
si.setMeshVertex();
si.initialize();
while (si.isCouplingOnGoing()) {
  si.readData();
  solve();
  si.writeData();
  si.advance();
}
```

# Future API - reset mesh

Brute-force API allows to completely redefine a mesh.

```
precice::SolverInterface si;
si.setMeshVertex();
si.initialize();
while (si.isCouplingOnGoing()) {
  si.readData();
  refine();
  si.resetMesh();
  si.setMeshVertex();
  solve();
  si.writeData();
  si.advance();
}
```

# Future API - update mesh

Anticipated API allows to remove and add vertices on the fly.

```
precice::SolverInterface si;
si.setMeshVertex();
si.initialize();
while (si.isCouplingOnGoing()) {
  si.readData();
  refine();
  si.removeMeshVertex();
  si.setMeshVertex();
  solve();
  si.writeData();
  si.advance();
}
```

# Part II

# **Challenges**

# Changing fundamental assumptions

1. What to reinitialize?
2. When to reinitialize?
3. How to orchestrate reinitialization?
4. How to reinitialize components?

# 1. What to reinitialize?

`initialize()`

1. Connect coupling partners
2. Determine communication pattern
3. Establish further connections
4. Setup watchpoints
5. Initialize coupling schemes and acceleration
6. Perform initial data exchange

**Steps 2-6 depend on mesh**

# 2. When to reinitialize?

`advance()`

1. Update time window
2. Map written data
3. Advance coupling schemes
   *acceleration, communication*
4. Map read data
5. Handle exports

**Steps 2-5 depend on mesh**

# 3. How to orchestrate reinitialization?

**Internal Orchestration**

*How do we keep all ranks of a participant in a consistent state?*

**External Orchestration**

*How do we keep all participants in a consistent state?*

# 3. How to orchestrate reinitialization?

**Internal Orchestration**

*How do we keep all ranks of a participant in a consistent state?*

**External Orchestration**

*How do we keep all participants in a consistent state?*

# Internal orchestration

**Each rank of a participant**

- knows what changed locally.
- needs to know what changed globally.

**Handshake at the begin of advance**

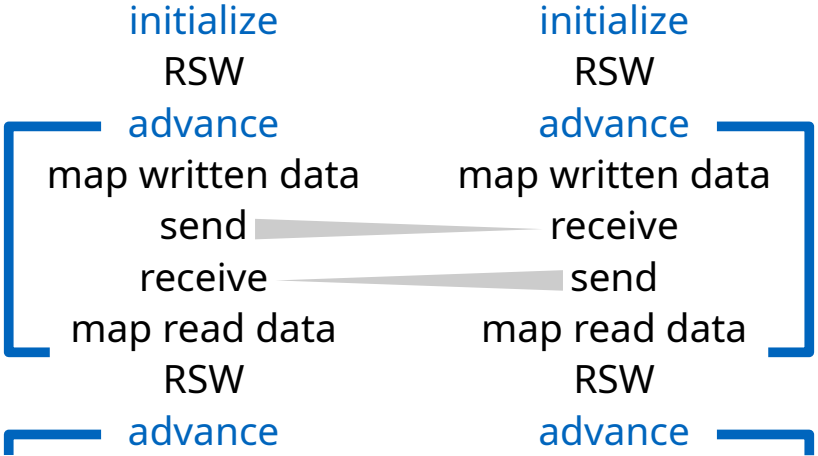
# External orchestration

**Each participant**

- knows what changed locally.
- needs to know what changed in connected participants.

**Handshake scheme depends on the coupling scheme**

## External orchestration for parallel coupling

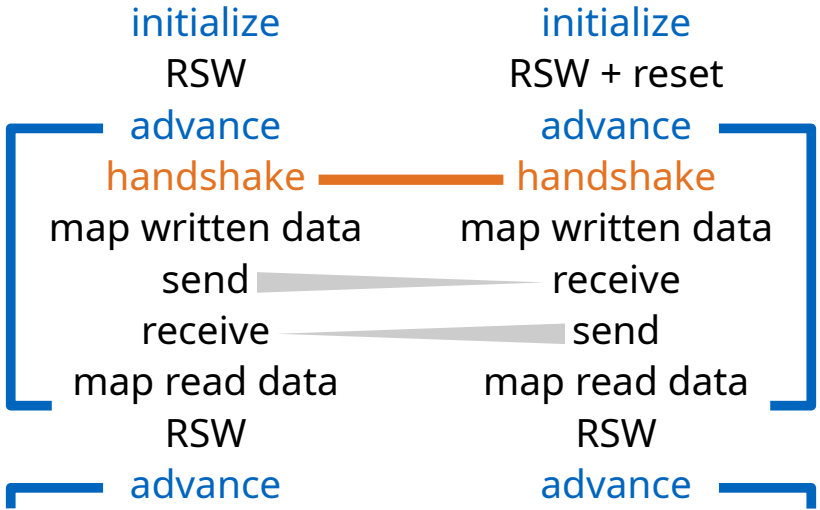



*RSW = Read Solve Write*

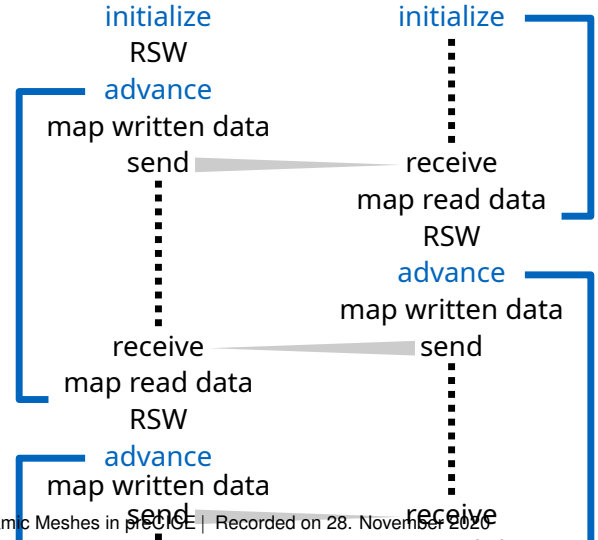
# External orchestration for parallel coupling


*RSW = Read Solve Write*

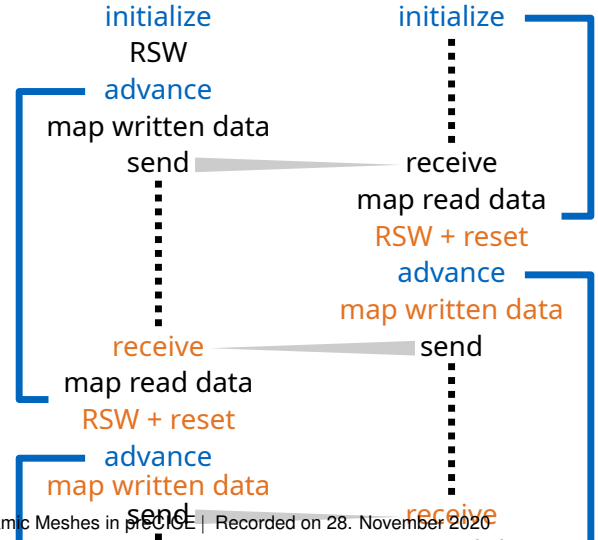
# External orchestration for parallel coupling



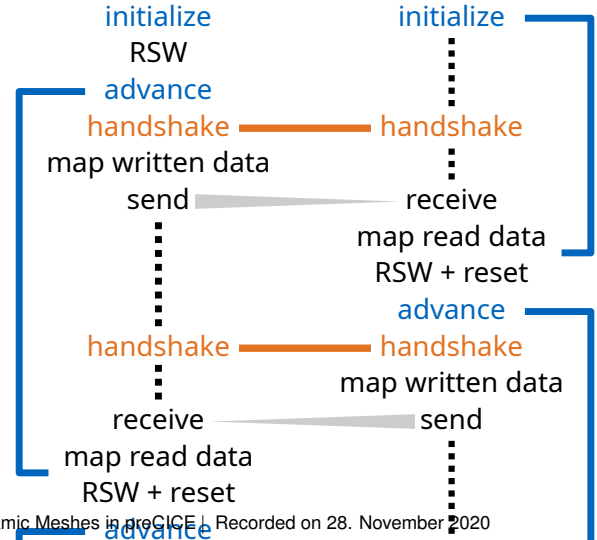
# External orchestration for serial coupling



# External orchestration for serial coupling



# External orchestration for serial coupling



# Orchestration - efficiency concerns

## API-driven mesh changes

- Reset possible at every time window
- Every rank of every participant may reset
- Cannot skip handshakes
- Unnecessary communication

# Orchestration - efficiency concerns

## API-driven mesh changes

- Reset possible at every time window
- Every rank of every participant may reset
- Cannot skip handshakes
- Unnecessary communication

## Limit possible mesh-changes in configuration

- Mark mesh as adaptive
- Allows to skip handshake

# 4. How to reinitialize components?

**Old assumption**

*It executes once during initialisation, thus runtime is not important.*

**New assumption**

*It may execute once every time window, thus runtime is crucial.*

**Thus many components need redesigning.**

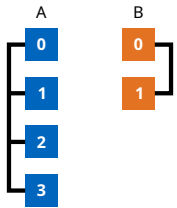
# Impact on components - examples

- Acceleration
- Mapping methods
- Communication

# Impact on communication

## The situation

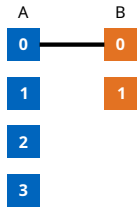
- Connections between ranks of a participant won't change.
- Main connections between participants won't change.
- Connections between ranks of participants may change.



# Impact on communication

## The situation

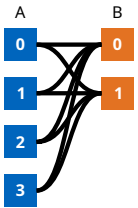
- Connections between ranks of a participant won’t change.
- Main connections between participants won’t change.
- Connections between ranks of participants may change.



# Impact on communication

## The situation

- Connections between ranks of a participant won't change.
- Main connections between participants won't change.
- Connections between ranks of participants may change.



# Impact on communication

## Brute-force solution

- Keep main connections
- Close connections between ranks
- Deduce new communication pattern
- Establish required connections

**Costly for massively parallel simulations.**

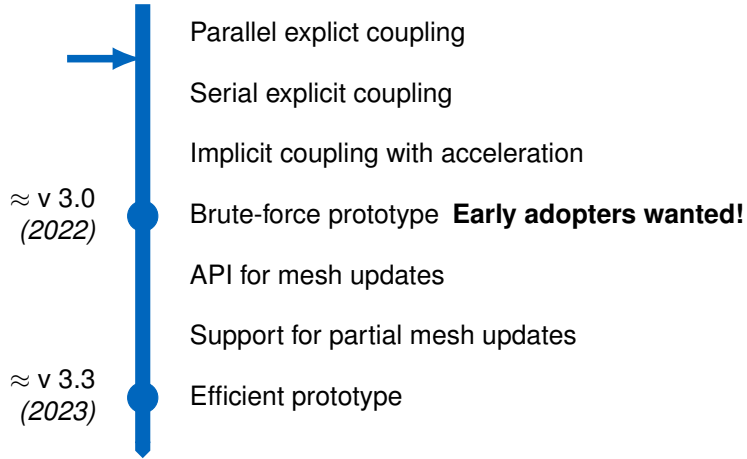
# Impact on communication

## Anticipated solution

- Deduce new communication pattern
- Establish required connections
- Close unnecessary connections

**Requires a connection manager that handles complete connection topology.**

# Roadmap


Parallel explict coupling
Serial explicit coupling
Implicit coupling with acceleration
≈ v 3.0
(2022)
Brute-force prototype **Early adopters wanted!**
API for mesh updates
Support for partial mesh updates
≈ v 3.3
(2023)
Efficient prototype

# Summary

- Requirement for many methods
  Dynamic-adaptive codes, immersed boundary
- Efficiency is a major concern
  Acceleration data, mapping methods, handshakes
- Orchestration is complex
  Multiple coupling schemes

# Thank you for your attention!

**Get in touch**

| | |
|---|---|
| **GitHub** | fsimonis |
| **E-Mail** | simonis@in.tum.de |

**preCICE**

| | |
|---|---|
| **GitHub** | precice |
| **Gitter** | gitter.im/precice/Lobby |
| **Discourse** | precice.discourse.group |