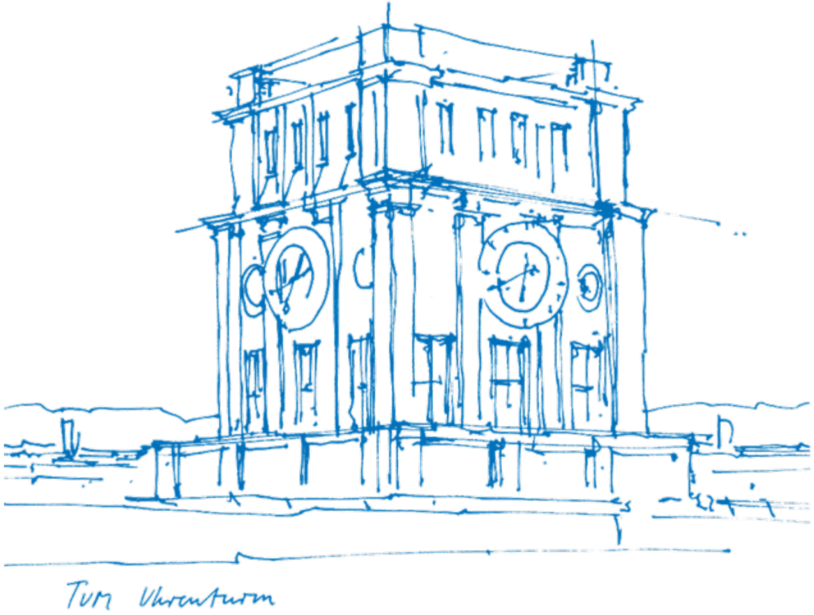
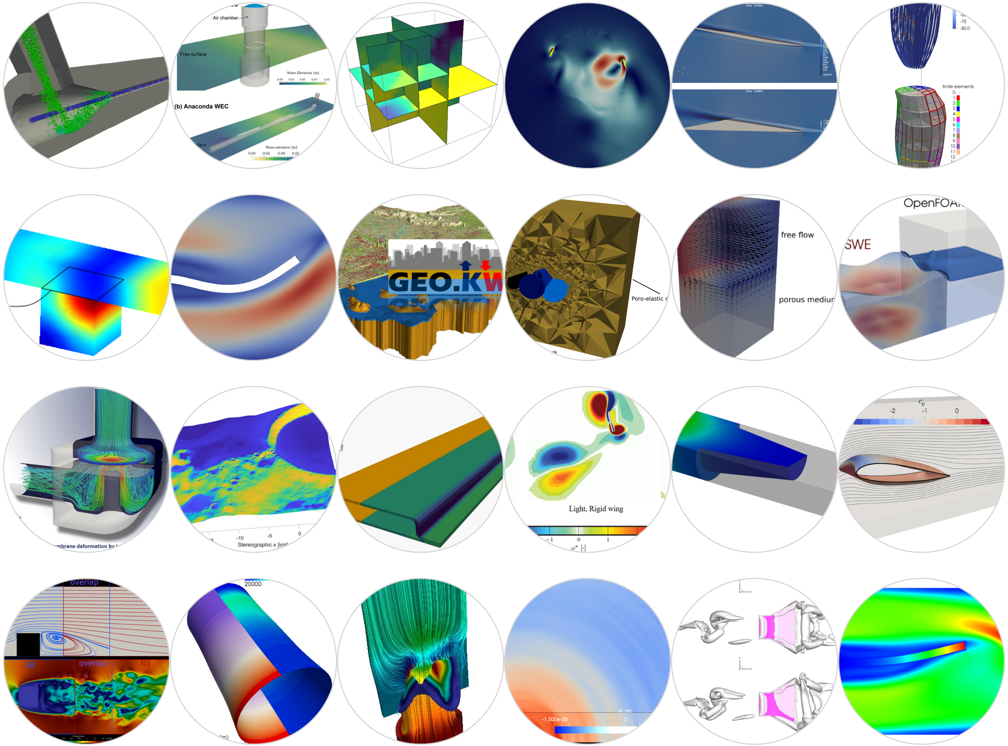
Dirichlet-Neumann waveform iteration with the coupling library preCICE

Benjamin Rodenberg, B. Uekermann, M. Schulte, H.J. Bungartz

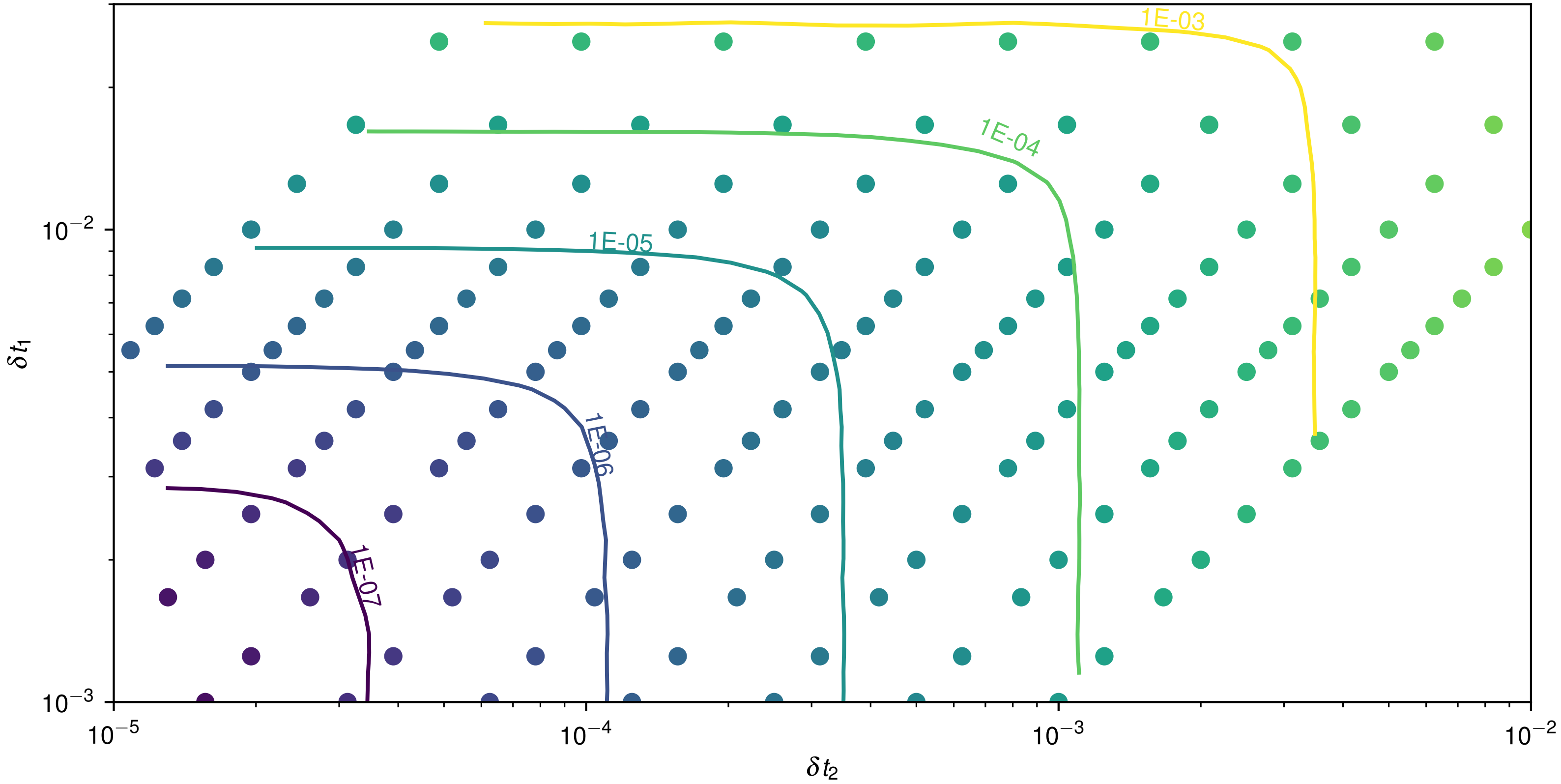
DD28, Jan. 31, 2024



Technical University of Munich, Department of Informatics


Air chamber
Free surface
(b) Anaconda WEC
finite elements
free flow
SWE
Light, Rigid wing
overlap
overlap
NS
20000

Partitioned = heterogeneous



Outline

What is preCICE?

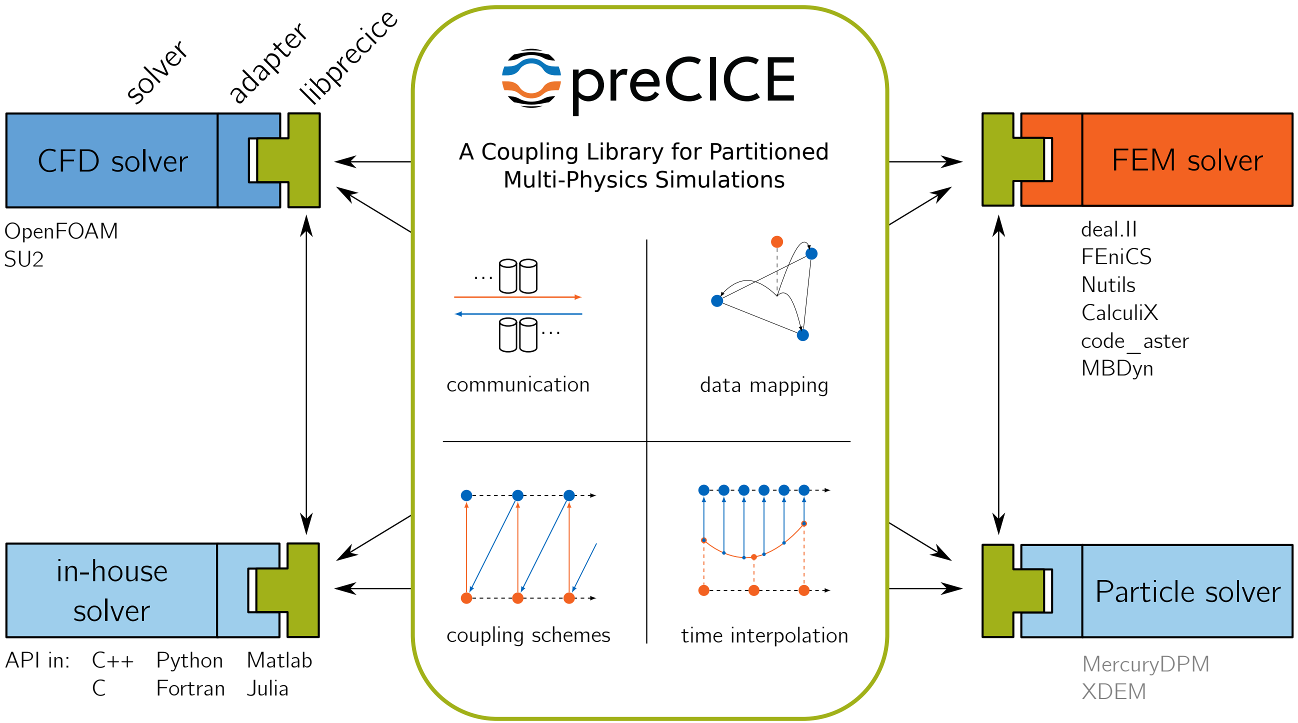
Why preCICE relies on domain decomposition

preCICE v3: waveform iteration ($=$ PinT)

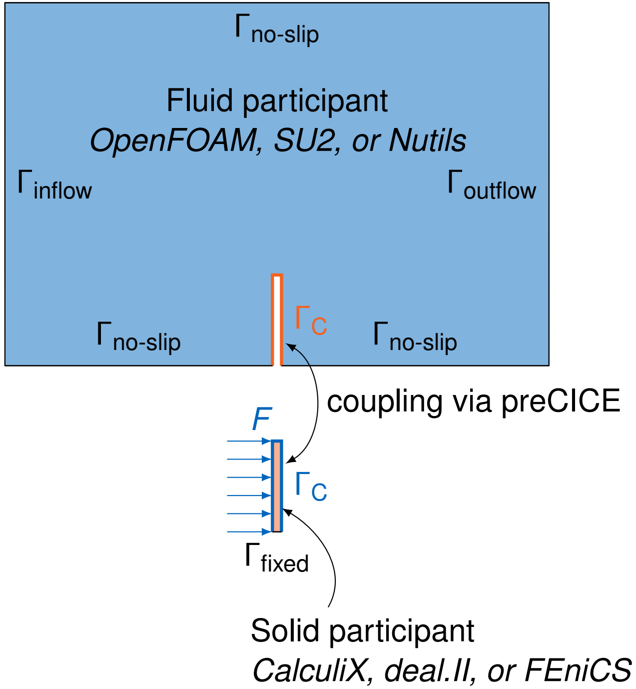
Partitioned heat equation

preCICE user interface + waveform iteration

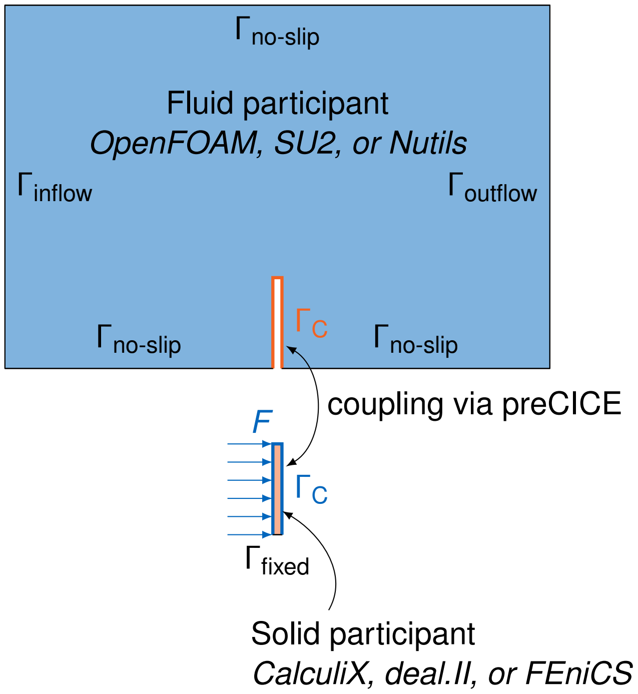
What is preCICE?



tutorials/perpendicular-flap



Why preCICE relies on domain decomposition



Divide

- OpenFOAM $\neq$ CalculiX
- preCICE $=$ *partitioned* multiphysics
- Dirichlet-Neumann $=$ black box

Conquer

- Fluid: $\mathscr{F}(d) = u$
- Solid: $\mathscr{S}(u) = d$

Boundary response maps

(= Poincaré-Steklov operator)

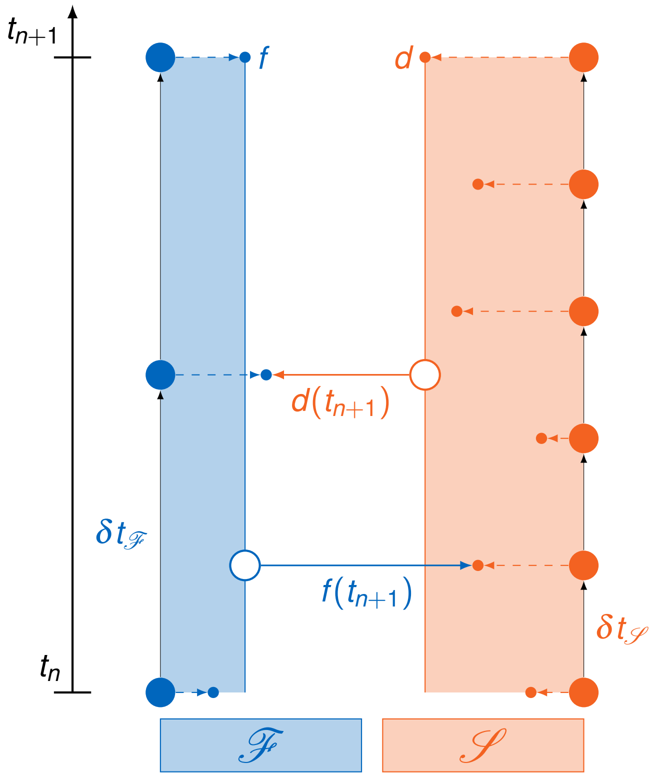
Combine

- $\mathscr{F}(\mathscr{S}(u^k)) = \tilde{u}^k$
- $\tilde{u}^k \xrightarrow{\mathscr{A}} u^{k+1}$

Picard iteration + acceleration

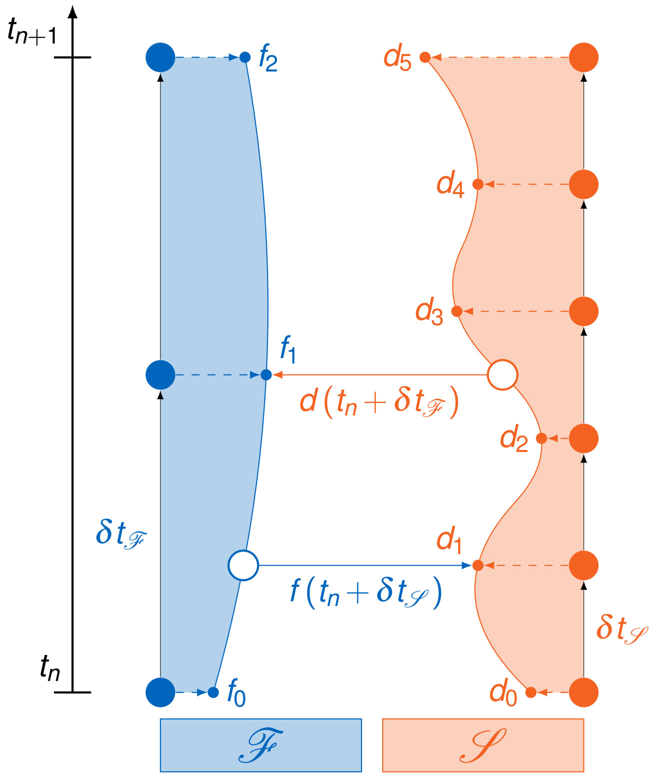
preCICE v2: single-value coupling

```
<data:vector name="Force" />
<data:vector name="Displ" />
...
<coupling-scheme:serial-implicit>
  <participants first="Fluid" second="Solid" />
  <exchange
    data="Force"
    from="Fluid"
    to="Solid" />
  <exchange
    data="Displ"
    from="Solid"
    to="Fluid" />
  ...
</coupling-scheme:serial-implicit>
```

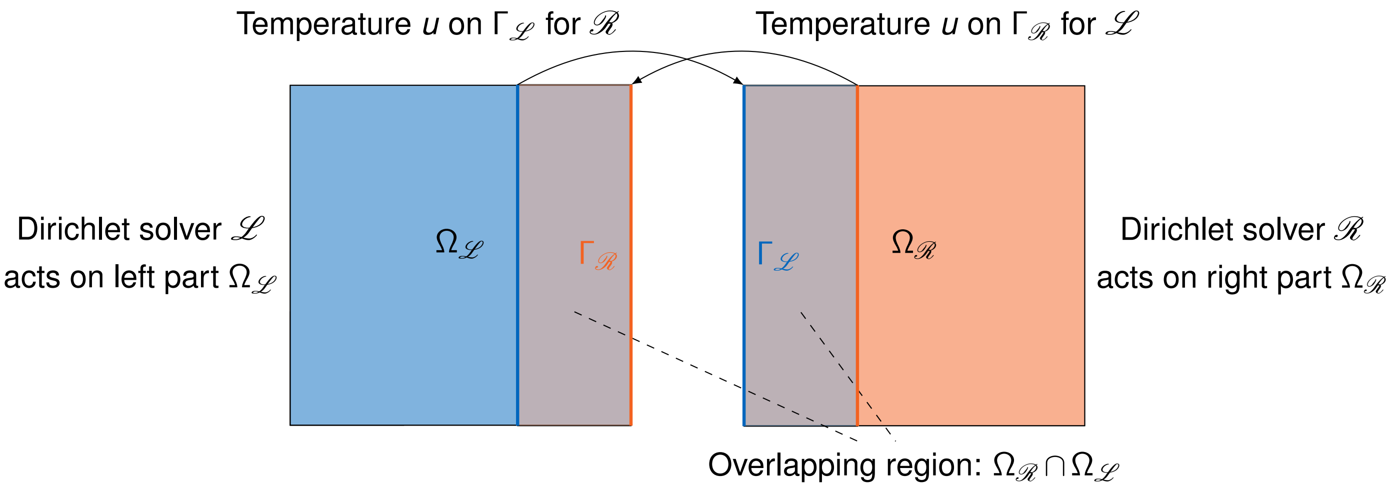


preCICE v3: waveform iteration (= PinT)

```
<data:vector name="Force" waveform-degree="2" />
<data:vector name="Displ" waveform-degree="3" />
...
<coupling-scheme:serial-implicit>
  <participants first="Fluid" second="Solid" />
  <exchange
    data="Force"
    from="Fluid"
    to="Solid"
    substeps="true" />
  <exchange
    data="Displ"
    from="Solid"
    to="Fluid"
    substeps="true" />
  ...
</coupling-scheme:serial-implicit>
```



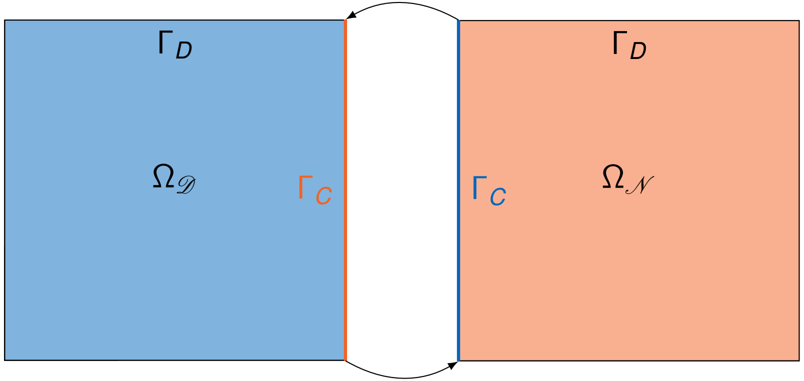
Partitioned heat equation



Partitioned heat equation

Dirichlet boundary condition on Γ_C for $\mathscr{D}$

temperature u_C



Dirichlet solver $\mathscr{D}$

(FEniCS, OpenFOAM, or Nutils)

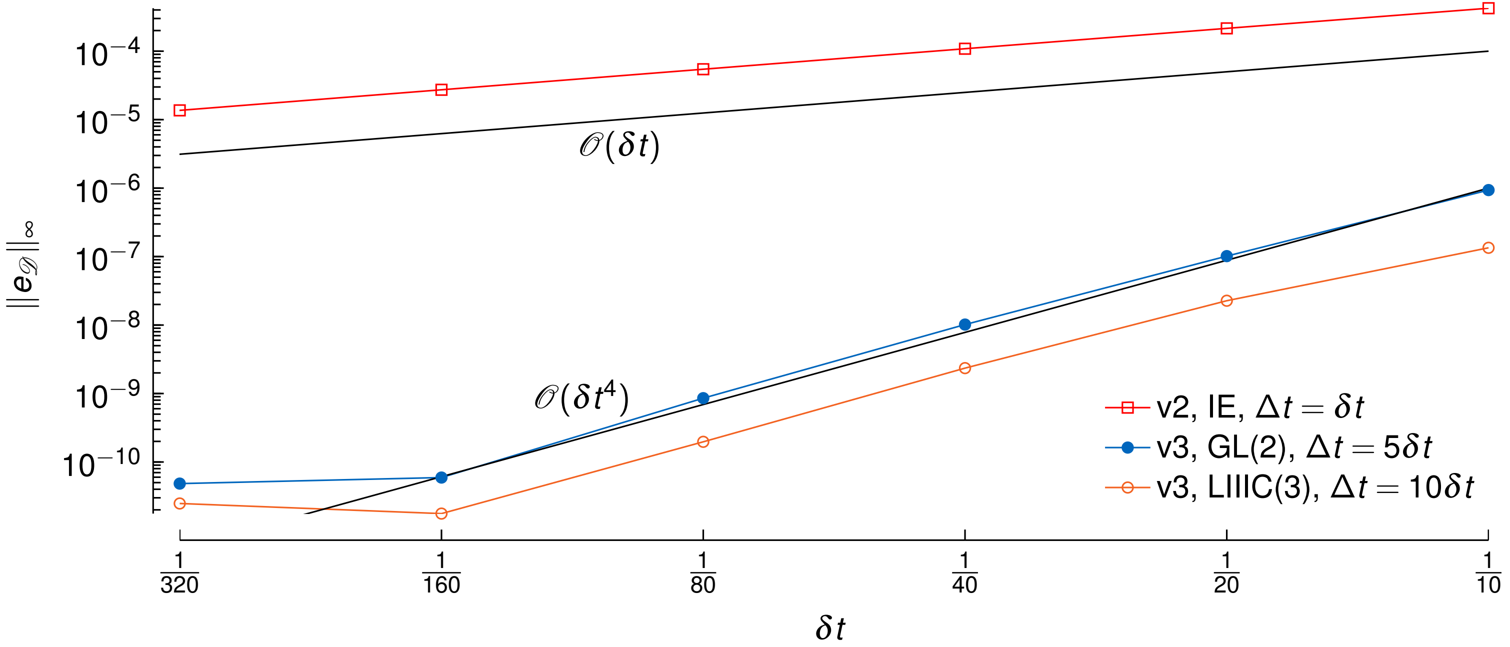
Neumann solver $\mathscr{N}$

(FEniCS, OpenFOAM, or Nutils)

heat flux $q_C = \vec{\nabla} u \,|_{\Gamma_C}$

Neumann boundary condition on Γ_C for $\mathscr{N}$

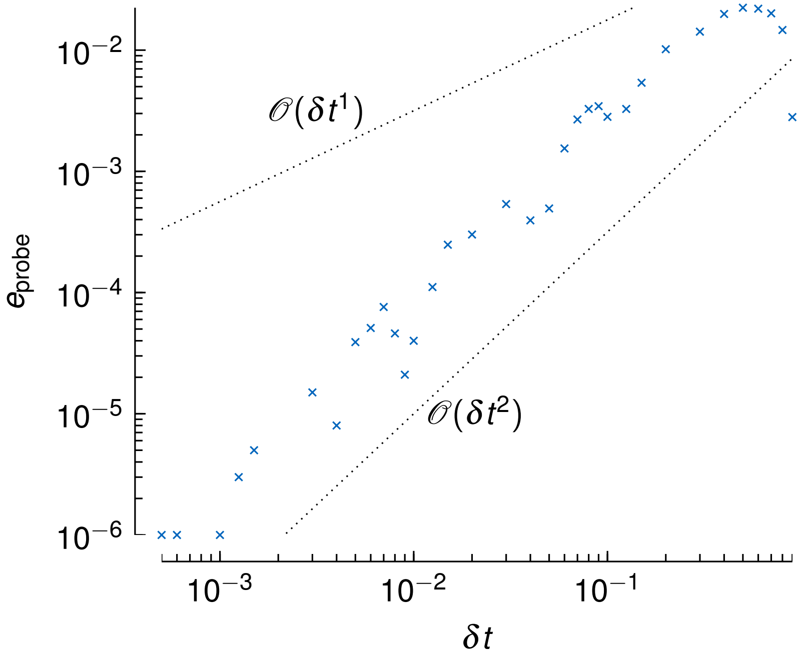
Partitioned heat equation



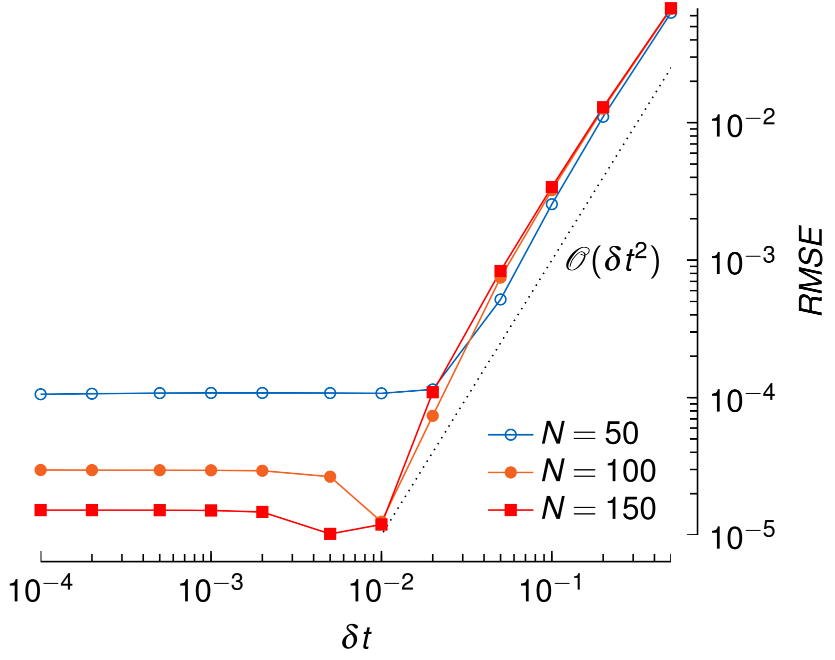
Bachelor's thesis by Niklas Vinnitchenko *Evaluation of Higher-Order Coupling Schemes with FEniCS-preCICE*[1]

[1] https://github.com/precice/tutorials/pull/415

Outlook: FS (atm) without I



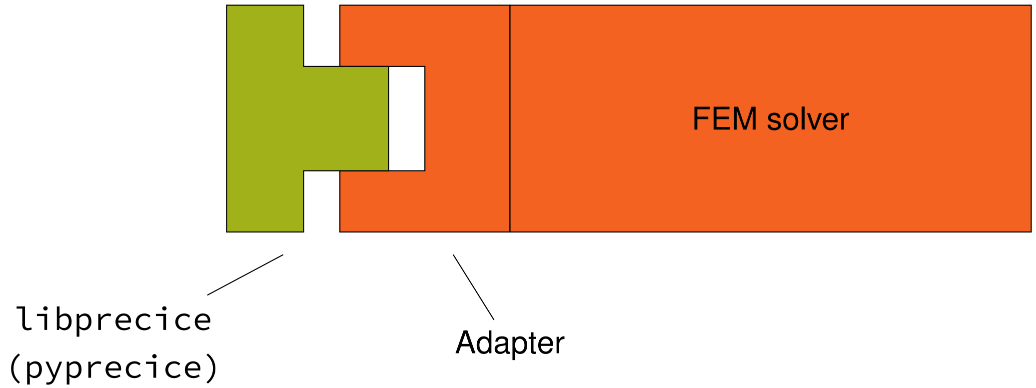
CalculiX beam (Probe at beam tip, compare to $\delta t = 10^{-5}$)



OpenFOAM Taylor-Green vortex

Guided research project by Marc Amorós Trepat *Review of higher-order time stepping schemes in open-source solvers*

preCICE user interface + waveform iteration



Goal: Black-box + higher-order interpolation + waveform iteration

- Numerics: Accuracy, convergence order, energy conservation
- API Design: Simple user interface!
- Move as much waveform iteration logic *inside* preCICE as possible
- preCICE v2.5 is the current version, preCICE v3 + new API is almost released

$\mathscr{F}(d) = u$ in python: API example

```python
# example: FSI coupling, perspective of fluid solver $\mathscr{F}$:
participant = precice.Participant("Fluid", "precice-config.xml")

# leaving out coupling mesh and data initialization

participant.initialize()

while participant.is_coupling_ongoing():
  # store checkpoint, if needed
  dt = participant.get_max_time_step_size()
  displacement = participant.read_data(dt)  # Read displacement at $t_n+\Delta t$: $d_{n+1}$
  forces = forces + dt * dfdt(displacements)  # Implicit Euler
  participant.write_data(forces)
  participant.advance(dt)
  # read checkpoint, if needed

participant.finalize()
```

API with RK4, with subcycling

```python
# example: FSI coupling, perspective of fluid solver $\mathcal{F}$:
# ...
participant.initialize()

while participant.is_coupling_ongoing():
  # store checkpoint, if needed
  precice_dt = participant.get_max_time_step_size()  # until end of window
  solver_dt = time_stepper.get_max_dt()  # stability, adaptivity
  dt = np.min([precice_dt, solver_dt])  # actual time step size $\delta t$
  # time_stepper represents s-stage RK scheme
  ts = time_stepper.rhs_eval_points(dt)  # $t_n+c_1, t_n+c_2 \ldots, t_n+c_s$
  displacements = [participant.read_data(t) for t in ts]  # $d(t_n+c_1), d(t_n+c_2) \ldots, d(t_n+c_s)$
  forces = time_stepper.do_step(displacements, dt)  # $f_{n+1} = f_n + \delta t \sum_{i=1}^{s} b_i k_i$
  participant.write_data(forces)
  participant.advance(dt)
  # read checkpoint, if needed

participant.finalize()
```

Conclusion

State of development

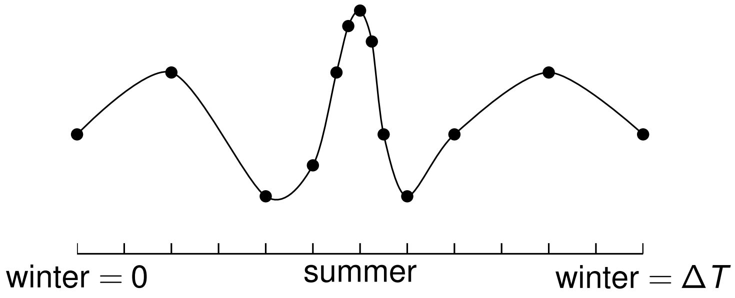
- Achievements: Black-box waveform iteration + usable API (+ multirate)
- Goal: Release preCICE v3. (we are very very close)
- If you want to try experimental version now: `https://github.com/precice/precice/pull/1914`

Where to be careful:

- Requirement: High order interpolation needs subcycling
- Restriction: Only simple acceleration with substeps, `https://github.com/precice/precice/pull/1834`

Many questions:

- Synchronization, subcycling, and performance?
- Fluid-structure interaction case?
- Adaptivity (inside window / across windows)



Community

preCICE Workshop 2023@Munich

Stay in touch?

- `precice.org/community`
- `precice.discourse.group`

Conferences

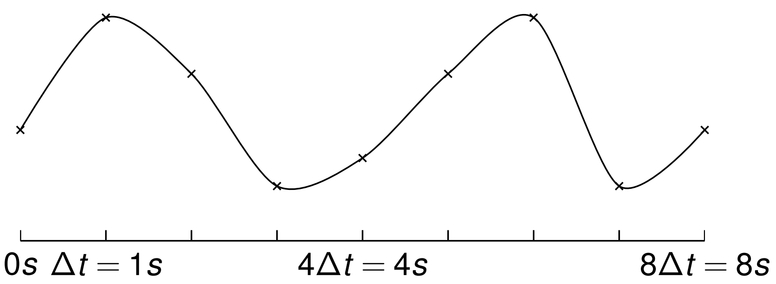
- WCCM + preCICE course
  July 2024@Vancouver
  Deadline: this Friday
  preCICE MS[1]
- preCICE Workshop
  Sept 2024@Stuttgart
- COUPLED
  2025@Sicily

[1] Multi-Physics and Multi-Scale Simulations with the Coupling Library preCICE (0415)

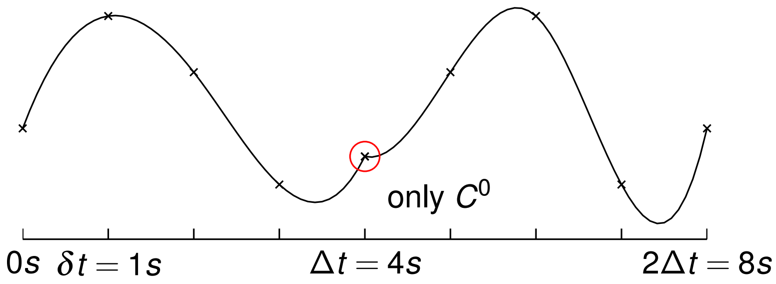
Subcycling

- Time window size $\Delta t \geq$ time step size δt_1 and δt_2.
- Do n time steps in window: $\Delta t = n_1 \delta t_1 = n_2 \delta t_2$
- Allows to create BSpline of degree $n - 1$. (Goal reached: Something better than linear interpolation)
- Restriction: Only use data of current window!
- Larger window + subcycling has impact on number of QN iterations[1]

Without subcycling



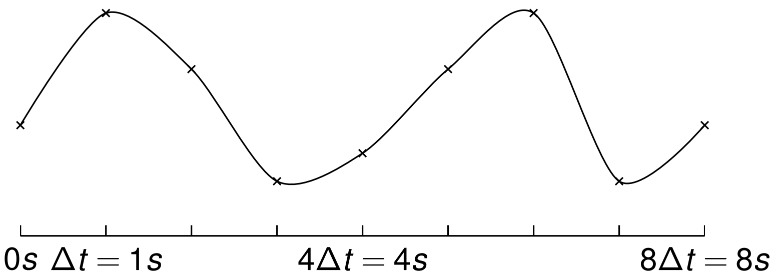
With subcycling (third order BSpline)



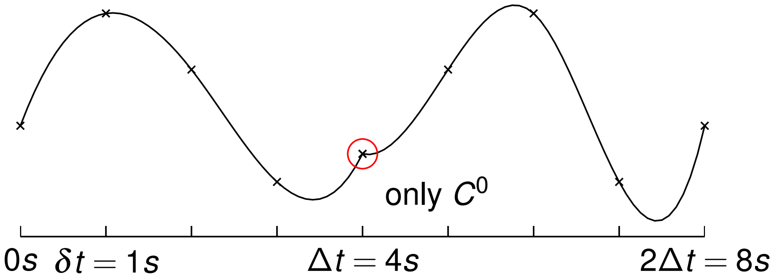
[1] Rüth, B, Uekermann, B, Mehl, M, Birken, P, Monge, A, Bungartz, H-J. Quasi-Newton waveform iteration for partitioned surface-coupled multiphysics applications. Int J Numer Methods Eng. 2021; 122: 5236– 5257. `https://doi.org/10.1002/nme.6443`

QN iterations

Without subcycling



With subcycling (third order BSpline)

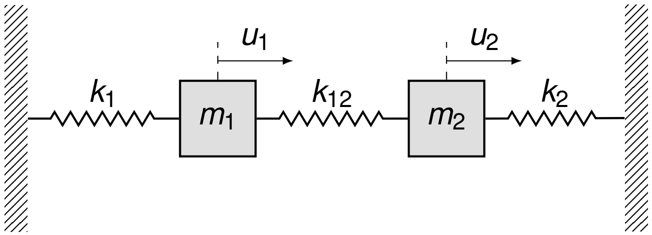


rQN-WI | Δt	0.5	0.1
WI$(1,1;1)$	7.85	5.45
WI$(5,5;1)$	10.95	7.48

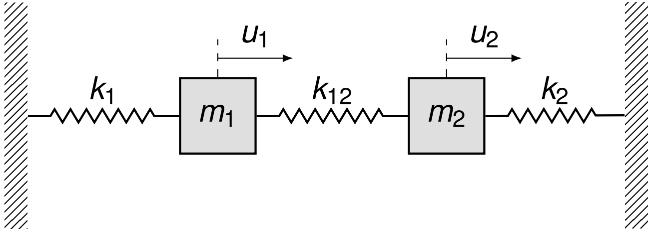
rQN-WI means we only use the data at the end of the window for Quasi-Newton. Different example case, but similar implementation. More possibilities shown in[1].

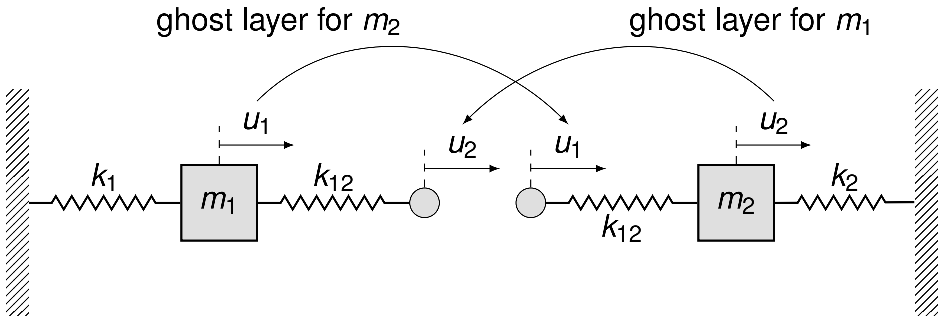
[1] Rüth, B, Uekermann, B, Mehl, M, Birken, P, Monge, A, Bungartz, H-J. Quasi-Newton waveform iteration for partitioned surface-coupled multiphysics applications. Int J Numer Methods Eng. 2021; 122: 5236– 5257. `https://doi.org/10.1002/nme.6443`

Oscillator example: Setup



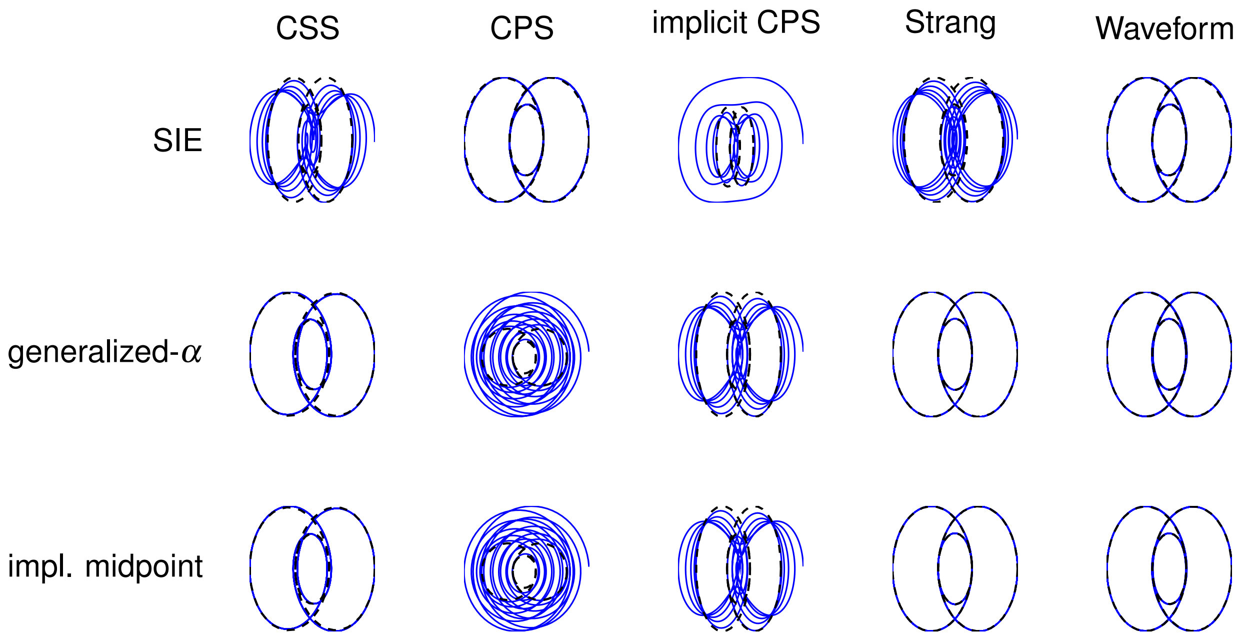
Oscillator example: Setup





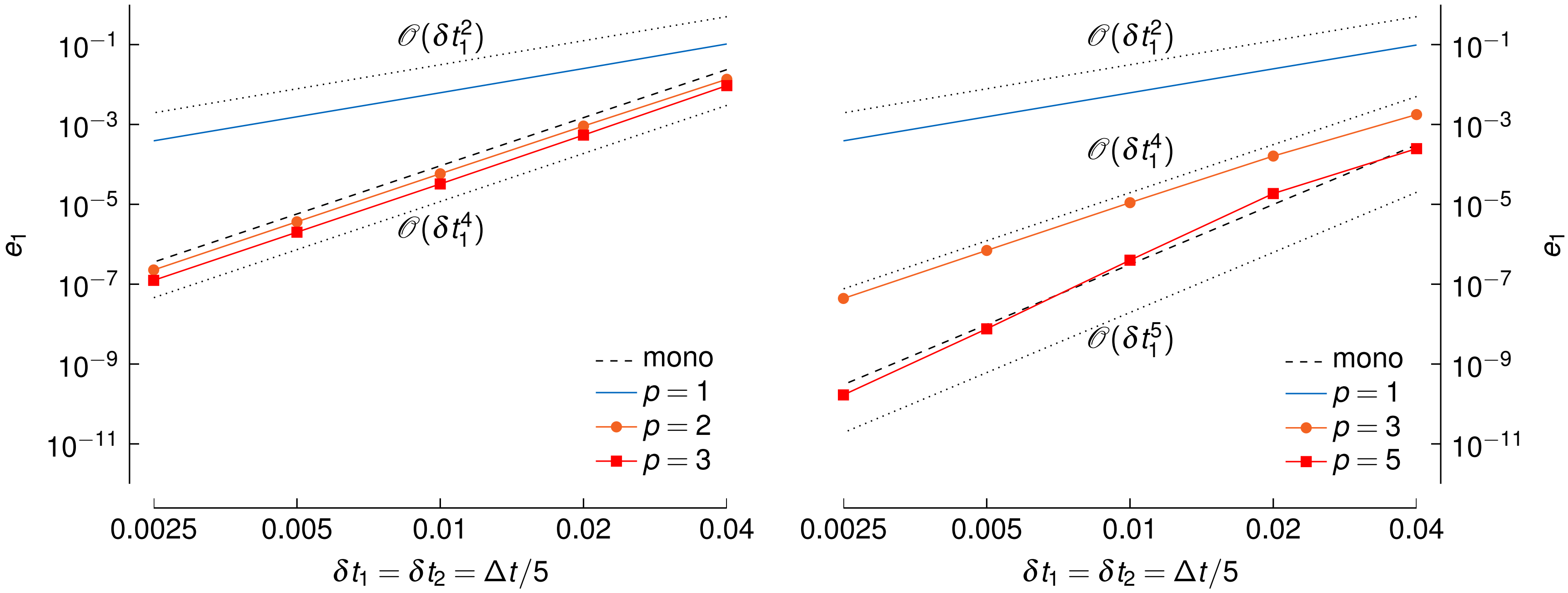
Not Dirichlet-Neumann coupling, but overlapping Schwarz for this setup. Still: Same idea.

Oscillator example: Energy conservation



A Simple Test Case for Convergence Order in Time and Energy Conservation of Black-Box Coupling Schemes. 2022.

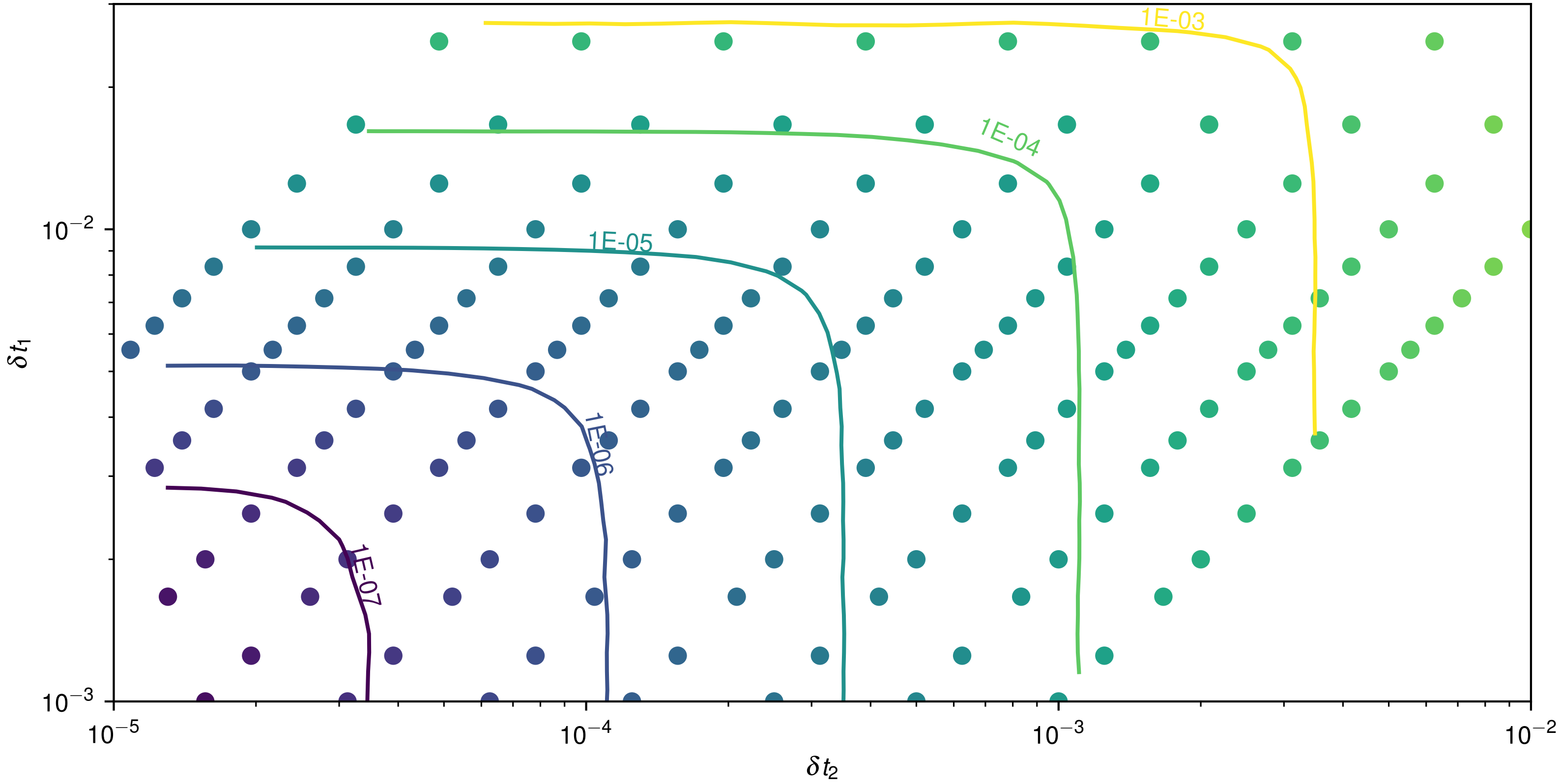
Oscillator example: Higher-order TS



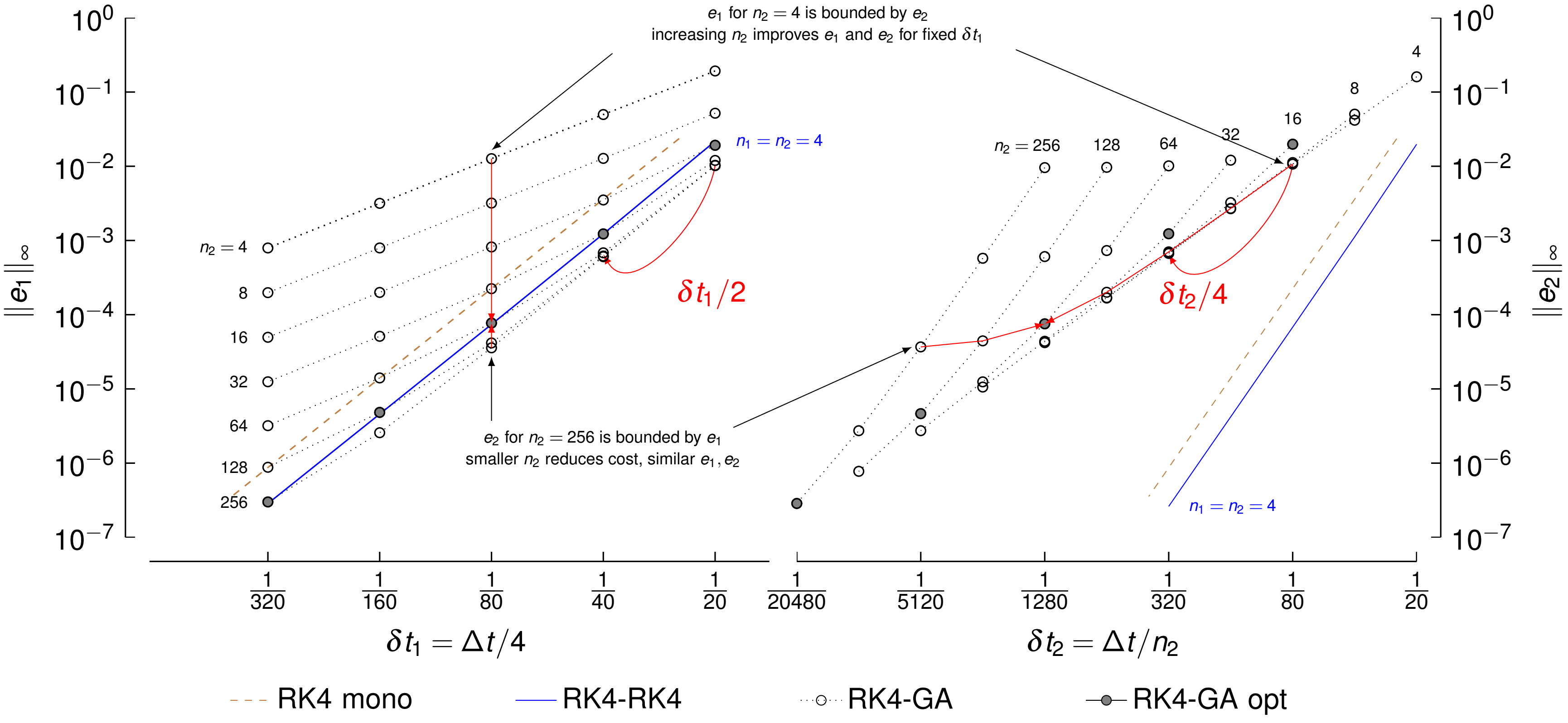
classic Runge-Kutta method

Radau IIA

Oscillator example: Different schemes, different δt



Oscillator example: Different schemes, different δt



RK4 in code

```
# determine time step size δt for this time step
dt = participant.get_max_time_step_size()

# compute stages k_i and evaluate waveform at times c_iδt
k1 = A.dot(x) + participant.read_data("Displ", 0.0 * dt)
k2 = A.dot(x + k1 * 0.5 * dt) + participant.read_data("Displ", 0.5 * dt)
k3 = A.dot(x + k2 * 0.5 * dt) + participant.read_data("Displ", 0.5 * dt)
k4 = A.dot(x + k3 * 1.0 * dt) + participant.read_data("Displ", 1.0 * dt)
# assemble new solution
x_new = x + dt / 6 * (k1 + 2 * k2 + 2 * k3 + k4)

# do writing
displ = extract_force(x_new)
participant.write_data("Force", cpl_write)
# end time step of size δt
precice_dt = participant.advance(dt)
```

API for subcycling

Read interpolated data from current time t_n:

```
displacement = precice.read_data(dt)
```

No more substeps:

```
precice.advance(precice_dt)
```

t_n

```
dt = 0
```

$t_{n+1} = t_n + \Delta t$

```
dt = precice_dt
```

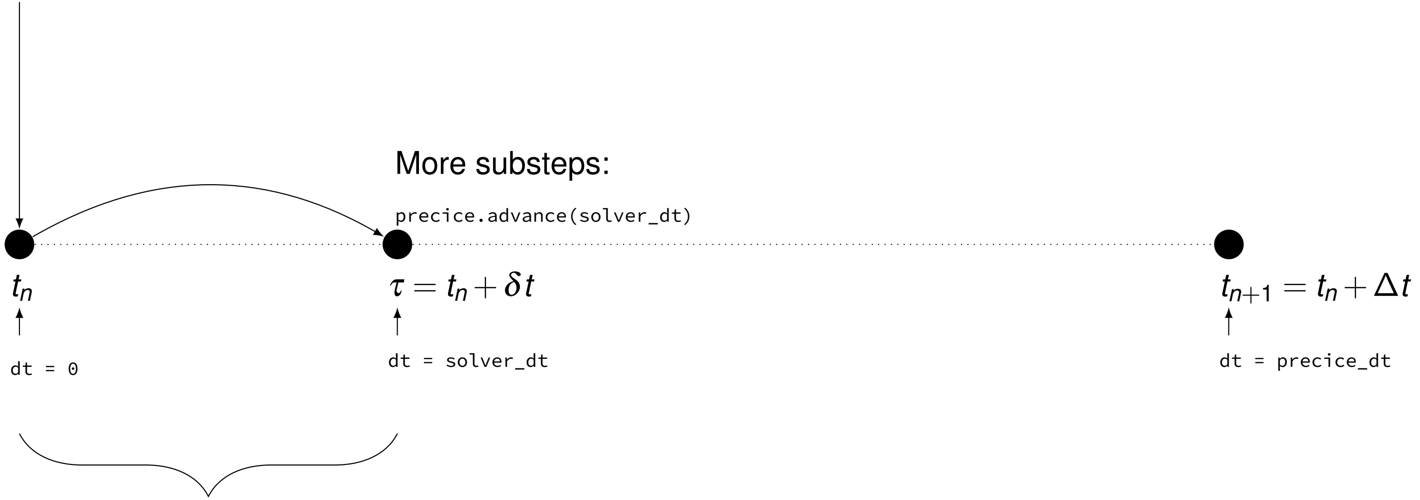
Complete time window size Δt

```
precice_dt = precice.get_max_time_step_size()
```

API for subcycling

Read interpolated data from current time t_n:

```
displacement = precice.read_data(dt)
```



Solver time step size δt

```
solver_dt < precice_dt
```

API for subcycling

Read interpolated data from current time t_n:

```
displacement = precice.read_data(dt)
```