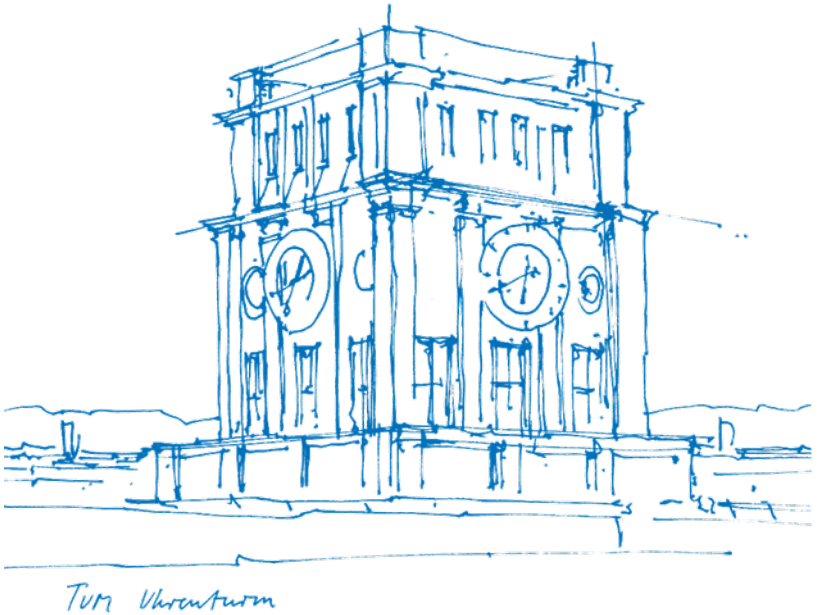
Waveform iteration in partitioned multiphysics with preCICE

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

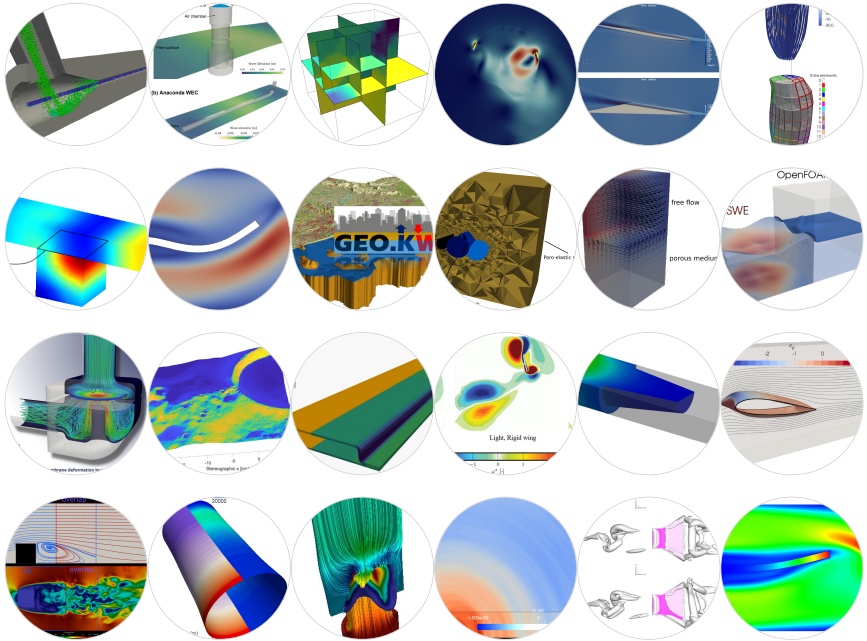
PinT 2024, Feb. 6, 2024



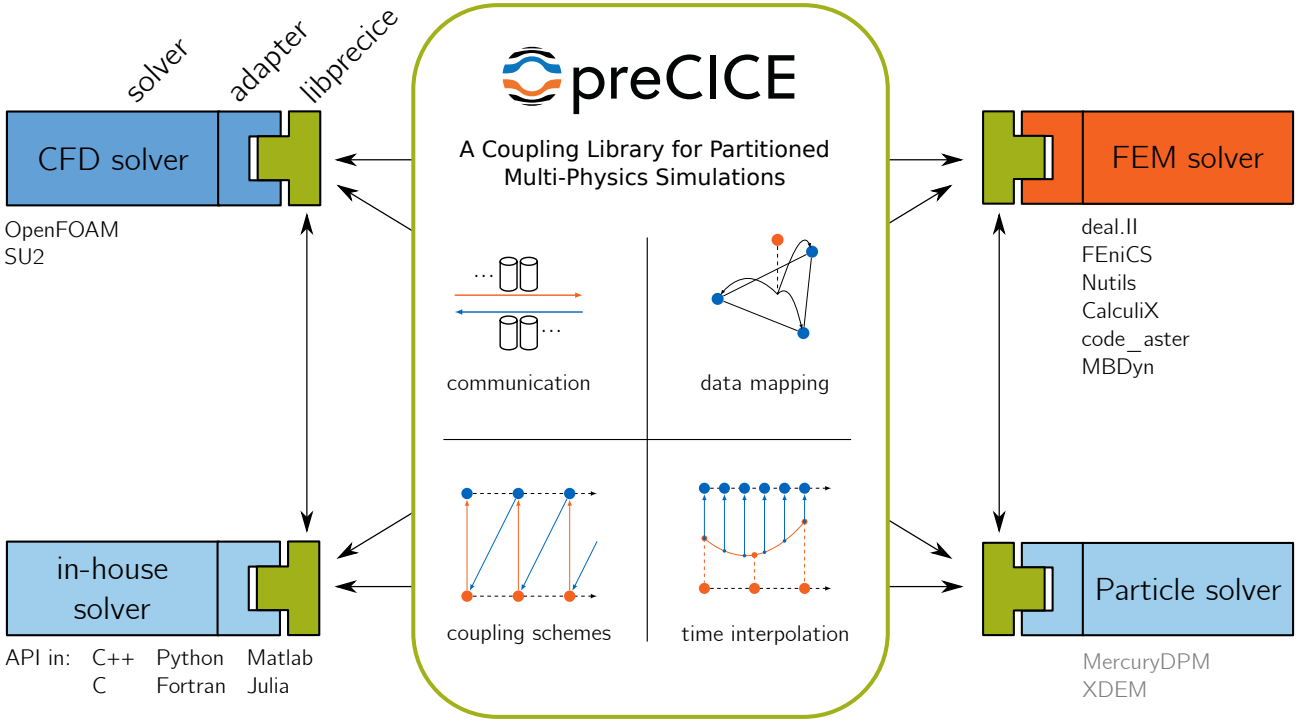
Technical University of Munich, School for Computation, Information and Technology

Outline

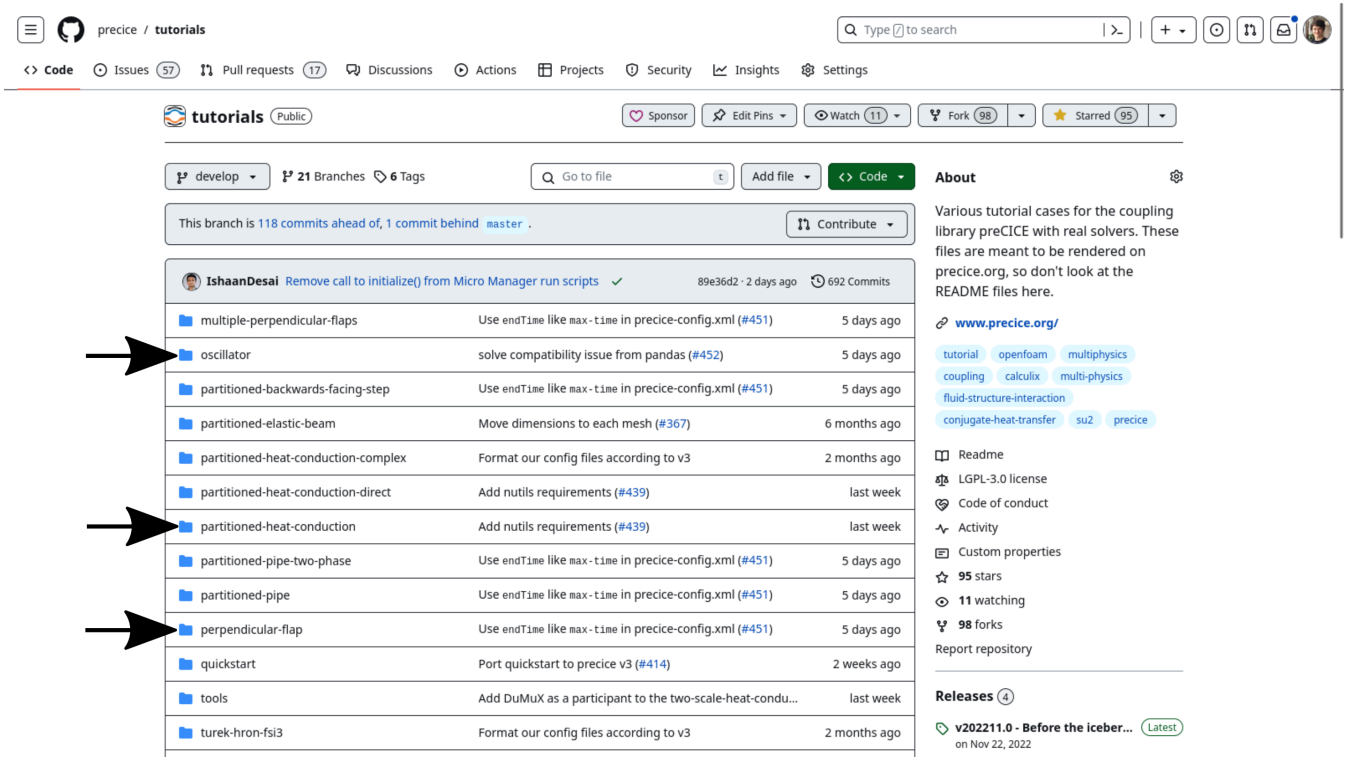
precice.org/community-projects



preCICE overview



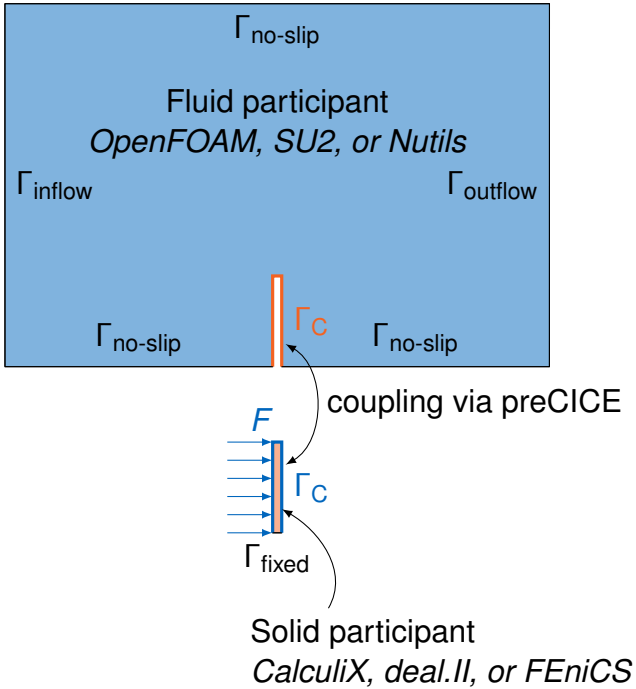
github.com/precice/tutorials



tutorials/perpendicular-flap

DEMO TIME

tutorials/perpendicular-flap



Divide

- OpenFOAM $\neq$ FEniCS
- Dirichlet-Neumann $=$ black box

Conquer

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

Boundary response maps

(= Poincaré-Steklov operator)

Combine

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

Picard iteration + acceleration

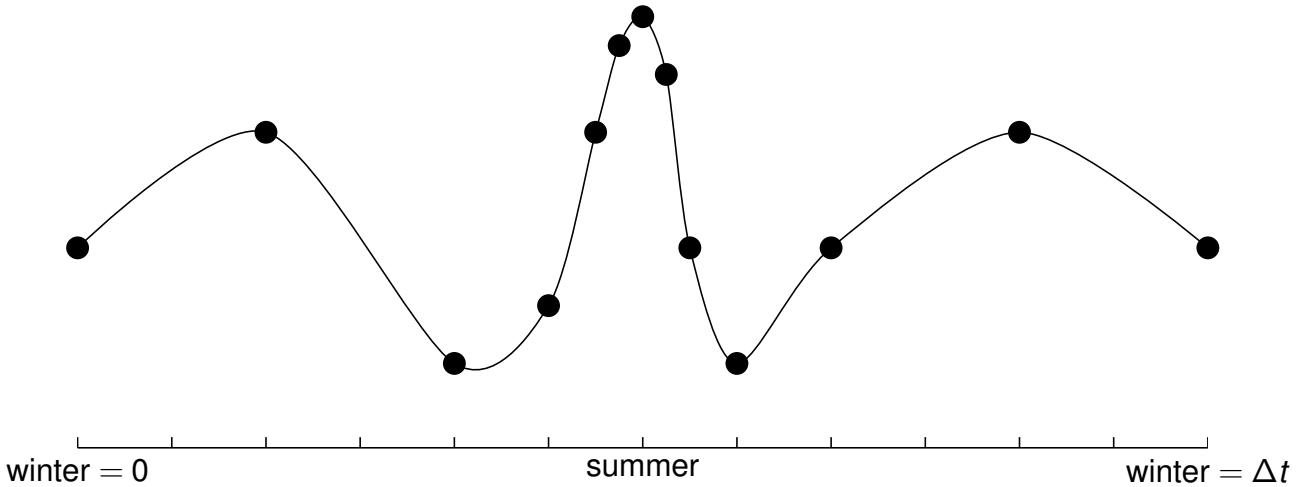
tutorials/perpendicular-flap

DEMO TIME

preCICE v3: Time interpolation

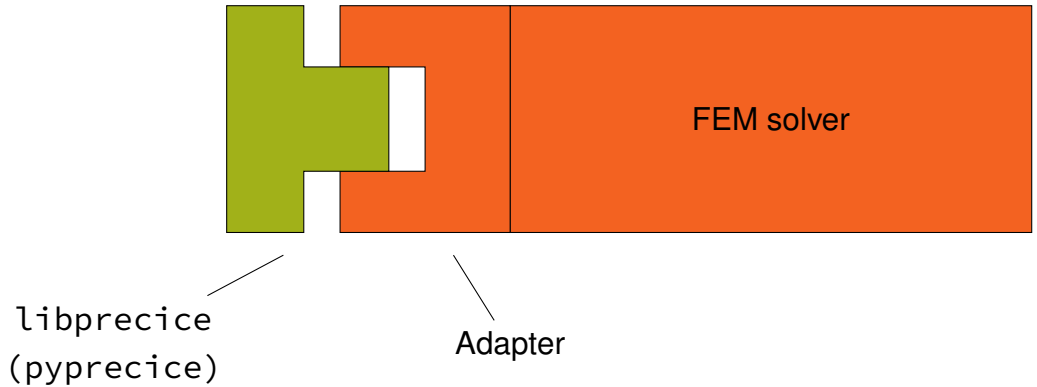
preCICE Workshop 2023: Two "ice sheet talks"[1]

- Yannic Fischler: *A preCICE-interface for the ice-sheet and sea-level system model*
- Daniel Abele: *Coupling an ice sheet model with satellite image based simulation of calving fronts*



[1] https://precice.org/precice-workshop-2023.html

Requirements for time interpolation in preCICE



Goal: Black-box + higher-order + multirate

- Numerics: Accuracy, convergence order, energy conservation
- API Design: Simple user interface!
- Move as much multirate/higher-order/interpolation logic *inside* preCICE as possible
- preCICE v3 + new API was just released[1]

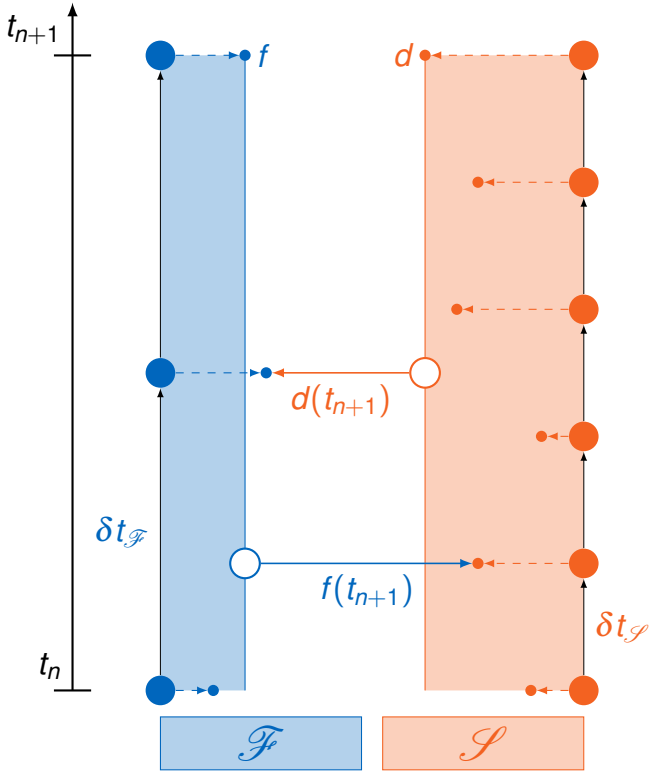
[1] https://github.com/precice/precice/releases/tag/v3.0.0

API for implicit Euler

```python
# example: FSI coupling, perspective of fluid solver 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 + Δt: d_{n+1}
  forces = forces + dt * dfdt(displacements)  # Implicit Euler
  participant.write_data(forces)
  participant.advance(dt)
  # read checkpoint, if needed

participant.finalize()
```

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>
```

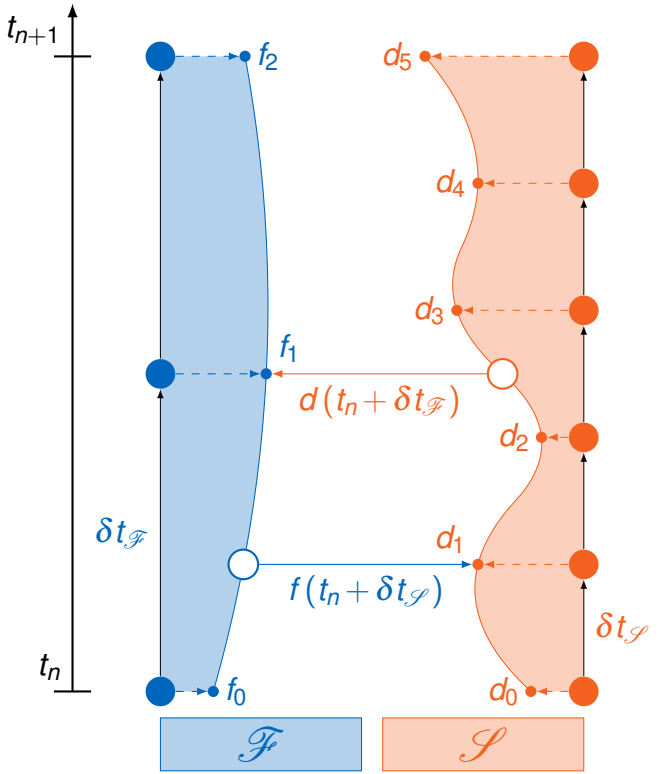


preCICE v3: waveform iteration

$$\mathcal{F}(\mathcal{S}(f^k)) = \tilde{f}^k$$

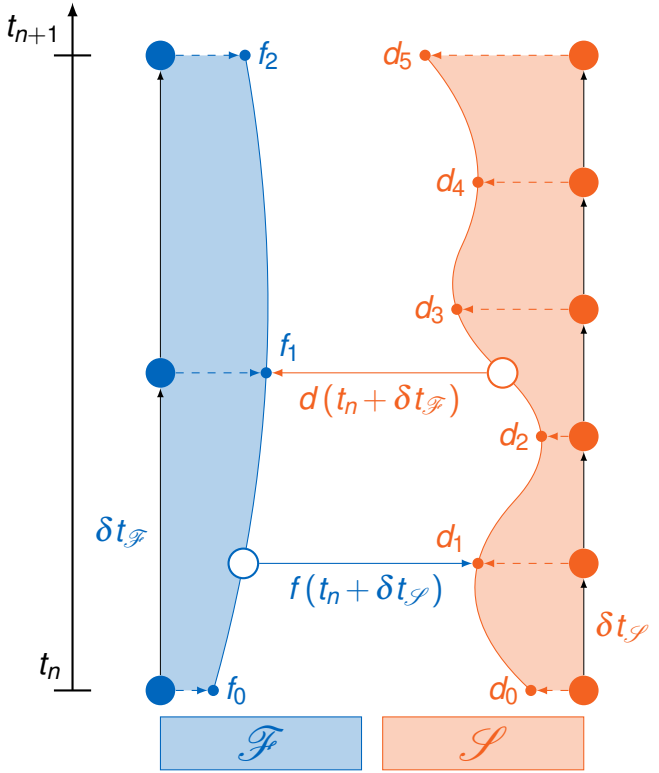
upgrade to preCICE v3

$$\mathcal{F}(\mathcal{S}(f(t)^k)) = \tilde{f}(t)^k$$



preCICE v3: waveform iteration

```
<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>
```



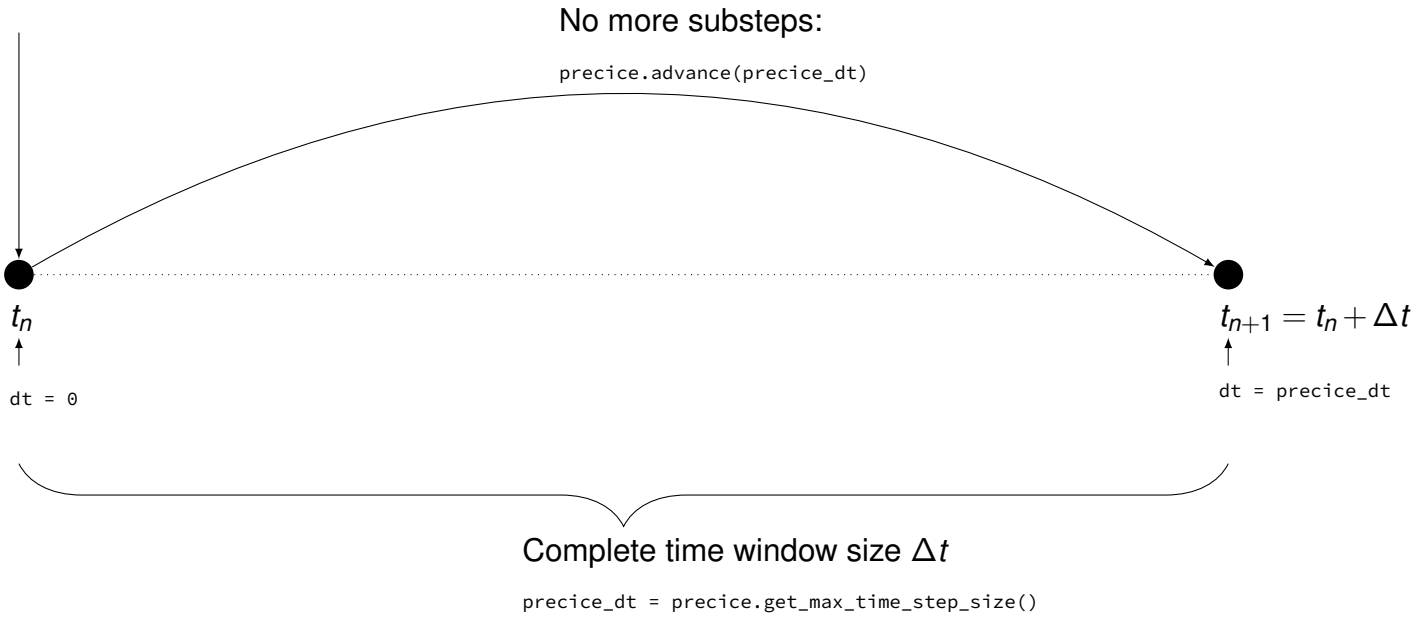
RK4 in code (simplified)

```
# 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_i δt
k1 = A.dot(u)                    + participant.read_data("Displ", 0.0 * dt)
k2 = A.dot(u + k1 * 0.5 * dt) + participant.read_data("Displ", 0.5 * dt)
k3 = A.dot(u + k2 * 0.5 * dt) + participant.read_data("Displ", 0.5 * dt)
k4 = A.dot(u + k3 * 1.0 * dt) + participant.read_data("Displ", 1.0 * dt)
# assemble new solution
u_new = u + dt / 6 * (k1 + 2 * k2 + 2 * k3 + k4)

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

API for multirate

Read interpolated data from current time t_n:

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



API for multirate

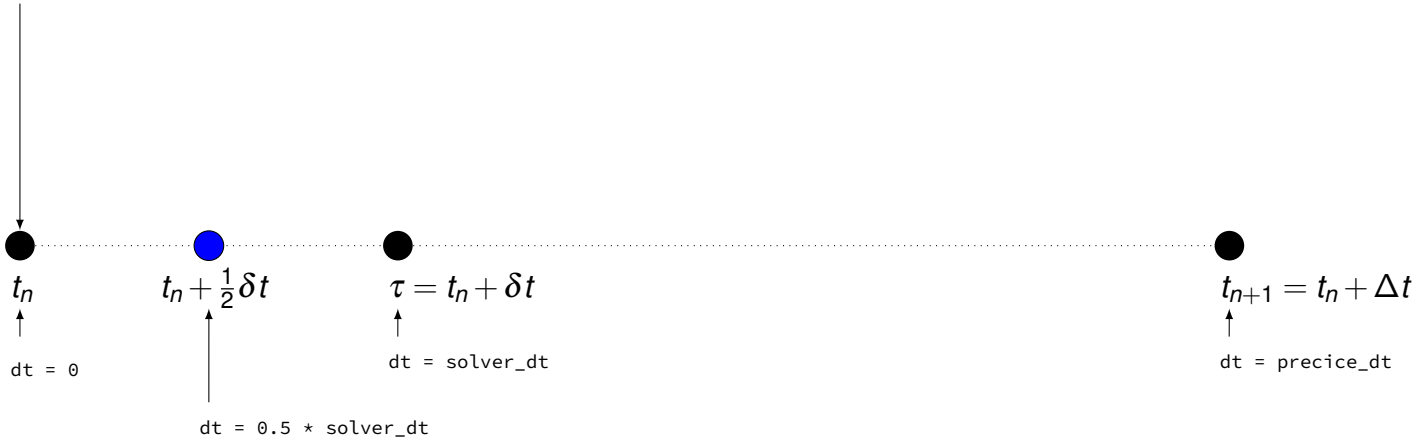
Read interpolated data from current time t_n:

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

API for multirate

Read interpolated data from current time t_n:

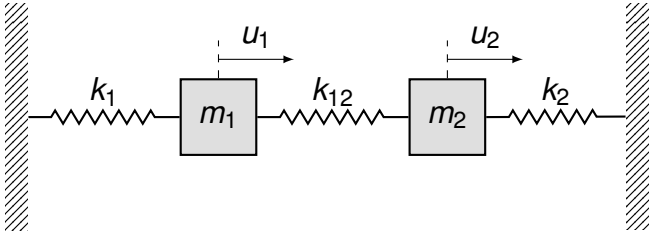
```
displacement = precice.read_data(dt)
```



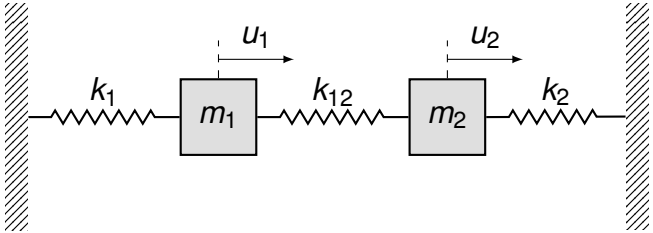
API with RK4 and multirate

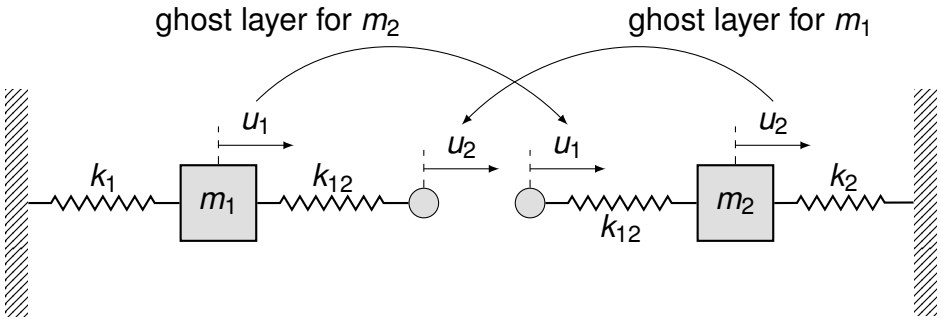
```python
# example: FSI coupling, perspective of fluid solver 𝓕:
# ...
participant.initialize()

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 δt
  # time_stepper represents s-stage RK scheme
  ts = time_stepper.rhs_eval_points(dt)  # t_n + c_1, t_n + c_2 ..., t_n + c_s
  displacements = [participant.read_data(t) for t in ts]  # d(t_n + c_1), d(t_n + c_2) ..., d(t_n + c_s)
  forces = time_stepper.do_step(displacements, dt)  # f_{n+1} = f_n + δt Σ_{i=1}^{s} b_i k_i
  participant.write_data(forces)
  participant.advance(dt)
  # read checkpoint, if needed

participant.finalize()
```

tutorials/oscillator

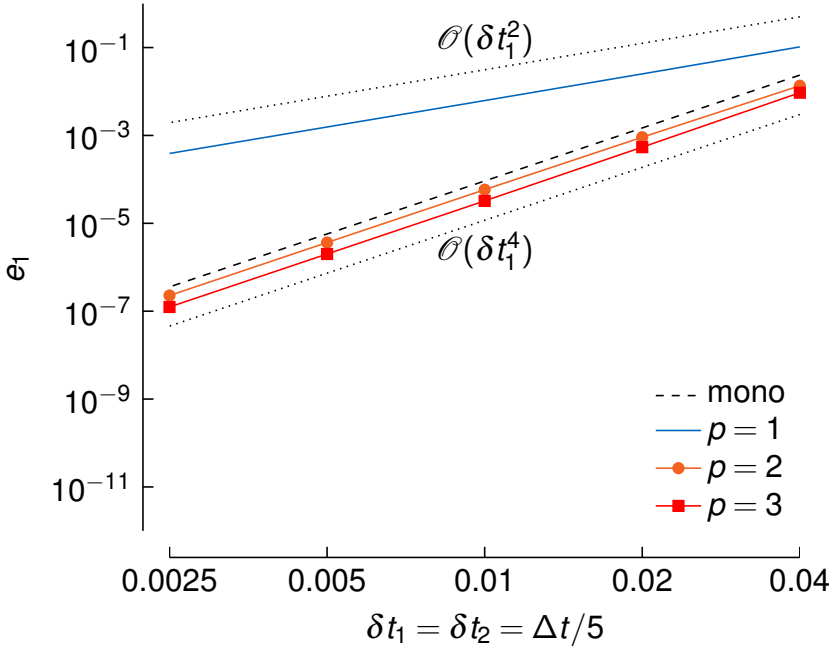


tutorials/oscillator

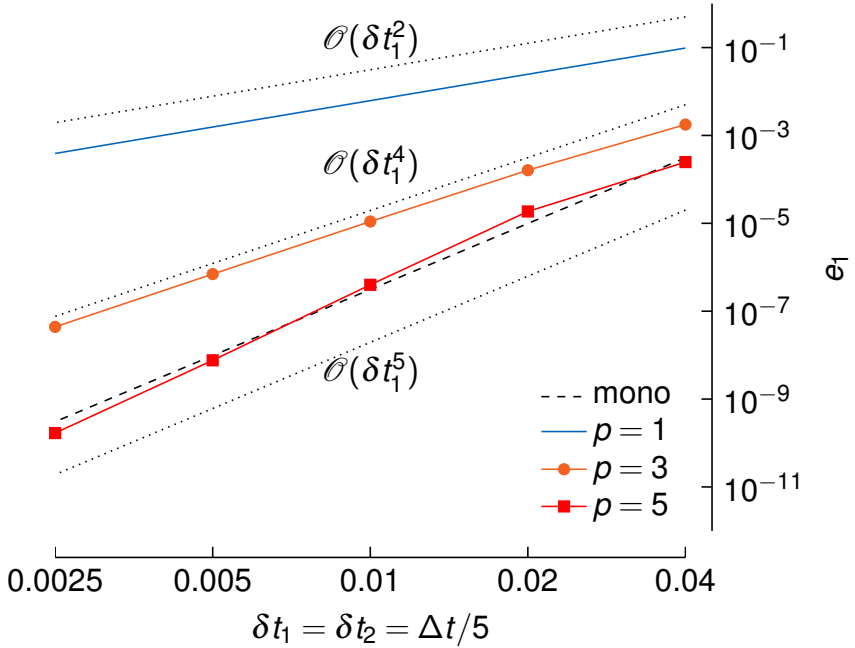




tutorials/oscillator

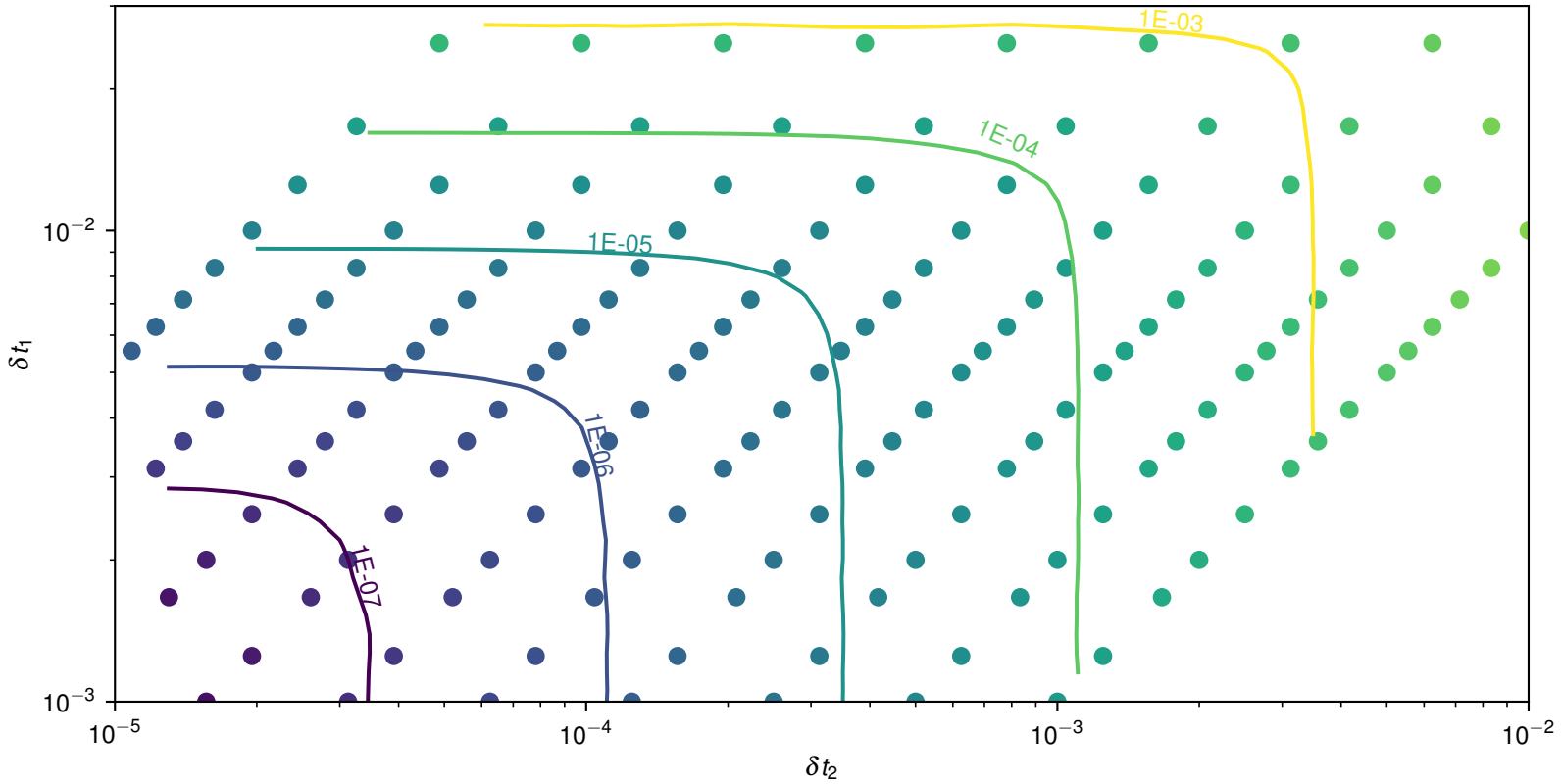


classic Runge-Kutta method

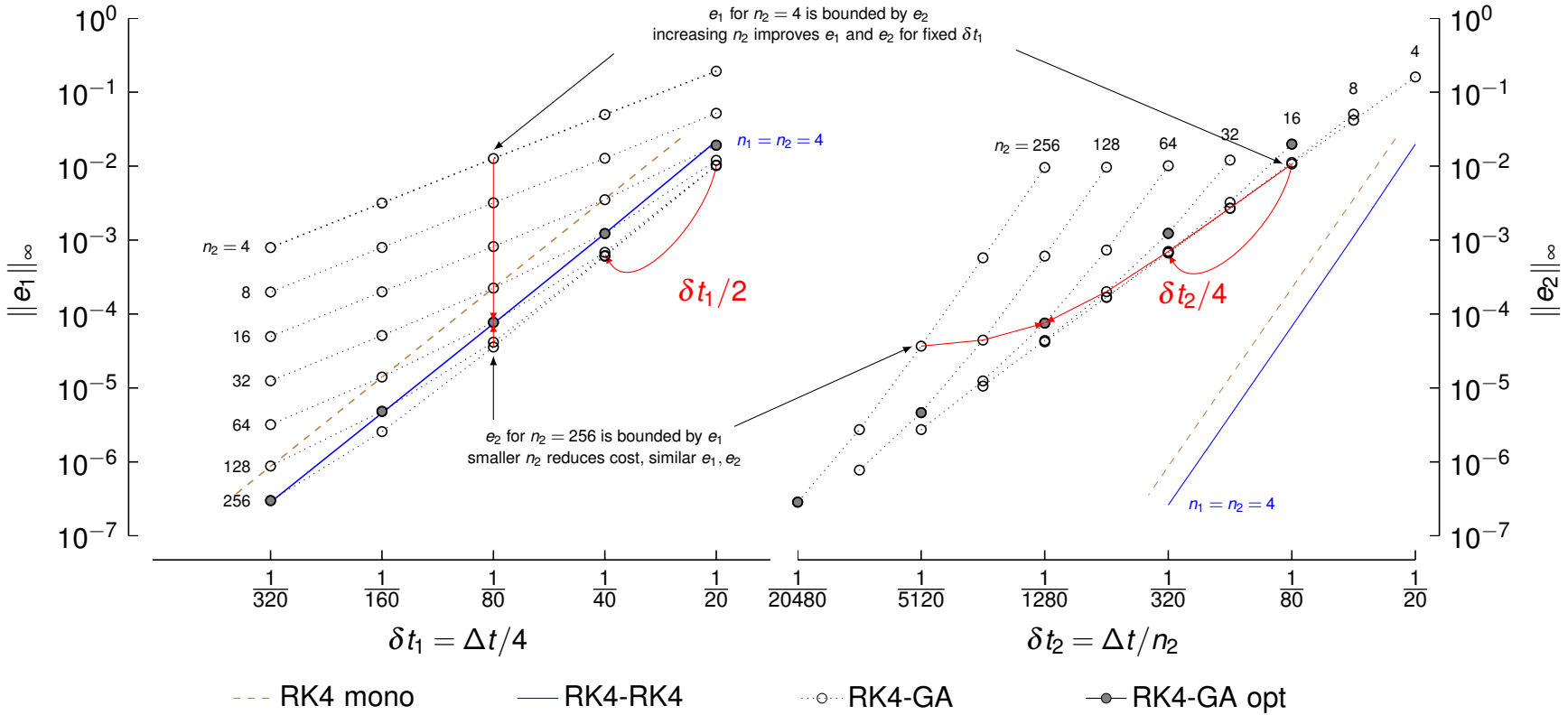


Radau IIA

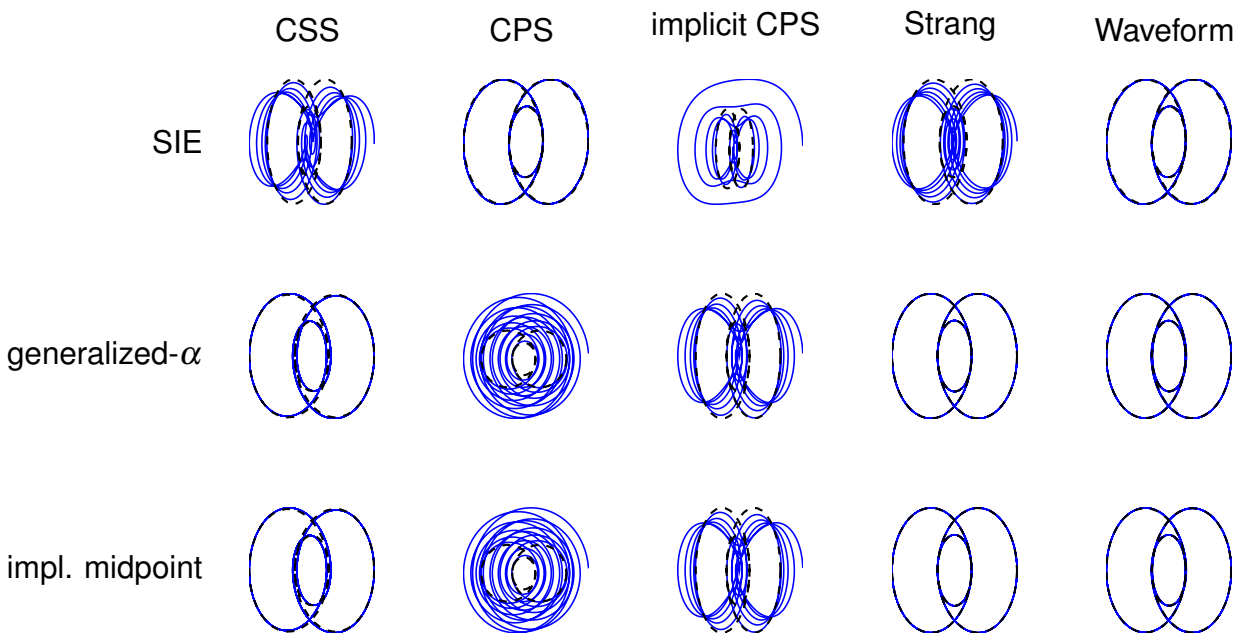
tutorials/oscillator



tutorials/oscillator

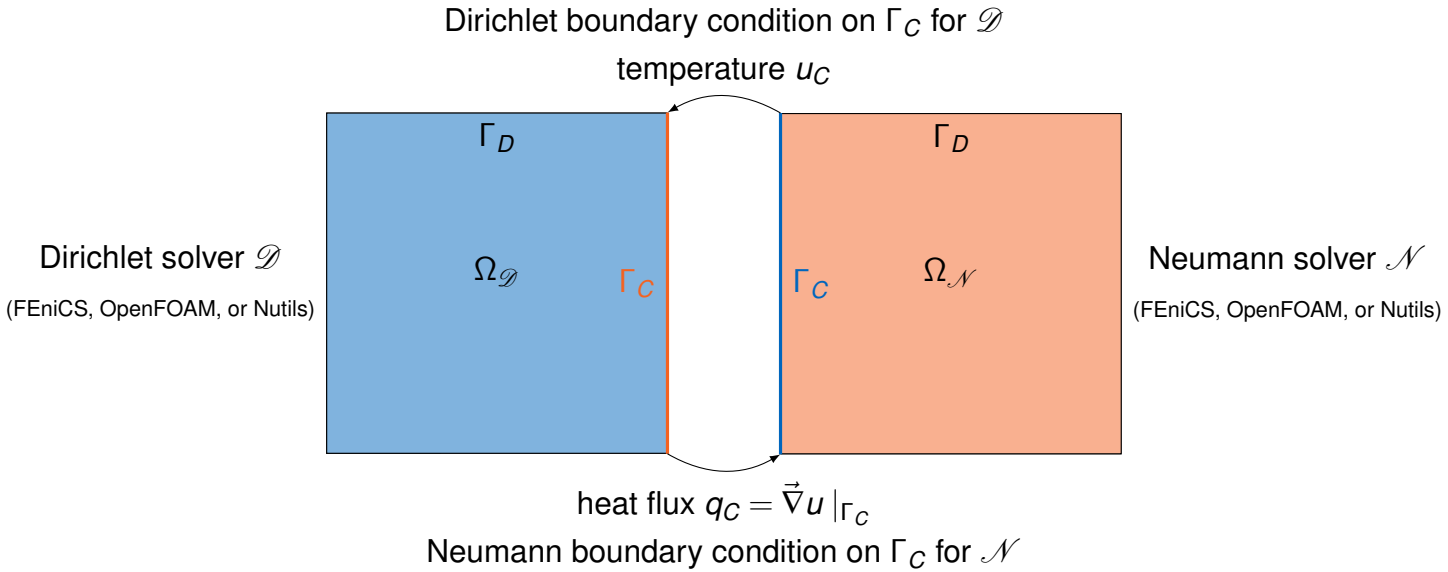


tutorials/oscillator

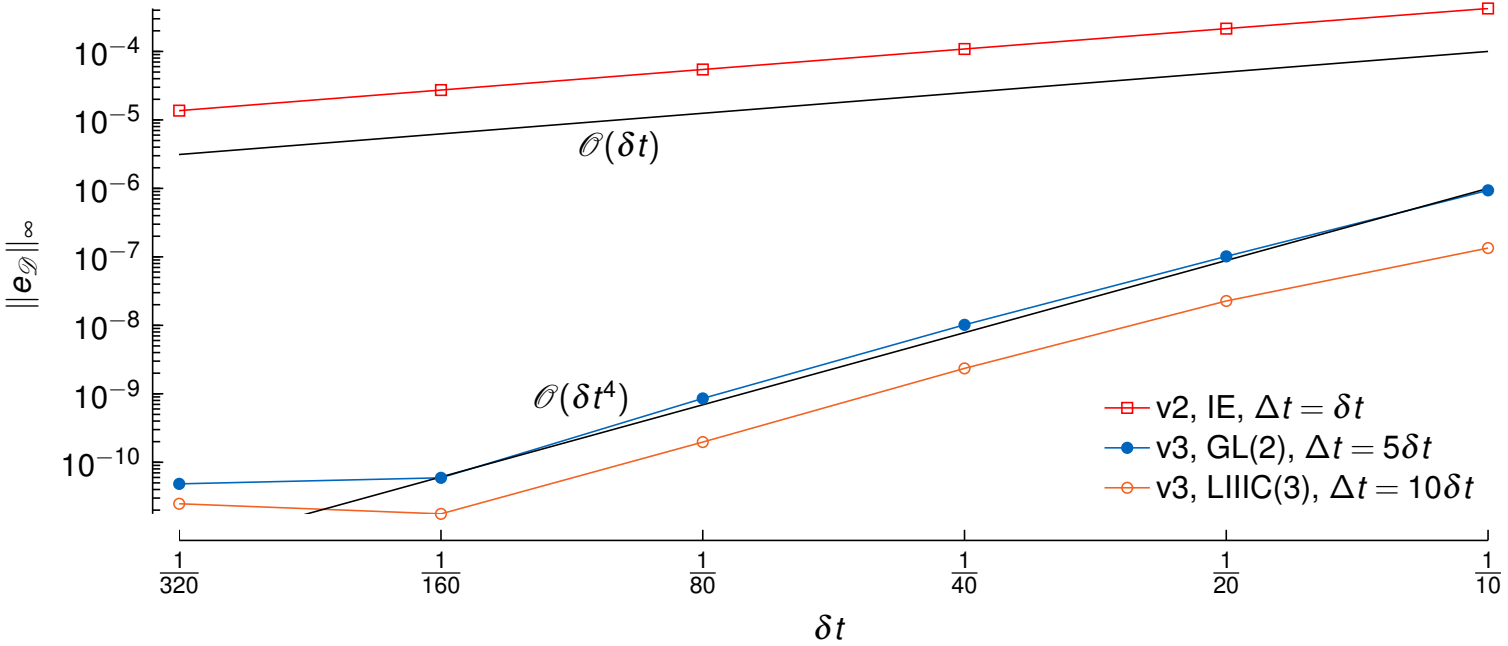


A Simple Test Case for Convergence Order in Time and Energy Conservation of Black-Box Coupling Schemes. 2022.

tutorials/partitioned-heat-conduction



tutorials/partitioned-heat-conduction



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

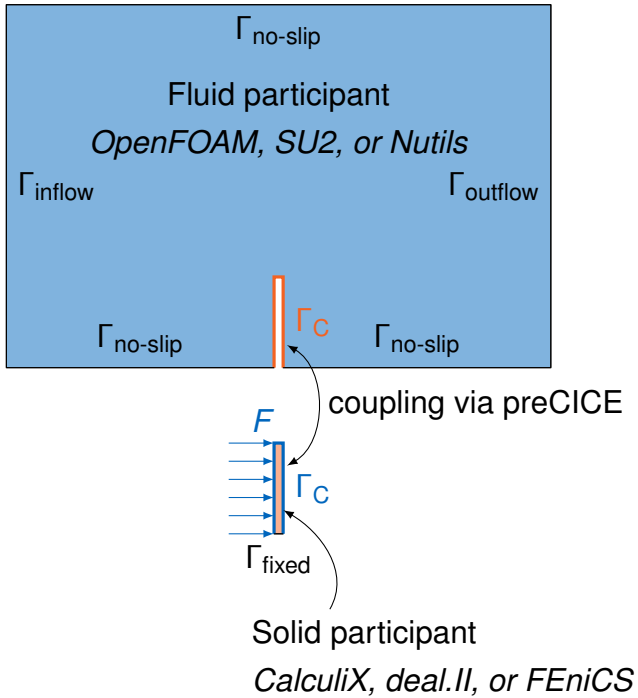
tutorials/partitioned-heat-conduction

Tooling: Higher-order time stepping

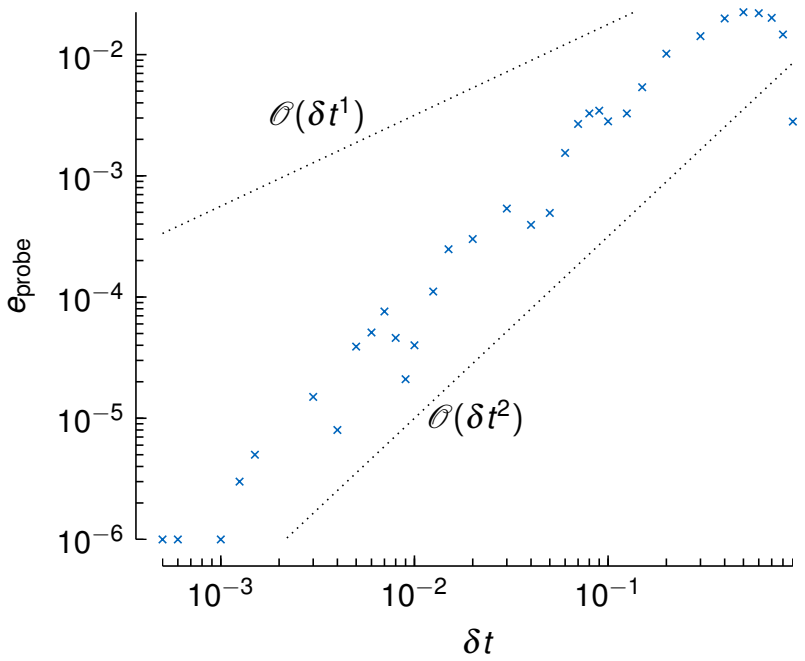
- FEniCS + pySDC
- `github.com/firedrakeproject/Irksome`[1]
  ```
  inner(Dt(u), v)*dx + inner(grad(u), grad(v))*dx - inner(rhs, v)*dx
  ```
- Applied general approach to FEniCS:
  - B.Sc. thesis Nikola Wullenweber monolithic higher-order
  - B.Sc. thesis Niklas Vinnitchenko partitioned higher-order
  - Need derivatives of B-splines!

[1] Farrell, Patrick E., Robert C. Kirby, and Jorge Marchena-Menendez. *Irksome: Automating Runge–Kutta time-stepping for finite element methods*

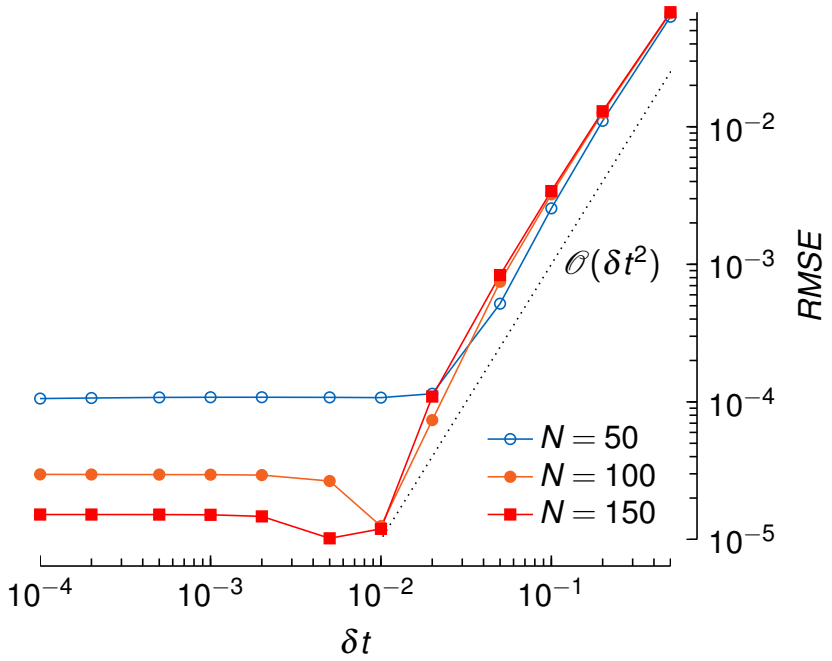
`tutorials/perpendicular-flap` (without coupling)



`tutorials/perpendicular-flap` (without coupling)



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*

Conclusion

State of development

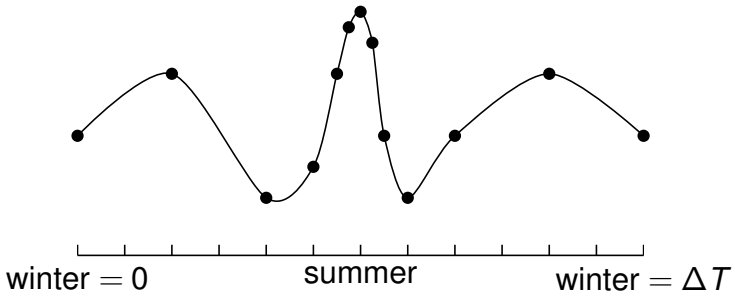
- Multirate time-stepping + black-box + usable API
- Time interpolation new in preCICE v3
- First tutorials use time interpolation

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

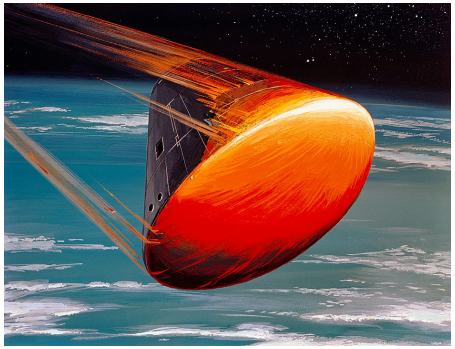
Stay in touch?

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

Conferences

- WCCM + preCICE course
  July 2024@Vancouver
- preCICE Workshop
  Sept 2024@Stuttgart
- COUPLED
  2025@Sicily

Rocket science

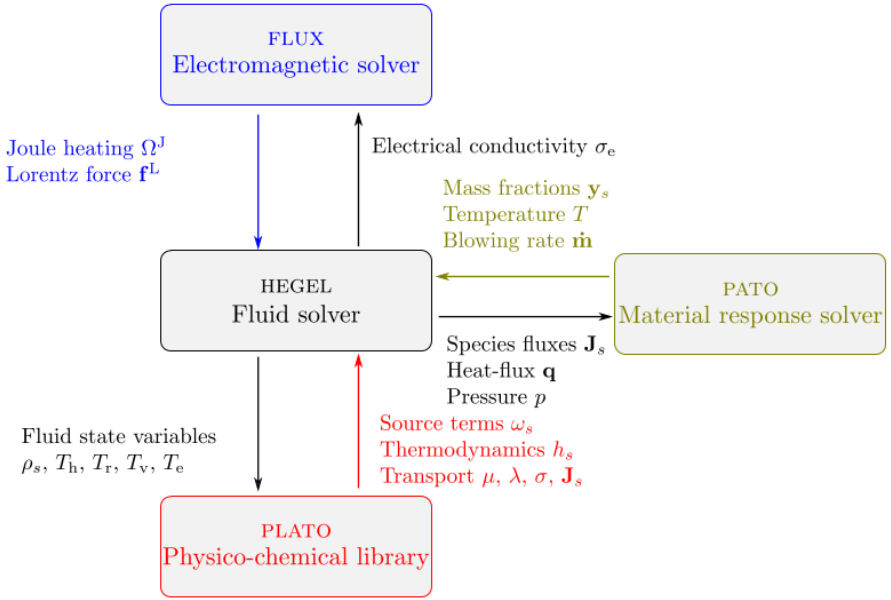


By North American Rockwell, Public Domain,
https://commons.wikimedia.org/w/index.php?curid=2466251

Partitioned solver for ICP[1] wind tunnels

- $\Delta t_{\text{FLUX}} \approx 10 \Delta t_{\text{HEGEL}} \approx 1000 \Delta t_{\text{PATO}}$
- HEGEL uses 3rd order RK
- PATO uses $\Delta t_{\text{PATO}} = 10^{-4} s = \Delta T_{\text{preCICE}} / 100$
- HEGEL uses $\Delta t_{\text{HEGEL}} = 10^{-2} s = \Delta T_{\text{preCICE}}$
- FLUX uses $\Delta t_{\text{FLUX}} = 10^{-1} s = 10 \Delta T_{\text{preCICE}}$ (???).

[1] Inductively Coupled Plasma

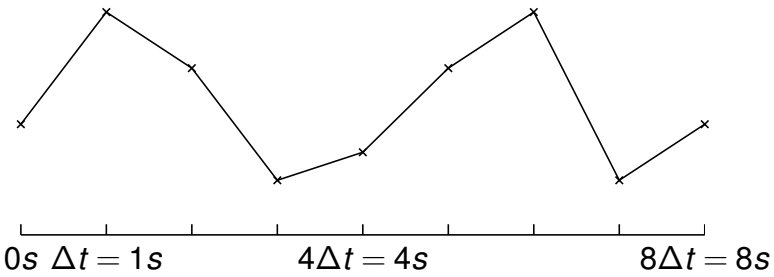


from *Alessandro Munafò, et al. A Multi-Physics Modeling Framework for Inductively Coupled Plasma Wind Tunnels. 2022.* https://doi.org/10.2514/6.2022-1011

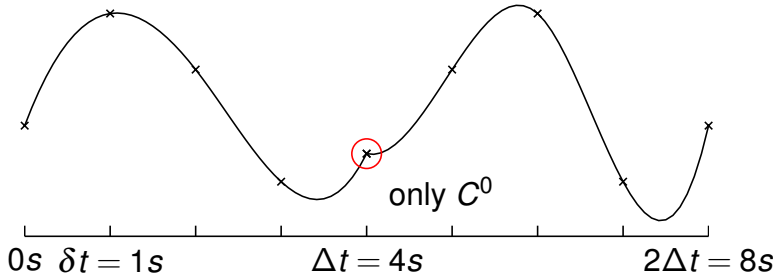
Subcycling

- 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

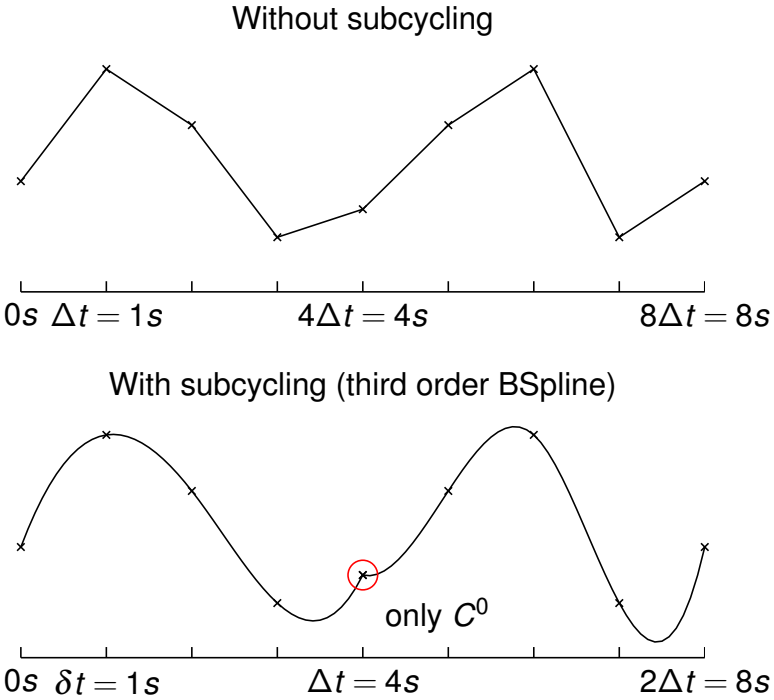


With subcycling (third order BSpline)



[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



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

$\Delta t \ggg \delta t$