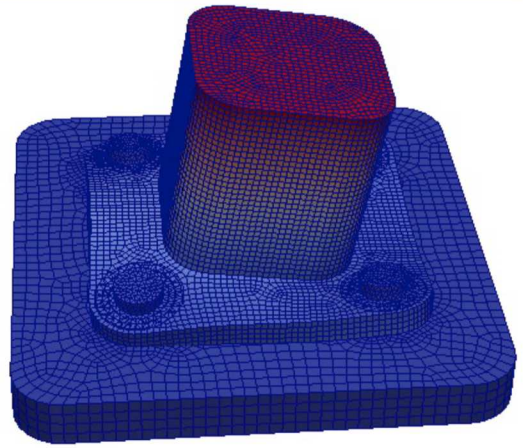
SAND2018-12240C



Tempus Time-Integration Package & Applications

Curtis Ober*, Roger Pawlowski, Sidafa Conde, Irina Tezaur, Eric Phipps, Edward Phillips, Alejandro Mota, Greg Philpot, Denis Ridzal, Mike Hansen, Travis Fisher

Sandia National Laboratories

Exceptional service in the national interest

Trilinos User-Developer Group Meeting
2:20-2:50pm October 24, 2018

U.S. DEPARTMENT OF ENERGY

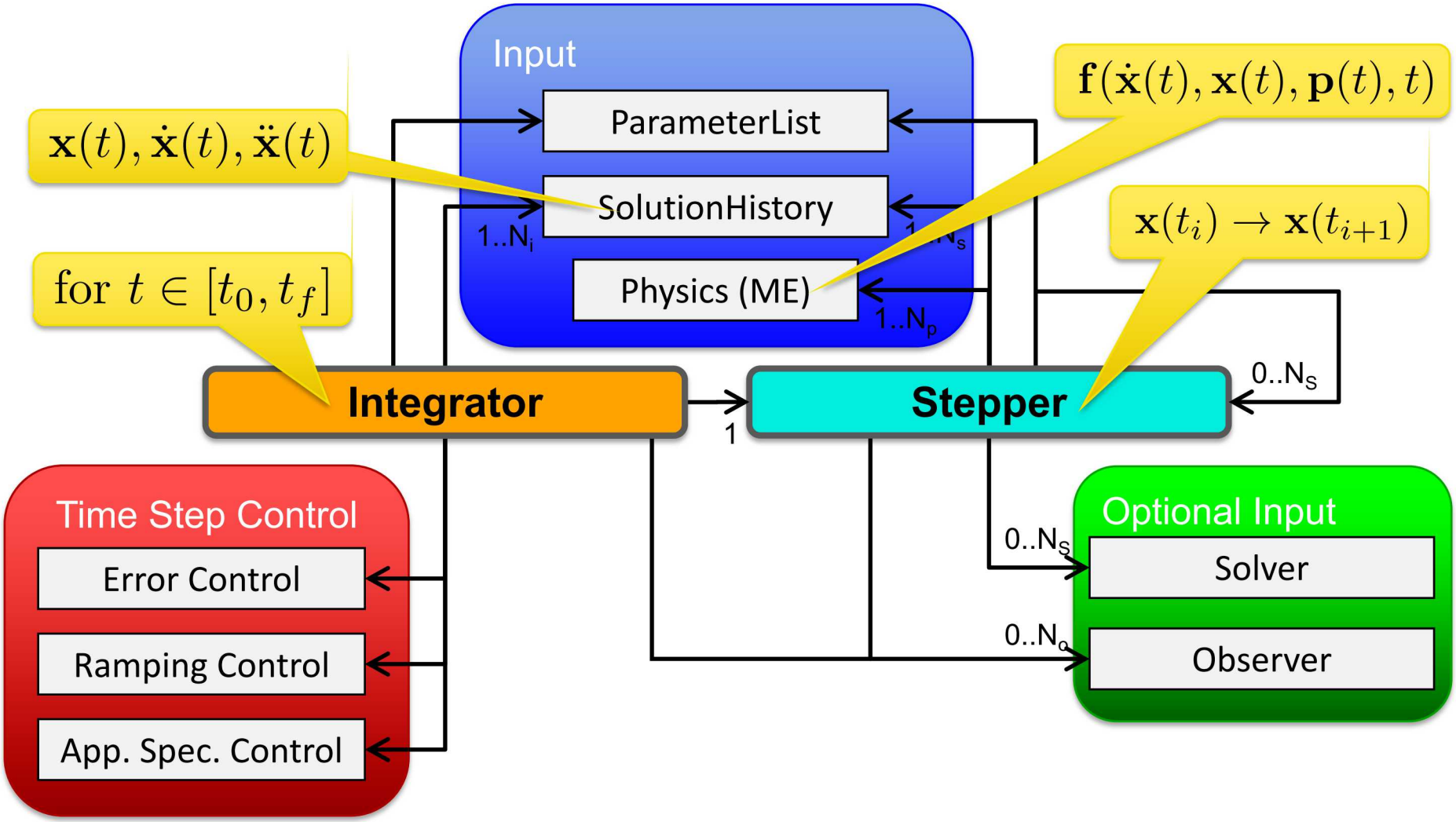
NNSA National Nuclear Security Administration

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.
SAND2018-XXXX

What is Tempus?

- New time-integration package to support advanced analysis techniques, including various time integrators and embedded sensitivity analysis for next-generation code architectures.
- Provide “out-of-the-box” and “build-your-own” capabilities
- Became Trilinos package February 2017
- Tempus incorporated into several applications
  - EMPIRE, SPARC, Albany (LCM, LandIce), Drekar
- Coupled with other Trilinos packages
  - NOX and ROL
  - Thyra and Teuchos

Tempus Interface



Tempus Steppers

- Forward Euler
- Backward Euler
- Explicit Runge-Kutta (ERK)
  - 12 + general Butcher Tableaus
- Diagonally Implicit Runge-Kutta (DIRK)
  - 13 + general Butcher Tableaus
- Newmark-β (3 forms)
- HHT-α
- IMEX-RK
  - 3 + general Butcher Tableaus
- Partitioned IMEX-RK
  - 3 + general Butcher Tableaus
- Leapfrog

New in FY2018

- BDF2
- Trapezoidal
- Bogacki-Shampine 3(2) Pair
- Merson 4(5) Pair
- SDIRK 2(1) Pair
- IRK 1 Stage Theta Method
  - $\boldsymbol{\theta}$ = 0.5 Implicit Midpoint
- EDIRK 2 Stage Theta Method
  - $\boldsymbol{\theta}$ = 0.5 Trapezoidal
- 1st Operator-Split
- Generalized-α Method
- …

Embedded Error Analysis

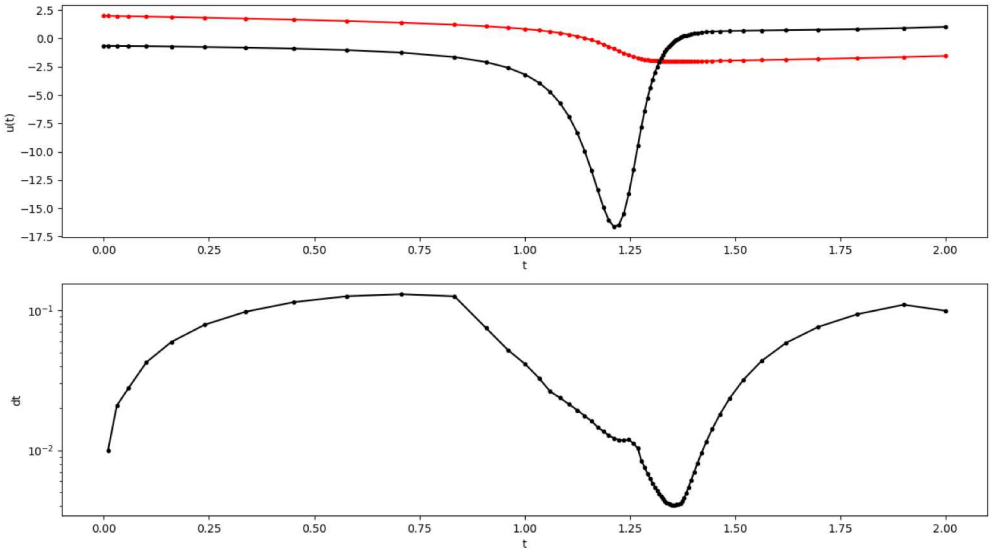
CENTER FOR COMPUTING RESEARCH
Sandia National Laboratories

Dev: Conde

- Butcher tableau of an ERK

$$\begin{array}{c|ccccc} 0 & 0 & & & & \\ c_2 & a_{21} & 0 & & & \\ c_3 & a_{31} & a_{32} & 0 & & \\ \vdots & \vdots & \vdots & \ddots & 0 & \\ c_s & a_{s1} & a_{s2} & \dots & a_{s,s-1} & 0 \\ \hline & b_1^T & b_2^T & \dots & b_{s-1}^T & b_s^T \\ & \tilde{b}_1^T & \tilde{b}_2^T & \dots & \tilde{b}_{s-1}^T & \tilde{b}_s^T \end{array}$$

Van der Pol Problem
w/ error controlled time-stepping



- Typically orders are

$$\tilde{p} = p - 1$$

- Global error vector

$$\tilde{e}_{n+1} = u_{n+1} - \tilde{u}_{n+1} = \Delta t \sum_{j=1}^{s} \left(b_j - \tilde{b}_j \right) F(y_j)$$

- For a given ε

$$\Delta t_{\text{new}} := \Delta t \left(\frac{\varepsilon}{\tilde{e}_{n+1}} \right)^{\frac{1}{p}}$$

Other controllers

$$(\Delta t)_{n+1} = (\Delta t)_n \left(e_n^{-k_1/p} e_{n-1}^{k_2/p} e_{n-2}^{-k_3/p} \right) \quad \text{PID}$$

$$(\Delta t)_{n+1} = (\Delta t)_n \left(e_n^{-k_1/p} e_{n-1}^{k_2/p} \right) \quad \text{PI}$$

$$(\Delta t)_{n+1} = (\Delta t)_n \left(e_n^{-k_1/p} \right) \quad \text{I}$$

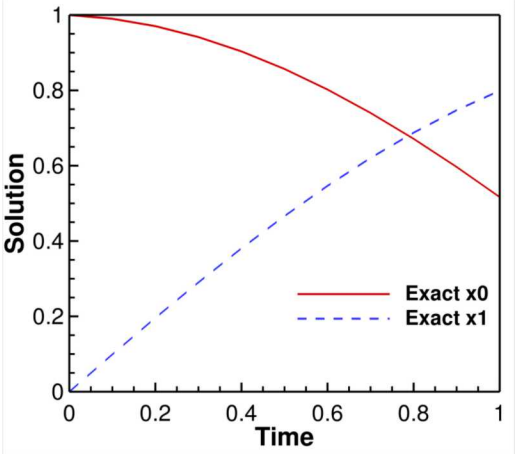
Tempus Sensitivity Analysis

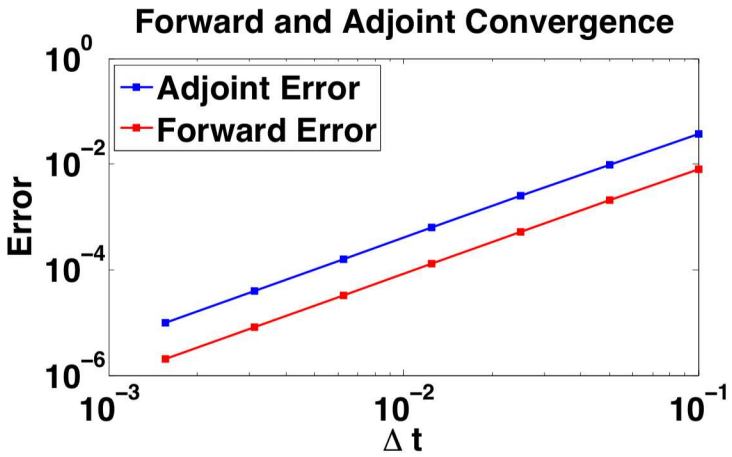
CENTER FOR COMPUTING RESEARCH
Sandia National Laboratories

Dev: Phipps

- Development of embedded optimization, UQ capabilities for ATDM and ECP applications
  - Provide analytic, transient adjoint sensitivity capabilities through Tempus
  - Enables efficient large-scale optimization and UQ
  - Adjoint sensitivities enable efficient derivative computation for very large-scale problems
- FY18: transient forward/adjoint sensitivity capabilities
  - Available for most steppers
  - BDF2 Demonstration
    - SinCos problem
    - Both achieve second-order accuracy!
  - Extensive regression coverage

SinCos Test





Tempus Interfaces for Optimization through ROL (Rapid Optimization Library)

CENTER FOR COMPUTING RESEARCH

Sandia National Laboratories

Dev: Ridzal, Phipps

- Solving optimization problems of the form:

$$\begin{aligned} &\text{minimize} && J(\dot{\mathbf{x}}(t), \mathbf{x}(t), \mathbf{p}(t), t) \\ &\text{subject to} && \mathbf{f}(\dot{\mathbf{x}}(t), \mathbf{x}(t), \mathbf{p}(t), t) = 0 \end{aligned}$$

Objective function

Constraint

- Two optimization interfaces: **reduced-space** and **full-space**.
- **Reduced-space interface: ROL::TempusReducedObjective**.
- Given a Thyra::ModelEvaluator for the physics equations, a parameter list for Tempus and an objective function, this interface builds the reduced objective function

$$\widetilde{J}(\dot{\mathbf{x}}(t), \mathbf{x}(t), \mathbf{p}(t), t) \text{ where } (\dot{\mathbf{x}}(t), \mathbf{x}(t)) \text{ solves } \mathbf{f}(\dot{\mathbf{x}}(t), \mathbf{x}(t), \mathbf{p}(t), t) = 0$$

  and computes its values and gradients.
- *Note: Time integration is handled fully by Tempus.*
- Additional work needed: better handling of transient response/objective functions, second derivatives, examples.

Tempus Interfaces for Optimization through ROL (Rapid Optimization Library)

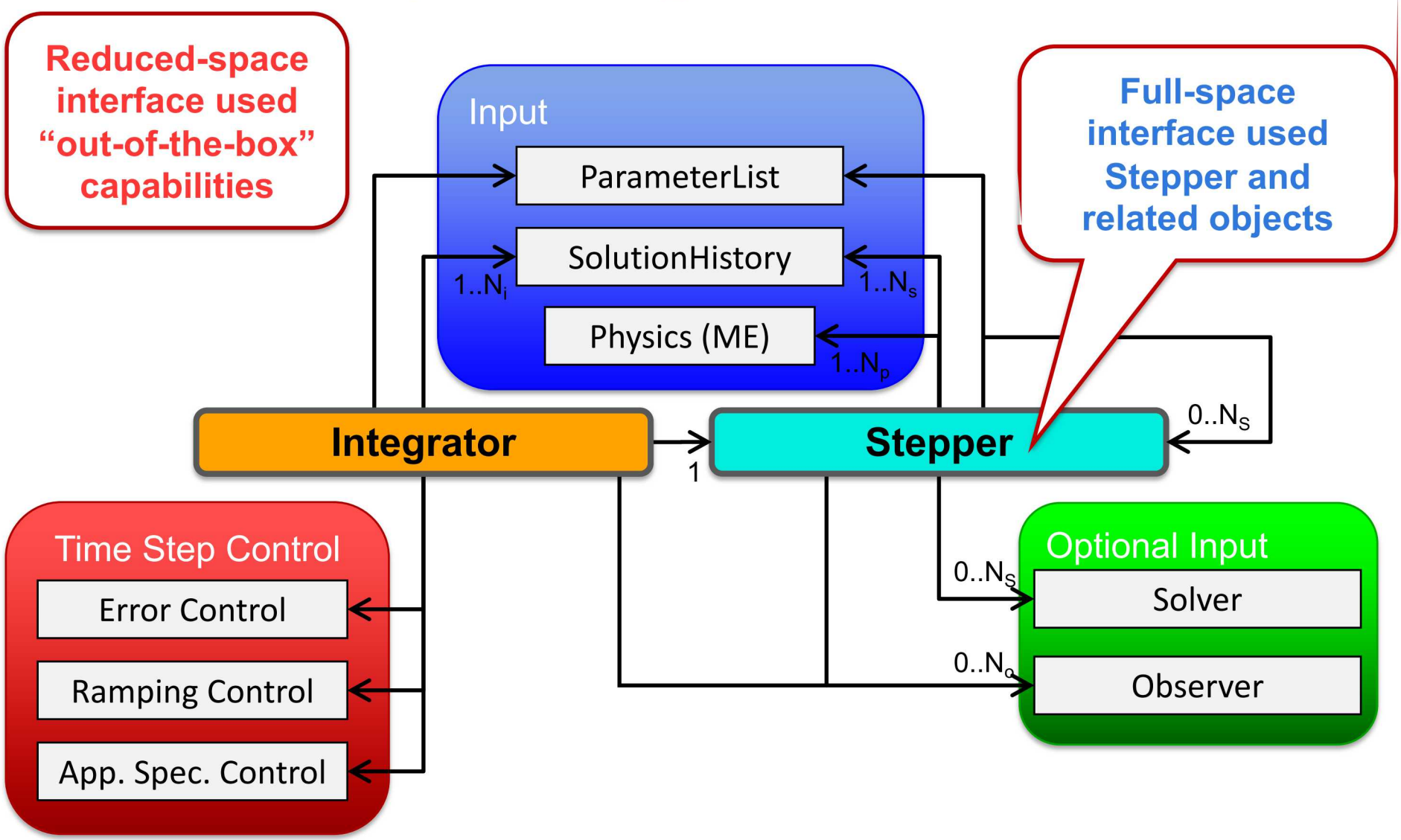
- **Full-space interface: ROL::TempusDynamicConstraint**.
- Given a Thyra::ModelEvaluator for the physics equations and a Tempus::Stepper, the ROL::DynamicConstraint object is built and used in conjunction with ROL::DynamicObjective.

ROL::DynamicConstraint

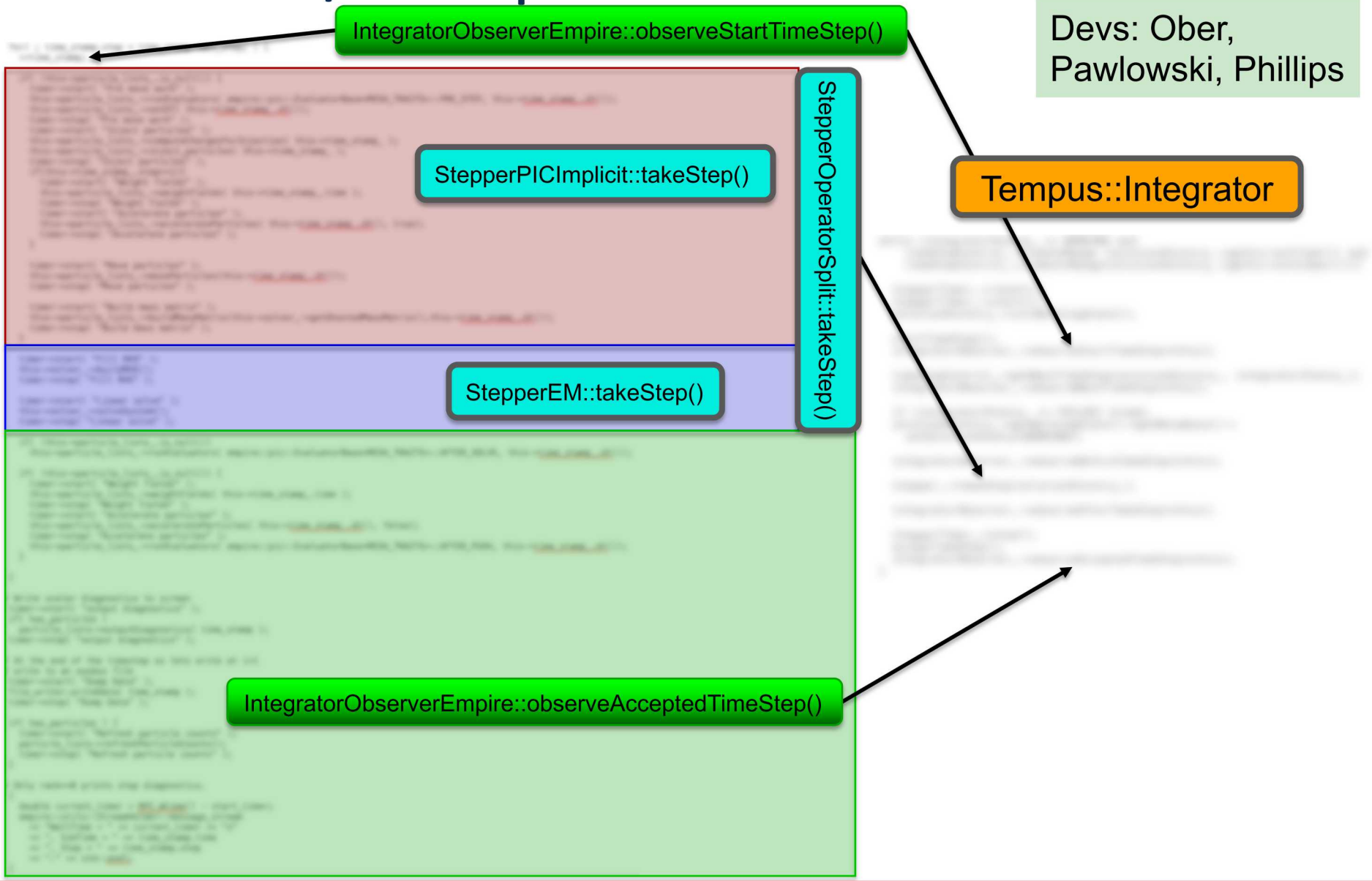
$$\mathbf{f}(\dot{\mathbf{x}}(t), \mathbf{x}(t), \mathbf{p}(t), t) = 0 \Rightarrow \mathbf{f}^i(\mathbf{x}_{i+1}, \mathbf{x}_i, \mathbf{p}_i, t_i) = 0,\ i = 1, \ldots, N$$

- ROL::Dynamic objects are defined on individual time steps. *The time integration "formula" is provided by Tempus, but ROL controls the time stepping loop.*
- **Enables:** parallel-in-time optimization, optimization-driven storage compression, fine-grained derivative checks, etc.
- Additional work needed: support for multi-step schemes.
- Check out **rol/example/tempus**.

ROL's Tempus Usage



EMPIRE / Tempus

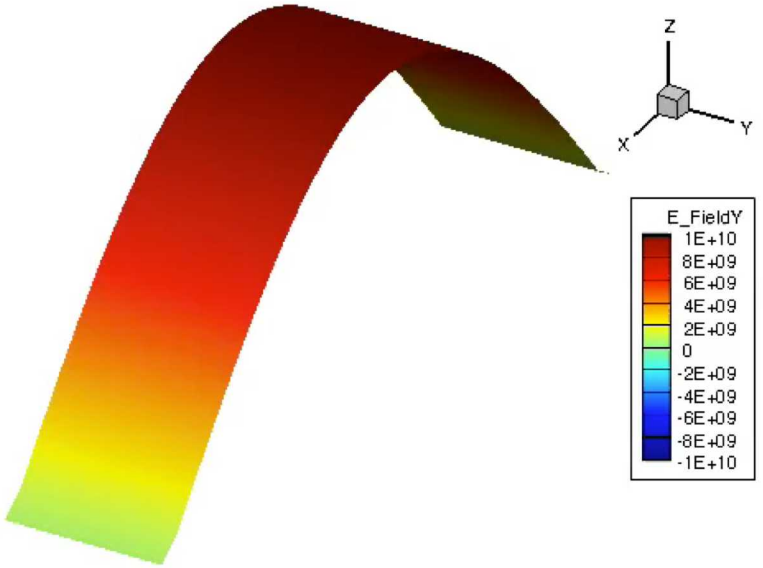


Devs: Ober, Pawlowski, Phillips

EMPIRE/Tempus Verification

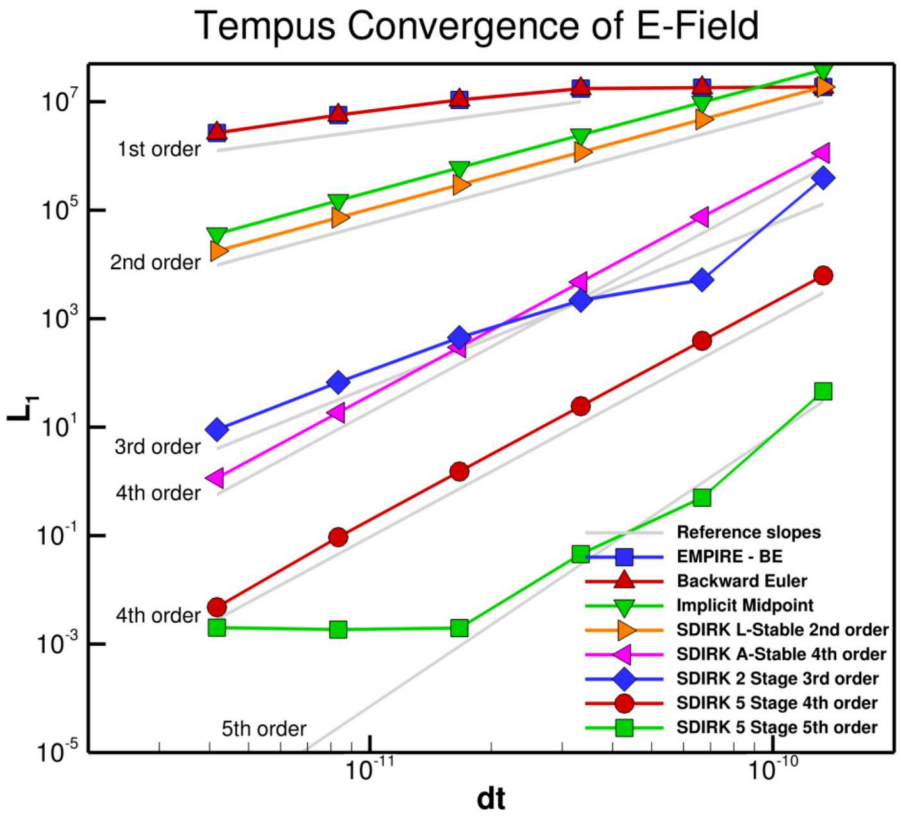
- Regression test – OscillatingEField1D
  - Exercises basic electromagnetic behavior without particles on a (quasi) 1D domain and simulates a single cycle of a half wavelength of a sinusoidal wave.
  - Has an exact solution

$$E_0(x,y,t) = 0.1 \times 10^{10}$$
$$E_1(x,y,t) = 1.0 \times 10^{10} \sin(\pi x) cos(c\pi t)$$
$$B(x,y,t) = -1.0 \times 10^{10} \cos(\pi x) sin(c\pi t)/c$$

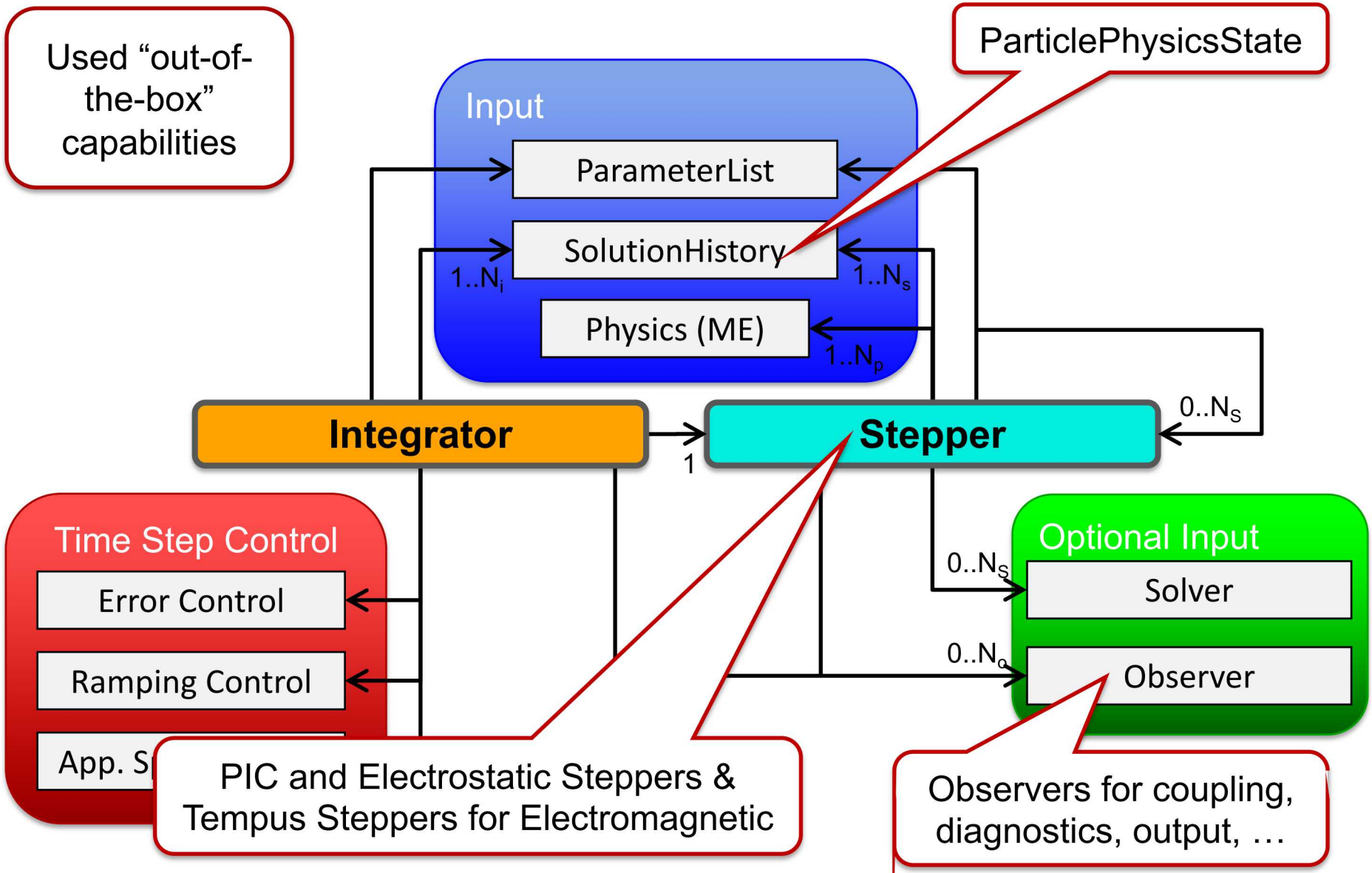


EMPIRE/Tempus Verification

- Tempus' and EMPIRE's Backward Euler match to ~4 digits!
  - Including non-asymptotic region at large dt
- Tempus' 2nd and 4th order Steppers provide improvements.



EMPIRE's Tempus Usage



SPARC/Tempus

CENTER FOR COMPUTING RESEARCH
Sandia National Laboratories

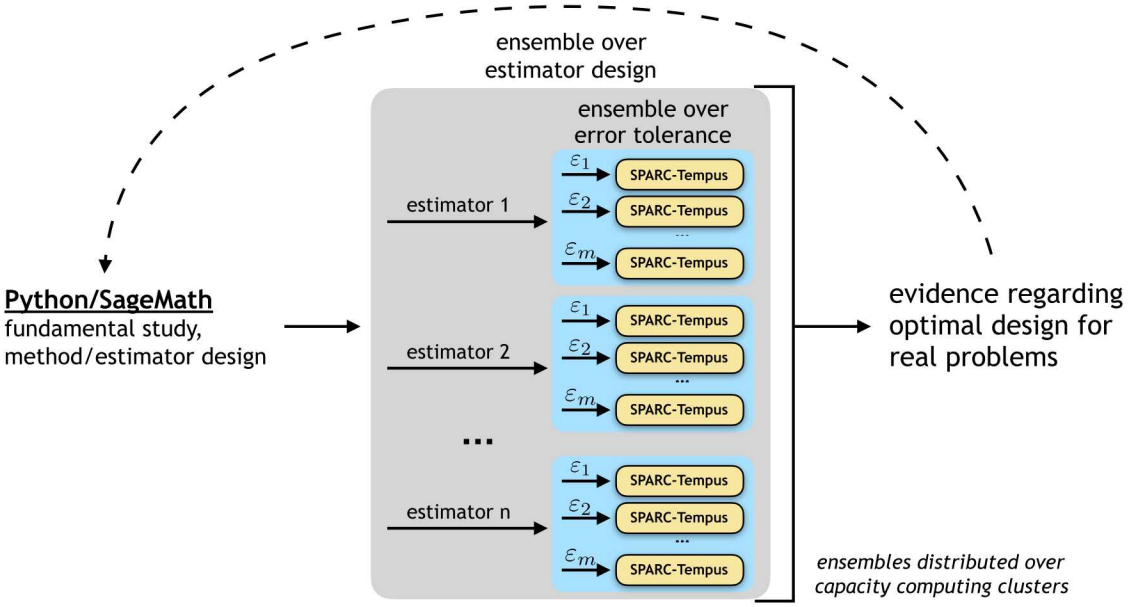
Dev: M. Hansen, Fisher

Using Tempus to study and design better embedded error estimators for direct numerical simulation with explicit Runge-Kutta methods

High-order spatial discretizations and complex physics models do not admit simple bounds on the time step for stability/accuracy.

- Discretization and model error often dominate, so we want **stable** time-stepping with as **large a time step as possible**.

Studying how **linear stability properties of the RK method and its embedded estimator** impact the ability of error-controlled time-stepping to **run near the stability limit** (max time step).

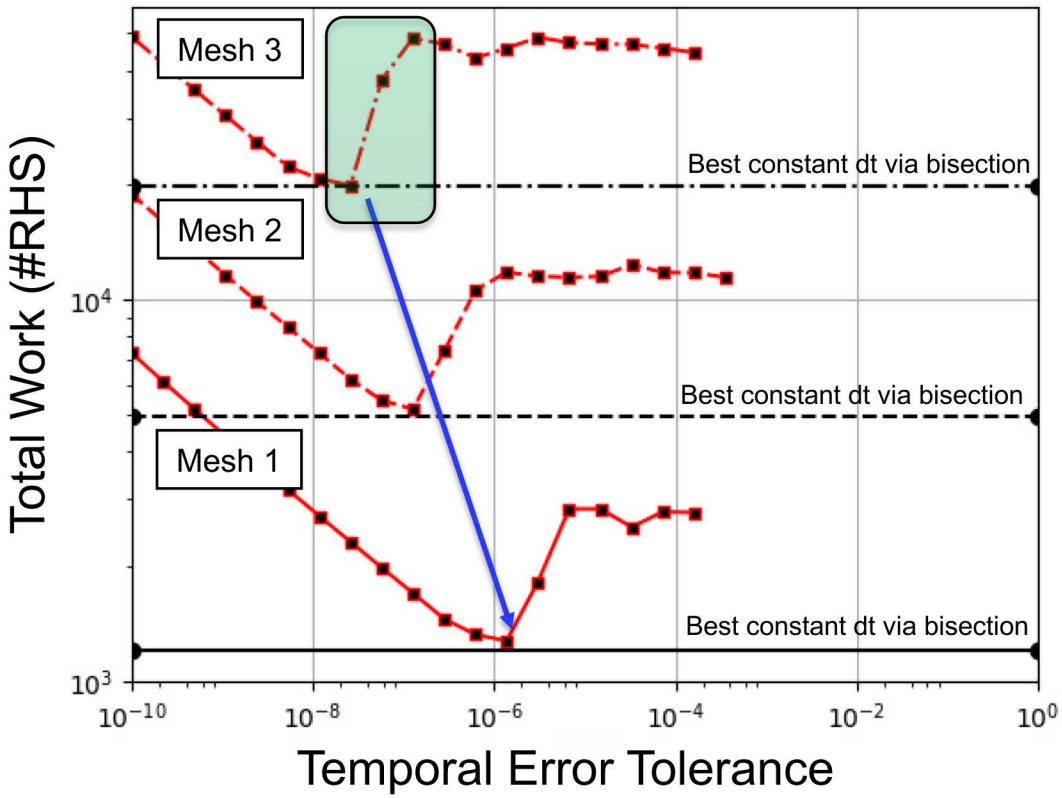


SPARC/Tempus

The existing error estimator is more stable than the solver, thus failing to signal instability and demanding **expensive** step rejection for stability at high temporal error tolerances

Adaptive time-stepping is capable of running at the stability boundary, but only for a tiny window of the error tolerance.

Furthermore this window depends on the mesh resolution, and specific problem. No generality!



- **Method:** 6-stage, 4th-order ERK subset of the IMEX of Kanevsky et al. (2007)
- **Canonical "viscous shock" problem:** 1-D viscous, nonlinear Navier-Stokes equations

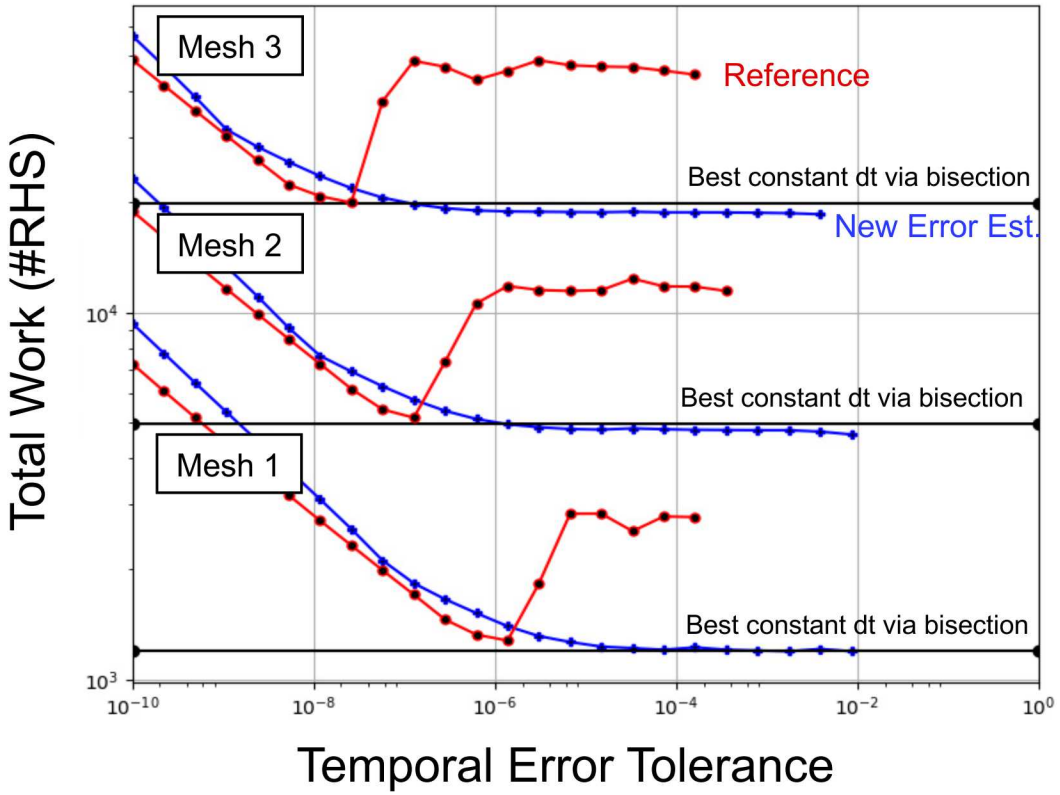
SPARC/Tempus

A newly-designed embedded error estimator shows little sensitivity to the tolerance or mesh resolution, running efficiently at the stability boundary to minimize computational cost of a simulation

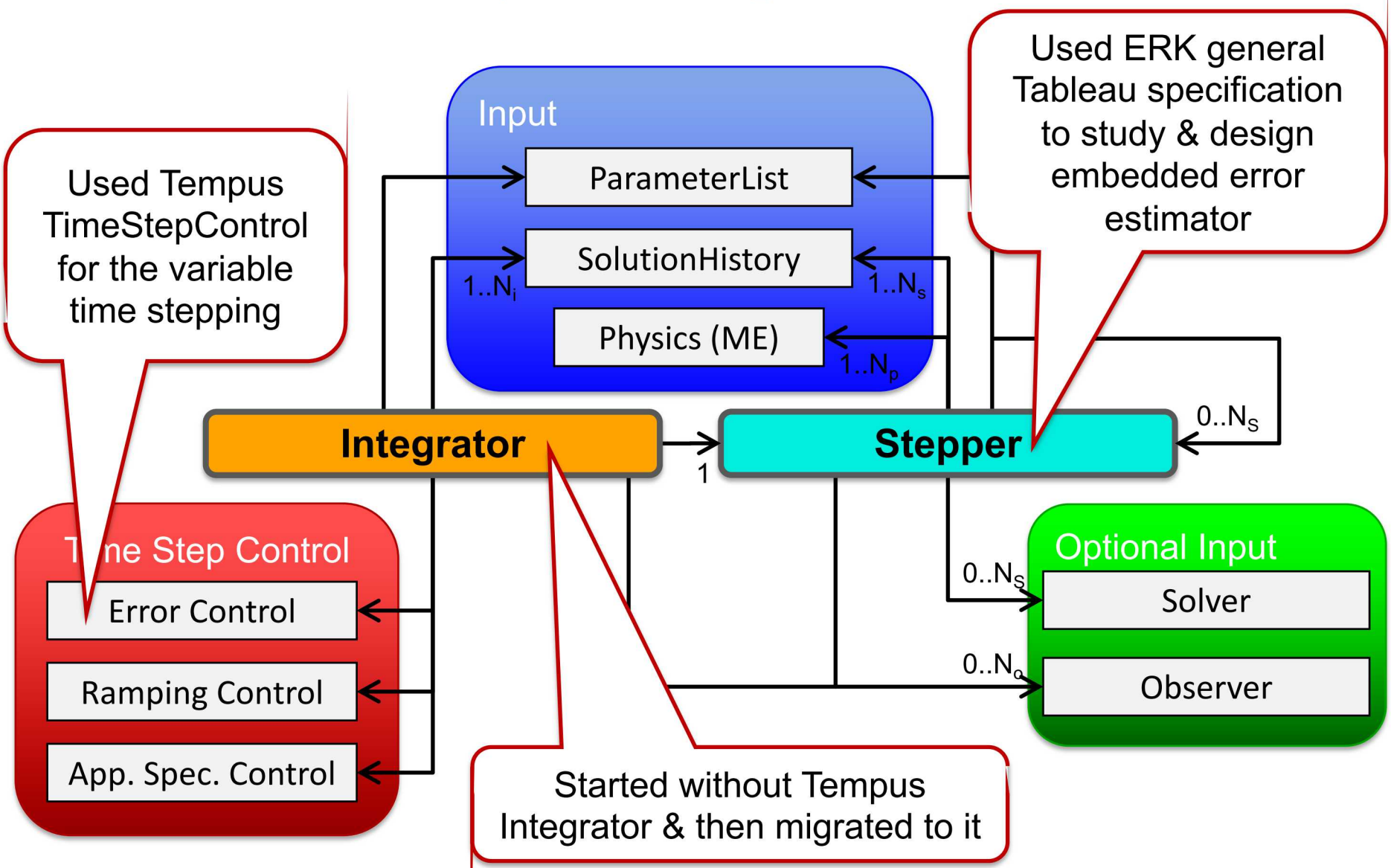
Designed specially to be relatively unstable prior to the stability boundary of the solver.

Similar for other canonical problems

- Viscous shock
- Isentropic Vortex
- Manufactured Solution



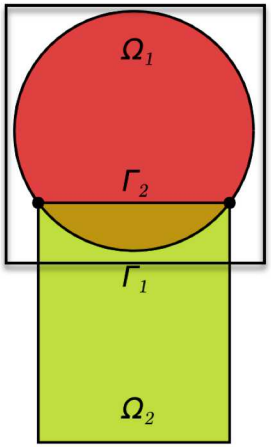
SPARC's Tempus Usage



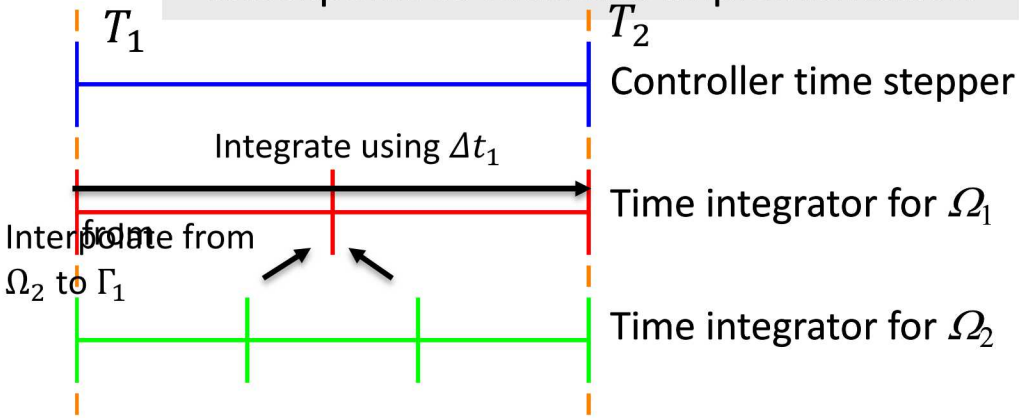
Albany/LCM - Schwarz Alternating Method for Dynamic Multiscale Coupling

CENTER FOR COMPUTING RESEARCH
Sandia National Laboratories

Dev: Tezaur, Mota, Phlipot



Controller time stepper = convenient checkpoint to facilitate implementation



Step 0: Initialize $i = 0$ (controller time index).

Step 1: Advance Ω_1 solution from time T_i to time T_{i+1} using time-stepper in Ω_1 with time-step Δt_1, using solution in Ω_2 interpolated to Γ_1 at times $T_i + n\Delta t_1$.

Step 2: Advance Ω_2 solution from time T_i to time T_{i+1} using time-stepper in Ω_2 with time-step Δt_2, using solution in Ω_1 interpolated to Γ_2 at times $T_i + n\Delta t_2$.

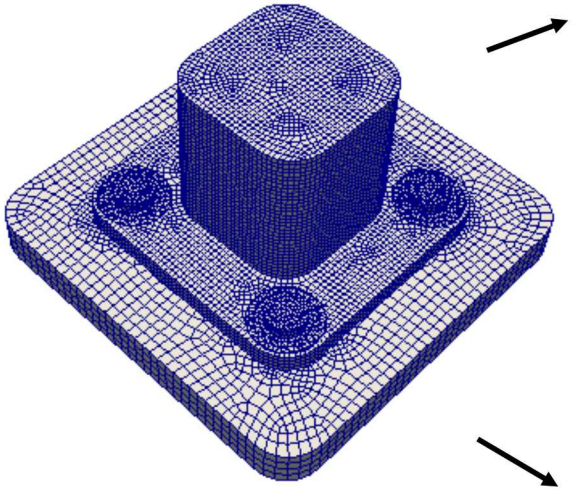
Step 3: Check for convergence at time T_{i+1}.

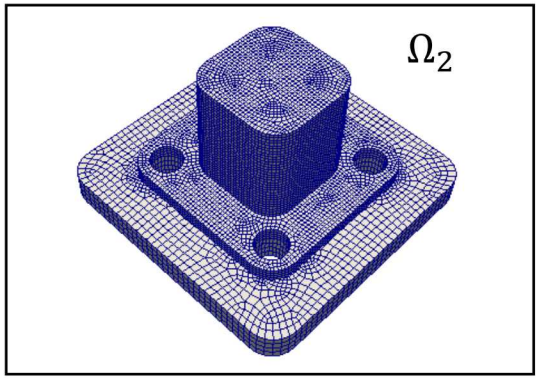
- If unconverged, return to Step 1.

Example #4: Bolted Joint Problem

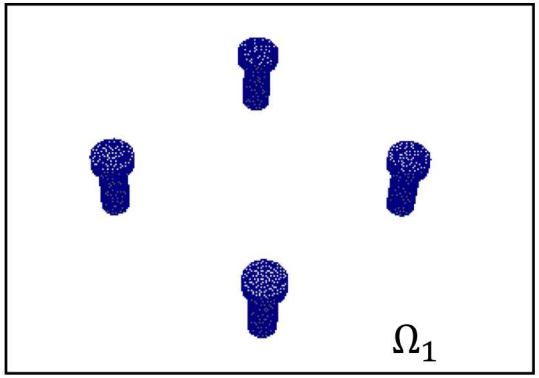
Problem of ***practical scale***.

- Schwarz solution compared to single-domain solution on composite tet 10 mesh.



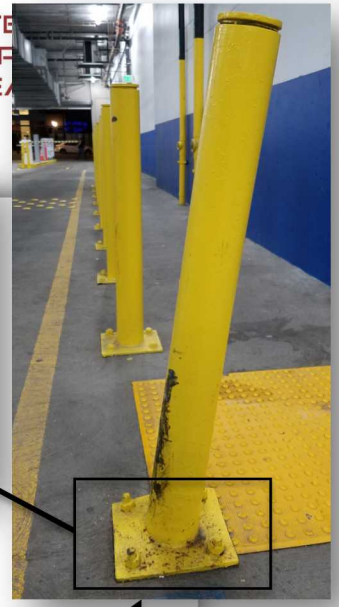


- BC: x-disp = 0.02 at T = 1.0e-3 on top of parts.
- Run until T = 5.0e-4 w/ dt = 1e-5 + implicit Newmark with analytic mass matrix for composite tet 10s.



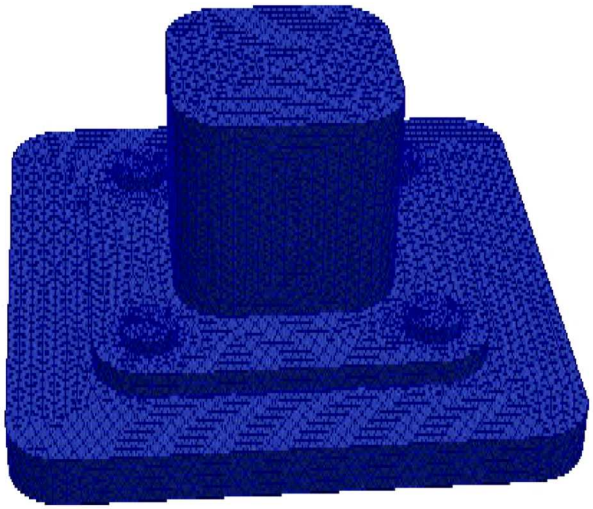
- $\Omega_1 =$ bolts (composite tet 10), $\Omega_2 =$ parts (hex).
- ***Inelastic J_2 material model*** in both subdomains.
  - Ω_1: steel
  - Ω_2: steel component, aluminum (bottom) plate

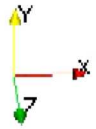
Example #4: Bolted Joint Problem



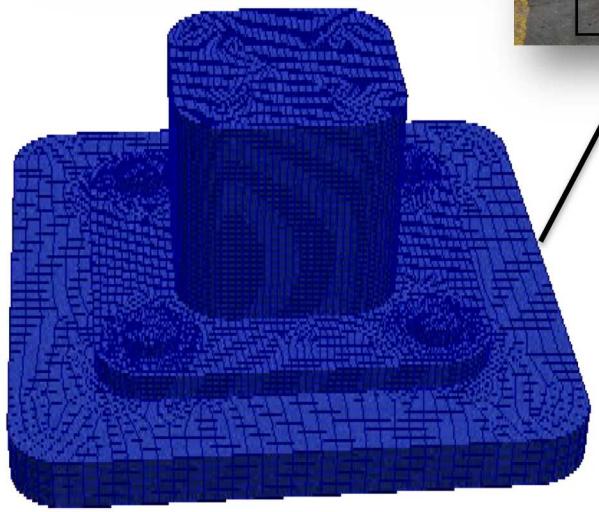
Time: 0.0000000

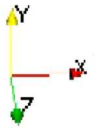
x-displacement





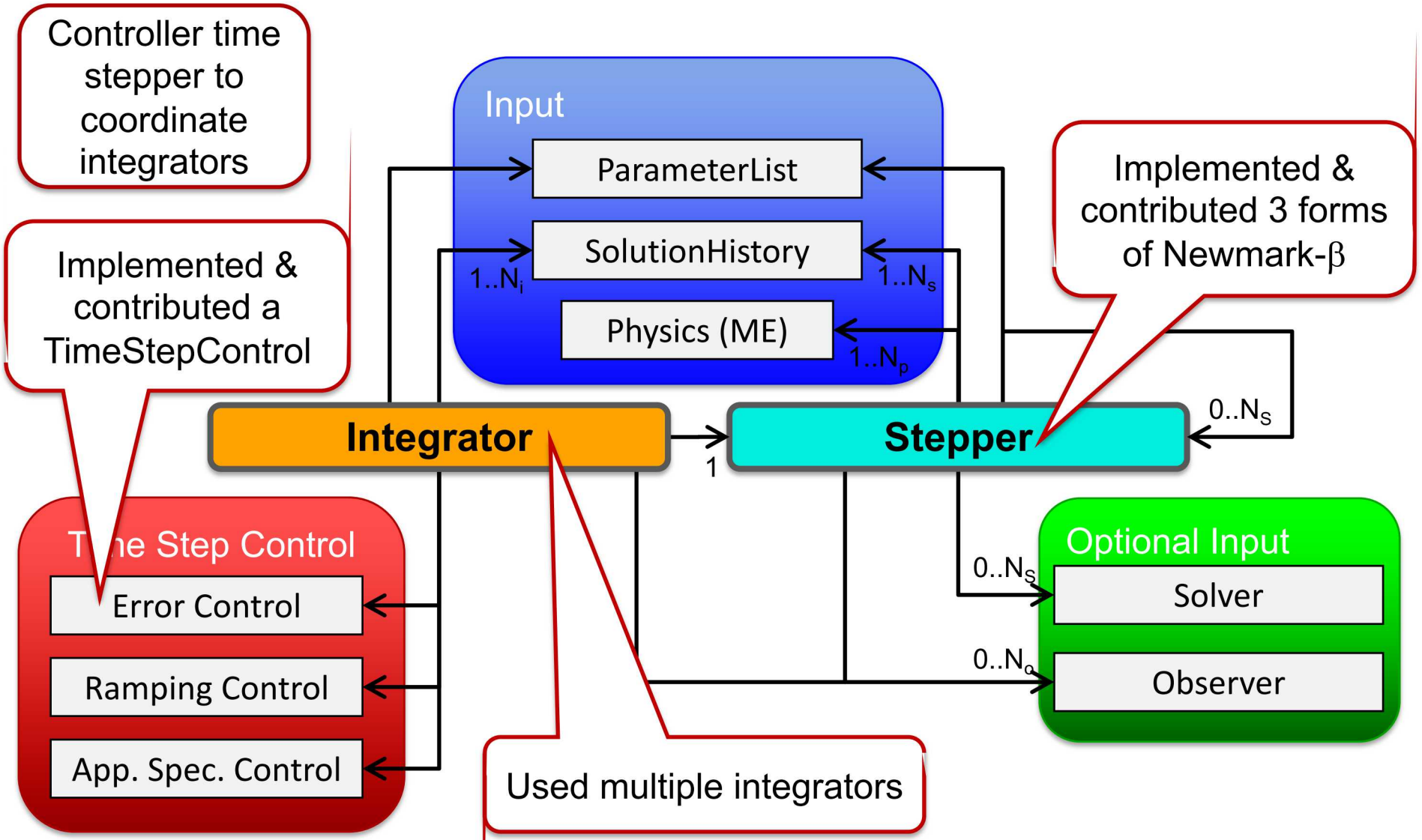
Single Ω





Schwarz

Albany/LCM's Tempus Usage



Summary

- Continued development of Tempus
  - 8 new Steppers
  - Embedded Error analysis
  - Sensitivity Analysis
  - Transient Optimization with ROL for Reduced and full space
- Application usage of Tempus
  - EMPIRE
  - SPARC
  - Albany (LCM and LANDICE)
  - Drekar (not shown here)

Backup Slides

Forward Problems

- Explicit ODE

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{p}(t), t) \qquad \text{for } t \in [t_0, t_f]$$
$$\mathbf{x}(t_0) = \mathbf{x}_0(\mathbf{p}_0)$$

$$\ddot{\mathbf{x}}(t) = \mathbf{f}(\dot{\mathbf{x}}(t), \mathbf{x}(t), \mathbf{p}(t), t) \qquad \text{for } t \in [t_0, t_f]$$
$$\mathbf{x}(t_0) = \mathbf{x}_0(\mathbf{p}_0), \quad \dot{\mathbf{x}}(t_0) = \dot{\mathbf{x}}_0(\mathbf{p}_0)$$

For example

$$\mathbf{M}\,\dot{\mathbf{x}}(t) + \hat{\mathbf{f}}(\mathbf{x}(t), \mathbf{p}(t), t) = 0$$
$$\dot{\mathbf{x}}(t) = -\mathbf{M}^{-1}\,\hat{\mathbf{f}}(\mathbf{x}(t), \mathbf{p}(t), t)$$
$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{p}(t), t)$$

- Fully Implicit DAE/ODE

$$\mathbf{f}(\dot{\mathbf{x}}(t), \mathbf{x}(t), \mathbf{p}(t), t) = 0 \qquad \text{for } t \in [t_0, t_f]$$
$$\mathbf{x}(t_0) = \mathbf{x}_0(\mathbf{p}_0)$$

$$\mathbf{f}(\ddot{\mathbf{x}}(t), \dot{\mathbf{x}}(t), \mathbf{x}(t), \mathbf{p}(t), t) = 0 \qquad \text{for } t \in [t_0, t_f]$$
$$\mathbf{x}(t_0) = \mathbf{x}_0(\mathbf{p}_0), \quad \dot{\mathbf{x}}(t_0) = \dot{\mathbf{x}}_0(\mathbf{p}_0)$$

Tempus::SolutionState

- Primary Design Consideration
  - Encapsulate the state of the solution
  - Should able to restart the integration from a SolutionState
    - Needed for check-pointing and “undo”
- SolutionState contains
  - State – $\mathbf{x}(t), \dot{\mathbf{x}}(t), \ddot{\mathbf{x}}(t)$
  - MetaData – time, index, order, error, restartable, interpolated, …
  - StepperState – data that the stepper needs to restart
  - PhysicsState – any data needed for the physics

managed by Tempus

provided by Application

Tempus::SolutionHistory

- Storage mechanism for the solution history
  - A container of SolutionStates
  - Chronological and index access
  - Manages various access patterns
    - Adjoint sensitivities require storage of forward solution
      - Need check-pointing capabilities, e.g., Griewank
    - Some time integrators need past time steps, e.g., BDF methods
- Provide interpolation capabilities
  - Useful for adjoint sensitivities, e.g., stages of Runge-Kutta
  - Warning – interpolated solutions do not likely satisfy conservation!
  - Scavenge from Rythmos::InterpolationBuffer
- Possibly use Data Warehouse?
  - Storage can be in a variety locations, e.g.,
    - In-core, on disk, on-processor/GPU and mixed capabilities

Integrator/Stepper

- Fully Implicit DAE/ODE

$$\mathbf{f}(\ddot{\mathbf{x}}(t), \dot{\mathbf{x}}(t), \mathbf{x}(t), \mathbf{p}, t) = 0 \qquad \text{for } t \in [t_i, t_{i+1}] \quad \text{for } i = 0, \ldots, N-1$$

$$\mathbf{x}(t_i) = \mathbf{x}_i(\mathbf{p})$$
$$\dot{\mathbf{x}}(t_i) = \dot{\mathbf{x}}_i(\mathbf{p})$$
$$\ddot{\mathbf{x}}(t_i) = \ddot{\mathbf{x}}_i(\mathbf{p})$$

- Integrators
  - Is the time "loop"
  - Has a single Stepper
  - Determines time step size
  - Output results
  - Holds the SolutionHistory
  - Does not call ModelEvaluator
  - Interact w/App thru Observers

- Steppers
  - Take a **single** time step (PASS/FAIL)
  - Steppers require
    - ModelEvaluator(s),
    - SolutionState(s), and/or
    - Solver(s)
  - Can suggested dt
  - Can have sub-Steppers

Tempus::TimeStepControl

- Determine the time step for the Integrator
- Base class has basic capabilities
  - Simulation time min/max bounding
  - Time index min/max bounding
  - Time step size min/max bounding
  - Relative/Absolute maximum error
  - Order min/max bounding
  - Time-step adjustments for output
  - Maximum number of failures and consecutive failures
  - Ensure constant time steps
  - Incorporate Stepper suggested time step
- Derived classes for additional controls, e.g., ramping
- Also have application specific time-step control through Observer

Observer

- Observers are a means to “inject” app-defined functions within the time integration.
- Integrators and Steppers will have observers after every major component, e.g.,
  - Integrators
    - observeStartIntegrator()
    - observeStartTimeStep()
    - observeNextTimeStep()
    - observeBeforeTimeStep()
    - observeAfterTimeStep()
    - observeAcceptTimeStep()
    - observeEndIntegrator()
  - Stepper
    - observeStartStepper()
    - …
    - observeAcceptStep()
    - observeEndStepper()
- If Observer is not sufficient, application can “build their own” Integrator or Stepper.