

# PyPinT

Towards a framework for rapid prototyping of iterative parallel-in-time algorithms

May 28, 2014 | Torbjörn Klatt, Dieter Moser, Robert Speck | 3rd Workshop on Parallel-in-Time Integration

# Overview

- 1 Recap of existing Parallel-in-Time Approaches
- 2 The *PyPinT* Framework Explained
- 3 Goals of *PyPinT*
- 4 Proof of Concept



# PyPinT

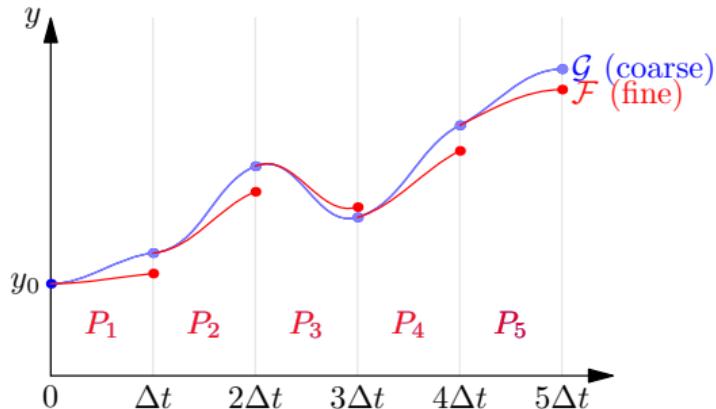
## Part I: Existing Parallel-in-Time Approaches

May 28, 2014 | Torbjörn Klatt, Dieter Moser, Robert Speck

# Parareal

## Reference

- coarse  $\mathcal{G}$  and fine  $\mathcal{F}$  propagators make parareal flexible and modular
- an initial value is improved iteratively
- order is controllable through the fine propagator  $\mathcal{F}$



[Courtesy of M. Emmett, LBNL]

$$y_{m+1}^{k+1} = \mathcal{F}(y_m^k) + \mathcal{G}(y_m^{k+1}) - \mathcal{G}(y_m^k)$$

# Integral Deferred Corrections

## Reference

$$y'(t) = f(t, y(t)), \quad y(0) = y_0$$

$$y(t) = y_0 + \int_0^t f(\tau, y(\tau)) d\tau, \quad y(0) = y_0$$

Using the residual

$$r(t) = f(t, \tilde{y}(t)) - \tilde{y}'(t)$$

to compute the error

$$\begin{aligned} e'(t) &= f(t, \tilde{y} + e) - f(t, \tilde{y}) + r'(t) \\ e(0) &= 0 \end{aligned}$$

for the next update

$$\tilde{y}_{j+1} = \tilde{y}_j + e_j$$

Using the residual

$$r(t) = y_0 - \tilde{y}(t) + \int_0^t f(\tau, y(\tau)) d\tau$$

to compute the error

$$e(t) = \int_0^t f(\tau, \tilde{y} + e) - f(\tau, \tilde{y}) d\tau + r(t)$$

for the next update

$$\tilde{y}_{j+1} = \tilde{y}_j + e_j$$

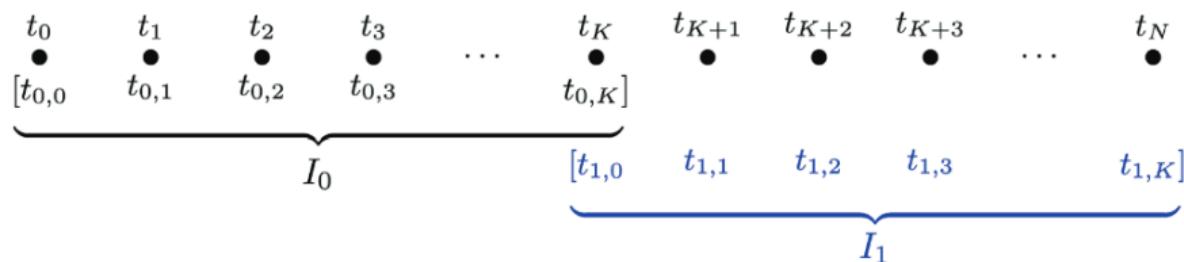
# Revisionist IDC

## Reference

One equation

$$u_{n+1}^{k+1} = u_n^{k+1} + \Delta t [F(u_{n+1}^{k+1}) - F(u_{n+1}^k)) + \mathcal{I}_n^{n+1}(F(\mathbf{u}^k))]$$

different strategies of computation



What is needed? Integrator, Evaluation of  $F$

# Multilevel SDC

## Reference

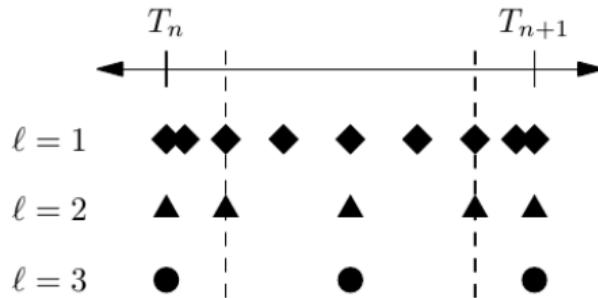
One equation

$$u_{n+1}^{k+1} = u_n^{k+1} + \Delta t [F(u_{n+1}^{k+1}) - F(u_{n+1}^k)] + \mathcal{I}_n^{n+1}(F(\mathbf{u}^k)) + \tau_n^k$$

different strategies of computation

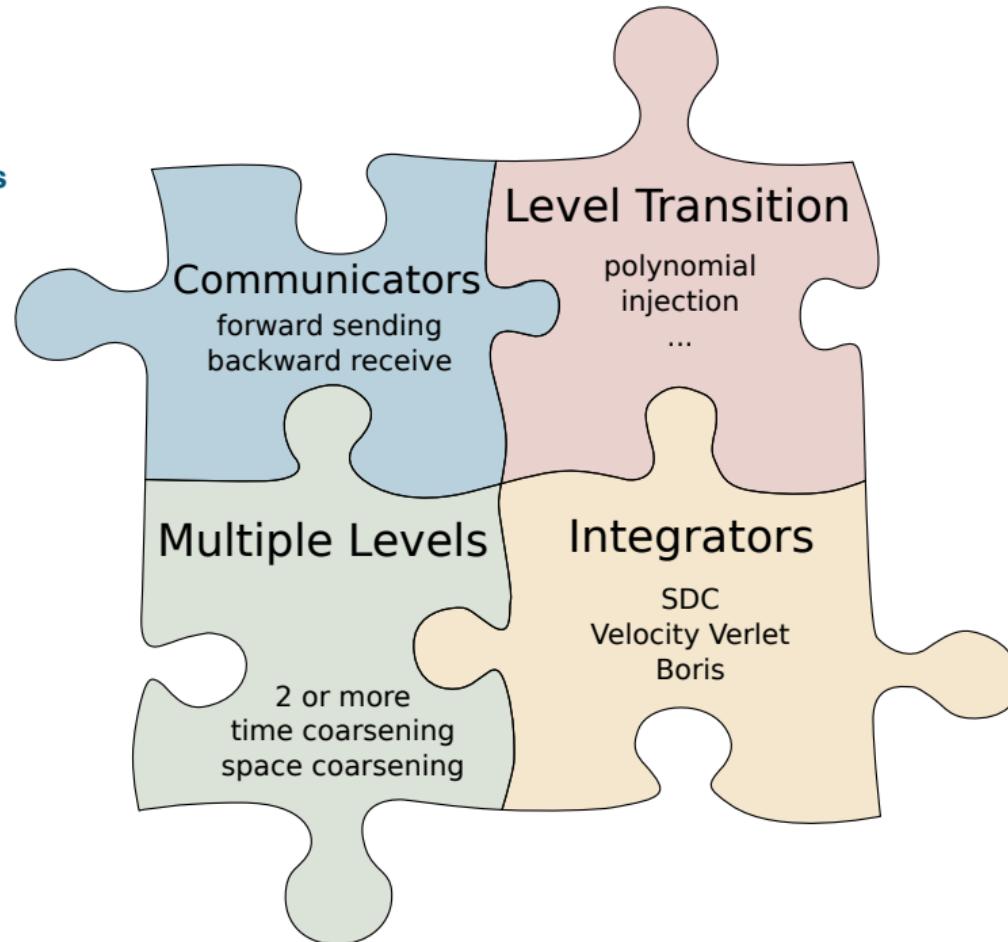
What is needed?

- Integrator
- Evaluation of  $F$
- Interpolation
- Restriction



# PFASST

## Building Blocks





# PyPinT

## Part II: The PyPinT Framework Explained

May 28, 2014 | Torbjörn Klatt, Dieter Moser, Robert Speck

## Basic Concept

- *Python ≥3.2* as language of choice
  - for ease of use and extensibility (cf. *NumPy*, *SciPy*)
- modular building blocks
  - for fast exchange of algorithms' building blocks
- well-conceived and intuitive abstract interfaces
  - for reusable code ensuring DRY principle
- integrated analyzation tools
  - for introspection and plotting (cf. *matplotlib*)
- usage of a sophisticated unit-testing framework
  - nobody writes bug-free code

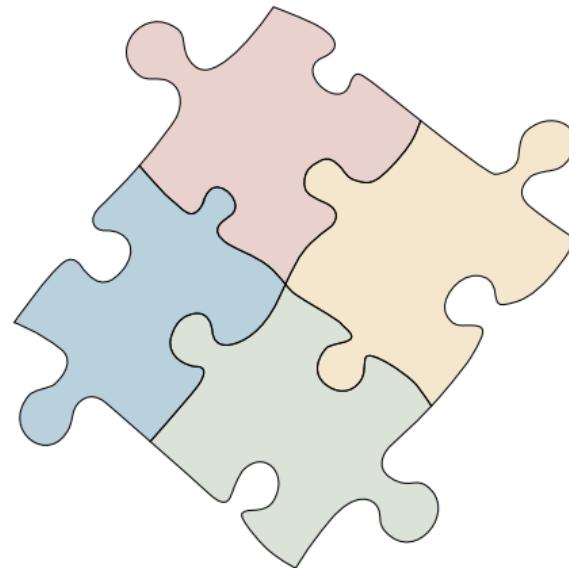


# Modules

## Abstract Modelling of PinT Algorithms

### pypint

- ~.problems
- ~.solvers
- ~.integrators
- ~.communicators
- ~.multi\_level\_providers
- ~.solutions
- ~.plugins

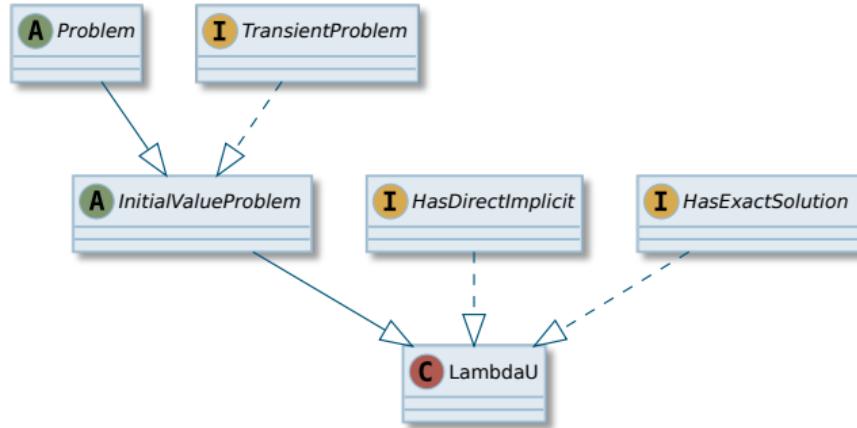


# Modules

## Abstract Modelling of PinT Algorithms

```
pypint
~.problems
~.solvers
~.integrators
~.communicators
~.multi_level_providers
~.solutions
~.plugins
```

- interfaces for problem setups
- generic problem with specializations via mixins



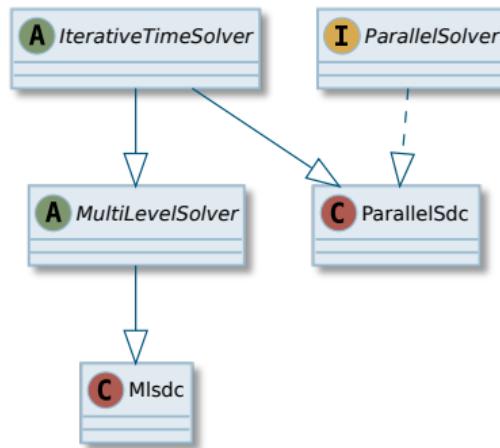
# Modules

## Abstract Modelling of PinT Algorithms

pypint

```
~.problems
~.solvers
~.integrators
~.communicators
~.multi_level_providers
~.solutions
~.plugins
```

- interfaces for iterative time solvers
- providing generic building blocks of solvers



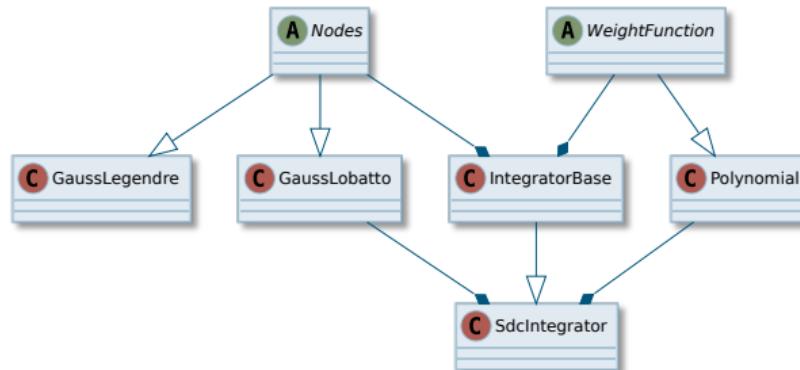
# Modules

## Abstract Modelling of PinT Algorithms

pypint

- ~.problems
- ~.solvers
- ~.integrators
- ~.communicators
- ~.multi\_level\_providers
- ~.solutions
- ~.plugins

- providers of various quadrature nodes and weight functions  
e.g. Gauss-Lobatto, Gauss-Legendre, polynomial, etc.
- providers aggregated in integrators  
for intuitive methods:  
.q\_matrix.apply(data) and .s\_matrix.apply(data, target\_node)



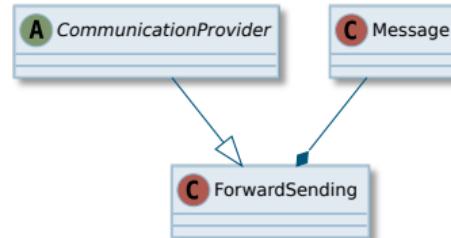
# Modules

## Abstract Modelling of PinT Algorithms

pypint

```
~.problems
~.solvers
~.integrators
~.communicators
~.multi_level_providers
~.solutions
~.plugins
```

- generic abstract interfaces for communication patterns e.g. implemented as *forward sending*, etc.
- extendable basic message buffer holding data, solver flags and other meta information

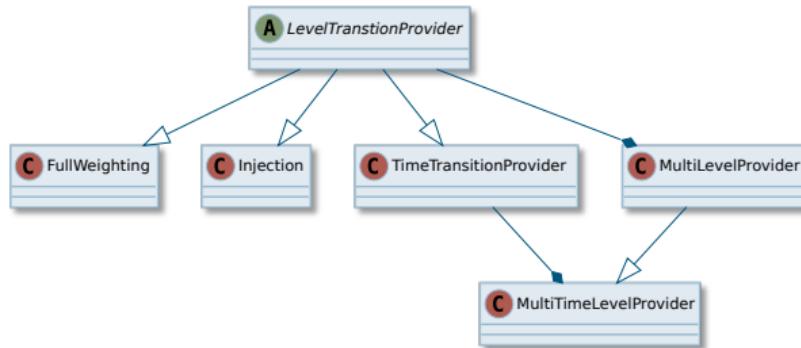


# Modules

## Abstract Modelling of PinT Algorithms

```
pypint
~.problems
~.solvers
~.integrators
~.communicators
~.multi_level_providers
~.solutions
~.plugins
```

- abstract interface for generic level transitions  
e.g. *full weighting, injection, Lagrange-polynomial integration* etc.
- containers for multiple levels  
providing access to integrators of each level
- unified generic interface for transitions between levels  
via methods `.restrict(data, fine_id, coarse_id)` and  
`.prolongate(data, coarse_id, fine_id)`



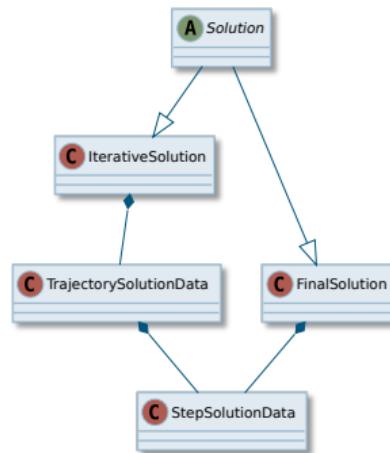
# Modules

## Abstract Modelling of PinT Algorithms

pypint

```
~.problems
~.solvers
~.integrators
~.communicators
~.multi_level_providers
~.solutions
~.plugins
```

- containers for solution data + meta information
  - e.g. time-node/step-wise, trajectory (consecutive time nodes)
- container interface for complete solutions
  - e.g. only last time node of last iteration or all nodes of all iterations



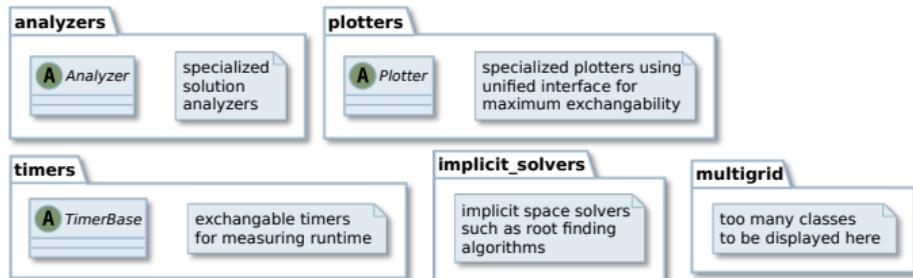
# Modules

## Abstract Modelling of PinT Algorithms

pypint

- ~.problems
- ~.solvers
- ~.integrators
- ~.communicators
- ~.multi\_level\_providers
- ~.solutions
- ~.plugins

- containers for solution data + meta information
  - e.g. time-node/step-wise, trajectory (consecutive time nodes)
- container interface for complete solutions
  - e.g. only last time node of last iteration or all nodes of all iterations





# PyPinT

## Part III: Goals for PyPinT

May 28, 2014 | Torbjörn Klatt, Dieter Moser, Robert Speck

## Goals for PyPinT

- providing generic framework for many PinT algorithms

## Goals for PyPinT

- providing generic framework for many PinT algorithms  
⇒ easy comparability of different algorithms' building blocks

## Goals for PyPinT

- providing generic framework for many PinT algorithms  
⇒ easy comparability of different algorithms' building blocks
- intuitive user-expandability through well-defined interfaces

## Goals for PyPinT

- providing generic framework for many PinT algorithms  
⇒ easy comparability of different algorithms' building blocks
- intuitive user-expandability through well-defined interfaces  
⇒ fast exchangability and prototyping of different building blocks

## Goals for PyPinT

- providing generic framework for many PinT algorithms  
⇒ easy comparability of different algorithms' building blocks
- intuitive user-expandability through well-defined interfaces  
⇒ fast exchangability and prototyping of different building blocks
- well-tested and well-documented implementations of PinT algorithms

## Goals for PyPinT

- providing generic framework for many PinT algorithms  
⇒ easy comparability of different algorithms' building blocks
- intuitive user-expandability through well-defined interfaces  
⇒ fast exchangability and prototyping of different building blocks
- well-tested and well-documented implementations of PinT algorithms  
⇒ suited for educational use

## Goals for PyPinT

- providing generic framework for many PinT algorithms  
⇒ easy comparability of different algorithms' building blocks
- intuitive user-expandability through well-defined interfaces  
⇒ fast exchangability and prototyping of different building blocks
- well-tested and well-documented implementations of PinT algorithms  
⇒ suited for educational use
- open-source hosted on  GitHub

## Goals for PyPinT

- providing generic framework for many PinT algorithms  
⇒ easy comparability of different algorithms' building blocks
- intuitive user-expandability through well-defined interfaces  
⇒ fast exchangability and prototyping of different building blocks
- well-tested and well-documented implementations of PinT algorithms  
⇒ suited for educational use
- open-source hosted on  GitHub  
⇒ reaching young & rising developers and scientists promoting PinT algorithms



# PyPinT

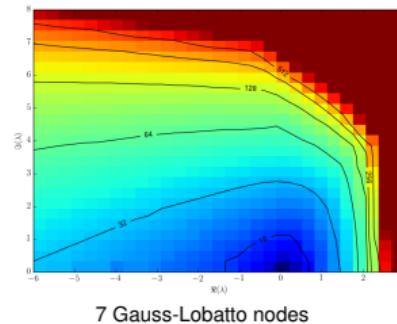
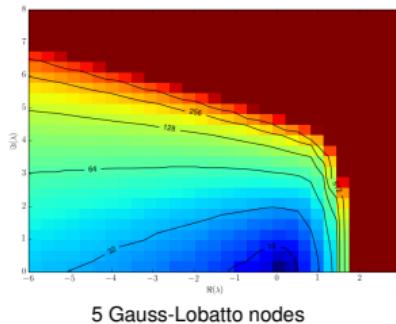
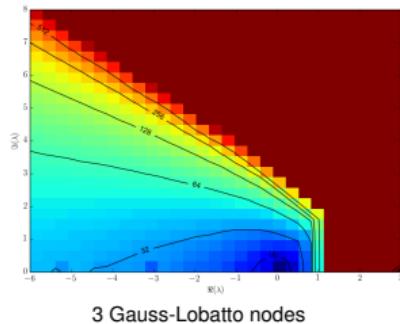
## Part IV: Proof of Concept

May 28, 2014 | Torbjörn Klatt, Dieter Moser, Robert Speck

# Implementation of SDC and MLSDC

Convergence Regions (number iterations for residual  $\leq 10^{-14}$ ) of  $u'(x, t) = \lambda u(x, t)$ ,  $t \in [0, 1]$ ,  $\lambda \in \mathbb{C}$

SDC



MLSDC



# Implementation of SDC and MLSDC

## Python Script for a SDC run with 3 Gauss-Lobatto nodes

```
problem = LambdaU(lmbda=complex(-1.0, 1.0))

comm = ForwardSendingMessaging()
solver = ParallelSdc(communicator=comm)

comm.link_solvers(previous=comm, next=comm)
comm.write_buffer(value=problem.initial_value,
                  time_point=problem.time_start)

solver.init(problem=problem, num_nodes=3)

solution = solver.run(SemiImplicitSdcCore)
```



# Thank you for your attention!

## Questions?

(now or later)

PyPinT is on  GitHub: <https://github.com/Parallel-in-Time/PyPinT>

Torbjörn Klatt  
Juelich Supercomputing Centre  
Building 16.3, Room 022  
Tel: +49 2461 61 96452  
eMail: t.klatt@fz-juelich.de

Dieter Moser  
Juelich Supercomputing Centre  
Building 16.3, Room 022  
Tel: +49 2461 61 96453  
eMail: d.moser@fz-juelich.de

Robert Speck \*  
Juelich Supercomputing Centre  
Building 16.3, Room 131  
Tel: +49 2461 61 1644  
eMail: r.speck@fz-juelich.de

\* corresponding author