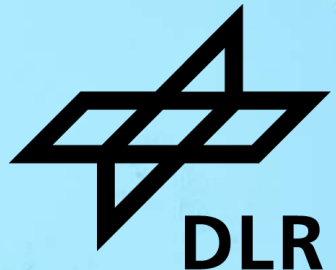


SPLISS: EINSATZ EINES STANDARDISIERTEN GLEICHUNGSSYSTEMLÖSERS IN CFD-ANWENDUNGEN

Olaf Krzikalla et.al.

HPCN Workshop 2026 Dresden



Spliss – A sparse linear system solver for aeronautical CFD



Common library for (approximatively) solving a linear equation system with characteristics from aeronautical CFD in an HPC ecosystem



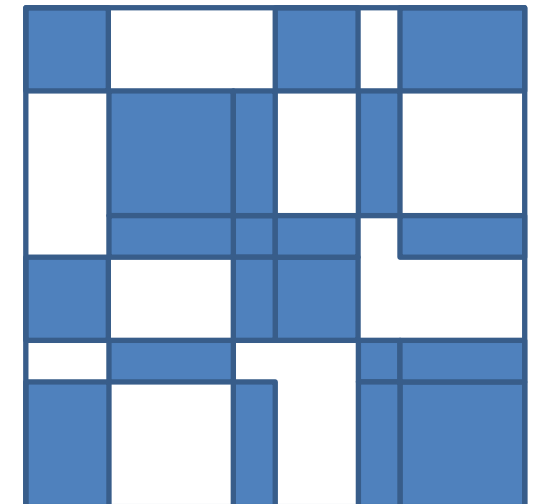
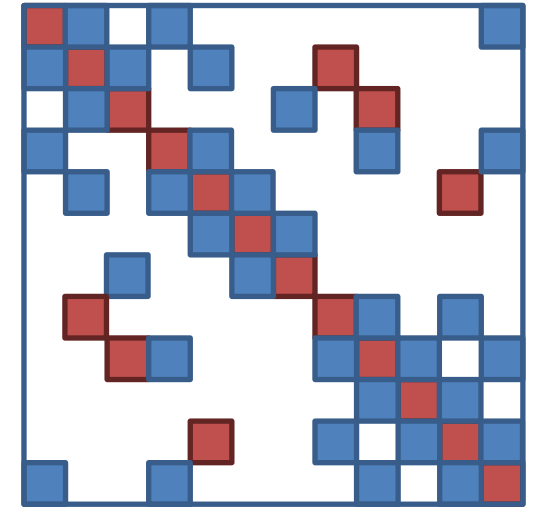
Spliss – A sparse linear system solver for aeronautical CFD



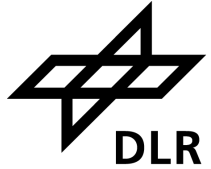
Common library for (approximatively) solving a linear equation system with characteristics from aeronautical CFD in an HPC ecosystem

Sparse Matrices

- Dense blocks with a fixed block size or variable block sizes
- Mixed data types: e.g. some entries are complex, others real



Spliss – A sparse linear system solver for aeronautical CFD



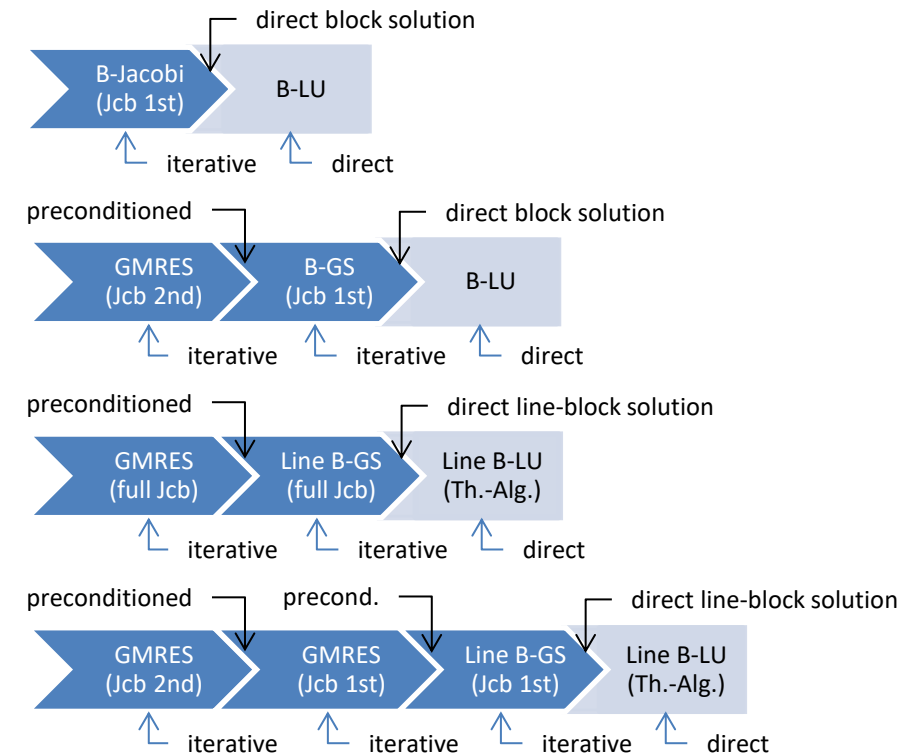
Common library for (approximatively) solving a linear equation system with characteristics from aeronautical CFD in an HPC ecosystem

Sparse Matrices

- Dense blocks with a fixed block size or variable block sizes
- Mixed data types: e.g. some entries are complex, others real

Solver

- Combinable components (as preconditioner)
- Robust methods for stiff CFD problems



A brief history of Spliss



Motivation behind a separate development:

- focus on contemporary and upcoming HPC architectures
- tailor specifically for CFD applications
- can be used at least across the DLR



Precondition: Spliss and CODA are developed in parallel

- demands for linear operators, matrix and solver configurations may arise during CODA development and usage
- HPC optimizations should be instantly usable in CODA

Spliss in CODA



Integration in CODA aims for maximum flexibility:

- Python dictionary for solver configuration
- supports matrix-free operators

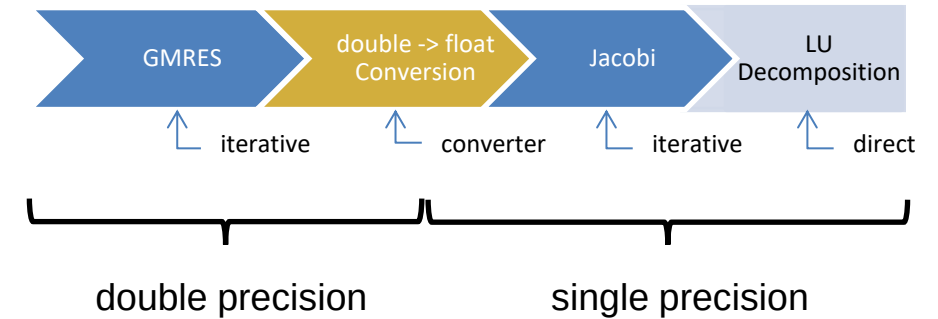
```
"solver stack": [  
  {  
    "type": "rGMRES",  
    "requested tolerance": 0.1,  
    "maximum iterations": 30,  
    "Krylov subspace size": 30,  
    "use flexible variant": False,  
    "orthogonalization mode": "auto",  
    "Jacobian": {  
      "type": "matrix-free AD",  
    },  
  },  
  {  
    "type": "Jacobi",  
    "requested tolerance": 0.0,  
    "maximum iterations": 50,  
    "relaxation": 0.9,  
    "Jacobian": {  
      "type": "matrix",  
    },  
  },  
  {  
    "type": "Lines Inversion",  
  },  
]
```

Evolution of solver configurations in CODA

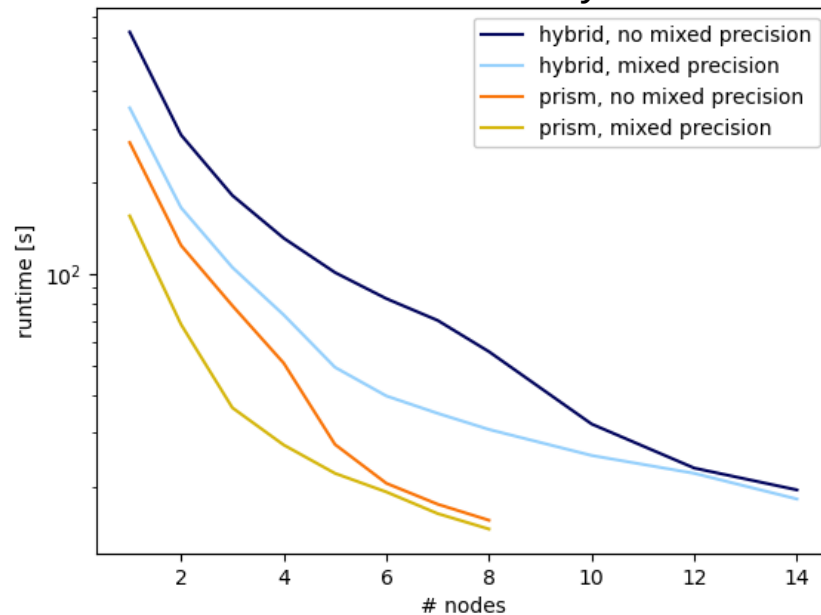
Mixed Precision Solver Component

Reduce memory footprint of inner hot loops

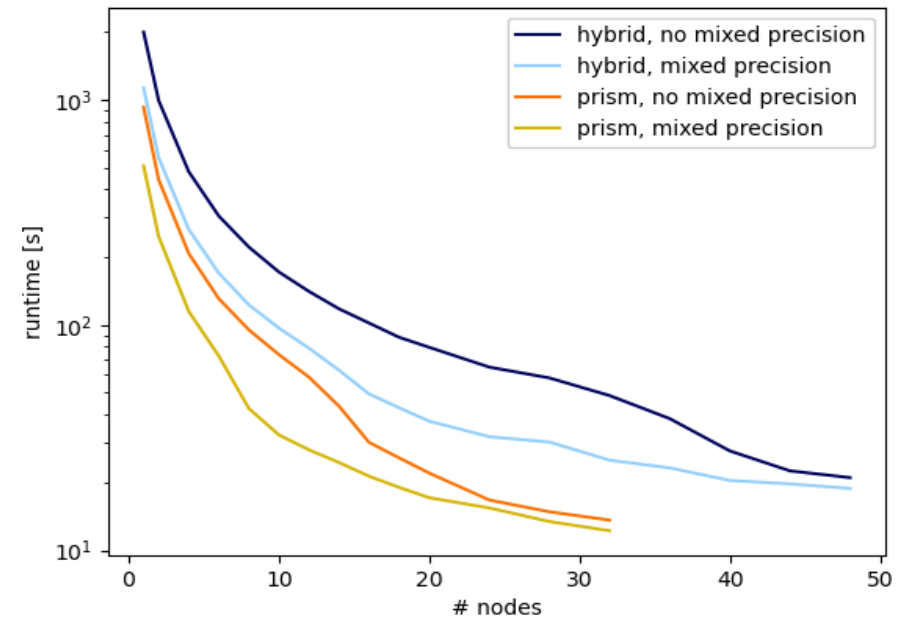
→ Now considered best practice in CODA



Runtimes for tiny mesh



Runtimes for coarse mesh

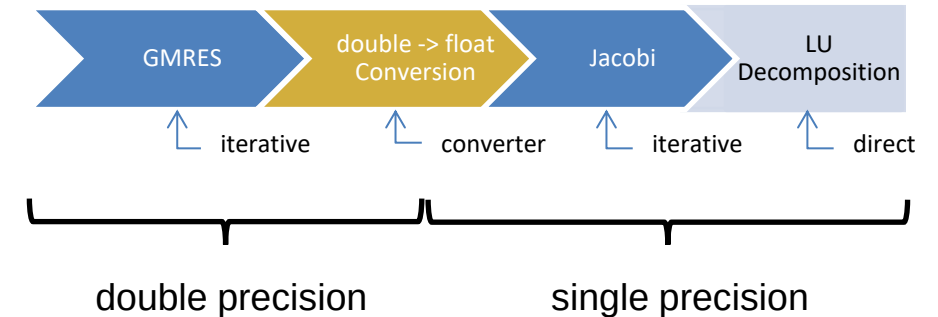


Evolution of solver configurations in CODA

Mixed Precision Solver Component

Reduce memory footprint of inner hot loops

→ Now considered best practice in CODA



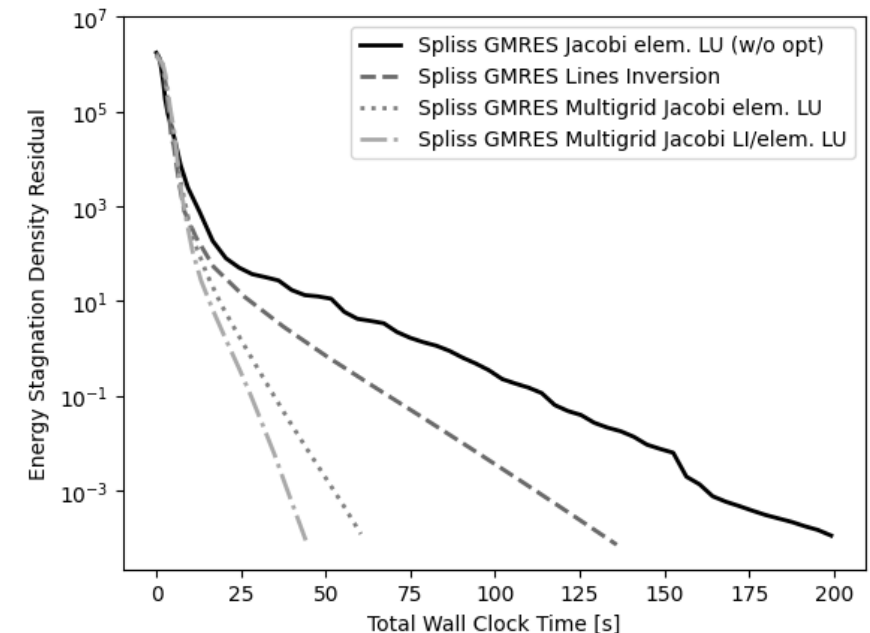
Multigrid solver

Comprehensive study on solver configurations during CODA EIS

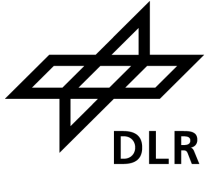
→ Linear multigrid is significantly faster in almost any testcase

→ Lines inversion is a must have for multigrid

Comparison of Spliss Algorithms Tested



Evolution of solver configurations in CODA



Current standard Spliss solver configurations in CODA:

- all configurations provide an optional mixed-precision adapter
 - Mostly the default option
- matrix-free operators for outer solvers

Other solver configurations possible, e.g. via Python interface or FSUI.

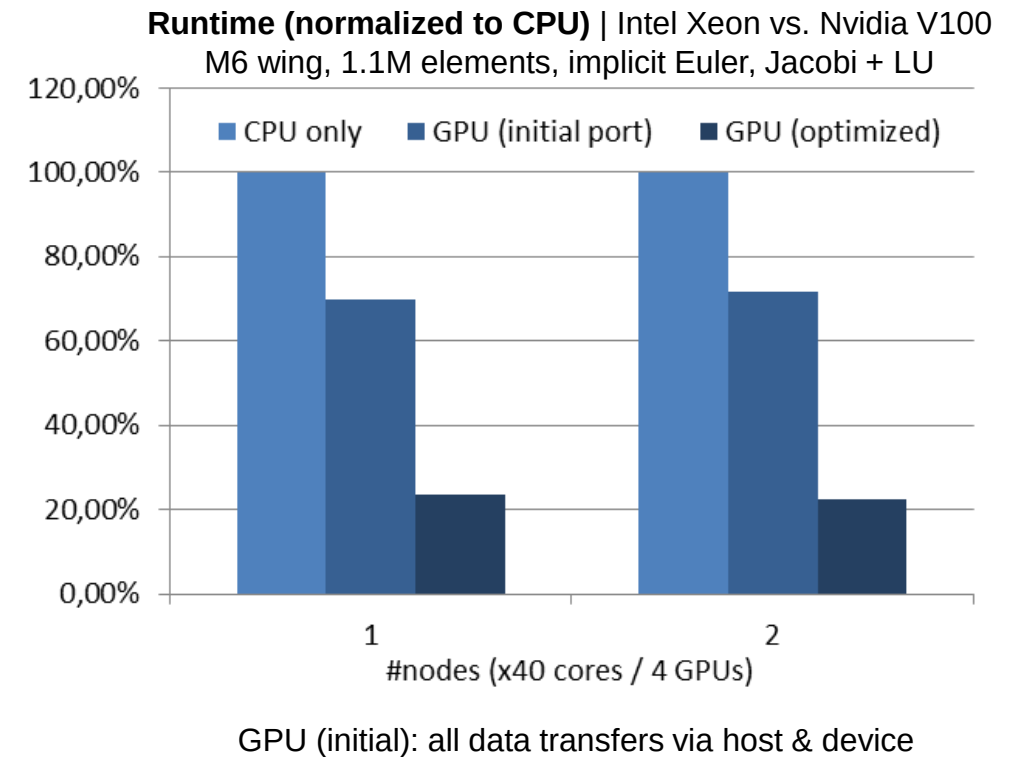
GMRES
JacobiLU
JacobiLU
JacobiLinesInversion
BiCGSTABLU
GaussSeidelLU
(Restarted)GMRESLU
(Restarted)GMRESJacobiLU
(Restarted)GMRESJacobiLU
RestartedGMRESJacobiLinesInversion
GMRESFieldSplitJacobiLU
GMRESFieldSplitJacobiLU
GMRESFieldSplitJacobiLinesInversion
GCROdynDRLU
GCROdynDRJacobiLU
GCROdynDRJacobiLinesInversion
GCROdynDRJacobiLU
RestartedGCROdynDRGaussSeidelLU
RestartedGMRESGaussSeidelLU
RestartedFGMRESGMRESILU
RestartedGMRESLMGLinesInversionLU

Evolution of HPC optimizations in CODA



Effect of HPC optimizations in CODA

- Nvidia MPS (multi-process service) allows multiple processes to efficiently offload to the same GPU
- CUDA-aware MPI communication to eliminate unnecessary device-to-host transfers
- GPUDirect accelerations at runtime allow communication without involvement of host memory



Precondition: Spliss acts as a replacement for the built-in Gauss-Seidel (GS) & Symmetric Successive Over-Relaxation (SSOR) solvers, which

- are used for structured and unstructured grids
- are specifically optimized for properties of structured grids
- operate mostly on fixed block size (5), sometimes on dynamic block size

Direct integration of solver configurations in the TRACE code:

```
auto stack = InitSolverStack(matrix);  
stack.template Append<LUDecomposition>();  
stack.template Append<GaussSeidel>(criterion, ...);
```

Evolution of Spliss due to TRACE



Optimizations in Spliss driven by TRACE experiences

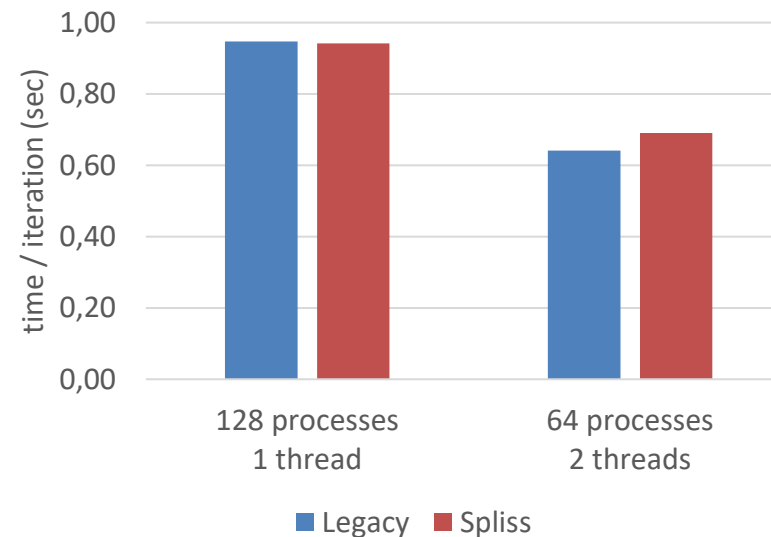
Spliss optimization	Approximate performance gains
Permutation; cache optimizations	50%
Access to raw data	5 - 20%
Eisenstat trick	30%
Communication for Color 0 only ¹	200 - 500%

(1) just recently added to Spliss; before that, it had been a patch in TRACE

Spliss in TRACE



typical two-stage turbine configuration (~10 mil nodes)
measured on CARA turin



Advantages of Spliss in TRACE:

- GPU support
- GMRES enables particular Low-Mach and Harmonic-Balance configurations

Precondition: Spliss is meant to serve as a substitute for the built-in solvers

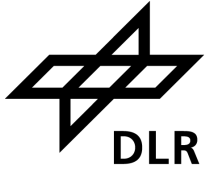
- one particular block size defined at compilation time
- HYDRA is written in Fortran → dedicated Spliss-HYDRA interface required
- Vision: text-based interface (similar to FSUI-CODA)

```
{
  "solver": {
    "type": "rGMRES",
    "use flexible variant": true,
    "requested tolerance": 0.1,
    "Krylov subspace size": 20,
    "maximum iterations": 20,
    "orthogonalization mode": "classical Gram-Schmidt",
    "successor": {
      "type": "ScalarTypeAdapter",
      "mixed precision type": "matrixOnly",
      "successor": {
        "type": "Jacobi",
        "requested tolerance": 1e-3,
        "maximum iterations": 50,
        "relaxation": 0.8,
        "successor": {
          "type": "Lines Inversion",
          "line definition threshold": 10
        }
      }
    }
  }
}
```

Why Spliss?

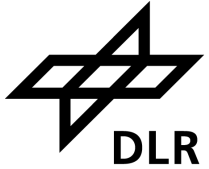
There are plenty of other solver libraries available.

A conjecture about in-house software development



P = a relevant problem (includes target hardware)
 S_{own} = in-house software
 S_{3p} = third-party software

A conjecture about in-house software development



$$\forall t \exists P \exists S_{3p} (S_{3p}(P) \geq S_{own}(P))$$

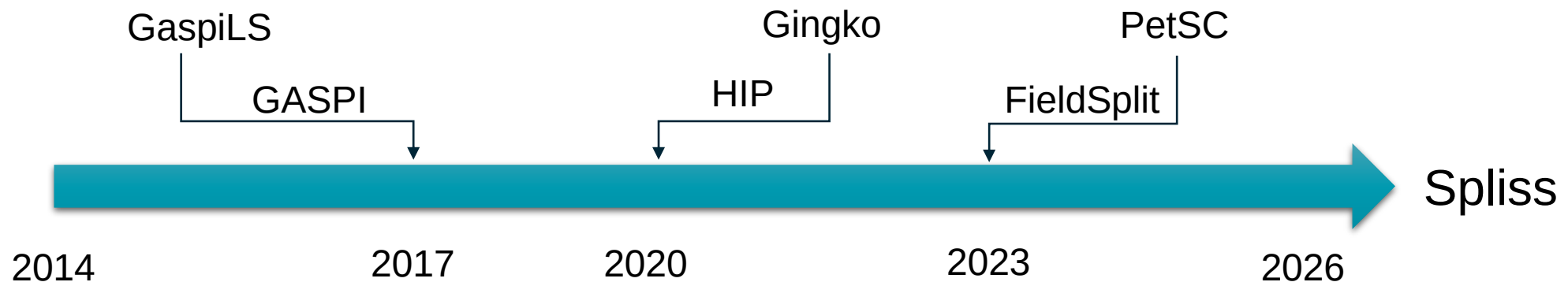
where:

t = product life time

P = a relevant problem (includes target hardware)

S_{own} = in-house software

S_{3p} = third-party software



A conjecture about in-house software development

$$\exists S_{3p} \forall t \forall P (S_{3p}(P) \geq S_{own}(P))$$

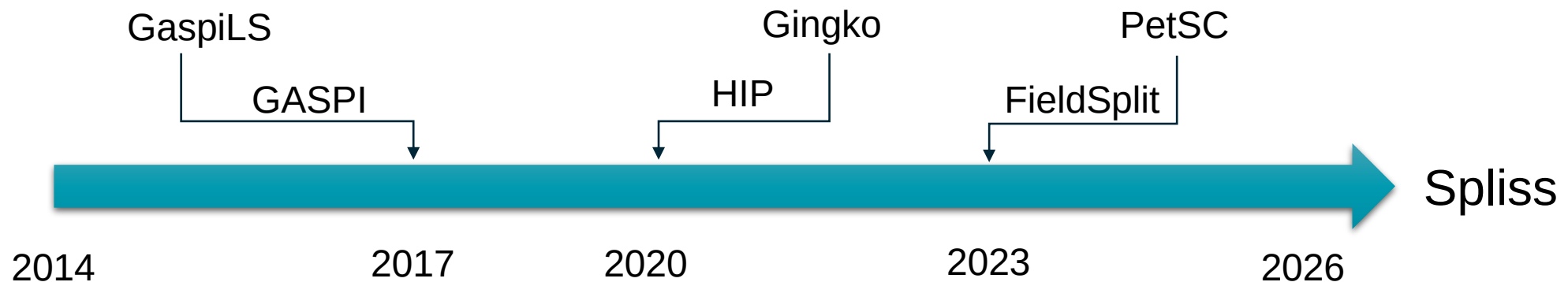
where:

t = product life time

P = a relevant problem (includes target hardware)

S_{own} = in-house software

S_{3p} = third-party software



The reason for in-house software development



$\forall t \forall P \rightarrow \text{Continuity}$

- continuous and consistent development over years
- continuous coverage of the problem space
 - support contemporary and upcoming hardware
 - quick responses to changes in the problem space possible
- continuous pricing and licensing
- continuous cooperation
 - between DLR institutes
 - with external partners

- Growing user base
 - CODA, TRACE, FSMeshDefo are established
 - HYDRA, B2000++ are on the way
- Support for contemporary HPC hardware
 - GPU support is a strong motivation for all users
- Continuous development and implementation of algorithms
 - serves all users

Has Spliss fulfilled its strategic aims?

→ Yes

Topic: **Spliss: Einsatz eines standardisierten
Gleichungssystemlösers in CFD-Anwendungen**

Date: 2026-07-01

Author: Olaf Krzikalla

Institute: DLR-SP

Image sources: All images “DLR (CC BY-NC-ND 3.0)” unless otherwise stated