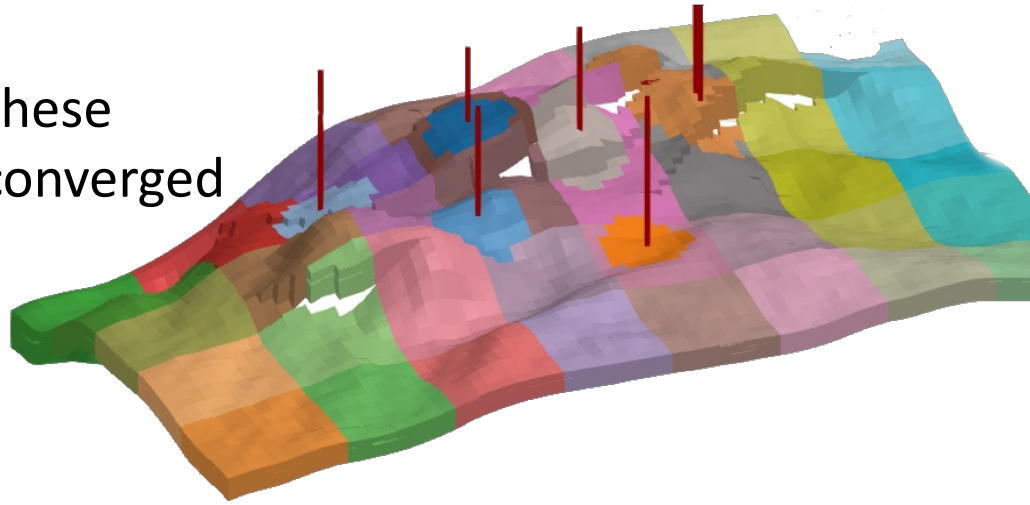
Non-linear domain decomposition

NLDD implementation in OPM Flow

Atgeirr Flø Rasmussen, SINTEF
Olav Møyner, SINTEF

NLDD methods

- Basic idea
  - Newton: does same amount of work for every cell
  - Some cells converge quickly, some slowly
  - Decomposing the domain enables us to separate these
  - Do not waste effort improving areas already well converged
- Basic requirements
  - Partitioning the full problem into smaller domains
  - A way to solve «local» problems



- «NLDD»
  - Alternate between solving locally and performing a global Newton iteration
- ASPEN/ASPIN
  - After solving locally, perform a global update designed to be a Newton update for the full method

Nonlinear domain decomposition

NLDD (simple):

- Solve nonlinear problem in each local domain with Dirichlet boundaries, one full time step
- Do a fully implicit global correction
- Not theoretically guaranteed to preserve quadratic convergence
- Can use Gauss-Seidel ordering (subdomains get updated boundary conditions)

ASPEN (more sophisticated):

- Solve nonlinear problem in each local domain with Dirichlet boundaries, one full time step
- Formulate a nonlinearly preconditioned global system for the overall update
- Improved convergence and reuses Jacobian evaluations from the local solves
- Theoretically well-founded, but not straightforward to handle local solve failures

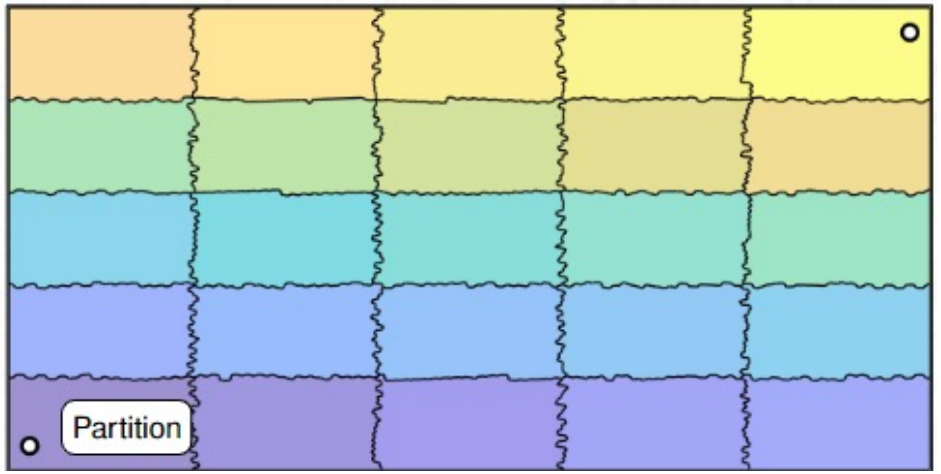
Both methods have been implemented, choice was made to focus on NLDD:

- Gauss–Seidel iterations improve convergence substantially
- ASPEN framework makes it difficult to apply solution strategy adaptively
- Goal is *implementation in commercial-grade codes*: Avoid invasive changes to assembly and convergence checking

Nonlinear domain decomposition

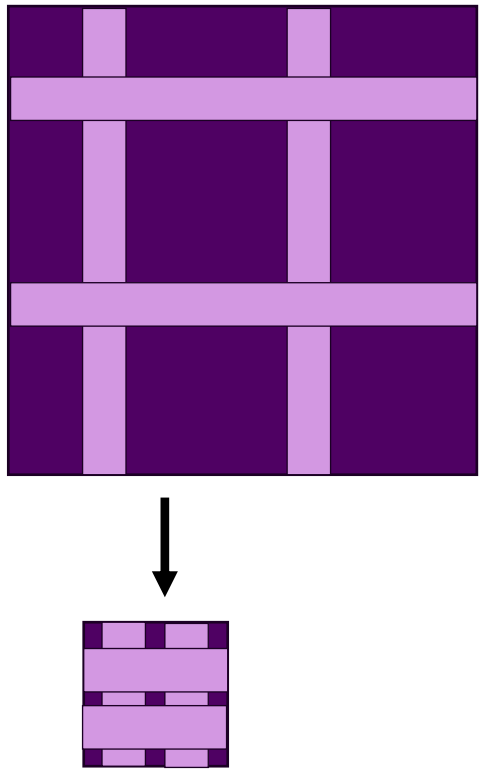
- Divide model into several subdomains (default size ~1000 cells each)
- For each time-step, do the following:
  1. Solve all local domains nonlinearly in sequence from high to low pressure, updating Dirichlet boundary conditions as we go
  2. Use solution from local solves to reassemble global system and do single Newton iteration,
  3. Repeat 1 and 2 until global convergence

Don't worry *too much* about proven properties of global update…



Overview of implementation

- Basic premises:
  - Do not hinder or complicate other developments
  - No performance cost for regular Newton
- Overall structure
  - Global view and matrix always exists
  - If NLDD: local domain views and matrices also created
  - Assembly process for local domains writes to *global* residual vector and Jacobian matrix
    - No local renumbering, essentially no change to assembly routines required
  - Copy data from global to local matrix/residual before local linear solve



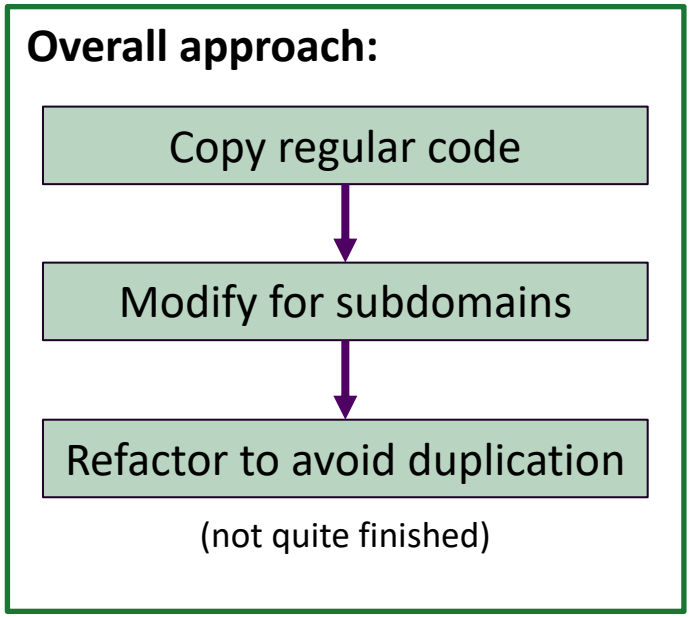
Rows/cols of a domain in global Jacobi matrix…

…extracted to local Jacobi matrix

What did we have to do?

New or changed components:

- A subgrid view class
  - Could have done without but this allowed more compatibility
- Code for creating domain subpartitions
  - Defaults to using the Zoltan partitioner (same as MPI domain partitioning)
- Ordering of domains for Gauss-Seidel variant
- Local Newton and solver logic, updates
- Calling linear solver for single domains
  - Had to add explicit ”run serially” option to avoid unintended communication
- Local domain convergence checking
- Error handling, dealing with failed local domain solves
- Domain-aware well handling (avoid acting on wells not in your domain)

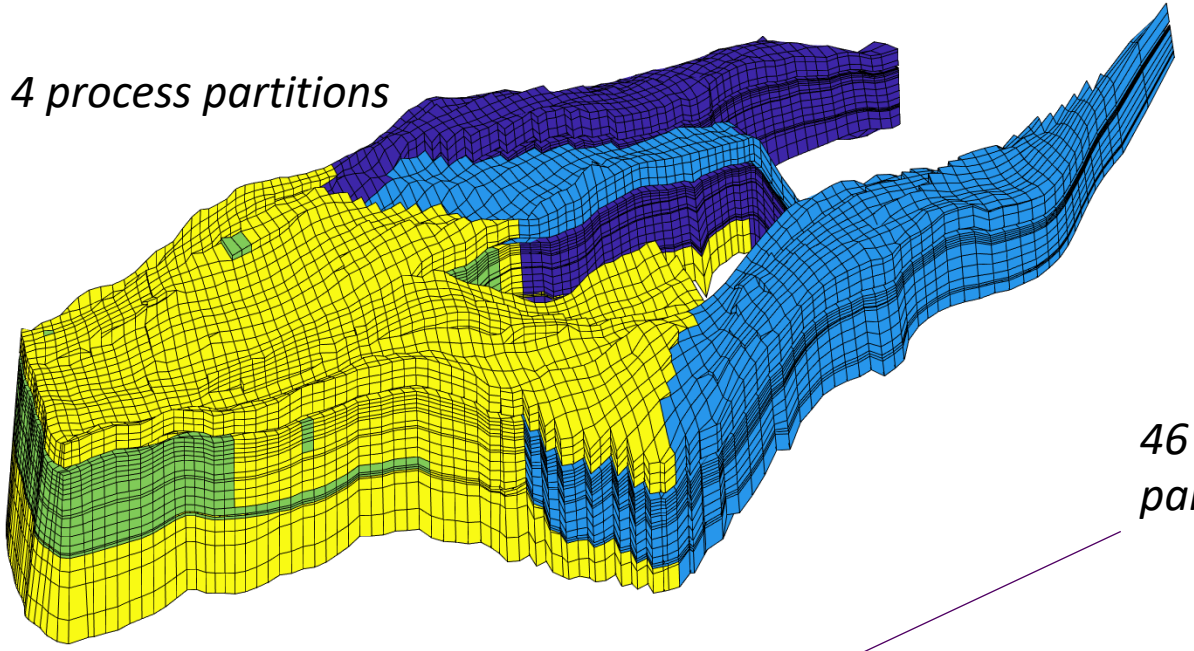



```
ConvergenceReport getConvergence(…)  ⟶  ConvergenceReport getDomainConvergence(const Domain& domain, …)
```

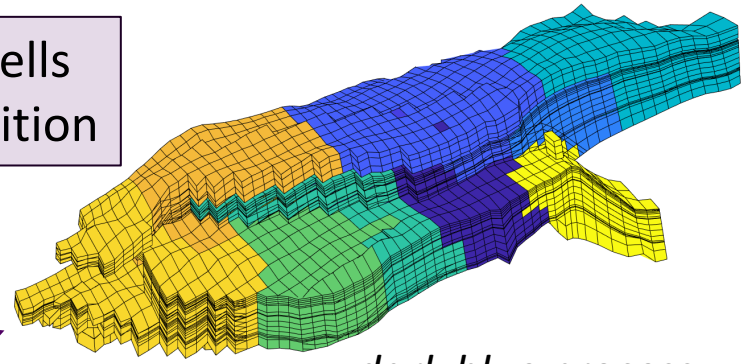
Partitioning and parallel approach

- Wells are kept within a single NLDD domain
  - Simplifies treatment
  - Probably a good idea in any case?
  - Domains with strange shapes, or disjoint: *not a big problem* for NLDD
    - Unlike MPI partitioning, causes no extra communication
- MPI parallel approach
  - Each MPI rank/process has multiple NLDD subdomains
  - Each NLDD subdomain exists only on a single rank
  - Local solves default to Gauss-Seidel/multiplicative approach within each rank
  - Communicate boundaries after all ranks complete local solve
  - Independent solves on each rank => Jacobi/additive approach between ranks

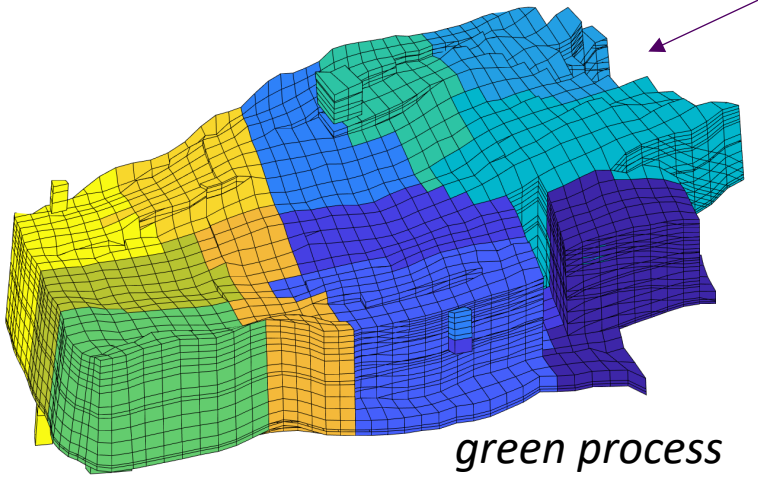
Example MPI and subdomain partitions

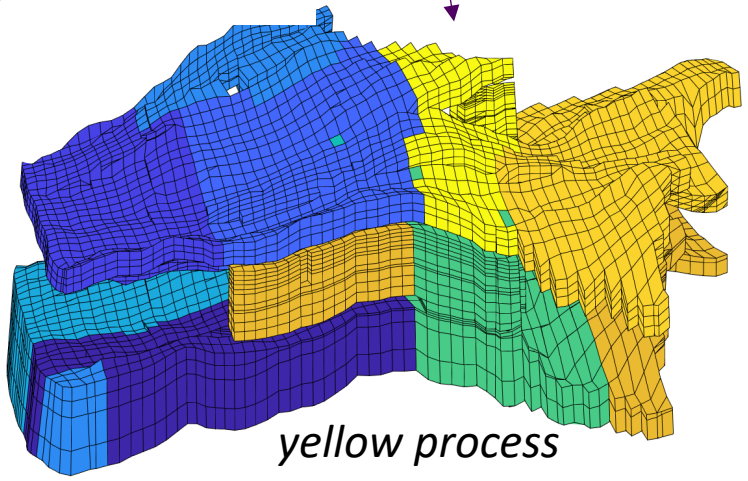


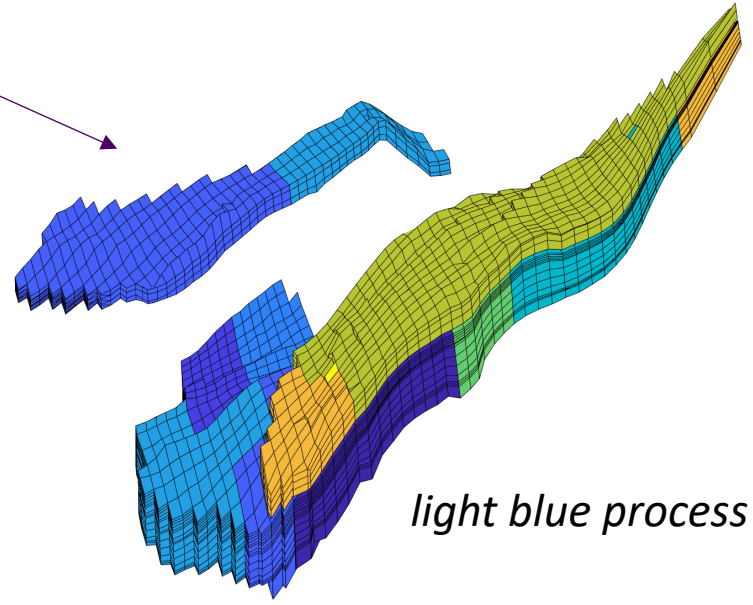
Default: ca. 1000 cells per subdomain partition



46 subdomain partitions (total)



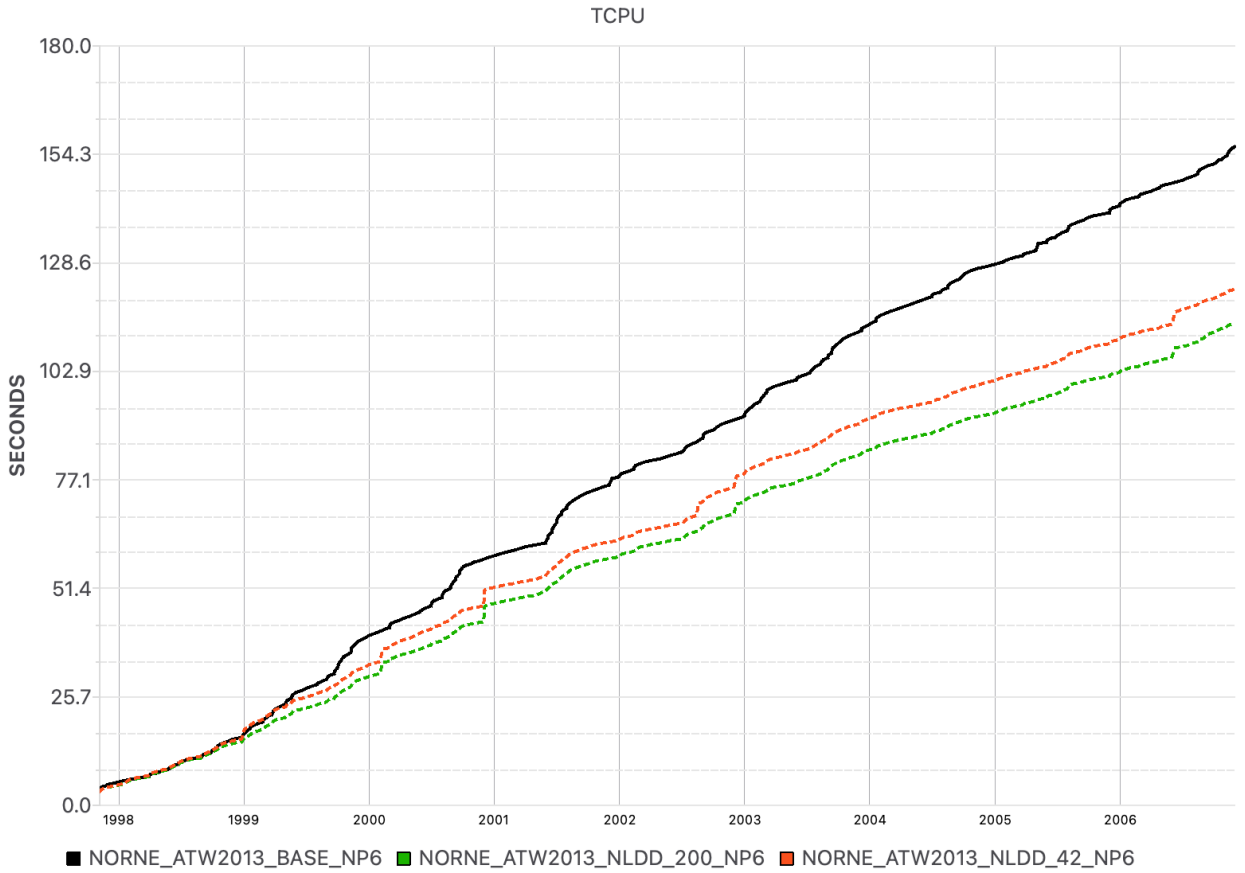




Results: Norne

- Norne open field case
  - 45k cells, 3 phase black oil
- Run with 6 MPI processes

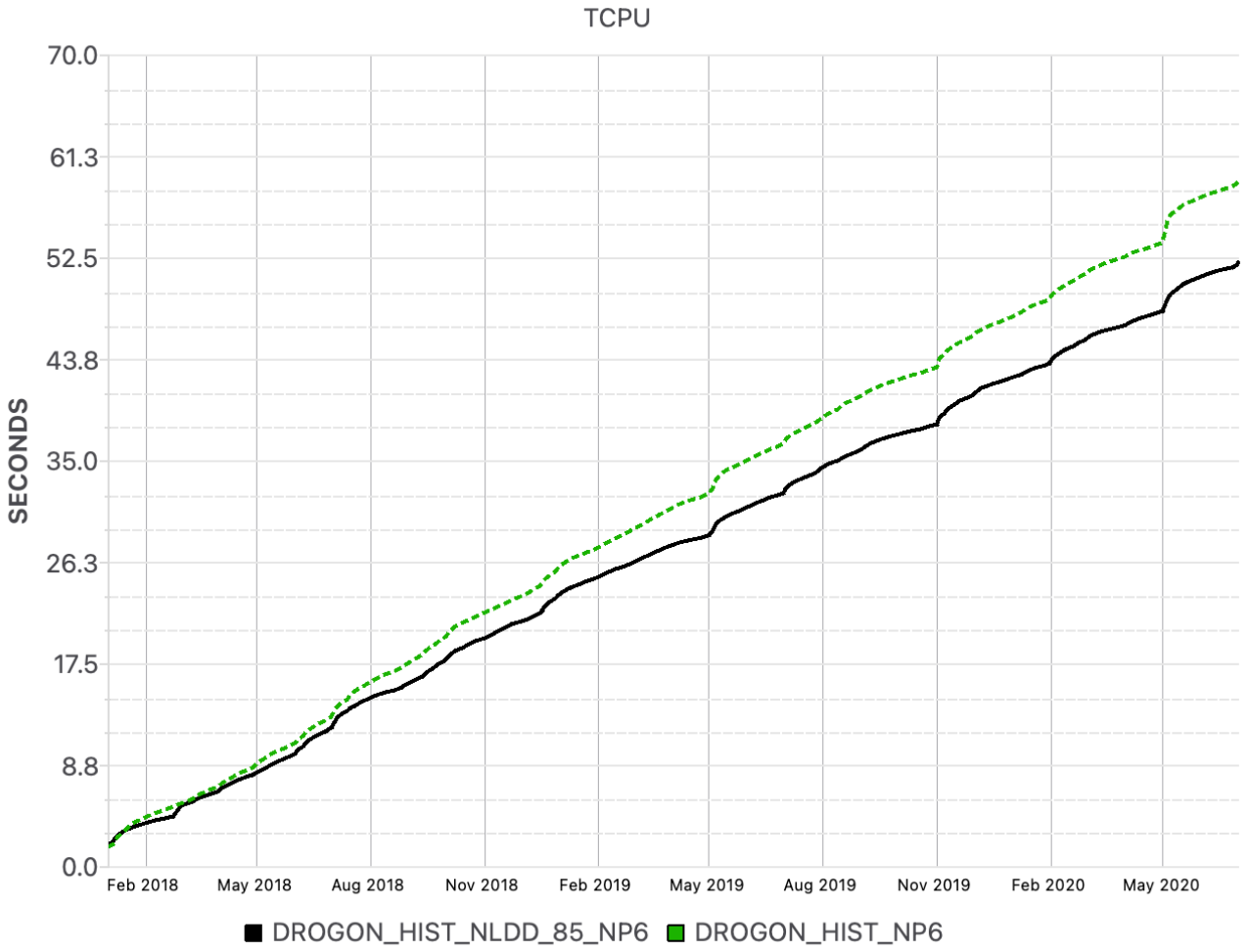
Method	Newton	NLDD, 42 domains	NLDD, 200 domains
Total runtime (s)	156.1	122.3	114.2



Results: Drogon

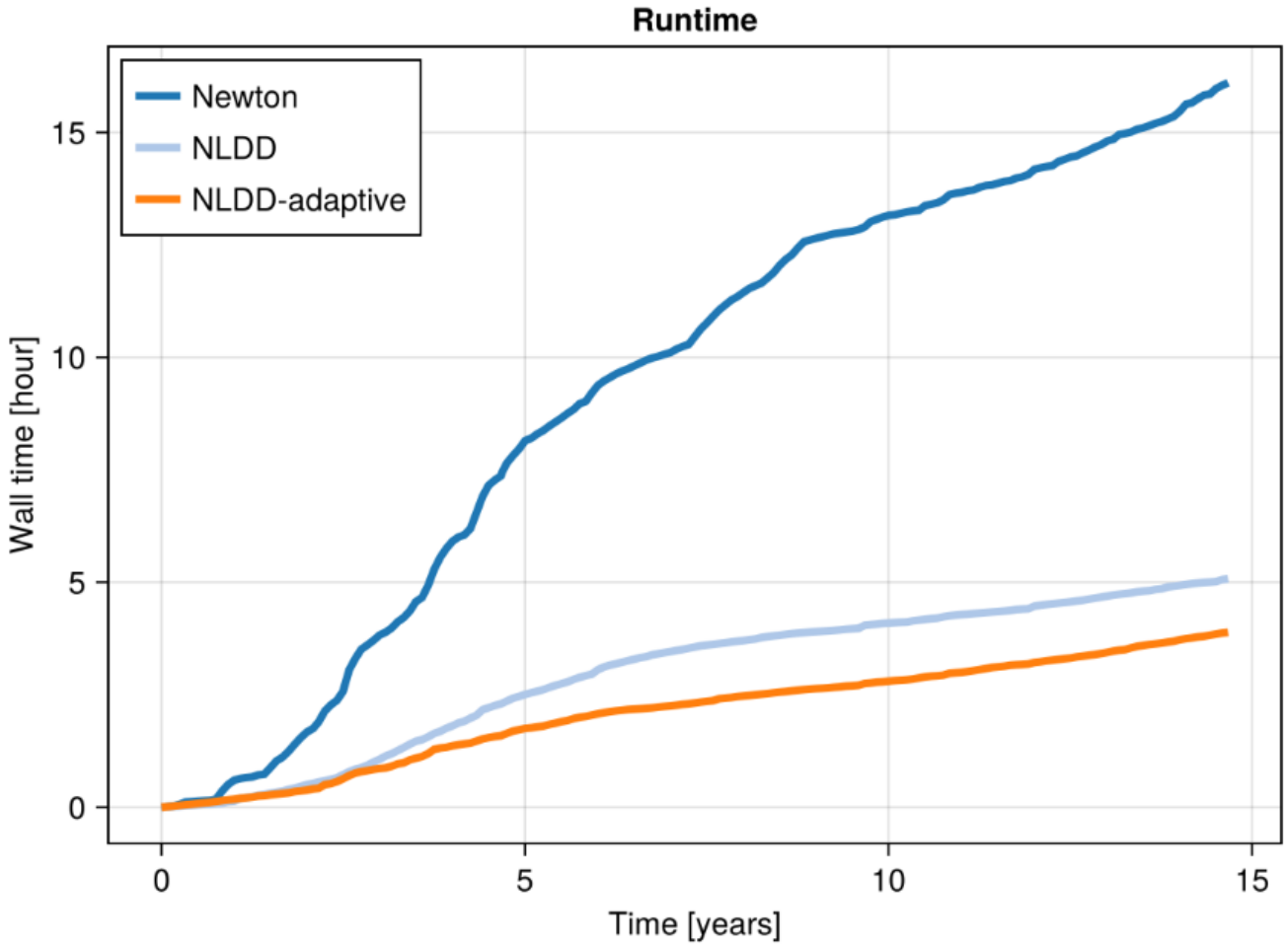
- Open artificial field case
  - 70k cells, 3 phase black oil
- Run with 6 MPI processes

Method	Newton	NLDD
Total runtime (s)	59.1	52.1



Results: Proprietary model

- Results on some proprietary models shows great potential
- Not (yet) universally across models or implementations (Jutul and OPM)
- Result on right from Jutul



Ongoing and future work

- Adaptivity
  - Smarter switching between methods (Newton, NLDD)
  - Skipping local solves in subdomains with small changes
  - Adapting tolerances locally to situation
- Robustness
  - Better handling of subdomain local solve convergence failures
  - Detecting problematic situations
    - Global (Newton) iterations and local solves push in opposite directions
    - Repeated failures to converge for single or few subdomains
  - Relation to timestepping, avoiding time step cuts
  - Improving default tuning parameters, providing good heuristics for choosing
- Performance
  - Avoid expensive checks and unnecessary linearizations
  - Improved property evaluation and assembly helps NLDD
    - NLDD reduces work done in global linear solves, and increases work in assembly