is paper describes objective technical results and analysis. Any subjective views or opinions that might be expres
the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Governm

SAND2015-2819C

Partial Differential Equations Solver Resilient to Simulated Soft and Hard Faults

F.Rizzi[†], K.Morris[†], K.Sargsyan[†], C.Safta[†], P.Mycek[‡], O.LeMaitre[‡], H.Najm[†] O.Knio[‡], B.Debusschere[†]

[†]Sandia National Laboratories, Livermore, CA
[‡]Duke University, Durham, NC

SIAM-CSE
– March 2015 –

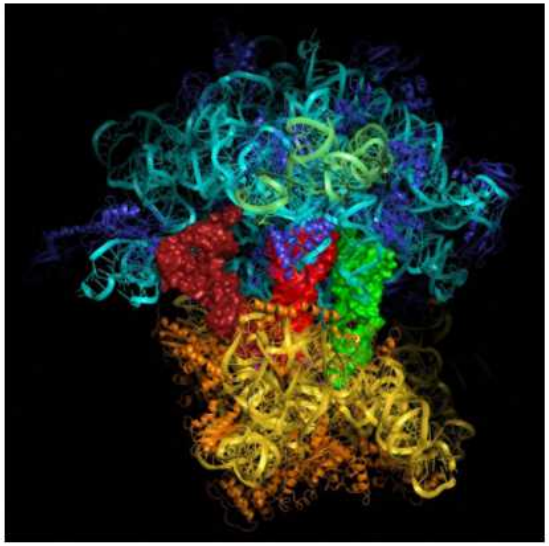
Supported by the US Department of Energy (DOE)
Advanced Scientific Computing Research (ASCR)

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

ories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of San
well International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under cont

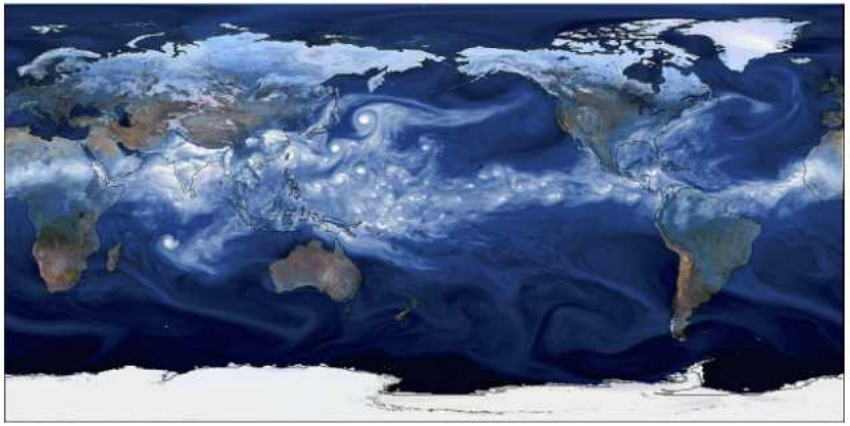
Exascale: Paradigm For Future Scientific Problems

- Open doors to new discoveries.
- Aerospace engineering, energy, astrophysics, biology, climate modeling, national security, etc.
- Challenges?



Molecular model of a bacterial ribosome. (MRC Lab of Molecular Biology, web)

Blue-brain project (web).



Global-scale simulation of storms (LBNL, web).

Challenges

1. New architectures capable of combining tens of thousands of CPUs and/or accelerators.
2. Growing energy costs [a]
   - Currently: 1/2 megawatts per petaflop/s.
   - Projection: multiply by 1,000 to get to exascale, not affordable.
   - Target: 50 MW for exascale by 2020.
3. Software developers will have to learn how to build programs that can make use of the new architecture.

[a] http://spectrum.ieee.org/computing/hardware/nextgeneration-supercomputers

- Concurrency/parallelism: nodes/cores/threads.
- Cost: data movement dominates.
- Locality: keep data local, reduce communication.
- **Resiliency.**

Faults: Soft and Hard Errors

- Exascale will push hardware to the limits.
- Mean time between failures (MTBF)
  - Currently: order of days/weeks [a]
  - Exascale: Many more components $\Rightarrow$ MTBF $\sim$ hours/minutes?
- Hard faults: nodes dying/hanging, network failures.
- Soft (silent) faults: changes in system caused by external forces, e.g. bit-flips, masked errors. Difficult to detect.
  - Cosmic rays.
  - Variation in voltage, temperature.
- Conventional approaches may not be effective
  - Time to save or restart from checkpoint may exceed MTBF.
- Need to **evolve algorithms**, not just hardware.

[a] TOWARD EXASCALE RESILIENCE: 2014 UPDATE, F.Cappello et al., SUPERCOMPUTING FRONTIERS AND INNOVATIONS, 2014

Overview

What? Domain-decomposition-based solver for 2D partial differential equations (PDEs).

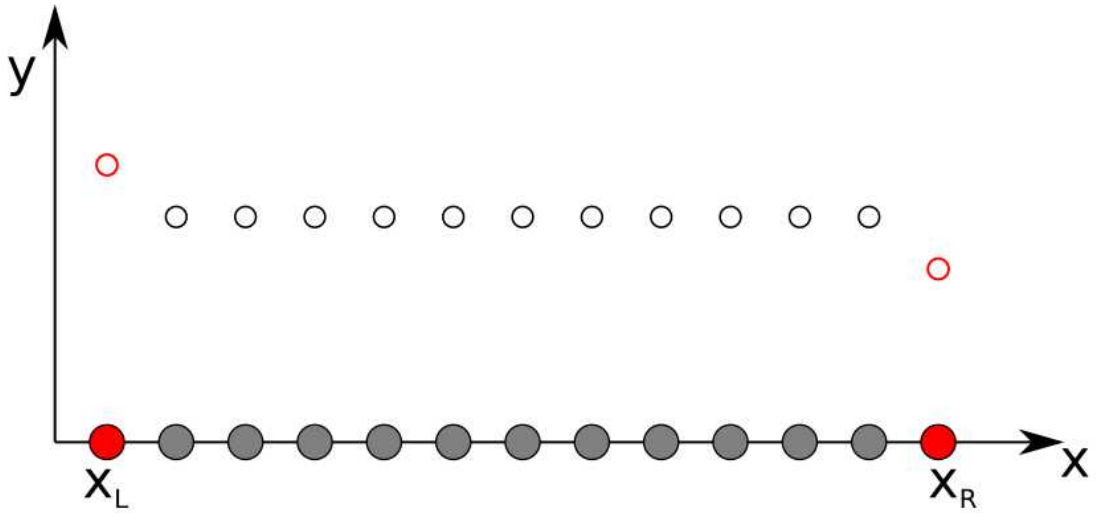
How? Recasting the original PDE problem as a sampling problem, followed by a resilient data manipulation to achieve the final solution update.

Why? Resilient to both soft and hard faults.

We do not characterize all types of system faults that can occur, but focus solely on the information that a simulation provides.

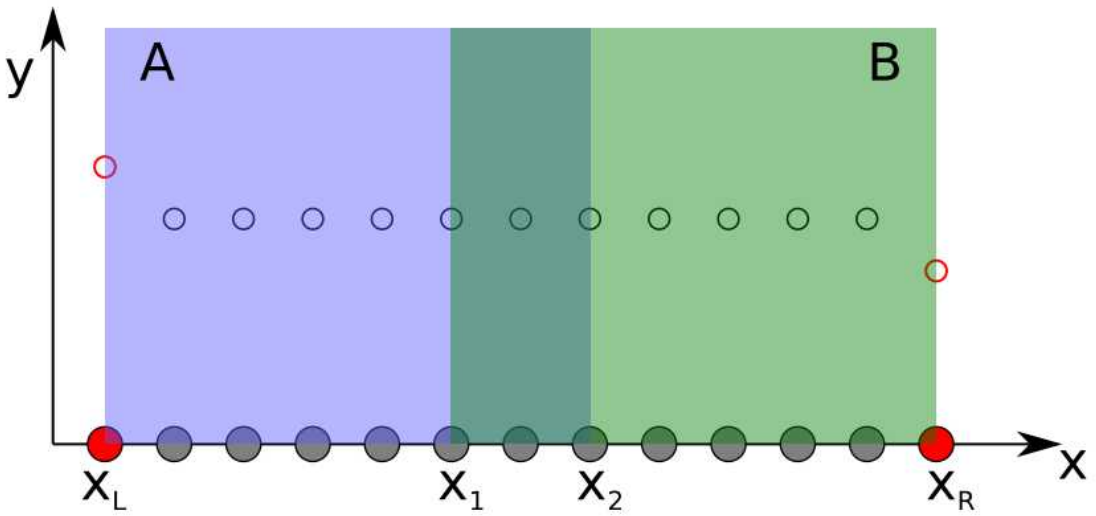
Algorithm Overview

- Grid with current state.
- Partition space with overlapping subdomains.
- Define sampling range for each boundary.
- Sample uniformly and solve PDE locally.
- Use samples to build boundary maps:
  $y_1 = f(y_2, y_L) = a + by_2 + cy_L$
  $y_2 = g(y_1, y_R) = d + ey_2 + fy_R$
- y_L, y_R are known BC.
- Fixed BC $\rightarrow$ solve for new state: (y_1^*, y_2^*).



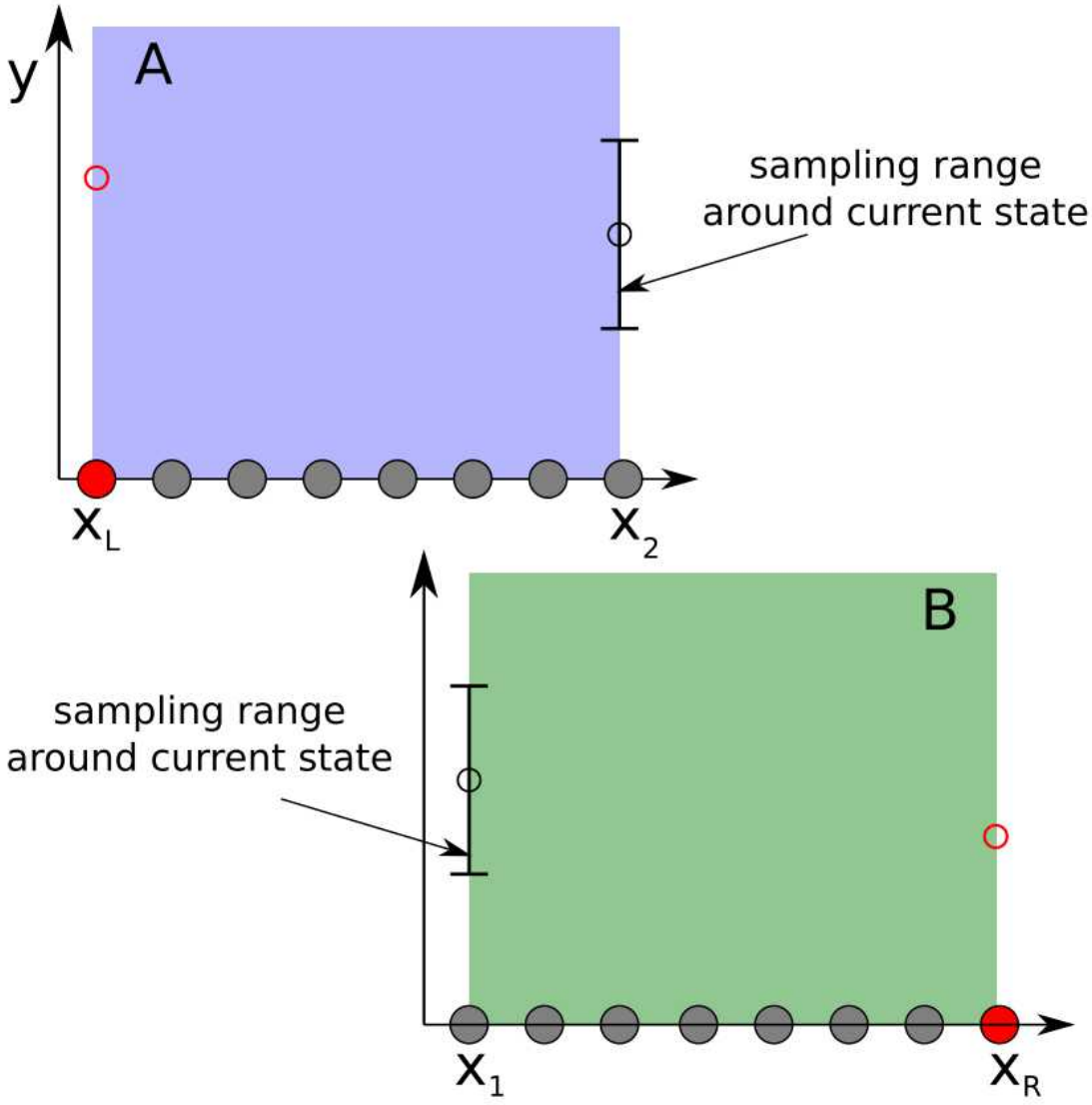
Algorithm Overview

- Grid with current state.
- Partition space with overlapping subdomains.
- Define sampling range for each boundary.
- Sample uniformly and solve PDE locally.
- Use samples to build boundary maps:
  $y_1 = f(y_2, y_L) = a + by_2 + cy_L$
  $y_2 = g(y_1, y_R) = d + ey_2 + fy_R$
- y_L, y_R are known BC.
- Fixed BC $\rightarrow$ solve for new state: (y_1^*, y_2^*).



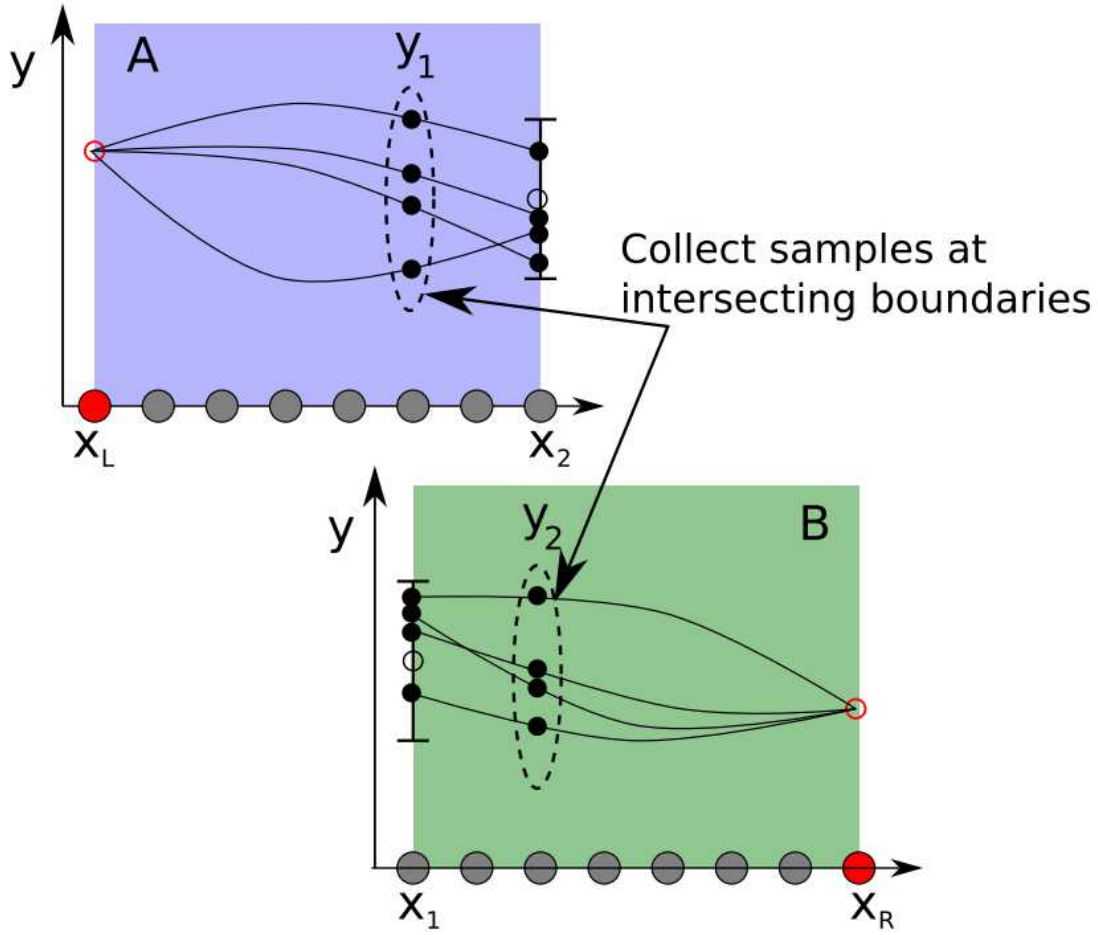
Algorithm Overview

- Grid with current state.
- Partition space with overlapping subdomains.
- Define sampling range for each boundary.
- Sample uniformly and solve PDE locally.
- Use samples to build boundary maps:
  $y_1 = f(y_2, y_L) = a + by_2 + cy_L$
  $y_2 = g(y_1, y_R) = d + ey_2 + fy_R$
- y_L, y_R are known BC.
- Fixed BC $\rightarrow$ solve for new state: (y_1^*, y_2^*).



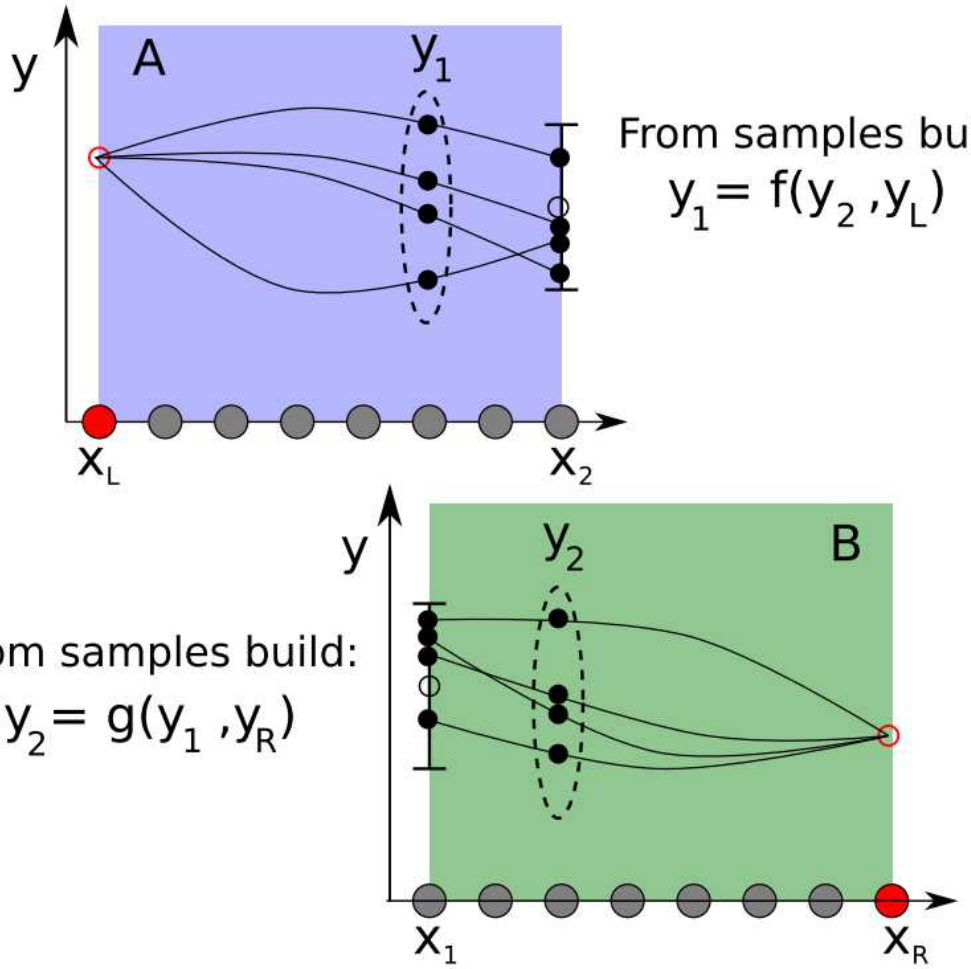
Algorithm Overview

- Grid with current state.
- Partition space with overlapping subdomains.
- Define sampling range for each boundary.
- Sample uniformly and solve PDE locally.
- Use samples to build boundary maps:
  $y_1 = f(y_2, y_L) = a + by_2 + cy_L$
  $y_2 = g(y_1, y_R) = d + ey_2 + fy_R$
- y_L, y_R are known BC.
- Fixed BC $\rightarrow$ solve for new state: (y_1^*, y_2^*).



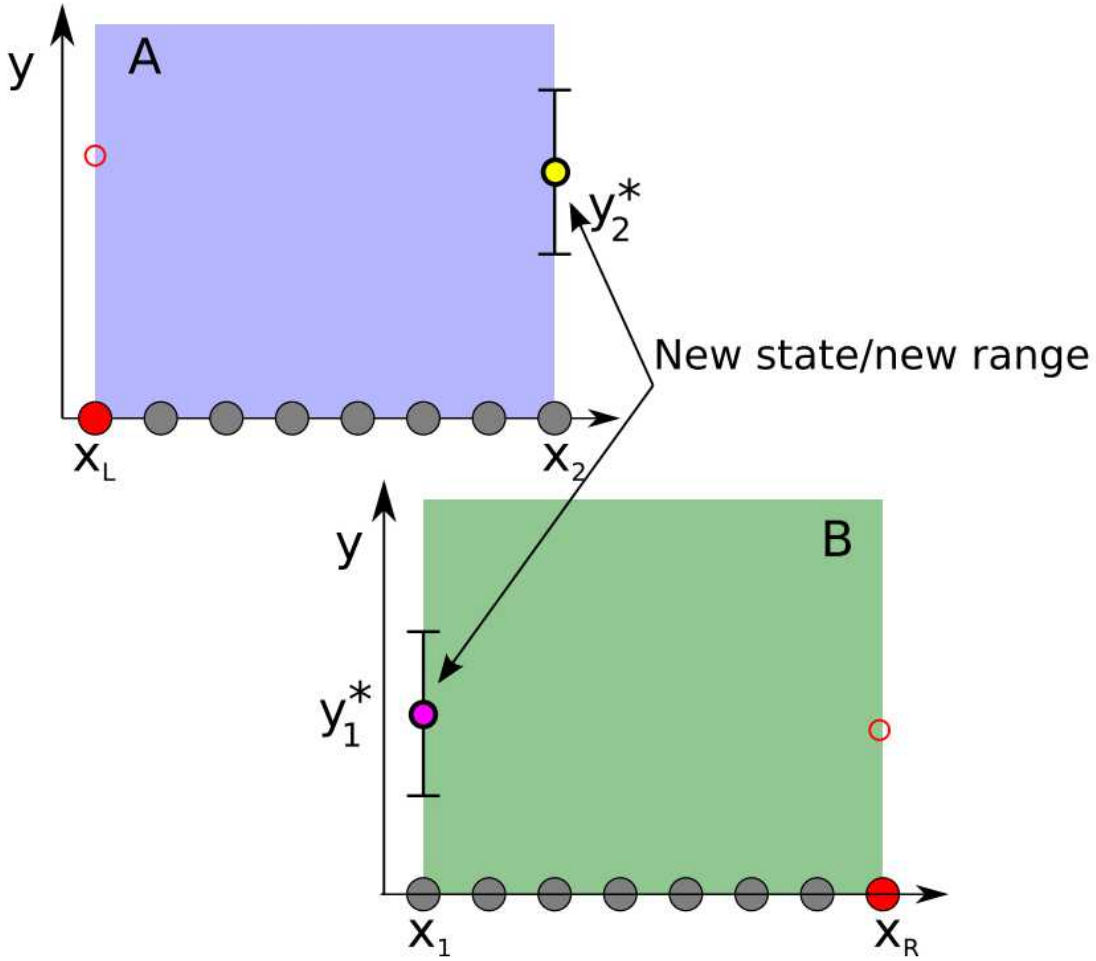
Algorithm Overview

- Grid with current state.
- Partition space with overlapping subdomains.
- Define sampling range for each boundary.
- Sample uniformly and solve PDE locally.
- Use samples to build boundary maps:
  $y_1 = f(y_2, y_L) = a + by_2 + cy_L$
  $y_2 = g(y_1, y_R) = d + ey_2 + fy_R$
- y_L, y_R are known BC.
- Fixed BC $\rightarrow$ solve for new state: (y_1^*, y_2^*).

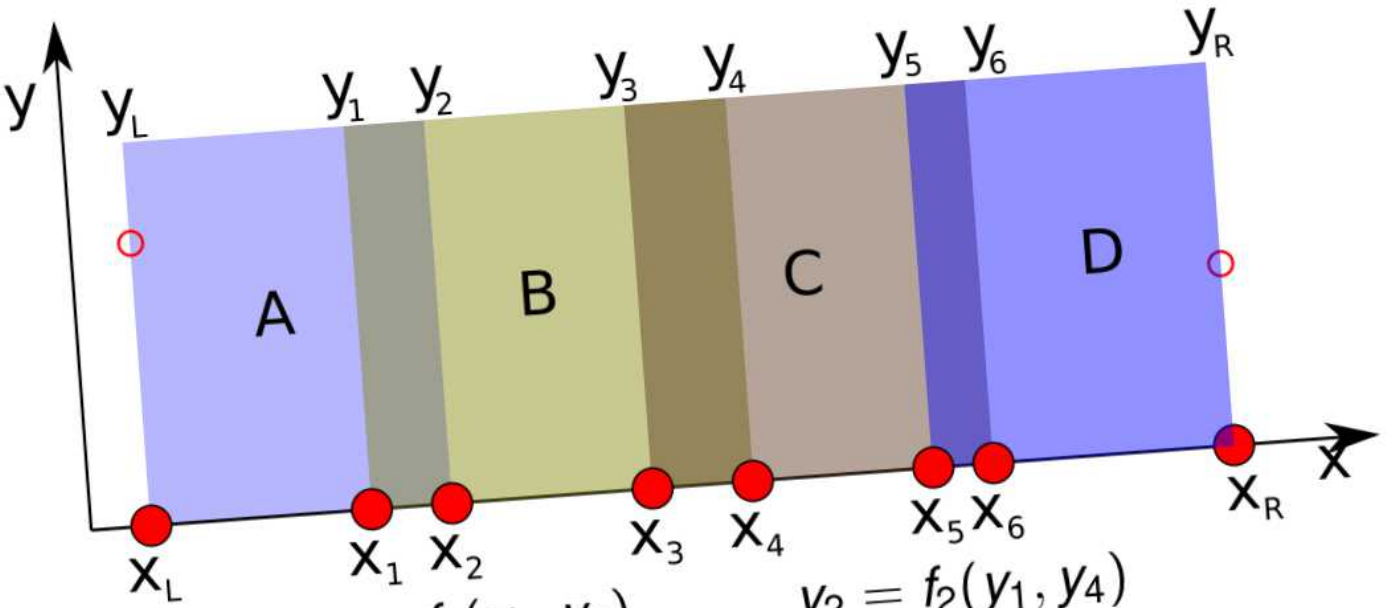


Algorithm Overview

- Grid with current state.
- Partition space with overlapping subdomains.
- Define sampling range for each boundary.
- Sample uniformly and solve PDE locally.
- Use samples to build boundary maps:
  $y_1 = f(y_2, y_L) = a + by_2 + cy_L$
  $y_2 = g(y_1, y_R) = d + ey_2 + fy_R$
- y_L, y_R are known BC.
- Fixed BC $\rightarrow$ solve for new state: (y_1^*, y_2^*).



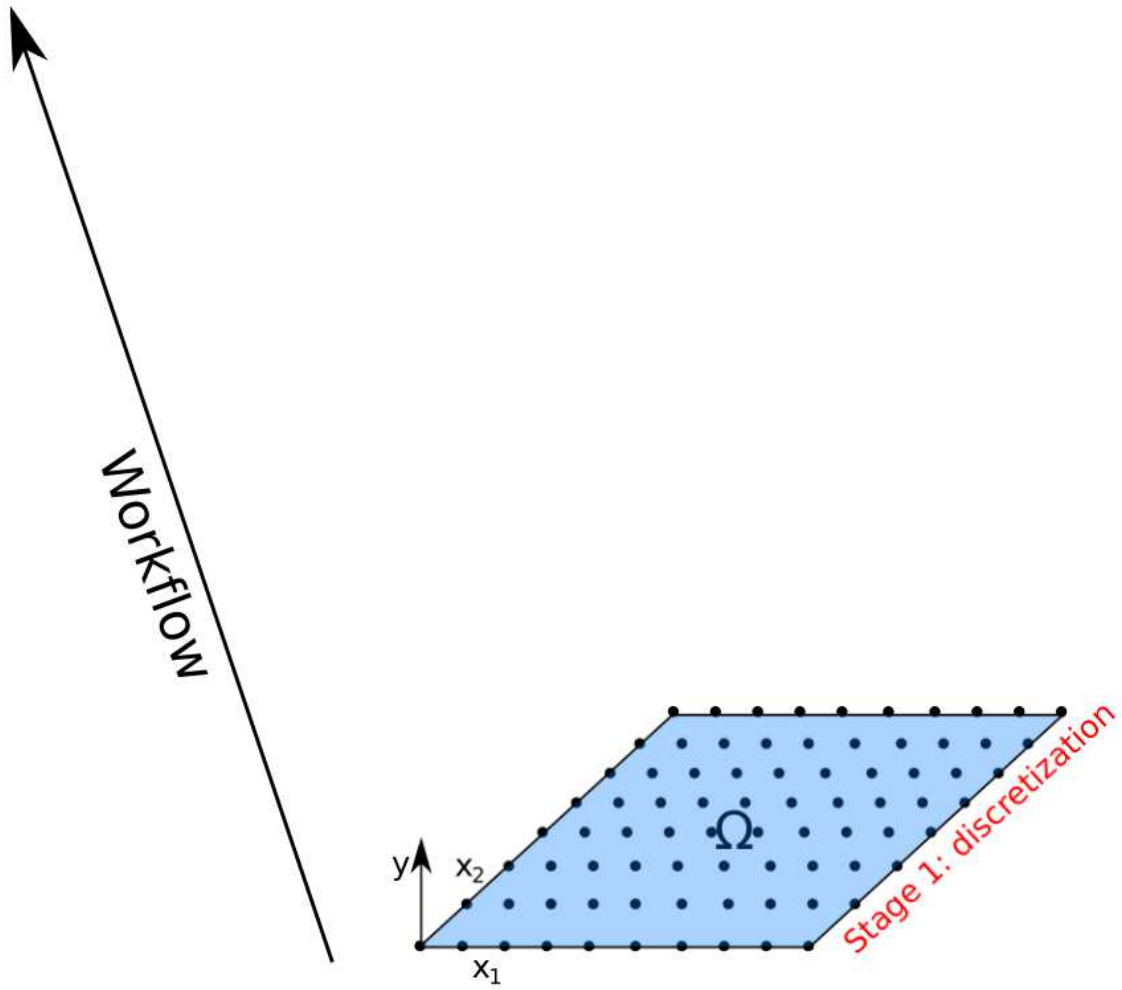
Extension



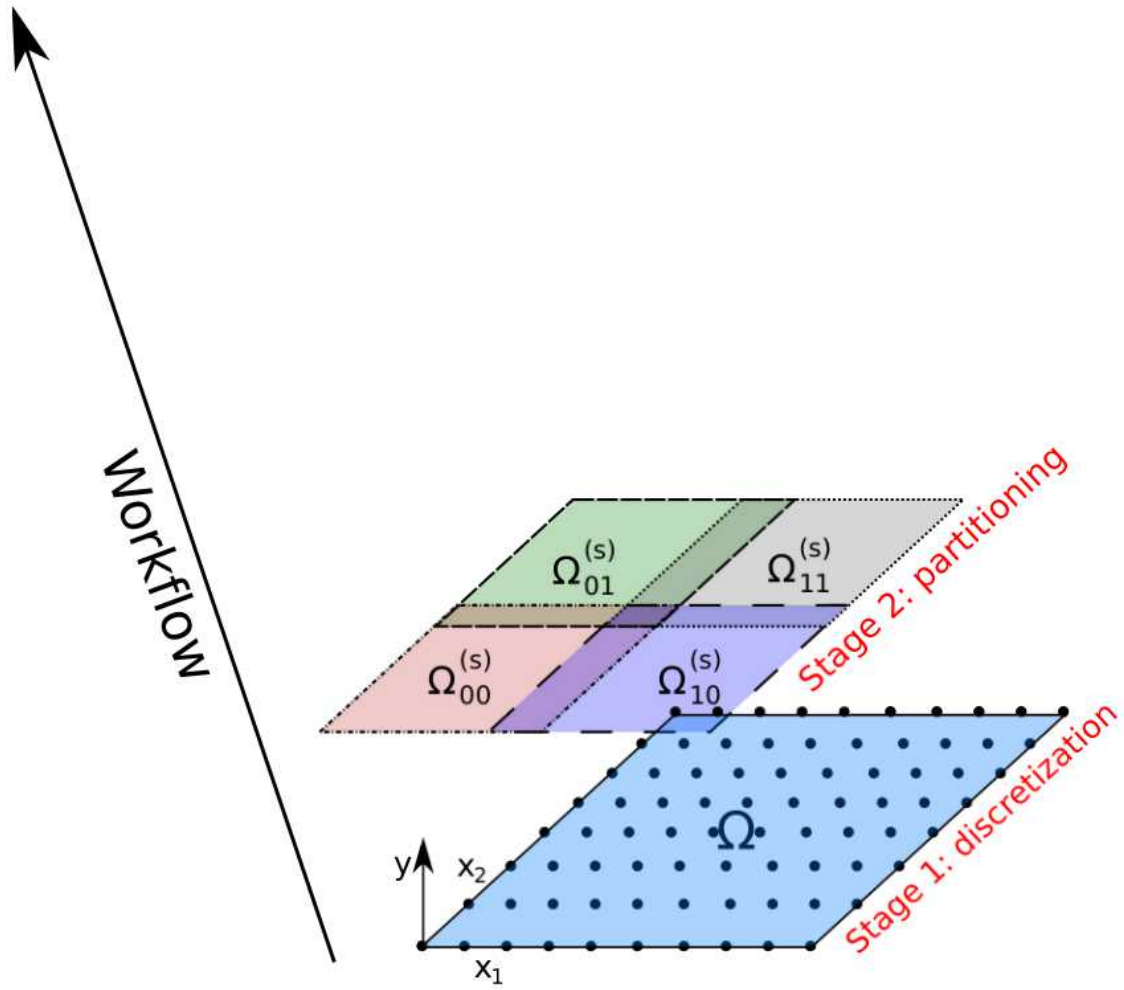
$$y_1 = f_1(y_L, y_2) \qquad y_2 = f_2(y_1, y_4)$$
$$y_3 = f_3(y_1, y_4) \qquad y_4 = f_4(y_3, y_6)$$
$$y_5 = f_5(y_3, y_6) \qquad y_6 = f_6(y_5, y_R)$$

Same procedure but inner boundary maps are 2D

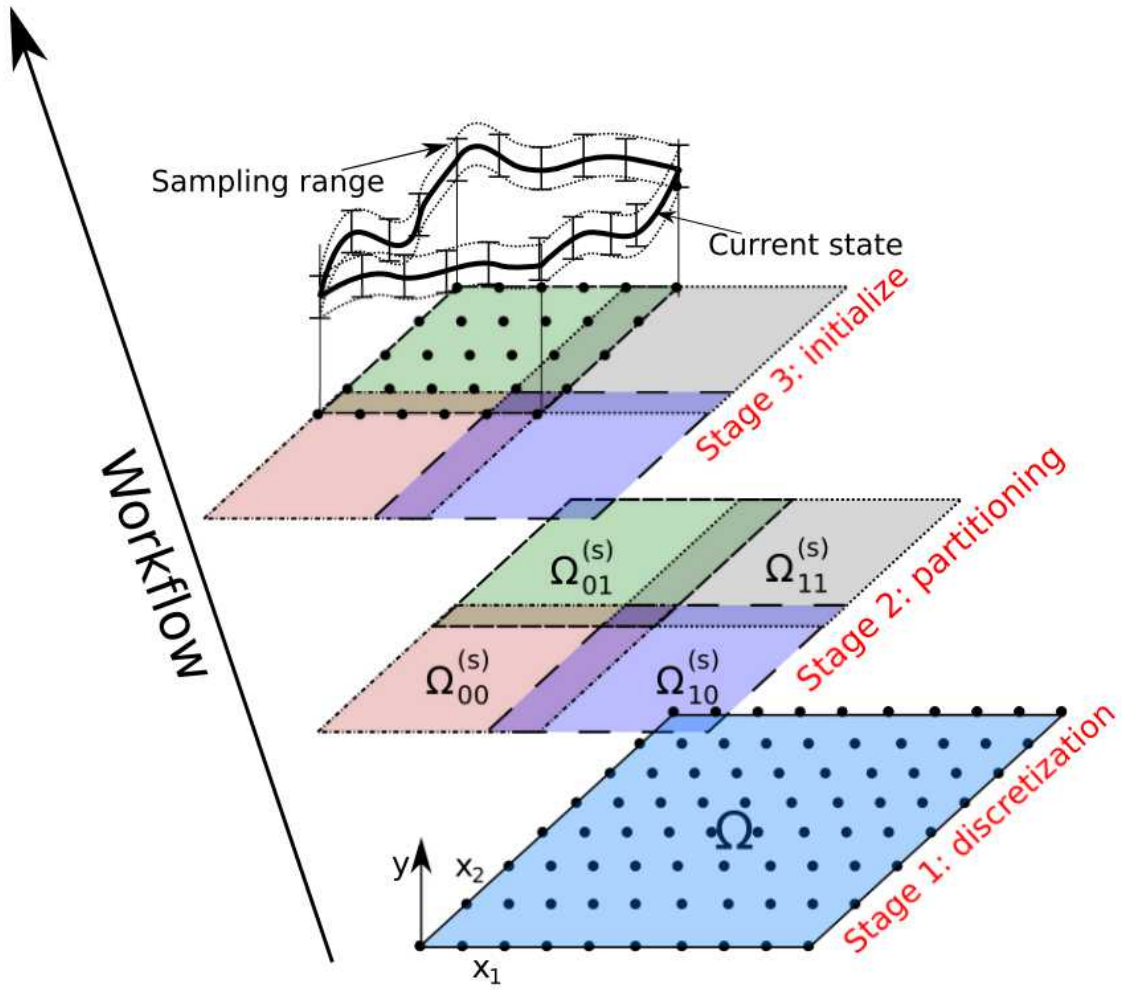
Extension



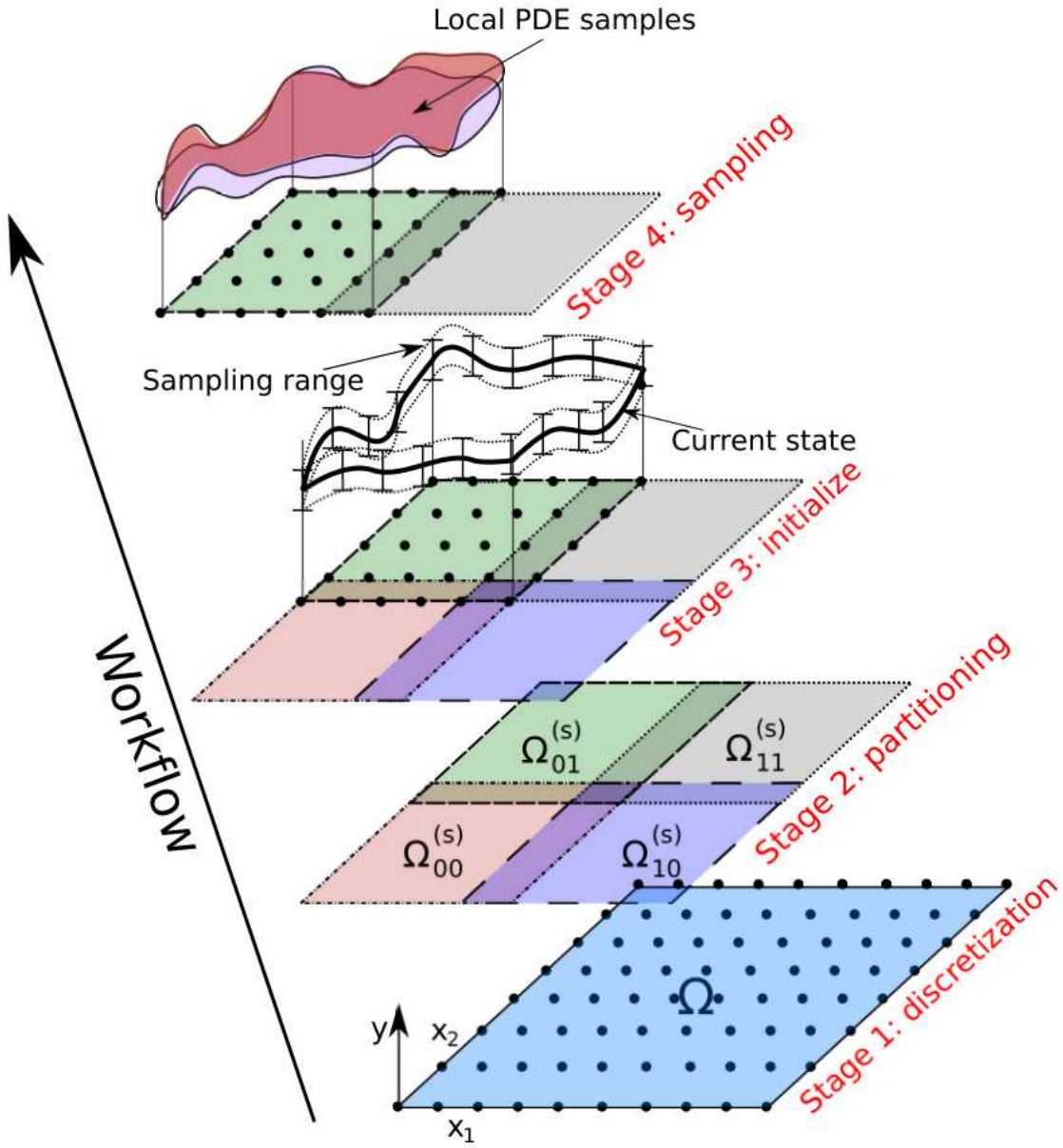
Extension


Workflow
$\Omega_{01}^{(s)}$
$\Omega_{11}^{(s)}$
$\Omega_{00}^{(s)}$
$\Omega_{10}^{(s)}$
Stage 2: partitioning
Ω
y
x_2
x_1
Stage 1: discretization

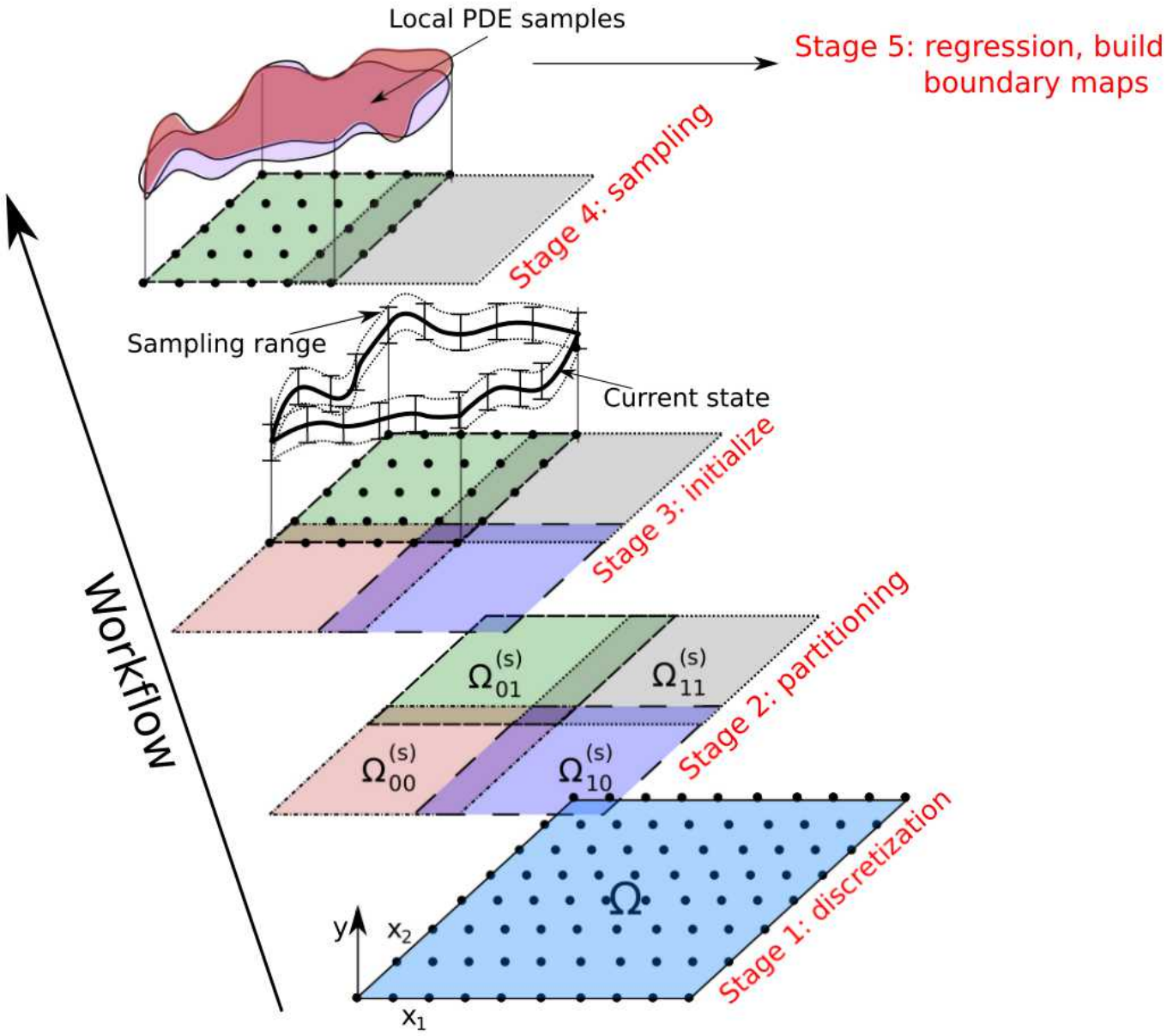
Extension


Workflow
Sampling range
Current state
Stage 3: initialize
Stage 2: partitioning
Stage 1: discretization
$\Omega_{01}^{(s)}$
$\Omega_{11}^{(s)}$
$\Omega_{00}^{(s)}$
$\Omega_{10}^{(s)}$
Ω
y
x_2
x_1

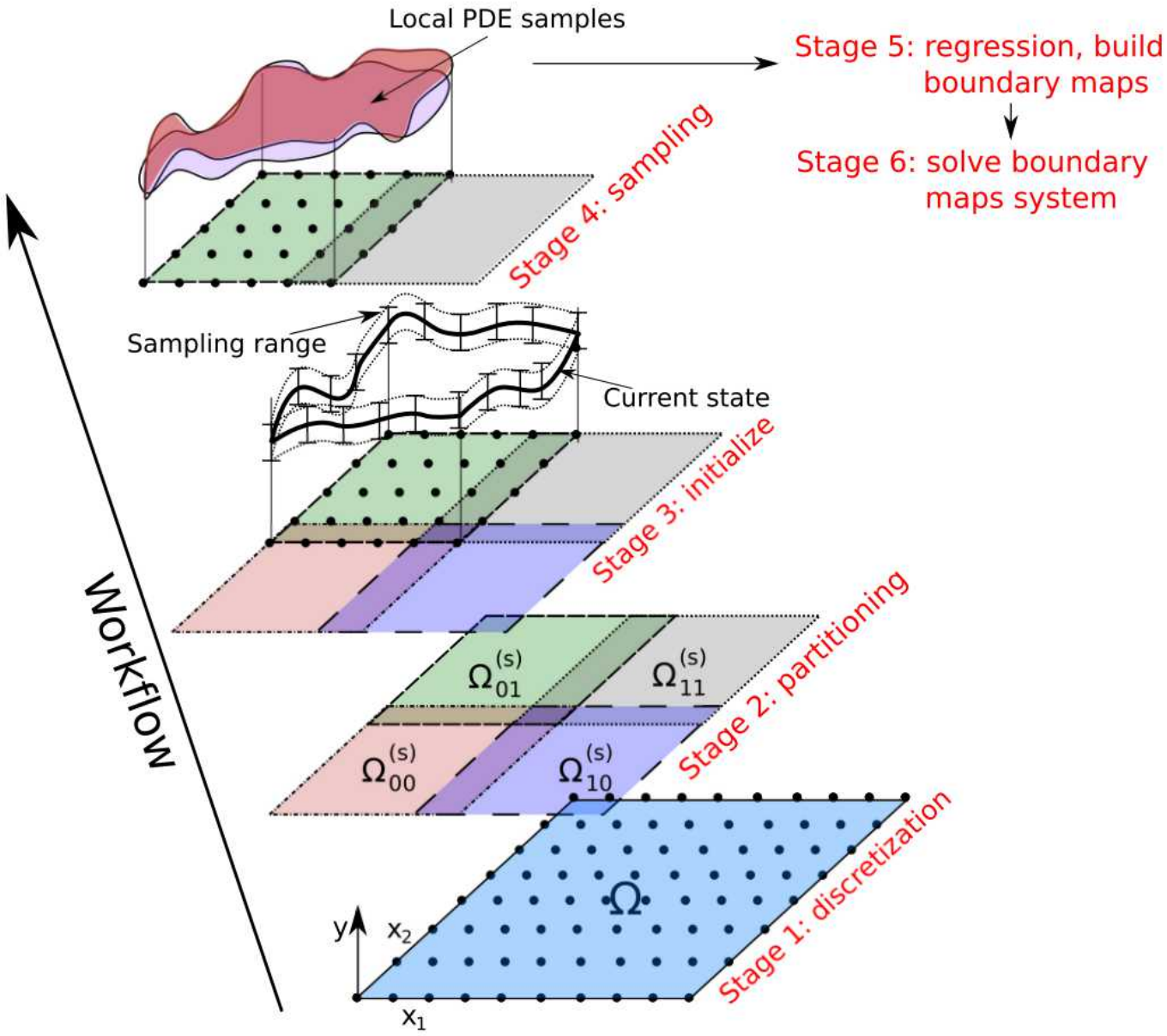
Extension


Local PDE samples
Stage 4: sampling
Sampling range
Current state
Stage 3: initialize
Workflow
$\Omega_{01}^{(s)}$
$\Omega_{11}^{(s)}$
$\Omega_{00}^{(s)}$
$\Omega_{10}^{(s)}$
Stage 2: partitioning
Ω
y
x_2
x_1
Stage 1: discretization

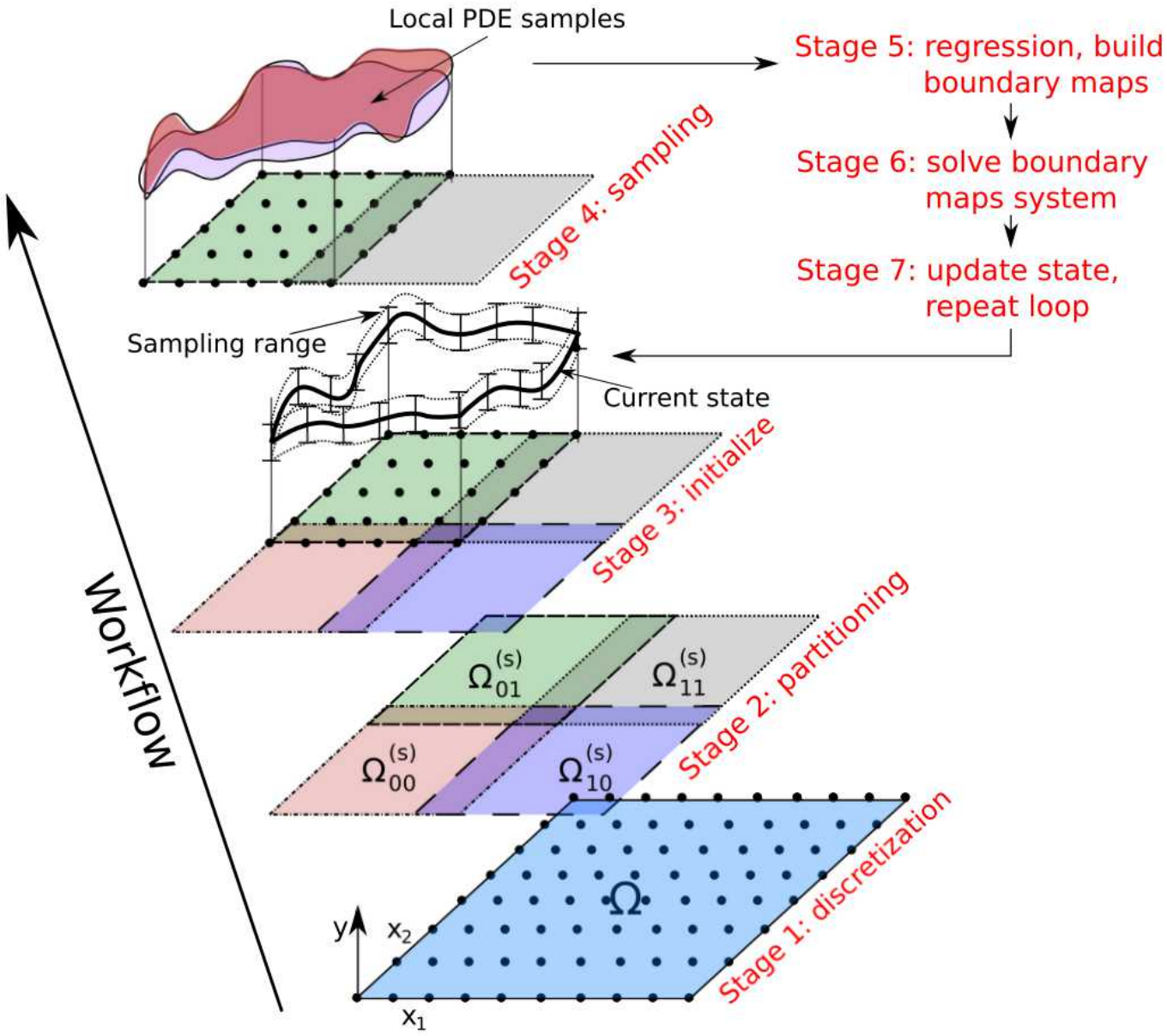
Extension


Local PDE samples
Stage 5: regression, build boundary maps
Stage 4: sampling
Sampling range
Current state
Stage 3: initialize
Workflow
$\Omega_{01}^{(s)}$
$\Omega_{11}^{(s)}$
$\Omega_{00}^{(s)}$
$\Omega_{10}^{(s)}$
Stage 2: partitioning
Ω
y
x_2
x_1
Stage 1: discretization

Extension

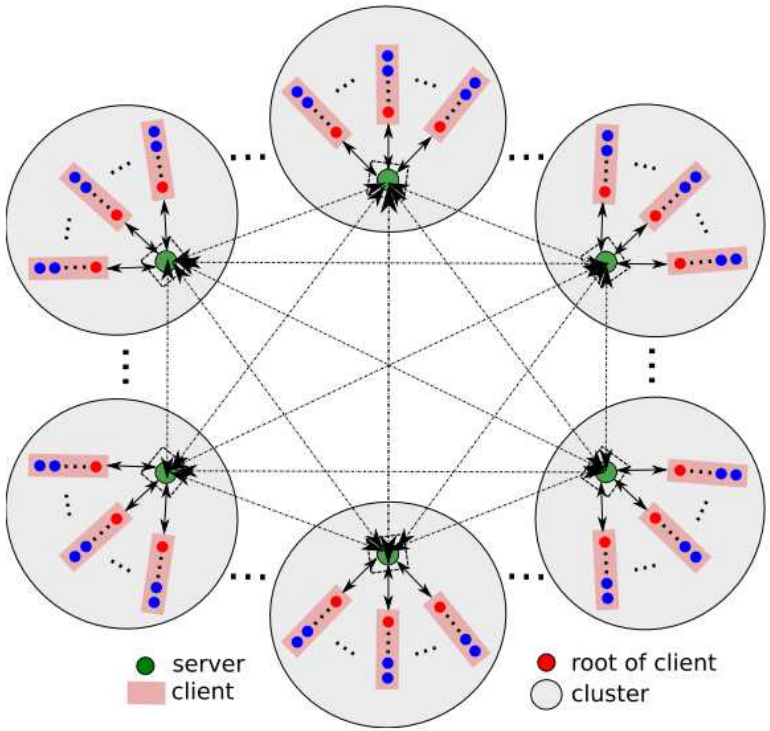

Local PDE samples
Stage 5: regression, build boundary maps
Stage 6: solve boundary maps system
Stage 4: sampling
Sampling range
Current state
Stage 3: initialize
Workflow
$\Omega_{01}^{(s)}$
$\Omega_{11}^{(s)}$
$\Omega_{00}^{(s)}$
$\Omega_{10}^{(s)}$
Stage 2: partitioning
Ω
y
x_2
x_1
Stage 1: discretization

Extension


Local PDE samples
Stage 5: regression, build boundary maps
Stage 6: solve boundary maps system
Stage 7: update state, repeat loop
Stage 4: sampling
Sampling range
Current state
Stage 3: initialize
Workflow
$\Omega_{01}^{(s)}$
$\Omega_{11}^{(s)}$
$\Omega_{00}^{(s)}$
$\Omega_{10}^{(s)}$
Stage 2: partitioning
Ω
y
x_2
x_1
Stage 1: discretization

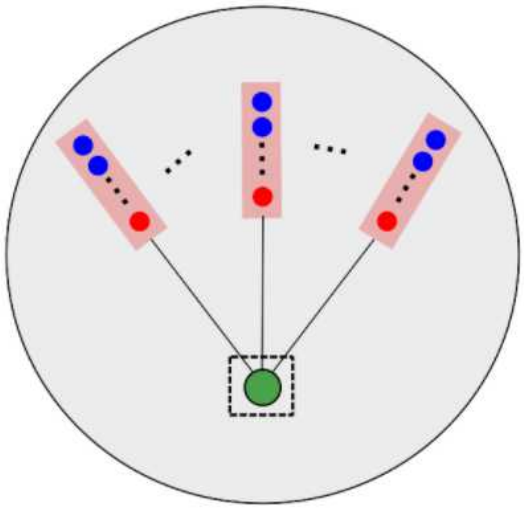
Implementation, Faults and Resilience

- MPI-based parallel code in C++.
- "Task-manager" framework:
  - Separate data and computation.
  - Data stored on "sandboxed" servers.
  - Operations as tasks that clients execute.
- Subdomains (state) live on servers. Clients are used for computations.
- Resilient to hard faults (clients crashing): hard faults $\rightarrow$ missing information.
- Need capability within algorithm to deal with missing data...



Implementation, Faults and Resilience

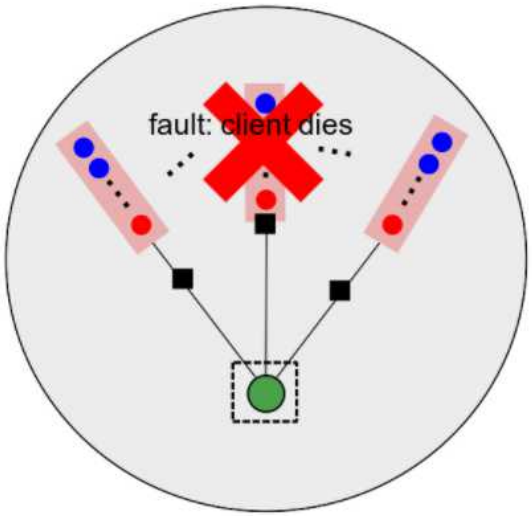
- MPI-based parallel code in C++.
- "Task-manager" framework:
  - Separate data and computation.
  - Data stored on "sandboxed" servers.
  - Operations as tasks that clients execute.
- Subdomains (state) live on servers. Clients are used for computations.
- Resilient to hard faults (clients crashing): hard faults $\rightarrow$ missing information.
- Need capability within algorithm to deal with missing data...



Implementation, Faults and Resilience

- Hard fault:

what? Client dies.
how? Set idle.
result? Lost task.



- Fault occurrence as a Poisson process: $F(t) = 1 - \exp^{-rt}$.
- Six different failure rates, r, explored.

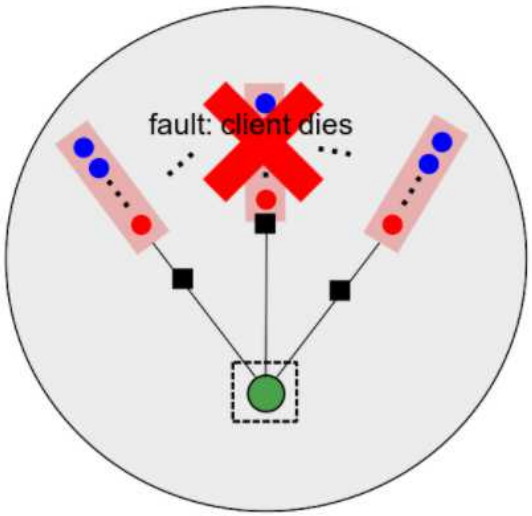
Implementation, Faults and Resilience

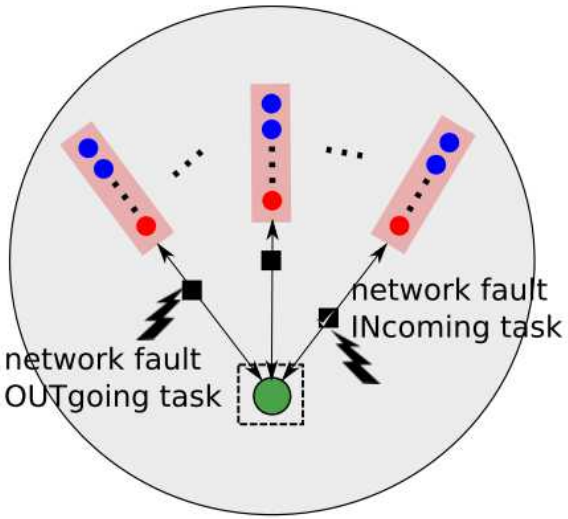
- Hard fault:

what? Client dies.
how? Set idle.
result? Lost task.

- Network fault:

what? Communication.
how? Bit-flip task's data.
result? Corrupted data.





- Fault occurrence as a Poisson process: $F(t) = 1 - \exp^{-rt}$.
- Six different failure rates, r, explored.

Implementation, Faults and Resilience

- Hard fault:

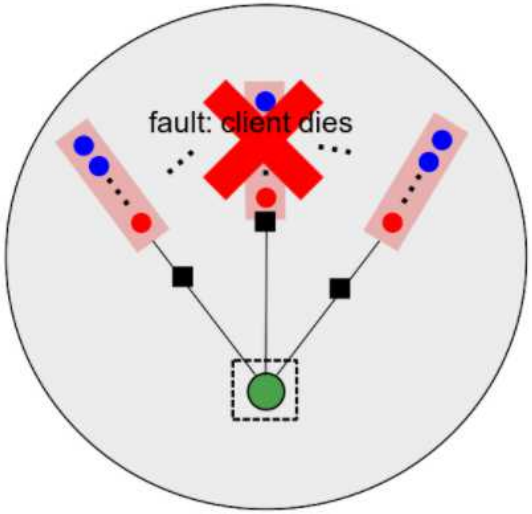
what? Client dies.
how? Set idle.
result? Lost task.

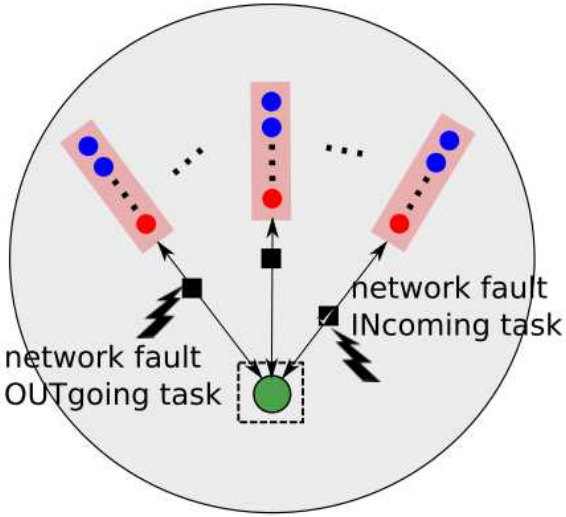
- Network fault:

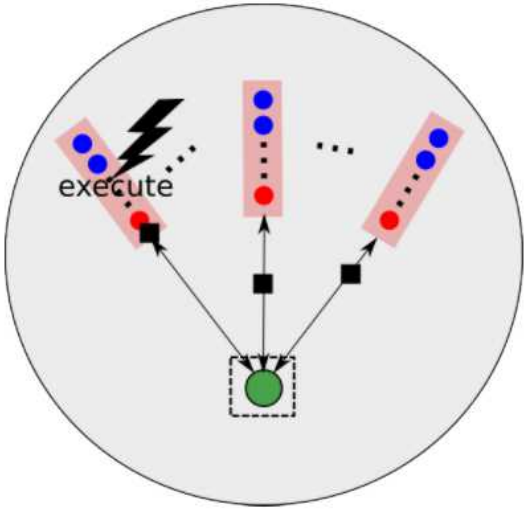
what? Communication.
how? Bit-flip task's data.
result? Corrupted data.

- Execution fault:

what? Computation.
how? Bit-flip task's data.
result? Corrupted data.





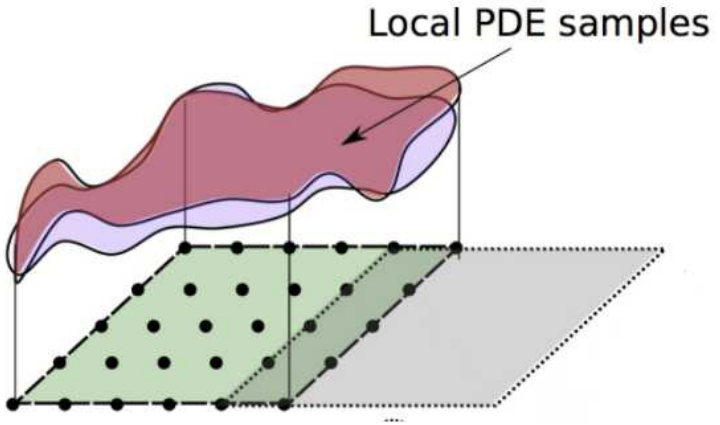


- Fault occurrence as a Poisson process: $F(t) = 1 - \exp^{-rt}$.
- Six different failure rates, r, explored.

Core Tasks and Soft Faults

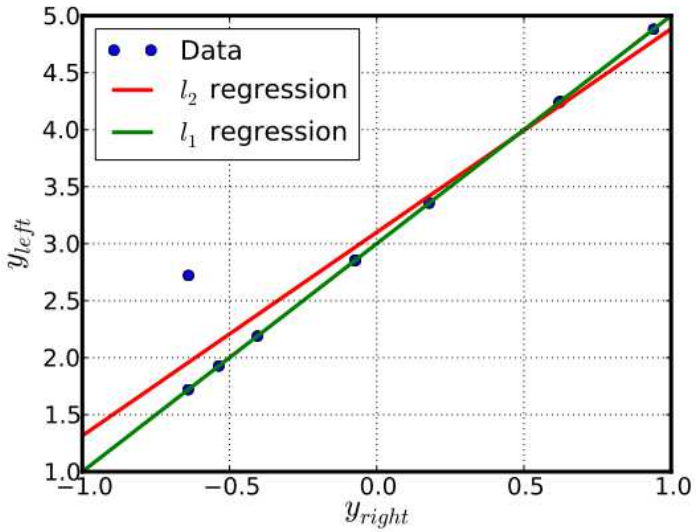
Sampling

- PDE solve in subdomain given BC.
- Keep generating samples until we have sufficient data set.
- *N* samples to guard against faults.



Regression

- Build boundary maps from PDE samples.
- Faults yield corrupted samples.
- ℓ_1 noise model (Laplace) as filter.
- Analogy with compressed sensing: find solution with as few non-zero residuals as possible



Test Case: 2D steady-diffusion equation

- Elliptic PDE:

$$\nabla \cdot \big(k(\boldsymbol{x})\nabla y(\boldsymbol{x})\big) = g(\boldsymbol{x}), \quad \textit{for} \;\; \boldsymbol{x} \in \Omega = (0,1)^2$$

- Homogeneous Dirichlet BC: $y|_{\partial\Omega} = 0$.
- Diffusivity and source term:

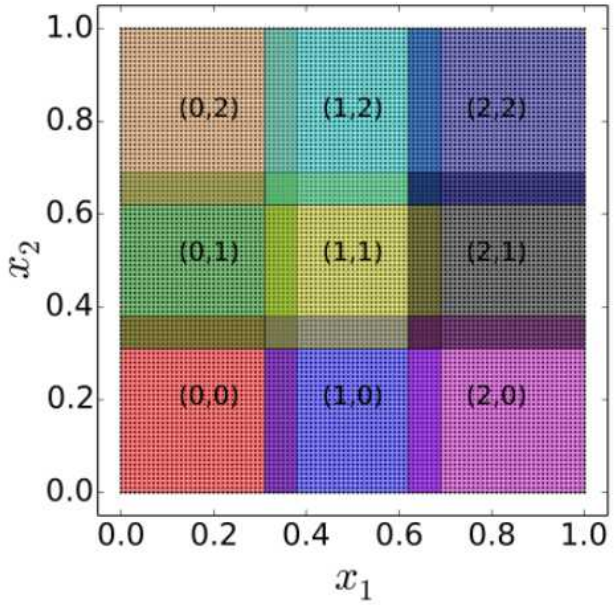
$$k(x_1, x_2) = 0.5 * (9.0 + 9.0 * \tanh\left(\frac{d(x_1, x_2)}{0.01}\right) + 1.0$$

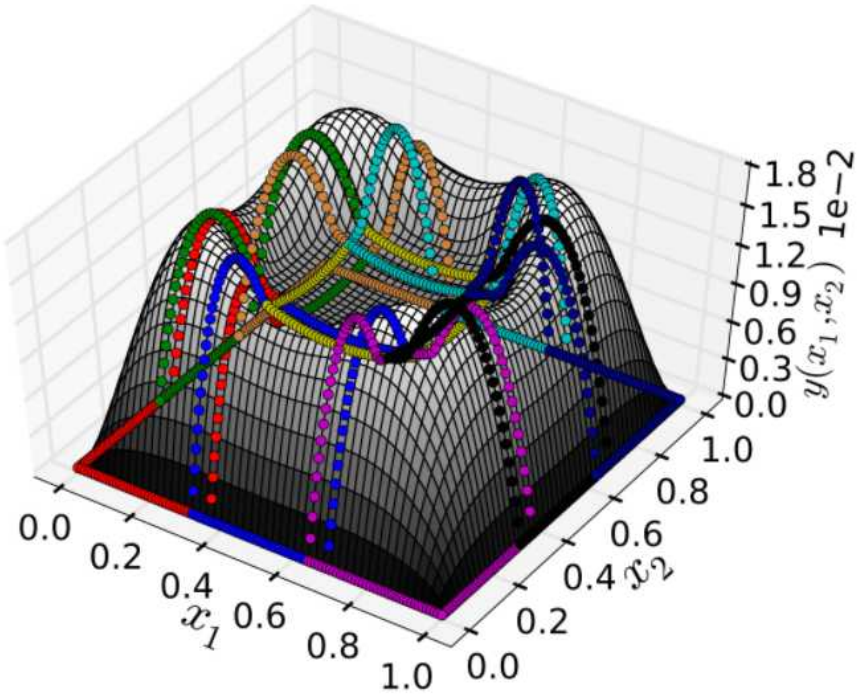
$$g(x_1, x_2) = 0.5 * (2.0 + 2.0 * \tanh\left(\frac{d(x_1, x_2)}{0.01}\right) - 1.0$$

  where $d(x_1, x_2) = 0.25 - \sqrt{(x_1 - 0.5)^2 + (x_2 - 0.5)^2}$.
- Non-trivial solution due to the steep gradient.
- Numerical solution based on 2nd-order finite-differences.

No faults

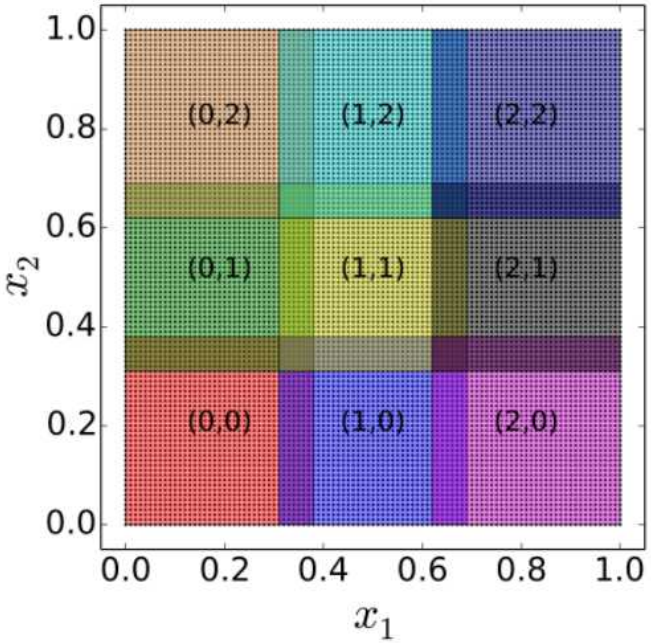
- Linear PDE $\Rightarrow$ boundary maps linear $\Rightarrow$ solution in one iteration.
- Length of overlap does not matter for linear problem.
- 3×3 subdomains, underlying uniform grid with $n_x = n_y = 101$.
- Solution is obtained(needed) only at boundaries.



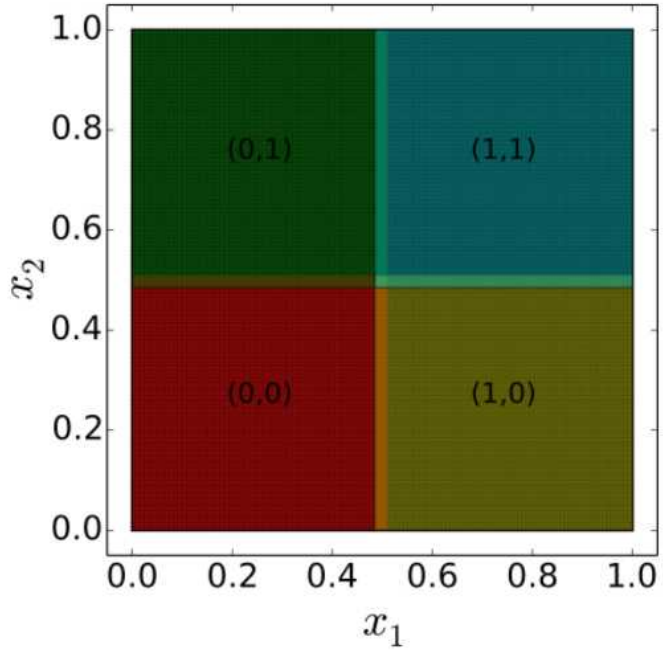


Two Problem Sizes

- With faults, we want to explore the effect of problem size.
- Problem size affects: communication, computation time, and faults occurrence.
- Task-manager configuration: 1 server, 62 clients.
- 40 ensemble runs.



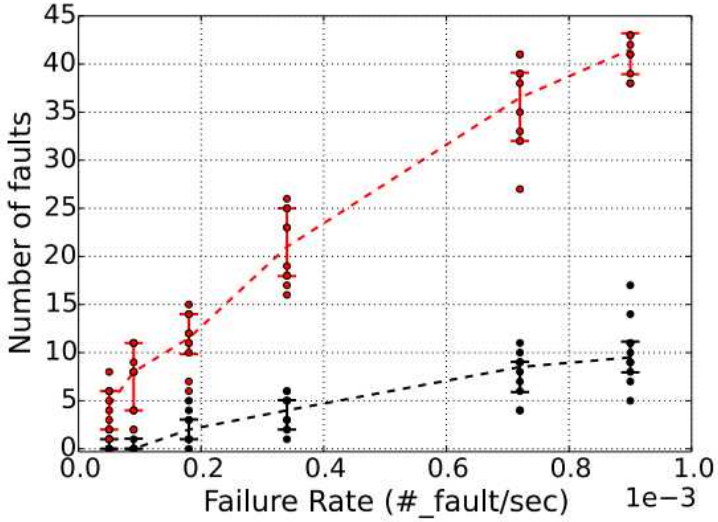
$n_x = n_y = 101$
(tag=$S9g_s37^2$)



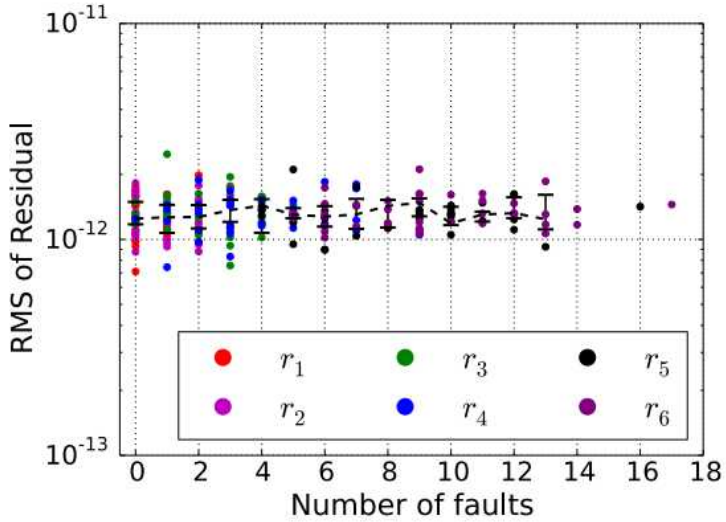
$n_x = n_y = 201$
(tag=$S4g_s103^2$)

Hard faults

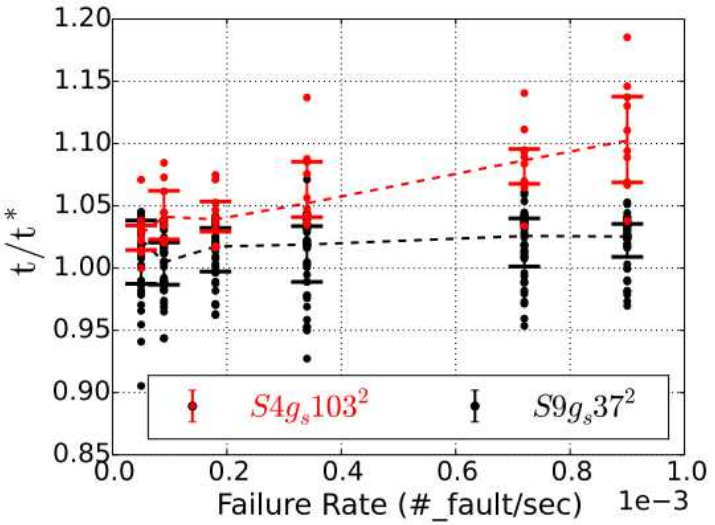
- Bigger problem is more expensive, yielding more faults.
- $S9g_s37^2$: bounded by communication cost.
- $S4g_s103^2$: sufficiently intense to reveal growing trend for runtime.
- Convergene in all cases.



Number of faults vs. failure rate.



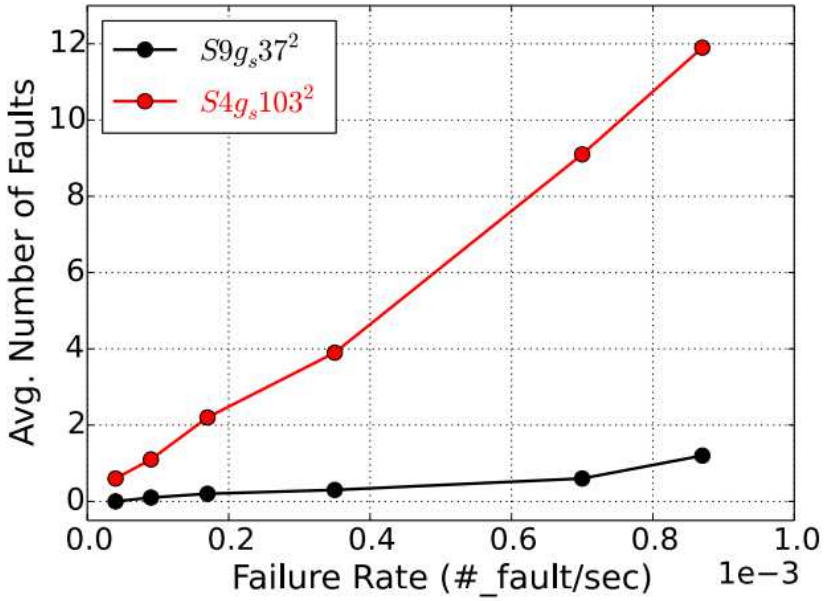
Residual vs. # of faults ($S9g_s37^2$).



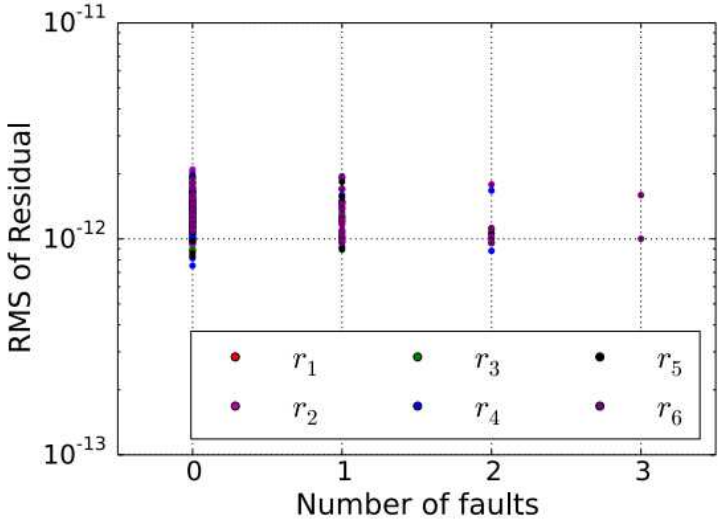
Normalized runtime vs. fault rate.

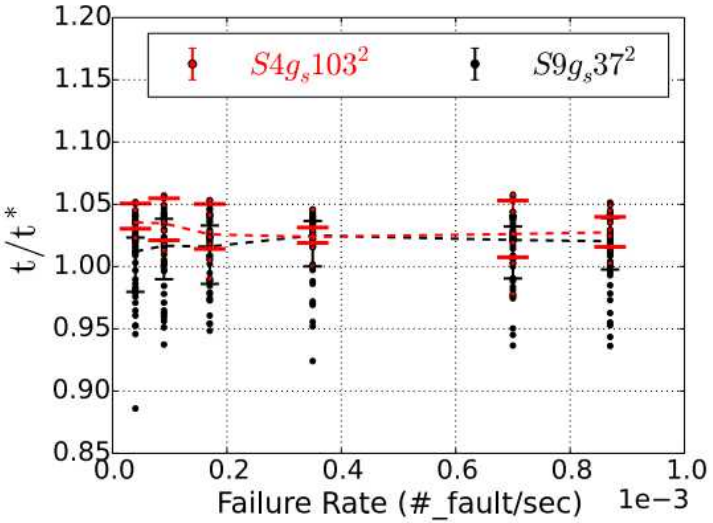
Computation faults

- Bigger problem, more faults.
- Overall, few faults: execution time is small.
- Minimally affect runtime, too few faults.
- Convergene in all cases.



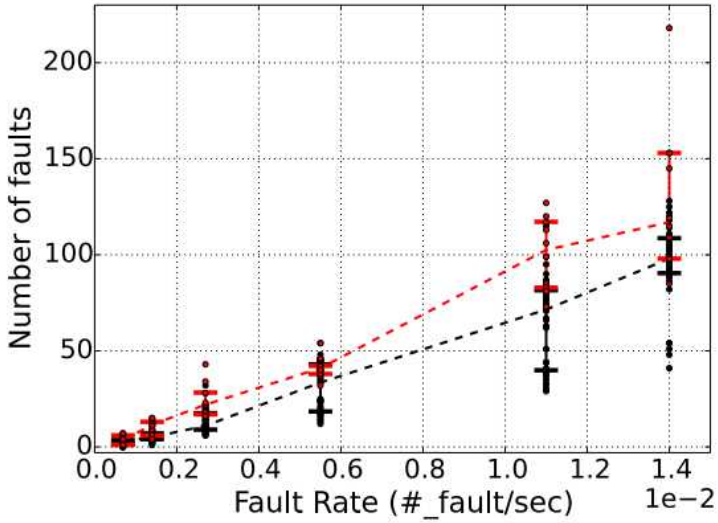
Avg. Number of faults vs. failure rate.



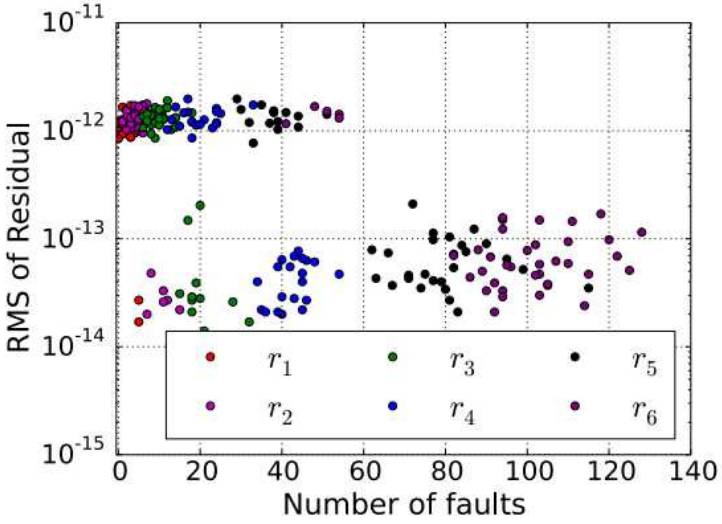


Network/Communication faults

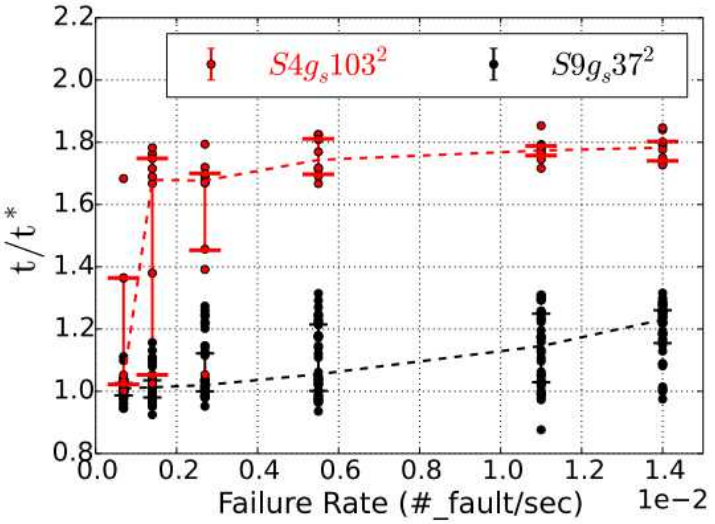
- Number of faults similar between small and large problem.
- Larger problem more affected by network faults because each task is more expensive to rerun.
- Convergence achieved in all cases.



Number of faults vs. failure rate.



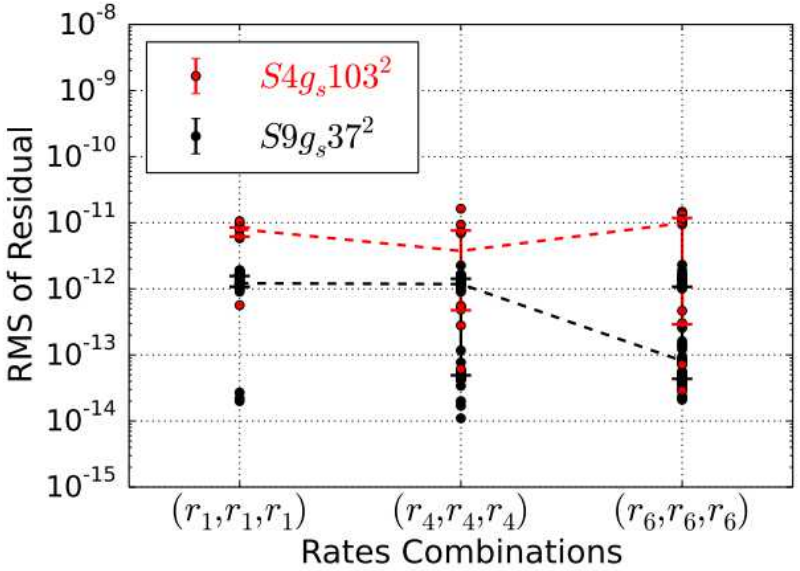
Residual vs. # of faults.

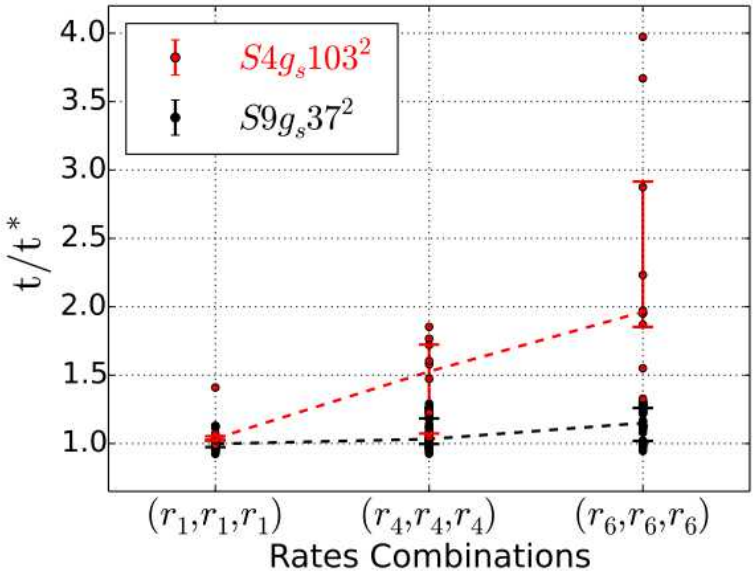


Normalized runtime vs. fault rate.

Mixed faults

- How does algorithm behave when all faults are activated and can occur concurrently?
- Three gradually increasing rates, from r_1 to r_6.
- Convergence is achieved in all cases.
- Runtime doubles for the largest case for highest rate.





Conclusions and Ongoing Work

- Approach/implementation provides resilience to:
  - Silent / Soft errors such as bit-flips.
  - Missing data due to communication issues or node failures.
- Sampling/decomposition approach provides concurrency/parallelism.
- Convergence is achieved in all cases.
- Ongoing work
  - Effective solution update for non-linear problems.
  - Dimensionality reduction.
  - Scalability for more complex problems.
  - How does it compare to regular solvers?

Acknowledgments

- This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under Award Number 13-016717.
- Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.