



A Component-Based Strategy for Scientific Application Development

Andy Salinger, SNL

SWP4XS = (Scalability)³

- | | | |
|---------------------------------|--------------------------|---------------------------------|
| 1. Scalable Algorithms | $O(\text{Cores}^1)$ | <i>E.g.</i> Multi-Level Prec |
| 2. Scalable Development | $O(\text{Developers}^1)$ | <i>E.g.</i> Build/Test/Release |
| 3. Scalable Capability Delivery | $O(\text{Libraries}^1)$ | <i>E.g.</i> Abstract interfaces |

Technical Strategy: Create, use, and improve a common base of modular, independent-yet-interoperable, software components.

“Components” = Libraries Software Quality Tools
 Interfaces Demonstration Applications

List of Reusable Math Libraries and Software Tool Components

Analysis Tools (black-box)

| |
|-------------------|
| Optimization |
| UQ (sampling) |
| Parameter Studies |
| V&V, Calibration |
| OUU, Reliability |

Analysis Tools (embedded)

| |
|----------------------|
| Nonlinear Solver |
| Time Integration |
| Continuation |
| Sensitivity Analysis |
| Stability Analysis |
| Constrained Solves |
| Optimization |
| UQ Solver |

Linear Algebra

| |
|---------------------|
| Data Structures |
| Iterative Solvers |
| Direct Solvers |
| Eigen Solver |
| Preconditioners |
| Matrix Partitioning |

Architecture-Dependent Kernels

| |
|--------------|
| Multi-Core |
| Accelerators |

Composite Physics

| |
|-----------------------|
| MultiPhysics Coupling |
| System Models |
| System UQ |

Mesh Tools

| |
|---------------------|
| Mesh I/O |
| Inline Meshing |
| Partitioning |
| Load Balancing |
| Adaptivity |
| Remeshing |
| Grid Transfers |
| Quality Improvement |
| DOF map |

Utilities

| |
|-------------------|
| Input File Parser |
| Parameter List |
| Memory Management |
| I/O Management |
| Communicators |

PostProcessing

| |
|-----------------|
| Visualization |
| Verification |
| Model Reduction |

Mesh Database

| |
|-------------------|
| Mesh Database |
| Geometry Database |
| Solution Database |

Data-Centric Algs

| |
|--------------------|
| Graph Algorithms |
| SVDs |
| Map-Reduce |
| Linear Programming |
| Network Models |

Local Fill

| |
|------------------------|
| Discretizations |
| Discretization Library |
| Field Manager |

Derivative Tools

| |
|----------------------|
| Sensitivities |
| Derivatives |
| Adjoints |
| UQ / PCE Propagation |

Physics Fill

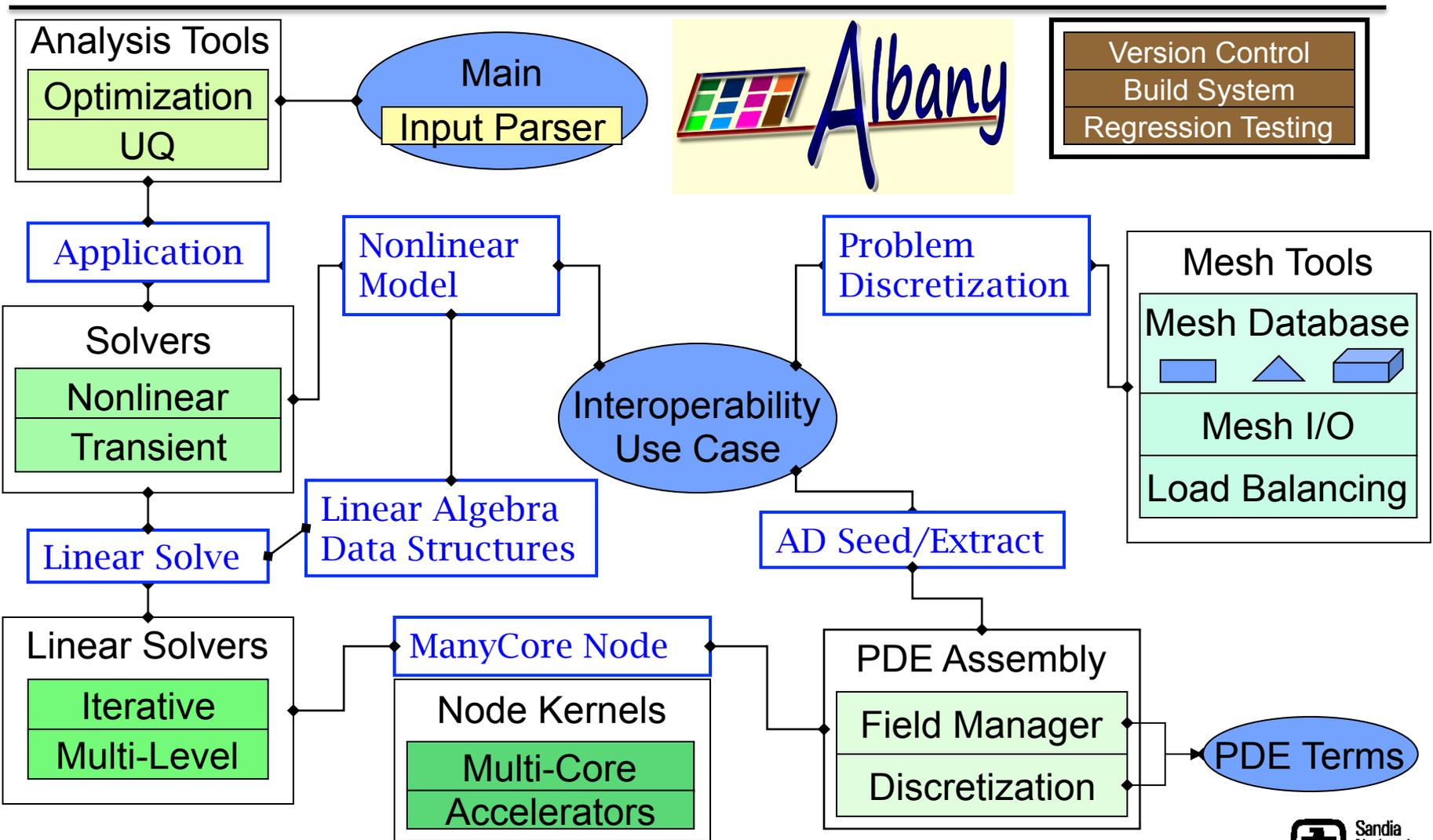
| |
|--------------------|
| Element Level Fill |
| Material Models |
| Objective Function |
| Constraints |
| Error Estimates |
| MMS Source Terms |

Software Quality

| |
|---------------------|
| Version Control |
| Regression Testing |
| Build System |
| Backups |
| Verification Tests |
| Mailing Lists |
| Unit Testing |
| Bug Tracking |
| Performance Testing |
| Code Coverage |
| Porting |
| Web Pages |
| Release Process |

Anatomy of a Component-Based Application:

Software Quality Tools Libraries Interfaces Demo Apps





Benefits of Building Applications from a Base of Components Include ...

Benefits of building and using **Libraries**:

- Code reuse
- Amortization of verification/maturation expenses
- ✓ Developed by expert
 - Access to continued development
- Efficient development by small teams
- Modularity improves agility
- Decreased application code base
 - Better encapsulation of protected code

Benefits of abstract **Interfaces**:

- Modularity improves agility, flexibility, extensibility, maintainability
- ✓ Multiple capabilities can be delivered together (e.g. all Trilinos linear solvers)

Benefits of adopting a standard set of **Software Quality Tools**:

- Tool development can be done by experts
- ✓ Sharing common tools improves agility of staff between projects
- Unified deployment of libraries decreases barriers to interdisciplinary research

Benefits of building **Demonstration Apps**:

- ✓ Provides a template for a new codes
- ✓ Demonstrates capability integration
- Drives interface development
- Defines scope of new libraries&interfaces
- Identifies gaps in our library coverage



Challenges of Building Applications from a Base of Components Include ...

Challenges:

- ✓ Requires more sophisticated software engineering
- ✓ Dependence on library developers that are not bound to your project
 - Credit for a project success may not go to the component developers
- Short-term decreases in productivity
 - Learning curve for using a library versus hand-coding
- Software support mortgage
- General-purpose implementations may lag in performance behind hand-tuned

The Components Technical Strategy Implies a Business Strategy as well

