Community building through software design

This talk:
https://jedbrown.org/files/20170221-SI2Community.pdf

Jed Brown jed@jedbrown.org (CU Boulder)
Collaborators: Barry Smith (ANL), Matt Knepley (Rice), Karl Rupp (TU Wien), and the rest of the PETSc team
Thanks to: DOE, NSF, Intel

SI2 Meeting, 2017-02-21

Firetran!

- Renders HTML 10% faster than Firefox or Chrome.
- but only if there is no JavaScript
  - recompile to use JavaScript
- Character encoding compiled in
- Mutually incompatible forks
- No confusing run-time proxy dialogs, edit file and recompile
- Proxy configuration compiled in
- For security, HTTP and HTTPS mutually incompatible
- Address in configuration file, run executable to render page
- Tcl script manages configuration file
- Plan to extend script to recompile Firetran with optimal features for each page
- Open source, but private development

Firetran!

- Renders HTML 10% faster than Firefox or Chrome.
- but only if there is no JavaScript
  - recompile to use JavaScript
- Character encoding compiled in
- Mutually incompatible forks
- No confusing run-time proxy dialogs, edit file and recompile
- Proxy configuration compiled in
- For security, HTTP and HTTPS mutually incompatible
- Address in configuration file, run executable to render page
- Tcl script manages configuration file
- Plan to extend script to recompile Firetran with optimal features for each page
- Open source, but private development

Firetran!

- Renders HTML 10% faster than Firefox or Chrome.
- but only if there is no JavaScript
  - recompile to use JavaScript
- Character encoding compiled in
- Mutually incompatible forks
- No confusing run-time proxy dialogs, edit file and recompile
- Proxy configuration compiled in
- For security, HTTP and HTTPS mutually incompatible
- Address in configuration file, run executable to render page
- Tcl script manages configuration file
- Plan to extend script to recompile Firetran with optimal features for each page
- Open source, but private development

Firetran!

- Renders HTML 10% faster than Firefox or Chrome.
- but only if there is no JavaScript
  - recompile to use JavaScript
- Character encoding compiled in
- Mutually incompatible forks
- No confusing run-time proxy dialogs, edit file and recompile
- Proxy configuration compiled in
- For security, HTTP and HTTPS mutually incompatible
- Address in configuration file, run executable to render page
- Tcl script manages configuration file
- Plan to extend script to recompile Firetran with optimal features for each page
- Open source, but private development

Firetran!

- Renders HTML 10% faster than Firefox or Chrome.
- but only if there is no JavaScript
  - recompile to use JavaScript
- Character encoding compiled in
- Mutually incompatible forks
- No confusing run-time proxy dialogs, edit file and recompile
- Proxy configuration compiled in
- For security, HTTP and HTTPS mutually incompatible
- Address in configuration file, run executable to render page
- Tcl script manages configuration file
- Plan to extend script to recompile Firetran with optimal features for each page
- Open source, but private development

Firetran!

- Renders HTML 10% faster than Firefox or Chrome.
- but only if there is no JavaScript
  - recompile to use JavaScript
- Character encoding compiled in
- Mutually incompatible forks
- No confusing run-time proxy dialogs, edit file and recompile
- Proxy configuration compiled in
- For security, HTTP and HTTPS mutually incompatible
- Address in configuration file, run executable to render page
- Tcl script manages configuration file
- Plan to extend script to recompile Firetran with optimal features for each page
- Open source, but private development

Firetran!

- Renders HTML 10% faster than Firefox or Chrome.
- but only if there is no JavaScript
  - recompile to use JavaScript
- Character encoding compiled in
- Mutually incompatible forks
- No confusing run-time proxy dialogs, edit file and recompile
- Proxy configuration compiled in
- For security, HTTP and HTTPS mutually incompatible
- Address in configuration file, run executable to render page
- Tcl script manages configuration file
- Plan to extend script to recompile Firetran with optimal features for each page
- Open source, but private development

Firetran!

- Renders HTML 10% faster than Firefox or Chrome.
- but only if there is no JavaScript
  - recompile to use JavaScript
- Character encoding compiled in
- Mutually incompatible forks
- No confusing run-time proxy dialogs, edit file and recompile
- Proxy configuration compiled in
- For security, HTTP and HTTPS mutually incompatible
- Address in configuration file, run executable to render page
- Tcl script manages configuration file
- Plan to extend script to recompile Firetran with optimal features for each page
- Open source, but private development

Firetran!

- Renders HTML 10% faster than Firefox or Chrome.
- but only if there is no JavaScript
  - recompile to use JavaScript
- Character encoding compiled in
- Mutually incompatible forks
- No confusing run-time proxy dialogs, edit file and recompile
- Proxy configuration compiled in
- For security, HTTP and HTTPS mutually incompatible
- Address in configuration file, run executable to render page
- Tcl script manages configuration file
- Plan to extend script to recompile Firetran with optimal features for each page
- Open source, but private development

Firetran!

- Renders HTML 10% faster than Firefox or Chrome.
- but only if there is no JavaScript
  - recompile to use JavaScript
- Character encoding compiled in
- Mutually incompatible forks
- No confusing run-time proxy dialogs, edit file and recompile
- Proxy configuration compiled in
- For security, HTTP and HTTPS mutually incompatible
- Address in configuration file, run executable to render page
- Tcl script manages configuration file
- Plan to extend script to recompile Firetran with optimal features for each page
- Open source, but private development

Firetran struggles with market share

- Status quo in many scientific software packages
- Why do we tolerate it?
- Is scientific software somehow “different”?

Computational Science & Engineering Challenges

- Model fidelity: resolution, multi-scale, coupling
  - Mathematical, computational, and modeling challenges
  - Best software capability written with different assumptions
  - Engages broader scientific and engineering community
  - Transient simulation is not weak scaling: $\Delta t \sim \Delta x$
- Analysis using a sequence of forward simulations
  - Inversion, data assimilation, optimization, experimental design
  - Quantify uncertainty, risk-aware decisions
  - Many nested loops, challenge to expose parallelism or exploit commonalities
- Increasing relevance $\Longrightarrow$ external requirements on time
  - Policy: 5 SYPD to inform IPCC
  - Weather, manufacturing, field studies, disaster response
  - Mistakes become costly
- "weak scaling" [...] will increasingly give way to "strong scaling"

  [The International Exascale Software Project Roadmap, 2011]

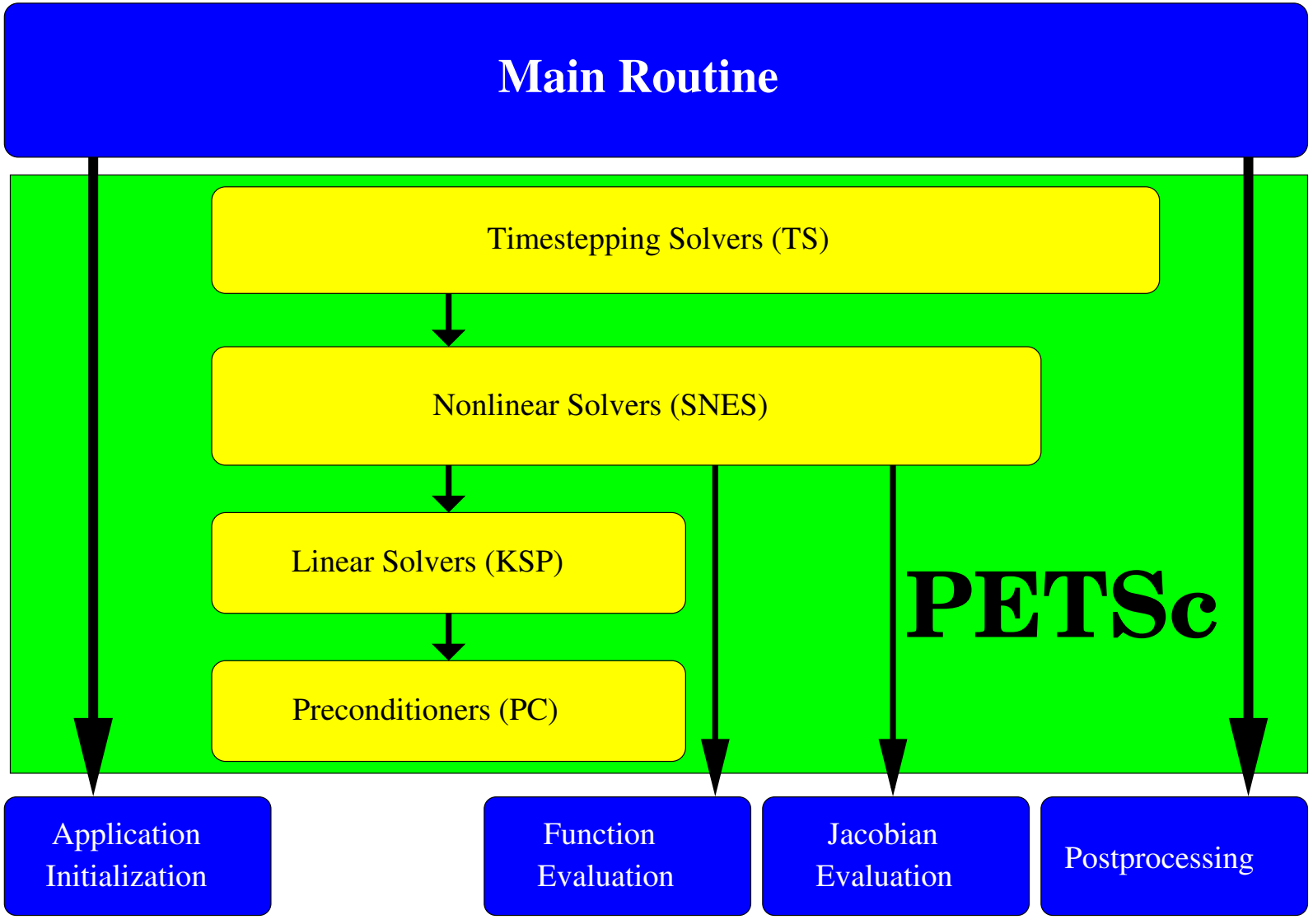
Usability: Packaging and distribution

- Code must be portable – any compiler, any platform
  - Need automatic tests to confirm
  - Including quirky HPC systems, or equivalent environments (containers)
- Developers underestimate challenge of installing software
- User experience damaged even when user's fault (broken environment)
- Package managers (Debian APT, RedHat RPM, MacPorts, Homebrew, etc.)
- Binary interface stability critical to packagers

Compile-time configuration

- configuration in build system: ad-hoc public API
- over-emphasis on “efficiency”
- templates are compile-time
  - combinatorial number of variants
- compromises on-line analysis capability
- create artificial IO bottlenecks
- offloads complexity to scripts and “workflow” tools
- limits automation and testing of calibration
- maintaining consistency complicates provenance
- PETSc Fail: mixing real/complex, 32/64-bit int

Flow Control for a PETSc Application



User modifications versus plugins

- Fragmentation is expensive and should be avoided
- Maintaining local modifications causes divergence
- Better to contain changes to a plugin
- `dlopen()` and register implementations in the shared library
- Invert dependencies and avoid loops
  - `libB` depends on `libA`
  - Want optional implementation of `libA` that uses `libB`
  - `libA-pluginB` depends on both `libA` and `libB`
  - `libA` loads its plugins at run-time
- Static libraries are anti-productive (tell your computing center)
  - Can sort-of do plugins with link line shenanigans
    - `LDLIBS="-lB $(libA-config -libs)"` dynamically search and include plugins (and their dependencies)
    - Constructor in `libA-plugin*` registers itself with `libA`
    - `cc -o app user-app.c -lB -lA-pluginB -lB -lA`
  - Still no reliable and ubiquitous way to handle transitive dependencies

User-developer false dichotomy

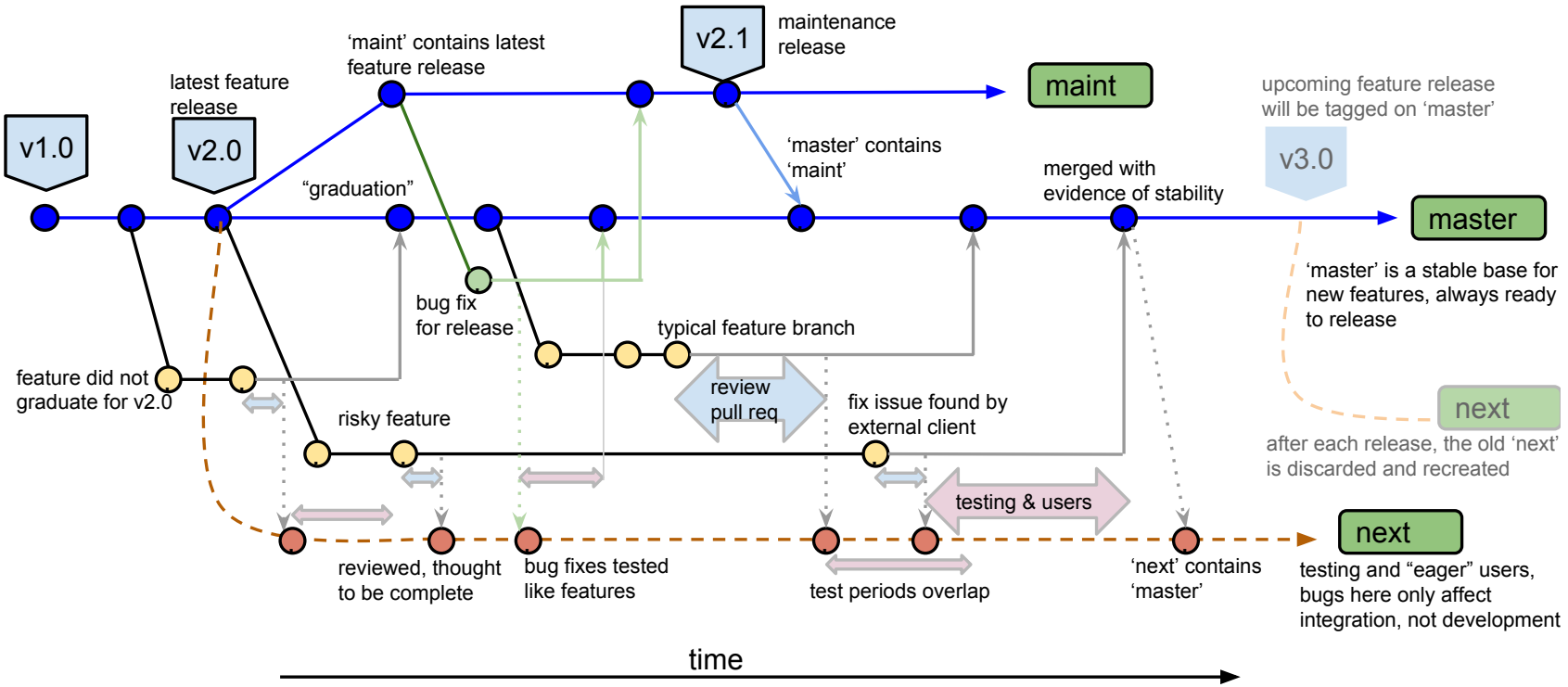
> *the distinction between "users" and "developers" is actively harmful — Matthew Turk (2013)*

- A plugin architecture tricks library users into becoming developers
- Write code for yourself, then contribute to community
- Obstacles
  - dirty, non-portable code
  - unnecessary assumptions or ad-hoc problem-specific data
- Portable types and utility functions, enable compiler warnings
- Interfaces can encourage users to avoid bad dependencies
  - Input arguments are same as library, have to do something to directly access application data
  - Fully custom extensions must also be possible
- Design for debuggability, document debugging tips
- Narrow vulnerability surface: input and output validation around extension points

Upstreaming and community building

- Maintainers should provide good alternatives to forking
- Welcoming environment for contributions
- Empower users – all major design decisions discussed in public
  - cf. Chatham House/“Harvey Birdman” Rule of copyleft-next
  - `https://github.com/richardfontana/hbr/blob/master/HBR.md`
- Privacy, “scooping”, openness
  - My opinion: social problem, deal with using social means
- Major tech companies have grossly underestimated cost of forking
- In science, we cannot pay off technical debt incurred by forking
- Provide extension points to reduce cost of new development

Simplified gitworkflows(7)



first-parent history of branch

merge history (not first-parent)

merges to be discarded when 'next' is rewound at next release

merge in first-parent history of 'master' or 'maint' (approximate "changelog")

merge to branch 'next' (discarded after next major release)

commit in feature branch (feature branches usually start from 'master')

commit in bug-fix branch (bug-fix branches usually start from 'maint' or earlier)

Review of library best practices

- Namespace everything
  - headers, libraries, symbols (all of them)
  - use `static` and visibiliy to limit exports
- Avoid global variables
- Avoid environment assumptions; don't claim shared resources
  - `stdout`, `MPI_COMM_WORLD`
- Document interface stability guarantees, upgrade path
- Binary interface stability
- User debuggability
- Documentation and examples
- Portable, automated test suite
- Flexible error handling
- Support

Application, Framework, or Library?

- "I'm an end-user application. The top of the stack."
  - Wishful thinking much? Engineers script mouse clicks around commercial GUI applications all the time.
- "Framework X is opinionated – it saves you time"
  - It makes unwarranted assumptions about the environment
  - Not to be confused with Good Defaults
- "You don't put AMR into your application, you put your application into AMR."

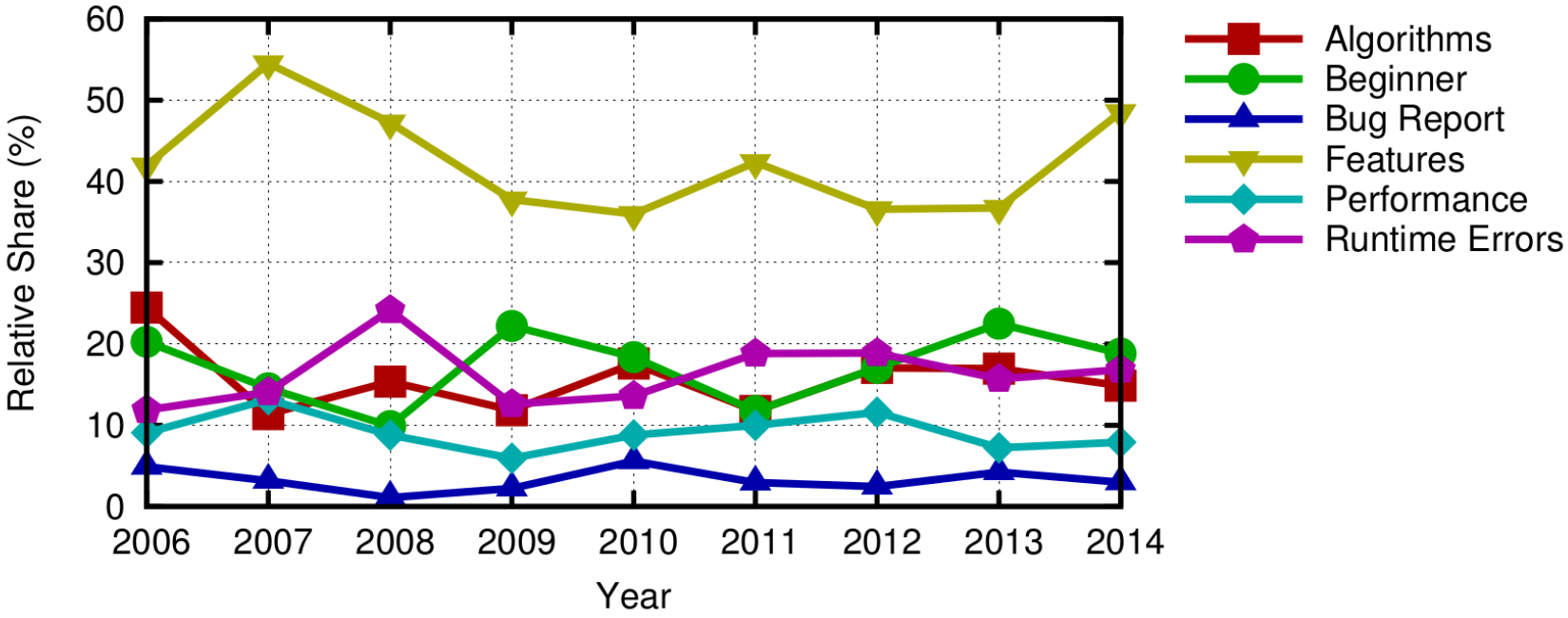
  `yt` is best thought of not as an application, but as a library for asking and answering questions about data. — Matthew Turk (2013)

- To embrace advanced analysis is to concede that higher levels exist and will need to operate your code. A programmatic API is a priority.

Choose dependencies wisely, but practically

- Licenses
  - PETSc has a permissive license (BSD-2); anything more restrictive must be optional
  - ParMETIS license prohibits modification and redistribution
  - But bugs don't get fixed, even with patches and reproducible tests
  - Result: several packages now carry patched versions of ParMETIS – license violation and namespace collision
- Parallel ILU from Hypre
  - Users Manual says PILUT is deprecated – use EUCLID
  - EUCLID has memory errors, evidently not supported
  - Repository is closed; PETSc doesn't have resources to maintain
  - Tough luck for users
- Encapsulation is important to control complexity
- Reconfiguring indirect dependencies breaks encapsulation
- Single library may be used by multiple components in executable
  - diamond dependency graph
  - conflict unless same version/configuration can be used for both

Support: `petsc-users` mailing list



- 964 emails in 2006 $\rightarrow$ 3947 emails in 2014
- Also have `petsc-dev` and `petsc-maint`
- Hard to tell at first contact if user is worth helping
  - Lots of work
  - Success stories are very satisfying
- 12 contributors in 2006–2007, 46 contributors in 2015

Verification and Validation

Verification without validation is sport; validation without verification is magic. — Anthony Scopatz

- Verification: solving the equations right
  - Manufactured solutions
  - Mesh refinement studies
  - Benchmarks for non-smooth/emergent behavior
  - Can include in automated tests
- Validation: solving the right equations
  - Comparison with observations
  - Do we have good initial/boundary conditions?
  - Data assimilation

Performance

- “We have to do it this way because of performance!”
  - static memory allocation only (complexity bubbles up, prevents composition)
  - no indirect function calls (virtual functions, callbacks; prevents extensibility)
  - template specialization everywhere (huge binaries)
- “Implicit solvers don’t scale”
  - Runs explicit diffusion instead
  - Bystanders choke on Gordon Bell Reflux
- Granularity is key: minimize scope, but don’t over-reduce
  - E.g., BLIS microkernel
- Lack of inlining hurts by spoiling vectorization more than anything
- Packing is very often an acceptable cost

End-to-end performance

- Education
- Preprocessing/custom implementation
- HPC Queue
- Execution time
  - Solvers
- I/O
- Postprocessing/visualization

Credit

- Citations are academic currency
- Encourage citing some living document
  - new developers can become authors
  - PETSc criteria: when you provide support and maintenance for your contributions
- Impossible to cite all transient dependencies
- But important to cite those that matter, regardless of branding
- `PetscCitationsRegister("@article...")`. run with `-citations` to see which modules were used.
- Decouple distribution from branding
  - Some people insist on controlling distribution, for licensing or branding reasons.
  - Rare in practice: most would rather contribute upstream

Outlook

- Social aspects
- Licenses, CLA versus Developer Certificate of Origin
- Scientific software shouldn't be "special"
- Usability is essential
- Plugins are wonderful for users, contributors,developers
- Just-in-time compilation is a useful abstraction
- Reviewing patches/educating contributors is a thankless task, but crucial
- Plan for support, making your life easier also helps users
- Versatility is needed for model coupling and advanced analysis
- Abstractions must be durable to changing scientific needs
- Plan for the known unknowns and the unknown unknowns
- The real world is messy!

References

Jed Brown, Matthew G Knepley, and Barry F Smith.
Run-time extensibility and librarization of simulation software.
Computing in Science & Engineering, 17(1):38–45, 2015.

Matthew J Turk.
Scaling a code in the human dimension.
In *Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery*, page 69. ACM, 2013.

Wolfgang Bangerth and Timo Heister.
What makes computational open source software libraries successful?
Computational Science & Discovery, 6(1):015010, 2013.

William D. Gropp.
Exploiting existing software in libraries: Successes, failures, and reasons why.
In *Proceedings of the SIAM Workshop on Object Oriented Methods for Inter-operable Scientific and Engineering Computing*, pages 21–29. SIAM, 1999.

Ulrich Drepper.
How to write shared libraries, 2002–2011.
`http://www.akkadia.org/drepper/dsohowto.pdf`.