

Non-Traditional Approaches to HPC Software Development



Presented at SOS 2014

March 19, 2014

Allen McPherson

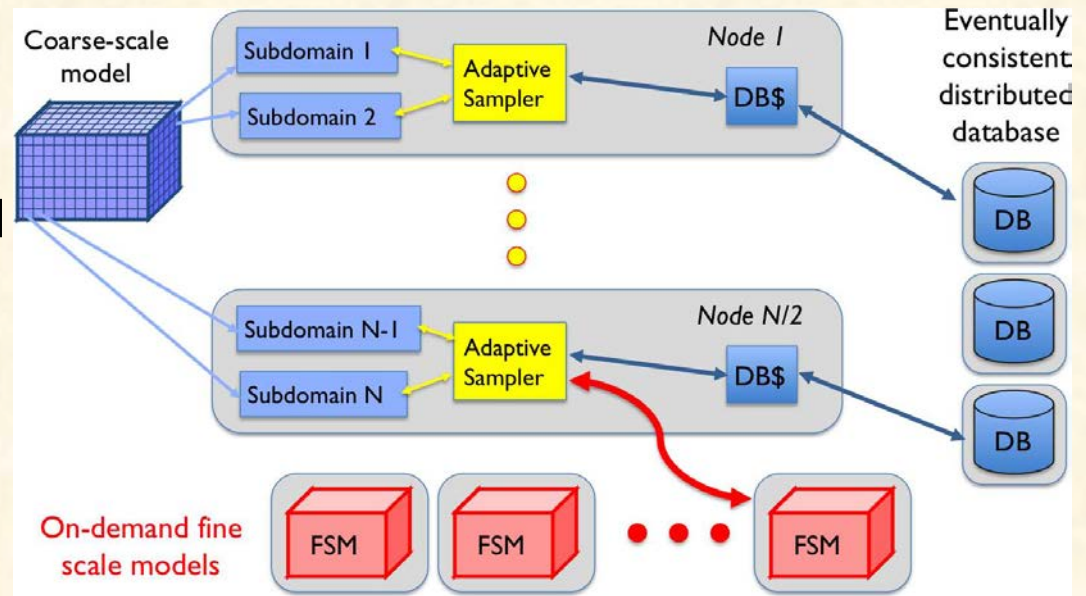
Los Alamos National Laboratory

Outline

- About ExMatEx and multi-scale applications
- Brief discussion of “programming models” and software stacks
- DSLs within the ExMatEx project
- A service-based ExMatEx software stack
- CoHMM multi-scale proxy application: implementations & results
- Future work leveraging ExMatEx efforts

Brief Introduction to ExMatEx

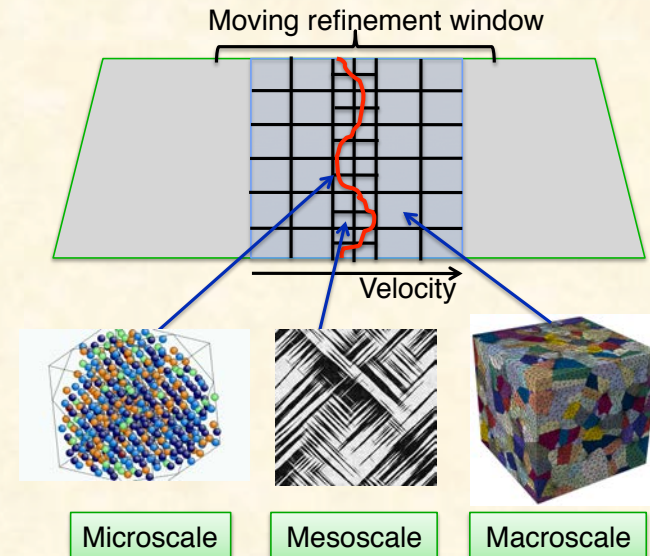
- Multi-scale materials
- Application driven
- Computer science focused
- ASCR Co-Design Center
 - LANL, LLNL, SNL, ORNL
 - Stanford, Caltech
 - \$4M/yr for 5 yrs
 - » starting third year



- Work in many areas: molecular dynamics, proxies, programming models, DSLs, multi-scale algorithms, vendor interface, runtimes, software stacks, etc.
- More info at <http://exmatex.org>

ExMatEx application: dynamic and multi-scale

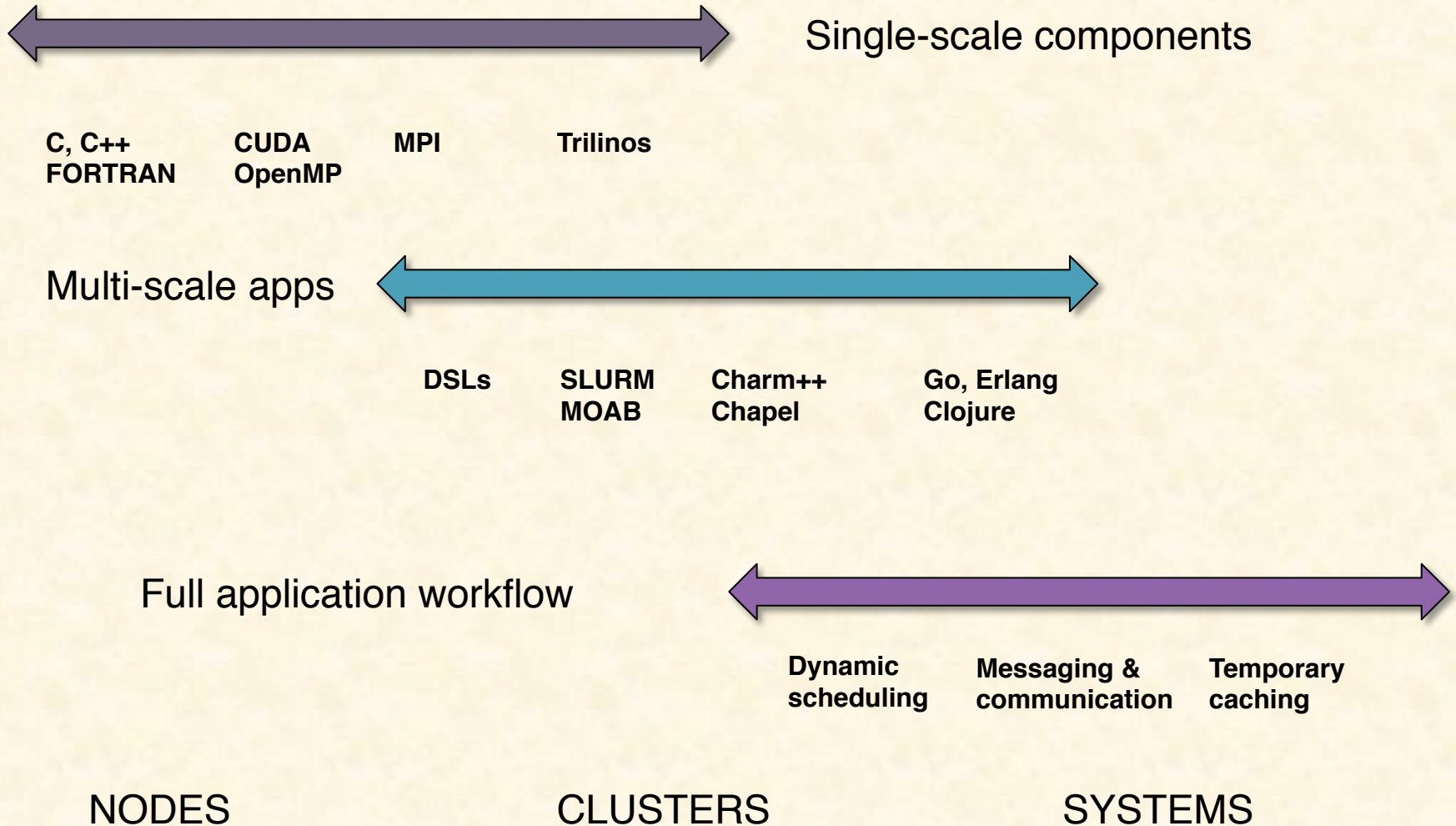
- ExMatEx apps are...
 - *Multi-scale*
 - *Dynamic*
- Multi-scale applications integrate components...
 - *That dynamically interact with each other on-the-fly*
- Components can be...
 - *Serial (single core)*
 - *Single node*
 - » Multi-core
 - » Accelerated (e.g. GPU)
 - *Multi-node*
 - » Groups of the above
 - » Existing libraries
- Additional requirements: fault tolerance, in situ analysis, etc.



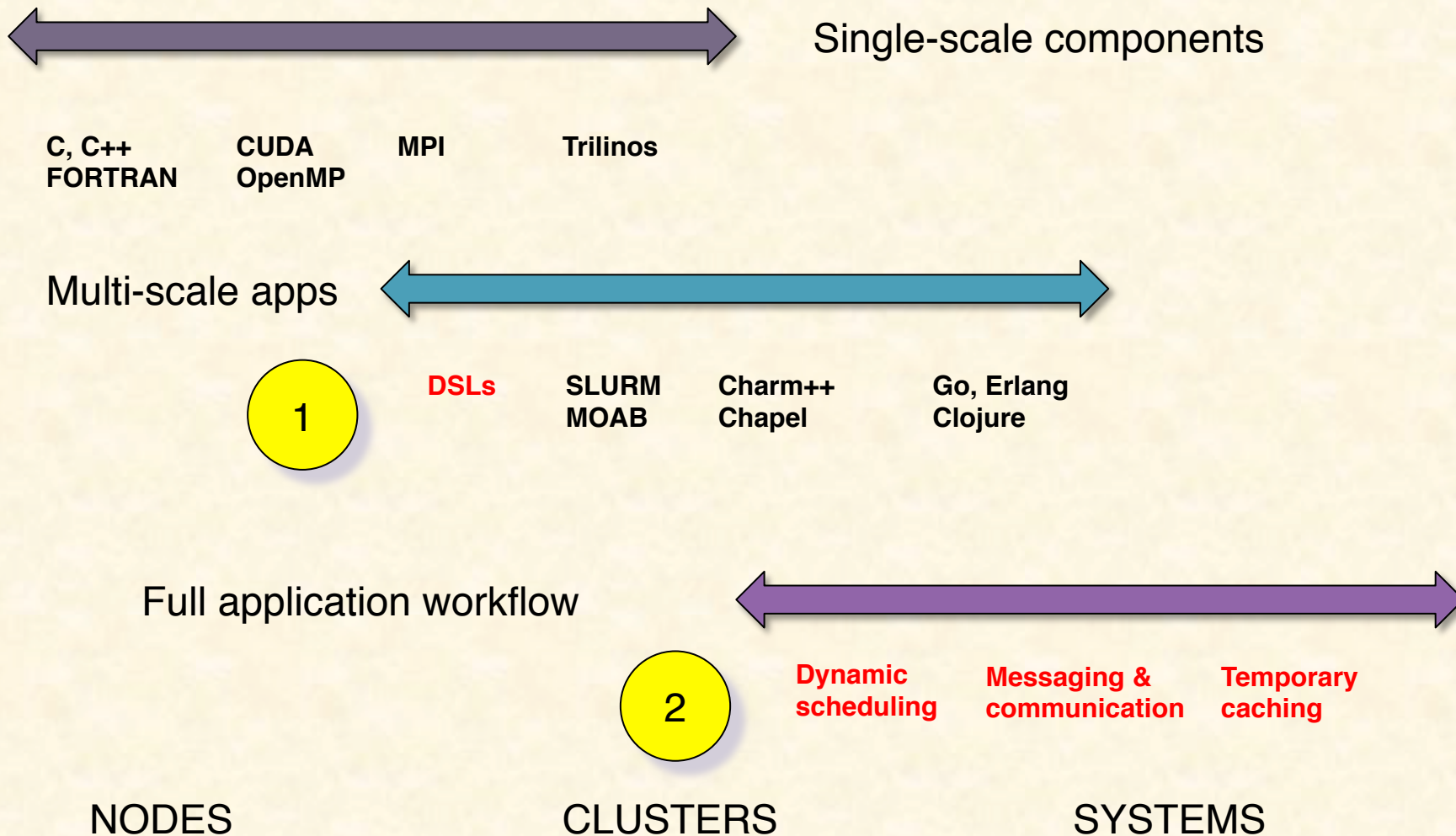
Traditional HPC Software Stack

- Current HPC software stack
 - *FORTRAN, C, C++, “X” (and libraries)*
 - *MPI*
 - *Static scheduler*
- Application writer does everything
 - *Load balancing, fault tolerance, dynamic communication patterns, dynamic task scheduling, data migration, code migration*
- More powerful tools can help
- Multiple ways to implement these application capabilities
 - *“Monolithic” languages*
 - *System services*
- Insulate developers and users with APIs and abstractions
- Polyglot approach—no single uber HPC programming model/language

Polyglot Development Environment



Polyglot Development Environment



Domain Specific Languages

- DSLs can insulate application developers from API complexity
- Small, focused languages for specific, restricted, problem domain
 - *Potential for productivity, portability, and performance*
 - *Not a new concept: SQL, LaTeX, Unix shell*
- We naturally have the required domain-restricted problem space
 - *At most, a few computational scales with scoped domains*
 - » Molecular dynamics: particles, force kernels, etc.
 - » Continuum: meshes, calculations on mesh elements (e.g. cells, vertices)
- Our goal is to design and implement DSLs for *ExMatEx domains*
 - *Co-design semantics of language with problem domain specialists*
 - *Develop compiler infrastructure that enables those DSLs to interoperate*
 - » Amongst themselves (for required multi-scale computation)
 - » With external languages to leverage other capabilities (e.g. solvers)
- DSLs applicable beyond ExMatEx (perhaps by broadening semantics)

Example of the Liszt DSL

```
val Position = FieldWithLabel[Vertex,Float3]("position")
val Temperature = FieldWithConst[Vertex,Float](0.0f)
val Flux = FieldWithConst [Vertex,Float](0.0f)
val JacobiStep = FieldWithConst[Vertex,Float](0.0f)
var i = 0;
while (i < 1000) {
  for (e <- edges(mesh)) {
    val v1 = head(e)
    val v2 = tail(e)
    val dP = Position(v1) - Position(v2)
    val dT = Temperature(v1) - Temperature(v2)
    val step = 1.0f/(length(dP))
    Flux(v1) += dT*step
    Flux(v2) -= dT*step
    JacobiStep(v1) += step
    JacobiStep(v2) += step
  }
  for (p <- vertices(mesh)) {
    Temperature(p) += 0.01f*Flux(p)/JacobiStep(p)
  }
  for (p <- vertices(mesh)) {
    Flux(p) = 0.f; JacobiStep(p) = 0.f;
  }
  i += 1
}
```

Mesh Elements

Topology Functions

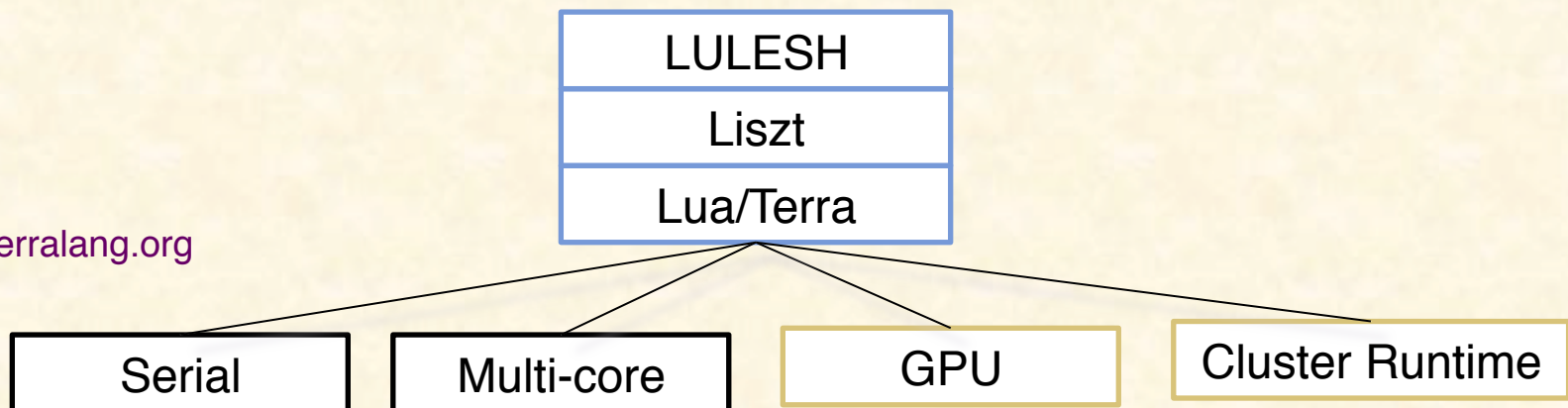
Fields (Data storage)

Parallelizable for

Implementation of DSL compiler: Terra

- Stanford University collaboration (Prof. Pat Hanrahan)
- Terra: low-level system programming language for building DSLs
 - *Enables JIT compiled DSLs*
 - *Enables interoperability with existing applications and libraries*
- Designed to interoperate seamlessly with Lua
 - *Lua is high-level scripting language*
 - *Use Lua to meta-program Terra*
 - *Lua code can generate arbitrary Terra programs at runtime*

<http://terralang.org>



Terra status and Y3 DSL efforts

- Terra compiler infrastructure built and released open source
 - *Using LLVM*
 - *Support for vector instructions*
 - <http://terralang.org>
- Terra compiling test DSLs at reasonable performance
 - *Matmult, stencils, nbody*
- Terra implementation of Liszt underway
 - *Compiling and generating code for single-core runtime*

Evolving the HPC Software Stack

- Recall that developer shouldn't be required to “do everything”
- An ExMatEx software stack—system **services** provide support
 - *Node-level work still focused on “X”*
- Leverage “web” and “cloud” software services
- Many of today's successful startups use diverse software stacks
 - *Build an application*
 - *Scale to 100's of millions of users*
 - *Sell your self to Facebook for...*
 - » \$1B: Instagram
 - » \$19B: WhatsApp (450M users on Erlang with 10 engineers!)
- Identify gaps and shortcomings in these technologies
 - *Are there areas where engineering dollars can enable adoption?*
- Must carefully manage granularity to absorb overheads

Is performance the only metric?

- How performant are your codes now, really?
- Performance should be one of many goals when developing a code
- Performance can conflict with other goals
 - *Time to solution*
 - *Code agility*
 - *Code maintainability*
 - *Developer productivity*
- Codes must be build to well defined (perhaps evolving) requirements
 - *Workload & workflow...*
 - *...drive realistic performance targets*
 - *Agility requirements*
 - *Without requirements it's difficult to judge success*

Fundamental System Services

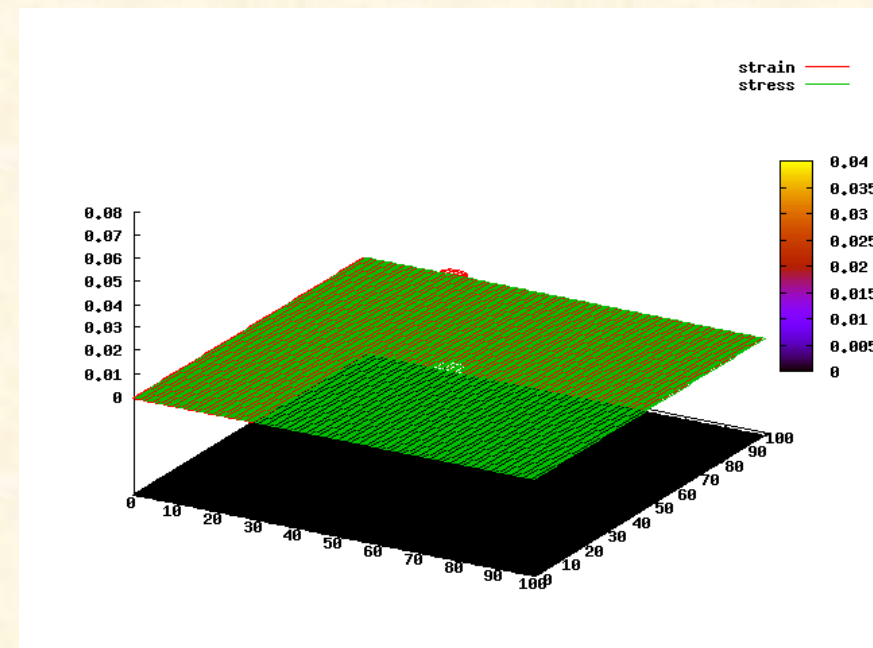
- Scheduling
 - *Spark, Mesos (more “cloudy”)*
 - *CnC, Erlang, etc. (“steal” a service from a “monolithic” language)*
- Messaging
 - *MPI (single component)*
 - *ZeroMQ, RabbitMQ, NSQ*
- Caching
 - *NoSQL databases*
 - » MongoDB, Cassandra, redis, RAMCloud
 - *Potentially most useful*
 - » Fault tolerance
 - » Material properties, EOS (service!), etc.
 - » Avoid redundant computation
 - » *Communication*

Caveat: Not Advocating “Cloud HPC”

- This is not about fitting HPC into map reduce paradigm
 - *People are already doing good work in this area*
 - *Map reduce is one computing paradigm built on fundamental services*
 - *Multi-scale HPC codes are just another paradigm on same services*
- This is not a cloud data center solution
 - *This is a bridge too far for our systems infrastructure*
 - *Maybe some day, way down the road*
 - » Virtualization!
 - » Heterogeneous systems
 - » Incremental procurements
 - *Most of the service software can run in user space*
 - » Least disruptive for systems folks
 - » Dynamic apps run within a traditional static partition

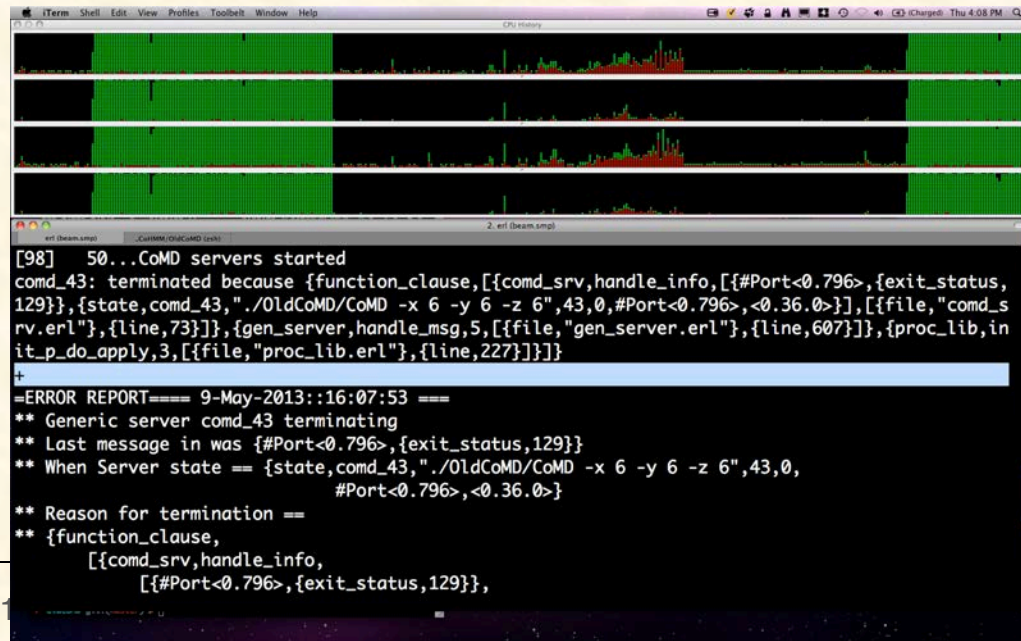
CoHMM: multi-scale materials proxy app

- High strain-rate dynamics, continuum mechanics coupled to molecular dynamics (CoMD proxy)
- Heterogeneous Multi-scale Method with adaptivity
- ExMatEx proxy app for experimentation with service-based CS approach (extended by LANL co-design summer school)



CoHMM: Early Erlang implementation

- Erlang
 - *Distributed, concurrent, functional language with reliability features built in*
 - » In use for over 20 years building async/concurrent, fault tolerant apps
 - § Other languages have similar, but not all, features (Go, Clojure, Scala/AKKA)
 - *Explore language/runtime/resiliency space*
 - » Launch independent CoMD instances (binary CoMD linked in)
 - » Message passing from CoMD's to coarse scale solver
 - » Fault tolerance through Erlang's *supervisors*



```
[98] 50...CoMD servers started
comd_43: terminated because {function_clause, [{comd_srv, handle_info, [{#Port<0.796>, {exit_status, 129}}, {state, comd_43, "/OldCoMD/CoMD -x 6 -y 6 -z 6", 43, 0, #Port<0.796>, <0.36.0>}], [{file, "comd_srv.erl"}, {line, 73}]}], {gen_server, handle_msg, 5, [{file, "gen_server.erl"}, {line, 607}], {proc_lib, init_p_do_apply, 3, [{file, "proc_lib.erl"}, {line, 227}]}]}

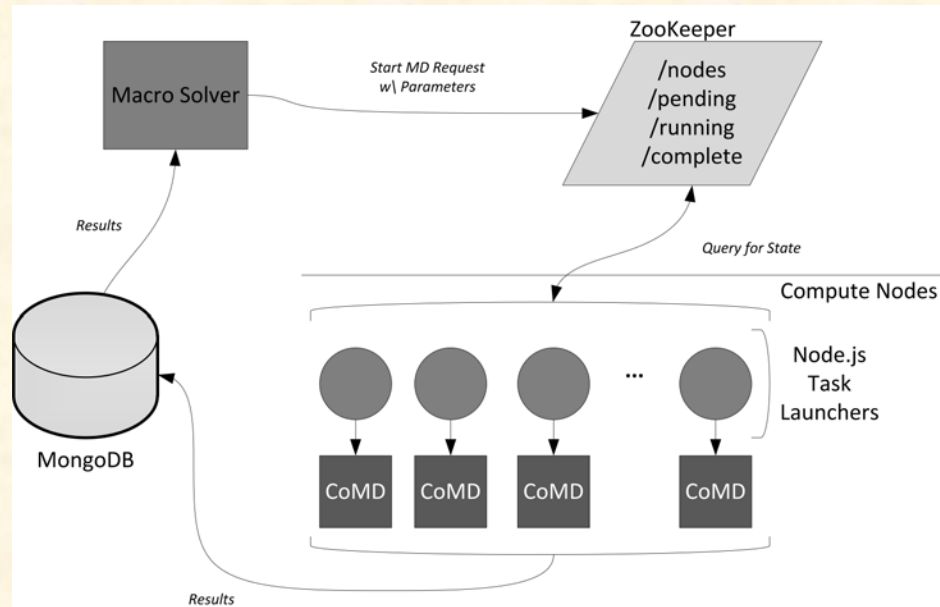
===== 9-May-2013::16:07:53 =====
** Generic server comd_43 terminating
** Last message in was {#Port<0.796>, {exit_status, 129}}
** When Server state == {state, comd_43, "/OldCoMD/CoMD -x 6 -y 6 -z 6", 43, 0, #Port<0.796>, <0.36.0>}

** Reason for termination ==
** {function_clause,
    [{comd_srv, handle_info,
      [{#Port<0.796>, {exit_status, 129}},
```


CoHMM: Early “Cloud” implementation

- Most “cloudy” of these three proxies
 - *These technologies usually not seen in scientific simulation apps*
- Apache ZooKeeper
 - *Distribute computation to pool of nodes*
- Node.js
 - *Launch computations (real CoMD)*
 - *Stateless, run and exit*
- redis
 - *NoSQL database*
 - *Used to communicate results*
 - » CoMD stores results
 - » 1D HMM code reads results

```
terminal window: mitchell@centos:~/multiscale-demo
File Edit View Search Terminal Help
Task Completed: '/tasks/complete/t-0000000189' / '1,-1,188,/home/mitchell/multiscale-demo/CoMD/CoMD,-s 1.01'
# children = 443, First child = t-0000000188
taskInfo = 1,-1,187,/home/mitchell/multiscale-demo/CoMD/CoMD,-s 1.01
path = /tasks/running/t-0000000188
DEBUG: params var = 1,-1,187,/home/mitchell/multiscale-demo/CoMD/CoMD,-s 1.01
Launching with: >/home/mitchell/multiscale-demo/CoMD/CoMD -s 1.01
Task exited with code: 0
Task Completed: '/tasks/complete/t-0000000086' / '1,-1,54,/home/mitchell/multiscale-demo/CoMD/CoMD,-s 1.01'
# children = 442, First child = t-0000000085
taskInfo = 1,-1,55,/home/mitchell/multiscale-demo/CoMD/CoMD,-s 1.01
path = /tasks/running/t-0000000085
DEBUG: params var = 1,-1,55,/home/mitchell/multiscale-demo/CoMD/CoMD,-s 1.01
Launching with: >/home/mitchell/multiscale-demo/CoMD/CoMD -s 1.01
Task exited with code: 0
Task Completed: '/tasks/complete/t-0000000188' / '1,-1,187,/home/mitchell/multiscale-demo/CoMD/CoMD,-s 1.01'
# children = 441, First child = t-0000000080
taskInfo = 1,-1,60,/home/mitchell/multiscale-demo/CoMD/CoMD,-s 1.01
path = /tasks/running/t-0000000080
DEBUG: params var = 1,-1,60,/home/mitchell/multiscale-demo/CoMD/CoMD,-s 1.01
Launching with: >/home/mitchell/multiscale-demo/CoMD/CoMD -s 1.01
```



Co-design Summer School

- Los Alamos IS&T Co-Design Summer School
 - For recruiting and advertising LANL's co-design work
 - Small (6), multi-disciplinary team of students
 - 50/50 mix of US/FN
 - Work on co-design problem
 - » 2011 & 2012: LANL CoCoMANS LDRD
 - » 2013: ExMatEx
 - Publish results
 - » Open source, reports, talks, posters
 - » Students @ SC, SIAM, nVidia GTC
 - 2013
 - » ExMatEx software stack experiments
 - » CoHMM proxy application

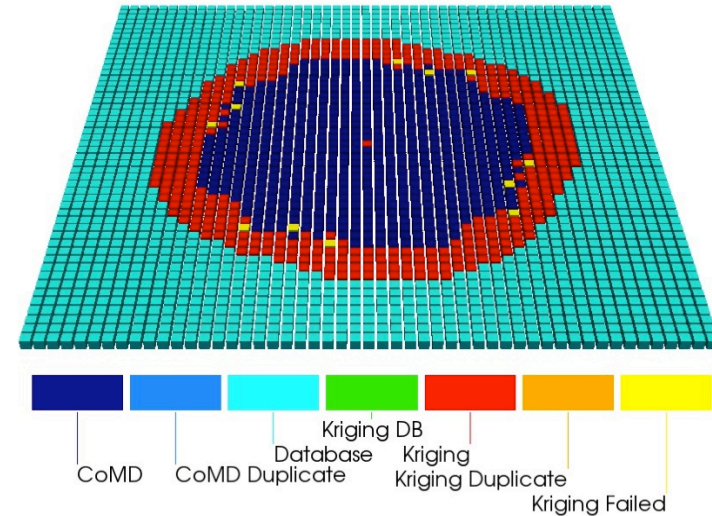


Summer School: 2013 Students

Name	School	Area
Robert Pavel	University of Delaware	CS
Axel Rivera	University of Utah	CS
Venmugil Elango	Ohio State	CS
Emmanuel Cieren	Laboratoire Bordelais de Recherche en Informatique	HPC
Dominic Roehm	Universität Stuttgart	Physics
Bertrand Rouet-Leduc	École Normale Supérieure	Physics

CoHMM: summer school implementation

- Key CS idea: use open source software for fundamental services: scheduling, messaging, caching
- Acceleration with adaptive task *scheduling* only where needed
- Acceleration by *caching* previously computed results
- Fault tolerance by *caching* particle positions for *scheduling* a restart at the node level—computation runs through a failure

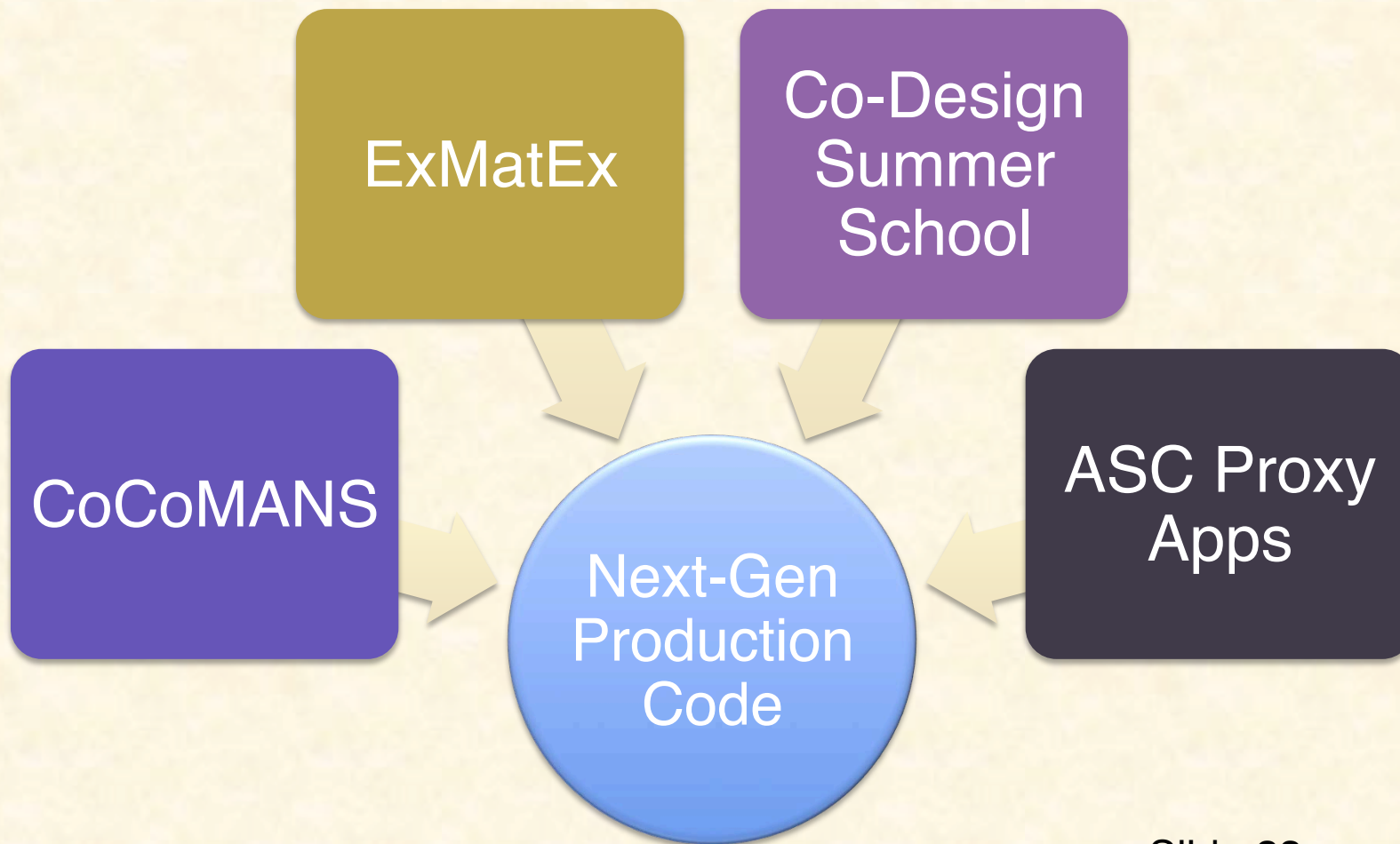


System	Dimension	Adaptive	Database	Fault Tolerant	Status
HPX	Bugs and lack of documentation. Triage it away.				Abandoned
Scioto	1D, 2D	AMR, Kriging	redis	No	OK
Pathos	1D	Yes	No	Process	OK
Intel CnC	2D	No	No	No	OK
Charm++	Synthetic benchmarks only. Evaluate load-balance.				Eval. only
Spark	1D, 2D	AMR, Kriging	redis	CoMD atom	OK
Mesos	Evaluated favorably. Installation issues.				Eval. only
Swift	1D	No	No	Process	CoMD 1.0
Erlang	1D	No	No	Process	CoMD 1.0
Scala	1D	No	No	No	Simple MD
"Cloud"	1D	No	multiple	Process	CoMD 1.1

CoHMM: summer school lessons learned

- Detailed technical analysis still in preparation...
 - *3 papers: 1 accepted, 1 submitted, 1 in endless prep*
- Early information and “lessons learned”
 - *Scale to low-100’s of nodes, low-1000’s of cores*
 - *Testing at 50x50 grid*
 - *CoMDs take on the order of 8-20 seconds, HMM fast*
 - *Under these constraints, overheads are low*
 - » Scheduling of CoMD runs (negligible)
 - » Read/write performance of in-memory database
 - § < 10% for 1000 processes hammering DB)
 - » Service provided by runtime systems made many, varied implementations possible in short amount of time (10 weeks)

Future Work



Slide 23

ExMatEx Contact Information

- <http://exmatex.org>
 - *Project web site*
 - *“Research Areas” → “Runtime Systems” for info related to this talk*
 - » *Publications*
 - § *CoHMM and Summer School (1 accepted, 1 in submission, 1 in preparation)*
 - *“News” announcing publication status and proxy release*
- <https://github.com/exmatex>
 - *Project open source site*
 - *CoMD 1.1*
- exmatex@lanl.gov
 - *Project mailing list*
 - exmatex-leads@lanl.gov *if you don’t want to spam entire list*
- mcpherson@lanl.gov
 - *For copies of talks and paper, or to discuss this approach*