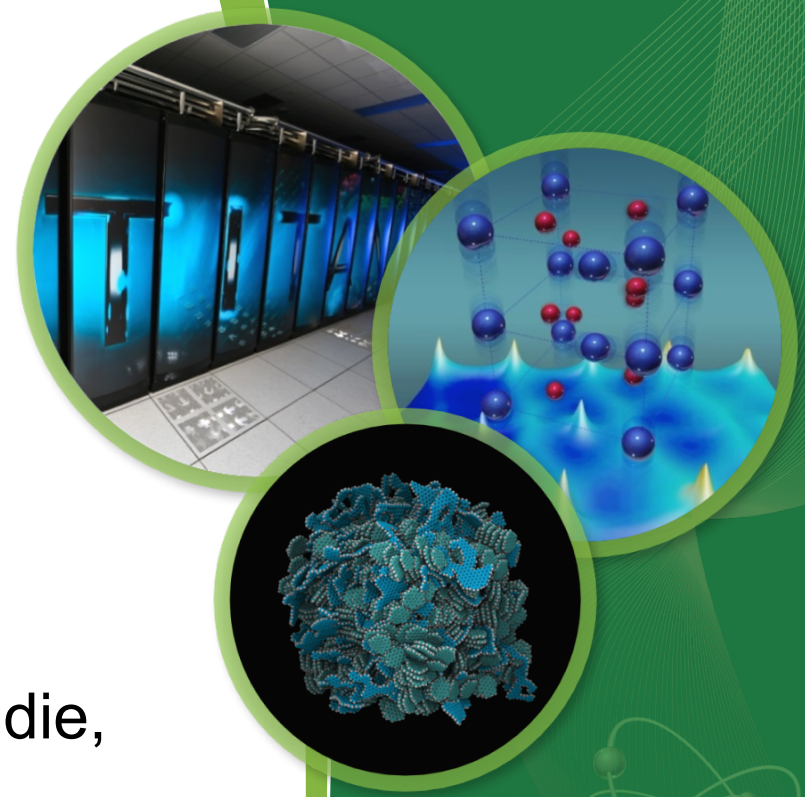
If Virtualization is the Answer, what was the Question?



David E. Bernholdt, Geoffroy Vallee, Thomas Naughton, Stuart Slattery, Damien Lebrun-Grandie, and Barney Maccabe
Oak Ridge National Laboratory

in collaboration with the Hobbes Exascale OS/R Team

This work has been supported by US Department of Energy, Office of Science, Advanced Scientific Computing Research program, and *performed in part at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC for the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.*

OAK RIDGE National Laboratory

Convergence through Federation, don't count on unification alone

- Many benefits for a unified (converged) stack
  - Common code base – less code to maintain
  - Common build environments – fewer target platforms
- Many challenges …
  - Won't happen in the near future
    - Likely an 80% solution
  - Effort to migrate code to the new stack
  - Would inhibit innovation
  - Won't extend to sources of data
    - What's the next grand convergence?

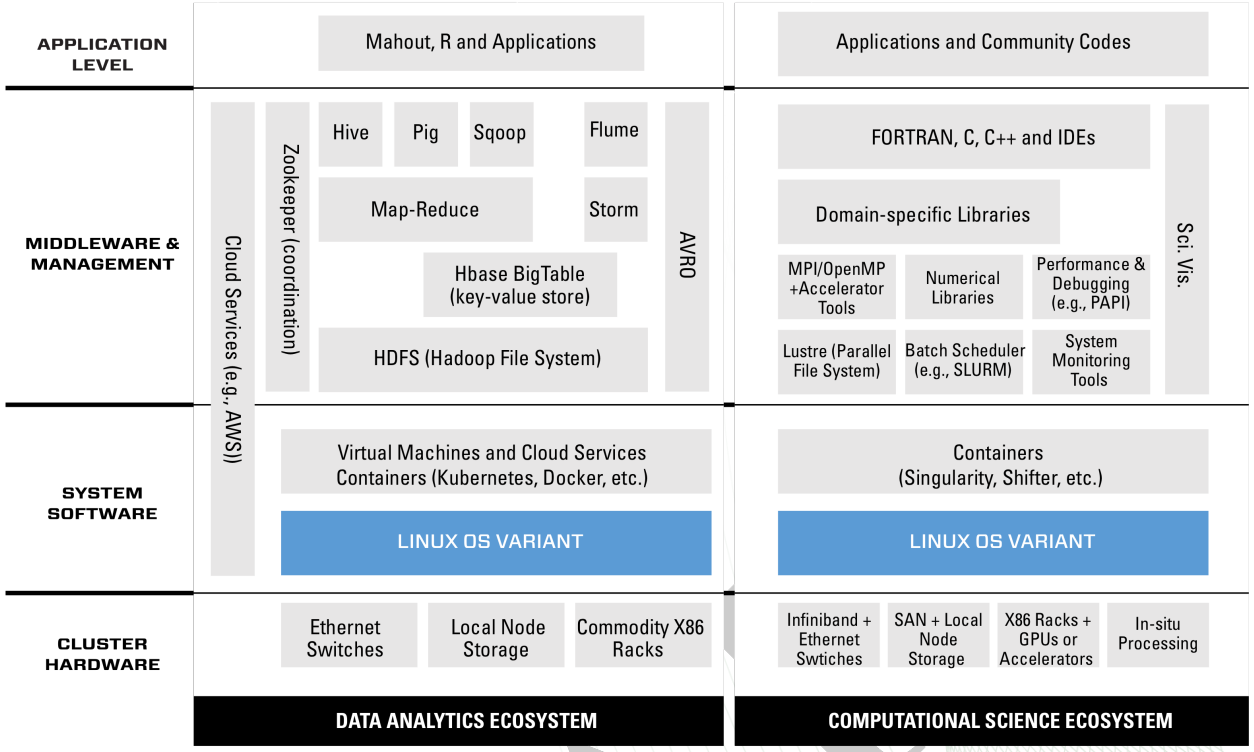


Figure 1: The two software ecosystems

A Good Buzzword Takes on a Life of its Own

- Technological buzzwords usually start as a novel solution to one or more challenges facing the community
- They become buzzwords when the excitement (hype?) about the solution overtakes the memory of the motivating challenge
  - Then they become the solution for *everything*
  - And we forget the problems we were really trying to solve in the first place
- Virtualization has been a popular buzzword in HPC for 10+ years
- Containers more recent, but (currently) more hype
  - **For purposes of this presentation, containers are just another virtualization approach**

Virtualization

We can solve any problem by introducing an extra level of indirection
Butler Lampson (who credited David Wheeler)

- Virtualization is the ultimate “level of indirection”
  - Usually implies multiplicity & isolation
  - Mechanisms: processes, containers, virtual machines, …
- Virtualization in HPC: CUG 2006, Lugano
  - “Recent Trends in Operating Systems and Their Applicability to HPC”
    - Maccabe, Bridges, Brightwell, Riesen
  - Argued for virtualization as the basis for HPC system software
    - Xen was new and primarily aimed at banking systems (the “cloud” wasn’t ubiquitous)
- Original Goal: “Bring your own stack”
  - Fun for OS research….. *Not very compelling for most people*
  - Potential for near “bare metal” performance….. *May conflict with productivity concerns*
  - MM5 failed after a Linux upgrade….. *Still important, but not for active development communities*

So What *Is* Virtualization Good for?

- Good is relative to ***value to HPC applications***
- Application **packaging**, distribution, and deployment
  - Ability to bundle up the complete software stack supporting an application
  - Ability to convey those bundles to other users
  - Ability to launch (applications from) those bundles on (various) HPC systems
  - Ability to develop on your laptop and run on an HPC system

  Build the app and all its dependencies on a new system, or bring a complete image with you?
- Runtime **resource partitioning**/management (*maybe*)
  - Significant motivation for cloud environments
  - HPC usage model is very different – we don't share nodes among multiple users
  - Perhaps control over sharing of resources for a single complex simulation would be useful
  - Many of the HPC use cases end up looking contrived

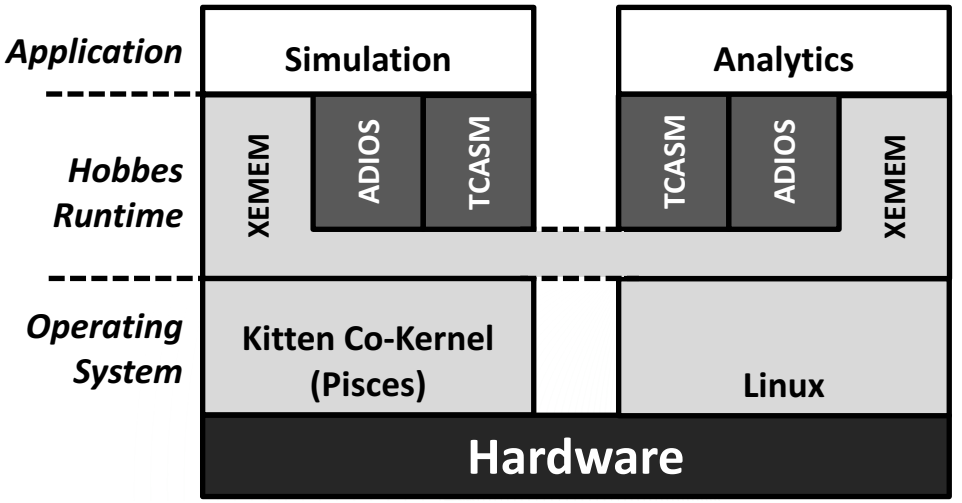
Consider Two Use Cases for Packaging in HPC

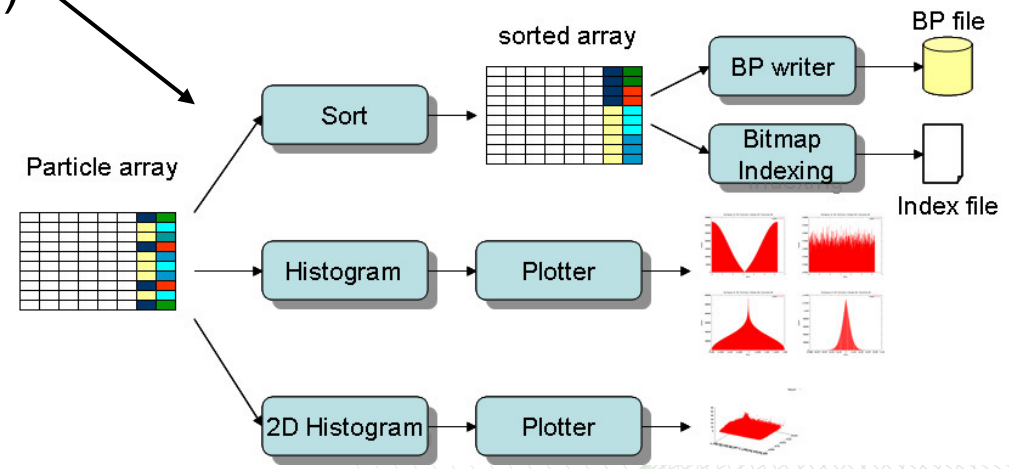
- **Composition** of complex applications
  - Many computational science applications have extensive dependencies
  - Increasingly, multi-scale multi-physics, multi-… applications are being created by coupling of codes which were previously standalone

- **Reproducibility** of simulations
  - Reproducibility of scientific research in general, and computational science research in particular is under increasing scrutiny
  - Concerns about ability to reproduce one's own work in the future
  - Concerns about ability of the community to appropriately peer review work

Initial Composition Demonstration: Simulation + Analytics

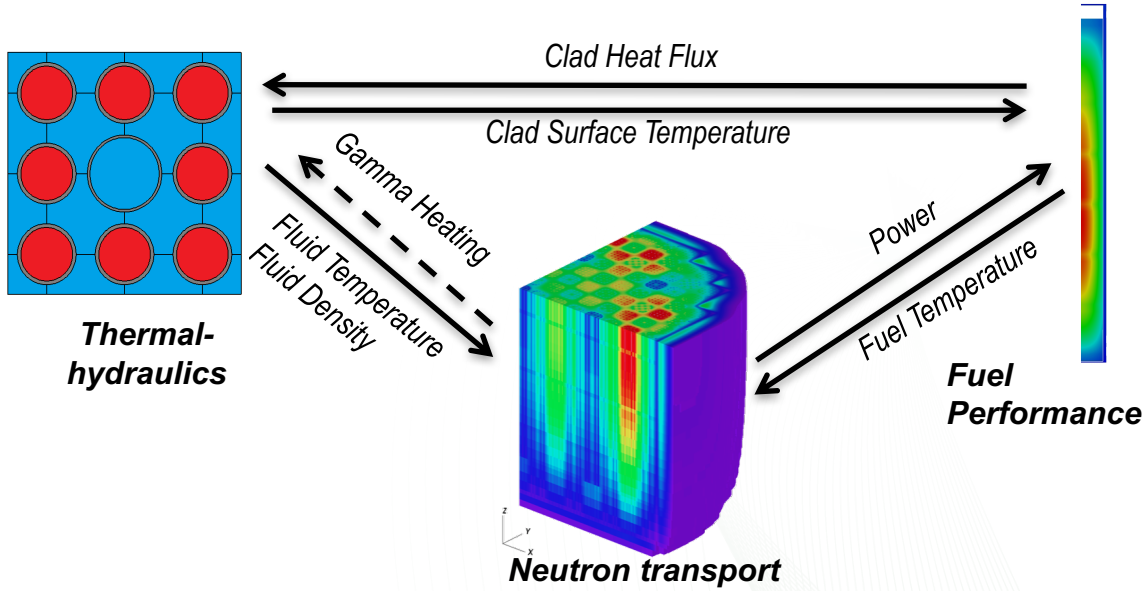
- *Demonstrated three different model applications in VM-based environment*
  - ***Kocoloski, et al., System-Level Support for Composition of Applications, ROSS'15***
  - *Using ADIOS or TCASM as application-level API for composition*
- Crack detection in molecular dynamics simulations
  - LAMMPS molecular dynamics
  - Bonds analytics (calculate bond distances)
- Plasma microturbulence
  - GTC-P plasma microturbulence
  - PreDatA analytics (histogramming of particle properties)
- Neutronics energy spectrum analysis
  - SNAP neutronics proxy application
  - Energy spectrum analysis





Modern Complex Multi-{Physics,Scale,…} Applications

- Computational components representing different types of physics
- Coupled by exchanging boundary conditions, etc.
- Often created by forcing standalone "single-physics" applications to interact
- The mechanics of the coupling (typically) consume vast amounts of human effort
  - The design and development of the standalone apps probably never considered coupling
  - Different representations for the data
  - Integrating the code base itself

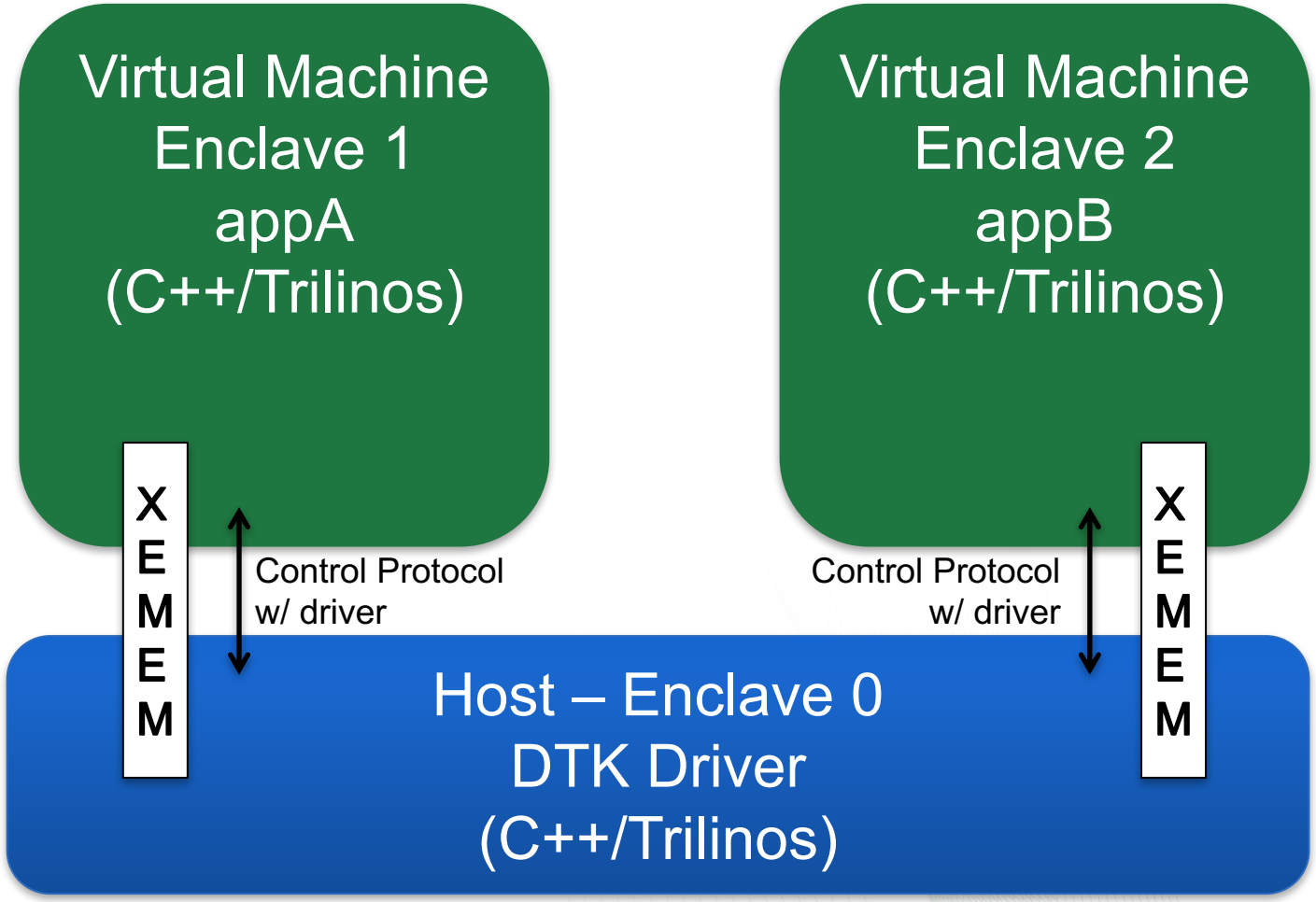


DTK Addresses (Only) One Aspect of the Problem

- Data Transfer Kit is a library for parallel solution transfer in multiphysics and multiscale simulations
  - https://ornl-cees.github.io/DataTransferKit/
- Defines adapters (for mesh libraries, etc.) operators (for transformations), etc.
- Minimal modifications to target components to enable coupling
- **But DTK assumes that the entire application is a single executable**
  - This is often a much larger challenge than instrumenting the code for the coupling
  - Need a common software stack with all conflicting dependencies harmonized
- **Example: CASL spends 3 FTEs (~4% of budget) just on resolving dependency conflicts across their multiphysics components**

Can We Make this Easier?

- What if we could allow each component to use its “natural” dependency stack?
  - Use virtualization to provide necessary isolation between “enclaves”
  - Need ability to exchange memory and (minimal) control between enclaves
- XEMEM – cross-enclave shared memory segments
  - Based on XPMEM
- Simple command queues between enclaves
- Developed in Hobbes Exascale OS/R project

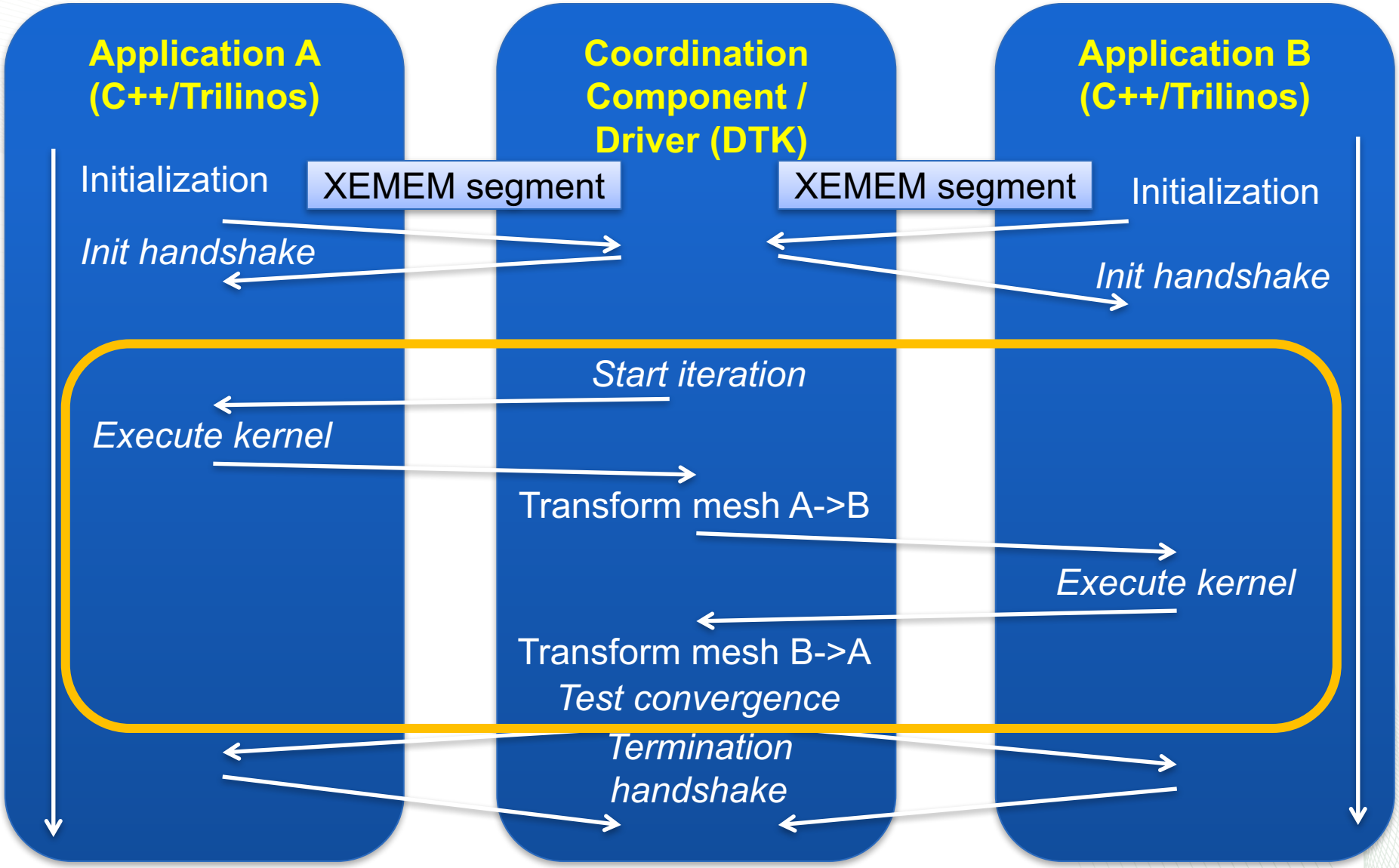


Same basic approach can be used with virtual machines, containers, or processes as the enclaves

Notional Sequence of Operations for Multi-Enclave Sim

Initialization allocates and registers XEMEM shared memory segments for mesh data structures

Main execution loop



Sketch of Code for App Initialization (both appA and appB)

- Hobbes initialization
- Initialization handshake with the driver
- Allocate buffer for the mesh that will be shared with the driver
- Mesh data passed to the driver
- Start execution loop

Some new code required to initialize Hobbes environment

src — gvh@livingstone:~ — vi appA.cpp — 81×78

gvh@livingsto... | gvh@hal9000... | gvh@livin...

```
rc = _setup_hobbes (&db, NULL, NULL);
if (rc != 0)
{
    fprintf (stderr, "ERROR: _setup_hobbes() failed\n");
    goto exit_fn_on_error;
}

/*
 * Setting up 2 comamnd queues for bi-directional commands with the
 * driver.
 */
rc = _app_handshake (app, &hcq_to_driver, &hcq_from_driver);
if (rc == -1)
{
    fprintf (stderr, "ERROR: _app_handshake() failed\n");
    goto exit_fn_on_error;
}

/*
 * We now have 2 cmd queues for bi-directional communication; we finish
 * the initialization of the app
 */
src_coord._export(app);

printf ("Sharing mesh meta-data\n");
_a = (double*)src_coord.data();
meta_data          = (meta_data_t*)_a;
meta_data->layout  = (uint64_t)layout;
meta_data->num     = (uint64_t)src_num;
meta_data->dim     = (uint64_t)space_dim;

printf ("Notifying driver that we are up...\n");
rc = _app_notify_driver (hcq_from_driver);
if (rc == -1)
{
    fprintf (stderr, "ERROR: _app_notify_driver() failed\n");
    goto exit_fn_on_error;
}

printf ("Waiting for the ACK from the driver before we start running...\n");
rc = _app_wait (hcq_to_driver);
if (rc == -1)
{
    fprintf (stderr, "ERROR: _app_wait() failed\n");
    goto exit_fn_on_error;
}

/* We initialize the buffer for now for verification */
{
    for (unsigned i = 0; i < src_num; ++i)
    {
        coord[0] = (double) std::rand() / (double) RAND_MAX + comm_rank;
        coord[1] = (double) std::rand() / (double) RAND_MAX;
        coord[2] = (double) std::rand() / (double) RAND_MAX;
        /* Transfer this into the kernel execution */
        _a[i+0*src_num] = coord[0];
        _a[i+1*src_num] = coord[1];
        _a[i+2*src_num] = coord[2];
    }
}

/*
 * Now we start to execute, letting the driver coordinating everything.
 */
printf ("Starting computation...\n");
for (iter = 0; iter < NB_ITERS; iter++)
{
    rc = appA_iteration (src_coord, hcq_from_driver, hcq_to_driver);
    if (rc != 0)
    {
        fprintf (stderr, "ERROR: appA_iteration() failed\n");
        goto exit_fn_on_error;
    }
}

printf ("All done, finalizaing...\n");
```

Sketch of Code for App Execution Loop (appA and appB similar)

- Computation – kernel using the mesh (unmodified)
- Notify the driver that the computation succeeded
- Wait for the driver

Some new code required to handle control transfers

src — gvh@livingstone:~ — vi appA.cpp — 73×46

gvh@livingstone... | gvh@hal9000: ~... | gvh@livingstone... | gvh@livingstone...

```
#include "dbapi.h"
#include "hobbes_db.h"
#include "demo_constants.h"
#include "demo_app.h"

hdb_db_t hobbes_master_db;

int
appA_iteration (Hobbes::vector<double>  mesh,
                hcq_handle_t            hcq_from_driver,
                hcq_handle_t            hcq_to_driver)
{
    int rc;

    /* We run the computation on the mesh */
    rc = _app_execute((double*)mesh.data());
    if (rc == -1)
    {
        fprintf (stderr, "ERROR: _app_execute() failed\n");
        goto exit_fn_on_error;
    }

    /* Computation is completed, we let the driver know that the
       mesh is ready to be transfered to appA */
    rc = _app_notify_driver (hcq_from_driver);
    if (rc == -1)
    {
        fprintf (stderr, "ERROR: _app_notify_driver() failed\n");
        goto exit_fn_on_error;
    }

    /* We wait a signal from the driver to let us know that the
       mesh can be used again for another iteration. */
    rc = _app_wait (hcq_to_driver);
    if (rc == -1)
    {
        fprintf (stderr, "ERROR: _app_wait() failed\n");
        goto exit_fn_on_error;
    }

    return 0;

 exit_fn_on_error:
    return -1;
}
```

Sketch of Code for DTK Driver

- Wait for appA to finish main computation
- Transform appA mesh to appB
- Signal appB and wait for it to finish main computation
- Transform appB mesh to appA
- Signal appA to run main computation

Some different code required to handle control transfers

MPI_forum — gvh@livingstone:~/projects/hobbes/codes/h...

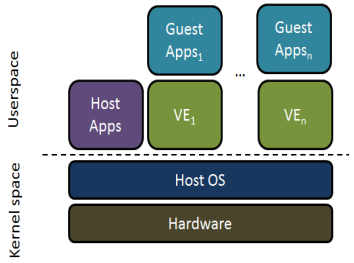
x8r@ranl... | gvh@sal... | root@livi... | gvh@livin...

```
int
driver_iteration (char                      *appA,
                  char                      *appB,
                  Hobbes::vector<double>    appA_vec,
                  Hobbes::vector<double>    appB_vec,
                  DTK_Map                   *dtk_map_appA_to_appB,
                  DTK_Map                   *dtk_map_appB_to_appA,
                  uint64_t                  src_layout,
                  uint64_t                  tgt_layout,
                  uint64_t                  dim,
                  hcq_handle_t              hcq_to_appA,
                  hcq_handle_t              hcq_from_appA,
                  hcq_handle_t              hcq_to_appB,
                  hcq_handle_t              hcq_from_appB)
{
    int     rc;
    void    *appA_buffer;
    void    *appB_buffer;

    appA_buffer = appA_vec.data();
    appB_buffer = appB_vec.data();

    /* We wait for the app to run and signal us so we can pull the data */
    rc = _wait_for_app (appA, hcq_to_appA);
    if (rc == -1)
    {
        fprintf (stderr, "ERROR: _wait_for_app() failed\n");
        goto fn_exit_on_error;
    }

    fprintf (stdout, "Transforming the mesh...\n");
    DTK_Map_apply (dtk_map_appA_to_appB,
                   (double const*)appA_buffer,
                   DTK_INTERLEAVED,
                   (double*)appB_buffer,
                   DTK_INTERLEAVED,
                   (int)dim,
                   false);

    rc = _signal_app (appB, hcq_from_appB);
    if (rc == -1)
    {
        fprintf (stderr, "ERROR: _signal_app() failed\n");
        goto fn_exit_on_error;
    }

    rc = _wait_for_app (appB, hcq_to_appB);
    if (rc == -1)
    {
        fprintf (stderr, "ERROR: _wait_for_app() failed\n");
        goto fn_exit_on_error;
    }

    fprintf (stdout, "Transforming the mesh...\n");
    DTK_Map_apply (dtk_map_appB_to_appA,
                   (double const*)appB_buffer,
                   DTK_BLOCKED, //(DTK_Data_layout)tgt_layout,
                   (double*)appA_buffer,
                   DTK_BLOCKED, //(DTK_Data_layout)src_layout,
                   (int)dim,
                   false);

    rc = _signal_app (appA, hcq_from_appA);
    if (rc == -1)
    {
        fprintf (stderr, "ERROR: _signal_app() failed\n");
        goto fn_exit_on_error;
    }

    return 0;

 fn_exit_on_error:
    return -1;
}
                                                          408,0-1       60%
```

Implementation Details

Processes

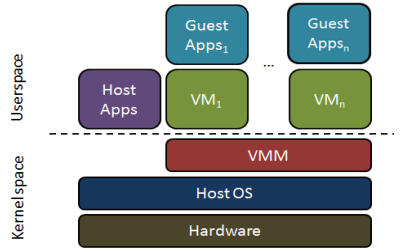
- Operating system
  - Single kernel
- System view
  - Separate memory spaces
- User view
  - Must keep multiple stacks straight
- General remarks
  - Native performance



Containers

- Operating system
  - Single kernel
- System view
  - Isolate resources & context of processes
- User view
  - Ability to customize stack (above kernel)
- General remarks
  - Performance
    - Near native (w/o network isolation via overlays)
  - Examples
    - **Docker**, LXC

Used in current demo →



Virtual Machines

- Operating system
  - Multiple kernels (host/guest)
- System view
  - Virtualize the hardware & control access
- User view
  - Ability to fully customize guest kernel/system
- General remarks
  - Performance
    - 95-98% of native
  - Examples
    - Xen (type-I), KVM (type-II), **Palacios** (type-II)

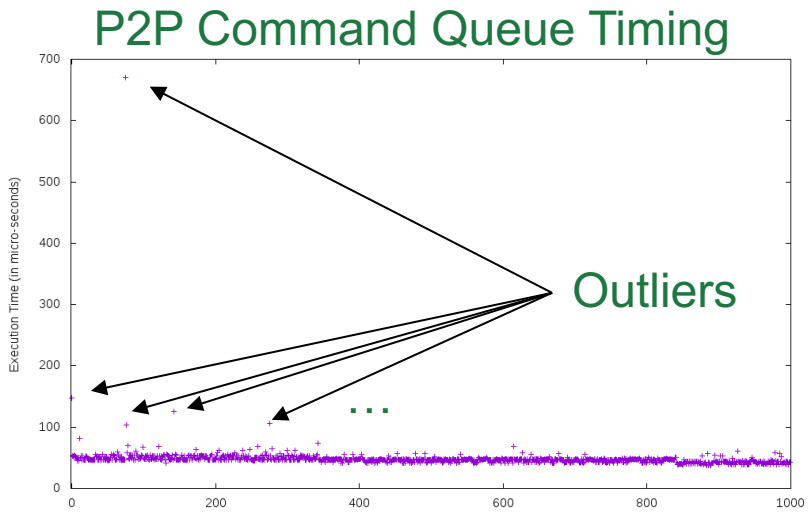
Kevin Pedretti will talk more about the Hobbes environment and in particular the VM-based implementation

Performance Comparison

Measurement	Baseline (µs)	Process (µs)	Container (µs)	Virtual (µs)
Command queue	48.3 ± 20.8	48.3 ± 20.8	37.1 ± 83.0	
appA -> appB mesh transformation	14,080 ± 852	9,986 ± 692	9,400 ± 135	
appB -> appA mesh transformation	17,298 ± 143	13,570 ± 111	13,153 ± 127	
Complete application iteration	31,395 ± 660	25,132 ± 312	23,422 ± 469	

Large standard deviations are due to a small number of outliers. Not yet understood.

Demonstration mesh is very small, so overheads will be accentuated.



Test mesh contains 3,000 3d points.
Timings averaged over 500 iterations (1000 for command queue).
***Host platform**: vintage (2011) 4x 8-core AMD Opteron @ 2.0 GHz, 128 GB RAM, RHEL 7.3*
***Container** uses Docker with Ubuntu 16.04 guests.*
***Virtual** uses Palacios with minimal Busybox guests.*

Next Steps

- Understand (and ameliorate) performance variability
- Extend to MPI parallel
  - (Current demonstration is single node, due to delays in MPI support in Palacios)
  - Should “just work”
- Integrate XEMEM allocation into Trilinos (for example) data structures
  - Could be seen as small extension of what DTK already does, providing “adapters” for many popular mesh libraries
- Demonstrate other composite application scenarios
  - Simulation + analytics done in ROSS’15 paper
  - Coupled multiphysics done here
- Look for use cases where resource partitioning/management is really useful

And now for something completely different….. Reproducibility

Reproducibility Terminology

- **Reviewable Research**: Descriptions of the research methods can be independently assessed and judged credible (may not imply reproducibility)
- **Replicable Research**: Tools are made available that would allow one to duplicate the results of the research (for example, running the author's code to produce the plots in the paper)
- **Confirmable Research**: Main conclusions of the research can be attained independently, without the use of software provided by the author. (But using complete description of algorithms, methodology, etc.)
- **Auditable Research**: Sufficient records have been archived so that research can be defended later, if necessary or differences between independent conformations resolved. Archive may be private
- **Open or Reproducible Research**: Auditable research made openly available

Terminology used by ACM, based on V. Stodden, D. H. Bailey, J. Borwein, R. J. LeVesque, W. Rider, and W. Stein. 2013. Setting the Default to Reproducible: Reproducibility in Computational and Experimental Mathematics. (2013). https://icerm.brown.edu/tw12-5-rcem/icerm_report.pdf

Virtualization and Reproducibility Thought Experiment

- What does it take to “reproduce” an application?
  - Dependent libraries
  - Compilers? (code generation details)
  - System software? (dependencies not in the form of libraries)
  - OS kernel?
  - Processor and other hardware?
- Containers and virtual machines can encapsulate some or all of these
  - VMs could even allow hardware to be emulated
- This is not an original idea – much prior experimentation
- But the devil is in the details, and this is not yet routine

Conclusions

- For HPC, virtualization techniques are primarily useful as a means to facilitate **packaging, distribution, and deployment** of complex computational science and engineering applications
  - Encapsulate and isolate dependencies
  - Eliminate the need for every user to build
- **Reproducibility** is an increasingly prominent concern that virtualization can help
- **Application composition** is another
  - **Single model and API supports process-, container-, or VM-based composition**
    - Uses only shared memory segments (XEMEM) and command queues
  - Can be used with minimal intrusion into code
  - High-performance virtualization techniques mean minimal performance impact
  - Complements tools like Data Transfer Kit (DTK) which focus on coupling/exchange of data at application level