So Why Can't I Checkpoint That?

Gene Cooperman*
gene@ccs.neu.edu

Khoury College of Computer Sciences
Northeastern University, Boston, USA

February 5, 2021
(Keynote talk: SuperCheck'21)

*Partially supported by NSF Grant OAC-1740218, and by grants from Intel Corporation, MemVerge, NERSC and Raytheon.

Table of Contents

1. A DMTCPer's View of the World
2. Challenge: Checkpointing the Hardware
3. Challenges for the Future of Checkpointing: Boundaries and Plumbing

Outline

1. A DMTCPer's View of the World
2. Challenge: Checkpointing the Hardware
3. Challenges for the Future of Checkpointing: Boundaries and Plumbing

DMTCP: Distributed MultiThreaded CheckPointing

The speaker has led the DMTCP project for 15 years:

The DMTCP project is currently supported by:

*

CAVEAT: The remaining slides attempt to present a broader view of some of the highlights in the history of checkpointing, as informed by the history of DMTCP. Any such undertaking is dangerous, due to accidental omissions of important past results. The speaker apologizes in advance for such omissions.

*Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

Brief History of Transparent Checkpointing in Supercomputing

- `https://checkpointing.org/` : 19 checkpointing packages listed from before 2010, including Condor's landmark checkpointing
- 2006: BLCR: `https://crd.lbl.gov/departments/computer-science/class/research/past-projects/BLCR/` ; "Berkeley Lab Checkpoint/Restart (BLCR) for Linux Clusters", J. of Physics: Conf. Ser. **46**(067), 2006 (forerunner as tech. report in 2003)
- And several MPI packages incorporated BLCR to checkpoint MPI: MPICH, MVAPICH, Open MPI
- 2006: MVAPICH (incorporating BLCR): Gao et al., "Application-Transparent Checkpoint/Restart for MPI Programs over InfiniBand", Int. Conf. on Parallel Processing (ICPP'06) (team of DK Panda)

Brief History of Transparent Checkpointing in Supercomputing (cont.)

- 2009: DMTCP: Ansel et al., "DMTCP: Transparent Checkpointing for Cluster Computations and the Desktop", Int. Parallel and Distributed Processing Symp. (IPDPS'09) (forerunner by Rieker et al., PDPTA'06) (team of G. Cooperman)
- 2009: Open MPI: (incorporating BLCR): `https://www.open-mpi.org/faq/?category=ft#cr-support` (based on Hursey et al., "Interconnect agnostic checkpoint/restart in Open MPI", *Proc. of ACM Int. Symp. on High Performance Distributed Computing* (HPDC'09))

Some Highlights of Application-Specific Checkpointing in Supercomputing

- 2003: Stewart and Edwards, "The SIERRA Framework for Developing Advanced Parallel Mechanics Applications", *Large-Scale PDE-Constrained Optimization*, LNCSE **30** (2003) (checkpointing added later) *(and note, in general save-workspace features of Matlab and other very high-level languages)*
- 2003: Bronevetsky et al., "Automated Application-level Checkpointing of MPI Programs", Proc. of Ninth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'03) (based on: CCIFT (Cornell Compiler for Inserting Fault-Tolerance))
- 2013: Zheng et al., "FlexIO: I/O Middleware for Location-Flexible Scientific Data Analytics", Int. Parallel and Distributed Processing Symp. (IPDPS'13) (team of 12 co-authors)

Some Highlights of Application-Specific Checkpointing in Supercomputing (cont.)

- 2016: Di and Cappello, "Fast Error-Bounded Lossy HPC Data Compression with SZ", IEEE Int. Parallel and Distributed Processing Symp. (IPDPS'16)
- 2019: CRAFT (supporting **SCR** and **ULFM**): "CRAFT: A Library for Easier Application-Level Checkpoint/Restart and Automatic Fault Tolerance", IEEE Trans. on Parallel and Distributed Systems **30(3)** (2019)
- 2019: VeloC: Bogdan et al., "Veloc: Towards High Performance Adaptive Asynchronous Checkpointing at Large Scale", IEEE Int. Parallel and Distributed Processing Symp. (IPDPS'19)

Outline

1. A DMTCPer's View of the World
2. Challenge: Checkpointing the Hardware
3. Challenges for the Future of Checkpointing: Boundaries and Plumbing

Raising the Sights for DMTCP (and other projects)

In the earliest days of the DMTCP project for transparent checkpointing, we checkpointed a single process on a single computer on a single computer, and we announced that we're done.
"Once you try to checkpoint the network, you no longer have a closed system. Any process can reach out to anywhere on the Internet. We're done now."
OUR EARLY SLOGAN (We don't use it much nowadays.): "You can't checkpoint the world!"

But then:
"Well, we can try to include MPI on a cluster, but you have to promise that it will just be the process and TCP/IP. So, we're done now."

Raising the Sights for DMTCP (and other projects): cont.

But then:
"Okay, we can try to checkpoint InfiniBand. It's a network interconnect. But you have to promise that it will just be the process and the network interconnect. So, we're done now."

But then:
"Okay, some of those nodes are using CUDA with GPU accelerators. And it's sort of a co-processor to the CPU. But you have to promise that it will just be the process, the network interconnect, and any co-processors. So, we're done now."

But Why Can't I Also Checkpoint the Hardware?

So far, the history so far has been one of meeting the needs of scaling up: more CPU cores; bigger and better network interconnects; etc.
But with the rising adoption of compute accelerators, we *also* need to grow in a different direction.

- VNC trick (It's been with us almost since the days of the cavemen. As soon as VNC was invented, some nameless person from the depths of history must have asked, "Can I use that for checkpointing")
  [Don't worry. We'll explain the VNC trick later.]
- CUDA for GPUs
- Tensor Processing Units (TPU), AI chips, FPGAs, big data (Hadoop and Spark), future networks on a chip(?), etc.

But Why Can't I Also Checkpoint the Hardware? (cont.)

- Cray GNI network interconnect: support for HPC interconnects other than InfiniBand (see talk in this Symposium on MANA: Chouhan et al.)
- OpenGL on GPUs: support for OpenGL for rendering farms for CGI in Hollywood (see talk on OpenGL: Hou (Li presenting))
- possible hybrid of application-specific and transparent checkpointing (see talk in this Symposium on VeloC: Bogdan et al.)

Outline

1. A DMTCPer's View of the World
2. Challenge: Checkpointing the Hardware
3. Challenges for the Future of Checkpointing: Boundaries and Plumbing

Challenges for the Future of Checkpointing: Boundaries and Plumbing

Here, I hypothesize the central challenge for the future of checkpointing.

> *THESIS: Checkpointing the hardware is difficult. In the past, SOLUTION 1, below, was popular. In the future, SOLUTION 2 will be more popular.*

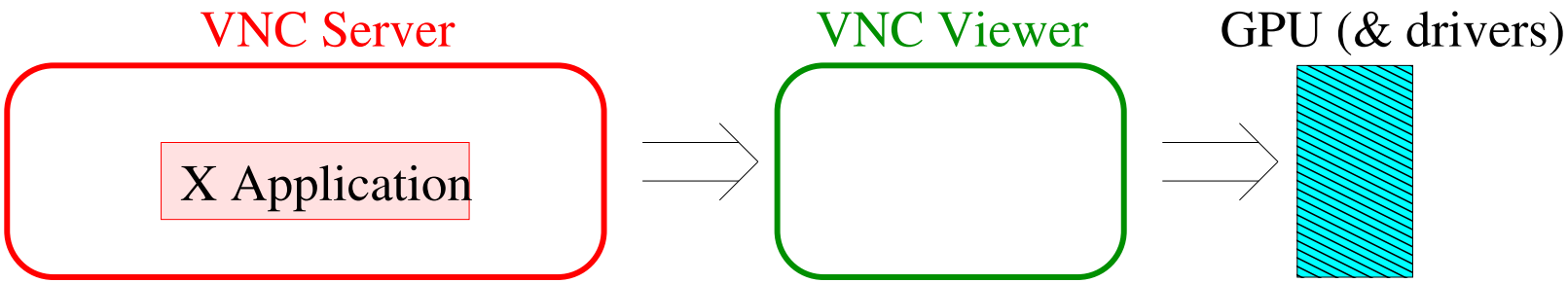
(Constructive criticism are welcome.)

1. **SOLUTION 1:** If something is difficult to checkpoint, then disconnect it, checkpoint what's left over, and then reconnect it.
2. **SOLUTION 2:** If something is difficult to checkpoint, then isolate it as a separate proxy process, separate address space, or whatever.
   Advantage: Checkpoint *without disconnecting*

Also, see an early inspiration for boundaries from process virtualization: Kapil Arya, *User-Space Process Virtualization in the Context of Checkpoint-Restart and Virtual Machines*, PhD thesis, Northeastern U., 2014 (Section 1.3: "Process Virtualization")

SOLUTION 1: The VNC Trick

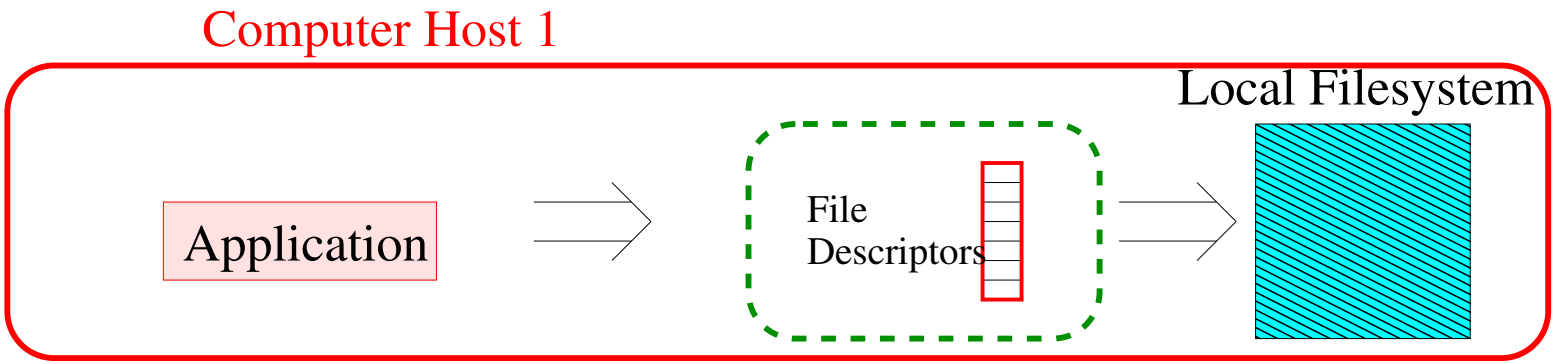
1. **vncserver :1**
2. **Start X Application**
3. **vncviewer localhost:1**



Kill VNC viewer:

Checkpoint
Restore:vncviewer localhost:1 # This recreates the VNC viewer.

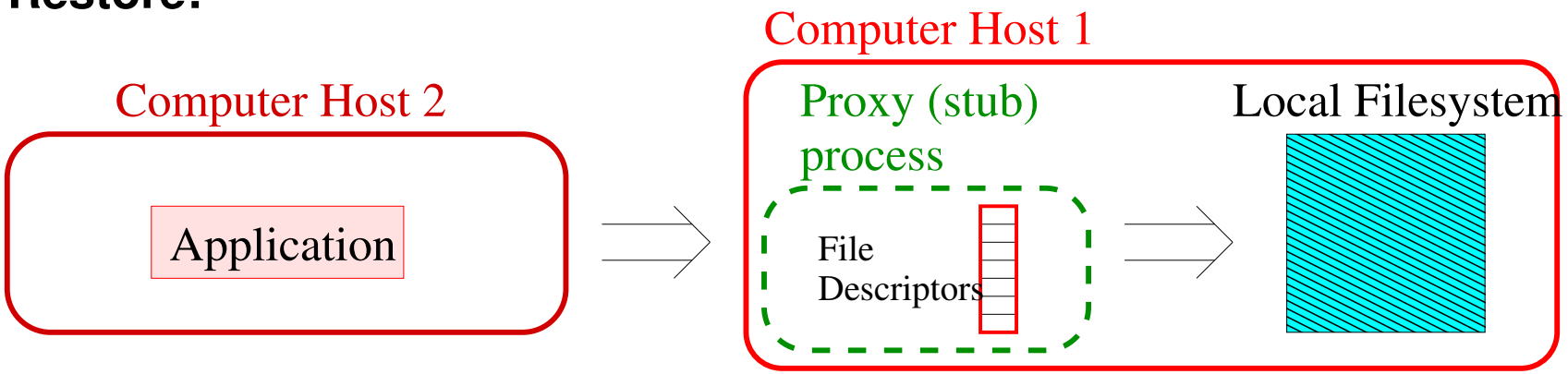
SOLUTION 2: The Condor Trick



Checkpoint

Migrate Application from Host 1 to Host 2

Restore:



SOLUTION 2: The VMGL Trick

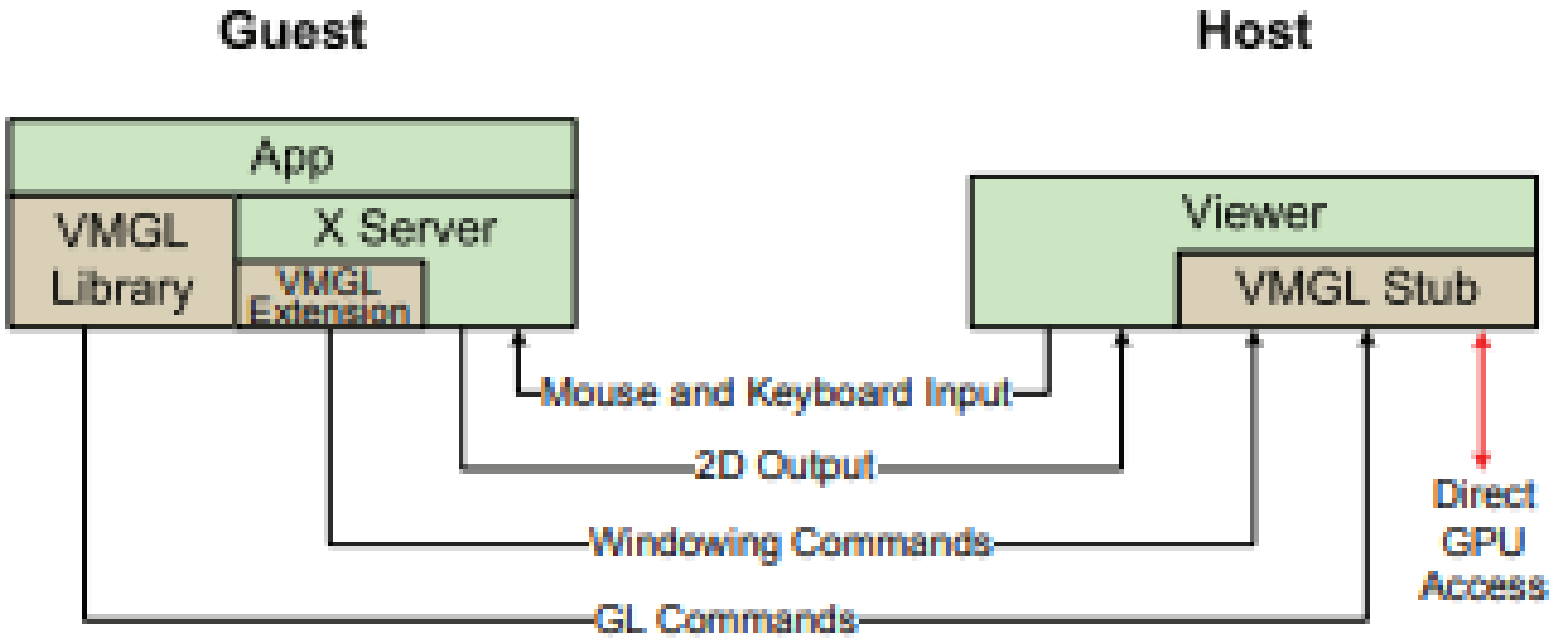


Figure 1: VMGL Architecture

(Lagar-Cavilla et al., "VMM-Independent Graphics Acceleration", VEE'07)

SOLUTION 1: Notable Examples

1. If something is difficult to checkpoint, then disconnect it, checkpoint what's left over, and then reconnect it.
   - VNC (disconnect the VNC server, checkpoint, and reconnect);
   - 2006: MVAPICH (Gao et al.) (disconnect the InfiniBand network, checkpoint, and reconnect)
   - 2009: Open MPI (Hursey et al.) (disconnect the current network (TCP, InfiniBand, ...), checkpoint, and reconnect with a new network)

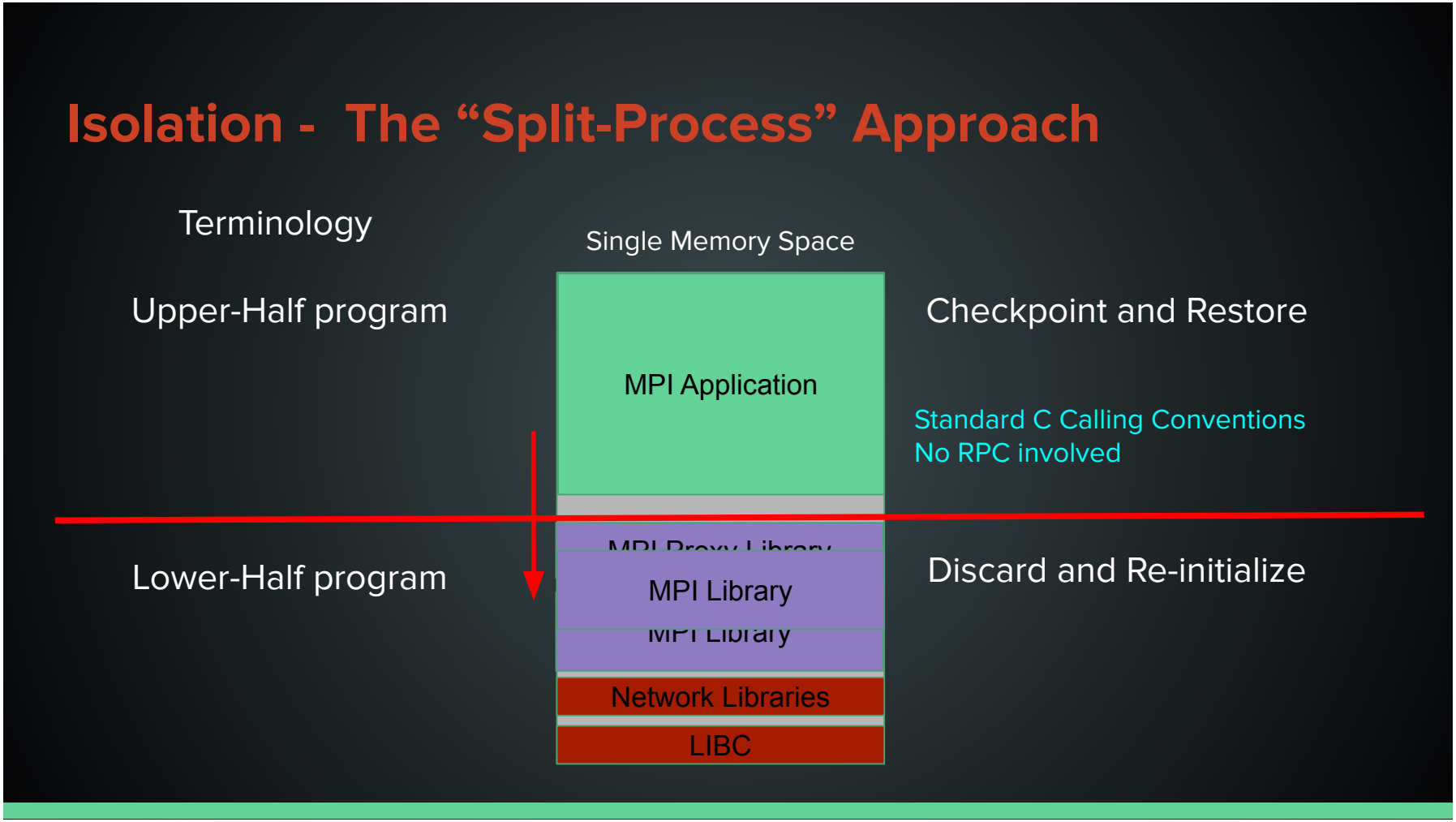
SOLUTION 2: Notable Examples

2. If something is difficult to checkpoint, then isolate it as a separate proxy process, separate address space, or whatever.
   Advantage: Checkpoint *without disconnecting*
   - 1999: Condor: Zandy et al., "Process Hijacking", HPDC'99 Checkpoint on one host, but restart on another host: stub process on original host remains available for local files
   - 2007: OpenGL: Lagar-Cavilla et al., VMGL: "VMM-independent Graphics Acceleration", VEE'07
   - 2011: OpenCL for GPUs: Takizawa et al., "CheCL: Transparent Checkpointing and Process Migration of OpenCL Applications", IPDPS'11
   - 2018: CUDA for GPUs: Garg et al., "CRUM: Checkpoint-Restart Support for CUDA's Unified Memory", Cluster'18
   - 2019: MPI: Garg et al., "MANA for MPI: MPI-Agnostic Network-Agnostic Transparent Checkpointing", HPDC'19
   - 2020: CUDA for GPUs: Jain et al., "CRAC: Checkpoint-Restart Architecture for CUDA with Streams and UVM", SC'20
   - 2021: ?? OpenGL ?? (collaboration between MemVerge and DMTCP)

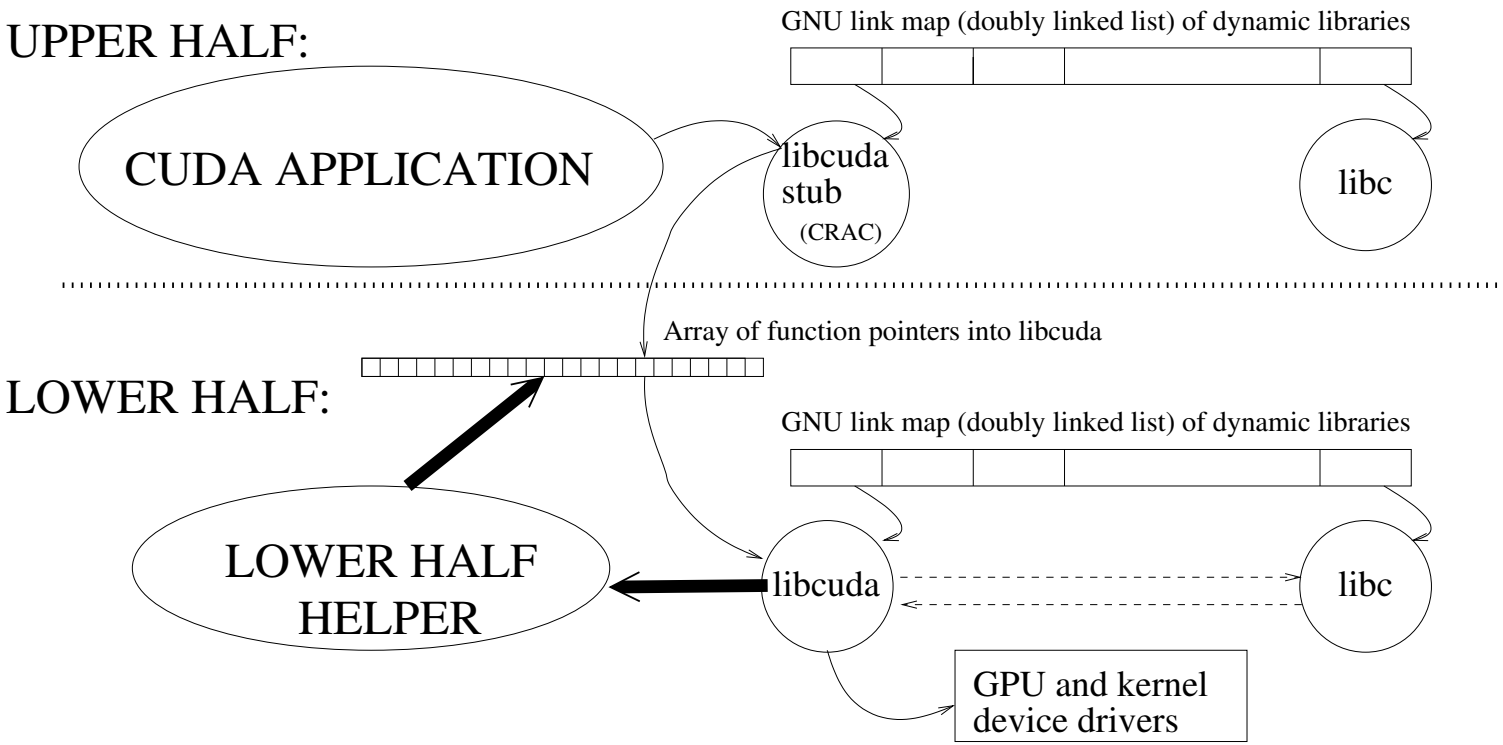
SOLUTION 2: Proxies, Split Processes and User-Space Kernel Loaders

- **SOLUTION 2a (Proxies)**
  - 1999: Condor: Zandy et al., "Process Hijacking", HPDC'99
  - 2007: VMGL: Lagar-Cavilla et al., "VMM-independent Graphics Acceleration", VEE'07
  - 2011: OpenCL for GPUs: Takizawa et al., "CheCL: Transparent Checkpointing and Process Migration of OpenCL Applications", IPDPS'11
  - 2018: CUDA for GPUs: Garg et al., "CRUM: Checkpoint-Restart Support for CUDA's Unified Memory", Cluster'18
- **SOLUTION 2b (Split Processes)**
  - 2019: MPI: Garg et al., "MANA for MPI: MPI-Agnostic Network-Agnostic Transparent Checkpointing", HPDC'19
  - 2020: CUDA for GPUs: Jain et al., "CRAC: Checkpoint-Restart Architecture for CUDA with Streams and UVM", SC'20
  - 2021: ?? OpenGL ?? (collaboration between MemVerge and DMTCP)

SOLUTION 2b: The "MANA for MPI" Trick



SOLUTION 2b: Split Processes: "CRAC for CUDA"



(from Jain et al., "CRAC: Checkpoint-Restart Architecture for CUDA with Streams and UVM", SC'20)

SOLUTION 2b: Split Processes: the User-Space Kernel Loader

In both MANA for MPI and CRAC for CUDA, an essential ingredient is our user-space kernel loader. We need to load **two programs** into a **single address space** (for a single process — and maybe with just one thread).

1. Launch the application
   - Step 1: Launch a small lower-half program as a stub or helper process, but linked to all libraries to access the kernel drivers and the hardware.
   - Step 2: The initial process (running lower half) calls our **kernel loader** to load the end-user application as the upper half in the single address space.
   - Step 3: On checkpoint save only the upper half.
2. Restart the application
   - Step 1: Launch the small lower-half program as before.
   - Step 2: The initial process (running lower half) calls a **restart routine** to load ckpt image file as the restored upper half of the single address space.

Detail: On Intel x86-64 CPUs only, the upper half and lower half inherit different values of the Intel FS register. Setting the FS register in Linux is a privileged operation requiring a system call (and associated overhead). This restriction is to be removed with the FSGSBASE Linux patch. (v13 of this patch is now under review.)

Conclusion: Boundaries and Plumbing

In this tour of checkpointing, we have seen a recurring theme:

Boundaries and Plumbing

1. **"Newer and different hardware in supercomputing is inevitable."** But we can't checkpoint the hardware. (Exception: some hardware has save- and restore-machine-state, for debugging. We can leverage that!)
2. **"Boundaries are good."** They imply that we don't need to checkpoint what's on the other side of the boundary. Boundaries are even better when they approximate a well-established standard. (In SOLUTION 1 (proxies), recall Condor's use of the POSIX API to the C runtime library, VMGL's use of the OpenGL standard, CRUM's use of the CUDA de facto standard.)
3. **"And Boundaries are bad."** Crossing the boundary can impose high overhead. If it's a proxy process (SOLUTION 1), then we must often copy large buffers from the application process to a proxy process.
4. **"The cure for larger boundaries is better plumbing."** In SOLUTION 2 (split processes), the buffer is *not* copied. Only a simple pointer needs to be passed across the boundary.

Checkpoints 'R' Us: We in Supercomputing know Ckpting; Who Else Can Use Our Technology?

1. **Big data:** The HFS (Hadoop FileSystem) or GFS (Google FileSystem) is the boundary. But we can save our large intermediate files at checkpoint time, and return to them during restart.
2. **Fuzzing in cyber-security:** extend fuzzing deeper into the execution; return to interesting intermediate states
3. **Model checking for formal verification:** Return to interesting intermediate states for deeper exploration
4. **"Deep Debugging"** (of MPI, threads, distributed applications): Run deeper into the execution, while taking periodic checkpoints; return to last checkpoint before a crash
5. **Higher quality software:** Checkpoint periodically; On a crash, ask user's permission to nclude the last checkpoint in the bug report (not just the state information at the time of the crash).

Moral: A checkpoint can be a first-class object. Let's use it.

Thank you

Thank you.