Implement Checkpointing for Android

(to speed up boot time and development process)

Jim Huang (黃敬群) <jserv@0xlab.org>

Kito Cheng (程皇嘉) <kito@0xlab.org>

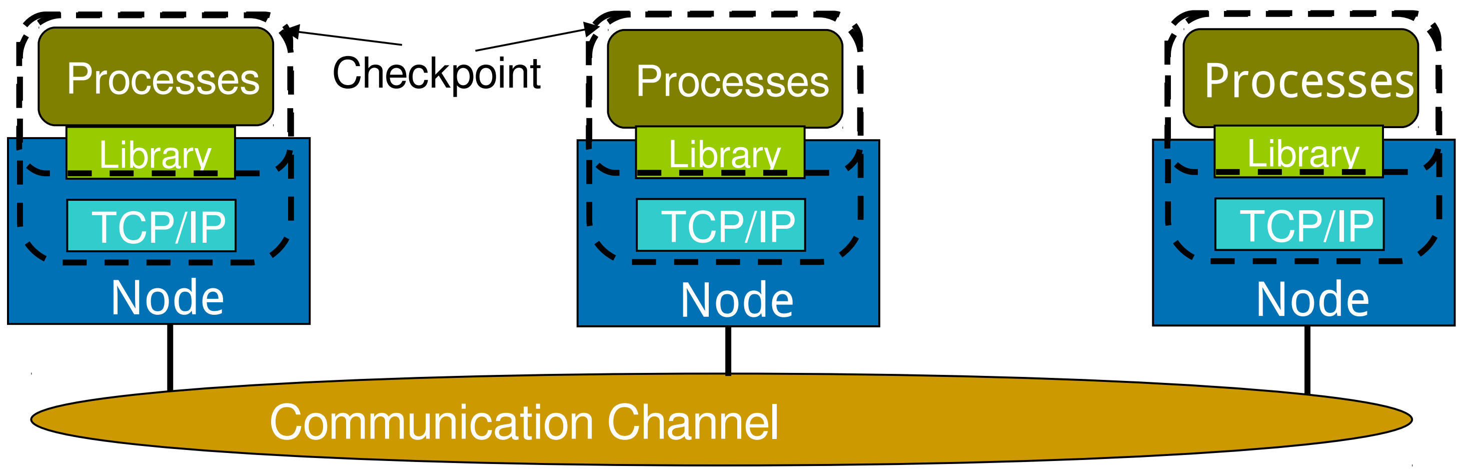
Developer, 0xlab

Nov 5, 2012 ! Embedded Linux Conference Europe

Rights to copy

Basic Idea: Process Migration

- Process Migration (in past applications)
  - Distributed Load Balancing
  - Efficient Resource Utilization
- Crash Recovery and Rollback Transaction
  - Useful to system admin



Checkpointing

- From Wikipedia:
  - … is a technique for inserting fault tolerance into computing systems. It basically consists of storing a snapshot of the current application state, and later on, use it for restarting the execution in case of failure.
- Used in distributed shared memory systems
- Even used in reversible debugger
- Different from virtual machine level snapshot/resume mechanisms
  - Checkpointing emphasizes on process level.

Ideas about Checkpointing for Android

- Resume to stored state for faster Android boot time
- Better product field trial experience due to regular checkpointing
- Deploy problematic states for engineering analysis and debugging transparently
- Q&A stress test purpose

Expectations of Checkpointing

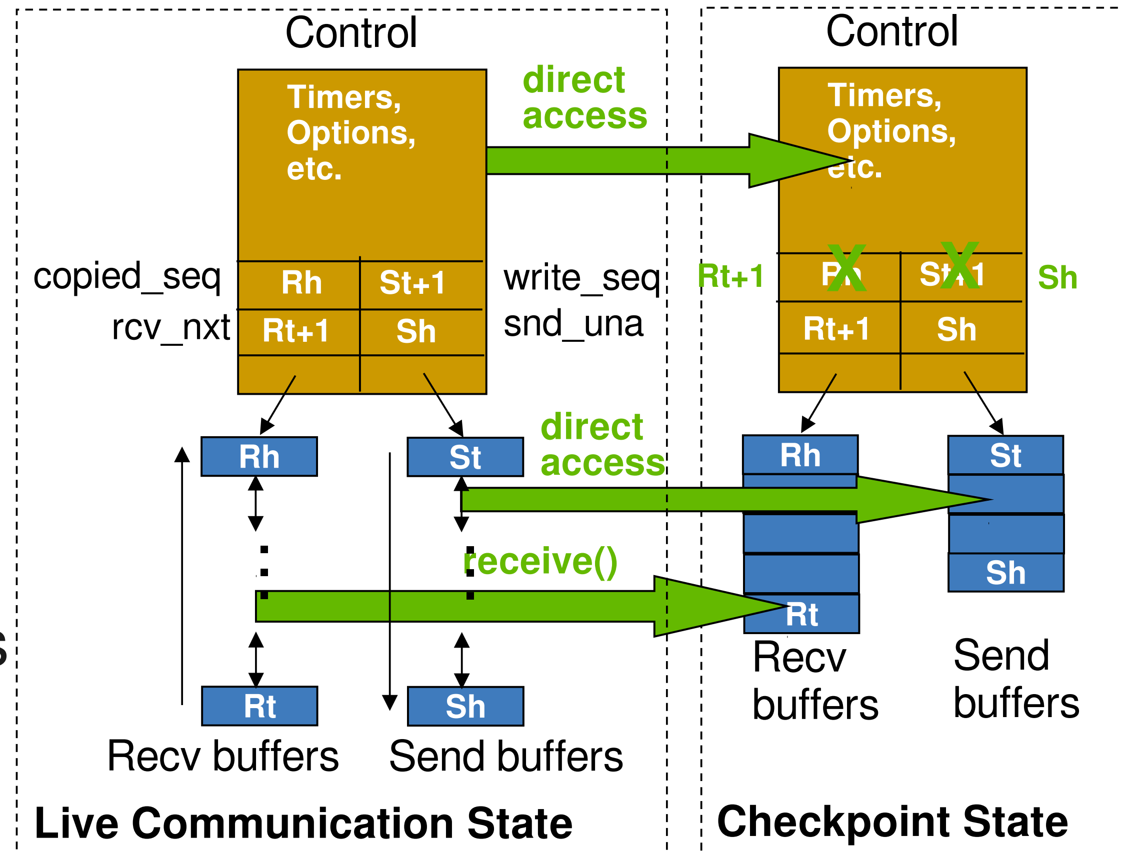
- Application-transparent
  - supports applications without modifications or recompilation
- Supports a broad class of applications
  - Databases
  - parallel / MPI apps
  - desktop apps
- Comprehensive support for user-level state, kernel-level state, and distributed computation and communication state
- Supported on unmodified Linux kernel
  - checkpoint-restart should be integrated by addons

Challenges in checkpoint and restore

- Network stack will continue to execute even after application processes are stopped
- No system call interface to read or write control state
- No system call interface to read send socket buffers
- No system call interface to write receive socket buffers
- Consistency of control state and socket buffer state

Communication state checkpoint

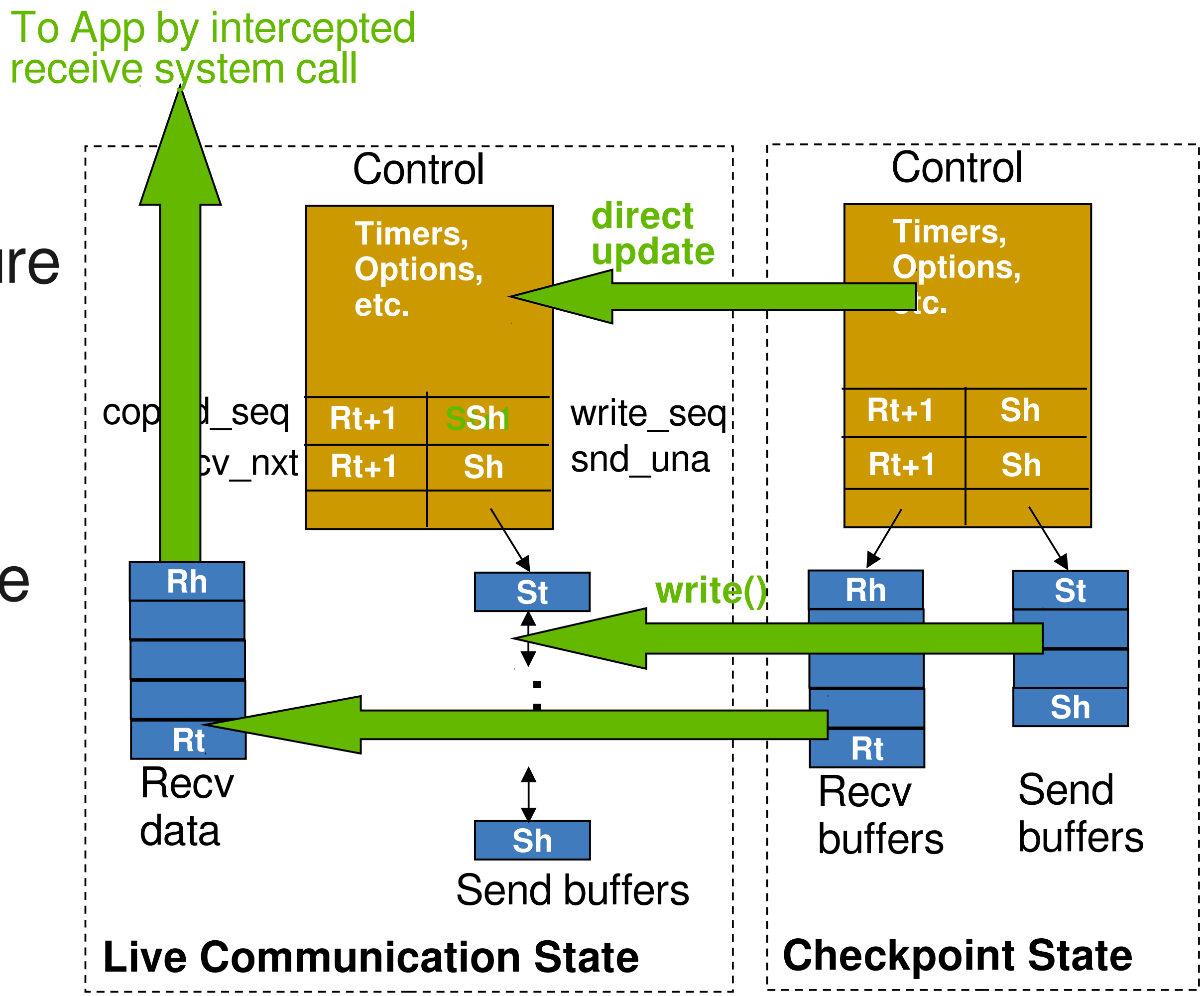
- Acquire network stack locks to freeze TCP processing
- Save receive buffers using socket receive system call in peek mode
- Save send buffers by walking kernel structures
- Copy control state from kernel structures
- Modify two sequence numbers in saved state to reflect empty socket buffers



State for one socket

Communication state restart

- Create a new socket
- Copy control state in checkpoint to socket structure
- Restore checkpointed send buffer data using the socket write call
- Deliver checkpointed receive buffer data to application on demand



State for one socket

Existing Checkpointing mechanisms

- CryoPID
  – http://cryopid.berlios.de/
- BLCR (Berkeley Lab Checkpoint/Restart)
  – https://ftg.lbl.gov/projects/CheckpointRestart/
- DMTCP
  – http://dmtcp.sourceforge.net/

Implementation Considerations

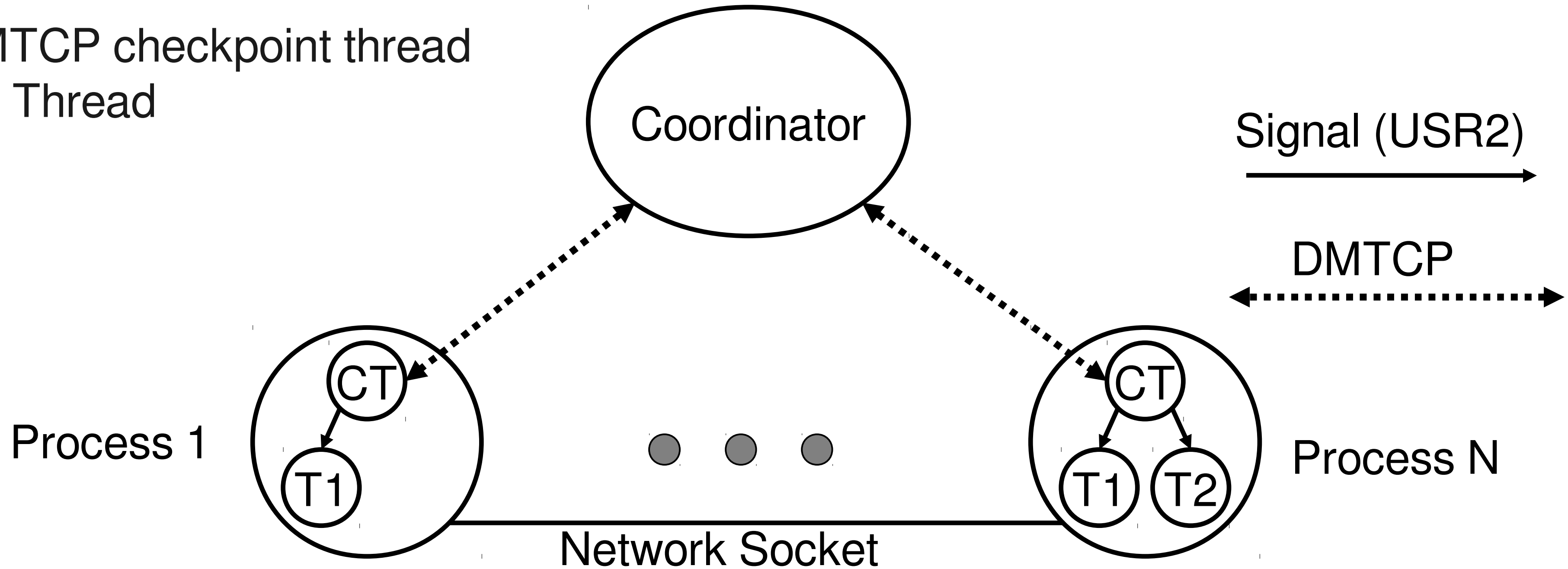
- Checkpointing can be implemented in
  - kernel modifications + helpers in userspace
  - pure userspace
- Introduce a virtualization layer groups processes into specific states with private virtual name space
  - Intercepts system calls to expose only virtual identifiers (e.g., vpid)
  - Preserves resource names and dependencies across migration
- Mechanism to checkpoint and restart states
  - User and kernel-level state
  - Primarily uses system call handlers
  - File system not saved or restored

DMTCP

- **D**istributed **M**ulti-**T**hreaded **C**heck**P**ointing.
- Works with Linux Kernel 2.6.9 and later.
- Supports sequential and multi-threaded computations across single/multiple hosts.
- Entirely in user space (no kernel modules or root privilege).
  - Transparent (no recompiling, no re-linking).
- Written in Northeastern University and MIT and under active development since 2006.
- License: GNU LGPL (allows freely using and linking)

Process Structure

CT = DMTCP checkpoint thread
T = User Thread

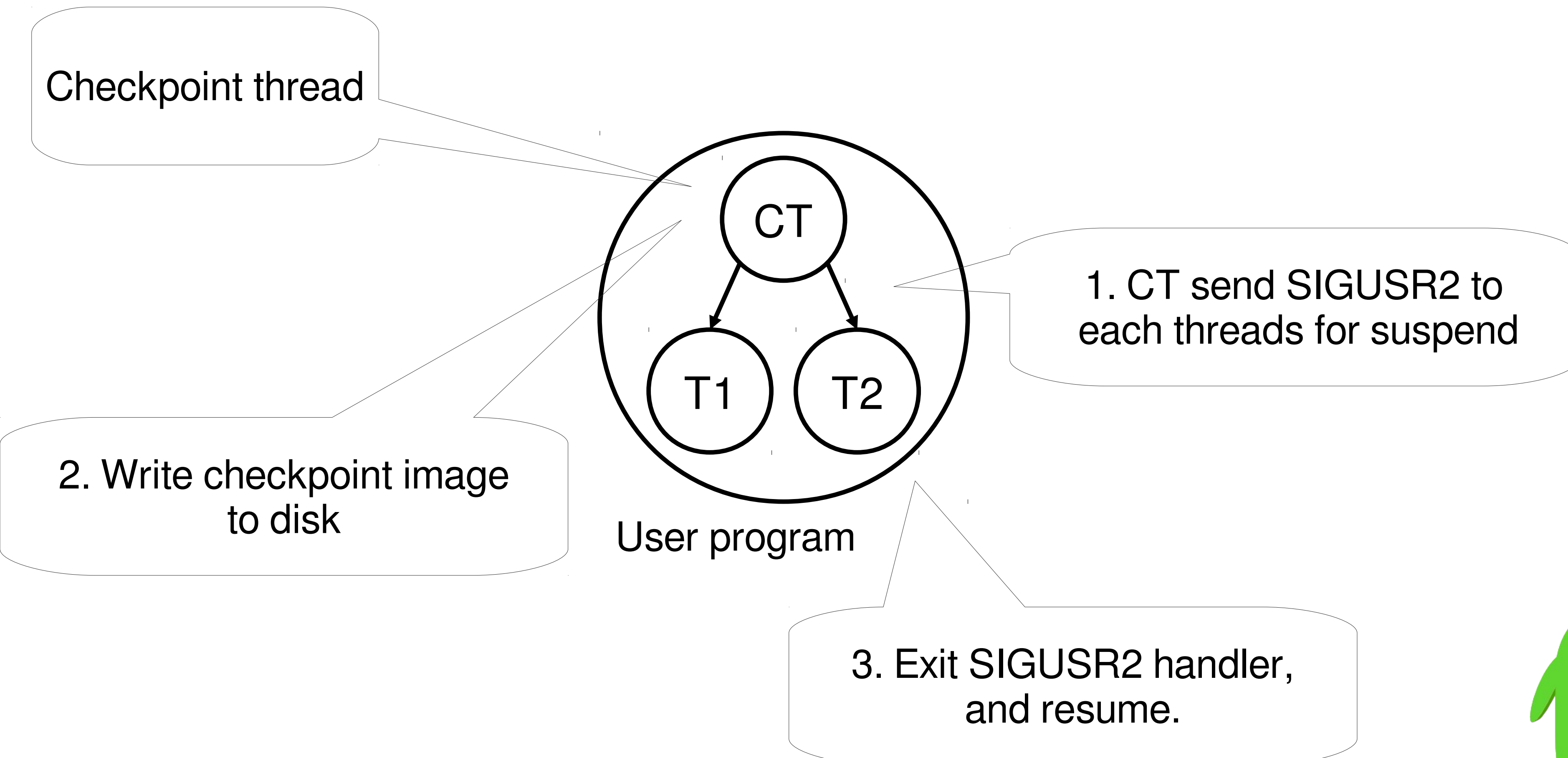



```
dmtcp_checkpoint <EXE> # starts coordinator
dmtcp_command -c       # talks to coordinator
dmtcp_restart ckpt_<EXE>-*.dmtcp
```

- Coordinator: a stateless synchronization server for the distributed checkpointing algorithm.
- Checkpoint/Restart performance related to size of memory, disk write speed, and synchronization.

How DMTCP works (1/4)

- MTCP : component for checkpoint single-process
- SIGUSR2: Used internally from checkpoint thread to user threads.



How DMTCP works (2/4)

- LD_PRELOAD: Transparently preloads checkpoint libraries `dmtcphijack.so` which installs libc wrappers and checkpointing code.
- Wrappers: Only on less heavily used calls to libc
  - `open, fork, exec, system, pipe, bind, listen, setsockopt, connect, accept, clone, close, ptsname, openlog, closelog, signal, sigaction, sigvec, sigblock, sigsetmask, sigprocmask, rt_sigprocmask, pthread_sigmask`
  - Overhead is negligible.

```
...
fd = open(path,
         flags);
...
```

User Program

```
int open(const char *path,
         int flags){
...
funcs[_open](path, flags);
...
}
```

dmtcphijack.so

```
int open(const char *path,
         int flags){
...
}
```

libc.so

How DMTCP works (3/4)

- Additional wrappers when process id & thread id virtualization is enabled
  - `getpid, getppid, gettid, tcgetpgrp, tcsetprgrp, getgrp, setpgrp, getsid, setsid, kill, tkill, tgkill, wait, waitpid, waitid, wait3, wait4`

```
...
pid = getpid();
...
```

User Program

```
int getpid(){
...
real_pid = funcs[_getpid]();
return pid_table[real_pid];
}
```

dmtcphijack.so

```
int getpid(){
...
}
```

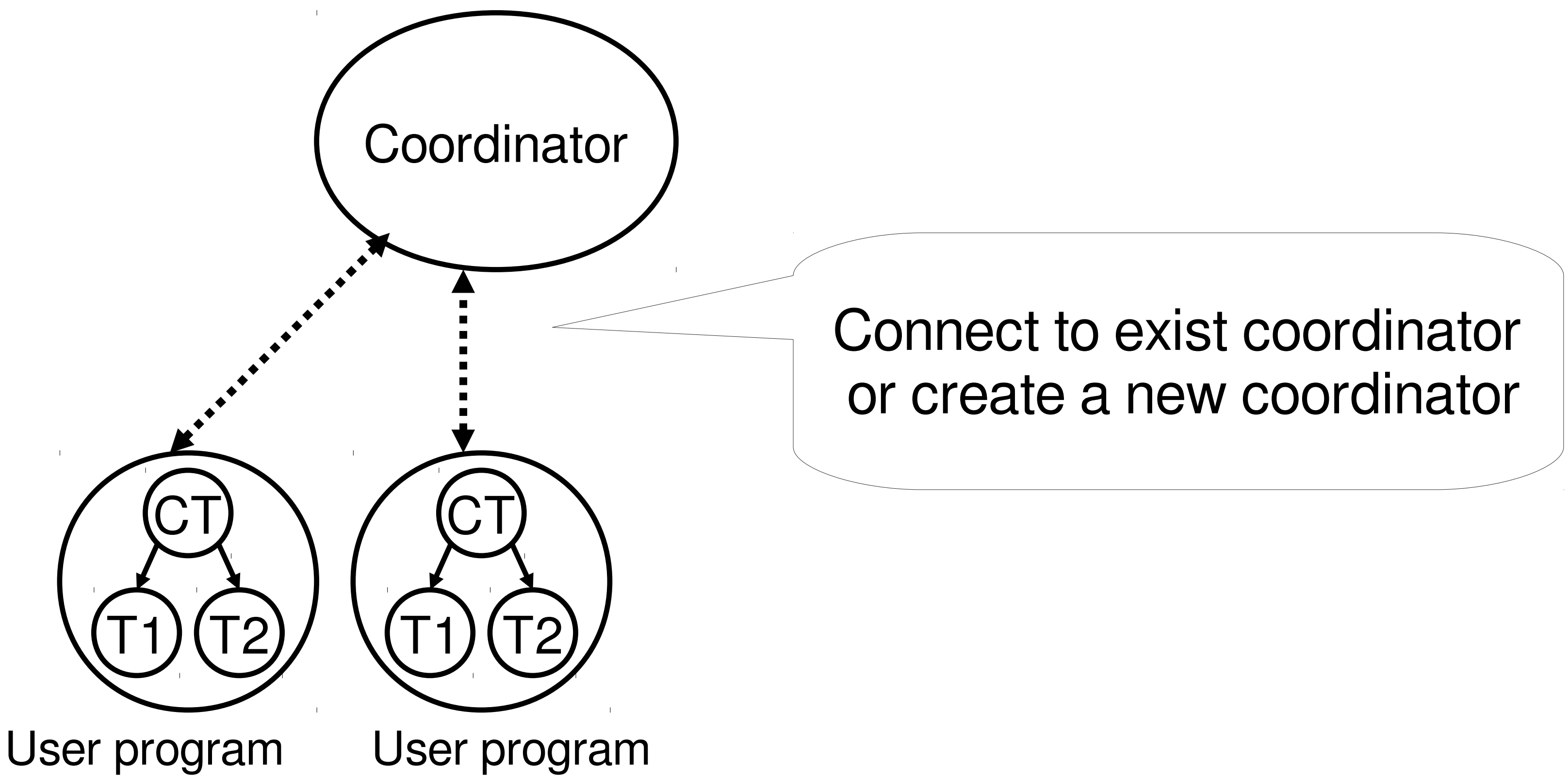
libc.so

How DMTCP works (4/4)

- Checkpoint image compression on-the-fly (default).
- Currently only supports dynamically linking to libc.so. Support for static libc.a is feasible, but not implemented.

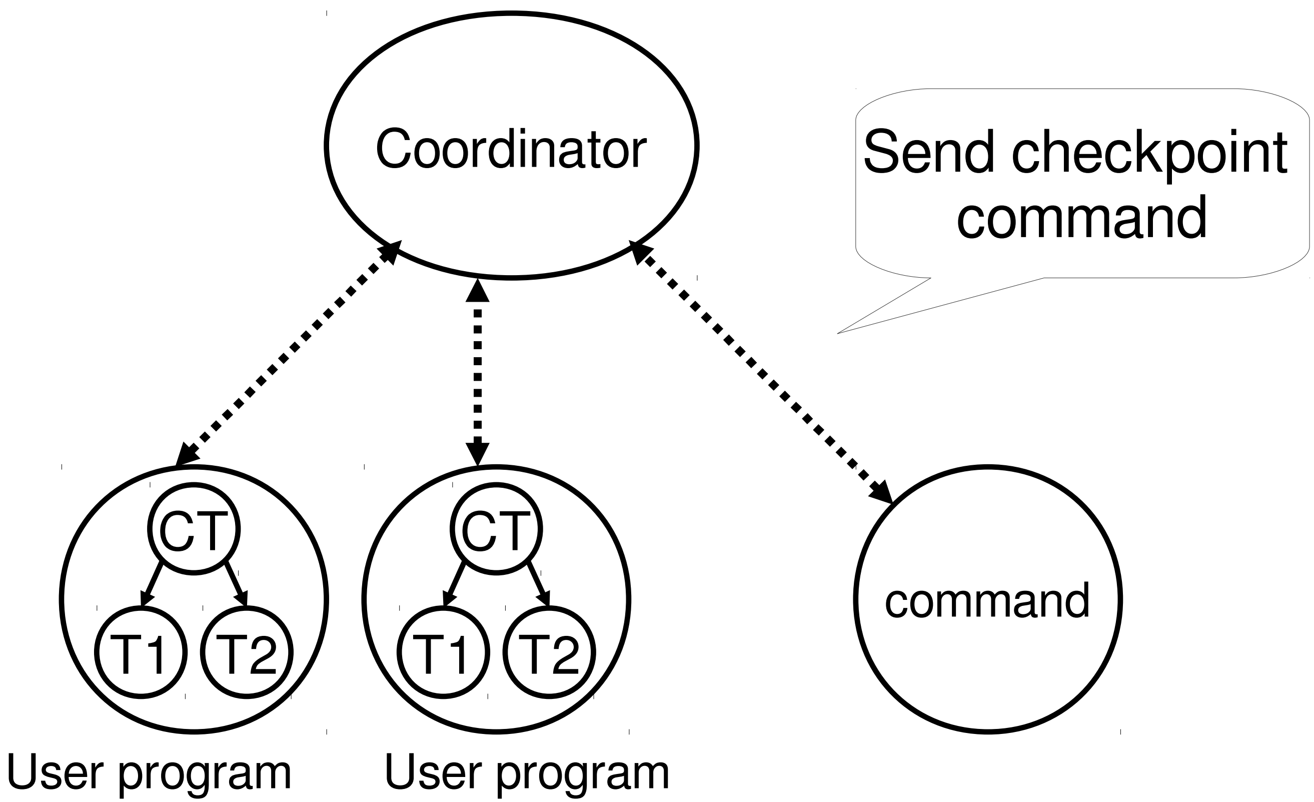
Checkpoint under DMTCP(1/7)

- dmtcphijack.so and libmtcp.so present in executable's memory.
  - **dmtcp_checkpoint** <EXE>



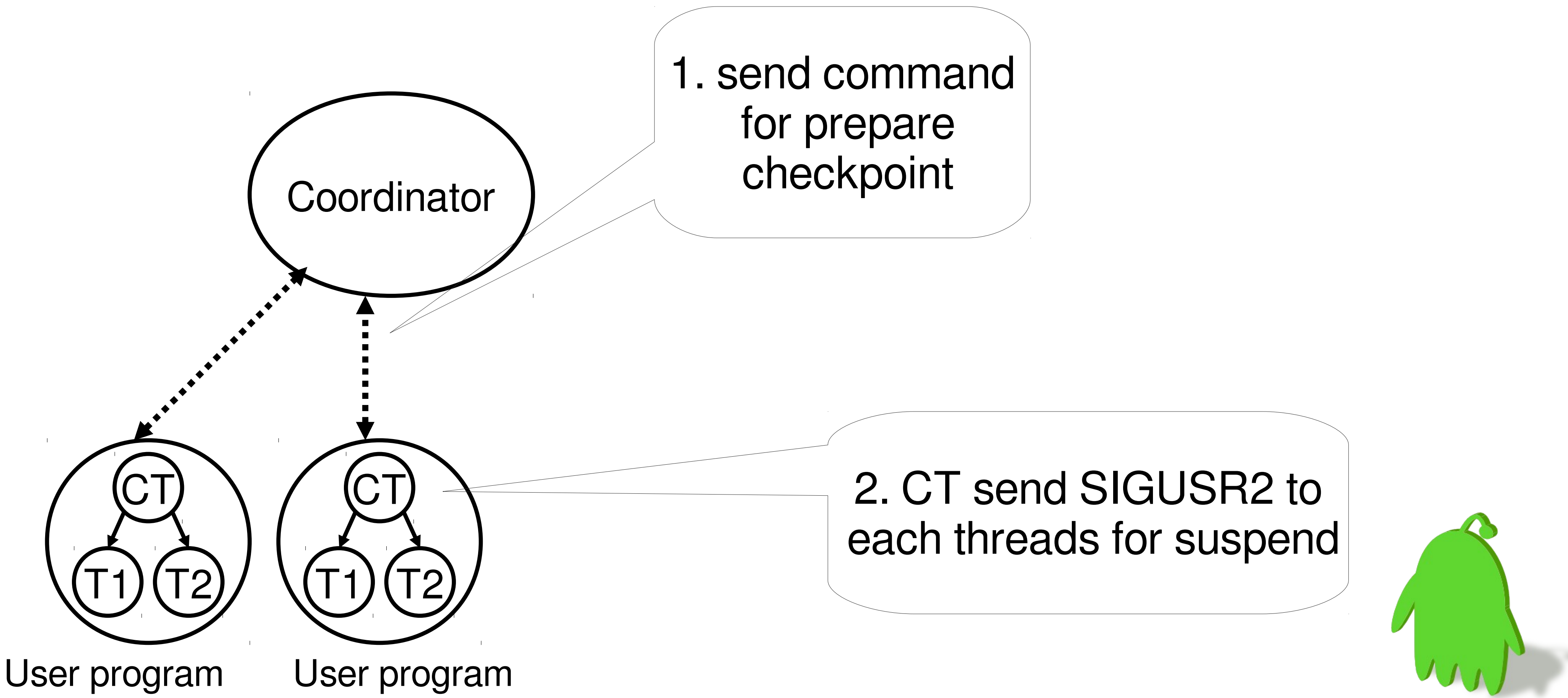
Checkpoint under DMTCP(2/7)

- Ask coordinator process for checkpoint via dmtcp_command.
  - **`dmtcp_command -c`**
- DMTCP also provides API to send command or query status



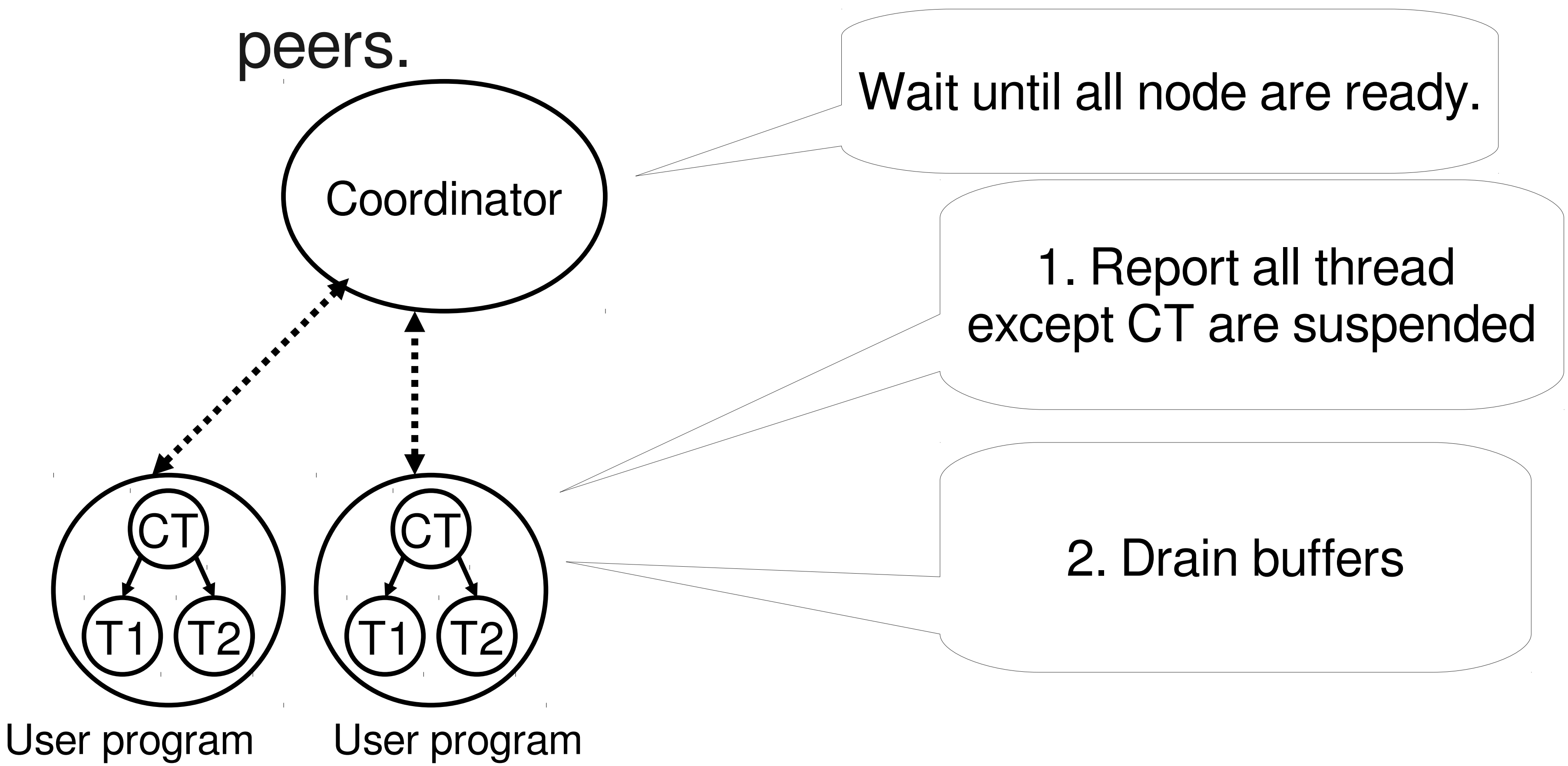
Checkpoint under DMTCP(3/7)

- Suspend user threads with SIGUSR2.



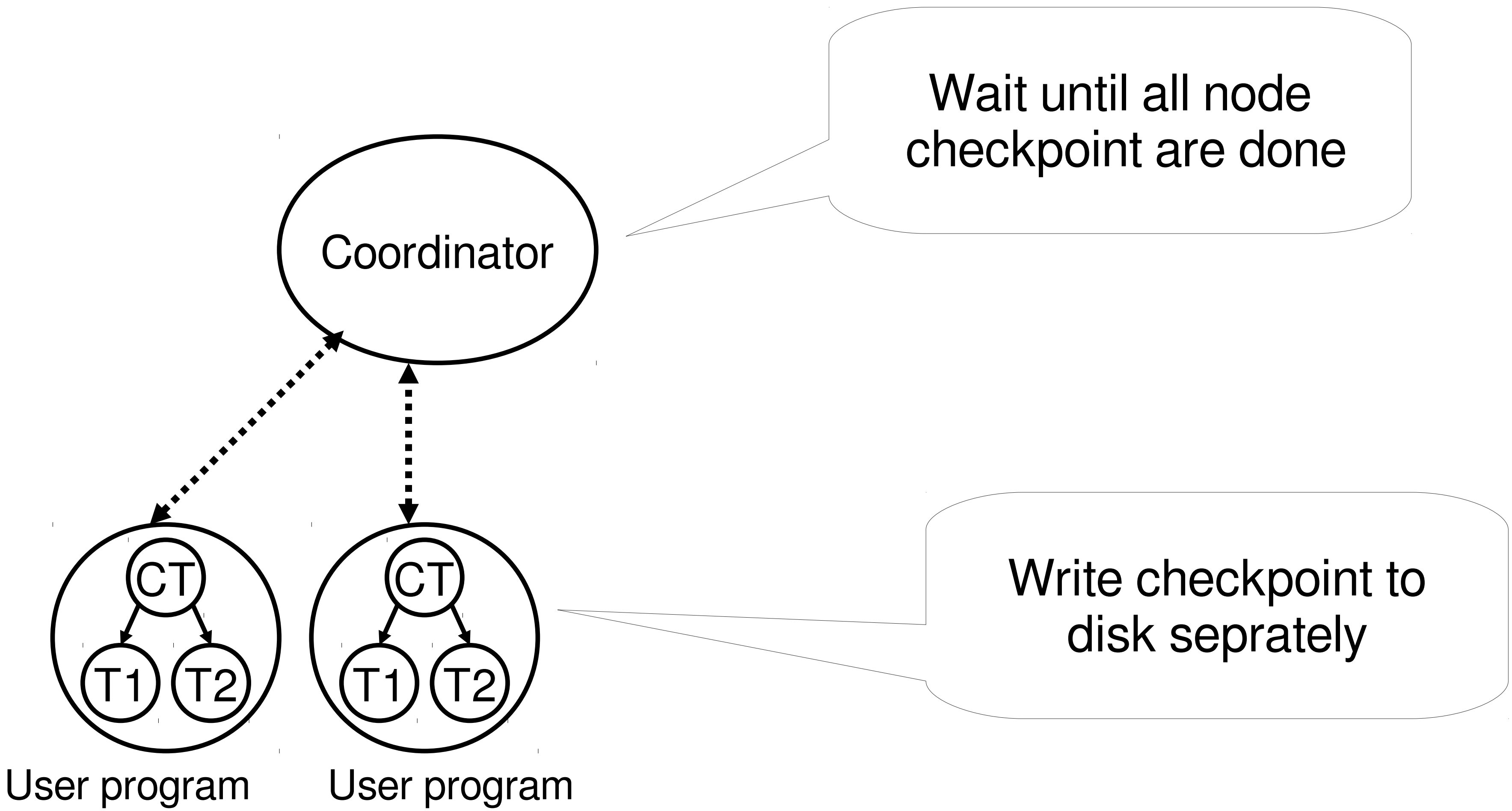
Checkpoint under DMTCP(4/7)

- Pre-checkpoint stage
- Synchronize every node and elect shared file descriptor leaders.
- Drain kernel buffers and do network handshake with peers.



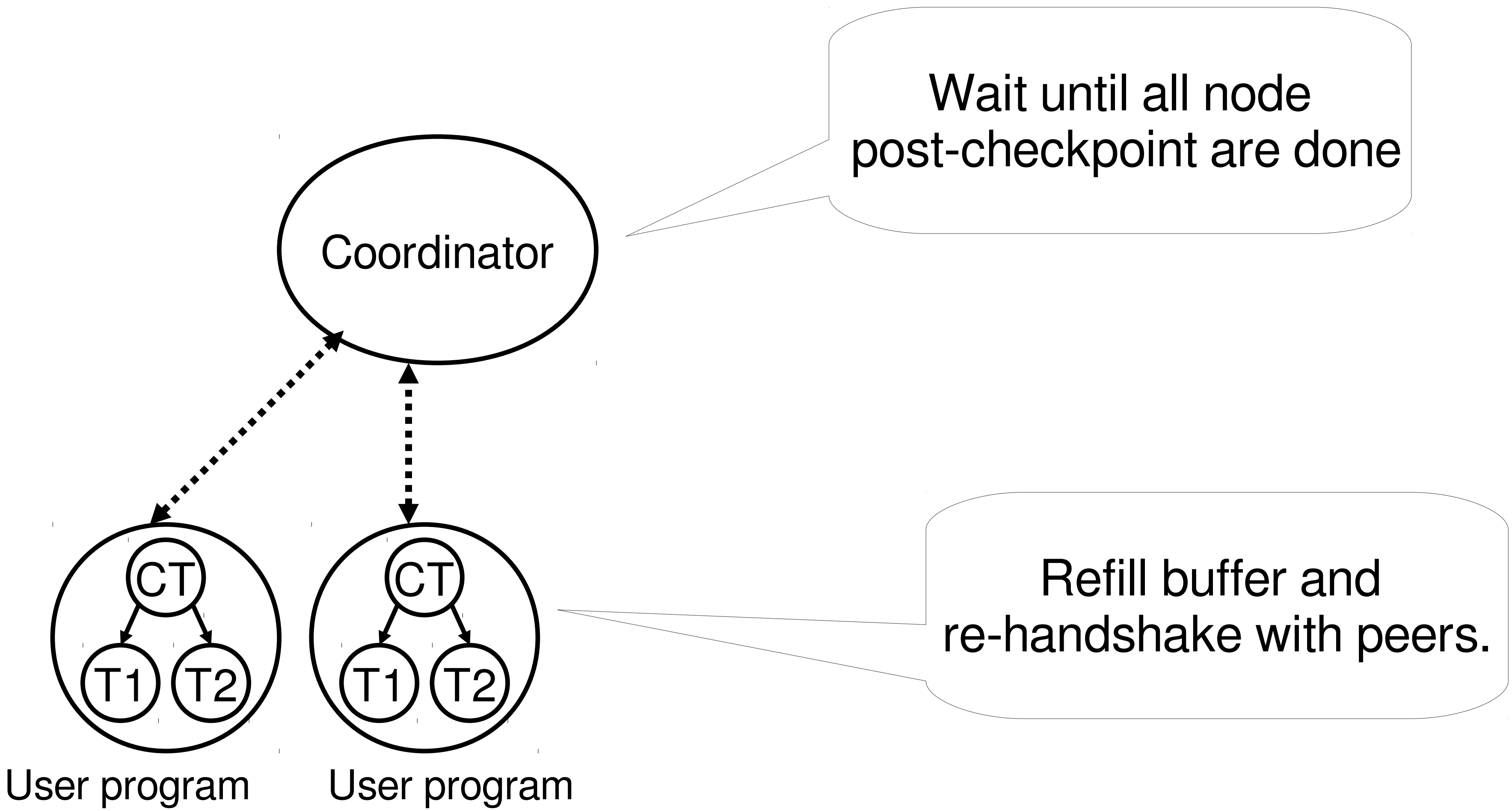
Checkpoint under DMTCP(5/7)

- Write checkpoint to disk
  - One checkpoint file per process
  - ckpt_<EXE>_<uid>.dmtcp



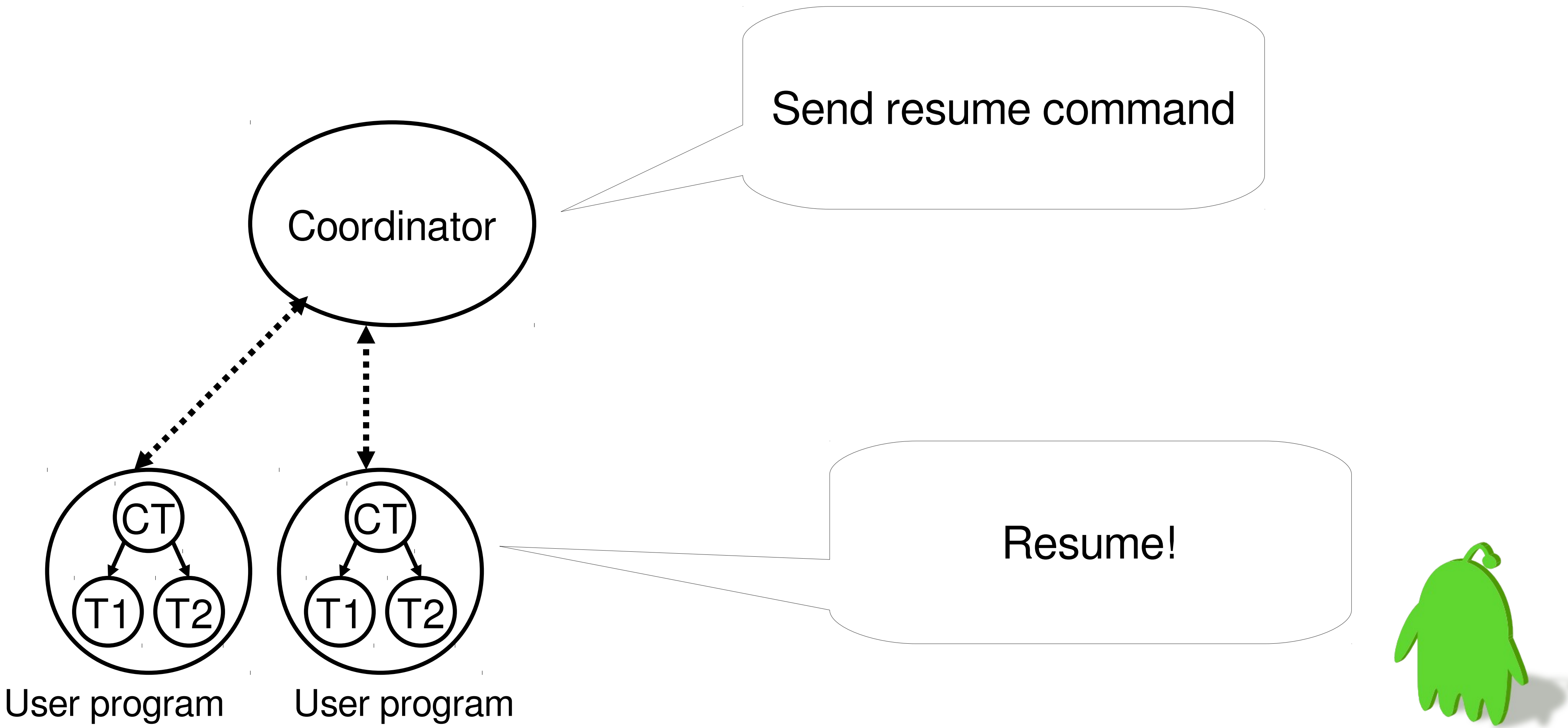
Checkpoint under DMTCP(6/7)

- Post-Checkpint stage
- Refill kernel buffers



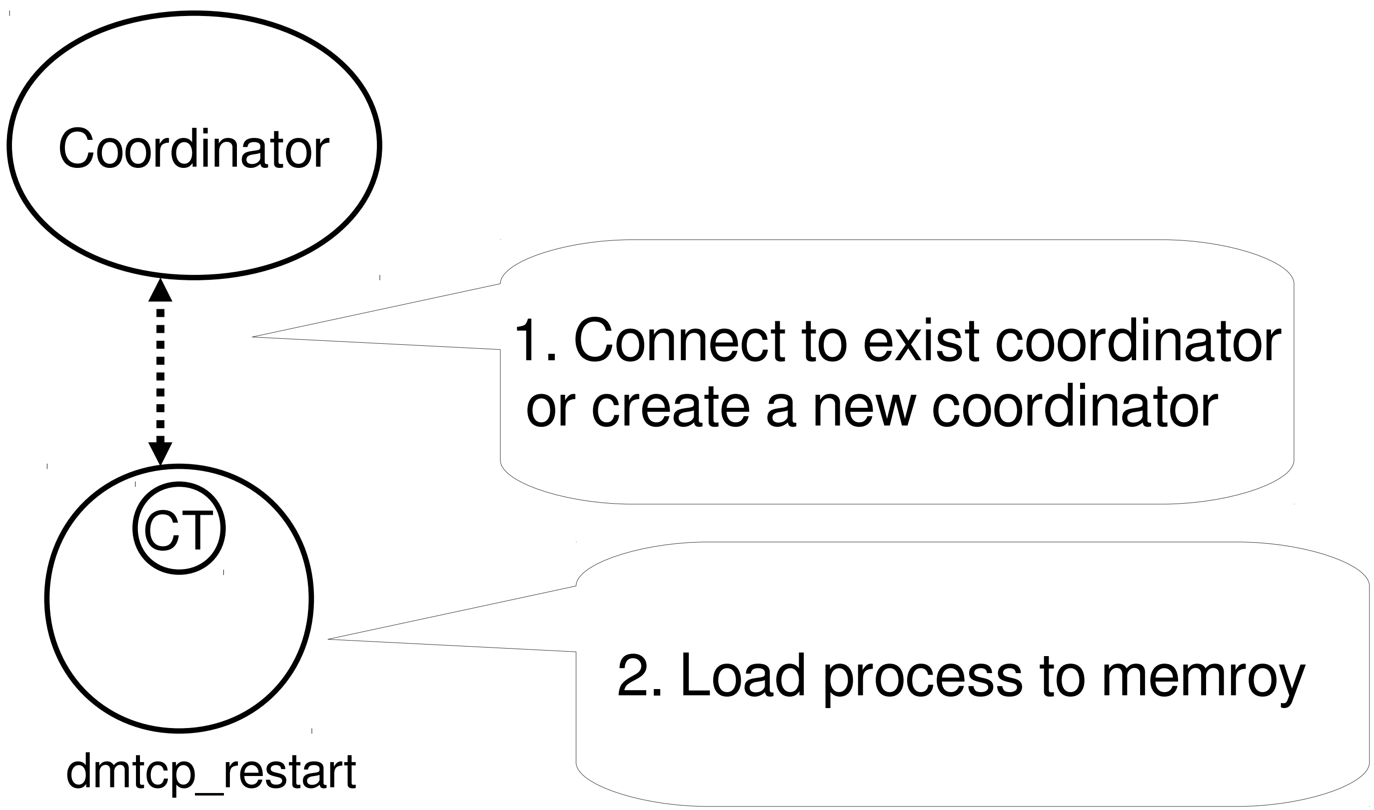
Checkpoint under DMTCP(7/7)

- Resume user threads.



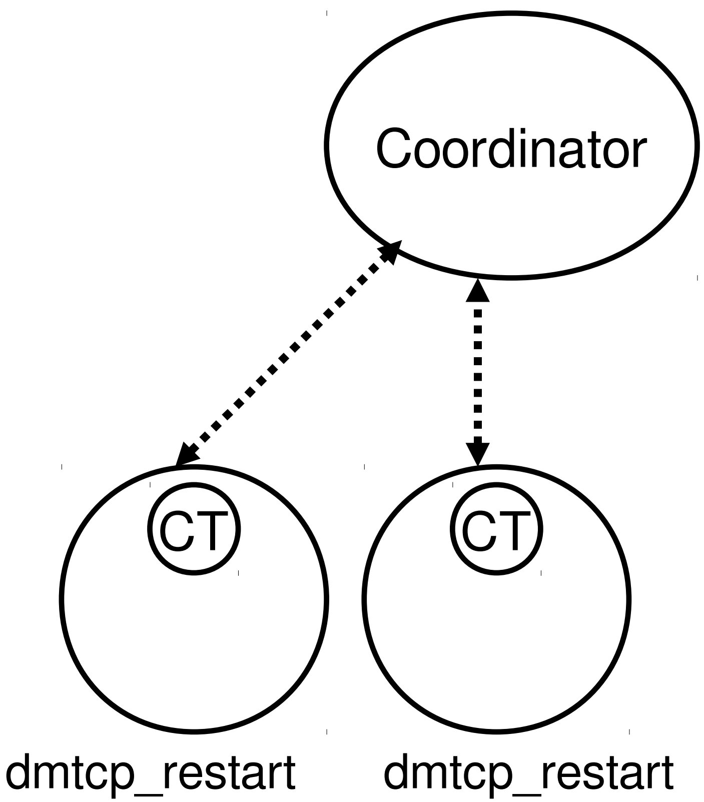
Restart under DMTCP(1/6)

- Restart Process loads in memory.
  - **dmtcp_restart** ckpt_<EXE>_<uid>.dmtcp



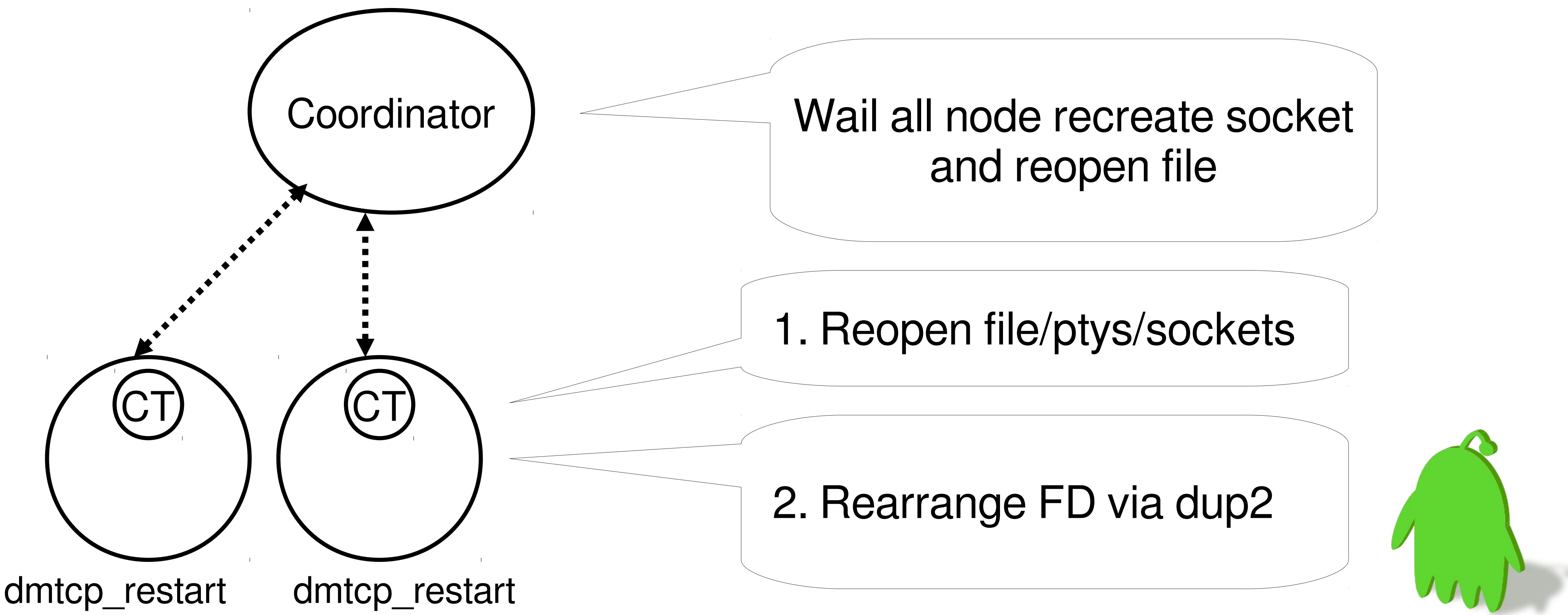
Restart under DMTCP(2/6)

- Fork user program



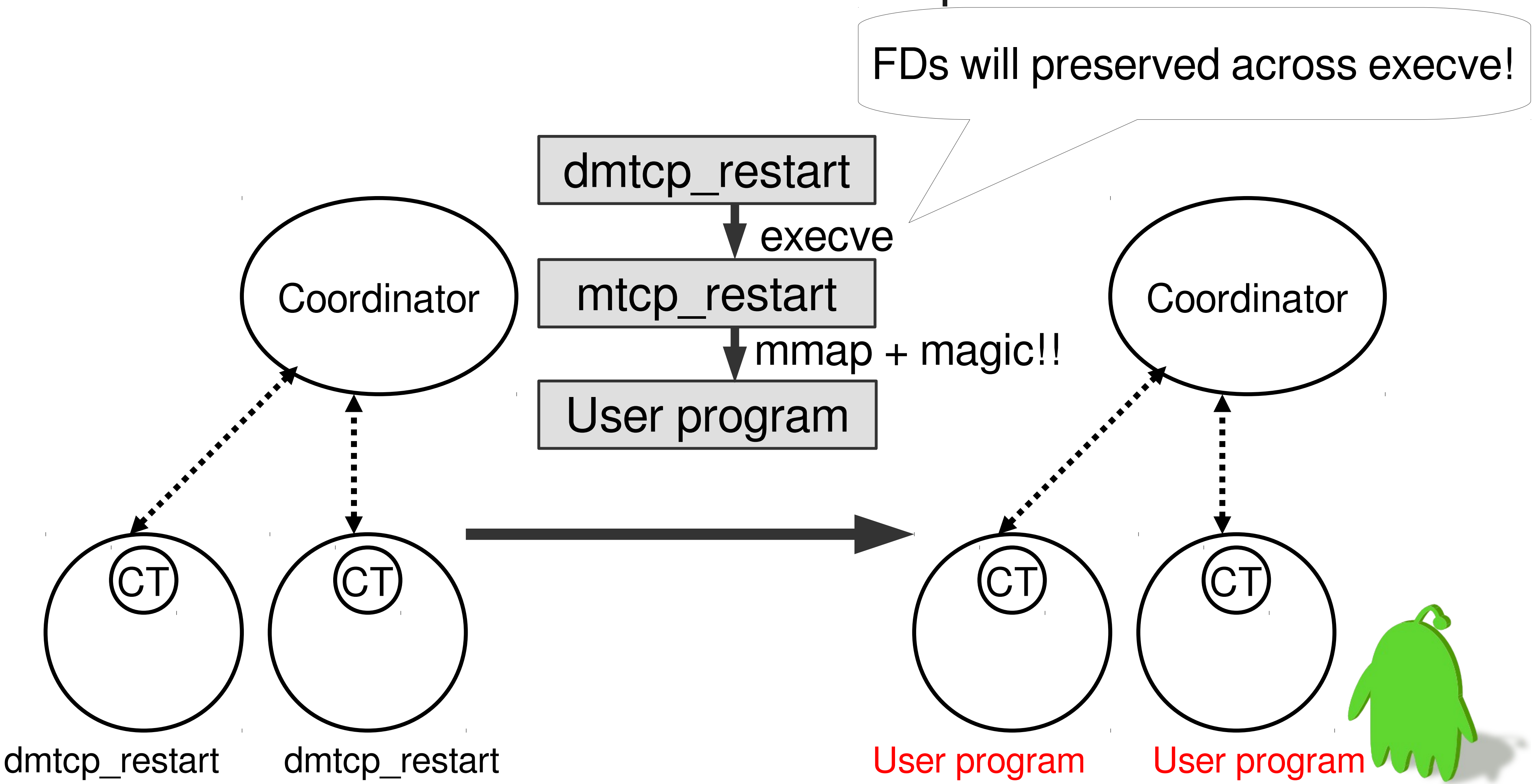
Restart under DMTCP(3/6)

- Reopen files and recreate ptys
- Recreate and reconnect sockets
- Rearrange file descriptors to initial layout



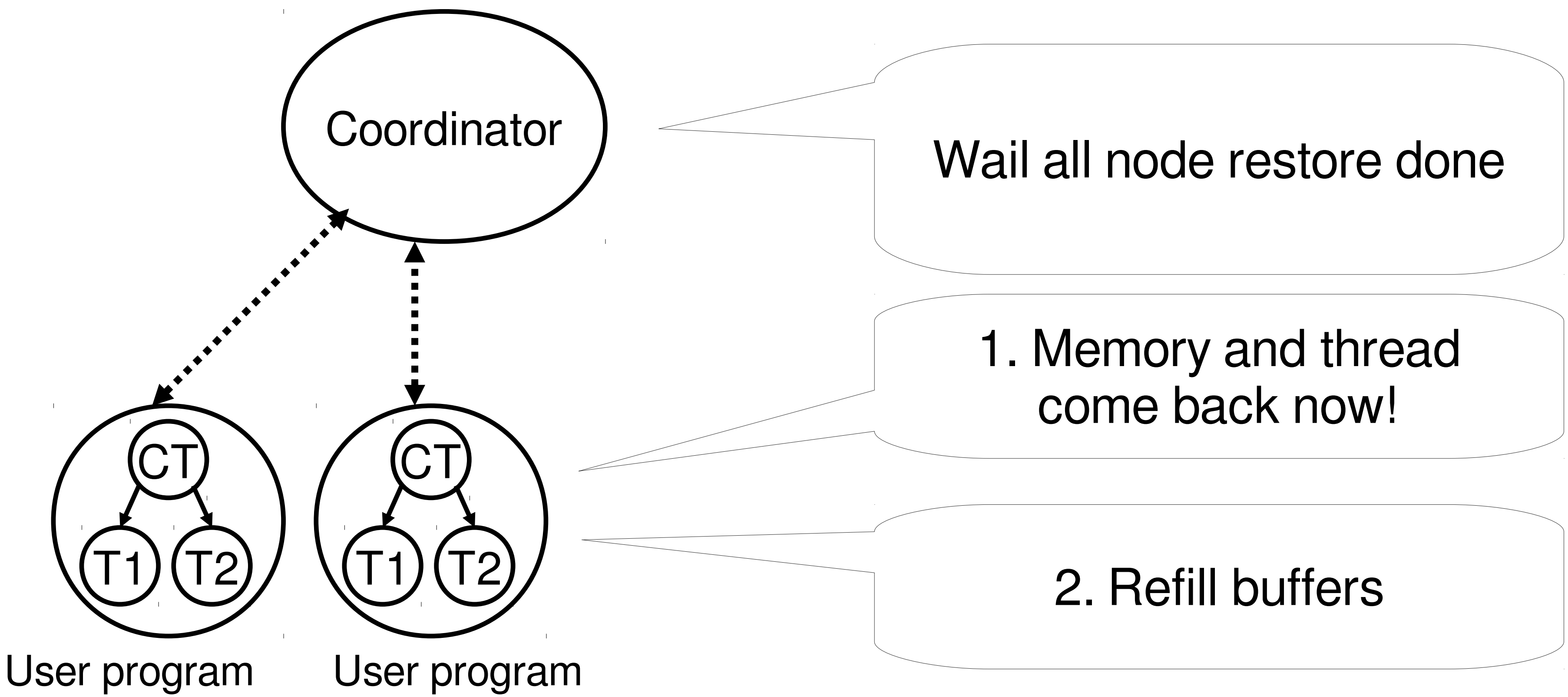
Restart under DMTCP(4/6)

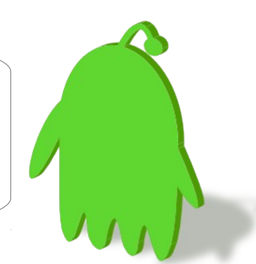
- Restore memory content.
- Restore stack status for checkpoint thread.



Restart under DMTCP(5/6)

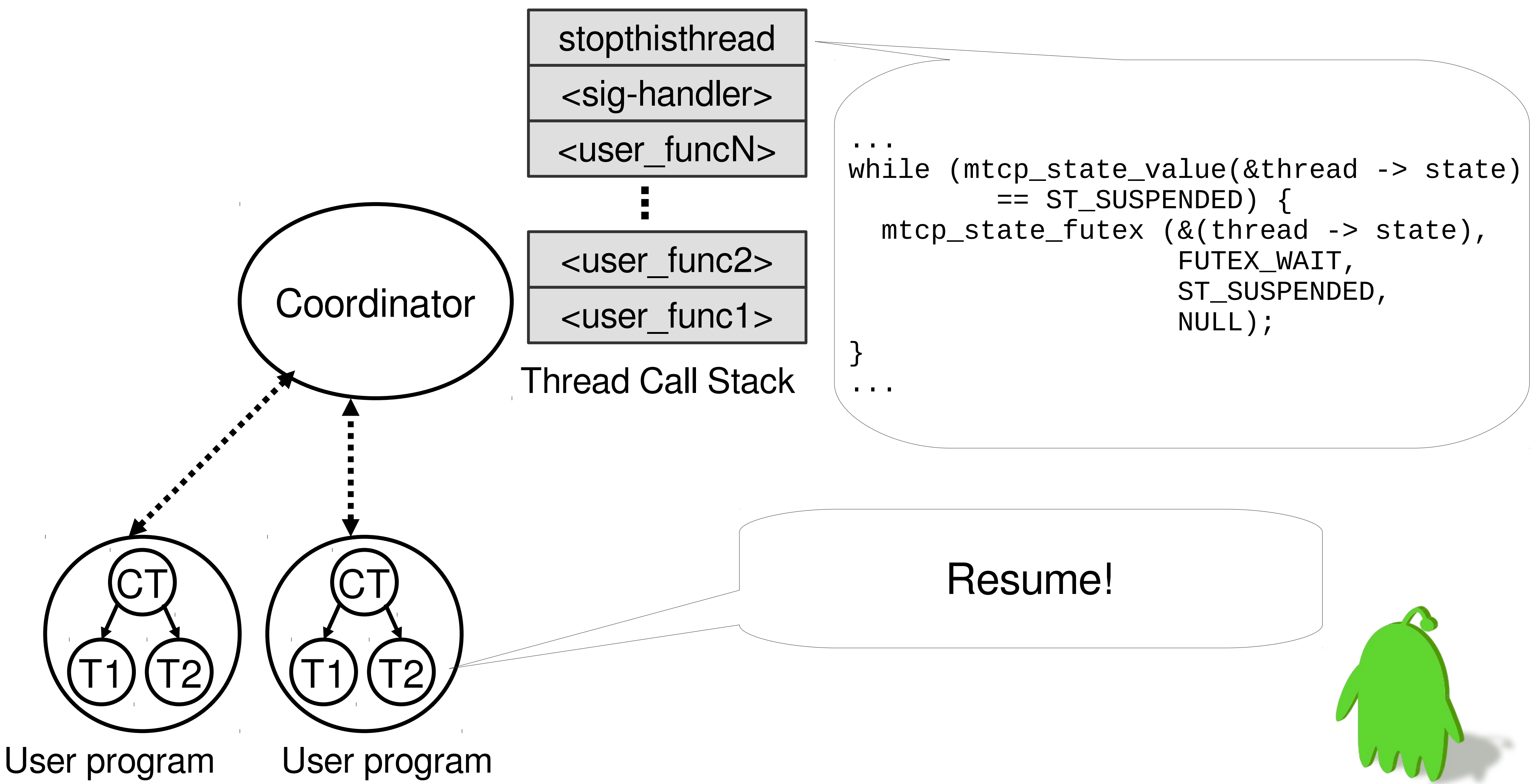
- Restore other threads.
  - Recreate thread and restore stack and context.
  - Restore back to the post-checkpint stage
- Refill kernel buffer



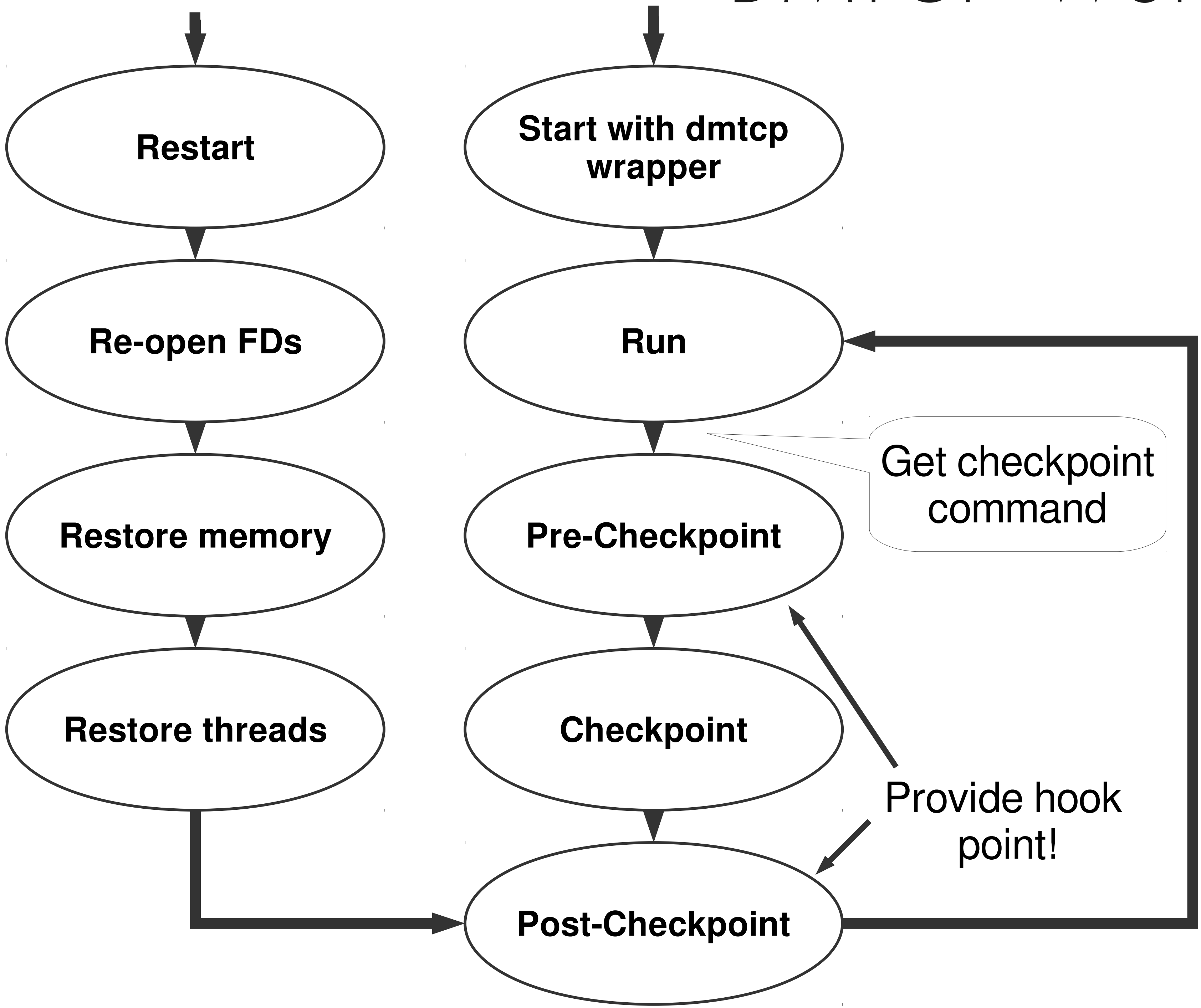


Restart under DMTCP (6/6)

- Resume user threads



DMTCP Workflow



OS Features supported by DMTCP

- Threads, mutexes/semaphores, fork, exec
- Shared memory (via mmap), TCP/IP sockets, UNIX domain sockets, pipes, ptys, terminal modes, ownership of controlling terminals, signal handlers, open and/or shared fds, I/O (including the readline library), parent-child process relationships, process id & thread id virtualization, session and process group ids, and more…

DMTCP/Android: Additional Features

(LGPL; separated from Android)

- ARM Architecture support
  - Verified on Samsung Galaxy S2 + Android 4.0
- Binder IPC
  - Client: supported
  - Server: partially supported
- Ashmem: supported
- Logger: supported
- Properties: supported
- Wakelocks: Not supported

Source code is available

https://github.com/**0xlab**/**dmtcp-android**
https://github.com/**0xlab**/**android-checkpoint**

Support new FD type in DMTCP

- In DMTCP, every FD has an associated `Connection`:
  - **`TcpConnection`**, **`FileConnection`**, **`PtyConnection`**
- Implement a new subclass of Connection if you want to support a new FD type for Android:
  - **`AshmemConnection`**, **`BinderConnection`**, **`LoggerConnection`**, **`PropertyConnection`**
- Also, implement the **`preCheckpoint`**, **`postCheckpint`**, and any others if needed.

Interface of Connection

```
class Connection {
  public:
    virtual void preCheckpoint (const dmtcp::vector<int>&,
                                KernelBufferDrainer&);
    virtual void postCheckpoint (const dmtcp::vector<int>&,
                                 bool);
    virtual void restore (const dmtcp::vector<int>&,
                          ConnectionRewirer *);
    virtual bool isDupConnection (const Connection&,
                                  dmtcp::ConnectionToFds&);
    virtual void doLocking (const dmtcp::vector<int>&);
    virtual void saveOptions (const dmtcp::vector<int>&);
    virtual void restoreOptions (const dmtcp::vector<int>& );
    virtual void doSendHandshakes (const dmtcp::vector<int>&,
                                   const dmtcp::UniquePid&);
    virtual void doRecvHandshakes (const dmtcp::vector<int>&,
                                   const dmtcp::UniquePid&);
    virtual void restartDup2 (int, int);
  protected:
    virtual void serializeSubClass (jalib::JBinarySerializer&);
};
```

Android Binder support for DMTCP

- BinderConnection
  - Reopen **`/dev/binder`** and reset ioctl parameters
  - Restore the mmap region
- Hijack the whole libbinder
  - Prevent libbinder from interpreting data twice
  - Implement necessary DMTCP hooks: **`preCheckpoint`**, **`postCheckpoint`**, **`postRestart`**
    - Re-initialize libbinder in **`postRestart`**
- The server part is partially supported because binder server is calling a blocked ioctl and blocking the whole checkpoint process.
  - We implement an early checkpoint stage to suspend such kind of threads.

More extensions in DMTCP/Android

- Improve the hook system in DMTCP
  - Original design only allows one set hook function.
  - Allow more than one set hook function in DMTCP/Android.
- Implement per thread callback hook
  - Restore the DVM internal thread info
- Add barrier and synchronize mechanisms to DMTCP
  - In order to make precise program checkpointing.

Android specific modifications

- Reorder code in framework
  - registerZygoteSocket()
    - The socket is inherited from the parent process `init`, which implies we can not handle it in DMTCP.
  - Move few initializations later than the checkpoint process since the current binder support is incomplete.
- Reserve the ashmem's file descriptor
  - Original behavior is to close the fd after mmap
  - DMTCP binds connection to one fd, so the connection will be destroyed if that fd is closed.
- Implement the missing PThread function in bionic libc
  - pthread_tryjoin_np is required by DMTCP,

    but it s not implemented in original bionic.

Technical Issues when modifying DMTCP

- ARM Architecture support is incomplete.
  - We are going to contribute back to upstream.
- Different TLS implementation semantics between glibc and bionic libc
  - DMTCP/Android follows the techniques used in Android´s OpenGL ES package which links and defers to the slot of TLS in bionic libc. Not elegant, but it works
- PThread implementation expectation is quite different
  - AOSP master branch is merging libc from NetBSD, so it should be better for compatibility.
- Behavior of dynamic linker differs a lot in bionic libc.
- Flags in dlopen() is not really functional.
- The way to find symbol in bionic libc differs: weak symbol

Checkpoint for Zygote

- Experiment environment:
  - Android-x86 ICS emulated by VirtualBox
  - Host: Thinkpad x200 (Intel Core 2 Due @ 2.4 GHz)

	with gzip	without gzip
Checkpoint time	~10s	~5.5s
Restart time	~0.5s	~0.2s
Image size	~3M	~17M

Observations from logcat

```
--------- beginning of /dev/log/system
I/Vold    ( 1270): Vold 2.1 (the revenge) firing up
D/Vold    ( 1270): Volume usb state changing -1 (Initializing) -> 0 (No-Me
I/Netd    ( 1271): Netd 1.0 starting
I/        ( 1275): ServiceManager: 0x8062b50
I/        ( 1276): ServiceManager: 0x804fb98
I/AudioFlinger( 1276): Loaded primary audio interface from LEGACY Au
I/AudioFlinger( 1276): Using 'LEGACY Audio HW HAL' (audio.primary) a
...
D/AudioHardware( 1276): ### setVoiceVolume: 1.000000
I/AudioPolicyService( 1276): [1276]Loaded audio policy from LEGACY A
E/BatteryService( 1382): usbOnlinePath not found
D/AndroidRuntime( 1902):
D/AndroidRuntime( 1902): >>>>>> AndroidRuntime START com.androi
D/AndroidRuntime( 1902): CheckJNI is ON
I/SamplingProfilerIntegration( 1902): Profiling disabled.
I/Zygote  ( 1902): Preloading classes...
D/dalvikvm( 1902): GC_EXPLICIT freed 35K, 85% free 399K/2560K, pa
...
I/Zygote  ( 1902): ...preloaded 379 resources in 548ms.
D/dalvikvm( 1902): GC_EXPLICIT freed 20K, 1% free 6417K/6467K, pa
I/Zygote  ( 1902): ...preloaded 31 resources in 13ms.
D/dalvikvm( 1902): GC_EXPLICIT freed 14K, 1% free 6418K/6467K, pa
D/dalvikvm( 1902): GC_EXPLICIT freed 5K, 1% free 6412K/6467K, pau
D/dalvikvm( 1902): GC_EXPLICIT freed <1K, 1% free 6412K/6467K, pa
I/dalvikvm( 1902): System server process 1911 has been created
```

Normal bootup log message

```
--------- beginning of /dev/log/system
I/Vold    ( 1270): Vold 2.1 (the revenge) firing up
D/Vold    ( 1270): Volume usb state changing -1 (Initializing) -> 0 (No-Media)
I/Netd    ( 1271): Netd 1.0 starting
I/        ( 1275): ServiceManager: 0x8062b50
I/        ( 1276): ServiceManager: 0x804fb98
I/AudioFlinger( 1276): Loaded primary audio interface from LEGACY Audio H
I/AudioFlinger( 1276): Using 'LEGACY Audio HW HAL' (audio.primary) as the
....
D/AudioHardware( 1276): ### setVoiceVolume: 1.000000
I/AudioPolicyService( 1276): [1276]Loaded audio policy from LEGACY Audio
D/dalvikvm( 1373): GC_EXPLICIT freed 14K, 1% free 6418K/6467K, paused
D/dalvikvm( 1373): GC_EXPLICIT freed 5K, 1% free 6412K/6467K, paused 0
D/dalvikvm( 1373): GC_EXPLICIT freed <1K, 1% free 6412K/6467K, paused
I/dalvikvm( 1373): System server process 1382 has been created
```

Bootup log message with restart

Android Boottime with DMTCP

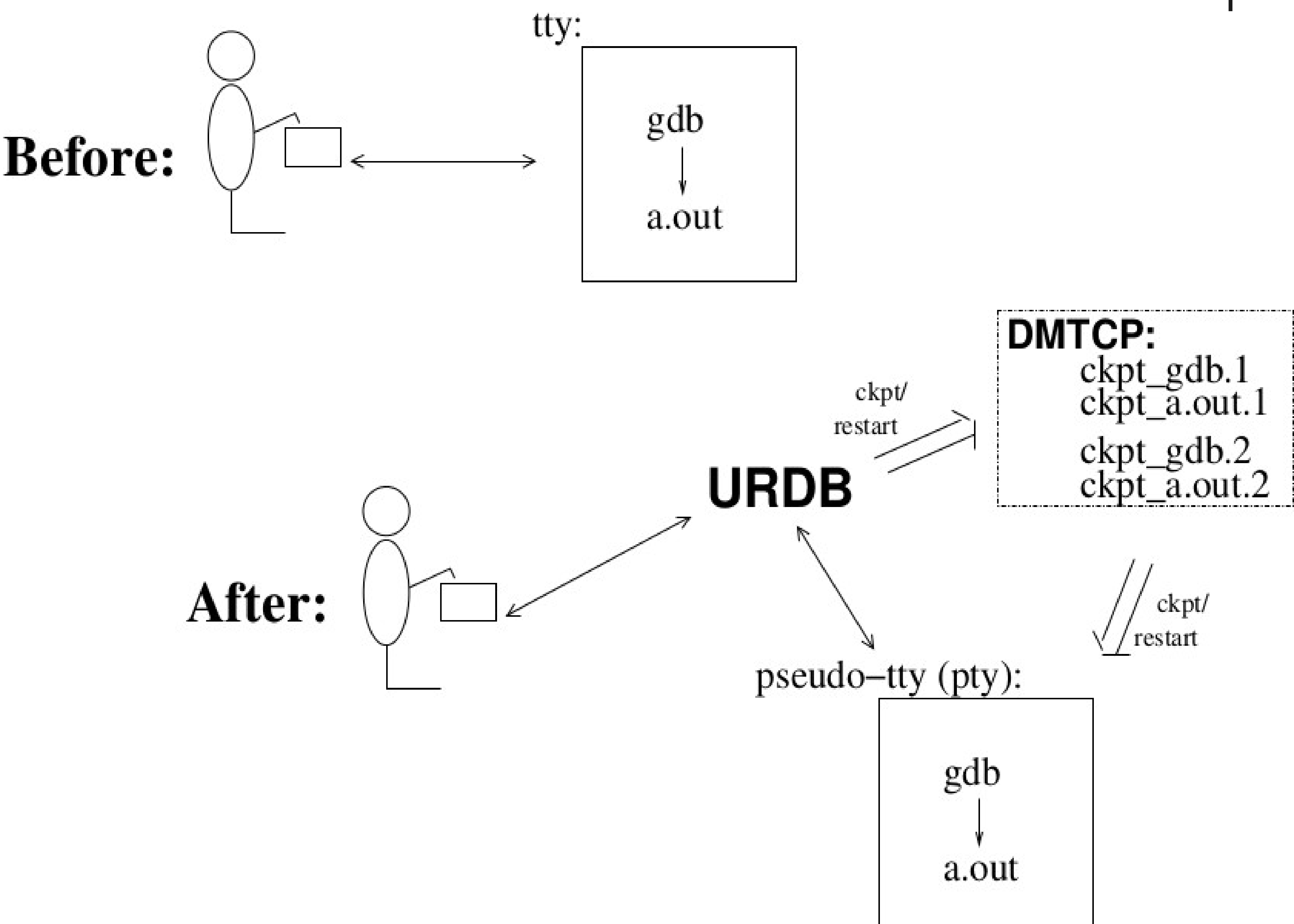
	1	2	3	4	5	6	7	8	9	10	avg
Boottime	27.96	27.95	32.89	26.59	32.33	32.36	33.22	32.99	36.47	32.85	31.56
Boottime with DMTCP (w/ gzip)	15.02	15.86	15.13	14.88	14.57	14.43	14.73	14.22	13.97	14.61	14.74
Boottime with DMTCP (w/o gzip)	14.98	15.13	14.61	13.90	14.72	14.84	15.46	15.06	15.32	15.39	14.94

Measured by uptime in **`onStart`**() of Launcher2

Reversible Debugger based on DMTCP

- URDB is a universal reversible debugger that can add reversibility gained through the use of DMTCP.
- A user debugging with gdb would then ask URDB to go back to a point in time to when:
  - the expression had a correct value
  - the next statement would cause the expression to take on an incorrect value.
- Project page: http://sourceforge.net/projects/urdb/

Reversible Debugger using Checkpointing



Reference

- “DMTCP: An New Linux Checkpointing Mechanism for Vanilla Universe Job”, Condor Project, University of Wisconsin-Madison
- “Checkpointing using DMTCP, Condor, Matlab and FReD”, Gene Cooperman, Northeastern University, Boston
- URDB: Universal Reversible Debugger

  http://arxiv.org/abs/0910.5046
- Cruz: Application-Transparent Distributed Checkpoint-Restart on Standard Operating Systems, G. (John) Janakiraman, Jose Renato Santos, Dinesh Subhraveti, Yoshio Turner, HP Labs

0xlab

http://0xlab.org