



Shorten Device Boot Time for Automotive IVI and Navigation Systems

Jim Huang (黃敬群) <jserv@0xlab.org>

Dr. Shi-wu LO <shiwulo@gmail.com>

May 28, 2013 / Automotive Linux Summit (Spring)

Rights to copy

© Copyright 2013 **0xlab**

<http://0xlab.org/>

contact@0xlab.org

Attribution – ShareAlike 3.0

You are free

Corrections, suggestions, contributions and translations
are welcome!

- to copy, distribute, display, and perform the work
- to make derivative works
- to make commercial use of the work

Latest update: May 28, 2013

Under the following conditions

- **BY:** **Attribution.** You must give the original author credit.
- **Share Alike.** If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.
- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

License text: <http://creativecommons.org/licenses/by-sa/3.0/legalcode>



Goal of This Presentation

- Propose a practical approach of the mixture of ARM hibernation (suspend to disk) and Linux user-space checkpointing
 - to shorten device boot time
- An intrusive technique for Android/Linux
 - minimal init script and root file system changes are required
- Boot time is one of the key factors for Automotive IVI
 - mentioned by “*Linux Powered Clusters*” and “*Silver Bullet of Virtualization (Pitfalls, Challenges and Concerns Continued)*” at ALS 2013
 - highlighted by “*Boot Time Optimizations*” at ALS 2012



About this presentation

- joint development efforts of the following entities
 - **0xlab** team - <http://0xlab.org/>
 - **OSLab**, National Chung Cheng University of Taiwan, led by Dr. Shi-wu Lo. "Software platform on SoC" project at ITRI/ICL, supported by MOEA of Taiwan.
 - **DMTCP** Project at Northeastern University, led by Dr. Gene Cooperman
- The implementation is public for reference
 - Kernel part: GNU GPLv2
 - Userspace: GNU LGPL
- Some areas of the implementation might be patented. We don't expect to discuss here.



Disclaimer of Warranties

- Technical background knowledge in this presentation is based on the experience with our customers / partners such as CSR and MediaTek.
 - We share non-confidential parts to endorse the collaboration of Linux ecosystem.
- We make no warranty, either express or implied with respect to any product, and specially disclaim the correctness or real-world behavior.



Agenda

(1) Concepts: Analysis, Strategies

(2) Boot Time Reduction:

from traditional to our experiments

(3) ARM Hibernation

(4) Checkpointing

(5) Mixed model




Concepts: Analysis & Strategies
(no silver bullet, but you can optimize iteratively)



Boot Time Measurement





[page](#) [discussion](#) [view source](#) [history](#)

Boot Time

Contents [\[hide\]](#)

- 1 Introduction
- 2 Technology/Project Pages
 - 2.1 Measuring Boot-up Time
 - 2.2 Technologies and Techniques for Reducing Boot Time
 - 2.2.1 Bootloader speedups
 - 2.2.2 Kernel speedups
 - 2.2.2.1 File System issues
 - 2.2.3 User-space and application speedups
 - 2.2.4 Suspend related improvements
 - 2.2.5 Miscellaneous topics
 - 2.2.6 Uninvestigated speedups
- 3 Articles and Presentations
 - 3.1 Case Studies
- 4 Additional Projects/Mailing Lists/Resources
 - 4.1 Replacements for SysV 'init'
 - 4.1.1 busybox init
 - 4.1.2 upstart
 - 4.1.3 Android init
 - 4.1.4 systemd
 - 4.2 Kexec
 - 4.3 Splash Screen projects
 - 4.4 Others
 - 4.4.1 Apparently obsolete or abandoned material
- 5 Companies, individuals or projects working on fast booting
- 6 Boot time check list

navigation

- [Main Page](#)
- [Community portal](#)
- [Current events](#)
- [Recent changes](#)
- [Help](#)
- [Volunteering](#)
- [Popular Pages](#)
- [Who's Online](#)

search

toolbox

- [What links here](#)
- [Related changes](#)
- [Special pages](#)
- [Printable version](#)
- [Permanent link](#)



http://elinux.org/Boot_Time

Traditional Linux Environment:

printk, initcall_debug, bootchart, strace, oprofile,
perf, ..



Bootchart

- `$ bootchart bootchart.tgz -f png`

Boot chart for serenity.klika.si (Sun Apr 10 13:33:49 CEST 2005)

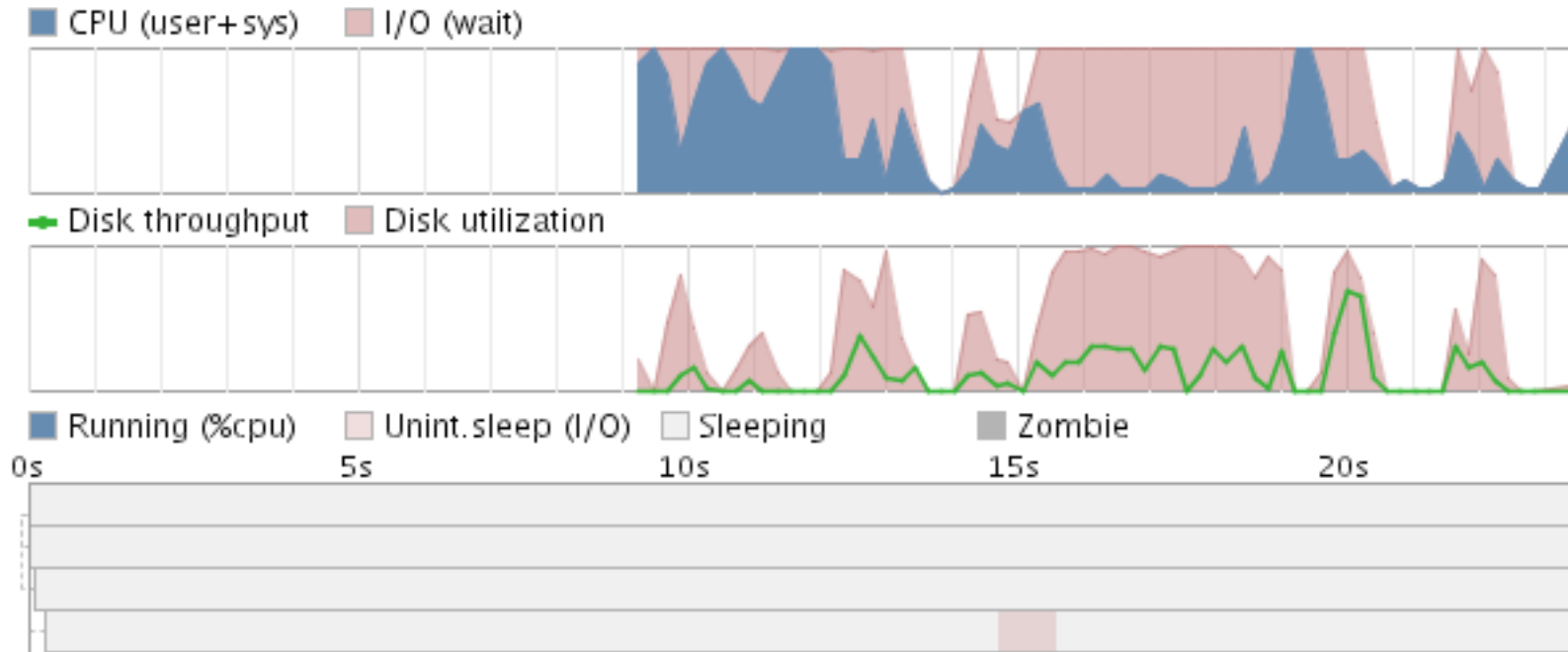
uname: Linux 2.6.11-1.1233_FC4 #1 Fri Apr 8 08:56:16 EDT 2005 i686

release: Fedora Core release Rawhide (Rawhide)

CPU: Intel(R) Pentium(R) M processor 1500MHz (1)

kernel options: ro root=LABEL=/ init=/sbin/bootchartd rhgb

time: 1:15



Strace

- Trace system calls during process execution and output timing information.

```
$ strace -tt ls
15:11:04.243357 execve("/bin/ls", ["ls"], [/* 51 vars */]) = 0
15:11:04.244252 brk(0) = 0x234f000
15:11:04.244458 access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT
15:11:04.244676 mmap(NULL, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f1444794000
15:11:04.244852 access("/etc/ld.so.preload", R_OK) = -1 ENOENT
15:11:04.245096 open("/etc/ld.so.cache", O_RDONLY) = 3
```



OProfile

- Output Example

```
$ opreport --exclude-dependent
CPU: PIII, speed 863.195 MHz (estimated)
Counted CPU_CLK_UNHALTED events (clocks processor is not
halted)...
450385 75.6634 cc1plus
60213 10.1156 lyx
29313 4.9245 XFree86
11633 1.9543 as
10204 1.7142 oprofiled
7289 1.2245 vmlinux
7066 1.1871 bash
6417 1.0780 oprofile
6397 1.0747 vim
...
```



Perf

- Recording:

```
$ perf record -a -f  
^C
```

```
[ perf record: Woken up 1 times to write data ]
```

```
[ perf record: Captured and wrote 0.288 MB perf.data (~12567  
samples)]
```

- Output

```
$ perf report --sort comm,dso,symbol|head -10
```

```
# Events: 1K cycles
```

```
#
```

```
# Overhead      Command           Shared Object      Symbol
```

```
# .....  
#
```

35.47%	firefox	libxul.so	[.] 0xc1b3a7
3.08%	firefox	libcairo.so.2.11000.2	[.] 0xff88
2.98%	Xorg	Xorg (deleted)	[.] 0xe201c
2.51%	firefox	firefox	[.] 0x2726
1.49%	Xorg	[kernel.kallsyms]	[k] find_vma
0.93%	perf_3.0.0	perf_3.0.0	[.] hex2u64

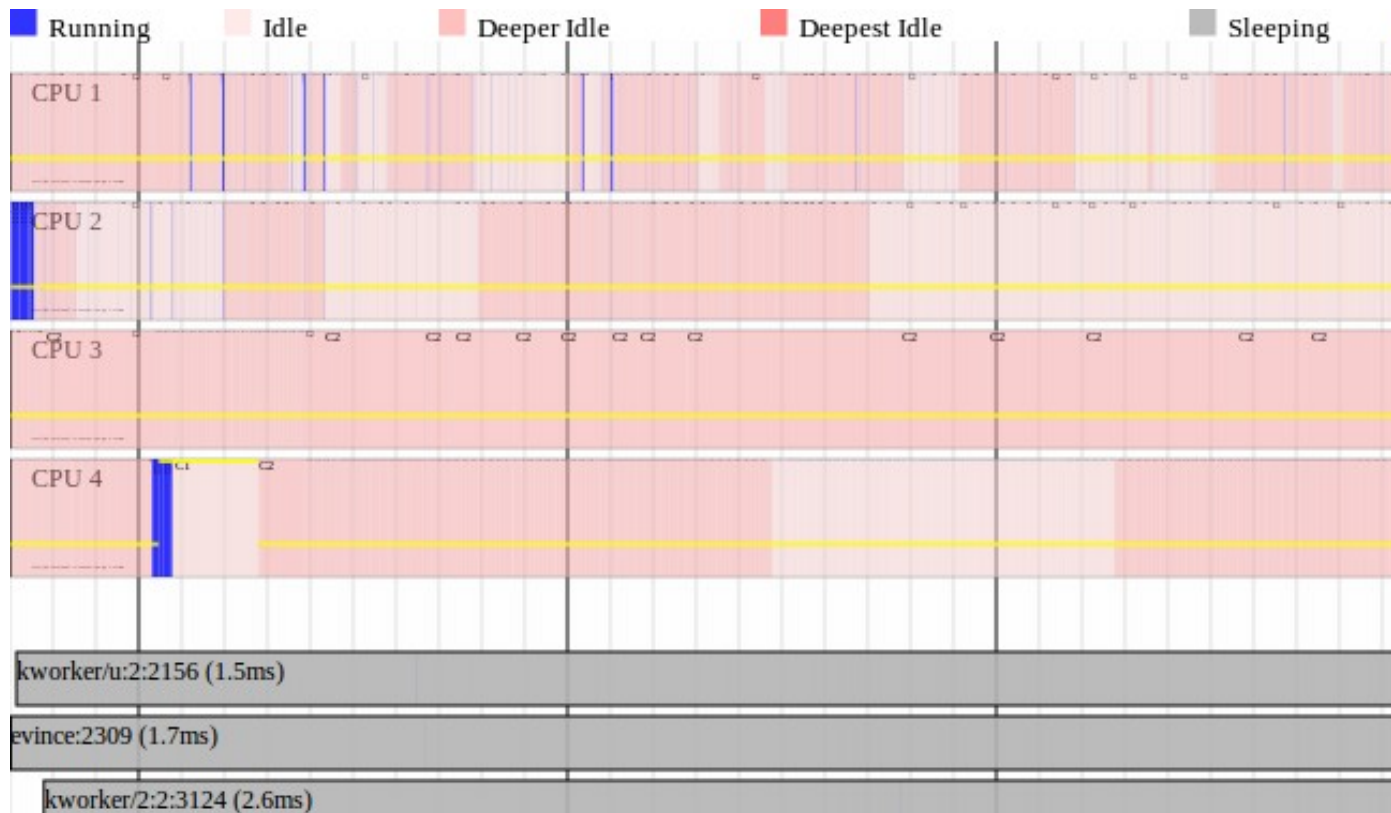


Perf

- Timechart

\$ perf timechart record

\$ perf timechart



Android Environment



Bootchart

- Original “bootchartd” is not suitable on embedded devices.
- Android re-implemented in its “init”.



Bootchart

- To build:
\$ cd system/core/init
\$ touch init.c
\$ mm INIT_BOOTCHART=true



Bootchart

- To run:
`$ adb shell 'echo 120 > /data/bootchart-start'`
- Remember the /data directory must be write able during boot.
- Use `grab-bootchart.sh` to retrieve the data.



Strace

- Modify init.rc from

```
service zygote /system/bin/app_process \  
    -Xzygote /system/bin \  
    --zygote --start-system-server
```

- To

```
service zygote /system/xbin/strace -tt \  
    -o/data/boot.strace \  
    /system/bin/app_process -Xzygote \  
    /system/bin \  
    --zygote --start-system-server
```



Logcat

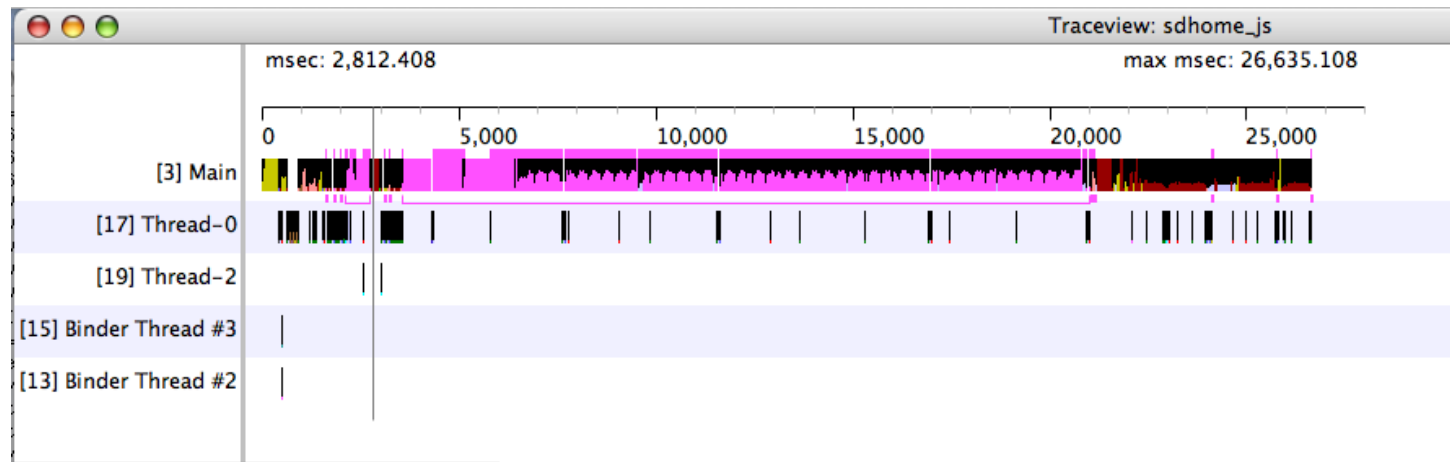
- Android log utility
- Can output timing information
- Adjust loglevel to 6 in init.rc
 - **Displays time spent for each command**



Dalvik Method Tracer

- Method tracer is built into Dalvik
- Use DDMS or using calls inside source to collect data.

```
// start tracing to "/sdcard/calc.trace"  
Debug.startMethodTracing("calc");  
// ...  
// stop tracing  
Debug.stopMethodTracing();
```



Stopwatch

- Not real stopwatch
- A utility in Android Framework for measuring C++ code.
- Output result to system log

```
#include <utils/StopWatch.h>
...
{
    Stopwatch watch("blah");
    /* your codes here */
}
```



Android Boot Time Analysis



Boot-loader Init

- Usually constant time
- Avoid init hardware multiple times
- Ensure to use maximum CPU frequency
- Use faster NAND/MMC reading mechanism



Kernel Init

- Mostly usual suspects
 - `ip_auto_config`
 - USB init
 - Flash driver initialization
- Follow the standard Kernel optimizing guide:
 - http://elinux.org/Boot_Time
- Avoid loading unneeded kernel module at boot time



Zygote Class Preloading

- Android Framework has thousands of Java classes
- Preloaded by Zygote and instantiated in its heap
- To improve Application startup time and save memory
- Controlled by resource: preloaded-classes
 - `frameworks/base/preloaded-classes`



Zygote Class Preloading

- Can use the tool in framework to adjust the list:
\$ adb logcat > logcat.txt
\$ java -p preload.jar Compile logcat.txt logcat.compiled
\$ java -p preload.jar PrintCsv logcat.compiled
- Google Android Developer Dianne Hackborn said:
 - The content of the file is a “black art”
 - You can adjust this as much as you like
 - But the result maybe suboptimal



PackageManager Package Scanning

- Every APK is scanned at boot time
- Package management code is inefficient
- Uses mmaped files means each access will cause page fault
- ParseZipArchive() scans entire APK for only one AndroidManifest.xml file



System Services Starting

- Last stage of Android boot
- Start every base service
- Zygote start SystemServer process
- Start native service (SurfaceFlinger, AudioFlinger) first
- Start each service sequentially



Boot Time Reduction



Boot Time Reduction:

The traditional approaches mentioned by many developers/speakers already



Boot-loader Improvements:

Write our faster one!

- Qi Boot-loader

- Developed by Openmoko and Oxlab
- Only one stage boot-loader
- Small footprint ~30K
- Currently support
 - IMX31, Samsung 24xx, Beagleboard, Pandaboard
- KISS concept
 - Boot device and load kernel

	Qi Boot-loader	U-Boot + XLoader
Size	~30K	~270K+20K
Time to Kernel	< 1 s	> 5s
Usage	Product	Engineering
Code	Simple	Complicated

Another example:

“Boot Time Optimizations”, Alexandre Belloni, ELCE 2012



Kernel Boot Time

- Follow the standard Kernel optimizing guide:
 - http://elinux.org/Boot_Time
- Minimize kernel size
- Use compression or not
- Enable embedded options
- Avoid loading unneeded kernel module at boot time



Optimize Android Init

- Parallize init tasks
 - insmod cannot be parallized
 - Use external scripts to init at background
- Start services on demand



Optimize Class Preloading

- Trade-off between preload class and application startup time
- Split class to more packages to reduce dependency
- Save init'd heap for later use
- Share heaps between zygote and children



Filesystem Optimization

- According to reasearch by Linaro Kernel WG
- Use correct NAND configuration will improve the performance
- MMC controllers are often optimized for particular usage / filesystem
- Adjust the filesystem partition scheme



Toothpaste Effect

- Observed by Sony Developer Tim Bird
- “When you squeeze a tube of toothpaste, sometimes it just moves the toothpaste somewhere else in the tube, and nothing actually comes out.”



Example:

“Trying to Improve Android Boot Time With Readahead”, Tim Bird, ABS 2011



Lessons learnt

- Complex userspace environments like Android usually contribute the most to boot time.
 - (reported) < 1 s boot time often refers to minimal kernel + simplified initscript, which doesn't help
- GregKH: "If you want a faster D-Bus, rewrite the daemon / library, don't mess with the kernel." at ALS 2013
- Propose an intrusive technique for Android/Linux
 - Mixture of Hibernation and Userspace checkpointing



Part I:

Hibernation based Technologies

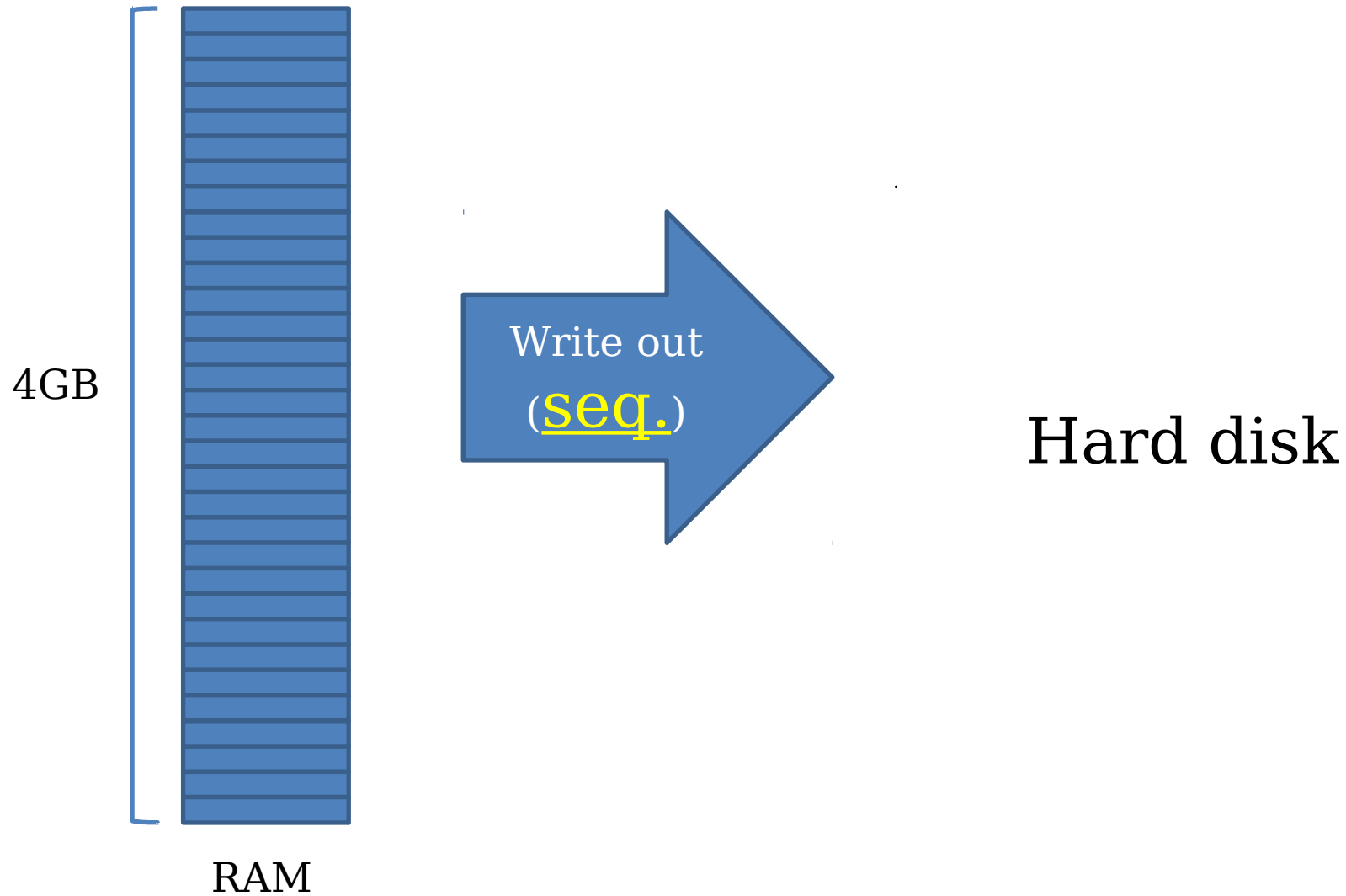


ARM Hibernation Solutions

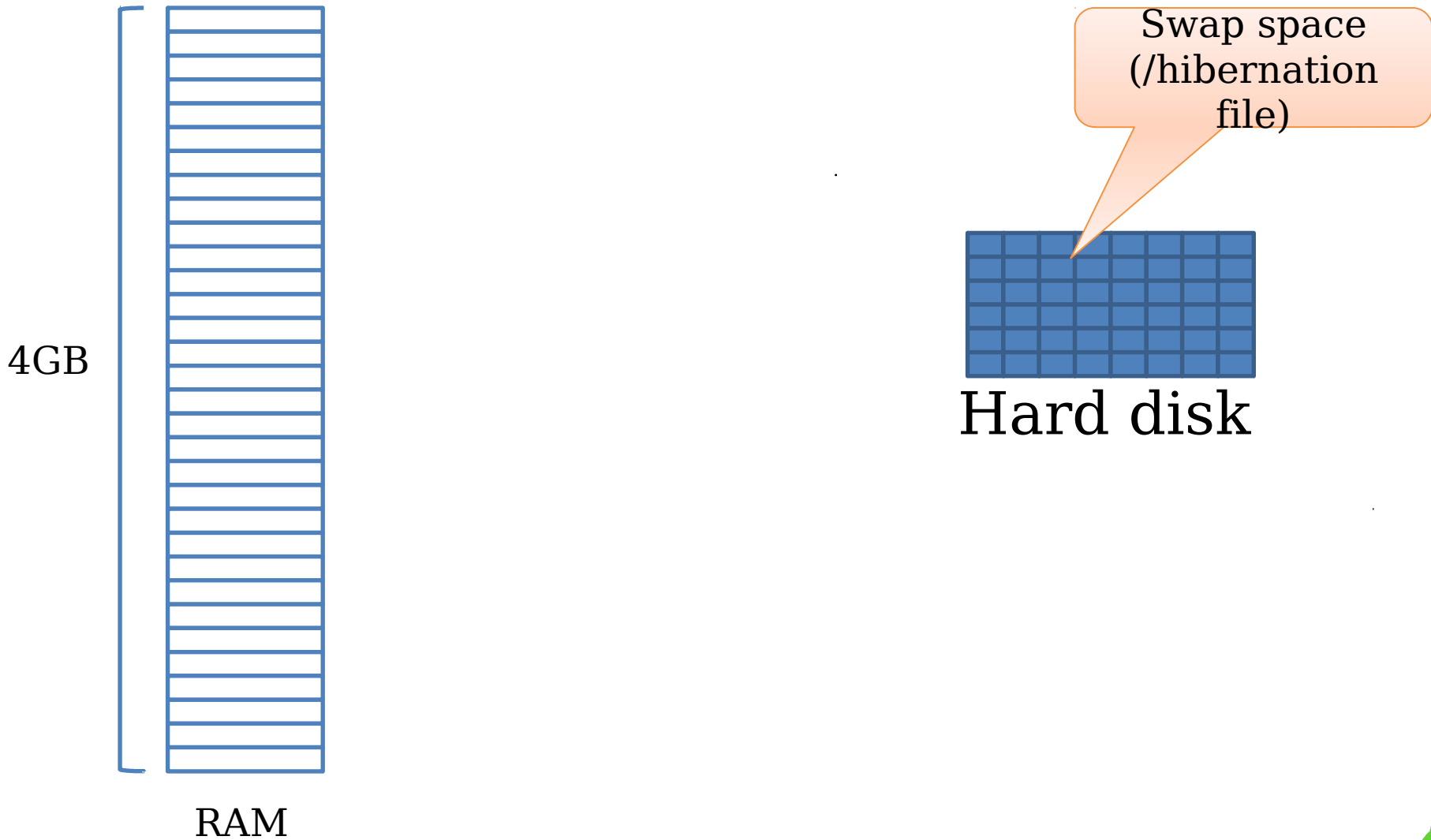
- swsusp hibernation
 - Reference: “Extending the swsusp Hibernation Framework to ARM”, Russell Dill at ELC 2013
- QuickBoot (proprietary)
- FastON



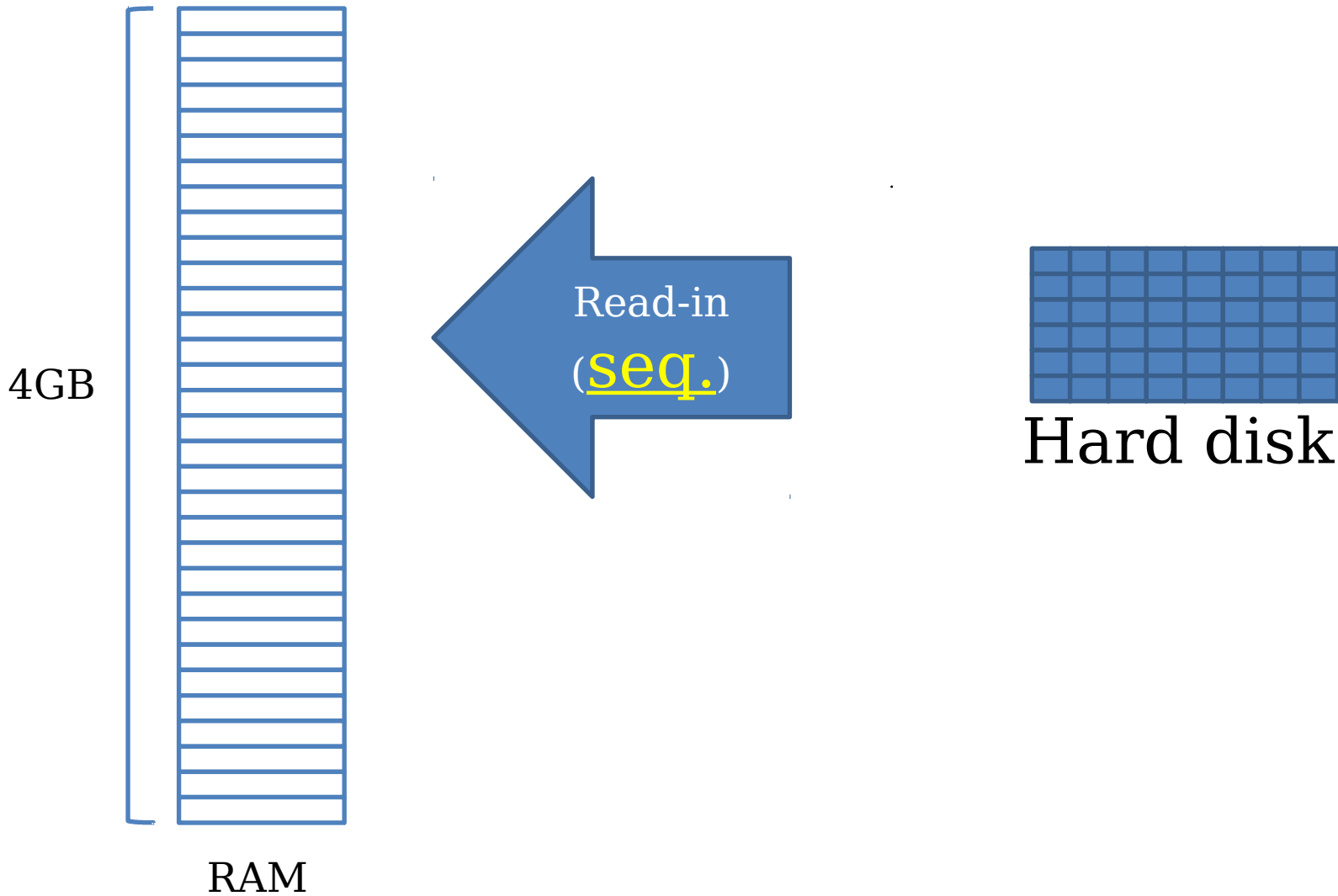
ARM Hibernation Diagram (1)



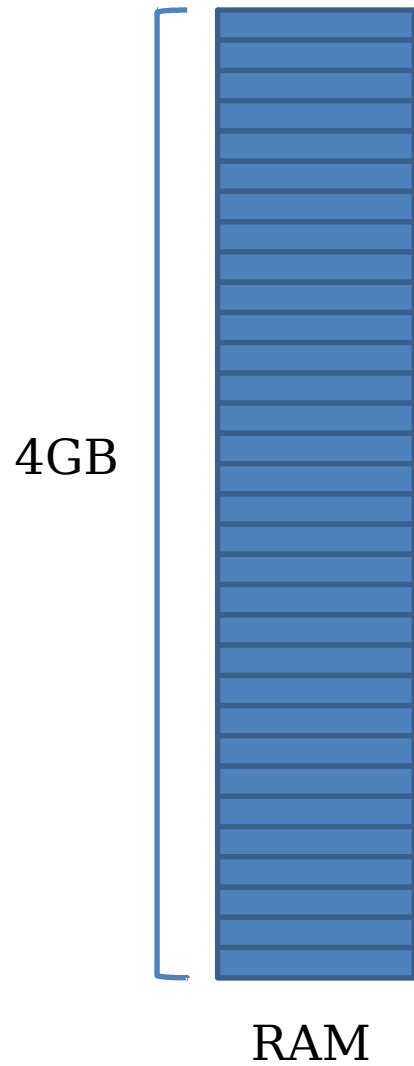
ARM Hibernation Diagram (2)



ARM Hibernation Diagram (3)



ARM Hibernation Diagram (4)

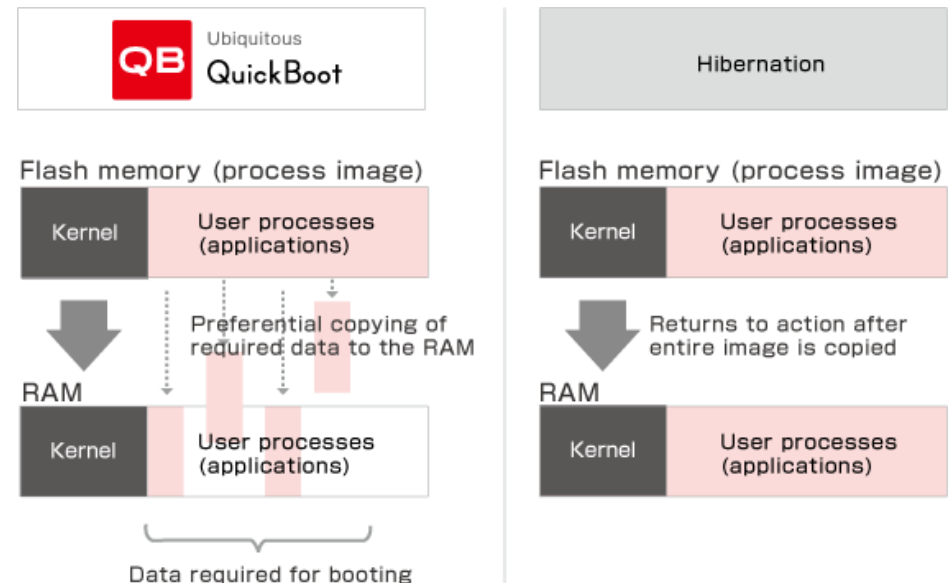


Hard disk



QuickBoot

- Developed by Japanese company, Ubiquitous
- Demand loading of required page from flash
- Requires deep integration of hardware and software
- No visible information;
No production record

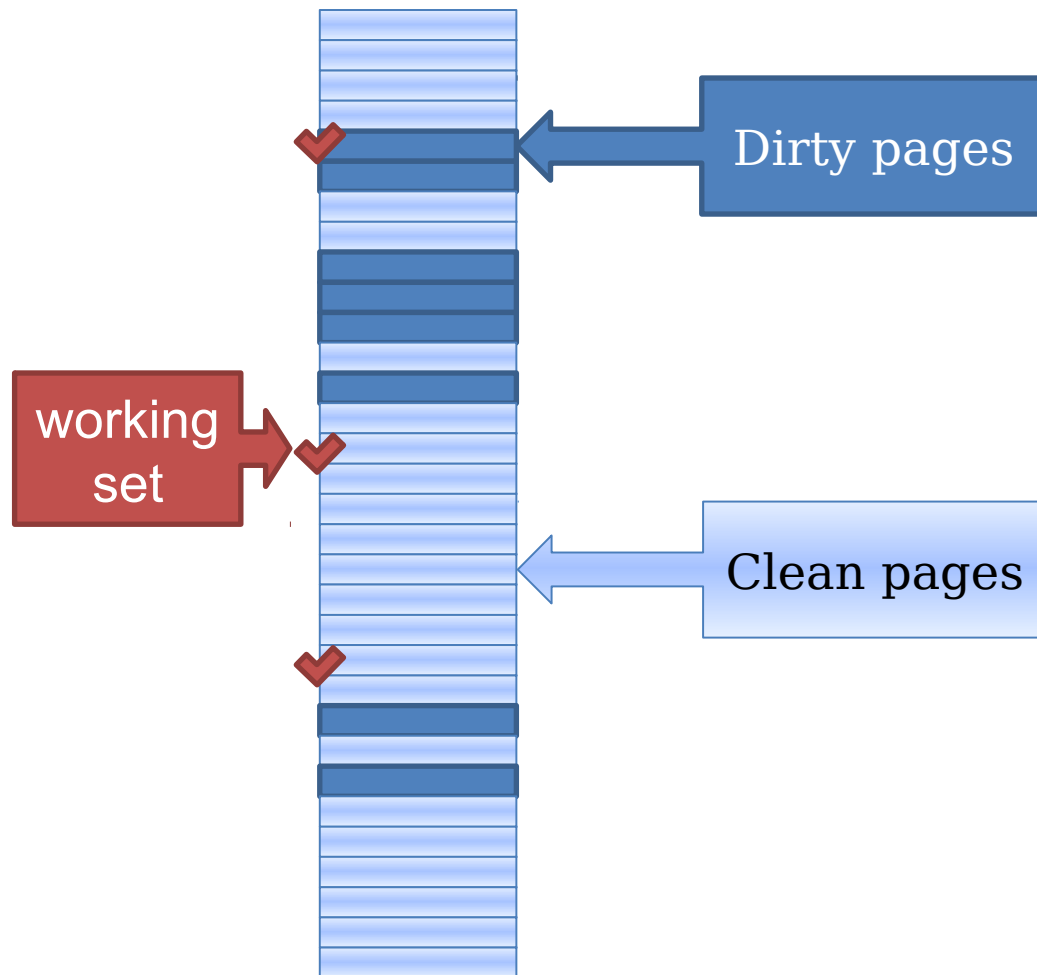


FastON

- Developed by OSLab of National Chung Cheng University in Taiwan
 - Based on existing technologies thus requires little modifications to userspace
 - Release clean-pages before suspend
 - Swap out dirty-pages before save image
 - Image size reduced leads to faster resume time.
- Reference: “*Swap-before-hibernate: a time efficient method to suspend an OS to a flash drive*”, Dr. Shi-wu Lo



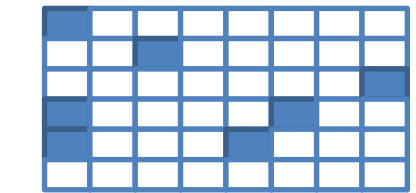
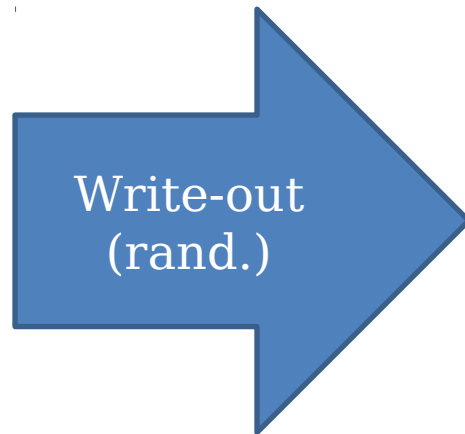
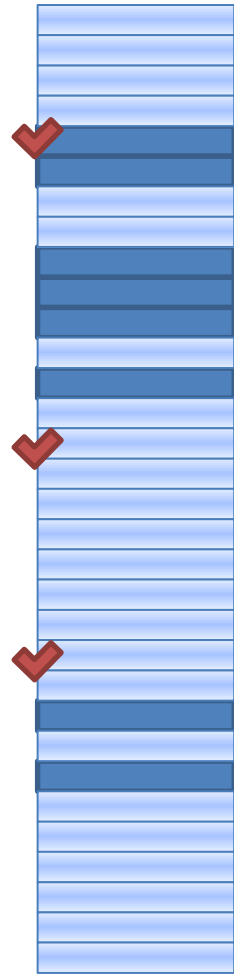
Memory Analysis



1. Most of the memory status is unmodified (clean), with an exact copy in secondary storage.
2. The size of working set is very very small.



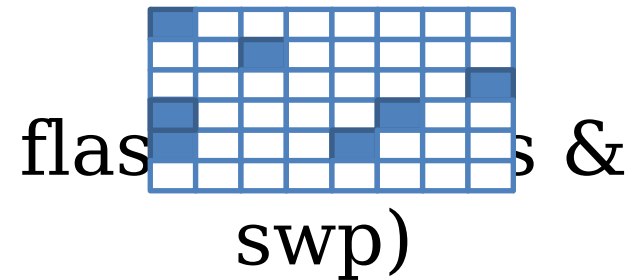
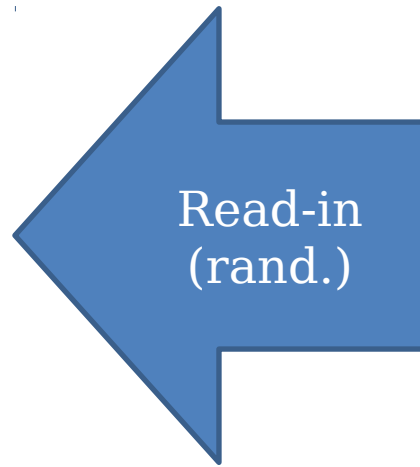
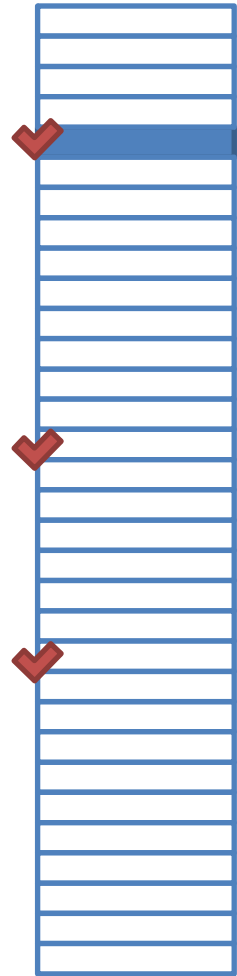
FastON Diagram (1)



flash drive



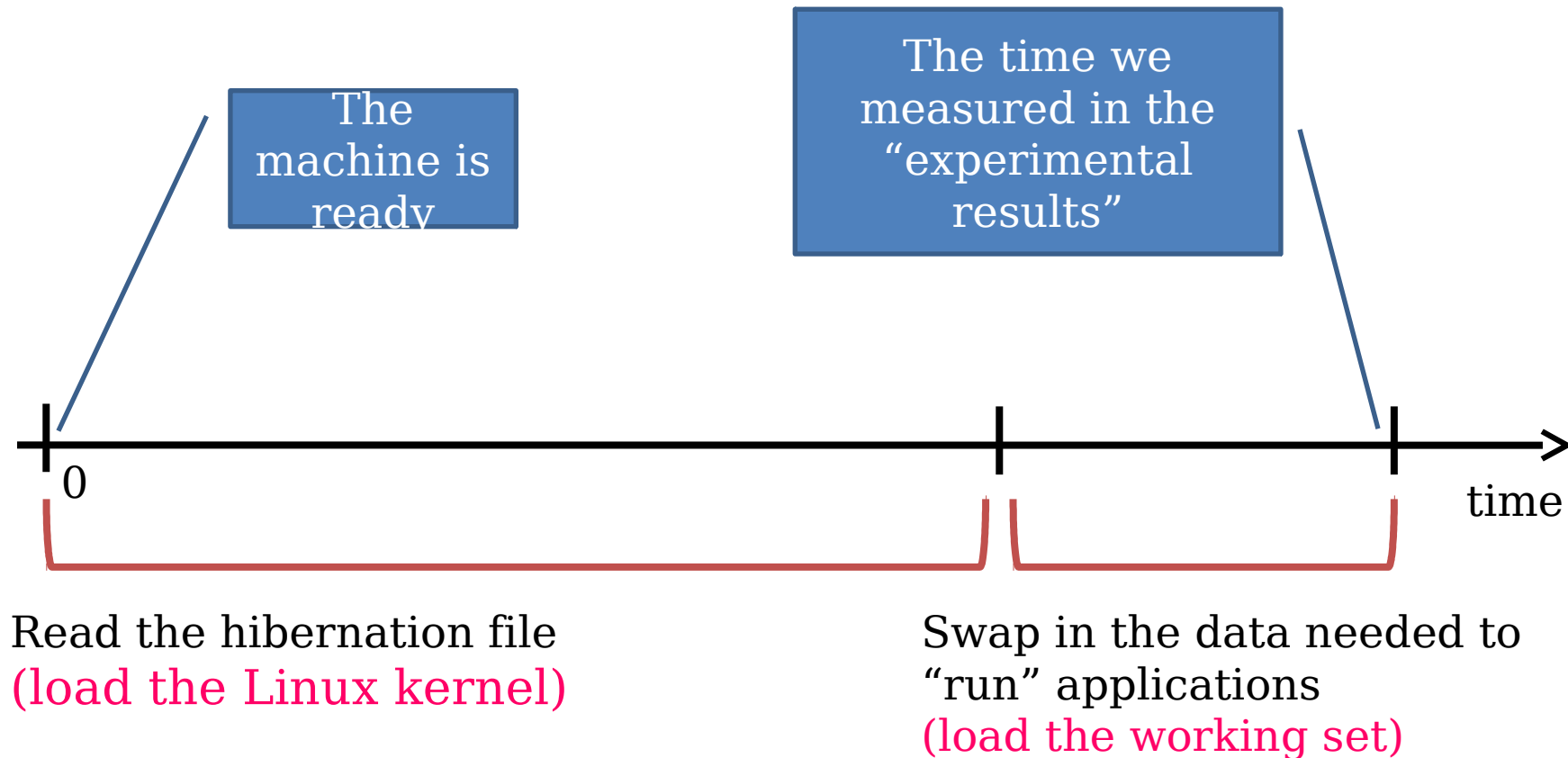
FastON Diagram (2)



Only working set is
read and resumed!



FastON



The definition of "run" is we can start **to operate** the applications **very smoothly**. (for example, scroll down and up, key in)



Boot Time: Original vs. FastON

Original:

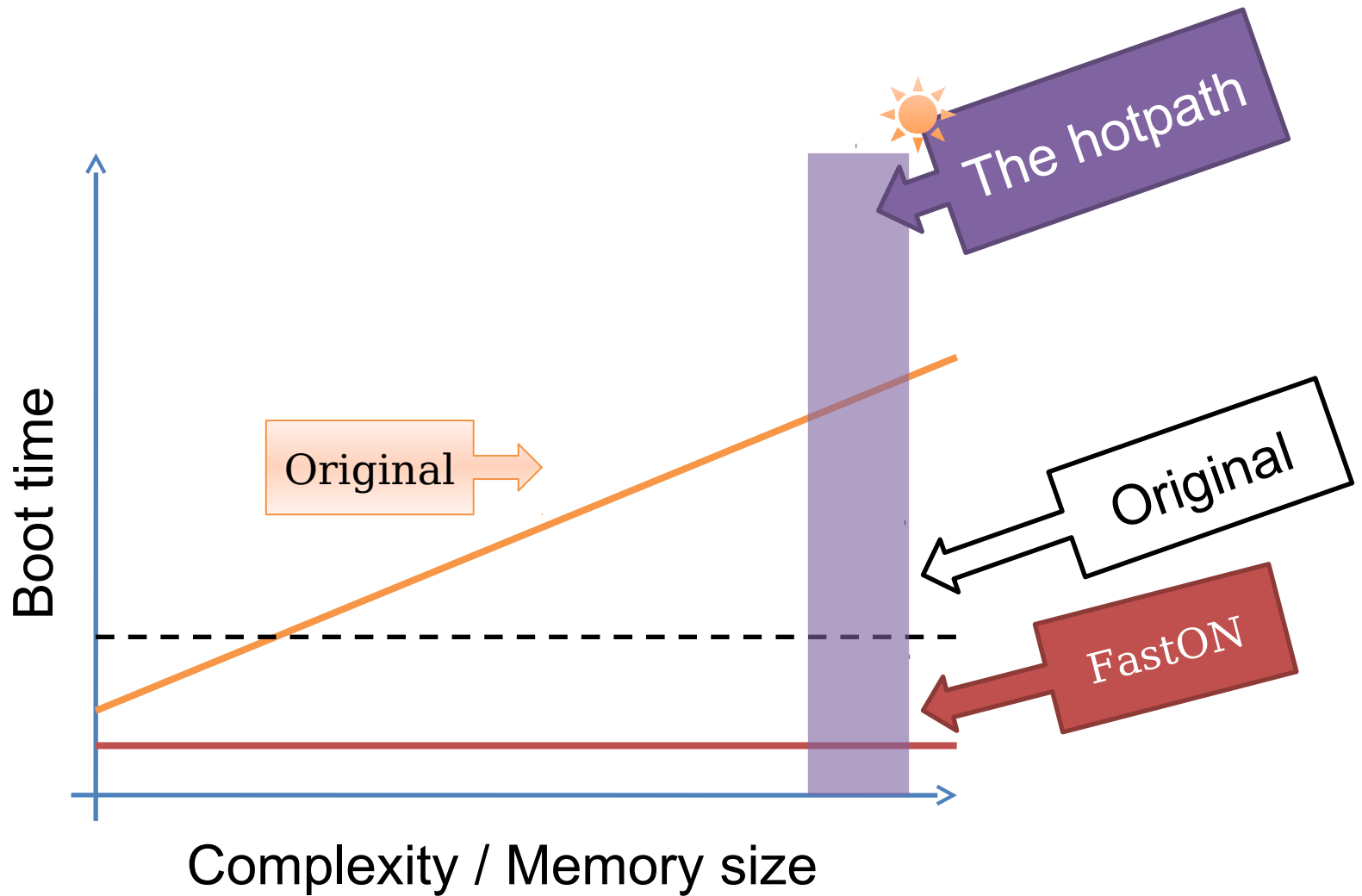
$$\frac{\textit{sizeof}(\textit{main_memory})}{\textit{speed}_{seq.}} + \textit{device_init}$$

FastON:

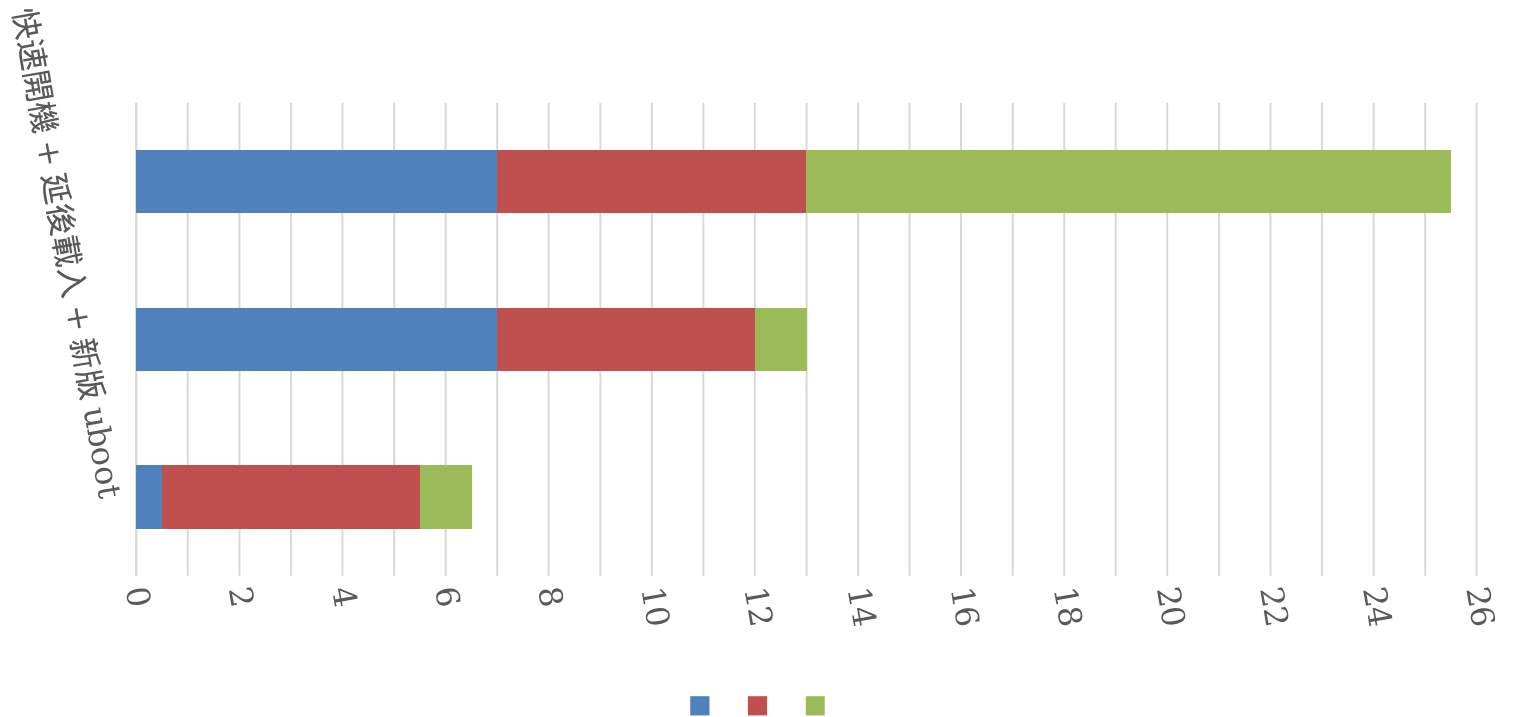
$$\frac{\textit{sizeof}(\textit{hiberfile})}{\textit{speed}_{seq.}} + \frac{\textit{sizeof}(\textit{workingset})}{\textit{speed}_{rand.(16k)}} + \textit{device_init}$$



Boot Time: Original vs. FastON



Boot Time: Original vs. FastON



Android Wakelocks & TuxOnIce





TuxOnIce Patch

- TuxOnIce (was Software Suspend 2) is a hibernation patchset
- Can save images to different locations
- Can use different compression algorithms



Suspend to Disk

```
select 927 (putmethod.latin), adj 1, size 3, to kill
select 950 (ndroid.settings), adj 15, size 1, to kill
select 1081 (m.android.music), adj 15, size 348, to kill
send sigkill to 1081 (m.android.music), adj 15, size 348
select 927 (putmethod.latin), adj 1, size 3, to kill
select 950 (ndroid.settings), adj 15, size 1, to kill
select 1081 (m.android.music), adj 15, size 348, to kill
send sigkill to 1081 (m.android.music), adj 15, size 348
select 927 (putmethod.latin), adj 1, size 3, to kill
select 950 (ndroid.settings), adj 15, size 1, to kill
select 1081 (m.android.music), adj 15, size 348, to kill
send sigkill to 1081 (m.android.music), adj 15, size 348
select 927 (putmethod.latin), adj 1, size 3, to kill
select 950 (ndroid.settings), adj 15, size 1, to kill
select 1081 (m.android.music), adj 15, size 348, to kill
send sigkill to 1081 (m.android.music), adj 15, size 348
done
PM: Wrote 126216 kbytes in 30.01 seconds (4.20 MB/s)
PM: S|
Power down.
System halted.
PM: Please power down manually
```

Resume

```
Post atomic.
Reading caches...
...20%...40%...60%...80%
Cleaning up...
Restarting all filesystems ...
Restarting tasks ... Restarting tasks ...
usb 1-2: USB disconnect, address 2
usb 1-2.1: USB disconnect, address 3
usb0: unregister 'smsc95xx' usb-ehci-omap.0-2.1, smsc95xx USB 2.0 Ethernet
done.
TuxOnIce debugging info:
- TuxOnIce core : 3.2-rc2
- Kernel Version : 2.6.32
- Compiler vers. : 4.4
- Attempt number : 1
- Parameters : 0 663552 0 0 -2 0
- Overall expected compression percentage: 0.
- Compressor is 'lzo'.
  Compressed 210587648 bytes into 74864969 (64 percent compression).
- Block I/O active.
  Used 18432 pages from swap on /dev/block/mmcblk0p3.
- Max outstanding reads 403. Max writes 6120.
  Memory_needed: 1024 x (4096 + 200 + 72) = 4472832 bytes.
  Free mem throttle point reached 256.
- Swap Allocator enabled.
  Swap available for image: 68274 pages.
- I/O speed: Write 6 MB/s, Read 22 MB/s.
- Extra pages : 0 used/2000.
- Result : Succeeded.
#
#
```

Android Wakelocks

- An aggressive approach to save device power
- Use wakelocks to prevent device going suspend
- Porting TOI to Android has to deal with wakelocks because MMC driver might hold a wakelock
- Source available: <http://gitorious.org/0xlab-kernel>

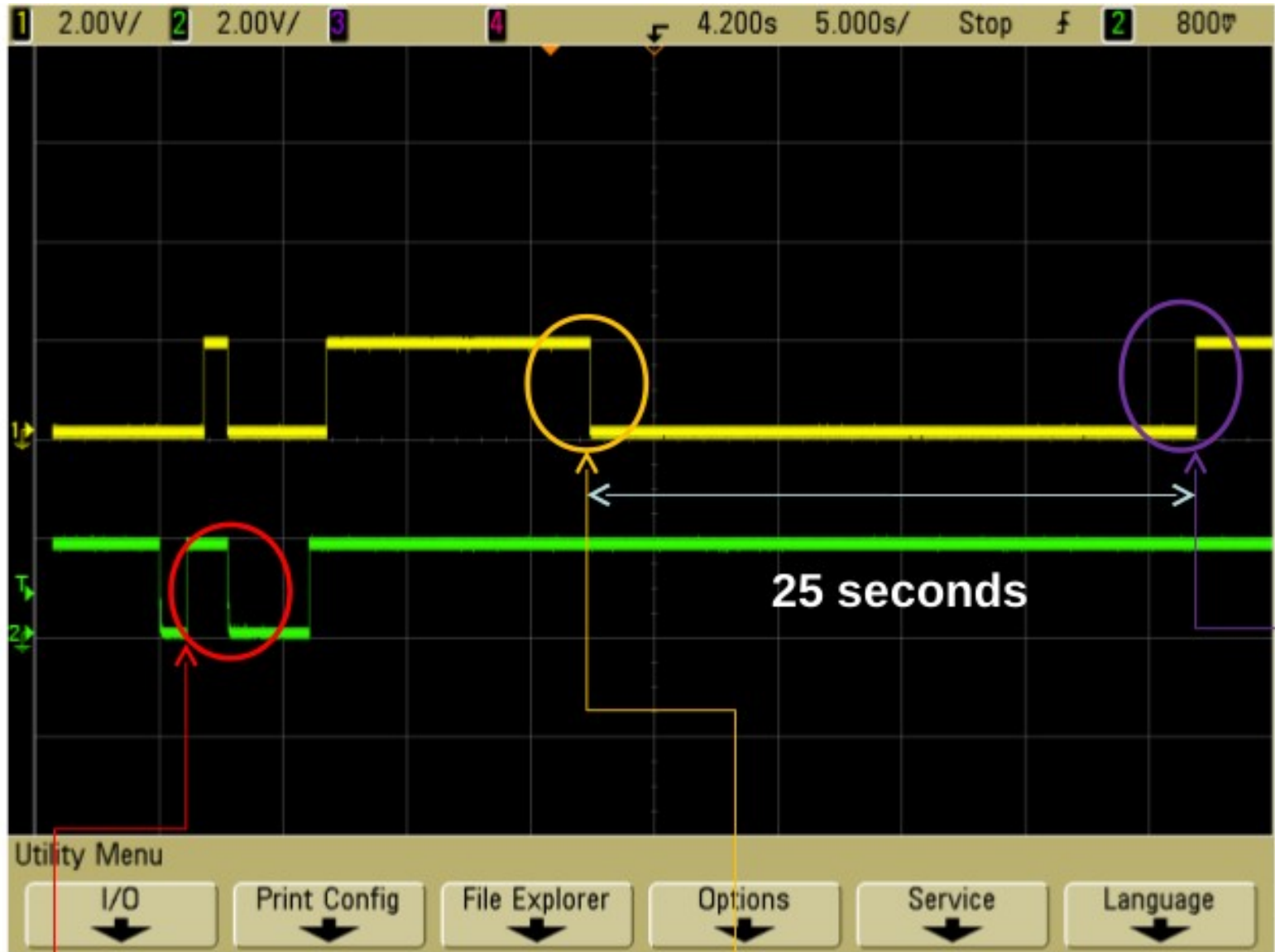


Example: Android on Beagleboard

- OMAP353x; ARM Cortex-A8 600 Mhz; 256MB RAM
- Kernel: 2.6.32 + TOI patch
- Bootloader: Qi
- Android 2.x system



Original Boot Sequence

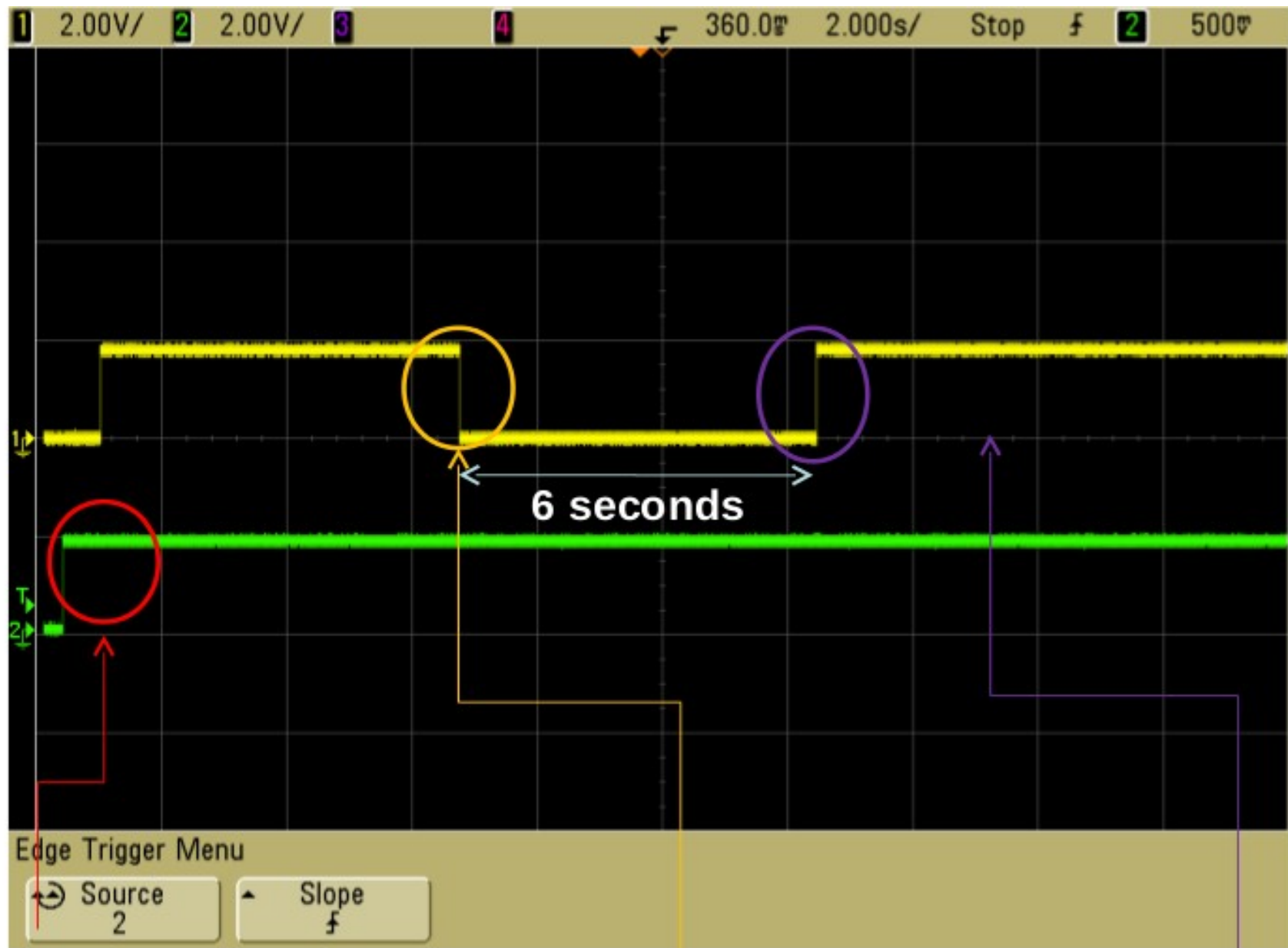


GPIO signal of
Key-guard event

GPIO signal of
stepping into kernel

Power On Reset

Hibernation Boot



Power On Reset

GPIO signal of
stepping into kernel

GPIO signal of
Key-gurad event

Further optimizations:
No Need for full-featured
Bootloader when resuming



R-Loader

- Normal suspend-to-disk approach has many duplicated effort
 - **Boot-loader** inits some hardwares
 - **Boot-loader** loads the normal kernel image
 - **Kernel** inits some hardwares again
 - **Kernel** loads the suspended kernel image
 - **Kernel** resumes, inits some hardwares again



R-Loader

- 0xlab proposed “Resume-Loader”
 - A customized “snapshot” boot
- R-Loader inits some hardware then reads the suspended kernel image as fast as possible
- Jump directly to the resume point
- Kernel will takeover the job and inits reset hardwares



Advanced Topics for integrating Hibernation with Android

- Save the heap image (like core dump) of Zygote after preloading classes
- Modify Dalvik to make hibernation image after system init and before Launcher startup
- Parallize Android init
- Cache & Share JITed code fragment



Part II:

Userspace solution: Checkpointing



Hibernation looks fine, why another?

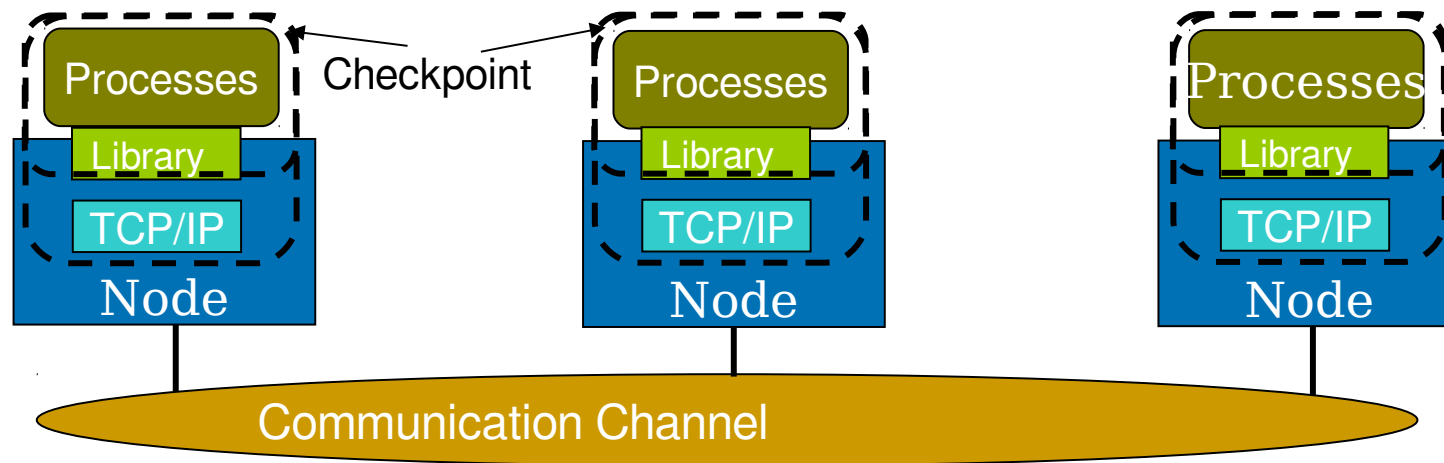
- Hibernation is sometimes harmful to userspace especially for recovering from file system or updating packages, that is crucial for Linux-based “smart” devices
- Post-actions are still necessary: connectivity, firmware management, security, etc.
- Broken drivers are impossible to work with hibernation framework, but they usually appear from chip vendors though.



Basic Idea of checkpointing:

Process Migration

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



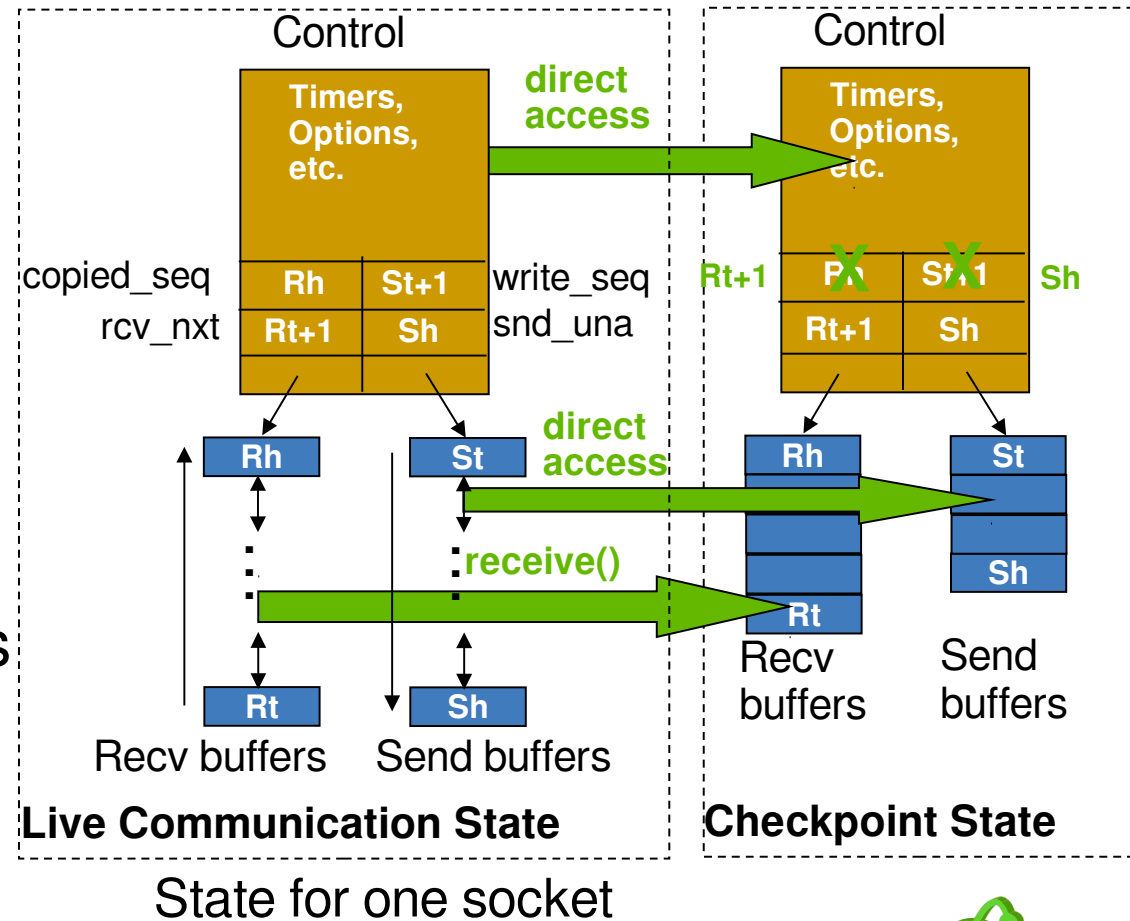
Ideas about Checkpointing for Android/Linux

- 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



Communication state checkpoint

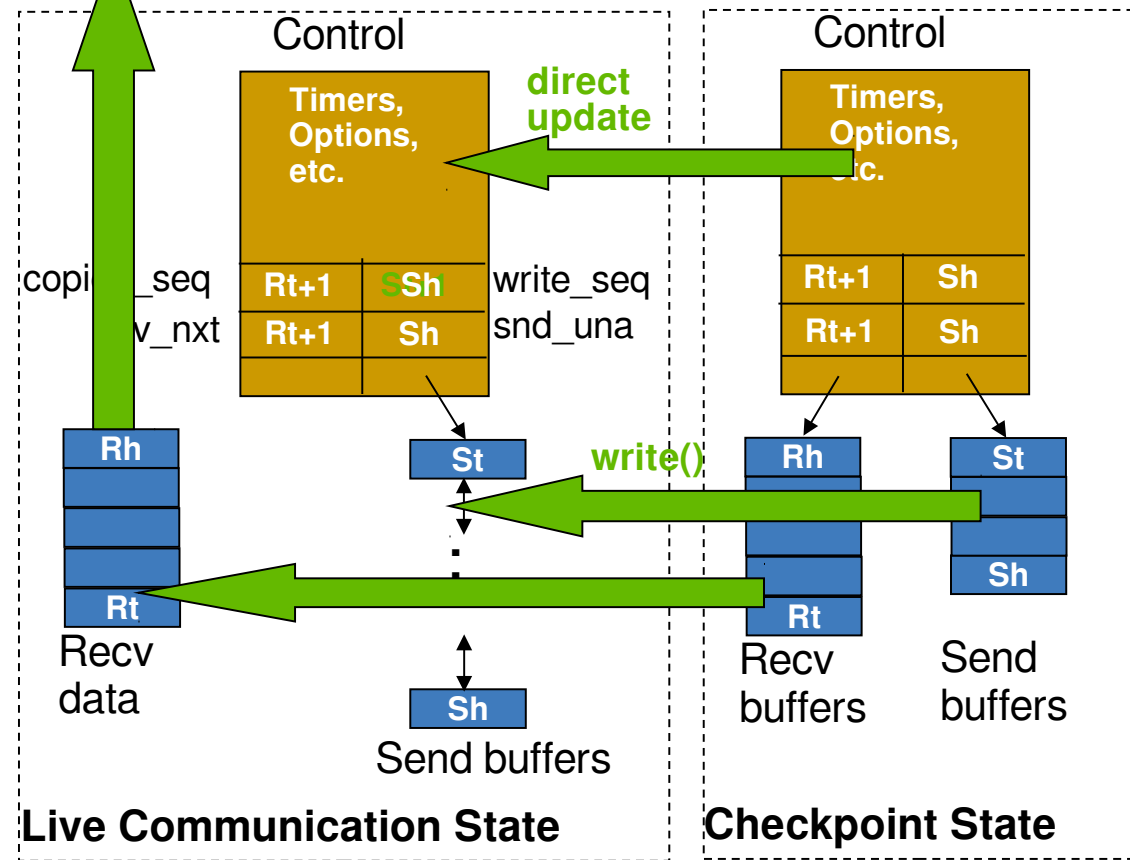
- Acquire network stack locks to freeze TCP processing
- Save receive buffers using socket receive system call in peek mode
- Save send buffers by walking kernel structures
- Copy control state from kernel structures
- Modify two sequence numbers in saved state to reflect empty socket buffers



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

To App by intercepted
receive system call



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

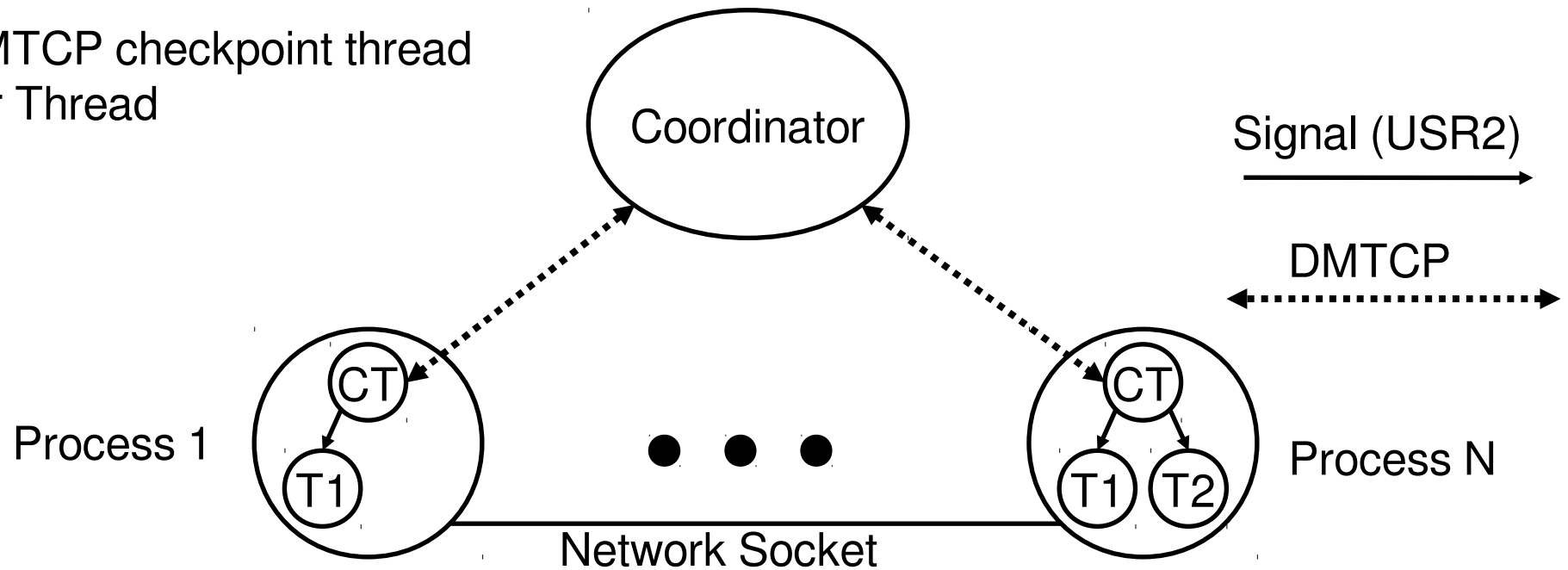


- **D**istributed **M**ulti-**T**hreaded **C**heck**P**ointing.
- Works with Linux Kernel 2.6.9 and later.
- Supports sequential and multi-threaded computations across single/multiple hosts.
- Entirely in user space (no kernel modules or root privilege).
 - Transparent (no recompiling, no re-linking).
- Developed 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



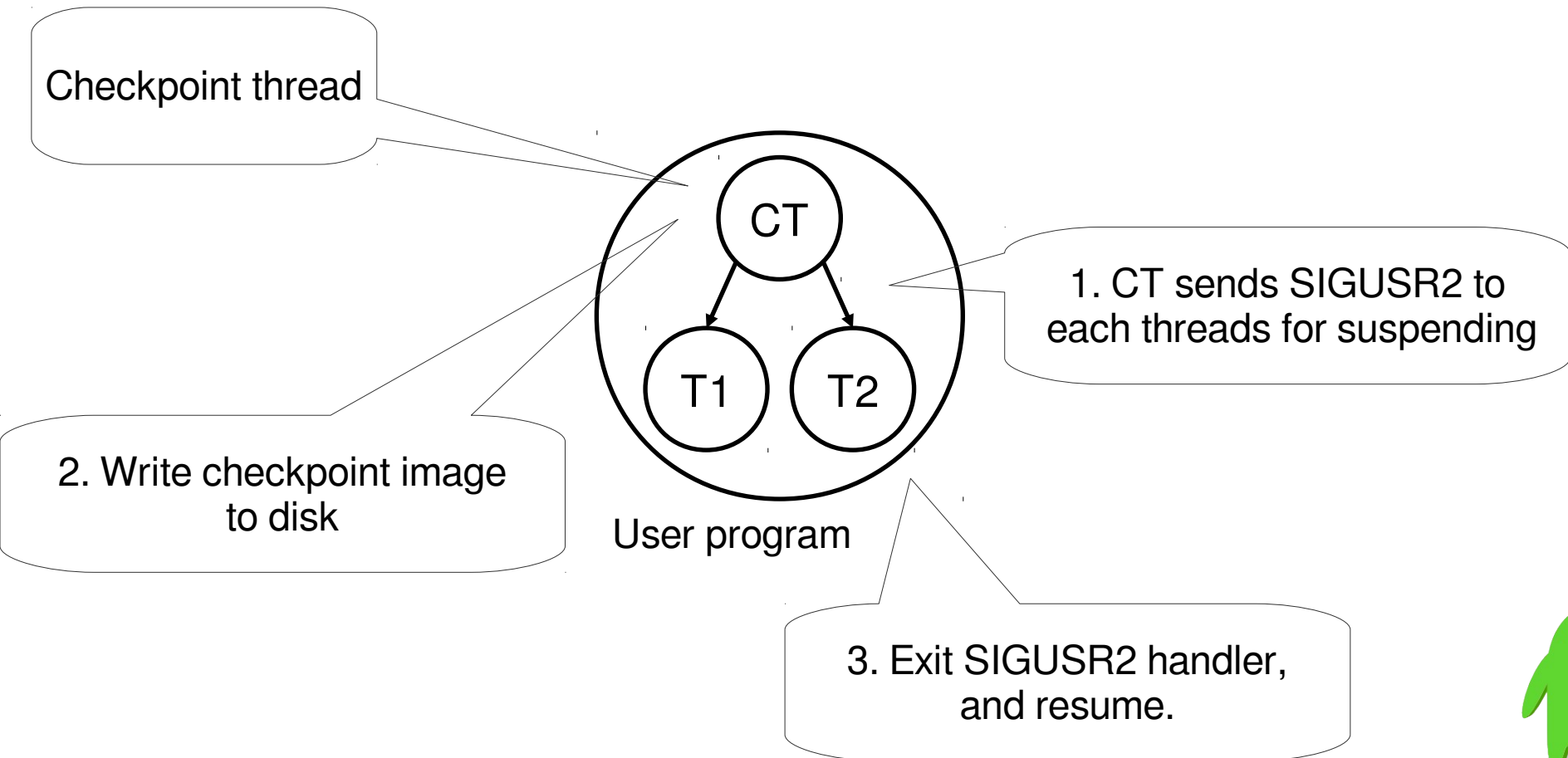
```
dmtcp_checkpoint <EXE> # starts coordinator  
dmtcp_command -c      # talks to coordinator  
dmtcp_restart ckpt_<EXE>-* .dmtcp
```

- 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.



How DMTCP works (3/4)

- Additional wrappers when process id & thread id virtualization is enabled
 - getpid, getppid, gettid, tcgetpgrp, tcsetpgrp, getgrp, setpgrp, getsid, setsid, kill, tkill, tkill, 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



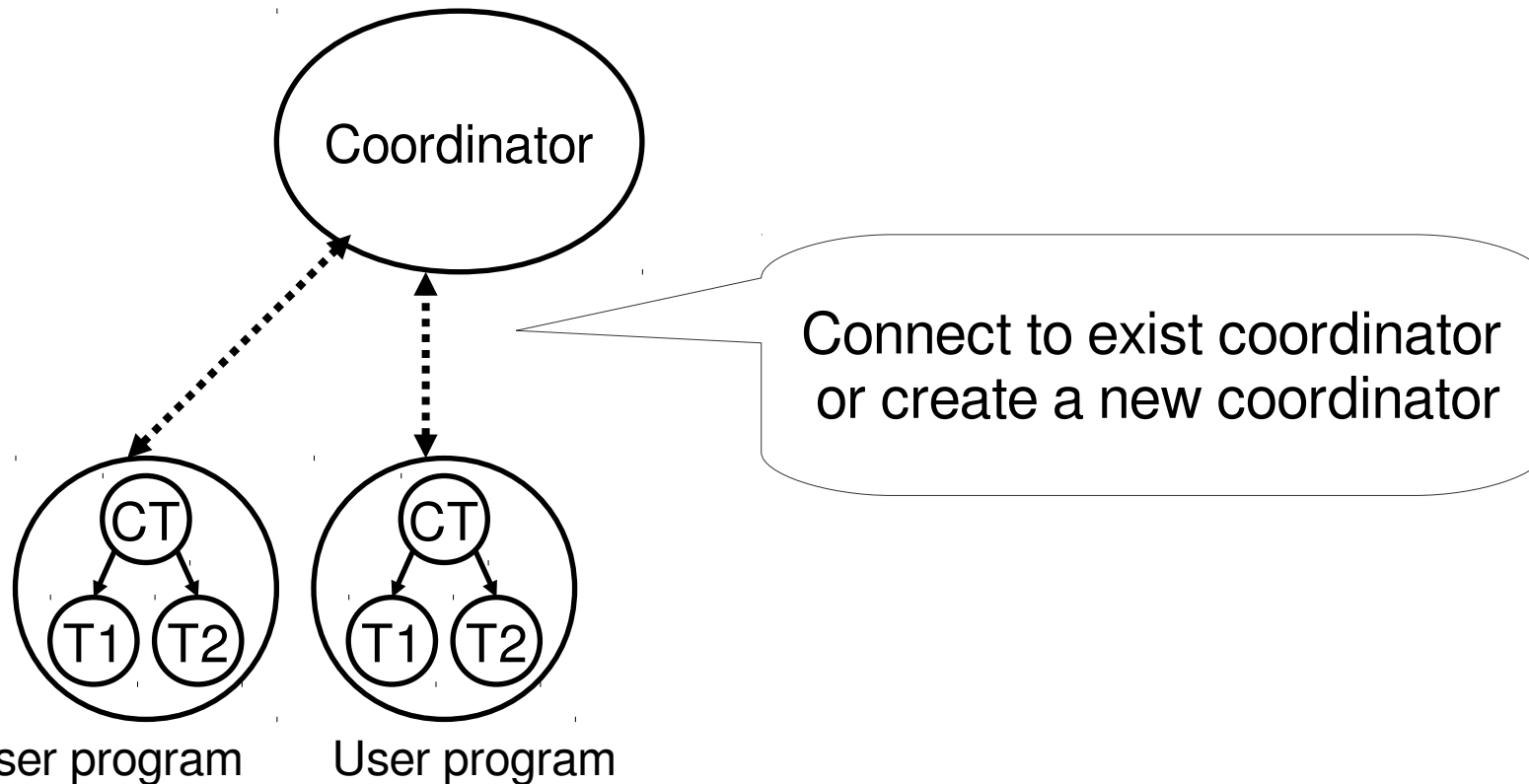
How DMTCP works (4/4)

- 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.



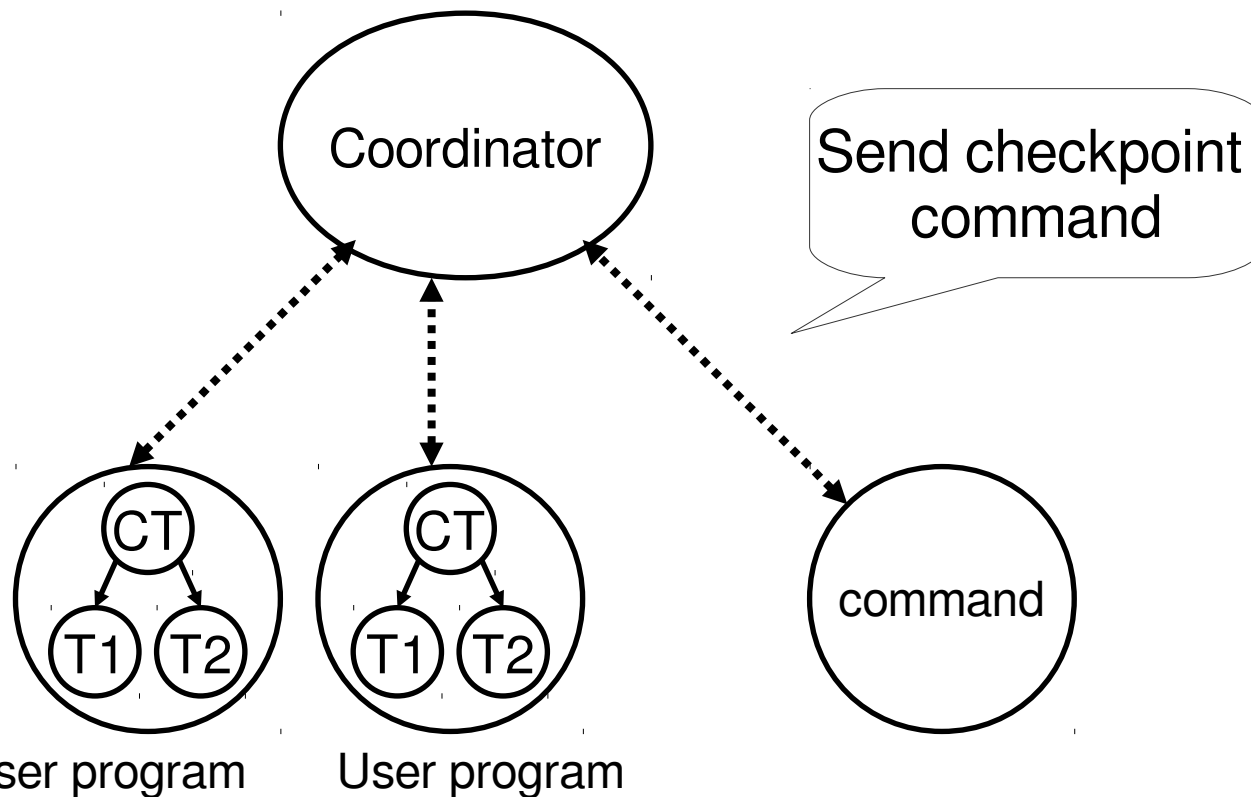
Checkpoint under DMTCP_(1/7)

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



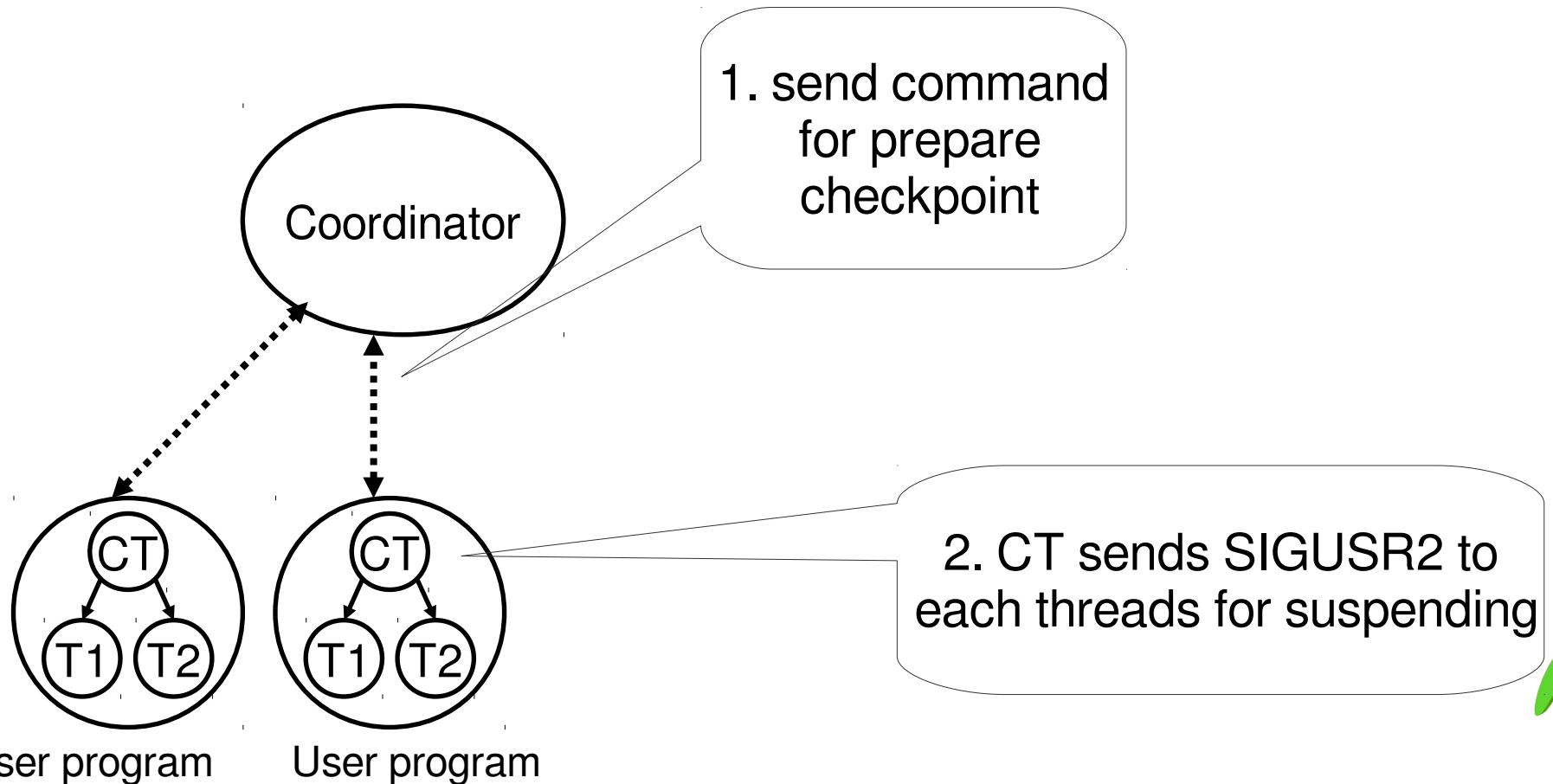
Checkpoint under DMTCP_(2/7)

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



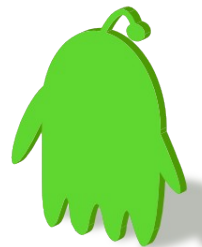
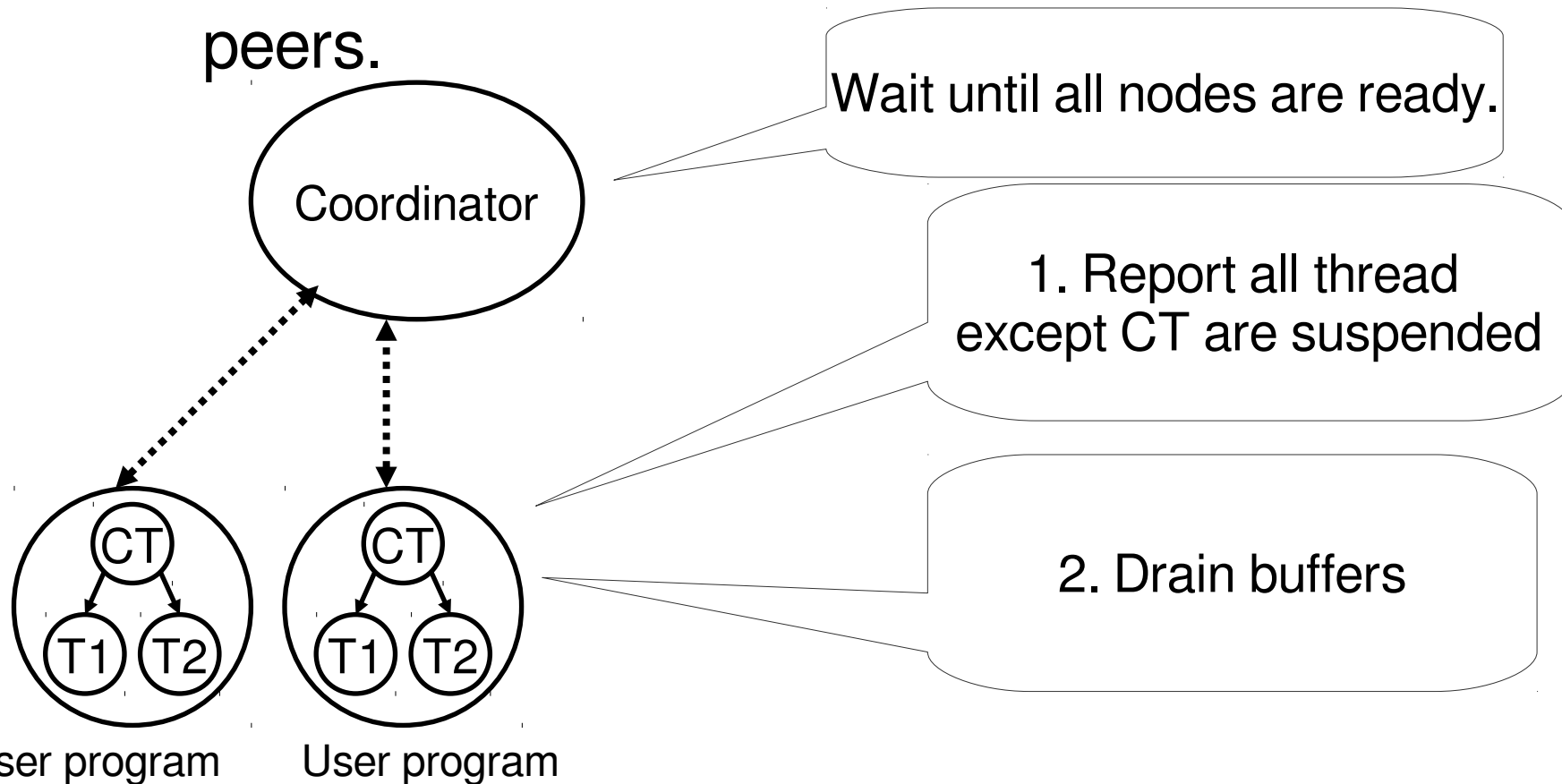
Checkpoint under DMTCP_(3/7)

- Suspend user threads with SIGUSR2.



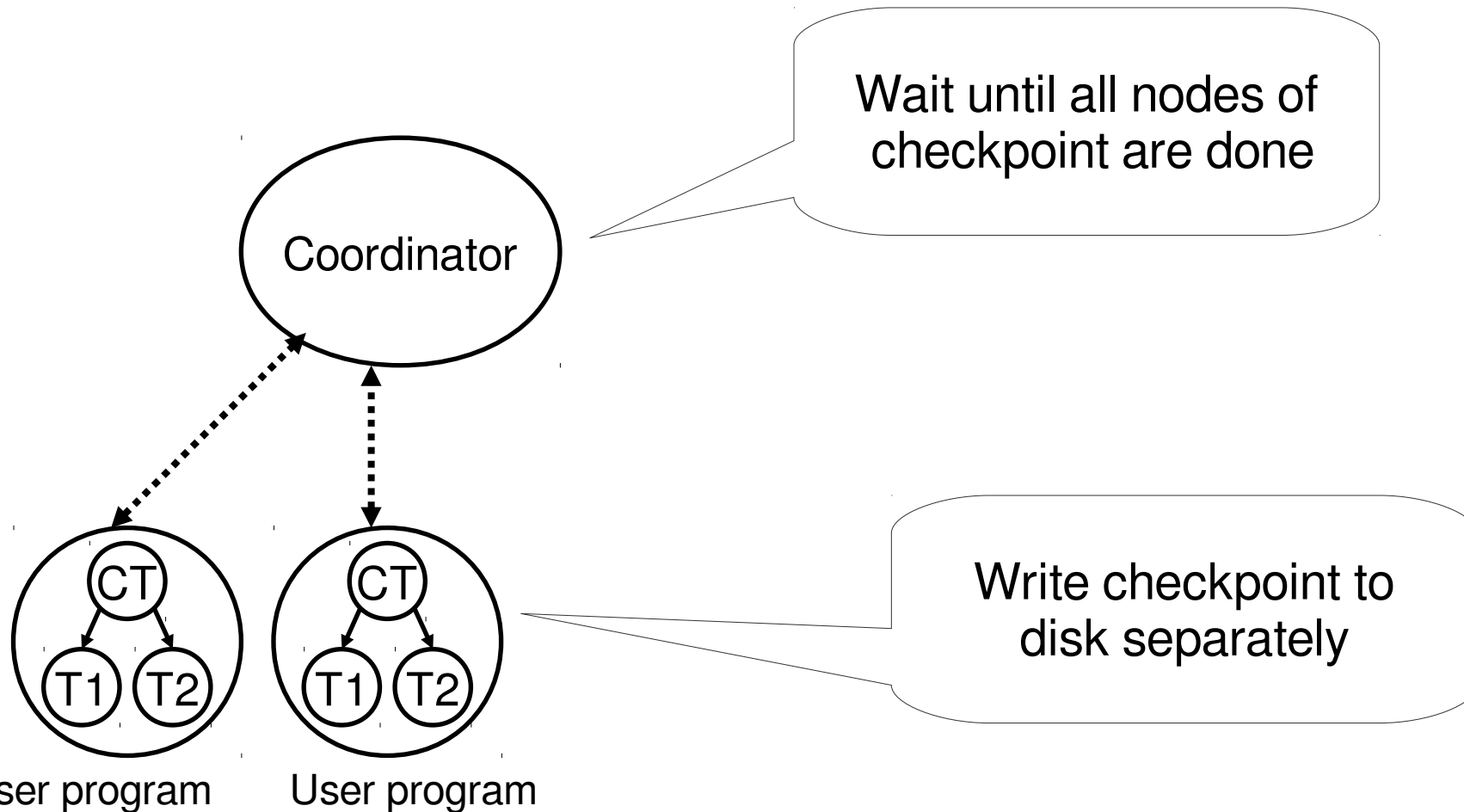
Checkpoint under DMTCP_(4/7)

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



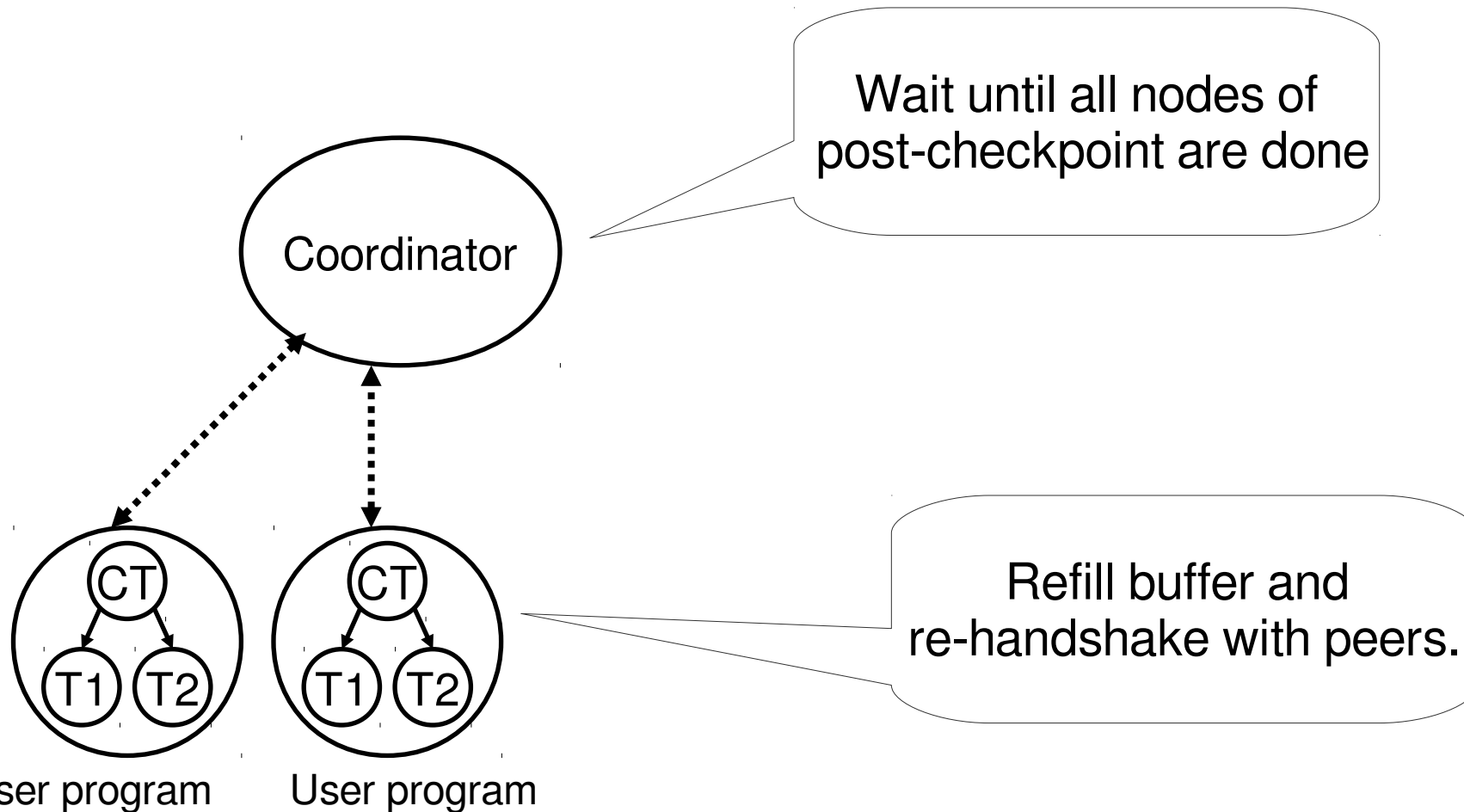
Checkpoint under DMTCP_(5/7)

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



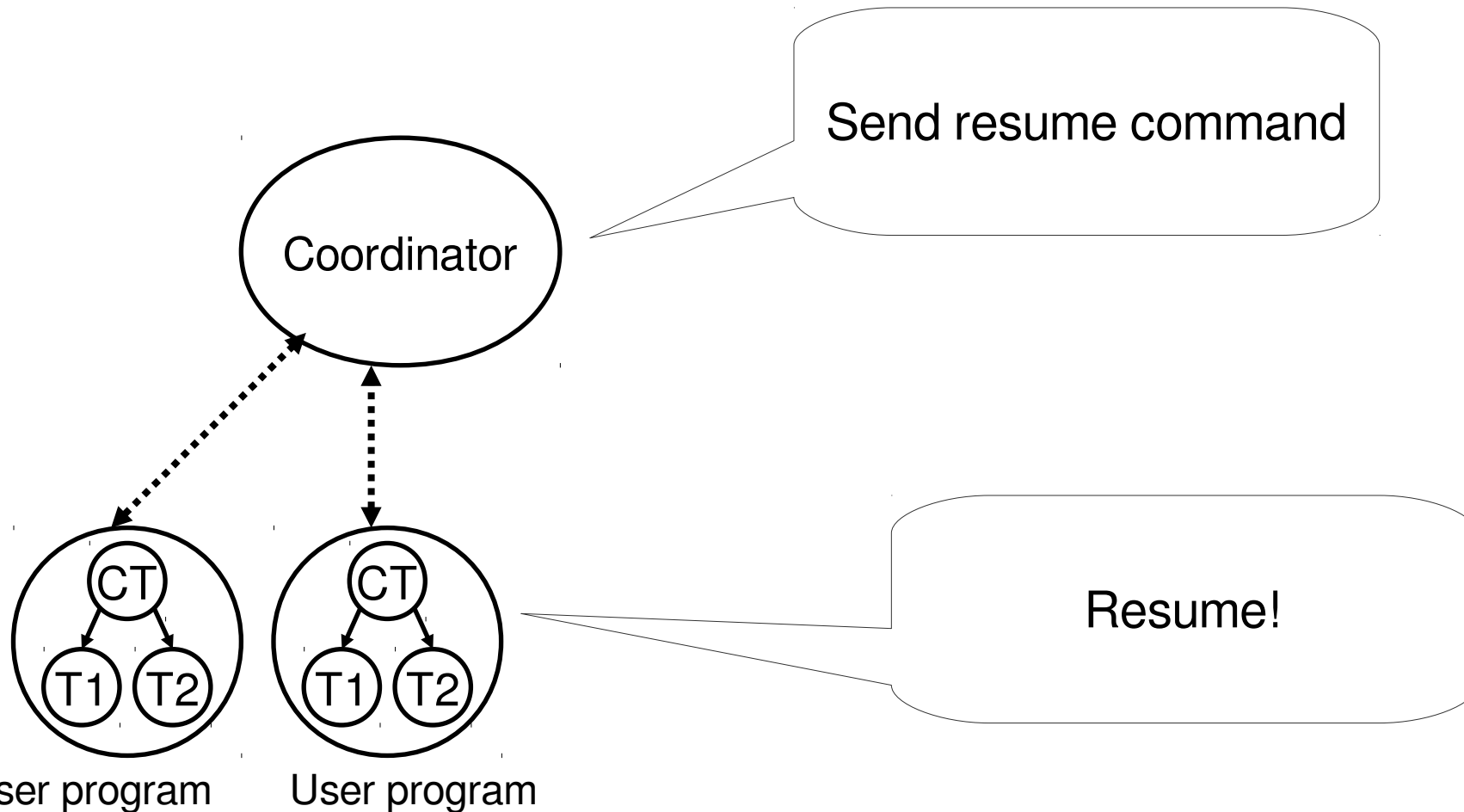
Checkpoint under DMTCP_(6/7)

- Post-Checkpoint stage
- Refill kernel buffers



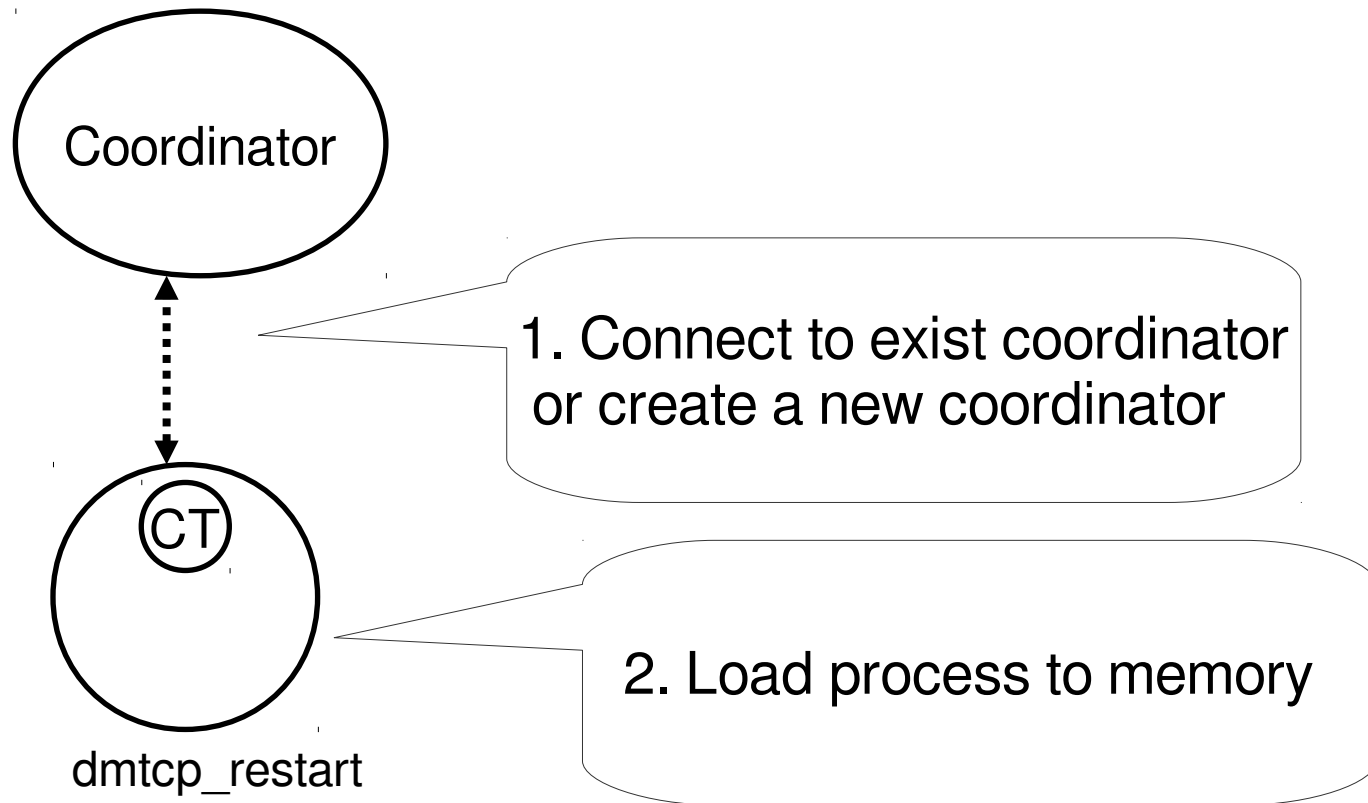
Checkpoint under DMTCP_(7/7)

- Resume user threads.



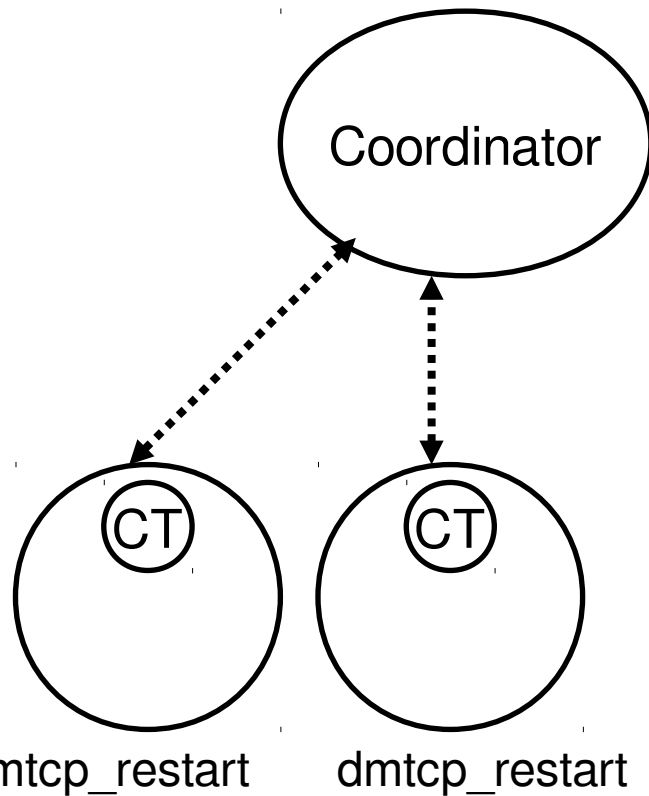
Restart under DMTCP_(1/6)

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



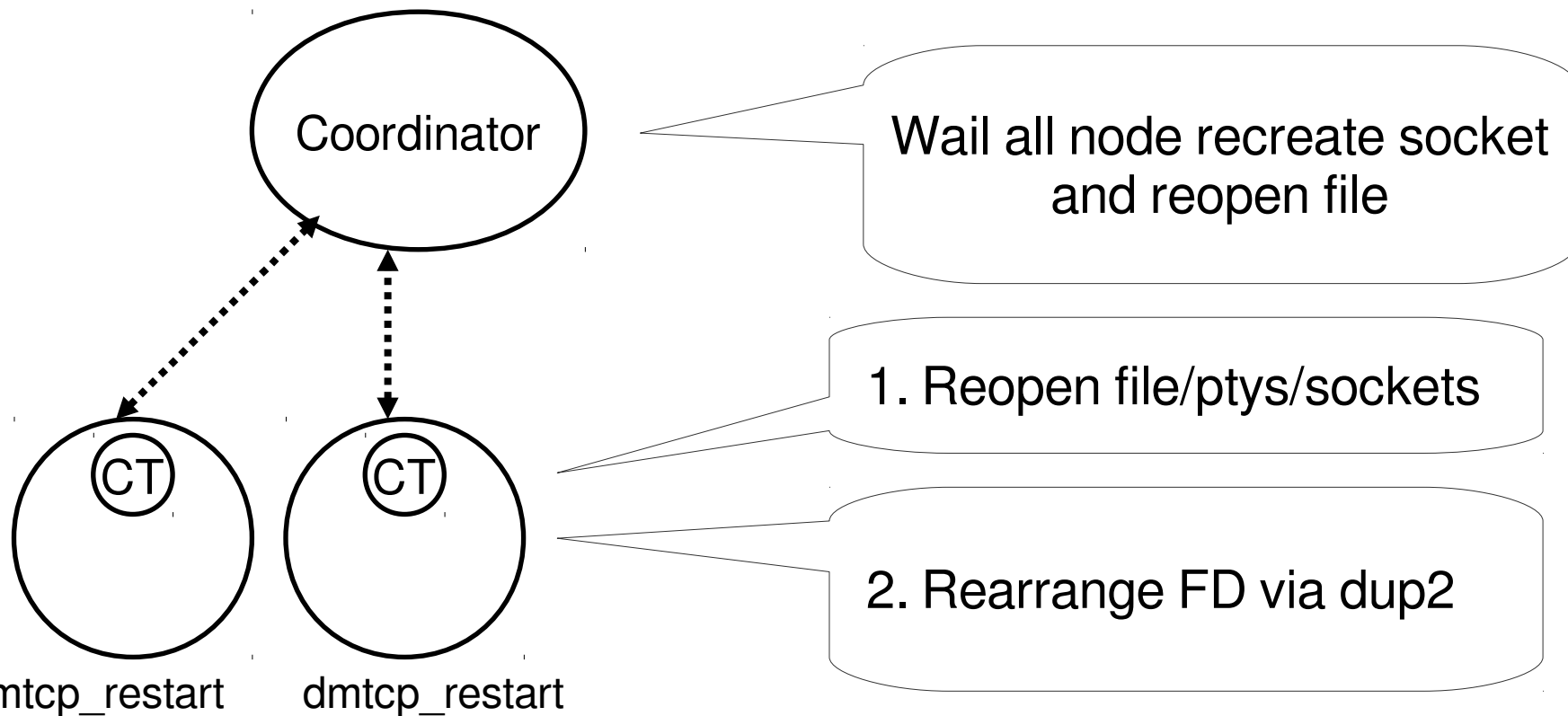
Restart under DMTCP_(2/6)

- Fork user program



Restart under DMTCP_(3/6)

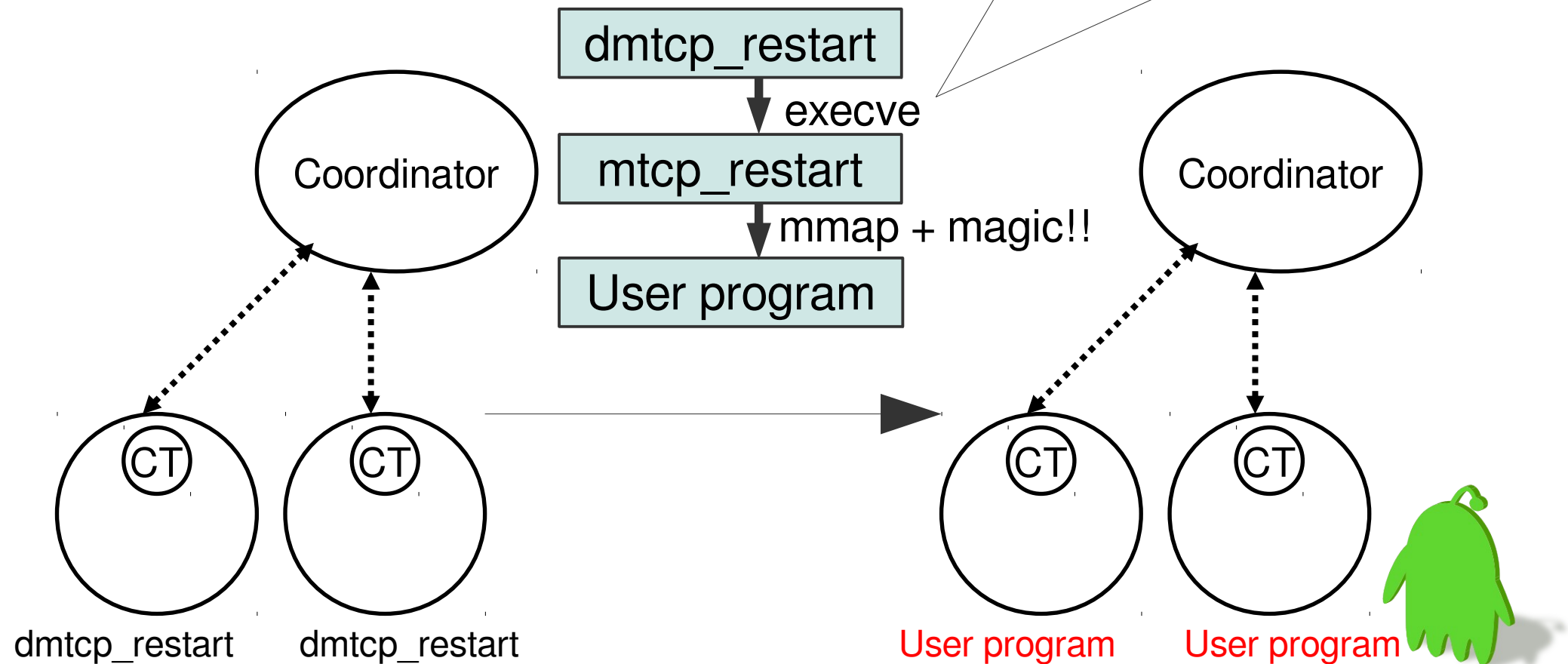
- Reopen files and recreate ptys
- Recreate and reconnect sockets
- Rearrange file descriptors to initial layout



Restart under DMTCP_(4/6)

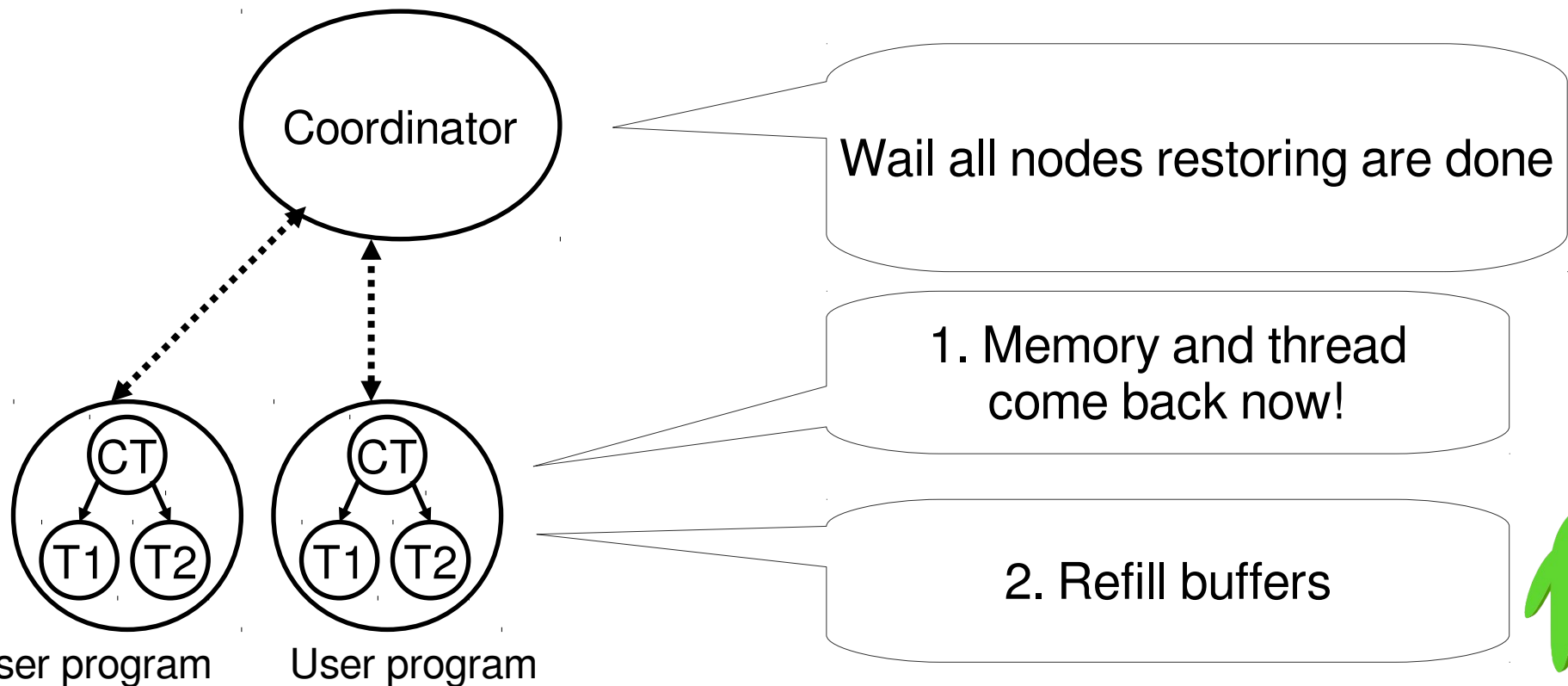
- Restore memory content.
- Restore stack status for checkpoint thread.

FDs will be preserved across execve!



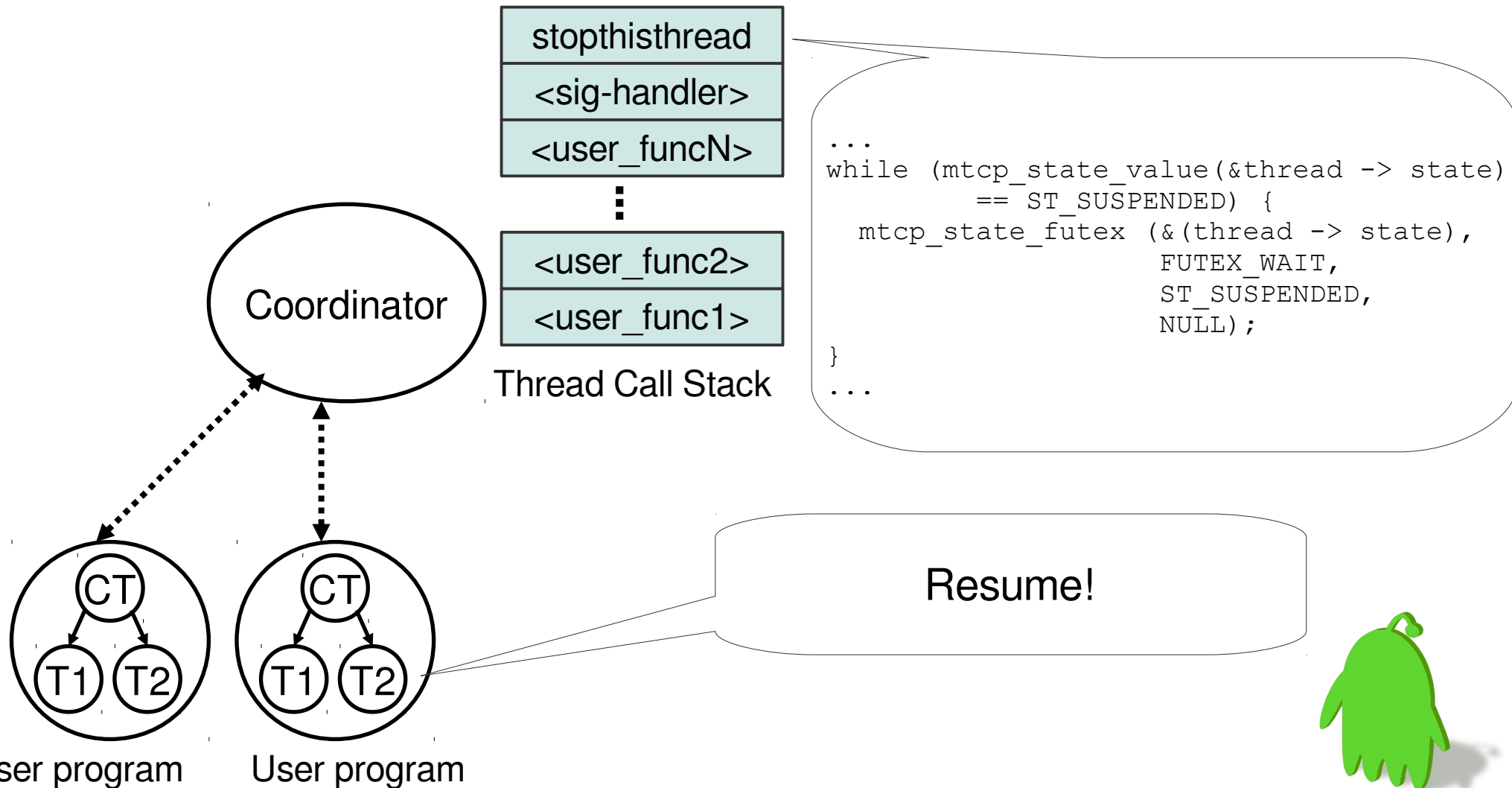
Restart under DMTCP_(5/6)

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

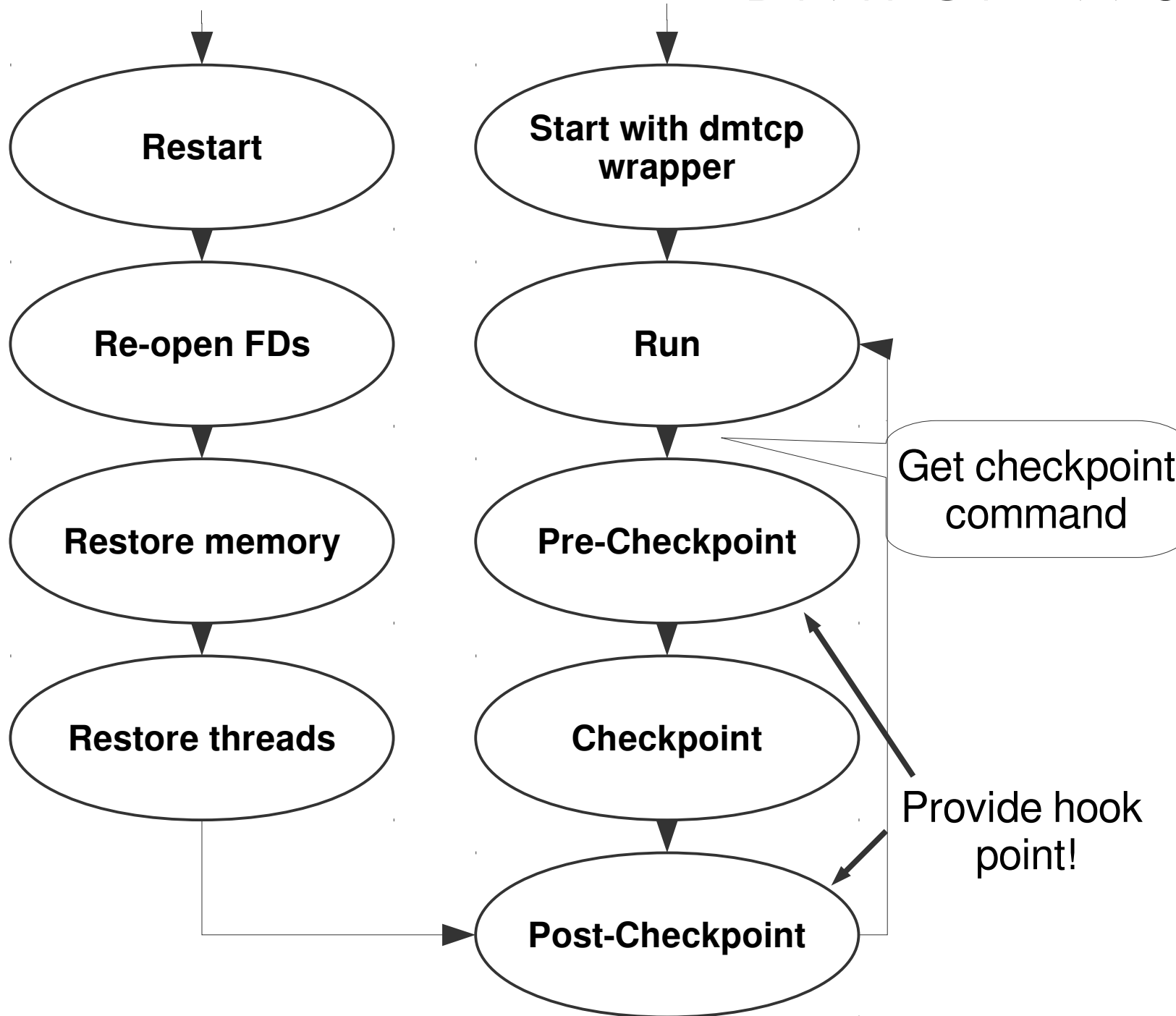


Restart under DMTCP_(6/6)

- 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

Already merged in upstream DMTCP



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.
- 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 on ARM Cortex-A9 dual (1GHz; Memory: 512 MB)

	with gzip	without gzip
Checkpoint time	~8s	~4.5s
Restart time	~0.3s	~0.1s
Image size	~3M	~17M



Observations from logcat

<pre>----- 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 HAL 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 HAL E/BatteryService(1382): usbOnlinePath not found D/AndroidRuntime(1902): D/AndroidRuntime(1902): >>>>> AndroidRuntime START com.android.systemui 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, paused 0ms ... I/Zygote (1902): ...preloaded 379 resources in 548ms. D/dalvikvm(1902): GC_EXPLICIT freed 20K, 1% free 6417K/6467K, paused 0ms I/Zygote (1902): ...preloaded 31 resources in 13ms. D/dalvikvm(1902): GC_EXPLICIT freed 14K, 1% free 6418K/6467K, paused 0ms D/dalvikvm(1902): GC_EXPLICIT freed 5K, 1% free 6412K/6467K, paused 0ms D/dalvikvm(1902): GC_EXPLICIT freed <1K, 1% free 6412K/6467K, paused 0ms I/dalvikvm(1902): System server process 1911 has been created</pre>	<pre>----- 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 HAL 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 HAL D/dalvikvm(1373): GC_EXPLICIT freed 14K, 1% free 6418K/6467K, paused 0ms D/dalvikvm(1373): GC_EXPLICIT freed 5K, 1% free 6412K/6467K, paused 0ms D/dalvikvm(1373): GC_EXPLICIT freed <1K, 1% free 6412K/6467K, paused 0ms I/dalvikvm(1373): System server process 1382 has been created</pre>
--	---

Normal bootup log message

Bootup log message with restart



Part III:

Mixed Model



Mixed Model: Hibernation + Checkpointing

- Basic Idea
 - Swap out dirty-pages before save image
 - Reducing image size leads to faster resume time
 - Only the “static” working set is suspended, other parts utilizes checkpointing
 - Room for keeping __broken__ device drivers
- However, the state machine is relatively intricate because of application framework awareness.



Technical Challenges of Mixed Model

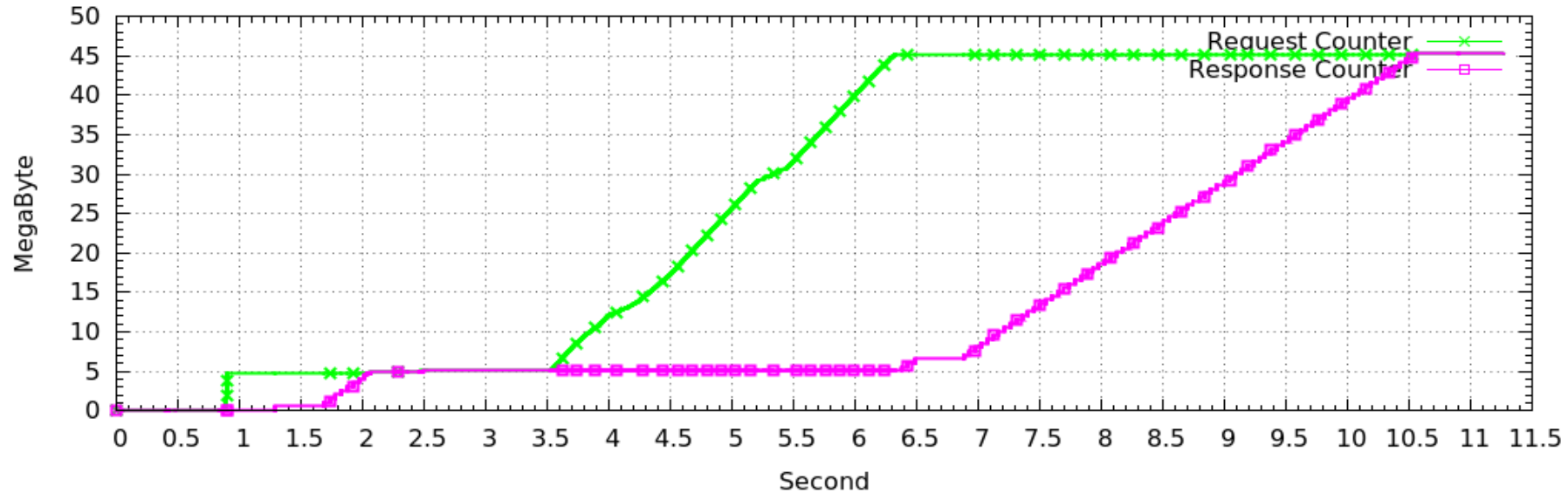
- Have to modify Android/Tizen framework in order to figure the proper checkpoint to perform.
 - It is not straightforward since recent application frameworks depend on web rendering engine as the core component.
 - Working-set is hard to be minimized accordingly.
- Separate the device initializations into two areas:
 - essential zone: taken for Hibernation
 - post-resuming: used for checkpointing, firmware management, and connectivity → problem of maintenance
- Storage access time is crucial since we touch for several times
 - I/O scheduler modifications are under investigation



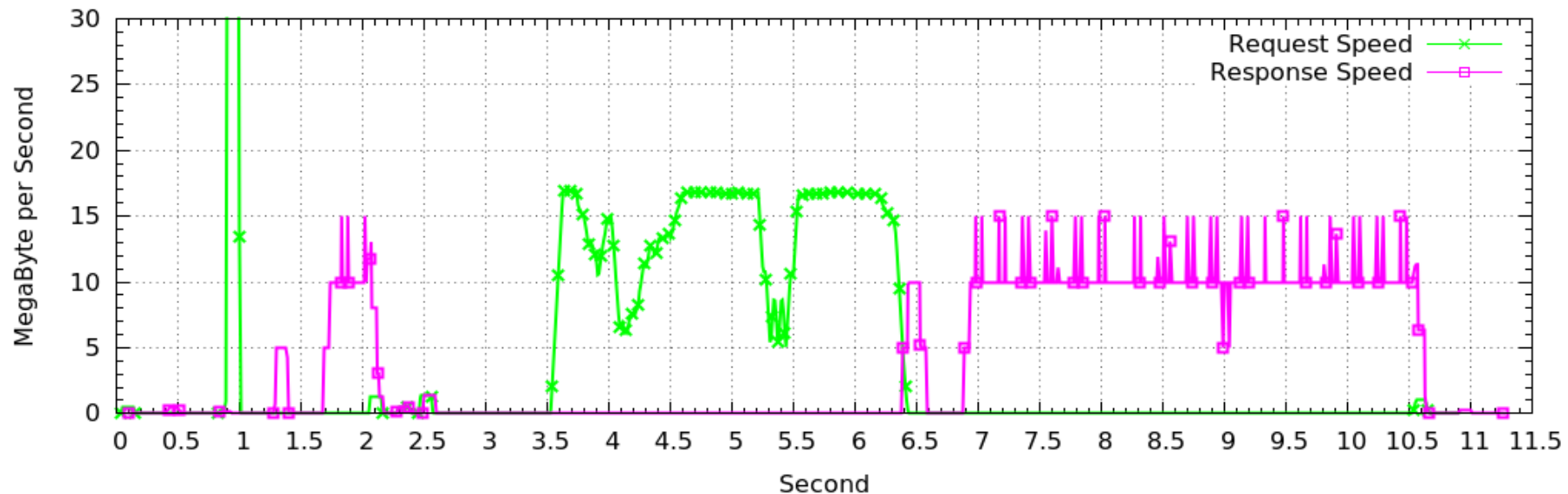
Timing diagram of Hibernation stage

(when request speed > response speed, it means I/O operations are busy)

SD Card Request/Response Chart at Hibernate Stage

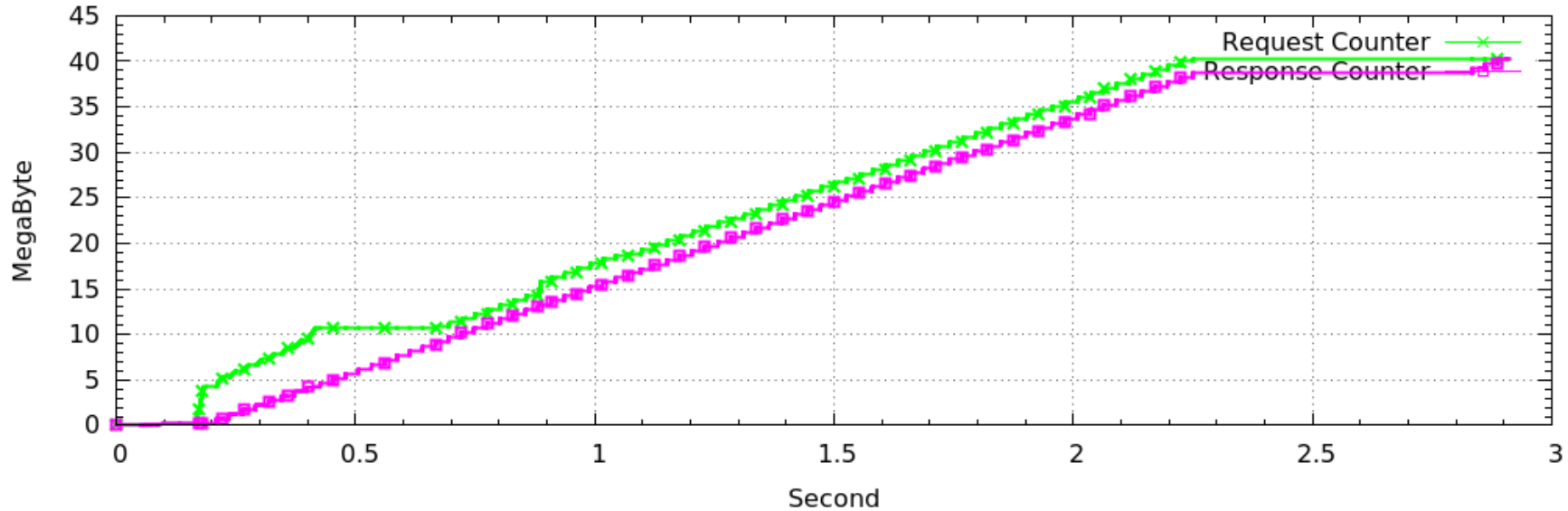


SD Card Request/Response Chart at Hibernate Stage

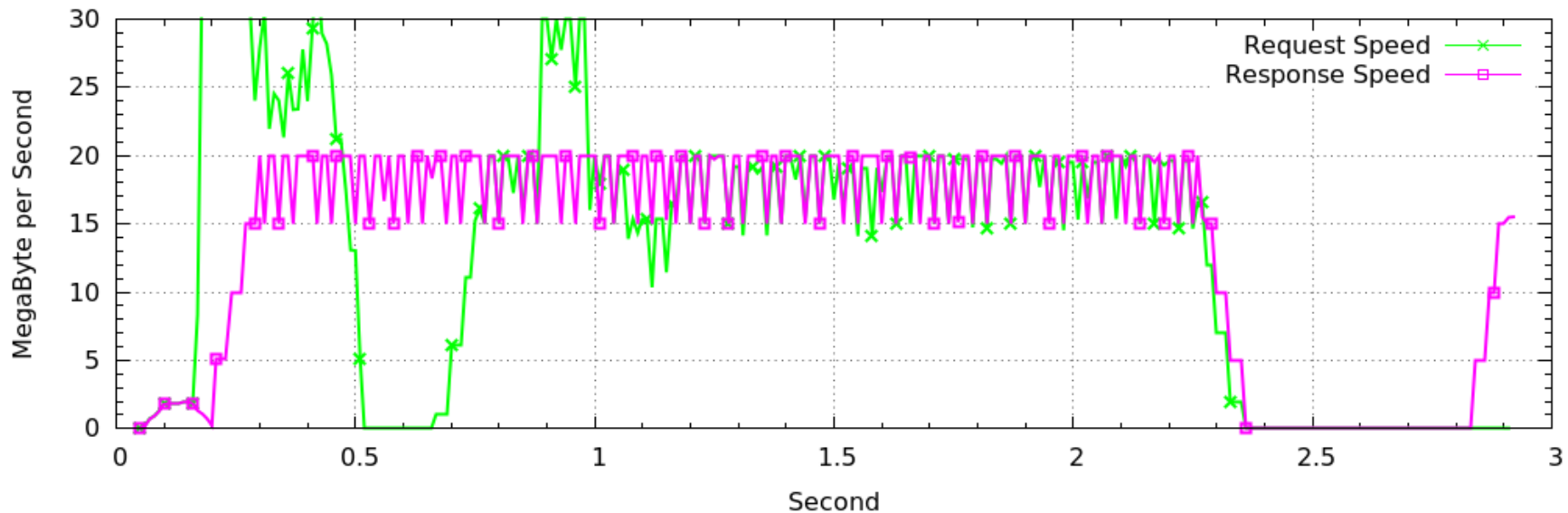


Time diagram of Device Resuming

SD Card Request/Response Chart at Resume Stage



SD Card Request/Response Chart at Resume Stage



Conclusion

- No silver bullet for device boot time optimizations
- Developing an intrusive technique for Android or complex Linux systems is possible for design aspects
- The quality of device driver and file system causes essential impacts as well no matter how a new technique is introduced.
- Mixed model can be used for the production of dedicated systems such as digital TV and automotive IVI.



Reference

- Embedded Linux Wiki: http://elinux.org/Boot_Time
- “*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
- “*Boot Time Optimizations*”, Alexandre Belloni, ELCE 2012
- “*Extending the swsusp Hibernation Framework to ARM*”, Russell Dill, ELC 2013





<http://0xlab.org>