Enforced Deterministic Execution in the Linux Kernel

Chris Cotter

Parallel Programming

- Hardware transitioning from single to multiple cores
- Software is following suit
- Nondeterministic programming model dominates
- Easy to write programs
- Difficult to write correct programs

Nondeterminism

- Race conditions
  - Low-level races (e.g. accessing linked list without proper locking)
  - High-level races (e.g. “time of check to time of use” security exploits)
- Deadlock
- Unpredictable

Overcoming nondeterminism

- Record-and-replay debuggers
- Deterministic environments
  - Special hardware
  - Custom programming languages/compilers
  - Userspace deterministic schedulers

Overcoming nondeterminism

- Record-and-replay debuggers

  High overhead, doesn't actually remove nondeterminism

- Deterministic environments
  - Special hardware
  - Custom programming languages/compilers
  - Userspace deterministic schedulers

Overcoming nondeterminism

- Record-and-replay debuggers

  High overhead, doesn't actually remove nondeterminism

- Deterministic environments
  - Special hardware

    Not commonly available

  - Custom programming languages/compilers

- Userspace deterministic schedulers

Overcoming nondeterminism

- Record-and-replay debuggers

  High overhead, doesn't actually remove nondeterminism

- Deterministic environments
  - Special hardware

    Not commonly available

  - Custom programming languages/compilers

    Restrictive, not familiar

  - Userspace deterministic schedulers

Overcoming nondeterminism

- Record-and-replay debuggers

  High overhead, doesn't actually remove nondeterminism

- Deterministic environments
  - Special hardware

    Not commonly available

  - Custom programming languages/compilers

    Restrictive, not familiar

  - Userspace deterministic schedulers

    Arbitrary (unpredictable) schedules, buggy or malicious programs can interfere with scheduler

Determinator (Aviram et al.)

- "Efficient System-Enforced Deterministic Parallelism" OSDI 2010
- System-enforced OS written from the ground up
- Novel programming model: "naturally and pervasively deterministic"

Determinator (Aviram et al.)

- "Efficient System-Enforced Deterministic Parallelism" OSDI 2010
- System-enforced OS written from the ground up
- Novel programming model: "naturally and pervasively deterministic"
- High-level user library emulates many familiar abstractions
  - Fork/join
  - Pthreads-like API
  - In-memory, deterministic file system

Determinator (Aviram et al.)

- Runs on conventional hardware (x86)
- Programs written in general-purpose languages (C)

Determinator (Aviram et al.)

- Runs on conventional hardware (x86)
- Programs written in general-purpose languages (C)
- Evaluated compute bound applications against Linux
  - Coarse-grained applications run efficiently with little overhead
  - Fine-grained parallel applications have unacceptable overhead

Determinator (Aviram et al.)

- Written from scratch
  - Little application, library support
  - Limited uptake outside academic world

Determinator (Aviram et al.)

- Written from scratch
  - Little application, library support
  - Limited uptake outside academic world
- Linux: mature, widely deployed OS (desktops, mobile, embedded)

Determinator (Aviram et al.)

- Written from scratch
  - Little application, library support
  - Limited uptake outside academic world
- Linux: mature, widely deployed OS (desktops, mobile, embedded)

Can we make Linux deterministic using Determinator's programming model?

Determinator (Aviram et al.)

- Written from scratch
  - Little application, library support
  - Limited uptake outside academic world
- Linux: mature, widely deployed OS (desktops, mobile, embedded)

Can we make Linux deterministic using Determinator's programming model?

Let's give it a shot...

Outline

- Introduce parallelism and Determinator
- **Recapitulate Determinator paper**
- Design goals/non-goals for deterministic Linux
- Implementation
- Evaluation
- Conclusion

Sources of nondeterminism

- Primary cause: timing dependent data races
- Four main categories

Sources of nondeterminism

- Primary cause: timing dependent data races
- Four main categories
  - Explicit nondeterminism

Sources of nondeterminism

- Primary cause: timing dependent data races
- Four main categories
  - Explicit nondeterminism
  - Shared program state

Sources of nondeterminism

- Primary cause: timing dependent data races
- Four main categories
  - Explicit nondeterminism
  - Shared program state
  - Nondeterministic scheduling abstractions

Sources of nondeterminism

- Primary cause: timing dependent data races
- Four main categories
  - Explicit nondeterminism
  - Shared program state
  - Nondeterministic scheduling abstractions
  - Globally shared namespaces

Explicit nondeterminism

- Network packets, time of day, random numbers
- Semantically relevant

Explicit nondeterminism

- Network packets, time of day, random numbers
- Semantically relevant
- Determinator turns these inputs into explicit I/O
  - Controllable
  - Can be recorded for replay

Shared state

- Threads share memory, processes share file system
- Determinator adopts *private workspace model*

Shared state

- Threads share memory, processes share file system
- Determinator adopts *private workspace model*
- Eliminate shared memory, no file system access

Shared state

- Threads share memory, processes share file system
- Determinator adopts *private workspace model*
- Eliminate shared memory, no file system access
- Writes to memory not globally visible until explicit synchronization

Shared state

- Threads share memory, processes share file system
- Determinator adopts *private workspace model*
- Eliminate shared memory, no file system access
- Writes to memory not globally visible until explicit synchronization
- Eliminates read-write races

Shared state

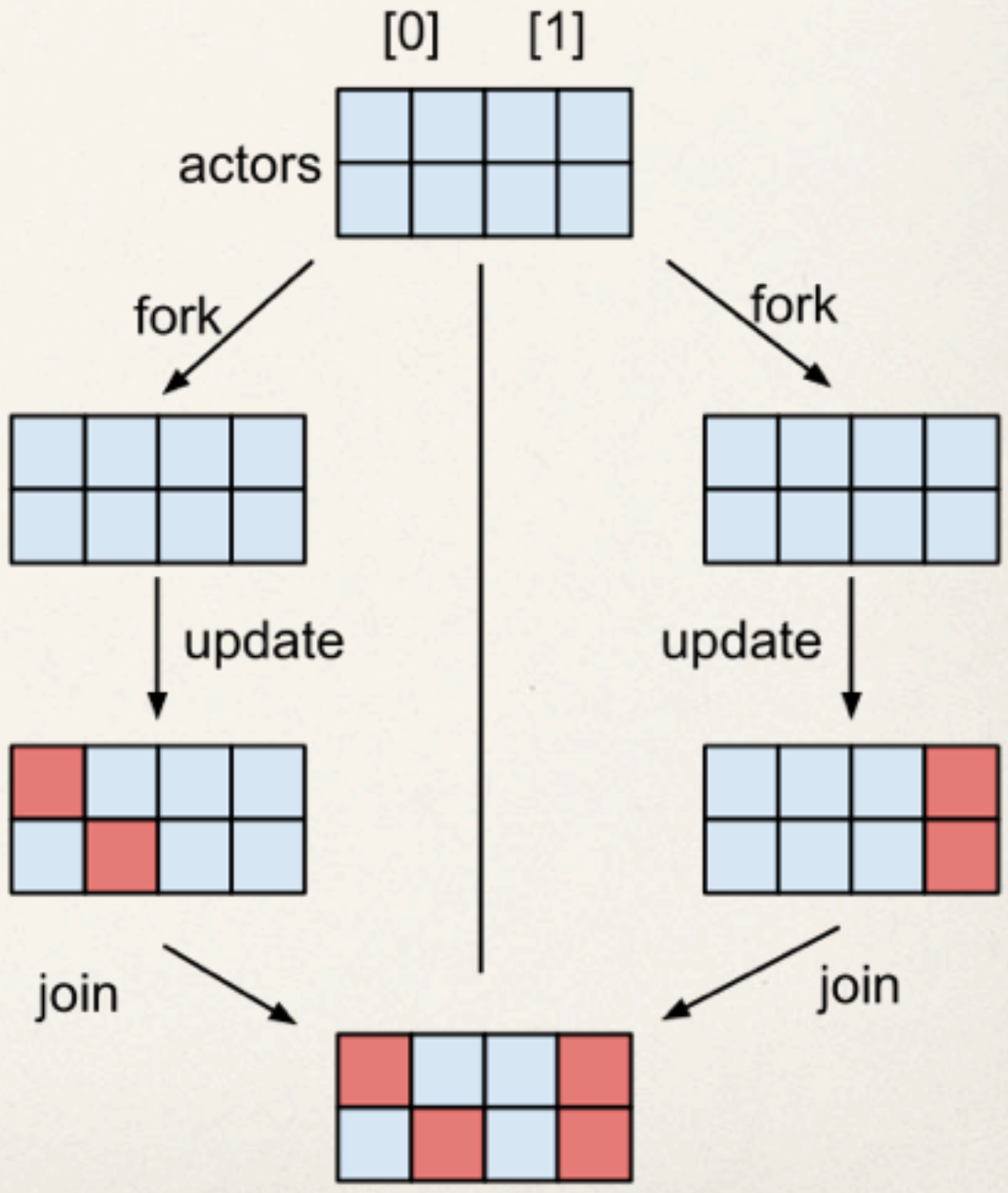
- Threads share memory, processes share file system
- Determinator adopts *private workspace model*
  - Eliminate shared memory, no file system access
  - Writes to memory not globally visible until explicit synchronization
  - Eliminates read-write races
  - Write-write races detected by OS, generate exception

Shared state

```
struct actor_state actor[nactors];

update_actor(i)
  // Examine other actor state and update
  //  actor[i] accordingly.

main()
  // Initialize actor[] state
  for(time = 0; ; time++)
    for (i = 0; i < nactors; i++)
      if (thread_fork(i) == IN_CHILD)
        update_actor(i);
        thread_exit();
    for (i = 0; i < nactors; i++)
      thread_join(i);
```



Source: [1]

Shared state

```
struct actor_state actor[nactors];

update_actor(i)
  // Examine other actor state and update
  //  actor[i] accordingly.

main()
  // Initialize actor[] state
  for(time = 0; ; time++)
    for (i = 0; i < nactors; i++)
      if (thread_fork(i) == IN_CHILD)
        update_actor(i);
        thread_exit();
    for (i = 0; i < nactors; i++)
      thread_join(i);
```

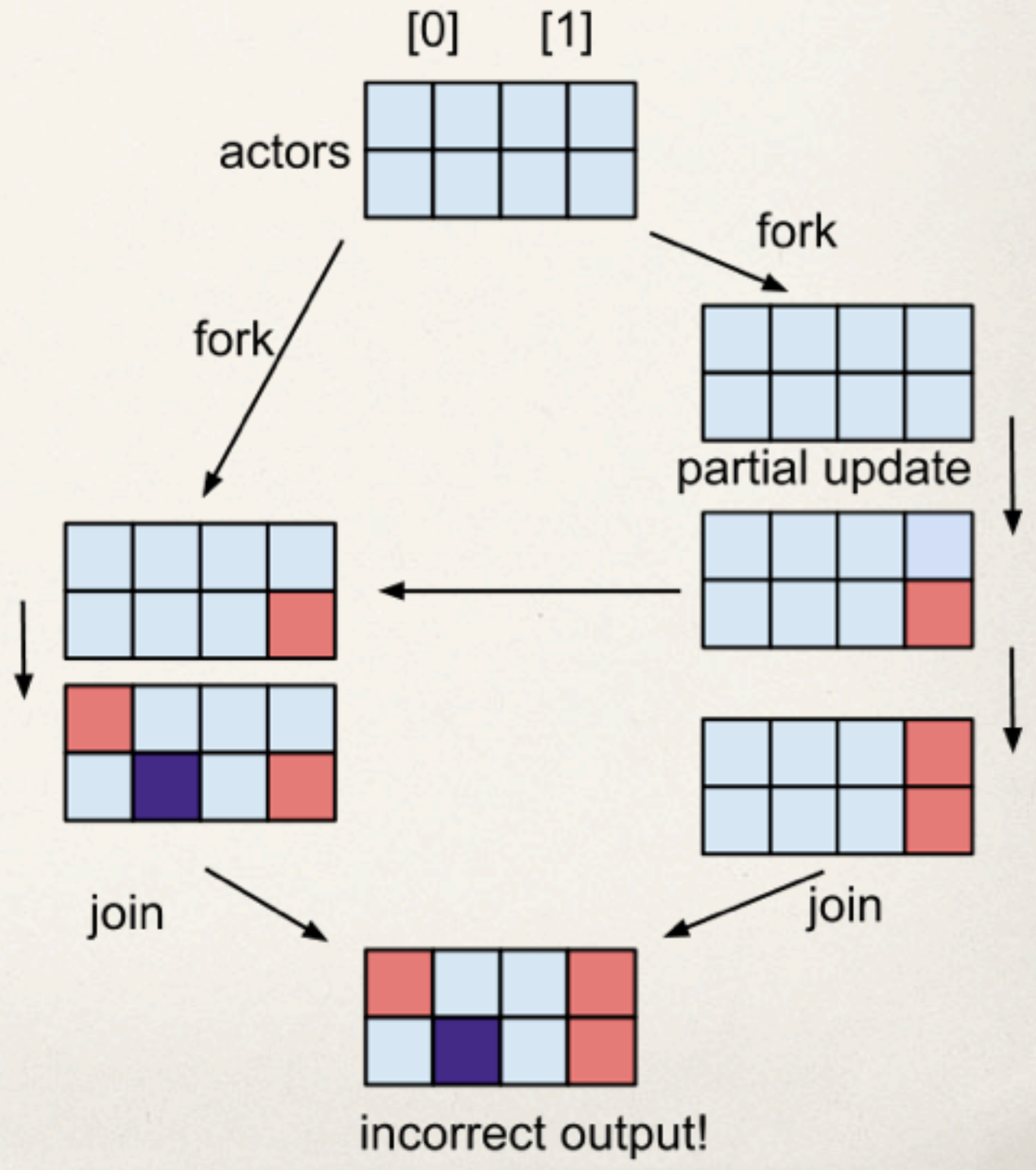
Source: [1]

Shared state

```
struct actor_state actor[nactors];

update_actor(i)
  // Examine other actor state and update
  //  actor[i] accordingly.

main()
  // Initialize actor[] state
  for(time = 0; ; time++)
    for (i = 0; i < nactors; i++)
      if (thread_fork(i) == IN_CHILD)
        update_actor(i);
        thread_exit();
    for (i = 0; i < nactors; i++)
      thread_join(i);
```



Source: [1]

Scheduling abstractions

- Mutex acquisition is racy
- Can record lock acquisition sequence
- Still arbitrary and unpredictable

Scheduling abstractions

- Mutex acquisition is racy
- Can record lock acquisition sequence
- Still arbitrary and unpredictable
- Determinator only allows naturally deterministic synchronization abstractions

Scheduling abstractions

- Mutex acquisition is racy
- Can record lock acquisition sequence
- Still arbitrary and unpredictable
- Determinator only allows naturally deterministic synchronization abstractions
- Fork/join

Global namespaces

- OS APIs expose globally visible namespaces
  - Process IDs
  - Temporary filenames: `mktemp()`
  - Memory management: `malloc()`

Global namespaces

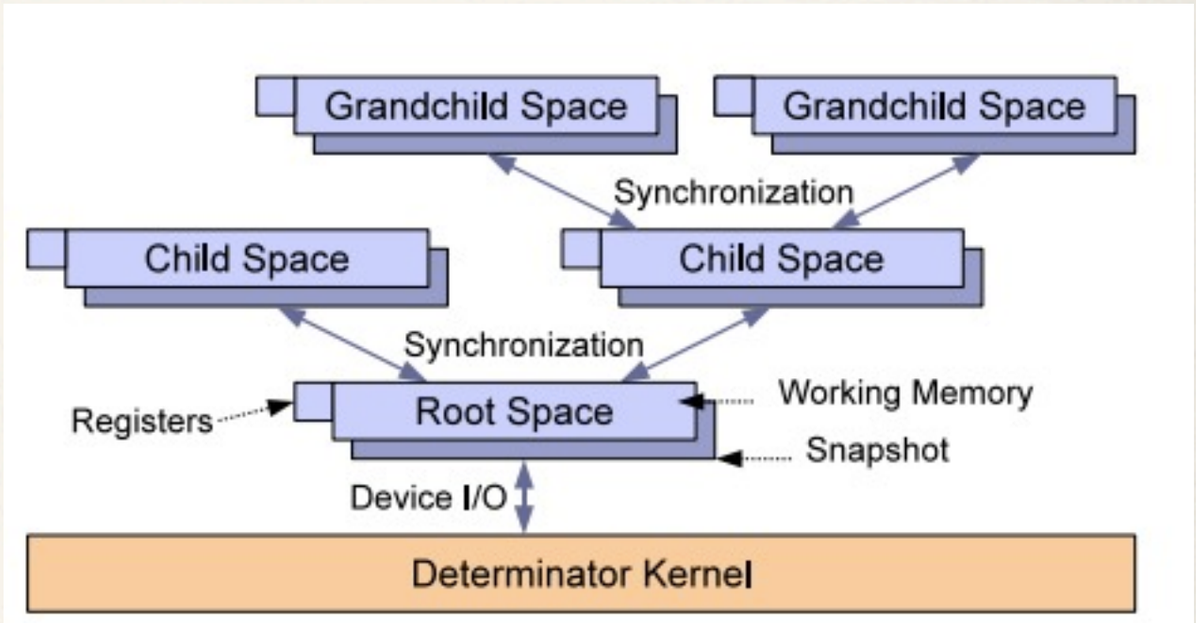
- OS APIs expose globally visible namespaces
  - Process IDs
  - Temporary filenames: `mktemp()`
  - Memory management: `malloc()`
- Determinator doesn't choose identifiers, user programs do

Global namespaces

- OS APIs expose globally visible namespaces
  - Process IDs
  - Temporary filenames: `mktemp()`
  - Memory management: `malloc()`
- Determinator doesn't choose identifiers, user programs do
- E.g. fork() takes explicit process ID as input

Determinator Kernel

- Nested process model (hierarchical)
- Interact only with parent and children
- Children cannot outlive parent
- No shared memory, file system
- No direct I/O access
- Only the *root* process may access I/O



Source: [2]

Kernel Interface

- Put() - parent interacts with child
  - Copy data into child
  - “Snapshot” memory image into child
  - Start child execution

Kernel Interface

- Put() - parent interacts with child
  - Copy data into child
  - “Snapshot” memory image into child
  - Start child execution
- Get() - parent interacts with child
  - Copy data from child
  - “Merge” memory changes since last snapshot

Kernel Interface

- Put() - parent interacts with child
  - Copy data into child
  - “Snapshot” memory image into child
  - Start child execution
- Get() - parent interacts with child
  - Copy data from child
  - “Merge” memory changes since last snapshot
- Ret() - Interacts with parent
  - Blocks execution until parent issues a Put

Kernel Interface

- Put and Get take an explicit child process ID argument
- Put and Get block until a child issues a Ret
- Processor exceptions (e.g. divide-by-zero) generate implicit Ret
- These requirements ensure synchronization at well defined points
- Put/Get/Ret "make the [process] hierarchy a deterministic Kahn network"

High-level Library

- Similar to standard C library
- Emulate Unix-style abstractions
- Process/thread management
- In-memory file system with Unix-like interface
- I/O facilities (stdin/stdout via in-memory file system)

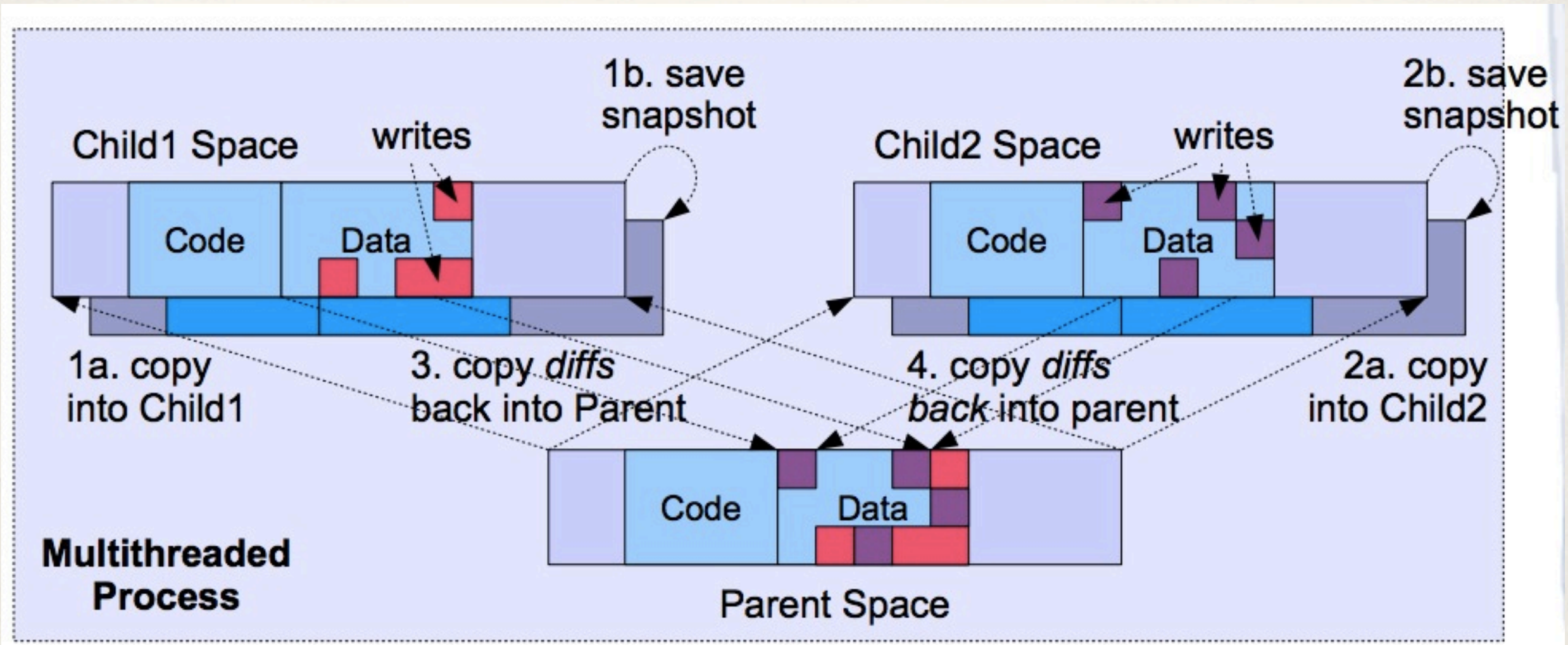
Example: fork/join/exec

- To fork a thread, use Put to snapshot the parent's virtual memory image into a new child and starts the child
- To join on a thread, issue a Get to merge the child's virtual memory changes into the parent
- To exec, the user library forks a new child, reads the new executable from the in-memory file system into the child, then starts the child

Example: Snapshot/Merge

- Snapshot duplicates page tables via copy-on-write into the child
- Creates second duplicate as a reference for Merge comparison
- Merge does a three way diff
- Examine page table entries: check dirty bits
- Only do byte-by-byte comparison when parent and child page table entries are dirtied

Example: Snapshot/Merge



Source: [1]

Outline

- Introduce parallelism and Determinator
- Recapitulate Determinator paper
- **Design goals/non-goals for deterministic Linux**
- Implementation
- Evaluation
- Conclusion

Design Goals

- We wish to make x86_64 Linux *deterministic*
- Adapt Determinator's kernel to Linux
- Legacy nondeterministic applications shall run alongside deterministic applications
- Legacy applications must be rewritten
- Write high-level user library as Determinator does
- Improve in-memory file system capabilities

Design Non-goals

- May not apply all of Determinator's features
  - Distributed computing
  - Put/Get support instruction limits, copying child subtree
- Linux features may be restricted
  - Huge pages (4MB vs 4KB)
  - Shared library support

Challenges

Challenges

- All challenges identified by Determinator apply
- Additionally, Linux supports signals, pipes, System V IPC
- Process model too flexible (reparenting)
- How to allow nondeterministic applications?
- Can't have only *root* be nondeterministic
- Can't use standard C library: uses nondeterministic syscalls

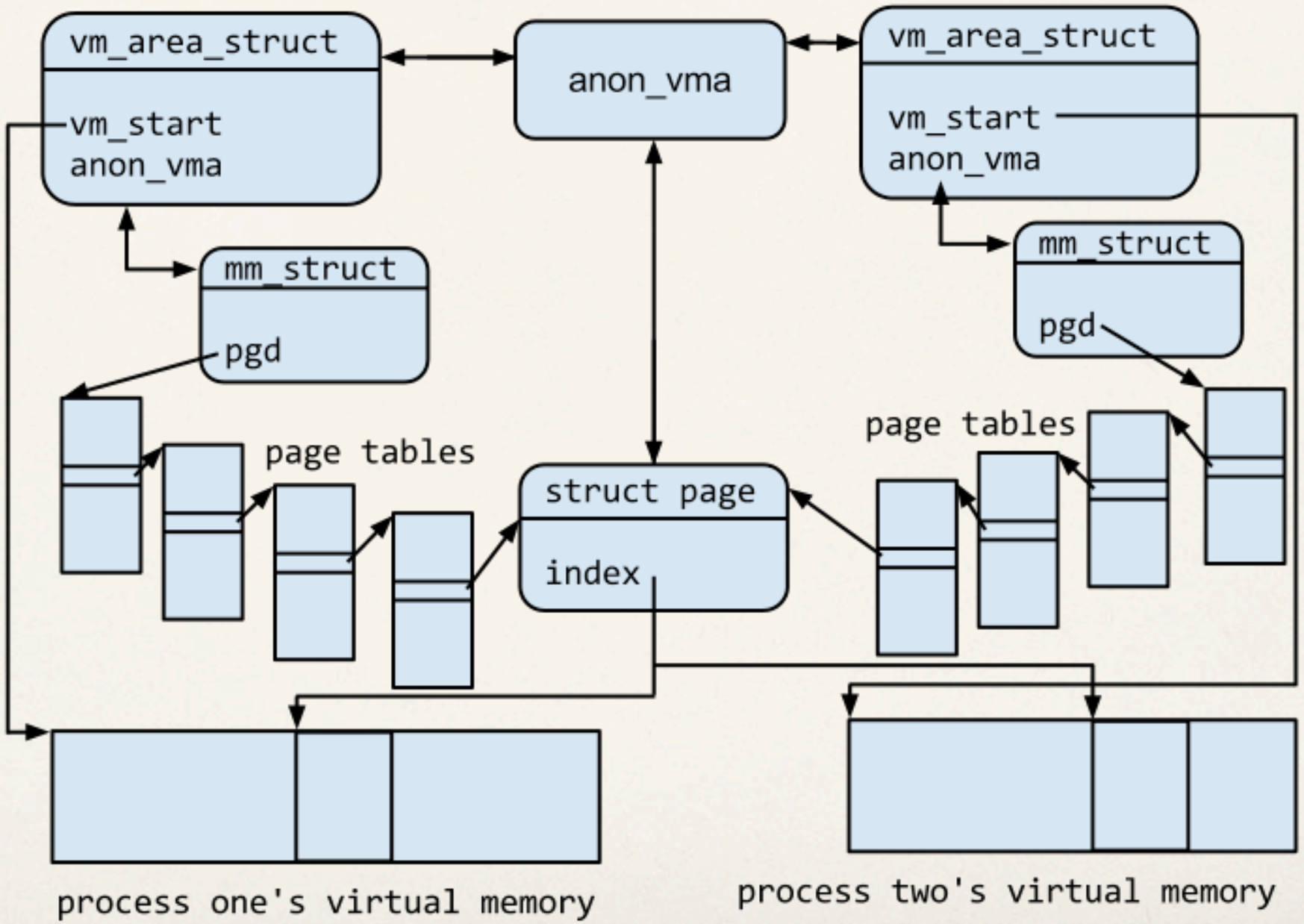
Challenges

- Memory subsystem complexity
- Determinator only deals with page tables
- Linux has many abstraction layers
- Supports memory mapped files, huge pages, swapping to disk

Challenges

- Memory subsystem complexity
- Determinator only deals with page tables
- Linux has many abstraction layers
- Supports memory mapped files, huge pages, swapping to disk
- These are *implementation* challenges for a first time Linux kernel developer

Challenges: memory subsystem



Outline

- Introduce parallelism and Determinator
- Recapitulate Determinator paper
- Design goals/non-goals for deterministic Linux
- **Implementation**
- Evaluation
- Conclusion