Concurrency and security

Dr Robert N.M. Watson
Computer Laboratory
University of Cambridge

Part II Security
22 February 2012

UNIVERSITY OF CAMBRIDGE

Outline

- What is concurrency?
- How does it relate to security?
- System call wrappers case study
- Lessons learned

concurrent (adj):

Running together in space, as parallel lines; going on side by side, as proceedings; occurring together, as events or circumstances; existing or arising together; conjoint, associated.

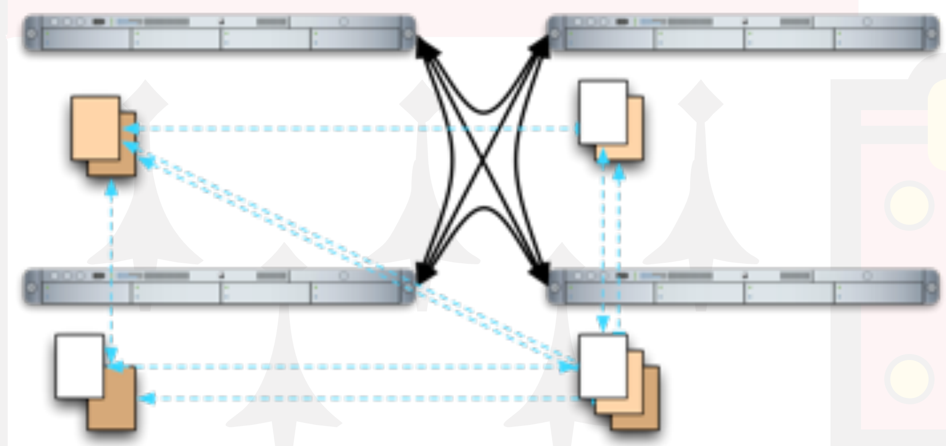
Oxford English Dictionary, Second Edition

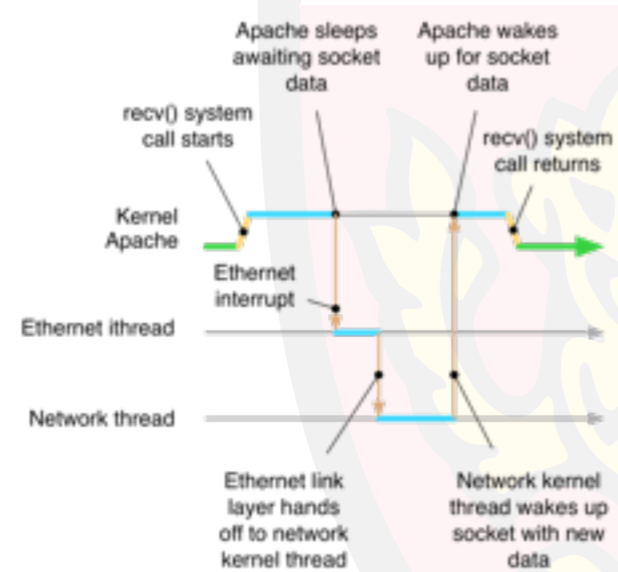
Concurrency

- Multiple computational processes **execute at the same time** and may **interact with each other**
- Concurrency leads to the **appearance of non-determinism**

Finding concurrency

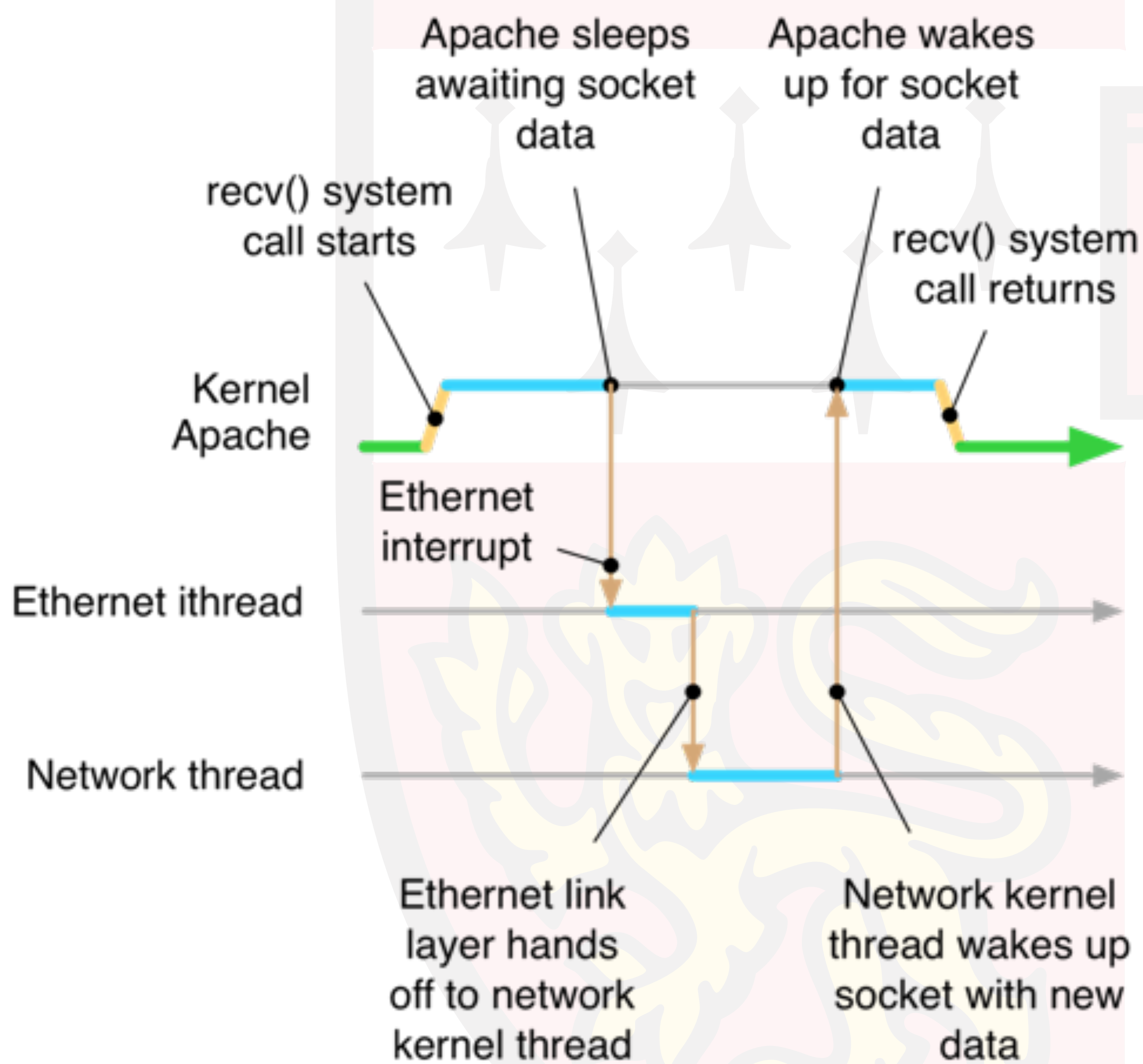
- Interleaved or asynchronous computation
- Parallel computing
- Distributed systems





Local concurrency

- Interleaved or asynchronous execution on a single processor
- More efficient use of computation resources
- Mask I/O latency, multitasking, preemption



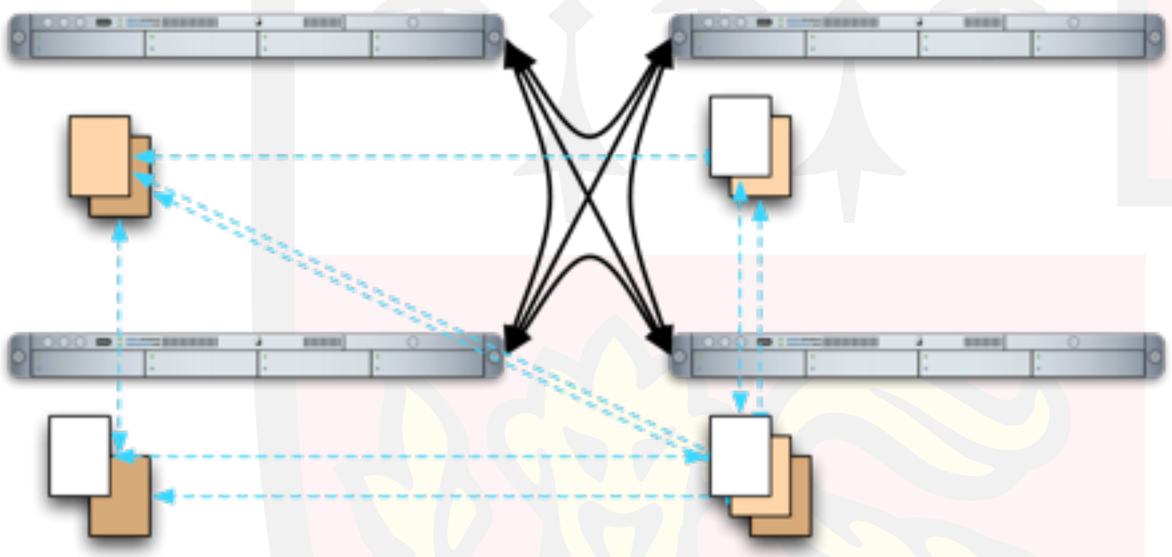
UNIVERSITY OF CAMBRIDGE

Shared memory multiprocessing

- Multiple CPUs with shared memory
- Possibly asymmetric memory speed/topology
- Weaker memory model: writes order weakened, explicit synchronisation
- New programming models

Message passing and distributed systems

- Protocol-centric approach with explicit communication
- Synchronous or asynchronous
- Explicit data consistency management
- Distributed file systems, databases, etc.



UNIVERSITY OF CAMBRIDGE

Concurrency research

- Produce more concurrency and parallelism
- Maximise performance
- Represent concurrency to the programmer
- Identify necessary and sufficient orderings
- Detect and eliminate incorrectness
- Manage additional visible failure modes

Practical concerns

- Performance
- **Consistency of replicated data**
- Liveliness of concurrency protocols
- Distributed system failure modes

Consistency models

- Semantics when accessing replicated data concurrently from multiple processes
  - Strong models support traditional assumptions of non-concurrent access
  - Weak models exchange consistency for performance improvement
- Critical bugs arise if mishandled

ACID properties

- Database transaction properties
  - **A**tomicity - all or nothing
  - **C**onsistency - no inconsistent final states
  - **I**solation - no inconsistent intermediate states
  - **D**urability - results are durable

Serialisability

- Results of concurrent transactions must be equivalent to outcome of a possible serial execution of the transactions
  - Serialisable outcomes of {A, B, C}:
    - A B C   A C B   B A C
      B C A   C A B   C B A
- Strong model that is easy to reason about

Weaker consistency

- Strong models expose latency/contention
- Desirable to allow access to stale data
  - Timeouts: DNS caches, NFS attribute cache, x509 certificates, Kerberos tickets
  - Weaker semantics: AFS last close, UNIX passwd/group vs. in-kernel credentials
- Must reason carefully about results

Concurrency and security

- Abbot, Bisbey/Hollingworth in 1970's
- **Inadequate synchronisation** or **unexpected concurrency** leads to violation of security policy
- **Race conditions**
- Distributed systems, multicore notebooks, ... this is an urgent issue

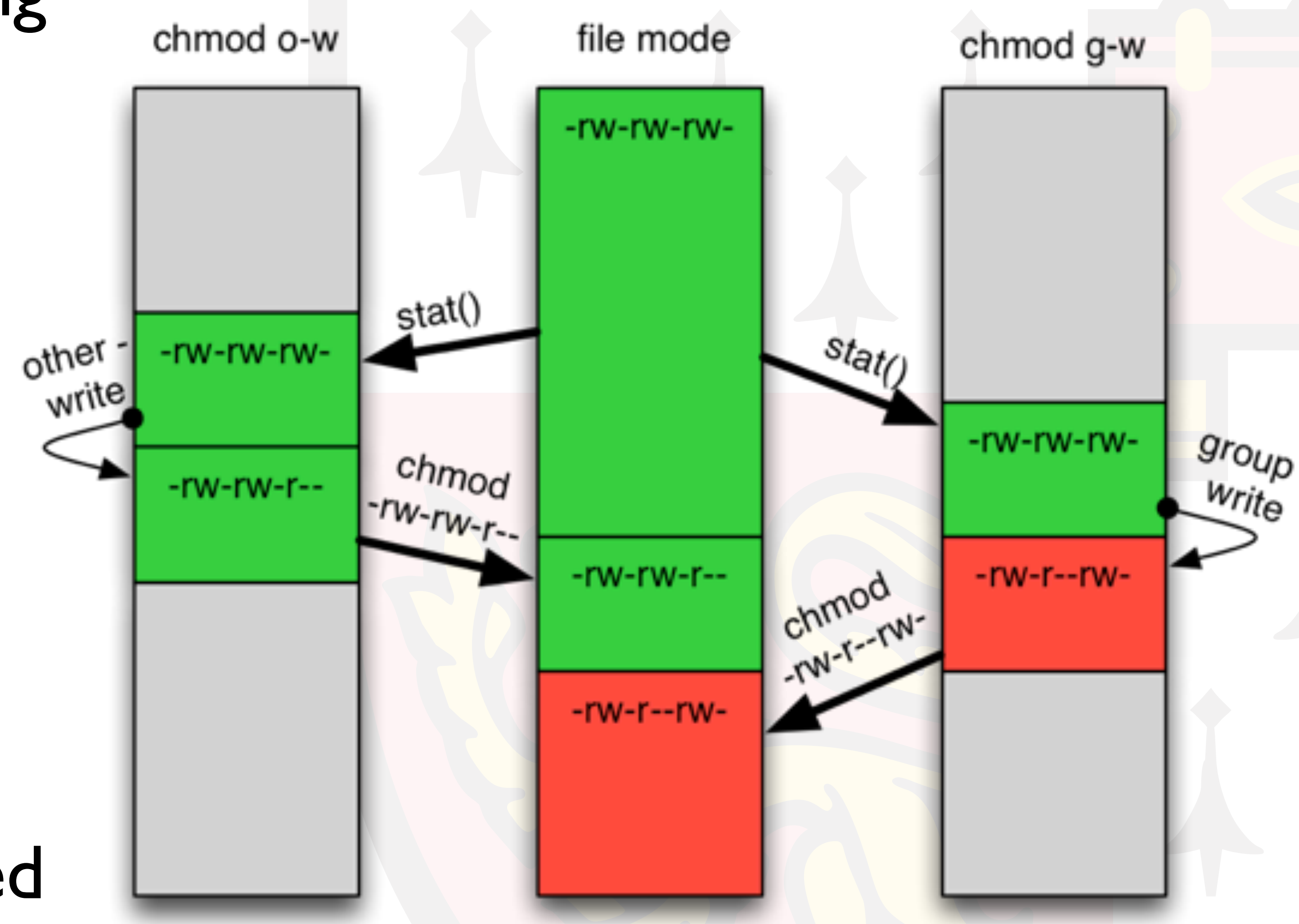
UNIVERSITY OF CAMBRIDGE

Concurrency vulnerabilities

- When **incorrect concurrency management** leads to **vulnerability**
  - Violation of **specifications**
  - Violation of **user expectations**
  - **Passive** - leak information or privilege
  - **Active** - allow adversary to extract information, gain privilege, deny service...

Example passive vulnerability

- Simultaneously executing UNIX **chmod** with update syntax
  - chmod g-w file
- stat() and chmod() syscalls can't express update atomically
- Both commands succeed but only one takes effect



UNIVERSITY OF CAMBRIDGE

The challenge

- Reasoning about security and concurrency almost identical
- "Weakest link" analysis
- Can't exercise bugs deterministically in testing due to state explosion
- Debuggers mask rather than reveal bugs
- Static and dynamic analysis tools limited

From concurrency bug to security bug

- Vulnerabilities in security-critical interfaces
  - Races on arguments and interpretation
  - Atomic “check” and “access” not possible
- Data consistency vulnerabilities
  - Stale or inconsistent security metadata
  - Security metadata and data inconsistent

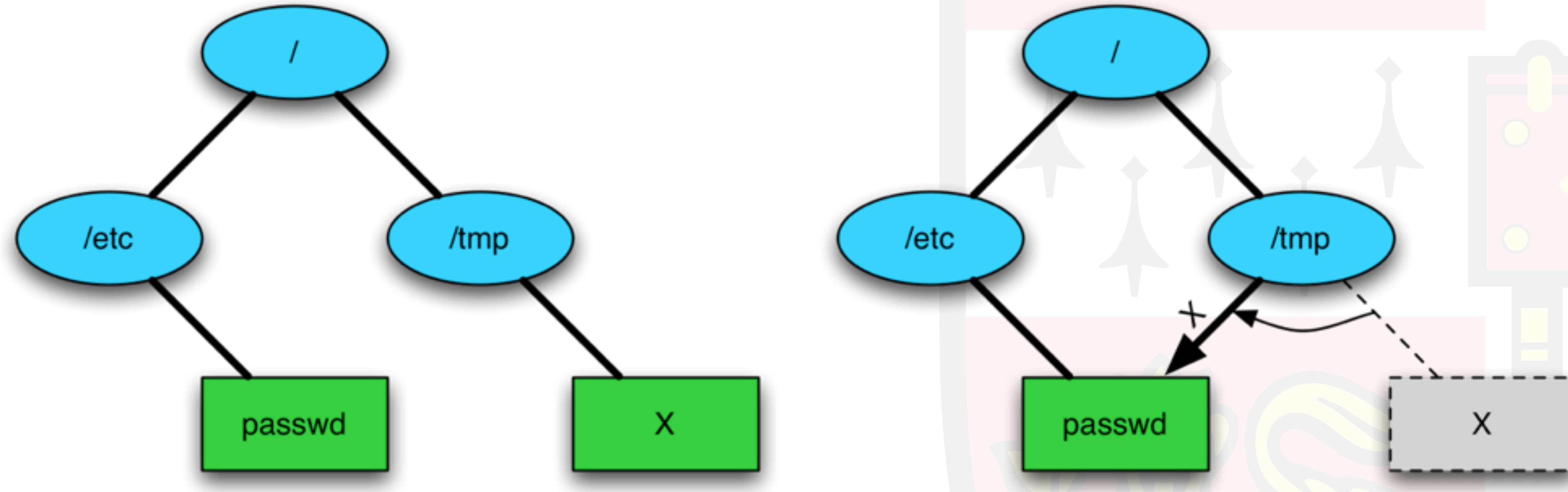
Learning by example

- Consider three vulnerability types briefly
  - /tmp race conditions
  - SMT covert channels
- Detailed study
  - System call wrapper races

/tmp race conditions

- Bishop and Dilger, 1996
- UNIX file system APIs allow non-atomic sequences resulting in vulnerability
- Unprivileged processes manipulate /tmp and other shared locations
- Then race against privilege processes to replace targets of open(), etc.

/tmp races (cont)



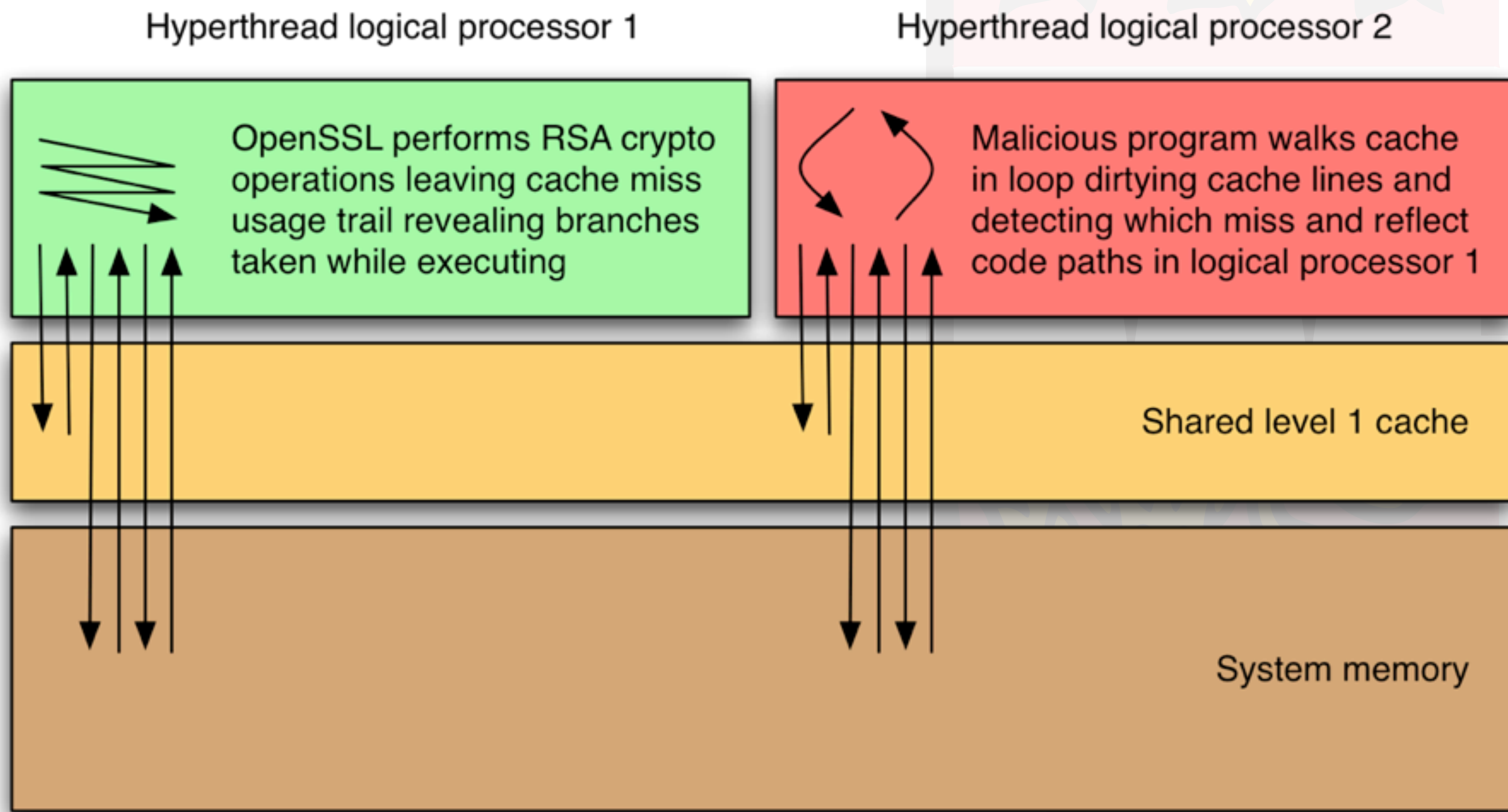
access() system call
traverses /tmp/X to file

open() system call
traverses /tmp/X symlink
to /etc/passwd

SMT side channels

- Percival 2005, Bernstein 2005, Osvik 2005
- Covert/side channel channels historically considered an academic research topic
- Symmetric multithreading, Hyper-threading, and multicore processors share caches
- Possible to extract RSA, AES key material by analysing cache misses on shared cache

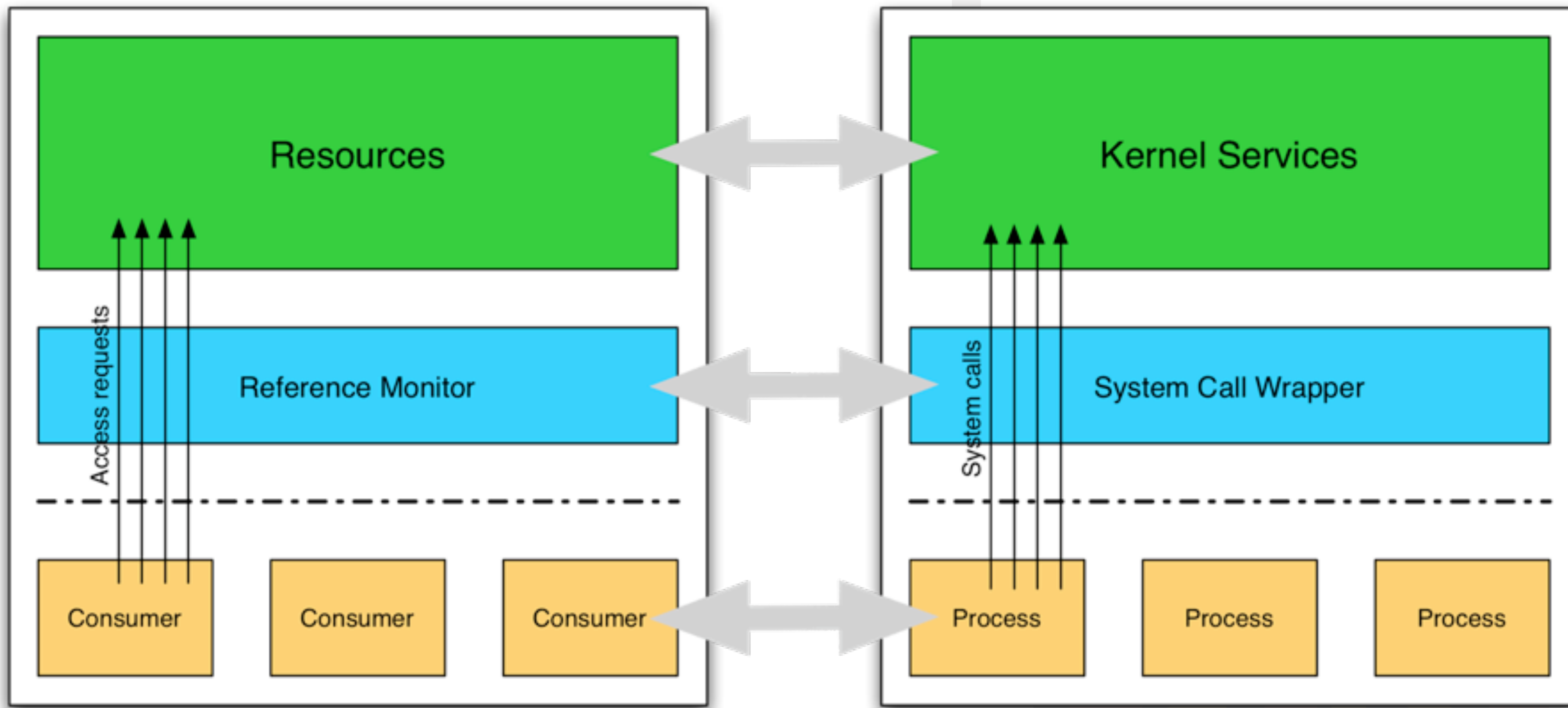
SMT covert channels



System call wrapper vulnerabilities

- Our main case study: system call wrappers
- Widely-used security extension technique
- No OS kernel source code required
- Pre- and post-conditions on system calls
- Application sand-boxing and monitoring
- Frameworks: GSWTK, Systrace, CerbNG

System call wrappers as a reference monitor



Are wrappers a reference monitor?

- Reference monitors (Anderson 1972)
  - Tamper-proof: in kernel address space
  - Non-bypassable: can inspect all syscalls
  - Small enough to test and analyse: security code neatly encapsulated in one place
- Perhaps they count?

Or not

- No time axis in neat picture
  - System calls are not atomic
  - Wrappers definitely not atomic with system calls
- Opportunity for race conditions on copying and interpretation of arguments and results

Race conditions to consider

- **Syntactic races** - indirect arguments are copied on demand, so wrappers do their own copy and may see different values
- **Semantic races** - even if argument values are the same, interpretations may change between the wrapper and kernel

Types of system call wrapper races

- TOCTTOU - time-of-check-to-time-of-use
- TOATTOU - time-of-audit-to-time-of-use
- TORTTOU - time-of-replacement-to-time-of-use

Goals of the attacker

- Bypass wrapper to perform controlled audited, or modified system calls

```
open("/sensitive/file", O_RDWR)
write(fd, virusptr, viruslen)
connect(s, controlledaddr, addrlen)
```

- Can attack indirect arguments: paths, I/O data, socket addresses, group lists, ...

Racing in user memory

- User process, using concurrency, will replace argument memory in address space between wrapper and kernel processing
- Uniprocessor - force page fault or blocking so kernel yields to attacking process/thread
- Multiprocessor - execute on second CPU or use uniprocessor techniques

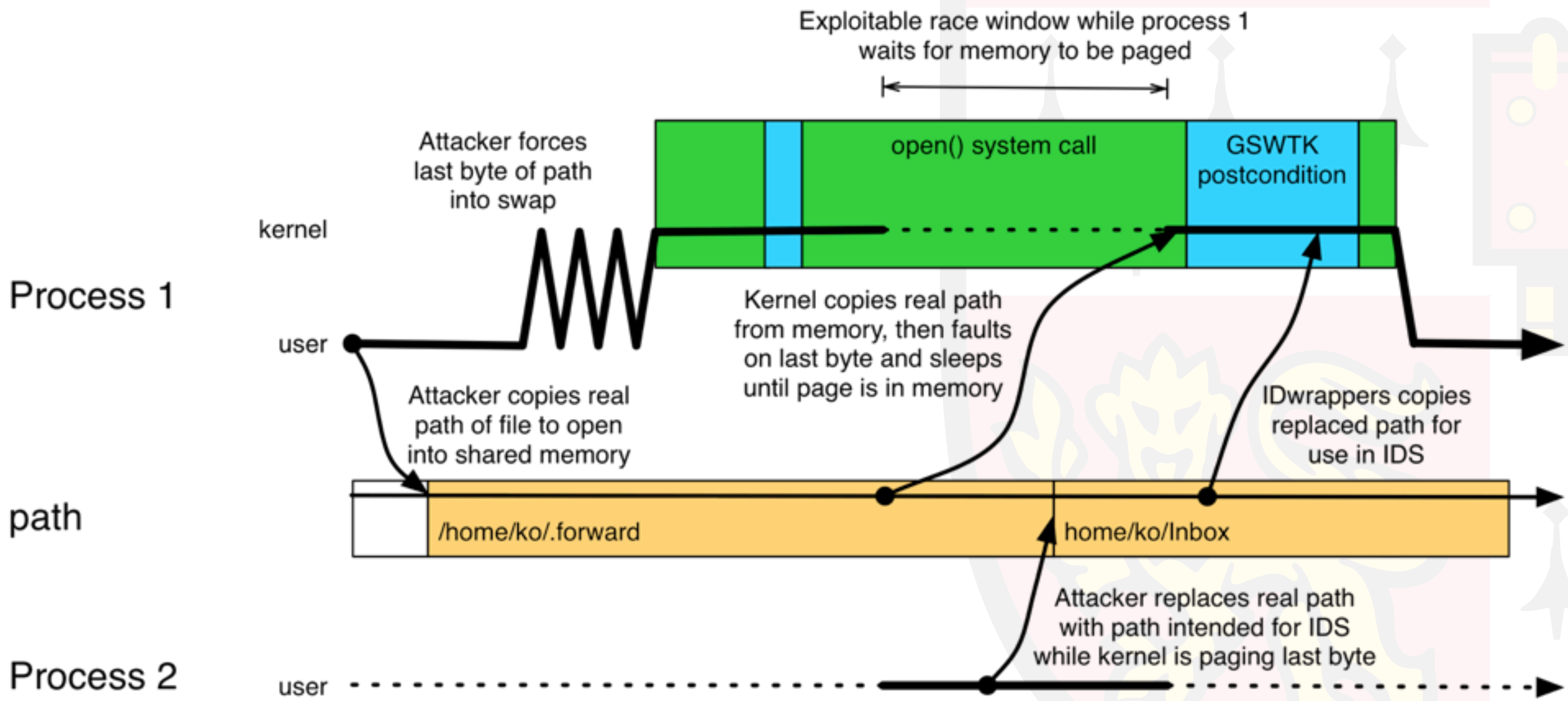
Practical attacks

- Consider attacks on three wrapper frameworks implementing many policies
  - Systrace [sudo, sysjail, native policies]
  - GWSTK [demo policies and IDwrappers]
  - CerbNG [demo policies]
- Attacks are policy-specific rather than framework-specific

Uniprocessor example

- Generic Software Wrappers Toolkit (GSWTK) with IDwrappers
    - Ko, Fraser, Badger, Kilpatrick 2000
    - Flexible enforcement + IDS framework
    - 16 of 23 demo wrappers vulnerable
- Employ page faults on indirect arguments

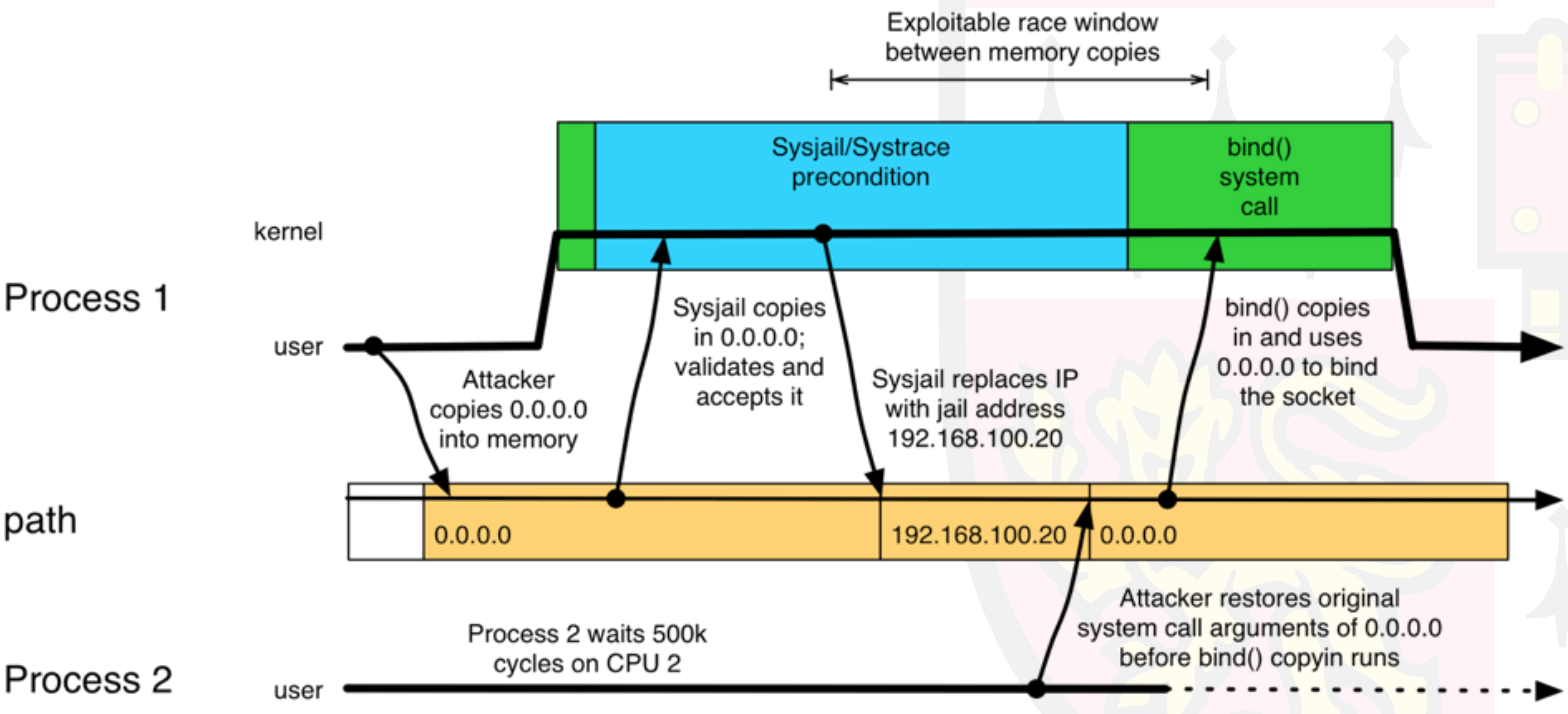
UP GSWTK exploit



Multiprocessor example

- Sysjail over Systrace
  - Provos, 2003; Dzonsons 2006
  - Systrace allows processes to instrument system calls of other processes
  - Sysjail implements FreeBSD's "jail" model on NetBSD and OpenBSD with Systrace
- Employ true parallelism to escape Sysjail

SMP Systrace exploit



UNIVERSITY OF CAMBRIDGE

Implementation notes

- OS paging systems vary significantly
- On SMP, race window sizes vary
  - TSC a good way to time attacks
  - Systrace experiences 500k cycyle+ windows due to many context switches; others much faster
- Both techniques are extremely reliable

Defence against wrapper races

- Serious vulnerabilities
  - Bypass of audit, control, replacement
- Easily bypassed mitigation techniques
- Interposition requires reliable access to syscall arguments, foiled by concurrency
- More synchronisation, message passing, or just not using system call wrappers...

Lessons learned

- Concurrency bugs are a significant security threat to complex software systems
- Developing and testing concurrent programs is extremely difficult
- Static analysis and debugging tools are of limited utility, languages are still immature
- SMP and distributed systems proliferating

Concurrency principles for secure software

1. Concurrency is hard — avoid it
2. Strong consistency models are easier to understand and implement than weak
3. Prefer multiple readers to multiple writers
4. Prefer deterministic invalidation to time expiry of cached data

Principles II

5. Don’t rely on atomicity that can’t be supported by the underlying platform
6. Message passing, while slower, enforces a protocol-centric analysis and can make reasoning and debugging easier
7. Document locking or message protocols with assertions that see continuous testing

Principles III

8. Defending against side channels is difficult (impossible), but critical for crypto
9. Remember that every narrow race window can be widened in a way you don't expect
10. Always test on slow hardware