Multi-tasking in PHP

About Me

- Formerly CTO for Company52
- Currently work at Brandmovers on Northside Drive
- Self-taught
- Full-time geek since 2007

Multitasking
ROFLCAT.COM

Use Cases

- System resources are not the bottleneck
- Batch processing involving an API:
  - E-Mail
  - Geocoding
  - Electronic billing
- Daemons

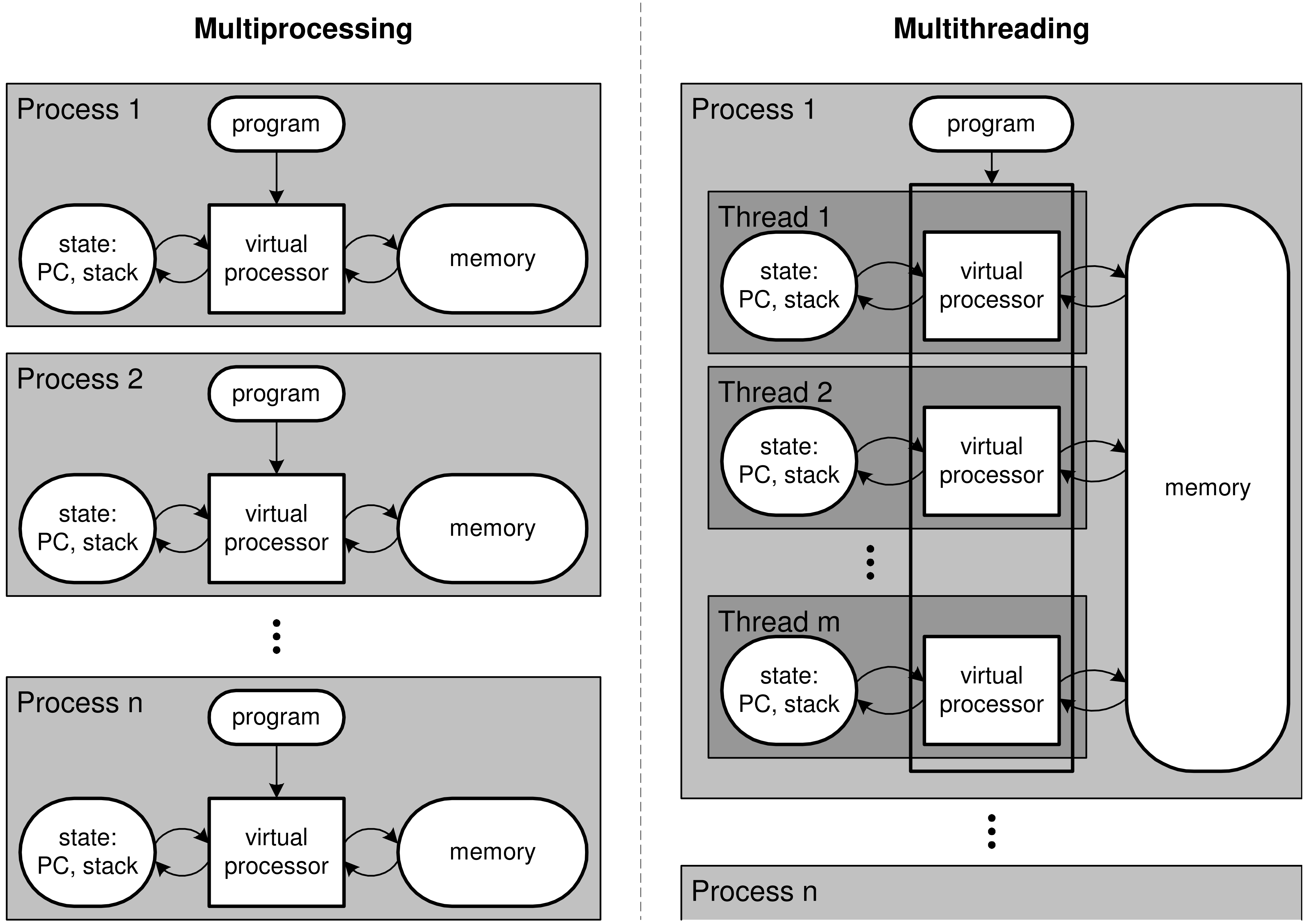
Alternatives

- Gearman
- curl_multi_*
- Other scripting languages

Theory

Two ways to multi-task

Multi-processing	Multi-threading
Separate memory	Shared memory
Errors are isolated	Errors are not isolated
Separate permissions	Same permissions
Linux/UNIX	Windows



Courtesy www.fmc-modeling.org

Multiprocessing

- “The simultaneous execution of two or more programs by separate CPUs under integrated control.”
- Clones the entire process, except resources
- Copy-on-write memory

Forking

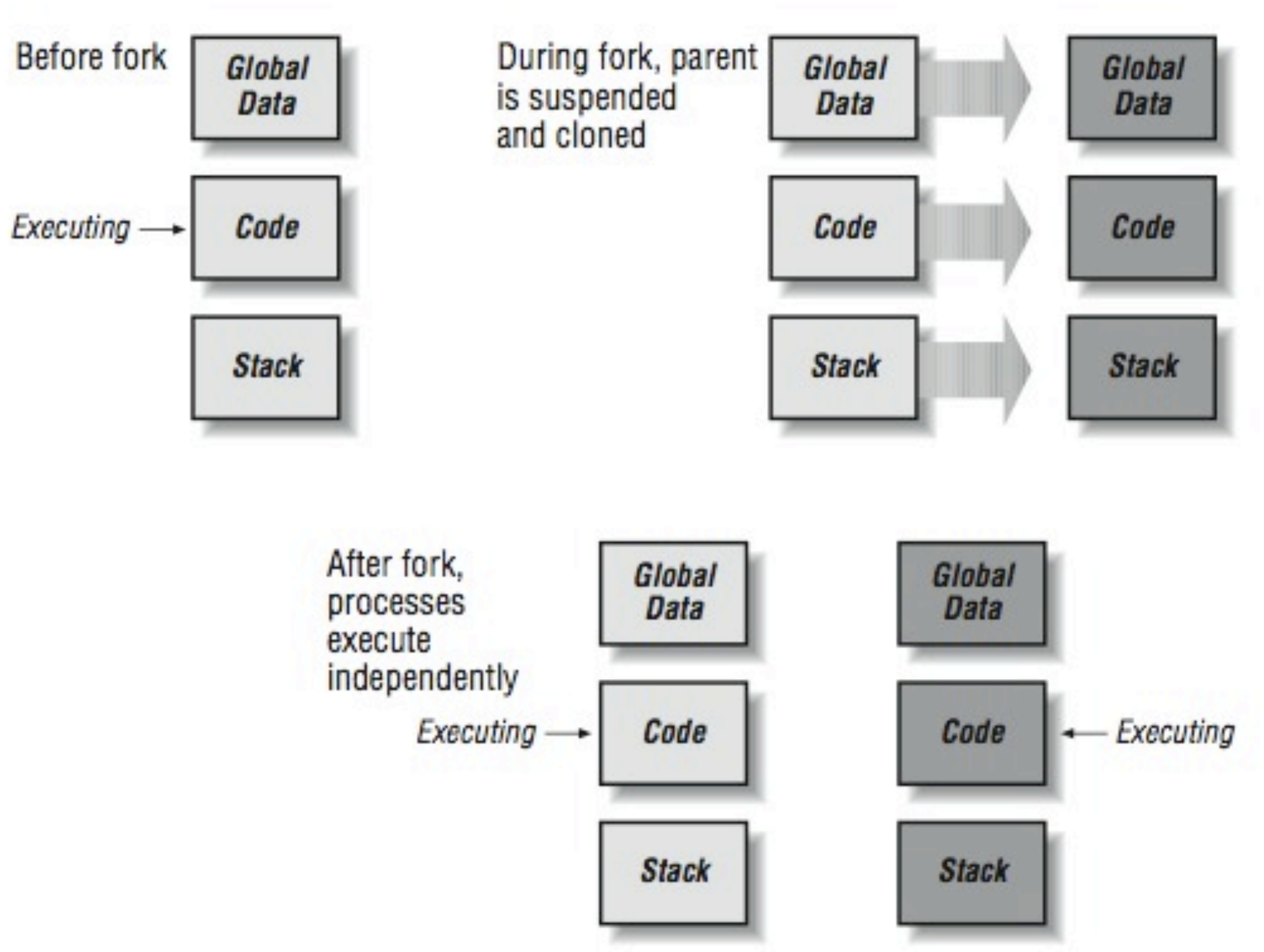


Diagram courtesy cnx.org

Child Process

- A cloned copy of a parent process
- Receives a new process ID and a parent process ID
- Does some work
- Dies

...sort of.

Photo Credit: Christopher Brian (2011 Toronto Zombie Walk)

Parent Responsibilities

- Reproduction
- Monitors child process status
- “Reap” zombie processes

Process Signals

Signal	Description
SIGCHLD	Child process died
SIGINT	User Interrupt
SIGTERM	Terminate
SIGKILL	Forcibly terminate

PHP Implementation

Requirements

- Unix-like operating system
- PHP 4.1+
- PHP PCNTL extension
  (compile with --enable-pcntl)
- PHP Semaphore extension, optional
  (--enable-sysvsem, --enable-sysvshm, --enable-sysvmsg)
- Plenty of memory
- Multiple CPU cores

Overview

1. Define signal handlers
2. Fetch a dataset
3. Fork off one child process for each item
4. Stop forking when a threshold is reached, and sleep
5. Reap a child process whenever SIGCHLD is received
6. If there's more work to do, fork more processes
7. When all child processes have been reaped, terminate

```
declare(ticks = 1);

// Setup our signal handlers
pcntl_signal(SIGTERM, "signal_handler");
pcntl_signal(SIGINT,  "signal_handler");
pcntl_signal(SIGCHLD, "signal_handler");
```

```
function signal_handler($signal)
{
    switch ($signal)
    {
        case SIGINT:
        case SIGTERM:
            // kill all child processes
            exit(0);
        case SIGCHLD:
            // reap a child process
            reap_child();
        break;
    }
}
```

```
$pid = pcntl_fork();

switch($pid)
{
    case 0:
        // Child process
        call_user_func($callback, $data);
        posix_kill(posix_getppid(), SIGCHLD);
        exit;
    case -1:
        // Parent process, fork failed
        throw new Exception("Out of memory!");
    default:
        // Parent process, fork succeeded
        $processes[$pid] = TRUE;
}
```

Repeat for each unit of work

```
function reap_child()
{
    // Check if any child process has terminated,
    // and if so remove it from memory
    $pid = pcntl_wait($status, WNOHANG);
    if ($pid < 0)
    {
        throw new Exception("Out of memory");
    }
    elseif ($pid > 0)
    {
        unset($processes[$pid]);
    }
}
```

Demo Time!

http://gist.github.com/4212160

easy
STAPLES®

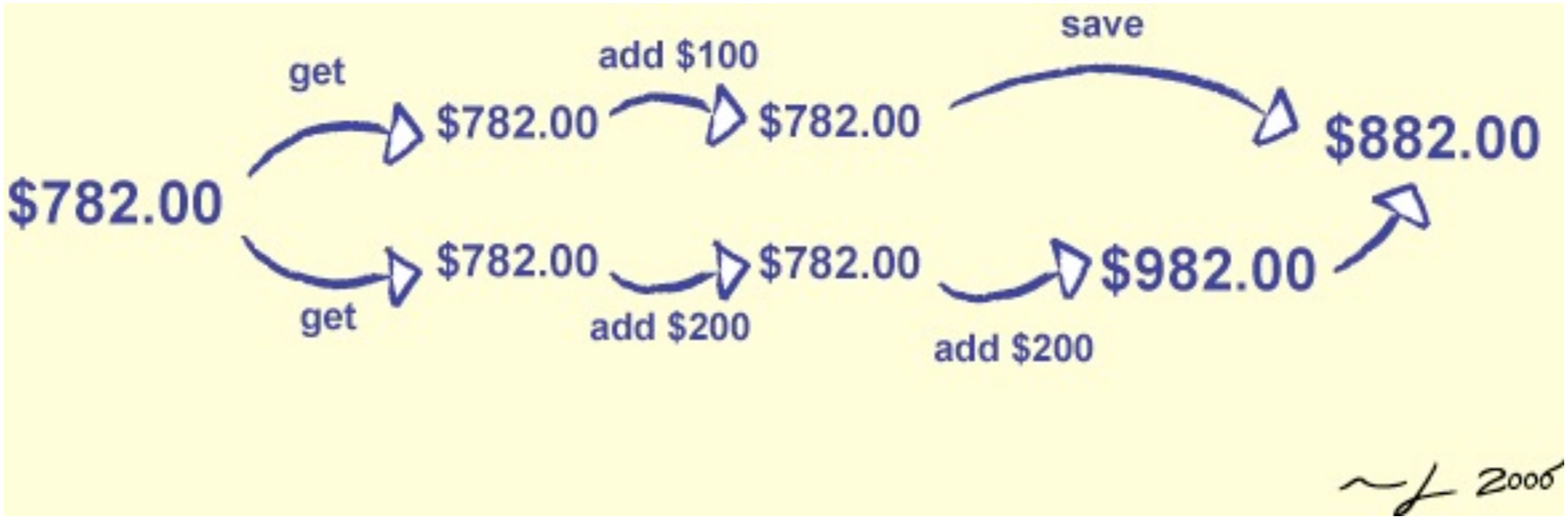
DON'T:

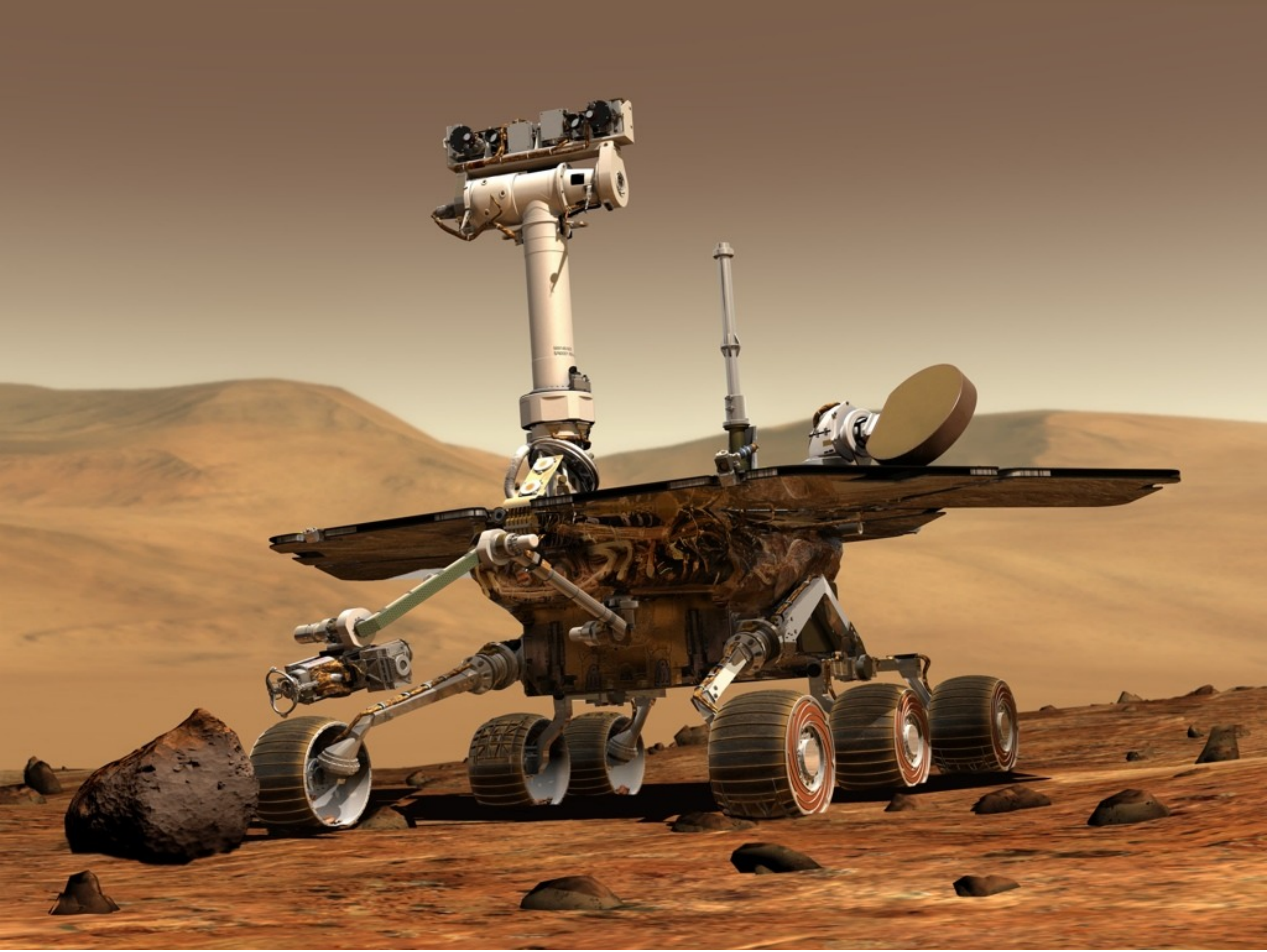
- DON'T fork a web process (CLI only!)
- DON'T overload your system
- DON'T open resources before forking
- DO respect common POSIX signals
- DO remove zombie processes
- DO force new database connections in children
  mysql_reconnect($s, $u, $p, TRUE);

Challenges

Race Conditions

- A logic bug where the result is affected by the sequence or timing of uncontrollable events
- Adding debug logic can change timing
- Dirty reads
- Lost data
- Unpredictable behavior
- Deadlocks, hanging, crashing


$782.00
get
$782.00
add $100
$782.00
save
$882.00
get
$782.00
add $200
$782.00
add $200
$982.00
~J 2006



Solutions

- Handle I/O in the parent process exclusively
- Manage resources with semaphores and/or mutexes

Semaphores

- Semaphore = atomically updated counter
- Mutex = binary semaphore with ownership
- PHP: sem_get(), sem_release()
- Transactional databases use semaphores

Deadlocks

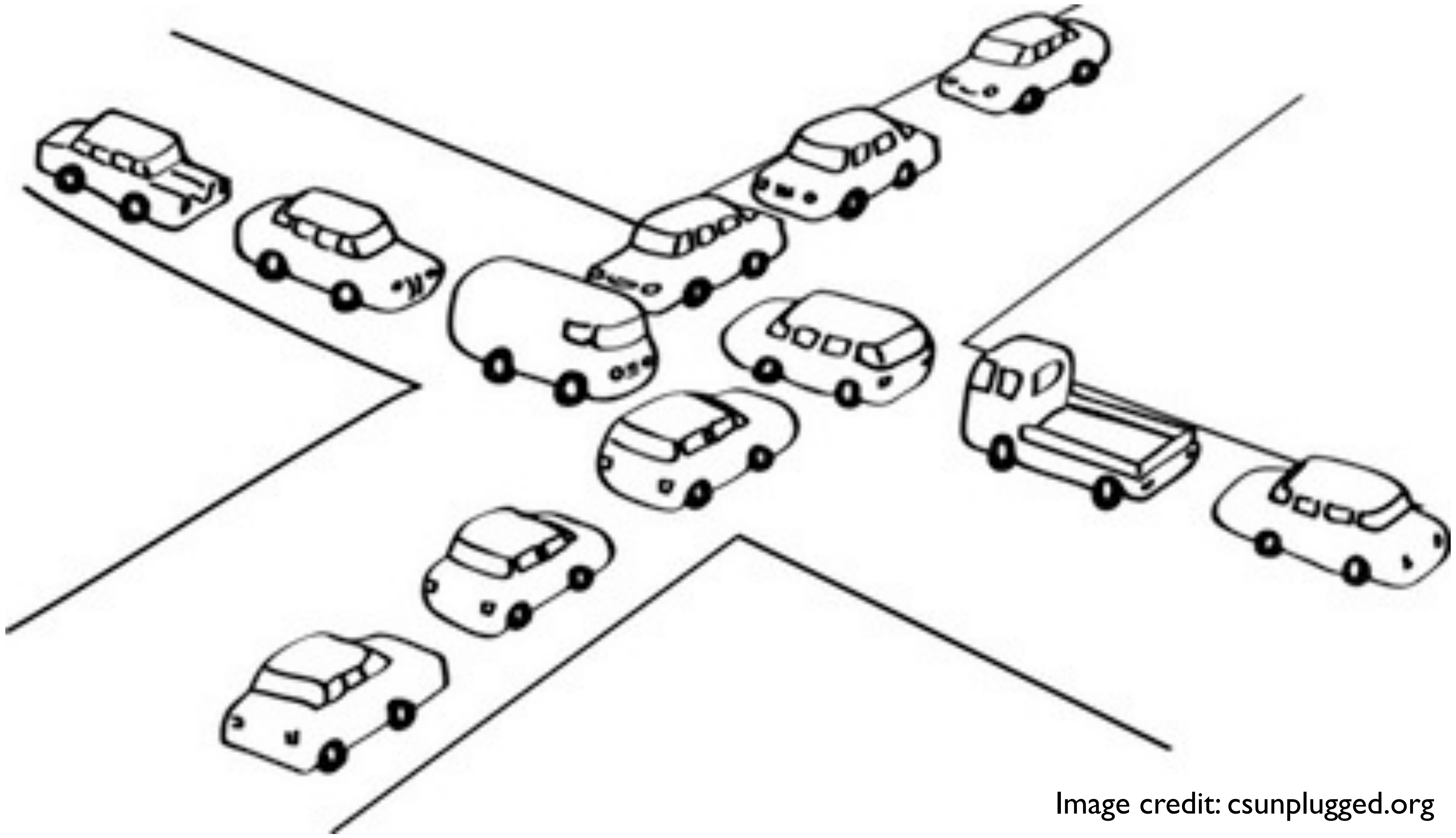


Image credit: csunplugged.org

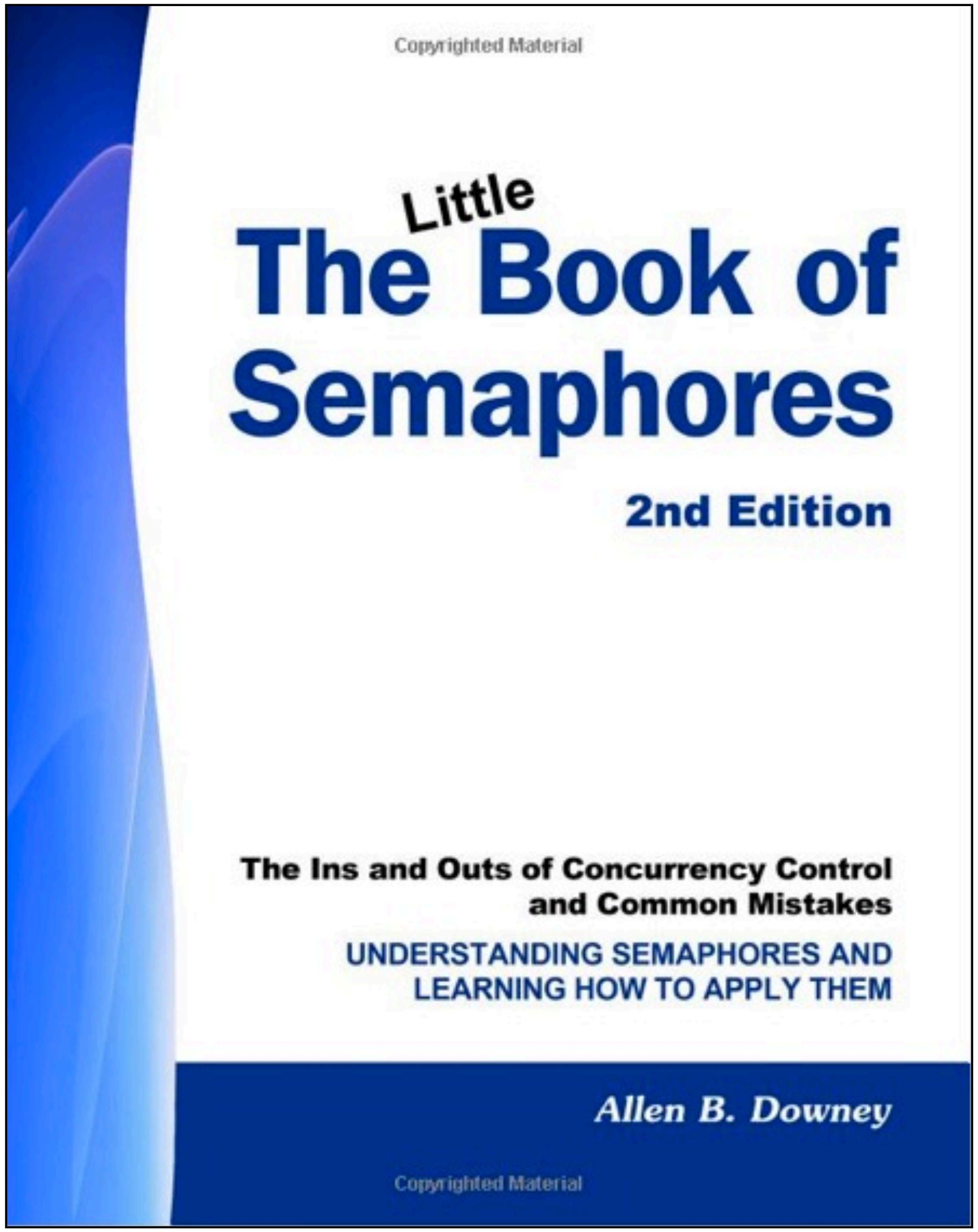
Bonus Slides

Shared Memory

- Advanced inter-process communication
- Pass data back to the parent process
- PHP Shared Memory extension (--enable-shmop)
- PHP System V Shared Memory extension (--enable-sysvshm)
  - More robust
  - Compatible with other languages

Daemonization

- Fork, kill parent
- Orphaned child process continues running
- Signal and error handling are critical
- Server daemons usually fork child processes to handle requests


Copyrighted Material

The Little Book of Semaphores

2nd Edition

The Ins and Outs of Concurrency Control and Common Mistakes

UNDERSTANDING SEMAPHORES AND LEARNING HOW TO APPLY THEM

Allen B. Downey

Copyrighted Material

Thank You!

@compwright

http://bit.ly/atlphpm