



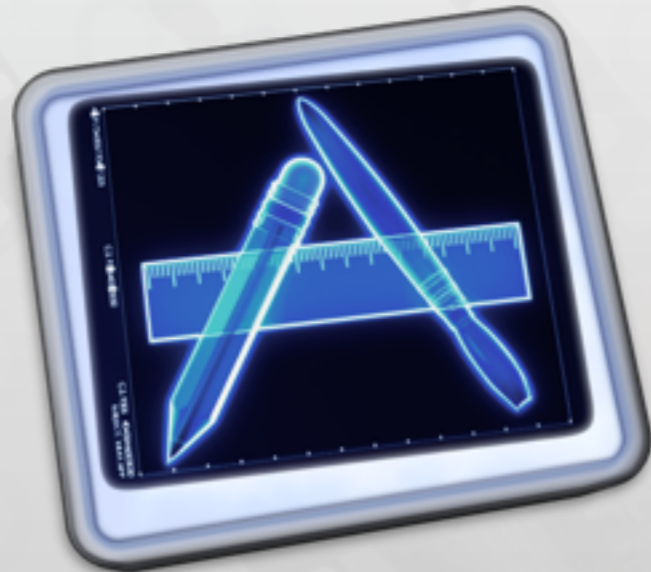
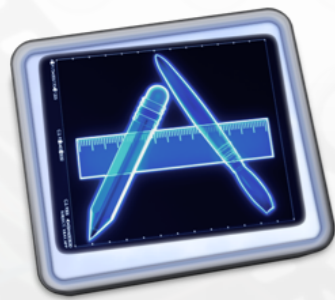
iPhone Application Programming

Lecture 8: Instruments

*Moritz Wittenhagen
Media Computing Group
RWTH Aachen University*

Winter Semester 2013/2014

<http://hci.rwth-aachen.de/iphone>



Instruments



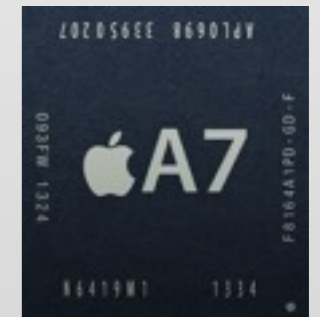
Where is the problem?

```
int main(int argc, const char * argv[])
{
    @autoreleasepool
    {
        for(int i = 0; i < HUGE_VAL; i++)
        {
            NSArray *array = @[];
            [[array retain] autorelease];
        }
    }
    return 0;
}
```



Analyzing Runtime Behavior

- Memory
 - How much is used?
 - When is it allocated / freed?
- CPU Time
 - Where is it spent?
 - How is it distributed between multiple CPUs?



How is my algorithm doing things?
Not: Is it doing them correctly?

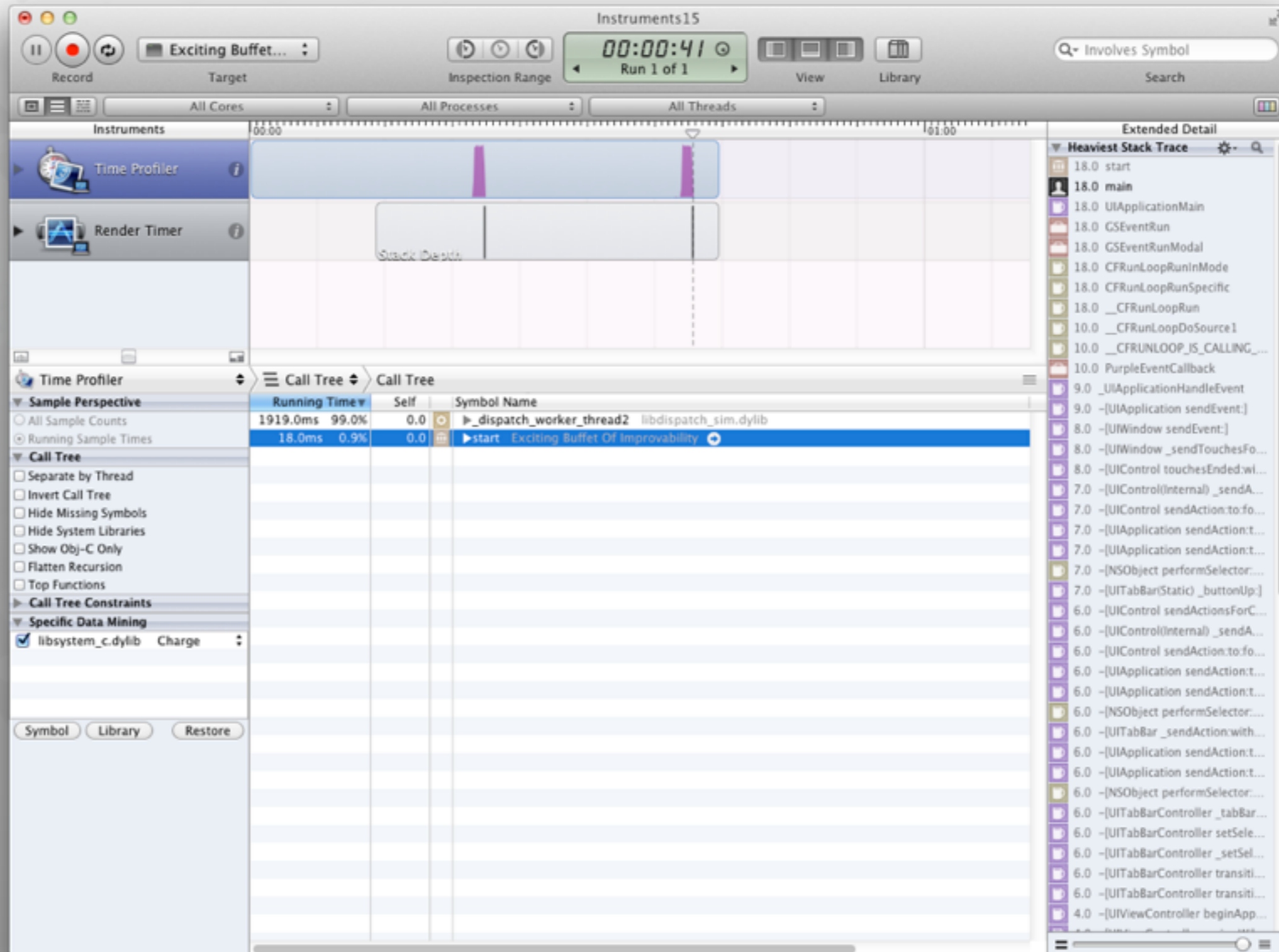


Instruments UI

Track
view
Data
view

Extended
details

Strategies



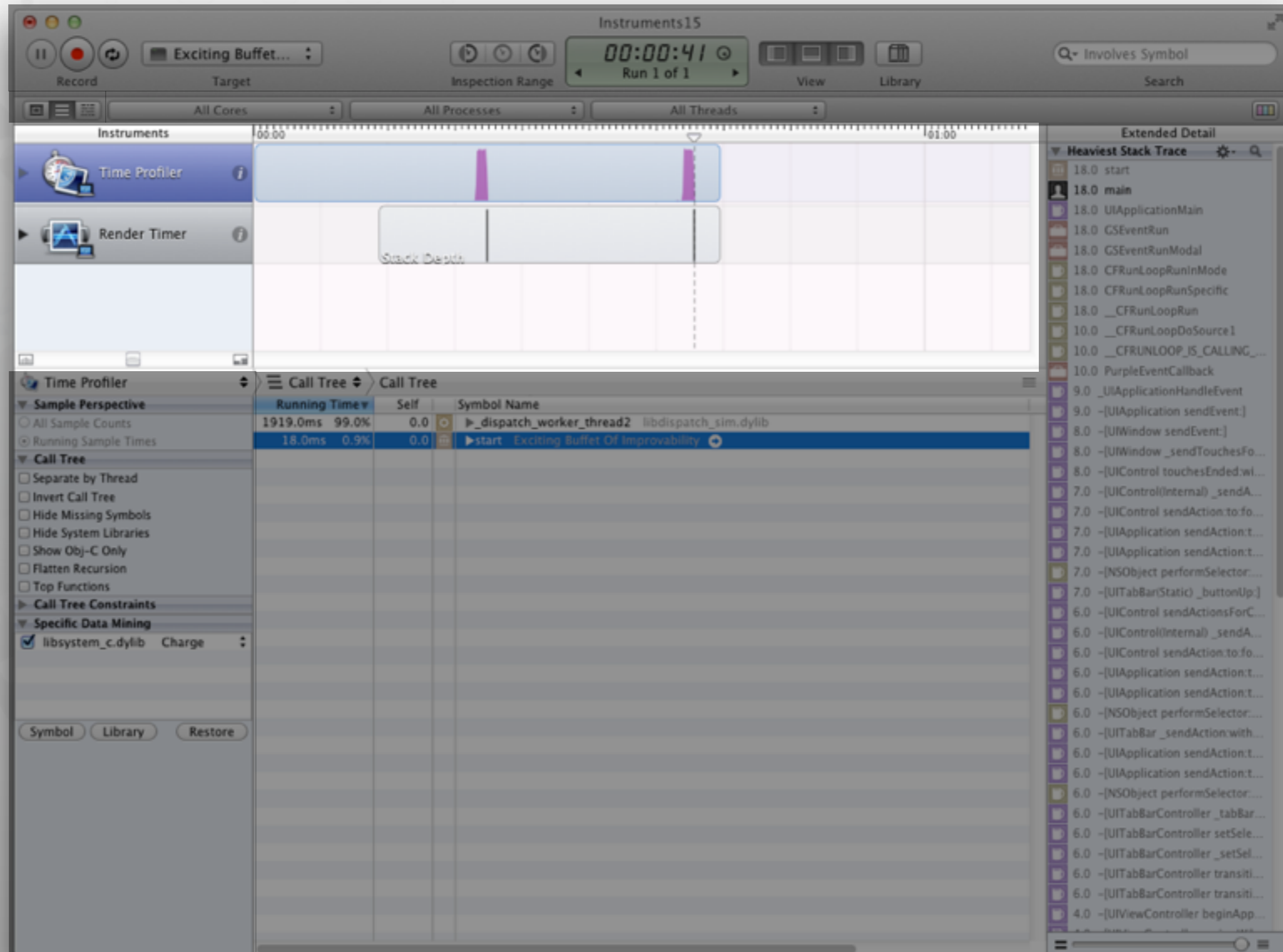


Instruments UI

Track
view
Data
view

Extended
details

Strategies





Instruments UI

Track
view
Data
view

Extended
details

Strategies

The screenshot displays the Instruments application interface. At the top, there are control buttons for Record, Target (Exciting Buffet...), Inspection Range (00:00:41, Run 1 of 1), View, and Library. Below this is a navigation bar with filters for All Cores, All Processes, and All Threads. The main area is divided into two panes: the left pane shows the Instruments list with 'Time Profiler' and 'Render Timer' selected, and the right pane shows the 'Extended Detail' view with a 'Heaviest Stack Trace' list. The 'Call Tree' pane is also visible, showing a table of running times and symbol names.

Running Time	Self	Symbol Name
1919.0ms 99.0%	0.0	_dispatch_worker_thread2 libdispatch_sim.dylib
18.0ms 0.9%	0.0	start Exciting Buffet Of Improvability



Instruments UI

Track
view
Data
view

Extended
details

Strategies

The screenshot displays the Instruments application interface. At the top, there are control buttons for Record, Target (Exciting Buffet...), Inspection Range (00:00:41, Run 1 of 1), View, and Library. Below this is a navigation bar with filters for All Cores, All Processes, and All Threads. The main area is divided into three sections: Instruments (Time Profiler and Render Timer), Call Tree (showing a table of running times), and Extended Detail (Heaviest Stack Trace).

Running Time	Self	Symbol Name
1919.0ms	99.0%	0.0 ▶_dispatch_worker_thread2 libdispatch_sim.dylib
18.0ms	0.9%	0.0 ▶start Exciting Buffet Of Improvability

The Heaviest Stack Trace on the right lists the following functions from top to bottom:

- 18.0 start
- 18.0 main
- 18.0 UIApplicationMain
- 18.0 GSEventRun
- 18.0 GSEventRunModal
- 18.0 CFRunLoopRunInMode
- 18.0 CFRunLoopRunSpecific
- 18.0 __CFRunLoopRun
- 10.0 __CFRunLoopDoSource1
- 10.0 __CFRUNLOOP_IS_CALLING_...
- 10.0 PurpleEventCallback
- 9.0 _UIApplicationHandleEvent
- 9.0 -[UIApplication sendEvent:]
- 8.0 -[UIWindow sendEvent:]
- 8.0 -[UIWindow _sendTouchesFo...
- 8.0 -[UIControl touchesEnded:wi...
- 7.0 -[UIControl(internal)_sendA...
- 7.0 -[UIControl sendAction:tofo...
- 7.0 -[UIApplication sendAction:t...
- 7.0 -[UIApplication sendAction:t...
- 7.0 -[NSObject performSelector:...
- 7.0 -[UITabBarStatic]_buttonUp:]
- 6.0 -[UIControl sendActionsForC...
- 6.0 -[UIControl(internal)_sendA...
- 6.0 -[UIControl sendAction:tofo...
- 6.0 -[UIApplication sendAction:t...
- 6.0 -[UIApplication sendAction:t...
- 6.0 -[NSObject performSelector:...
- 6.0 -[UITabBar _sendAction:with...
- 6.0 -[UIApplication sendAction:t...
- 6.0 -[UIApplication sendAction:t...
- 6.0 -[NSObject performSelector:...
- 6.0 -[UITabBarController _tabBar...
- 6.0 -[UITabBarController setSele...
- 6.0 -[UITabBarController _setSel...
- 6.0 -[UITabBarController transiti...
- 6.0 -[UITabBarController transiti...
- 4.0 -[UIViewController beginApp...

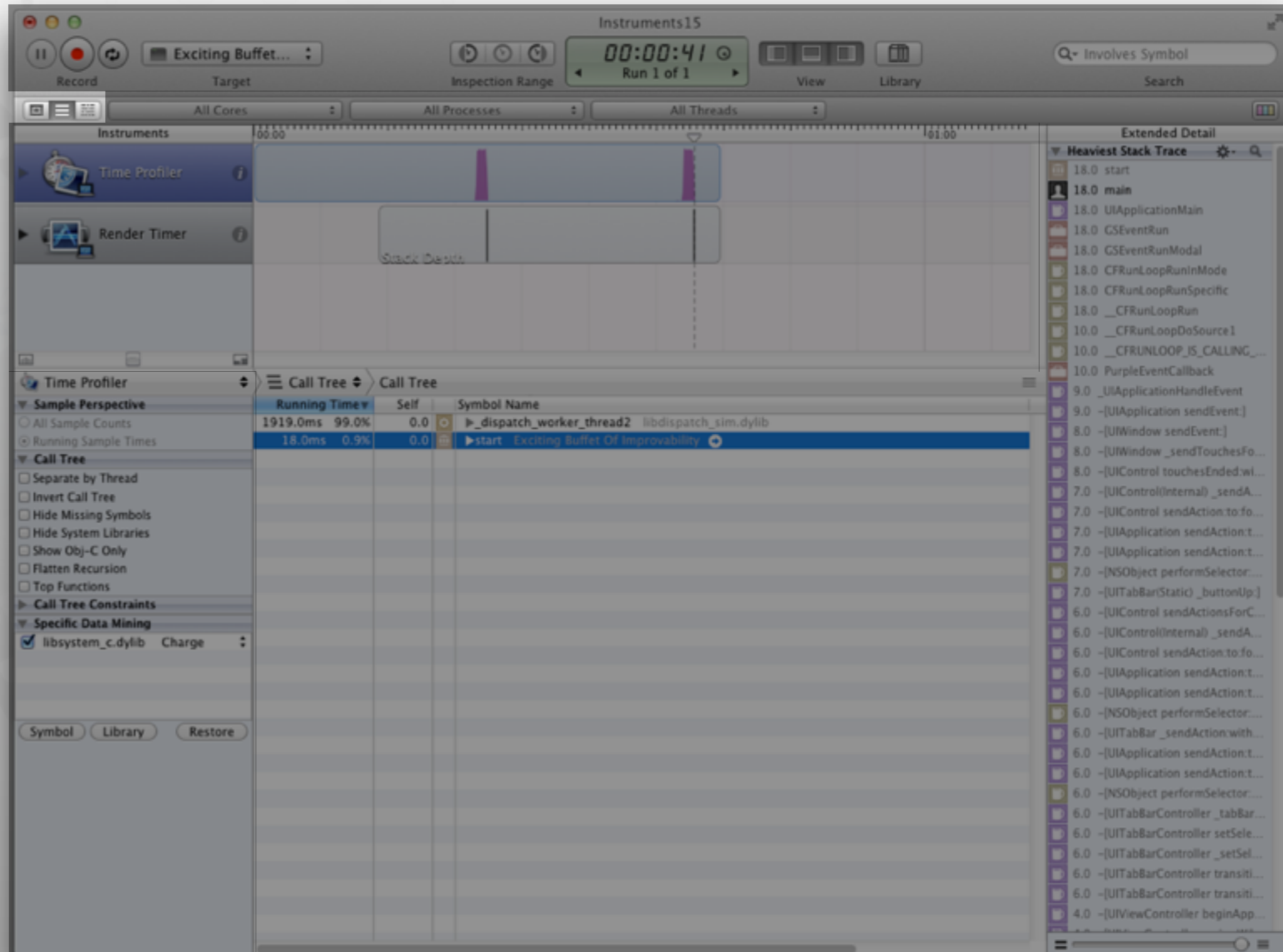


Instruments UI

Track
view
Data
view

Extended
details

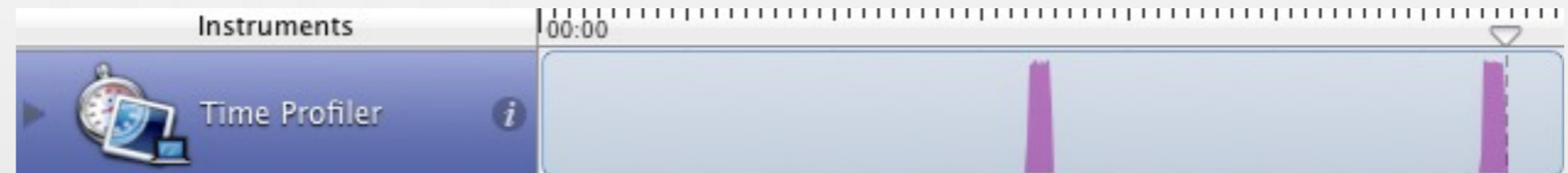
Strategies



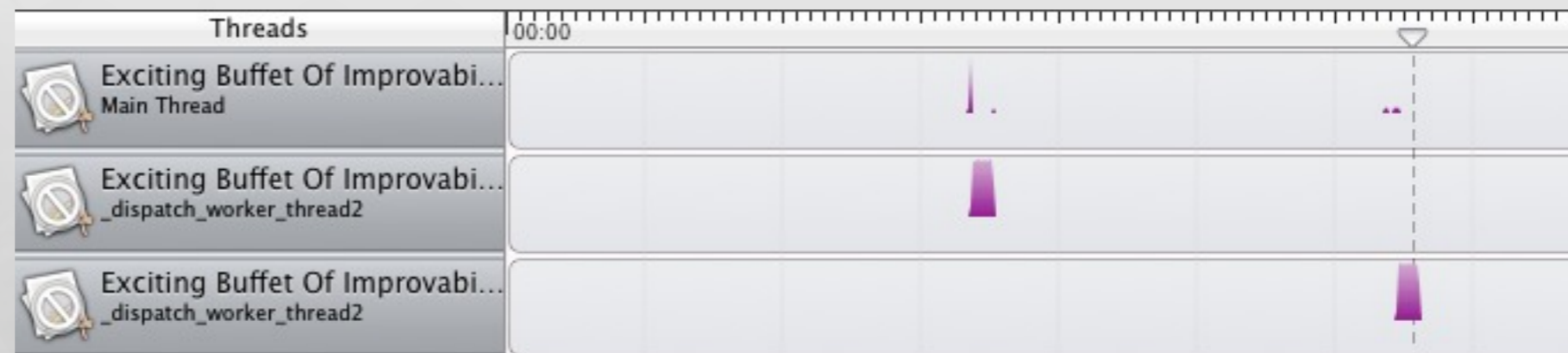


Strategies (Example)

- Instruments



- Threads



- CPUs

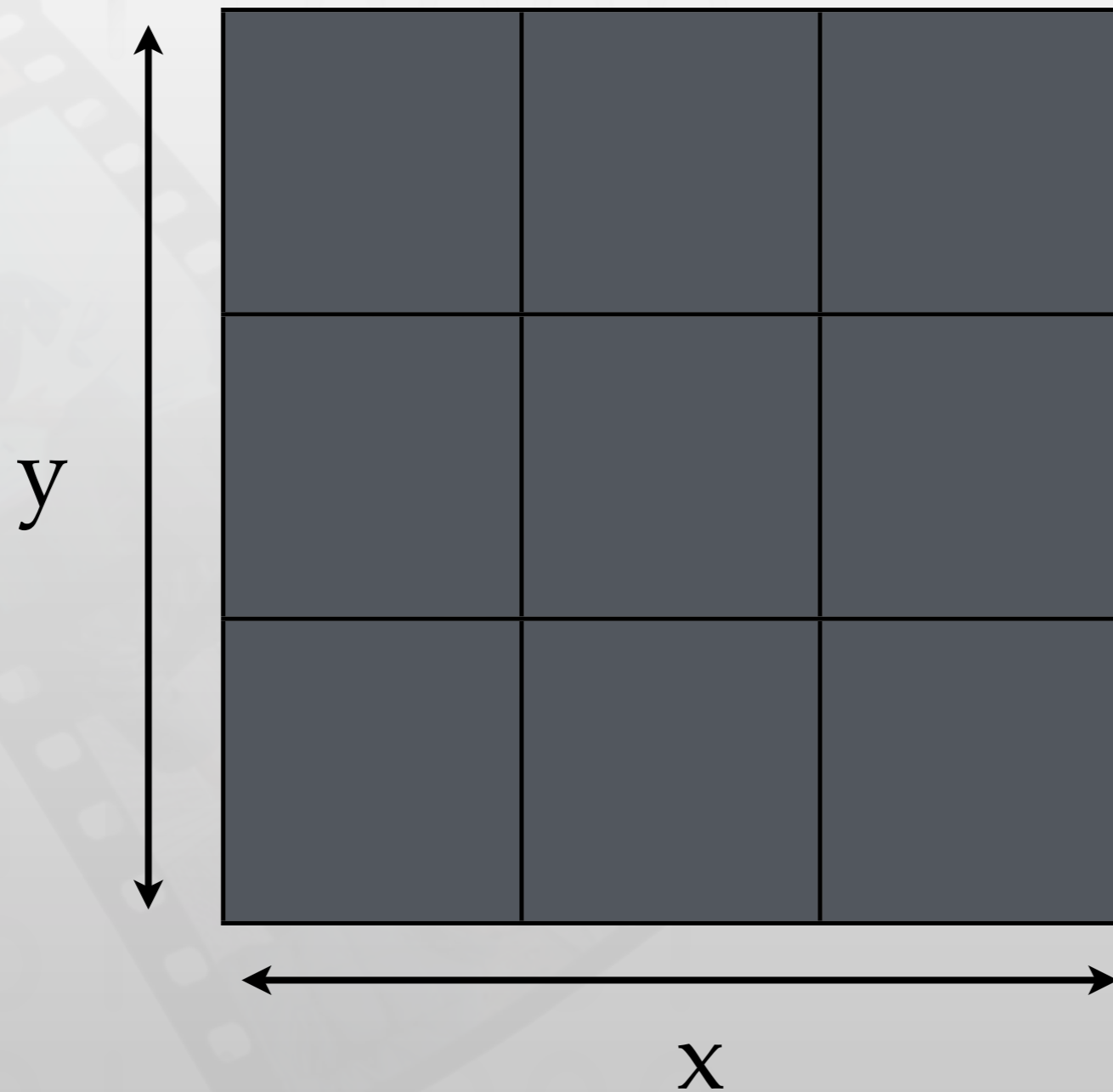




Demo Project

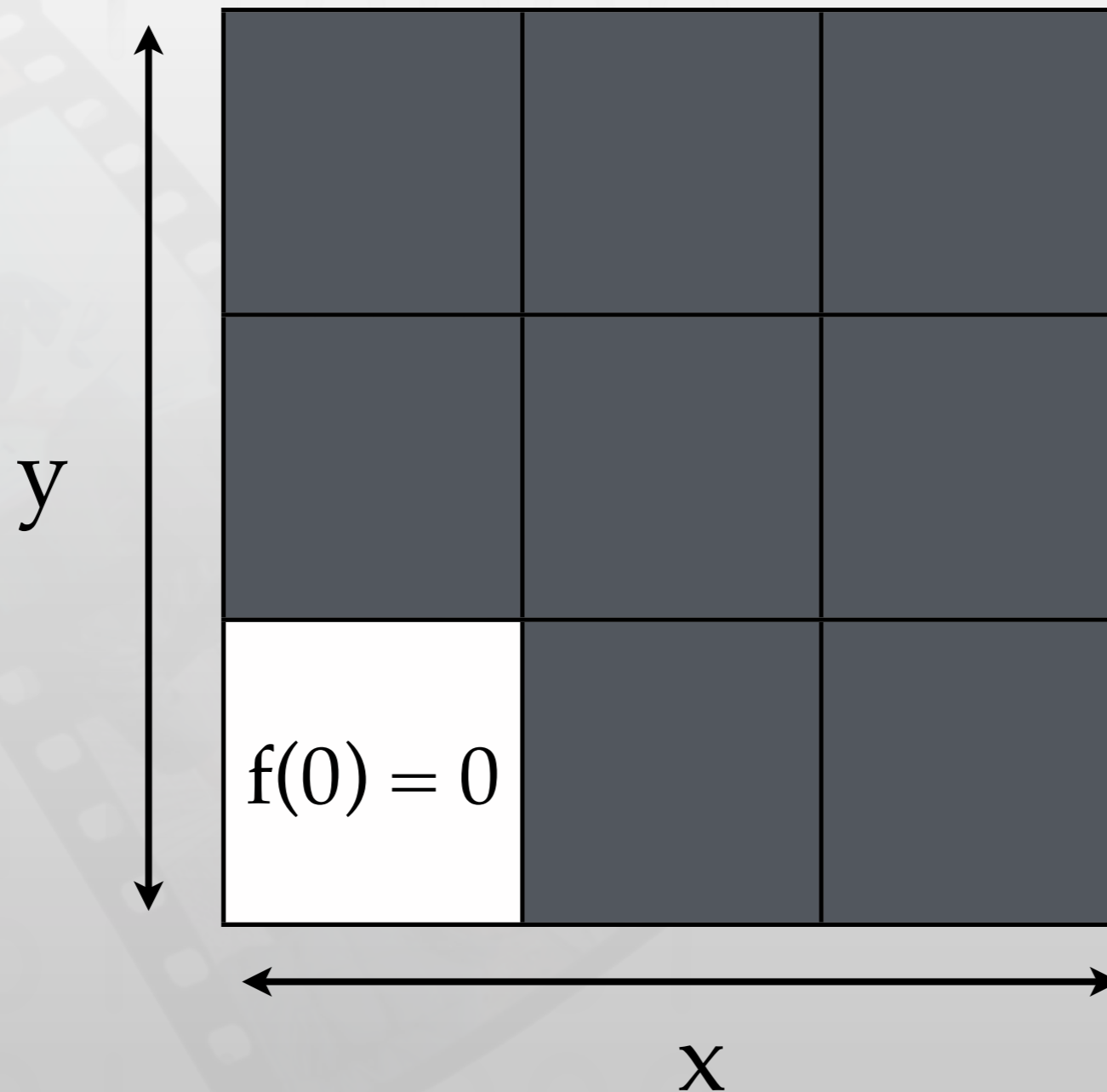


$$f(x) = x$$



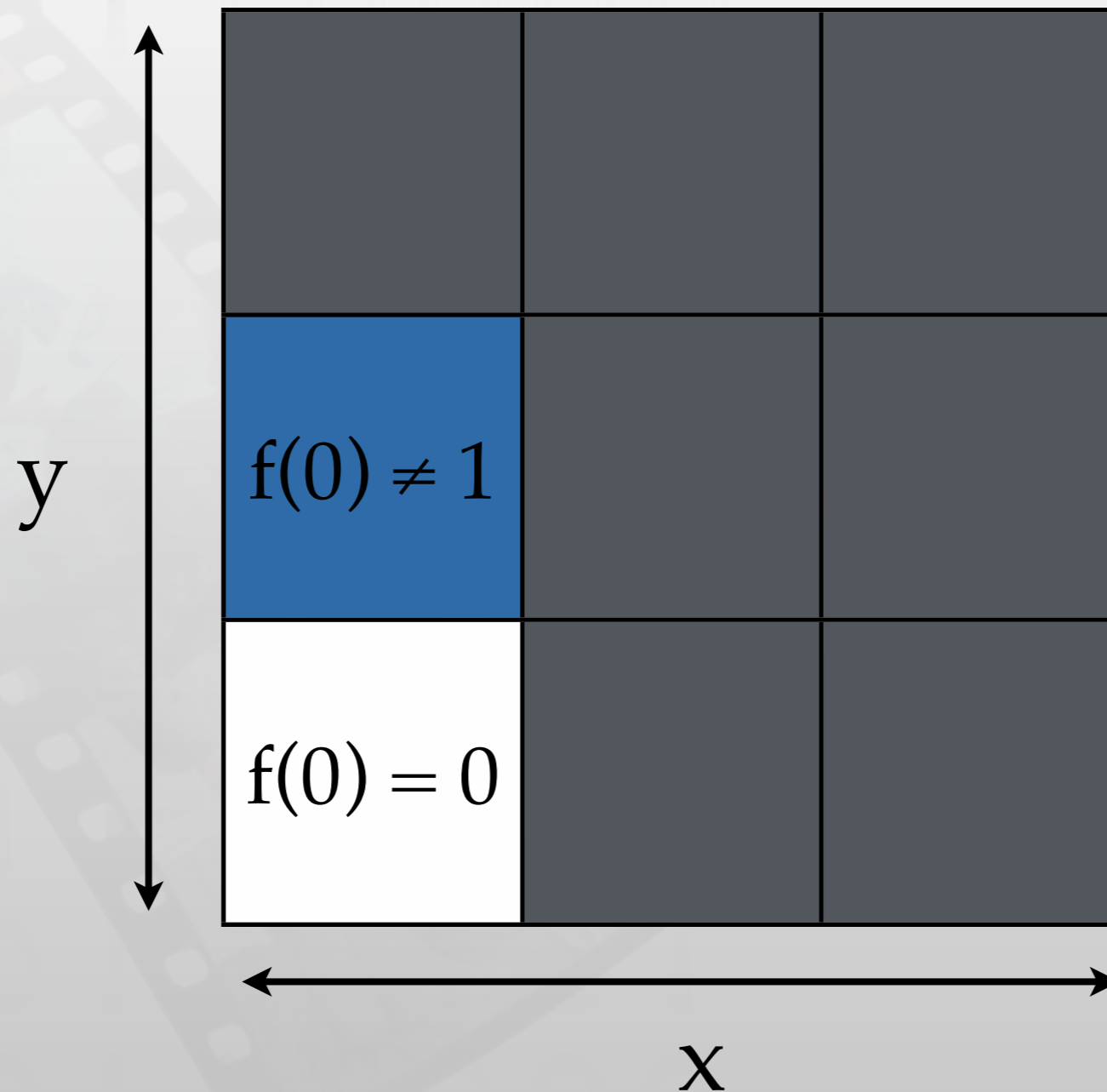


$$f(x) = x$$



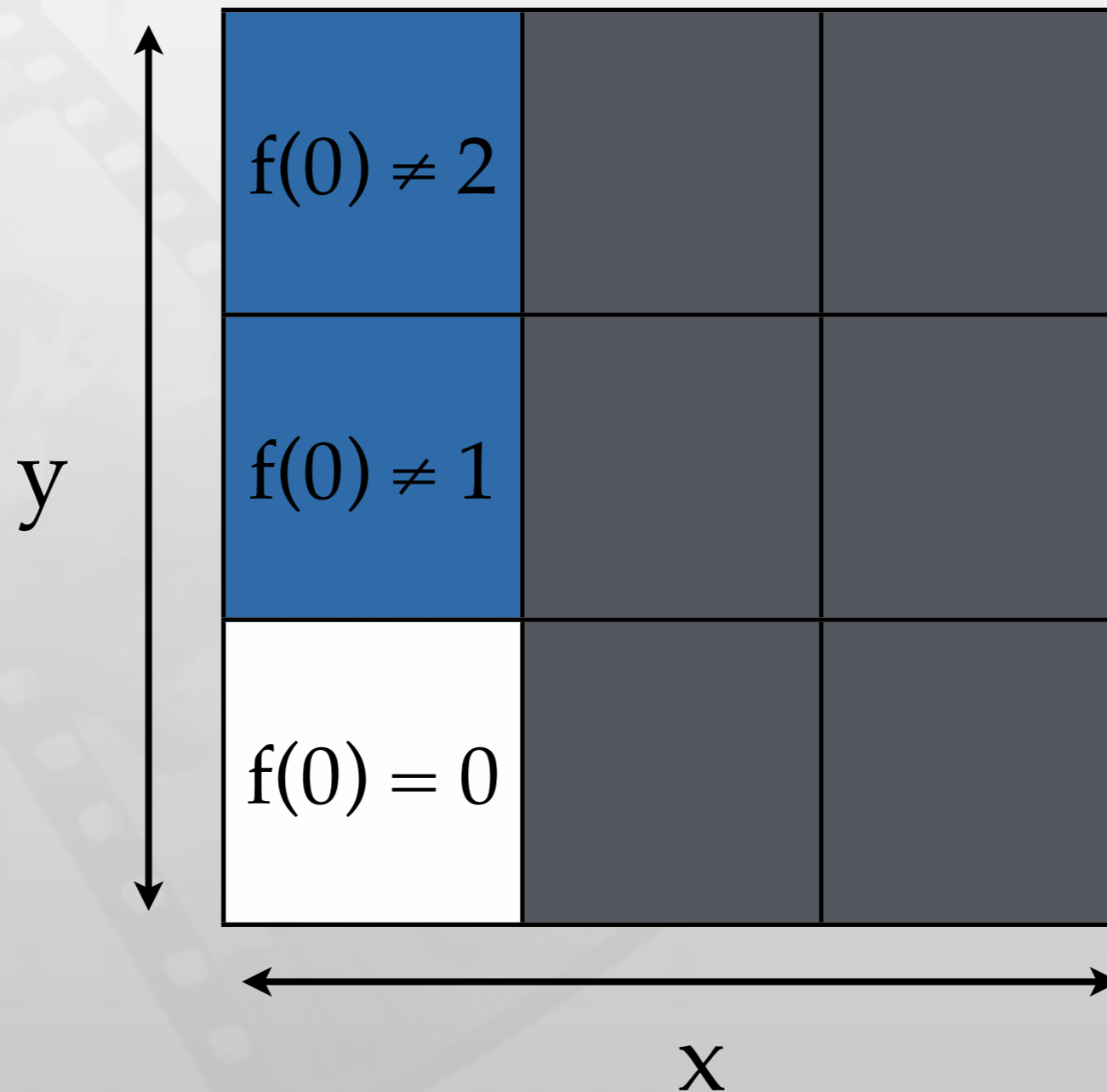


$$f(x) = x$$



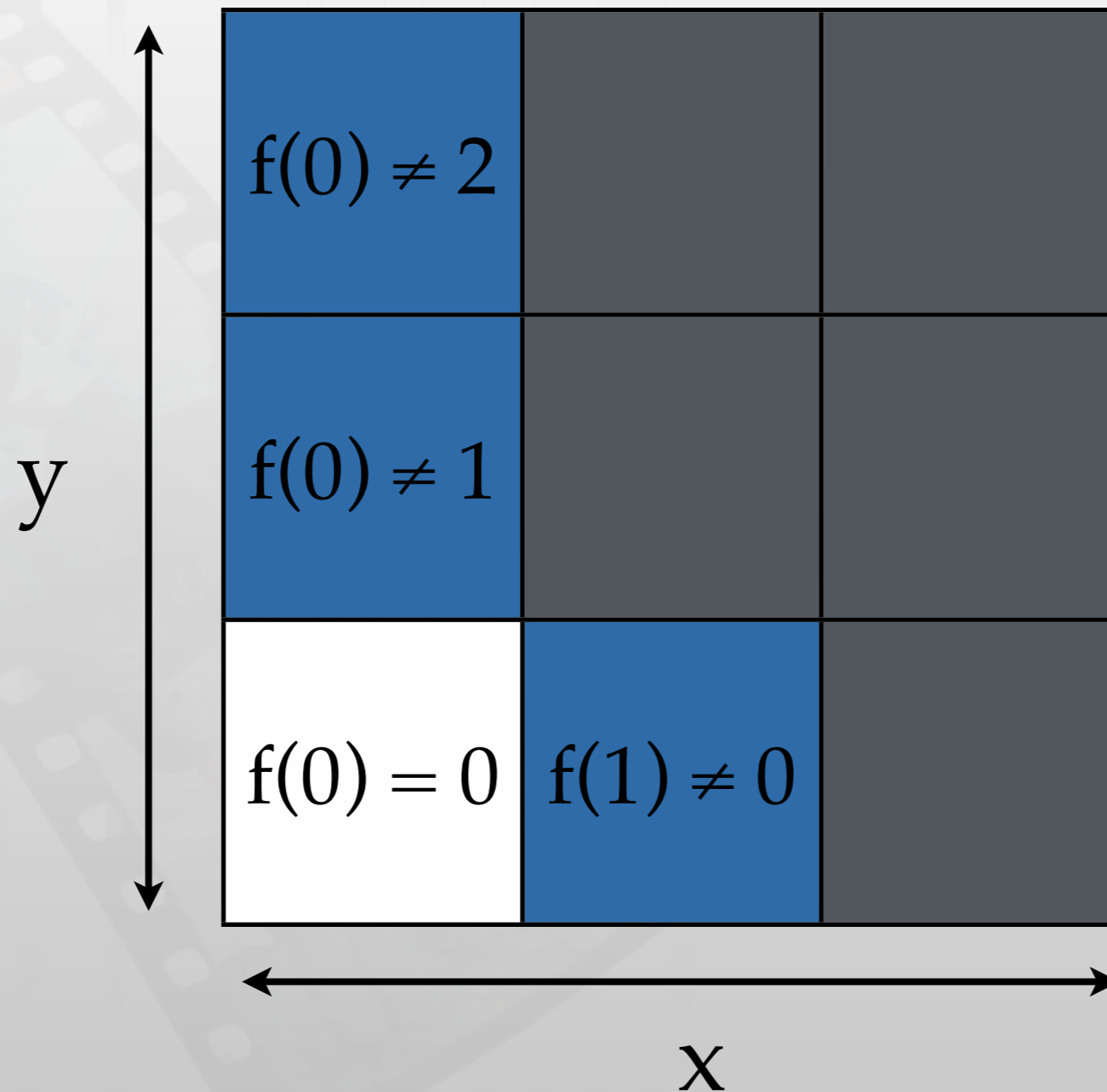


$$f(x) = x$$



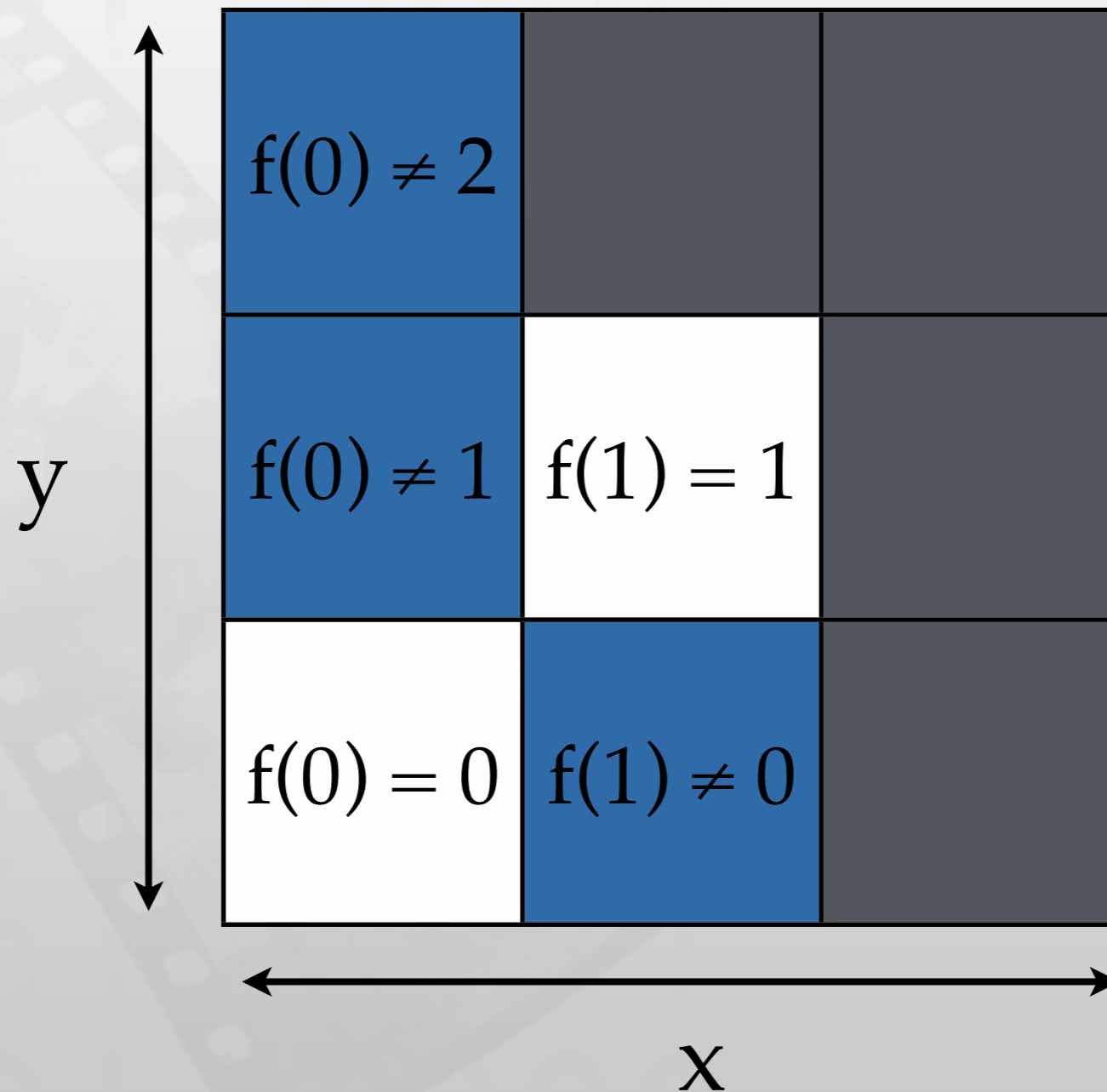


$$f(x) = x$$





$$f(x) = x$$





$$f(x) = x$$

y

$f(0) \neq 2$	$f(1) \neq 2$	$f(2) = 2$
$f(0) \neq 1$	$f(1) = 1$	$f(2) \neq 1$
$f(0) = 0$	$f(1) \neq 0$	$f(2) \neq 0$

x

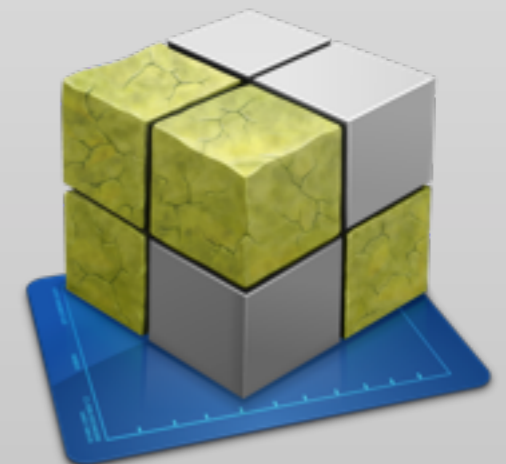


Simple Optimization Demo



Memory Analysis

- **Allocation**
 - Monitors memory allocation and reference counting
- **Leaks**
 - Checks for inaccessible memory
 - Finds retain cycles
- **Zombies**
 - Checks for freed memory being accessed





Leaks





Leaks



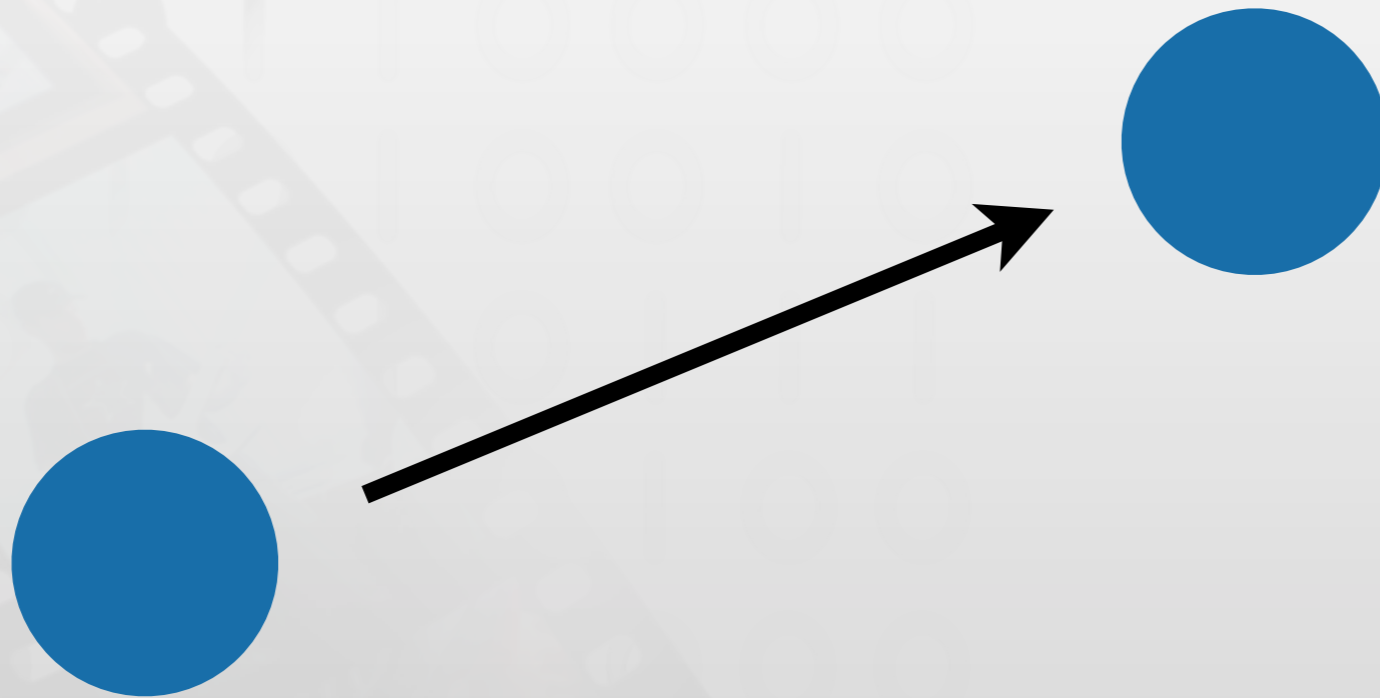


Retain Cycles



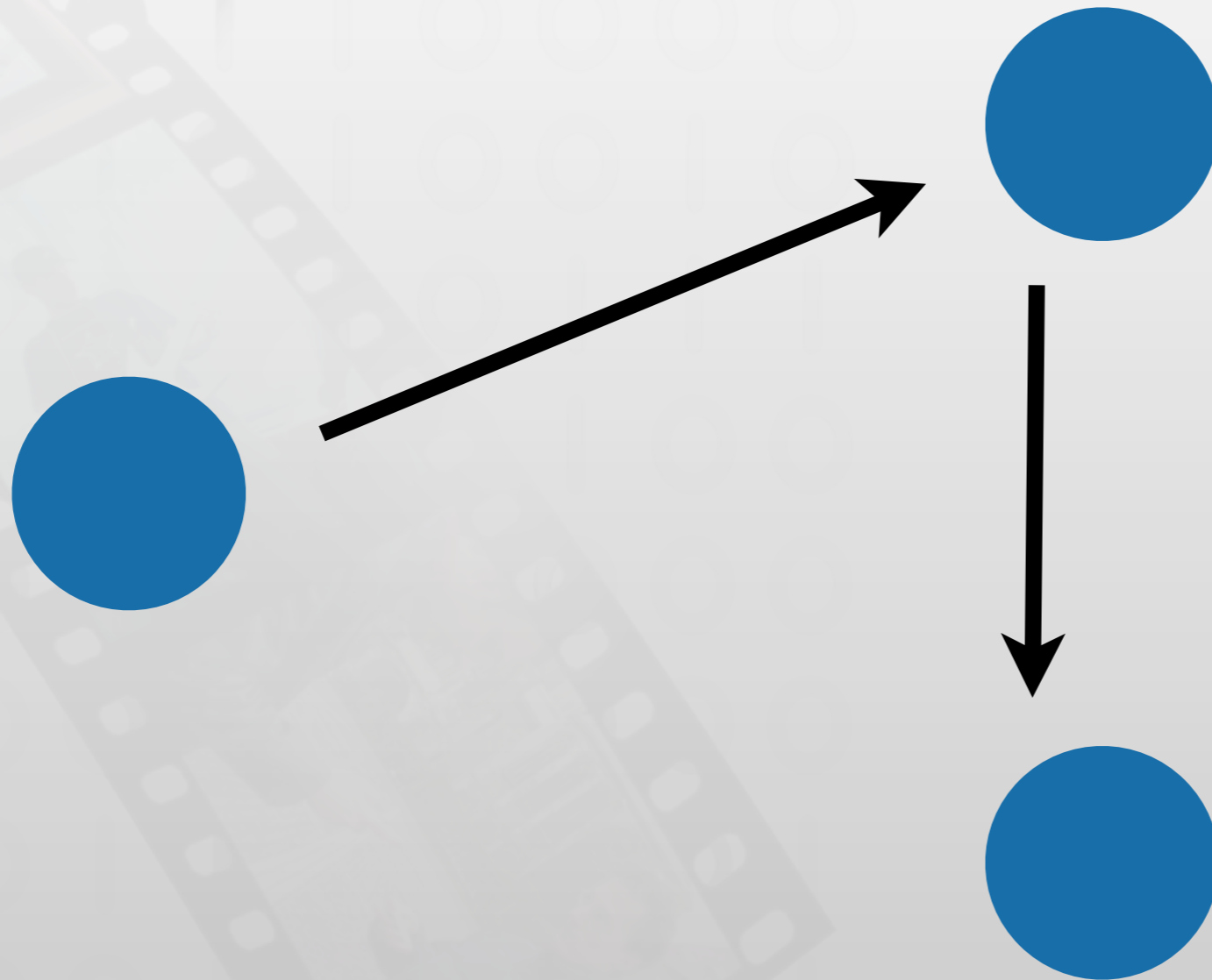


Retain Cycles



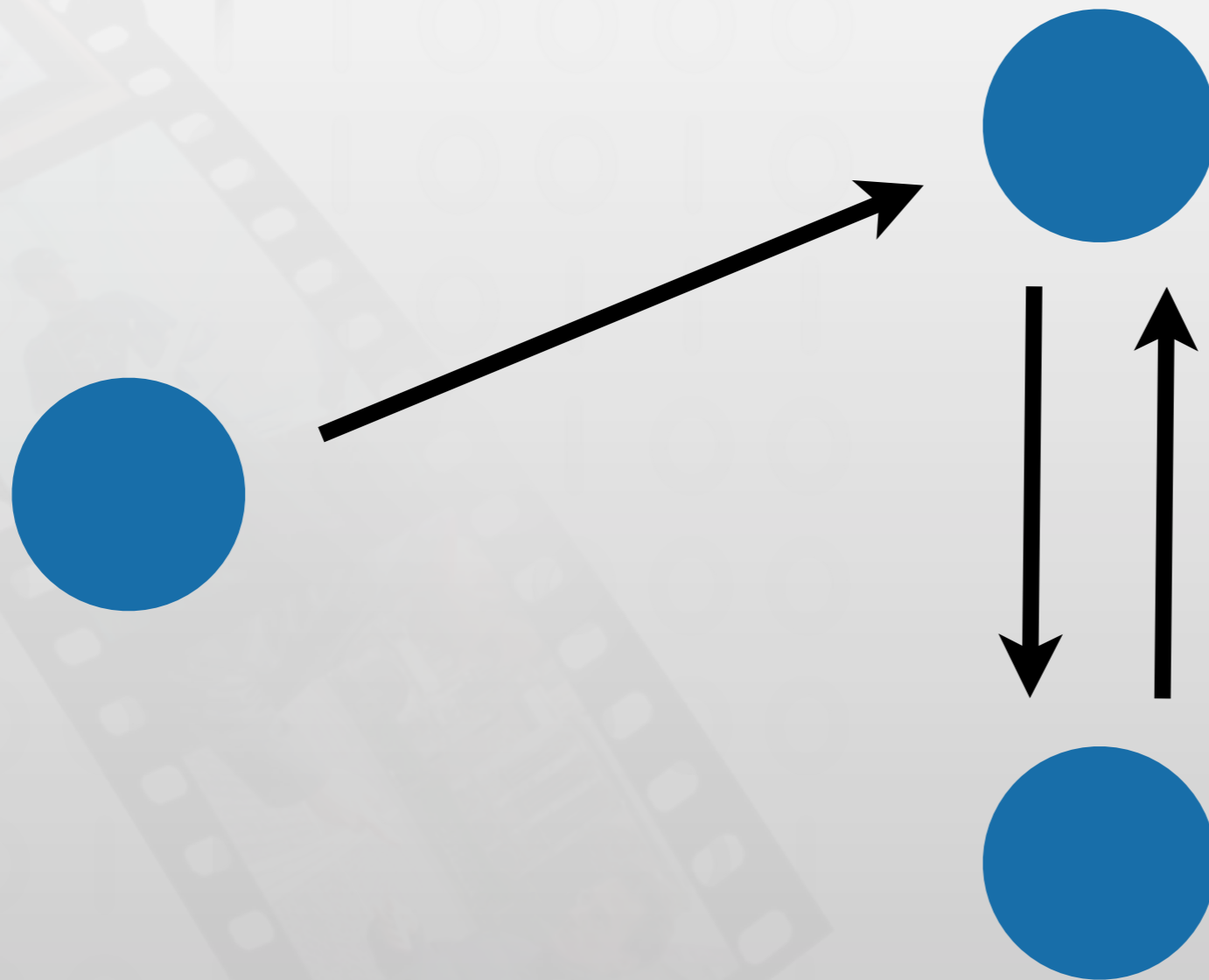


Retain Cycles



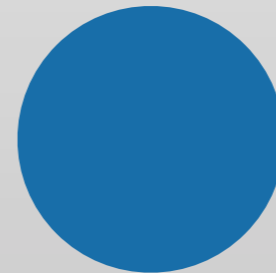


Retain Cycles





Retain Cycles





Allocations & Leaks Demo



Zombies

Freed memory being accessed



- “Good” Zombies
 - Obvious crashes
 - You release, system reuses, you try to access
 - Crash (usually `EXC_BAD_ACCESS`)
- Bad Zombies
 - No crash, or crash at strange location
 - You release, you allocate something, you try to access
 - Weird side-effects



Zombies Demo



Using Memory Instruments

- When you are done with a task: **Leaks**
- Whenever you get strange crashes or inexplicable values: **Zombies**
- You can use the simulator



Profiling



- Check in regular intervals what the CPU is doing
- Time Profiling
 - Where does the CPU spend time?
 - Distribution of work between threads / CPUs
- System Trace
 - What is the system doing?
 - Thread scheduling
 - Paging
 - System calls





Time Profiling Demo



Using Time Profiling



- When your app seems too slow
 - Identify hotspots
 - Identify opportunities for parallelization
 - Identify parallelization issues (e.g. forced serial execution)
- Use on iOS Device



System Trace Demo



Using System Trace



- When results of Time Profiling are insufficient
 - Excessive context switching
 - Paging issues
 - Find opportunities to group system calls
- Use on iOS Device



$$f(x) = x$$

Layout in Memory 

Order of Drawing 

y

$f(0) \neq 2$	$f(1) \neq 2$	$f(2) = 2$
$f(0) \neq 1$	$f(1) = 1$	$f(2) \neq 1$
$f(0) = 0$	$f(1) \neq 0$	$f(2) \neq 0$

x

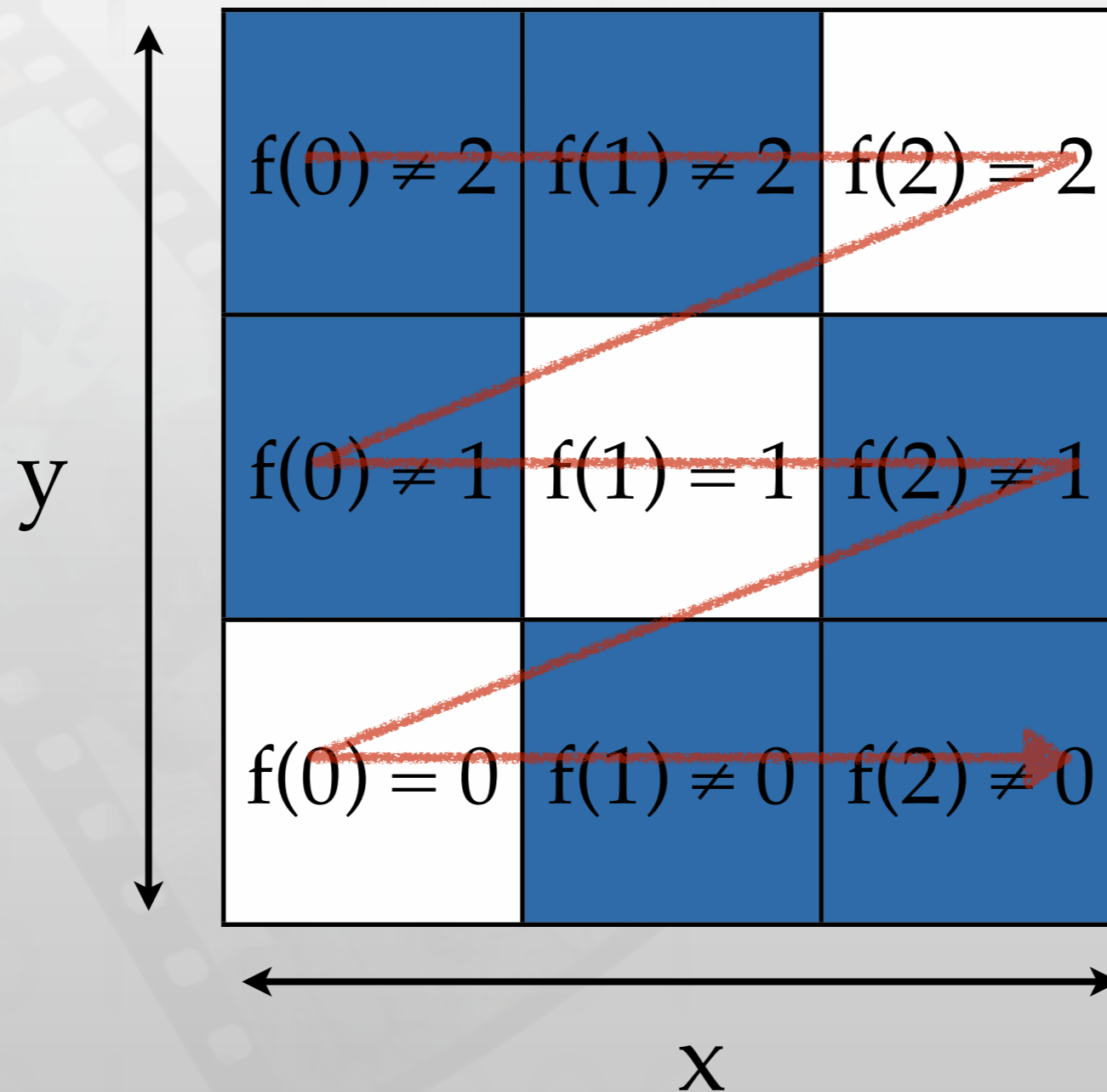


$$f(x) = x$$

Layout in Memory



Order of Drawing



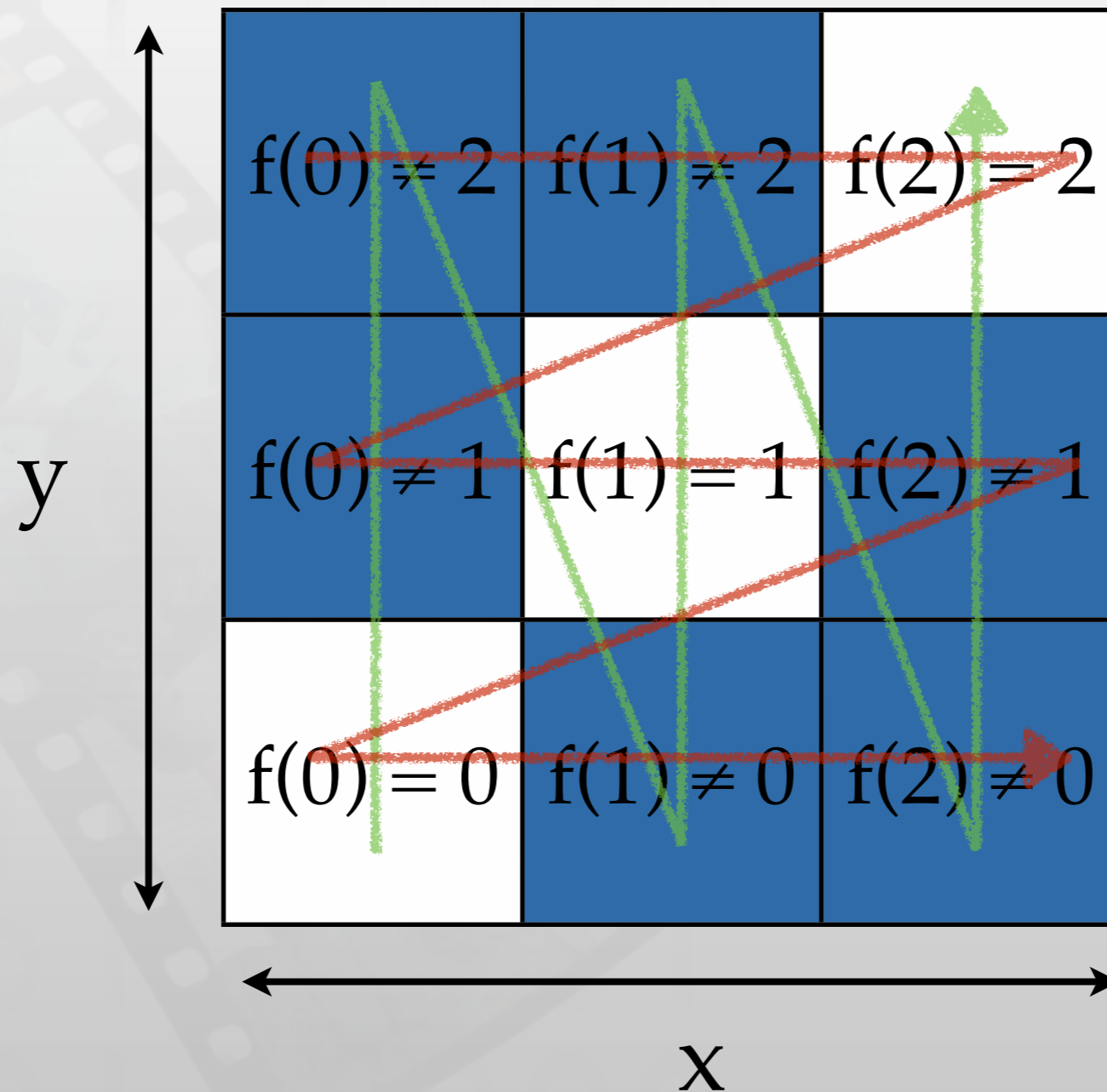


$$f(x) = x$$

Layout in Memory



Order of Drawing





$$f(x) = x$$

Layout in Memory



Order of Drawing



y

	$f(0) \neq 2$	$f(1) \neq 2$	$f(2) = 2$
	$f(0) \neq 1$	$f(1) = 1$	$f(2) \neq 1$
	$f(0) = 0$	$f(1) \neq 0$	$f(2) \neq 0$

x

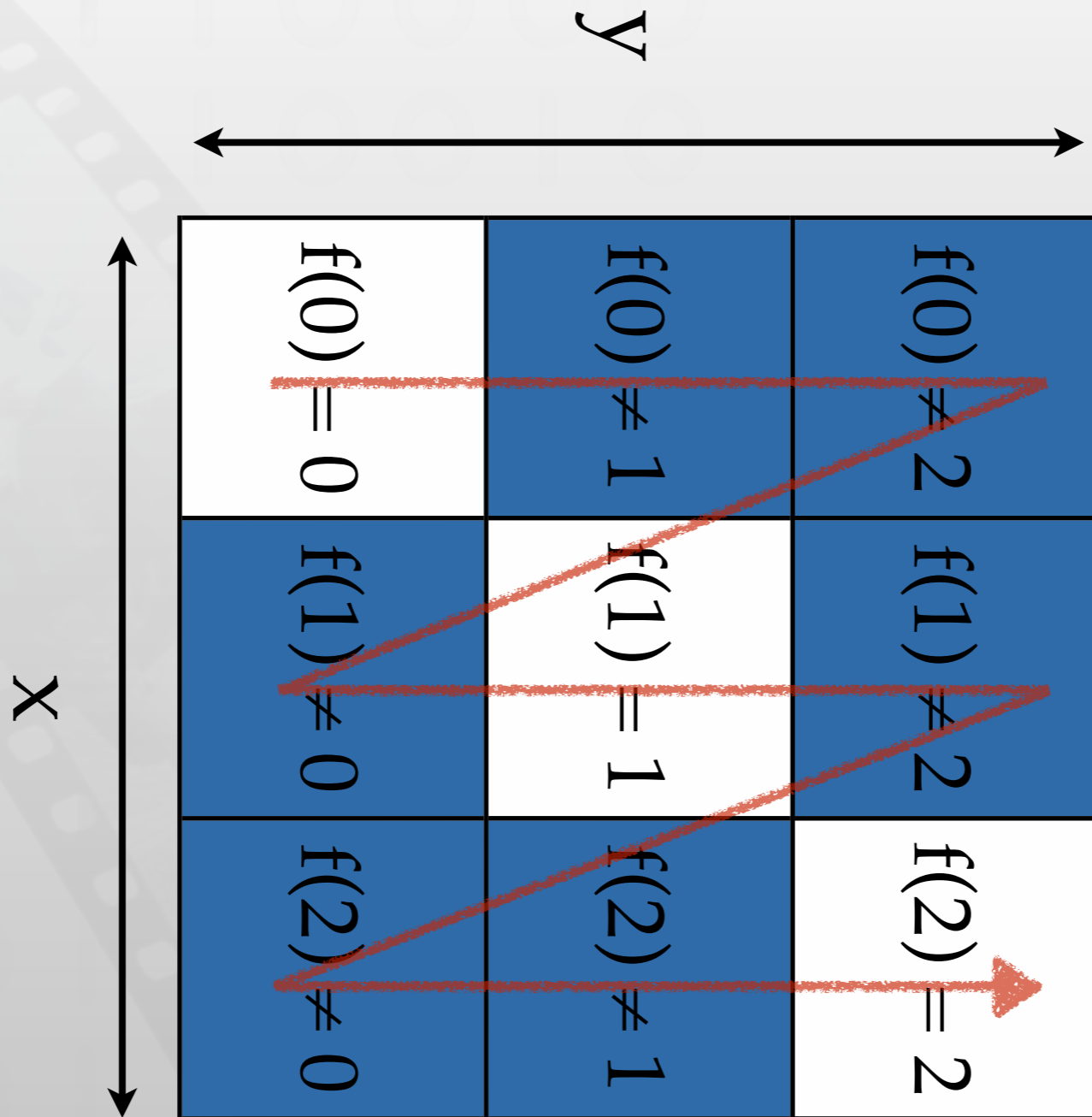


$$f(x) = x$$

Layout in Memory



Order of Drawing



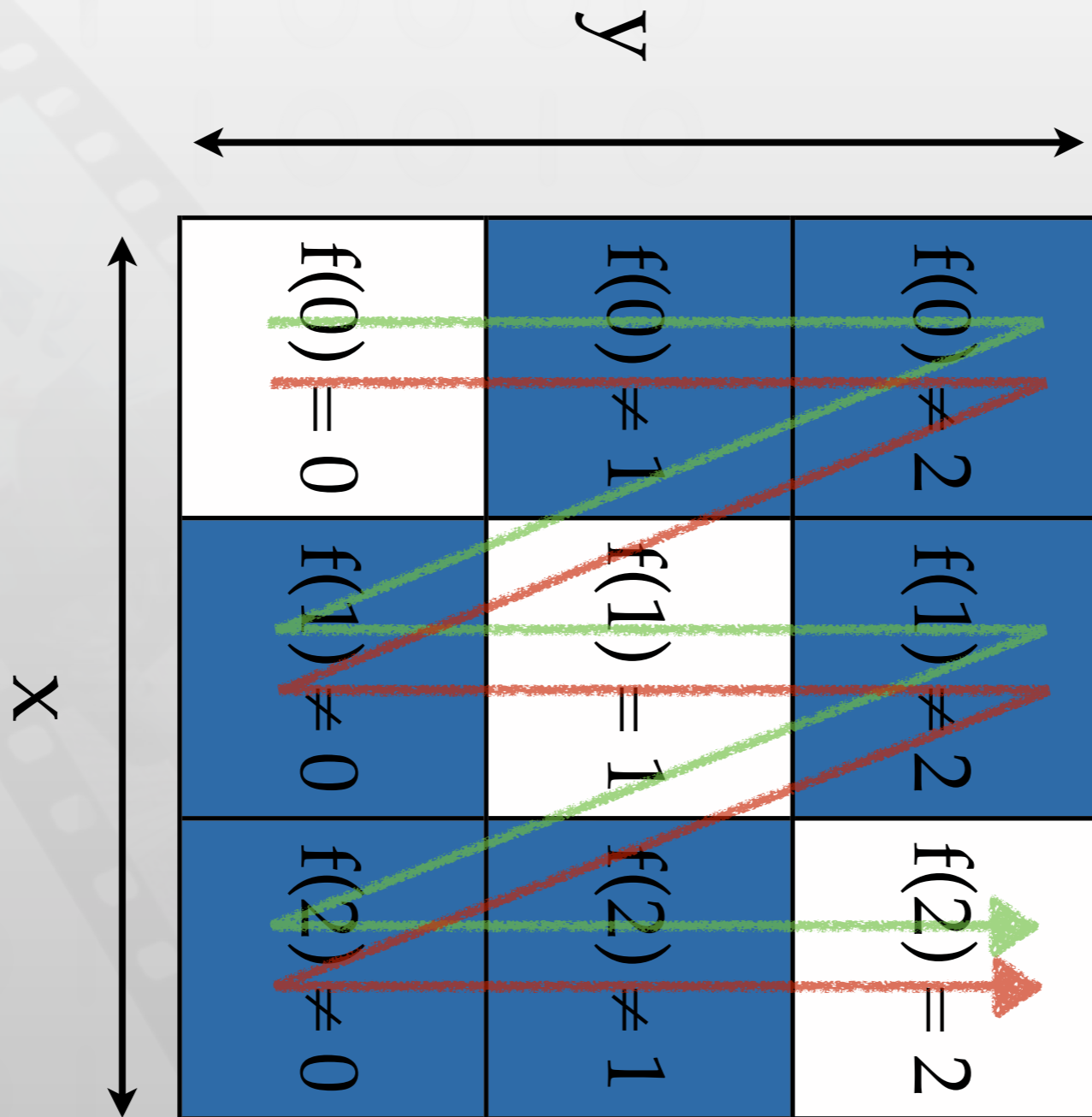


$$f(x) = x$$

Layout in Memory



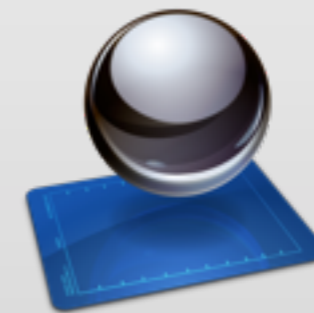
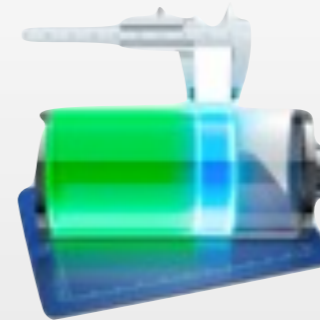
Order of Drawing





Other Instruments

- Energy Diagnostic
- Core Animation
- OpenGL ES
- System Usage
- UI Automation





Summary

- **General**
 - Find bugs at runtime
 - Increase algorithmic efficiency
- **Profiling**
 - Identify bottlenecks
 - Parallelization
- **Memory Instruments**
 - Sanity checks to find leaks and zombies
 - Increase memory efficiency