Back to the Futures:

Incremental Parallelization of Existing Sequential Runtimes

James Swaine[1] Kevin Tew[2] Peter Dinda[1]
Robert Bruce Findler[1] Matthew Flatt[2]

[1]Northwestern University
[2]University of Utah

Slow-Path Barricading

- Incremental
- Seq. performance intuition carries over
- Low development investment
- Good scaling (negligible sequential overhead)

An Observation

- Runtime "fast-path" operations generally have few side effects
- Thus, safe for parallelism

Slow-Path Barricading

- Partition operations into 3 categories:
  - Safe (run in parallel)
  - Unsafe (runtime side effects)
  - A few others (a priori unsafe, but important)

- Safety may be dependent on arguments

Slow-Path Barricading

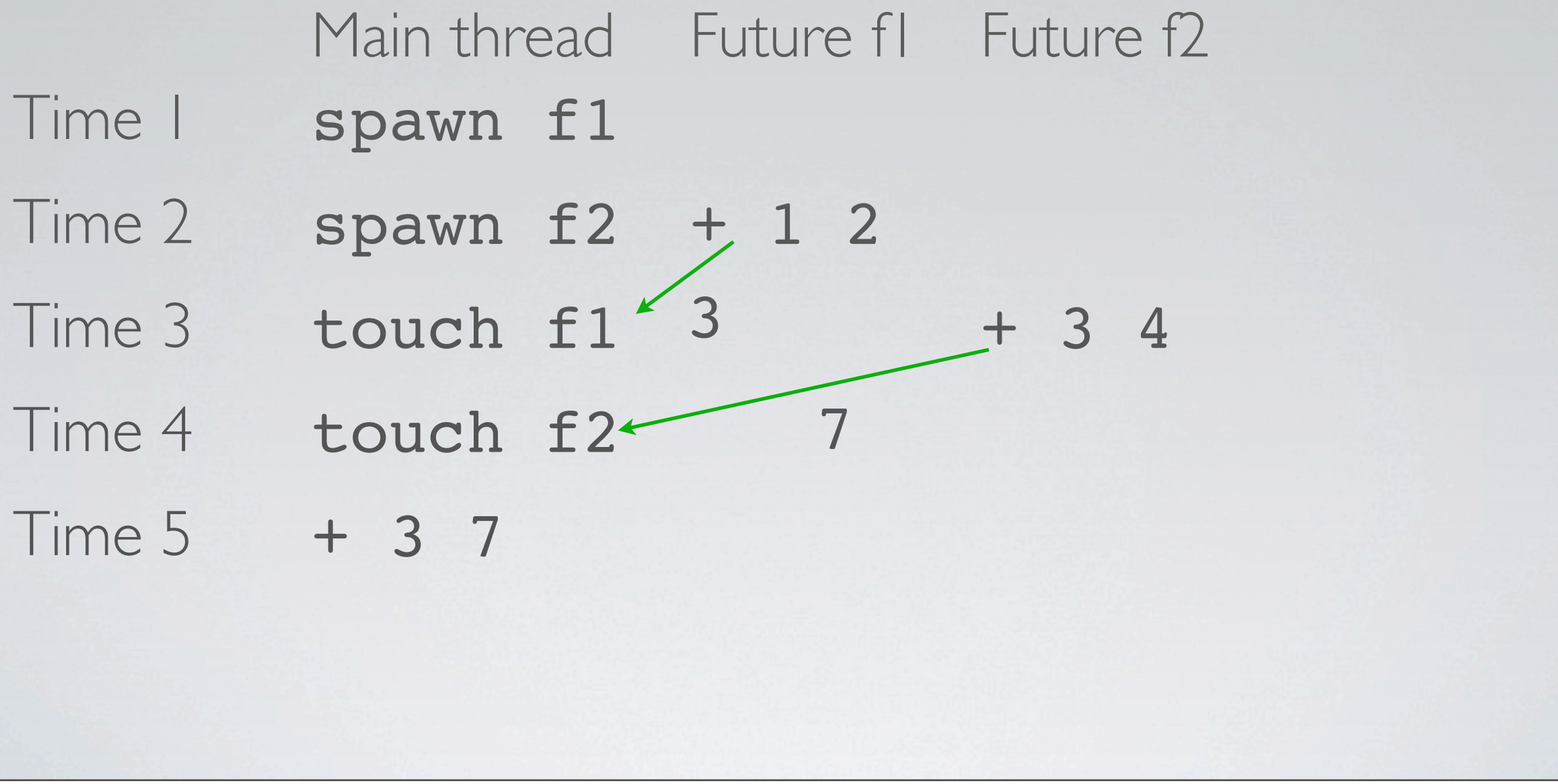
- One **runtime thread** where everything is safe
- Barricades active on all other threads:
  - Detect and intercept unsafe ops
  - Halt a thread until unsafe op can be completed by runtime thread
- Add primitives allowing programmer to explicitly donate the runtime thread's time to the barricaded thread, allowing it to pass through and continue

Racket Language Extension

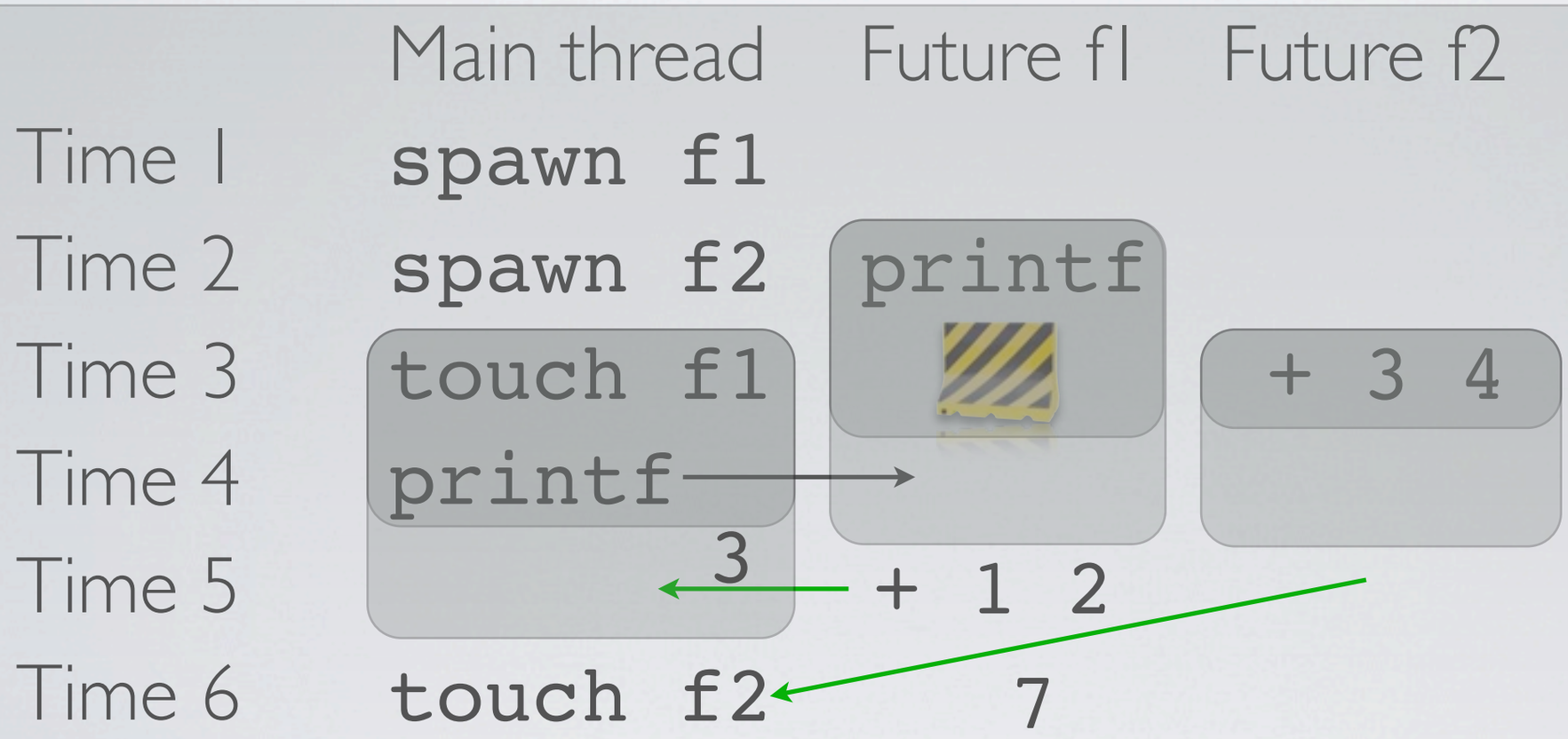
```
future : (-> α) -> α future

touch  : α future -> α
```

```
(let ([f1 (future
           (λ () (+ 1 2)))]
      [f2 (future
           (λ () (+ 3 4)))])
  (+ (touch f1) (touch f2)))
```



```
(let ([f1 (future
           (λ ()
             (printf "Hello!\n")
             (+ 1 2)))]
      [f2 (future (λ () (+ 3 4)))])
  (+ (touch f1) (touch f2)))
```



Racket Implementation

- Racket runtime:
  - Substrate for the Racket language
  - 100,000+ lines of C code
  - Simple, eager JIT compiler
  - Global data includes:
    - Execution state (exception handlers)
    - Symbol table
    - Macro expansion caches
    - GC metadata

Racket Operations

Safe	Unsafe	Other
+	+	allocation
/	/	JIT compilation
unsafe-fl+	hash-set!	
unsafe-fl/	printf	
unsafe-vector-ref	vector-ref	
unsafe-vector-set!	printf	
	call/cc	
	write	
	read	
	open-input-file	
	error	

Barricades in Racket

- All code JIT compiled (if possible)
- Fast-path ops - inlined
- Slow-path ops - C functions

"Other" Operations

- We leverage Racket's user-level thread infrastructure for:
  - Allocation
  - JIT compilation
- Racket threads: preemptive to programmers, cooperative to runtime
- Cooperation points allow for polling

Garbage Collection

- GC = special form of synchronized operation (stop the world)

- Cooperation points become barriers

Slow-Path Barricading

- ☑ Incremental
- ☑ Seq. performance intuition carries over
- Low development investment
- Good scaling (negligible sequential overhead)

Development Person-Hours (Racket)

- Performed by non-expert (no prior knowledge) and runtime developer

Expert	Non-Expert (me)	Total
41	536	577

Parrot Implementation

- Parrot runtime:
  - Register-based virtual machine
  - Pluggable runloop allows switching between interpreters
  - Dynamic (virtual functions)
- Each bytecode is checked prior to execution for safety
  - Includes argument checking

Development Person-Hours (Racket)

- Performed by expert (active runtime implementation contributor)

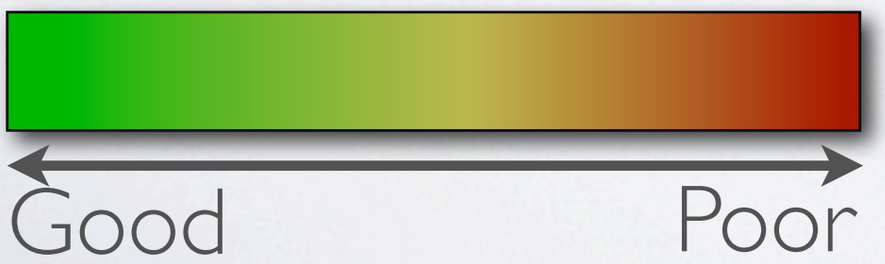
Expert	Non-Expert	Total
52	-	52

Performance Evaluation

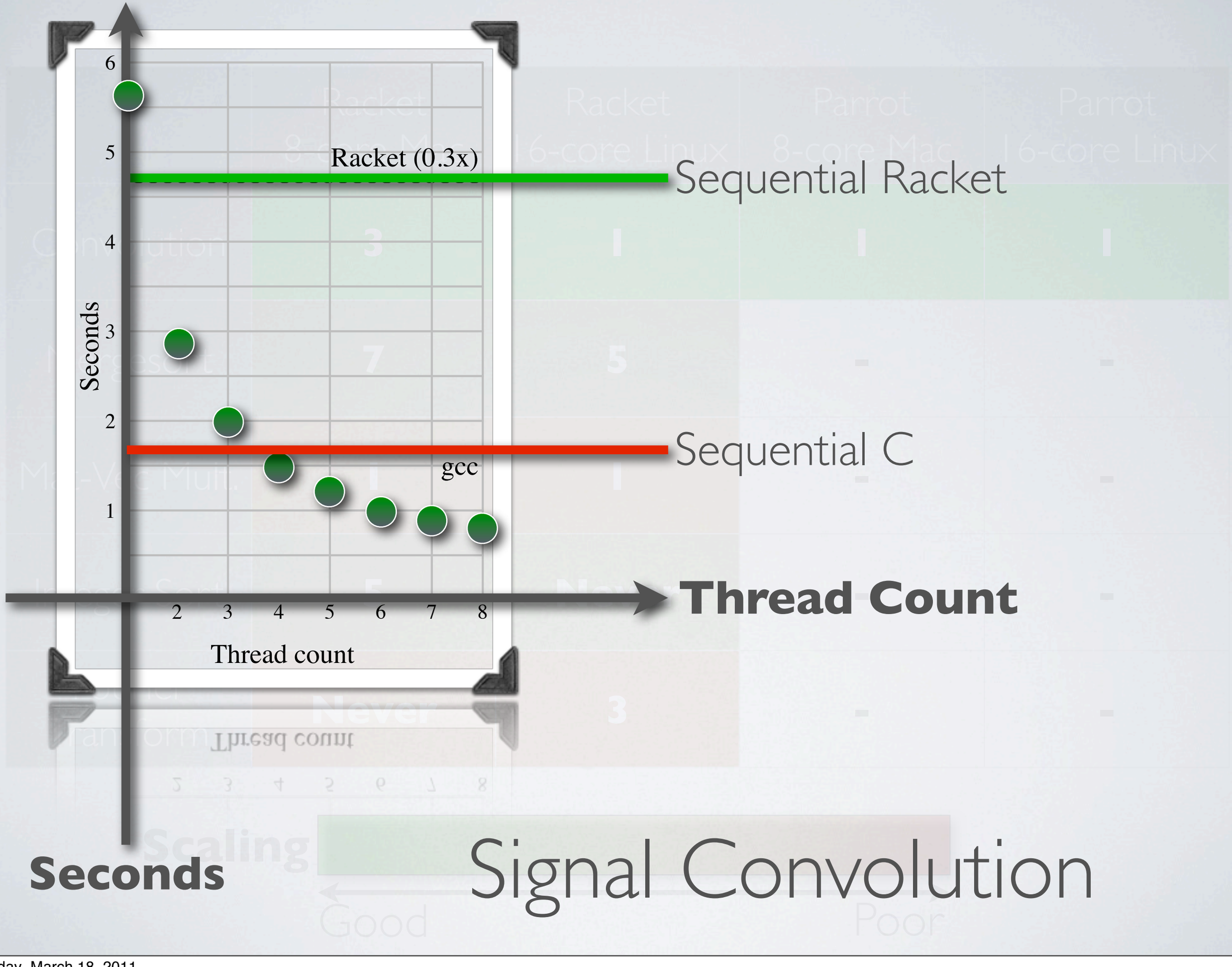
- 3 microbenchmarks
  - Signal convolution
  - Mergesort
  - Sparse matrix-vector multiplication
- 2 NAS Parallel Benchmarks kernels
  - Integer Sort
  - Fourier Transform
- 2 test machines:
  - 8-core workstation (Mac OS X)
  - 16-core mid-range server (Linux)

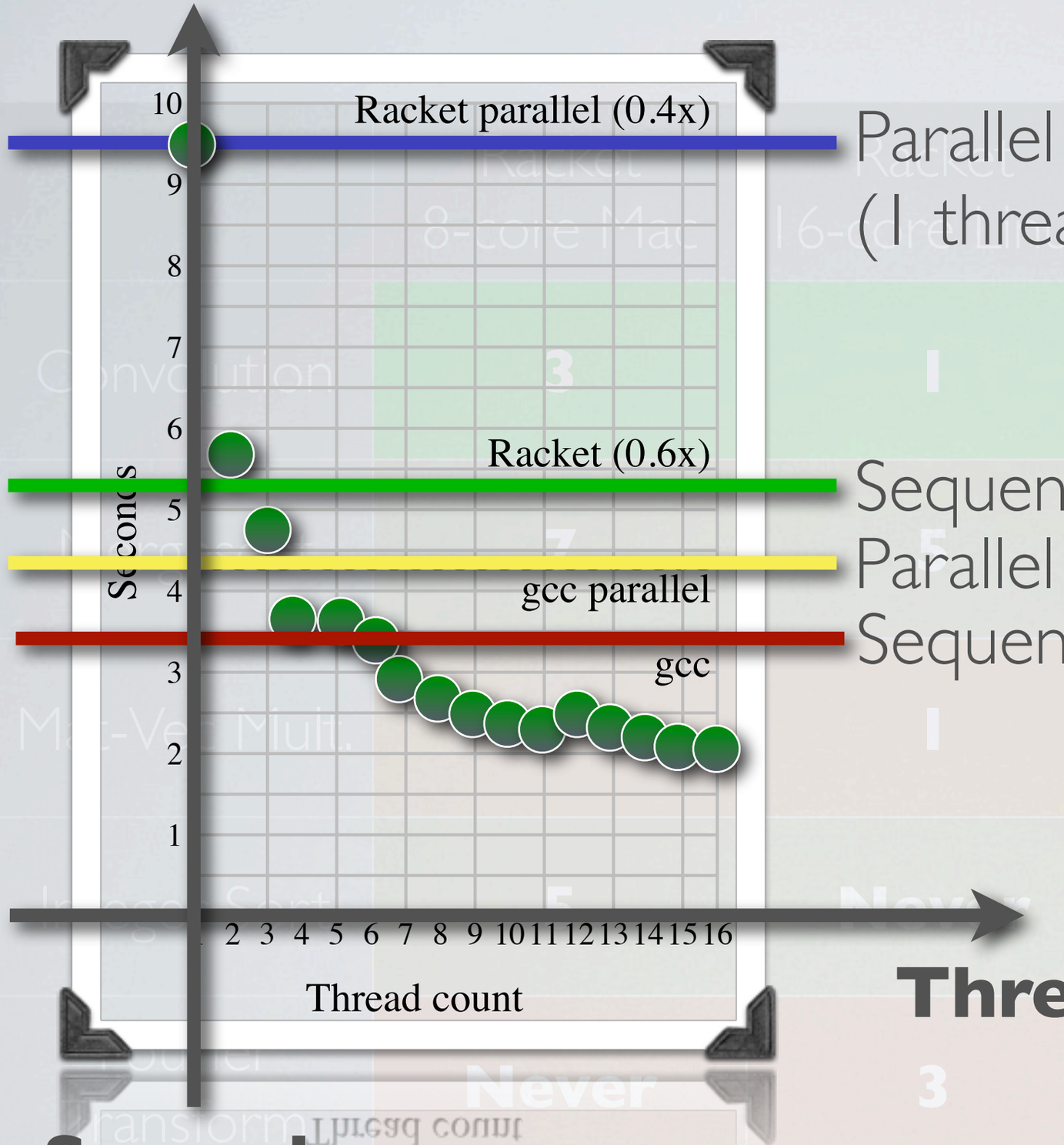
	Racket 8-core Mac	Racket 16-core Linux	Parrot 8-core Mac	Parrot 16-core Linux
Convolution	**4**	**2**	**2**	**2**
Mergesort	**8**	**6**	-	-
Mat-Vec Mult.	**2**	**2**	-	-
Integer Sort	**6**	**Never**	-	-
Fourier Transform	**Never**	**4**	-	-

Scaling



Values = # of threads to beat sequential impl.


6
5
4
3
2
1
Seconds
Racket (0.3x)
gcc
2
3
4
5
6
7
8
Thread count
Sequential Racket
Sequential C
Thread Count
Seconds
Signal Convolution



Parallel Racket (1 thread)

Sequential Racket

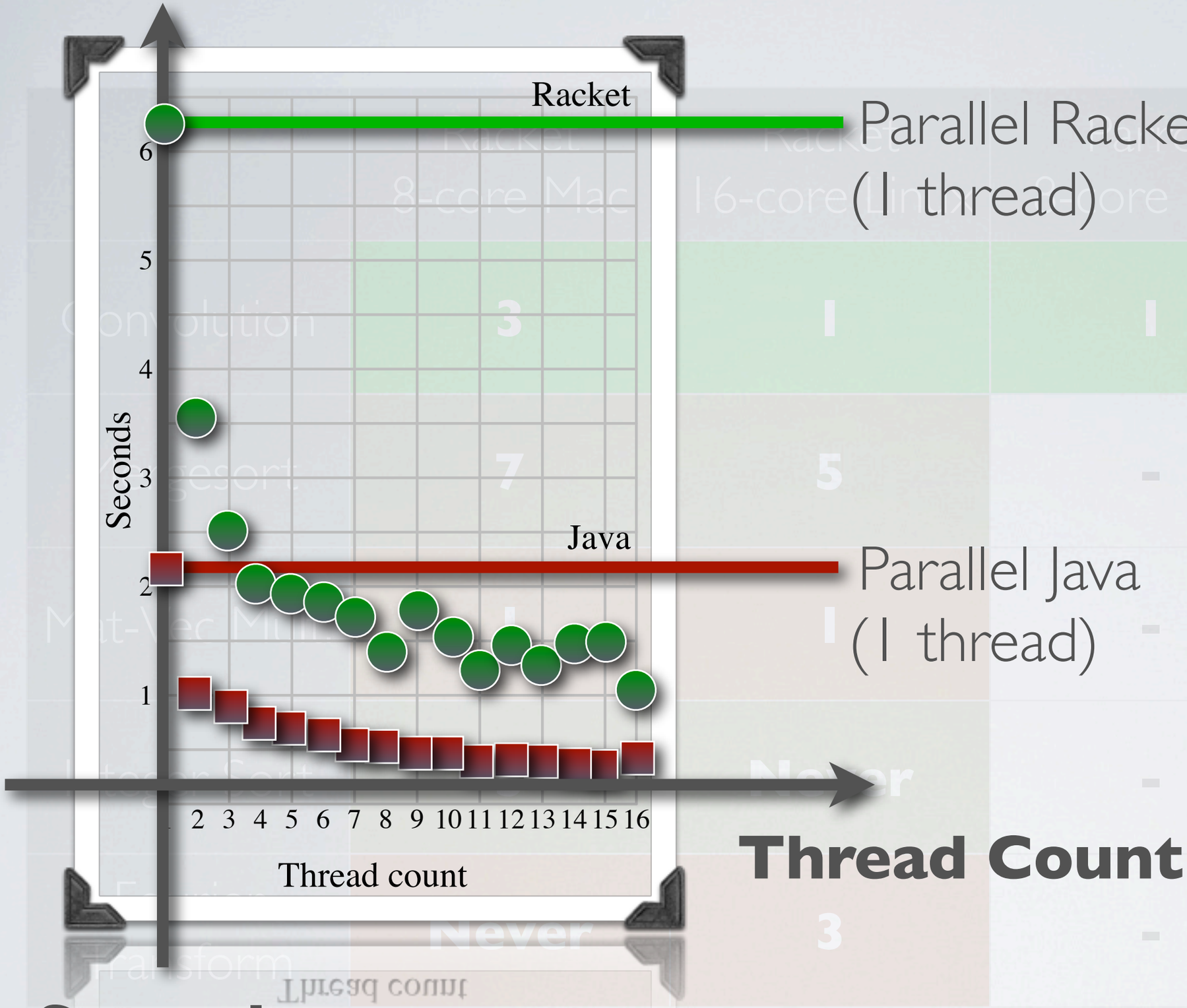
Parallel C

Sequential C

Thread Count

Seconds

Parallel Mergesort



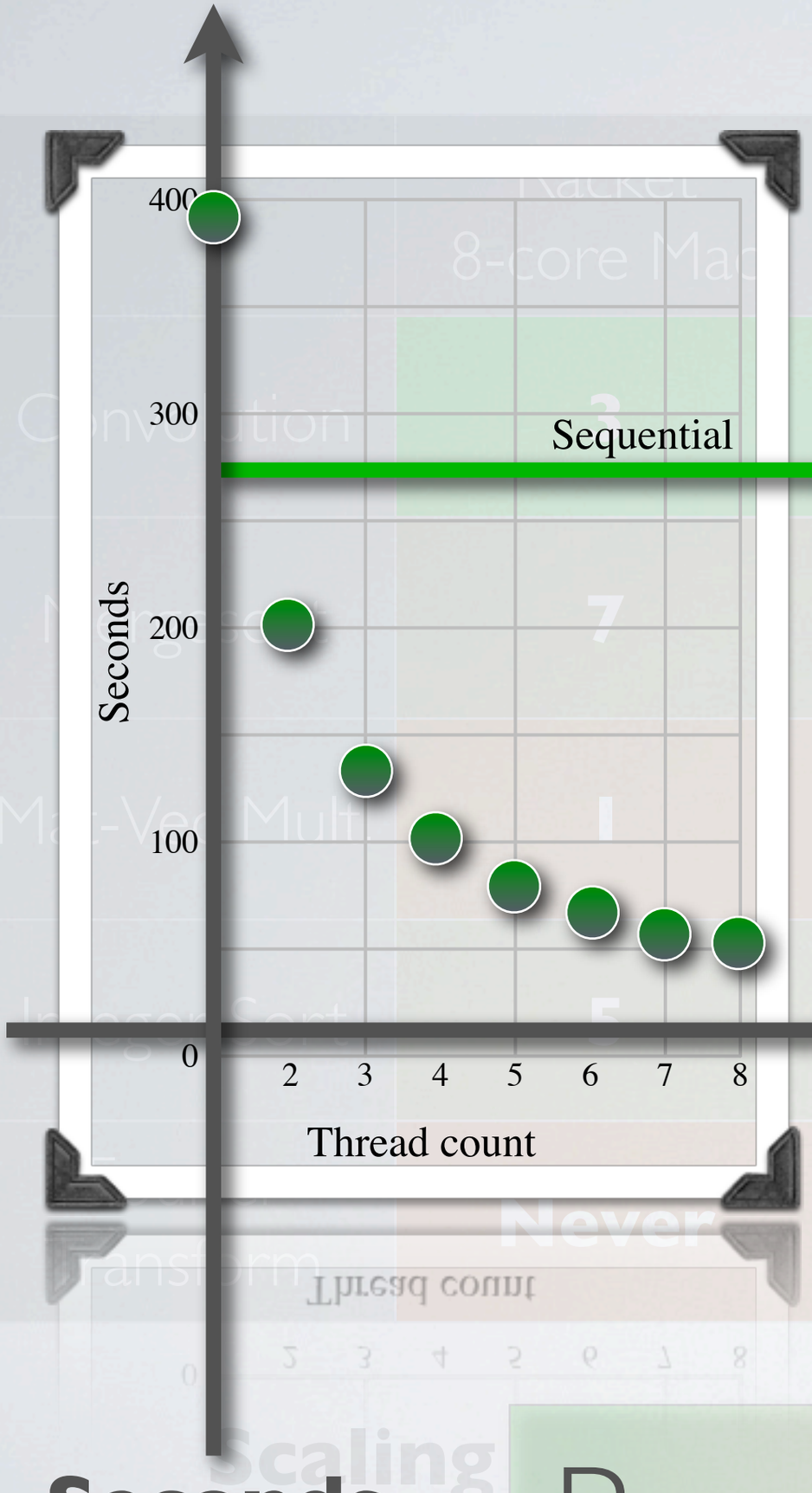
Parallel Racket
(1 thread)

Parallel Java
(1 thread)

Thread Count

Seconds

NAS Fourier Transform



Sequential Parrot

Thread Count

Seconds

Parrot Signal Convolution

Slow-Path Barricading

- ☑ Incremental
- ☑ Seq. performance intuition carries over
- ☑ Low development investment
- ☑ Good scaling (negligible sequential overhead)

Thanks!

- Try parallel Racket today: http://racket-lang.org/download/
- Try slow path barricading in your runtime system; the main system developer should be able to add it within a few weeks of work