Creating Real-Time Linux Applications

Paul E. McKenney
IBM Distinguished Engineer & CTO Linux
Linux Technology Center

Overview

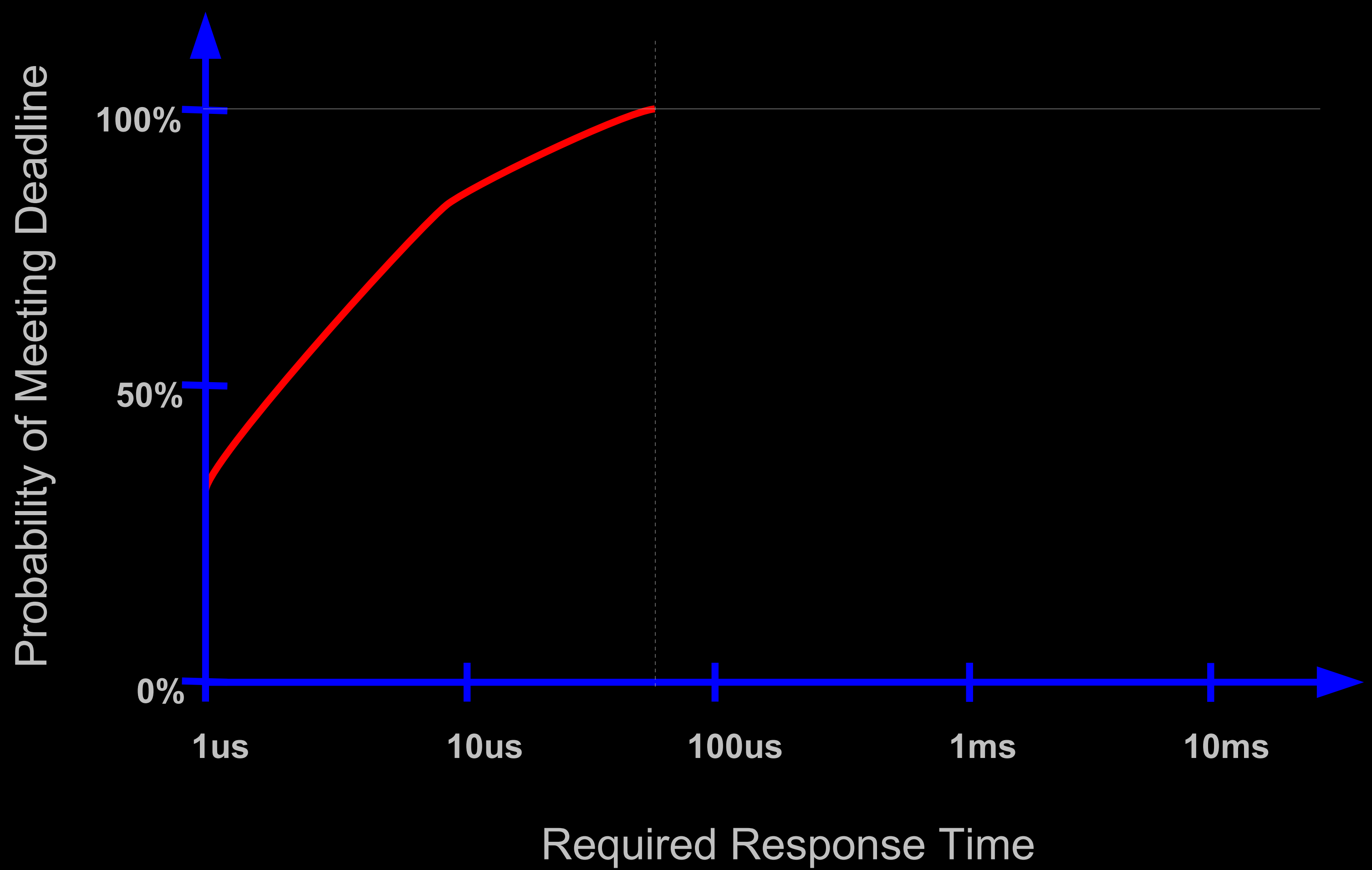
- Introduction to Performance, Scalability, and Real-Time Issues on Modern Multicore Hardware: Is Parallel Programming Hard, And If So, Why?
- Performance and Scalability Technologies in the Linux Kernel
- Creating Performant and Scalable Linux Applications
- Real-Time Technologies in the Linux Kernel
- Creating Real-Time Linux Applications

Overview

- What Is “Real Time”?
- What Real-Time Applications?
- Example Real-Time Application
- Example Real-Fast Application
- Real Time vs. Real Fast: How to Choose
- Course Summary

What is “Real Time”?

Hard or Soft Real Time?



Definitions of Hard Real Time

1. A Hard Real-Time System Will **Always** Meet its Deadlines
2. A Hard Real-Time System Will Either (1) Meet its Deadlines, or (2) Give Timely Failure Indication
3. A Hard Real-Time System Will Always Meet its Deadlines (in Absence of Hardware Failure)
4. A Hard Real-Time System Will Pass a Specified Test Suite

- Which definition is appropriate? Why, and under what conditions?

Hard Realtime: Problem With Definition #1

- If you have a hard realtime system...
  - I have a hammer that will make it miss its deadlines!

Hard Realtime: Problem With Definition #2

- I have a “hard realtime” system
  - It simply always fails!

Hard Realtime: Problem With Definition #3

- “Rest assured, sir, that if your life support fails, your death will most certainly not have been due to a software problem!!!”

Hard Realtime: Problem With Definition #4

- This definition can cause purists severe heartburn and cognitive dissonance

- But this definition is what is used in “real life”
- Real systems are too complex to admit first-principles mathematical analysis
  - Perhaps this will change with improved tooling

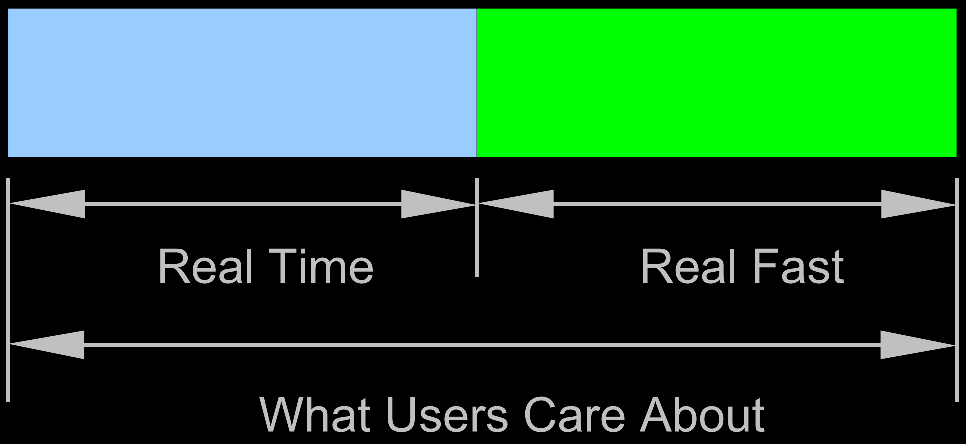
What Does Real-World Real-Time Entail?

- Quality of Service (Beyond "Hard"/"Soft")
  - Services Supported
    - Probability of meeting deadline absent HW failure
    - Deadlines supported
  - Performance/Scalability for RT & non-RT Code
- Amount of Global Knowledge Required
- Fault Isolation
- HW/SW Configurations Supported

- "But Will People Use It?"

Real Time and Real Fast: Useful Definitions

- Real Time
  - OS: "how long before work starts?"
- Real Fast
  - Application: "once started, how quickly is work completed?"
- This Separation Can Result in Confusion!

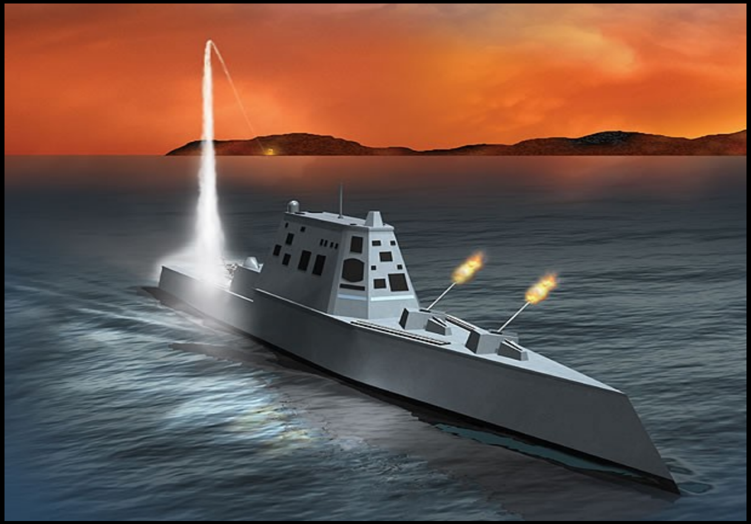


What Real-Time Applications?

What Real-Time Applications?

... In Search of Death ...

In Search of Life ...



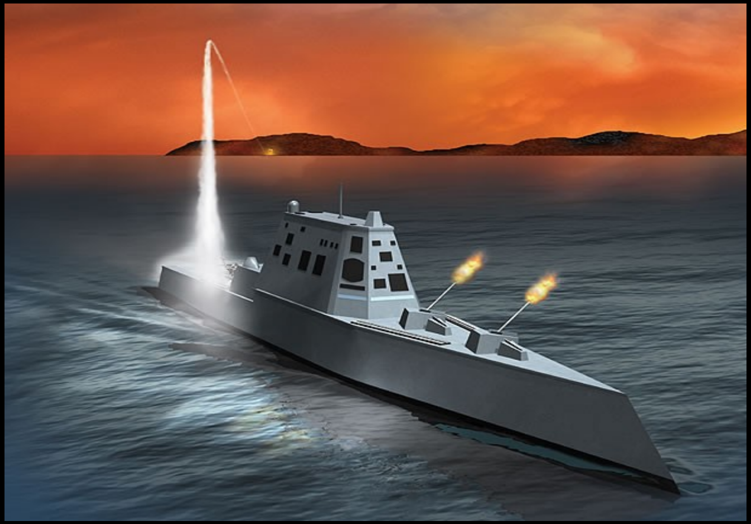
... In Search of Money

What Real-Time Applications?

In Search of Life ...

(Or In Search of Knowledge, If You Prefer.)

... In Search of Death ...



... In Search of Money

What Real-Time Applications?

- Industrial control
- Embedded devices
  - PDAs, cellphones, TVs, refrigerators, cars, ...
- Military
- Scientific
- Financial
- Commercial
  - Break with traditional practice: real-time systems no longer standalone, but tied into enterprise IT systems

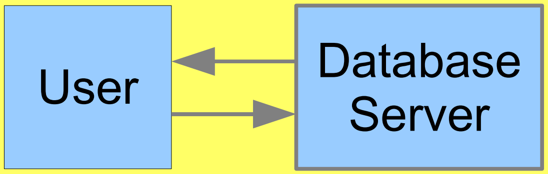
Historical Latency Trends, Revisited

- Traditional response time limits on the order of 1-2 seconds
- In contrast, 100ms is perceived as ideal, 1 second just barely acceptable, and 10 seconds as unacceptable.
  - http://www.bohmann.dk/articles/response_time_still_matters.html
- Improved response times gain business:
  - http://www.akamai.com/en/resources/pdf/casestudy/Akamai_CaseStudy_SKF.pdf
  - http://www.zend.com/products/zend_platform
  - http://www-306.ibm.com/software/tivoli/products/composite-application-mgr-rtt/
- Numerous other products and services to measure/improve web response times
- Improvement from 1 second to 100ms represents an hour per month savings for employees who use the web heavily (one page view per two minutes)
- Gameset generation moving into positions with IT purchasing authority
- This group has grown up with sub-reflex response from computers

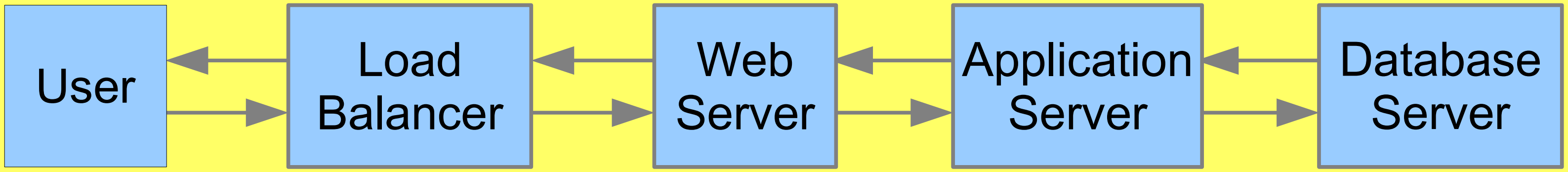
- Endgame: 100ms end-to-end response time
  - translates into smaller per-machine response times!

But Latencies Accumulate

Before the web (late 1980s):



On the web:

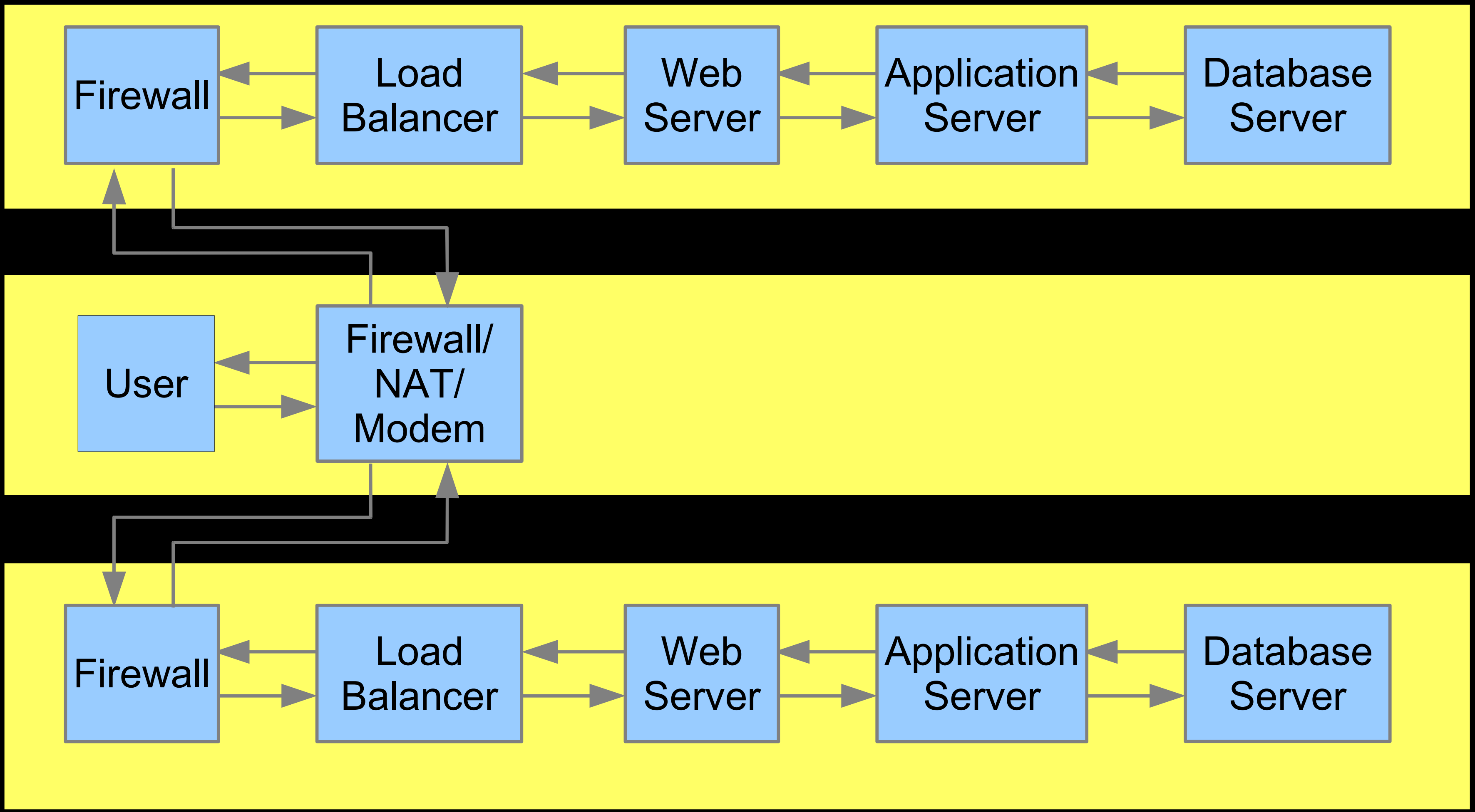


Machines must be seven times faster to give same worst-case overall latency!!!
(Situation less demanding for soft realtime.)

Allocating a Latency Budget for Web Application

- Start with a 200ms budget:
  - Assume we need to meet 200ms 99% of the time (3σ)
    - Based on w3.ibm.com's variability, consumes 24% of budget: leaves 152ms
  - Assume 90 ms for Internet latencies (based on w3.ibm.com again): leaves 62ms
  - Assume 50ms for database to execute transaction: leaves 12ms
  - Spread over 10 machines (not counting database backend): leaves 1.09ms per machine. Some of which are running web-application servers using Java.
- Moving to a 100ms budget:
  - Assume we need to meet 100ms 99% of the time (3σ)
    - Based on w3.ibm.com's variability, consumes 24% of budget: leaves 76ms
  - Assume 90 ms for Internet latencies (based on w3.ibm.com again): puts us 14ms in the red.
- Endgame: whatever can be provided
  - Internet latencies will be the bottleneck – greater emphasis on edge servers
  - Also on private-network bypass for heavy-traffic localities

Latency Accumulation With Web 2.0



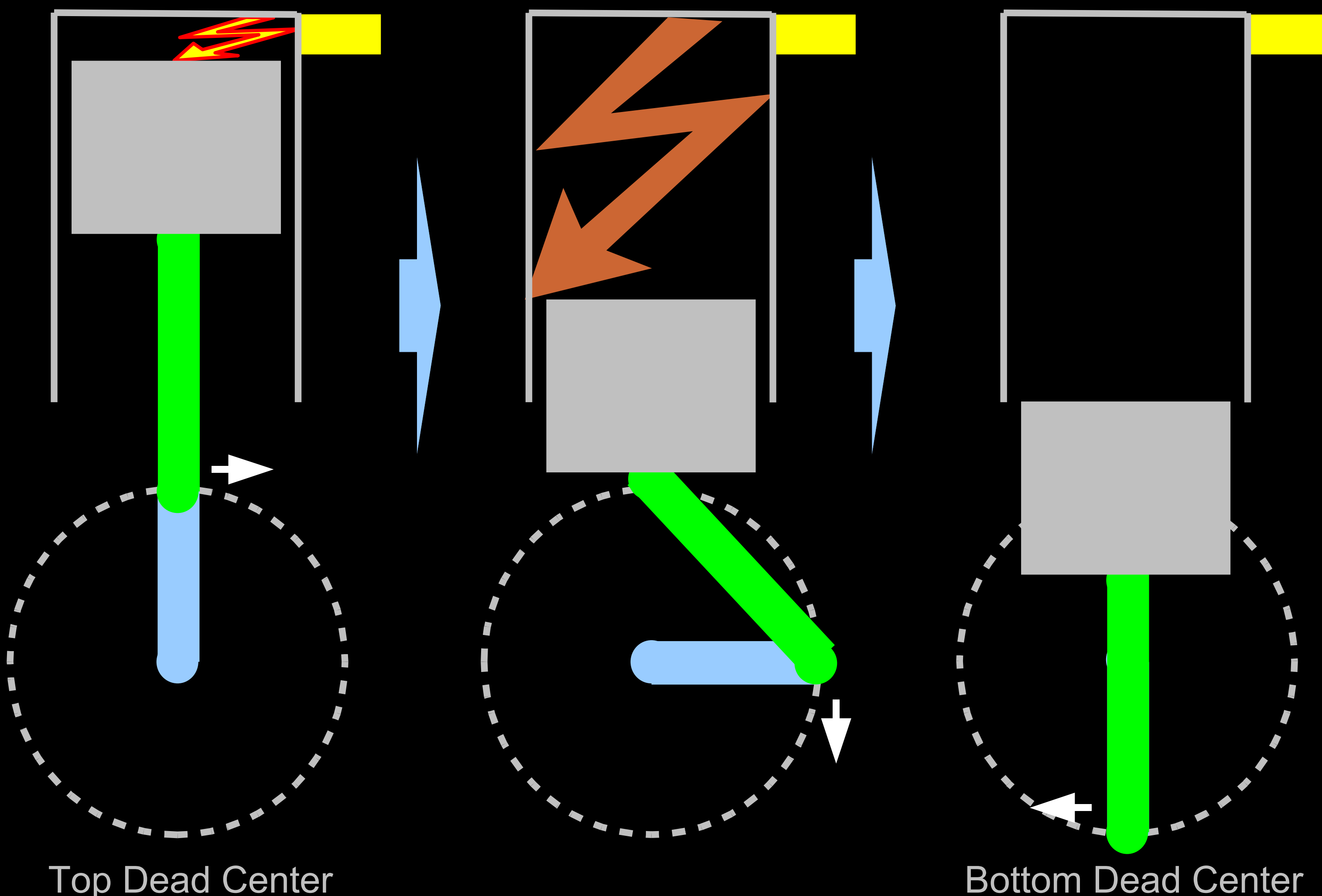
Machines must be more then twenty times faster to give same overall latency!!!

Example Real-Time Application

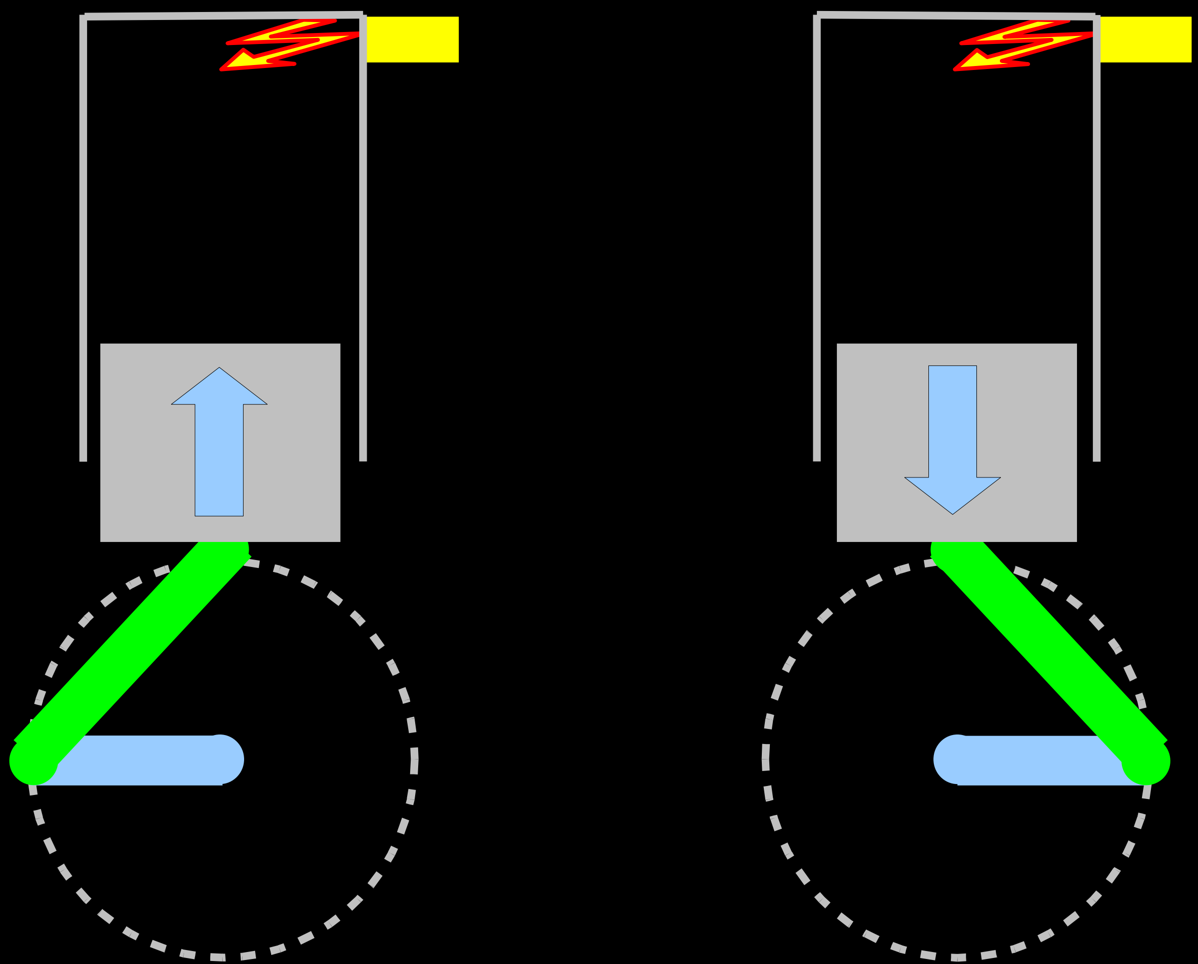
Example Real-Time Application: Fuel Injection

- Mid-sized industrial engine
  - Fuel injection within one degree surrounding “top dead center”
- 1500 RPM rotation rate
  - 1500 RPM / 60 sec/min = 25 RPS
  - 25 RPS * 360 degrees/round = 9000 degrees/second
  - About 111 microseconds per degree
  - Hence need to schedule to within about 100 microseconds

Fuel Injection: Conceptual Operation



Too Early and Too Late Are Bad



Injecting Too Early
(Exaggerated)

Injecting Too Late
(Exaggerated)

Fanciful Code Operating Injectors

```
struct timespec timewait;

angle = crank_position();
timewait.tv_sec = 0;
timewait.tv_nsec = 1000 * 1000 * 1000 * angle / 9000;
nanosleep(&timewait, NULL);
inject();
```

Fuel Injection Test Program

```
if (clock_gettime(CLOCK_REALTIME, &timestart) != 0) {
        perror("clock_gettime 1");
        exit(-1);
}
if (nanosleep(&timewait, NULL) != 0) {
        perror("nanosleep");
        exit(-1);
}
if (clock_gettime(CLOCK_REALTIME, &timeend) != 0) {
        perror("clock_gettime 2");
        exit(-1);
}
```

Bad results, even on -rt kernel build!!! Why?

Test Program Needs MONOTONIC

```
if (clock_gettime(CLOCK_MONOTONIC, &timestart) != 0) {
        perror("clock_gettime 1");
        exit(-1);
}
if (nanosleep(&timewait, NULL) != 0) {
        perror("nanosleep");
        exit(-1);
}
if (clock_gettime(CLOCK_MONOTONIC, &timeend) != 0) {
        perror("clock_gettime 2");
        exit(-1);
}
```

Still bad results, even on -rt kernel build!!! Why?

Test Program Needs RT Priority

```
struct sched_param sp;

sp.sched_priority = sched_get_priority_max(SCHED_FIFO);
if (sp.sched_priority == -1) {
        perror("sched_get_priority_max");
        exit(-1);
}
if (sched_setscheduler(0, SCHED_FIFO, &sp) != 0) {
        perror("sched_setscheduler");
        exit(-1);
}
```

Still sometimes bad results, even on -rt kernel build!!! Why?

Test Program Needs mlockall()

```
if (mlockall(MCL_CURRENT | MCL_FUTURE) != 0) {
        perror("mlockall");
        exit(-1);
}
```

Better results on -rt kernel: nanosleep jitter < 20us, 99.999% < 13us
(4-CPU 2.2GHz x86 system with RT firmware – your mileage will vary)

More than 3 *milliseconds* on non-realtime kernel!!!
(Though improved on more recent kernels with high-resolution timers.)

Fuel Injection: Further Tuning Possible

- On multicore systems:
  - Affinity time-critical tasks onto private CPU
    - (Can often safely share with non-realtime tasks)
  - Affinity IRQ handlers away from time-critical tasks
- Carefully select hardware and drivers
- Carefully select kernel configuration
  - Depends on hardware in some cases

Example Real-Fast Application

Bring RT Magic to Non-Real-Time Application

```
tar -xjf linux-2.6.24.tar.bz2
cd linux-2.6.24
make allyesconfig > /dev/null
time make -j8 > Make.out 2>&1
cd ..
rm -rf linux-2.6.24
```

Kernel Build: Performance Results

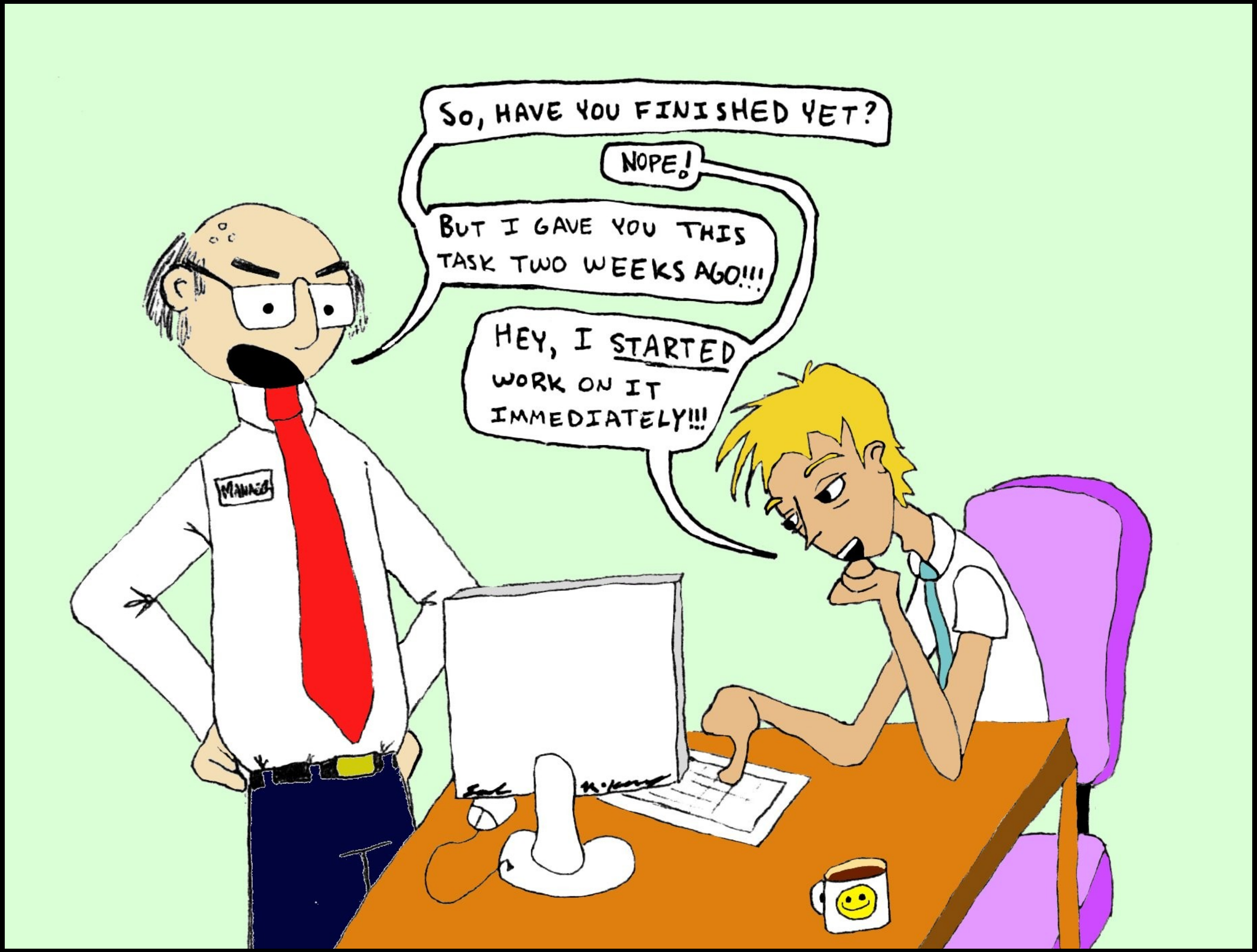
		Real Fast(s)	Real Time (s)	Speedup
real	Average	1332.6	1556.2	0.86
	Std. Dev.	14.6	22.4	
user	Average	3012.2	2964.7	1.02
	Std. Dev.	12.7	17.5	
sys	Average	316.6	657	0.48
	Std. Dev.	1.4	9.2	

Smaller is better, real-time kernel ***not*** helping...

Real Time vs. Real Fast: Throughput Comparison

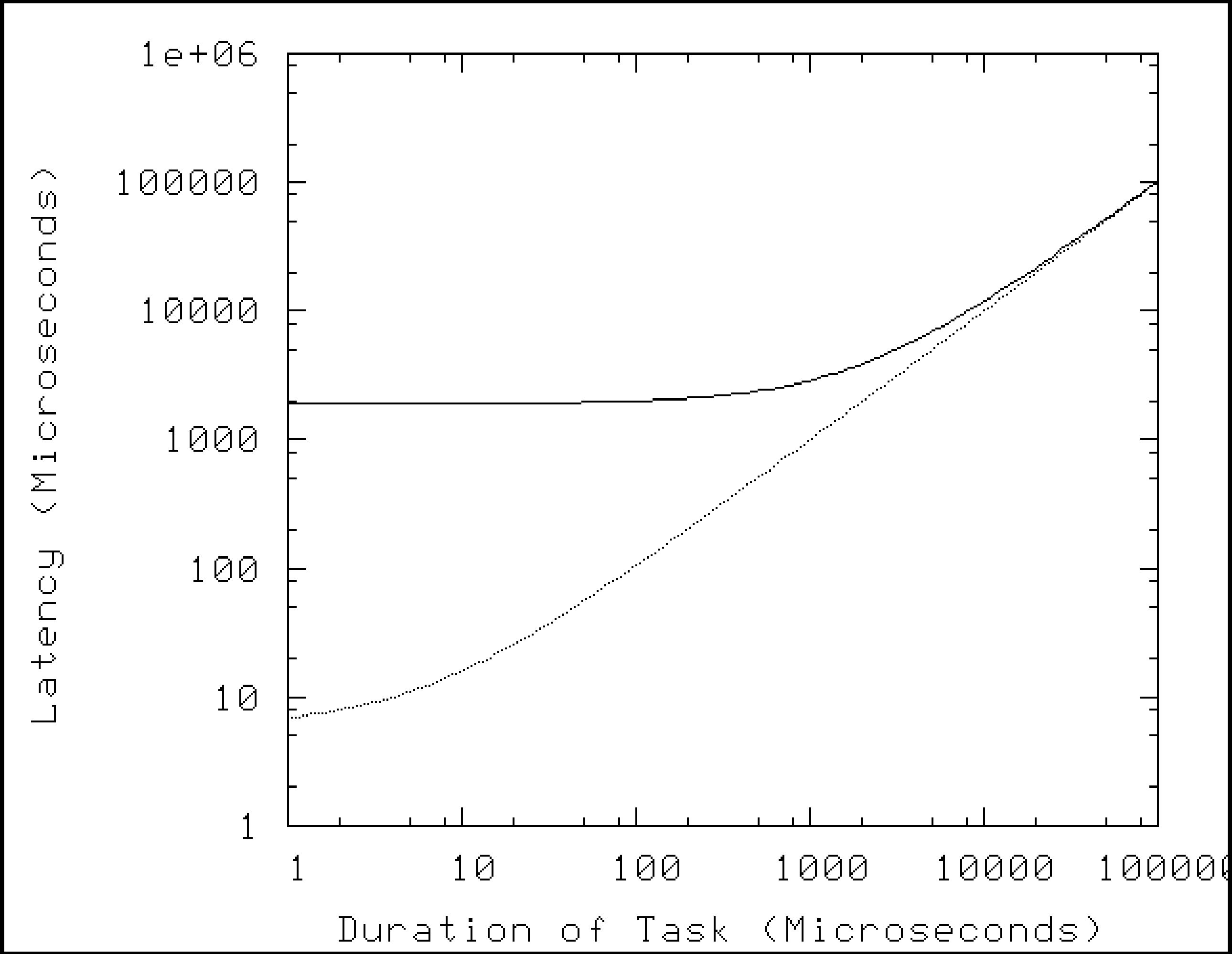
- Real-time system starts more quickly
  - Proverbial hare
- Real-fast system has opportunity to catch up
  - Proverbial tortoise
- Tradeoff based on task duration

The Dark Side of Real Time



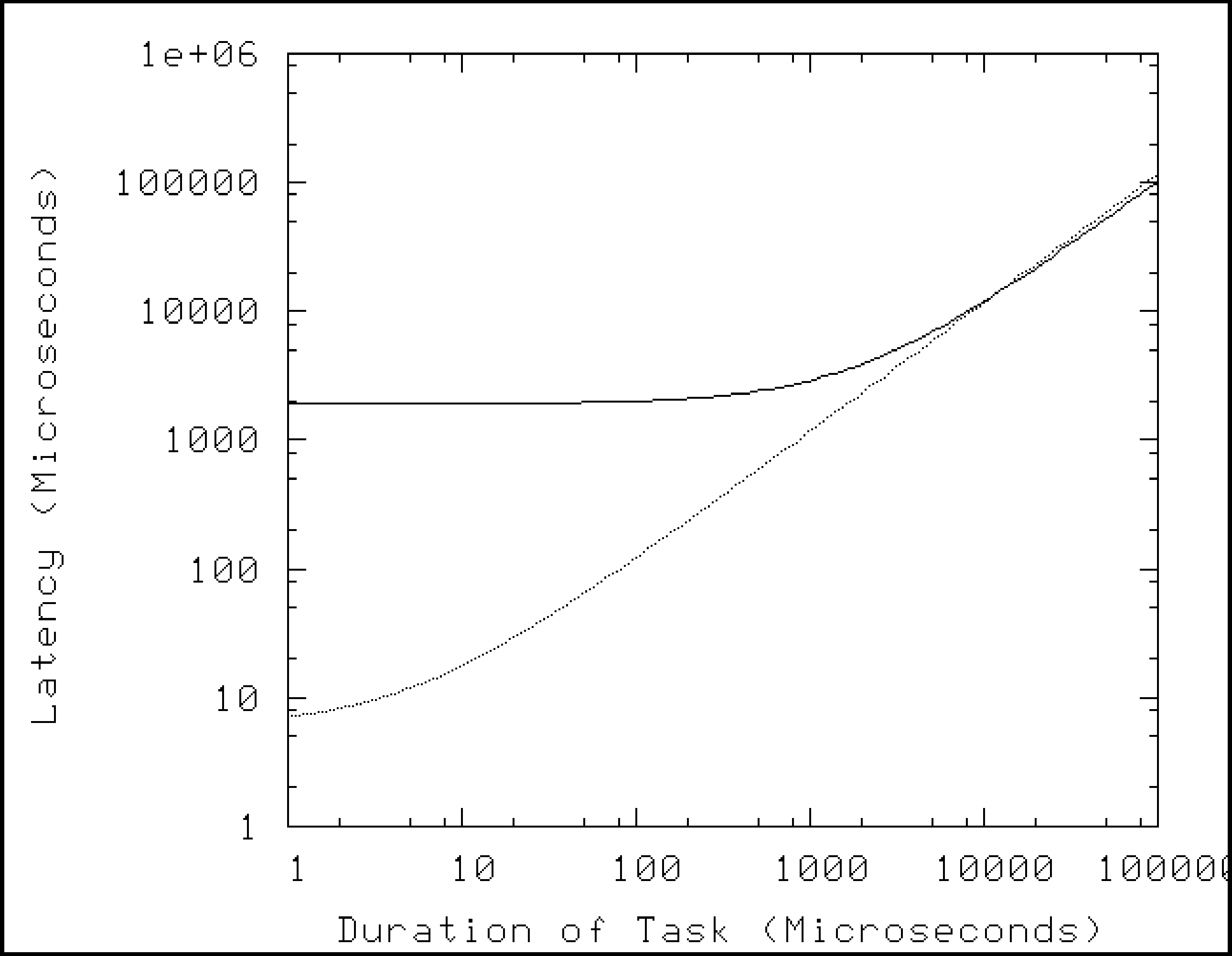
The Dark Side of Real Fast

Real Time vs. Real Fast Throughput: No Penalty



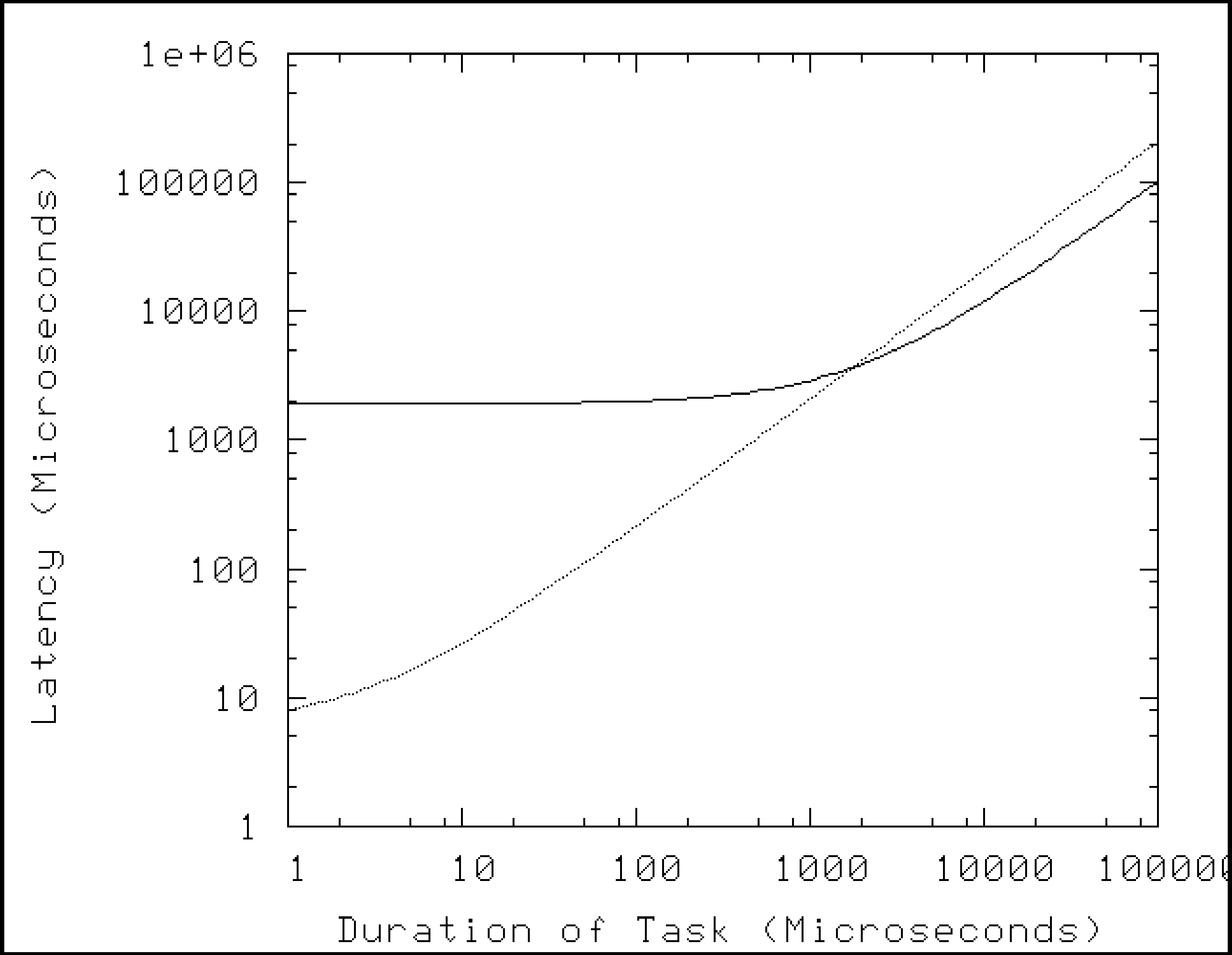
For example, heavy floating-point workloads

Real Time vs. Real Fast Throughput: "real" Penalty



Mixed workloads

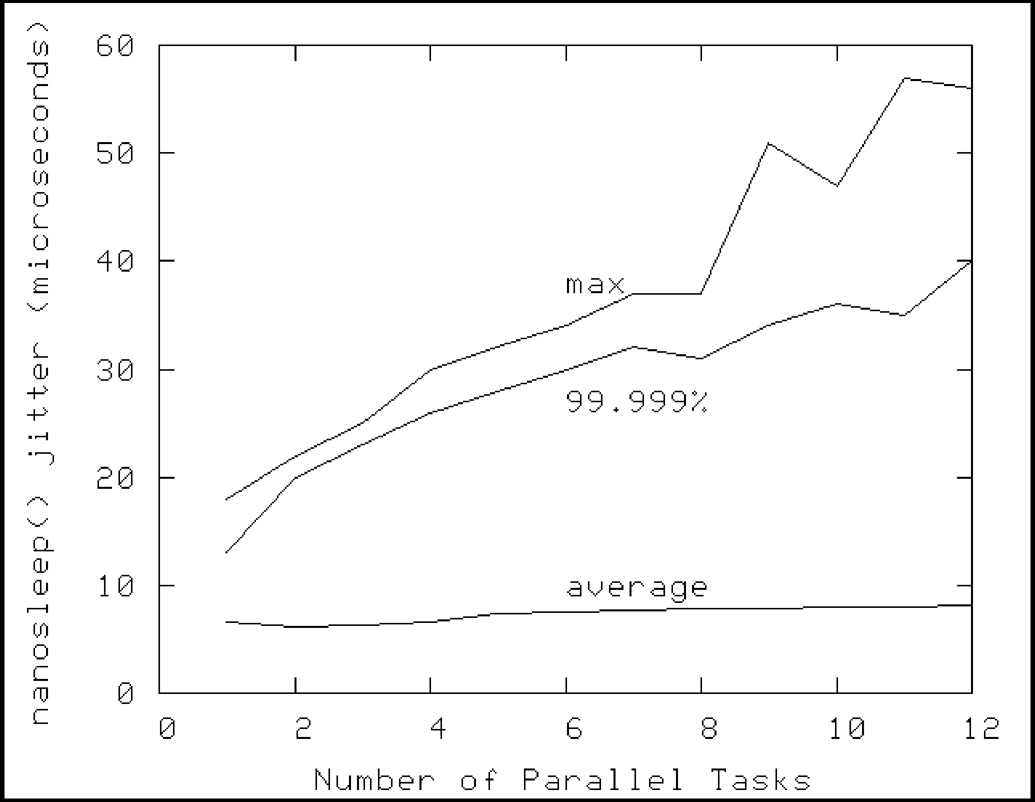
Real Time vs. Real Fast Throughput: “sys” Penalty



Filesystem I/O workloads: “don't do that”

Real-Time Latency vs. CPU Utilization

- CPU Utilization by Real-Time Tasks
  - Can be avoided by time-slotting
  - Which happens naturally in piston engines

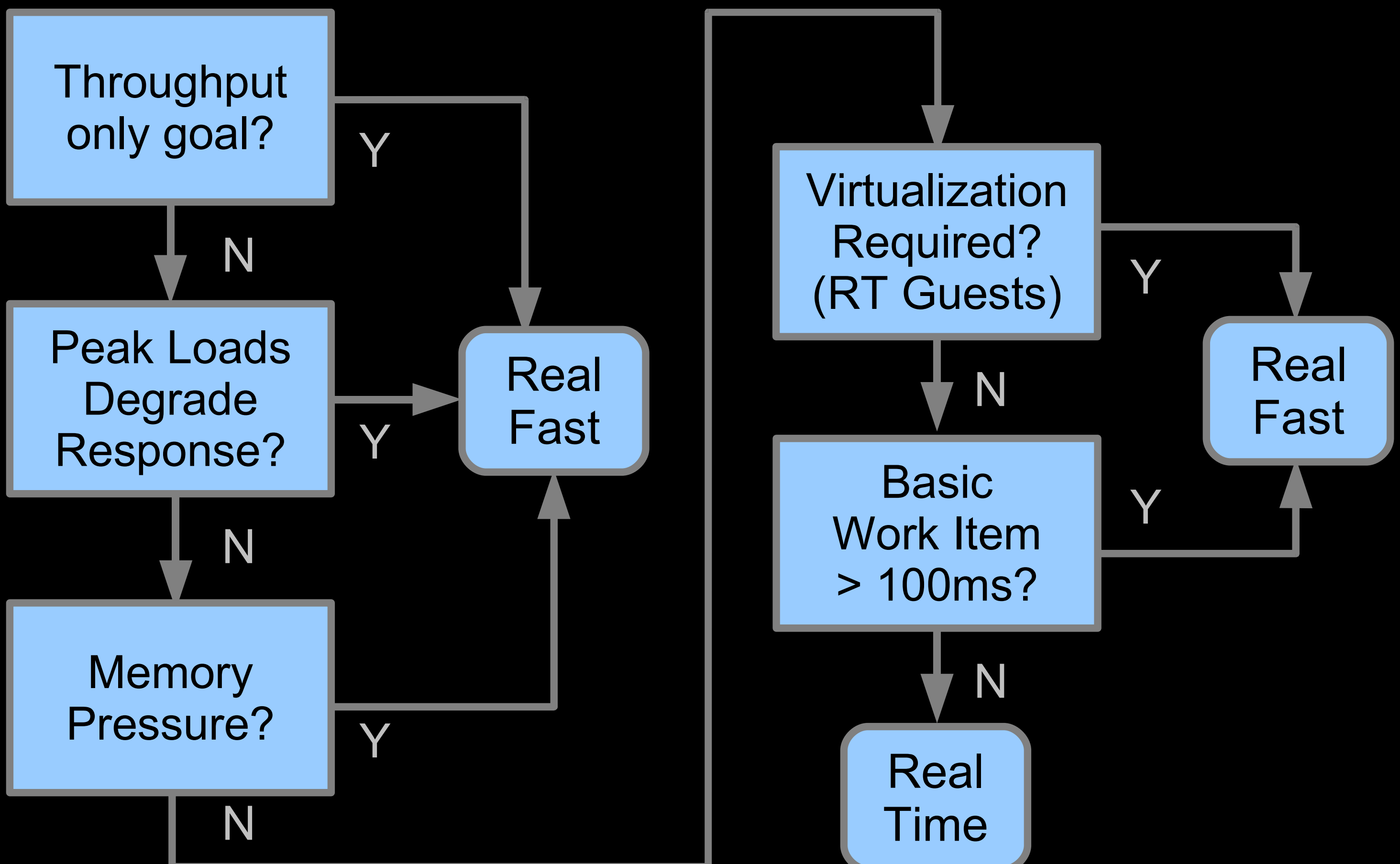


Sources of Real-Time Overhead

- Memory utilization due to mlockall()
- Increased locking overhead
  - Context switches, priority inheritance, preemptable RCU
- Increased irq overhead
  - Threaded irqs (context switches)
  - Added delay (and interactions with rotating mass storage)
- Increased real-time scheduling overhead
  - Global distribution of high-priority real-time tasks
- High-resolution timers

Real Time vs. Real Fast: How to Choose

Real Time vs. Real Fast: How to Choose

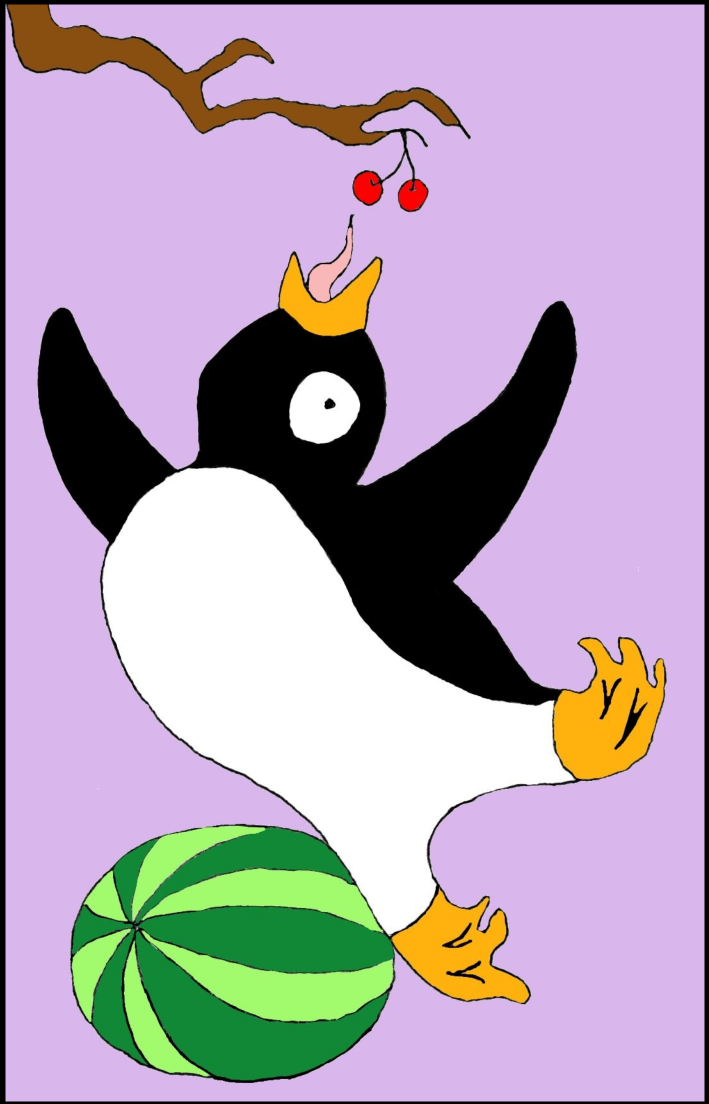


Longer Term: Avoid the Need to Choose

- Reduce Overhead of Real-Time Linux!
  - Easy to say, but...
  - Reduce locking overhead (adaptive locks)
  - Reduce scheduler overhead (ongoing work)
  - Optimize threaded irq handlers
  - Eliminate networking reader-writer-lock bottlenecks (ongoing work)
  - And my "evil plan" from yesterday
- Note that partial progress is beneficial
  - Reduces the need to choose
  - Harvest the low-hanging fruit

Low-Hanging Fruit

Harvest it.
Don't trip over it!



Acknowledgments

- Ingo Molnar
- Thomas Gleixner
- Sven Deitrich
- K. R. Foley
- Gene Heskett
- Bill Huey
- Esben Neilsen
- Nick Piggin
- Steve Rostedt
- Michal Schmidt
- Daniel Walker
- Karsten Wiese
- Gregory Haskins

- And many many more...

Legal Statement

- This work represents the view of the author and does not necessarily represent the view of IBM.
- IBM and IBM (logo) are trademarks or registered trademarks of International Business Machines Corporation in the United States and/or other countries.
- Linux is a registered trademark of Linus Torvalds.
- Other company, product, and service names may be trademarks or service marks of others.

Questions?

To probe further:

- Applications:
  - http://www.cotsjournalonline.com/pdfs/2003/07/COTS07_softside.pdf (In search of death)
  - http://www.nytimes.com/2006/12/11/technology/11reuters.html?ei=5088&en=e5e9416415a9eeb2&ex=1323493200... (In search of money)
  - http://www.linuxjournal.com/article/9361 (Enterprise real-time)
  - http://www.b-eye-network.de/view-articles/3365 (Time value of information)
  - http://searchenterpriselinux.techtarget.com/news/article/0,289142,sid39_gci1309889,00.html (Order of magnitude decrease in response time required over 5 years time)
- Extreme Real Time:
  - “Temporal inventory and real-time synchronization in RTLinuxPro”, Victor Yodaiken, http://www.yodaiken.com/papers/sync.pdf
- Rants:
  - “Against Priority Inheritance”, Victor Yodaiken, http://www.linuxdevices.com/articles/AT7168794919.html
  - “Priority Inheritance: The Real Story”, Doug Locke, http://www.linuxdevices.com/articles/AT5698775833.html
  - “Soft Real Time Continues to Sag”, Victor Yodaiken, http://www.yodaiken.com/w/2006/10/soft-real-time-continues-to-sag/

Course Summary

Course Summary

- Know the hardware
  - Lower-level code required more detailed knowledge
  - Atomics operations, memory barriers, and cache misses are (still) extremely expensive
- “Free” is a very good price
- Don't forget to check for sequential bugs
  - If it is my code, check initialization carefully
- “Hard Real Time” means different things to different people
  - But the customer is always right!!!
- And finally...

Course Summary