Random Testing of Interrupt-Driven Software

John Regehr
University of Utah

Integrated stress testing and debugging

Random interrupt testing

Semantics of interrupts

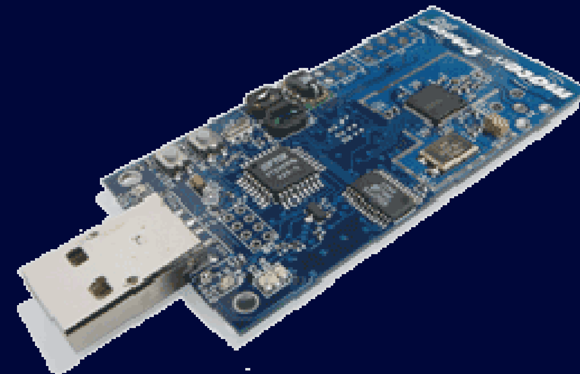
Source-source transformation

Delta debugging

Static stack analysis

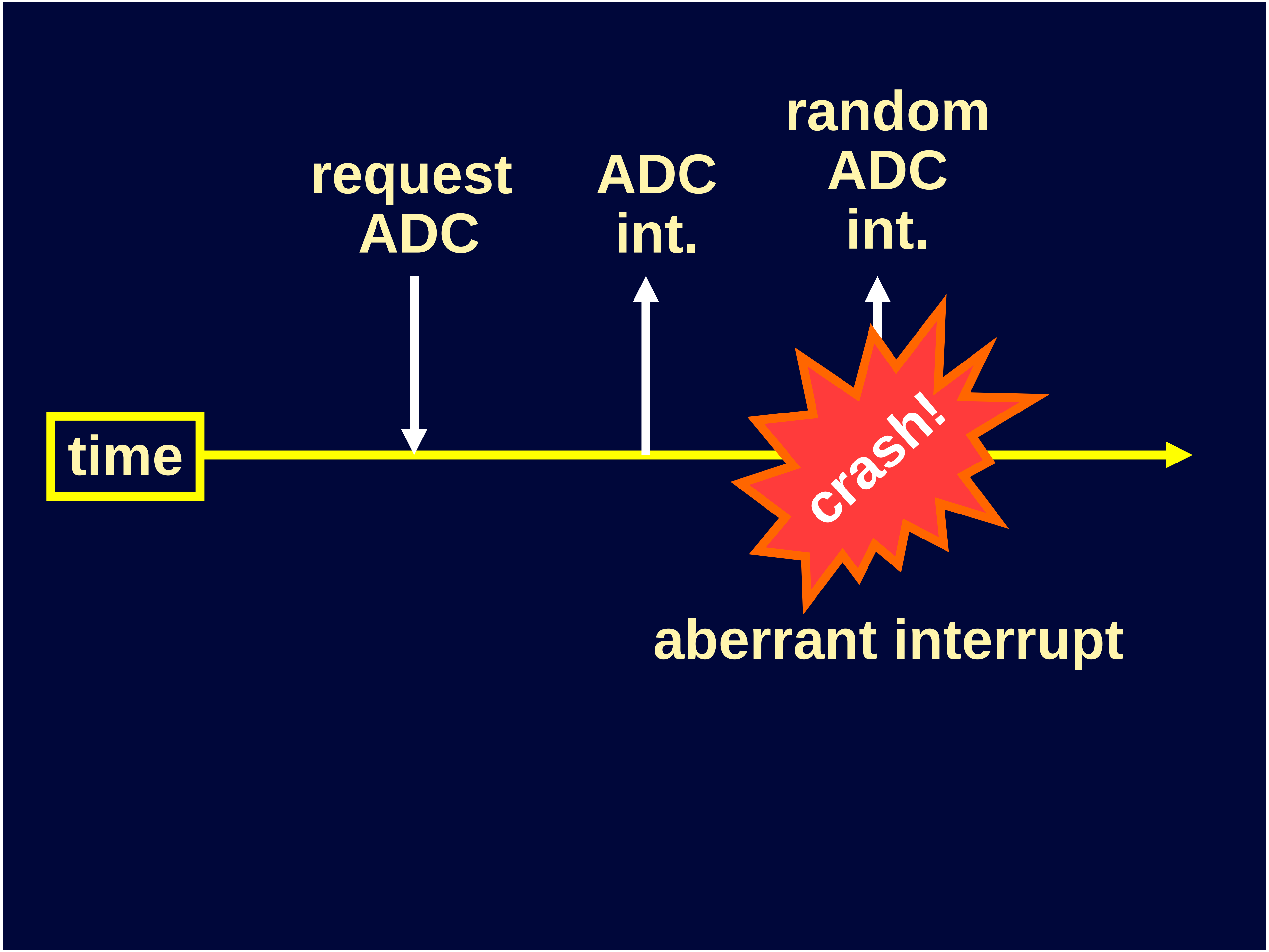
Genetic algorithms

- **Goal: Stress testing and debugging for interrupt-driven embedded software**
- **Why?**
  - **Interrupts hard to get right**
  - **Regular testing typically exercises small part of state space**
  - **Stress testing tends to improve software quality**
  - **Interrupt-driven software used in safety-critical applications**

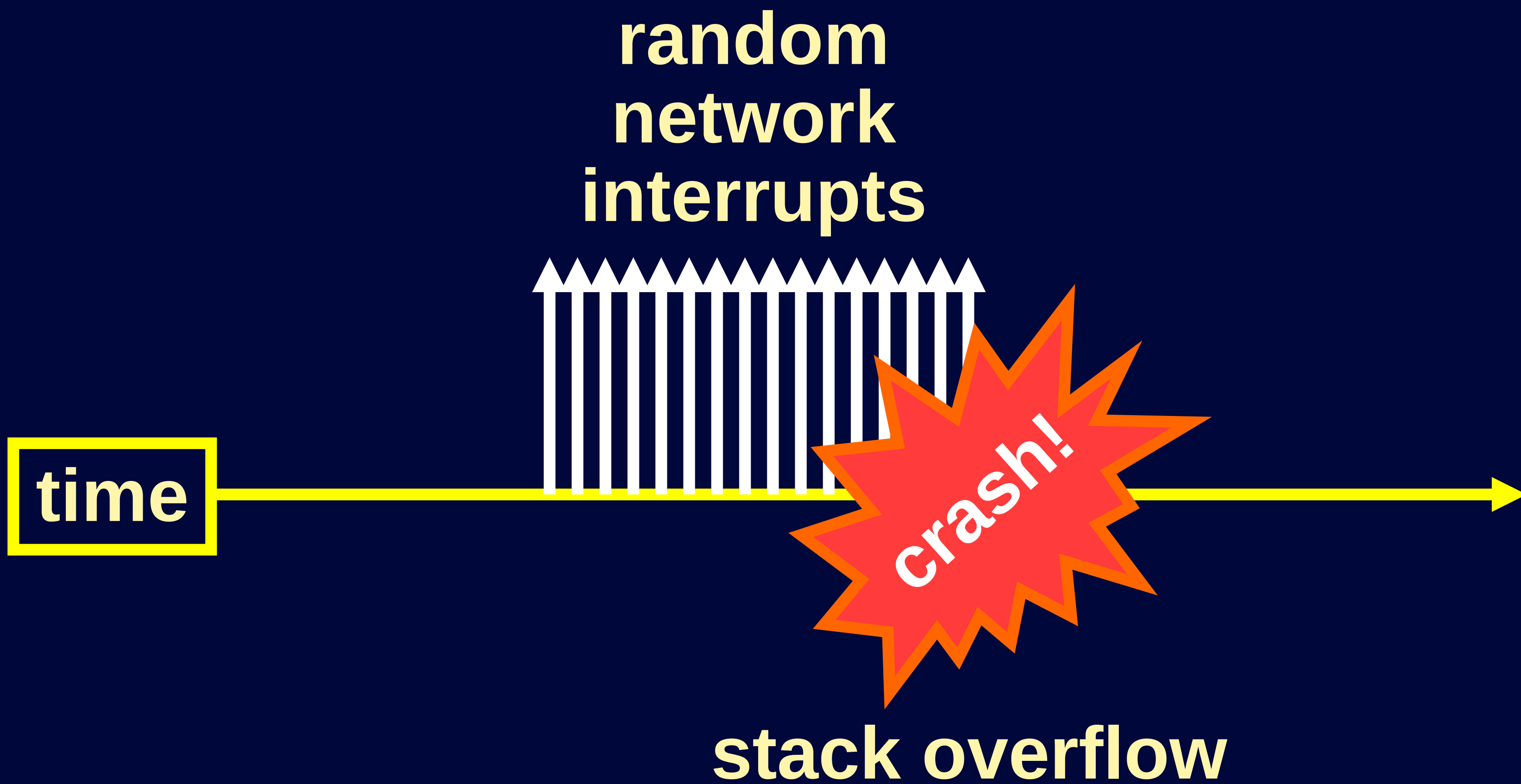


- **Specific case: Sensor network nodes running TinyOS**
  - **Strongly interrupt-driven**
  - **Application code runs in interrupt mode**
  - **Highly resource constrained**
  - **Distributed and opaque – magnifies effects of bugs**

- **Obvious stress testing technique:**
  - **Random interrupt testing – fire interrupts at random times**
- **Potential show stoppers:**
  - **Random interrupts can violate application semantics**
  - **Interrupts can reenter and overflow the stack**



request ADC
ADC int.
random ADC int.
time
crash!
aberrant interrupt



- Many embedded systems permit reentrant interrupts

- **Problem: Interrupts arriving at inconvenient times break applications**
- **Solution: Restrict interrupt arrivals**
- **First classify each interrupt vector**
  - **Requested – arrives in response to an action taken by the system**
  - **Spontaneous – may arrive at any time**

- Restricted Interrupt Discipline (RID):
  - Requested interrupts – only permit when a request is outstanding
  - Spontaneous interrupts – only permit when the interrupt isn't already running

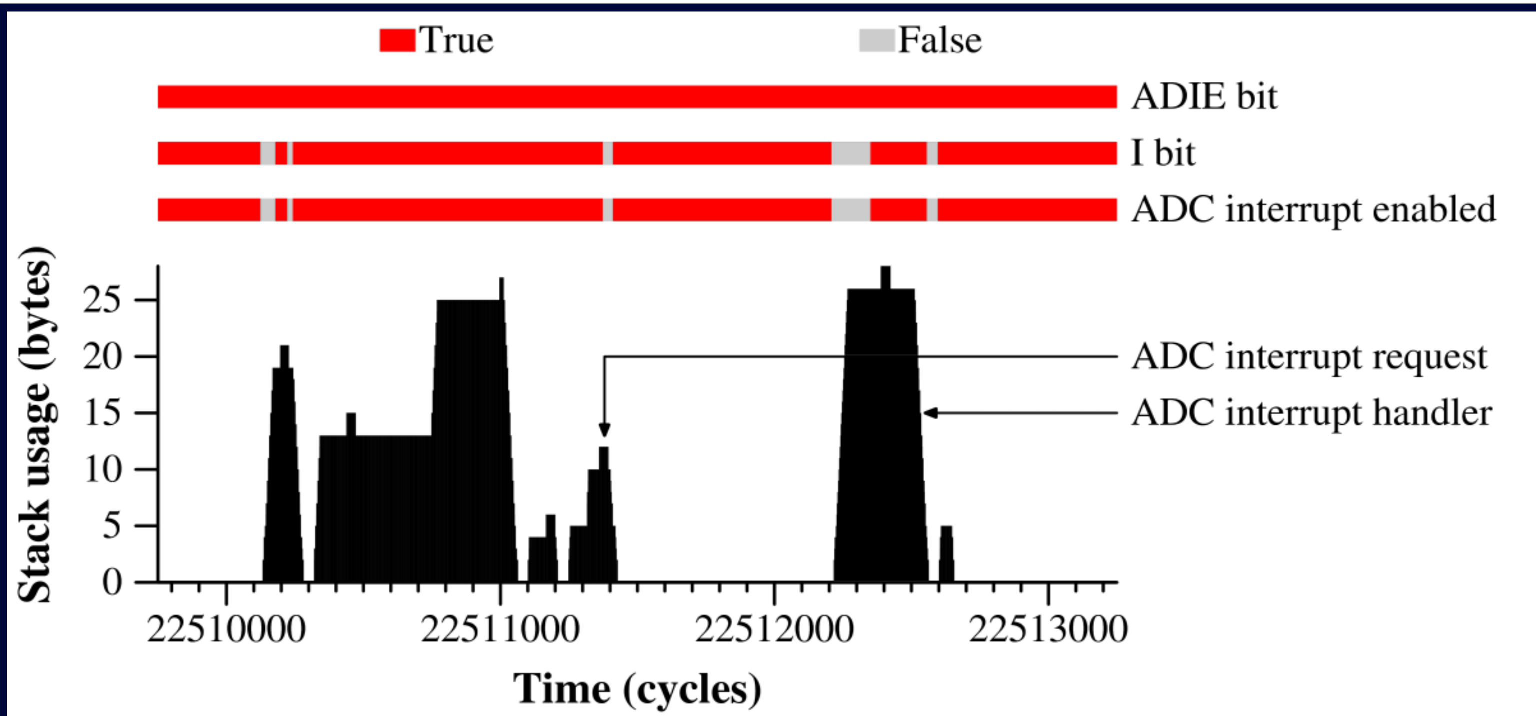
Implementing RID

1. Annotate interrupt requests
2. Ensure that device initialization code leaves each interrupt disabled
3. Run system through a source-to-source translator
   - Enable interrupt upon request
   - Disable requested interrupts upon interrupt
   - Suppress reentrant interrupts

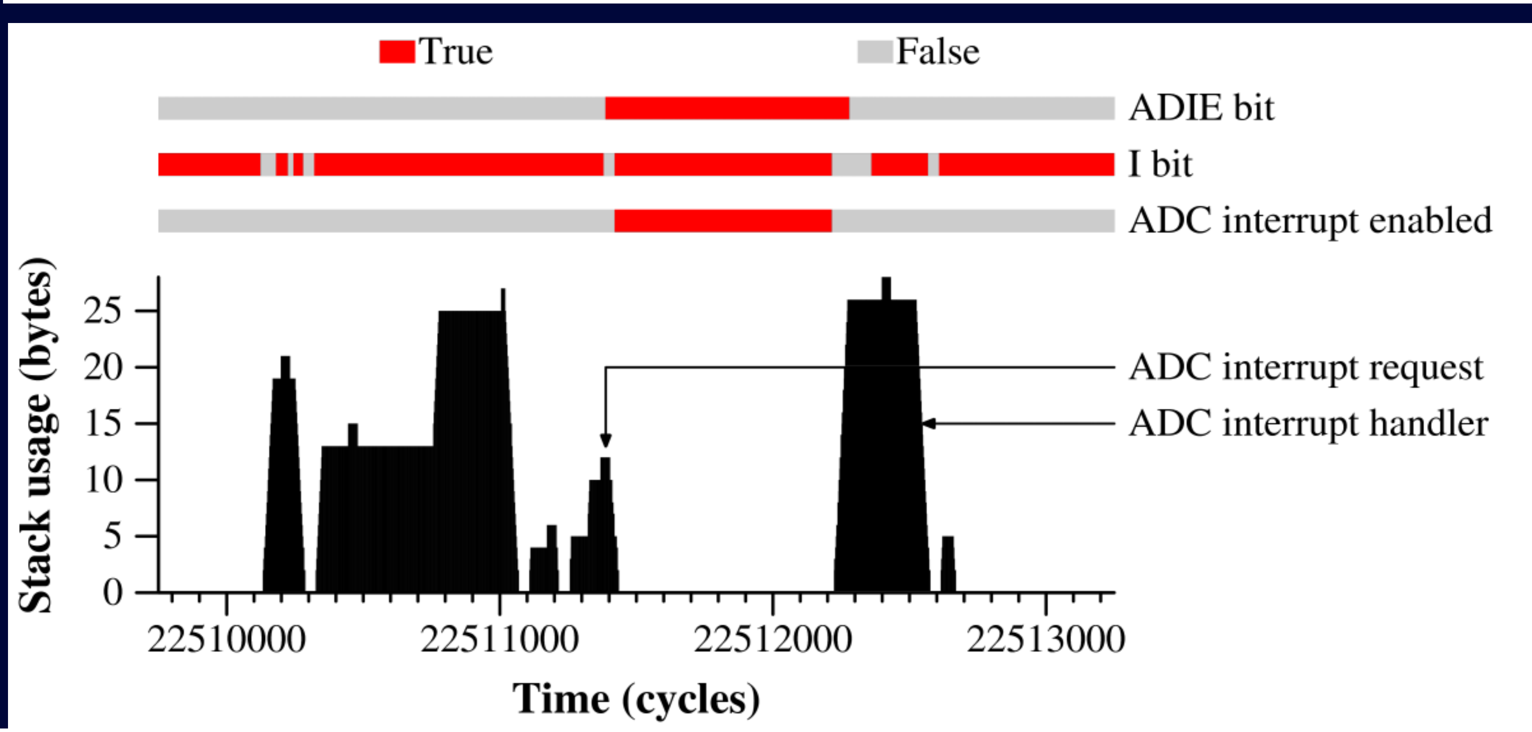
RID in TinyOS

- Implemented RID for five interrupt vectors
- Only bottom-level device driver files modified
  - A few LOC modified per vector
  - Normal developers don't touch these files
- Use custom CIL extension for src-src translation of C code output by nesC compiler

Without RID



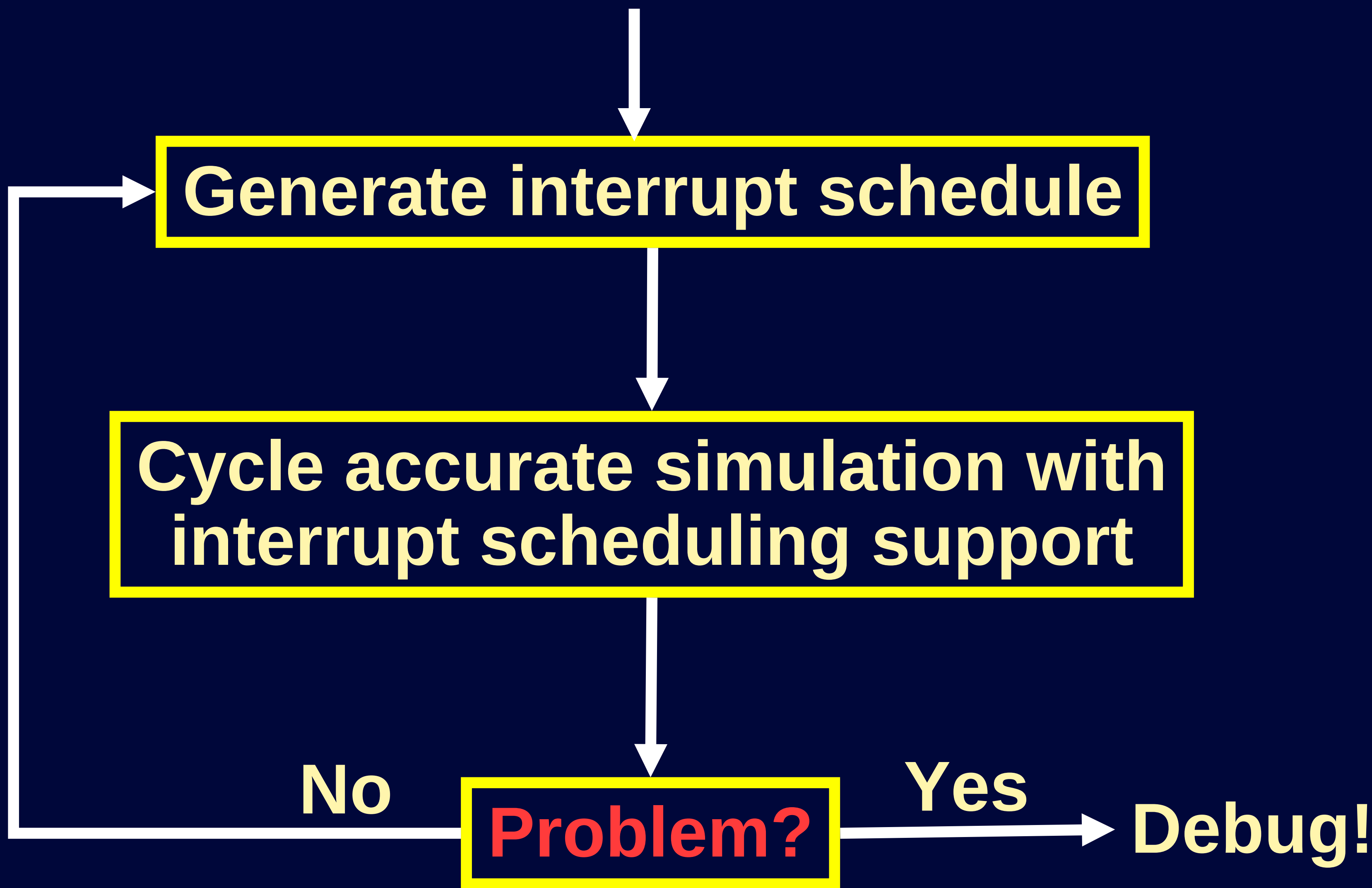
With RID



RID Benefits

- Enables random testing by suppressing aberrant and reentrant interrupts
- Hardens embedded system with respect to unexpected interrupts after deployment
  - SW bugs can cause these
  - So can loose wires, EMI, or other HW problems

Back to Random Testing



Interrupt Schedules

- List of pairs
  - (vector #, firing time)
- Schedule generator parameterized by density for each interrupt vector

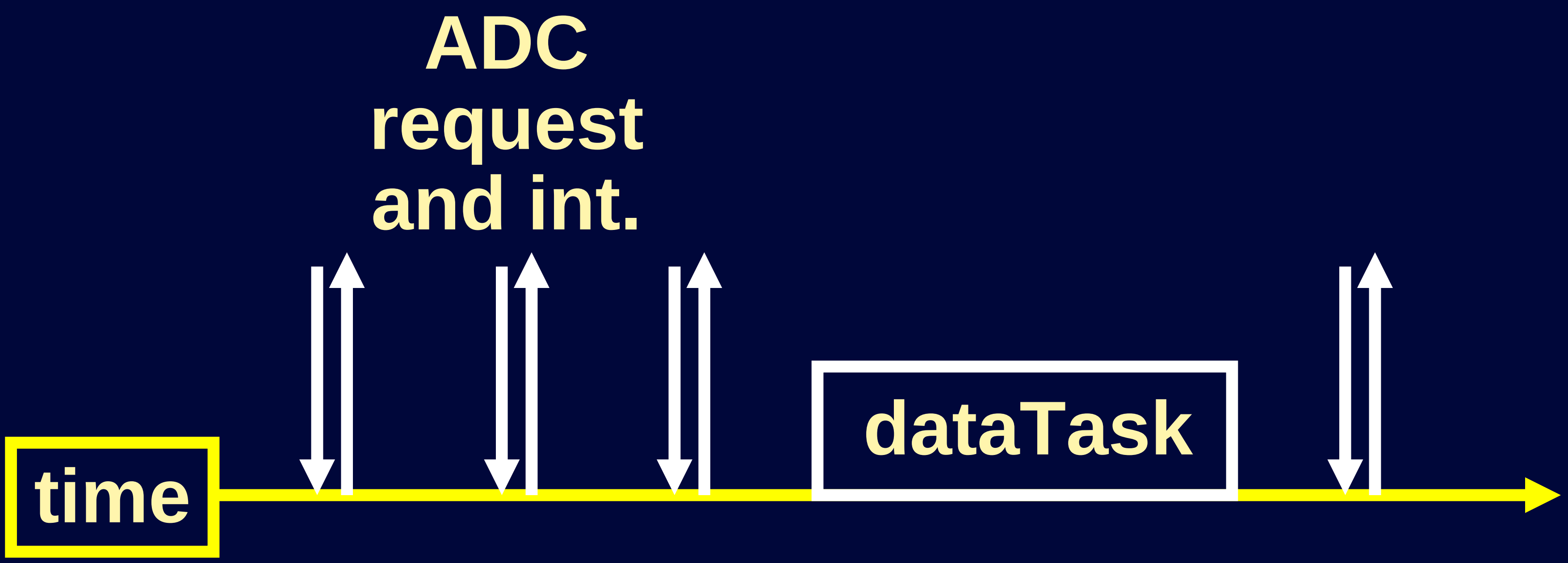
Simulator Support

- We hacked Avrora – sensor net simulator from UCLA
  - Our interrupt scheduling patches now included in the distribution

Detecting Failure

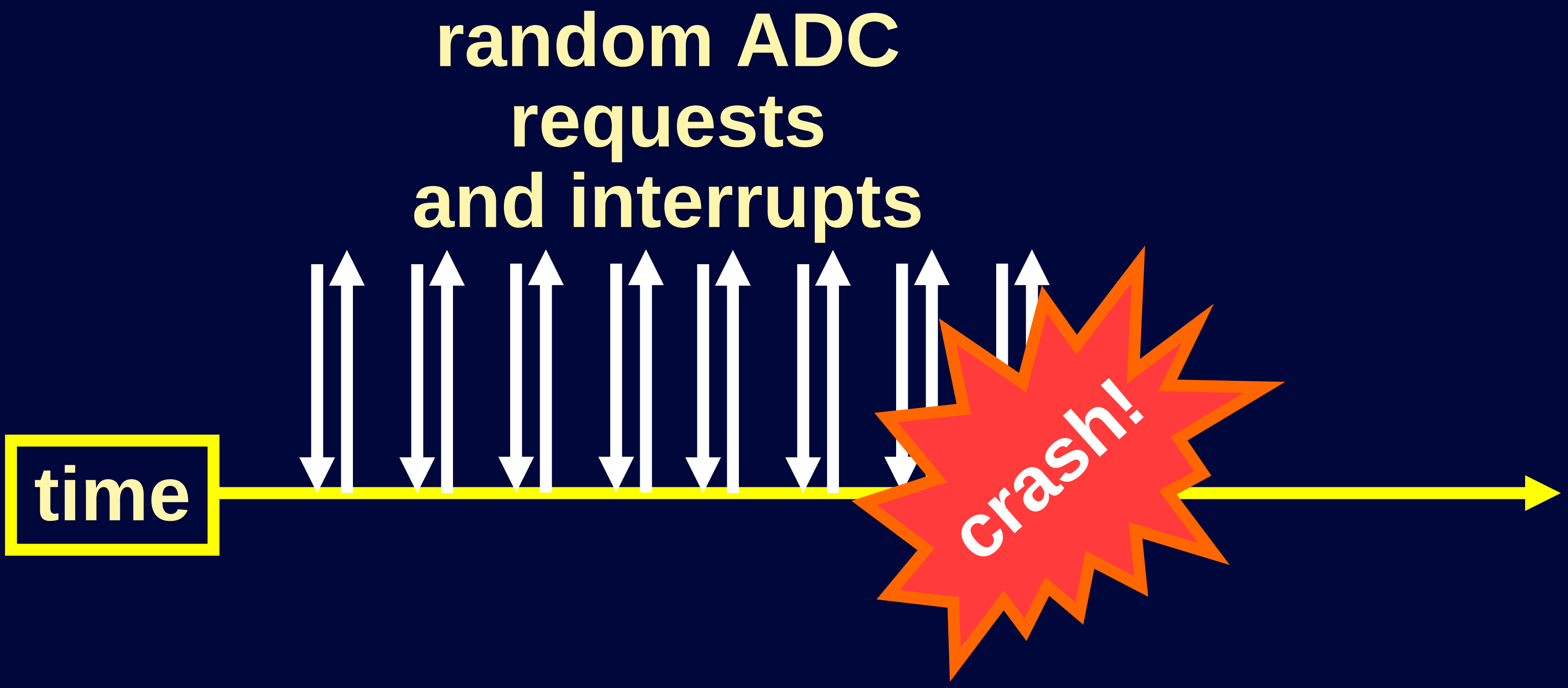
1. **Ask the application – See if it responds to network packets**
2. **Ask the simulator – Avrora reports illegal memory access and illegal instructions**

TinyOS Oscilloscope Bug



- Interrupt stores data into array
- dataTask resets buffer pointer
- No interlock between interrupt and task

TinyOS Oscilloscope Bug

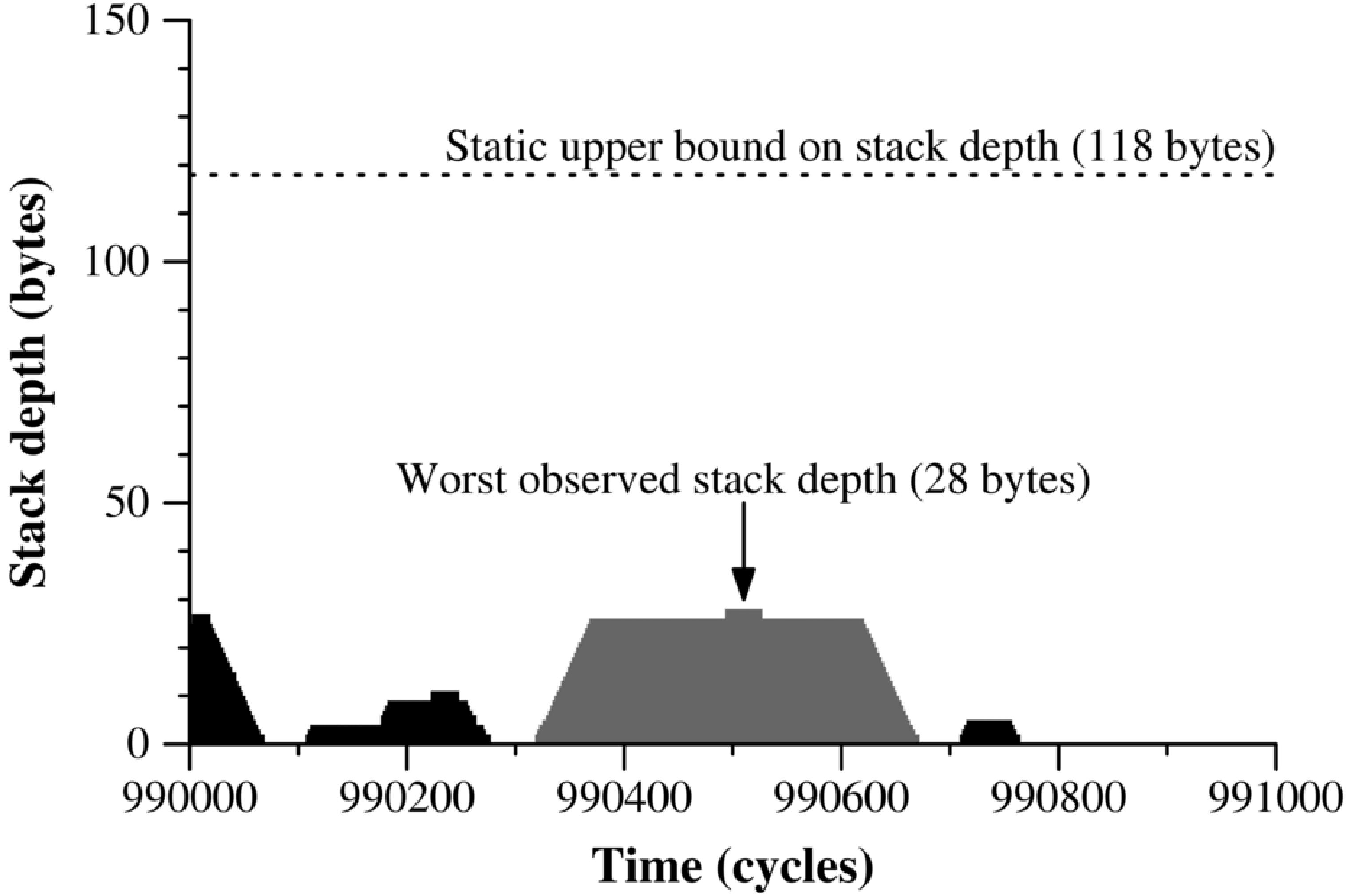


- Buffer overrun kills the system unless dataTask runs on time

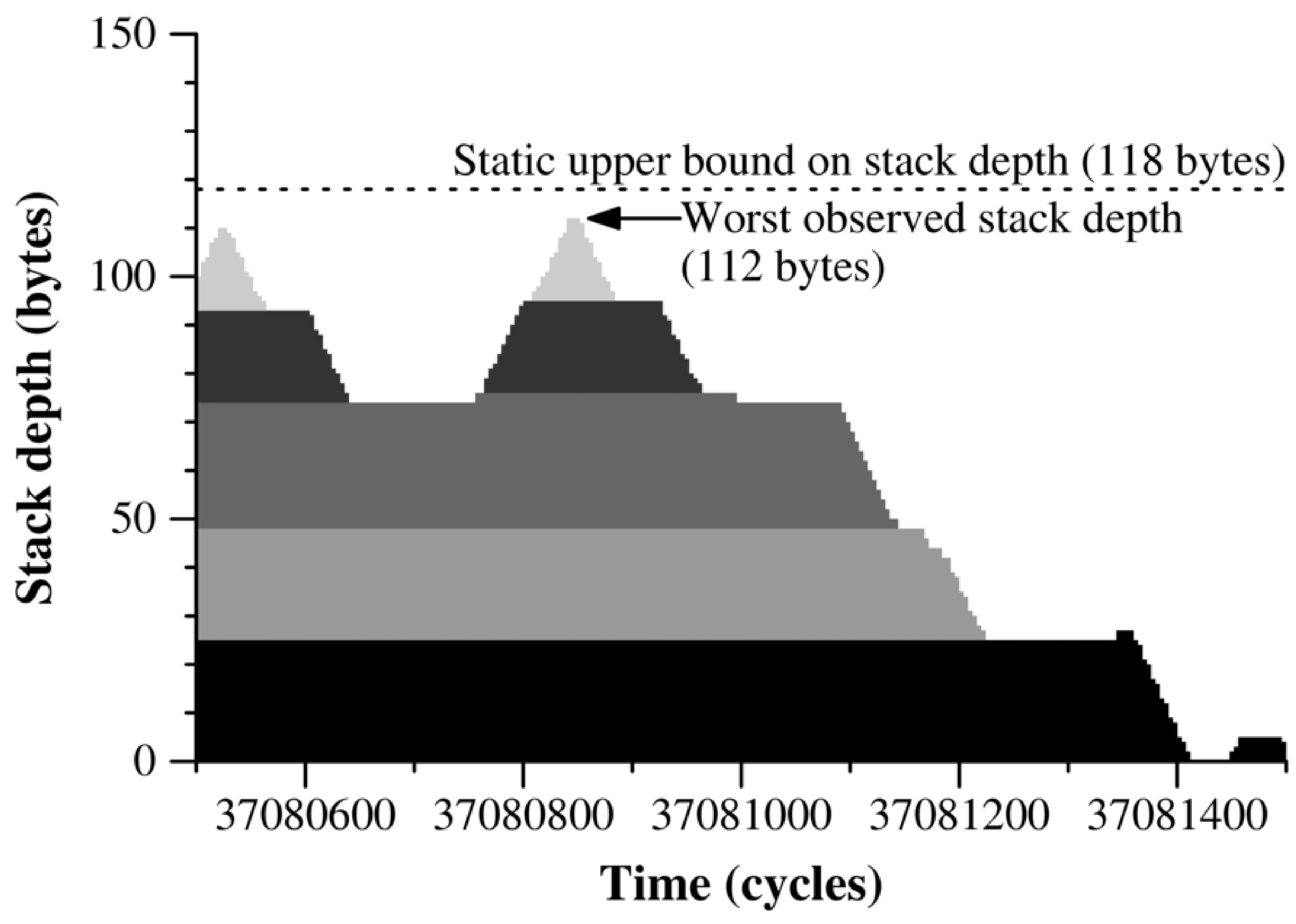
- **Original interrupt schedule that triggers bug is > 300,000 interrupts**
  - **Hard to tell what went wrong!**
- **Used “delta debugging” algorithm to minimize schedule**
  - **Can trigger bug with just 75 interrupts**
  - **Bug much easier to find now**
- **Fixing the bug: Easy – add array bounds check**

- Problem: Stack overflow kills sensor network programs
- Solution: Compute WC stack depth through static analysis of binaries
- Lingering questios:
  - Is the bound actually conservative?
  - If so, how pessimistic is the bound?
- Answer: Testing

Stack Depth w/o Random



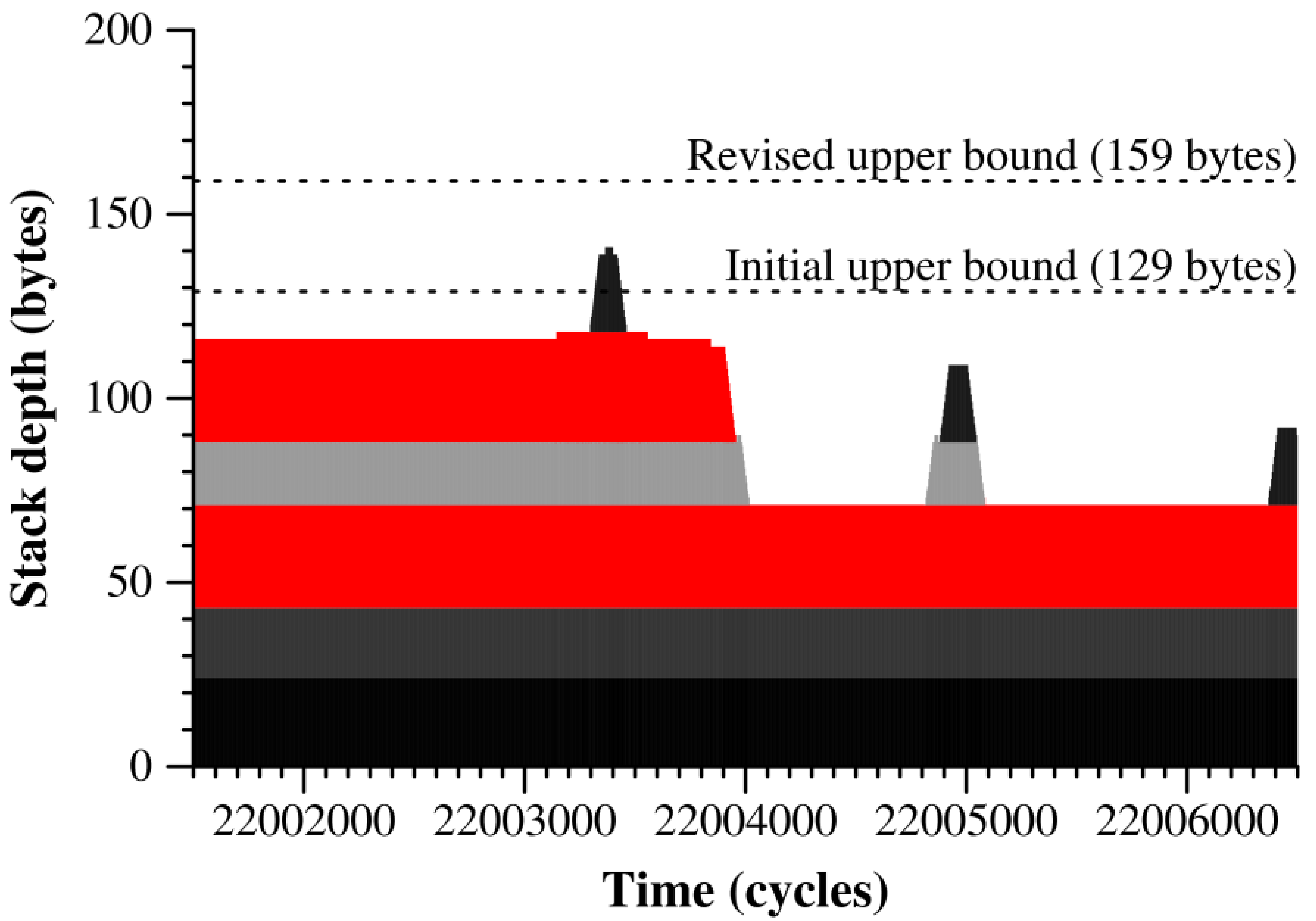
Stack Depth w/Random



Finding Deep Stacks

- Pure random testing doesn't cut it
  - Program behavior surprisingly sensitive to interrupt schedule density and structure
  - Even running overnight did not find schedules that make deep stacks
- Solution: Genetic algorithm evolves better interrupt schedules
  - About 100 generations to find deepest stack
  - 3 hours CPU time

Revising a Stack Depth Bound



Conclusions

- Random interrupt testing: Good
- Restricted Interrupt Discipline makes it work
  - Src-src transformation makes RID easy to implement
  - GA does directed search for interesting schedules
  - Delta finds interesting subsets of large interrupt schedules