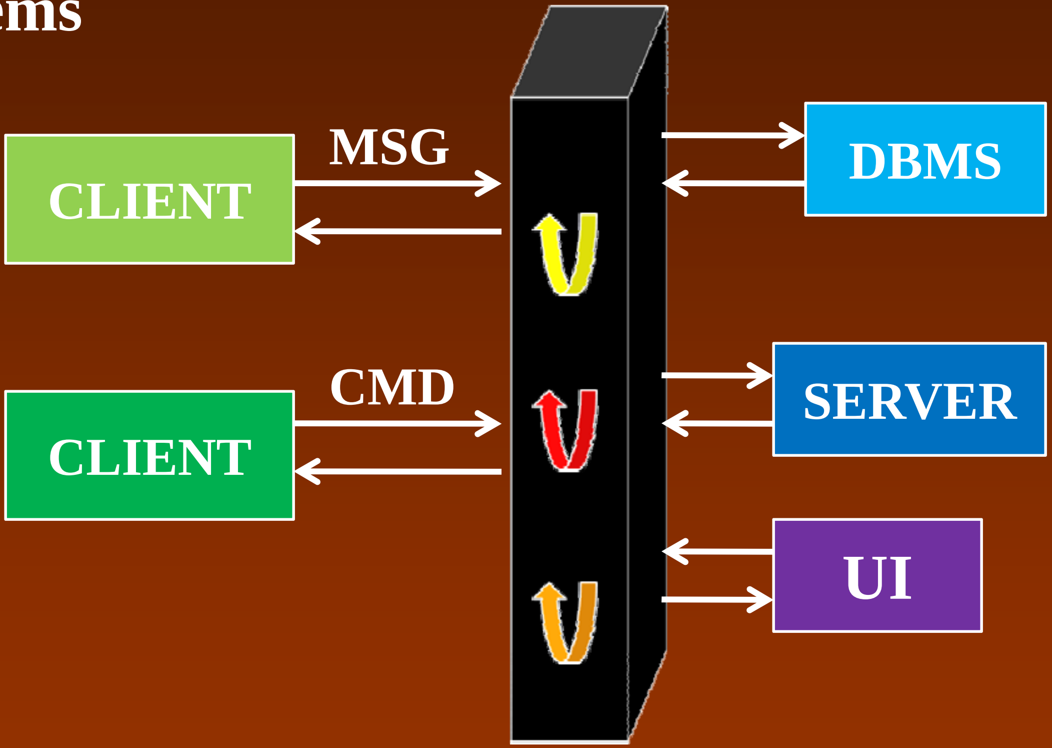
Dynamic Detection of Event Handlers

Steven P. Reiss
Brown University

Server Application

- Today's complex systems
- Interactions
  - Messages
  - Replies
  - UI (w/ updates)
- Response
  - DB reply
  - Computation
- Multiple threads



What Is Interesting

- Information per interaction
  - Performance for particular events
    - Inputs, transactions, user actions, …
  - Outputs associated with particular inputs
  - Events that result from other events
    - Protocol model
  - Dependencies between interactions
- Analysis per interaction
- How threads are used

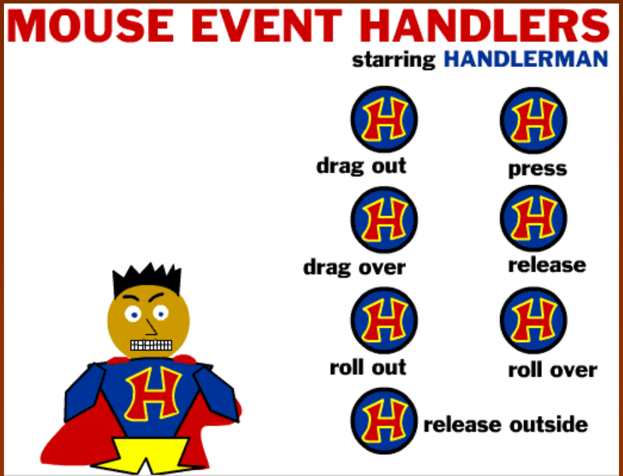
What Is Required

- Understanding event processing
  - How events are processed
  - When events are processed
  - Where events are processed
  - What events are processed
  - What happens when processing events
- Associating threads with events
- FIRST STEP:
  - IDENTIFY THE EVENT HANDLERS

What Is An Event Handler

- "An asynchronous callback subroutine that handles inputs received in a program" (Wikipedia)
- Code of the form

```
LOOP
   E = Get next event
   Process event E
END
```



Complications

- Code to get event can have many forms:
  - Get next event in a routine
  - Get next event from a queue
  - Get next event by waiting on interrupt/notify
  - Process the read in line (socket read)
    - Check for complete message, loop if not
  - Callback from user interface
  - Callback from asynchronous I/O
  - Observer pattern (publish-subscribe)

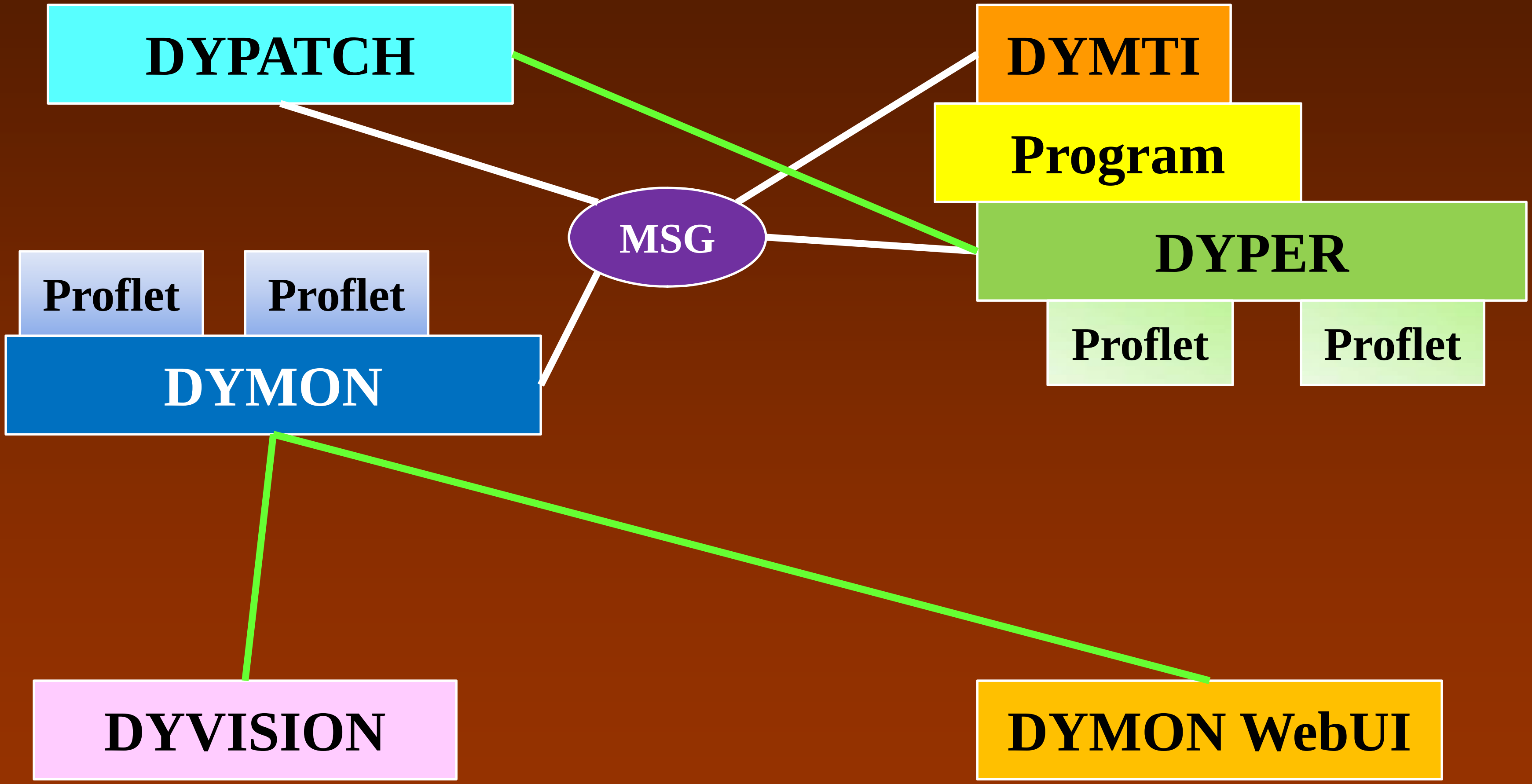
Complications

- Code for event can have many forms
  - Call one routine
  - Call multiple routines
  - Switch to detect event type
    - Then call appropriate routine for event type
  - Parse/decode the message, then call handler
  - Debugging/logging statements
  - …

DYPER/DYMON

- Dynamic analysis of performance issues
  - Fixed overhead analysis
  - Works with multiple dimensions
    - CPU, Memory, I/O, Heap, Threads, Sockets, …
    - Each represented by a proflet
- Proflets have two components
  - One collects data
    - Based on stack samples
    - Based on scheduled detailed analysis
  - One analyzes and reports the result

Basic Architecture



Reaction Proflet

- Find event handlers
  - Accumulate performance data per event
- Components
  - Data collection to find event handlers
    - Based on stack traces
  - Data analysis to identify the handlers
  - Performance analysis based on handlers
    - Counts of time spent in each
    - Detailed analysis by instrumenting the handlers

Data Collection

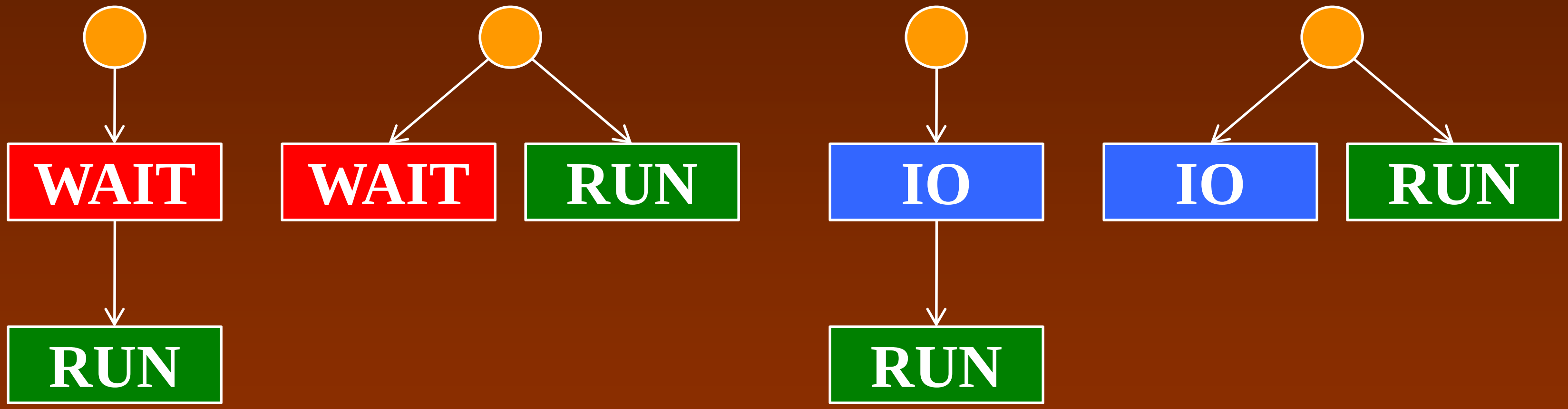
- Detect callbacks
  - User routines called from system code
    - Usually (when does it not work?)
- Consolidating call information
  - Build a trie of calls
  - Accumulated from all threads
  - Leaf is stack base
    - Children are routines called
    - Only do user routines
  - Keep state counts for each node
    - RUN, IO, WAIT

Data Collection Example

```
<REACTIONS LAST='1216066762950' MONTIME='119488' SAMPLES='3765' TSAMPLES='8017'>
<CALLBACK STACK='21' USER='edu.brown.cs.cs032.crawler.crawl.CrawlSwingParser$Callback@handleStartTag' />
<CALLBACK STACK='2' USER='edu.brown.cs.cs032.crawler.crawl.CrawlSwingParser$Callback@handleText' />
<TRIE>
<TRIENODE CLASS='edu.brown.cs.cs032.crawler.crawl.CrawlThread' IO='0' METHOD='run' RUN='0' WAIT='0'>
<TRIENODE CLASS='edu.brown.cs.cs032.crawler.crawl.CrawlMain' IO='0' METHOD='getNextUrl' RUN='0' WAIT='59753'>
<TRIENODE CLASS='edu.brown.cs.cs032.crawler.url.UrlManager' IO='0' METHOD='beginProcessing' RUN='0' WAIT='0'>
<TRIENODE CLASS='edu.brown.cs.cs032.crawler.url.UrlManager' IO='1' METHOD='getDirectory' RUN='0' WAIT='0' />
</TRIENODE>
</TRIENODE>
<TRIENODE CLASS='edu.brown.cs.cs032.crawler.crawl.CrawlThread' IO='0' METHOD='processUrl' RUN='0' WAIT='0'>
<TRIENODE CLASS='edu.brown.cs.cs032.crawler.url.UrlHandle' IO='449' METHOD='saveHeader' RUN='0' WAIT='0' />
<TRIENODE CLASS='edu.brown.cs.cs032.crawler.url.UrlHandle' IO='0' METHOD='saveHtml' RUN='1' WAIT='0' />
<TRIENODE CLASS='edu.brown.cs.cs032.crawler.crawl.CrawlParser' IO='0' METHOD='parse' RUN='0' WAIT='0'>
<TRIENODE CLASS='edu.brown.cs.cs032.crawler.crawl.CrawlSwingParser' IO='21' METHOD='localParse' RUN='10' WAIT='0' />
</TRIENODE>
<TRIENODE CLASS='edu.brown.cs.cs032.crawler.url.UrlHandle' IO='1' METHOD='endProcessing' RUN='1' WAIT='0' />
<TRIENODE CLASS='edu.brown.cs.cs032.crawler.crawl.CrawlThread' IO='2' METHOD='readContents' RUN='1' WAIT='0' />
</TRIENODE>
</TRIENODE>
<TRIENODE CLASS='edu.brown.cs.cs032.crawler.crawl.CrawlMain' IO='0' METHOD='main' RUN='0' WAIT='0'>
<TRIENODE CLASS='edu.brown.cs.cs032.crawler.crawl.CrawlMain' IO='0' METHOD='process' RUN='0' WAIT='0'>
<TRIENODE CLASS='edu.brown.cs.cs032.crawler.crawl.CrawlMain' IO='0' METHOD='loadUrls' RUN='0' WAIT='0'>
<TRIENODE CLASS='edu.brown.cs.cs032.crawler.url.UrlManager' IO='3763' METHOD='normalizeNewUrl' RUN='2' WAIT='0' />
</TRIENODE>
</TRIENODE>
</TRIENODE>
</TRIE>
</REACTIONS>
```

Data Analysis

- Handle callbacks
- Look for Patterns in the trie



- Ensure significance based on counts
  - Both relative and absolute
  - Cutoffs determined experimentally

Results Example

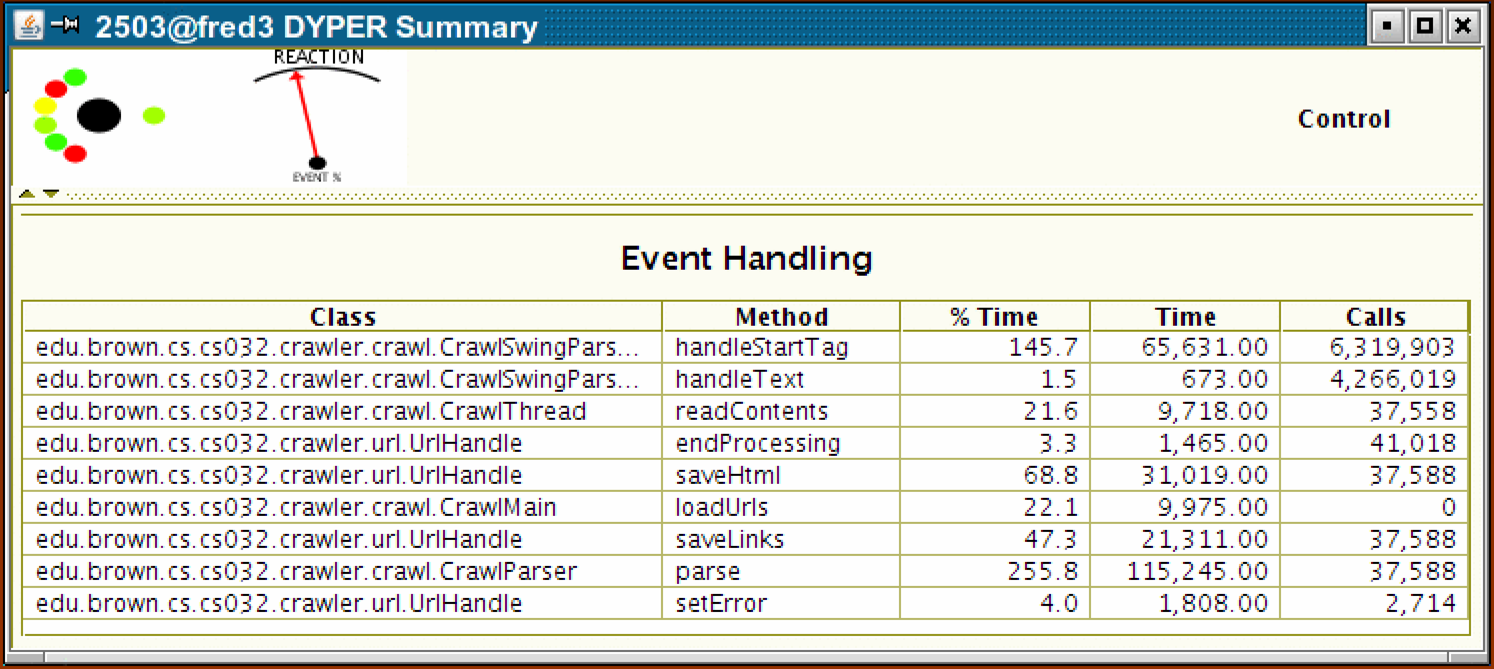
```
<REACTION TOTTIME='493.81' TOTSAMP='39489'>
<CALLBACK METHOD='edu.brown.cs.cs032.crawler.crawl.CrawlSwingParser$Callback@handleEndTag' />
<CALLBACK METHOD='edu.brown.cs.cs032.crawler.crawl.CrawlSwingParser$Callback@handleStartTag' />
<CALLBACK METHOD='edu.brown.cs.cs032.crawler.crawl.CrawlSwingParser$Callback@handleError' />
<CALLBACK METHOD='edu.brown.cs.cs032.crawler.crawl.CrawlSwingParser$Callback@handleText' />
<EVENT METHOD='edu.brown.cs.cs032.crawler.url.UrlHandle@saveHtml' TYPE='NODE_IO' />
<EVENT METHOD='edu.brown.cs.cs032.crawler.crawl.CrawlMain@addRedirectUrl' TYPE='NODE_IO' />
<EVENT METHOD='edu.brown.cs.cs032.crawler.url.UrlHandle@endProcessing' TYPE='NODE_IO' />
<EVENT METHOD='edu.brown.cs.cs032.crawler.crawl.CrawlMain@loadUrls' TYPE='NODE_WAIT' />
<EVENT METHOD='edu.brown.cs.cs032.crawler.url.UrlHandle@openConnection' TYPE='NODE_IO' />
<EVENT METHOD='edu.brown.cs.cs032.crawler.url.UrlHandle@setError' TYPE='NODE_IO' />
<EVENT METHOD='edu.brown.cs.cs032.crawler.crawl.CrawlThread@readContents' TYPE='NODE_IO' />
<EVENT METHOD='edu.brown.cs.cs032.crawler.url.UrlHandle@setRedirectHtml' TYPE='NODE_IO' />
<EVENT METHOD='edu.brown.cs.cs032.crawler.crawl.CrawlParser@parse' TYPE='NODE_IO' />
<EVENT METHOD='edu.brown.cs.cs032.crawler.url.UrlHandle@saveLinks' TYPE='NODE_IO' />
</REACTION>
```

Results

- Works on a wide variety of applications
  - Peer-to-peer server
  - Particle simulator
  - Web crawler
  - Code search engine
  - Simple test cases
- Some extra items detected
  - Access$100, uses of reflection
  - Others that are effectively handlers, but not thought of as such
    - currentTimeMillis()

Applications: DYMON

- **Reaction statistics in DYMON**



Event Handling

Class	Method	% Time	Time	Calls
edu.brown.cs.cs032.crawler.crawl.CrawlSwingPars...	handleStartTag	145.7	65,631.00	6,319,903
edu.brown.cs.cs032.crawler.crawl.CrawlSwingPars...	handleText	1.5	673.00	4,266,019
edu.brown.cs.cs032.crawler.crawl.CrawlThread	readContents	21.6	9,718.00	37,558
edu.brown.cs.cs032.crawler.url.UrlHandle	endProcessing	3.3	1,465.00	41,018
edu.brown.cs.cs032.crawler.url.UrlHandle	saveHtml	68.8	31,019.00	37,588
edu.brown.cs.cs032.crawler.crawl.CrawlMain	loadUrls	22.1	9,975.00	0
edu.brown.cs.cs032.crawler.url.UrlHandle	saveLinks	47.3	21,311.00	37,588
edu.brown.cs.cs032.crawler.crawl.CrawlParser	parse	255.8	115,245.00	37,588
edu.brown.cs.cs032.crawler.url.UrlHandle	setError	4.0	1,808.00	2,714

Other Applications

- Creating a model of thread behavior
  - Based on what events a thread deals with
  - Based on what each event does
    - Thread states: CPU, WAIT, IO, BLOCK, …
  - Can be used to predict performance
    - Based on # threads, # processors, …
- Building a model of message processing
  - What messages cause other messages
  - Determining the message protocols

Conclusion

- Simple dynamic analysis
  - Can identify event handlers
  - Can serve as a basis for event-based analysis
  - Very low overhead
- The TRIE-counting techniques generalize
  - Building model of thread processing
  - Common understanding of behavior
    - Statistical basis
    - That combines multiple threads or processes
- IT WORKS

Acknowledgements

- NSF support: CCR0613162
- Code is available
  - ftp://ftp.cs.brown.edu/u/spr/wadi.tar.gz
  - Part of the WADI project

Questions / Comments