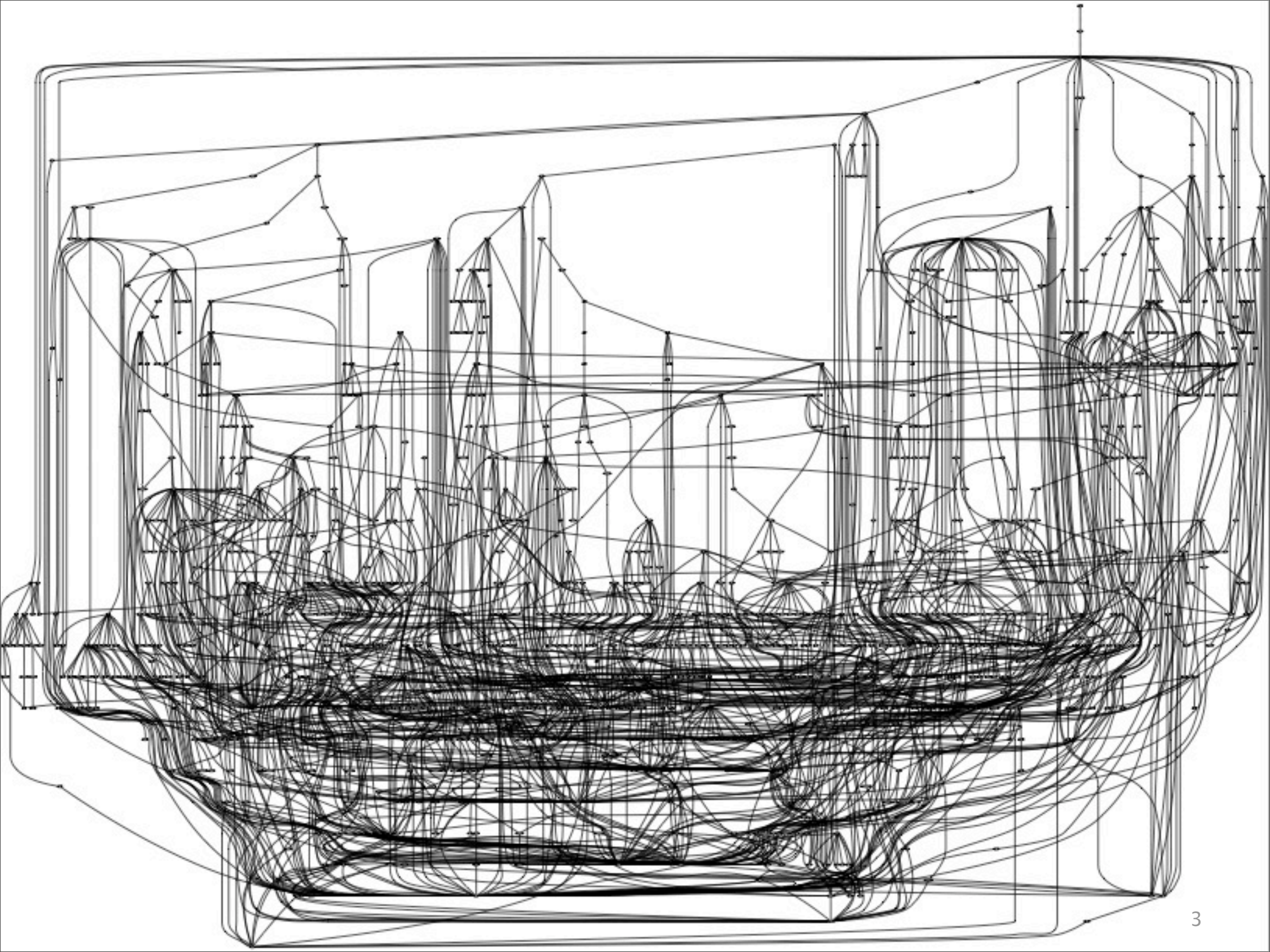
Tralfamadore
Unifying Source Code and Execution Experience
(short paper)

Geoffrey Lefebvre, Brendan Cully,
Dutch Meyer, Michael Feeley,
Norm Hutchinson and Andrew Warfield

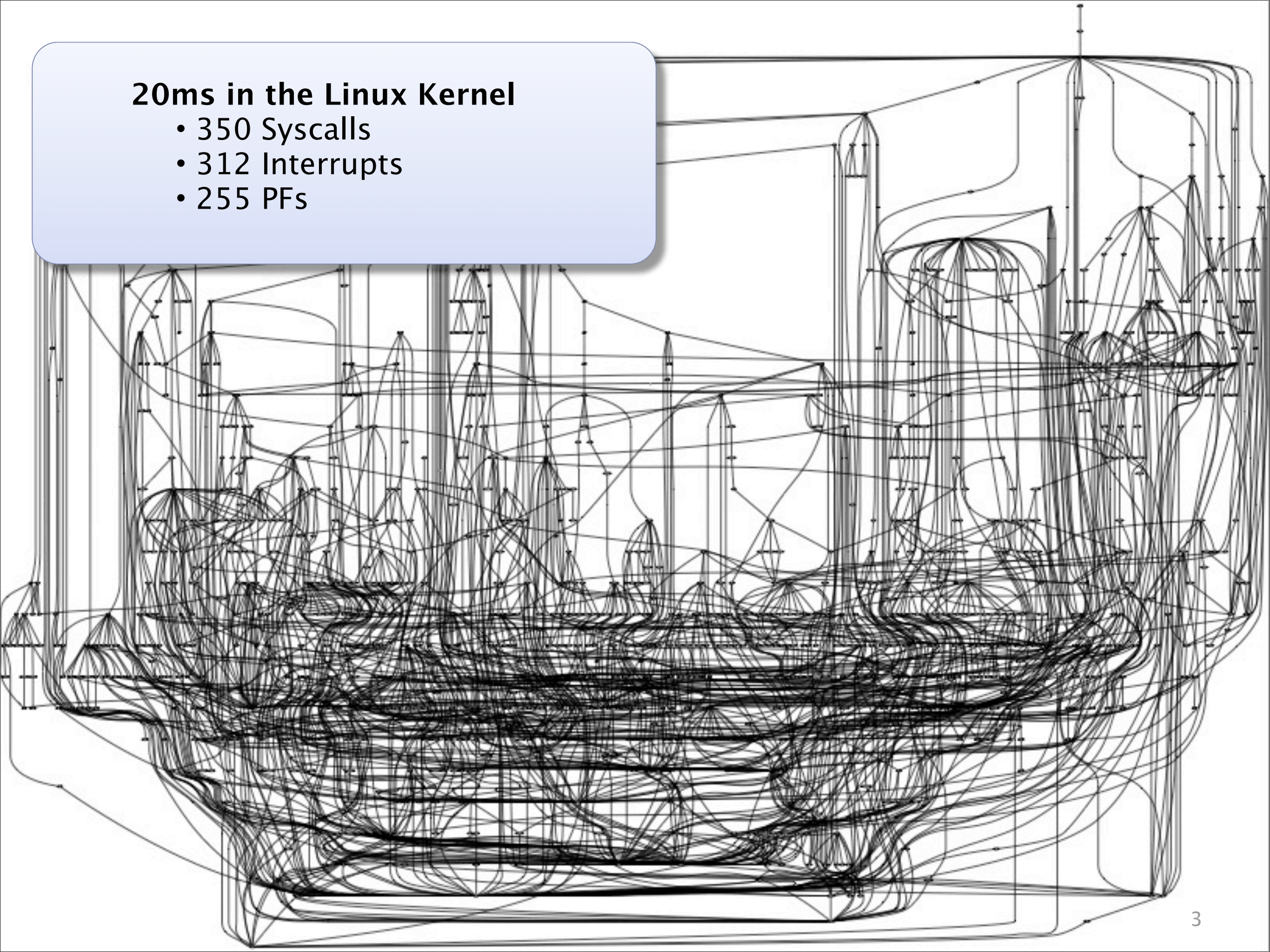
University of British Columbia

```
1760 int netif_rx(struct sk_buff *skb)
1761 {
1762      struct softnet_data *queue;
1763      unsigned long flags;
1764
1765      /* if netpoll wants it, pretend we never saw it */
1766      if (netpoll_rx(skb))
1767          return NET_RX_DROP;
1768
1769      if (!skb->tstamp.tv64)
1770          net_timestamp(skb);
1771
1772      /*
1773       * The code is rearranged so that the path is the most
1774       * short when CPU is congested, but is still operating.
1775       */
1776      local_irq_save(flags);
1777      queue = &__get_cpu_var(softnet_data);
1778
1779      __get_cpu_var(netdev_rx_stat).total++;
1780      if (queue->input_pkt_queue.qlen <= netdev_max_backlog) {
1781          if (queue->input_pkt_queue.qlen) {
1782 enqueue:
1783              dev_hold(skb->dev);
1784              __skb_queue_tail(&queue->input_pkt_queue, skb);
1785              local_irq_restore(flags);
```



20ms in the Linux Kernel

- 350 Syscalls
- 312 Interrupts
- 255 PFs



```
1760 int netif_rx(struct sk_buff *skb)
1761 {
1762      struct softnet_data *queue;
1763      unsigned long flags;
1764
1765      /* if netpoll wants it, pretend we never saw it */
1766      if (netpoll_rx(skb))
1767          return NET_RX_DROP;
1768
1769      if (!skb->tstamp.tv64)
1770          net_timestamp(skb);
1771
1772      /*
1773       * The code is rearranged so that the path is the most
1774       * short when CPU is congested, but is still operating.
1775       */
1776      local_irq_save(flags);
1777      queue = &__get_cpu_var(softnet_data);
1778
1779      __get_cpu_var(netdev_rx_stat).total++;
1780      if (queue->input_pkt_queue.qlen <= netdev_max_backlog) {
1781          if (queue->input_pkt_queue.qlen) {
1782 enqueue:
1783              dev_hold(skb->dev);
1784              __skb_queue_tail(&queue->input_pkt_queue, skb);
1785              local_irq_restore(flags);
```

Well…
lets just **record everything!**

- Record system execution to disk
- Capture all effects of execution
  - –All updates to registers & memory
- Provide tools to analyze after the fact
  - –Run analysis on the trace instead of the running system

Well…
lets just **record everything!**

- Record system execution to disk
- C
- F

```
…
I[0849e46c]:LDB[0862e548]=00
I[0849e470]:EFL=00000246
E[ef,0000,0849e472]:LDL[c0429778]=00608e94 LDL[c042977c]=c0108e00
LDL[c1204060]=0000ffff LDL[c1204064]=00cf9a00 LDL[c1207104]=cf51bff8
LDW[c1207108]=0068 LDL[c1204068]=0000ffff LDL[c120406c]=00cf9300
STL[cf51bff4]=0000007b STL[cf51bff0]=bfffc700 STL[cf51bfec]=00000246
STL[cf51bfe8]=00000073 STL[cf51bfe4]=0849e472
S[SS]=0068,00000000,ffffffff,00cf9300 R[ESP]=cf51bfe4
S[CS]=0060,00000000,ffffffff,00cf9a00 BR=c0108e94 EFL=00000046
I[c0108e94]:STL[cf51bfe0]=ffffff10 R[ESP]=cf51bfe0
I[c0108e99]:EFL=00000046
I[c0108e9a]:STL[cf51bfdc]=00000000 R[ESP]=cf51bfdc
I[c0108e9c]:STL[cf51bfd8]=0000007b R[ESP]=cf51bfd8
…
```

Well…
lets just **record everything!**

- Record system execution to disk
- Capture all effects of execution
  - –All updates to registers & memory
- Provide tools to analyze after the fact
  - –Run analysis on the trace instead of the running system

System Overview

Target system

x86 execution capture

Analysis server

System Overview

Target system

Analysis server

x86 execution capture

1. Capture a detailed trace of execution.

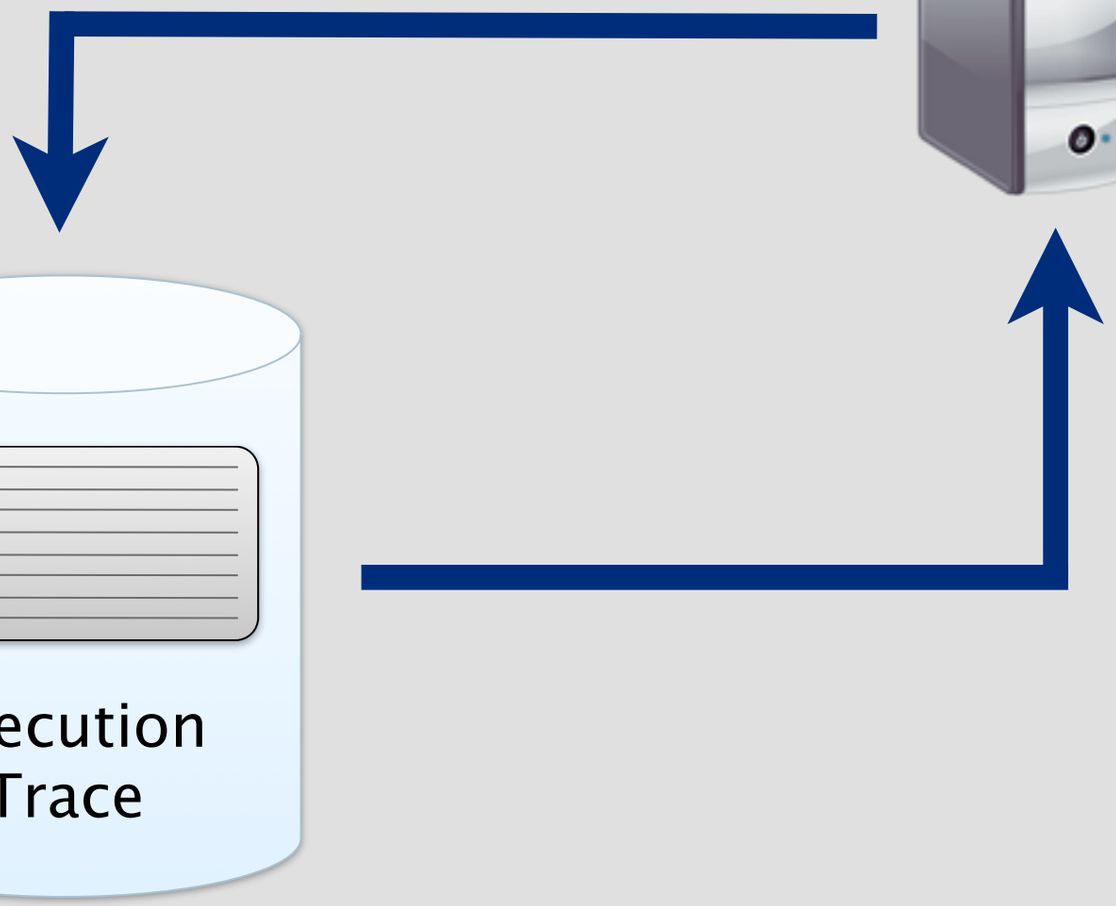
System Overview

Target system

x86 execution capture

"How is this function used?"

Analysis server



2. Queries to search, select, and interrogate history.

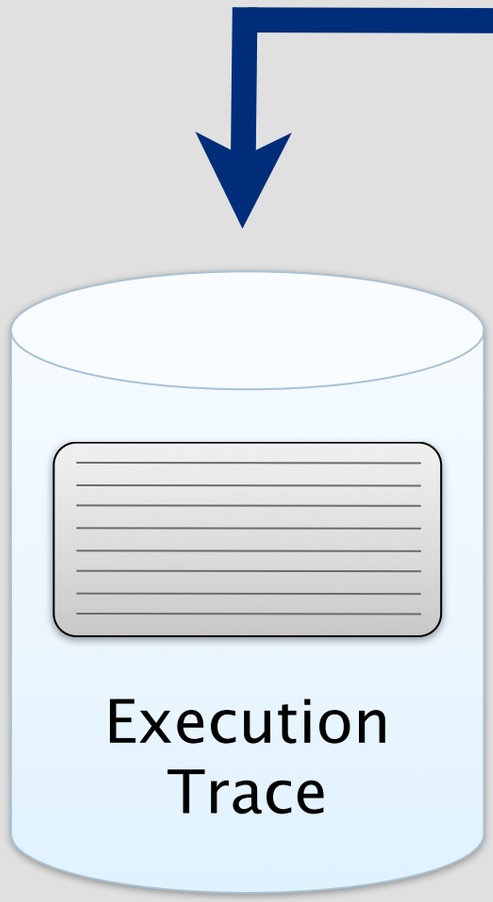
System Overview

Target system

x86 execution capture

"Are all accesses to struct X protected by lock Y? "

Analysis server



3. Refined queries over the same execution.

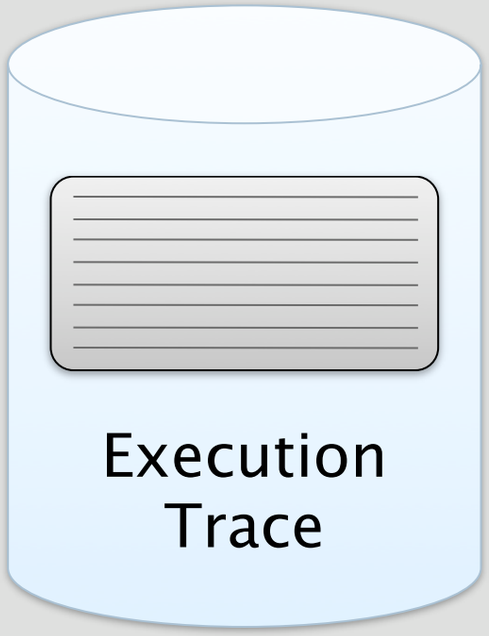
System Overview

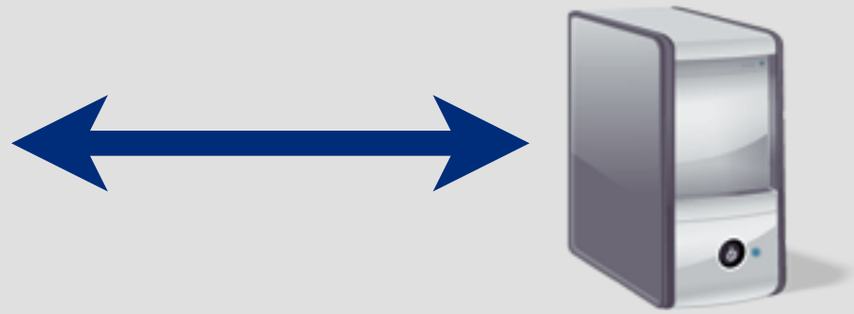
Target system

x86 execution capture

Analysis server

"Is this parameter ever null?"





4. The trace can be shared between users.

Challenge: Traces are huge!

- Kernel compilation workload
  - User space & kernel: ~8TB (kernel: ~400GB)
  - Disks are cheap :-)
  - Trace compression
  - Deterministic record/replay
    - Trace as soft state
- Analysis must be interactive
  - Workload is parallelizable
  - Most queries only need a small fraction of the trace
    - Index the trace

Challenge: The Semantic Gap

- Traces are very low level
  - Consists of CPU instructions and effects
  - Huge blobs of opaque data
- Reconstruct execution to a level people can understand
  - Threads, function invocation & return etc.
  - Or even higher:
    - object allocation, release, access
    - synchronization primitives (lock, unlock)
    - fork, exec, etc.

And now the fun stuff...

- Working prototype
  - programmatic interface

- Implemented web-based source browser
  - Provide a set of predefined queries:
    - Control flow paths
    - Argument values
    - Dataflow tracking of pointer arguments

Summary

- **Analyze execution itself** instead of analyzing software as it executes.
- **Tralfamadore** is all about treating program analysis as a database/data mining problem.
- Please come and try/crash our demo during the poster session!

"The creatures were friendly, and they could see in four dimensions. They pitied Earthlings for being able to see only three. They had many wonderful things to teach Earthlings about time."

-- A description of the Tralfamadorians from Kurt Vonnegut's "Slaughterhouse Five"

http://nystromkw.files.wordpress.com/2008/03/a_tralfamadorian_by_animatedpunk.jpg