Summer Institute on
Software Architecture

Embedded Systems Architecture 4: Methodical Optimization

Instructor: Calton Pu

calton.pu@cc.gatech.edu

Georgia Tech

Overall Structure (Day 1)

- Introduction to modern embedded systems
  - Ubiquitous computing as a vision for integrating future embedded systems
  - From embedded to resource constrained systems
  - Some basic techniques for constructing real-time embedded system software
- Principled embedded software infrastructure
  - Survey of real-time scheduling algorithms: static, dynamic priority, static priority dynamic
  - I/O processing and networking for embedded systems

Georgia Tech

Overall Structure (Day 2)

- Automotive embedded software architecture
  - Component-based software engineering
  - Case study on automotive embedded software
- **Sampling of methodical optimization of embedded software**
  - **Specialization of system software**
  - **Code generation and translation**
  - **Aspect-oriented programming**

Georgia Tech

Outline

- An Overview of Specialization
  - Static Specialization
  - Dynamic Specialization
  - Optimistic Specialization
- Specialization Toolkit
  - Tempo Specializer
- Specialization Examples
- Specialization in Infopipe

Georgia Tech

Specialization

- Operating system – too generic
- Specialization
  - A technique for optimizing systems code
  - An application of partial evaluation
  - Specialized, simplified component
  - Better performance!

Georgia Tech

Partial Evaluation

```
int Multiply(int a, int b)
{
   c = a * b;
   return c;
}

// What if we know the value of a?
```

Georgia Tech

Specialization Predicate

- Terminology
  - "page_size = 4K" is a specialization predicate
  - page_size is a specialization predicate term
  - 4K is a value
- Predicate characteristics
  - Static
  - Dynamic

Georgia Tech

Static Specialization

- Static predicates
- Benefits
  - "Off-line" specialization: no runtime overhead
- Limitations
  - Values must be known prior to runtime
  - Relatively few specialization predicates
  - Can't exploit runtime, or even boot-time knowledge

Georgia Tech

Dynamic Specialization

- Dynamic predicates – must hold
- Benefits
  – Exploits starting-time knowledge
- Limitations
  – Runtime overhead
  – Requires a fast runtime specializer
  – Specialization predicates must hold for remainder of the system lifetime

Georgia Tech

Optimistic Specialization

- Dynamic predicates
  – Need not hold for entire system lifetime
- Benefits
  – Can be used generally in OS code
- Limitations
  – Correctness: detecting when specialization predicates hold and cease to hold (guarding)
  – Performance: overhead of enabling and disabling specialized components (replugging)

Georgia Tech

Challenges

- Hard to identify predicates
  - Need system experts
- Hard to ensure correctness
  - Where to guard
- Error-prone, tedious work

- Solution: Specialization toolkit

Tempo Specializer

- Charles Consel, G. Muller, and team
- Based on partial evaluation
  - Generates C code
  - Find static and dynamic code
  - System programming features
  - Compile-time and run-time specialization
  - Need human help

MemGuard

- Detect changes of predicate terms
  - Uses virtual memory protection
  - Protection fault handler checks for violation before writes complete
- Effectiveness?
  - Correctness guaranteed
  - High overhead
  - Page-grained guarding

Georgia Tech

TypeGuard

- Static tool to detect updates
  - Finds all uses of a specified type
  - Reports line numbers for updates and leaks
  - Overloading and aliasing complicate instance-based approaches
- Effectiveness?
  - Finer-grained guarding
  - Correctness not guaranteed due to lack of type-safety in C
  - Still useful (false positives managable)

Georgia Tech

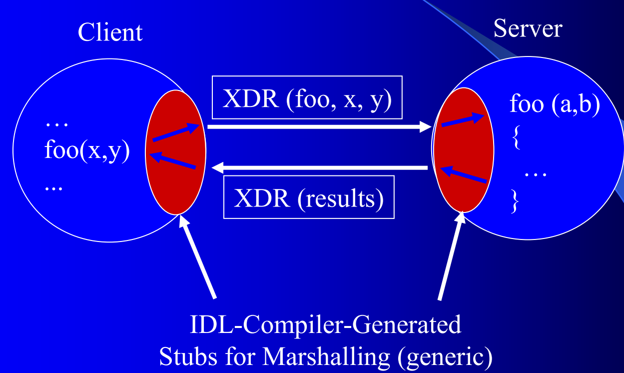
Replugger

- Implemented at function granularity (atomic swap of function pointers)
- Synchronizes replugging threads and normal threads

Specialization Examples

- Static specialization of Sun RPC
- Dynamic specialization of BPF
- Optimistic specialization of Linux signals

Example 1: Remote Procedure Call



Specializing RPC

- Predicates known at compile time
  - Message system parameters
    `(BUFSIZE == 8800)`
  - Processor-specific parameters
    `(sizeof (long) == 4)`
  - Exact purpose of marshalling routines
    `(x_op == XDR_ENCODE)`

Specializing RPC(2)

- Static specialization
  - Applied at client and server
  - Tempo processes IDL compiler output + specialization predicates
  - C compilation of client and server code and specialized stubs

Simple Example

```
bool_t xdr_long(xdrs,lp)
       XDR *xdrs;
       long *lp;
{
       if( xdrs->x_op == XDR_ENCODE )
              return XDR_PUTLONG(xdrs,lp);
       if( xdrs->x_op == XDR_DECODE )
              return XDR_GETLONG(xdrs,lp);
       ...
       return FALSE;
}
```

Specialization predicate for encoding:

`xdrs->x_op == XDR_ENCODE`

Resulting specialized function can be inlined:

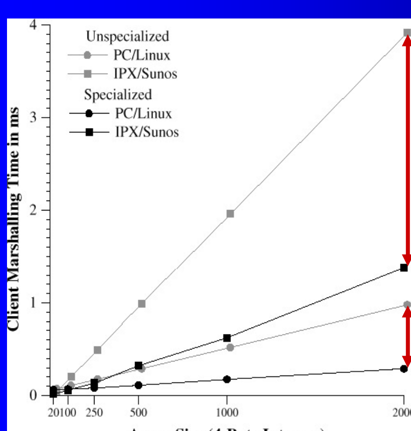
`XDR_PUTLONG(xdrs, lp)`

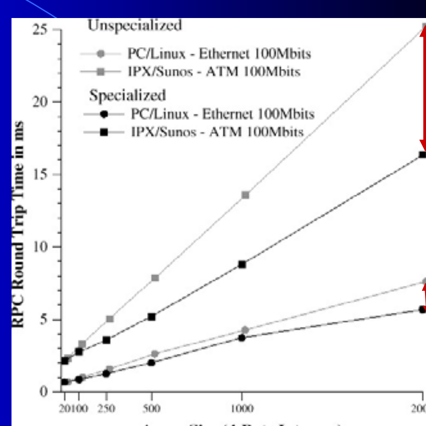
More Opportunities

- Avoid buffer boundary check
- Avoid return value check
- Loop unrolling
- Others

Georgia Tech

RPC Performance Results





Georgia Tech

Impact on Code Size

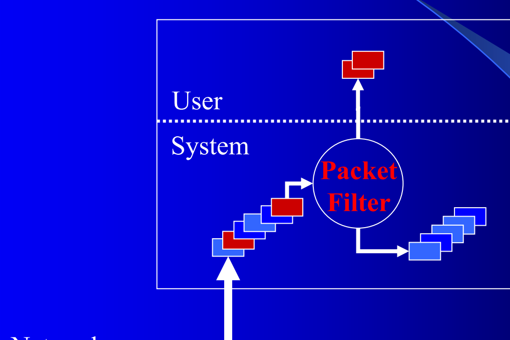
Message Size (bytes)

	20	100	500	1000	2000
Generic	20004	20004	20004	20004	20004
Specialized	24340	27540	33540	63540	111348

Code size of SunOS binaries (in bytes)

Georgia Tech

Example 2: Packet Filtering



Georgia Tech

The Berkeley Packet Filter

A DSL interpreter for filtering network packets

```
tcpdump -d host x
{     …
    create BPF_program from
  arguments;
    pcap_set
  (BPF_progr
    pcap_loop
    return;
}
```

```
for ever
{
 read packet;
 if bpf_filter (BPF_program, packet, …)
       upcall to print packet;
}
```


Specializing BPF

- Option 1 - Static specialization
- Option 2 - Dynamic specialization
  - When program is presented at execution time
  - Statically specialize BPF interpreter for a constant BPF program of unknown value
    - generates a runtime specializer + binary templates
  - Dynamically specialize when BPF program value is known
    - fill template holes, evaluate static parts

Performance Results for BPF

Time taken to process 10MB data (~10,000 packets):

Program	Run time	Interpretation time
Null (unavoidable overhead)	2.6 sec	NA
Original	4.34	1.74
Static specialization	2.84	0.24
Dynamic specialization	3.35	0.75

Georgia Tech

Example 3: Signal Delivery

- Signals
  - Asynch. communication among processes
  - System call: `kill(pid, sig)`
  - OS delivers signal and invokes handler at receiving process
- Common execution patterns
  - Repeated use of same signal to same process
  - Locality exists, but sessions are not explicit

Georgia Tech

Signal Delivery in Linux

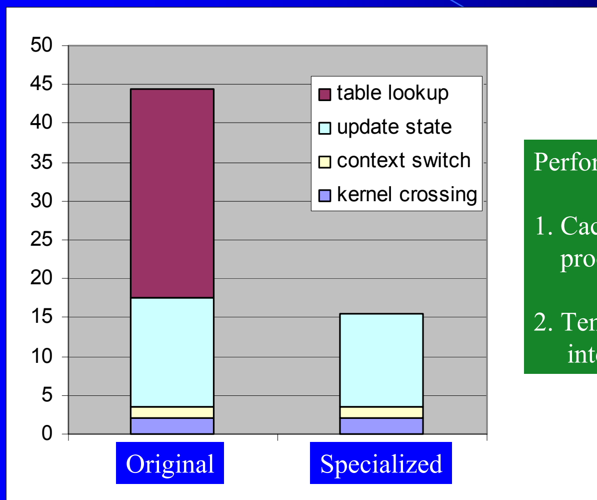
```
kill (pid, sig)
```

- sys_kill (int pid, int sig) -- enter kernel
- ↓ kill_proc (int pid, int sign, int priv) -- search process table for pid
- ↓ send_sig (int sig, task_struct * p, int priv) -- check for permission
- ↓ generate (int sig, task_struct * p) -- deliver the signal

Specializing Signal Delivery

- Problem - couldn’t recognize sessions:
  - Cache last signal sent, and destination
  - First call: test for repeat, invoke generic code
  - Second call: detect repeat, enable specialization, invoke specialized code
  - Subsequent calls: invoke specialized code if it’s a repeat, else disable specialized code
- Optimistic specialization
  - Assumes no changes to process state
  - Guards to detect updates to task_struct

Performance Results for Signals



Performance effects:

1. Caching reduces cost of process table lookup
2. Tempo reduces cost of interpreting of process state

Georgia Tech

Advantages of Specialization

- Several opportunities
  - Communication links: TCP, Shared memory, Function, …
  - Wire formats: XML, XDR, Raw structure, …
- Systematic code transformation
  - Explicitly identified invariants
  - Guarding of invariants guarantees correctness

Georgia Tech

Discussion

- Methodical improvement of system software code (with some correctness guarantees)
- Application to production code?
  - HP-UX file system (SOSP'95)
  - TCP/IP

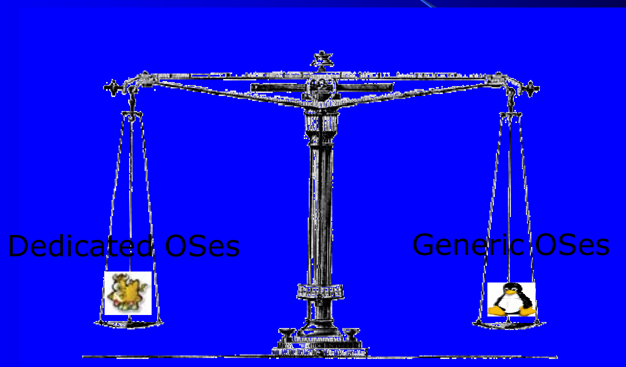
Georgia Tech

Specialization in RTES

- Code Customization
- Remote Customization Infrastructure
- Virtualization of memory
- Case study: TCP/IP
- Performance Evaluation
- Bhatia et al [LCN'04] Best Paper, Bhatia et al [EmSoft'04]

Georgia Tech

Dedicated Vs. Generic OSes

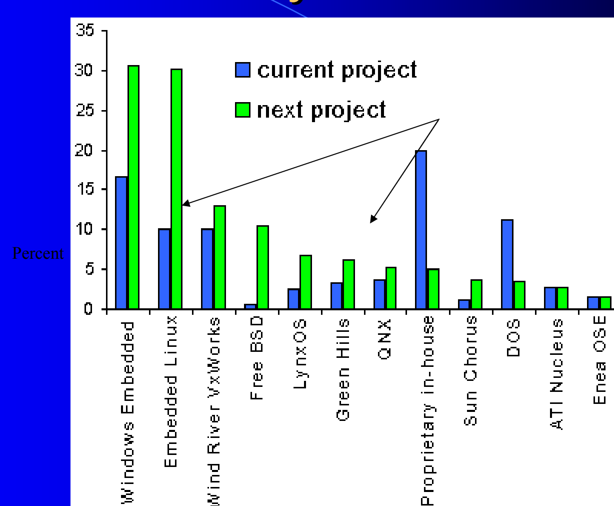


+ Deeply customized, compact, fast, well-suited
- Lack of support for standards

+ Support for standards
- Generic, coarse grained abstractions

Georgia Tech

Industry Trends



Embedded Systems Developers Survey, 2002
Source: Evans Data Corp.

Georgia Tech

Generic abstractions

Coarse grained building blocks

```
if (poll (listen_pfds, n, -1) > 1) {
    foreach(pfd, listen_pfds) {
        if (hi_r(pfd->revents))
            queue(
                accept(fd, addr, addr_len));
        }
}
```

Performance:
3000+ conn/sec.

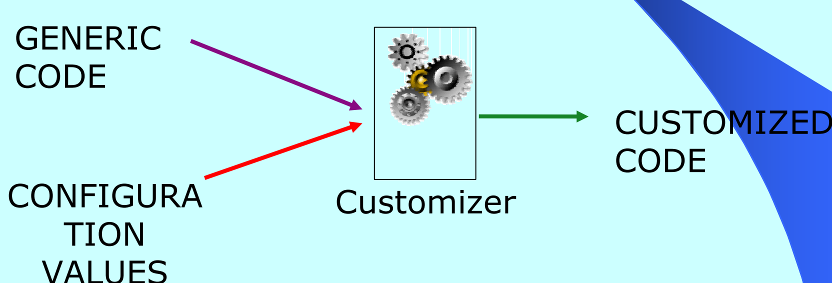
Concrete Operations

```
foreach(sock, my_sockets) {
    if (sock->sk->accept_queue) {
        sock->ops->accept(sock, new_sock,
                          O_NONBLOCK);
    }
}
```

Performance:
7000+ conn/sec.

Overheads: memory transfers, context switches, sanity checks, data structures

Code Customization



Code Customization

```
int tcp_mini_sendmsg (struct sock *sk, void *msg, int size)
{
    int tocopy=0, copied=0;
    while (tocopy = (size < sk->tcp->mss) ? size : mss) {
        if (copied = (free_space (sk->write_queue.prev.space))) {
            if (copied > tocopy) copied = tocopy;
            add_data (sk->write_queue.prev, msg, copied);
            size = size - copied; msg = msg + copied;
        }
        else {
            struct skbuff *skb = alloc_new_skb();
            add_data(skb, msg, tocopy);
            size = size - tocopy; msg = msg + tocopy;
            entail (sk->write_queue, skb);
        }
    }
return size;
}
```

Georgia Tech

Customization Context

```
int tcp_mini_sendmsg (struct sock *sk, void *msg, int size)
{
    int tocopy=0, copied=0;
    while (tocopy = (size < sk->tcp->mss) ? size : mss) {
        if (copied = (free_space (sk->write_queue.prev.space))) {
            if (copied > tocopy) copied = tocopy;
            add_data (sk->write_queue.prev, msg, copied);
            size = size - copied; msg = msg + copied;
        }
        else {
            struct skbuff *skb = alloc_new_skb();
            add_data(skb, msg, tocopy);
            size = size - tocopy; msg = msg + tocopy;
            entail (sk->write_queue, skb);
        }
    }
return size;
}
```

Georgia Tech

Binding Time Analysis

```
int tcp_mini_sendmsg (struct sock *sk, void *msg, int size)
{
    int tocopy=0, copied=0;
    while (tocopy = (size < sk->tcp->mss) ? size : mss) {
       if (copied = (free_space (sk->write_queue.prev.space))) {
           if (copied > tocopy) copied = tocopy;
           add_data (sk->write_queue.prev, msg, copied);
           size = size - copied; msg = msg + copied;
       }
       else {
           struct skbuff *skb = alloc_new_skb();
           add_data(skb, msg, tocopy);
           size = size - tocopy; msg = msg + tocopy;
           entail (sk->write_queue, skb);
       }
    }
return size;
}
```


Customized code

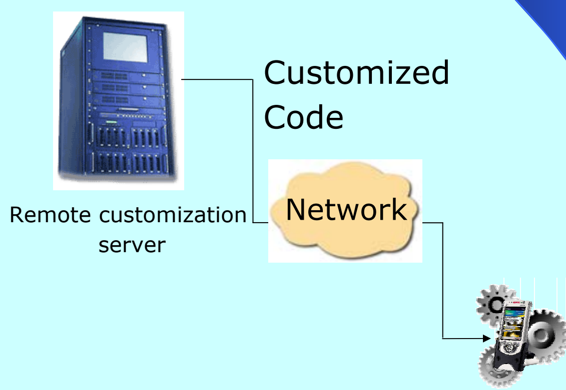
Customization context:

size=1400
sk={…}

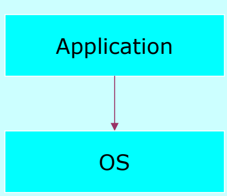
Customized code:

```
int tcp_mini_sendmsg (void *msg)
{
    struct skbuff *skb = alloc_new_skb();
    add_data(skb, msg, 1400);
    entail (sk->write_queue, skb);
    return 0;
}
```

Remote Customization

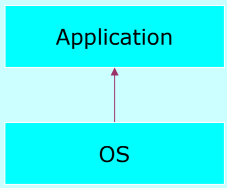


How it's used




```
token = do_customize_send(…);

for (i=0;i<100000;i++) {
  customized_send (token, buffer);
}
```

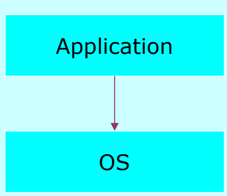
How it’s used




```
token = do_customize_send(...);

for (i=0;i<100000;i++) {
  customized_send (token, buffer);
}
```

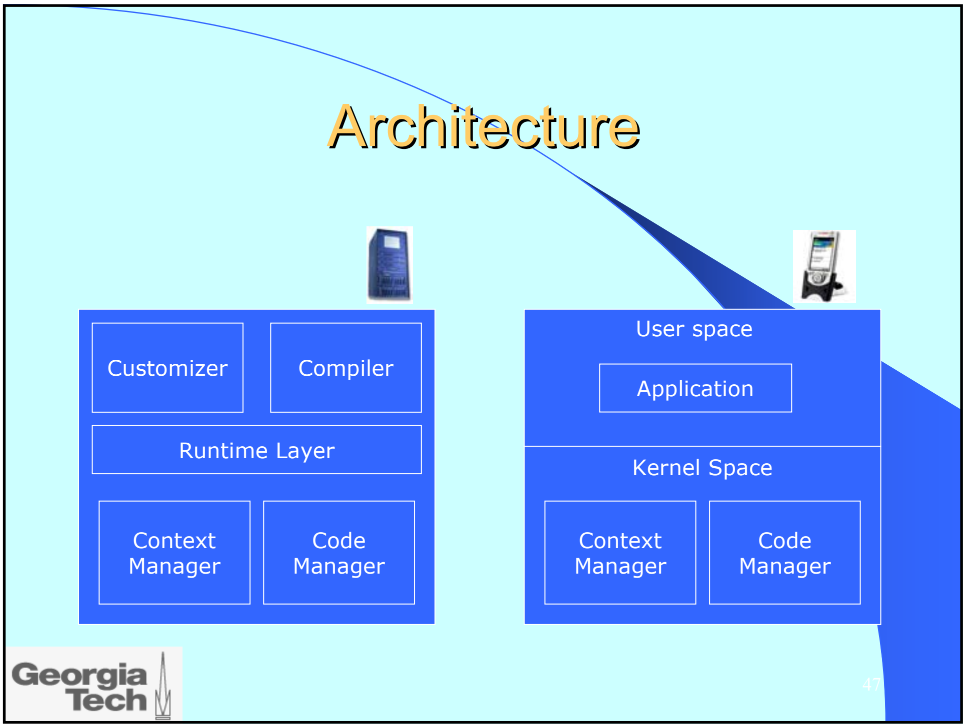
Georgia Tech

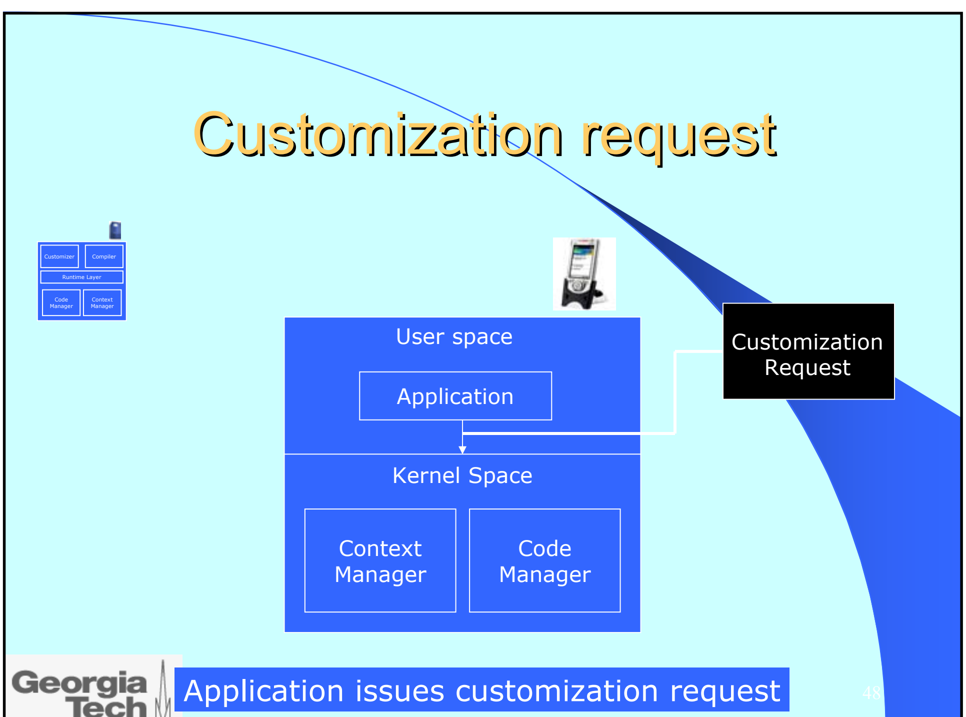
How it’s used

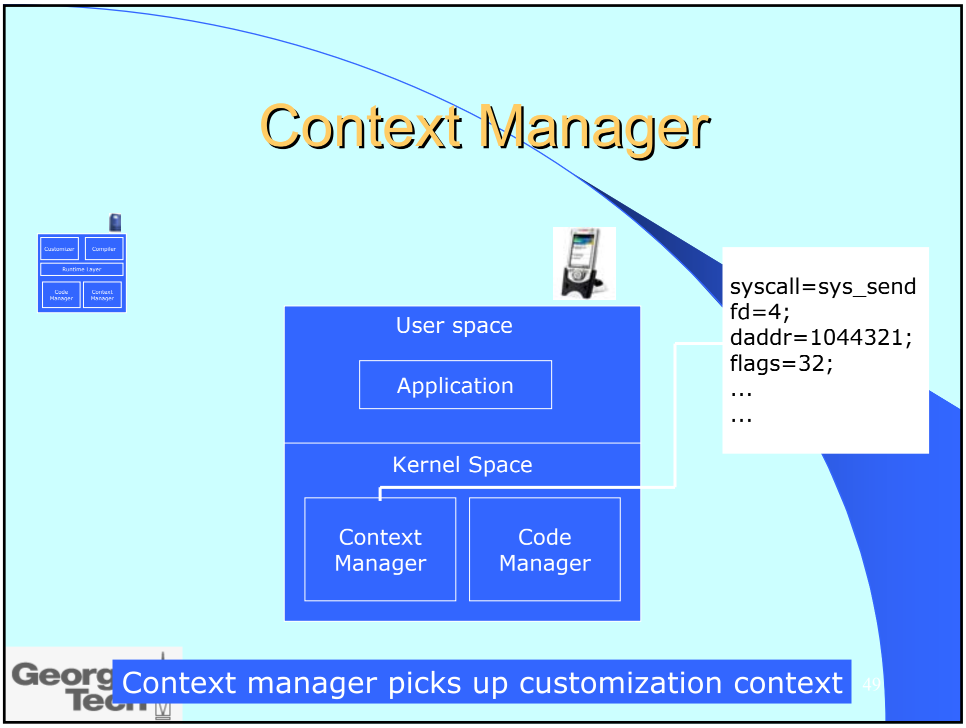


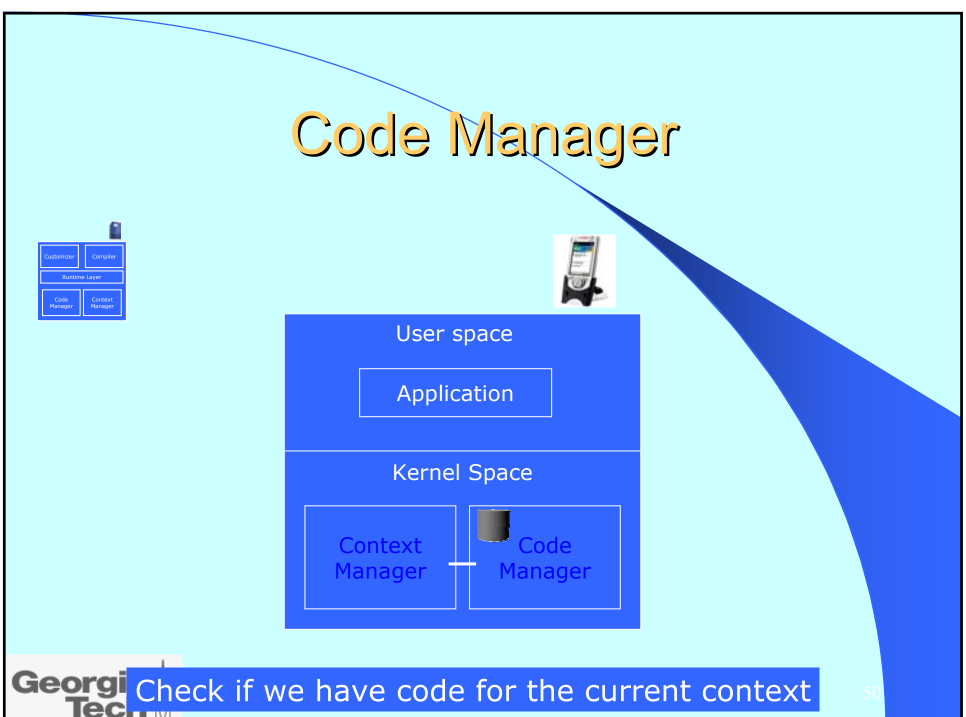

```
token = do_customize_send(...);

for (i=0;i<100000;i++) {
  customized_send (token, buffer);
}
```

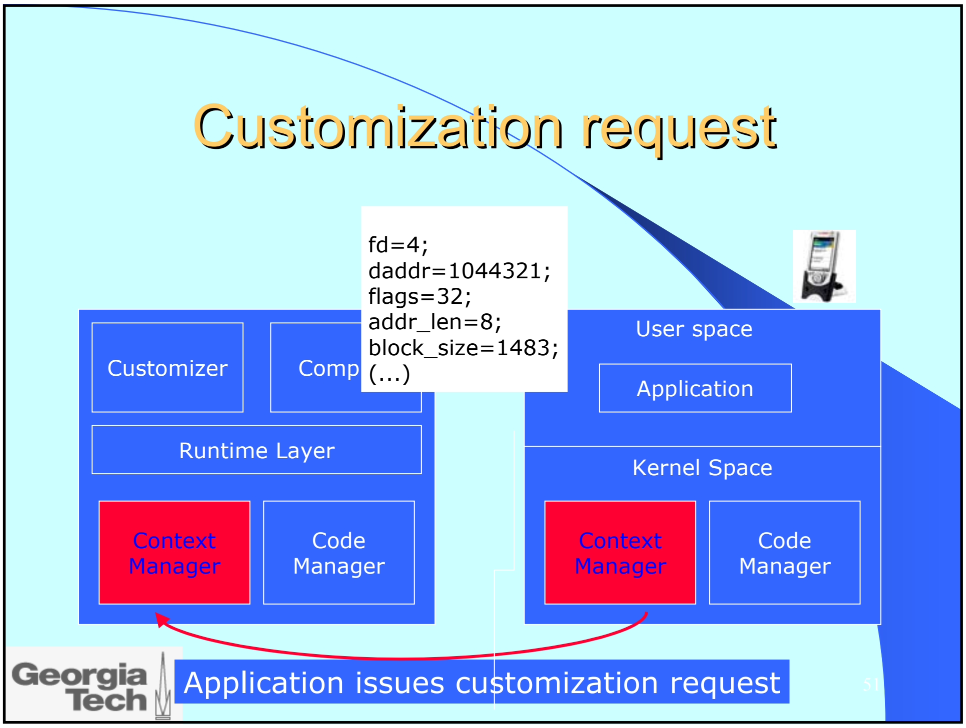
Georgia Tech

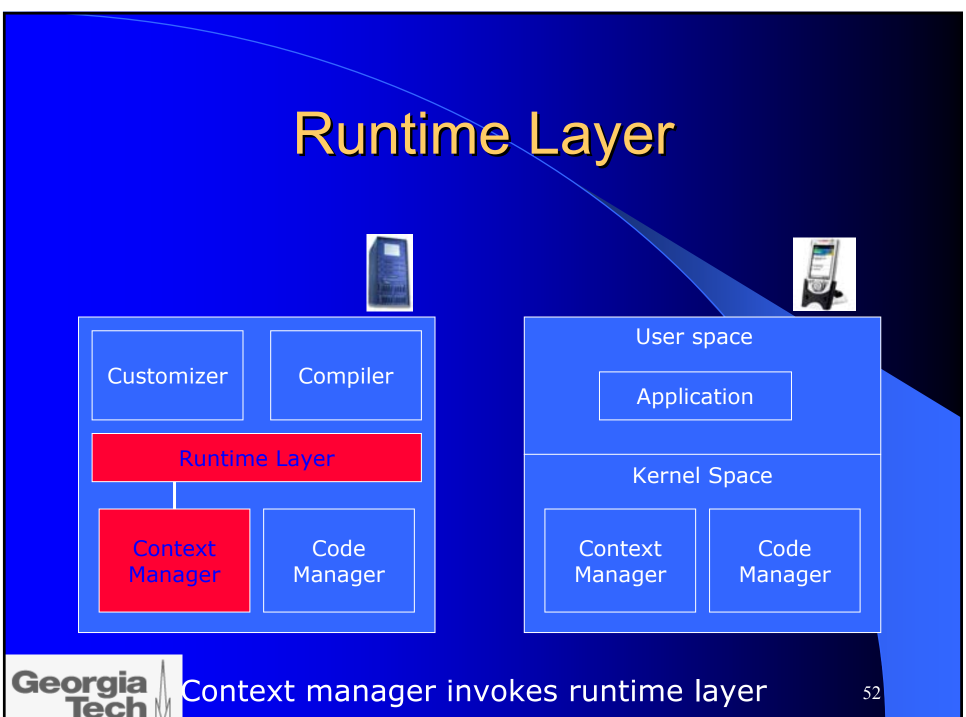

Architecture
Customizer
Compiler
Runtime Layer
Context Manager
Code Manager
User space
Application
Kernel Space
Context Manager
Code Manager
Georgia Tech
47

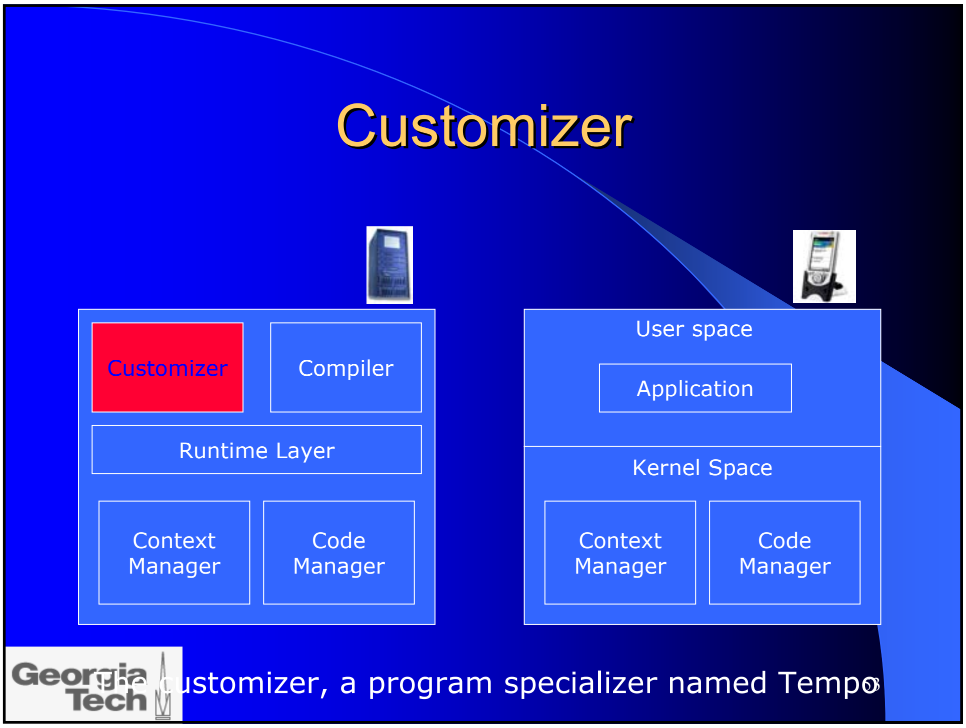

Customization request
Customizer
Compiler
Runtime Layer
Code Manager
Context Manager
User space
Application
Kernel Space
Context Manager
Code Manager
Customization Request
Georgia Tech
Application issues customization request
48

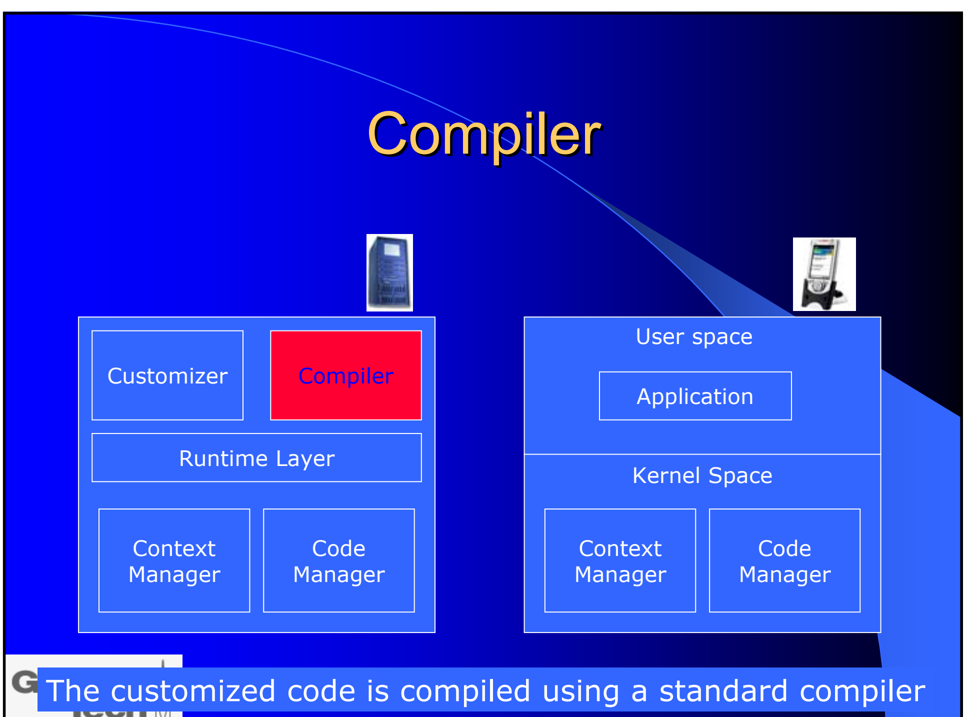

Context Manager
Customizer
Compiler
Runtime Layer
Code Manager
Context Manager
User space
Application
Kernel Space
Context Manager
Code Manager
syscall=sys_send
fd=4;
daddr=1044321;
flags=32;
...
...
Context manager picks up customization context
49

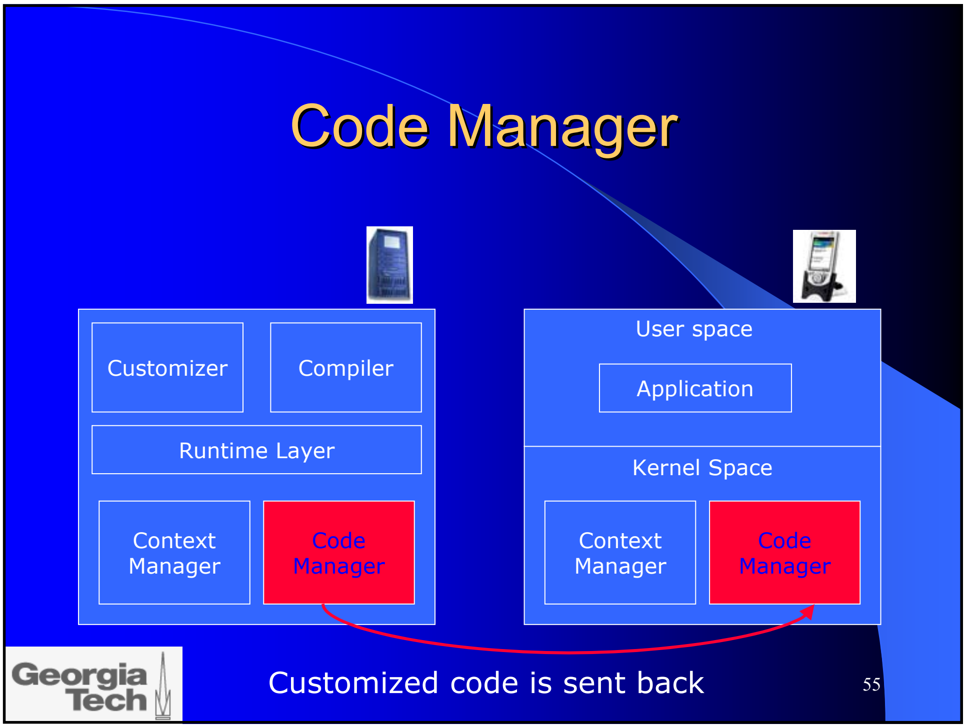

Code Manager
Customizer
Compiler
Runtime Layer
Code Manager
Context Manager
User space
Application
Kernel Space
Context Manager
Code Manager
Check if we have code for the current context
50


Customization request
fd=4;
daddr=1044321;
flags=32;
addr_len=8;
block_size=1483;
(…)
Customizer
Comp
Runtime Layer
Context Manager
Code Manager
User space
Application
Kernel Space
Context Manager
Code Manager
Georgia Tech
Application issues customization request
51

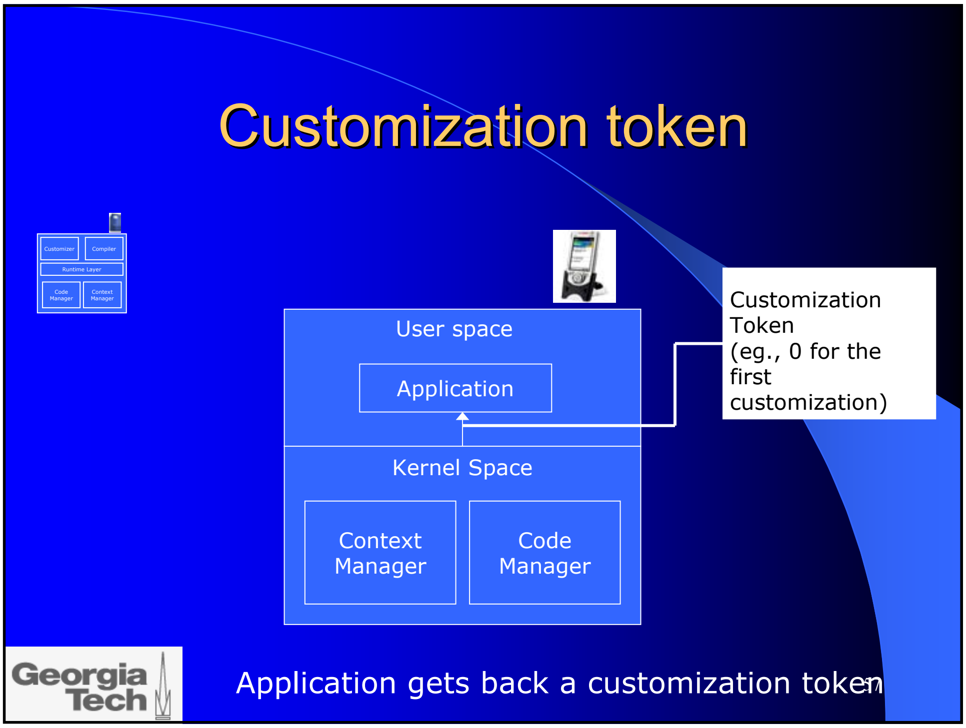

Runtime Layer
Customizer
Compiler
Runtime Layer
Context Manager
Code Manager
User space
Application
Kernel Space
Context Manager
Code Manager
Georgia Tech
Context manager invokes runtime layer
52

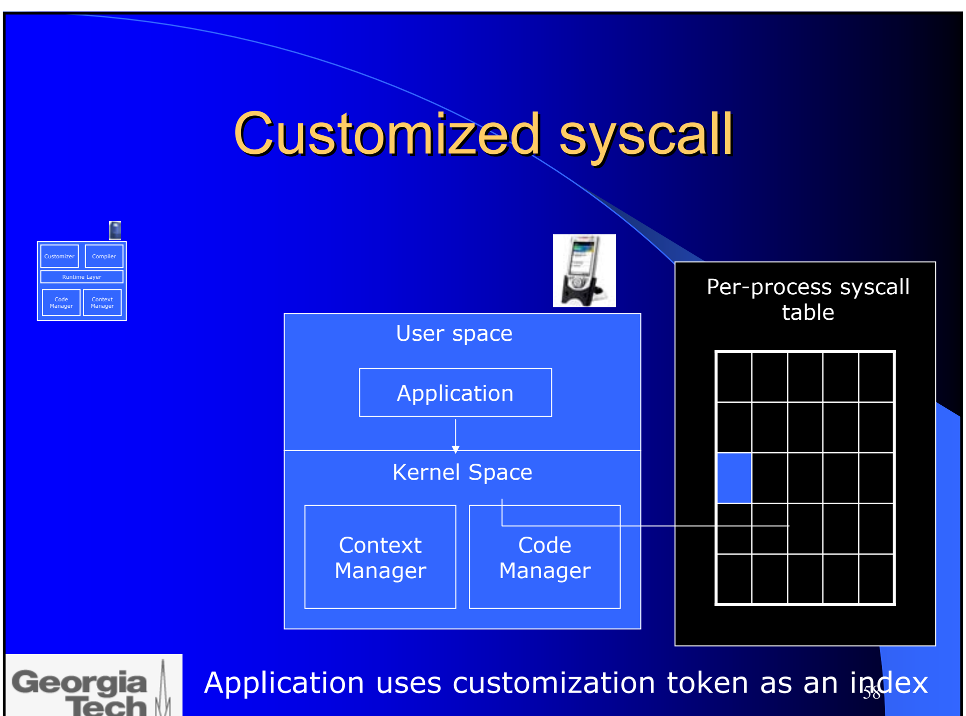

Customizer
Customizer
Compiler
Runtime Layer
Context Manager
Code Manager
User space
Application
Kernel Space
Context Manager
Code Manager
ustomizer, a program specializer named Tempo


Compiler
Customizer
Compiler
Runtime Layer
Context Manager
Code Manager
User space
Application
Kernel Space
Context Manager
Code Manager
The customized code is compiled using a standard compiler

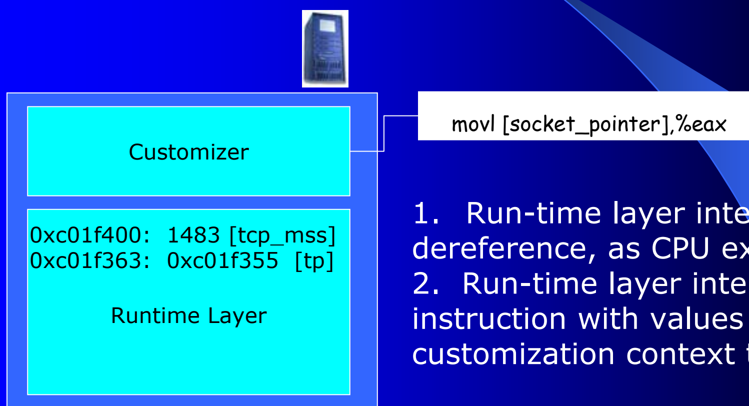

Code Manager
Customizer
Compiler
Runtime Layer
Context Manager
Code Manager
User space
Application
Kernel Space
Context Manager
Code Manager
Georgia Tech
Customized code is sent back
55


Customization token
Customizer
Compiler
Runtime Layer
Context Manager
Code Manager
User space
Application
Kernel Space
Context Manager
Code Manager
Georgia Tech
56


Customization token
Customizer
Compiler
Runtime Layer
Code Manager
Context Manager
User space
Application
Kernel Space
Context Manager
Code Manager
Customization Token (eg., 0 for the first customization)
Georgia Tech
Application gets back a customization token
57


Customized syscall
Customizer
Compiler
Runtime Layer
Code Manager
Context Manager
User space
Application
Kernel Space
Context Manager
Code Manager
Per-process syscall table
Georgia Tech
Application uses customization token as an index
58

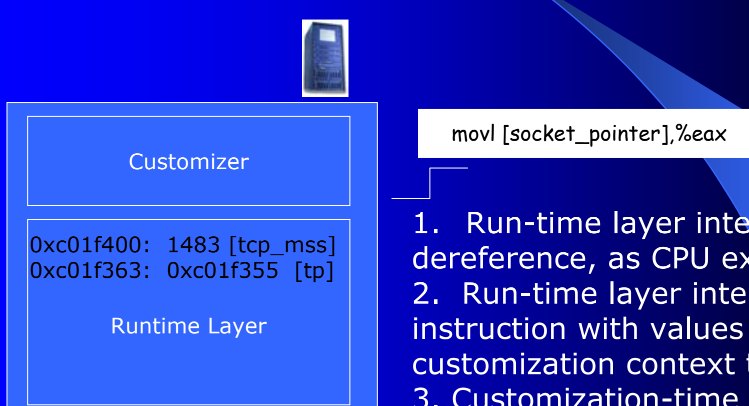
Access to client side memory



1. Run-time layer intercepts dereference, as CPU exception.
2. Run-time layer interprets instruction with values in customization context table.

Georgia Tech

Access to client side memory



1. Run-time layer intercepts dereference, as CPU exception.
2. Run-time layer interprets instruction with values in customization context table.
3. Customization-time functions Executed on client

Georgia Tech

Customization Opportunities

- Mappings between socket descriptors and low level structures
- Routing decisions for every send()
- Socket options interpreted
- Dependencies on buffer sizes

Georgia Tech

Optimizations performed

- Straight-lining code by removing branches
- Constant value propagation
- Loop unrolling
- Function inlining
- Etc.

Georgia Tech

Results: Improvements in performance and code size

- Execution time decreased by ~26%
- Code size decreased by a factor of >15
- Throughput improvements:
  - UDP - PIII: 13% 486: 27% iPAQ: 18%
  - TCP - PIII: 10% 486: 23% iPAQ: 13%

Georgia Tech

Specialization Overhead

- Overhead = customization time + network transfer time (< 1 sec)
- Bottleneck => execution of customizer + compiler
- Eventually, bottleneck => network transfer time
- When so, bound = (1 + X)*RTT

Georgia Tech

Summary

- Problem: Services in generic OSes are slow and bloated
- Solution: Dynamic/remote code customization
- Assessment: Exec time… -25%, throughput… +20%, code size… -15x

Georgia Tech

Discussion

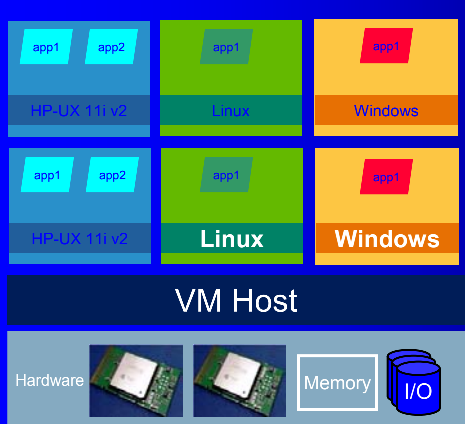
- Need generic platform (can’t start from scratch for each project)
- Need to customize for many projects
  - Apply principle approaches (e.g., specialization)
  - Recognize the difficulties

Georgia Tech

Virtualization

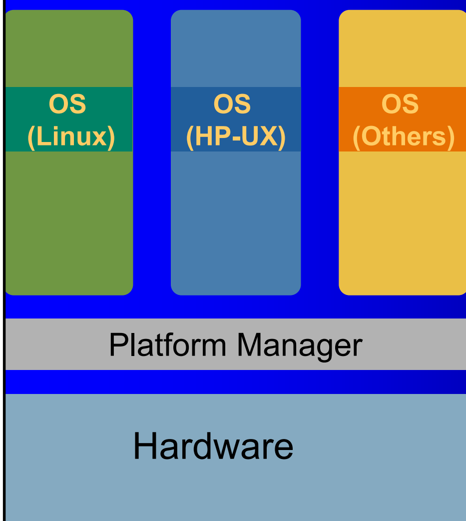
- Some examples of virtualized systems
- Many choices of virtualization
- Specialization of virtualized systems

HP Integrity Virtual Machines



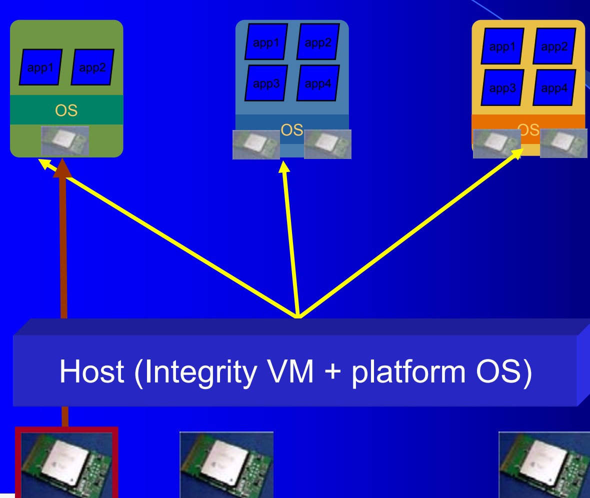
- Sub CPU virtual machines with shared I/O
- Resource guarantees as low
  as 5% CPU granularity
- OS fault and security isolation
- Supports all (current and future) HP Integrity servers
- Designed for multi OS
  - HP-UX 11i guest
  - Linux guest
  - Windows guest
  - OpenVMS guests in future

HP Integrity Virtual Machines

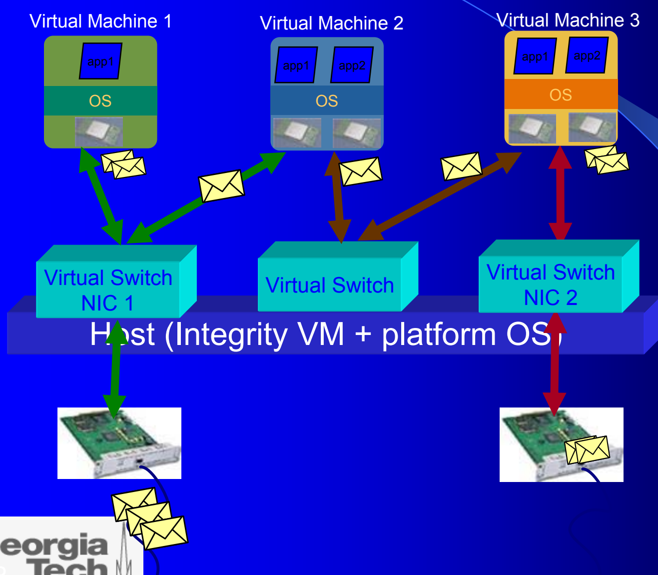


- HP Integrity VM
  - Multi-OS (HP-UX, Linux, …)
  - Sub-CPU granularity
  - I/O Device Sharing
  - Memory Isolation
- Benefits
  - Capacity on Demand
  - Software Development and Testing
  - Rolling Update

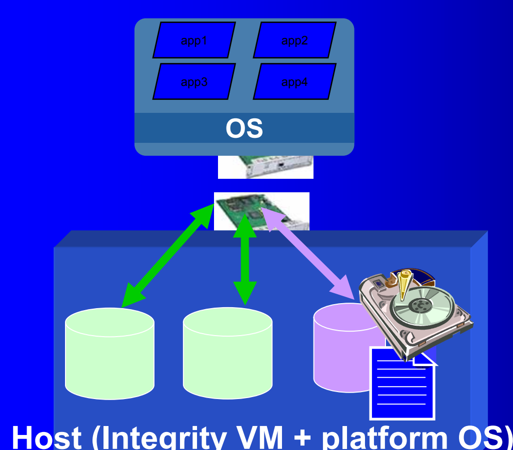
Dynamic CPU Allocation

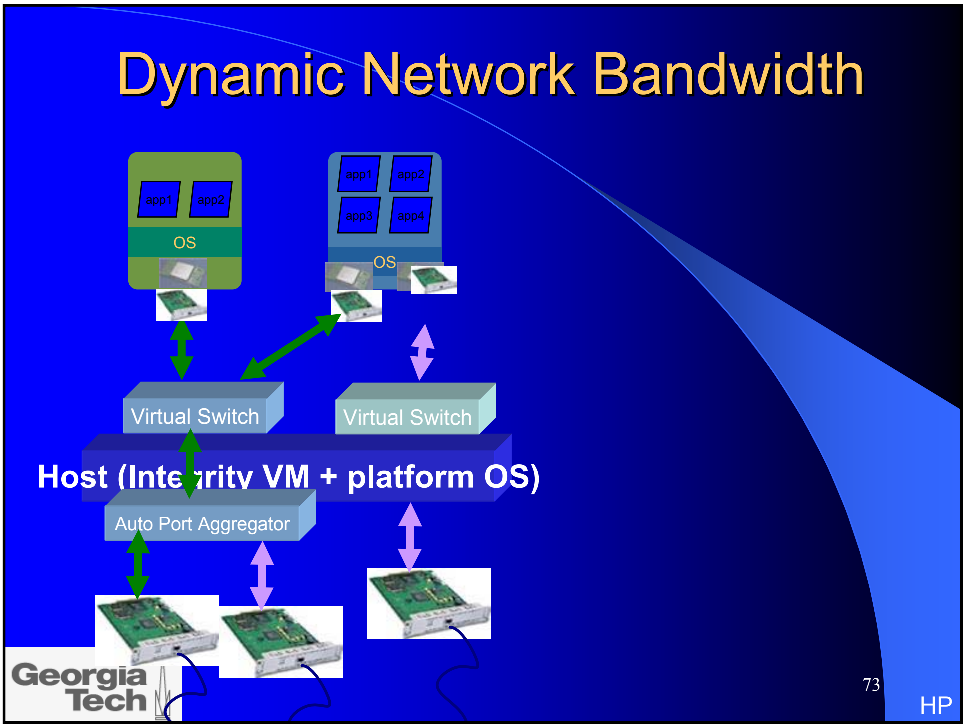


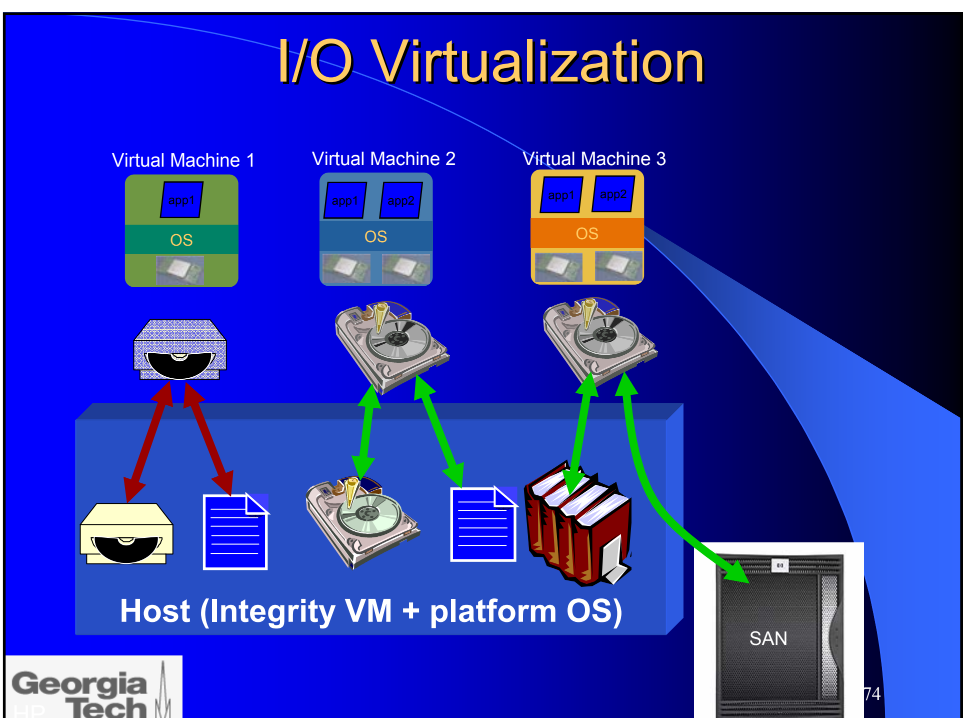
Dynamic Networking Sharing

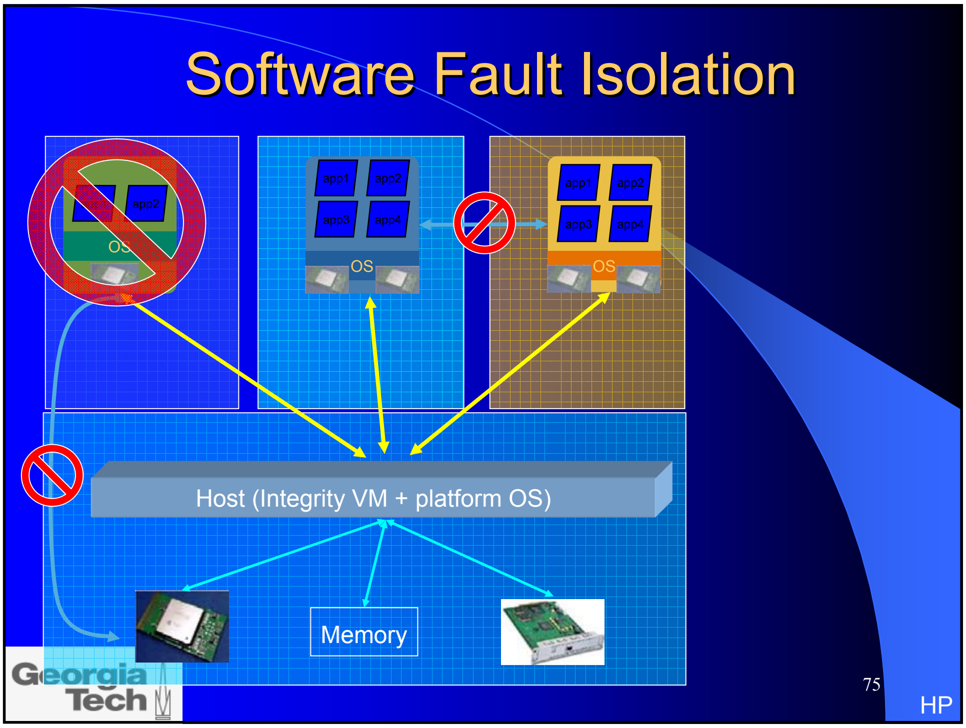


Dynamic Storage Capacity

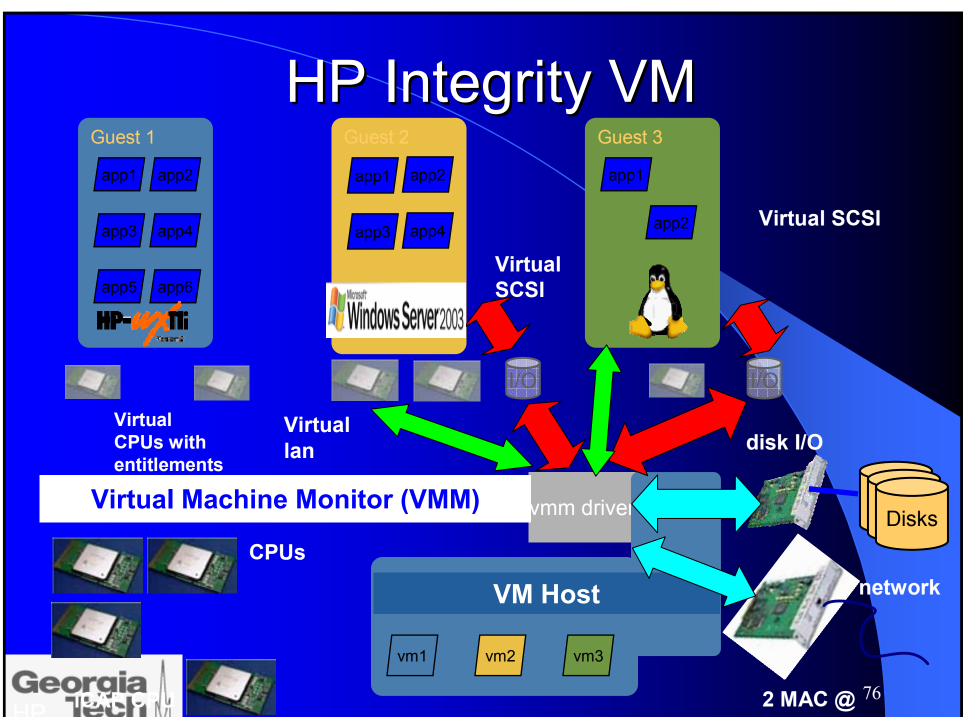



Dynamic Network Bandwidth
app1
app2
OS
app1
app2
app3
app4
OS
Virtual Switch
Virtual Switch
Host (Integrity VM + platform OS)
Auto Port Aggregator
Georgia Tech
73
HP


I/O Virtualization
Virtual Machine 1
Virtual Machine 2
Virtual Machine 3
app1
OS
app1
app2
OS
app1
app2
OS
Host (Integrity VM + platform OS)
SAN
Georgia Tech
74



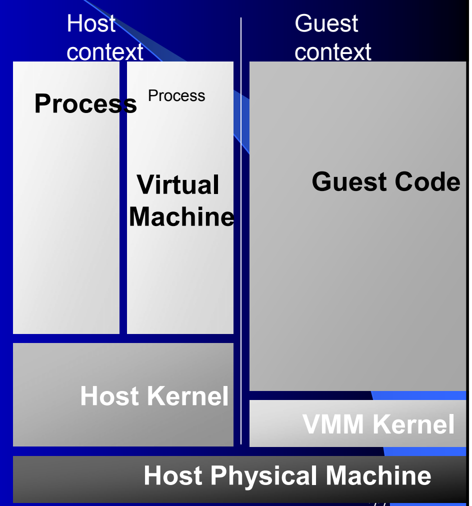
Software Fault Isolation
app1
app2
app3
app4
OS
Host (Integrity VM + platform OS)
Memory
Georgia Tech
75
HP



HP Integrity VM
Guest 1
app1
app2
app3
app4
app5
app6
Guest 2
Windows Server 2003
Guest 3
Virtual SCSI
Virtual SCSI
I/O
Virtual CPUs with entitlements
Virtual lan
disk I/O
Virtual Machine Monitor (VMM)
vmm driver
Disks
CPUs
VM Host
network
vm1
vm2
vm3
2 MAC @
76
Georgia Tech

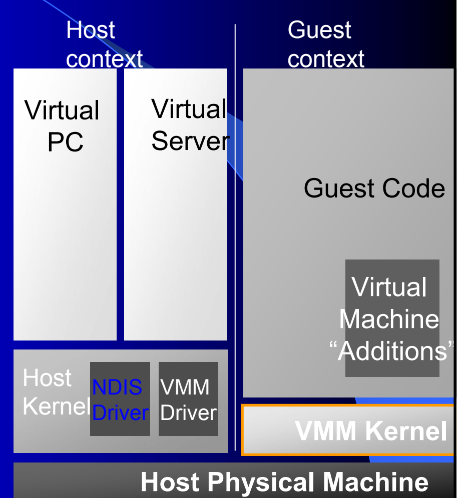
Hosted VM Model

- Windows as "host"
  - Resources for each VM alloc. from host
  - All I/O with external devices is performed through the host
- "Guest" runs within a separate context
  - Independent address space
  - Specialized "VMM" kernel



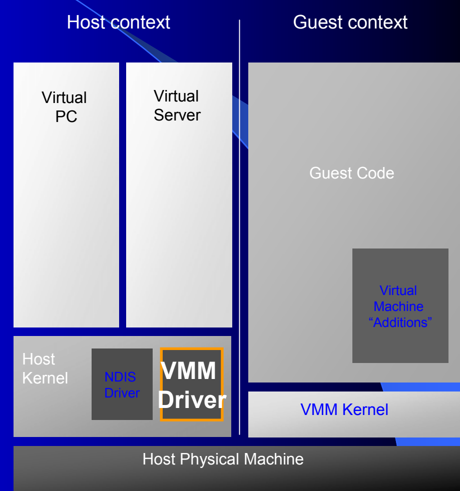
VMM Kernel

- Thin layer, all in assembly
- Code executed at ring-0
- Exception handling
- External Interrupt pass-through
- Page table maintenance
- Located within a 32MB area known as the "VMM work area"
- Work area is relocatable
- One VMM instance per virtual processor



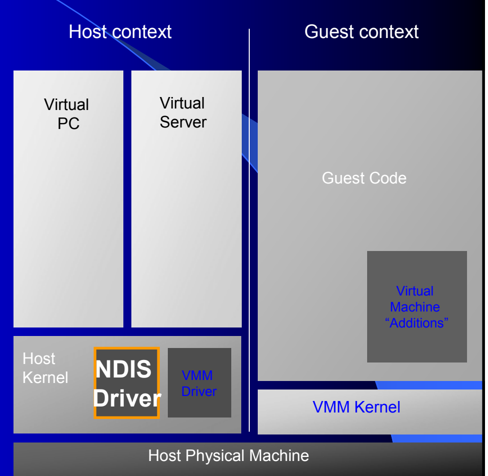
VMM Driver

- Provides kernel-level VM-related services
  - CreateVirtualMachine
  - CreateVirtualProcessor
  - ExecuteVirtualProcessor
- Implements context switching mechanism between the host and guest contexts
- Loads and bootstraps the VMM kernel
- Security: repackaging the VMM kernel code into the VMM driver



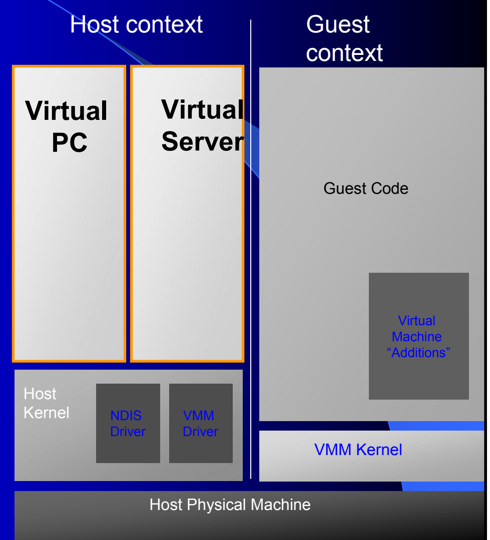
NDIS Filter Driver

- Allows VM to send and receive Ethernet packets via physical Ethernet hardware
- Spoofs unique MAC addresses for virtual NICs
- Injects packets into host Ethernet stack for guest-to-hostnetworking



Virtual PC/Virtual Server

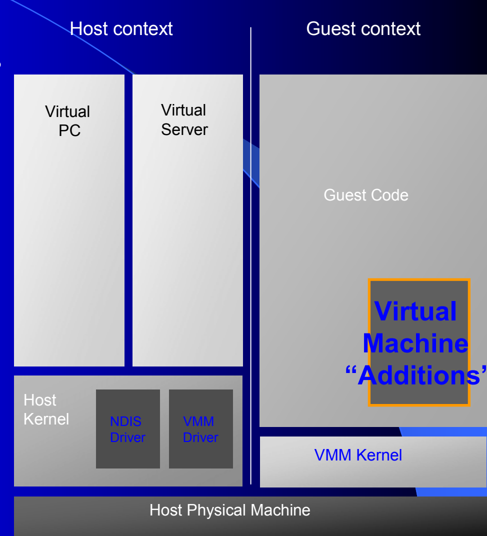
- Device emulation modules
- Resource allocation
- VM configuration creation & editing
- VM control (start, stop, pause, save)
- Scripting APIs
- User interaction
- Host side of guest/host integration features



Microsoft

VM "Additions"

- Collection of components installed within the guest environment by the user
- Implement optimizations
  - Video
  - SCSI
  - Networking (in the future)
  - Guest kernel patches
- Implement guest half of guest/host integration features
  - Clipboard sharing
  - File drag and drop
  - Arbitrary video resizing



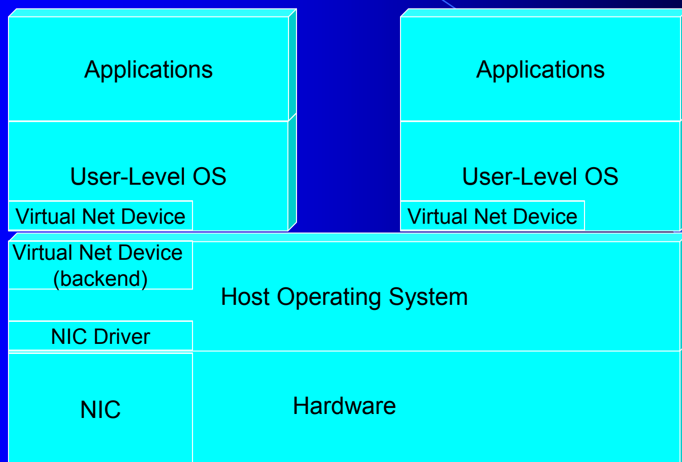
Specialization in VMs

- Efficient Packet Processing in User-Level OS: Study of UML
- User-level OS: User-Mode Linux (UML)

User-Level OS

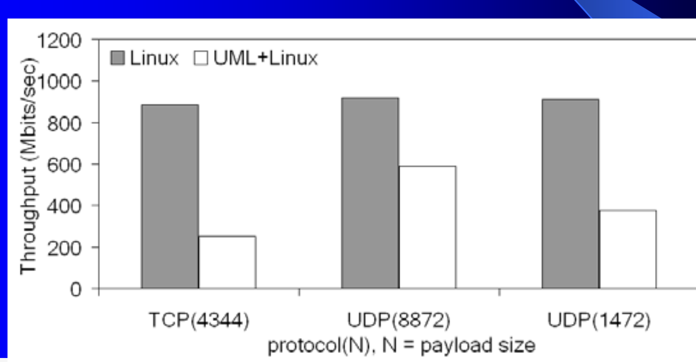
- One form of system virtualization
- A ULOS = A process in host kernel
- Pros
  - Higher resource utilization
  - Fault and security isolation
  - Easy maintenance, installation, diagnosis
- Cons
  - Performance

ULOS Architecture



Network Performance of ULOS

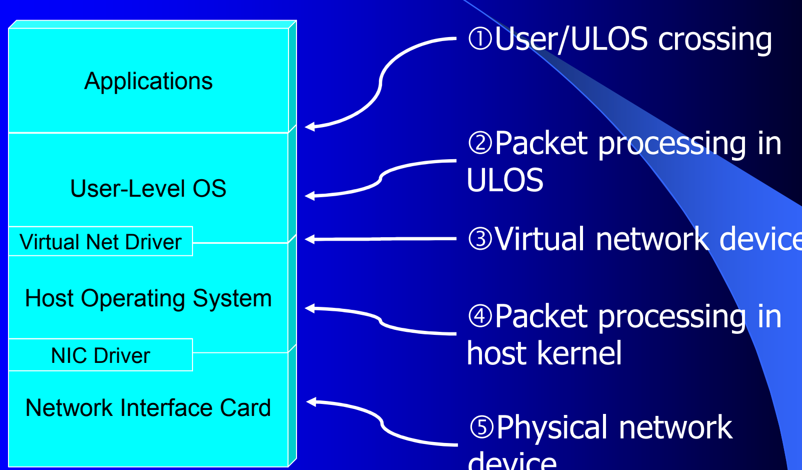
- Maximum network throughput comparison



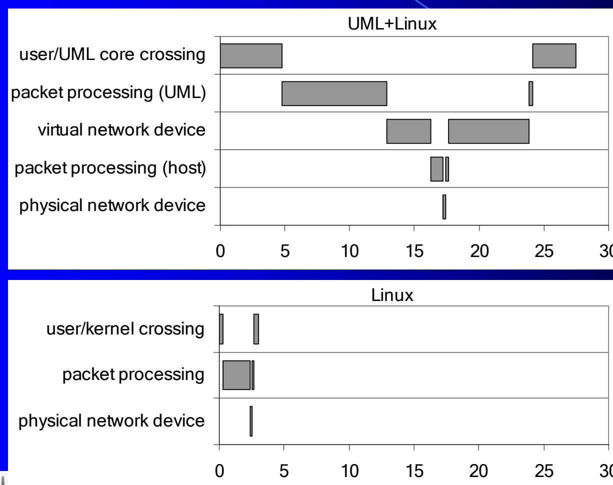
Why Slow?

- Privilege management
  - ULOS reuses the native kernel code
  - Impedance mismatch
- Memory management
  - Extra layers induce more copies
  - Virtual address translation
- Additional software instructions
  - More layers mean more instructions

Layering Analysis



Comparing with Linux



Georgia Tech

Five Optimization Techniques

- User-level Signal Masking
- Aggregated System Calls
- Address Translation Cache
- Shared Socket Buffer
- Specialized Network Stack

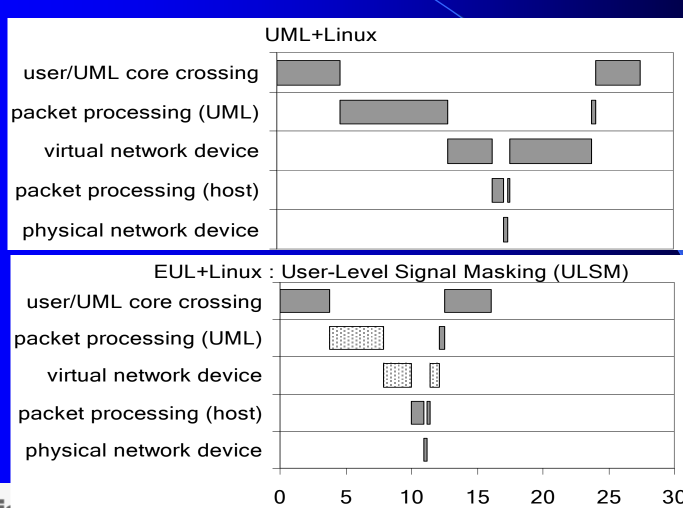
- EUL: Enhanced User-mode Linux

Georgia Tech

User-Level Signal Masking

- Interrupt in host kernel = process signal in ULOS
- Disabling interrupt = masking signals
- Masking signals using system calls is expensive
- Solution: implement signal masking in user-level

ULSM Effect



Aggregated System Calls

- To emulate system call services in ULOS
- ULOS core intercepts syscalls from an app
  - By using ptrace(), exit()
- Multiple calls of ptrace(), exit()
  - Passing and returning arguments, resuming and waiting
  - Cause multiple boundary crossings
- Solution: aggregate multiple ptrace()s
  - 30% reduction to ULOS system call invocation

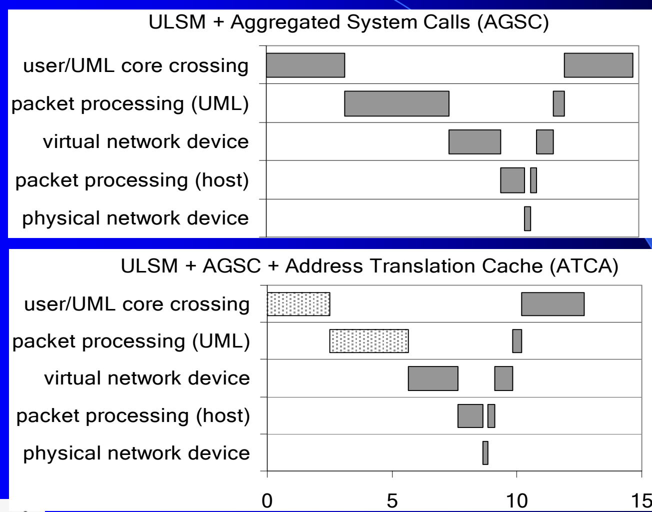
Georgia Tech

Address Translation Cache

- Three address space – application, ULOS core, host kernel
- Address translation from app to ULOS is implemented in software
- Solution: TLB-like cache to speed up the address translation

Georgia Tech

ATCA Effect



Shared Socket Buffer

- Three different address spaces
  - Can have up to two copies
- One additional copy compared to native OS
- Solution: allocate shared memory between ULOS and host kernel
  - No copy from ULOS core to host kernel
  - Reduced Up to 40% virtual NIC latency

Specialized Network Stack

- To reduce CPU instructions
- Specialize networks stack using quasi-invariant
  - IP addresses, port numbers, sock options, ...
- Up to 13% reduced packet processing time

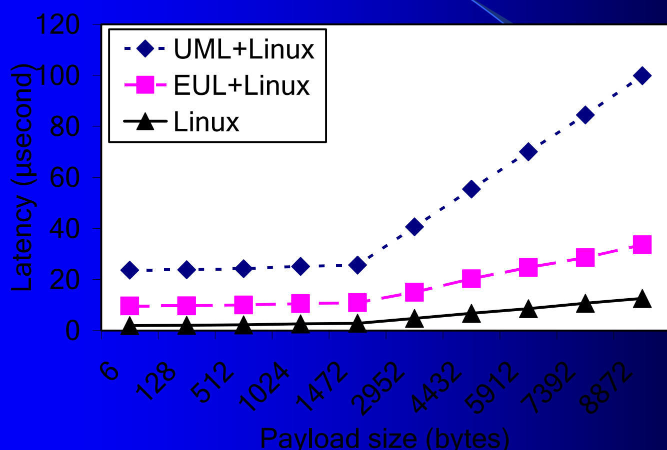
Georgia Tech

Evaluation

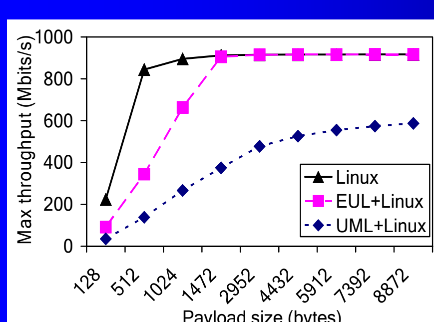
- Experimental Setup
  - 1GB network
  - Pentium4 3GHz, 512KB L2 Cache, 1GB mem
  - Ttcp for measuring network throughput
  - Linux, UML+Linux, EUL+Linux, XenLinux+Xen
- Packet processing latency, max throughput
- Web server benchmark (httperf)

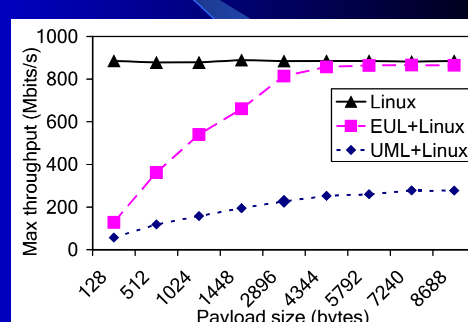
Georgia Tech

Packet Processing Latency

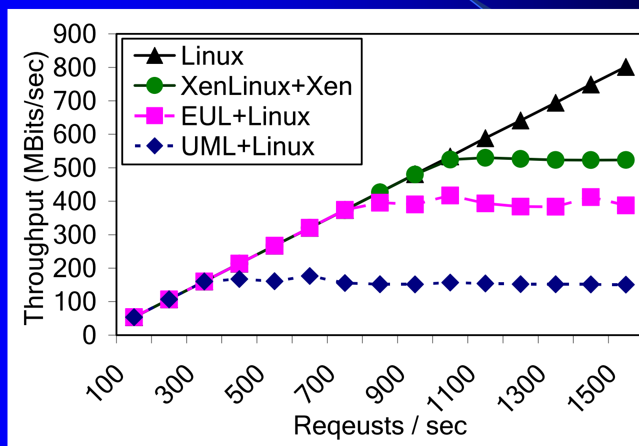


Maximum Throughput





Web Server Throughput



Summary

- ULOS: a good use of virtualization
  - But, poor performance
- Optimization techniques can help
  - Comparable network throughput to native Linux
  - Reduced latency by more than half
- Fast ULOS is possible and feasible

Discussion

- Principled optimization of system code for virtual environments
- How to apply principled code manipulation in general for RTES?

Georgia Tech

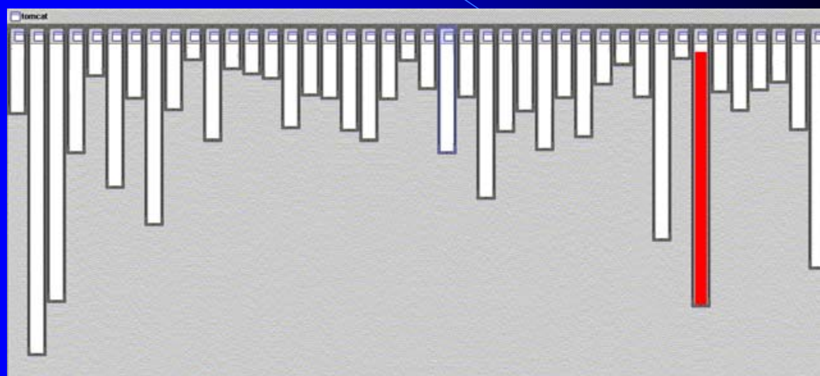
Quick Intro to AOP

- AOP – Aspect Oriented Programming
  - Kiczales et al, Xerox PARC
- AOP is a method to address serious problems in large programs
  - Tangled code
- Slide credit: tutorials from AspectJ.org

Georgia Tech

good modularity

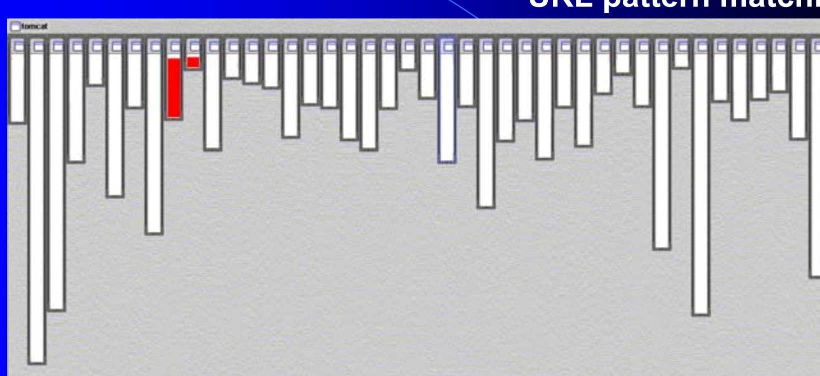
XML parsing



- XML parsing in org.apache.tomcat
  - red shows relevant lines of code
  - nicely fits in one box

good modularity

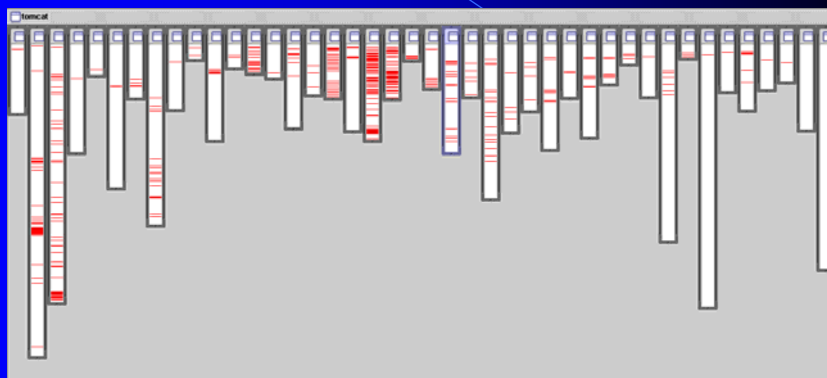
URL pattern matching



- URL pattern matching in org.apache.tomcat
  - red shows relevant lines of code
  - nicely fits in two boxes (using inheritance)

problems like…

logging is not modularized



- where is logging in org.apache.tomcat
  - red shows lines of code that handle logging
  - not in just one place
  - not even in a small number of places

Georgia Tech

problems like…

session expiration is not modularized

Georgia Tech

aspectj.org

AOP idea

- Crosscutting is inherent in complex systems
  - have a clear purpose
  - have a natural structure
    - defined set of methods, module boundary crossings, points of resource utilization, lines of dataflow…
- Capture the structure of crosscutting concerns explicitly...
  - in a modular way
  - with linguistic and tool support
- Aspects are
  - well-modularized crosscutting concerns

Georgia Tech

AspectJ Basics

- 1 overlay onto Java
  - dynamic join points
    - "points in the execution" of Java programs
- 4 small additions to Java
  - pointcuts
    - pick out join points and values at those points
      - primitive, user-defined pointcuts
  - advice
    - additional action to take at join points in a pointcut
  - inter-class declarations (aka "open classes")
  - aspect
    - a modular unit of crosscutting behavior
      - comprised of advice, inter-class, pointcut, field,constructor and method declarations

Georgia Tech

AOP Summary

- AOP advantages
  - same benefits of good modularity
  - but for crosscutting concerns
  - at design and development-time
- AspectJ language
  - more: advice, inter-type declarations, cflow…
- AspectJ tools
  - crosscutting structure is explicit
  - presented consistently in task-specific views

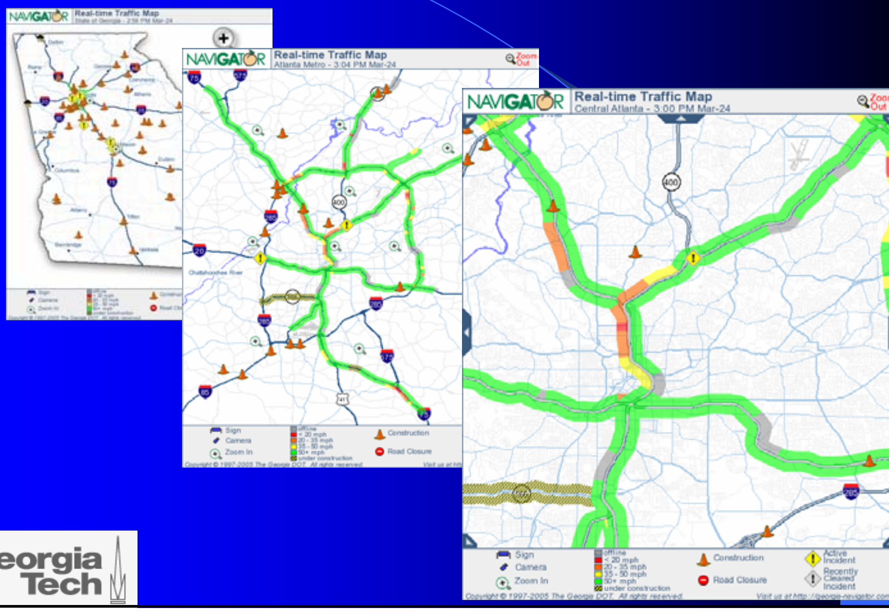
Georgia Tech

Code Generation and Distributed Systems

- Code generation since 1983 (RPC Stub Gen)
- Our focus is source-source translation
- Motivated by constant changes in requirements:
  - Changes due to external forces: merger/acquisitions, standards formulation/adoption, industry evolution
  - Changes due to internal forces: goals, functionality refinement, reuse to solve new (related) problems
- *Generator should evolve with its target domain*

Georgia Tech

Real-Time Information Apps



Georgia Tech

Infopipe Infrastructure

- Information flow applications beyond static RPC calls and web services
  - Continuous data creation, consumption
  - Data is "live"
- Heterogeneous platforms
- Dynamic environments

Georgia Tech

Three Key Challenges

Problem: Provide a toolkit for Infopipes that offers

- Abstraction mapping
- Interoperable heterogeneity
- Flexible customization

It turns out, these are hard to do simultaneously…here's why

Georgia Tech

Generator Requirements

- Extensible input
  - Mutable specifications
- Pliable generator
  - Accommodate mutable specifications
  - Partial implementations of target platforms
- Modular output
  - Customized solutions

Clearwater uses XML/XSLT to achieve E-P-M

Georgia Tech

Extensible Input

- DSLs are restricted to a problem
  - Frequently users ask for extensions
- Requirements/standards may change
- Want the ability to formulate new problems
- Practical utility
  - Specification grammar right the first time?

Georgia Tech

Pliable Generator

- Input: Allows DSL content to change
- Output: Generator can implement partial specifications
- Practical utility
  - Encourages experimentation & research
  - Implies low overhead changes

Georgia Tech

Modular Output

- Supports customization
- "One size fits all" code fits no one
- Orthogonality for aspects of a problem
- Offers hook for other input specifications
- Encourages customization reuse

Georgia Tech

Clearwater Overview

- XML
  - Extensible input
- XSLT
  - Pliable generator
- Combined
  - Modular output

Georgia Tech

XML: Extensible input

- Easily extensible (through new elements)
- No grammar maintenance
- Few syntactic rules

Georgia Tech

Example Extensible Input

```
<datatype name="FloatArray">
  <arg name="SIZE" type="integer"/>
  <arg name="buff" type="string"/>
</datatype>
<pipe name="UAV">
  <subpipes>
    <subpipe name="Sender"
             pipeOf="Sender"/>
    <subpipe name="Receiver"
             pipeOf="Receiver"/>
  </subpipes>
  <connections>
    <connection comm="ECho">
      <from pipe="Sender"
    port="out1"/>
      <to pipe="Receiver"
    port="in1"/>
    </connection>
  </connections>
</pipe>
```

```
<datatype name="FloatArray">
  <arg name="SIZE" type="integer"/>
  <arg name="buff" type="string"/>
</datatype>
<filter name="GREY">
  <in type="ByteArray"/>
  <out type="ByteArray"/>
</filter>
<pipe name="UAV">
    <subpipes>
      <subpipe                         name="Sender"
    pipeOf="Sender"/>
      <subpipe
           name="Receiver"
    pipeOf="Receiver"/>
    </subpipes>
    <connections>
      <connection comm="ECho">
        <from pipe="Sender" port="out1"/>
        <to pipe="Receiver" port="in1"/>
        <use-filters>
          <use-filter name="GREY"/>
        </use-filters>
      </connection>
    </connections>
  </pipe>
```

Georgia Tech

XSLT: Pliable generator - input

- Accommodate extensible input
- XPath is standard
- Programmatic interface with specification
  - Predicates are powerful extraction tools
- Structure-shy interaction model
  - Ignore what you don't understand

Georgia Tech

Example Pliability

```
<datatype name="ppmType">
  <arrayArg name="mag"
         type="char" size="2"/>
  <arg name="width" type="long"/>
  <arg name="height" type="long"/>
  <arg name="maxval" type="long"/>
  <arg name="pictureSize"
    type="integer"/>
  <arrayArg name="picture"
      type="byte"
    size="pictureSize"/>
</datatype>
<pipe  lang="CPP"
    class="ReceivingPipe">
  <apply-aspect
    name="receiver_gpce.xsl"/>
   <ports>
     <inport name="in"
    type="ppmType"/>
  </ports>
</pipe>
```

```
<pipe  lang="CPP"
   class="ReceivingPipe">
    <apply-aspect
   name="receiver_gpce.xsl"/>
   <ports>
     <inport name="in"
   type="ppmType">
       <datatype name="ppmType">
         <arrayArg name="mag"
           type="char" size="2"/>
        <arg name="width"
             type="long"/>
         <arg name="height"
              type="long"/>
         <arg name="maxval"
              type="long"/>
         <arg name="pictureSize"
              type="integer"/>
         <arrayArg name="picture"
              type="byte"
   size="pictureSize"/>
     </datatype>
</inport>
  </ports>
</pipe>
```

```
<pipe  lang="CPP"
    class="ReceivingPipe">
  <datatype name="ppmType">
    <arrayArg name="mag"
           type="char" size="2"/>
    <arg name="width"
     type="long"/>
    <arg name="height"
        type="long"/>
    <arg name="maxval"
        type="long"/>
    <arg name="pictureSize"
         type="integer"/>
    <arrayArg name="picture"
           type="byte"
           size="pictureSize"/>
  </datatype>
  <apply-aspect
    name="receiver_gpce.xsl"/>
   <ports>
    <inport name="in"
    type="ppmType"/>
  </ports>
</pipe>
```

XPath: /xip//datatype//arg[@type='long']

Georgia Tech

XSLT: Pliable generator - output

- Support for new platforms
- *Template* invocation by name or pattern
- Stylesheets allows for imports
- Output templates can be shared
- Language independent (C, C++, Java)
- Allows XML to be inserted in templates

Georgia Tech

XML+XSLT: Modular Output

- Combine extensibility of XML with XSLT
- Insert tags into XSLT to mark blocks of code
- E.g. startup, marshall, unmarshall
- Allows post-generation changes through XML weaving

Georgia Tech

Modularity Example

Generator Template

```
// shutdown all our connections
int infopipe_<xsl:value-of select="$thisPipeName"/>_shutdown()
{
  <jpt:pipe point="shutdown">
  // shutdown incoming ports <xsl:for-each select="./ports/inport">
  infopipe_<xsl:value-of select="@name"/>_shutdown(); </xsl:for-each>
  // shutdown outgoing ports <xsl:for-each select="./ports/outport">
  infopipe_<xsl:value-of select="@name"/>_shutdown(); </xsl:for-each>
  </jpt:pipe>
  return 0;
}
```

Template Output

```
// shutdown all our connections
int infopipe_sender_shutdown()
{
  <jpt:pipe point="shutdown">
  // shutdown incoming ports

  // shutdown outgoing ports
infopipe_ppmOut_shutdown();
  </jpt:pipe>
  return 0;
}
```


Customization example

Base

```
int infopipe_ppmOut_startup()
{
  char *conninfo;
  char *port;
  int portNum;
  struct sockaddr_in sin;
  struct hostent *hptr;
  ppmOut.data = 0; // NULL ptr initially
  lookup( "", "aspectTest/receiver", "ppmIn", PUB
  port = strchr( conninfo, ':' );
  portNum = atoi( port + 1 );
  *port = 0; // null term end of host name
  fprintf(stdout, "Connection to %s:%d.\n\n", con
  ppmOutSocket = socket(PF_INET,SOCK_STREAM,0);
  sin.sin_family = AF_INET;
  sin.sin_port = htons(portNum);
  hptr = gethostbyname( conninfo );
  memcpy(&sin.sin_addr.s_addr,hptr->h_addr_list[0]
  if( connect(ppmOutSocket, (struct sockaddr *)&s
    fprintf(stderr, "Unable to connect to aspectT
    fprintf(stderr, " at location '%s:%d'\n\n", c
```

```
int infopipe_ppmOut_startup()
{
  char *conninfo;
  char *port;
  int portNum;
  struct sockaddr_in sin;
  struct hostent *hptr;
if (pthread_create ( &control_thread_id, NULL, control_thread, NULL) !=0 )
  {
    perror("Unable to create control thread");
    exit(0);
  }
  ppmOut.data = 0; // NULL ptr initially
  lookup( "", "aspectTest/receiver", "ppmIn", PUBLISH_FILE, &conninfo );
  port = strchr( conninfo, ':' );
  portNum = atoi( port + 1 );
  *port = 0; // null term end of host name
  fprintf(stdout, "Connection to %s:%d
  ppmOutSocket = socket(PF_INET,SOCK_S
  sin.sin_family = AF_INET;
  sin.sin_port = htons(portNum);
  hptr = gethostbyname( conninfo );
  memcpy(&sin.sin_addr.s_addr,hptr->h_
  if( connect(ppmOutSocket, (struct so                              {
    fprintf(stderr, "Unable to connect                           n");
    fprintf(stderr, " at location '%s:%d'\n\n", conninfo, portNum);
```

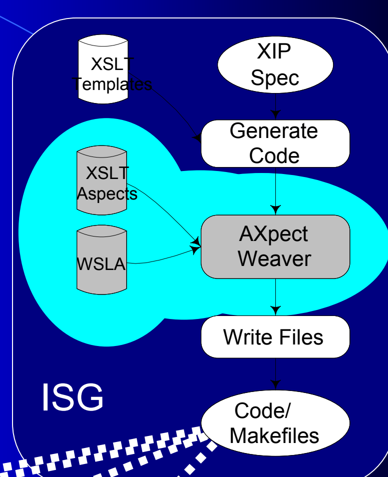
New – yellow is custom control thread creation code added to startup

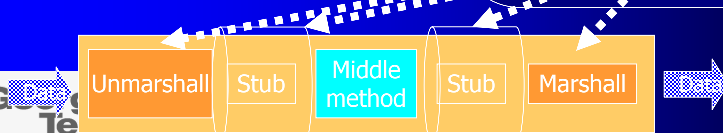
Clearwater Generators

- ISG – horizontal domain
  - For Infopipes
  - Multi-platform
  - Supports Spi, GUI, WSLA
- ACCT – vertical domain
  - For enterprise application deployment
  - Maps Cauldron to SmartFrog or scripts

ISG

- XIP in, code out
- C or C++ language output
- Choice of communication pkg





AXpect

Addresses

- *Modular output*

- XML tags map domain structures to code (joinpoints)
- Use XSLT/XPath to find these tags (pointcuts)
- Augment/replace in gen'd code (advice)
- Allows multiple language weaving

AXpect – Template

- Replaceable code example

```
// startup all our connections
int infopipe_<xsl:value-of select="$thisPipeName"/>_startup()
{

  // insert signal handler startup here

  <jpt:pipe point="startup">
  // start up outgoing ports <xsl:for-each select="./ports/outport">
  infopipe_<xsl:value-of select="@name"/>_startup(); </xsl:for-each> //

  // start up incoming ports <xsl:for-each select="./ports/inport">
  infopipe_<xsl:value-of select="@name"/>_startup(); </xsl:for-each> //

  <xsl:for-each select="./ports/inport">
  <xsl:if test="not(@receiveloop='false')">
  infopipe_<xsl:value-of select="./@name"/>_receiveloop();
  </xsl:if> </xsl:for-each> //
  </jpt:pipe>

  return 0;
}
```

Joinpoint for startup code in template, start and end

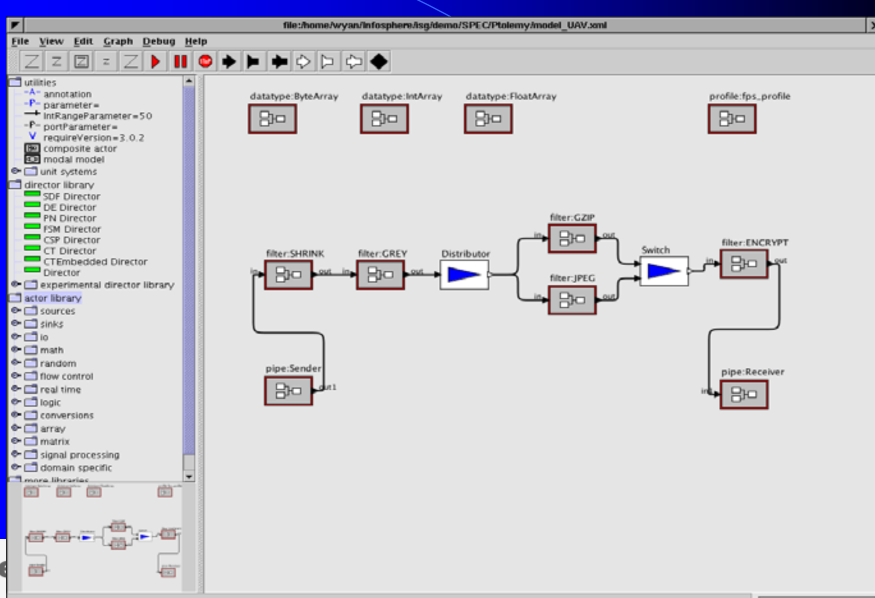
AXpect - Aspect

```
<xsl:template
    match="//filledTemplate[@name=$pipename]
                          [@inside=$inside]//jpt:pipe-middle">
  struct timeval base;
  struct timeval end;
  <jpt:time-process>
  // take timing here
  gettimeofday(&amp;base,NULL);
  <xsl:copy>
    <xsl:apply-templates select="@*|node()"/>
  </xsl:copy>
  gettimeofday(&amp;end,NULL);
  usec_to_process = (end.tv_sec - base.tv_sec  ) *
                 1e6 + (end.tv_usec - base.tv_usec);
  fprintf(stdout,"Time to process: %ld\n", usec_to_process);
  </jpt:time-process>
</xsl:template>
```

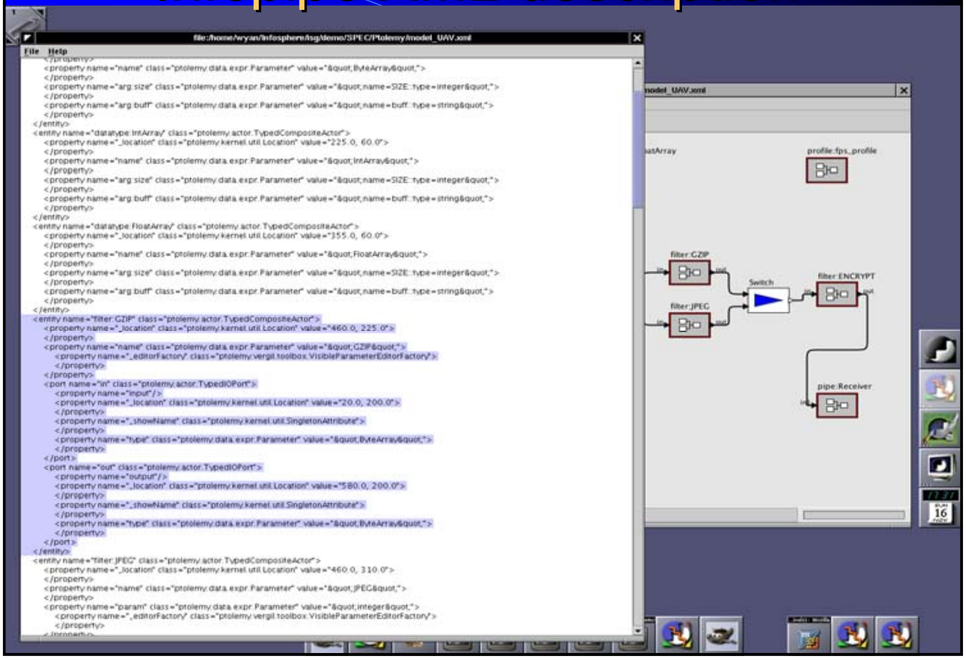
XSLT
C code
Pointcut
Joinpoint

Georgia Tech

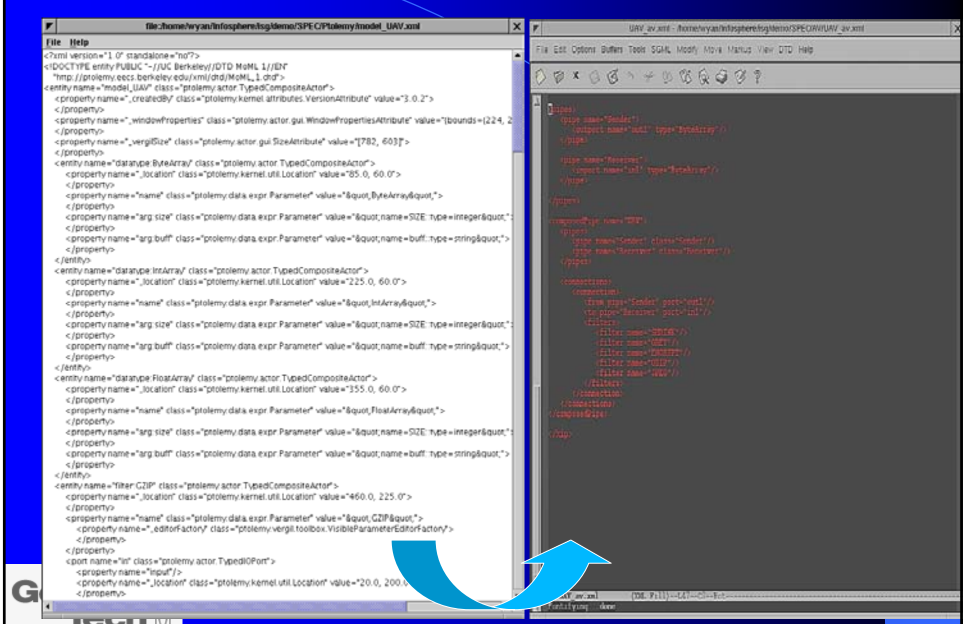
Infopipe Specification



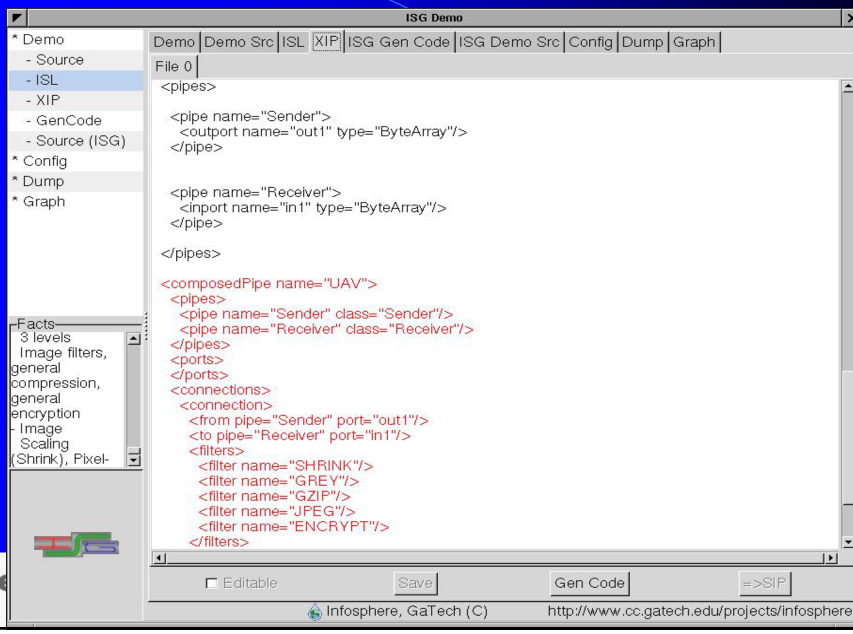
Infopipe XML description



Ptolemy => ISL/XIP



XIP => Gen code



An Infopipe Aspect

```
<xsl:template
    match="//filledTemplate[@name=$pipename]
                               [@inside=$inside]//jpt:pipe-middle">
  struct timeval base;
  struct timeval end;
  <jpt:time-process>
  // take timing here
  gettimeofday(&amp;base,NULL);
  <xsl:copy>
    <xsl:apply-templates select="@*|node()"/>
  </xsl:copy>
  gettimeofday(&amp;end,NULL);
  usec_to_process = (end.tv_sec - base.tv_sec  ) *
                1e6 + (end.tv_usec - base.tv_usec);
  fprintf(stdout,"Time to process: %ld\n", usec_to_process);
  </jpt:time-process>
</xsl:template>
```

XSLT
C code
Pointcut
Joinpoint

Georgia Tech

0:Base Code

- Just TCP communication
- Binding, connecting, marshalling, etc.
- No application code
- No QoS

Georgia Tech

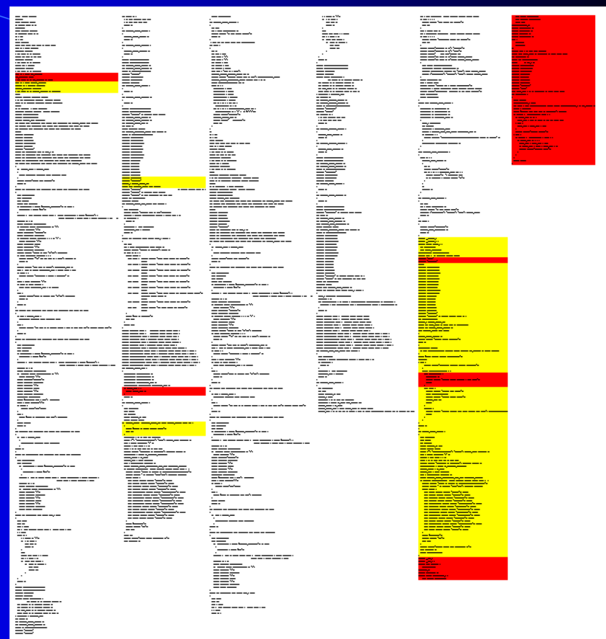
1:Sender Control

- Sender-side Feedback channel
- MUXs control messages
- Binds, connects
- TCP transport

Georgia Tech

2:Sender WSLA

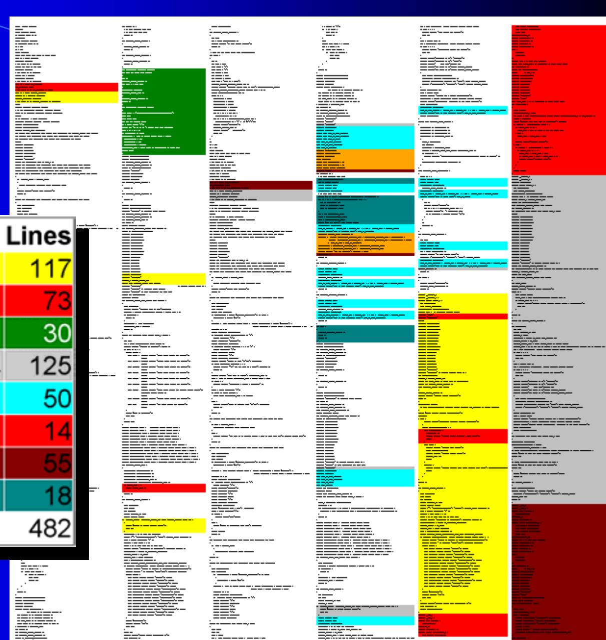
- Implements SLA pieces for sender
- Param'ed by the WSLA
- Can send messages over the control channel
- Responds to receiver feedback



Georgia Tech

The Entire Infopipe Application

Aspect	Lines
control_sender.xsl	117
sla_sender.xsl	73
sender.xsl	30
control_receiver.xsl	125
timing.xsl	50
cpumon.xsl	14
sla_receiver.xsl	55
receiver.xsl	18
TOTAL	482



Georgia Tech

Where the Code Goes

Receiver-side

Aspect / Affected File	Makefile	receiver.h	receiver.c	ppmIn.h	ppmIn.c	control.h	control.c	sla.c	sla.h	# Lines Added
timing			X		X					50
control_receiver	X		X		X	(X)	(X)			125
cpumon			X							14
sla_receiver	X		X			X	X	(X)	(X)	55

Sender-side

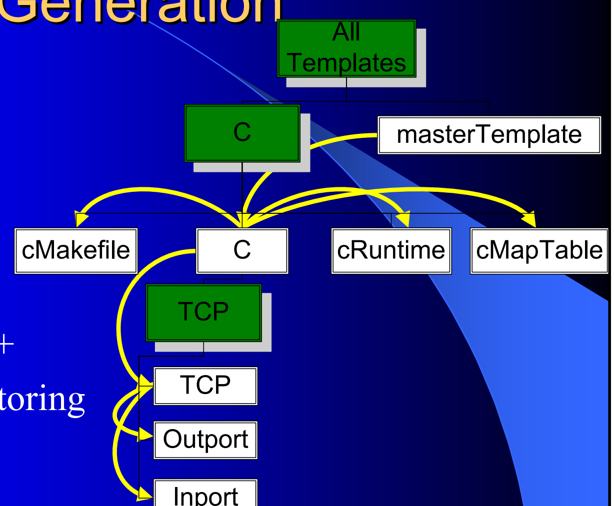
Aspect / Affected File	Makefile	sender.h	sender.c	ppmOut.h	ppmOut.c	control.h	control.c	sla.c	sla.h	# Lines Added
control_receiver	X				X	(X)	(X)			117
sla_receiver	X				X	X	X	(X)	(X)	73
Total Aspect Lines										**434**
Base Implementation										**976**
Complete Application										**1410**

QoS code affects 13 of 18 files (from 6 AXpect files)

QoS code is ≈30% of total

ISG: Template-based Generation

- XSLT & Source
- Grab information directly from XIP
- ISG calls master template
- Parallel set for C++
- Amenable to refactoring



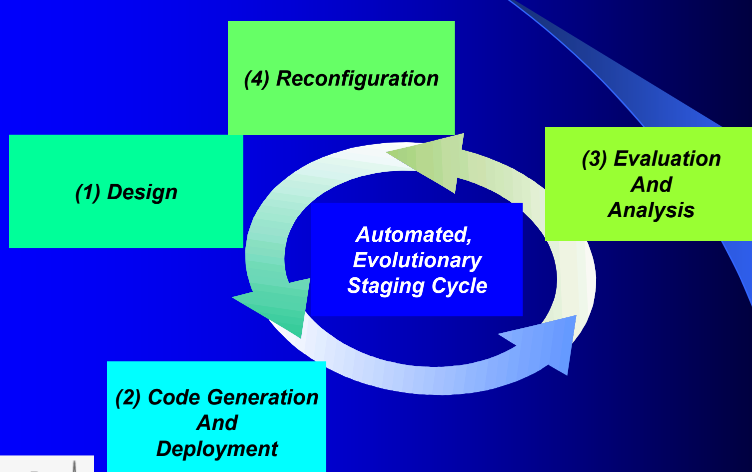
Georgia Tech

ISG: Observations

- C and C++ generation can share templates
  – 10% of template code at present
- Sharing between communications platforms
  – C TCP and ECho share about 20%
- Further factorizations might enhance code sharing
  – Benefit: improved interoperability

Georgia Tech

Staging of *N*-Tier Applications



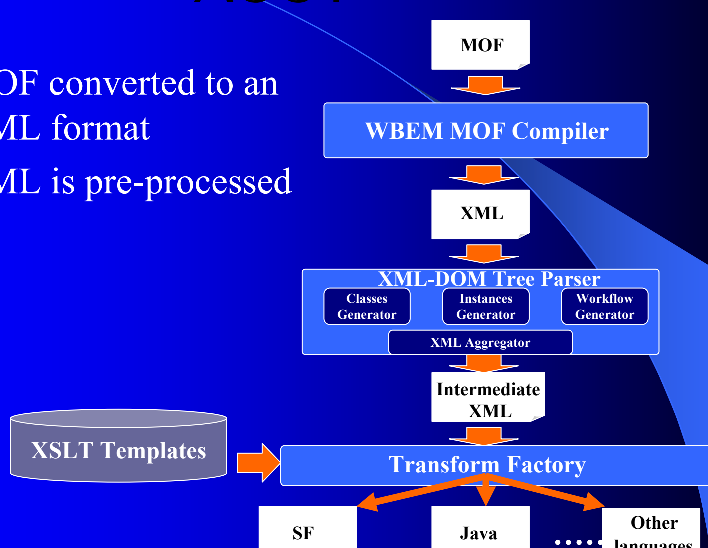
Georgia Tech

ACCT Code Generator

- Input policy documents
  - Provide deployment constraints
  - Describe hardware and software
- Perform resource assignment (via Cauldron)
  - Output (MOF) has no execution support
- Translate into toolkit specifications
  - Target is SmartFrog

ACCT

- MOF converted to an XML format
- XML is pre-processed



ACCT Transformation

```
instance of LogicalServer {
  Id = "Tomcat_LS1";
  Caption = "Tomcat Logical Server";
  Description = "Logical Server for Tomcat ";
  IpAddress = "130.207.5.228";
  HostName = "artemis.cc.gatech.edu";
};
instance of LogicalServerInLogicalApplication
{
  LogicalApplication = "Tomcat\";
  LogicalServer =Tomcat_LS1\";
};
instance of LogicalApplication {
  Id = "Tomcat";
  Version = "5.0.19";
  Caption = "Tomcat";
  Description = "Tomcat application Server";
};
instance of LogicalApplication {
  Id = "MySQLDriver";
  Version = "3.0.11";
  Caption = "MySQLDriver";
  Description = "MySQL driver";
};
instance of Activity {
  Id = "Tomcat_Installation";
  ActivityType = "script";
};
instance of Activity {
  Id = "Tomcat_Installation";
  ActivityType = "script";
};
Instance of ActivityPredecessorActivity {
  DependenceType="Finish-Start";
  AntecedentActivity="Tomcat_Installation";

DependentActivity="MySQLDriver_installation
";
};
```

(a) MOF

```
 <Instance                    Name="Tomcat"
Class="LogicalApplication">
  <Variable
Name="Id"Type="string">Tomcat</Variable>
  <Variable Name="Version"Type="string">
    5.0.19</Variable>
  <Variable Name="Entity" Type="string">
    Activity_Tomcat_Installation</Variable>
  <Variable Name="Host" Type="string">
    artemis.cc.gatech.edu</Variable>
 </Instance>
  <Workflow>
    <Work Type="Execution"></Work>
    <Work Type="EventSend">
      <To> MySQLDriver_Installation</To></Work>
    <Work Type="Terminate">
     Tomcat_Installation </Work>
  </Workflow>
 <Instance  Name="MySQLDriver"
           Class="LogicalApplication">
  <Variable Name="Id" Type="string">
    MySQLDriver</Variable>
  <Variable Name="Version" Type="string">
    3.0.11</Variable>
  <Variable Name="Entity" Type="string">
    Activity_MySQLDriver_Installation</Variable>
  <Variable Name="Host" Type="string">
    demeter.cc.gatech.edu</Variable>
 </Instance>
  <Workflow>
   <Work Type="OnEvent">
     <From> Tomcat_Installation</From> </Work>
   <Work Type="Execution"></Work>
   <Work Type="Terminate">
     MySQLDriver_Installation</Work>
  </Workflow>
```

(b) XML

```
sfProcessComponentName  "Tomcat_Installation";
LogicalApplication_Tomcat  extends  LogicalApplication {
     Id   "Tomcat";
     Version   "5.0.19";
     Activity LAZY ATTRIB   Activity_Tomcat_Installation;
     sfProcessHost   "artemis.cc.gatech.edu";
}
Activity_Tomcat_Installation  extends  Activity {
     Id   "Tomcat_Installation";
     Entity   LAZY ATTRIB LogicalApplication_Tomcat;
}
-- extends EventSend {
     sendTo eventQueue:queue_Tomcat_Ignition;
     event "Activity_Tomcat_Installation_FS";
}
-- extends Terminator {
     kill eventQueue:queue_Tomcat_Installation;
}
sfProcessComponentName "MySQLDriver_Installation";
-- extends OnEvent {
    registerWith queue_MySQLDriver_Installation ;
    Activity_Tomcat_Installation_FS extends DoNothing
}
LogicalApplication_MySQLDriver extends LogicalApplicatio
     Id   "MySQLDriver";
     Version   "3.0.11";
     ActivityLAZYATTRIBActivity_MySQLDriver_Installation;
     sfProcessHost   "demeter.cc.gatech.edu";
}
Activity_MySQLDriver_Installation extends Activity {
     Id   "MySQLDriver_Installation";
    Entity  LAZY ATTRIB   LogicalApplication_MySQLDriver;
}
-- extends Terminator {
    kill eventQueue:queue_MySQLDriver_Installation;
}
```

(c)

ACCT: Observations

- Now reused inside another tool
  - Mulini – enterprise application staging
- Extended to support new target
  - Shell scripts
  - Partial implementation (but low-cost)

Summary

- Extensibility, Pliability, Modularity
  - Good to have in distributed systems work
  - For us, modularity/AOP is great
- XML and XSLT support E-P-M
  - Examples in vertical, horizontal domains
  - Seem to have good *generator* modularity
- XSLT caveats
  - Can have heavy “syntax”
  - Looking for good replacements

Georgia Tech

Discussion

- Principled manipulation of code (to preserve correctness)
  - Specialization of source programs
  - Code generation (from specifications)
  - AOP in code generation and weaving

Georgia Tech