# Towards an Internet-wide Distributed System for Media Stream Processing & Delivery

Richard West, Xin Qi, Gabriel Parmer, Jason Gloudon, Gerald Fry

Boston University

Boston, MA

richwest@cs.bu.edu

Computer Science

# Introduction



- Internet growth has stimulated development of data- rather than CPU-intensive applications
  - e.g., streaming media delivery, interactive distance learning, webcasting (e.g., SHOUTcast)
- Peer-to-peer (P2P) systems now popular
  - Efficiently locate & retrieve data (e.g., mp3s)
  - e.g., Gnutella, Freenet, Kazaa, Chord, CAN, Pastry
- To date, limited work on scalable delivery & processing of (potentially real-time) data streams

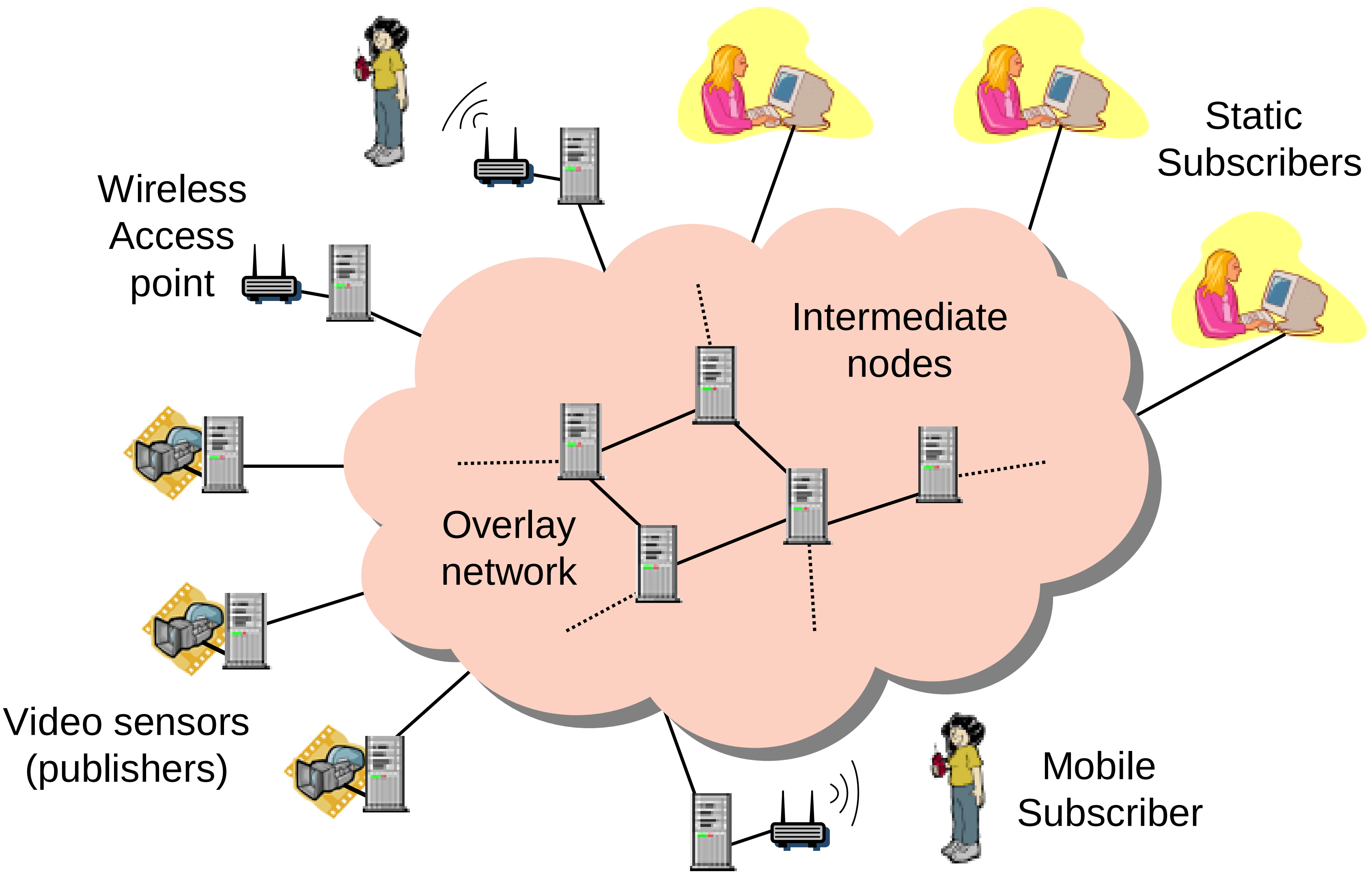
# Introduction (2)



- Aim:
  - Build an Internet-wide distributed system for delivery & processing data streams
    - Implement logical network of end-systems
    - Support multiple channels connecting publishers to 1000s of subscribers w/ own QoS constraints
- Rationale:
  - Narada provided case for end-system multicast
  - Rely only on IP uni-cast routing at network-level
  - Overlay routing provides flexibility for app-specific data processing

# “Big Picture”




Wireless
Access
point
Static
Subscribers
Intermediate
nodes
Overlay
network
Video sensors
(publishers)
Mobile
Subscriber

# Research Goals



- Logical overlay topologies for scalable QoS-constrained routing
  - Leverage ideas from P2P systems & parallel (NUMA) computer architectures
  - Combine scalable properties of P2P systems such as Chord, CAN & Pastry w/ service guarantees of systems such as Narada
- Efficient end-host software architecture, supporting:
  - App-specific stream processing / routing
  - Resource monitoring
  - Overlay management

# Contributions



- (1) Analysis of k-ary n-cubes for scalable overlay topologies
  - Optimized initial configurations
  - Comparison of routing algorithms
  - Dynamic host relocation in logical space based on QoS constraints
- (2) End-host architecture design
  - Efficient support for app-specific service extensions
  - Provide safety
  - Avoid context-switch overheads
  - Reduce communication costs

# Part 1: Scalable Overlay Topologies



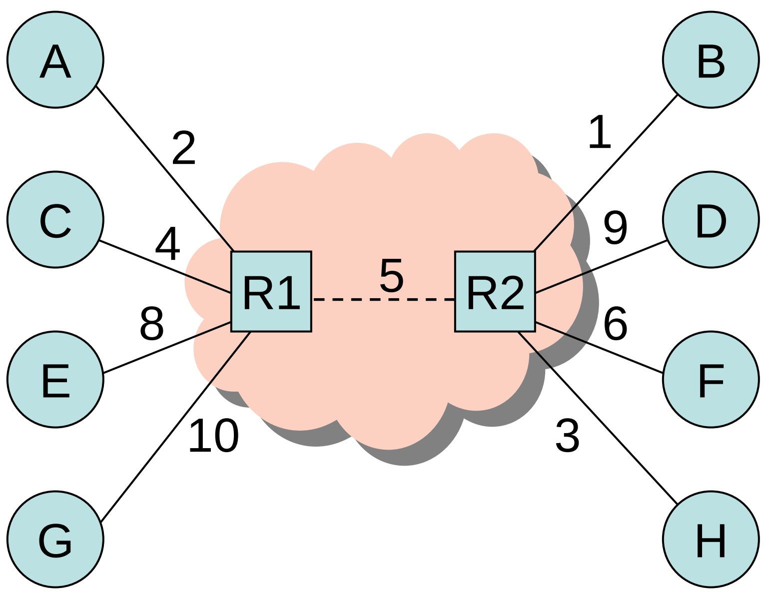
- NUMA architectures have scalable interconnects
  - e.g., hypercubes – SGI Origin 2/3000
- P2P systems based on distributed hashing implicitly construct torus or k-ary-n-cube topologies connecting end-hosts
  - e.g., Chord, CAN, Pastry
  - For a system of M hosts:
    - O(log M) routing state per node
    - O(log M) hops between source and destination to find desired info

# Overlay Routing Example

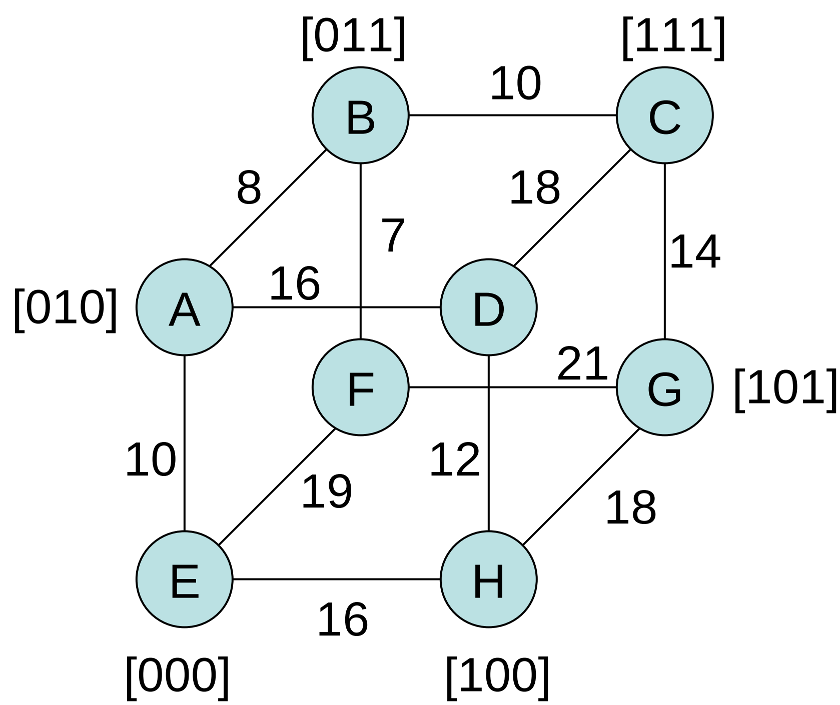


- Overlay is modeled as an undirected k-ary n-cube graph
- An edge in the overlay corresponds to a uni-cast path in the physical network

**Physical view**




**Logical view**



# Definition of k-ary n-cube Graphs



- A k-ary n-cube graph is defined by two parameters:
  - n = # dimensions
  - k = radix (or base) in each dimension
- Each node is associated with an identifier consisting of n base-k digits
- Two nodes are connected by a single edge iff:
  - their identifiers have n-1 identical digits, and
  - the ith digits in both identifiers differ by exactly 1 (modulo k)

# Properties of k-ary n-cube Graphs

Computer Science

- $M = k^n$ nodes in the graph
- If k = 2, degree of each node is n
- If k > 2, degree of each node is 2n
- Worst-case hop count between nodes:
  - $n\lfloor k/2 \rfloor$
- Average case path length:
  - $A(k,n) = n \lfloor (k^2/4) \rfloor 1/k$
- Optimal dimensionality:
  - n = ln M
  - Minimizes A(k,n) for given k and n

# Logical versus Physical Hosts



- Mapping between physical and logical hosts is not necessarily one-to-one
  - M logical hosts
  - m physical hosts
- For routing, we must have m <= M
  - Destination identifier would be ambiguous otherwise
- If m < M, then some physical host(s) must perform the routing functions of multiple logical nodes

# M-region Analysis

Computer Science

- Hosts joining / leaving system change value of m
  - Initial system is bootstrapped with overlay that optimizes A(k,n)
- Let M-region be range of values for m for which A(k,n) is minimized
- Consider two graphs corresponding to $(k_1,n_1)$ and $(k_2,n_2)$:
  - Suppose $k_1 n_1 = k_2 n_2$ and $k_1^{n_1} > k_2^{n_2}$
  - The graph corresponding to $(k_1,n_1)$ is desirable

# Calculating M-regions

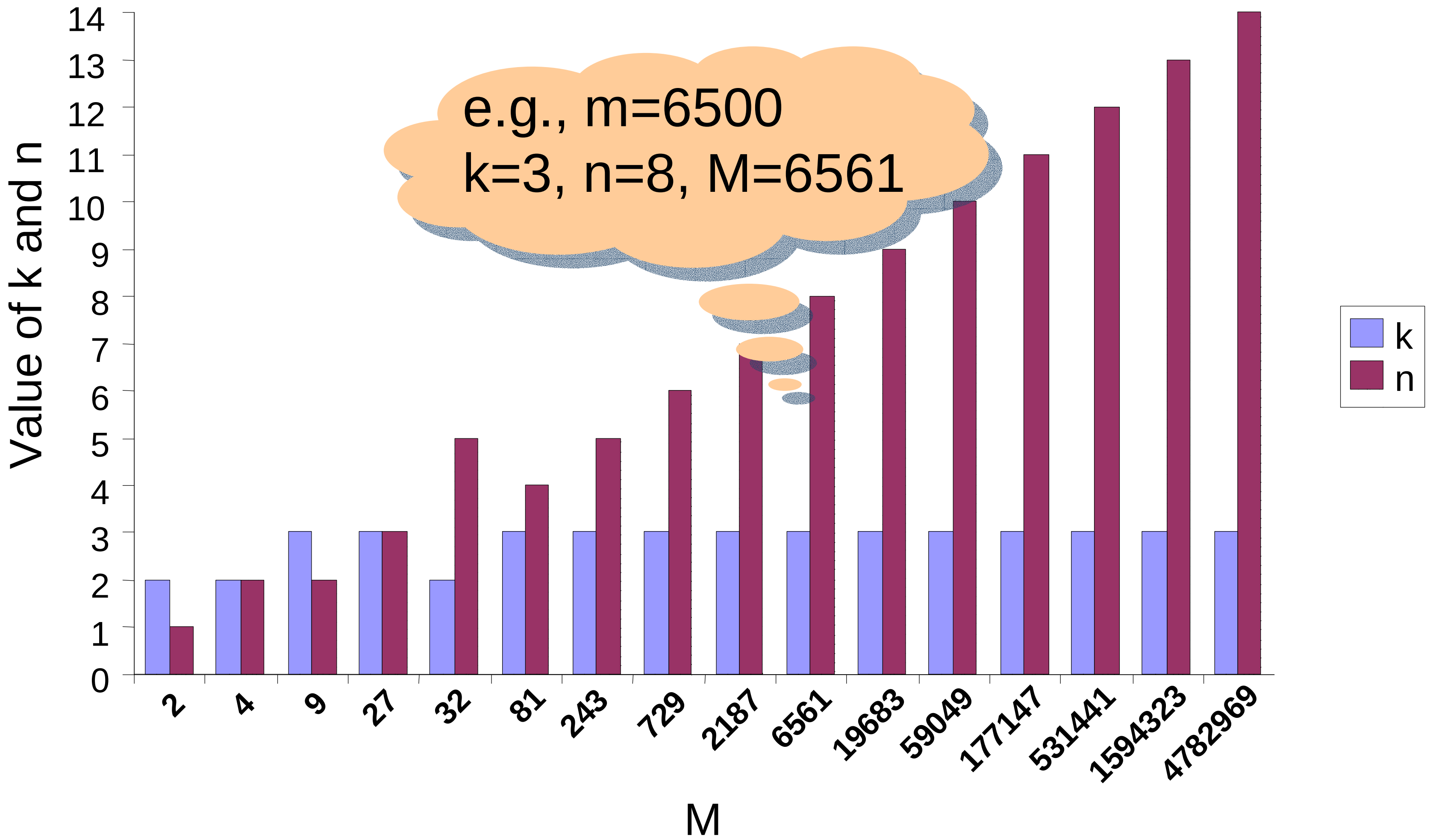
Computer Science

```
Calculate_M-Region(int m) {
    i = 1;   k = j = 2;
    while (M[i,j] < m) i++; // Start with a hypercube
    n = i;
    maxM = M[i,j];
    minA = A[i,j];
    incj = 1;
    while (i > 0) {
        j += incj; i--;
        if ((A[i,j] <= minA) && (M[i,j] > maxM)) {
            incj = 1;
            maxM = M[i,j];
            minA = A[i,j];
            n = i; k = j;
        }
        else incj = 0;
      }
    return k, n;
  }
```

Try to find the largest M such that: m <= M & A(k,n) is minimized

# M-regions




e.g., m=6500
k=3, n=8, M=6561
Value of k and n
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
2
4
9
27
32
81
243
729
2187
6561
19683
59049
177147
531441
1594323
4782969
M
k
n

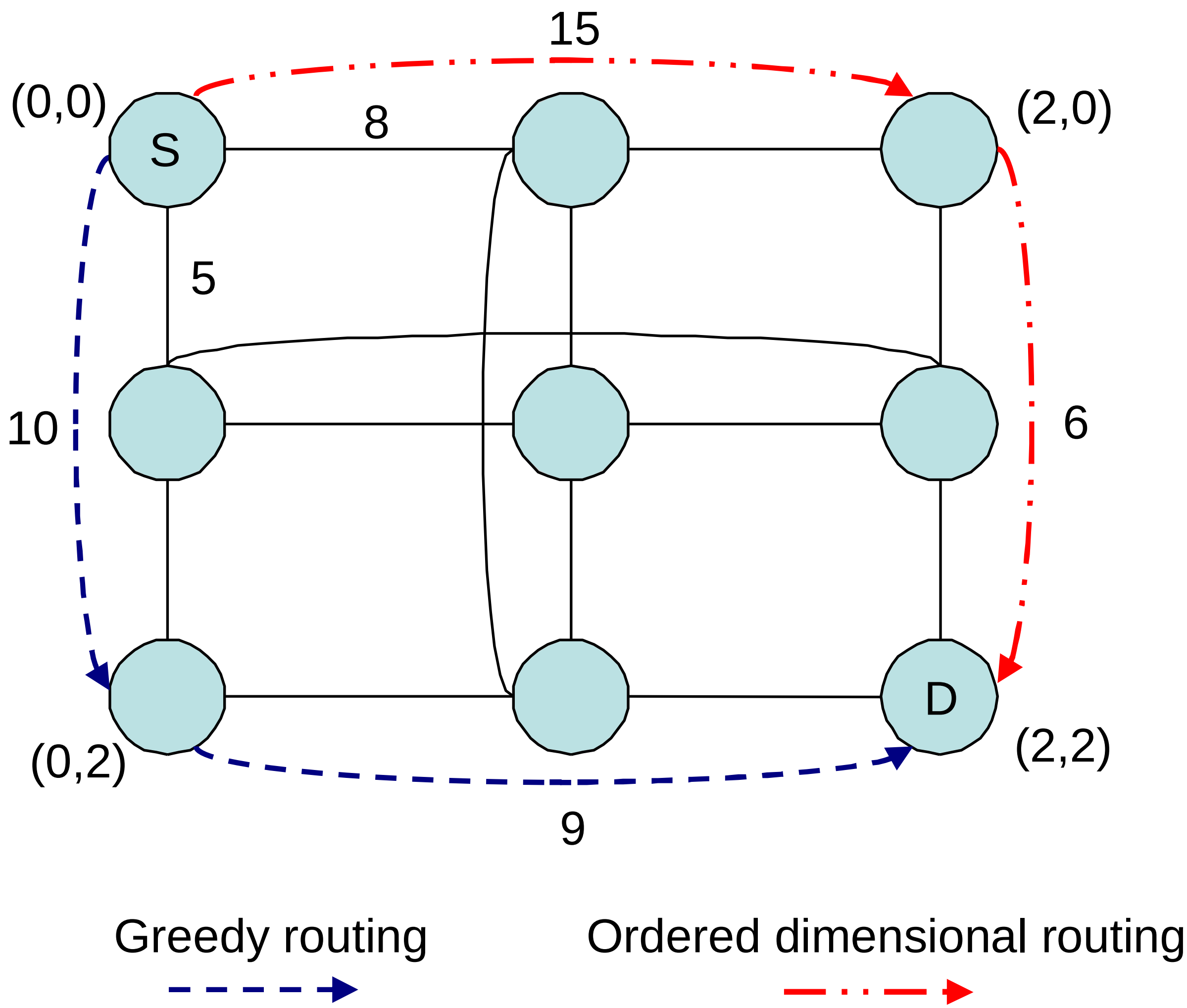
# Overlay Routing



- Three routing policies are investigated
  - Ordered Dimensional Routing (ODR)
  - Random Ordering of Dimensions (Random)
  - Proximity-based Greedy Routing (Greedy)
    - Forward message to neighbor along logical edge with lowest cost that reduces hop-distance to destination
- Experimental analysis done via simulation written in C
  - 5050 routers in physical topology (transit-stub)
  - 65536 hosts

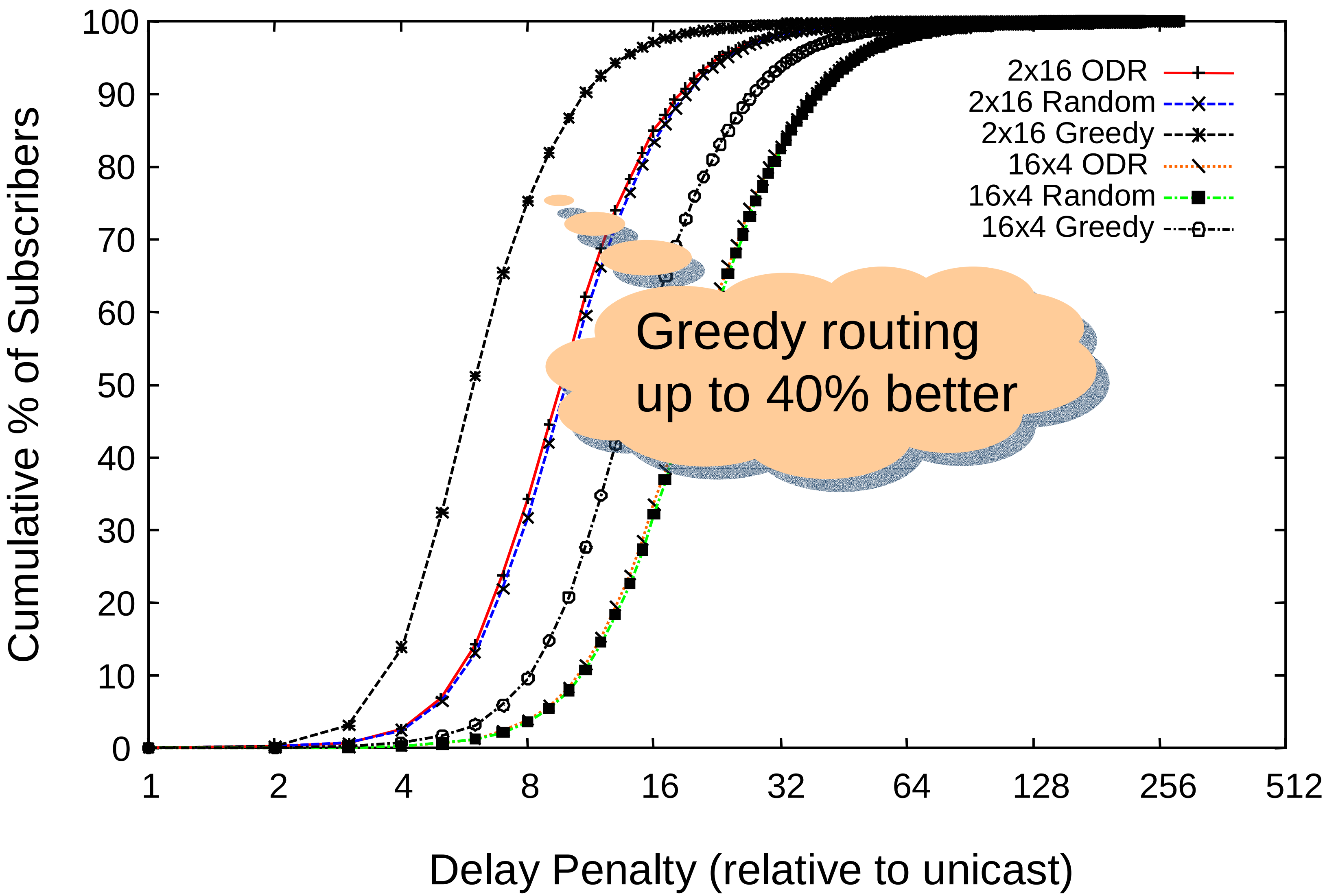
# Greedy-based Routing Example





# Overlay Routing: 16D Hypercube versus 16-ary 4-cube

Computer Science



# Adaptive Node Assignment



- Initially, hosts are assigned random node IDs
- Publisher hosts announce availability of channels
  - Super-nodes make info available to peers
- Hosts subscribing to published channels specify QoS constraints (e.g., latency bounds)
- Subscribers may be relocated in logical space
  - to improve QoS
  - by considering "physical proximities" of publishers & subscribers

# Adaptive Node Assignment (2)



```
Subscribe (Subscriber S, Publisher P, Depth d) {
  if (d == D) return;

  find a neighbor i of P such that
    i.cost(P) is maximal for all neighbors

  if (S.cost(P) < i.cost(P))
    swap logical positions of i and S;
  else
    Subscribe (S, i, d+1);
}
```

- Swap S with node i up to D logical hops from P

# Simulation Results

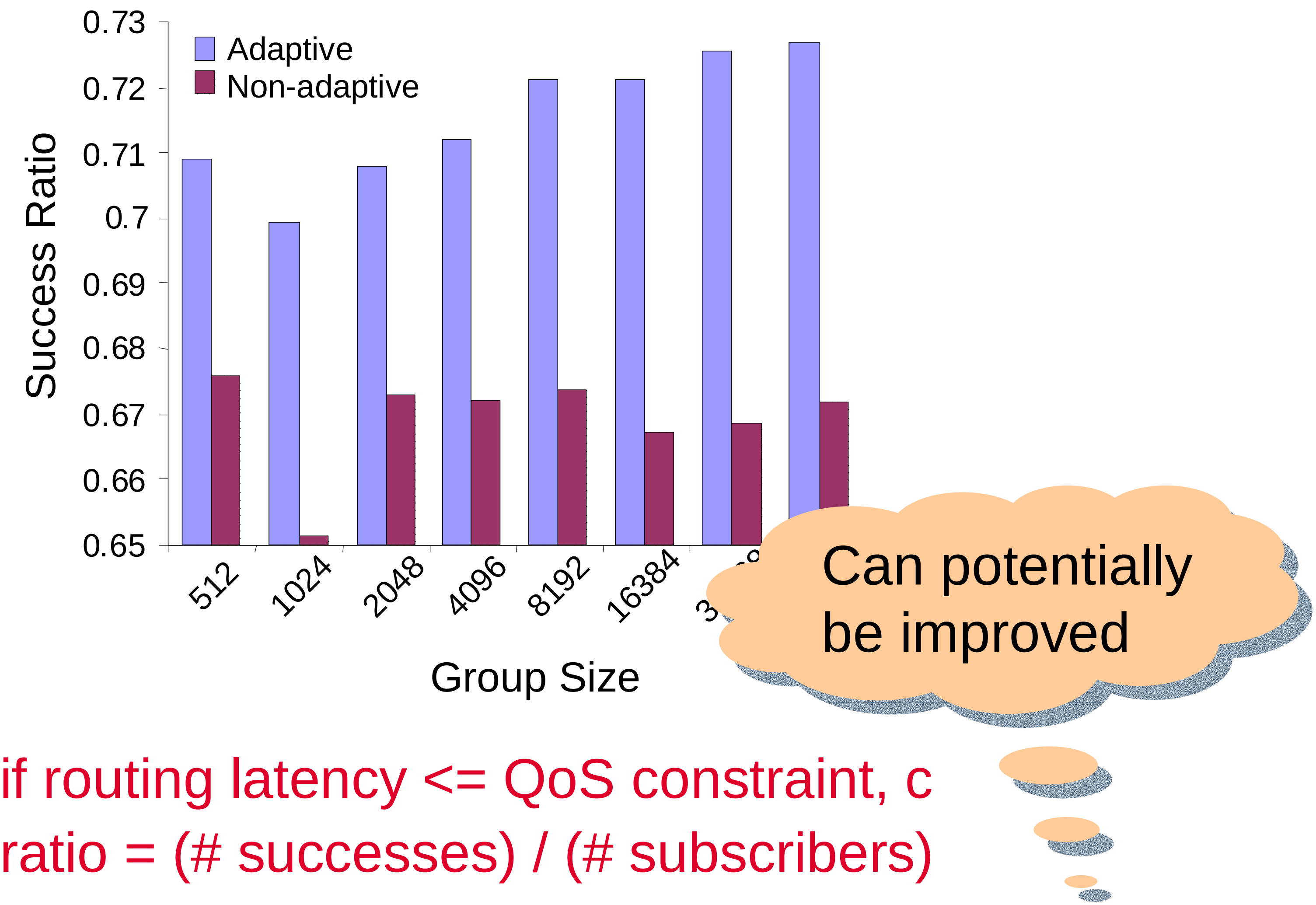


- Randomly generated physical topology with 5050 routers
- M=65536 and topology is a 16D hypercube
- Randomly chosen publisher plus some number of subscribers with QoS (latency) constraints

- Adaptive algorithm used with D=1
- Greedy routing performed with & without adaptive node assignment

# Success Ratio vs Group Size

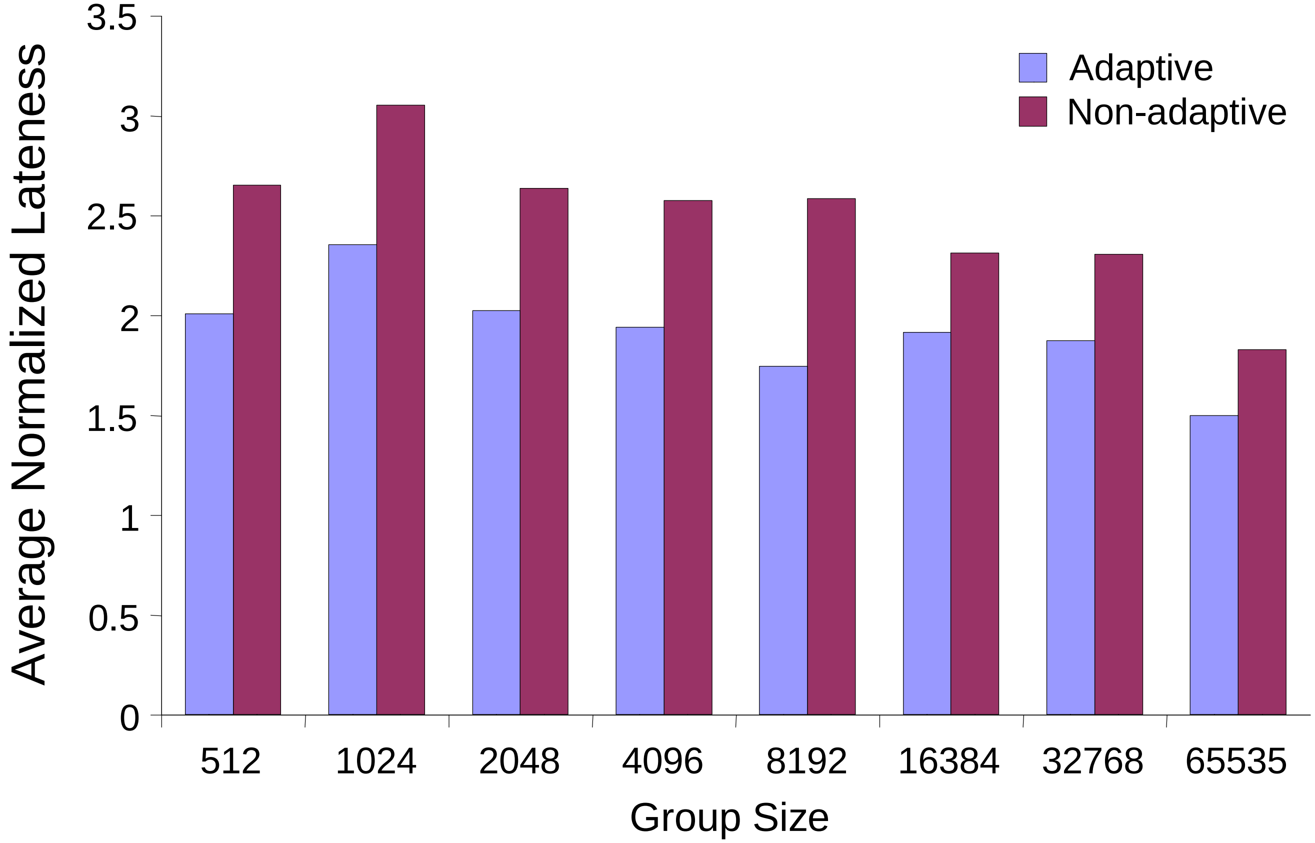





- Success if routing latency <= QoS constraint, c
- Success ratio = (# successes) / (# subscribers)
- Adaptive node assignment shows up to 5% improvement

# Lateness versus Group Size

Computer Science




- Normalized lateness = 0, if S.cost(P) <= c
- Normalized lateness = (S.cost(P)-c)/c, otherwise
- Adaptive method can yield >20% latency reduction

# Adaptive Node ID Assignment



- Initial results look encouraging
- Improved performance likely if adaptation considers nodes at greater depth,D, from publishers
  - Expts only considered D=1
- Adaptive node assignment attempts to minimize maximum delay between publishers and subscribers

# Link Stress

Computer Science

- Previously, aimed to reduce routing latencies
- Important to consider physical link stress:
  - Avg times a message is forwarded over a given link, to multicast info from publisher(s) to all subscribers
- New “split-based greedy” alg:
  - Use greedy routing BUT…
  - At each hop check neighbor to see if already a subscriber
  - If so, route via neighbor if total delay from publisher to subscriber is reduced, compared to pure greedy approach

# Link Stress Simulation Results



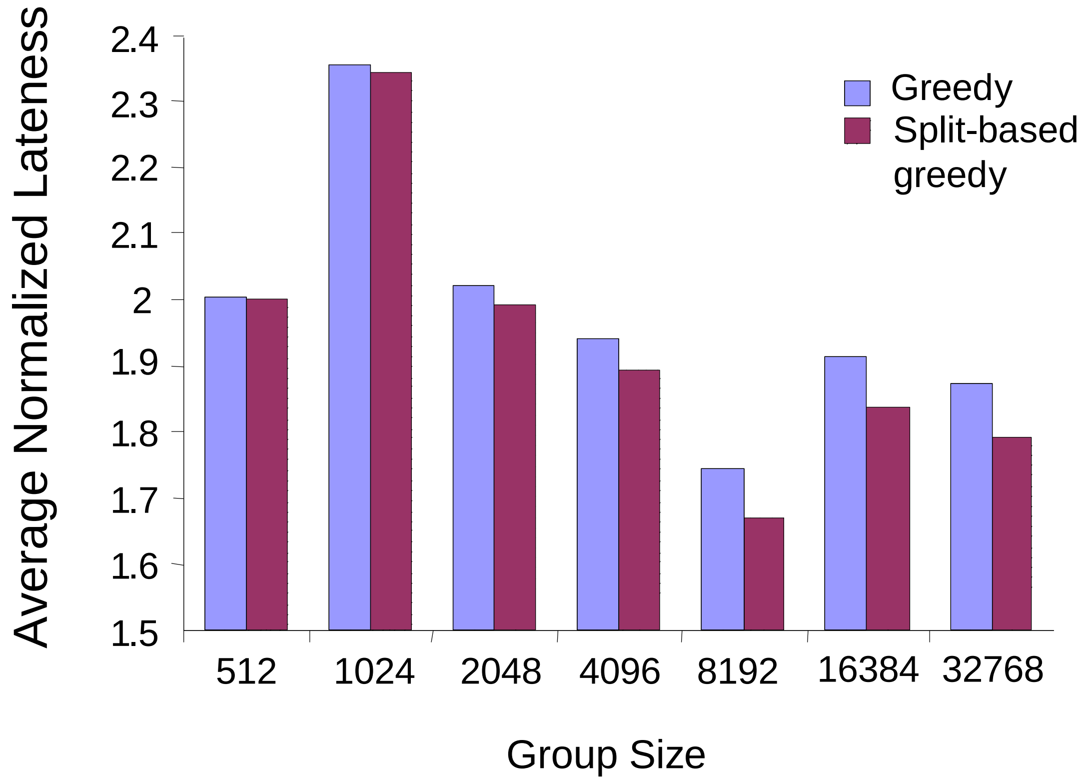
- 16D hypercube overlayed on random physical network
- Randomly chosen publisher plus varying groups of subscribers
- Multicast trees computed from union of routing paths between publisher and each subscriber

  - Compare greedy versus "split-based" greedy algorithm
  - Compare avg physical link stress:

$$\frac{\text{(\# times message is forwarded over a link)}}{\text{(\# unique links required to route msg to all subscribers)}}$$

# Lateness versus Group Size

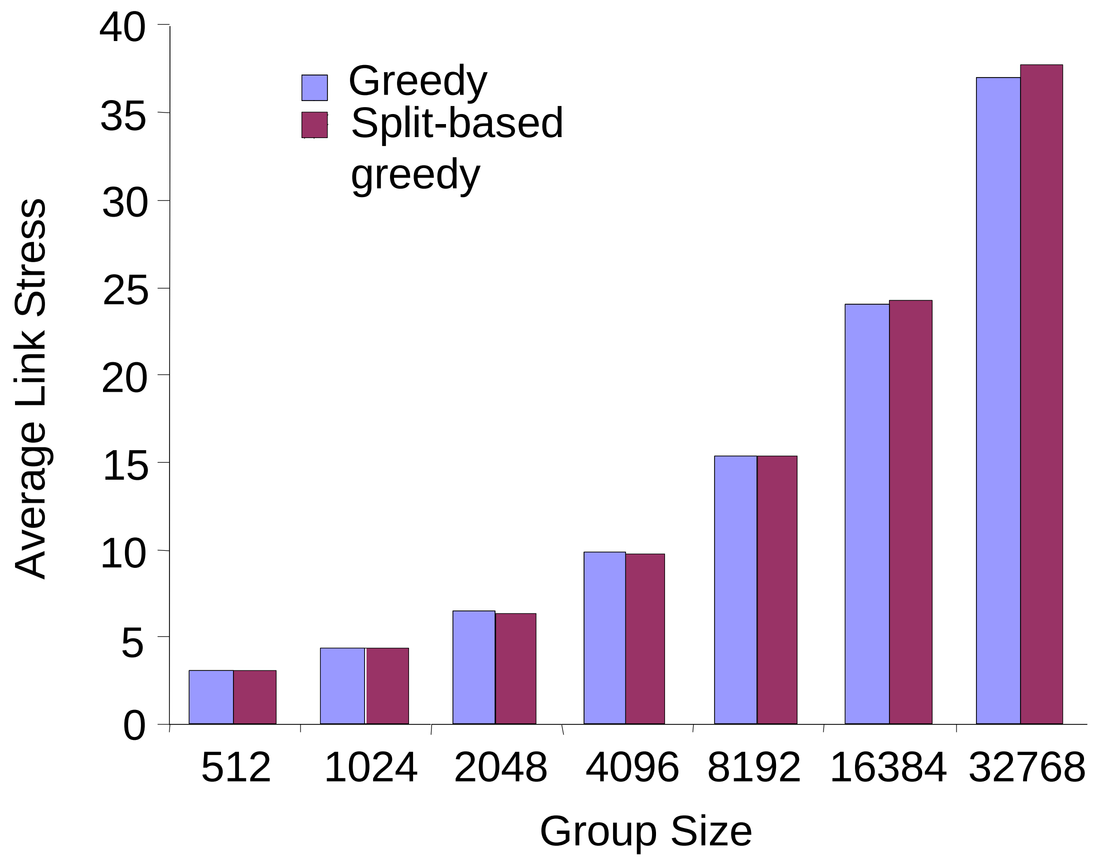
Computer Science




- Variations in lateness (for pairs of columns) due in part to random locations of subscribers relative to publisher

# Link Stress versus Group Size






- “Split-based” greedy performs worse as group size increases
- Appears to be due to slightly greater intersection of physical links for multicast tree (i.e. fewer physical links)

# Conclusions



- Analysis of k-ary n-cube graphs as overlay topologies
  - Minimal average hop count
  - M-region analysis determines optimal values for k and n
- Greedy routing
  - Leverages physical proximity information
  - Significantly lower delay penalties than existing approaches based on P2P routing
- Adaptive node ID re-assignment for satisfying QoS constraints

# Future and Ongoing Work



- Further investigation into alternative adaptive algorithms
- How does changing the overlay structure affect per-subscriber QoS constraints?
- Currently building an adaptive distributed system
  - QoS guarantees of NARADA
  - Scalability of systems such as Pastry/Scribe

# Part 2: End-system Architecture



- Aim is to modify COTS systems to support efficient methods of application and system extensibility
- Why?
  - To support efficient app-specific routing & processing of data on end-systems also used for other purposes
- Approach
  - User-level sandboxing:
    - Provide efficient method for isolating and executing extensions
    - Provide efficient method for passing data between user-level and network interface

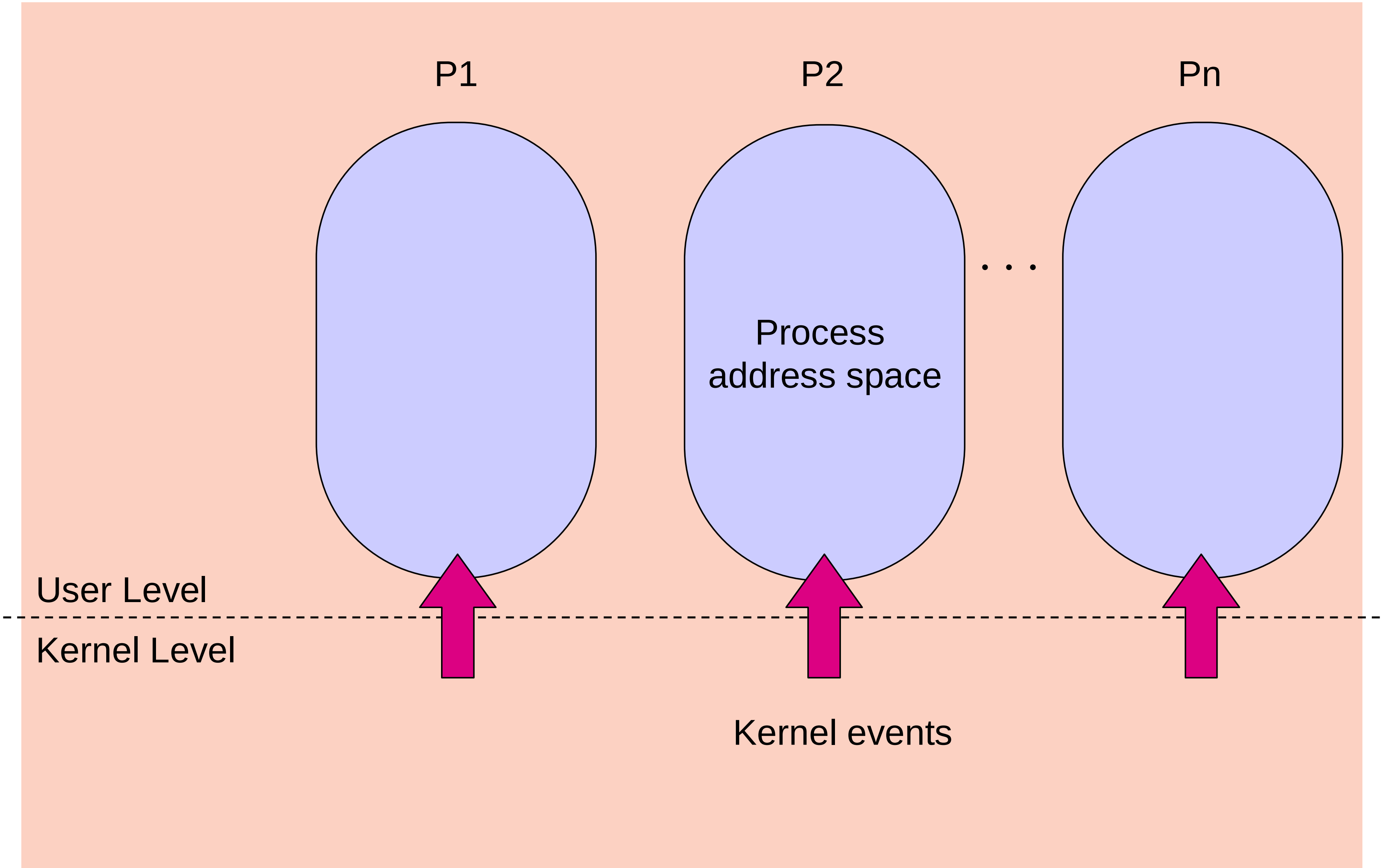
# User-Level Sandboxing (ULS)



- Provide safe environment for service extensions
- Separate kernel from app-specific code
- Use only page-level hardware protection
  - Rely on type-safe languages e.g., Cyclone for memory safety of extensions, or require authorization by trusted source
- Approach does not require special hardware protection features
  - Segmentation
  - Tagged TLBs

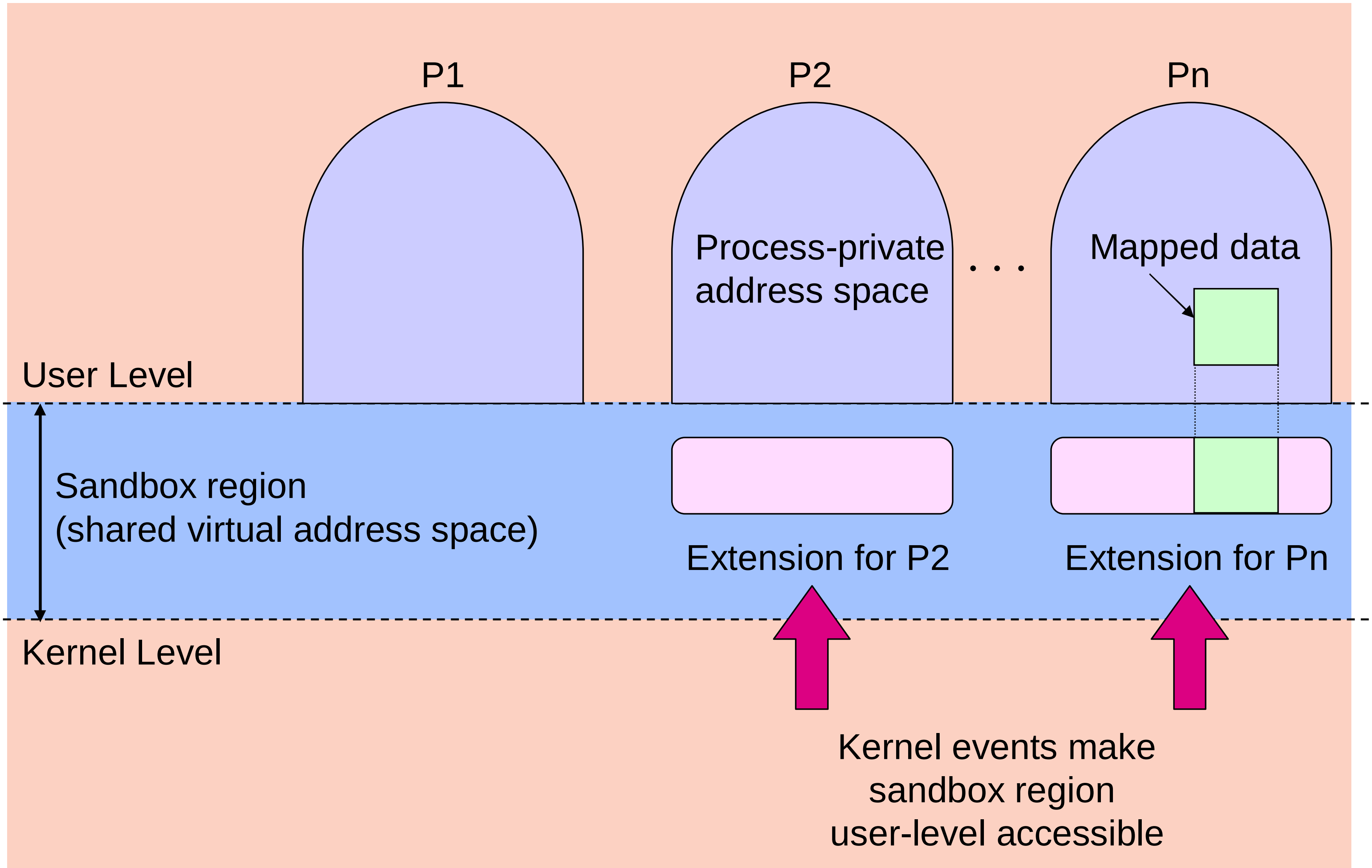
# Traditional View of Processes





# Sandbox Region Shared by Processes





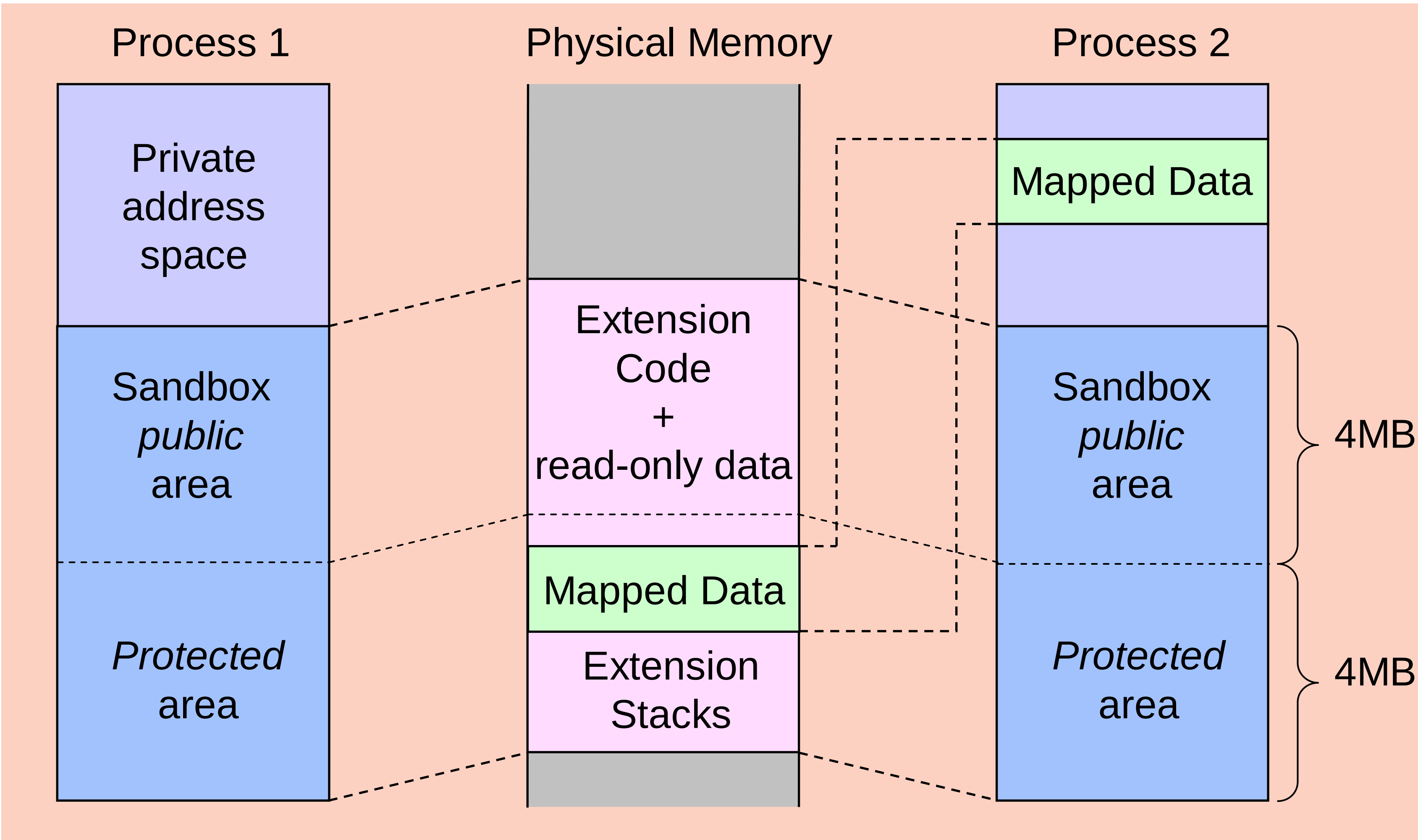
# ULS Implementation



- Modify address spaces of all processes to contain one or more shared pages of virtual addresses
  - Shared pages used for sandbox
    - Normally inaccessible at user-level
    - Kernel upcalls toggle sandbox page protection bits & perform TLB invalidate on corresponding page(s)
- Current x86 approach
  - 2x4MB superpages (one data, one code)
  - Modified libc to support mmap, brk, shmget etc
  - ELF loader to map code into sandbox
  - Supports sandboxed threads that can block on syscalls

# Virtual-to-Physical Memory Mapping

Computer Science



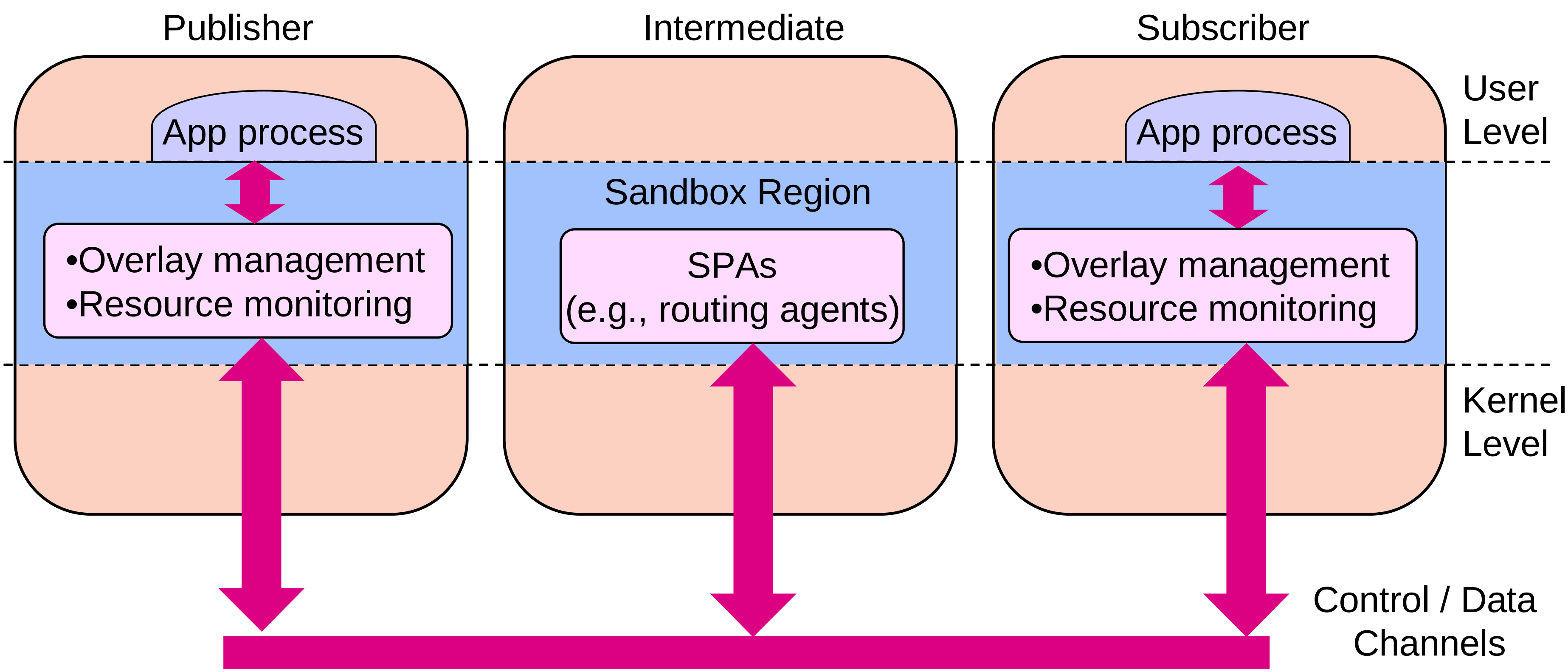
# ULS Implementation (2)



- Fast Upcalls
  - Leverage SYSEXIT/SYSENTER on x86
    - Support traditional IRET approach also
- Kernel Events
  - Generic interface supports delivery of events to specific extensions
  - Each extension has its own stack & thread struct
    - Extensions share credentials (including fds) with creator
  - Events can be queued ala POSIX.4 signals

# End-hosts “Big Picture”





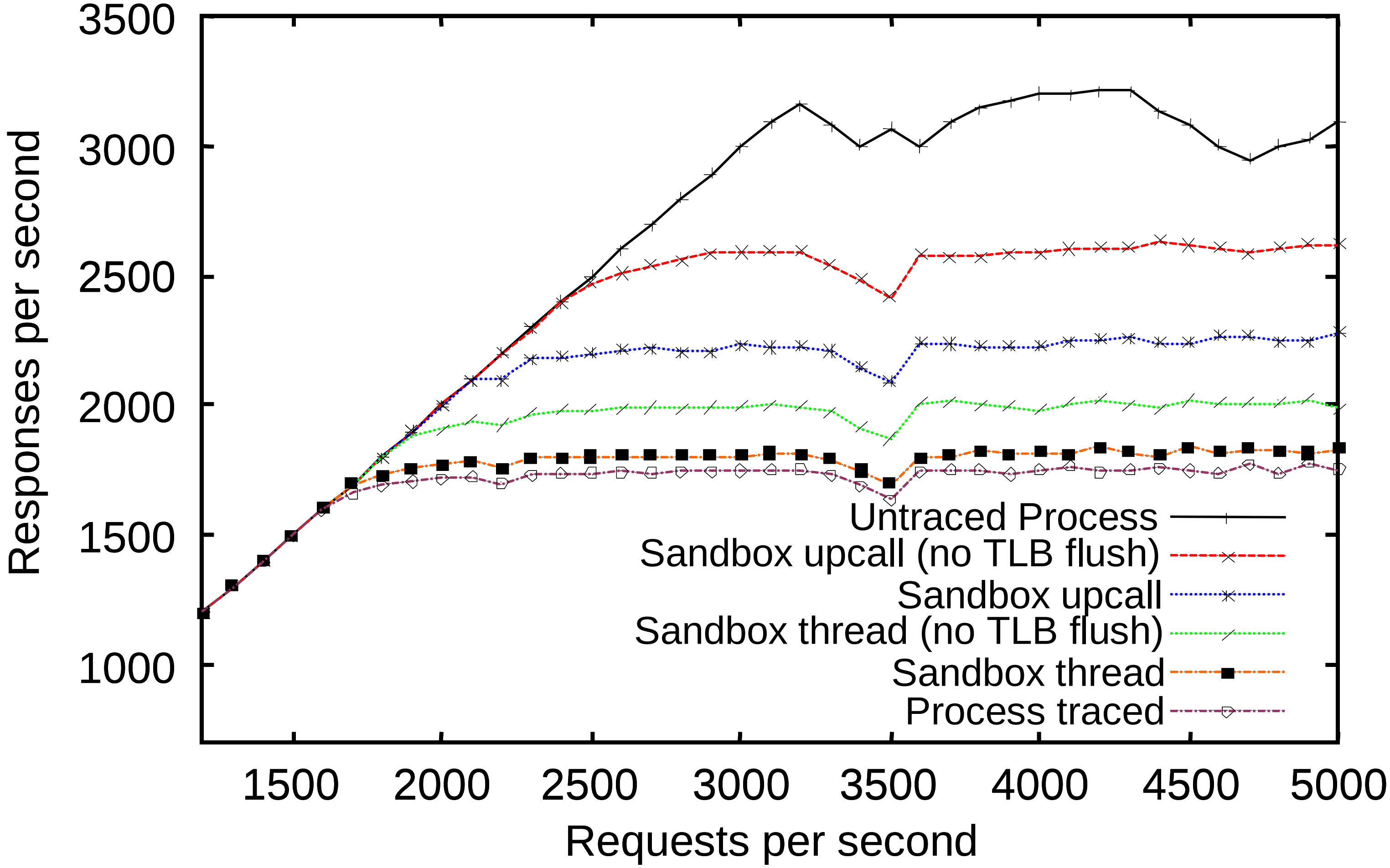
# Preliminary Performance Studies

Computer Science

- (a) Interposition
  - Simple syscall tracing extensions based on ptrace
  - Compare tradition ptrace implementation against:
    - Upcall handler implementation in sandbox
    - Kernel-scheduled thread in sandbox
- (b) Inter-Protection Domain Communication
  - Look at overheads of IPC between thread pairs
    - Exchange 4-byte messages
    - Vary the working set of one thread to assess costs

# Interposition Agents: ptrace of system calls

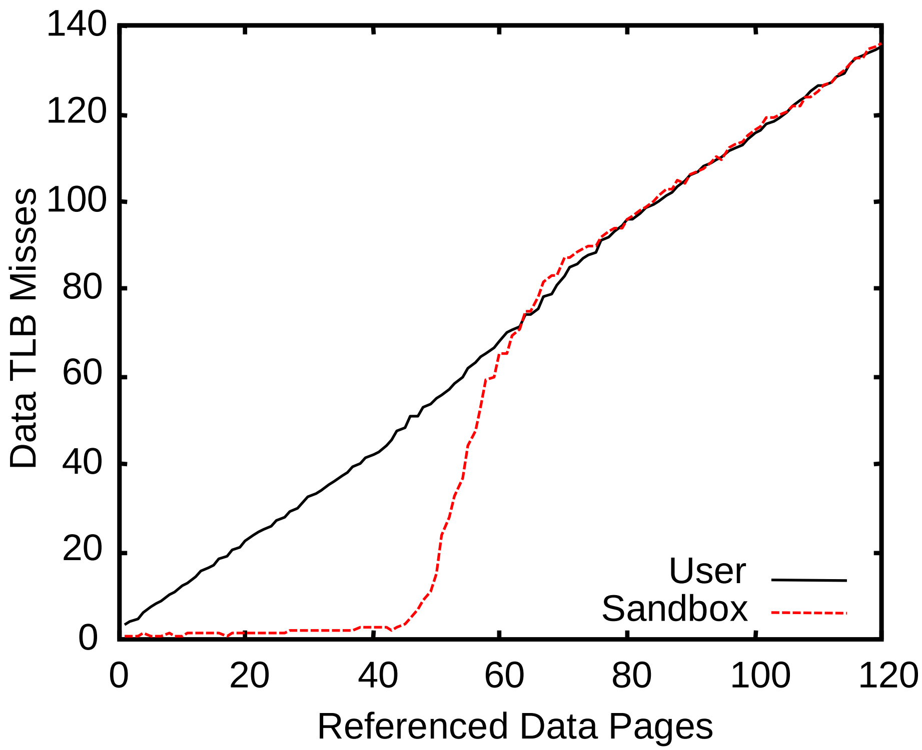
Computer Science

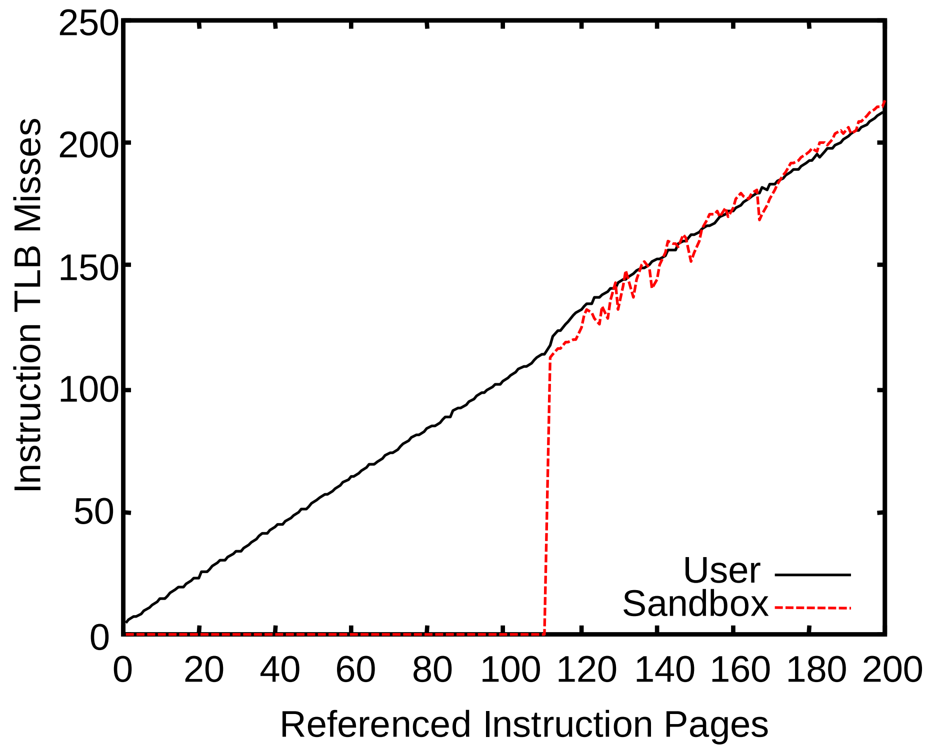



- Experiments on a 1.4GHz Pentium 4 w/ patched Linux 2.4.9
- Ptraced thttpd web server under range of HTTP request loads

# Data and Instruction TLB Misses

Computer Science

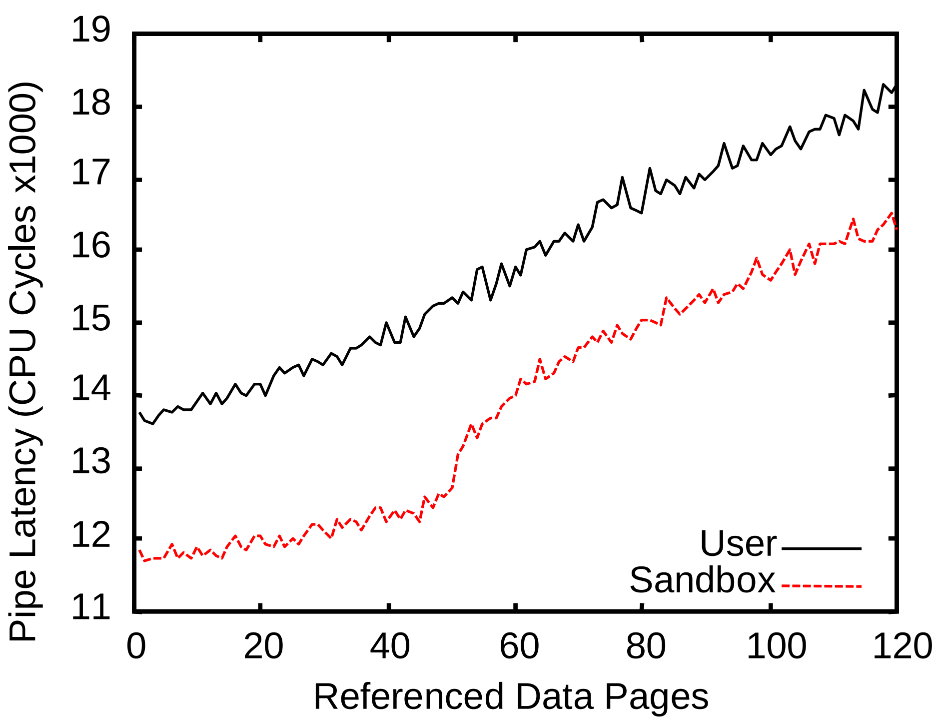


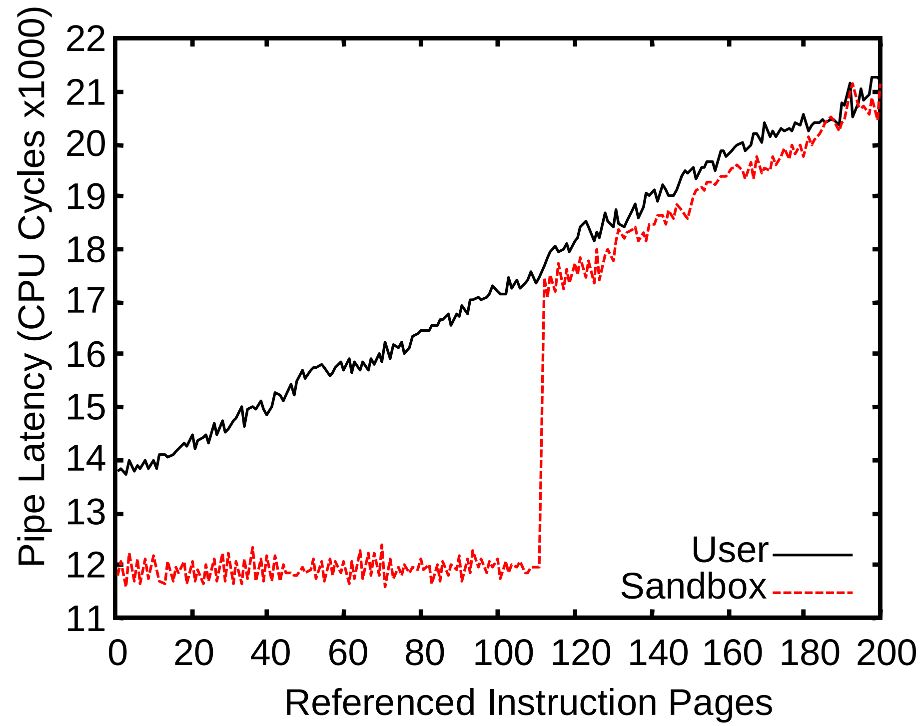




- Inter-protection domain communication costs
- Costs of 4-byte messages between two threads using pipes
- Vary working set of one process-private thread while other is in sandbox

# Pipe Latency

Computer Science




- Pipe latency remains lower for RPC with sandboxed thread
  - Even when data TLB miss rates are similar
- NOTE: d-TLB sizes simulated by thread reading 4 bytes of data from addresses spaced 4160 bytes apart. i-TLB sizes simulated using relative jumps to instructions 4160 bytes apart.

# Conclusions



- Sandboxed extensions can improve performance of traditional services (e.g., ptrace)

- IPC costs reduced due to reduction in thread context-switching overheads
  - No need to flush/reload TLB entries when switching between a sandboxed thread and process private address space

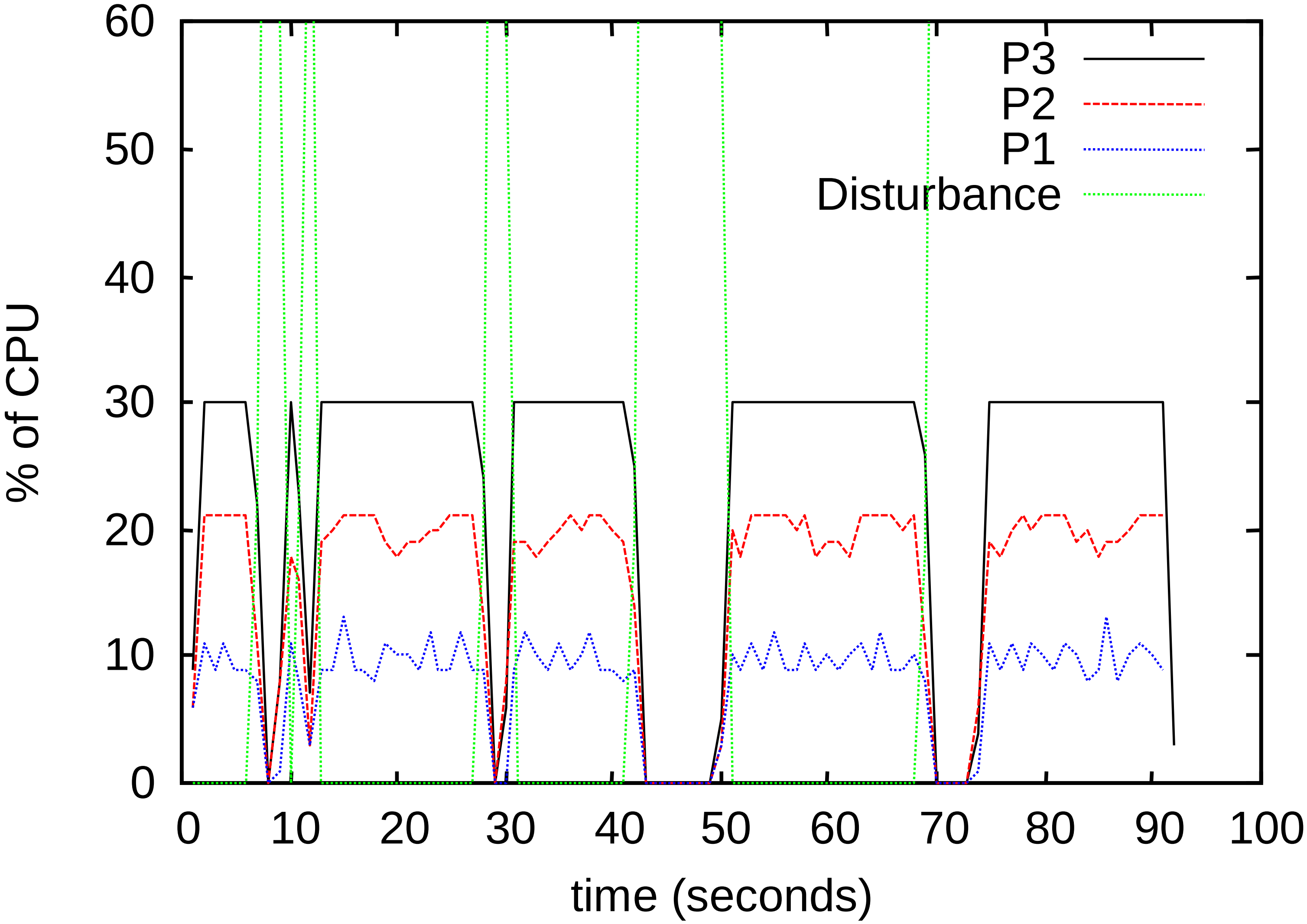
# System Service Extensions



- Can we implement system services in the sandbox?
- Here, we show performance of a CPU service manager (CPU SM)
  - Attempt to maintain CPU shares amongst real-time processes on target in presence of background disturbance
  - Use a MMPP disturbance w/ avg inter-burst times of 10s and avg burst lengths of 3 seconds
  - CPU SM runs a PID control function to adjust thread priorities

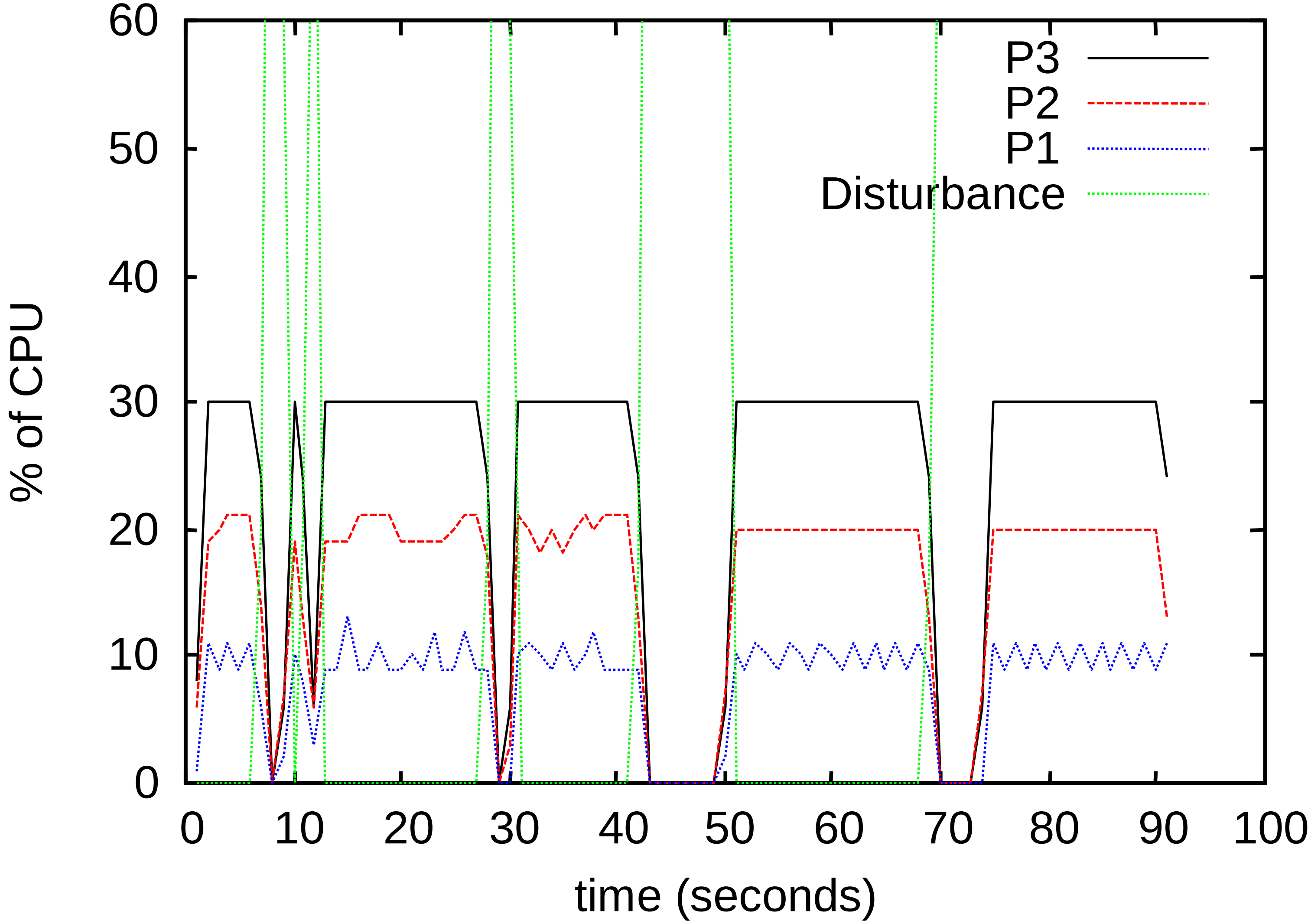
# CPU SM: User-level Process

Computer Science



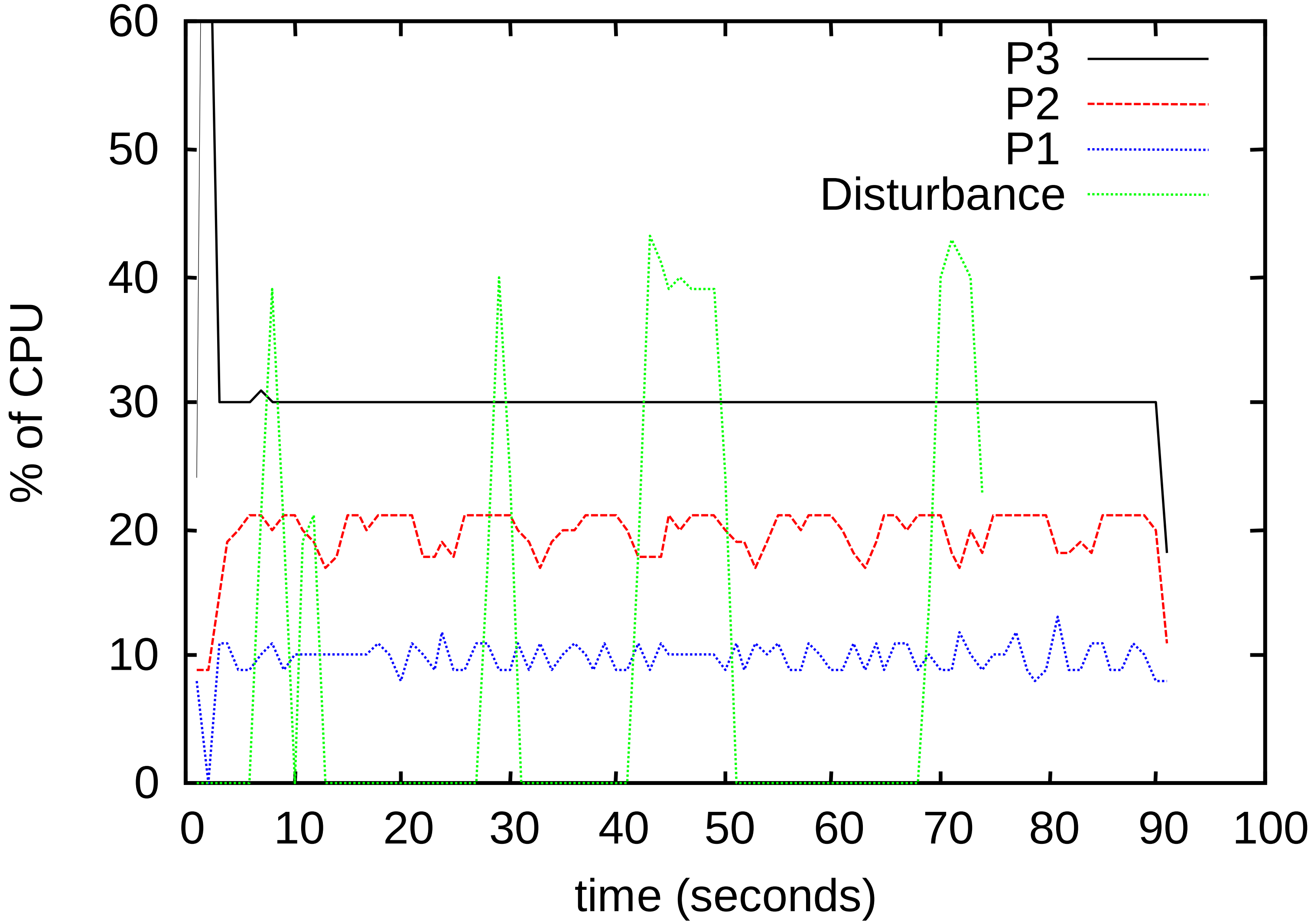
# CPU SM: Sandbox Thread

Computer Science



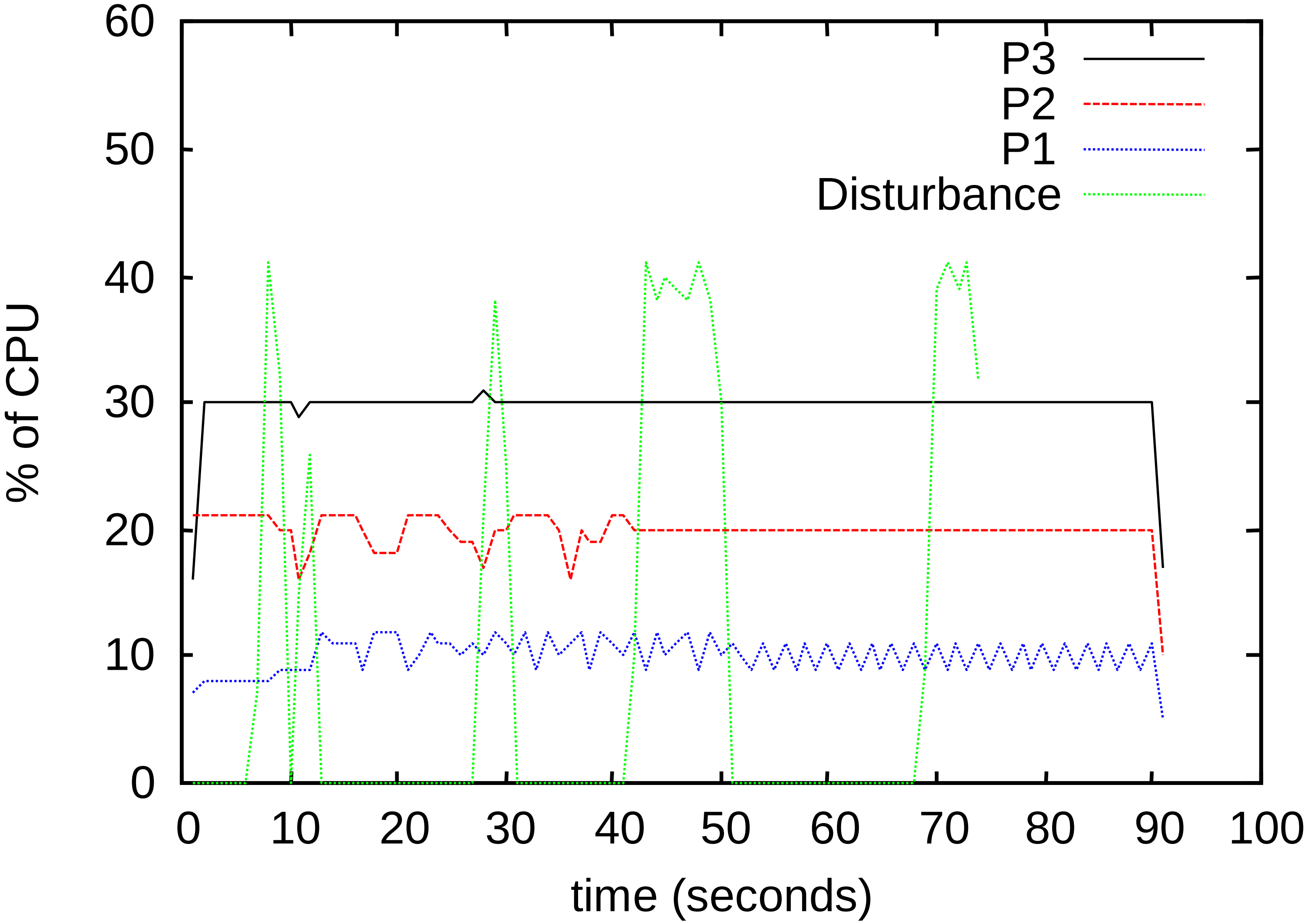
# CPU SM: Pure Upcall





# CPU SM: Kernel

Computer Science



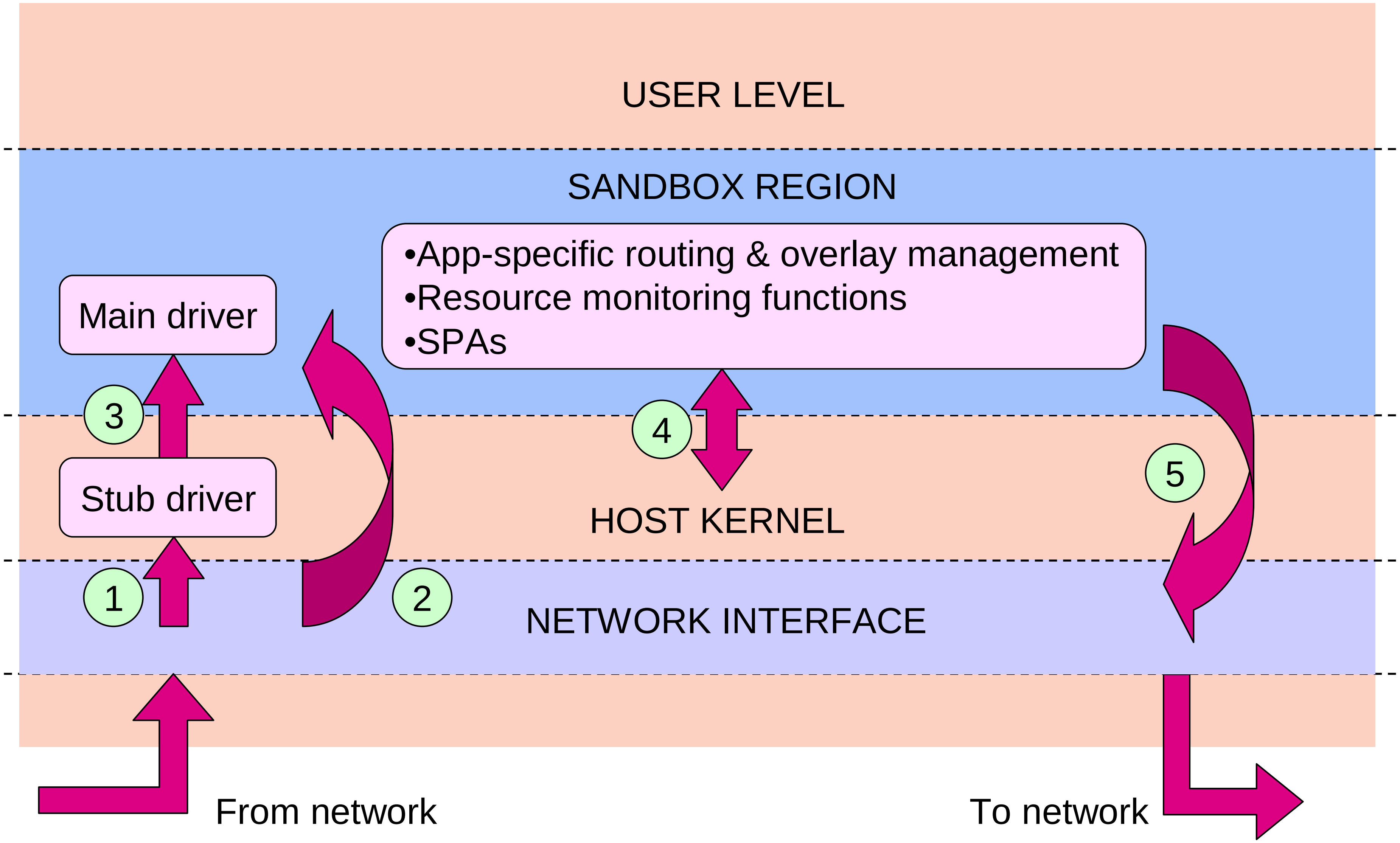
# Efficient Communications



- Aim to extend sandbox with features to allow direct access to hardware
- First step: provide support for efficient communication between sandbox and NIC
  - Avoid data copying via kernel
  - Similar to U-Net
  - Unlike U-Net, do not need special hardware for “zero copy”

# End-system Architecture





# Communication Performance



- Preliminary tests use UML to implement networking stack in the sandbox
- Results show data forwarding between socket pairs done at user-level is almost as good as using khttpd in the kernel
  - Sandboxed network protocol stack yields increased throughput compared to using UML in a traditional process

# Summary



- Aim is to use ideas from overlay routing and user-level sandboxing to implement an Internet-wide distributed system
  - Provide efficient support for app-specific services and scalable data delivery