Execution Paradigms in Mobile Computing

Computer Science, KAIST
Hwansoo Han

SIGPL Winter School 2006

Mobile Computer?

High-end PDA

CPU: Intel PXA270 624MHz
RAM: 64M DRAM

My Desktop

CPU: AMD Athlon64 X2 4400+ 2.21GHz
RAM: 1G DRAM

<<

Gap in Experience

Goal

Give mobile users desktop experience

Goal

Sorry, nearly desktop experience

How?

Call for Help through Wireless Network

Future of Mobile Devices

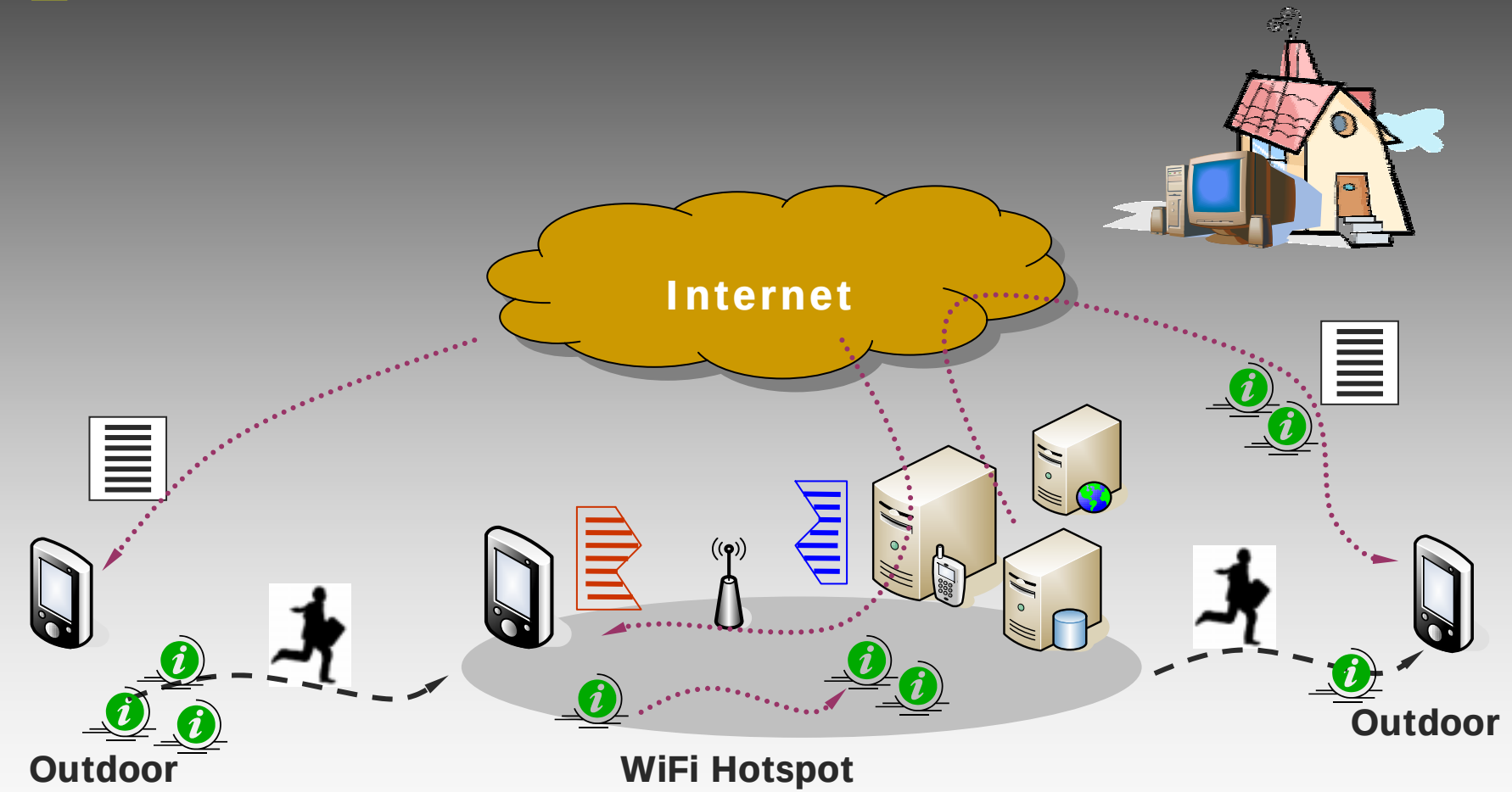
- **Increasingly, people expect more…**
  - More than just voice from their phone
  - More than just schedule management from their PDA
  - Consistent experiences across stationary and mobile devices
- **Obstacle: Resource constraints**
  - CPU, memory, battery, storage, etc.
  - Hardly upgradeable
  - It makes mobile applications rather primitive

Mobile devices need leverage to meet the expectation

Execution Paradigms

- **Surrogate computing**
  - Computation & data offloading
    - AIDE-ChaiVM [DCS'02], LSCF [WMCSA'04], SlimExecution [ISWPC'06]
  - Multiple surrogate servers including home server across WAN
    - Slingshot [MobiSYS'05]
    - Replicate all execution/events at multiple servers
    - Clients select the fastest updates
- **Thin-client computing**
  - Remote execution on server
  - Focus on user I/O (display updates & user interaction)
  - Adopt techniques in VNC, SUN Ray, Microsoft RDP, X windows
  - MobiDesk [MobiCom'04], THINC [SOSP'05]

Surrogate Computing



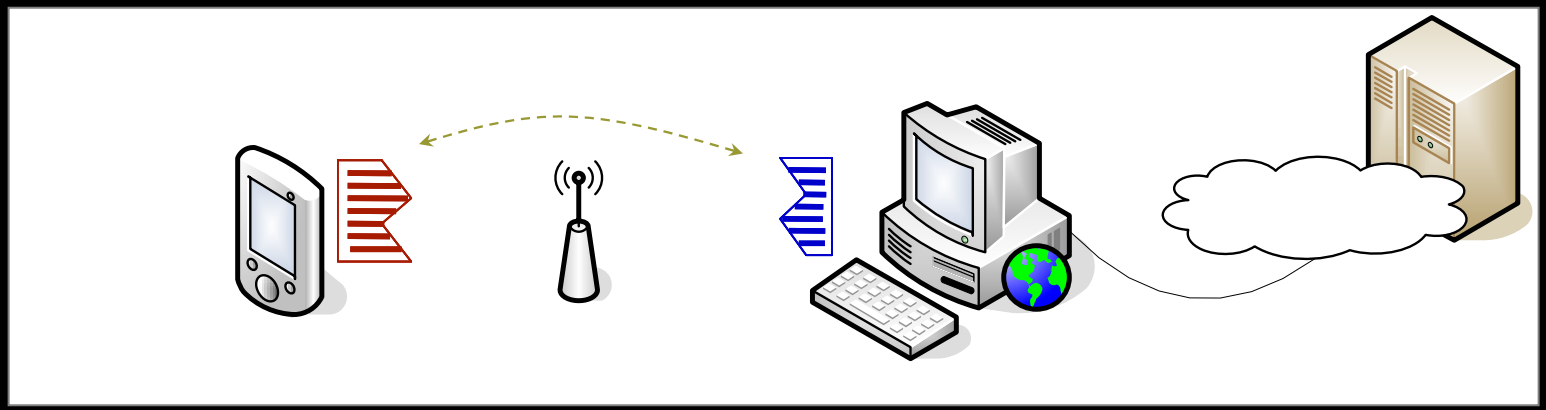
Issues

- **Partitioning**
  - Granularity
    - Code (class vs. method)
    - Data (class vs. object)
  - Function calls & data references
  - Beneficial offloading
- **Transparency**
  - Application development
  - User experience
- **Session continuation**
  - Surrogate changes & hotspot changes
  - Suspend and continue

Slim Execution [ISWPC'06]

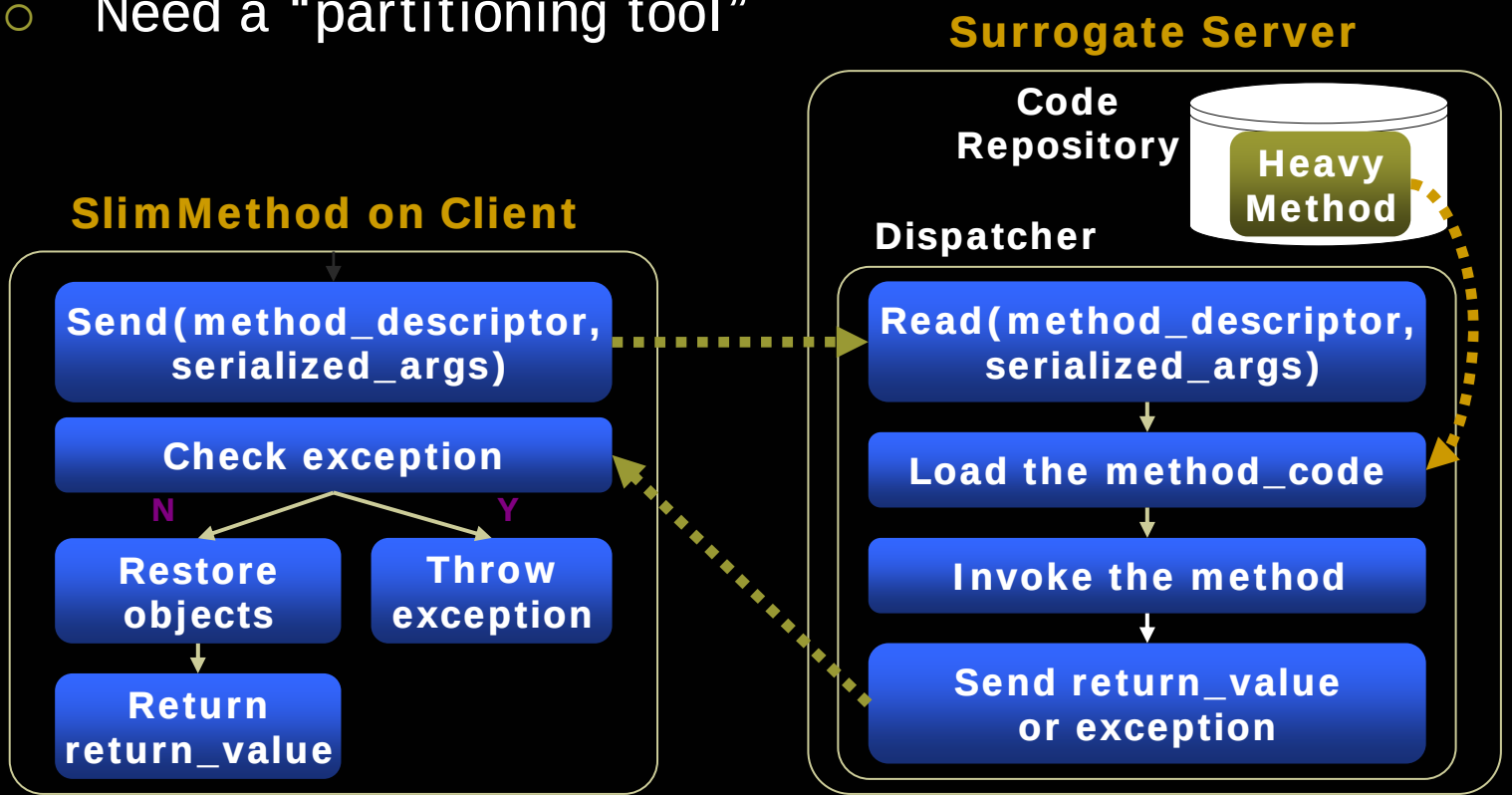
- **Idea: Call for Help through Wireless Network**
  - Offload intensive computations to surrogate server
  - Augment the resources of mobile devices

- **Slim Execution**
  - Transparent Method Offloading



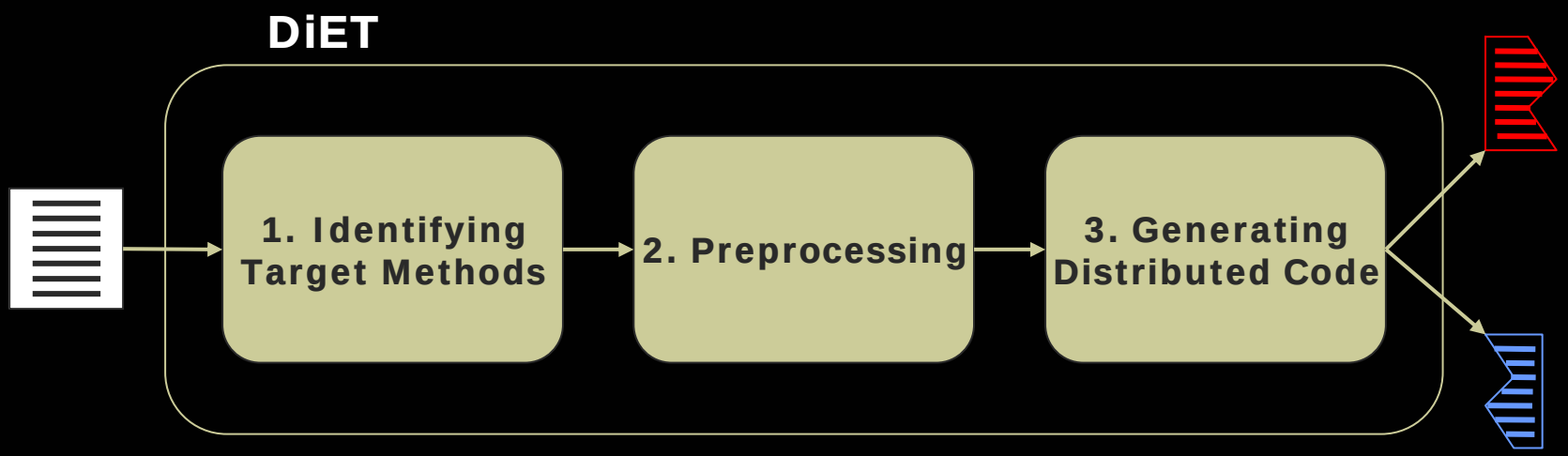
Transparent Method Offloading

- Offload computations at method level
- Without any customization on JVM
- Partition monolithic applications automatically
  - Need a “partitioning tool”



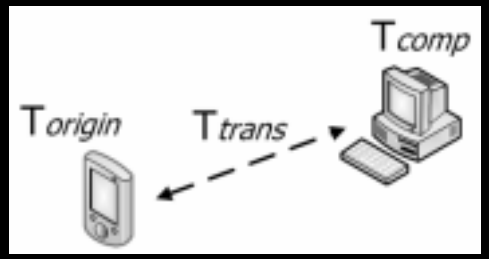
Partitioning Tool: DiET

- **Distributed Execution Transformer (DiET)**
  - Automatic distributed code generation tool
    - Statically partitions java applications
    - Resource-intensive parts are left on server
    - Using bytecode engineering library (BCEL)



Beneficial Offloading

- **Profiling**
  - Data size
  - Computation time: T_{origin} and T_{comp}
  - Network throughput



- **Method selection guide**
  - Required bandwidth test

$$T_{comp} + T_{trans} < T_{origin}$$

which yields

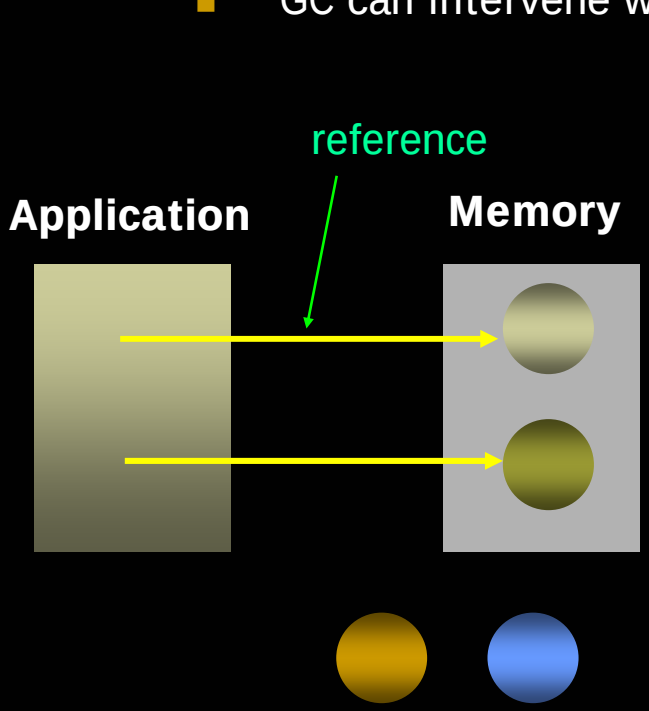
$$\frac{Data\_Size}{T_{origin} - T_{comp}} < Network\_Throughput$$

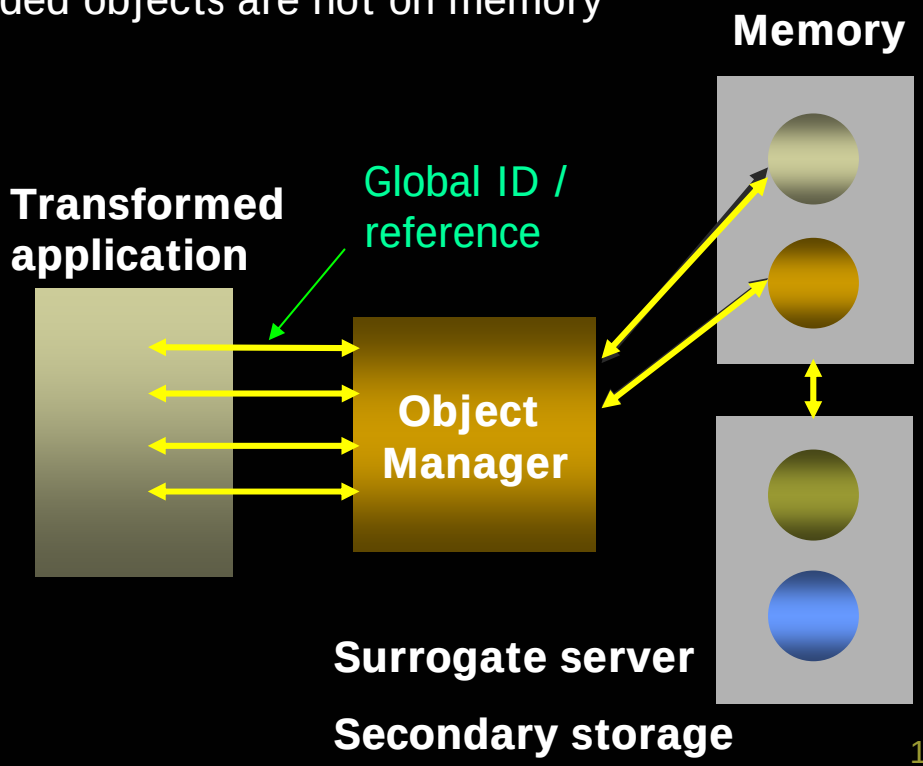
$$where, \quad T_{trans} = \frac{Data\_Size}{Network\_Throughput}$$

- **Reduce networking overhead**
  - Communication aggregation
  - Send only necessary fields not the whole object

Object Virtualization

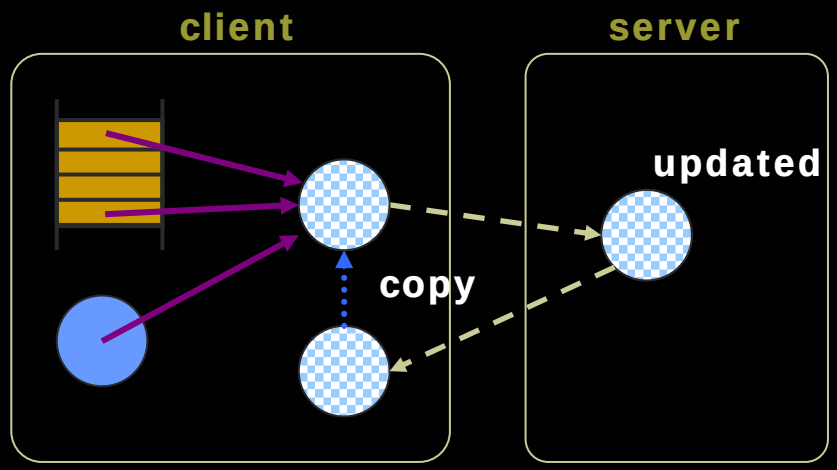
- **Object management on MMU-less devices**
  - Object on secondary storage and local server
  - Global ID and reference management
  - ChaiVM implemted with object handle
    - GC can intervene when needed objects are not on memory





Object Consistency

- **Choices for object placement**
  - Fix objects at home (client or server)
  - Distribute objects on clients and servers
- **Resolve remote references**
  - Copy necessary objects back and forth
  - Use proxy objects
- **Object Consistency**
  - Restore the objects updated in the server side
  - Copy-back updated fields

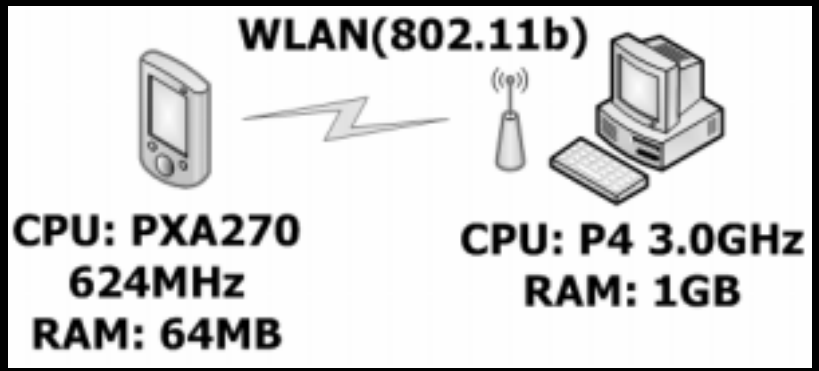


Experiment with SciMark2

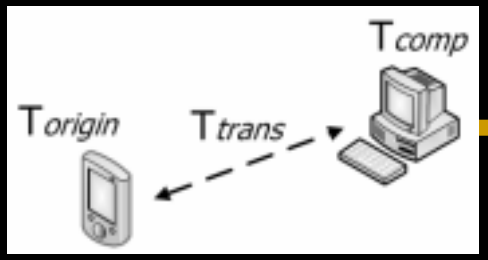
- **Scimark2**
  - Five kernels for intensive floating-point computations
  - Similar to multimedia data processing code

ID	Name	Input Size
FFT	Fast Fourier Transform	65536
SOR	Jacobi Successive Over-relaxation	256x256
MC	Monte Carlo integration	65536
SM	Sparse matrix Multiply	256x256
LU	Dense LU matrix factorization	256x256

- **Configuration**



Experimental Results



- Execution Time

Kernel	T_{origin}	Slim (T_{comp}+T_{trans})		Improved
FFT	22 sec	18 sec	0.745 + 17.618	17 %
LU	35 sec	11 sec	0.473 + 10.764	67 %
MC	1 sec	0.082 sec	0.066 + 0.016	92 %
SOR	0.4 sec	4 sec	0.004 + 4.378	-886 %
SM	0.2 sec	6 sec	0.003 + 6.075	-2746 %

- Code Size (Byte)

Original	Soot	Slim bytecode	Server bytecode
26.9 KB	15.9 KB	13 KB (-50%)	10 KB

Experimental Results

$$T_{comp} + T_{trans} < T_{origin}$$

which yields

$$\frac{Data\_Size}{T_{origin} - T_{comp}} < Network\_Throughput$$

$$where, \quad T_{trans} = \frac{Data\_Size}{Network\_Throughput}$$

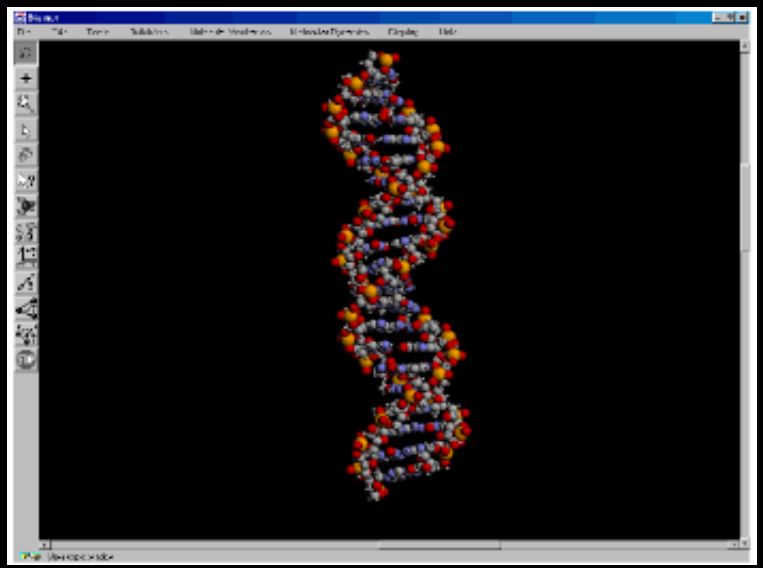
- **Required Throughput**

Kernel	Time (T_{orig} - T_{comp})	Transferred Data Size (org vs. minimized)		Required Throughput
FFT	21 sec	8 MB	8 MB	394 KBps
LU	34 sec	8 MB	6 MB	185 KBps
MC	0.9 sec	81 B	29 B	0.03 KBps
SOR	0.4 sec	4 MB	2 MB	4,773 KBps
SM	0.2 sec	6 MB	3 MB	15,222 KBps

(Network Throughput: 400 ~ 500KBps)

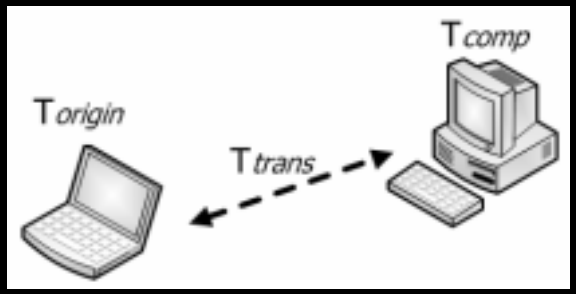
Experiment with Biomer

- Biomer: bio-molecular modeling package
  - Interactive modeling and simulated annealing
- Memory-intensive Java application



Experimental Results

- **Execution Time (sec)**
  - JVM heap size: 64 MB
  - Transferred data size: 7 MB → 6.9 MB
  - Required throughput: 38 KBps
  - Network throughput: 400 ~ 500 KBps



Original (T_{origin})	Slim (T_{comp} + T_{trans})	Improved
213 sec	44 (29 + 15) sec	79 %

- **Code Size (Byte)**

Original	Soot	Slim bytecode	Server bytecode
636 KB	476 KB	481 KB (-24%)	40.5 KB

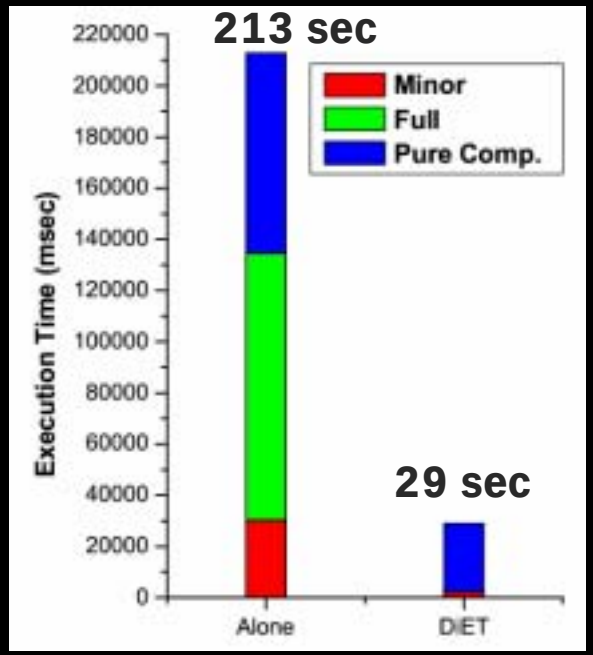
Experimental Results

- Garbage collection time (sec) and frequency

	Minor		Full		Total comp.
Alone	30 sec	4396	104 sec	1320	213 sec
DiET	2 sec	543	0.1 sec	2	29 sec

- Component ratio

	Minor	Full	Pure comp.
Alone	14 %	49 %	37 %
DiET	8 %	0.1 %	91.9 %

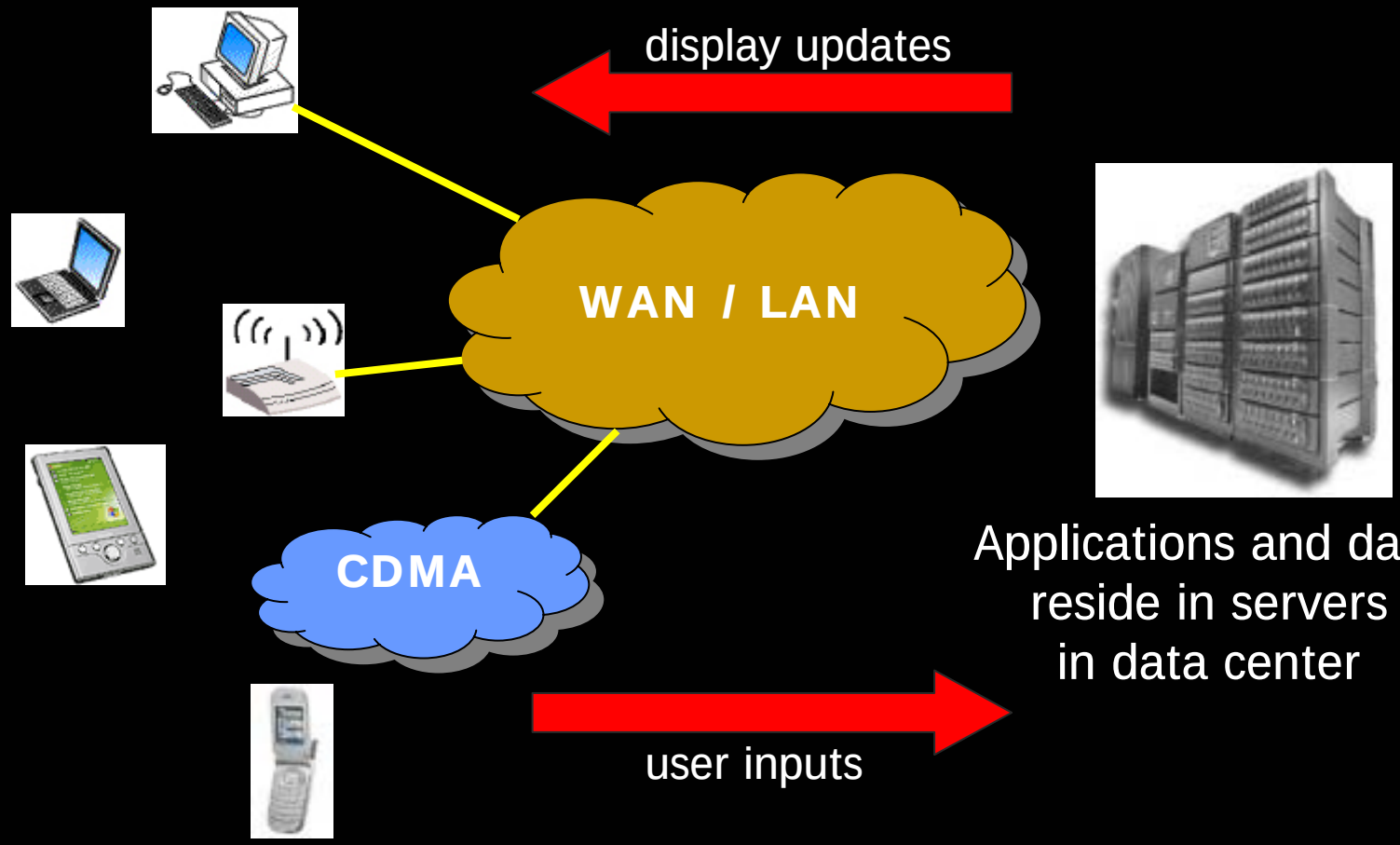


Conclusions for Surrogate Computing

- **Transparent Method offloading**
  - Relieves mobile devices of resource constraints
    - 43~79% performance improvement for heavy computations
    - 24~50% reduction in code size
  - Enables more various services for mobile devices

- **Transparency provided by**
  - Automatic distributed code generation
  - For performance boost, many necessary analyses needed

Thin-Client Computing



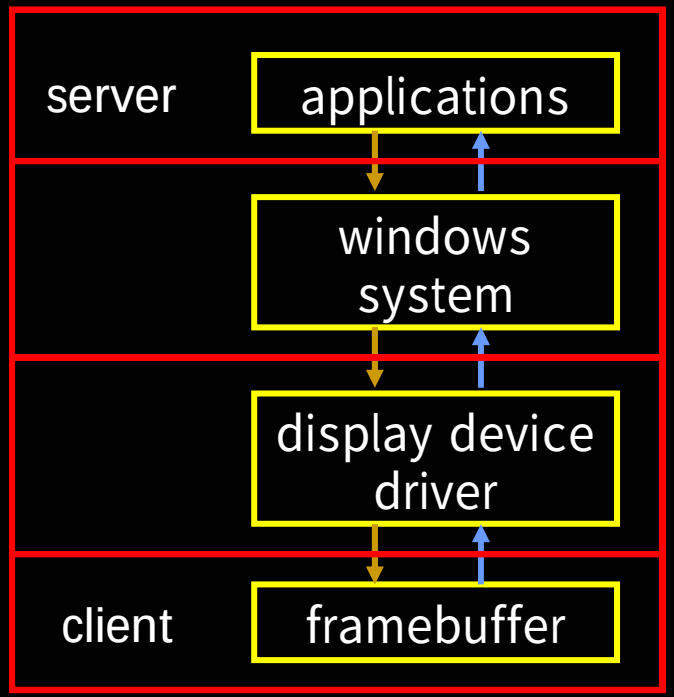
THINC [SOSP'05]

- **Benefits of THINC (Thin-client Internet Computing)**
  - Good solution to management and security issues
    - Almost no maintenance to clients
    - Not storing sensitive data on mobile clients
  - Stateless client can provide persistent, personalized computing environment
  - Sharing computing resources among many users
  - Adopt techniques in Microsoft RDP, Sun Ray, VNC, X windows,

- **High fidelity visual and interactive experience**
  - Previous goal was running office productivity tools in LAN
  - Current goal is web browsing, video application in WAN
  - In the future, interactive 3D graphics in LAN/WAN

Execution of GUI Applications

- **High level GUI APIs on client**
  - UI and application logic need synch.
  - GUI system is large and complex
  - Could be bandwidth efficient
    - Using rich, complex display primitives
- **Lower level APIs on client**
  - No synchronization overhead
  - No GUI software on clients
  - Could be bandwidth intensive
    - Display command compression on servers



Variety of Thin Clients

- **RDP, MetaFrame, Tarantella**
  - Use rich sets of low-level graphics commands (similar to X)
  - Not necessarily provide substantial gains in bandwidth
    - Particularly on multimedia content
    - Complexity results in slower responsiveness, performance degradation in WAN
- **Sun Ray, THINC**
  - Translate into simple 2D drawing primitives
  - Difficult to translate effectively
- **VNC, GoToMyPC, PC Anywhere**
  - Reduce everything to raw pixel
  - Need encoding or compression

THINC Architecture and Protocol

- **Virtualize video device layer**
  - Intercept applications' drawing commands, encode them using THINC display commands , and send them over the network
  - Can use existing display system functionality
  - Seamlessly work with existing applications, window systems, and OS
  - Support for graphics acceleration architectures (XAA, KAA, EXA)

command	description
RAW	Display raw pixel data at a given location
COPY	Copy frame buffer area to specified coordiantes
SFILL	Fill an area with a given pixel color value
PFILL	Tile an area with a given pixel pattern
BITMAP	Fill a region using a bitmap image

THINC Display Commands (similar to SUN Ray)

Translation Layer

- **Translates display commands as they occur and efficiently encode them into THINC's commands**
  - More efficient translation than raw pixel transfer
- **Decouple display command translation and network transmission**
  - Aggregate small display updates to large one
  - Take advantage of communication aggregation
- **Maintain a command queue for a display region**
  - Can process multiple updates to the same region as a single entity
  - Can evict irrelevant updates due to overwriting updates
    - Distinguish them as complete, partial, transparent overwrites
- **Support for offscreen drawing, audio/video applications**

Command Delivery

- **Command buffer per-client consists of command queues**
  - 10 queues with power of 2 data sizes
  - Shortest-Remaining-Size-First for minimum mean response time
  - Preemptive schedule based on remaining size to send
  - Real-time queue for high interactivity needs
- **Client-side need correct display updates**
  - Update commands are delivered out of order
  - Dependency checks
    - with consideration of full/partial/transparent overlaps
- **Server-push compared to client-pull**
  - No round trip delay for minimum display response time
  - Could congest network
    - Servers need to block sending but buffer updates

Experimental Results

- **THINC server (Linux, virtual video device driver in X)**
- **THINC client**
  - X application, Java client, window client, window PDA client
- **Three configurations for clients**
  - LAN Desktop: 1024*768, 100Mbps network
  - WAN Desktop: 1024*768, 100 Mbps with 66ms RTT network
  - 802.11g PDA: 320*240, 24Mbps network

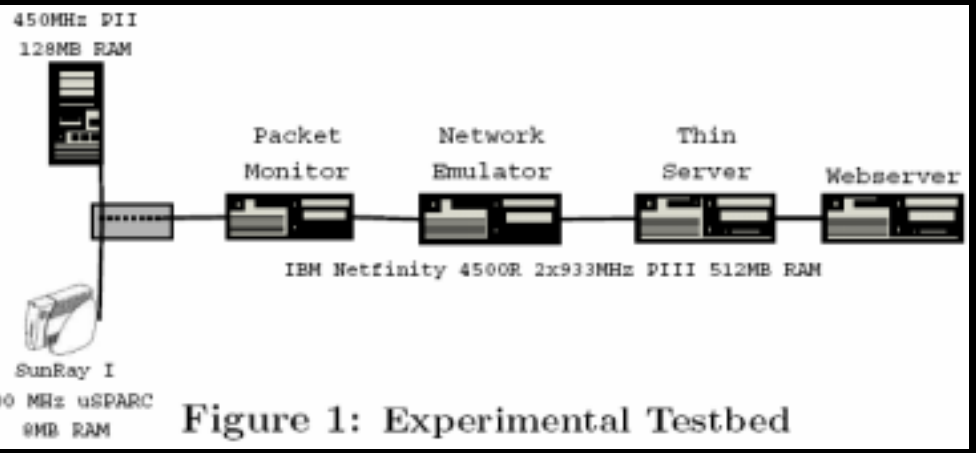


Figure 1: Experimental Testbed

Name	PlanetLab	Location	Distanc
NY	yes	New York, NY, USA	5 mile
PA	yes	Philadelphia, PA, USA	78 mile
MA	yes	Cambridge, MA, USA	188 mile
MN	yes	St. Paul, MN, USA	1015 mile
NM	no	Albuquerque, NM, USA	1816 mile
CA	no	Stanford, CA, USA	2571 mile
CAN	yes	Waterloo, Canada	388 mile
IE	no	Maynooth, Ireland	3185 mile
PR	no	San Juan, Puerto Rico	1603 mile
FI	no	Helsinki, Finland	4123 mile
KR	yes	Seoul, Korea	6885 mile

Experimental Results (Web)

- **Browse a sequence of 54 web pages**
  - Client processing time (slant pattern)
  - Download time (solid)
- **Data transferred**
  - GoToMyPC use aggressive compression sacrificing long latency
  - VNC, Sun Ray use adaptive compression scheme

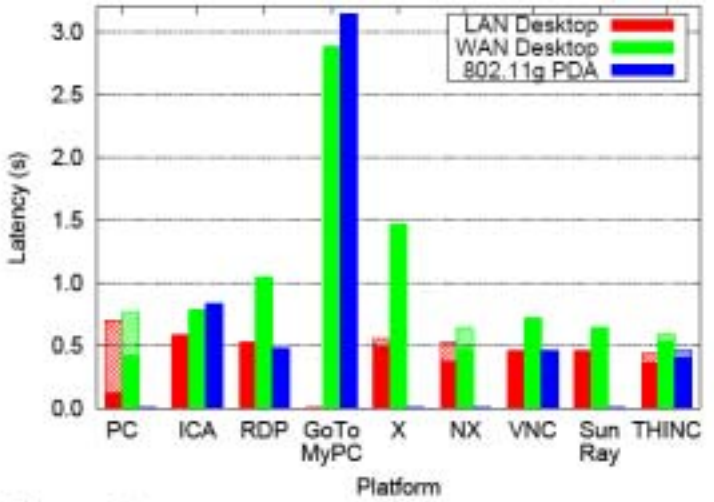


Figure 2: Web Benchmark: Average Page Latency

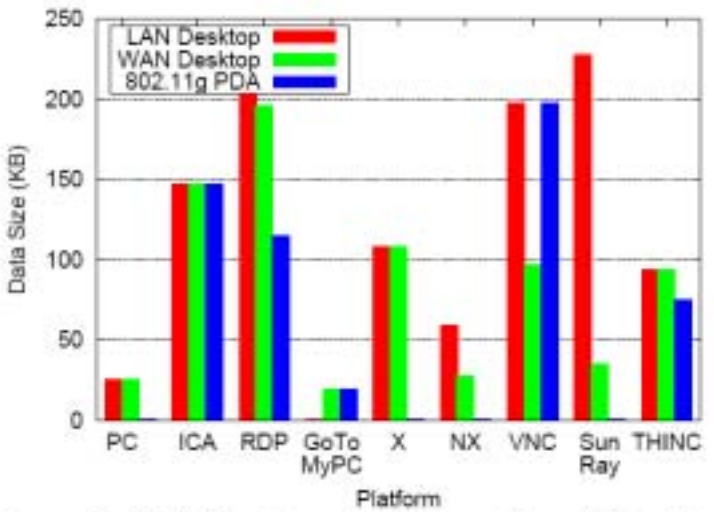
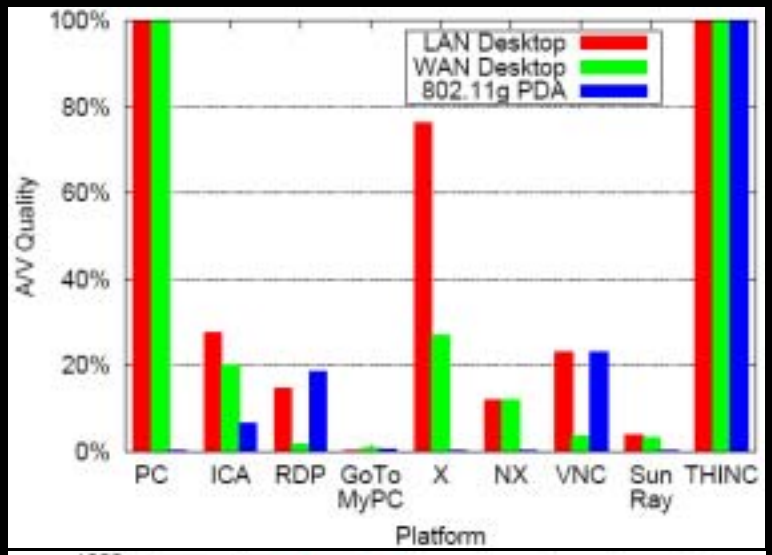


Figure 3: Web Benchmark: Average Page Data Transferred

Experimental Results (A/V app)

- **Video quality preserved only in THINC**
  - Other thin-clients cannot distinguish video updates
  - THINC uses YUV encoding that can be directly handed to client video H/W
    - YV12 encoding in MPEG efficiently compress RGB data w/o loss of quality for human eyes
    - No color space conversion in S/W
    - No scaling in S/W



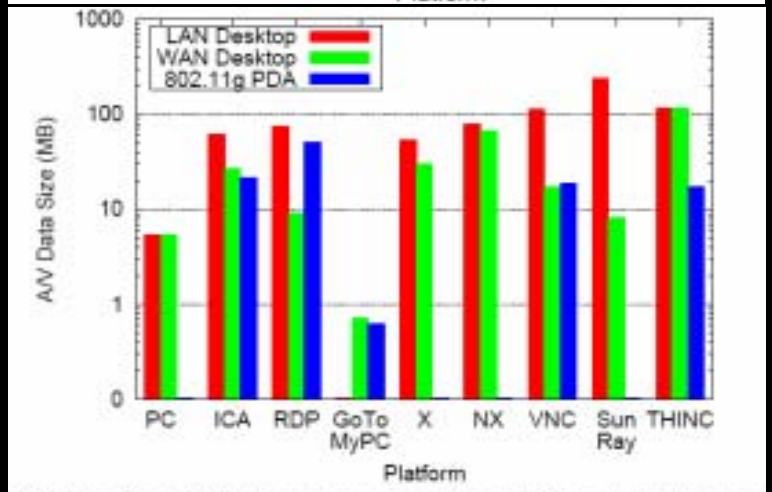


Figure 6: A/V Benchmark: Total Data Transferred (GoToMyPC and VNC are video only)

Conclusions for Thin-Client Comp.

- **Thin-client computing is one way to provide better computing experience in network computing**
  - Including mobile network, WAN, as well as LAN
- **Sophisticated decision needed where we separate graphic-rich applications**
  - Window system vs. device driver vs. framebuffer
  - Hybrid approach depends on applications?
    - E.g. special treatment to distinguish video data
- **Still need more study for 3D interactive applications**