The Case for Cyber Foraging

Rajesh Krishna Balan

Carnegie Mellon University

Joint work with:

J. Flinn (Univ. Of Michigan),

M. Satyanarayanan (CMU),

S. Sinnamohideen (CMU) ,

H. Yang (CMU)

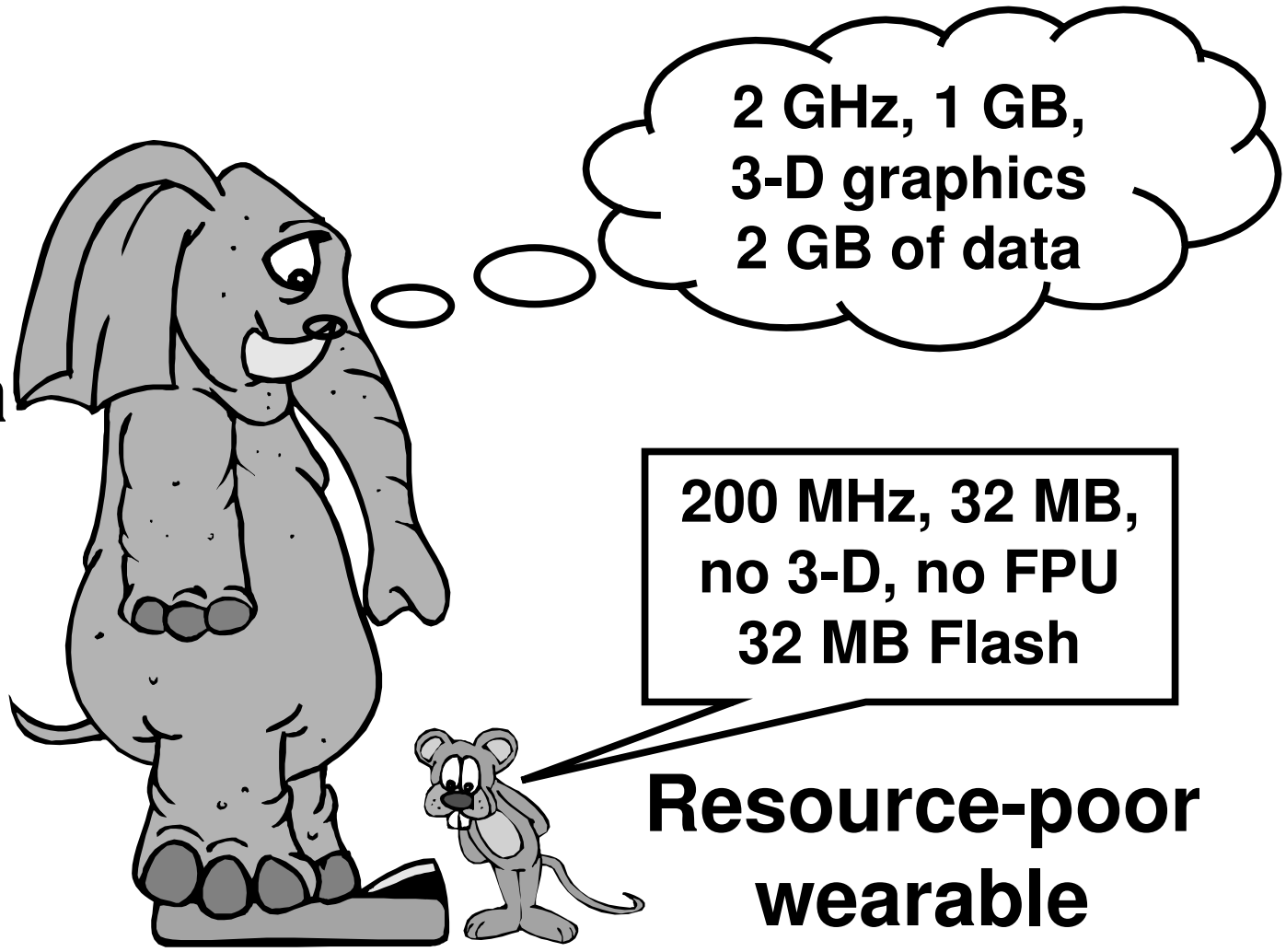
2 Broad class of applications

- Personal Productivity Applications
  - Email
  - Calendar
- Computationally Intensive Interactive Applications
  - Speech Recognition
  - Language Translation
  - Augmented Reality

Motivation: Handhelds are weak!

- Resource intensive App

- Huge Data Sets



Poor performance!

Solution: Cyber Foraging

- "To live off the land"
- Use resources in environment to augment device capabilities by using surrogates
- 2 methods
  - Data Staging
  - Remote Execution

The Big Picture

- Data Staging
  - Caching of large amounts of data
  - Handhelds with limited storage can access this data fast
  - Security and authentication
- Remote Execution
  - Uses remote servers to augment computational capabilities of handhelds
  - Enables computationally intensive applications
- Service Discovery
  - Discover servers used by previous two mechanisms

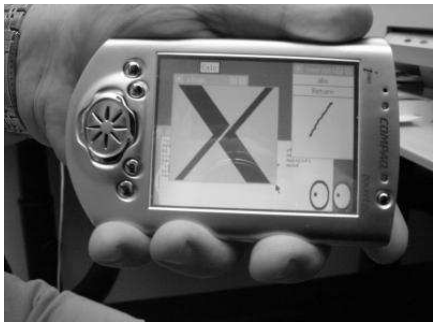
Roadmap

- Data Staging
- Remote Execution
- Service Discovery

Data Staging: Motivation

- End-to-end latency across the Internet isn't getting better
  - Physical limits
  - Routers, firewalls
  - Shows up in interactive file access delays
  - Crucial for small to medium files
- Can overcome this by caching & prefetching, but …
  - Handheld clients don't have enough resources
  - Cache consistency
- *Can untrusted and unmanaged computers help?*

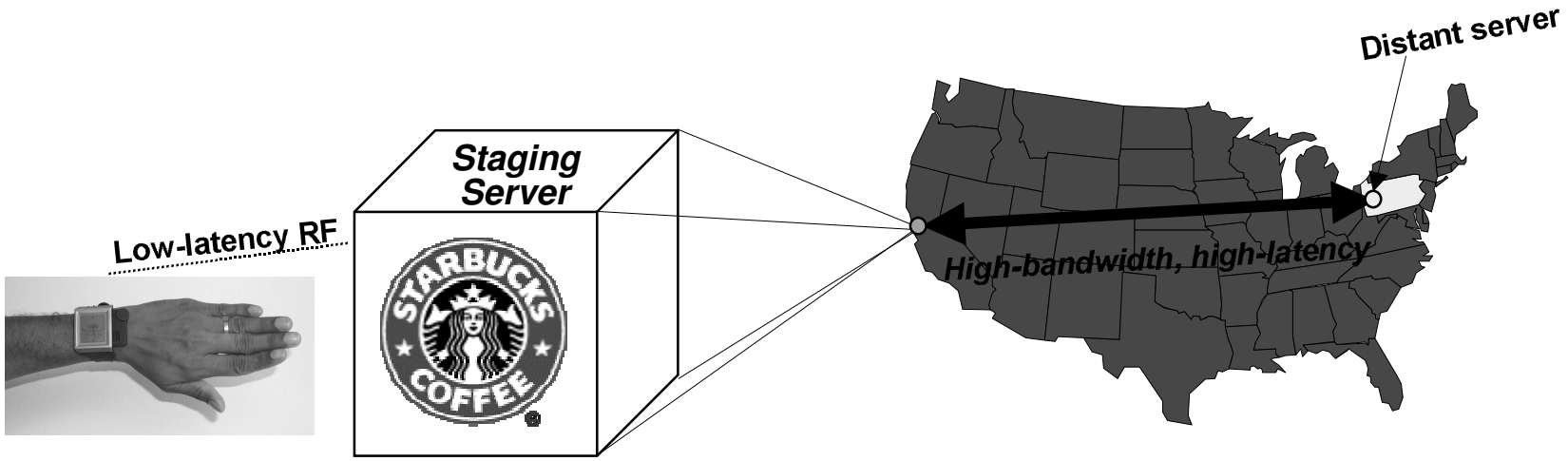
Yes!!



Data staging: Mechanism

- Coda clients speculatively prefetch data :
  - Nearby surrogate runs staging server
  - Used like a second level cache
  - Cache misses serviced by staging server

- Surrogates deployed in high-usage areas



Security

- Must provide level of security users expect
- *But surrogate is untrusted*

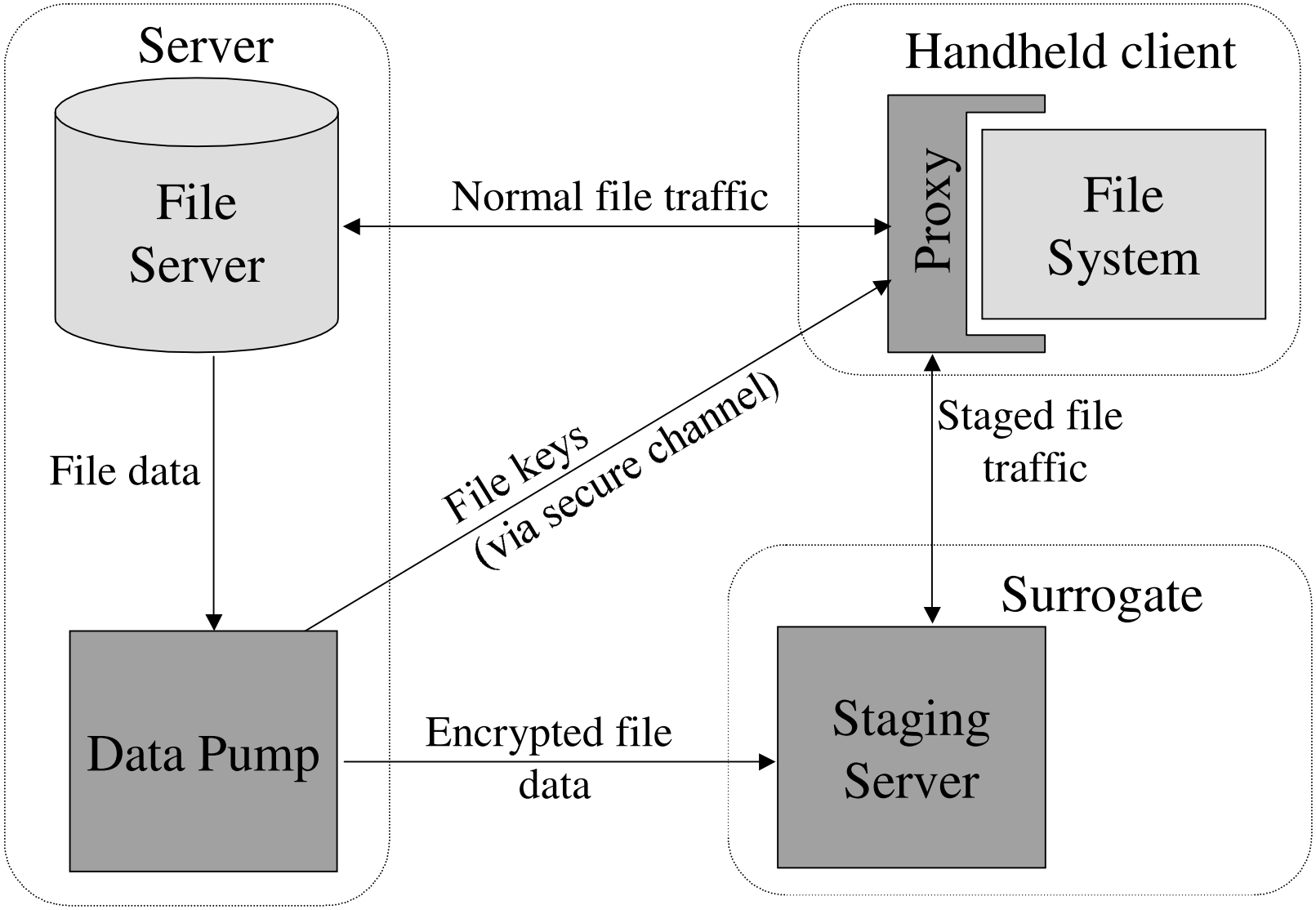
- Use end-to-end encryption
  - Only store encrypted data on surrogate
  - Client caches keys and checksums
  - Only need access control for keys

Ease of management

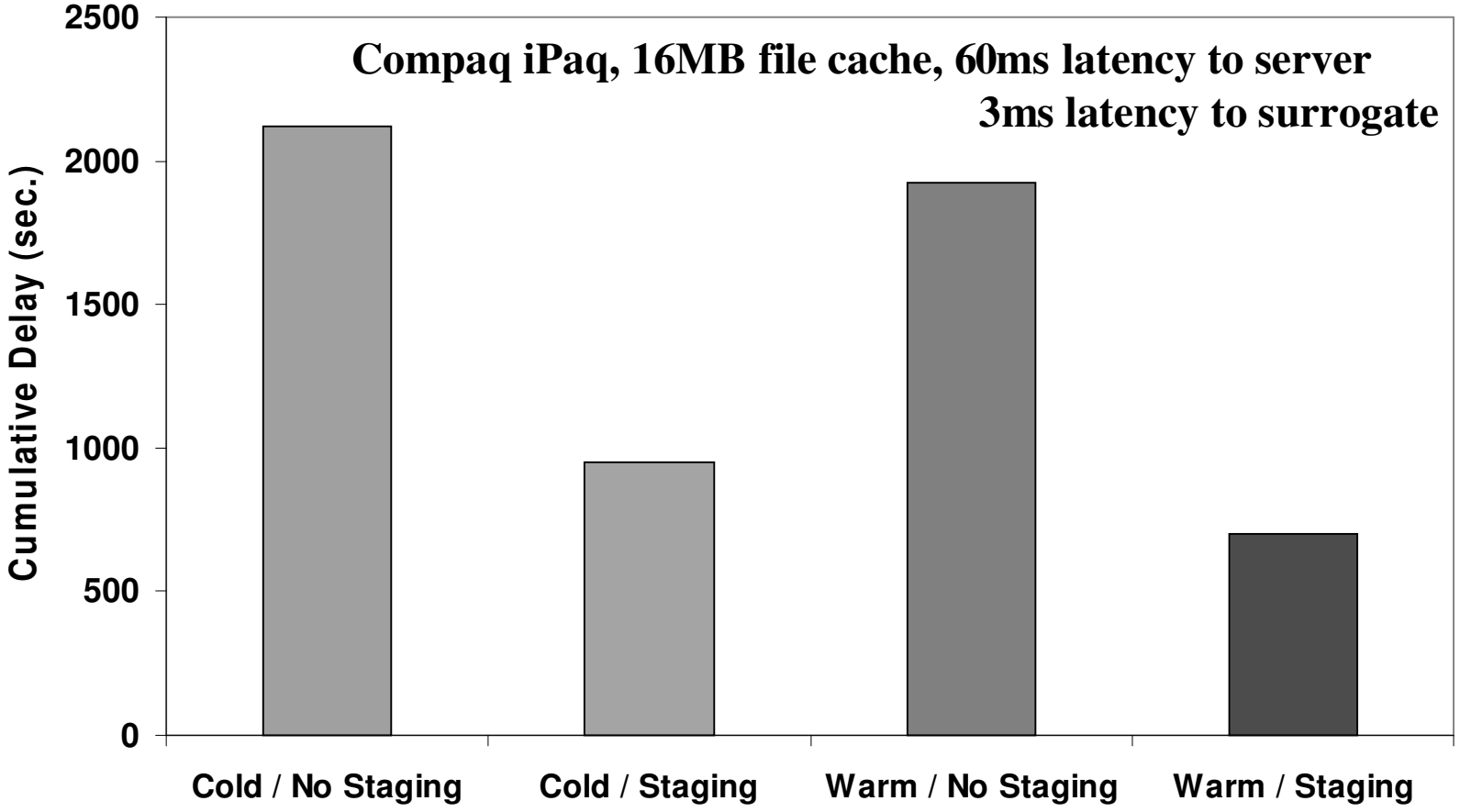
- Surrogate is simple to manage and deploy
  - No per-client persistent state on surrogate
  - Consistency maintained by client
  - Minimalistic set of operations
  - Uses commodity software (Apache)

- No modifications to base filesystem
  - Proxy based approach
  - Proxy encapsulates FS specific code

The "Gory" Details



Benefit for image viewing



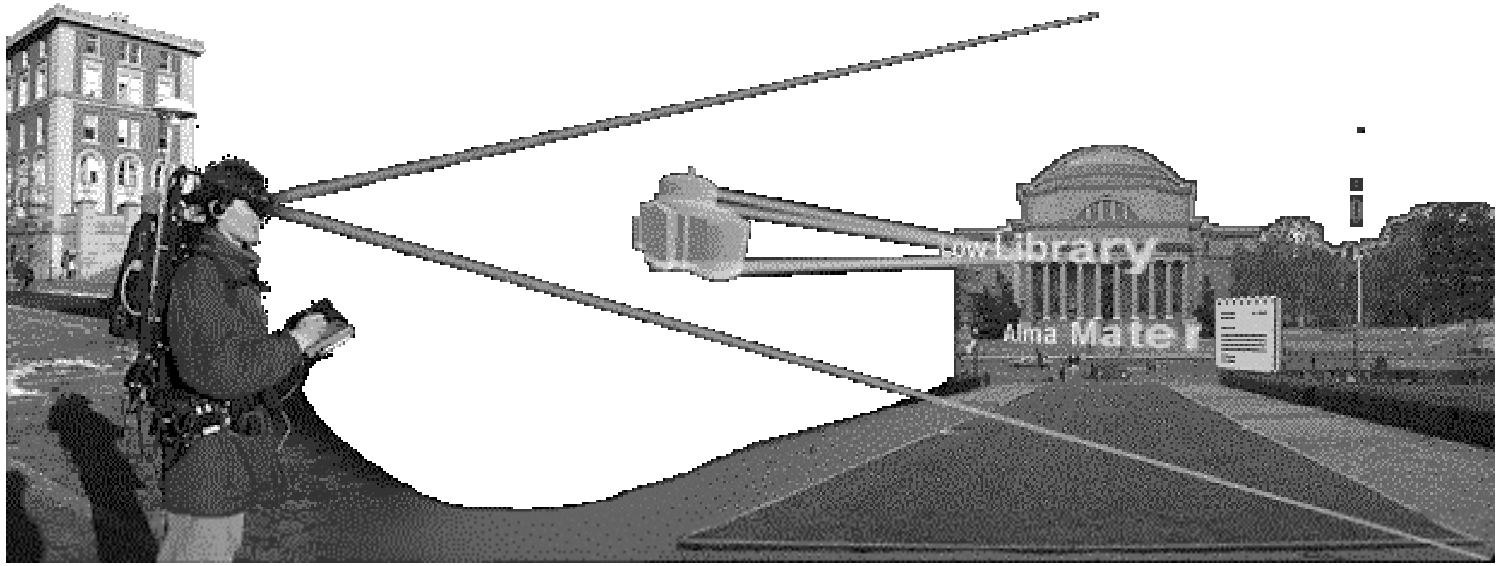
Data staging reduces cumulative delay up to 64%

Roadmap

- Data Staging
- Remote Execution
- Service Discovery

Motivation: mobile interactive applications

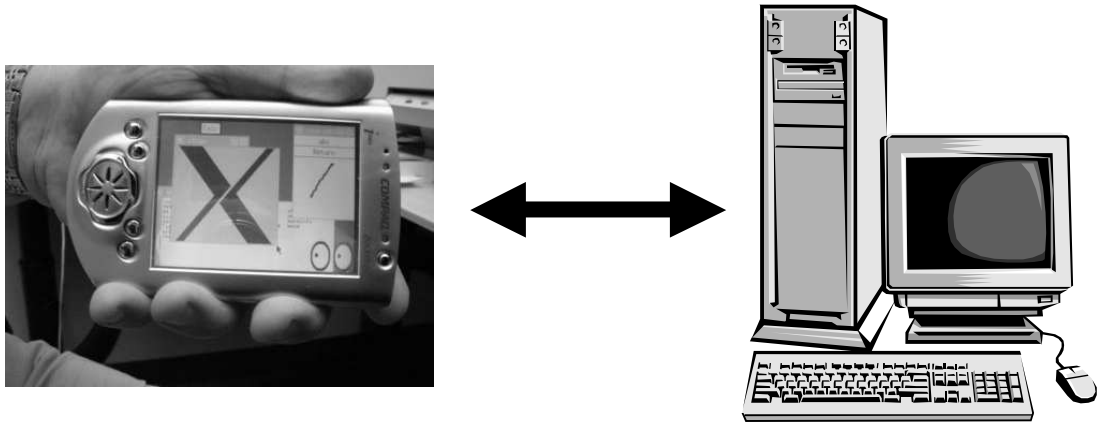
- speech recognition, language translation, augmented reality, …
  - Resource-heavy, but need bounded response time



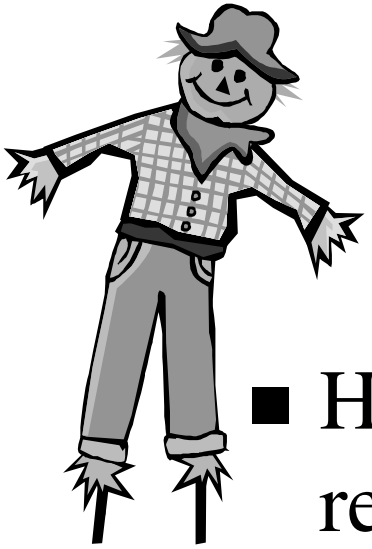
Columbia U. MARS project

Solution: Remote Execution

- Augment capabilities of handhelds by using nearby servers



- But how do you make legacy applications use remote execution?
- And get good performance as well?



Strawman Solution

- Heavily modify each application to use remote execution
  - Tweak every last drop of performance
- Requires ~3- 4 grad student months per reasonably sized application
  - Grad students have nothing else to do anyway right?? ☺
- Method does not scale and is not agile

Solution: Tactics

- Concise description of application's remote execution capabilities
  - Only the useful remote partitions are described
  - Can be captured in a compact declarative form
  - Allows use of stub generators to ease programming burden
- Tradeoff between code migration and static partitioning

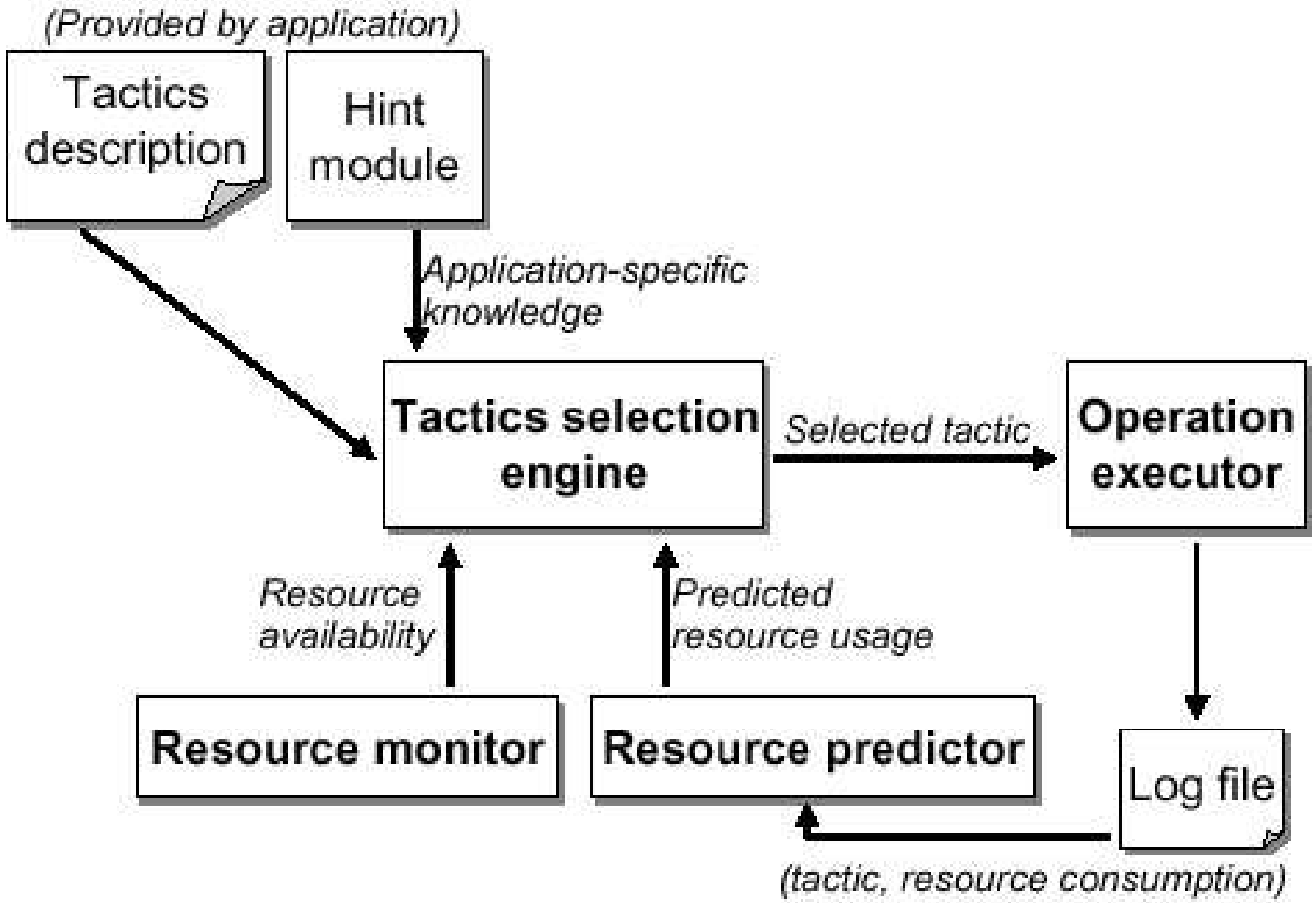
Example Tactic

```
APPLICATION pangloss-lite;

/* RPC Specifications */
RPC server_dict    (IN string line, OUT string dict_out);
RPC server_ebmt  (IN string line, OUT string ebmt_out);
RPC server_lm      (IN string gloss_out, IN string dict_out,
                     IN string ebmt_out, OUT string translation);

/* Tactics (Useful Ways to Combine the RPCs) */
TACTIC dict            = server_dict & server_lm;
TACTIC ebmt            = server_ebmt & server_lm;
TACTIC dict_ebmt    = (server_dict, server_ebmt) & server_lm;
```

Chroma


(Provided by application)
Tactics description
Hint module
Application-specific knowledge
Tactics selection engine
Selected tactic
Operation executor
Resource availability
Predicted resource usage
Resource monitor
Resource predictor
Log file
(tactic, resource consumption)

But is it FAST??

- Yes!!
- Tactics capture useful remote partitions of application
  - No time spent looking at non-useful partitions
- Performance is comparable to hand-modified application
  - Constant overhead added by runtime to best statically decided case
  - Overhead is low and reasonable

Adjusting to Environment

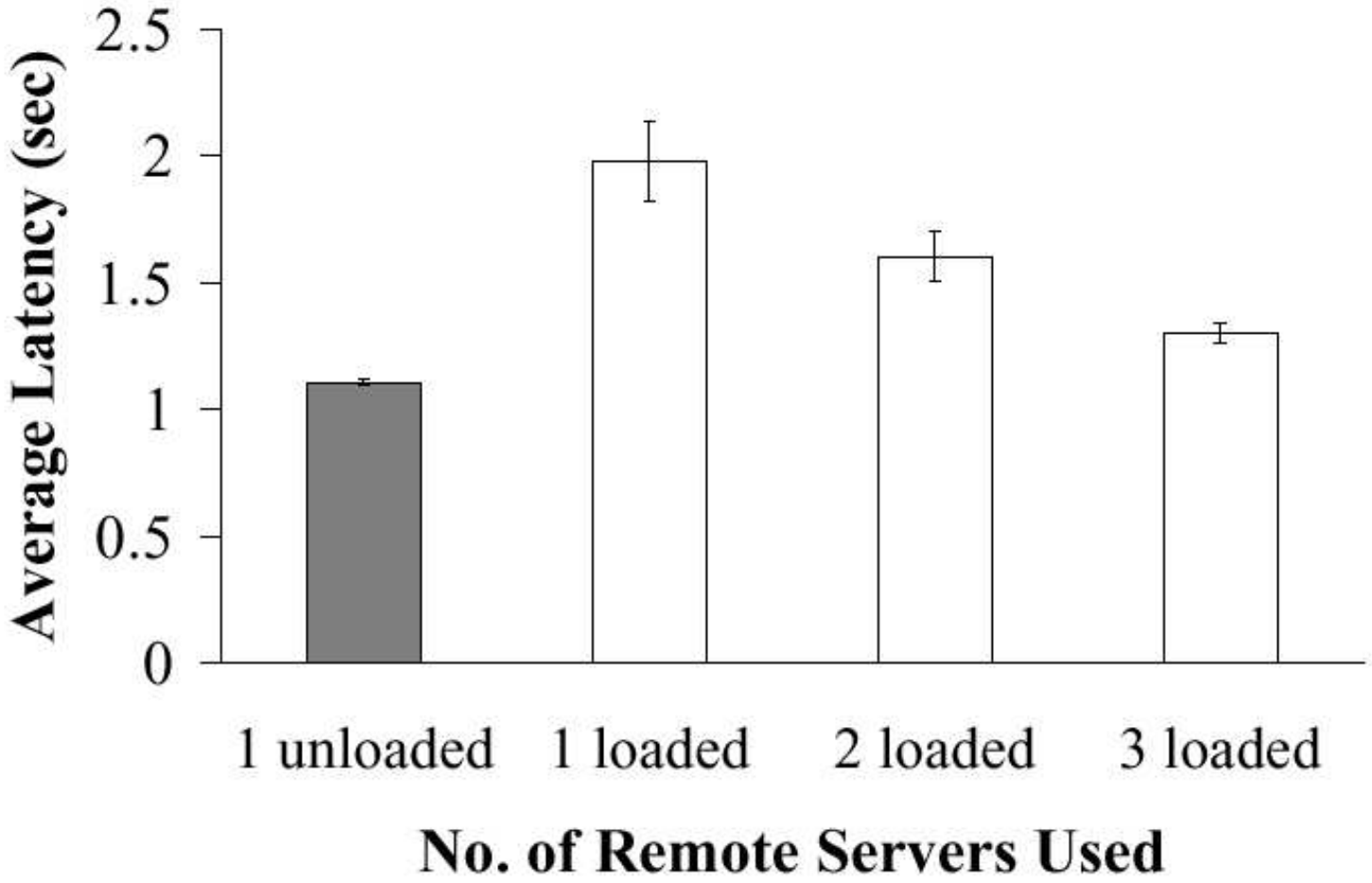
- Availability of compute servers varies wildly
  - Mobility ⇒ Cannot expect just one situation

Resource Poor

Resource Rich
(Smart Rooms etc.)

- Tactics allow us to automatically use extra resources in environment
  - Without modifying the application

Tactics: Using Extra Servers



Still able to get performance improvements with loaded servers!!

Roadmap

- Data Staging
- Remote Execution
- Service Discovery

One Ring to Bind Them All?

- Abundance of service discovery protocols
- Every site has its own favourite
  - Mobile applications cannot assume any particular mechanism
- We built a middleware service to
  - Leverage existing mechanisms
  - Isolate applications from changes in underlying mechanism

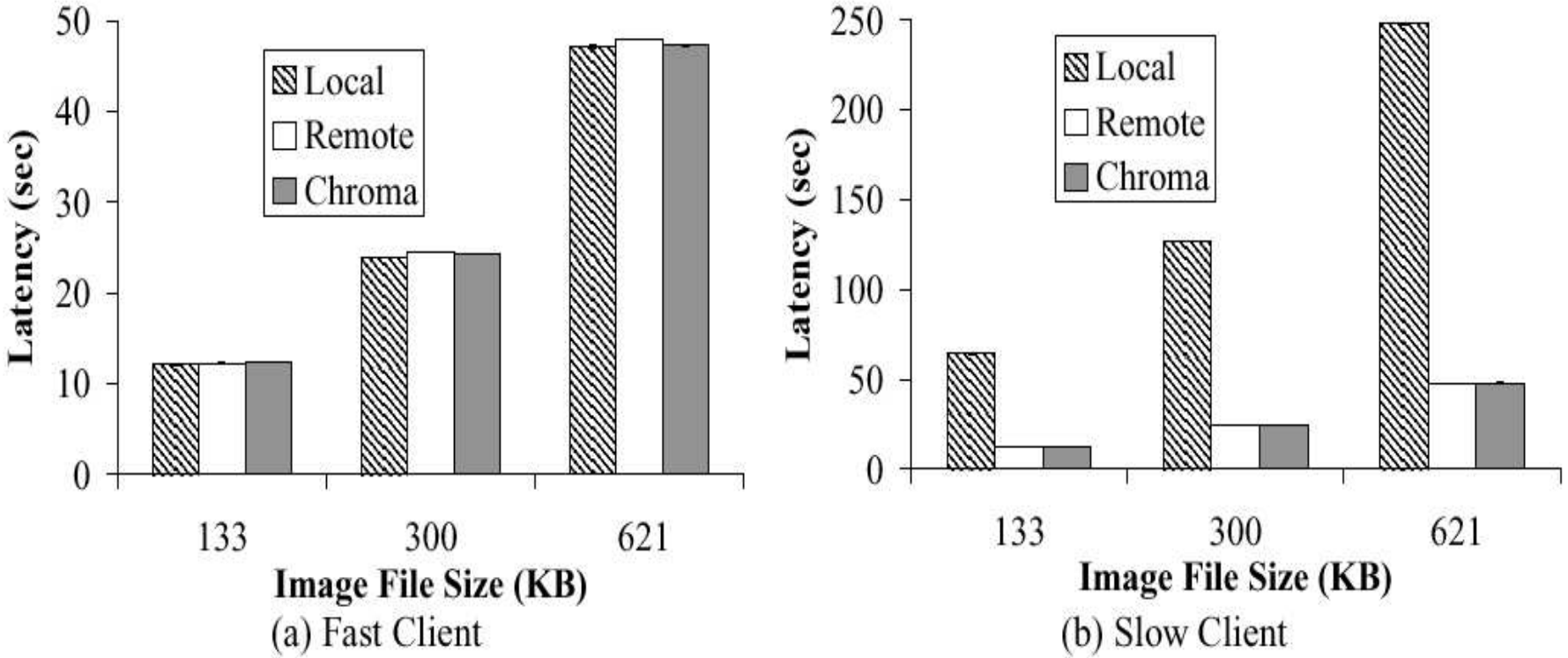
Current Work

- Data Staging:
  - Proxies for other file systems – e.g. NFS
  - Cache management policies
  - Prediction mechanisms
- Remote Execution:
  - Creation of resource allocation policies
  - Adapting software engineering methods to ease development times

Conclusions

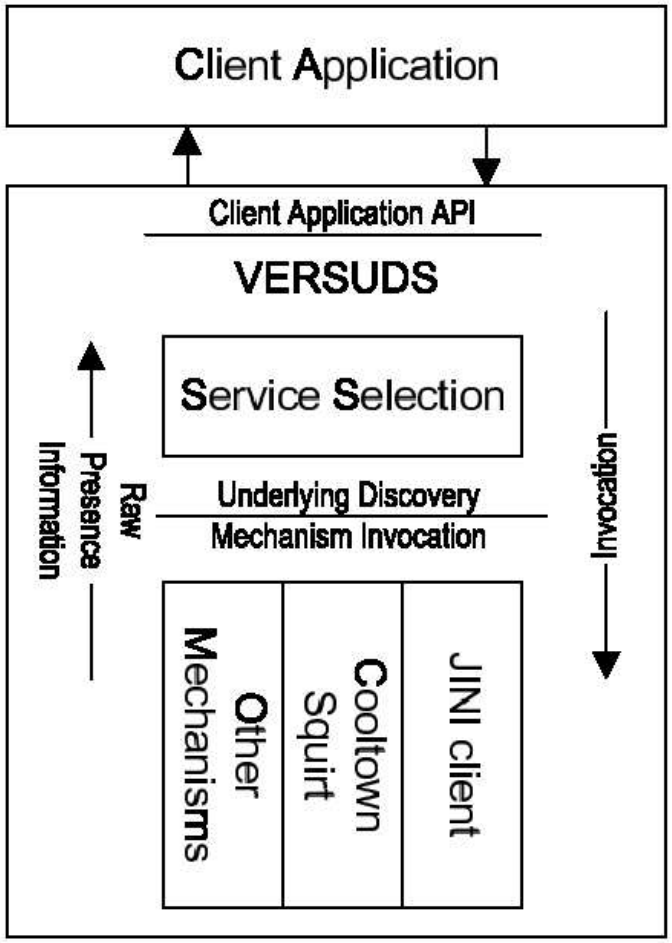
- Cyber Foraging is a vision of mobile computing
- Presented two methods of doing it
  - Data Staging
  - Remote Execution

Tactics Don’t Hurt



The latency that was achieved by executing face remotely and locally for all inputs on both clients is shown. We see that Chroma managed to achieve the lowest possible latency by choosing the right place to execute face in all cases.

Middleware Service (VERSUDS)



- Virtual layer on top of existing service discovery mechanisms
  - Supports common functionality
- Provides standard API to applications
- Provides isolation to mobile applications

Related Work

- Data Staging
  - Oceanstore
  - Peer to Peer Systems (FreeNet, Gnutella …)
  - CDNs (Akamai)
- Remote Execution
  - Odyssey, Spectra, Puppeteer, Abacus …
  - Declarative languages (4GLs, Little languages)
  - Corba, Java RMI