eXCloud:

Transparent Runtime Support for Scaling Mobile Applications in Cloud

Ricky K. K. Ma, King Tin Lam, Cho-Li Wang

13 Dec 2011

Systems Research Group
Department of Computer Science
The University of Hong Kong

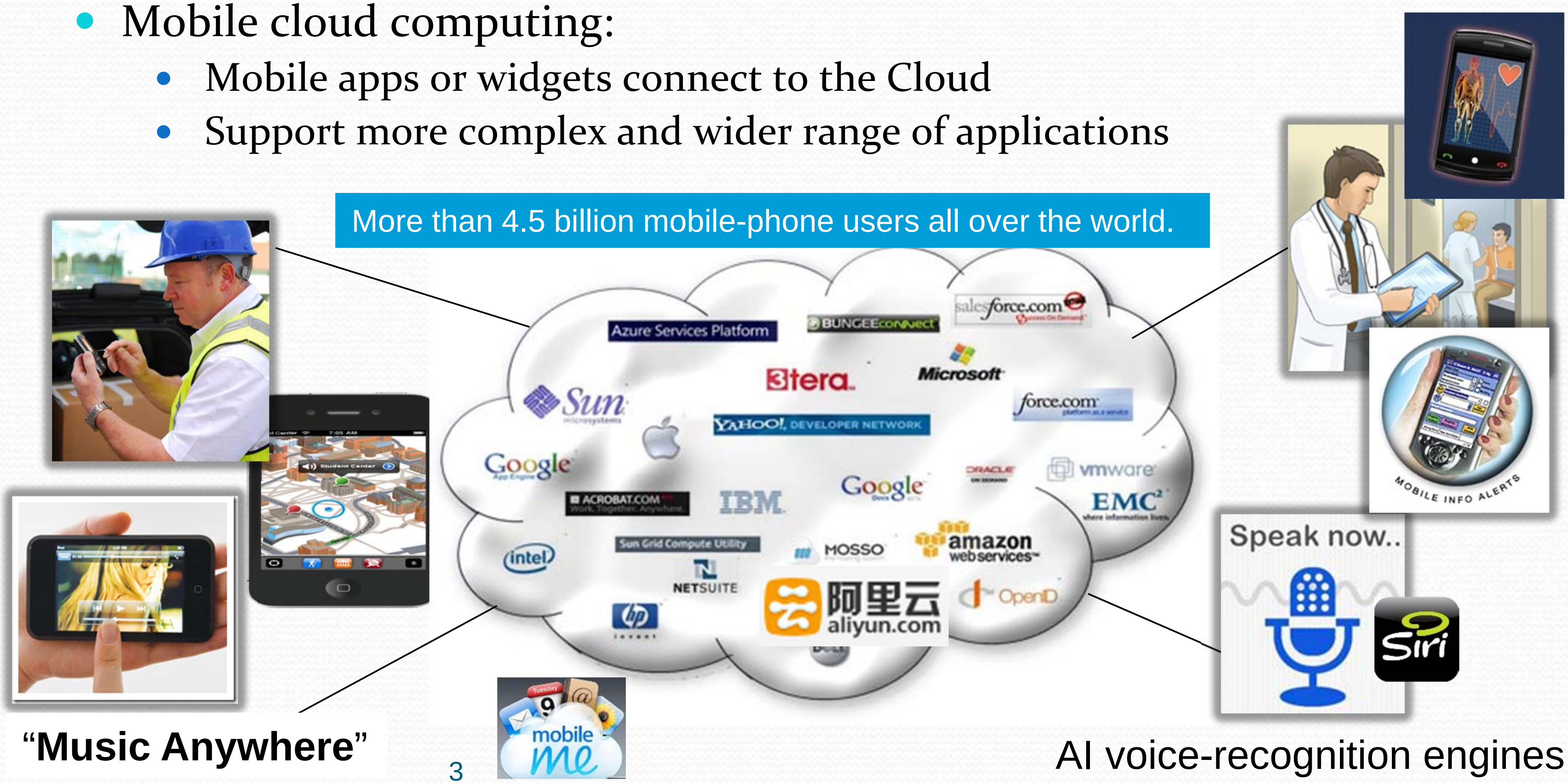
Outline

- Research background and motivation
- Execution model for resource utilization
- System design and implementation
- Performance evaluation

Background

- Cloud computing:
  - Computing power + data storage moved to the Web (data centers)
  - PC → thin clients
- Mobile cloud computing:
  - Mobile apps or widgets connect to the Cloud
  - Support more complex and wider range of applications

More than 4.5 billion mobile-phone users all over the world.



“Music Anywhere”

AI voice-recognition engines

Mobile cloud computing

- **Several benefits in shifting computing to the cloud**
  - More computing power, large memory and storage
  - Rich software libraries
- **Scaling (up) mobile applications**
  - To run mobile applications with more computing power
  - To allow mobile applications to use more resources
- **Privacy & Security Concerns**
  - March 2009, a bug in Google caused documents to be shared without the owners' knowledge.
  - July 2009, a breach in Twitter allowed a hacker to obtain confidential documents.
  - **Not all data/computing should be done in Cloud**

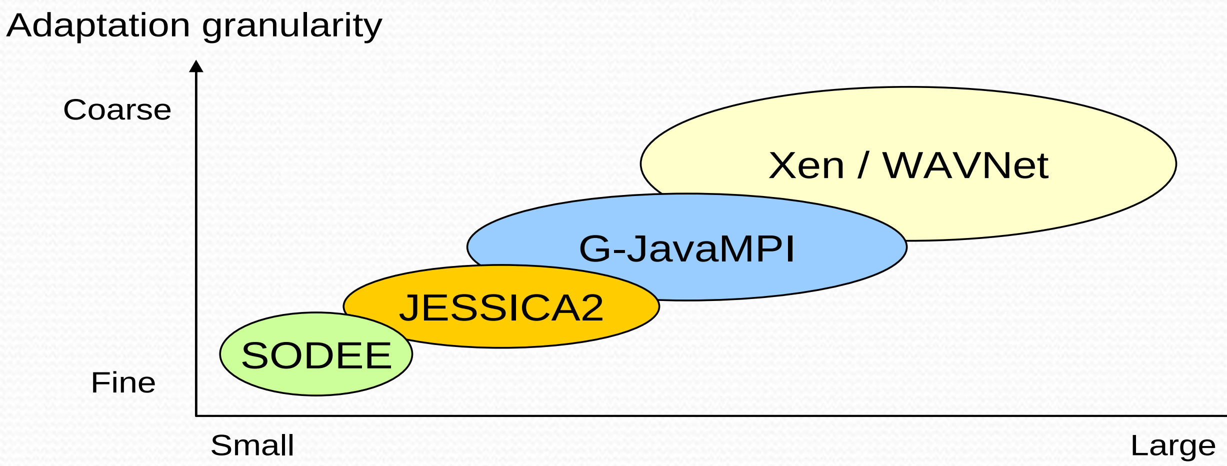
Cloud Interoperability

- **Cloud APIs still proprietary.!!**
  - **Data Lock-In:** Customers cannot easily extract their data and programs from one site to run on another
- **Standard Cloud APIs**
  - **More than 30 standards organizations are currently drafting cloud computing standards**
    - **Difficult (5000 APIs in 2011)**
    - Hinder the development of clouds + Easy to attack
  - Libraries that talks to various clouds (Google, Amazon, ..)
    - Deltacloud (Red Hat), Libcloud (Rackspace, phyton-based), jclouds (Java-based), Simple Cloud API (IBM, Zend, Microsoft)
- **No APIs** => eXCloud
  - **Total transparency: migration-transparent, location-independent, "cloud-transparent"**

HKU eXCloud Project:

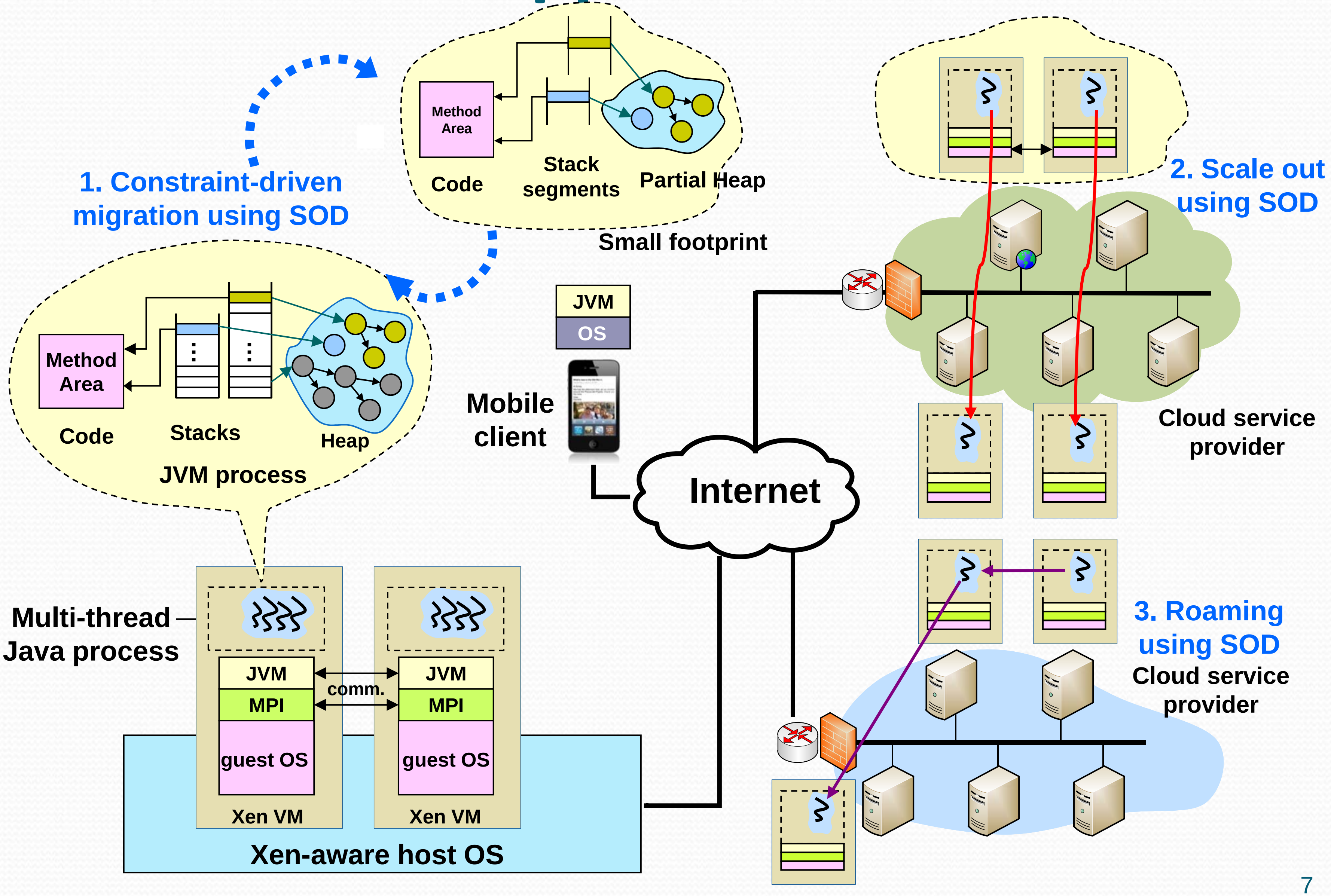
Multi-level Mobility Support

Granularity	Migration Technique (System)	Target System Type (Area)
Frame level	Stack-on-demand (SODEE)	Cloud, cloudlet, mobile network (WAN/LAN)
Thread level	Thread migration (JESSICA2)	Cluster (LAN)
Process level	Process migration (G-JavaMPI)	Grid (WAN/LAN)
VM level	Live VM migration (Xen)	Cluster (LAN)
	Wide-area live VM migration (WAVNet)	Cloud, p2p/desktop cloud (WAN)



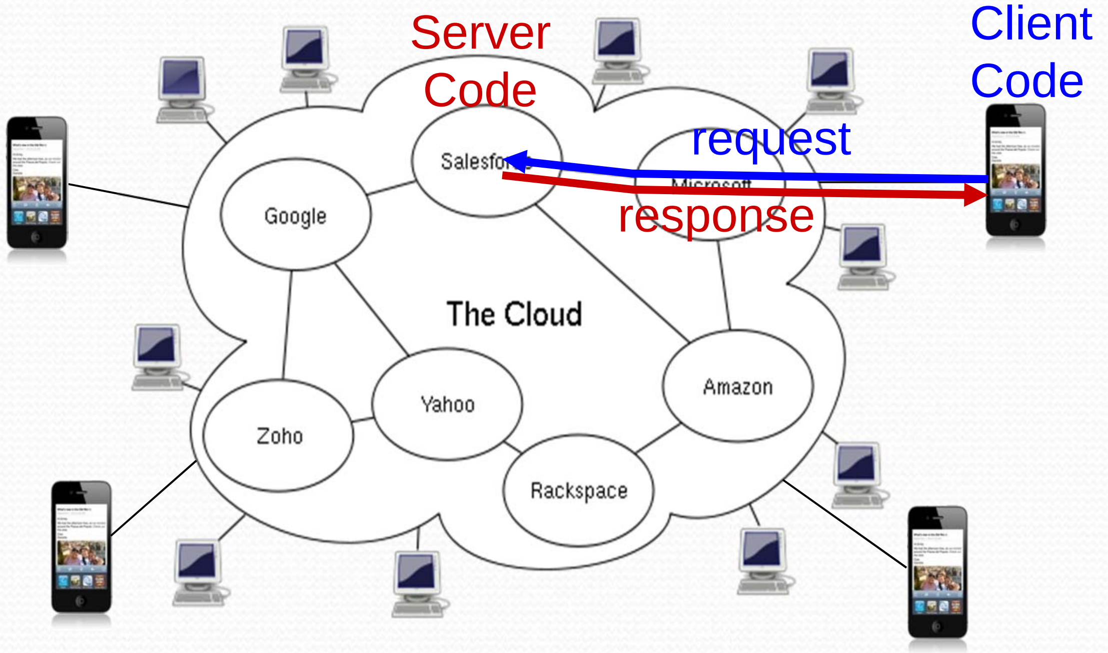
Allow multi-level task migration ranging from VM instance, process, thread, to stack frame.

eXCloud: Application Scenarios



Current client-server model for mobile cloud computing

- Requests sent to web servers (cloud service providers)
- Application executes completely within the Cloud
- Results sent back to mobile clients

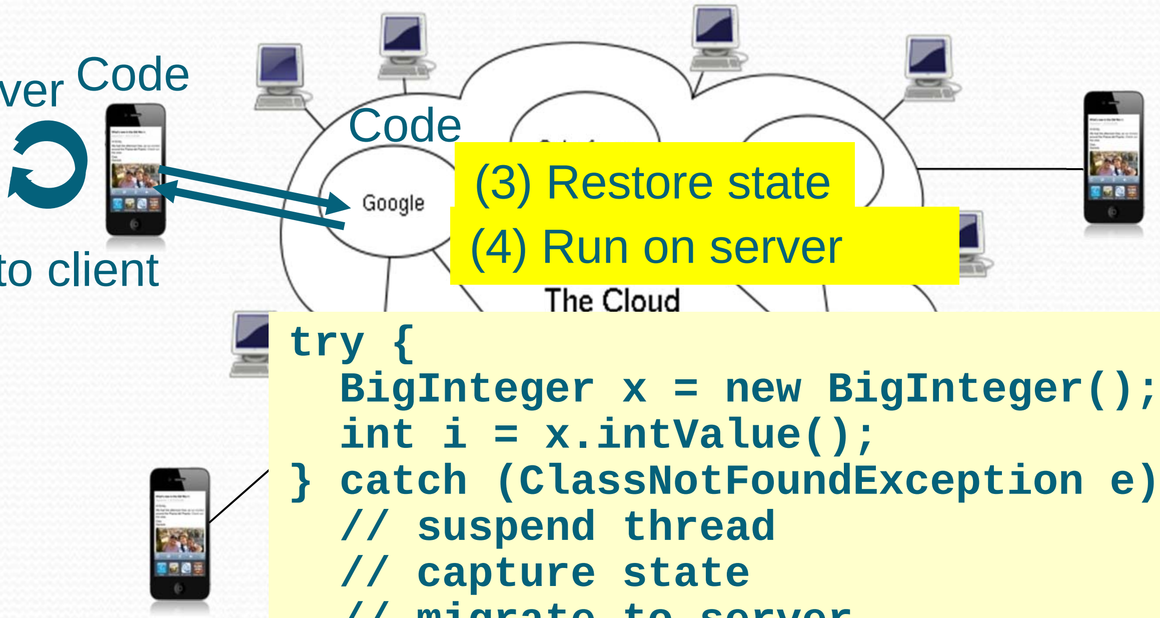


eXCloud: elastic execution model

- **Lightweight task migration :** only “**needed code**” (not the whole program) + “**state**” are migrated (Mobility is bidirectional)
- **On-demand mobility** : migration is triggered only when missing library class on the device JVM (J2ME), or insufficient memory.
  - Other migration policies, e.g., driven by resource constraints (CPU power, network, battery power), data locality, cost saving ..

(1) Suspend and capture state

(2) Migrate **code & state** to server



(3) Restore state

(4) Run on server

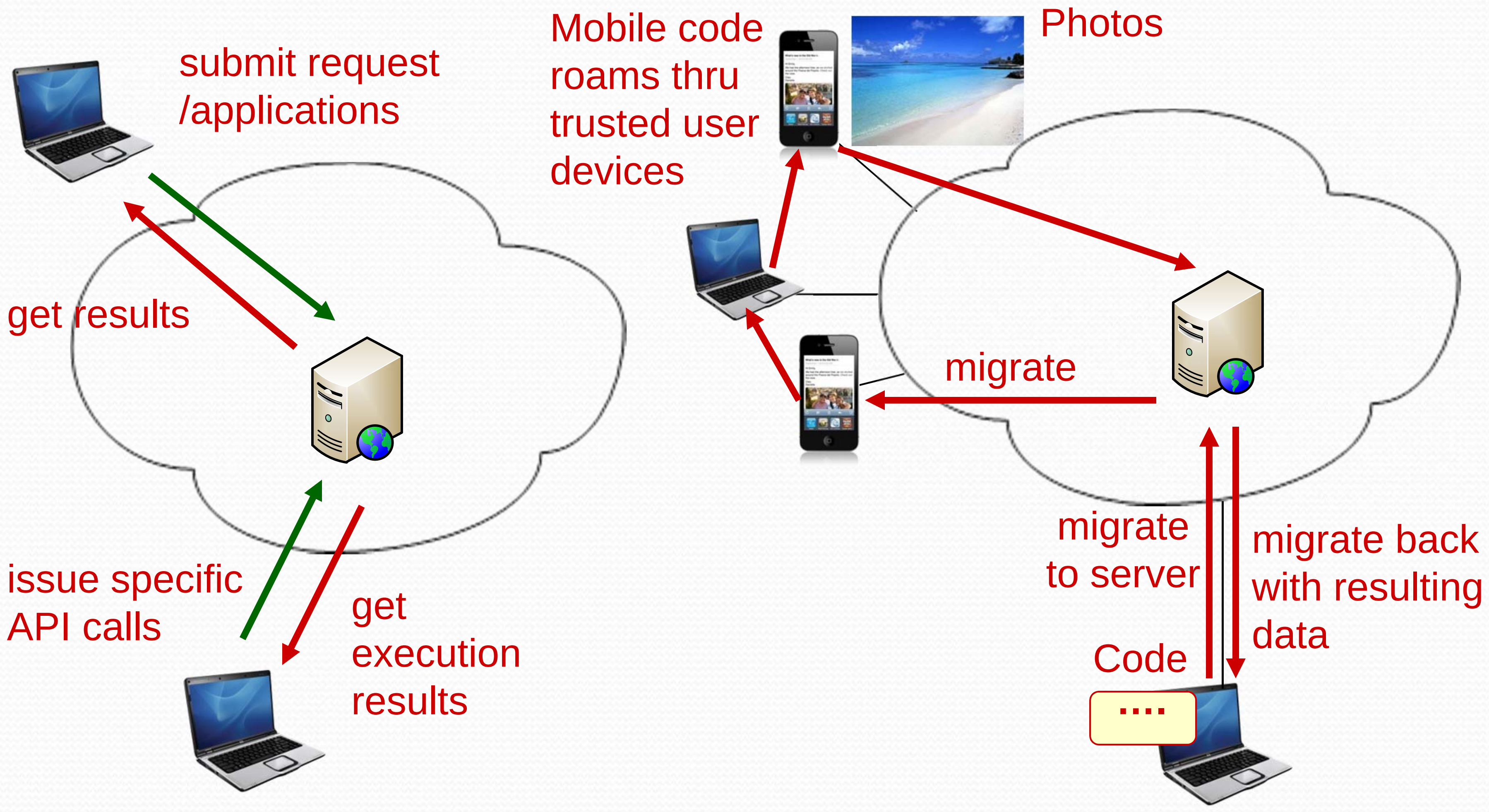
(5) Migrate **state** back to client

(6) Restore state

(8) run locally for finishing remaining execution.

```
try {
  BigInteger x = new BigInteger();
  int i = x.intValue();
} catch (ClassNotFoundException e){
  // suspend thread
  // capture state
  // migrate to server
}
```

Seamless integration of mobile nodes



submit request /applications

get results

issue specific API calls

get execution results

Mobile code roams thru trusted user devices

Photos

migrate

migrate to server

migrate back with resulting data

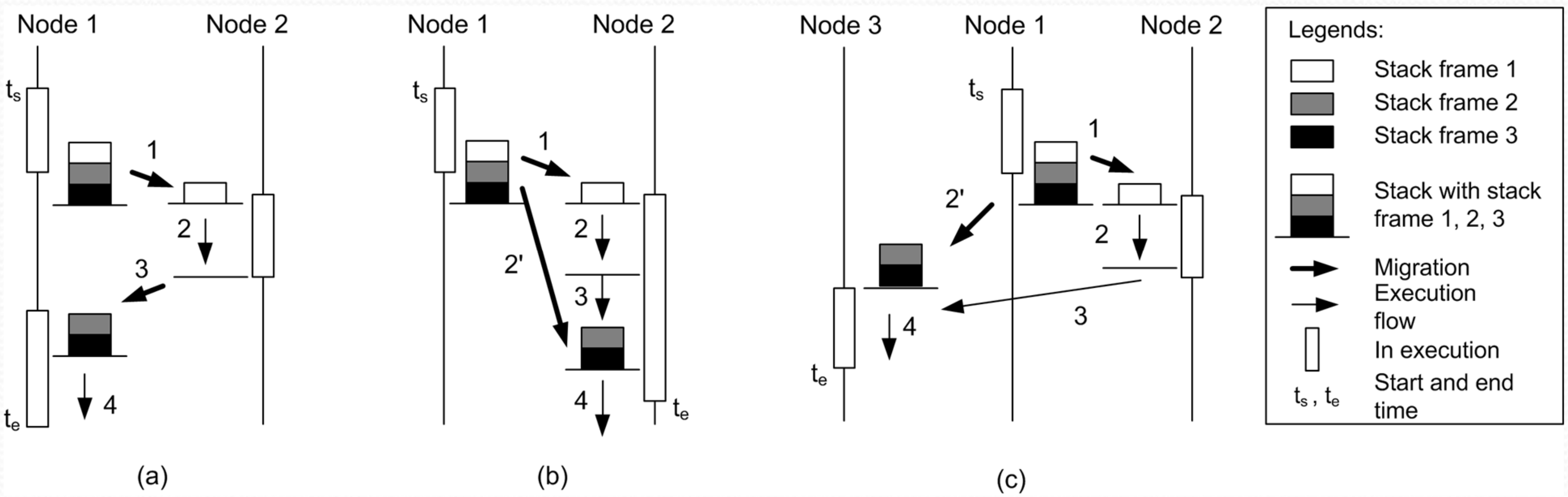
Code

Traditional client-server model

our elastic mobility model

Benefits of our new execution model

- **Integrated seamlessly with the mobile clients**
  - Allow better operability of cloud
  - **No need to write separate client and server codes**
- **Elastic use of users' devices**
  - More powerful mobile applications can be built

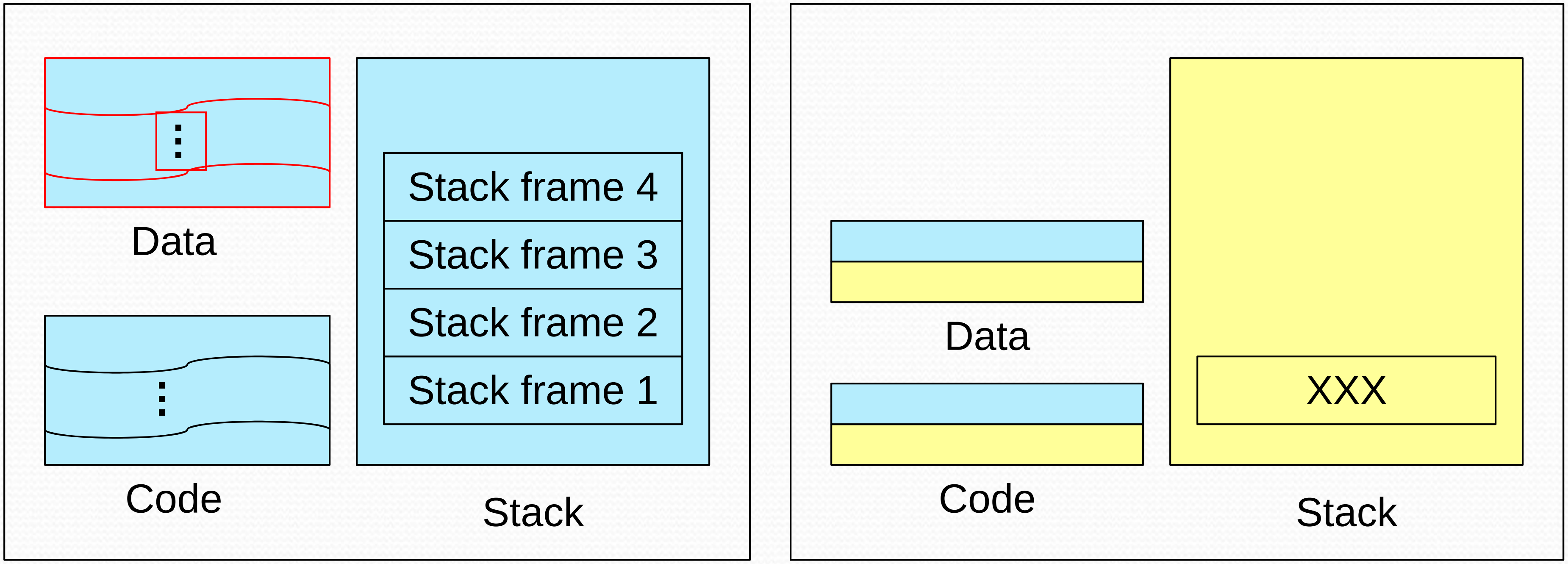


(a) "Remote Method Call" (b) thread migration (c) "Task Roaming"

eXCloud

- Middleware system for mobile cloud computing
  - **No modification of underlying system**
- **Allow multi-level task migration**
  - ranging from VM instance to stack frame
- Seamlessly integration mobile devices and cloud nodes
  - allow utilization of resources in different nodes.
    - So as to achieve scaling (up) of mobile applications
- **Stack-on-demand** approach (focus of this paper) is used to support the mobility

- **Stack-On-Demand (SOD)**
  - Allow lightweight task migration



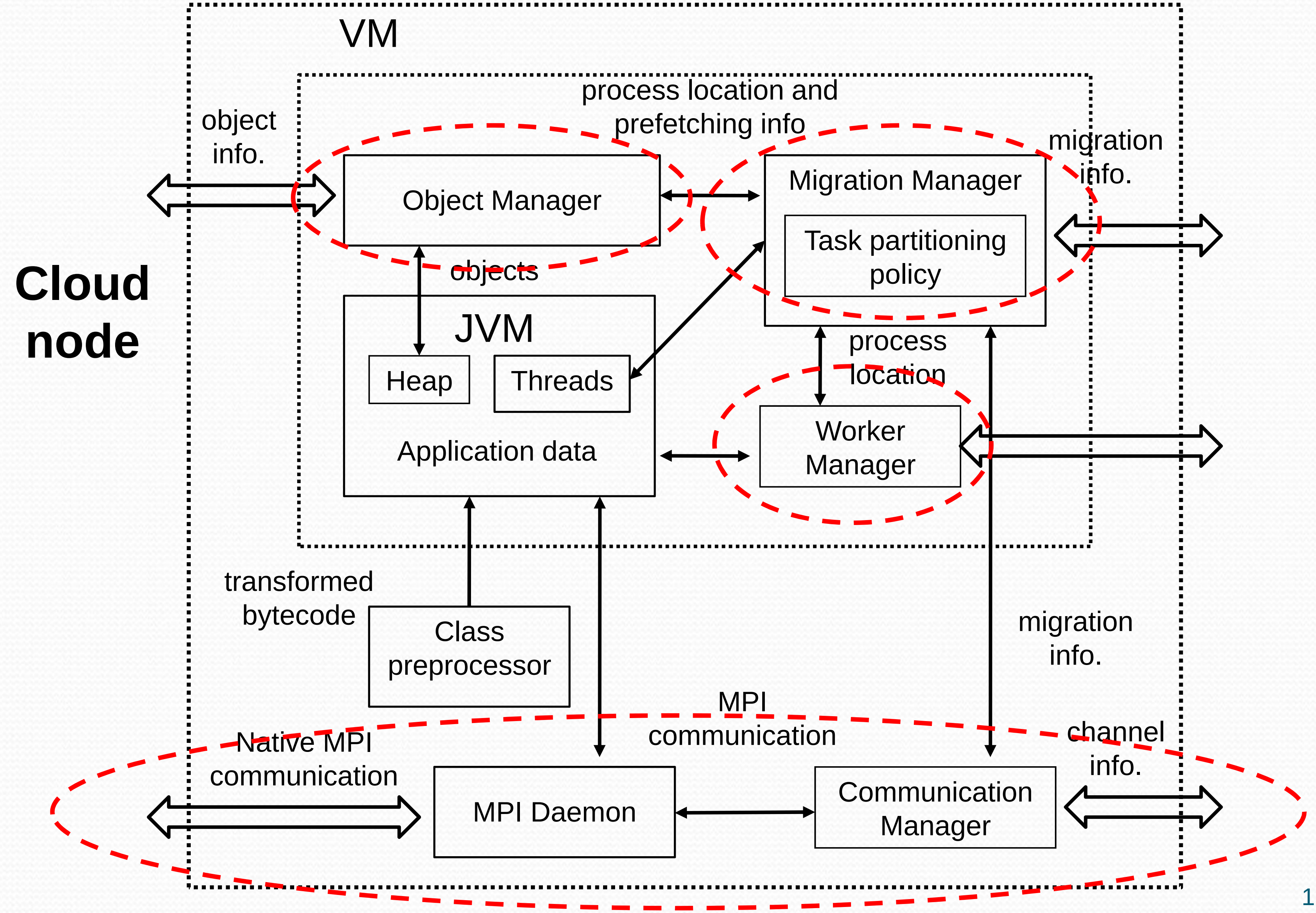
Migrating task on
Source Node

Worker process on
Destination Node

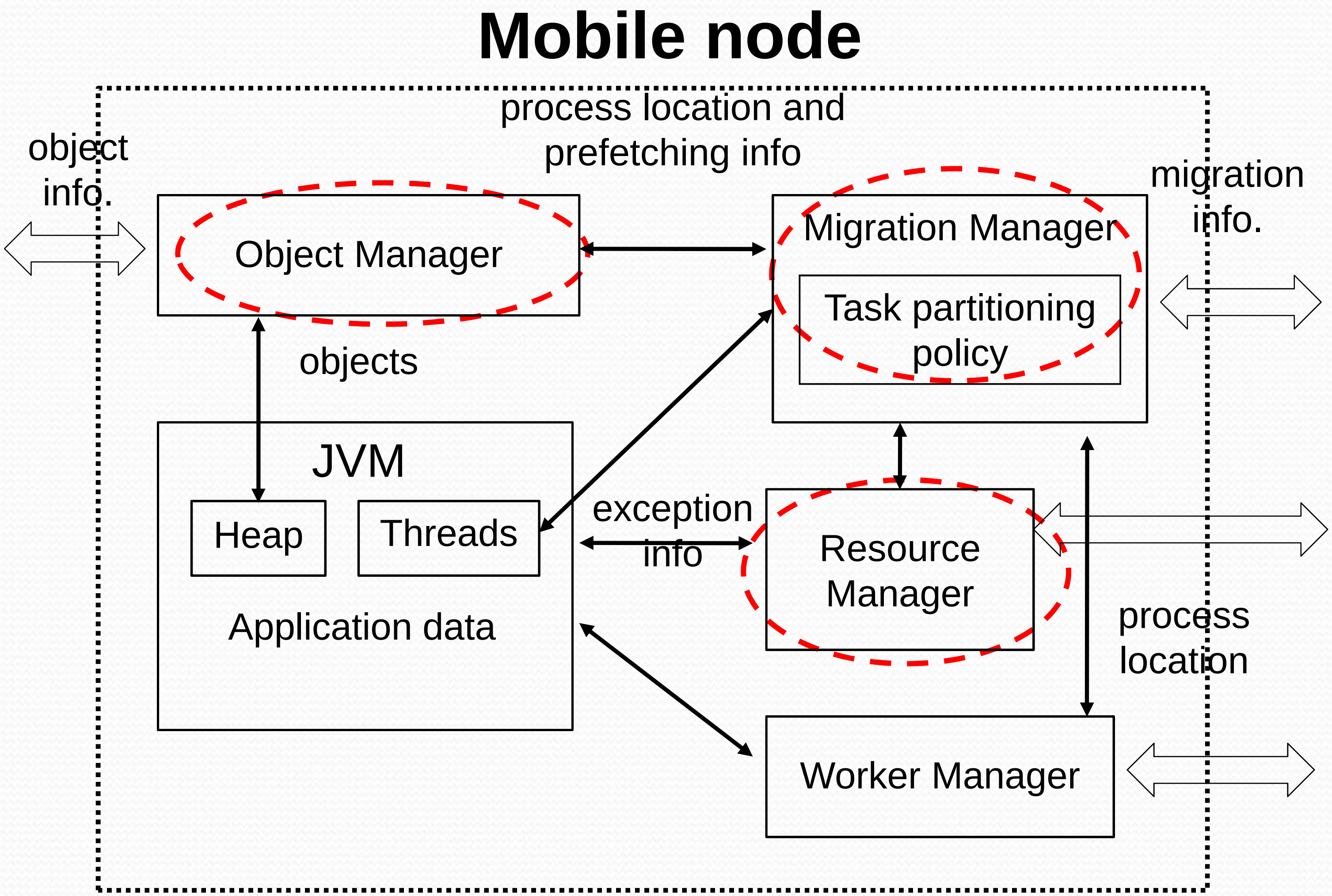
System Design

- **Design goal**
  - **Low overhead**
    - Allow lightweight task migration. Induce low overhead, especially during normal execution when there is no migration
  - **Transparency**
    - No need for users to modify their programs and libraries
  - **Portability**
    - No need to use a specific JVM.
  - **Adaptation to new environment**
    - allow to use resources in new location to utilize resources
- **Our approach**
  - Bytecode instrumentation is taken by Class Preprocessor in the preprocessing step, which is taken offline

System Architecture (SOD)



System architecture (SOD)



Performance Evaluation

- **Platform A**
  - **Cluster nodes**
    - Each node: 2 x Intel 6-Core Xeon 2.66 GHz, 48GB DDR3 RAM
    - OS: RedHat Enterprise Linux AS 4.6 (32 bit) with Xen 3.0.3
    - JVM: Sun JDK 1.6 (64 bit), nodes interconnected by Gigabit Ethernet
- **Platform B**
  - **Cluster nodes**
    - Each node: 2 x Intel 4-Core Xeon 2.53 GHz, 32GB DDR3 RAM
    - OS: Fedora 11 x86_64 with Xen
    - JVM: Sun JDK 1.6 (64 bit), nodes interconnected by Gigabit Ethernet
  - **Mobile nodes**
    - **iPhone 4 handset**: 800MHz ARM CPU, 512 MB RAM
    - JVM: JamVM 1.5.1b2-3; Java class library: GNU Classpath 0.96.1-3
    - Connected to Cluster through Wi-Fi (bandwidth controlled by a router)

Performance Evaluation

- Focus on performance of task migration of SOD

Evaluations	Description	Platform used	Nodes involved
A	Overhead analysis	A	cloud nodes
B	Scaling out for parallel programs	A	cloud nodes
C	Migration from mobile device to cloud node	B	cloud nodes + mobile nodes
D	Migration from cloud node to mobile device	B	cloud nodes + mobile nodes

Evaluation A: Overhead Analysis

- **Testing programs**

App	Description	Max. stack height	Total field size (byte)
Fib	Calculate 46th Fib. No.	46	< 10
NQ	Solve N-Queens problem with board size 14	16	< 10
FFT	Calculate 256-point 2D FFT	4	> 64M
TSP	Solve travelling Salesman Problems with 12 cities	4	~ 2500

- Testing Migration Technique
  - Stack-On-Demand Migration (SOD) in the execution engine (SODEE)
  - Java Process Migration (G-JavaMPI)
    - Use Sun JDK and JVMTI , and perform eager-copy migration
  - Java Thread Migration (JESSICA2)
    - Use Kaffe VM
- Migrations taken in 2 nodes

Execution time of different systems

	Execution Time (sec)					
	SODEE on Xen		*JESSICA2 on Xen*		*G-JavaMPI on Xen*	
App	*w / mig*	*w/o mig*	*w / mig*	*w/o mig*	*w / mig*	*w/o mig*
Fib	**12.78**	12.70	47.31	47.25	16.45	12.68
NQ	**7.72**	7.67	37.49	37.30	7.94	7.64
FFT	**3.60**	3.56	16.54	19.45	3.67	3.59
TSP	**10.8**	10.6	253.6	250.2	15.13	10.75

- *Execution time with SOD migration is the shortest*

- **Migration overhead** = execution time w/ migration – execution time w/o migration

	Migration Overhead (ms)		
	SODEE on Xen	*JESSICA2 on Xen*	*G-JavaMPI on Xen*
Fib	83	**60**	3770
NQ	**49**	193	299
FFT	**13**	96	84
TSP	**194**	3436	4381

•*SOD has the smallest migration overhead among most of the applications*

•*Migration overhead of SOD ranges* ***from 13ms to 194ms***

•*It can be* ***1/45 – 1/6*** *of G-JavaMPI*

Migration latency in different systems

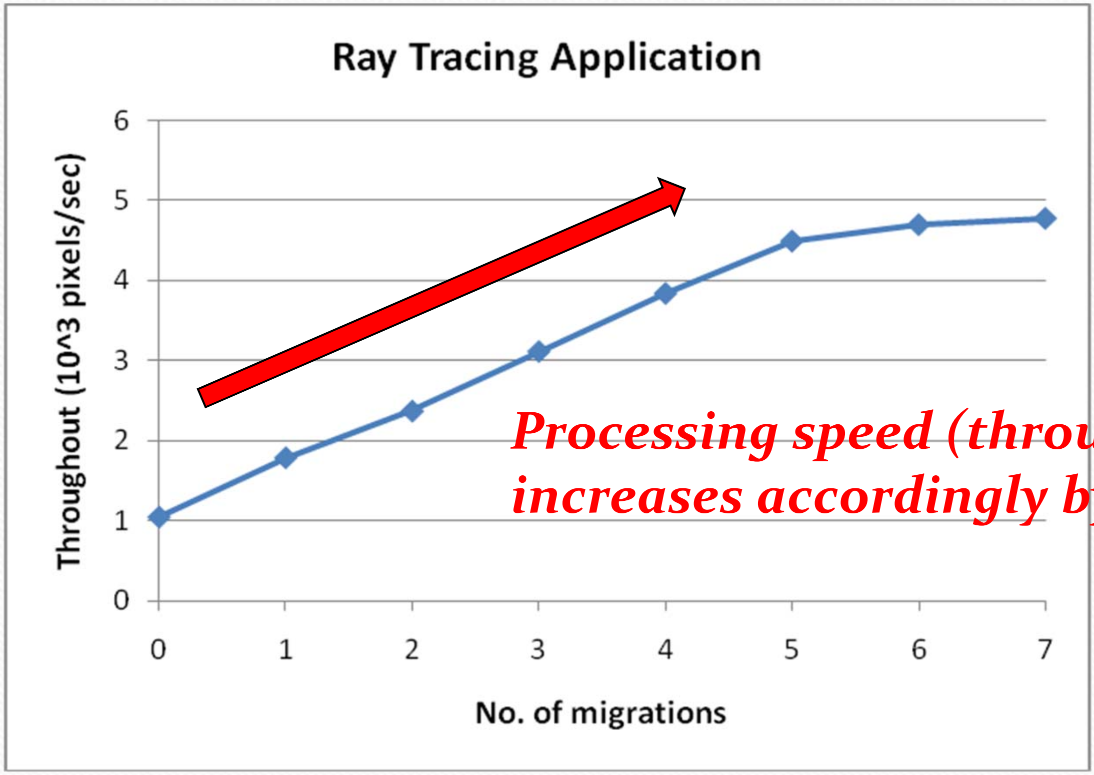
- *Migration latency of SOD migration is the smallest among the applications.*

App	SODEE on Xen			G-JavaMPI on Xen			JESSICA2 on Xen		
	Mig. latency (ms)			*Mig. latency (ms)*			*Mig. latency (ms)*		
	Capture	*Transfer*	*Restore*	*Capture*	*Transfer*	*Restore*	*Capture*	*Transfer*	*Restore*
Fib	6.31			894.73			**12.75**		
	0.25	2.71	3.4	42.5	2.44	45	0.2	10.3	2.26
NQ	6.8			69.25			**8.06**		
	0.32	2.89	3.6	35.5	2.81	31	0.11	1.73	6.23
FFT	**19.39**			3659.6			59.08		
	0.35	14.9	4.1	742	2440	477	0.08	2.4	56.6
TSP	8.08			78.84			**19.4**		
	0.3	2.8	5	32	4.46	42	0.05	10.6	8.74

- **Transfer time** = time needed for the state data, upon being ready for transfer, to reach the destination

Evaluation B: Scaling out by SOD Migration

- *Application: parallel Java ray-tracing program using MPI*
- *Starts with all processes executed in a single node.*
- *Scale-out by migrating rendering worker processes to idle nodes.*



Processing speed (throughput) increases accordingly by scaling out.

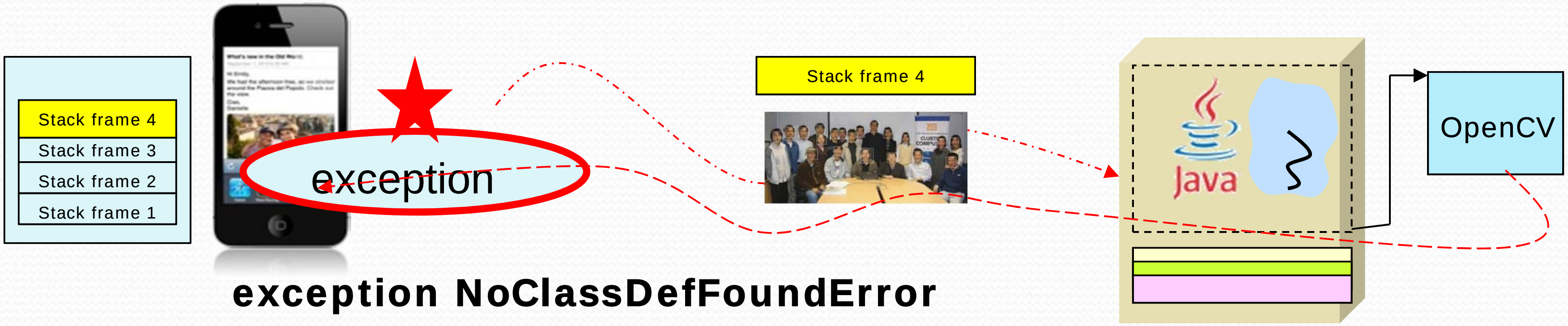
Evaluation C: Migration from mobile device to cloud node

- Migrate computation-intensive tasks from mobile devices to cluster nodes.
- *The performance gain through migration are* ***3 to 56 times.***
- *Total migration latency is larger due to the lower processing power of mobile nodes and WiFi connection*

	exec. time w/o mig. (s)	exec. time w/ mig. (s)	gain	capture time (ms)	transfer time (ms)	restore time (ms)	total migration latency (ms)
Fib	56.79	0.99	**x56**	140.33	94.33	11.67	**246.33**
NQ	32.67	1.04	**x30**	183.26	86.31	10.52	**280.09**
FFT	6.06	1.26	**x3.8**	156.48	232.46	14.58	**403.52**

SOD : Face Detection on Cloud

apps	capture time (ms)	transfer time (ms)	restore time (ms)	total migration latency (ms)
DBRetrieve	85	76	6	167
FaceDetect	103	155	7	265

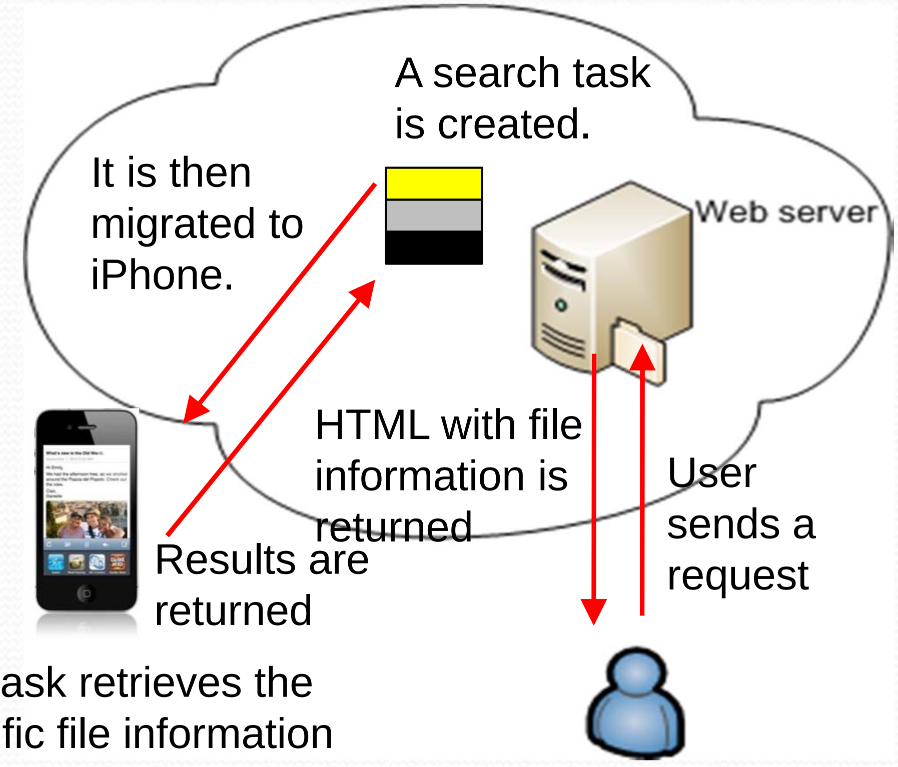


exception NoClassDefFoundError

Evaluation D: Migration from cloud node to mobile device

- Memory footprint (in server): 31,907,096 bytes (~30MB)
- Memory footprint (in iPhone): 852,544 bytes (~833KB)
- SOD avoids memory consumption (**up to 97%**)
- As there are active network connections between the server program and clients, the need of migrating native states are avoided

- Current settings
  - 5 directories with images
  - empty directory name “ip4”



A search task is created.

It is then migrated to iPhone.

HTML with file information is returned

User sends a request

Results are returned

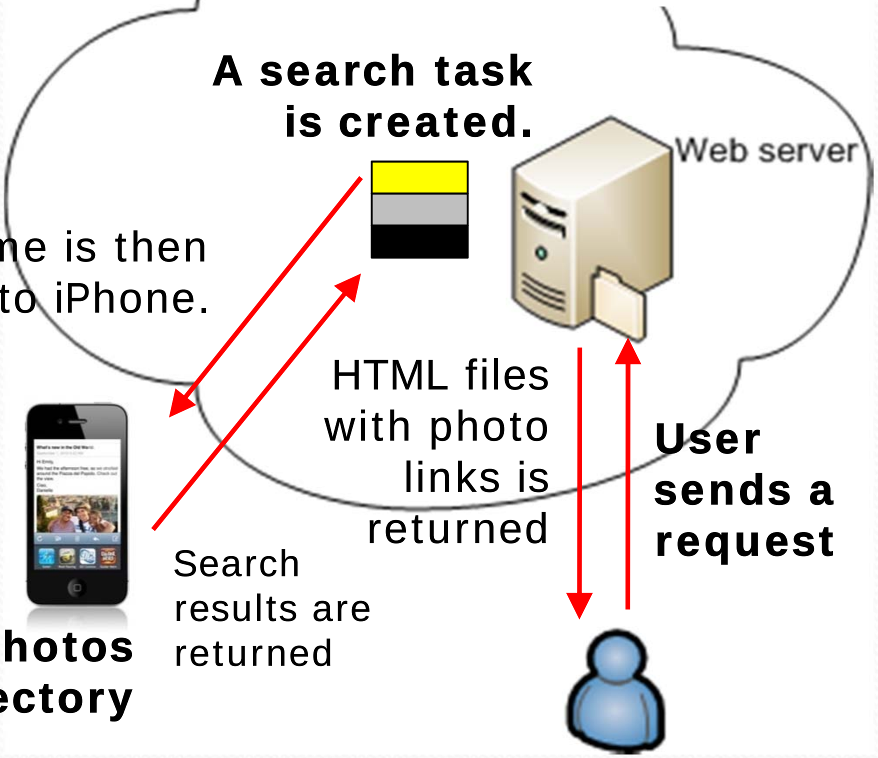
The task retrieves the specific file information

SOD: “Mobile Spider” on iPhone

Bandwidth (kbps)	Capture time (ms)	Transfer time (ms)	Restore time (ms)	Migration time (ms)
50	14	1674	40	1729
128	13	1194	50	1040
384	14	728	29	772
764	14	672	31	717

Size of class file and state data = 8255 bytes

Migration from cloud node to mobile devices



A photo sharing Cloud service

A search task is created.

Stack frame is then migrated to iPhone.

HTML files with photo links is returned

User sends a request

Search results are returned

(with Wi-Fi connection)

The task searches for photos available in the specific directory

Conclusion and Future Work

- A middleware system eXCloud is introduced
  - To provide seamless, multi-level task mobility support at different granularity
- Stack-On-Demand execution model is used
  - To allow lightweight partial state migration to allow migration among cloud nodes and mobile nodes.
- Experiments show that
  - SOD induces less overhead than other migration system for most of the benchmarks
  - Significant performance gains in mobile devices are archived by utilizing cloud resources.
- Various policies can be further explored
  - Migration, prefetching, task distribution

Thank you!

Q & A