Multi-language Applications and Systems

Chandra Krintz
Laboratory for Research on
Adaptive Compilation Environments (RACE)
Computer Science Dept.
Univ. of California, Santa Barbara

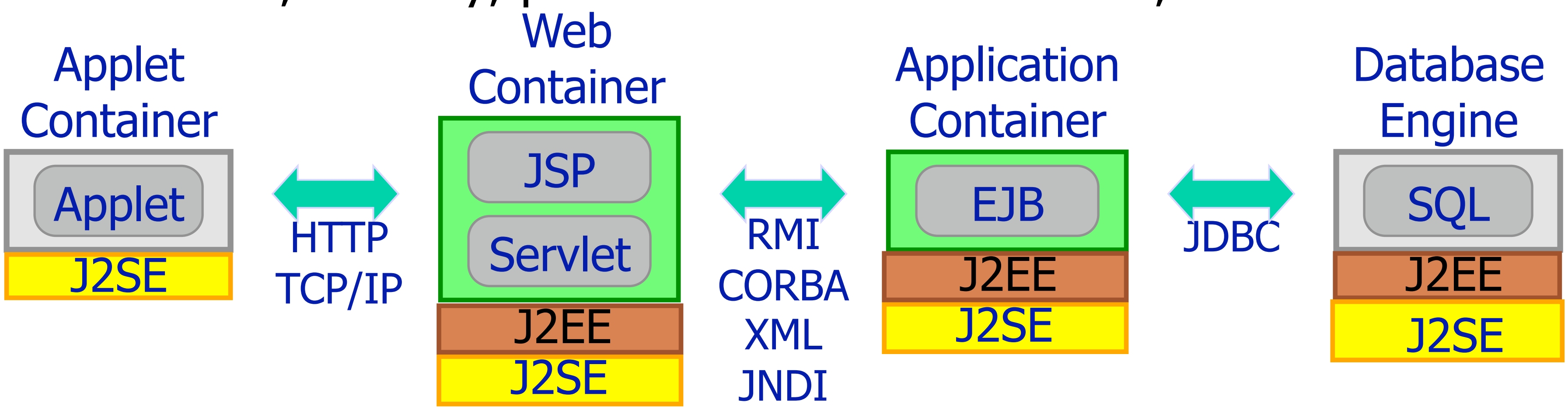
VEESC
September 3, 2010

Modern Software and Systems

- Hardware/architecture evolution
  - Low cost, high performance, memory-rich, multicore, virtualization support
- Distributed cluster computing
  - Web services, parallel/concurrent tasks, cloud computing
- Software as components, modules, tiers
  - Executed within own runtime (execution engine)
  - Reuse, mobility, process-level fault tolerance, isolation

Modern Software and Systems

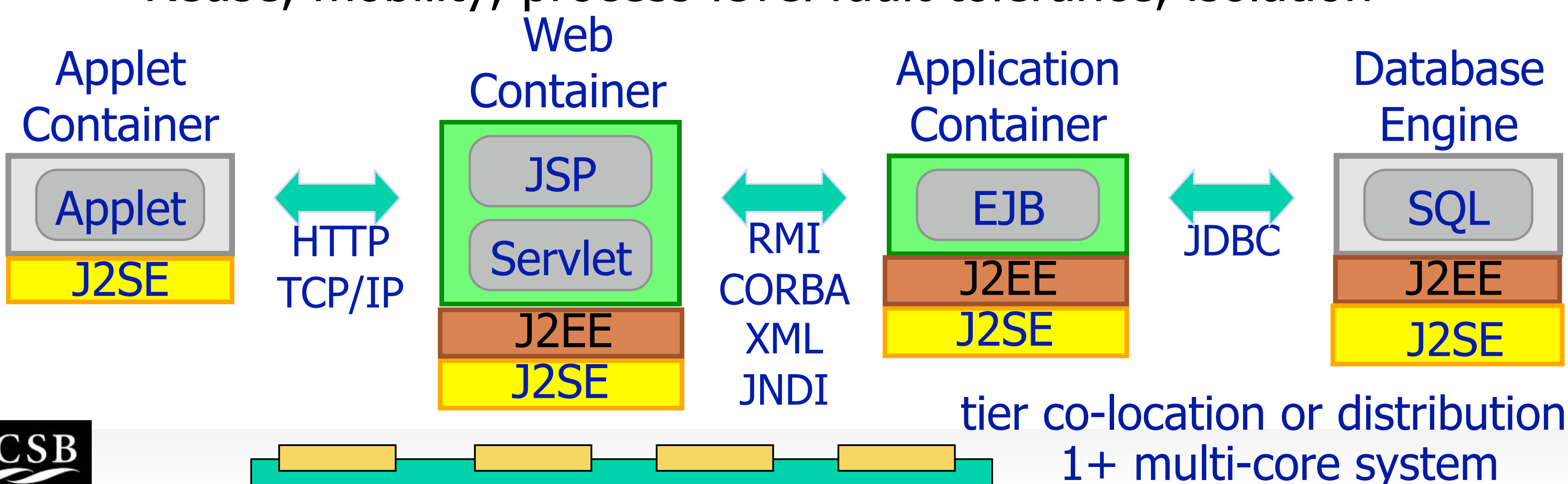
- Hardware/architecture evolution
  - Low cost, high performance, memory-rich, multicore, virtualization support
- Distributed cluster computing
  - Web services, parallel/concurrent tasks, cloud computing
- Software as components, modules, tiers
  - Executed within own runtime (execution engine)
  - Reuse, mobility, process-level fault tolerance, isolation



Traditional Java Enterprise / Web 1.0

Modern Software and Systems

- Hardware/architecture evolution
  - Low cost, high performance, memory-rich, multicore, virtualization support
- Distributed cluster computing
  - Web services, parallel/concurrent tasks, cloud computing
- Software as components, modules, tiers
  - Executed within own runtime (execution engine)
  - Reuse, mobility, process-level fault tolerance, isolation



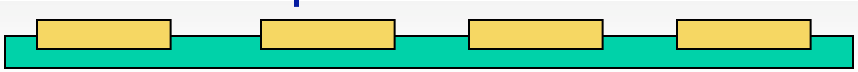
UCSB

Modern Software and Systems

- Hardware/architecture evolution
- Distributed cluster computing

- Software as components, modules, tiers
  - Executed within own runtime (execution engine)
  - Reuse, mobility, process-level fault tolerance, isolation
  - **Web 2.0, web services, cloud systems**
    - Presentation layer: Javascript, Ruby, Java, Python
    - Server-side logic: PHP, Perl, Java, Python, Ruby
    - Computations: MapReduce streaming (multi-language)
    - Database, key-value store: C++, Java, + query languages
  - Others (HPC): Python, Ruby, R with C, C++
  - Frameworks, IDES facilitate development and deployment

component co-location or distribution



1+ multi-core system

Why One Language is Not Enough

- Programmer preference, expertise
- Amenability to addressing the particular problem that the component is designed to solve
- Library and framework support
- Speed of development
  - Fast prototyping, software understanding
  - Easy and transparent dynamic updates
  - Implementation, testing, debugging
  - SWE practice (agility, pairs)
- Performance
- Portability
  - Availability of language runtimes (interpreters)

Choosing one means accepting limitations for 1+ metrics

Why One Language is Not Enough

- No one actually writes much code anymore…
  - Large numbers of programmers make their code available via the web (freely available and licensed open source)
    - Written in the language chosen by the author(s)
- Open source has experienced a surge in popularity, support, and participation
  - Participation by vast numbers of developers and users
    - Ideas for features, feedback, bug fixes
    - Short feedback/release loop
    - Online resources (FAQs, forums) save provide searchable support
    - Potential for viral, wide-spread use, free advertising
- Free software (open APIs)
  - Mashups
- Available packages

UCSB

Cross-language Interoperability

- Python, Javascript, Perl, PHP, Ruby, Java, C/C++, .Net, …
  - Mixed-environment debugging
- Cross-language/process communications technology
  - RPC, messaging
    - Thrift, HTTP/s, REST, SOAP, RPC, COM, SIP, SWIG, CORBA
    - For more than just web services: Map-Reduce (MR), MR-streaming, MPI
  - Data exchange formats
    - Protocol Buffers, XML, JSON

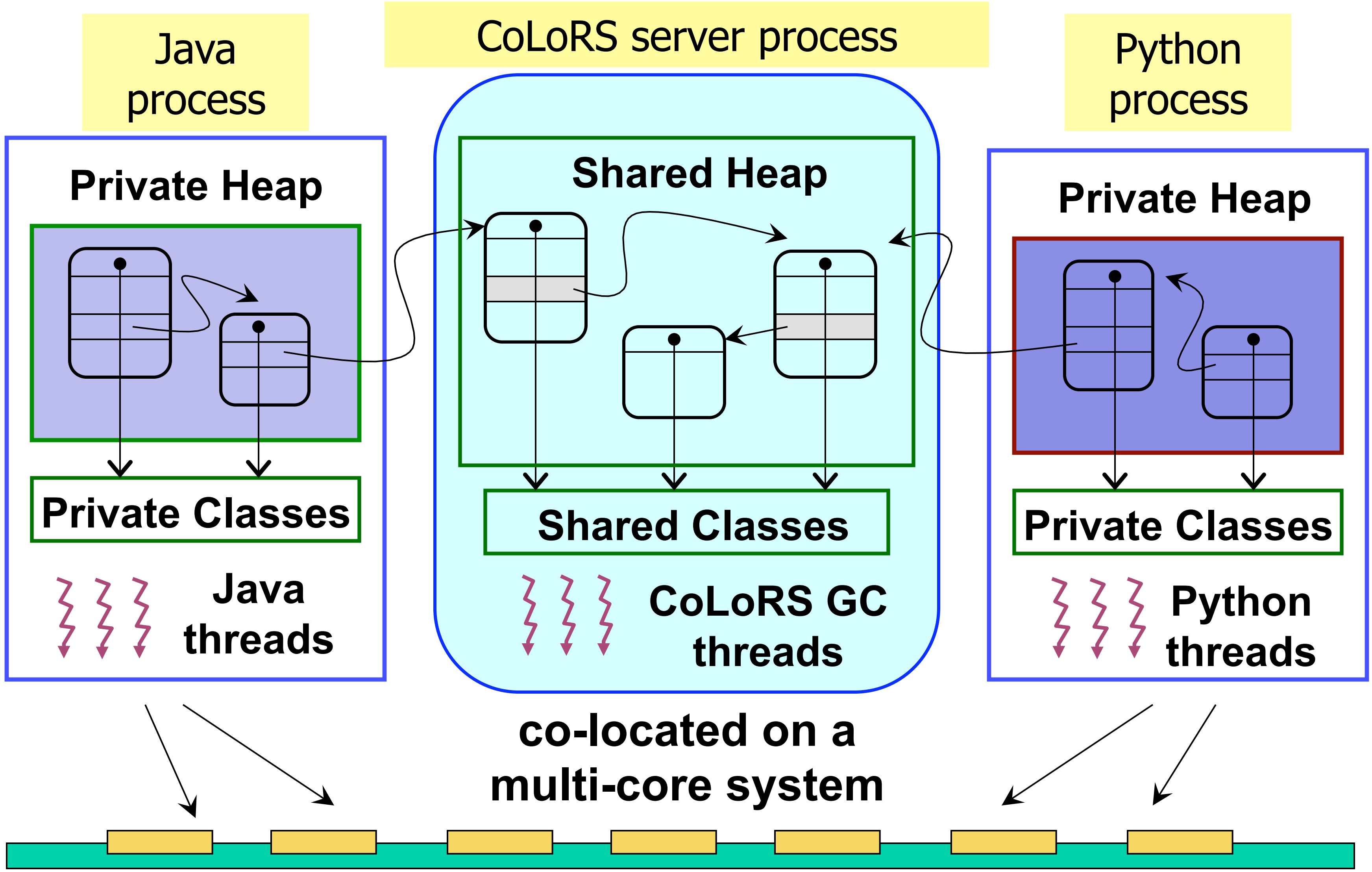
UCSB

Cross-language Interoperability

- Python, Javascript, Perl, PHP, Ruby, Java, C/C++, .Net, …
  - Mixed-environment debugging
- Cross-language/process communications technology
  - RPC, messaging
    - Thrift, HTTP/s, REST, SOAP, RPC, COM, SIP, SWIG, CORBA
    - For more than just web services: Map-Reduce (MR), MR-streaming, MPI
  - Data exchange formats
    - Protocol Buffers, XML, JSON
  - Exploit co-location of runtimes and virtual machines (system-level, guest VMs)
    - **CoLoRS** – Co-Located Runtime Sharing (OOPSLA'10)
      - Direct, type-safe object sharing across language runtimes
      - Transparent / automatic replacement of high overhead RPC and messaging protocols

UCSB

Co-located Runtime Sharing (CoLoRS)



UCSB

CoLoRS Contributions

- Object and memory model
  - Objects and classes shared between programs written in dynamic and static languages
  - Static-dynamic hybrid: efficiency with flexibility of dynamic class modifications via versioning and type mapping
- Type system
  - Preserves language-specific type-safety w/o new type rules
- Shared-memory garbage collector
  - Parallel, concurrent, on-the-fly GC that guarantees termination
    - No system-wide pauses, non-moving
- Synchronization in shared-memory
  - Simple, fast, yet same semantics as monitor synchronization
- CoLoRS support for HotSpot, cPython, and C++
  - Requires runtime modification, C++ source2source translation

CoLoRS Benefits

- CoLoRS support for HotSpot, cPython, and C++
  - 2-5% overhead: virtualization of memory access, write barriers
  - For co-located runtime communication performance
    - Multiple orders of magnitude improvements in latency
    - And throughput:

	Throughput in calls/ms; CoLoRS/RPC in parenthesis							
	bool	int	float	string	nodes:1	nodes:2	nodes:3	nodes:4
CORBA	173.22 (11)	82.67 (26)	83.20 (27)	75.96 (15)	14.67 (13)	4.68 (15)	1.83 (17)	0.86 (17)
ProtoBuf	31.73 (59)	30.98 (70)	34.32 (65)	26.43 (43)	2.85 (68)	0.88 (78)	0.36 (85)	0.17 (91)
REST	23.17 (81)	22.45 (97)	21.89 (102)	22.94 (50)	8.73 (22)	2.66 (26)	0.91 (34)	0.31 (49)
Thrift	237.04 (8)	283.23 (8)	274.37 (8)	149.08 (8)	15.38 (13)	4.27 (16)	1.80 (17)	0.87 (17)
CoLoRS	1876.08 (1)	2175.32 (1)	2231.45 (1)	1144.87 (1)	193.66 (1)	68.61 (1)	30.61 (1)	15.08 (1)

    - Due to avoidance of data serialization
      - Not due simply to the use of shared memory surprisingly
        - Localhost communication is optimized in Linux (0-copy)

Cross-language Interoperability

- Python, Javascript, Perl, PHP, Ruby, Java, C/C++, .Net, …
  - Mixed-environment debugging
- Cross-language/process communications technology
  - RPC, messaging
    - Thrift, HTTP/s, REST, SOAP, RPC, COM, SIP, SWIG, CORBA
    - For more than just web services: Map-Reduce (MR), MR-streaming, MPI
  - Data exchange formats
    - Protocol Buffers, XML, JSON
  - Exploiting co-location of runtimes and virtual machines (system-level, guest VMs)
    - CoLoRS – Transparent (or programmatic), type-safe sharing of objects across different language runtimes that are co-located on the same physical system
    - VSHMem – shared memory support for Xen

Modern Apps and Software

- Python, Javascript, Perl, PHP, Ruby, Java, C/C++, .Net, R
  - Modular, componentized, easily distributed
- Cross-language/process communications technology
  - Efficient RPC, messaging programmatically & when distributed
  - Transparent shared memory when co-located

- Requires distributed runtime support for
  - Efficient and scalable interoperation of components
    - Elasticity, load balancing, code/data/component scheduling, resource utilization, optimization, …
  - **Our approach: Cloud computing**
    - Remote/easy access to distributed and shared cluster resources
      - CPU/storage/network resources
    - Infrastructures, platforms, software "*as-a-Service*"

UCSB

3 types of cloud computing

- Infrastructure: Amazon Web Services (EC2, S3, EBS)
  - Virtualized, isolated (CPU, Network, Storage) systems on which users execute entire runtime stacks
    - Fully customer self-service
  - Open APIs (IaaS standard), scalable services
- Platform: Google App Engine, Microsoft Azure
  - Scalable program-level abstractions via well-defined interfaces
  - Enable construction of network-accessible applications
  - Process-level (sandbox) isolation, complete software stack
- Software: Salesforce.com
  - Applications provided to thin clients over a network
  - Customizable

UCSB

Cloud Computing

- Remote access to distributed and shared cluster resources
  - Has experienced a rapid uptake in the commercial sector
    - **Public clouds** – your software/apps on others' systems
    - Users **rent** a small fraction of vast resource pools
      - Advertised service-level-agreements (SLAs)
      - Resources are **opaque** and **isolated**
    - Offer high availability, fault tolerance, and extreme scale
    - **Private clouds**
      - Virtualized cluster management for local clusters
      - Support for elasticity (growing and shrinking of resource use)
      - Avoid vendor lock-in, facilitate test-drives -- features of public clouds are also useful in private setting

UCSB

Cloud Computing from UCSB

- Open source private cloud solutions
  - That implement the **open APIs of popular public clouds**
    - **Eucalyptus** – open source implementation of Amazon Web Services (AWS) over Xen, KVM, VMWare (Dr. Rich Wolski)
    - **AppScale** – open source implementation of Google App Engine for execution over Xen, KVM, Eucalyptus, AWS

    - Provide familiarity and easy transparent use
      - Engenders a large user community
    - **Hybrid (public-private) cloud support**
    - Leverage extant software offerings and multiple languages
    - Facilitate use of clouds technologies for more than just web services: HPC, data-intensive computing

Eucalyptus AppScale

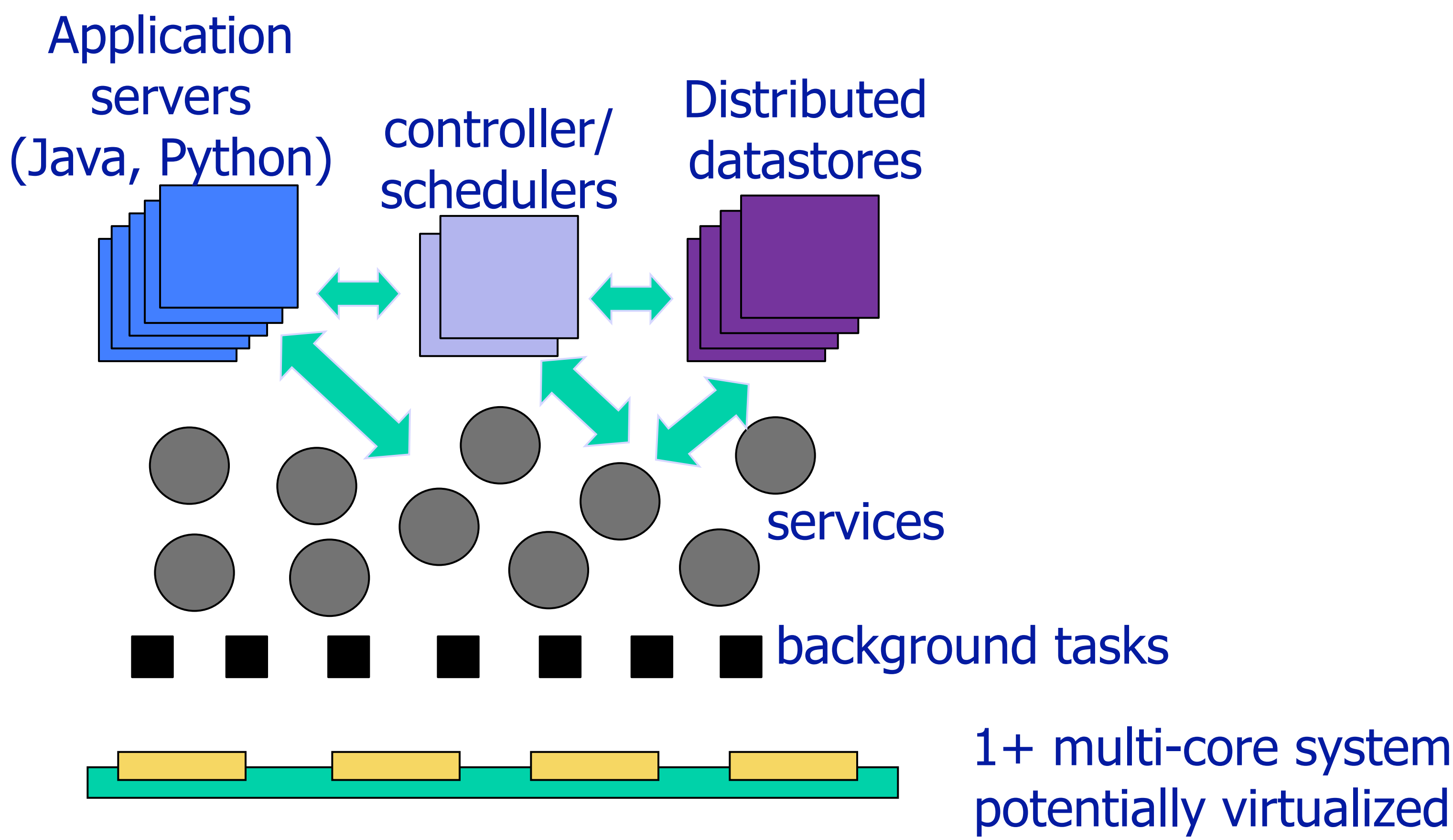
UCSB

Open Source Cloud Computing from UCSB

- **IaaS** Eucalyptus
  - Open-source implementation of all AWS APIs
  - Robust, highly-available, scalable emulation
  - Cluster/data center support over Xen, KVM, VMWare
  - www.eucalyptus.com Dr. Rich Wolski
- **PaaS:** AppScale
  - Open-source implementation of Google App Engine APIs
  - Pluggable (services), scalable, fault tolerant
  - Runs over virtualization or IaaS layer: AWS, Eucalyptus
  - appscale.cs.ucsb.edu

UCSB

AppScale Cloud Platform

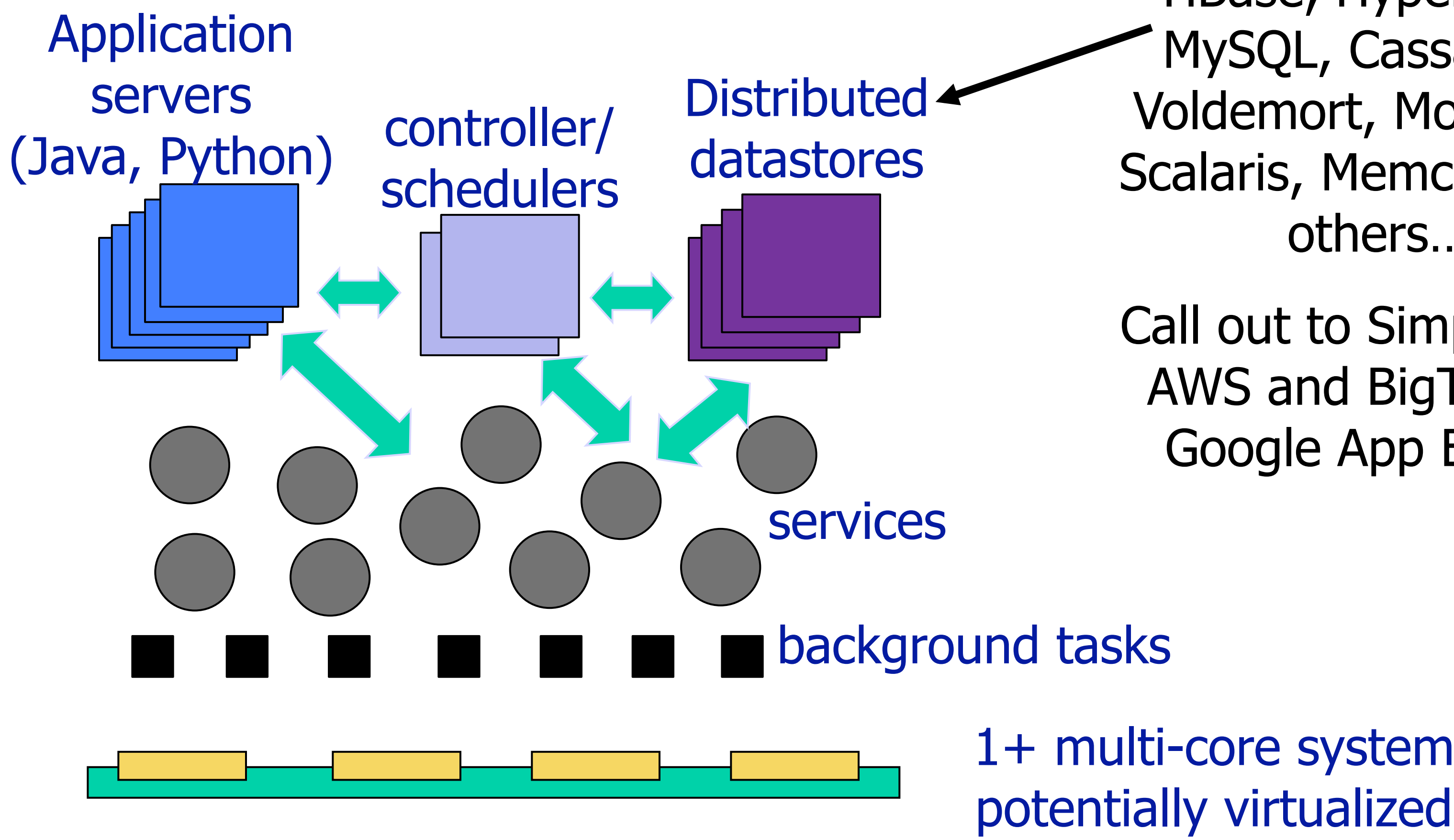


Pluggable

Elastic – grow and shrink with demand

Components run in one or more clouds (public and private)

AppScale Cloud Platform



HBase, Hypertable, MySQL, Cassandra, Voldemort, MongoDB, Scalaris, MemcacheDB, others…

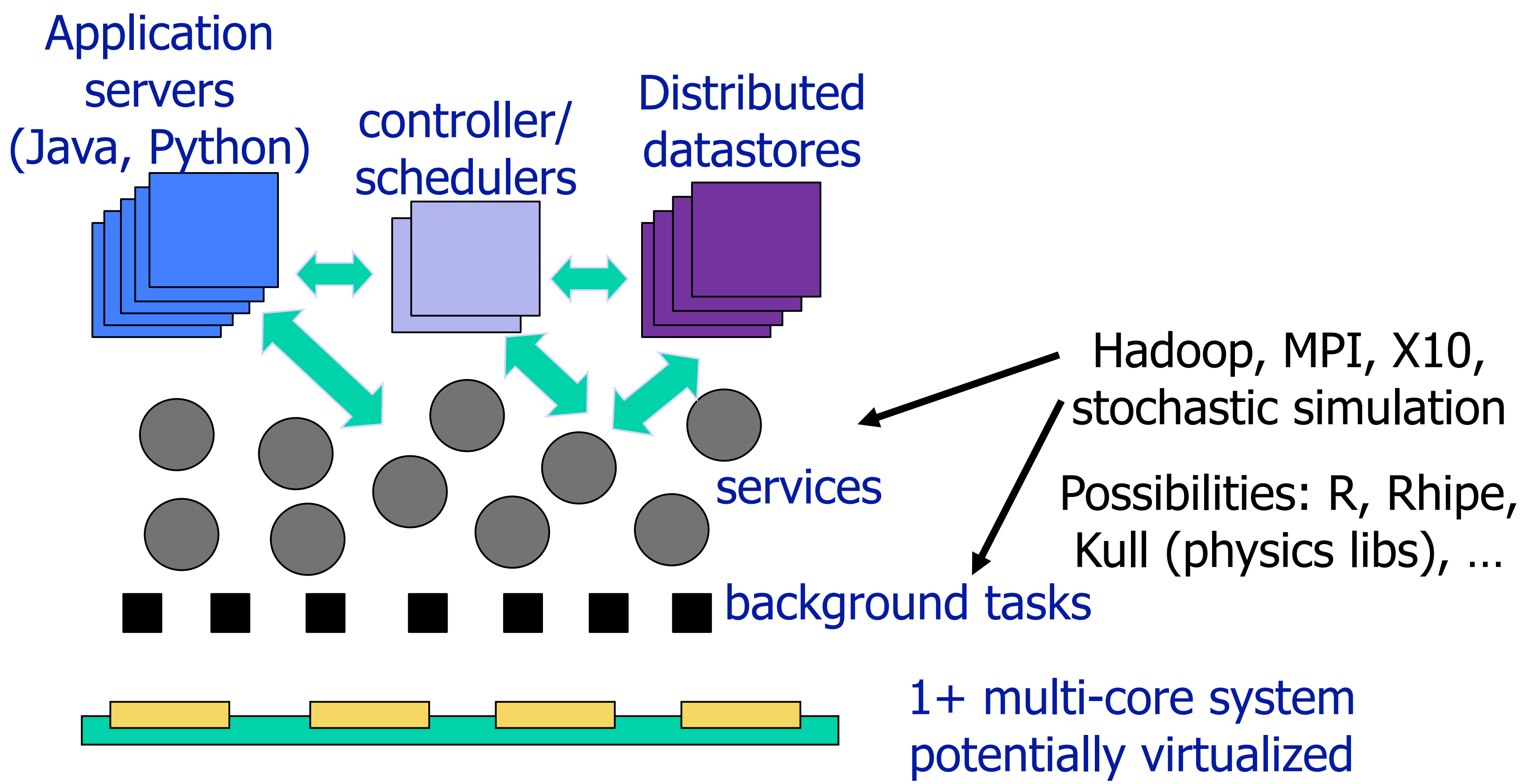
Call out to SimpleDB in AWS and BigTable in Google App Engine

Pluggable

Elastic – grow and shrink with demand

Components run in one or more clouds (public and private)

AppScale Cloud Platform



Pluggable

Elastic – grow and shrink with demand

Components run in one or more clouds (public and private)

UCSB

Summary

- Multi-language, multi-component software is here to stay
  - Dynamic and static languages must interoperate efficiently
  - Efficient technologies for cross-runtime communication
    - RPC, message-passing, object sharing via shared memory
- Distributed system support for easy deployment, scale
  - Cloud computing – remote access to cpu/storage/networking
  - Open source systems for private/hybrid cloud use
    - Bring benefits of cloud computing to local cluster resources
    - The same interfaces as public/proprietary clouds
- Together offer potential for new research and technological advance in high-performance and scientific computing
  - Use of dynamic languages in applications and systems
  - Profiling/monitoring, optimization, scaling, scheduling

Thanks!

- Students and Visitors!
  - Chris Bunch, Jovan Chohan, Navraj Chohan, Nupur Garg, Matt Hubert, Jonathan Kupferman, Puneet Lakhina, Yiming Li, Nagy Mostafa, Yoshihide Nomura (Fujitsu), Raviprakash Ramanujam, Michal Weigel
- Support
  - Google, IBM Research, National Science Foundation

http://www.cs.ucsb.edu/~racelab

http://appscale.cs.ucsb.edu/

- Extra slides on CoLoRS follow

CoLoRS Object Model

- Every value is an object in CoLoRS (no primitive types)
- Space-efficient static-dynamic hybrid object model
  - Versioning and type mapping
  - Matching based on type name and field set
    - Shared classes are read only
  - Versions for same class name
    - Different memory layout
    - Different field sets
    - Allows for fields to be dynamically added/removed
    - Shared objects class pointer may point to different versions
- Type system
  - Preserves language-specific type-safety w/o new type rules
    - Illegal field access on private type is not violated by mapping
  - No data definition language

UCSB

CoLoRS Usage

- Requires runtime extensions
  - Identify VM object/class model and its relationships to CoLoRS
    - Object model and GC
  - Virtualize object accesses
    - Separate shared/private path
    - Field accesses, method calls, synchronization
    - Insert calls to CoLoRS API
      - Prohibit shared to private ptrs
  - Define a type mapping for builtins and user-defined types

Shared	Java	Python
integer	byte,short,int, long, char, Byte, Short, Integer, Long, Character	int
float	float, double, Float, Double	float
boolean	boolean, Boolean	bool
string	String	str
binary	byte[]	bytearray
list	List, ArrayList, Object[], int[], float[],T[], …	list, tuple
set	Set, HashSet	set, frozenset
map	Map, HashMap	dict

UCSB

CoLoRS Usage (Continued...)

- Requires runtime extensions
  - Virtualization of library support for builtin types
    - For transparency of language-specific interfaces
  - Add a CoLoRS GC thread and shared-root-dump support
  - Setup TCP/IP server socket and shmem attach/detach

Shared	Java	Python
integer	byte,short,int, long, char, Byte, Short, Integer, Long, Character	int
float	float, double, Float, Double	float
boolean	boolean, Boolean	bool
string	String	str
binary	byte[]	bytearray
list	List, ArrayList, Object[], int[], float[],T[], …	list, tuple
set	Set, HashSet	set, frozenset
map	Map, HashMap	dict

UCSB

CoLoRS API

- Object copyToSharedMemory(Object root);
- Object allocate(Class objectClass);
- Object allocate(Class containerClass, int length);
- boolean isObjectShared(Object obj);
- ObjectRepository findOrCreateRepository(String key);
  - Repositories provide nonblocking get/set between VMs
  - Object reference exchange
- ObjectChannel findOrCreateChannel(String key);
  - Channels provide blocking send/receive between VMs
  - Object reference exchange
- Type getSharedType(Object obj);
  - For reflective inspection

Garbage Collection

- Goal: exploit available CPUS and avoid system-wide pauses
- CoLoRS GC
  - Parallel: multiple GC threads
  - Concurrent: most work is interleaved with program threads
  - Non-moving: requirement since many languages assume that objects do not move
    - Mark-sweep style
  - Snap-shot at the beginning (SATB)
  - Thread-local allocation buffers (TLABs)
- Extant approaches cannot be used in CoLoRS
  - Require multiple system-wide handshakes
  - Mutators must check whether they need to respond to handshakes during execution
  - Thread-level (CoLoRS requires VM-level operation)

Garbage Collection

- Goal: exploit available CPUS and avoid system-wide pauses
- CoLoRS GC
  - Parallel: multiple GC threads
  - Concurrent: most work is interleaved with program threads
  - Non-moving: requirement since many languages assume that objects do not move
    - Mark-sweep style
  - Snap-shot at the beginning (SATB)
  - Thread-local allocation buffers (TLABs)
  - Abstract private VM memory management to 1 operation
    - Shared root reporting (w/o any implementation requirements)
    - If this can be done without pausing the program
      - CoLoRS GC introduces zero pauses

Experimental Methodology

- Implemented in
  - openjdk6: HotSpot (server compiler and interpreter)
  - cPython
- Benchmarks
  - Overhead (no use of shared memory when available)
    - Java: Dacapo, SpecJBB
    - Python: PyBench, programming language shootout suite
  - Performance evaluation: Case study for RPC, messaging
    - Response time and throughput (call or transaction rate)
    - CORBA, Thrift, Protocol Buffers, and REST
      - Vs the same protocols with CoLoRS support
    - End-to-end server-client performance for two real applications
      - Cassandra datastore
      - Hadoop Distributed File System (HDFS)
      - Colors provides a cache

UCSB

CoLoRS Performance for Popular RPC Systems

- For different data types (nodes:x is a binary tree depth x)

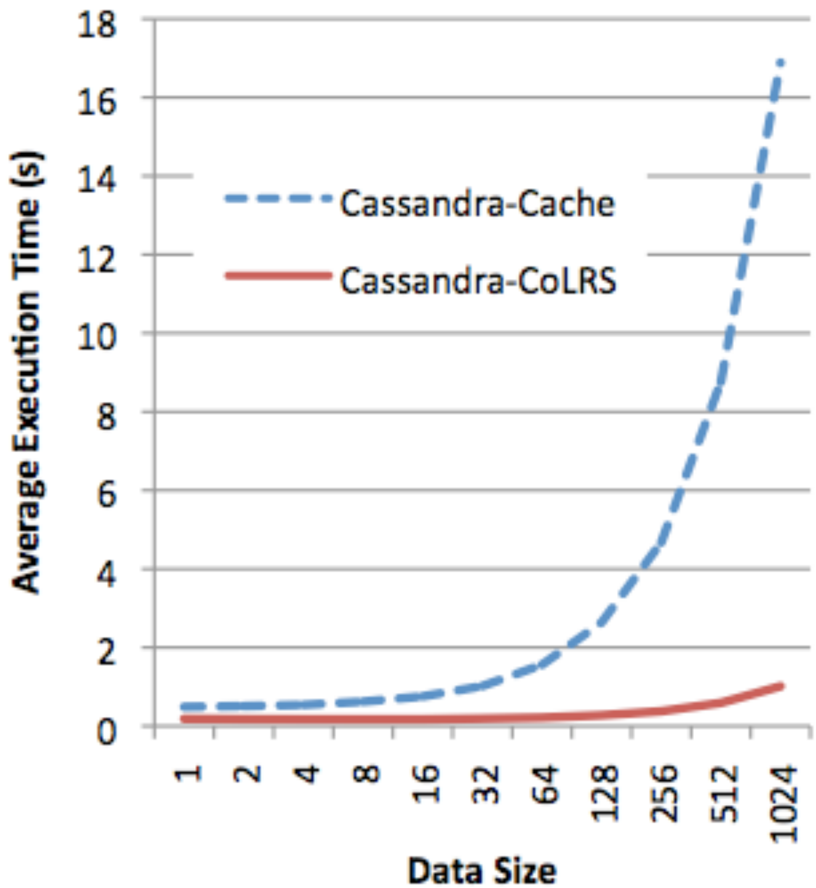
	Throughput in calls/ms; CoLoRS/RPC in parenthesis							
	bool	int	float	string	nodes:1	nodes:2	nodes:3	nodes:4
CORBA	173.22 (11)	82.67 (26)	83.20 (27)	75.96 (15)	14.67 (13)	4.68 (15)	1.83 (17)	0.86 (17)
ProtoBuf	31.73 (59)	30.98 (70)	34.32 (65)	26.43 (43)	2.85 (68)	0.88 (78)	0.36 (85)	0.17 (91)
REST	23.17 (81)	22.45 (97)	21.89 (102)	22.94 (50)	8.73 (22)	2.66 (26)	0.91 (34)	0.31 (49)
Thrift	237.04 (8)	283.23 (8)	274.37 (8)	149.08 (8)	15.38 (13)	4.27 (16)	1.80 (17)	0.87 (17)
CoLoRS	1876.08 (1)	2175.32 (1)	2231.45 (1)	1144.87 (1)	193.66 (1)	68.61 (1)	30.61 (1)	15.08 (1)

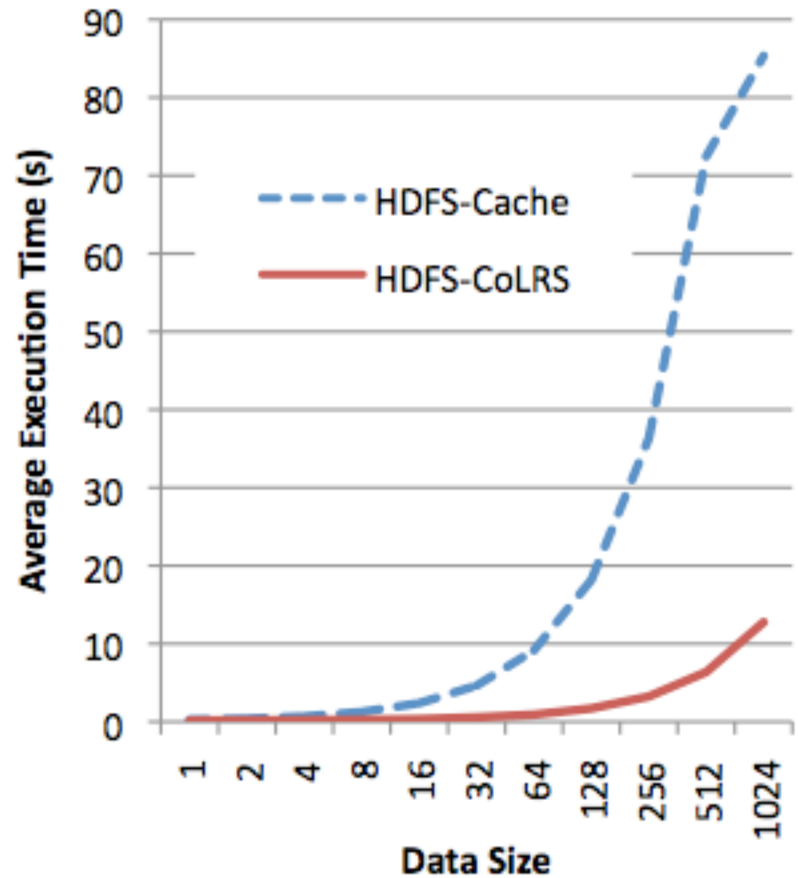
	Latency in msecs; RPC/CoLoRS in parenthesis							
	bool	int	float	string	nodes:1	nodes:2	nodes:3	nodes:4
CORBA	0.62 (14)	0.65 (19)	0.62 (14)	0.63 (14)	0.68 (17)	0.82 (15)	1.13 (17)	1.92 (19)
ProtoBuf	0.22 (5)	0.31 (9)	0.21 (5)	0.23 (5)	0.55 (14)	1.32 (23)	2.90 (44)	6.02 (58)
REST	3.89 (90)	3.89 (113)	4.00 (89)	3.92 (90)	4.07 (101)	4.80 (85)	7.35 (111)	9.94 (96)
Thrift	0.09 (2)	0.10 (3)	0.11 (3)	0.12 (3)	0.19 (5)	0.35 (6)	0.74 (11)	1.38 (13)
CoLoRS	0.04 (1)	0.03 (1)	0.04 (1)	0.04 (1)	0.04 (1)	0.06 (1)	0.07 (1)	0.10 (1)

- Performance gains due to serialization avoidance

UCSB

CoLoRS for Applications





- Performance gains due to serialization avoidance

UCSB

CoLoRS Overhead

- Due to virtualization of
  - Libraries (builtins)
  - Object field access
  - Synchronization
  - Method dispatch
  - Allocation/GC
- Provision of transparency
- When no sharing occurs

	Benchmark	Execution Time (s)	CoLoRS % Overhead
Python	binarytrees	6.79	3.39
	fannkuch	1.97	4.57
	mandelbrot	15.32	7.18
	meteorcontest	2.25	1.78
	nbody	8.67	2.08
	spectralnorm	14.31	5.73
	pybench	3.92	5.20
	pystone	4.09	5.87
	Geomean	**5.56**	**4.05**
Java	antlr	2.40	8.40
	bloat	6.34	6.30
	chart	6.19	6.10
	eclipse	24.54	4.70
	fop	2.11	7.70
	hsqldb	3.35	3.60
	jython	8.35	4.50
	luindex	7.50	9.00
	lusearch	4.25	1.40
	pmd	6.92	8.60
	xalan	5.97	0.00
	Geomean	**5.63**	**1.62**
		Throughput	
	jbb'00	112726.00	5.30
	jbb'05	54066.00	1.30
	Geomean	**78068.20**	**2.62**

UCSB