Towards Enabling High-Performance for Multi-Language Programs and Systems

Chandra Krintz

Laboratory for Research on

Adaptive Compilation Environments (RACE)

Computer Science Dept.

Univ. of California, Santa Barbara

PSI EtA (ψ η) Keynote

October 17, 2010

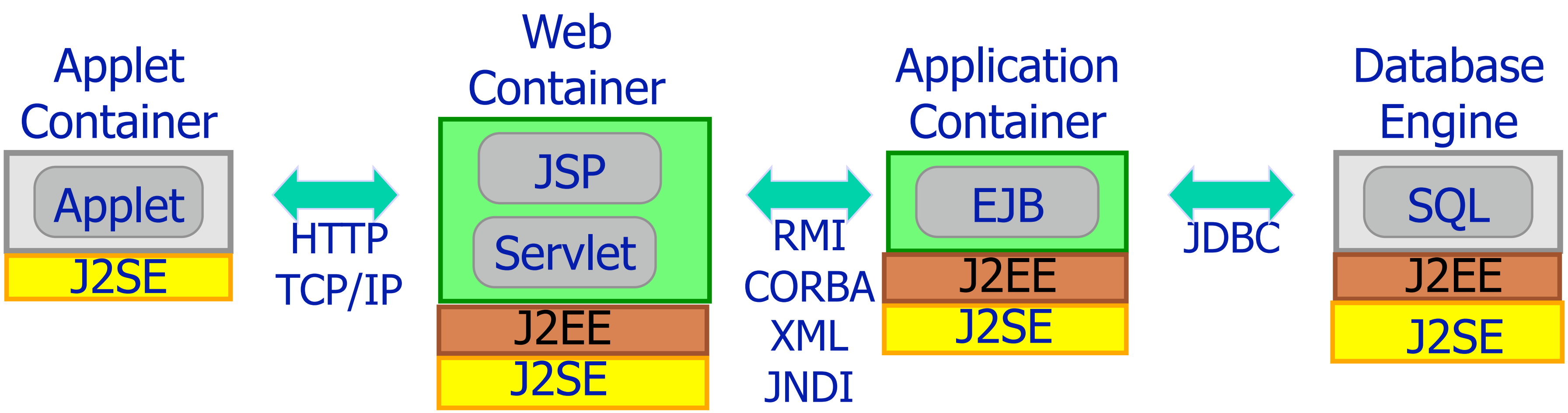
Modern Software & Systems: Recent Changes

- Hardware/architecture evolution
  - Low cost, high performance, memory-rich, multicore, virtualization support
- Distributed cluster computing
  - Web services, parallel/concurrent tasks, virtualized clusters (guestVMs), cloud computing
- The people who are developing applications/software
  - Productivity programmers vs specialists/experts
- Software as components, modules, tiers
  - Isolated via runtime and potentially virtual machine monitor
  - Reuse, mobility, multiple levels of fault tolerance, isolation

UCSB

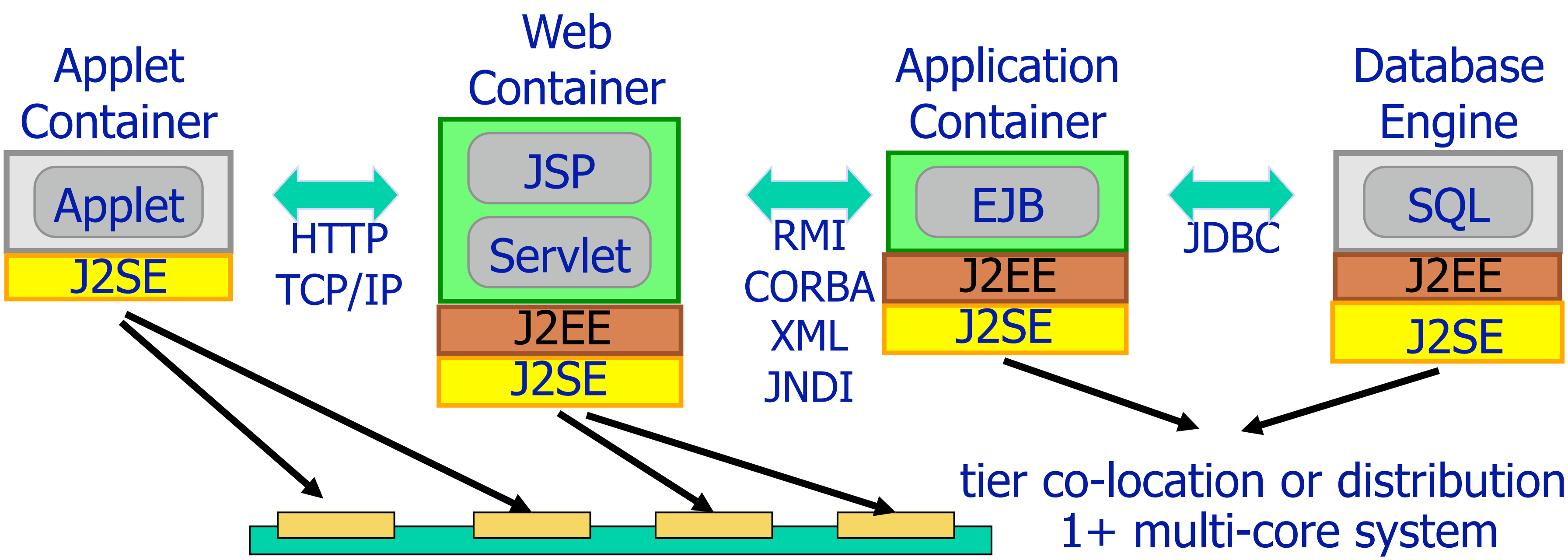
Modern Software and Systems

- Hardware/architecture evolution
- Distributed cluster computing
- Software as components, modules, tiers
  - Reuse, mobility, multi-level fault tolerance, isolation



Modern Software and Systems

- Hardware/architecture evolution
- Distributed cluster computing
- Software as components, modules, tiers
  - Reuse, mobility, multi-level fault tolerance, isolation



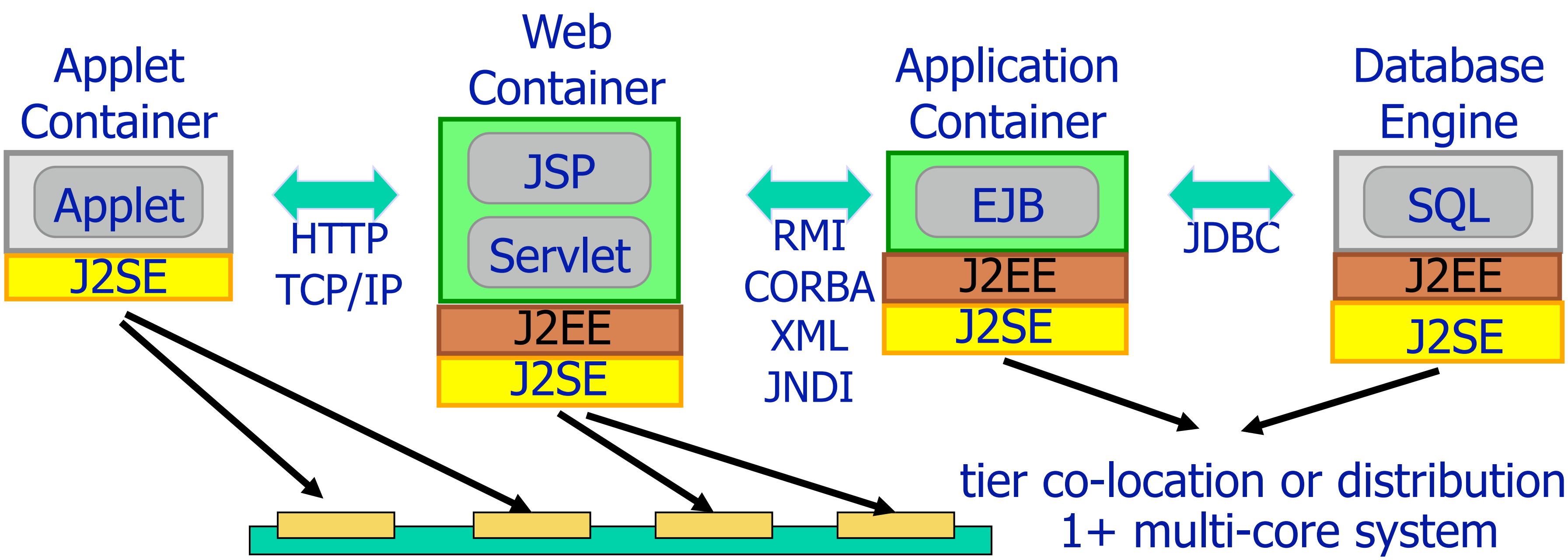
Modern Software and Systems

- Hardware/architecture evolution
- Distributed cluster computing
- Software as components, modules, tiers
  - Executed within own runtime and/or guestVM
    - Reuse, mobility, process-level fault tolerance, isolation
  - **Multi-language -- Web 2.0, web services, cloud systems**
    - Presentation layer: Javascript, Ruby, Java, Python
    - Server-side logic: PHP, Perl, Java, Python, Ruby
    - Computation: MapReduce streaming (multi-language)
    - Database, key-value store: C++, Java, + query languages
  - Next-generation distributed systems require support for
    - HPC: Python, Ruby, R -- with C, C++
    - Concurrency: Thorn, X10
  - Frameworks, IDES facilitate development and deployment

UCSB

Modern Software and Systems

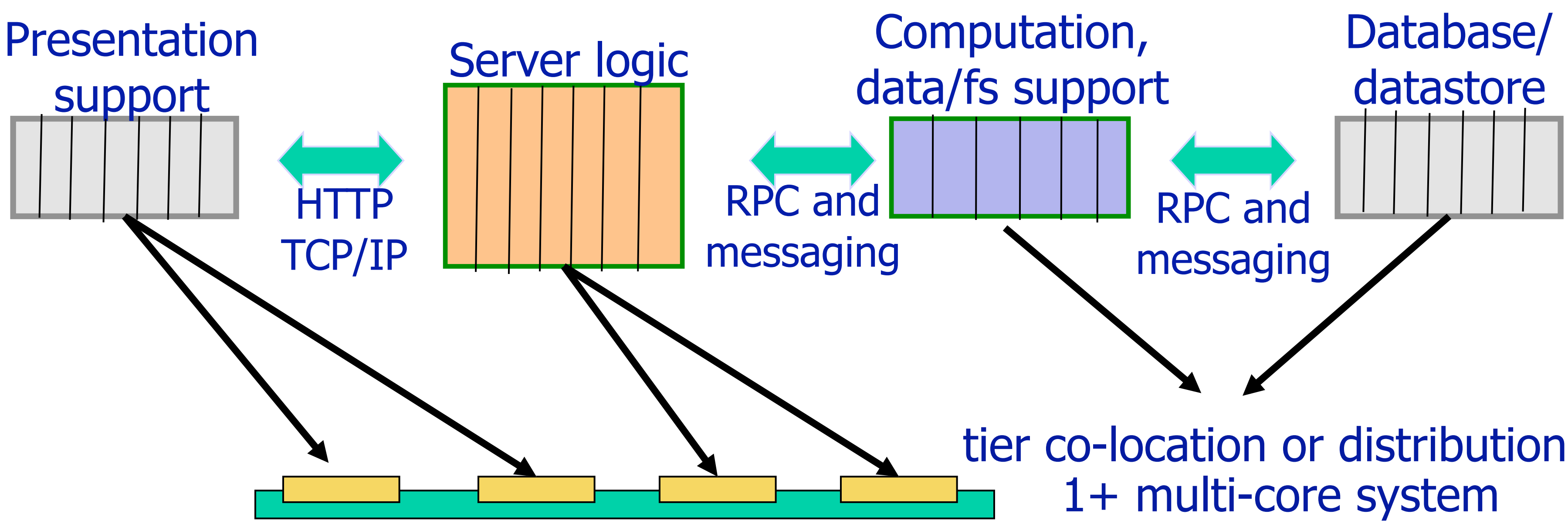
- Hardware/architecture evolution
- Distributed cluster computing
- Software as components, modules, tiers
  - Reuse, mobility, multi-level fault tolerance, isolation



Traditional Java Enterprise / Web 1.0

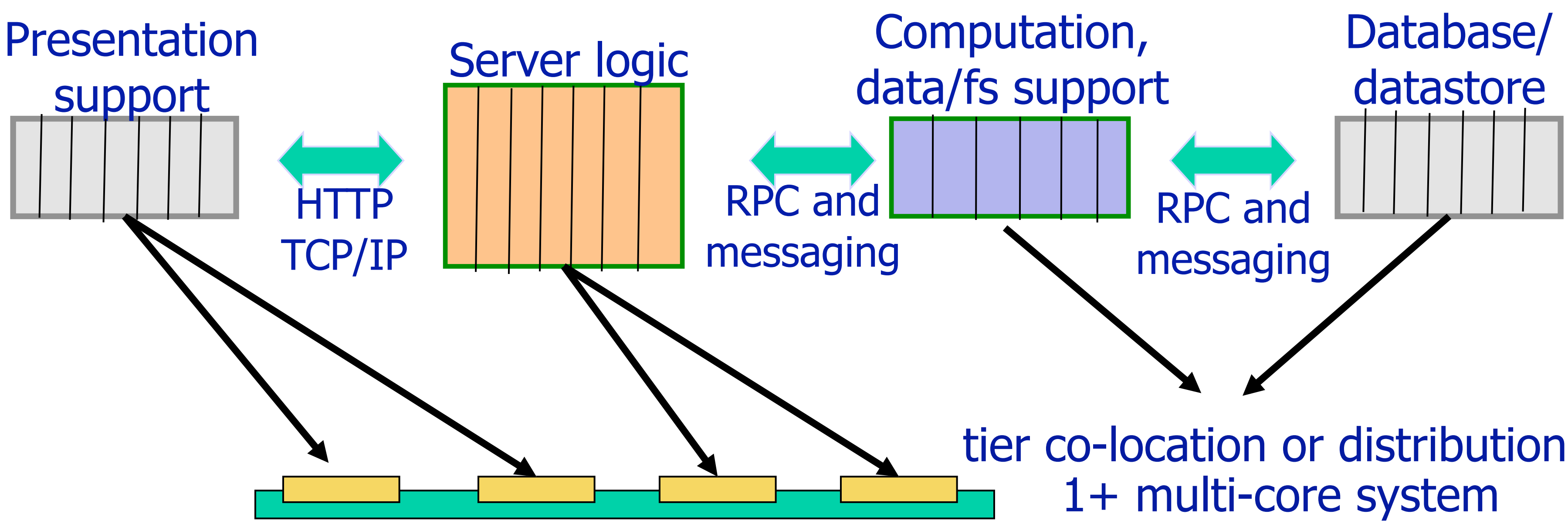
Modern Software and Systems

- Hardware/architecture evolution
- Distributed cluster computing
- Software as components, modules, tiers
  - Reuse, mobility, multi-level fault tolerance, isolation
  - Multi-language: Javascript,Ruby,Java,Python,PHP,C/C++,*QL,…



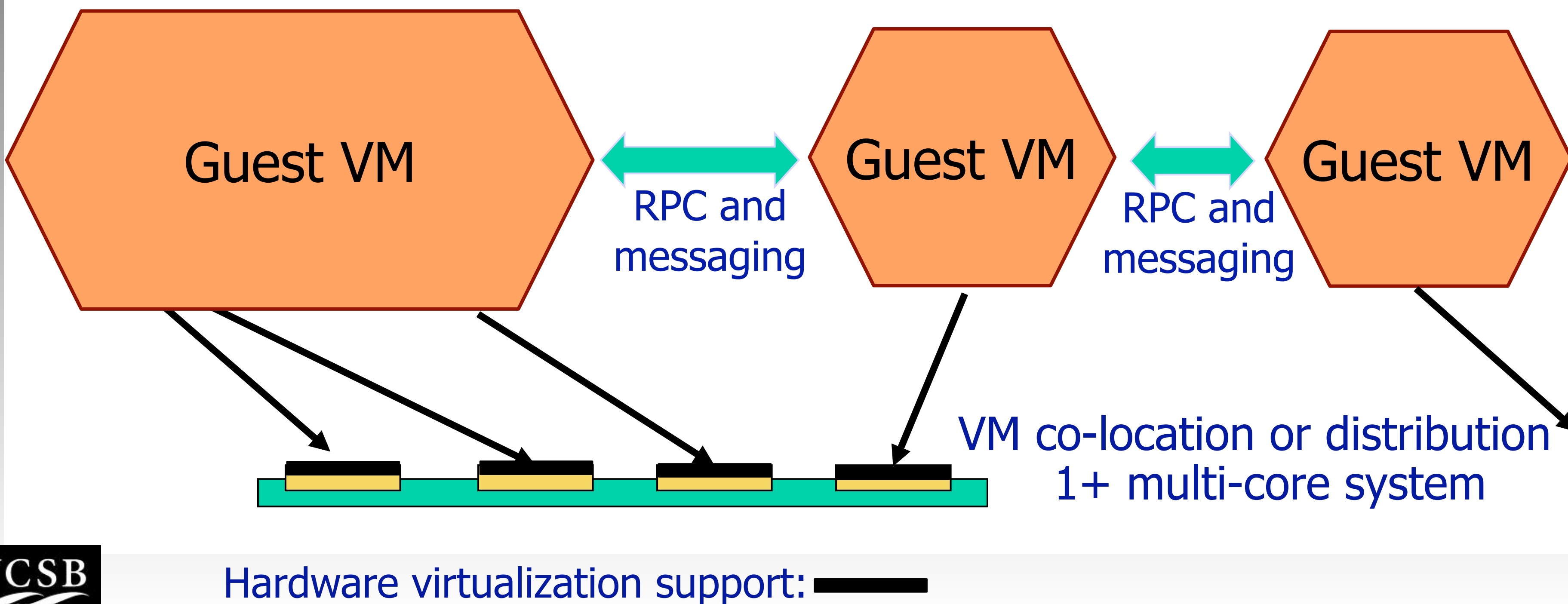
Modern Software and Systems

- Hardware/architecture evolution
- Distributed cluster computing
- Software as components, modules, tiers
  - Reuse, mobility, multi-level fault tolerance, isolation
  - Multi-language: Javascript,Ruby,Java,Python,PHP,C/C++,*QL,…



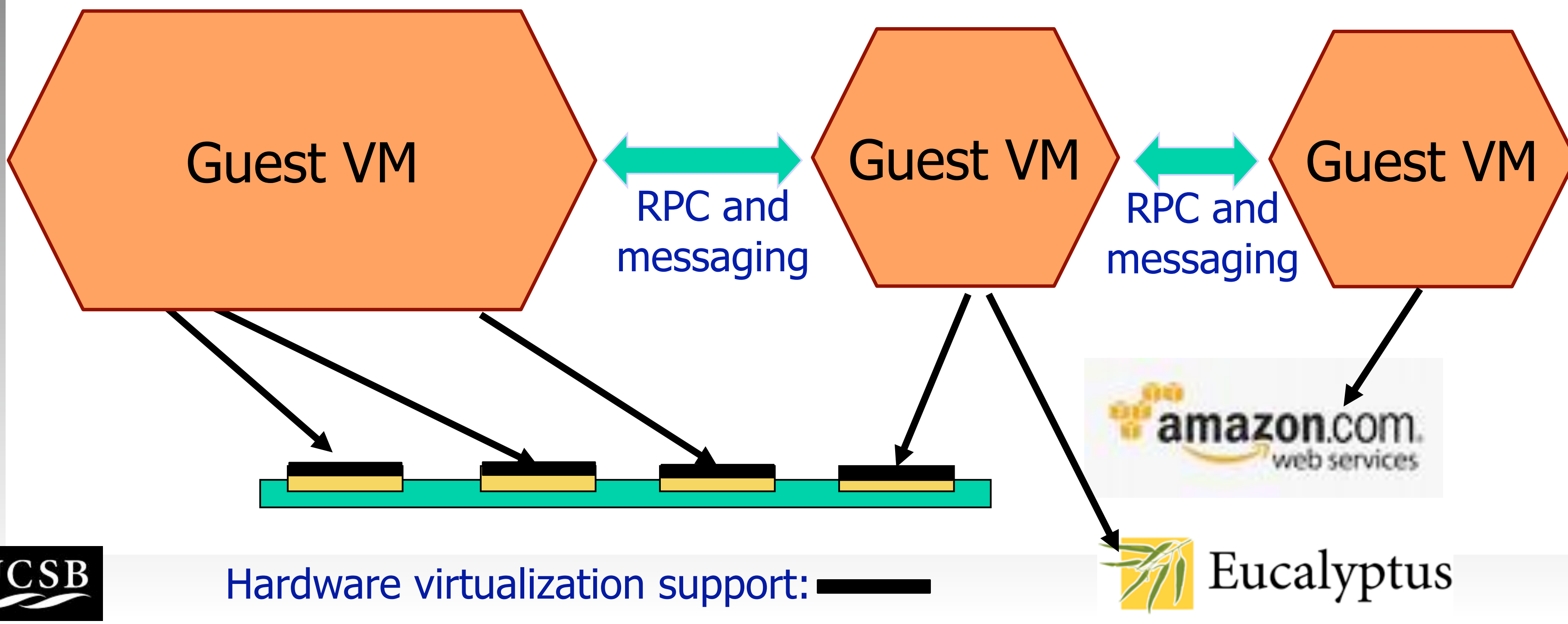
Modern Software and Systems

- Hardware/architecture evolution
- Distributed cluster computing
- Software as components, modules, tiers, guestVMs
  - Reuse, mobility, multi-level fault tolerance, isolation
  - Multi-language: Javascript,Ruby,Java,Python,PHP,C/C++,*QL,…
    - Within and across VMs



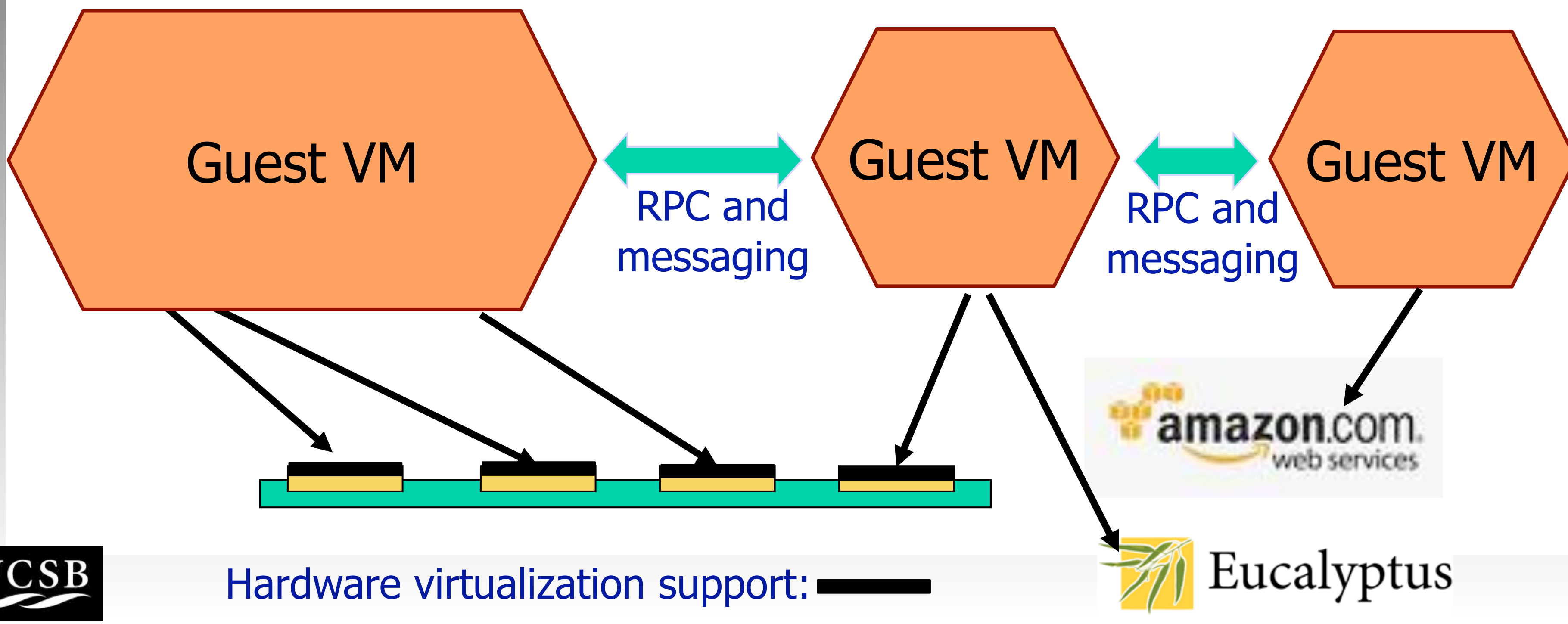
Modern Software and Systems

- Hardware/architecture evolution
- Distributed cluster computing
- Software as components, modules, tiers, guestVMs
  - Reuse, mobility, multi-level fault tolerance, isolation
  - Multi-language: Javascript,Ruby,Java,Python,PHP,C/C++,*QL,…
    - Within and across VMs



Modern Software and Systems

- Hardware/architecture evolution
- Distributed cluster computing
- Software as components, modules, tiers, guest VMs
  - Reuse, mobility, multi-level fault tolerance, isolation
  - **Multi-language**: Javascript,Ruby,Java,Python,PHP,C/C++,*QL,…
    - Within and across VMs



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)

UCSB

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) provide searchable support
    - Potential for viral, wide-spread use, free advertising
- Free software (open APIs)
  - Mashups, cloud/web services, software-as-a-service
- Available packages, libraries

Challenges to Modern Distributed Systems

- Traditional distributed systems problems
  - Fault tolerance/discovery, naming, scheduling/load balancing, synchronization, communication, compute/data locality
  - Integrated development, programmer productivity
  - Configuration & deployment
  - Isolation & quality of service
  - Monitoring, performance profiling, debugging
  - Performance optimization, scaling, & energy efficiency

UCSB

Challenges to Modern Distributed Systems

- Traditional distributed systems problems
  - Fault tolerance/discovery, naming, scheduling/load balancing, synchronization, communication, compute/data locality
  - Integrated development, programmer productivity
  - Configuration & deployment
  - Isolation & quality of service
  - Monitoring, performance profiling, debugging
  - Performance optimization, scaling, & energy efficiency
  - Only now, we need support for
    - Multiple languages and their runtime systems
    - Interoperation with extant services, software, systems
    - Pay-per-use (SLAs), cost (monetary, power/energy, time)
    - Portable execution on disparate software infrastructures

UCSB

Challenges to Modern Distributed Systems

- Traditional distributed systems problems
  - Fault tolerance/discovery, naming, scheduling/load balancing, synchronization, communication, compute/data locality
  - Integrated development, programmer productivity
  - Configuration & deployment
  - Isolation & quality of service
  - Monitoring, performance profiling, debugging
  - Performance optimization, scaling, & energy efficiency
  - Only now, we need support for
    - Multiple languages and their runtime systems
    - Interoperation with extant services, software, systems
    - Pay-per-use (SLAs), cost (monetary, power/energy, time)
    - **Portable execution on disparate software infrastructures**

UCSB

Challenges to Modern Distributed Systems

- Traditional distributed systems problems
  - Fault tolerance/discovery, naming, scheduling/load balancing, synchronization, **communication**, compute/data locality
  - Integrated development, programmer productivity
  - Configuration & deployment
  - Isolation & quality of service
  - Monitoring, performance profiling, debugging
  - Performance optimization, scaling, & energy efficiency
  - Only now, we need support for
    - **Multiple languages and their runtime systems**
    - Interoperation with extant services, software, systems
    - Pay-per-use (SLAs), cost (monetary, power/energy, time)
    - Portable execution on disparate software infrastructures

UCSB

Challenges to Modern Distributed Systems

- Traditional distributed systems problems
  - Fault tolerance/discovery, naming, scheduling/load balancing, synchronization, communication, compute/data locality
  - Integrated development, programmer productivity
  - Configuration & deployment
  - Isolation & quality of service
  - Monitoring, performance profiling, debugging
  - Performance optimization, scaling, & energy efficiency
  - Only now, we need support for
    - Multiple languages and their runtime systems
    - Interoperation with extant services, software, systems
    - Pay-per-use (SLAs), cost (monetary, power/energy, time)
    - **Portable execution on disparate software infrastructures**

UCSB

Develop/Deploy-ment of Distributed Apps/Services

- What is your ideal?
  - Write-once, run anywhere
    - Laptop, local cluster, across multiple clusters
    - In public/hybrid clouds: Amazon AWS, Eucalyptus clusters, Google App Engine, Microsoft Azure, … others
  - Wide variety of scalable, high-performance services & libraries
    - Well-defined APIs
  - Aware of
    - Cost
    - Price-performance or price-scale
  - Automatic
    - Scaling (of different metrics)
    - Performance optimization and customization
      - Component level, parallelization, load-balancing, cost
    - Deployment and configuration of libraries and services

Our Approach

- Leverage advances in cloud computing
- Cloud computing

Our Approach

- Leverage advances in cloud computing
- Cloud computing
  - Remote/easy access to distributed & shared cluster resources
    - Isolated CPUs, storage, networking, services made available via web interfaces

Our Approach

- Leverage advances in cloud computing
- Cloud computing
  - Remote/easy access to distributed & shared cluster resources
    - Isolated CPUs, storage, networking, services made available via web interfaces
  - Culmination of grid/cluster/utility/elastic computing
    - Exploits advances in processor, virtualization, systems technology
  - Public: pay-per-use (service level agreements (SLAs))
    - Users rent small fraction of resources owned by others
      - Amazon, Microsoft, Google, others…
  - Private: similar distributed system support for your cluster
    - Proprietary and open source options

Cloud computing

- 3 types: as-a-Service (aaS)
  - 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

Our Approach

- Leverage advances in cloud computing
- Why not just use extant cloud systems?

Our Approach

- Leverage advances in cloud computing
- Why not just use extant cloud systems?
  - Public
    - Privacy of code and data
    - Potential vendor "lock-in"
    - Cost (even though currently very low)
    - Availability reliance
    - Resource/application constraints
    - Opaque system, closed implementations
  - Private
    - Proprietary (cost), closed implementations
  - Open source
    - Infrastructure only (fully user self-service customization, deployment, etc.) – not necessarily developer focused

Our Approach

- Leverage advances in cloud computing
- Why not just use extant cloud systems?
  - Public
    - Privacy of code and data
    - Potential vendor "lock-in"
    - Cost (even though currently very low)
    - Availability reliance
    - Resource/application constraints
    - Opaque system, closed implementations
  - Private
    - Proprietary (cost), closed implementations
  - Open source
    - Infrastructure only (fully user self-service customization, deployment, etc.) – not necessarily developer focused

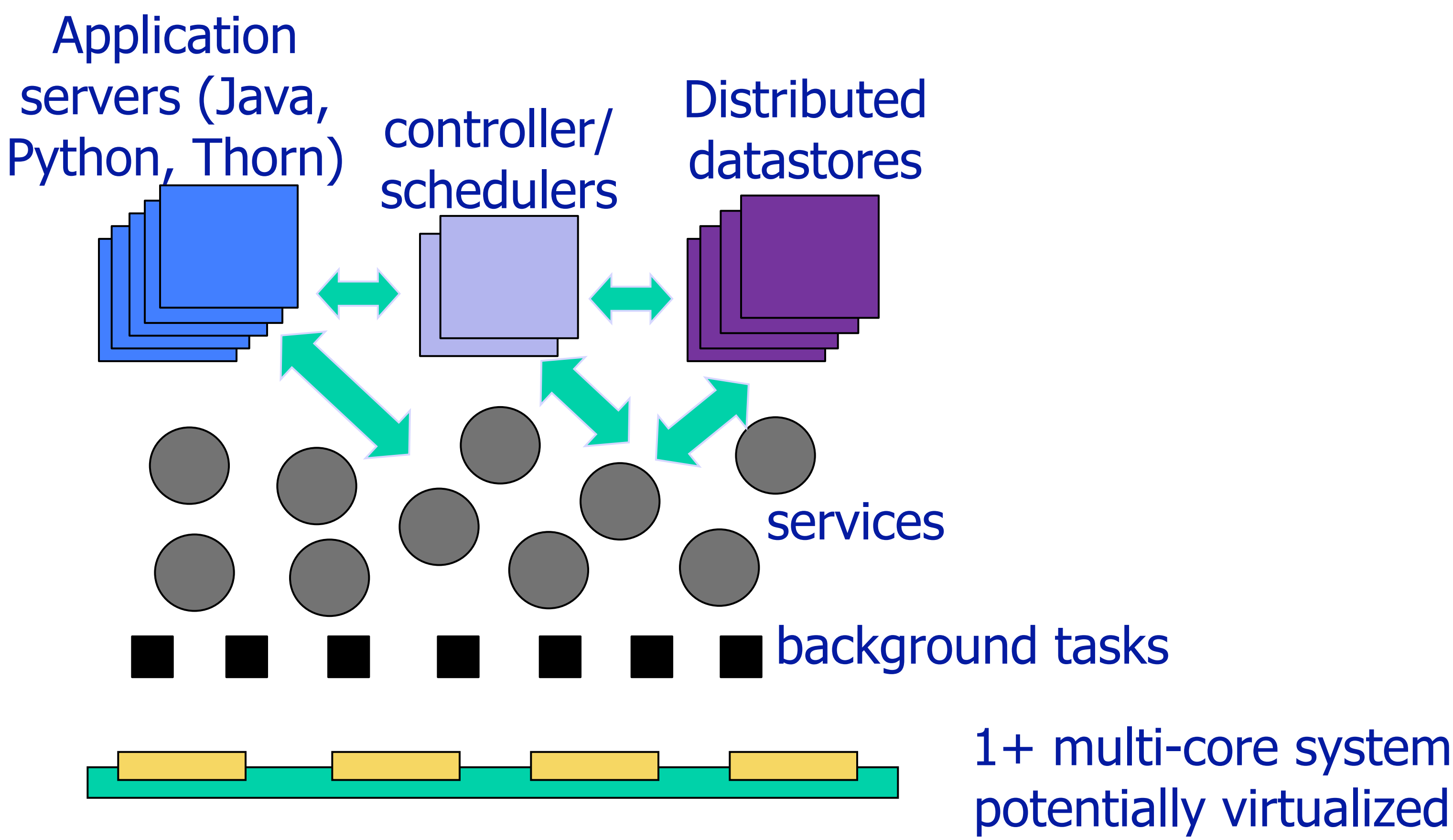
☹ **Lots of non-standard APIs!**

Our Approach: Portable Cloud Platform

- Leverage advances in cloud computing
- AppScale (http://appscale.cs.ucsb.edu)
  - Implementation of different extant cloud APIs
    - Using different programming languages
    - Starting place: Google App Engine (GAE) – familiarity, users, apps
  - Execution over
    - Cloud infrastructures: Amazon Web Services, Eucalyptus
    - Cloud platforms: GAE, Azure (under development)
    - Virtualization layers: Xen, KVM, none
  - Automatic
    - Configuration and deployment of libraries and services
    - Monitoring of distributed system performance data
  - Test drive in a private setting before moving to a public cloud
    - Evaluate different cloud services

UCSB

AppScale: Cloud Platform Portability



Pluggable

Elastic – grow and shrink with demand

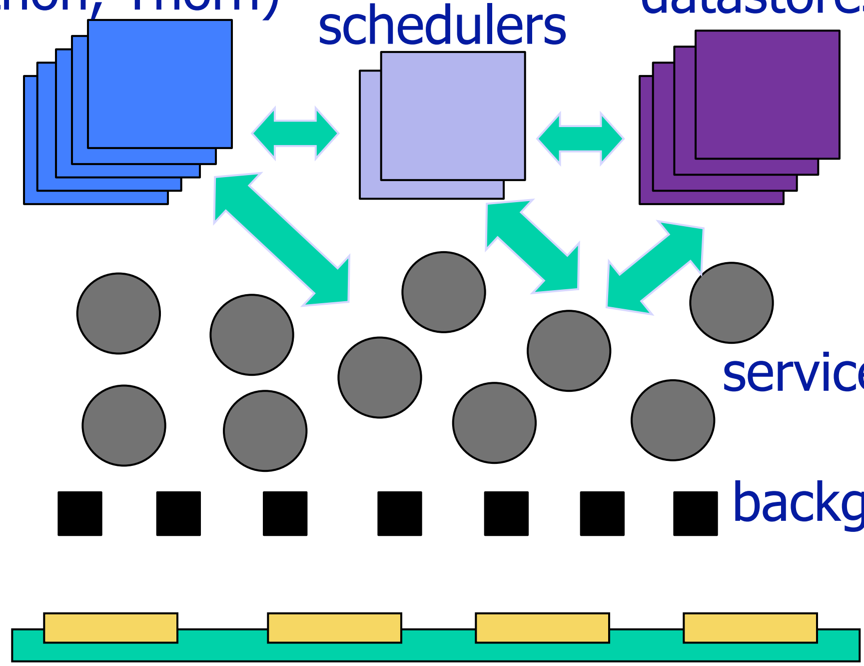
Components run in one or more clouds (public and private)

AppScale: Cloud Platform Portability

Application servers (Java, Python, Thorn)

controller/ schedulers

Distributed datastores



services

background tasks

1+ multi-core system potentially virtualized

Pluggable

Elastic – grow and shrink with demand

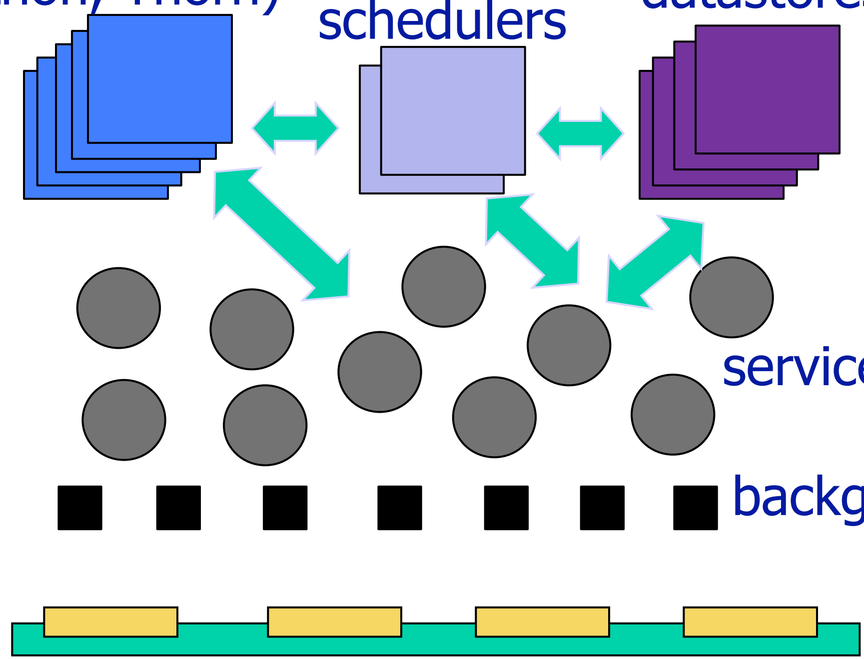
HBase, Hypertable, MySQL, Cassandra, Voldemort, MongoDB, Scalaris, MemcacheDB, others…

Transaction support

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

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

AppScale: Cloud Platform Portability



GAE APIs: Mail, images, user, messaging, tasks

As well as Hadoop, MPI, X10, queues, stochastic simulation

Others: R, Rhipe, Kull graph processing,…

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

Portable Cloud Platform Research

- Hybrid cloud support
  - Multi-cloud scheduling and scaling
  - Employ services from different cloud systems concurrently
- Multi-level monitoring and profiling
  - Static and dynamic language runtimes, HPMs, system level
  - Feedback directed optimization, scaling (up/down), ld balancing
- Transparent, portable execution
  - Laptop, your cluster, public and private clouds
- New application domains
  - HPC services & libraries, map-reduce, large-scale data analytics
- Cloud language support
  - For new and extant languages, cloud specific functionality

Challenges to Modern Distributed Systems

- Traditional distributed systems problems
  - Fault tolerance/discovery, naming, scheduling/load balancing, synchronization, communication, compute/data locality
  - Integrated development, programmer productivity
  - Configuration & deployment
  - Isolation & quality of service
  - Monitoring, performance profiling, debugging
  - Performance optimization, scaling, & energy efficiency
  - Only now, we need support for
    - Multiple languages and their runtime systems
    - Interoperation with extant services, software, systems
    - Pay-per-use (SLAs), cost (monetary, power/energy, time)
    - **Portable execution on disparate software infrastructures**

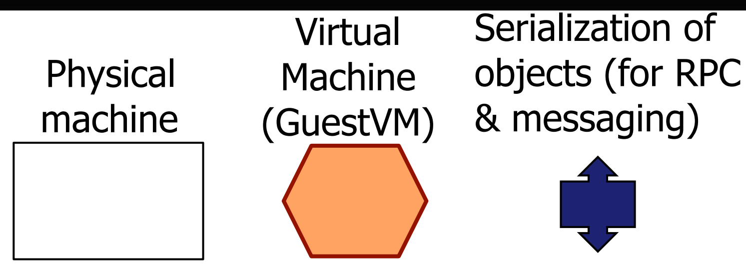
UCSB

Challenges to Modern Distributed Systems

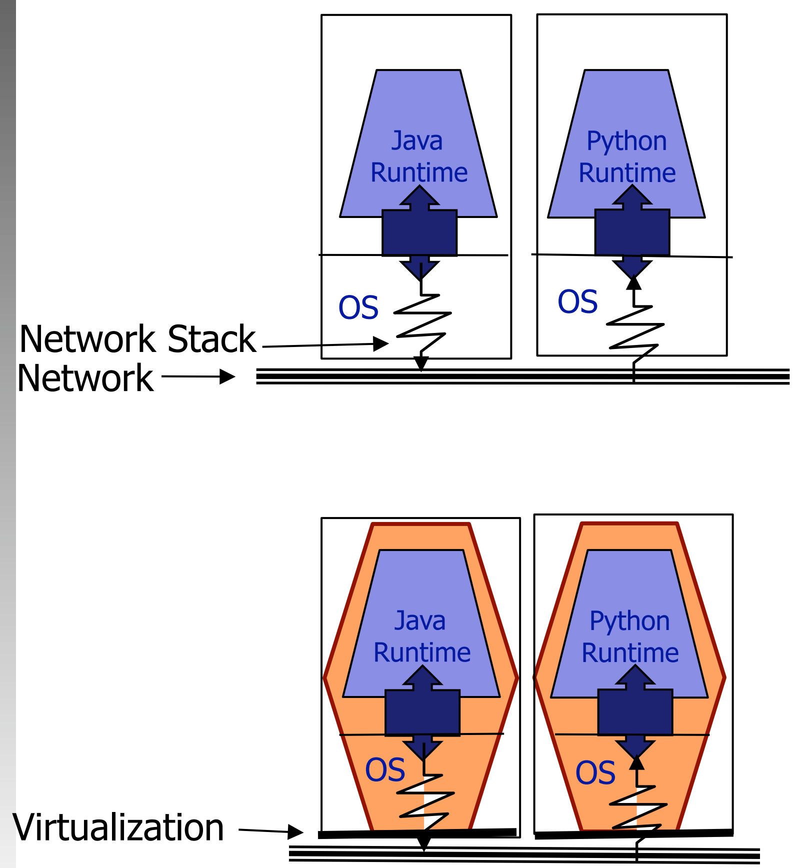
- Traditional distributed systems problems
  - Fault tolerance/discovery, naming, scheduling/load balancing, synchronization, **communication**, compute/data locality
  - Integrated development, programmer productivity
  - Configuration & deployment
  - Isolation & quality of service
  - Monitoring, performance profiling, debugging
  - Performance optimization, scaling, & energy efficiency
  - Only now, we need support for
    - **Multiple languages and their runtime systems**
    - Interoperation with extant services, software, systems
    - Pay-per-use (SLAs), cost (monetary, power/energy, time)
    - Portable execution on disparate software infrastructures

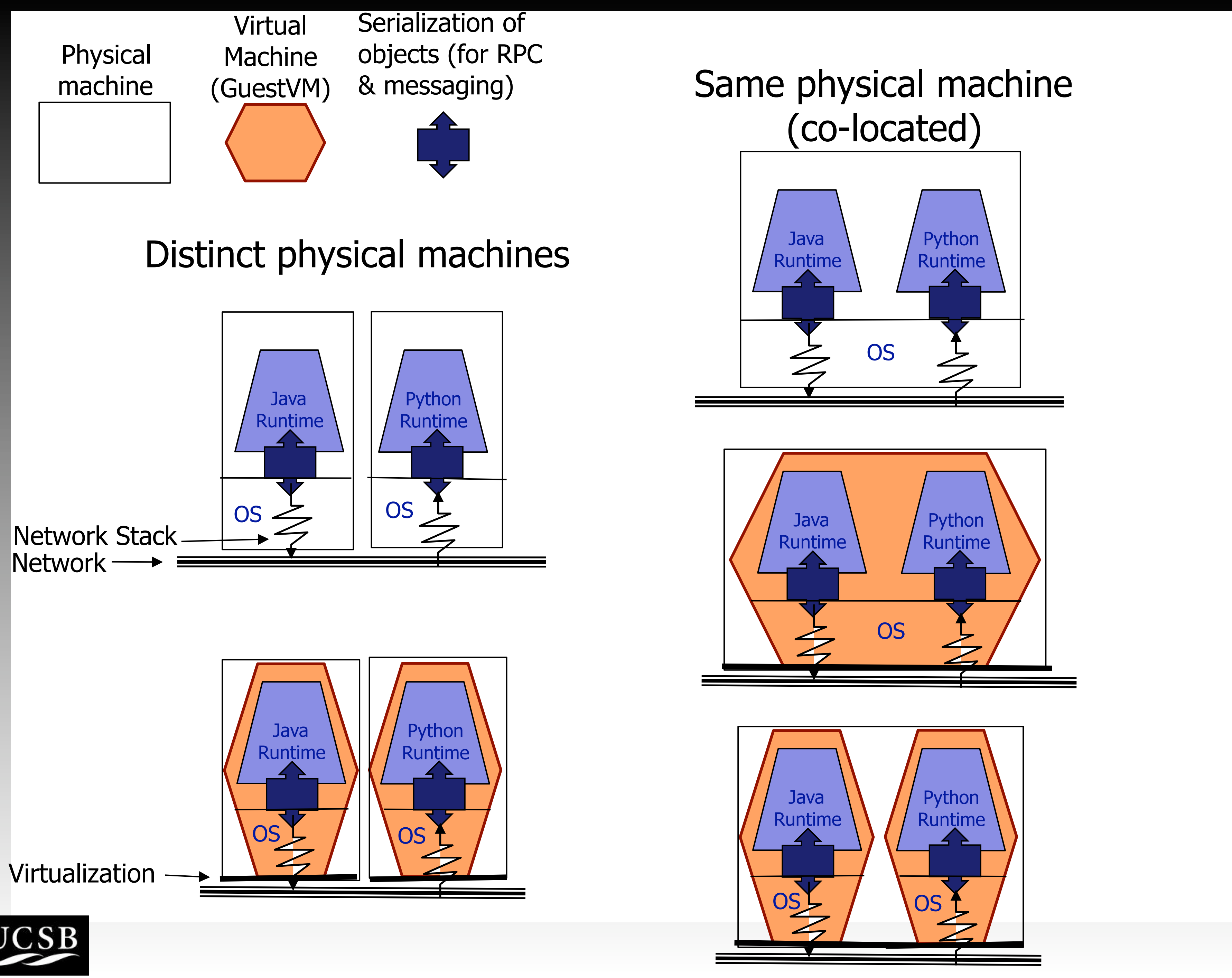
Efficient Cross-Language Communication

- Interoperating components can be executing…



Distinct physical machines



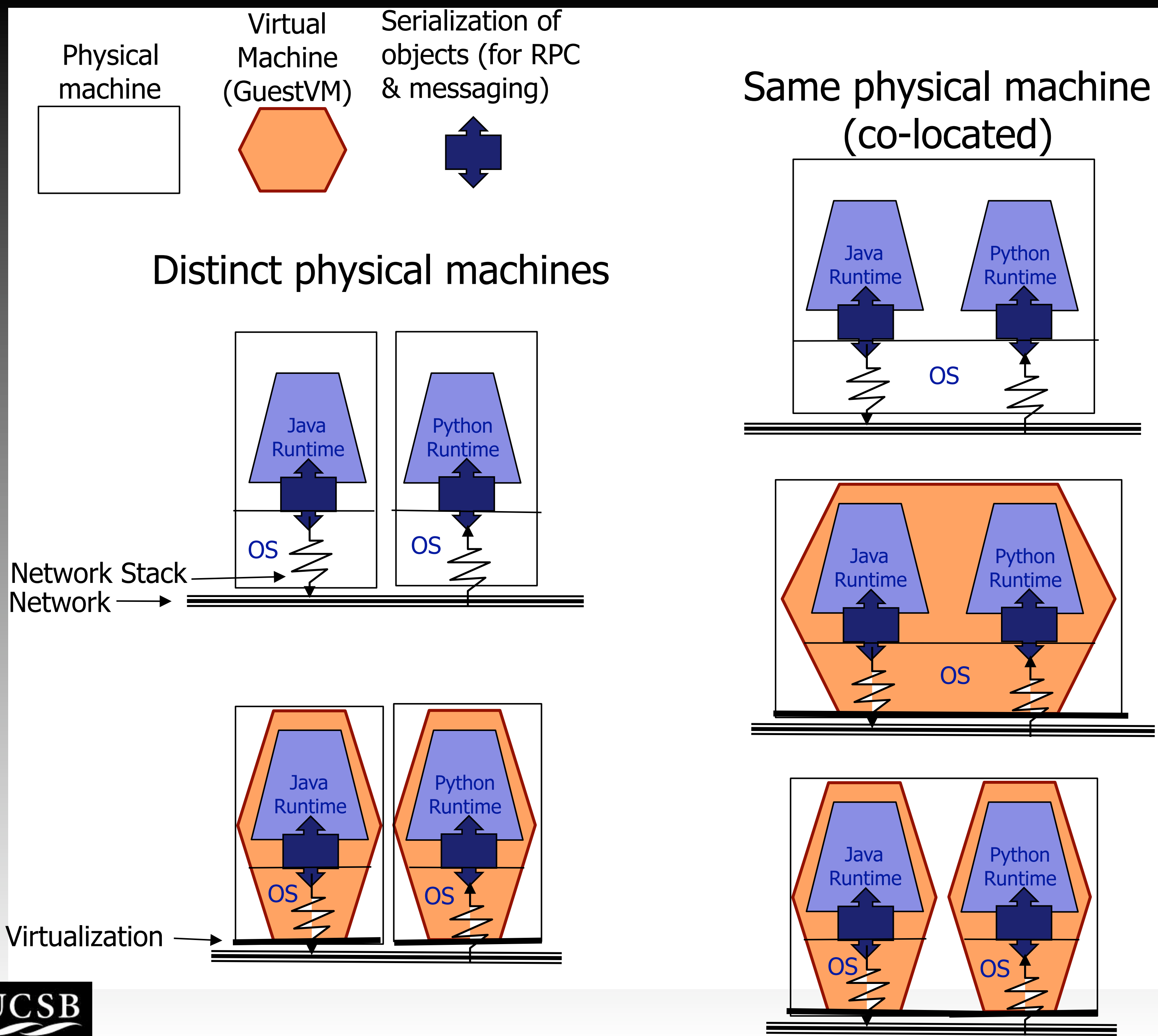


Physical machine
Virtual Machine (GuestVM)
Serialization of objects (for RPC & messaging)
Distinct physical machines
Same physical machine (co-located)
Java Runtime
Python Runtime
OS
Network Stack
Network
Virtualization
UCSB

Cross-language Communication & Coordination

- Python, Javascript, Perl, PHP, Ruby, Java, C/C++, .Net, …
- 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
  - Benefits from these technologies
    - Programmer productivity
      - Abstraction, portability, copy semantics
  - Limitations
    - Require serialization and encoding of data/objects
    - Network communication

UCSB


Physical machine
Virtual Machine (GuestVM)
Serialization of objects (for RPC & messaging)
Distinct physical machines
Same physical machine (co-located)
Java Runtime
Python Runtime
OS
Network Stack
Network
Virtualization

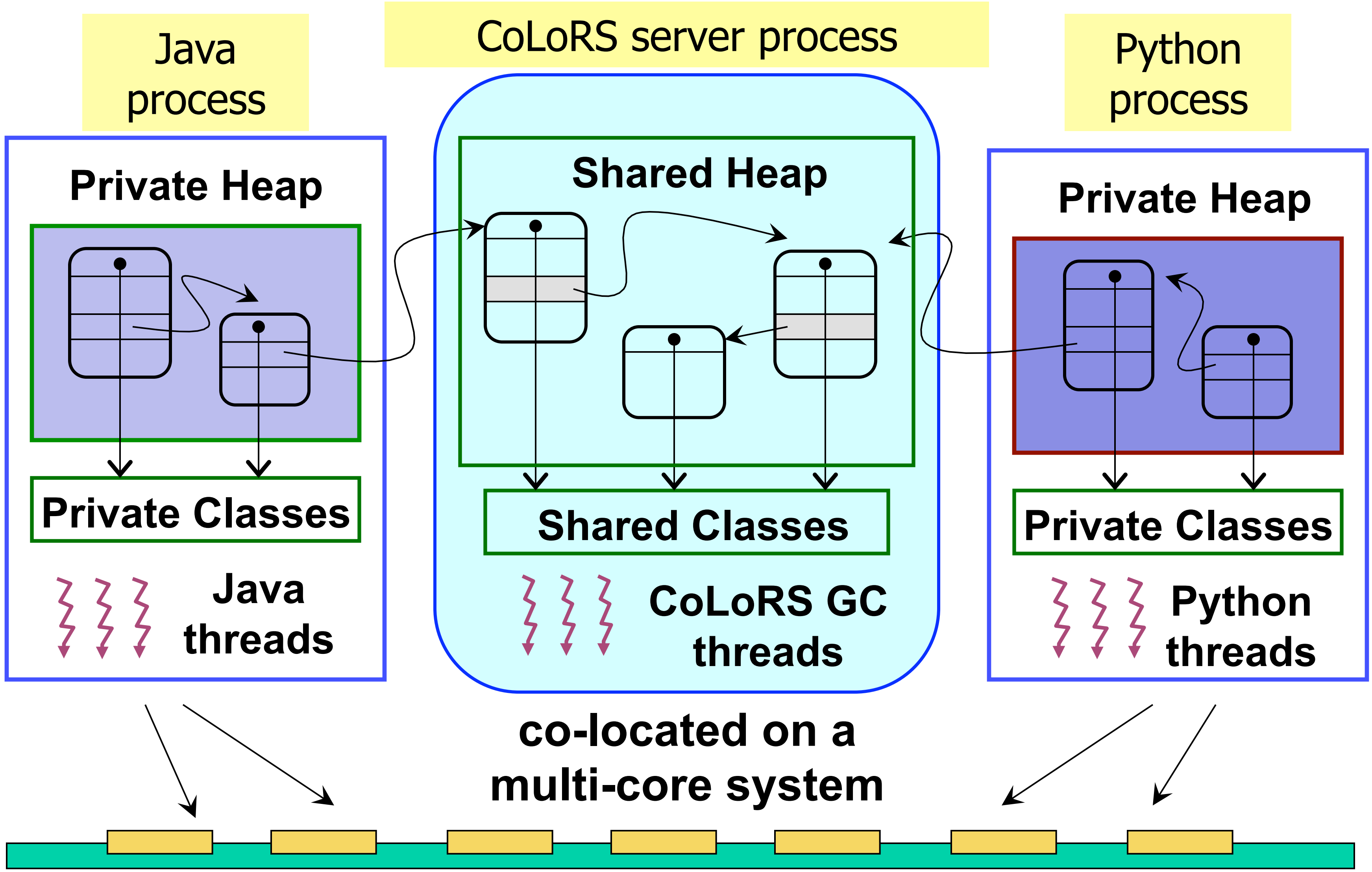
UCSB

Cross-language Communication & Coordination

- Python, Javascript, Perl, PHP, Ruby, Java, C/C++, .Net, …
- 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
    - **CoLoRS** – Co-Located Runtime Sharing (OOPSLA'10)
      - **Transparent / automatic replacement of high overhead RPC and messaging protocols**
      - Direct, type-safe object sharing across language runtimes is also possible

UCSB

Co-located Runtime Sharing (CoLoRS)

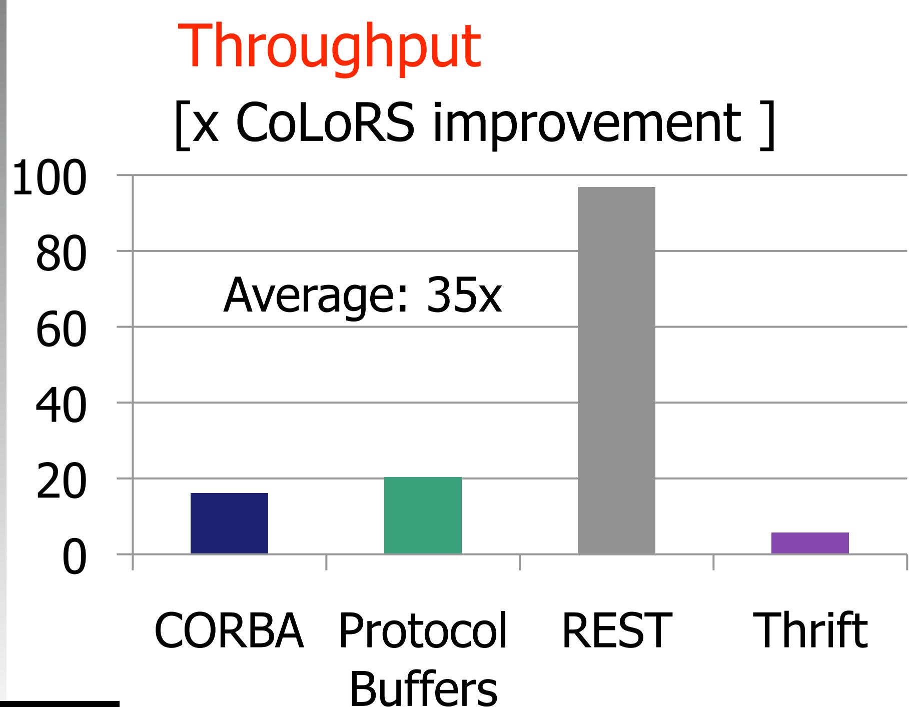


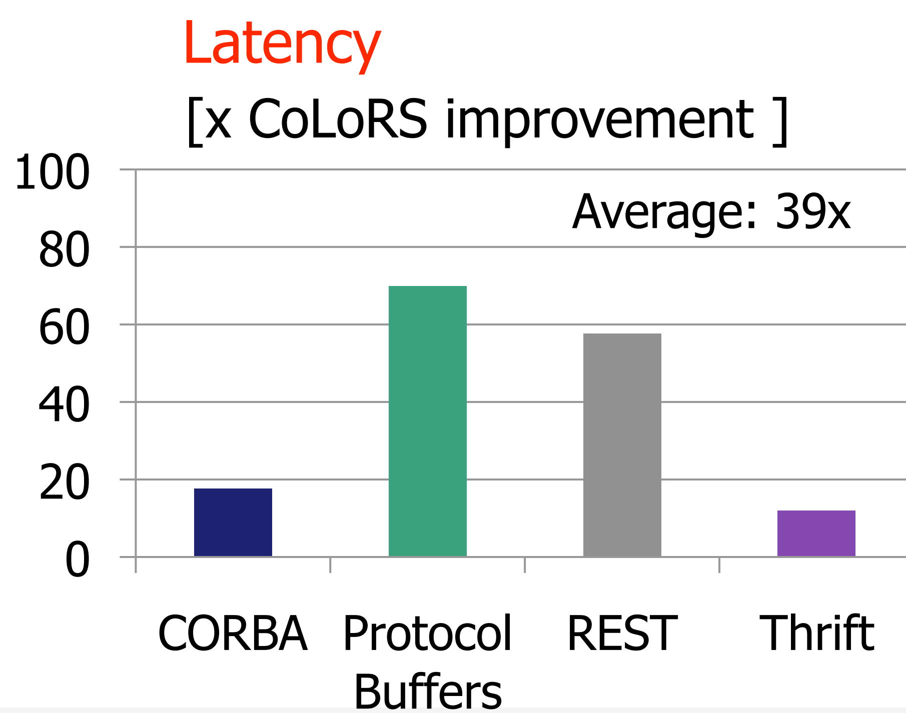
CoLoRS Contributions

- Object and memory model
  - Objects and classes shared between programs written in dynamic and static languages
  - Static-dynamic hybrid – fast yet flexible
- 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 Evaluation: Microbenchmarks

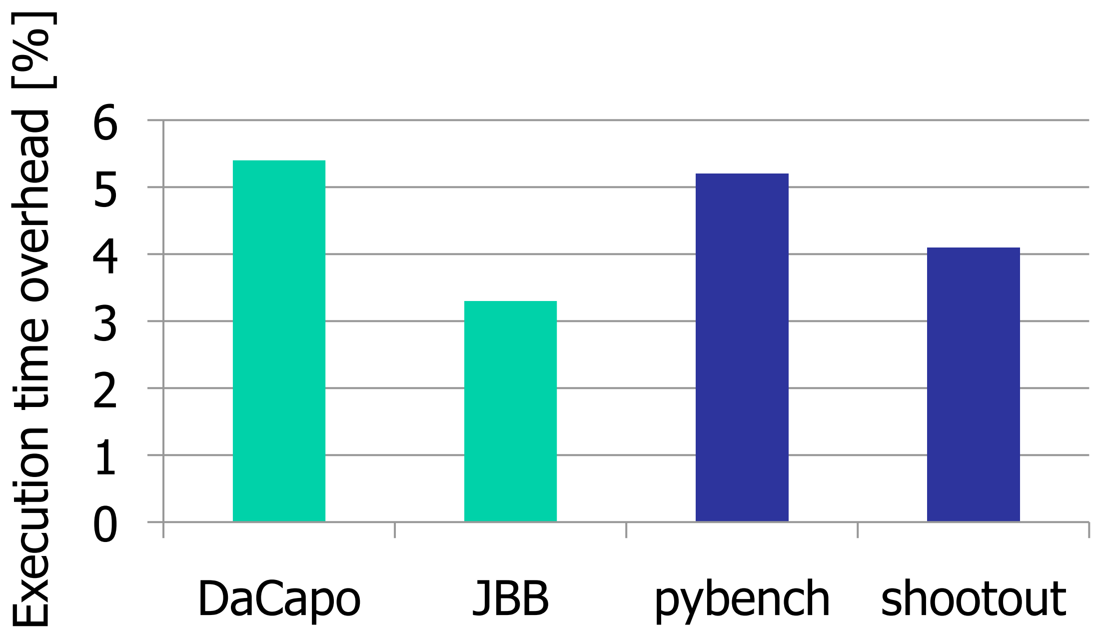
- Four cross-language RPC systems
  - Python client; Java server
  - Employ primitive and user-defined data types





CoLoRS Evaluation: Overhead

- MRE virtualization impacts performance
  - Field access, method calls, synchronization, write barriers, allocation, GC, core libraries
- CoLoRS-oblivious programs: standard Java and Python benchmarks



Overhead is below 9% and 5% on average

Summary

- 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
    - Support interfaces of popular public/proprietary clouds
    - Single platform for write-once, run-anywhere distributed apps
- 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
- Together offer potential for new research and technological advance in high-performance and scalable computing
  - Profiling, optimization, scaling, scheduling, communication, languages, development/deployment, …

UCSB

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/

UCSB