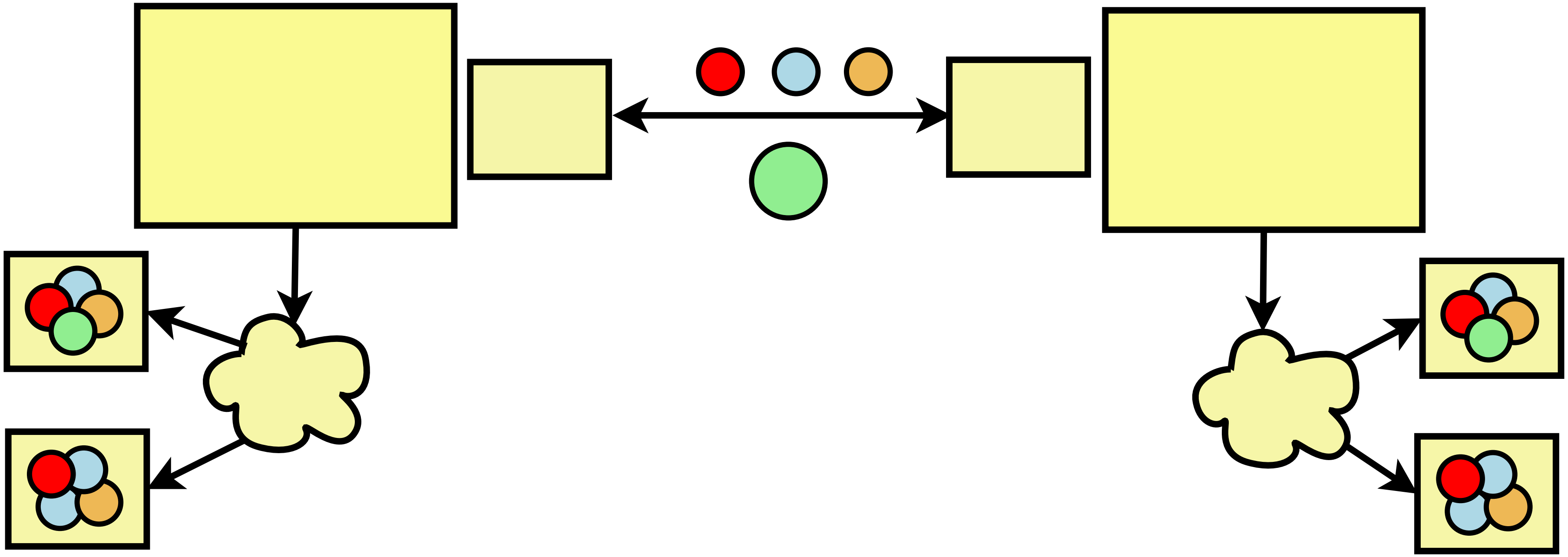
The Measured Performance of Communication and Serialization Primitives

Brian Blose, Phillip Burdette, and Gregory M. Kapfhammer
Department of Computer Science
Allegheny College

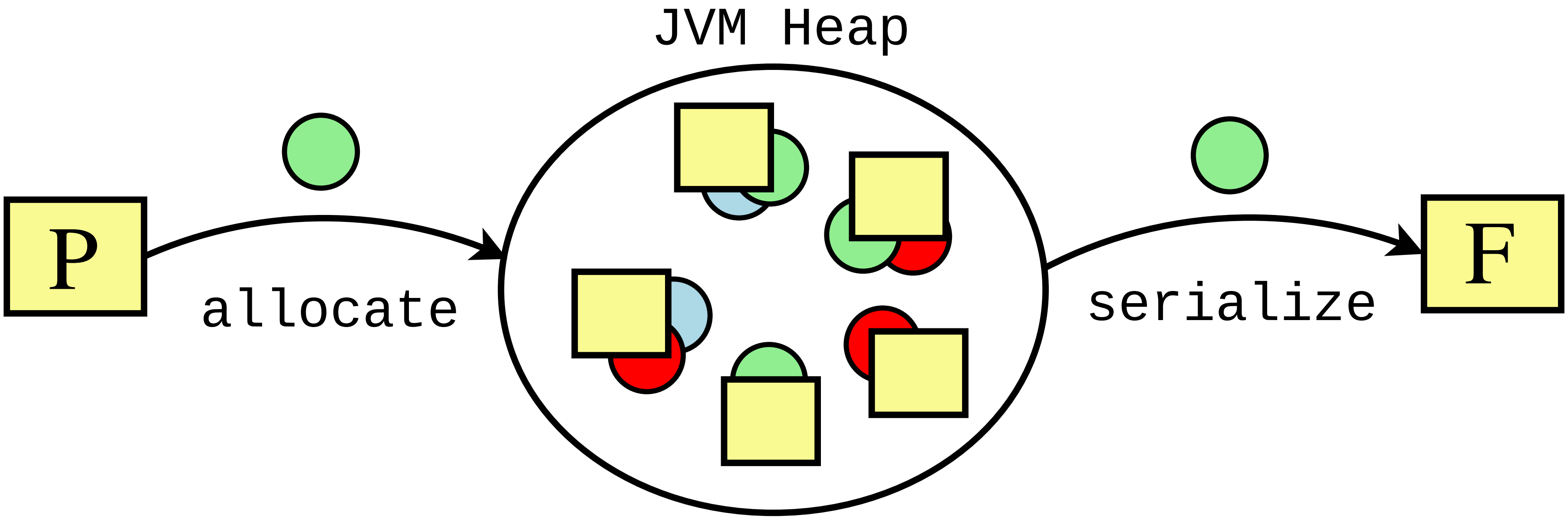
http://cs.allegheny.edu/~gkapfham/

Communication Primitives



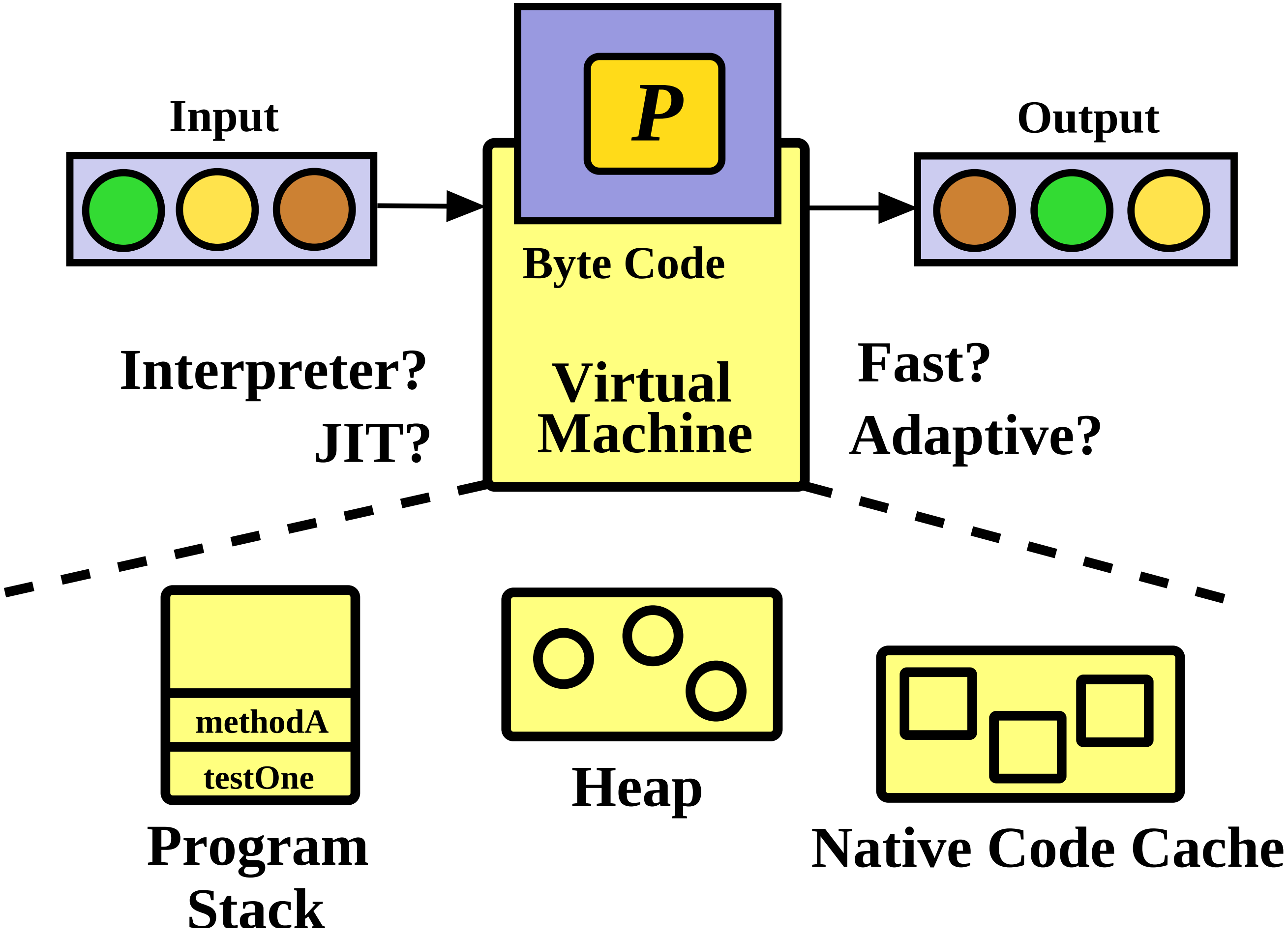
- How does object encoding impact communication?
- **Contribution**: A benchmarking framework to compare the performance of sockets and XML-RPC

Serialization Primitives



- How does object encoding impact serialization?
- **Contribution**: A benchmarking framework to compare the performance of binary and XML serialization

Program Execution with a JVM



- JVM implementation and configuration impacts performance

Experiment Design

- **Communication**: sockets and XML-RPC
- **Serialization**: XStream, JBoss, Java Serialize and Externalize
- Select Java 1.5.0, GNU/Linux with kernel 2.6.12, 3 GHz P4, 1 GB main memory, 1 MB L1 Cache, CPU hyperthreading
- Use operating system and language-based timers to calculate response time and space overheads
- Execute ten trials and calculate arithmetic means, standard deviations, and confidence intervals
- Understand internal behavior of the Java virtual machine

Micro Benchmarks

Experiment	Sent by client	Received by client
SS	Single primitive	Single primitive
SV	Single primitive	Vector
VS	Vector	Single primitive
VV	Vector	Vector

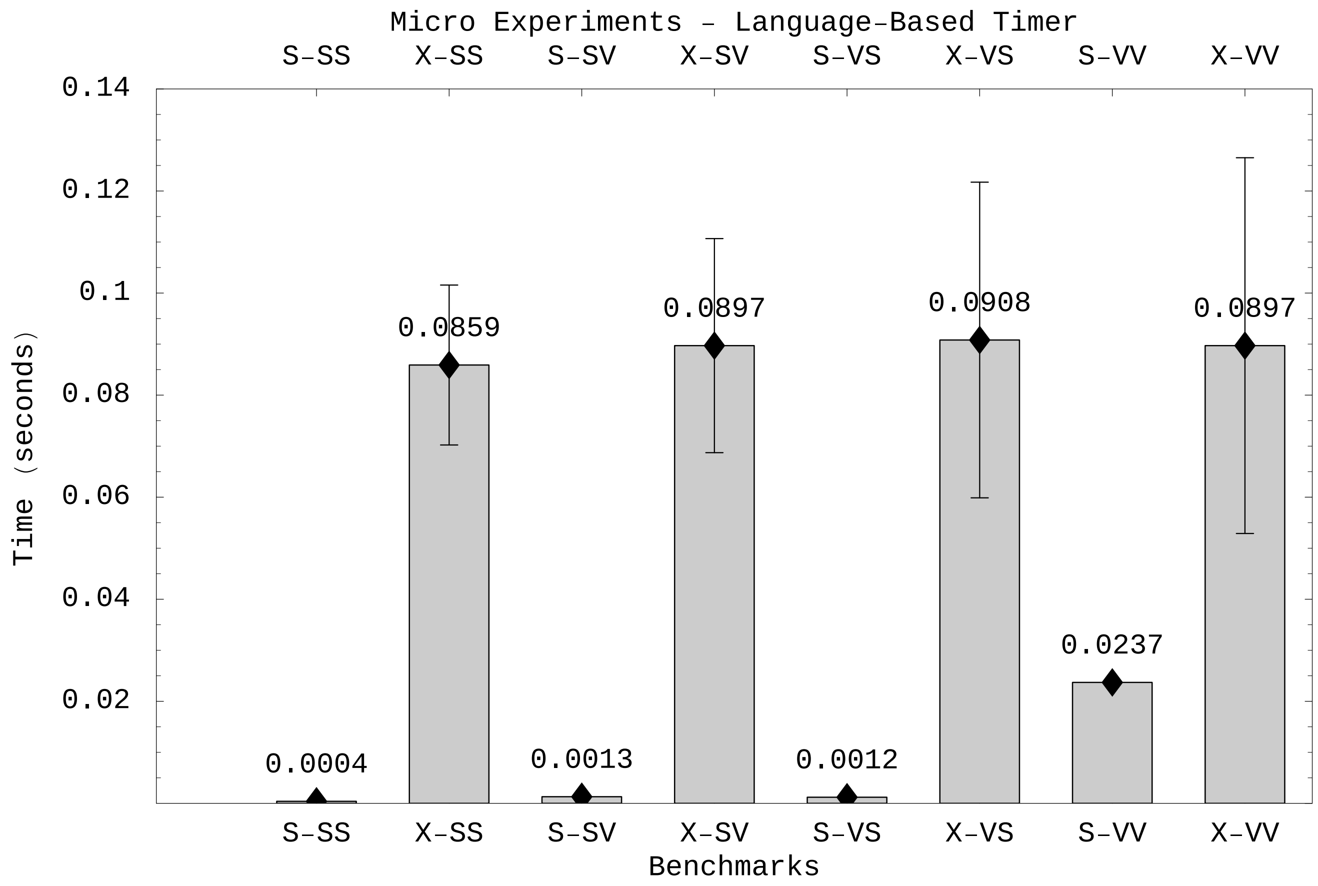
- Use benchmarks similar to those proposed by Allman et al.
- Implement the benchmarks in the Java language
- *ExperimentCampaign* framework uses Perl and Mathematica

Micro Benchmarks II

Experiment	Sent by client	Received by client
FIND (SS)	Single primitive	Single primitive
FACT (SV)	Single primitive	Vector
GCD (VS)	Vector	Single primitive
REV (VV)	Vector	Vector

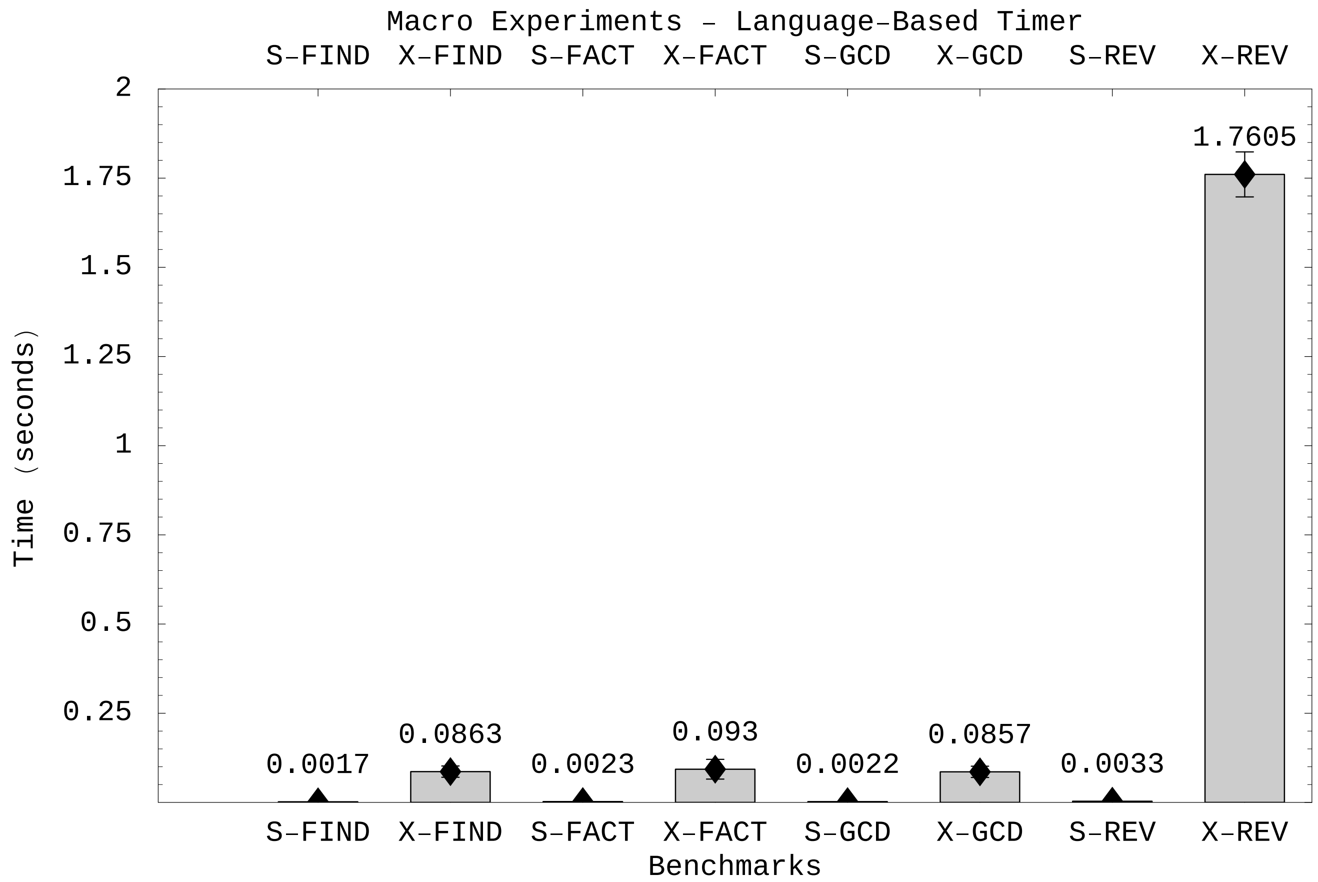
- Benchmarks use sockets and Apache XML-RPC
- Benchmarks perform a simple computation on the server
- Configure the client and server to execute on same node

Micro Benchmark I



- XML-RPC shows greater response time with more dispersion

Micro Benchmark II



- X-REV exhibits high response time due to string parsing

Using Very Large Vectors

$size(V)$	$size(V)$ (bytes)	$R(\text{VV}, S)$ (sec)	$R(\text{VV}, X)$ (sec)
5000	80,520	0.298	0.347
10000	161,000	0.598	0.523
50000	927,720	18.784	1.697

- At smaller vector sizes sockets demonstrate slightly better response times
- XML-RPC shows better response time when $size(V) = 50000$: *why?*

Explanatory Power of GC

$size(V)$	YGC (count)	YGC (sec)	FGC (count)	FGC (sec)
5000	16	.008	0	0
10000	63	.023	4	.050
50000	1645	.697	663	10.375

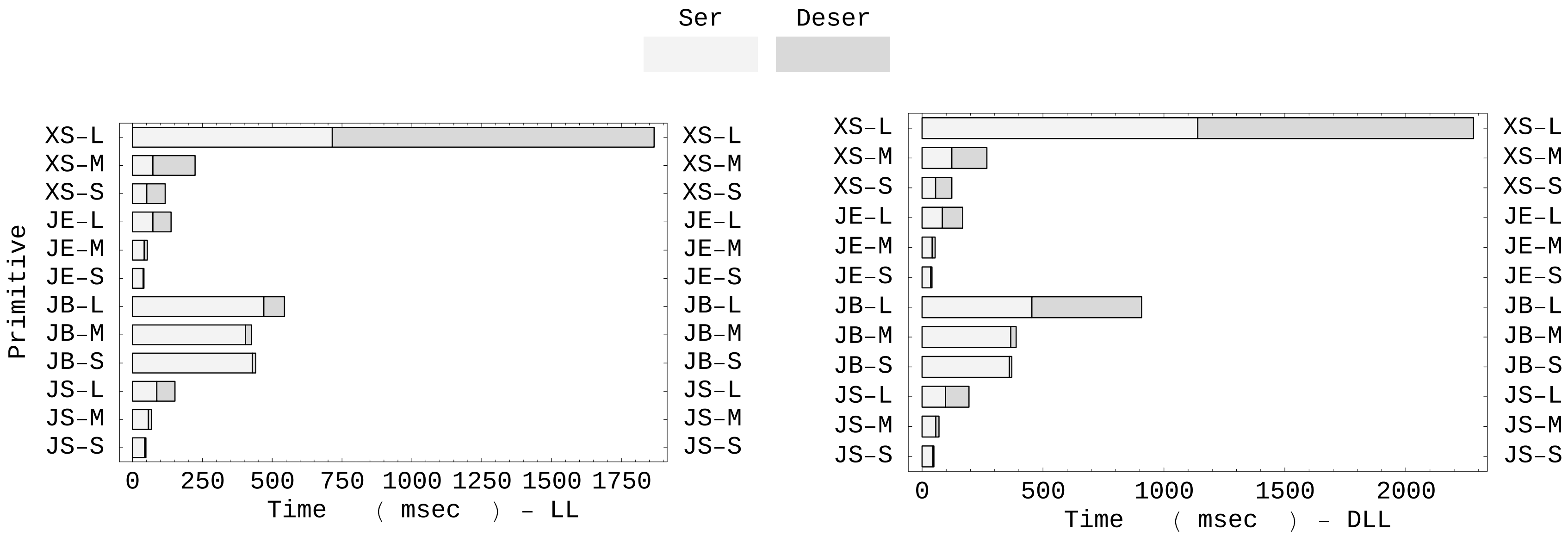
$size(V)$	YGC (count)	YGC (sec)	FGC (count)	FGC (sec)
5000	14	.016	0	0
10000	27	.022	1	.020
50000	123	.695	5	.143

- Varying the heap size of socket JVM yields similar results

GC Allocation Rate

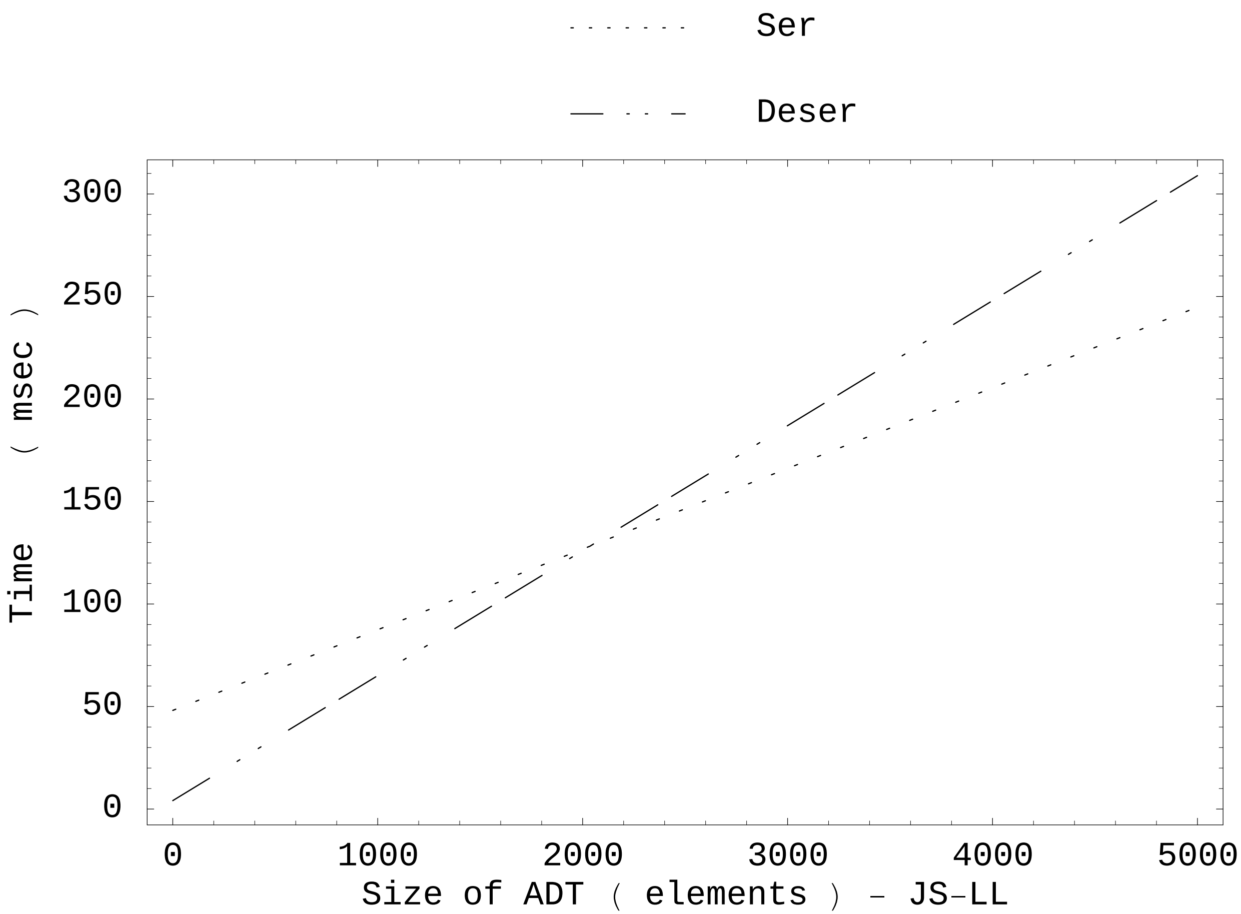
- S-VV allocates $710,374,184$ bytes and X-VV only allocates $54,101,312$ bytes
- At benchmark termination, S-VV has $4,773,224$ bytes and X-VV has $7,234,520$ bytes of live objects
- Sockets use char[] and XML-RPC uses java.nio.CharBuffer
- Can we use past GC behavior to predict future program performance?

Serialization Response Time



- Serialize and deserialize a `LinkedList`
- XS-L exhibits high response time due to parsing and validation
- JS and JE demonstrate a low response time

Serialization Trade-Offs (JS)



- JS response time varies as ADT size increases (not for XS)

Conclusions

- A suite of benchmarks to measure the performance of communication and serialization primitives
- Experiments reveal a trade-off in the performance of the two primitives
- Extend the study to new primitives and JVMs
- Focus on remote communication, long running benchmarks, and the measurement of throughput
- Consider the use of new abstract data types
- What are your suggestions?