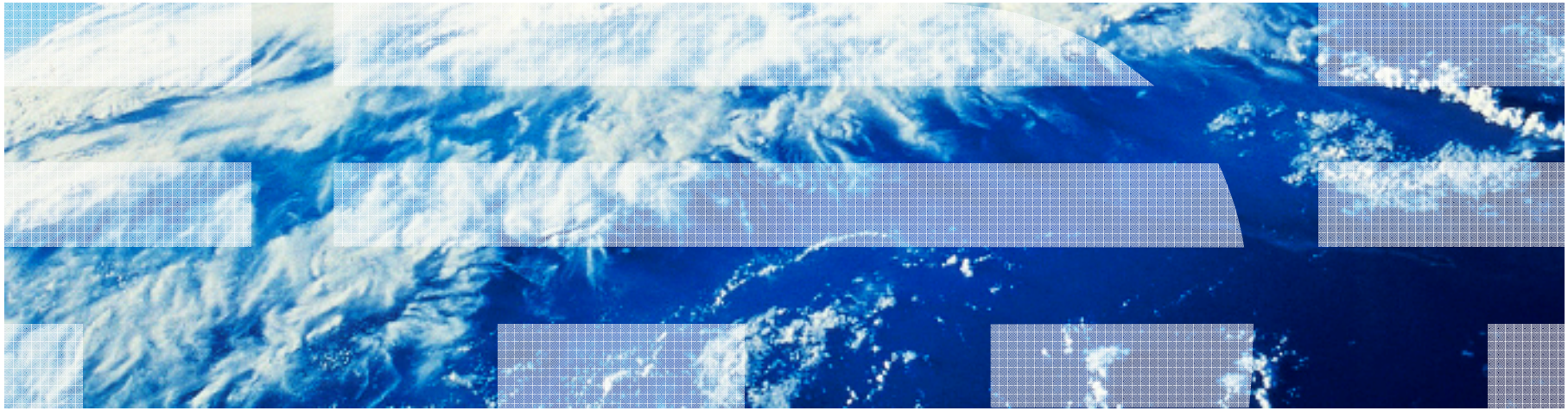
String Deduplication for Java-based Middleware in Virtualized Environments

Michihiro Horie, Kazunori Ogata, Kiyokuni Kawachiya, Tamiya Onodera

IBM Research - Tokyo



Duplicated strings found on Web Application Server

- Example of duplicated strings created by Apache DayTrader for handling 2,000 requests

Allocated objects	String value
14,483	*“META-INF/services/org.apache.xerces.xni.parser.XMLParserConfiguration”*
2,021	*“(<B><FONT color="#ff0000">-1.00%</FONT></B>)<IMG src="images/arrowdown.gif“ width="10" height="10" border="0"></IMG>”*
2,459	*“java.lang.String"*
2,126	*“false”*
2,000	*“SELECT t0.HOLDINGID, t0.ACCOUNT_ACCOUNTID, t0.PURCHASE DATE, t0.PURCHASEPRICE, t0.QUANTITY, t2.SYMBOL, t2.CHANGE1, t2.COMPANYNAME, t2.HIGH, t2.LOW, t2.OPEN1, t2.PRICE, 2.VOLUME FROM holdingejb t0 INNER JOIN accountejb t1 ON t0.ACCOUNT_ACCOUNTID = t1.ACCOUNTID LEFT OUTER JOIN quoteejb t2 ON t0.QUOTE_SYMBOL = t2.SYMBOL WHERE (t1.PROFILE_USERID = ?)”*
917	*“true”*

Duplicated strings found on Web Application Server

- Example of duplicated strings created by Apache DayTrader for handling 2,000 requests

Allocated objects	
14,483	*"META-INF/services/org.apache.xerces.xni.parser.XMLParserConfiguration"*
2,021	*"(<B><FONT color="#ff0000">-1.00%</FONT></B>)<IMG src="images/arrowdown.gif" width="10" height="10" border="0"></IMG>"*
2,459	*"java.lang.String"*
2,126	*"false"*
2,000	*"SELECT t0.HOLDINGID, t0.ACCOUNT_ACCOUNTID, t0.PURCHASE DATE, t0.PURCHASEPRICE, t0.QUANTITY, t2.SYMBOL, t2.CHANGE1, t2.COMPANYNAME, t2.HIGH, t2.LOW, t2.OPEN1, t2.PRICE, 2.VOLUME FROM holdingejb t0 INNER JOIN accountejb t1 ON t0.ACCOUNT_ACCOUNTID = t1.ACCOUNTID LEFT OUTER JOIN quoteejb t2 ON t0.QUOTE_SYMBOL = t2.SYMBOL WHERE (t1.PROFILE_USERID = ?)"*
917	*"true"*

File path to configuration files

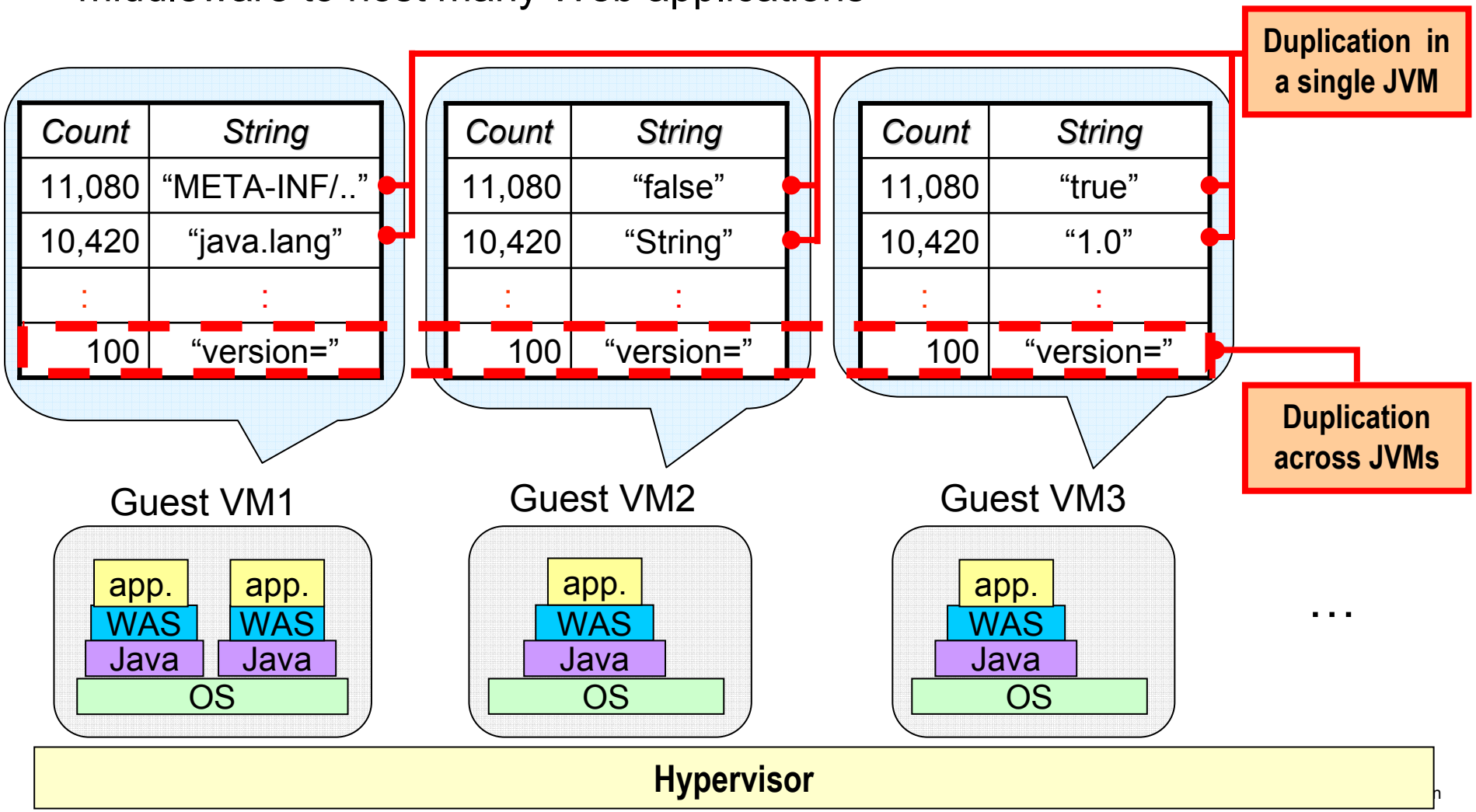
Part of HTML

Tokens after parsing configuration files

SQL statement

String duplications in a single JVM and across JVMs

- A server in a PaaS datacenter often runs multiple copies of the same middleware to host many Web applications



To increase VMs runnable in one physical machine

- Datacenters want to increase the number of guest VMs.
  - For example, we had a real customer who wanted to run more than 100 guest VMs.

- To reduce memory footprint, we focus on reducing string data in Java workloads running on guest VMs
  - In a Java workload, `String` and `char[]` occupy more than 20% of live Java heap

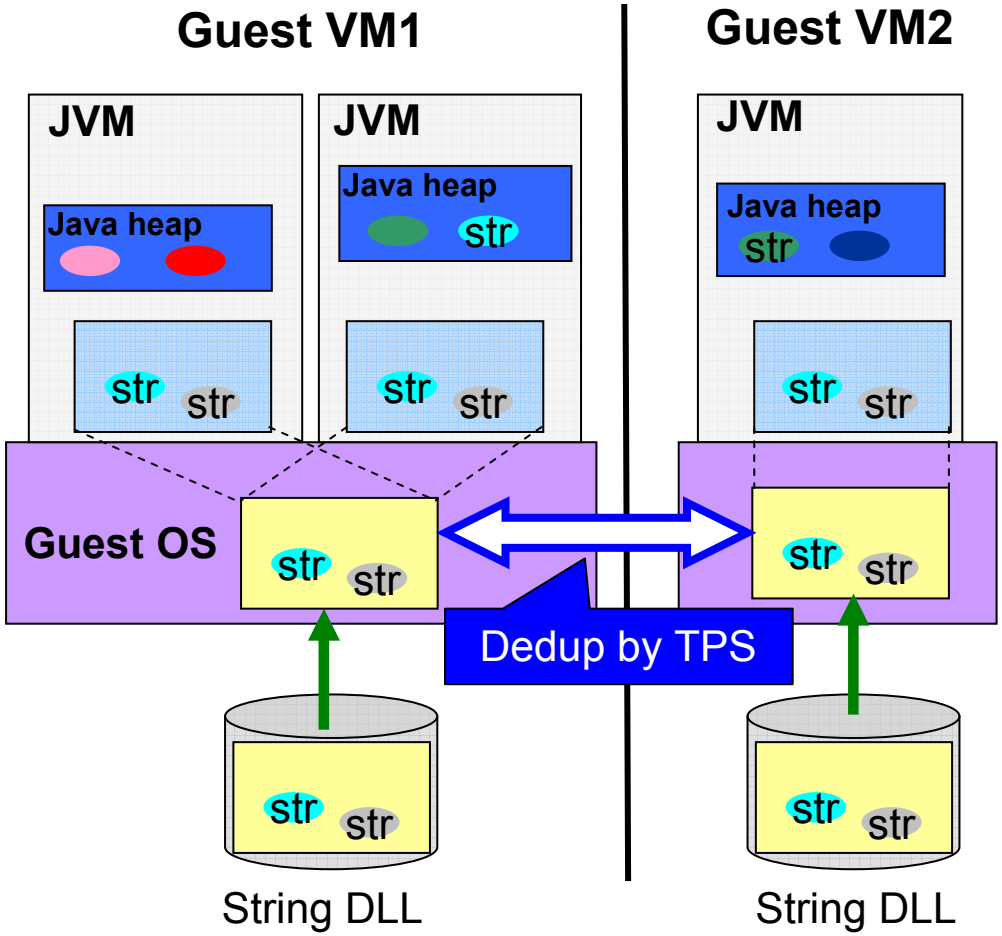
Two approaches: String deduplication in a single JVM and across JVMs

(Ⅰ) Deduplication in a single JVM

- Selectively intern duplicated `String` and `char[]` objects in a single JVM
- For short-lived objects

(Ⅱ) Deduplication across JVMs

- Preload strings using DLL
- Share strings across guest VMs using hypervisor's page sharing feature such as TPS (Transparent Page Sharing)
- For long-lived objects



Deduplication in a single JVM

- Our approach: selective intern by checking calling contexts
  - Step-1: Offline profiling
    1. Recording calling contexts and created strings
    2. Finding calling contexts each of which allocates a single heavily duplicated string
  - Step-2: Selective intern
    - Applications are modified to check calling context with small overhead and intern allocated string if applicable
    - Without selective intern, performance degradation occurs
      - Our experiment on a Web application showed more than 10% of degradation when all strings are interned

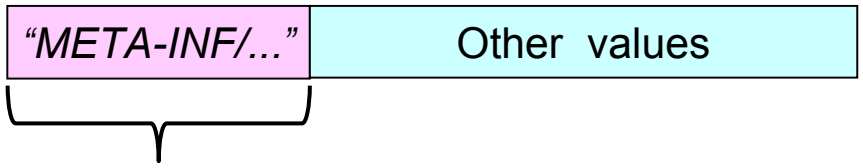
Find calling contexts allocating heavily duplicated strings

- Example: the most heavily duplicated string in the table in page 2

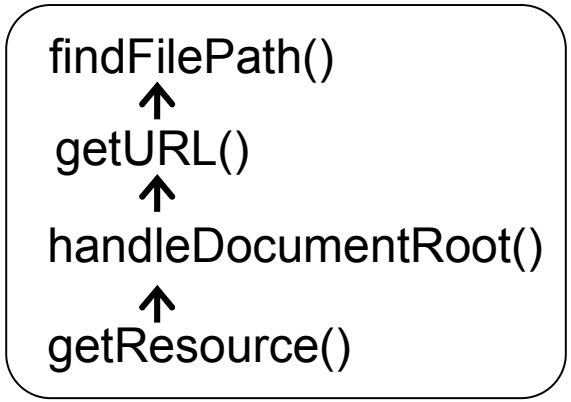
Allocated objects	String value
14,483	*"META-INF/services/org.apache.xerces.xni.parser.XMLParserConfiguration"*

- We found some calling contexts always create the same values of strings
- For example, this calling context created more than 10,000 duplication of the value above

String objects allocated by findFilePath()



Created 10,000+ duplication in context:



Low-overhead Calling Context Checking

- Only check the entry and the exit of a calling context
  - Selectively invoke toUnifiedString() at runtime
  - Uses an Aspect-Oriented Programming(AOP) language to change code at post-compile time

Original code

First method of profiled calling context

```
void getResources() {
   handleDocumentRoot();
}
```

The method to call intern

```
String findFilePath() {
   // do some work
}
```

Modified code

```
void getResources() {
   // registers thread ID
   handleDocumentRoot();
}
```

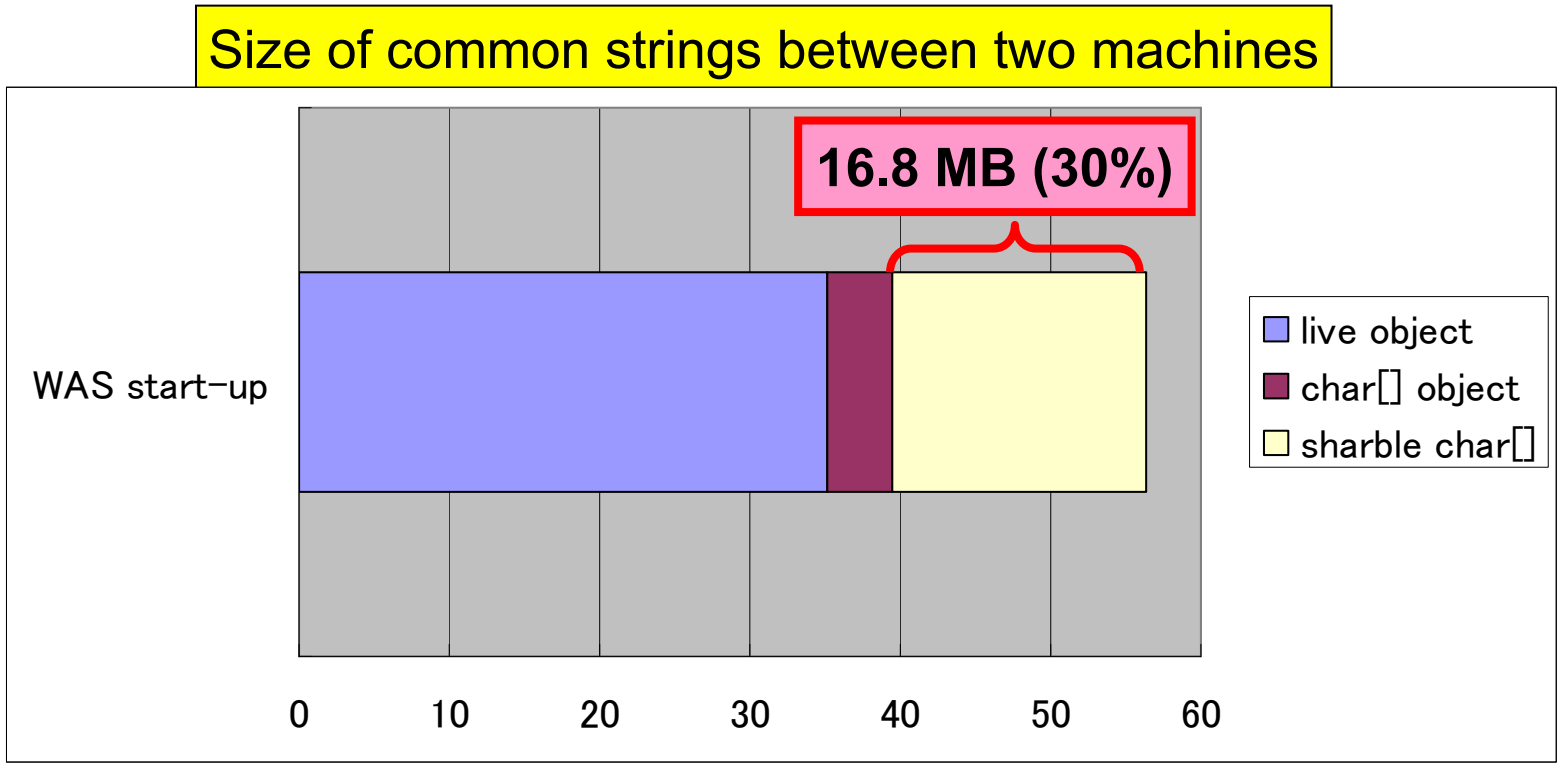
```
String findFilePath() {
   if (/* same thread */)
      return toUnifiedString(...);
   else
      // do original work
}
```

Deduplication across JVMs

- Our approach: sharing common strings by using a DLL
  - Step-1: Offline profiling
    - Creating heap dumps on multiple machines by running the same middleware and similar web applications
    - Collecting common strings among the dumps for:
      - Extracting strings whose values are specified by middleware or web applications
      - Excluding machine-specific strings, such as IP addresses and host names
  - Step-2: Sharing strings across JVMs by using a DLL
    - Store `String` and `char[]` to the DLL in the same format as those objects in the Java heap
    - Each JVM loads the DLL at start-up time

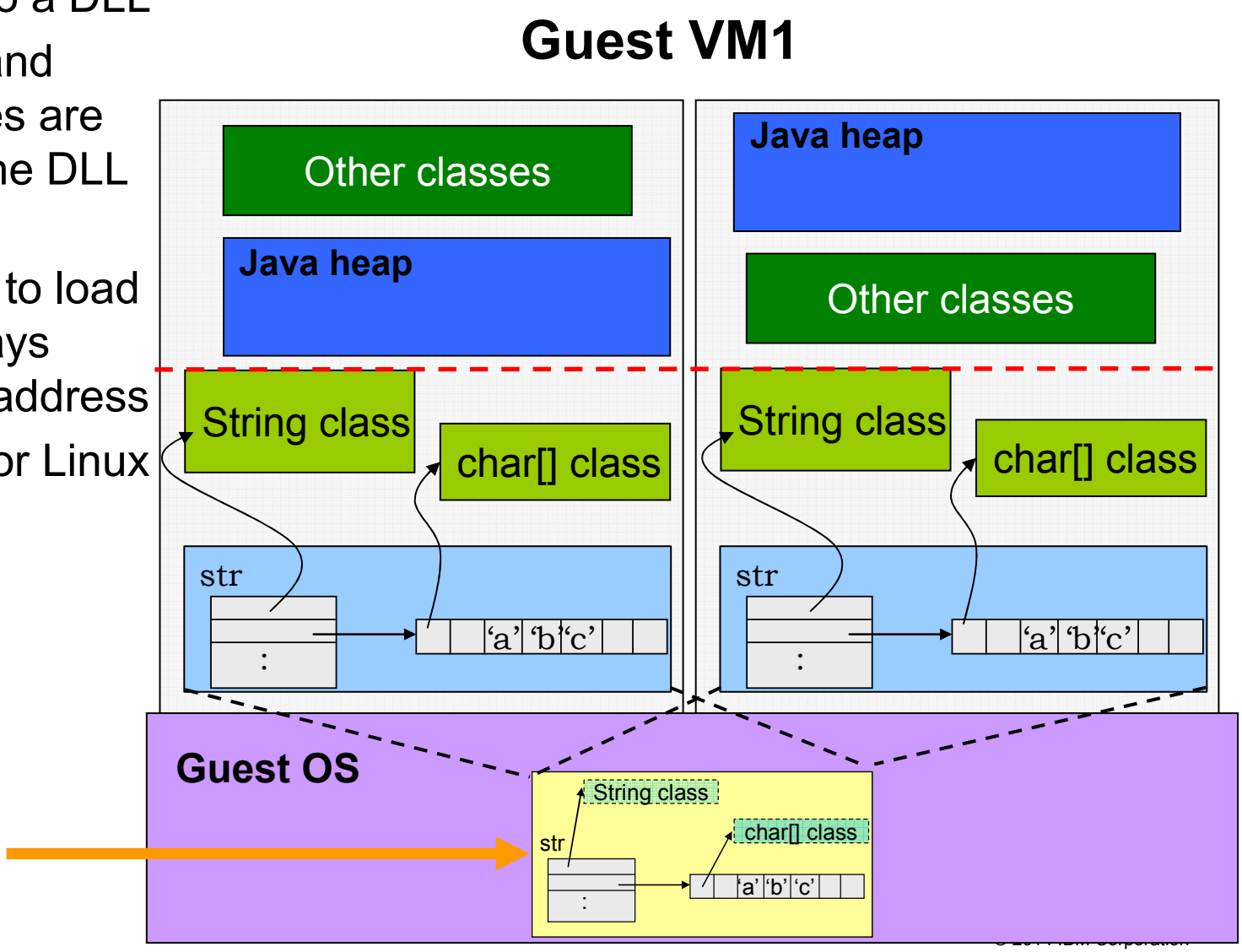
Common strings among JVMs

- In the heap dumps after running Web application servers on two different machines, we found 30% common strings
  - They are long-lived objects



Deduplication across JVMs in a single guest VM

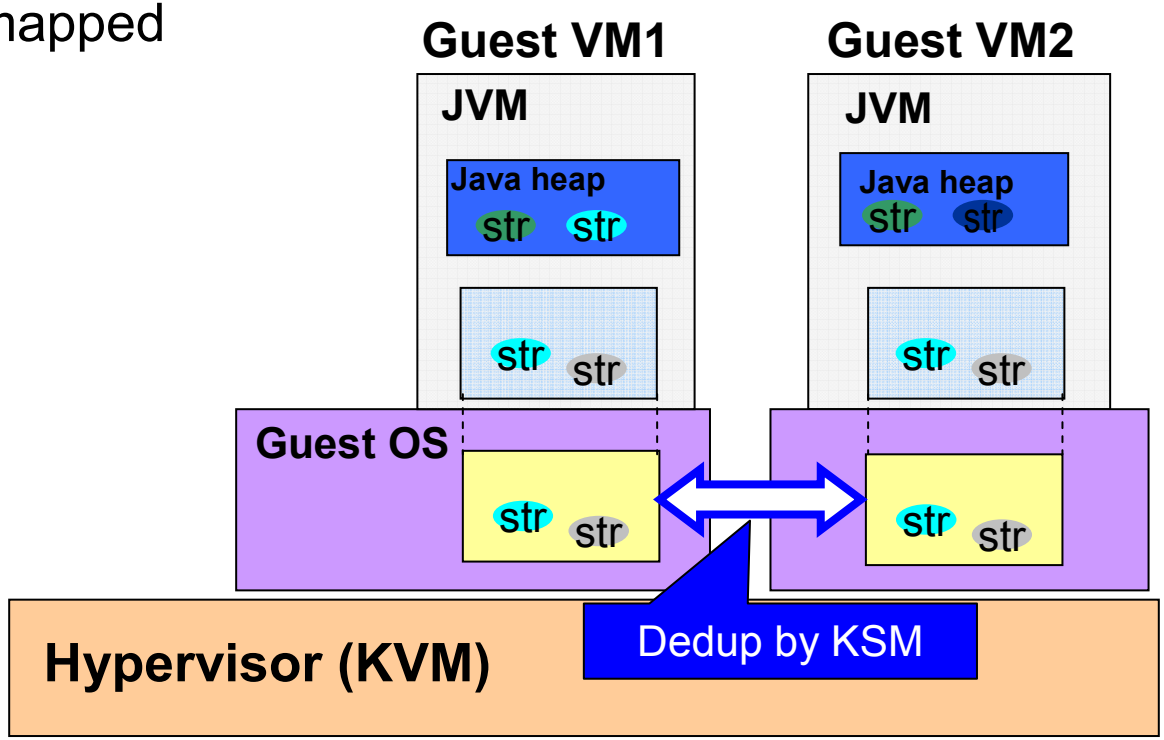
- Store strings into a DLL
- Class pointers and object references are decided when the DLL is created
  - We modified to load the DLL always at the same address
  - *(cf.) prelink* for Linux



Deduplication even across guest VMs

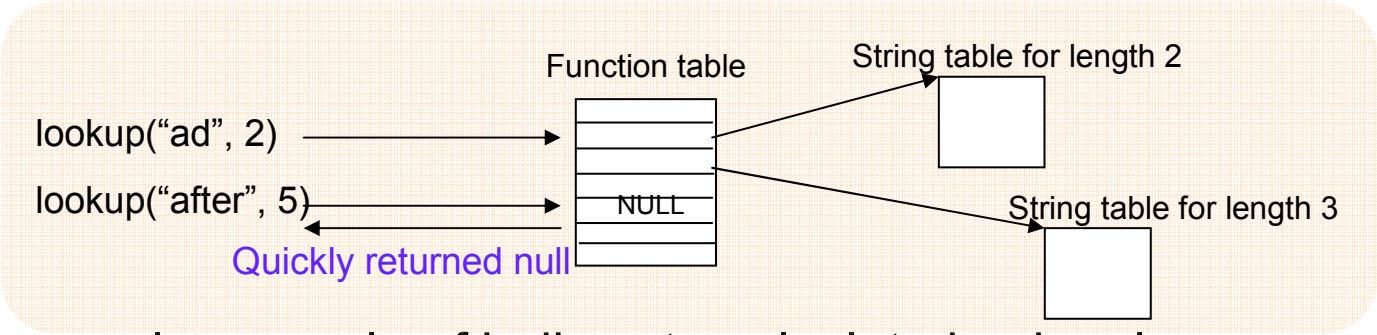
- We make use of the feature of KSM (Kernel Samepage Merging) on KVM (Kernel-based Virtual Machine)

- KSM merges mapped memory into a single page
  1. KSM detects that mapped DLLs in guest VMs are the same
  2. KSM deduplicates the mapped DLLs into one

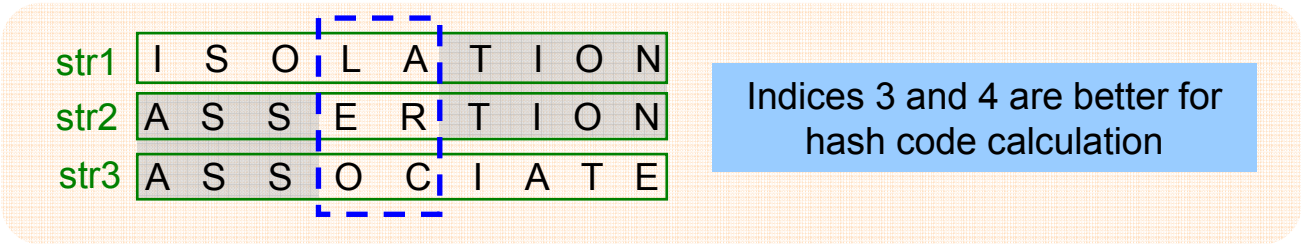


Fast lookup mechanism

1. Separate into small groups of strings from the viewpoint of the length



2. Pick up only a couple of indices to calculate hash values



3. Choose one hash function that yields the least collision of hash value

```
I_32 hash = ch1 * ch2;
return  (hash * hash);
```

```
I_32 hash = ch1 * ch2;
return  (hash * hash);
```

```
I_32 hash = ch1 * ch2 * ch3;
return  (hash * hash);
```

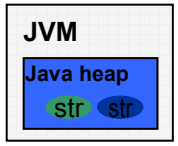
Experiment environment

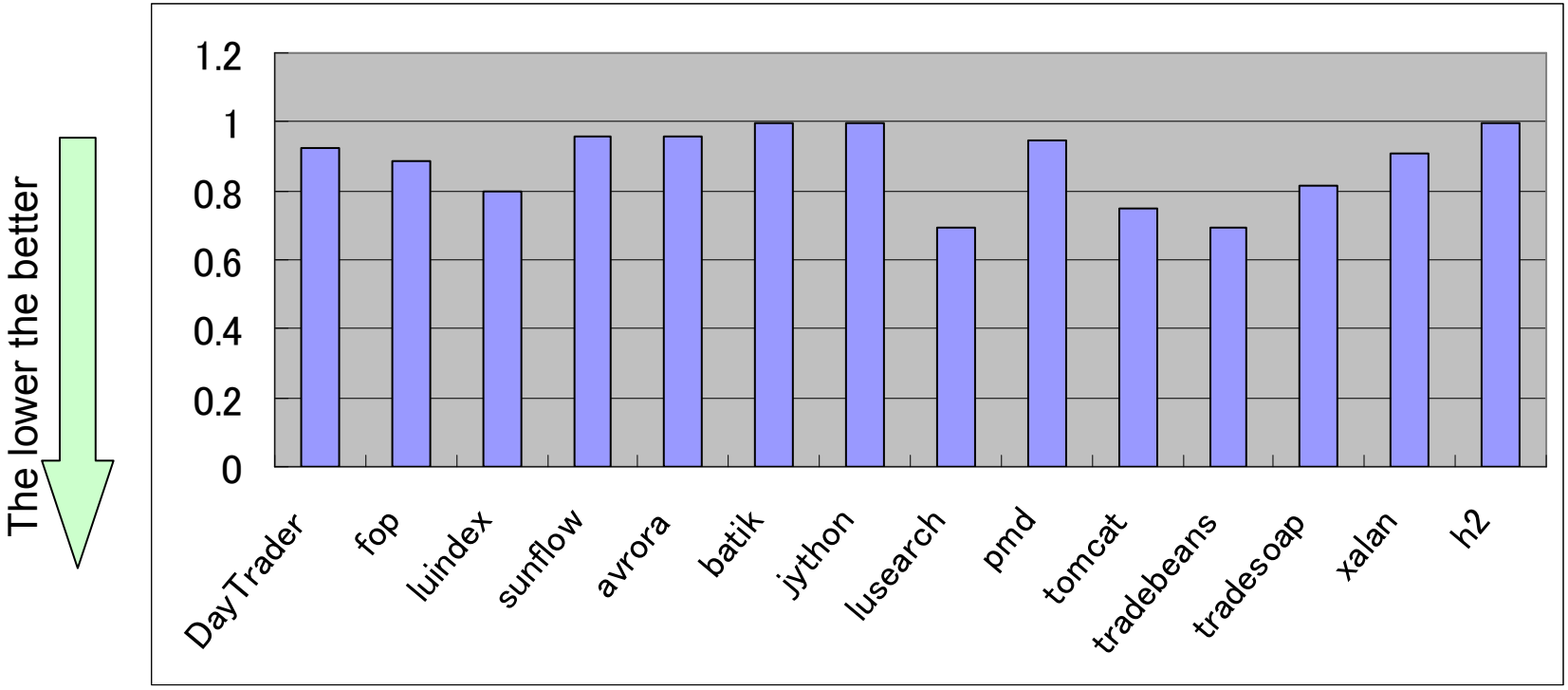
Physical machine environment	
Machine	IBM BladeCenter LS21
CPU	Dual-core Opteron (2.6 GHz), 2 sockets
RAM size	4.00 GB
Host OS	RedHat Enterprise Linux 6.1 (2.6.32-131.0.15.el6.x86_64.debug)
Hypervisor	KVM (kvm-83-224.el5)
Number of Guest VMs	3 VMs
Guest VM environment	
Virtual CPU	2 virtual CPUs
Guest memory size	1.50 GB
Guest OS	RedHat Enterprise Linux 6.2 (2.6.32-220.el6.x86_64.debug)
WAS version	8.0.0.0
JVM	IBM Java J9 VM for Java 6, 64bit
GC policy	Generational GC
Maximum and minimum sizes of Java heap	1.25 GB
Size for the tenured space	1.00 GB
Size for the nursery space	0.25 GB

Reduced memory allocation by up to 30%

- On average, 12% reduction in the total allocation size of objects

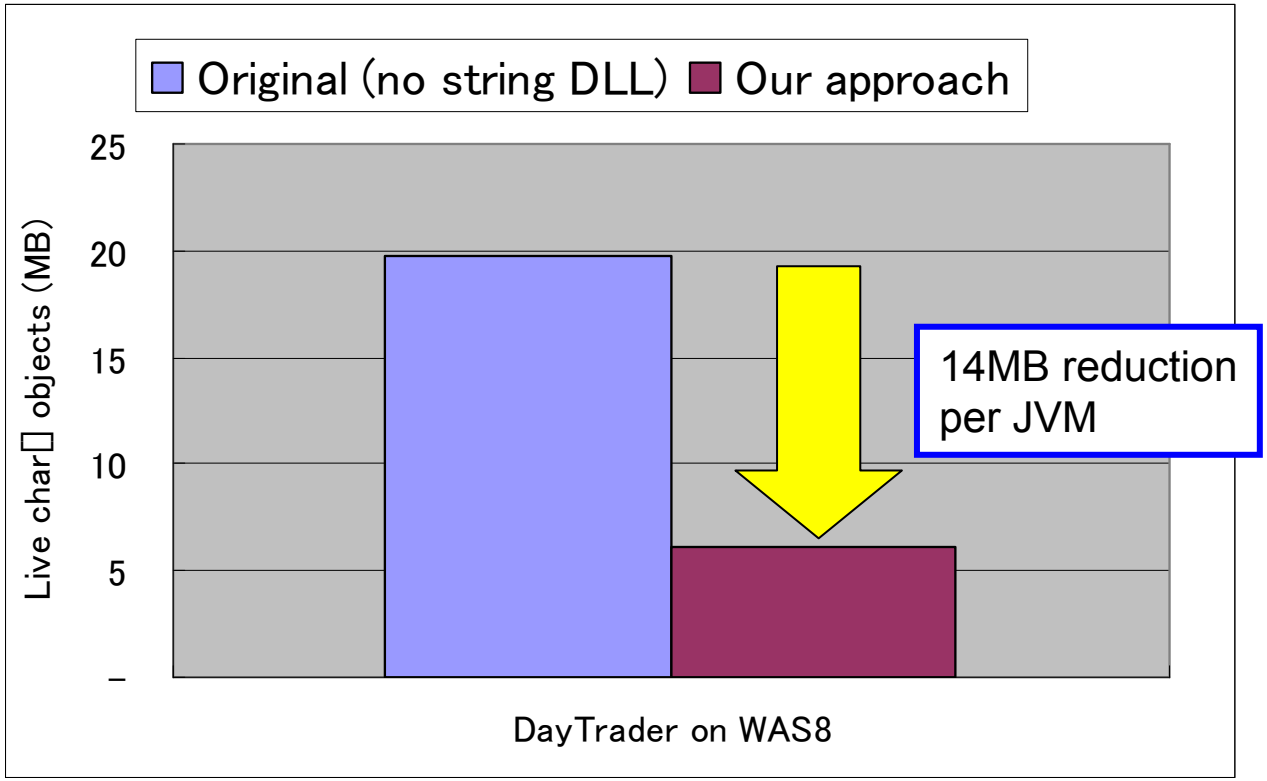
Single JVM



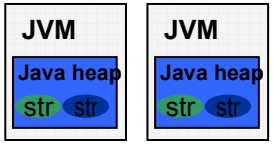


Reduced 70% of live char[] objects

- Total amount of memory reduces proportionally to the number of guest VMs
  - For example, we can reduce 1.4GB with 100 virtual machines

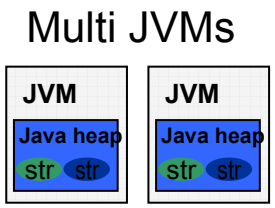


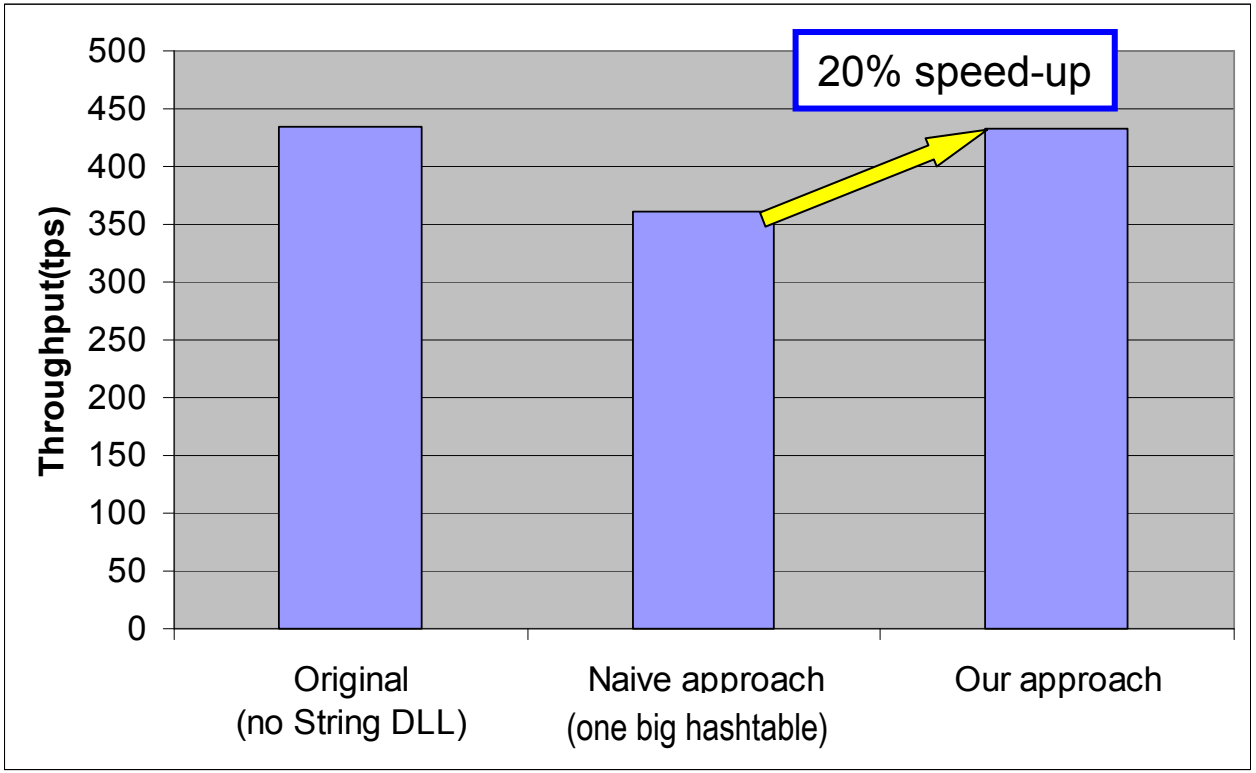
Multi JVMs

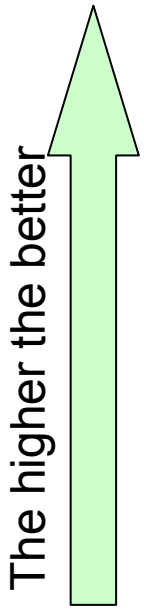


Low overhead for looking up the string DLL

- We need to look up a string in the DLL when a String constructor is invoked
- No performance degradation compared to original
  - 20% speed-up compared to a naive approach







Concluding remarks

- Found that many duplicated strings existed
  - In a single JVM and across JVMs
- Proposed two approaches for string deduplication
  - Selective interning
  - Preload common string objects using DLL
- Reduced amount of char[] objects by:
  - 12% at runtime of web applications
  - 70% in live char[] objects