

*WWC5*

*Austin, TX. Nov. 2002*

# **PennBench: A Benchmark Suite for Embedded Java**

G. Chen, M. Kandemir, N. Vijaykrishnan,  
And M. J. Irwin  
Penn State University

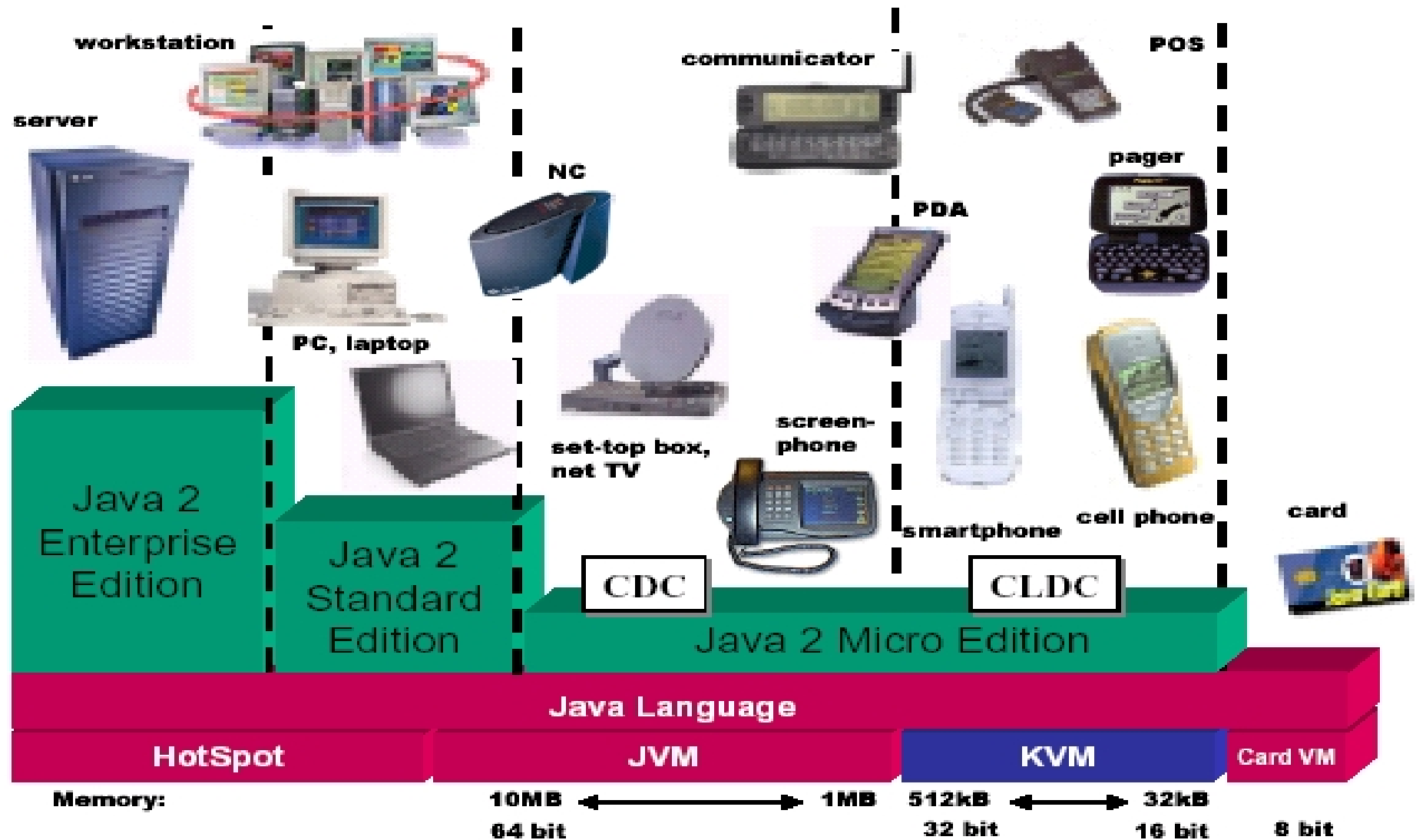
***<http://www.cse.psu.edu/~mdl>***

# Outline

- Introduction to Embedded Java
- Introduction to PennBench Suite
- Characterizations
  - Method invocation behavior
  - Memory allocation behavior
- Conclusions

# Introduction to Embedded Java

# J2ME - Java for Embedded Devices

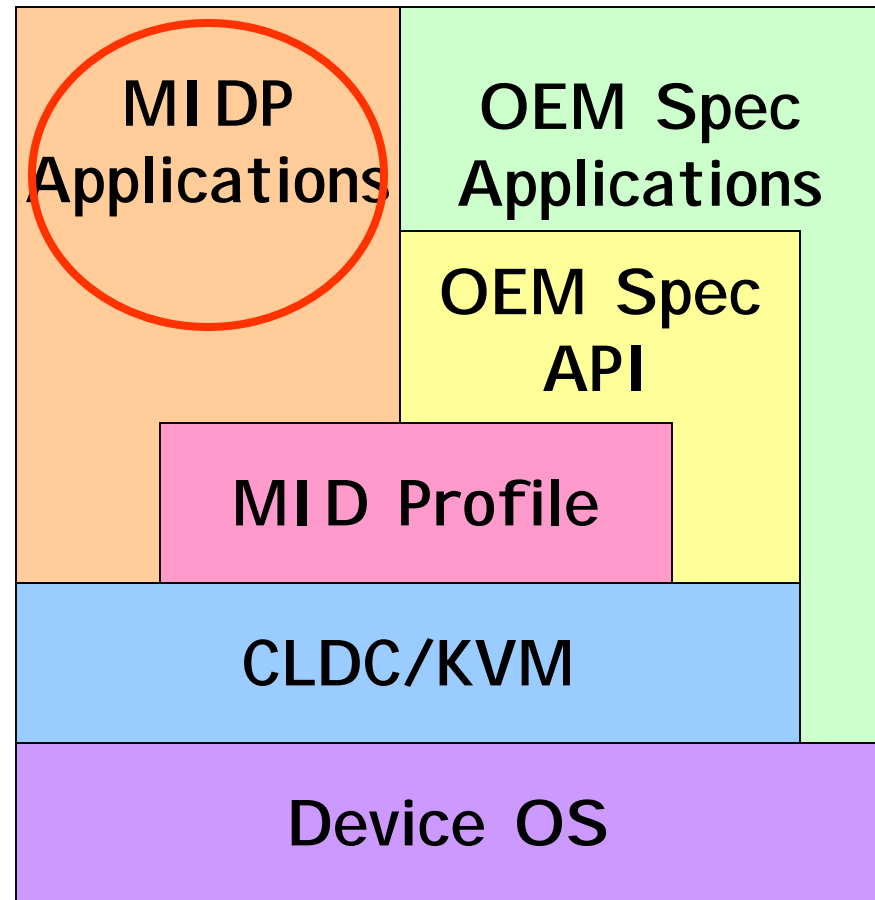


From SUN Microsystem

# Why Java?

1. Dynamic delivery of applications and services
2. Cross-platform compatibility
3. Enhanced user experience
4. Disconnected access
5. Security
  - *"We can allow third-party application developers who we don't know to put applications in the telephone because we know that Java™ technology will protect the phone."*  
-- Brian Modra, software architect, Nokia

# Architecture of a Java Enabled Device



# MIDP (Mobile Information Device Profile) & CLDC (Connected Limited Device Configuration)

- **Standardized Java APIs**
  - Developed utilizing Java Community Process (JCP) Program
  - Supported by major mobile phone manufactures
- **Targeting at resource constrained devices**
  - Limited memory: usually no more than 1MB
  - Limited CPU speed
  - Limited energy supply: powered by batteries
  - Ex. mobile phones, entry level PDAs

# PennBench Suite

# Why not SPEC JVM98 Benchmarks?

	MIDP Devices	SPEC JVM98
Heap Memory	Typically $\leq 1\text{MB}$	<ul style="list-style-type: none"><li>• 10MB to run,</li><li>• 32MB+ for reasonable performance</li></ul>
Computation	Usually Light	Heavy
User Interactivity	Typically	No
Networking	Yes	No

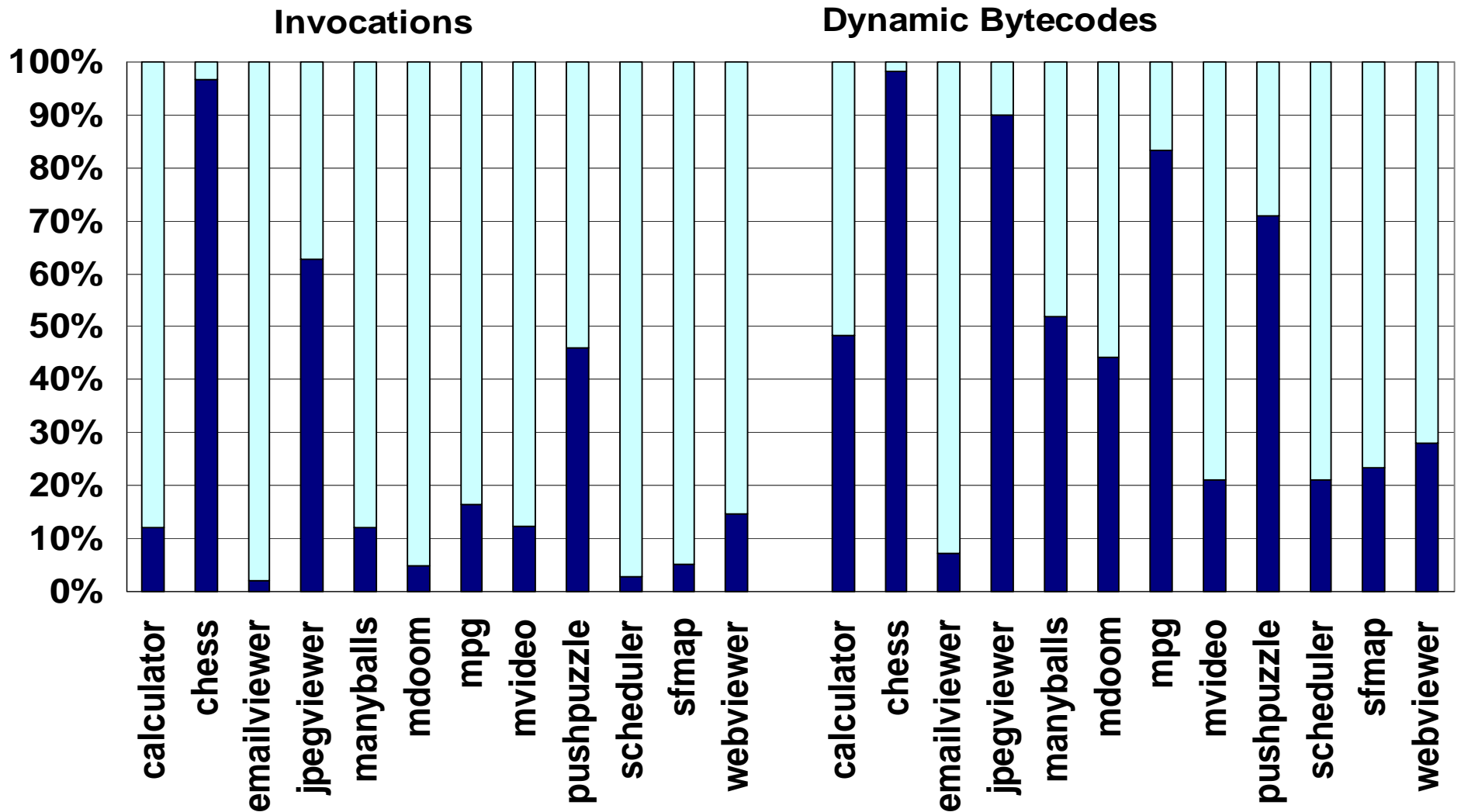
# Guidelines for benchmark selection

- Be able to run on MIDP/CLDC devices
- Include both utility and entertainment applications
- Include different types of applications
  - CPU intensive, Memory intensive, and GUI intensive
- Touch all the major components of a MIDP/CLDC device
- Be useful or interesting
- Publicly available

# Method Invocation Behavior of PennBench Suite

# Method Invocations Breakdown

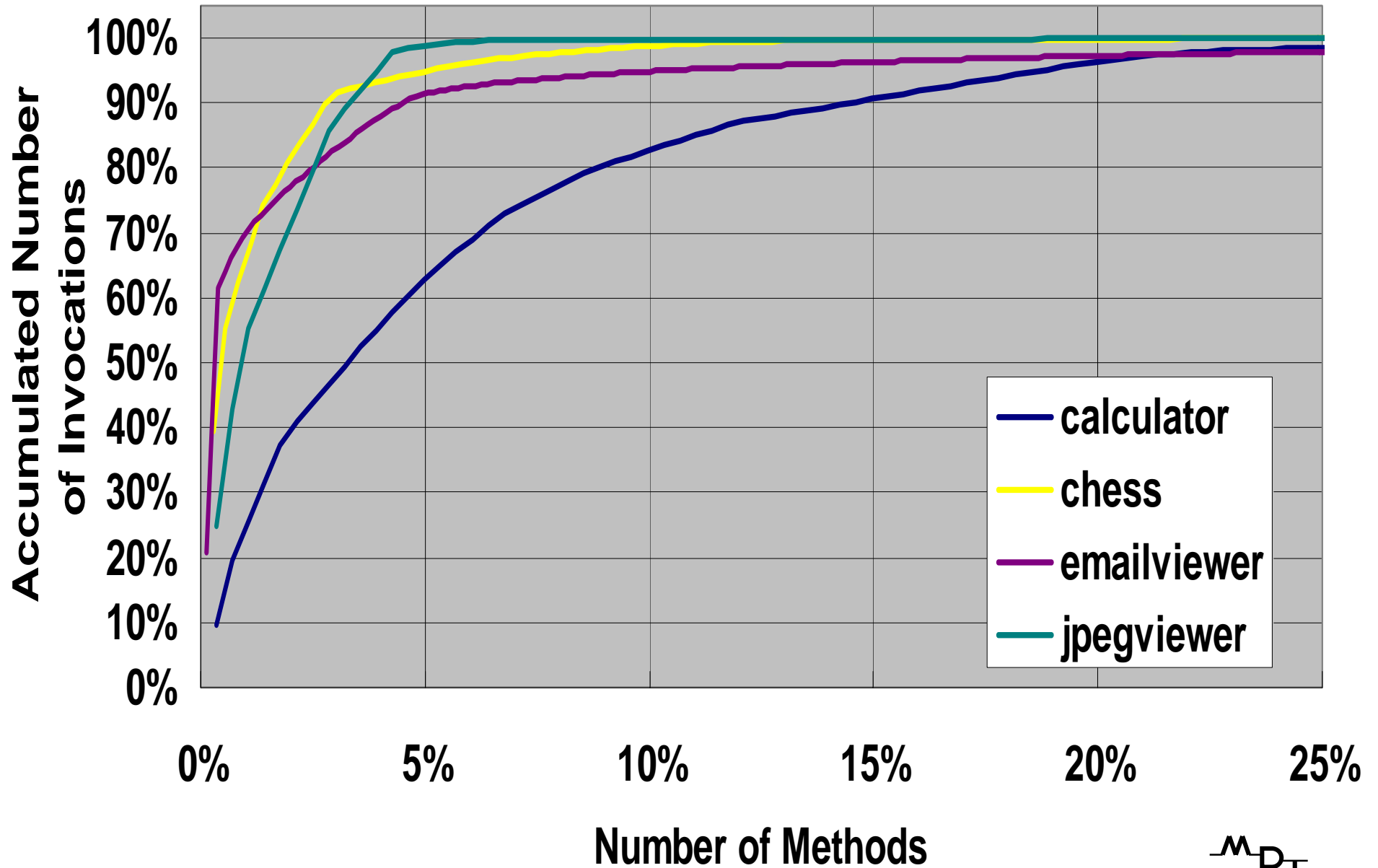
■ User □ Library



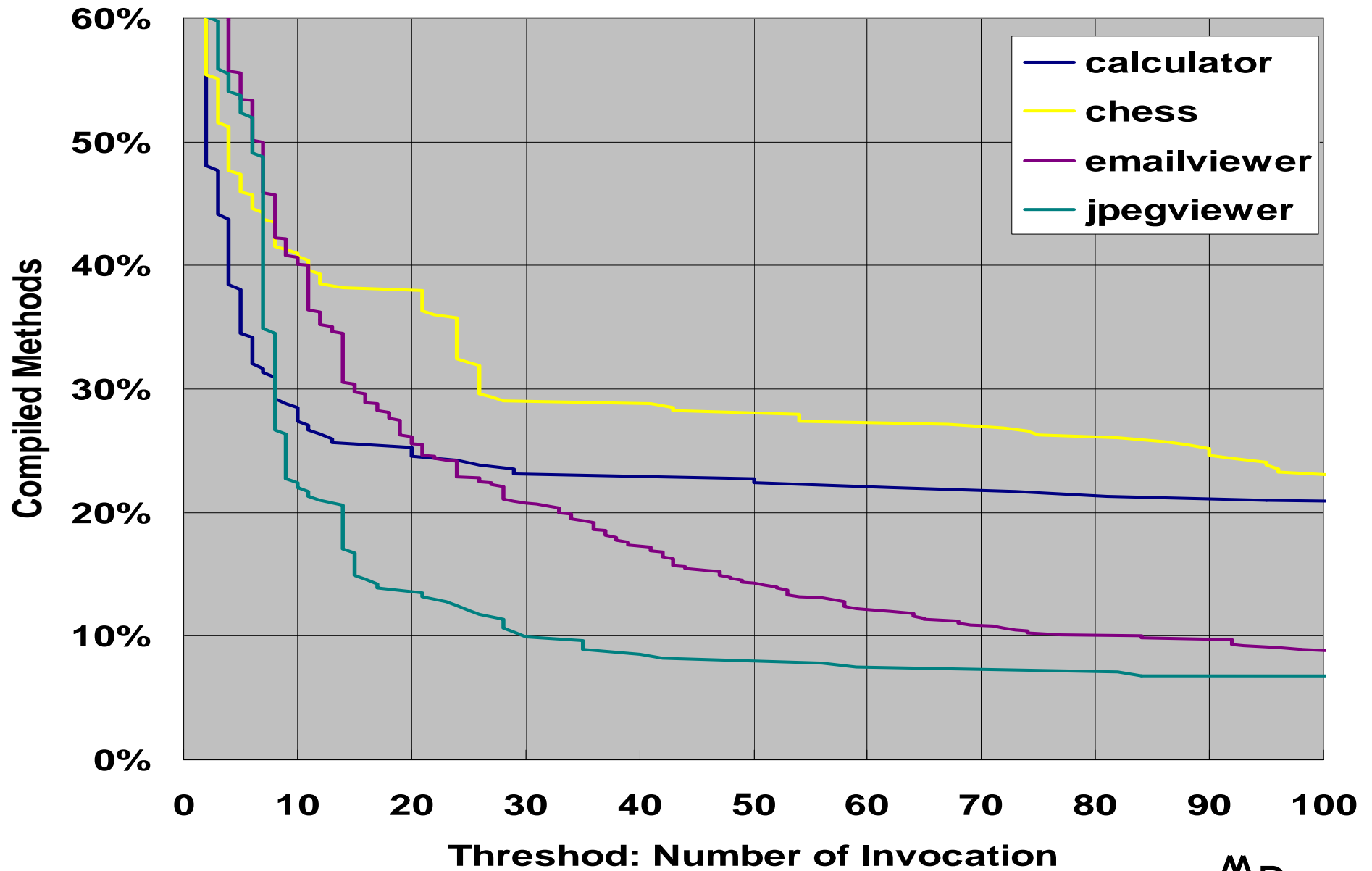
# Just-In-Time Compilation and Hot Methods

- Bytecode interpreter is slow
- JIT - dynamically compile the bytecode into native code at runtime
- JIT has heavy overheads
  - The compilation cost is amortized in the future invocations of the compiled methods
  - Only the hot (frequently used) methods can pay off the compilation cost
  - How many methods are hot?
  - How to identify hot methods?

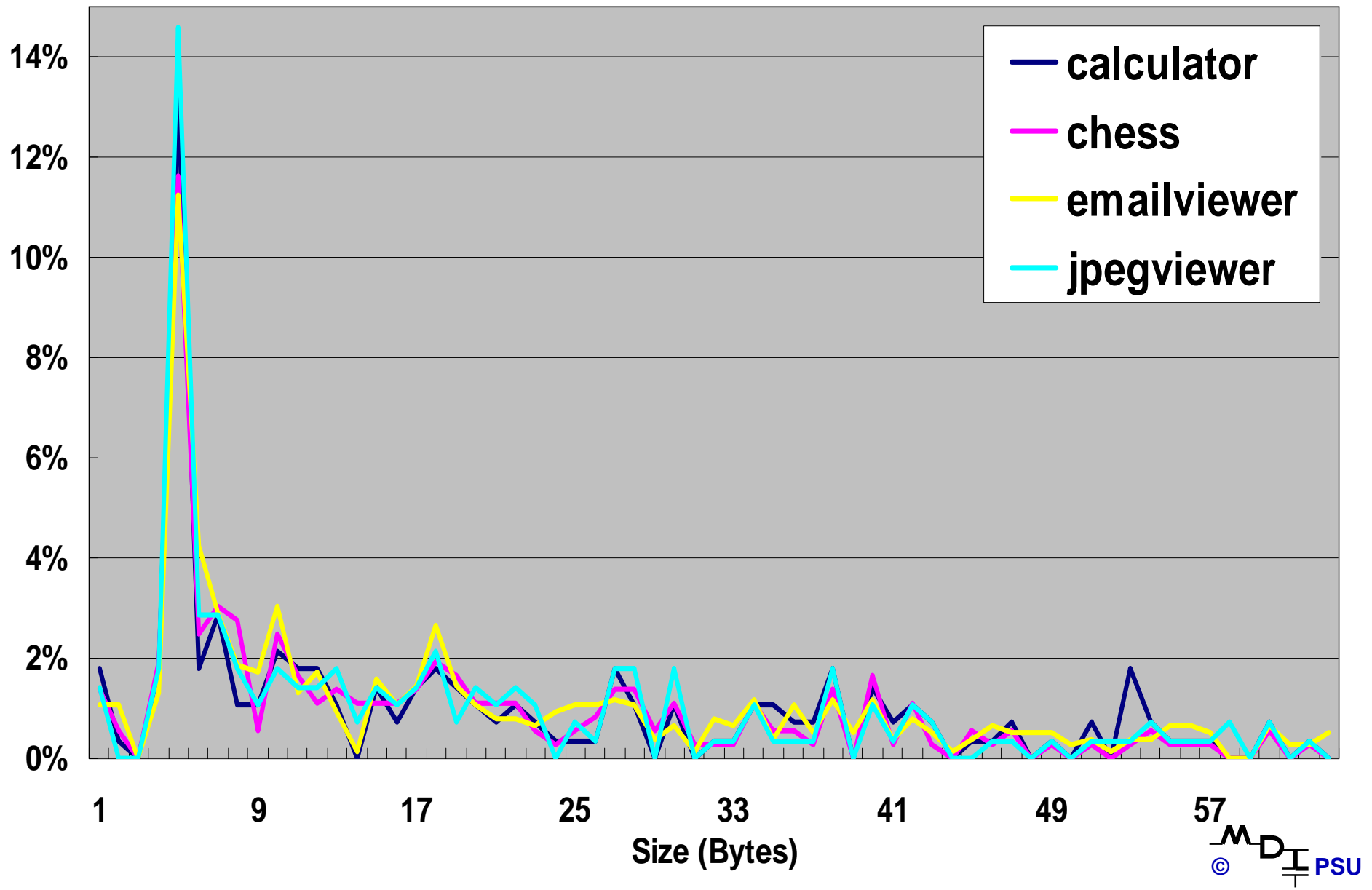
# Hot methods



# Threshold for Just-In-Time Compilation

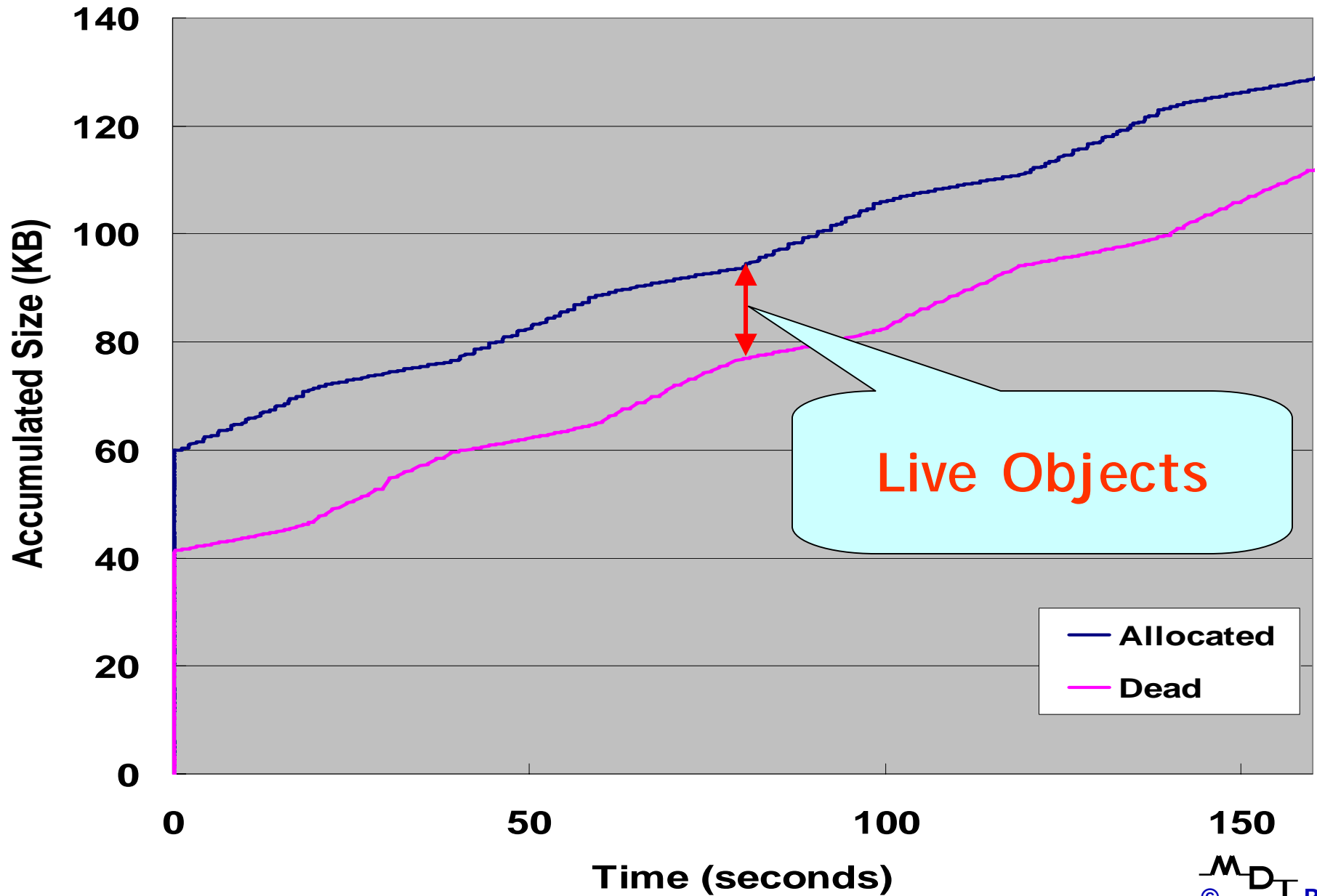


# Distribution of Method Sizes

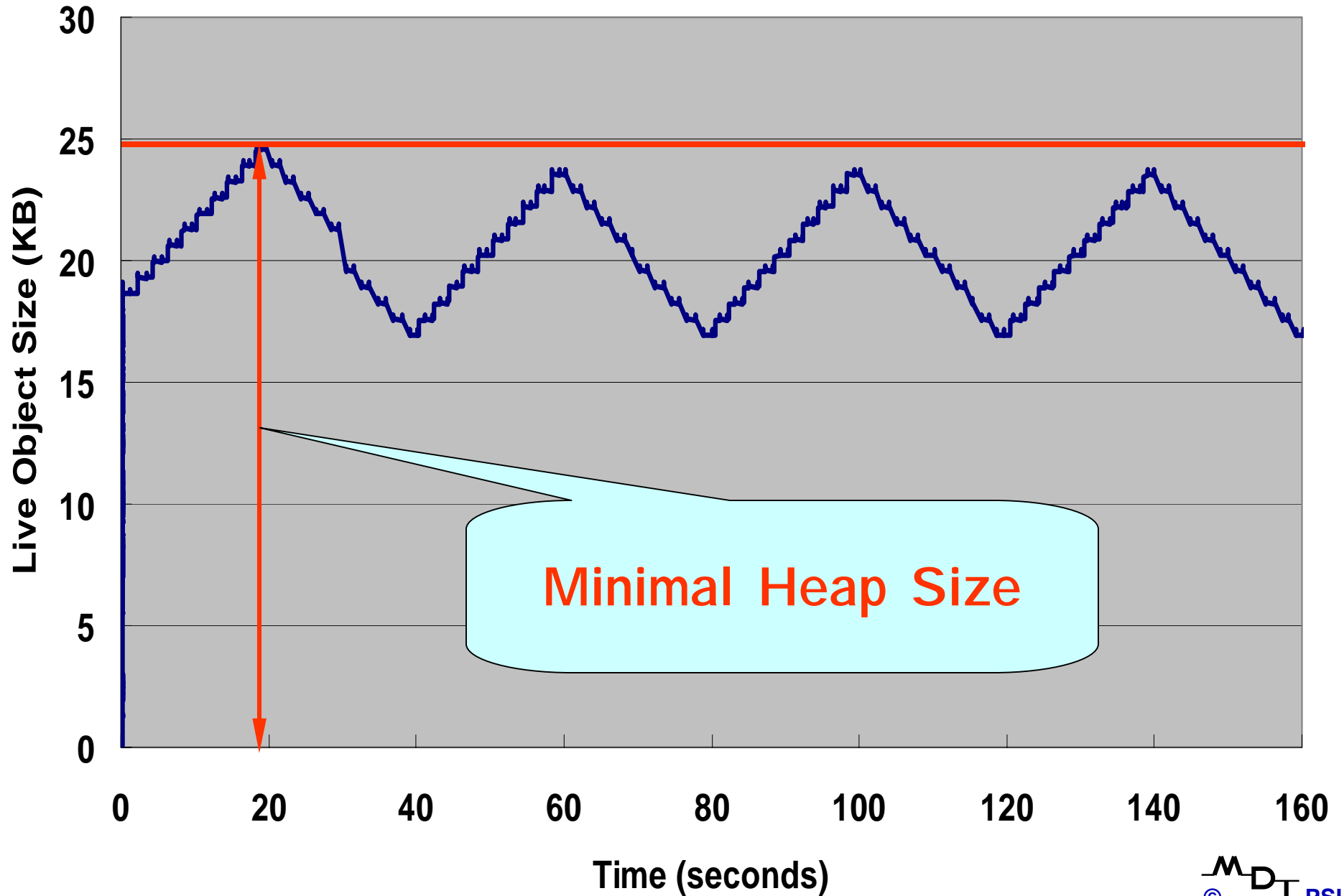


# Memory Allocation Behavior of PennBench Suite

# Accumulated Memory Footprints

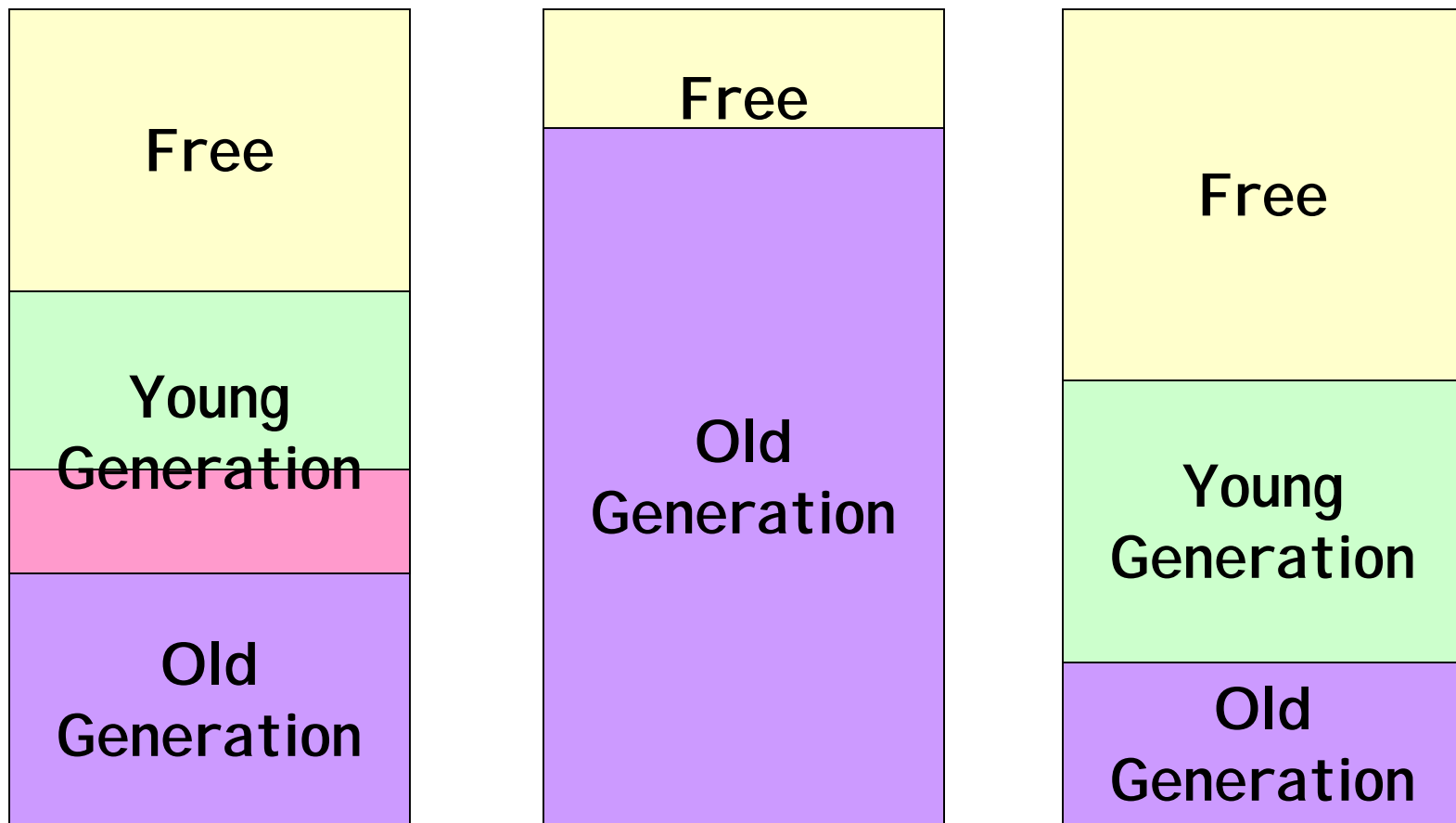


# Memory Footprints (Cont.)



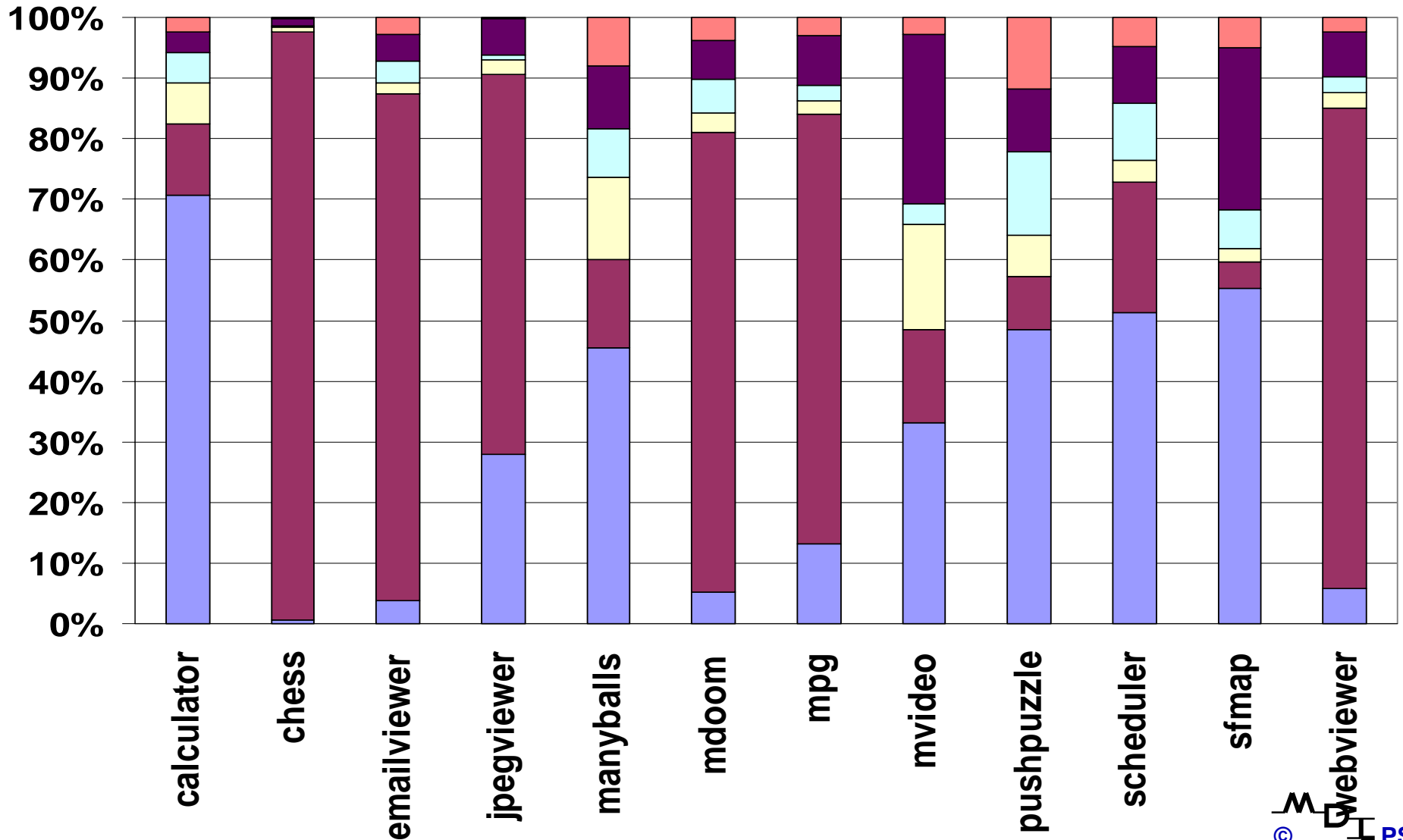
# Generational Garbage Collection

- Most objects live for very short time
- A small percentage of them live much longer



# Distribution of Object Lifetime

256B 1KB 4KB 16KB Other Permanent



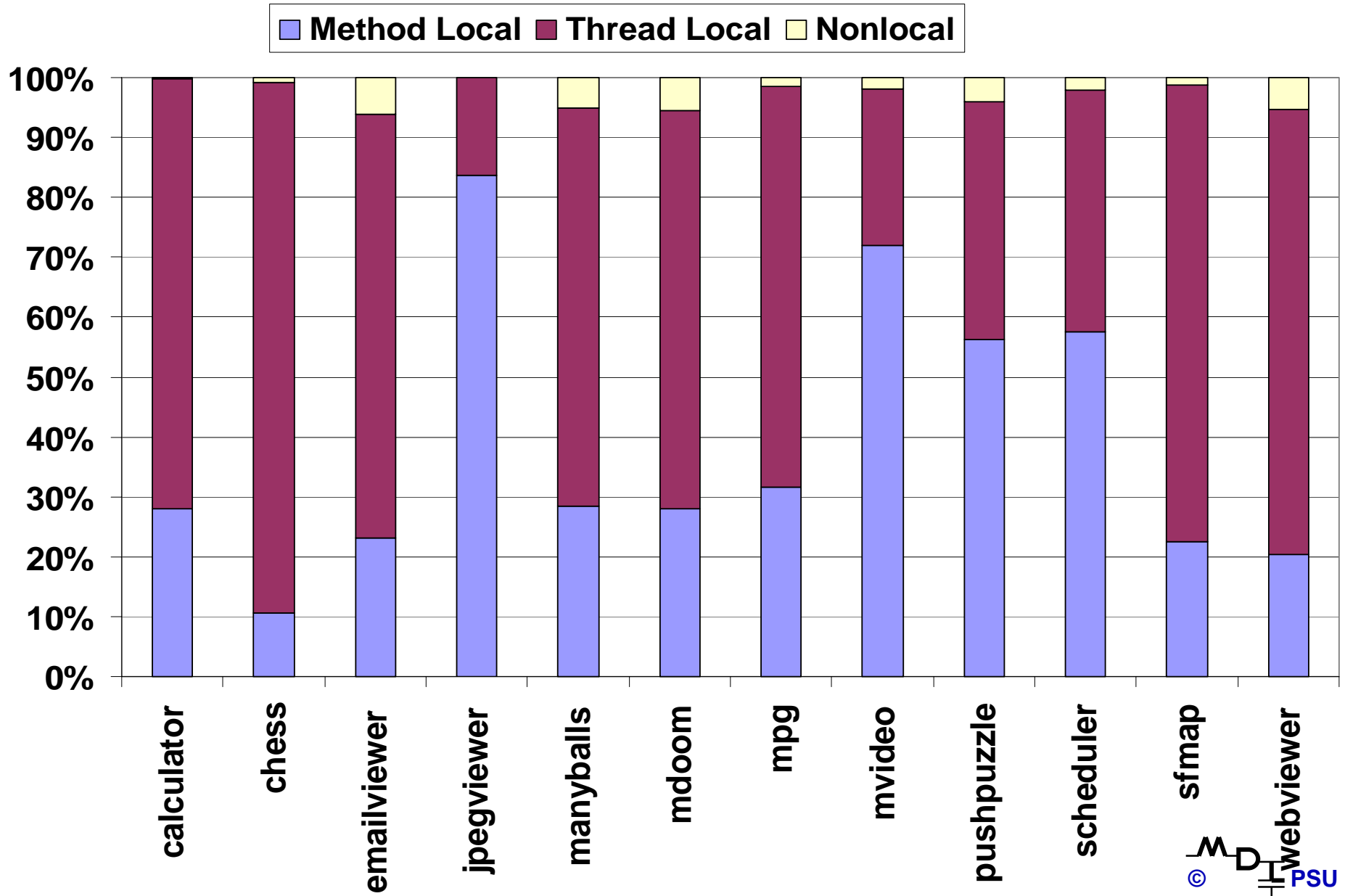
# Method Local Object

- Object O is local to method M if:
  - O is created in method M; and
  - O becomes garbage before M returns
- Stack Allocation
  - Allocate method local objects in the stack frame of their creating method
  - Method local objects are automatically collected when the frames are popped up due to method return or uncaught exceptions

# Thread Local Object

- Object  $O$  is local to thread  $T$  if:
  - $O$  is created by thread  $T$ ; and
  - No thread other than  $T$  accesses  $O$
  - A method local object is also thread local
- Elimination of synchronizations
- Scheduler-controlled memory turnoff:
  - Allocate thread local objects in the memory banks private to the thread
  - The scheduler sets the memory banks of non-active threads to sleep mode to conserve energy

# Method and Thread Local Objects



# Conclusions

- Our work confirms the observations that have been made on high performance Java systems
  - Only a small portion of methods are hot
  - Generational behavior of objects
  - A big portion of objects are method/thread local

## Conclusions (cont.)

- A benchmark suite for embedded Java is necessary
  - Most benchmarks for high performance systems are not able to run in embedded environments
  - The system configuration parameters that are optimal for resource-rich systems may not be optimal in resource-constraint environments

Thank you!

[gchen@cse.psu.edu](mailto:gchen@cse.psu.edu)

# PennBench Suite

Benchmark	Description
calculator	A calculator running on MIDP
chess	Play chess game with computer
emailviewer	A light-weight implementation of POP3 client
jpegviewer	Display a JPEG image on the screen

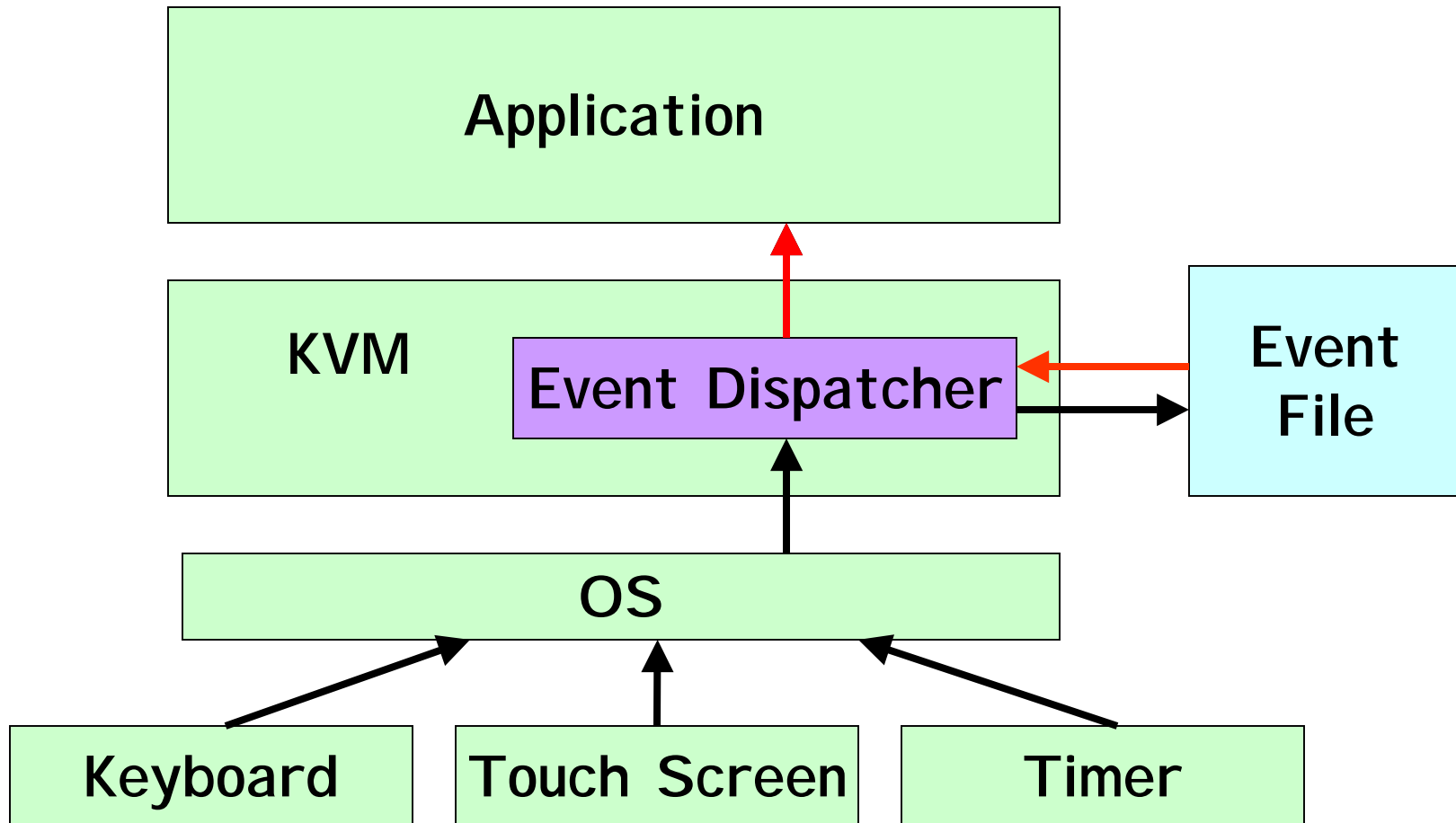
# PennBench Suite (cont.)

Benchmark	Description
manyballs	Draw many bouncing balls on the screen; each ball corresponds to a Java thread
mdoom	3D shooting game
mpg	MPEG movie player (local)
mvideo	Stream movie player (through HTTP)

# PennBench Suite (cont.)

Benchmark	Description
pushpuzzle	A conventional puzzle game
scheduler	Personal monthly scheduler
sfmap	Digital map on MI DP
webviewer	WWW browser

# Event Play Back Mechanism



# Property access method

## Java code

```
class String {  
    ... ..  
    private int count;  
    ... ..  
    int length() {  
        return count;  
    }  
    ... ..  
}
```

## Bytecode

```
Method int length()  
    0 aload_0  
    1 getfield #22  
    4 ireturn
```