Steve Poole
May 2013

Compact Off-Heap Structures in the Java Language

Important Disclaimers

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY.

WHILST EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED.

ALL PERFORMANCE DATA INCLUDED IN THIS PRESENTATION HAVE BEEN GATHERED IN A CONTROLLED ENVIRONMENT. YOUR OWN TEST RESULTS MAY VARY BASED ON HARDWARE, SOFTWARE OR INFRASTRUCTURE DIFFERENCES.

ALL DATA INCLUDED IN THIS PRESENTATION ARE MEANT TO BE USED ONLY AS A GUIDE.

IN ADDITION, THE INFORMATION CONTAINED IN THIS PRESENTATION IS BASED ON IBM'S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM, WITHOUT NOTICE.

IBM AND ITS AFFILIATED COMPANIES SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION.

NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, OR SHALL HAVE THE EFFECT OF:

- CREATING ANY WARRANT OR REPRESENTATION FROM IBM, ITS AFFILIATED COMPANIES OR ITS OR THEIR SUPPLIERS AND/OR LICENSORS

About Me

Steve Poole

Works at IBM's Hursley Laboratory in the UK

Involved in IBM Java VM development since before Java was 1

Currently leading IBM's OpenJDK technical engagement

contact spoole at uk.ibm.com

"Compact Off-Heap Structures in the Java Language"

This technology is being developed to help address two important pressures on Java

1: How to improve Java interop with Non Java applications

2: How to maximize Java performance for large heaps

What see and what you'll learn

"Compact Off-Heap Structures in the Java Language"

A walk-through of a prototype technology being developed by IBM

You'll learn:

- Why IBM thinks this capability should be added to Java
- Why we need your input and support
- How easy it is to get started using this technology

Part one - modern challenges for Java

Here's a common style of conversation:

"I need a Java binding to drive our new graphics API"

"You do know the new API is written in 'C'?"

"So? - use JNI"

"You want this binding to be slow?"

"What about NIO or Unsafe?

"Unsafe needs privileged access, NIO means hardcoding offsets - not flexible. AND you need to know 'C'

"Oh.. OK then,don't bother"

Interop : Java only speaks Java

- Getting data into and out of Java always requires some form of serialization process
- Interaction with native data structures in memory are particularly problematic
- JNI is the slowest but safest
  - But you need a good C / C++ knowledge
- Unsafe and NIO are faster but more challenging to use
  - They both have their own programming 'model'
- If the Java side is in control of storage layouts it's easier
- If you're mapping existing native structures its much,much,much more difficult
- And, finally, what about cross platform support?

Interop: Why can't I...

reference data stored like this

```
typedef struct {

        int red,green,blue;
        float vx,vy;
        SDL_Rect rec;
        float x,y;

    }  RECT;
```

C structure

Interop: Why can't I

as if it looked like this?

```
typedef struct {

        int red,green,blue;
        float vx,vy;
        SDL_Rect rec;
        float x,y;

    }  RECT;
```

C structure

```
public class Rect {

    int red,green,blue;
    float vx,xy;
    SDLRect rec;
    float x,y;

}
```

Java structure

Interop: The simple answer is..

You can if you play by Java rules

```
typedef struct {

        int red,green,blue;
        float vx,vy;
        SDL_Rect rec;
        float x,y;

  }  RECT;
```

copy

```
public class Rect {

    int red,green,blue;
    float vx,xy;
    SDLRect rec;
    float x,y;

}
```

C structure

Java structure

Interop: The simple answer is..

There are ‘good’ reasons why not

```
[red]
[green]
[blue]
[vx]
[vy]
[rec.x]
[rec.y]
[rec.h]
[rec.w]
[x]
[y]
```

Internal C structure
+ types are not a good match

```
[header]
[red]
[green]
[blue]
[x]
[y]
[vx]
[vy]
[instance-ptr]
```

```
[header]
[x]
[y]
[h]
[w]
```

Internal Java structure
At the liberty of the JVM

Interop: The simple answer is..

There are ‘good’ reasons why not

[red]
[green]
[blue]
[vx]
[vy]
[rec.x]
[rec.y]
[rec.h]
[rec.w]
[x]
[y]

The ironic point - the JVM is very aware of native types and layouts. Part of its job is to the necessary translations for you - very fast

[header]
[red]
[green]
[blue]
[x]
[y]
[vx]
[vy]
[instance-ptr]

[header]
[x]
[y]
[h]
[w]

Internal C structure
+ types are not a good match

Internal Java structure
At the liberty of the JVM

Challenge 1: Improving native memory access

- Under the covers the JVM could easily support a high speed interop..
  - –But the semantics of native data interaction, allocation mechanisms and generally dealing with the 'It's not a object' problem can't all be dealt with in a hidden way
- We need to be able provide annotations at the application / Java class level to explain to the JVM that sets of Java classes represent a native storage layout
  - –effectively indicating groups of objects that work together and have the same lifecycle and allocation characteristics

Another common style of conversation:

"I just turned on large pages and it went slower"

"What do the experts say?"

"Too many TLB misses "

"Whats a TLB?"

"I have no idea"

"Did you get any advice?"

"Yes - but I didn't understand it"

"What are you going to do?"

"I just turned off large pages"

Heap Sizes

Pop Quiz:

Q1 - How big is the biggest Java heap you know?

Megabytes, Gigabytes, Terrabytes, Petabytes, Exabytes?

Heap Sizes

Pop Quiz:

Q1 - How big is the biggest Java heap you know?

Megabytes, Gigabytes, Terrabytes, Petabytes, Exabytes?

A1 - It doesn’t matter - they are all coming...

Challenges of (very) large heaps for Java

- Large heap sizes are enabled through large (huge) memory page facilities at OS and processor level

- Tradeoffs in modern processor design mean that effective use of large memory pages will require specific characteristics of application design.

- The main requirement is “Locality of reference”

> Ignoring this requirement can impact your throughput significantly!

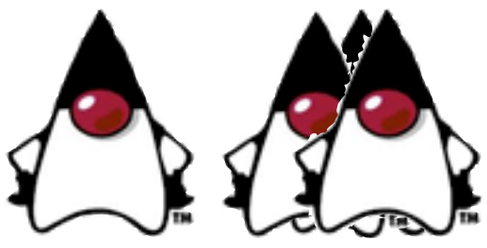
Challenges of (very) large heaps for Java

- Large heap sizes are enabled through large (huge) memory page facilities at OS and processor level

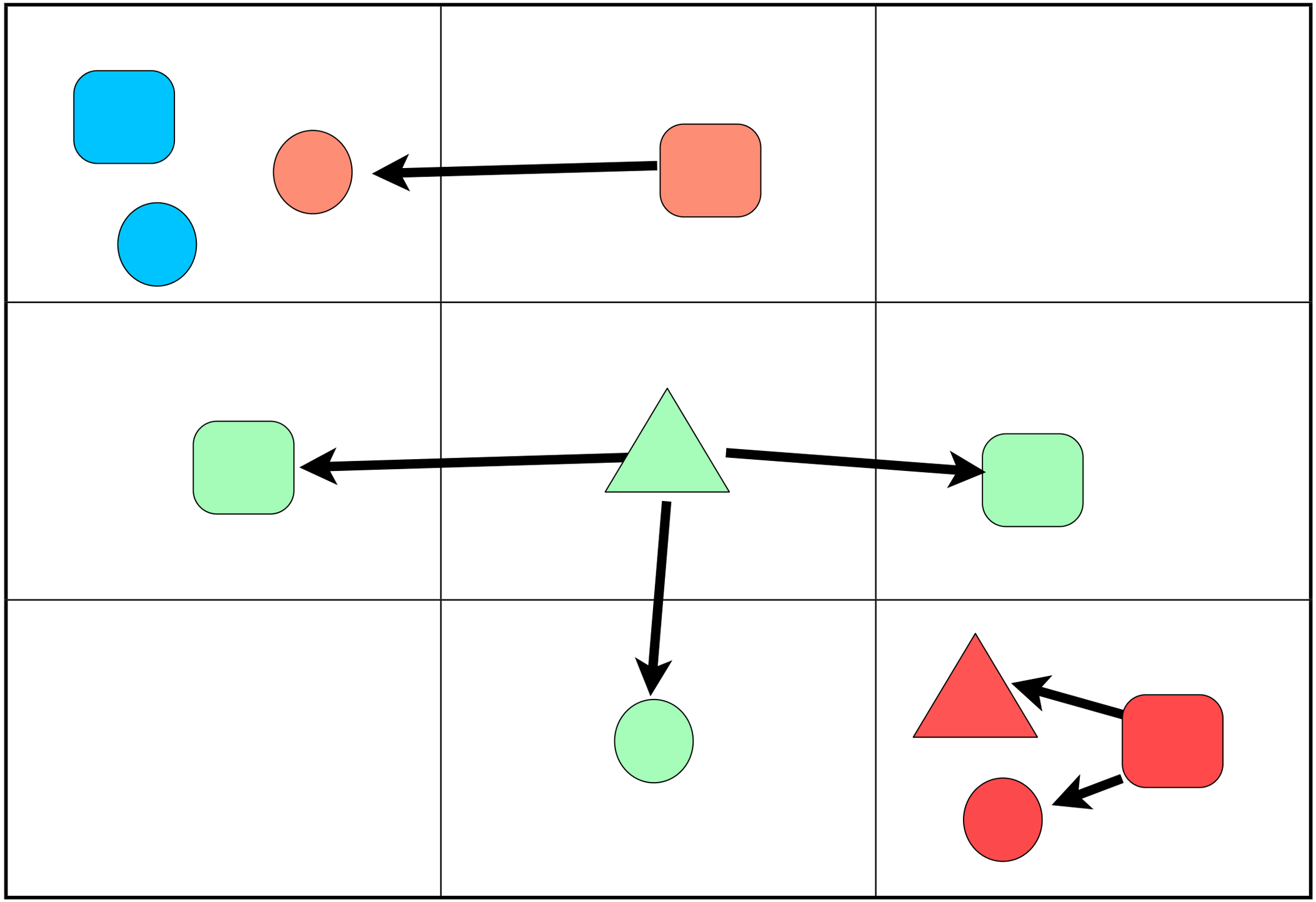
- Tradeoffs in modern processor design mean that effective use of large memory pages will require specific characteristics of application design.

- The main requirement is “Locality of reference”

Ignoring this requirement can impact your throughput significantly!

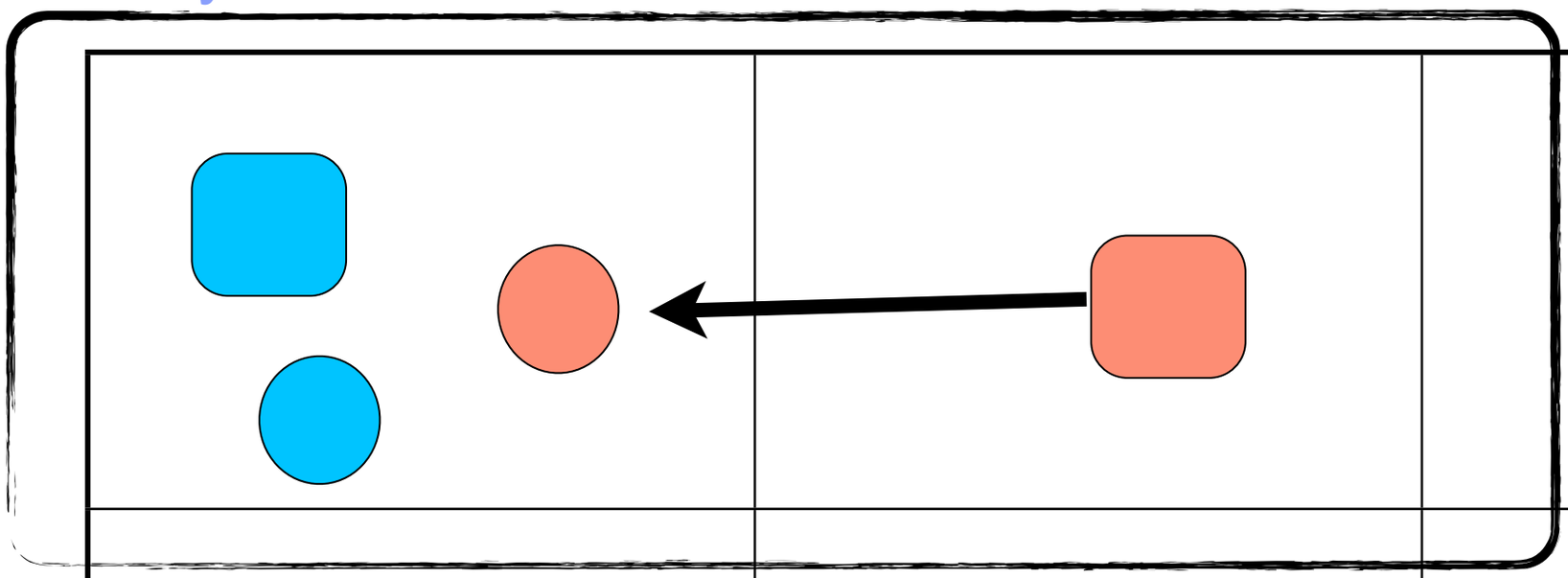


Locality of reference



Model of a Java heap - cells represent memory pages

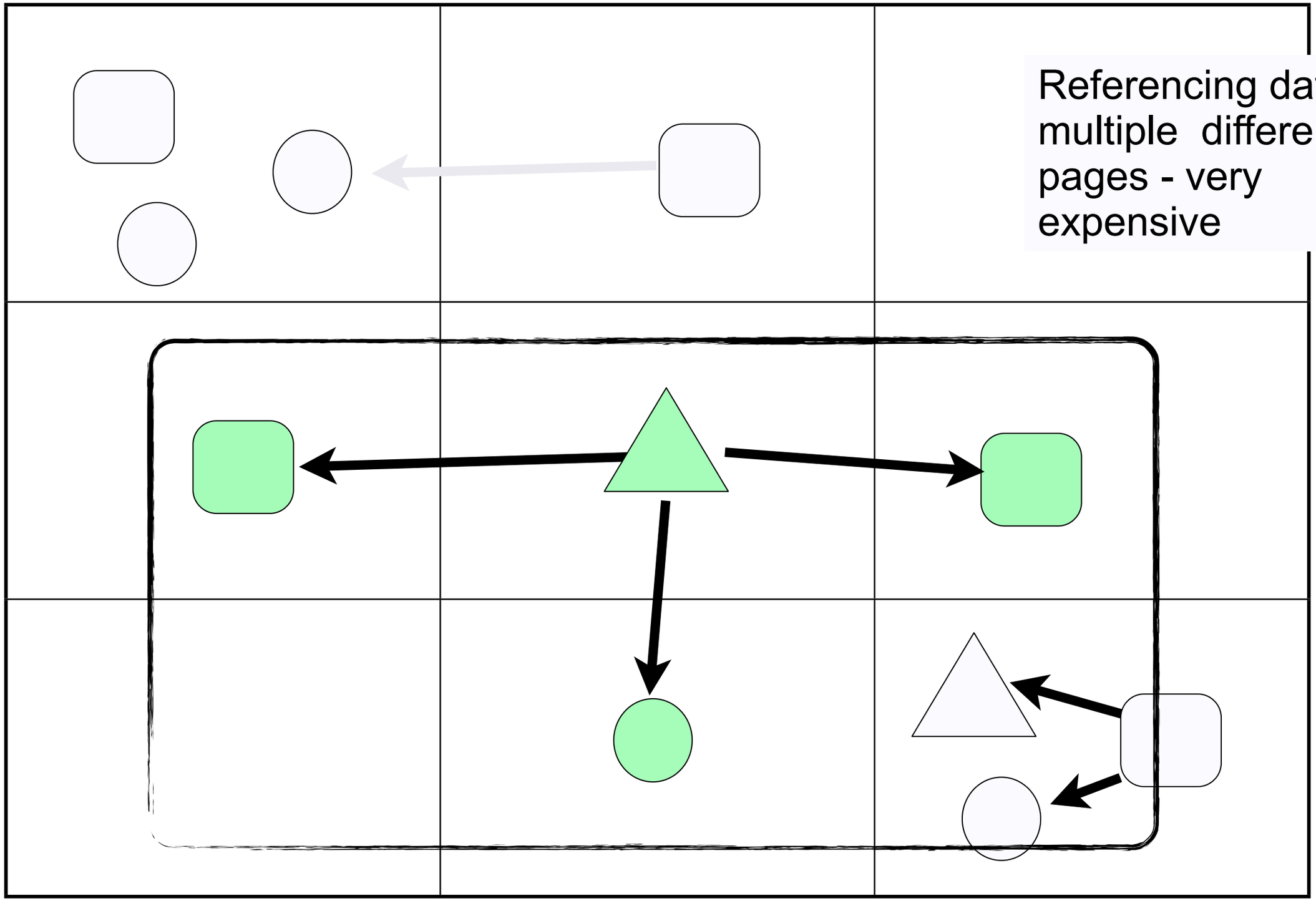
Locality of reference



Just referencing data on a different page is expensive
You may even affect someone else's perfomance

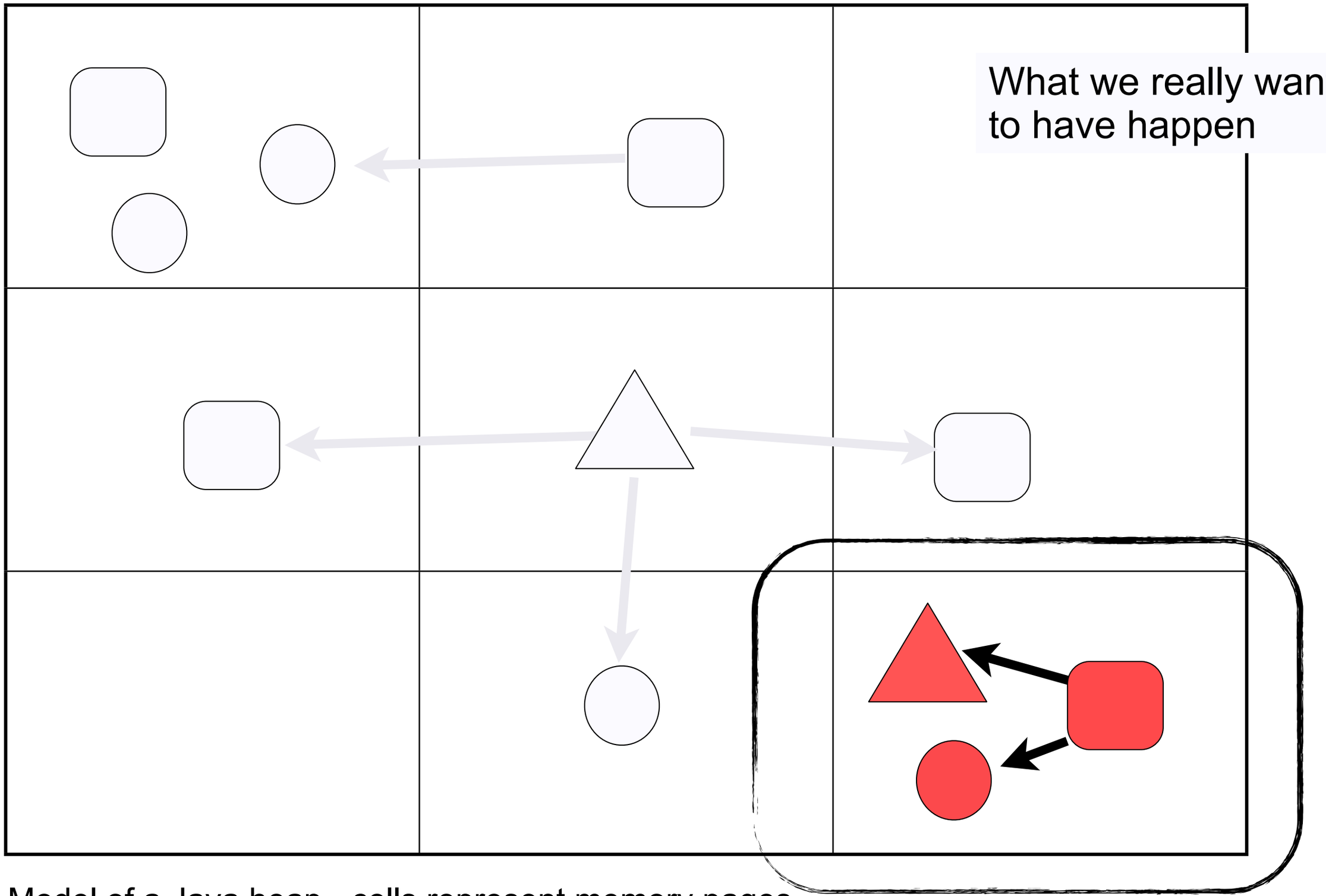
Model of a Java heap - cells represent memory pages

Locality of reference



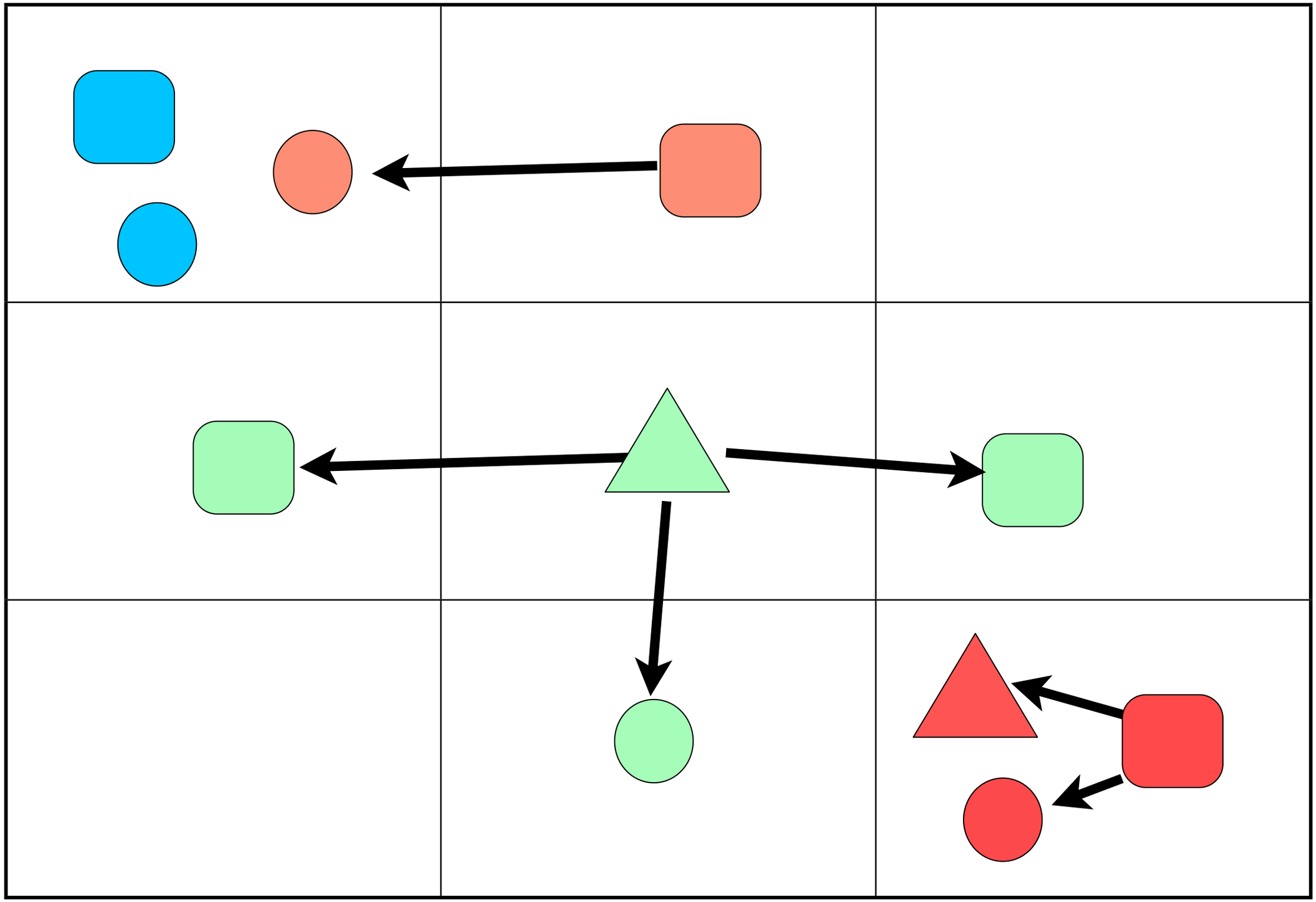
Model of a Java heap - cells represent memory pages

Locality of reference



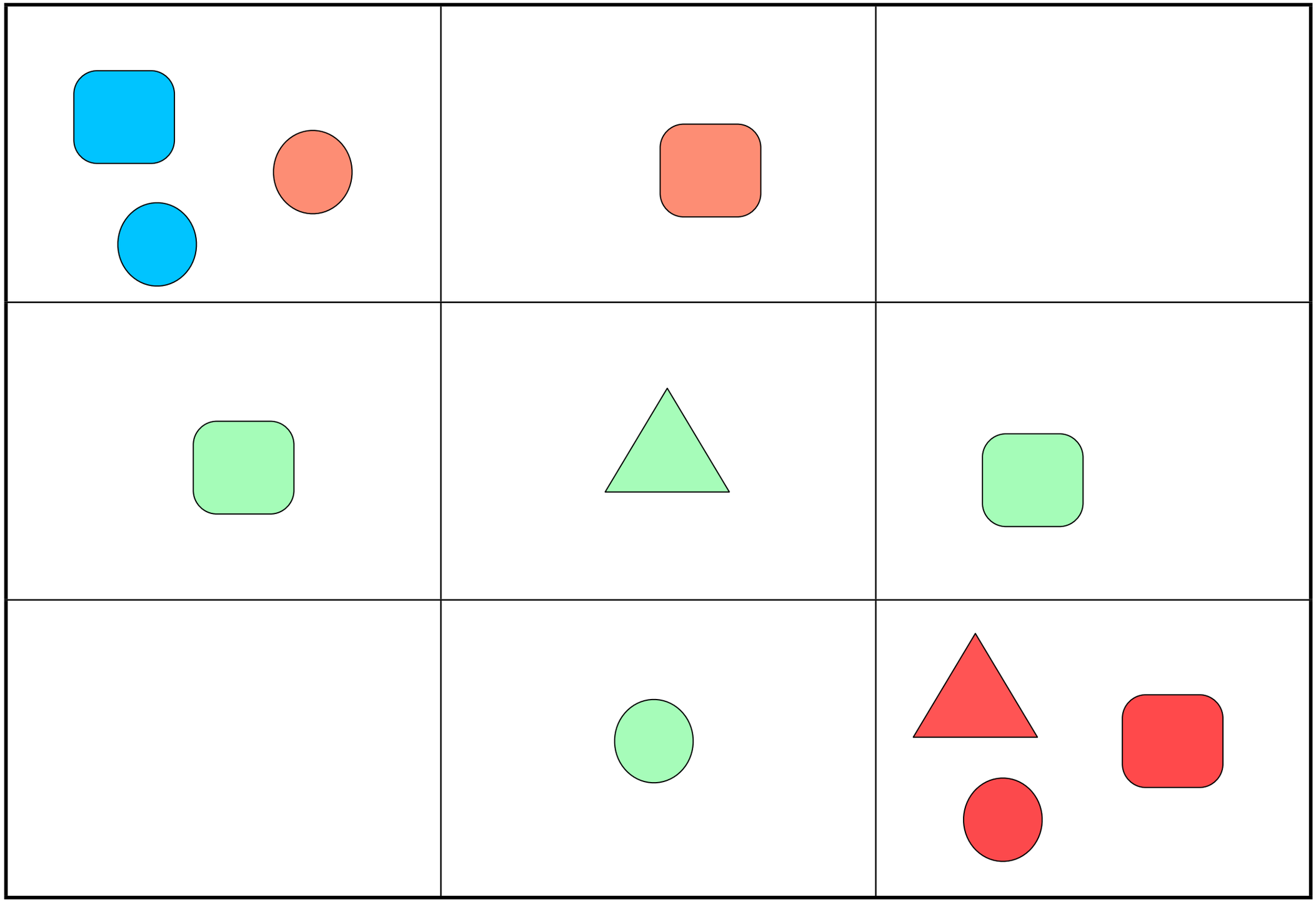
Model of a Java heap - cells represent memory pages

Locality of reference



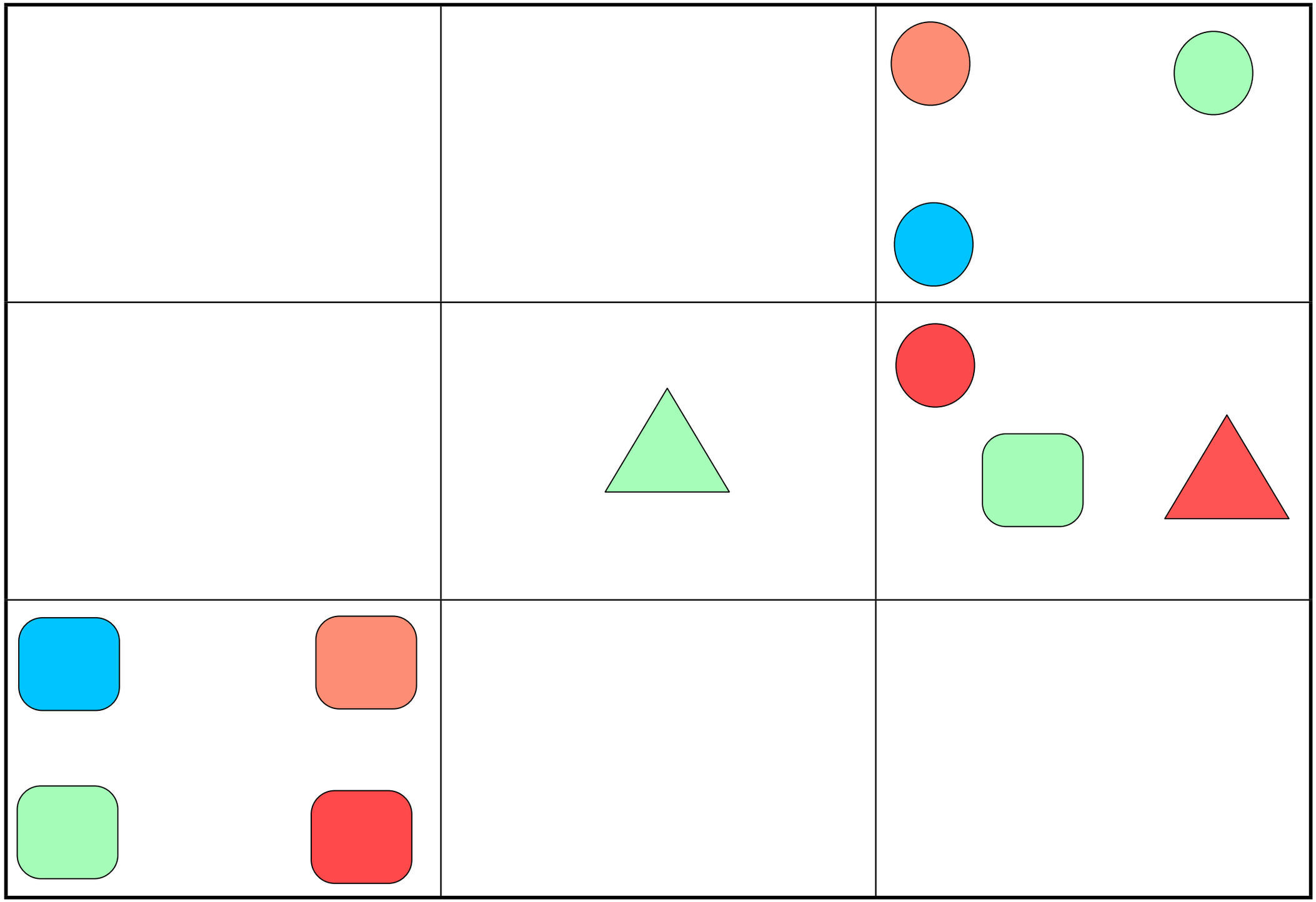
Model of a Java heap - cells represent memory pages

Locality of reference and GC models



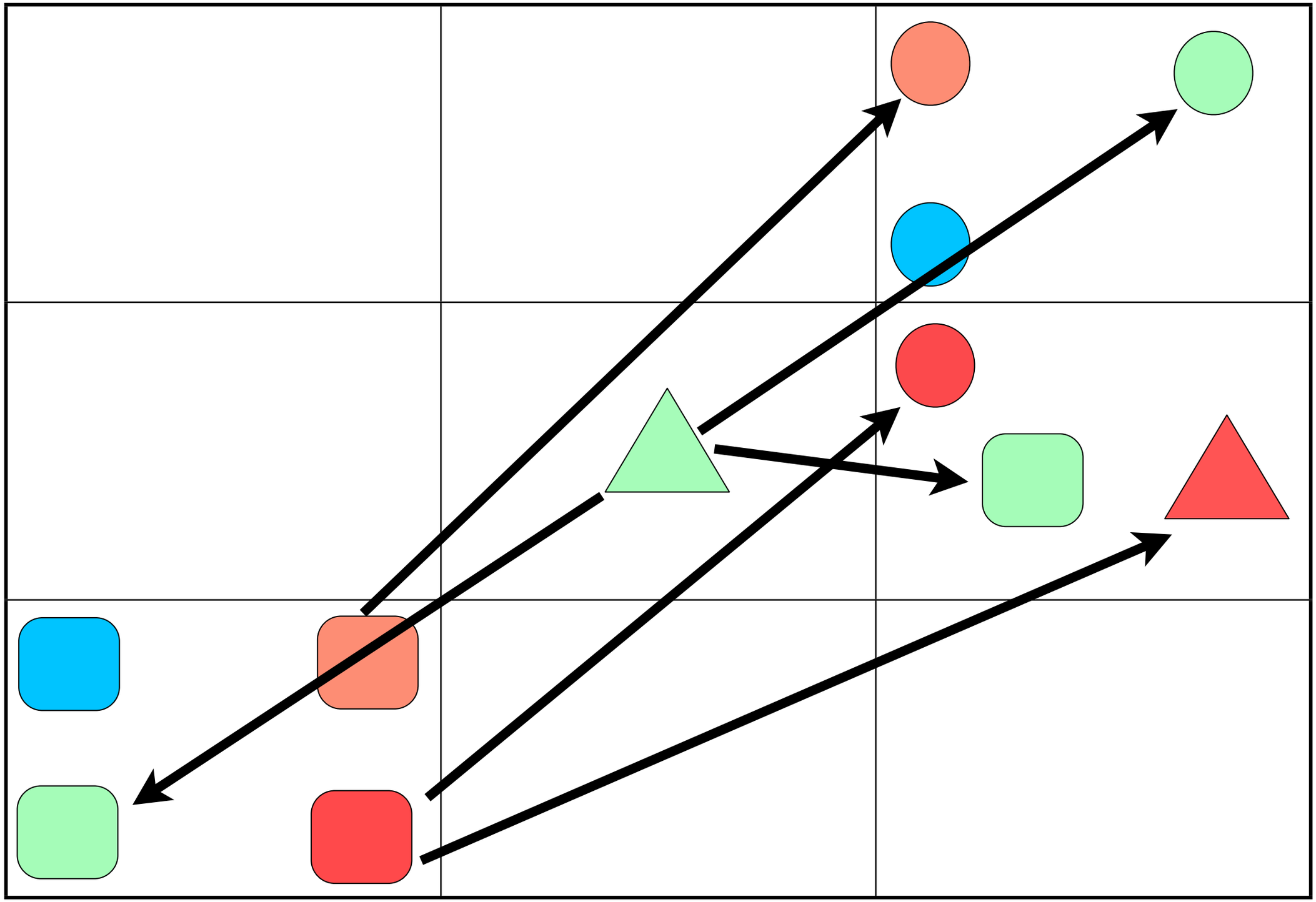
Model of a Java heap - cells represent memory pages

Locality of reference and GC models



Model of a Java heap - cells represent memory pages

Locality of reference and GC models



Model of a Java heap - cells represent memory pages

Challenge 2: locality of reference. It's just a GC problem isn't it?

- The JVM Garbage collectors can do a great deal to handle appropriate object allocation but ..
  - –Current allocation policies based on object size and life cycle characteristics are beginning to be challenged.
  - –New designs for GC are required and are being developed.
  - –However - heuristics and JIT runtime analysis only gets you so far...

- We need to be able provide annotations at the application / Java class level to explain to the JVM what to keep close together

–effectively indicating groups of objects that work together and have the same lifecycle and allocation characteristics

Challenge 2: locality of reference. It's just a GC problem isn't it?

- We need to be able provide annotations at the application / Java class level to explain to the JVM what to keep close together

–effectively indicating groups of objects that work together and have the same lifecycle and allocation characteristics

Similar requirements for different challenges

Challenge 1: Improving native memory access

- We need to be able provide annotations at the application / Java class level to explain to the JVM what to keep close together

–effectively indicating groups of objects that work together and have the same lifecycle and allocation characteristics

Challenge 2: locality of reference. It's just a GC problem isn't it?

- We need to be able provide annotations at the application / Java class level to explain to the JVM what to keep close together

–effectively indicating groups of objects that work together and have the same lifecycle and allocation characteristics

Part two - ‘Packed Objects’

A simple screen saver style program

Draw 100 000 random sized rectangles using SDL

Move them around the screen and get them to change direction if they hit the sides

```
typedef struct {

      int red,green,blue;
      float vx,vy;
      SDL_Rect rec;
      float x,y;

  }  RECT;
```

RGB colour elements

position adjustment vectors

SDL structure contains location , height and width

float version of location - makes it easier to handle different h/w speeds

Demonstrations

Demo	Frames per second	number of JNI calls per frame
Standard - all C code calling the SDL graphics routines		
Mixed C and JNI. Java updates the location of each element		
Mixed C with PackedObjects - Java updates the location of each element		

main logic for demo 1

```
while active
  for each RECT
        increment the x location by the x vector value
        increment the y location by the y vector value

        if(the x location is now out of bounds )
              reset the x location and invert the x vector

       if(the y location is now out of bounds )
              reset the y location and invert the y vector

       update the SDL_Rect with integer versions of the
       new  x & y locations

       Call the SDL paint routine to draw the RECT

 end
end
```

Demo 1 ...

Demonstrations

Demo	Frames per second	number of JNI calls per frame
Standard - all C code calling the SDL graphics routines	20	0
Mixed C and JNI. Java updates the location of each element		
Mixed C with PackedObjects - Java updates the location of each element		

main logic for demo 2 - mixed C and JNI

while active
  for each RECT
        call into Java via JNI passing in copies of x,y,vx,vy

increment the x location by the x vector value
increment the y location by the y vector value

if(the x location is now out of bounds)
reset the x location and invert the x vector

if(the y location is now out of bounds)
reset the y location and invert the y vector
Call a JNI native method passing in the updated values

Yes, we really are crossing the JNI boundary 2 times per rect, &-we are copying data every time.

update the RECT structure with the new values

update the SDL_Rect with integer versions of the new x & y locations

Call the SDL paint routine to draw the RECT

 end
end

Demo 2 ...

Demonstrations

Demo	Frames per second	number of JNI calls per frame
Standard - all C code calling the SDL graphics routines	20	0
Mixed C and JNI. Java updates the location of each element		
Mixed C with PackedObjects - Java updates the location of each element		

Demonstrations

Demo	Frames per second	number of JNI calls per frame
Standard - all C code calling the SDL graphics routines	20	0
Mixed C and JNI. Java updates the location of each element	11	200000 (~4MB per frame)
Mixed C with PackedObjects - Java updates the location of each element		

2 Million JNI calls per frame. Copying data, validating data everytime. AND ensuring there is no chance of optimization by the JIT.

main logic for demo 3 - mixed C and PackedObjects

?

Demo 3 ...

Demonstrations

Demo	Frames per second	number of JNI calls per frame
Standard - all C code calling the SDL graphics routines	20	0
Mixed C and JNI. Java updates the location of each element	11	200000 (~4MB per frame)
Mixed C with PackedObjects - Java updates the location of each element		

Demonstrations

Demo	Frames per second	number of JNI calls per frame
Standard - all C code calling the SDL graphics routines	20	0
Mixed C and JNI. Java updates the location of each element	11	200000 (~4MB per frame)
Mixed C with PackedObjects - Java updates the location of each element	18	1 (4 bytes per frame)

Packed Object version in more detail...

main logic for demo 1

```
while active
  for each RECT
        increment the x location by the x vector value
        increment the y location by the y vector value

        if(the x location is now out of bounds )
              reset the x location and invert the x vector

       if(the y location is now out of bounds )
              reset the y location and invert the y vector

       update the SDL_Rect with integer versions of the
       new  x & y locations

       Call the SDL paint routine to draw the RECT

 end
end
```

main logic for demo 3 - C and PackedObjects

```
while not active
  call Java via JNI
    for each RECT
          increment the x location by the x vector value
          increment the y location by the y vector value

        if(the x location is now out of bounds )
             reset the x location and invert the x vector

        if(the y location is now out of bounds )
             reset the y location and invert the y vector

        update the SDL_Rect with integer versions of the new  x & y locations

    end
   for each RECT

       Call the SDL paint routine to draw the RECT

   end
end
```

Only one JNI call per frame

Updating the native struct values directly

Once per frame...

```
(*env)->CallObjectMethod(env,obj,methodid,timestep);
```

The Java Demo class reference “LDemo;”

The update method “(F)V update”

smothing data

Once per program...

```
rectid=(*env)->AllocNativePackedArray(env,rectclazz,NUM_RECTS,rects);

(*env)->CallObjectMethod(env,obj,initid,rectid);
```

The Java objectid for the array

The Java method “([LDemo;)V init”

The Java class reference “LRect;”

The adress of the local C array ‘OffHeap”
OR leave NULL and get one created for you ‘OnHeap’

Interop: The simple answer is..

You can if you play by Java rules

```
typedef struct {

        int red,green,blue;
        float vx,vy;
        SDL_Rect rec;
        float x,y;

    } RECT;
```

copy

```
public class Rect {

    int red,green,blue;
    float vx,xy;
    SDLRect rec;
    float x,y;

}
```

C structure

Java structure

Interop: The simple answer is..

You can now

```
typedef struct {

      int red,green,blue;
      float vx,vy;
      SDL_Rect rec;
      float x,y;

} RECT;
```

==

```
public class Rect extends PackedObject{

      int red,green,blue;
      float vx,xy;
      SDLRect rec;
      float x,y;

}
```

```
public class SLDRect extends
PackedObject{

      int x,y,w,h;
}
```

All you need to do is tell Java the class is Packed.
It does all the rest.
Even cross platform!

C structure

Java structure

Interop: Consequences and questions

You can't synchronize on PackedObjects
or have a finalize() method.

Since you could be sharing the data with others outside the scope of the JVMs awareness

What about constructors?
Good question - serialization model?

```
public class Rect extends PackedObject{

    int red,green,blue;
    float vx,xy;
    SDLRect rec;
    float x,y;

}
```

```
public class SLDRect extends
PackedObject{

    int x,y,w,h;
}
```

This reference to a class must be a Packed Object too
No mixing that way.

What about Strings then?

@Length(n) char[] foo

In reality w & h are unsigned fields.

Seamless dealing with unsigned types
is not part of the prototype

Performance: Consequences

Since these PackedObjects can also be created ‘on’heap
We’ve got a possible model for dealing with locality of reference.
And, since we can use native types (such as real booleans and chars) we can squeeze more onto the heap.

```
public class Rect extends PackedObject{

    int red,green,blue;
    float vx,xy;
    SDLRect rec;
    float x,y;

}
```

```
public class SLDRect extends
PackedObject{

    int x,y,w,h;
}
```

Show's (almost) over folks

- You've seen a quick example of a prototype system developed by IBM
- We think this is an important technology and shows what could be done to in this area.
- We have a good prototype for fast interop, and a possible model for dealing with larger heaps

- We're working with Oracle and others on next steps to make this real.
- It's a long road - Spec and Language changes always are.

- Please help socialise.

- You too can try out this technology: Linux x86 only (well, and ZOS 31bit too)

- Google for "IBM Java 8 beta" and join. Its' free

The current prototype implementation surfaces a lot of the internals that will eventually just disappear

This approach makes it easier for us to quickly revise the design but means users of the prototype have to work harder

I've intentionally hidden the scaffolding during this talk

picture courtesy http://www.flickr.com/people/karen_od/

What you've seen and what you've learnt

"Compact Off-Heap Structures in the Java Language"

A walk-through of a prototype technology being developed by IBM

Google for "IBM java 8 beta"

You've learnt:

Why IBM thinks this capability should be added to Java

Java 9: we need greatly improved native data interop to keep Java vital (solving some of the large heap challenges would be good too)

Why we need your input and support

A change of direction for Java? Lets talk about it.

How easy it is to get started using this technology

"I need a Java binding to drive our new graphics API"

"You do know the new API is written in 'C'?"

"So? - use PackedObjects"

"OK!"

Thank you

any questions - 1 - now

2 - stop me during the conference

3 - email me spoole at uk . ibm . com