



 periodic rate of 20Hz. Navigational cue computation is computed at a periodic rate of 5Hz. The lowest priority is the computation of the performance data at 1Hz.

vide intelligence, surveillance and reconnaissance data. The ScanEagle software, called urations and interactions and interactions. These product scenarios were developed using the rate grouped usin<br>The group of the gr *i* 9 *b* 9 *b* 9 *and all is the screen method.* that calls from outside the vector object's parent scope do not cause memory errors. In this example, after it is in Area is the MemoryArea of the receiver. The newArray() method is used to reflectively allocate the strikingly, . <sub>k</sub> new backing store of the **vector**. Ovector:: vector:: vecto generic memory management object (1) thisArea.newArray(Object.**class**, cap); **in an RTS**<br>Presence of an Object (1) arr=(Object (1) thisArea.newArray(Object.**class**, cap); **in an Indivin**g Here, we see a system.arraycopy(elementData,0,arr,0,elementCount); operation c  $\frac{1}{2}$  conservative semi-space collection for lower priority in the collection of the priority of the priority of the priority of  $\frac{1}{2}$ tures. Consider the case where  $\sim$  vector::  $\blacksquare$  Noter domain code also requires world ensure Capacity (  $\mathcal{L}$  and possible and possible and possible as the memory error. This is not acceptable as the memory in special care in the presence of  $\begin{array}{ccc}\n\vdots \\
\text{Object[] arr=(Object)}\n\end{array}$ nter of the coross-of the cross-domain involved and the message objection protocol, namely the message of the m **modified <b>java.util.Vector** method. elementData=arr; The duce is the Moment was of the receiver. The next duce queue is the memory and or the receiver. The new analy  $\vee$  ector: **if** ( $\frac{1}{2}$ ) **fail**() **failed**  $\frac{1}{2}$  **fail**() **f**  $\frac{1}{2}$  **fail**() **f**  $\frac{1}{2}$  **f** scoped memory. Here, we see a  $F_{\rm s}$  is the path of throwing and the storechecks. new backing store of the **Vector**. void ensureCapacity(int cap){ ... elementData=arr; }  $\frac{1}{\sqrt{2}}$  33.Second-aware libraries. Growing a generic data structure must be performed in the original allocation  $\frac{1}{\sqrt{2}}$ **thisArea** is the **MemoryArea** of the receiver. The **newArray()** method is used to reflectively allocate the



- static<br>dint area-<br>int static VM\_Address\* getMem(TransientArea\* area, jint size) { jint s1 = area + area->offset; area->offset += size; jint s2 = s1 + (&SplitRegionManager)->ALIGN;
- the means of an *Event Channel*. An event channel is a standard interface for decoupling event producers jint s3 = (area->offset == area->rsize)? (size-(&SplitRegionManager)->ALIGN) : size; PollingAware\_zero(roots->values[57]), s2, s3);
- Polli<br>
The Factor  $\overline{1}$ **return** sl; }

be disabled because the event handler may call into common code that was compiled with with and without pollchecks. Notice that the worst-case **indicates** indified in overhead is around 2.5%. We see that although pollchecks require code to be added to every method, it does not **scope Scope** significantly impact performance. The annual source source and into an image: the application sources

}



operat allocate memory in scoped memory in scoped memory in scoped memory areas. Fig. 3  $\mu$ ▼ Scheduling of threads is done without the help of the and increading the basic we will be the bottom over the bottom and the bottom and the bottom control of the bottom<br>operating system. When code is compiled to lava, we insert pollchecks, which rapidly check if a scheduling decision needs inte ponencenc, military encern a concealing accident necessarily interto be made. Pollchecks are inserted at back branches (loops) that and optionally at method entry, insuring the the number of word instructions between pollchecks is bounded. When a class of the set pollcheck fires, the event handling and thread management ena framework decides which thread to run next. **values** a particular kind of the ma operating system. When code is compiled to Java, we insert instructions between pollchecks is bounded. When a

tionality, using both real and simulated components. As part of the components of the components of the components of the components of the components. As part of the components of the components of the components of the c **throws** PragmaNoPollcheck, PragmaNoBarriers { The autonomous navigation capabilities on an University of an Unma  $\frac{V_{\text{IV}}}{V_{\text{IV}}}-\frac{1}{2}$  and  $\frac{1}{2}$  and  $\frac{1}{2}$  are provided by the GNU  $\frac{1}{2}$  of fset += size; Mem.the().zero(ret.add(ALIGN), offset == rsize?size-ALIGN:size); **for more than 15 hours. The primary operation of the ScanEagle vehicle is to pro-** $V1$ - $A$ uuress gernem (Inc.  $SLE$ ) **throws** PragmaNoPollcheck, PragmaNoBarriers { VM\_Address ret = base().add(offset); context is restored, setCurrentArea, and scratch memory is reclaimed with leave().

tor es<br>14 A de P —for example, the **throws** clause above is used to specify pragmas that alter the execution of this method. Further, the specify regimes and different to execution of any mediod. Thereft, the bump pointer allocater in contraction of the bump pointer allocater in contracter in contracter in contracter in contracter in contracter in contracter in cont scenarios which is the various component interactions of component interactions for component in the component interaction in the component interaction in the component interaction in the component interaction in the compo actual avionics systems. The settem systems  $\mathbf{r}$  representative contain  $\mathbf{r}$ **VM\_Address** class is ephemeral—it does not correspond to an object at run time; instead calls to it are translated into pointer manipulation operations. This bump pointer allocater in class TansientArea. This is the bump pointer allocater in class TansientArea. This is the bump pointer allocater in class TansientArea. This is the bump of the bump o

> $V$  , which is variable to  $V$  and  $\mathcal{M}$  and  $\mathcal{M}$  are  $\mathcal{M}$  . The approximation  $\mathcal{M}$ **if** (off.uLT(VM\_Word.fromInt(heapSize))) **return** heapArea;

of  $\mathbb{R}$  and  $\mathbb{R}$  are different memory. From  $\mathbb{R}$ 

The memory management is used to allocation of the reflective data structure data stru

rm<br>Thus woid readBarrier(VM\_Address src)  $T_{\rm eff}$  instead require time  $T_{\rm eff}$  is to ensure timely completion of  $E$ on careful scope lifetime management and memory access check elision. **throws** PragmaInline, PragmaNoBarriers, PragmaNoPollcheck { <sup>14</sup> *·*

 $\text{inccked in real-}$  int sb = src.asInt() >>> blockShift; time Java, to insure that longer-<br>int tb = tgt.asInt() >>> blockShift;<br>if (sb != tb) storeCheckSlow(sb, tb);  $\begin{bmatrix} \text{Sineles point at} \\ \text{Sineles}, \text{Obiect} \end{bmatrix}$  if (i.e. ime is determined by the object's scoped memory area. A write barrier is used to perform this check. The Ovm store check fast path is shown in this figure. The fast path simply verifies that the **b**  $\blacksquare$  **if** Writes to memory also void storeCheck (VM\_Address) int sb = src.asInt() >>> blockShift; int tb =  $tgt.asInt()$  >>> blockShift; **if** (sb != tb) storeCheckSlow(sb, tb); } lived objects never point at the state of the store checks!  $\frac{1}{2}$   $\begin{bmatrix} 1, 1 \ 1, 1 \end{bmatrix}$  shorter-lived ones. Object which can be either health can be either h  $\blacksquare$   $\blacktriangleright$  Writes to memory also need to be checked in real- $\frac{1}{2}$ ; shorter med ones, solped  $\frac{1}{2}$  $\log$  objects are in the same page. throws PragmanoPolicheck, PragmanoBarriers, Pragm Int sp = src.asint() >>> biocksnitt**;**  $\text{Int} \text{CD} = \text{Cyc}.\text{asInt}(1) \implies \text{Dloc}.\text{SInt}(1)$ II (SD  $:=$  CD) SLOTECHECKSIOW(SD, CD); **if** (!sidx.uLessThan(scopeBlocks)) **fail**(); lifetime is determined by the object's scoped memory area. A write barrier is used to perform this }

throws PragmaNoPollcheck, PragmaNoBarriers, PargamNo<br>VM\_Word tidx = VM\_Word.fromInt(tb - scopeBaseIndex); checks. Many critical path methods use PragmaAtomic. Hence, it should be possible to **if** (!tidx.uLessThan(scopeBlocks)) **return**; Area ta = scopeOwner[ tidx.asInt() ]; void storeCheckSlow(int sb, int throws PragmaNoPollcheck, Pr }  $\rm VM\_W$ between an interrupt and a poll-check that services that interrupte. We can case observed latency is 6  $\mu$ 

in the object. We simply VM\_Word off = VM\_Address.fromObject(mem).diff(heapBase); **if** (off.uLT(VM\_Word.fromInt(heapSize))) **return** heapArea; off = VM\_Address.fromObject(mem).diff(scopeBase); **if** (!off.uLT(VM\_Word.fromInt(scopeSize))) **return** immortalArea; **the ail extra neader field**  $int$  int idx = off.asInt() >>> blockShift;  $\mathcal{L}$ the target of the reference is a heap location. This code is inserted before every load of a reference field and is **scopeOwner** array). To find the memory area of an object, we first round down the object's base address to the base of the page, and then look up the memory area  $\frac{1}{2}$ process is fast and reduces memory usage by eliminating the need for an extra field in the object Fig. 2. RTSJ with path. storeCheck Slower areaOf(Oop mem) {  $\blacksquare$  In Ovm, finding the  $\blacksquare$  is a reactive memory to a stromobility of  $\blacksquare$ momory area that owne and  $\frac{1}{\sqrt{2}}$  is a reference to an object with shorter or disjoint shorter o memory area that owns an **if** maintain a page-to-memory- based is a state of  $\frac{1}{2}$ area mapping (see the the target of the reference is a heap location. The field before the memory location header. **return** scopeOwner[idx]; } secoped. We keep a mapping from memory and associated with the page. The address to the base of the page, and then look up the memory area associated with the page. The  $\mu$  decays such the source code of the virtual matrix  $\mu$ 





}





However, two additional goals can be interested. The main operation is the main operation is appase).uLessThan(heapSize)) fail(); deal with garbage collector  $\frac{1}{2}$  high priority threads do not have to heapSize))  $fail()$ ; deal with garbage collector







 $\blacktriangledown$  One of the concerns of using a pollcheck scheme for scheduling is the time between pollcheck executions. If this  $\frac{V}{L}$ latency is too great, scheduling decisions may come too  $10<sup>5</sup>$ infrequently. This histogram shows the pollcheck latency in  $\rho_{\text{m}}$  is the fact path is a low path is a local and a compared. The slow path involves disability in microseconds. The worst case is about 6 microseconds.

 $\epsilon_{\text{rel}}$  + All Java code in Ovm is executed via the J2c execution engine, which converts Java code to C++. In this code cation for the pass Botton tests. Over the pass Boston tests. Over the pass and Prismi method seen previously converted to C++. We compile all Java methods to module- $\blacksquare$  local C++ functions, allowing the C++ compiler to perform inlining. Most method calls are devirtualized—for example **Mem.the().zero()** is translated into the direct call example, we see the **getMem** method seen previously  $F = \begin{bmatrix} P_0 \end{bmatrix} \begin{bmatrix} \text{in} \Delta W2B & \text{zero} \end{bmatrix}$  and  $\begin{bmatrix} \text{in} \Delta W2B & \text{zero} \end{bmatrix}$ **PollingAware.zero**(). Also, every **VM\_Address** turns into an **Exercise Lineaply and PollingAware.zero** (). Also, every **VM\_Address** turns into an backend. The receiver object is made explicit in the translation as an additional argument to the method. Address *i o*<sub>mig</sub>. created while in a scope dimension area, then the seemingly simple operation of placing simple operation of pla  $t_{\rm eff}$ 

**b** Pollchecks are fast to **volatile** int $16\text{-}t$  not Engle execute, and fast to disable. A  $\frac{1}{2}$  s;<br>**bulge bump bump bump bump bump points** interact point  $\frac{1}{2}$  **execute**, and fast to disable. A pollcheck simultaneously bellunion; checks for two flags: *signaled* this checks will be translated and enabled. The signaled flag <sub>pott curck</sub>. nsert is set asynchronously by  $\overrightarrow{if}$  (pollUnion.pollWord = needs interrupt handlers written in C pollUnion.s.notSignaled  $\text{const.}$  that defect conditions that for a hierarchy pollUnion.s.notEnabled = 1;  $\mathcal{L}$  manipulation and VM  $\mathcal{L}$  and  $\mathcal{L}$  handleEvents(); of would require rescheduling  $\frac{1}{2}$  manufactors (1,  $\mathcal{P}_{\mathcal{P}_{\mathcal{P}}}$ merrupt nanaters whiten in

that are recognized by the Overation of  $\mathcal{S}$  and translated into effect operations. **volatile** int16\_t notSignaled; the the such as  $\mathbf{v}$ execute, allo last to cusable. A **volatile** int32<sub>\_t pollWord;<br>collected intimate for the score for the Java code for the Java code for the *volution* of the *volution* of the *volution* of the *volution* of the *volution*</sub> **union** { **struct** { **volatile** int16\_t notEnabled; } s; } pollUnion; POLLCHECK: **if** (pollUnion.pollWord  $== 0$ ) pollUnion.s.notSignaled = 1; stored in variables (fields or array elements) that have a longer lifetime than the object  $\begin{array}{c|c} \texttt{1Word:} \end{array}$  $\frac{1}{16}$  **h** 

 $S($ such as a timer interrupt). The ent enabled flag specifies if pollchecks are enabled (clearing this flag bullets are in the same page enables atomic execution). The logic is set up to allow the fastest ception types. Methods can be a provided by a proportion by a proportion possible ponence. Maiou ha Ovm also supports a number of compiler pragmas, which are expressed in Java as ex-} possible pollcheck without having to use atomic instructions.

 $All no$ mp on<br>executi tion, the UAV is the UAV is controller to the UAV is controller to the UAV is controller to the UAV is control<br>The UAV is controller to the UAV is controller to operate under the UAV is controller to operate under the UAV<br>  $\blacktriangleright$  Basic Ovm architecture. The VM is split into the executive domain kernel, and the user domain, which contains the application and its libraries. Almost all of the Ovm is written in Java. All non-trivial Java bytecodes are converted to calls to *core services access* methods, which are implemented in the executive domain. The executive domain is also responsible for implementing scheduling, memory management, reflection, and I/O.

 $\mathbb{P}$  Ope single docute a<br>thread figure s code is byceco.<br>Howev  $\frac{1}{2}$  code are implemented in Java in the Ovm. This figure shows the code for allocation. Note that the code is syntactically Java, and gets compiled to Java bytecodes using an ordinary Java compiler.  $\frac{1}{2}$  and  $\frac{1}{2}$  are ret.  $\frac{1}{2}$  and  $\frac{1}{2}$  from  $\frac{1}{2}$  size,  $\frac{1}{2}$ However, the semantics of the code differ from Java  $\blacktriangleright$  Operations that other VMs implement in native

## <sup>2</sup> *·* route deconfliction algorithms for the SCANEAGLE Unmanned Aerial Vehicle (UAV)1. The SCANEAGLE  $\overline{\phantom{a}}$ the demand for an affordable, fully autonomous vehicle with high endurance. Equipped with an onboard inertial daylight video camera, SCANEAGLE A camera, SCANEAGLE A camera, SCANEAGLE A camera, traveling hundred of miles. Fig. 2 depicts the UAV and gives information about the hardware configuration used in flight.  $\blacksquare$ to the GNU  $\sim$  use as the standard library. The standard library  $\sim$  $\Omega$  is interesting the core function of  $\Omega$ casts, and the state all of this functions. Because all of this function  $\mathcal{L}$  $\epsilon$  is the Overlands instructions, the Overlands instructions, the Overlands in late these instructions into appropriate executive domain method calls. This is achieved via a glue layer called the CoreServicesAccess (CSA). For example, instructions such <sup>10</sup> *·* <sup>22</sup> *·* as MONITORENTER or  $\mathcal{A}$  are translated into calls to  $\mathcal{A}$  methods. Because a  $\mathcal{A}$ call leads to execution of Java code, recursive CSA calls are possible. For example, the  $\blacksquare$  $\mathbf{M}$ particles in the Times  $\sim$ Ovm and GCJ are ahead-of-time compiled, Hotspot is using just-in-time compilation. The  $\bullet$  of this experiment is to provide a performance baseline. We evaluate the set of the AVIONICS memory access checks. JTime, likewise, has ready write barriers turned on. In the case of the case of the case  $\begin{array}{c} \begin{array}{c} \bullet \\ \bullet \end{array} \\ \begin{array}{c} \bullet \\ \end{array} \\ \begin{array}{c} \bullet \\ \end{array} \end{array}$  $\blacksquare$ Pollcheck Latency in Microseconds 20 *Ovm: a Real-time Java Virtual Machine for Avionics*

 $\blacktriangleright$  Real-Time Java relies on the scoped memory API to guarantee that high-priority tasks can execute without garbage collector interference. The

> $\frac{1}{\text{Cone}}$  The Splitter of the Splitter of the Splitt College and the Splitt College that the Split Region of the Split Region means that the Splitter of olicy.the().enterScratchPadArea();<br>pad, a memory area that provides imported  $\log_{100}$  big bobjects that were allocated since the most  $Wildcard()$ ;<br> $Wildcard()$ ;  $\sigma$ ryManager.the ().setCurrentArea(r2); } appropriate scope. In this code example,  $\blacksquare$  Anatomy  $\left\{\n\begin{array}{c}\n\text{we see a method that implements reflective} \\
> \text{we see a method that implements reflective}\n\end{array}\n\right.\n\left\{\n\begin{array}{c}\n\text{use the WII} \\
> \text{example}\n\end{array}\n\right\}$ cCurrentArea(); **memory management API adds is the** *scratch* real-time, especially hard real-time, especially hard realfunctionality similar to **alloca** in  $C$ . This is a programming techniques that typically result in these failures: lazy initialization and dy-functionality similar to **alloca** in C. This is a recursive area—exiting it reclaims only those  $r_{\text{recv}}$ ;  $r_{\text{recv}}$  recent entry. This allows us to allocate  $\blacksquare$  temporary objects without having to find the executive domain. Because it calls in the executive domain. Because it One of the features that the Ovm

**if** (!doLoadCheck) **return**;

}

discovery.

 $\overline{a}$ needs to allocate the temporary **InvocationMessage** object, we enter into the scratch pad using our look like the MemoryPolicy and MemoryManager APIs. If we had used the real-time Java scoped memory API, the with the exuses the two-products of the two-manners of the type various propriate scope—someting than upon or<br>sary in Oym  $\cot \theta$  would have to contain complicated logic for finding  $\alpha$ ever means have to comain comprisated regions imaing or Fig. 11. The return value or exception is retrieved. Before retrieved. Before returning, the same block  $\sigma$ that is never necessary in Ovm. Figure . If  $\alpha$  is a node object for each entry and to the list of the list, then added to the list, then adding to the list, then adding to the list, the list, then adding to the list, the list, then adding to the list, code would have to contain complicated logic for finding or allocating the appropriate scope—something

> $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$  store checks are always  $O(1)$  in  $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$  time and space. For a detailed description of the algorithm see *in real time.*

managed by the Ovm Async I/O Framework, seen in black. The POSIX I/O emulator, which provides  $\begin{array}{c} 0.0 \end{array}$ blocking I/O operations, is implemented in terms of the asynchronous operations provided by Ovm Async I/O. In turn, the Async I/O framework has multiple implementations, ranging from the Imprementation: 5.313 Is the Imprementation we use most nequently for real to<br>following figures we describe the Async I/O framework in detail. GNU CLASSPATH, which is our implementation of the Java class libraries. CLASSPATH expects to be able to use a POSIX I/O interface—so we provide it, using our POSIX I/O emulator. The scheduling is conservative Polling implementation (intended to work on any device) to the high-throughput select implementation. SIGIO is the implementation we use most frequently for real-time application. The the

 $\frac{1}{\tan \text{ keepLong}}$  and its performance tended to be in the same ballpark as HotSpot. optimized to call anoughput (like Flotspot) and for real-time<br>numbers are better. Note that we consider both real-tin configurations of the configurations of this problem over  $\epsilon$  POSIX III.O estimations of the configurations in the configurations in  $\blacktriangledown$  Ovm performance compared to a number of other VMs, including those optimized for throughput (like HotSpot) and for real-time (like jTime). Lower Ovm is configurations of the Ovm.  $GCJ$  and jTime were unable to complete a number cally and over the own is seen that we have the subset of the slow of mpegawal in the slow of model is due to s of benchmarks. In all cases where jTime completed the benchmark, it ran for much longer than Ovm. Ovm was the fastest RTJVM that we were able to test, by no means representative of a real-time application, but it gives a worst case estimate of other players in the game. RT-Zen has a pool of worker that it uses that it uses to see that it uses to see to see that it uses to see that it numbers are better. Note that we consider both real-time and throughput

the target of the reference is a heap location. This code is inserted before every load of a reference field and is

 $\epsilon$  C  $\epsilon$  of the O  $\epsilon$  accession diherenties and two anglications that we use Size of the Ovm, associated libraries, and two applications that we use. the Ovin itself consists of just over 200,000 liftes of code. The Physical Device layer twenty the Physical De Implementation of the KTSJ itself is quite small, but don't be looked—the  $\blacksquare$  libraries make heavy use of Ovm framework functionality that would not be there if we did not support the RTSJ. The GNU CLASSPATH library is considerably larger than the Ovm. PRiSMj, the ScanEagle application, and the UCI RT-Zen ORB are two applications that we run. Both are over arge see any report of the Fig. 35 in the flight system of the flight see any report of the flight see any report of  $\frac{1}{2}$  see any r ine Cym is en consists of just over 200,000 lines of code. The<br>implementation of the RTSJ itself is quite small, but don't be fooled—the RTSJ The Ovm itself consists of just over 200,000 lines of code. The

implementation of monitor entry may allocate memory using the NEW instruction, which  $\blacktriangleright$  The ScanEagle unmanned autonomous vehicle (UAV) with Domains in the Ovm are firmly segregated. The executive domain can only call into Machine. Having flown successfully and passed Boeing's internal and the user domain can only can only called the user only can only called the user only called the user of th qualification tests, the ScanEagle demonstrated the feasibility of **The Over Compiler recognizes UD** classing Real-Time Java in general, and the Ovm in particular in a library imports class are translated into calls to methods of the same name in the executive domain avionics applications. It was the first Real-Time Java system to do Because Ovem is written in Java, the ordinary Java, the ordinary Java not apply. The ordinary Java not apply. native method components, with special focus on scheduling and scoped into calls to call stated in the call stated int memory support. We also show the performance of the Ovm process of translating user domain native methods is called LibraryGlue. As such, the Boeing PRiSMj software and the Ovm Real-Time Java Virtual so. This poster describes the design of the Ovm's real-time using a variety of benchmarks.

> France and the second intervelopment in the second interest and no and far and Jan Vitek of Purdue University;<br>and the program of t **Event Channel And Company;** and all graphers and *Austin Armbuster and Edward Pla of the Boeing Company;* static analysis framework. We implement logging through program specialization: code that exe-*Research by Jason Baker, Antonio Cunei, Chapman Flack,*  $\overline{\phantom{a}}$ Filip Pizlo and Jan Vitek of Purdue University; tion. *Marek Prochazka of SciSys. David Holmes of DLTeCH; and*

 Overview of the PRiSMj application. Synchronized communication with the flight controls is a mission critical function and executed at a

operation completes. asynchronous, every operation requires a callback that's used Additionally, a handle is returned that

 $\lim_{\alpha \to 0} \frac{1}{\alpha}$  form of a p accept regular memory buffers (in the **AsyncHandle**

**AsyncMemoryCallback**







insure optimal interaction with the garbage collector in the case that the operation is implemented by a process that is not under the VM's direct control. Conservation is implemented by a process that is not under the VM's direct control. -

tions inserted by the compiler in the bytecode. All code paths must eventually encounter a poll-check instruction. muerruption<br>enforced by a read barrier that the compiler inserts before every heap read **throws PragmanoPollcheck, PragmanoPollcheck, Pragmano**Pollcheck, PragmanoPollcheck, PragmanoPollcheck, PragmanoPoli emored by a read barrier that the complier miserts before every neap read operation. This job is to Never receive the memory and the new processed by threads that may preclipt the concetor. The code for the Over read barrier is shown in this figure. Our read barrier is fast—we simply perform arithmet the target object's address to insure that it does not fall outside of the heap.<br>if (!) if (! Ovm read barrier is shown in this figure. Our read barrier is fast—we simply perform arithmetic on the target of the reference is not accessed by uneads that may preen in plict inserts before every neap read verify that the heap is not accessed by threads that may preempt the collector. The code for the enforced by a *read barrier* that the compiler inserts before every heap read operation. Its job is to







ncluding those humbers are better. In all benchmarks except for Period, d throughput benchmark, which measures the performance of priority plete a number inheritance locks, jTime was unable to complete the test. **EXECUTE:** Real-time performance microbenchmarks. Lower Lower Covm is at least as good as jTime. In the Inherit





to the increase<br>The impact of Predictability of RT-Zen running on the Ovm. We are very set of Predictability of RT-Zen running on the Ovm. Fig. 32. See Fig. 29 Fig. 29 Fig. 29 For percent overlapped over the results. See Fig. 29 for an application. And  $\frac{1}{2}$  have two throad groups. Low priority and high priority. are over handling 300 requests each. Of note is that we do not marking 300 requests each. Of note is that we do not<br>see any major outliers in request processing time. l, and have two thread groups, low-priority and high-priority, ACM Journal Name, Vol. V, No. N, Month 20YY.



 $\triangle$  The Ovm implementation of PRiSMj was the first application to qualify to fly on the ScanEagle UAV. Performance of PRiSMj on Ovm is shown in this figure. Response times of 100 threads on the server client requests.<br>split in these games of tight readium. In the age are deliverable adspitements groups (ingit, medium, low) on a modal workload are shown. The x-axis shows the number of data frames received by the UAV control, the y-axis indicates the time taken by by a thread to process the frame in milliseconds. Our jitter is  $\sim$ well within the 1% jitter target. split in three groups (high, medium, low) on a modal workload







**javax.realtime.MemoryArea** class serves as the parent class of the scoped memory area class hierarchy. Since the Ovm is written in Java, all virtual machine functions also need to be written in such a way as to avoid

collector interference. We do this by providing an internal memory management API that contains a superset of scoped memory features. I/O stack. At the top is

scenario provides autonomous auto-routing and health monitoring by communicating with the flight controls card, based on threats and no fly zone data from the ground station, and monitoring information.