AZUL
SYSTEMS

Pauseless GC in the Azul JVM™

Cliff Click
James McIlree
Gil Tene
Michael Wolf

Copyright © 2005, Azul Systems, Inc. All rights reserved.

Some Azul Background

- Azul is building an appliance that runs Java
- The Networked Attached Processing idea:
  - JVM proxy or shim runs on host
  - Bytecode execution on appliance
  - Security, visibility, file access, firewalls, etc on host
- Custom CPU
  - Core is classic 3-adr RISC, 32 regs, 64bit datapaths
  - Read & Write barrier instructions
  - Easy JIT target but no direct bytecode execution
  - Per core 16K I+16K D cache, 8 cores share 1Meg L2
  - 3 8-core clusters per die (24 cores/die)
  - 16-way SMP x 24 cores/die = 384 cores

 Copyright © 2005, Azul Systems, Inc. All rights reserved.

Property of Azul Systems, Inc. not for redistribution

Outline

AZUL SYSTEMS

- **Major Phases**
- Marking Phase
- Relocation phase
- Remap phase
- Current Status

 Copyright © 2005, Azul Systems, Inc. All rights reserved. Property of Azul Systems, Inc. not for redistribution

GC Phases

- GC runs in 3 phases: Mark, Relocate, Remap
- Fully parallel & concurrent during all phases
- No rush to finish any given phase
  - No cost beyond *read barrier* in steady state of any phase
  - Mark phase **will** finish despite busy mutators
  - Can produce free memory at any time
- "Self-Healing" - Mutators help when blocked by GC
  - Mutators test each loaded OOP with a *read barrier*
  - If *read barrier* traps, mutator does a little work
  - Loading same OOP won't trap again
  - Once working set is handled, no more traps

Copyright © 2005, Azul Systems, Inc. All rights reserved.

Property of Azul Systems, Inc. not for redistribution

Mark – Relocate – Remap

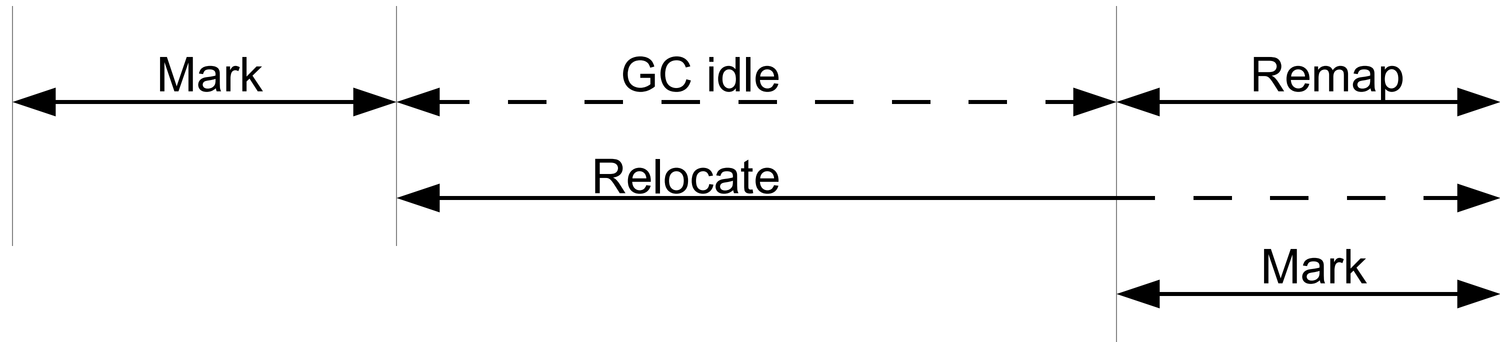
www.azulsystems.com

- Marking
  - Flip marking bit on all mutator thread roots
  - Follow & mark all live objects
  - Compute per-page liveness
- Relocation
  - TLB protect relocating 1M pages, move roots
  - Relocate remaining live objects out of protected pages
  - Free physical memory
- Remapping
  - Change all pointers to relocated objects
  - Free virtual memory

 Copyright © 2005, Azul Systems, Inc. All rights reserved.

Property of Azul Systems, Inc. not for redistribution

Mark – Relocate – Remap



- Always parallel & concurrent
- Next cycle Mark phase overlaps with Remap phase
- Can relocate (and free!) pages anytime after Mark
- Relocate stops at next Mark phase
- Future work: could relocate during Mark

 Copyright © 2005, Azul Systems, Inc. All rights reserved.

Property of Azul Systems, Inc. not for redistribution

Outline

- Major Phases
- **Marking Phase**
- Relocation phase
- Remap phase
- Current Status

 Copyright © 2005, Azul Systems, Inc. All rights reserved.

Property of Azul Systems, Inc. not for redistribution

Marking

AZUL SYSTEMS

- Flip marking bit on all roots
  - Initialize worklists
- Mark all live objects
  - Standard parallel work-stealing algorithms*
  - New objects marked live
  - Also compute per-page liveness
- Mutators *read barrier* loaded OOPs
  - 1 instruction, 1 cycle common case
- Finish Marks
  - Classic race with mutator discovery of large root removed

*Flood, Detlefs, Shavit, Zhang, JVM02,
"Parallel Garbage Collection for Shared Memory Multiprocessors"

 Copyright © 2005, Azul Systems, Inc. All rights reserved.

Property of Azul Systems, Inc. not for redistribution

Read Barrier and the NMT bit

www.azulsystems.com

- Steal 1 address bit (out of 64) from every OOP
  - The *Not-Marked-Thru* (**NMT**) bit
- *Read barrier* checks NMT bit
  - Traps if set wrong to fast user-mode handler
  - Trap handler informs Marker directly
  - Trap flips NMT bit in register to avoid propagating
  - Loaded value in Mutator working set, likely to load it again
  - Trap flips NMT bit in memory to avoid re-trapping
- "Self-Healing"
  - Once working set is marked, no more trapping

 Copyright © 2005, Azul Systems, Inc. All rights reserved.

Property of Azul Systems, Inc. not for redistribution

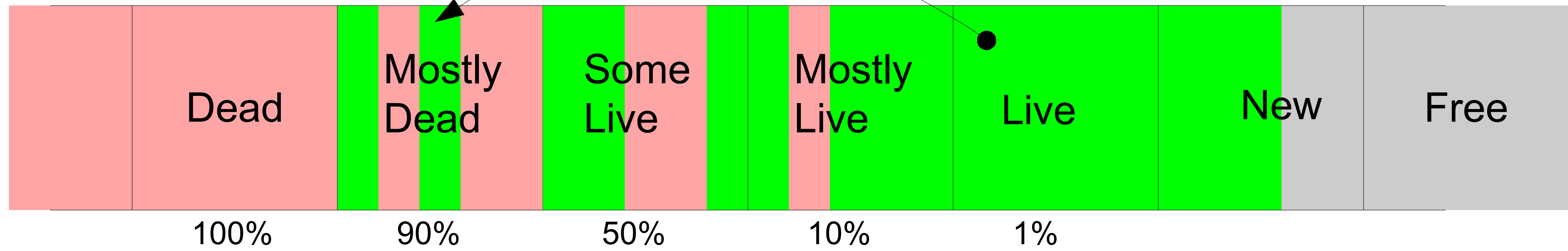
Outline

www.azulsystems.com

- Major Phases
- Marking Phase
- **Relocation phase**
- Remap phase
- Current Status

Copyright © 2005, Azul Systems, Inc. All rights reserved.
Property of Azul Systems, Inc. not for redistribution

Relocation #1



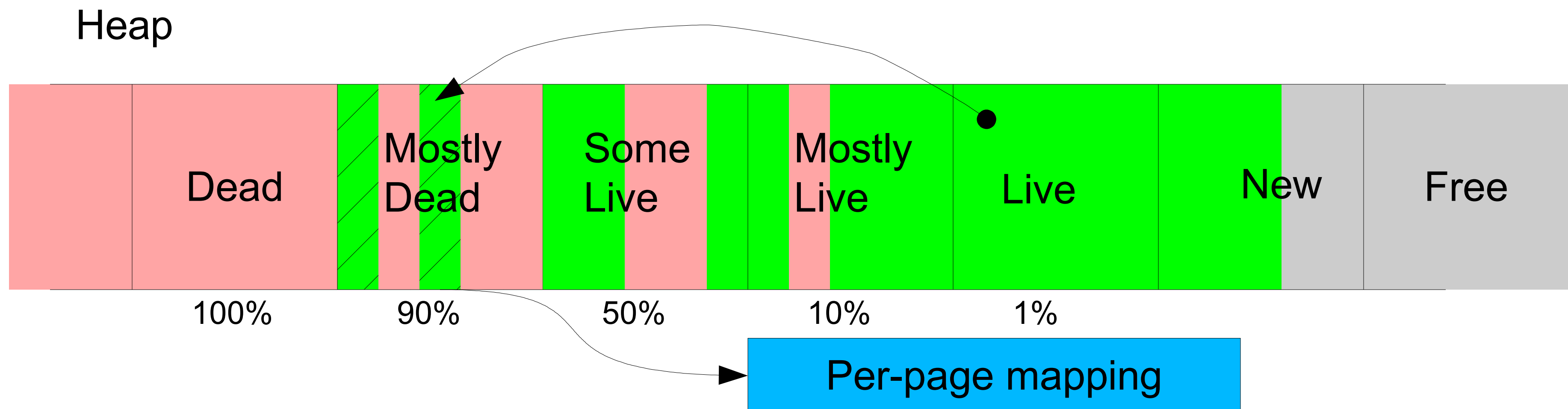
1. **Choose sparsely populated 1M pages**
   - **Marking phase found per-page free space**
2. Build side arrays to hold forwarding pointers

 Copyright © 2005, Azul Systems, Inc. All rights reserved.

Property of Azul Systems, Inc. not for redistribution

Relocation #2

www.azulsystems.com



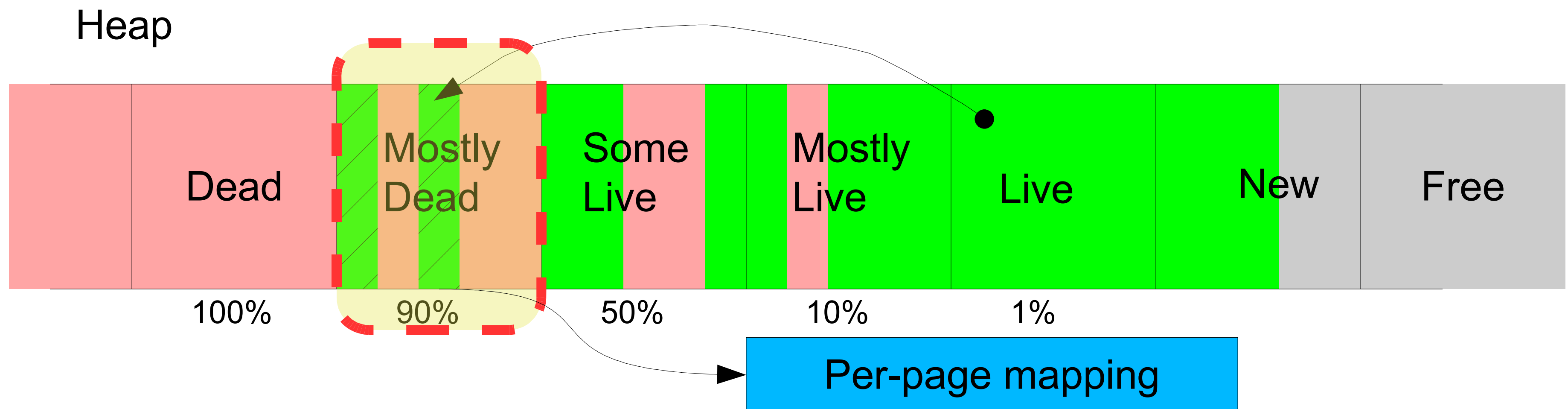
1. Choose sparsely populated 1M pages
   – Marking phase found per-page free space
2. **Build side arrays to hold forwarding pointers**
3. TLB-protect pages

 Copyright © 2005, Azul Systems, Inc. All rights reserved.

Property of Azul Systems, Inc. not for redistribution

Relocation #3

www.azulsystems.com



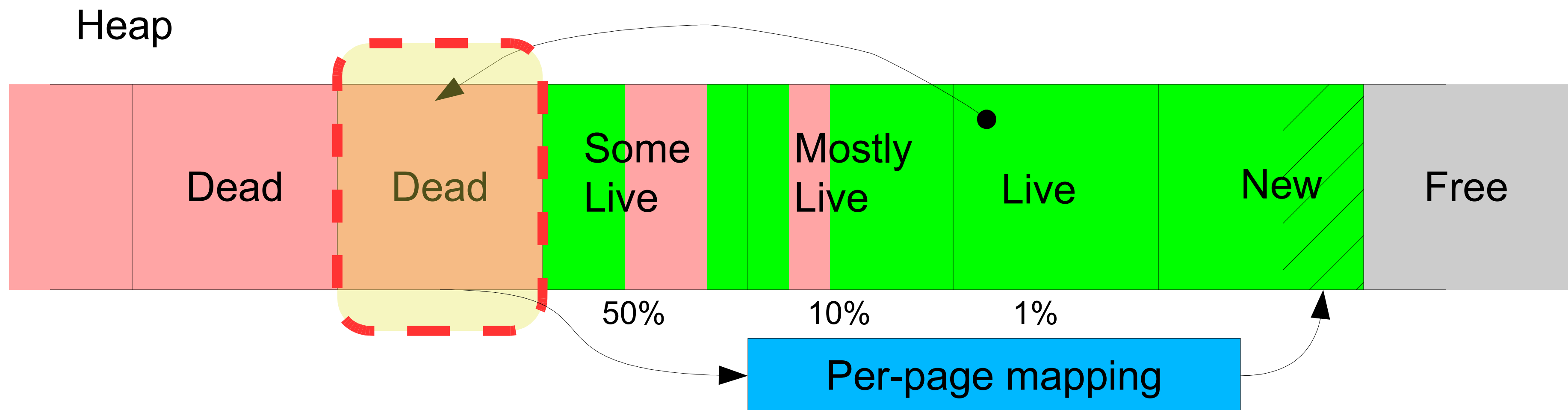
2. Build side arrays to hold forwarding pointers
3. **TLB-protect pages**
4. Copy live objects out
   – Leave dangling pointers for remap phase

 Copyright © 2005, Azul Systems, Inc. All rights reserved.

Property of Azul Systems, Inc. not for redistribution

Relocation #4

AZUL SYSTEMS



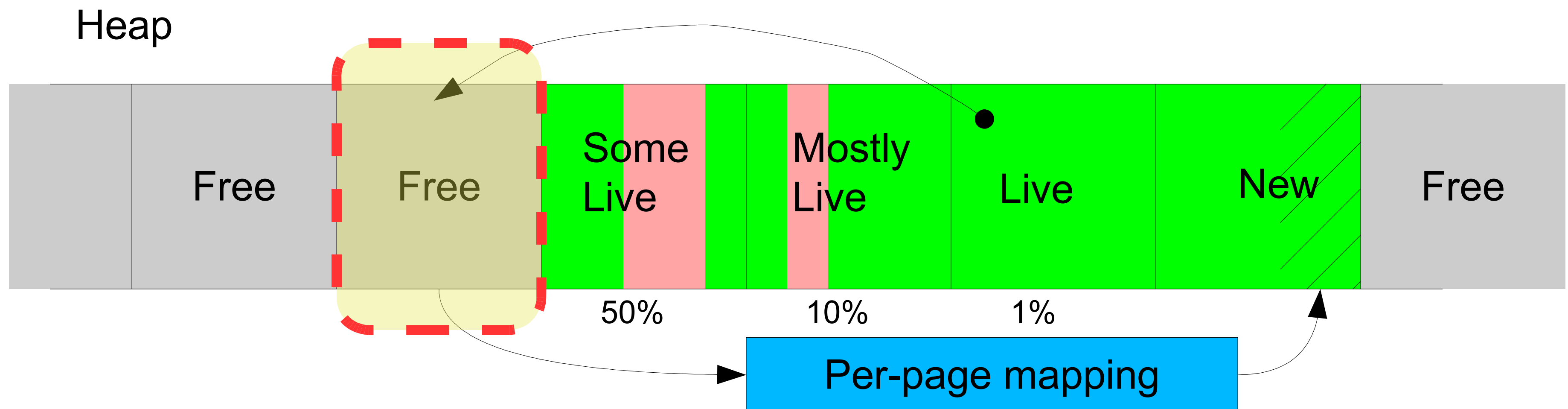
3. TLB-protect pages
4. **Copy live objects out**
   - **Leave dangling pointers for remap phase**
5. Free physical memory – OS immediately recycles

 Copyright © 2005, Azul Systems, Inc. All rights reserved.

Property of Azul Systems, Inc. not for redistribution

Relocation #5

www.azulsystems.com



4. Copy live objects out
   – Leave dangling pointers for remap phase
5. **Free physical memory – OS immediately recycles**
   – Mutator *read barrier* prevents propagating stale oop

 Copyright © 2005, Azul Systems, Inc. All rights reserved.

Property of Azul Systems, Inc. not for redistribution

Relocation and Self-Healing

www.azulsystems.com

- Mutator might load stale OOP to protected page
  - Then propagate it
- Instead, *read barrier* traps
  - Trap handler finds forwarding pointer, replaces in register
  - If object not yet copied,
    - Mutator copies it just like a GC thread would
- Loaded OOP is in Mutator working set
  - Mutator likely to load it again
  - Trap updates memory to avoid re-trapping
- “Self-Healing”
  - Once working set is relocated, no more trapping

 Copyright © 2005, Azul Systems, Inc. All rights reserved.
Property of Azul Systems, Inc. not for redistribution

Relocation Phase

- No “rush” to finish any relocation
  - Just run a little faster than allocation rate
- Relocated pages are mostly empty
  - Unlikely a mutator stalls on a page
  - Virtual memory is freed later, but we have lots of that
- Future work: could Relocate during next Mark phase
- Relocation needs valid mark bits
  - During Marking, mark bits are in flux
  - So use prior Mark phase bits
  - Need two sets of Mark bits

Copyright © 2005, Azul Systems, Inc. All rights reserved.
Property of Azul Systems, Inc. not for redistribution

Outline

www.azulsystems.com

- Major Phases
- Marking Phase
- Relocation phase
- **Remap phase**
- Current Status

 Copyright © 2005, Azul Systems, Inc. All rights reserved.

Property of Azul Systems, Inc. not for redistribution

Remap Phase

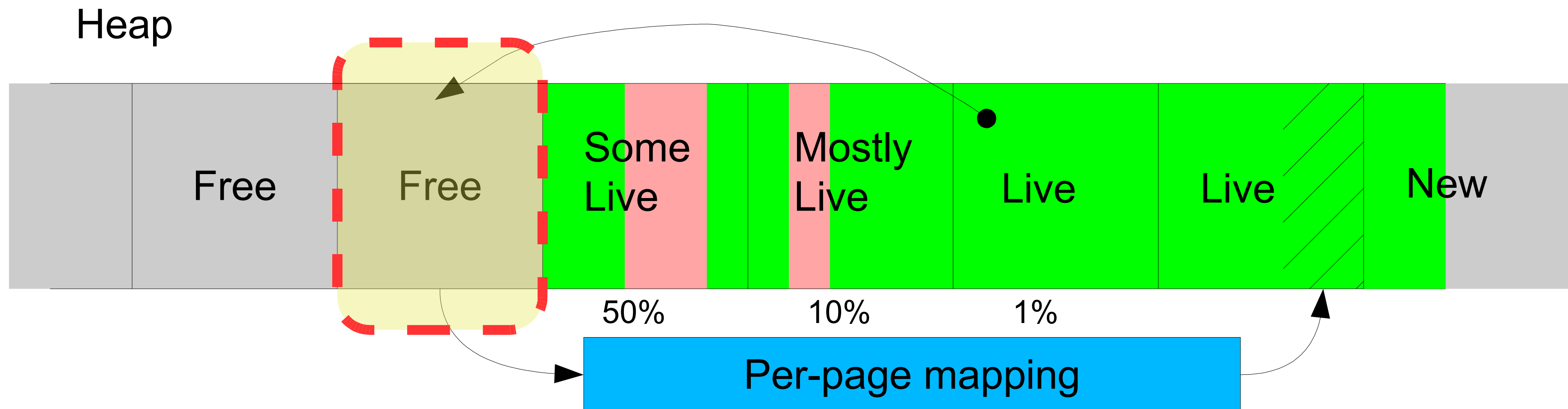
- Remap all OOPs to their forwarded locations
  - Physical memory is long gone
  - Forwarding pointers kept in side arrays
- Runs completely inside the next cycle's Mark phase
  - Or continuously via mutators' trap handlers
- Just execute a *read barrier* on each OOP
  - Will trap if not marked-through
  - Will trap if relocated
- At end no un-forwarded pointers remain
  - Deallocate virtual memory

 Copyright © 2005, Azul Systems, Inc. All rights reserved.

Property of Azul Systems, Inc. not for redistribution

Remapping

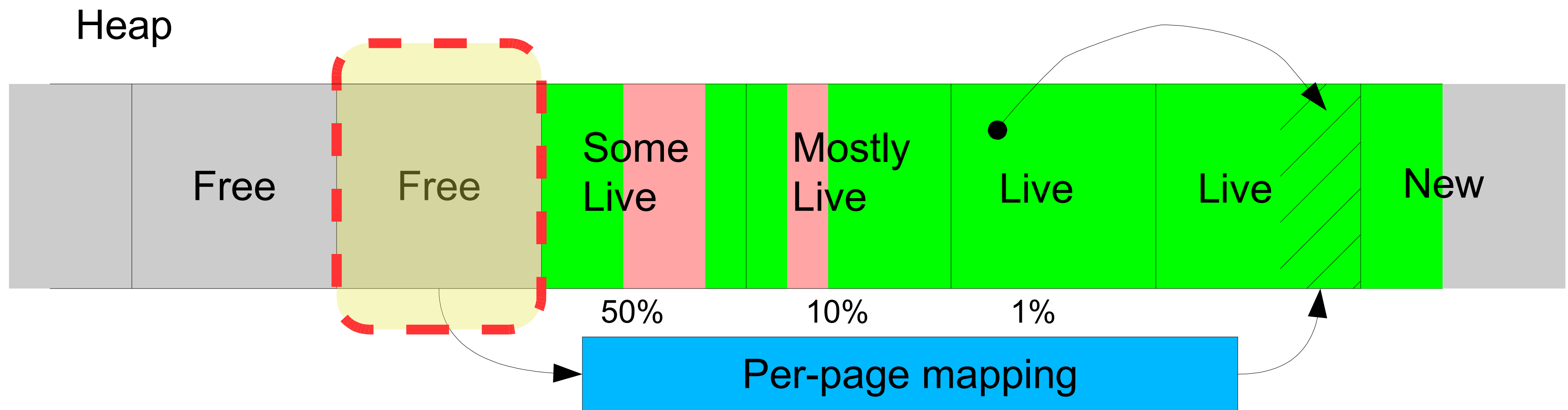
AZUL SYSTEMS



- **Either mutator or GC finds dangling pointer**
  - **Mutator does it via *read barrier* trap**

Copyright © 2005, Azul Systems, Inc. All rights reserved.
Property of Azul Systems, Inc. not for redistribution

Remapping

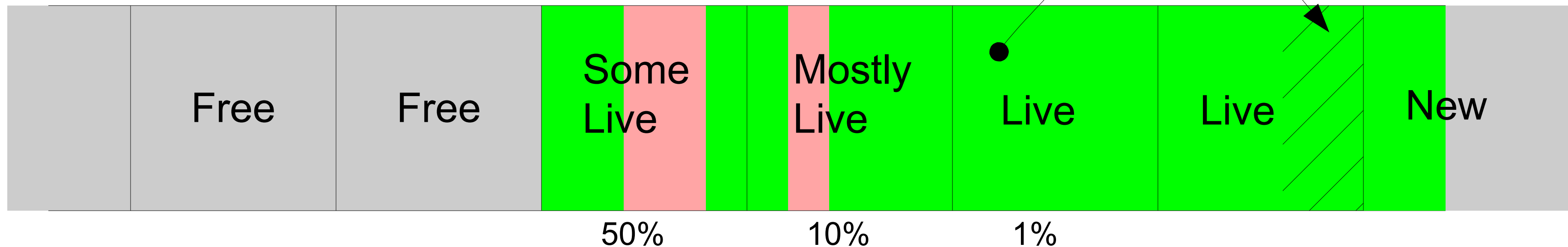


- Either mutator or GC finds dangling pointer
  - Mutator does it via TLB trap
- **Consult mapping, forward pointer**

 Copyright © 2005, Azul Systems, Inc. All rights reserved.
Property of Azul Systems, Inc. not for redistribution
www.azulsystems.com

Remapping

AZUL SYSTEMS



- Either mutator or GC finds dangling pointer
  - Mutator does it via TLB trap
- Consult mapping, forward pointer
- **At end, no dangling pointer**
- **Free virtual memory, side mappings**

Copyright © 2005, Azul Systems, Inc. All rights reserved.
Property of Azul Systems, Inc. not for redistribution

Outline

- Major Phases
- Marking Phase
- Relocation phase
- Remap phase
- **Current Status**

 Copyright © 2005, Azul Systems, Inc. All rights reserved.

Property of Azul Systems, Inc. not for redistribution

Some preliminary data

- 24 warehouse JBB with 4Gig heap, 20min run
- GC keeps up with 200Meg/sec allocation rate
- Max pause time: 37ms
- 3 STW pauses in current implementation
  - Initial marking STW (clear marks, flip roots)
  - Final marking STW (finish marks, mark code, sys dict, etc)
  - Relocation STW (TLB protect, relocate roots)
  - All have solutions worked out, being implemented

 Copyright © 2005, Azul Systems, Inc. All rights reserved. Property of Azul Systems, Inc. not for redistribution

Trap "Storms"

- After a desired-NMT flip or TLB protection
  - Mutators start trapping
- Mutators slow down in a "trap storm"
- But slowdown currently not measurable
  - Gross pause times dominate
  - No ticks in the profiler
- See ~1000 traps @ ~1000 clks each per "storm"
  - This is a tiny slowdown
  - Will need very low STW pauses to see effect

 Copyright © 2005, Azul Systems, Inc. All rights reserved. Property of Azul Systems, Inc. not for redistribution

Read Barriers

www.azulsystems.com

- Expensive in Software*
- So done in hardware!
- 1 extra instruction per Load
- "Looks like" a dead load but
  - with fancy TLB mappings
  - no L1 pollution on a hit
  - No trap on a NULL
  - fast user-mode trap handler
- Mutator thread pays load-use penalty
  - compiler can schedule around it

*Blackburn & Hosking, 2004, "Barriers: Friend or Foe," conditional read barriers cost 7-21%

Copyright © 2005, Azul Systems, Inc. All rights reserved.

Property of Azul Systems, Inc. not for redistribution

Future Work

- Generational version is planned
  - Will use *write-barrier* op
  - Includes cross-generation check
- Remove expensive STW pauses
  - Actually most STWs already meet goal of OS timeslice
- General tuning work
  - It's all in the implementation details
  - No interesting algorithmic work

Copyright © 2005, Azul Systems, Inc. All rights reserved.
Property of Azul Systems, Inc. not for redistribution

Legalese

www.azulsystems.com

AZUL SYSTEMS

My lawyer made me say this....

"Azul Systems, Azul, and the Azul arch logo are trademarks of Azul Systems, Inc. in the United States and other countries. Sun, Sun Microsystems, Java and all Java based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. Other marks are the property of their respective owners and are used here only for identification purposes."

Copyright © 2005, Azul Systems, Inc. All rights reserved.

Property of Azul Systems, Inc. not for redistribution

Q & A

Q & A

 Copyright © 2005, Azul Systems, Inc. All rights reserved.

Property of Azul Systems, Inc. not for redistribution