A Third Generation Smalltalk-80 Implementation

Patrick J. Caudill

Allen Wirfs-Brock

Computer Research Laboratory

Tektronix Laboratories

Tektronix Large Object Space Smalltalk

- High performance bytecode interpreter for 68020.
- Supports a very large number of objects.
- Regularized virtual machine design.

- Does not use an Object Table.
- Generation Scavenging based garbage collector.

- Commercial Product – Tek 4405/4406 Smalltalk.

First Generation Implementation

1981

- Part of Xerox sponsored Smalltalk evaluation project.
- "By the book" interpreter design.
- 16-bit oop, (32k object space).
- Implemented in Pascal for 68000.

- Performance approximately 2% of Dorodo:

  Extreme memory management overhead inherent in virtual machine design.

  Representational mis-matches.

  Unable to optimally utilize host machine resources.

Second Generaton Implementation.

1982–1983

- Goal : "Usable" level of performance.
- Assembly language based implementation for 68000/68010.
- Completely faithful to blue book specification (16-bit oops).
- Storage management overhead minimized via a mixed strategy collection scheme:

  Multiple context representations with stack based allocation.
  Deferred reference counting.
  Special allocation zone.
  Backup mark/sweep collector.

- Results:

  Performance 25% of Dorodo.

  Widely used.

  Evolved into commercial product, Tektronix 4404 Smalltalk.

User Reaction to 4404 Smalltalk

- Excellent tool for developing prototypes.
- Desire to use Smalltalk for production applications.
- 32,000 Object limit major impediment to the development of large applications.
- Performance generally adequate, "but if it was faster I could ..."

Design Problems

- Interactions between various storage management strategies very difficult to understand and maintain.
- Virtual machine design irregularities made it difficult to experiment with language extensions and new implementation techniques.

Goals for a Third Generation Implementation (1985)

- Provide support for a large number of objects.
- Increase performance, relative to existing interpreter.
- Remove virtual machine irregularities due to limited object space.
- Preserve basic semantics and functionality of virtual machine and image.

- Minimize implementation time.
- Maximize maintainability and extensibility.

Fundamental Design Decisions

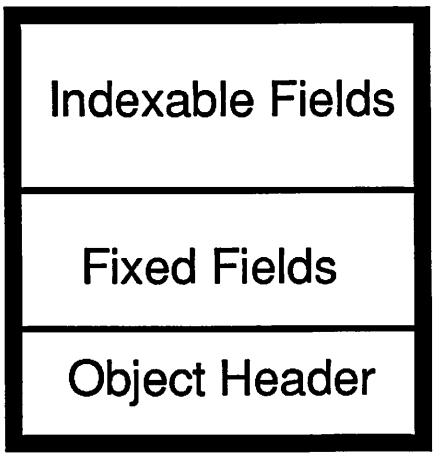
- Bytecode interpreter.
- 32-bit oops and very large objects.
- Generation Scavenging based storage management.
- No Object Table.
- New representation for CompiledMethods and other representational changes.
- Image changes as needed to accommodate new design.

Object Memory Architecture

Tag	Parameter
0	Virtual address of object
1	31-bit 2's complement integer

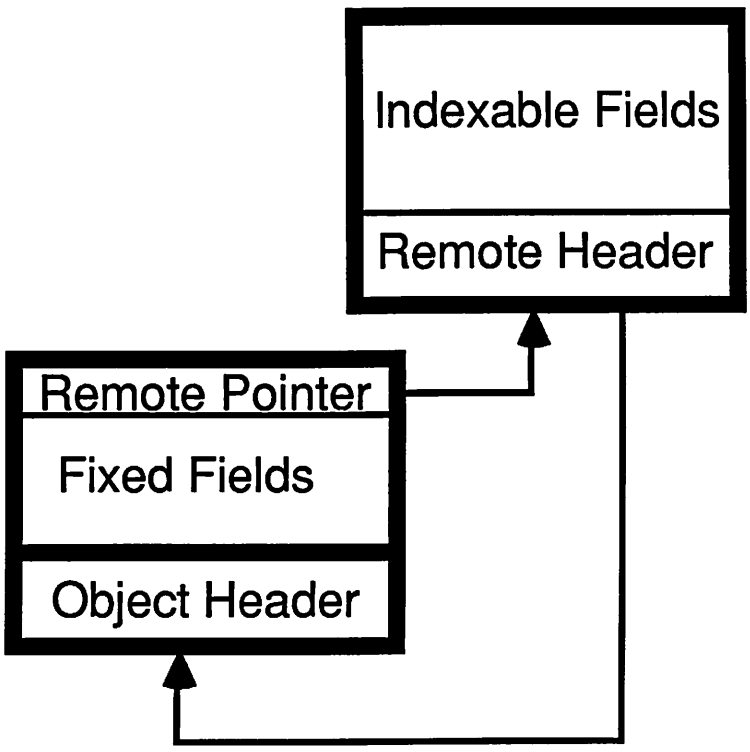
An OOP

Object Memory Architecture



Object Format
(Normal Indexable)

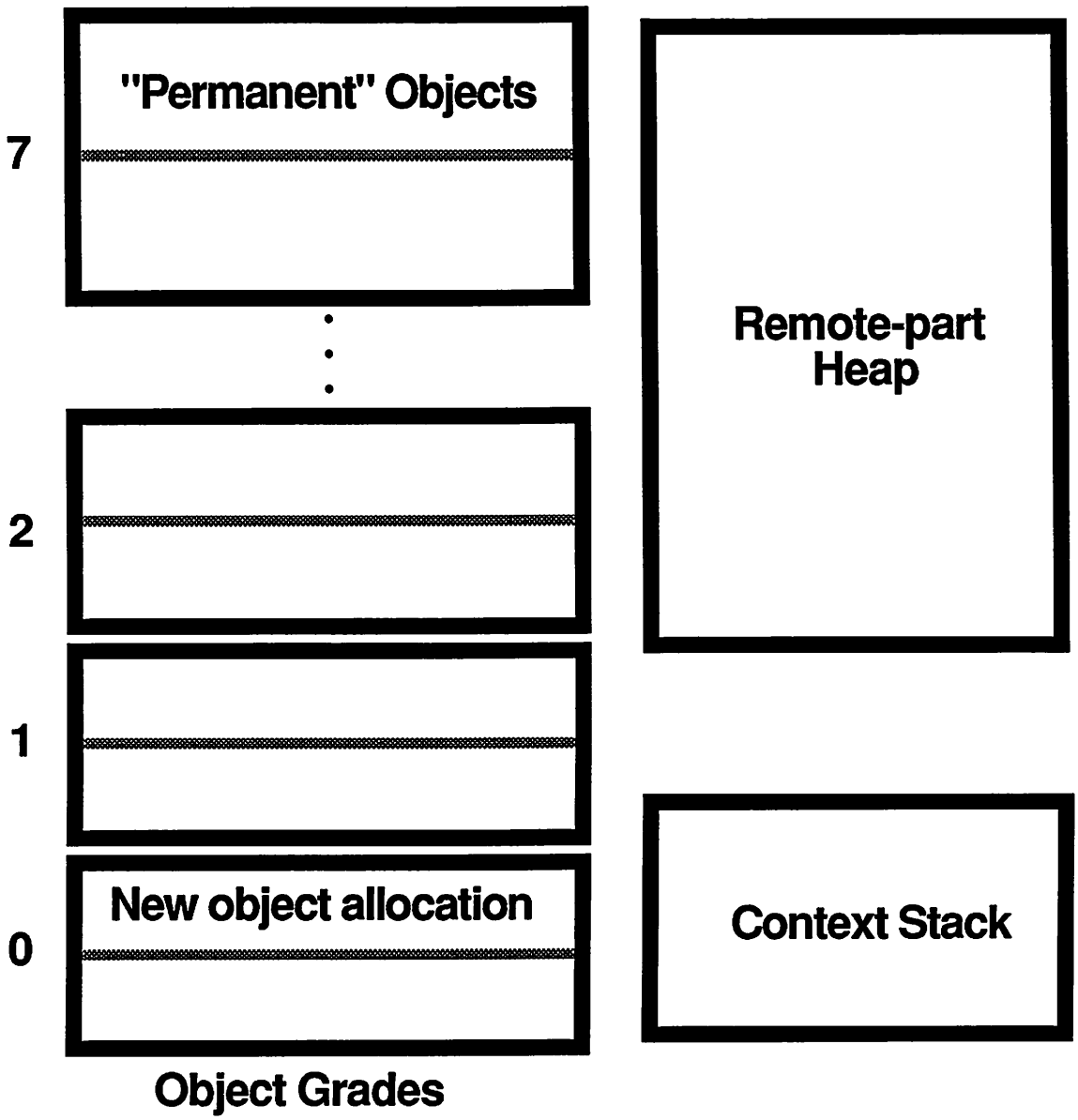
Object Memory Architecture



Object Format

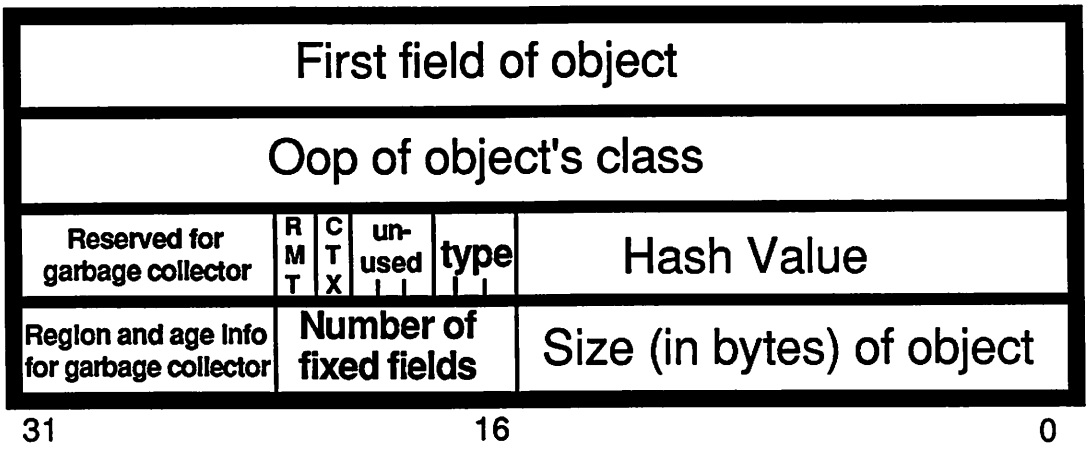
(Remote Indexable)

Object Memory Architecture



Storage Regions

Object Memory Architecture



CTX – 1 if this is a context object
RMT – 1 if this object has a remote indexable part

Type - 0 normal (non-indexable)
1 byte indexable
2 word (16-bit) indexable
3 long (32-bit) indexable
4 pointer indexable

Object Header Format

CompiledMethod Representation

Problems with CompiledMethods:

Violates object memory design rules by combining object references and binary data.

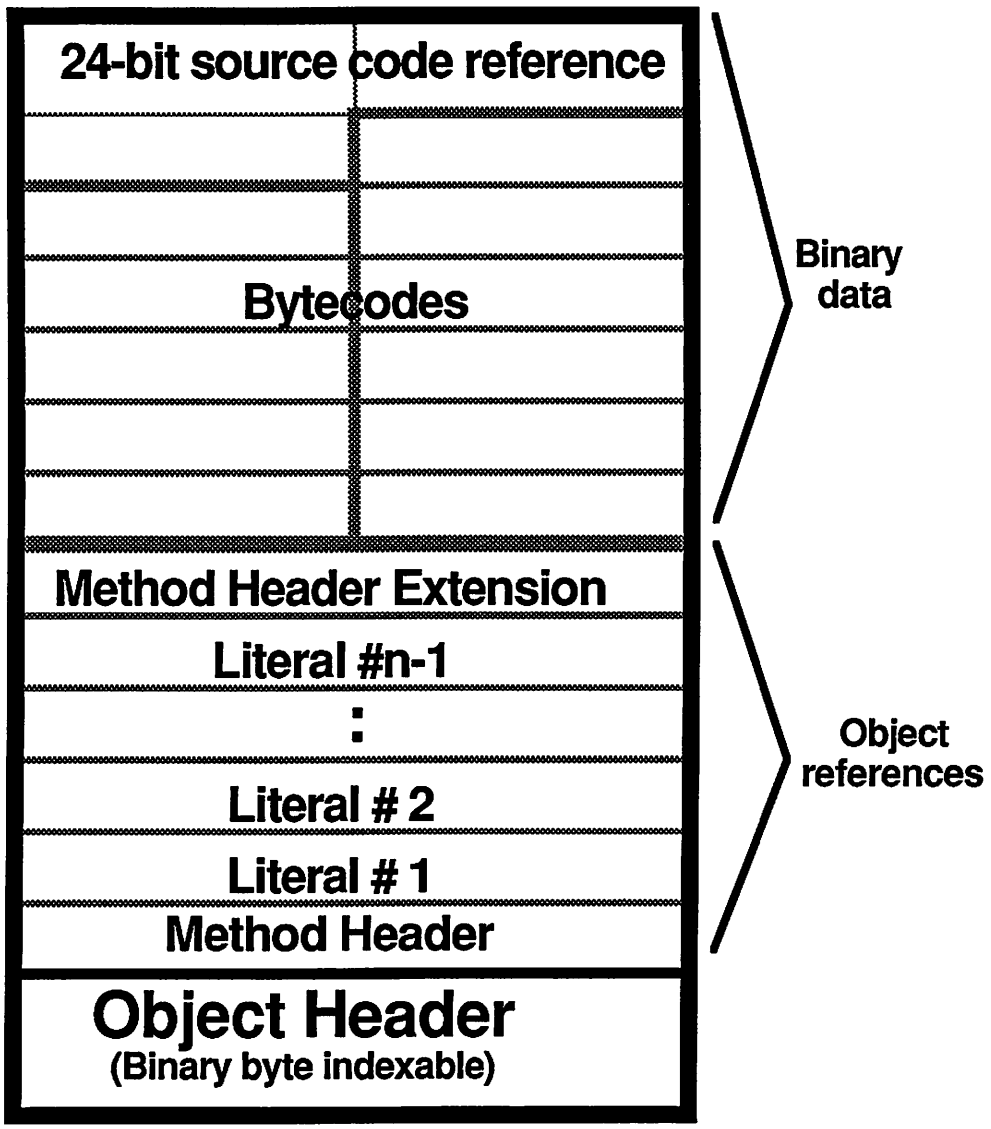
Requires special case code and special primitives to support.

High decoding overhead.

Cannot be subclassed.

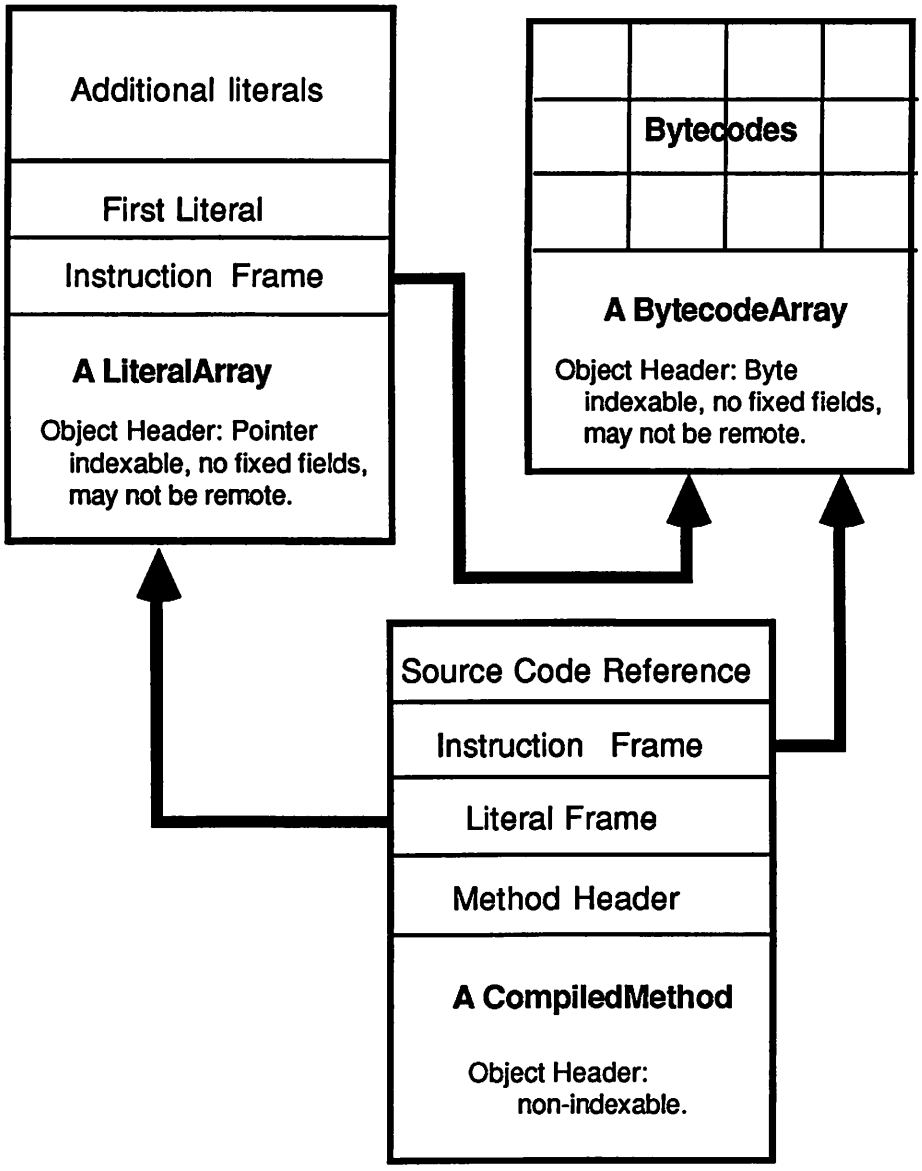
Inflexible source code reference.

CompiledMethod Representation



"Blue-book" design

CompiledMethod Representation



New design

Performance

Approximately 50% faster than our 2nd generation system when executing on same hardware.

Benchmarks at 95% of Dorodo.

Standard image has over 33,000 objects.

Images containing 60,000+ objects commonly used.

Statistics

100K+ bytes allocated per second.

~0.5% Reach grade 1.

~95% are never copied.

~3% of execution time spent scavenging.

Conclusions

Proves effectiveness of generation based scavenging and direct pointers for a high performance Smalltalk implementation.

Interpretation is a practical alternative for high performance Smalltalk implementations.

Begins to eliminate the small object / object spaces bias of Smalltalk system.