Allocation Folding Based on Dominance

Daniel Clifford, Hannes Payer, Michael Starzinger and Ben L. Titzer

International Symposium on Memory Management (ISMM) 2014

presented by Thomas Hütter

Concurrency & Memory Management Seminar 2015

Univ.- Prof. Dr. Christoph Kirsch

Department of Computer Sciences
University of Salzburg

Allocation Folding Based on Dominance

Daniel Clifford, Hannes Payer, Michael Starzinger and Ben L. Titzer

International Symposium on Memory Management (ISMM) 2014

presented by Thomas Hütter

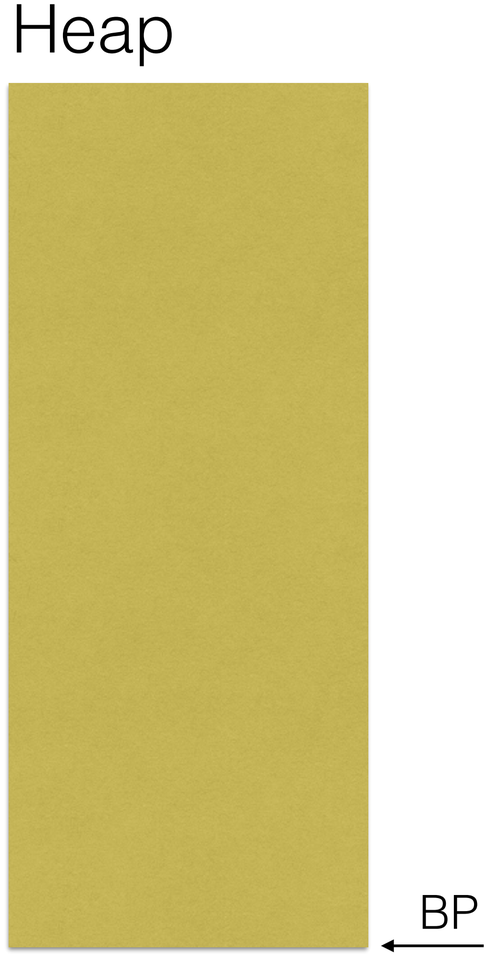
Concurrency & Memory Management Seminar 2015

Univ.- Prof. Dr. Christoph Kirsch

Department of Computer Sciences
University of Salzburg

Dynamic Memory Allocation

- Bump pointer allocation
  - Pointer (BP) marks end of last allocation

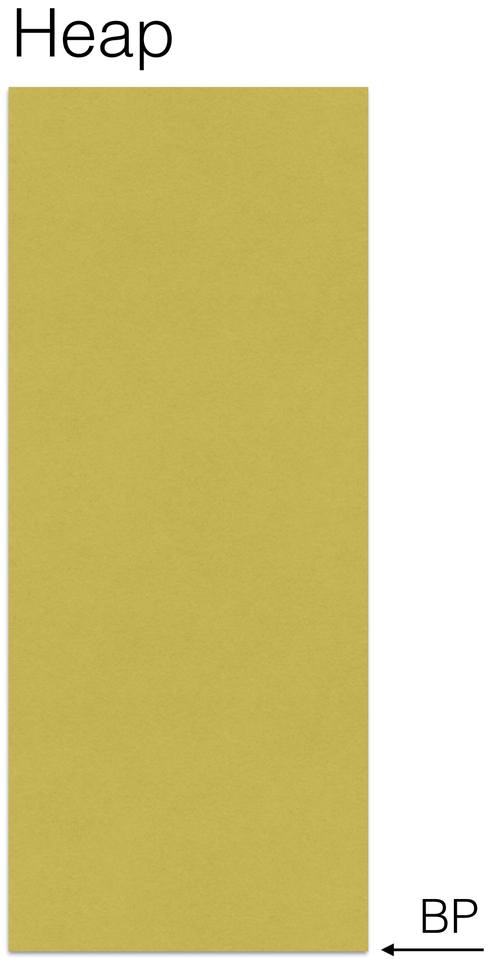


Dynamic Memory Allocation

- Bump pointer allocation
  - Pointer (BP) marks end of last allocation

- Example

```
void *p1 = malloc(sizeof(int));
```

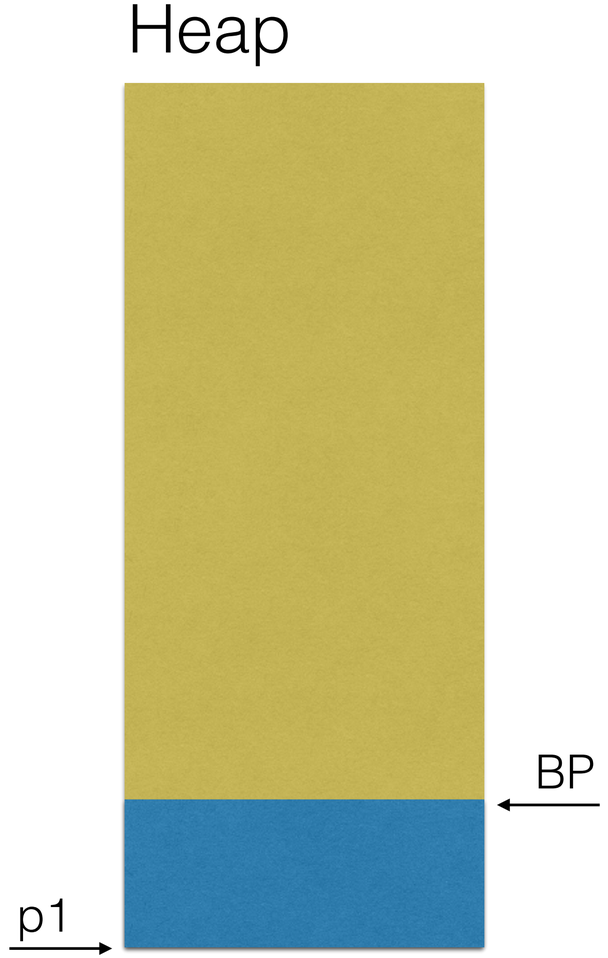


Dynamic Memory Allocation

- Bump pointer allocation
  - Pointer (BP) marks end of last allocation

- Example

```
void *p1 = malloc(sizeof(int));
```

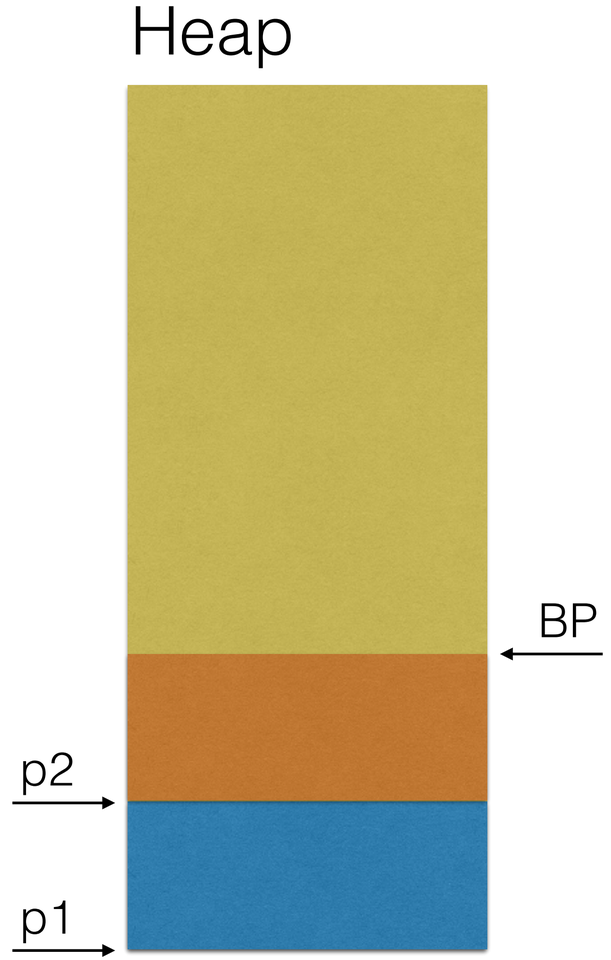


Dynamic Memory Allocation

- Bump pointer allocation
  - Pointer (BP) marks end of last allocation

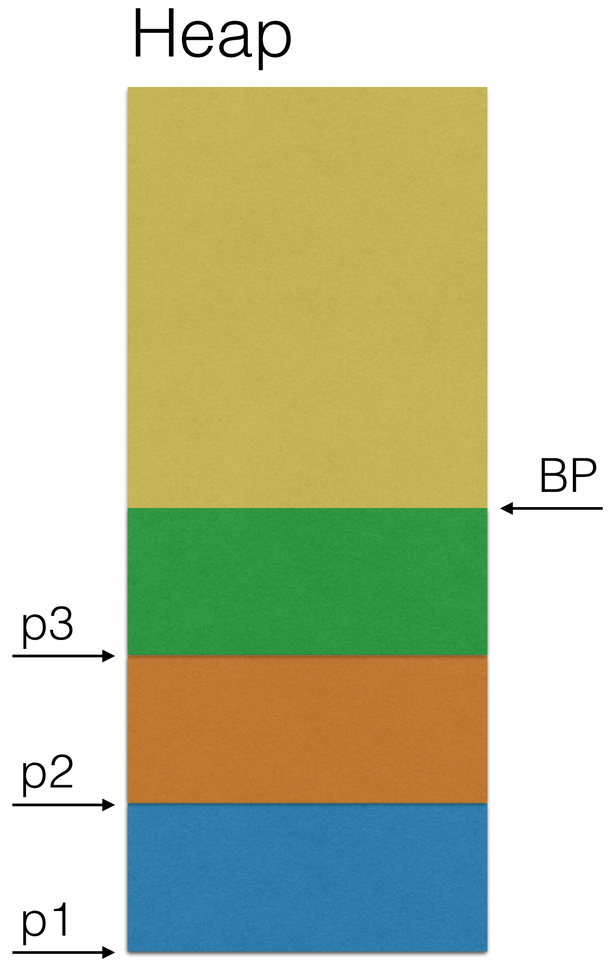
- Example

```
void *p1 = malloc(sizeof(int));
void *p2 = malloc(sizeof(int));
```



Dynamic Memory Allocation

- Bump pointer allocation
  - Pointer (BP) marks end of last allocation

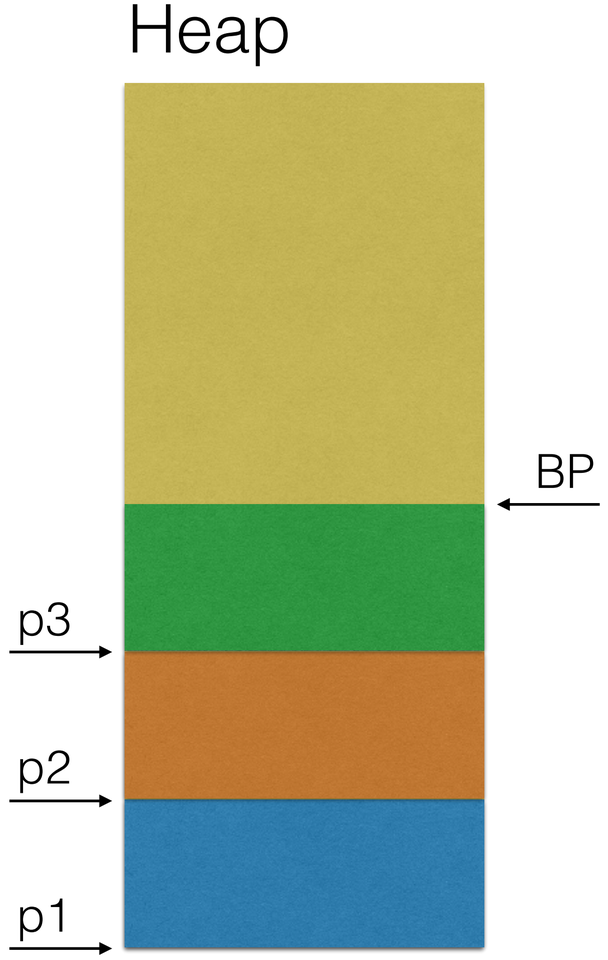


- Example

```
void *p1 = malloc(sizeof(int));
void *p2 = malloc(sizeof(int));
void *p3 = malloc(sizeof(int));
```

Dynamic Memory Allocation

- Bump pointer allocation
  - Pointer (BP) marks end of last allocation

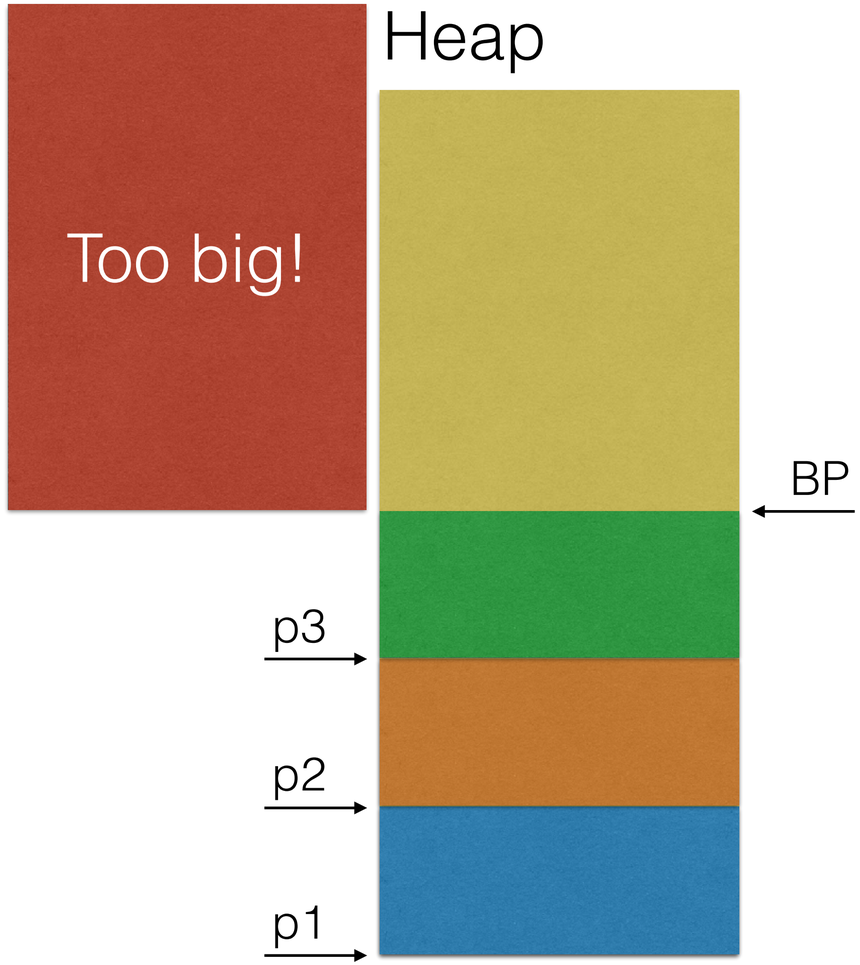


- Example

```
void *p1 = malloc(sizeof(int));
void *p2 = malloc(sizeof(int));
void *p3 = malloc(sizeof(int));
void *p4 = malloc(4*sizeof(int));
```

Dynamic Memory Allocation

- Bump pointer allocation
  - Pointer (BP) marks end of last allocation



- Example

```
void *p1 = malloc(sizeof(int));
void *p2 = malloc(sizeof(int));
void *p3 = malloc(sizeof(int));
void *p4 = malloc(4*sizeof(int));
```

Dynamic Memory Allocation

BOUND CHECKS

Dynamic Memory Allocation

BOUND CHECKS

Performance Overhead

Allocation Folding Based on Dominance

Daniel Clifford, Hannes Payer, Michael Starzinger and Ben L. Titzer

International Symposium on Memory Management (ISMM) 2014

presented by Thomas Hütter

Concurrency & Memory Management Seminar 2015

Univ.- Prof. Dr. Christoph Kirsch

Department of Computer Sciences
University of Salzburg

Allocation Folding

- Group allocations together

- Example

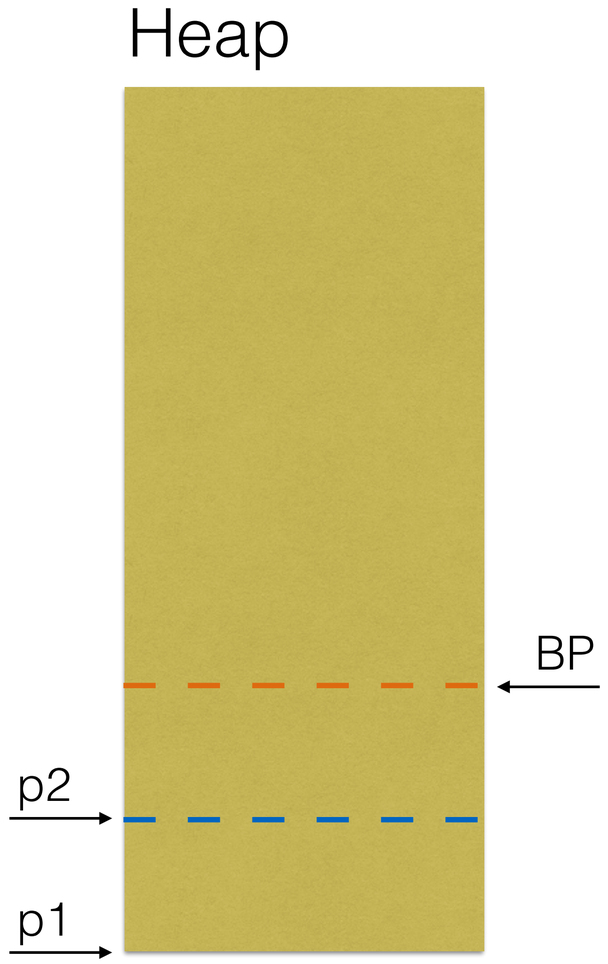
```
void *p1 = malloc(sizeof(int));
[…]
void *p2 = malloc(sizeof(int));
```

Allocation Folding

- Group allocations together

- Example

```
void *p1 = malloc(sizeof(int));
[…]
void *p2 = malloc(sizeof(int));
```

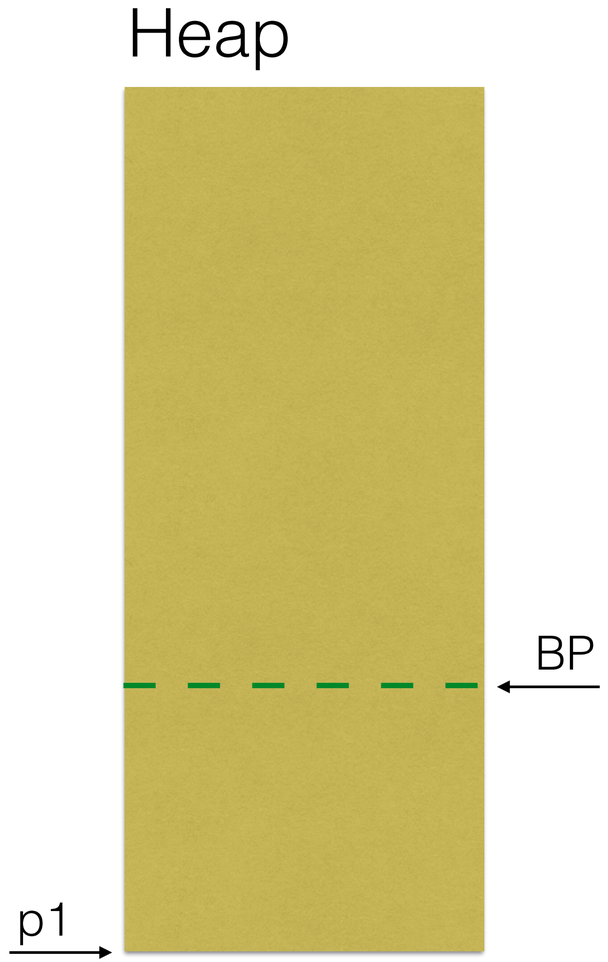


Allocation Folding

- Group allocations together

- Example

```
void *p1 = malloc(sizeof(int));
[…]
void *p2 = malloc(sizeof(int));
```

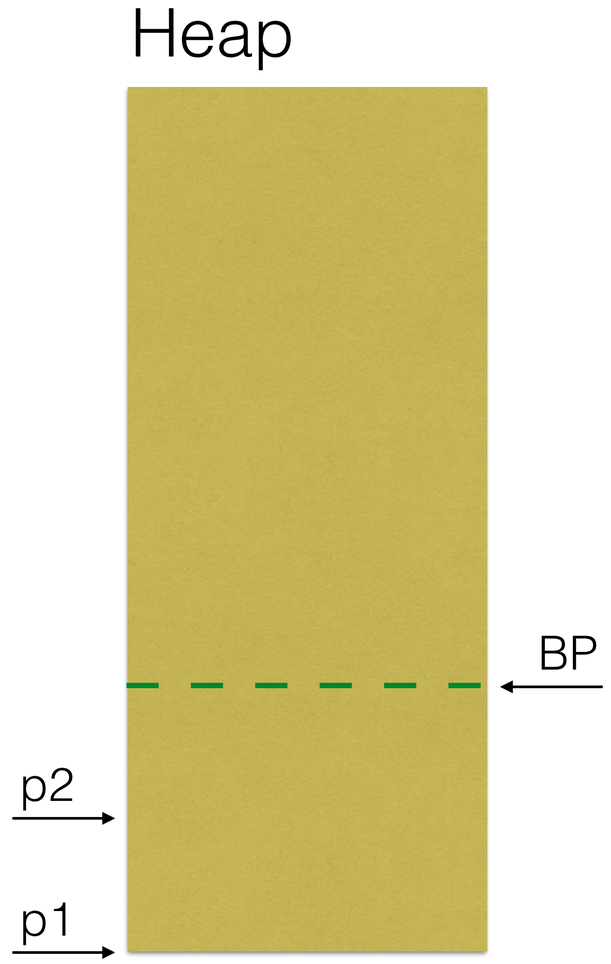


Allocation Folding

- Group allocations together

- Example

```
void *p1 = malloc(sizeof(int));
[…]
void *p2 = malloc(sizeof(int));
```

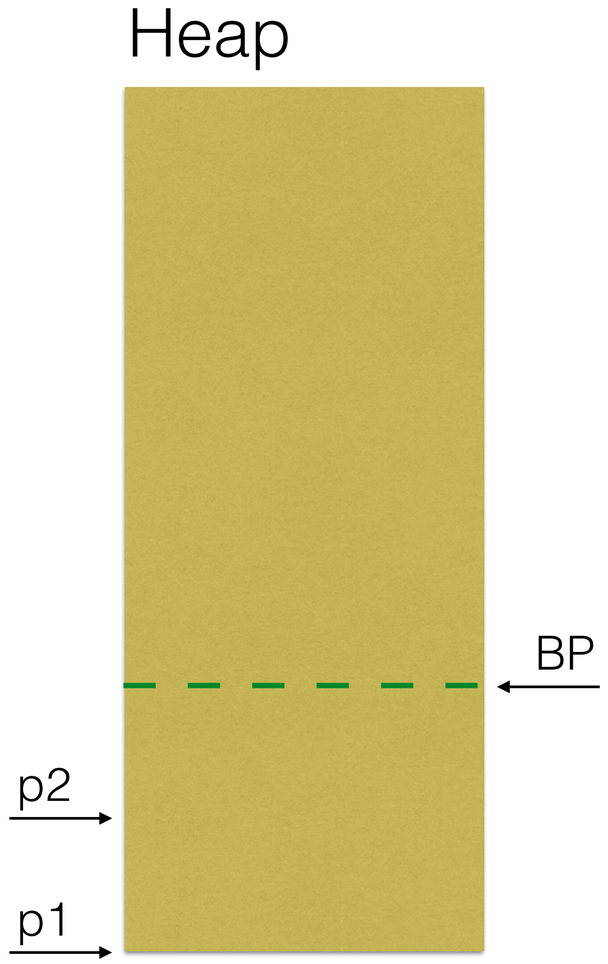


Allocation Folding

- Group allocations together

- Example

```
void *p1 = malloc(sizeof(int));
[…]    //be careful!
void *p2 = malloc(sizeof(int));
```



Invariant 1

Between two allocations A_1 and A_2, if no other operation that can move the object allocated at A_1 occurs, then space for the object allocated at A_2 could have been allocated at A_1 and then initialized at A_2, without ever having been observable to the garbage collector.

V8

- JavaScript virtual machine
- Open Source[1]
- C++
- Used in Google Chrome

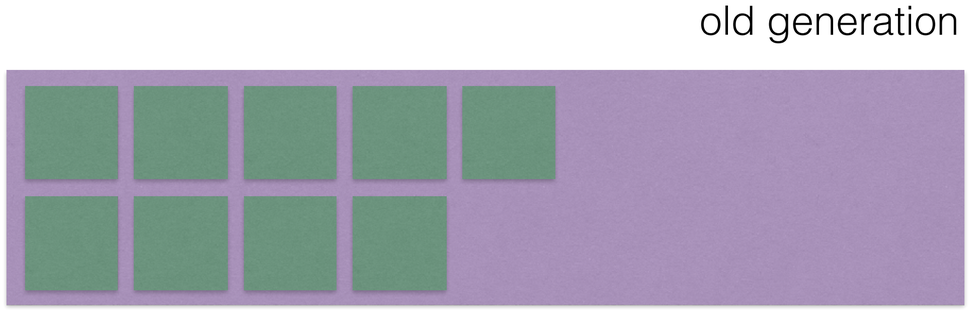
[1] https://code.google.com/p/v8

Garbage Collector

- Generational
  - New generation
    - Semi-space strategy
    - Scavenger
  - Old generation
    - Mark and sweep collector
      - Depth-first search
      - 3 colors

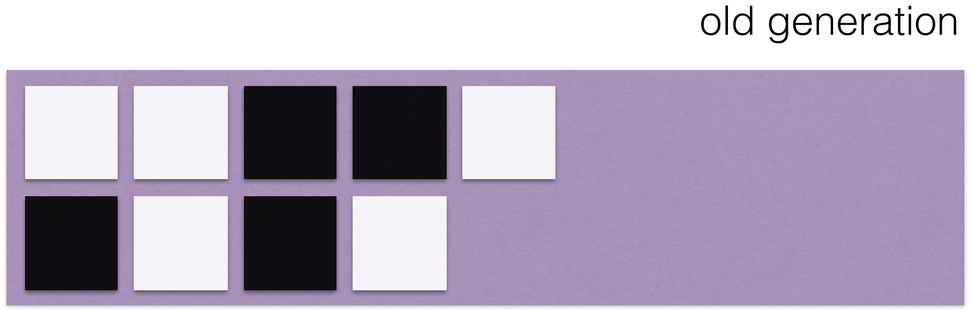
Garbage Collector

- Generational
  - New generation
    - Semi-space strategy
    - Scavenger
  - Old generation
    - Mark and sweep collector
      - Depth-first search
      - 3 colors



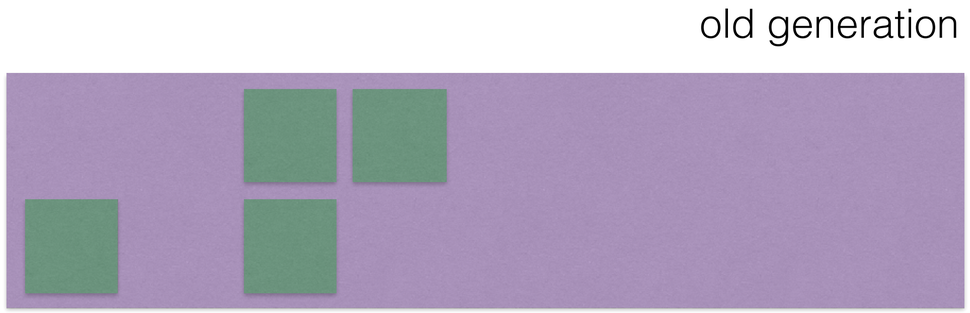
Garbage Collector

- Generational
  - New generation
    - Semi-space strategy
    - Scavenger
  - Old generation
    - Mark and sweep collector
      - Depth-first search
      - 3 colors



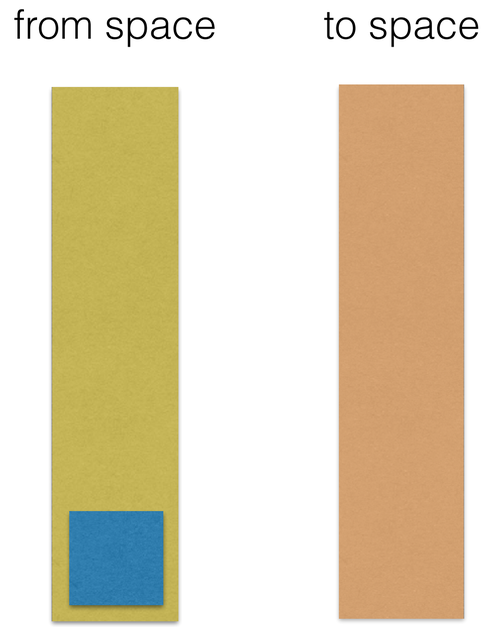
Garbage Collector

- Generational
  - New generation
    - Semi-space strategy
    - Scavenger
  - Old generation
    - Mark and sweep collector
      - Depth-first search
      - 3 colors



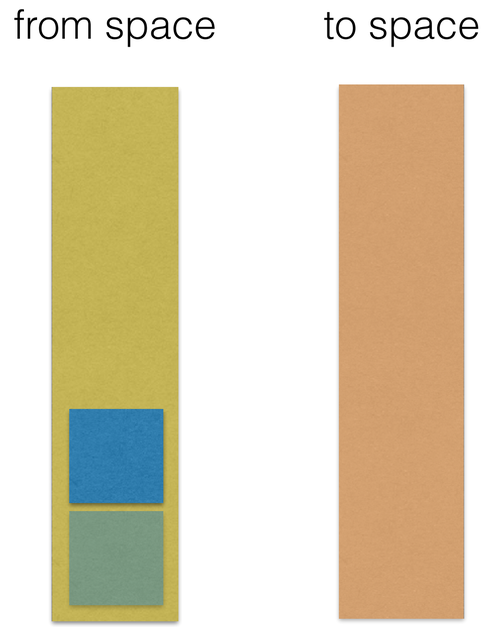
Garbage Collector

- Generational
  - New generation
    - Semi-space strategy
    - Scavenger
  - Old generation
    - Mark and sweep collector
      - Depth-first search
      - 3 colors



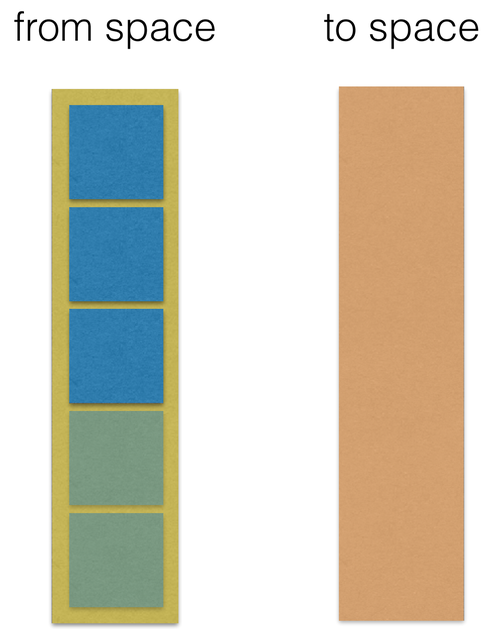
Garbage Collector

- Generational
  - New generation
    - Semi-space strategy
    - Scavenger
  - Old generation
    - Mark and sweep collector
      - Depth-first search
      - 3 colors



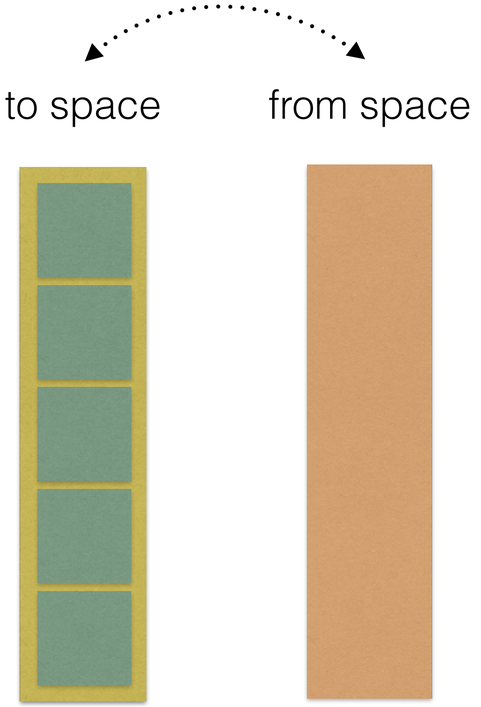
Garbage Collector

- Generational
  - New generation
    - Semi-space strategy
    - Scavenger
  - Old generation
    - Mark and sweep collector
      - Depth-first search
      - 3 colors



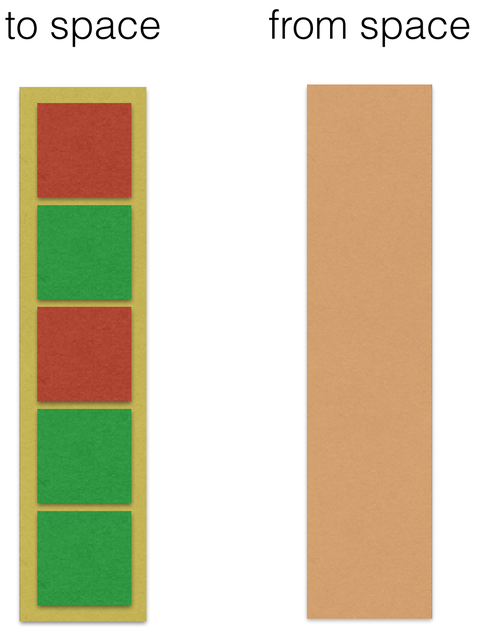
Garbage Collector

- Generational
  - New generation
    - Semi-space strategy
    - Scavenger
  - Old generation
    - Mark and sweep collector
      - Depth-first search
      - 3 colors



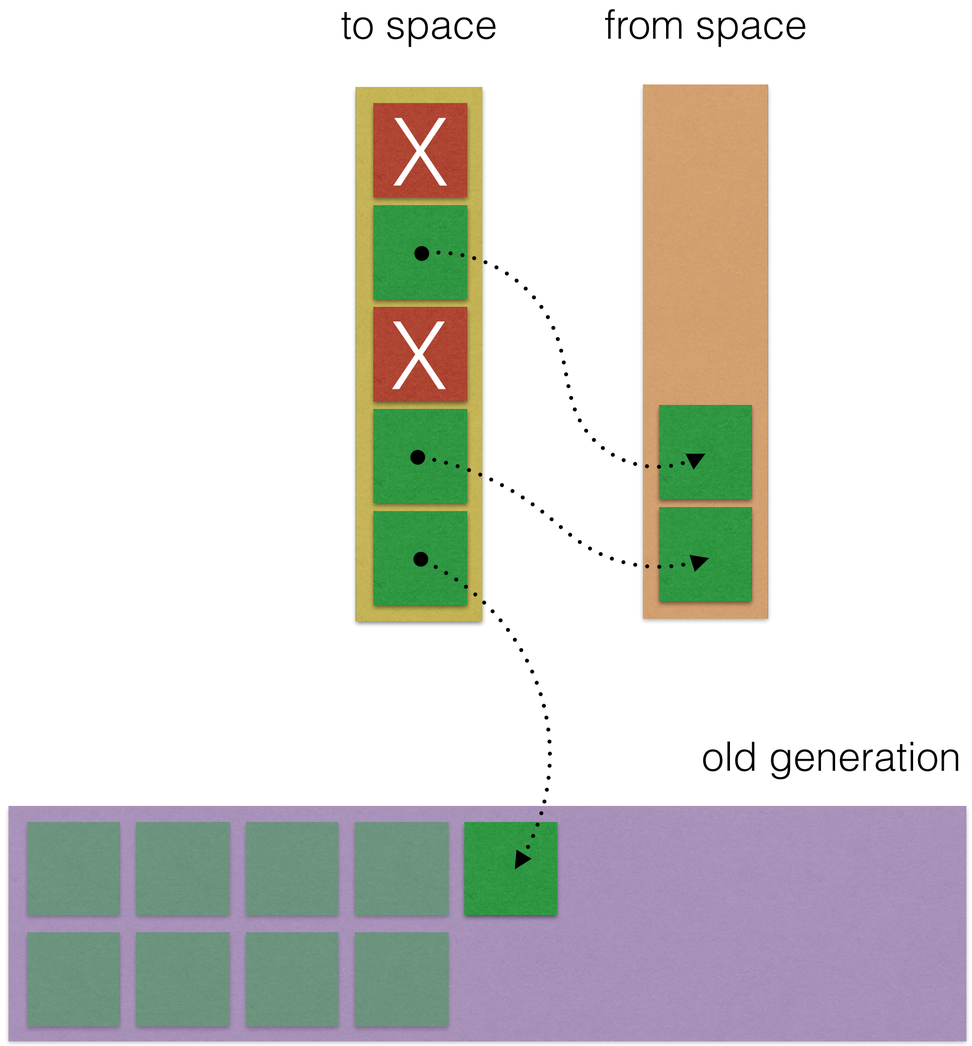
Garbage Collector

- Generational
  - New generation
    - Semi-space strategy
    - Scavenger
  - Old generation
    - Mark and sweep collector
      - Depth-first search
      - 3 colors



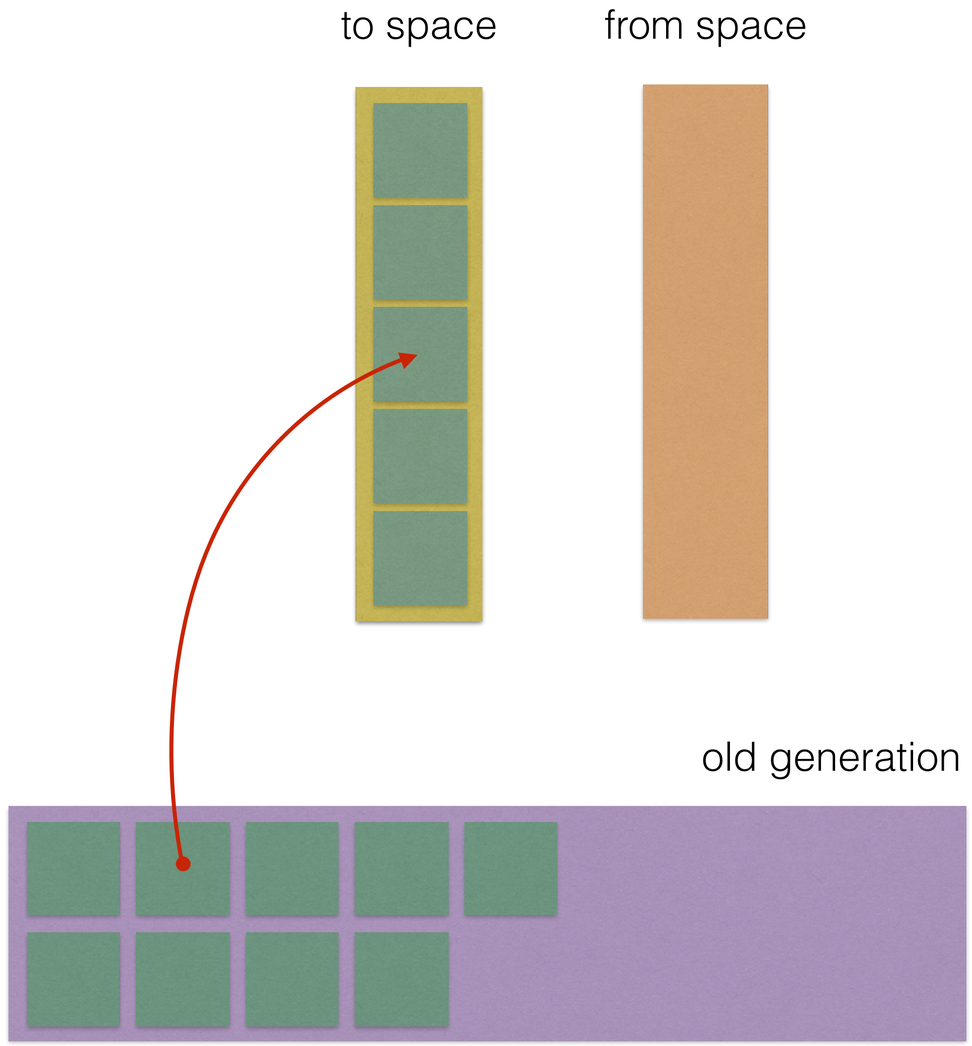
Garbage Collector

- Generational
  - New generation
    - Semi-space strategy
    - Scavenger
  - Old generation
    - Mark and sweep collector
      - Depth-first search
      - 3 colors



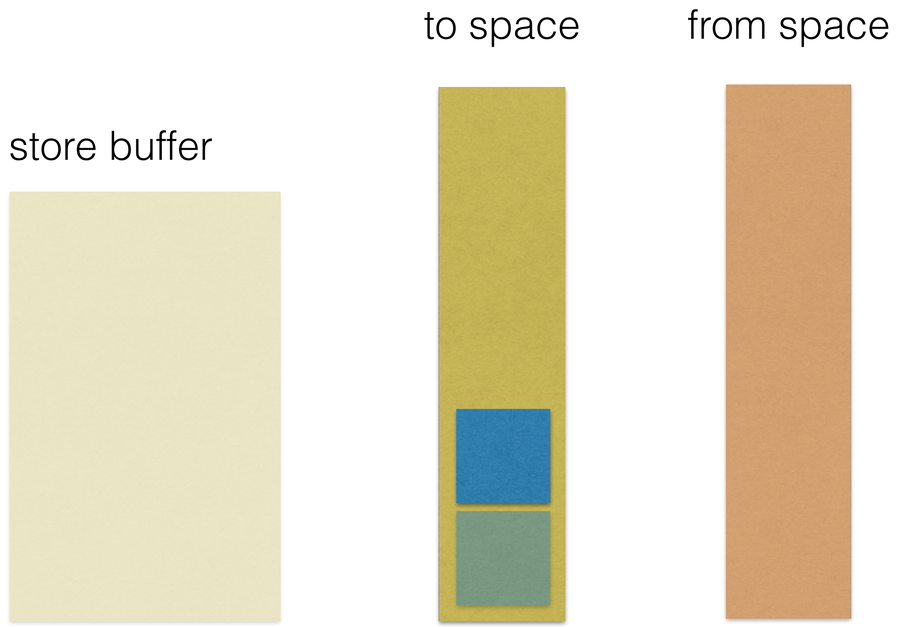
Garbage Collector

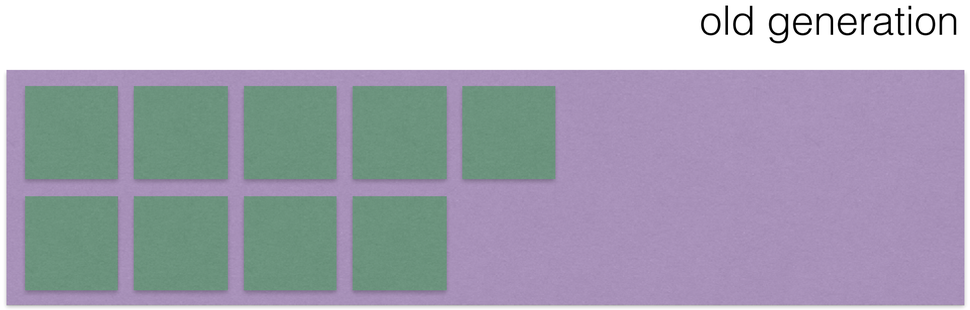
- Generational
  - New generation
    - Semi-space strategy
    - Scavenger
  - Old generation
    - Mark and sweep collector
      - Depth-first search
      - 3 colors



Garbage Collector

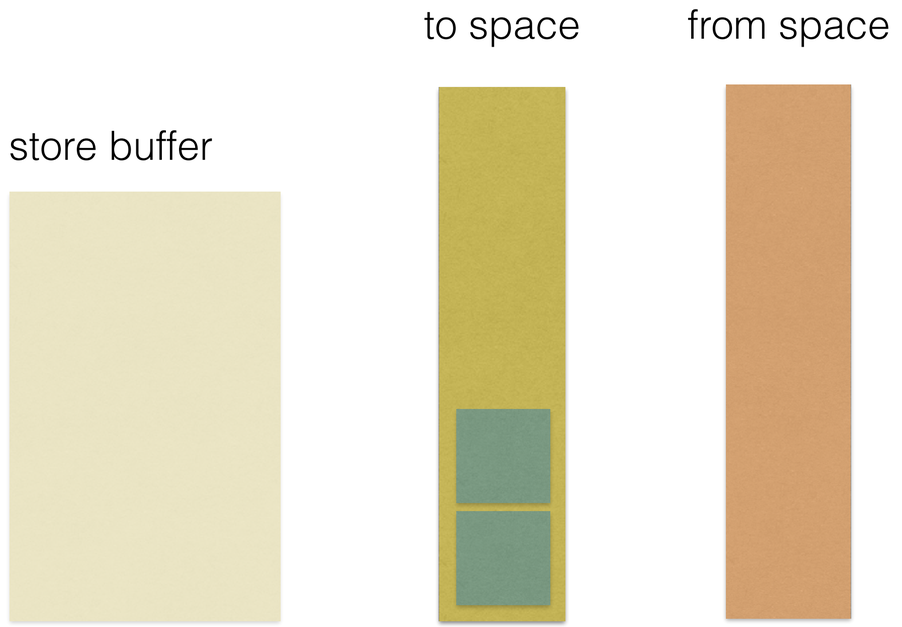
- Generational
  - New generation
    - Semi-space strategy
    - Scavenger
  - Old generation
    - Mark and sweep collector
      - Depth-first search
      - 3 colors
- Write barriers
  - Store buffer

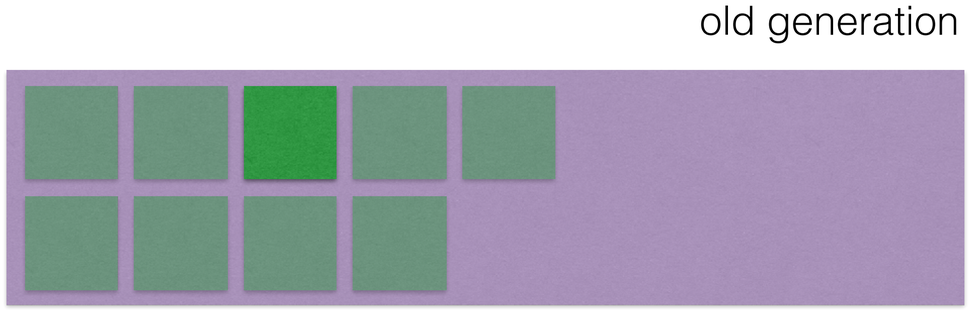




Garbage Collector

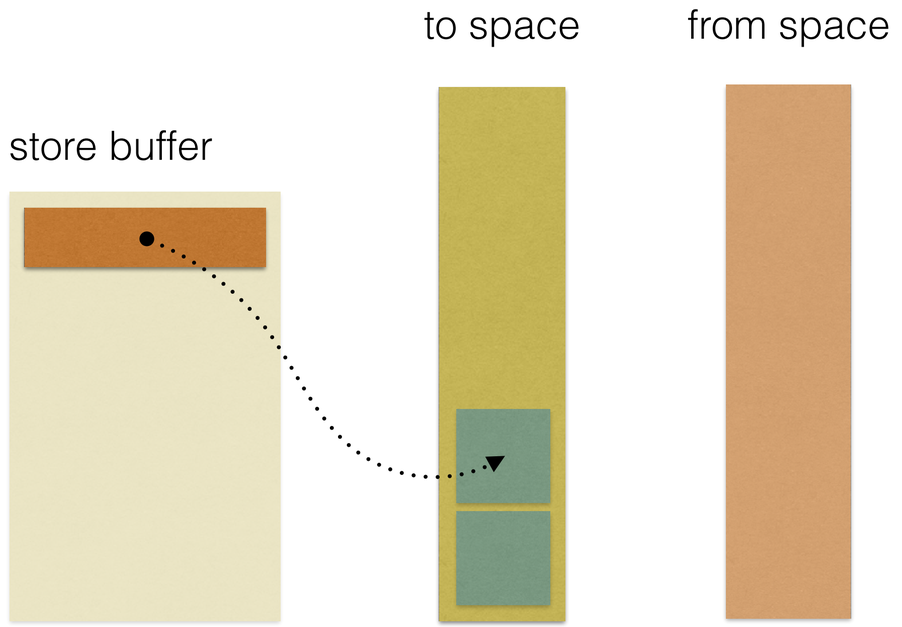
- Generational
  - New generation
    - Semi-space strategy
    - Scavenger
  - Old generation
    - Mark and sweep collector
      - Depth-first search
      - 3 colors
- Write barriers
  - Store buffer

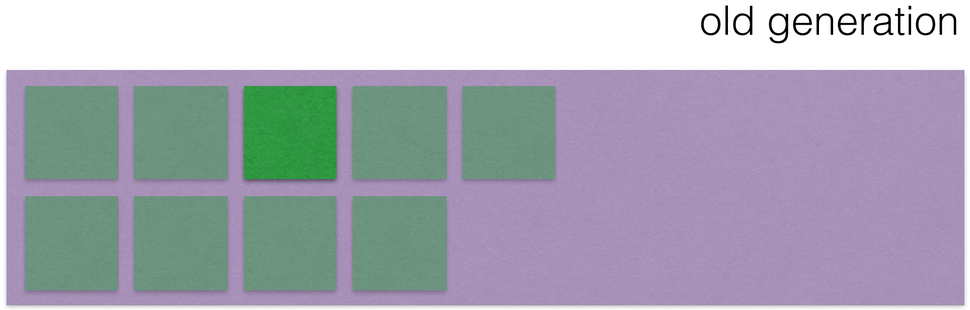




Garbage Collector

- Generational
  - New generation
    - Semi-space strategy
    - Scavenger
  - Old generation
    - Mark and sweep collector
      - Depth-first search
      - 3 colors
- Write barriers
  - Store buffer





Compilers

- Full compiler
  - Compiles everything
  - Unoptimized code
- Optimizing Compiler (Crankshaft)
  - Intermediate representation (IR)
  - Several compiler optimizations
    - Global value numbering (GVN)

Compilers

- Full compiler
  - Compiles everything
  - Unoptimized code
- Optimizing Compiler (Crankshaft)
  - Intermediate representation (IR)
  - Several compiler optimizations
    - Global value numbering (GVN)

Crankshaft IR Instruction	Dep	Chg
I_n = PARAMETER[K]		
I_n = CONSTANT[K]		
I_n = ARITH(I, I)		
I_n = LOAD[field](object)	Ψ	
I_n = STORE[field](object, value)		Ψ
I_n = ALLOC[space](size)	Λ	Λ
I_n = INNER[offset, size](alloc)		
I_n = CALL(I…)	*	*
I_n = PHI(I…)		
…		

Table 1: Simplified Crankshaft IR Instructions

Crankshaft IR (*Example*)

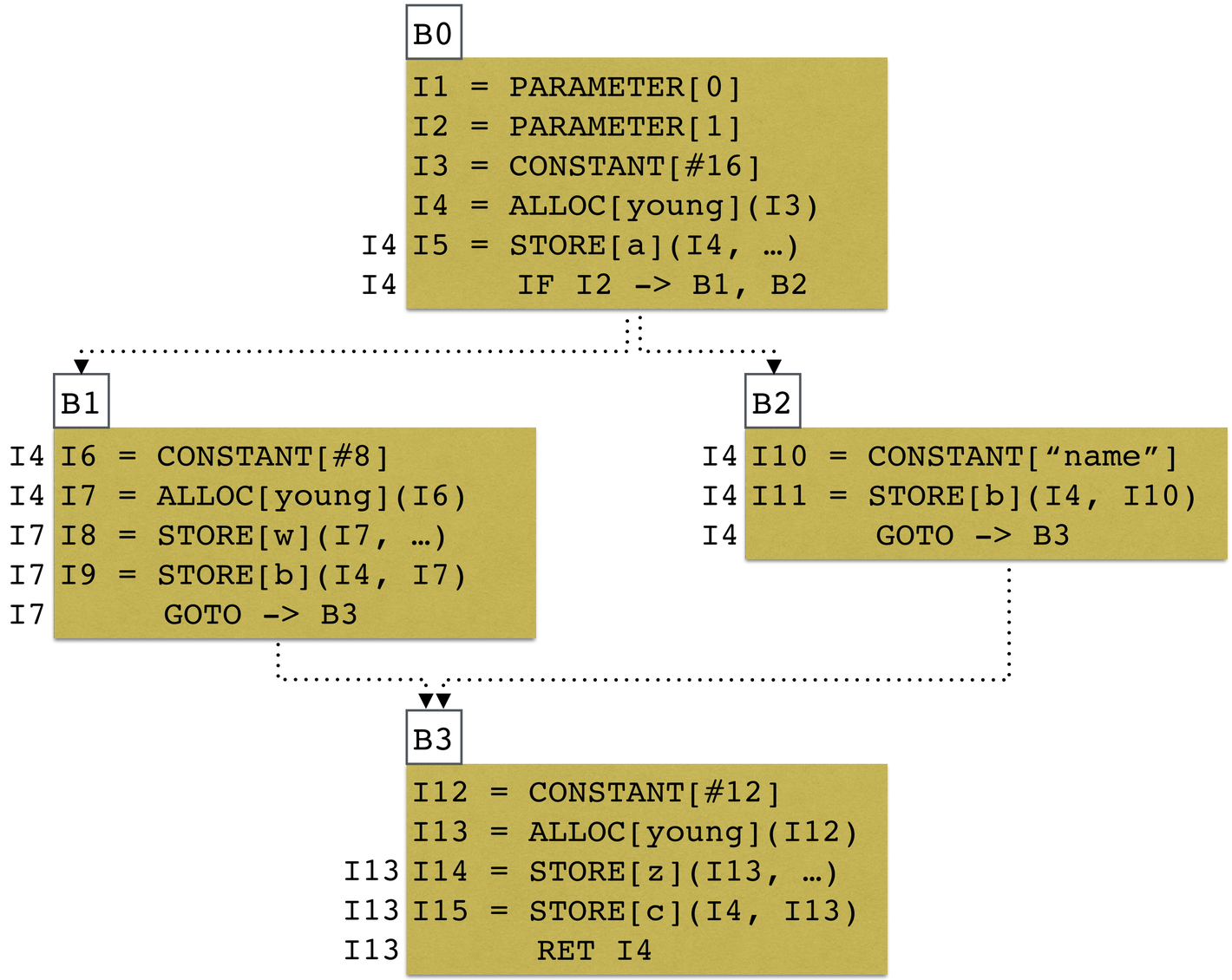


Figure 1: Control flow graph

Compilers

- Full compiler
  - Compiles everything
  - Unoptimized code
- Optimizing Compiler (Crankshaft)
  - Intermediate representation (IR)
  - Several compiler optimizations
    - Global value numbering (GVN)

Crankshaft IR Instruction	Dep	Chg
I_n = PARAMETER[K]		
I_n = CONSTANT[K]		
I_n = ARITH(I, I)		
I_n = LOAD[field](object)	Ψ	
I_n = STORE[field](object, value)		Ψ
I_n = ALLOC[space](size)	Λ	Λ
I_n = INNER[offset, size](alloc)		
I_n = CALL(I…)	*	*
I_n = PHI(I…)		
…		

Global Value Numbering (GVN)

- Eliminates redundant computations
- For pure operations
  - Arithmetic on primitives, math functions, etc.
  - Value numbering table
  - Example: `ARITH(`I_i`, `I_j`)`
    - If two `ARITH` instructions are value-equivalent (same operation and input values)
    - Second instruction is removed and reference to the first

Global Value Numbering (GVN)

- Important for allocation folding
- For impure operations
  - May be effected by side-effects

Global Value Numbering (GVN)

- Important for allocation folding
- For impure operations
  - May be effected by side-effects
  - Example:

```
L1 = LOAD[field](Oi)
...
...
...
L2 = LOAD[field](Oi)
```

value numbering table

Global Value Numbering (GVN)

- Important for allocation folding
- For impure operations
  - May be effected by side-effects
  - Example:

```
L1 = LOAD[field](Oi)
...
...
...
L2 = LOAD[field](Oi)
```

Crankshaft IR Instruction	Dep	Chg
`In = LOAD[field](object)`	Ψ	
`In = STORE[field](object, value)`		Ψ

value numbering table

Global Value Numbering (GVN)

- Important for allocation folding
- For impure operations
  - May be effected by side-effects
  - Example:

```
Ψ: L1 = LOAD[field](Oi)
   ...
   ...
   ...
Ψ: L2 = LOAD[field](Oi)
```

Crankshaft IR Instruction	Dep	Chg
I_n = LOAD[field](object)	Ψ	
I_n = STORE[field](object, value)		Ψ

value numbering table

Global Value Numbering (GVN)

- Important for allocation folding
- For impure operations
  - May be effected by side-effects
  - Example:

```
Ψ: L1 = LOAD[field](Oi)
   ...
   ...
   ...
Ψ: L2 = LOAD[field](Oi)
```

Crankshaft IR Instruction	Dep	Chg
In = LOAD[field](object)	Ψ	
In = STORE[field](object, value)		Ψ

value numbering table

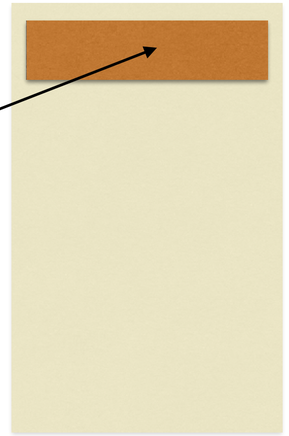
Global Value Numbering (GVN)

- Important for allocation folding
- For impure operations
  - May be effected by side-effects
  - Example:

```
Ψ: L1 = LOAD[field](Oi)
   ...
   ...
   ...
Ψ: L2 = LOAD[field](Oi)
```

Crankshaft IR Instruction	Dep	Chg
I_n = LOAD[field](object)	Ψ	
I_n = STORE[field](object, value)		Ψ

value numbering table



Global Value Numbering (GVN)

- Important for allocation folding
- For impure operations
  - May be effected by side-effects
  - Example:

```
Ψ: L1 = LOAD[field](Oi)
   ...
   ...
   ...
Ψ: L2 = LOAD[field](Oi)
```

Crankshaft IR Instruction	Dep	Chg
I_n = LOAD[field](object)	Ψ	
I_n = STORE[field](object, value)		Ψ

value numbering table

Global Value Numbering (GVN)

- Important for allocation folding
- For impure operations
  - May be effected by side-effects
  - Example:

```
Ψ: L1 = LOAD[field](Oi)
   ...
Ψ: S1 = STORE[field](Oi, Vj)
   ...
Ψ: L2 = LOAD[field](Oi)
```

Crankshaft IR Instruction	Dep	Chg
`In = LOAD[field](object)`	Ψ	
`In = STORE[field](object, value)`		Ψ

value numbering table

Global Value Numbering (GVN)

- Important for allocation folding
- For impure operations
  - May be effected by side-effects
  - Example:

```
Ψ: L1 = LOAD[field](Oi)
   ...
Ψ: S1 = STORE[field](Oi, Vj)
   ...
Ψ: L2 = LOAD[field](Oi)
```

Crankshaft IR Instruction	Dep	Chg
I_n = LOAD[field](object)	Ψ	
I_n = STORE[field](object, value)		Ψ

value numbering table

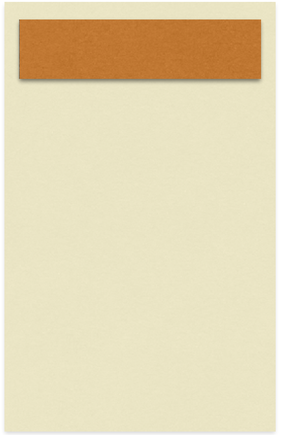
Global Value Numbering (GVN)

- Important for allocation folding
- For impure operations
  - May be effected by side-effects
  - Example:

```
Ψ: L1 = LOAD[field](Oi)
   ...
Ψ: S1 = STORE[field](Oi, Vj)
   ...
Ψ: L2 = LOAD[field](Oi)
```

Crankshaft IR Instruction	Dep	Chg
`In = LOAD[field](object)`	Ψ	
`In = STORE[field](object, value)`		Ψ

value numbering table



Global Value Numbering (GVN)

- Important for allocation folding
- For impure operations
  - May be effected by side-effects
  - Example:

```
Ψ: L1 = LOAD[field](Oi)
   ...
Ψ: S1 = STORE[field](Oi, Vj)
   ...
Ψ: L2 = LOAD[field](Oi)
```

Crankshaft IR Instruction	Dep	Chg
I_n = LOAD[field](object)	Ψ	
I_n = STORE[field](object, value)		Ψ

value numbering table

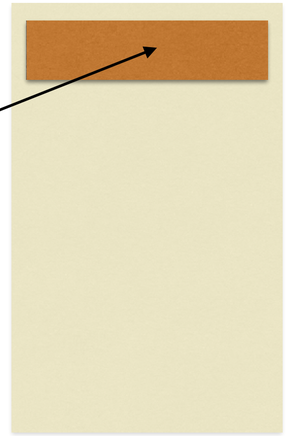
Global Value Numbering (GVN)

- Important for allocation folding
- For impure operations
  - May be effected by side-effects
  - Example:

```
Ψ: L1 = LOAD[field](Oi)
   ...
Ψ: S1 = STORE[field](Oi, Vj)
   ...
Ψ: L2 = LOAD[field](Oi)
```

Crankshaft IR Instruction	Dep	Chg
`In = LOAD[field](object)`	Ψ	
`In = STORE[field](object, value)`		Ψ

value numbering table



Allocation Folding Based on Dominance

Daniel Clifford, Hannes Payer, Michael Starzinger and Ben L. Titzer

International Symposium on Memory Management (ISMM) 2014

presented by Thomas Hütter

Concurrency & Memory Management Seminar 2015

Univ.- Prof. Dr. Christoph Kirsch

Department of Computer Sciences
University of Salzburg

Definition 1 (*ε-dominance*)

For a given effect ε, instruction D ε-dominates instruction I if and only if D occurs on every path from the function entry to I, and no path from D to I contains another instruction D' ≠ D that changes ε.

Predicate 1 (*Load Elimination*)

A load L_2=LOAD[$field_j$](O_i) *can be replaced with* L_1=LOAD[$field_j$](O_i) *if* L_1 *Ψ-dominates* L_2.

Definition 2 (*ε-dominator*)

For a given effect ε, instruction D is the ε-dominator of instruction I if and only if D ε-dominates I and D changes ε.

Allocation Folding

- `ALLOC`
  - Allocates a contiguous chunk of memory
- `INNER`
  - Computes effective address of a sub-region within an allocated chunk of memory

Crankshaft IR Instruction	Dep	Chg
I_n = ALLOC[space](size)	∧	∧
I_n = INNER[offset, size](alloc)		

Predicate 3 (*Allocation Folding on Crankshaft IR*)

Allocations A_1=`ALLOC[s](`K_1`)` *and* A_2=`ALLOC[s](`K_2`)` *are candidates for allocation folding if* A_1 *is the* Λ*-dominator of* A_2.

Allocation Folding (*Example*)

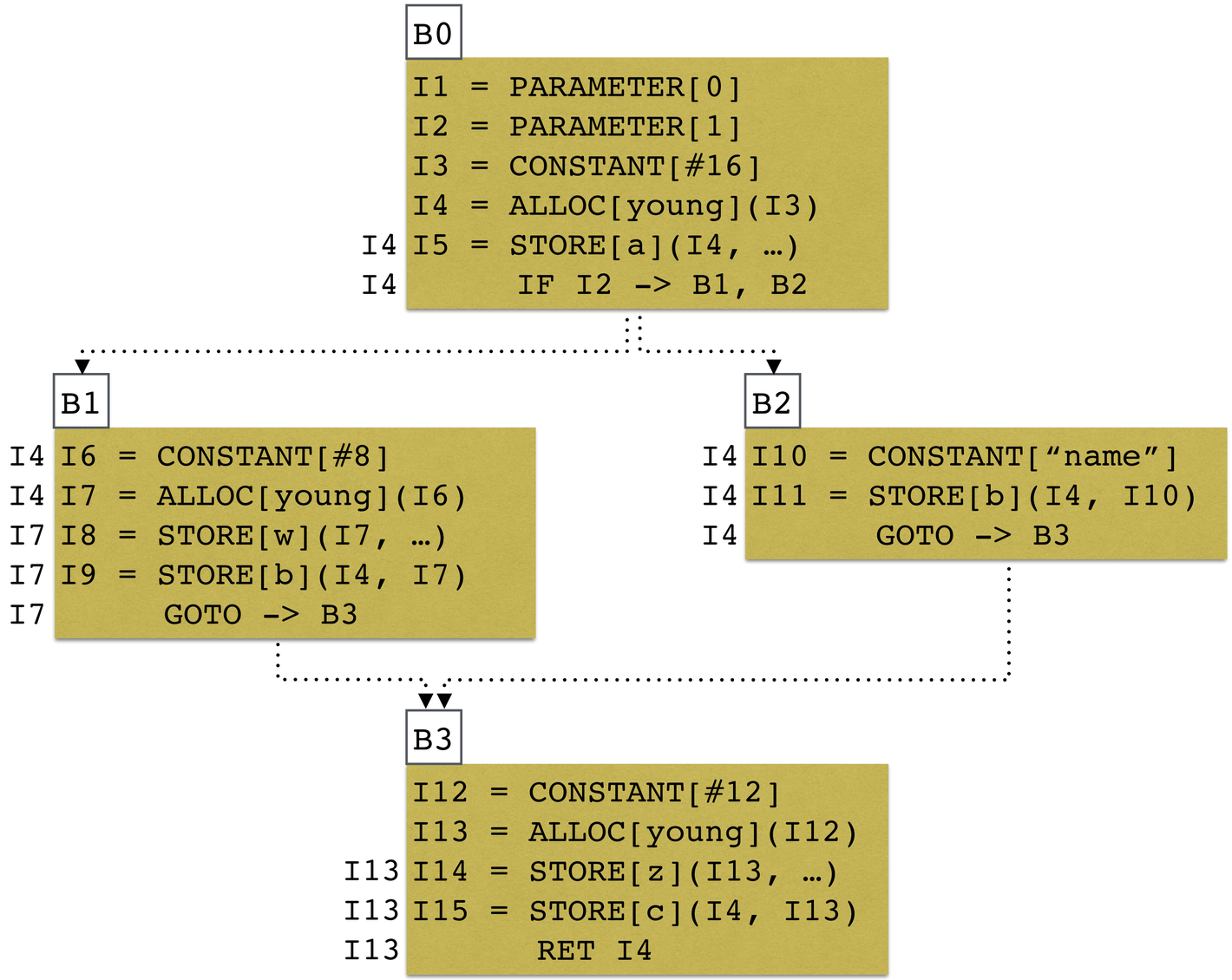


Figure 1: Control flow graph befor allocation folding.

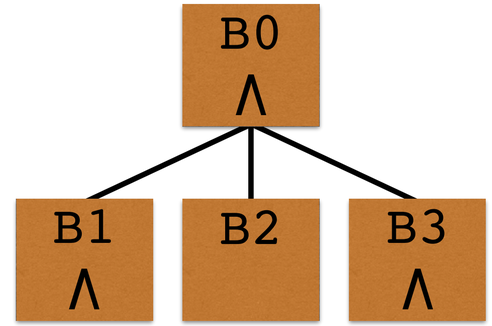


Figure 1a: Dominator tree.

Allocation Folding (*Example*)

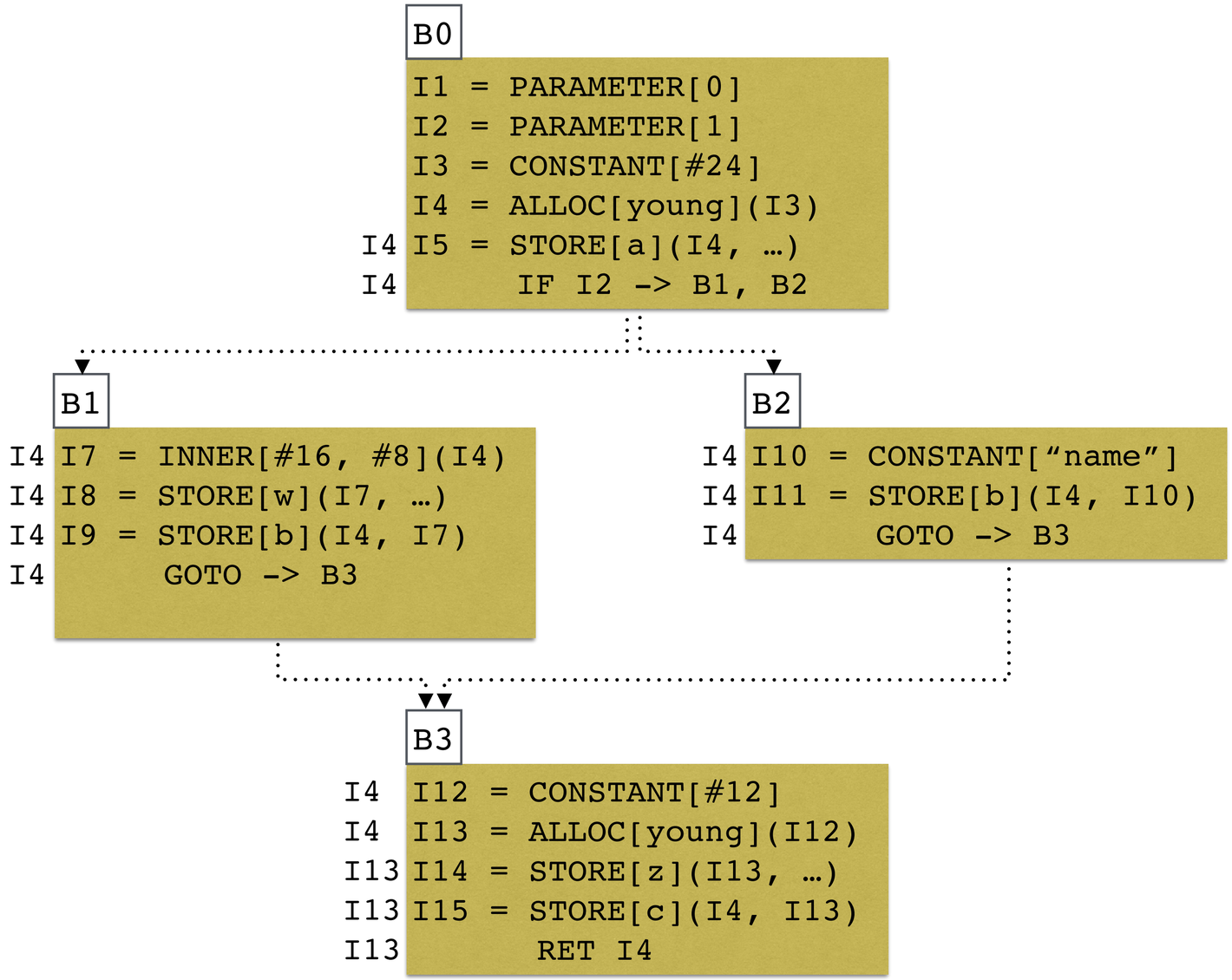


Figure 1: Control flow graph befor allocation folding.

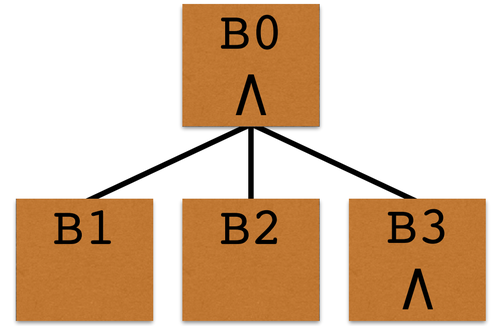


Figure 1a: Dominator tree.

Allocation Folding (*Example*)

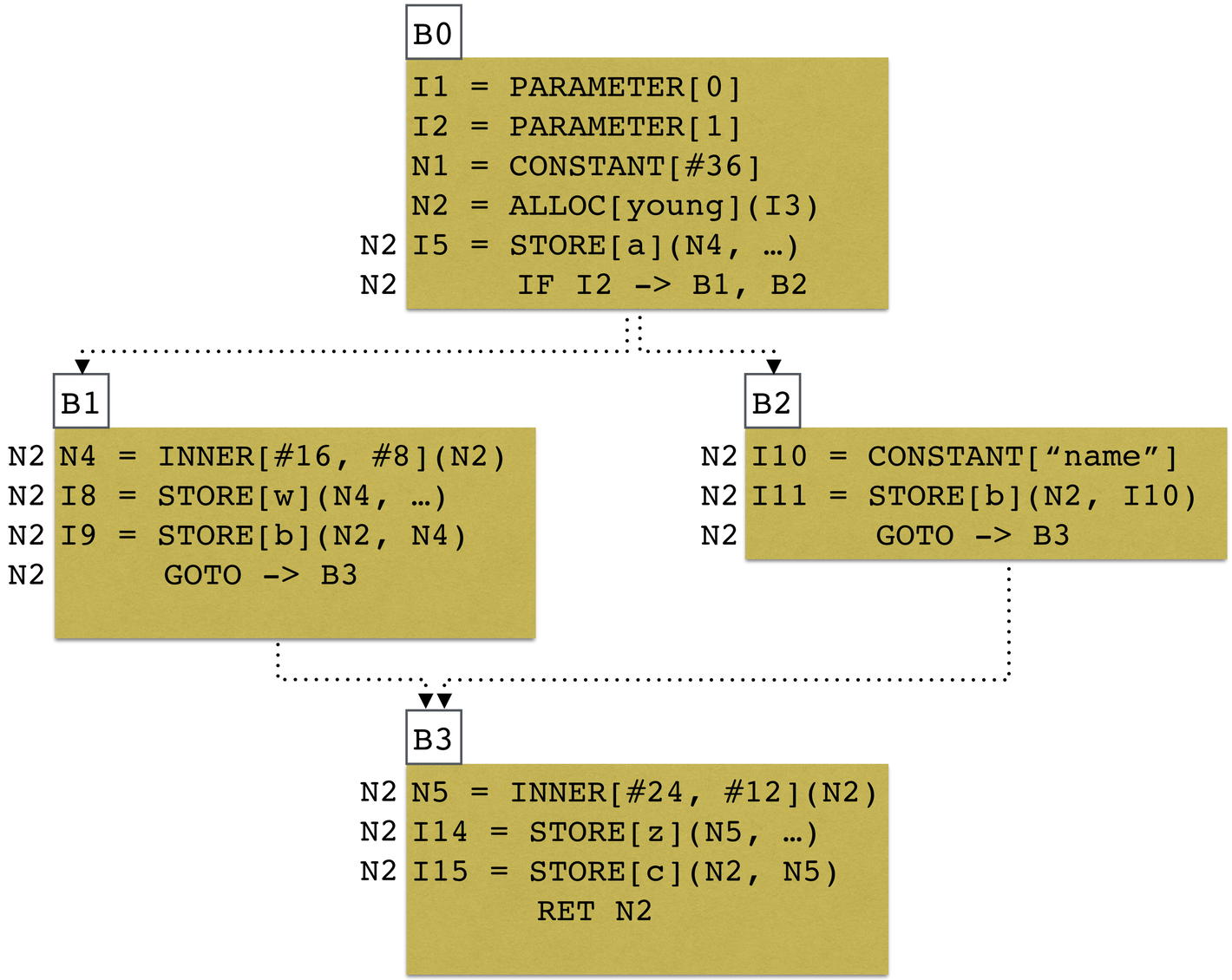


Figure 2: Control flow graph after allocation folding.

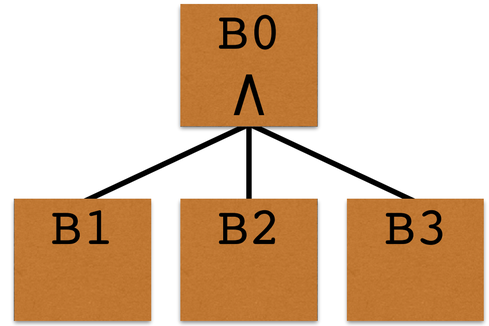


Figure 2a: Dominator tree.

Experiments

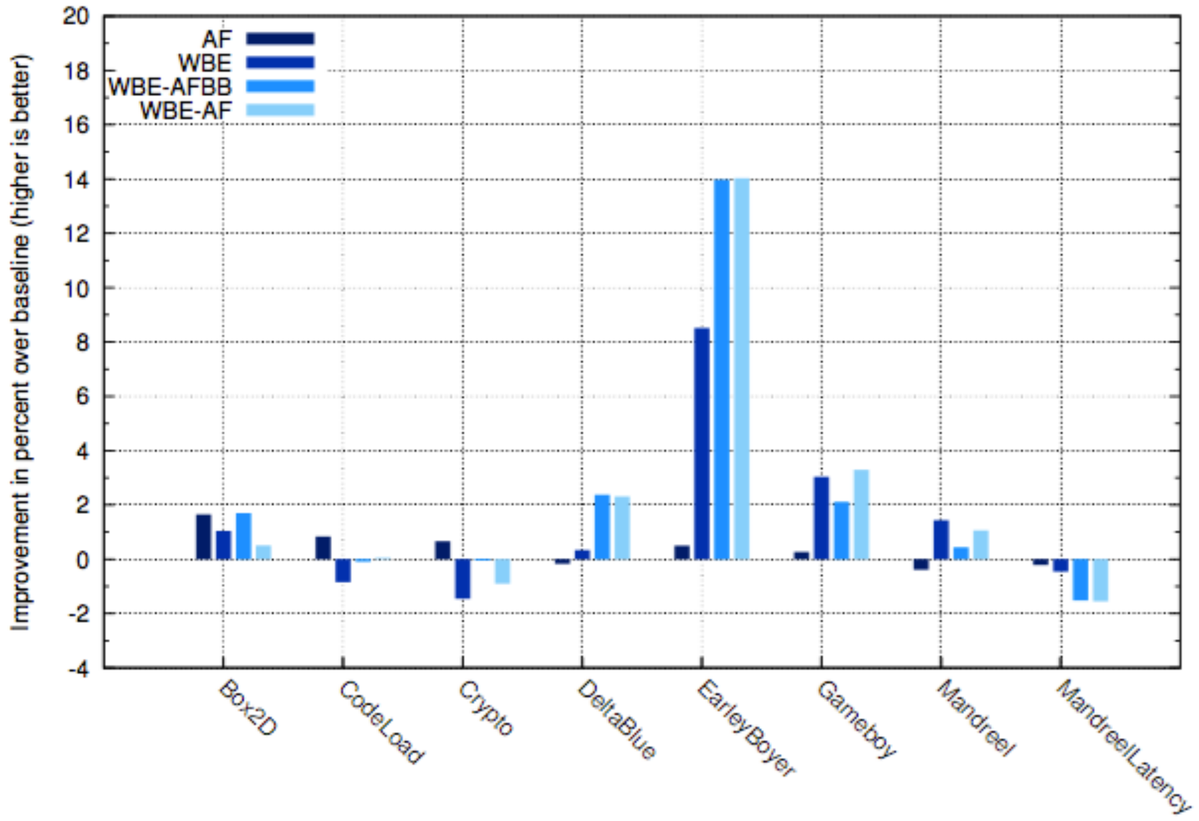
- v8 revision r18926
- X64 Server
  - Intel Core i5-2400 quad core 3.10GHz CPU
  - 80 GB main memory
  - Linux
- No relevant differences on IA32 and ARM
- Average of 20 repetitions

Experiments

- Benchmarks
  - Octane 2.0
  - Kraken 1.1
  - Four other hand selected benchmark

- Naming
  - Allocation folding (AF)
  - Write barrier elimination (WBE)
  - Write barrier elimination and allocation folding on basic blocks (WBE-AFBB)
  - Write barrier elimination and allocation folding (WBE-AF)

Throughput



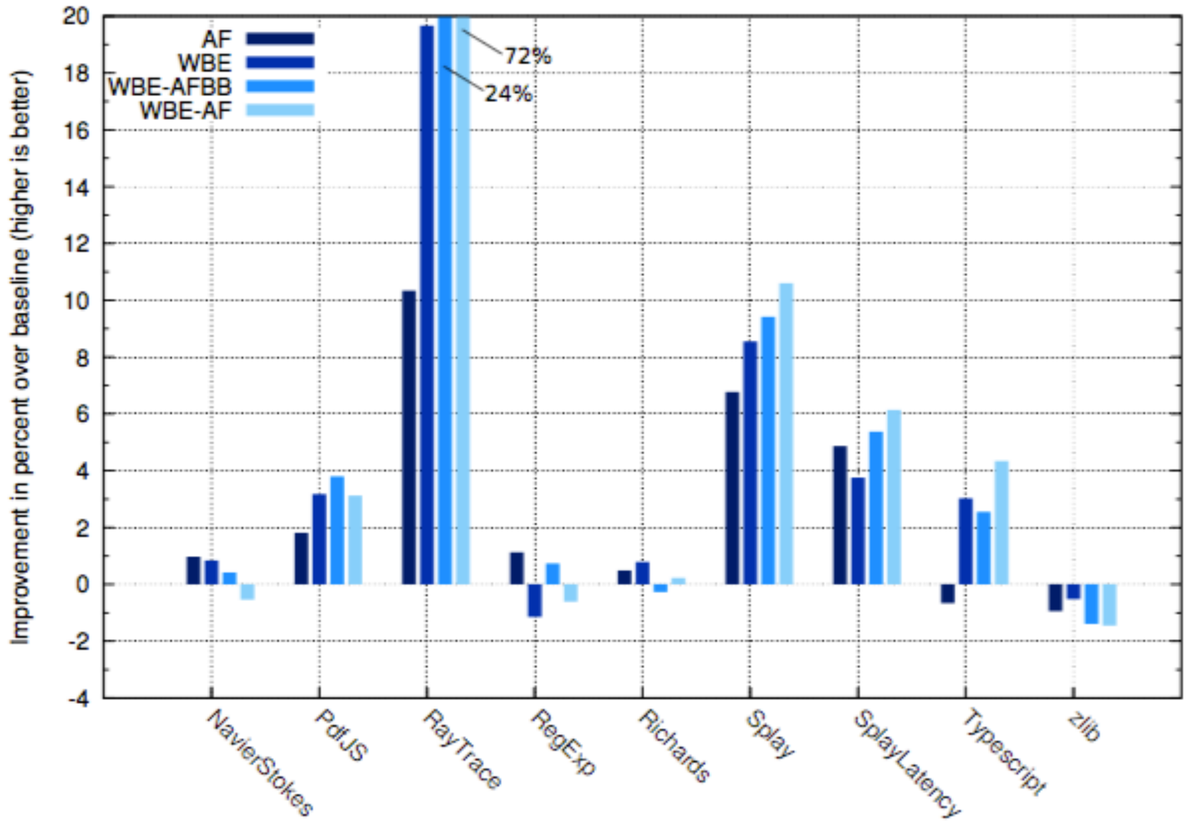


Figure 3: Improvement in percent of all configurations over the baseline on the Octane suite running on X64.

Throughput

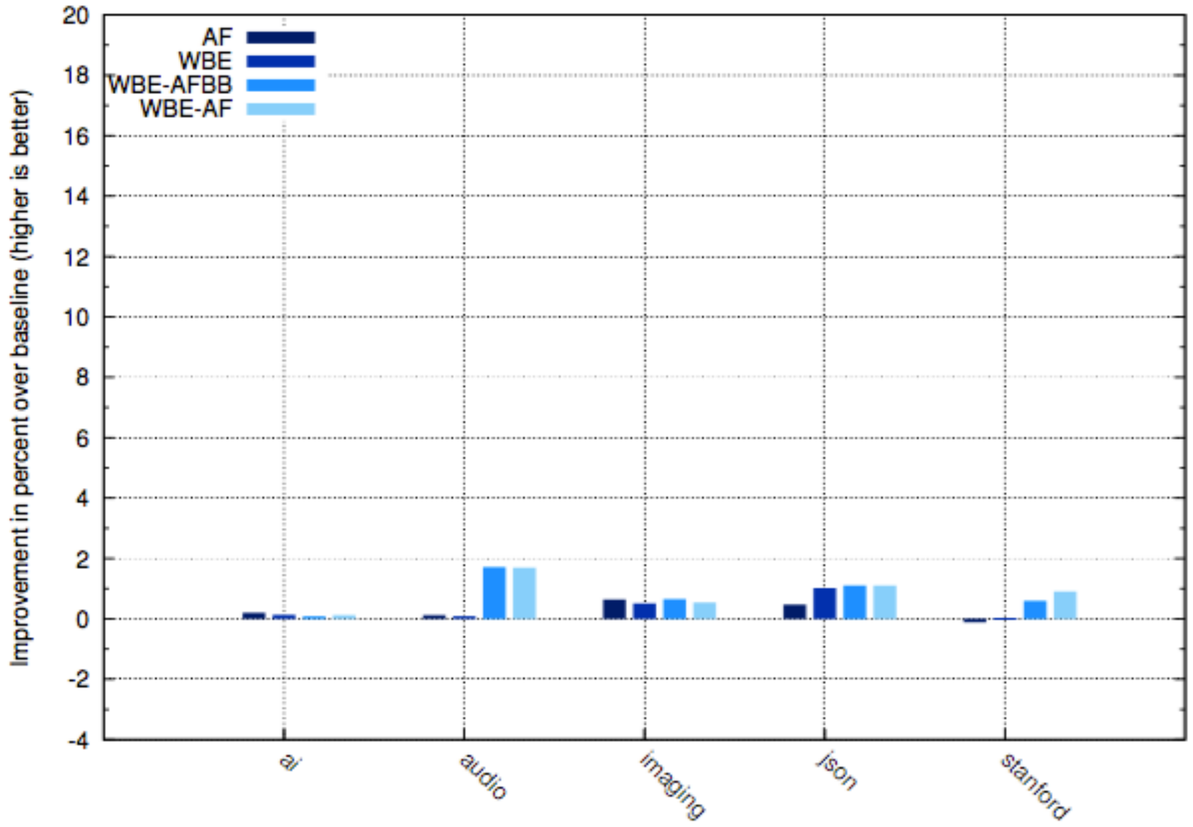


Figure 4: Improvement in percent of all configurations over the baseline on the Kraken suite running on X64.

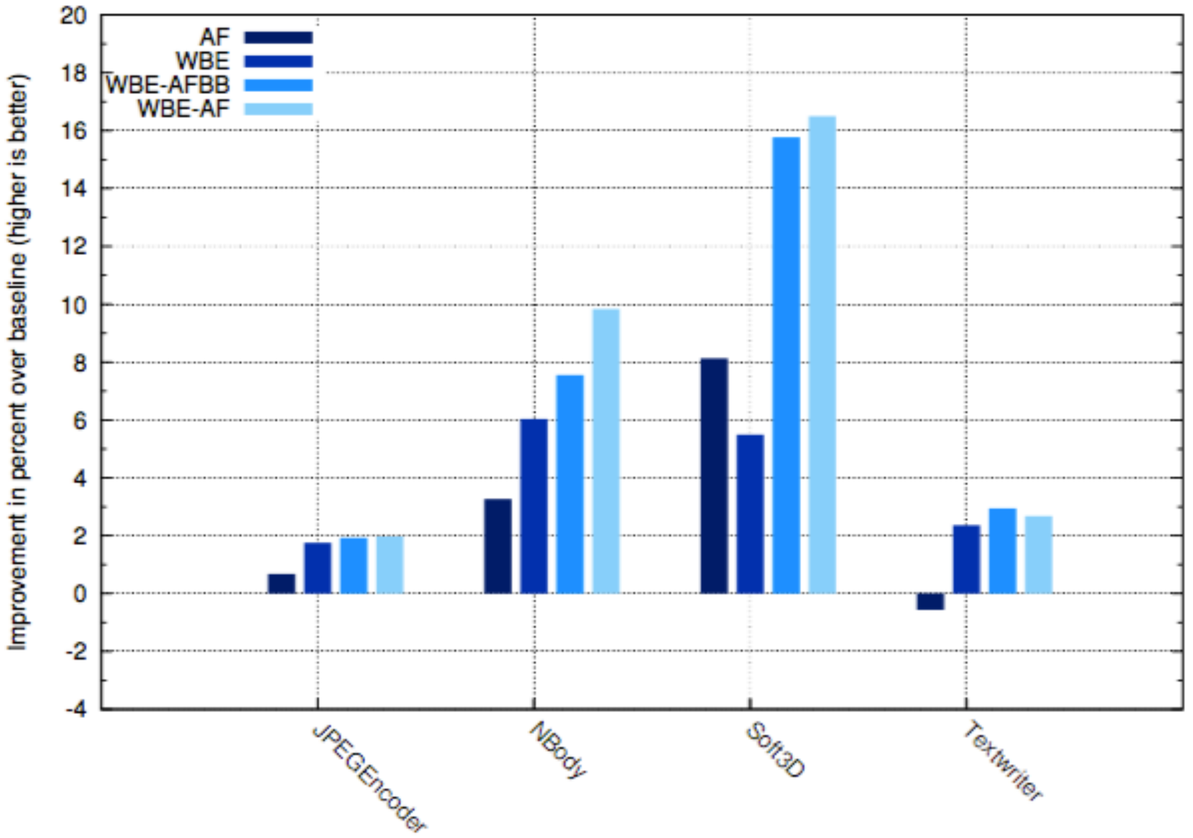


Figure 5: Improvement in percent of all configurations over the baseline on the hand-selected benchmarks running on X64.

Folded Allocations

Benchmark	AFBB [%]	AF [%]
Box2D	21	35
CodeLoad	28	28
Crypto	33	42
DeltaBlue	37	47
EarleyBoyer	26	42
Gameboy	3	3
Mandreel	38	38
Mandreel Latency	38	38
NavierStokes	18	18
PdfJS	29	31
RayTrace	11	65
RegExp	27	27
Richards	26	65

Benchmark	AFBB [%]	AF [%]
Splay	22	40
SplayLatency	22	40
Typescript	6	12
zlib	82	82
ai	25	25
audio	37	38
imaging	0	0
json	0	0
standford	23	24
JPEGEncoder	18	55
NBody	76	88
Soft3D	56	58
Textwriter	8	17

Table 2, 3, 4: Static proportion of folded allocation instructions in Octane, Kraken and hand-selected benchmarks.

Allocation Group Utilization

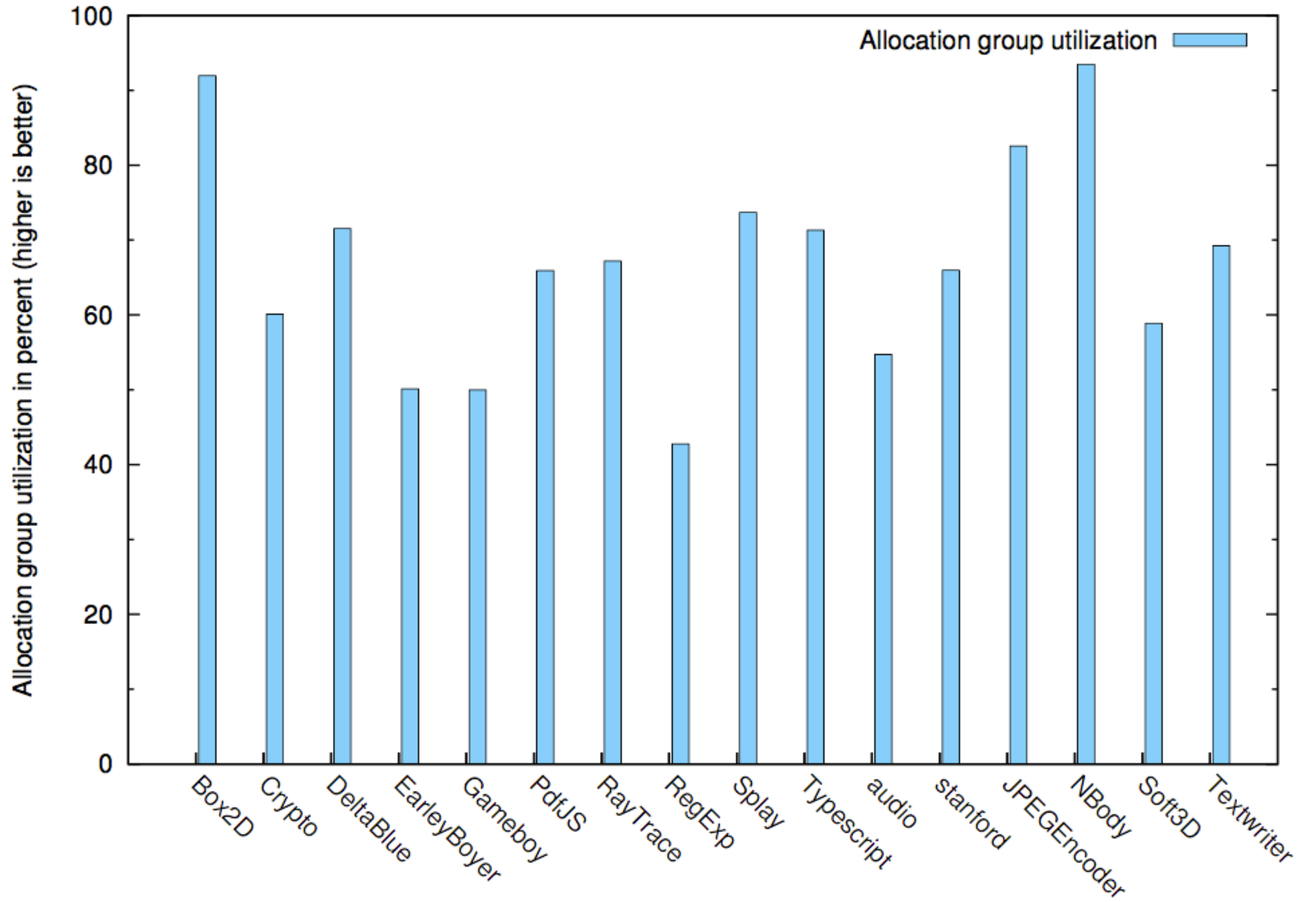


Figure 6: Allocation group utilization on X64.

Garbage Collection Overhead

	Baseline, WBE, WBE-AFBB (no fragmentation)			AF, WBE-AF (fragmentation)		
Benchmark	#Minor GCs	#Major GCs	GC time in ms	#Minor GCs	#Major GCs	GC time in ms
Box2D	111	5	99.37	111	5	100.97
CodeLoad	14	5	137.45	14	5	136.22
Cyrpto	62	0	2.25	69	0	1.5
DeltaBlue	585	0	44.57	587	0	45.11
EarleyBoyer	856	0	770.58	856	0	763.82
Gameboy	37	18	112.99	37	19	116.51
Mandreel	106	6	12.84	106	6	12.91
MandreelLatency	106	6	12.84	106	6	12.91
NavierStokes	16	0	2.18	16	0	2.26
PdfJS	555	26	772.54	555	29	813.28
RayTrace	2599	0	18.5	2610	0	22.11
RegExp	959	0	20.49	956	0	18.91
Richards	63	0	0.56	63	0	0.56
Splay	311	194	416.69	313	194	419.1
SplayLatency	311	194	416.69	313	194	419.1
Typescript	51	6	444.77	51	6	449.11
zlib	1	1	2.41	1	1	2.35

	Baseline, WBE, WBE-AFBB (no fragmentation)			AF, WBE-AF (fragmentation)		
Benchmark	Minor GCs	Major GCs	GC time in ms	Minor GCs	Major GCs	GC time in ms
ai	4	1	6.55	4	1	6.49
audio	42	6	18.37	43	6	18.31
imaging	2	4	6.45	2	4	6.33
json	21	2	4.16	21	2	4.26
stanford	45	4	16.96	45	4	16.97

	Baseline, WBE, WBE-AFBB (no fragmentation)			AF, WBE-AF (fragmentation)		
Benchmark	Minor GCs	Major GCs	GC time in ms	Minor GCs	Major GCs	GC time in ms
JPEGEncoder	18	1	17.89	18	1	17.51
NBody	597	0	0.31	620	0	0.31
Soft3D	437	0	57.44	493	0	105.67
TextWriter	2213	0	6.28	2230	0	10.13

Table 8, 9, 10: Number of minor collections, major collections, and total garbage collection time in ms with and without allocation folding in the Octane, Kraken Suite and hand-selected benchmarks on X64.

Thank you for your attention