СБЕР

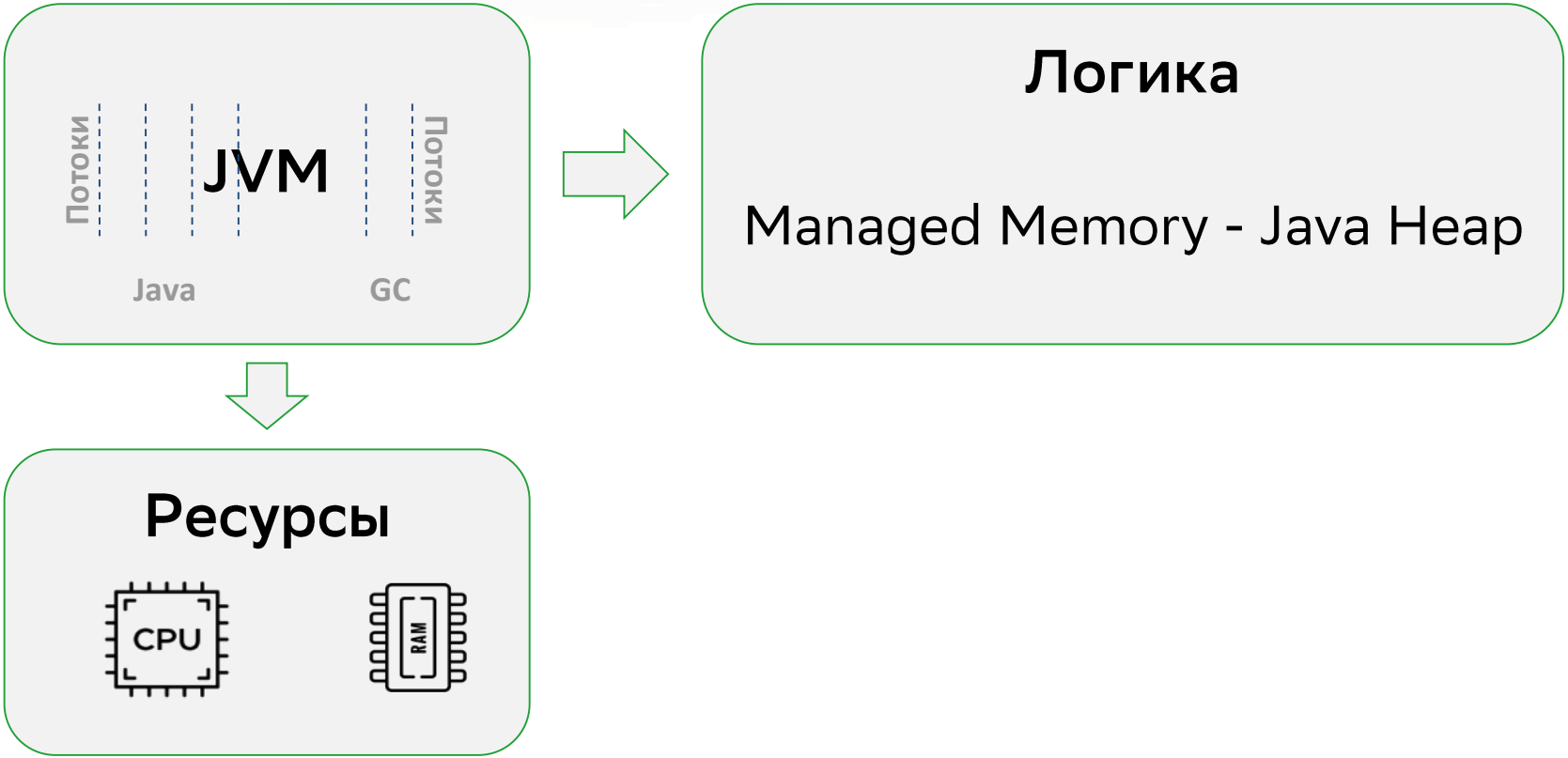
Точки соприкосновения: Java & GC

эксперт Владимир Воскресенский

спикер Силин Дмитрий

JVM

Области взаимодействия



Многопоточное программирование

synchronized

`mutex`

safepoint

lock-free

`barrier`

concurrent

Многопоточное программирование

synchronized

`mutex`

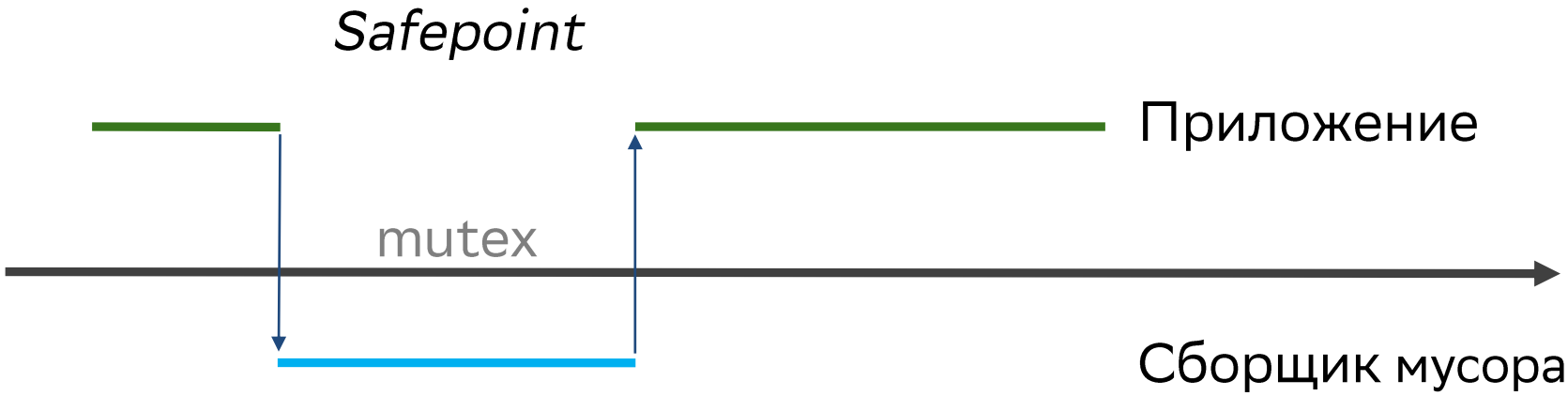
safepoint

lock-free

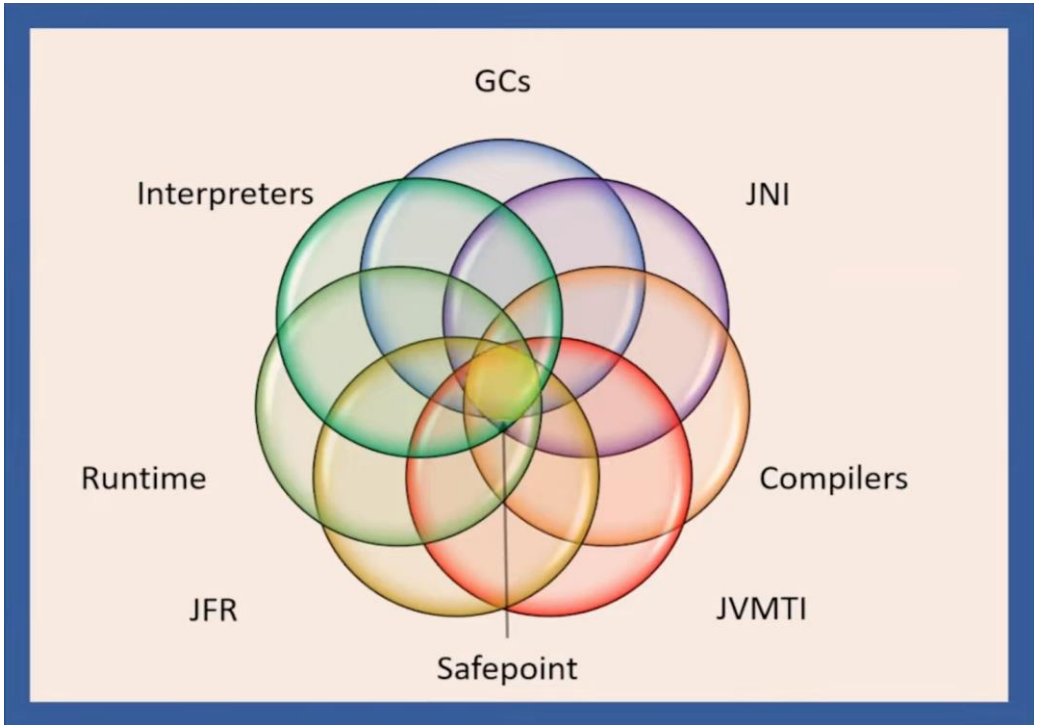
`barrier`

concurrent

Mutex и JVM

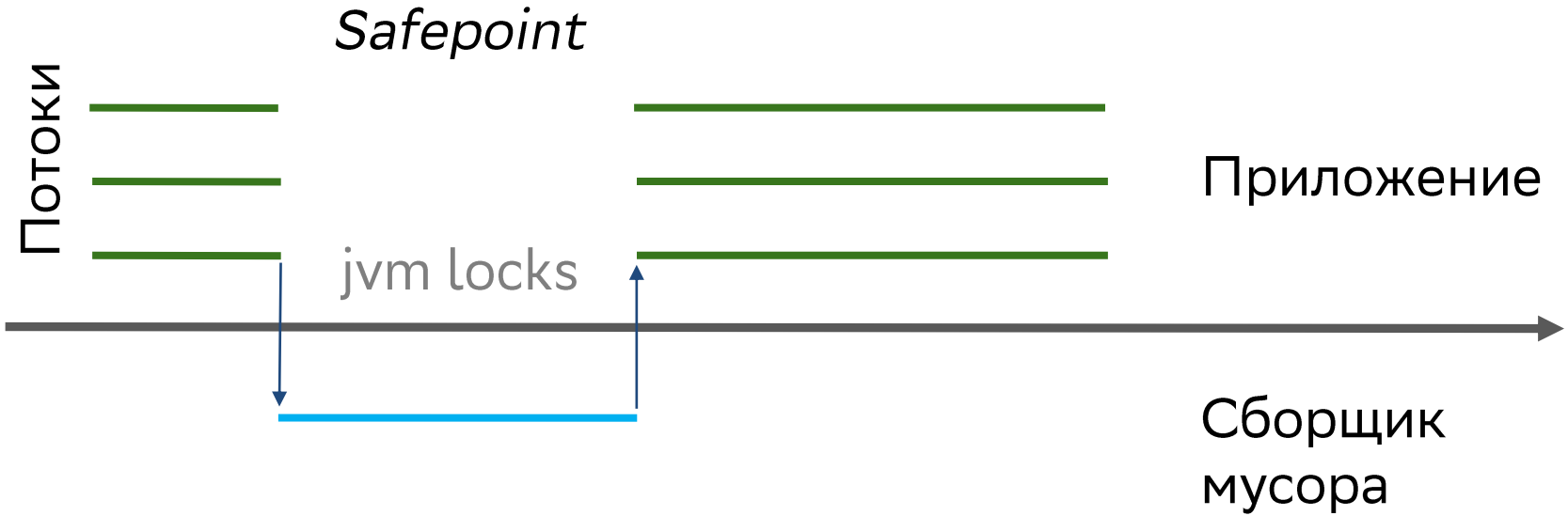


Safepoint



https://www.youtube.com/watch?v=JkbWPPNc4SI

Mutex в JVM



thread->jvm_lock : non-locked by self

- доступ к java heap:
- исполнение java code:
  - ждёт lock в java
  - работает в vm / native (jni)

thread->jvm_lock : locked by self

- ✅ исполнение java code
- ✅ доступ к java heap
  - регулярно проверяет не надо ли отдать jvm_lock

Safepoint poll [interpreter]

```
void TemplateTable::_goto() {
  transition(vtos, vtos);
  branch(false, false);
}
```

Safepoint poll [interpreter]

```
void TemplateTable::_goto() {
  void TemplateTable::branch(bool is_jsr, bool is_wide) {
    ...
    __ dispatch_only(vtos, true);
    ...
  }
```

Safepoint poll [interpreter]

```
void TemplateTable::_goto() {
```

```
  void TemplateTable::branch(bool is_jsr, bool is_wide) {
```

```
  void InterpreterMacroAssembler::dispatch_only(
                                  TosState state,
                                  bool generate_poll) {
    dispatch_base(state,
                  Interpreter::dispatch_table(state),
                  true,
                  generate_poll);
  }
```

Safepoint poll [interpreter]

```
1  void InterpreterMacroAssembler::dispatch_base(...,
2                                                bool generate_poll) {
3    ...
4    Label no_safepoint, dispatch;
5    if (table != safepoint_table && generate_poll) {
6      NOT_PRODUCT(block_comment("Thread-local Safepoint poll"));
7      testb(Address(r15_thread, JavaThread::polling_word_offset()),
8                    SafepointMechanism::poll_bit());
9
10     jccb(Assembler::zero, no_safepoint);
11     lea(rscratch1, ExternalAddress((address)safepoint_table));
12     jmpb(dispatch);
13   }
14   ...
15 }
```

Safepoint poll [interpreter]

```
1  void InterpreterMacroAssembler::dispatch_base(...,
2                                                bool generate_poll) {
3    ...
4    Label no_safepoint, dispatch;
5    if (table != safepoint_table && generate_poll) {
6      NOT_PRODUCT(block_comment("Thread-local Safepoint poll"));
7      testb(Address(r15_thread, JavaThread::polling_word_offset()),
8                    SafepointMechanism::poll_bit());
9
10     jccb(Assembler::zero, no_safepoint);
11     lea(rscratch1, ExternalAddress((address)safepoint_table));
12     jmpb(dispatch);
13   }
14   ...
15 }
```

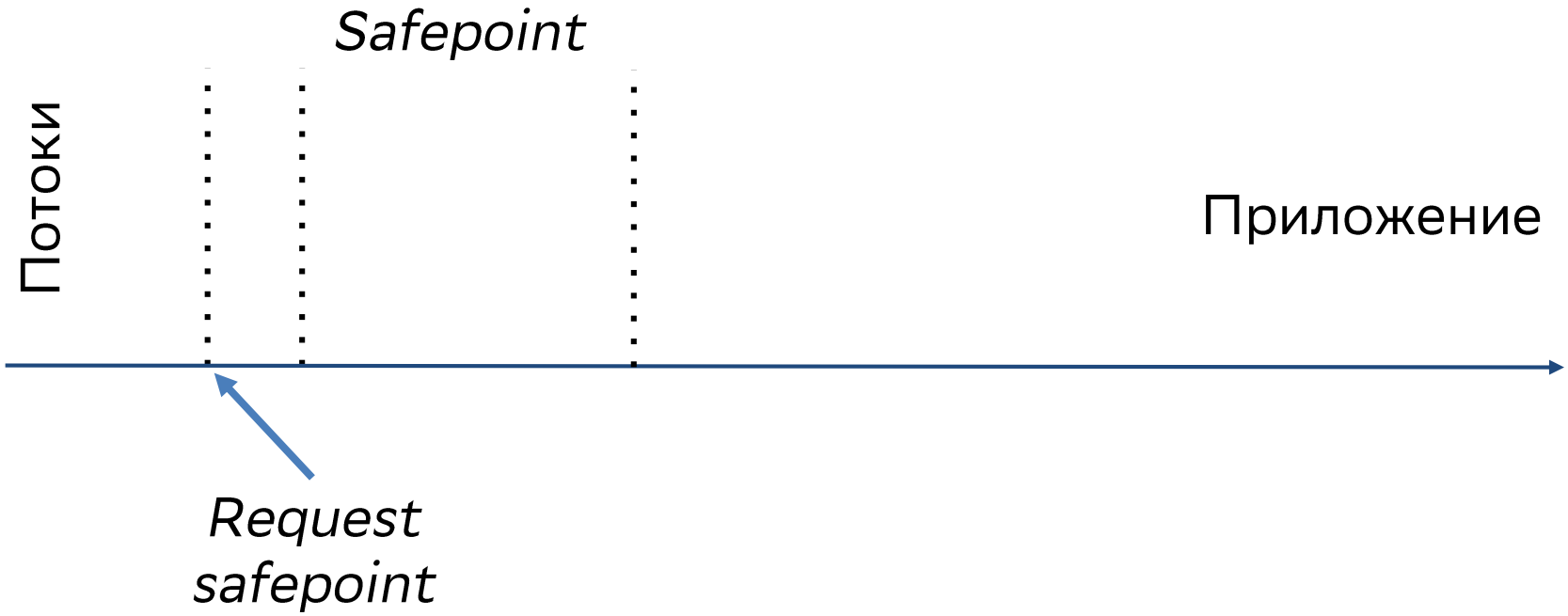
Safepoint poll [interpreter]

```
1  void InterpreterMacroAssembler::dispatch_base(...,
2                                                bool generate_poll) {
3    ...
4    Label no_safepoint, dispatch;
5    if (table != safepoint_table && generate_poll) {
6      NOT_PRODUCT(block_comment("Thread-local Safepoint poll"));
7      testb(Address(r15_thread, JavaThread::polling_word_offset()),
8                    SafepointMechanism::poll_bit());
9
10     jccb(Assembler::zero, no_safepoint);
11     lea(rscratch1, ExternalAddress((address)safepoint_table));
12     jmpb(dispatch);
13   }
14   ...
15 }
```

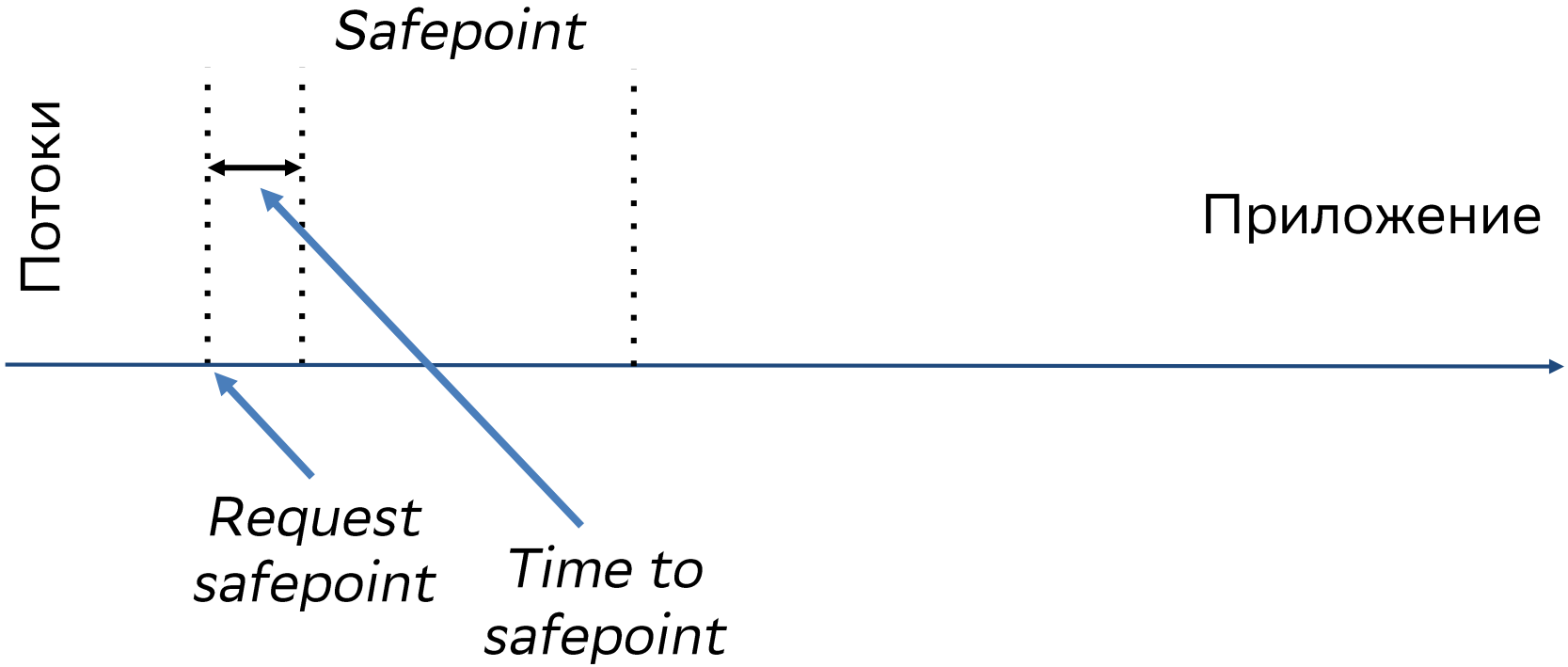
Safepoint poll [interpreter]

```
1  void InterpreterMacroAssembler::dispatch_base(...,
2                                                bool generate_poll) {
3    ...
4    Label no_safepoint, dispatch;
5    if (table != safepoint_table && generate_poll) {
6      NOT_PRODUCT(block_comment("Thread-local Safepoint poll"));
7      testb(Address(r15_thread, JavaThread::polling_word_offset()),
8                    SafepointMechanism::poll_bit());
9
10     jccb(Assembler::zero, no_safepoint);
11     lea(rscratch1, ExternalAddress((address)safepoint_table));
12     jmpb(dispatch);
13   }
14   ...
15 }
```

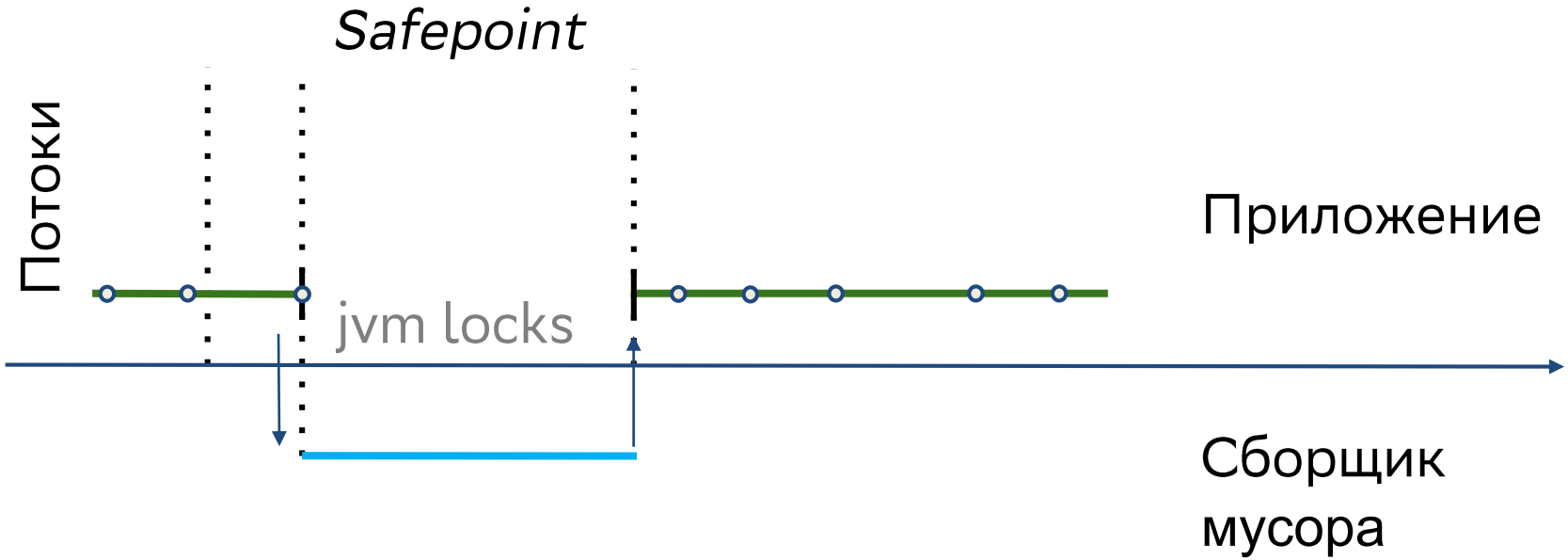
Механизмы синхронизации



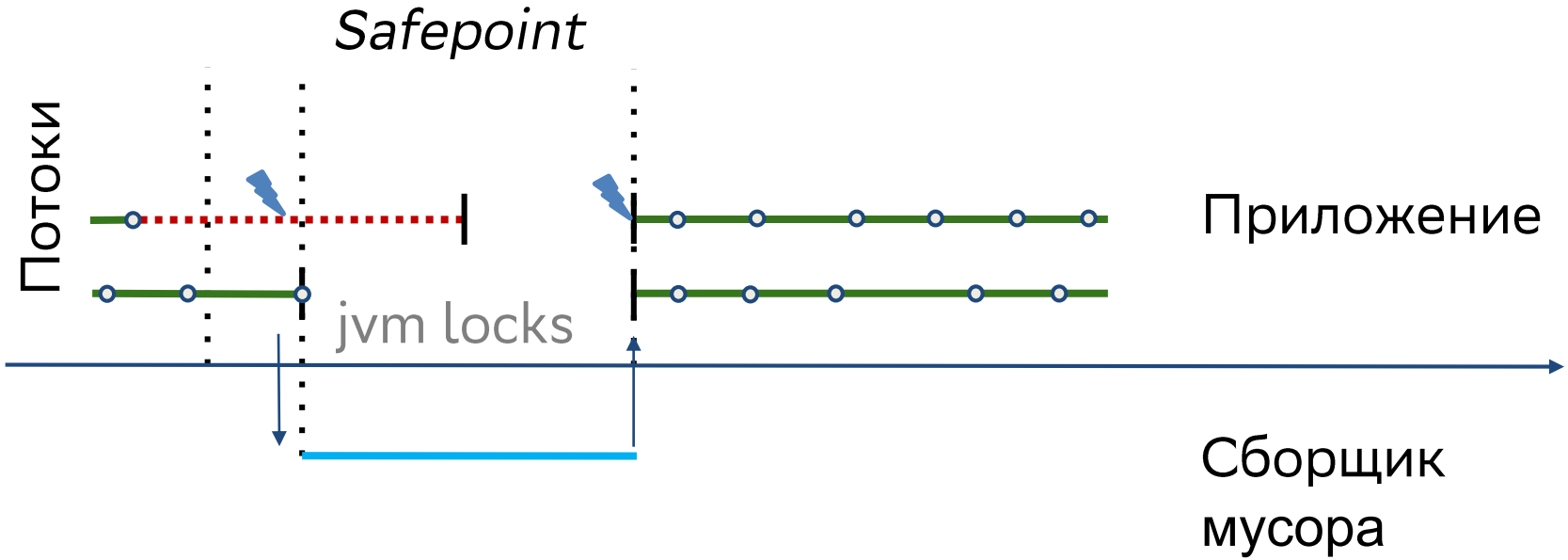
Механизмы синхронизации



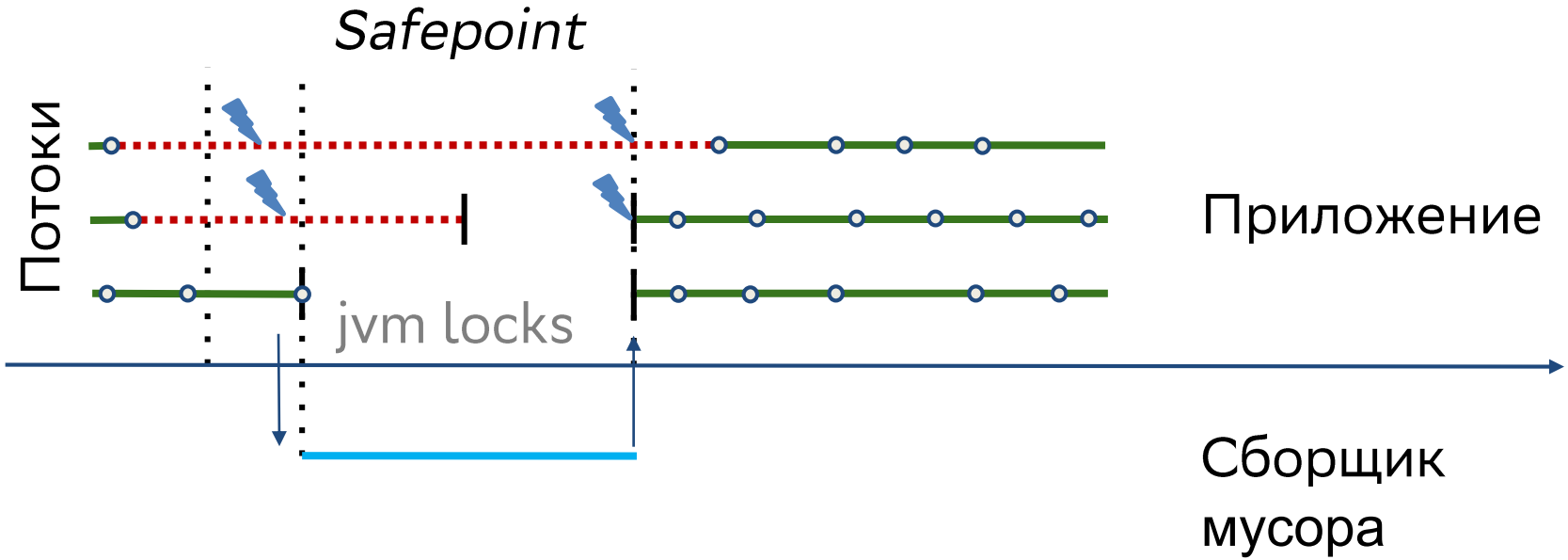
Safepoint



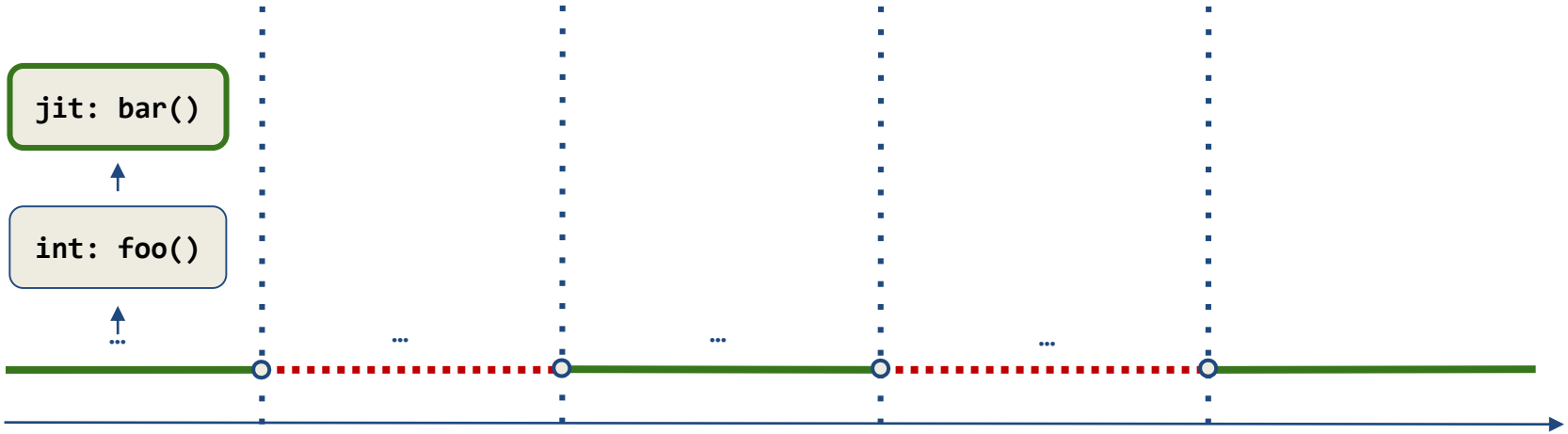
Safepoint



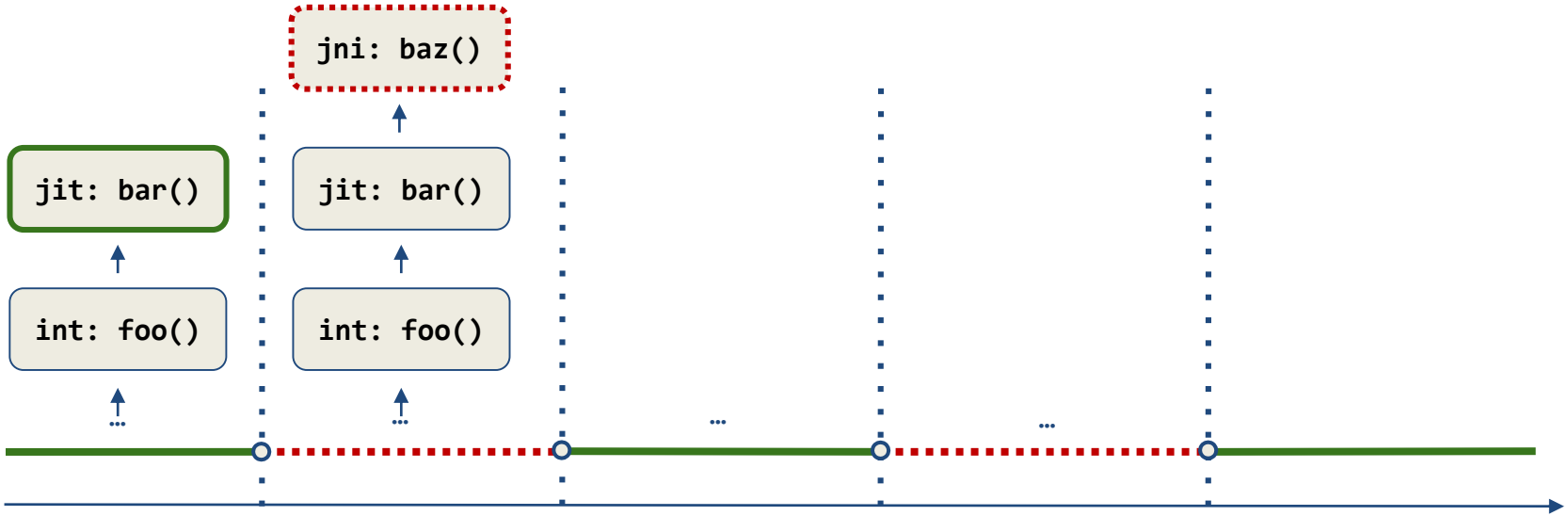
Safepoint



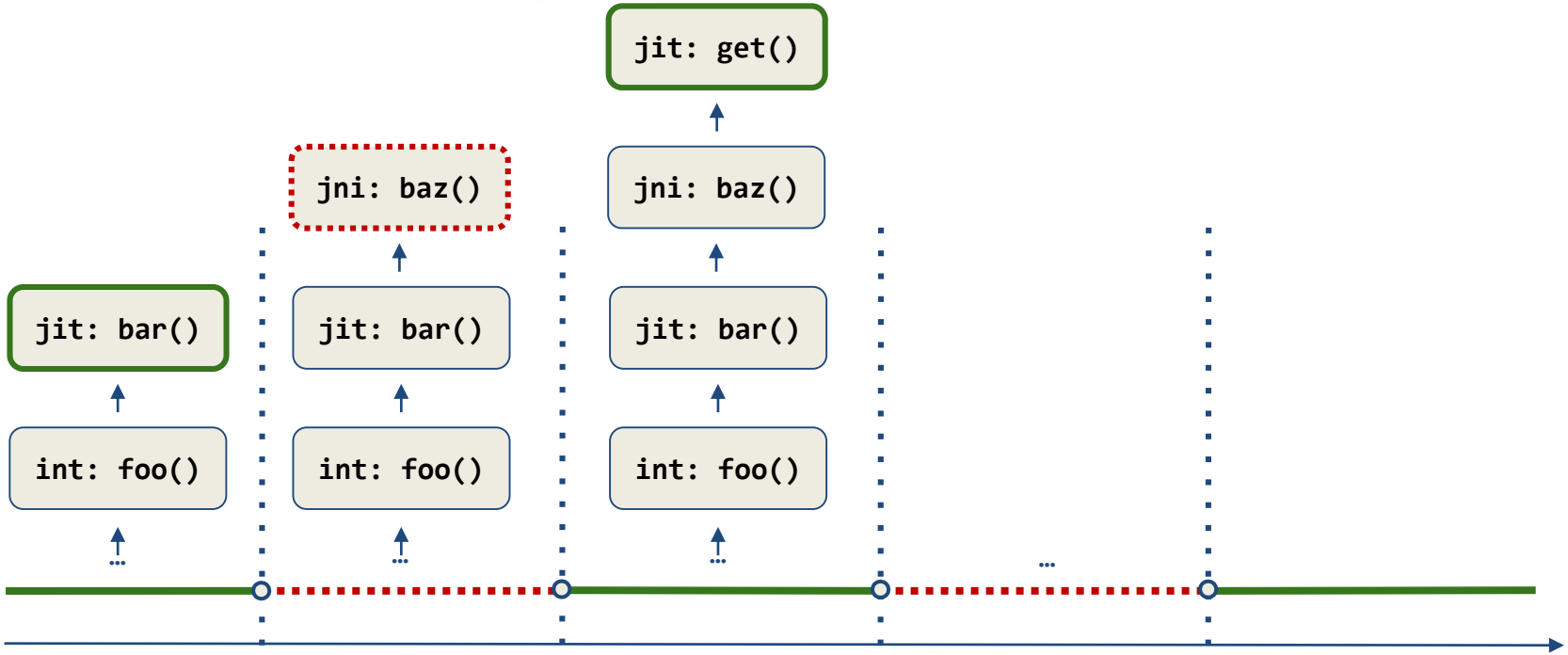
Stack



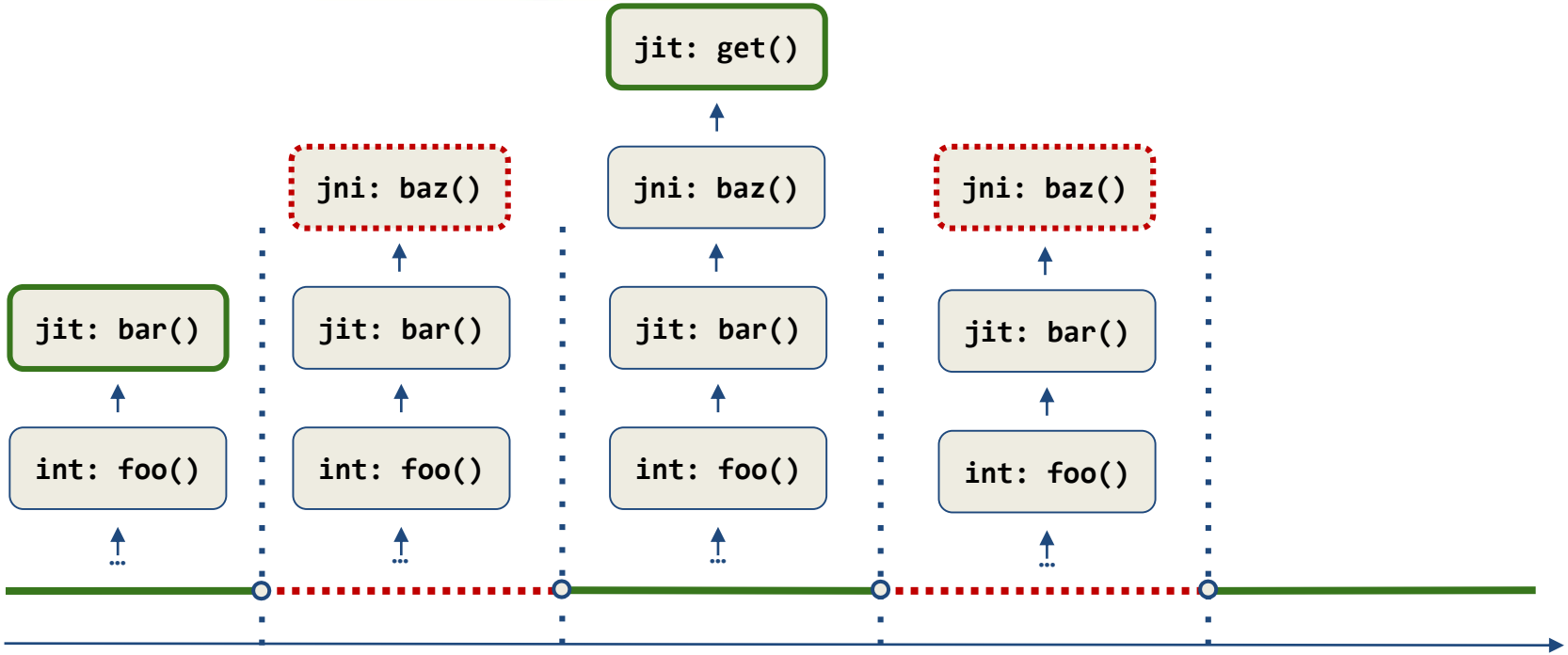
Stack



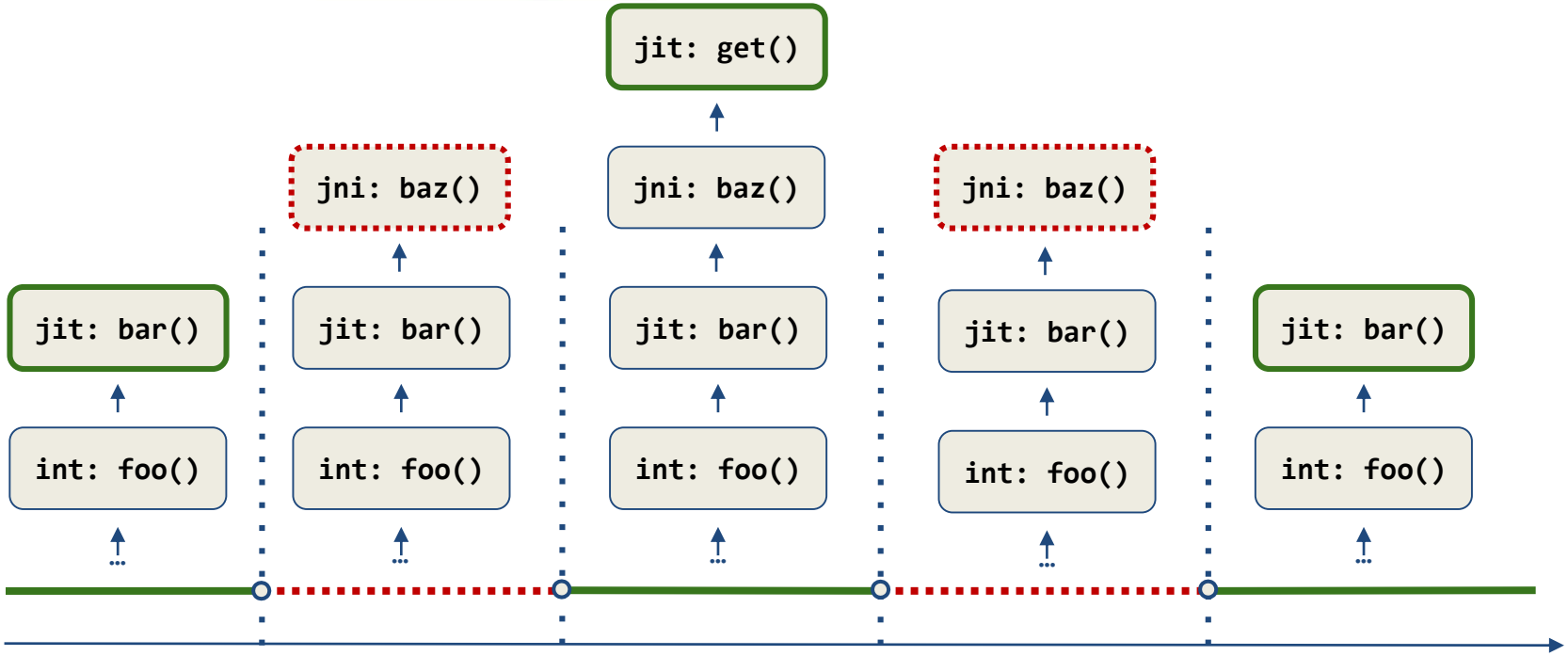
Stack


jit: get()
jni: baz()
jni: baz()
jit: bar()
jit: bar()
jit: bar()
int: foo()
int: foo()
int: foo()

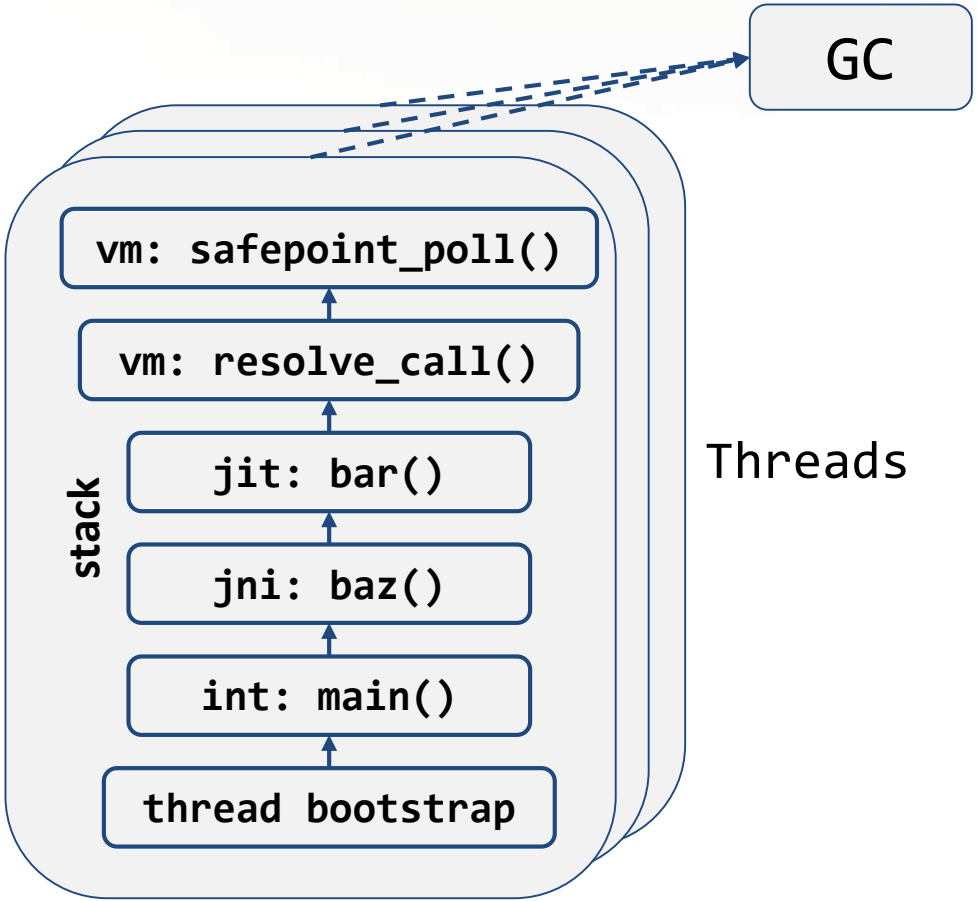
Stack


jit: get()
jni: baz()
jni: baz()
jni: baz()
jit: bar()
jit: bar()
jit: bar()
jit: bar()
int: foo()
int: foo()
int: foo()
int: foo()

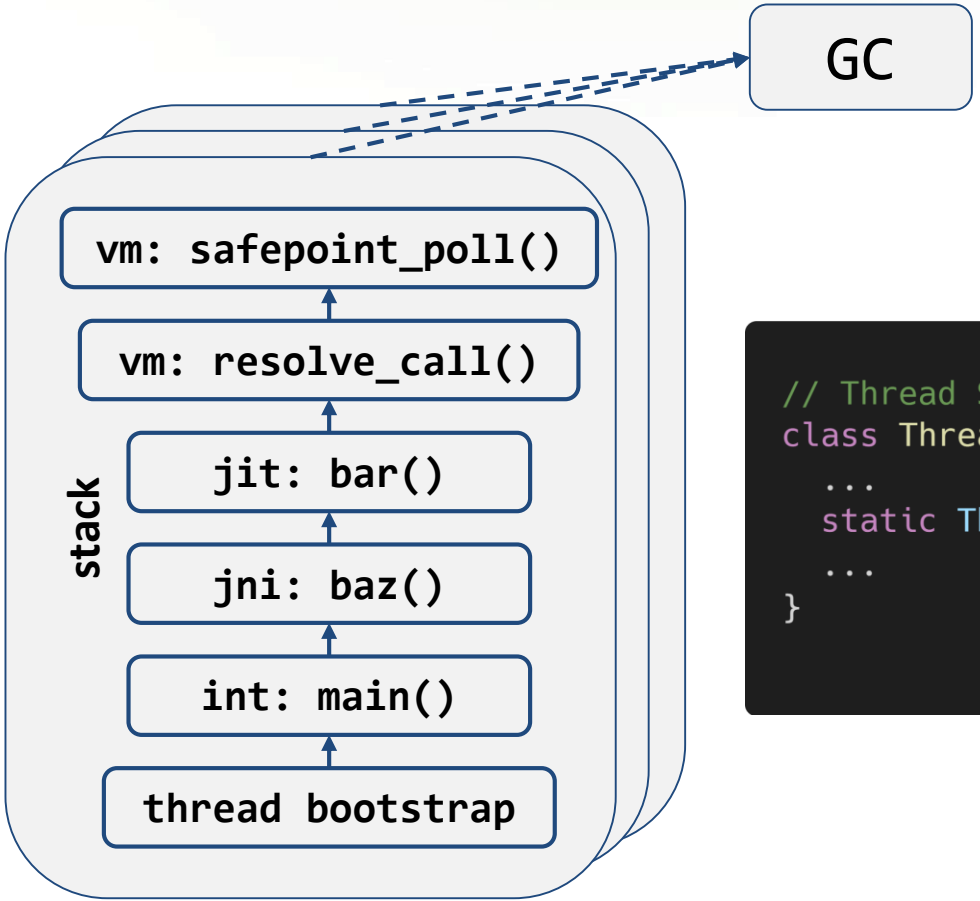
Stack


jit: get()
jni: baz()
jni: baz()
jni: baz()
jit: bar()
jit: bar()
jit: bar()
jit: bar()
jit: bar()
int: foo()
int: foo()
int: foo()
int: foo()
int: foo()

Come to safepoint



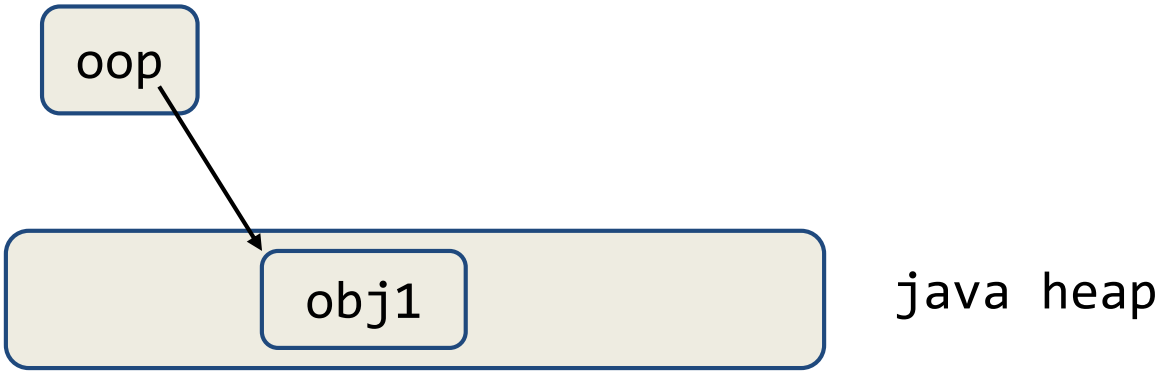
Iterate threads




```
// Thread Safe Memory Reclamation (Thread-SMR) support.
class ThreadsSMRSupport : AllStatic {
  ...
  static ThreadsList* volatile _java_thread_list;
  ...
}
```

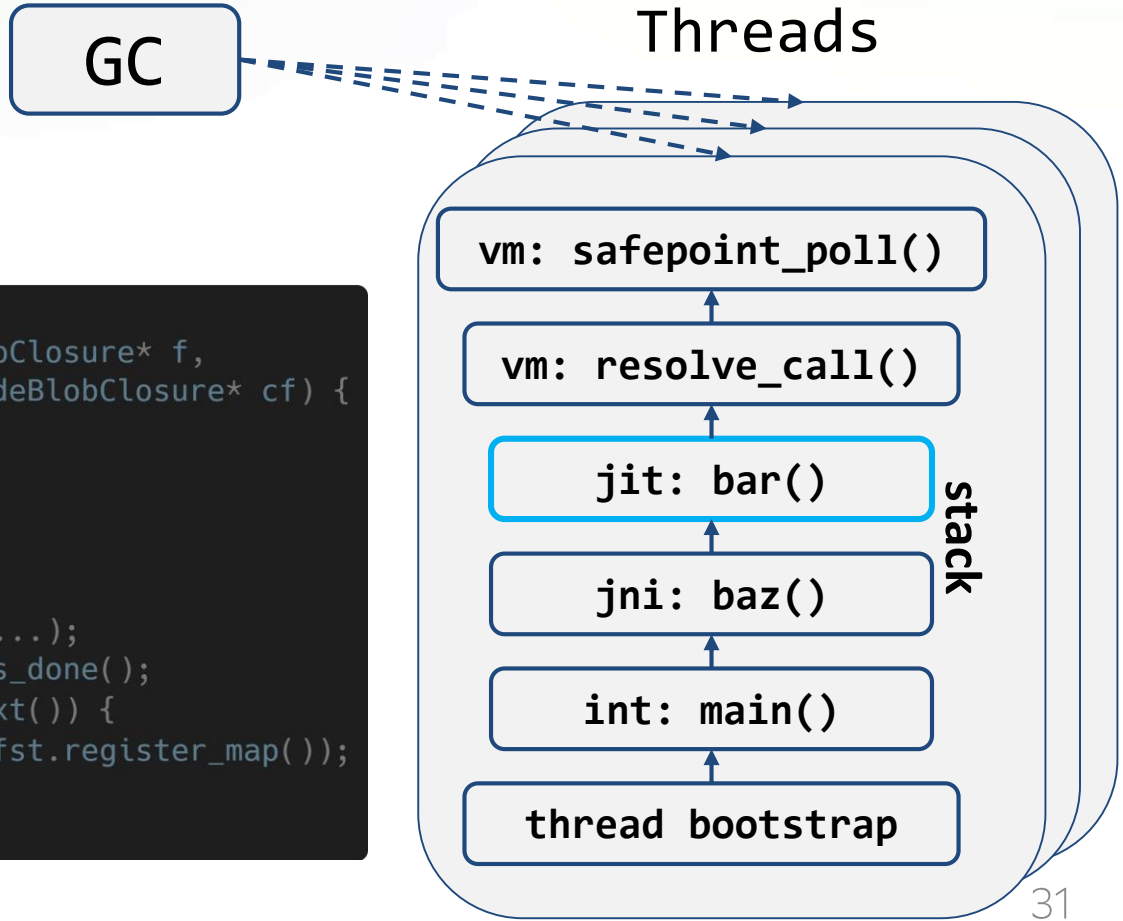
Oop

- ordinary object pointer
- address to java heap, where object begins



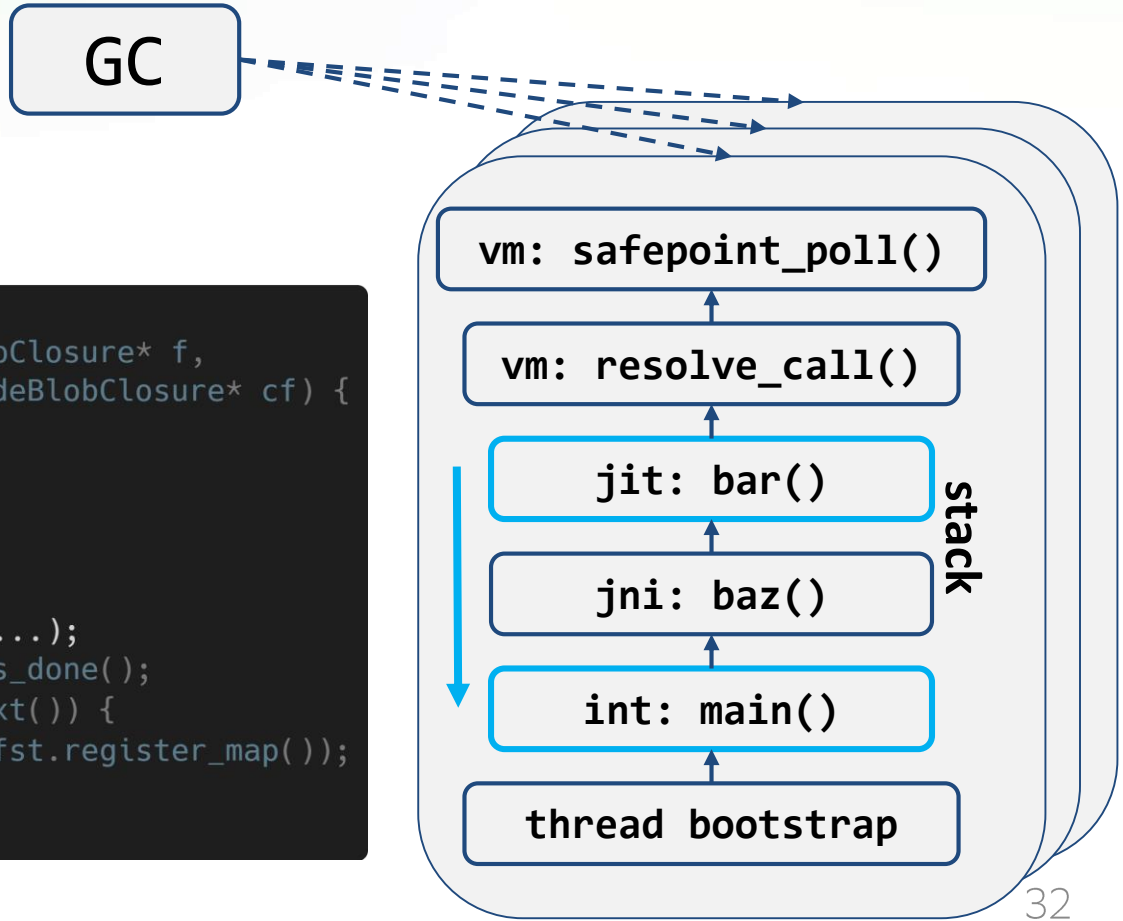
Find anchor

```
1 void JavaThread::oops_do_frames(OopClosure* f,
2                                 CodeBlobClosure* cf) {
3   if (!has_last_Java_frame()) {
4     return;
5   }
6   ...
7   // Traverse the execution stack
8   for (StackFrameStream fst(this, ...);
9                             !fst.is_done();
10                            fst.next()) {
11    fst.current()->oops_do(f, cf, fst.register_map());
12  }
13 }
```



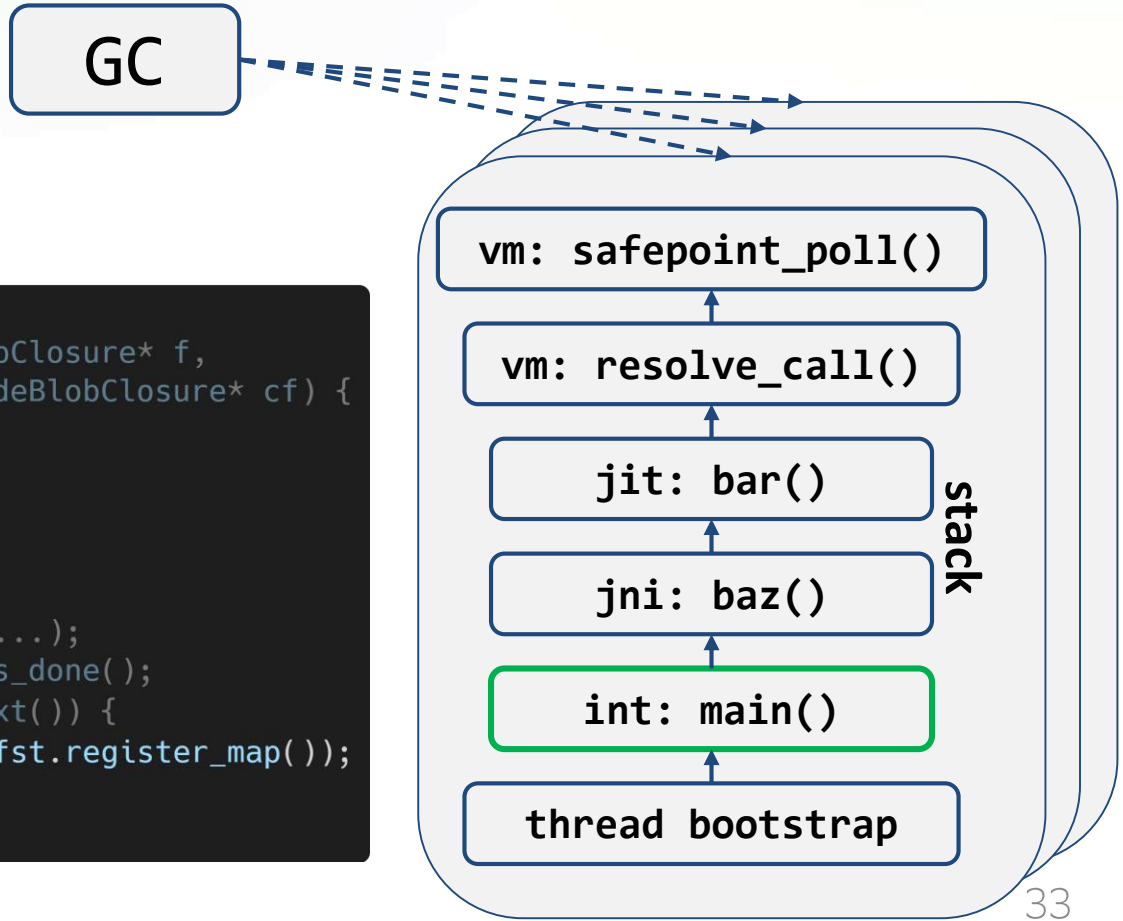
Iterate frames

```
1  void JavaThread::oops_do_frames(OopClosure* f,
2                                  CodeBlobClosure* cf) {
3    if (!has_last_Java_frame()) {
4      return;
5    }
6    ...
7    // Traverse the execution stack
8    for (StackFrameStream fst(this, ...);
9                              !fst.is_done();
10                             fst.next()) {
11     fst.current()->oops_do(f, cf, fst.register_map());
12   }
13 }
```



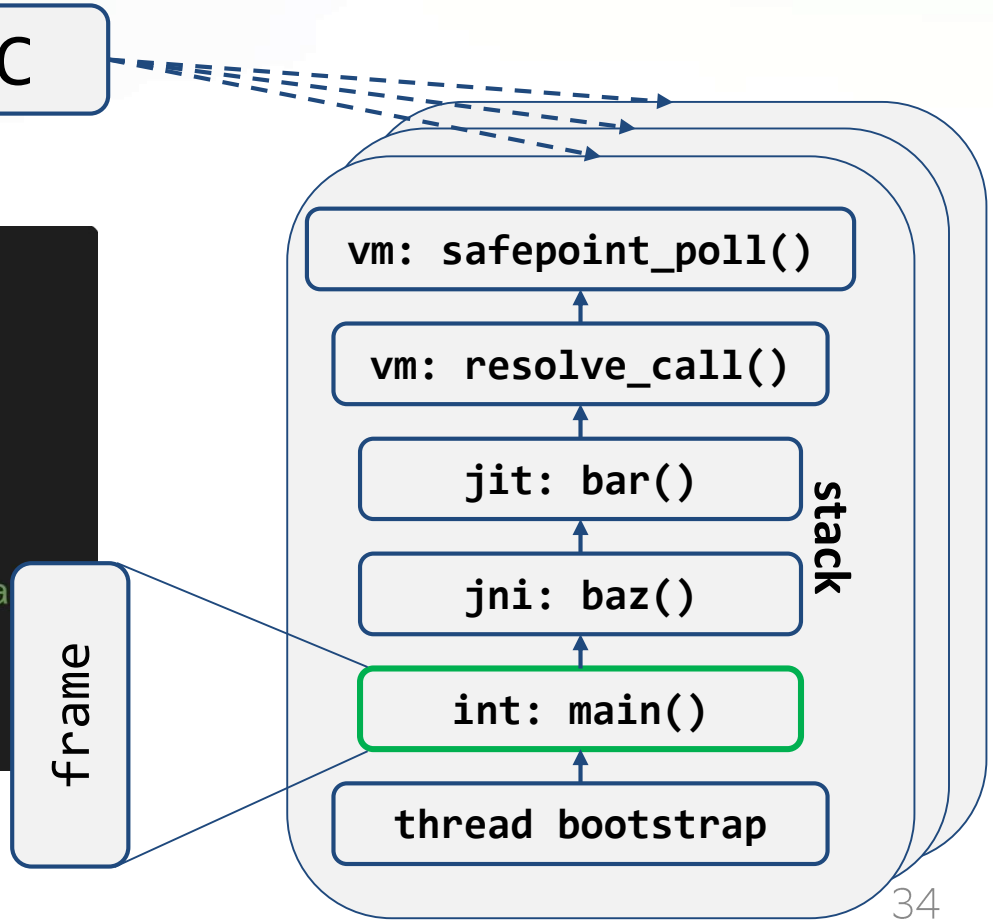
Do frame

```
1  void JavaThread::oops_do_frames(OopClosure* f,
2                                  CodeBlobClosure* cf) {
3    if (!has_last_Java_frame()) {
4      return;
5    }
6    ...
7    // Traverse the execution stack
8    for (StackFrameStream fst(this, ...);
9                              !fst.is_done();
10                             fst.next()) {
11     fst.current()->oops_do(f, cf, fst.register_map());
12   }
13 }
```

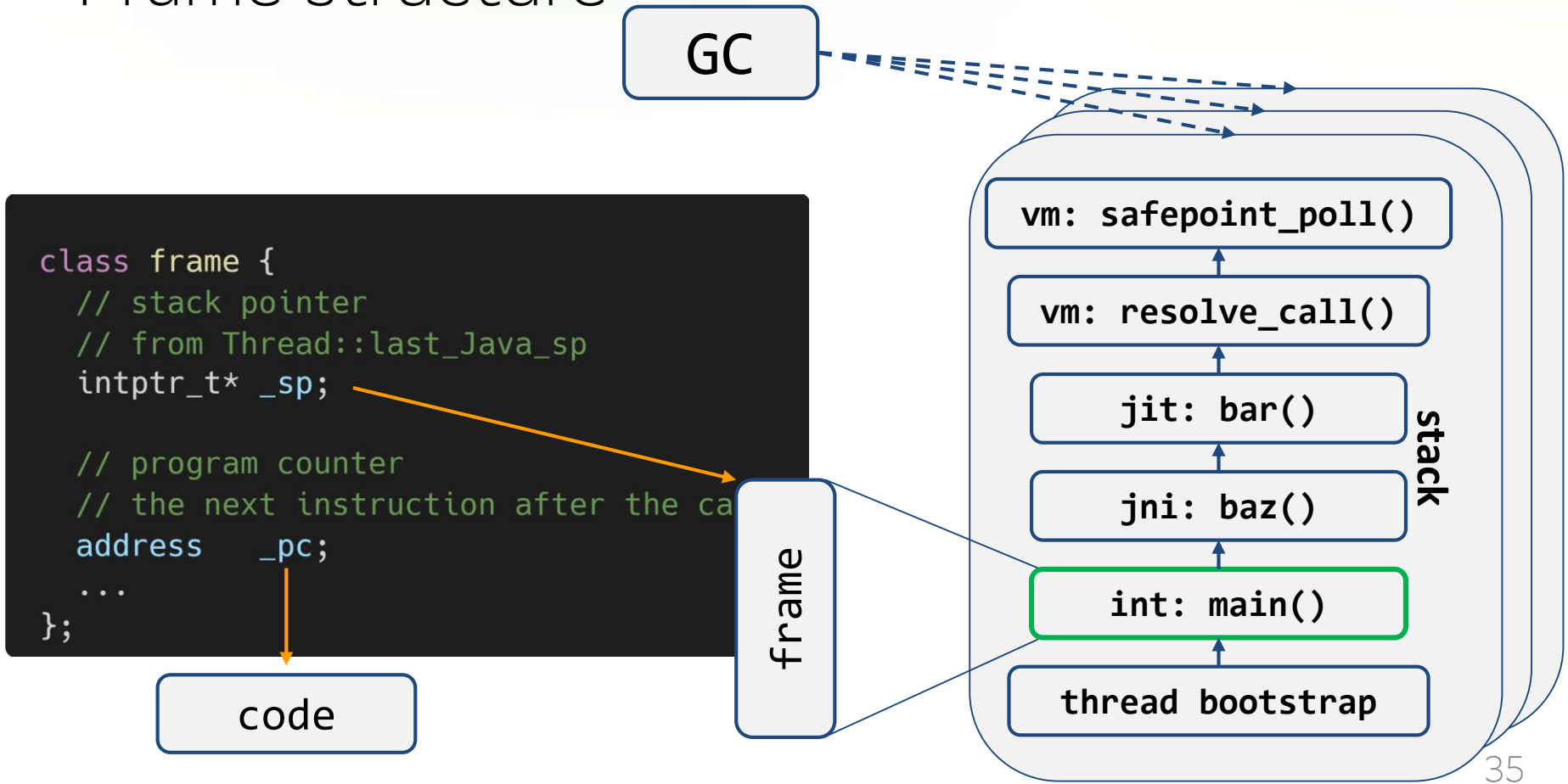


Frame structure

```
class frame {
  // stack pointer
  // from Thread::last_Java_sp
  intptr_t* _sp;

  // program counter
  // the next instruction after the ca
  address   _pc;
  ...
};
```

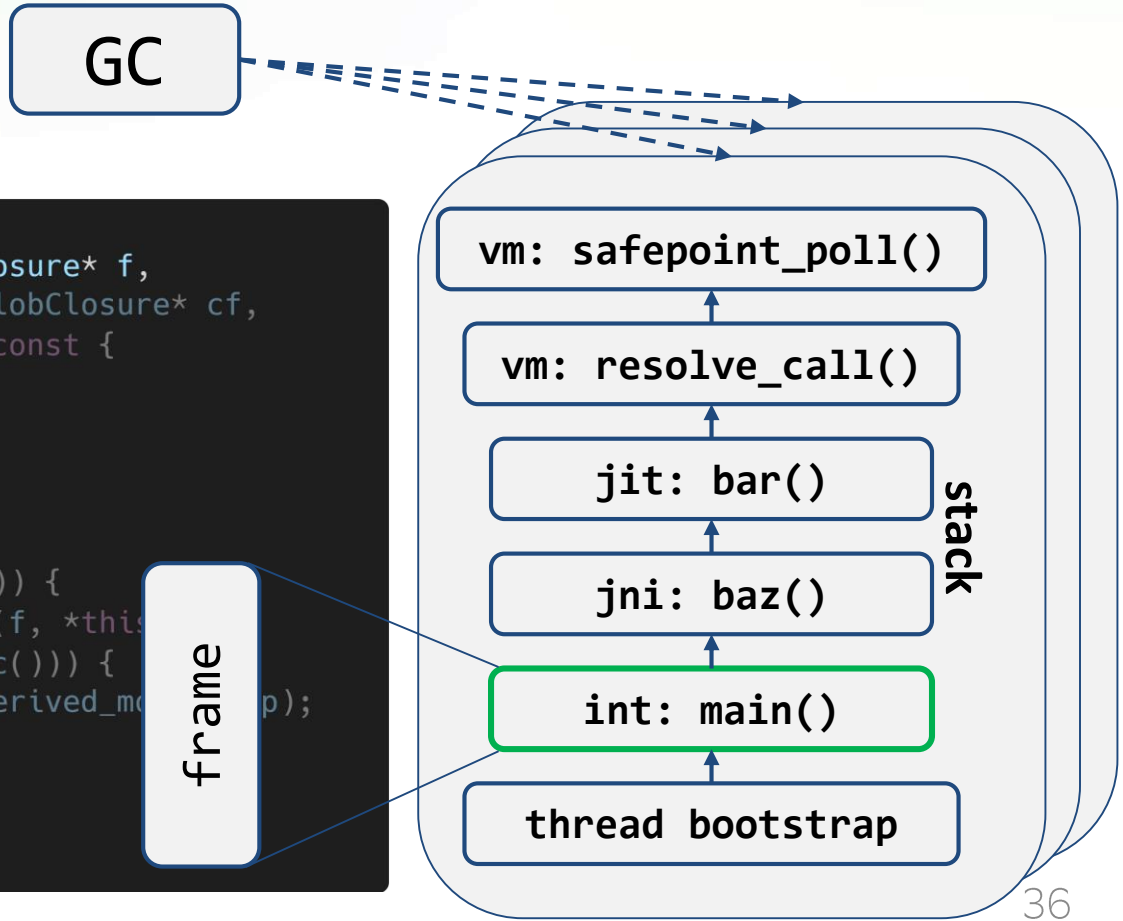


Frame structure

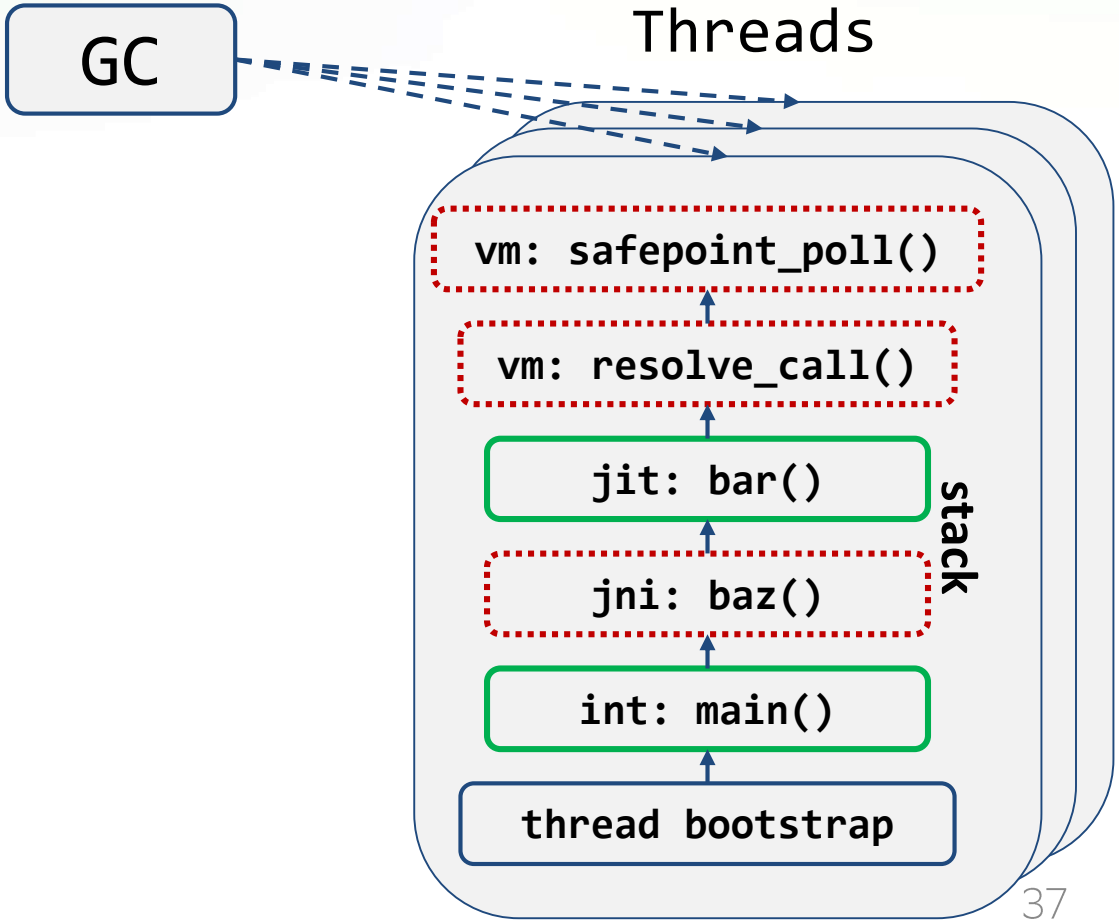


Inside frame

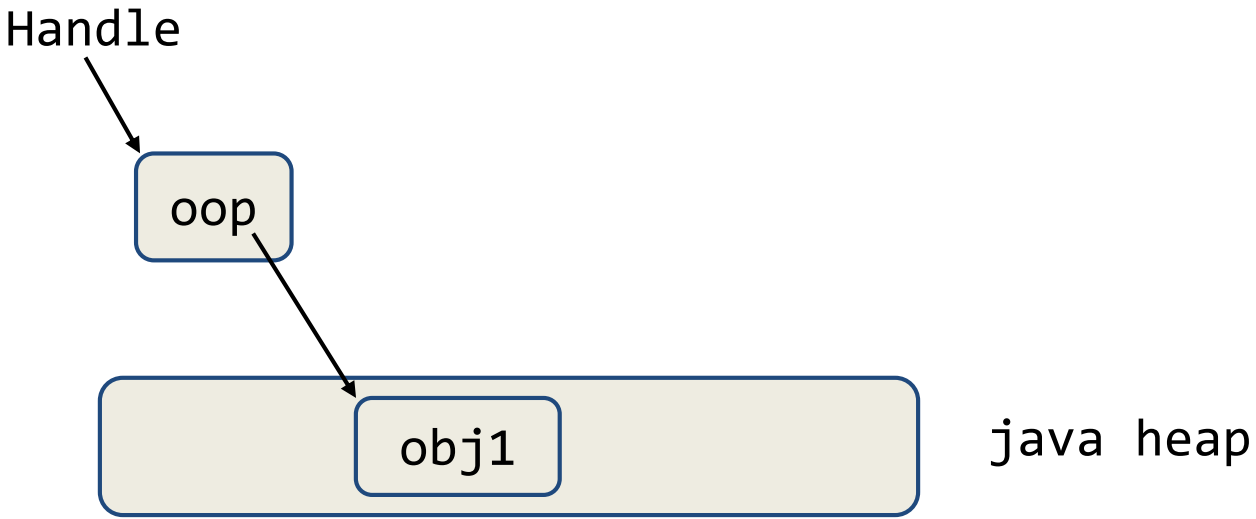
```
1 void frame::oops_do_internal(OopClosure* f,
2                               CodeBlobClosure* cf,
3                               ...) const {
4   ...
5   if (is_interpreted_frame()) {
6     oops_interpreted_do(f, ...);
7   } else if (is_entry_frame()) {
8     oops_entry_do(f, map);
9   } else if (is_upcall_stub_frame()) {
10    _cb->as_upcall_stub()->oops_do(f, *this
11  } else if (CodeCache::contains(pc())) {
12    oops_code_blob_do(f, cf, df, derived_m
13  } else {
14    ShouldNotReachHere();
15  }
16 }
```



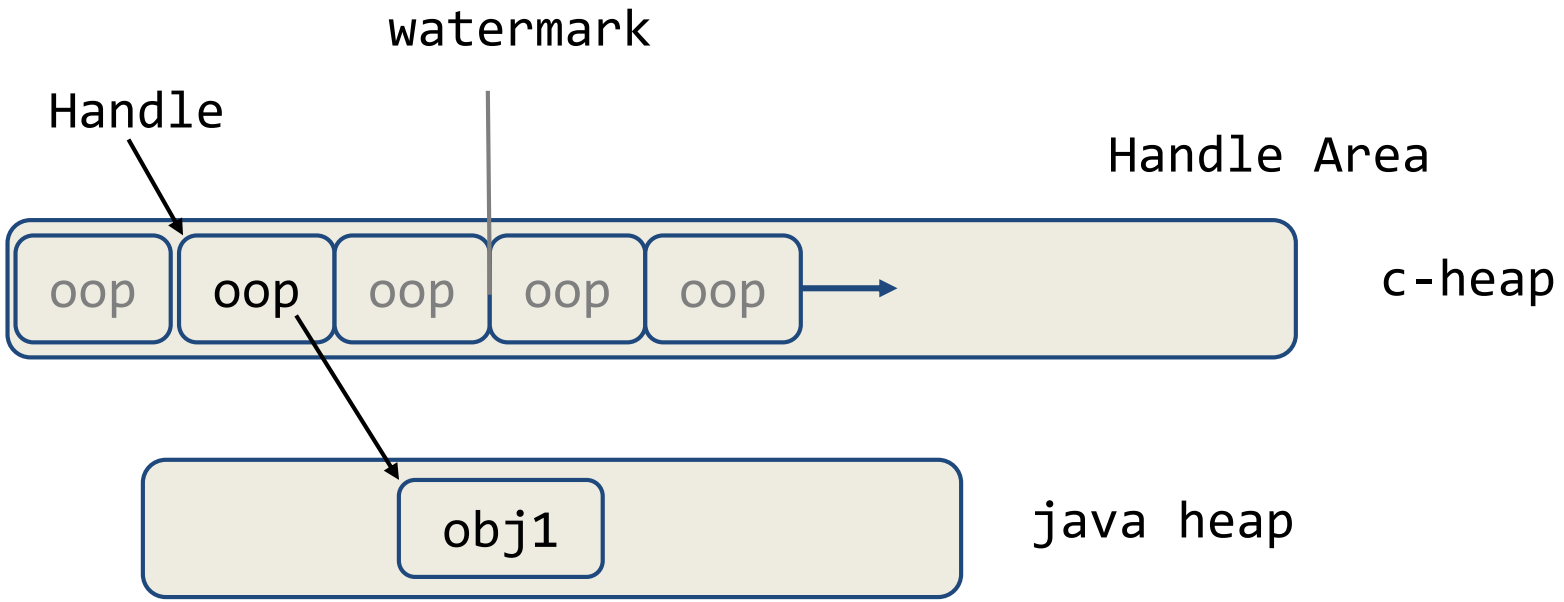
Thread stack



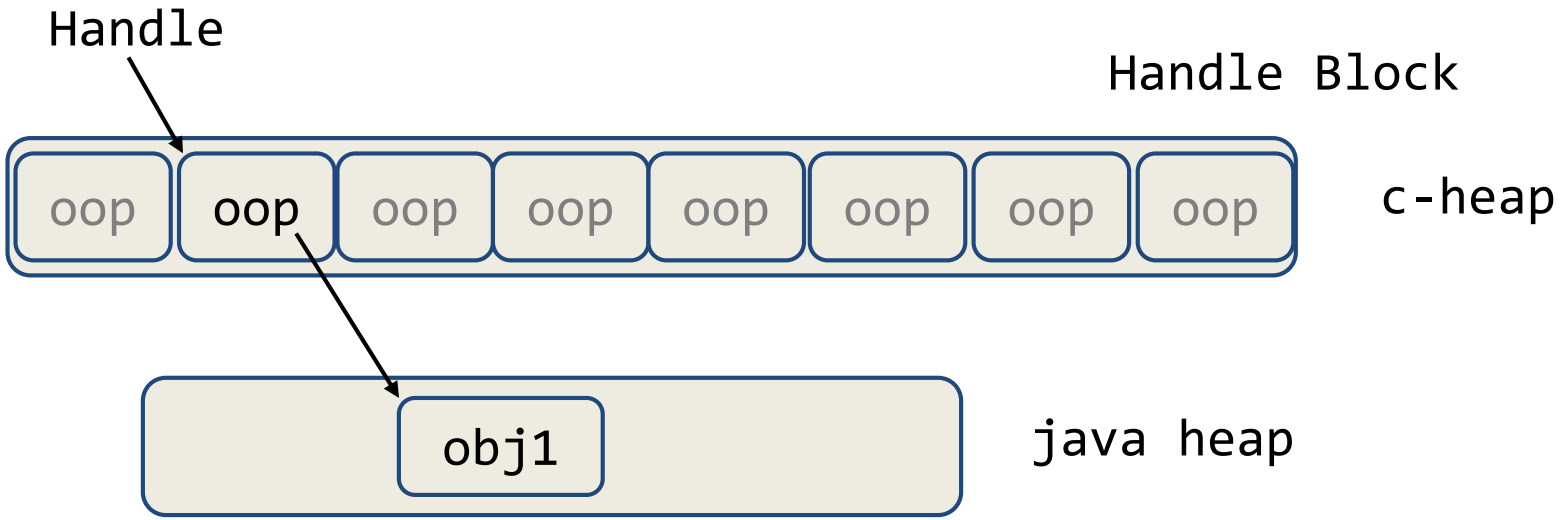
Handles



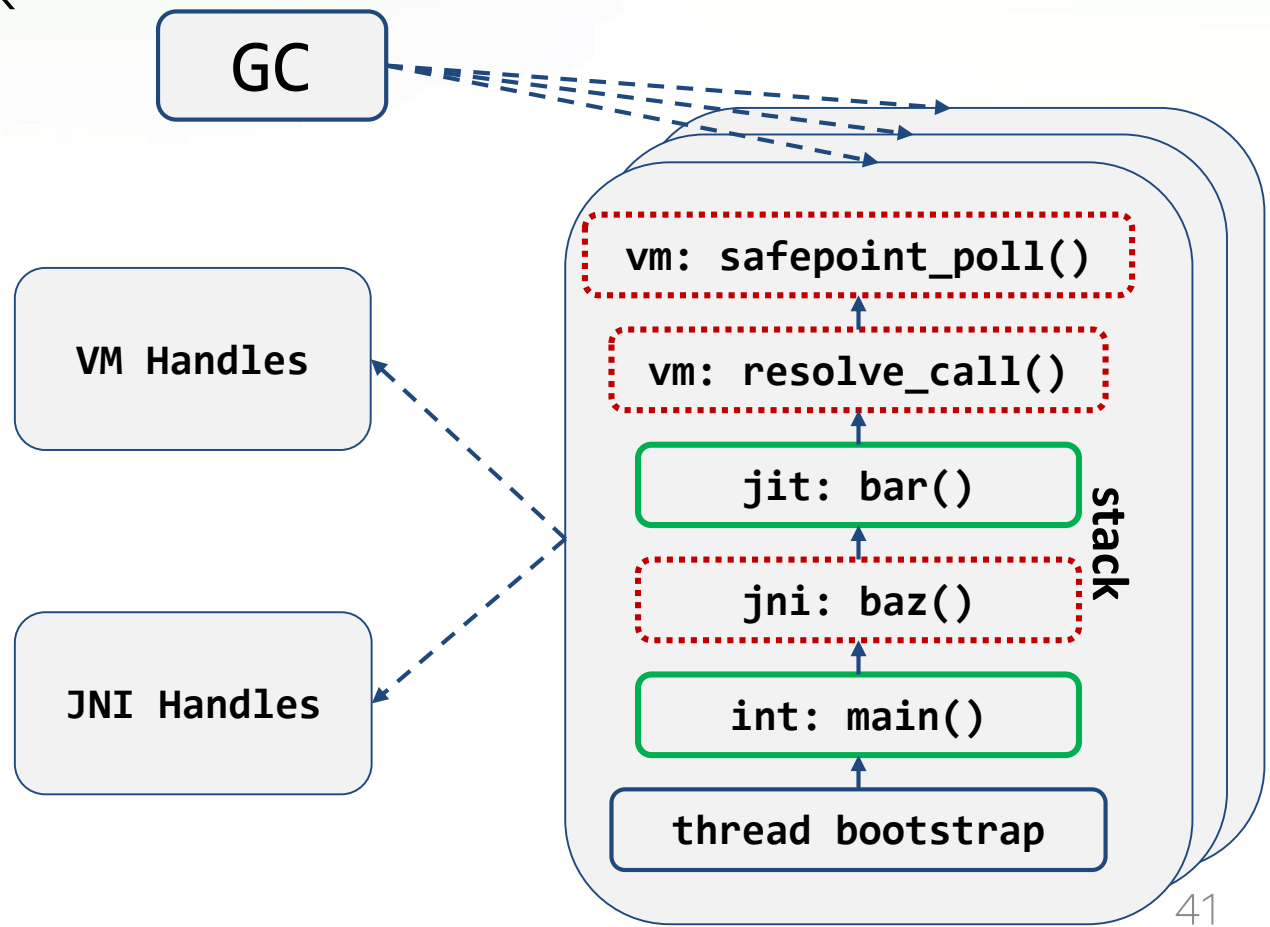
VM Handles


watermark
Handle
Handle Area
oop
oop
oop
oop
oop
c-heap
obj1
java heap

JNI Handles

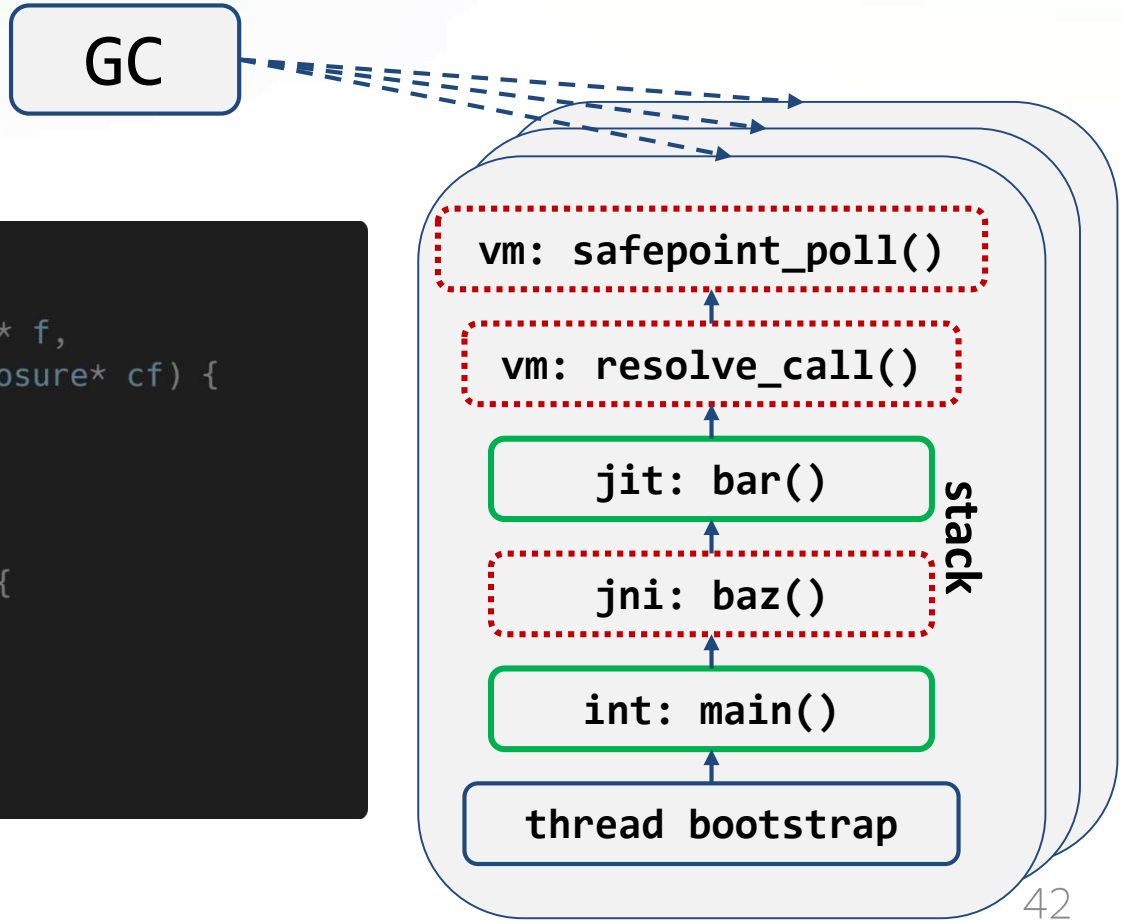

Handle
Handle Block
oop
oop
oop
oop
oop
oop
oop
oop
c-heap
obj1
java heap

Thread stack



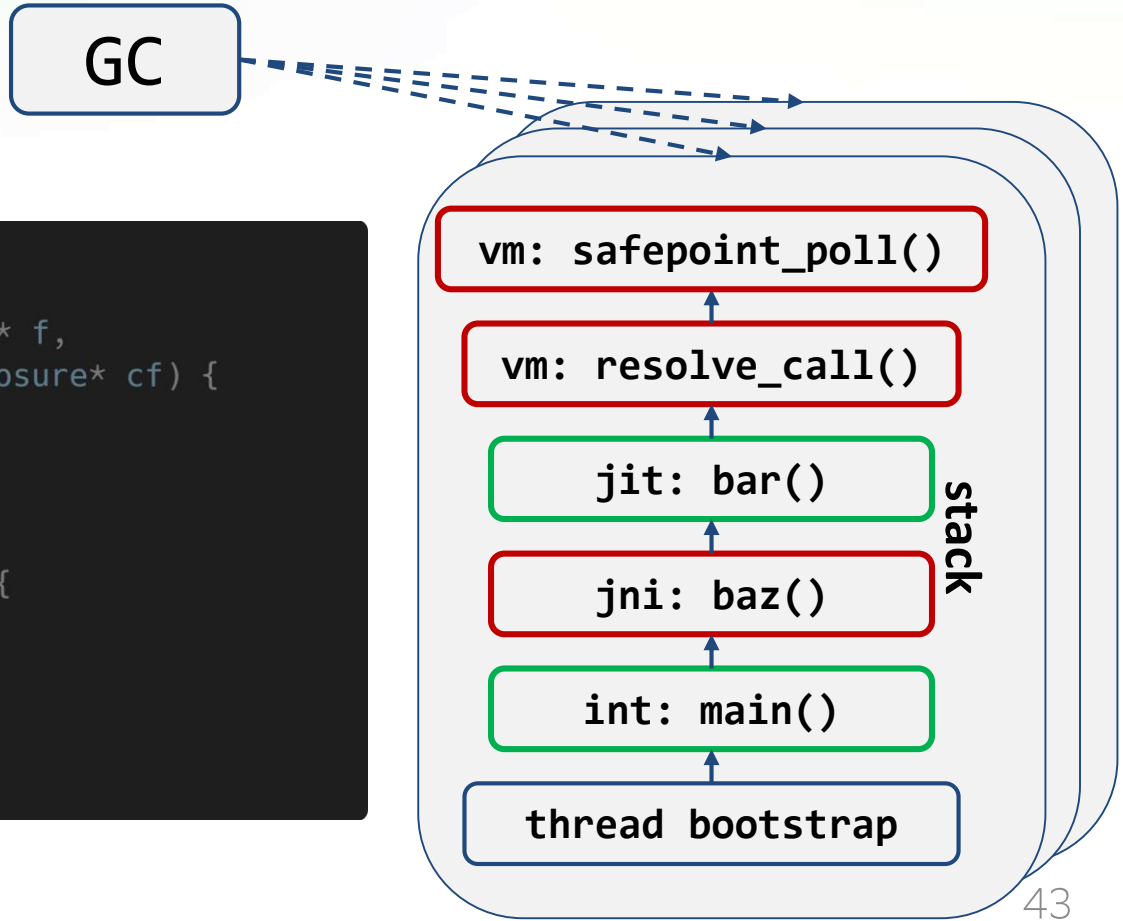
Do no frames

```
void JavaThread::oops_do_no_frames(
                          OopClosure* f,
                          CodeBlobClosure* cf) {
  ...
  // Traverse the GCHandles
  Thread::oops_do_no_frames(f, cf);

  if (active_handles() != nullptr) {
    active_handles()->oops_do(f);
  }
  ...
}
```



VM Handles

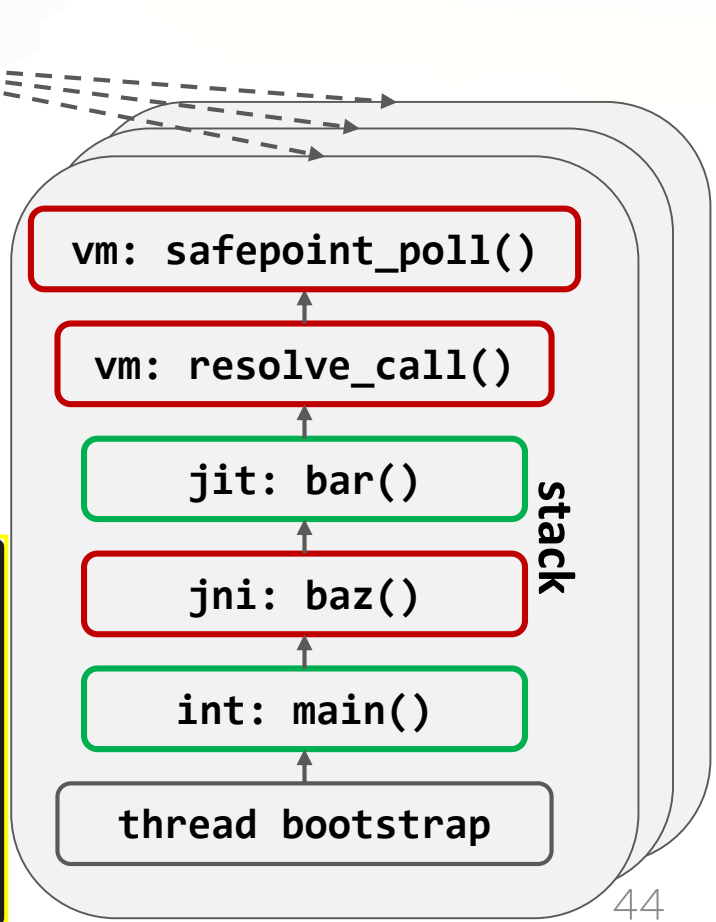
```
void JavaThread::oops_do_no_frames(
                          OopClosure* f,
                          CodeBlobClosure* cf) {
  ...
  // Traverse the GCHandles
  Thread::oops_do_no_frames(f, cf);

  if (active_handles() != nullptr) {
    active_handles()->oops_do(f);
  }
  ...
}
```



VM Handles

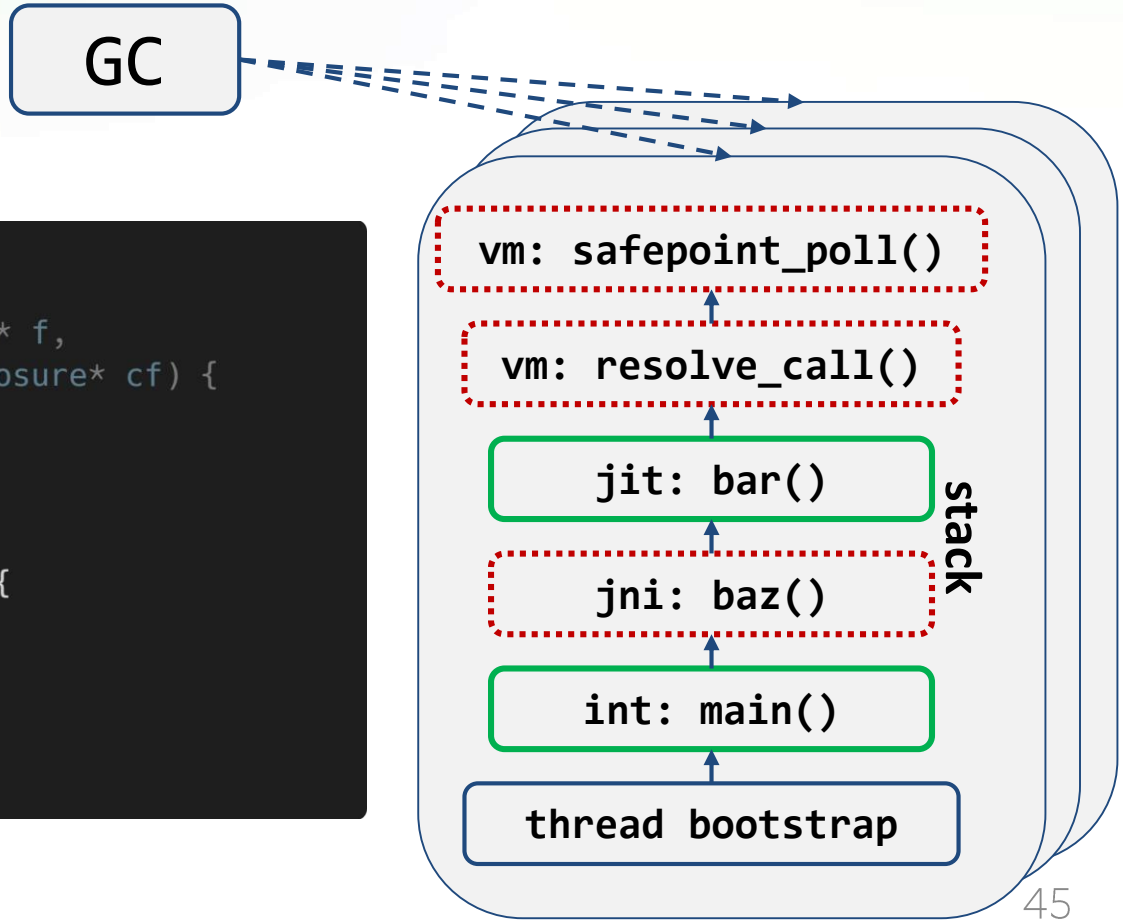
```
void JavaThread::oops_do_no_frames(
                        OopClosure* f,
                        CodeBlobClosure* cf) {
  ...
  // Traverse the GCHandles
  Thread::oops_do_no_frames(f, cf);
```

```
void Thread::oops_do_no_frames(
                        OopClosure* f,
                        CodeBlobClosure* cf) {
  // Do oop for ThreadShadow
  f->do_oop((oop*)&_pending_exception);
  handle_area()->oops_do(f);
}
```



JNI Handles

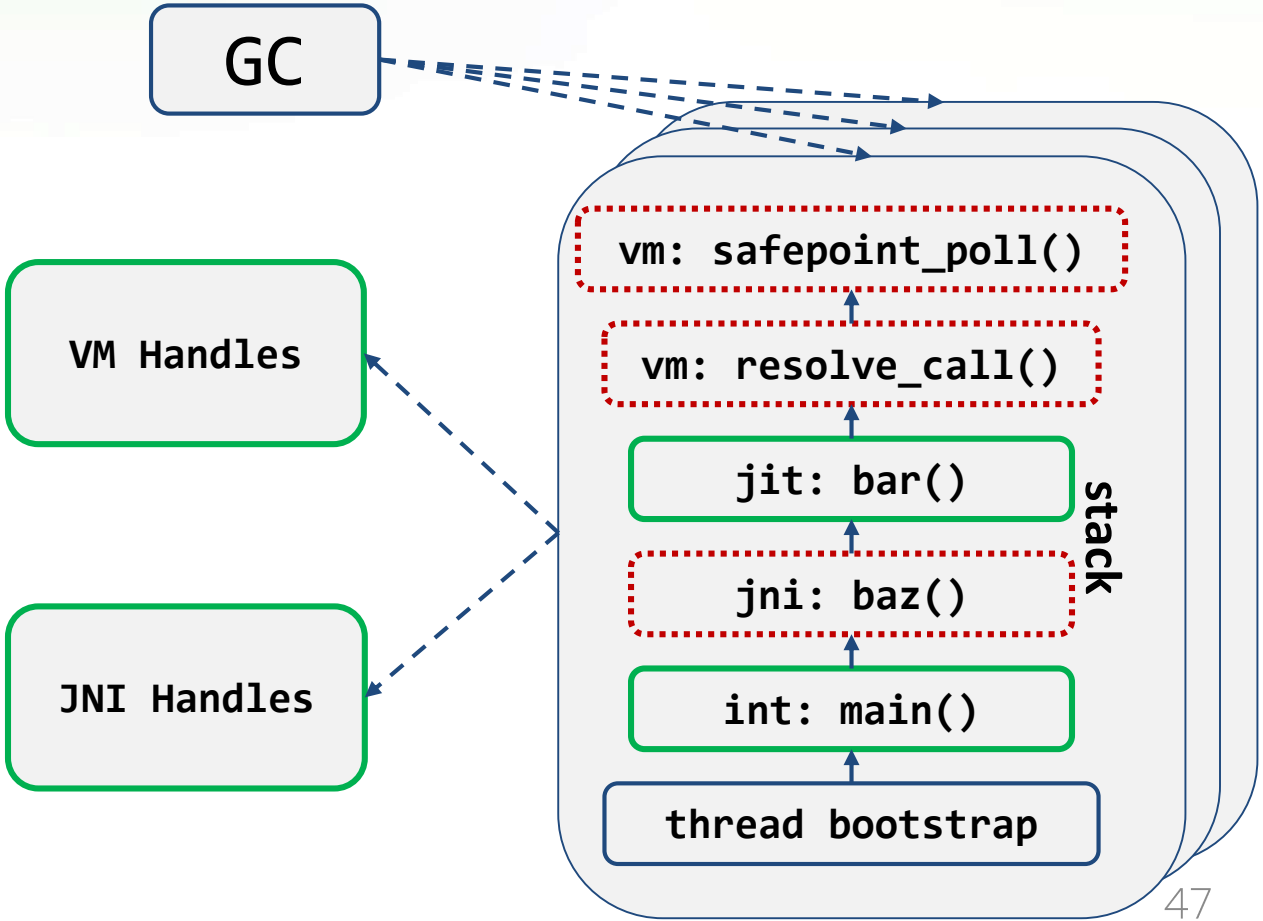
```
void JavaThread::oops_do_no_frames(
                          OopClosure* f,
                          CodeBlobClosure* cf) {
  ...
  // Traverse the GCHandles
  Thread::oops_do_no_frames(f, cf);

  if (active_handles() != nullptr) {
    active_handles()->oops_do(f);
  }
  ...
}
```



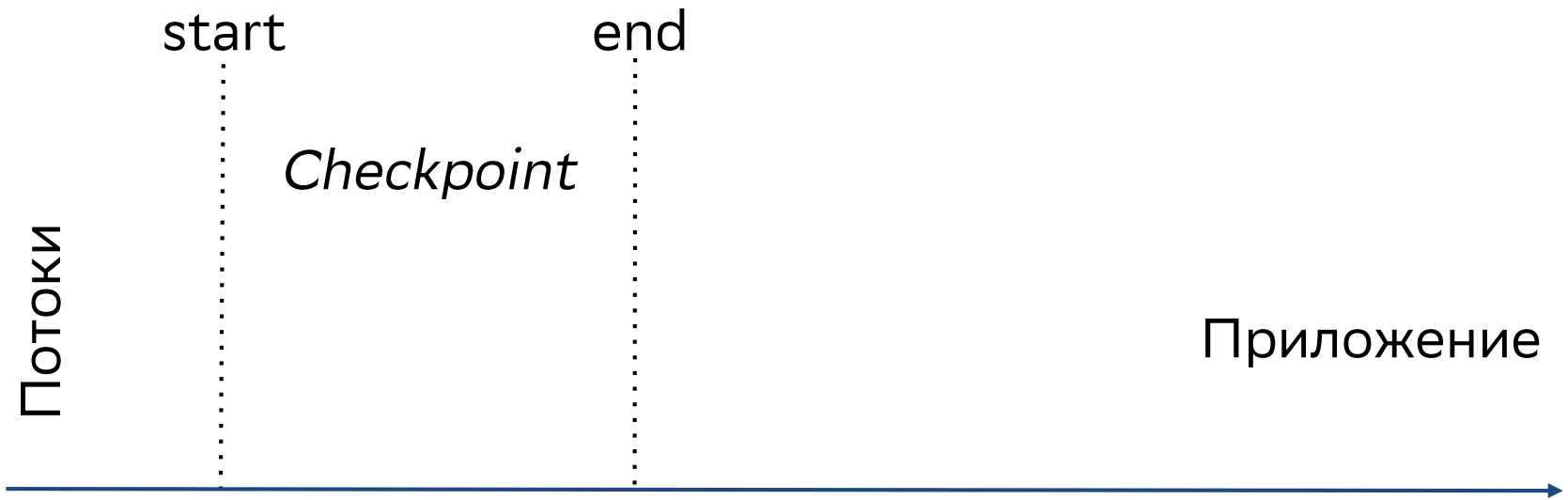
JNI Handles


GC
void JavaThread:
...
// Traverse th
Thread::oops_d
if (active_handles() != nullptr) {
active_handles()->oops_do(f);
}
...
}
class JavaThread: public Thread {
...
// Active_handles points to a block of handles
JNIHandleBlock* _active_handles;
...
}
int_poll()
ve_call()
bar()
stack
jni: baz()
int: main()
thread bootstrap

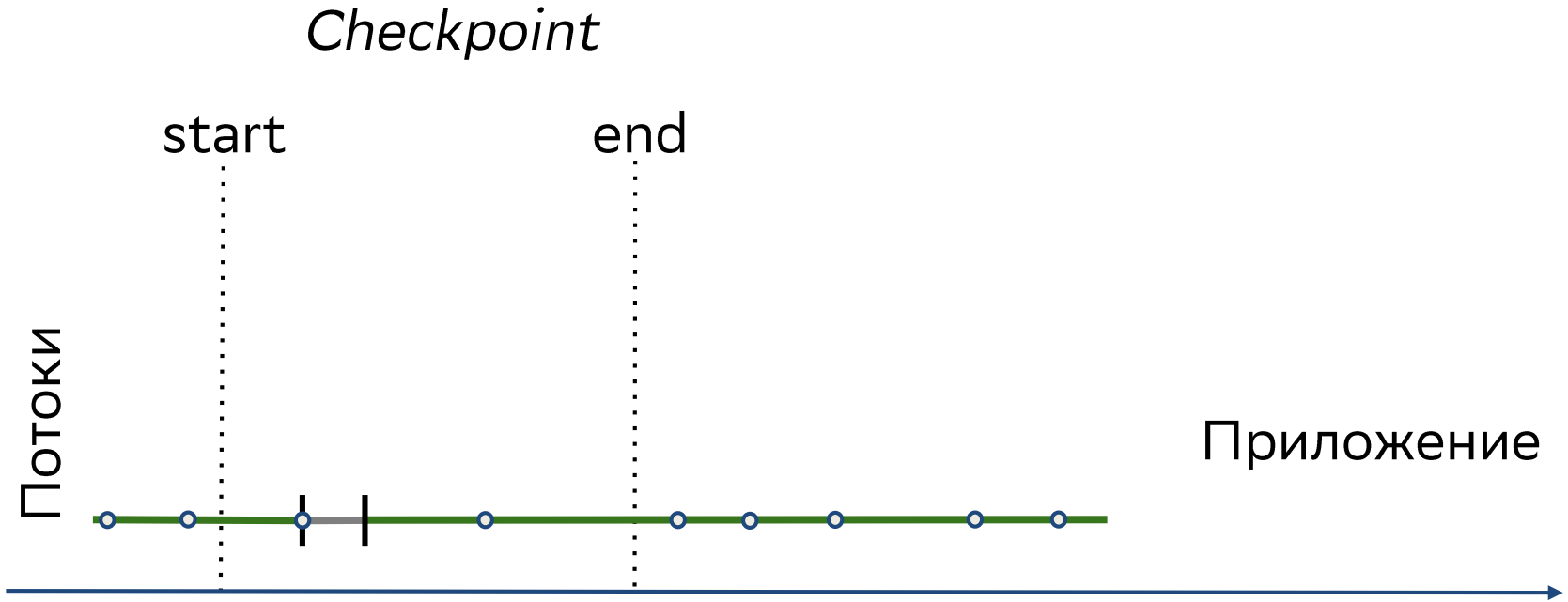
Done


GC
VM Handles
JNI Handles
vm: safepoint_poll()
vm: resolve_call()
jit: bar()
jni: baz()
int: main()
thread bootstrap
stack

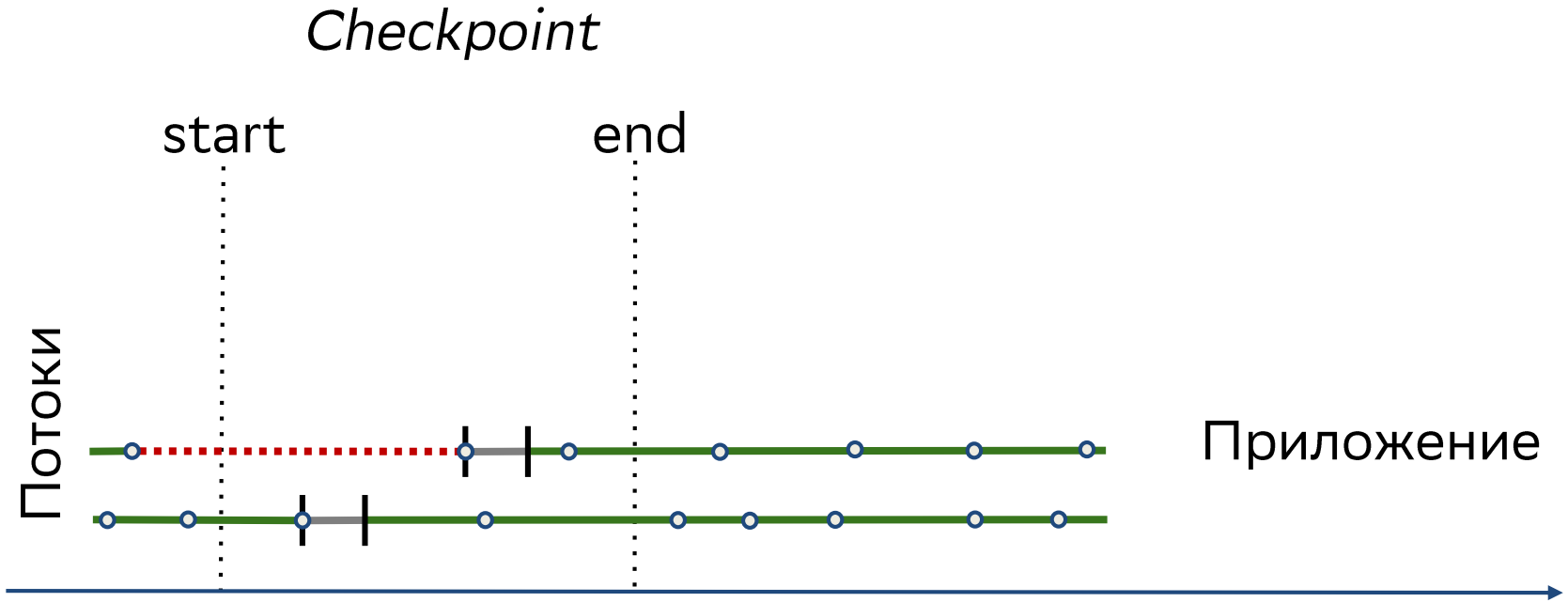
Checkpoint = thread-local handshake (JEP 312)



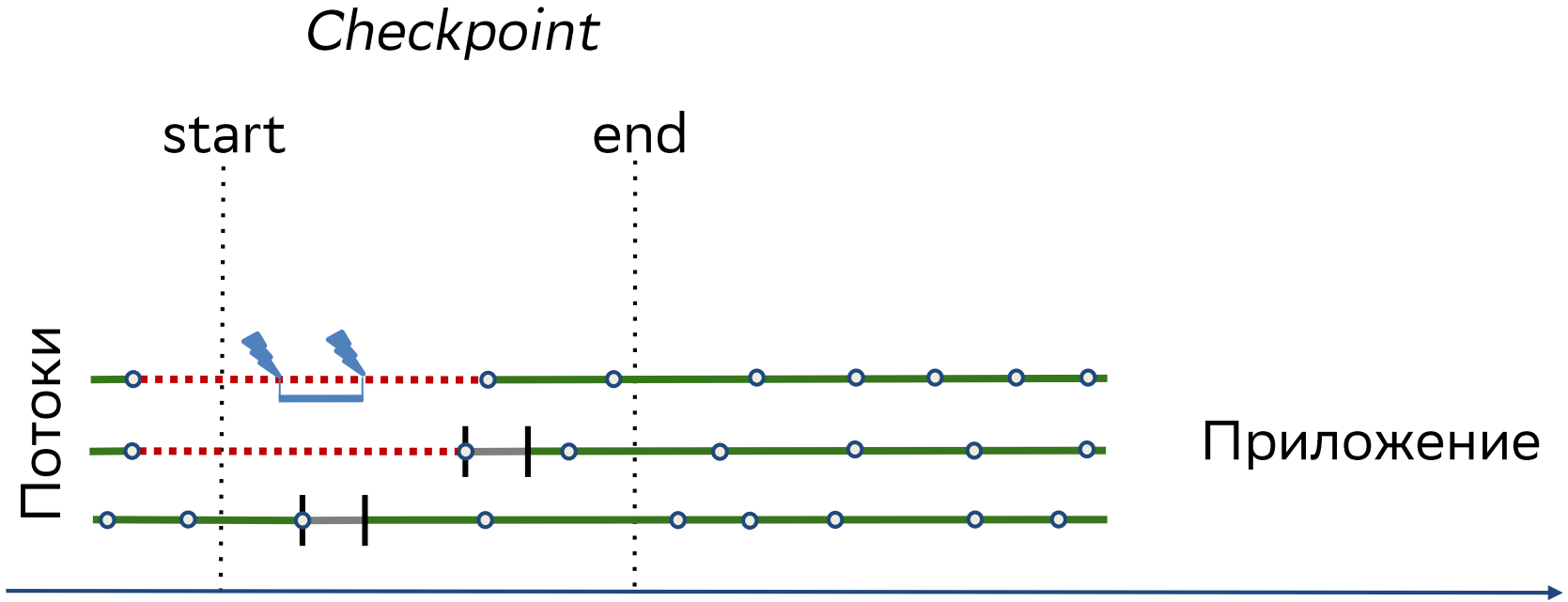
Checkpoint = thread-local handshake (JEP 312)



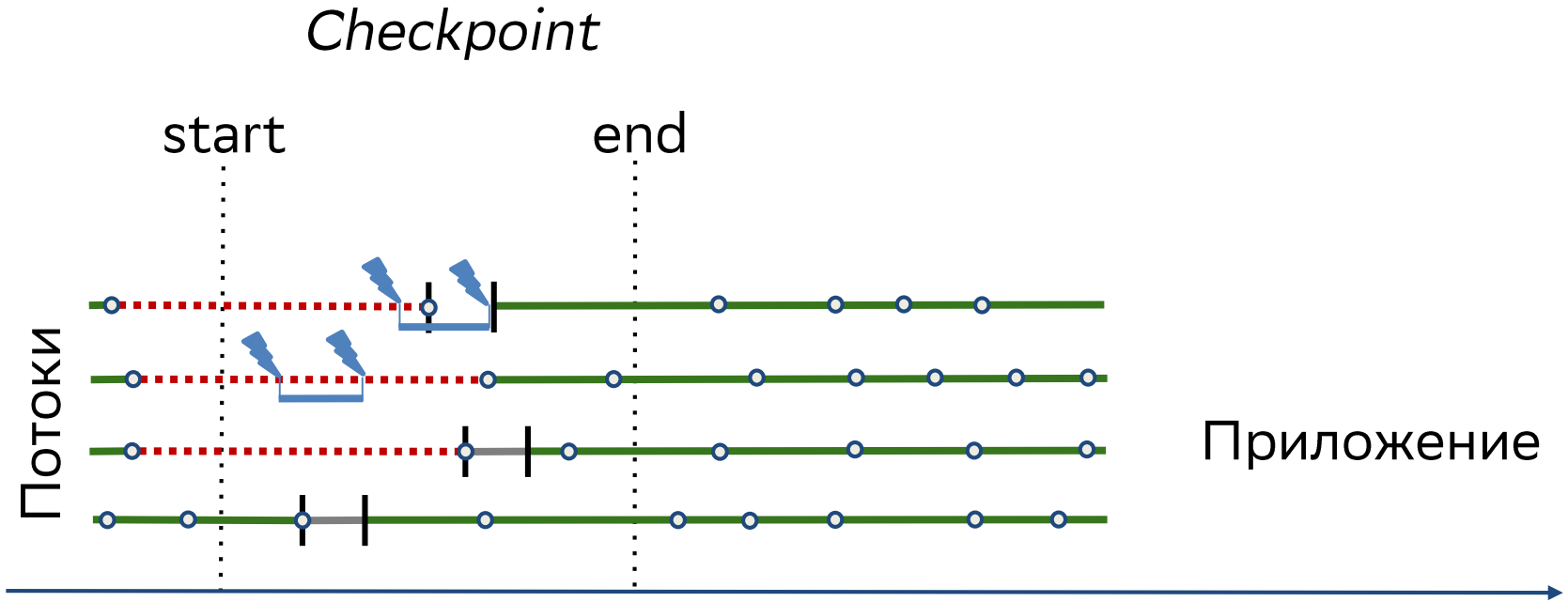
Checkpoint = thread-local handshake (JEP 312)



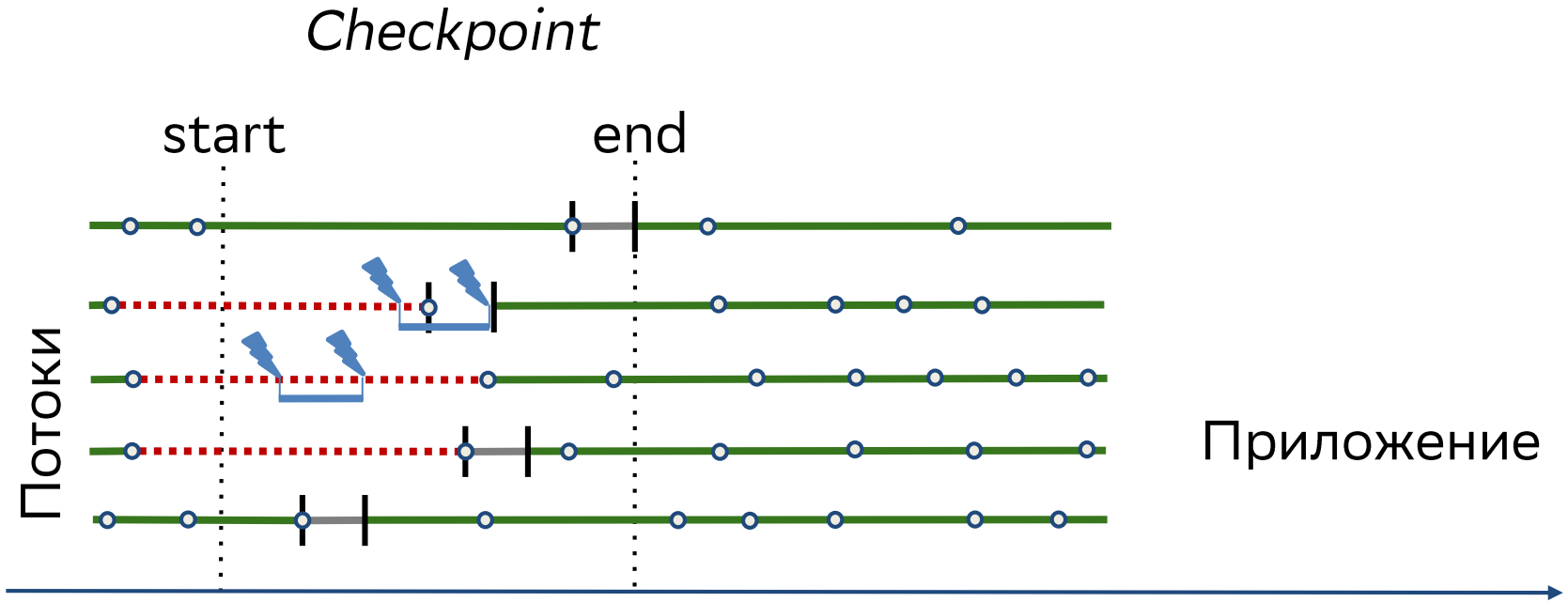
Checkpoint = thread-local handshake (JEP 312)



Checkpoint = thread-local handshake (JEP 312)



Checkpoint = thread-local handshake (JEP 312)



Многопоточное программирование

synchronized

`mutex`

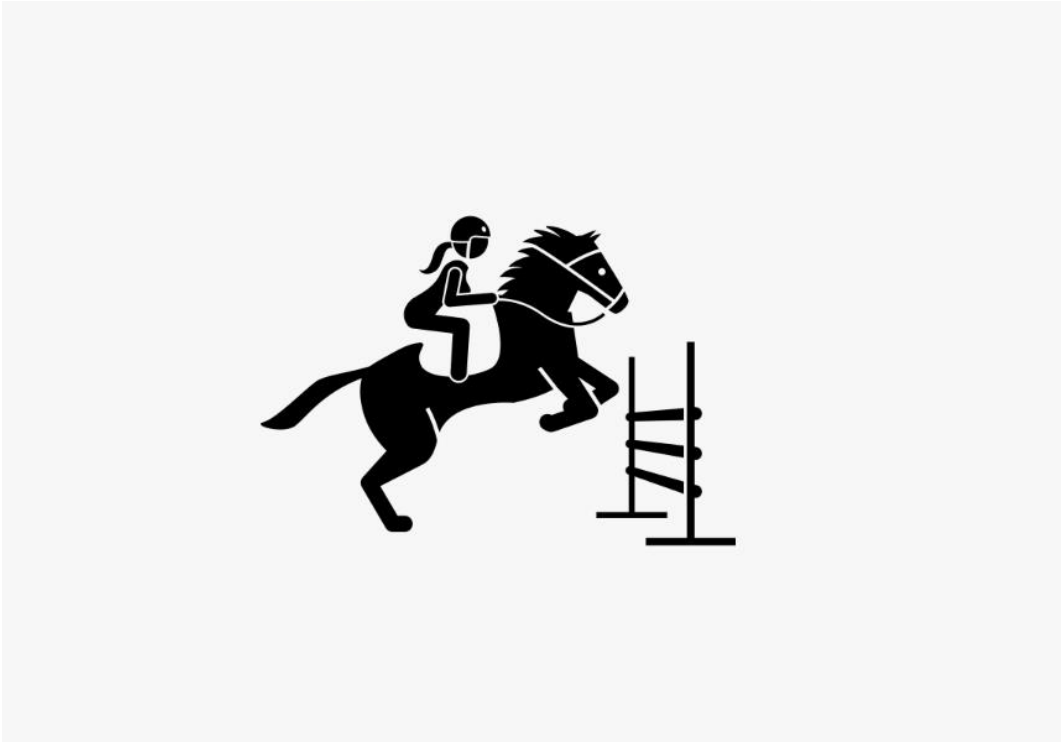
safepoint

lock-free

`barrier`

concurrent

Barrier



- synchronization point?
- memory ordering?
- **extra action!**

VM barriers

```
oop load_oop(oop* addr) {
    oop x = *addr;
    ... load barrier ...
    return x;
}

void store_oop(oop val, oop* addr)
{   ... pre-store barrier ...
    *addr = val;
    ... post-store barrier ...
}
```

VM barriers

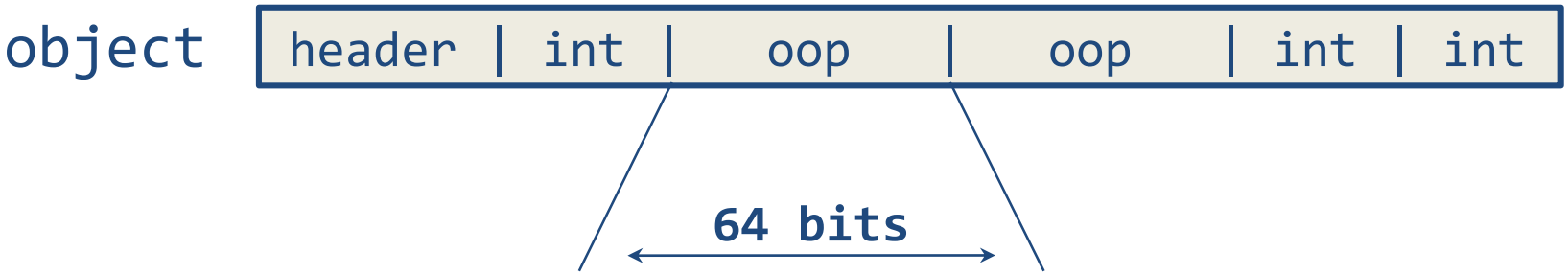
```
oop load_oop(oop* addr) {
    oop x = *addr;
    ... load barrier ...
    return x;
}

void store_oop(oop val, oop* addr)
{   ... pre-store barrier ...
    *addr = val;
    ... post-store barrier ...
}
```

VM barriers

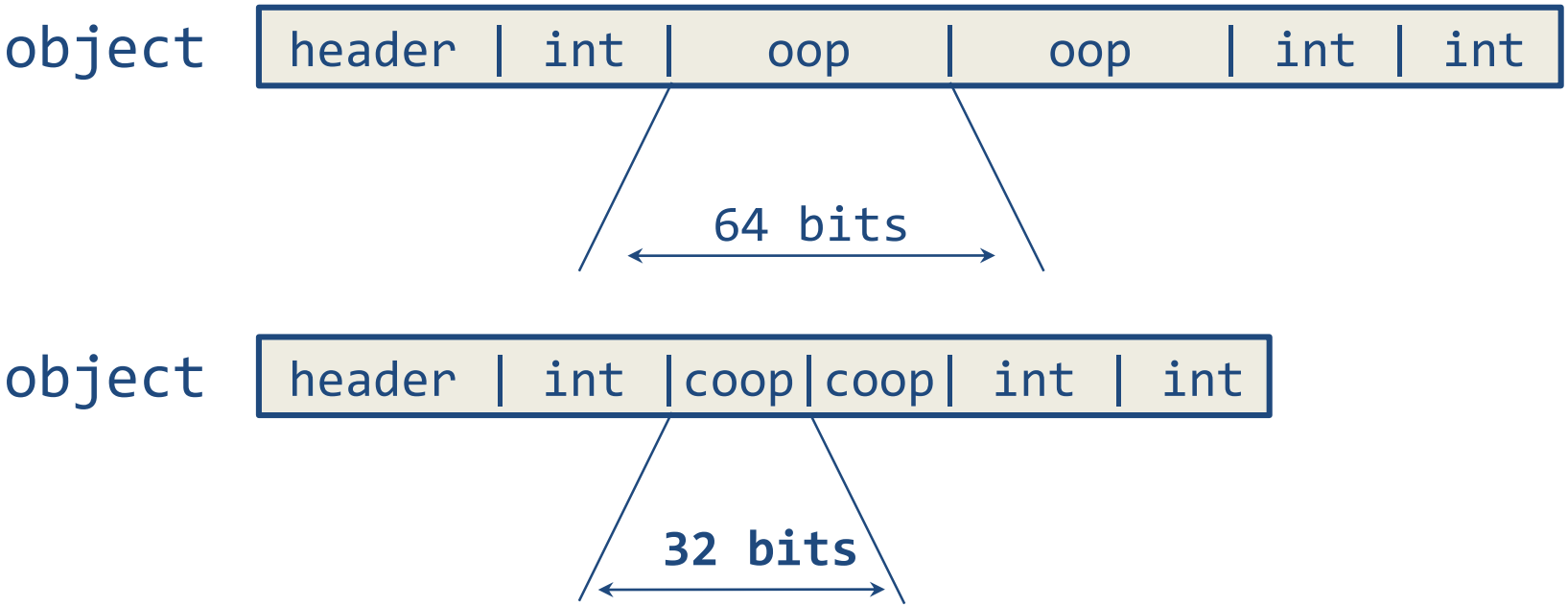
```
oop load_oop(oop* addr) {
    oop x = *addr;
    ... load barrier ...
    return x;
}

void store_oop(oop val, oop* addr)
{   ... pre-store barrier ...
    *addr = val;
    ... post-store barrier ...
}
```

Compressed oops

Compressed oops

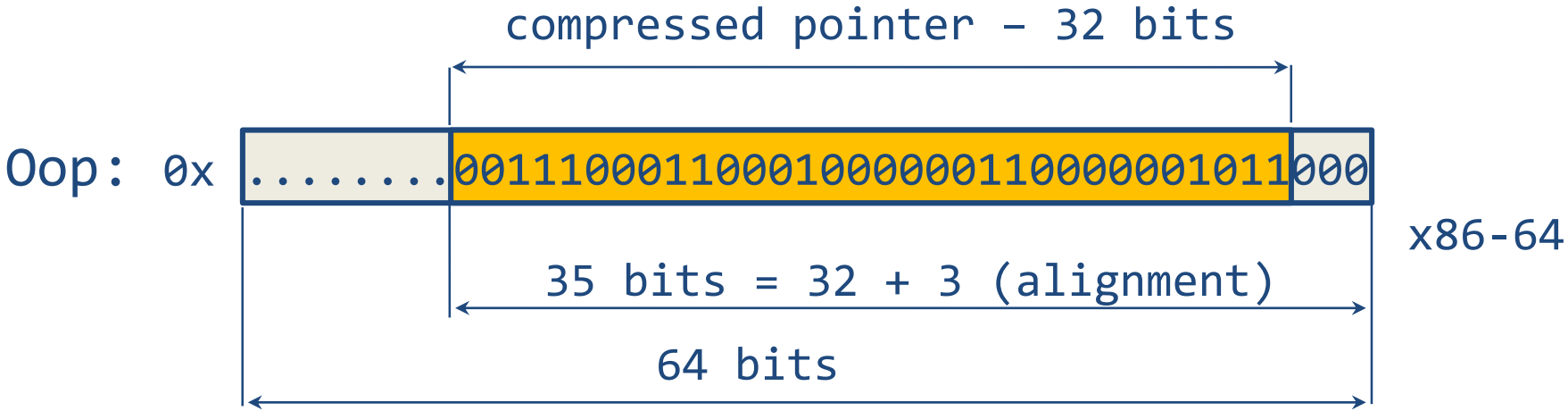


Compressed oops

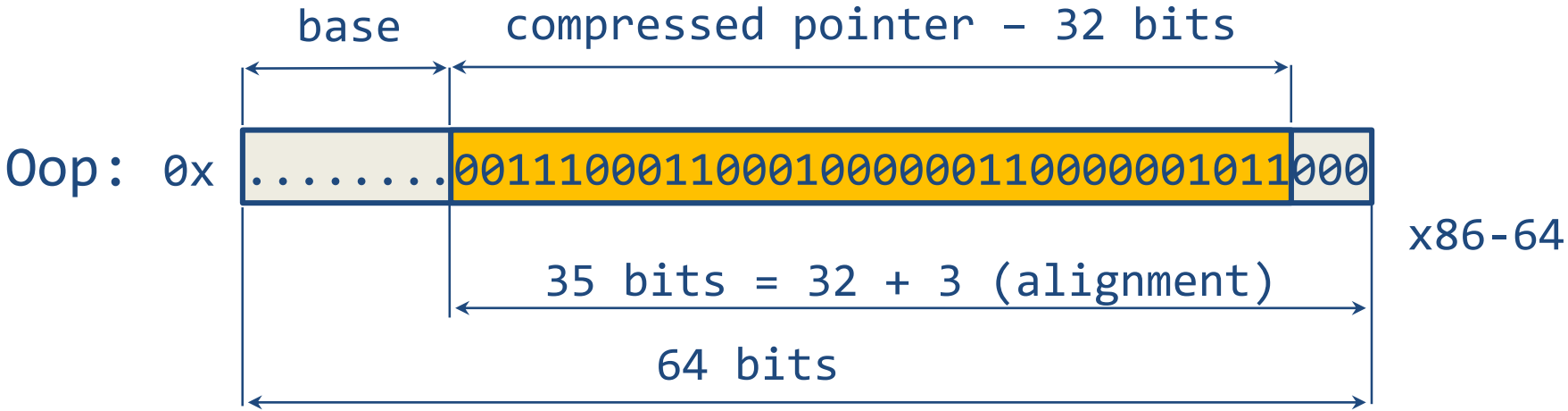


Compressed oops

Compressed oops



Compressed oops



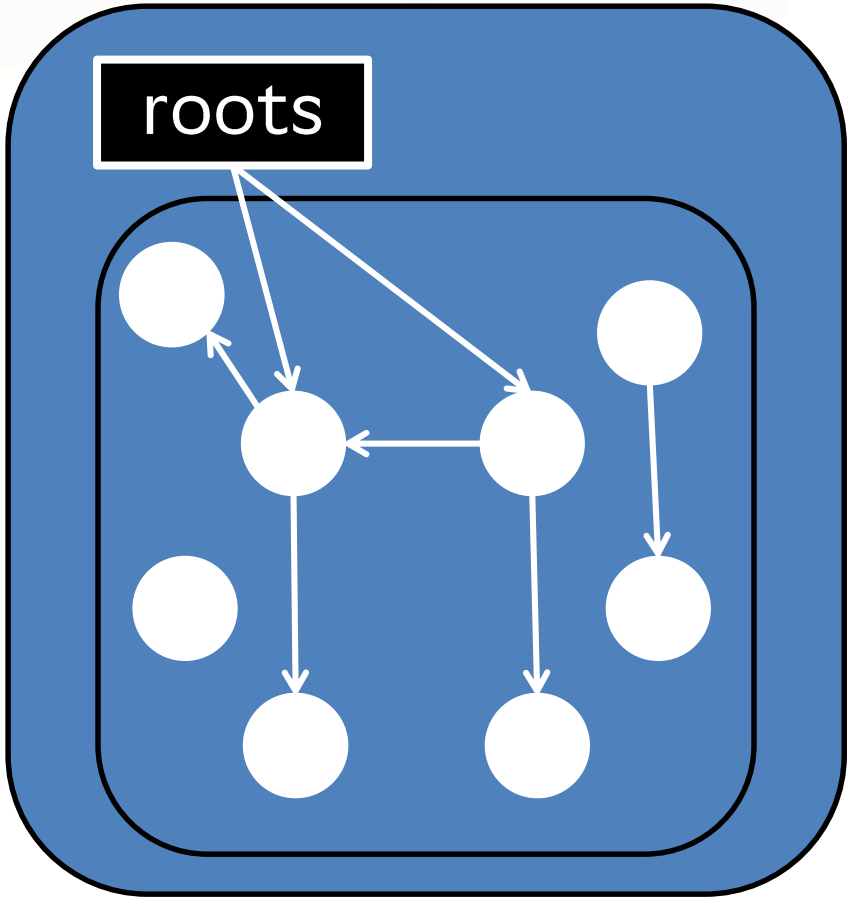
Compressed oops barrier

```
oop load_oop(narrowOop* addr) {
    narrowOop x = *addr;
    oop val = CompressedOops::base()                read barrier
              + ((oop)x << CompressedOops::shift())
    return val;
}

void store_oop(oop val, oop* addr) {
    narrowOop x = (val - CompressedOops::base())
                  >> CompressedOops::shift();
    *addr = x;
}
```

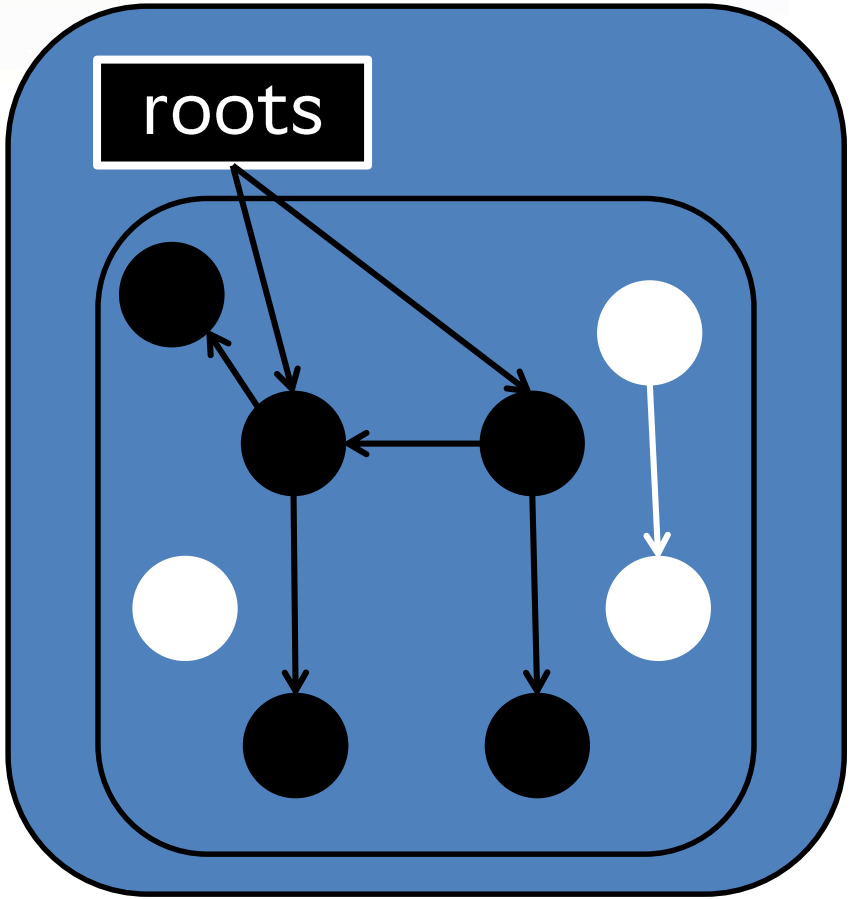
Compressed oops barrier

```
oop load_oop(narrowOop* addr) {
    narrowOop x = *addr;
    oop val = CompressedOops::base()                    read barrier
              + ((oop)x << CompressedOops::shift())
    return val;
}

void store_oop(oop val, oop* addr) {                    pre-write barrier
    narrowOop x = (val - CompressedOops::base())
                  >> CompressedOops::shift();
    *addr = x;
}
```

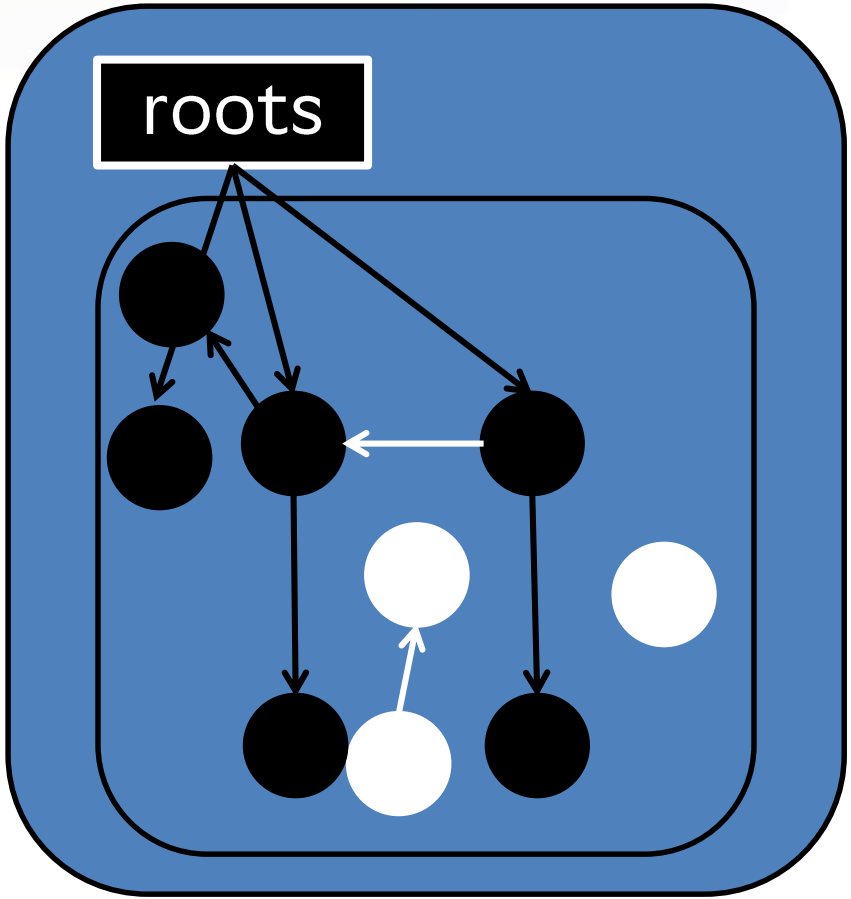
Cross generation references



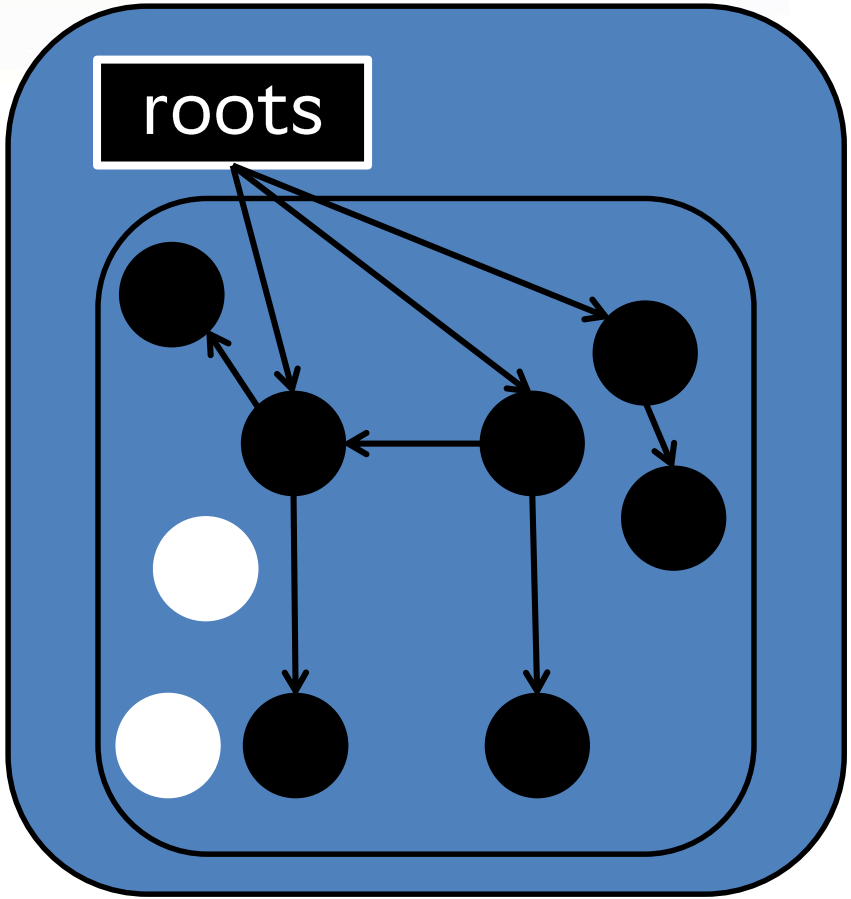
Cross generation references



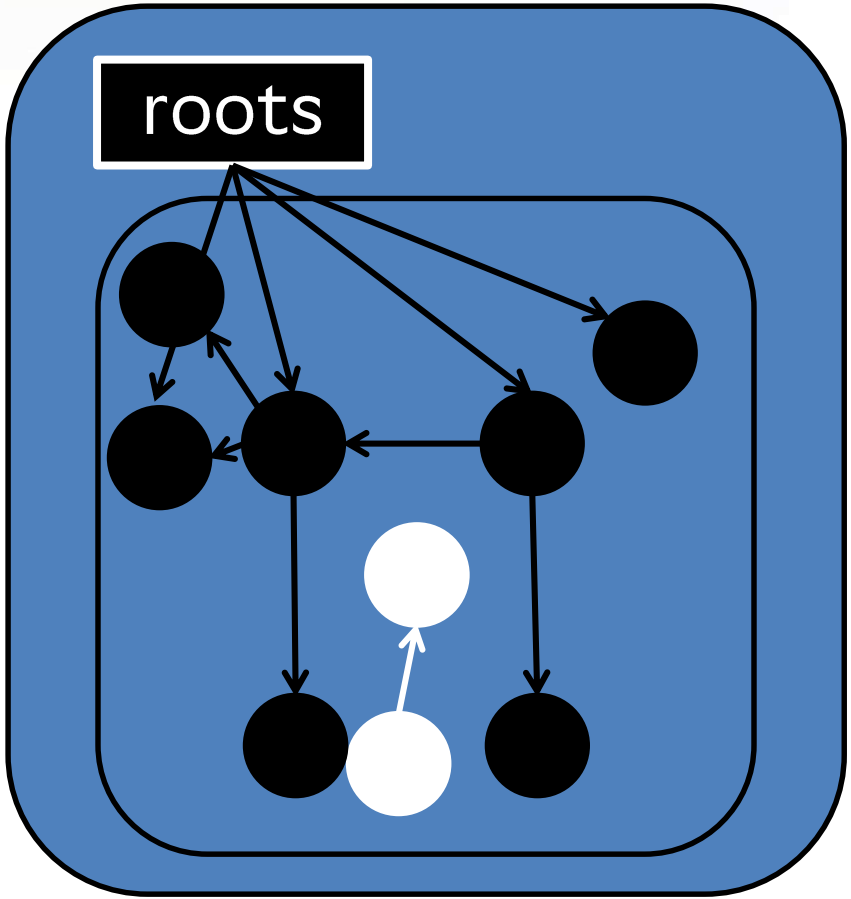
Cross generation references



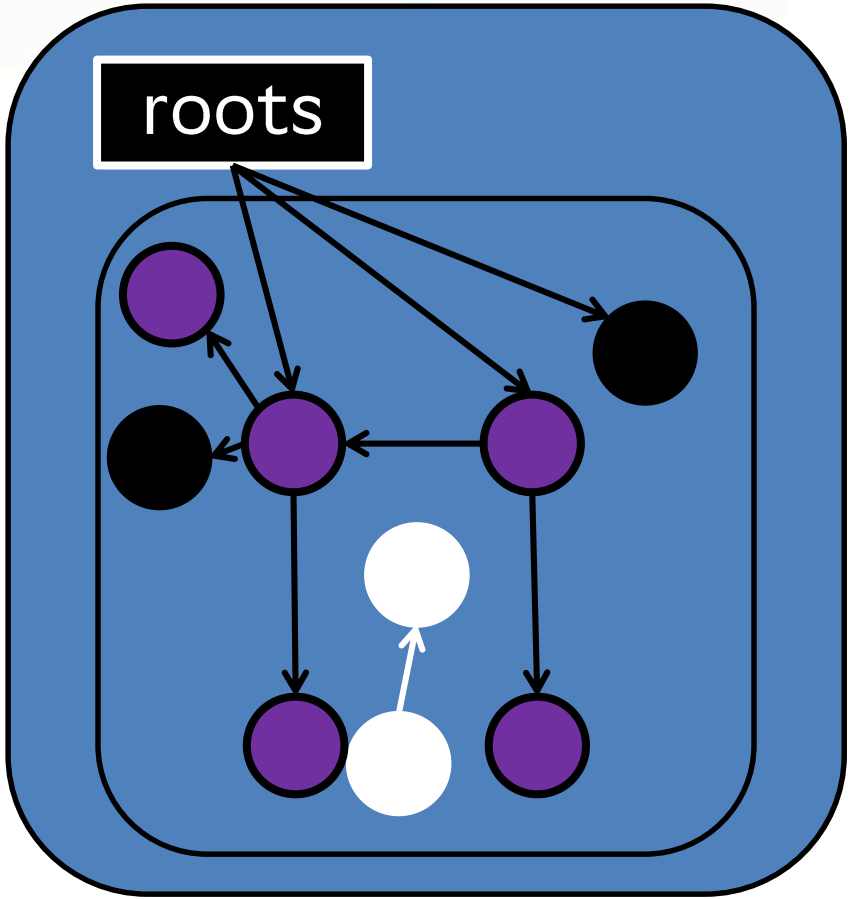
Cross generation references



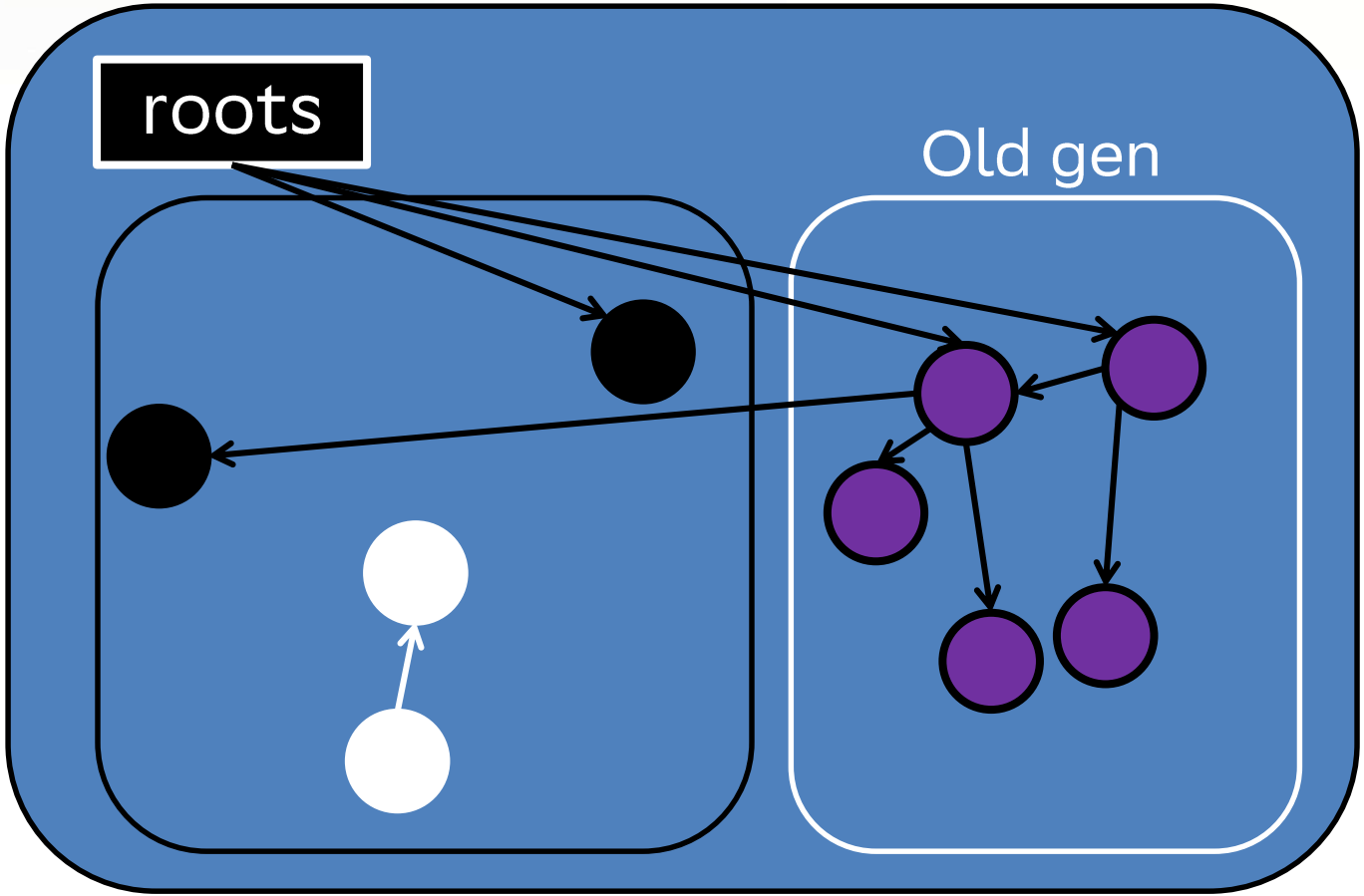
Cross generation references



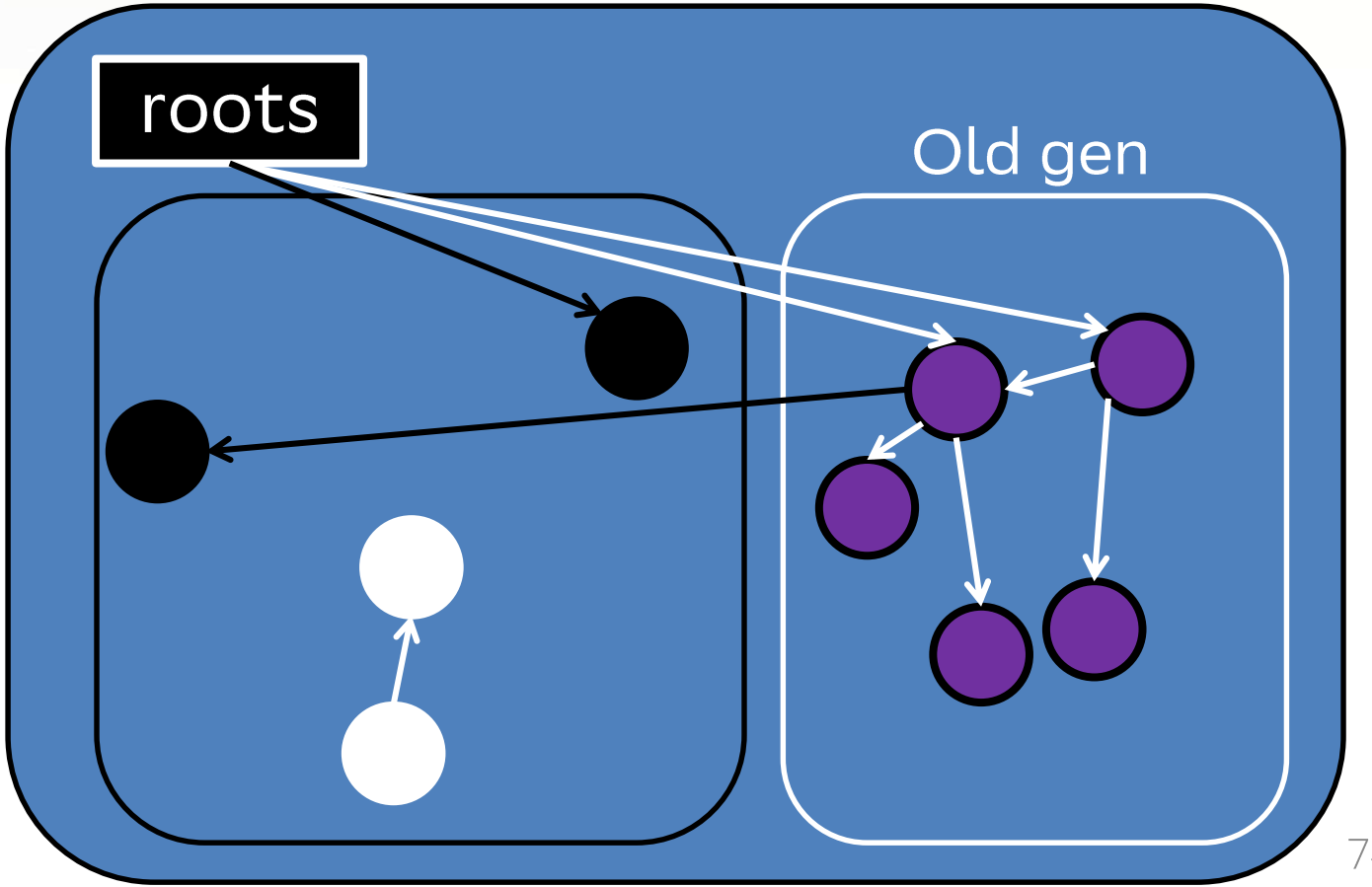
Cross generation references



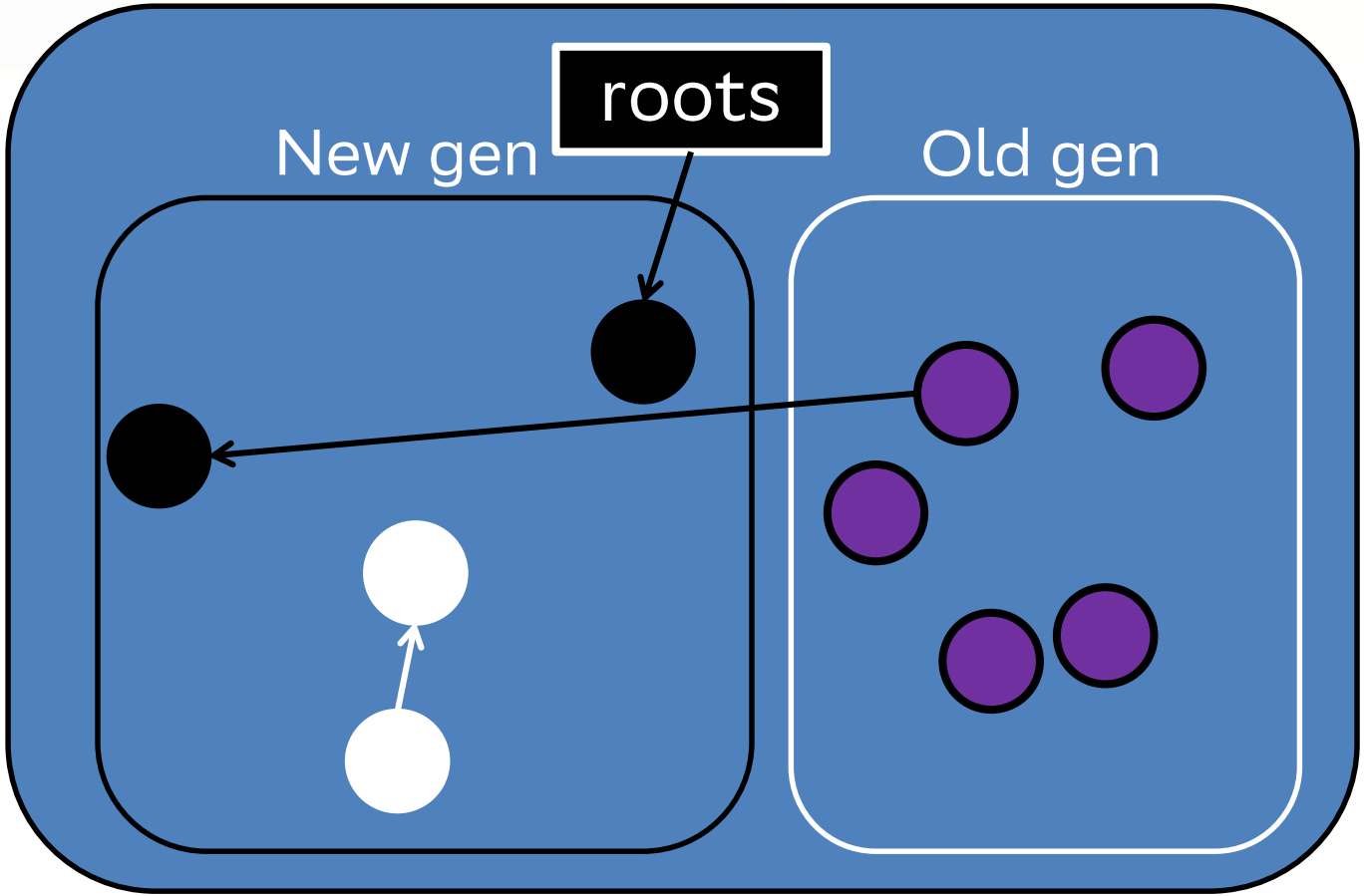
Cross generation references



Cross generation references



Cross generation references



Cross generation references barrier

```
void store_oop(oop val, oop* addr) {
    *addr = val;
    if (Oop::is_old(addr) && Oop::is_new(val)) {
      ... add to remembered set ...
    }
}
```

post-write barrier

Откуда берутся барьеры

Откуда берутся барьеры

```
 1 TemplateTable::aastore()                // templateTable_x86.cpp
 2 do_oop_store()                          // templateTable_x86.cpp
 3 MacroAssembler::store_heap_oop()        // macroAssembler_x86.cpp
 4 MacroAssembler::access_store_at() {     // macroAssembler_x86.cpp
 5   ...
 6   BarrierSet::barrier_set()->barrier_set_assembler()->store_at(...)
 7   ...
 8 }
 9
10 ZBarrierSetAssembler::store_at()        // zBarrierSetAssembler_x86.cpp
```

Откуда берутся барьеры

```
 1 TemplateTable::aastore()                // templateTable_x86.cpp
 2 do_oop_store()                          // templateTable_x86.cpp
 3 MacroAssembler::store_heap_oop()        // macroAssembler_x86.cpp
 4 MacroAssembler::access_store_at() {     // macroAssembler_x86.cpp
 5   ...
 6   BarrierSet::barrier_set()->barrier_set_assembler()->store_at(...)
 7   ...
 8 }
 9
10 ZBarrierSetAssembler::store_at()        // zBarrierSetAssembler_x86.cpp
```

Откуда берутся барьеры

```
 1 TemplateTable::aastore()                // templateTable_x86.cpp
 2 do_oop_store()                          // templateTable_x86.cpp
 3 MacroAssembler::store_heap_oop()        // macroAssembler_x86.cpp
 4 MacroAssembler::access_store_at() {     // macroAssembler_x86.cpp
 5   ...
 6   BarrierSet::barrier_set()->barrier_set_assembler()->store_at(...)
 7   ...
 8 }
 9
10 ZBarrierSetAssembler::store_at()        // zBarrierSetAssembler_x86.cpp
```

Откуда берутся барьеры

```
1  TemplateTable::aastore()                 // templateTable_x86.cpp
2  do_oop_store()                           // templateTable_x86.cpp
3  MacroAssembler::store_heap_oop()         // macroAssembler_x86.cpp
4  MacroAssembler::access_store_at() {      // macroAssembler_x86.cpp
5    ...
6    BarrierSet::barrier_set()->barrier_set_assembler()->store_at(...)
7    ...
8  }
9
10 ZBarrierSetAssembler::store_at()         // zBarrierSetAssembler_x86.cpp
```

Откуда берутся барьеры

```
 1 TemplateTable::aastore()              // templateTable_x86.cpp
 2 do_oop_store()                        // templateTable_x86.cpp
 3 MacroAssembler::store_heap_oop()      // macroAssembler_x86.cpp
 4 MacroAssembler::access_store_at() {   // macroAssembler_x86.cpp
 5   ...
 6   BarrierSet::barrier_set()->barrier_set_assembler()->store_at(...)
 7   ...
 8 }
 9
10 ZBarrierSetAssembler::store_at()      // zBarrierSetAssembler_x86.cpp
```

BarrierSetAssembler

```
1  void ZBarrierSetAssembler::store_at(...)
2    // Do ZGC work
3    {
4      store_barrier_fast(...)
5      ...
6      store_barrier_medium(...)
7      ...
8      __ MacroAssembler::call_VM_leaf(..., ZBarrierSetRuntime::
9           store_barrier_on_oop_field_without_healing_addr());
10     ...
11   }
12   __ bind(done);
13   // Store value
14   BarrierSetAssembler::store_at(masm, ...);
15 }
```

ZGC

generic

BarrierSetAssembler

```
1  void ZBarrierSetAssembler::store_at(...)
2    // Do ZGC work
3    {
4      store_barrier_fast(...)
5      ...
6      store_barrier_medium(...)
7      ...
8      __ MacroAssembler::call_VM_leaf(..., ZBarrierSetRuntime::
9           store_barrier_on_oop_field_without_healing_addr());
10     ...
11   }
12   __ bind(done);
13   // Store value
14   BarrierSetAssembler::store_at(masm, ...);
15 }
```

ZGC

BarrierSetAssembler

```
1  void ZBarrierSetAssembler::store_at(...)
2    // Do
3    {
4      stor
5      ...
6      stor
7      ...
8      __ M
9
10     ...
11   }
12   __ bin
13   // Sto
14   Barrie
15 }
```

```
1  void ZBarrierSetAssembler::store_barrier_medium(
2      MacroAssembler* masm,
3      Label& slow_path,
4      Label& slow_path_continuation) const {
5    ...
6    __ lea(rcx, ref_addr);
7    __ xorq(rax, rax);
8    __ movptr(rbx, Address(r15, ZThreadLocalData::store_good_mask_offset()));
9    __ lock();
10   __ cmpxchgq(rbx, Address(rcx, 0));
11   __ pop(rcx);
12   __ pop(rbx);
13   __ pop(rax);
14   __ jcc(Assembler::notEqual, slow_path);
15   ...
16 }
```

BarrierSetAssembler

```
1  void ZBarrierSetAssembler::store_at(...)
2    // Do ZGC work
3    {
4      store_barrier_fast(...)
5      ...
6      store_barrier_medium(...)
7      ...
8      __ MacroAssembler::call_VM_leaf(..., ZBarrierSetRuntime::
9           store_barrier_on_oop_field_without_healing_addr());
10     ...
11   }
12   __ bind(done);
13   // Store value
14   BarrierSetAssembler::store_at(masm, ...);
15 }
```

ZGC

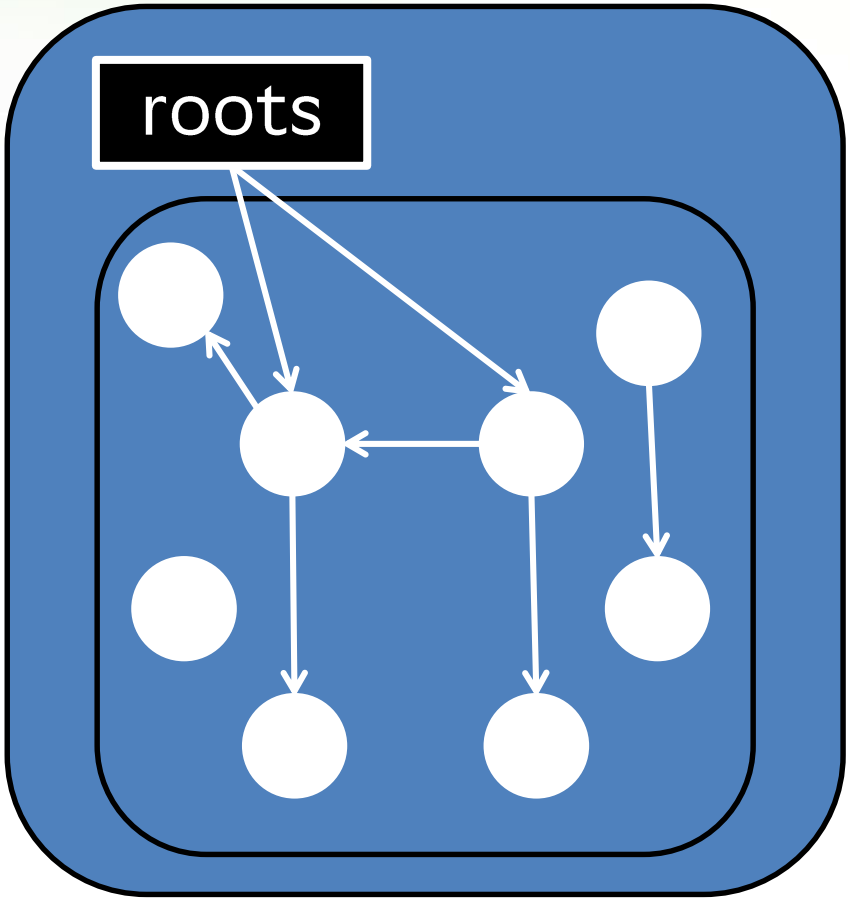
BarrierSetAssembler

```
1  void ZBarri
2    // Do ZGC
3    {
4      store_b
5      ...
6      store_b
7      ...
8      __ Macr
9           st
10     ...
11   }
12   __ bind(d
13   // Store
14   BarrierSe
15 }
```

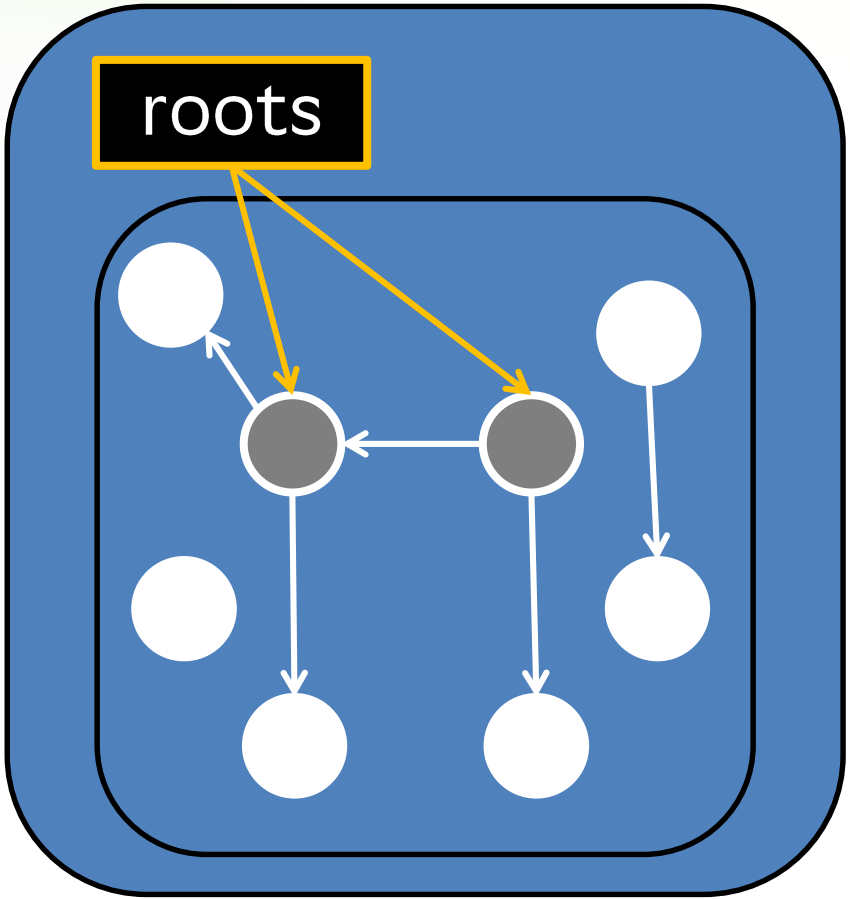
```
ZBarrierSetRuntime::store_barrier_on_oop_field_without_healing_addr()
ZBarrierSetRuntime::store_barrier_on_oop_field_without_healing()
ZBarrier::store_barrier_on_heap_oop_field()
ZBarrier::heap_store_slow_path // zBarrier.cpp

zaddress ZBarrier::heap_store_slow_path(...) {
  ZStoreBarrierBuffer* buffer =
ZStoreBarrierBuffer::buffer_for_store(heal);
  if (buffer != nullptr) {
    // Buffer store barriers whenever possible
    buffer->add(p, prev);
  } else {
    mark_and_remember(p, addr);
  }
  return addr;
}
```

Marking

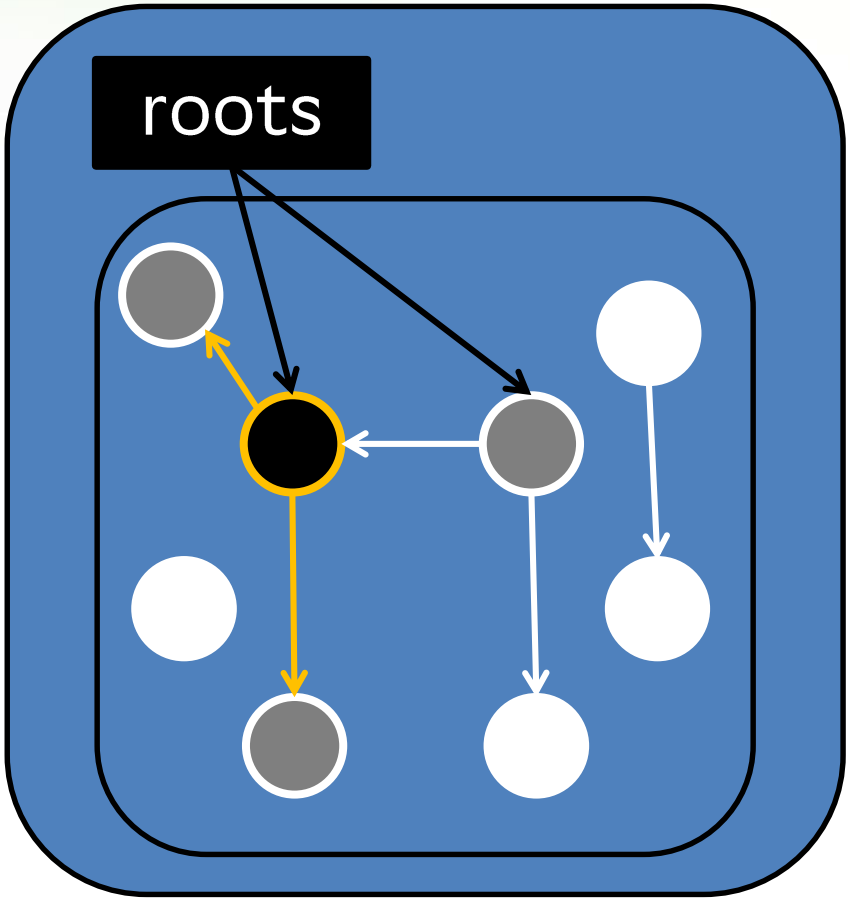
Marking



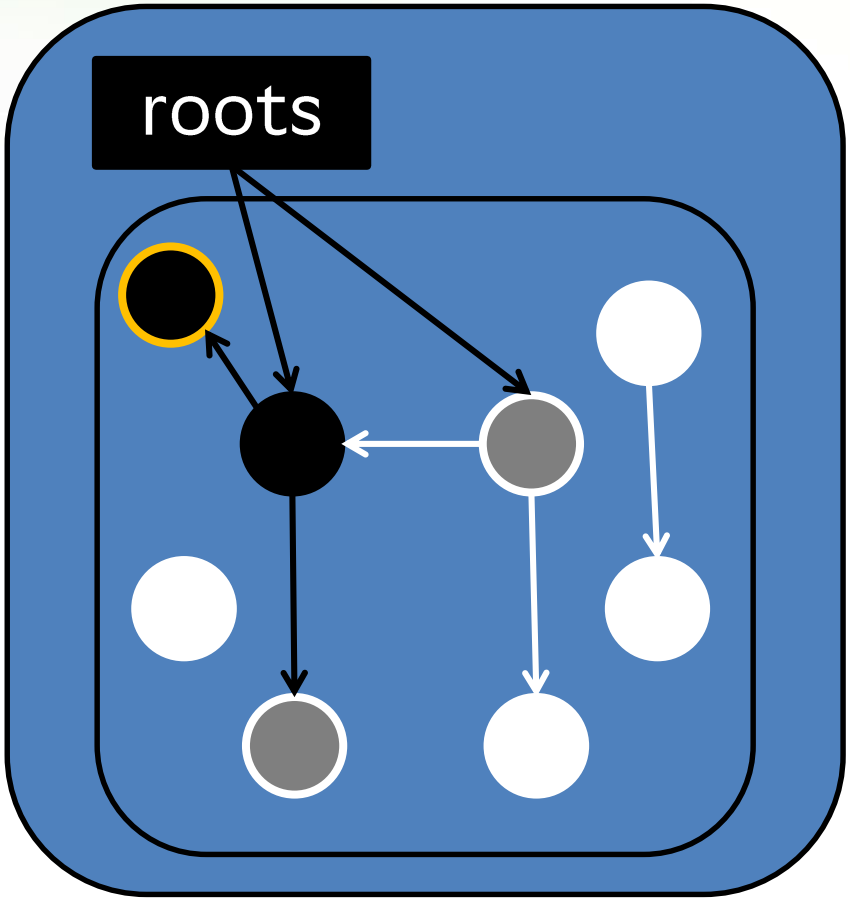
Marking



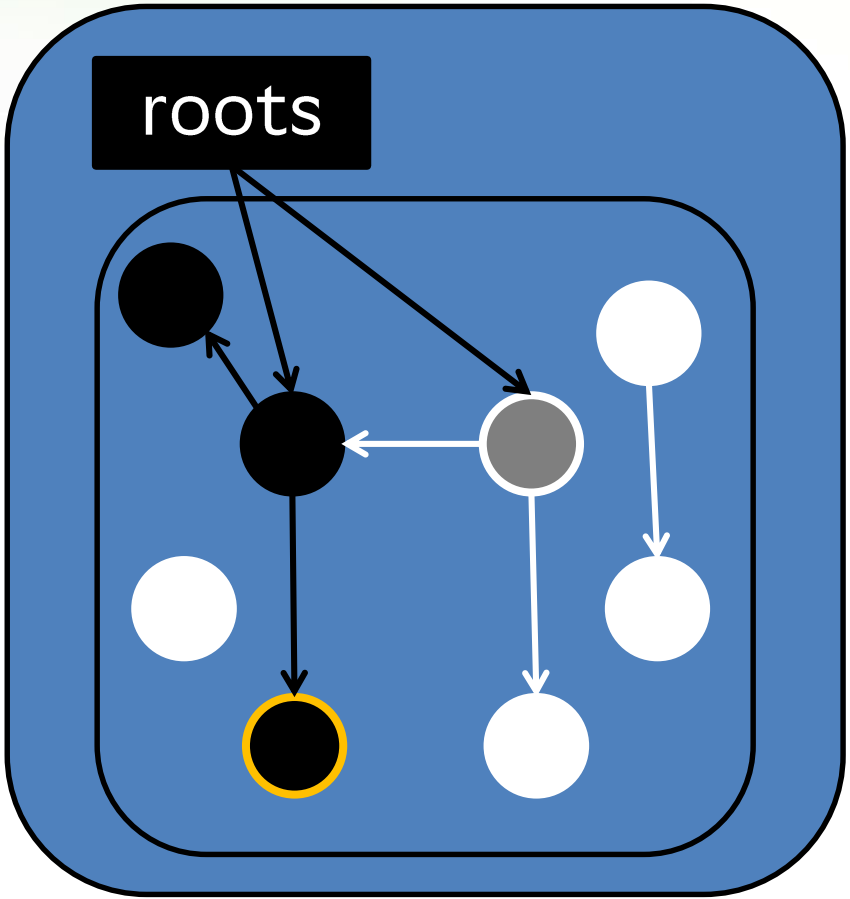
Marking



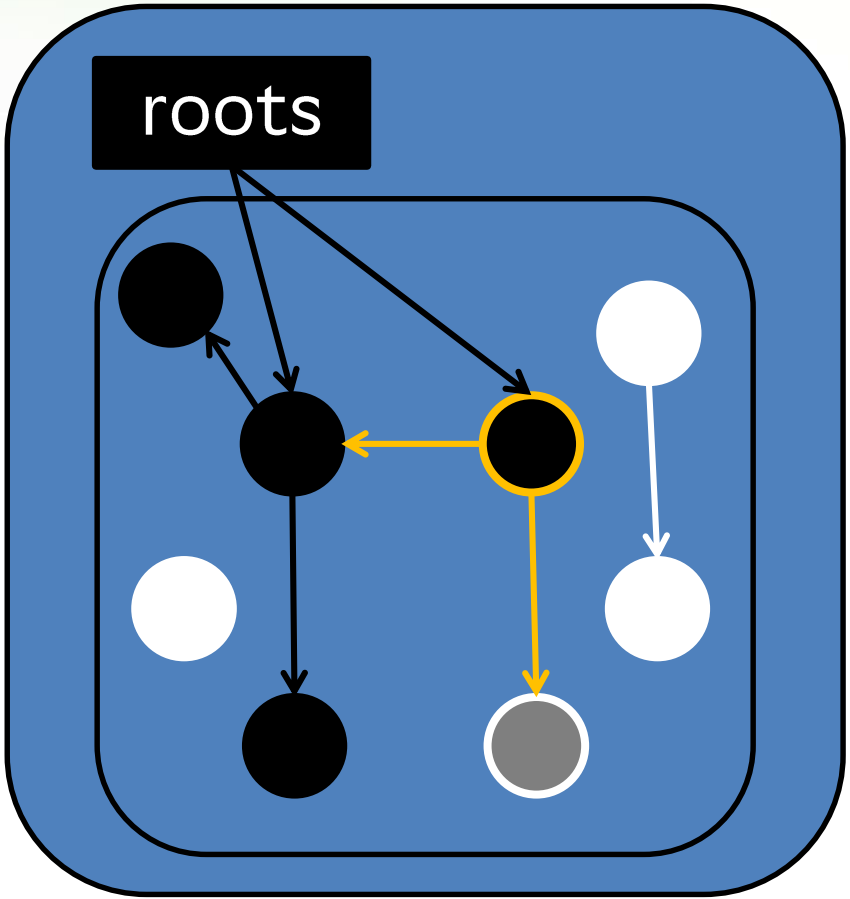
Marking



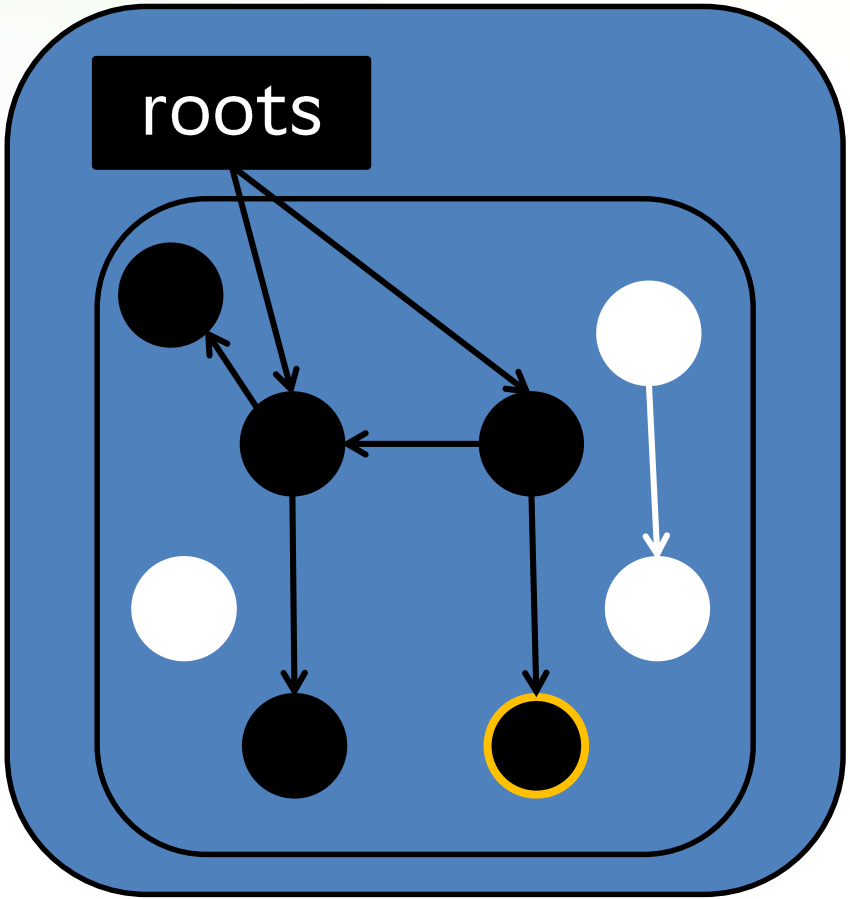
Marking



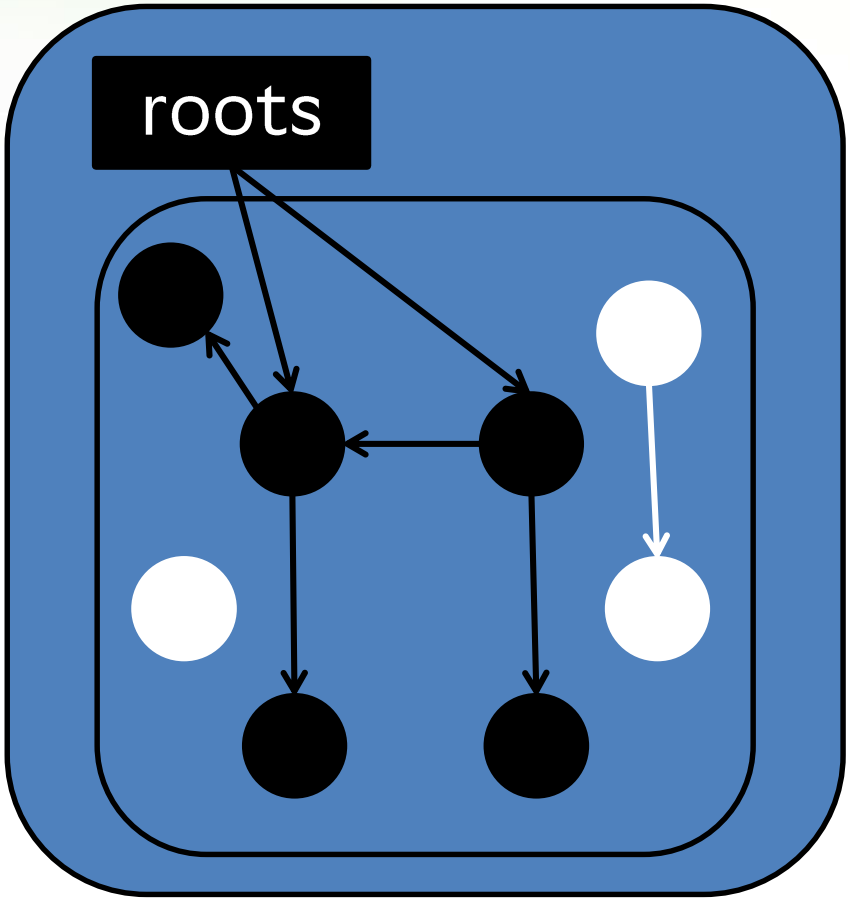
Marking



Marking

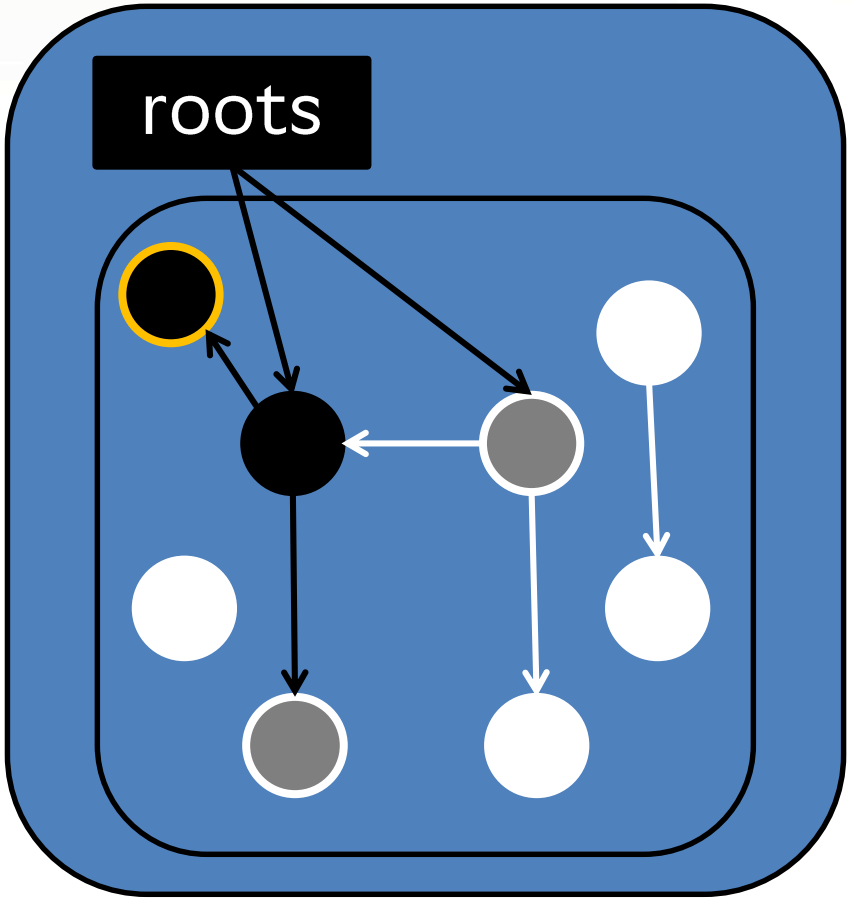


Marking

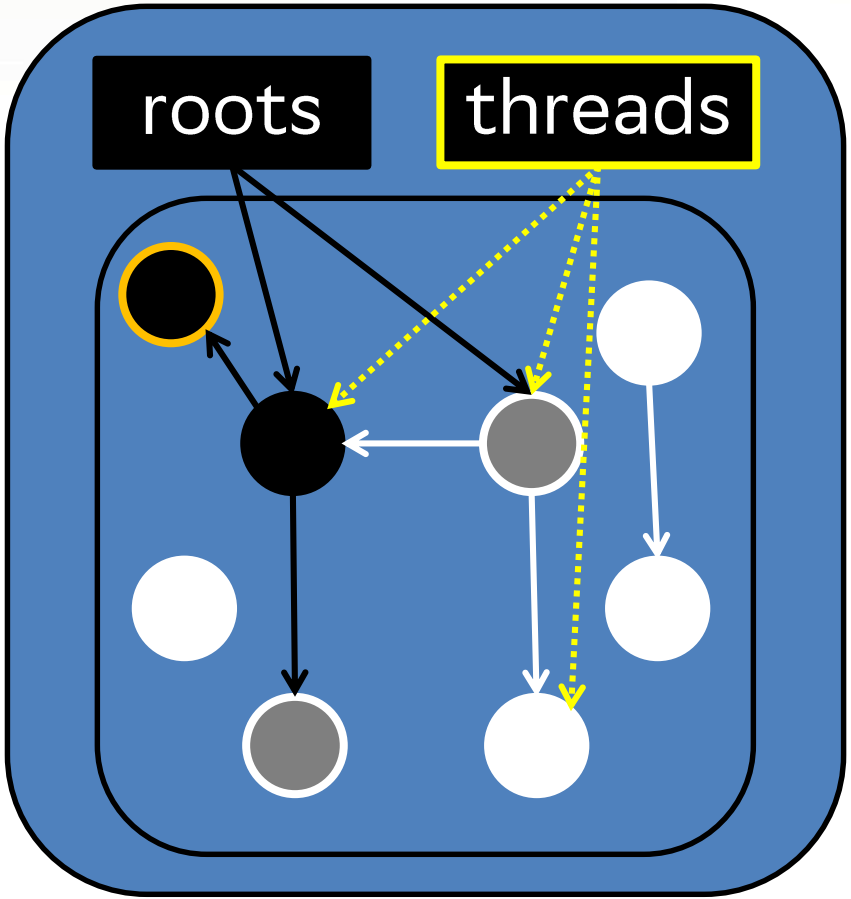


Concurrent Marking

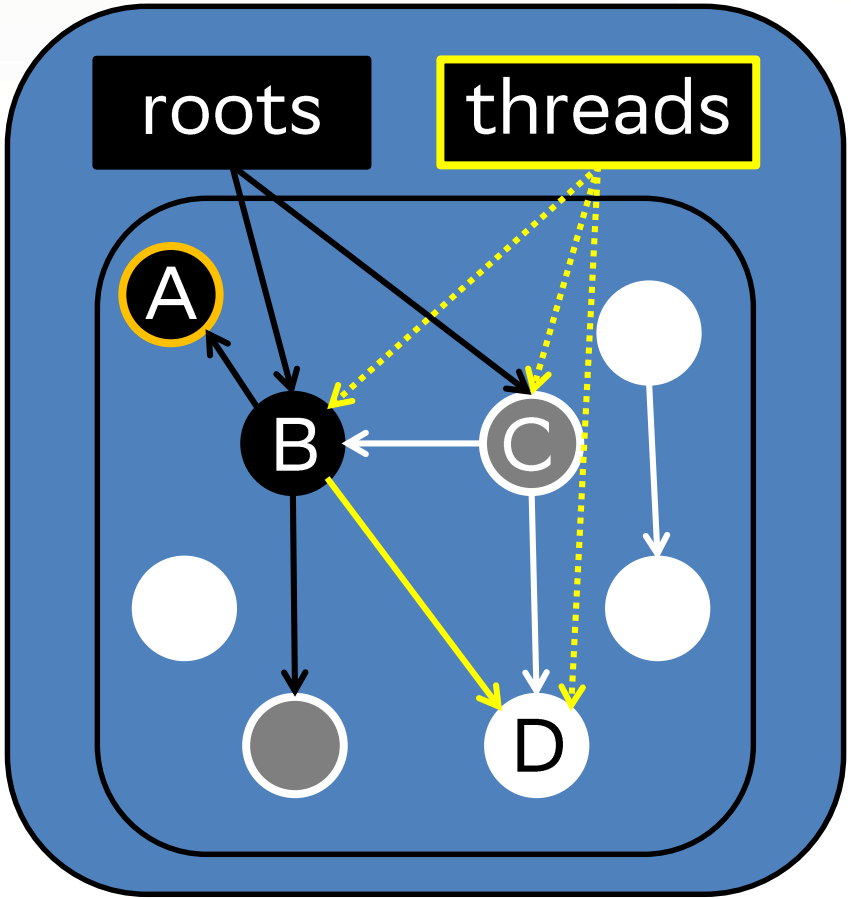
Concurrent Marking



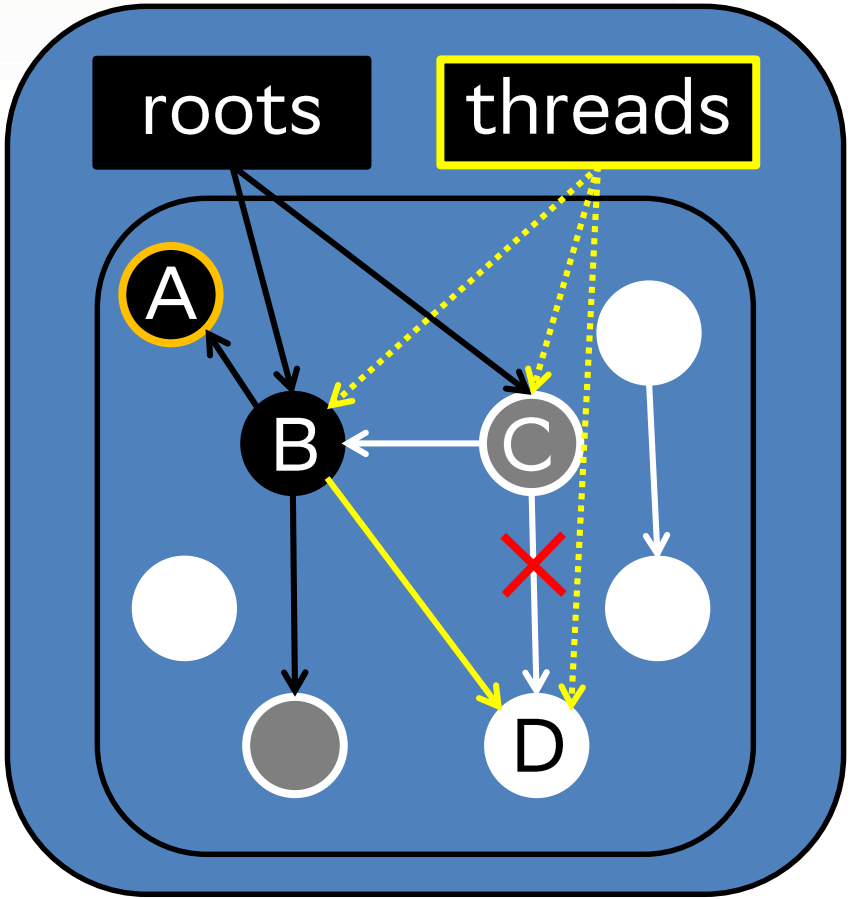
Concurrent Marking



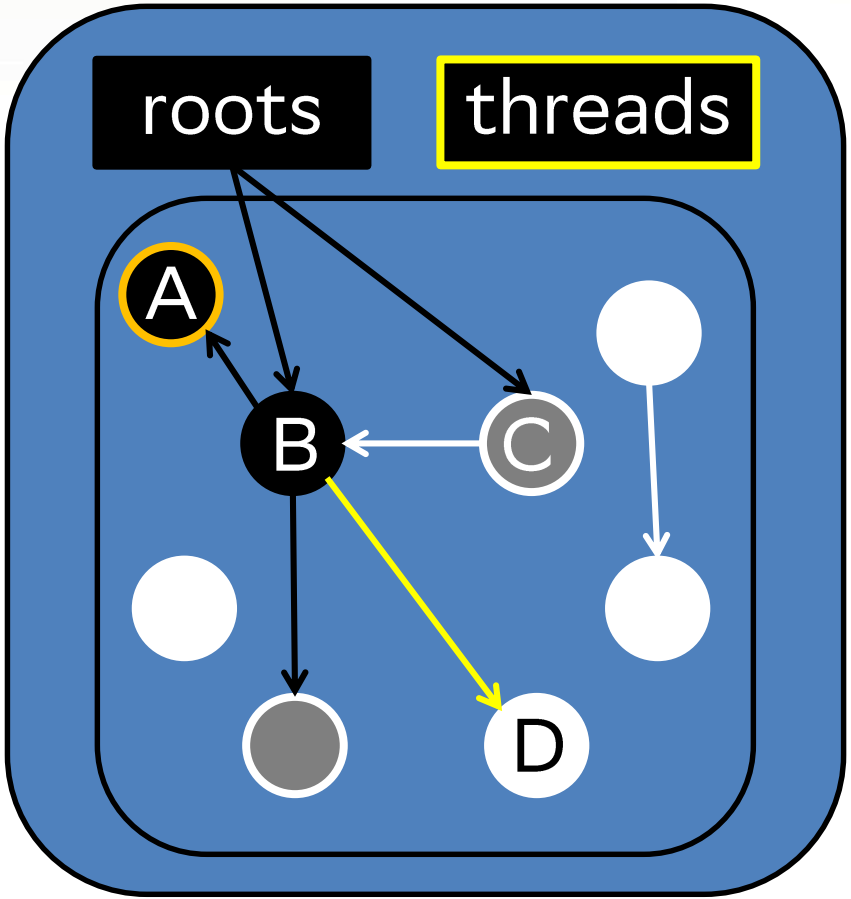
Concurrent Marking



Concurrent Marking

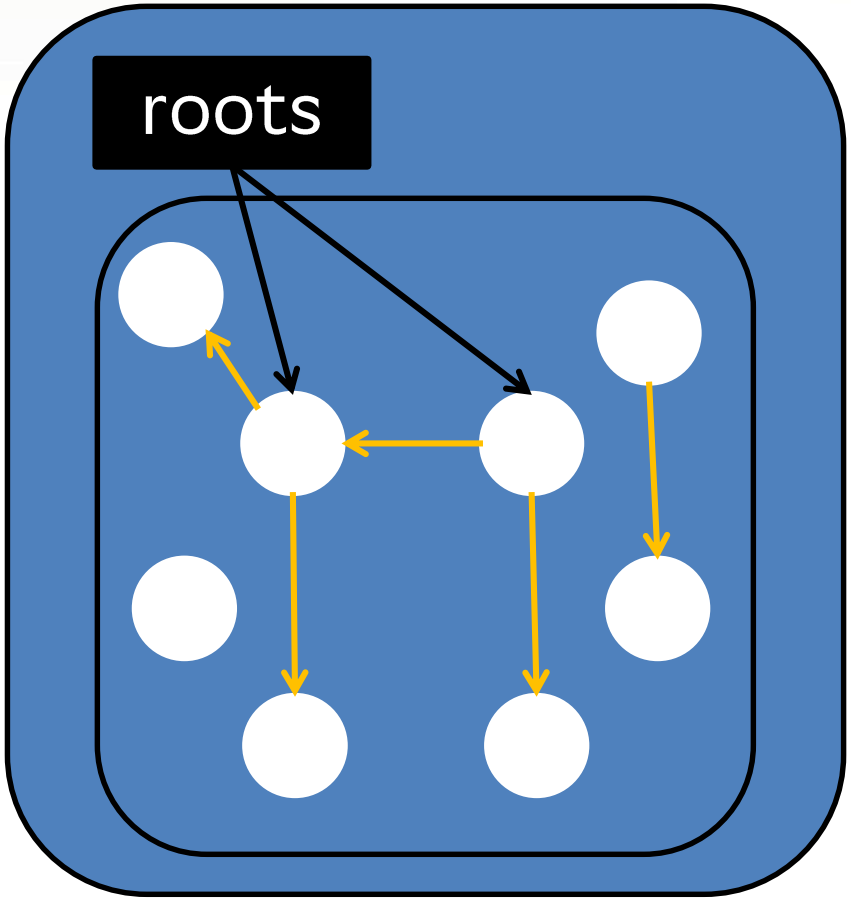


Concurrent Marking

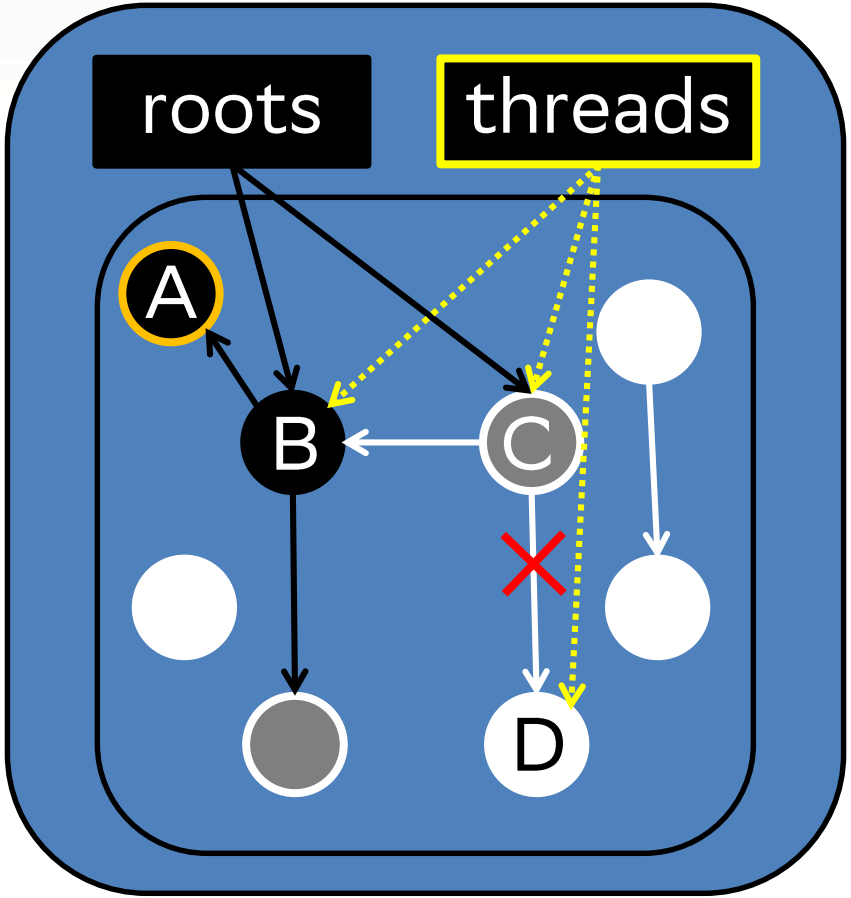


SATB: Snapshot at the beginning

SATB: Snapshot at the beginning



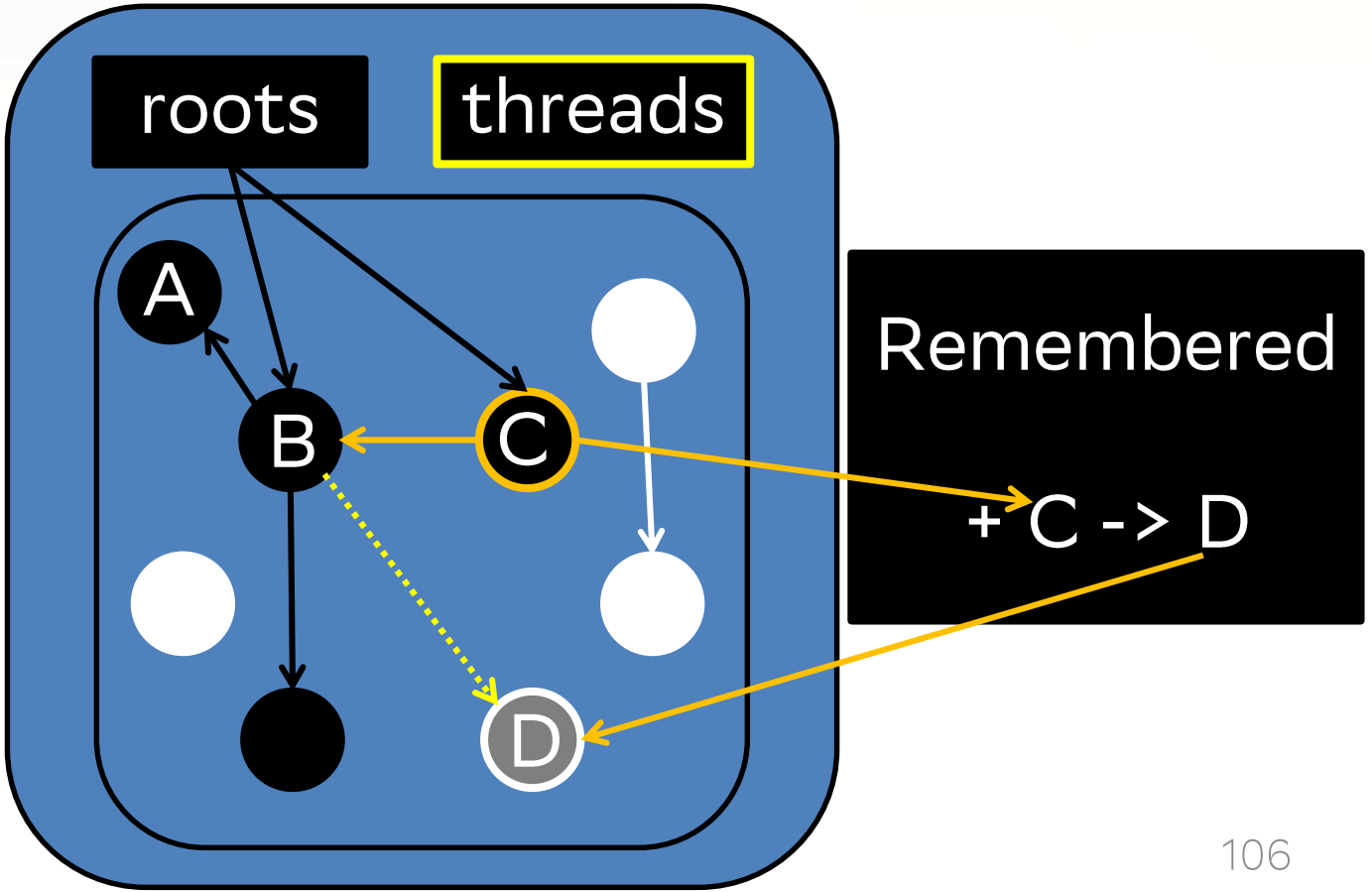
SATB: Snapshot at the beginning



Remembered

+ C -> D

SATB: Snapshot at the beginning

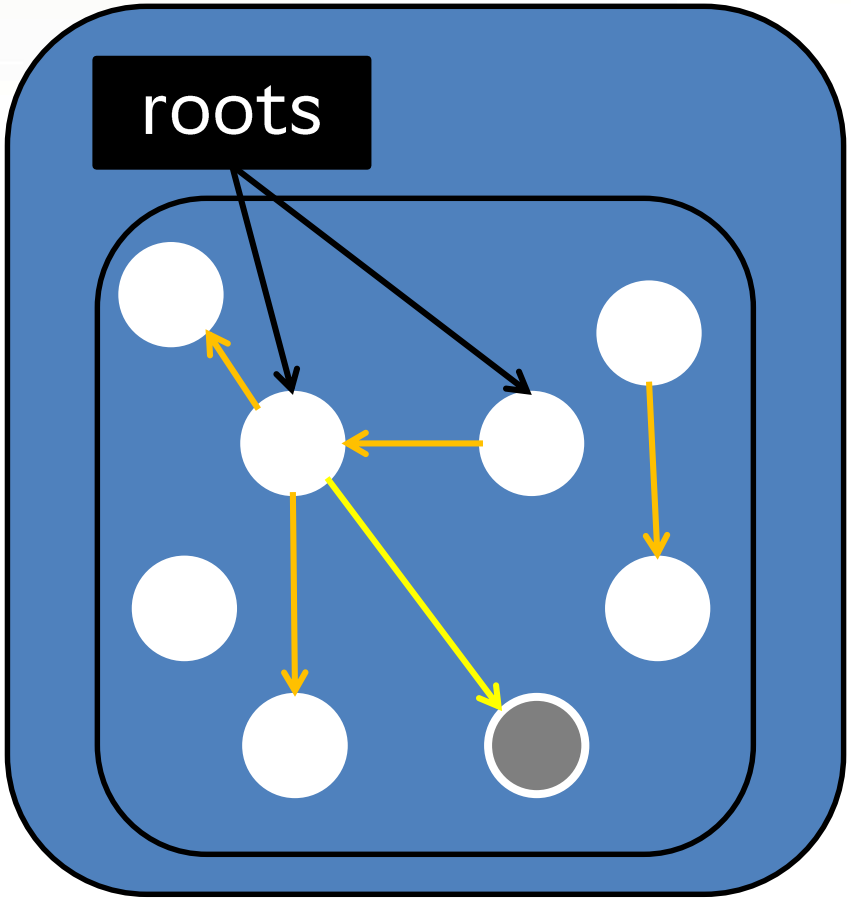


SATB: Snapshot at the beginning

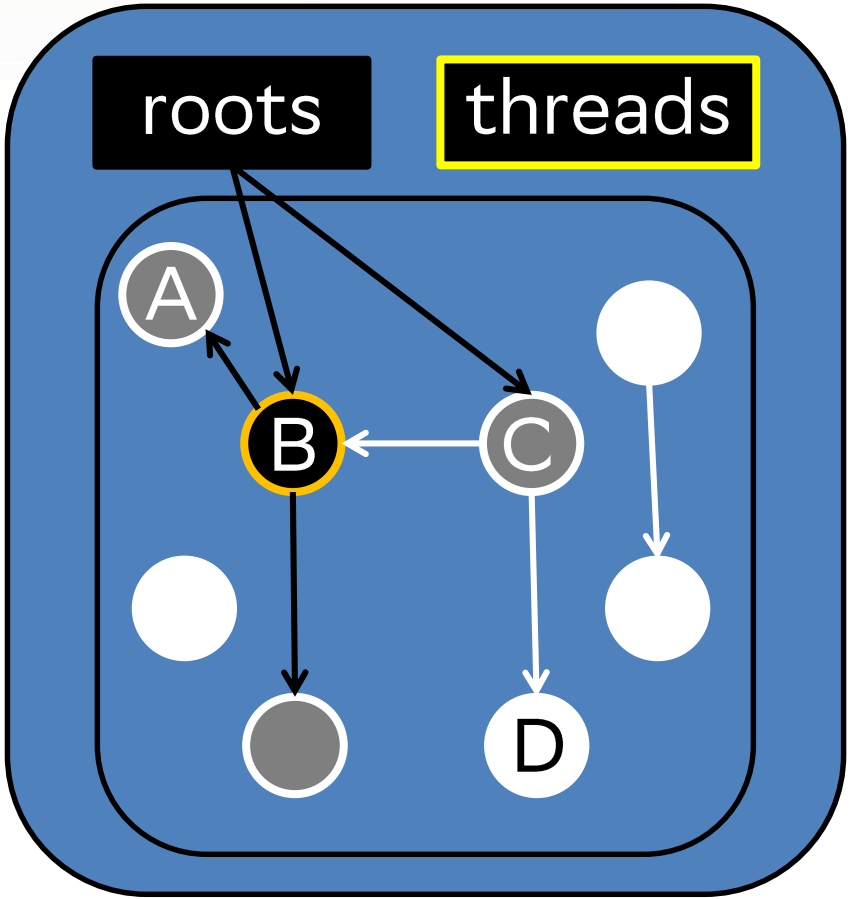
```
void store_oop(oop x, oop* addr) {
    GC::remember(/*from*/addr, /*to*/*addr);
    *addr = x;
}
```

Load Value Barrier (LVB)

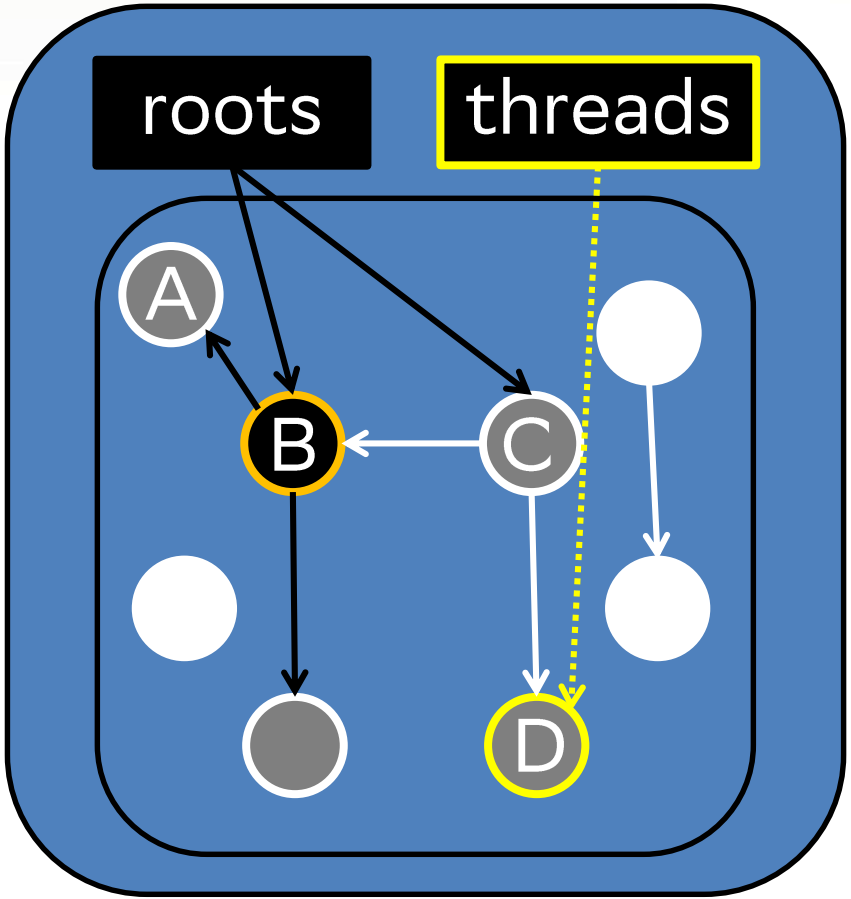
Load Value Barrier (LVB)



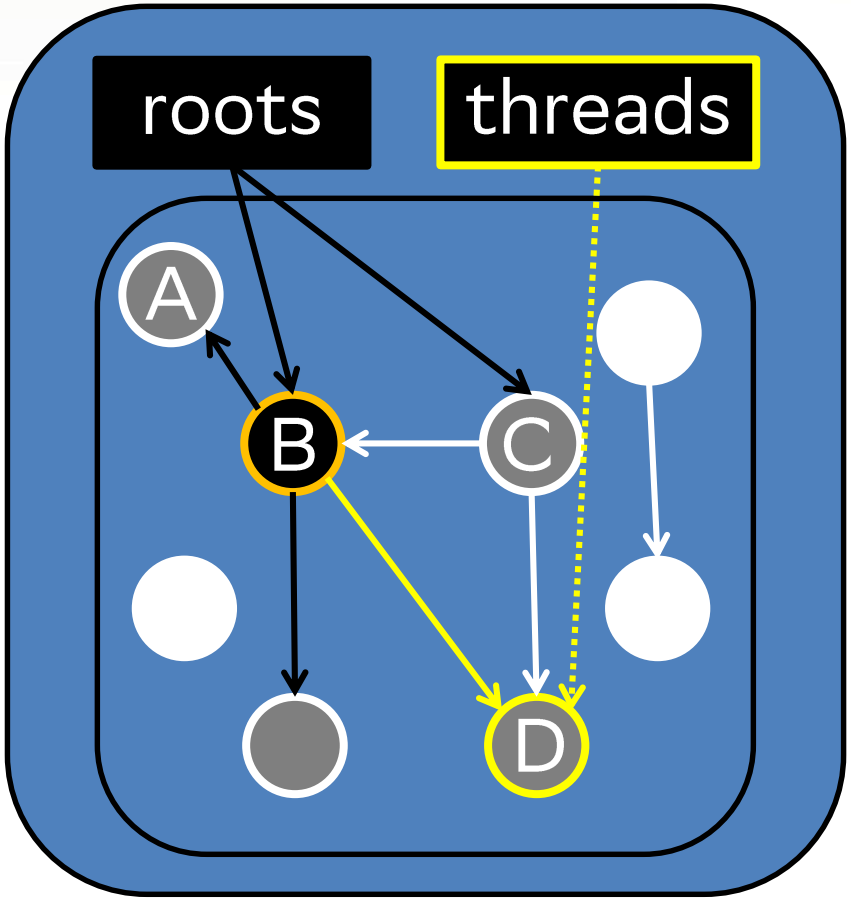
Load Value Barrier (LVB)



Load Value Barrier (LVB)



Load Value Barrier (LVB)

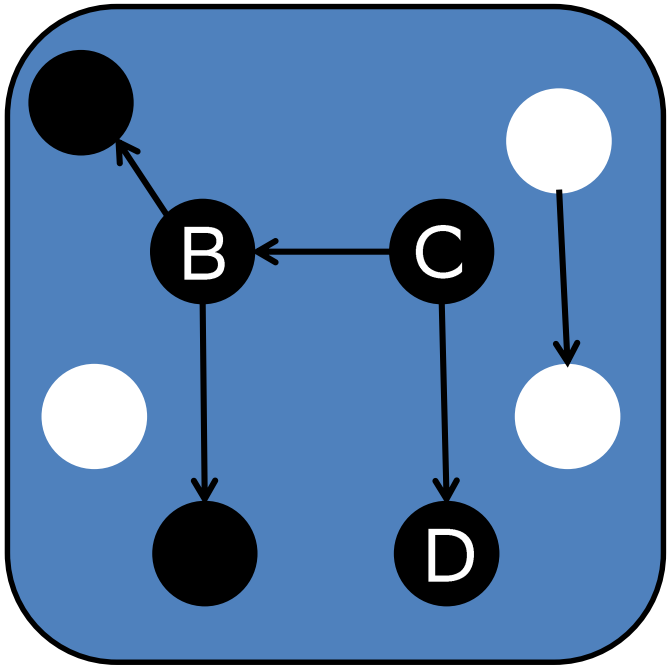


Load Value Barrier (LVB)

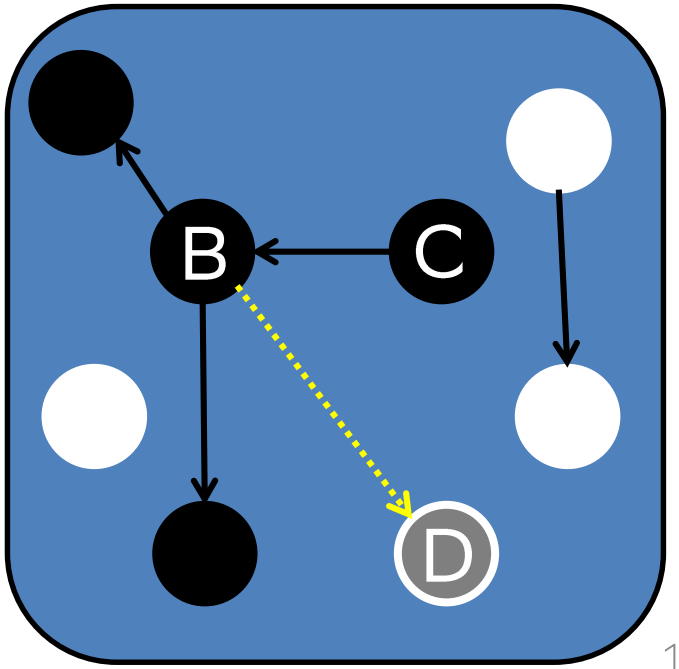
```
oop load_oop(oop* addr) {
    oop x = *addr;
    GC::mark(x);
    return x;
}
```

Load Value Barrier (LVB)

SATB: loads > stores



LVB: some might have died



Стоимость барьеров

- fast path, 1-10%

- slow path

Стоимость барьеров

- fast path, 1-10%
  - colored oops
  - nop

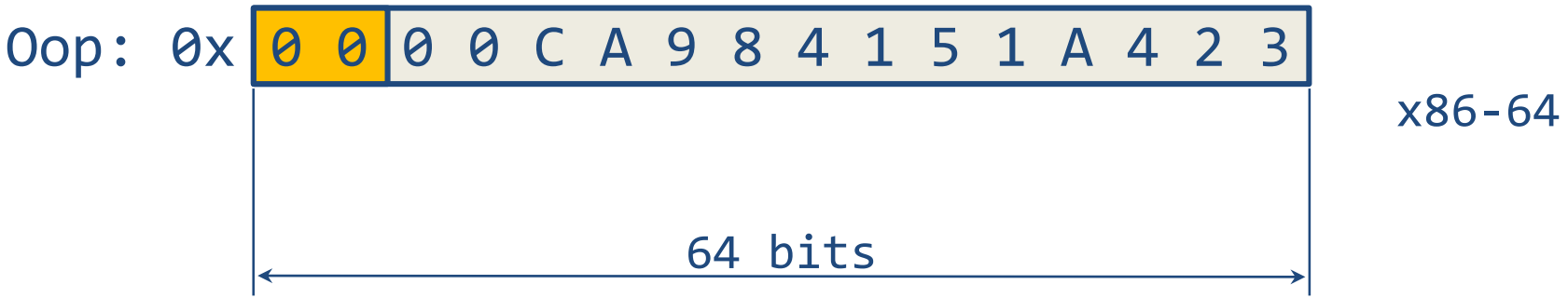
- slow path
  - self heal

Стоимость барьеров

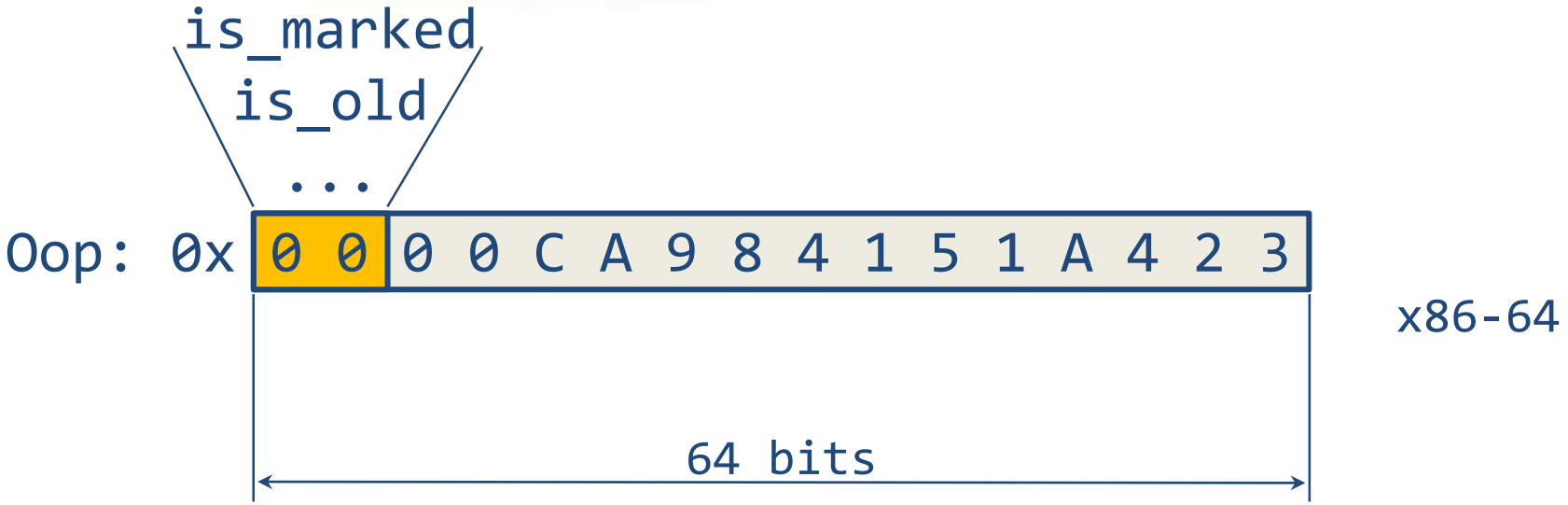
- fast path, 1-10%
  - **colored oops**
  - nop

- slow path
  - self heal

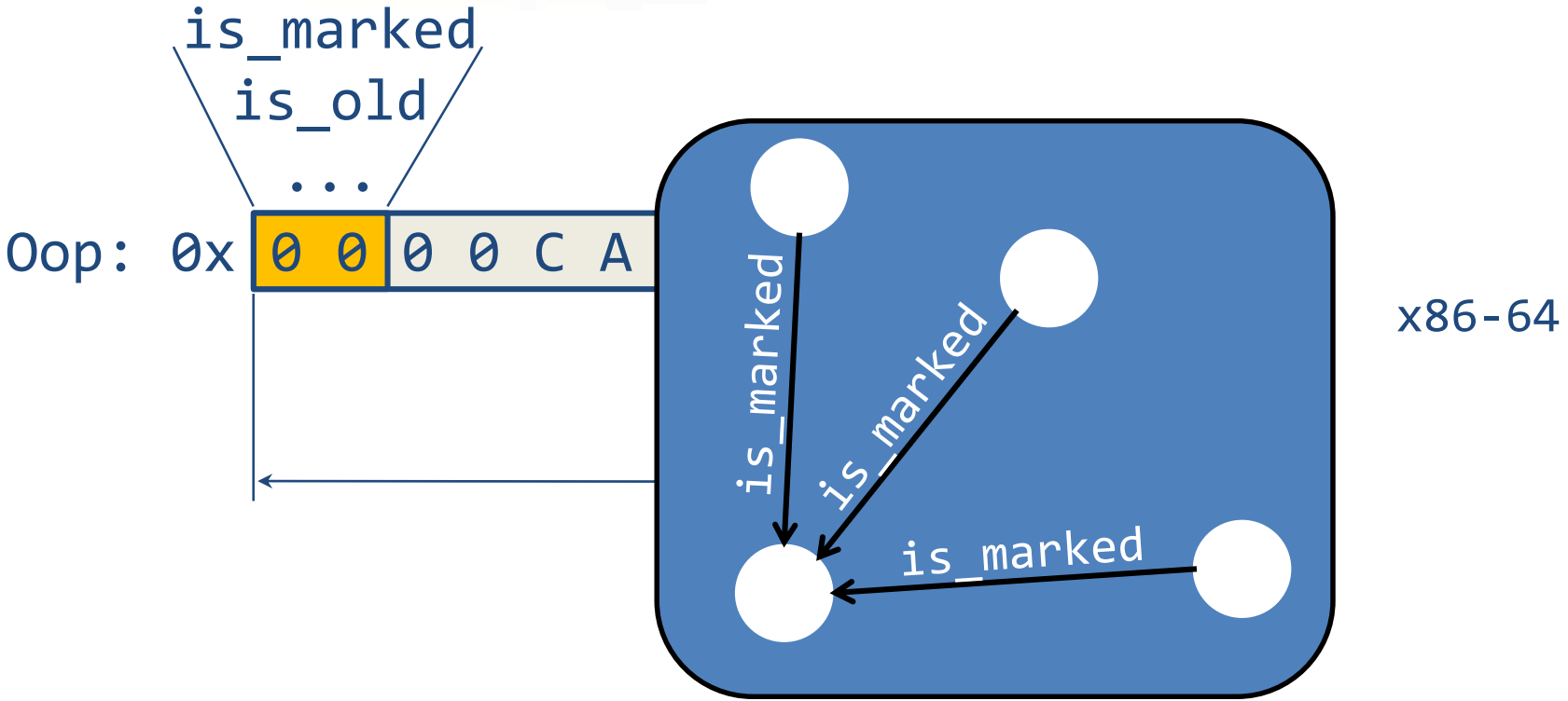
Colored oops



Colored oops

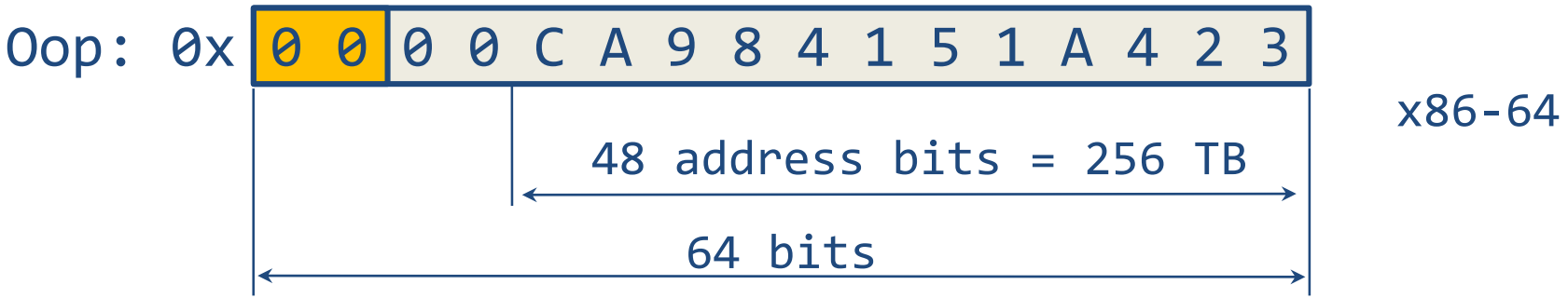


Colored oops

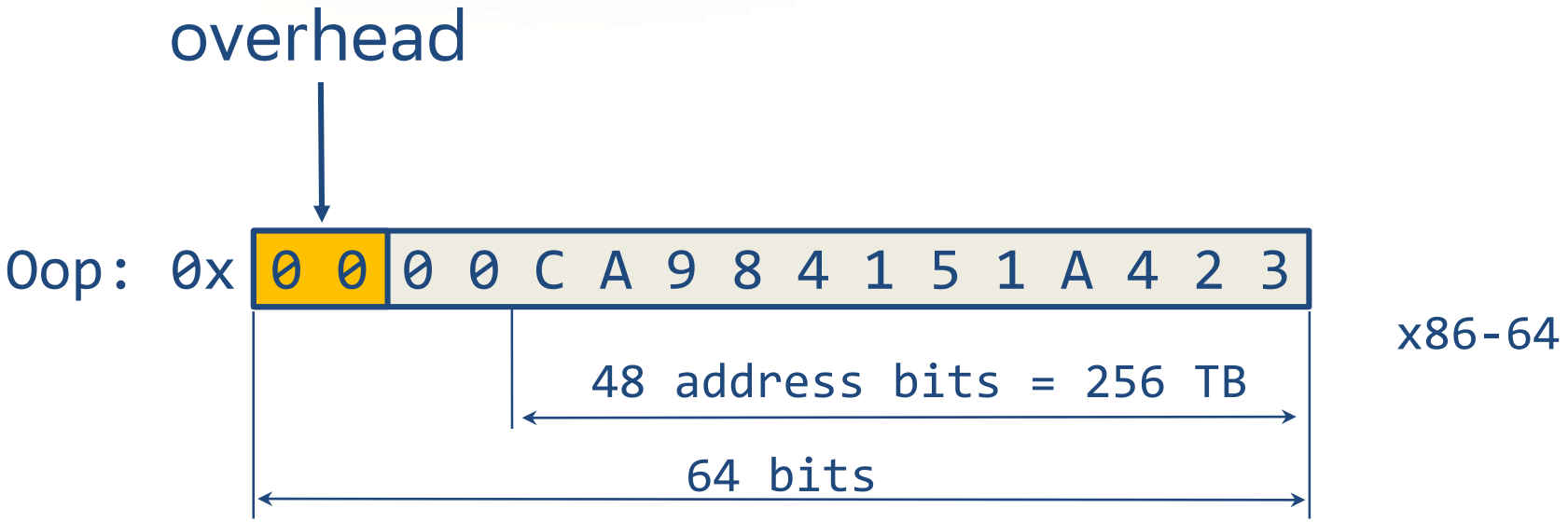


Colored oops

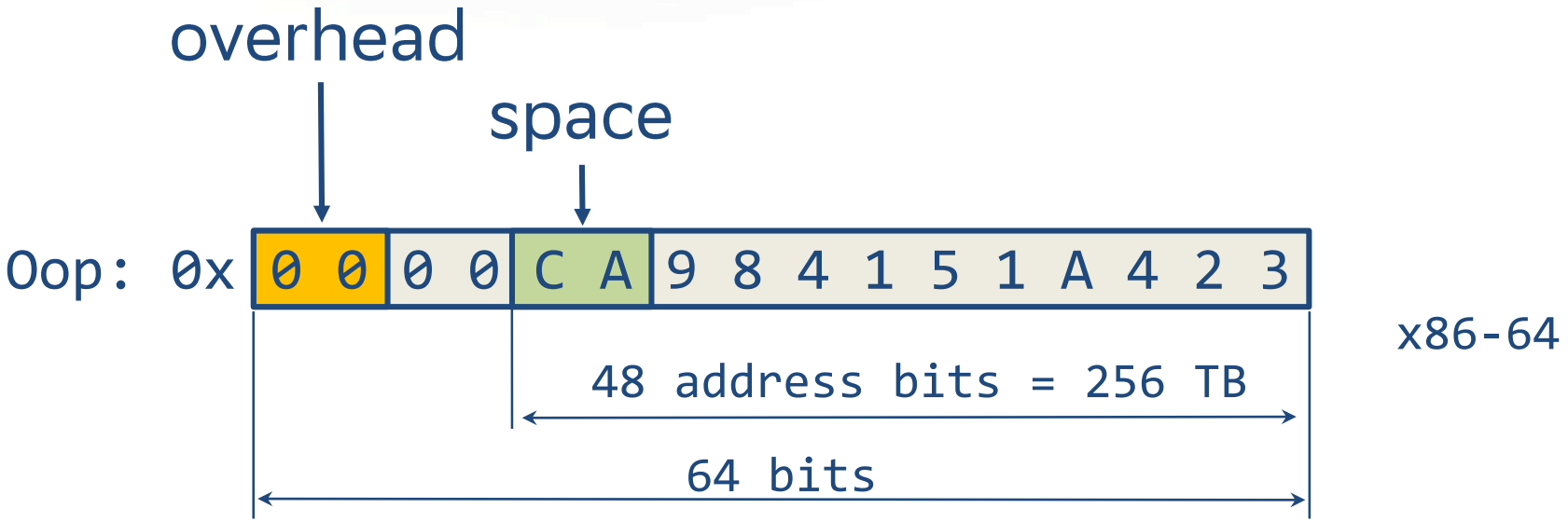
4-Level Paging – 48 bits
5-Level Paging – 57 bits



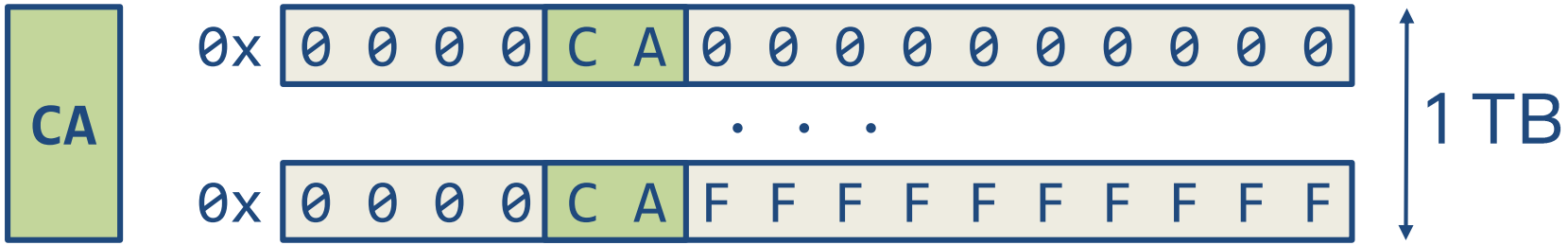
Colored oops



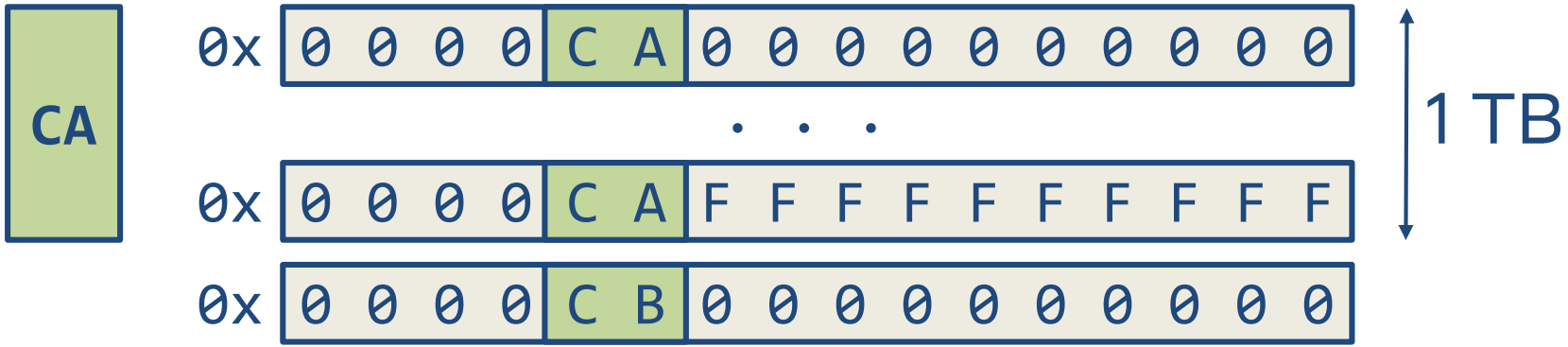
Colored oops



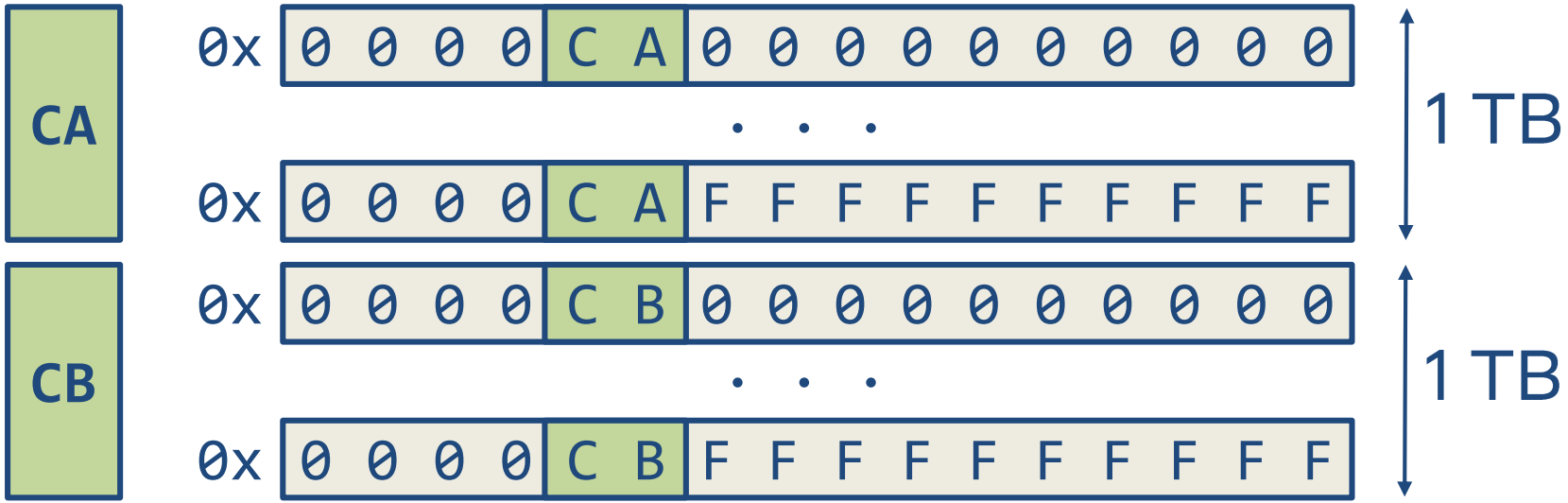
Colored oops - multimapping



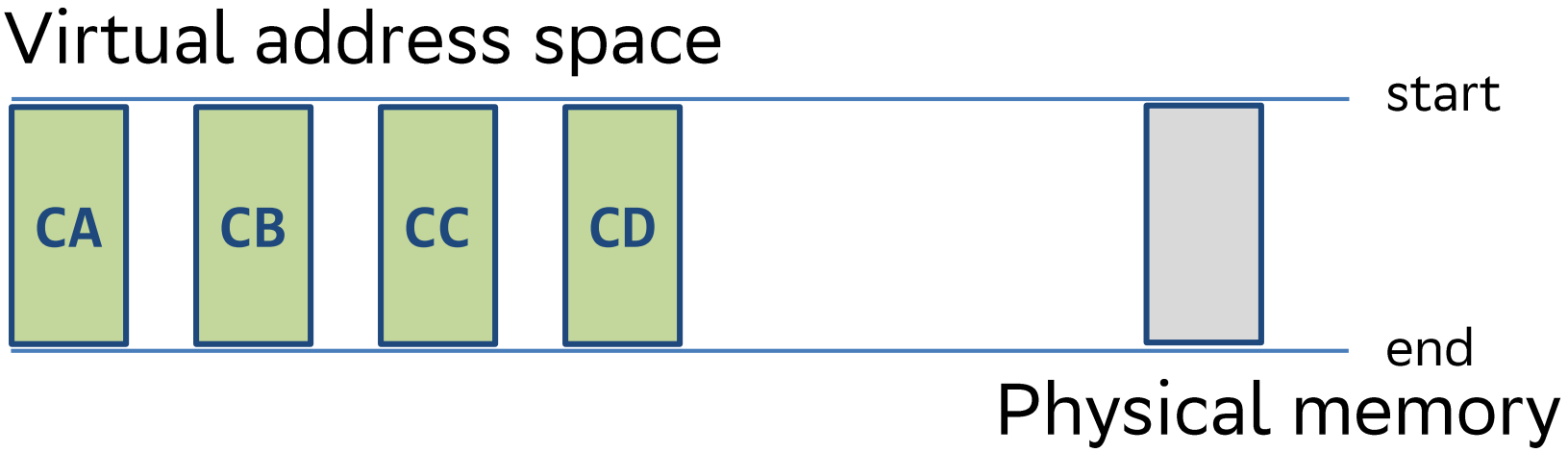
Colored oops - multimapping



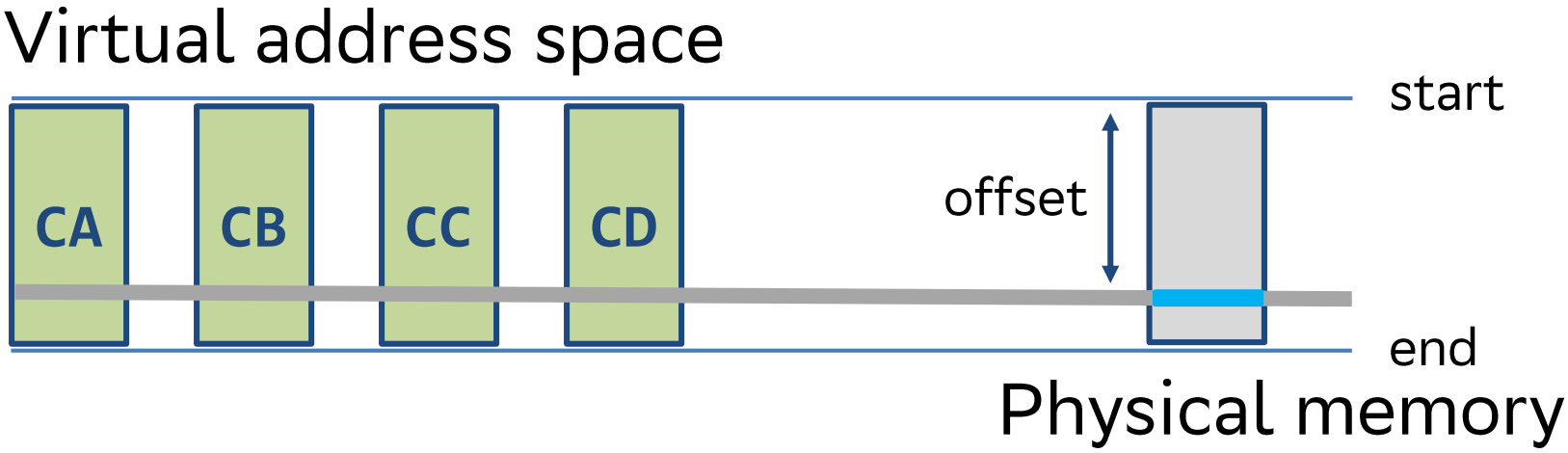
Colored oops - multimapping



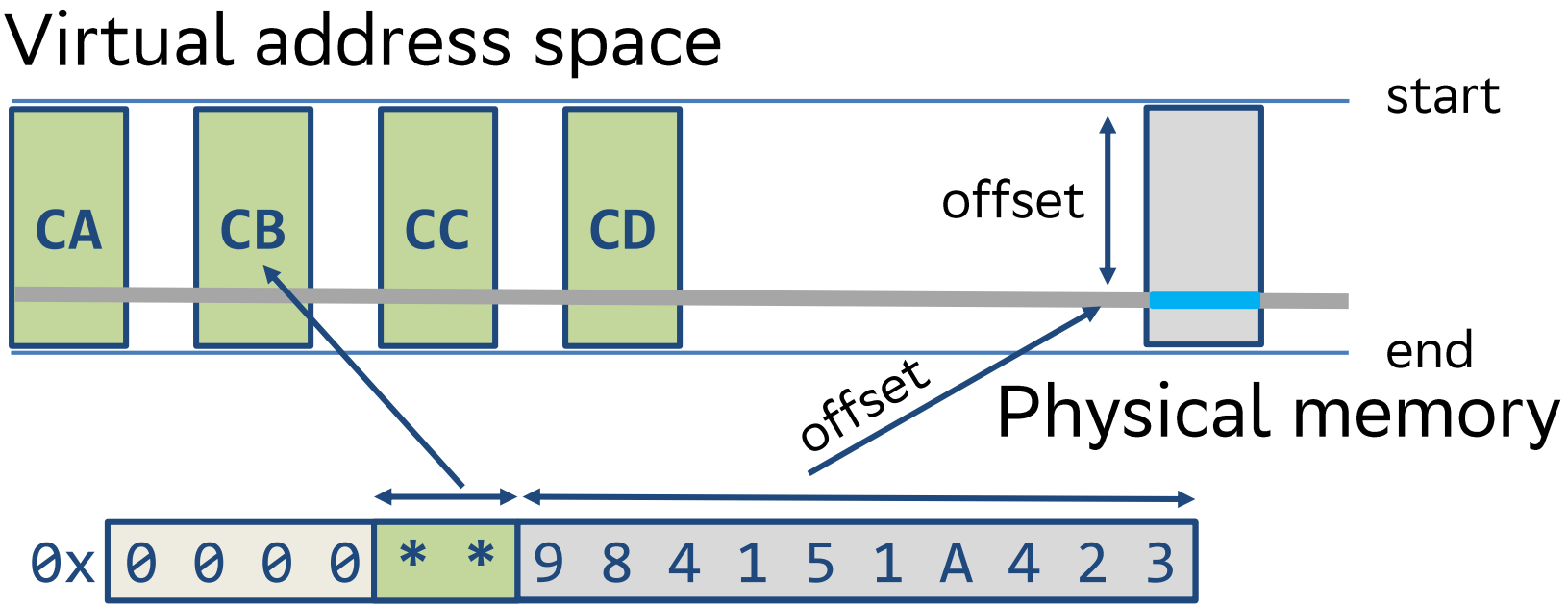
Colored oops - multimapping



Colored oops - multimapping



Colored oops - multimapping



Colored oops – cross gen references

```
void store_oop(oop val, oop* addr) {
    *addr = val;
    if (Oop::is_old(addr) && Oop::is_new(val)) {
      ... add to remembered set ...
    }
}
```

Colored oops – cross gen references

```
void store_oop(oop val, oop* addr) {
    *addr = val;
    if ((addr & Oop::is_old_bit) && (val & Oop::is_new_bit)) {
      ... add to remembered set ...
    }
}
```

Стоимость барьеров

- fast path, 1-10%
  - colored oops ✓
  - nop

- slow path
  - **self heal**

Self heal (LVB)

```
oop load_oop(oop* addr) {
    oop x = *addr;
    GC::mark(x);
    return x;
}
```

Self heal (LVB)

```
oop load_oop(oop* addr) {
    oop x = *addr;
    // check color
    if (!(x & GC::marked_bit)) {
        GC::mark(x);
        // self heal
        cmpxchg(addr, x, x | GC::marked_bit);
    }
    return x;
}
```

Стоимость барьеров

- fast path, 1-10%
  - colored oops ✓
  - **nop**

- slow path
  - self heal ✓

NOP LVB

```
oop load_oop(oop* addr) {
    oop x = *addr;
    if (x & GC::trap_mask) {
        GC::lvb_slowapth(addr, x);
    }
    return x;
}
```

Is GC running?

NOP LVB

```
oop load_oop(oop* addr) {
    oop x = *addr;
    if (x & GC::trap_mask) {
        GC::lvb_slowapth(addr, x);
    }
    return x;
}
```

```
oop load_oop(oop* addr) {
    oop x = *addr;
    if (x & GC::trap_mask) {
        GC::lvb_slowapth(addr, x);
    }
    return x;
}
```

Стоимость барьеров

- fast path, 1-10%
  - colored oops ✓
  - nop ✓
- slow path
  - self heal ✓

STW vs Concurrent

STW vs Concurrent

latency

throughput

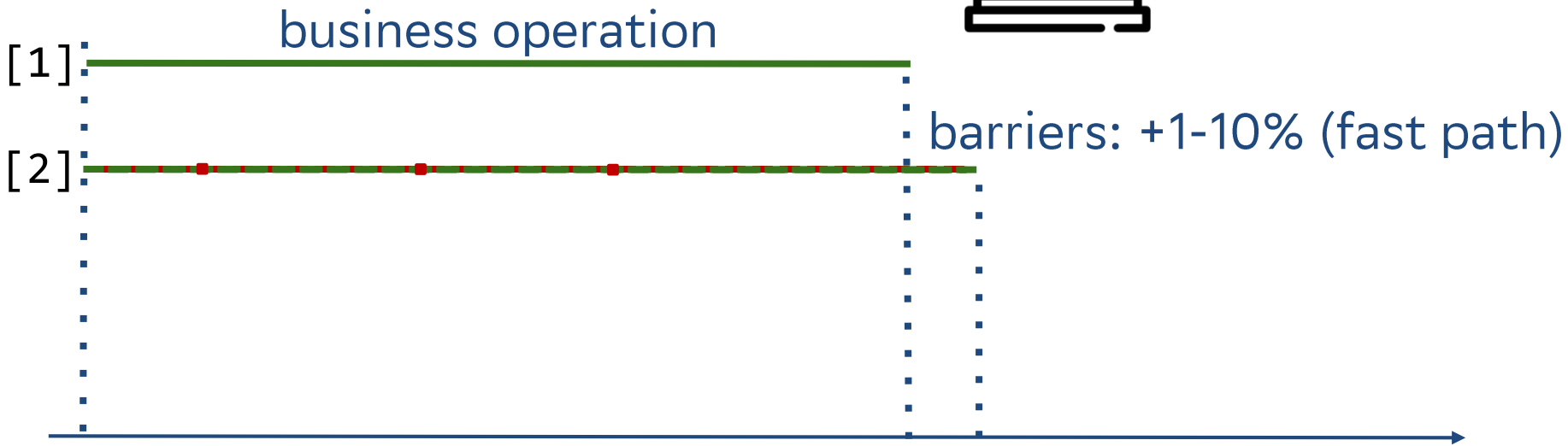
business operation

[1]

STW vs Concurrent

throughput

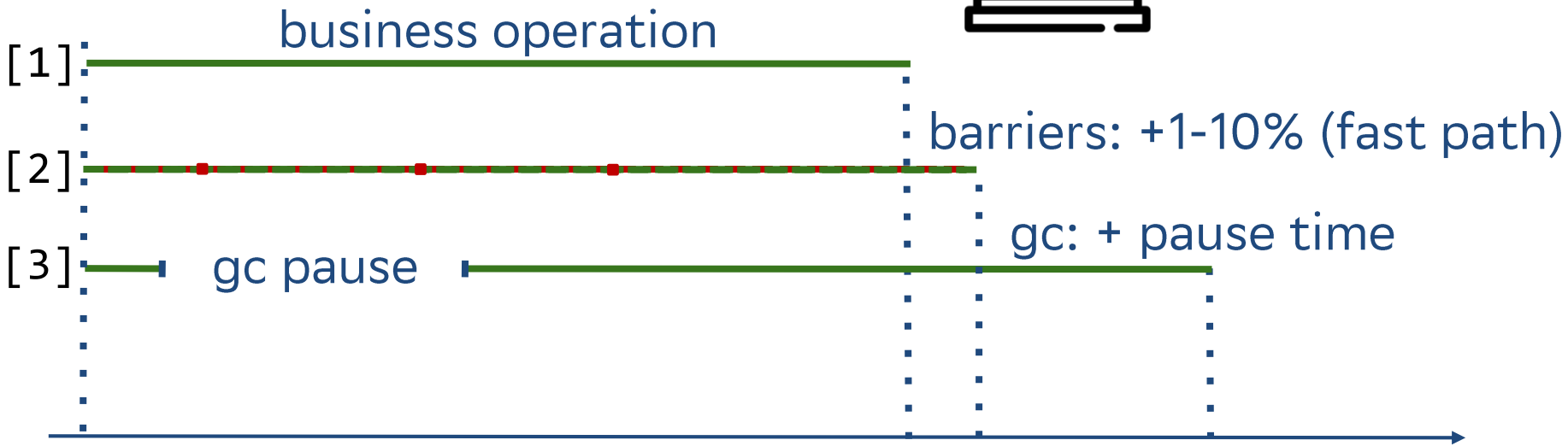
latency



STW vs Concurrent

throughput

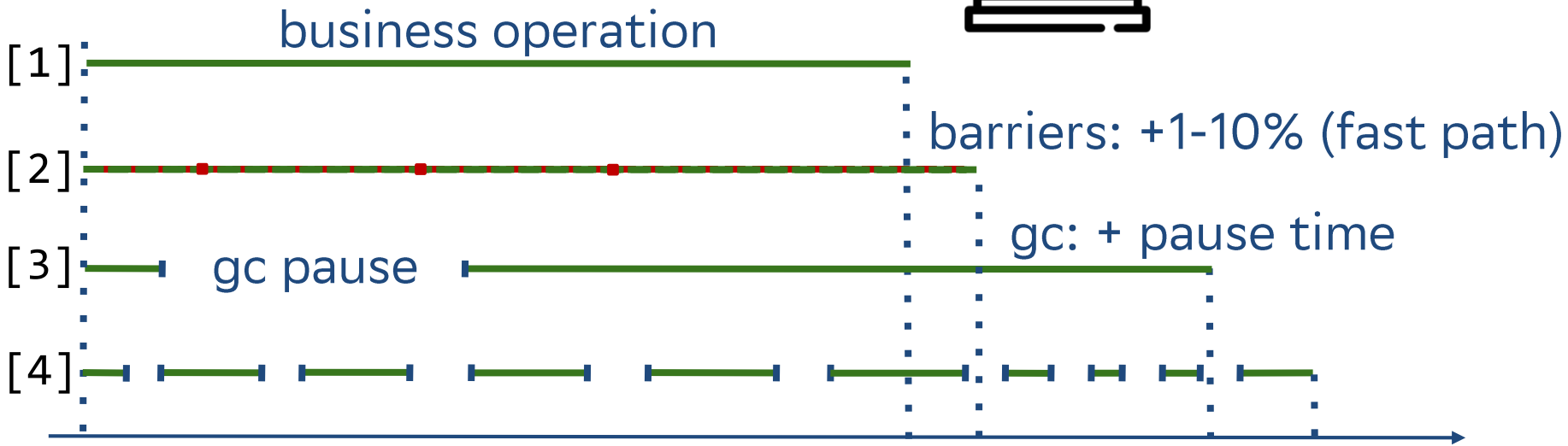
latency



STW vs Concurrent

throughput

latency



Q&A

эксперт Владимир Воскресенский

спикер Силин Дмитрий