ZSIM TUTORIAL – MEMORY SYSTEM

NATHAN BECKMANN

Massachusetts Institute of Technology

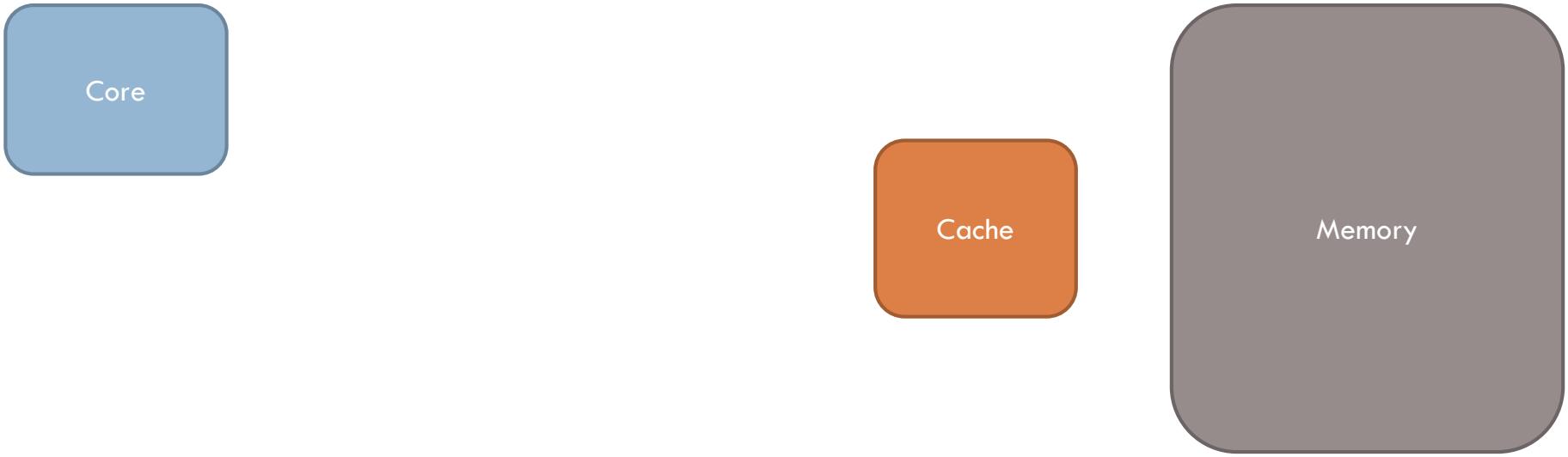
CSAIL

What we'll talk about

- ZSim has a full-featured memory system (originally designed for caches)

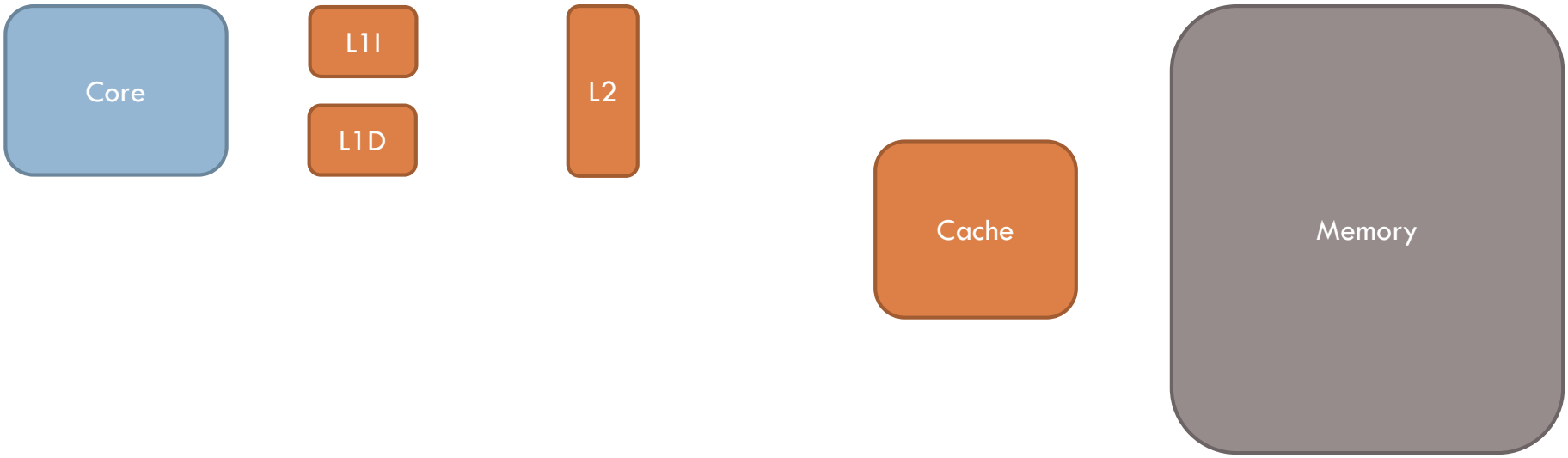
What we'll talk about

- ZSim has a full-featured memory system (originally designed for caches)



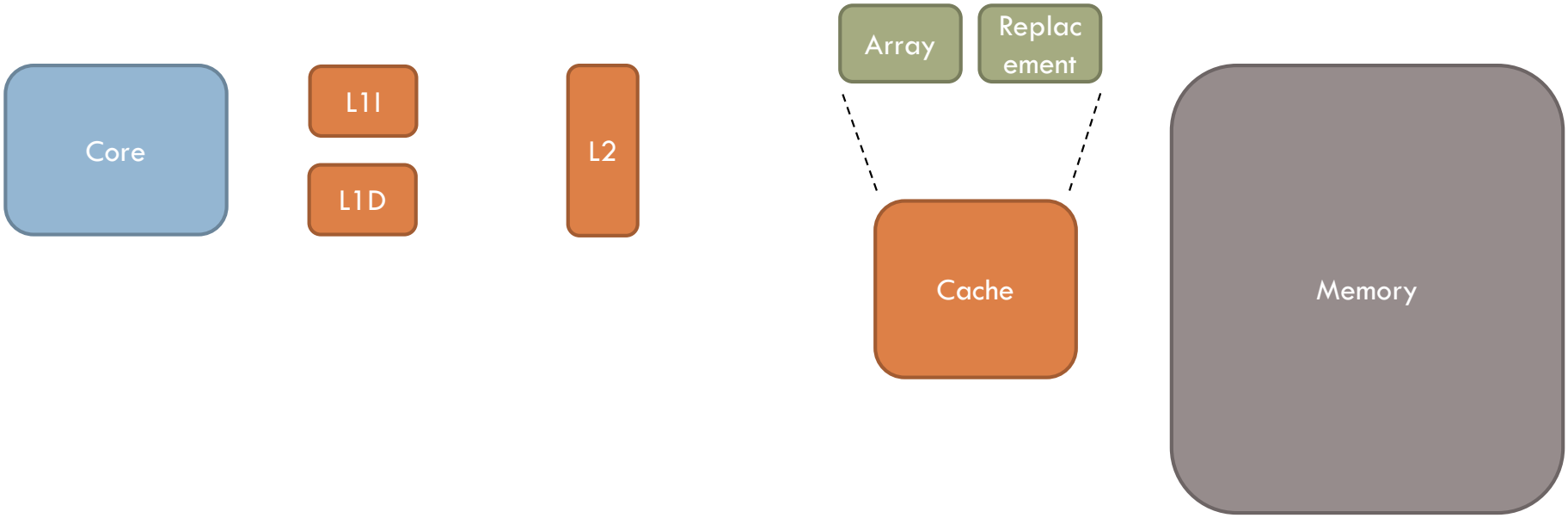
What we'll talk about

- ZSim has a full-featured memory system (originally designed for caches)



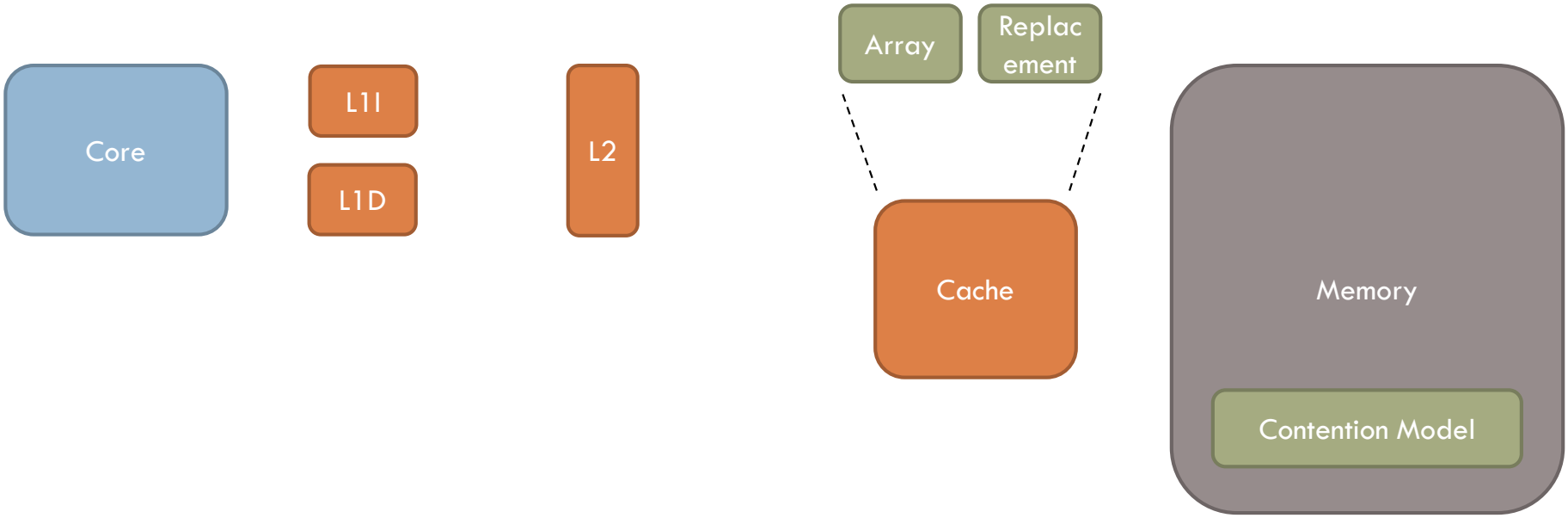
What we'll talk about

- ZSim has a full-featured memory system (originally designed for caches)



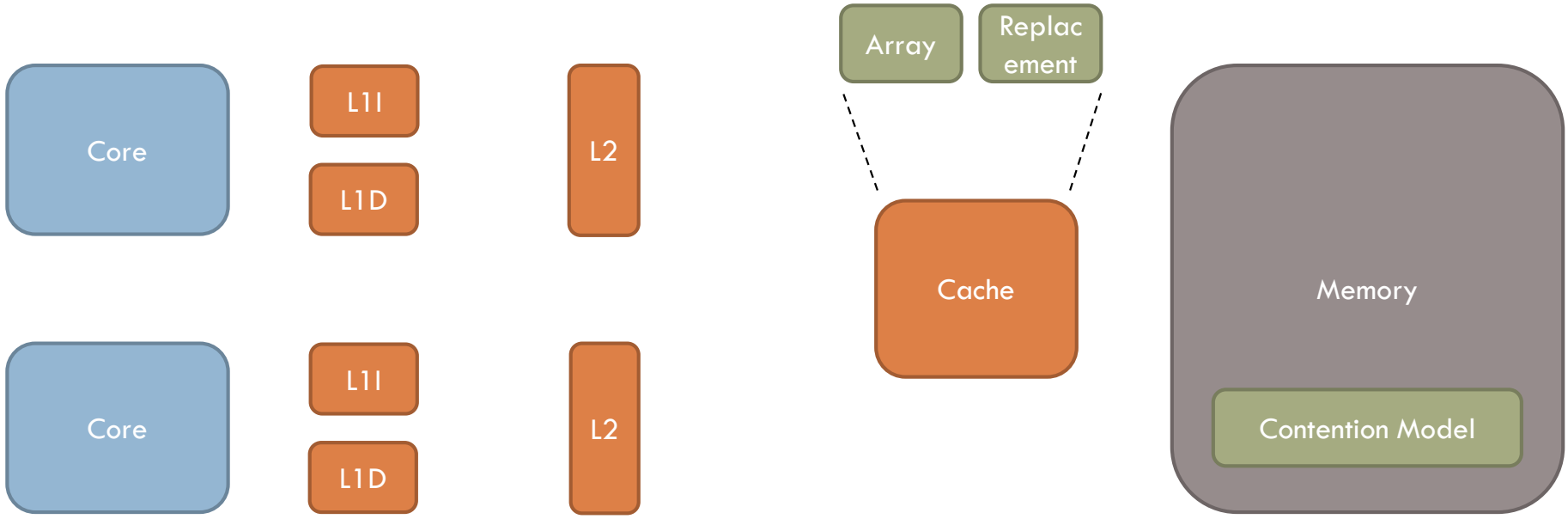
What we'll talk about

- ZSim has a full-featured memory system (originally designed for caches)



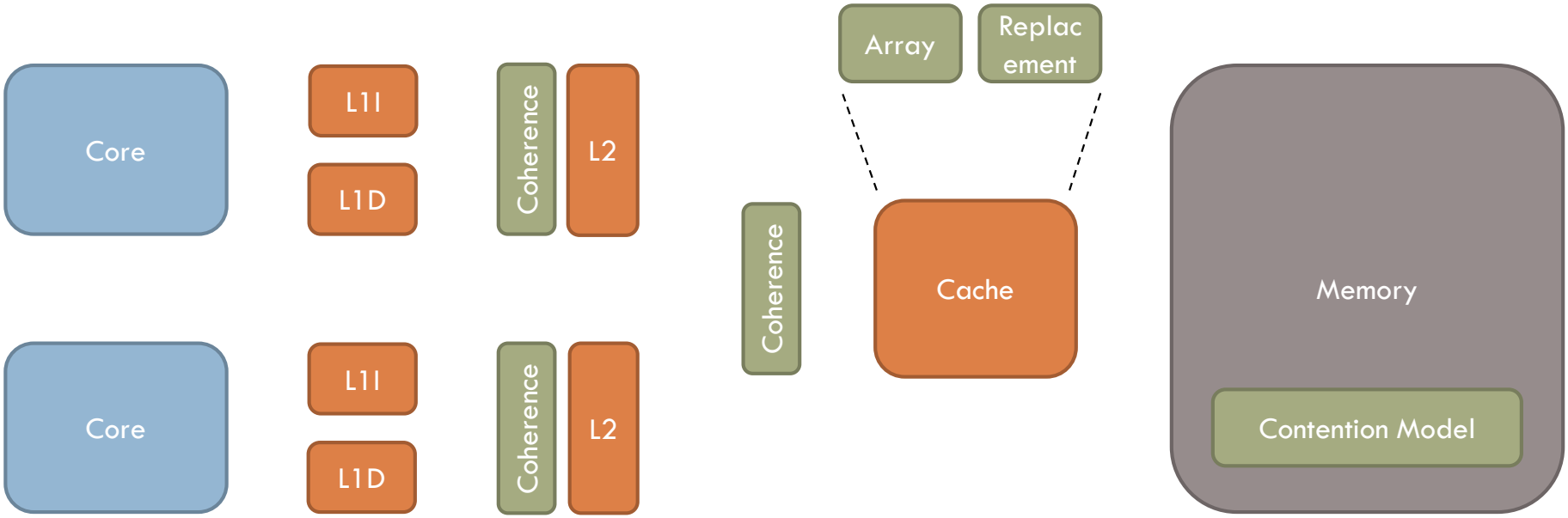
What we'll talk about

- ZSim has a full-featured memory system (originally designed for caches)



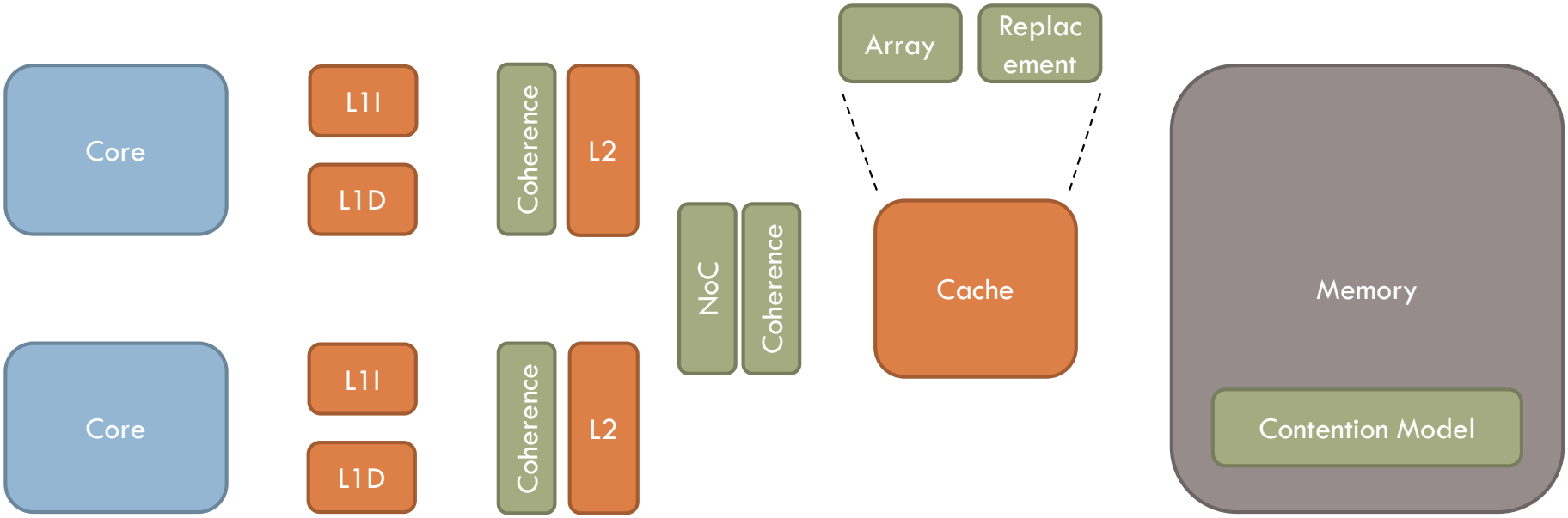
What we'll talk about

- ZSim has a full-featured memory system (originally designed for caches)



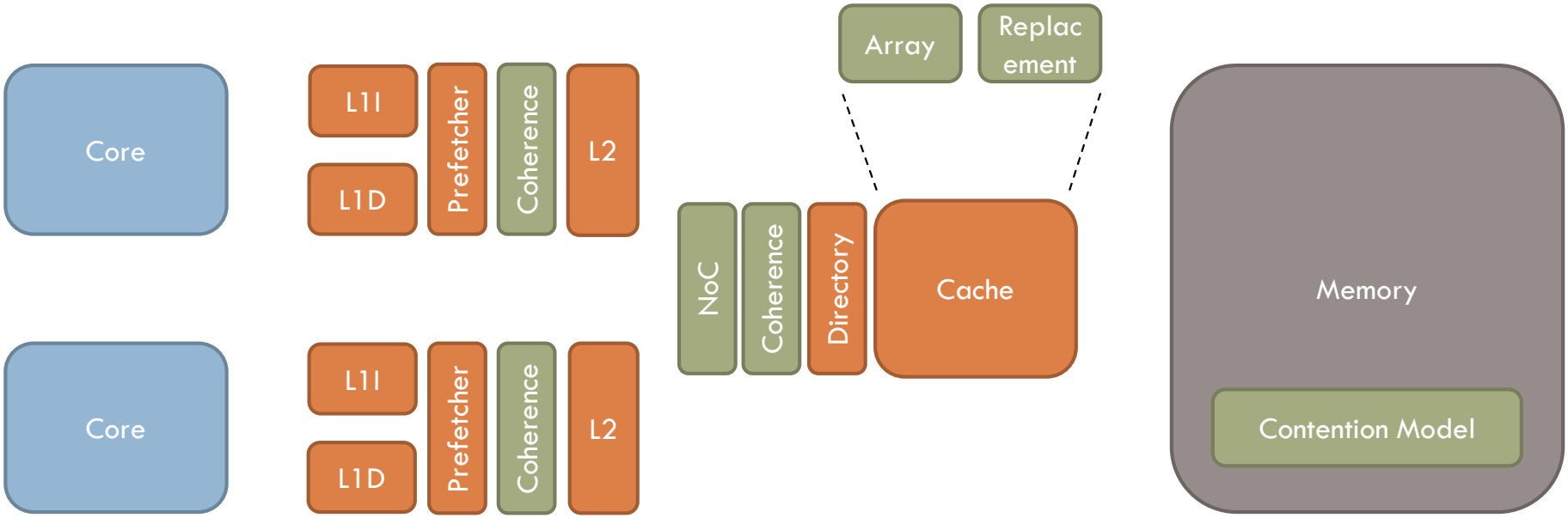
What we'll talk about

- ZSim has a full-featured memory system (originally designed for caches)



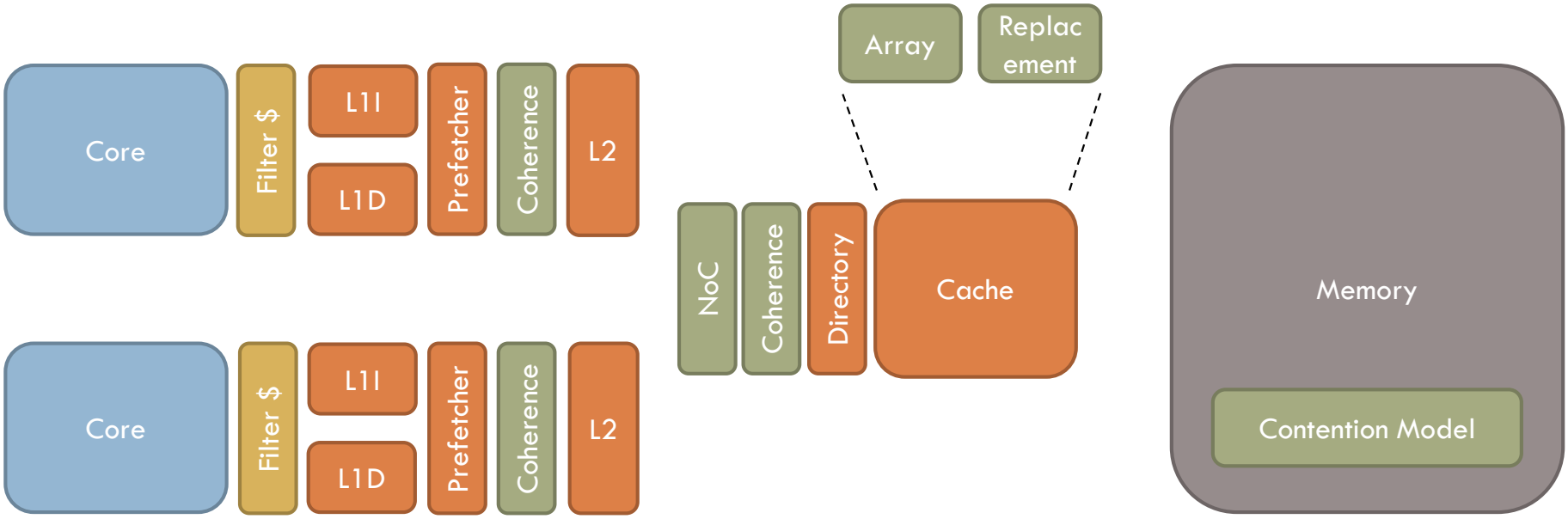
What we'll talk about

- ZSim has a full-featured memory system (originally designed for caches)



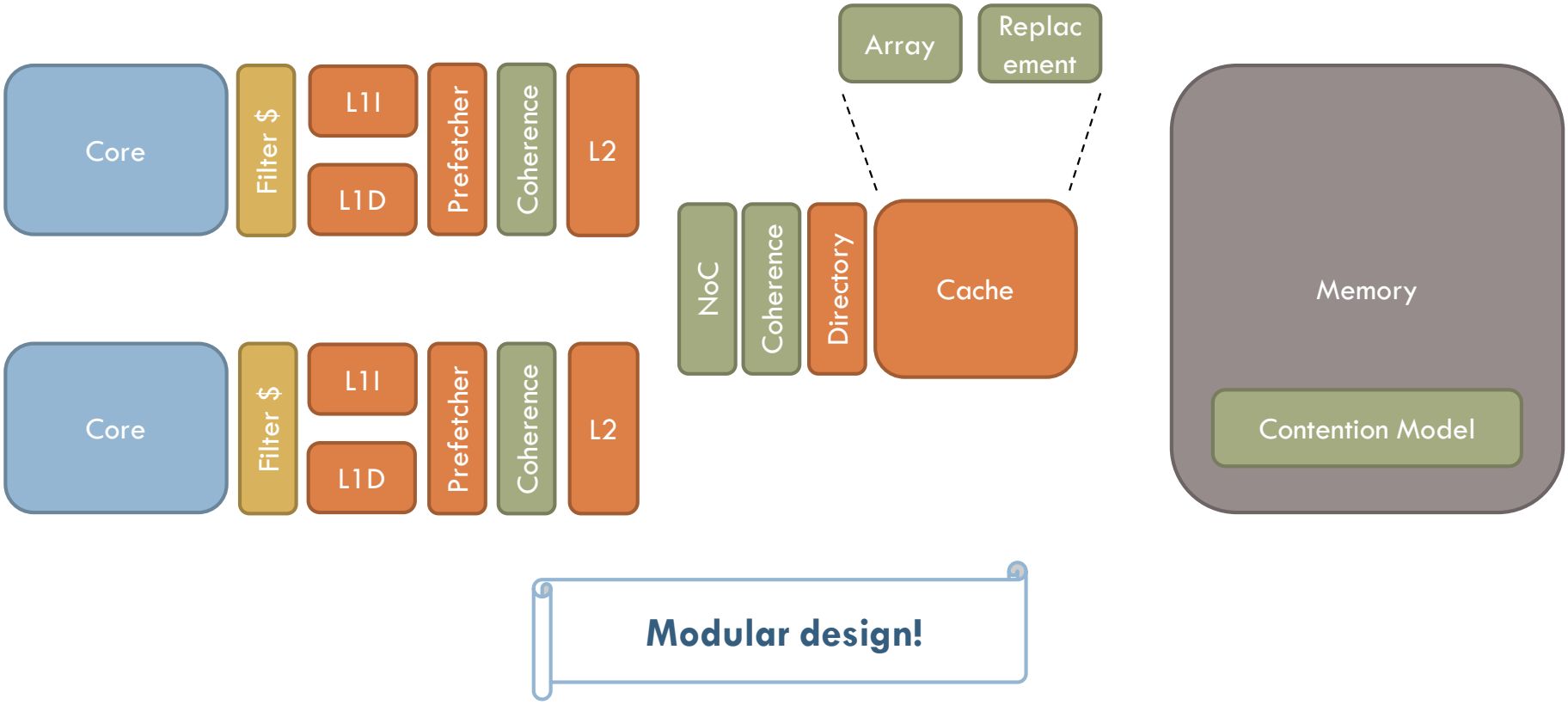
What we'll talk about

- ZSim has a full-featured memory system (originally designed for caches)



What we'll talk about

- ZSim has a full-featured memory system (originally designed for caches)

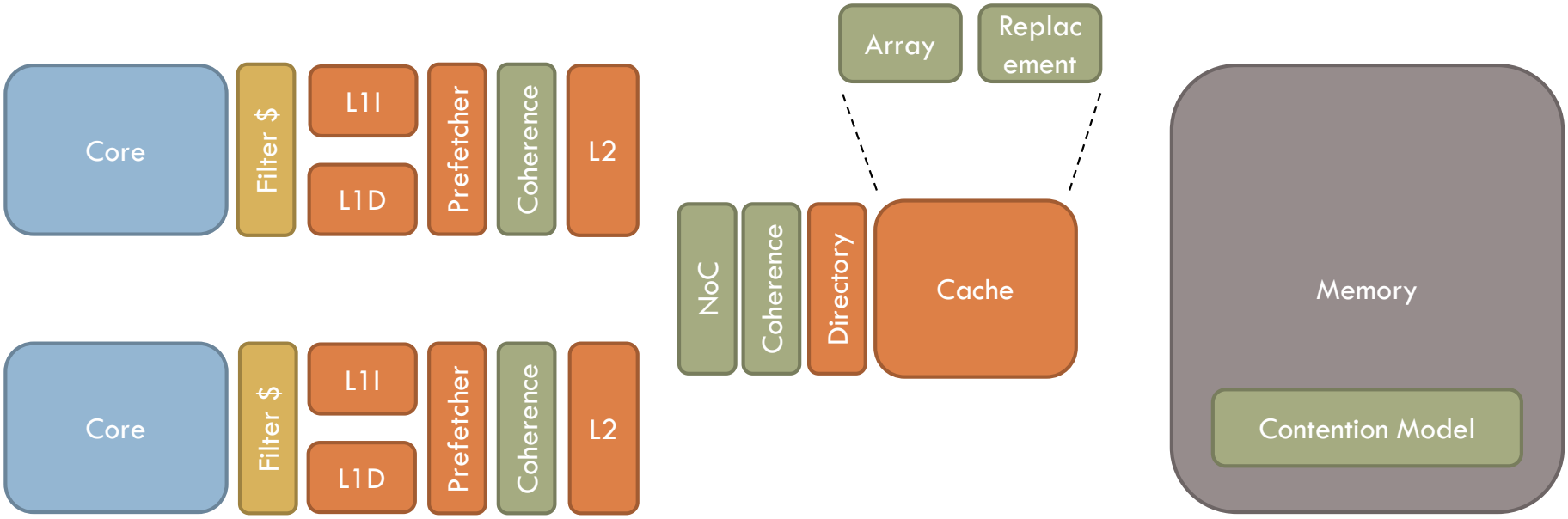


Topics covered

- ZSim memory system design & important classes/files
- Configuration options & available models
- How to extend zsim yourself (with example!)

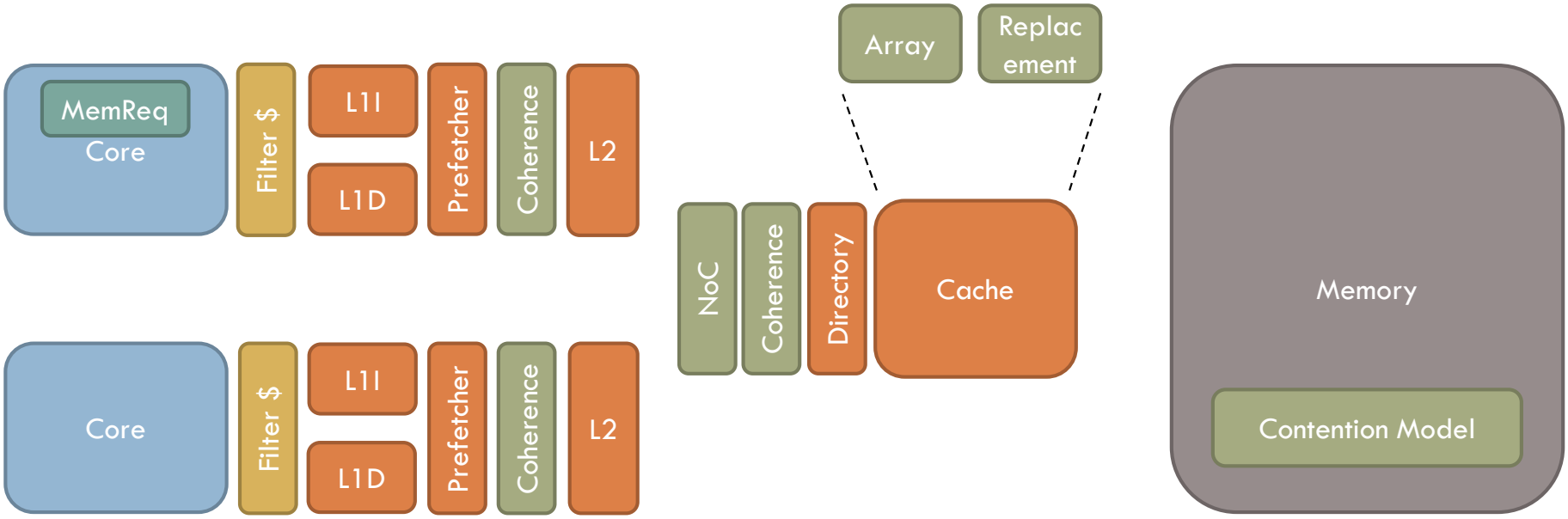
Example: "A day in the life of a memory request"

- Bound-phase function simulation
  - Some components add weave-phase modeling



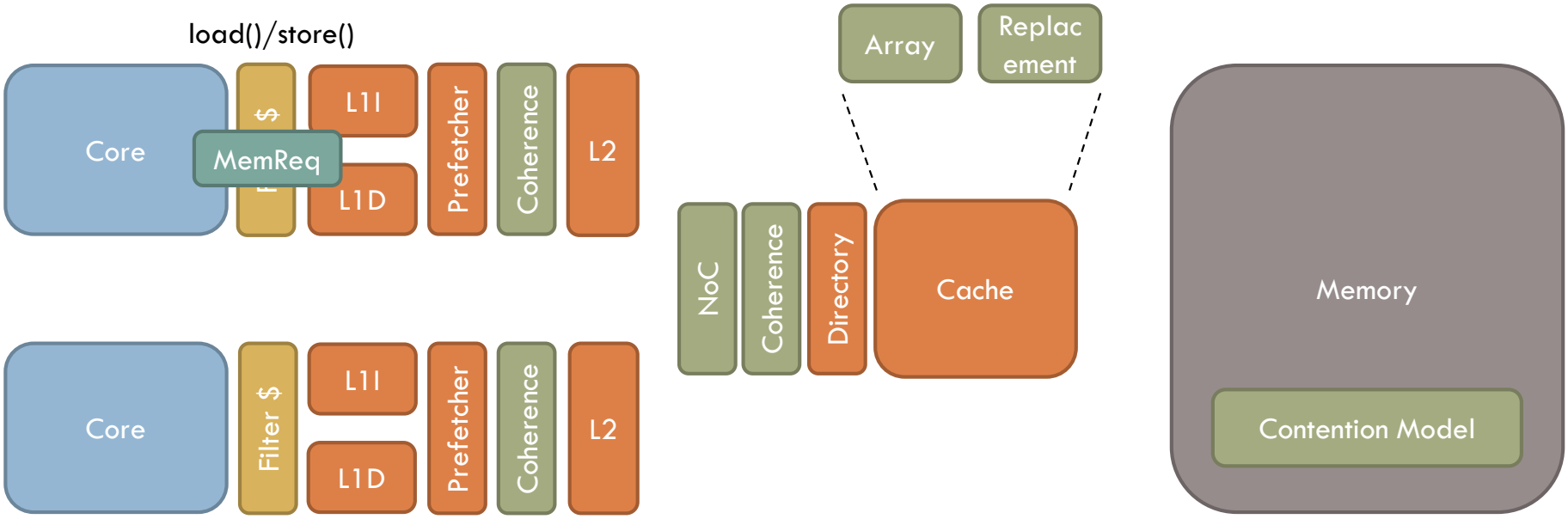
Example: “A day in the life of a memory request”

- Bound-phase function simulation
  - Some components add weave-phase modeling



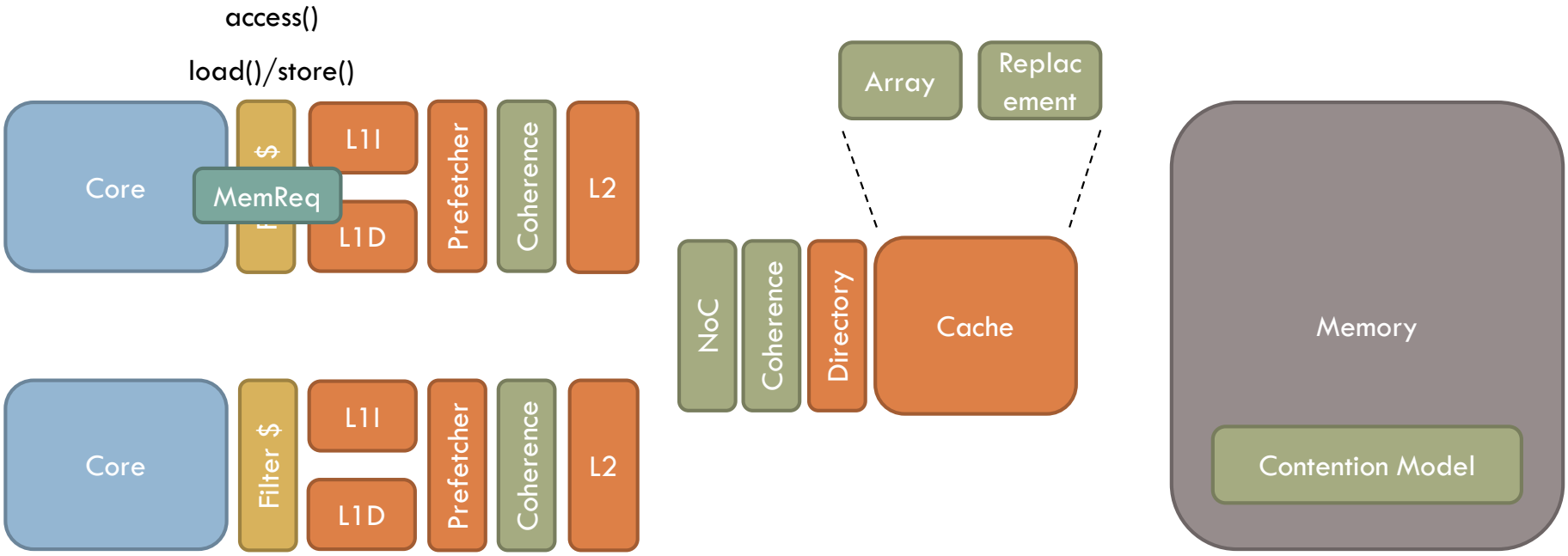
Example: "A day in the life of a memory request"

- Bound-phase function simulation
  - Some components add weave-phase modeling



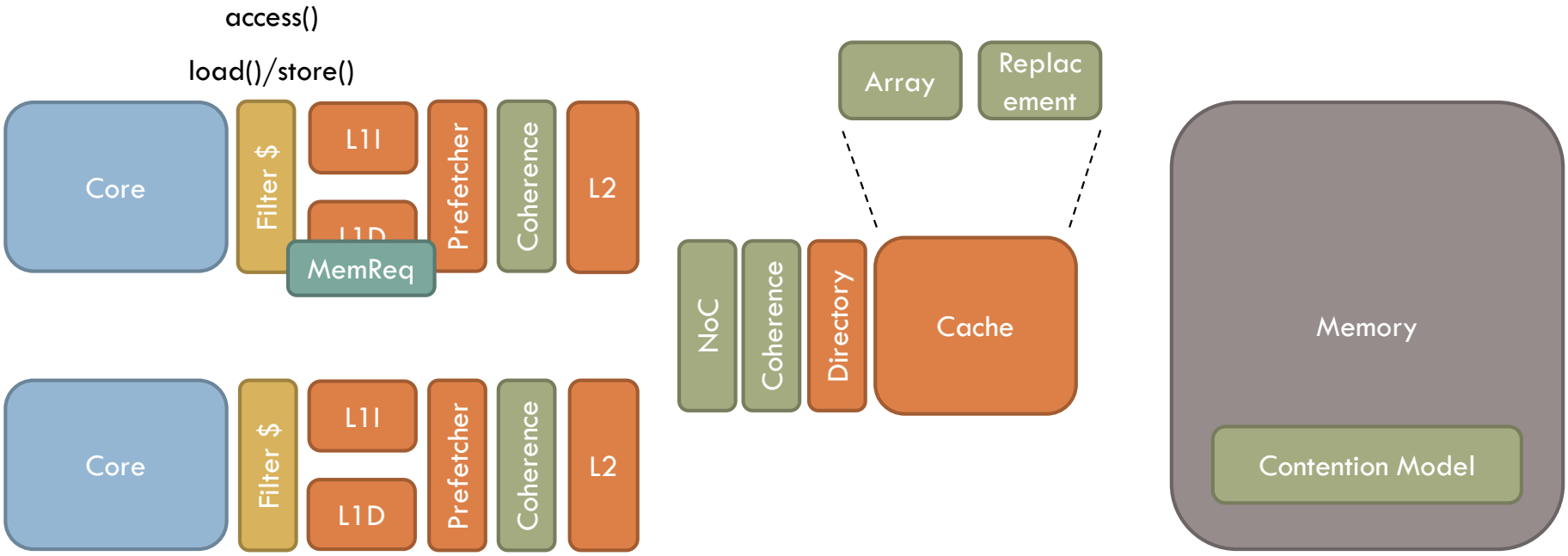
Example: "A day in the life of a memory request"

- Bound-phase function simulation
  - Some components add weave-phase modeling



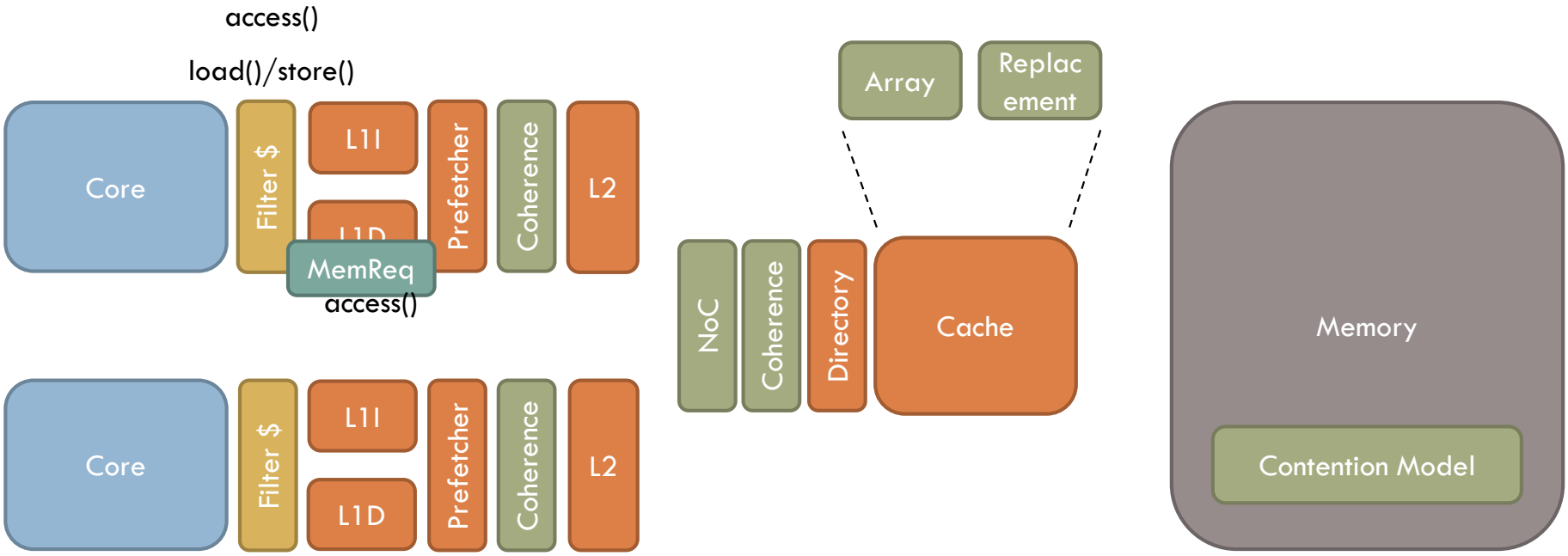
Example: "A day in the life of a memory request"

- Bound-phase function simulation
  - Some components add weave-phase modeling



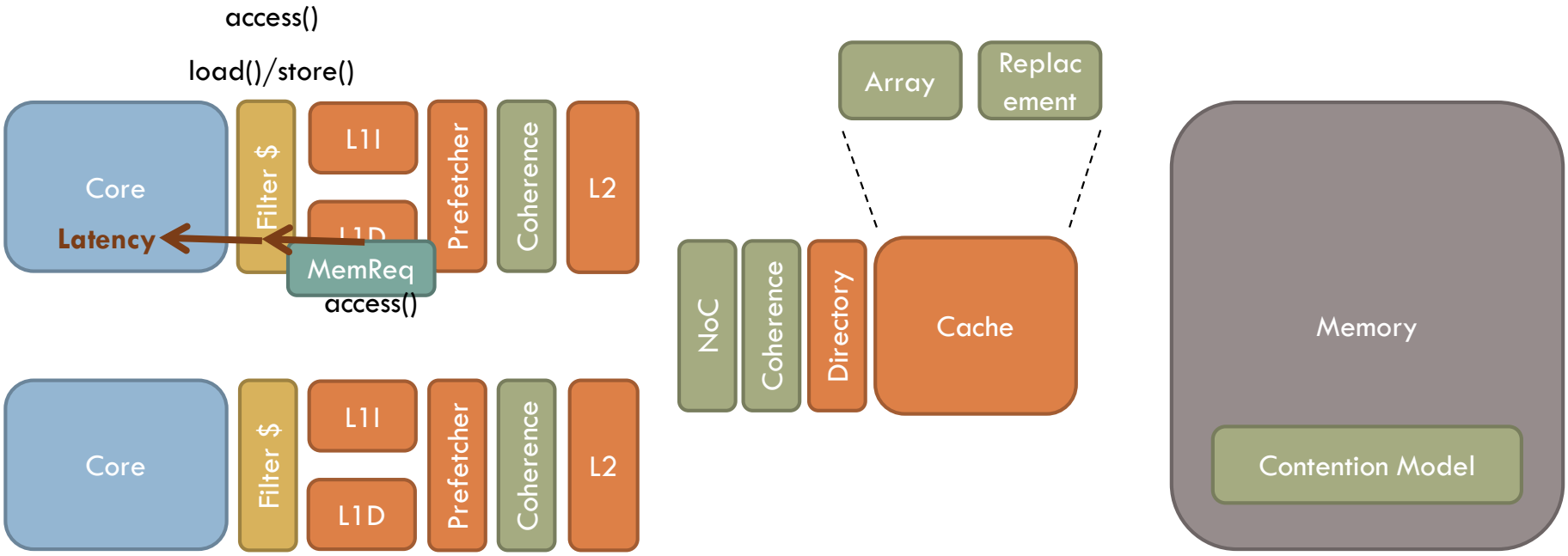
Example: "A day in the life of a memory request"

- Bound-phase function simulation
  - Some components add weave-phase modeling



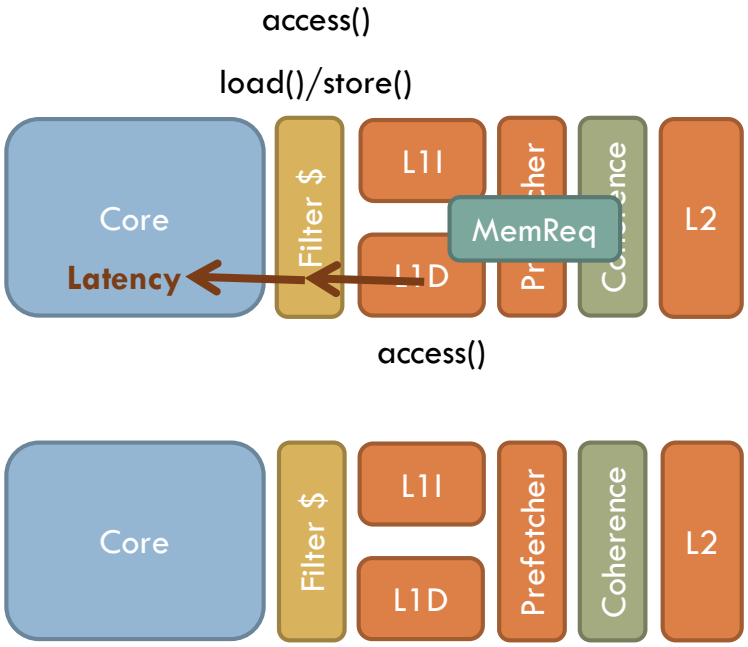
Example: "A day in the life of a memory request"

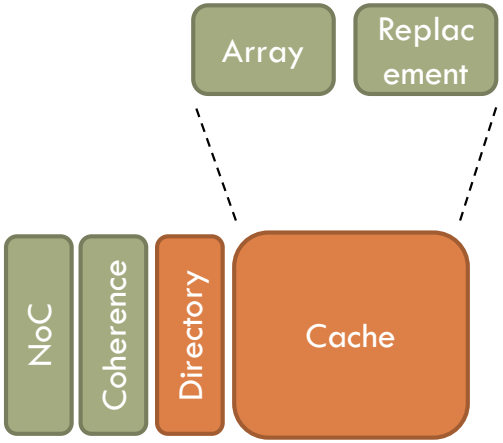
- Bound-phase function simulation
  - Some components add weave-phase modeling

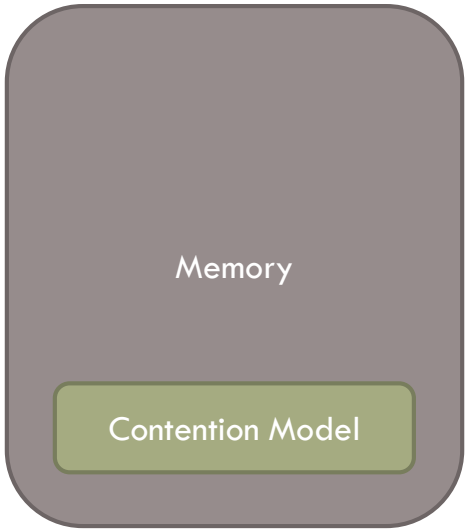


Example: “A day in the life of a memory request”

- Bound-phase function simulation
  - Some components add weave-phase modeling

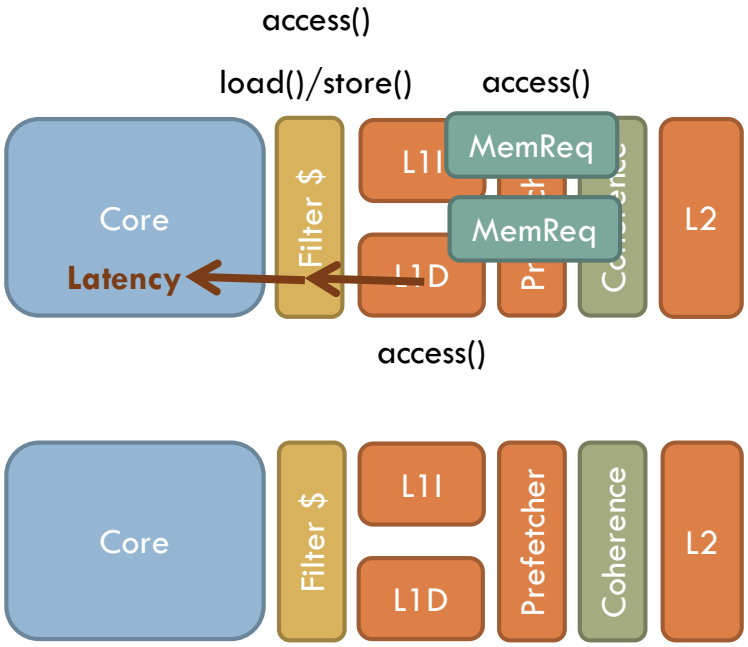


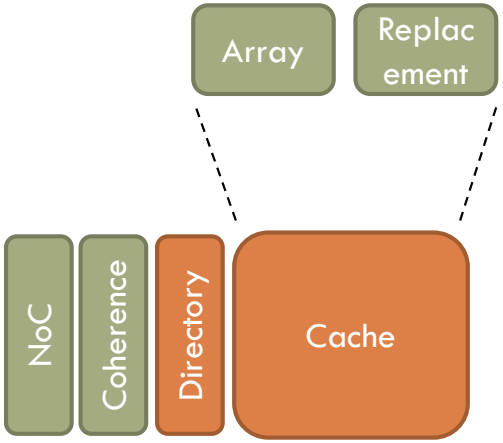


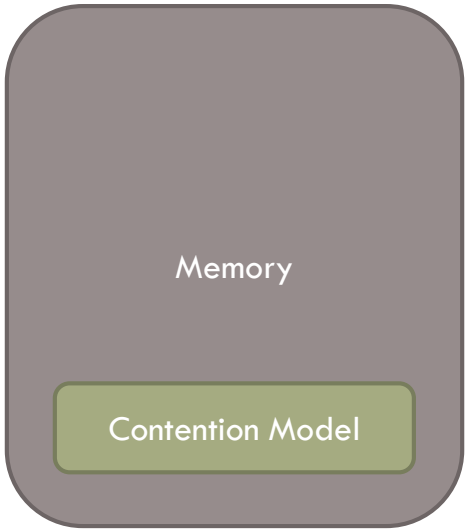


Example: "A day in the life of a memory request"

- Bound-phase function simulation
  - Some components add weave-phase modeling

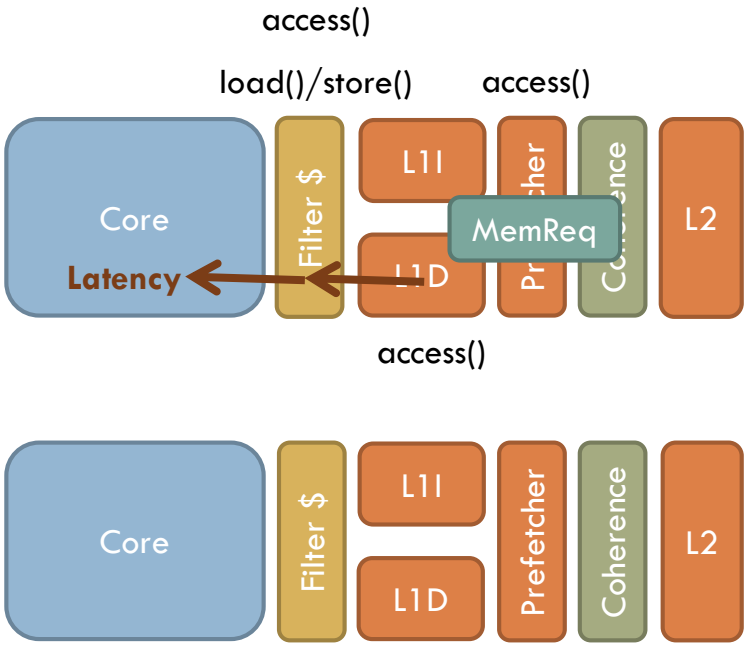


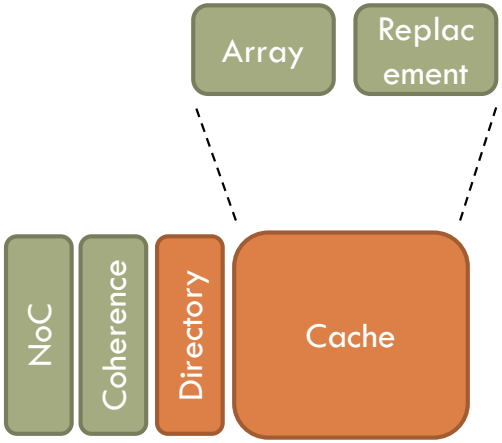


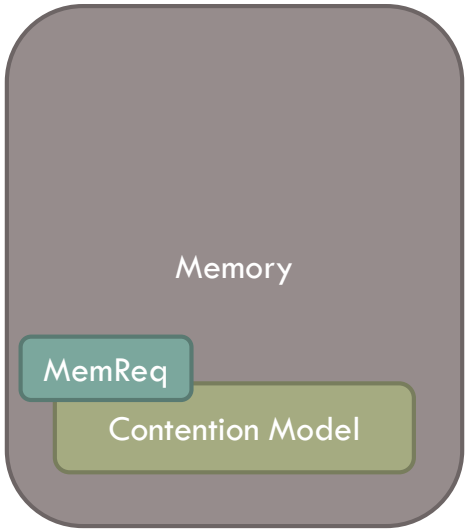


Example: “A day in the life of a memory request”

- Bound-phase function simulation
  - Some components add weave-phase modeling

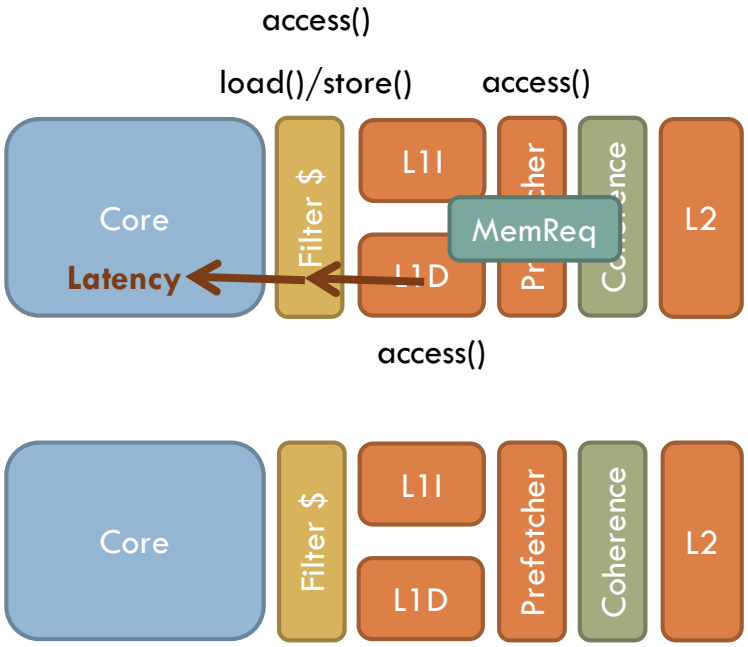


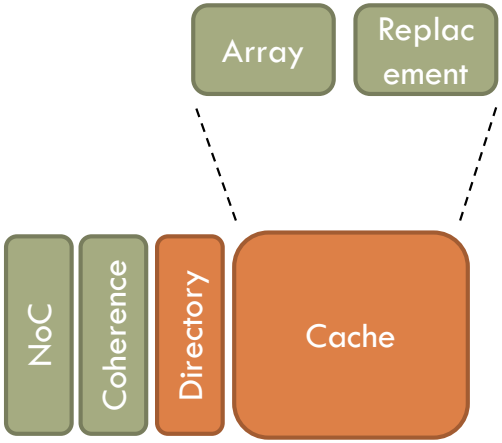


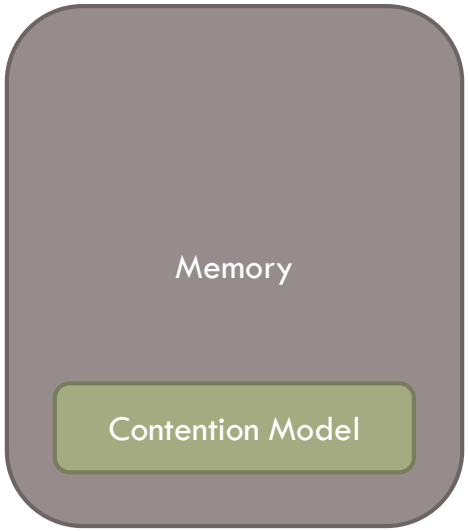


Example: “A day in the life of a memory request”

- Bound-phase function simulation
  - Some components add weave-phase modeling

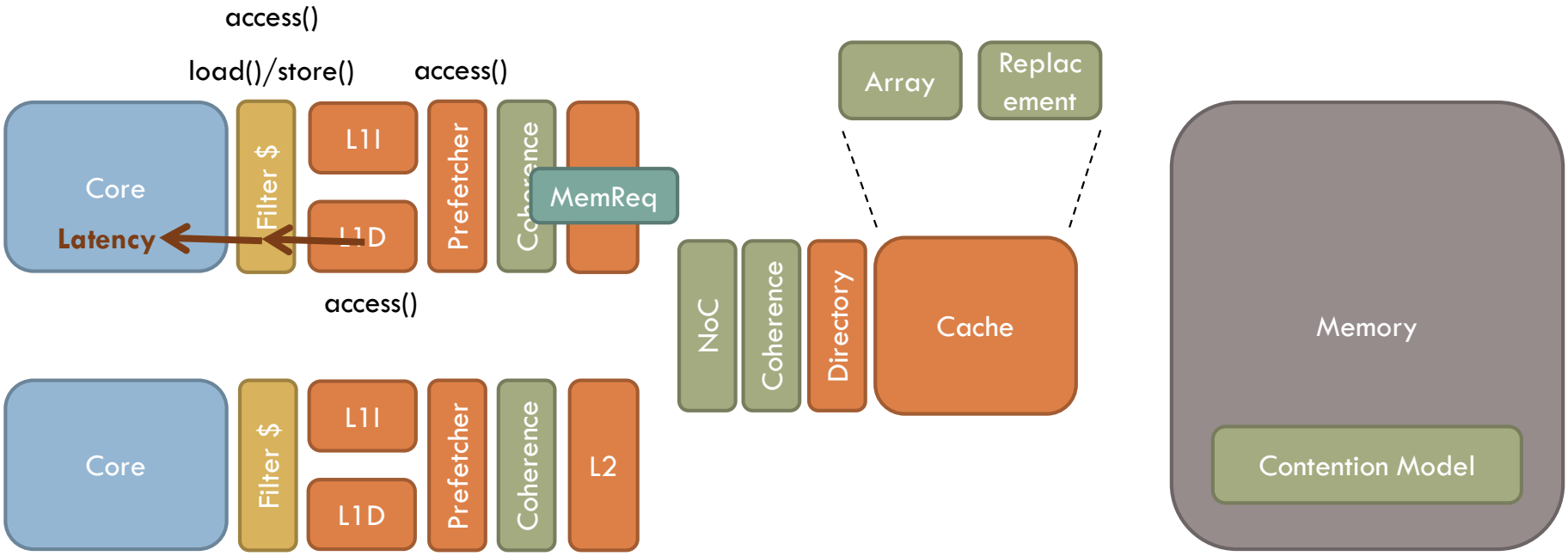






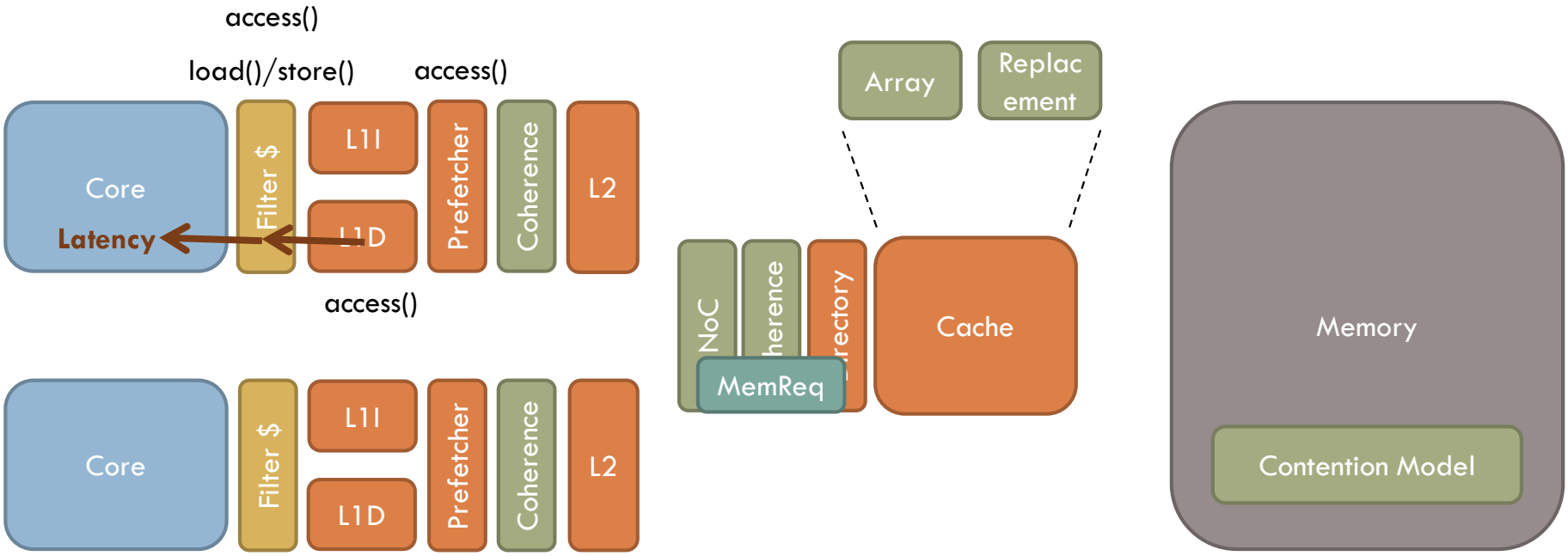
Example: "A day in the life of a memory request"

- Bound-phase function simulation
  - Some components add weave-phase modeling



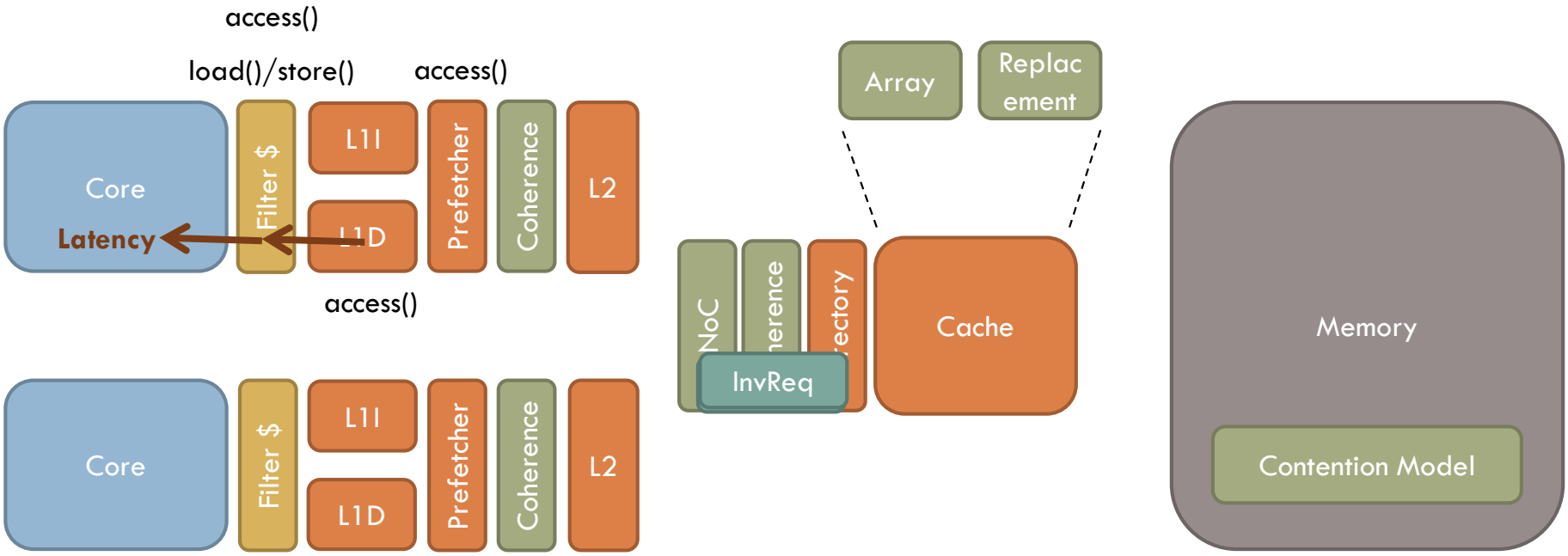
Example: "A day in the life of a memory request"

- Bound-phase function simulation
  - Some components add weave-phase modeling



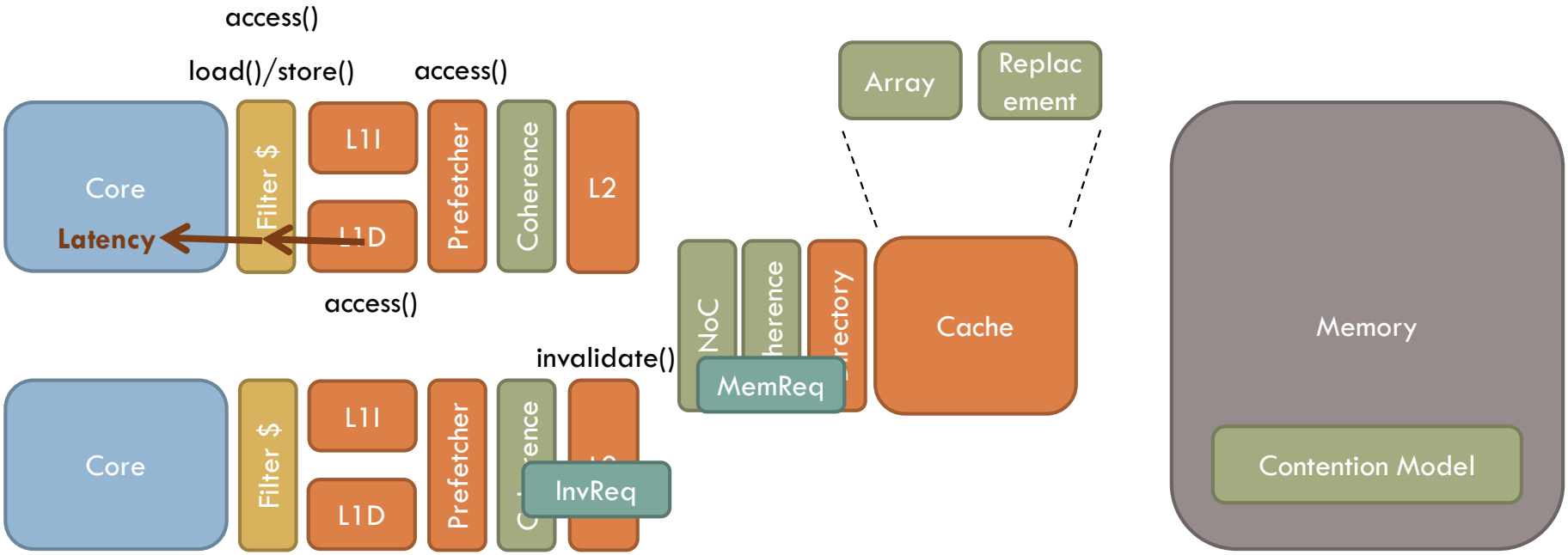
Example: "A day in the life of a memory request"

- Bound-phase function simulation
  - Some components add weave-phase modeling



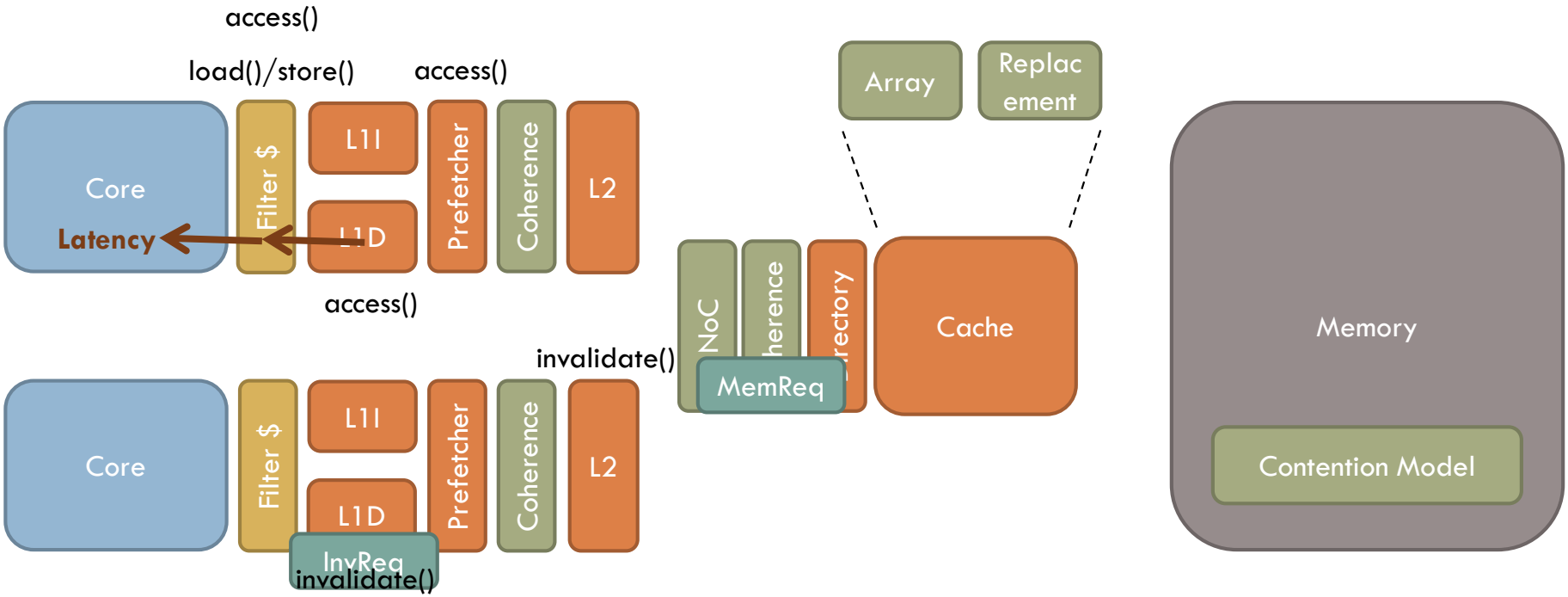
Example: "A day in the life of a memory request"

- Bound-phase function simulation
  - Some components add weave-phase modeling



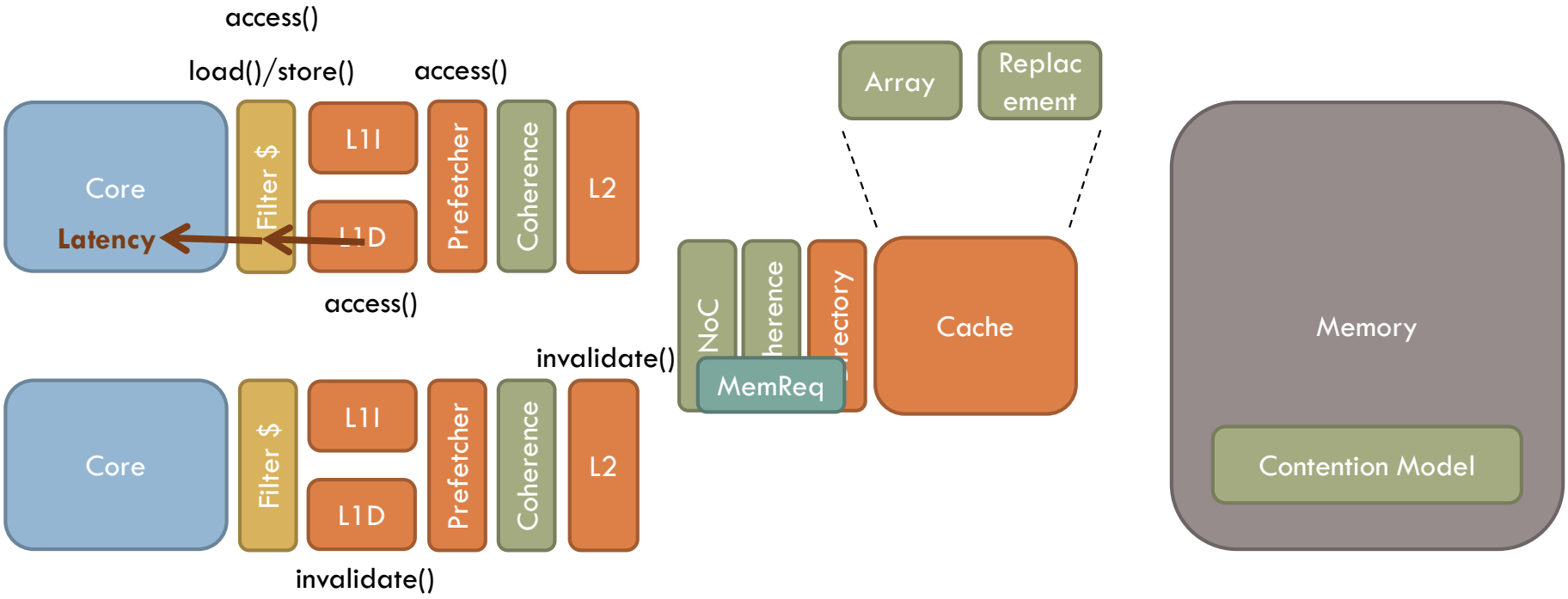
Example: "A day in the life of a memory request"

- Bound-phase function simulation
  - Some components add weave-phase modeling



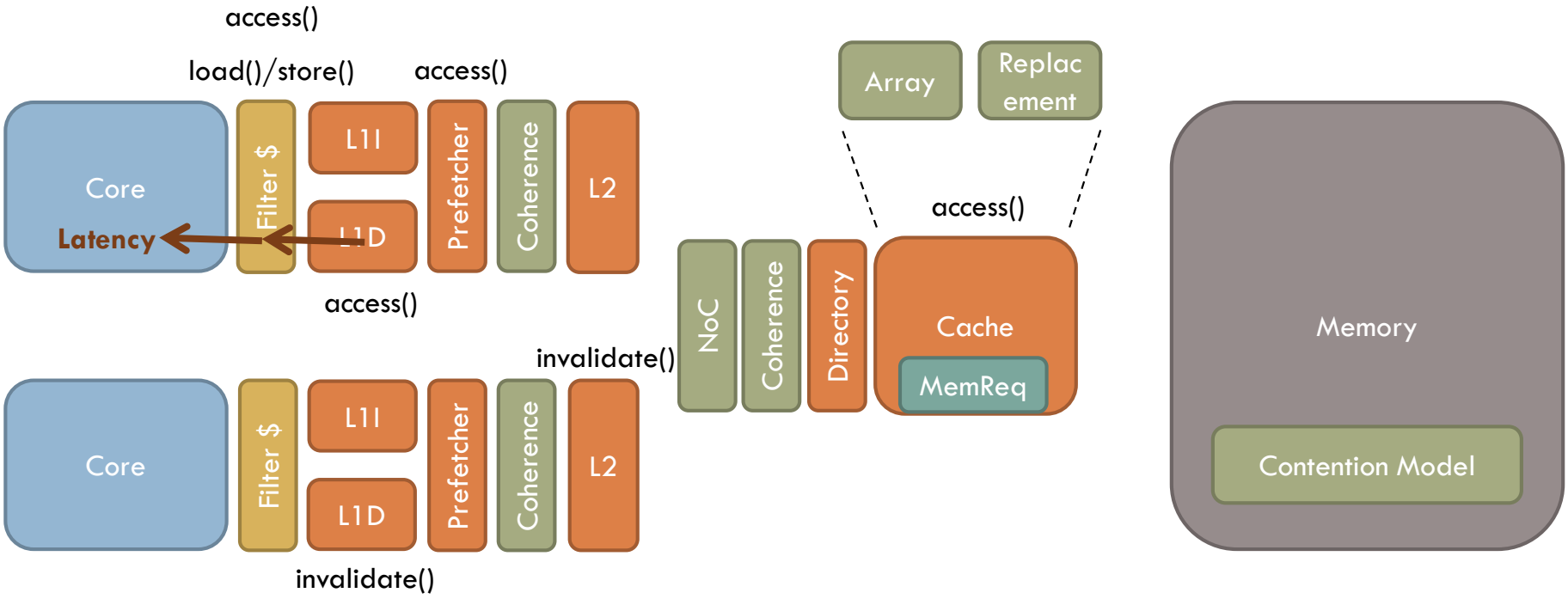
Example: "A day in the life of a memory request"

- Bound-phase function simulation
  - Some components add weave-phase modeling



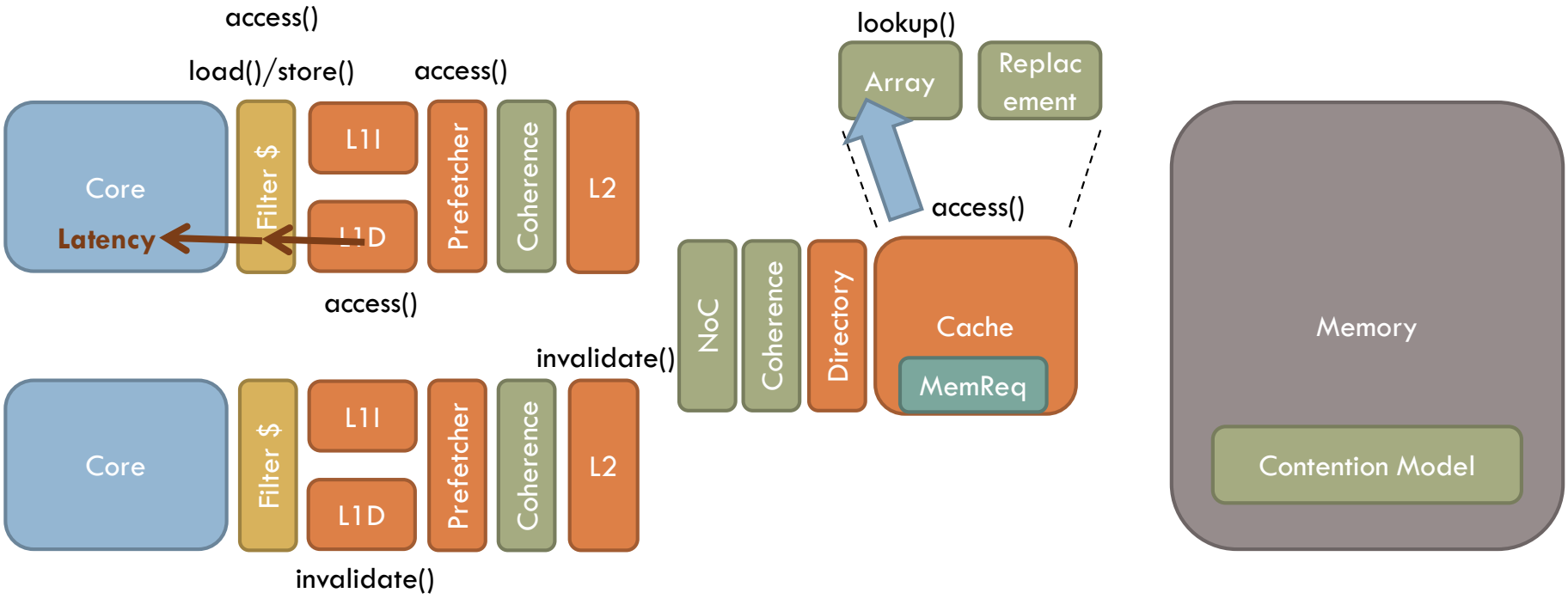
Example: "A day in the life of a memory request"

- Bound-phase function simulation
  - Some components add weave-phase modeling



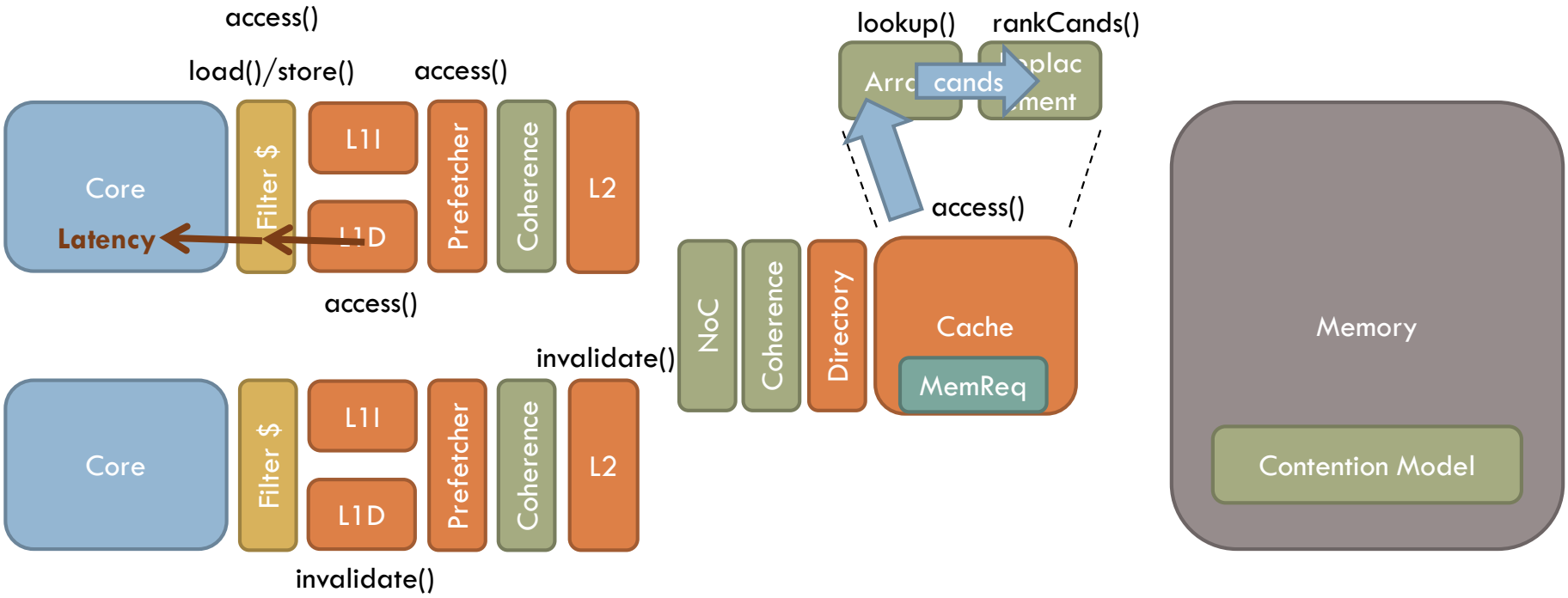
Example: "A day in the life of a memory request"

- Bound-phase function simulation
  - Some components add weave-phase modeling



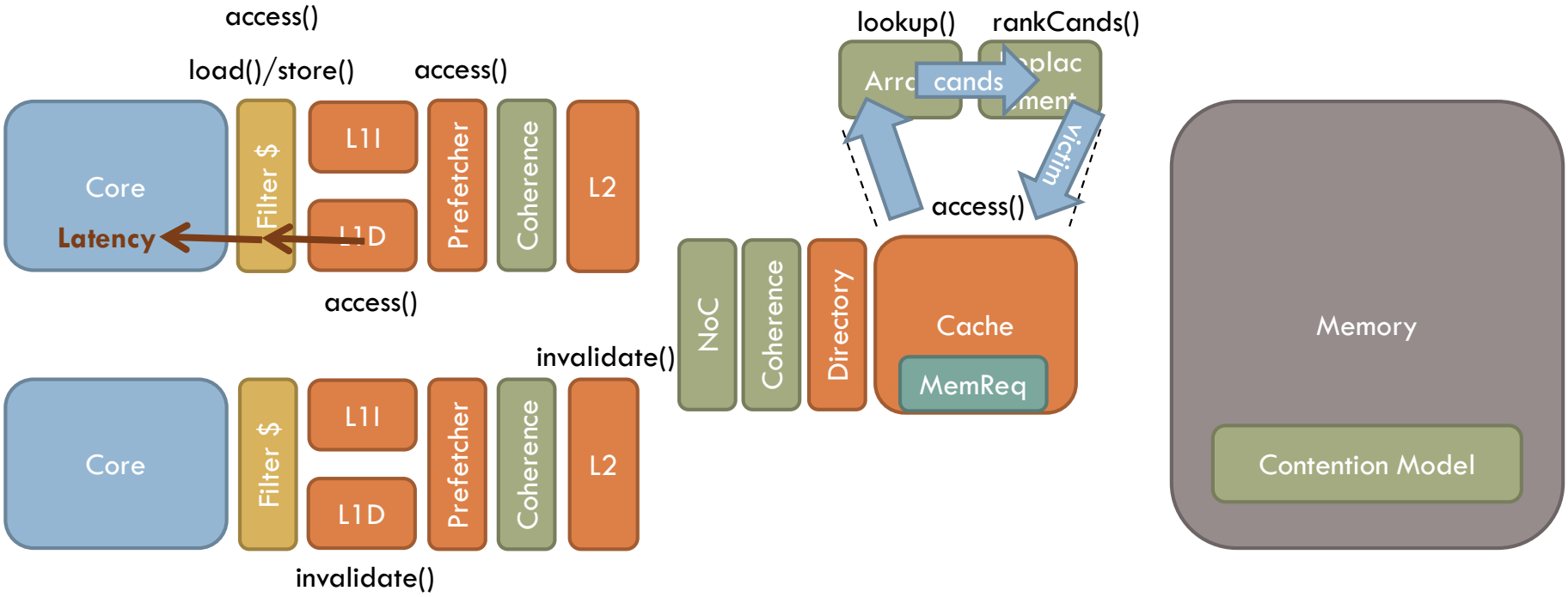
Example: "A day in the life of a memory request"

- Bound-phase function simulation
  - Some components add weave-phase modeling



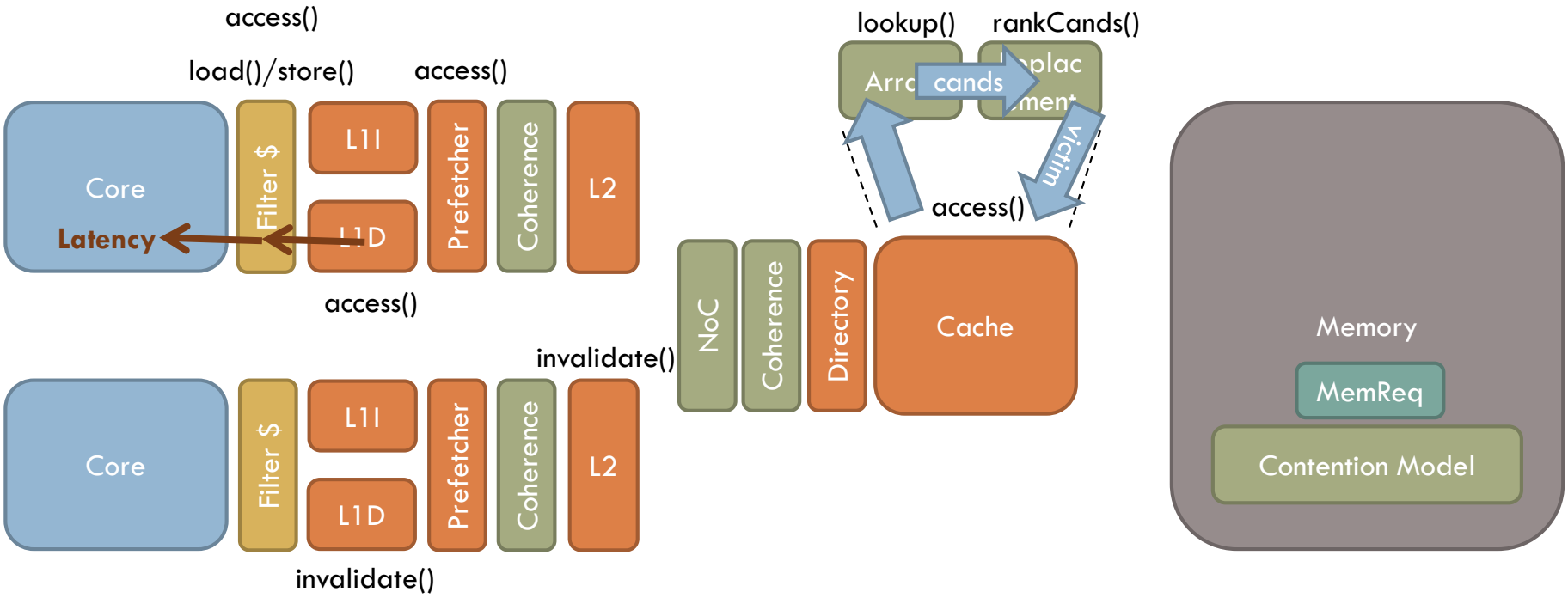
Example: "A day in the life of a memory request"

- Bound-phase function simulation
  - Some components add weave-phase modeling



Example: "A day in the life of a memory request"

- Bound-phase function simulation
  - Some components add weave-phase modeling



Important ZSim memory classes

MemReq

MemReq

- Represents an in-flight memory request

- Important fields:
  - uint64_t lineAddr – shifted address
  - AccessType type – GETS, GETX, PUTS, PUTX
  - uint64_t cycle – requesting cycle
  - MESIState* state – coherence state (M, E, S, or I)

- Important methods:
  - N/A

Important ZSim memory classes

MemReq

Important ZSim memory classes

MemReq

MemObject

MemObject

- Generic interface for things that handle memory requests
- Important fields:
  - N/A
- Important methods:
  - uint64_t access(MemReq& req) – performs an access and returns completion time

Implementing a simple model for main memory


```
class SimpleMemory : public MemObject {
    uint64_t latency;
    g_string name;

public:
    SimpleMemory(uint64_t _latency, g_string _name)
        : latency(_latency), name(_name) {};
    const char* getName() { return name.c_str(); }

    uint64_t access(MemReq& req) {
        switch (req.type) {
            case PUTS: case PUTX: // write
                *req.state = I;
            case GETS:
                *req.state = req.is(MemReq::NOEXCL)? S : E;
            case GETX:
                *req.state = M;
        }
        return req.cycle + latency;
    }
};
```

Implementing a simple model for main memory


```
class SimpleMemory : public MemObject {
    uint64_t latency;
    g_string name;

public:
    SimpleMemory(uint64_t _latency, g_string _name)
        : latency(_latency), name(_name) {};
    const char* getName() { return name.c_str(); }

    uint64_t access(MemReq& req) {
        switch (req.type) {
            case PUTS: case PUTX: // write
                *req.state = I;
            case GETS:
                *req.state = req.is(MemReq::NOEXCL)? S : E;
            case GETX:
                *req.state = M;
        }
        return req.cycle + latency;
    }
};
```

Set coherence in requestor

Implementing a simple model for main memory


```
class SimpleMemory : public MemObject {
    uint64_t latency;
    g_string name;

public:
    SimpleMemory(uint64_t _latency, g_string _name)
        : latency(_latency), name(_name) {};
    const char* getName() { return name.c_str(); }

    uint64_t access(MemReq& req) {
        switch (req.type) {
            case PUTS: case PUTX: // write
                *req.state = I;
            case GETS:
                *req.state = req.is(MemReq::NOEXCL)? S : E;
            case GETX:
                *req.state = M;
        }
        return req.cycle + latency;
    }
};
```

Set coherence in requestor

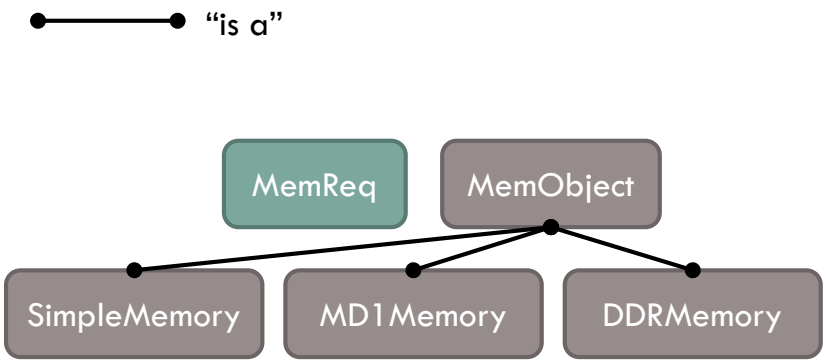
Completion cycle

Important ZSim memory classes

MemReq

MemObject

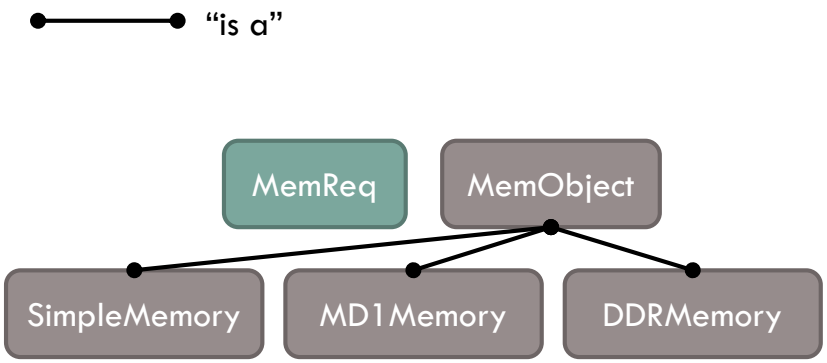
Important ZSim memory classes



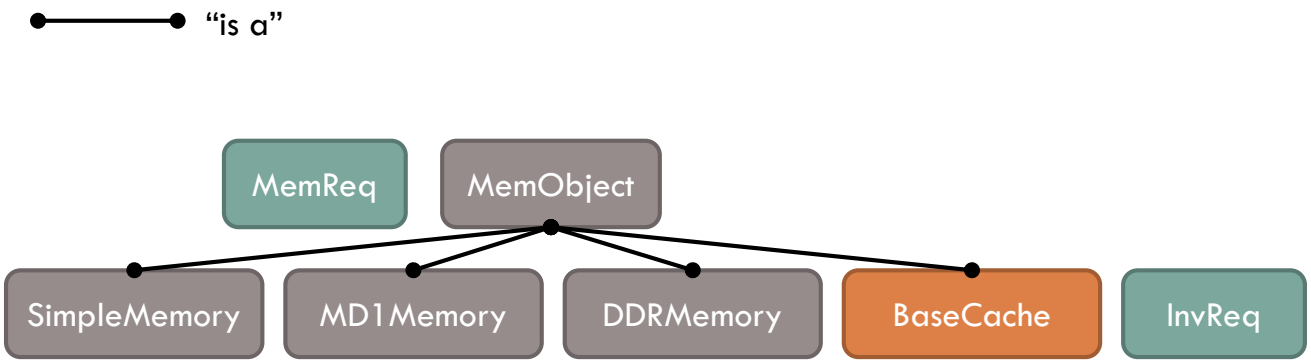
Memory controllers

- Different models for main memory
- SimpleMemory: fixed-latency, no contention
  - Important fields: latency
- MD1Memory: contention modeled using M/D/1 queue
  - Important fields: megabytesPerSecond (bandwidth), zeroLoadLatency, etc.
- DDRMemory & DRAMSimMemory: detailed modeling of DDR timings
  - Important fields: *lots* of configuration parameters (CAS, RAS, bus MHz)
  - Timings modeled in weave-phase
  - Requires TimingCore or OOO core models
  - Similar accuracy, but DDRMemory is much faster

Important ZSim memory classes



Important ZSim memory classes



InvReq

- Represents an invalidation request from coherence controller/directory

- Important fields:
  - uint64_t lineAddr – shifted address
  - InvType type – INV, INVX, FWD
  - uint64_t cycle – requesting cycle

- Important methods:
  - N/A

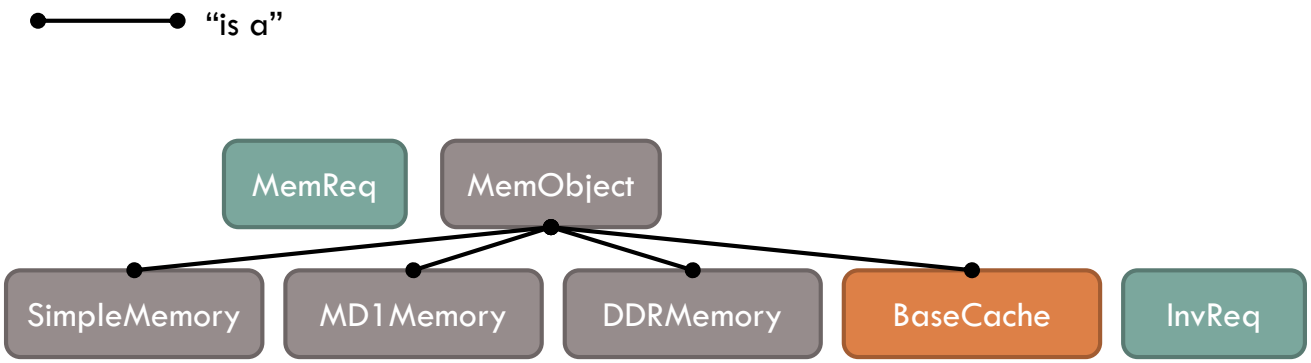
BaseCache

- Generic interface for cache-like objects

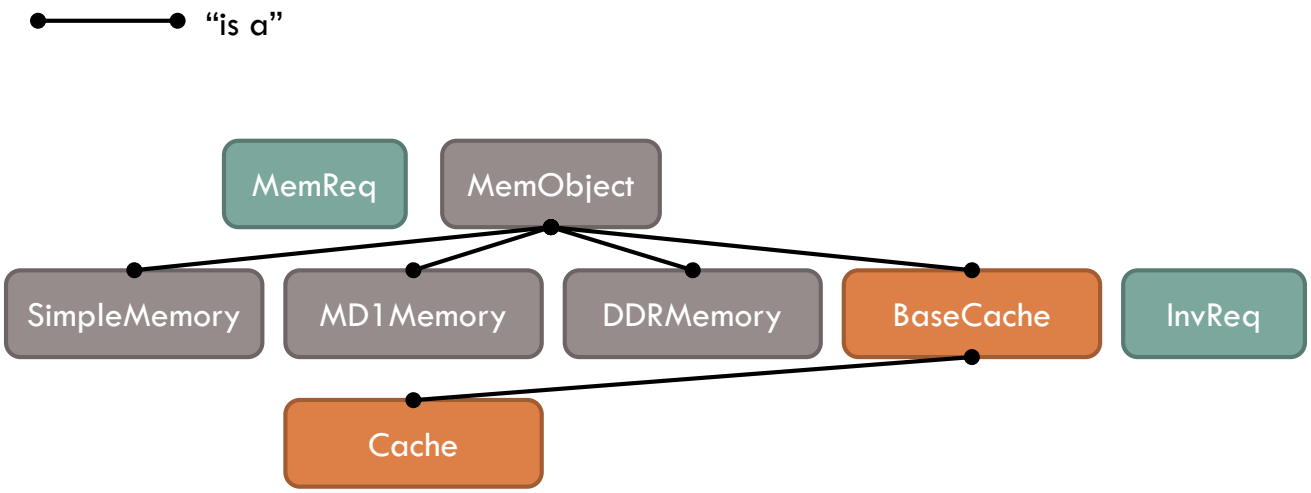
- Important fields:
  - N/A

- Important methods:
  - void setParents(…) – register the caches above it in the hierarchy
  - void setChildren(…) – register the caches below it in the hierarchy
  - uint64_t invalidate(const InvReq& req) – invalidate line locally & in children
  - uint64_t access(MemReq& req)

Important ZSim memory classes



Important ZSim memory classes



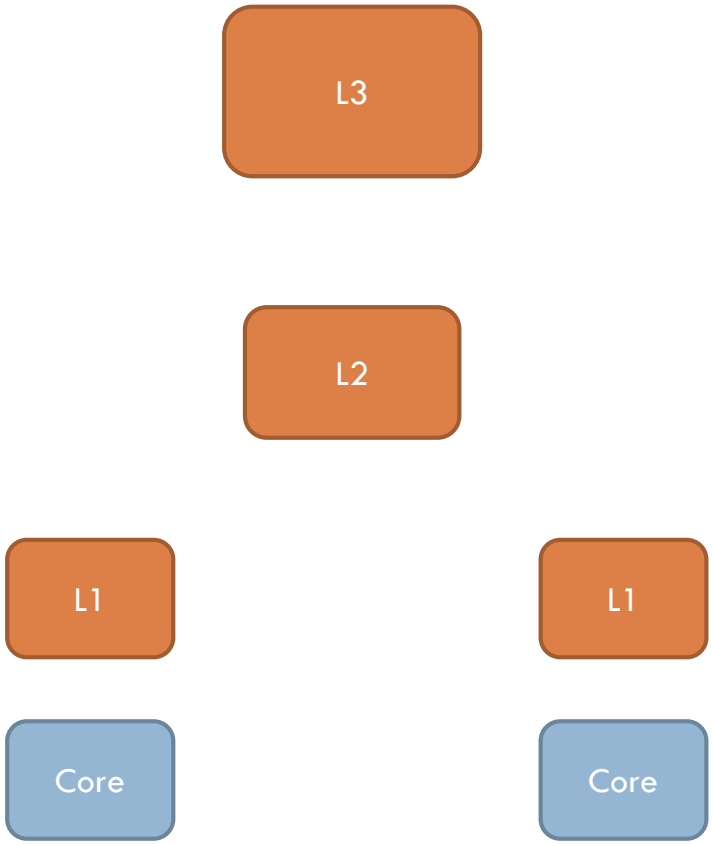
Cache

- Inclusive cache
  - Contains tag array, coherence controller, replacement policy (discussed later)
  - Adds logic to control these components

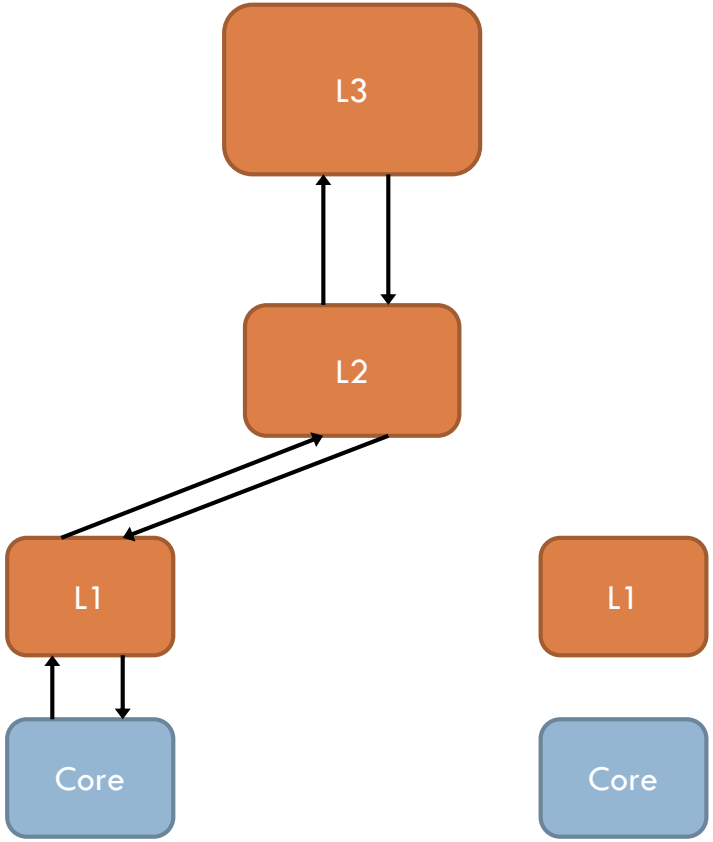
- Important fields (that aren't discussed later):
  - uint32_t accLat – access latency
  - uint32_t invLat – invalidation latency

- Important methods:
  - void setParents(…) – register the caches above it in the hierarchy
  - void setChildren(…) – register the caches below it in the hierarchy
  - uint64_t invalidate(const InvReq& req) – invalidate line locally & in children
  - uint64_t access(MemReq& req)

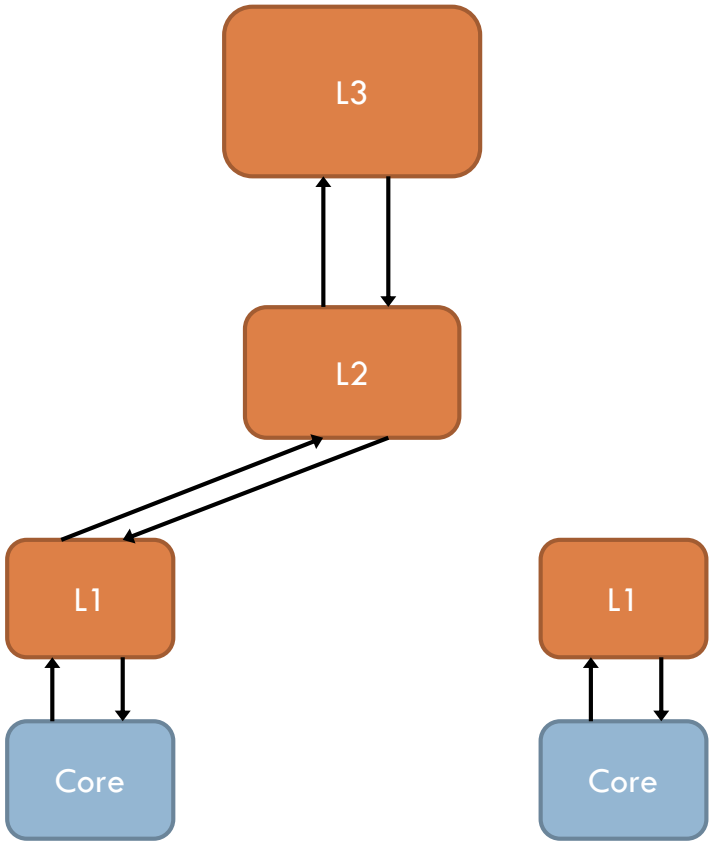
How ZSim allows concurrency



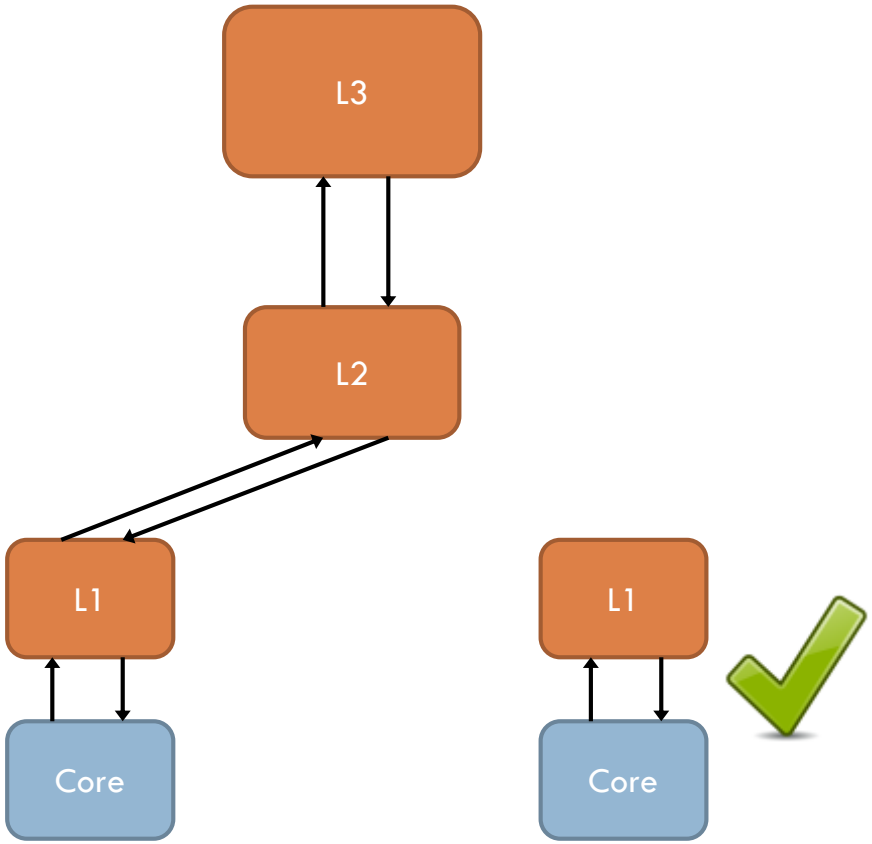
How ZSim allows concurrency



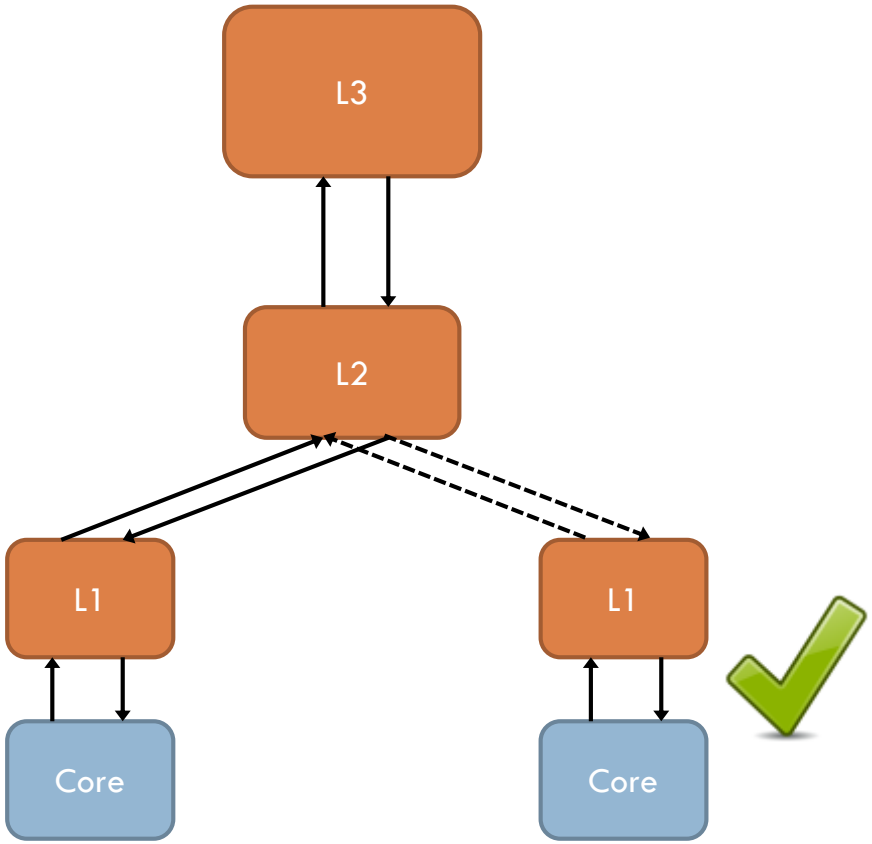
How ZSim allows concurrency



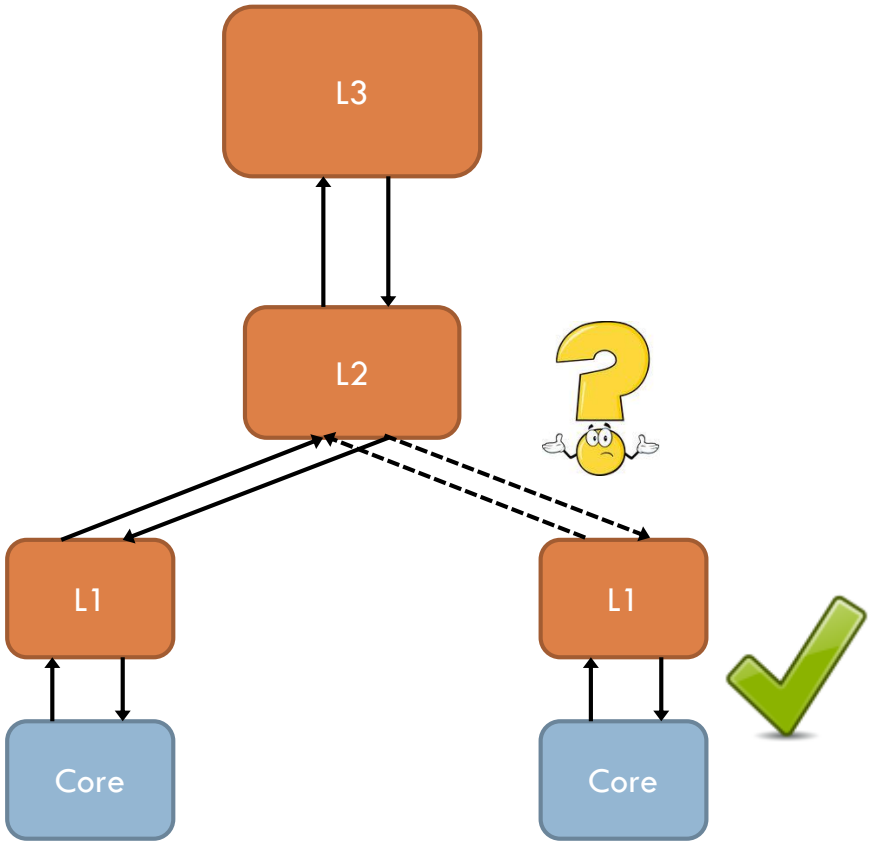
How ZSim allows concurrency



How ZSim allows concurrency

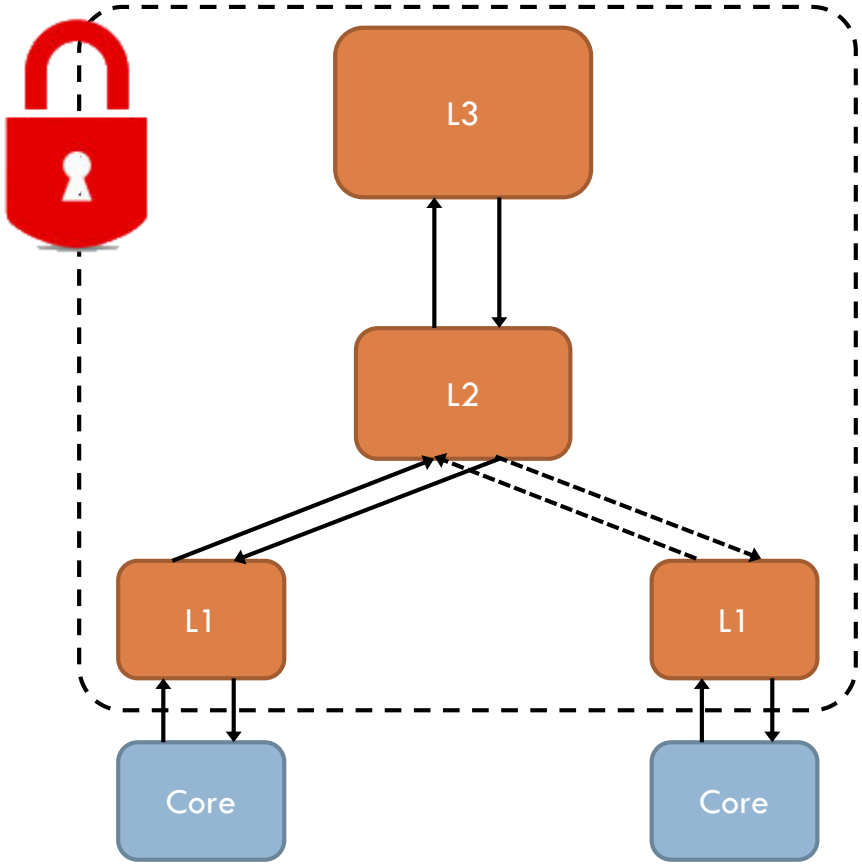


How ZSim allows concurrency



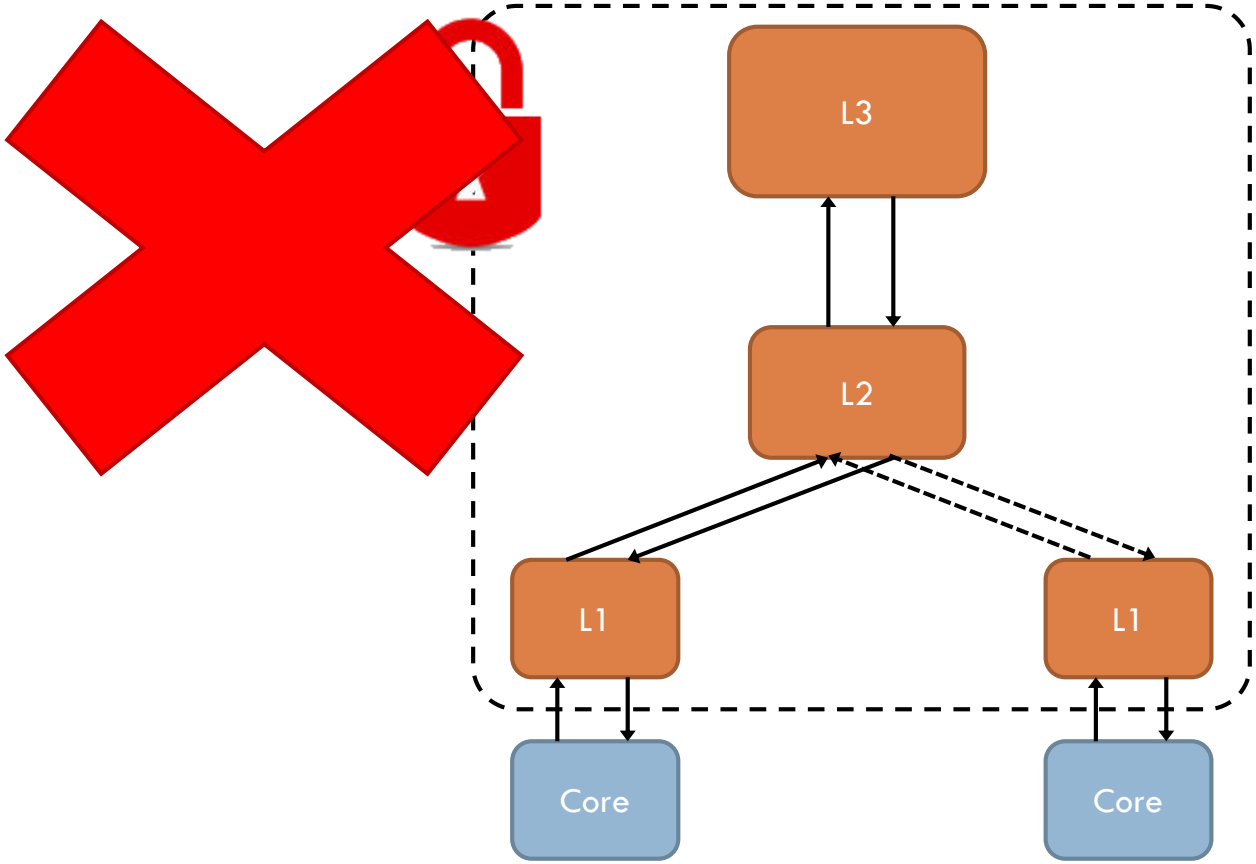
How ZSim allows concurrency

- Naïve “big lock” implementation won’t work



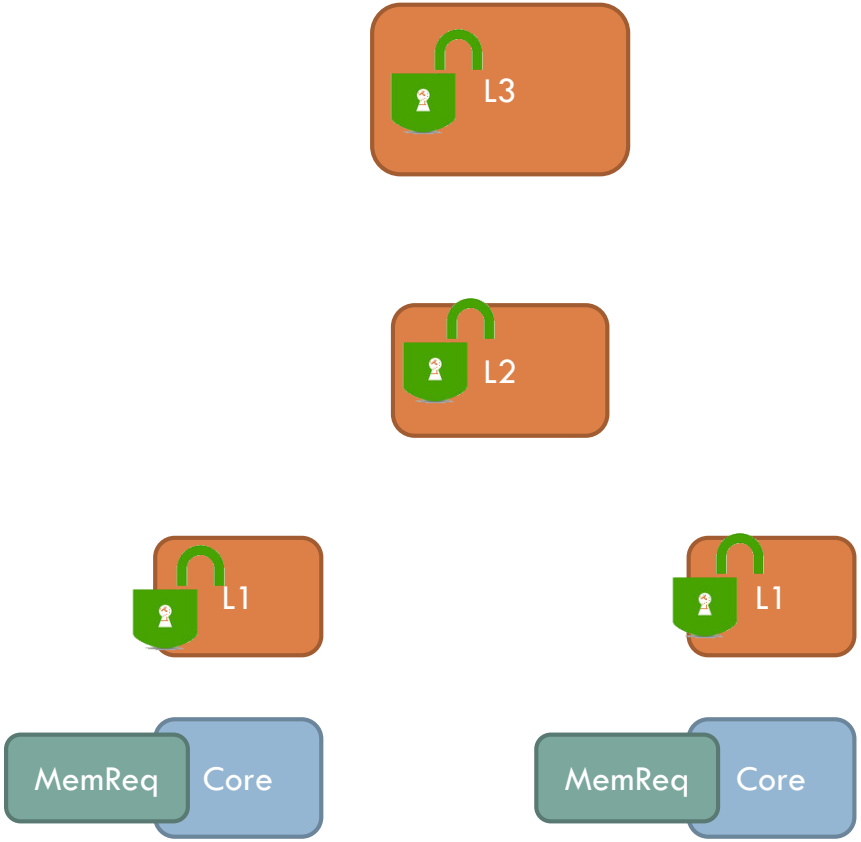
How ZSim allows concurrency

- Naïve “big lock” implementation won’t work



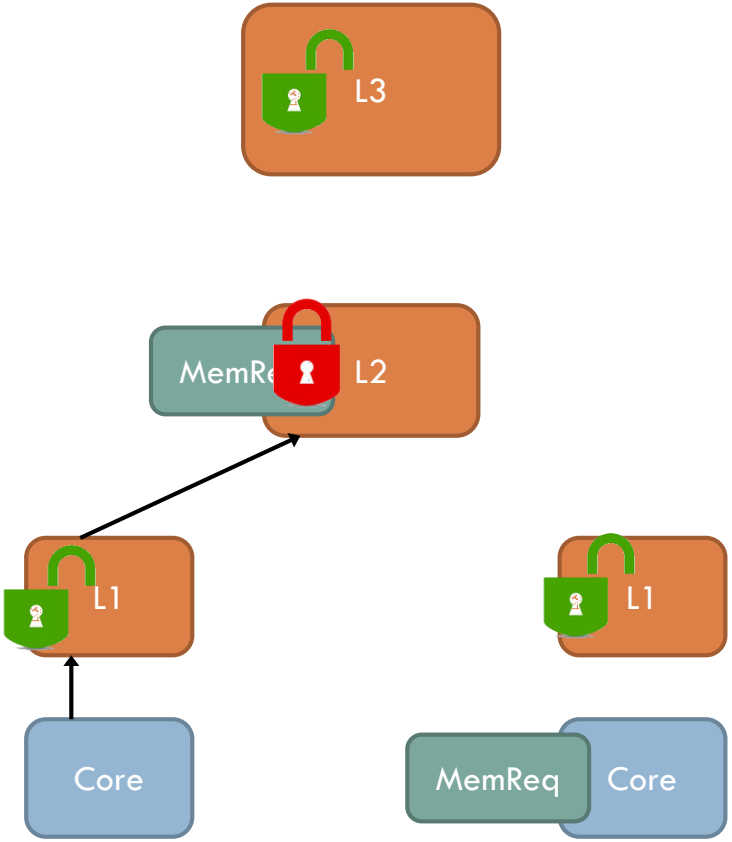
How ZSim allows concurrency

- There is concurrency available!



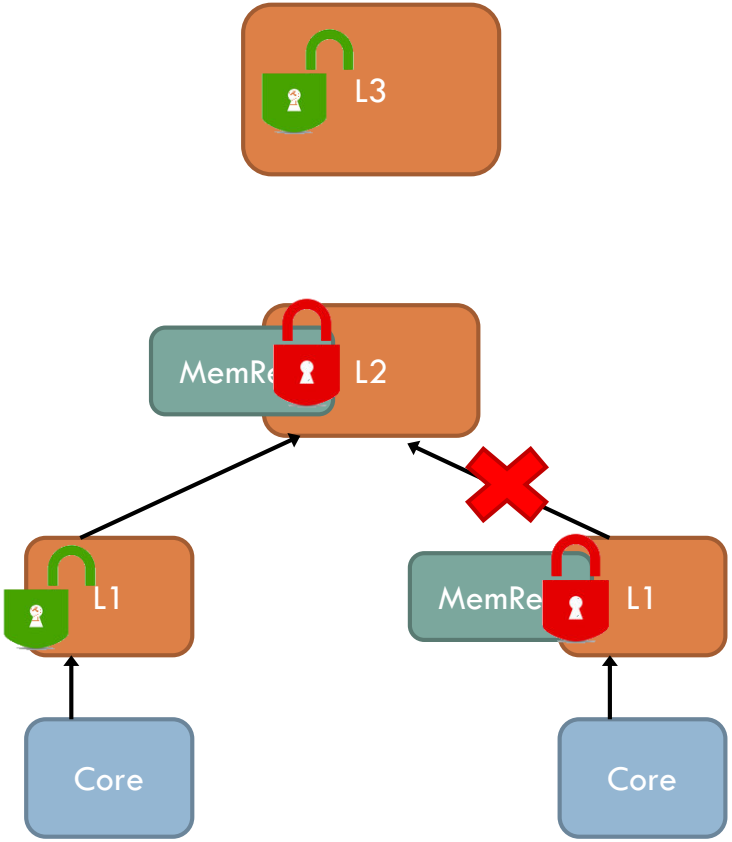
How ZSim allows concurrency

- There is concurrency available!



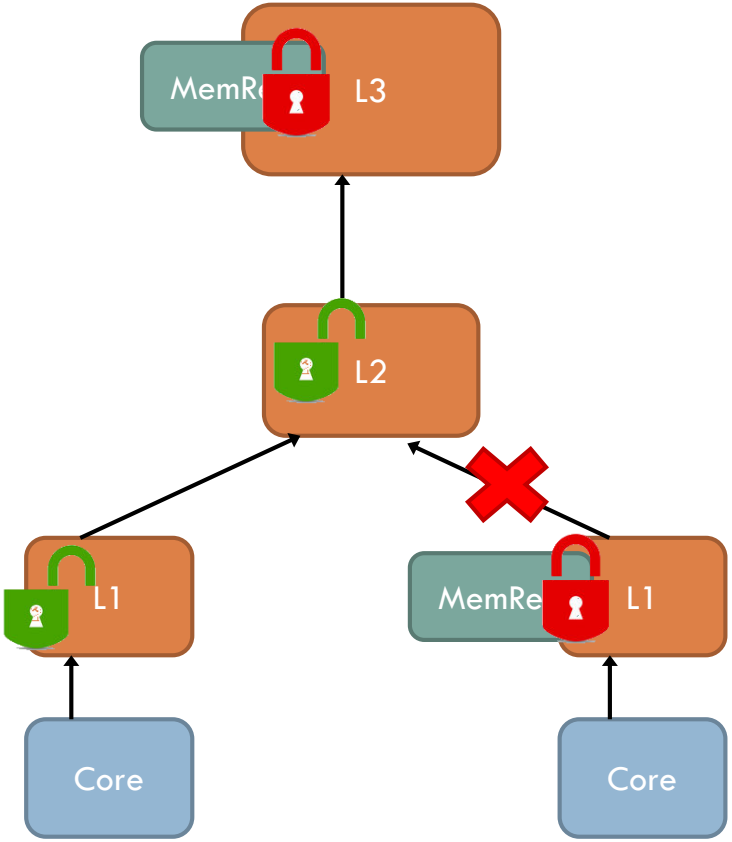
How ZSim allows concurrency

- There is concurrency available!



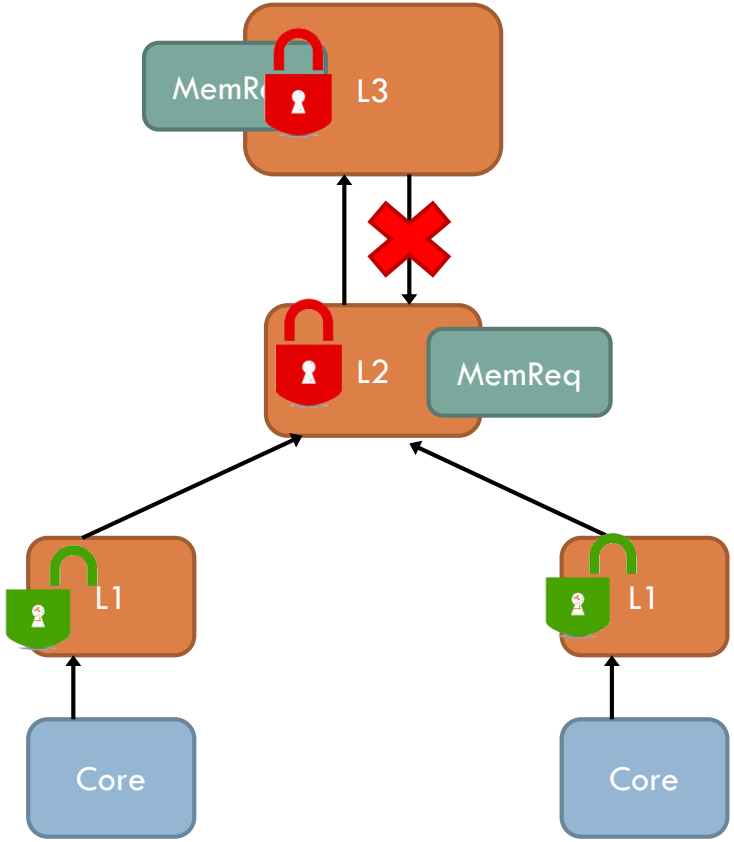
How ZSim allows concurrency

- There is concurrency available!



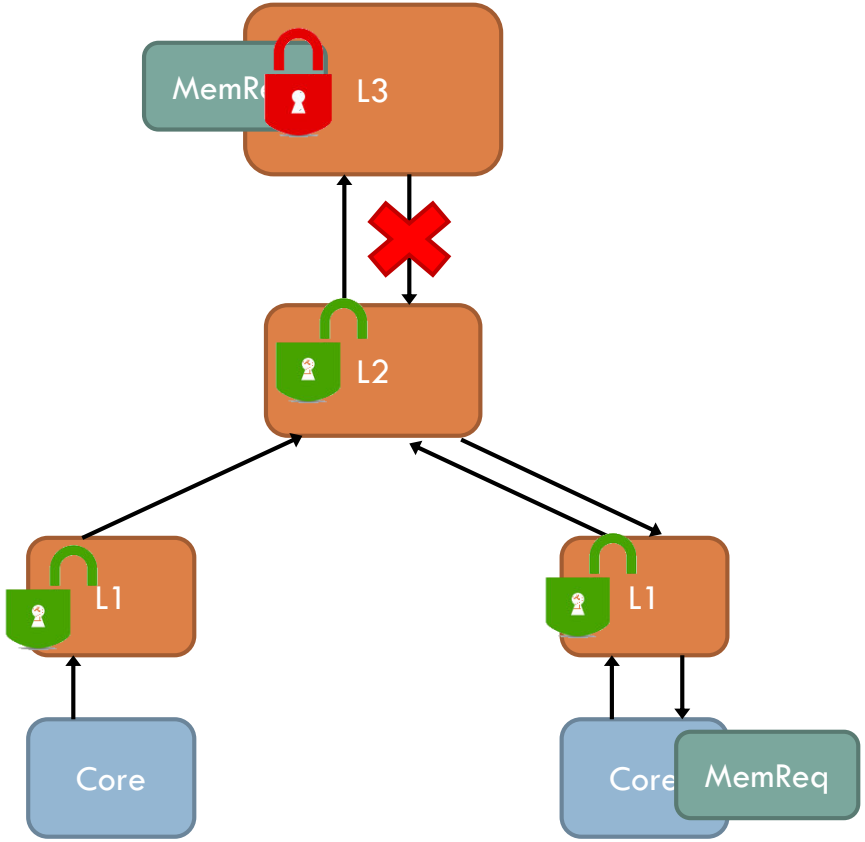
How ZSim allows concurrency

- There is concurrency available!



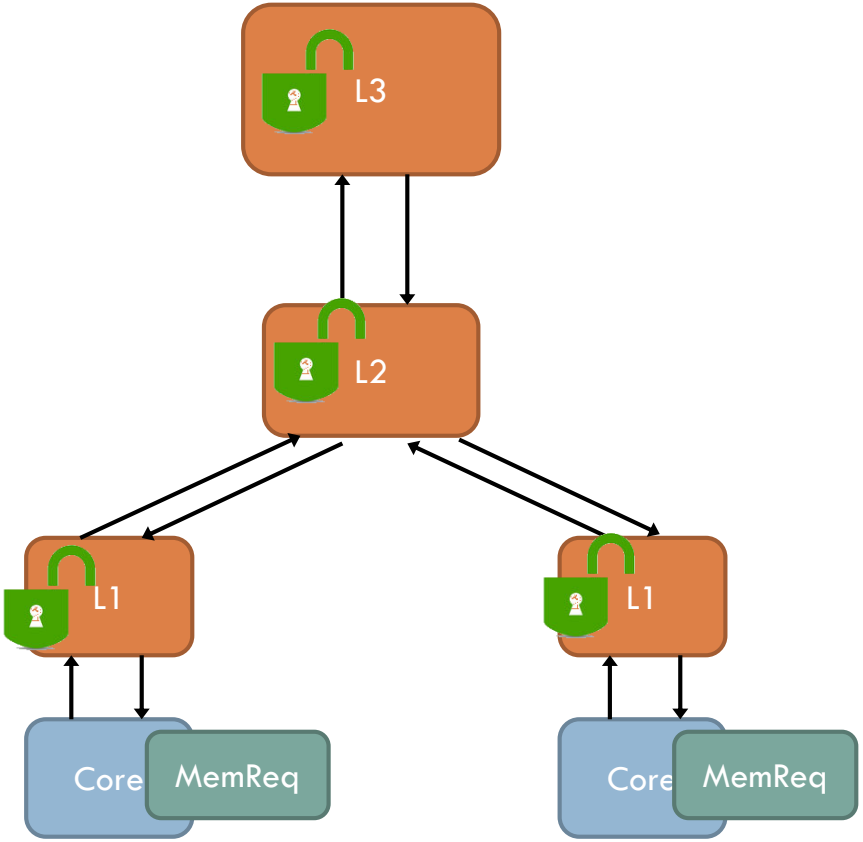
How ZSim allows concurrency

- There is concurrency available!



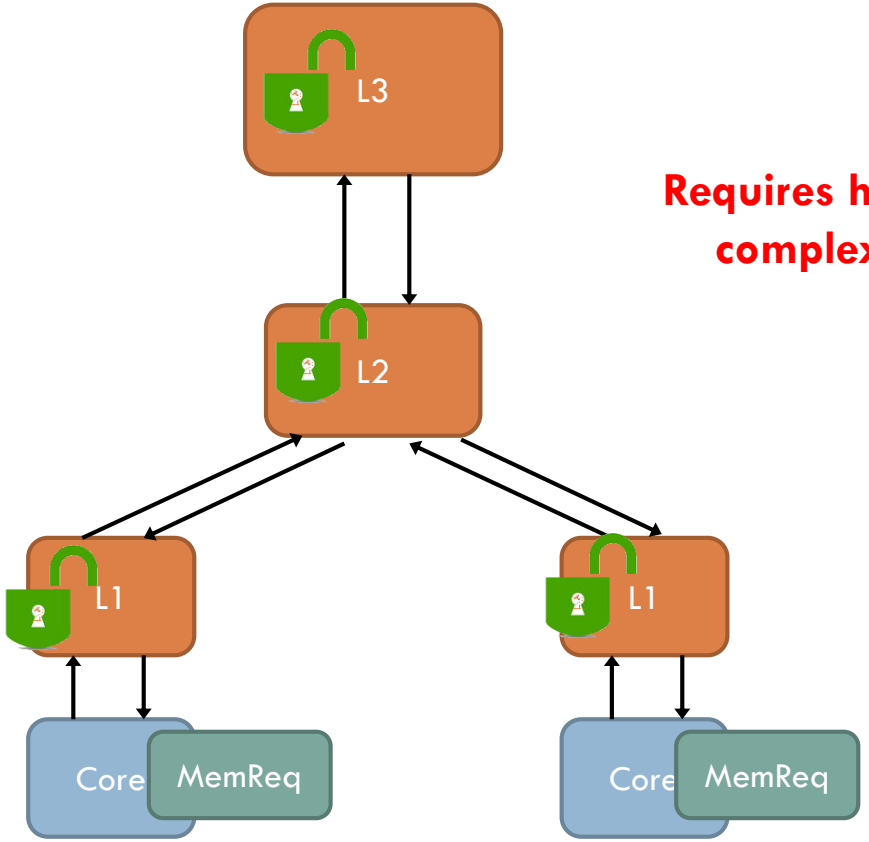
How ZSim allows concurrency

- There is concurrency available!



How ZSim allows concurrency

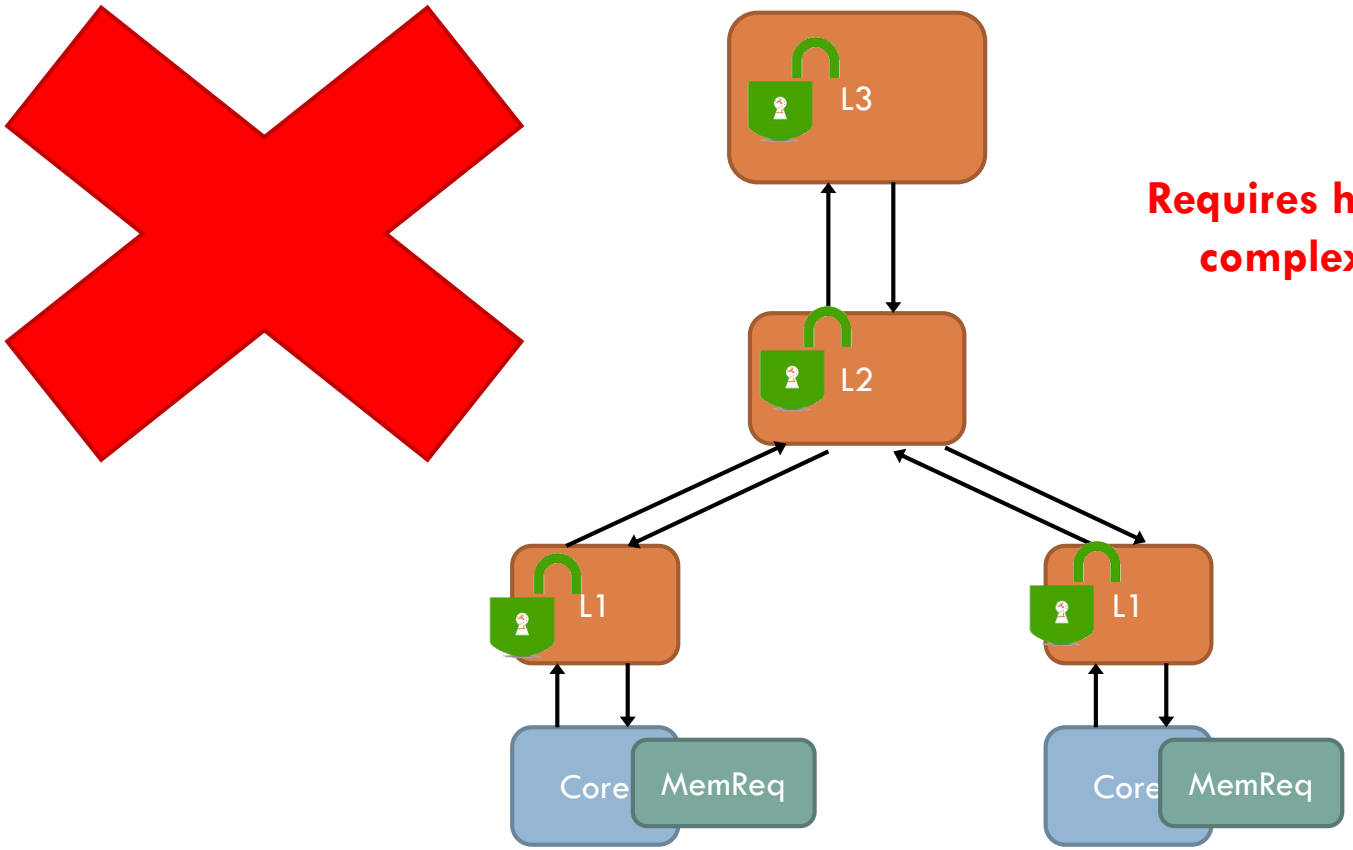
- There is concurrency available!



Requires handling many complex transients!

How ZSim allows concurrency

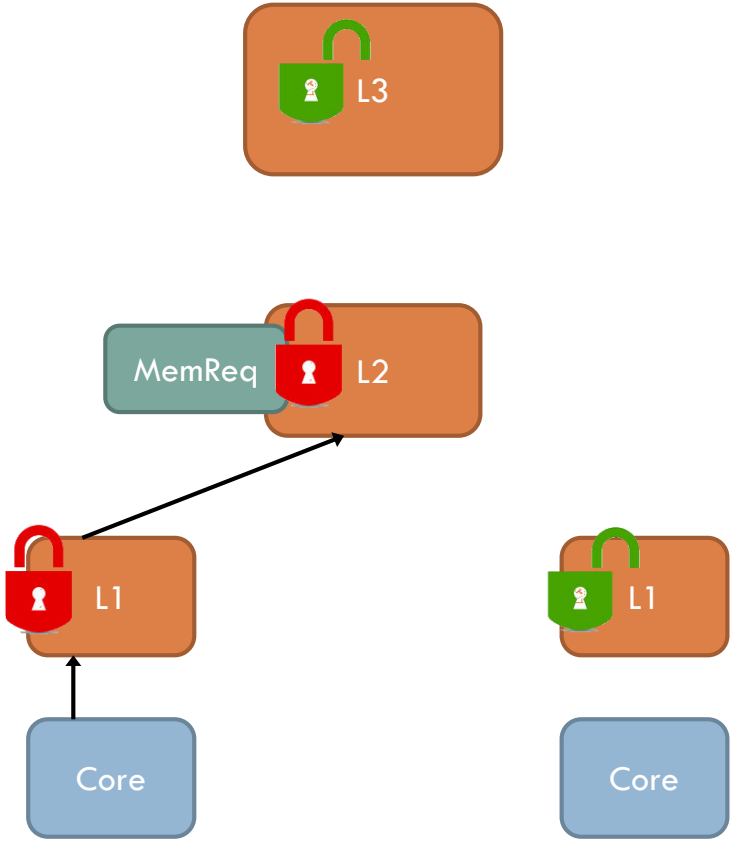
- There is concurrency available!



Requires handling many complex transients!

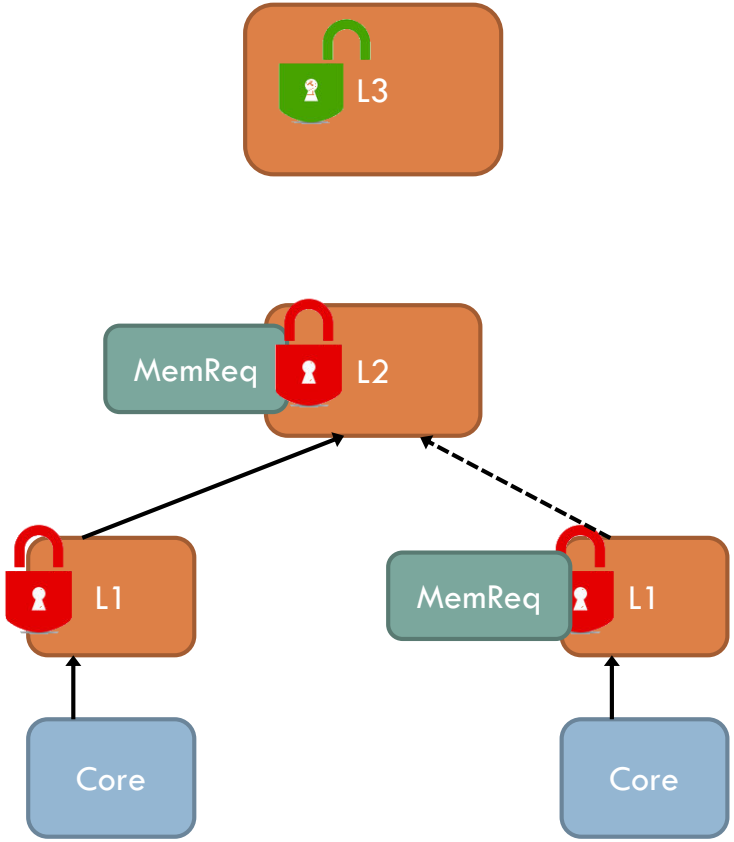
How ZSim allows concurrency

- Locking each cache leads to deadlock on invalidations



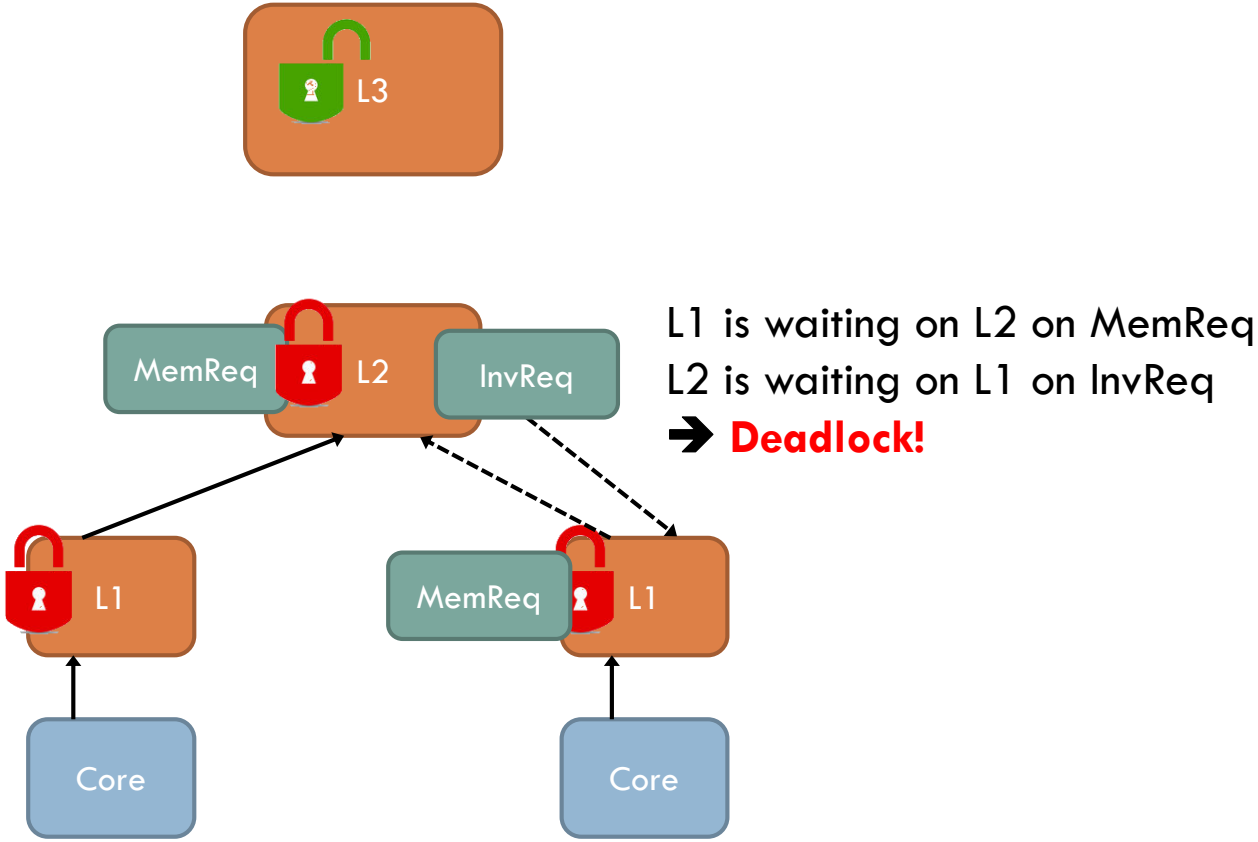
How ZSim allows concurrency

- Locking each cache leads to deadlock on invalidations



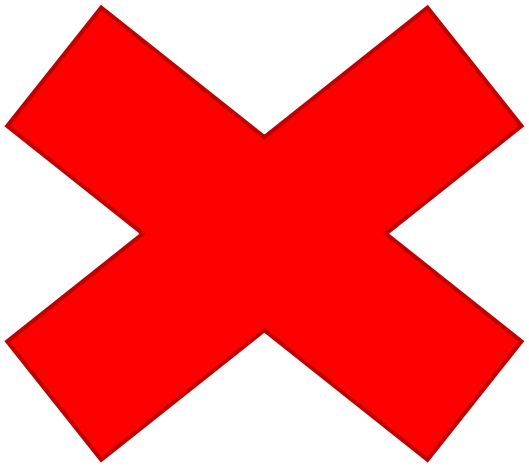
How ZSim allows concurrency

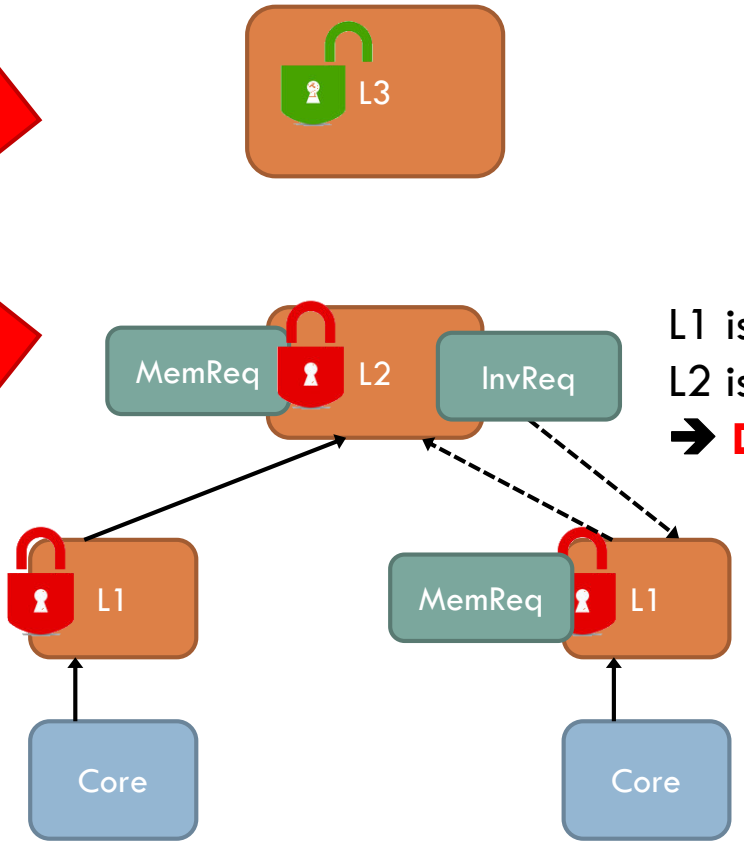
- Locking each cache leads to deadlock on invalidations



How ZSim allows concurrency

- Locking each cache leads to deadlock on invalidations





L1 is waiting on L2 on MemReq
L2 is waiting on L1 on InvReq
➔ **Deadlock!**

How ZSim allows concurrency

- Blocks more accesses going up, allows invalidations going down
- Caches have two locks: access lock + invalidation lock
- Invalidations are prioritized
  - Accesses acquire both locks
  - Invalidations need only invalidation lock

How ZSim allows concurrency

- Blocks more accesses going up, allows invalidations going down
- Caches have two locks: access lock + invalidation lock
- Invalidations are prioritized
  - Accesses acquire both locks
  - Invalidations need only invalidation lock

```
uint64_t Cache::access(MemReq& req) {
    invLock.acquire(); accLock.acquire();
    // look up address etc
    invLock.release()
    parent->access(req);
    // check if we got an invalidation!
    accLock.release();
    return completionTime;
}
```

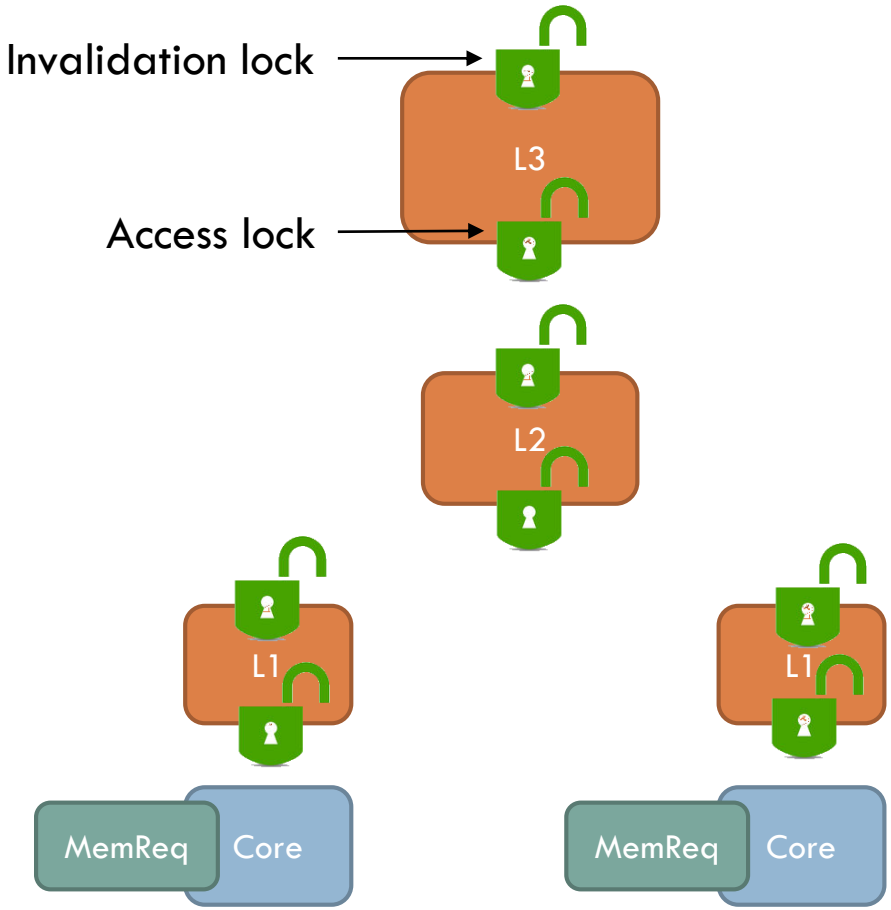
How ZSim allows concurrency

- Blocks more accesses going up, allows invalidations going down
- Caches have two locks: access lock + invalidation lock
- Invalidations are prioritized
  - Accesses acquire both locks
  - Invalidations need only invalidation lock

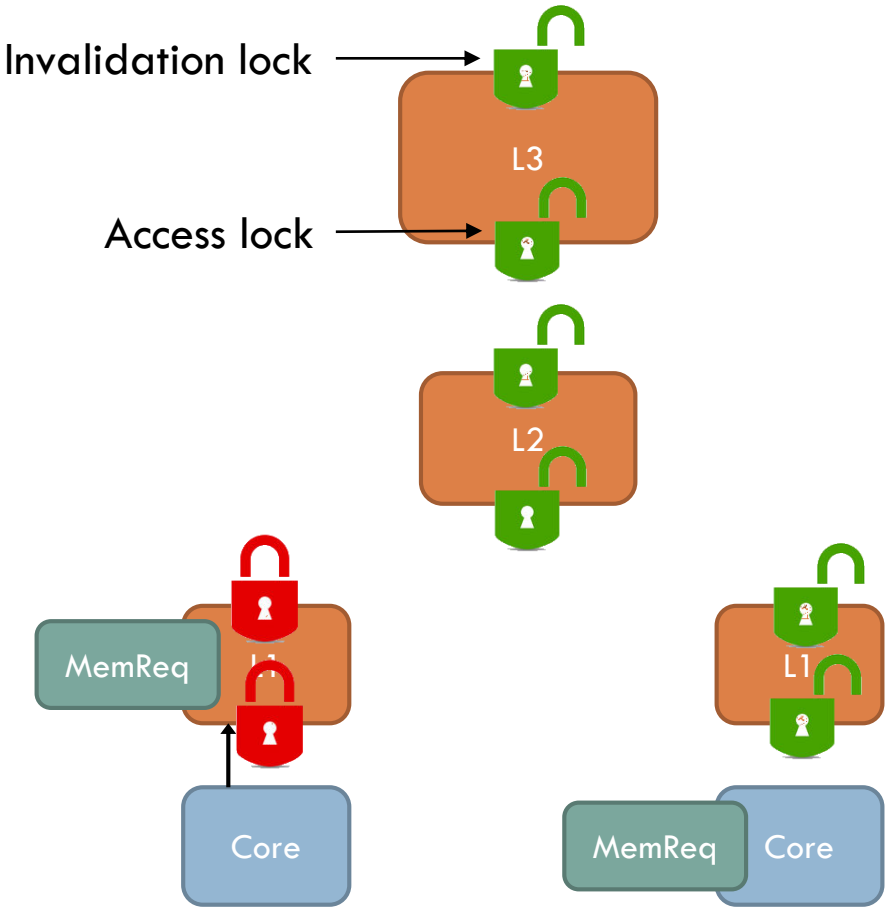
```
uint64_t Cache::access(MemReq& req) {
    invLock.acquire(); accLock.acquire();
    // look up address etc
    invLock.release()
    parent->access(req);
    // check if we got an invalidation!
    accLock.release();
    return completionTime;
}
```

```
uint64_t Cache::invalidate(InvReq& req) {
    invLock.acquire();
    // do invalidation
    children.invalidate(req);
    invLock.release()
    return completionTime;
}
```

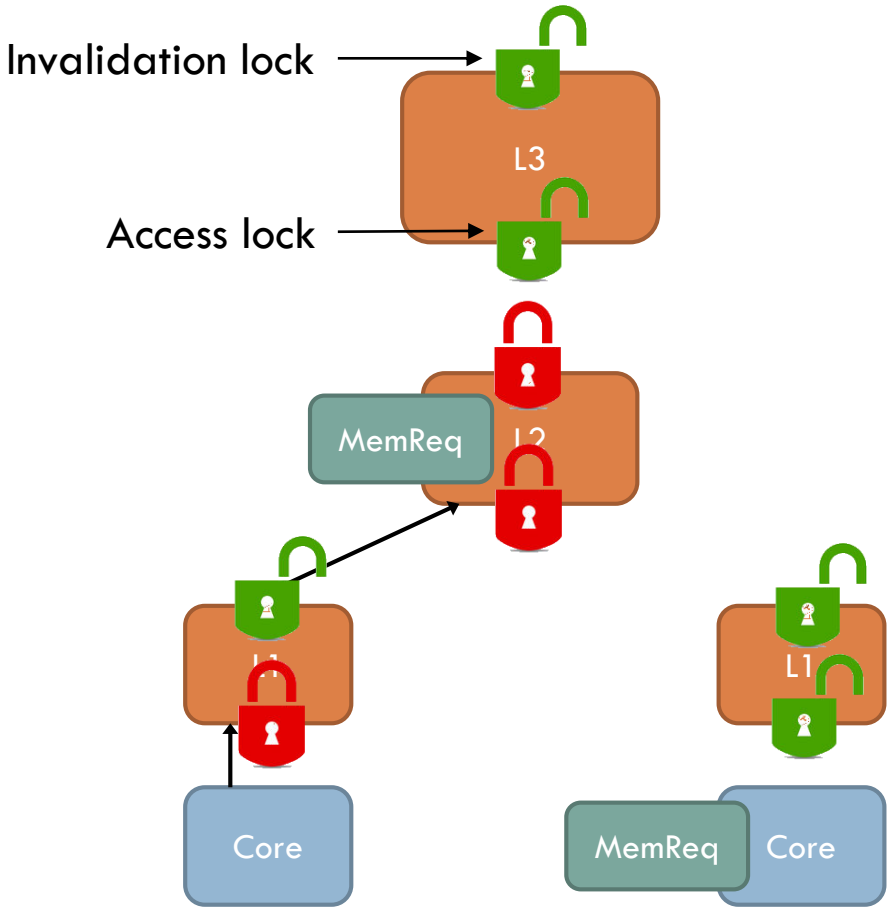
How ZSim allows concurrency



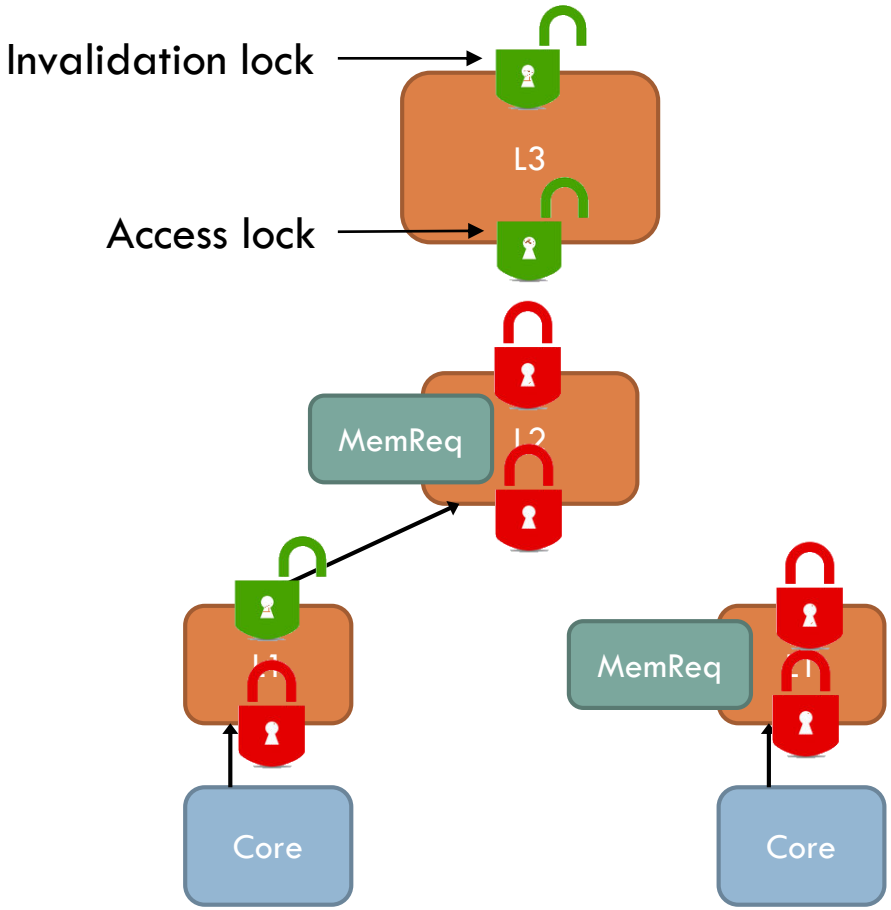
How ZSim allows concurrency



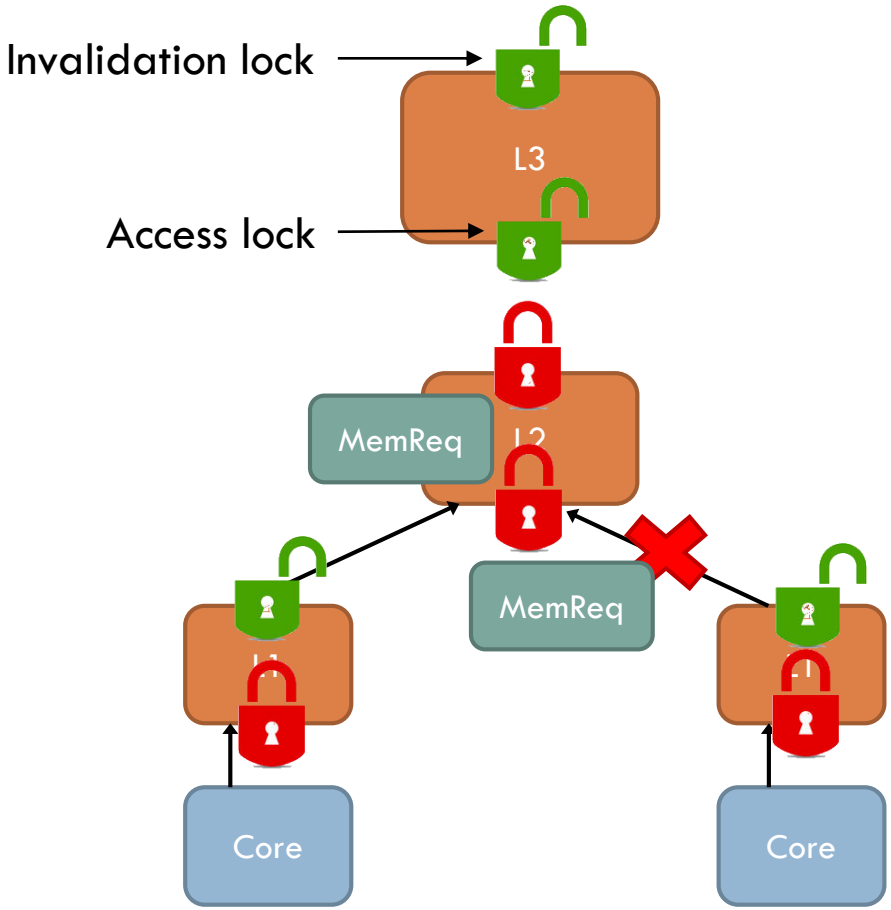
How ZSim allows concurrency



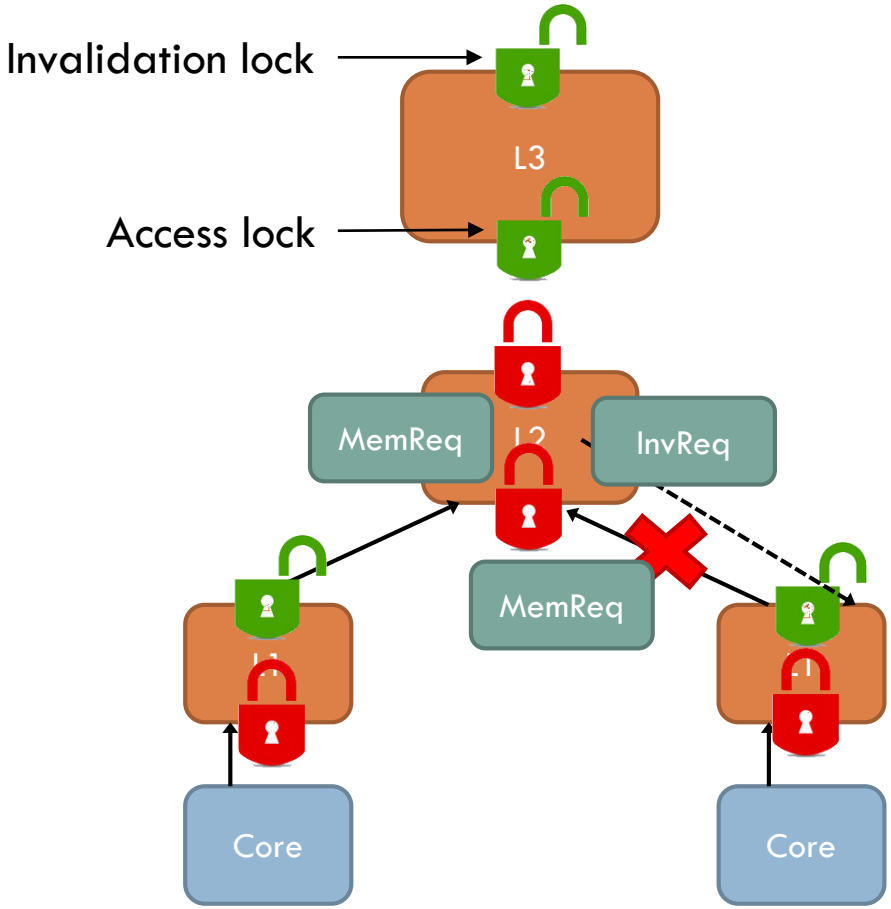
How ZSim allows concurrency



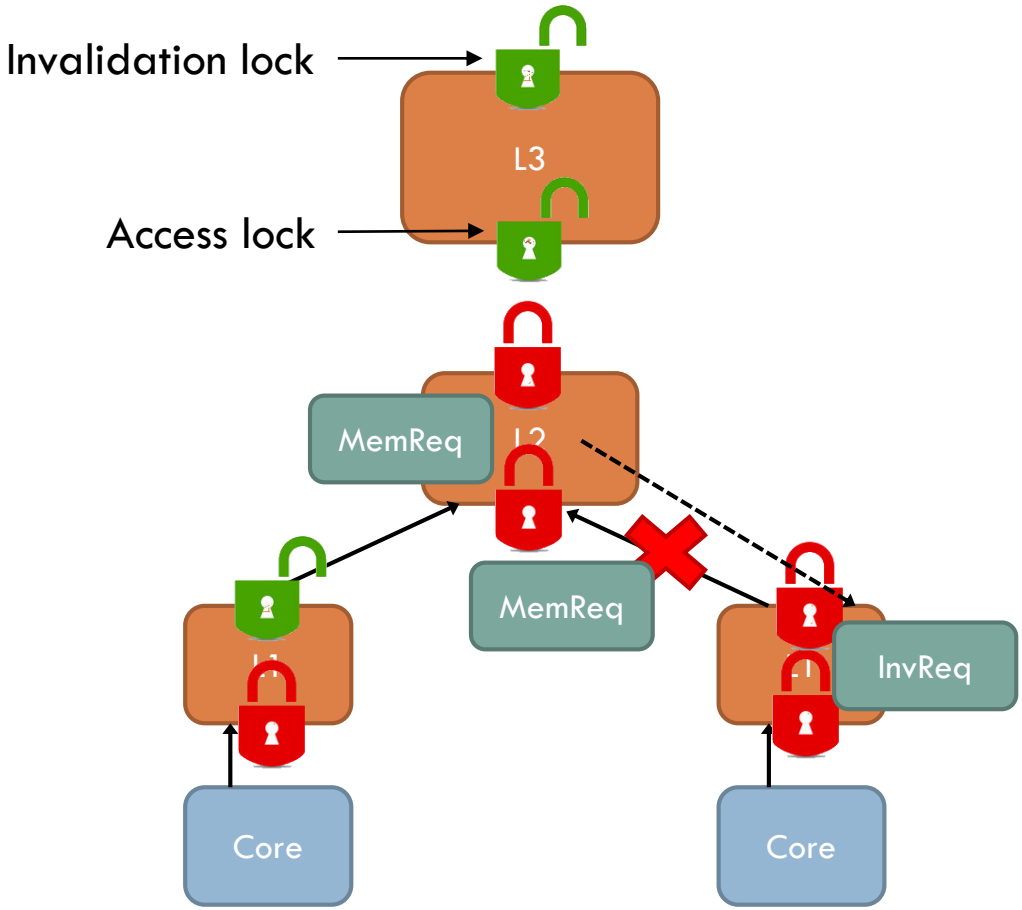
How ZSim allows concurrency



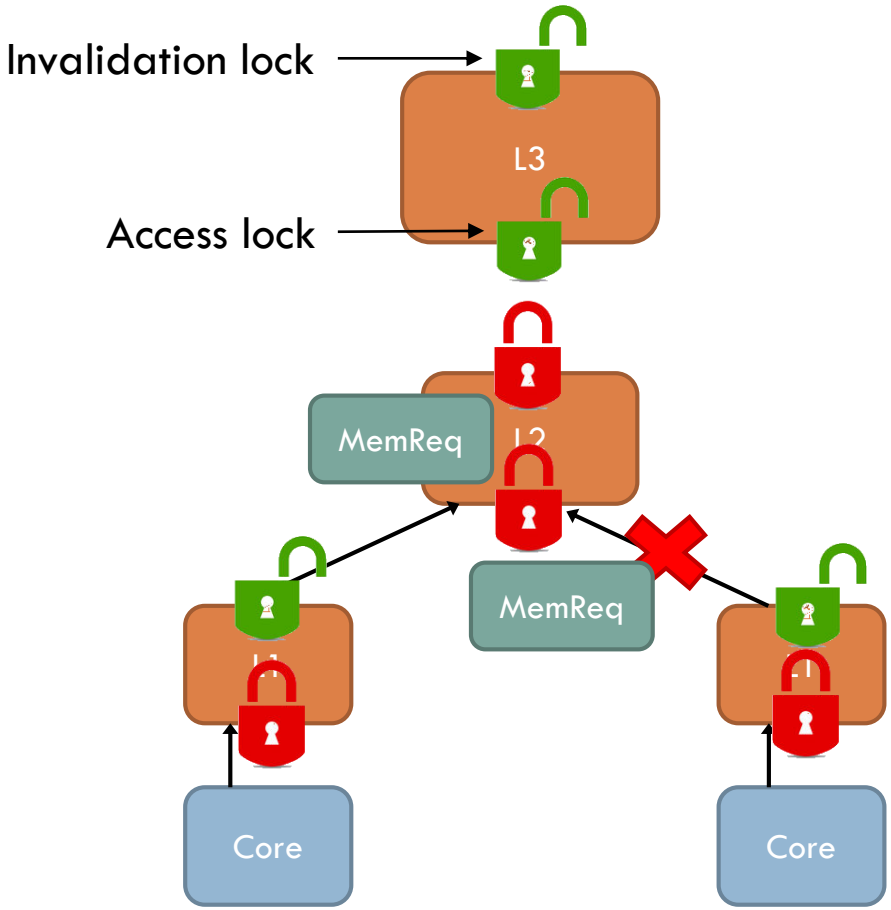
How ZSim allows concurrency



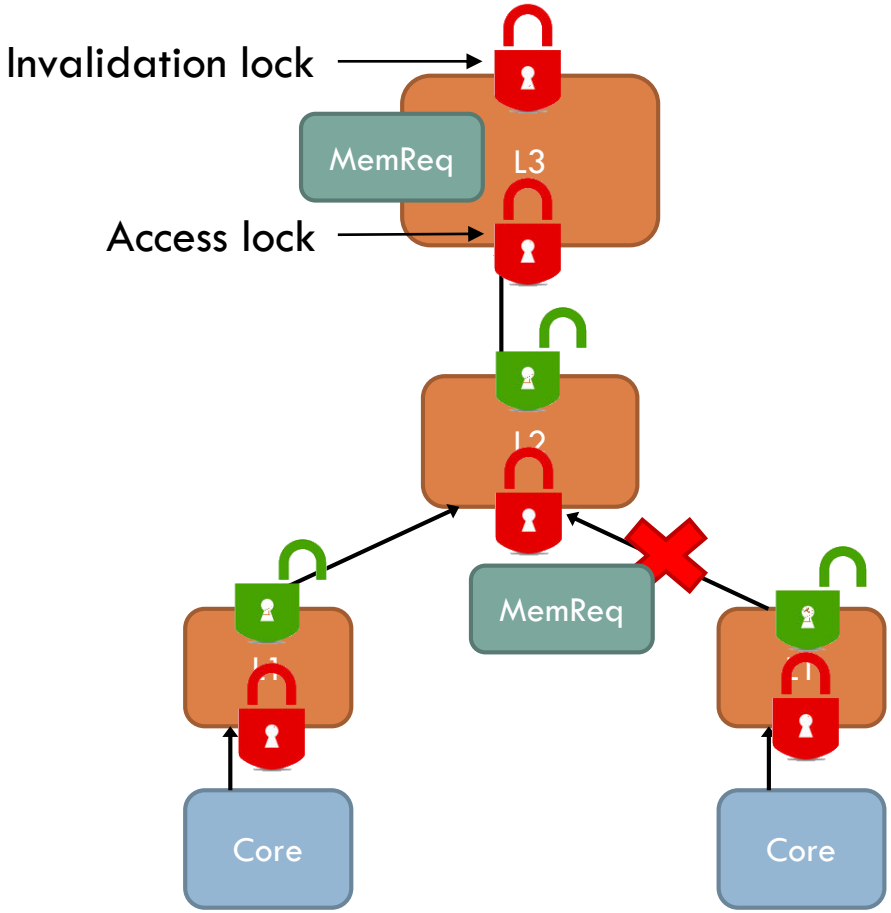
How ZSim allows concurrency



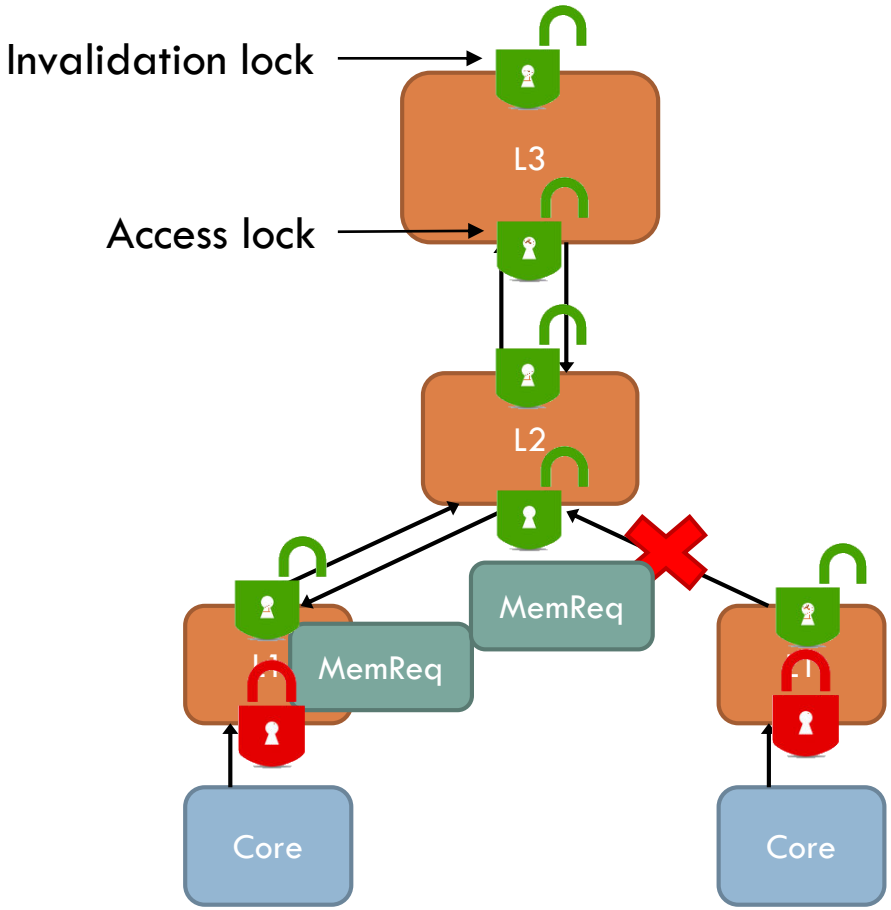
How ZSim allows concurrency



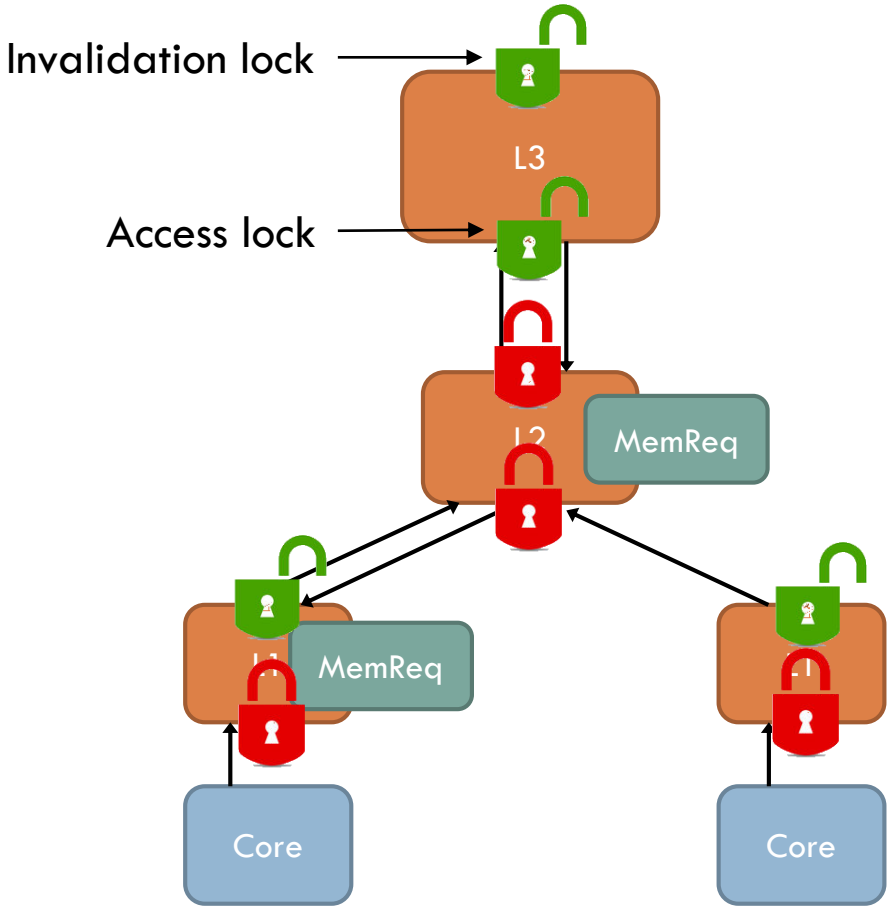
How ZSim allows concurrency



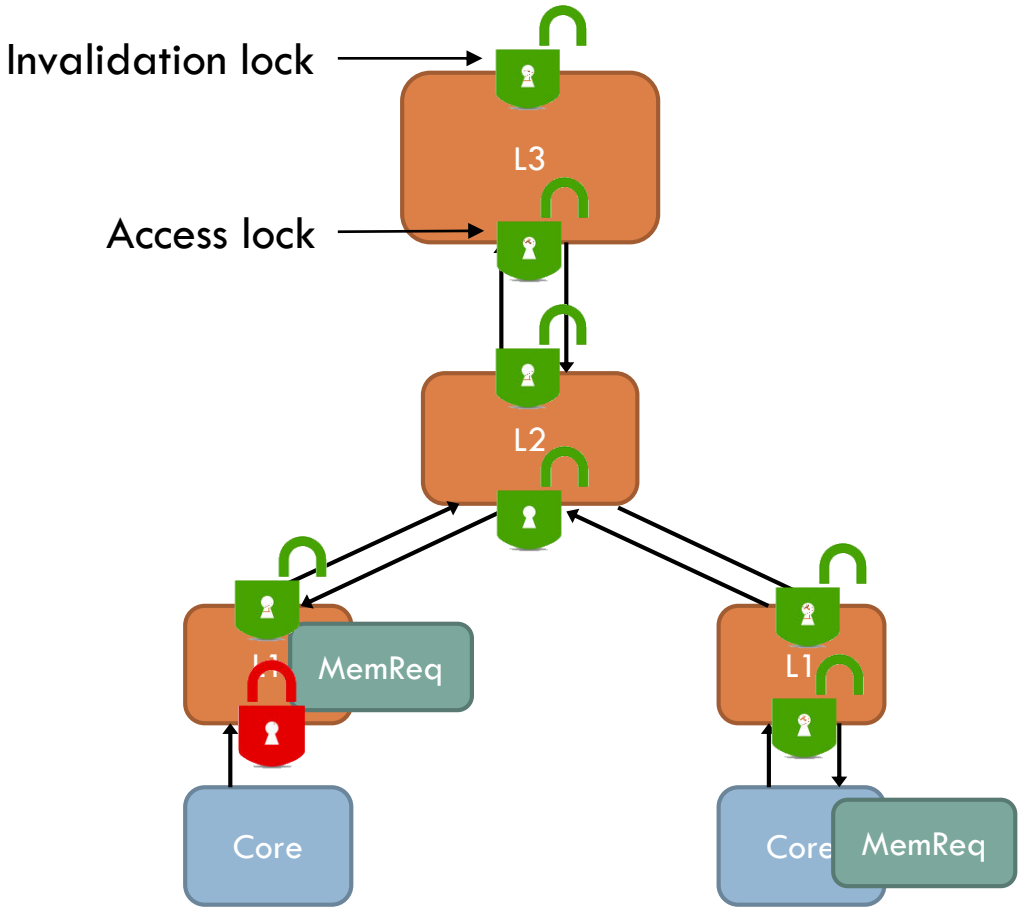
How ZSim allows concurrency



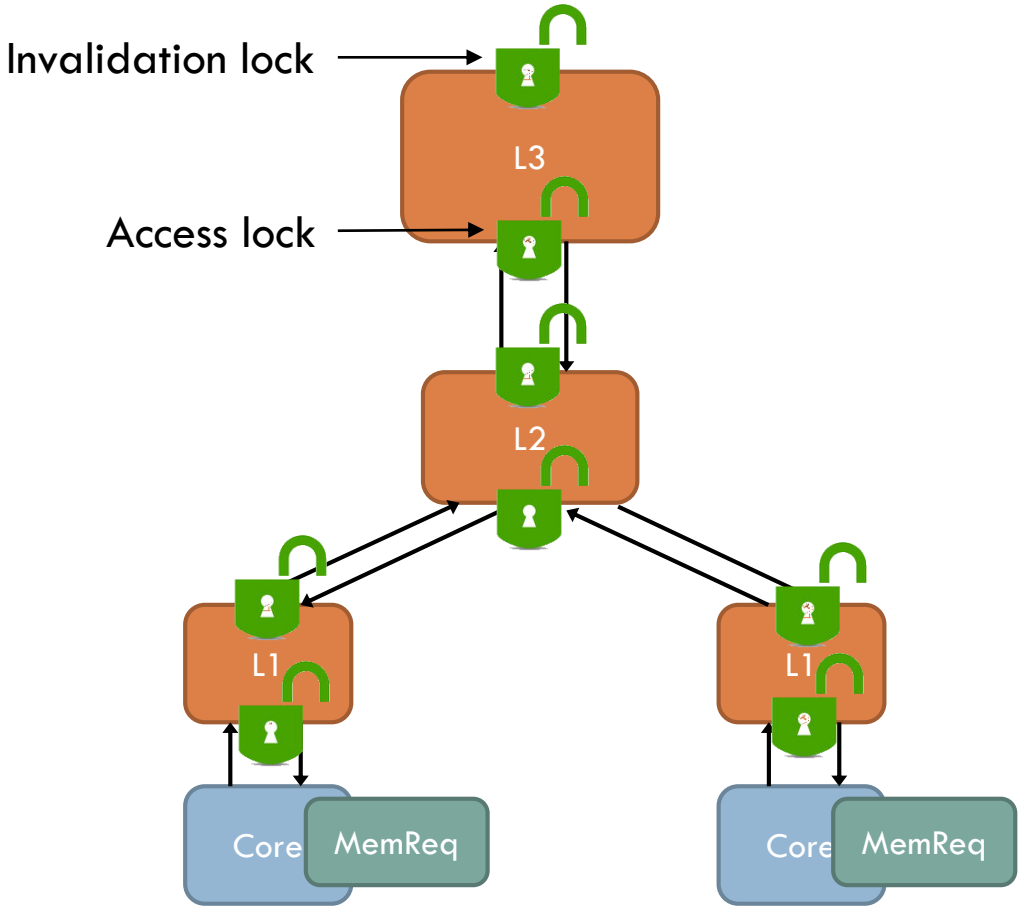
How ZSim allows concurrency



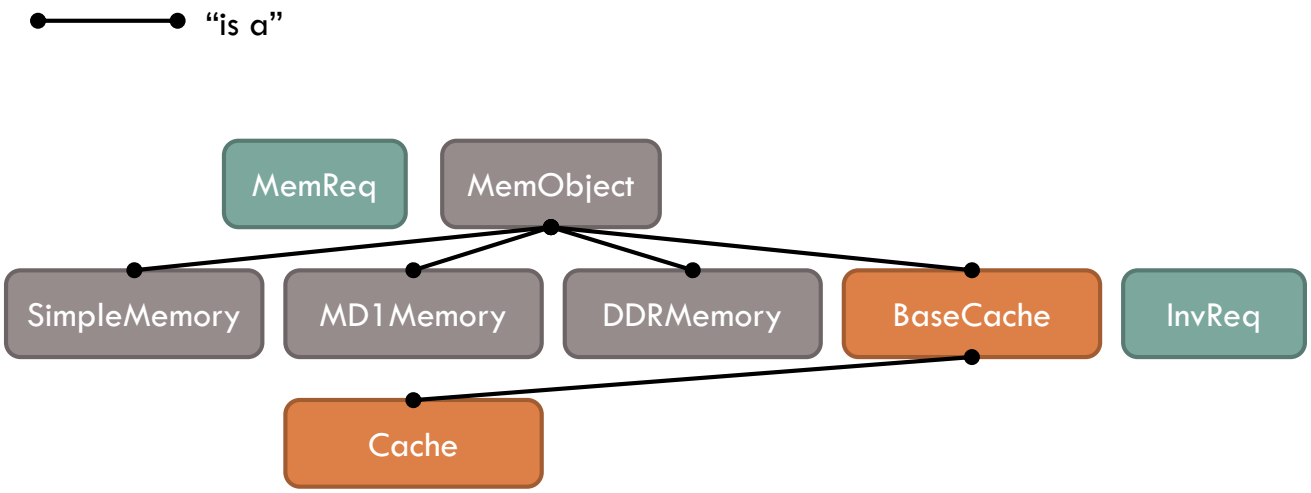
How ZSim allows concurrency



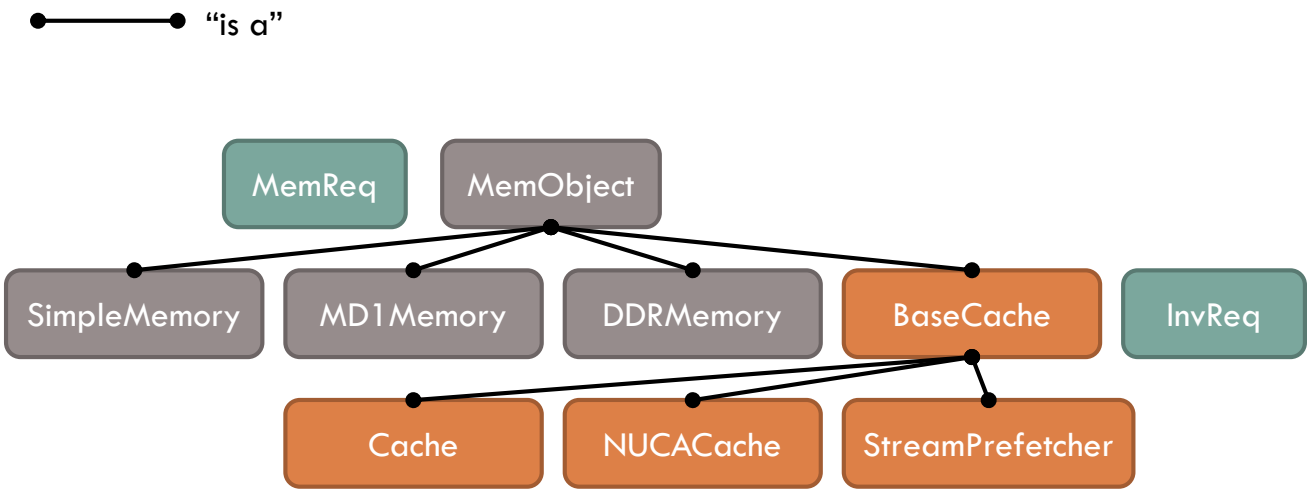
How ZSim allows concurrency



Important ZSim memory classes



Important ZSim memory classes



NUCACache

NUCACache

- Non-uniform cache access: banks distributed around the chip

- Important fields:
  - BankDir* bankDir – see below
  - g_vector<BaseCache*> banks – the distributed banks

- Important methods: none over BaseCache

NUCACache

- Non-uniform cache access: banks distributed around the chip
- Important fields:
  - BankDir* bankDir – see below
  - g_vector<BaseCache*> banks – the distributed banks
- Important methods: none over BaseCache
- Supports dynamic NUCA policies via **BankDir** class
  - uint32_t preAccess(MemReq& req) – Give destination bank
  - int32_t getPrevBank(MemReq& req, uint32_t curBank) – Get old bank (if moved)

NUCACache

- Non-uniform cache access: banks distributed around the chip
- Important fields:
  - BankDir* bankDir – see below
  - g_vector<BaseCache*> banks – the distributed banks
- Important methods: none over BaseCache
- Supports dynamic NUCA policies via **BankDir** class
  - uint32_t preAccess(MemReq& req) – Give destination bank
  - int32_t getPrevBank(MemReq& req, uint32_t curBank) – Get old bank (if moved)
- Wide-ranging support
  - First-touch, R-NUCA [Hardavellas ISCA'09], [Awasthi HPCA'09], idealized private D-NUCA [Herrero ISCA'10], Jigsaw [Beckmann PACT'13, Beckmann HPCA'15]
  - Some yet-to-be-released

NUCACache::access pseudo-code


```
uint64_t NUCACache::access(MemReq& req) {
    uint32_t bank = bankDir->preAccess(req);
    int32_t prevBank = bankDir->getPrevBank(req, bank);

    if (prevBank != -1 && bank != prevBank) {
        // move the line from prevBank to bank
    }

    uint64_t completionCycle = banks[bank]->access(req);
    return completionCycle;
}
```

Implementing your own D-NUCA

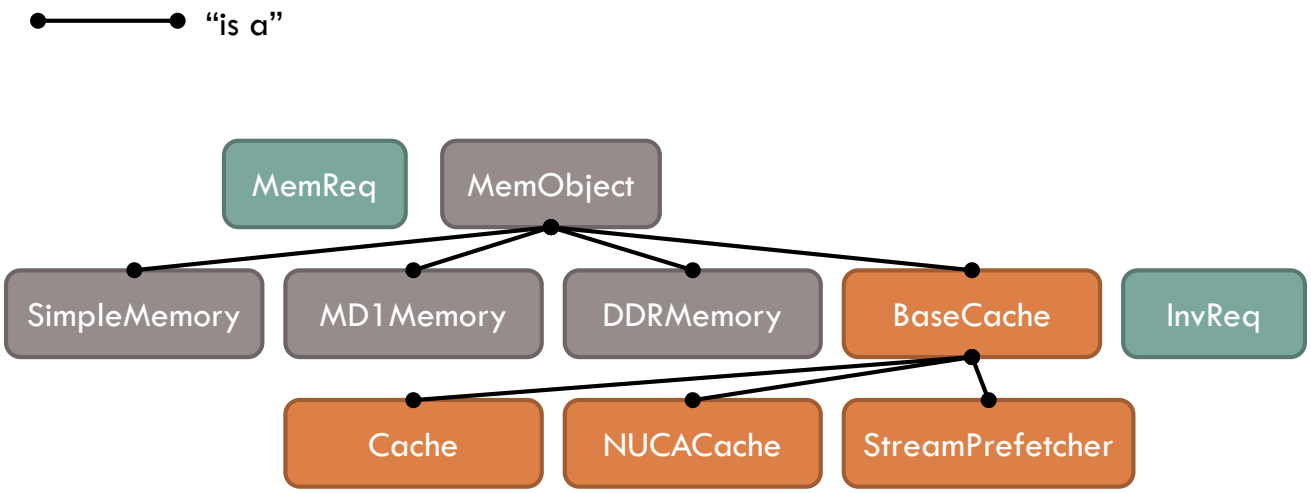
- Idealized "last-touch" bank dir that migrates lines to wherever they are referenced

```
uint32_t LastTouchBankDir::preAccess(MemReq& req) {
    uint32_t closestBank = nuca->getSortedRTTs(req.childId)[0].second;
    return closestBank;
}

int32_t LastTouchBankDir::getPrevBank(MemReq& req, uint32_t currentBank) {
    ScopedMutex sm(mutex); // avoid races
    auto prevBankId = lineMap.find(req.lineAddr);
    if (prevBankId == lineMap.end() || currentBank == *prevBankId) {
        return -1;
    } else {
        uint32_t prevBank = *prevBankId;
        *prevBankId = currentBank;
        return *prevBank;
    }
}
```

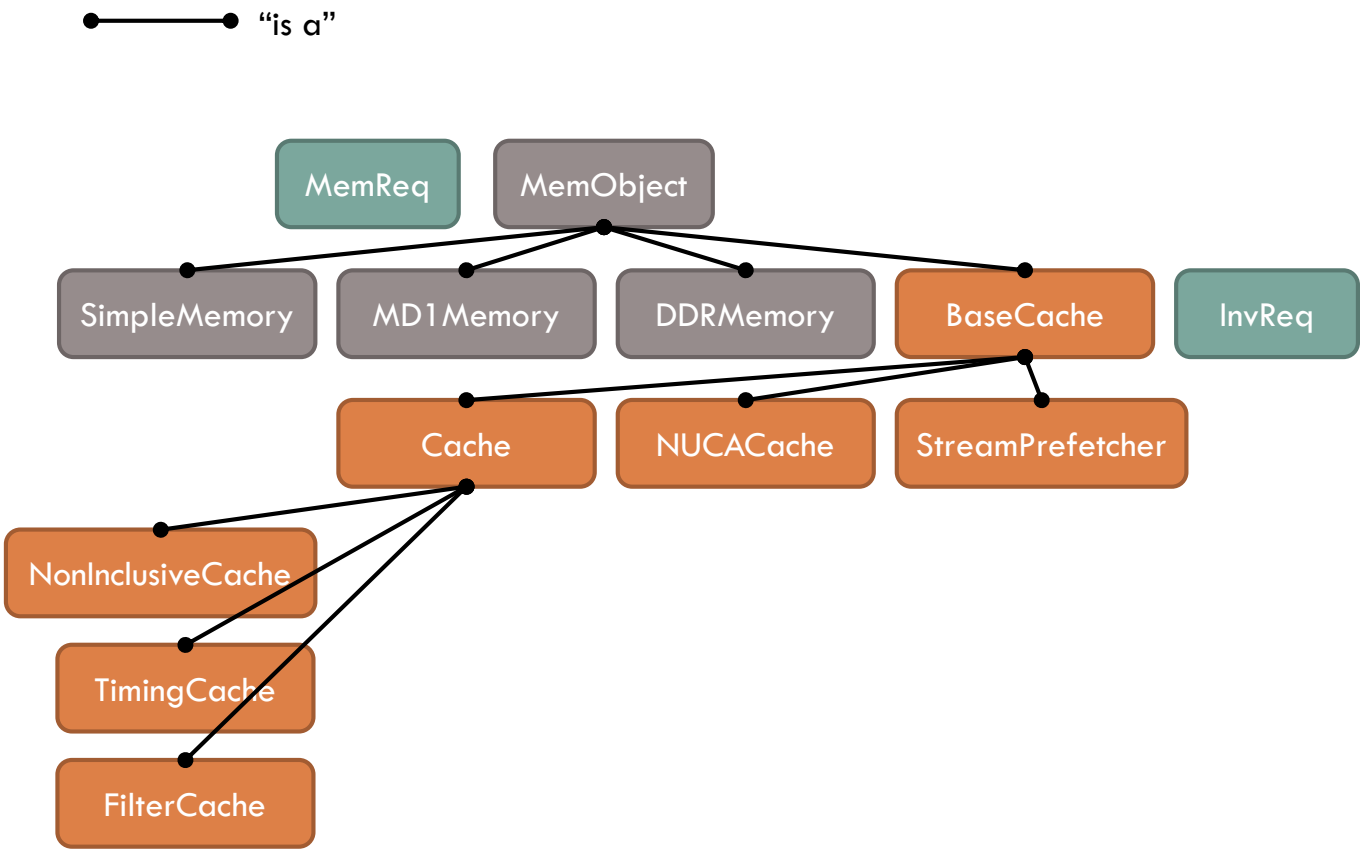
StreamPrefetcher

- Implements stream prefetcher
- Important fields:
  - Entry array[16] – the streams it is following
- Important methods: none over BaseCache
- Prefetcher will issue its own MemReqs to parents
  - Validated against Westmere

Important ZSim memory classes



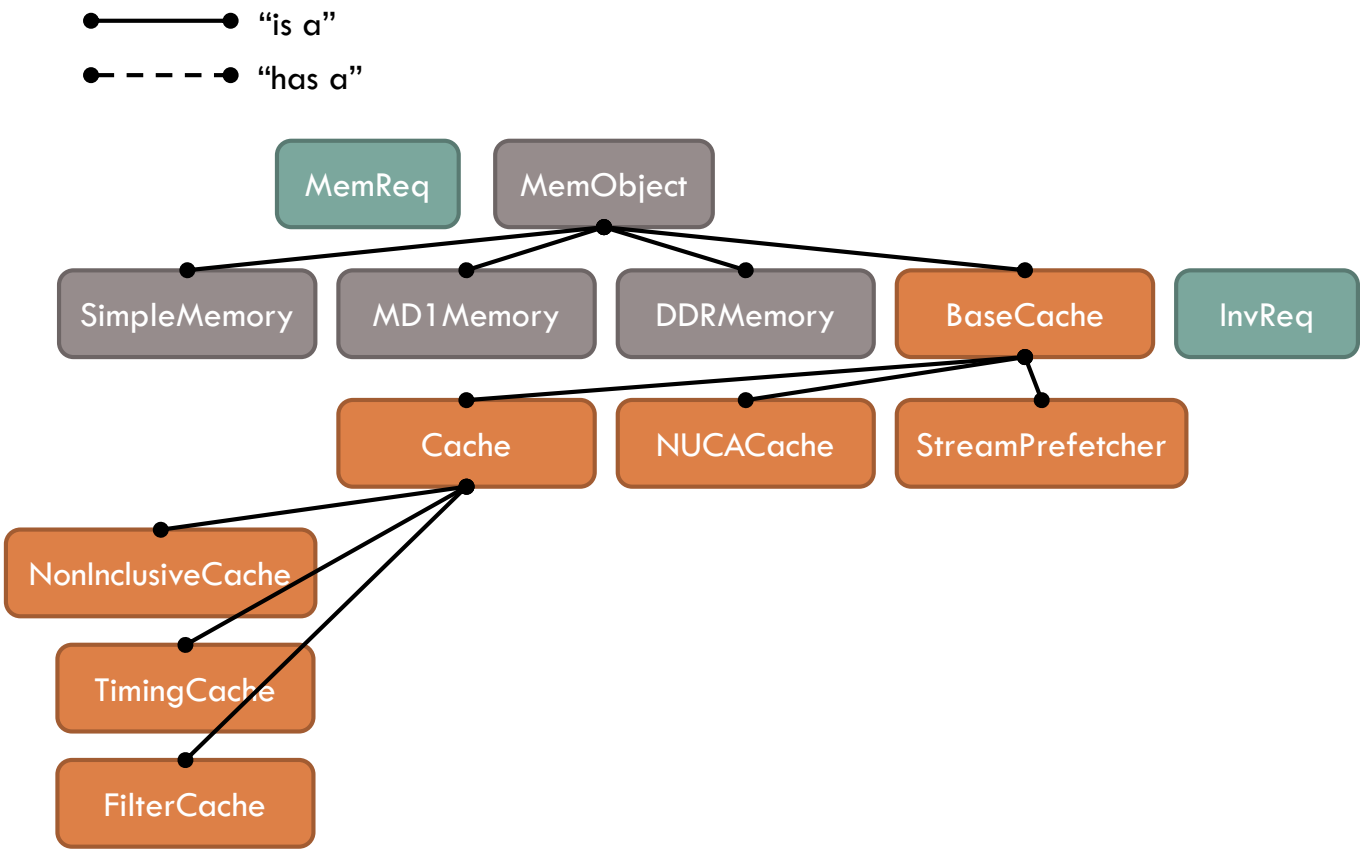
Important ZSim memory classes



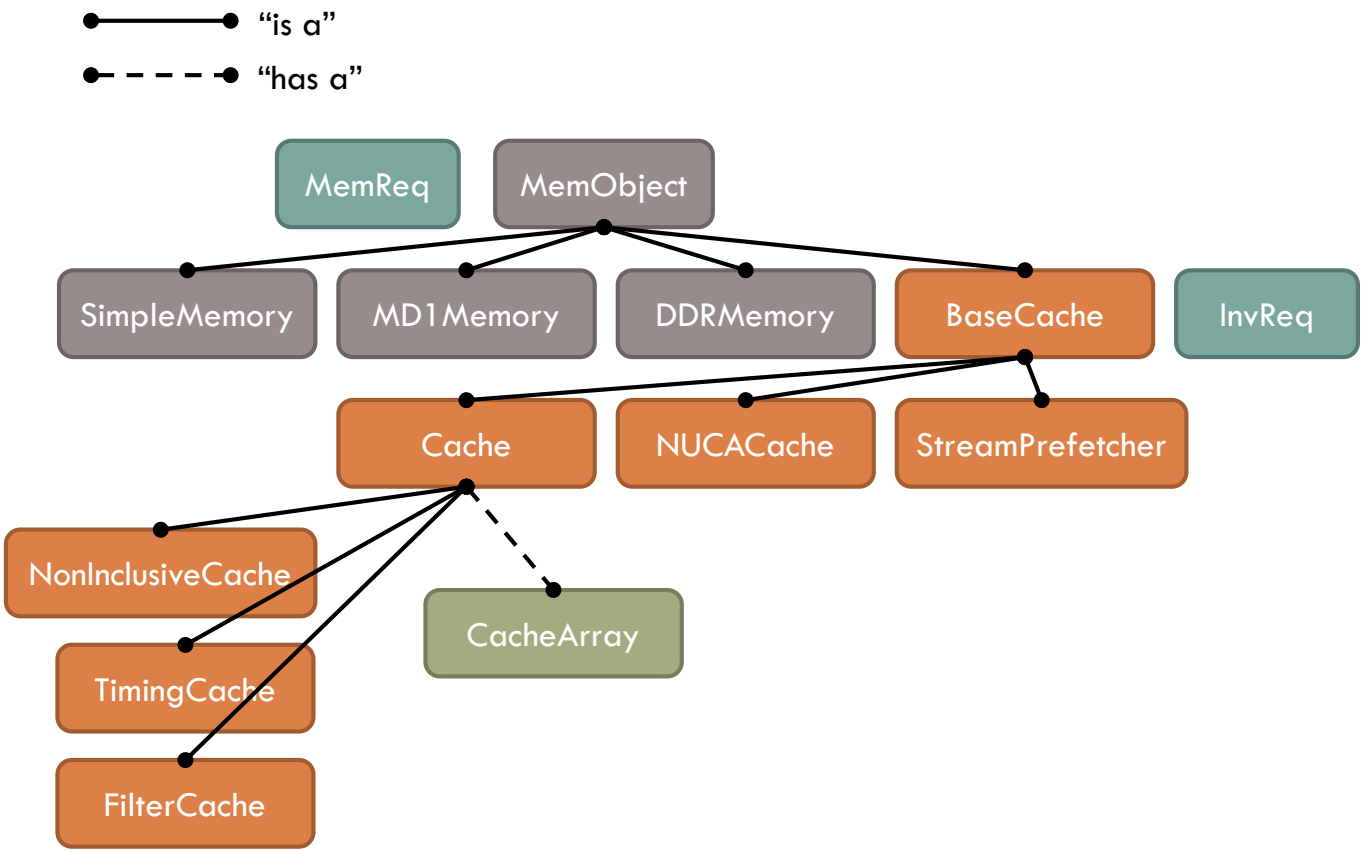
Other caches

- NonInclusiveCache – self explanatory, requires separate directory for coherence
- TimingCache – adds weave-phase models for cache contention
- FilterCache – boundary between core models & memory models
  - Speeds up simulator by accelerating loads & stores
  - Important methods: uint64_t load/store(Address vAddr, uint64_t curCycle)
  - FilterCache adds a virtually-indexed, direct-mapped cache to filter accesses before they reach the more expensive Cache-hierarchy
  - Filter is kept coherent and checks for timing hazards (eg, OOO store execution)

Important ZSim memory classes



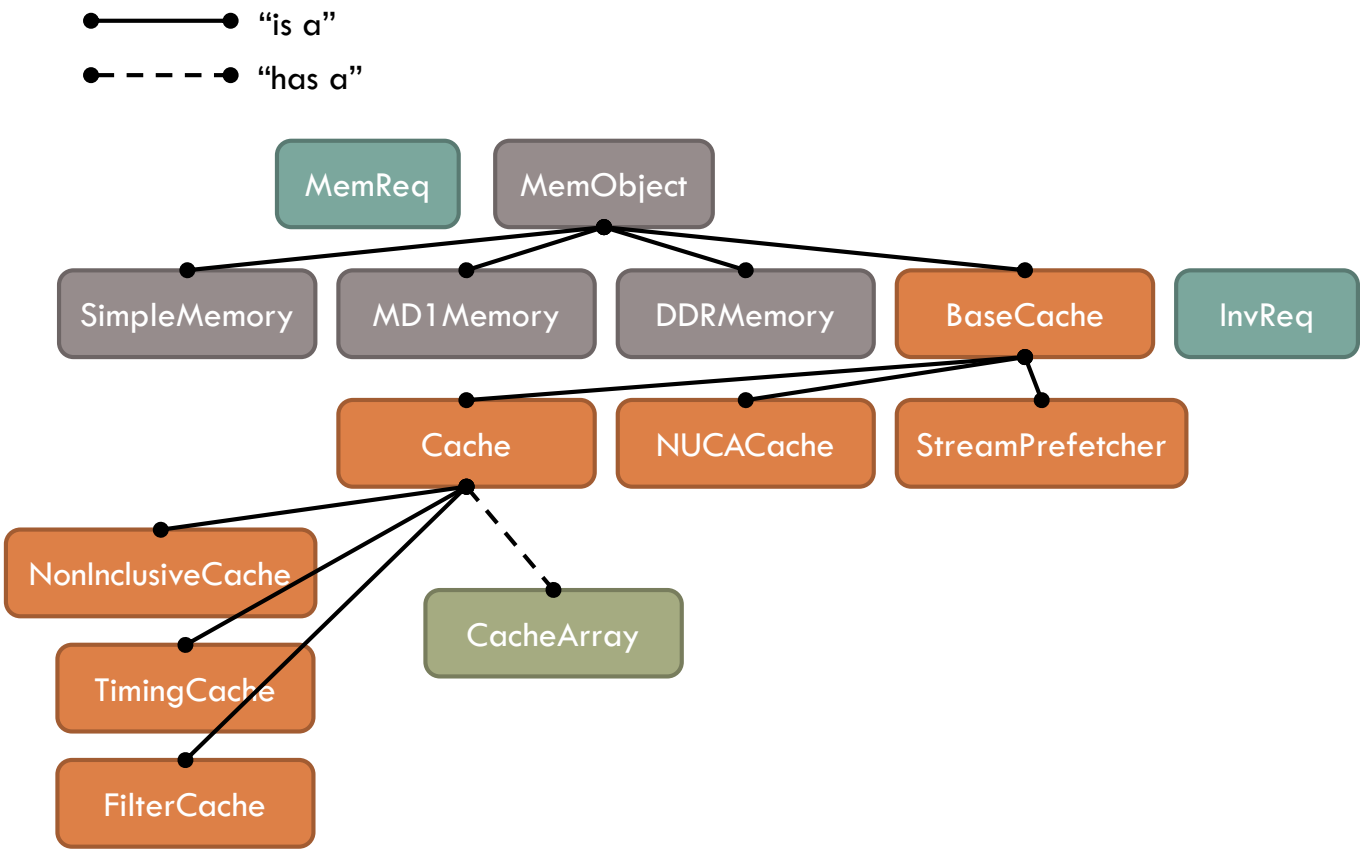
Important ZSim memory classes



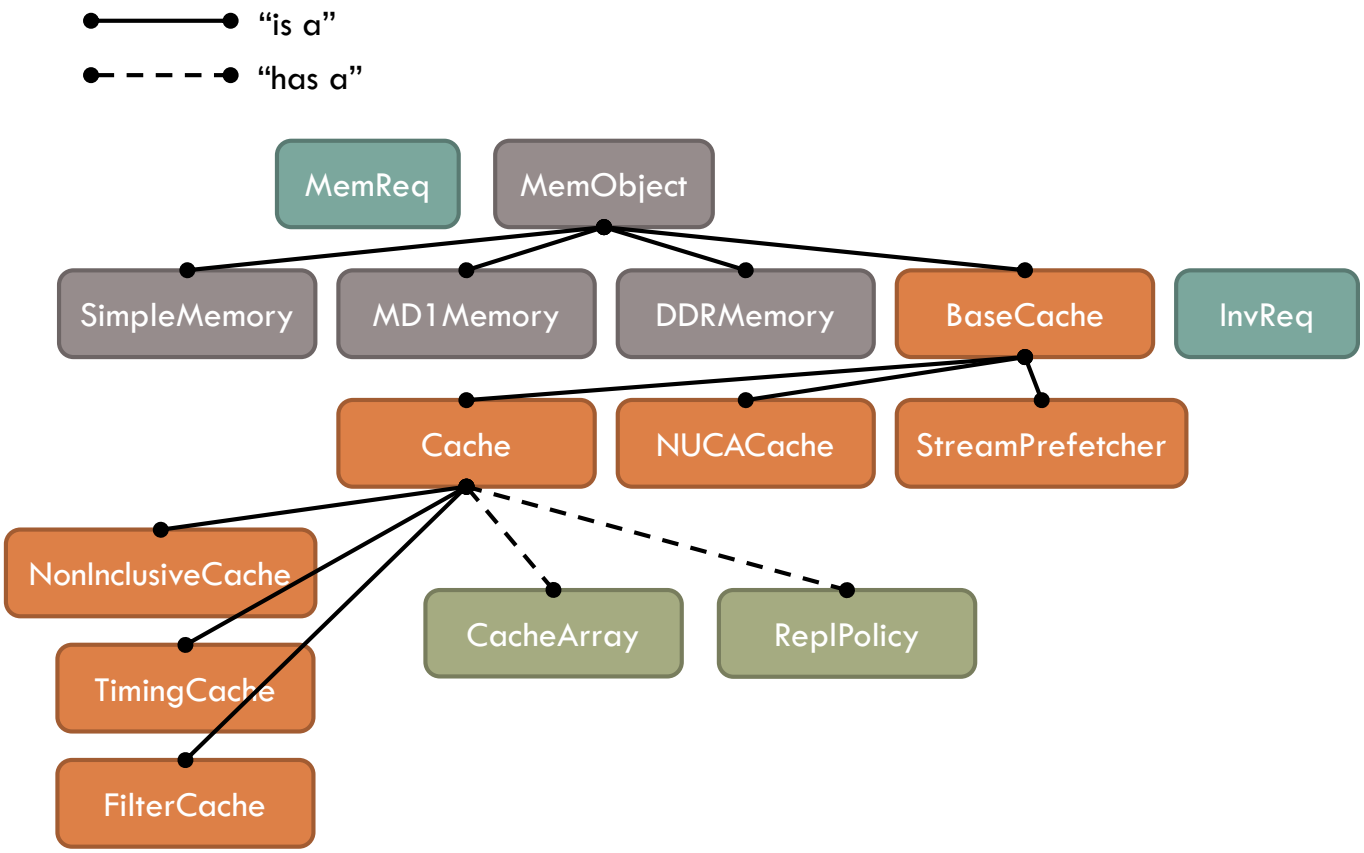
CacheArray

- Implements a tag array with different organizations
- Important fields: None
- Important methods:
  - int32_t lookup(…) – does the array hold this address? If so, which line is it?
  - uint32_t preinsert(…) – make space (i.e., find a victim to evict)
  - void postinsert(…) – allocate space (i.e., finalize eviction)
- Replacement split into phases to avoid invalidation races
- ZSim supports set associative, fully associative, zcaches
  - Compressed caches in development

Important ZSim memory classes



Important ZSim memory classes



ReplPolicy

- The replacement policy ☺
- Important fields: None
- Important methods:
  - void update(uint32_t id, const MemReq* req) – called upon hit
  - void replaced(uint32_t id) – called upon eviction
  - template<class C> uint32_t rankCands(const MemReq* req, C cands) – find a victim
    - For performance, this is optimized at compile time to different arrays
    - Different versions auto-generated from DECL_RANK_BINDINGS() macro
- ZSim supports LRU, pseudo-LRU, NRU, LFU, random, SRRIP, DRRIP, SHiP, PDP, and many more!

Example: Implementing LRU

- Timestamp-based implementation, evict the oldest line

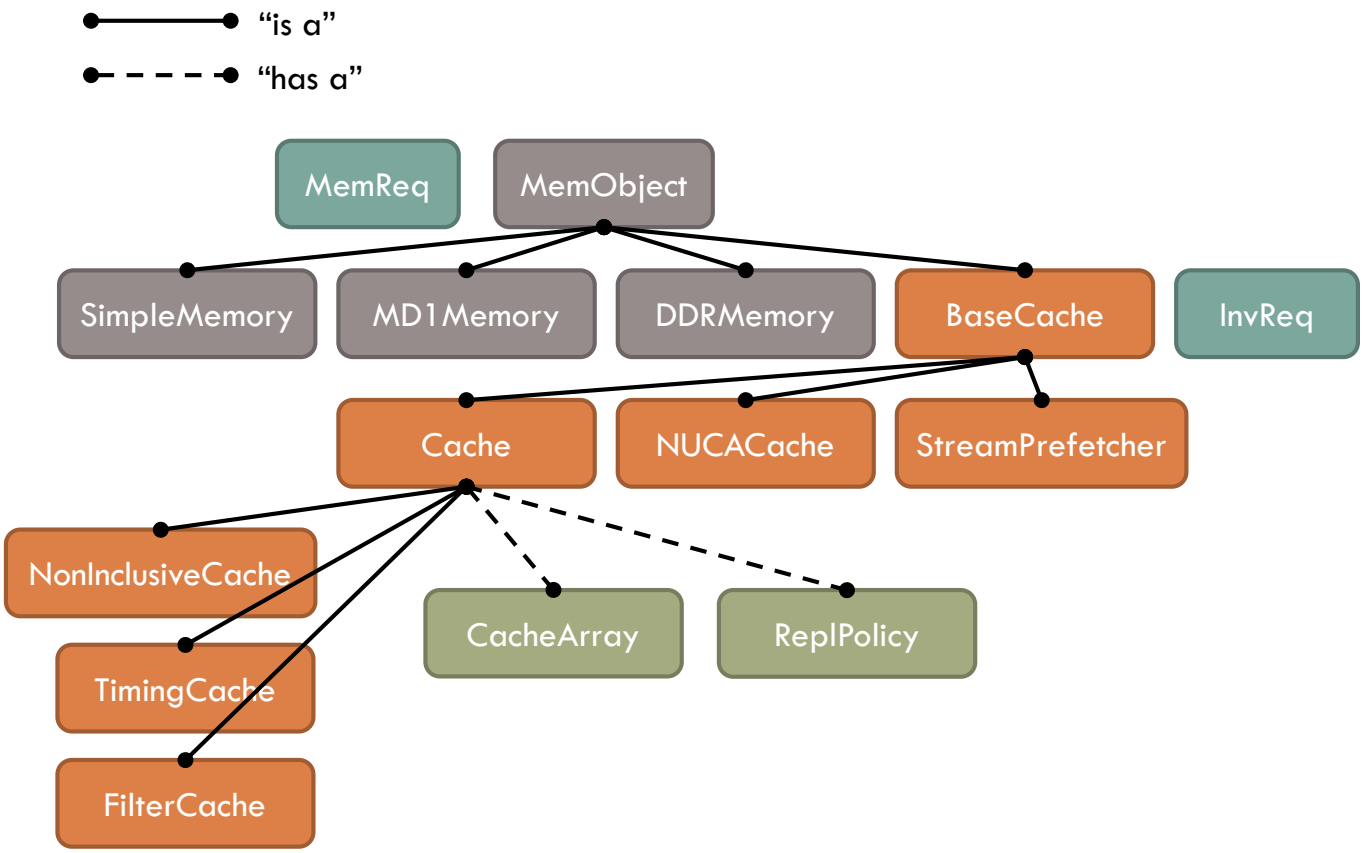
```
class LRUReplPolicy : public ReplPolicy {
    uint64_t timestamp; // global access count
    uint64_t* array;    // last-use timestamp per line
    uint64_t numLines;

public:
    explicit LRUReplPolicy(uint32_t _numLines) : timestamp(1),
numLines(_numLines) {
        array = gm_calloc<uint64_t>(numLines);
    }
    ~LRUReplPolicy() { gm_free(array); }

    void update(uint32_t id, const MemReq* req) { // called upon hit
        array[id] = timestamp++;
    }
    void replaced(uint32_t id) { // called upon eviction
        array[id] = 0;
    }
    …
```

Example: Implementing LRU

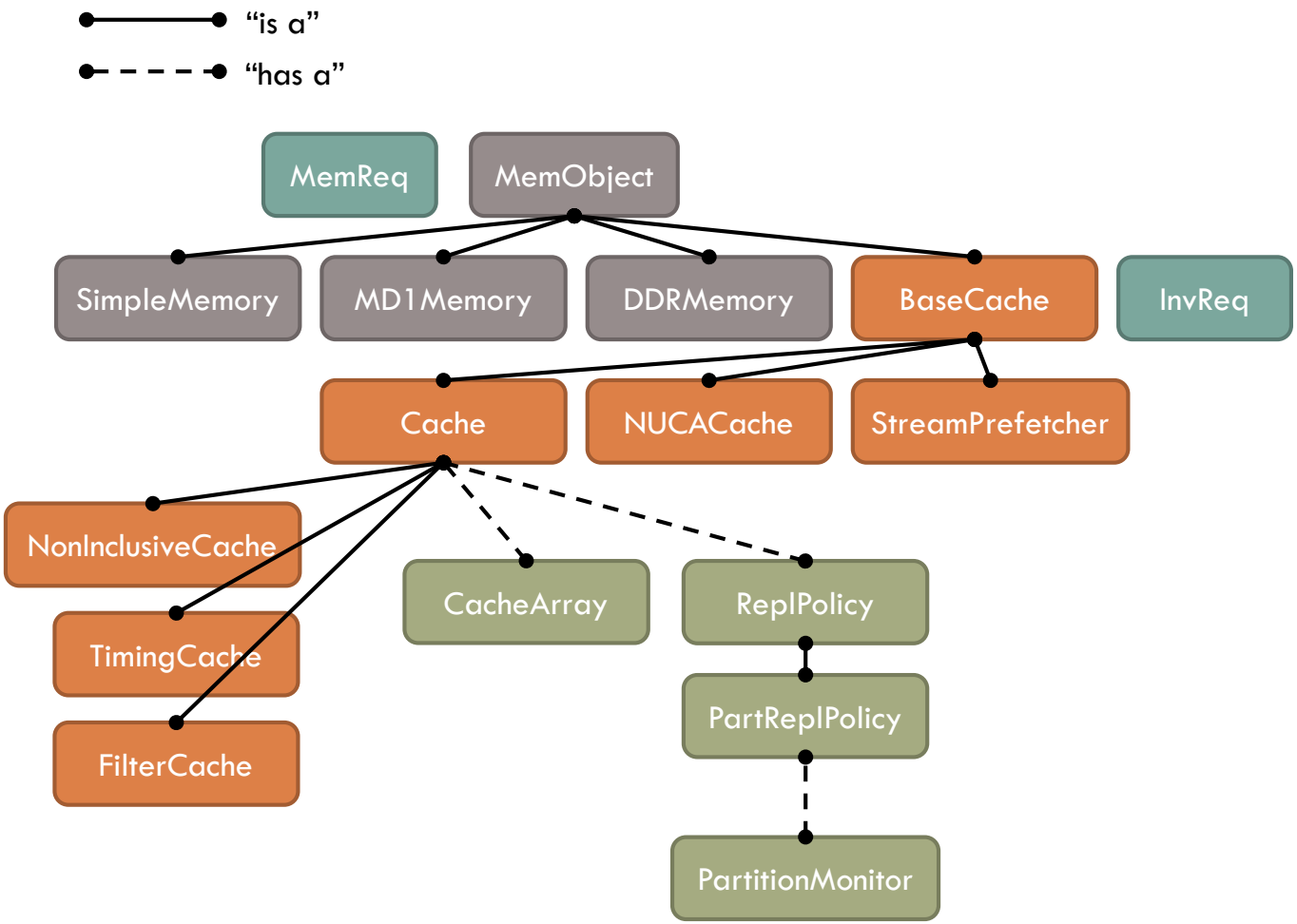
- Timestamp-based implementation, evict the oldest line

```
    …
    template<typename C>
    uint32_t uint32_t rank(const MemReq* req, C cands) {
        uint32_t bestCand = -1;
        for (auto ci = cands.begin(); ci != cands.end(); ci++) {
            if (array[*ci] == 0) { return *ci; }
            else if (timestamp – array[*ci] <
                     timestamp – array[bestCand]) {
                bestCand = *ci;
            }
        }
        return bestCand;
    }
    DECL_RANK_BINDINGS();
};
```

Important ZSim memory classes



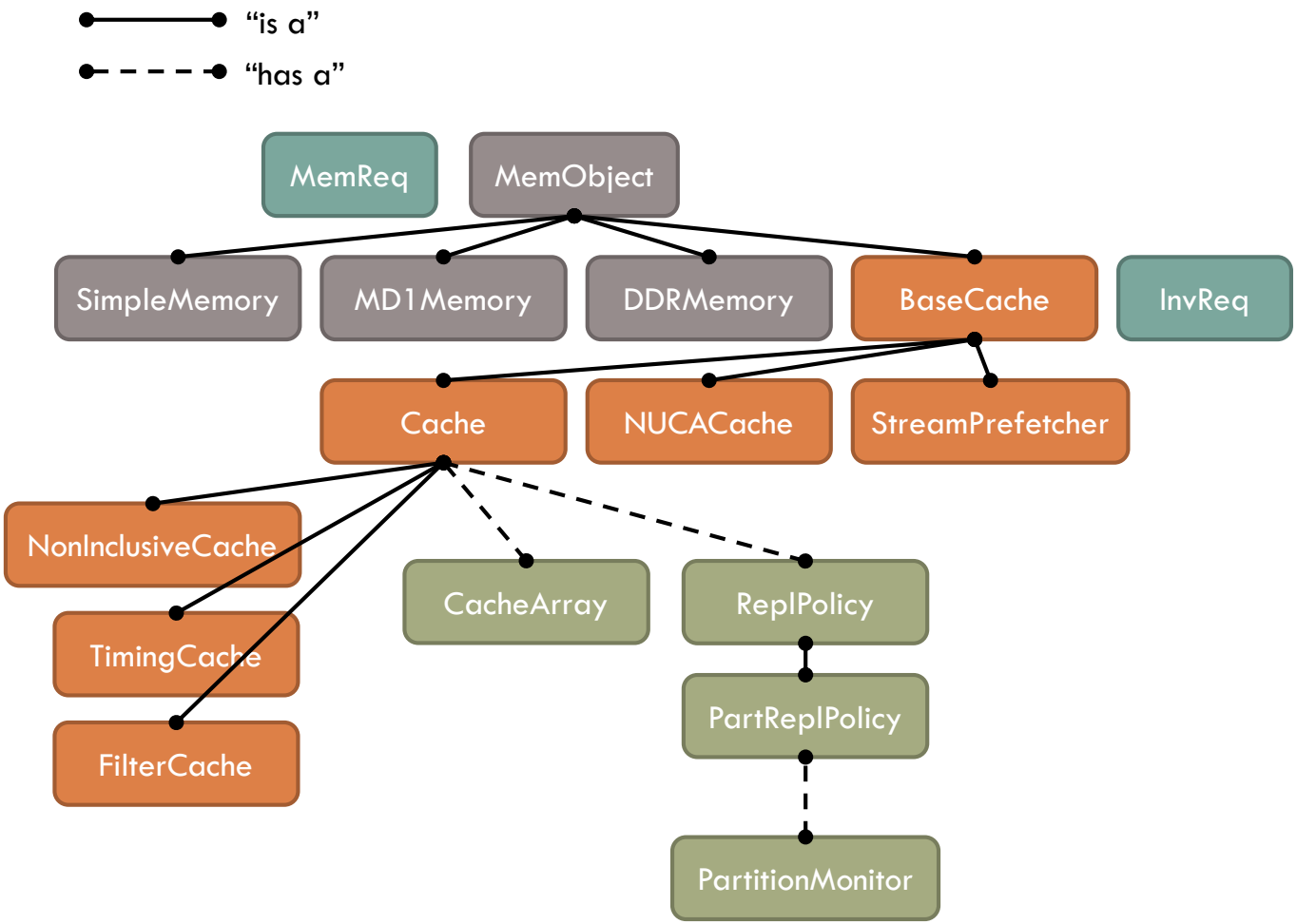
Important ZSim memory classes



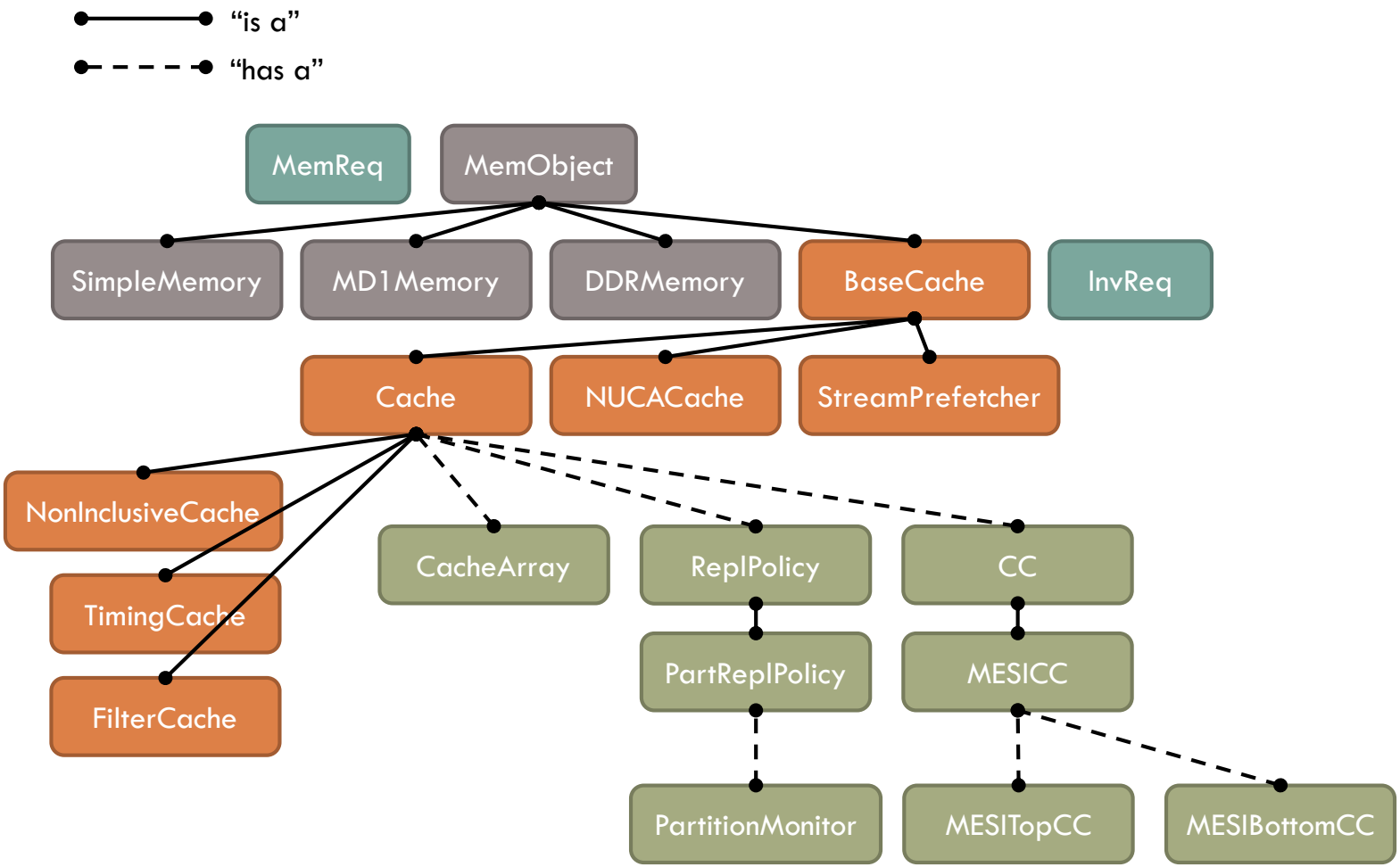
PartReplPolicy

- ZSim implements cache partitioning in the replacement policy
- Important fields:
  - PartitionMapper* mapper – maps MemReqs to a partition
  - PartitionMonitor* monitor – measure stats about different partitions, e.g. miss curves
- Important methods:
  - void setPartitionSizes(const uint32_t* sizes) – reset partition sizes
- ZSim supports way partitioning, idealized LRU partitioning, and Vantage

Important ZSim memory classes



Important ZSim memory classes



CC

- Implements coherence across cache levels
- Important fields: None
- Important methods:
  - void setParents/setChildren(…) – similar to Cache
  - bool startAccess(MemReq& req)
  - bool shouldAllocate(const MemReq& req)
  - uint64_t processEviction(…)
  - uint64_t processAccess(…)
  - void endAccess(const MemReq& req)
  - void startInv()
  - uint64_t processInv(…)
  - uint64_t numSharers(uint32_t lineId)
  - bool isValid(uint32_t lineId)
  - MESIState getState(uint32_t lineId)
  - bool isSharer(uint32_t lineId, uint32_t childId)

CC

- Implements coherence across cache levels
- Important fields: None
- Important methods:
  - void setParents/setChildren(…) – similar to Cache
  - bool startAccess(MemReq& req)
  - bool shouldAllocate(const MemReq& req)
  - uint64_t processEviction(…)
  - uint64_t processAccess(…)
  - void endAccess(const MemReq& req)

  Regular accesses

  - void startInv()
  - uint64_t processInv(…)

  - uint64_t numSharers(uint32_t lineId)
  - bool isValid(uint32_t lineId)
  - MESIState getState(uint32_t lineId)
  - bool isSharer(uint32_t lineId, uint32_t childId)

CC

- Implements coherence across cache levels
- Important fields: None
- Important methods:
  - void setParents/setChildren(…) – similar to Cache
  - bool startAccess(MemReq& req)
  - bool shouldAllocate(const MemReq& req)
  - uint64_t processEviction(…)
  - uint64_t processAccess(…)
  - void endAccess(const MemReq& req)

  Regular accesses

  - void startInv()
  - uint64_t processInv(…)

  Invalidations

  - uint64_t numSharers(uint32_t lineId)
  - bool isValid(uint32_t lineId)
  - MESIState getState(uint32_t lineId)
  - bool isSharer(uint32_t lineId, uint32_t childId)

CC

- Implements coherence across cache levels
- Important fields: None
- Important methods:
  - void setParents/setChildren(…) – similar to Cache
  - bool startAccess(MemReq& req)
  - bool shouldAllocate(const MemReq& req)
  - uint64_t processEviction(…)
  - uint64_t processAccess(…)
  - void endAccess(const MemReq& req)

  Regular accesses

  - void startInv()
  - uint64_t processInv(…)

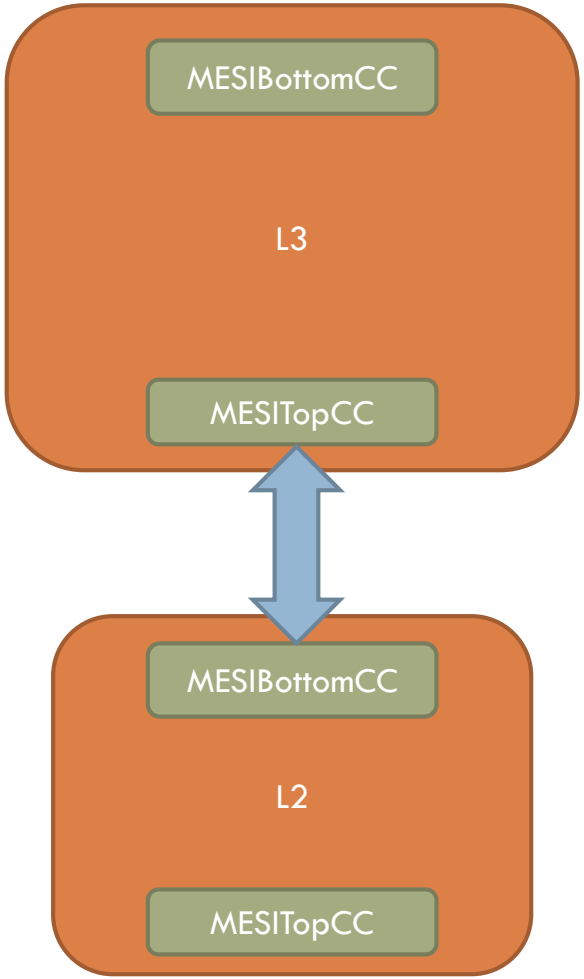
  Invalidations

  - uint64_t numSharers(uint32_t lineId)
  - bool isValid(uint32_t lineId)
  - MESIState getState(uint32_t lineId)
  - bool isSharer(uint32_t lineId, uint32_t childId)

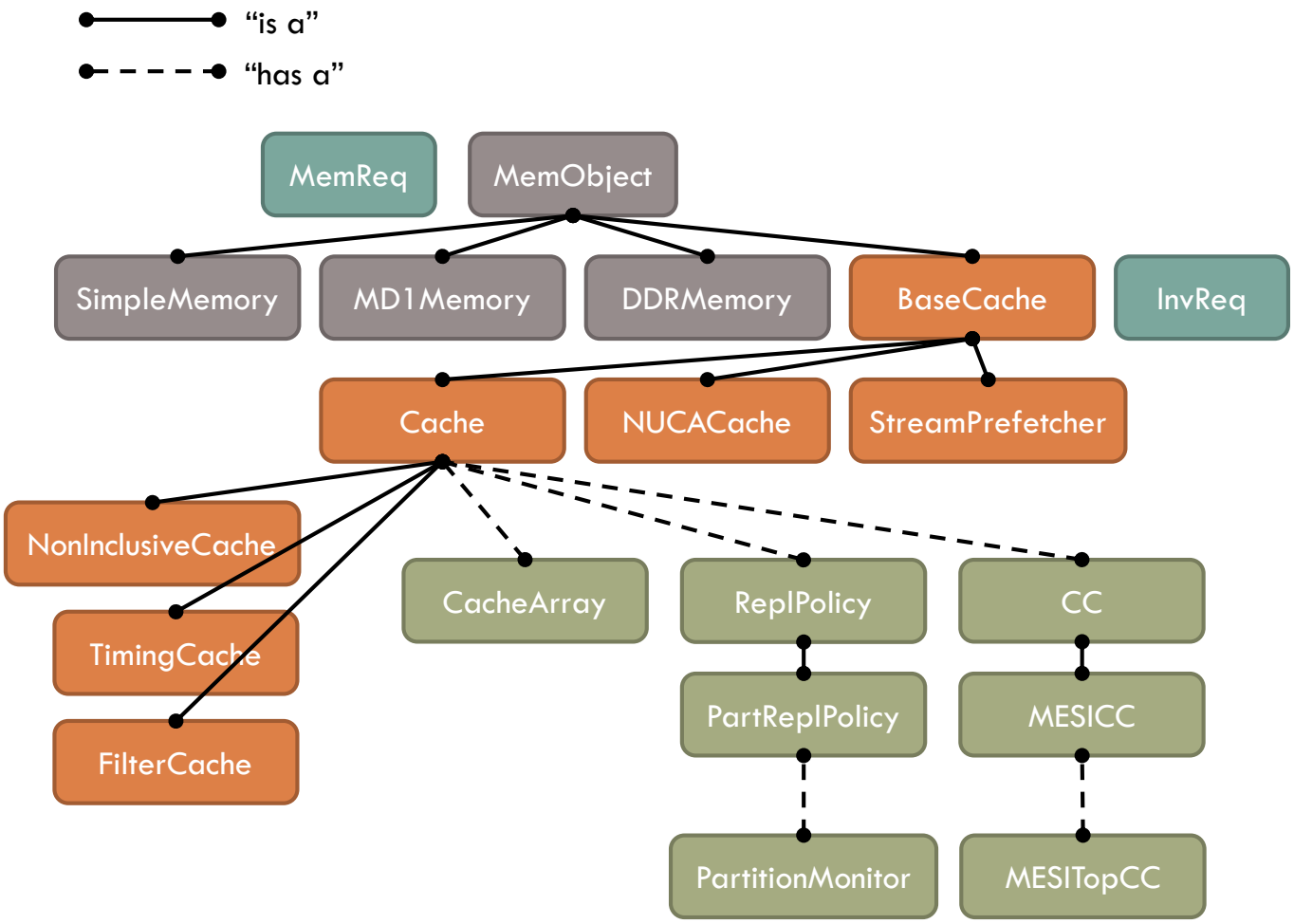
  Querying (e.g., ReplPolicies)

CC naming convention

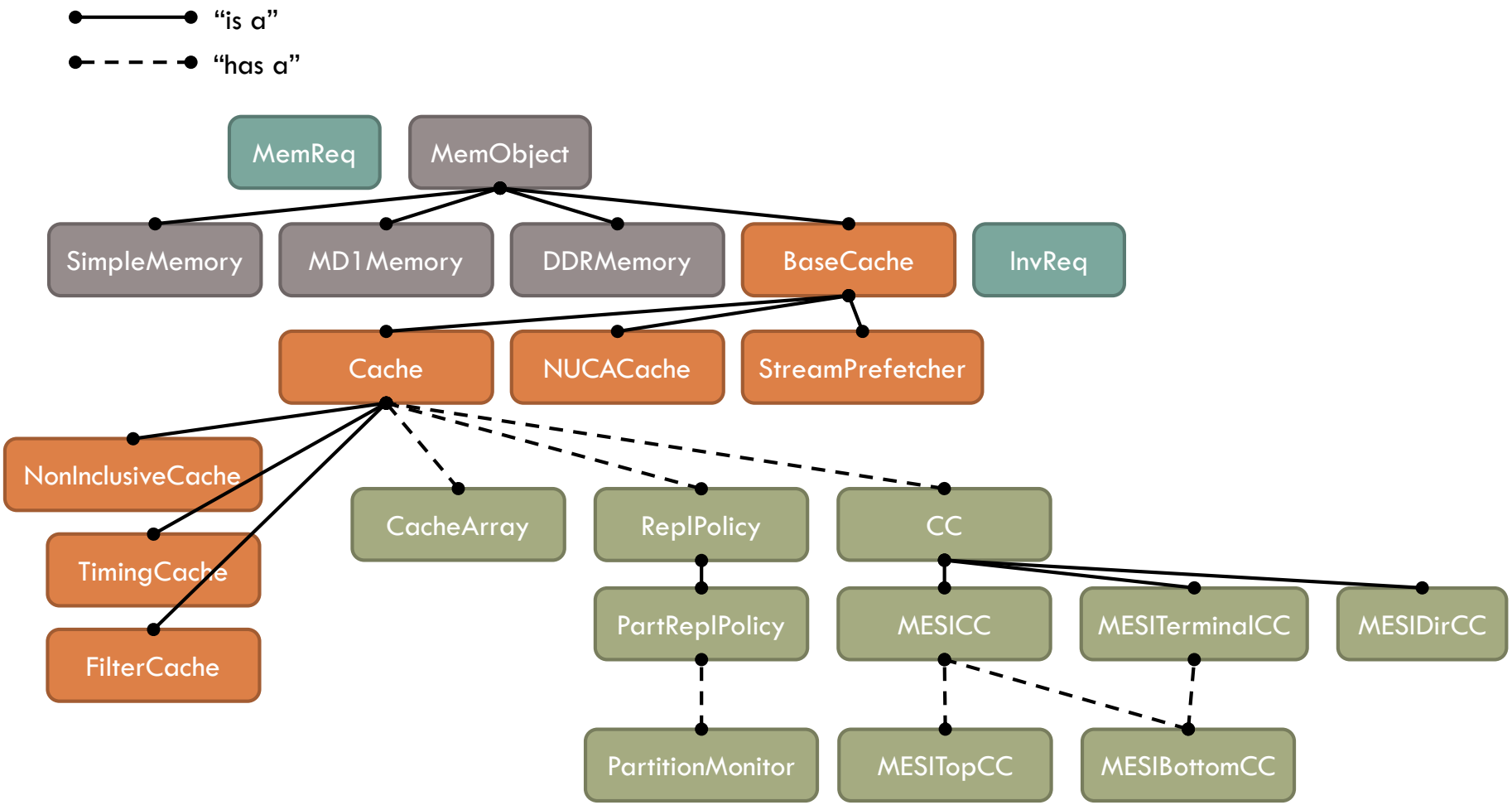
- Top ➔ Parent
- Bottom ➔ Child



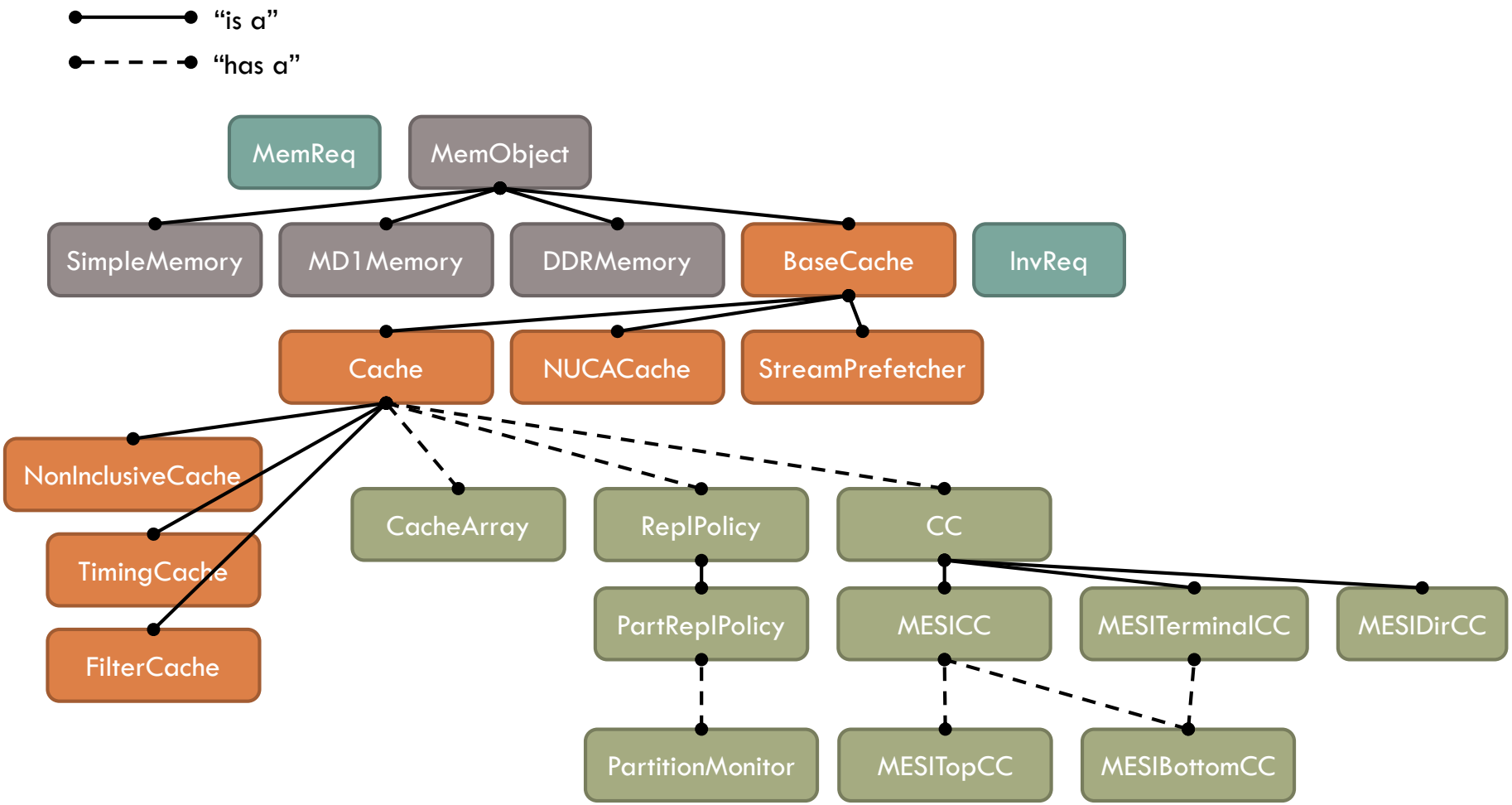
Important ZSim memory classes



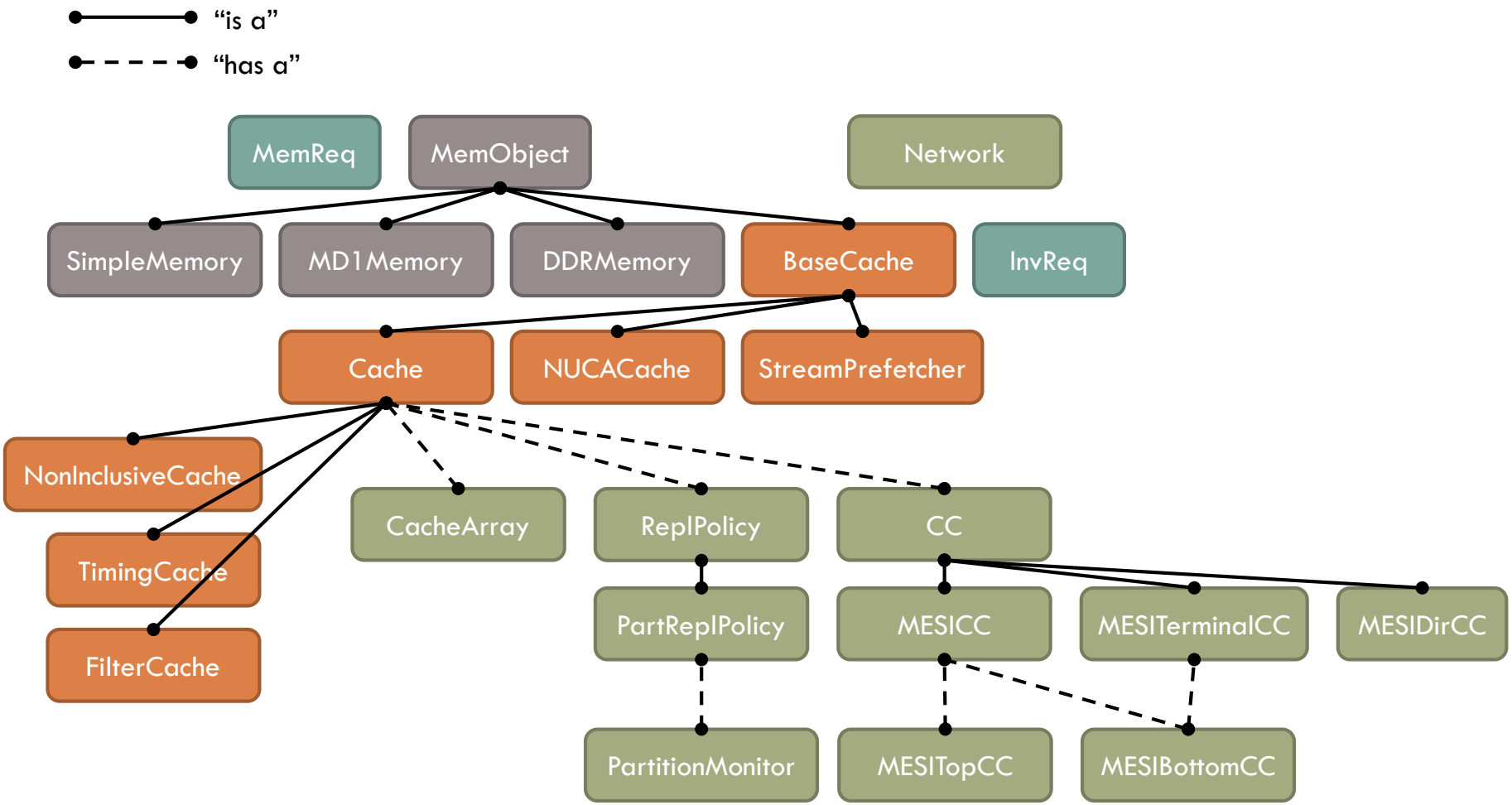
Important ZSim memory classes



Important ZSim memory classes



Important ZSim memory classes



Network

- Tracks round-trip latency between objects in the target system
- **ZSim does not currently model network contention**
- Important fields:
  - string ➔ uint32_t delayMap – maps object pairs (by name) to their round-trip communication latency
- Important methods:
  - Network(const char* filename) – initialize from a network file
  - uint32_t getRTT(const char* src, const char* dst) – Look up network latency
- Will show example network file in Config session later

ZSim memory system

Class	Is a	What is it?	File
MemReq		An in-flight memory request going up the cache hierarchy	memory_hierarchy.h
MemObject		Base object for anything that takes MemReqs. Provides access method.	memory_hierarchy.h
SimpleMemory	MemObject	Fixed-latency main memory.	mem_ctrls.h
MD1Memory	MemObject	M/D/1 queuing latency model for main memory.	mem_ctrls.h
DDRMemory	MemObject	Full DDR timing simulation, requires TimingEvents.	ddr_mem.h
InvReq		An invalidation going down the cache hierarchy	memory_hierarchy.h
BaseCache	MemObject	Base object for caches, prefetchers, etc. Provides setChildren/Parents and invalidate methods.	memory_hierarchy.h
Cache	BaseCache	An inclusive cache.	cache.h
NonInclusiveCache	Cache	A non-inclusive cache.	non_incl_cache.h
TimingCache	Cache	Connects timing events through hierarchy for DDR memory models.	timing_cache.h
FilterCache	Cache	Implements optimized load/store methods for efficiency.	filter_cache.h
NUCACache	BaseCache	Cache with distributed banks internally. Maps addresses to banks via BankDir object.	nuca_cache.h
StreamPrefetcher	BaseCache	Streaming prefetcher.	prefetcher.h
CacheArray		Tracks addresses in cache and structure of array.	cache_arrays.h
ReplPolicy		A replacement policy. Notified of accesses/evictions through update/replaced methods. Chooses victim in rankCands method.	repl_policies.h
PartReplPolicy	ReplPolicy	Implements cache partitioning. Adds setPartitionSizes method and monitoring.	part_repl_policies.h
PartitionMonitor		Monitors partition miss curves. Provides access and getMissCurve methods.	monitor.h
Network		getRTT method provides (fixed) latency between two modeled objects.	network.h
CC		Generic coherence controller (zsim implements MESI).	coherence_ctrls.h