# SIGNET: NETWORK-ON-CHIP FILTERING FOR COARSE VECTOR DIRECTORIES

Natalie Enright Jerger
University of Toronto

# Interaction of Coherence and Network



- Cache coherence protocol drives network-on-chip traffic
- Scalable coherence protocols needed for many-core architectures
- Consider interconnection network optimizations to help facilitate scalable coherence

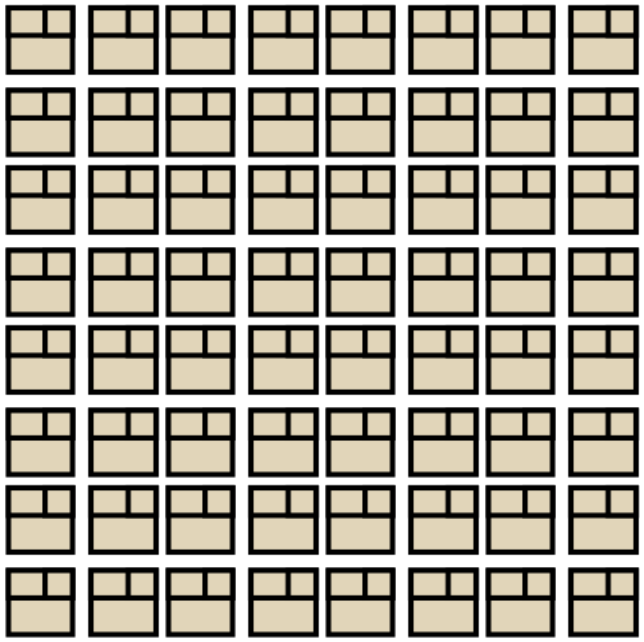
# Talk Outline

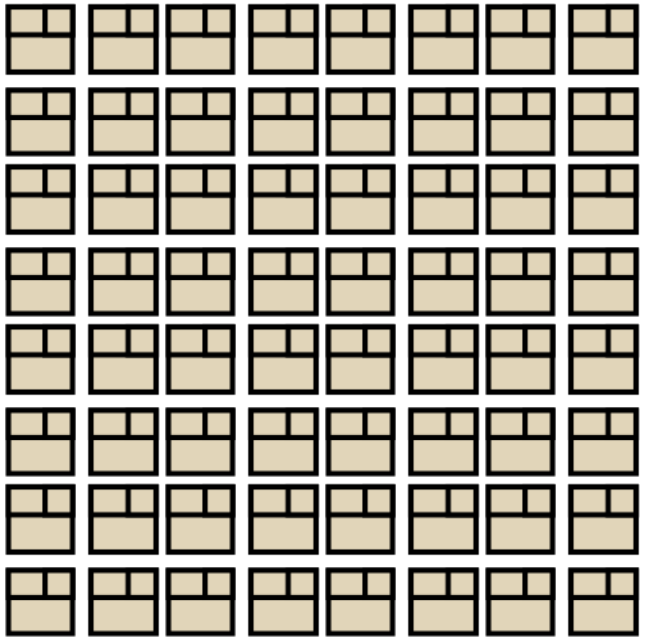


- Introduction
- Network-on-Chip Challenges with Scalable Coherence Protocol
- SigNet Architecture: Network filtering solution
- Evaluation
- Conclusion

# Many-Core Cache Coherence Challenges

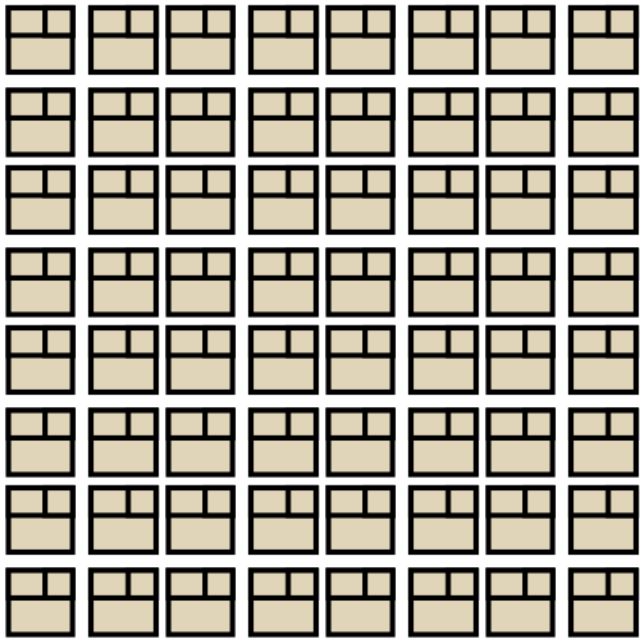


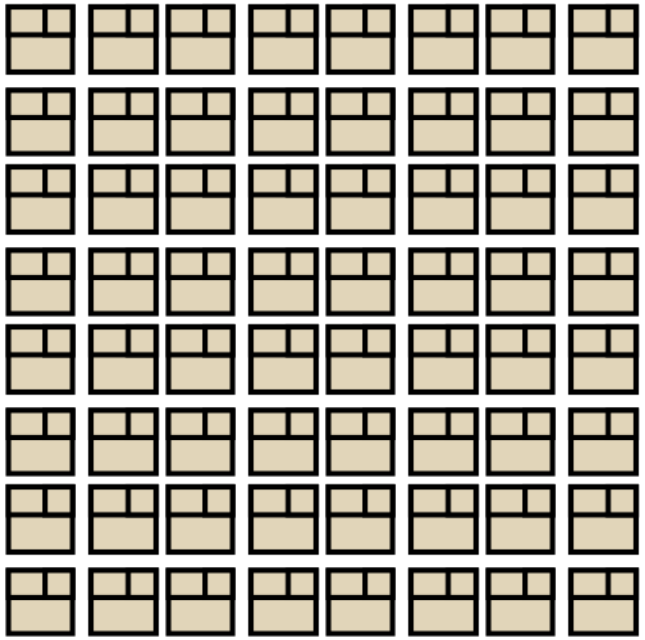




# Many-Core Cache Coherence Challenges



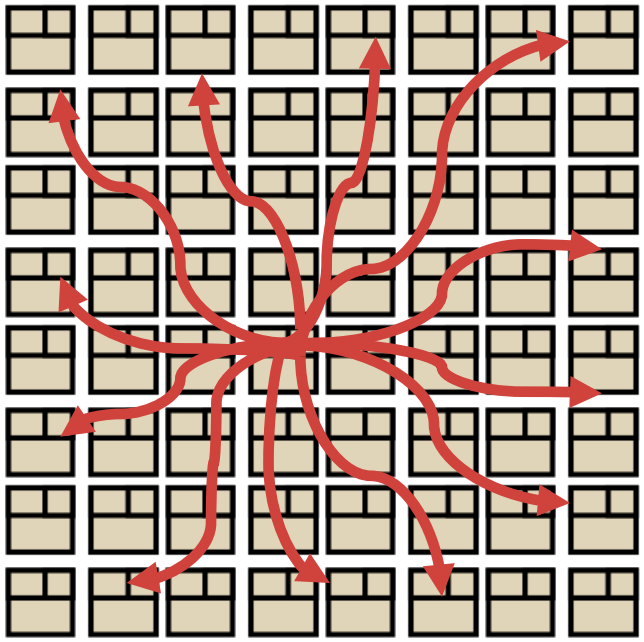


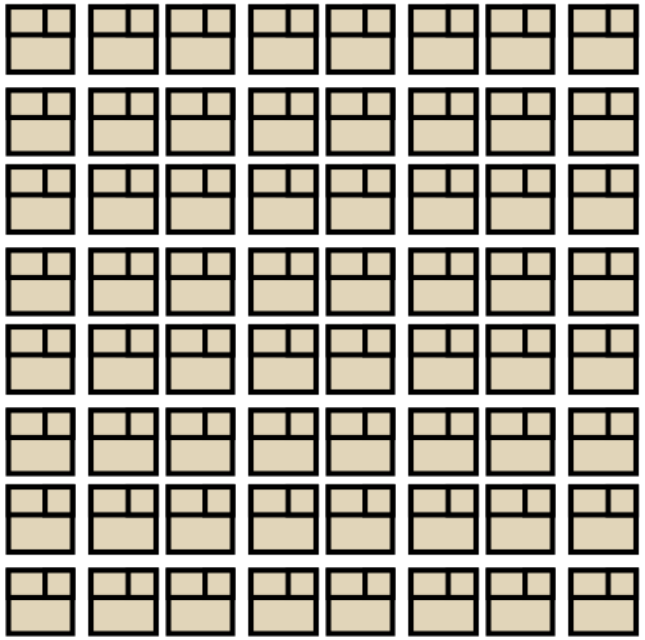


- Broadcast
  - Good latency
  - Poor scaling due to bandwidth requirements

# Many-Core Cache Coherence Challenges



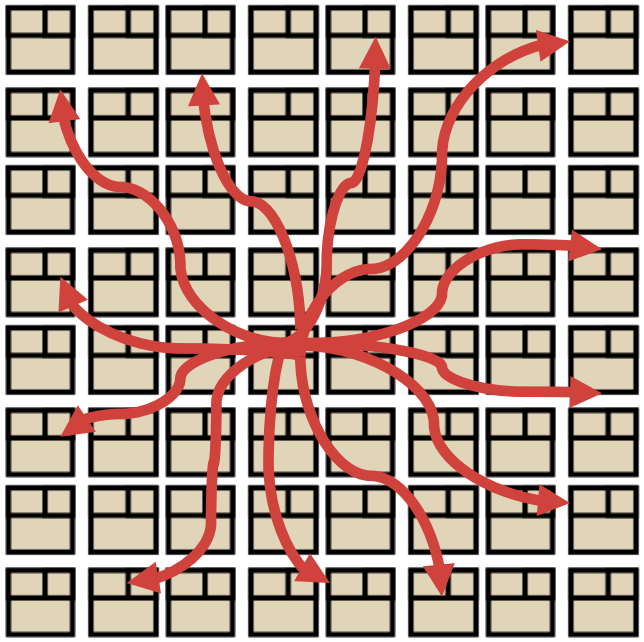




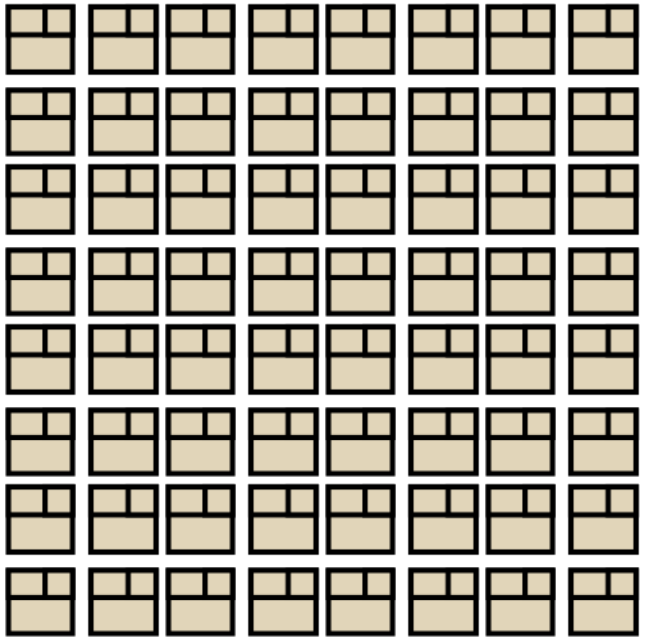
- Broadcast
  - Good latency
  - Poor scaling due to bandwidth requirements

# Many-Core Cache Coherence Challenges





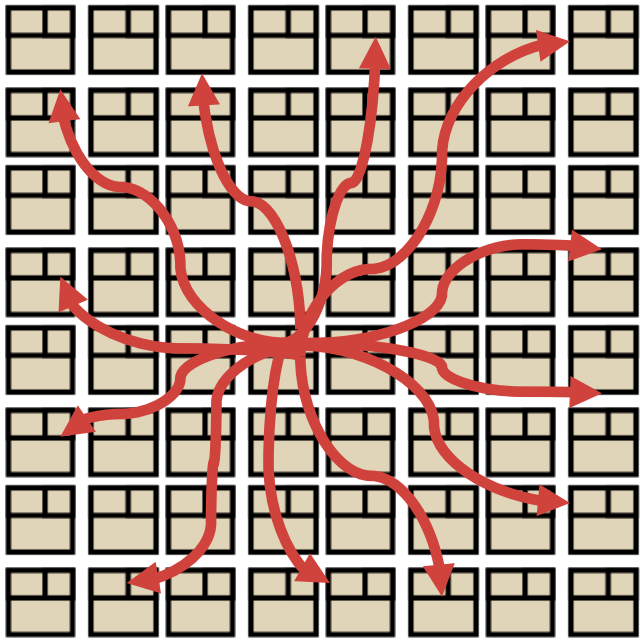
- Broadcast
  - Good latency
  - Poor scaling due to bandwidth requirements

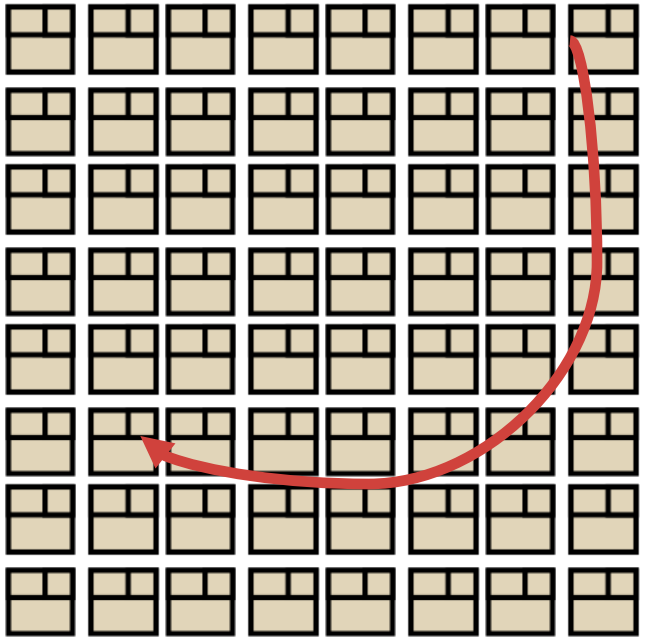


- Directory
  - Good scalability due to point to point communication
  - Storage overheads

# Many-Core Cache Coherence Challenges



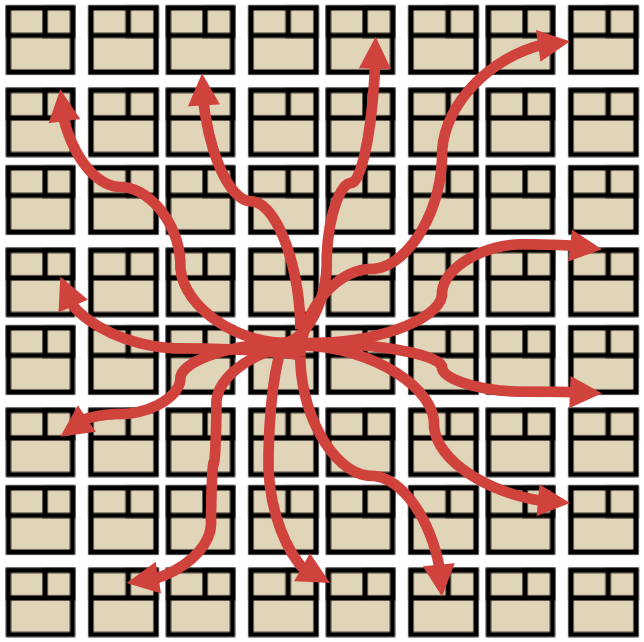




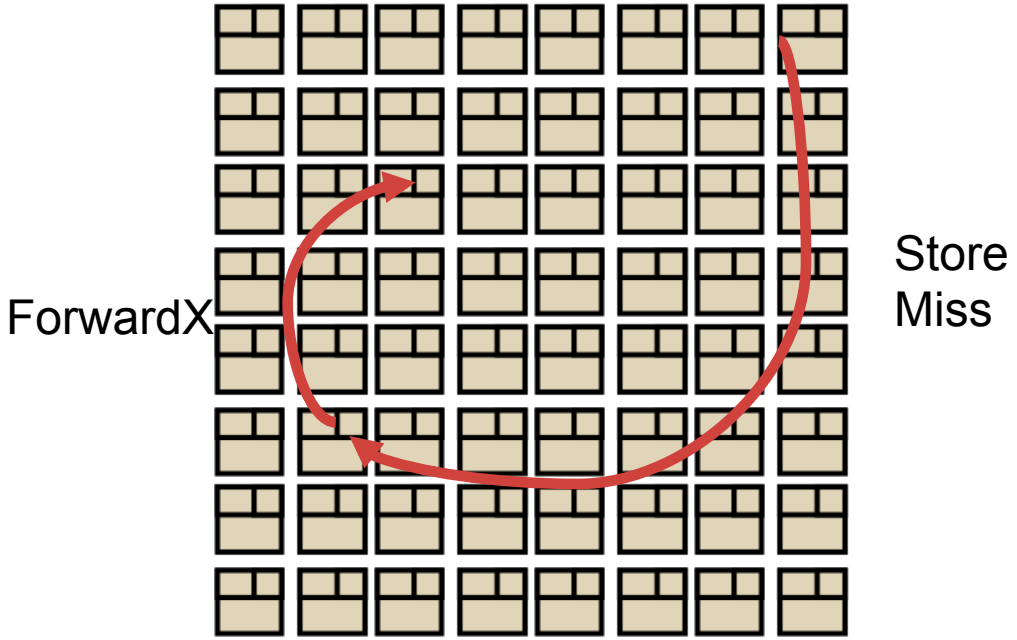

- Broadcast
  - Good latency
  - Poor scaling due to bandwidth requirements
- Directory
  - Good scalability due to point to point communication
  - Storage overheads

# Many-Core Cache Coherence Challenges





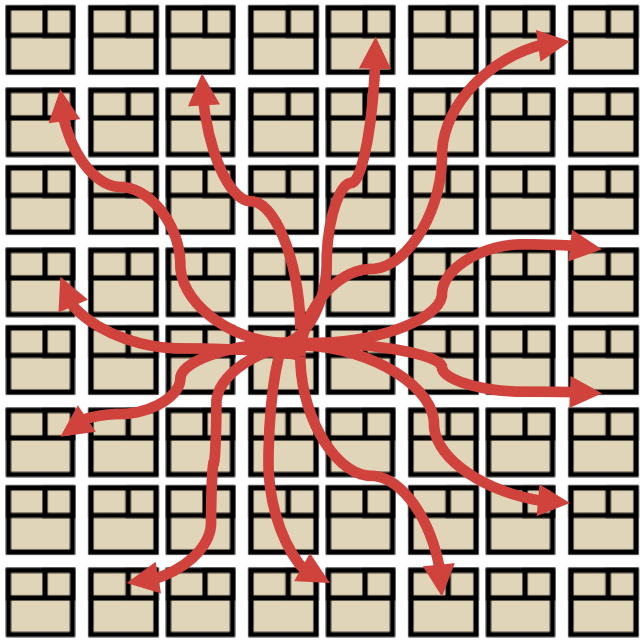
- Broadcast
  - Good latency
  - Poor scaling due to bandwidth requirements

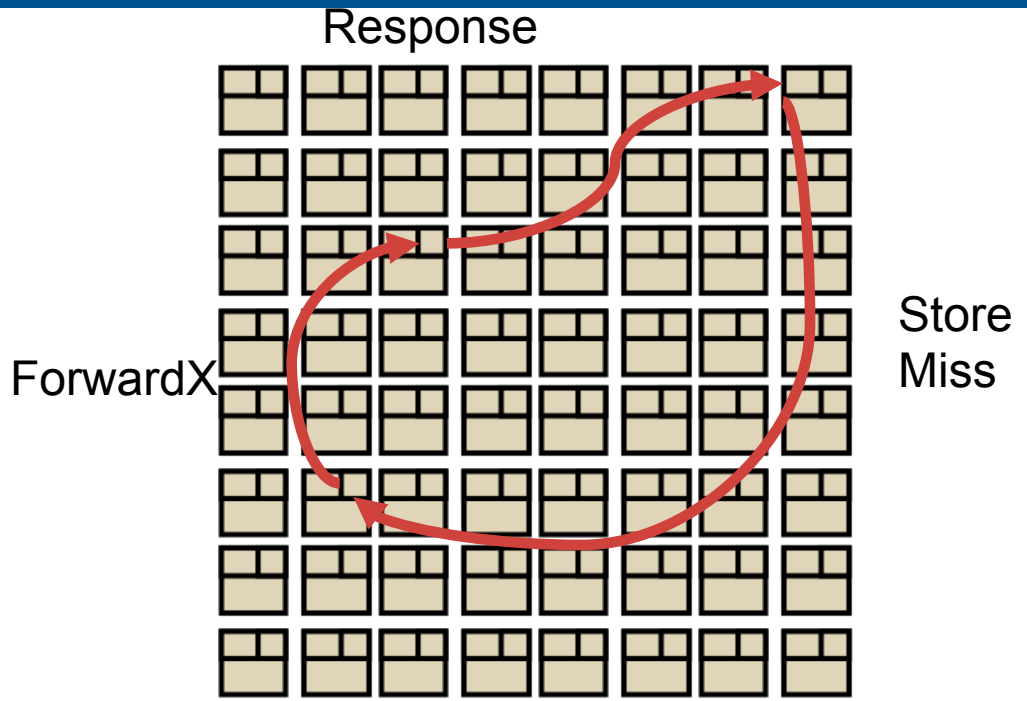



- Directory
  - Good scalability due to point to point communication
  - Storage overheads

# Many-Core Cache Coherence Challenges








- Broadcast
  - Good latency
  - Poor scaling due to bandwidth requirements
- Directory
  - Good scalability due to point to point communication
  - Storage overheads

# Scalable Cache Coherence



# Scalable Cache Coherence



- Directory protocol storage overheads
  - Single bit per core in sharing vector (full map)
  - 256 cores → 32 Bytes of overhead per cache line!

# Scalable Cache Coherence



- Directory protocol storage overheads
  - Single bit per core in sharing vector (full map)
  - 256 cores → 32 Bytes of overhead per cache line!

# Scalable Cache Coherence



- Directory protocol storage overheads
  - Single bit per core in sharing vector (full map)
  - 256 cores → 32 Bytes of overhead per cache line!

# Scalable Cache Coherence



- Directory protocol storage overheads
  - Single bit per core in sharing vector (full map)
  - 256 cores → 32 Bytes of overhead per cache line!

# Scalable Cache Coherence



- Directory protocol storage overheads
  - Single bit per core in sharing vector (full map)
  - 256 cores → 32 Bytes of overhead per cache line!

# Scalable Cache Coherence



- Directory protocol storage overheads
  - Single bit per core in sharing vector (full map)
  - 256 cores → 32 Bytes of overhead per cache line!

Cores 1, 2, 5 & 15 share cache line

# Scalable Cache Coherence



- Directory protocol storage overheads
  - Single bit per core in sharing vector (full map)
  - 256 cores → 32 Bytes of overhead per cache line!

Cores 1, 2, 5 & 15 share cache line

- Coarse Vector Directories
  - $Dir_iCV_r$
    - i: # of pointers
    - r: # of cores in region
  - Example: $Dir_2CV_2$
    - Requires 1/2 as much storage

# Scalable Cache Coherence



- Directory protocol storage overheads
  - Single bit per core in sharing vector (full map)
  - 256 cores → 32 Bytes of overhead per cache line!

Cores 1, 2, 5 & 15 share cache line

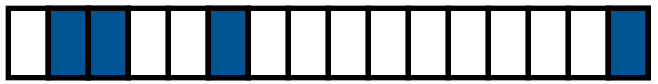
2 pointers

- Coarse Vector Directories
  - $Dir_iCV_r$
    - i: # of pointers
    - r: # of cores in region
  - Example: $Dir_2CV_2$
    - Requires 1/2 as much storage

# Scalable Cache Coherence

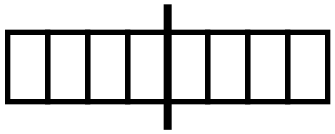


- Directory protocol storage overheads
  - Single bit per core in sharing vector (full map)
  - 256 cores → 32 Bytes of overhead per cache line!



Cores 1, 2, 5 & 15 share cache line

2 pointers

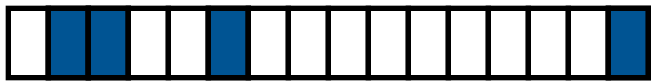


- Coarse Vector Directories
  - $Dir_iCV_r$
    - i: # of pointers
    - r: # of cores in region
  - Example: $Dir_2CV_2$
    - Requires 1/2 as much storage

# Scalable Cache Coherence

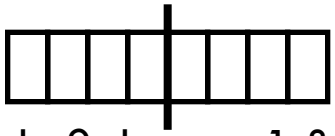


- Directory protocol storage overheads
  - Single bit per core in sharing vector (full map)
  - 256 cores → 32 Bytes of overhead per cache line!



Cores 1, 2, 5 & 15 share cache line

2 pointers



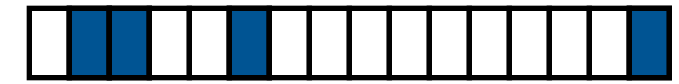
Only 2 sharers: 1 & 15

- Coarse Vector Directories
  - $Dir_iCV_r$
    - i: # of pointers
    - r: # of cores in region
  - Example: $Dir_2CV_2$
    - Requires 1/2 as much storage

# Scalable Cache Coherence

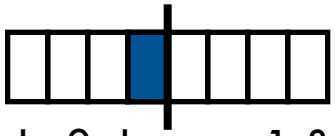


- Directory protocol storage overheads
  - Single bit per core in sharing vector (full map)
  - 256 cores → 32 Bytes of overhead per cache line!



Cores 1, 2, 5 & 15 share cache line

2 pointers



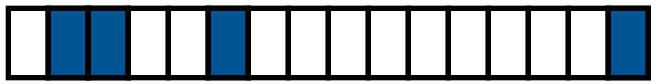
Only 2 sharers: 1 & 15

- Coarse Vector Directories
  - $Dir_iCV_r$
    - i: # of pointers
    - r: # of cores in region
  - Example: $Dir_2CV_2$
    - Requires 1/2 as much storage

# Scalable Cache Coherence



- Directory protocol storage overheads
  - Single bit per core in sharing vector (full map)
  - 256 cores → 32 Bytes of overhead per cache line!



Cores 1, 2, 5 & 15 share cache line

2 pointers

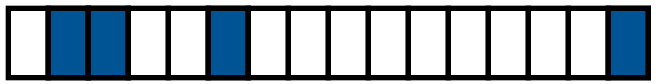
Only 2 sharers: 1 & 15

- Coarse Vector Directories
  - $Dir_iCV_r$
    - i: # of pointers
    - r: # of cores in region
  - Example: $Dir_2CV_2$
    - Requires 1/2 as much storage

# Scalable Cache Coherence



- Directory protocol storage overheads
  - Single bit per core in sharing vector (full map)
  - 256 cores → 32 Bytes of overhead per cache line!



Cores 1, 2, 5 & 15 share cache line

2 pointers

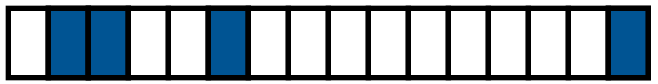
Only 2 sharers: 1 & 15

- Coarse Vector Directories
  - $Dir_iCV_r$
    - i: # of pointers
    - r: # of cores in region
  - Example: $Dir_2CV_2$
    - Requires 1/2 as much storage

# Scalable Cache Coherence



- Directory protocol storage overheads
  - Single bit per core in sharing vector (full map)
  - 256 cores → 32 Bytes of overhead per cache line!



Cores 1, 2, 5 & 15 share cache line

- Coarse Vector Directories
  - $Dir_iCV_r$
    - i: # of pointers
    - r: # of cores in region
  - Example: $Dir_2CV_2$
    - Requires 1/2 as much storage

2 pointers

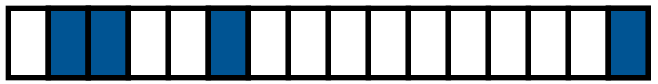
Only 2 sharers: 1 & 15

3rd sharer: overflow

# Scalable Cache Coherence



- Directory protocol storage overheads
  - Single bit per core in sharing vector (full map)
  - 256 cores → 32 Bytes of overhead per cache line!



Cores 1, 2, 5 & 15 share cache line

- Coarse Vector Directories
  - $Dir_iCV_r$
    - i: # of pointers
    - r: # of cores in region
  - Example: $Dir_2CV_2$
    - Requires 1/2 as much storage

2 pointers

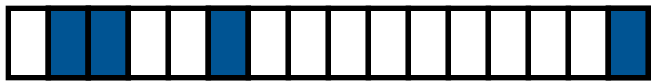
Only 2 sharers: 1 & 15

3rd sharer: overflow

# Scalable Cache Coherence



- Directory protocol storage overheads
  - Single bit per core in sharing vector (full map)
  - 256 cores → 32 Bytes of overhead per cache line!



Cores 1, 2, 5 & 15 share cache line

- Coarse Vector Directories
  - $Dir_iCV_r$
    - i: # of pointers
    - r: # of cores in region
  - Example: $Dir_2CV_2$
    - Requires 1/2 as much storage

2 pointers

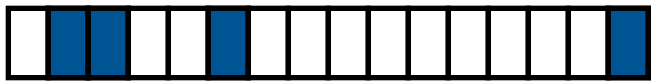
Only 2 sharers: 1 & 15

3rd sharer: overflow

# Scalable Cache Coherence



- Directory protocol storage overheads
  - Single bit per core in sharing vector (full map)
  - 256 cores → 32 Bytes of overhead per cache line!



Cores 1, 2, 5 & 15 share cache line

- Coarse Vector Directories
  - $Dir_iCV_r$
    - i: # of pointers
    - r: # of cores in region
  - Example: $Dir_2CV_2$
    - Requires 1/2 as much storage

2 pointers

Only 2 sharers: 1 & 15

3rd sharer: overflow

# Scalable Cache Coherence



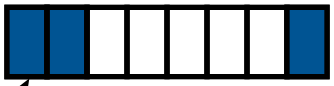
- Directory protocol storage overheads
  - Single bit per core in sharing vector (full map)
  - 256 cores → 32 Bytes of overhead per cache line!

Cores 1, 2, 5 & 15 share cache line

- Coarse Vector Directories
  - $Dir_iCV_r$
    - i: # of pointers
    - r: # of cores in region
  - Example: $Dir_2CV_2$
    - Requires 1/2 as much storage

2 pointers

Only 2 sharers: 1 & 15



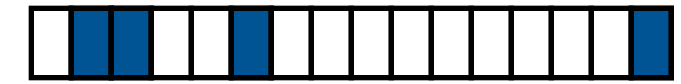
3rd sharer: overflow

3 sharers but 6 invalidations

# Scalable Cache Coherence

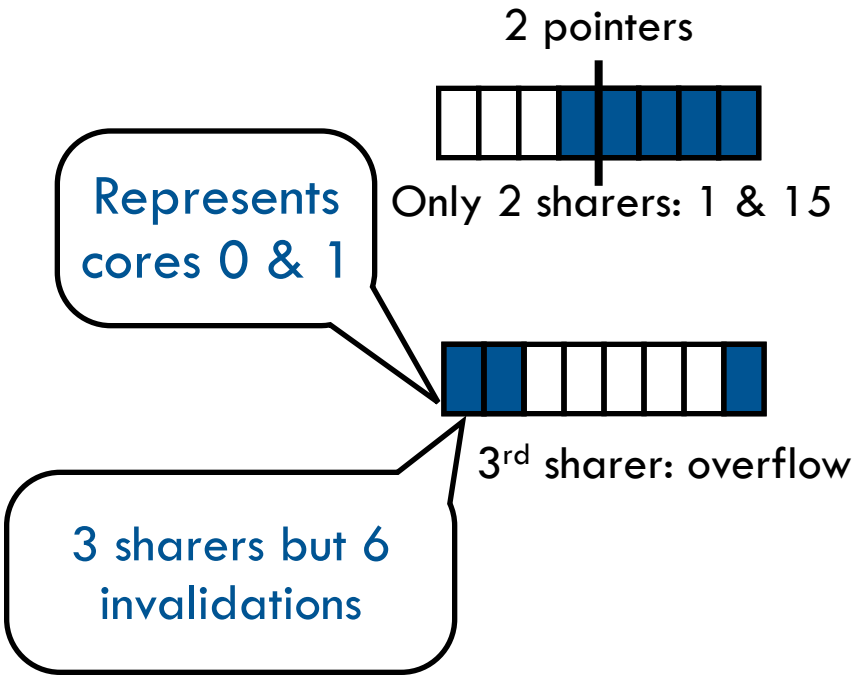


- Directory protocol storage overheads
  - Single bit per core in sharing vector (full map)
  - 256 cores → 32 Bytes of overhead per cache line!



Cores 1, 2, 5 & 15 share cache line

- Coarse Vector Directories
  - $Dir_iCV_r$
    - i: # of pointers
    - r: # of cores in region
  - Example: $Dir_2CV_2$
    - Requires 1/2 as much storage



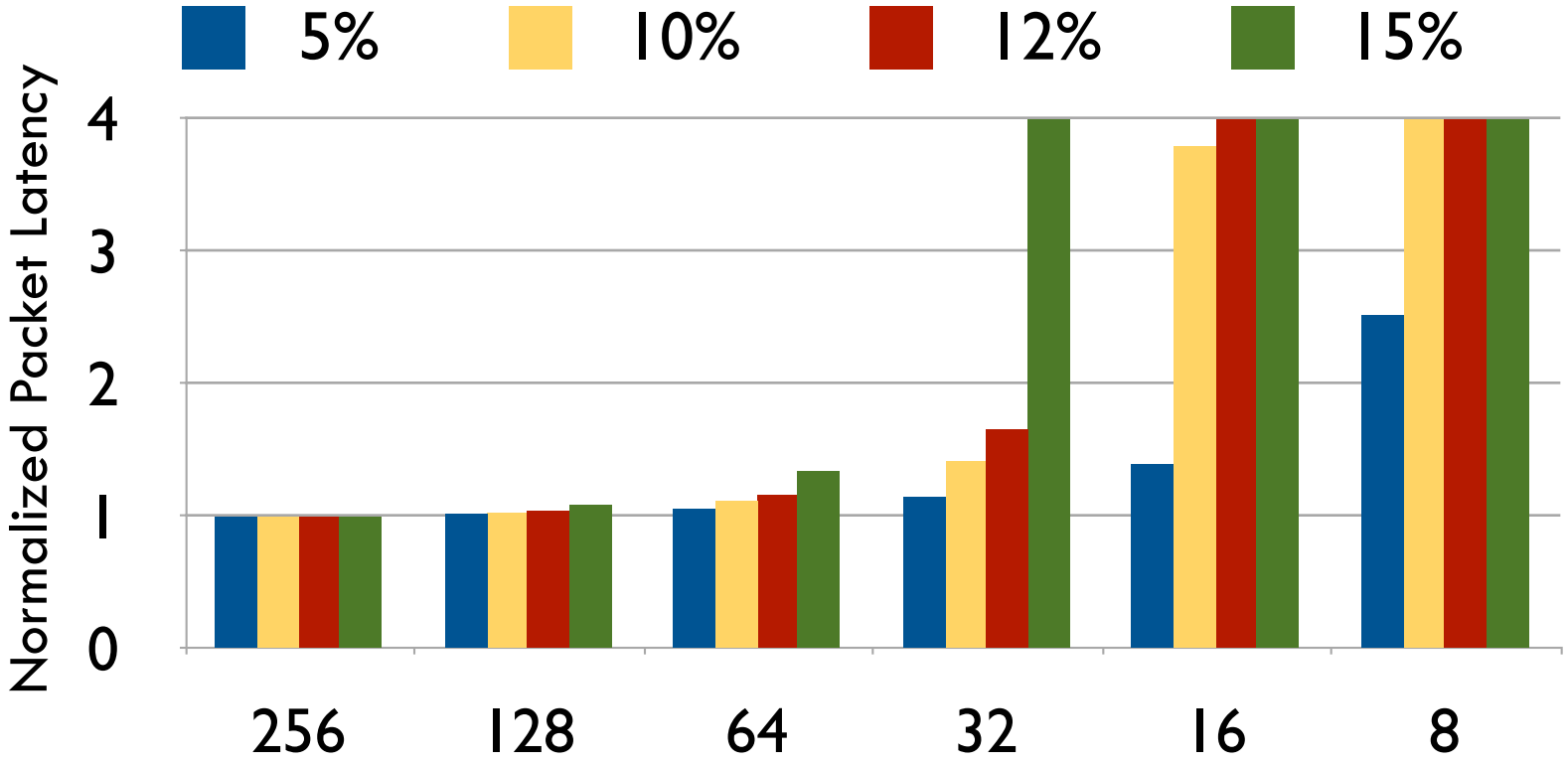
# Extraneous Invalidations with Coarse Vectors



- For many applications: number of sharers is small
  - 2 to 3
- When number of sharers exceeds *i*, pointers overflow
  - Directory entry operates in coarse mode
    - 1 bit represents multiple cores
- Imprecise sharing list
  - Extra processors will receive and acknowledge invalidation
  - Consumes network power
  - Requires additional cache lookups

# Latency Impact of Coarse Vectors

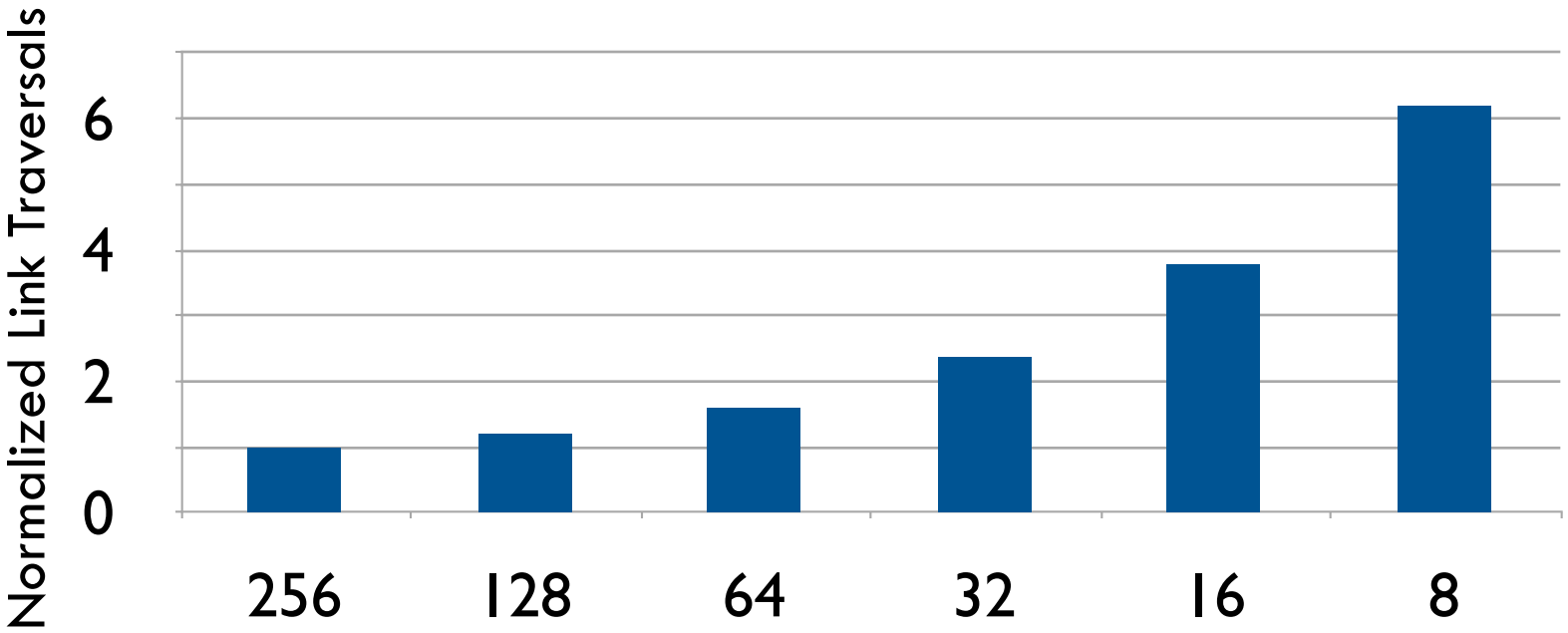





- Increase in coarseness leads to significant contention

# Bandwidth Impact

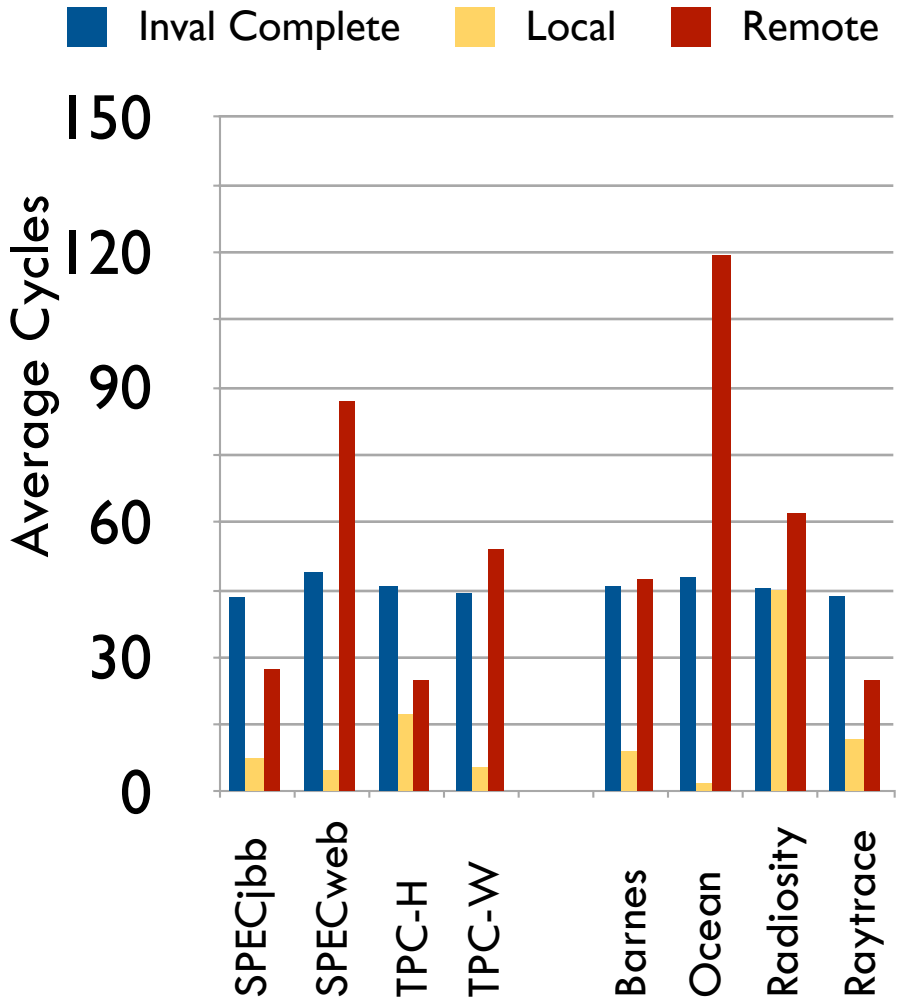





- Increased load results in decreased effectiveness of pipeline optimizations
- Increased dynamic power consumption

# System-level impact






- Coarse vectors increase average packet latency
- Increase completion time for invalidations
  - All acknowledgments must be received
- Delay subsequent requests to pending cache line

# SigNet Overview



- Coarseness reduces directory storage
  - But with significant potential impact on network
    - Due to extraneous invalidations
- Safely remove extraneous invalidations
  - Save power and reduce network contention
- Place cache summary information in routers
  - Counting Bloom filters used for cache summary signatures
  - Use summary information to filter network packets

# SigNet Bloom Filters



- Counting Bloom filters summarize cache information in routers
  - Signature of cache contents
- Bloom Filter Hit
  - Core exists between current node and destination that is caching an address mapping to same entry
- Bloom Filter Miss
  - None of the downstream caches are caching lines that map to this entry

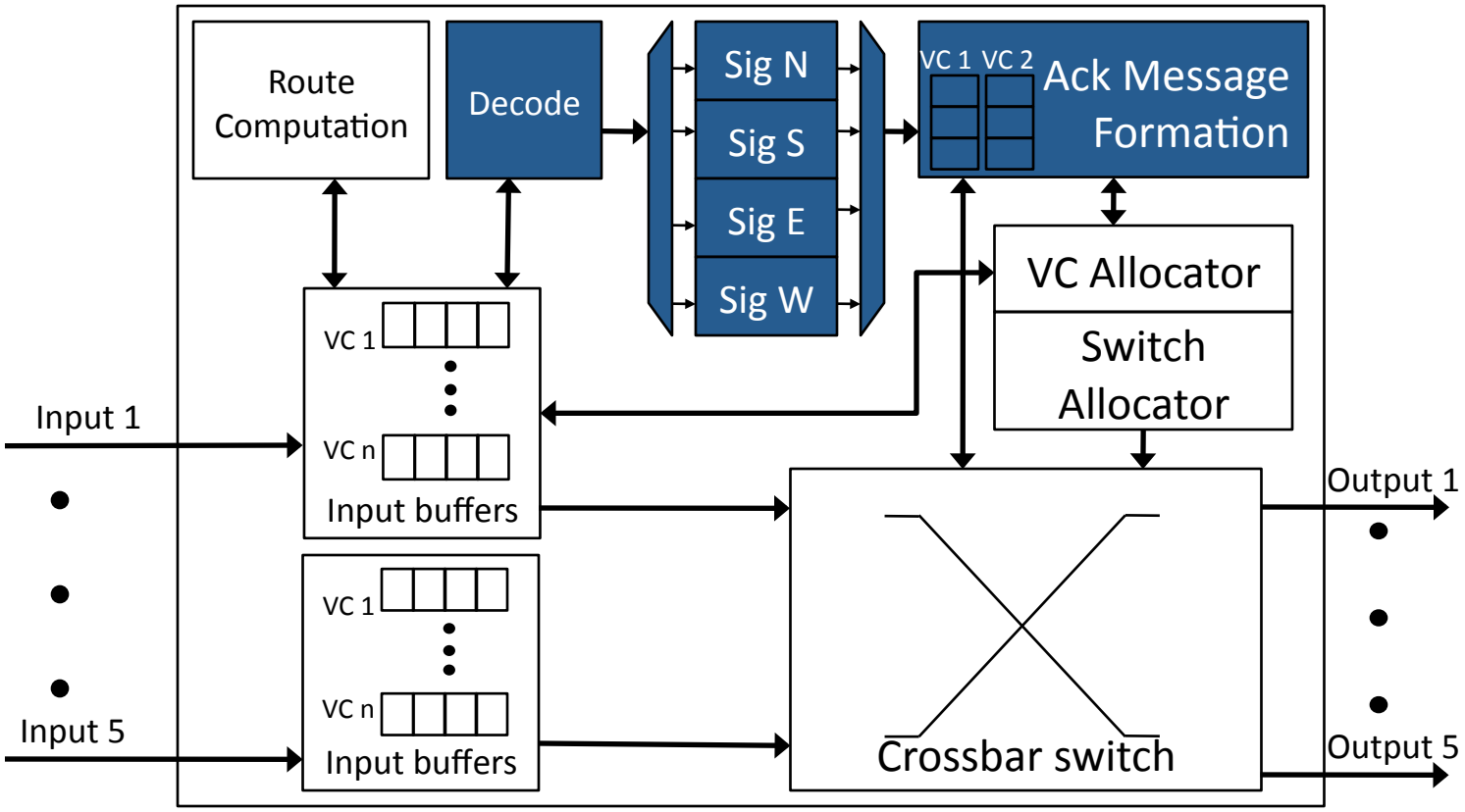
# Modifying Bloom Filters



- Bloom Filter Insertion
  - Cache misses increment counter as they travel to directory
- Bloom Filter Deletion
  - Writeback and invalidation acknowledgments decrement associated counter at routers between cache and directory

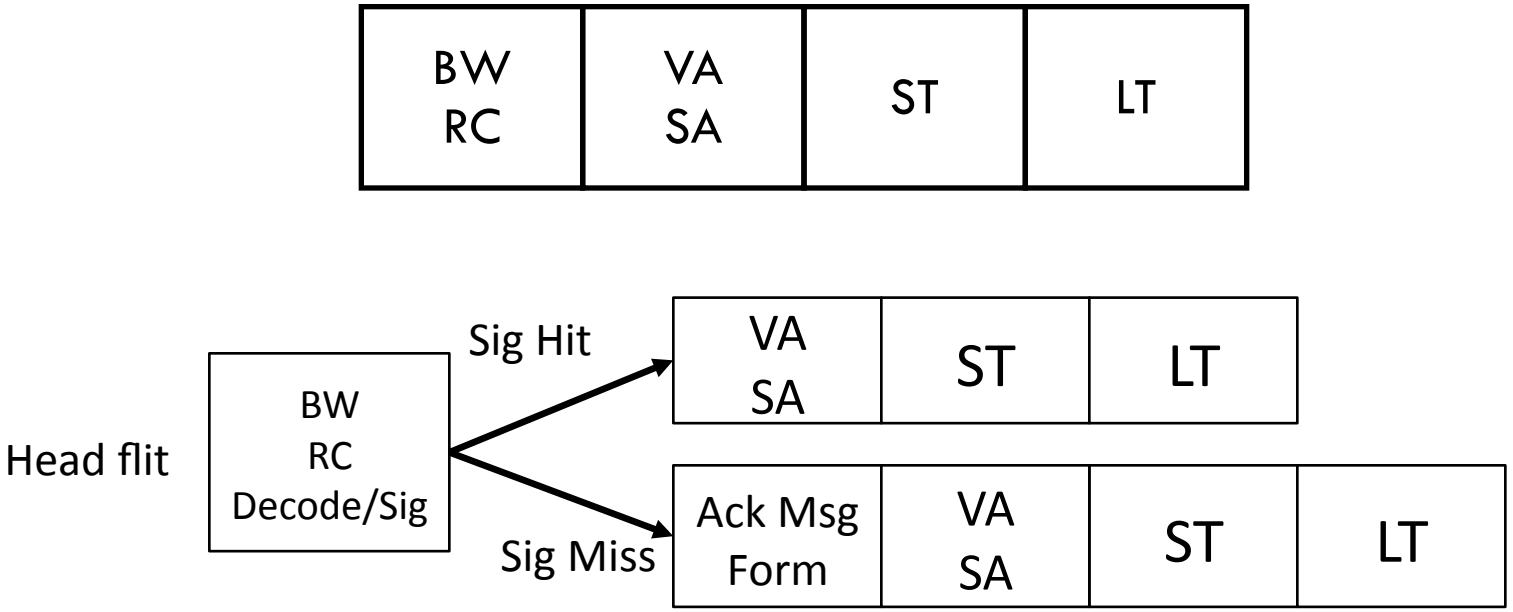
# SigNet Architecture





# SigNet Pipeline

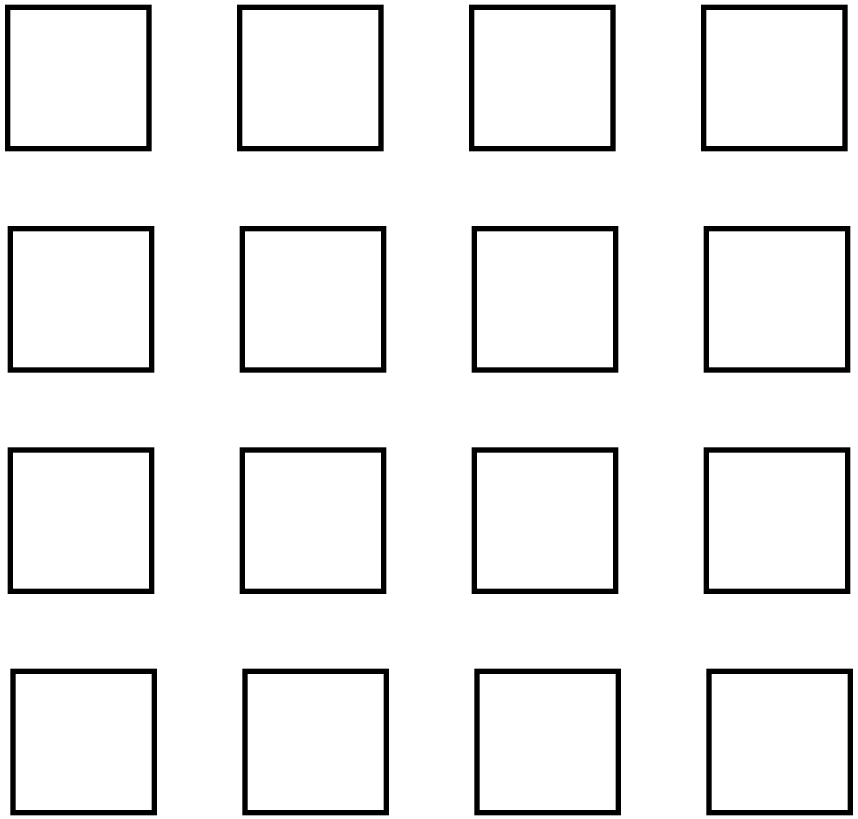





- Header flits traverse modified pipeline
  - If the packet needs to check/update signature

# SigNet Example



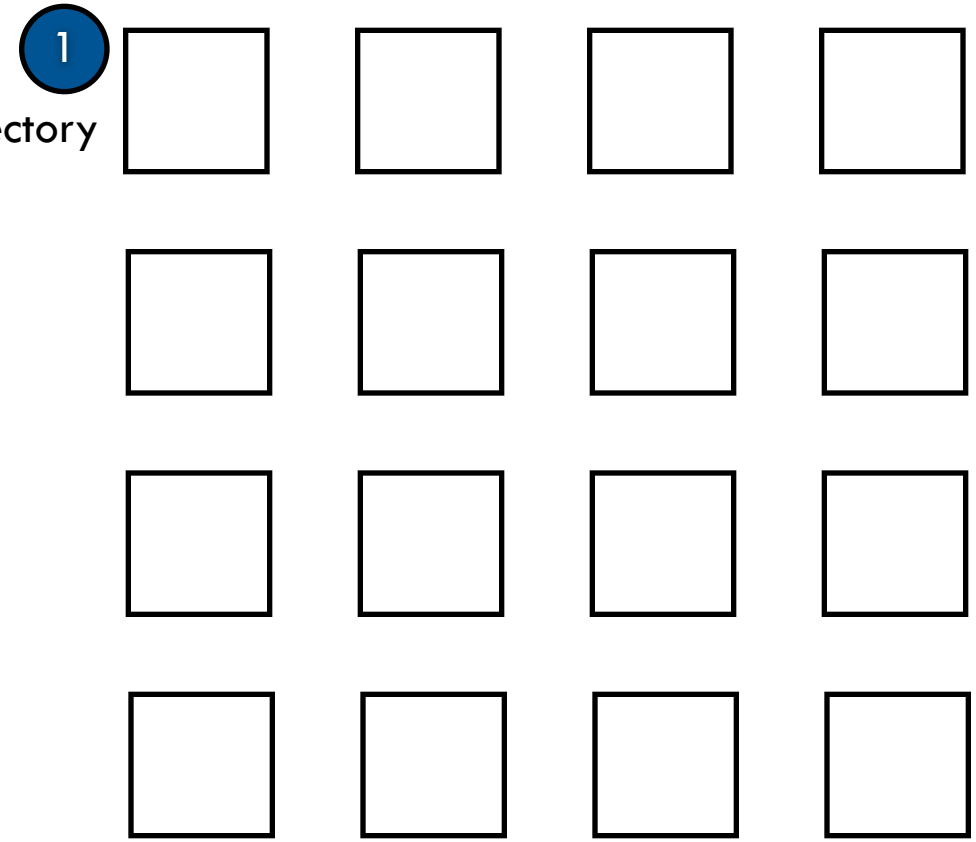


- Requests follow X-Y path
- Responses follow Y-X path

# SigNet Example



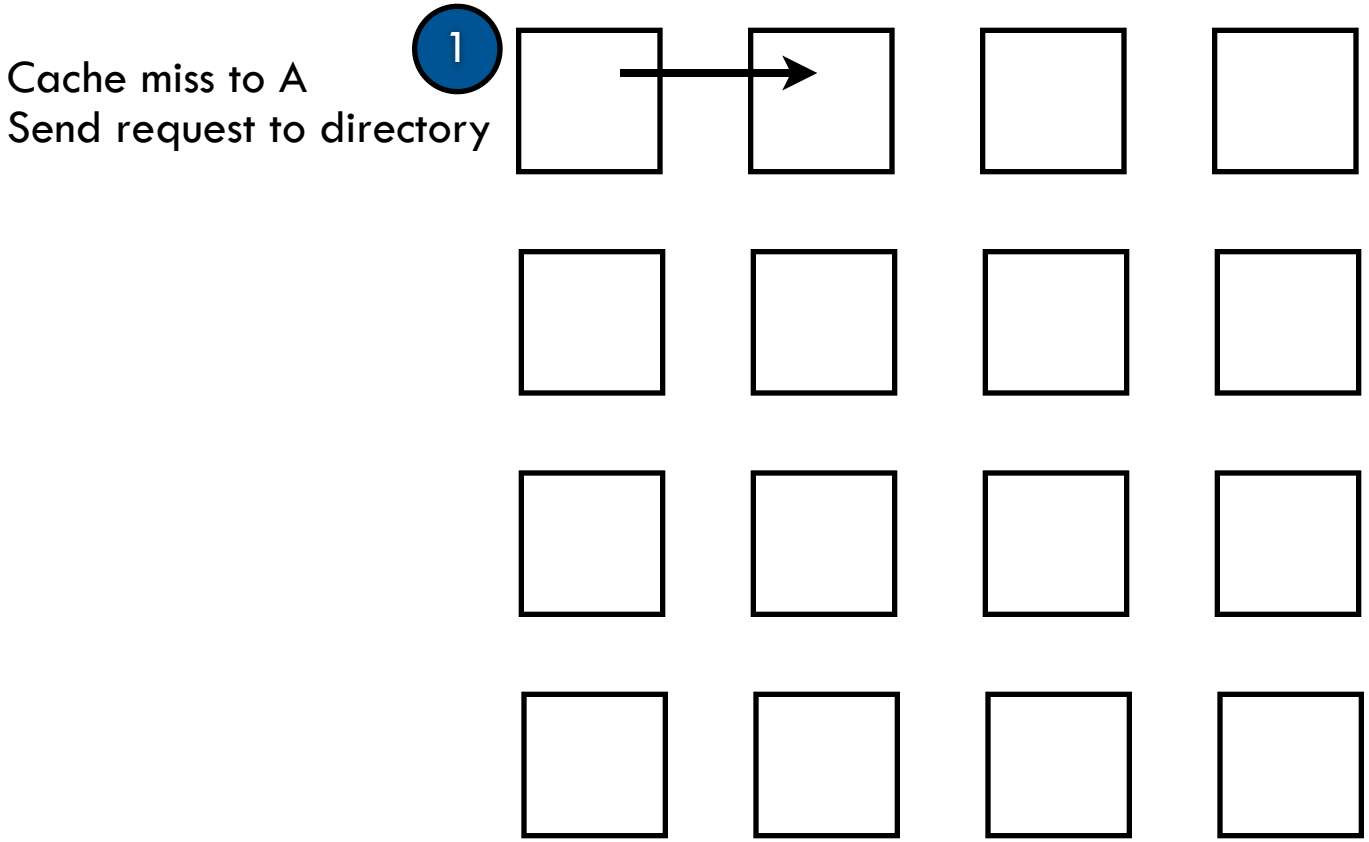
Cache miss to A
Send request to directory




- Requests follow X-Y path
- Responses follow Y-X path

# SigNet Example

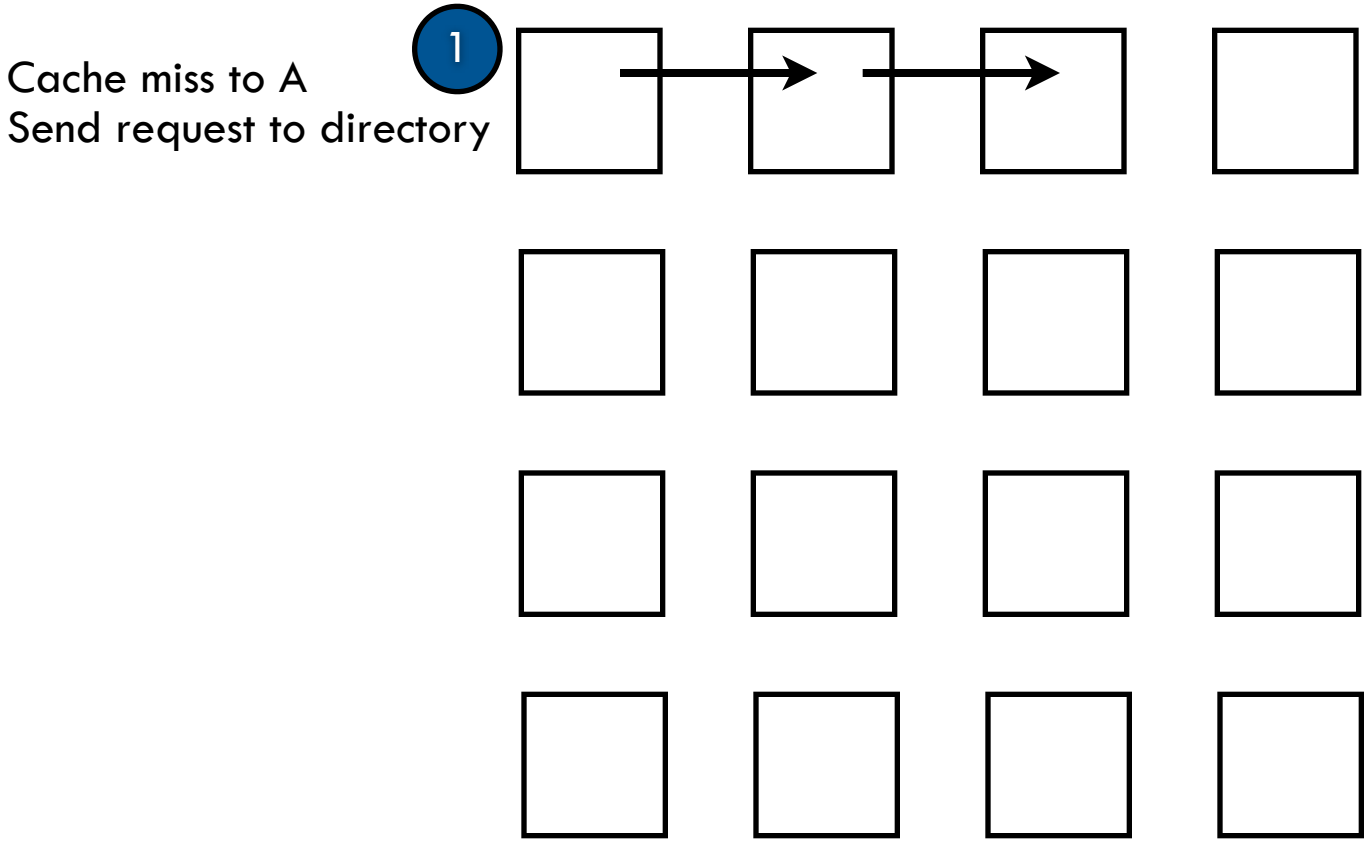





- Requests follow X-Y path
- Responses follow Y-X path

# SigNet Example

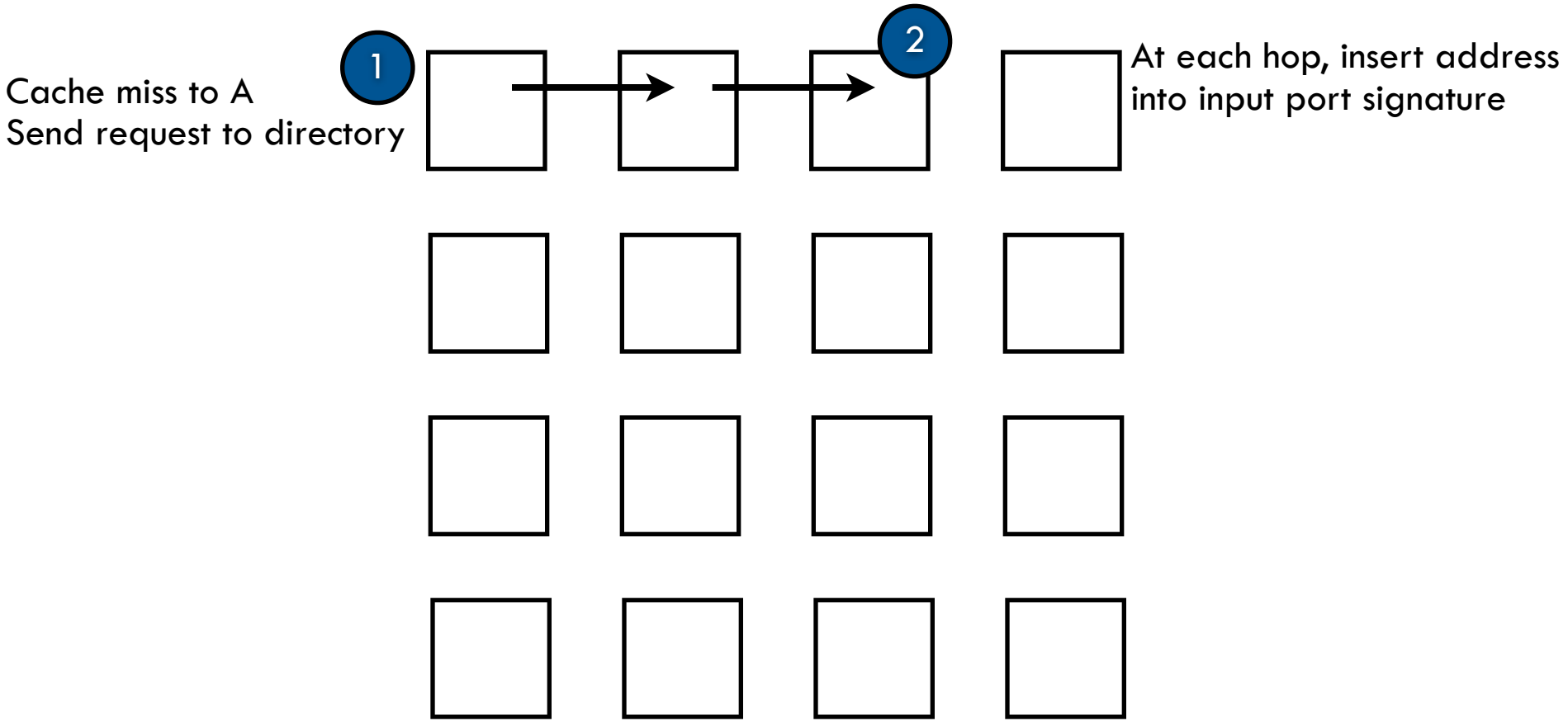





- Requests follow X-Y path
- Responses follow Y-X path

# SigNet Example

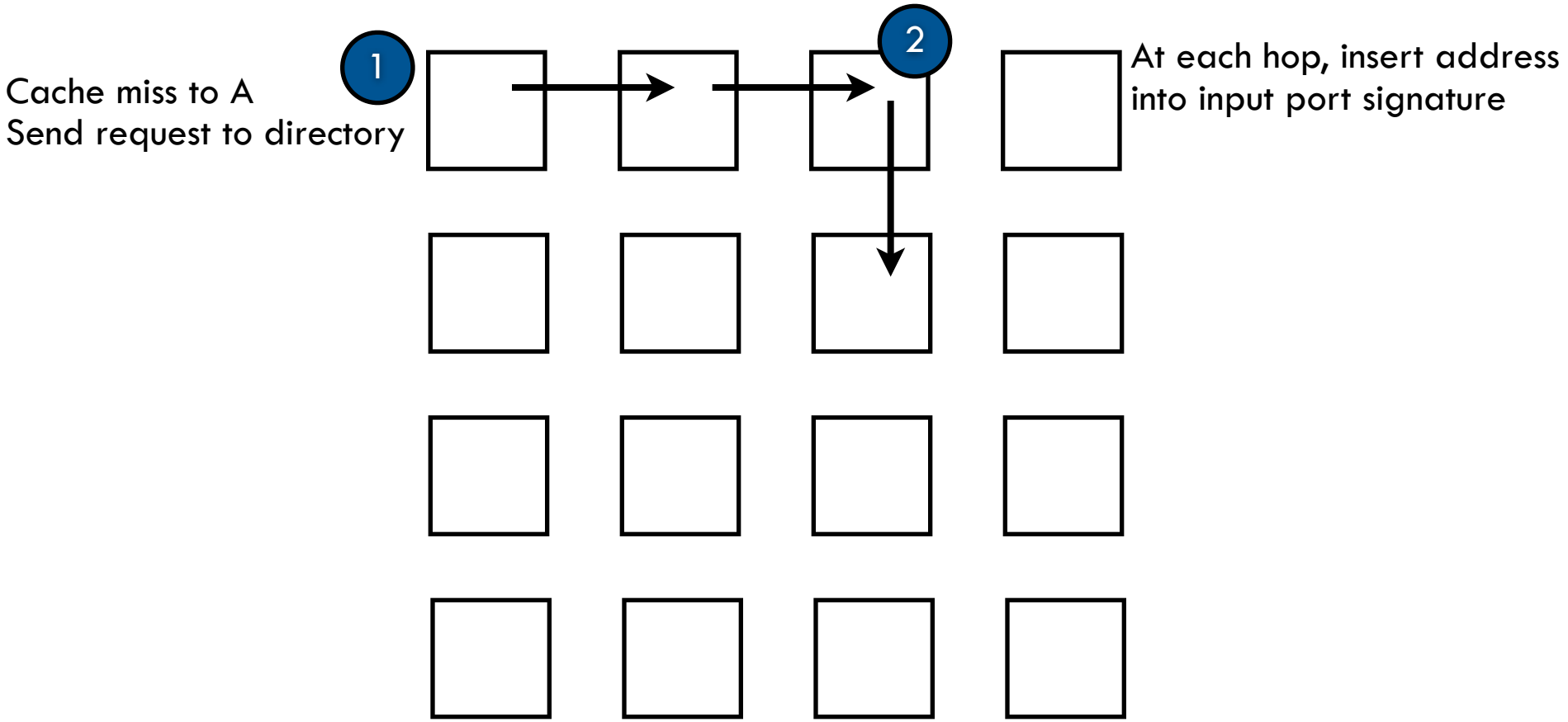





- Requests follow X-Y path
- Responses follow Y-X path

# SigNet Example

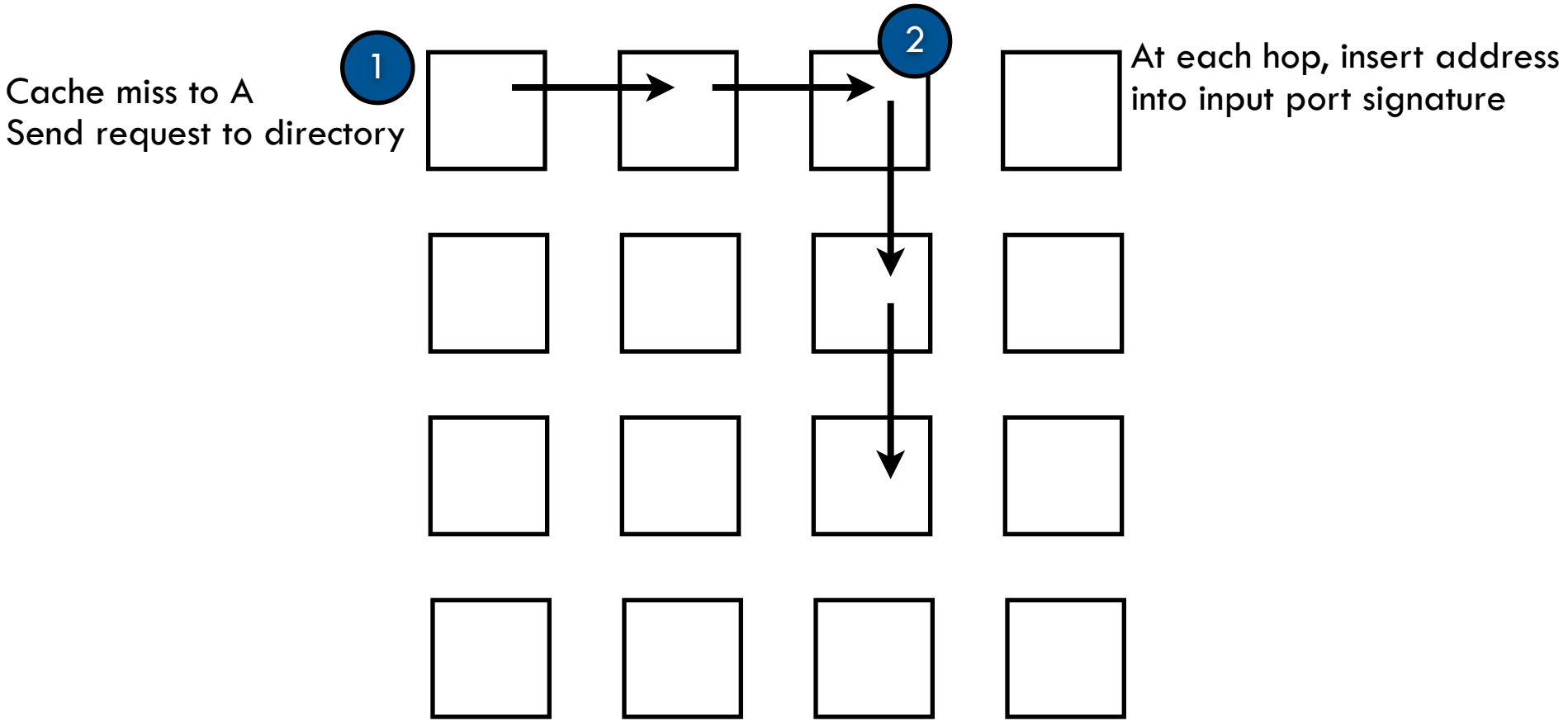





- Requests follow X-Y path
- Responses follow Y-X path

# SigNet Example

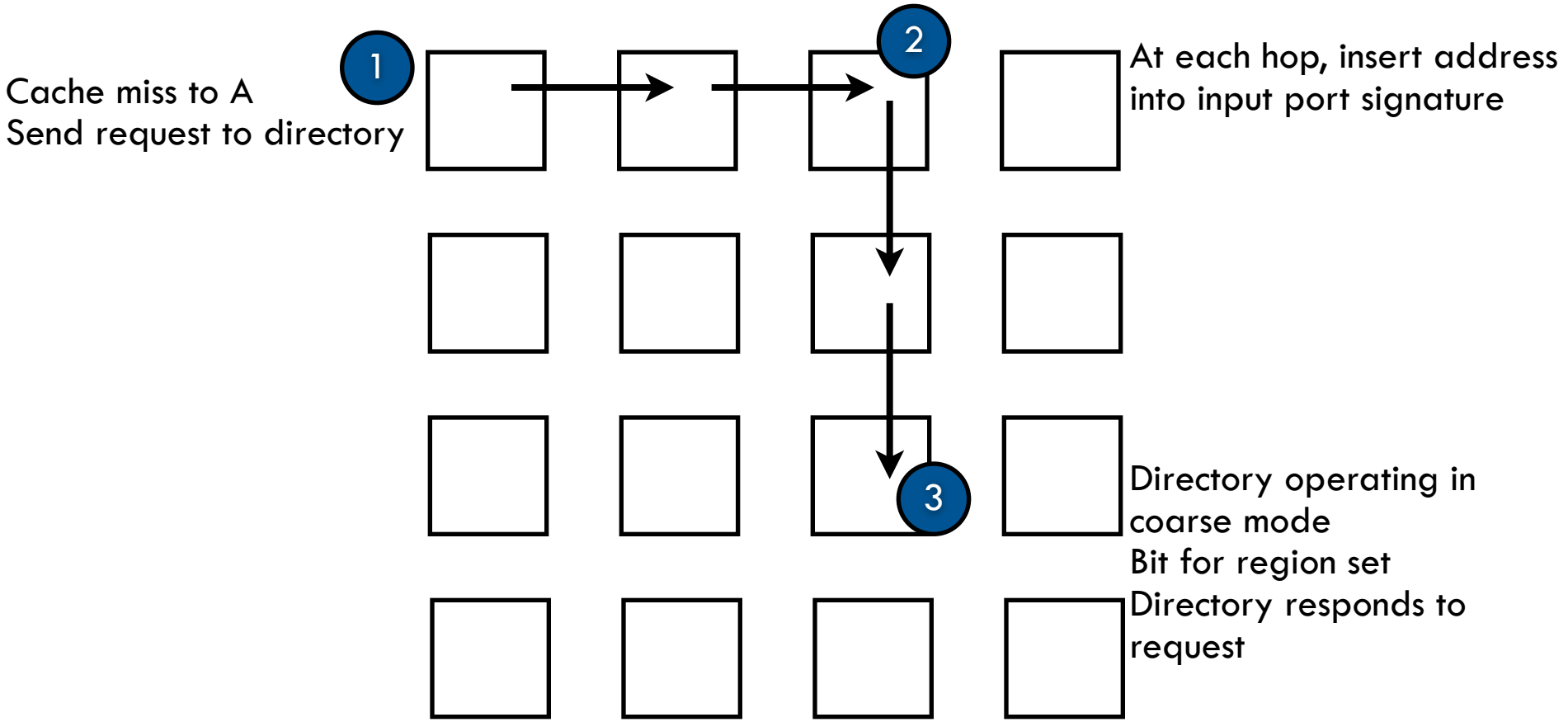





- Requests follow X-Y path
- Responses follow Y-X path

# SigNet Example

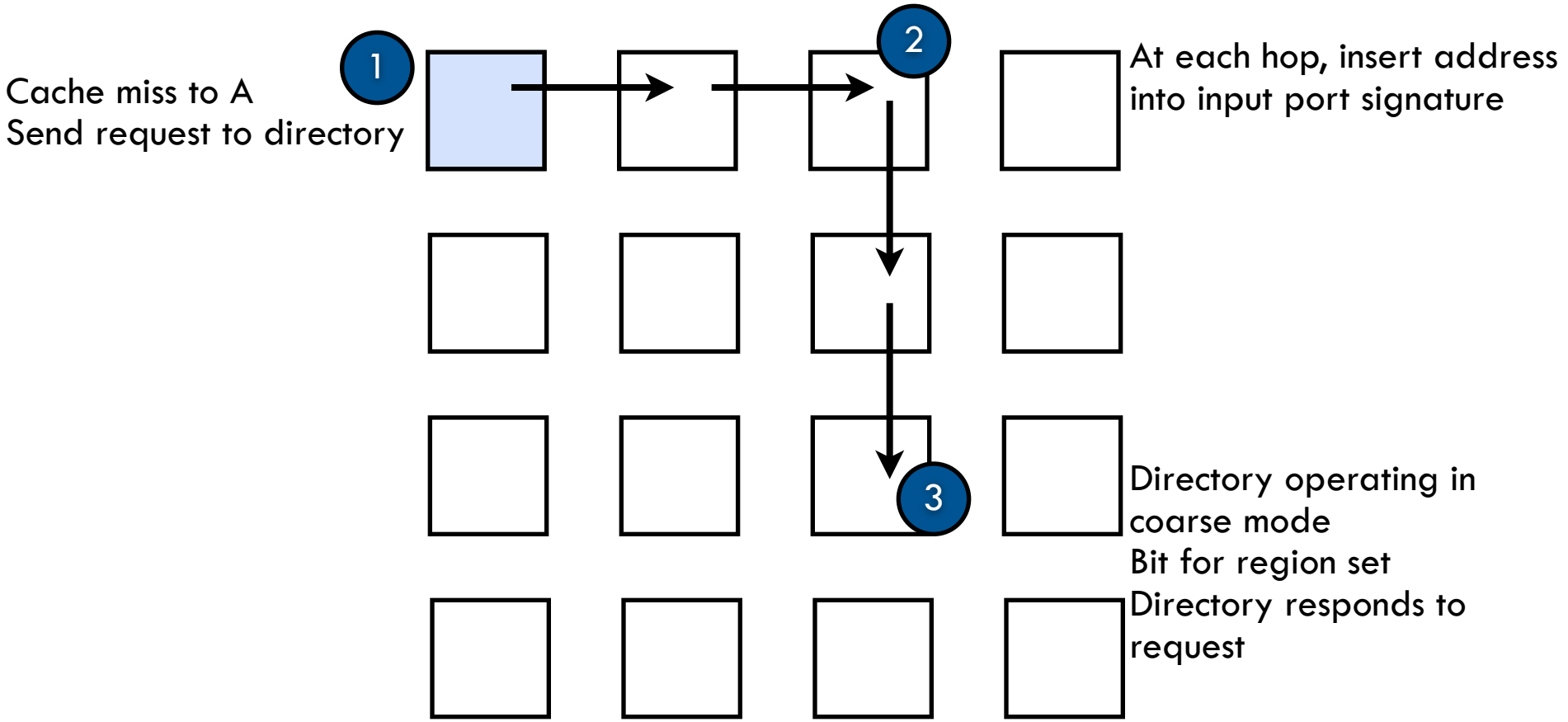





- Requests follow X-Y path
- Responses follow Y-X path

# SigNet Example

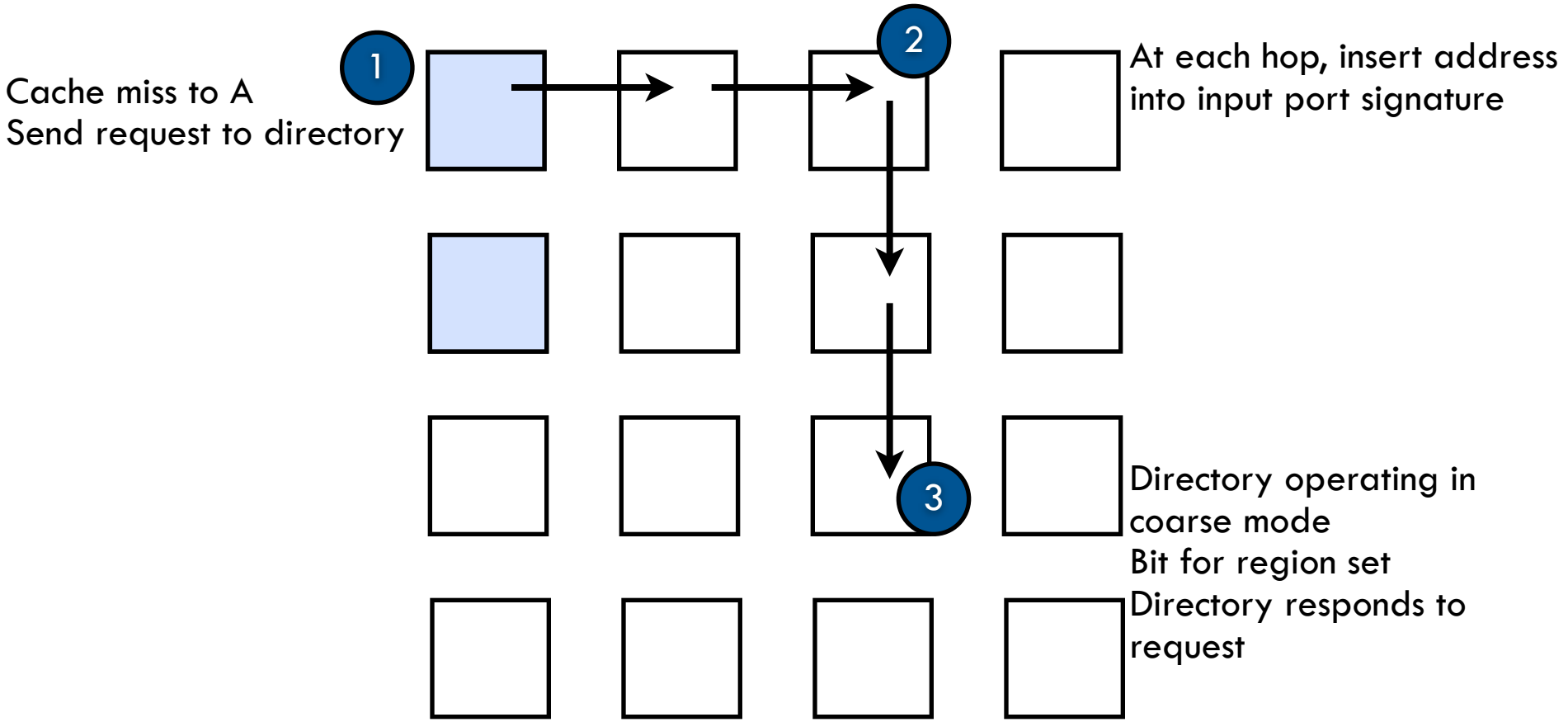





- Requests follow X-Y path
- Responses follow Y-X path

# SigNet Example

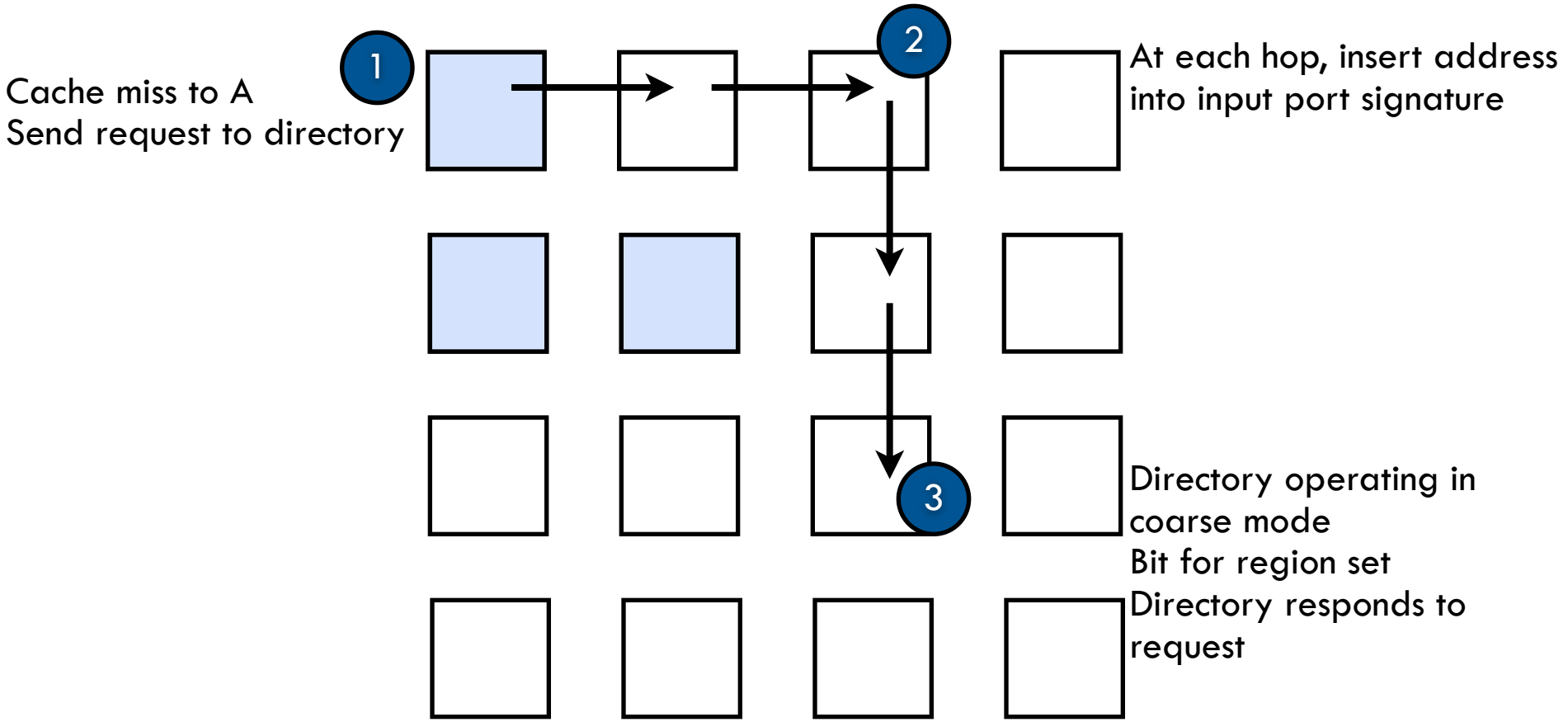





- Requests follow X-Y path
- Responses follow Y-X path

# SigNet Example

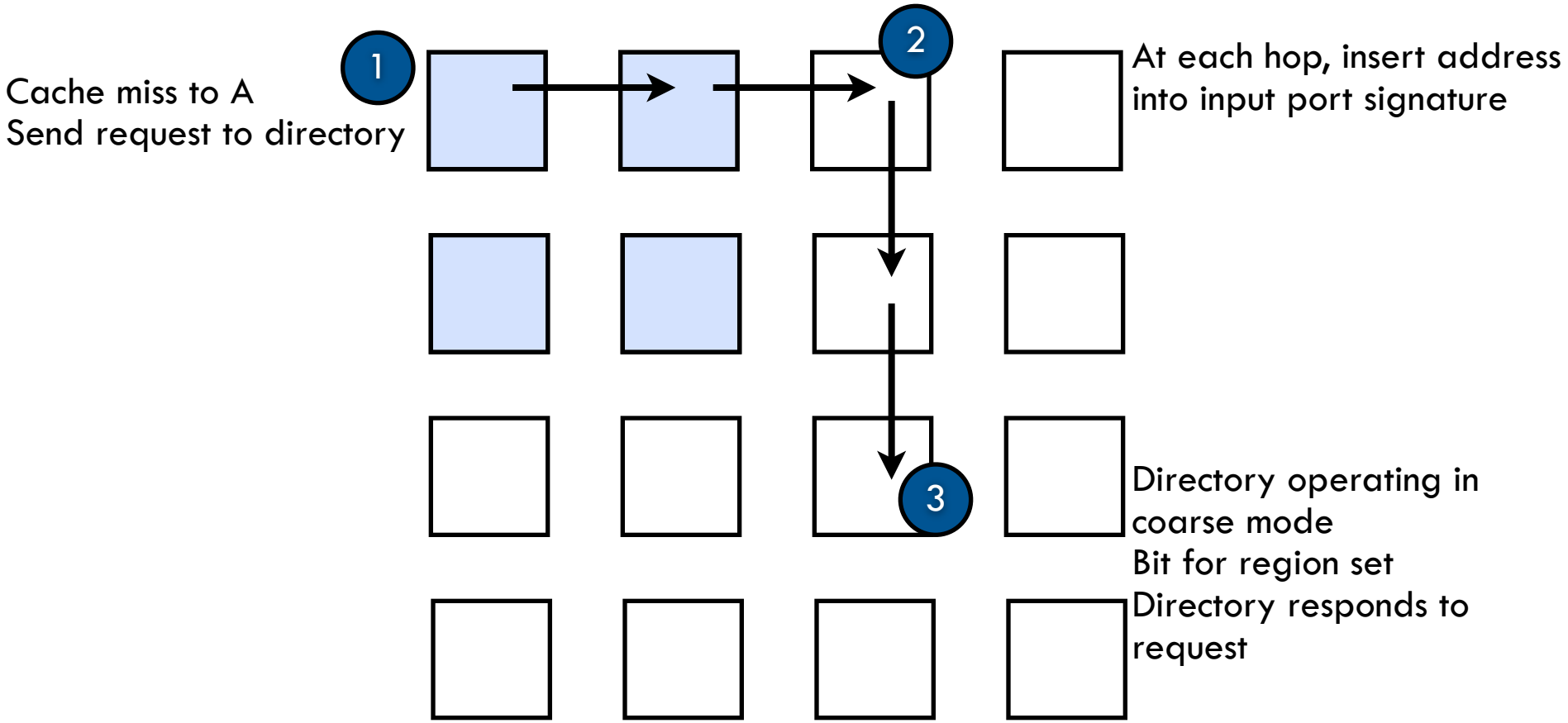





- Requests follow X-Y path
- Responses follow Y-X path

# SigNet Example

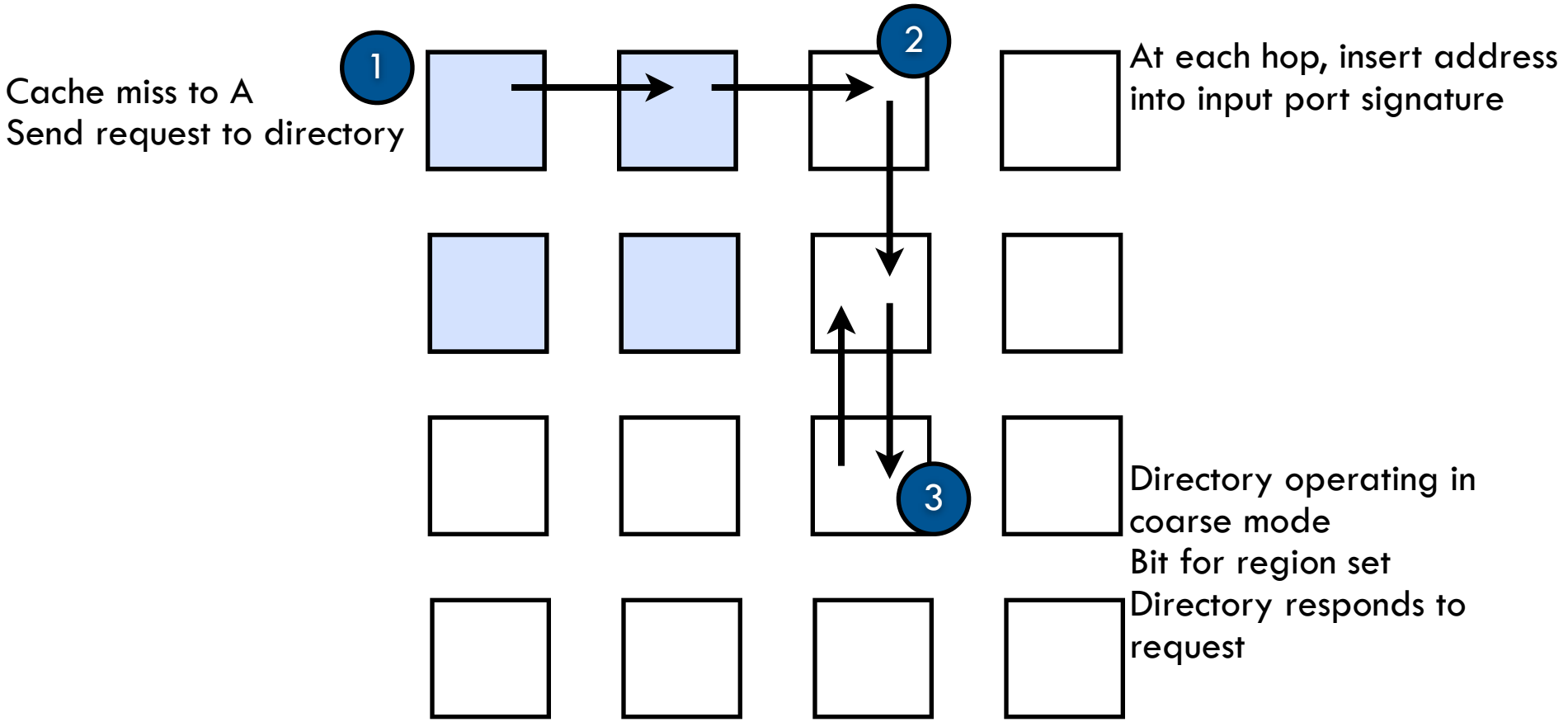





- Requests follow X-Y path
- Responses follow Y-X path

# SigNet Example

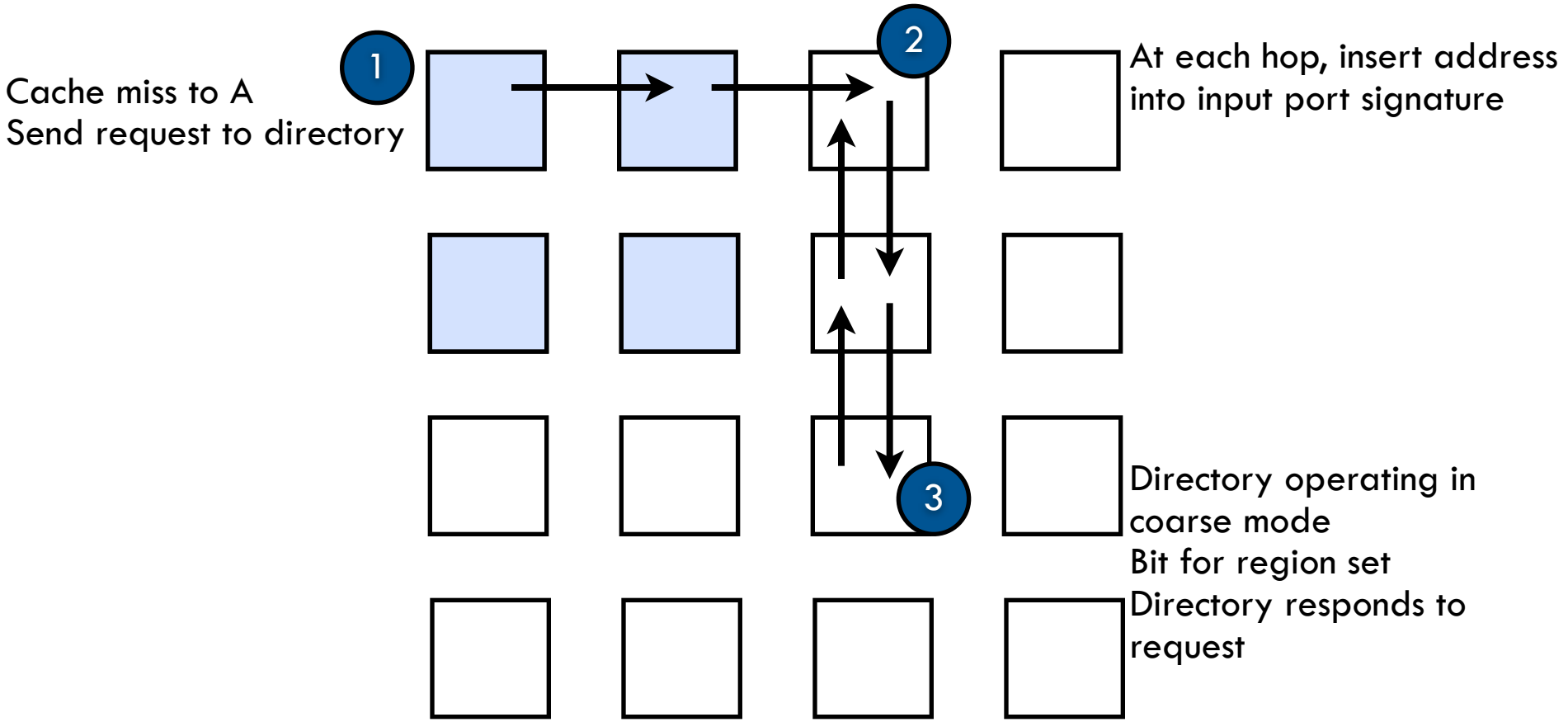





- Requests follow X-Y path
- Responses follow Y-X path

# SigNet Example

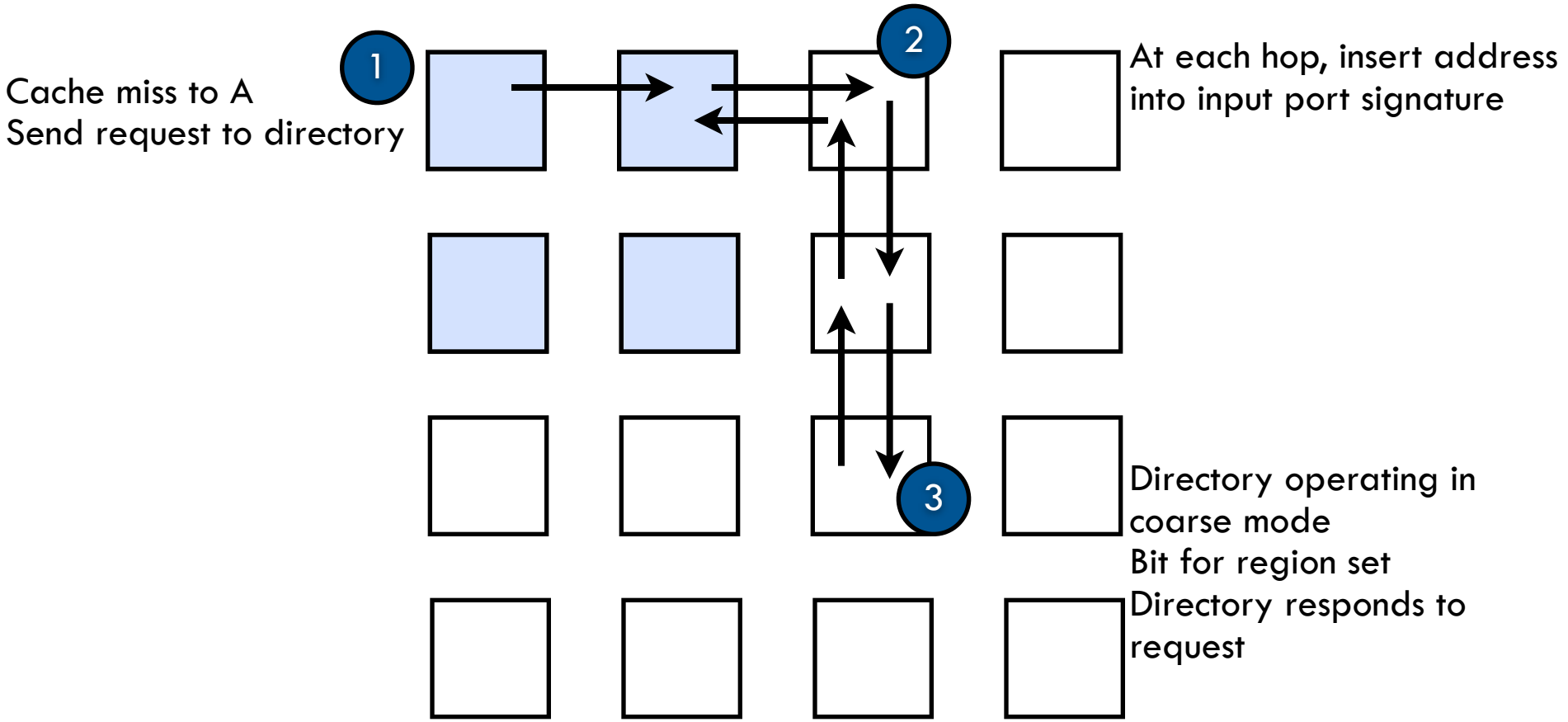





- Requests follow X-Y path
- Responses follow Y-X path

# SigNet Example

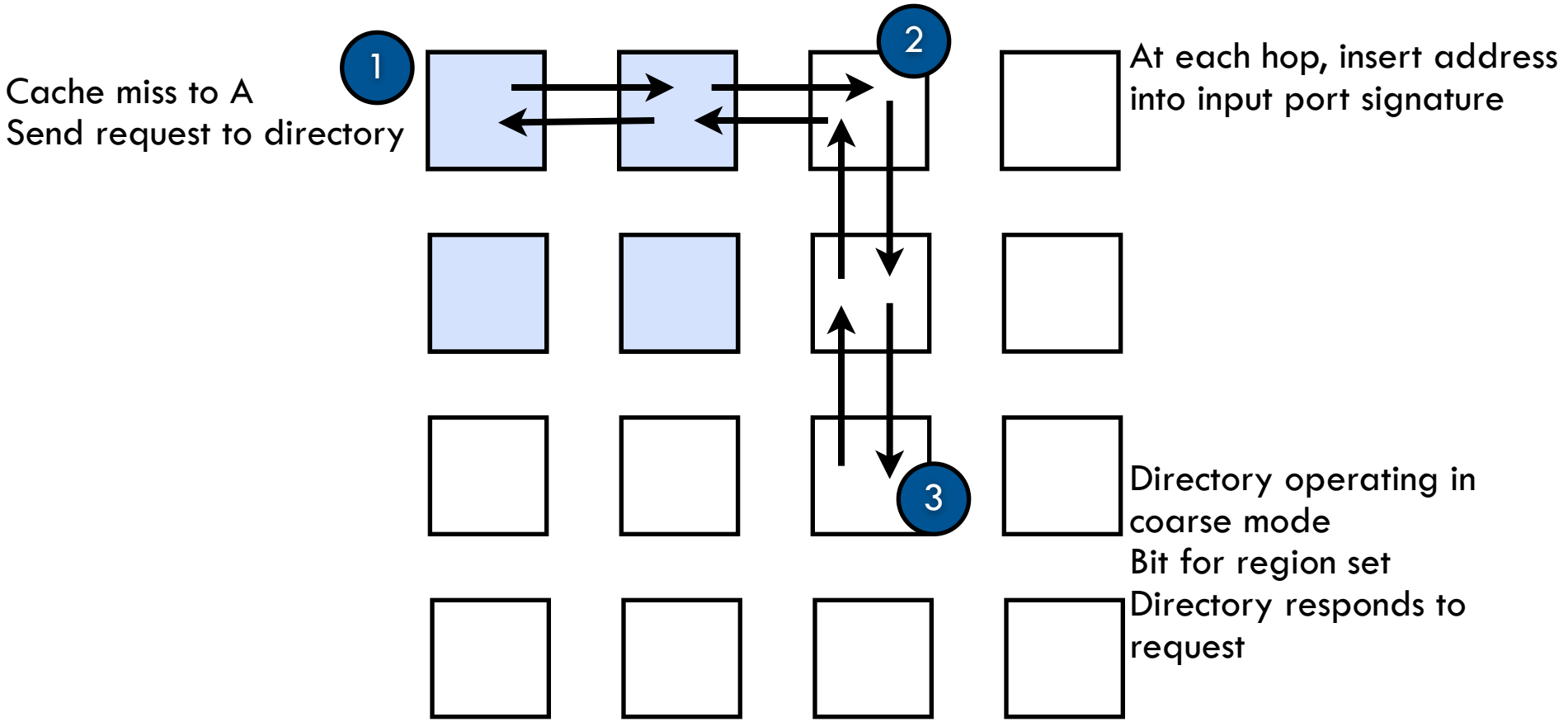





- Requests follow X-Y path
- Responses follow Y-X path

# SigNet Example

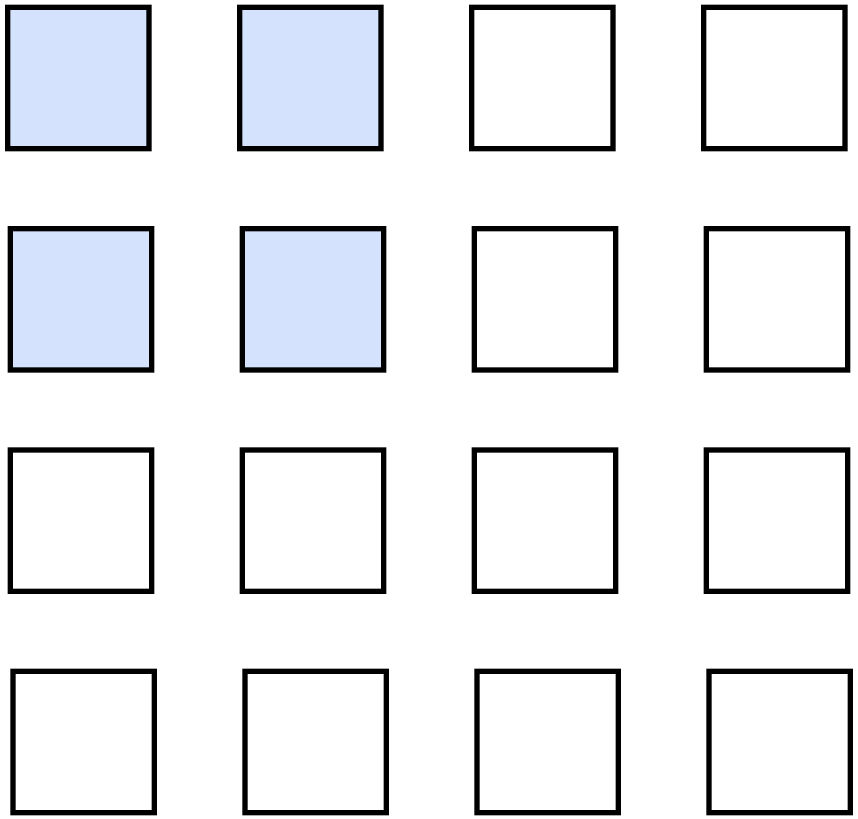





- Requests follow X-Y path
- Responses follow Y-X path

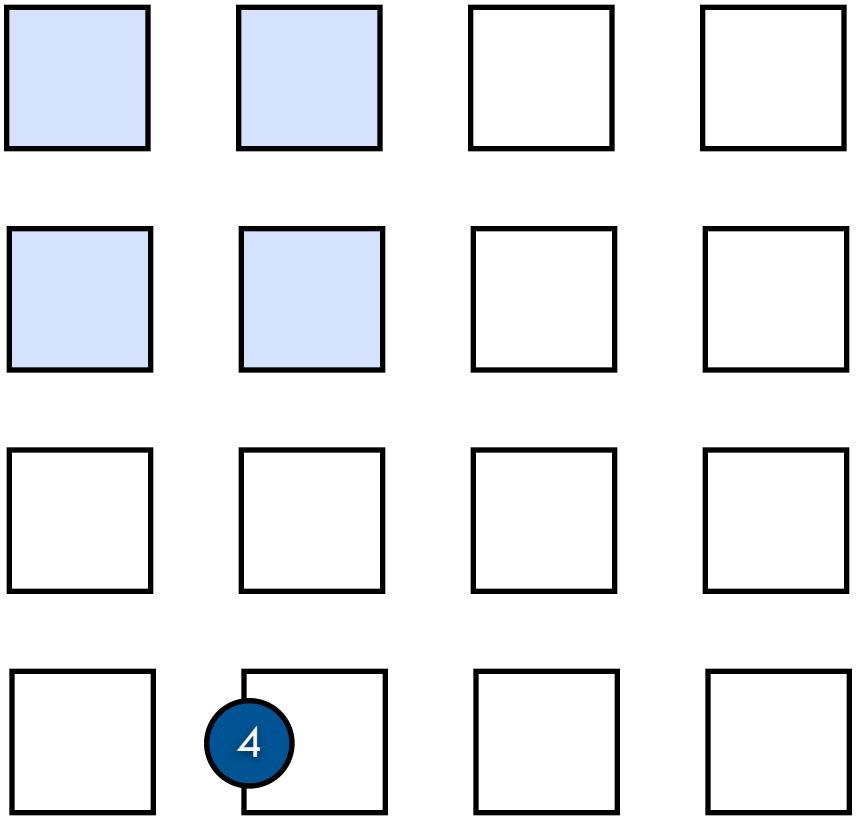
# SigNet Example (2)





# SigNet Example (2)

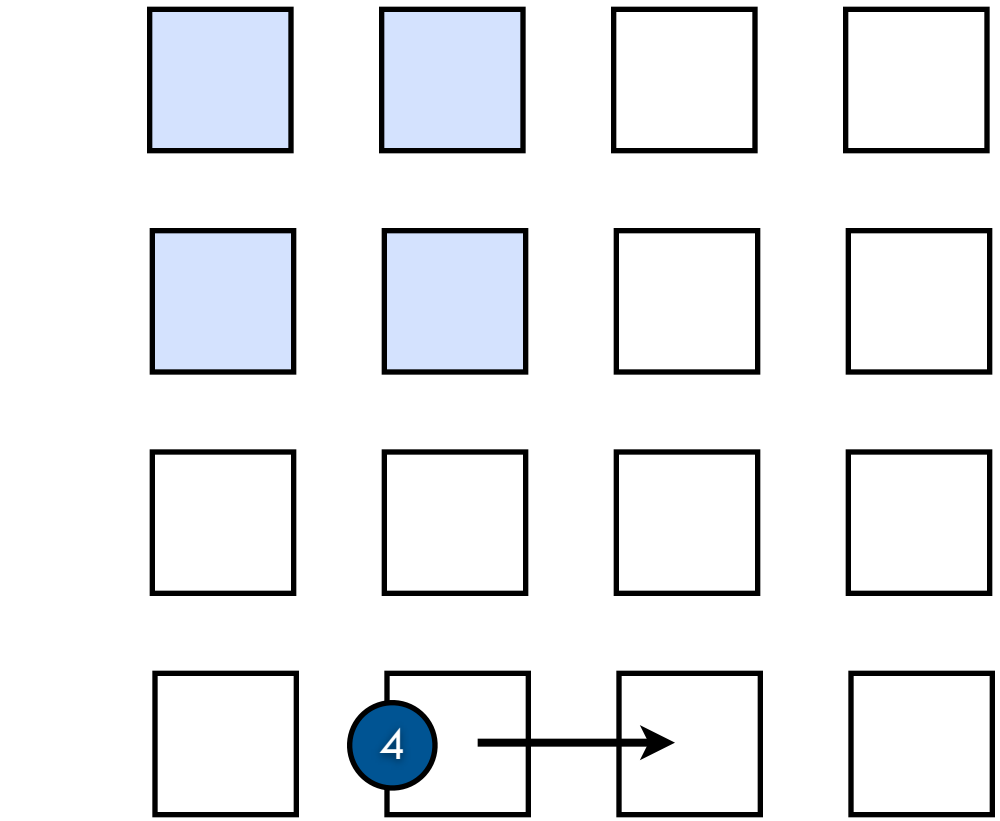





Write request for A

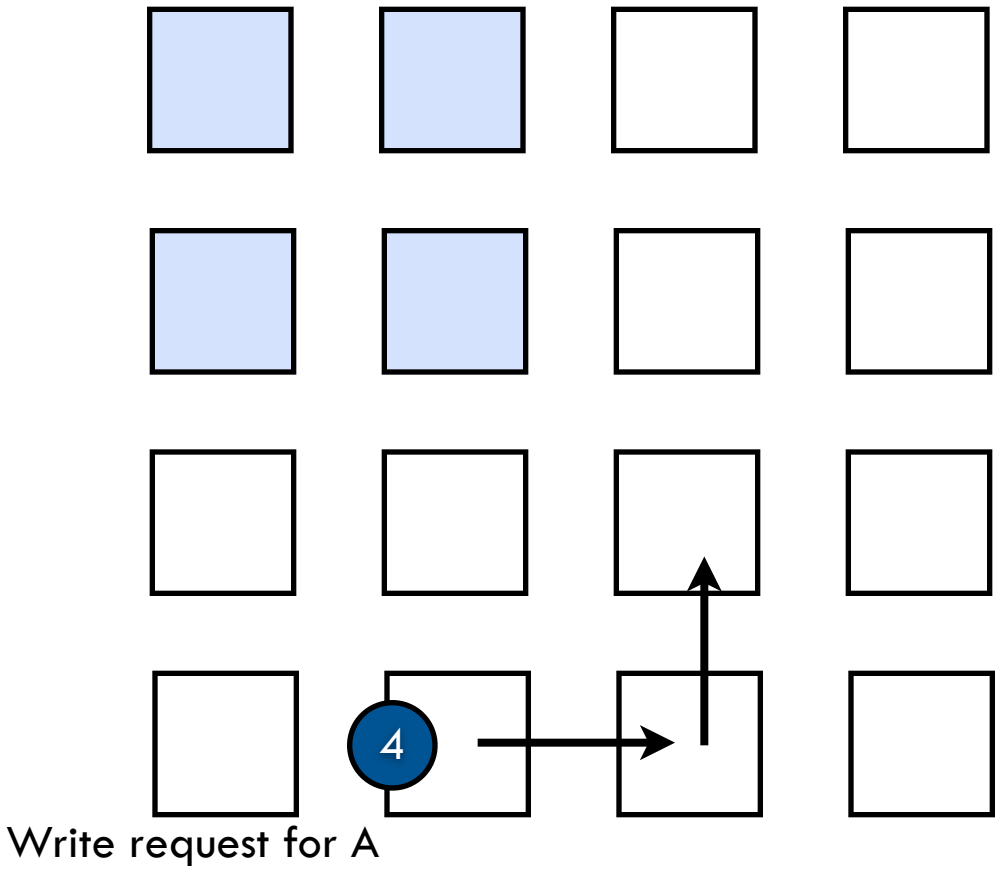
# SigNet Example (2)





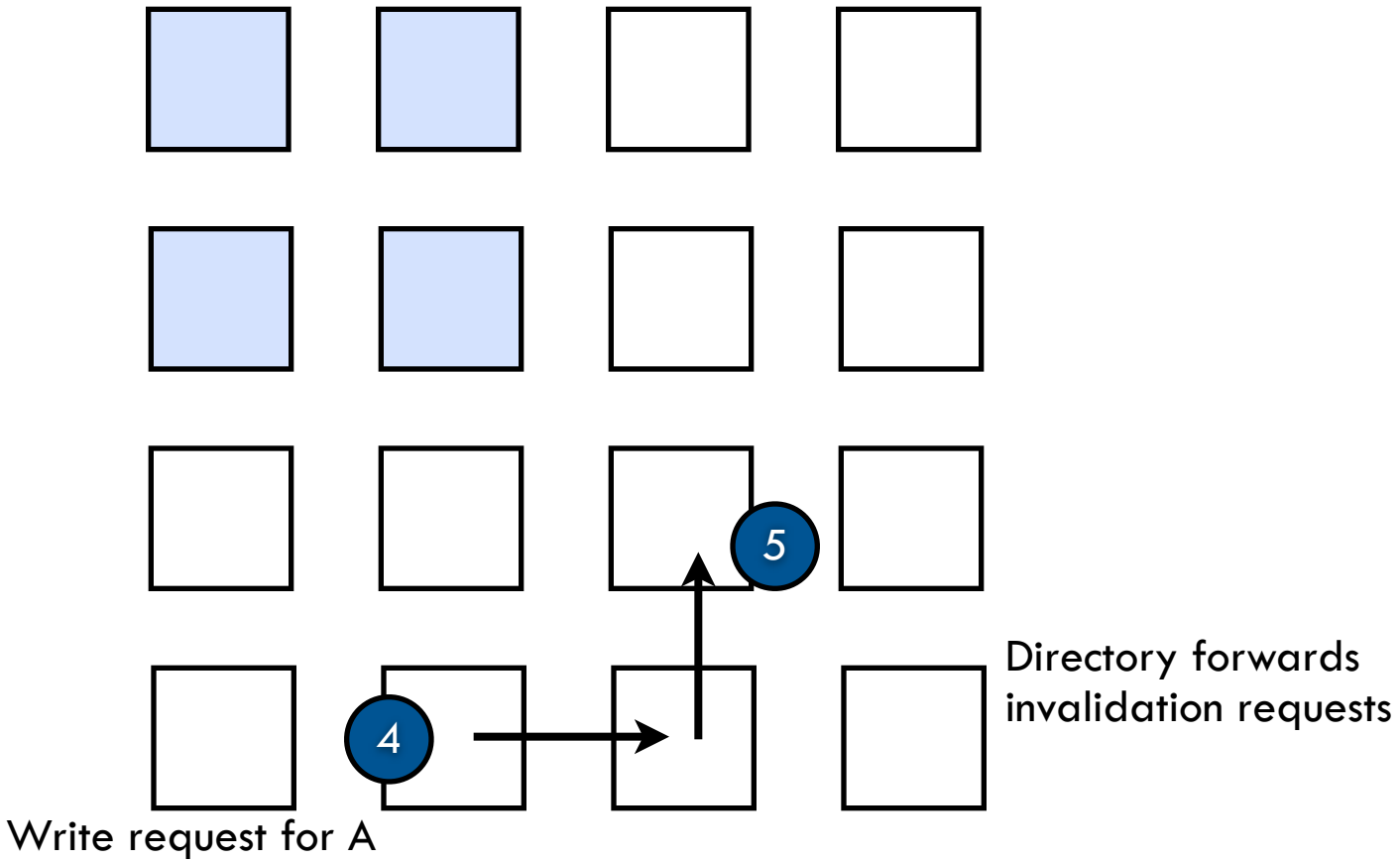
# SigNet Example (2)





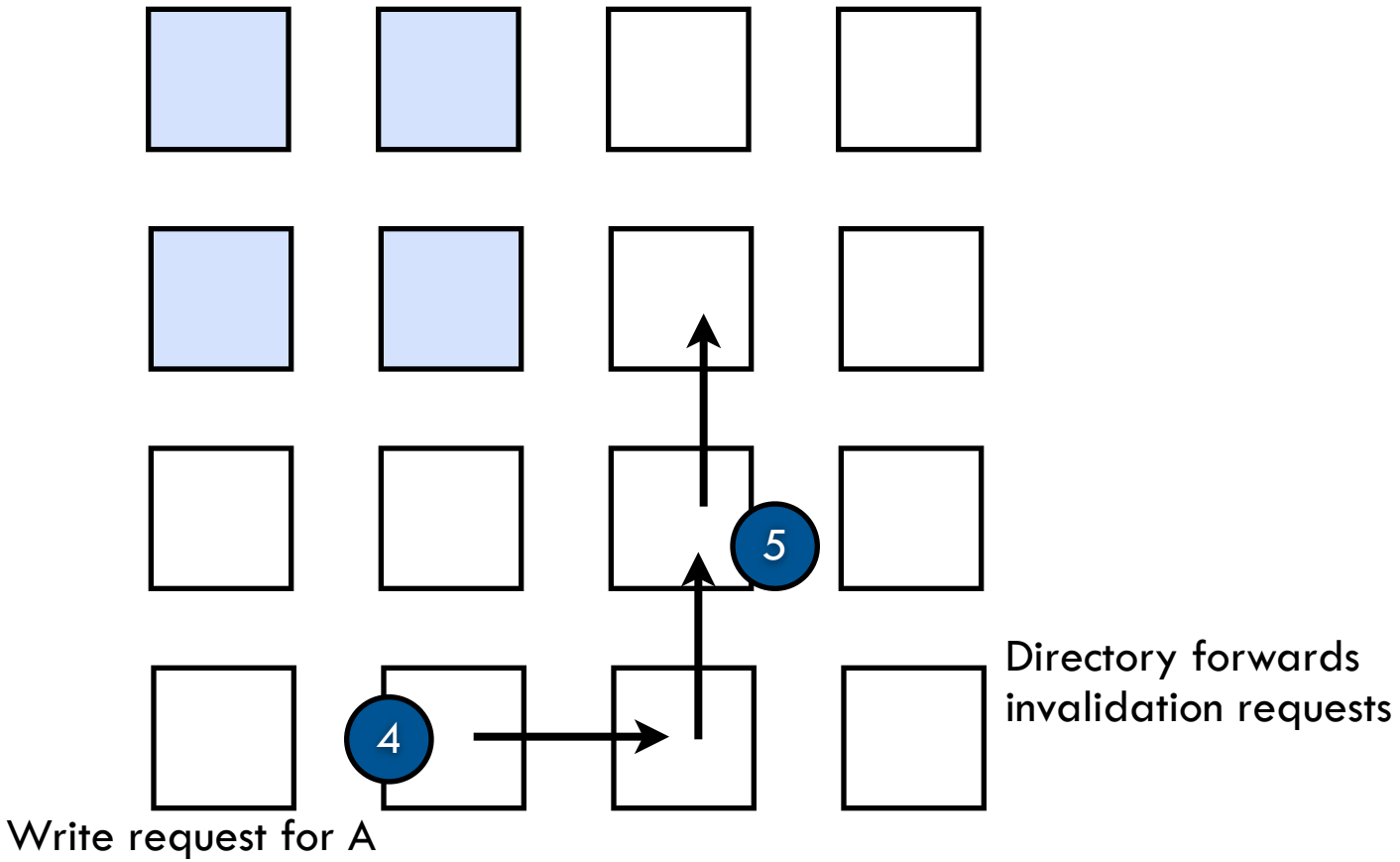
# SigNet Example (2)





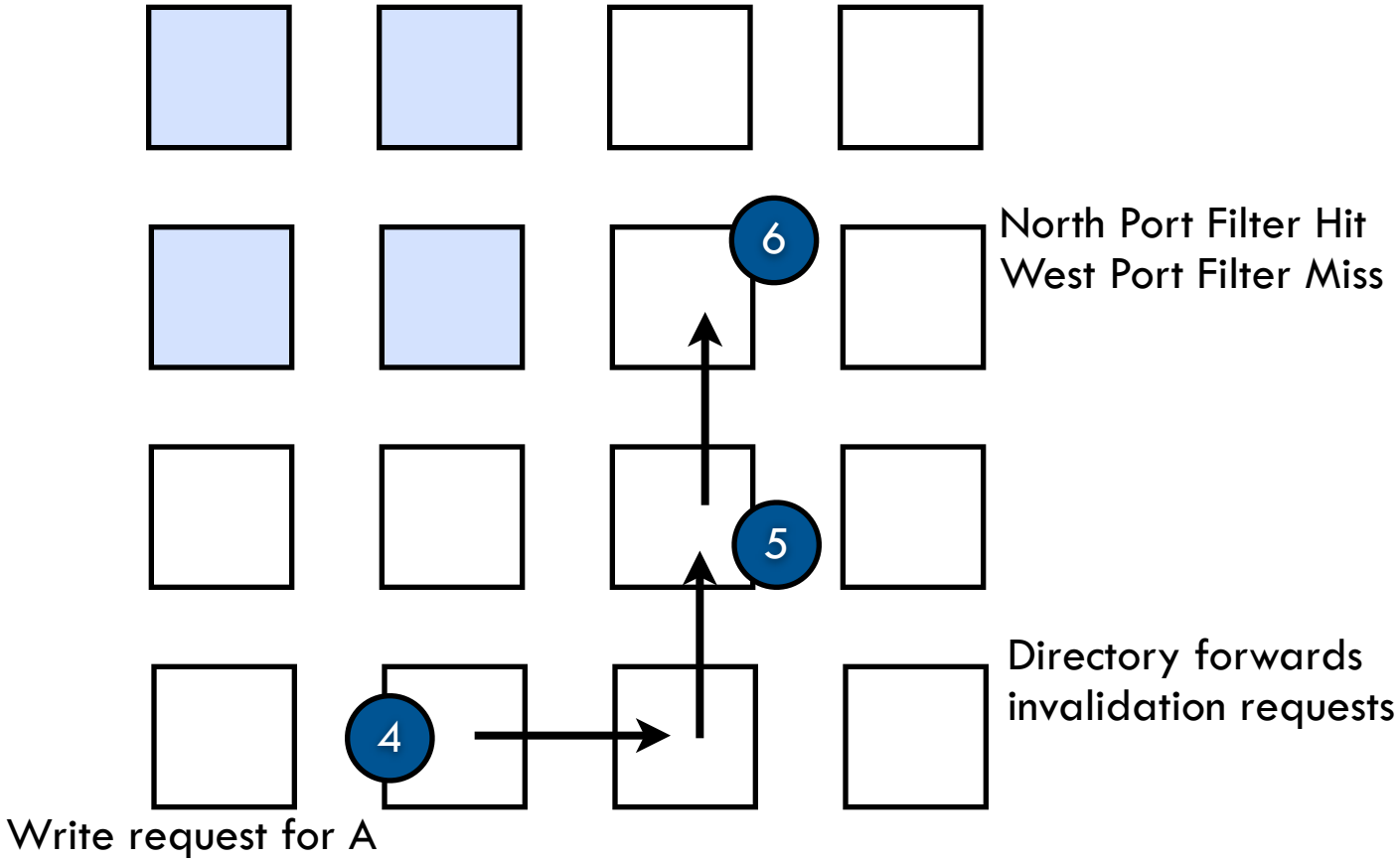
# SigNet Example (2)





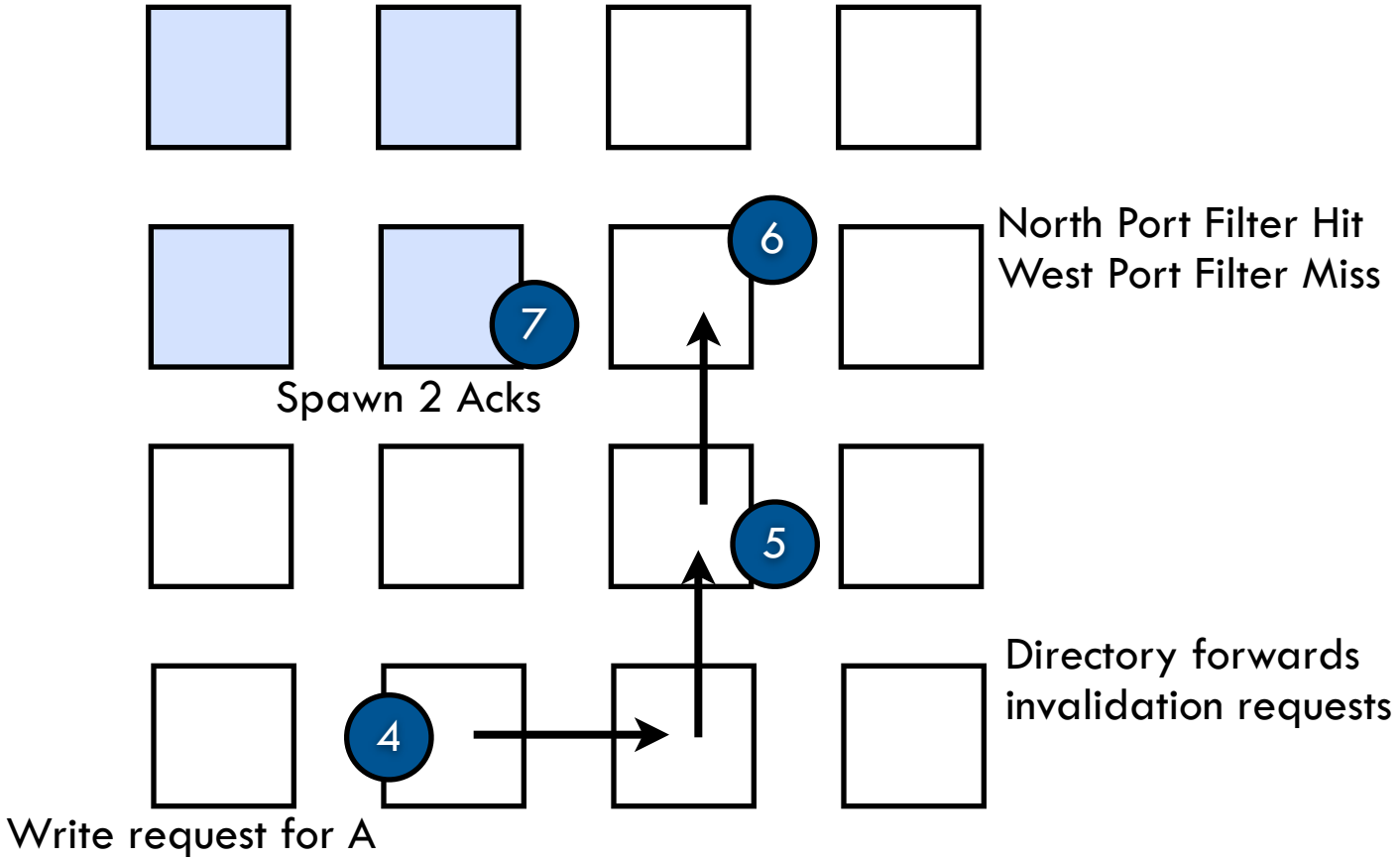
# SigNet Example (2)





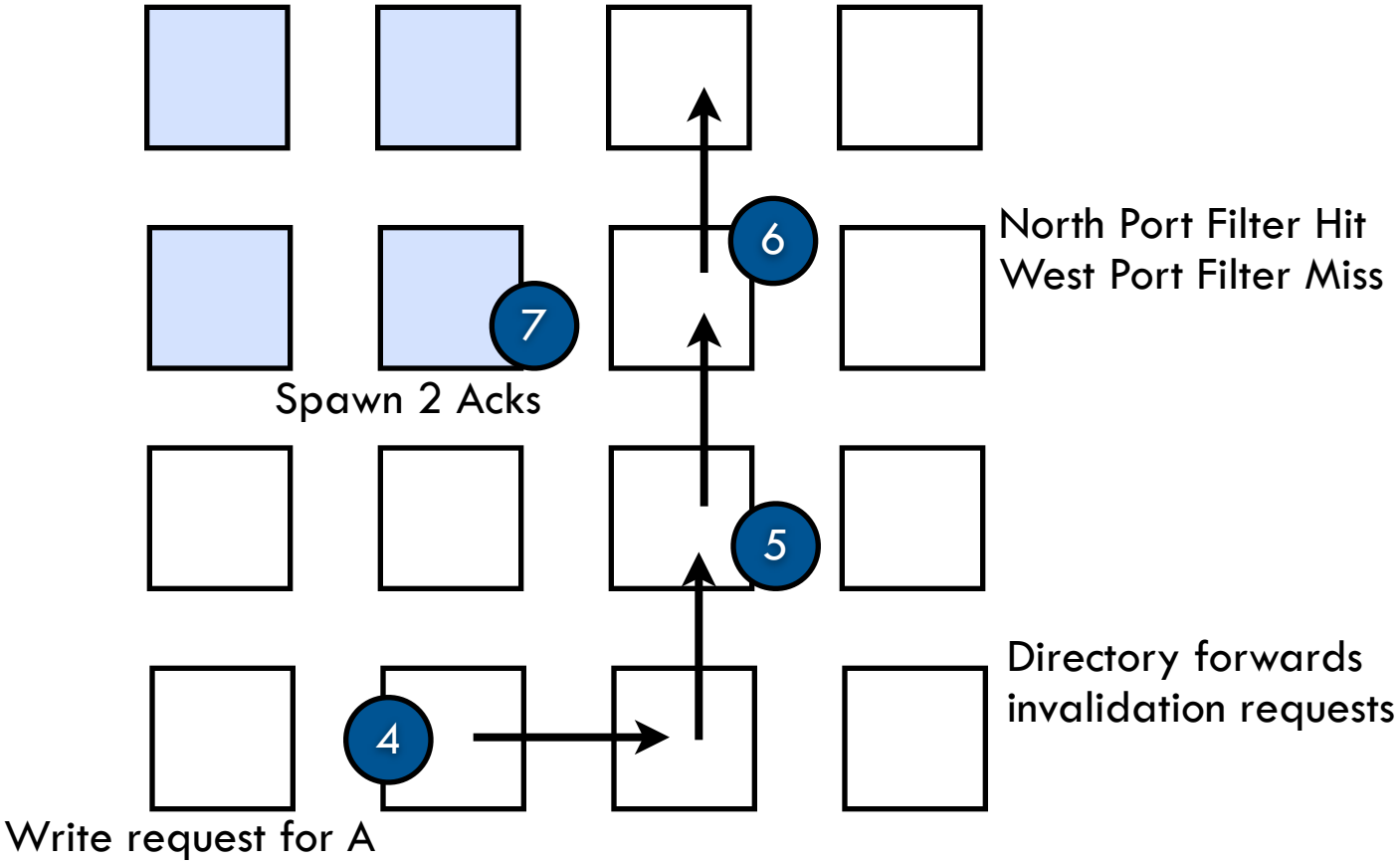
# SigNet Example (2)





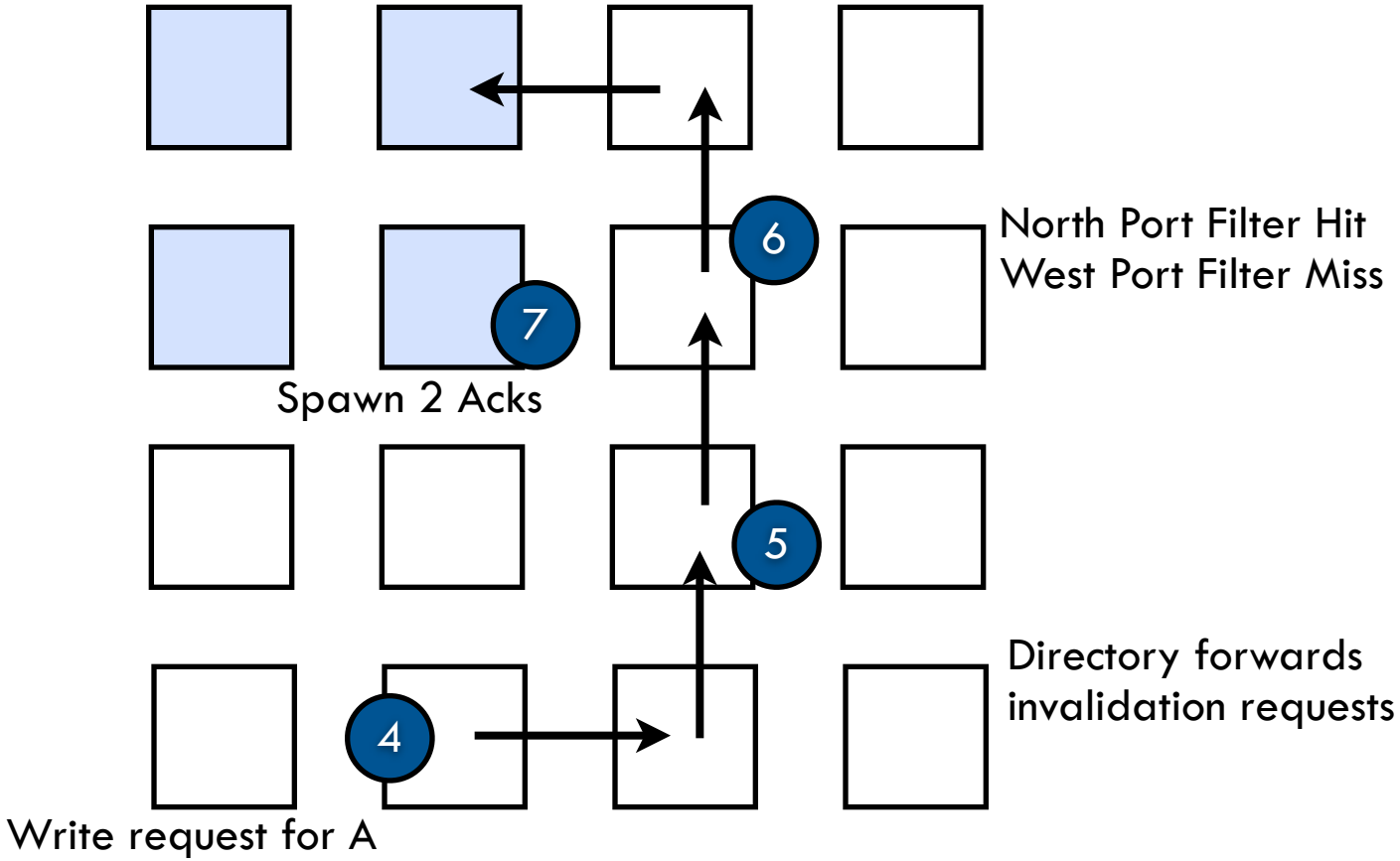
# SigNet Example (2)





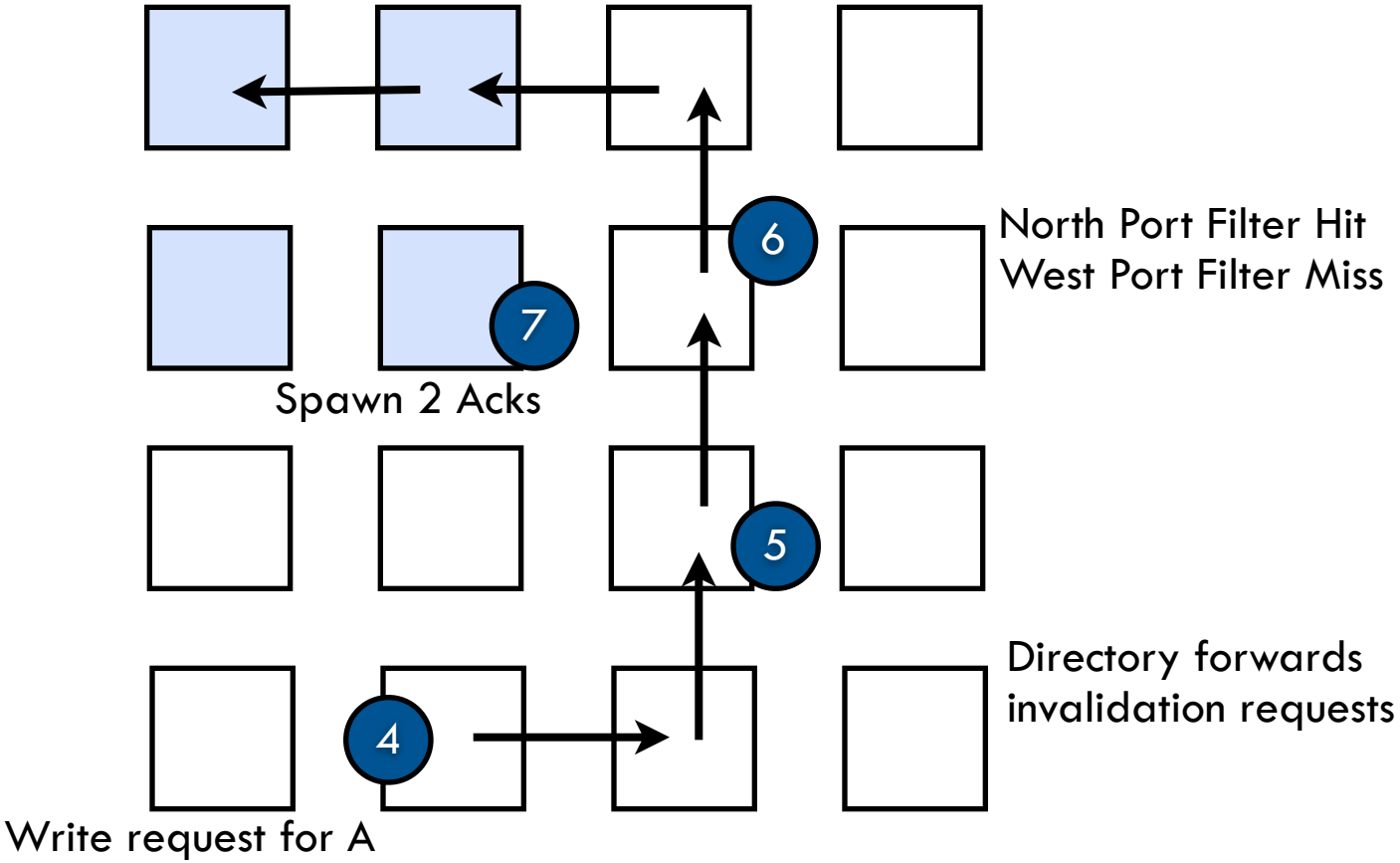
# SigNet Example (2)





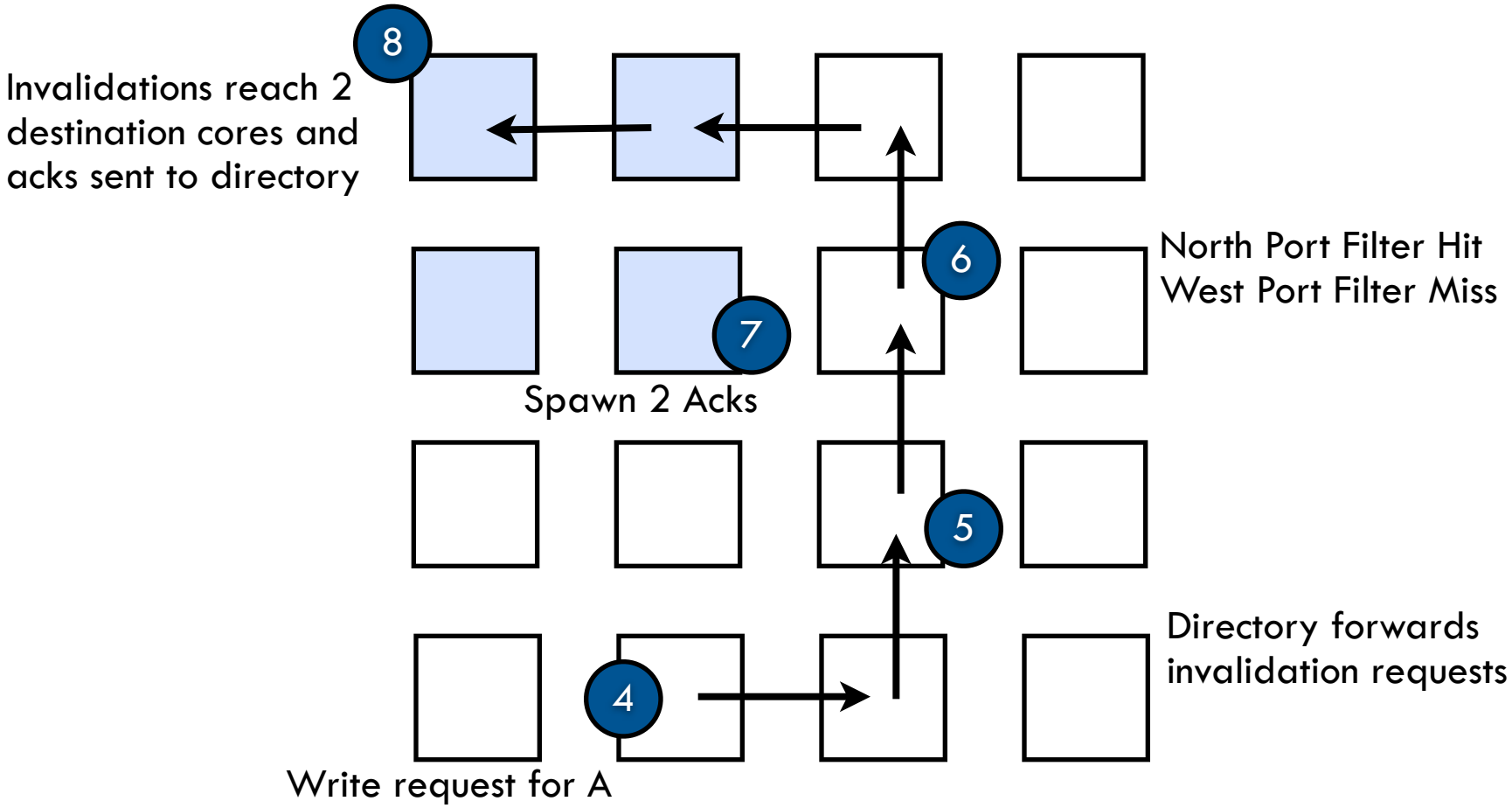
# SigNet Example (2)





# SigNet Example (2)





# SigNet Implementation



- Recall: Full-map directory requires 32 bytes per sharing vector for 256 cores
  - 50% overhead per cache line

- Evaluation uses 8K entry Bloom filters at each output port
  - Reduces overhead to 12.5% to 25% per cache line
    - Depends on size of counters and number of pointers in coarse vector directory

# Correctness and Utilization



- All cores caching a block must receive invalidation request
- Bloom filter can have false positives
  - Lessens performance benefit but correct
  - Cannot have false negatives
- Differences in utilization due to location and memory usage

# Simulation Methodology



| Network configuration | |
|---|---|
| Number of Nodes | 256 |
| Topology | 16 x16 mesh |
| Virtual Channels and Buffers | 4 VC/port<br>8 Buffers/VC |
| Link Width | 16 Bytes |
| Signature Size | 8192 |

# Simulation Methodology (2)

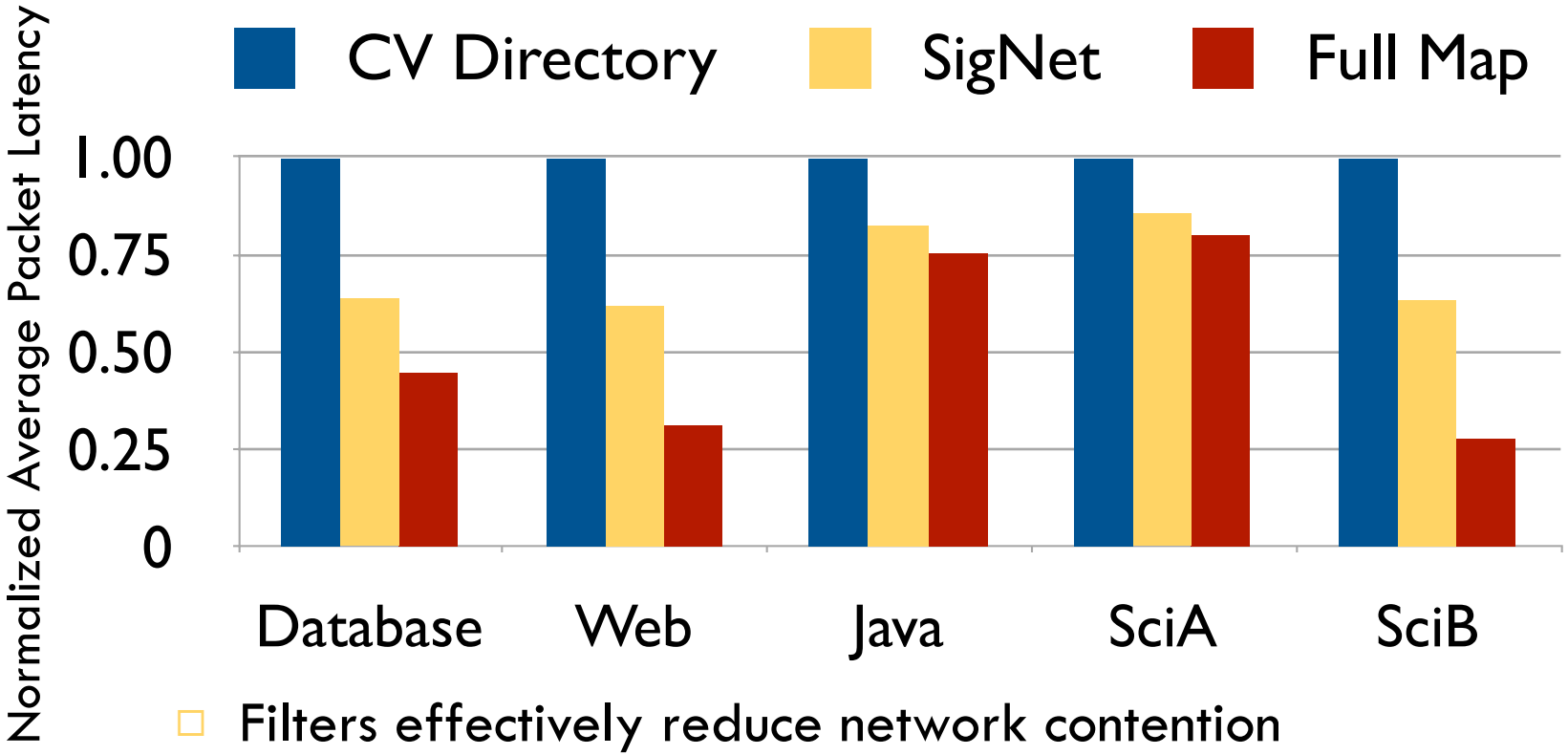


| Workload Parameters | | |
|---|---|---|
| **Name** | **%Invalidates** | **Average Sharers** |
| Database | 6.0 | 2.3 |
| Web | 3.5 | 3.8 |
| Java | 2.7 | 2.2 |
| Scientific A | 2.0 | 2.3 |
| Scientific B | 5.0 | 3.0 |

- Create synthetic benchmarks based on characteristics of 16-core workloads

# Results: 2 pointers, 16 cores per region

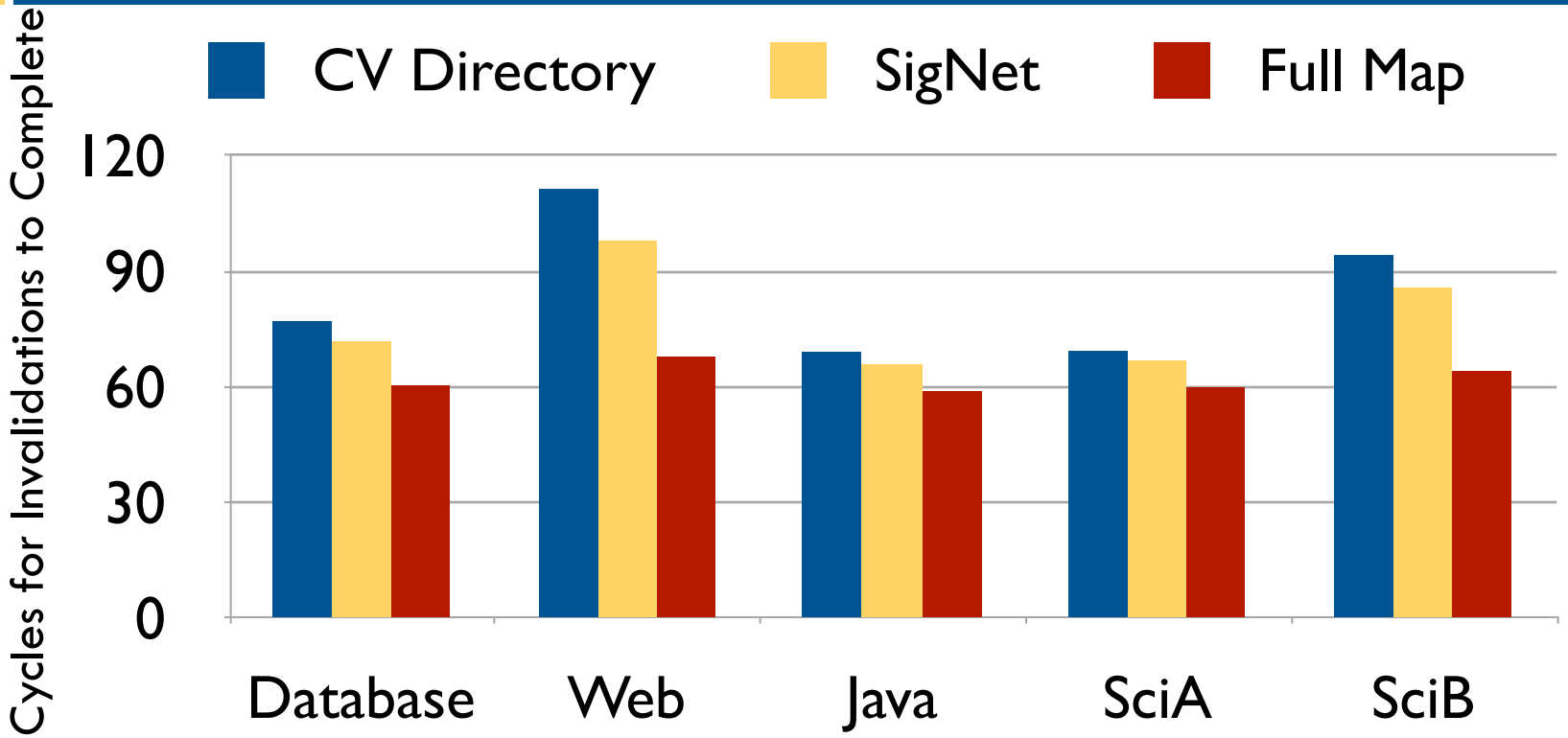





- Filters effectively reduce network contention
  - Lower average packet latency
- Additional research needed to further close gap with Full Map Directory

# Results: Invalidation Completion Time






- SigNet improves invalidation completion time
- Comparison with Pruning Caches in paper

# Related Work



- Interconnection Network Support
  - Pruning Caches
  - In-network coherence filters
- Cache Coherence Optimizations with Bloom filters
  - Jetty filters: reduce cache snoops
  - Tagless Coherence Directories
    - Reduce storage overheads

# Conclusions



- Characterize impact of CV directories
  - Significant power consumption and performance degradation
- Interconnect support to facilitate scalable cache coherence
  - SigNet: network filters to reduce extraneous invalidations

# Thank you



- Questions?