Stream Floating: Enabling Proactive and Decentralized Cache Optimizations

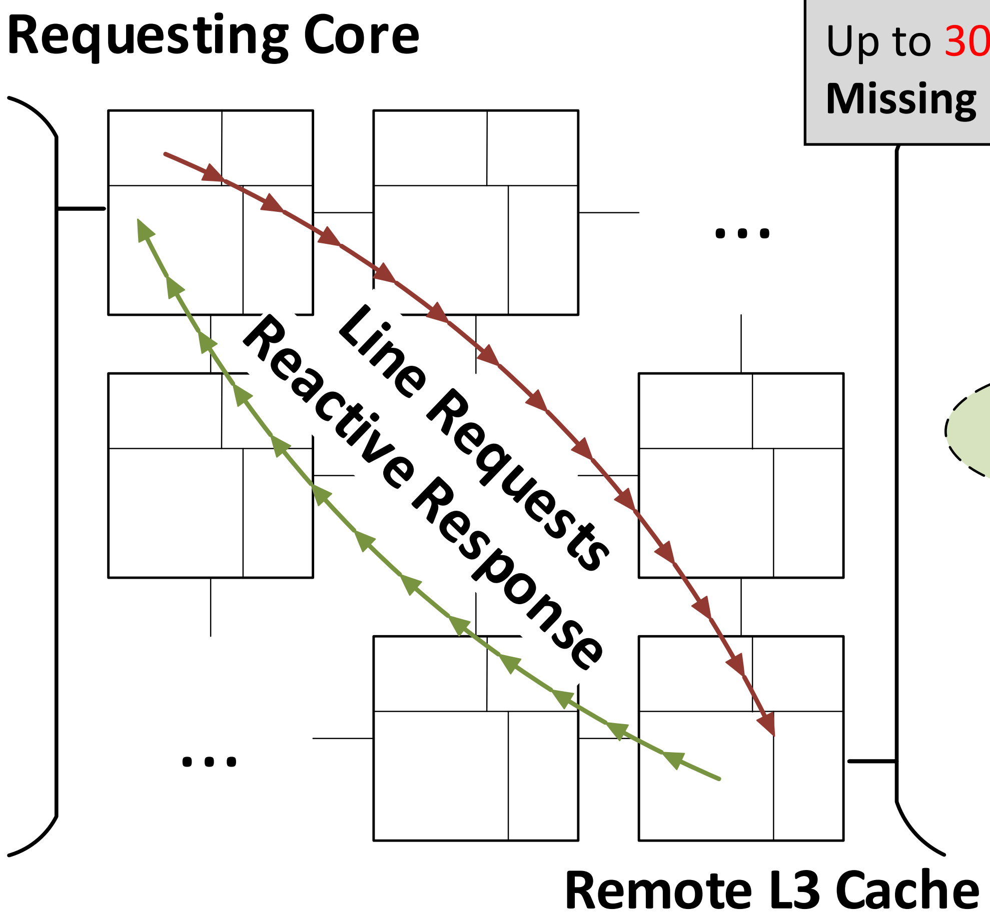
Zhengrong Wang[1], Jian Weng[1], Jason Lowe-Power[2], Jayesh Gaur[3], Tony Nowatzki[1]

[1]UCLA, [2]UC Davis, [3]Intel

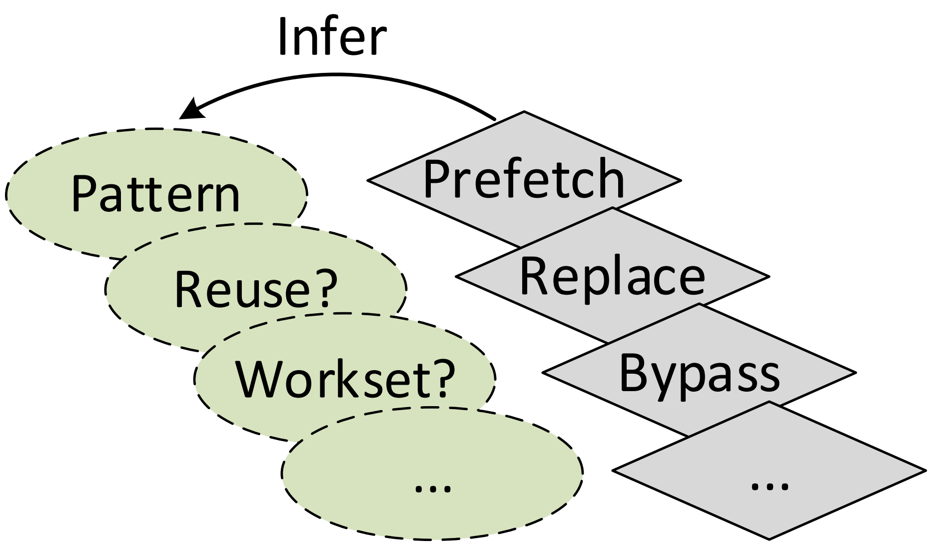
Feb. 2021

Information Gap → Reactive Cache System

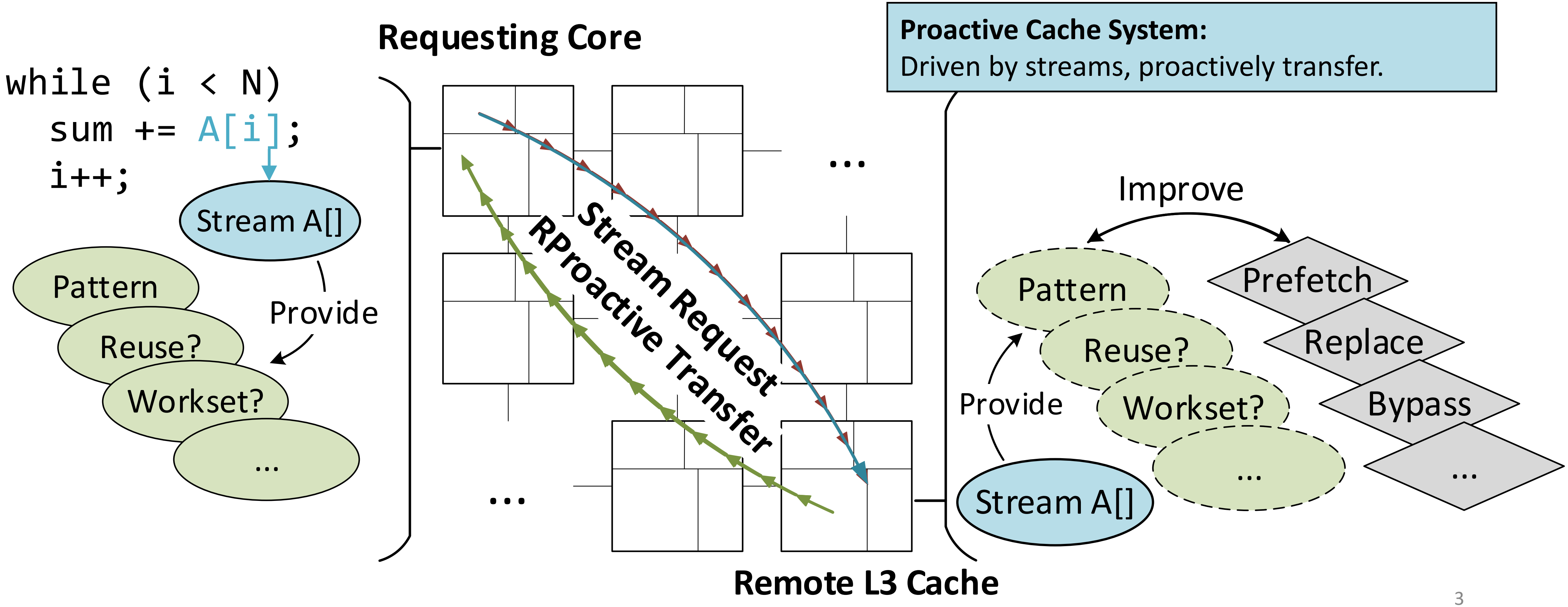
```
while (i < N)
  sum += A[i];
  i++;
```



Reactive Cache System:
72% cached lines without reuse.
Up to 30% extra control NoC traffic.
Missing holistic view of the program behavior.

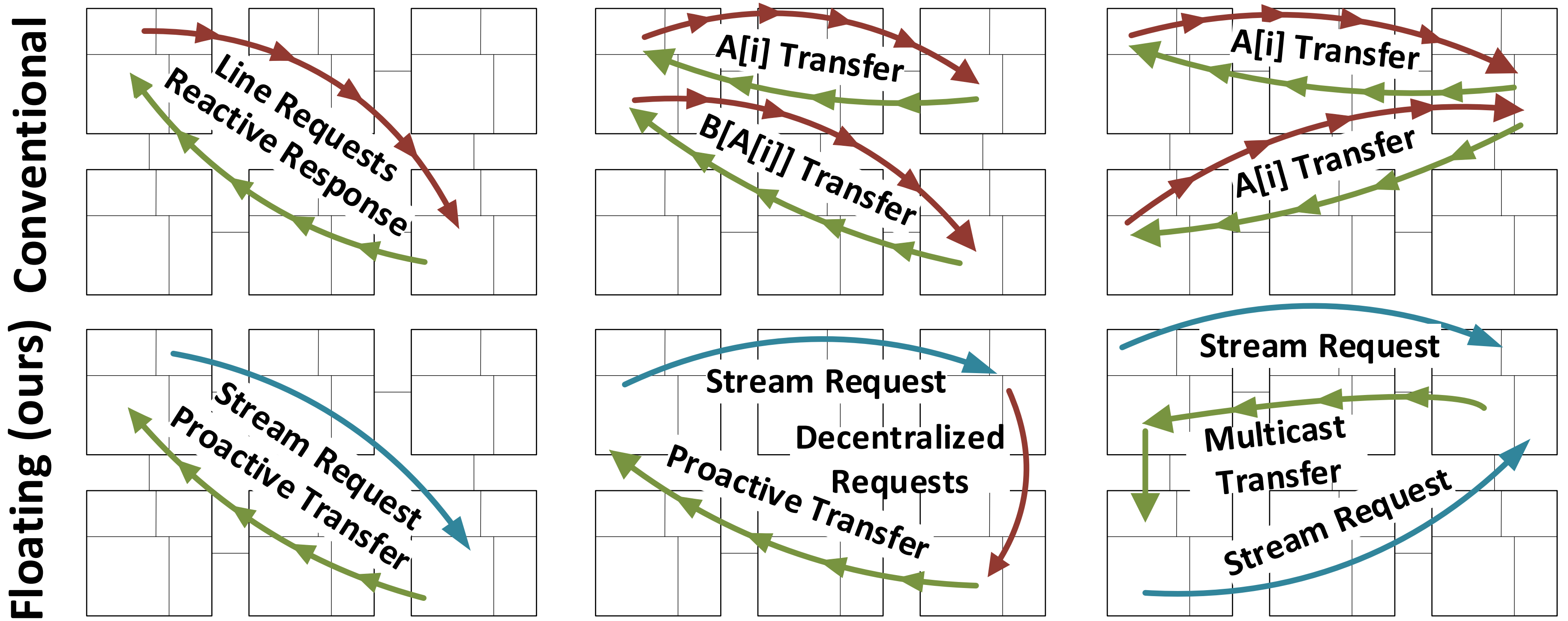


Streams to Bridge the Gap → Proactive Cache



Stream Floating → Proactive Cache System

Affine Access Pattern (eg. A[i])	Indirect Access Pattern (eg. B[A[i]])	Confluence Pattern (eg. A[i] on two cores)



Benefits

Affine Access Pattern	Indirect Access Pattern	Confluence Pattern
+ Less Request Traffic	**+ Less Request** (decentral)	**+ Less Response** (multicast)
+ Lower Latency	**+ Less Response** (subline)	**+ Lower LLC Bandwidth**

+ Lower Latency (proactive) **+ Less Thrashing** (non-cached streams)

Stream Floating Implementation

Original C Code

```
int i = 0;
while (i < N) {
  sum += B[A[i]];
  i++;
}
```

Configure

Usage

Advance

Stream Pseudo-Code

```
stream_cfg(B[A[]]);
while (i < N) {
  sum += stream_ld();
  stream_step();
}
```

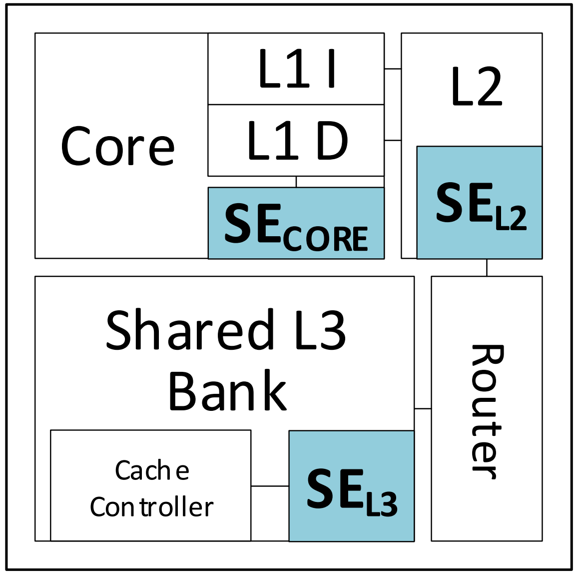
Static + dynamic information → floating decision.
e.g. reuse distance, aliased stores, hit rate in L2.
Offload entire stream pattern with one message.

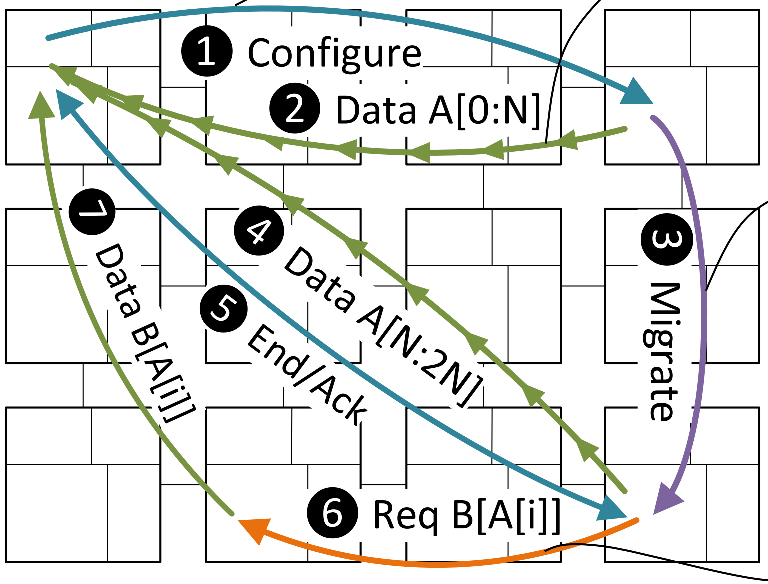
Proactively transfer stream data.
Stream data is not cached (bypass coherence).
Extend MESI with <u>**GetU**</u>ncached request.

Automatically migrate to next bank.
Keep streaming until no credits.
Released by StreamEnd message.

Weak consistency w. aliasing detection.
Strong consistency is also possible.
w. stream-grain coherence.

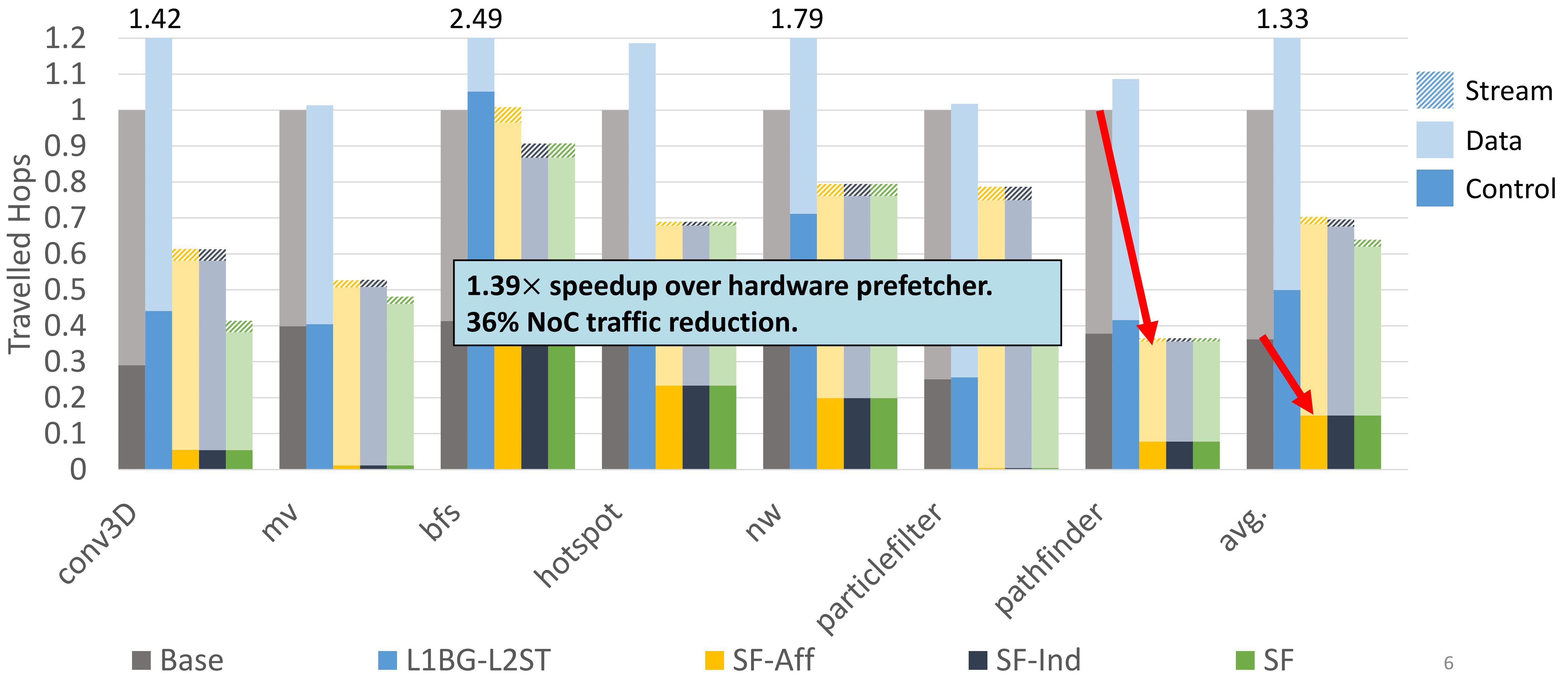
Indirect requests from remote SE_{L3}.
Decentralized address generation.



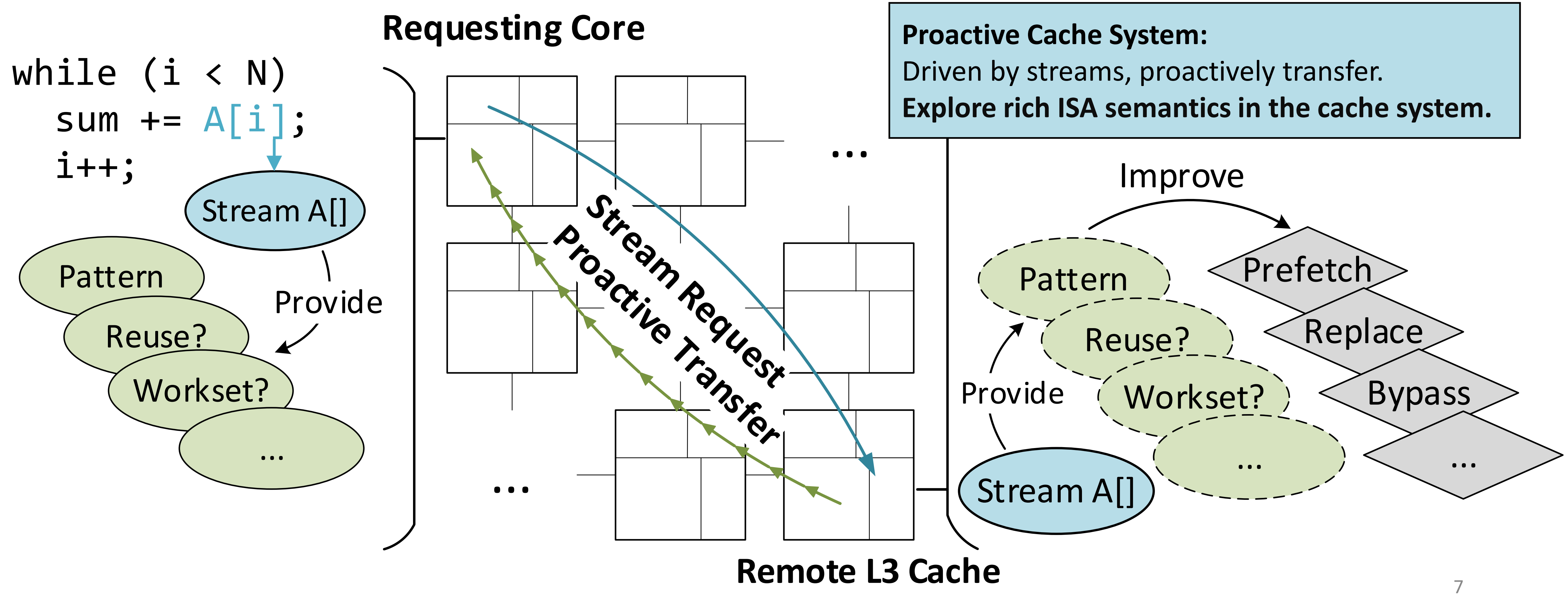


NoC Traffic Breakdown

- Baseline: No HW prefetcher or stream support.
- L1BG-L2ST: L1 Bingo spatial + L2 stride prefetcher.
- SF-X: Stream-floating with each optimization enabled.



Conclusion: Streams Enables Proactive Cache



Stream Floating: Enabling Proactive and Decentralized Cache Optimizations

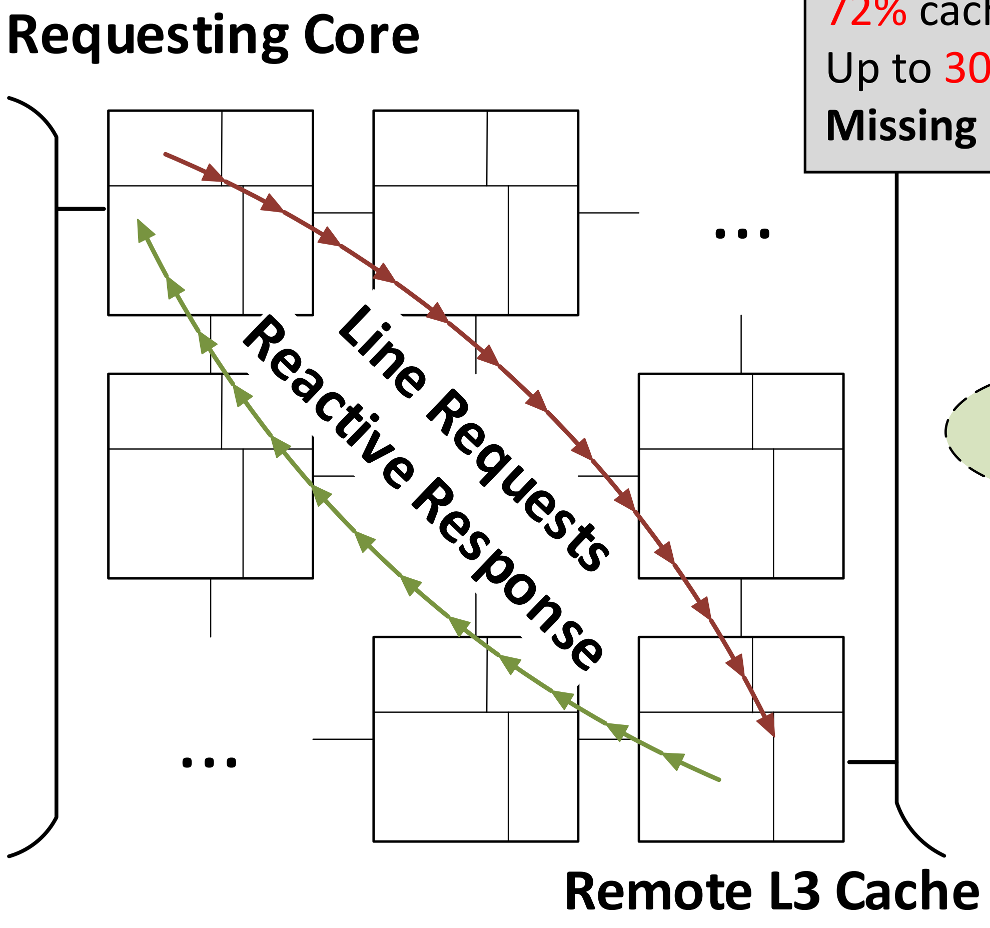
Zhengrong Wang[1], Jian Weng[1], Jason Lowe-Power[2],
Jayesh Gaur[3], Tony Nowatzki[1]

[1]UCLA, [2]UC Davis, [3]Intel

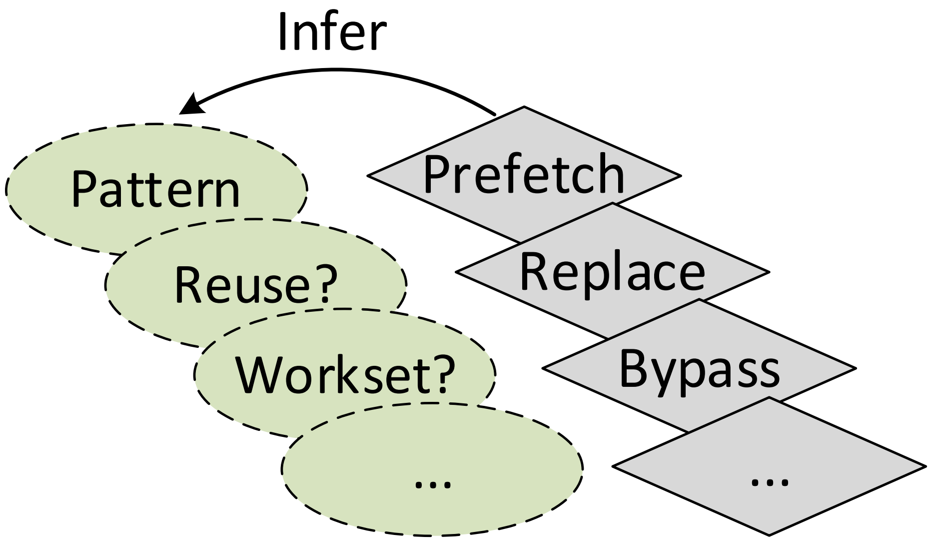
Feb. 2021

Information Gap → Reactive Cache System

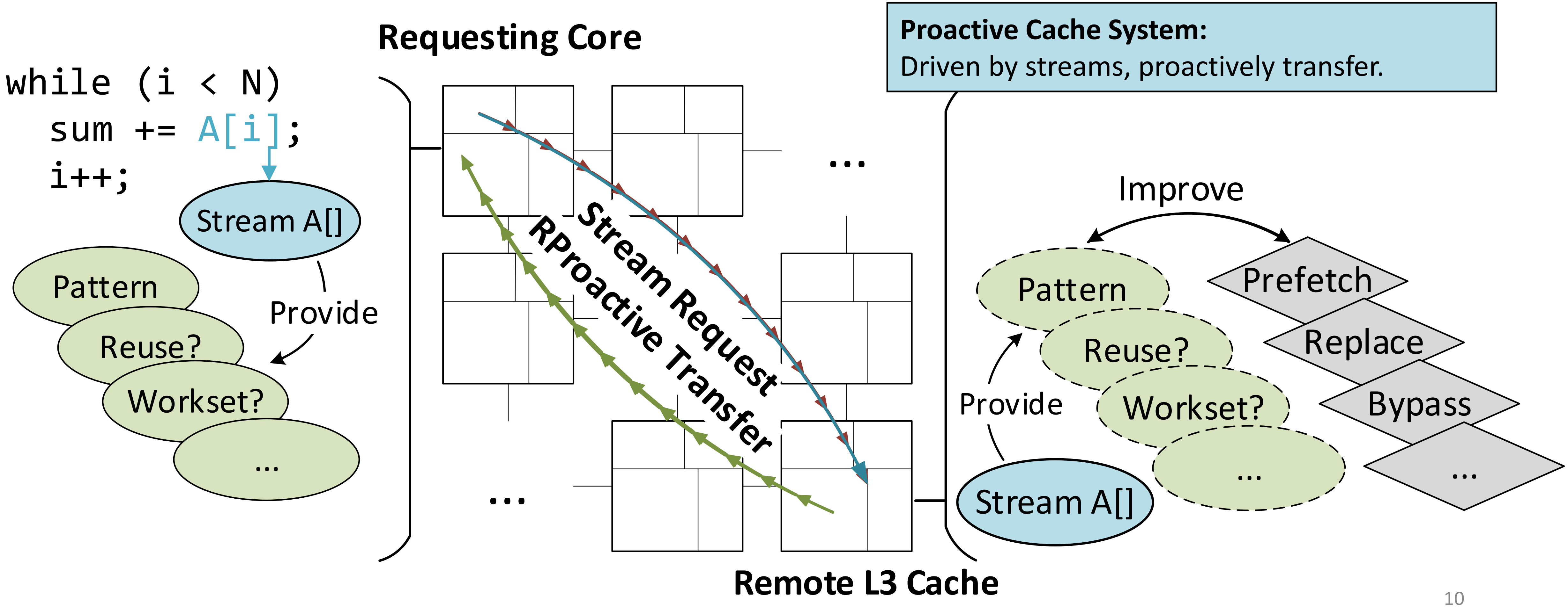
```
while (i < N)
  sum += A[i];
  i++;
```



Reactive Cache System:
72% cached lines without reuse.
Up to 30% extra control NoC traffic.
Missing holistic view of the program behavior.

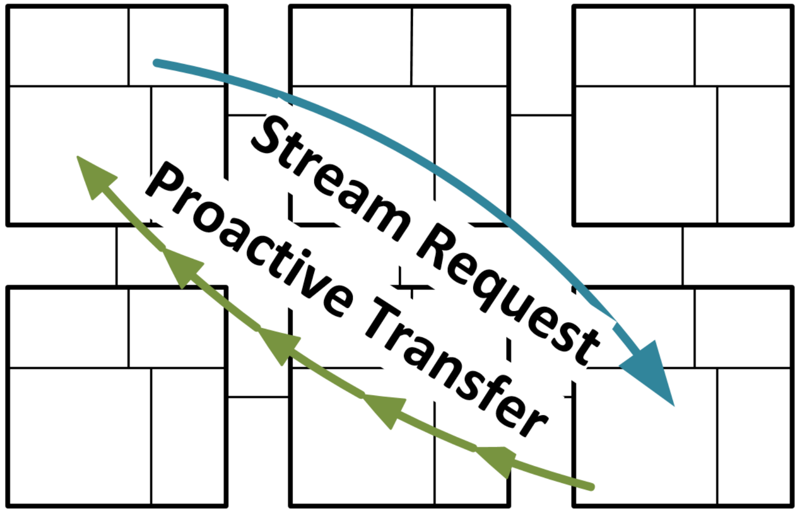


Streams to Bridge the Gap → Proactive Cache



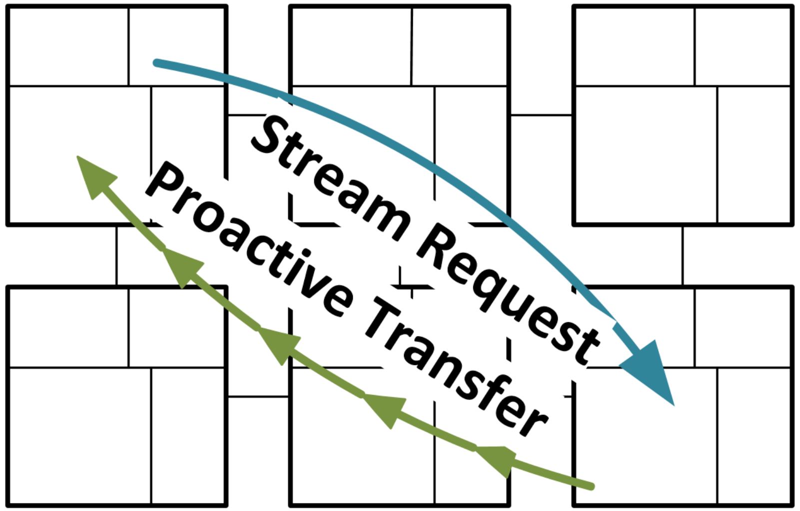
Stream Floating → Proactive Cache

- Expose stream patterns without reuse to shared L3 banks.
- Proactive cache system that driven by streams.
  - One request for an entire stream.
  - Accurate prefetch.
  - Simplified coherence protocol.
- **1.39× speedup over hardware prefetcher.**
- **36% NoC traffic reduction.**



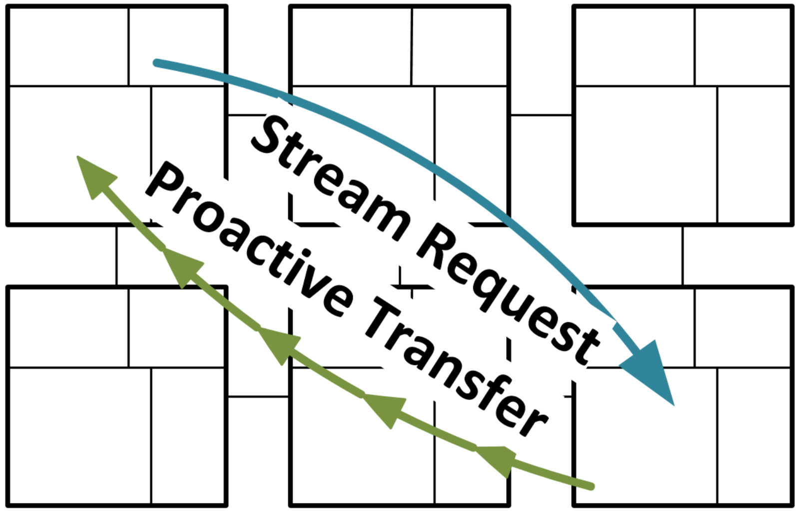
Outline

- Insights and Opportunities
- Stream Floating Implementation
- Coherence and Consistency
- Evaluation

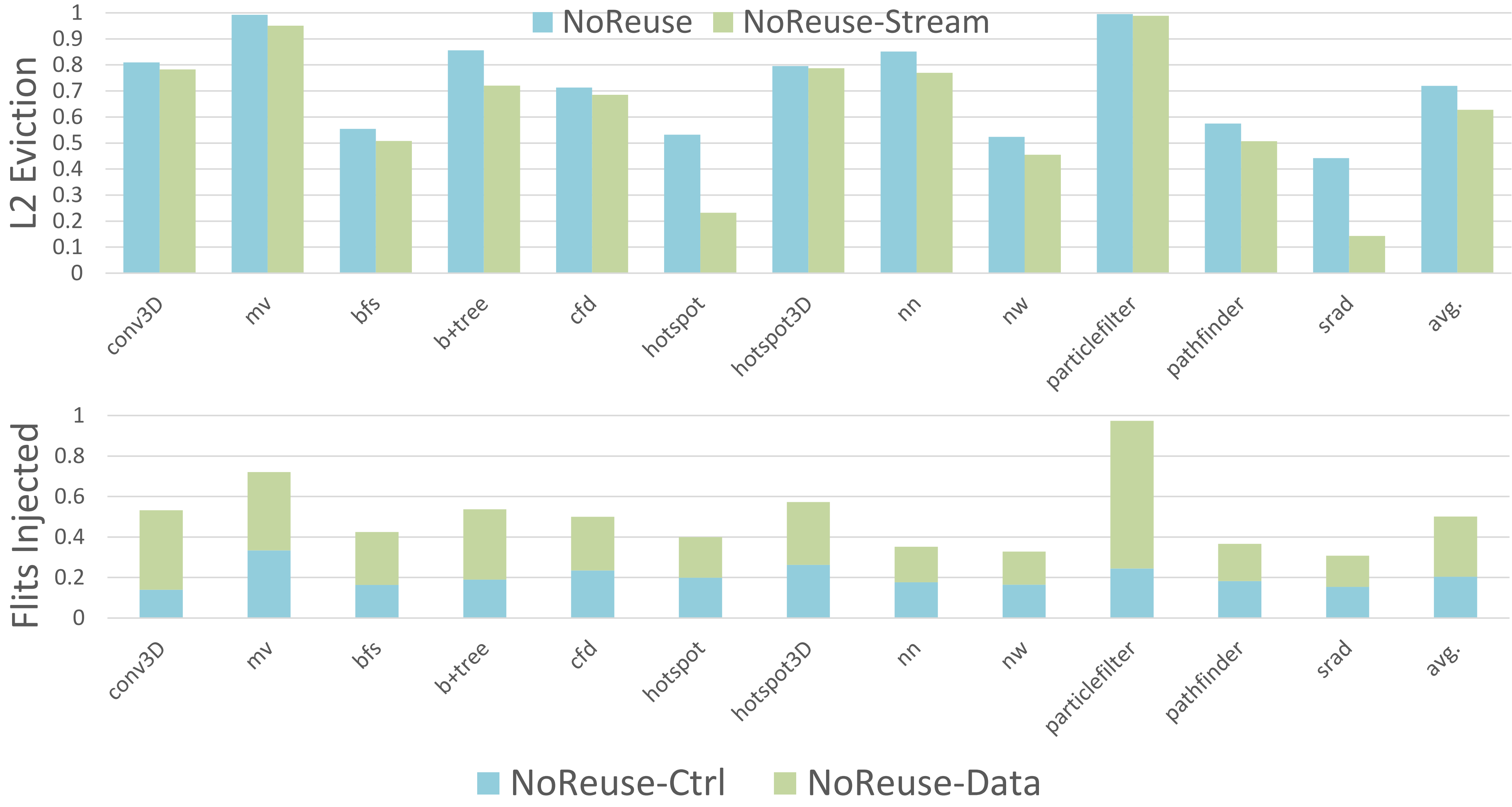


Outline

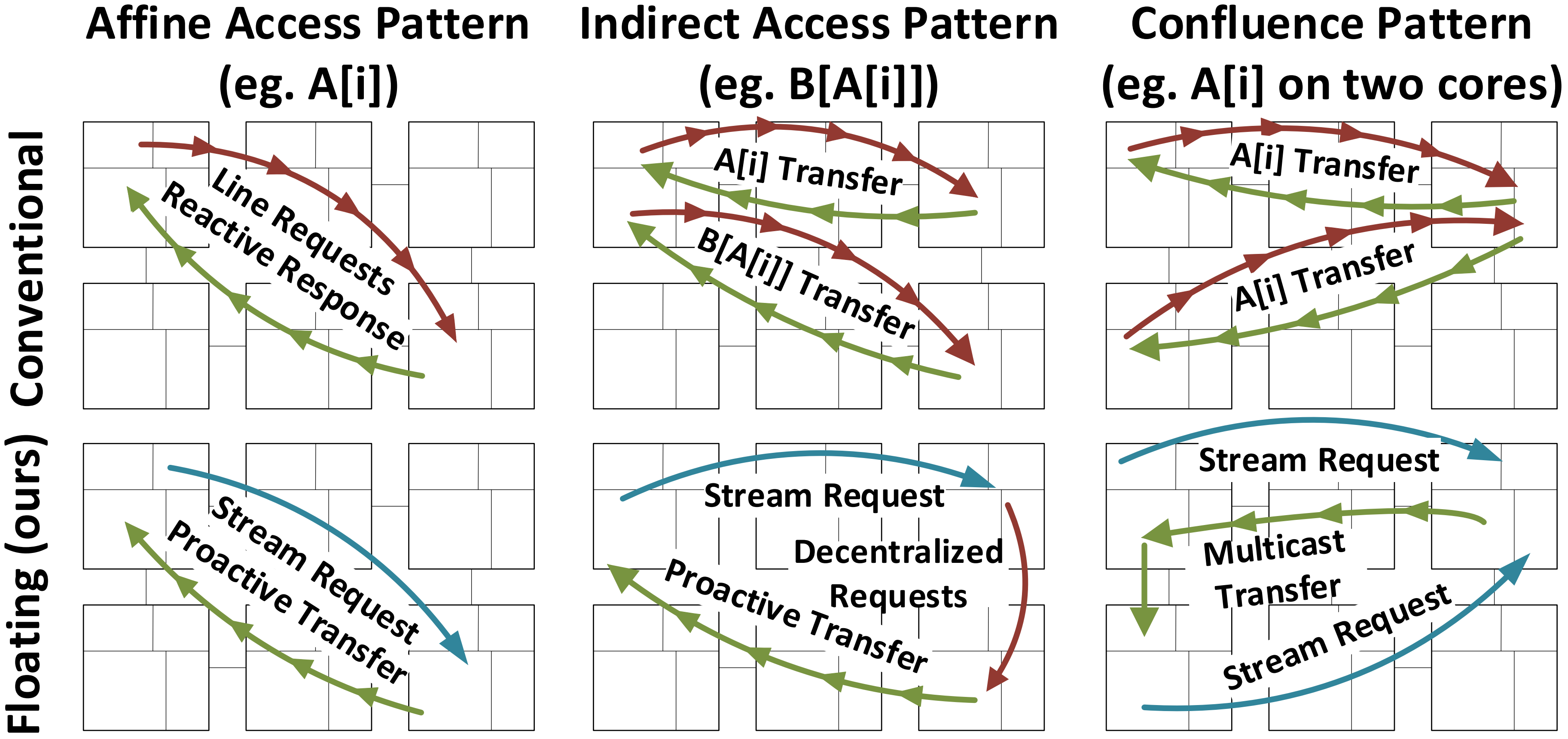
- Insights and Opportunities
- Stream Floating Implementation
- Coherence and Consistency
- Evaluation



Overheads of Caching Data without Reuse



Conventional vs. Stream Floating



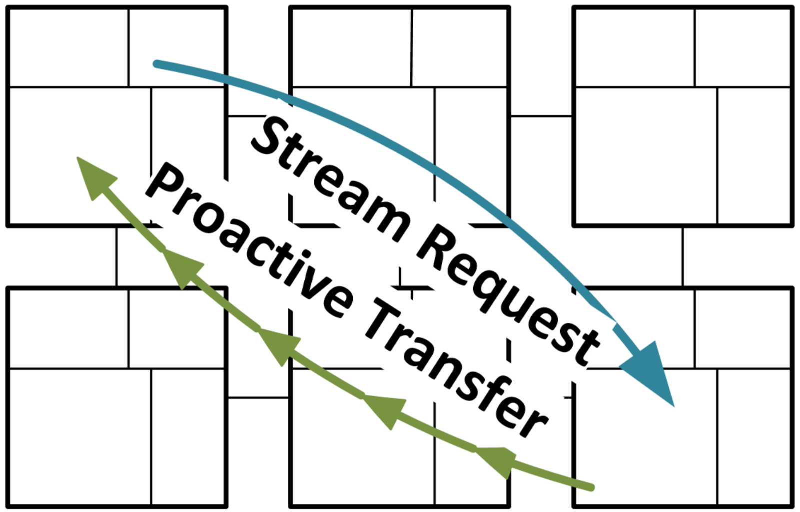
Benefits

+ Less Request Traffic + Lower Latency	+ **Less Request** (decentral) + **Less Response** (subline)	+ **Less Response** (multicast) + **Lower LLC Bandwidth**

+ **Lower Latency** (proactive) + **Less Thrashing** (non-cached streams)

Outline

- Insights and Opportunities
- Stream Floating Implementation
  - What to offload?
  - How to offload?
  - When to offload?
- Coherence and Consistency
- Evaluation



What to Offload: Streams

- Stream: A decoupled sequence of values/addresses [ISCA’ 19].
- Explicitly embedded in the ISA.
- Memory order defined by the first usage of the value.

Original C Code

```
int i = 0;
while (i < N) {
  sum += a[b[i]];
  i++;
}
```

Configure.

Use the value.

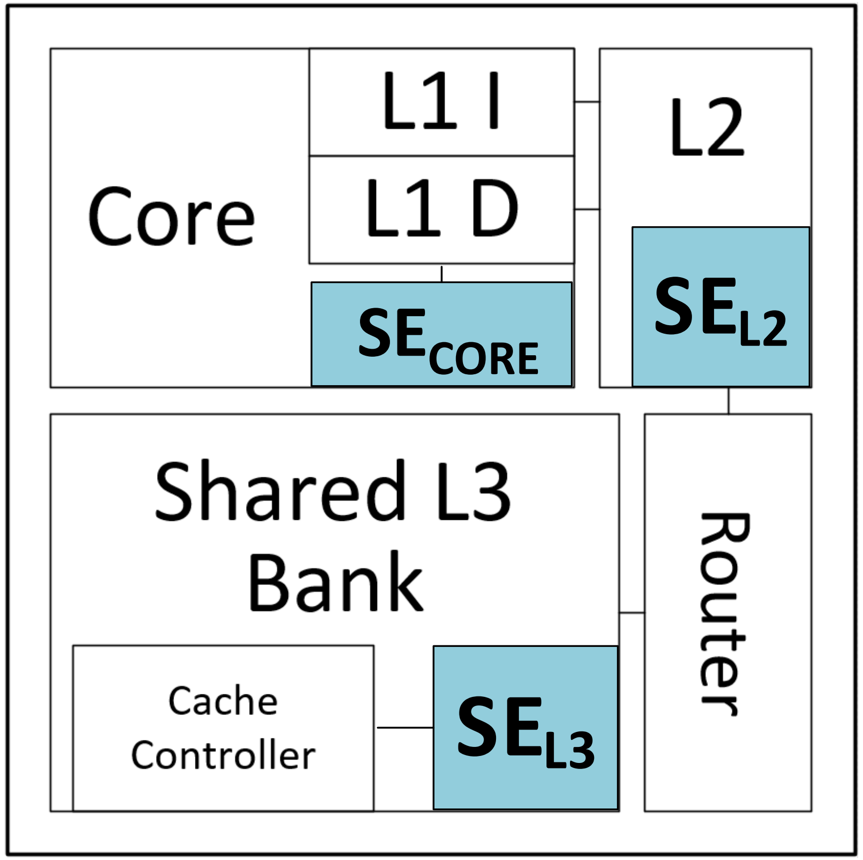
Advance.

Decoupled Stream Pseudo Assembly

```
while (i < N) {


}
```

Stream Engines



SE_{CORE}:
Mange stream configuration and issue stream requests.
Make and cancel offload decisions.

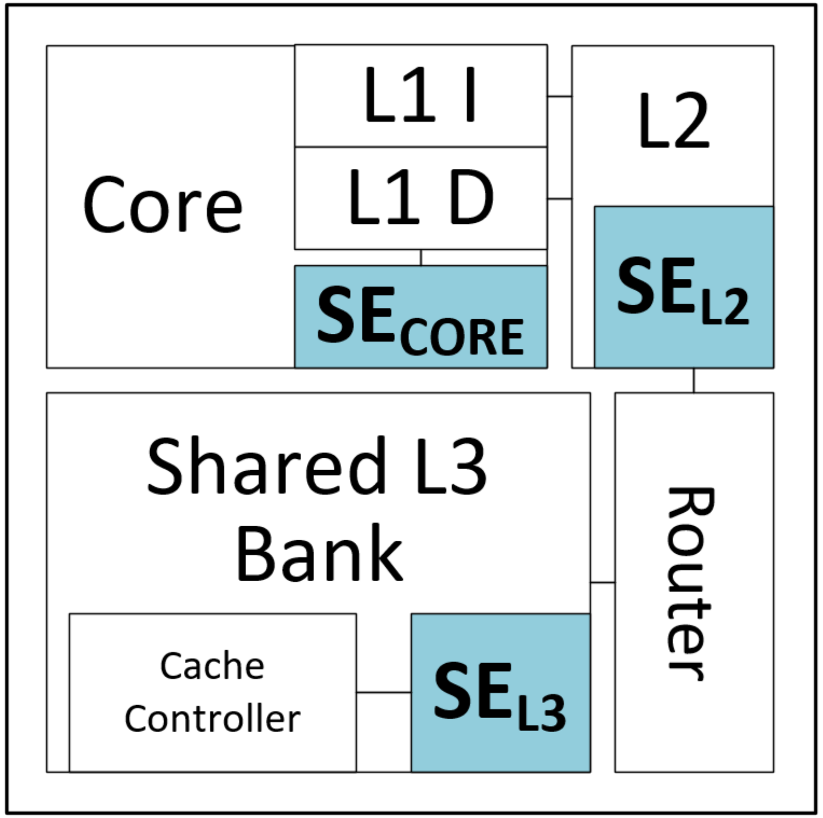
SE_{L2}:
Buffer stream data and match it with requests from SE_{CORE}.
Issue flow control credits to remote L3 bank (SE_{L3}).

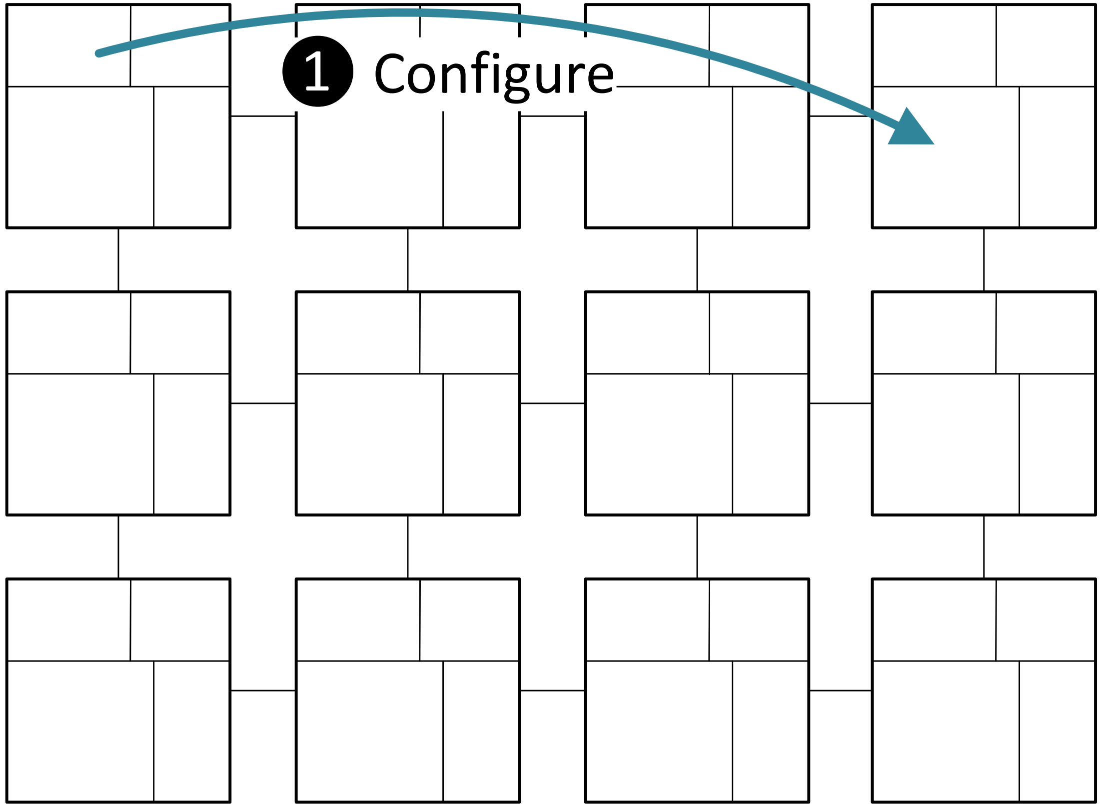
SE_{L3}:
Generate requests and stream back data to SE_{L2}.
Receives control messages from SE_{L2}, e.g. flow credits.

How to Offload: Configure Affine Stream A[i]

SE_{CORE} configures SE_{L2} with the affine stream pattern A[i].

SE_{L2} allocates the stream buffer, and forward configuration to SE_{L3} where A[0] is.

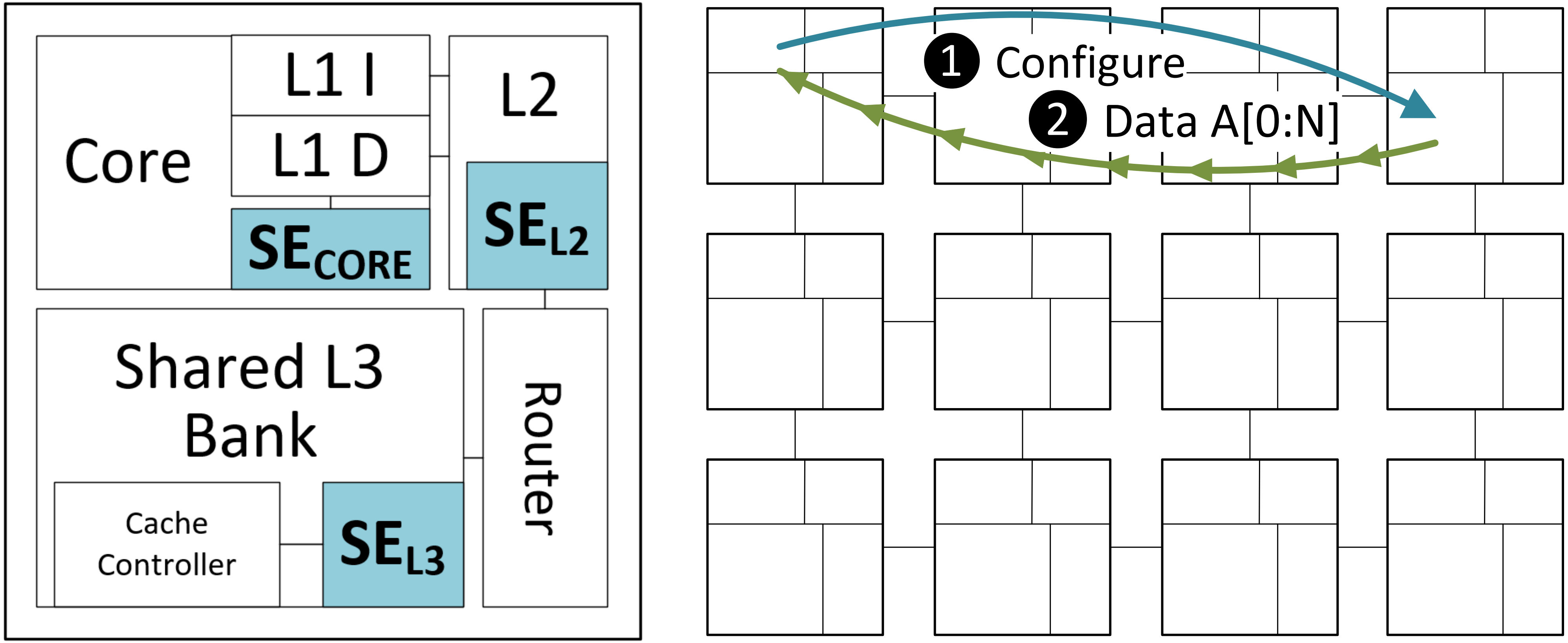




How to Offload: Proactively Stream Data to Core

SE_{L3} generates requests of A[i], translates (L2 TLB) and sends to L3 cache controller.

Data responses are buffered at SE_{L2} and later drained by requests from SE_{CORE}.

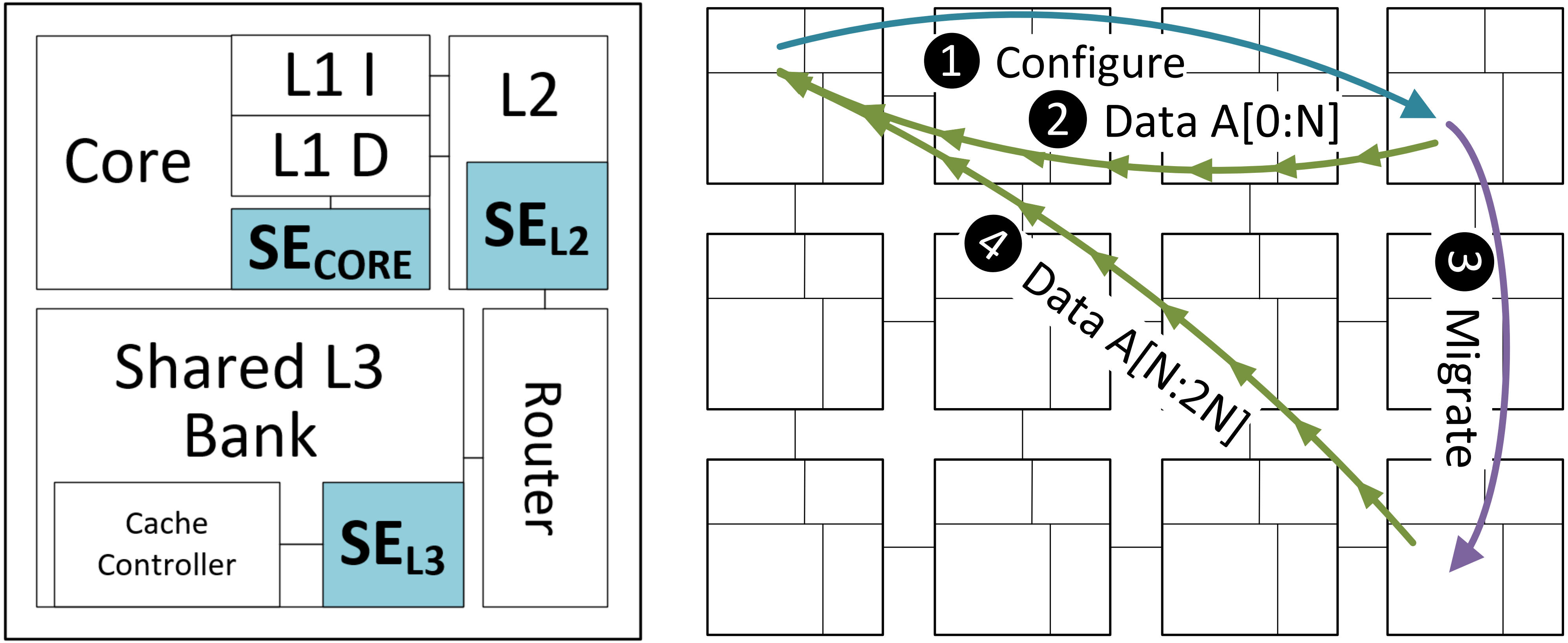


How to Offload: Flow Control and Migration

SE_{L2} sends out credits to SE_{L3} at coarse-granularity, further reduce traffic overhead.

Streams migrate to the next bank, and keep streaming until no credits.

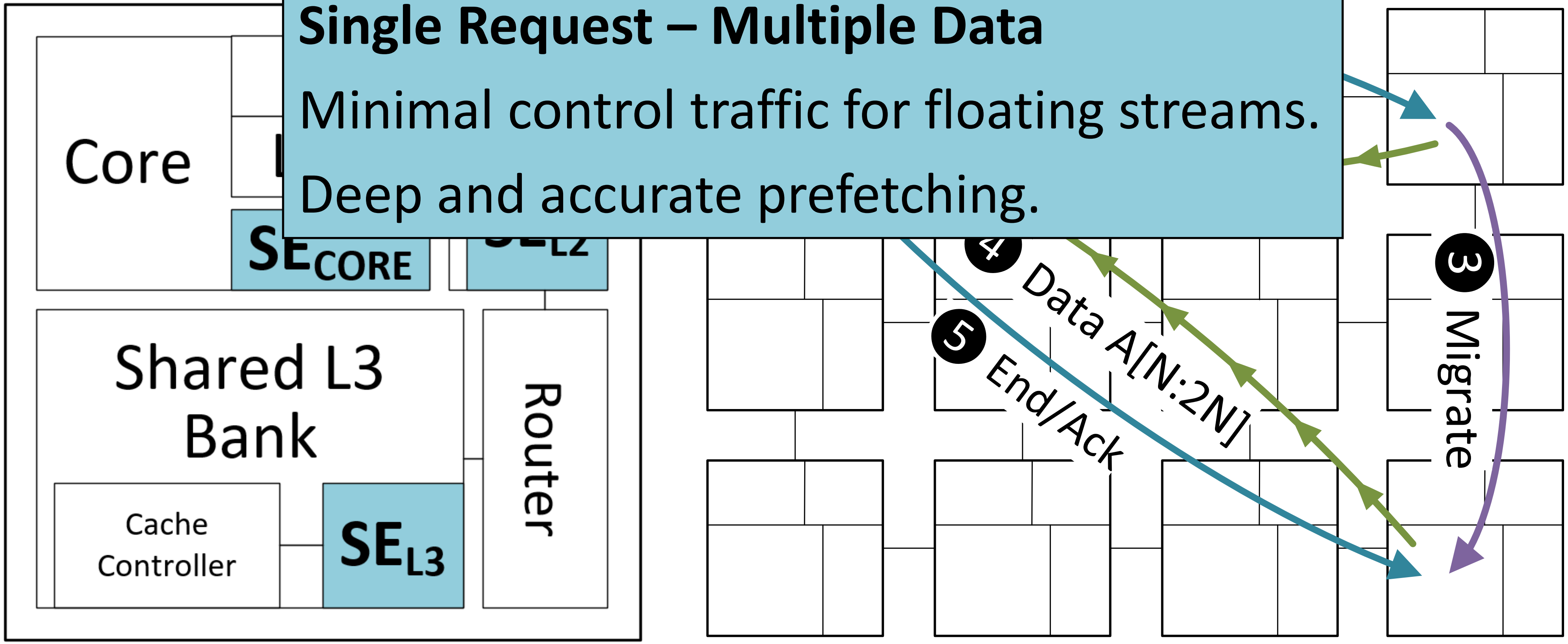
- Slightly increase interleave granularity to avoid too-frequent migrations.



How to Offload: End (Sink) the Stream

SE_{CORE} terminates streams by sending out StreamEnd messages.

SE_{L3} can directly release streams with known length.



Single Request – Multiple Data

Minimal control traffic for floating streams.

Deep and accurate prefetching.

How to Offload: Indirect Stream B[A[i]]

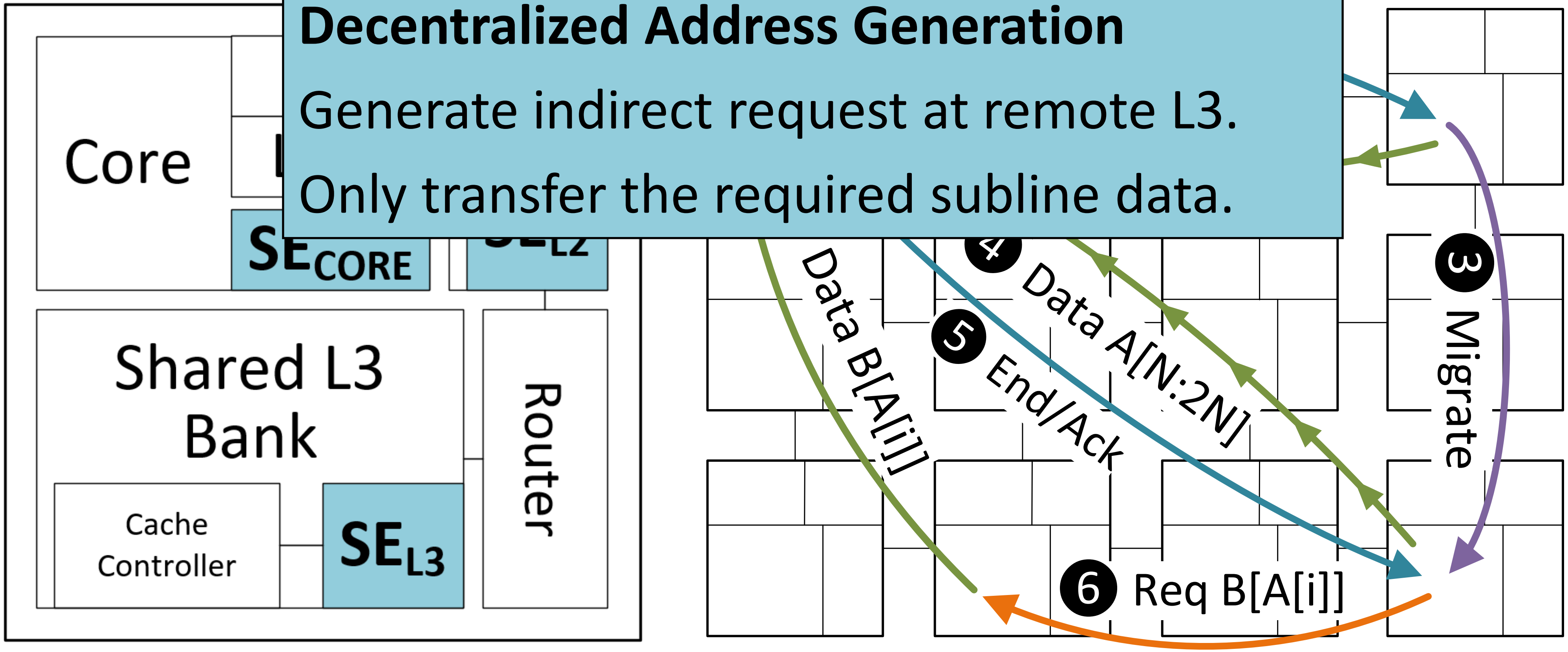
Associate indirect stream B[A[i]] with A[i].

SE_{L3} can directly send out indirect requests to the target L3 cache controller.

Decentralized Address Generation

Generate indirect request at remote L3.

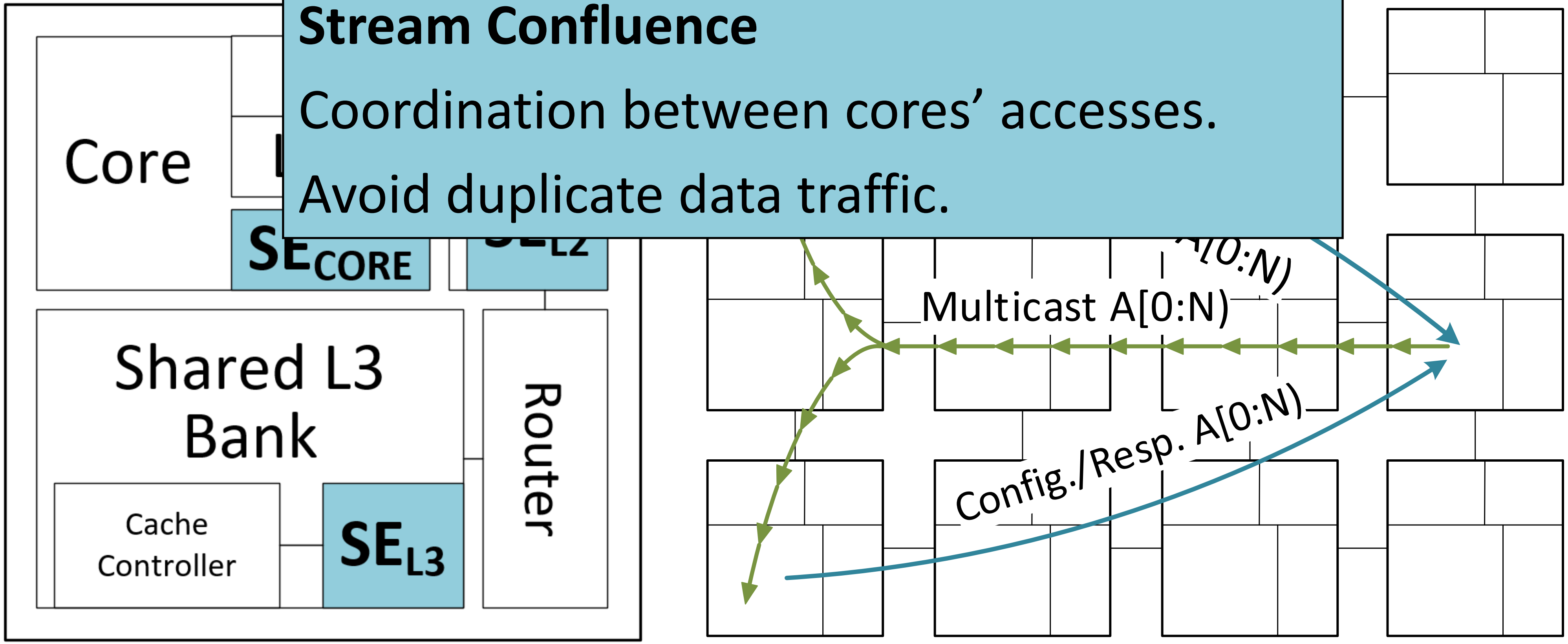
Only transfer the required subline data.



How to Offload: Stream Confluence

Neighboring cores may be requesting the same piece of data.

SE_{L3} can easily perform pattern matching and multicast data to different cores.



Stream Confluence

Coordination between cores' accesses.

Avoid duplicate data traffic.

When to Offload: Detect Floating Candidates

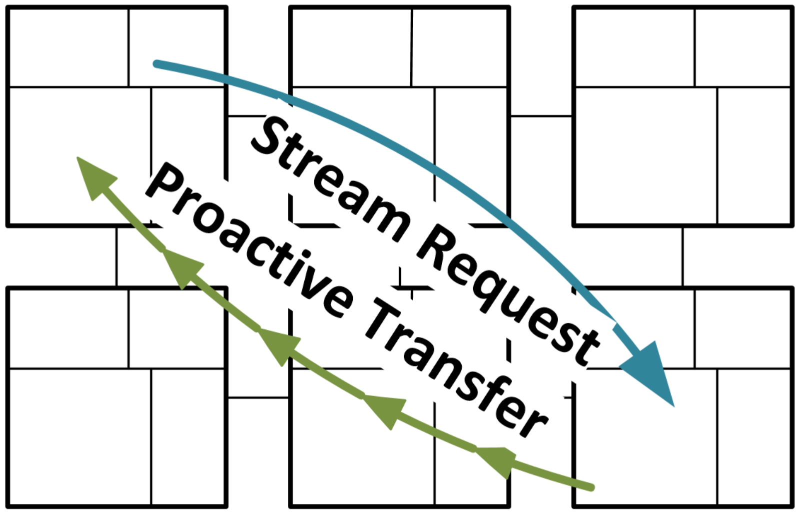
- Target: Streams with no reuse in the private cache, or aliasing.
- Static Information: Compiler Analysis.
  - E.g. are there writes to the address?
- Dynamic Information: Stream History Table.
- Only offload when passed both static & dynamic check.
  - SE_{CORE} can early terminate a floating stream, e.g. found aliasing.

Field	Description	Field	Description
`sid`	Stream id	`request`	# stream requests
`reuse`	# priv. cache reuses	`miss`	# priv. cache misses
`aliased`	Aliased with stores		

TABLE II: Stream History Table

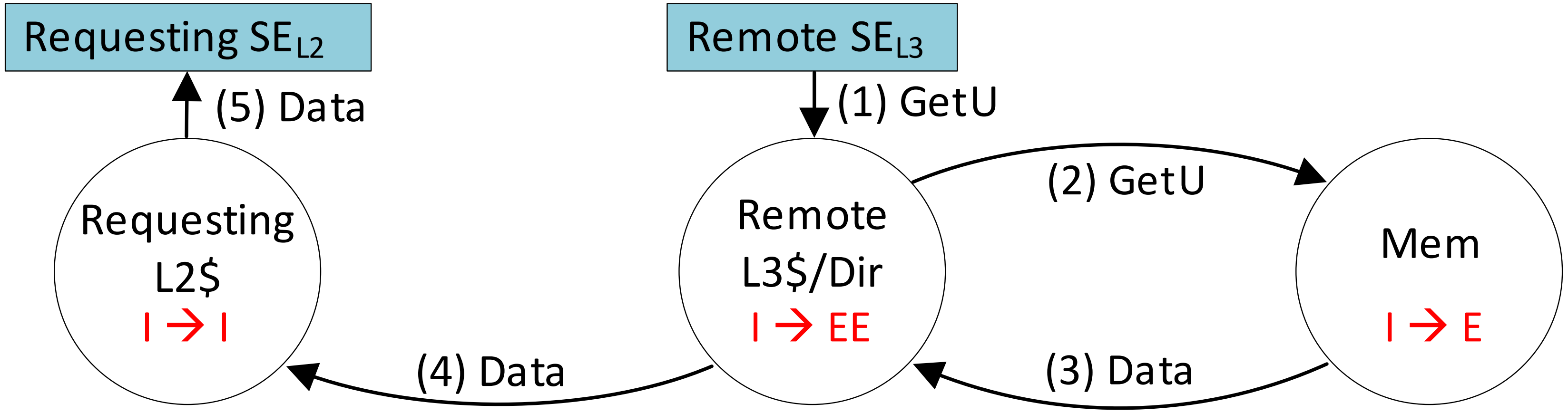
Outline

- Insights and Opportunities
- Stream Floating Implementation
- Coherence and Consistency
  - Support Weak Consistency
  - Support Strong Consistency
- Evaluation



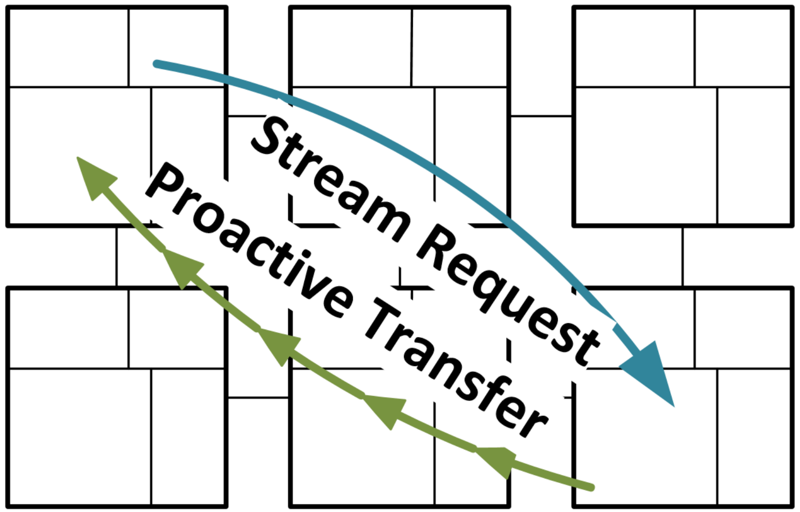
Support Weak Consistency: Uncached Stream Data

- Limit streams in synchronization-free region.
- Bypass coherence protocol: stream data is not cached.
  - Extend MESI protocol with uncached requests (GetU).
- Details about aliasing detection in the paper.
- Strong consistency with stream-grain coherence.



Outline

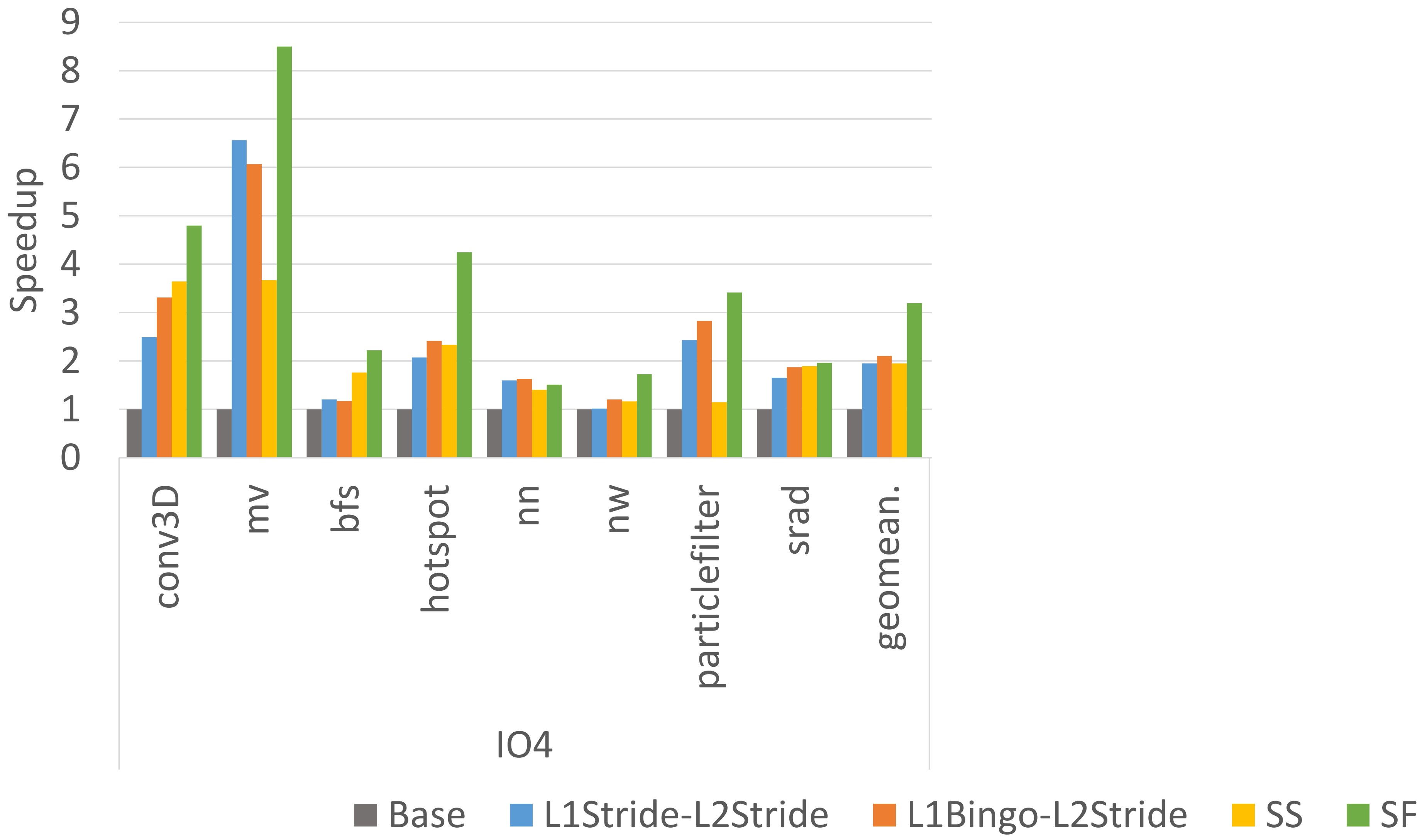
- Insights and Opportunities
- Stream Floating Implementation
- Coherence and Consistency
- Evaluation



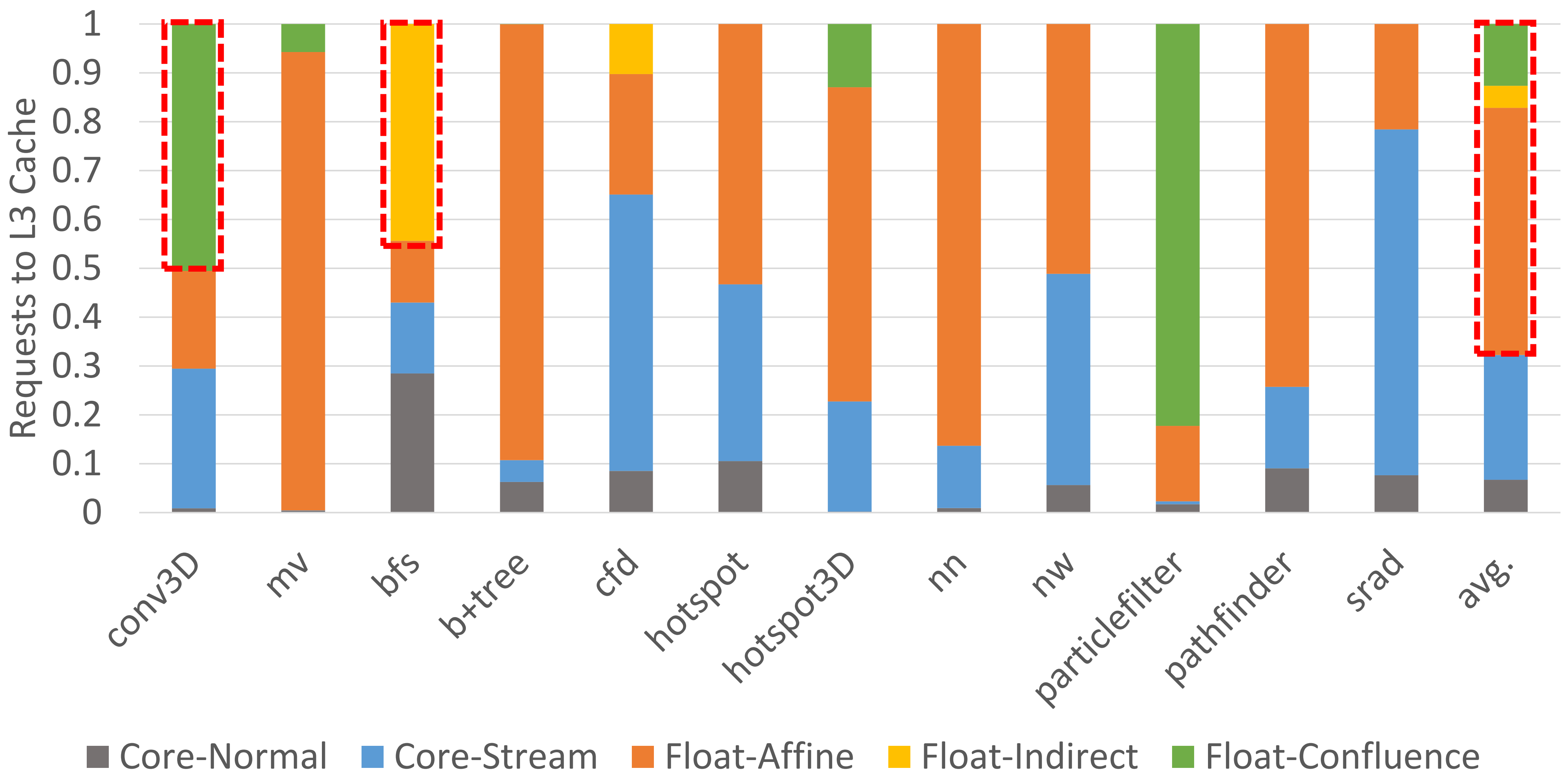
Configurations

- LLVM-based compiler to recognize streams and transform programs.
- Gem5 20.0 cycle-level execution-driven simulator.
- 12 data processing workloads from Rodinia and micro kernels.
  - Parallelized with OpenMP, with AVX-512 enabled.
- Configurations (see paper for details):
  - 8x8 mesh topology, 3-level MESI, 32kB L1 I/D, 256kB L2, 1MB L3.
  - Base: Baseline cores without prefetcher or stream support.
  - L1Stride-L2Stride: Stride prefetcher at both L1 and L2 cache level.
  - L1Bingo-L2Stride: Bingo spatial prefetcher at L1 and stride prefetcher at L2.
  - SS: Stream-specialized processor (stream support at core) [ISCA’ 19].
  - **SF: Stream floating, (offload stream to cache) [this work].**

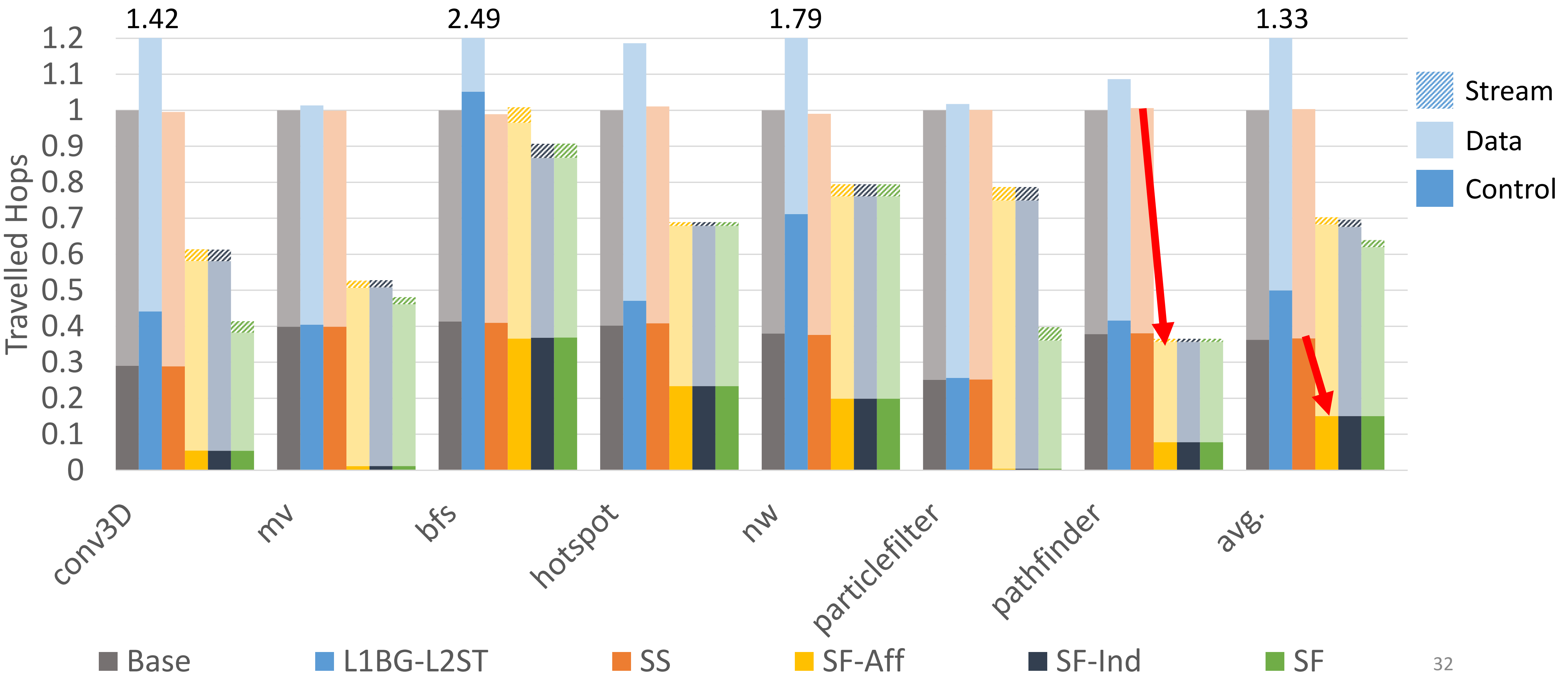
Overall Speedup with IO4 and OOO8



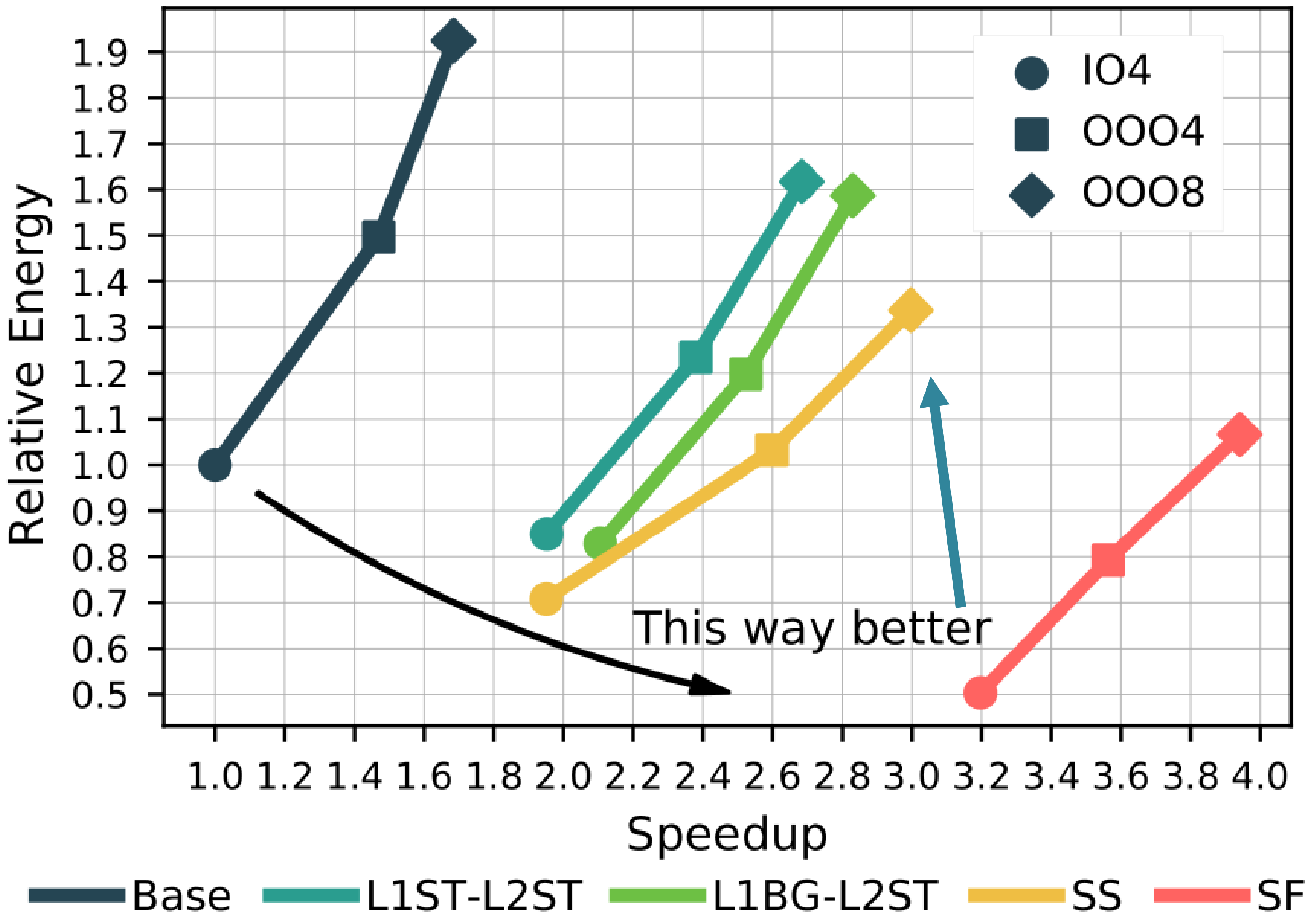
LLC Request Breakdown



NoC Traffic Breakdown



Energy vs. Speedup



Conclusion: Streams Enables Proactive Cache

```
while (i < N)
  sum += A[i];
  i++;
```

Proactive Cache System:
Driven by streams, proactively transfer.
Explore rich ISA semantics in the cache system.