Jenga: Software-Defined Cache Hierarchies

Po-An Tsai, Nathan Beckmann, and Daniel Sanchez

MIT CSAIL

Carnegie Mellon University School of Computer Science

Executive summary

- Heterogeneous caches are traditionally organized as a **rigid** hierarchy
  - Easy to program but introduce expensive overheads when hierarchy is not helpful

- Jenga builds **application-specific** cache hierarchies on the fly

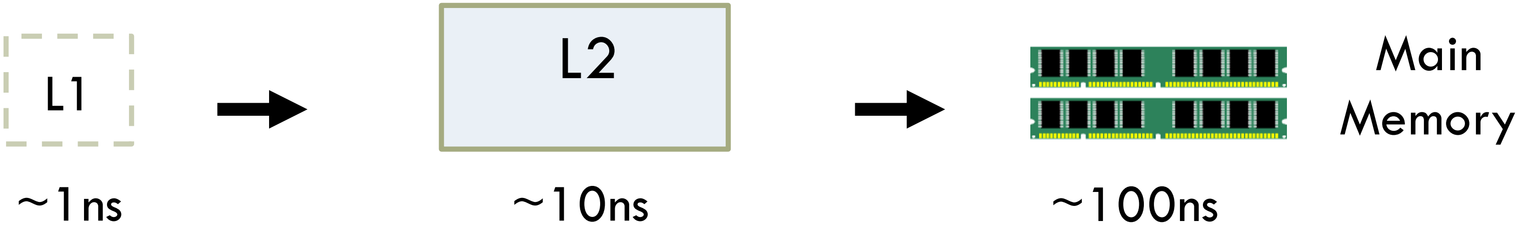
- Key contribution: New algorithms to find near-optimal hierarchies
  - Arbitrary application behaviors & changing resource constraints
  - Full system optimization at 36 cores in <1 ms

- Jenga improves EDP by up to 85% vs. state-of-the-art

Deep, rigid hierarchies are running out of steam

Deep, rigid hierarchies are running out of steam

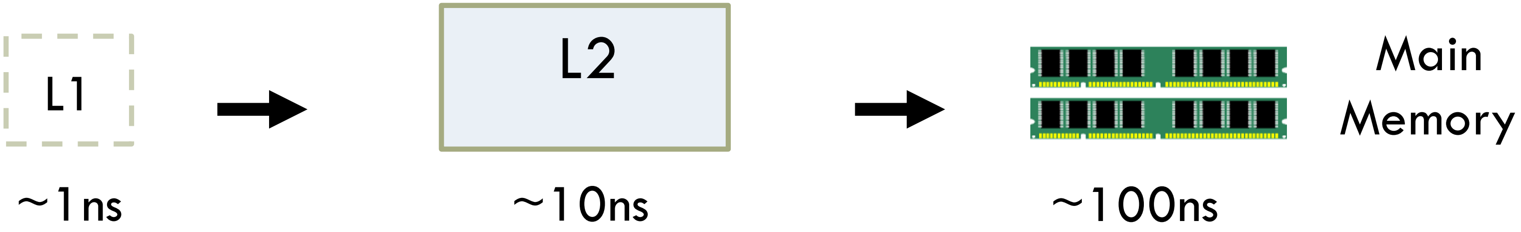
Past



Systems had few cache levels with widely different sizes and latencies

Deep, rigid hierarchies are running out of steam

Past



L1 → L2 → Main Memory

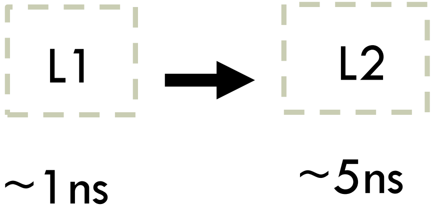
~1ns

~10ns

~100ns

Systems had few cache levels with widely different sizes and latencies

Now



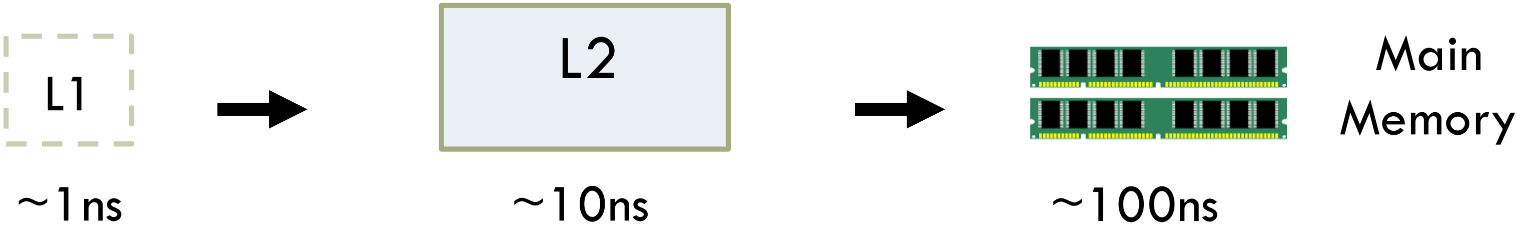
L1 → L2

~1ns

~5ns

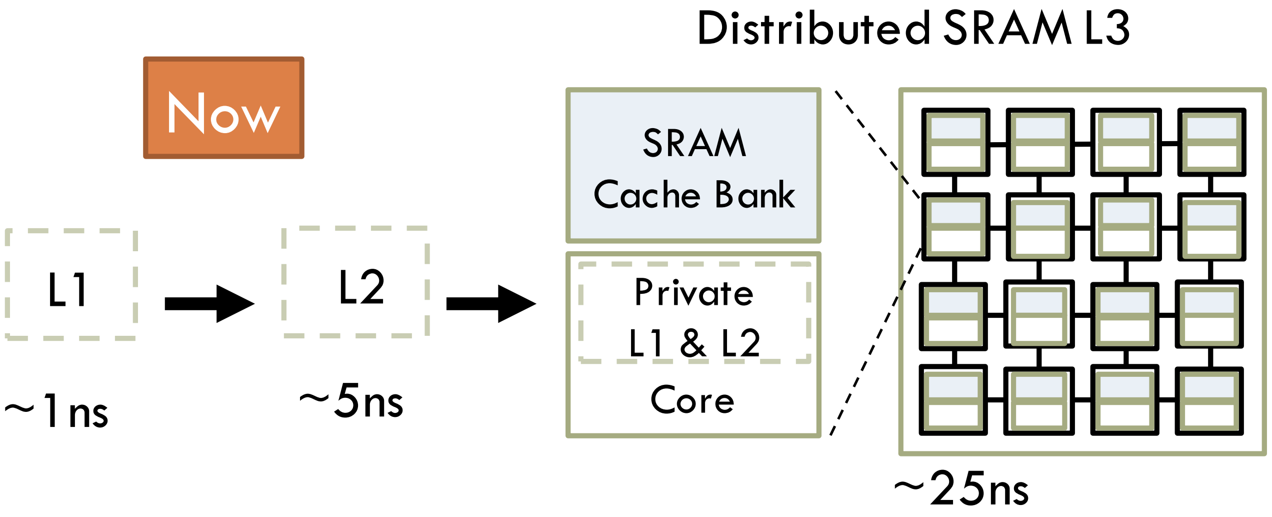
Deep, rigid hierarchies are running out of steam

Past



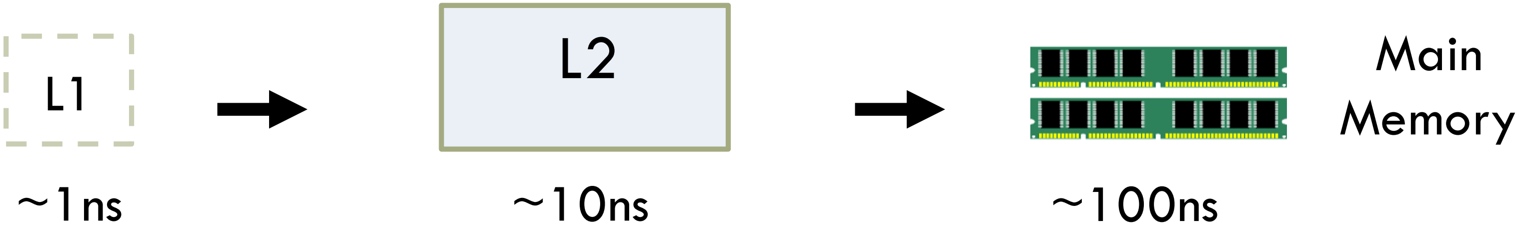
Systems had few cache levels with widely different sizes and latencies

Now



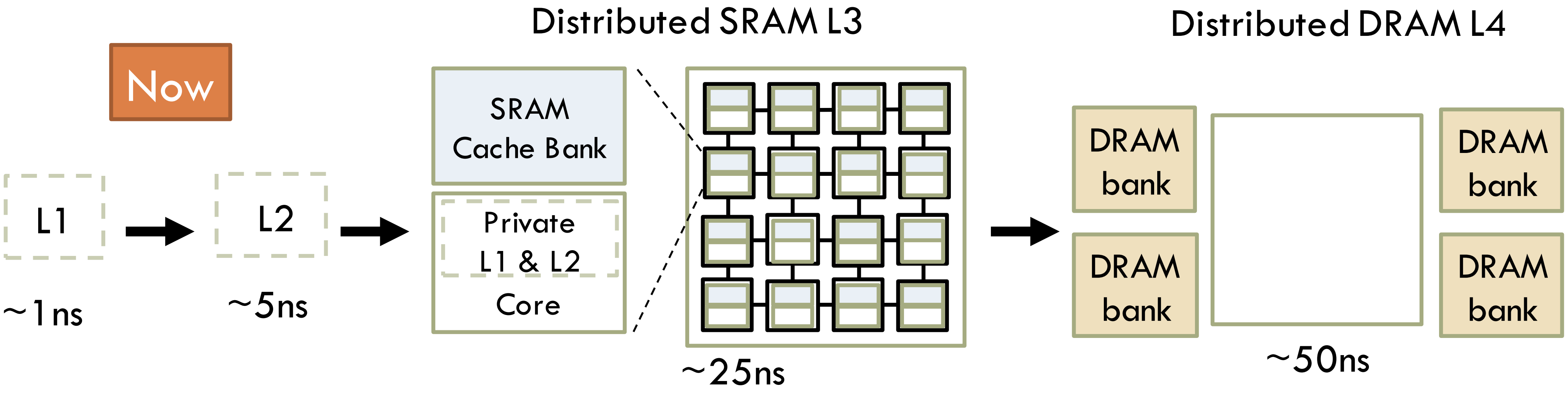
Deep, rigid hierarchies are running out of steam

Past



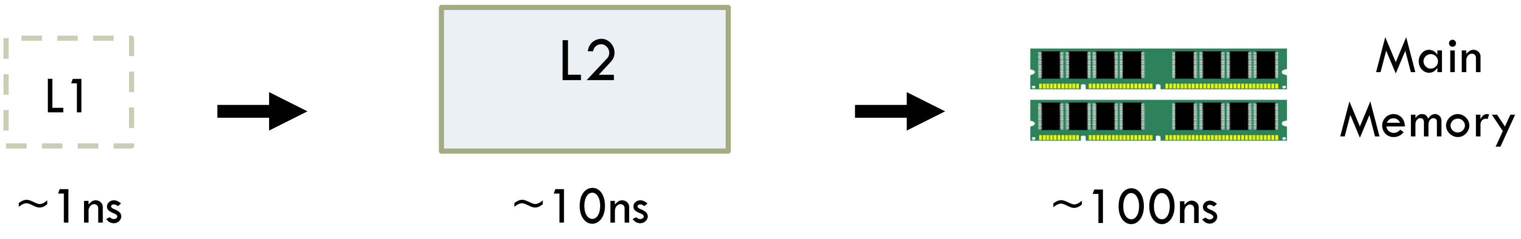
Systems had few cache levels with widely different sizes and latencies

Now



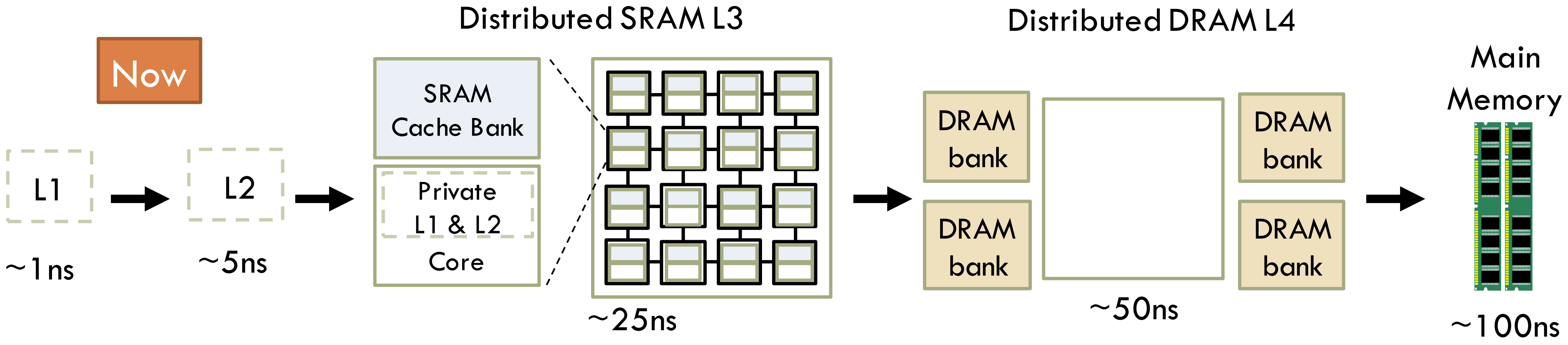
Deep, rigid hierarchies are running out of steam

Past



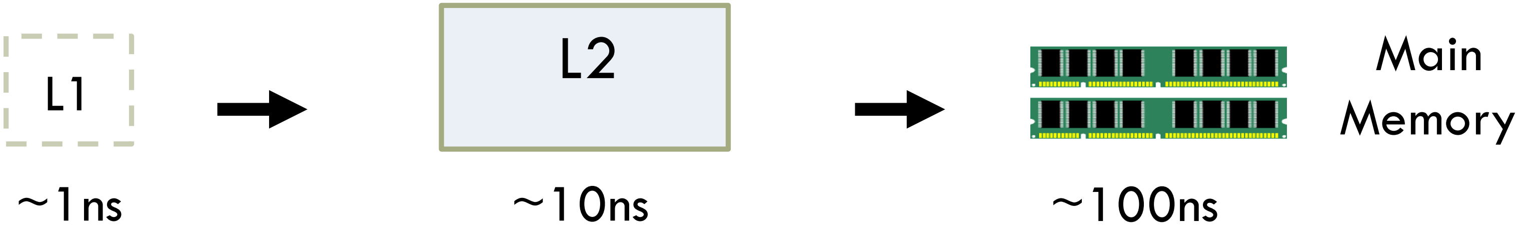
Systems had few cache levels with widely different sizes and latencies

Now



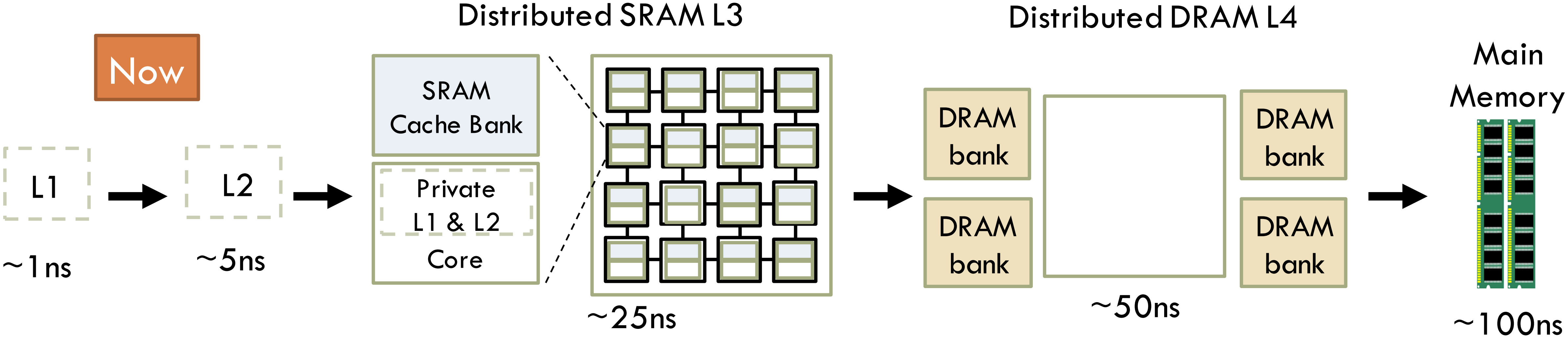
Deep, rigid hierarchies are running out of steam

Past



Systems had few cache levels with widely different sizes and latencies

Now



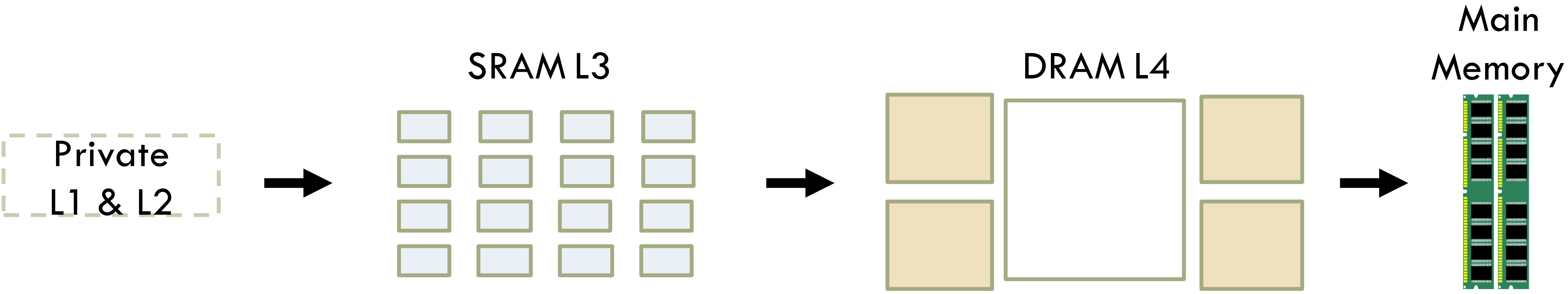
Higher overheads due to closer sizes and latencies across hierarchy levels

Rigid hierarchies must cater to the conflicting needs of many applications

- App 1: Scan through a 256MB array repeatedly

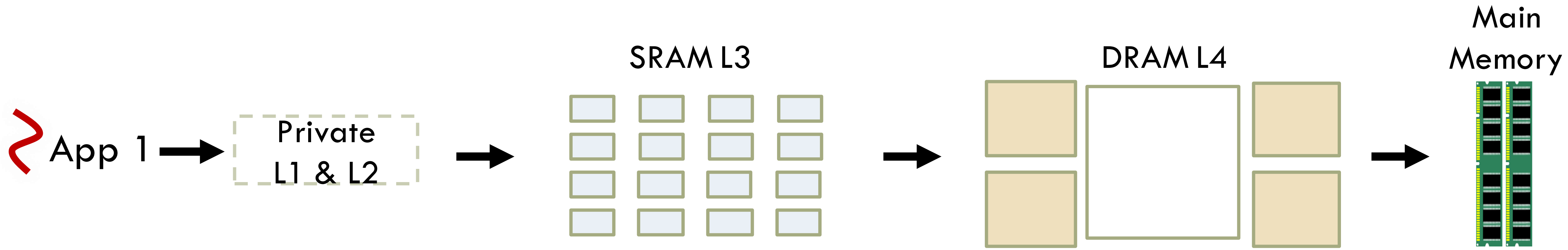
Rigid hierarchies must cater to the conflicting needs of many applications

- App 1: Scan through a 256MB array repeatedly



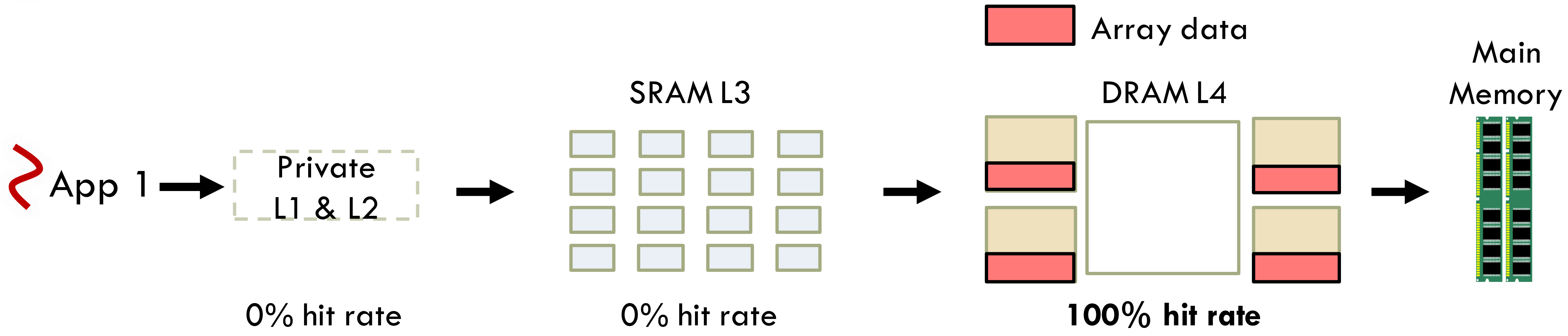
Rigid hierarchies must cater to the conflicting needs of many applications

- App 1: Scan through a 256MB array repeatedly



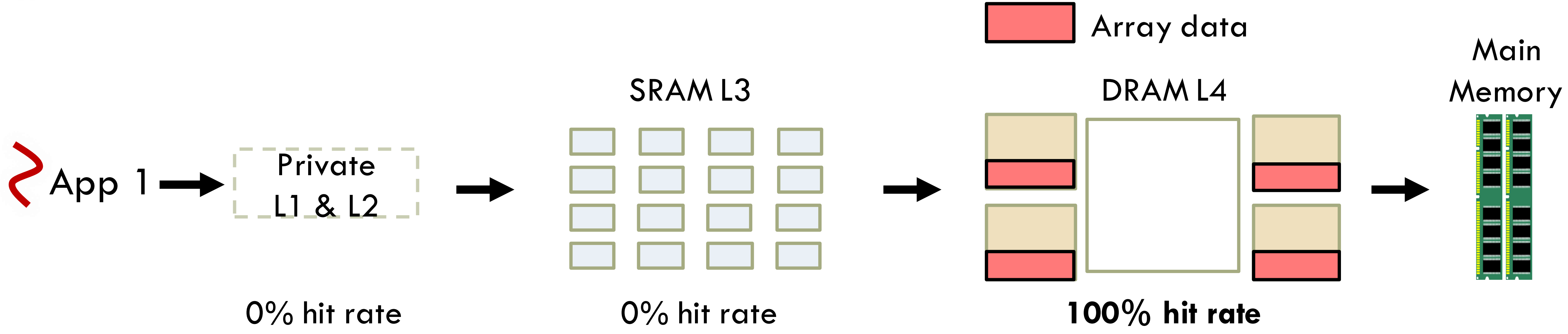
Rigid hierarchies must cater to the conflicting needs of many applications

- App 1: Scan through a 256MB array repeatedly



Rigid hierarchies must cater to the conflicting needs of many applications

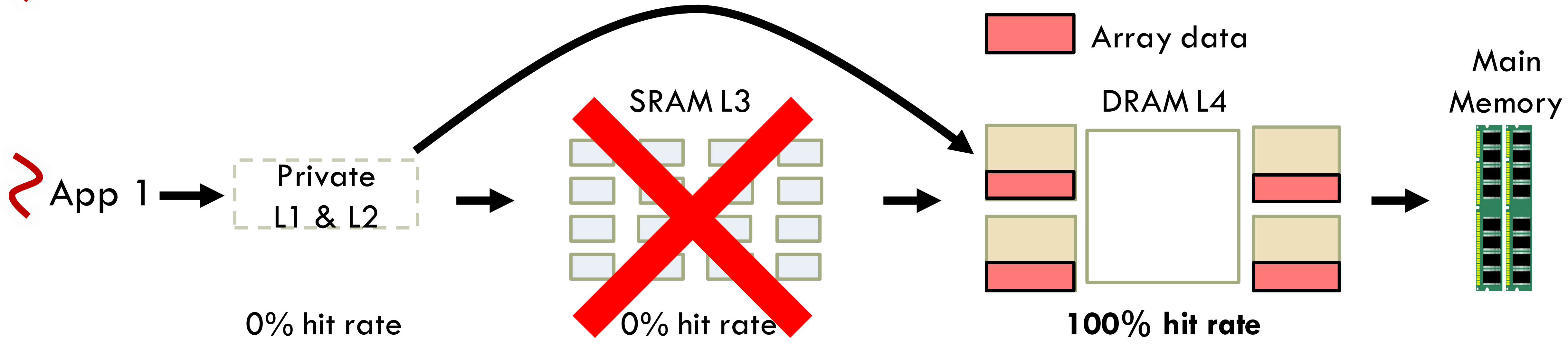
- App 1: Scan through a 256MB array repeatedly



Hit latency = ~5ns + ~25ns + ~50ns **= ~80ns**

Rigid hierarchies must cater to the conflicting needs of many applications

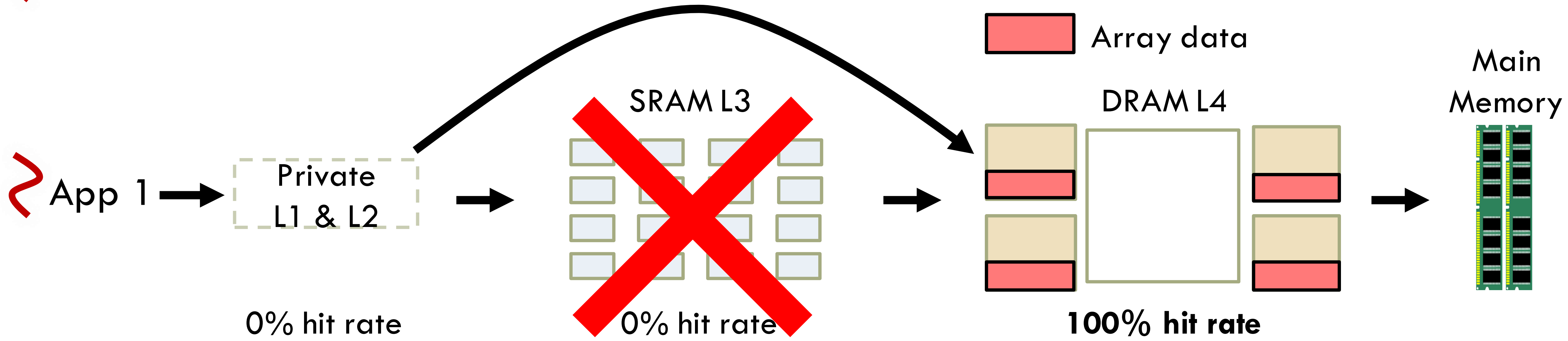
- App 1: Scan through a 256MB array repeatedly



Hit latency = ∼5ns + ∼~~25~~ns + ∼50ns **= ∼80ns**

Rigid hierarchies must cater to the conflicting needs of many applications

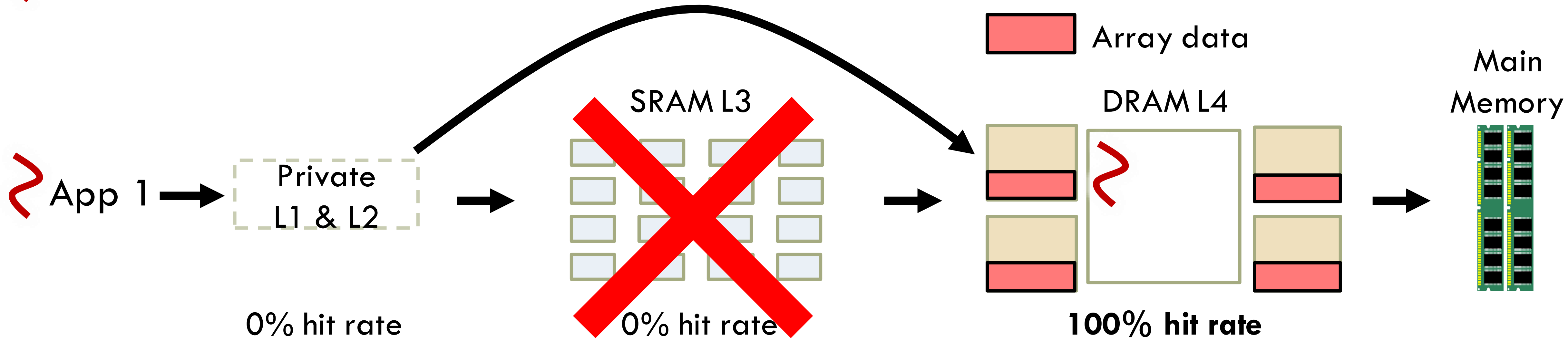
- App 1: Scan through a 256MB array repeatedly



Hit latency =	~5ns	+	~~~25~~ns	+	~50ns	**= ~80ns**
Hit latency =	~5ns	+	0ns	+	~50ns	**= ~55ns (30% lower)**

Rigid hierarchies must cater to the conflicting needs of many applications

- App 1: Scan through a 256MB array repeatedly

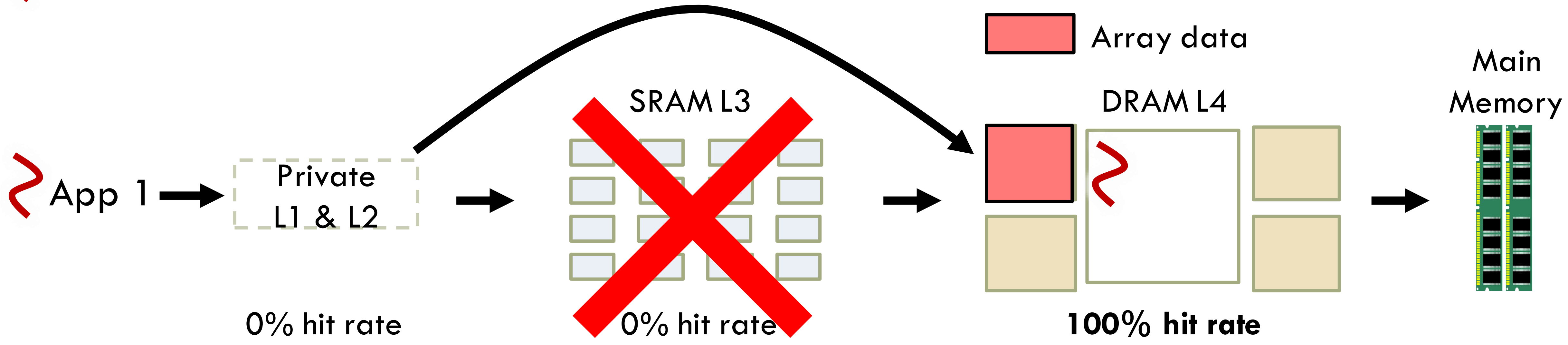


Hit latency = ~5ns + ∼~~25~~ns + ~50ns **= ~80ns**

Hit latency = ~5ns + 0ns + ~50ns **= ~55ns (30% lower)**

Rigid hierarchies must cater to the conflicting needs of many applications

- App 1: Scan through a 256MB array repeatedly



🟥 Array data

App 1 → Private L1 & L2 → SRAM L3 → DRAM L4 → Main Memory

0% hit rate

0% hit rate

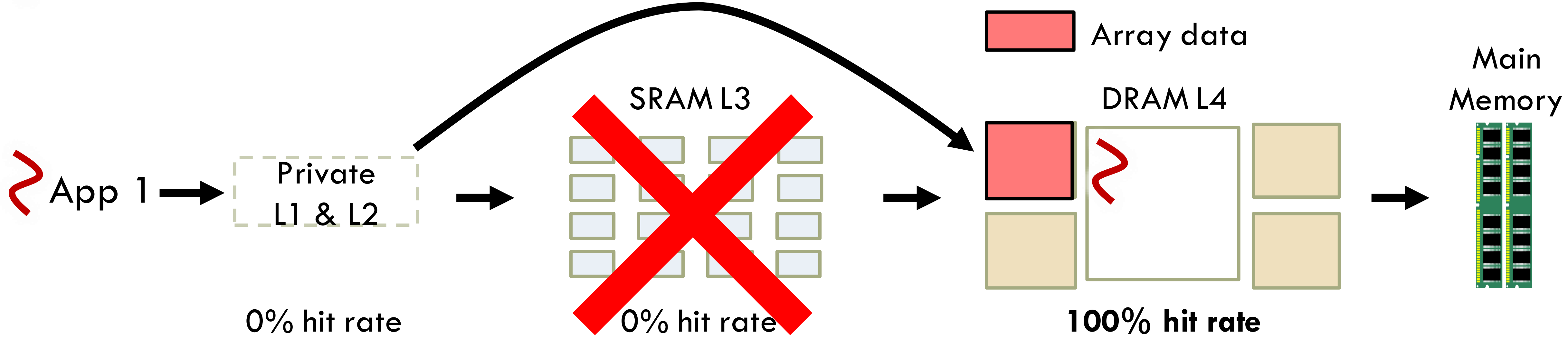
100% hit rate

Hit latency = ~5ns + ~25ns + ~50ns **= ~80ns**

Hit latency = ~5ns + 0ns + ~50ns **= ~55ns (30% lower)**

Rigid hierarchies must cater to the conflicting needs of many applications

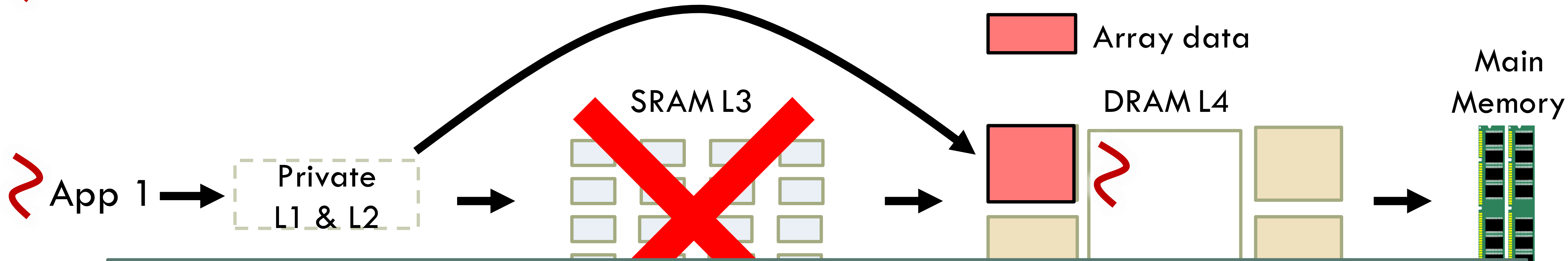
App 1: Scan through a 256MB array repeatedly



Hit latency =	~5ns	+	~~~25~~ns	+	~50ns	**= ~80ns**
Hit latency =	~5ns	+	0ns	+	~~~50~~ns	**= ~55ns (30% lower)**
Hit latency =	~5ns	+	0ns	+	~40ns	**= ~45ns (45% lower)**

Rigid hierarchies must cater to the conflicting needs of many applications

- App 1: Scan through a 256MB array repeatedly



Even the best rigid hierarchy is a bad compromise!
(See paper for details)

Hit la

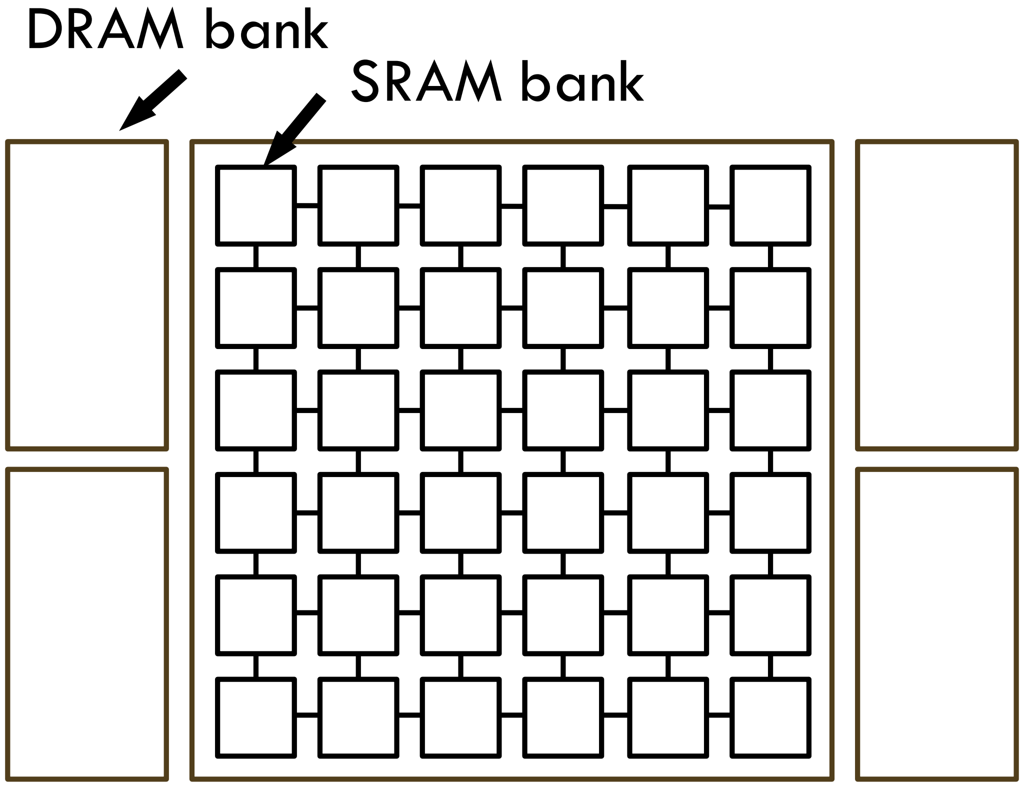
Hit latency = ~5ns + 0ns + ~5~~0~~ns **= ~55ns (30% lower)**

Hit latency = ~5ns + 0ns + ~40ns **= ~45ns (45% lower)**

Jenga: Software-defined cache hierarchies

Jenga: Software-defined cache hierarchies

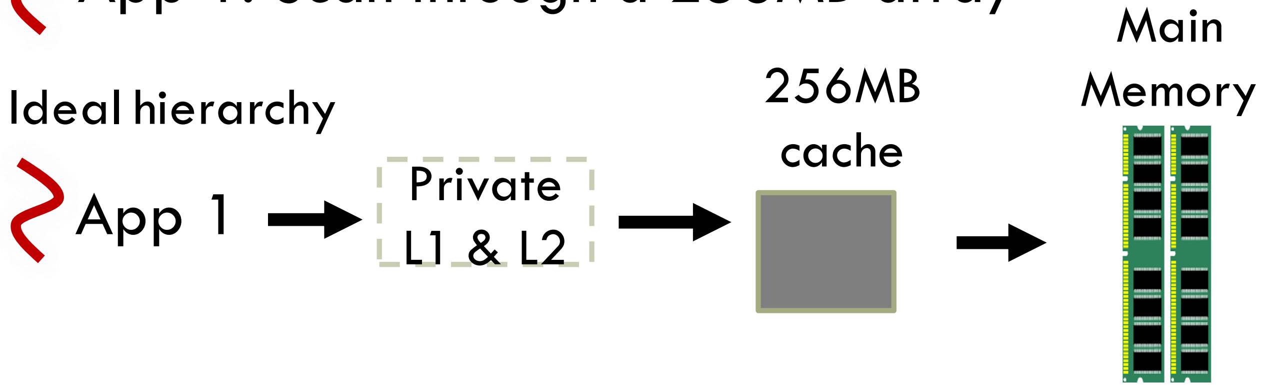
Jenga manages distributed and heterogeneous banks as a single resource pool and builds **virtual hierarchies** tailored to each application in the system.

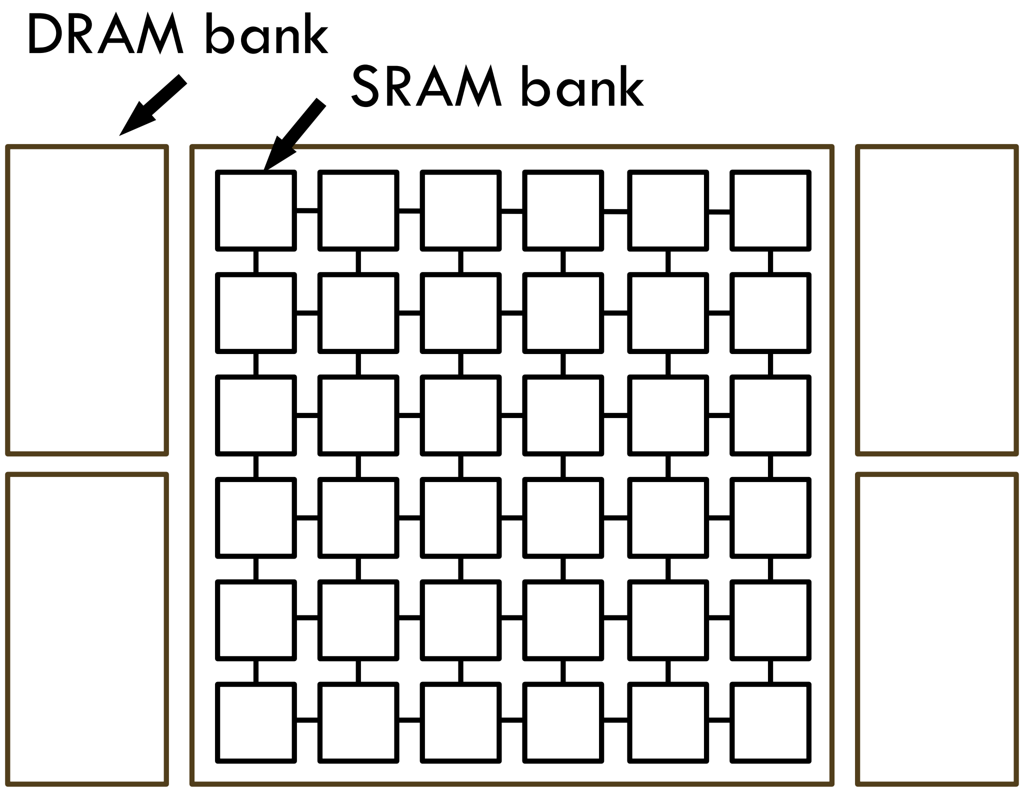


Jenga: Software-defined cache hierarchies

Jenga manages distributed and heterogeneous banks as a single resource pool and builds **virtual hierarchies** tailored to each application in the system.

App 1: Scan through a 256MB array

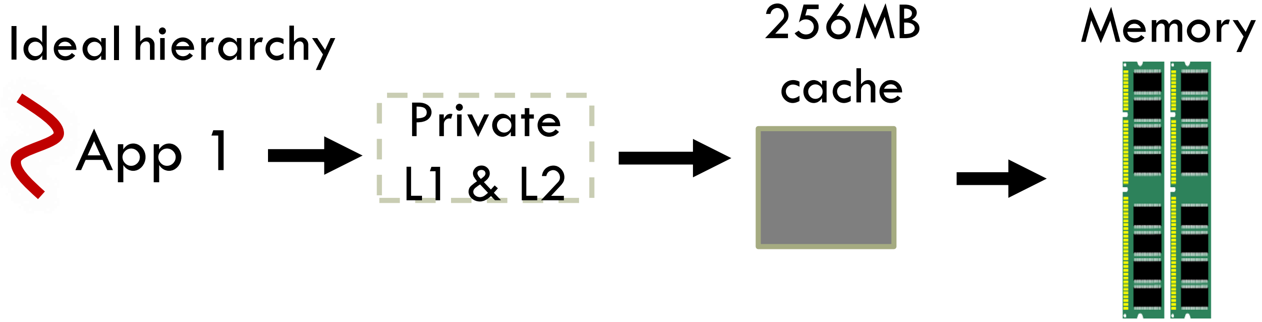


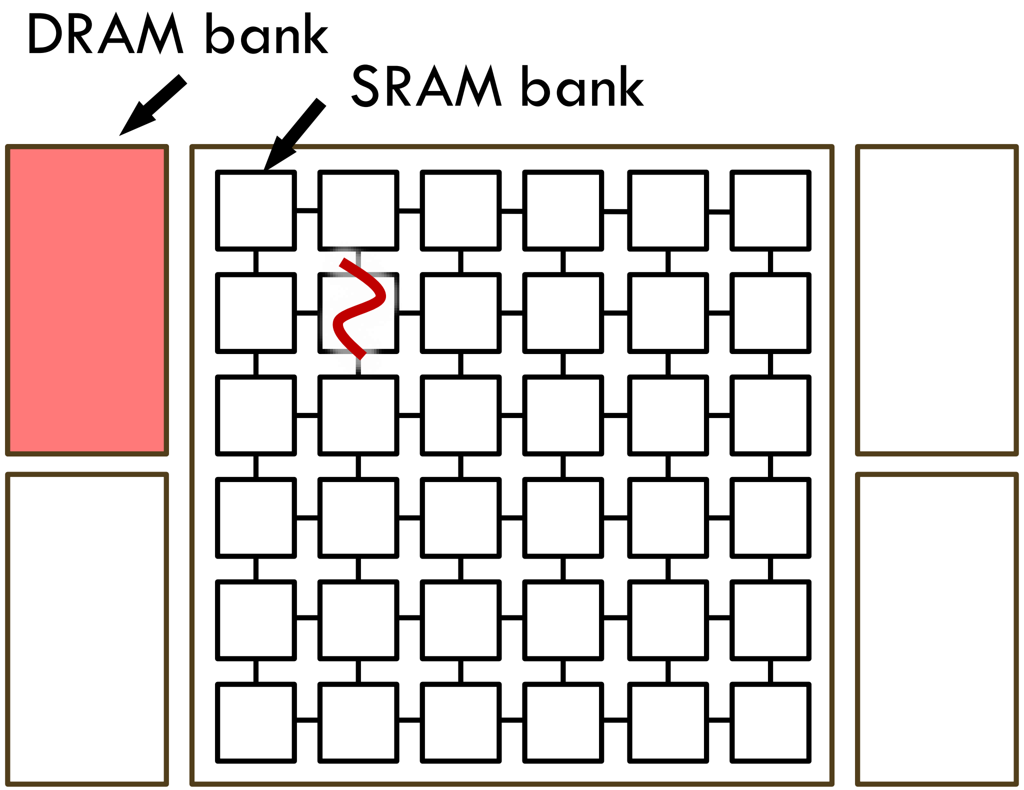


Jenga: Software-defined cache hierarchies

Jenga manages distributed and heterogeneous banks as a single resource pool and builds **virtual hierarchies** tailored to each application in the system.

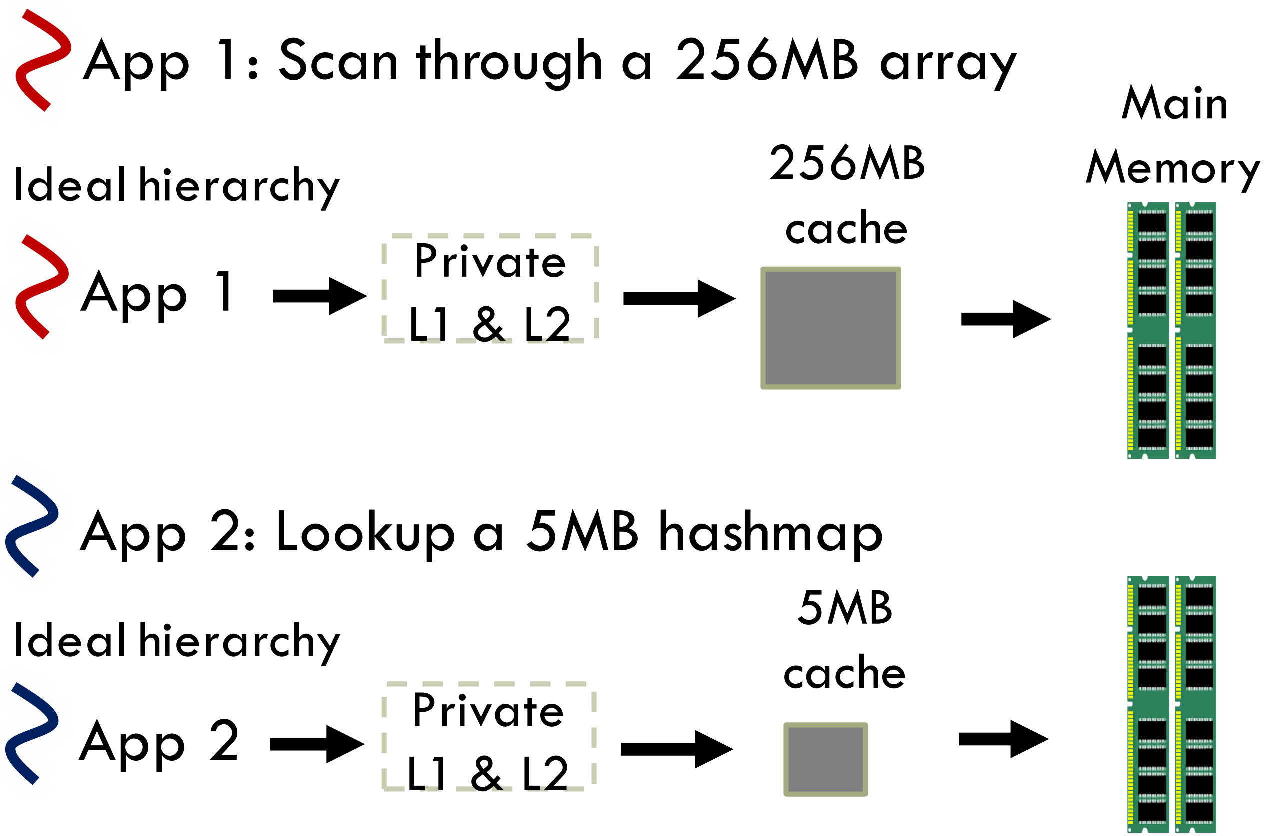
App 1: Scan through a 256MB array





Jenga: Software-defined cache hierarchies

Jenga manages distributed and heterogeneous banks as a single resource pool and builds **virtual hierarchies** tailored to each application in the system.



App 1: Scan through a 256MB array

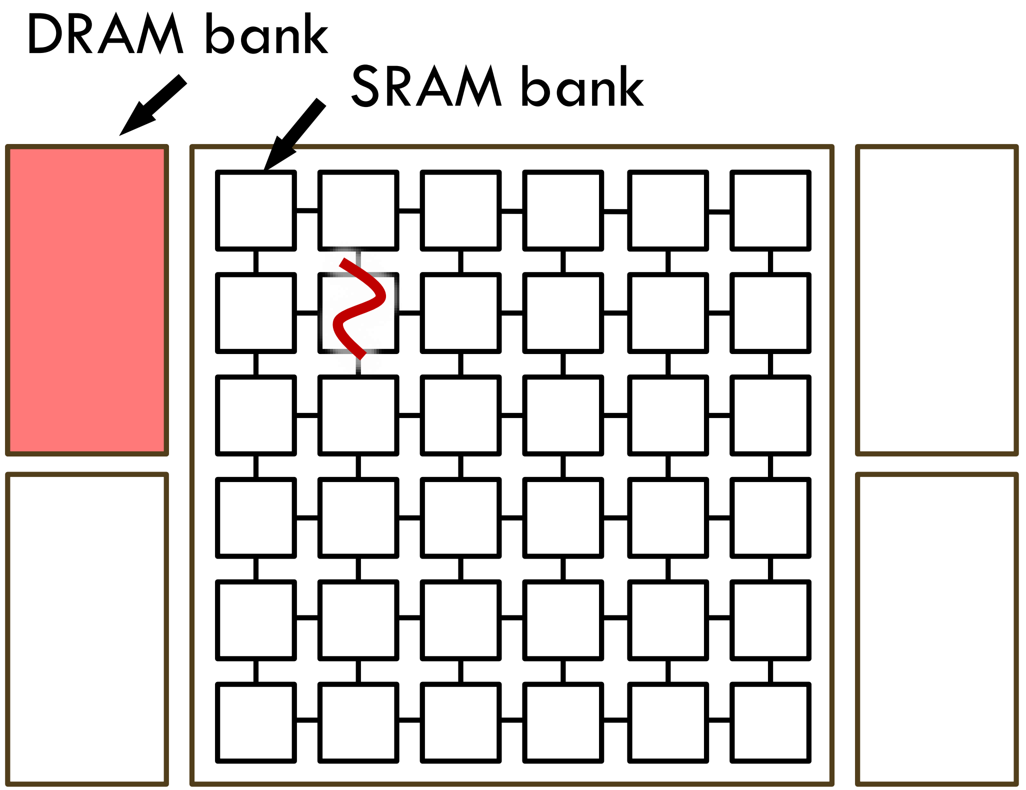
Ideal hierarchy

App 1 → Private L1 & L2 → 256MB cache → Main Memory

App 2: Lookup a 5MB hashmap

Ideal hierarchy

App 2 → Private L1 & L2 → 5MB cache →

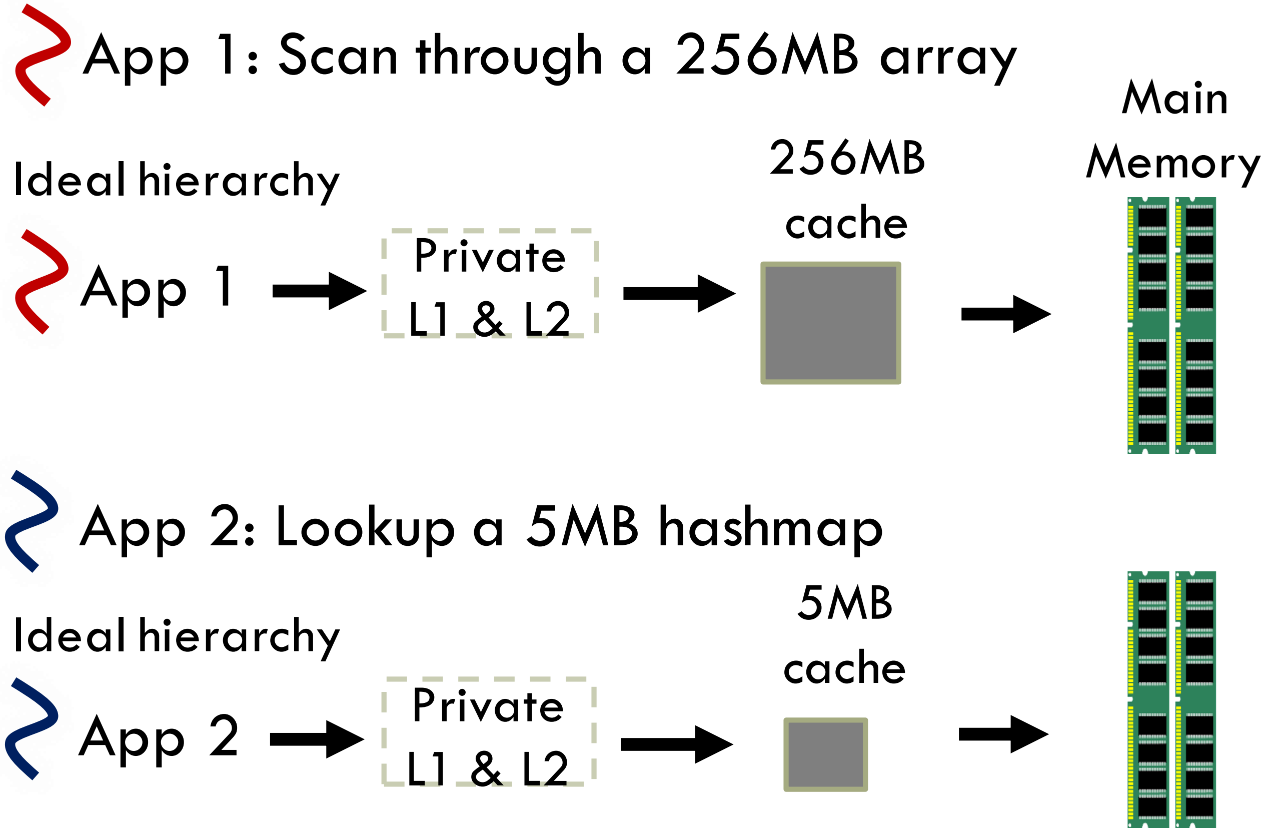


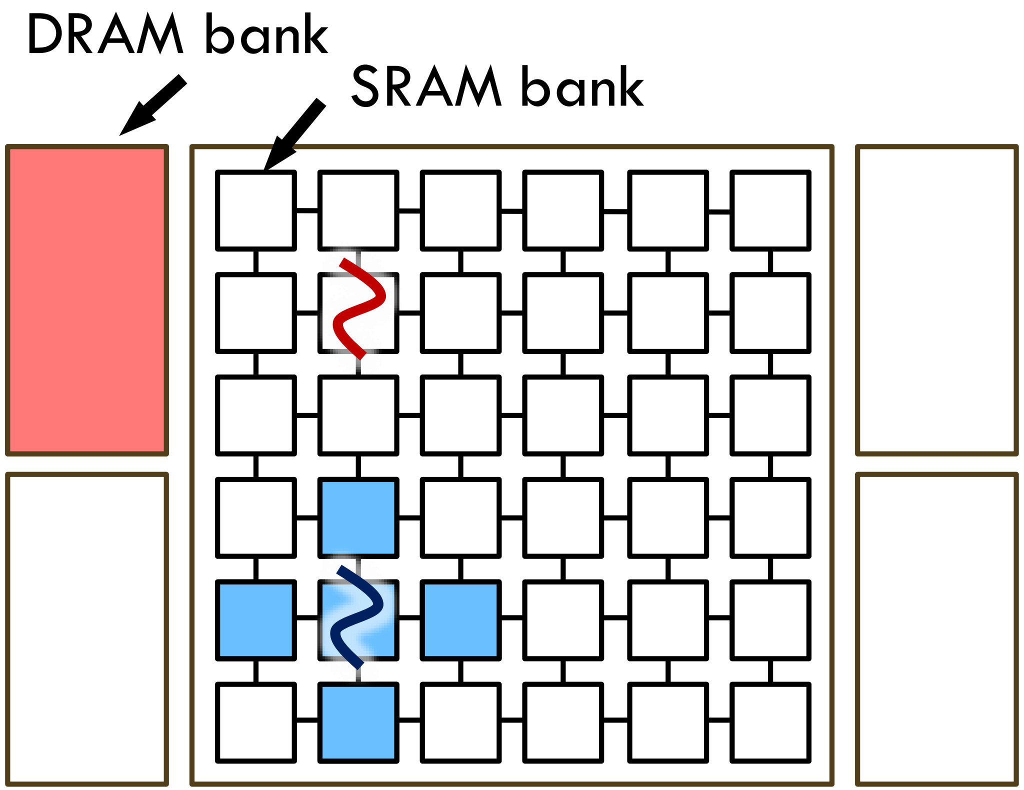
DRAM bank

SRAM bank

Jenga: Software-defined cache hierarchies

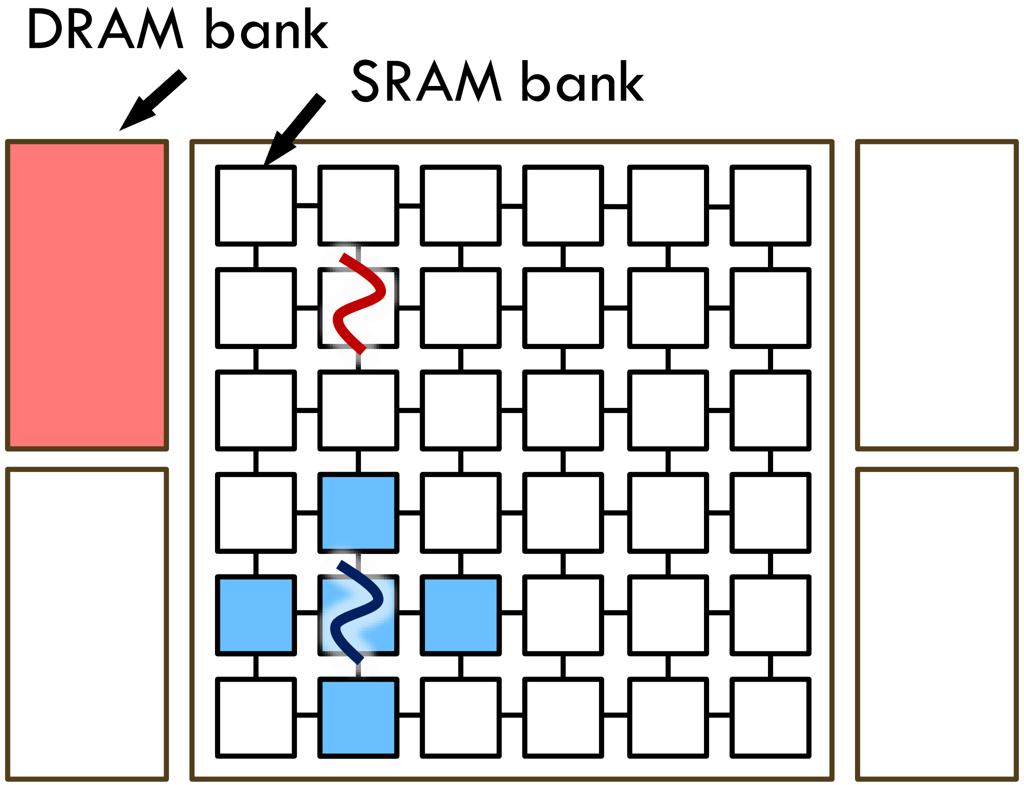
Jenga manages distributed and heterogeneous banks as a single resource pool and builds **virtual hierarchies** tailored to each application in the system.





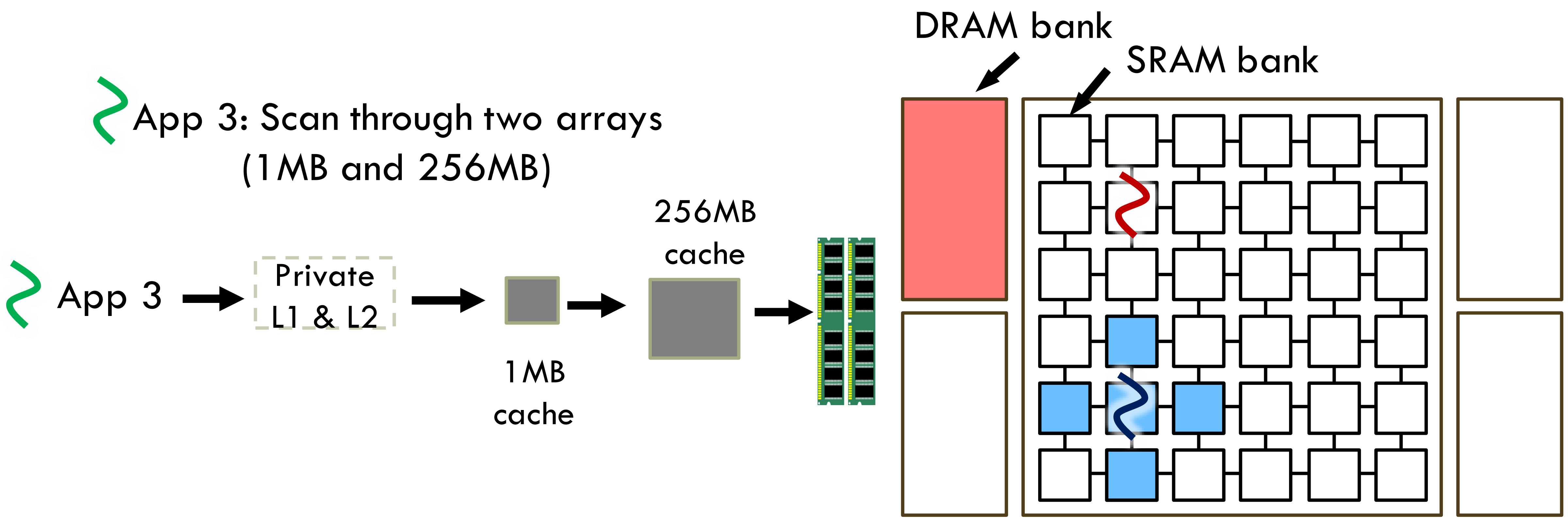
Jenga: Software-defined cache hierarchies

Jenga manages distributed and heterogeneous banks as a single resource pool and builds **virtual hierarchies** tailored to each application in the system.



Jenga: Software-defined cache hierarchies

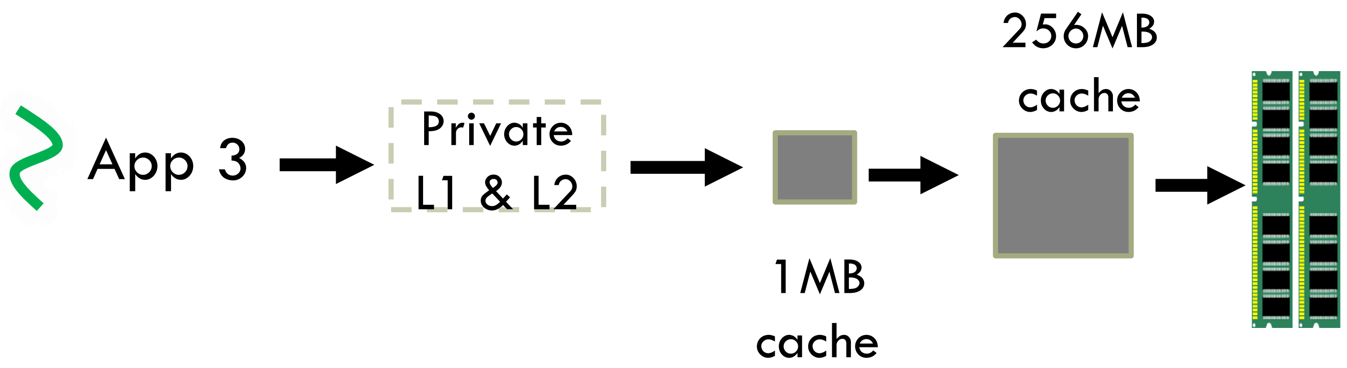
Jenga manages distributed and heterogeneous banks as a single resource pool and builds **virtual hierarchies** tailored to each application in the system.

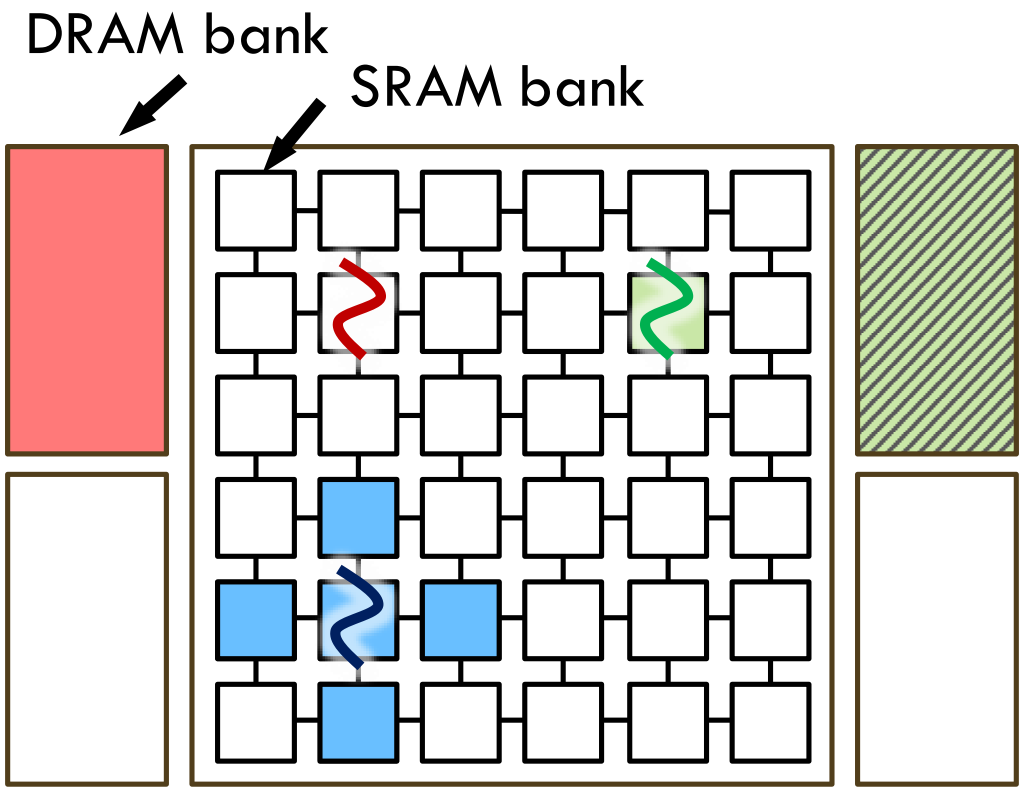


Jenga: Software-defined cache hierarchies

Jenga manages distributed and heterogeneous banks as a single resource pool and builds **virtual hierarchies** tailored to each application in the system.

App 3: Scan through two arrays (1MB and 256MB)

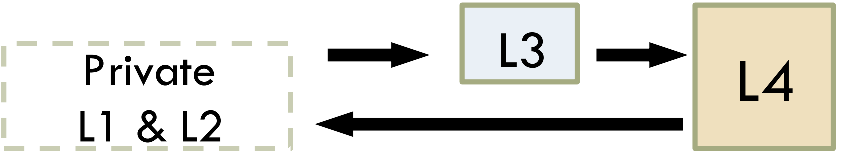




Prior work to mitigate the cost of rigid hierarchies

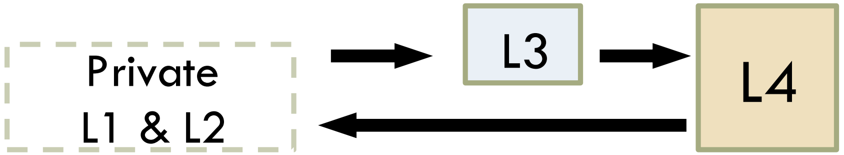
Prior work to mitigate the cost of rigid hierarchies

- Bypass levels to avoid cache pollutions
  - Do not install lines at specific levels
  - Give lines low priority in replacement policy

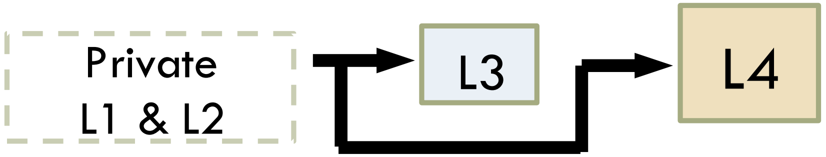


Prior work to mitigate the cost of rigid hierarchies

- Bypass levels to avoid cache pollutions
  - Do not install lines at specific levels
  - Give lines low priority in replacement policy

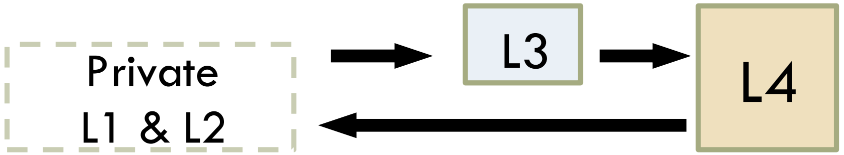


- Speculatively access up the hierarchy
  - Hit/miss predictors, prefetchers
  - Hide latency with speculative accesses

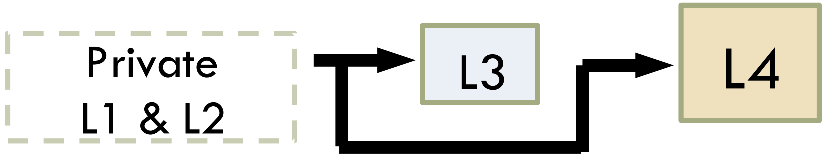


Prior work to mitigate the cost of rigid hierarchies

- Bypass levels to avoid cache pollutions
  - Do not install lines at specific levels
  - Give lines low priority in replacement policy



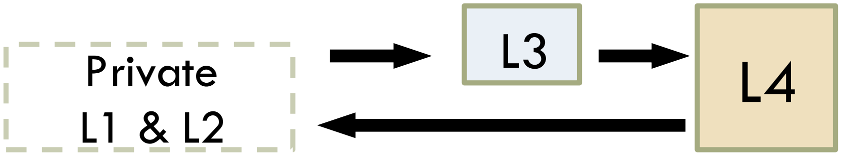
- Speculatively access up the hierarchy
  - Hit/miss predictors, prefetchers
  - Hide latency with speculative accesses



- They **must** still check all levels for correctness!
  - Waste energy and bandwidth

Prior work to mitigate the cost of rigid hierarchies

- □ Bypass levels to avoid cache pollutions
  - Do not install lines at specific levels
  - Give lines low priority in replacement policy



- □ Specu
  - Hit/
  - Hid

It’s better to build the right hierarchy and avoid the root cause: unnecessary accesses to unwanted cache levels

- □ They **must** still check all levels for correctness!
  - Waste energy and bandwidth

Jenga = flexible hardware + smart software

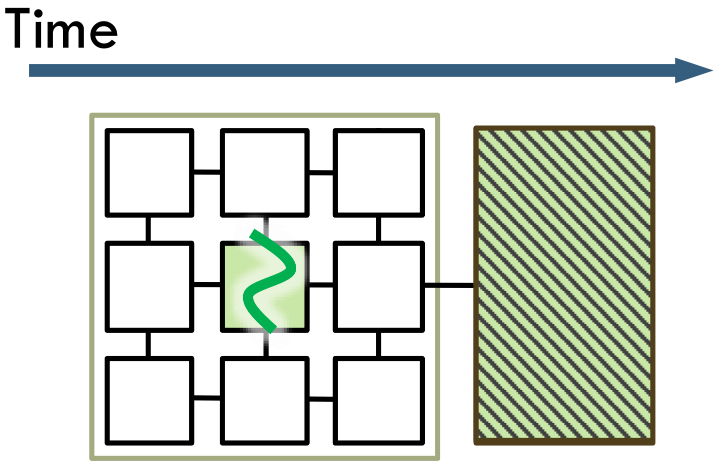
Software

Hardware

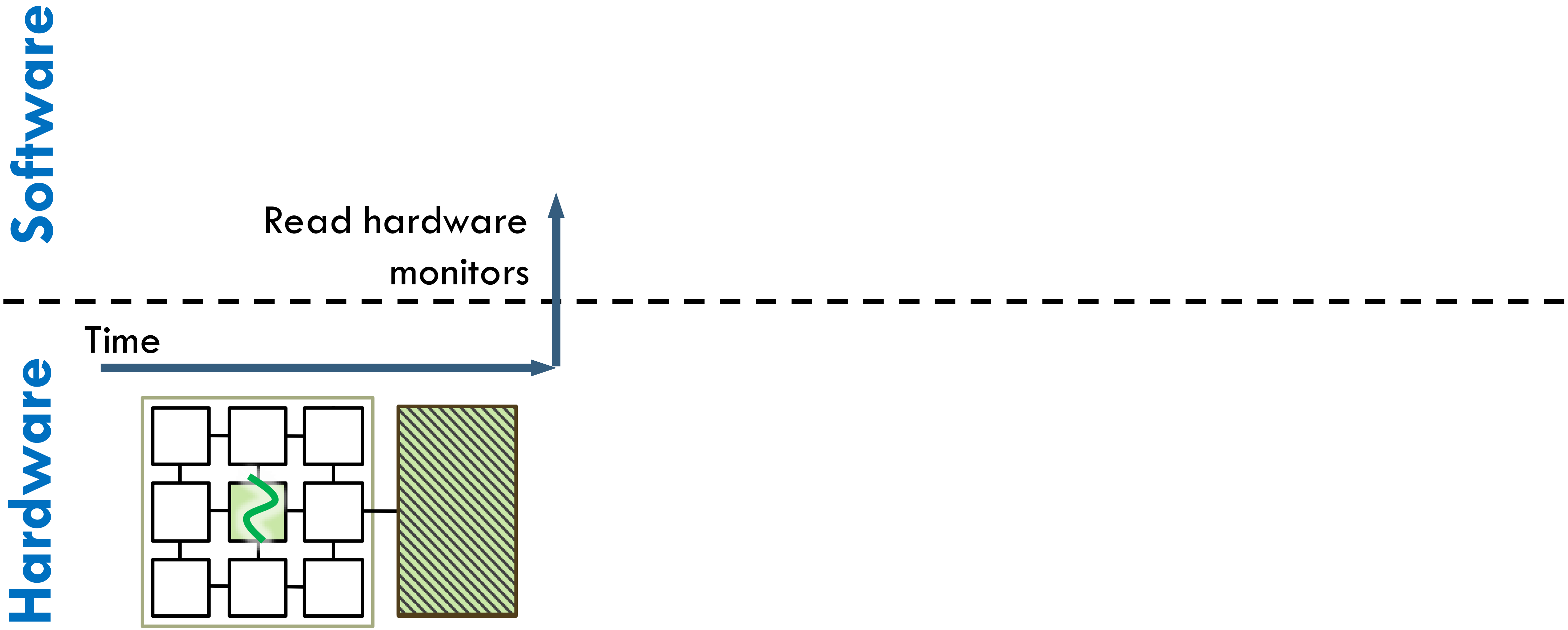
Jenga = flexible hardware + smart software

Software

Hardware



Jenga = flexible hardware + smart software



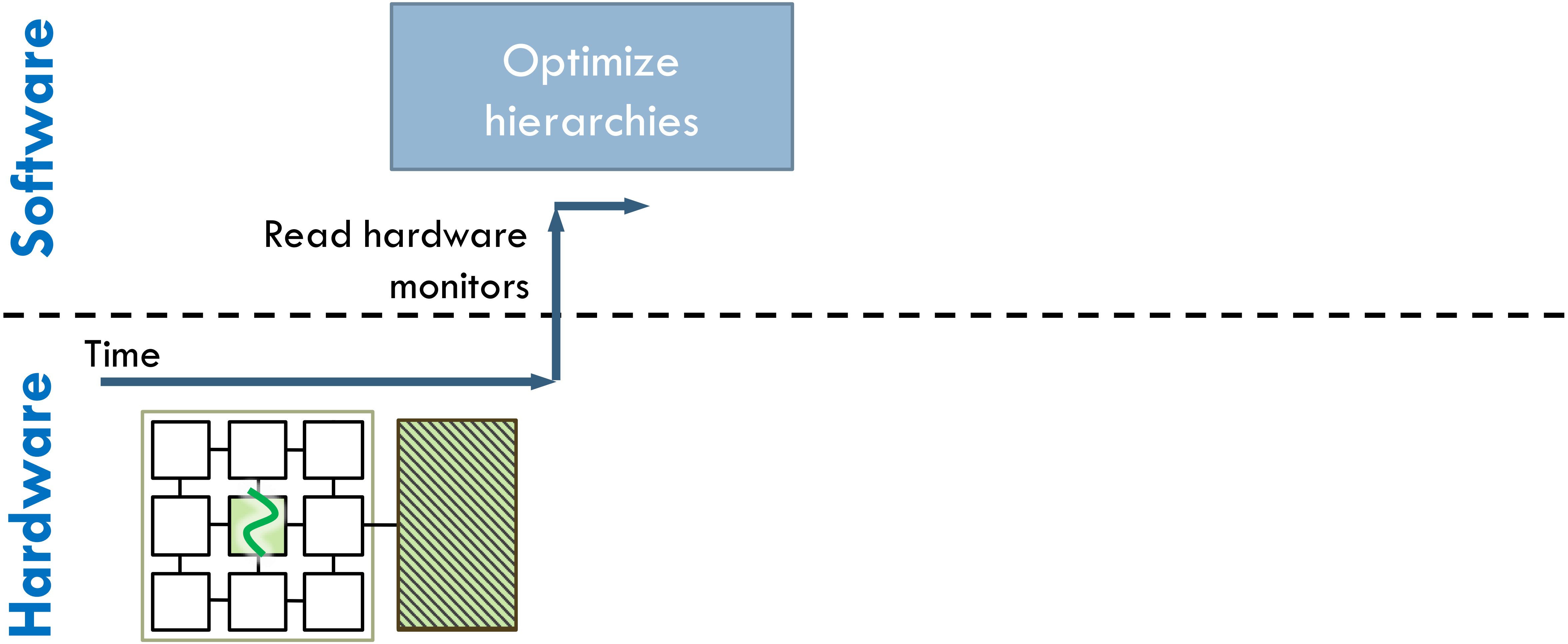
Software

Read hardware monitors

Time

Hardware

Jenga = flexible hardware + smart software



Software

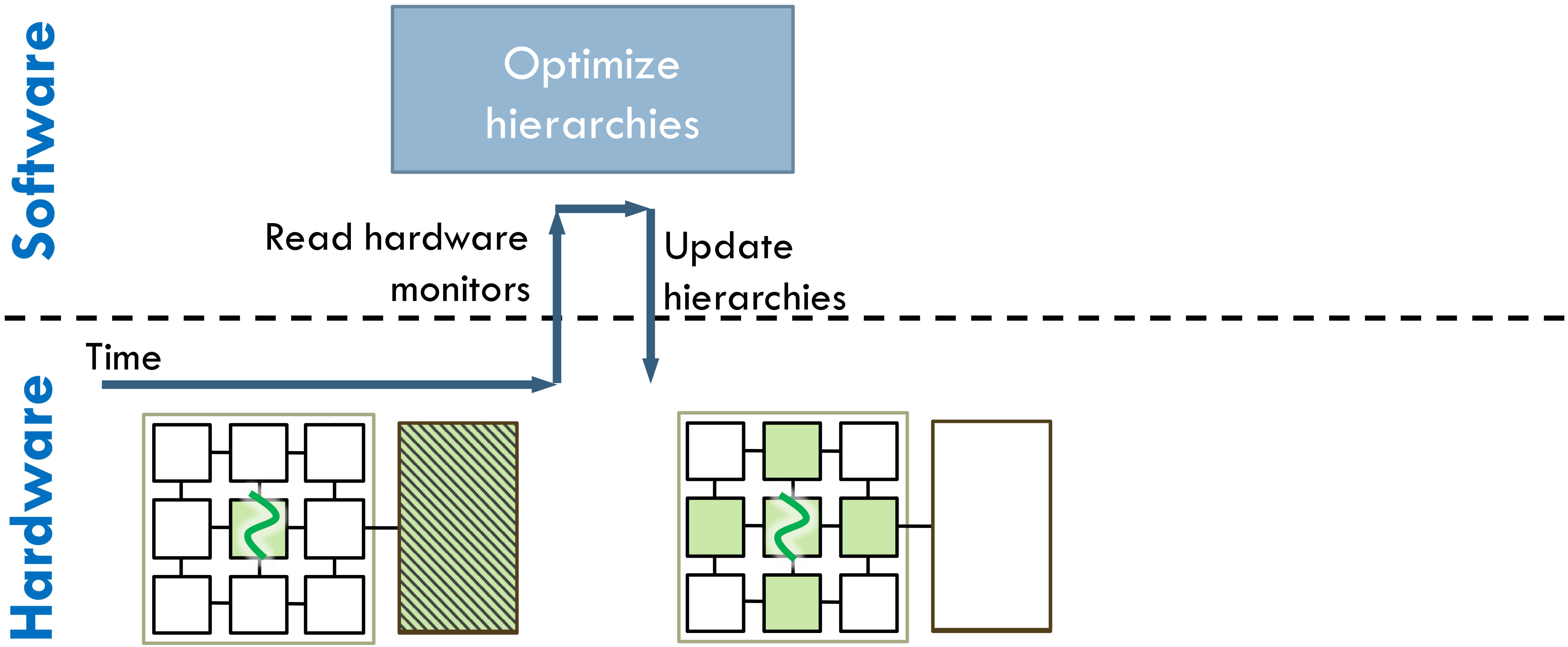
Optimize hierarchies

Read hardware monitors

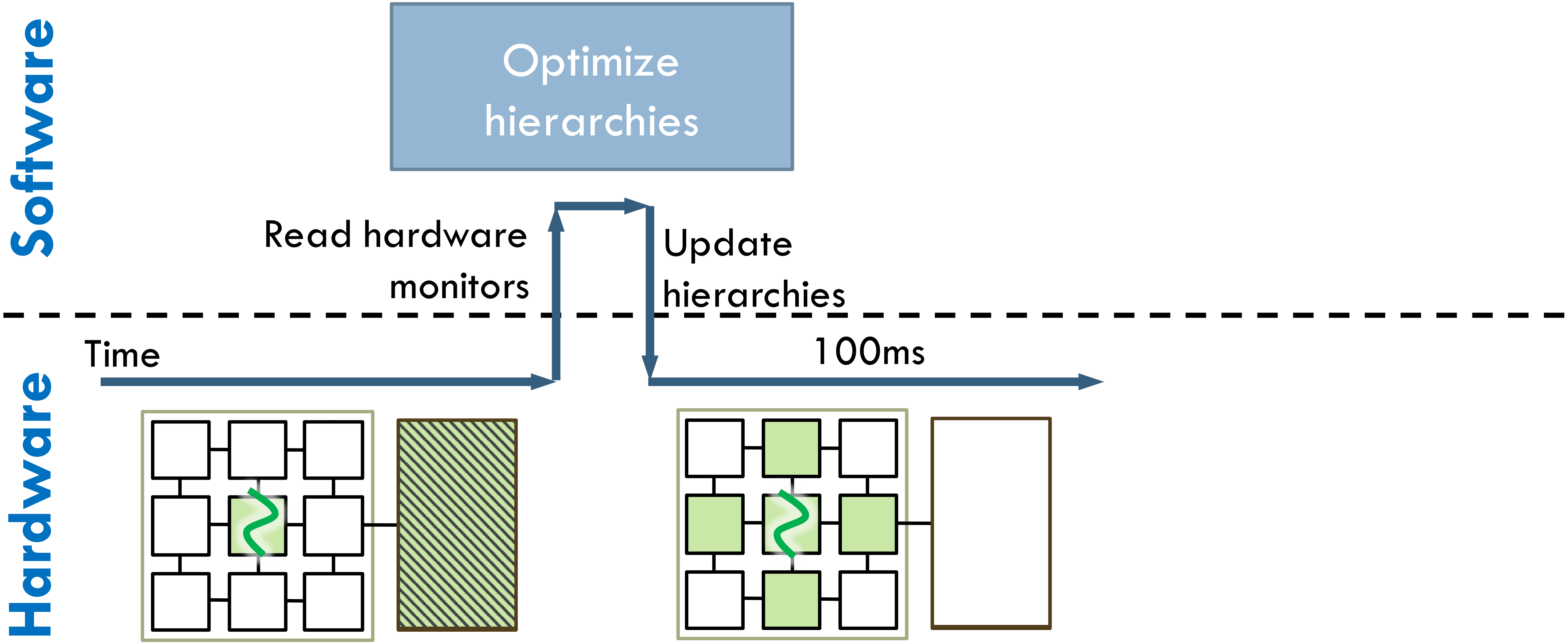
Time

Hardware

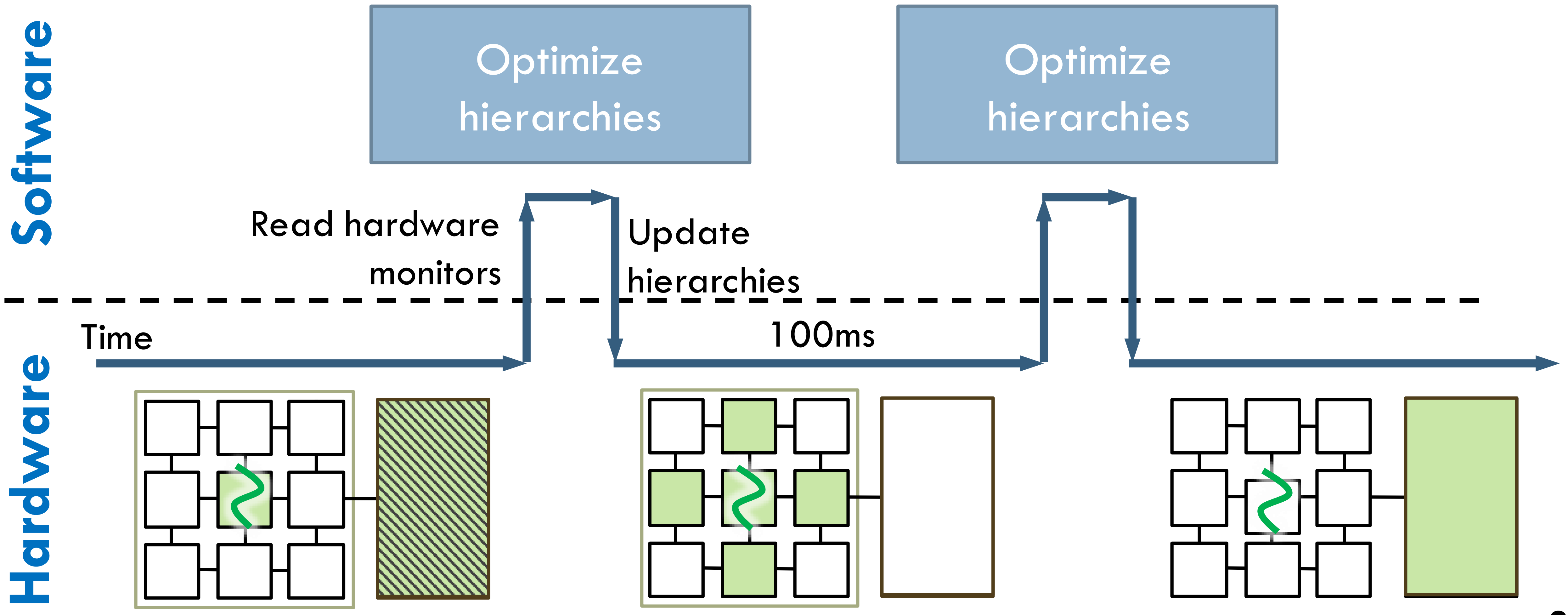
Jenga = flexible hardware + smart software



Jenga = flexible hardware + smart software



Jenga = flexible hardware + smart software

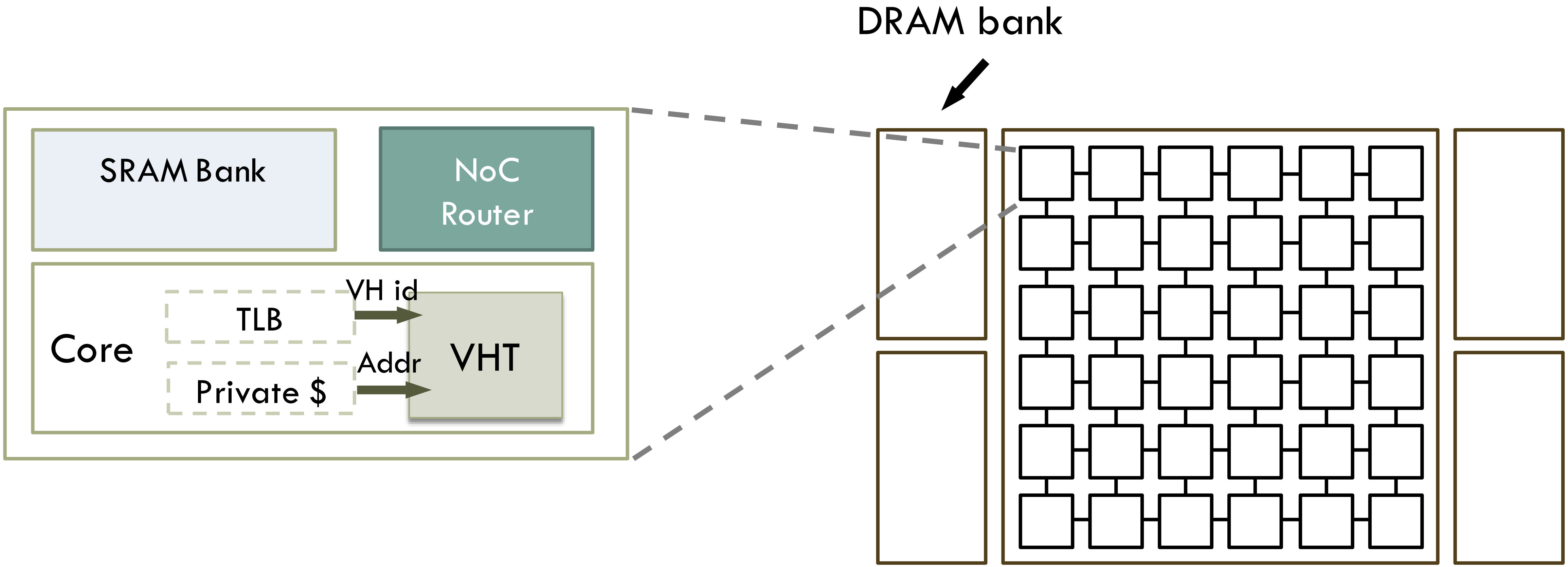


Jenga hardware: supporting virtual hierarchies (VHs)

- Cores consult **virtual hierarchy table (VHT)** to find the access path
  - Similar to Jigsaw [PACT'13, HPCA'15], but it supports two levels

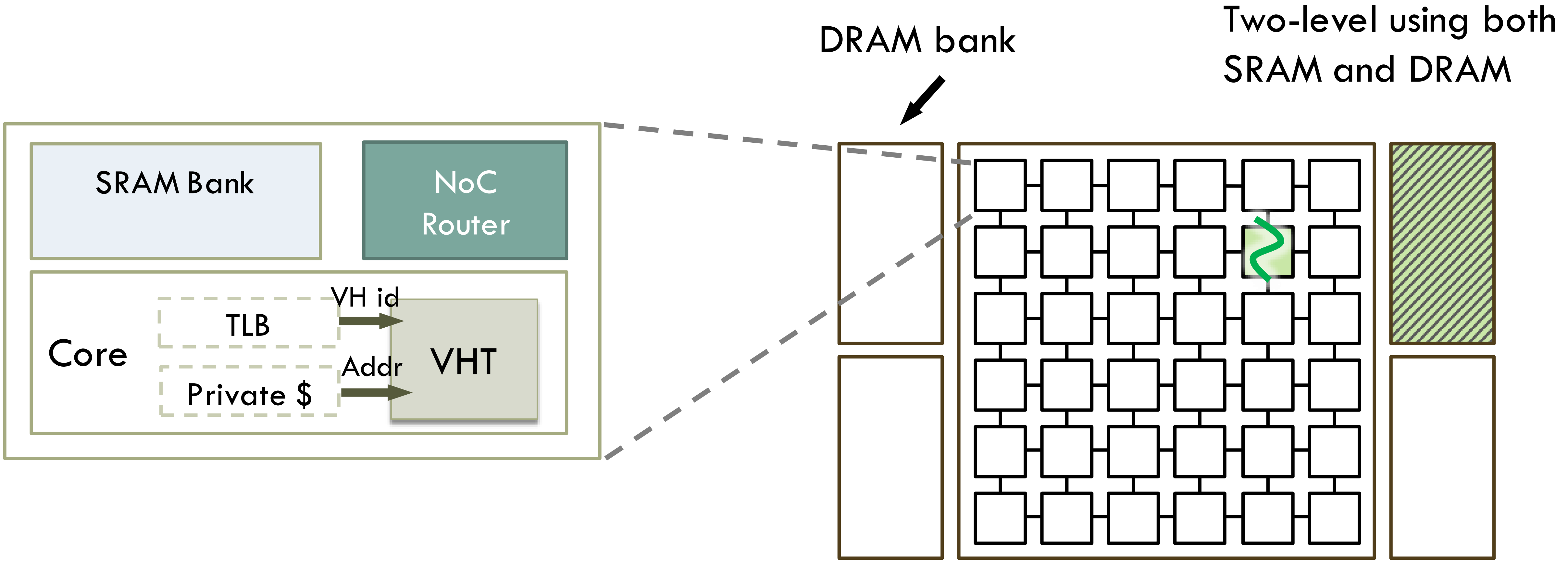
Jenga hardware: supporting virtual hierarchies (VHs)

- Cores consult **virtual hierarchy table (VHT)** to find the access path
  - Similar to Jigsaw [PACT'13, HPCA'15], but it supports two levels



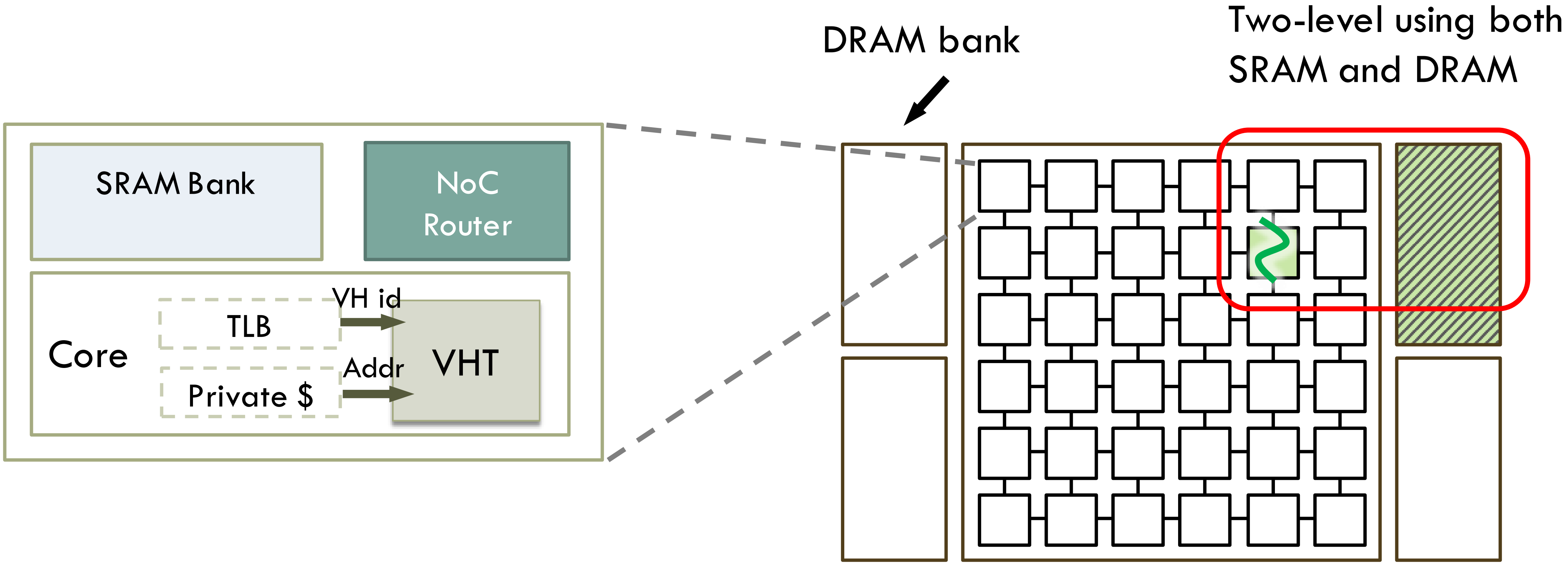
Jenga hardware: supporting virtual hierarchies (VHs)

- Cores consult **virtual hierarchy table (VHT)** to find the access path
  - Similar to Jigsaw [PACT'13, HPCA'15], but it supports two levels



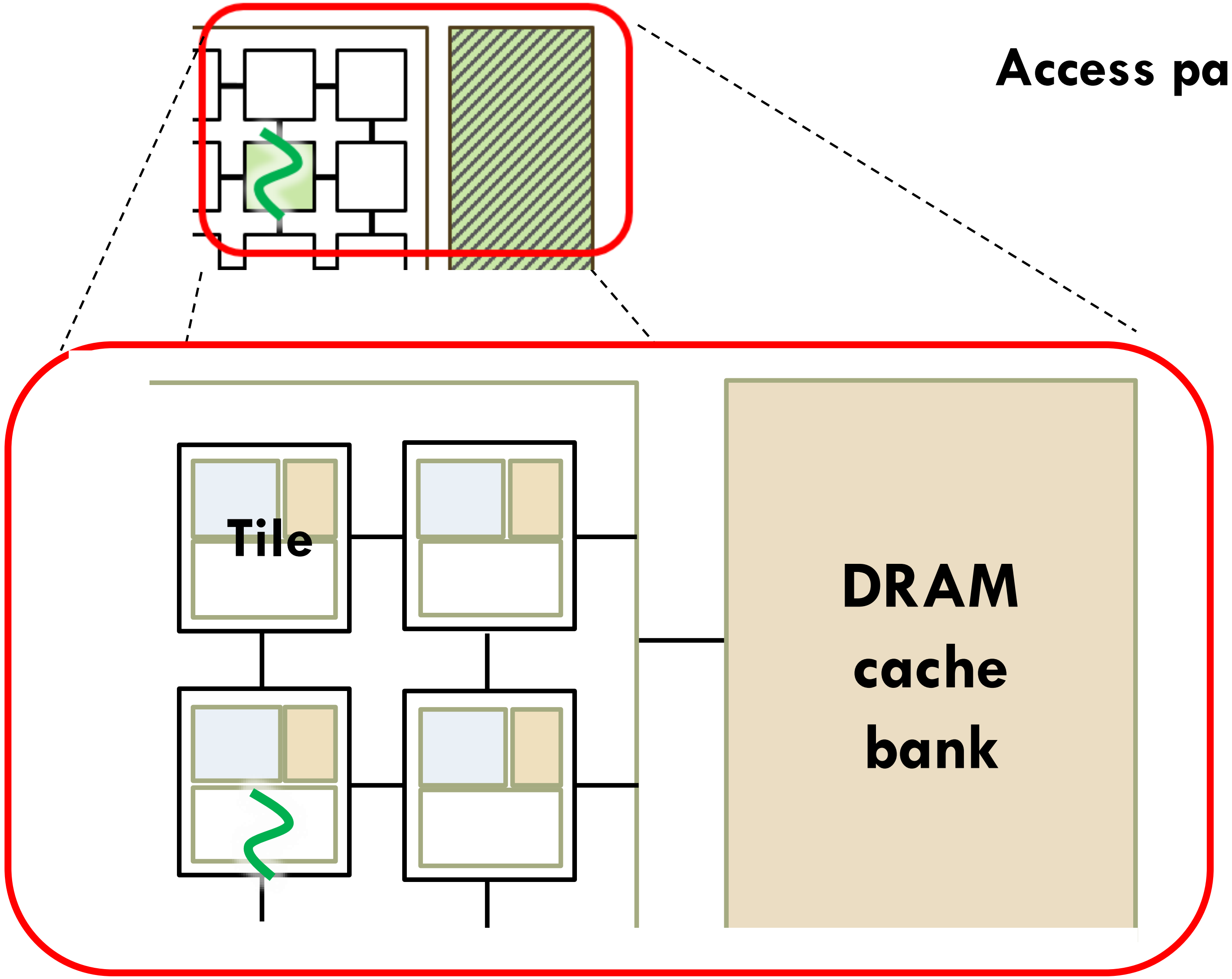
Jenga hardware: supporting virtual hierarchies (VHs)

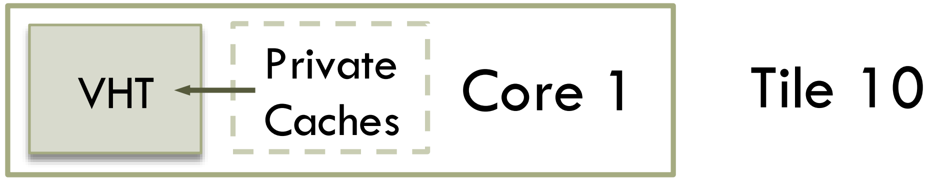
- Cores consult **virtual hierarchy table (VHT)** to find the access path
  - Similar to Jigsaw [PACT'13, HPCA'15], but it supports two levels



Accessing a two-level virtual hierarchy

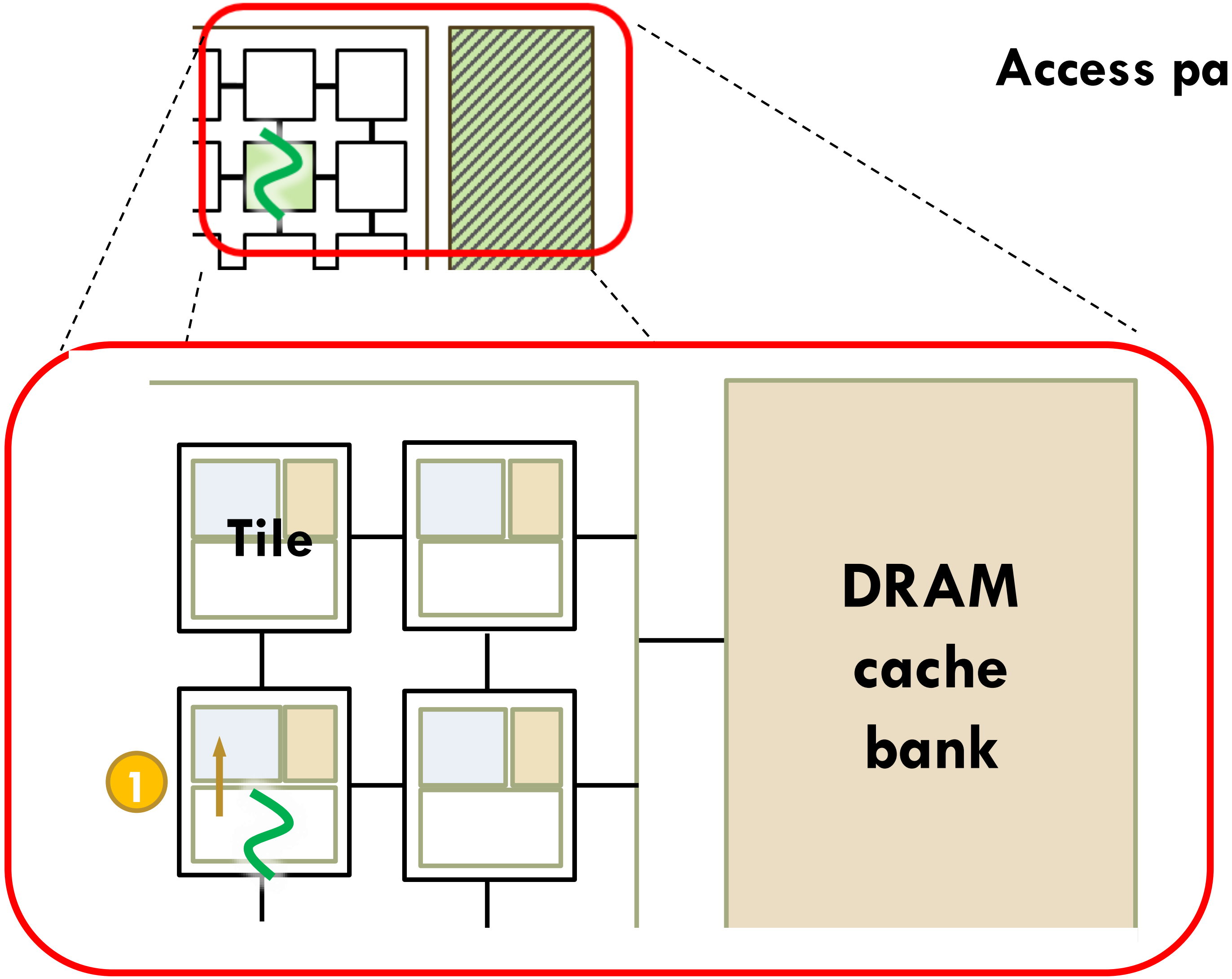
Access path: SRAM bank → DRAM bank → Mem

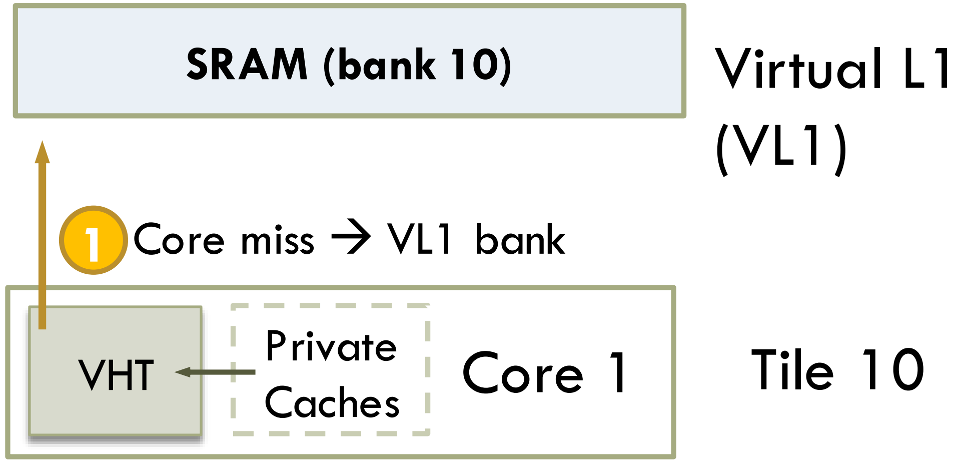




Accessing a two-level virtual hierarchy

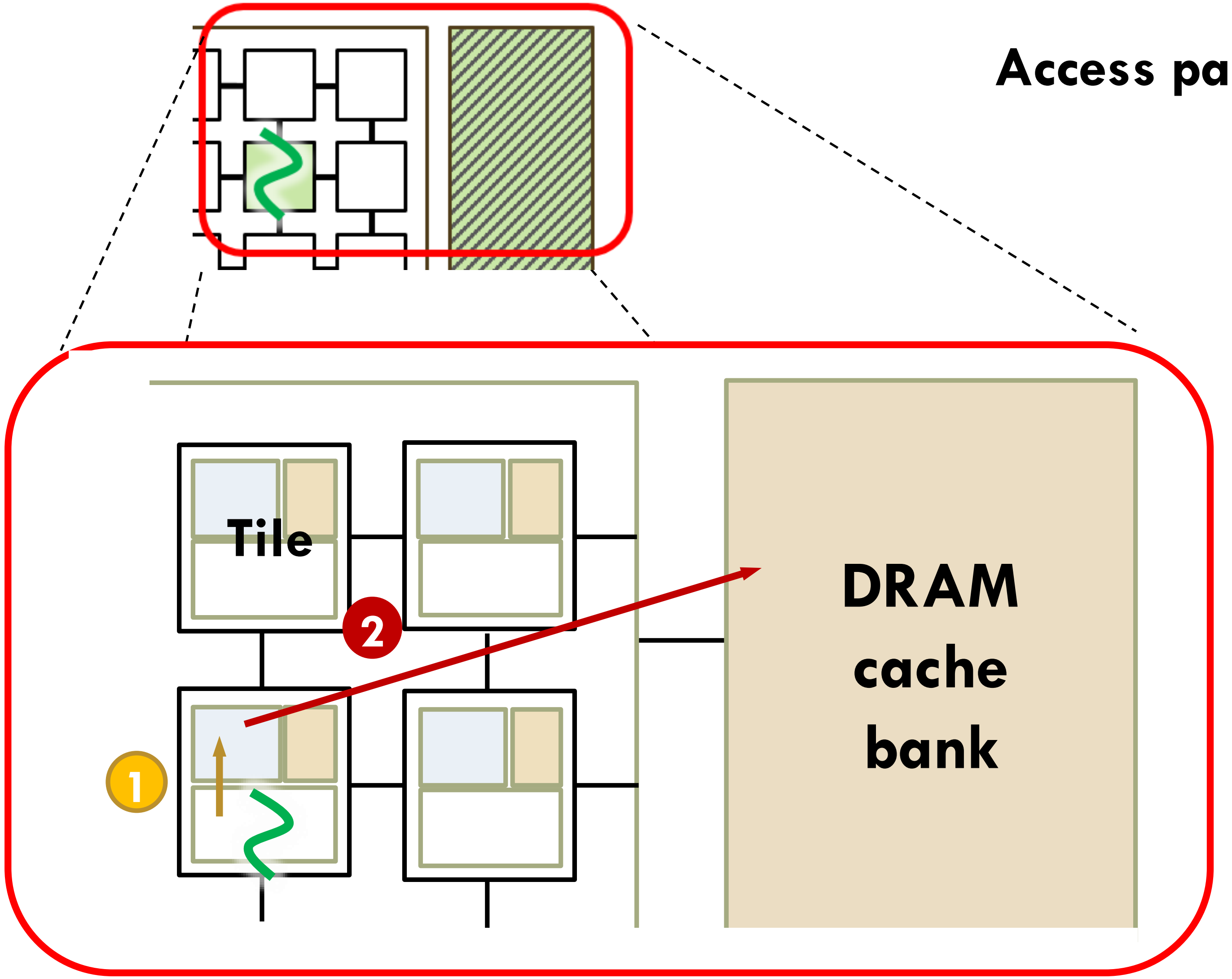
Access path: SRAM bank → DRAM bank → Mem

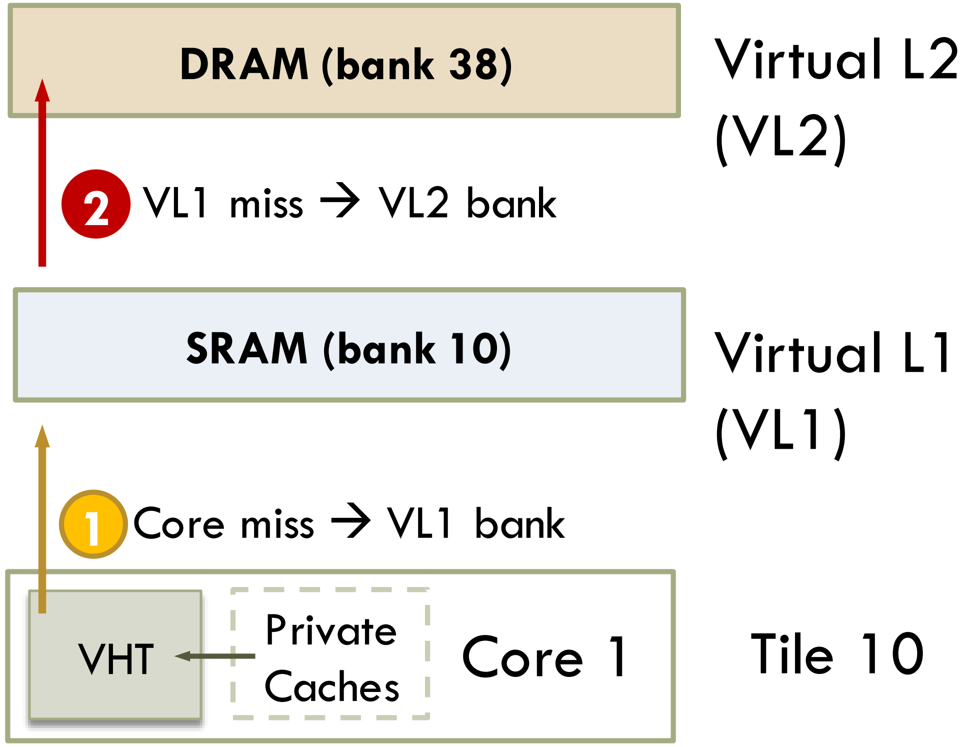




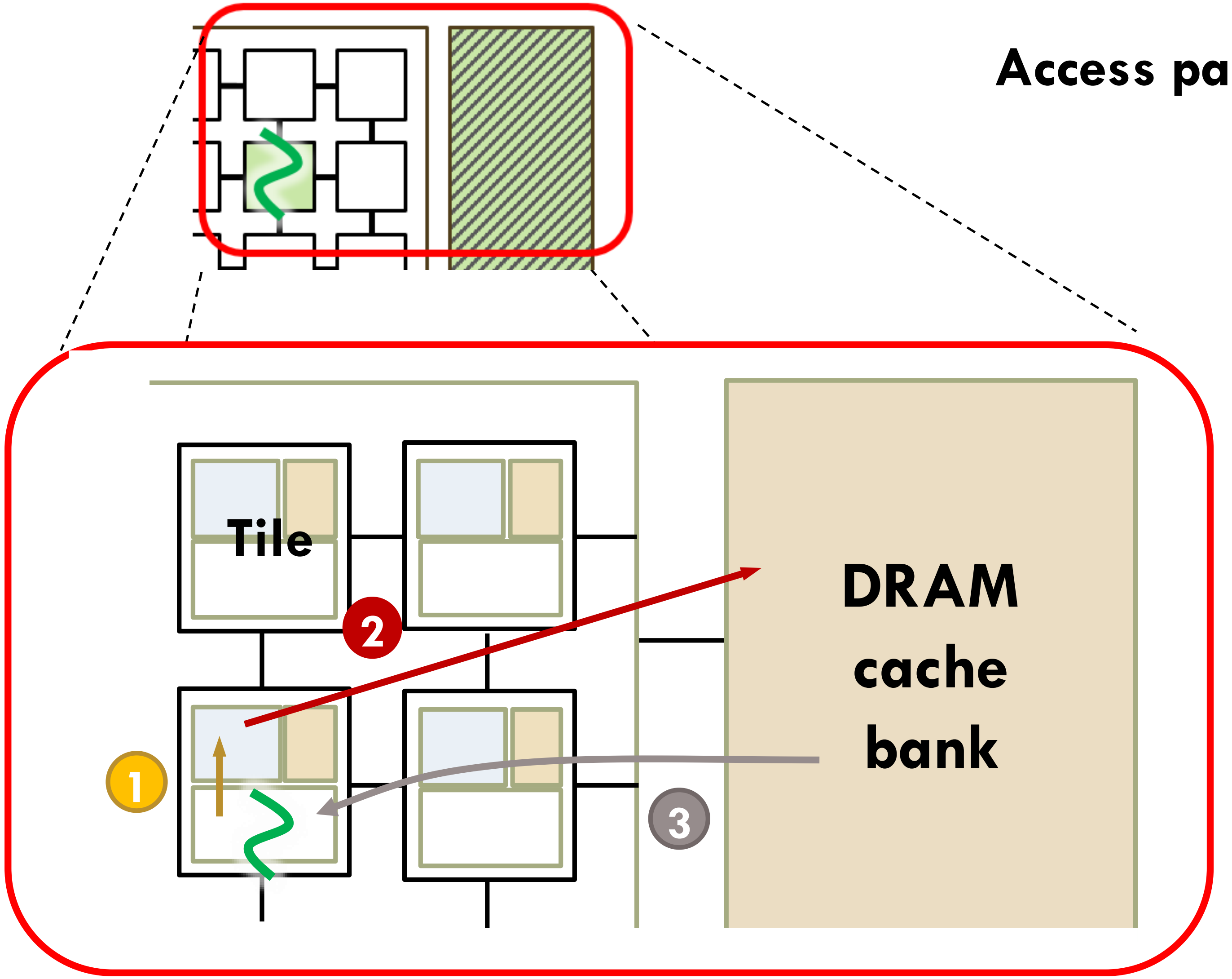
Accessing a two-level virtual hierarchy

Access path: SRAM bank → DRAM bank → Mem

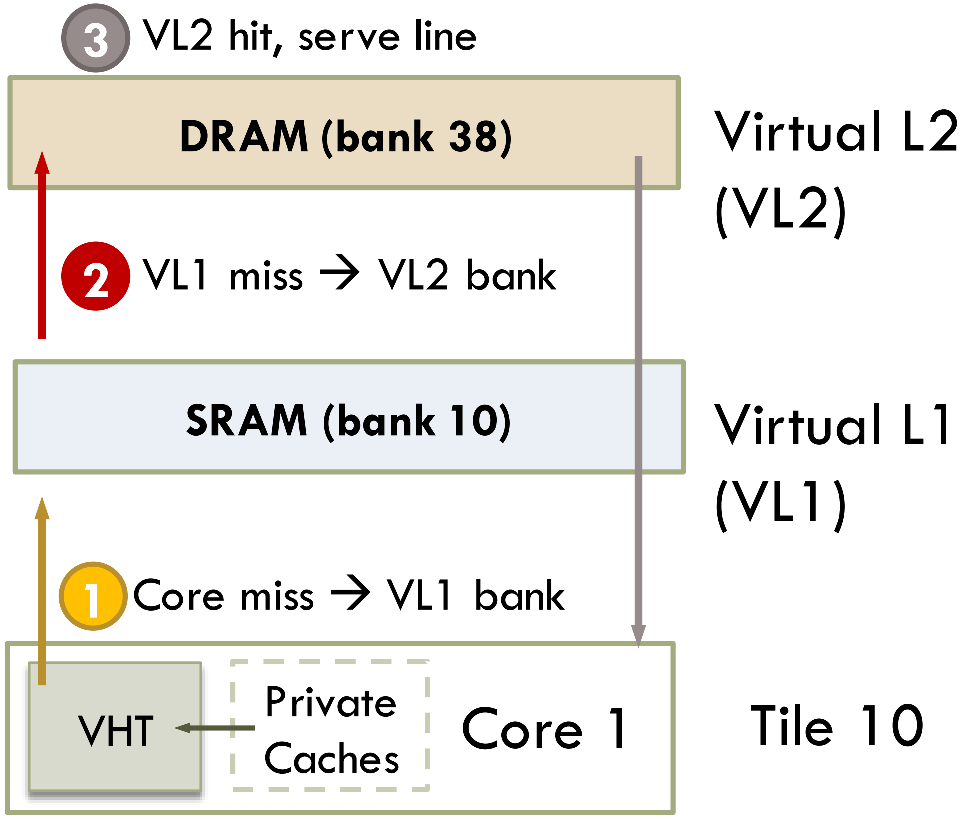




Accessing a two-level virtual hierarchy



Access path: SRAM bank → DRAM bank → Mem



3 VL2 hit, serve line

DRAM (bank 38)

Virtual L2 (VL2)

2 VL1 miss → VL2 bank

SRAM (bank 10)

Virtual L1 (VL1)

1 Core miss → VL1 bank

VHT

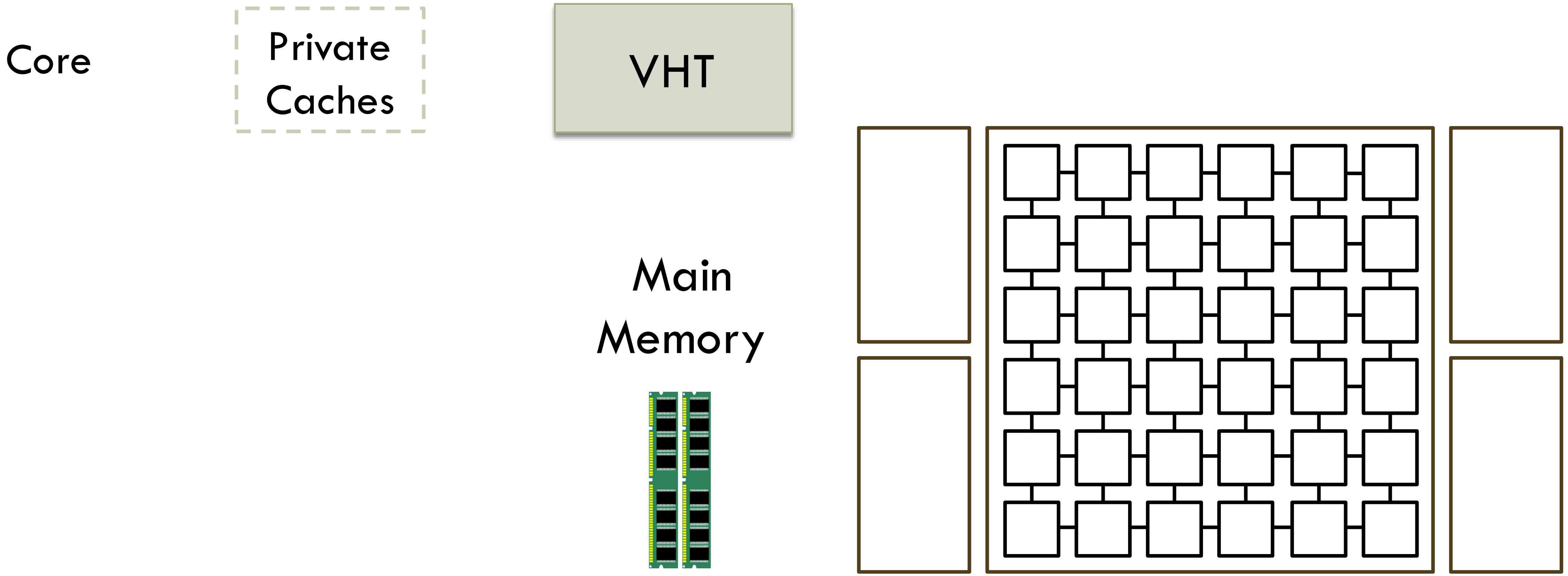
Private Caches

Core 1

Tile 10

Accessing an single-level VH using SRAM + DRAM

- □ With VHT, software can group any combinations of banks to form a VH



Accessing an single-level VH using SRAM + DRAM

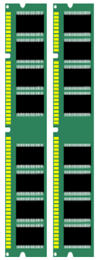
- With VHT, software can group any combinations of banks to form a VH

Core

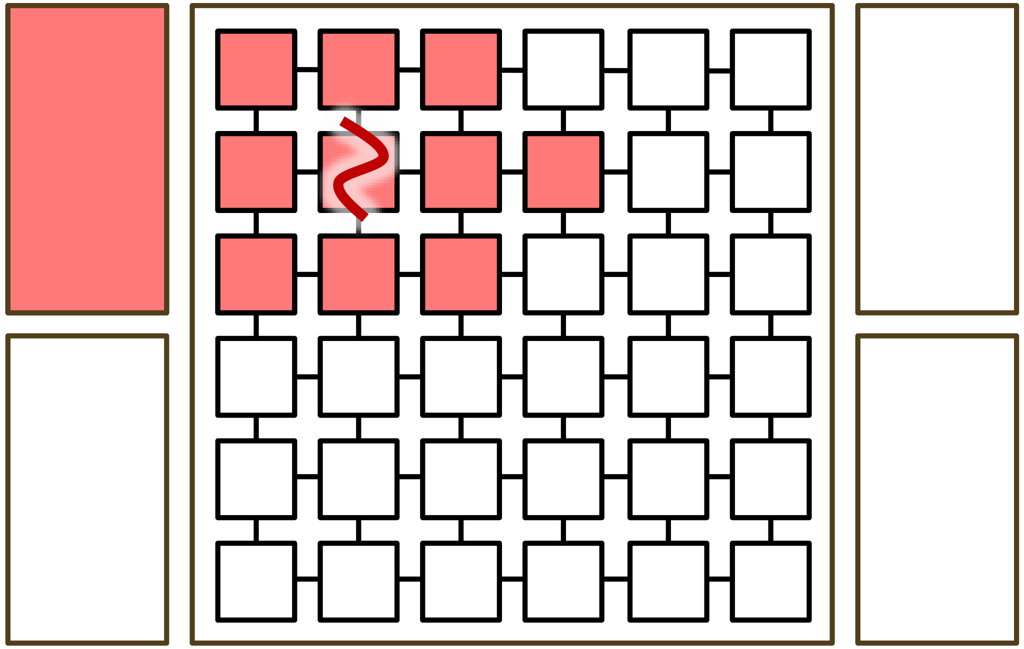
Private Caches

VHT

Main Memory

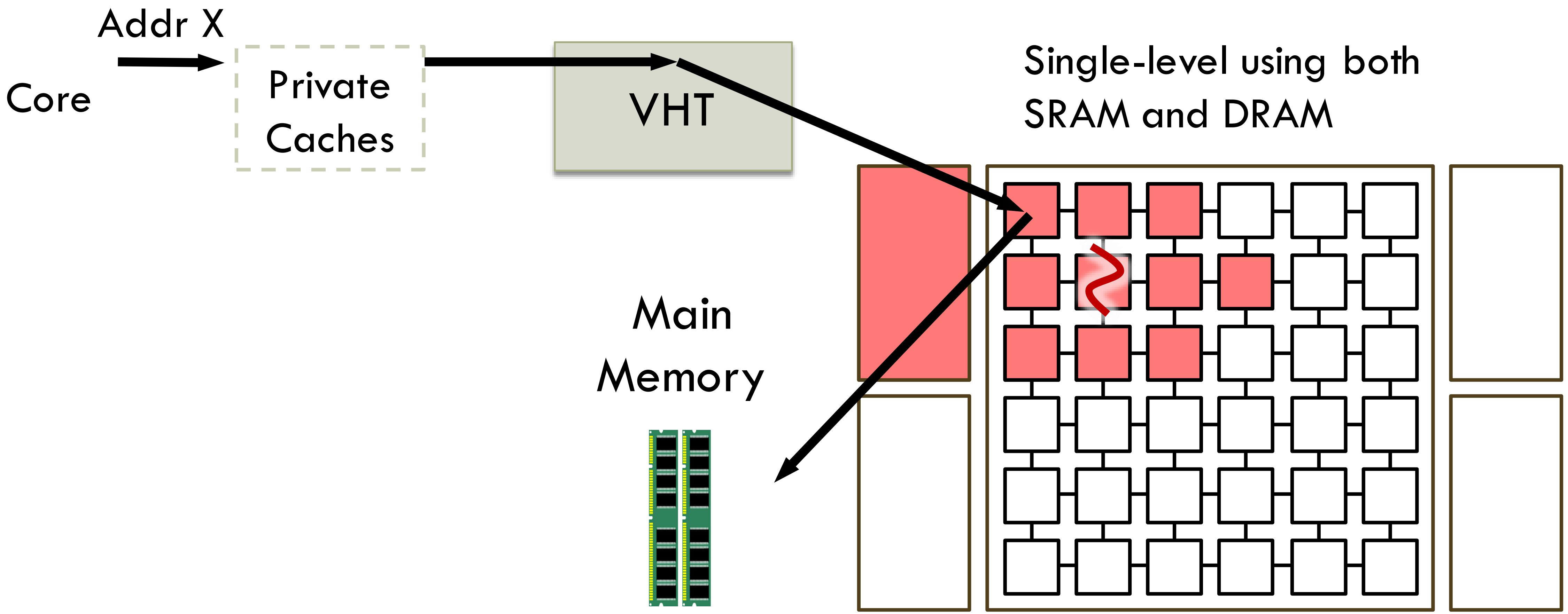


Single-level using both SRAM and DRAM



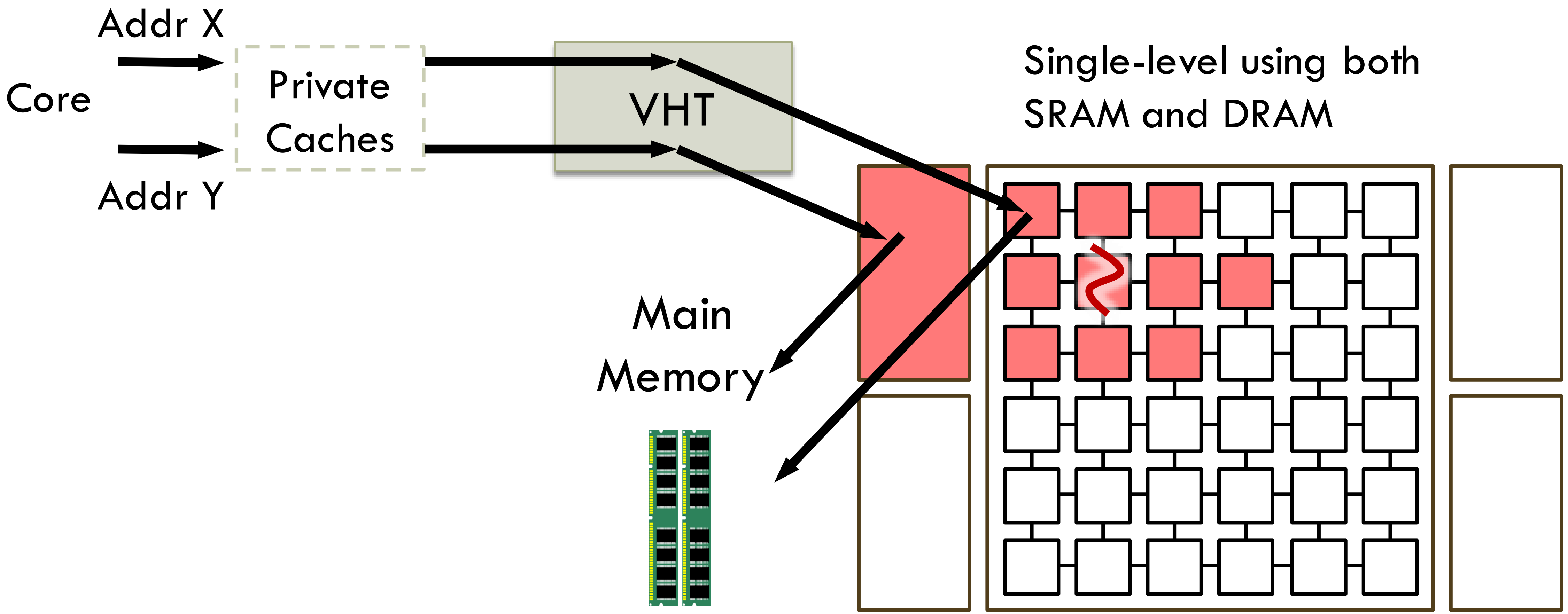
Accessing an single-level VH using SRAM + DRAM

- With VHT, software can group any combinations of banks to form a VH



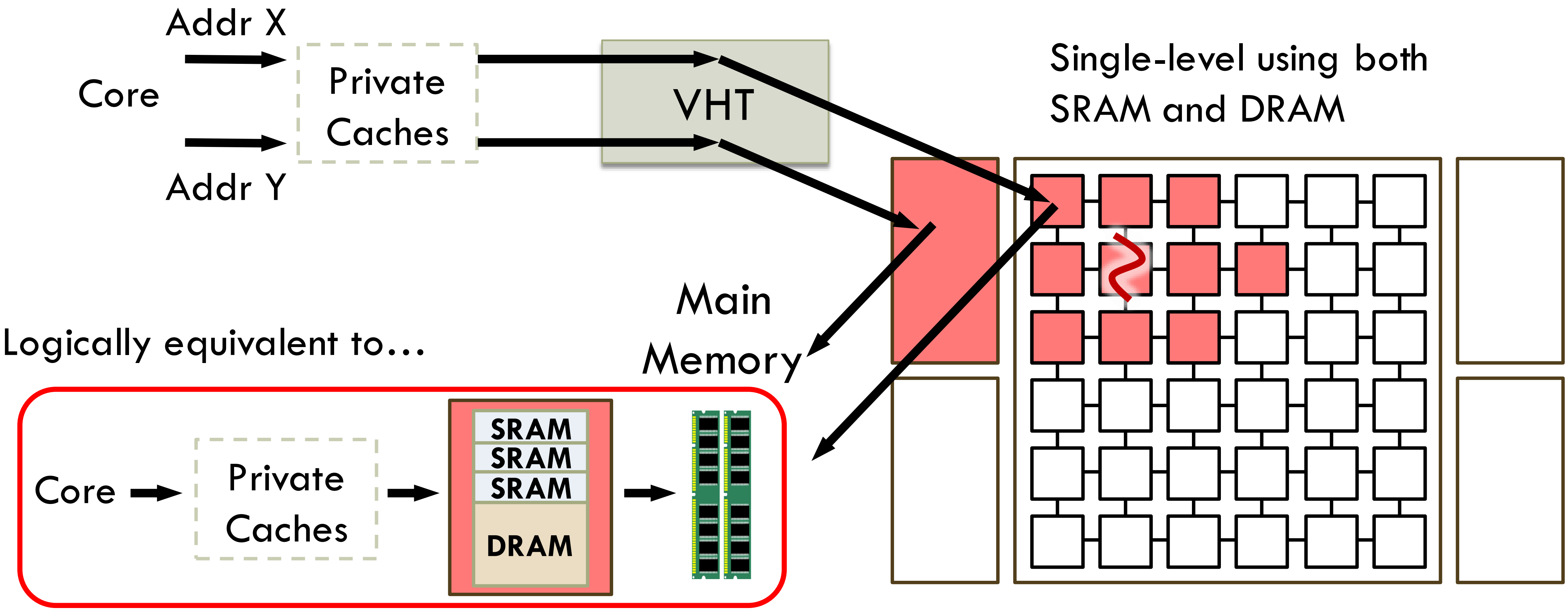
Accessing an single-level VH using SRAM + DRAM

- With VHT, software can group any combinations of banks to form a VH



Accessing an single-level VH using SRAM + DRAM

- With VHT, software can group any combinations of banks to form a VH

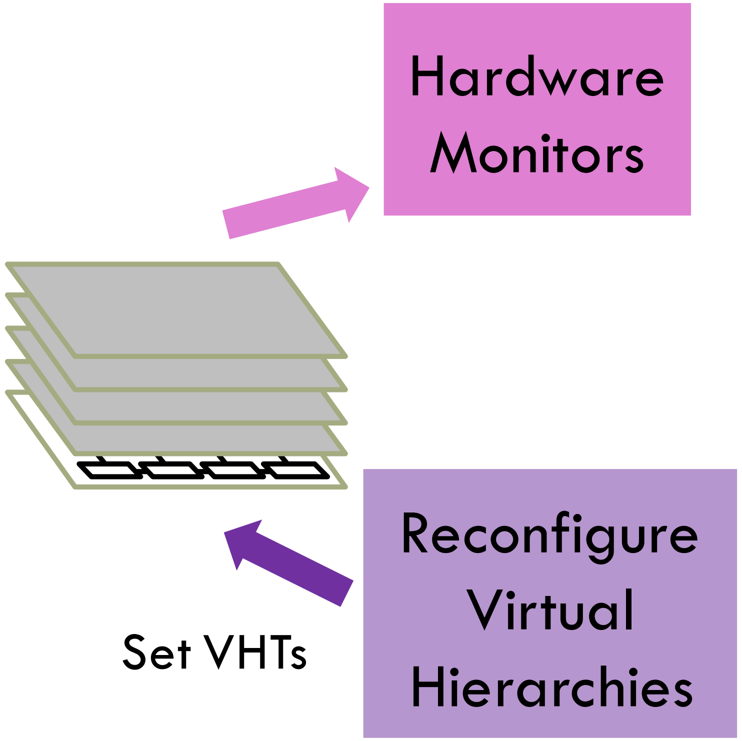


Jenga software: finding near-optimal hierarchies

□ Periodically, Jenga reconfigures VHs to minimize data movement

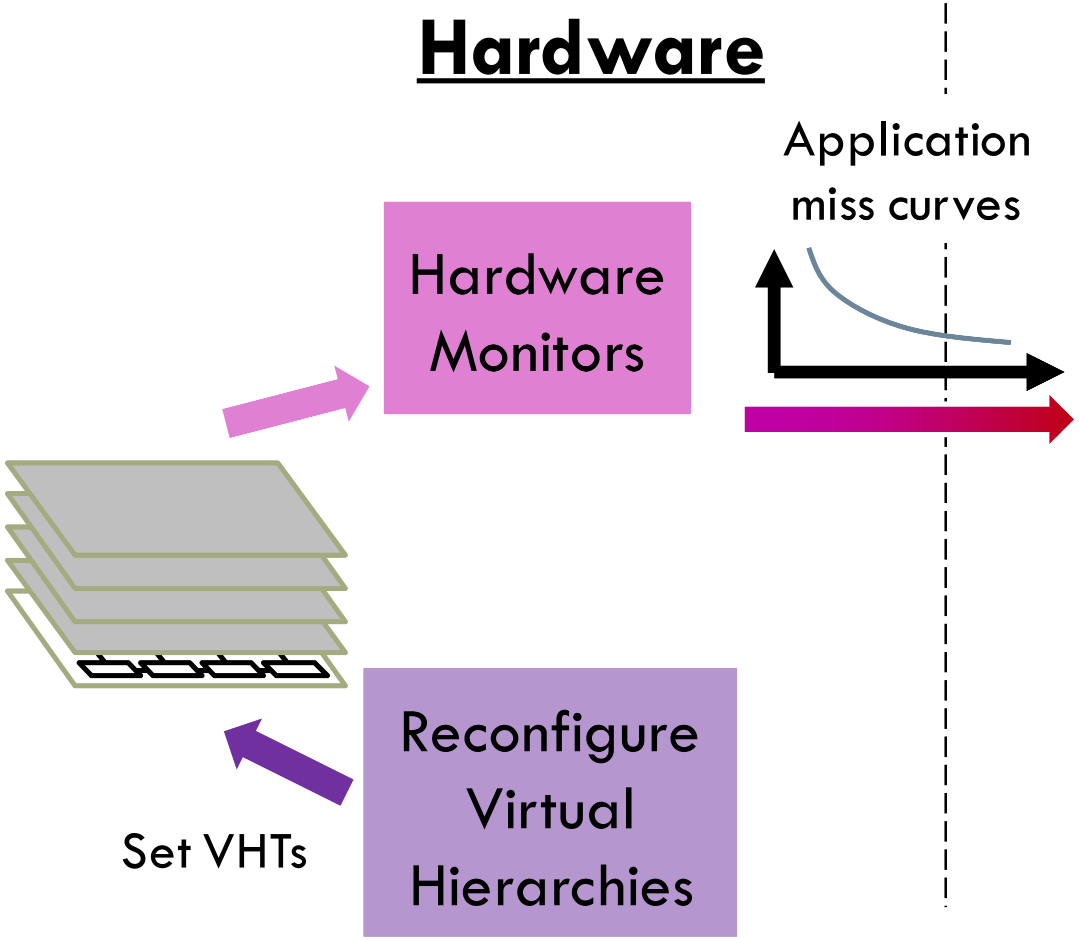
Hardware

Software



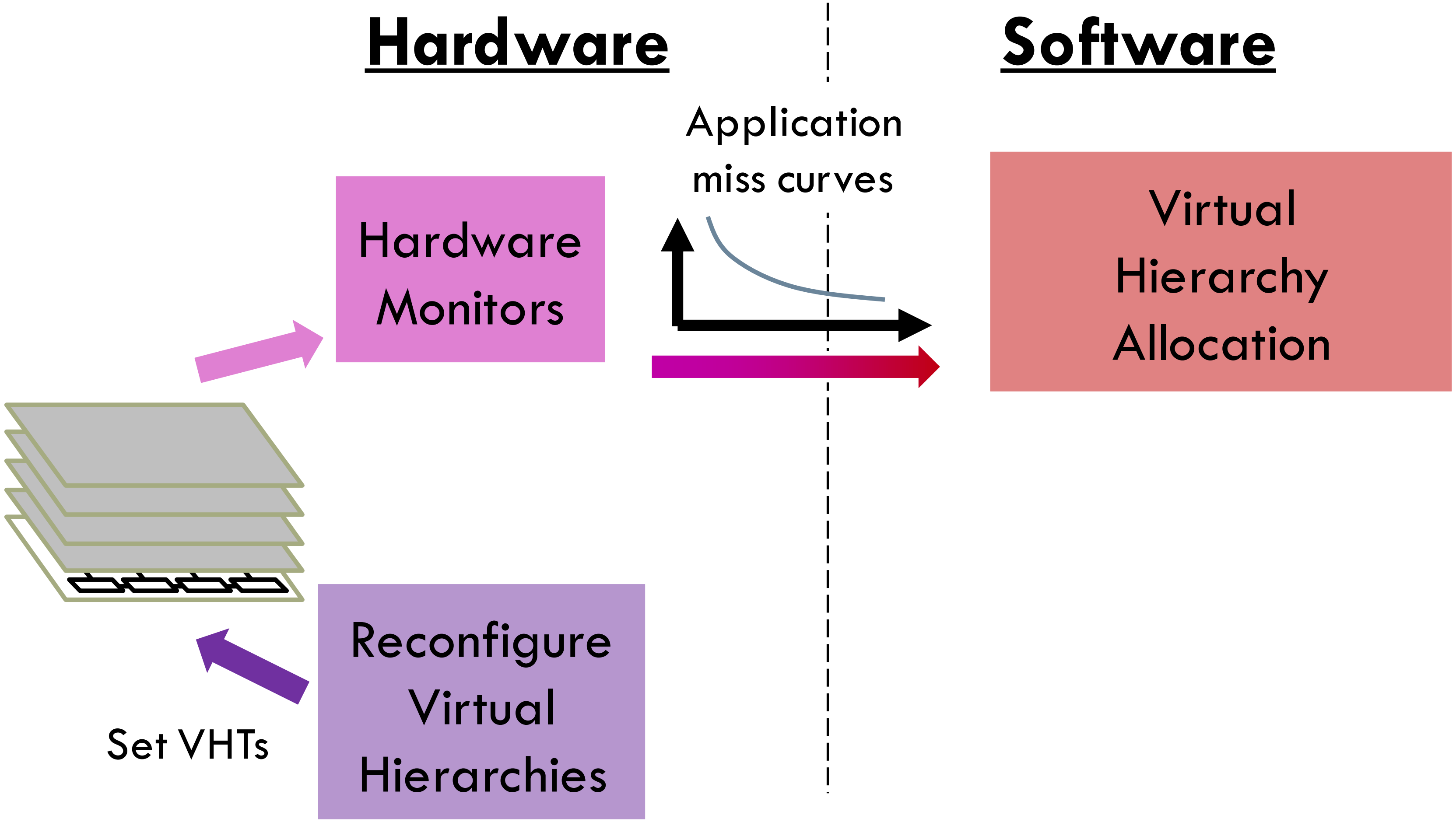
Jenga software: finding near-optimal hierarchies

- Periodically, Jenga reconfigures VHs to minimize data movement



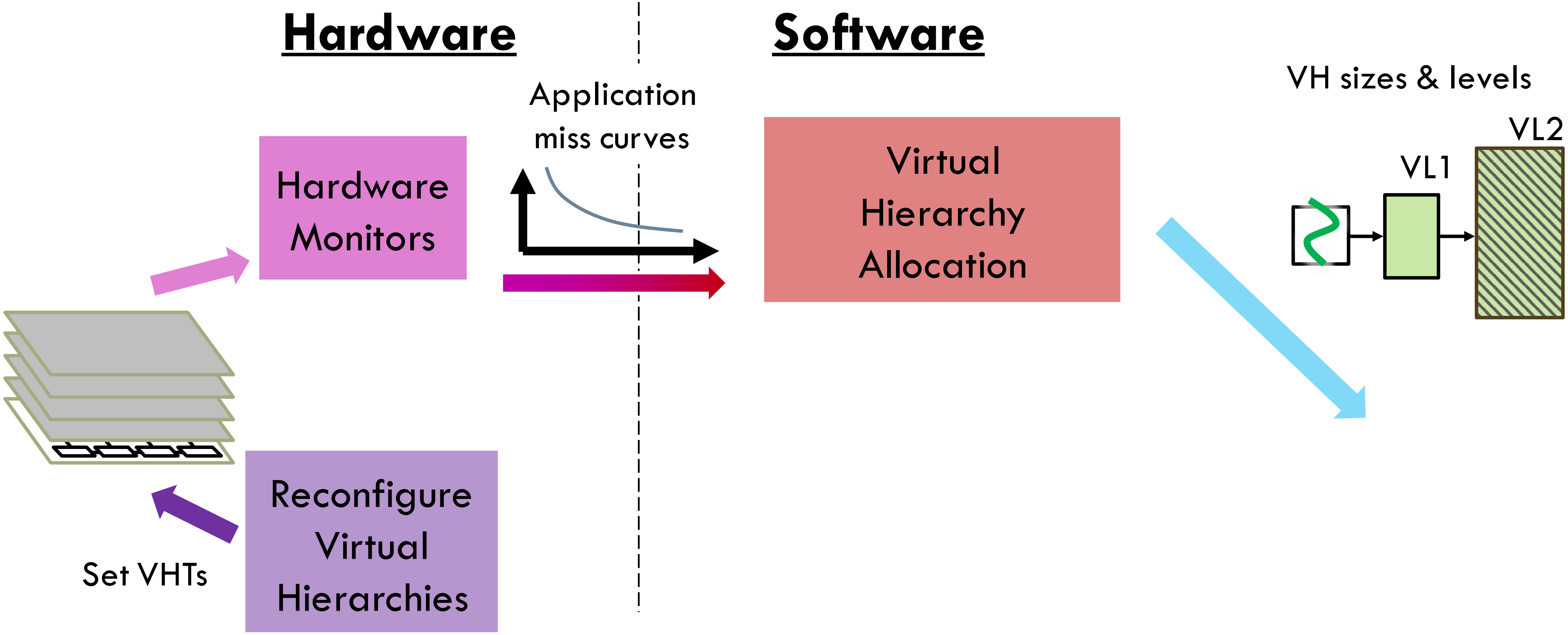
Jenga software: finding near-optimal hierarchies

- Periodically, Jenga reconfigures VHs to minimize data movement



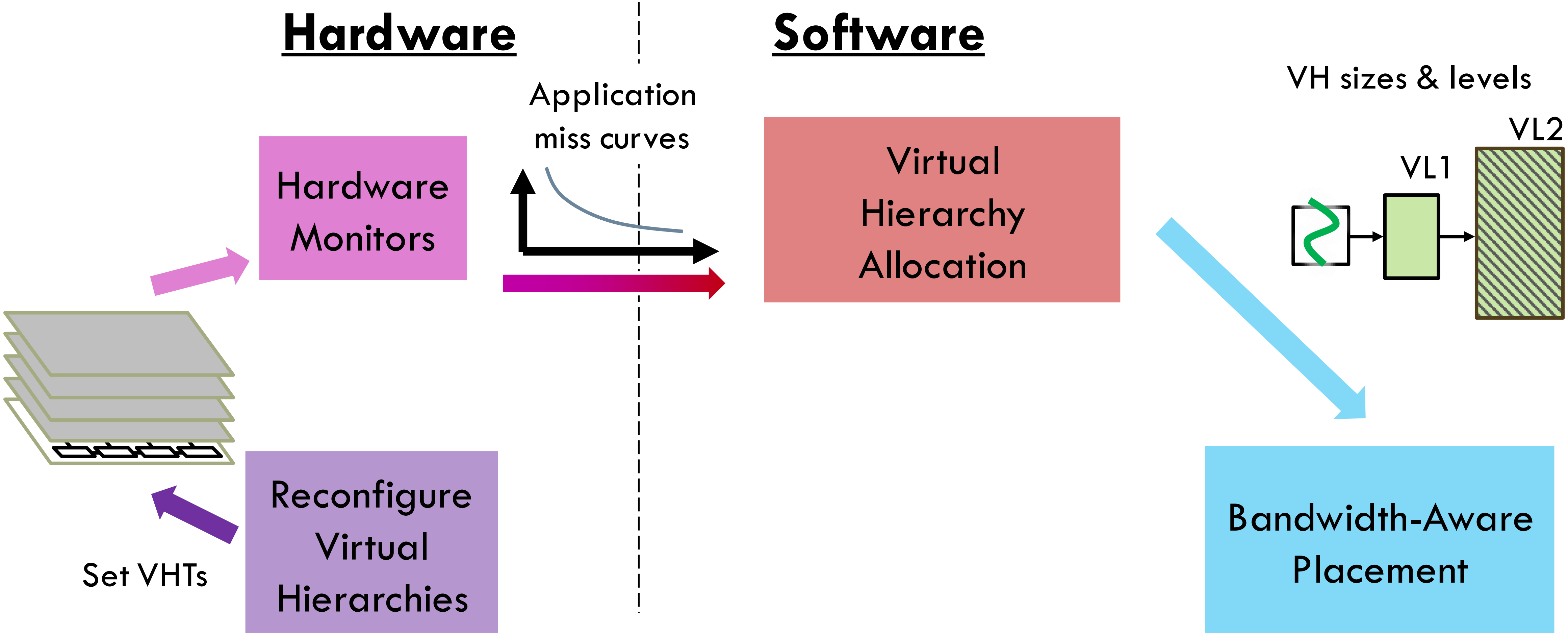
Jenga software: finding near-optimal hierarchies

□ Periodically, Jenga reconfigures VHs to minimize data movement



Jenga software: finding near-optimal hierarchies

- Periodically, Jenga reconfigures VHs to minimize data movement



Hardware

Software

Hardware Monitors

Application miss curves

Virtual Hierarchy Allocation

VH sizes & levels

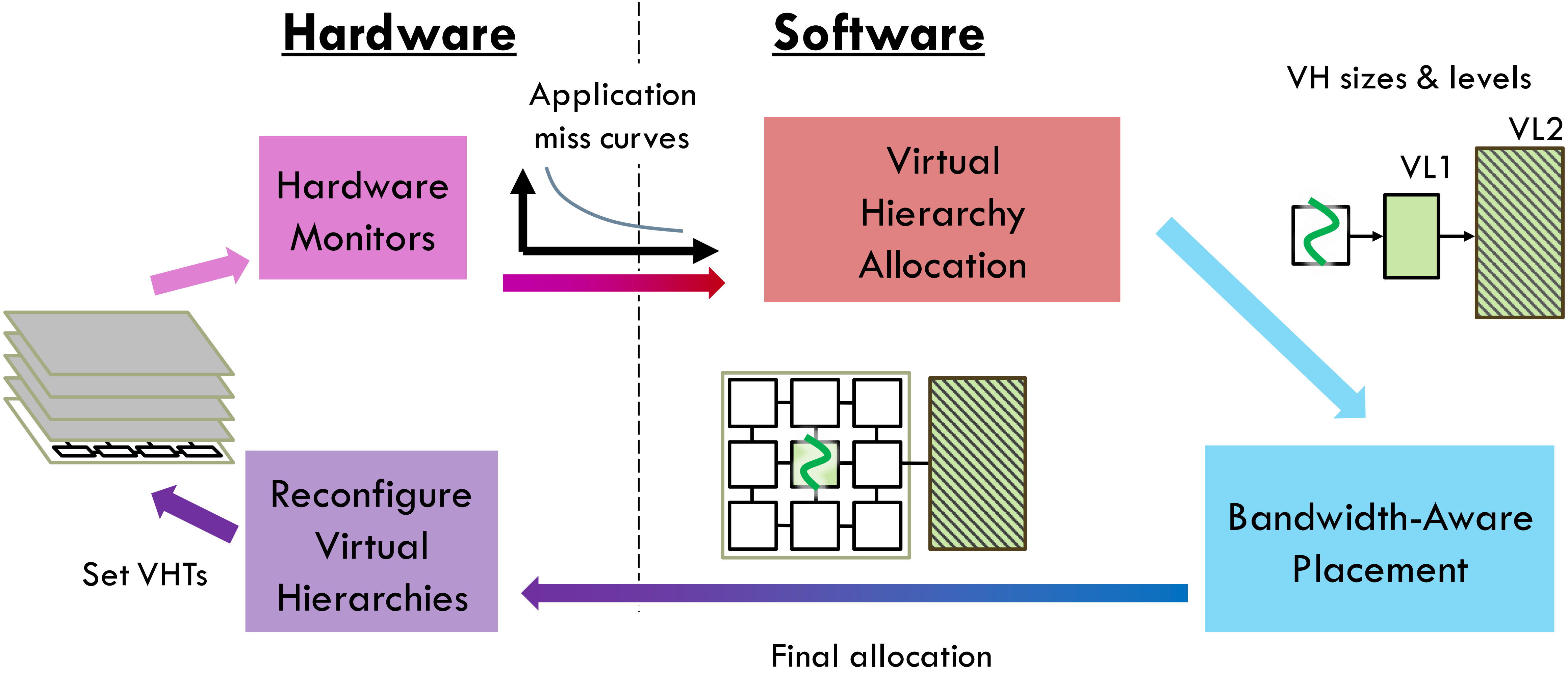
Bandwidth-Aware Placement

Reconfigure Virtual Hierarchies

Set VHTs

Jenga software: finding near-optimal hierarchies

- Periodically, Jenga reconfigures VHs to minimize data movement

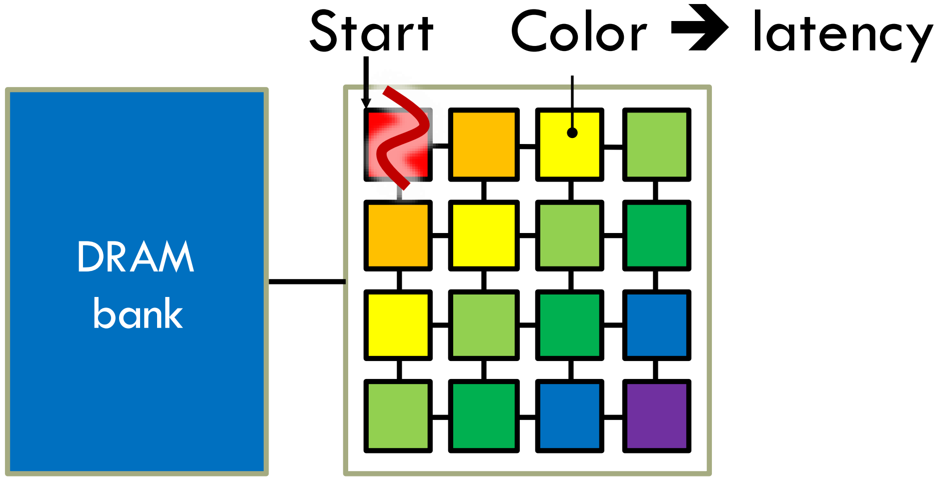


Modeling performance of heterogeneous caches

- Treat SRAM and DRAM as different "flavors" of banks with different latencies

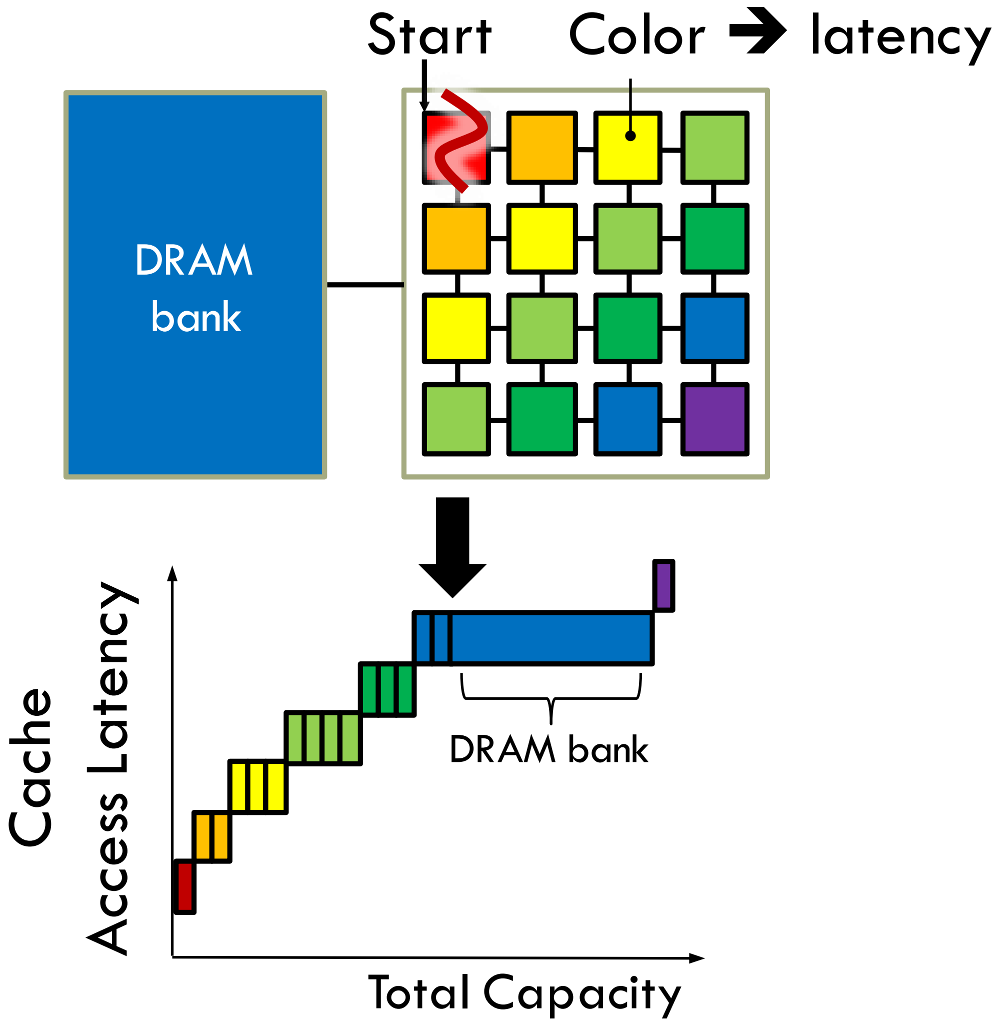
Modeling performance of heterogeneous caches

- Treat SRAM and DRAM as different "flavors" of banks with different latencies



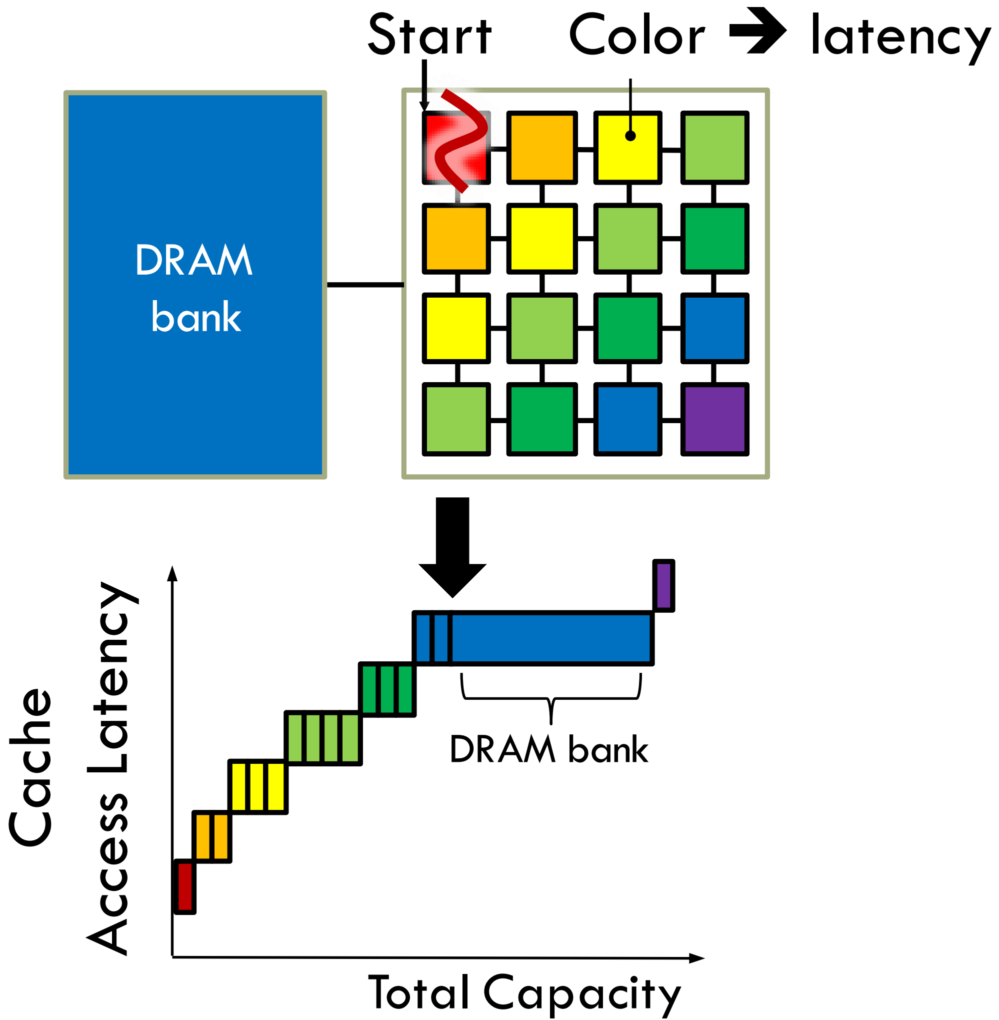
Modeling performance of heterogeneous caches

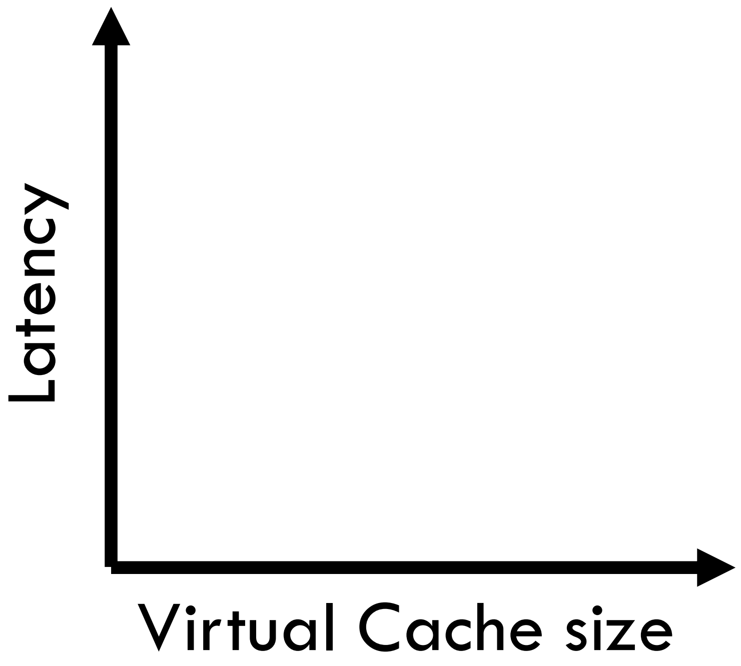
□ Treat SRAM and DRAM as different “flavors” of banks with different latencies



Modeling performance of heterogeneous caches

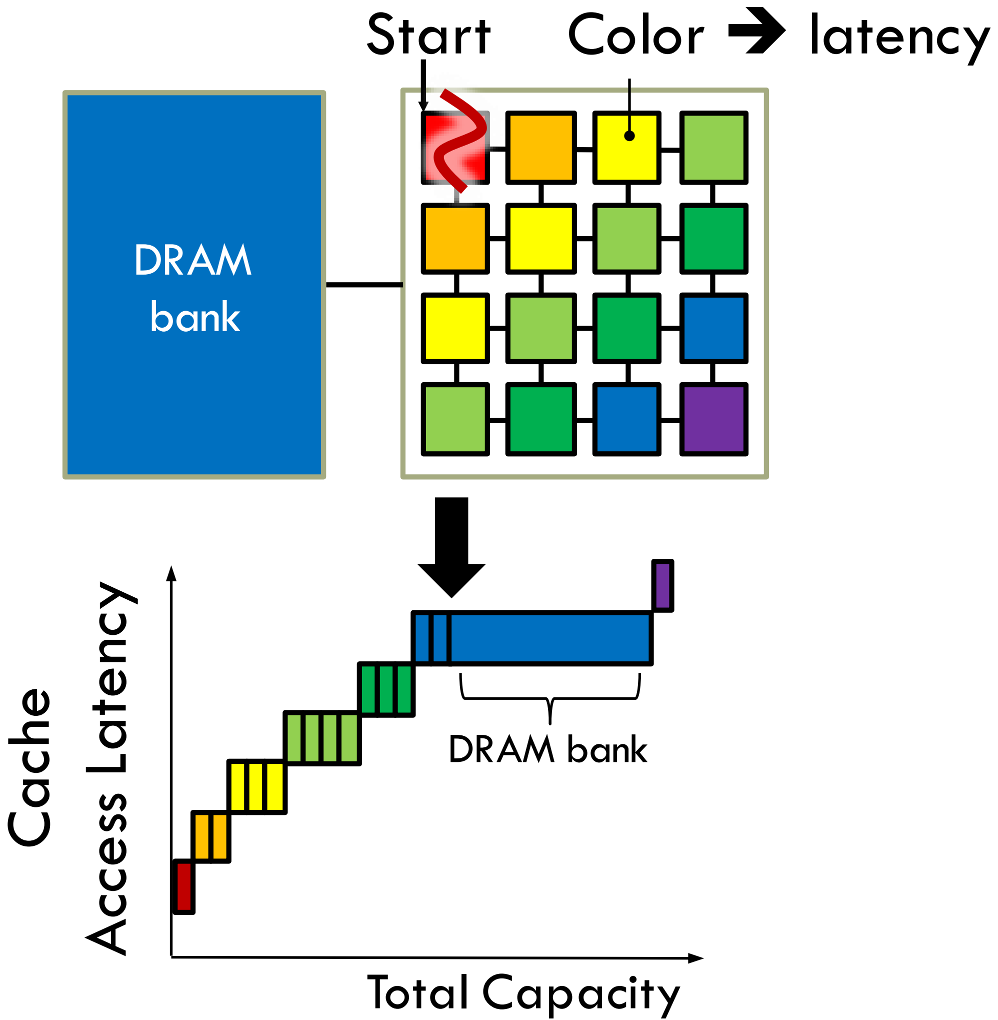
- Treat SRAM and DRAM as different "flavors" of banks with different latencies

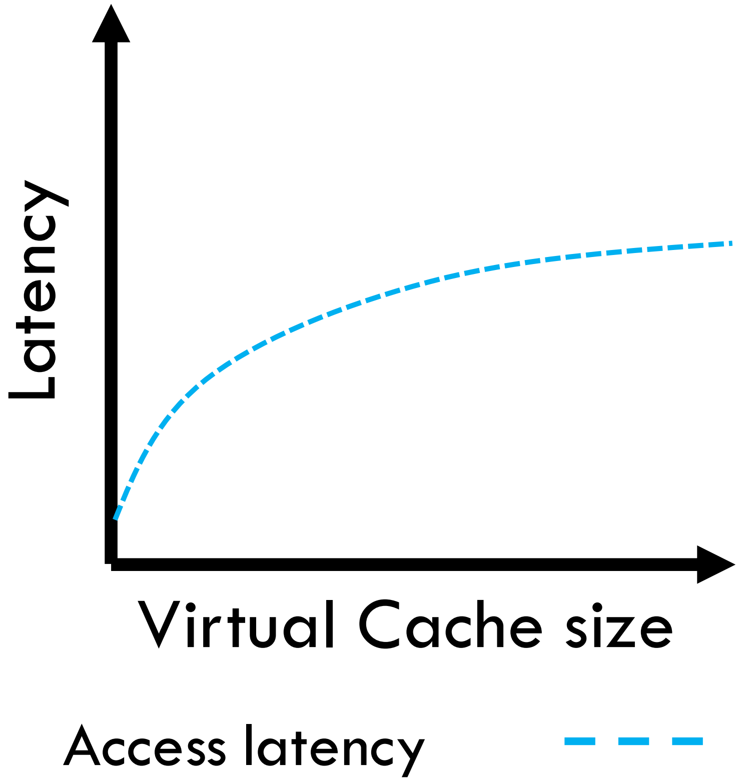




Modeling performance of heterogeneous caches

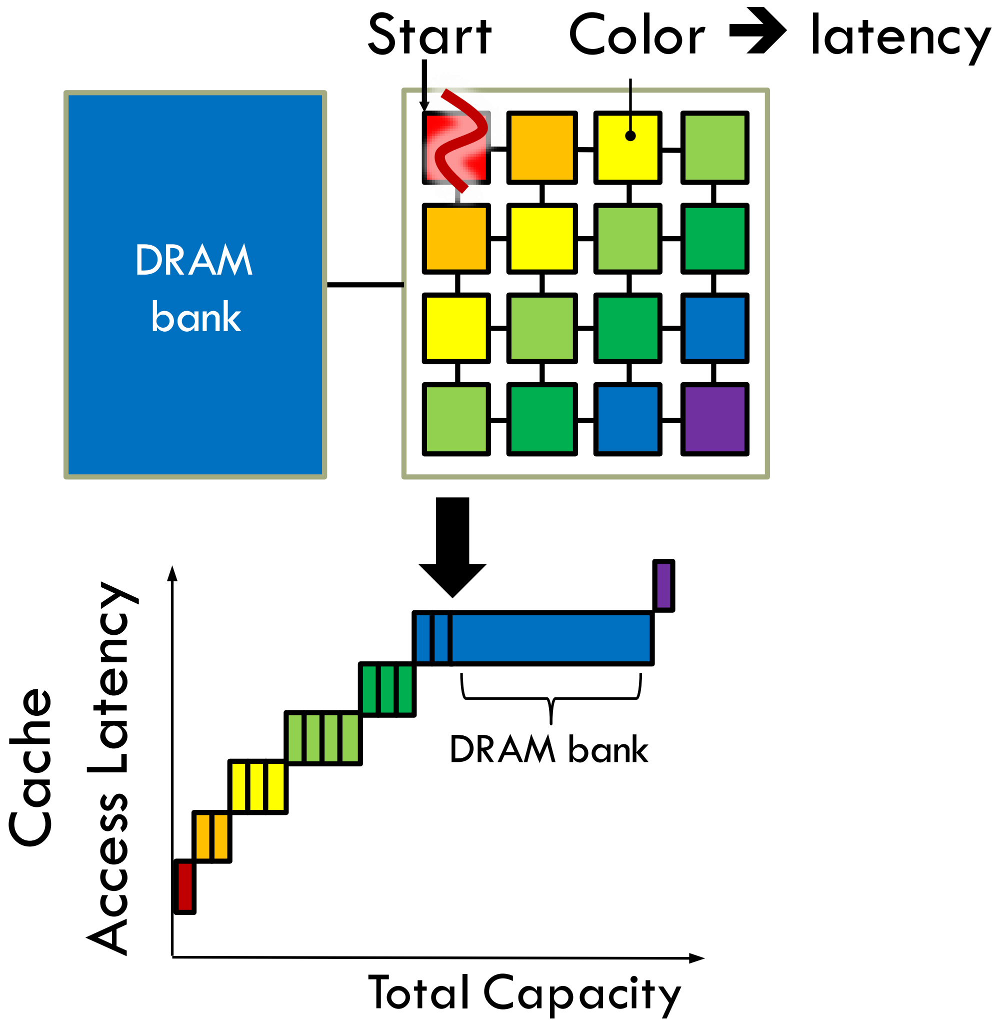
- Treat SRAM and DRAM as different "flavors" of banks with different latencies

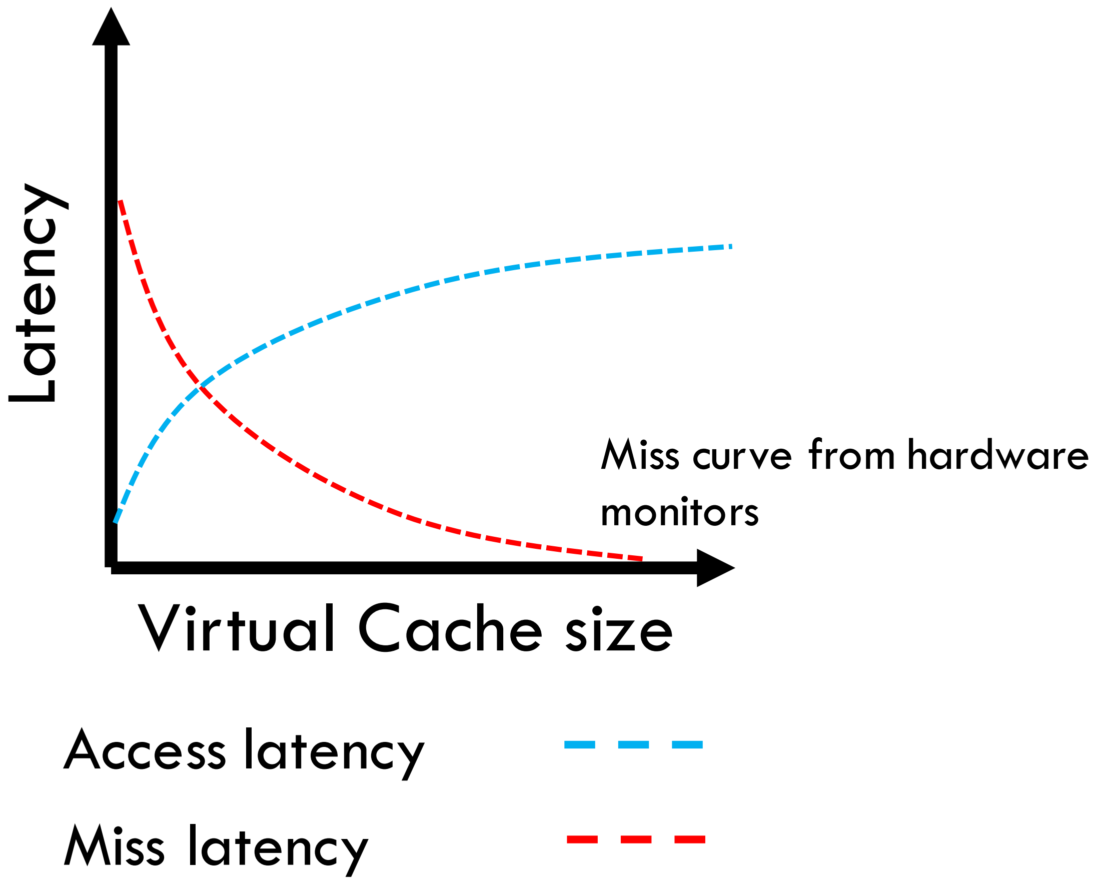




Modeling performance of heterogeneous caches

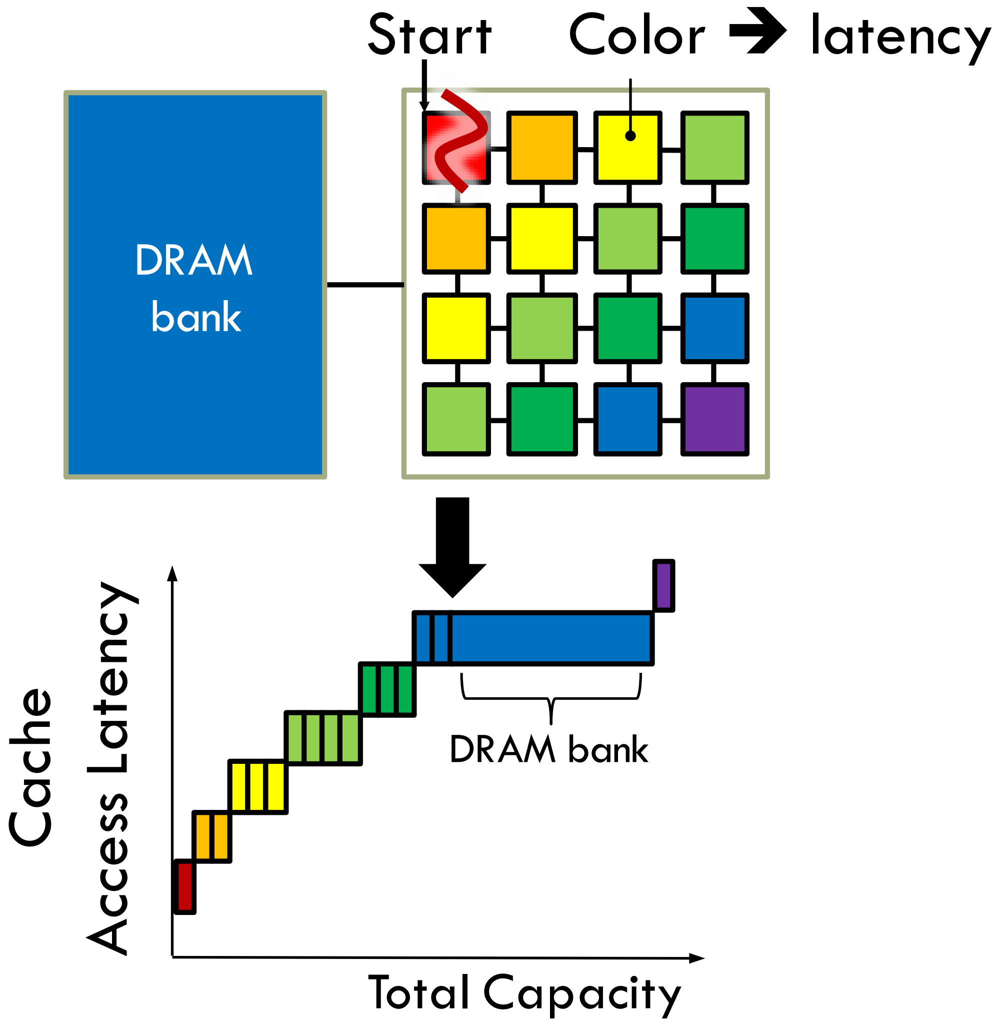
- Treat SRAM and DRAM as different “flavors” of banks with different latencies

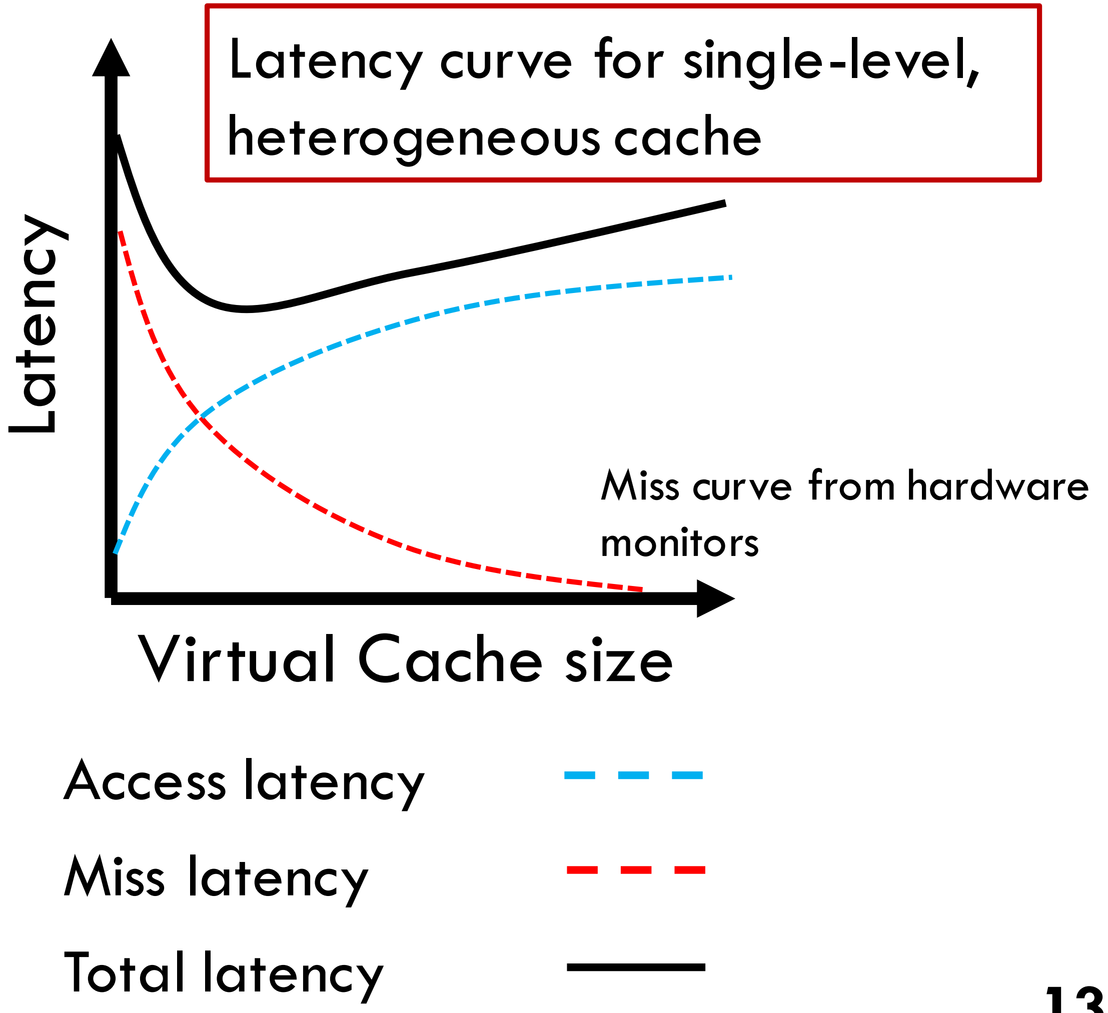




Modeling performance of heterogeneous caches

- Treat SRAM and DRAM as different "flavors" of banks with different latencies





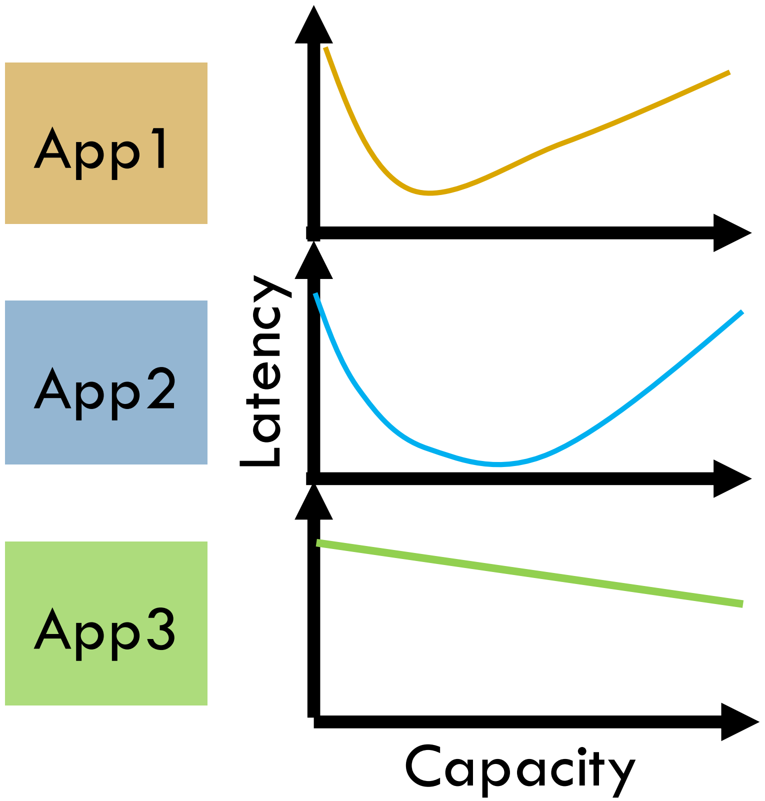
Optimizing hierarchies by minimizing system latency

Optimizing hierarchies by minimizing system latency

- Our prior work has proposed algorithms to take latency curves, allocate capacity and place them on chip to minimize system latency
  - But only builds *single-level* VHs

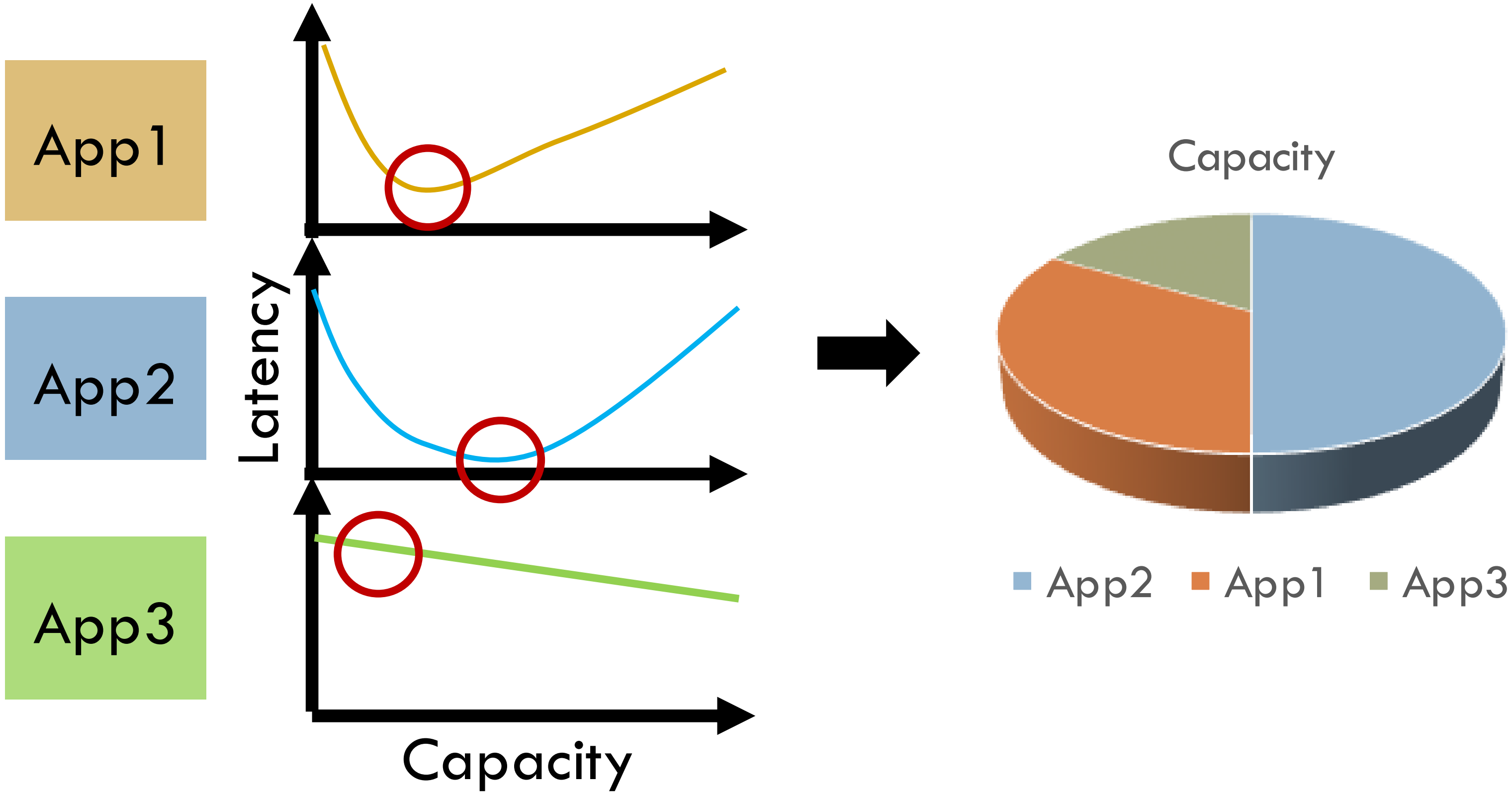
Optimizing hierarchies by minimizing system latency

- Our prior work has proposed algorithms to take latency curves, allocate capacity and place them on chip to minimize system latency
  - But only builds *single-level* VHs



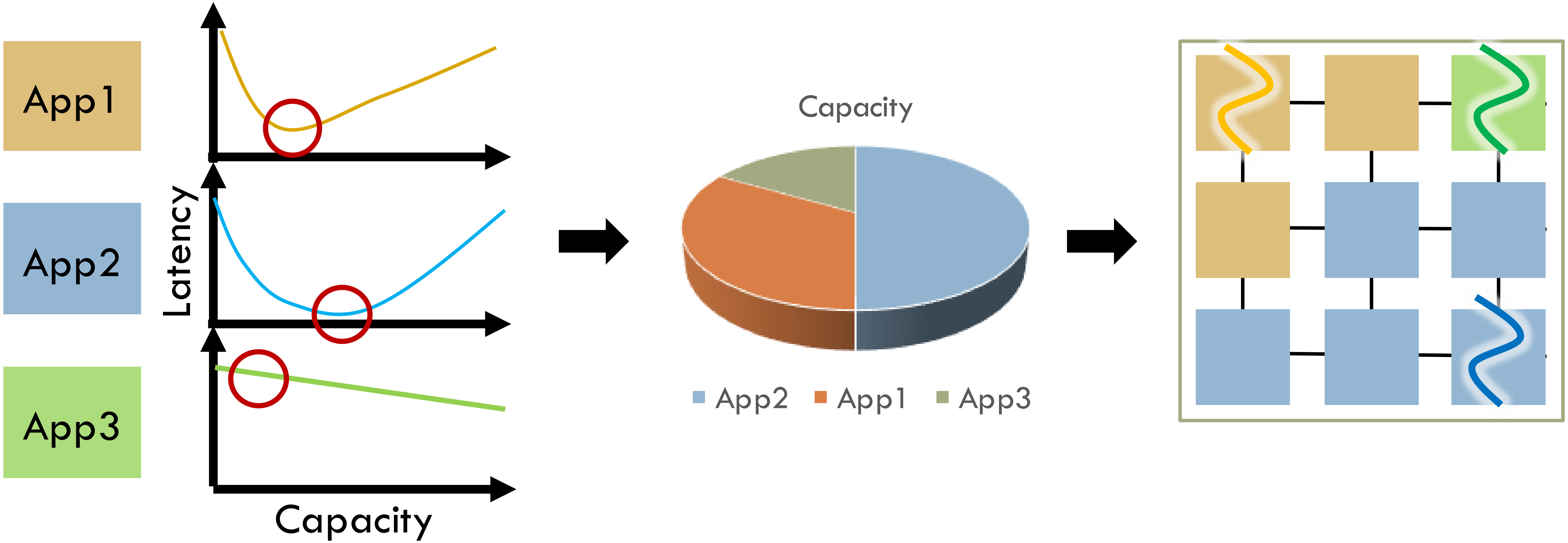
Optimizing hierarchies by minimizing system latency

- Our prior work has proposed algorithms to take latency curves, allocate capacity and place them on chip to minimize system latency
  - But only builds *single-level* VHs



Optimizing hierarchies by minimizing system latency

- Our prior work has proposed algorithms to take latency curves, allocate capacity and place them on chip to minimize system latency
  - But only builds *single-level* VHs



Multi-level hierarchies are much more complex

Multi-level hierarchies are much more complex

- Many intertwined factors
  - Best VL1 size depends on VL2 size
  - Best VL2 size depends on VL1 size
  - Should we have VL2? (Depends on total size)

Multi-level hierarchies are much more complex

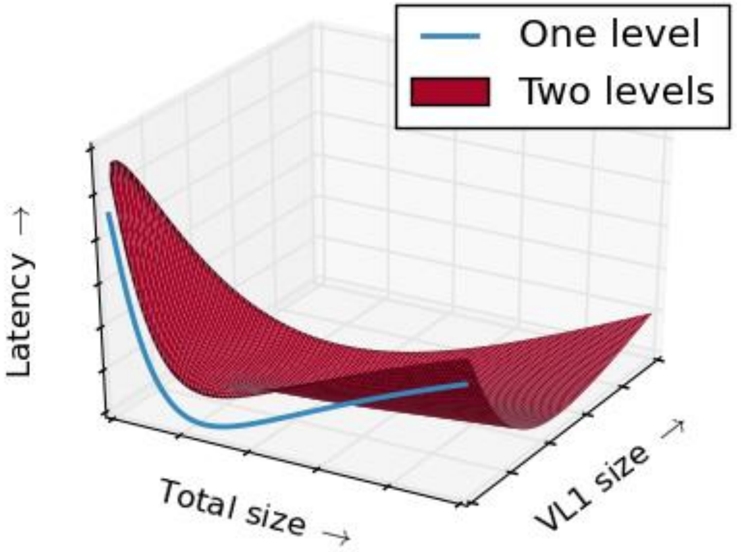
- Many intertwined factors
  - Best VL1 size depends on VL2 size
  - Best VL2 size depends on VL1 size
  - Should we have VL2? (Depends on total size)

- **Jenga encodes these tradeoffs in a single curve**
  - Can reuse prior allocation algorithms

How to get a latency curve for a multi-level VH

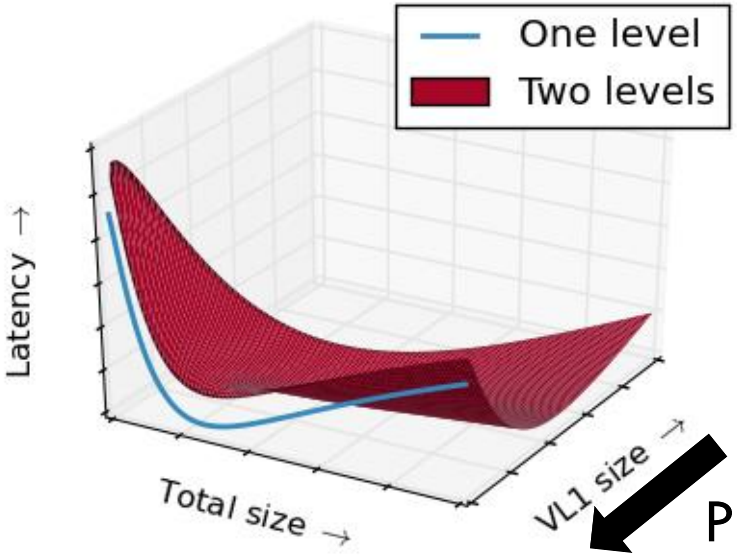
How to get a latency curve for a multi-level VH

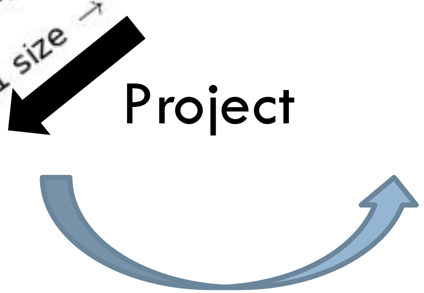
Two-level hierarchies form a latency *surface!*



How to get a latency curve for a multi-level VH

Two-level hierarchies form a latency *surface!*

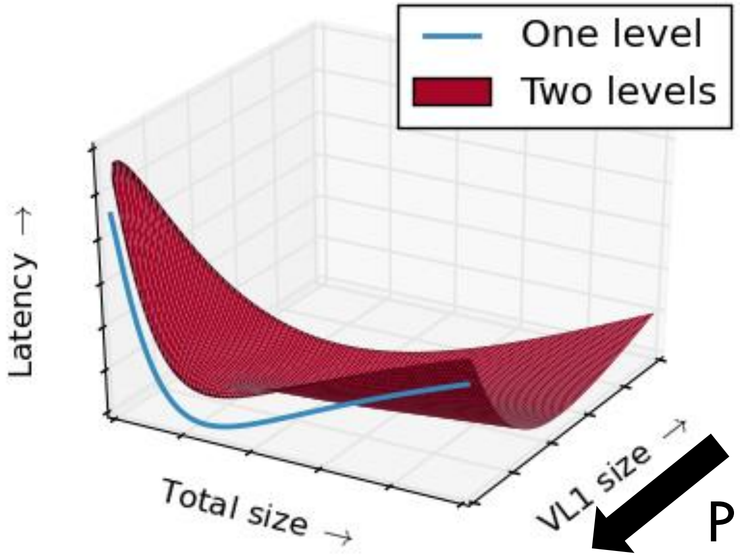




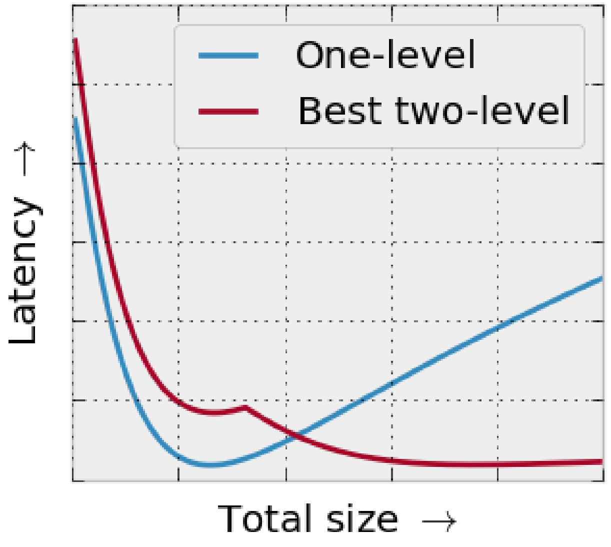
Best 1- and 2-level hierarchy at every size

How to get a latency curve for a multi-level VH

Two-level hierarchies form a latency *surface!*



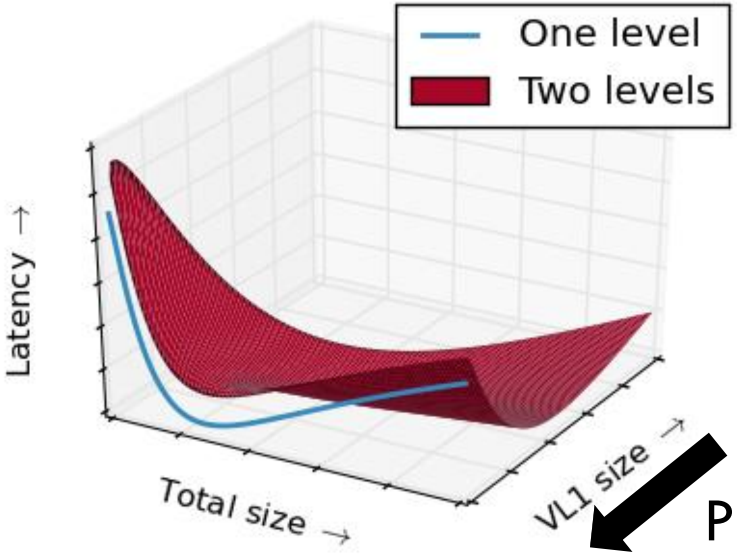
Project



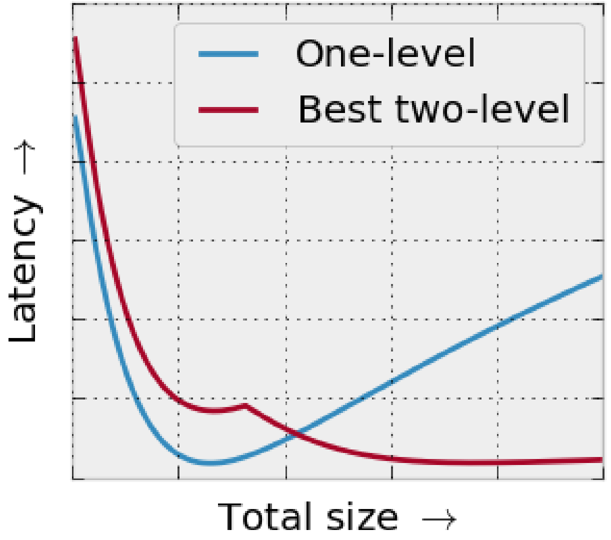
Best 1- and 2-level hierarchy at every size

How to get a latency curve for a multi-level VH

Two-level hierarchies form a latency *surface!*



Project

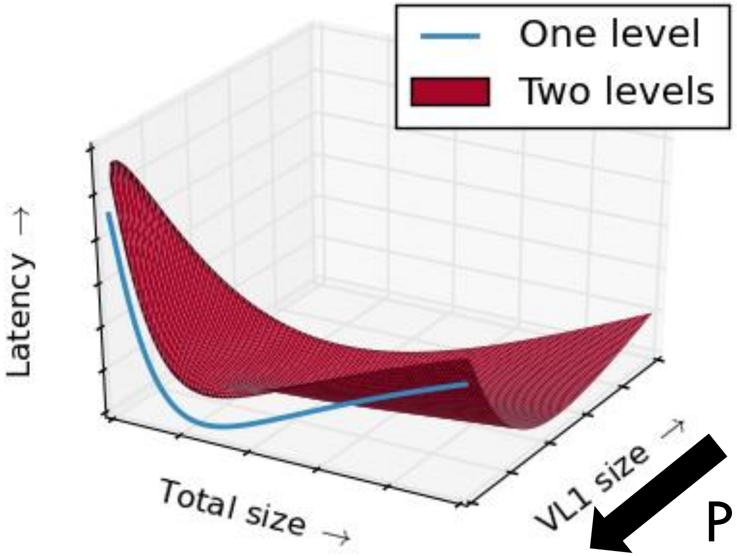


Best 1- and 2-level hierarchy at every size

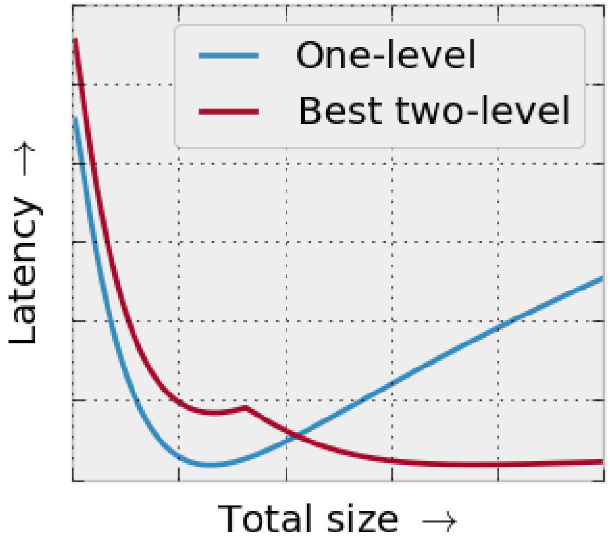
Best overall hierarchy at every size

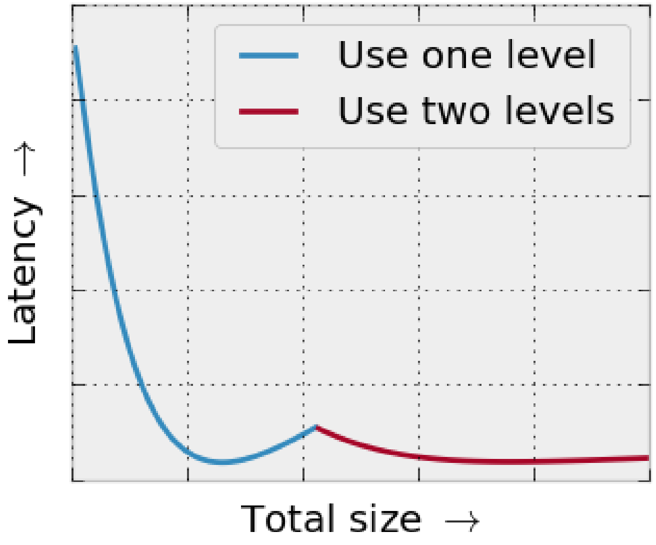
How to get a latency curve for a multi-level VH

Two-level hierarchies form a latency *surface!*



Project



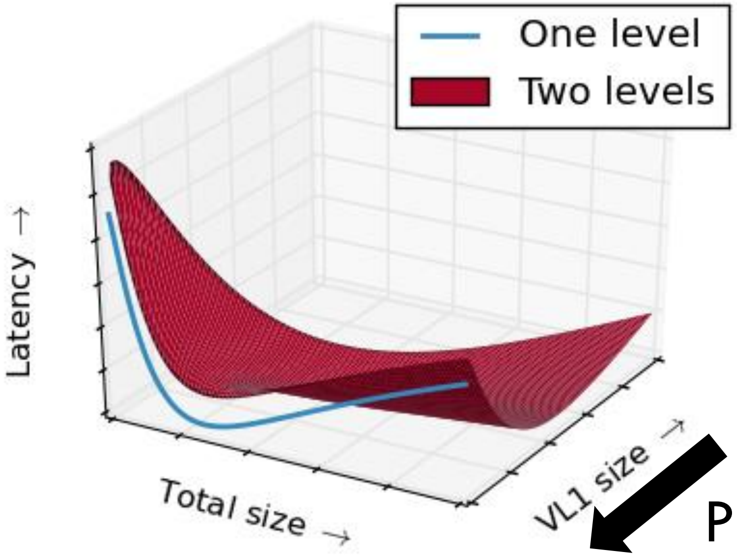


Best 1- and 2-level hierarchy at every size

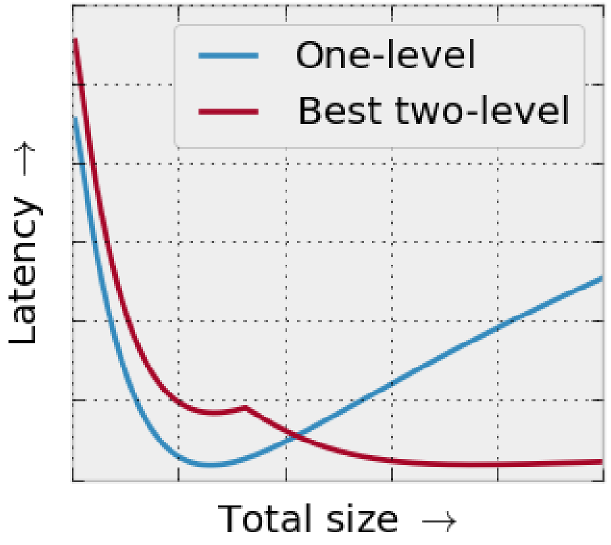
Best overall hierarchy at every size

How to get a latency curve for a multi-level VH

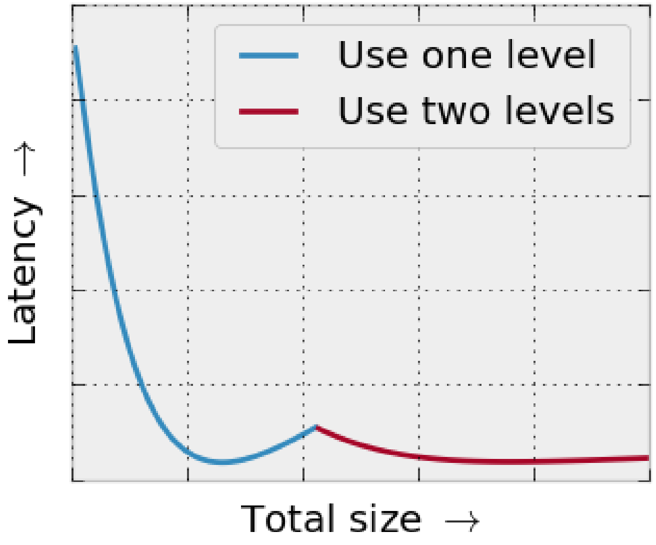
Two-level hierarchies form a latency *surface!*



Project



Curve lets us optimize multi-level hierarchies!

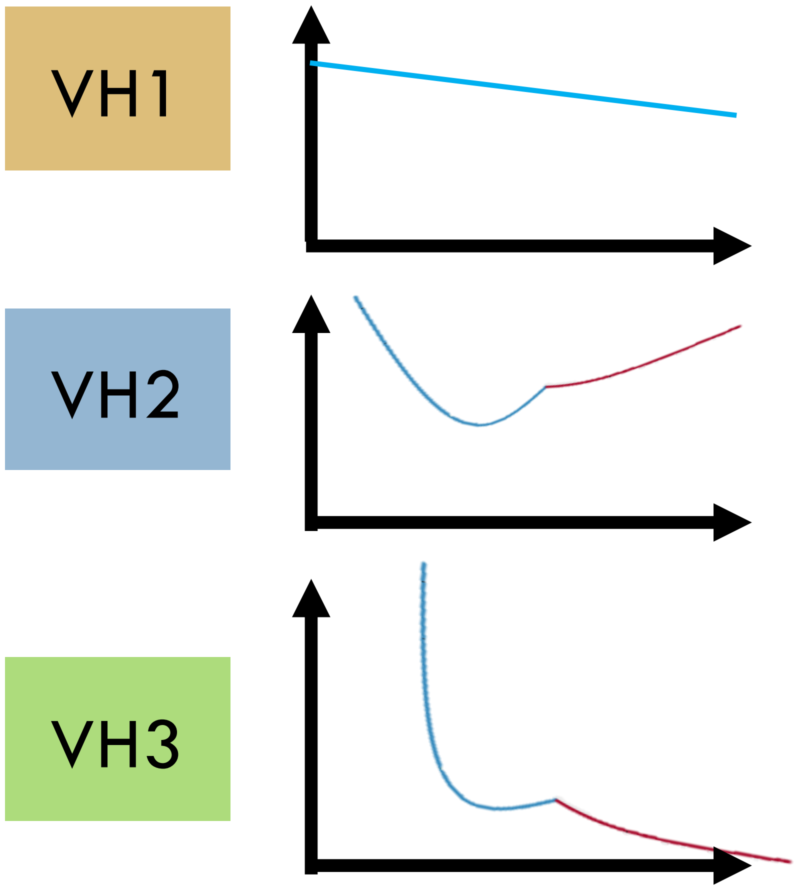


Best 1- and 2-level hierarchy at every size

Best overall hierarchy at every size

Allocating virtual hierarchies

Latency curves

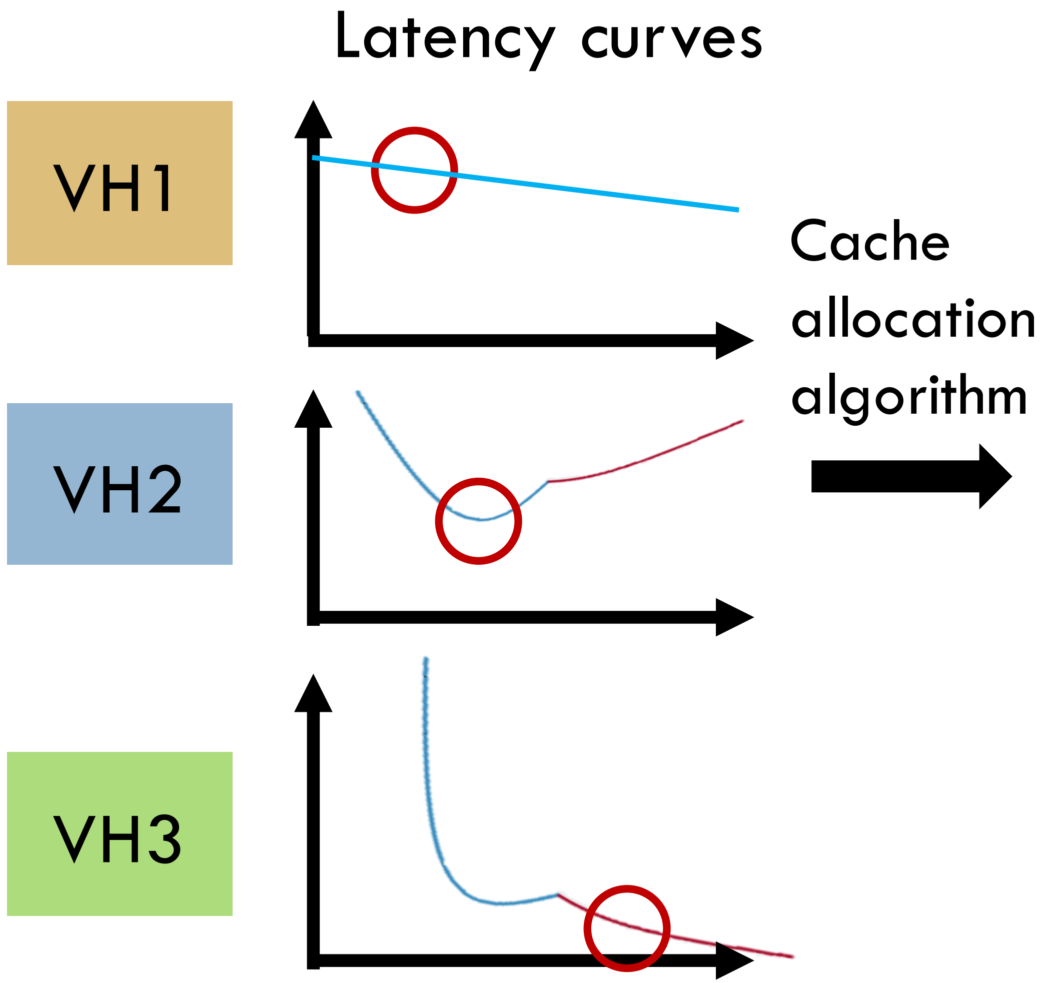


VH1

VH2

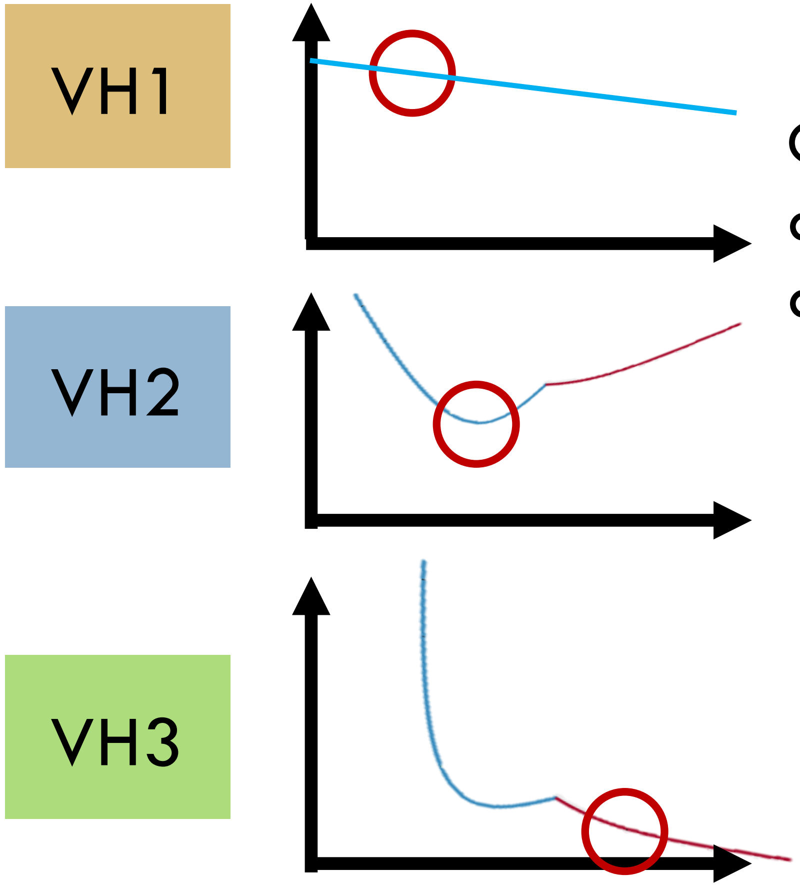
VH3

Allocating virtual hierarchies



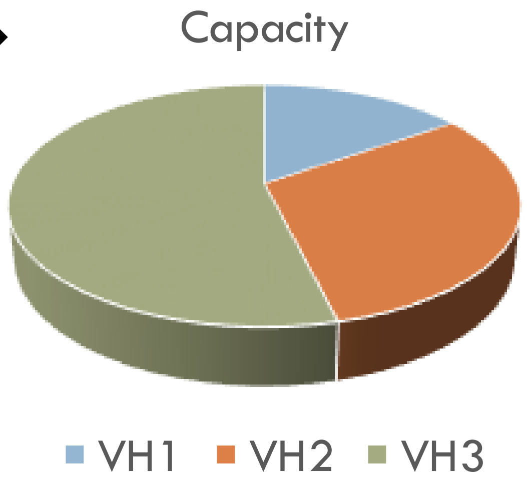
Allocating virtual hierarchies

Latency curves

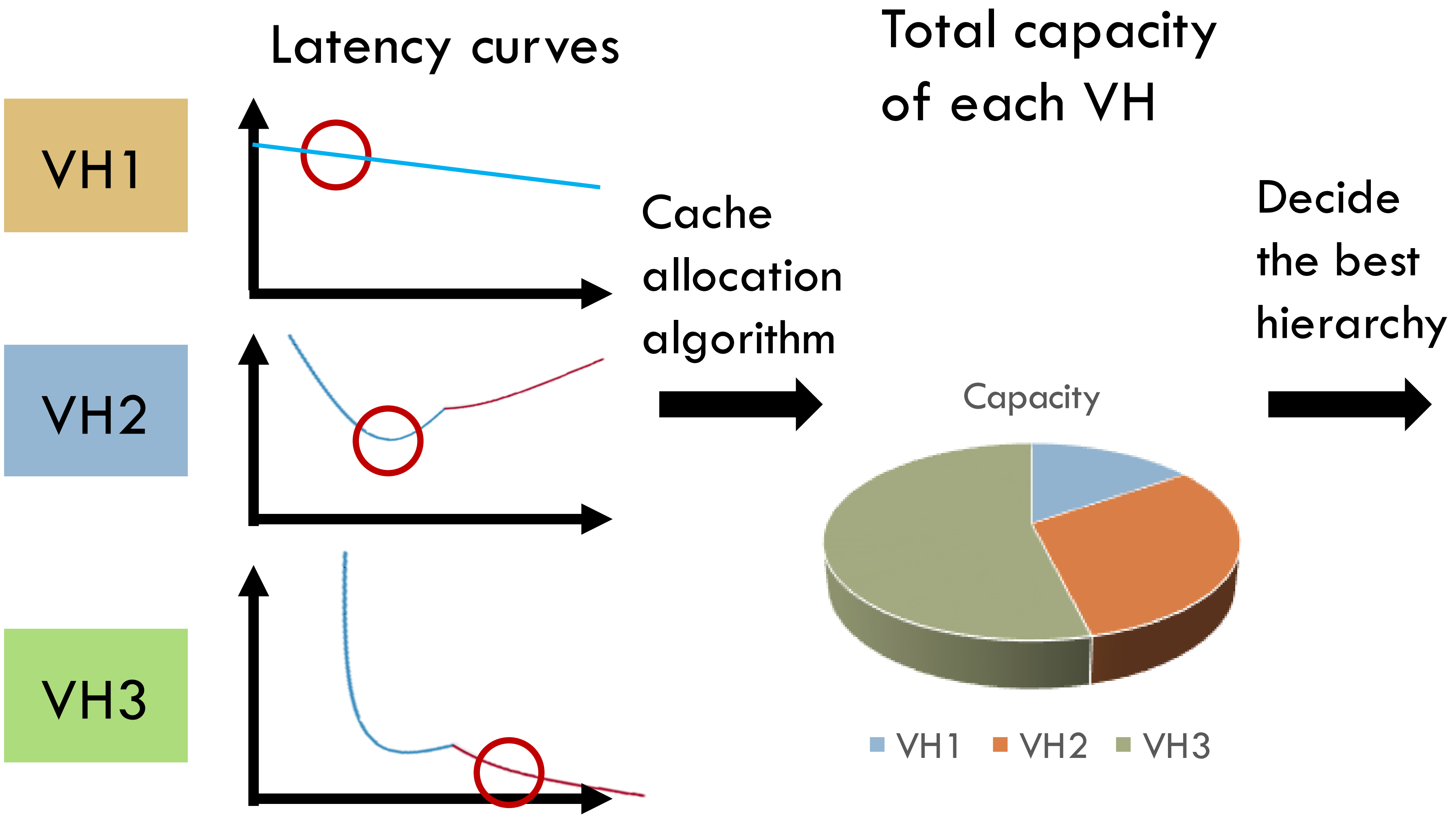


Cache allocation algorithm

Total capacity of each VH



Allocating virtual hierarchies



Latency curves

VH1

VH2

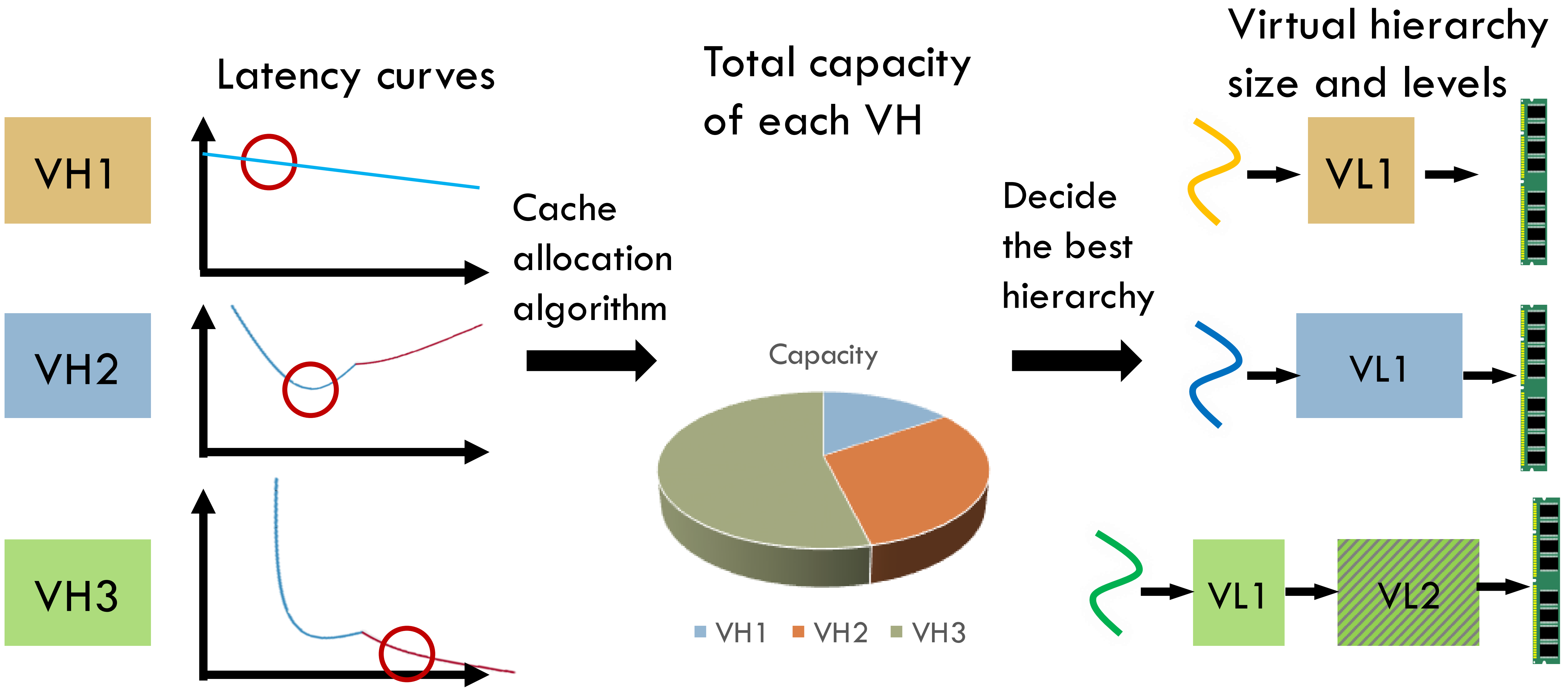
VH3

Cache allocation algorithm

Total capacity of each VH

Decide the best hierarchy

Allocating virtual hierarchies



Latency curves

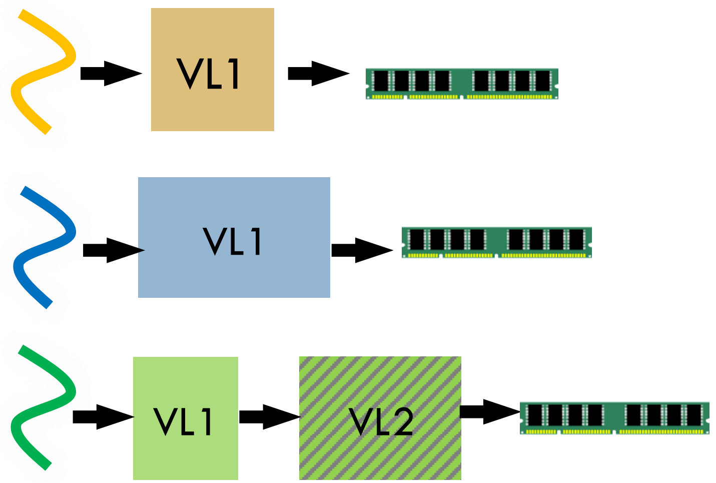
Cache allocation algorithm

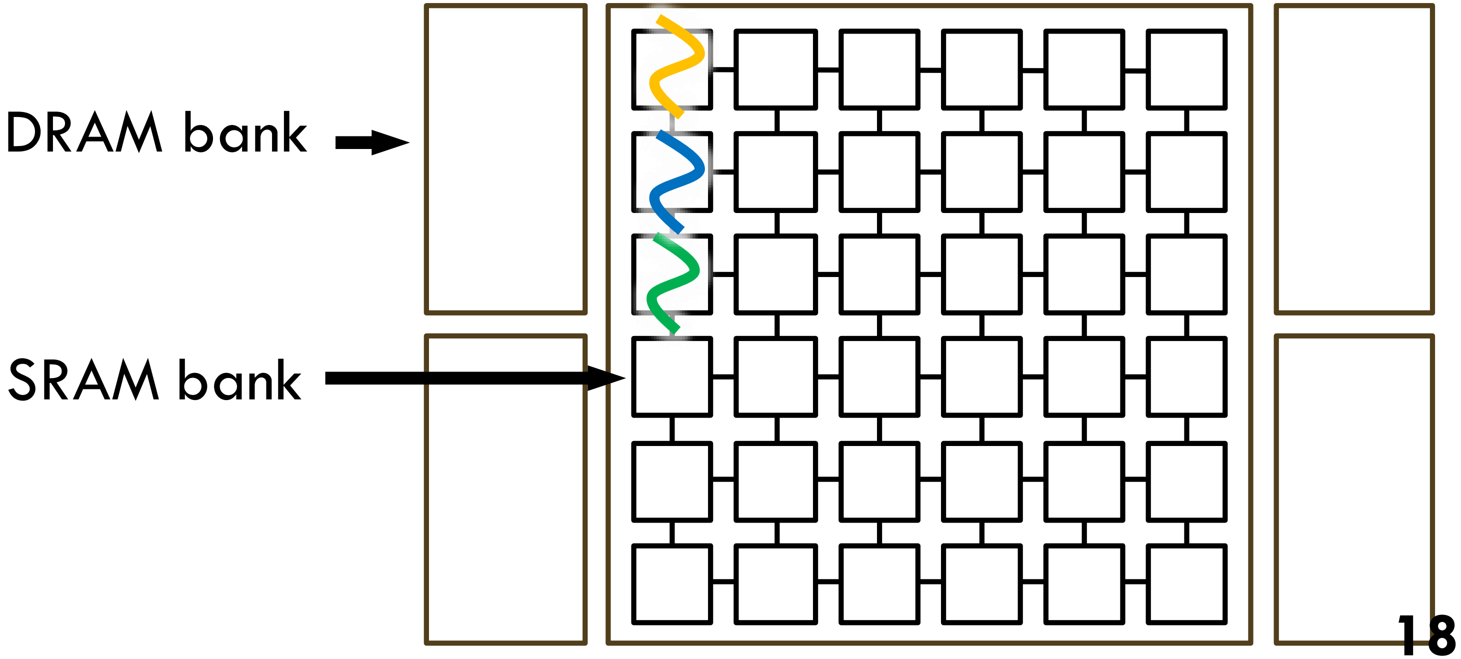
Total capacity of each VH

Decide the best hierarchy

Virtual hierarchy size and levels

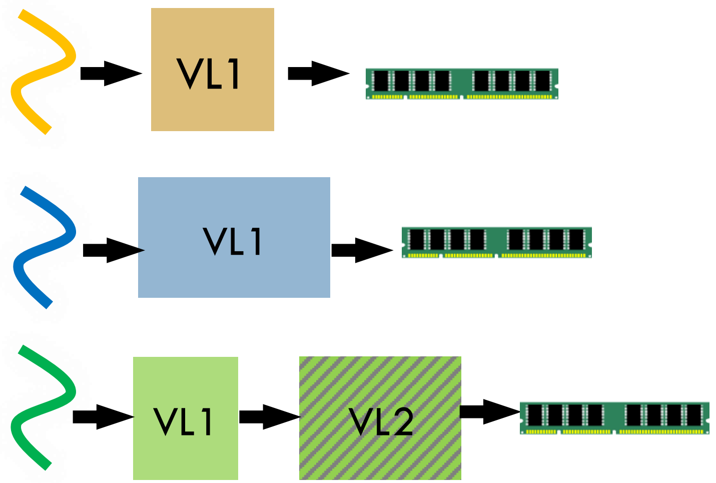
Bandwidth-aware virtual hierarchy placement

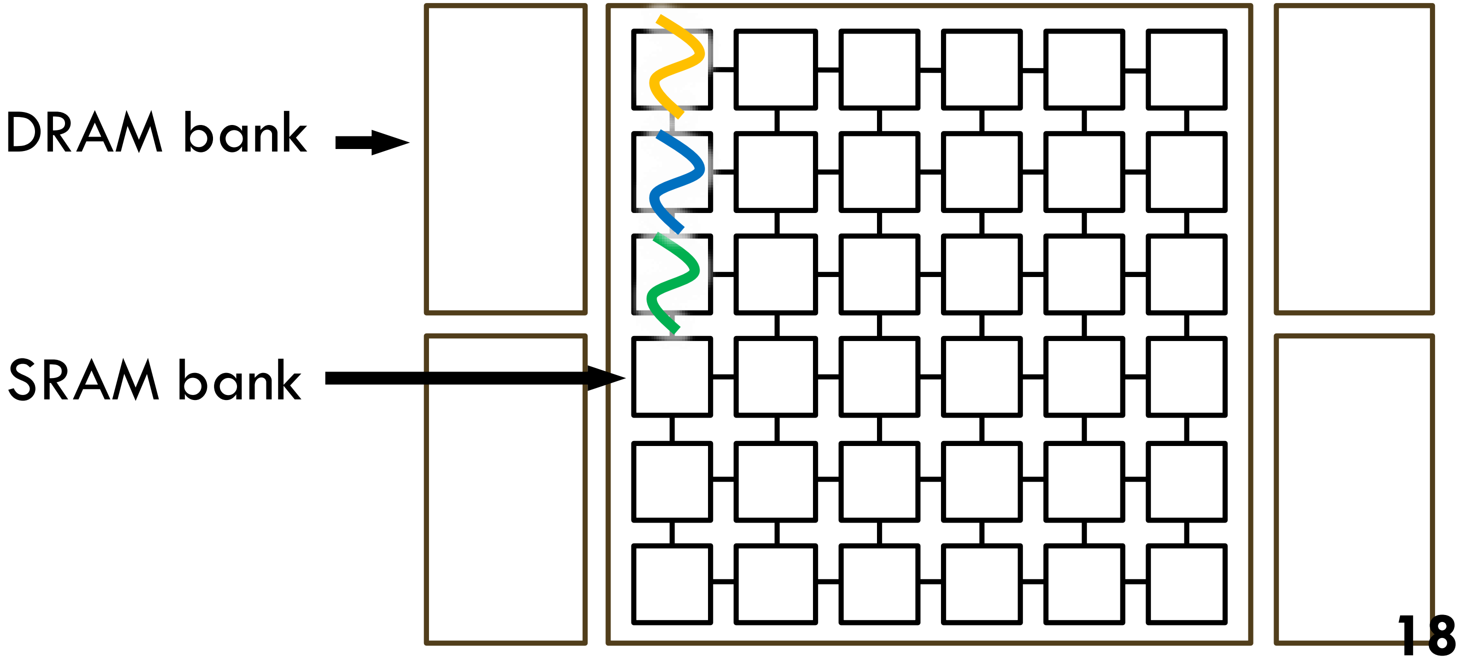




Bandwidth-aware virtual hierarchy placement

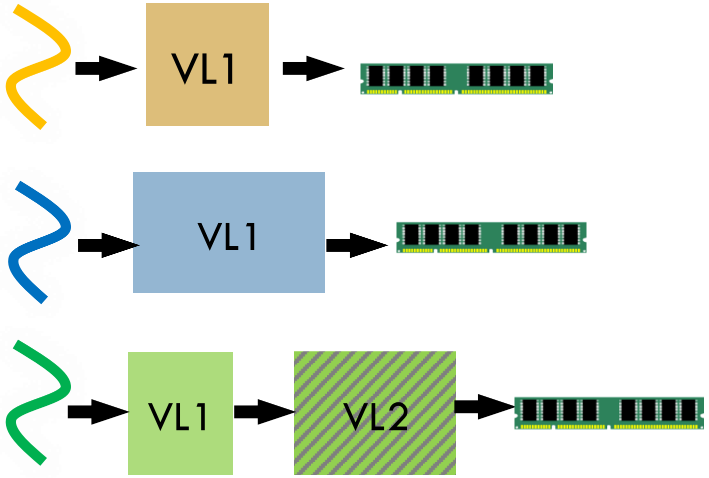
□ Place data close without saturating DRAM bandwidth

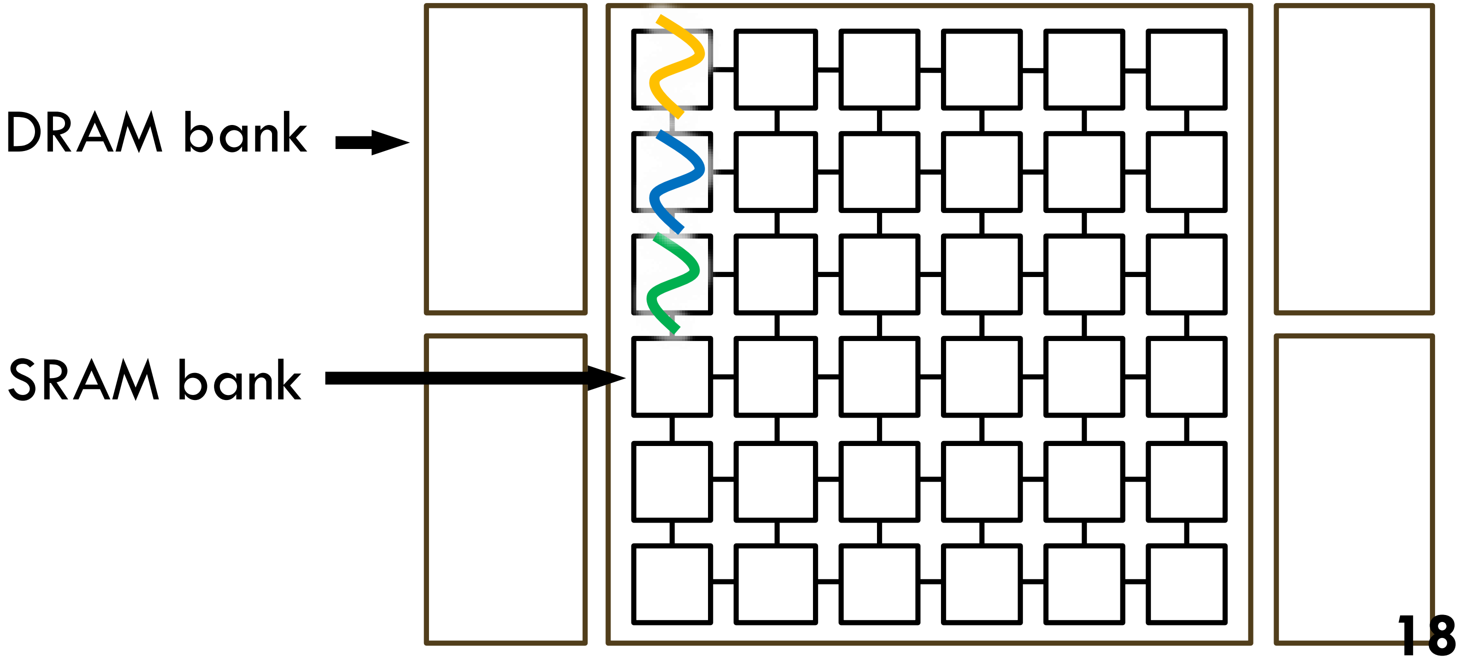




Bandwidth-aware virtual hierarchy placement

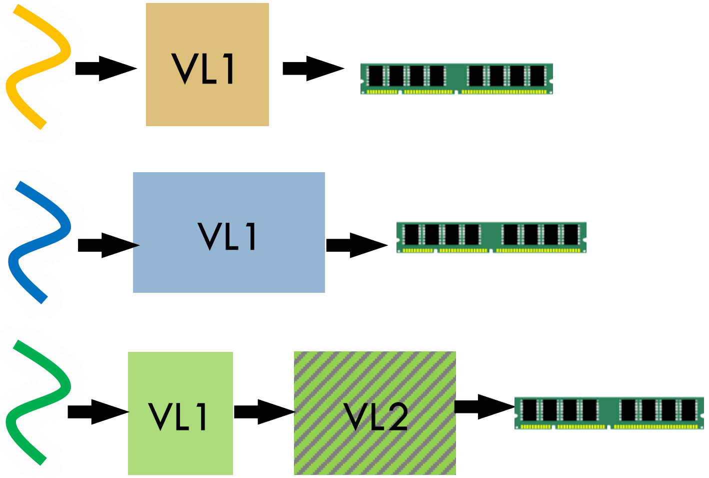
- Place data close without saturating DRAM bandwidth
- Every iteration, Jenga …
  - Chooses a VH (via an opportunity cost metric, see paper)
  - Greedily places a chunk of its data in its closest bank
  - Update DRAM bank latency

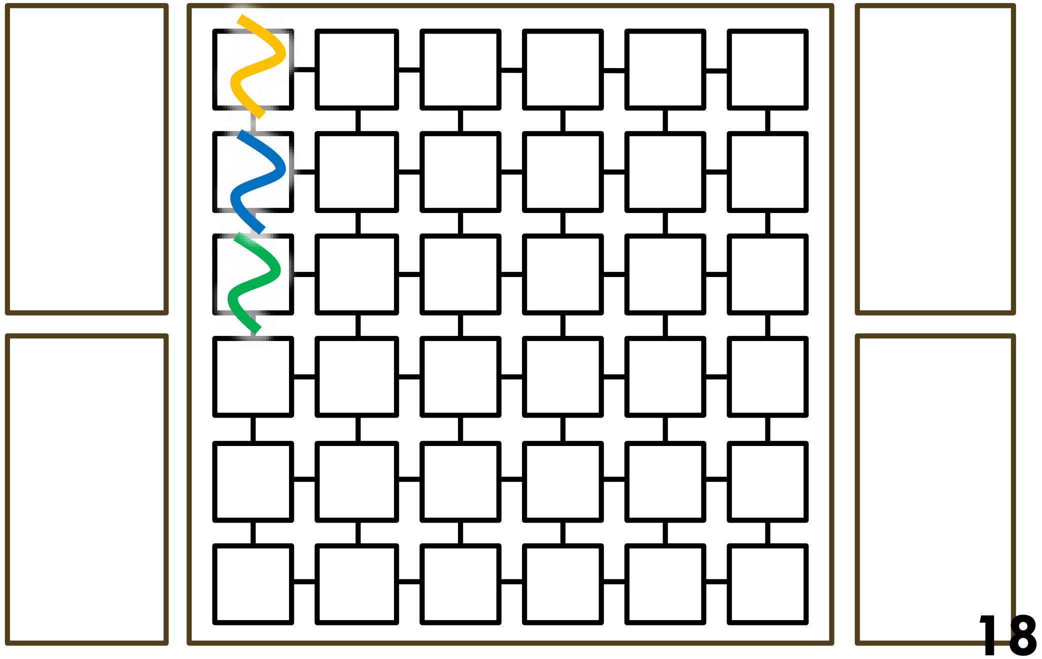




Bandwidth-aware virtual hierarchy placement

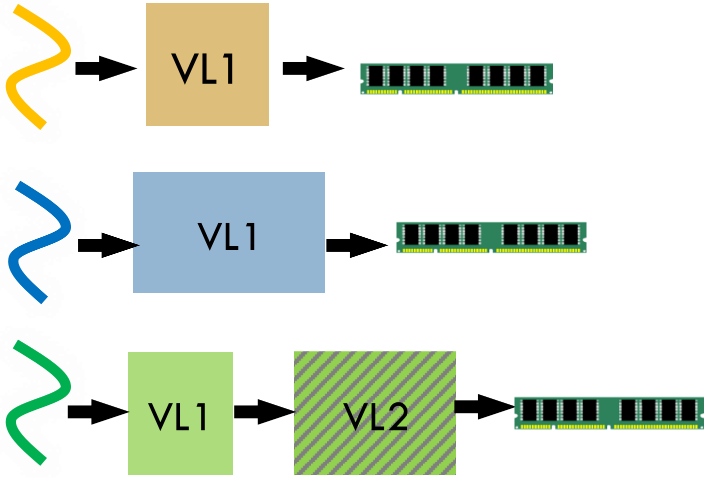
- Place data close without saturating DRAM bandwidth
- Every iteration, Jenga …
  - Chooses a VH (via an opportunity cost metric, see paper)
  - Greedily places a chunk of its data in its closest bank
  - Update DRAM bank latency

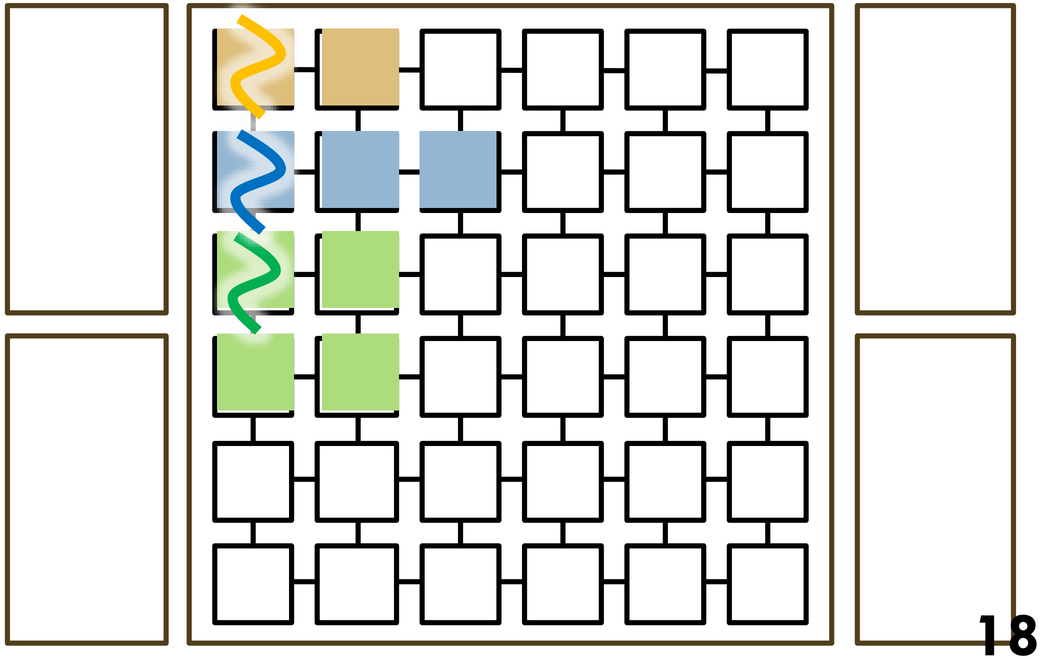




Bandwidth-aware virtual hierarchy placement

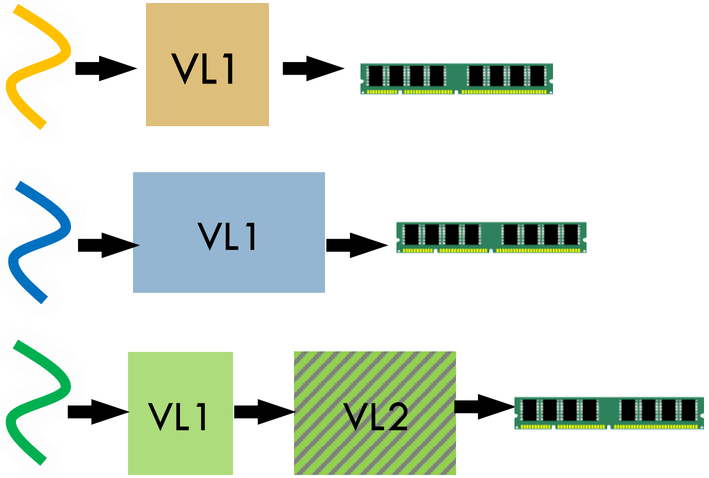
- Place data close without saturating DRAM bandwidth
- Every iteration, Jenga …
  - Chooses a VH (via an opportunity cost metric, see paper)
  - Greedily places a chunk of its data in its closest bank
  - Update DRAM bank latency





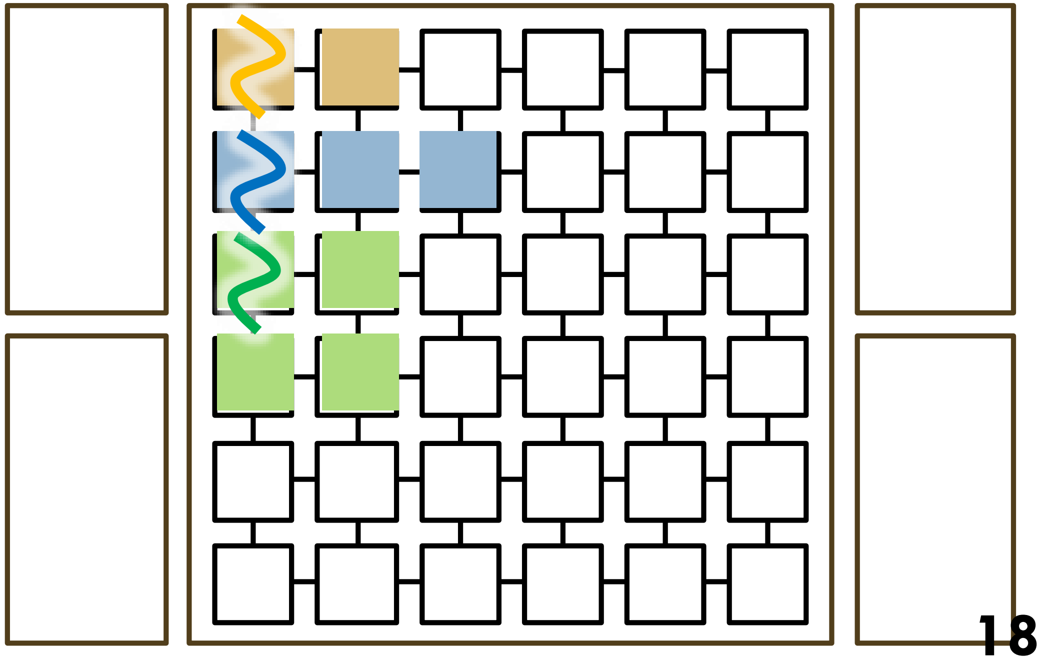
Bandwidth-aware virtual hierarchy placement

- Place data close without saturating DRAM bandwidth
- Every iteration, Jenga …
  - Chooses a VH (via an opportunity cost metric, see paper)
  - Greedily places a chunk of its data in its closest bank
  - Update DRAM bank latency



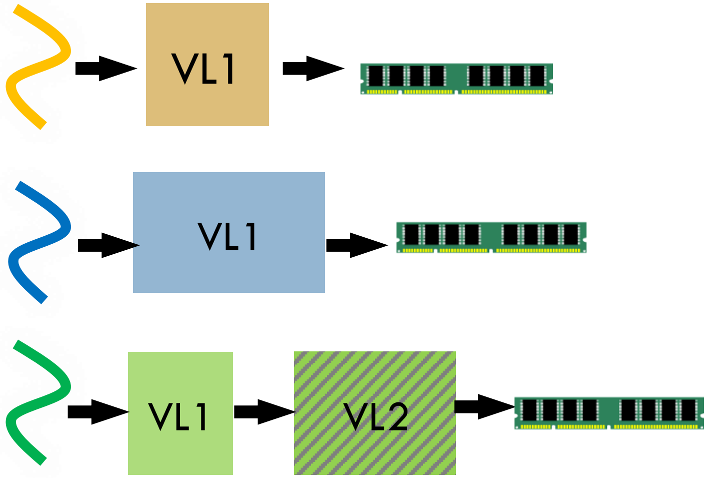
1.0X Latency

1.0X Latency



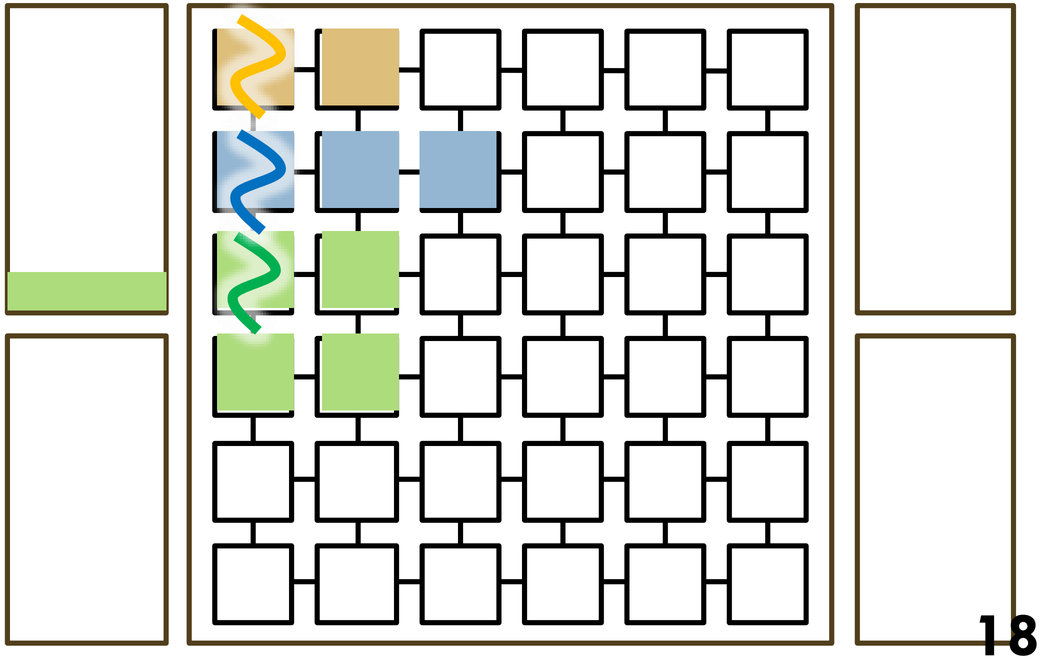
Bandwidth-aware virtual hierarchy placement

- Place data close without saturating DRAM bandwidth
- Every iteration, Jenga …
  - Chooses a VH (via an opportunity cost metric, see paper)
  - Greedily places a chunk of its data in its closest bank
  - Update DRAM bank latency



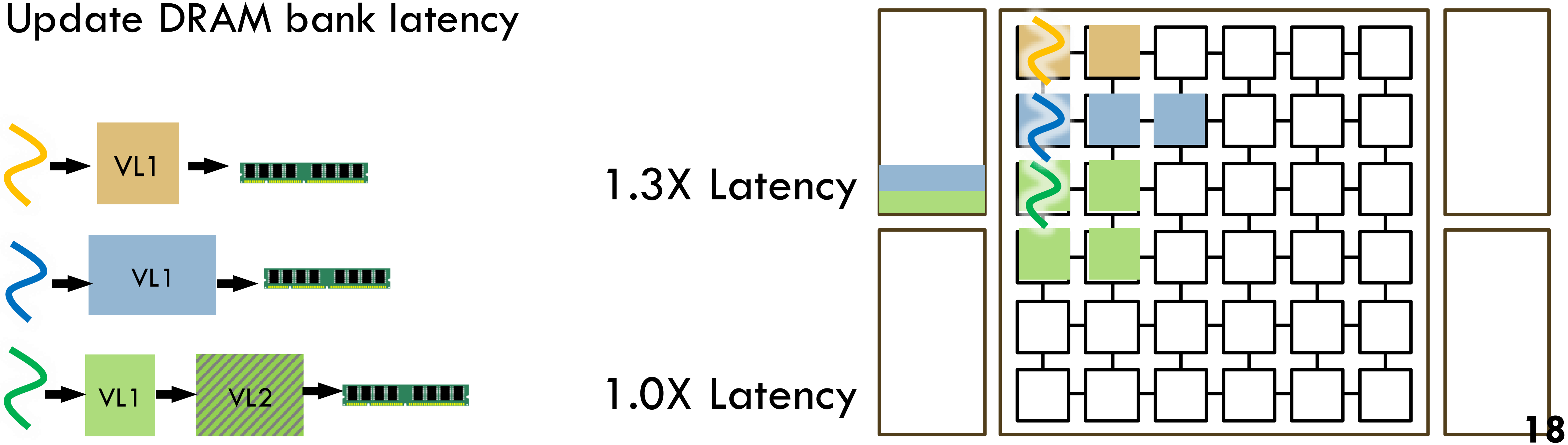
1.1X Latency

1.0X Latency



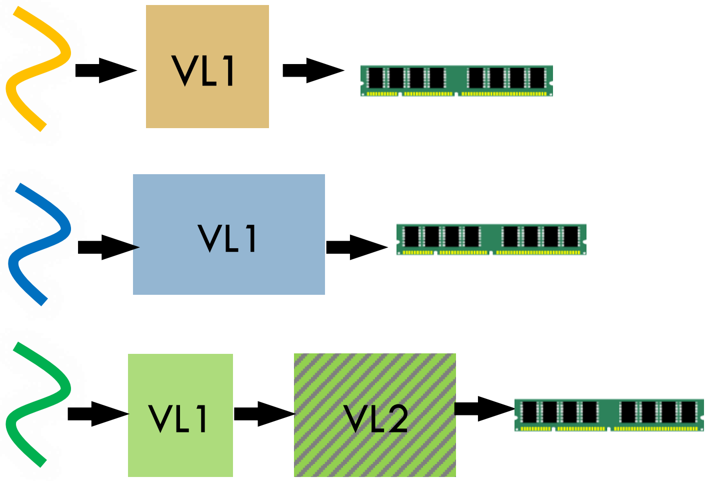
Bandwidth-aware virtual hierarchy placement

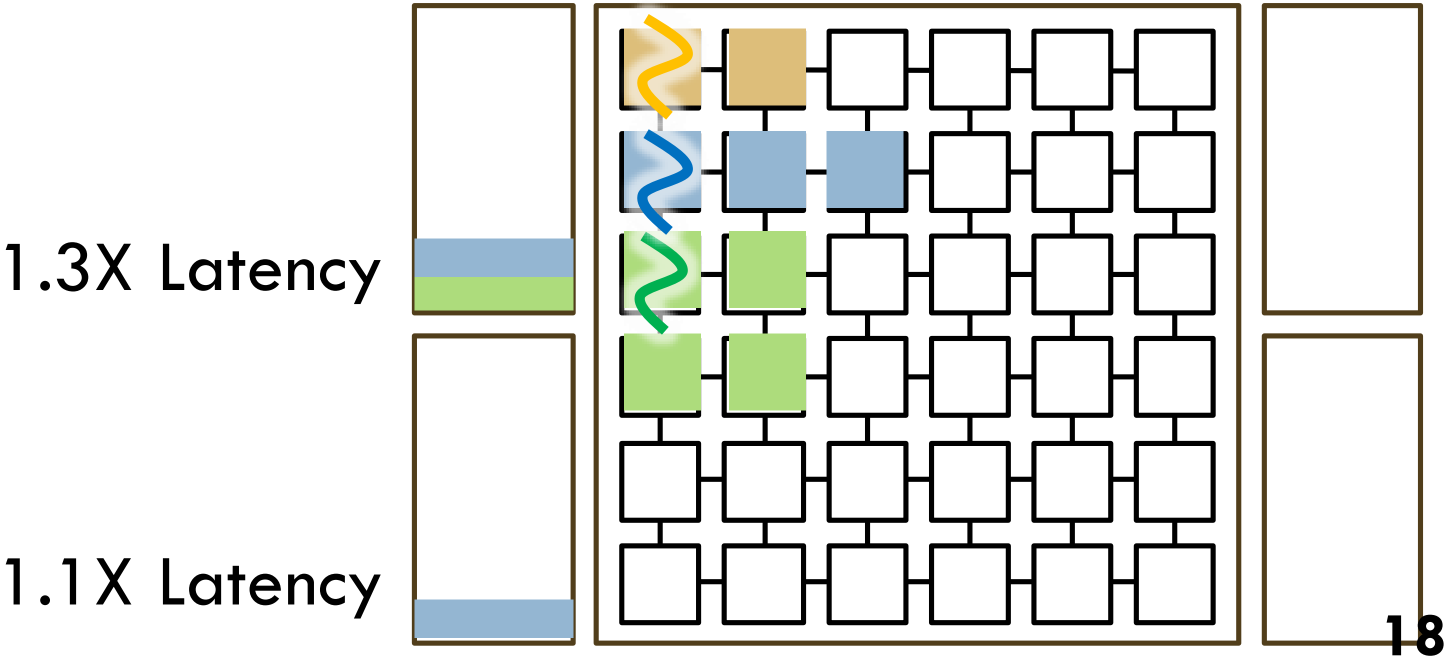
- Place data close without saturating DRAM bandwidth
- Every iteration, Jenga …
  - Chooses a VH (via an opportunity cost metric, see paper)
  - Greedily places a chunk of its data in its closest bank
  - Update DRAM bank latency



Bandwidth-aware virtual hierarchy placement

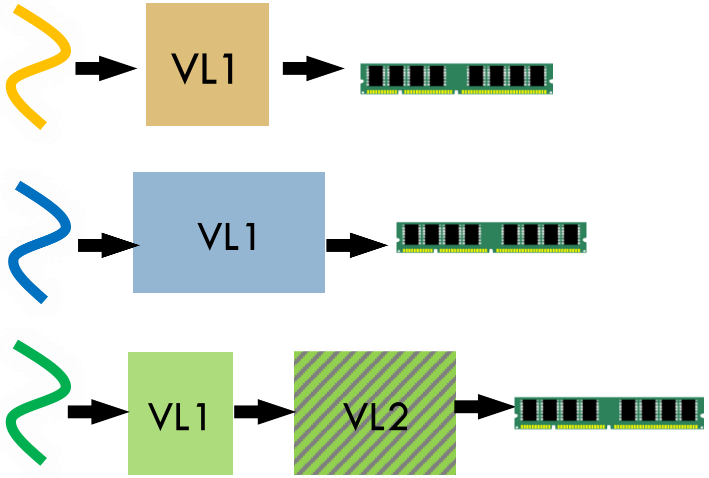
- Place data close without saturating DRAM bandwidth
- Every iteration, Jenga …
  - Chooses a VH (via an opportunity cost metric, see paper)
  - Greedily places a chunk of its data in its closest bank
  - Update DRAM bank latency





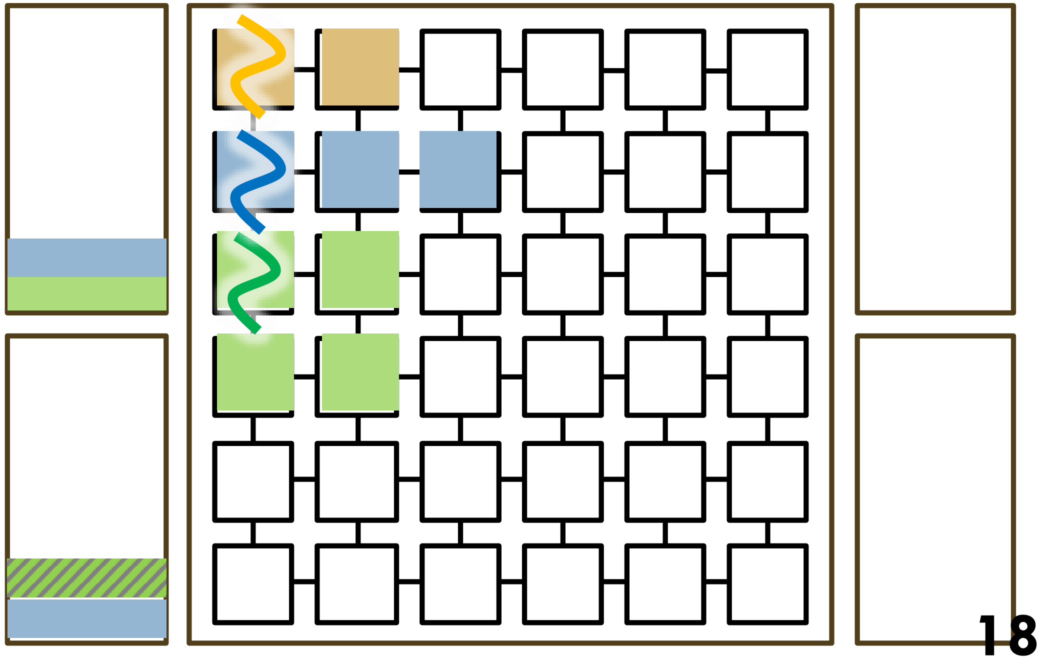
Bandwidth-aware virtual hierarchy placement

- Place data close without saturating DRAM bandwidth
- Every iteration, Jenga …
  - Chooses a VH (via an opportunity cost metric, see paper)
  - Greedily places a chunk of its data in its closest bank
  - Update DRAM bank latency



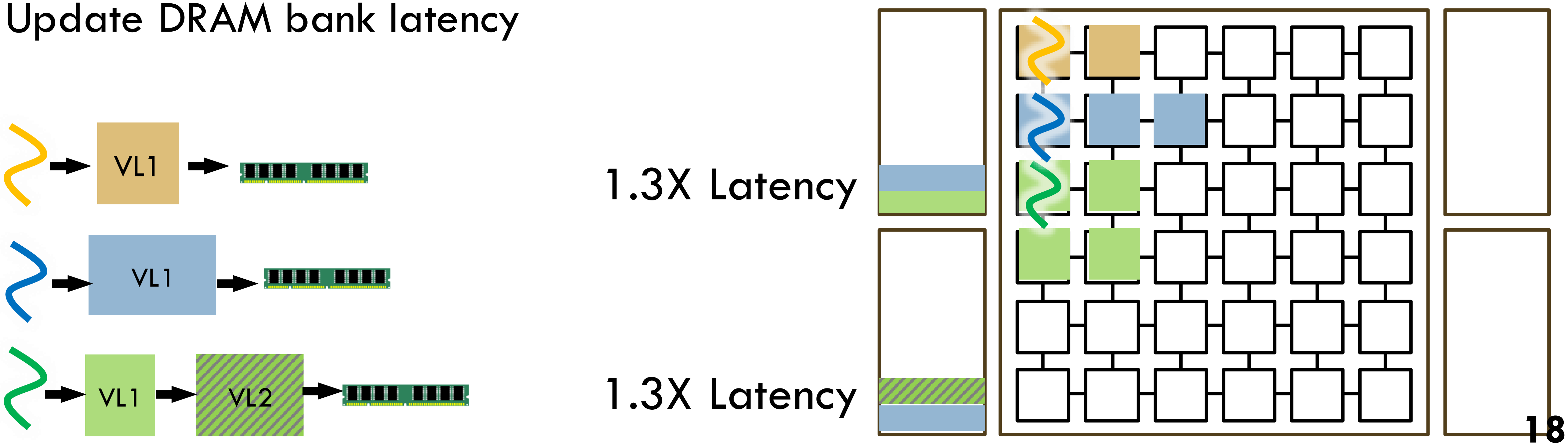
1.3X Latency

1.3X Latency



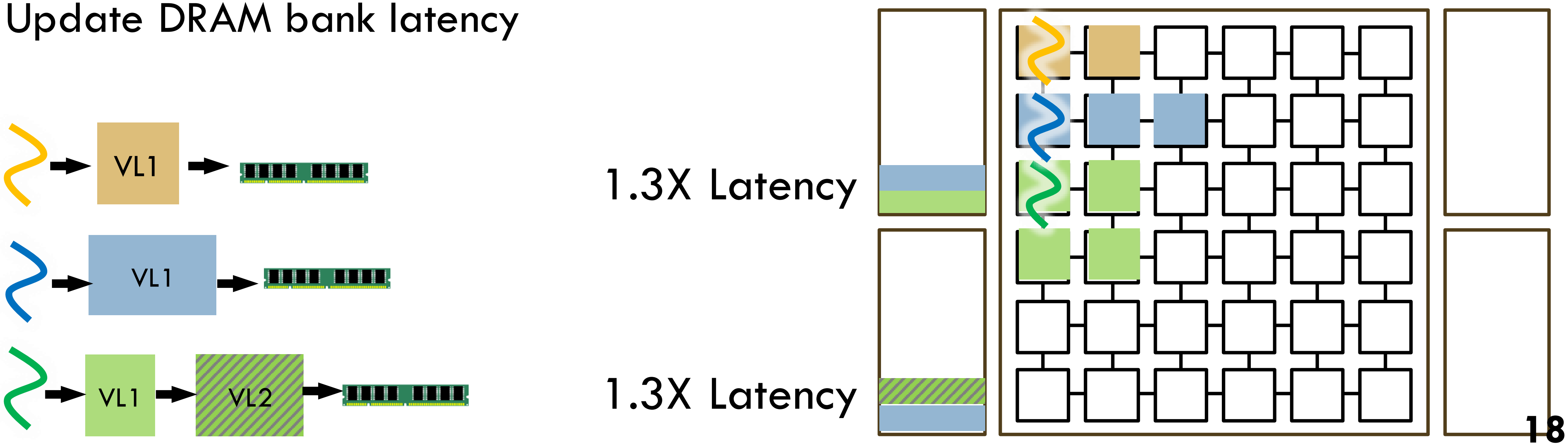
Bandwidth-aware virtual hierarchy placement

- Place data close without saturating DRAM bandwidth
- Every iteration, Jenga …
  - Chooses a VH (via an opportunity cost metric, see paper)
  - Greedily places a chunk of its data in its closest bank
  - Update DRAM bank latency



Bandwidth-aware virtual hierarchy placement

- Place data close without saturating DRAM bandwidth
- Every iteration, Jenga …
  - Chooses a VH (via an opportunity cost metric, see paper)
  - Greedily places a chunk of its data in its closest bank
  - Update DRAM bank latency



Jenga adds small overheads

Jenga adds small overheads

- Hardware overheads
  - VHT requires ~2.4 KB/tile
  - Monitors are 8 KB x 2/tile
  - In total, Jenga adds ~20 KB per tile, **4%** of the SRAM banks
  - Similar to Jigsaw

Jenga adds small overheads

- Hardware overheads
  - VHT requires ~2.4 KB/tile
  - Monitors are 8 KB x 2/tile
  - In total, Jenga adds ~20 KB per tile, **4%** of the SRAM banks
  - Similar to Jigsaw

- Software overheads
  - **0.4%** of system cycles at 36 tiles
  - Runs concurrently with applications; only needs to pause cores to update VHTs
  - Trivial to parallelize

See paper for …

- Hardware support for
  - Fast reconfiguration
  - Page reclassification
- Efficient implementation of hierarchy allocation
- OS integration

Evaluation

Evaluation

- Modeled system
  - 36 cores on 6x6 mesh
  - 18MB SRAM
  - 1GB Stacked DRAM

Evaluation

- Modeled system
  - 36 cores on 6x6 mesh
  - 18MB SRAM
  - 1GB Stacked DRAM

- Workloads
  - **36 copies of same app (SPECrate)**
  - **Random 36 SPECCPU apps mixes**
  -

Evaluation

- Modeled system
  - 36 cores on 6x6 mesh
  - 18MB SRAM
  - 1GB Stacked DRAM

- Workloads
  - **36 copies of same app (SPECrate)**
  - **Random 36 SPECCPU apps mixes**
  -

- Compared 5 schemes

	SRAM	DRAM
S-NUCA	Rigid L3	-
Alloy	Rigid L3	Rigid L4
Jigsaw	App-specific L3	-
JigAlloy	App-specific L3	Rigid L4
Jenga	App-specific Virtual Hierarchies	

Case study: 36 copies of xalanc

Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB

Case study: 36 copies of xalanc

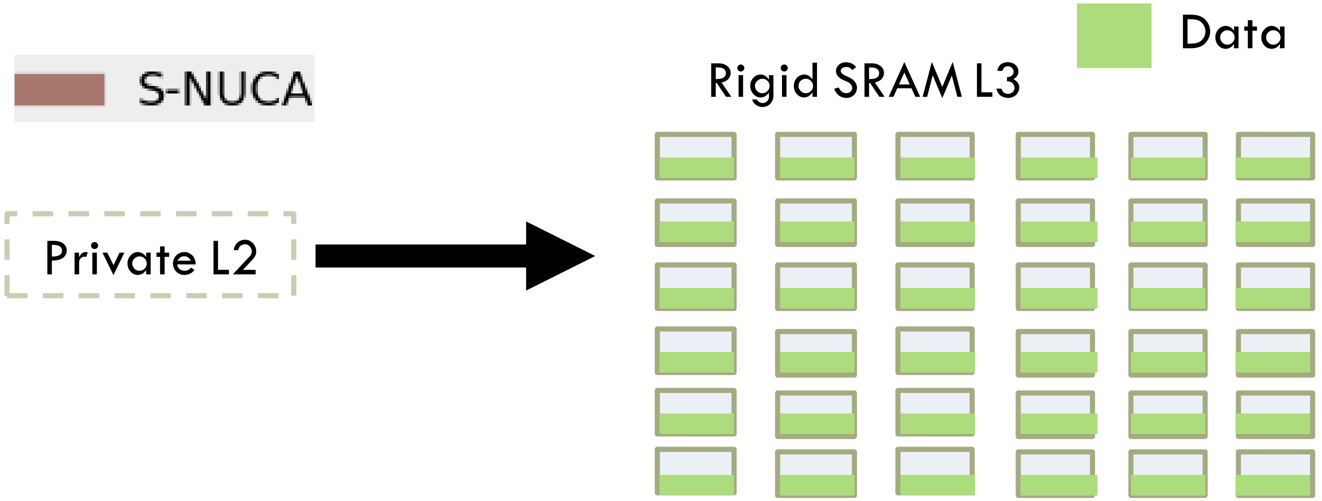
Working set: 6MB x 36 = 216 MB

🟫 S-NUCA

Private L2 →

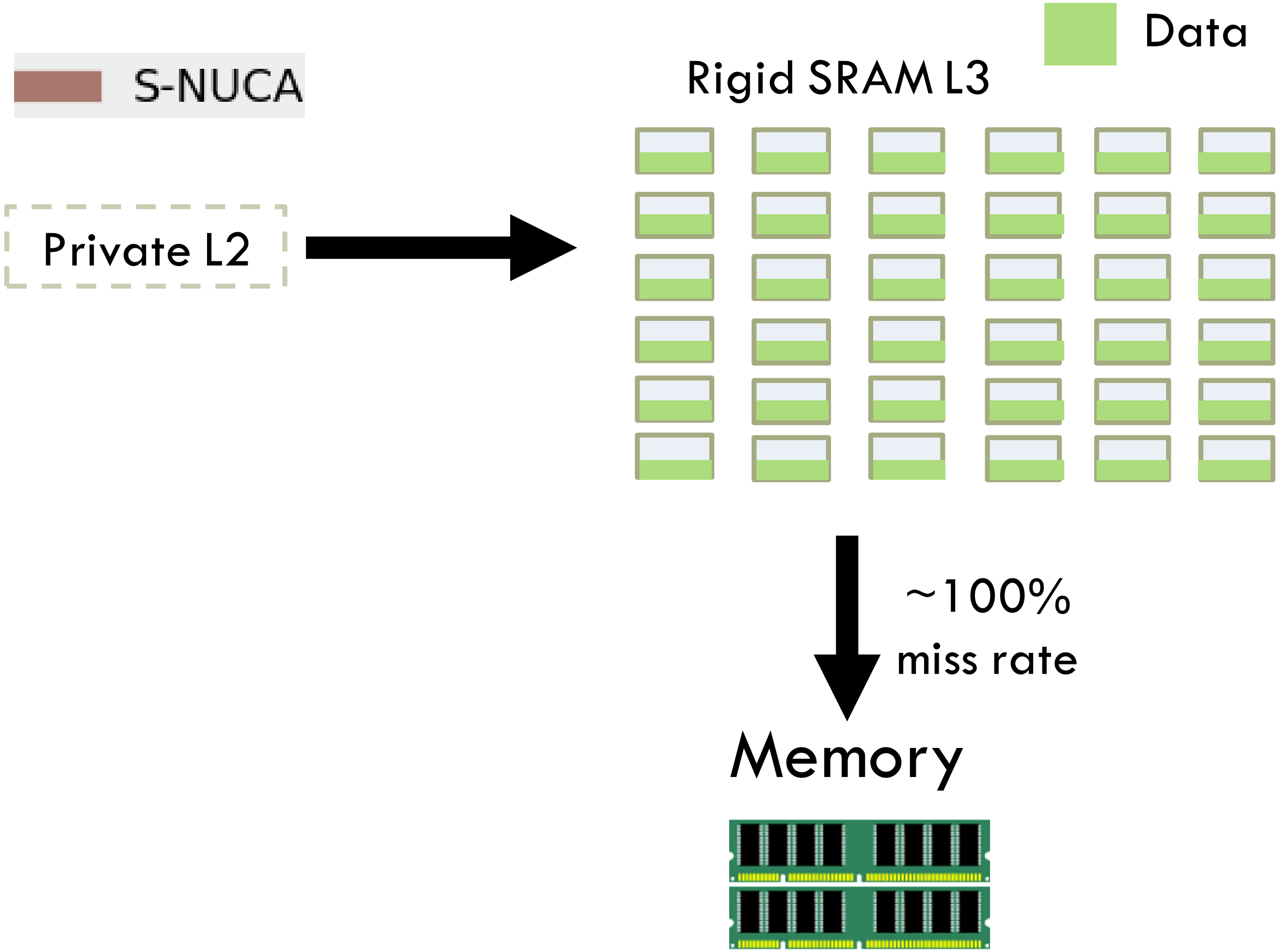
Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB



Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB



🟩 Data

🟫 S-NUCA

Rigid SRAM L3

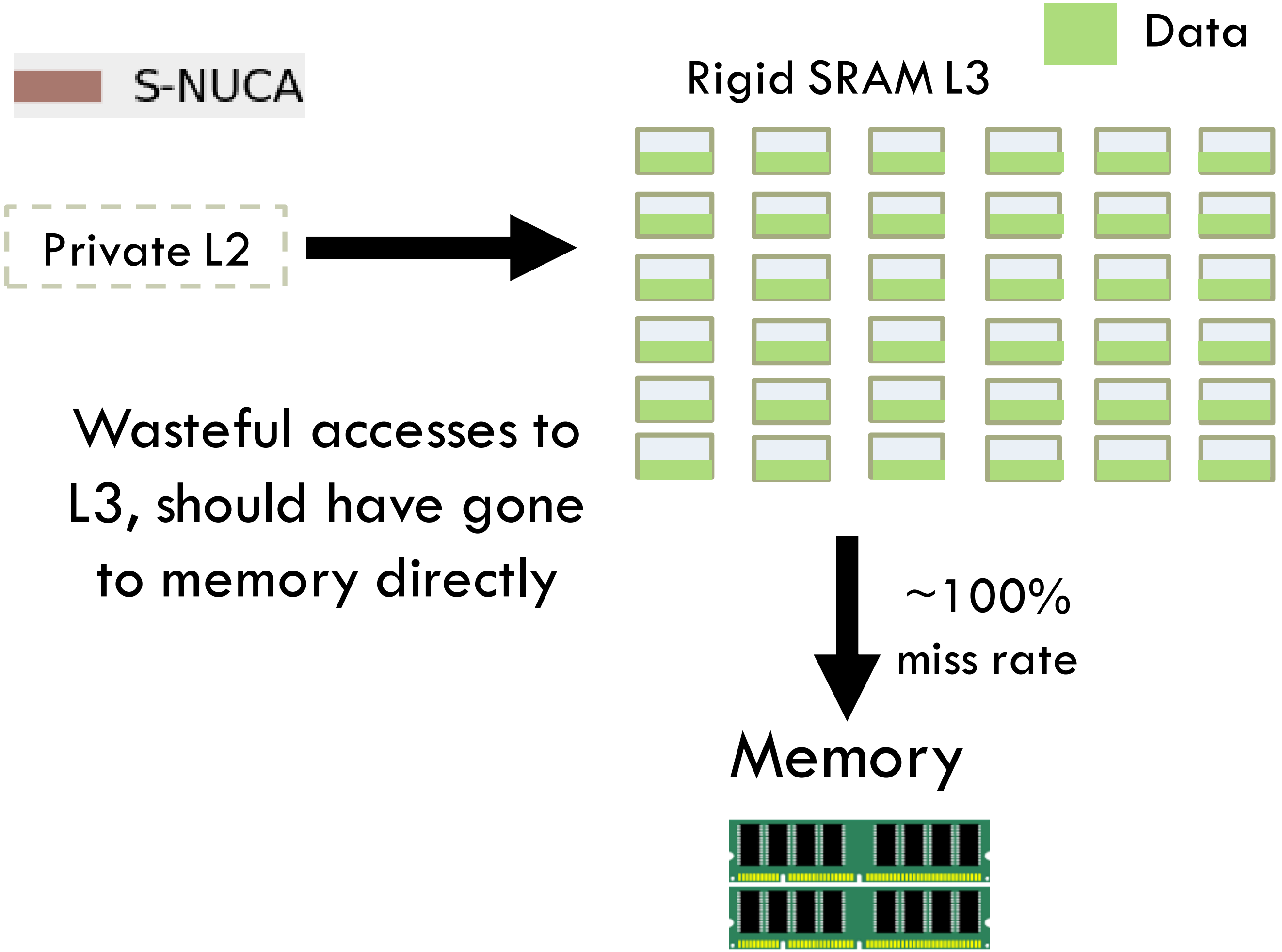
Private L2

~100% miss rate

Memory

Case study: 36 copies of xalanc

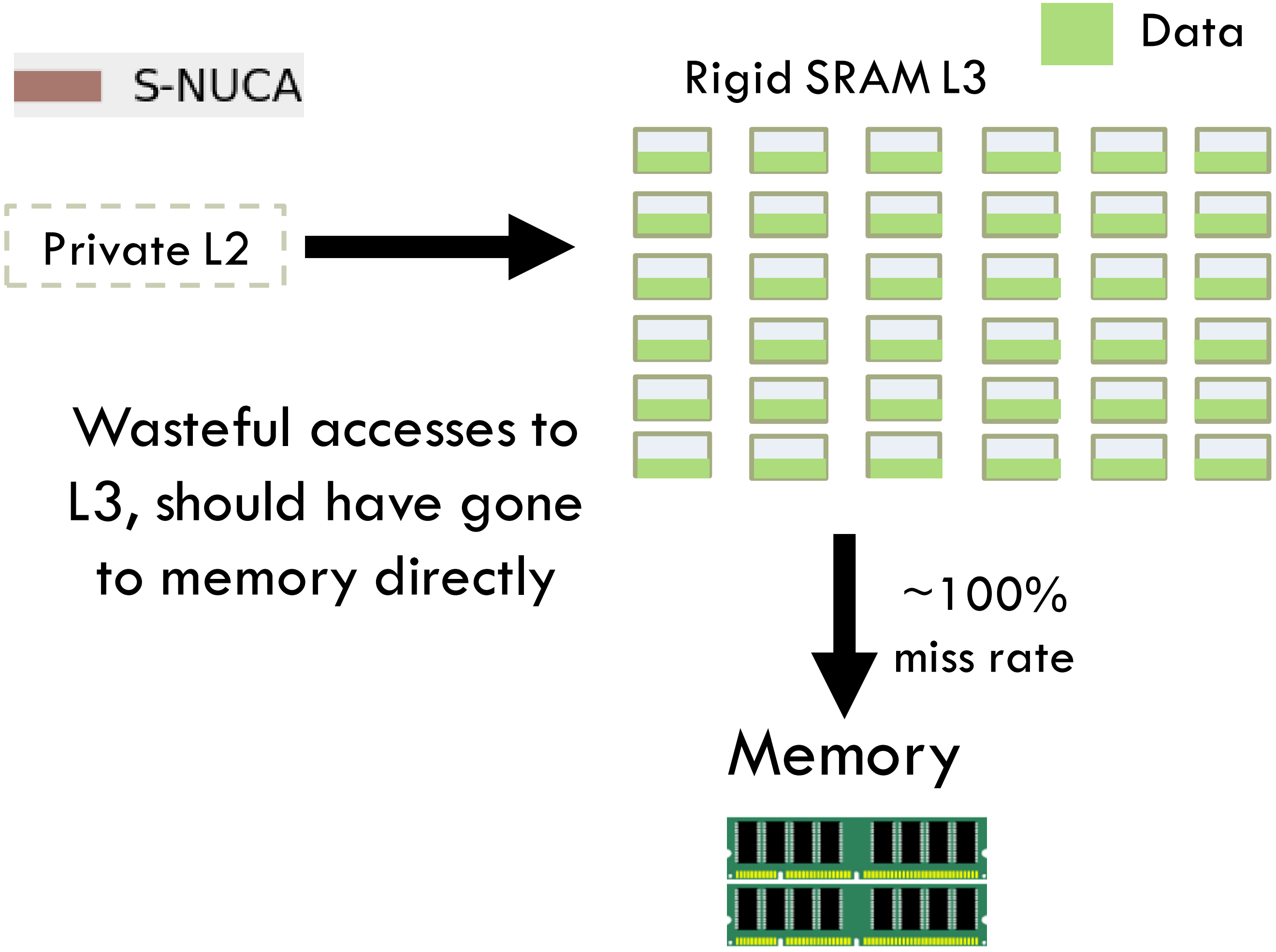
Working set: 6MB x 36 = 216 MB



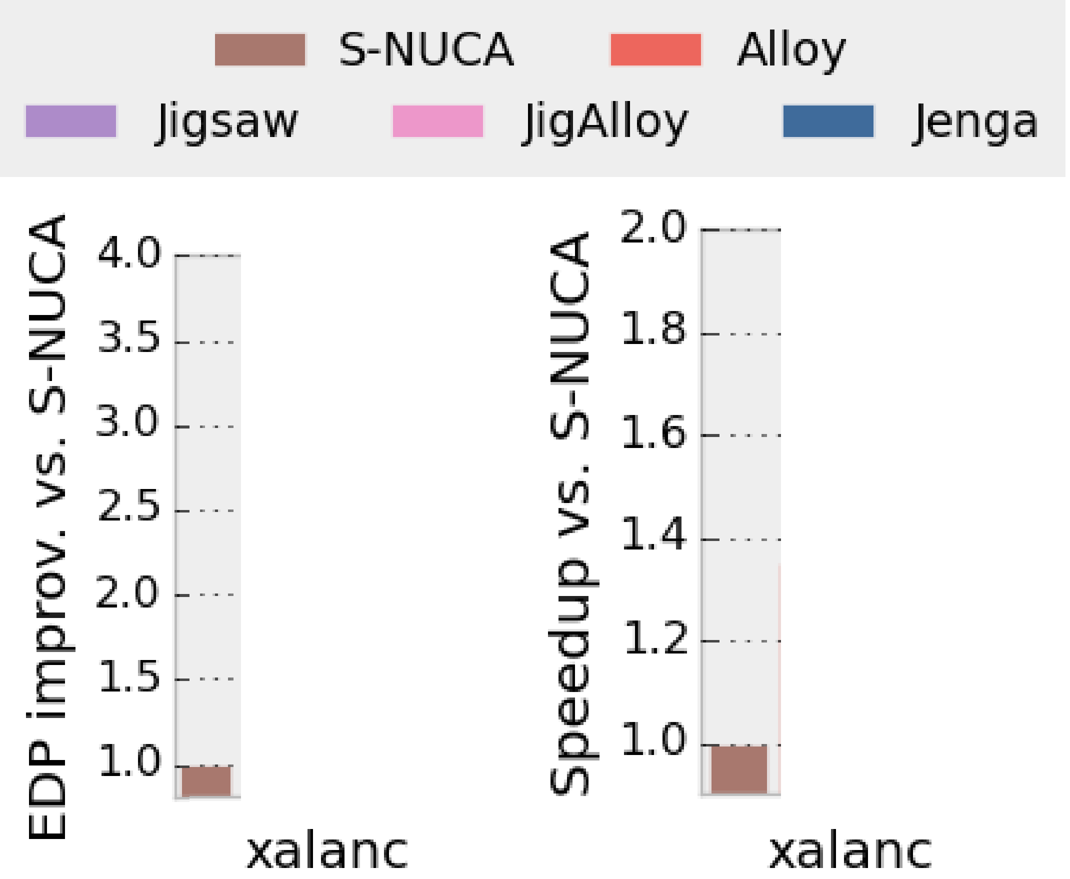
Wasteful accesses to L3, should have gone to memory directly

Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB



Wasteful accesses to L3, should have gone to memory directly

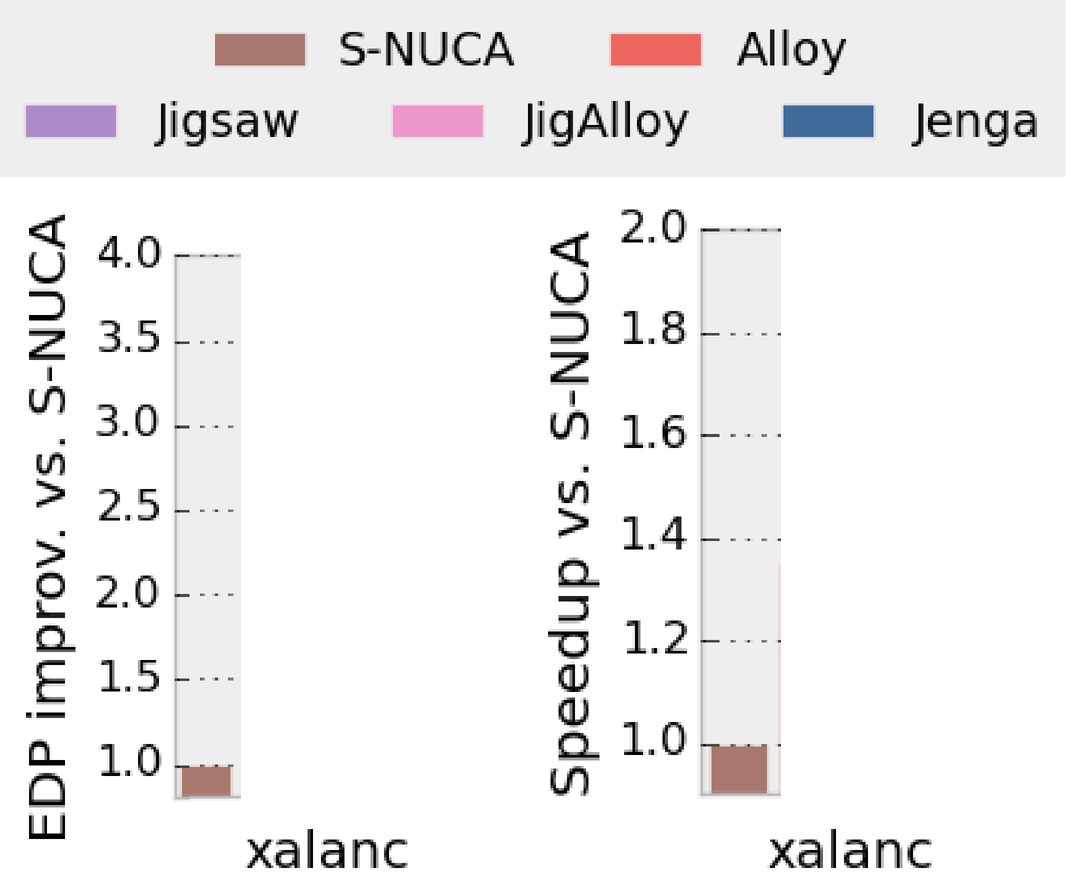


Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB

🟥 Alloy

Private L2

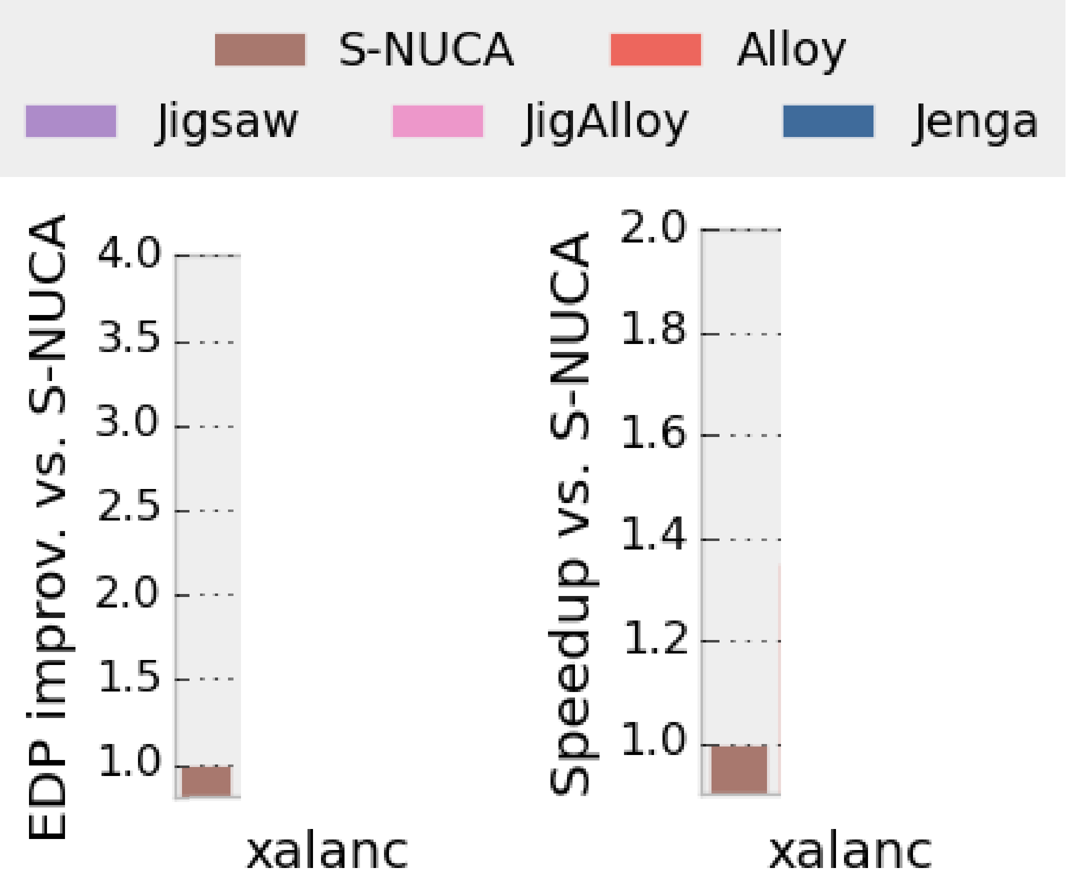


Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB

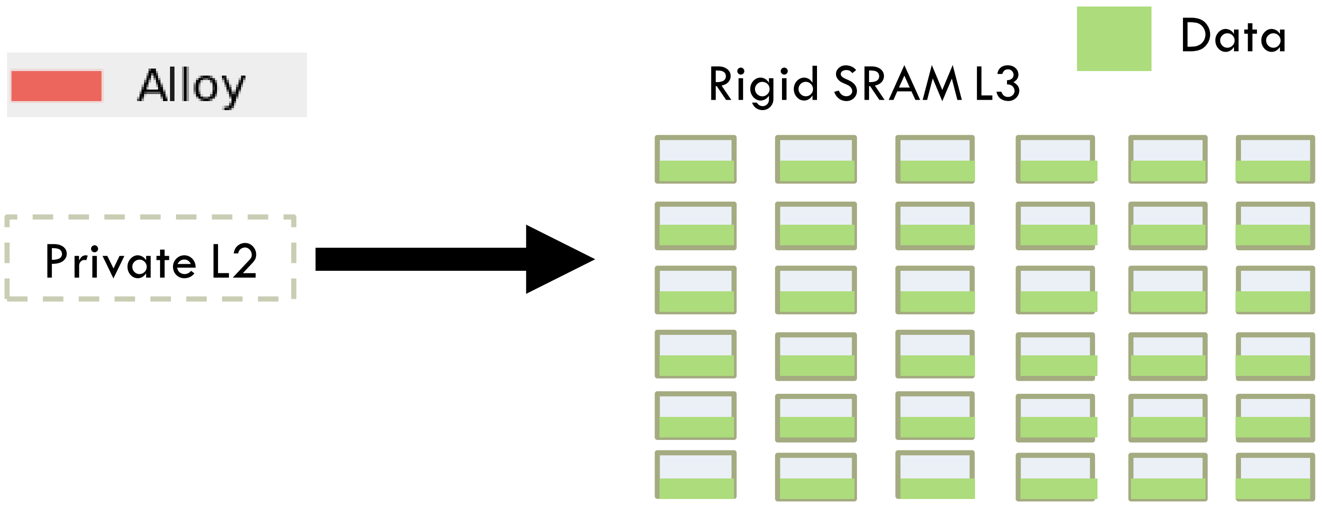
🟥 Alloy

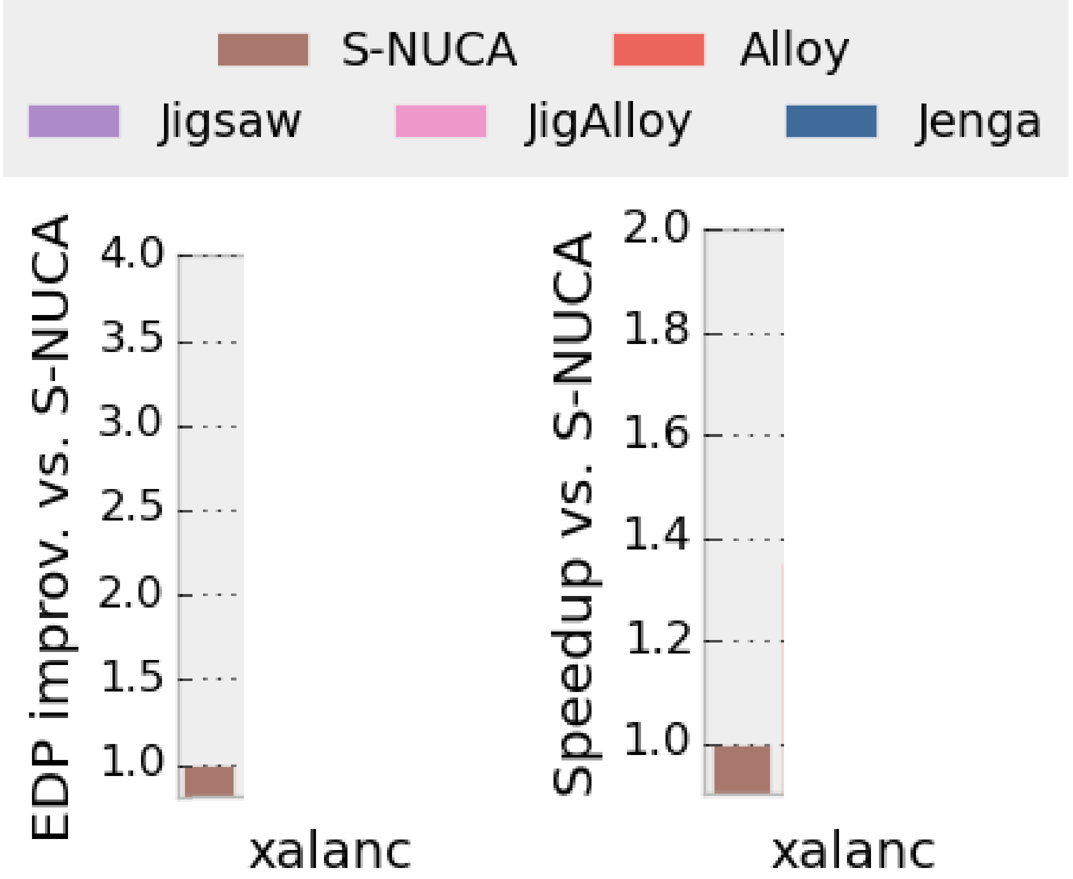
Private L2 →



Case study: 36 copies of xalanc

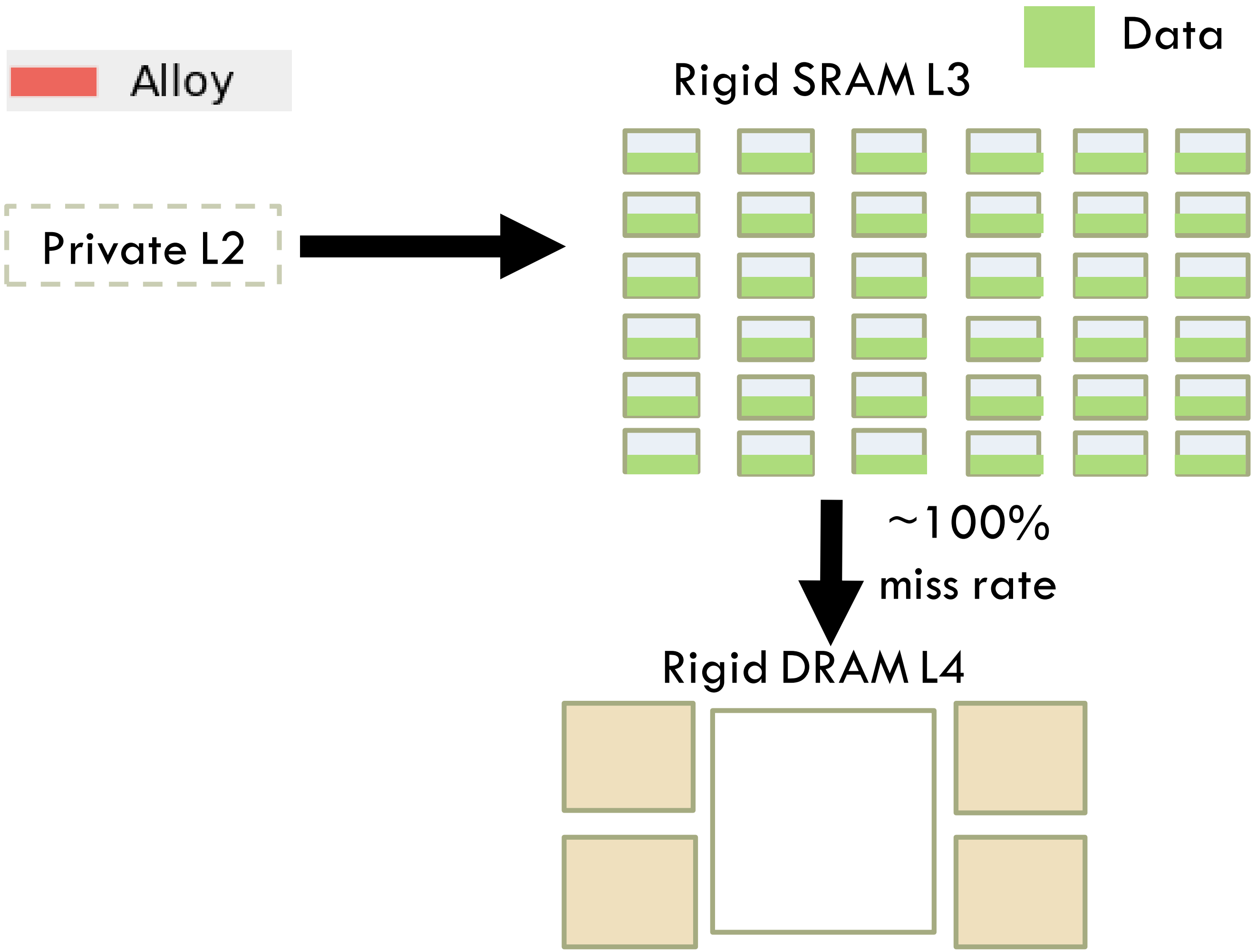
Working set: 6MB x 36 = 216 MB





Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB



🟥 Alloy

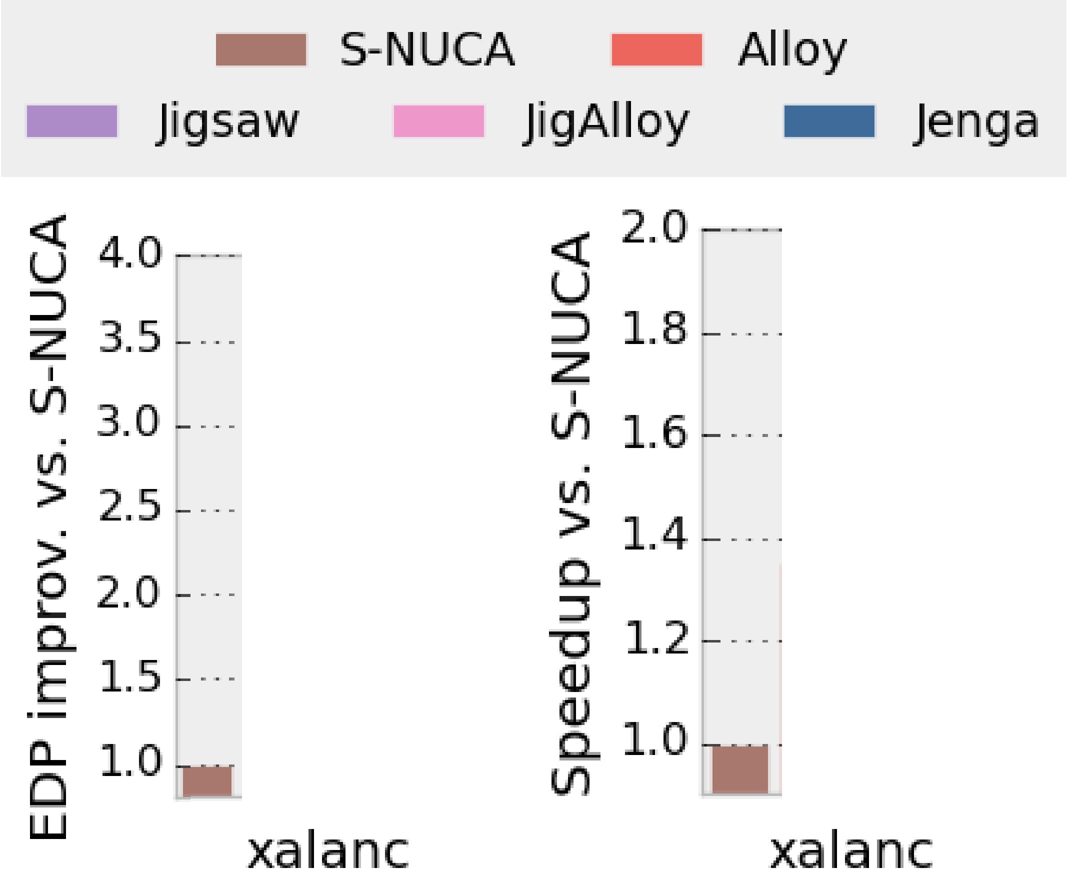
🟩 Data

Private L2

Rigid SRAM L3

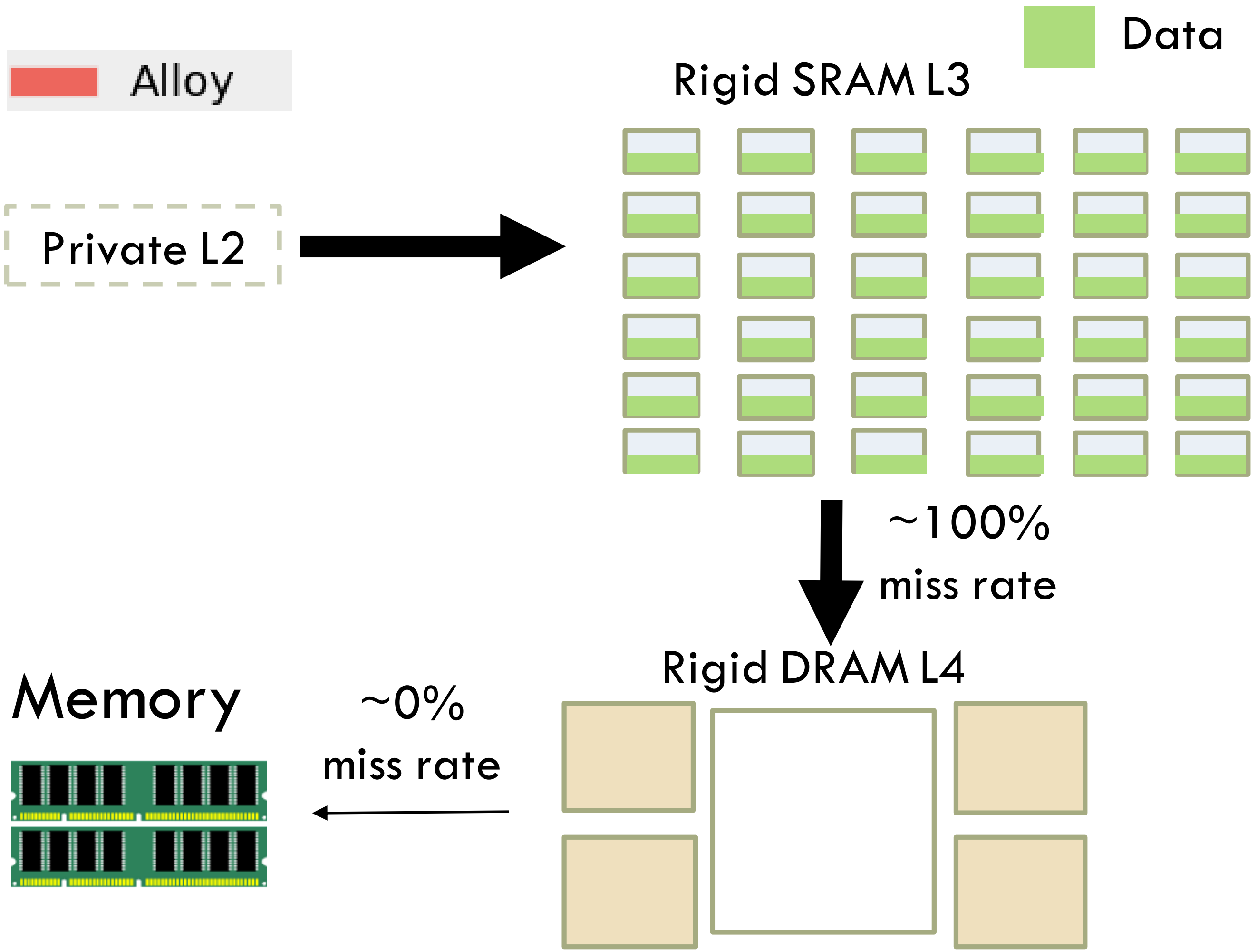
~100% miss rate

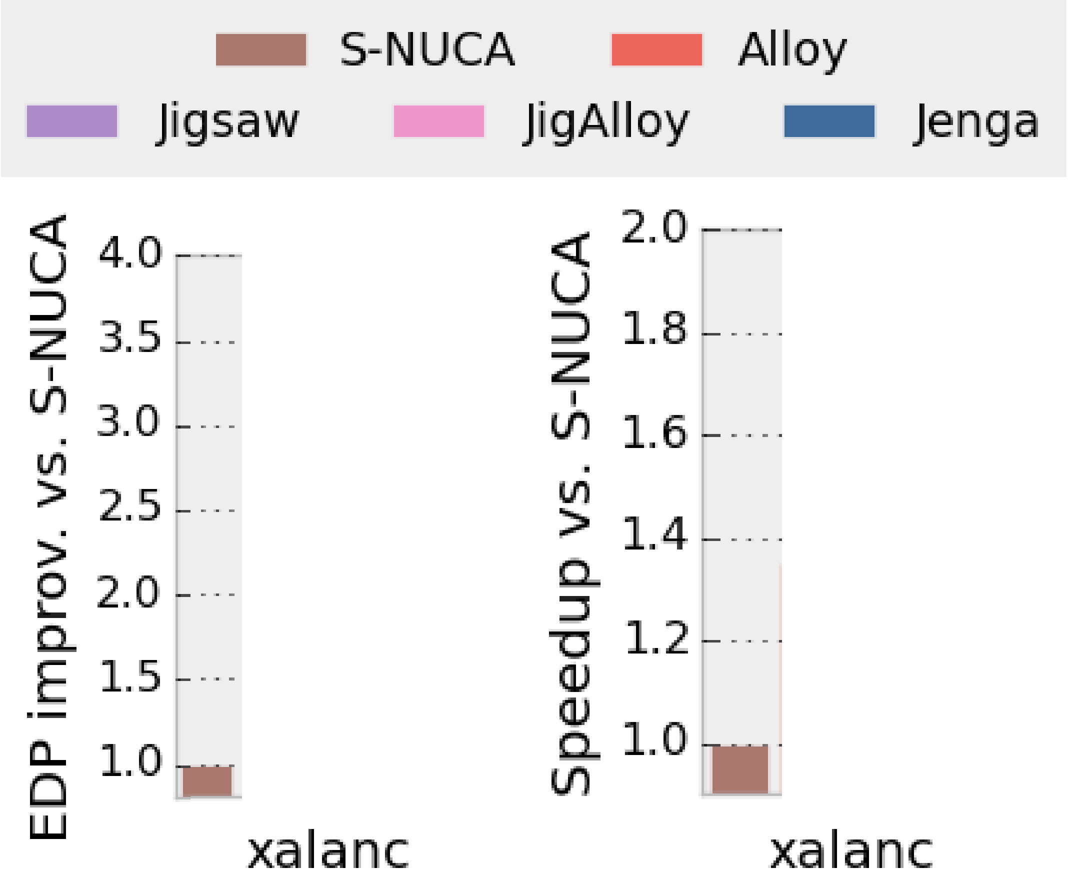
Rigid DRAM L4



Case study: 36 copies of xalanc

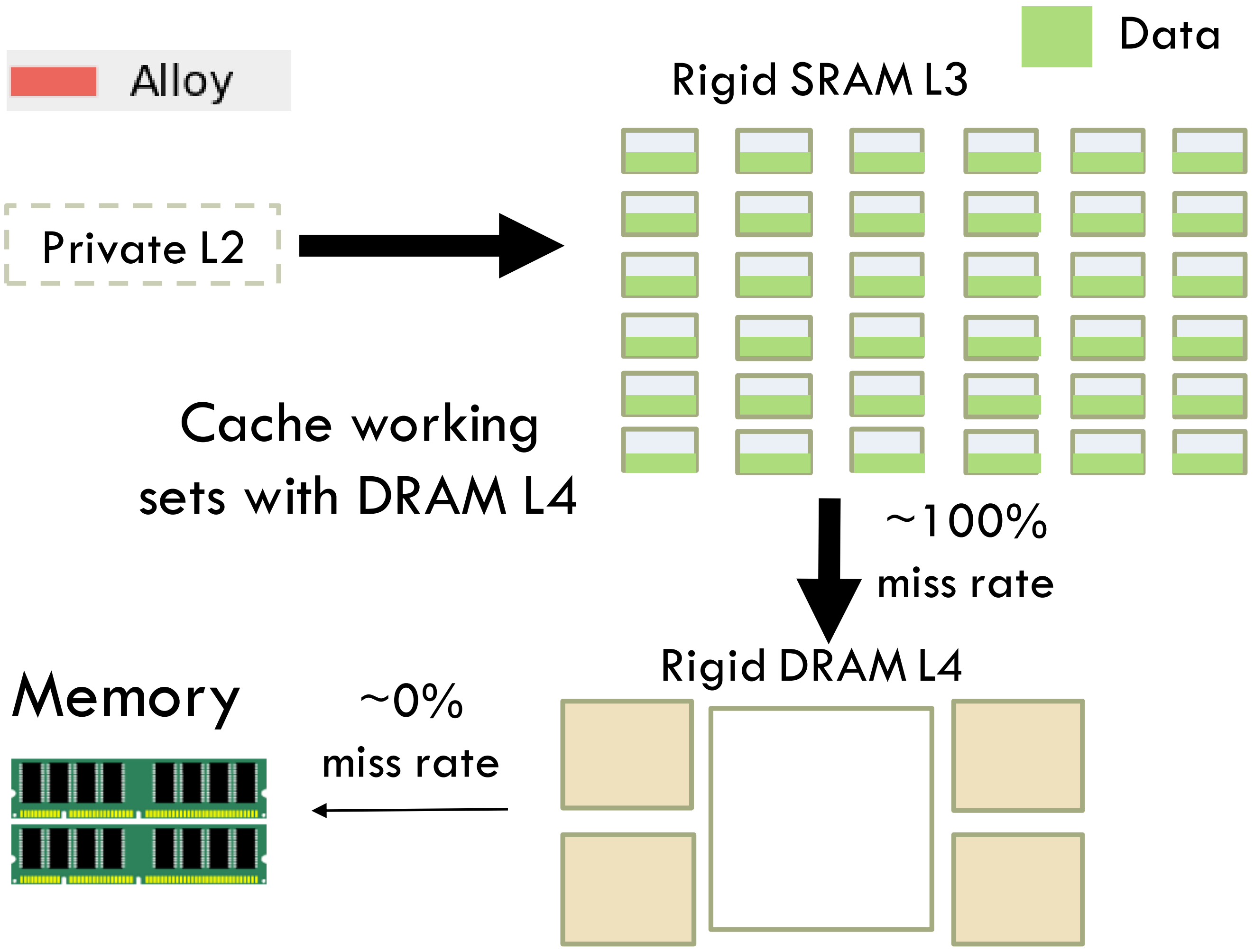
Working set: 6MB x 36 = 216 MB



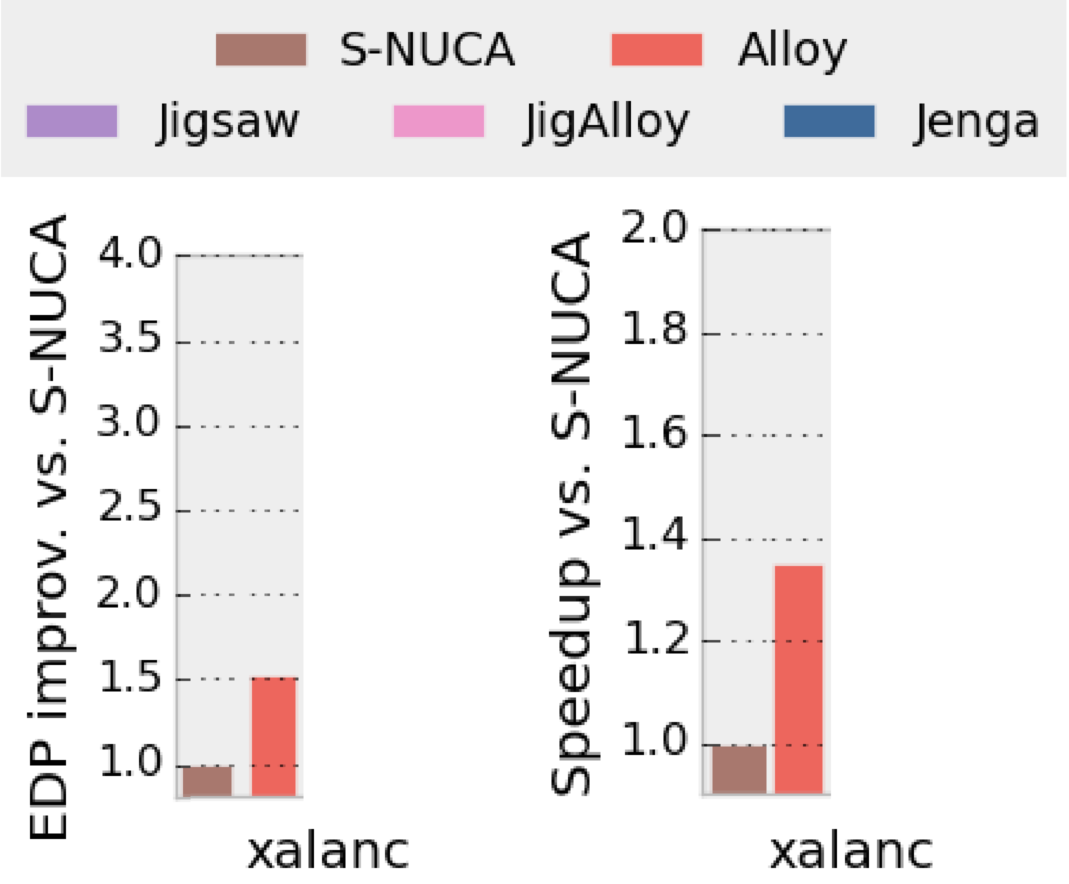


Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB

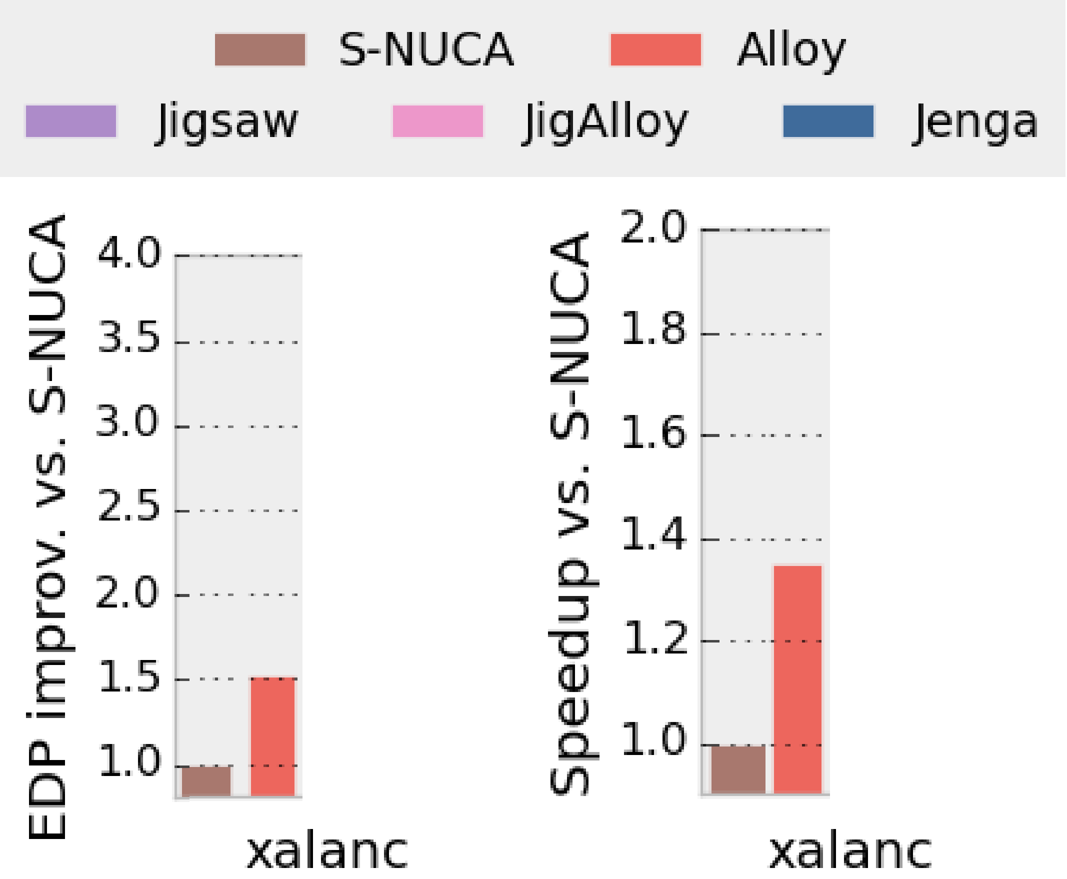


Cache working sets with DRAM L4



Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB

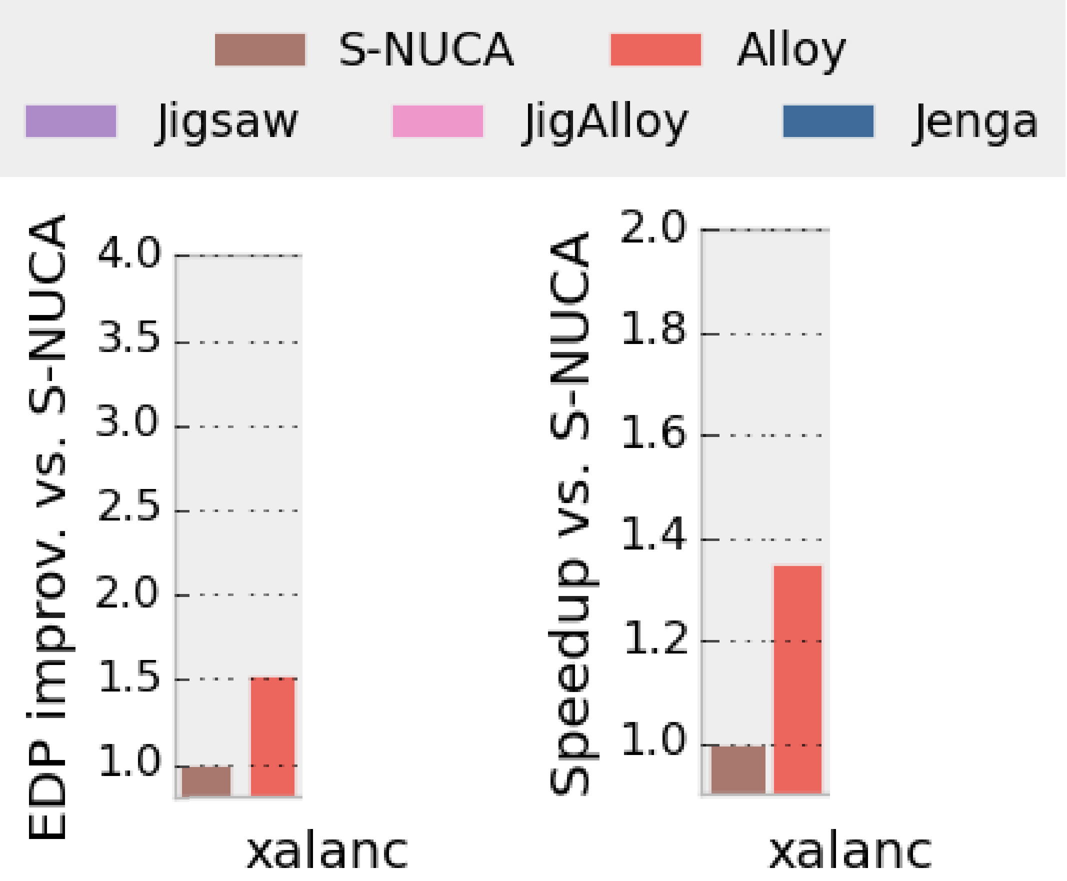


Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB

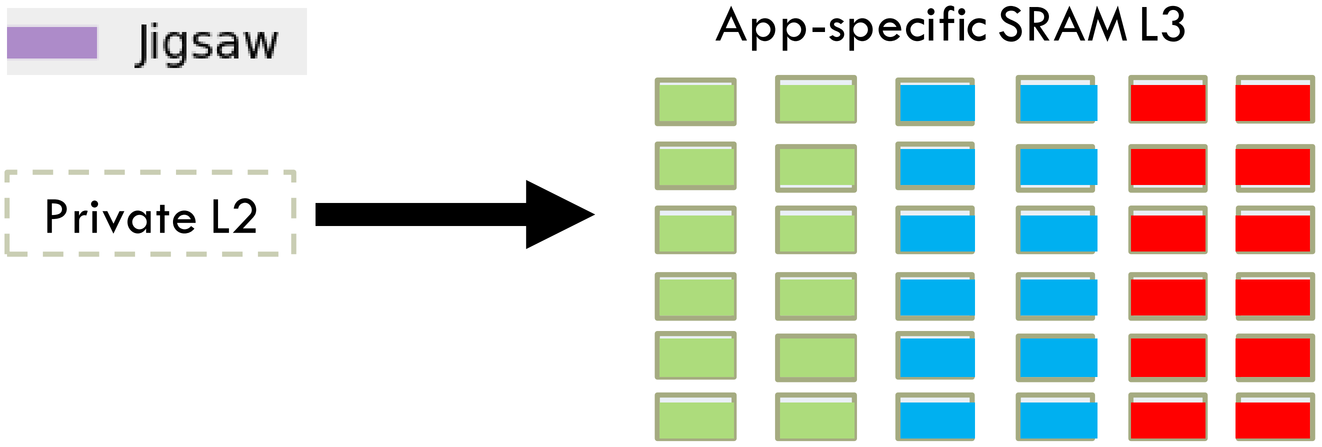
🟪 Jigsaw

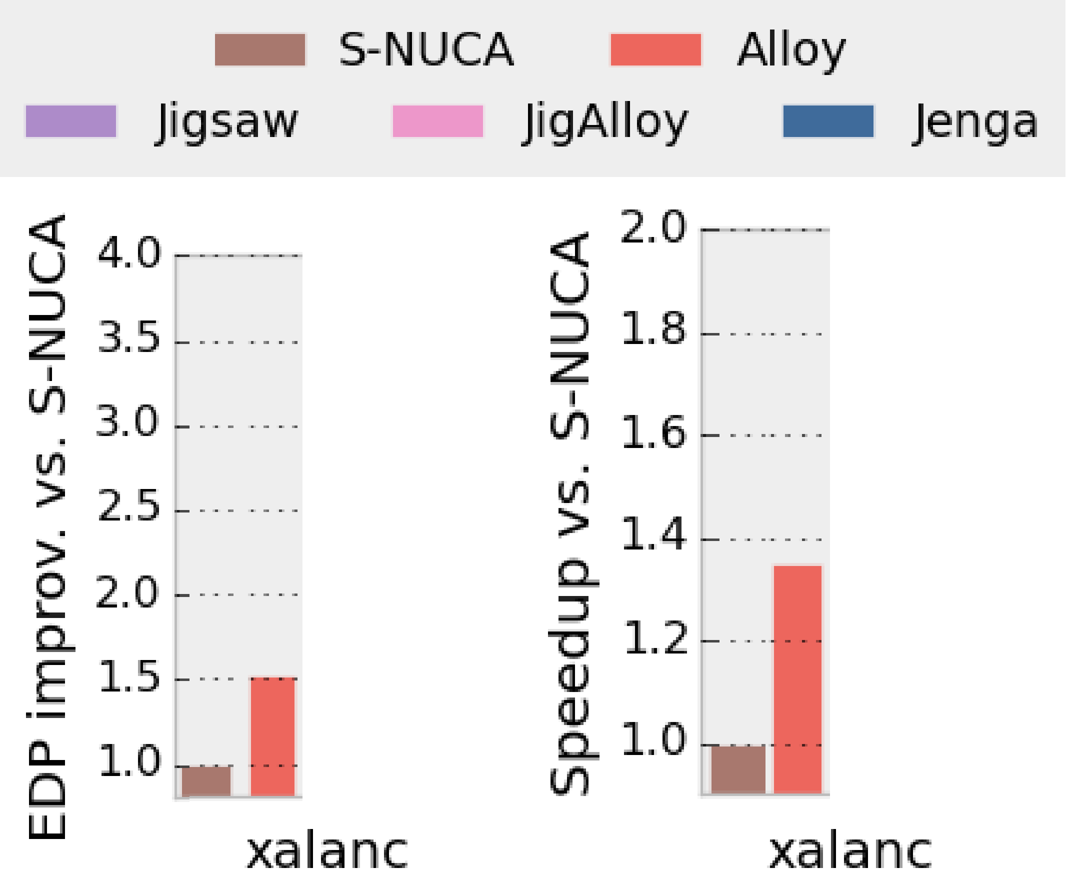
Private L2



Case study: 36 copies of xalanc

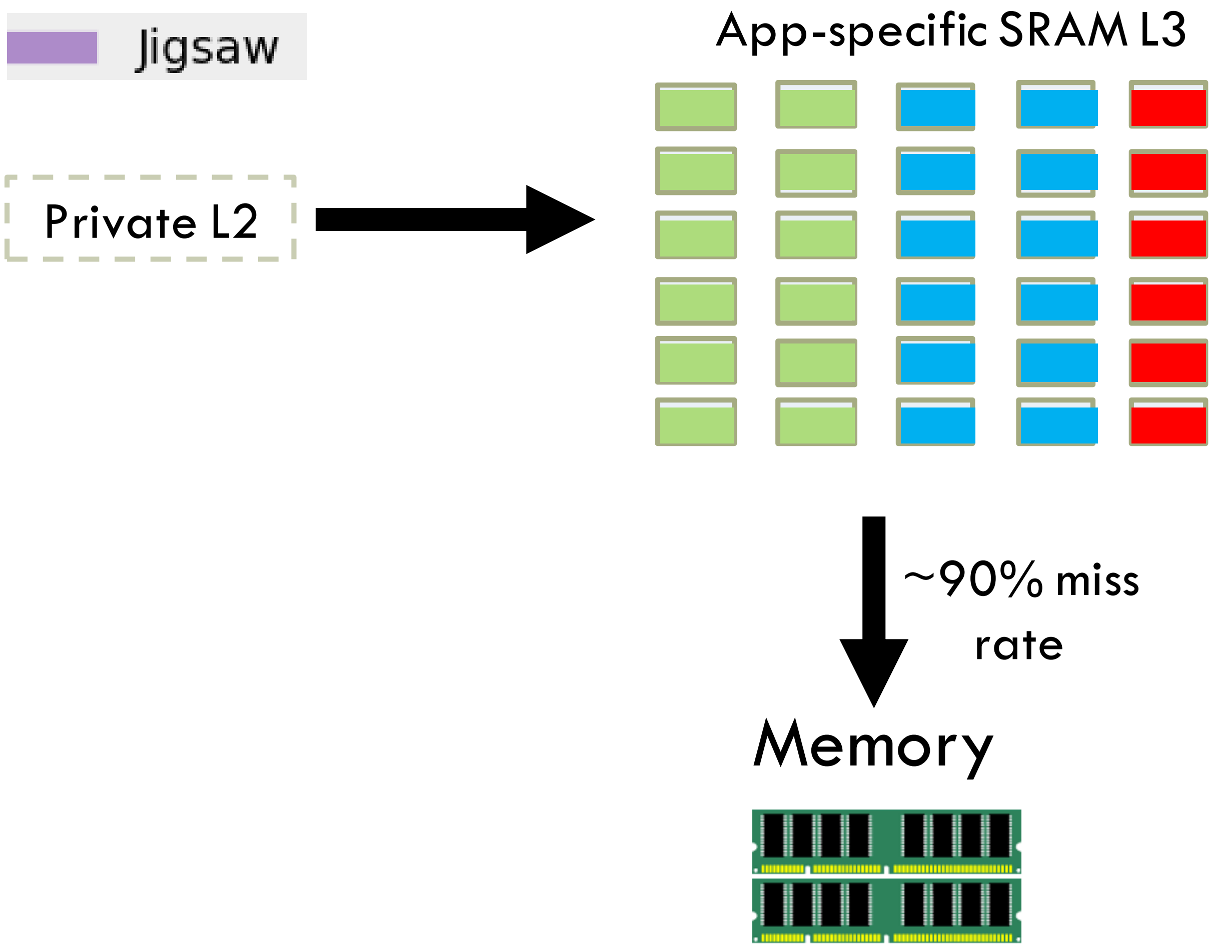
Working set: 6MB x 36 = 216 MB

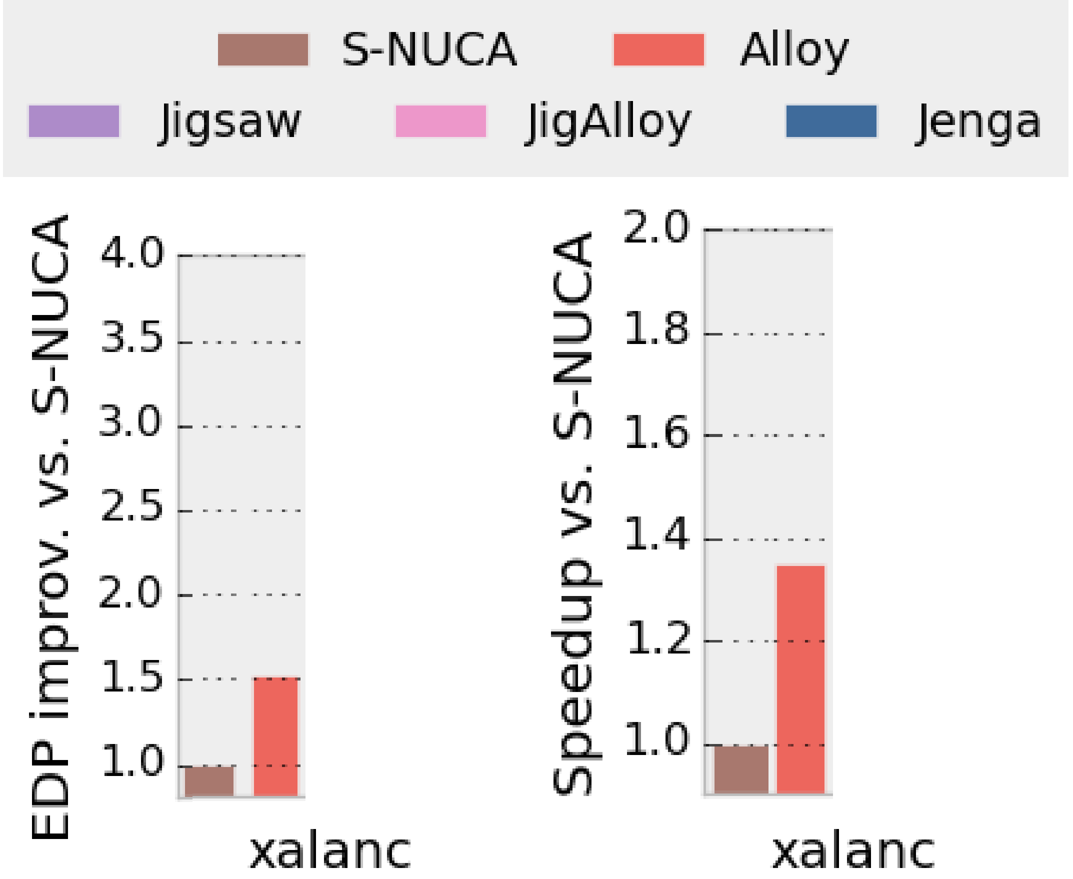




Case study: 36 copies of xalanc

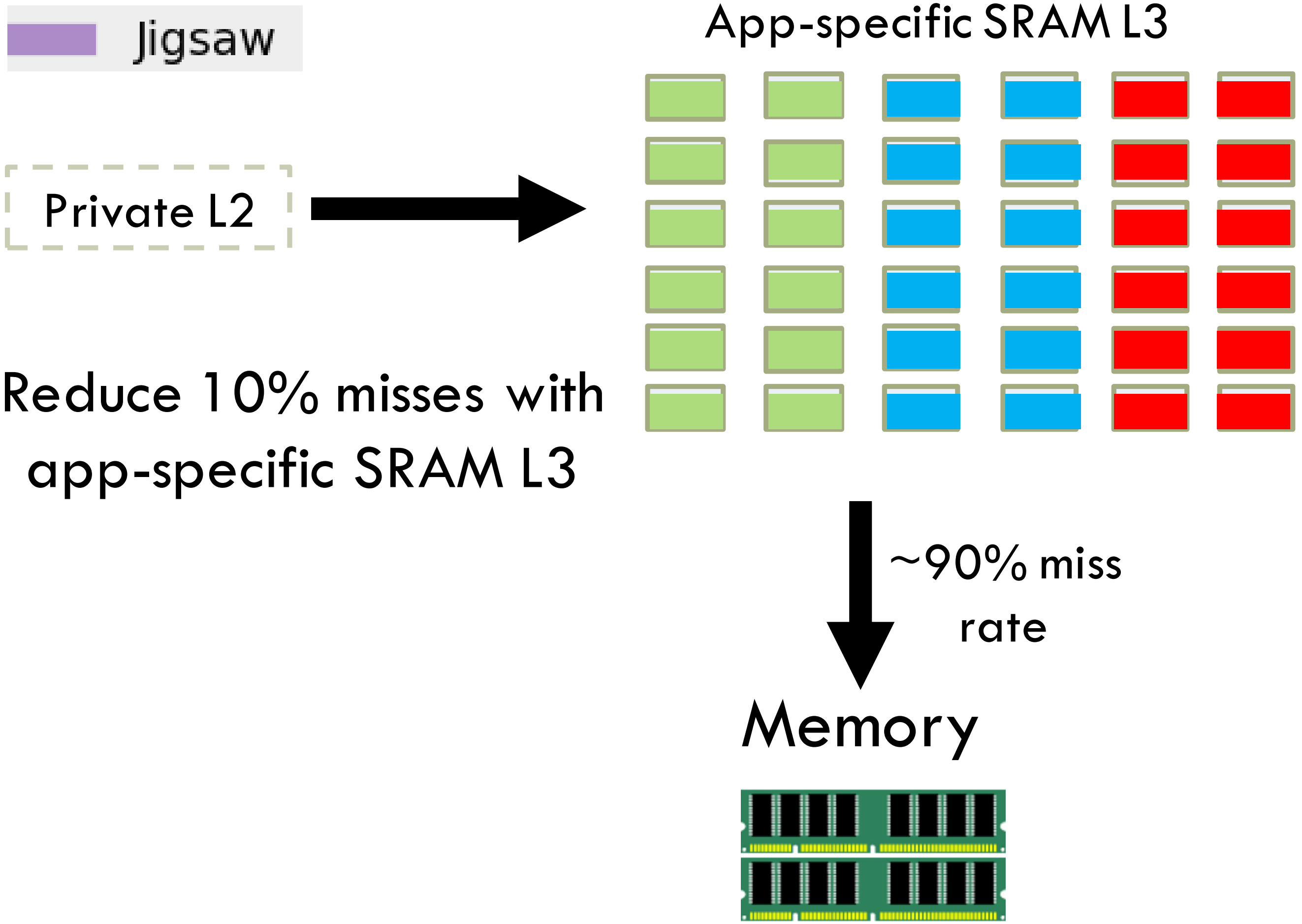
Working set: 6MB x 36 = 216 MB





Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB



Jigsaw

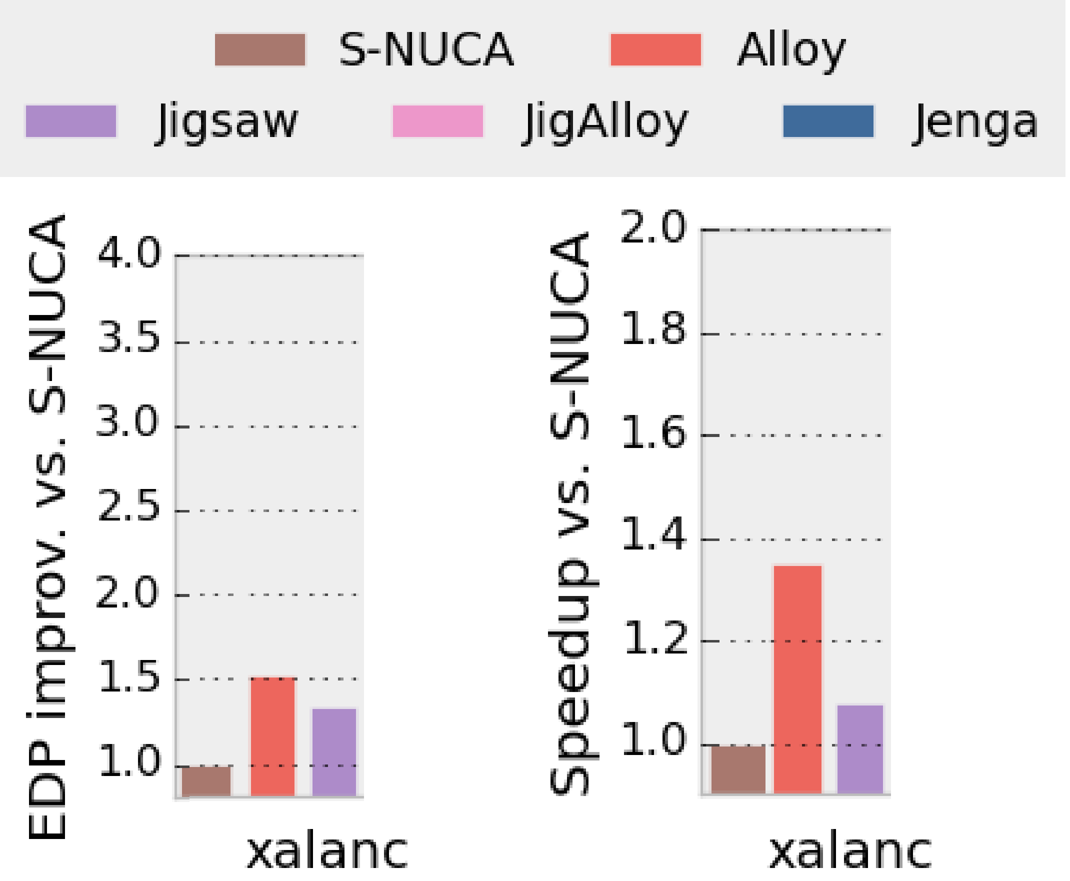
App-specific SRAM L3

Private L2

Reduce 10% misses with app-specific SRAM L3

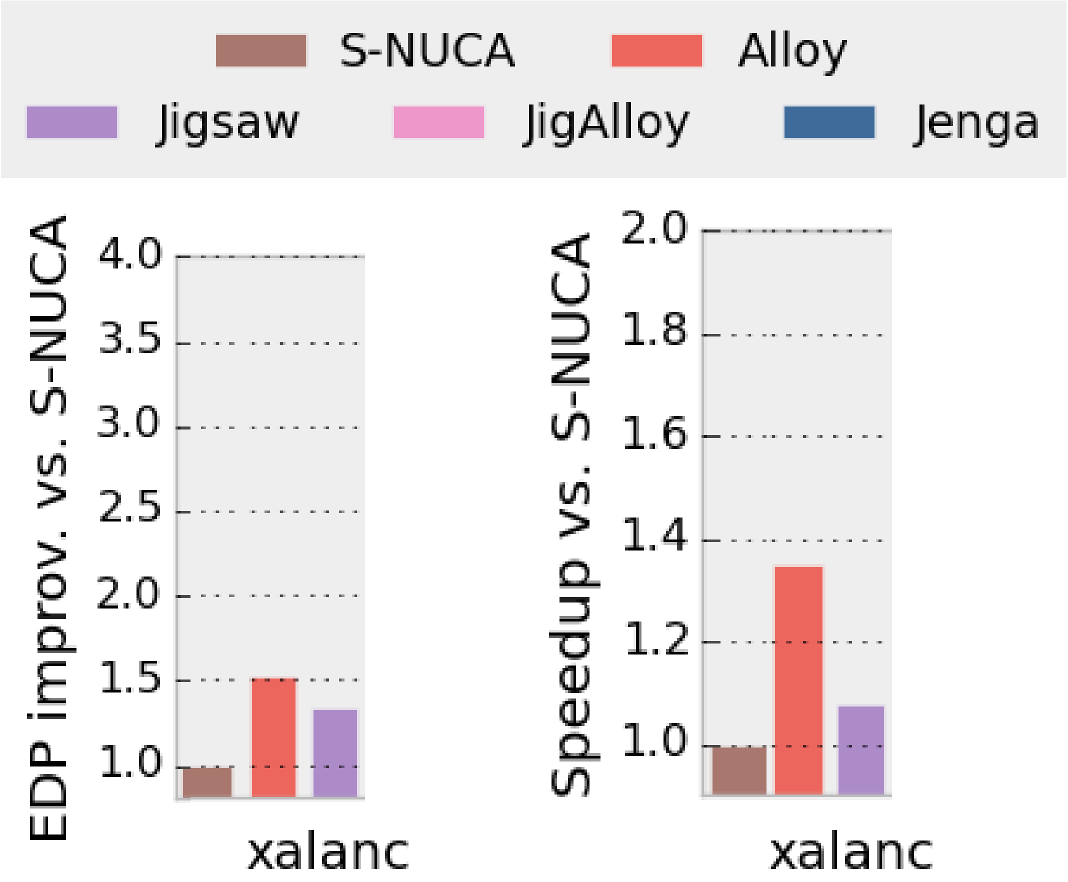
~90% miss rate

Memory



Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB

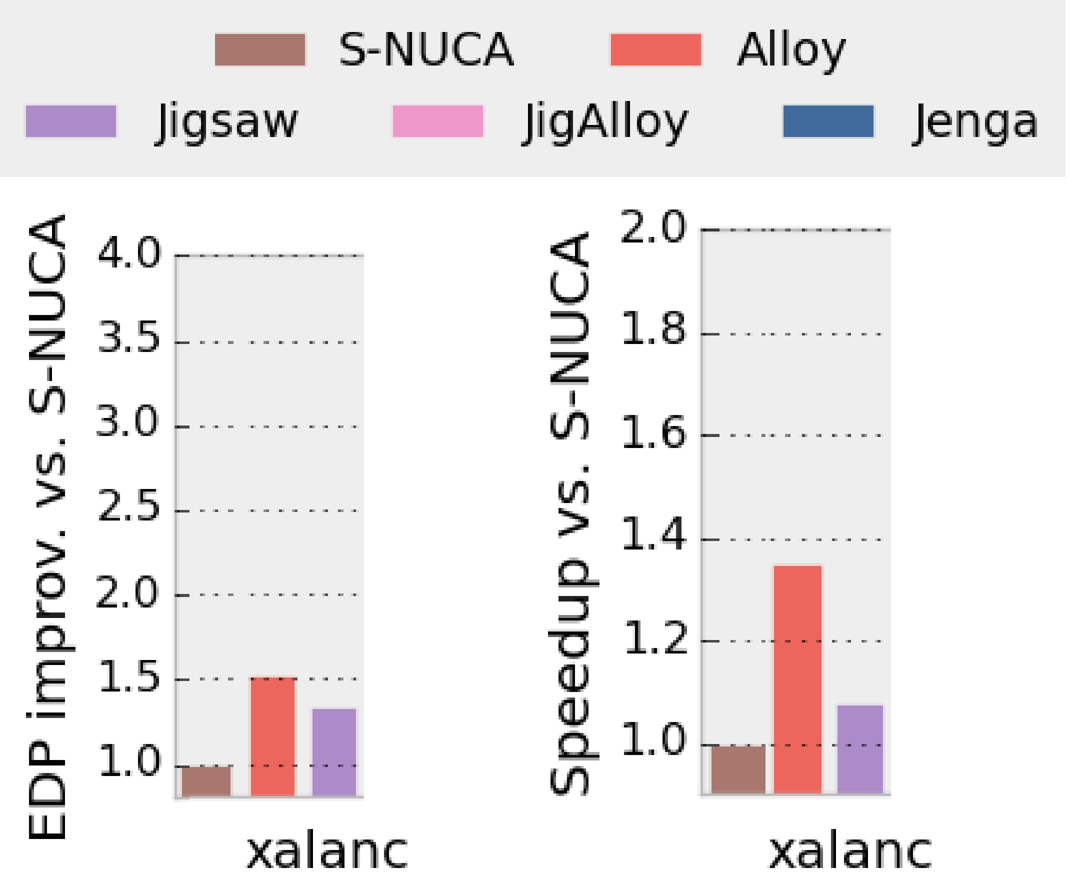


Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB

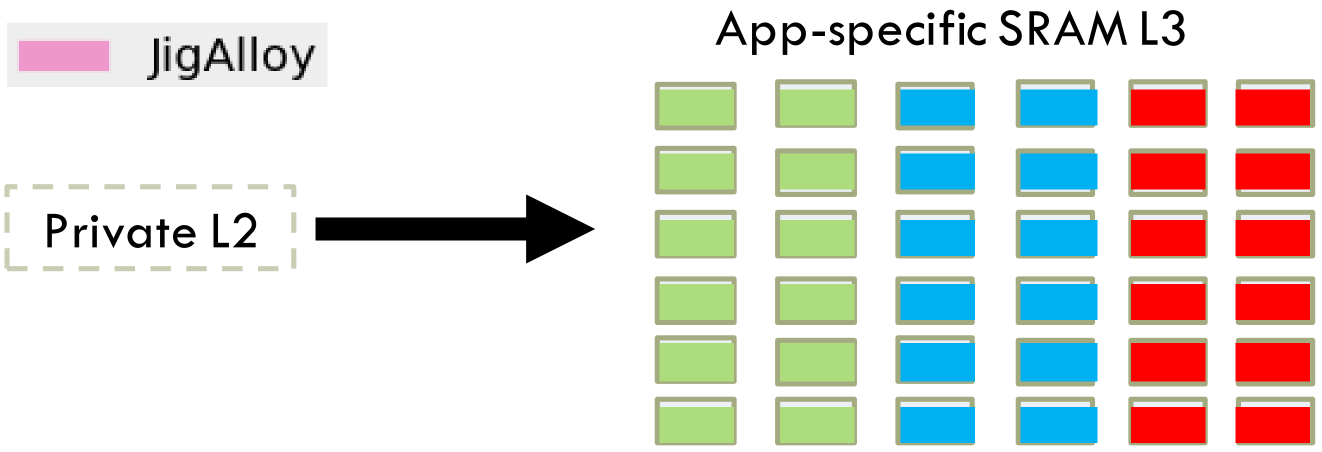
🟪 JigAlloy

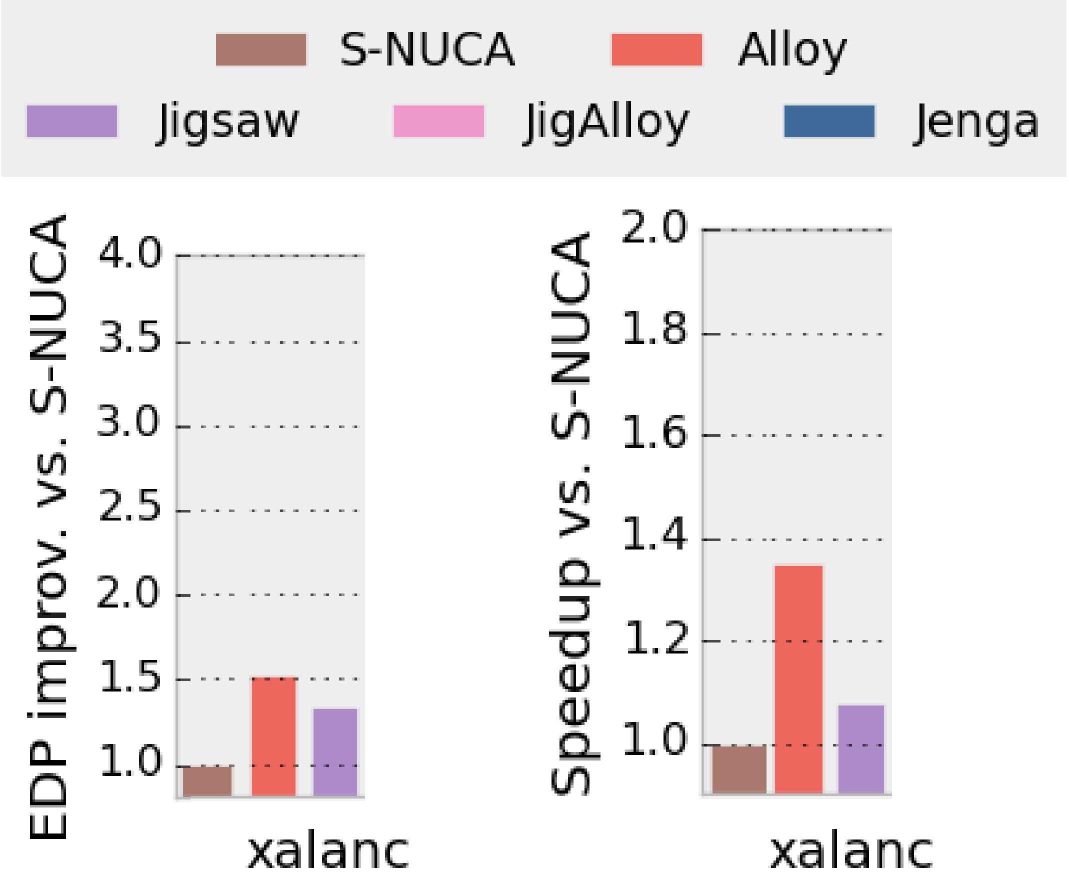
Private L2 →



Case study: 36 copies of xalanc

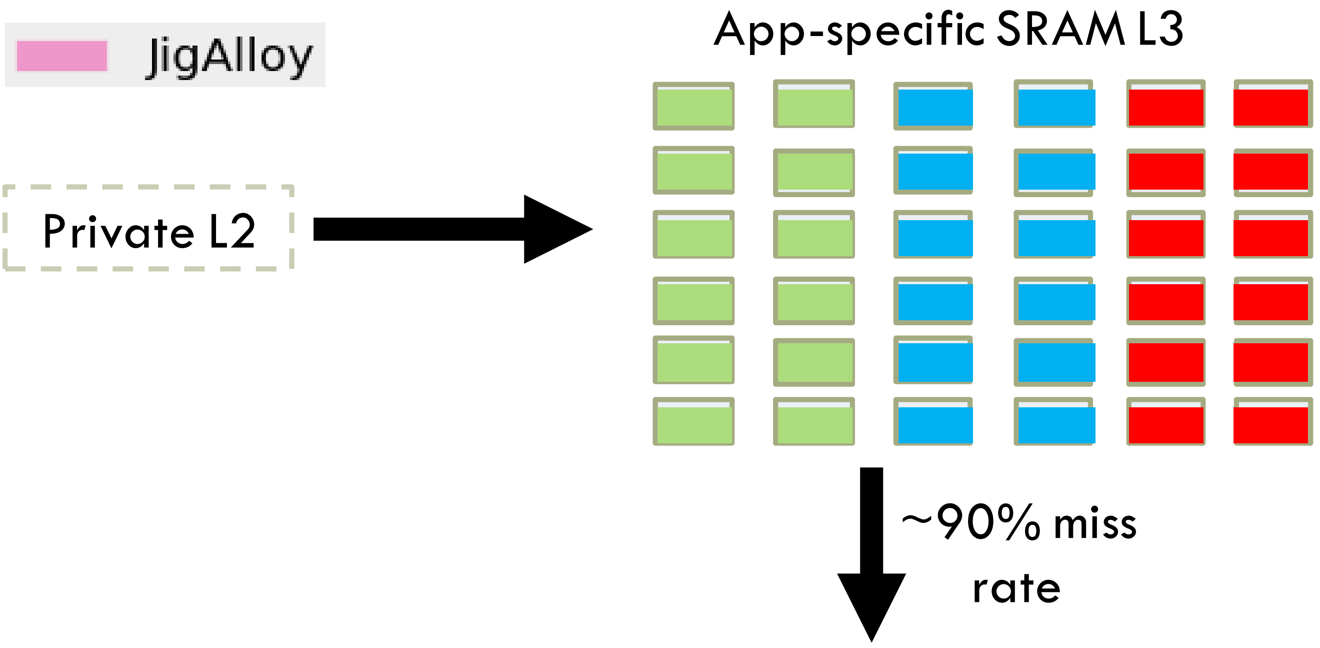
Working set: 6MB x 36 = 216 MB





Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB

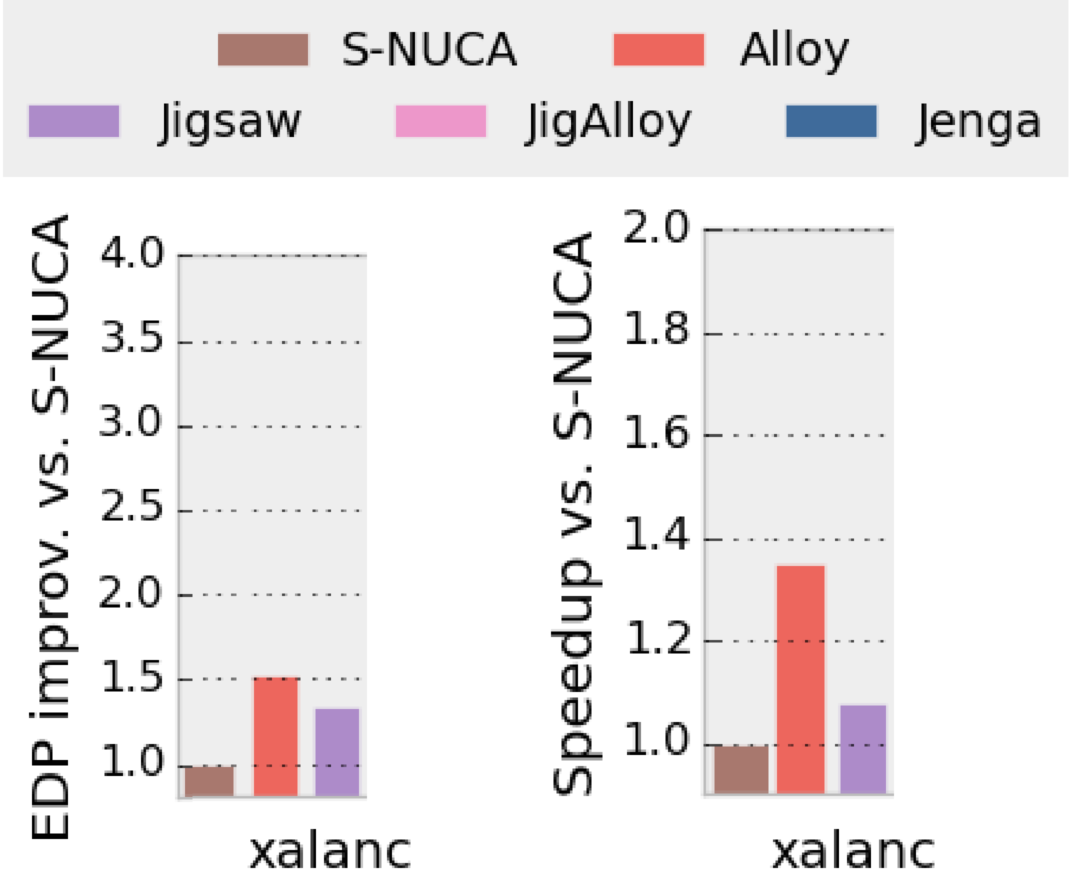


JigAlloy

App-specific SRAM L3

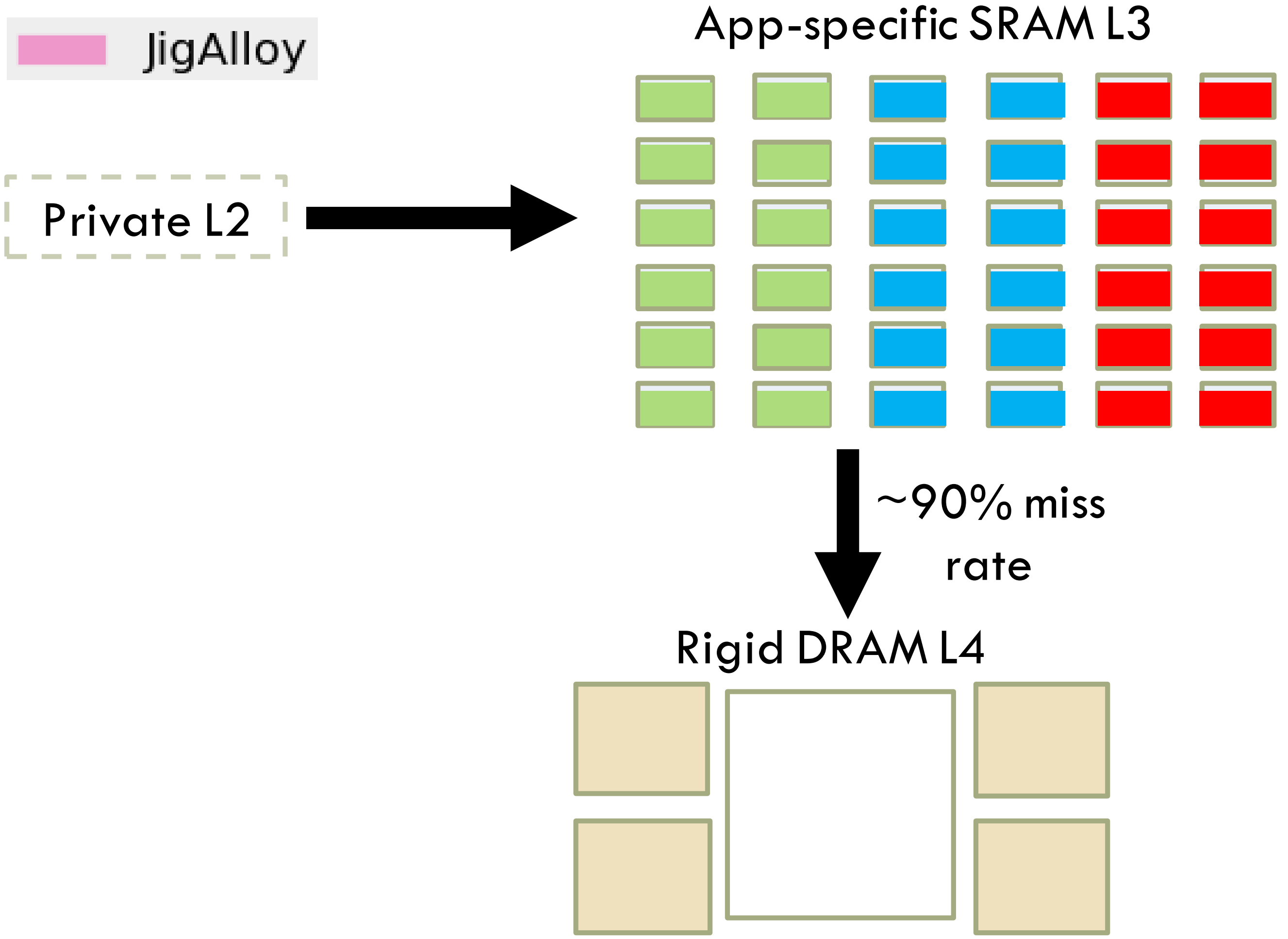
Private L2

~90% miss rate



Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB



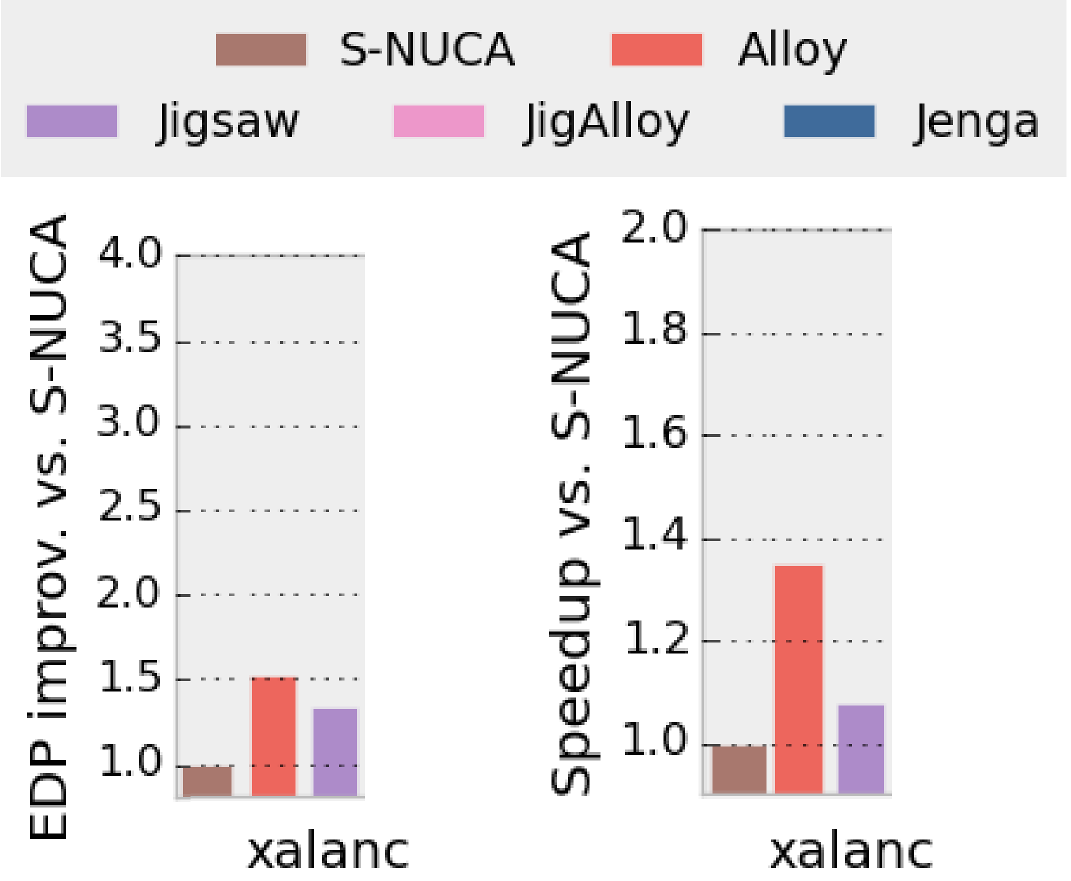
JigAlloy

Private L2

App-specific SRAM L3

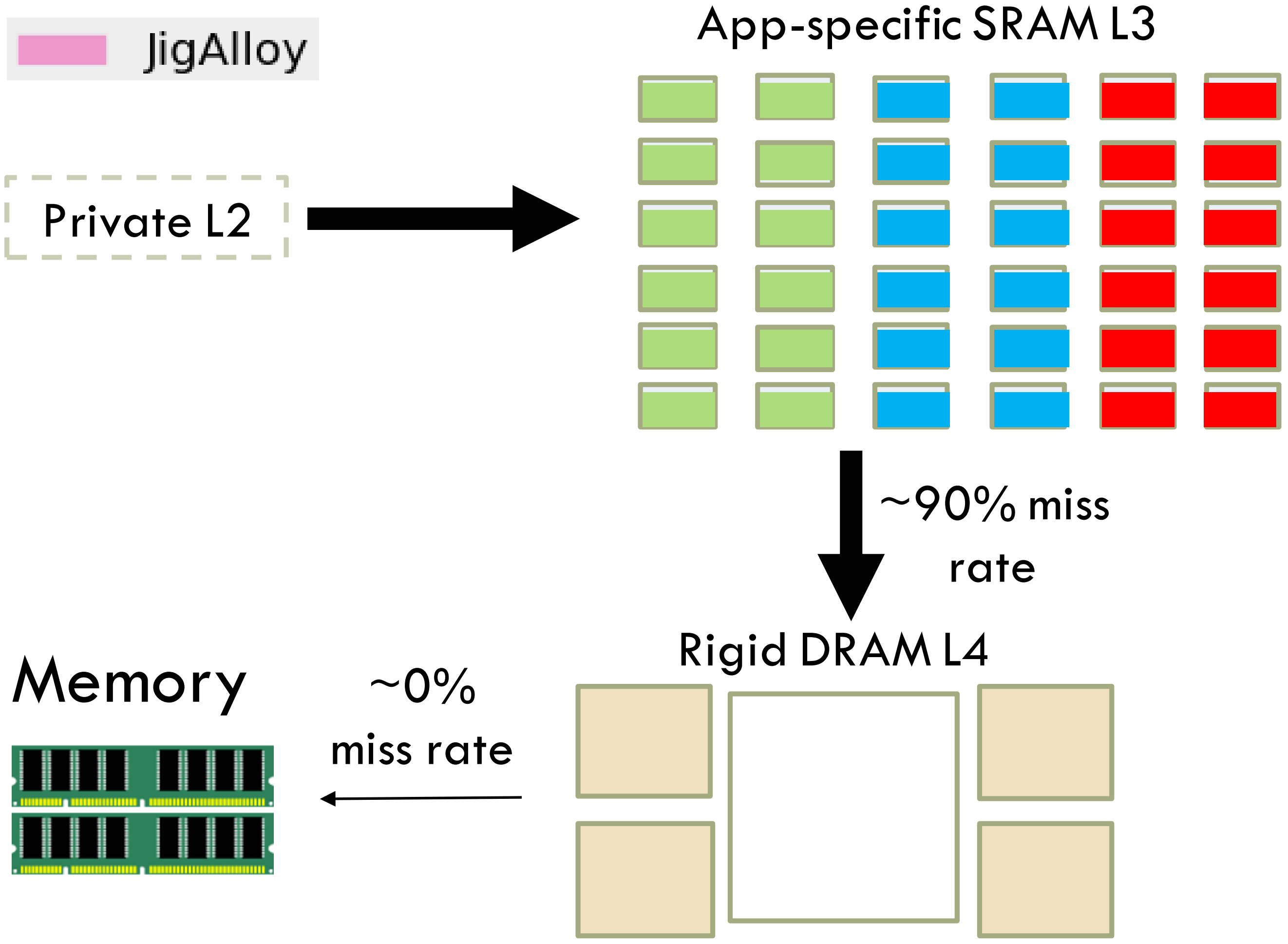
~90% miss rate

Rigid DRAM L4



Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB



JigAlloy

Private L2

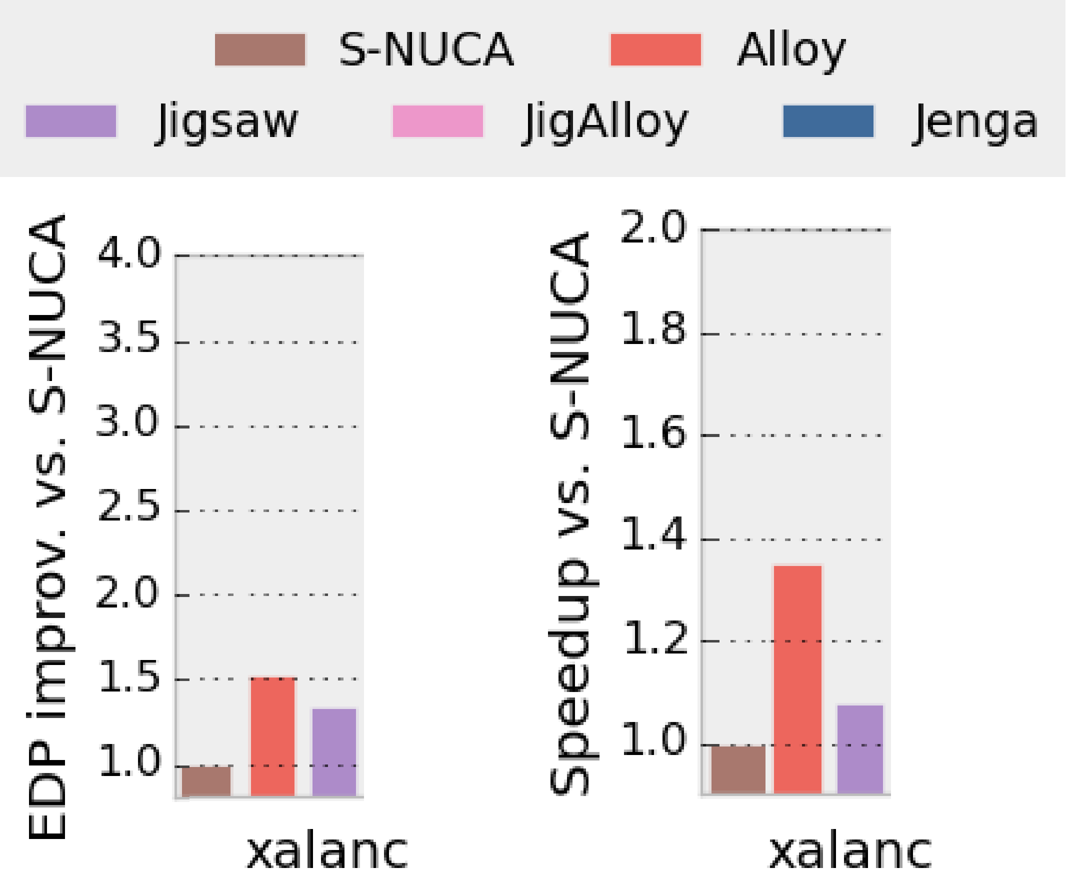
App-specific SRAM L3

~90% miss rate

Rigid DRAM L4

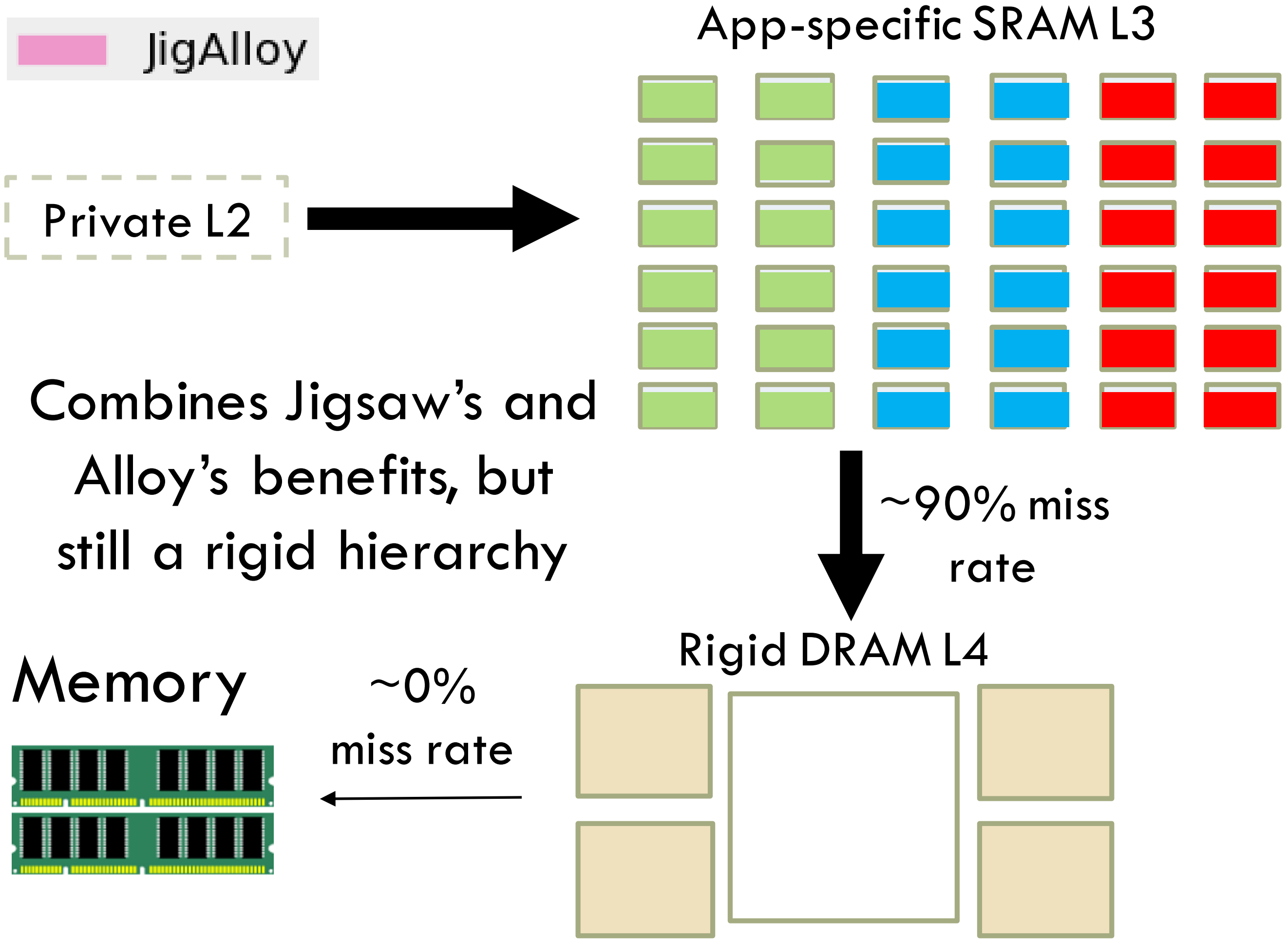
~0% miss rate

Memory

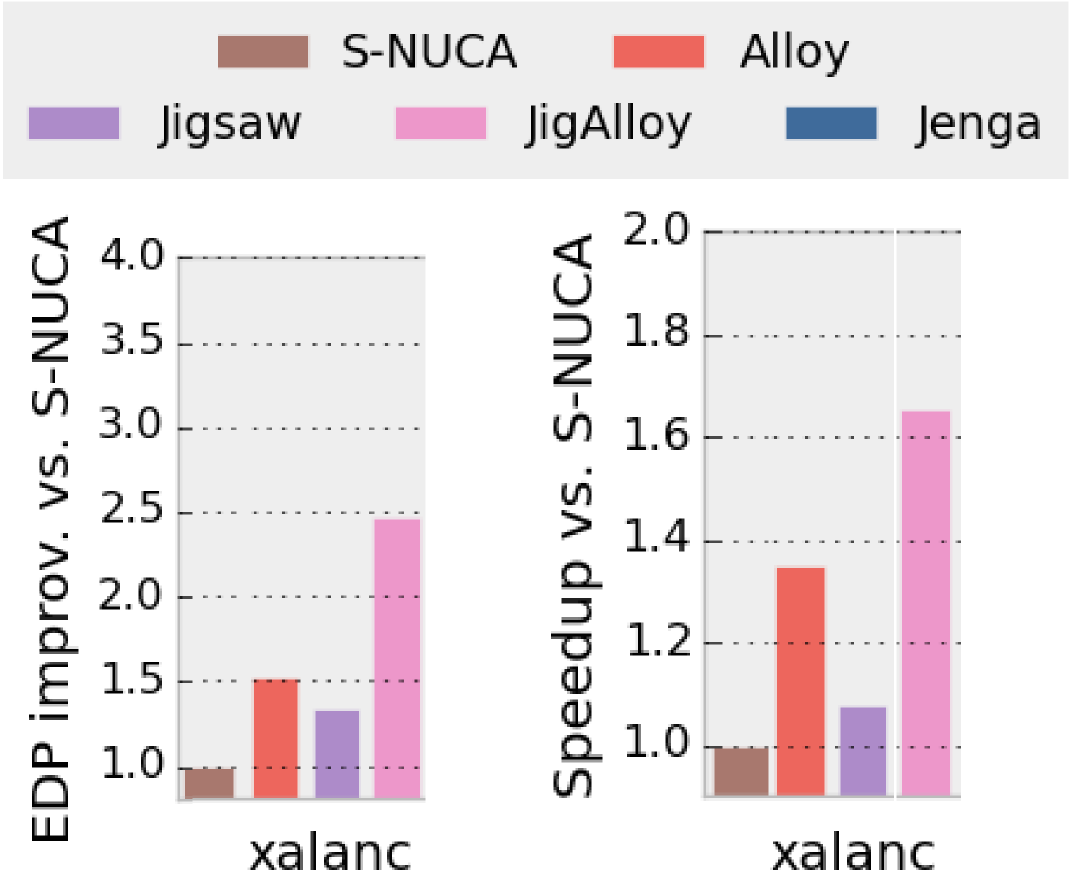


Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB

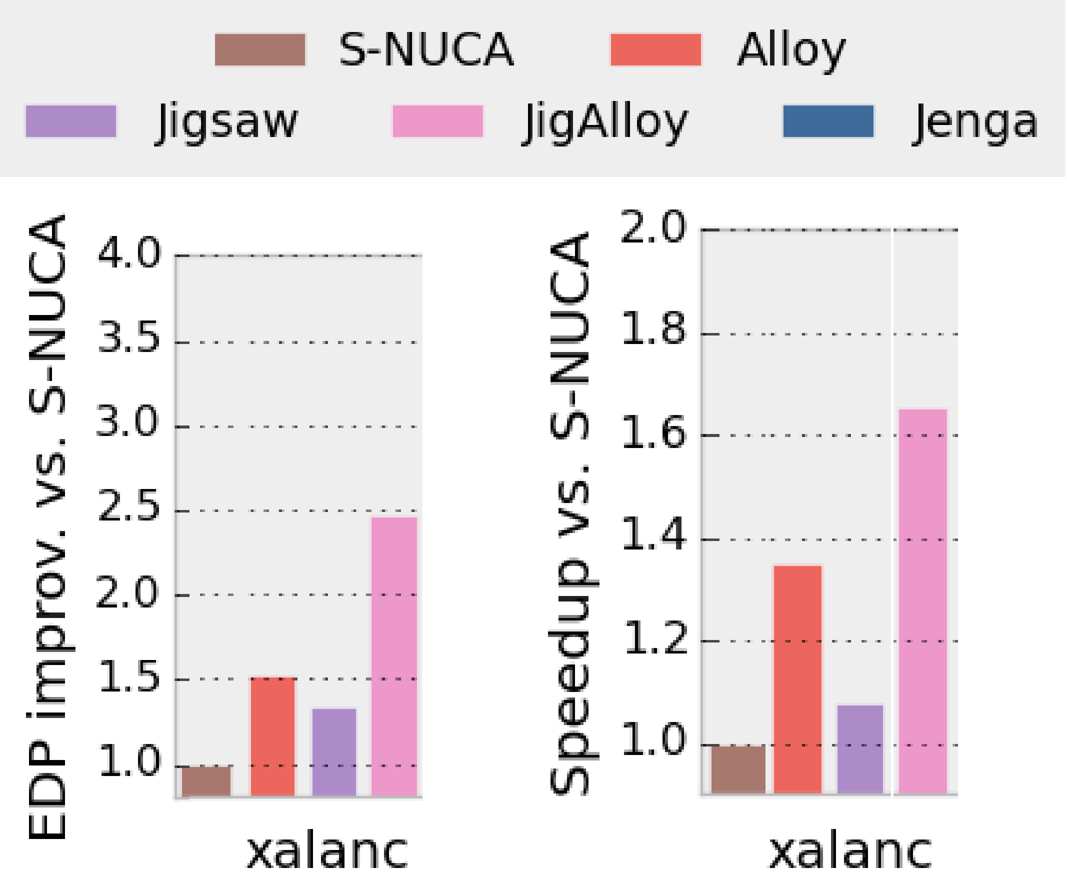


Combines Jigsaw's and Alloy's benefits, but still a rigid hierarchy



Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB

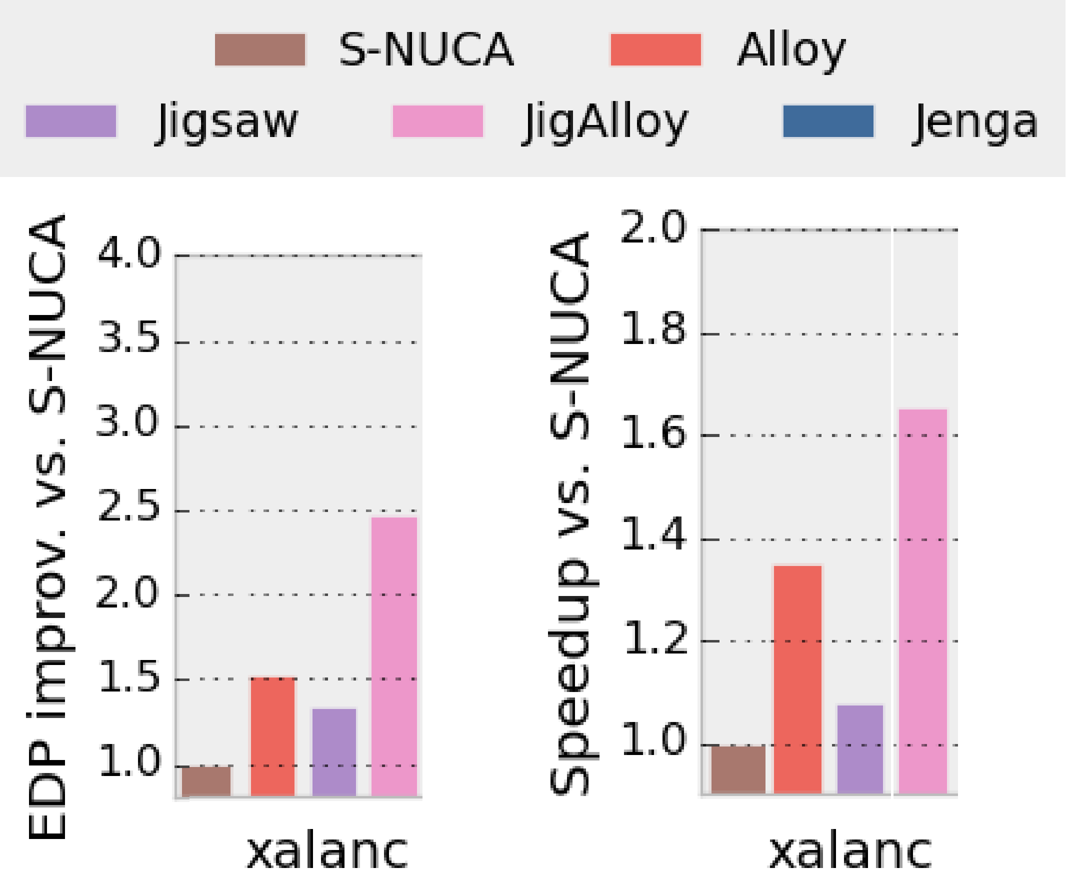


Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB

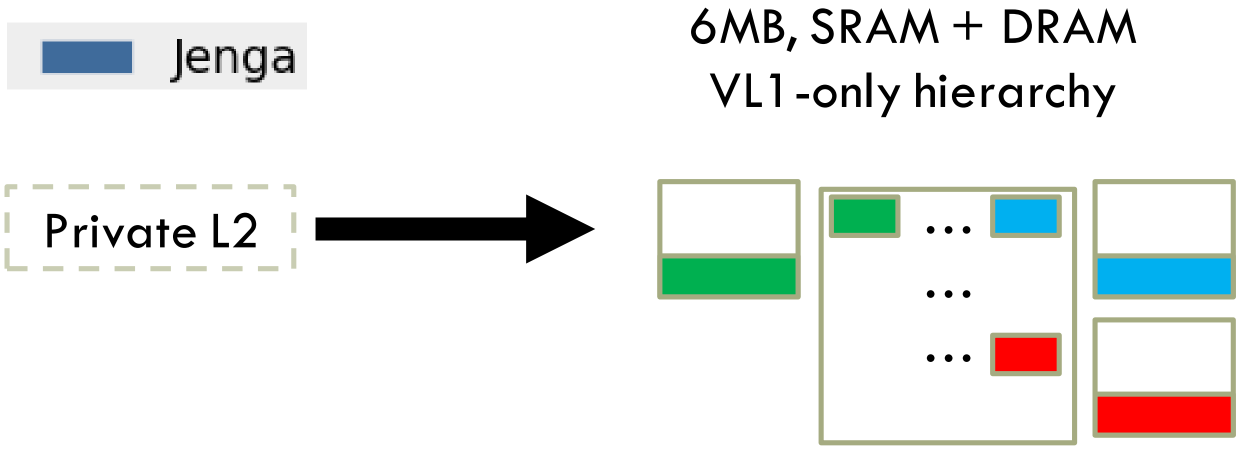
🟦 Jenga

Private L2



Case study: 36 copies of xalanc

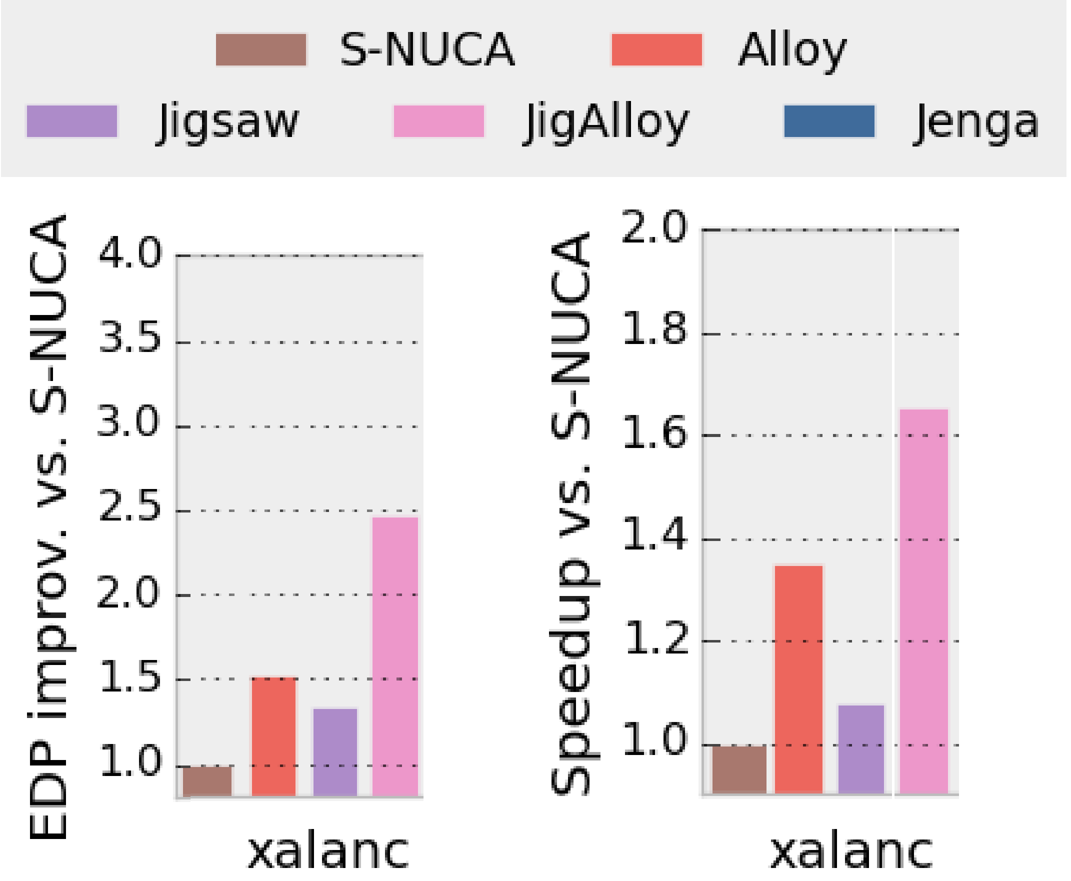
Working set: 6MB x 36 = 216 MB



Jenga

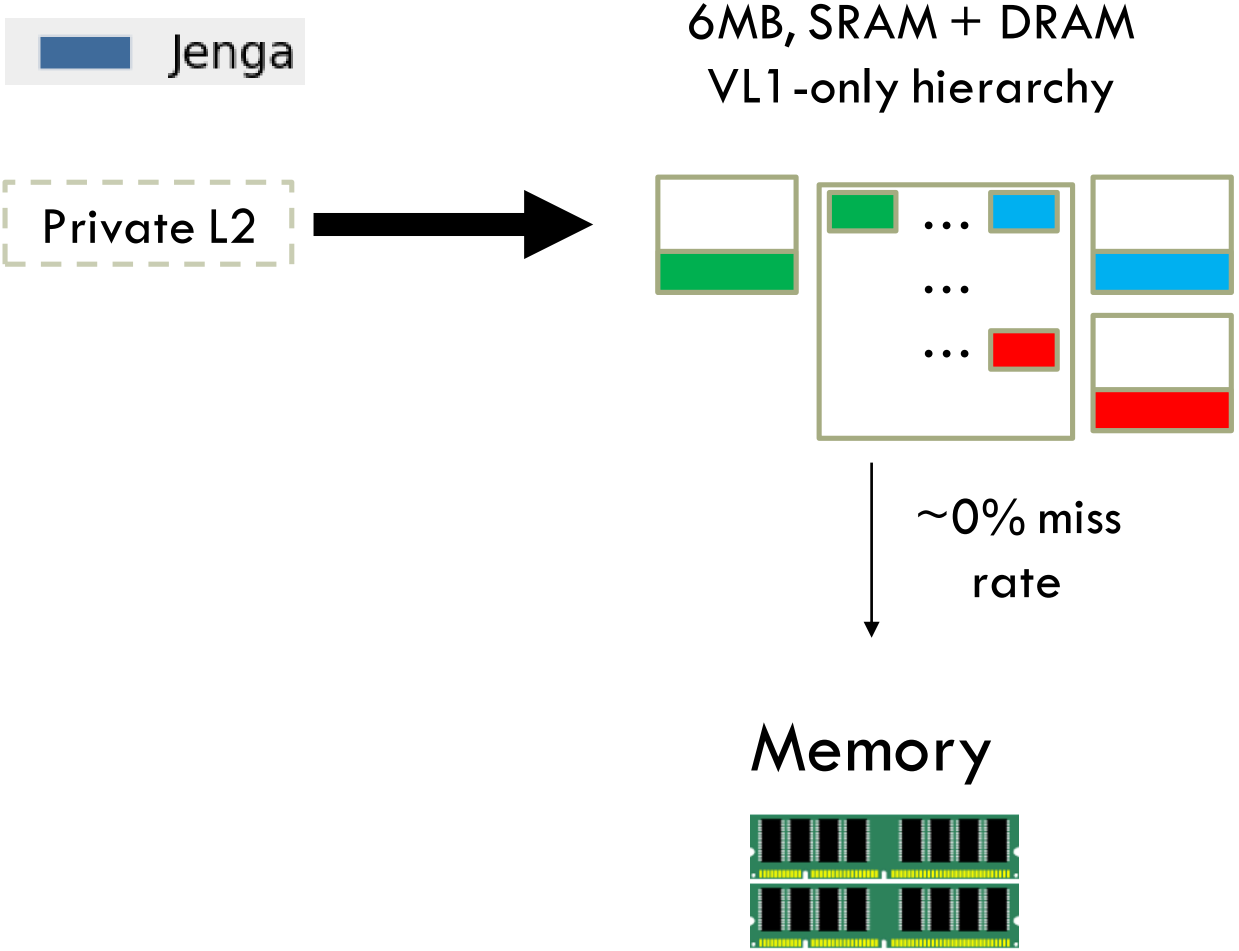
6MB, SRAM + DRAM
VL1-only hierarchy

Private L2



Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB



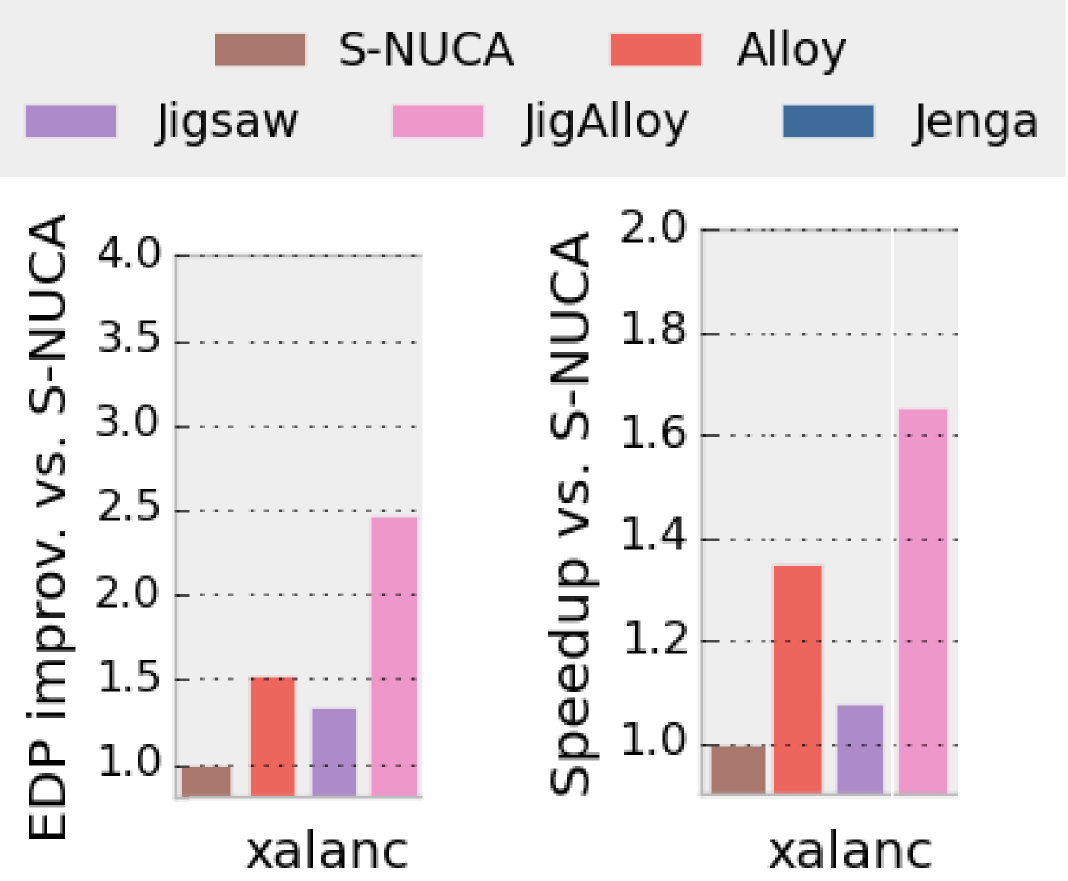
Jenga

Private L2

6MB, SRAM + DRAM
VL1-only hierarchy

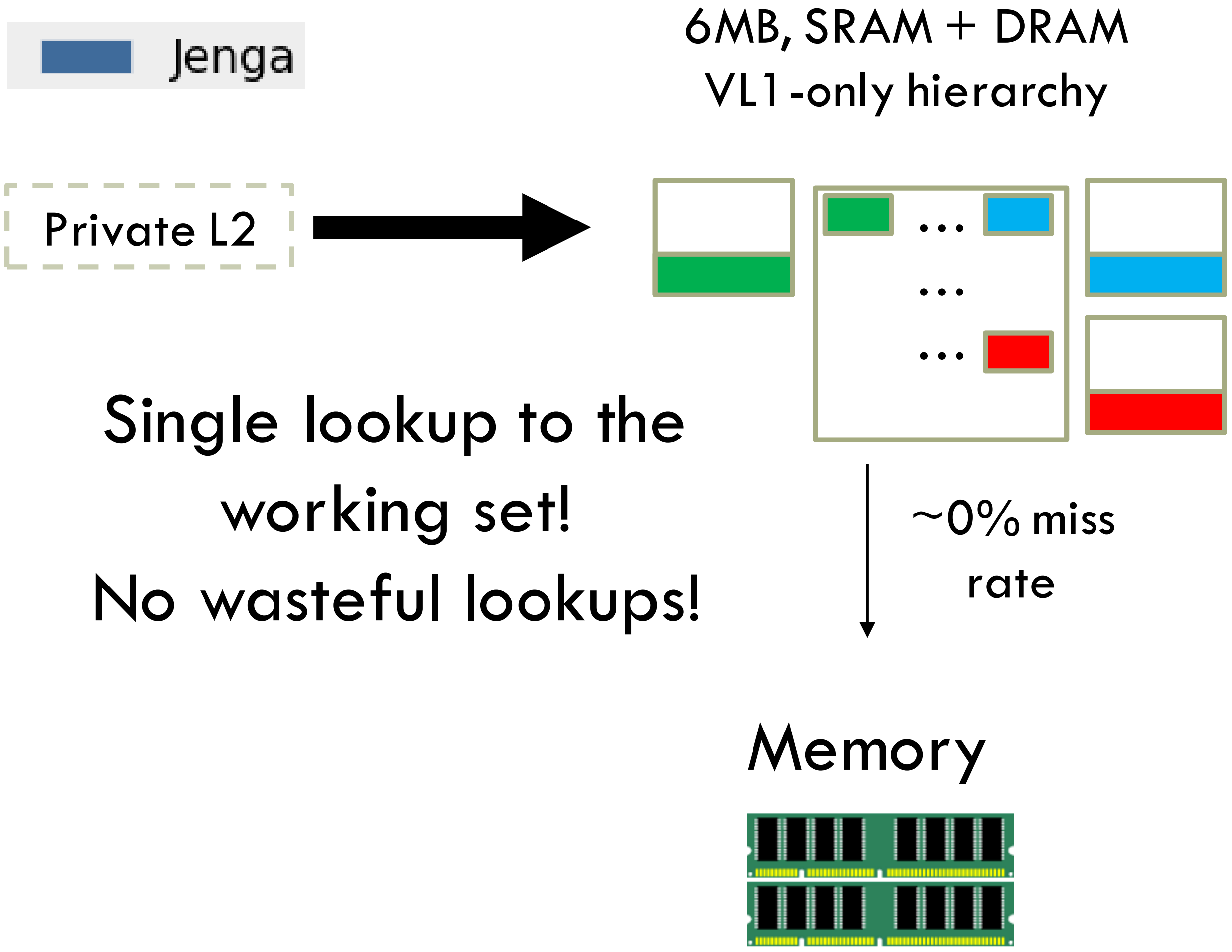
~0% miss rate

Memory



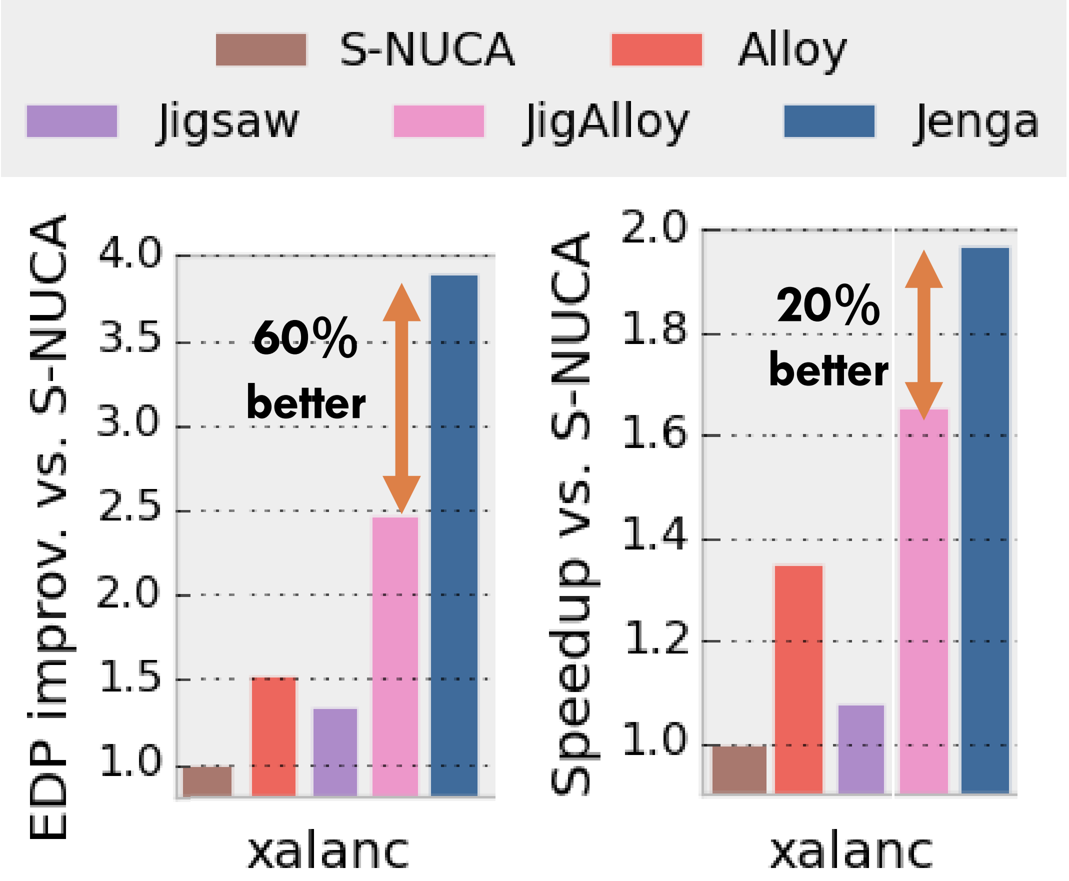
Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB



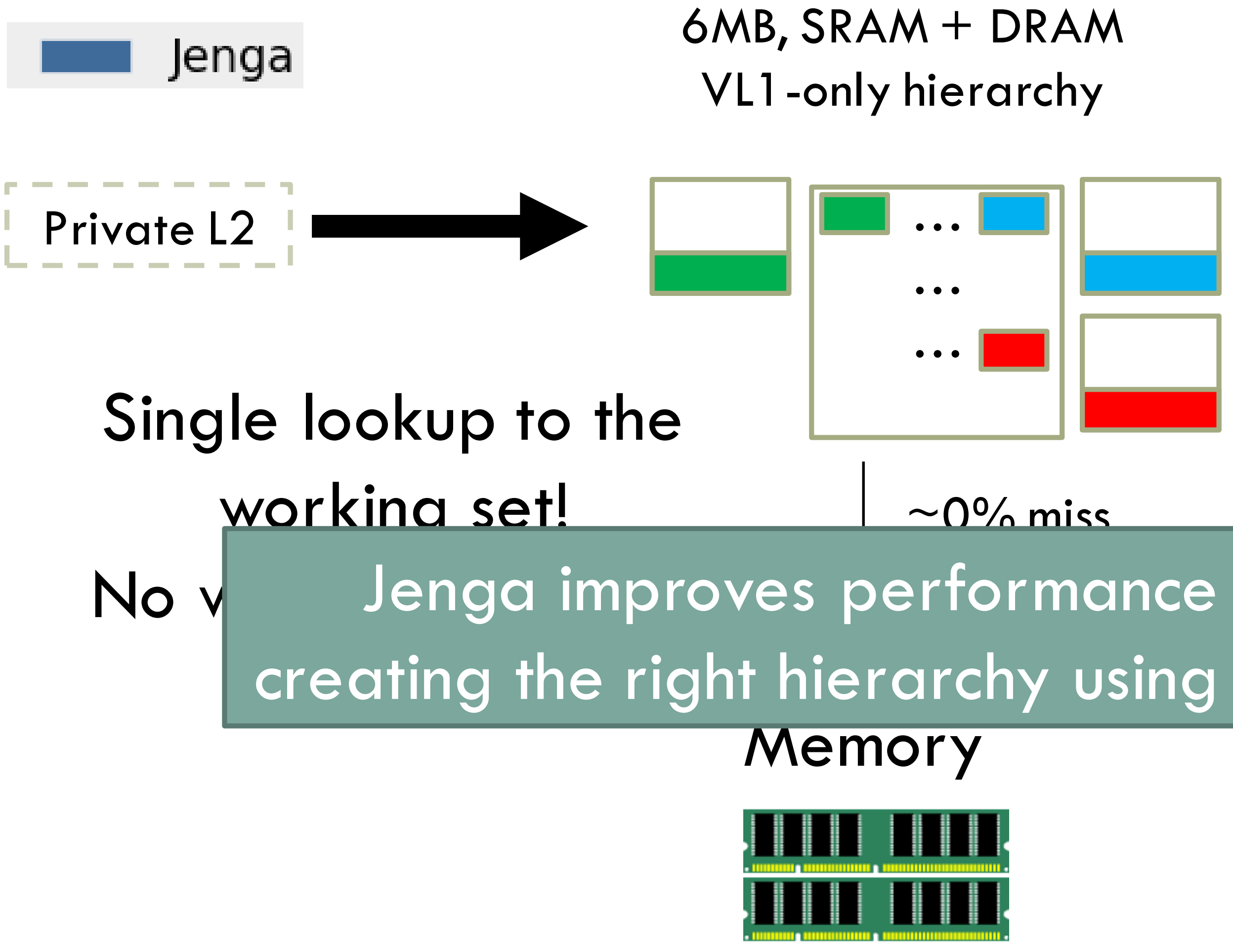
6MB, SRAM + DRAM
VL1-only hierarchy

Single lookup to the working set!
No wasteful lookups!



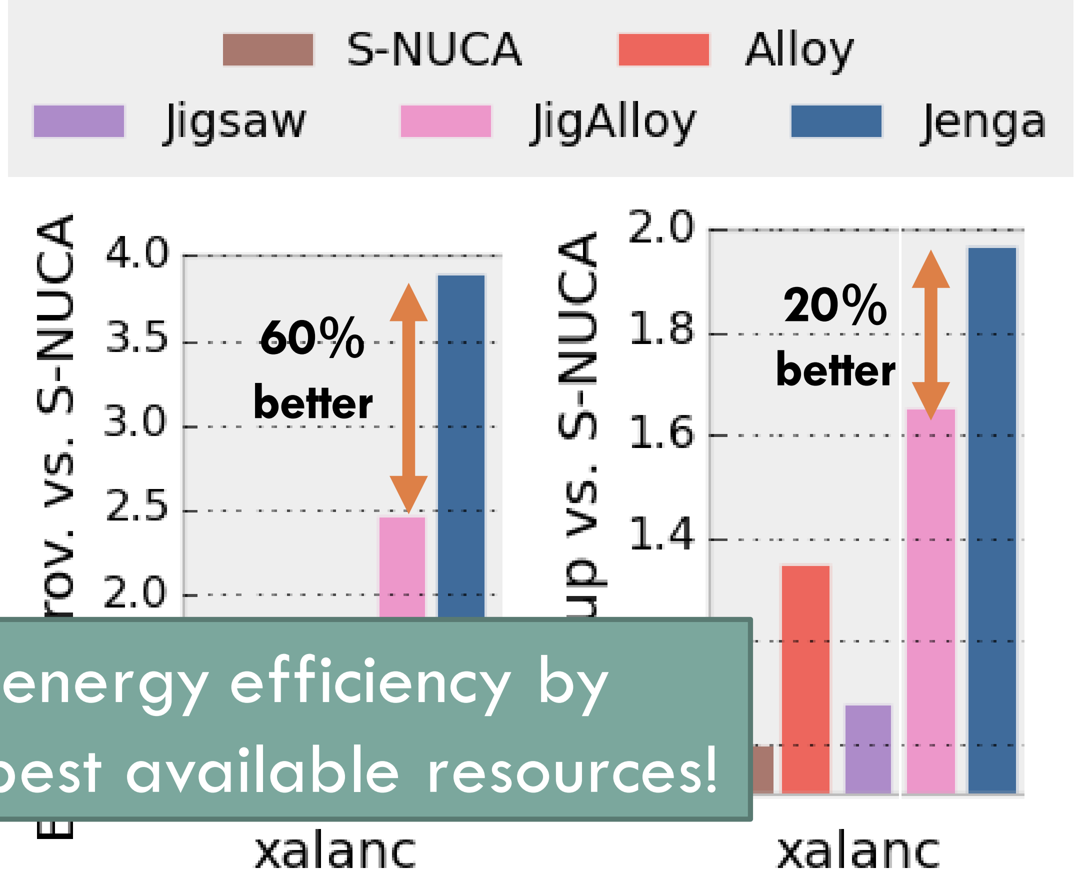
Case study: 36 copies of xalanc

Working set: 6MB x 36 = 216 MB



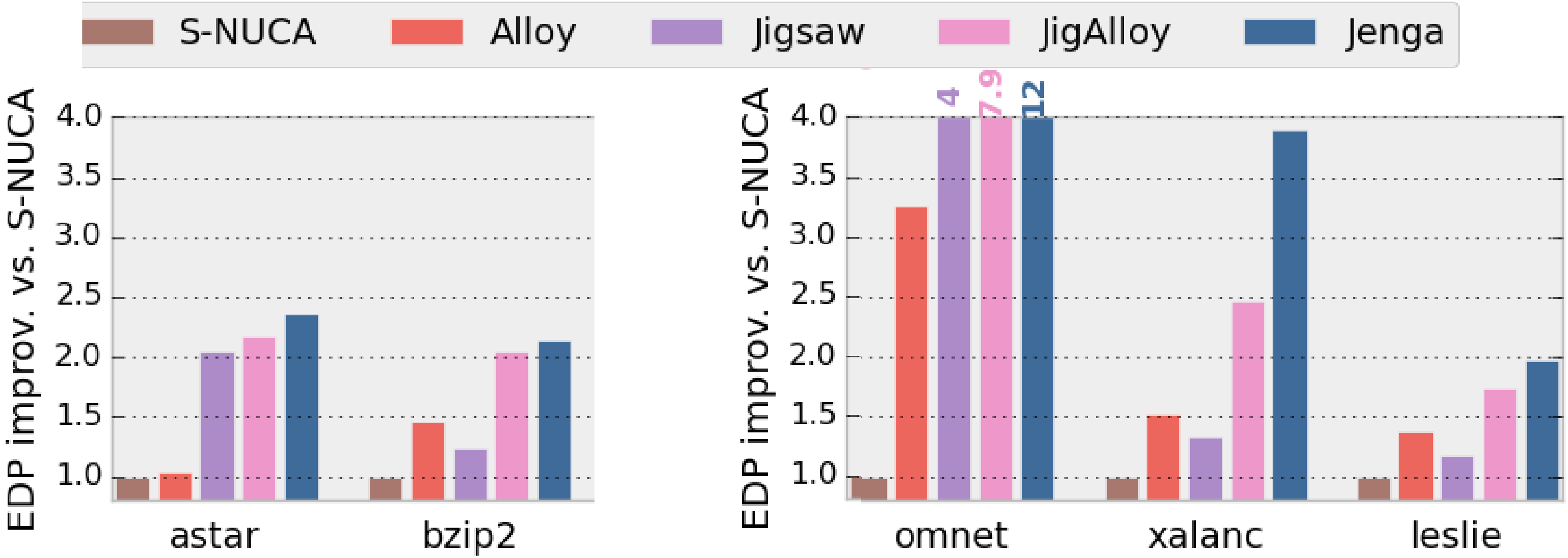
Single lookup to the working set!

No v



Jenga improves performance and energy efficiency by creating the right hierarchy using the best available resources!

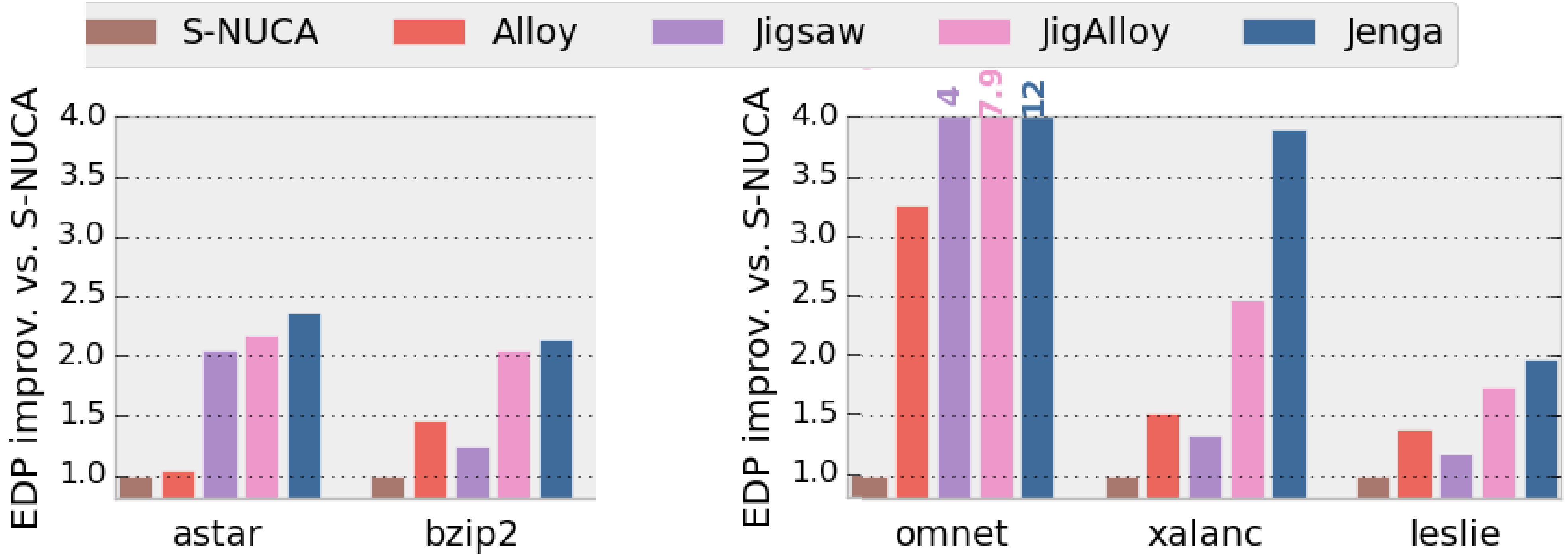
Jenga works across a wide range of behaviors



App with two-level working set

App with flat working set

Jenga works across a wide range of behaviors

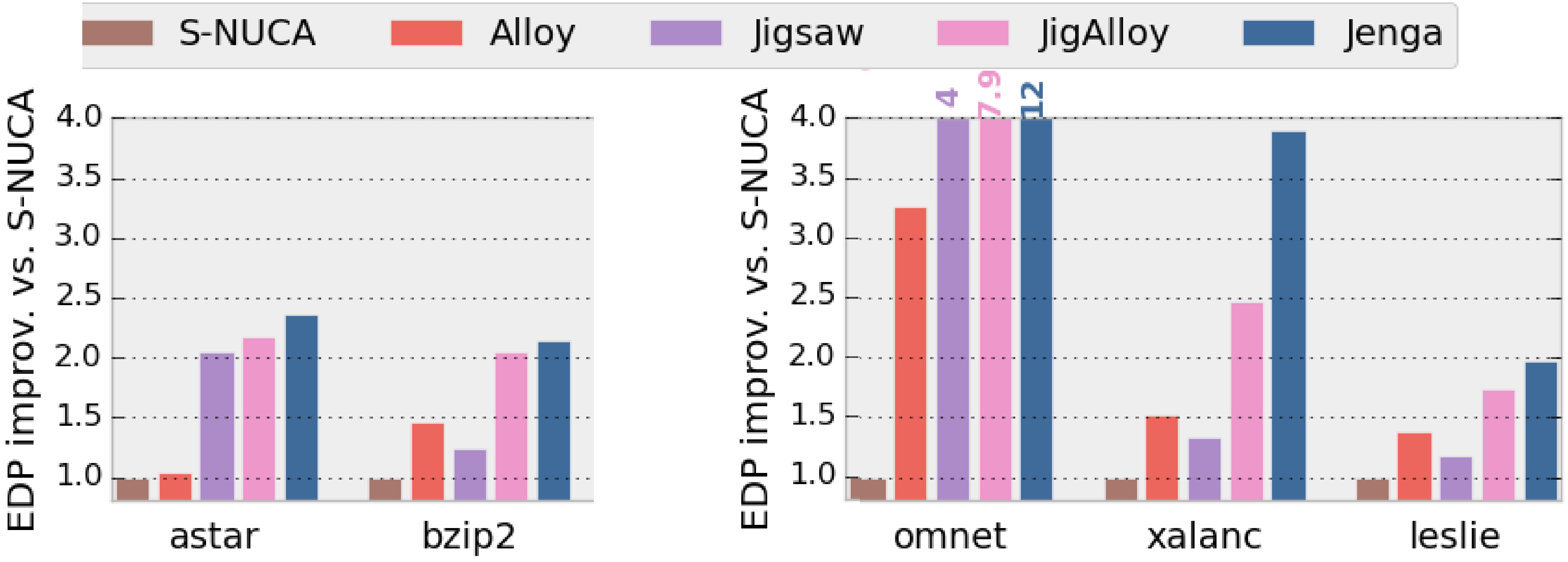


App with two-level working set

App with flat working set

Working set	0.5MB + 16MB	1MB + 8MB
Jenga VHs	SRAM VL1 DRAM VL2	SRAM+DRAM VL1 DRAM VL2

Jenga works across a wide range of behaviors

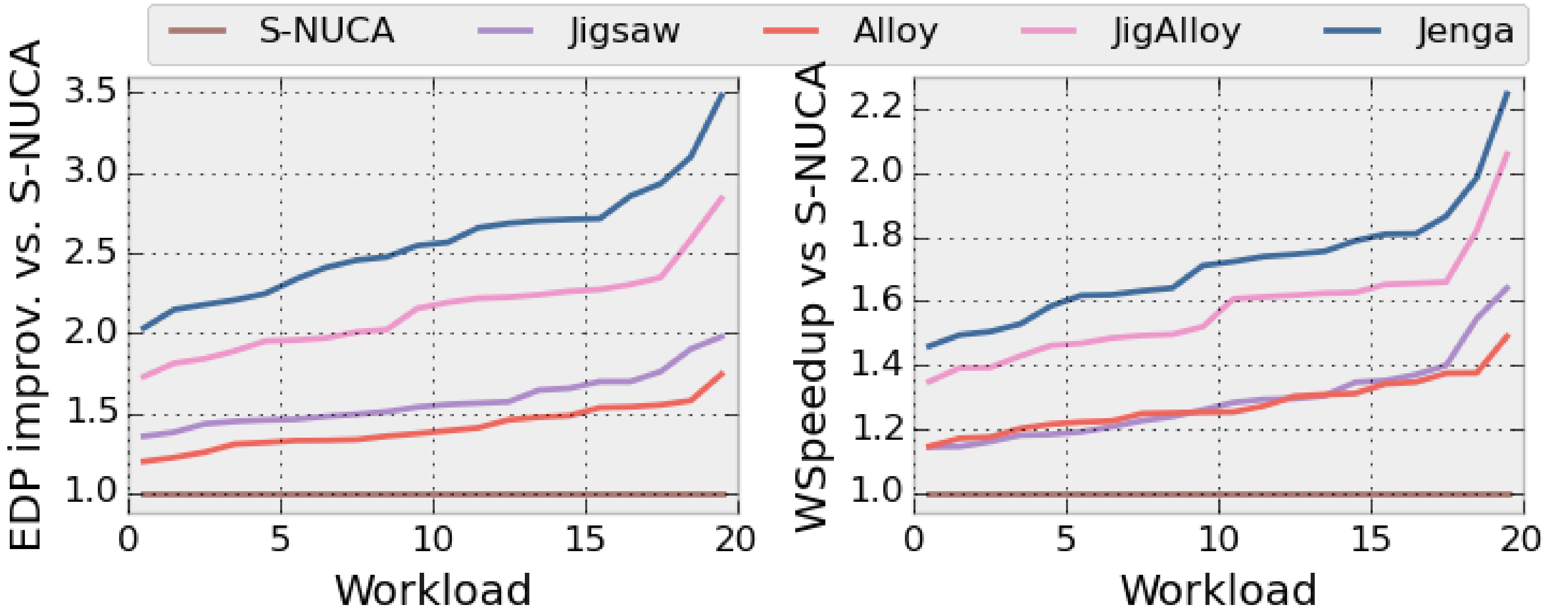


App with two-level working set

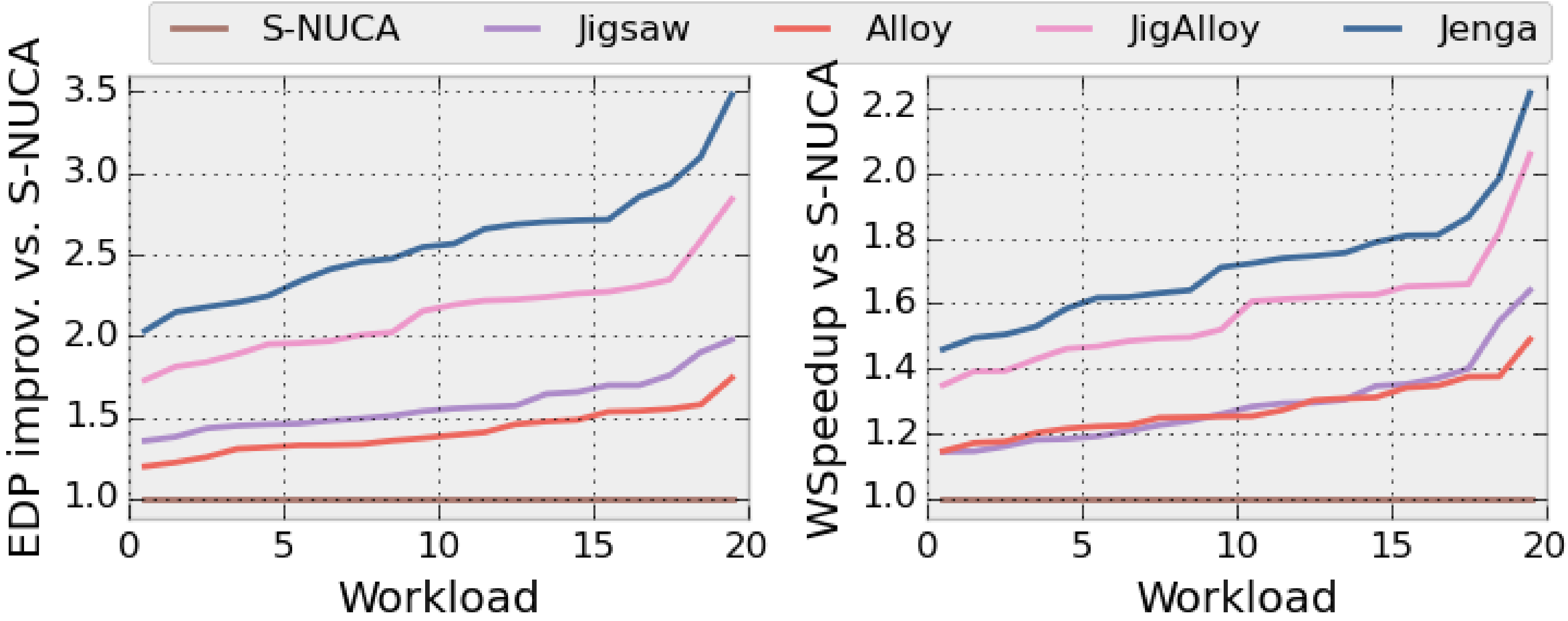
App with flat working set

Working set	0.5MB + 16MB	1MB + 8MB	2.5MB	8MB	>50MB
Jenga VHs	SRAM VL1 DRAM VL2	SRAM+DRAM VL1 DRAM VL2	SRAM+ DRAM VL1	SRAM+ DRAM VL1	DRAM VL1 or No caching

Jenga works for random multi-program mixes



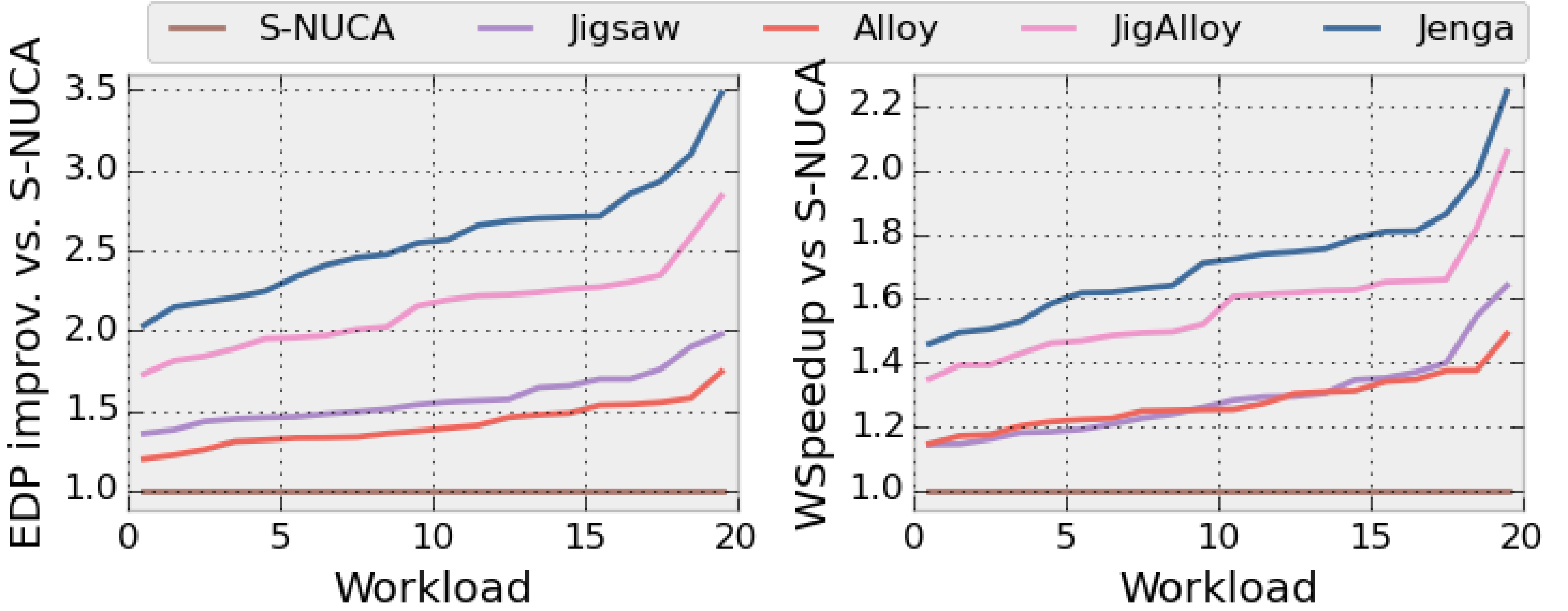
Jenga works for random multi-program mixes



2.6X over S-NUCA

20% over JigAlloy

Jenga works for random multi-program mixes



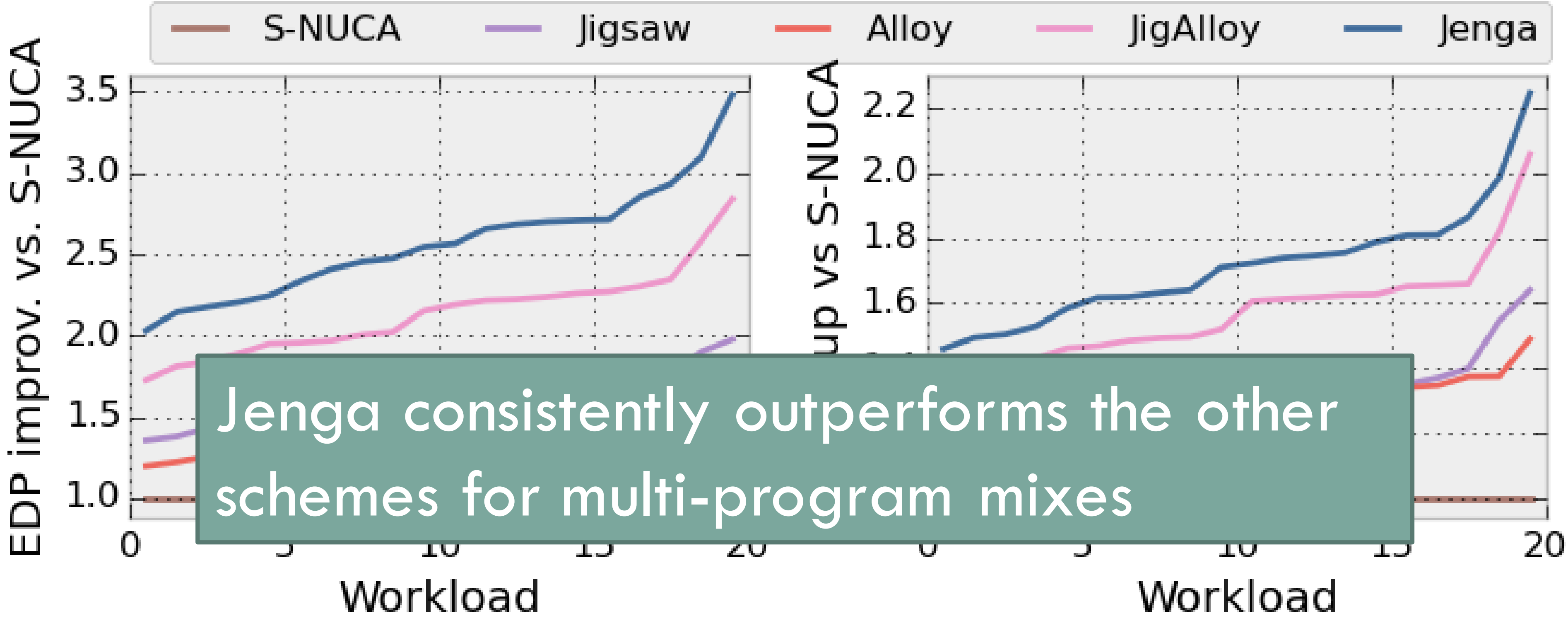
2.6X over S-NUCA

1.7X over S-NUCA

20% over JigAlloy

10% over JigAlloy

Jenga works for random multi-program mixes



Jenga consistently outperforms the other schemes for multi-program mixes

2.6X over S-NUCA

1.7X over S-NUCA

20% over JigAlloy

10% over JigAlloy

See paper for more results

□ Full result for SPECCPU-rate

□ Multithreaded apps

□ Sensitivity study for Jenga’s software techniques

□ 2.5D DRAM architectures

□ Jigsaw SRAM L3 + Jigsaw DRAM L4

□ And more

Conclusion

Conclusion

- Rigid, multi-level cache hierarchies are ill-suited to many applications
  - They cause significant overhead when they are not helpful

Conclusion

- Rigid, multi-level cache hierarchies are ill-suited to many applications
  - They cause significant overhead when they are not helpful

- We propose Jenga, a software-defined, reconfigurable cache hierarchy
  - Adopts application-specific organization on-the-fly
  - Uses new software algorithm to find near-optimal hierarchy efficiently

Conclusion

- Rigid, multi-level cache hierarchies are ill-suited to many applications
  - They cause significant overhead when they are not helpful

- We propose Jenga, a software-defined, reconfigurable cache hierarchy
  - Adopts application-specific organization on-the-fly
  - Uses new software algorithm to find near-optimal hierarchy efficiently

- Jenga improves both performance and energy efficiency, by up to 85% in EDP, over a combination of state-of-art techniques

Thanks! Questions?

- Rigid, multi-level cache hierarchies are ill-suited to many applications
  - They cause significant overhead when they are not helpful

- We propose Jenga, a software-defined, reconfigurable cache hierarchy
  - Adopts application-specific organization on-the-fly
  - Uses new software algorithm to find near-optimal hierarchy efficiently

- Jenga improves both performance and energy efficiency, by up to 85% in EDP, over a combination of state-of-art techniques

Jenga: Software-Defined Cache Hierarchies

Thank you for your attention!

Questions?

MIT CSAIL

Carnegie Mellon University
School of Computer Science