Fast RDMA-based Ordered Key-Value Store using Remote Learned Cache

Xingda Wei, Rong Chen, Haibo Chen

IPADS
INSTITUTE OF PARALLEL AND DISTRIBUTED SYSTEMS

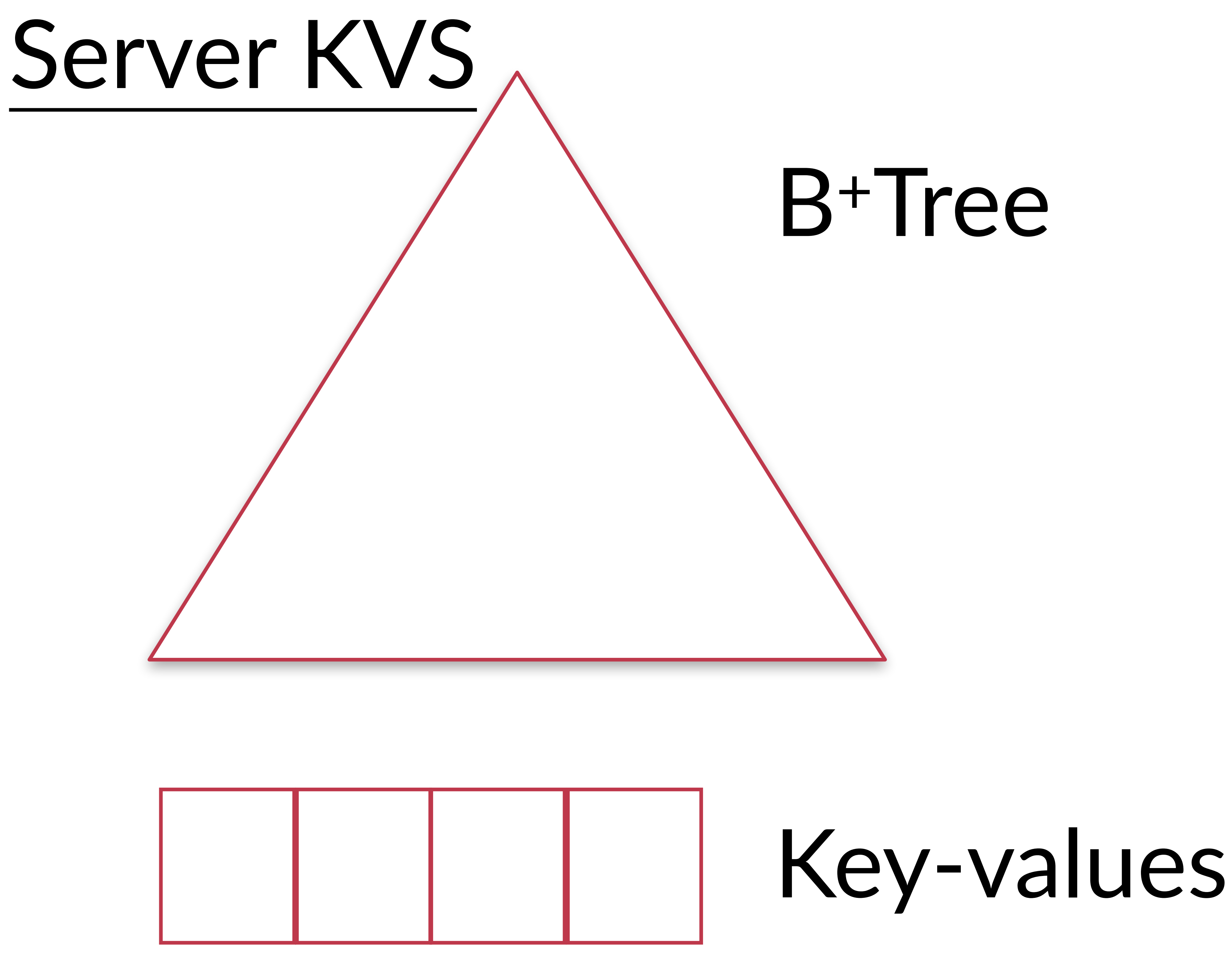
XSTORE

KVS: key pillar for distributed systems

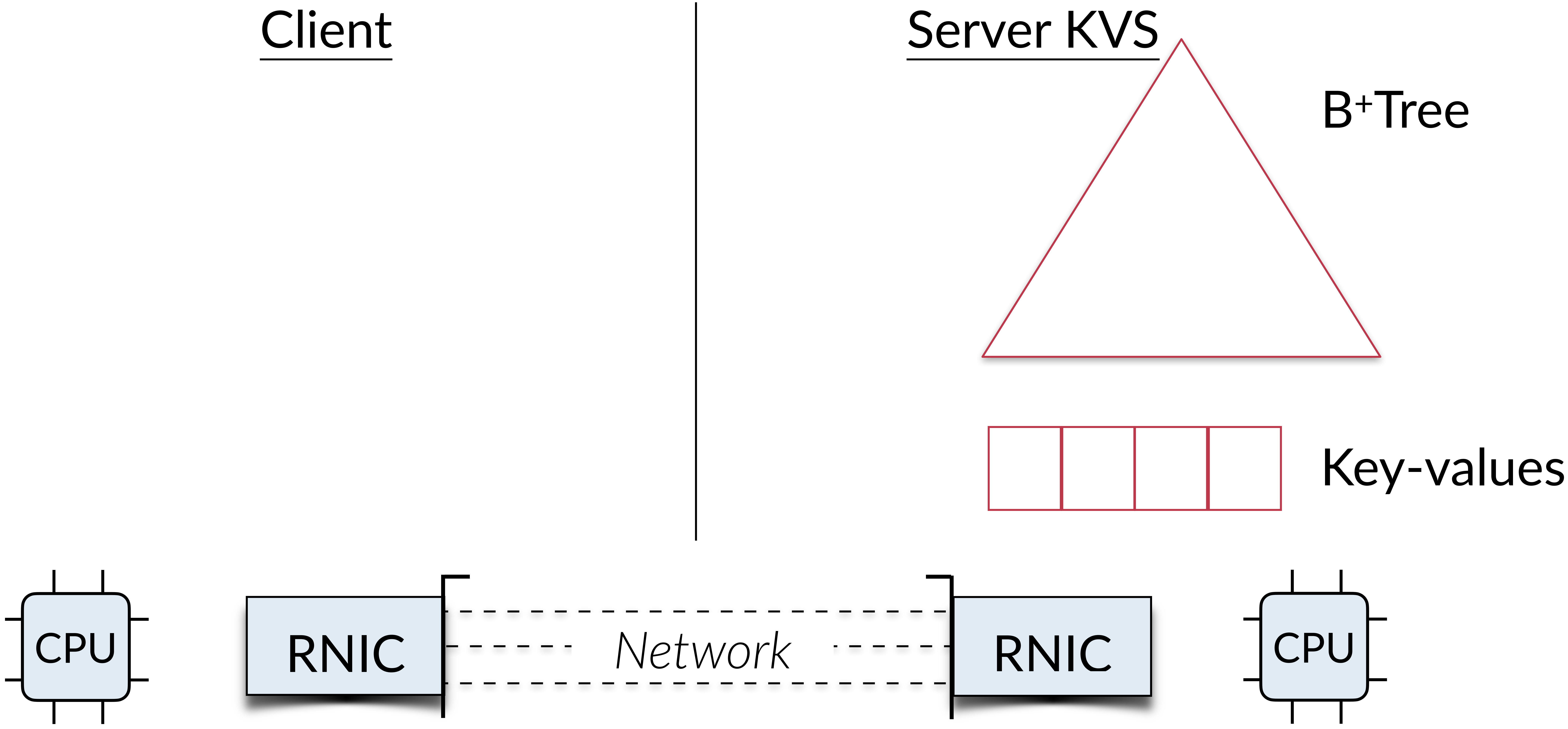
Important ***building block*** for

- Databases, GraphStore
- Web applications
- Cloud infrastructures
- Serverless platforms

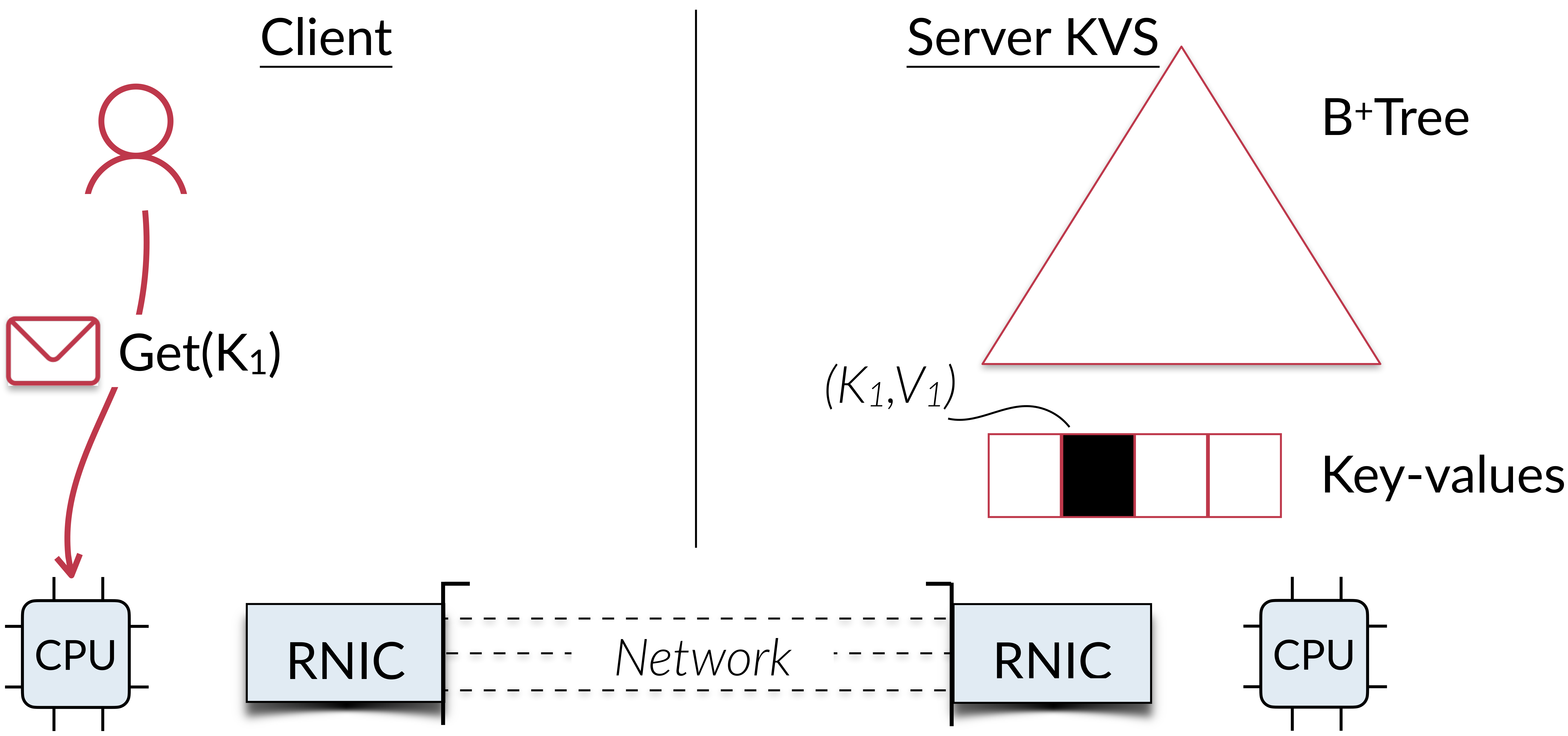
KVS: key pillar for distributed systems



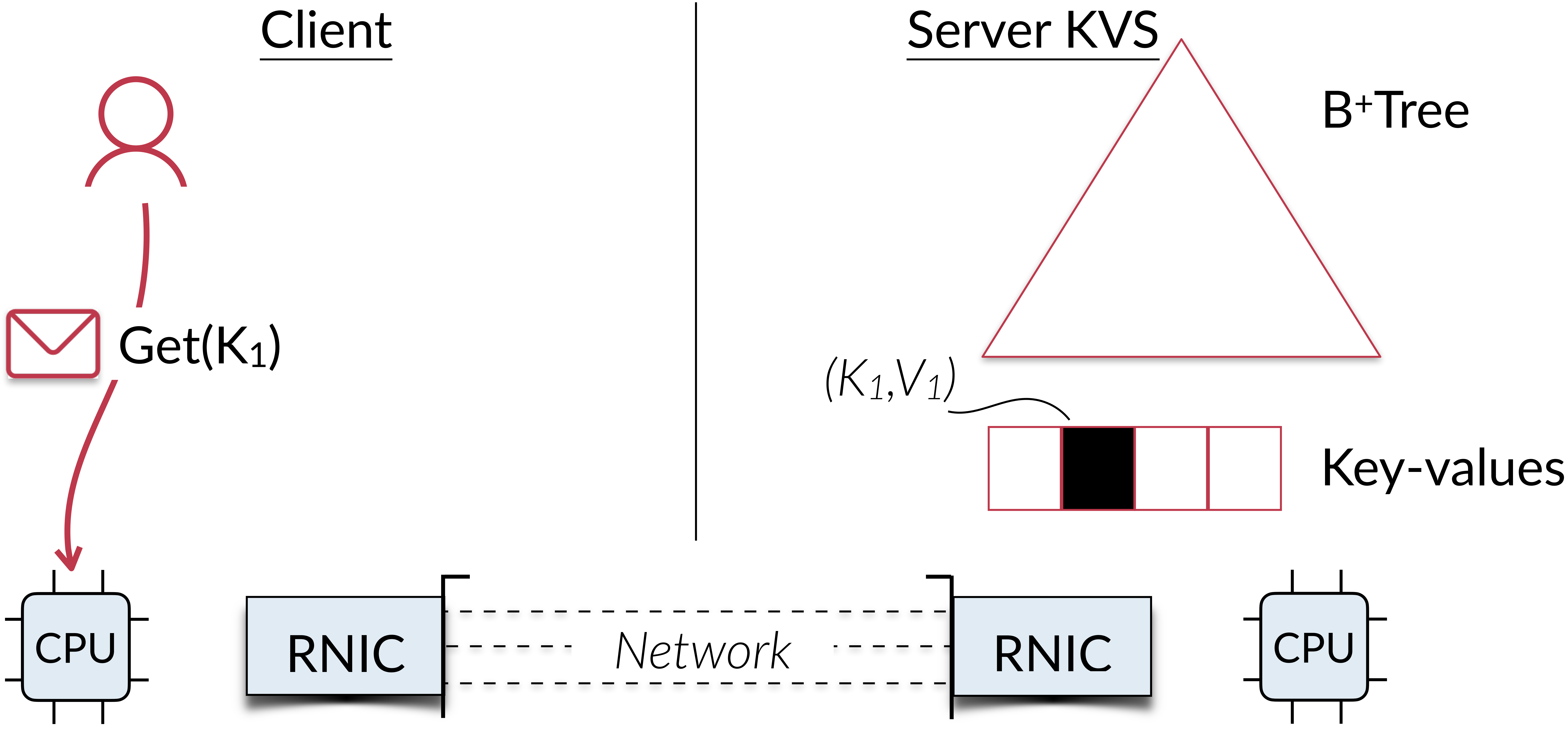
KVS: key pillar for distributed systems



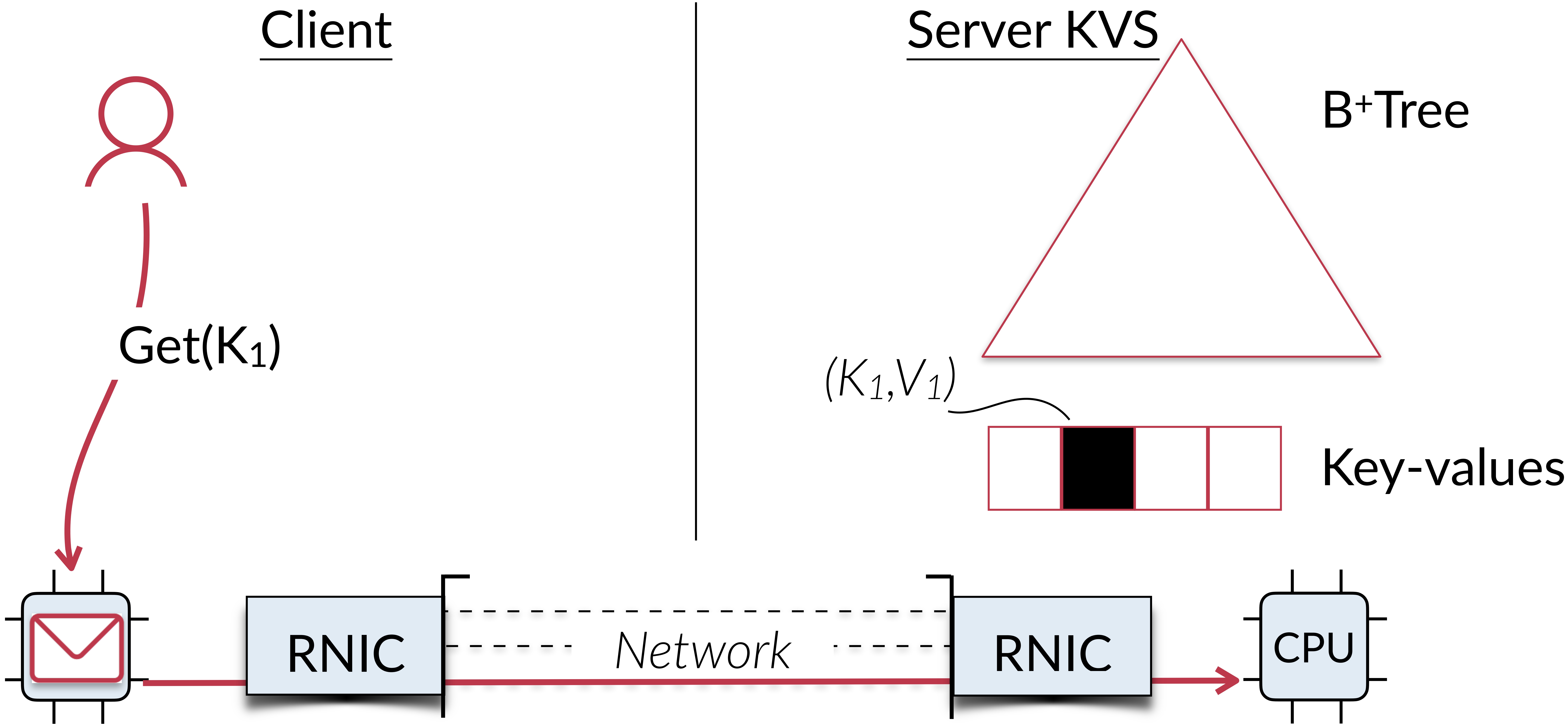
Traditional KVS uses RPC (Server-centric)



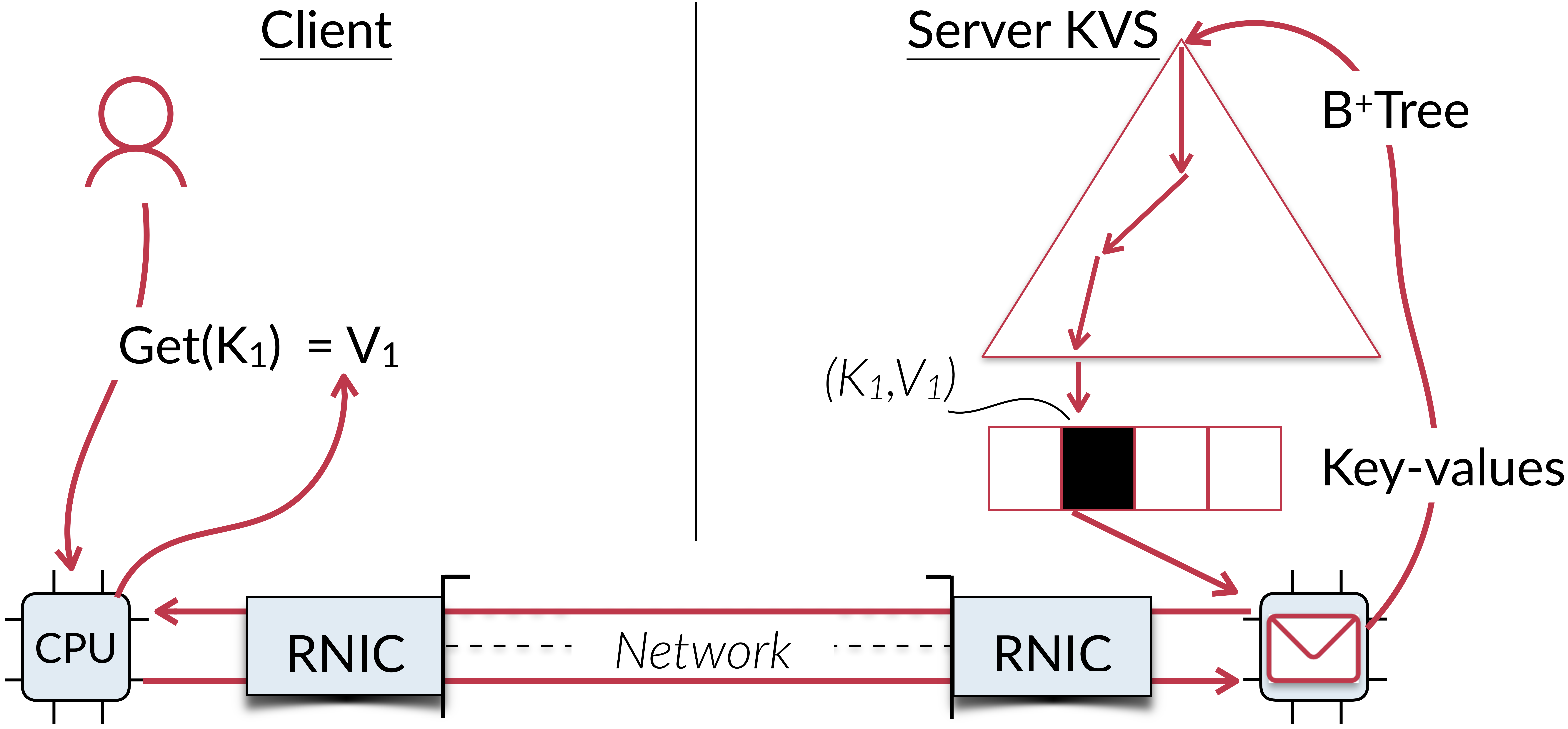
Traditional KVS uses RPC (Server-centric)



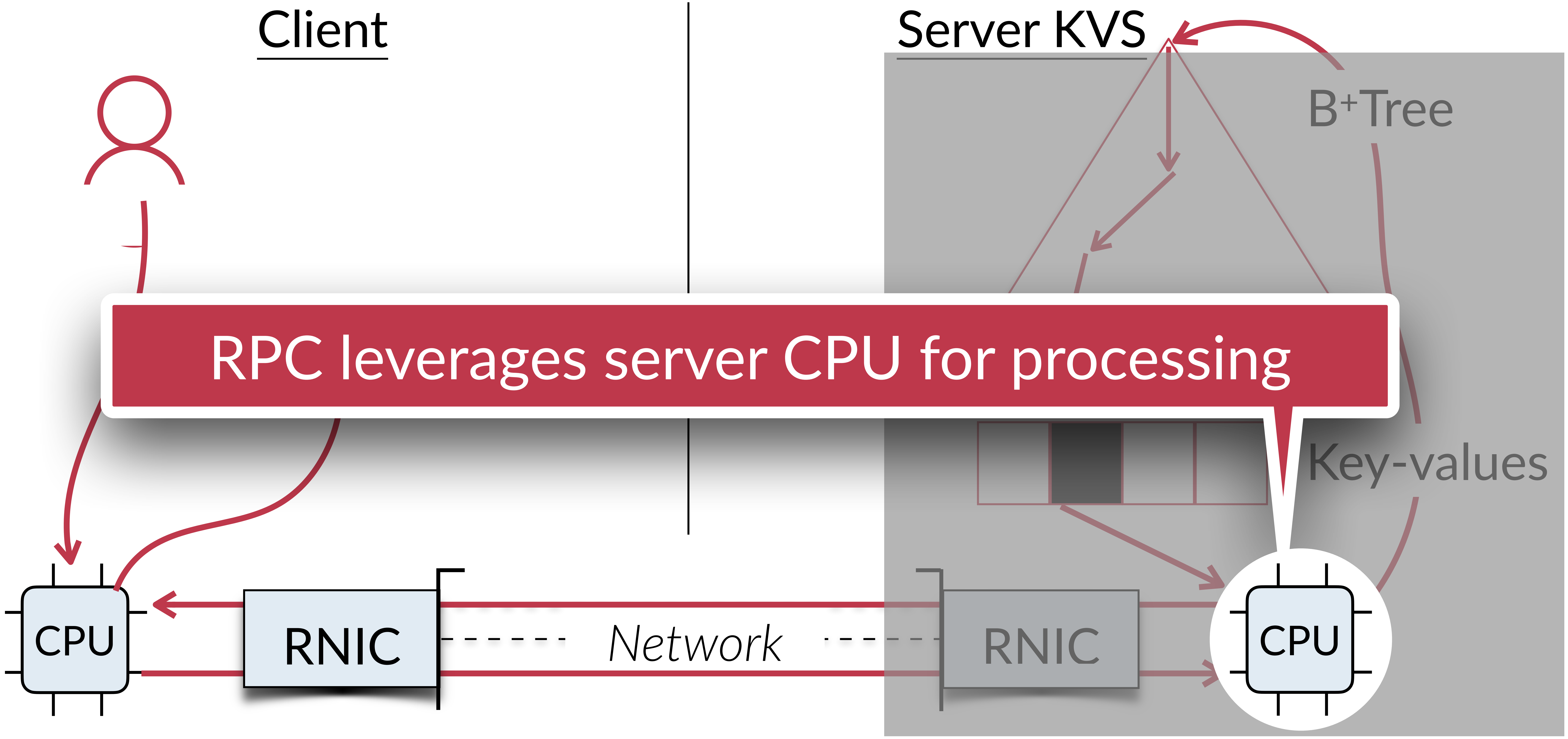
Traditional KVS uses RPC (Server-centric)



Traditional KVS uses RPC (Server-centric)



Traditional KVS uses RPC (Server-centric)

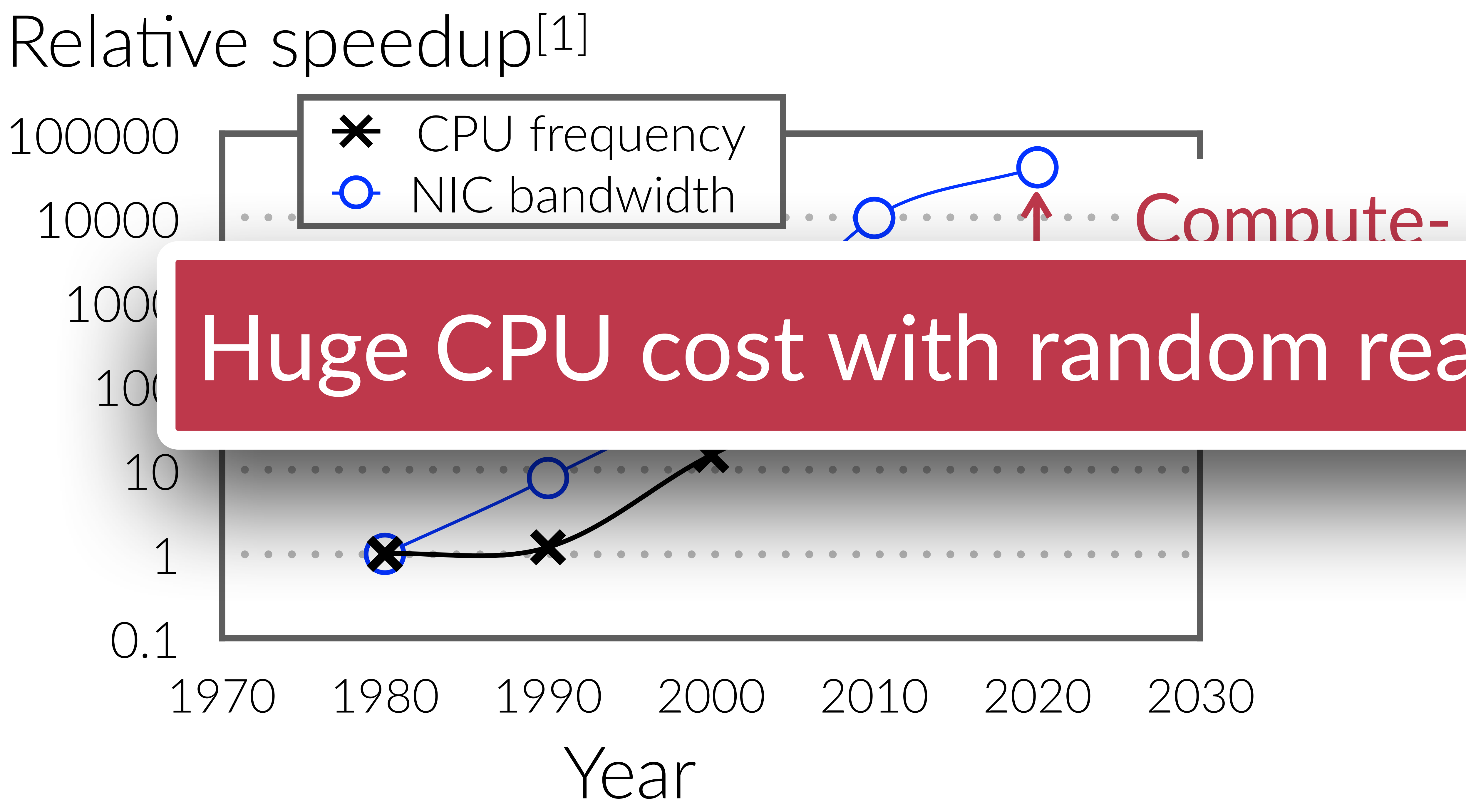


RPC leverages server CPU for processing

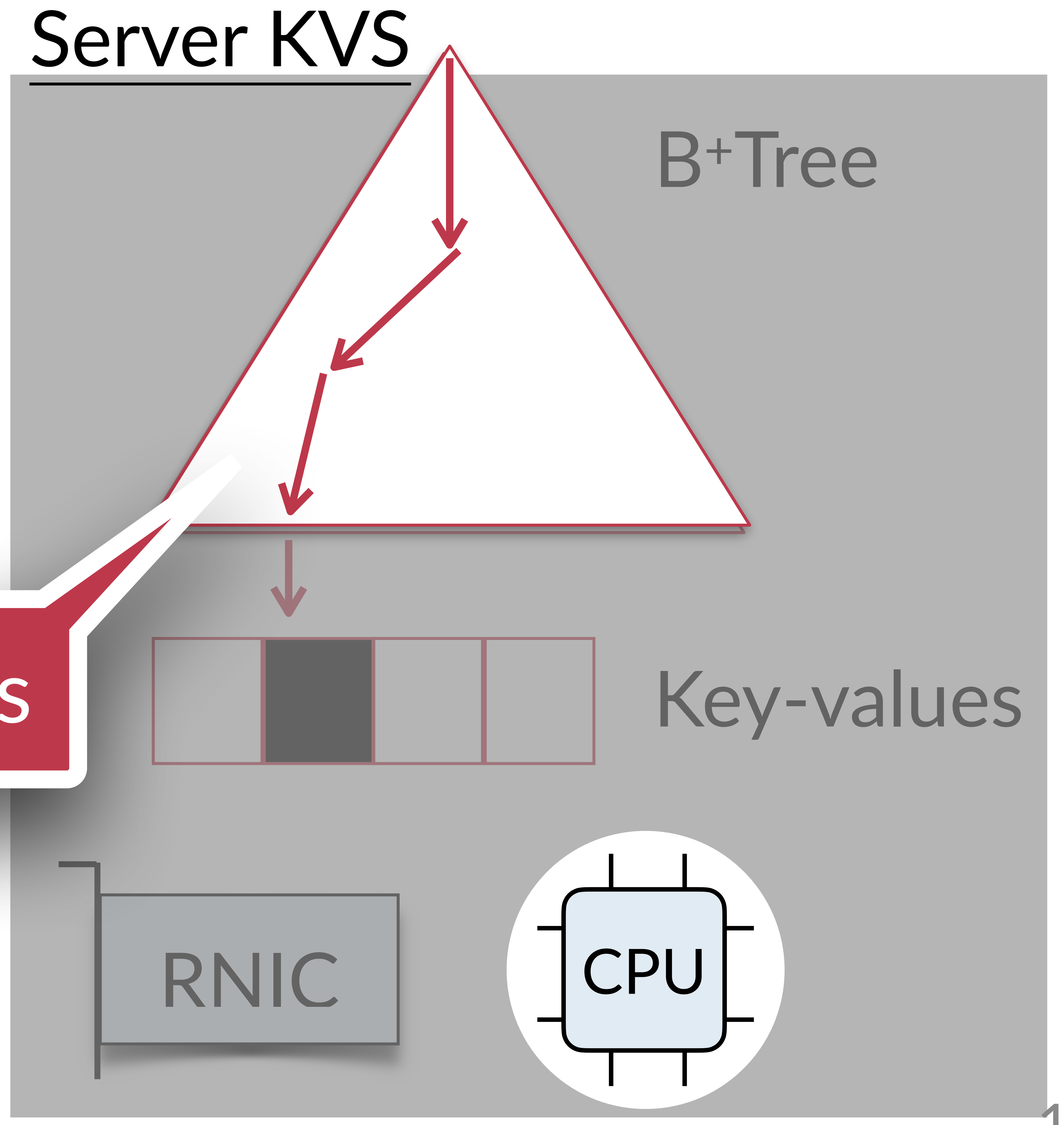
Server CPU is becoming the bottleneck

Increasing ***CPU-NIC gap***

- NIC's speed is growing faster !



Huge CPU cost with random reads

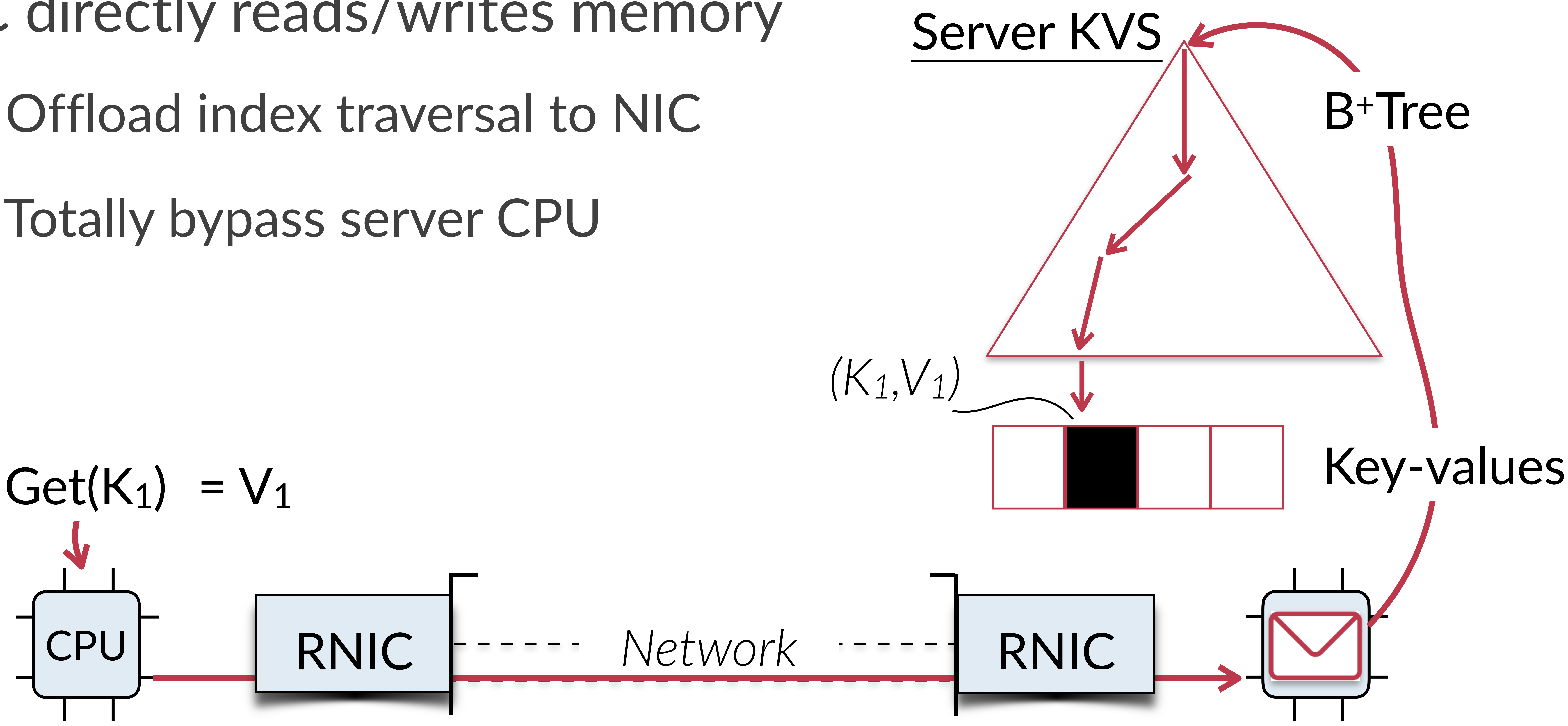


[1] Credits: StRoM: Smart Remote Memory @ Eurosys'20

Opportunity: one-sided RDMA (Client-direct)

NIC directly reads/writes memory

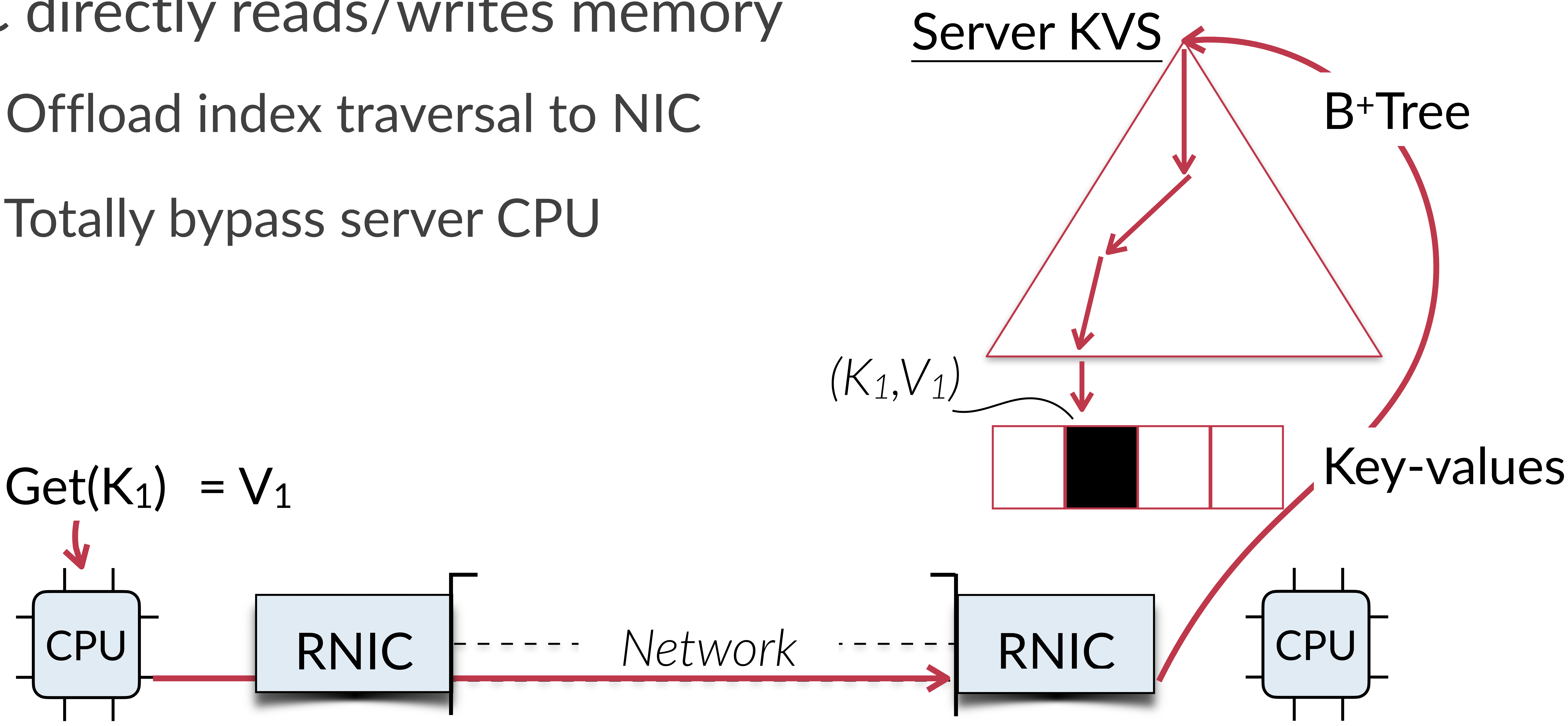
- Offload index traversal to NIC
- Totally bypass server CPU



Opportunity: one-sided RDMA (Client-direct)

NIC directly reads/writes memory

- Offload index traversal to NIC
- Totally bypass server CPU



Challenge: limited NIC abstraction

NIC only has ***simple*** abstractions

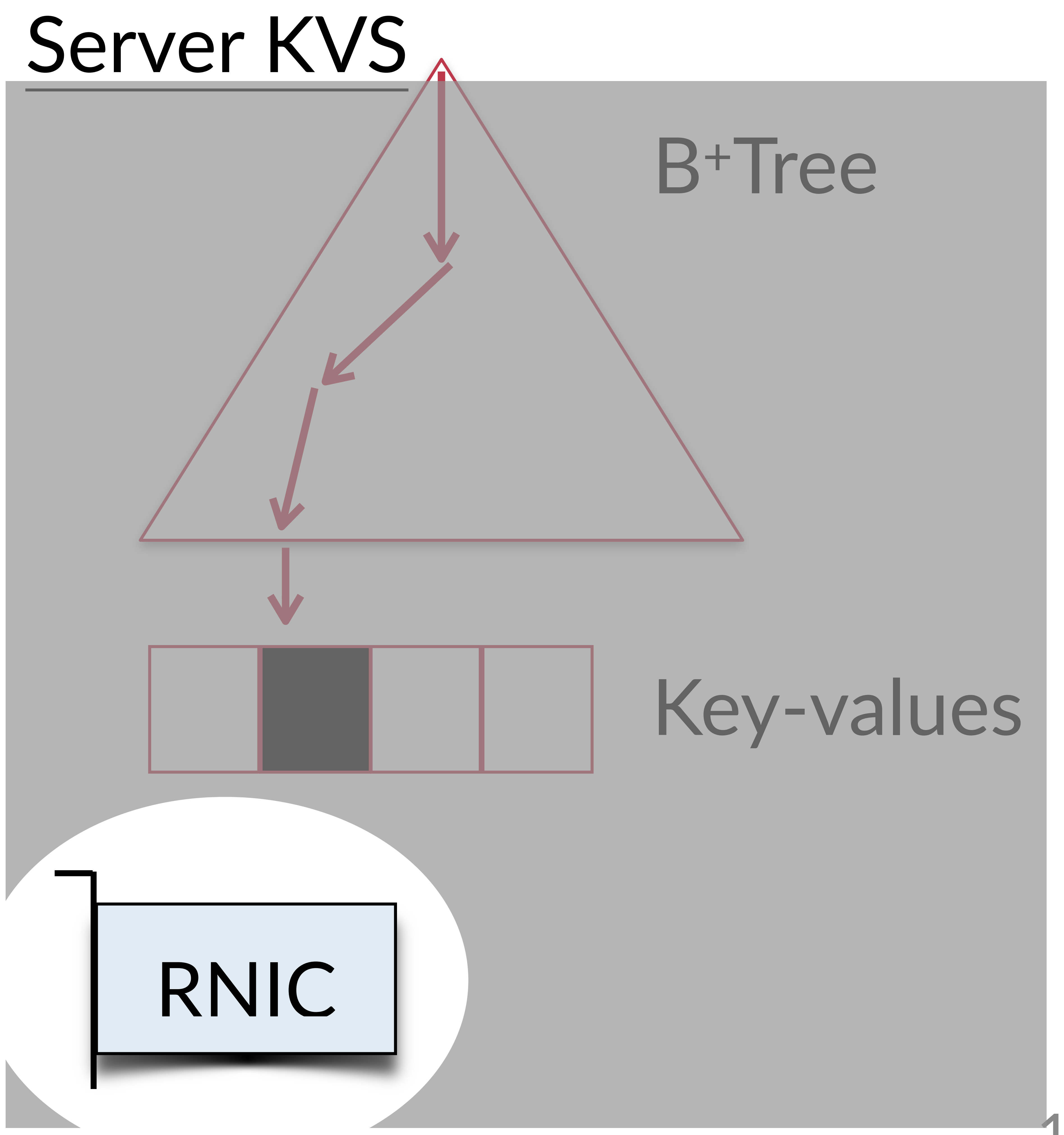
- e.g., memory ***read/write***

Works well for simple index structure

- e.g. ***HashTable, O(1)*** network RTT[1]

Inferior for complex index structure

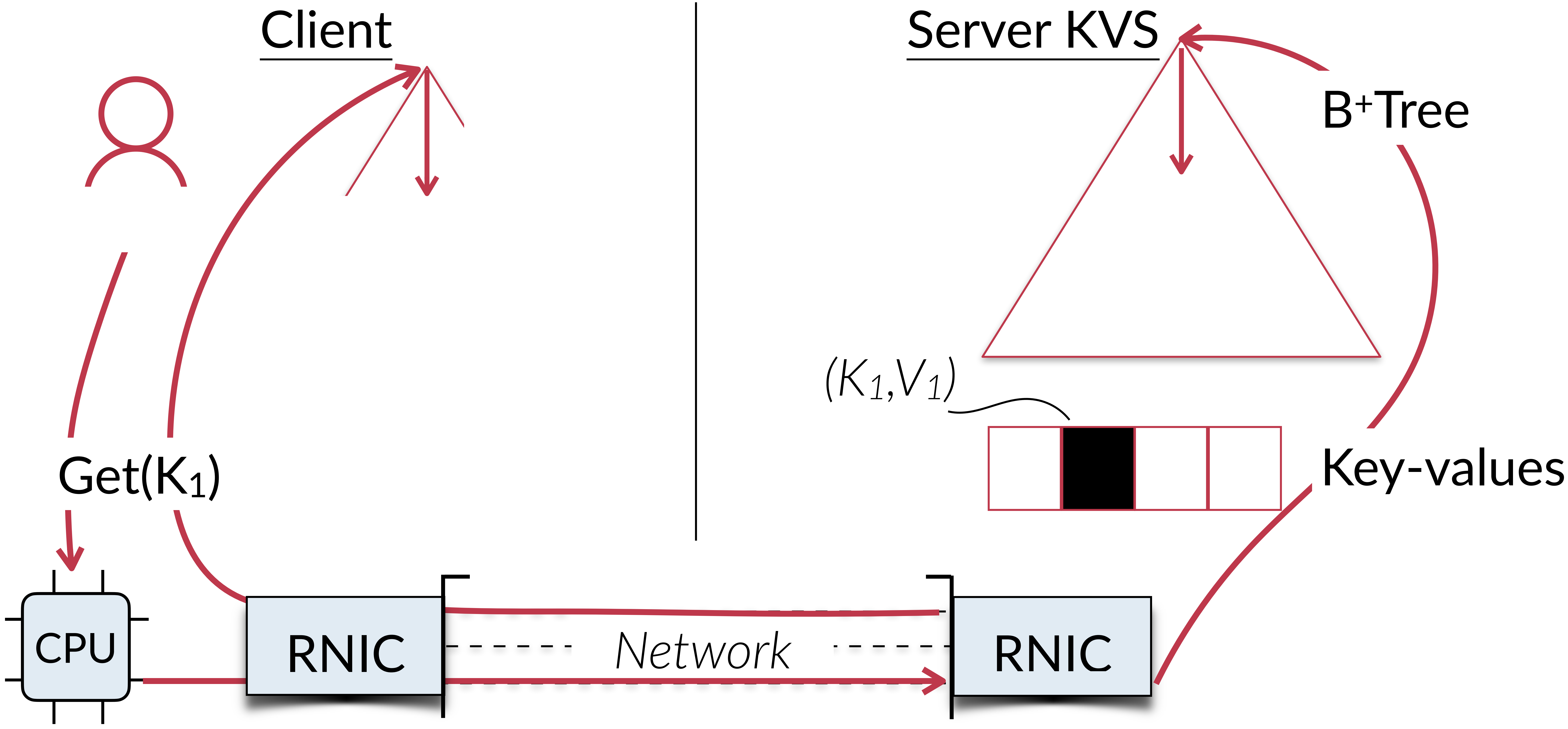
- e.g., ***B+Tree, O(log(n))***[2] network RTT



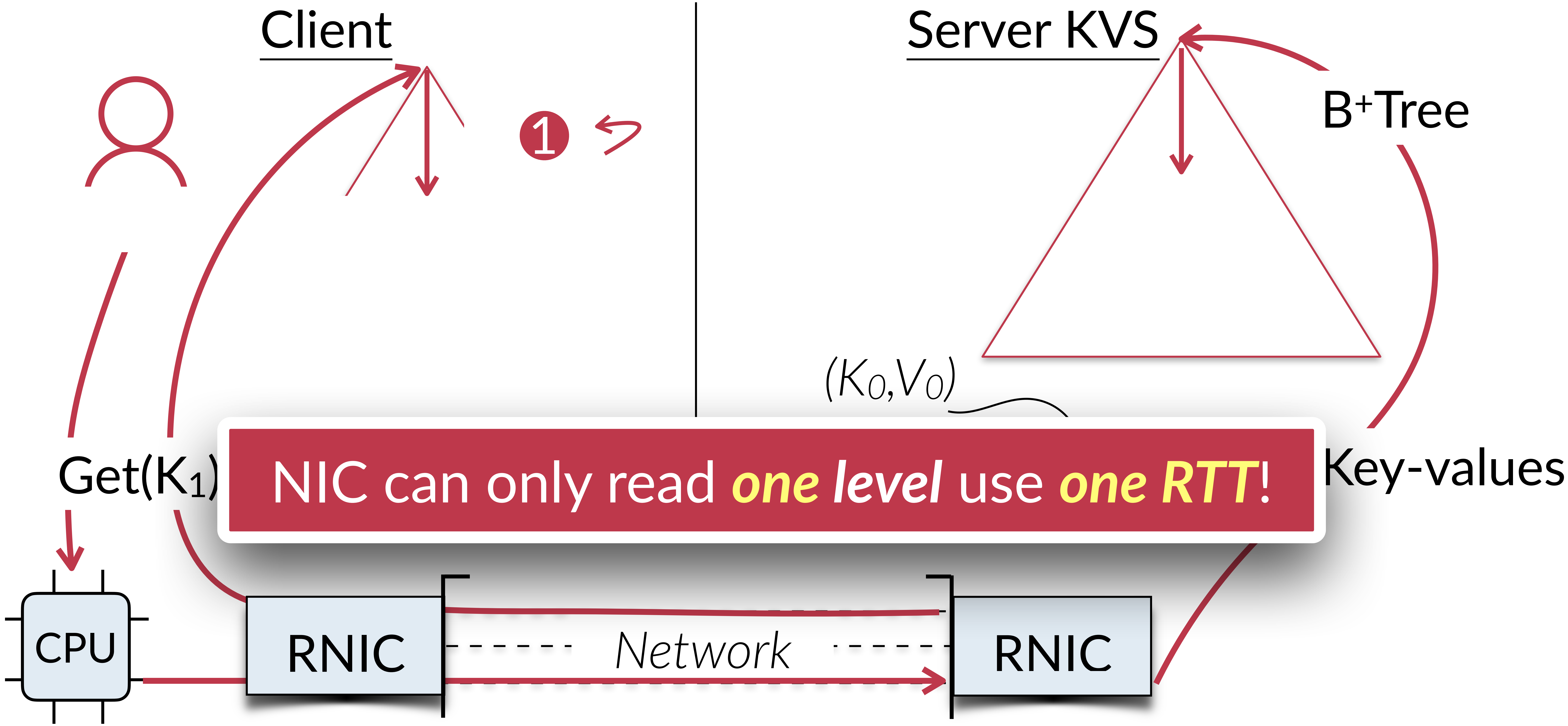
[1] RTT: roundtrip time

[2] n:the scale of the KVS

Challenge: limited NIC abstraction

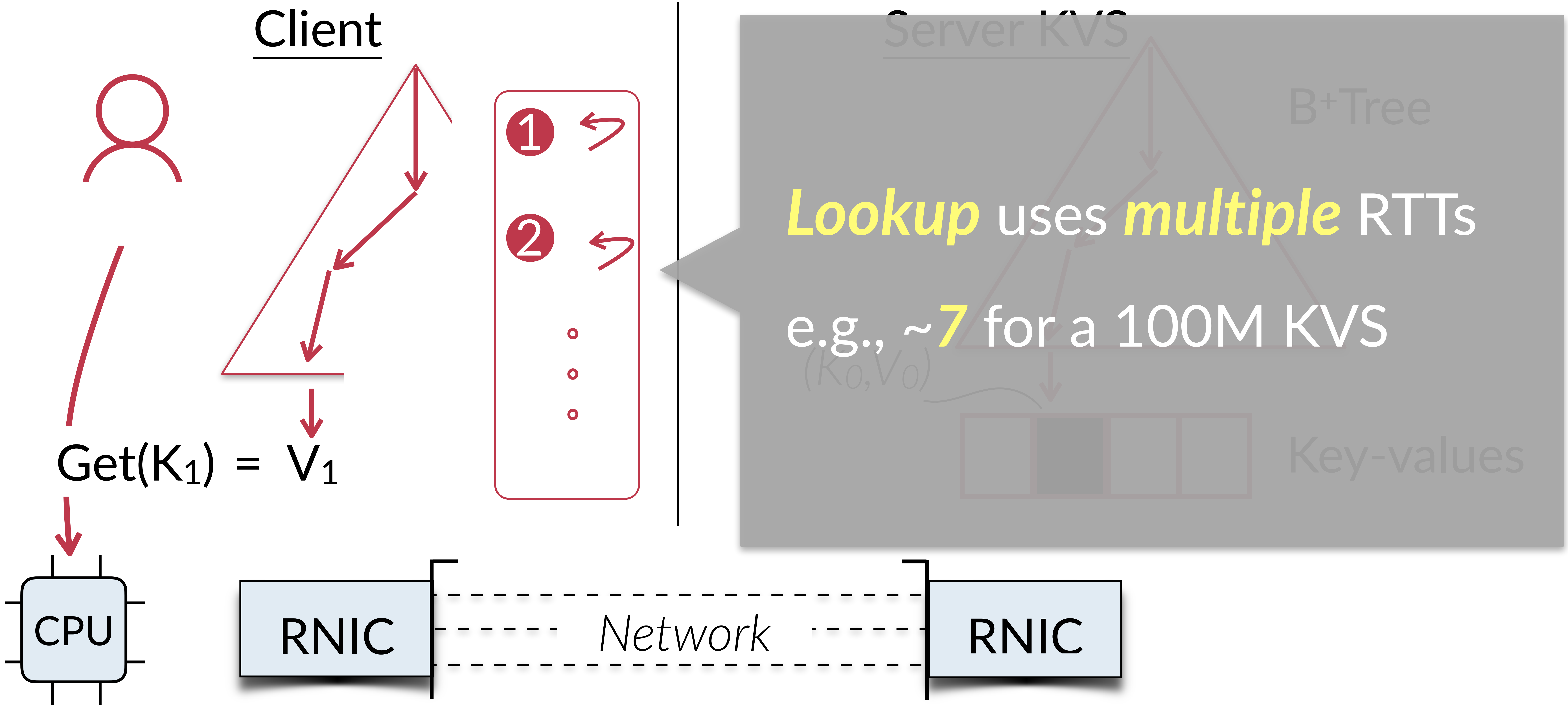


Challenge: limited NIC abstraction



NIC can only read ***one level*** use ***one RTT***!

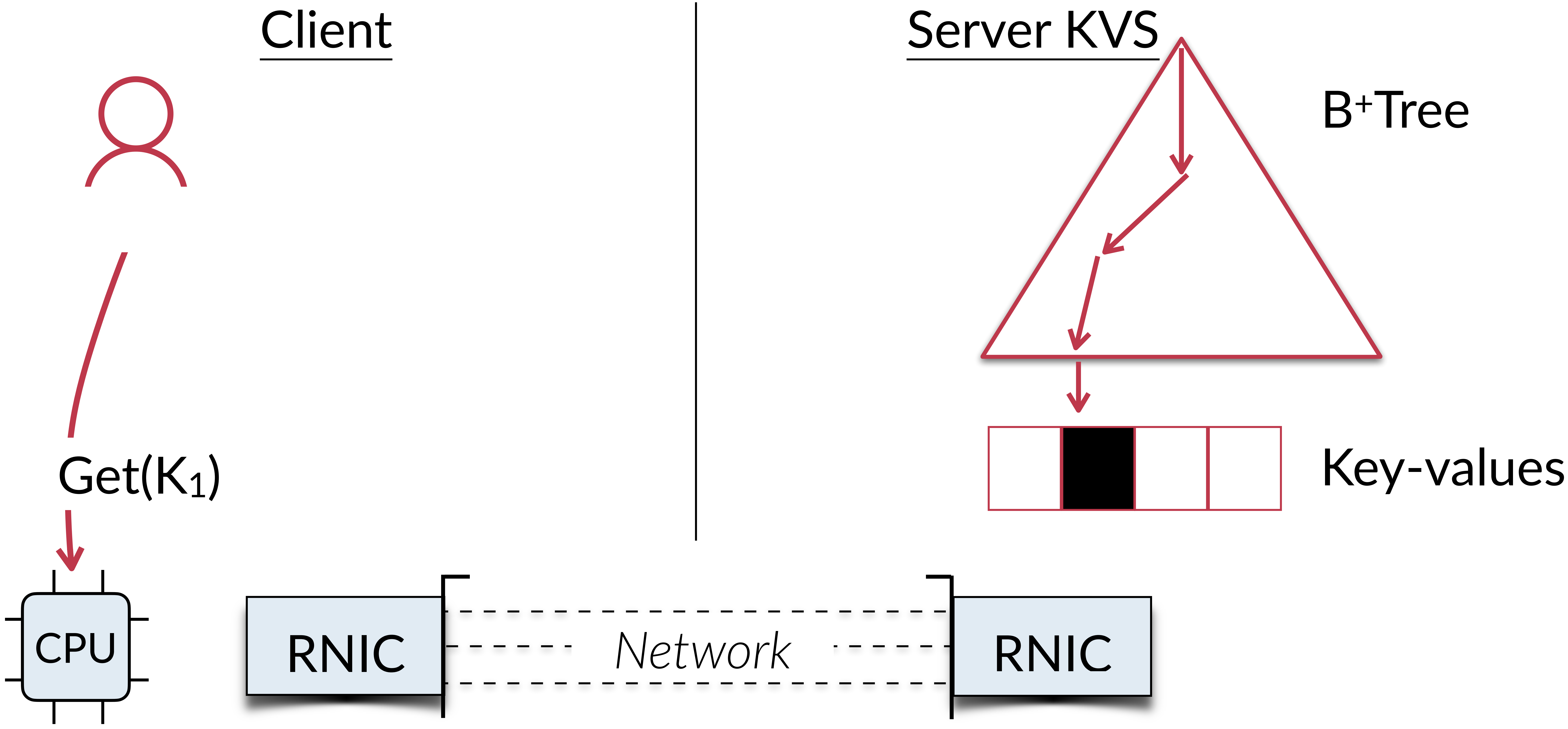
Challenge: limited NIC abstraction



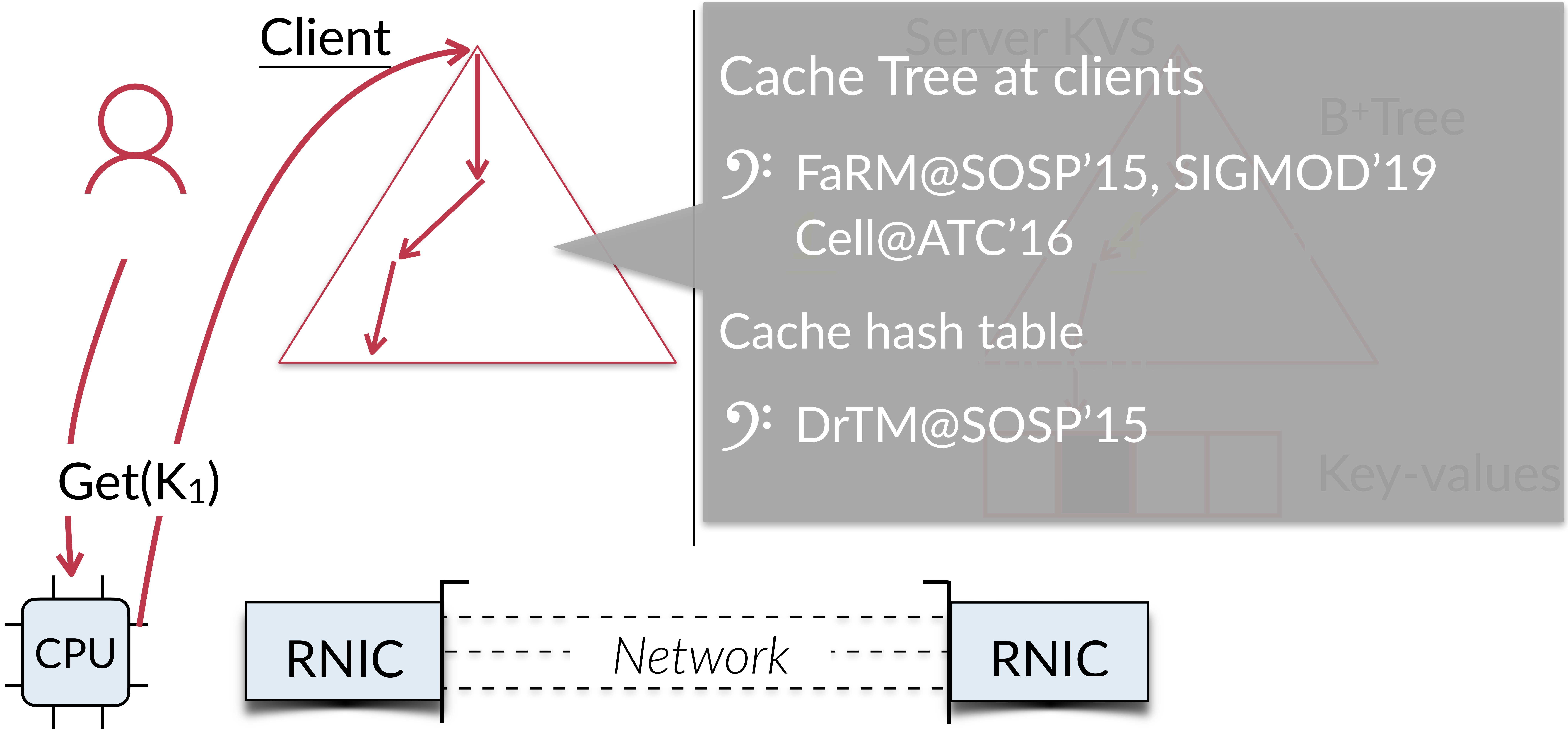
Lookup uses ***multiple*** RTTs

e.g., ~**7** for a 100M KVS

Existing systems adopt caching



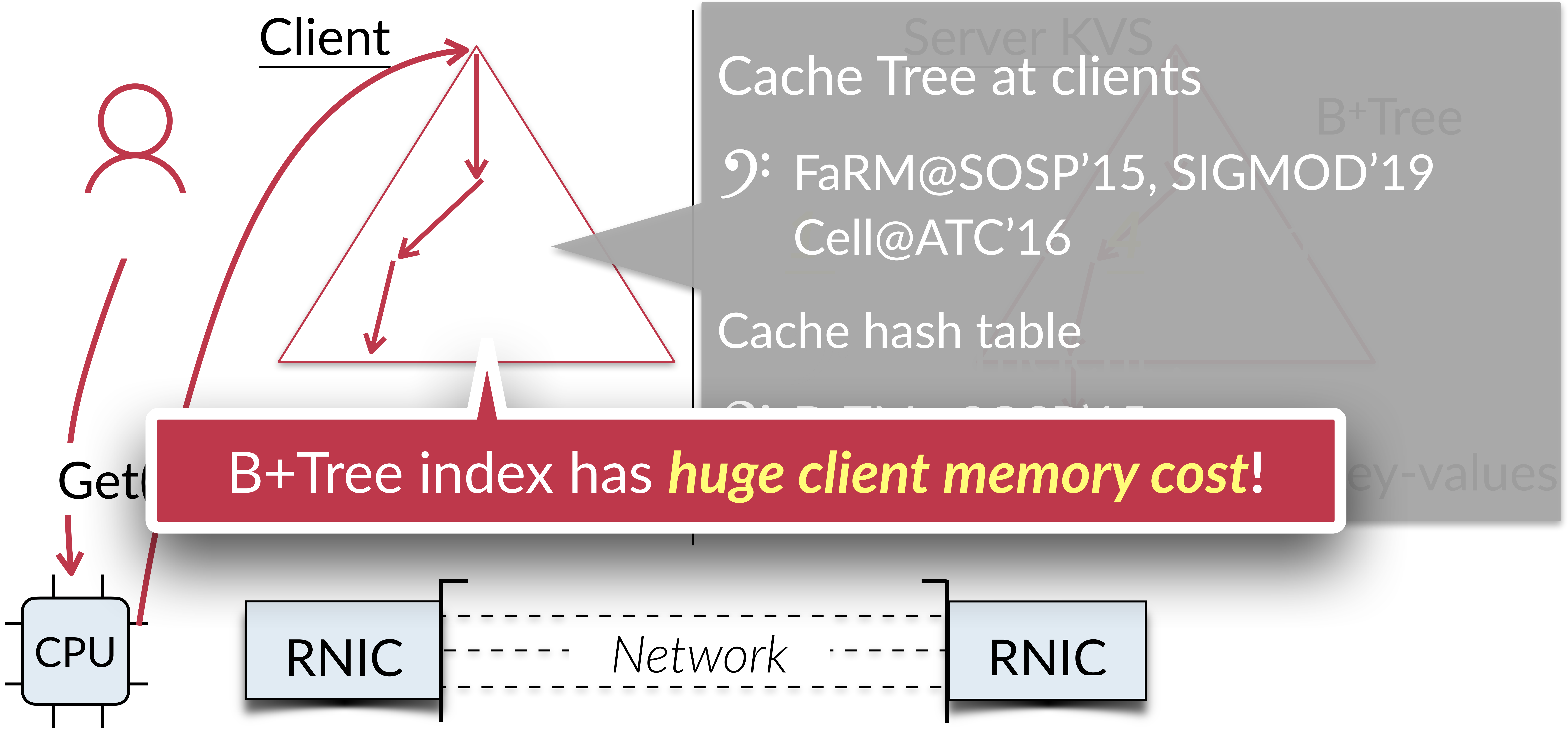
Existing systems adopt caching



Cache Tree at clients
- FaRM@SOSP'15, SIGMOD'19
  Cell@ATC'16

Cache hash table
- DrTM@SOSP'15

Existing systems adopt caching



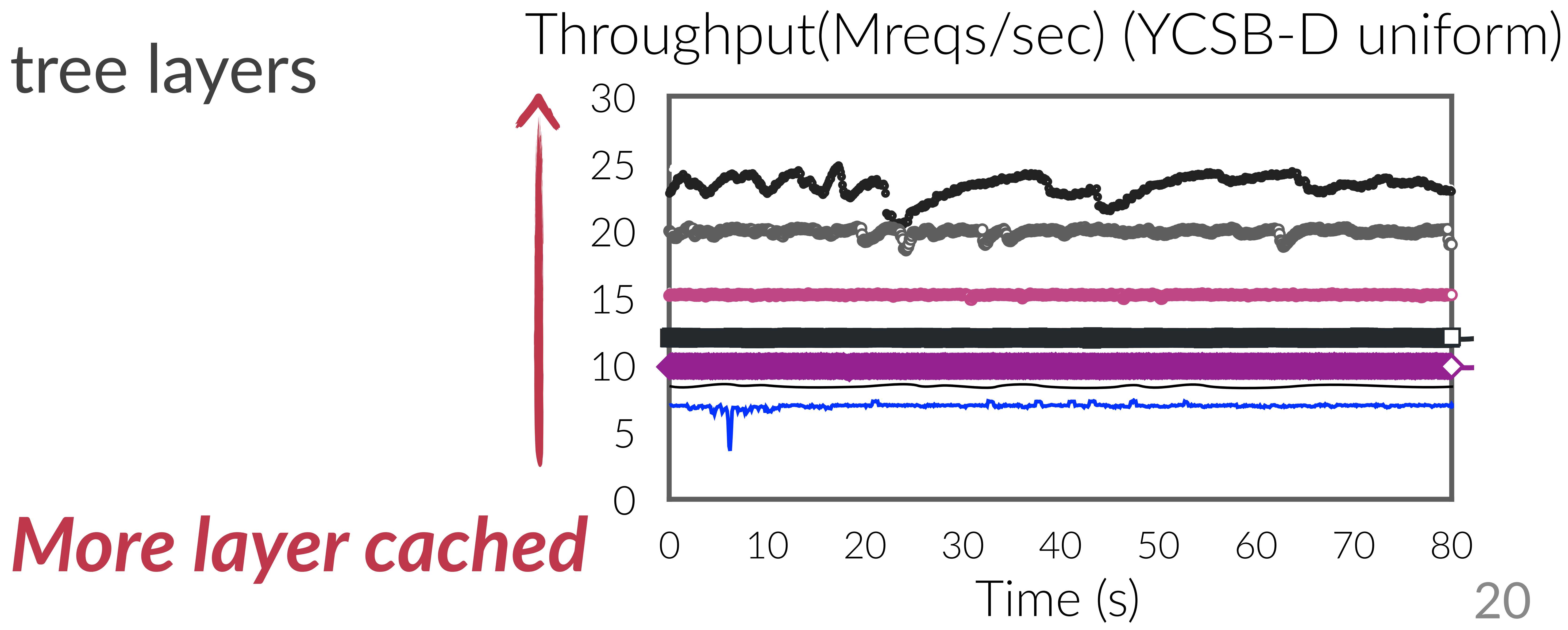
High cache miss cost for caching tree

Tree node size can be ***much larger*** than the KV

- e.g., 1K vs. 8B

Recursive invalidation under insertions

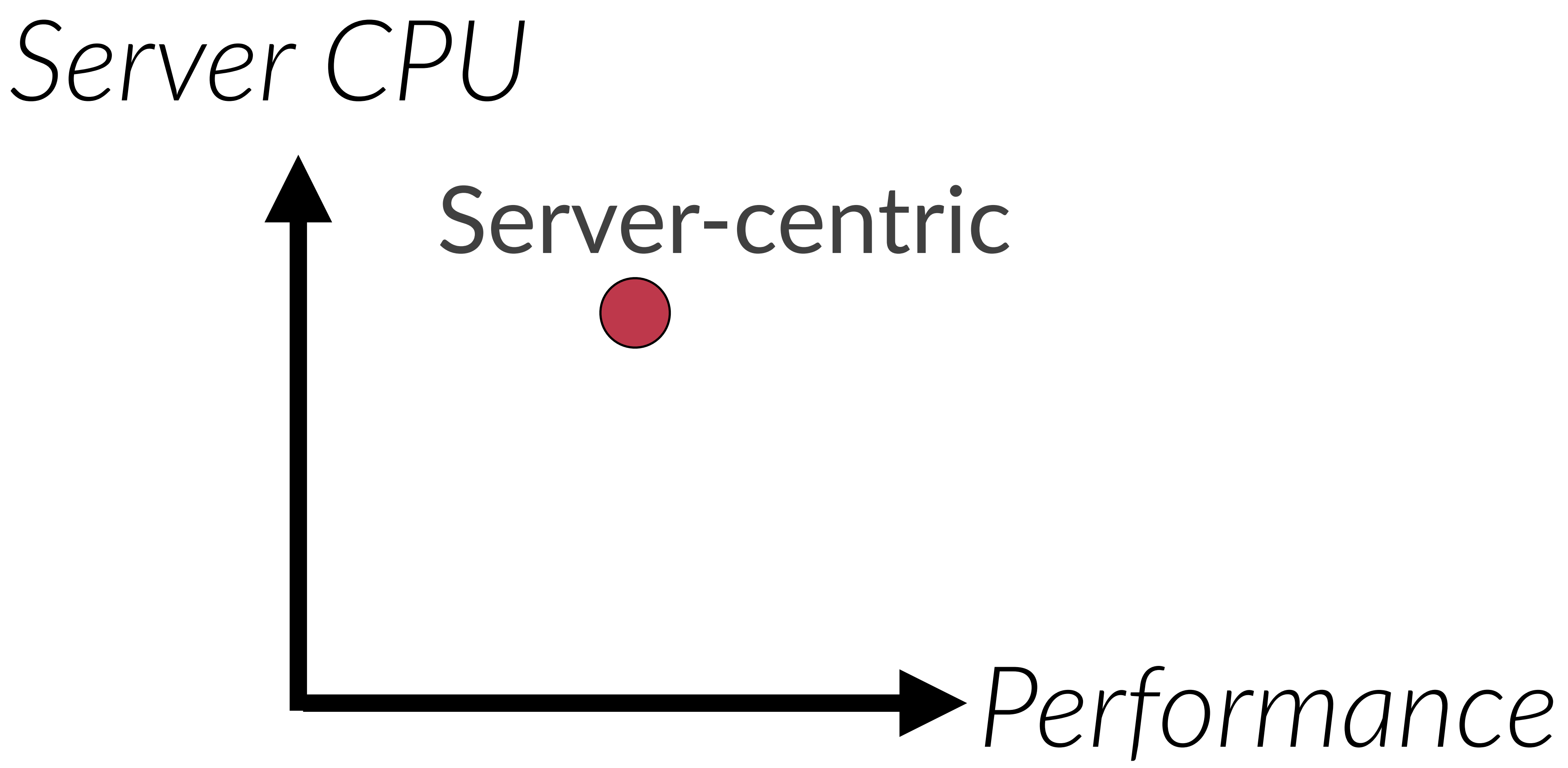
- When cache more tree layers



Trade-off of existing KVS

Server-centric KVS

- High CPU utilizations



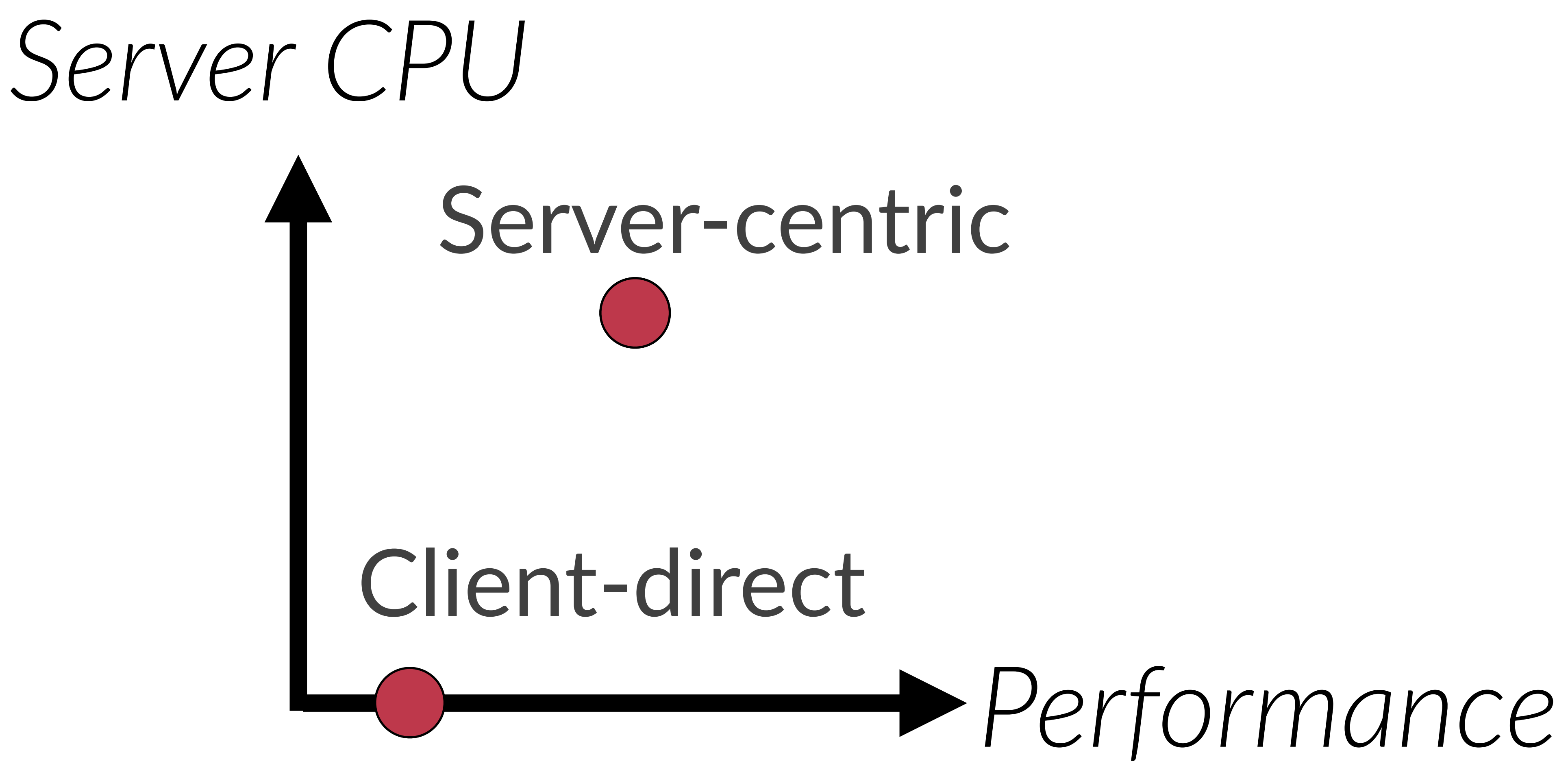
Trade-off of existing KVS

Server-centric KVS

- 𝄢: High CPU utilizations

Client-direct KVS

- 𝄢: Poor performance



Trade-off of existing KVS

Server-centric KVS

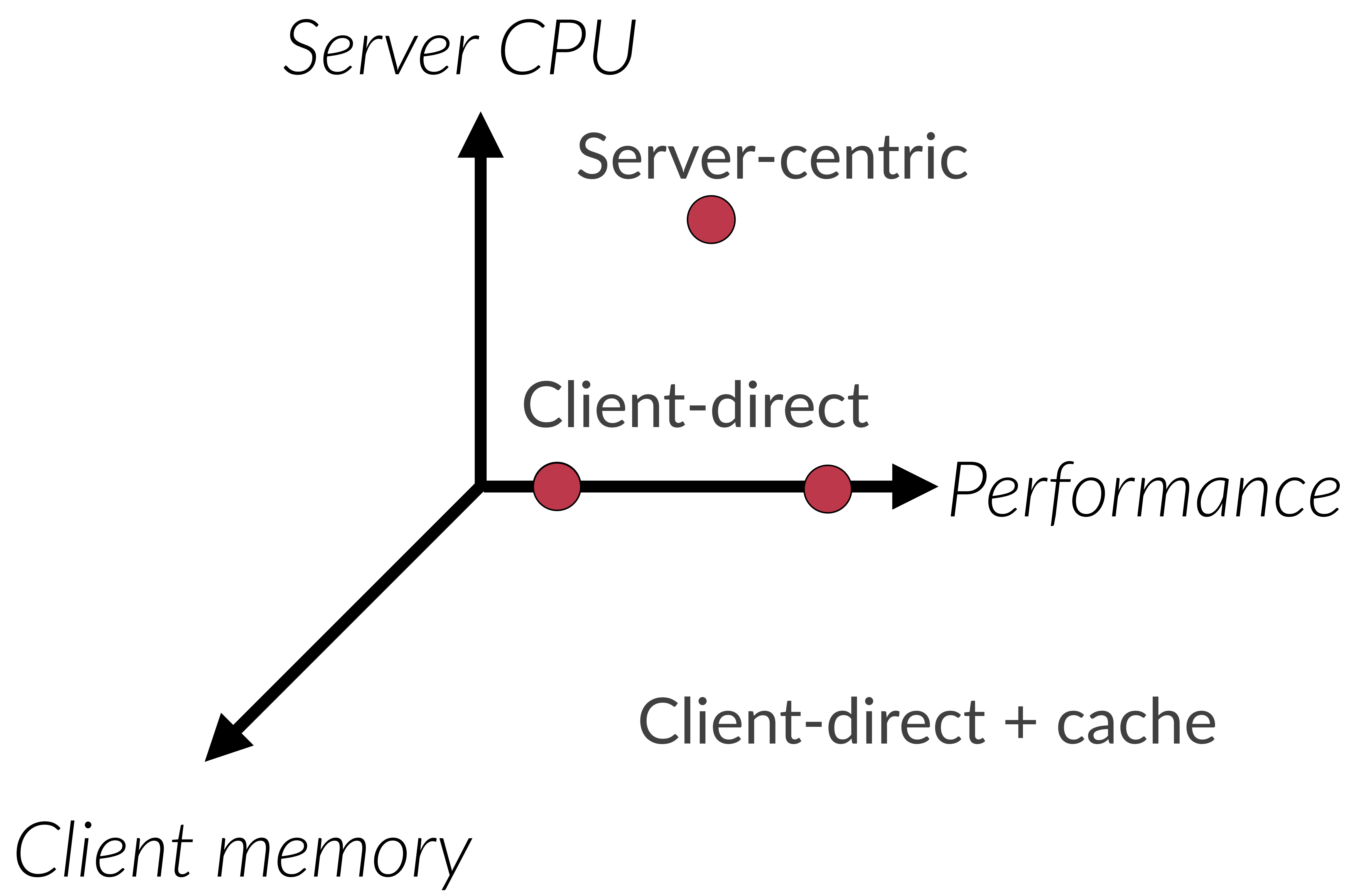
- 𝄢 High CPU utilizations

Client-direct KVS

- 𝄢 Poor performance

Client-direct KVS + cache

- 𝄢 High memory usage



Trade-off of existing KVS

Server-centric KVS

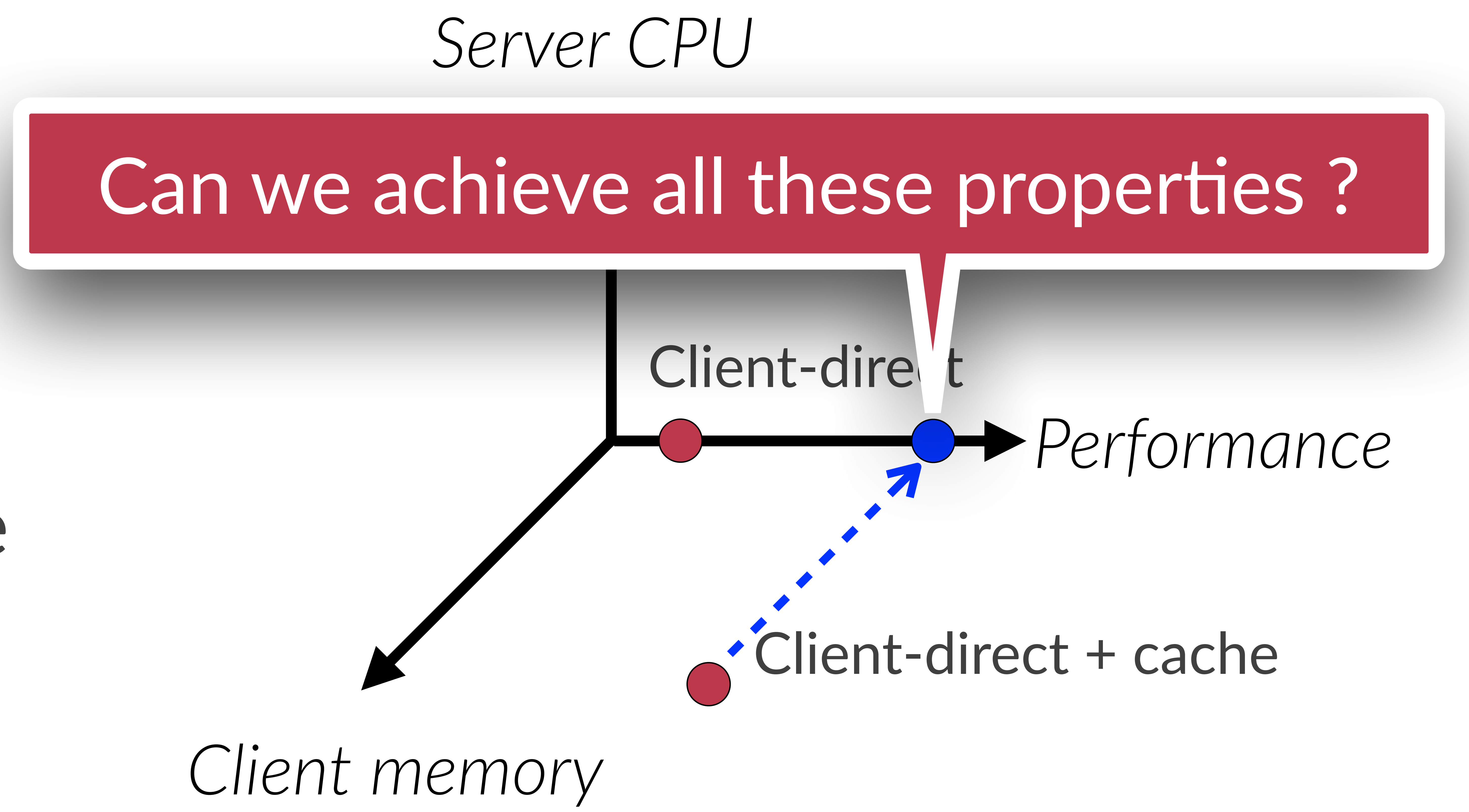
- 𝄢: High CPU utilizations

Client-direct KVS

- 𝄢: Poor performance

Client-direct KVS + cache

- 𝄢: High memory usage



Trade-off of existing KVS

Server-centric KVS

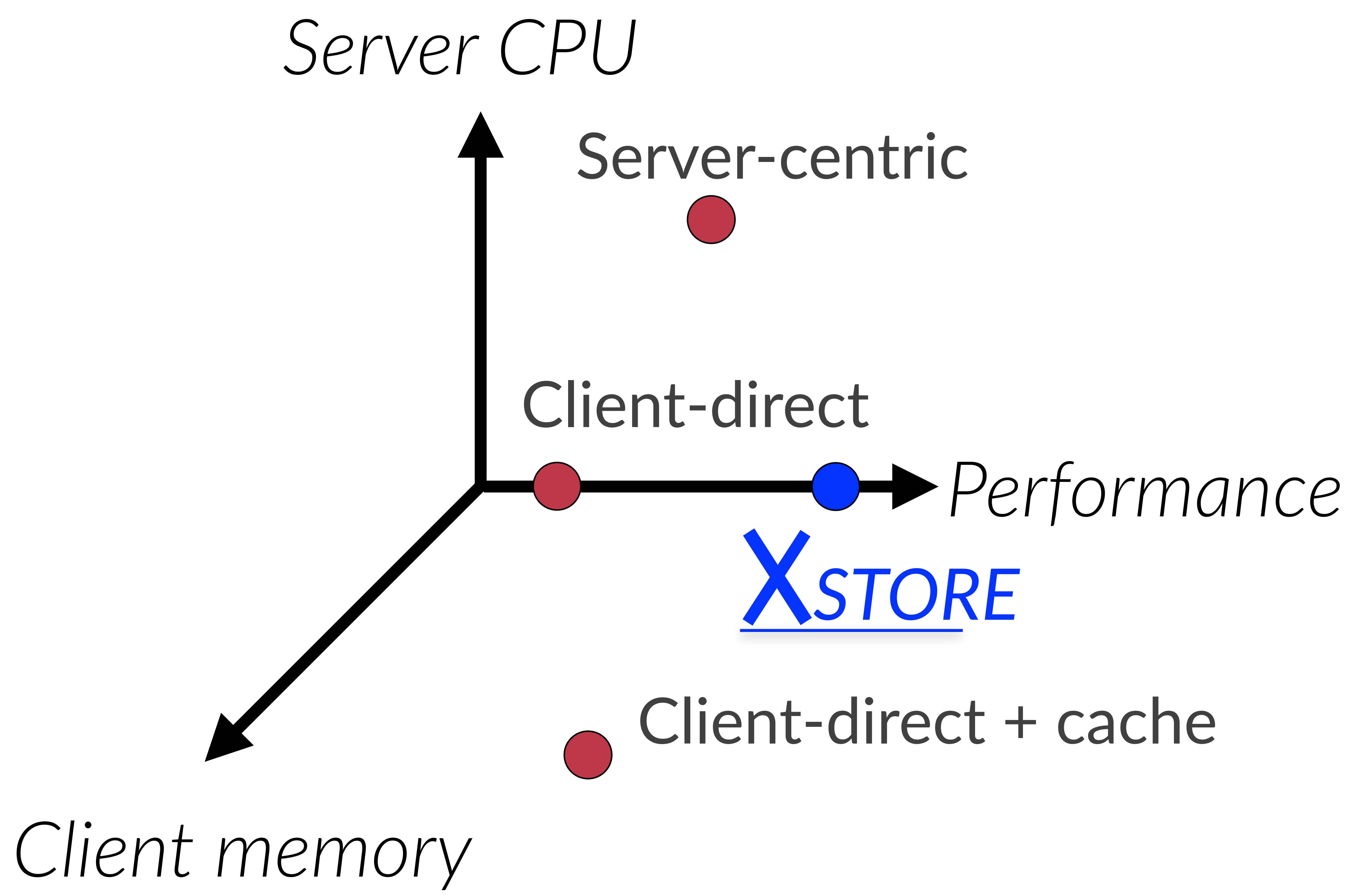
- High CPU utilizations

Client-direct KVS

- Poor performance

Client-direct KVS + cache

- High memory usage

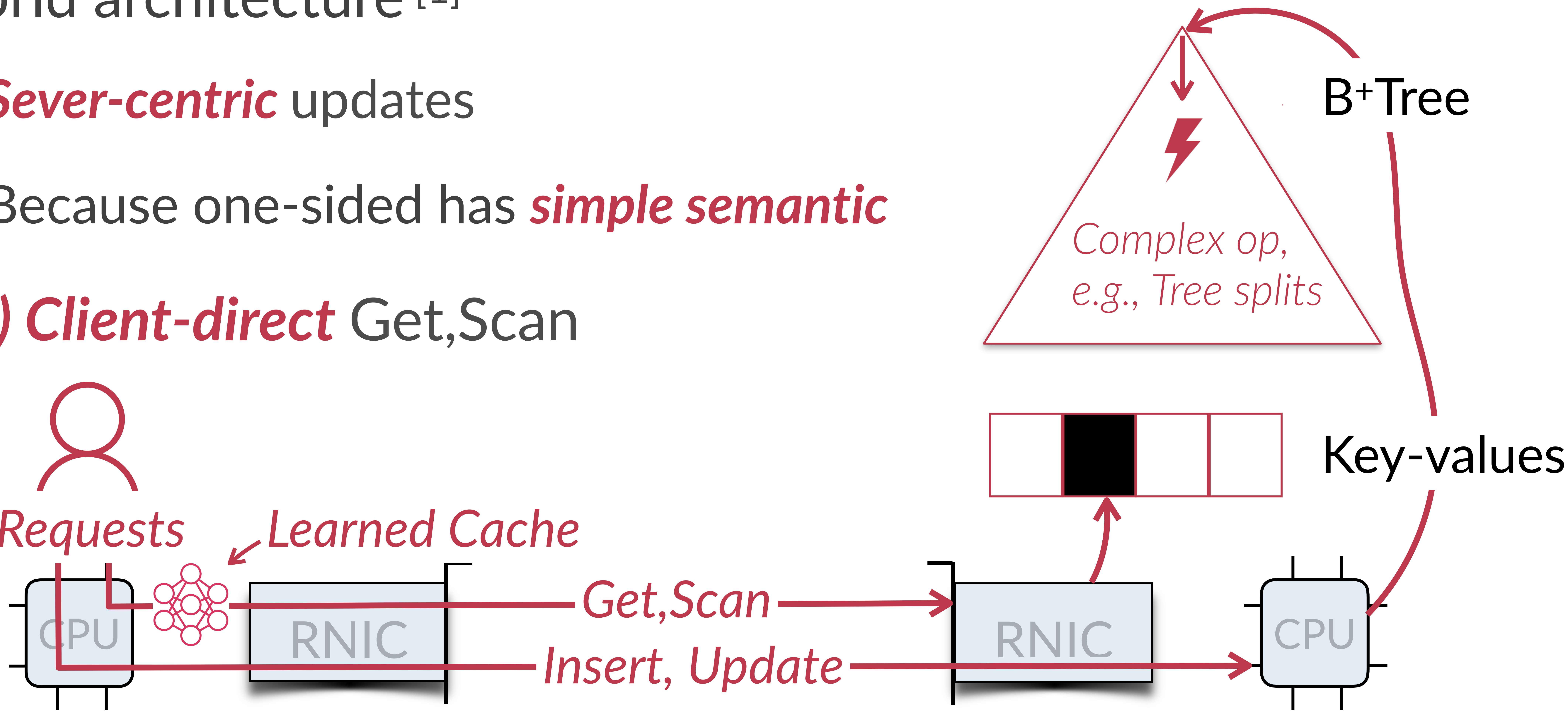


Overview of XSTORE

Hybrid architecture [1]

- ***Sever-centric*** updates
- Because one-sided has ***simple semantic***

O(1) Client-direct Get,Scan



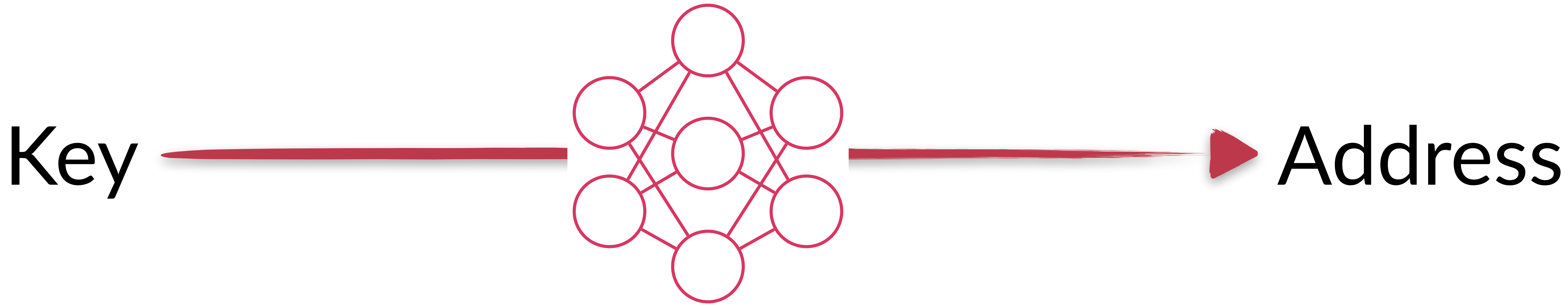
[1] Similar to existing RDMA-based KVS, e.g., FaRM@SOSP'15, Cell@ATC'16

Our approach: Learned cache

Using ***ML*** as the cache structure for tree-based index

Motivated by the ***learned index***[1]

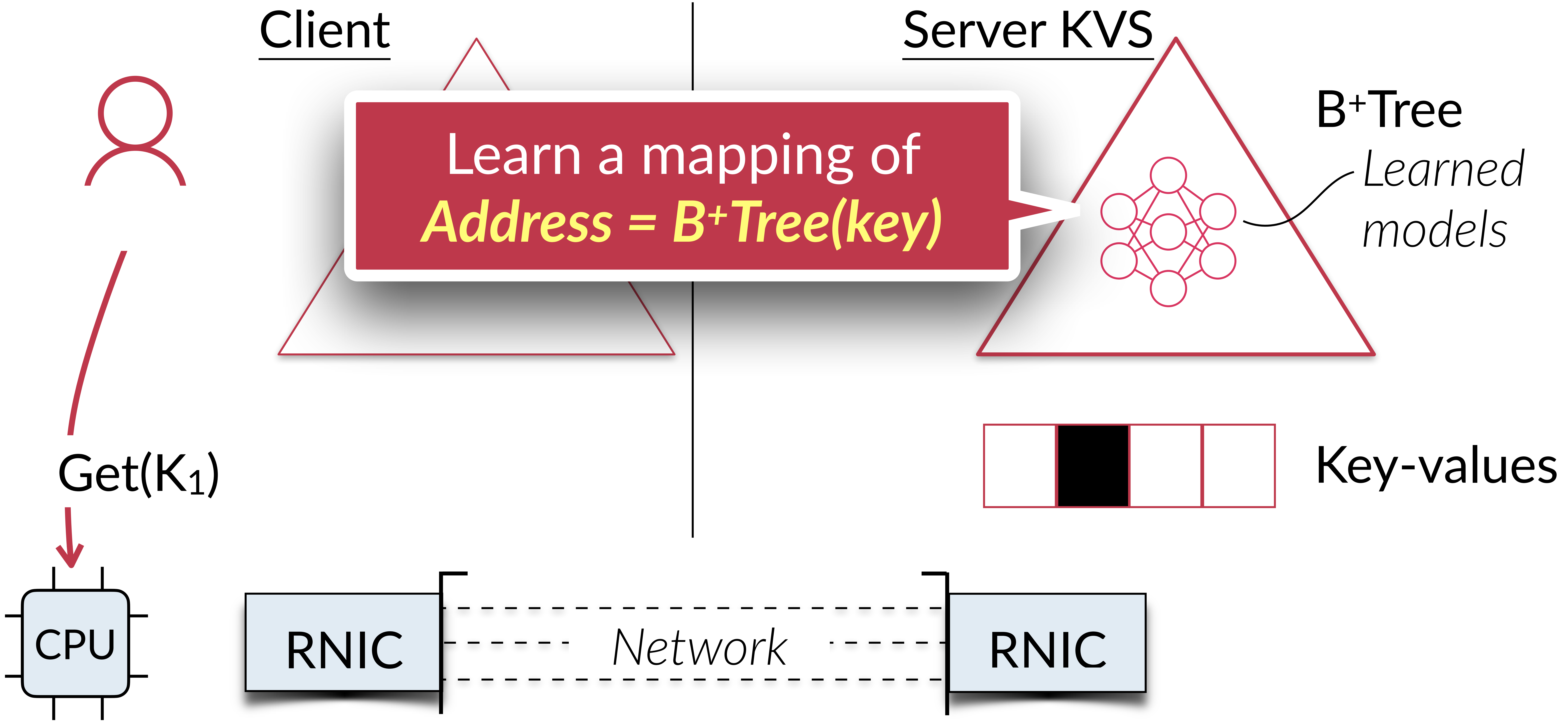
- Replace ***index traversal*** with ***calculation***
- The ML model can be ***orders of magnitude smaller*** than tree



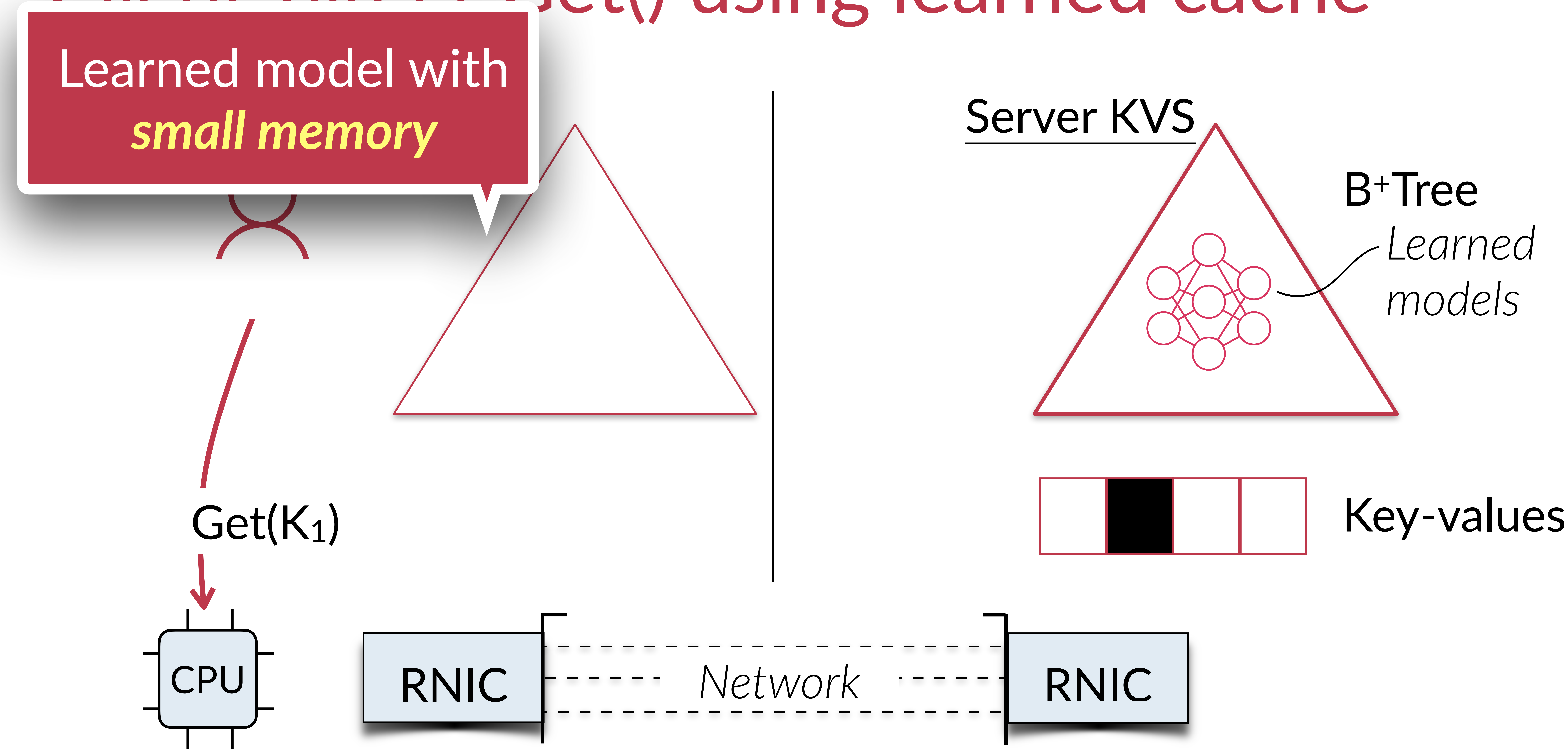
Machine Learning (ML) models

[1] The case for the learned index @ SIGMOD'18

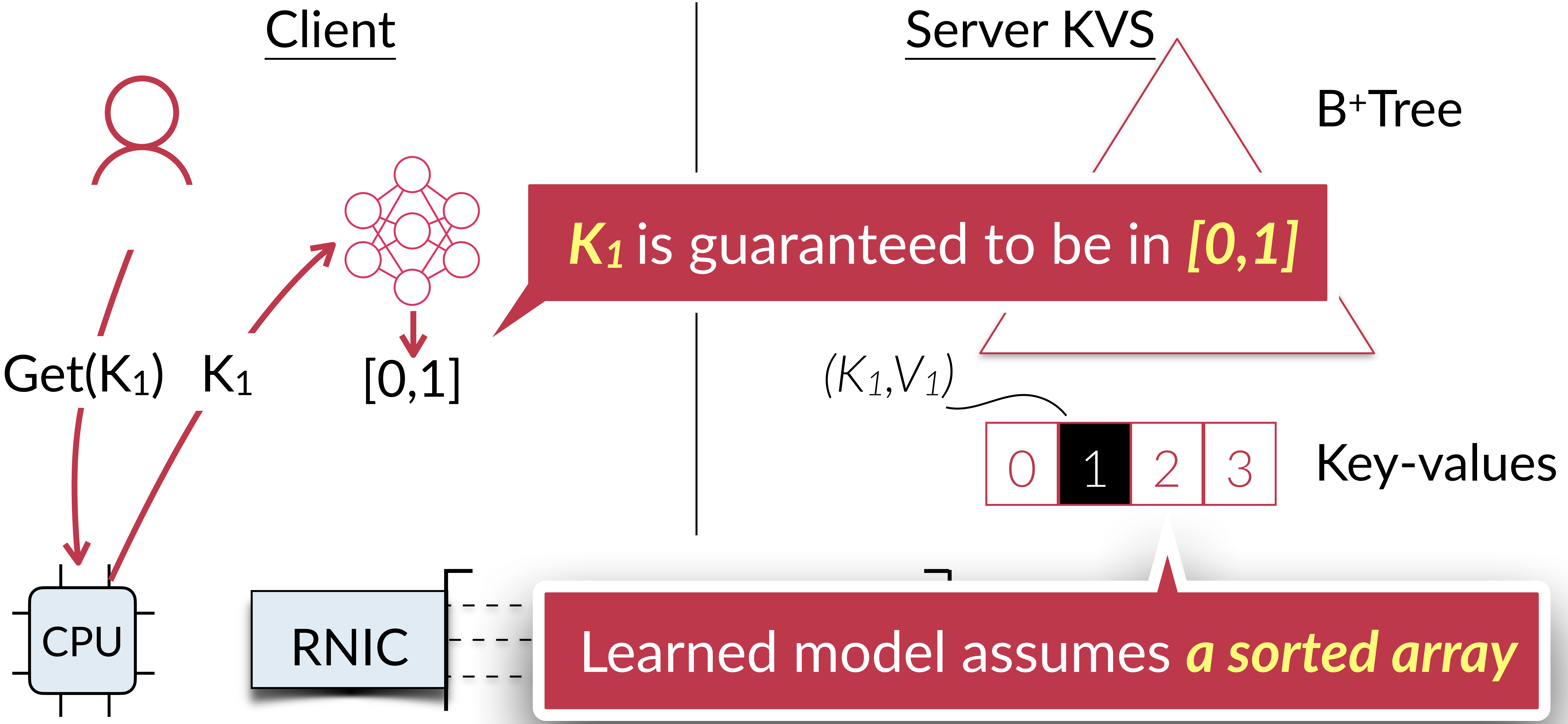
Client-direct Get() using learned cache



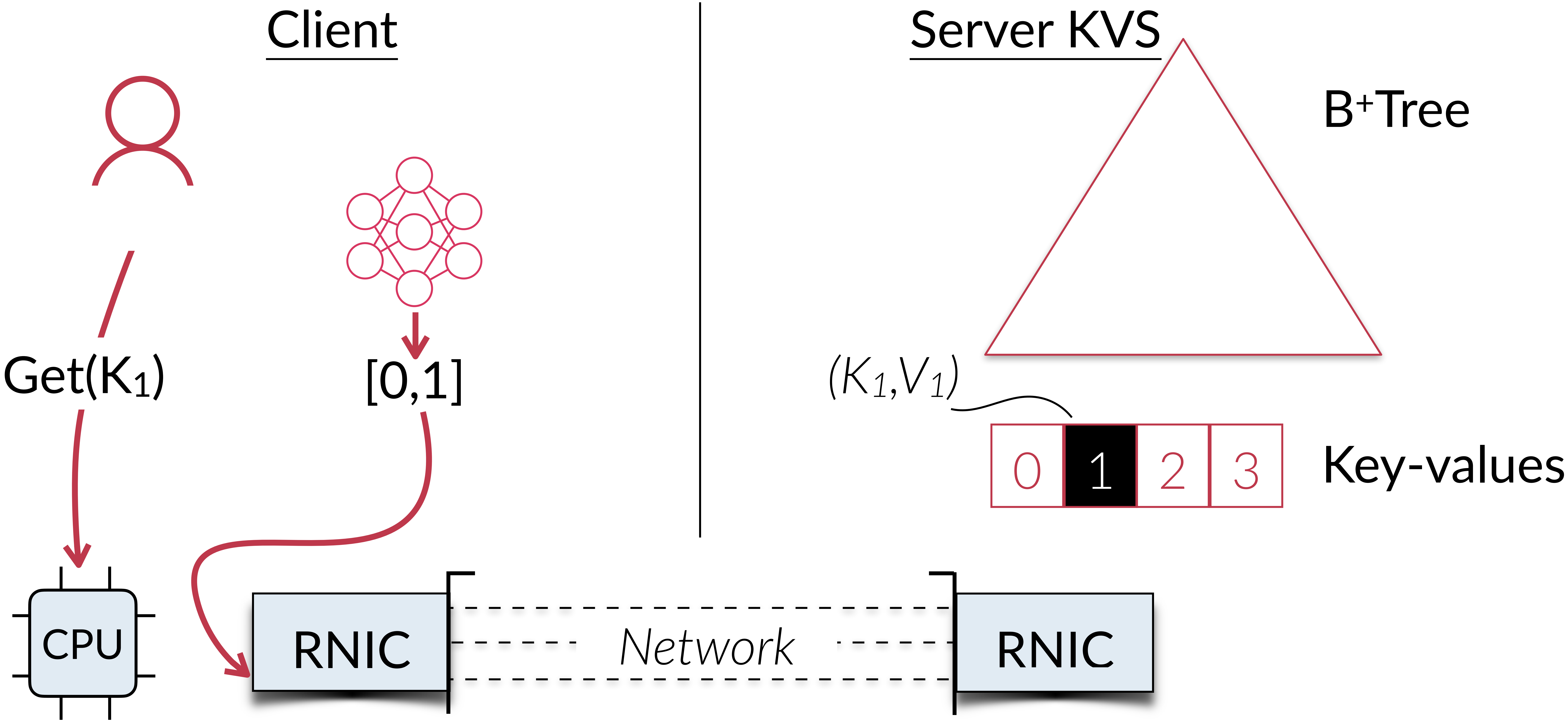
Client-direct Get() using learned cache



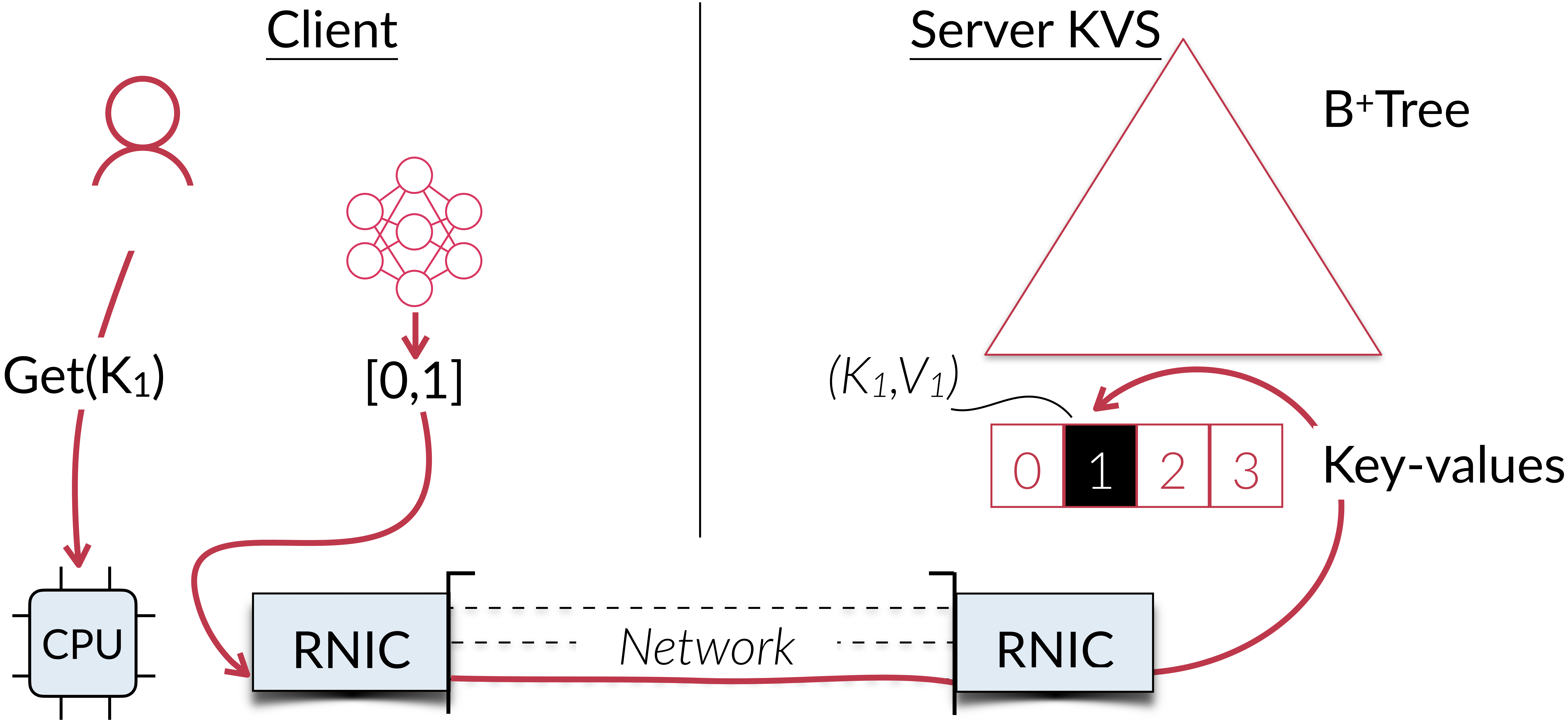
Client-direct Get() using learned cache



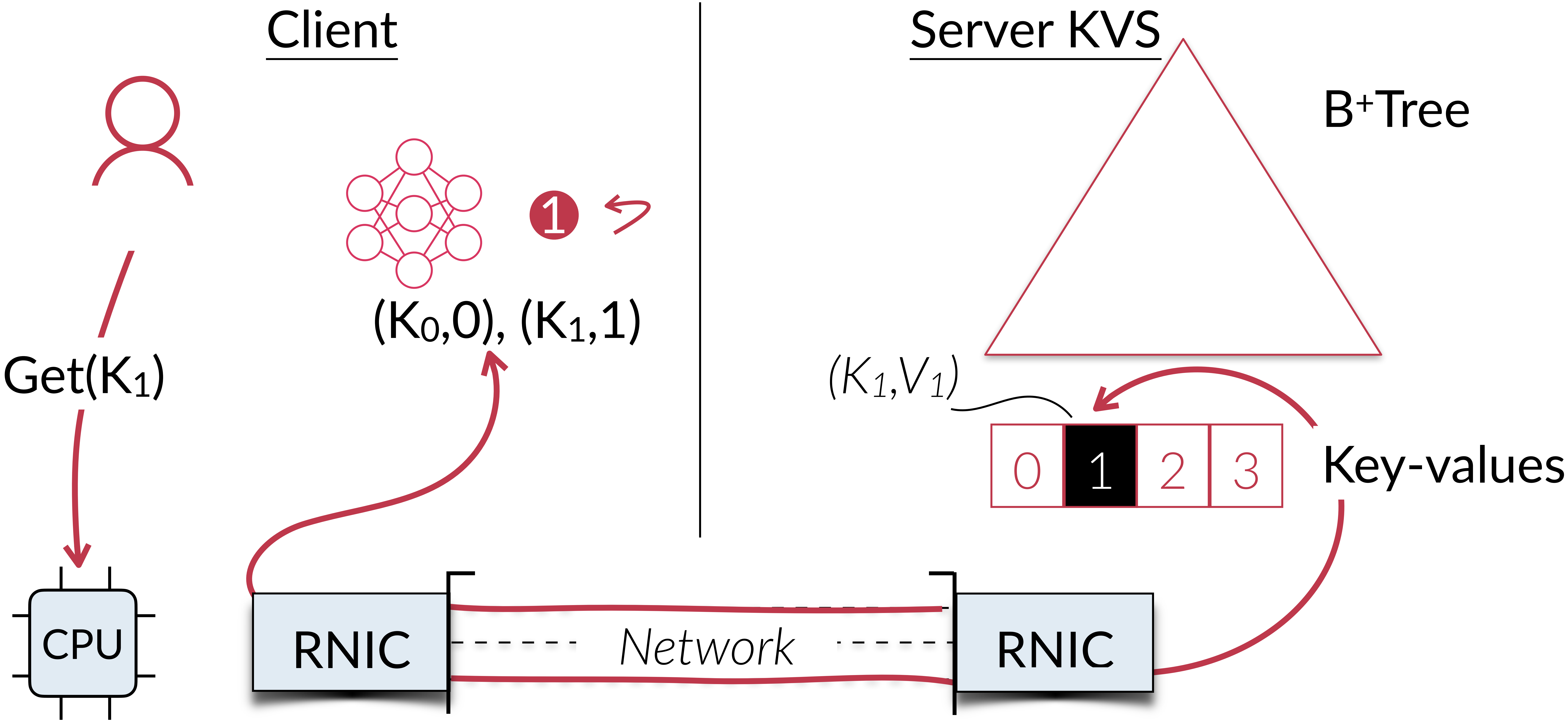
Client-direct Get() using learned cache



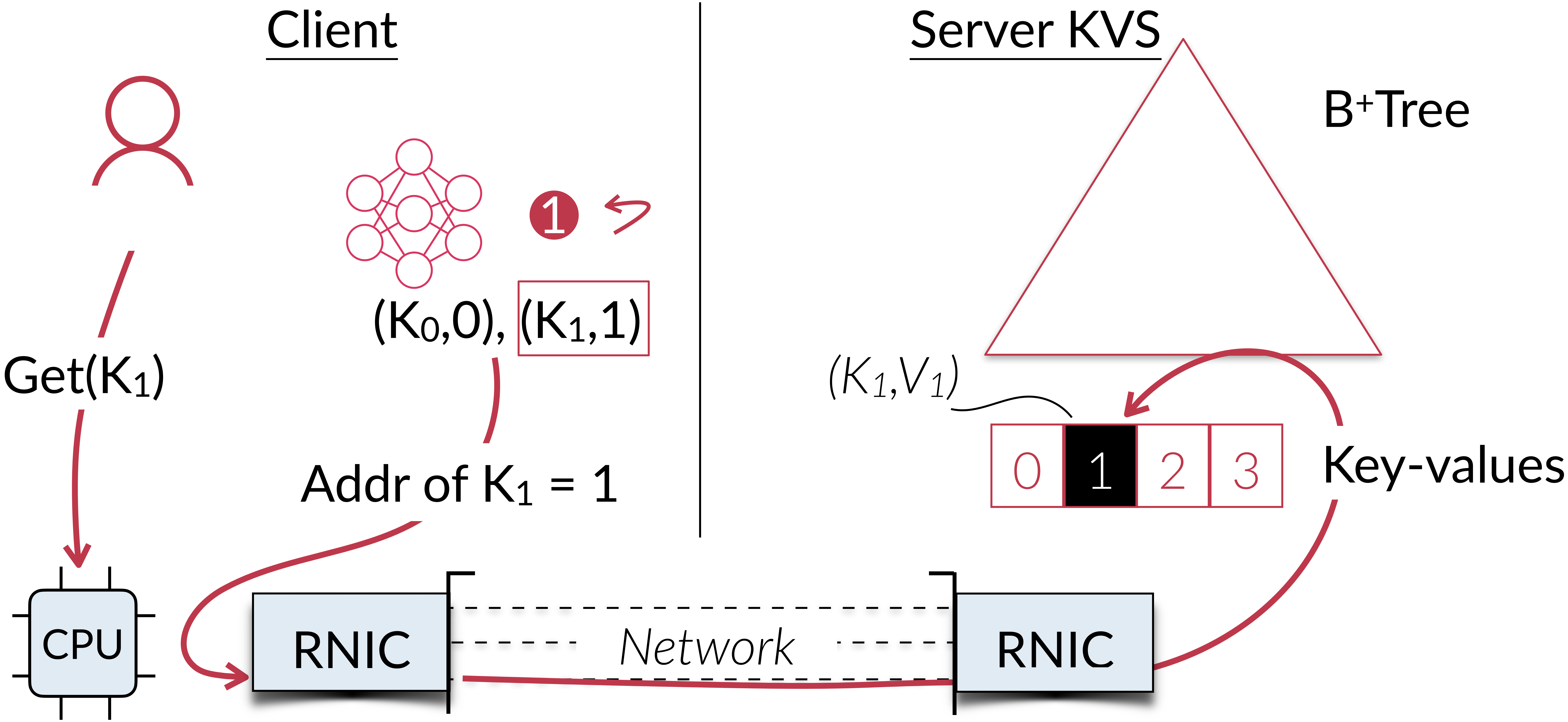
Client-direct Get() using learned cache



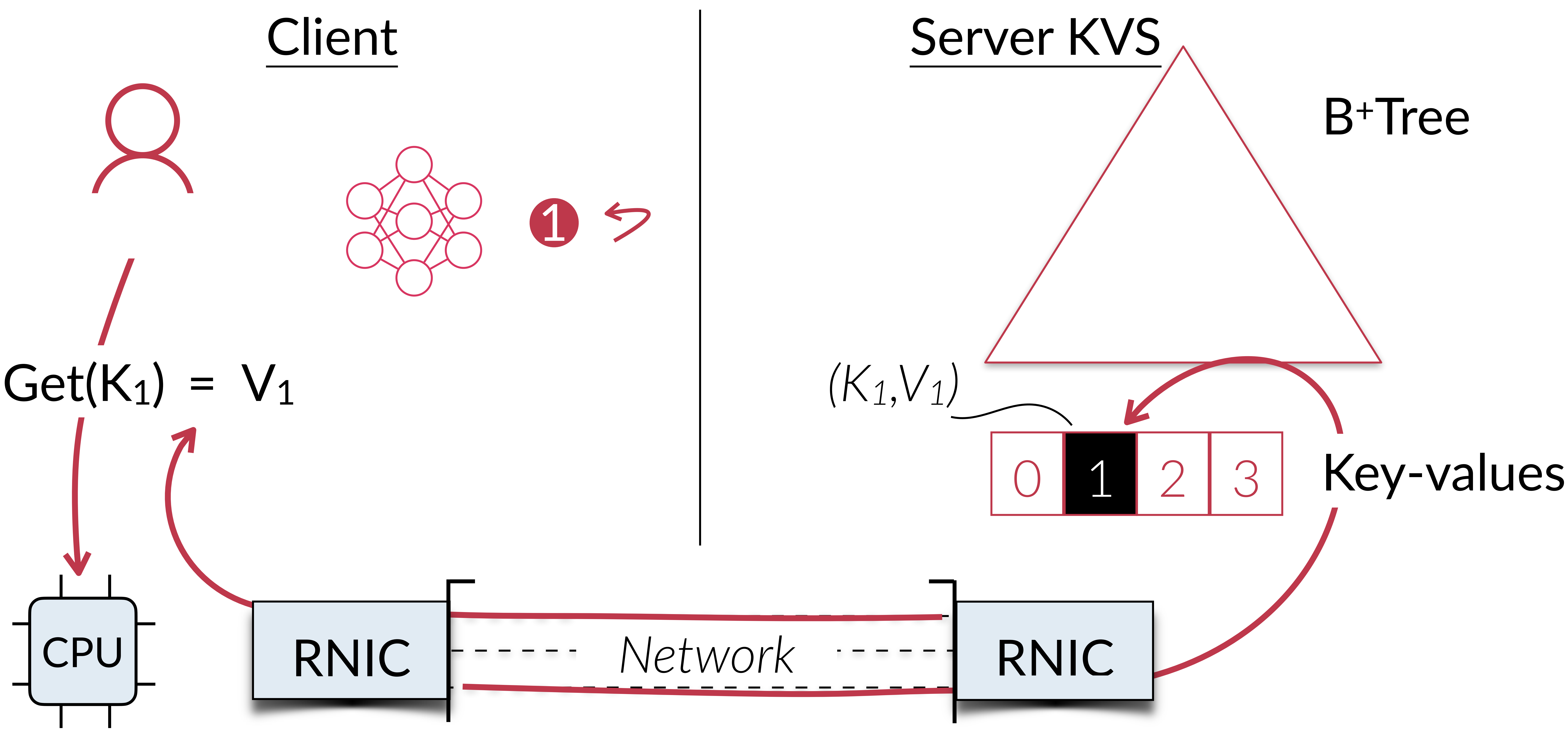
Client-direct Get() using learned cache



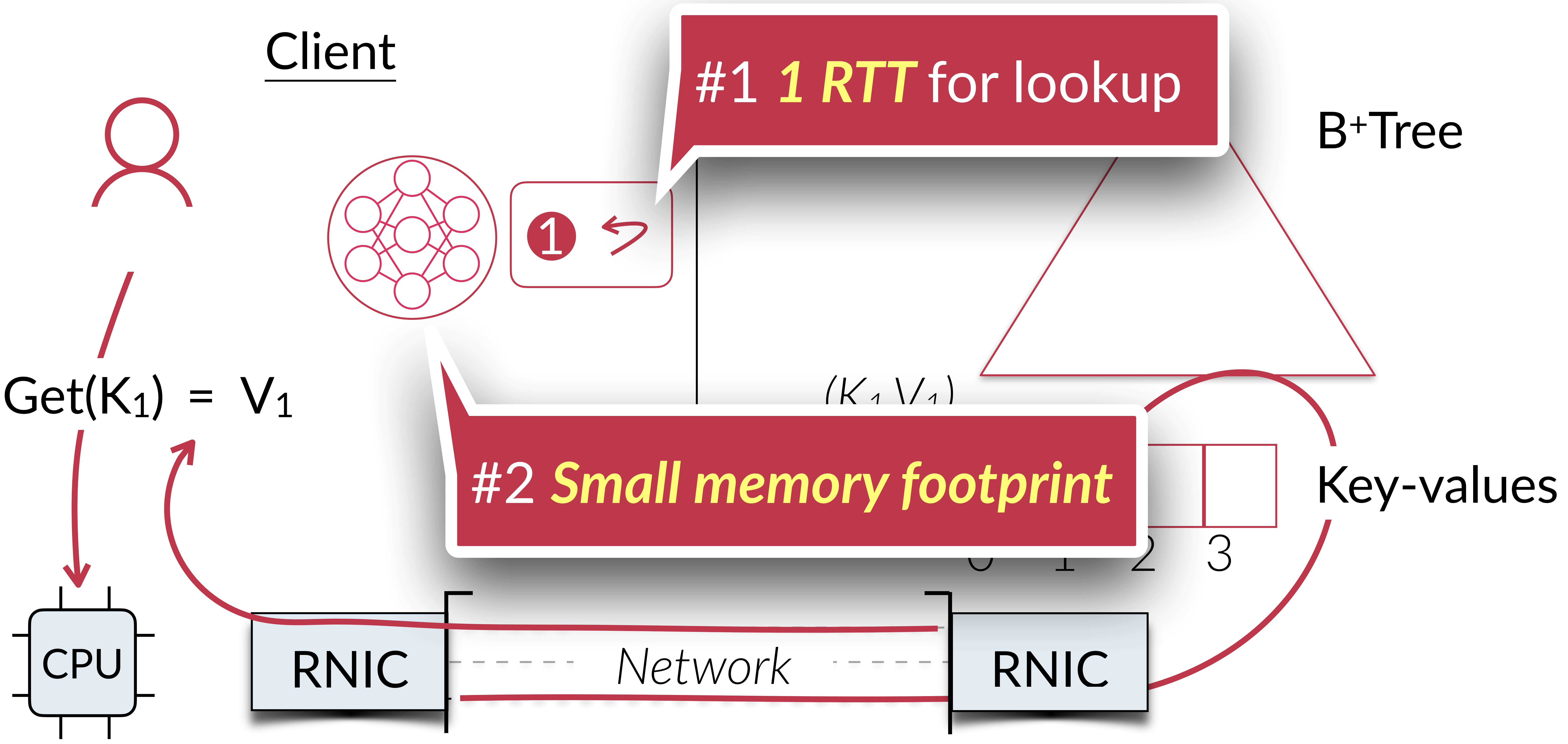
Client-direct Get() using learned cache



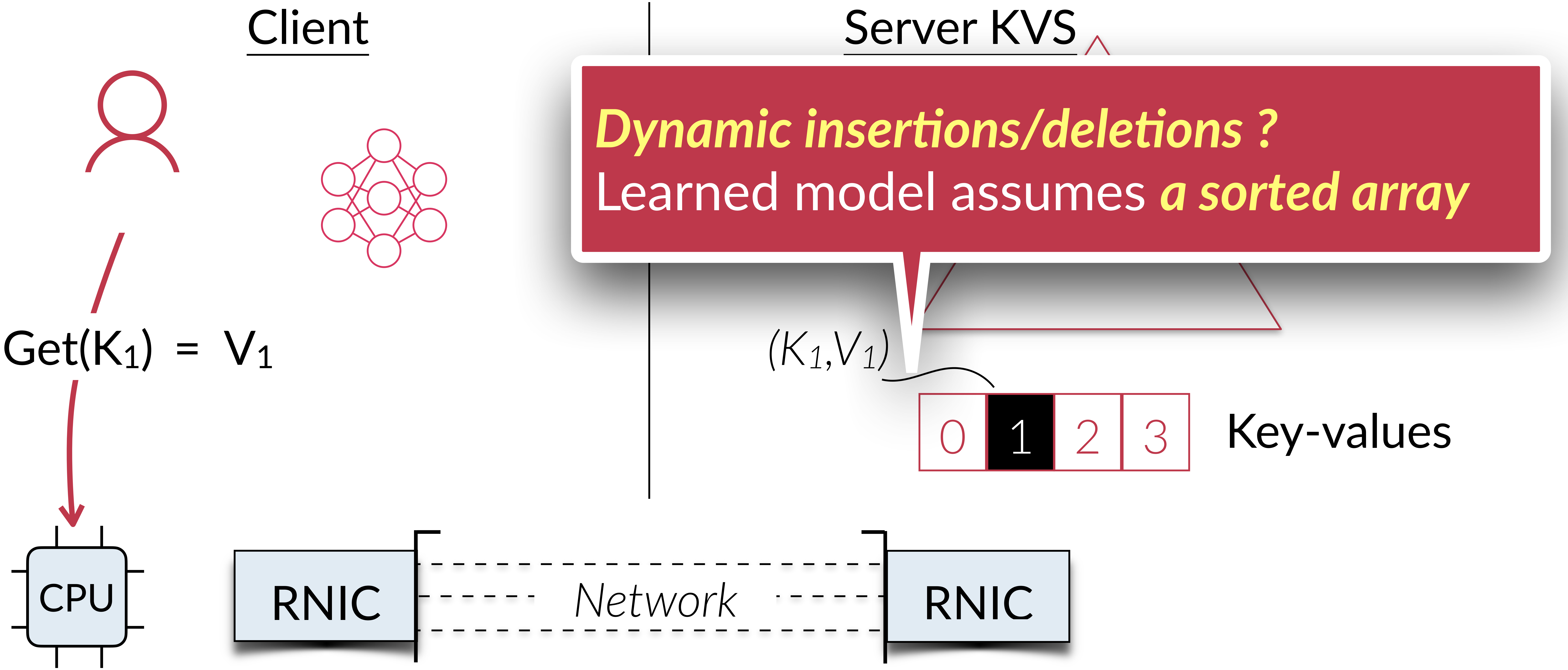
Client-direct Get() using learned cache



Benefits of the learned cache



Challenges of learned cache



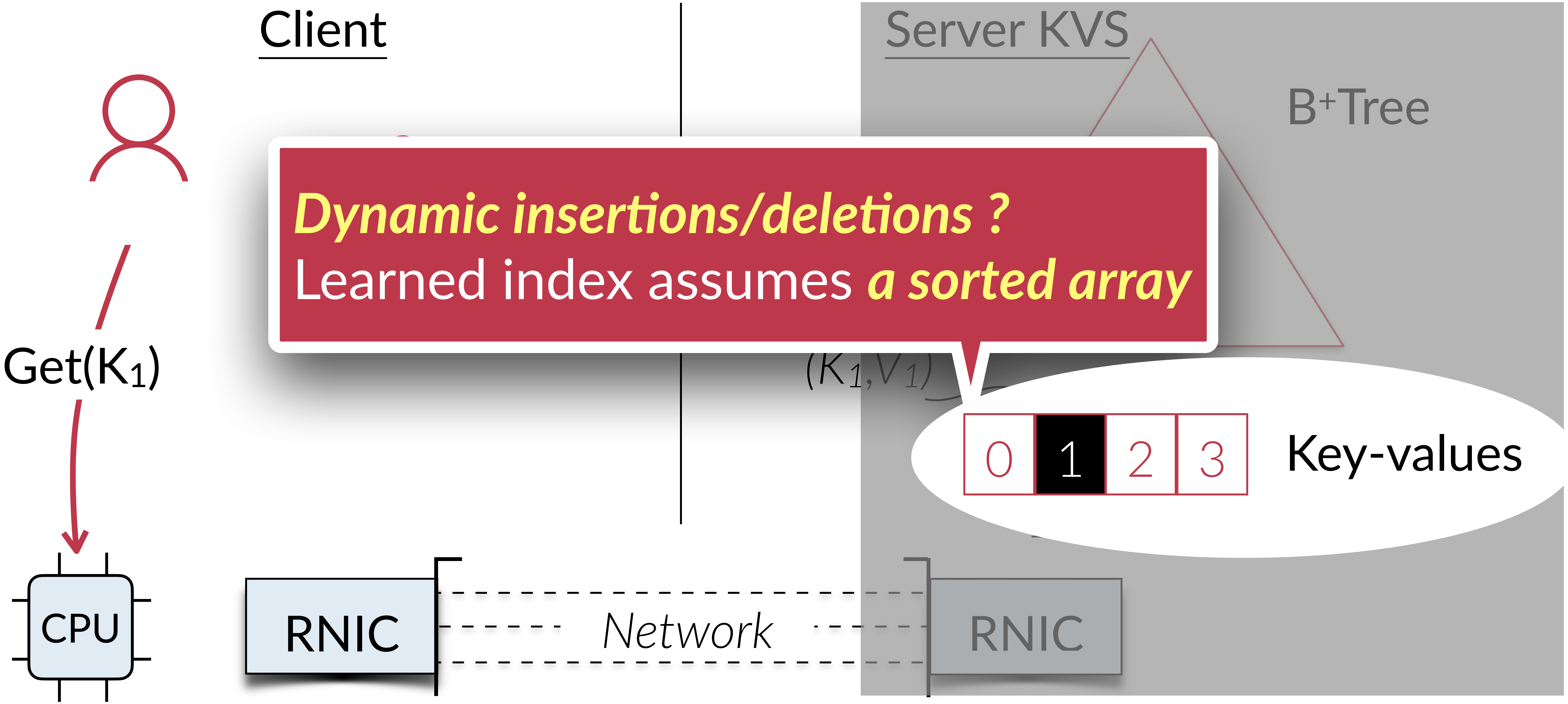
Outline of the remaining content

Server-side data structure for dynamic workloads

Client-side learned cache & TT

Performance evaluation of XSTORE

XSTORE stores KVs in B^+Tree leaf nodes



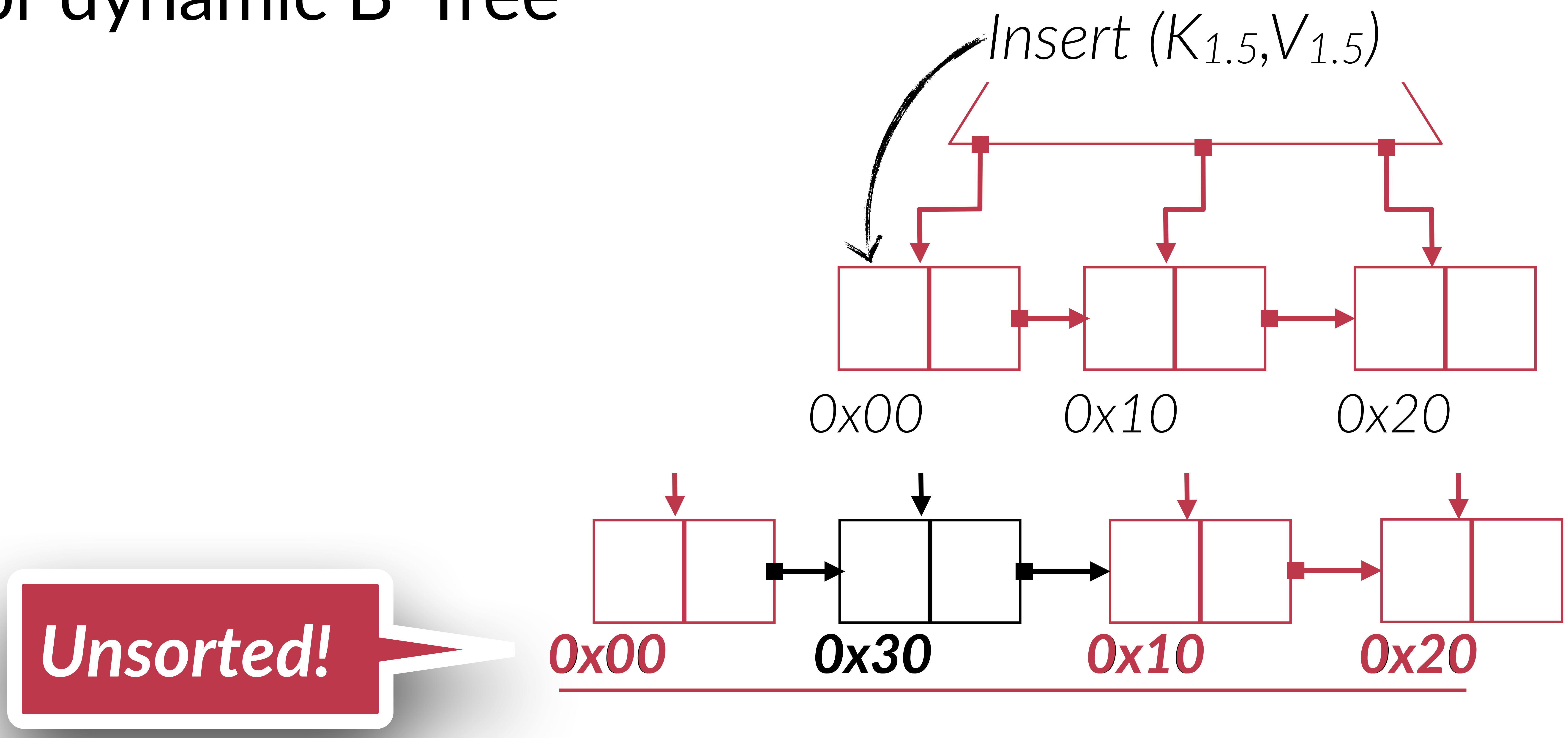
Dynamic insertions/deletions ?

Learned index assumes ***a sorted array***

Models cannot learn dynamic B+Tree address

Can only learn when the addresses are ***sorted***

Not the case for dynamic B+Tree



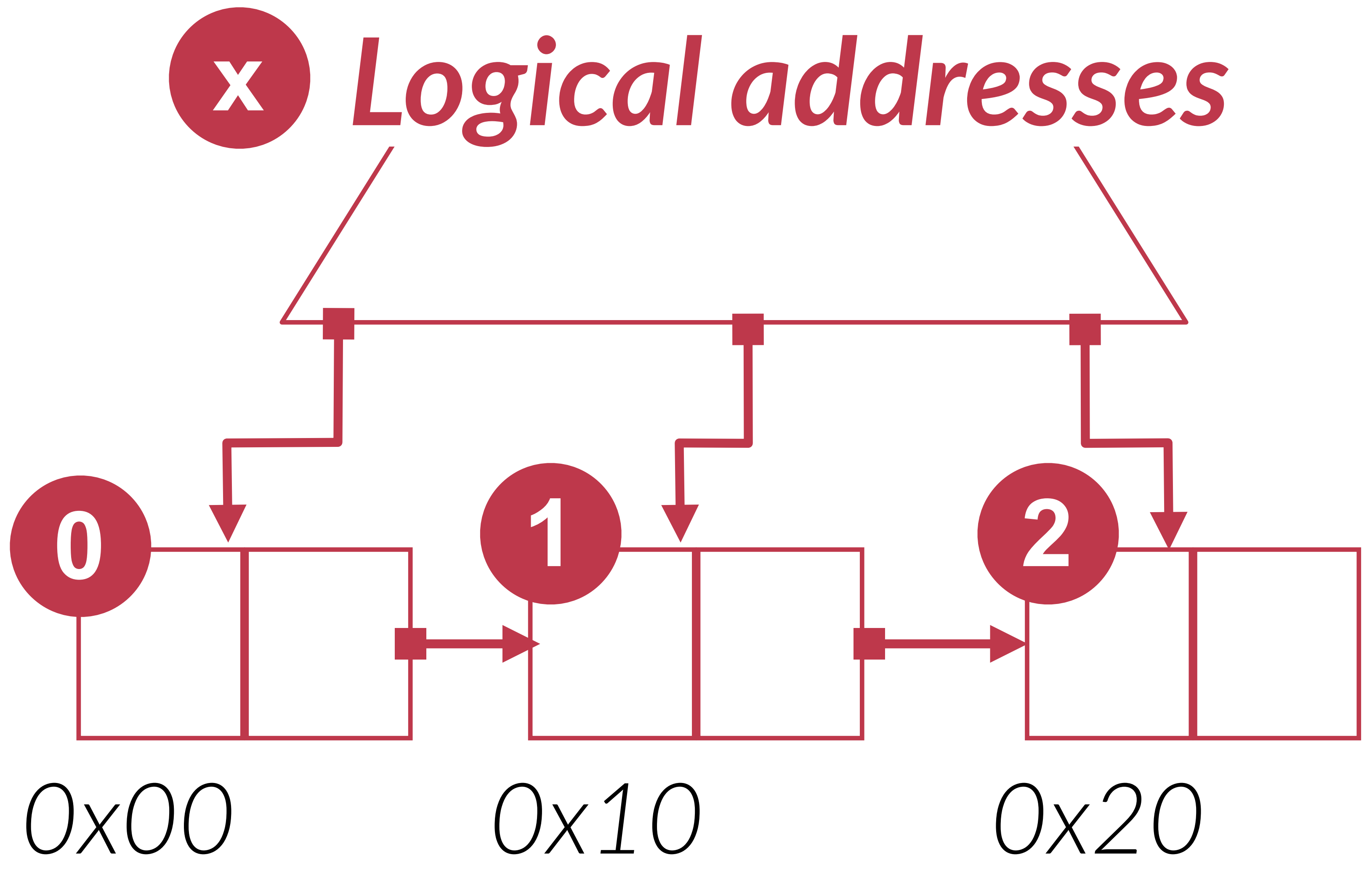
Solution: another layer of indirection

Observation: leaf nodes are logically sorted

- Assign logical addresses to leaf nodes

ML: key → logical

- Translation table (TT): logical → physical



Translation Table

0x00	0x10	0x20
0	1	2

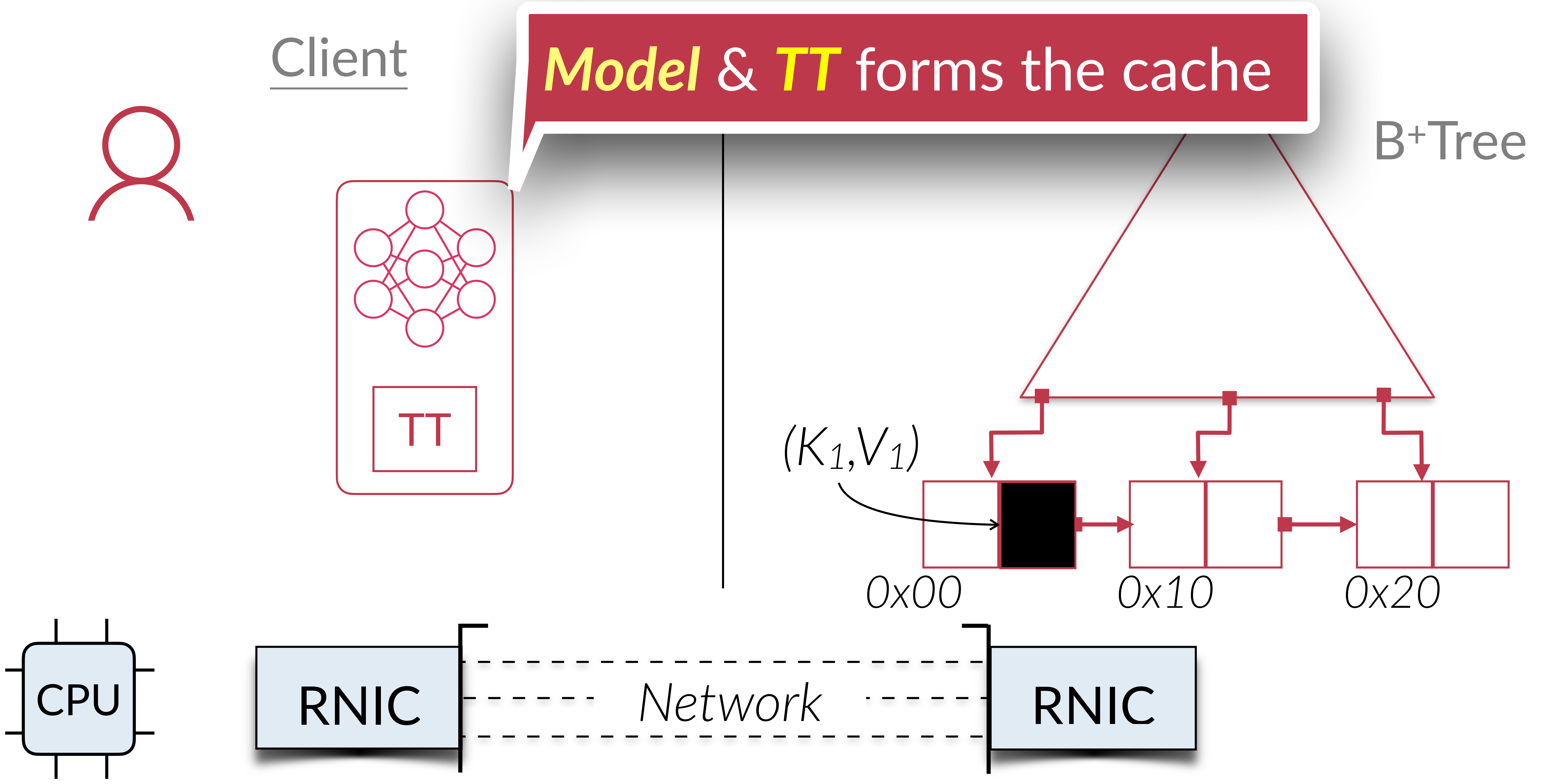
Outline of the remaining content

Server-side data structure for dynamic workloads

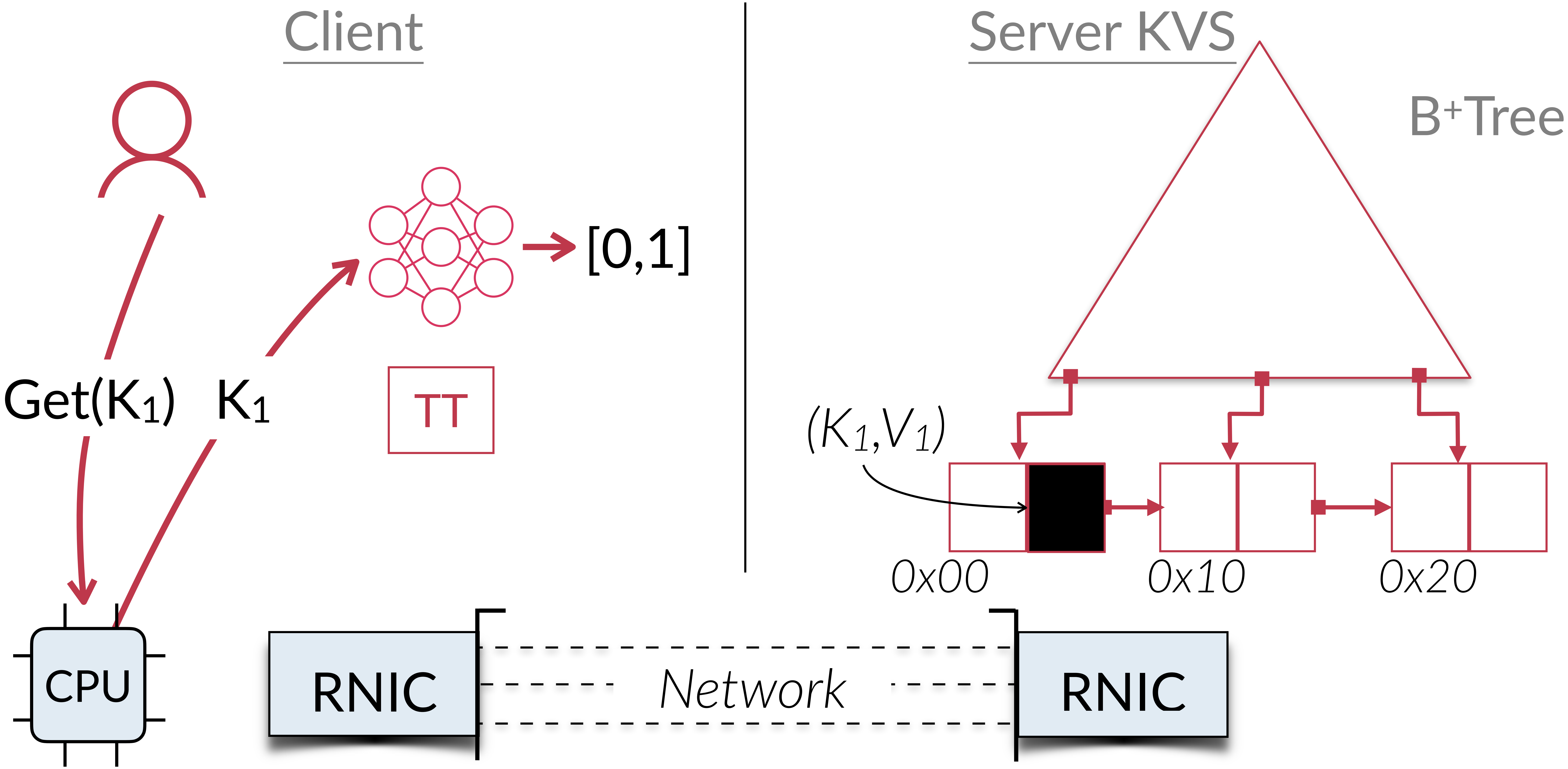
Client-side learned cache & TT

Performance evaluation of XSTORE

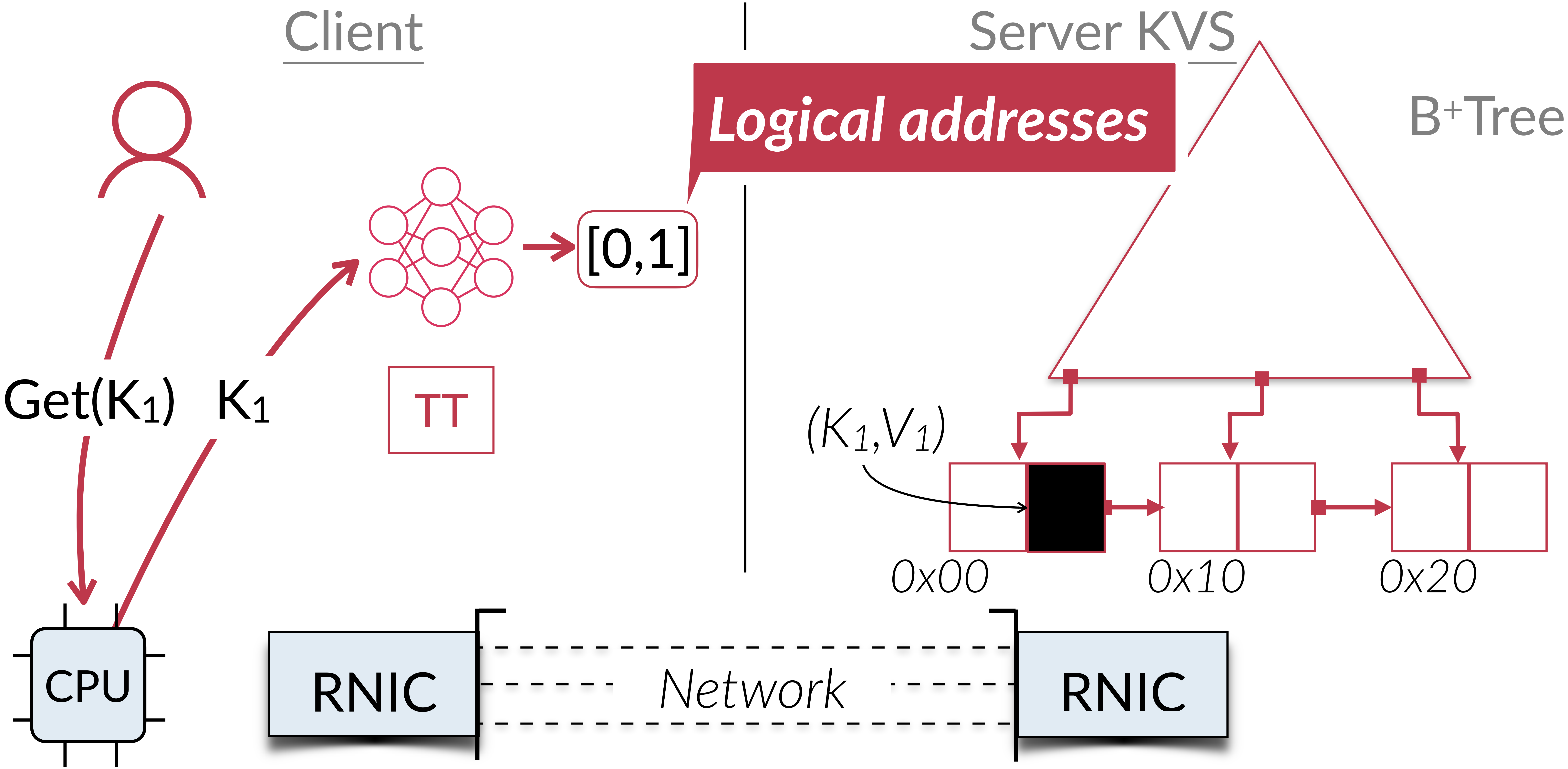
Client-direct Get() using model & TT



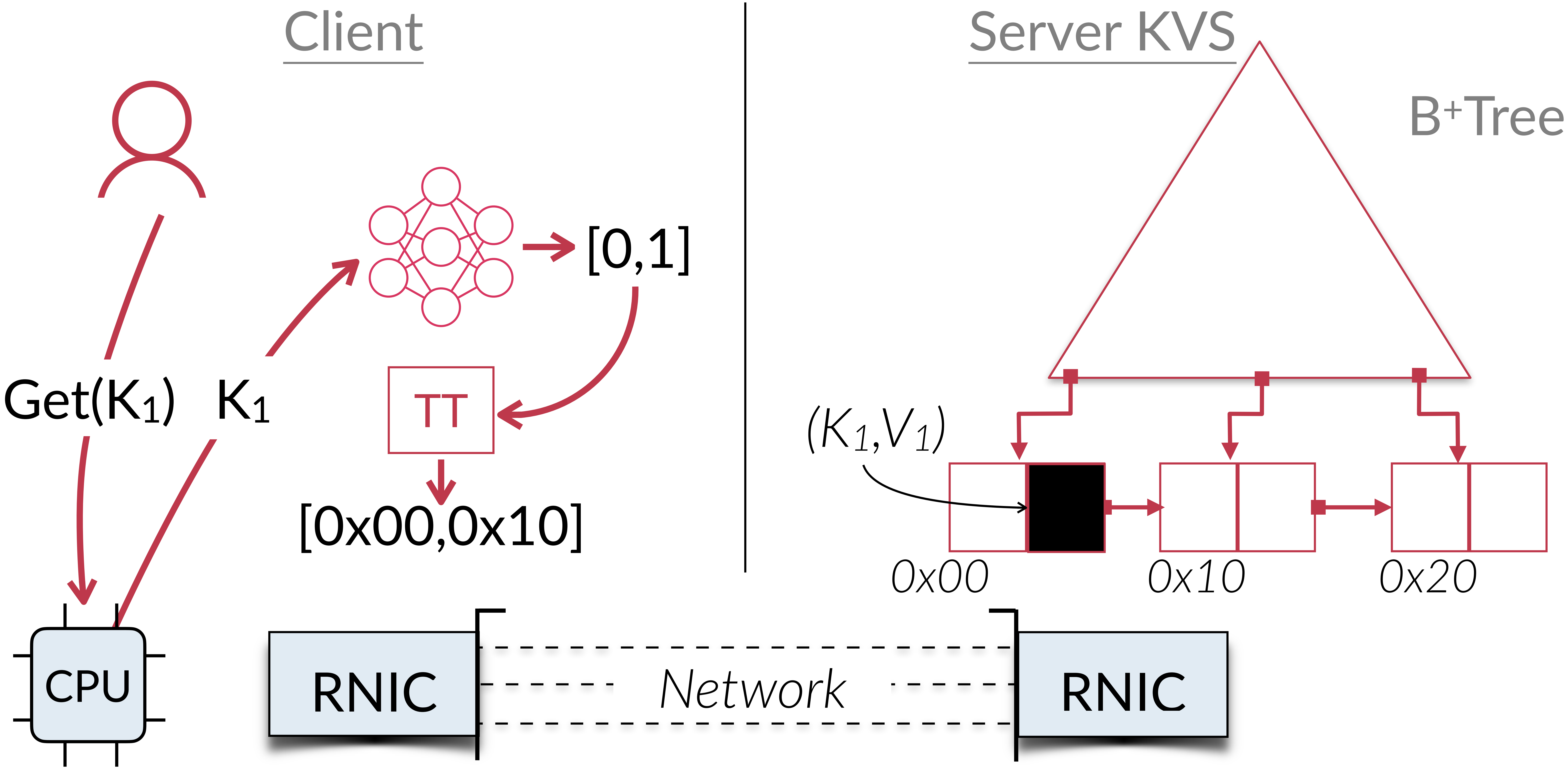
Client-direct Get() using model & TT



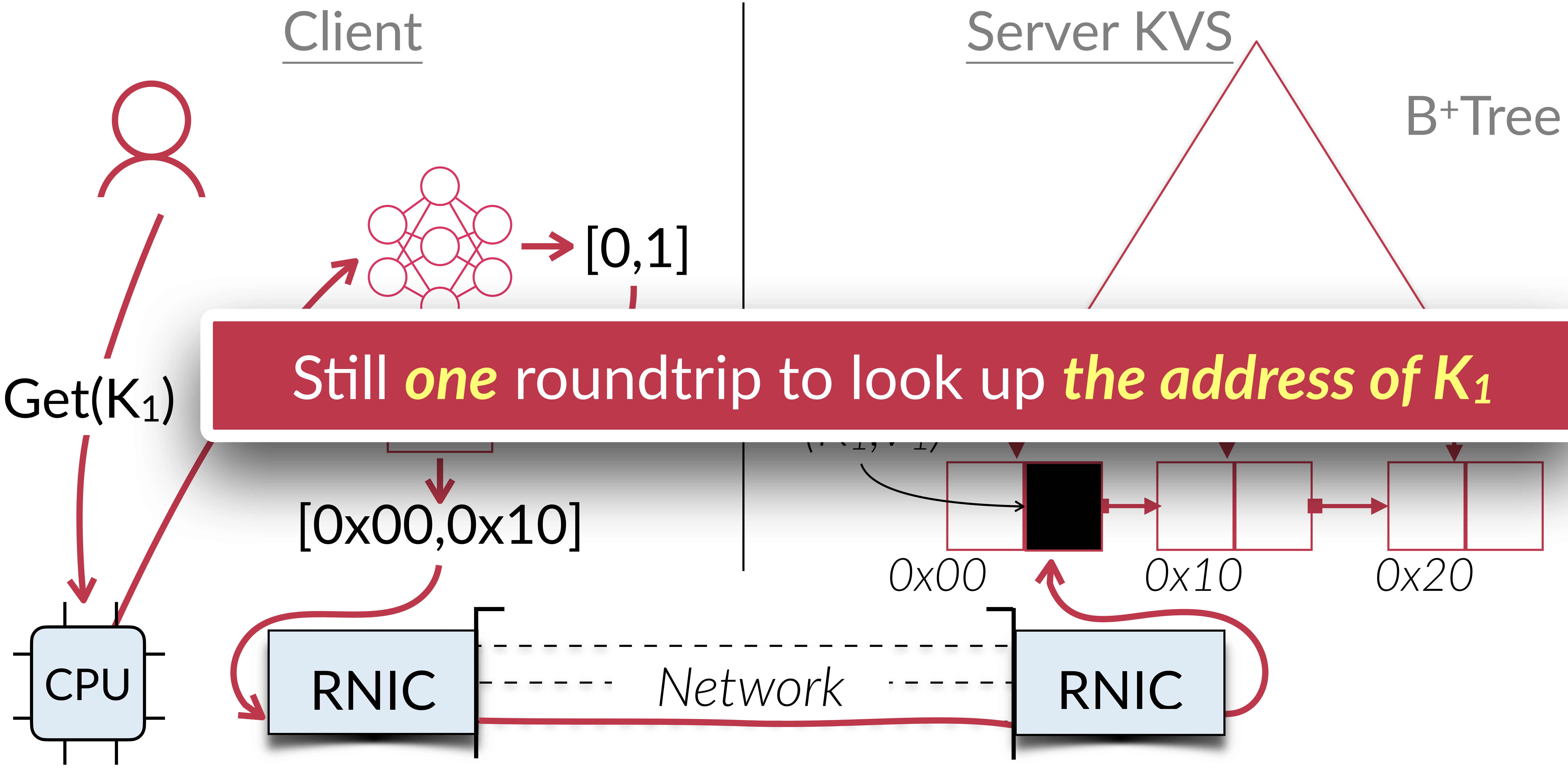
Client-direct Get() using model & TT



Client-direct Get() using model & TT



Client-direct Get() using model & TT

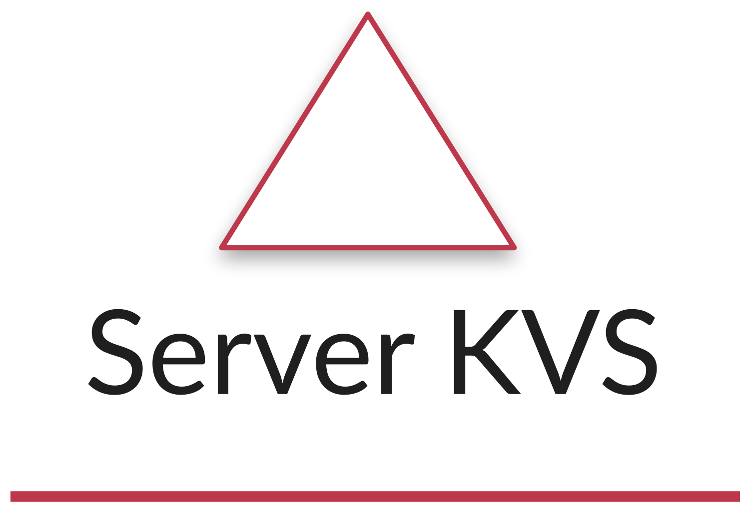


Still *one* roundtrip to look up *the address of K_1*

Model retraining

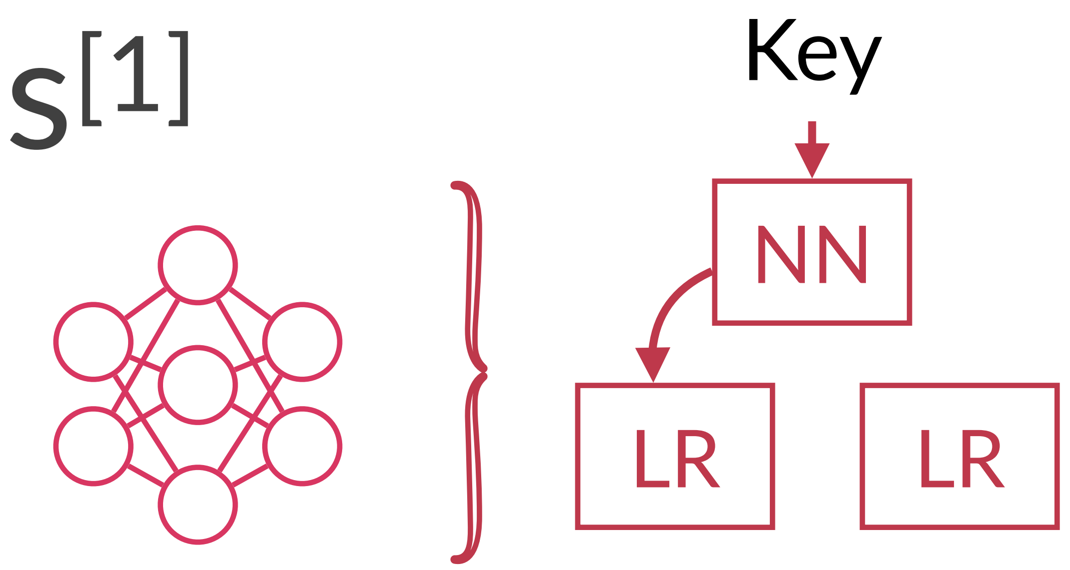
Model is retrained at server in background threads

- Small cost & extra CPU usage at the server



XSTORE uses a two-layer RMI to organize models[1]

- ***Fine-grained*** model retraining



[1] **R**ecursive **M**odel **I**nference, following "The case for the learned index @ SIGMOD'18"

Stale model handling

Background update causes ***stale learned models***

But stale learned models & TT could correctly ***find most keys***

- If the key is ***not moved***, a stale Model & TT still maintains correct

  Key → Logical → Physical

Many other design details & optimizations

Server-side operations

Find ***non-trained*** keys

Optimizations of speculative execution

Dynamic model expansion

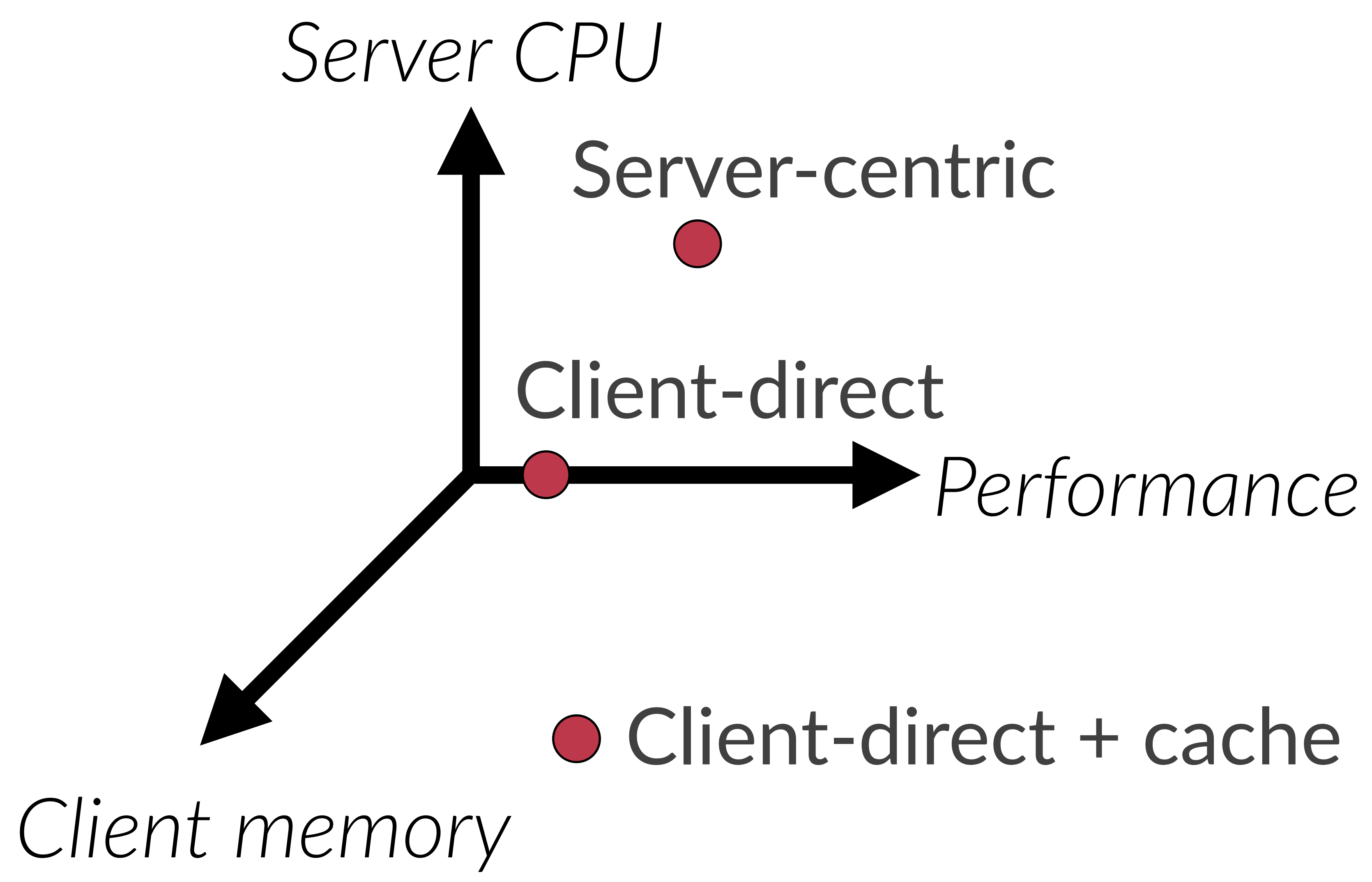
Fault tolerance of XSTORE

Scale-out XSTORE

Evaluation of XSTORE

We answer the following questions:

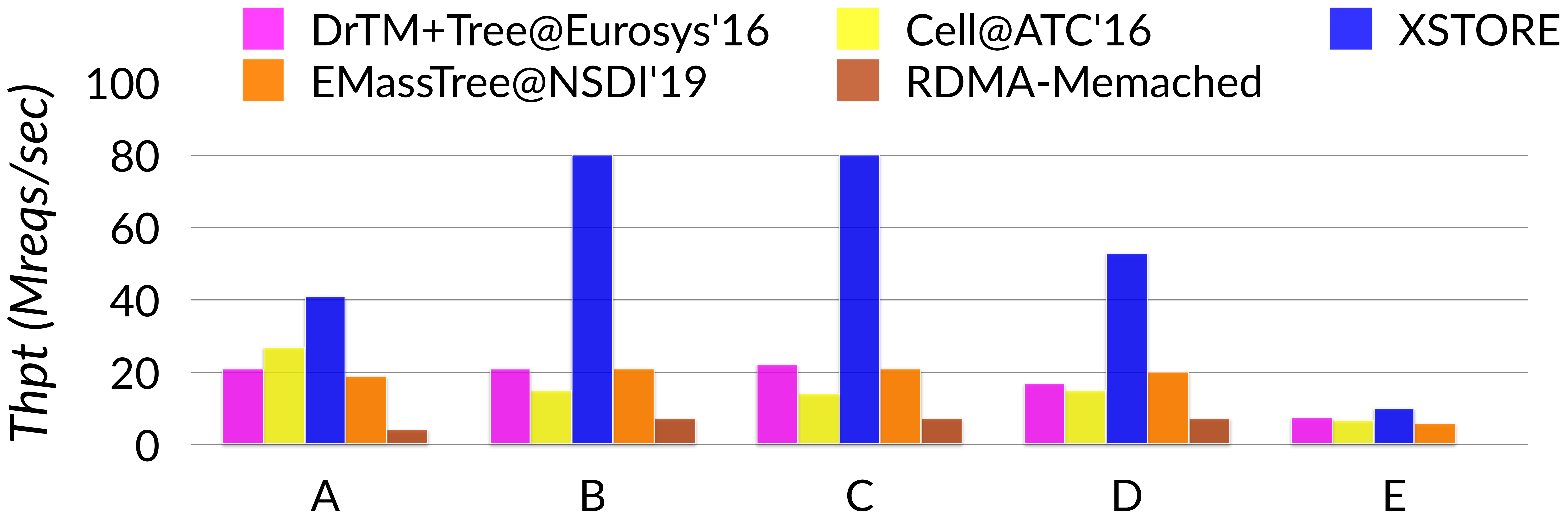
- Comparing to server-centric designs?
- Comparing to client-direct designs?
- Does XStore provide better trade-off?



CPU	RNIC
24 cores	2 * ConnectX-4

Performance of XSTORE on YCSB

100M KVs, uniform workloads

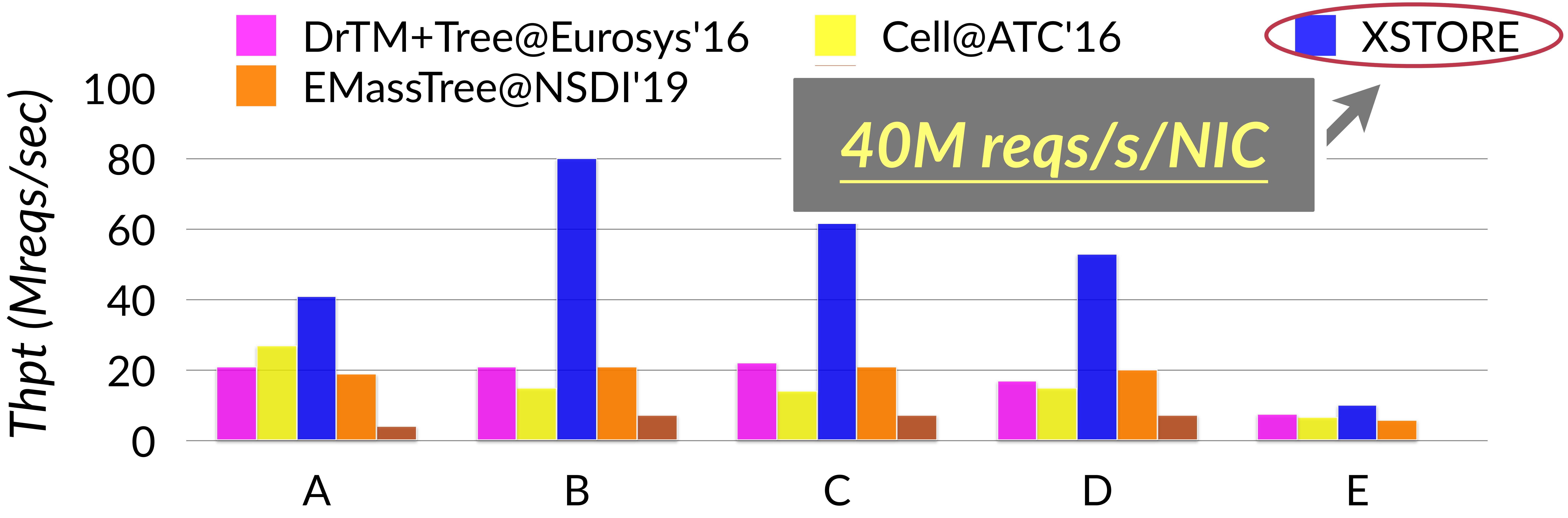


Workload	A	B	C	D	E
R(S)/(U/I)	50:50	90:10	100:0	95:5	95:5

[*] **R**ead, **S**can, **U**pdate, **I**nsert

Performance of XSTORE on YCSB

100M KVs, uniform workloads

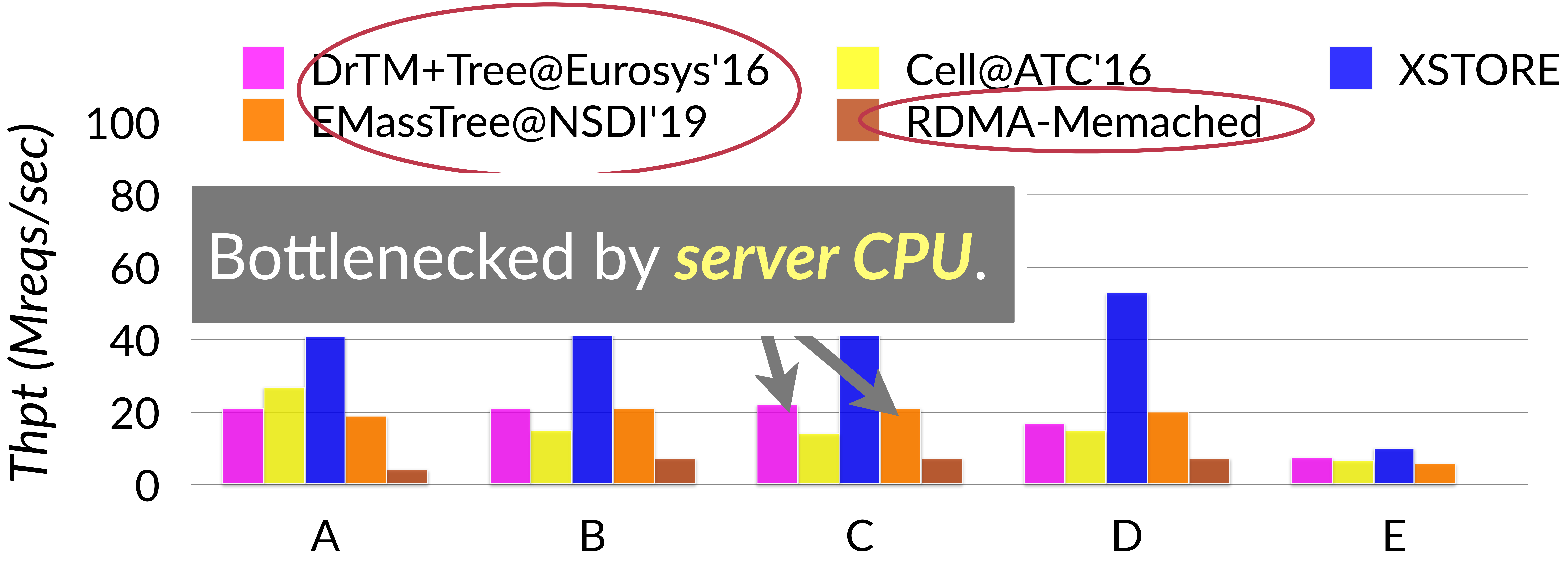


Workload	A	B	C	D	E
R(S)/(U/I)	50:50	90:10	100:0	95:5	95:5

[*] **R**ead, **S**can, **U**pdate, **I**nsert

Performance of XSTORE on YCSB

100M KVs, uniform workloads

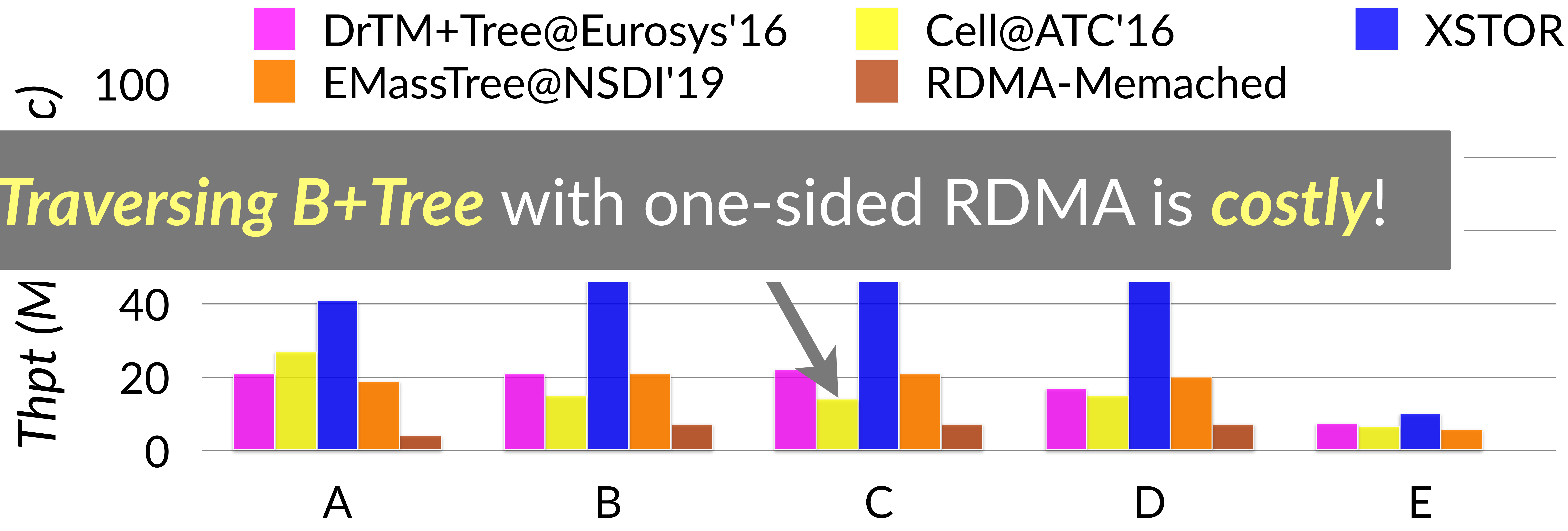


Workload	A	B	C	D	E
R(S)/(U/I)	50:50	90:10	100:0	95:5	95:5

[*] **R**ead, **S**can, **U**pdate, **I**nsert

Performance of XSTORE on YCSB

100M KVs, uniform workloads



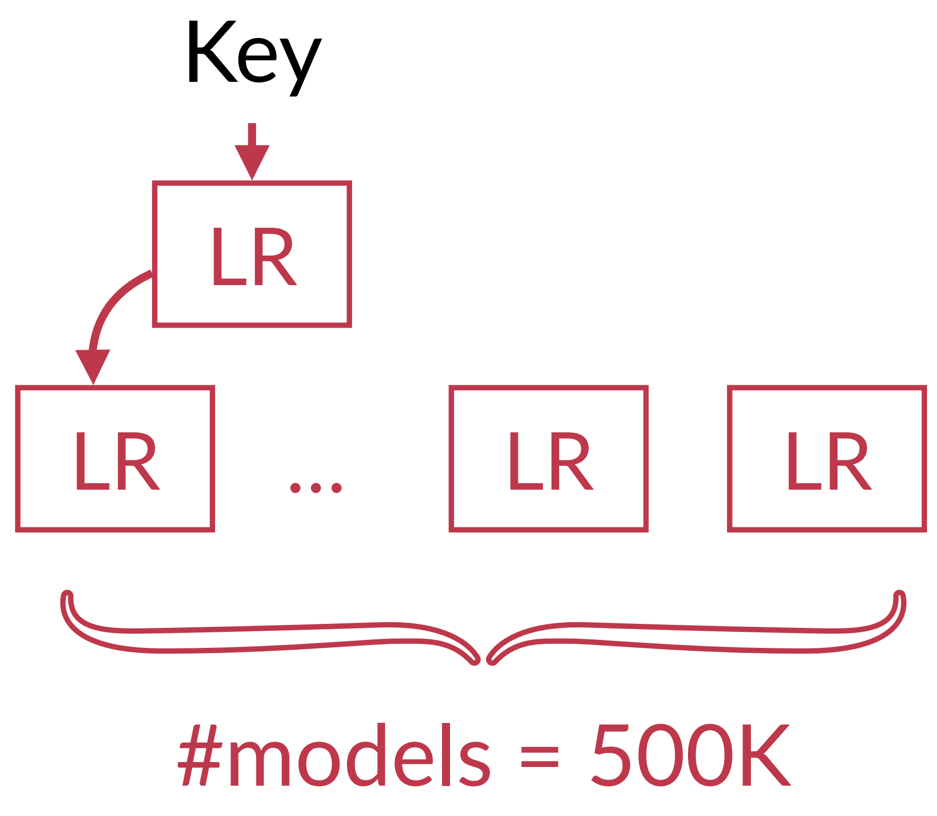
Workload	A	B	C	D	E
R(S)/(U/I)	50:50	90:10	100:0	95:5	95:5

[*] Read, Scan, Update, Insert

The XCache in details

For a 100M KVs YCSB dataset

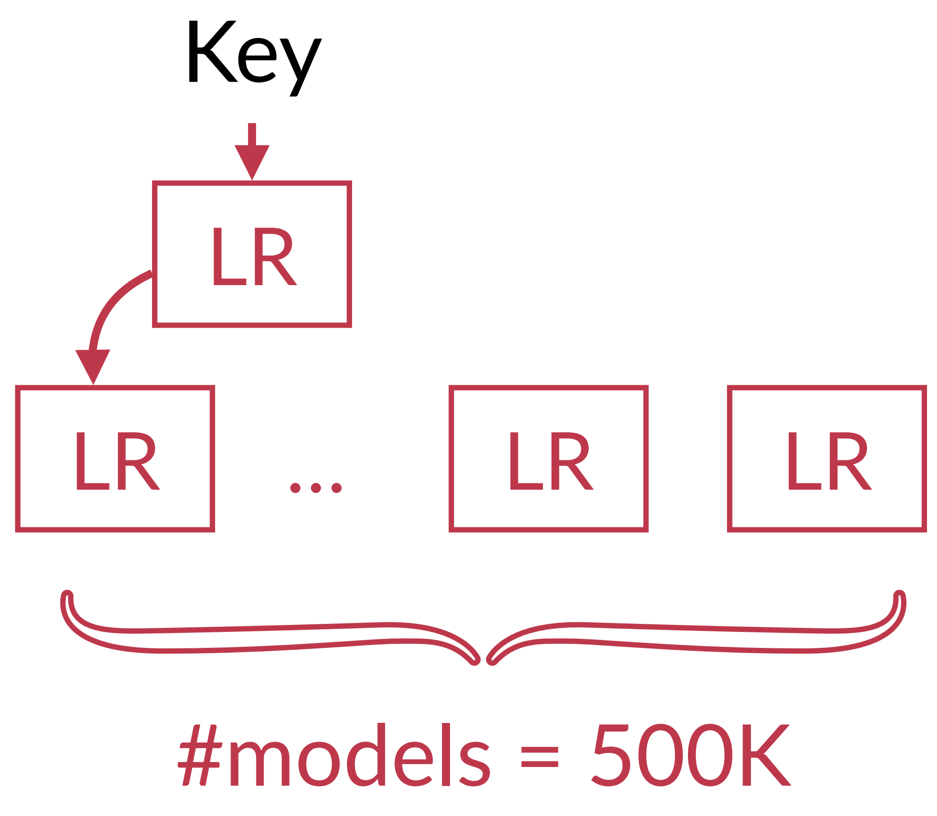
- 500K Linear regression as models, each 14B
- ~ 8μs to retrain each model
- ~ 8s to train the entire cache



The XCache in details

For a 100M KVs YCSB dataset

- ***500K*** Linear regression as models, each 14B
- ~ 8µs to retrain each model
- ~ 8s to train the entire cache

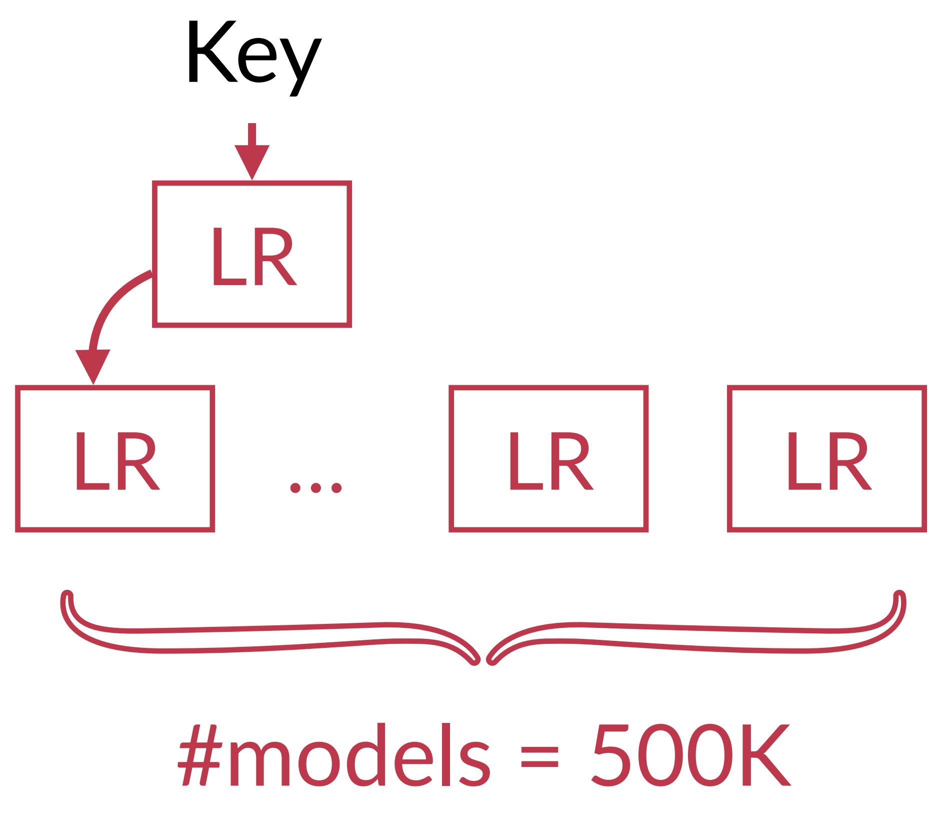


Small model to fit the dataset

The XCache in details

For a 100M KVs YCSB dataset

- ***500K*** Linear regression as models, each 14B
- ***~ 8µs*** to retrain each model
- ~ 8s to train the entire cache



Quick retrain under dynamic workload

Sensitive to the dataset

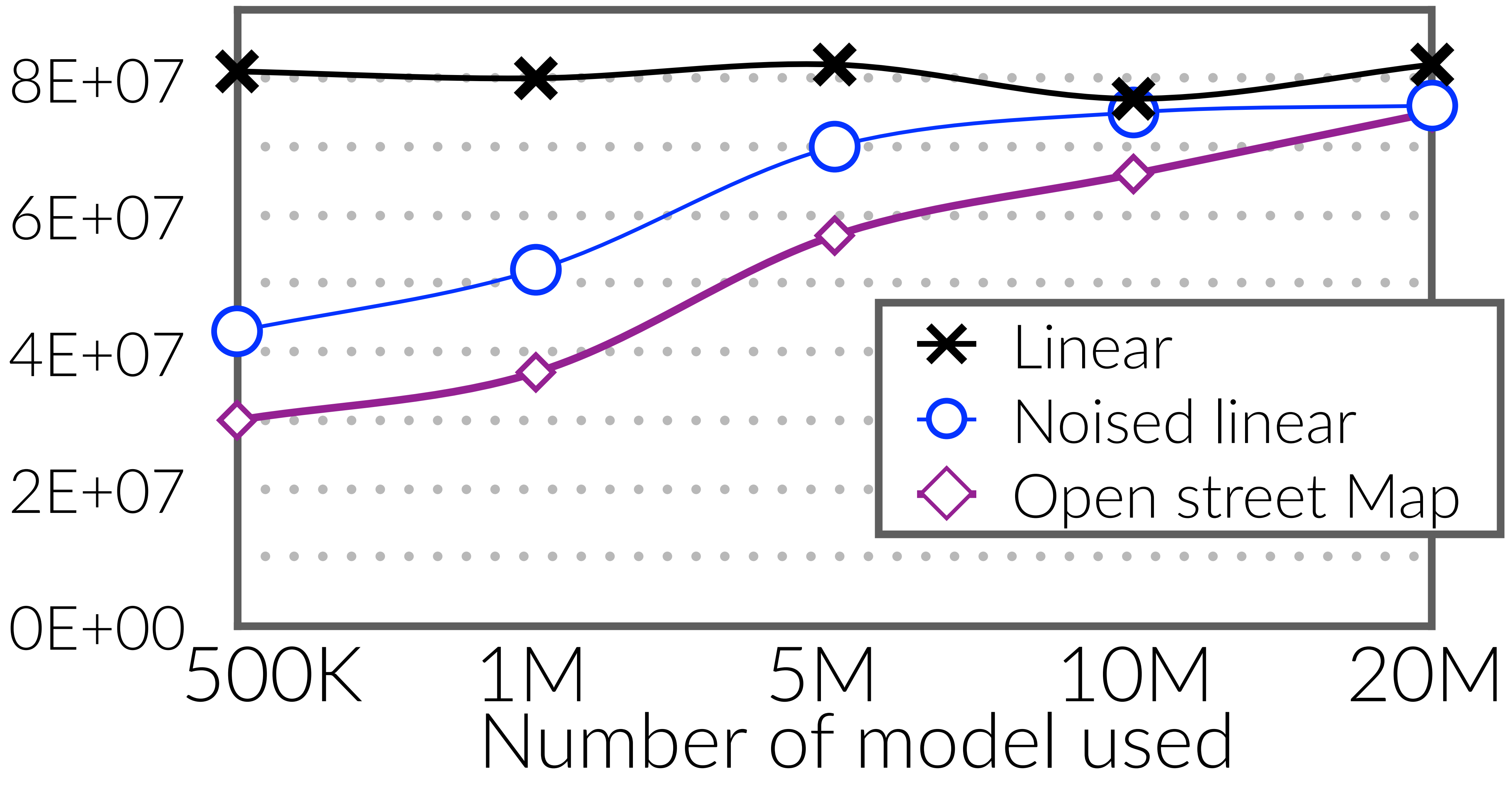
Different dataset has different accuracy

𝄢: May affect the performance

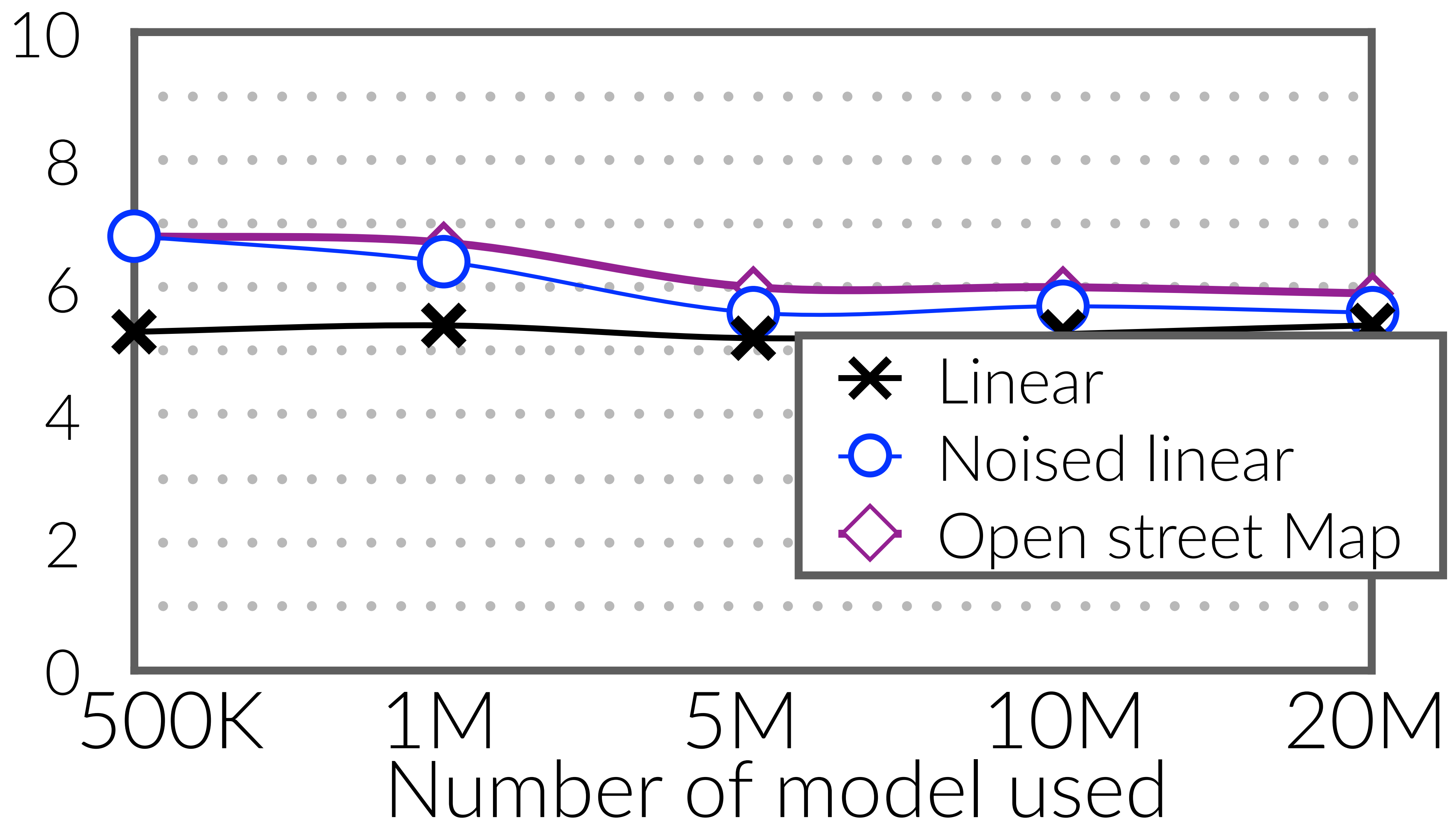
Name	Workloads
Linear	e.g., YCSB,TPC-C
Noised Linear	e.g., YCSB
Open street map	e.g., OpenStreetMap

Throughput drop due to increased error for complex dataset

Peak throughput (100M dataset)



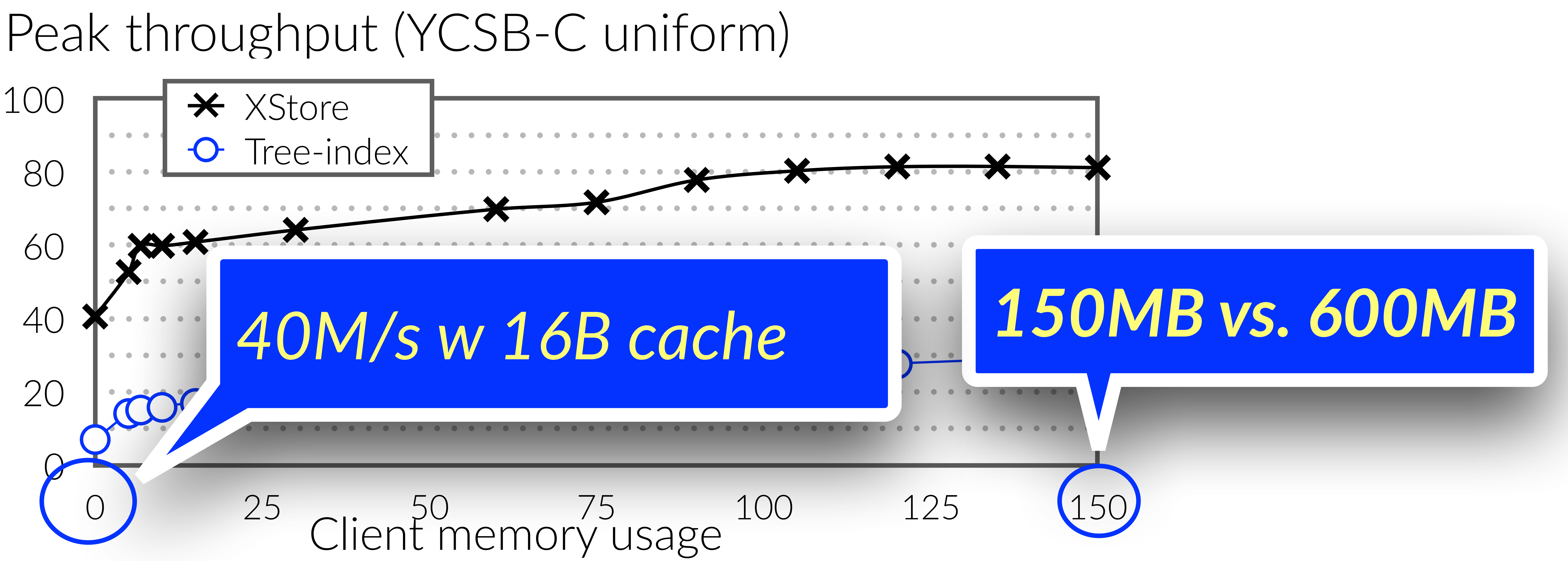
Average latency (µs)



Learned cache vs. Tree-based cache

XStore provides better ***memory-performance trade-off***

𝄢 YCSB-C uniform workload



Current limitations and future work

XSTORE currently only supports fixed-length keys

- Our paper describes our plan to support variable-length keys

Focus on simple models (e.g., LR)

- Efficient upon retraining under dynamic workloads
- May results in huge error for complex data distribution
- Trade-off: retraining speed vs. accuracy vs. memory

Orthogonal to the design of XSTORE

Conclusion

XSTORE provides ***a new design*** for RDMA-enabled KVS

- First adopts the learned models for one-sided RDMA READ

XSTORE provides better trade-offs:

- ***Server-side CPU*** vs. ***Client-side memory*** vs. ***Performance***

Please check XSTORE@

- https://ipads.se.sjtu.edu.cn/projects/xstore

Thanks & QA