How to stop underutilization and love multicores

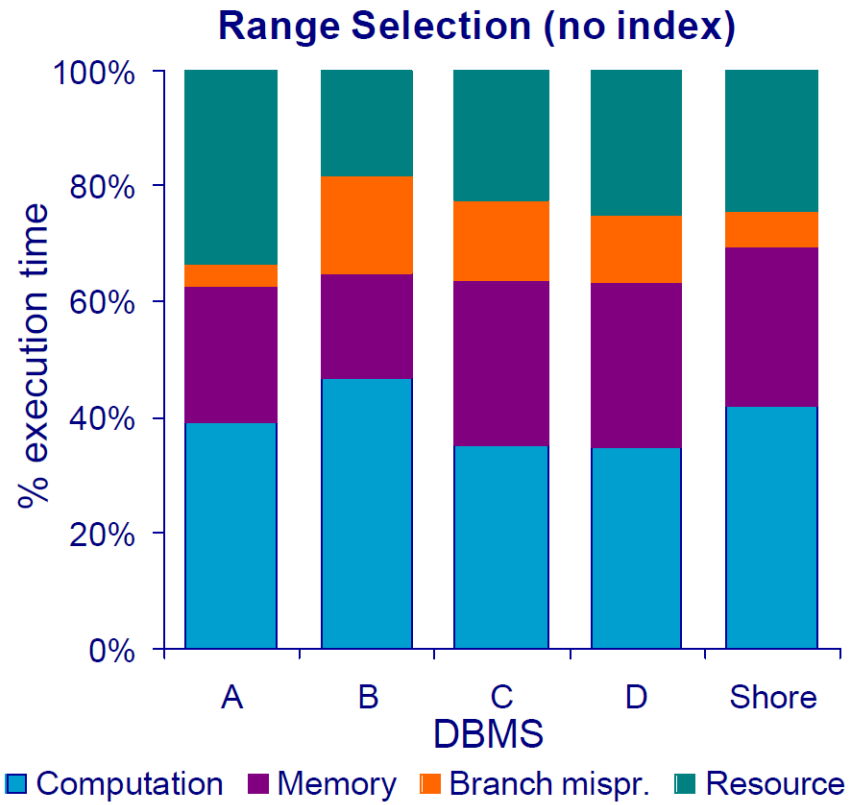
Anastasia Ailamaki (EPFL)
Erietta Liarou (EPFL)
Pınar Tözün (IBM Almaden)
Danica Porobic (EPFL)
Iraklis Psaroudakis (EPFL, SAP)

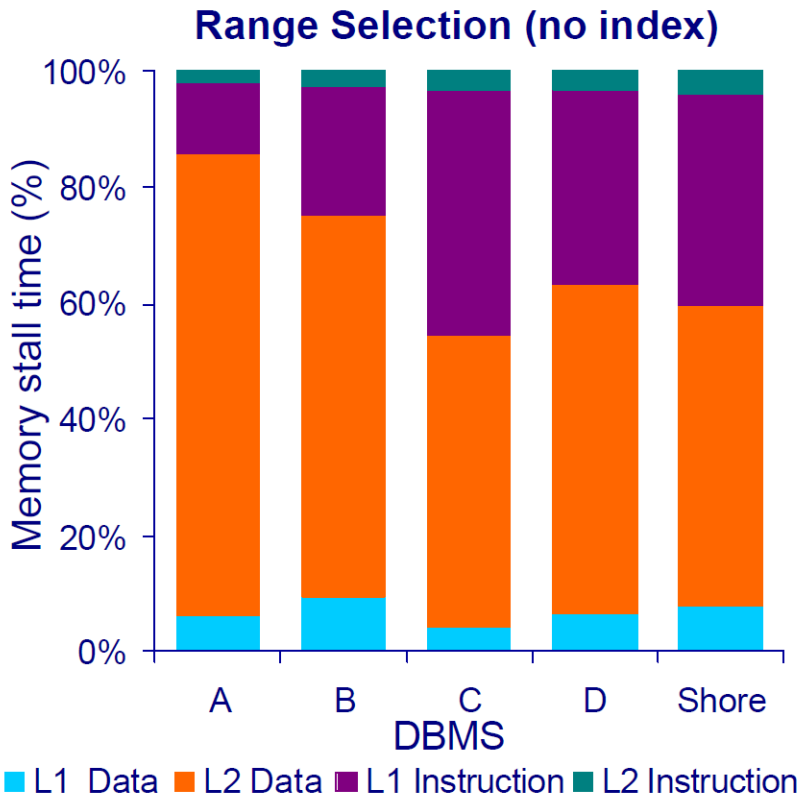
http://tinyurl.com/LoveMulticores

DATA-INTENSIVE APPLICATIONS AND SYSTEMS

once upon a time …

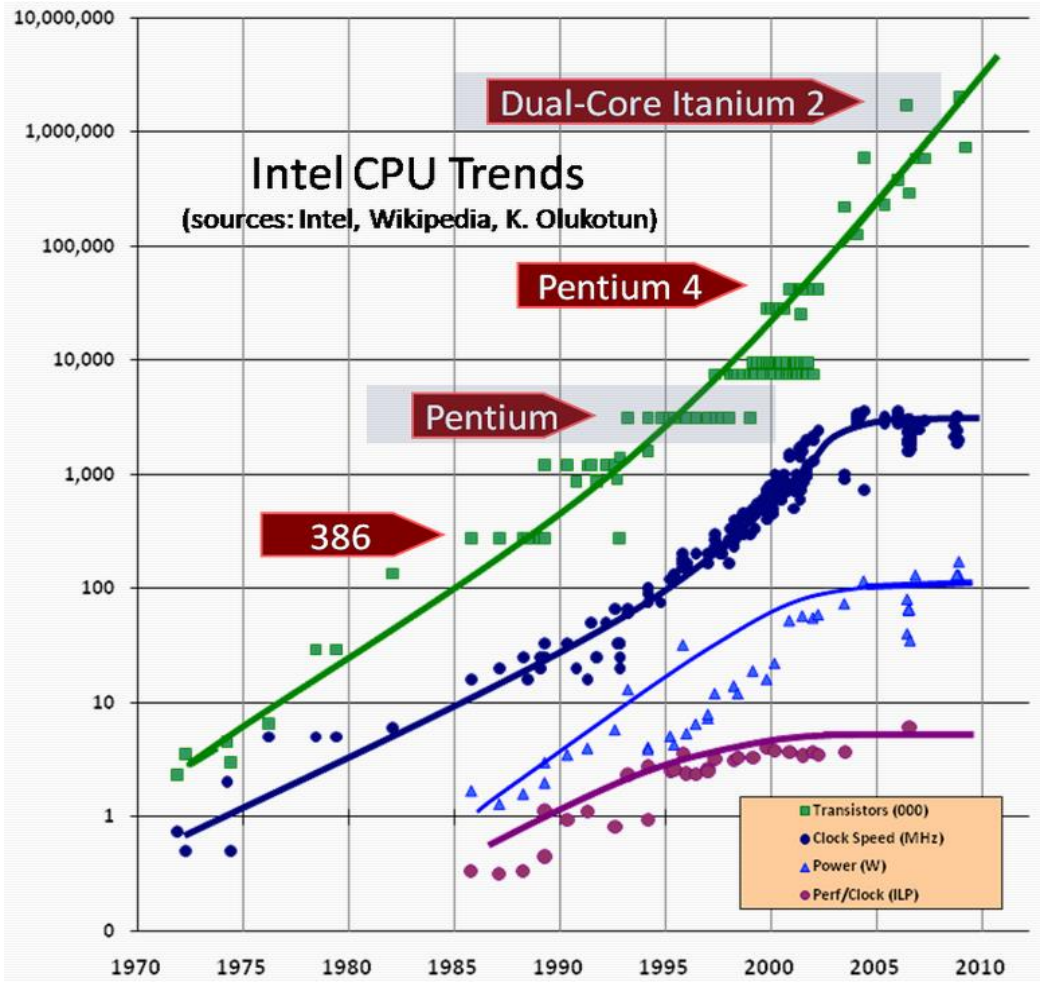
[VLDB99]





processor stalled >50% of the time

Moore's law

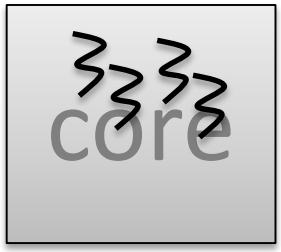


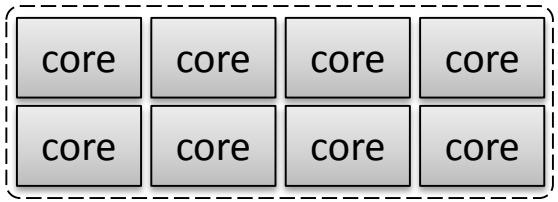
doubling of transistor counts continues

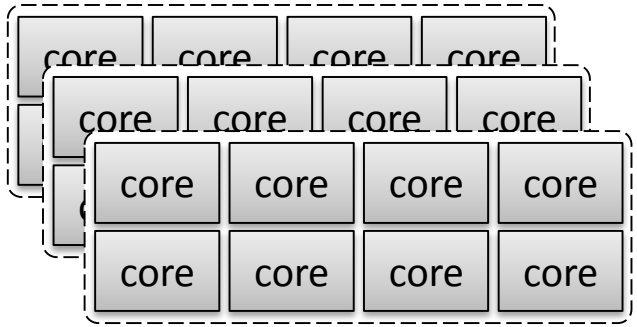
clock speeds and power hit the wall

processor trends

2005







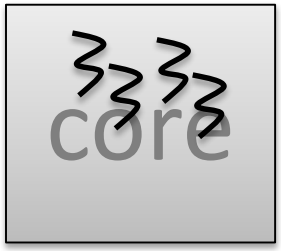
pipelining
ILP
multithreading

multicores
(CMP)

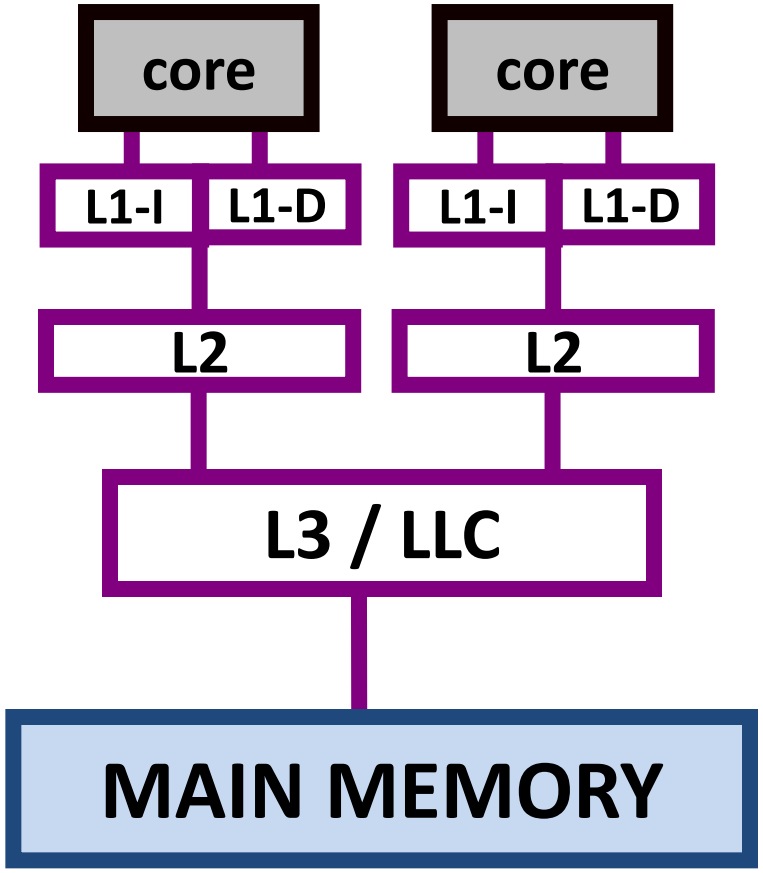
multisocket
multicores

goal: scalability

vertical dimension: cores & caches



pipelining
ILP
multithreading

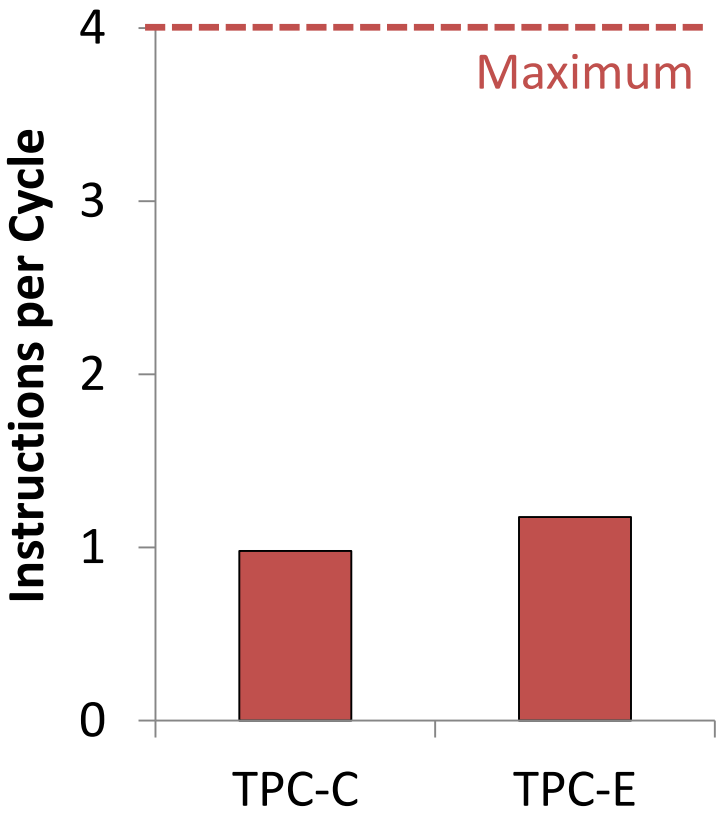


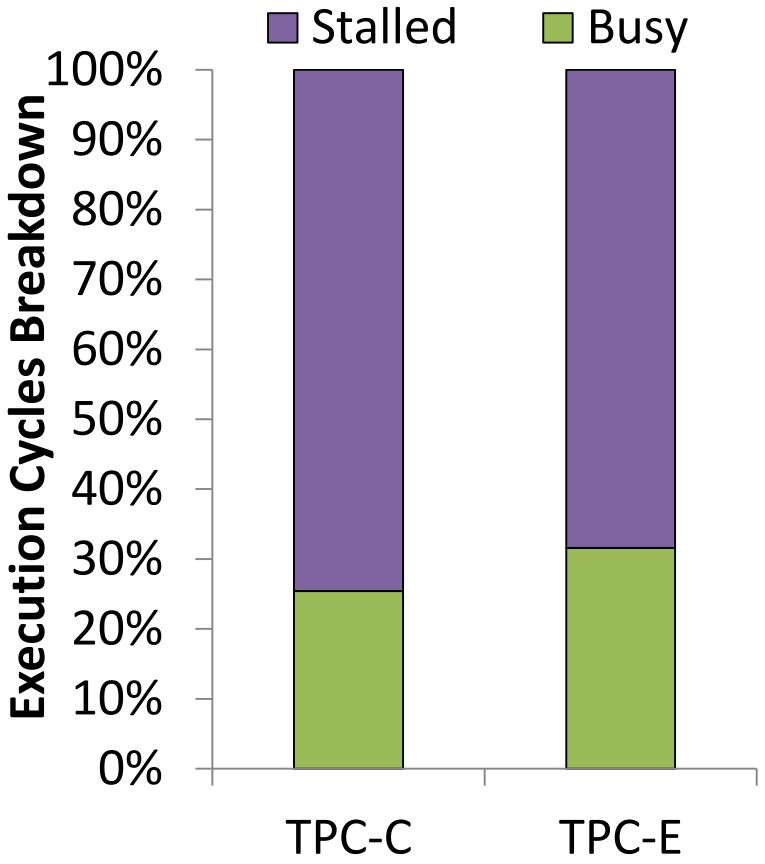
implicit parallelism & memory matters

now: cores & cache utilization

[EDBT13]

at peak throughput on Shore-MT, Intel Xeon X5660

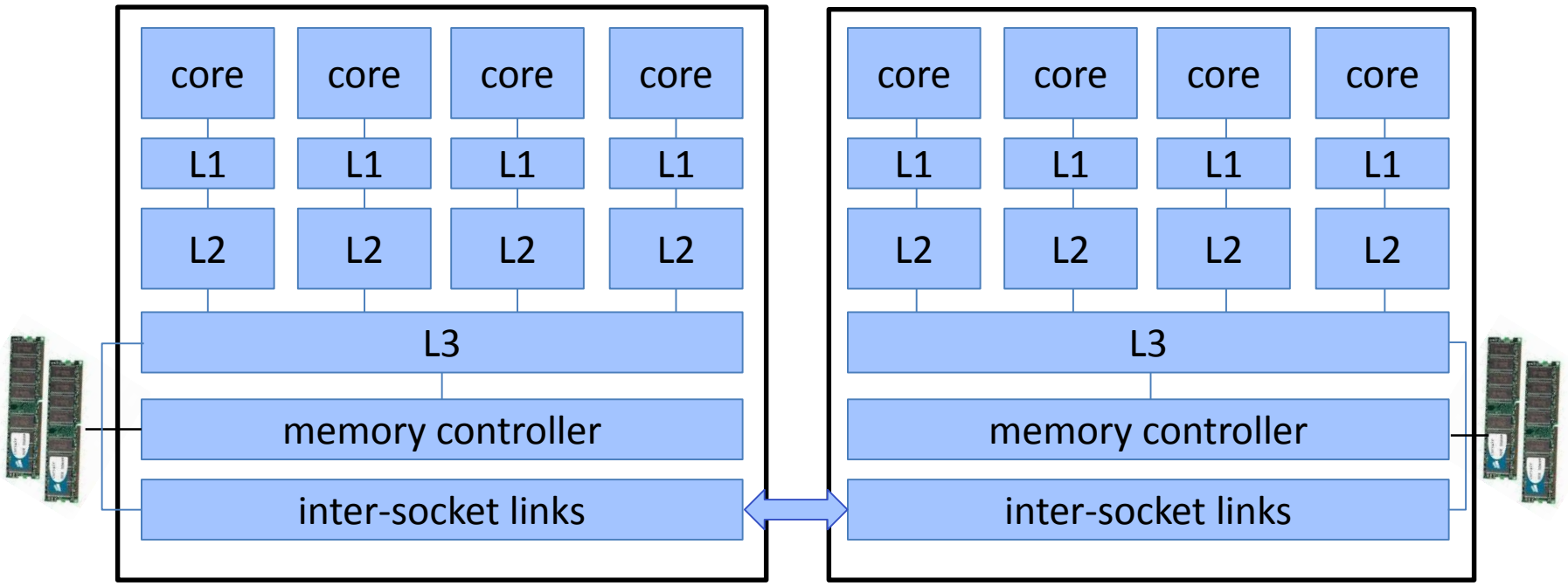




IPC < 1 on a 4-issue machine

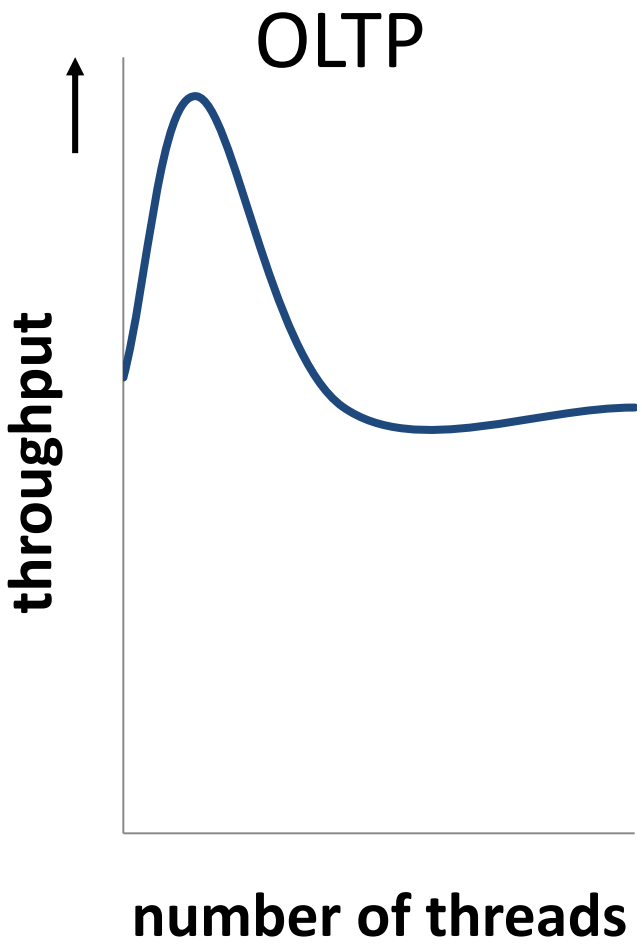
70% of the execution time goes to stalls

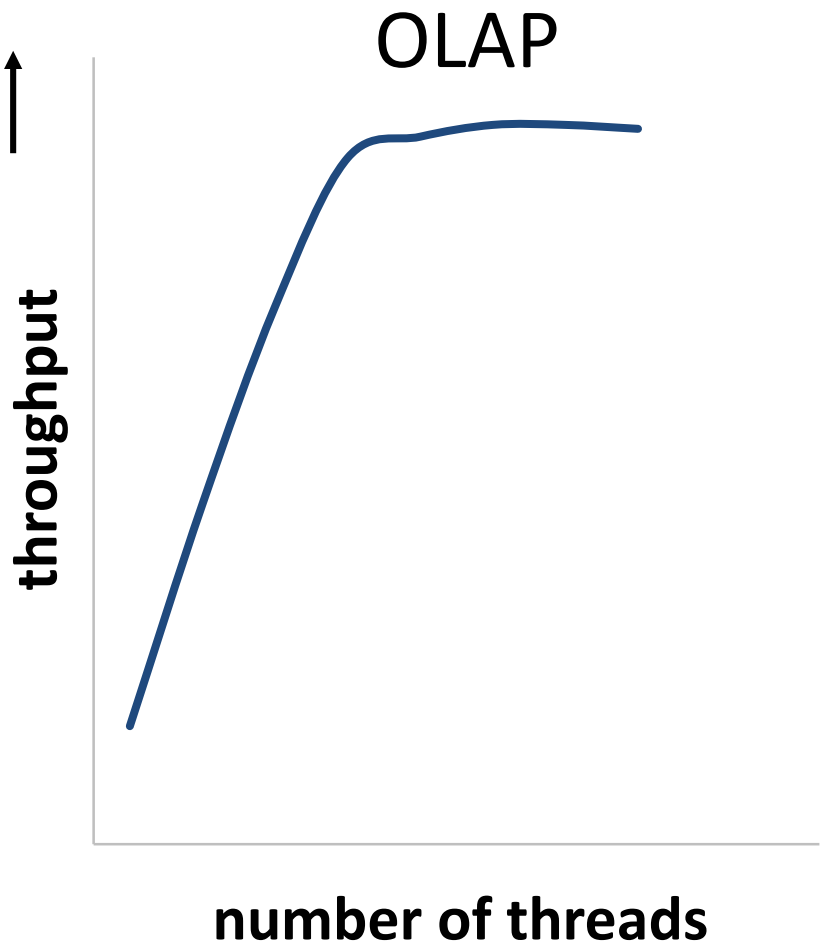
horizontal dimension: cores & sockets



exploit abundant parallelism

workload scalability on multicores





access latency

memory bandwidth

stopping underutilization

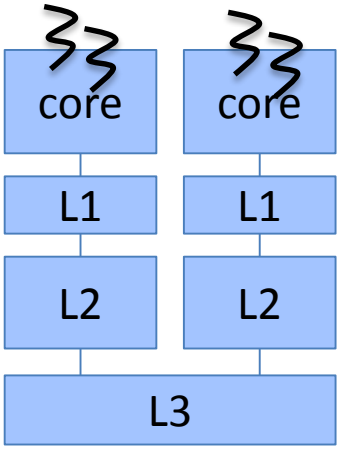
- how to adapt traditional execution models to fully exploit modern hardware?

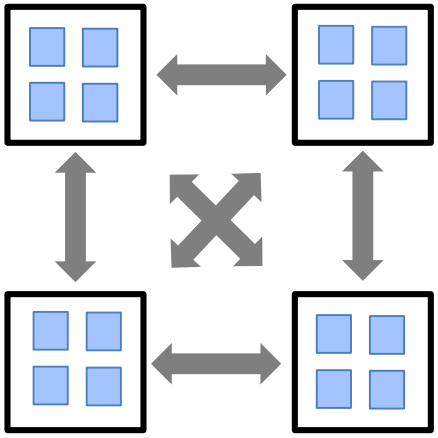
- how to maximize data & instruction locality at the right level of the memory hierarchy?

- how to continue scaling-up despite many cores and non-uniform topologies?

utilization

scalability



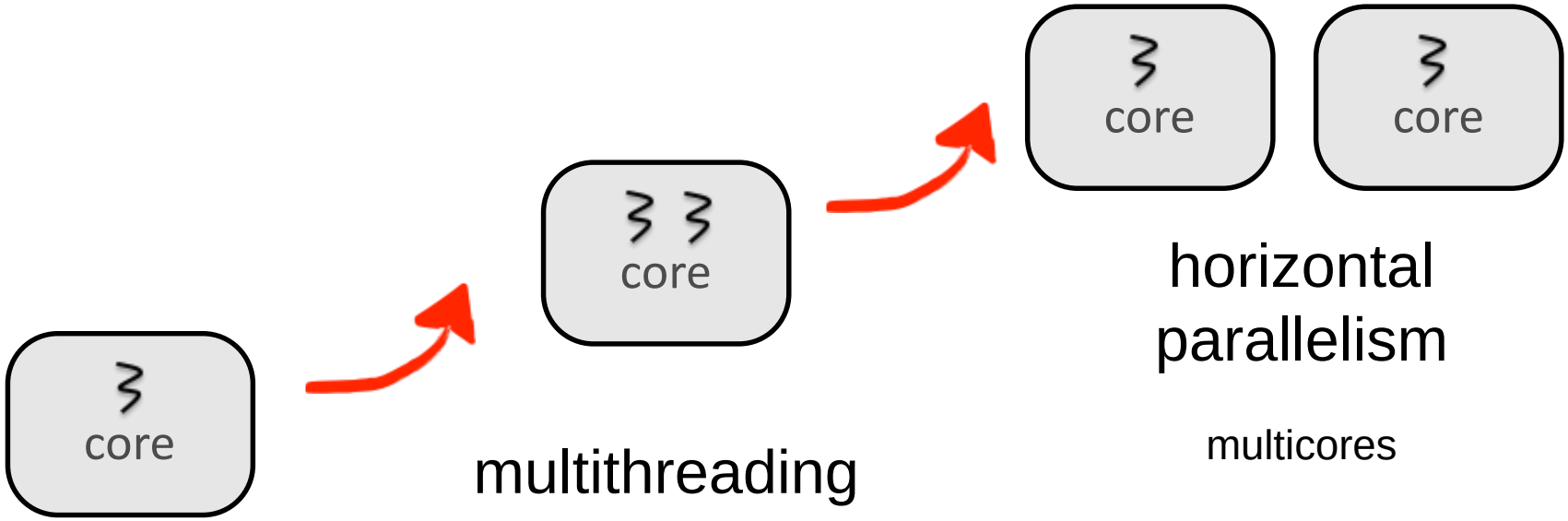


exploiting core's resources
minimizing memory stalls

scaling up OLTP
scaling up OLAP
conclusions

http://tinyurl.com/tutorial-2015-feedback

modern parallelism



instruction & data parallelism

pipelining
superscalar
SIMD

multithreading

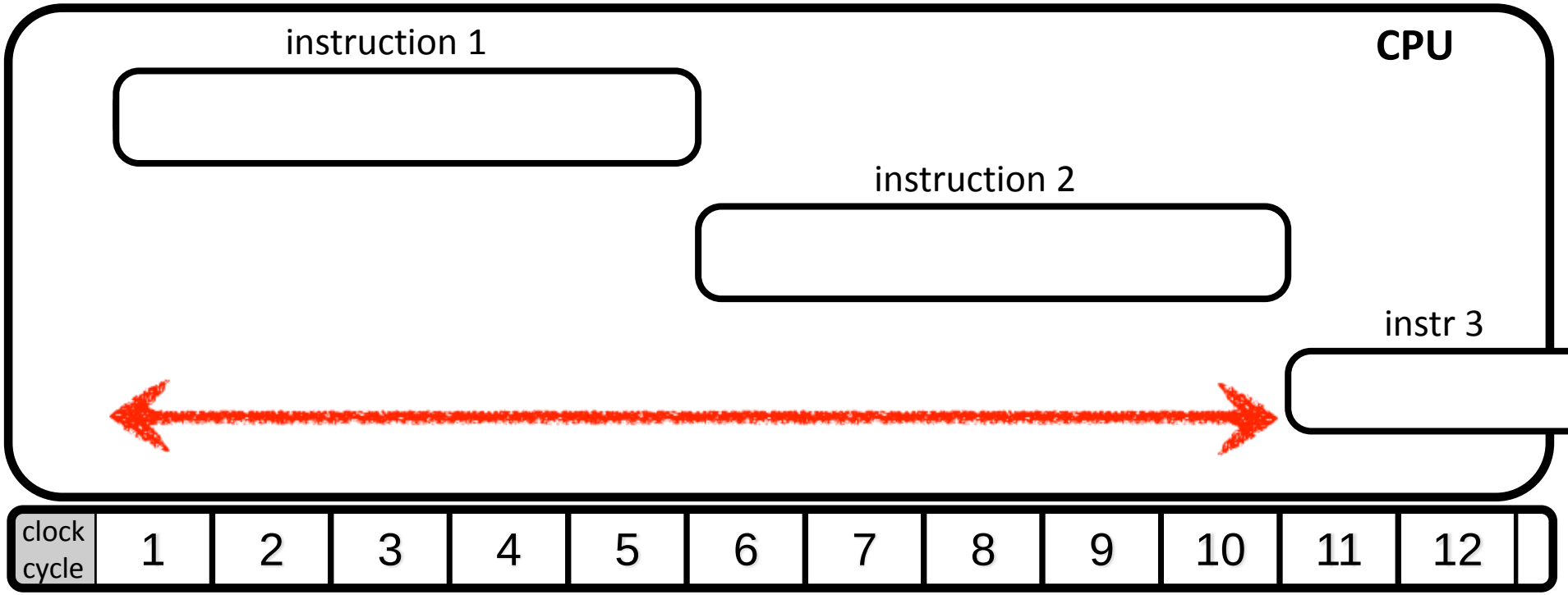
SMT

horizontal parallelism

multicores

subscalar CPUs

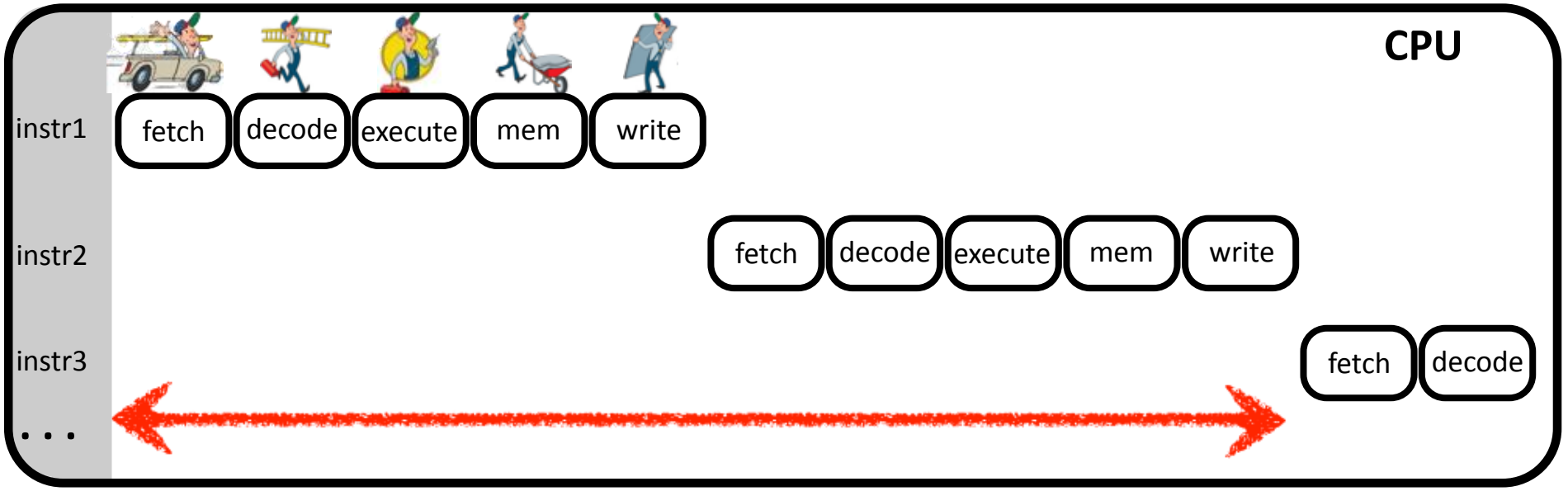
one instruction at a time



... ten cycles to complete 2 instructions!

subscalar CPUs

one instruction at a time

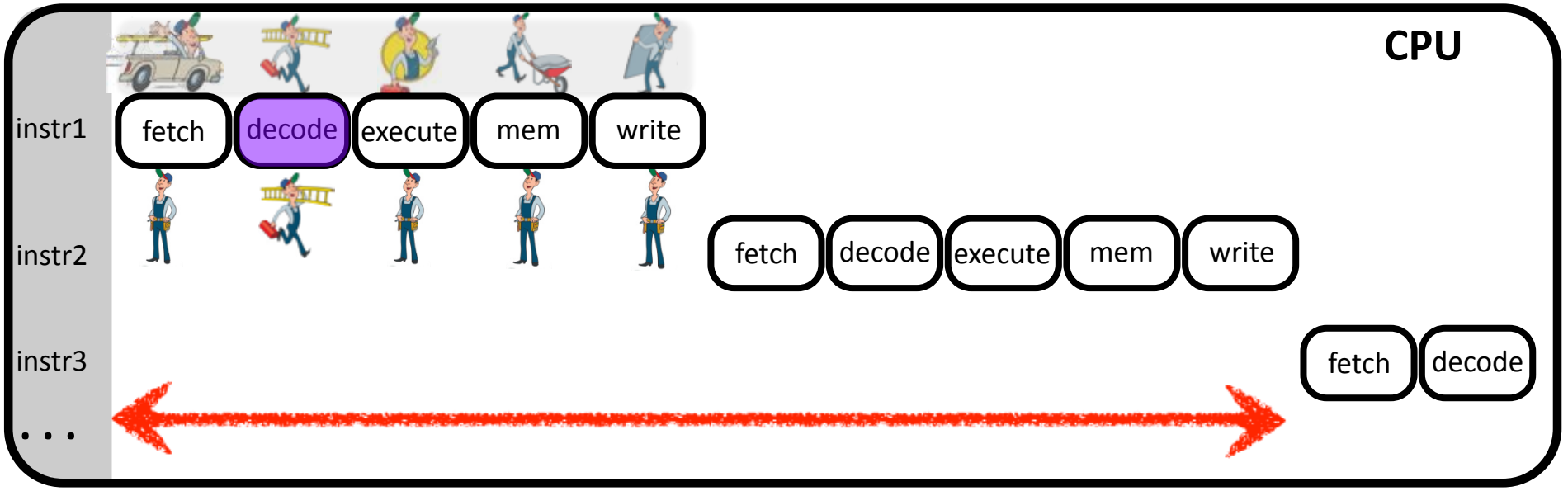


clock cycle	1	2	3	4	5	6	7	8	9	10	11	12

... ten cycles to complete 2 instructions!

subscalar CPUs

one instruction at a time



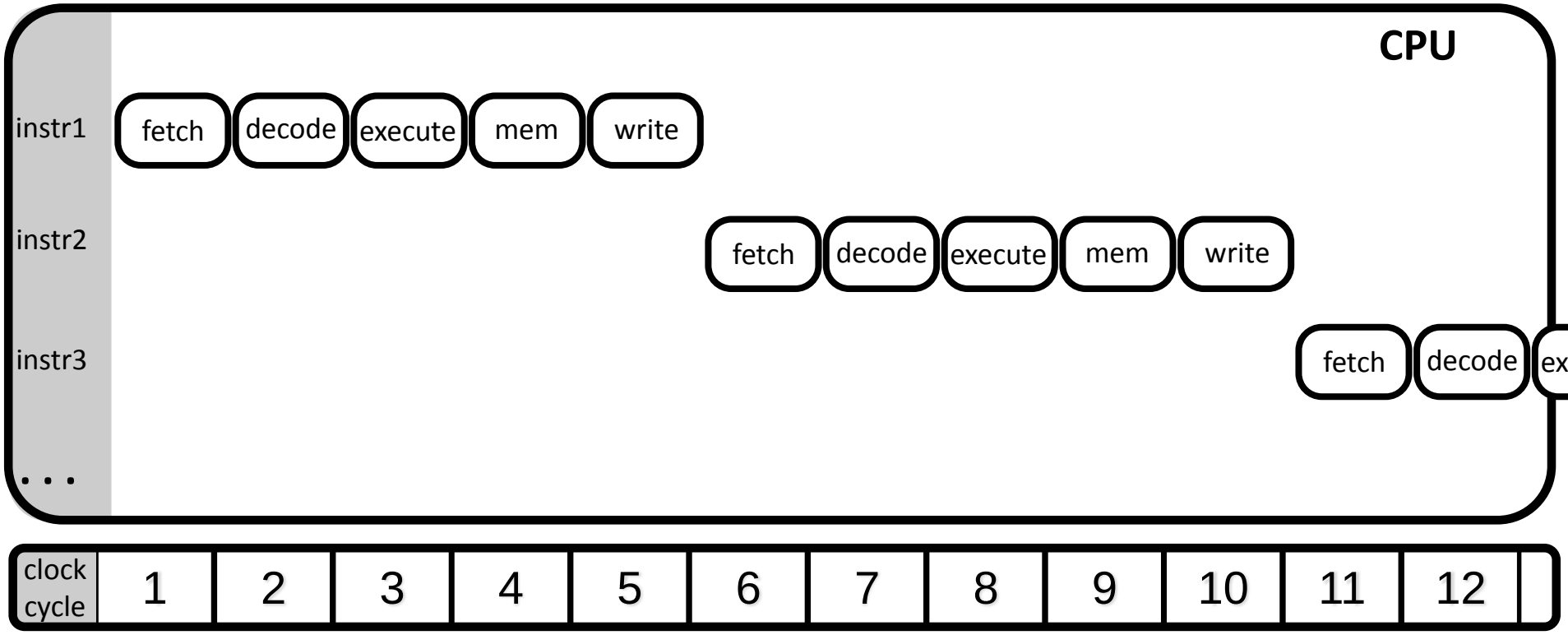
clock cycle	1	2	3	4	5	6	7	8	9	10	11	12

... ten cycles to complete 2 instructions!

fundamental way to parallelize

Instruction pipelining:

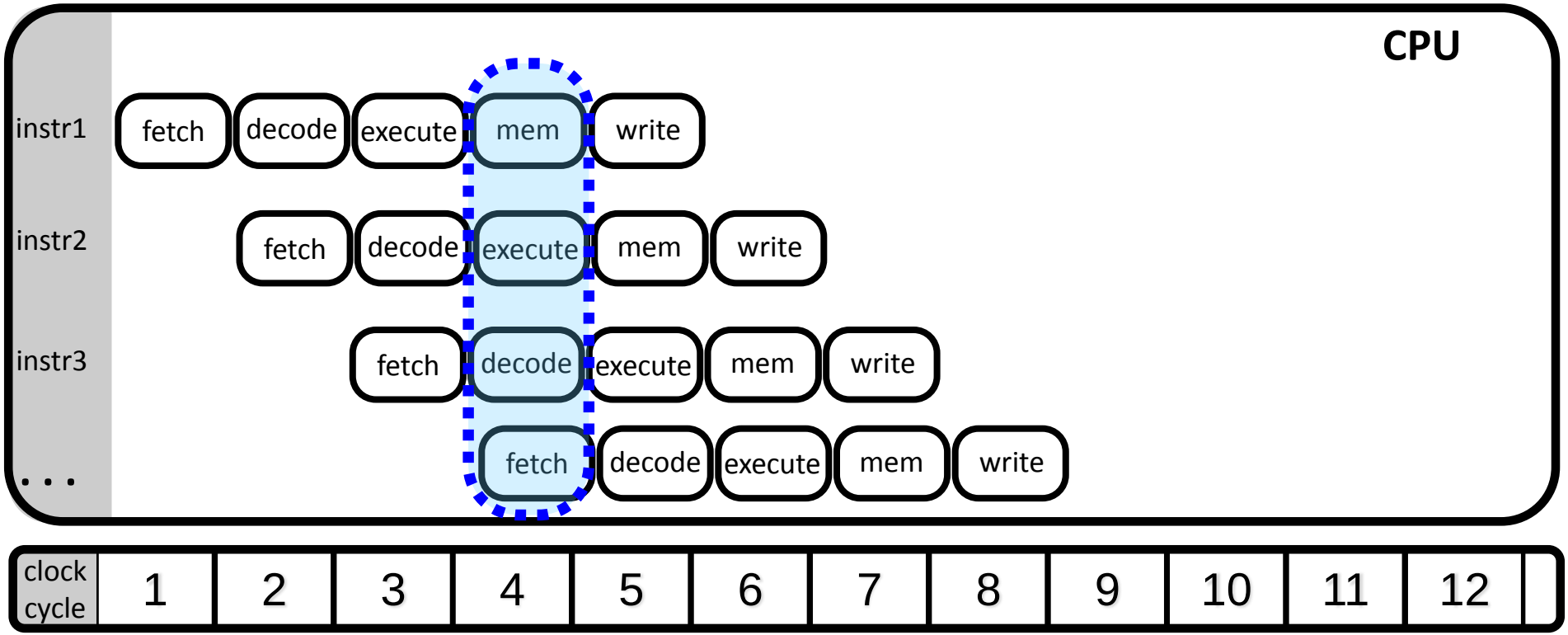
multiple instructions can be partially overlapped



fundamental way to parallelize

Instruction pipelining:

multiple instructions can be partially overlapped

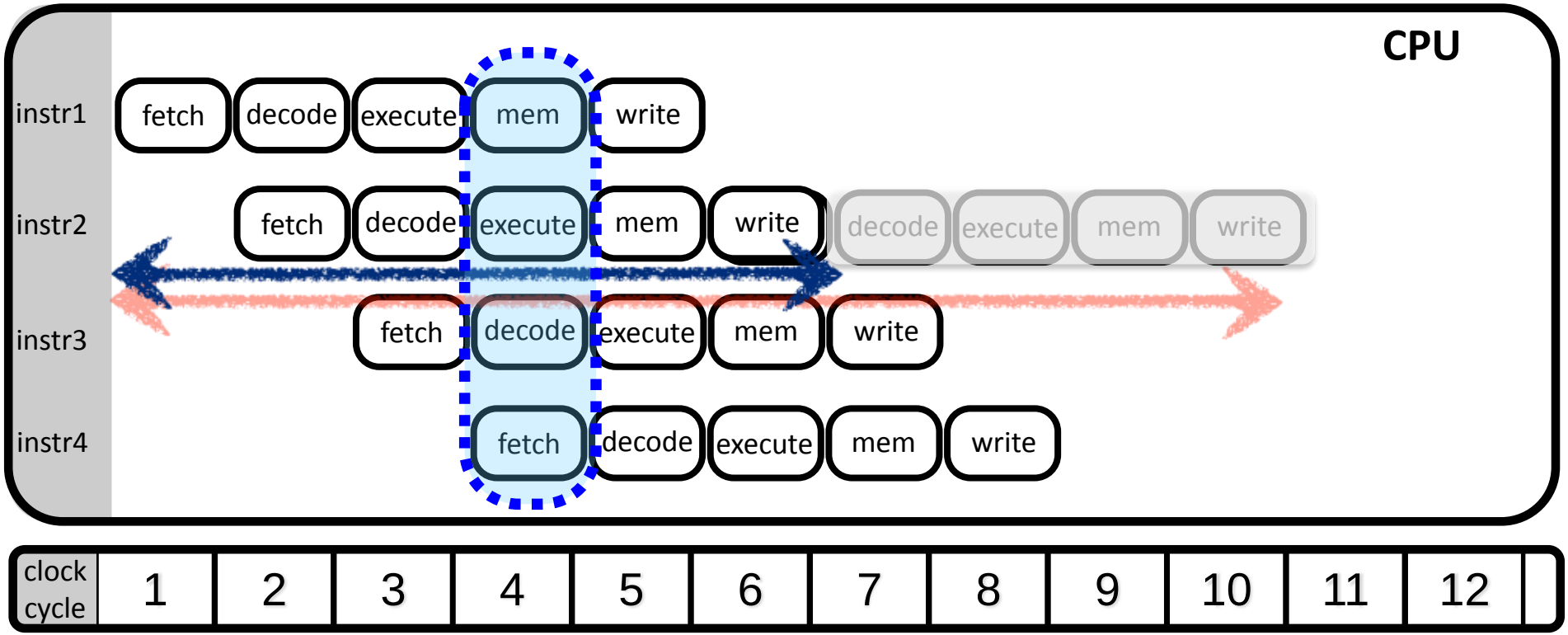


increase the utilization of on-die execution resources

fundamental way to parallelize

Instruction pipelining:

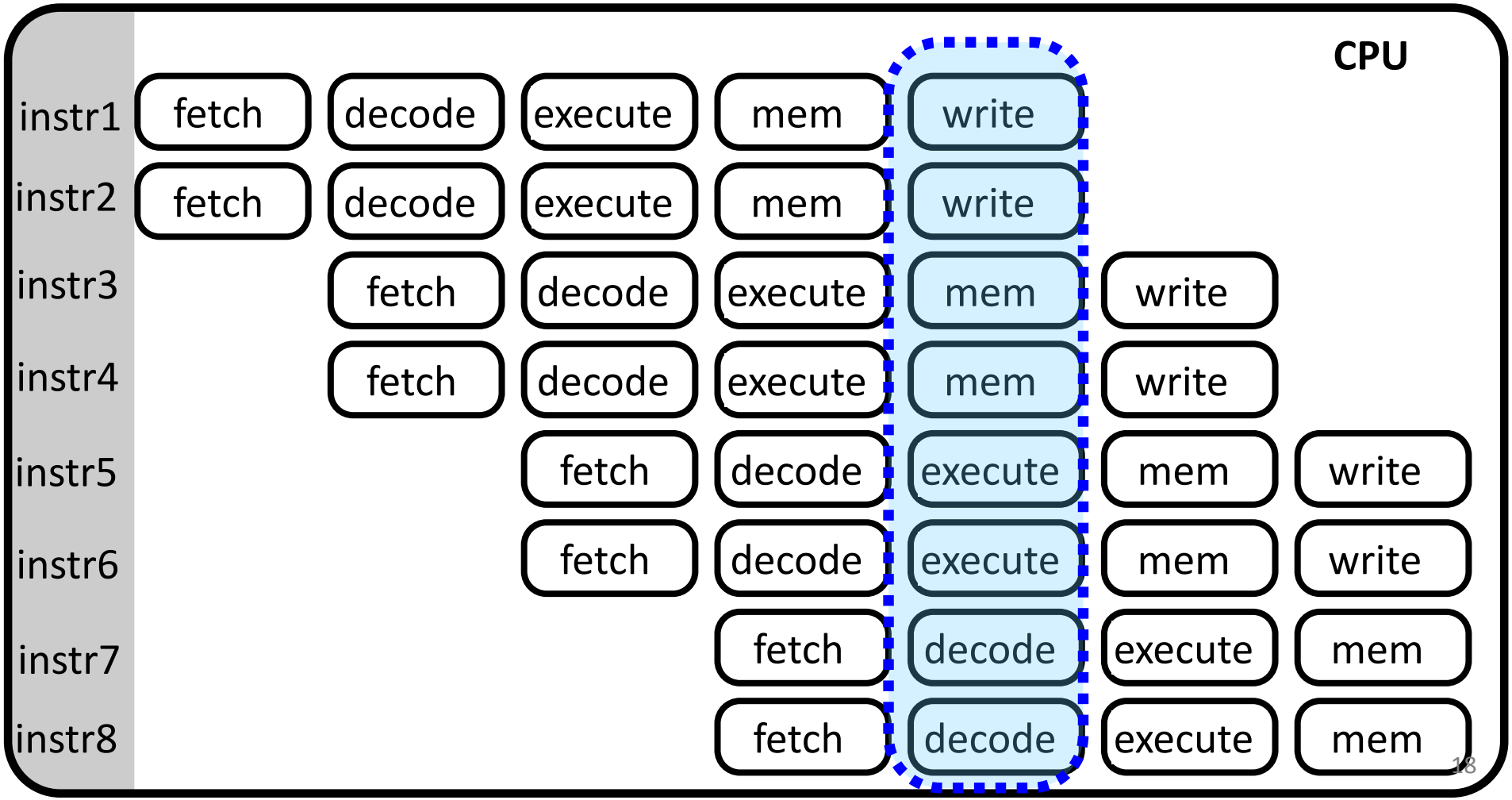
multiple instructions can be partially overlapped



increase the instruction throughput

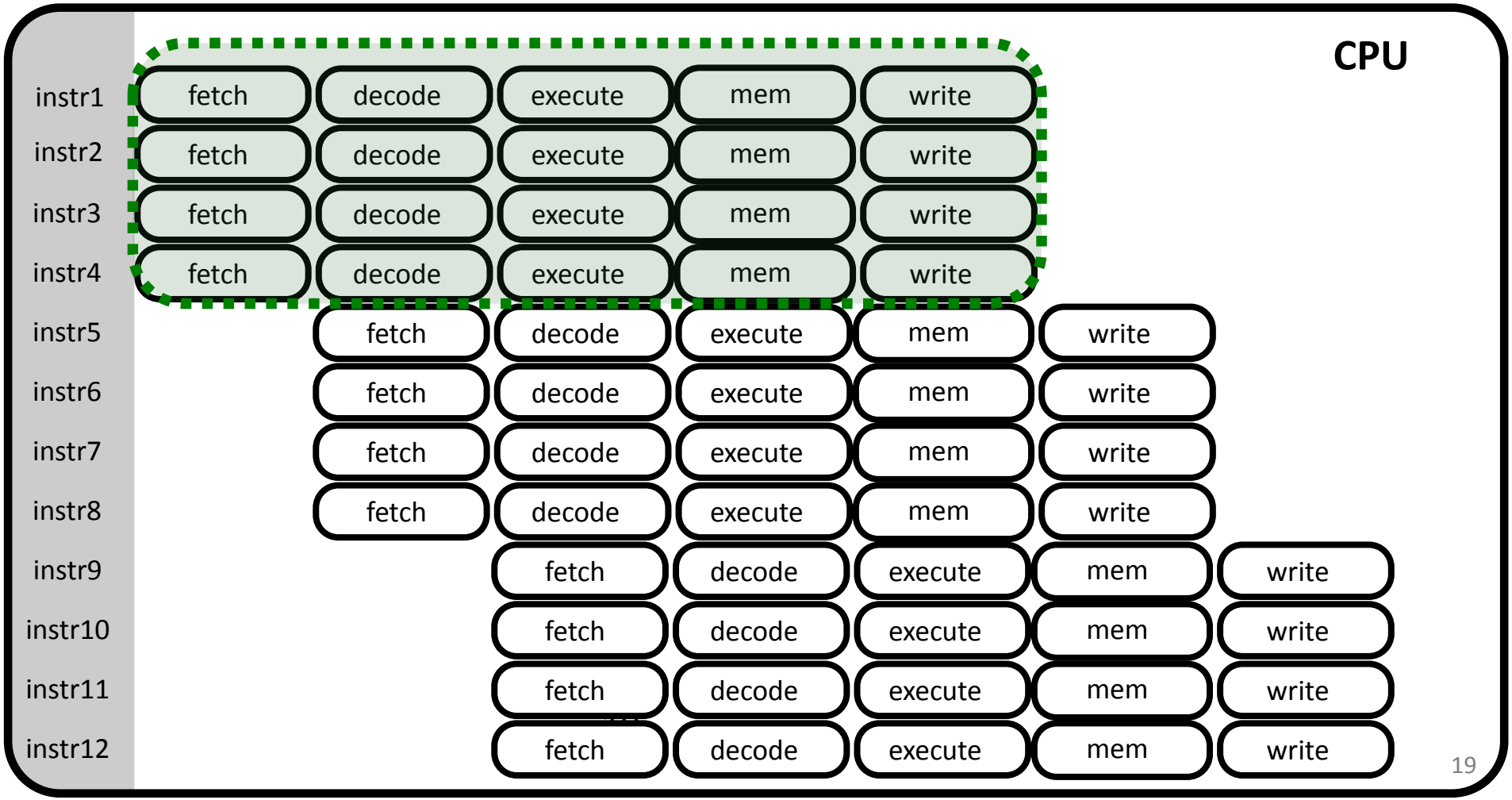
superscalar cpu

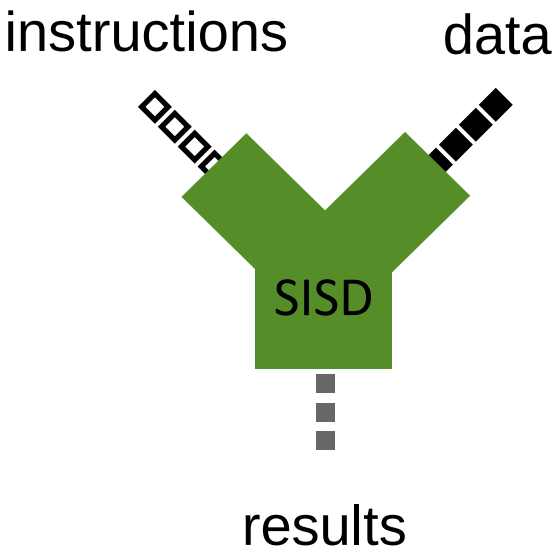
more than one instructions during a clock cycle

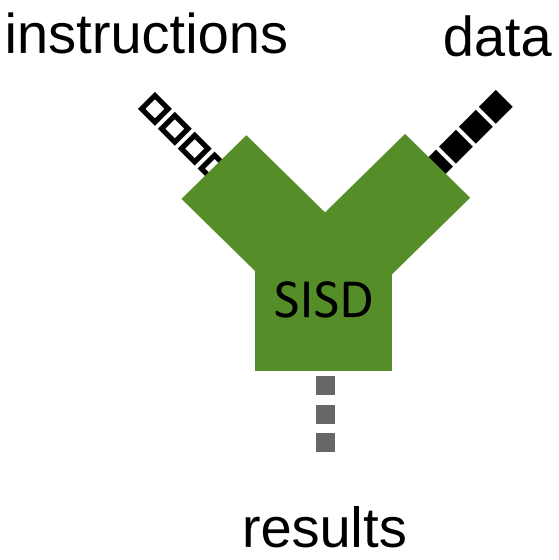


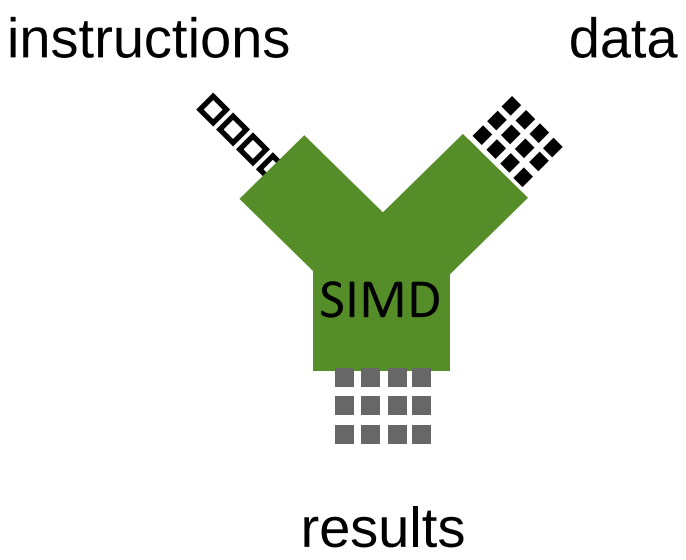
superscalar cpu

4 instructions during a clock cycle

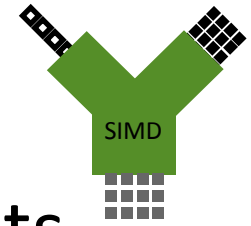



instructions
data
SISD
results


instructions
data
SISD
results


instructions
data
SIMD
results

SIMD (single instruction multiple data)



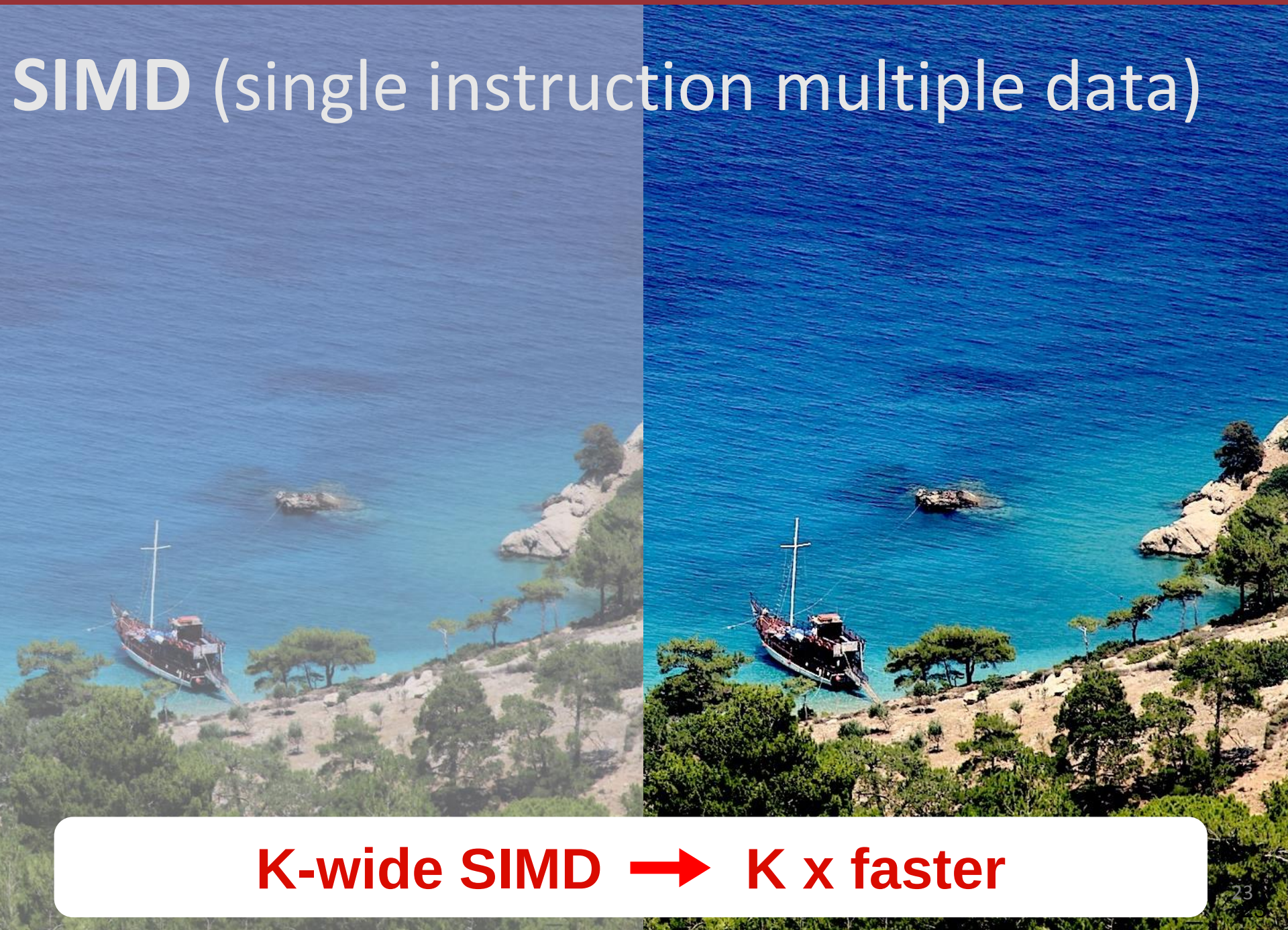
Apply an instruction to multiple data elements

- allows parallelism
- process of **K** elements at a time → speedup of **K**

processing large arrays of numeric values

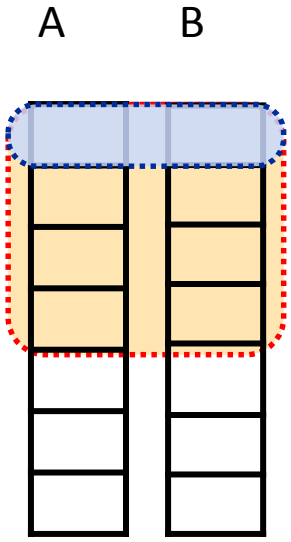
e.g., the same value is being added to a large number of data points

SIMD (single instruction multiple data)

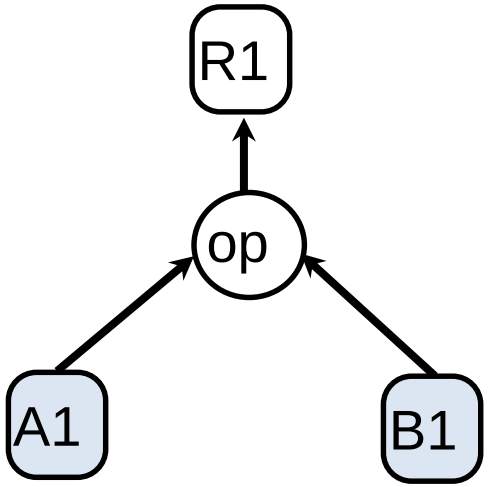


K-wide SIMD → K x faster

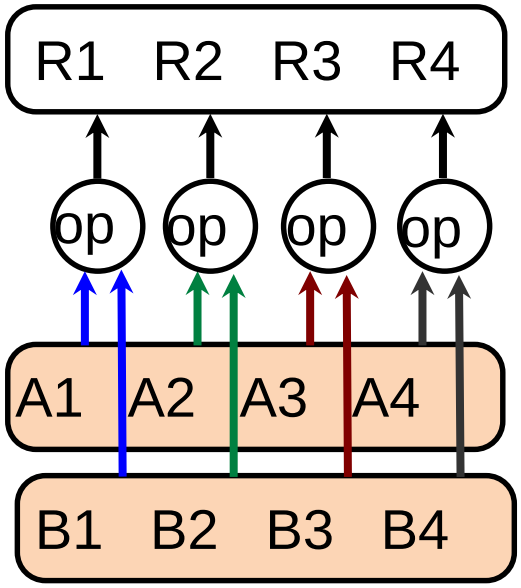
SISD to SIMD



traditionally (SISD)



SIMD

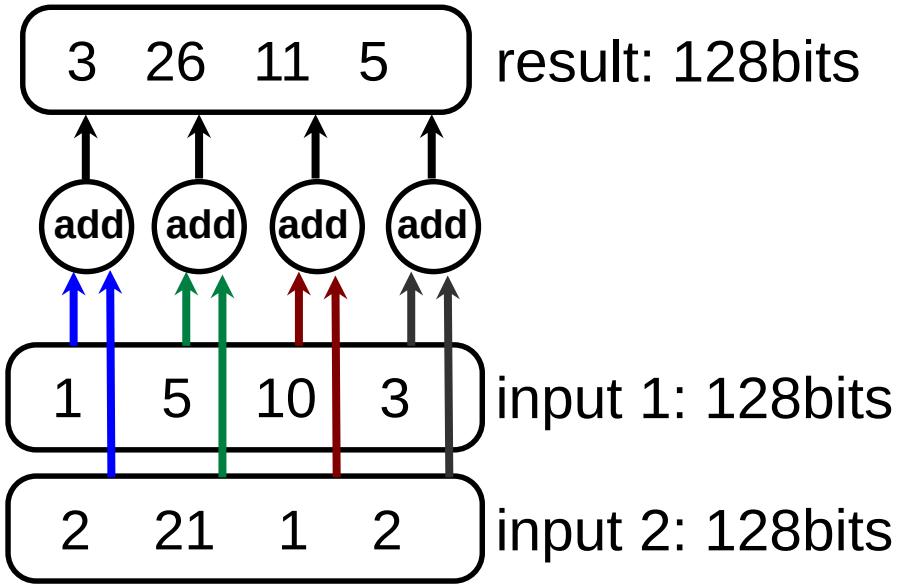


apply the same action on multiple data values
with the same cost as for 1 value

SIMD (single instruction multiple data)

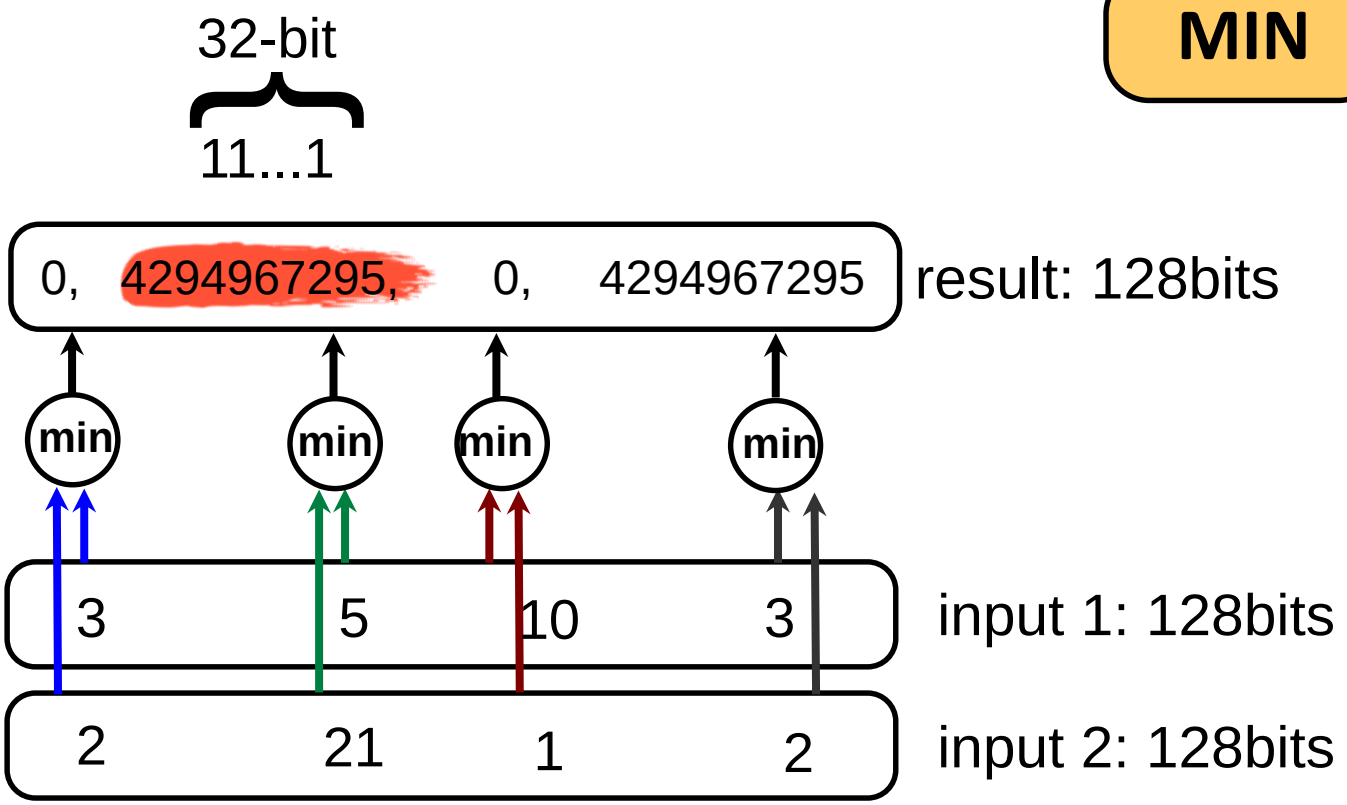
SUM

runs at the same time



SIMD (single instruction multiple data)

MIN



runs at
the same time

SIMD (single instruction multiple data)

SUM

[SIGMOD02b]

How:

assembly or compilers provide special commands

```
for(i=0;i<N;i++)
res+=a[i]
```

```
for (i=0;i<N;i+=4)
res[i,i+1,i+2,i+3]=SIMD_add(res[i,i+1,i+2,i+3], a[i,i+1,i+2,i+3])
```

\+ corner cases

ignoring the for-loop code

we will do 4 times less instructions

SIMD (single instruction multiple data)

SUM

[SIGMOD02b]

```
for (i=0;i<N;i+=4)
res[i,i+1,i+2,i+3]=SIMD_add(res[i,i+1,i+2,i+3], a[i,i+1,i+2,i+3])
```

what is next?

SIMD (single instruction multiple data)

SUM

[SIGMOD02b]

```
for (i=0;i<N;i+=4)
res[i,i+1,i+2,i+3]=SIMD_add(res[i,i+1,i+2,i+3], a[i,i+1,i+2,i+3])
```

what is next?

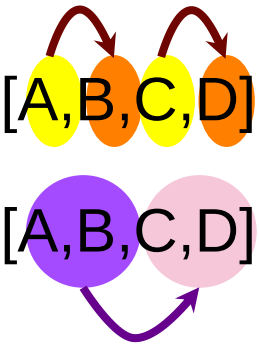
CPU registers

SIMD registers

shuffle ⤮

SIMD_shuffle32: [A,B,C,D]->[B,A,D,C]

SIMD_shuffle64: [A,B,C,D]->[C,D,A,B]



SIMD (single instruction multiple data)

SUM

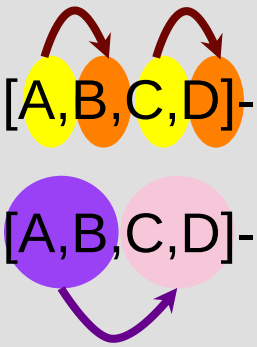
[SIGMOD02b]

```
for (i=0;i<N;i+=4)
res[i,i+1,i+2,i+3]=SIMD_add(res[i,i+1,i+2,i+3], a[i,i+1,i+2,i+3])
```

what is next?

SIMD_shuffle32: [A,B,C,D]->[B,A,D,C]

SIMD_shuffle64: [A,B,C,D]->[C,D,A,B]

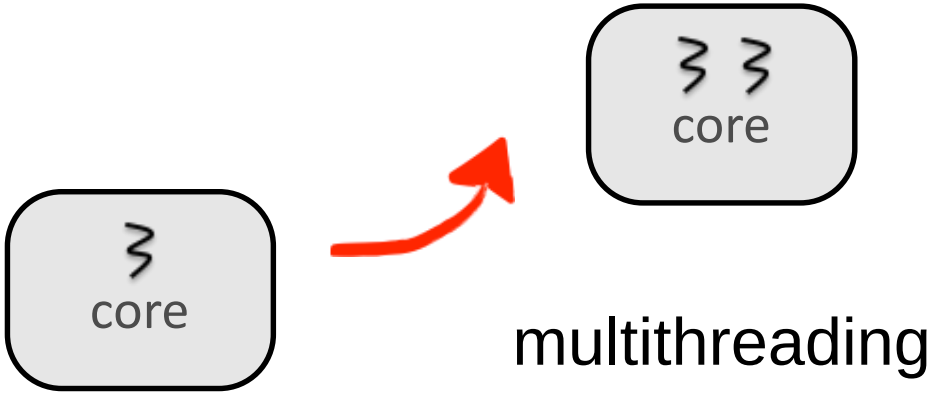


```
t1=SIMD_shuffle32(res)
t2=SIMD_add(res,t1)
t3=SIMD_shuffle64(t2)
res=SIMD_add(t2,t3)
```

SIMD (single instruction multiple data)

column-store model helps as data is already packed in dense arrays

modern parallelism



multithreading

SMT

instruction & data parallelism

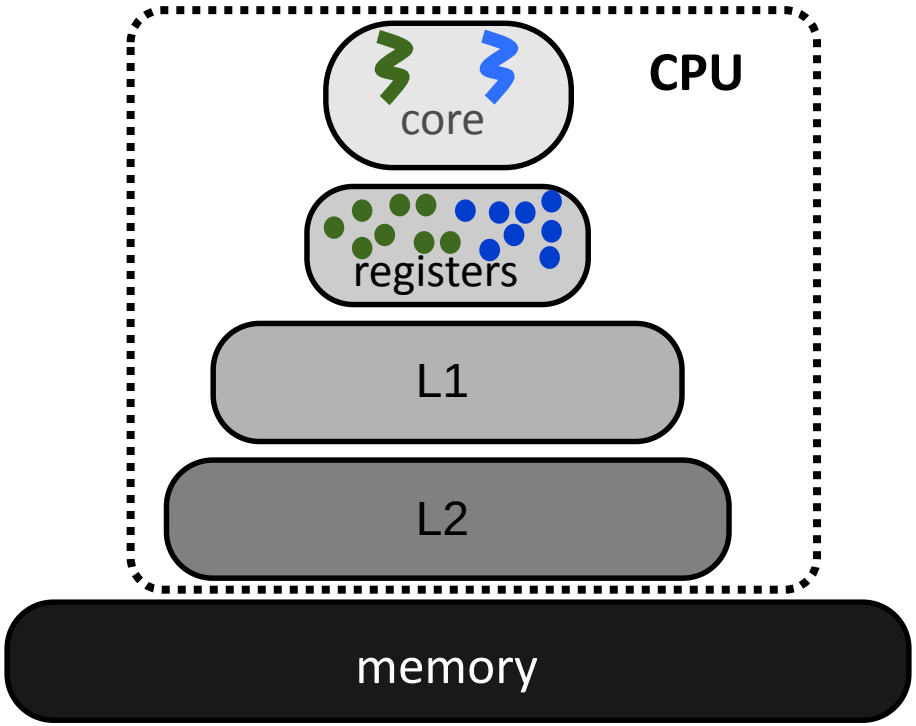
pipelining

superscalar

SIMD

SMT (simultaneous multithreading)

A SMT processor pretends to be multiple *logical* processors (one per instruction stream).

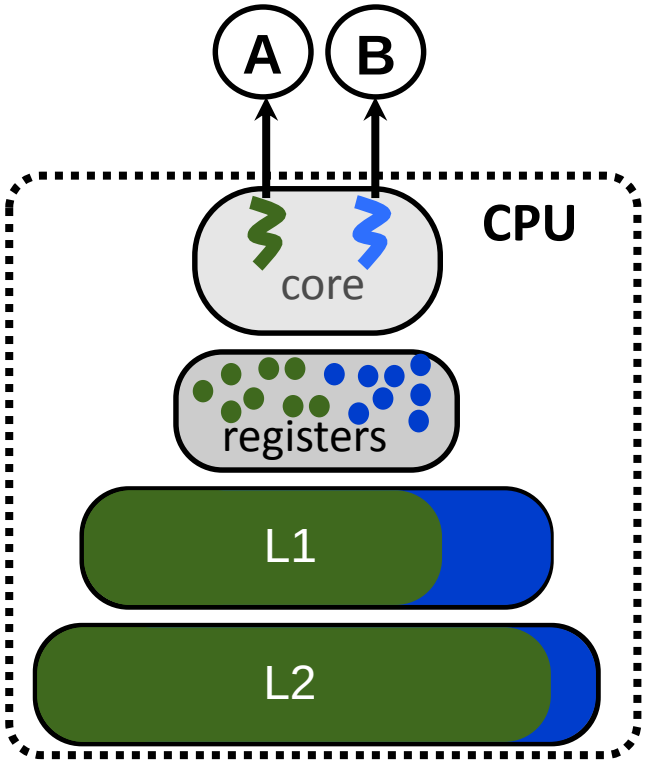


“30% performance gain” --Intel

if one thread stalls another one can continue

SMT – treat logical as physical

[VLDB05b]

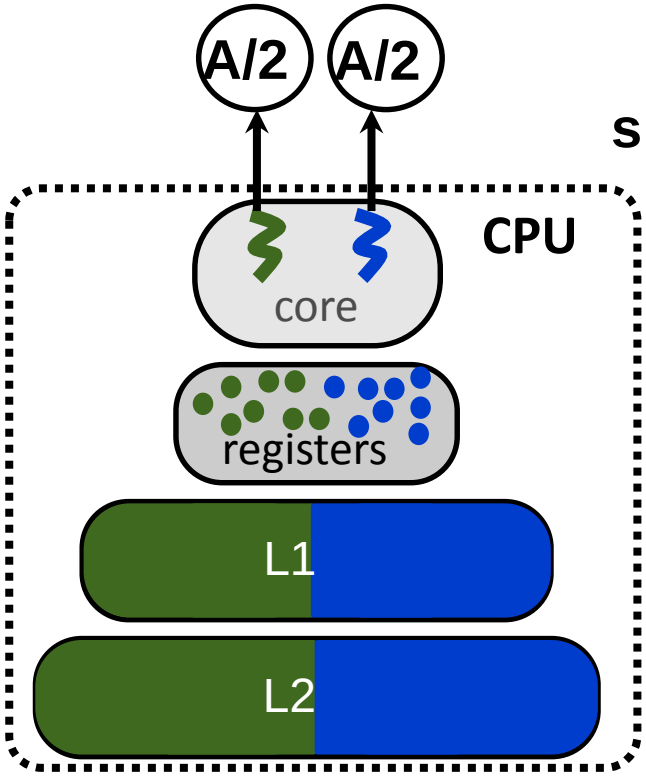


minimal code changes

ignorance of resource sharing

competition for execution units

SMT – multithreaded operators



[VLDB05b]

share input and output data in the cache

read
- odd tuples
- even tuples

write
- separate output buffers
- \+ merging step

⚠ *no longer preserving the order of input records*

✓ beneficial for instruction & data cache performance

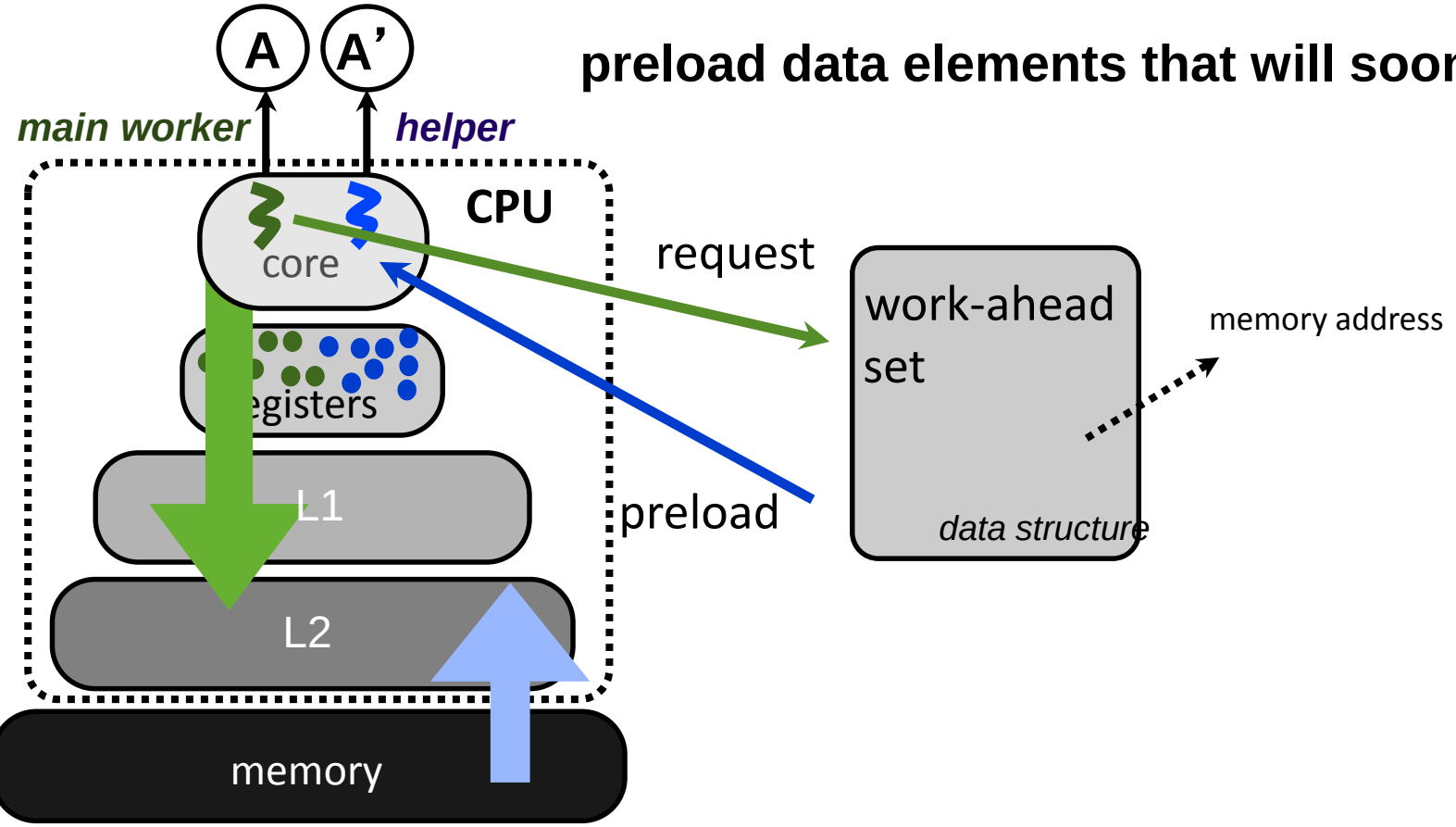
✗ reimplementation of dbms operators

⚠ partitioning and merging

preloading in SMT

[VLDB05b]

preload data elements that will soon be needed

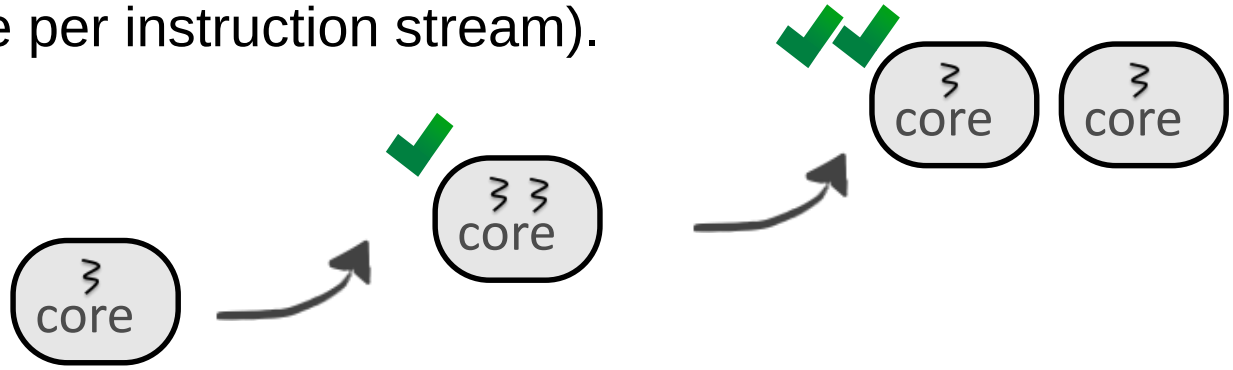


use one thread for the computation

and the other to manage resources

SMT (simultaneous multithreading)

A SMT processor pretends to be multiple *logical* processors (one per instruction stream).



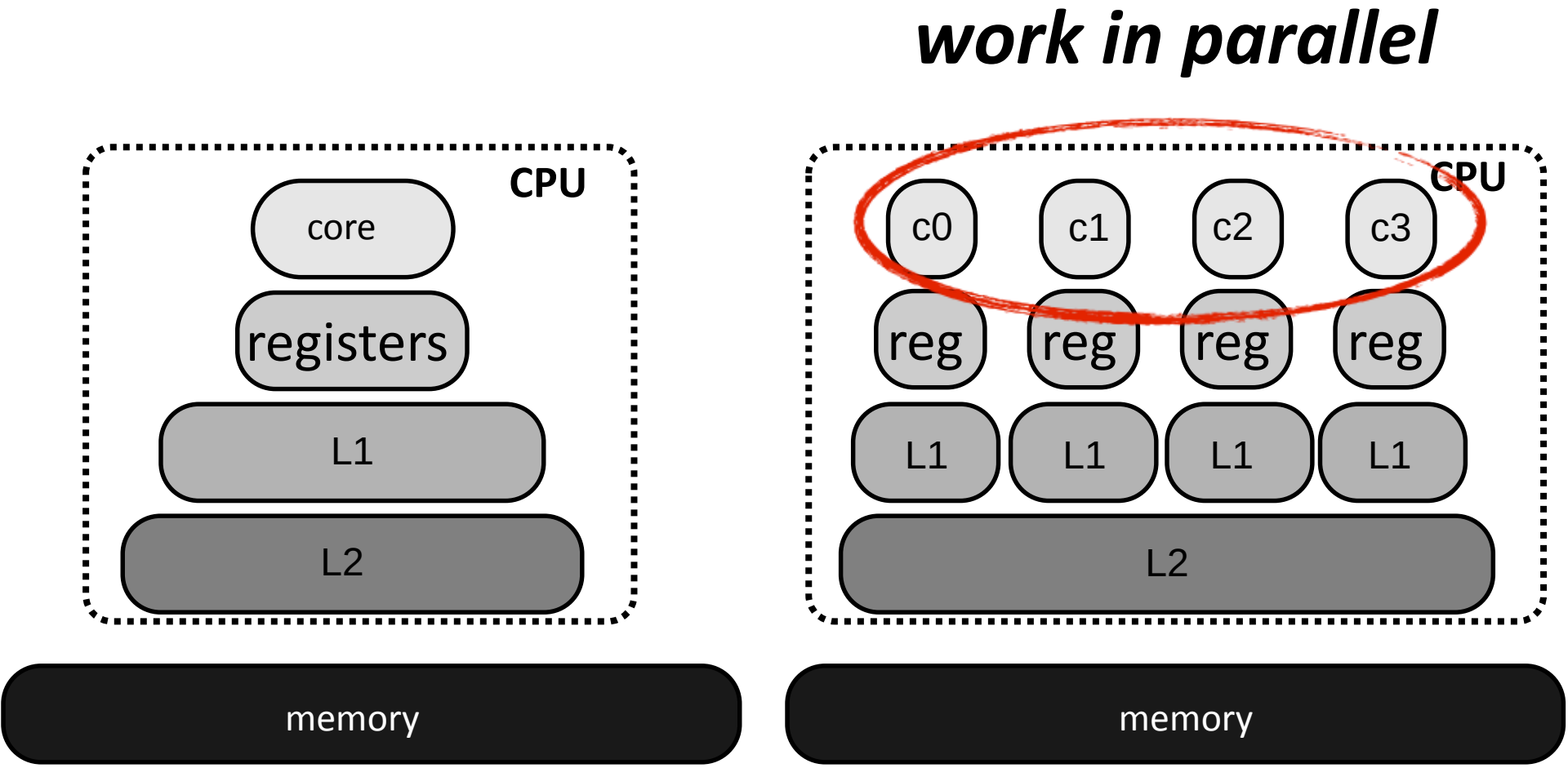
better than single threaded:

- increase thread-level parallelism
- improve processor utilization when one thread blocks

not as good as two physical cores

- cpu resources are shared, not replicated

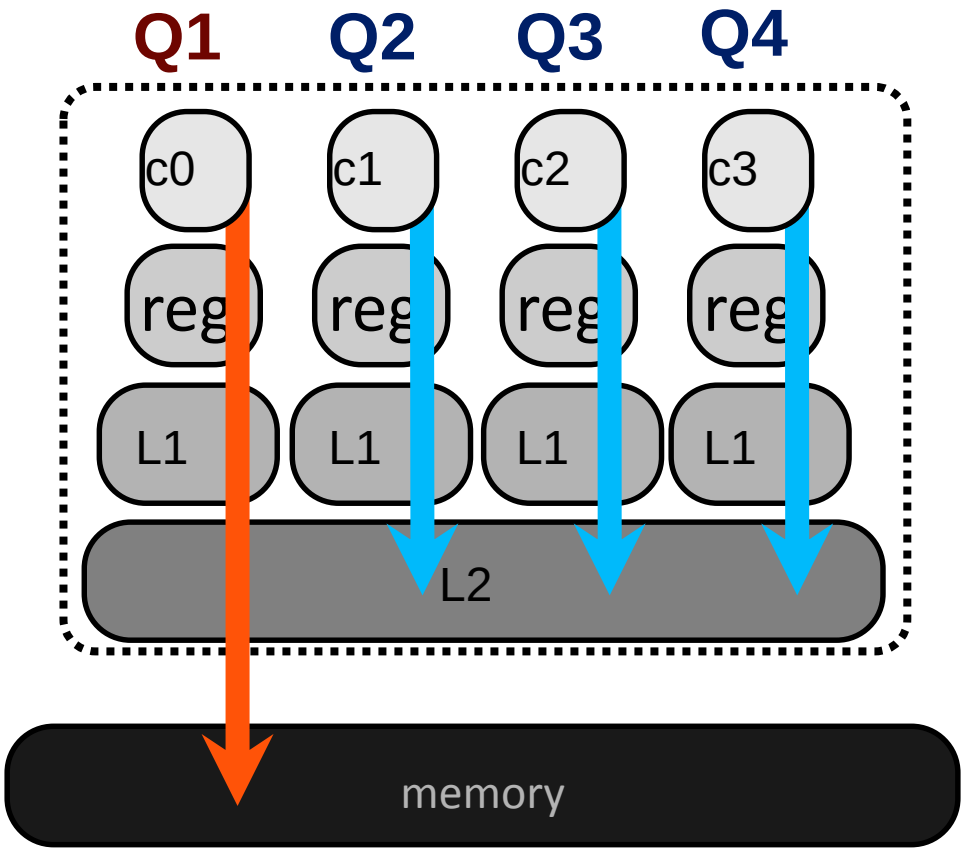
from single core to multi-cores



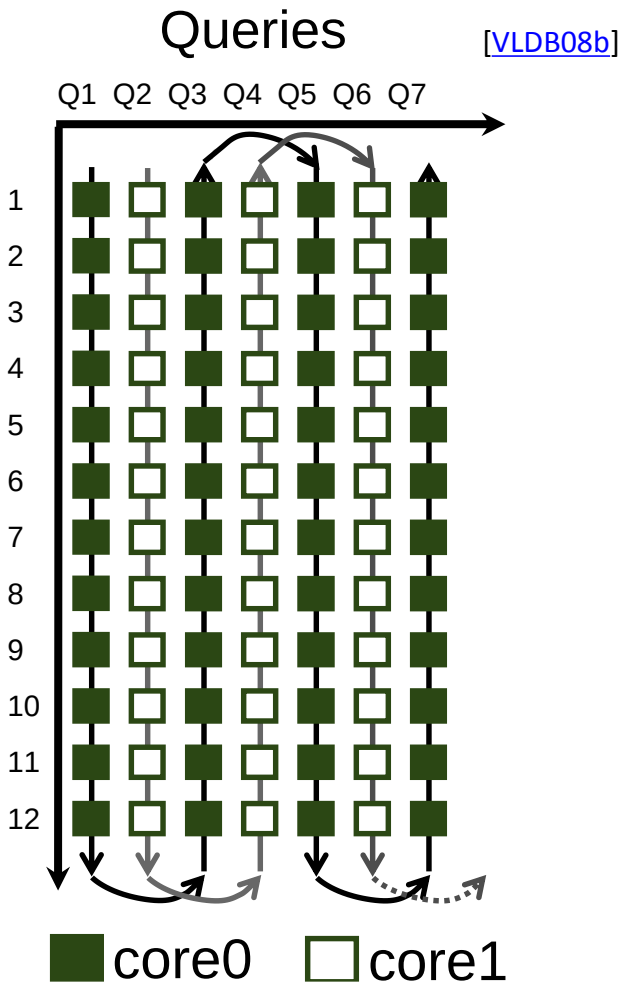
how do we keep cpu at 100% ?

scan in multicores

1 core for each query



Data blocks

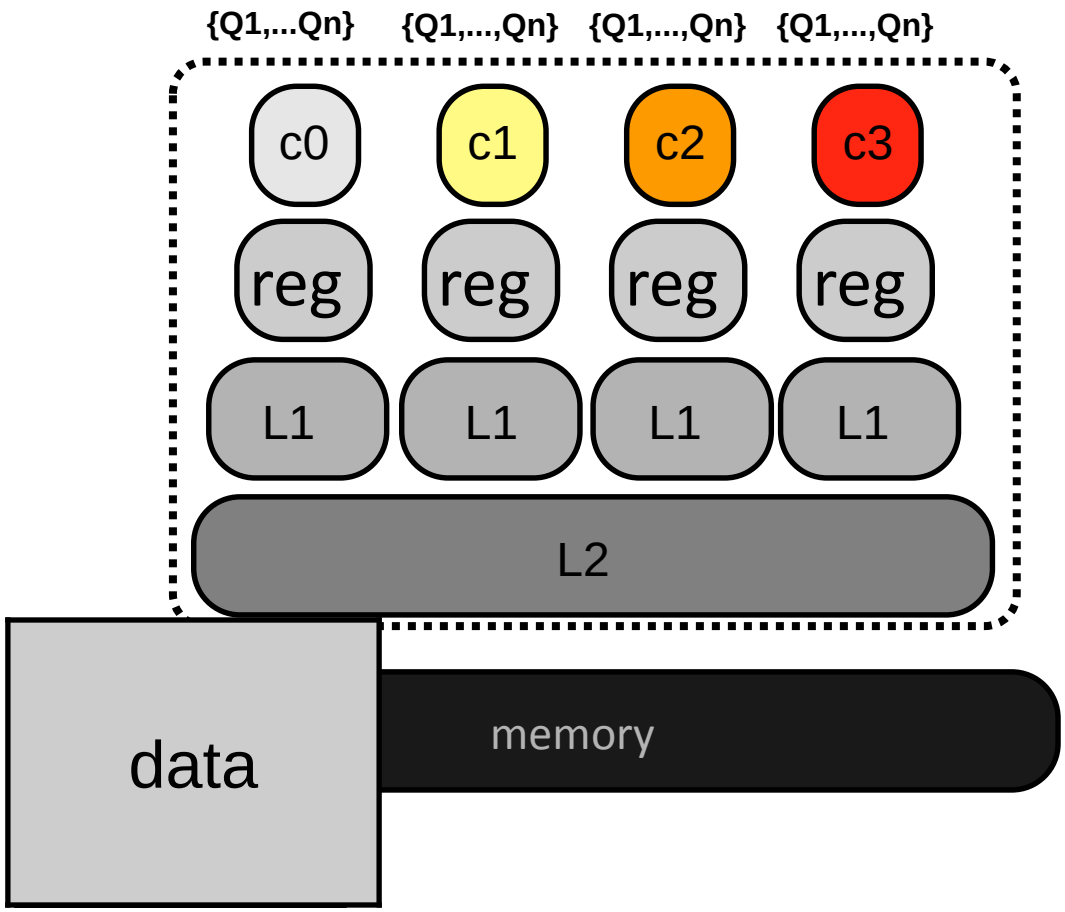


achieve limited I/O sharing via the convoy phenomenon

scan in multicores

1 core for each table scan

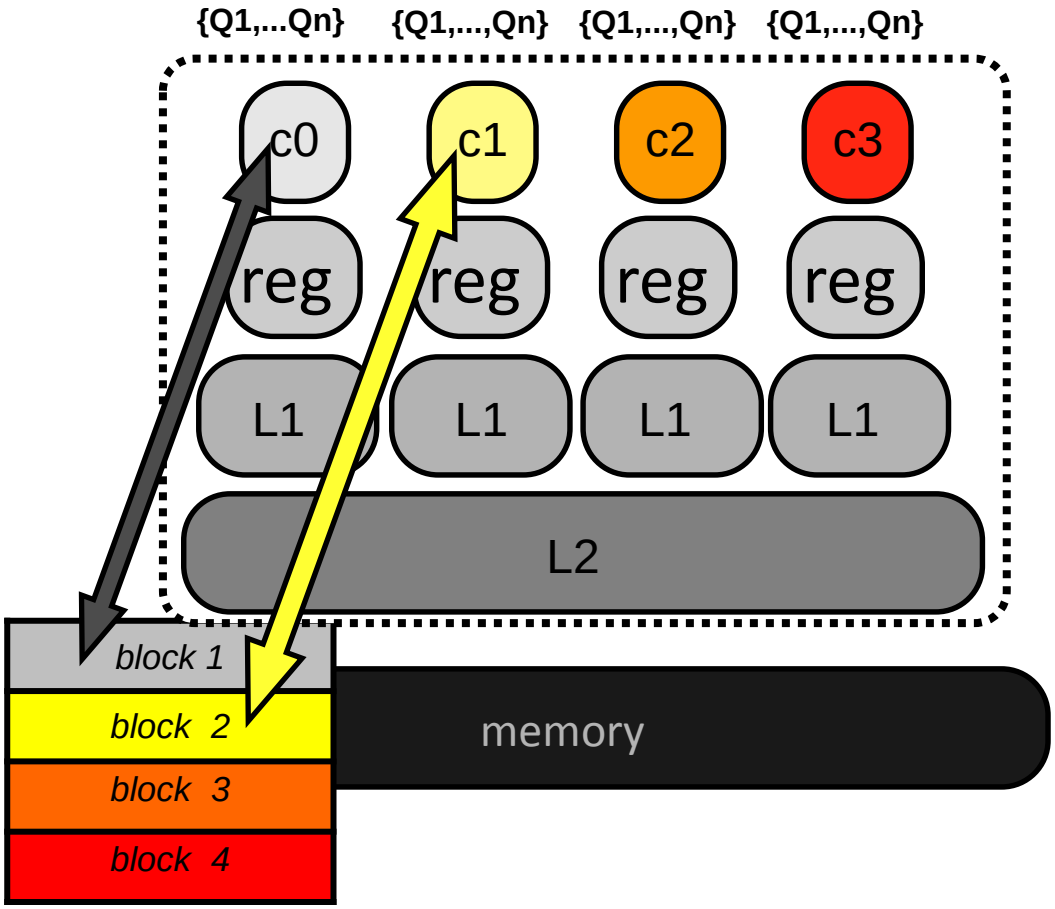
[VLDB08b]

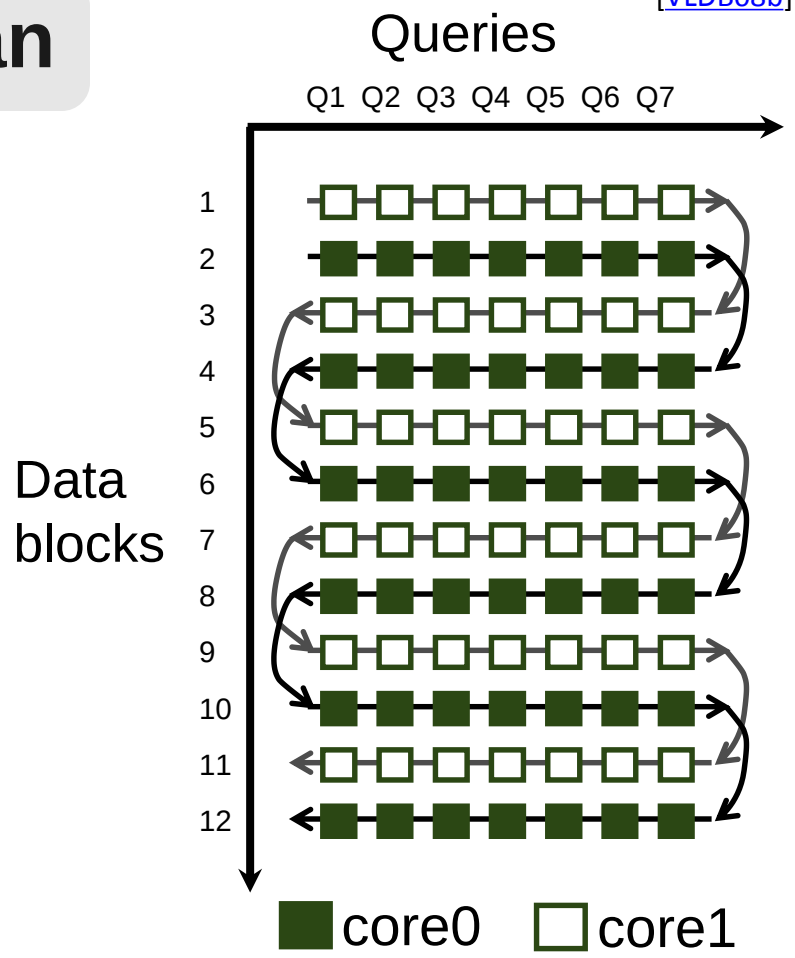


scan in multicores

1 core for each table scan

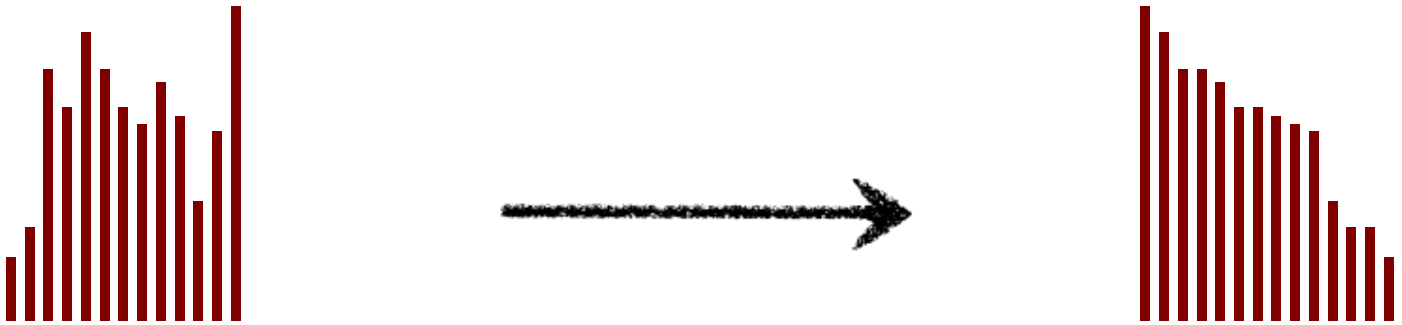
[VLDB08b]

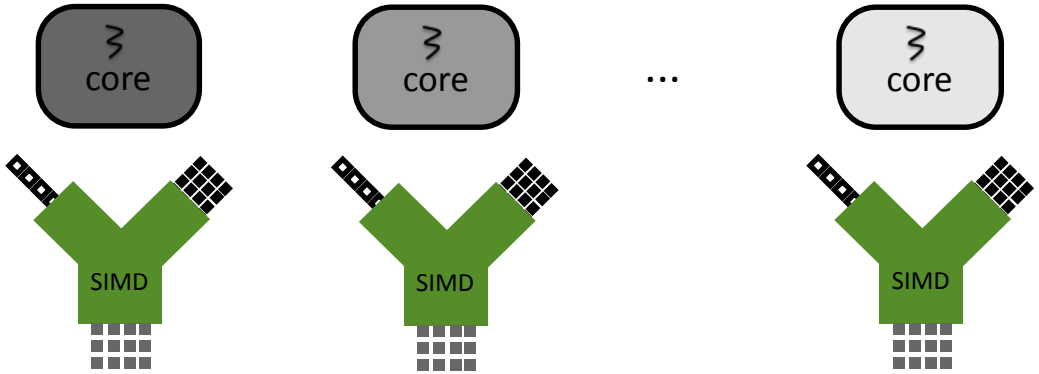




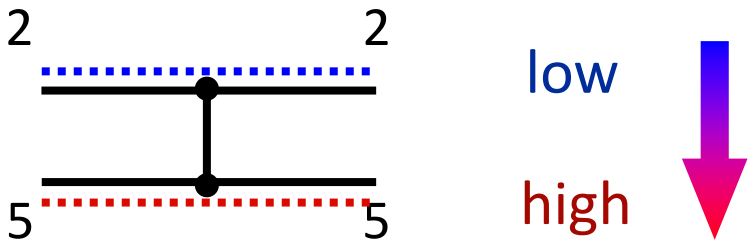
load into caches once + share => reduce cache misses

sorting on multicore SIMD

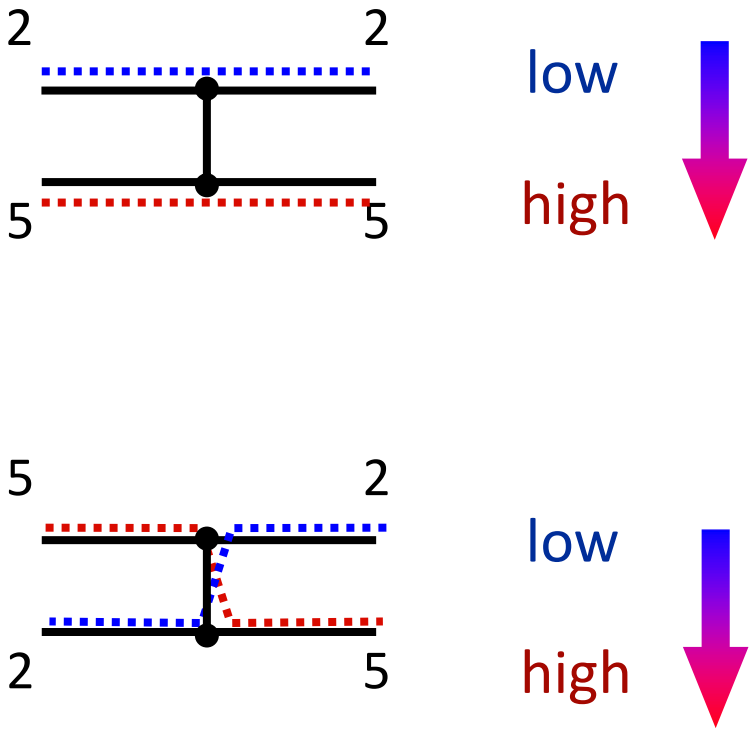




sorting network

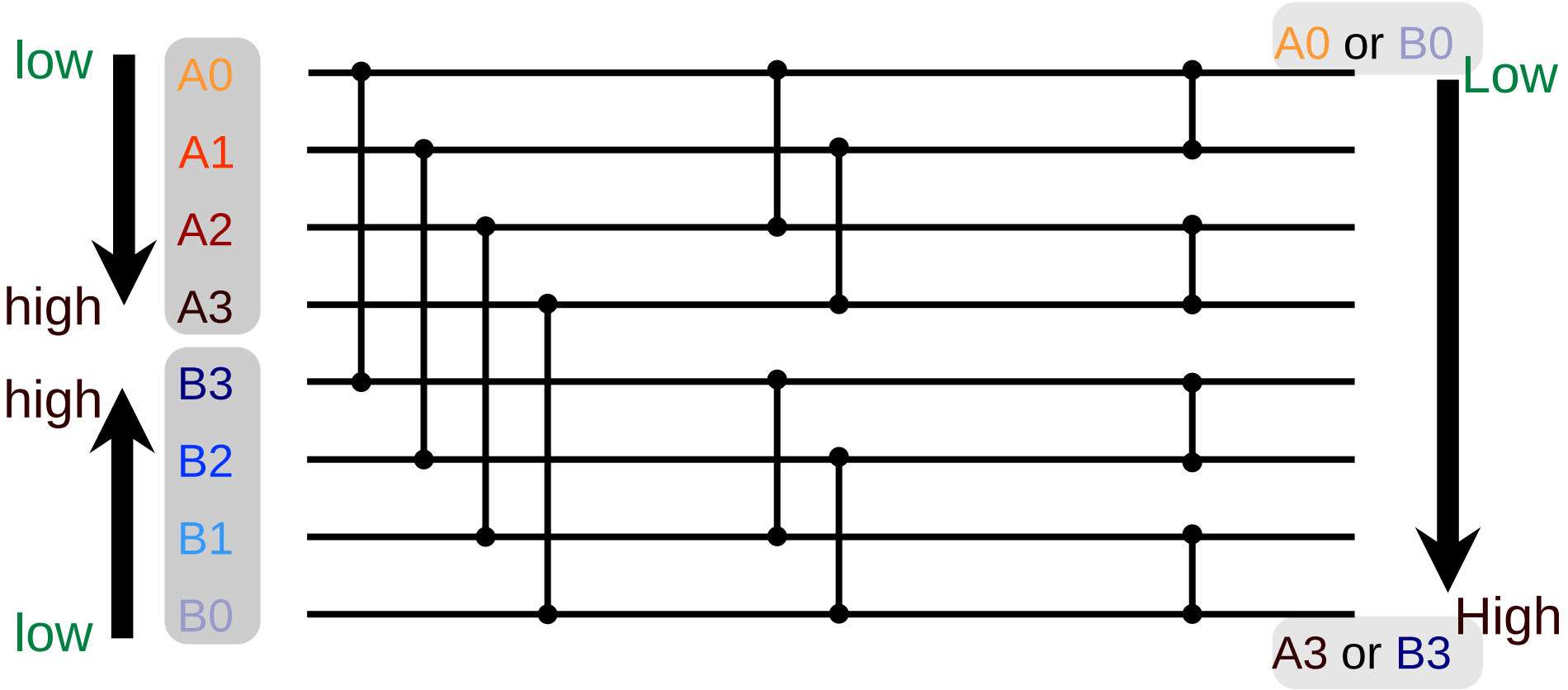


sorting network



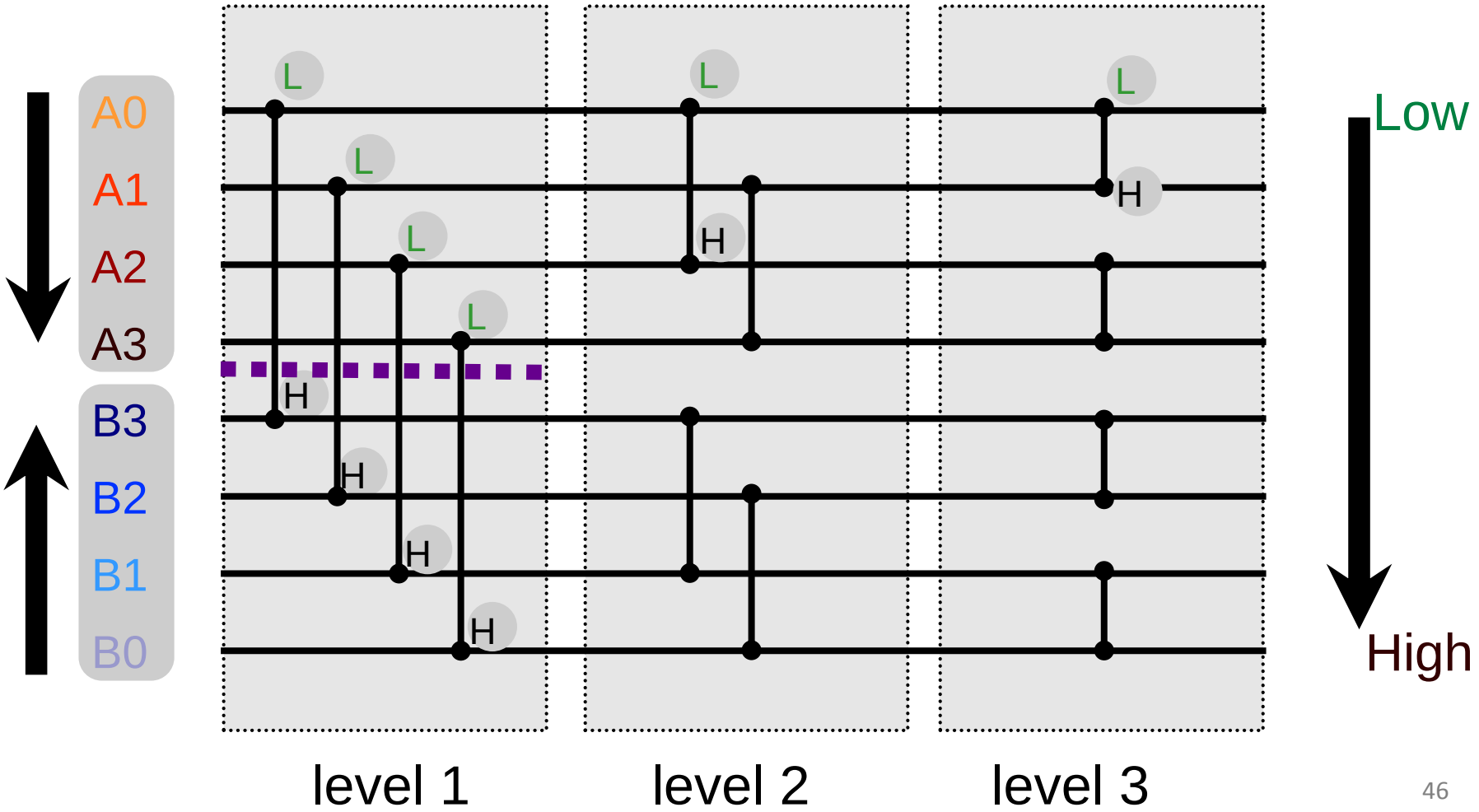
bitonic merge kernel

[VLDB08a]



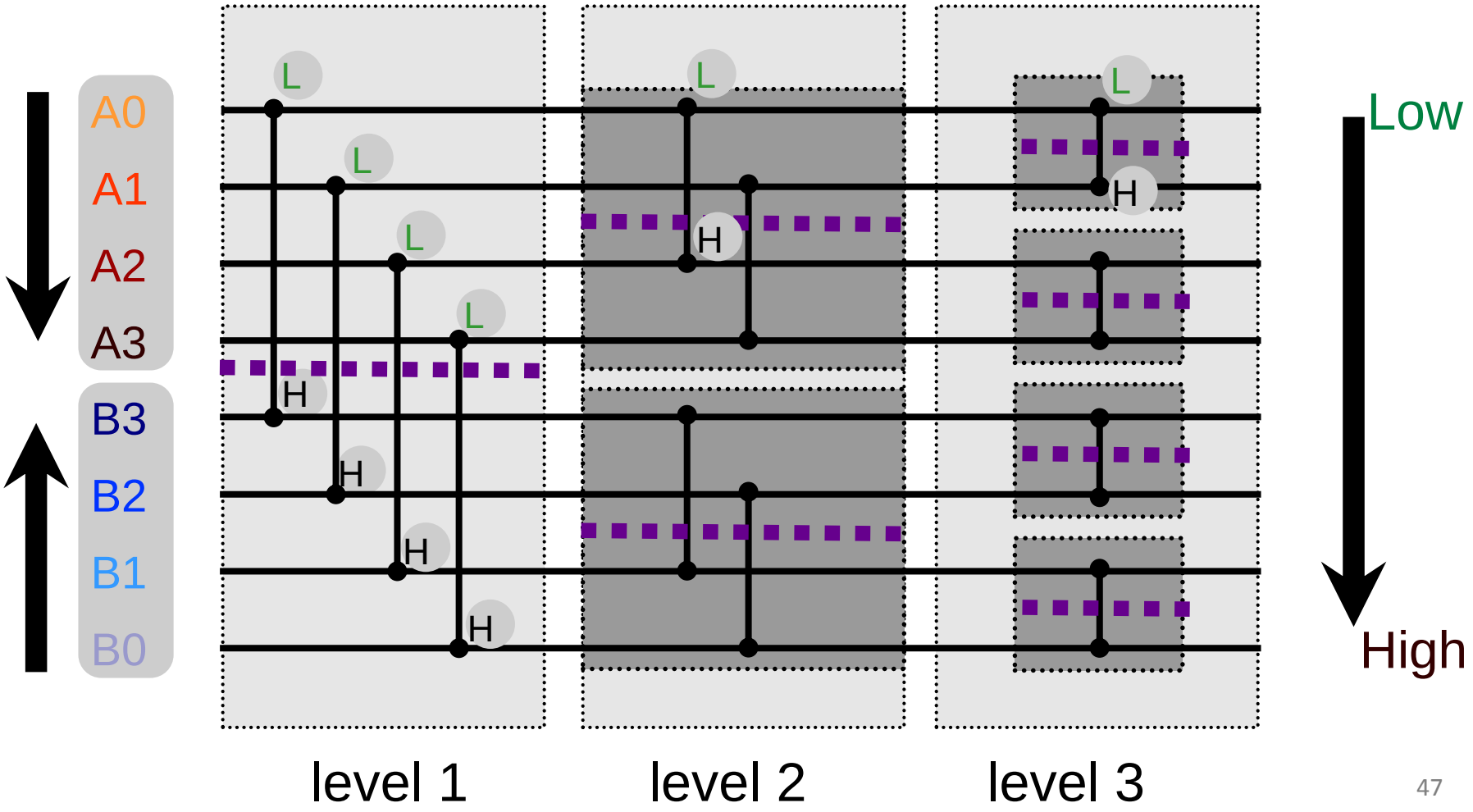
bitonic merge kernel

[VLDB08a]



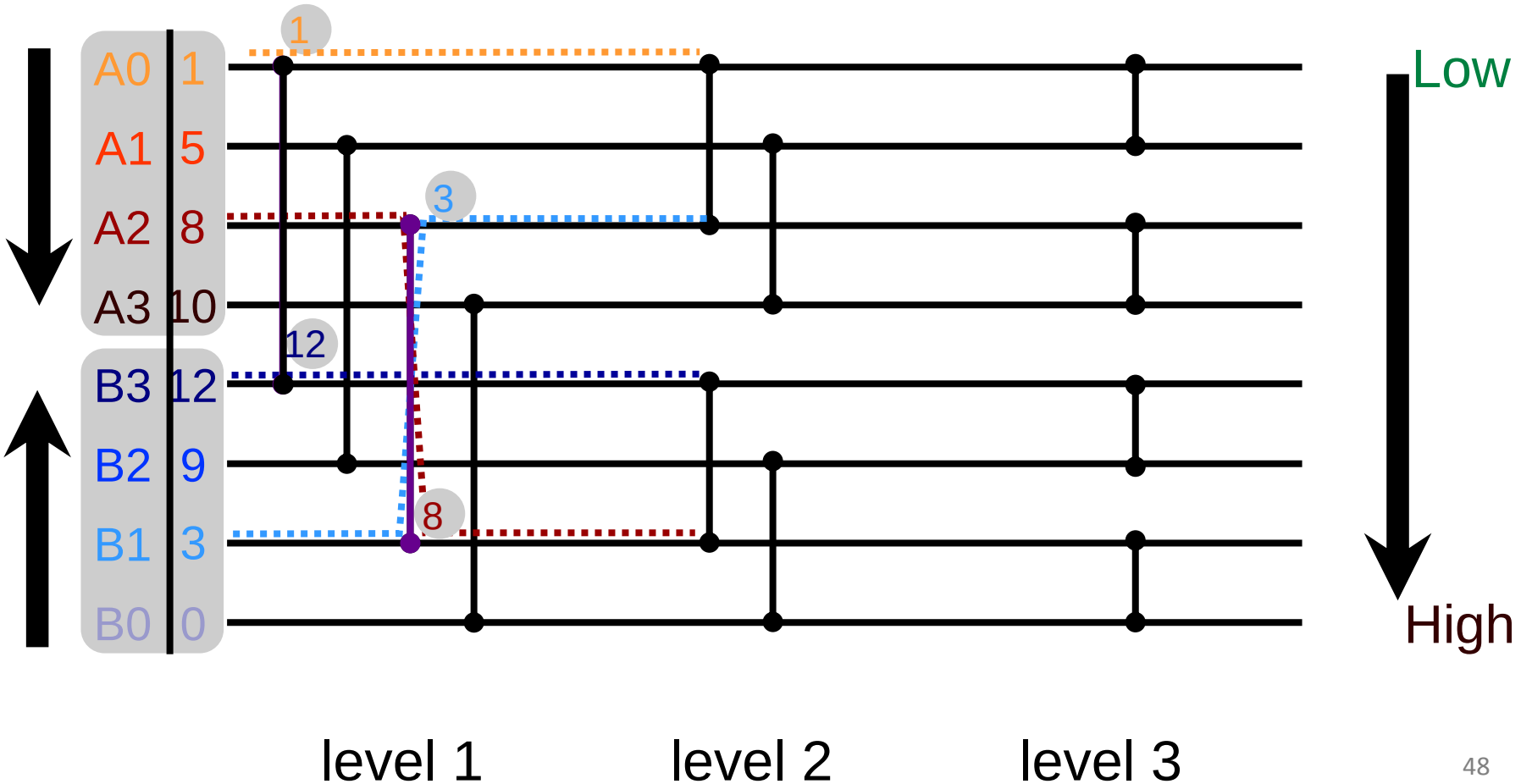
bitonic merge kernel

[VLDB08a]



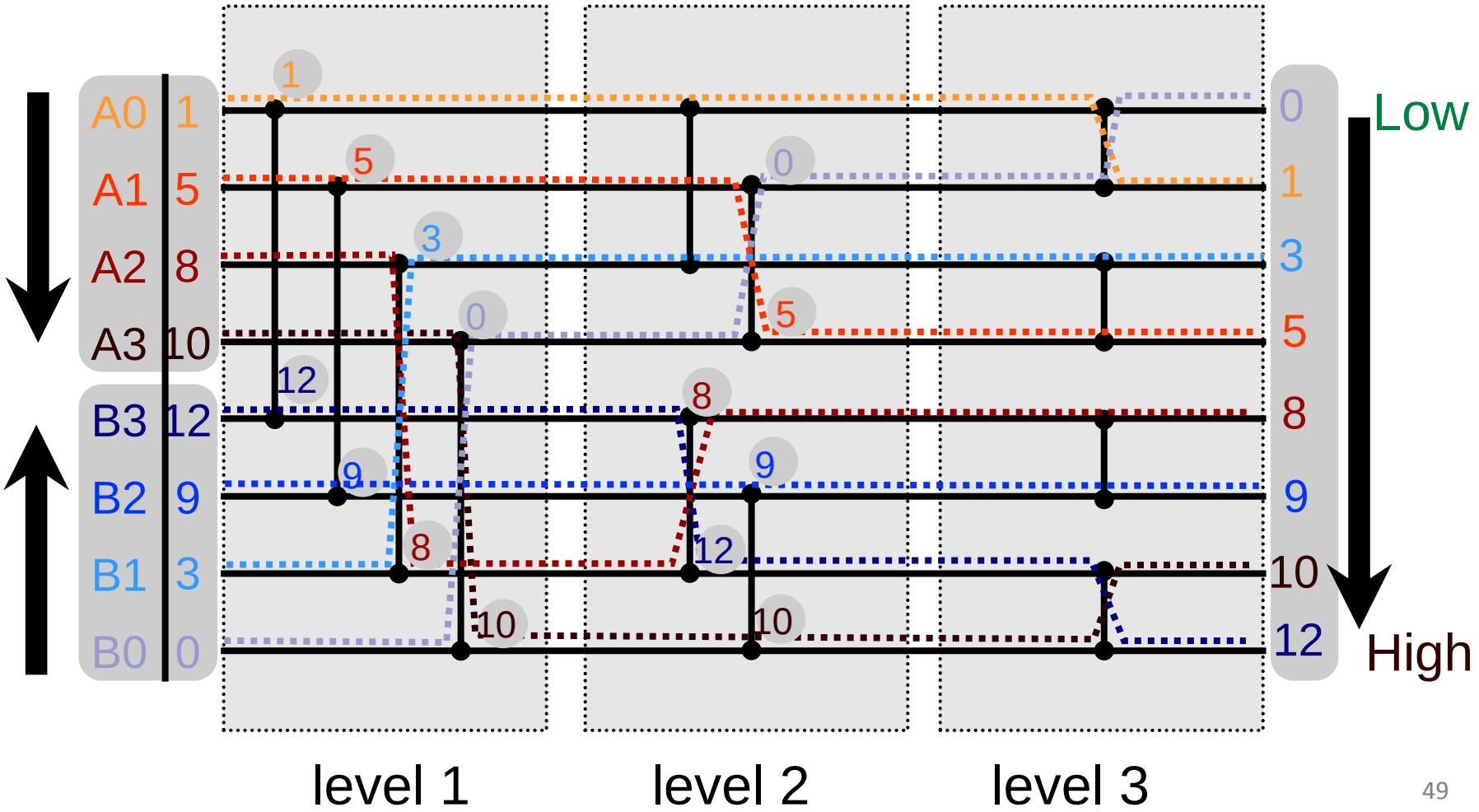
bitonic merge kernel

[VLDB08a]



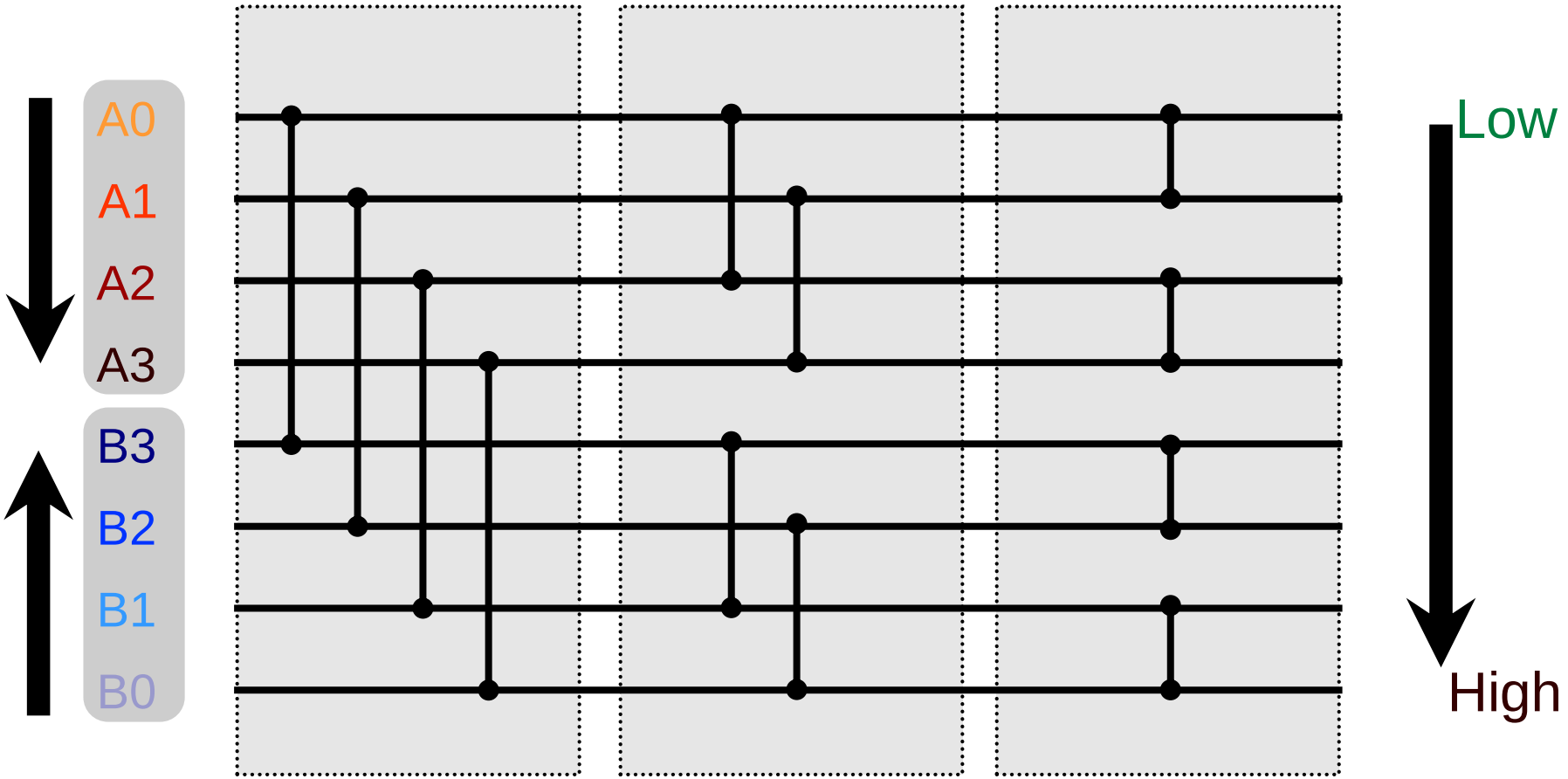
bitonic merge kernel

[VLDB08a]



bitonic merge kernel

[VLDB08a]

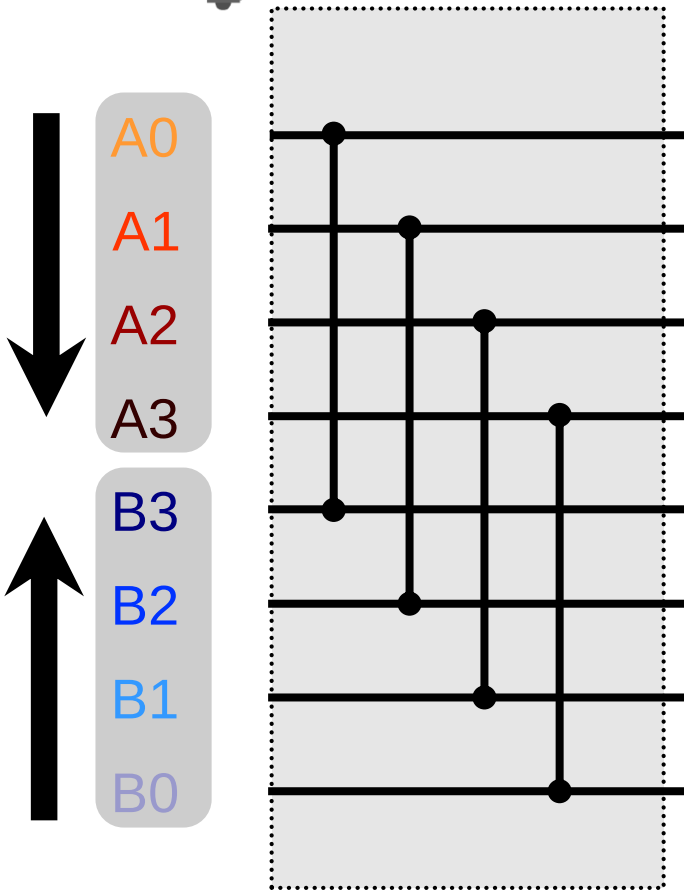


bitonic merge network in SIMD multicores?

bitonic merge kernel with SIMD

[VLDB08a]

simultaneous comparisons



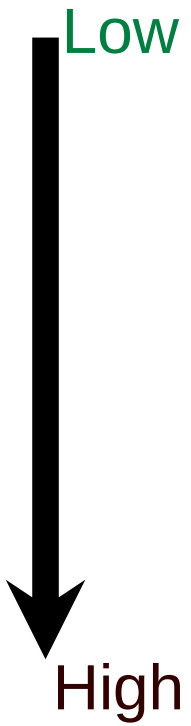
level 1

L1=**SIMD_min**(A,B);

H1=**SIMD_max**(A,B);

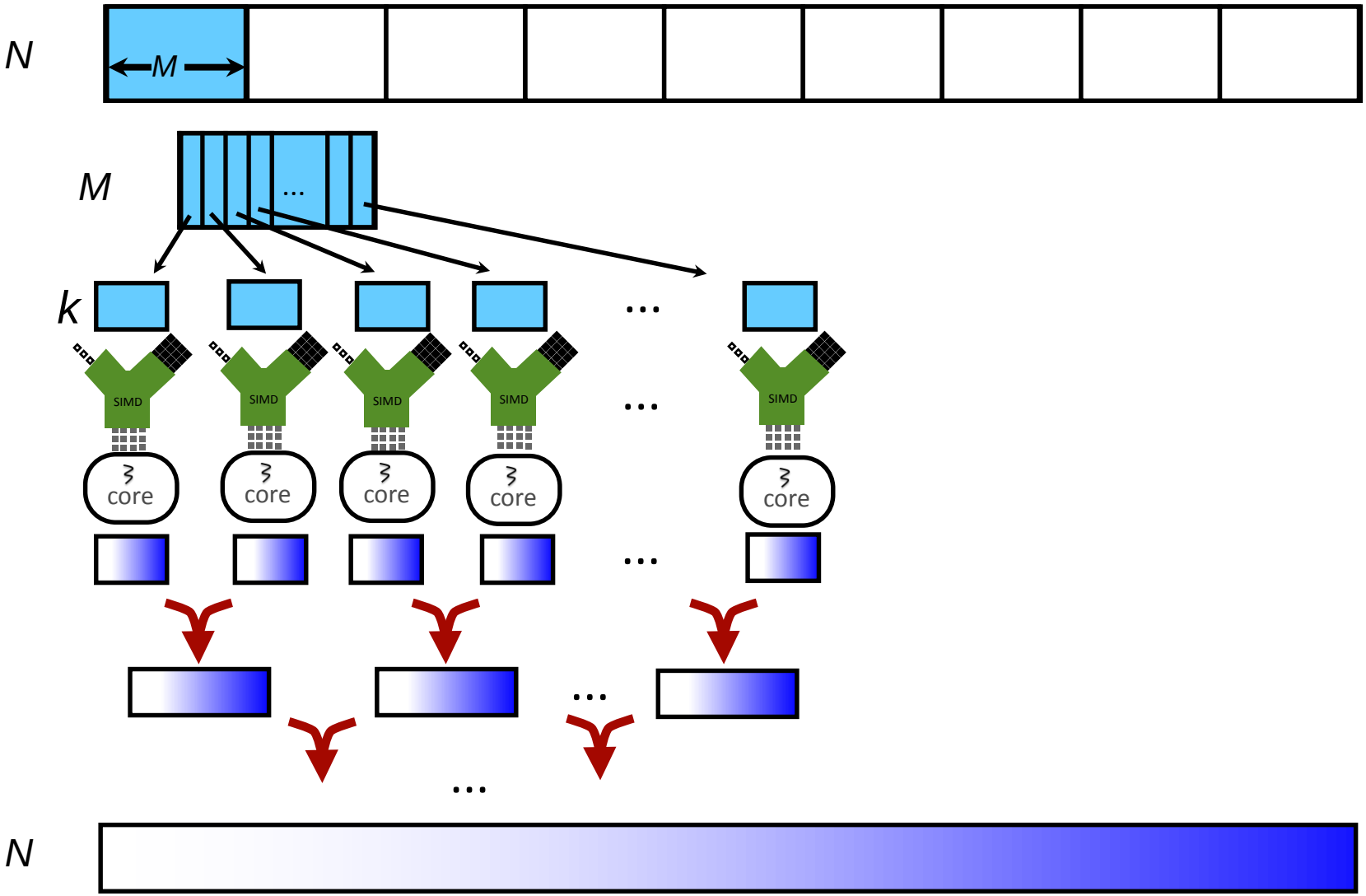
L1p=**SIMD_shuffle**(L1);

H1p=**SIMD_shuffle**(H1);



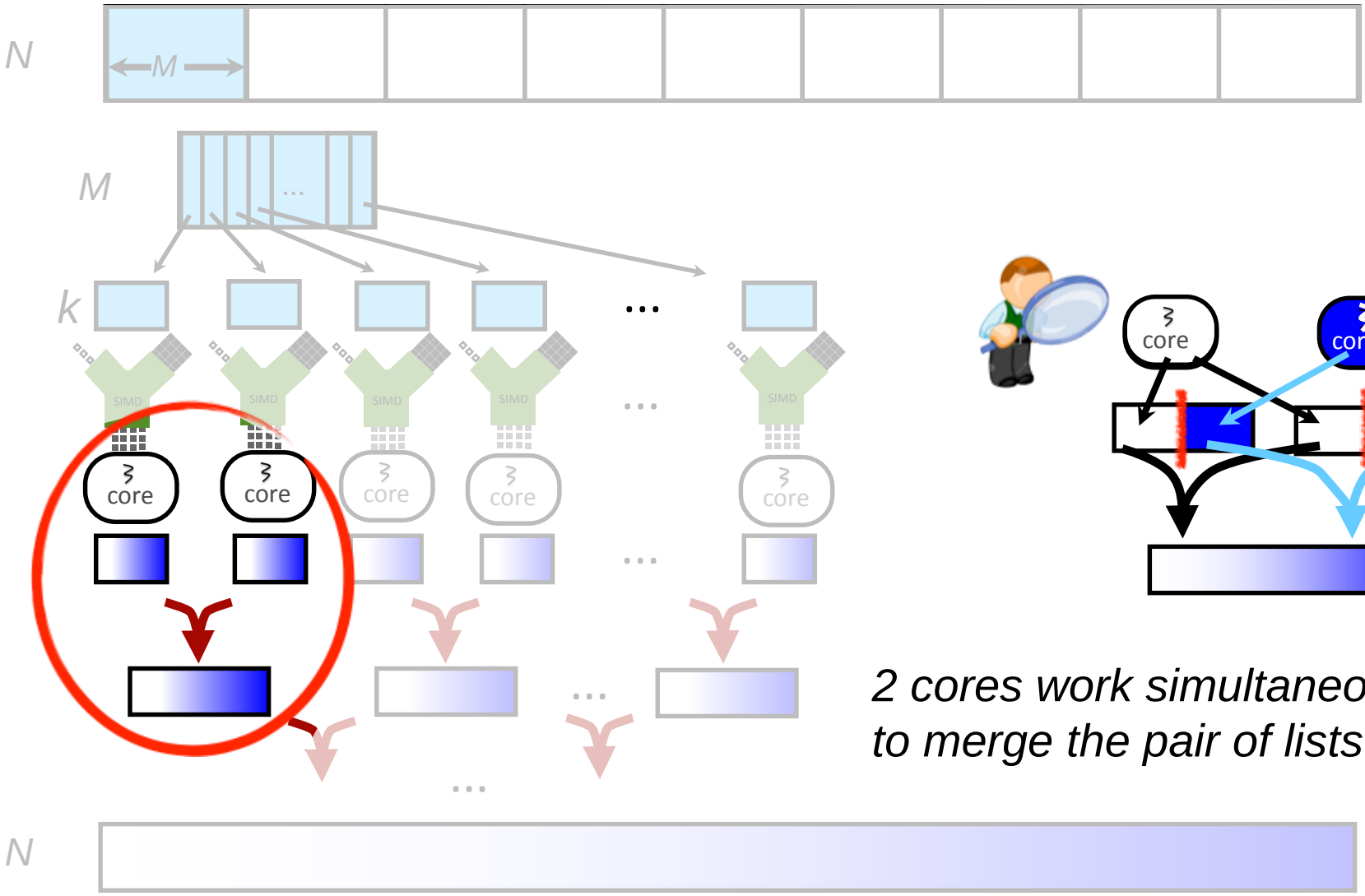
sorting on multicore SIMD

[VLDB08a]



sorting on multicore SIMD

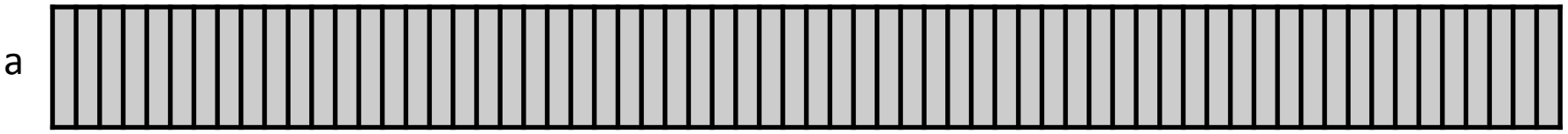
[VLDB08a]



parallel adaptive indexing

[DaMoN14a]

Select max(a)
From R
Where a>v;



parallel adaptive indexing

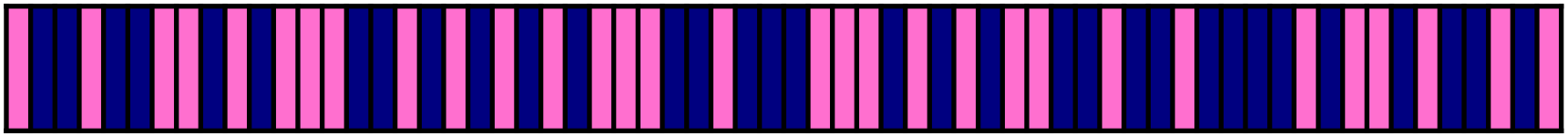
[DaMoN14a]

Select max(a)
From R
Where a>v;

● > v
● < v

uncracked piece

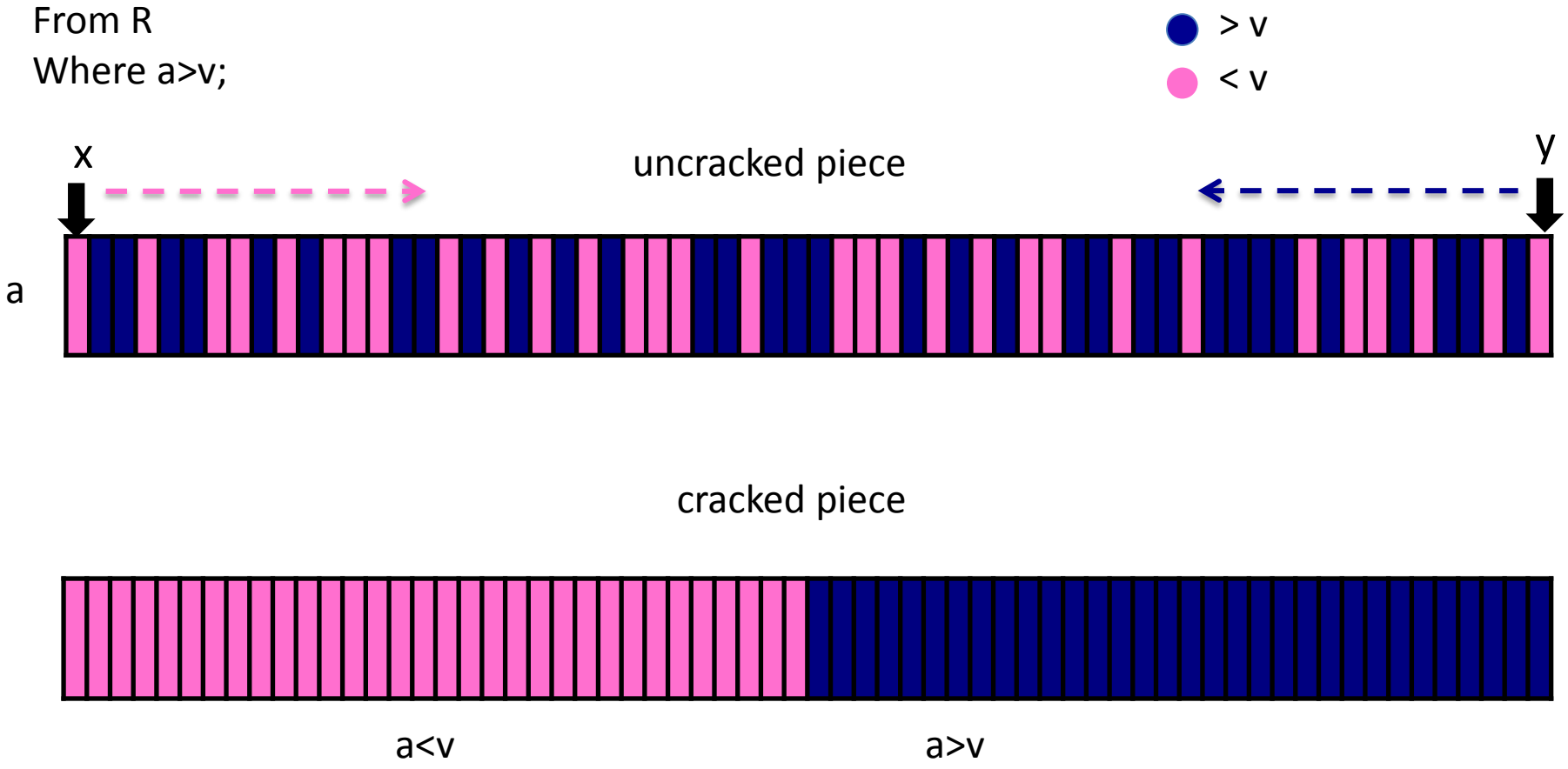
a



parallel adaptive indexing

[DaMoN14a]

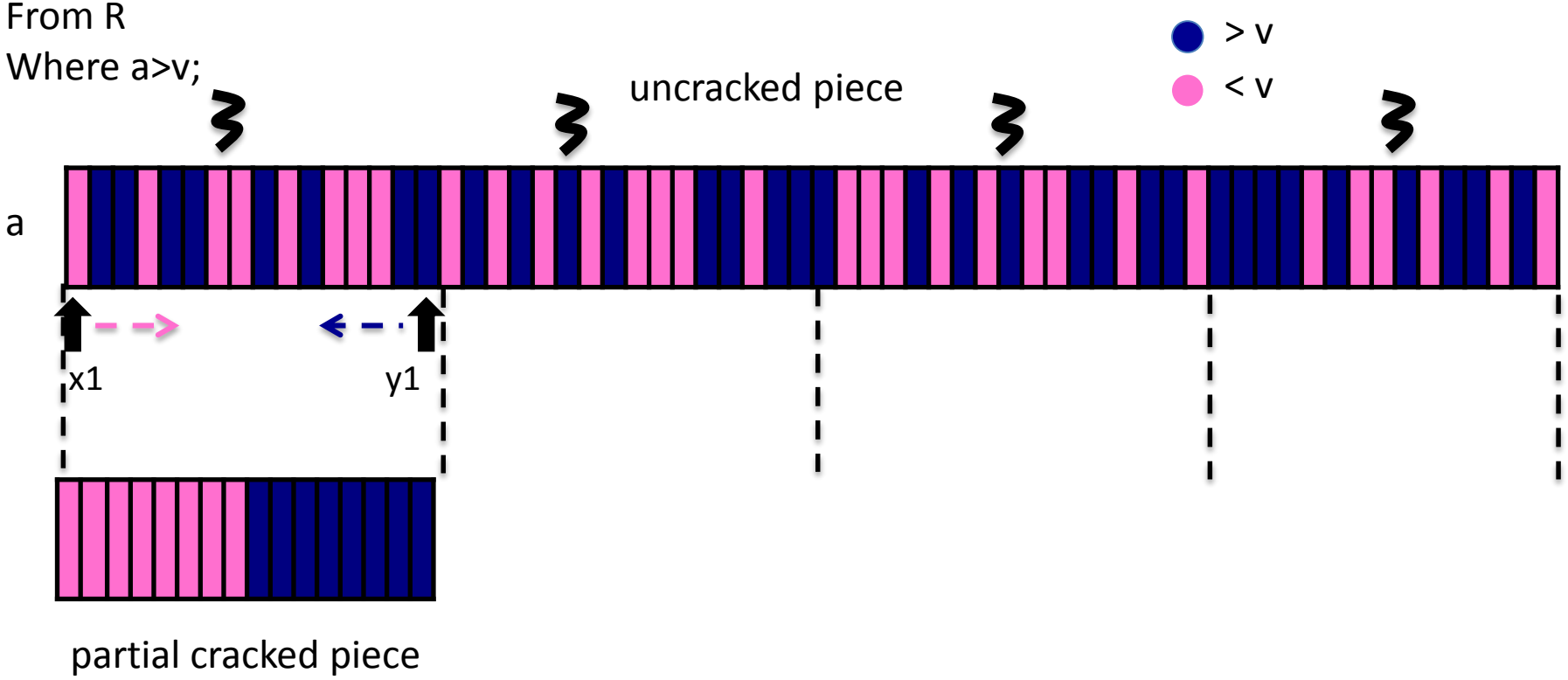
Select max(a)
From R
Where a>v;



parallel adaptive indexing

[DaMoN14a]

Select max(a)
From R
Where a>v;

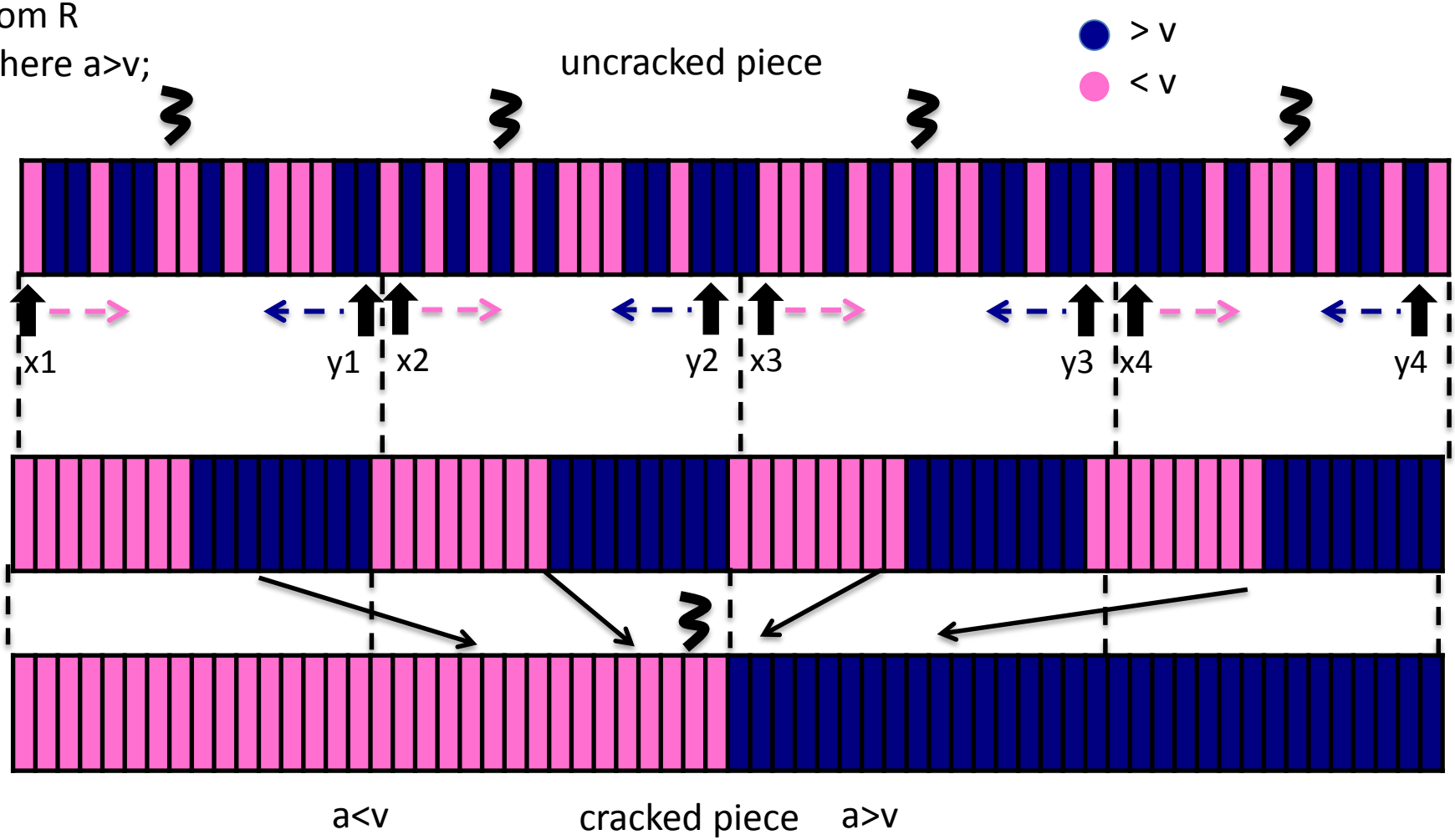


each thread cracks a partition

parallel adaptive indexing

[DaMoN14a]

Select max(a)
From R
Where a>v;

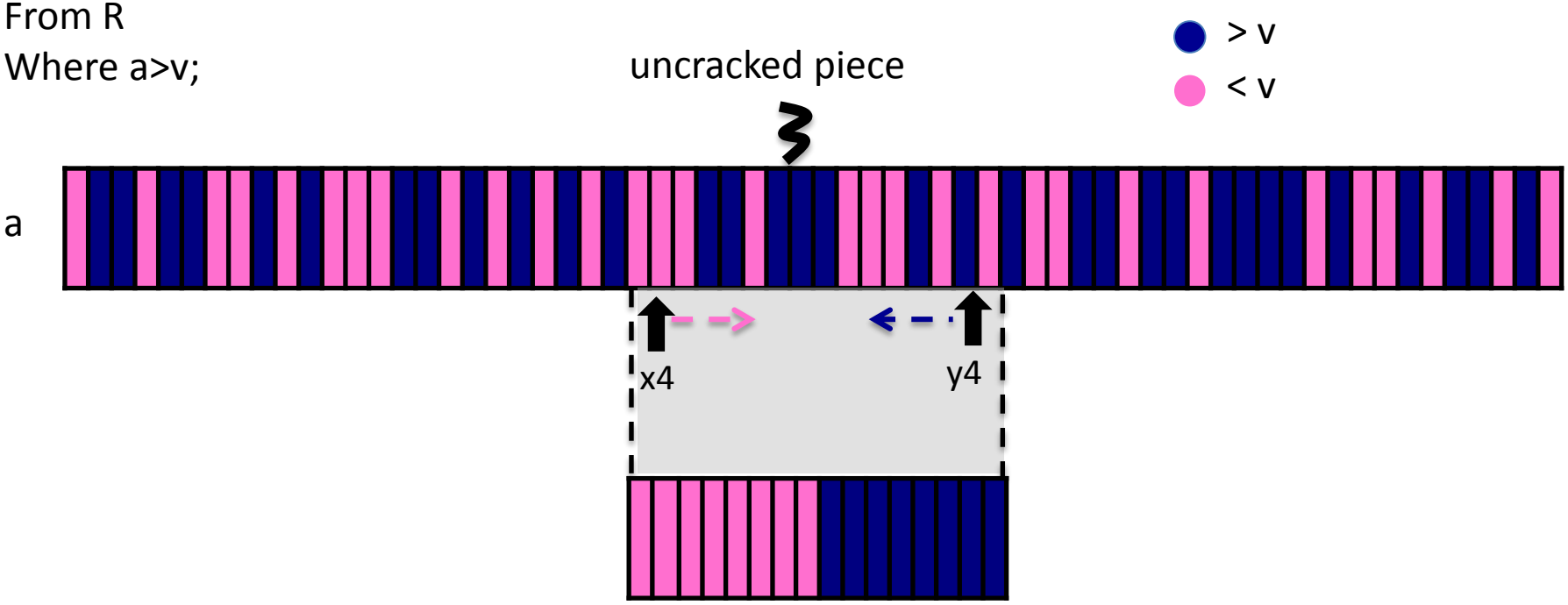


relocation during merge cause many case misses

parallel adaptive indexing

[DaMoN14a]

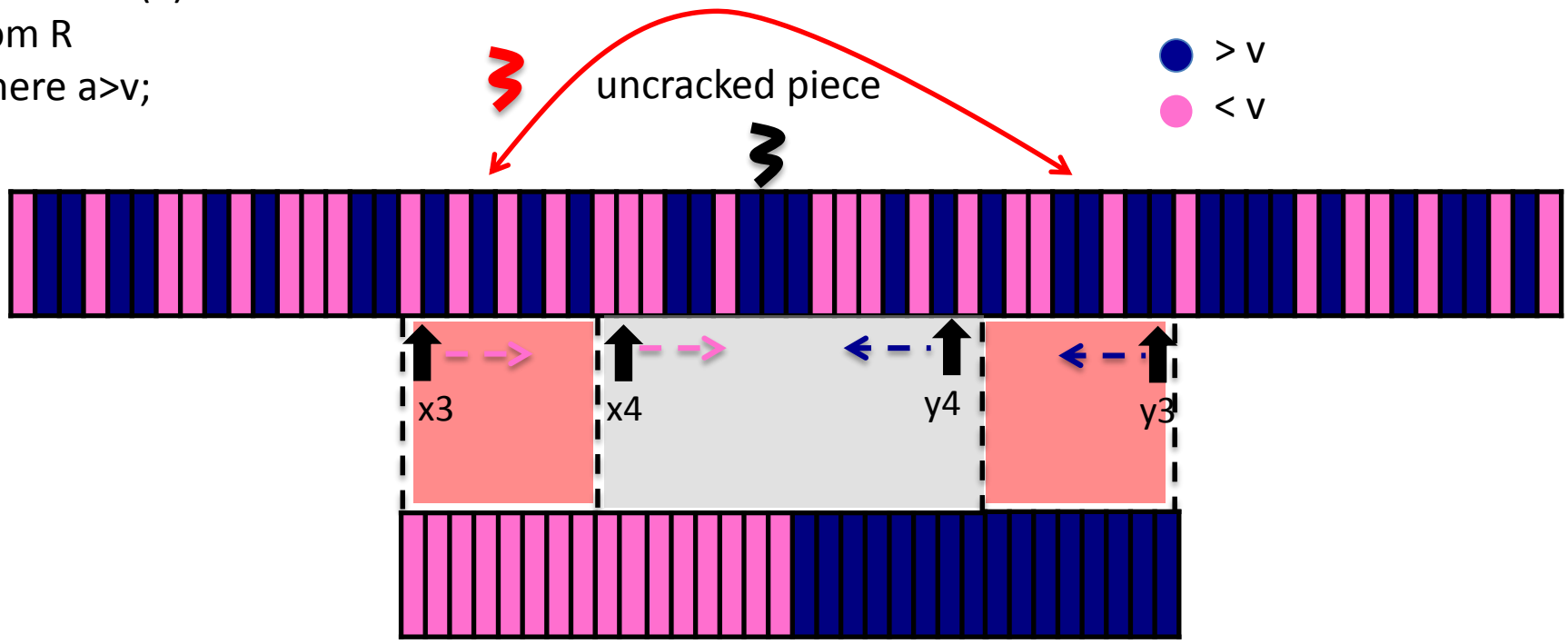
Select max(a)
From R
Where a>v;



parallel adaptive indexing

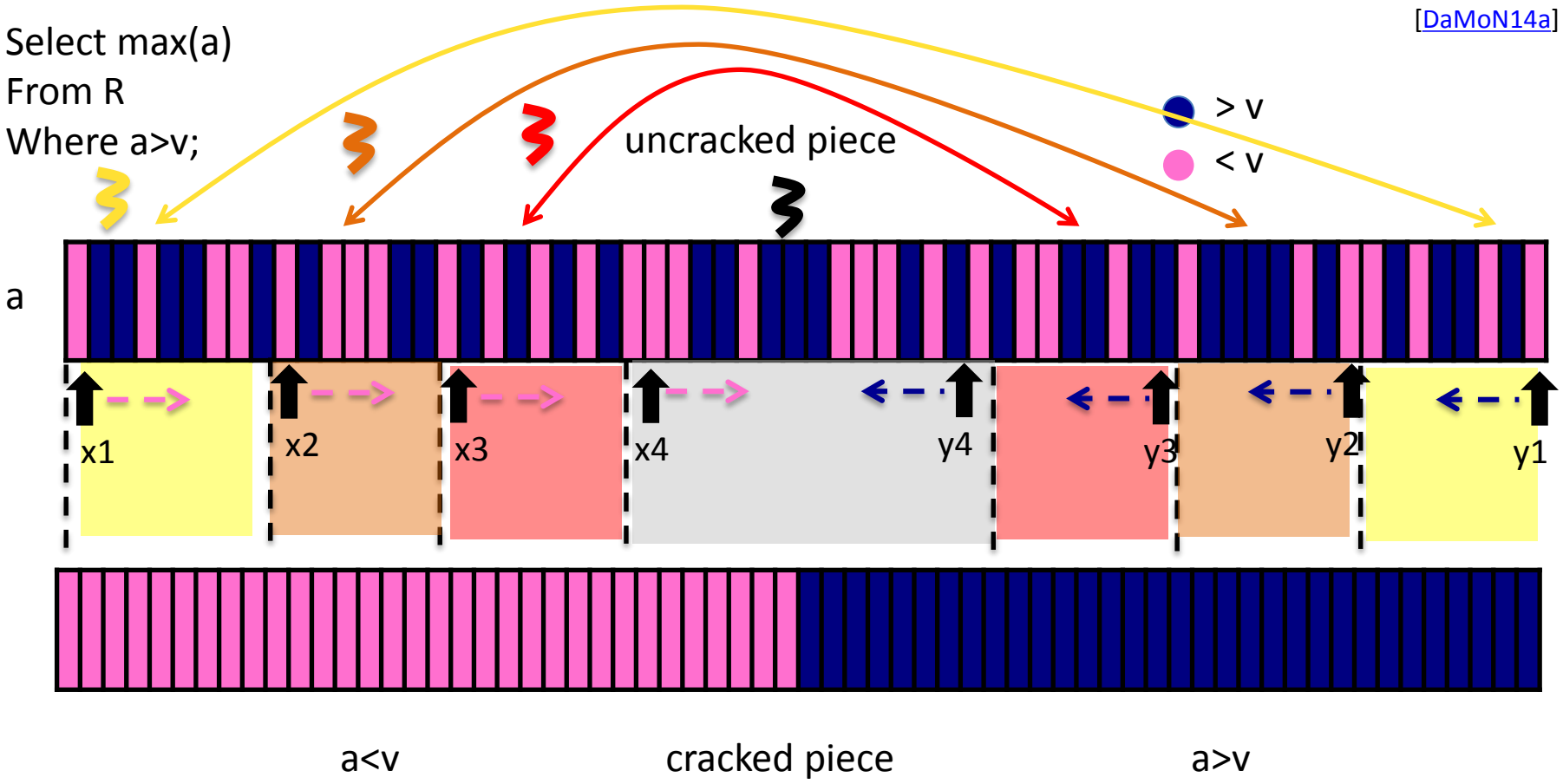
[DaMoN14a]

Select max(a)
From R
Where a>v;



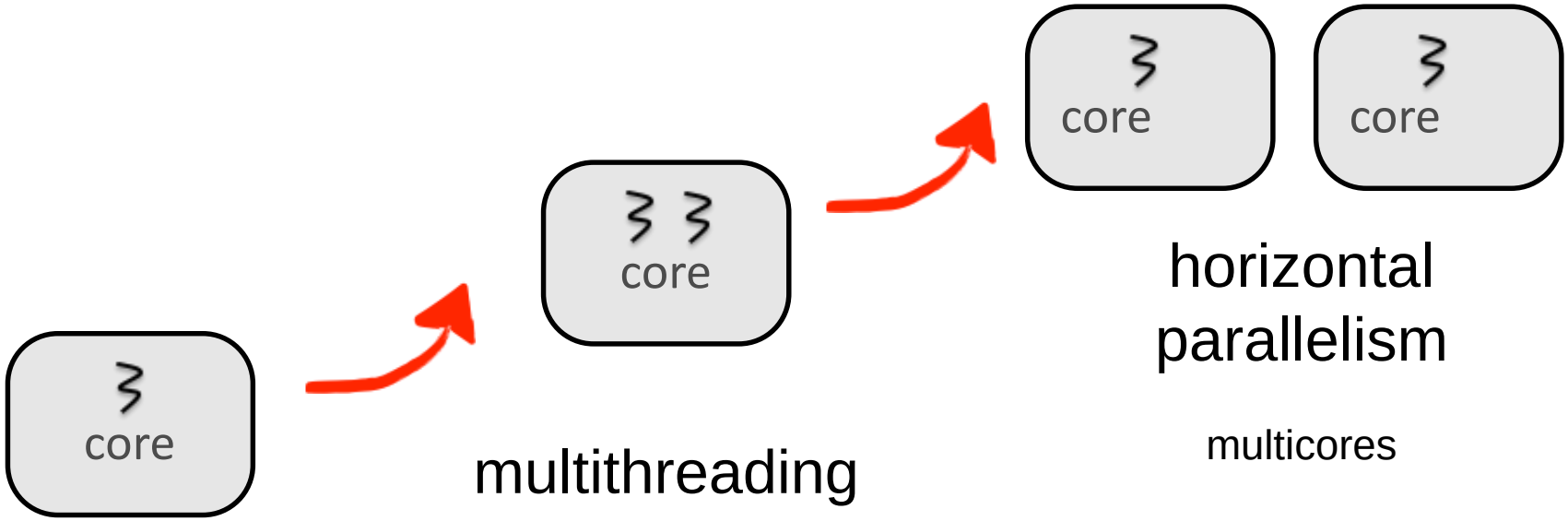
parallel adaptive indexing

[DaMoN14a]



we move less data during the merge phase

modern parallelism



instruction & data parallelism

pipelining
superscalar

SIMD

multithreading

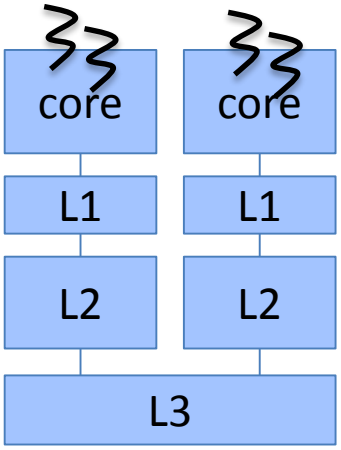
SMT

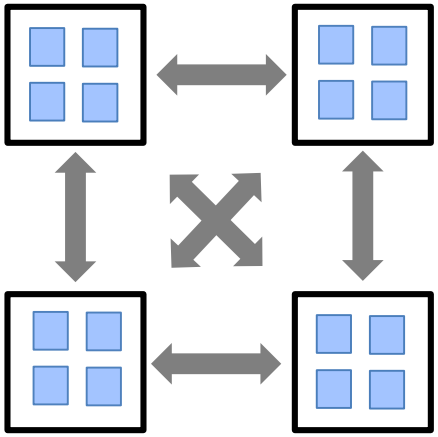
horizontal parallelism

multicores

utilization

scalability





exploiting core's resources

minimizing memory stalls

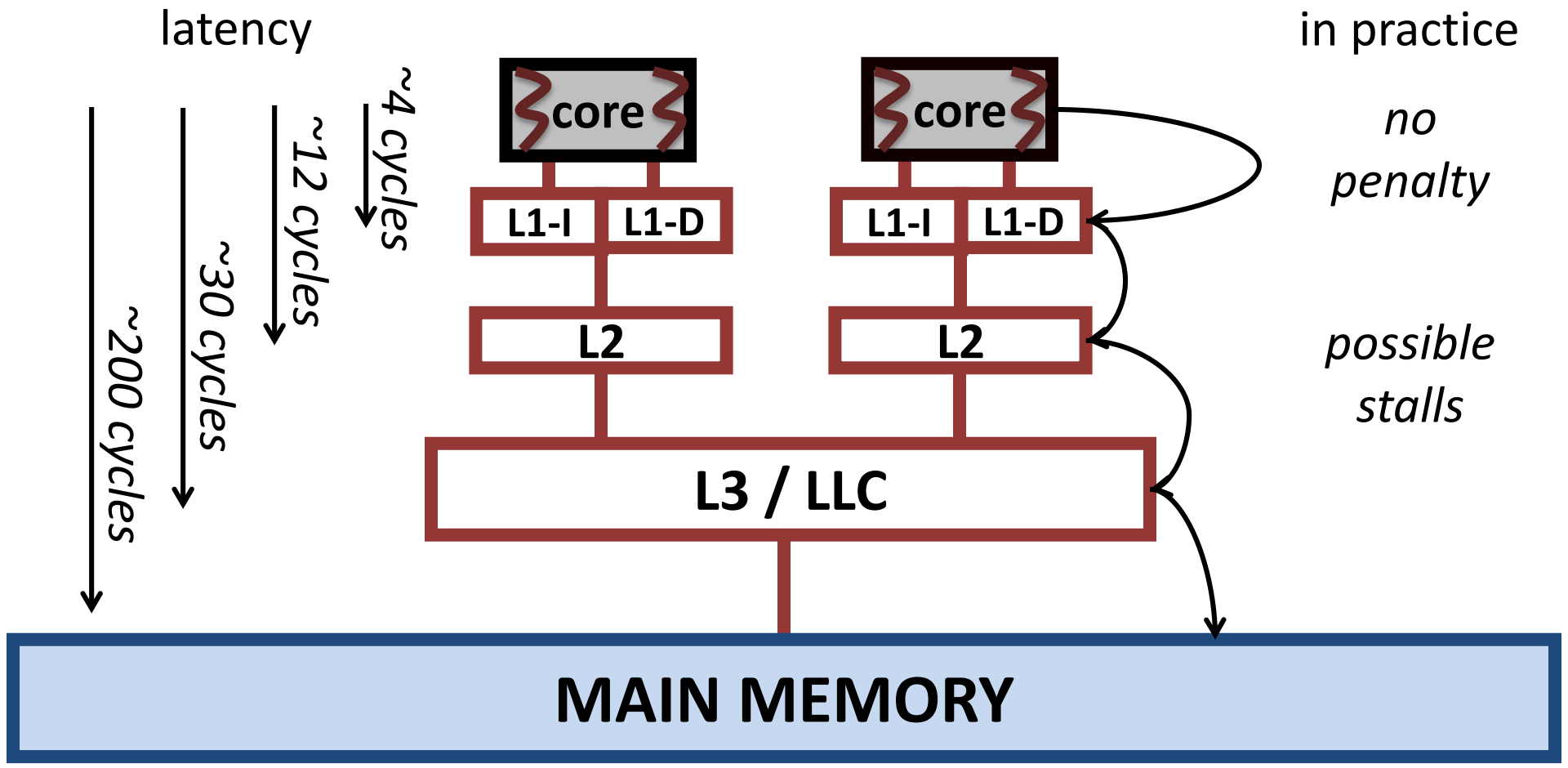
scaling up OLTP

scaling up OLAP

conclusions

http://tinyurl.com/tutorial-2015-feedback

today's memory hierarchy

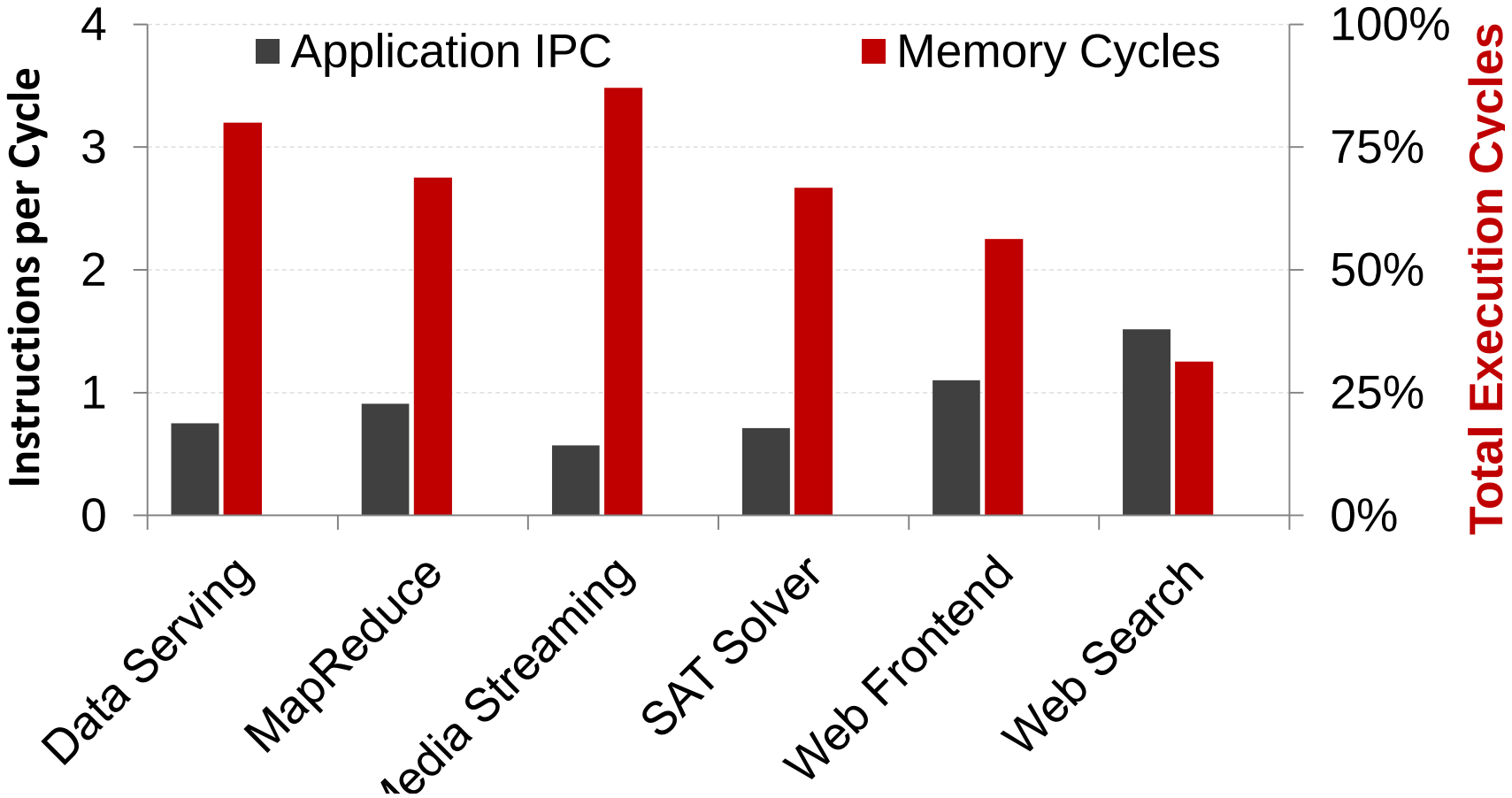


stalls → wasted power & $$$$

stalls in cloud workloads

graph courtesy of Ferdman et al. [ASPLOS12]

CloudSuite on Intel Xeon X5670

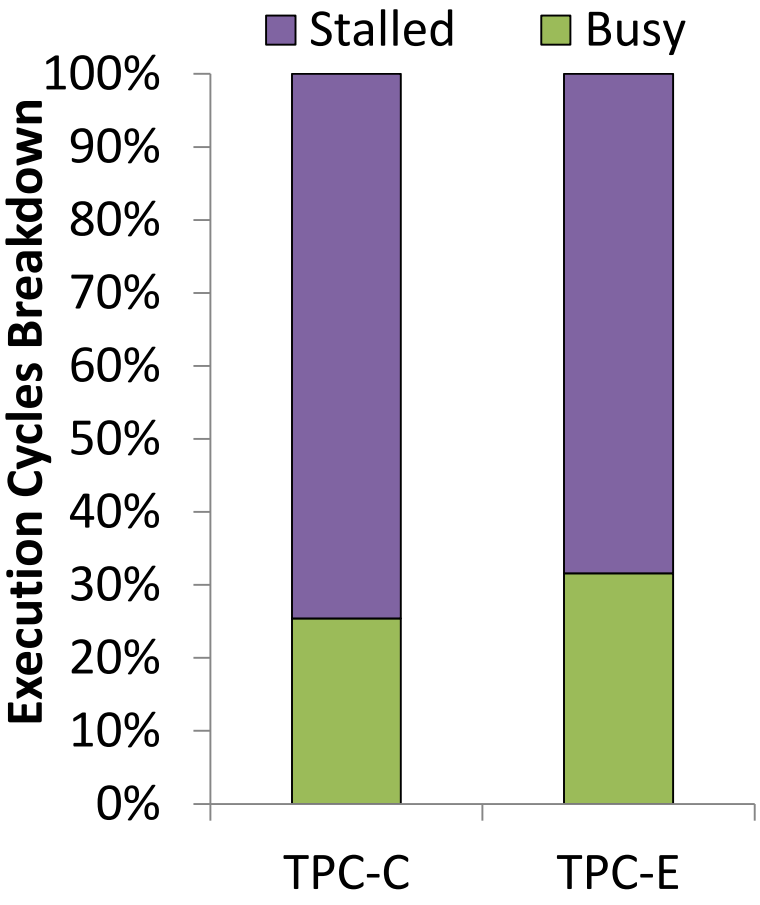


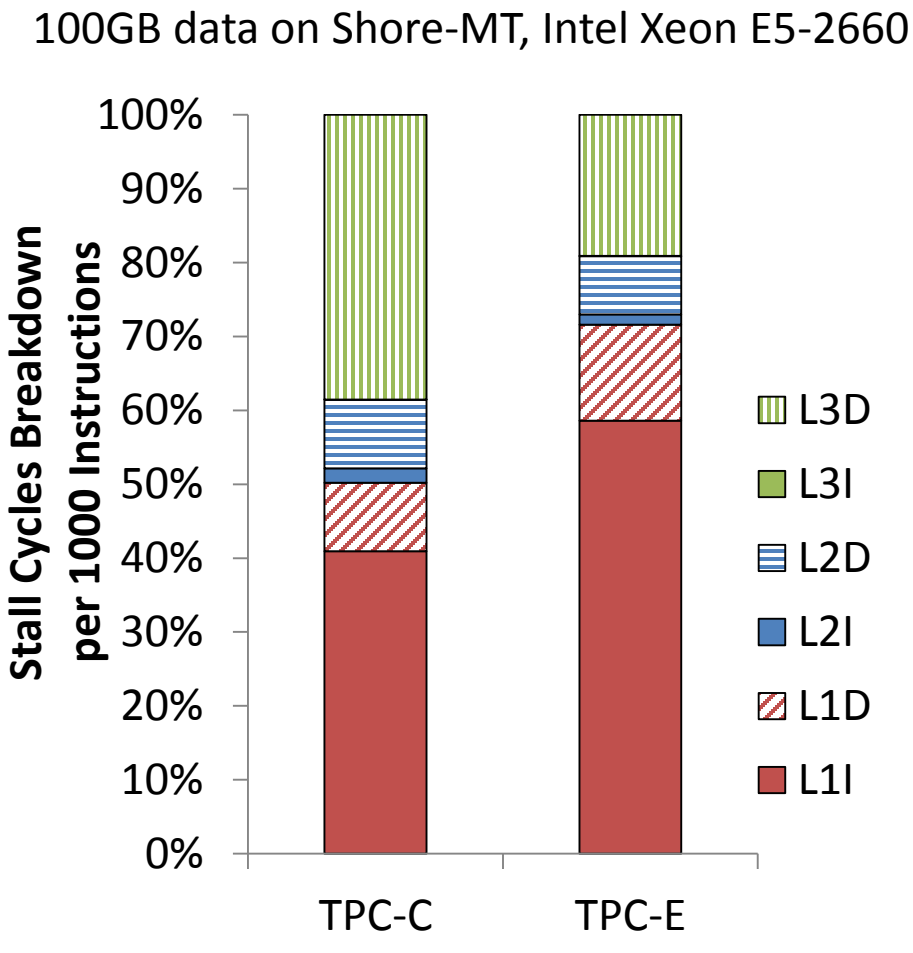
~1 instructions per cycle

> 50% of the time goes to stalls on average

sources of memory stalls

[DaMoN13, EDBT13]





L1-I & LLC data misses dominate the stall time

for data intensive applications …

- 50%-80% of cycles are stalls
  - *Problem:*
    instruction fetch & long-latency data misses
  - *Instructions* need more *capacity*
  - *Data misses* are *compulsory*

- Focus on maximizing:
  - *L1-I locality & cache line utilization for data*

minimizing memory stalls

prefetching
- light
- temporal stream
- software-guided

being cache conscious
- code optimizations
- alternative data structures/layout
- vectorized execution

exploiting common instructions
- computation spreading

prefetching – lite

[ISCA90, MICRO00]

… or text-book prefetching

- next-line: miss A → fetch A+1
- stream: miss A, A+1 → fetch A+2, A+3

✓ favors sequential access & spatial locality

✗ instructions: branches, function calls

- branch prediction

✗ data: pointer chasing

- stride: miss A, A+20 → fetch A+40, A+60

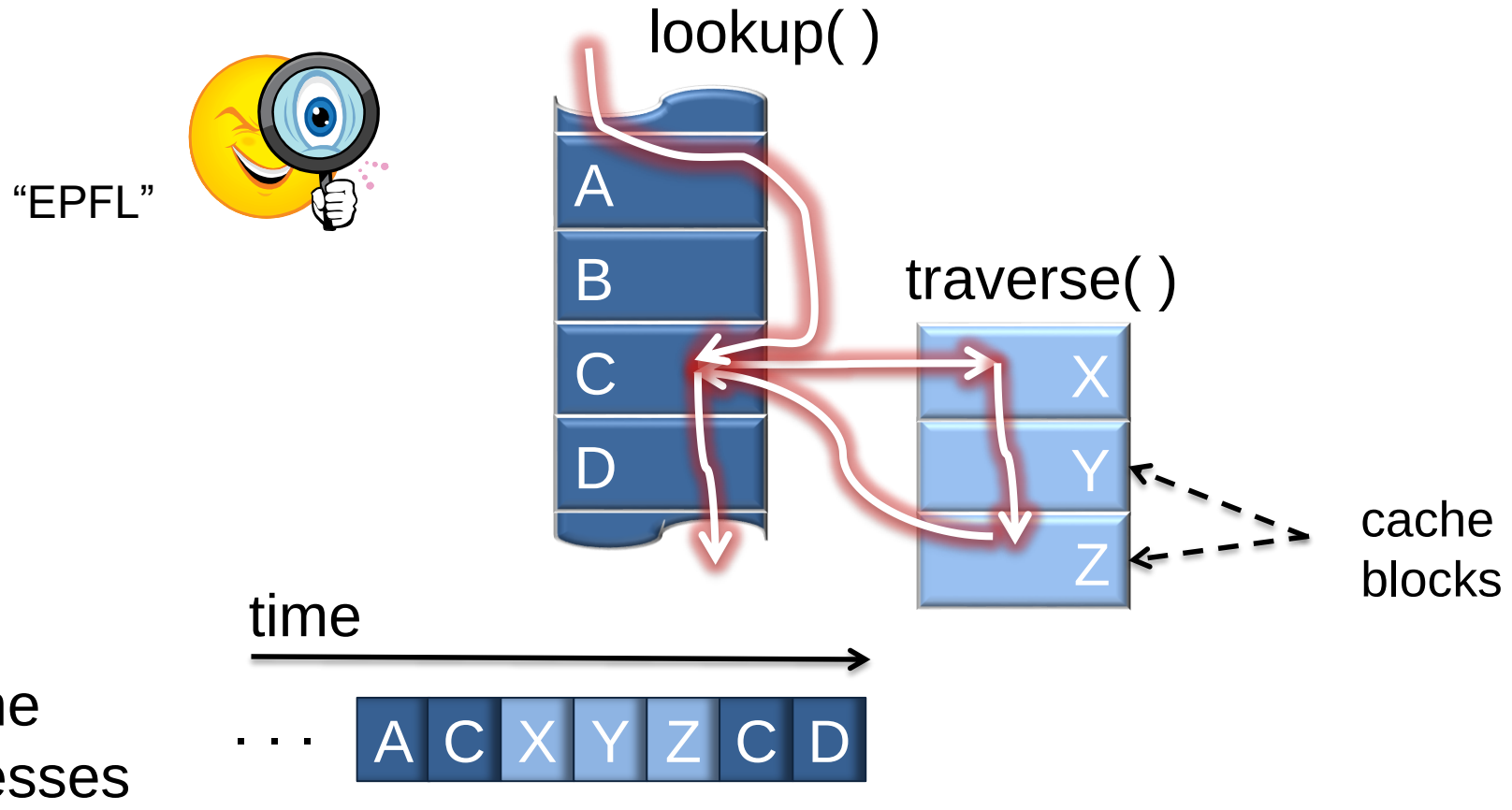
preferred on real hardware due to simplicity

though, memory stalls are still too high

temporal streaming

slide courtesy of Cansu Kaynak

[ISCA05, MICRO11a, MICRO13a]

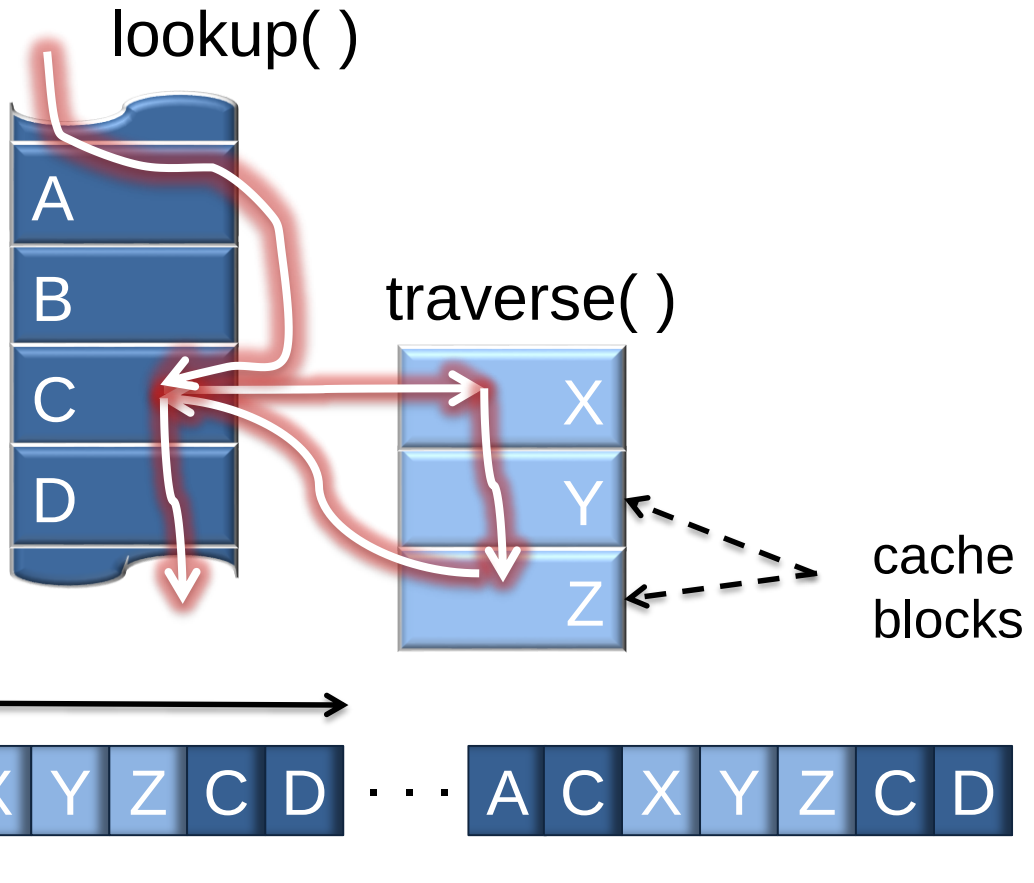


slide courtesy of Cansu Kaynak

temporal streaming

[ISCA05, MICRO11a, MICRO13a]

"ICDE"

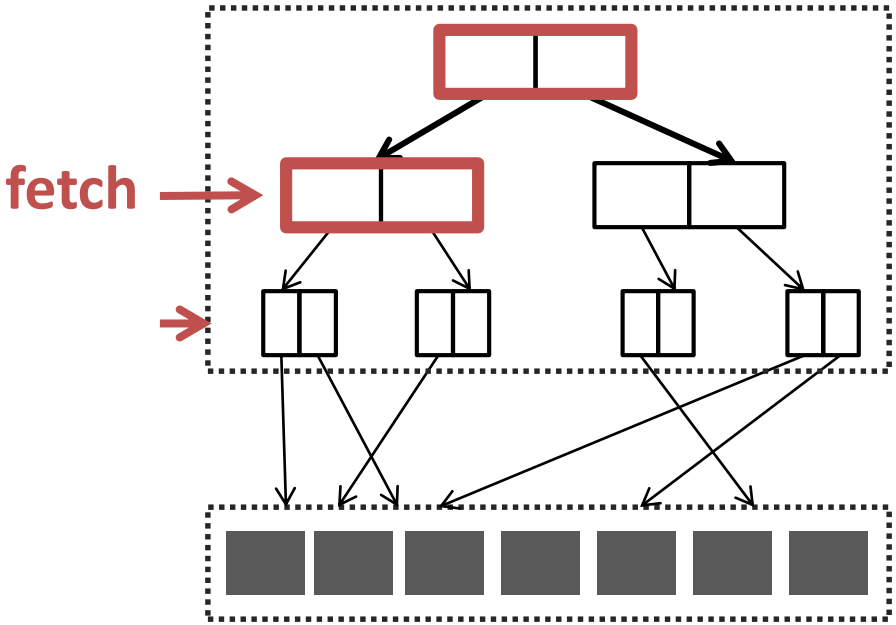


exploits recurring control flow

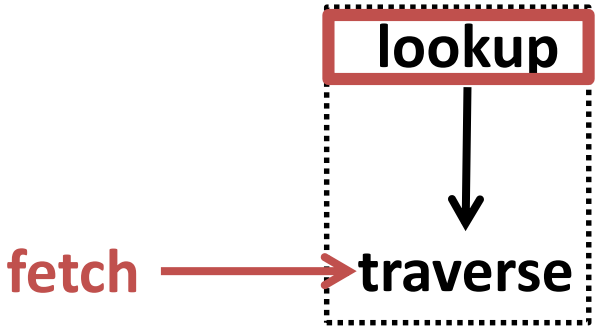
high space cost

software-guided prefetching [Eurosys12, TOCS03]

data



instructions



only for data on real hardware

minimizing memory stalls

prefetching
- light
- temporal stream
- software-guided

being cache conscious
- code optimizations
- alternative data structures/layout
- vectorized execution

exploiting common instructions
- computation spreading

code optimizations

[ISCA01, ICDE10, PVLDB11a, SIGMOD13a]

- simplified code
  - in-memory databases have smaller instruction footprint
- better code layout
  - minimize jumps → exploit next line prefetcher
  - profile-guided optimizations (static)
  - just-in-time (dynamic)
- query compilation into machine/naïve code
  - e.g., HyPer, Hekaton, MemSQL

cache conscious data layouts

[SIGMOD85, CIDR05, VLDB05a]

erietta	blue
pinar	black
danica	green
iraklis	orange

16 bytes columns

goal:
maximize cache line utilization & exploit next-line prefetcher

row stores: good for OLTP
accessing many columns

column stores: good for OLAP
accessing a few columns

cache lines (64bytes)

erietta	blue	pinar	black

row store

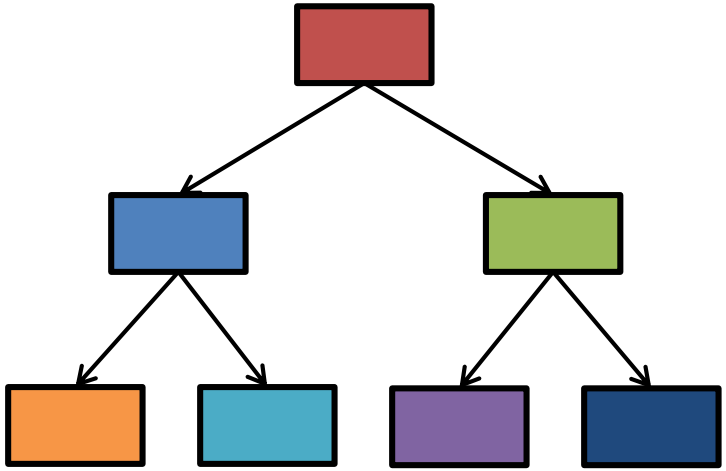
erietta	pinar	danica	iraklis

column store

cache conscious data structures

[SIGMOD02a, VLDB06]

index tree



in memory

lookup-heavy workload

scan-heavy workload

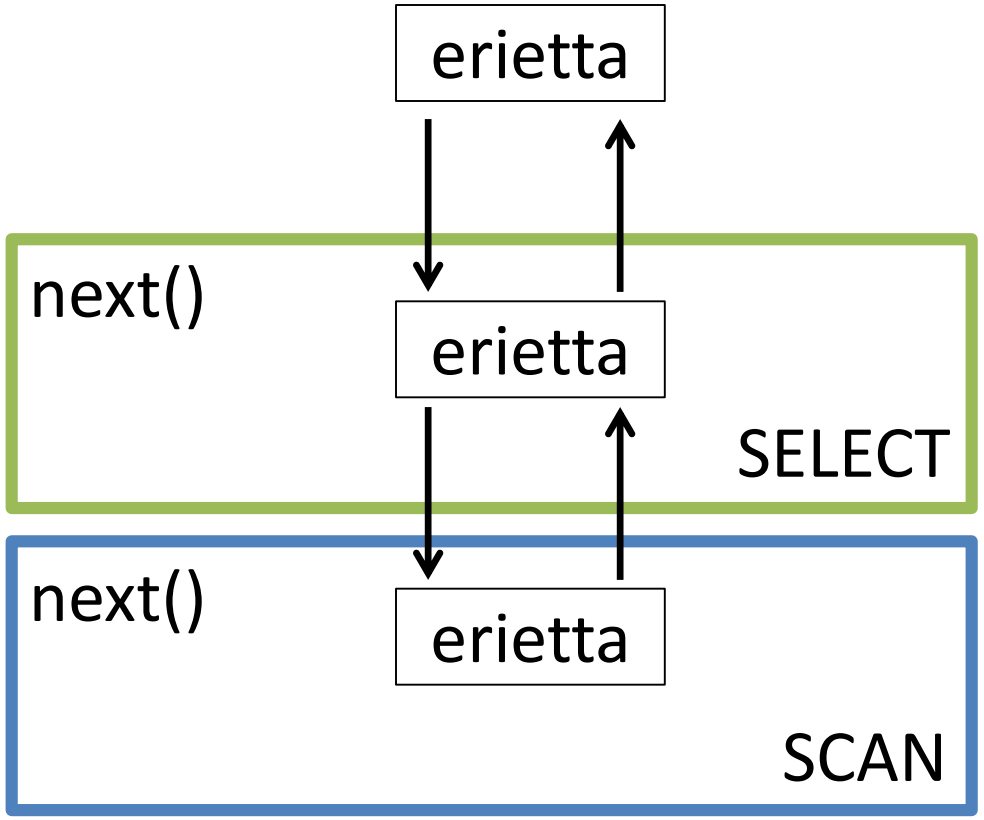
+ align nodes to cache lines

goal: maximize cache line utilization & exploit next-line prefetcher in tree probe

volcano iterator model

[CIDR05]

erietta	blue
pinar	black
danica	green
iraklis	orange
. . .	. . .

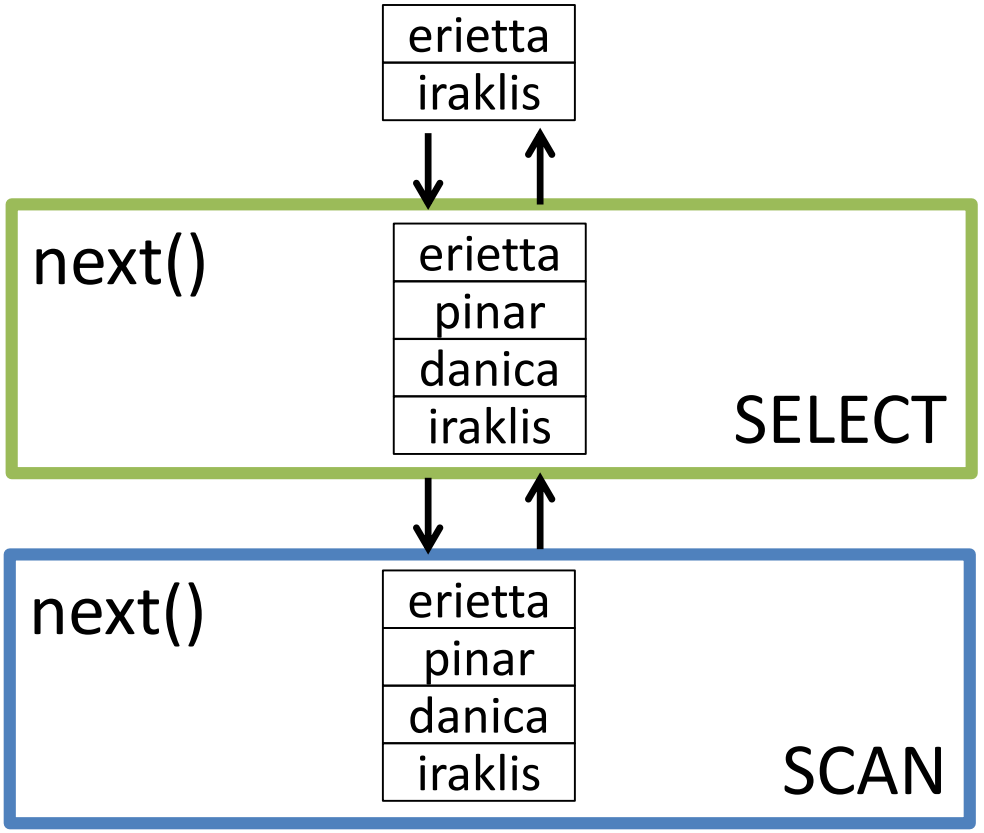


✗ **poor data & instruction cache locality**

vectorized execution

[CIDR05]

erietta	blue
pinar	black
danica	green
iraklis	orange
. . .	. . .



✓ **good data & instruction cache locality**

✓ **allows exploiting SIMD**

minimizing memory stalls

prefetching
- light
- temporal stream
- software-guided

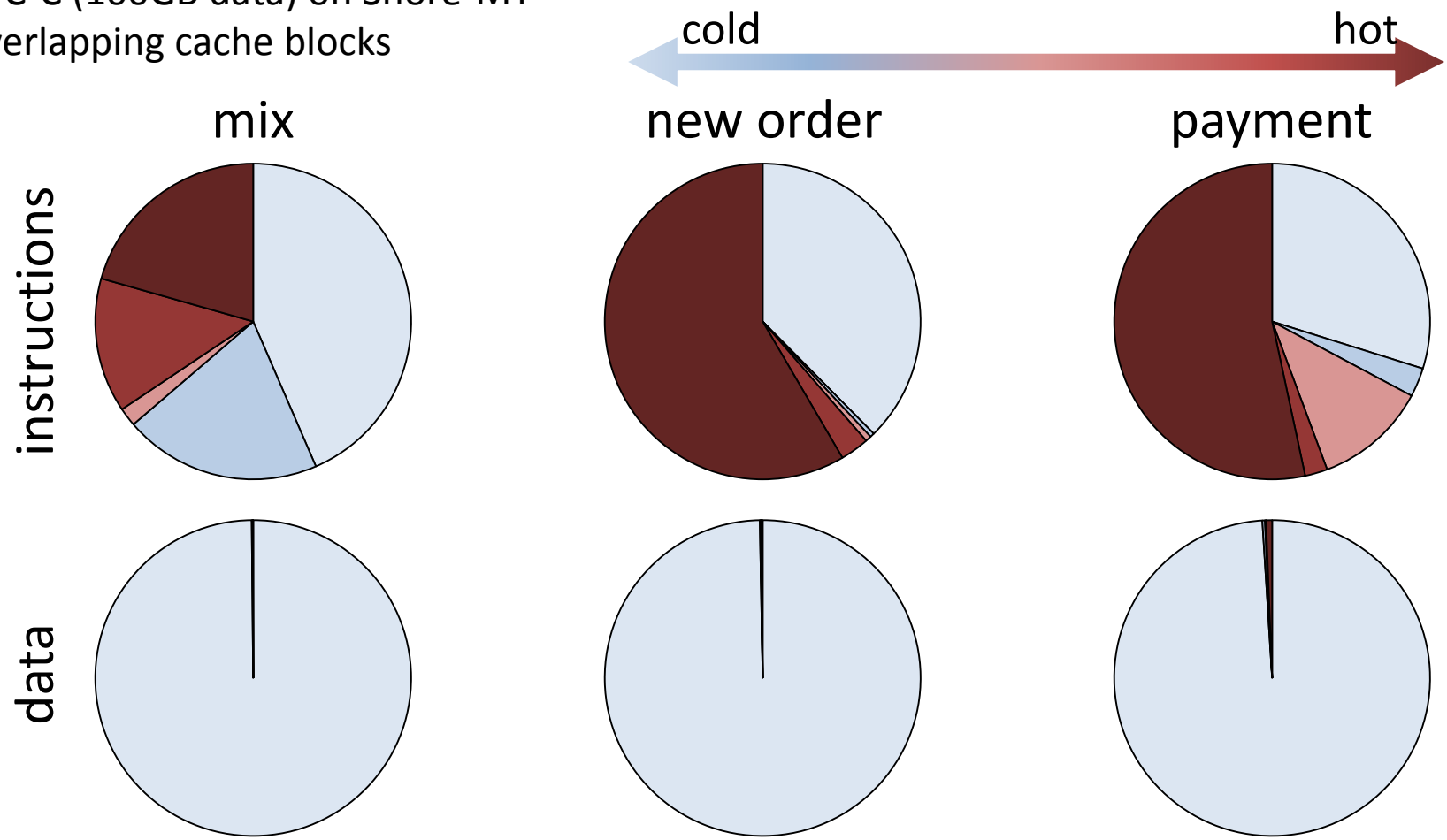
being cache conscious
- code optimizations
- alternative data structures/layout
- vectorized execution

exploiting common instructions
- computation spreading

instruction & data overlap

[PVLDB14f]

TPC-C (100GB data) on Shore-MT
overlapping cache blocks

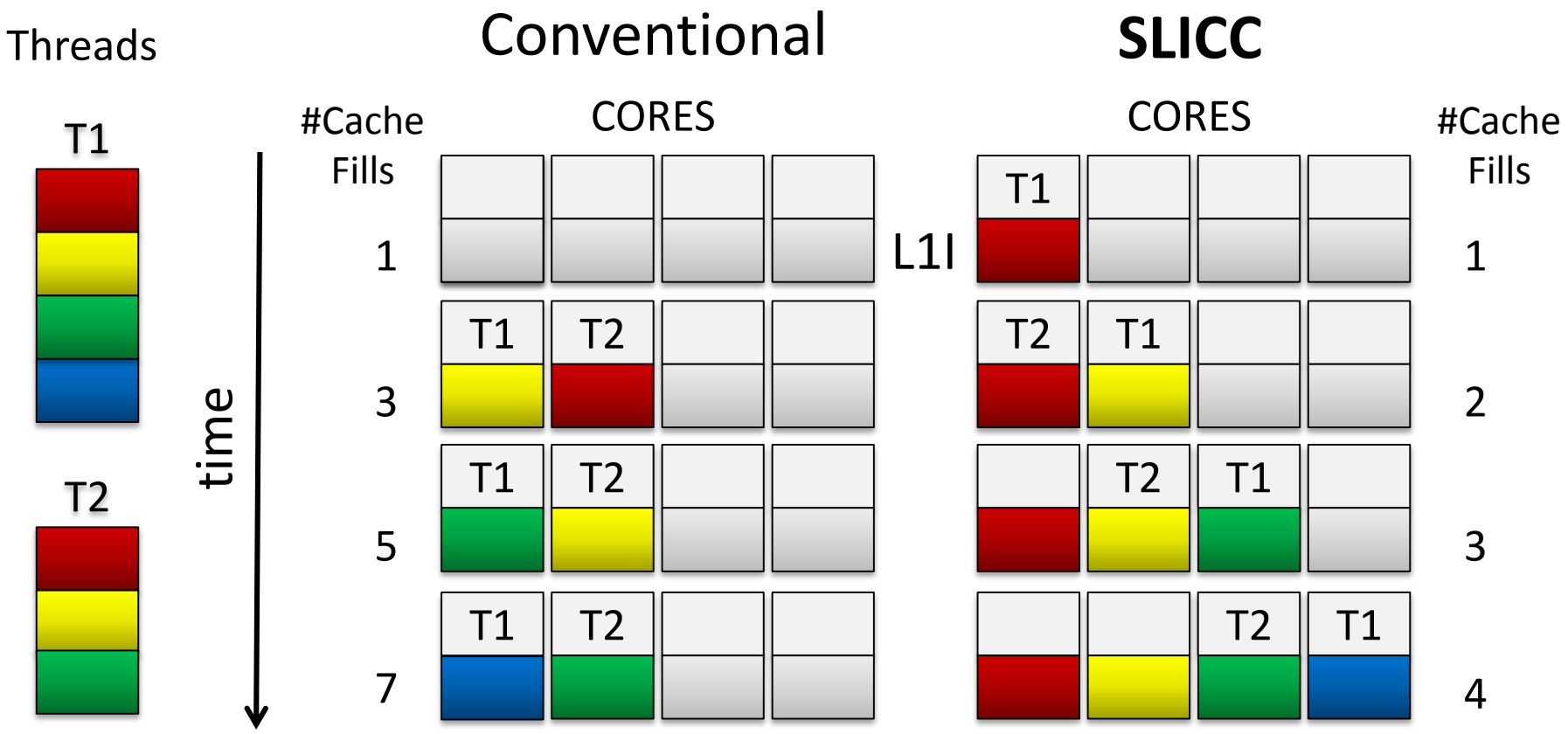


overlap: significant for instructions & low for data

higher overlap in same-type transactions

computation spreading

[ASPLOS06, MICRO12]



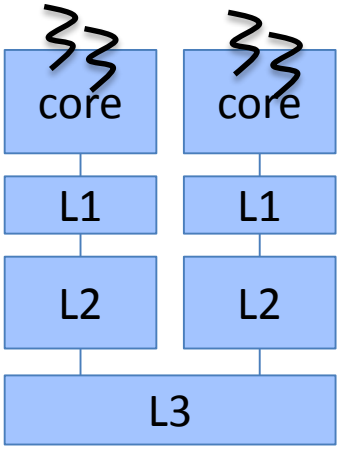
exploits aggregate L1-I & instruction overlap

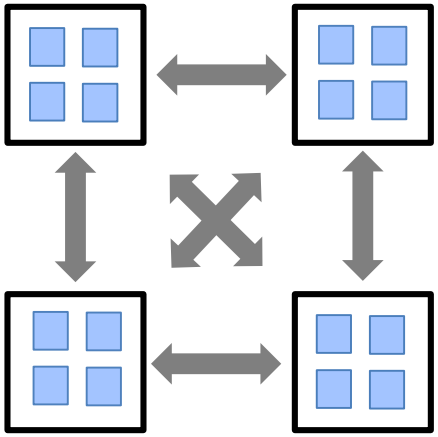
need to track recent misses and cache contents

summary

- DBMSs underutilize a core's resources
- Problem 1: L1-I misses
  - due to capacity
  - minimized footprint &
    illusion of a larger cache by maximizing re-use
- Problem 2: LLC data misses
  - compulsory
  - maximize cache-line utilization through
    cache-conscious algorithms and layout

utilization **scalability**





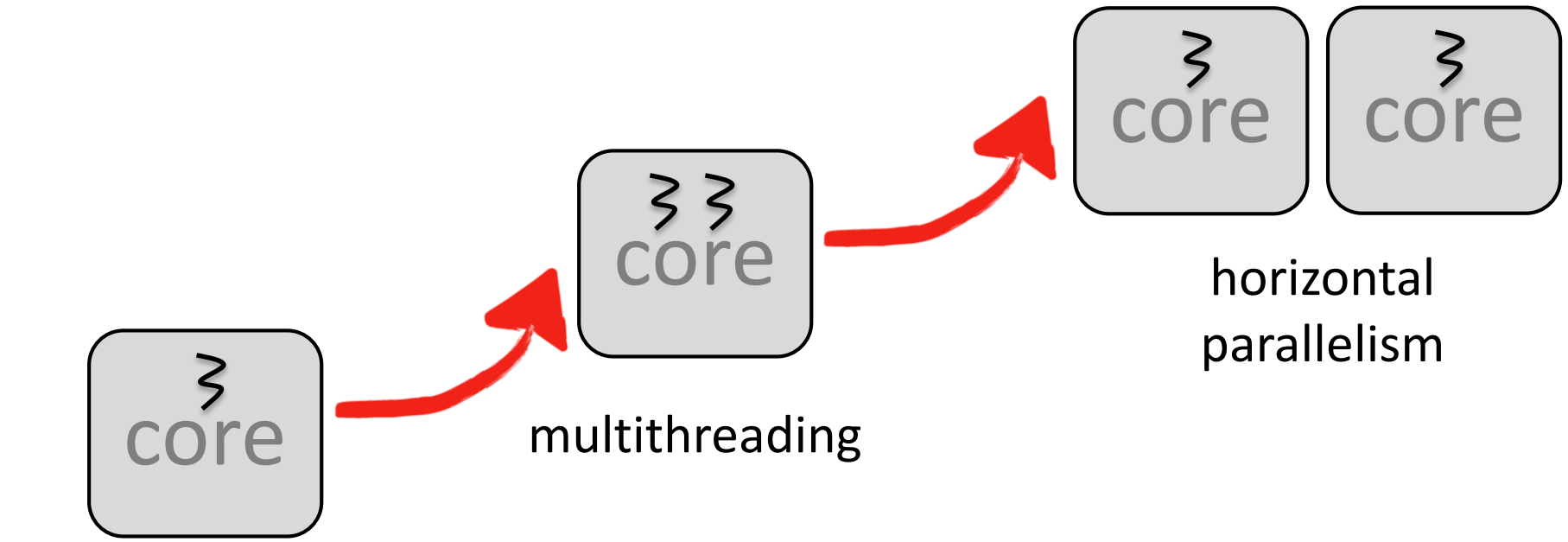
exploiting core's resources
minimizing memory stalls

scaling up OLTP
scaling up OLAP
conclusions

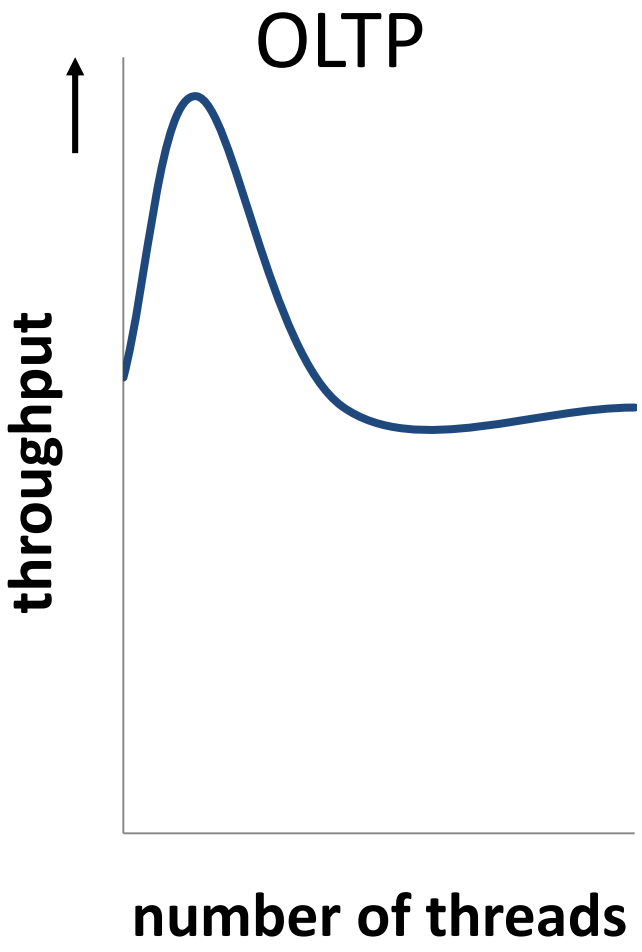
http://tinyurl.com/LoveMulticores

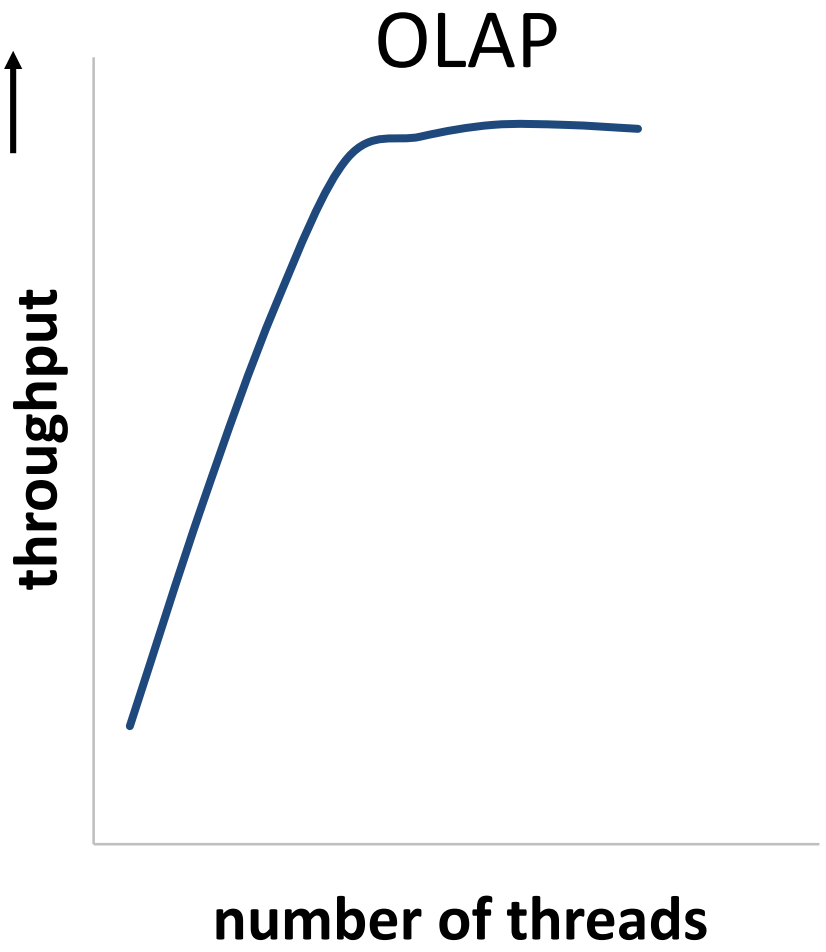
http://tinyurl.com/tutorial-2015-feedback

modern parallelism



challenges when scaling up

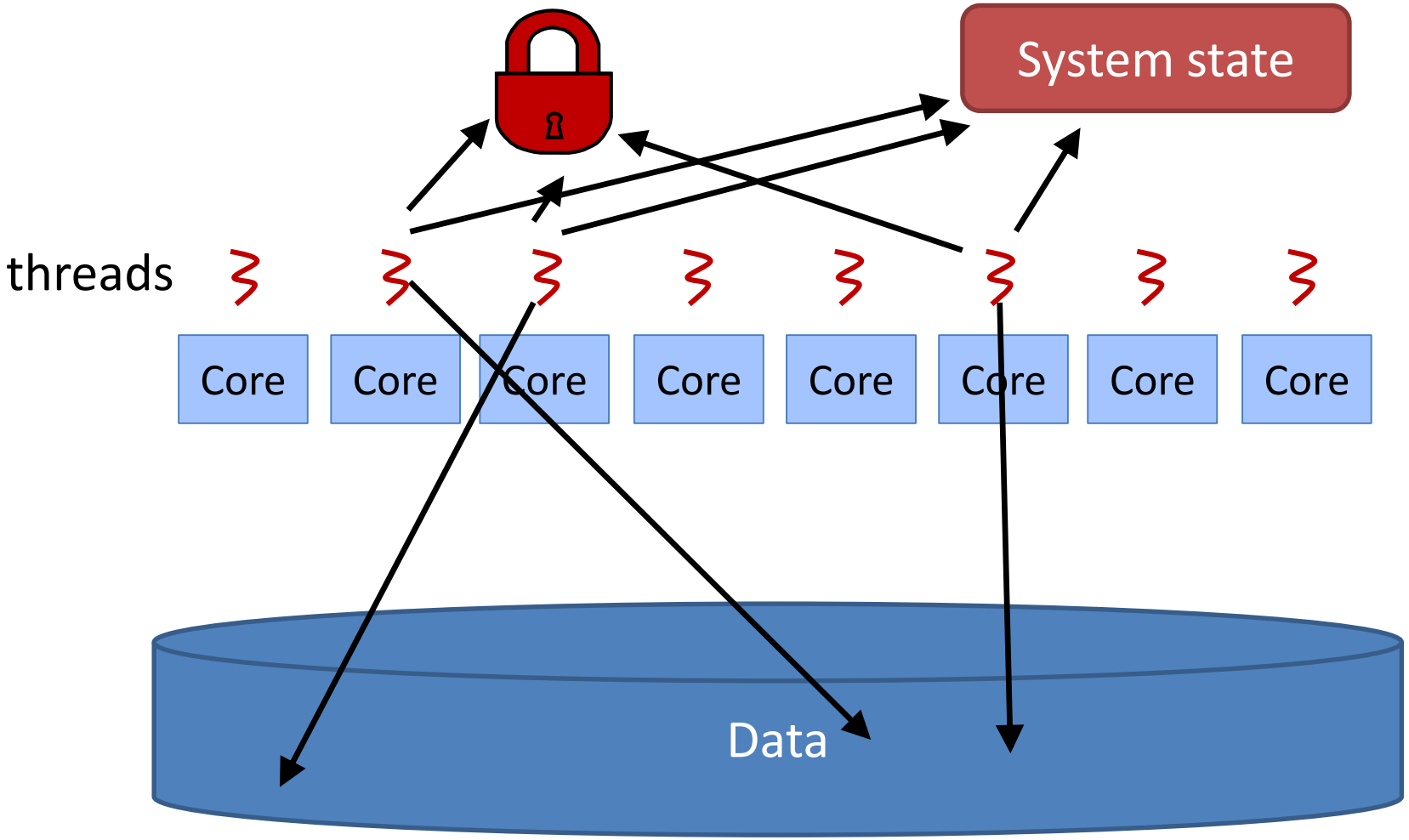




access latency

memory bandwidth

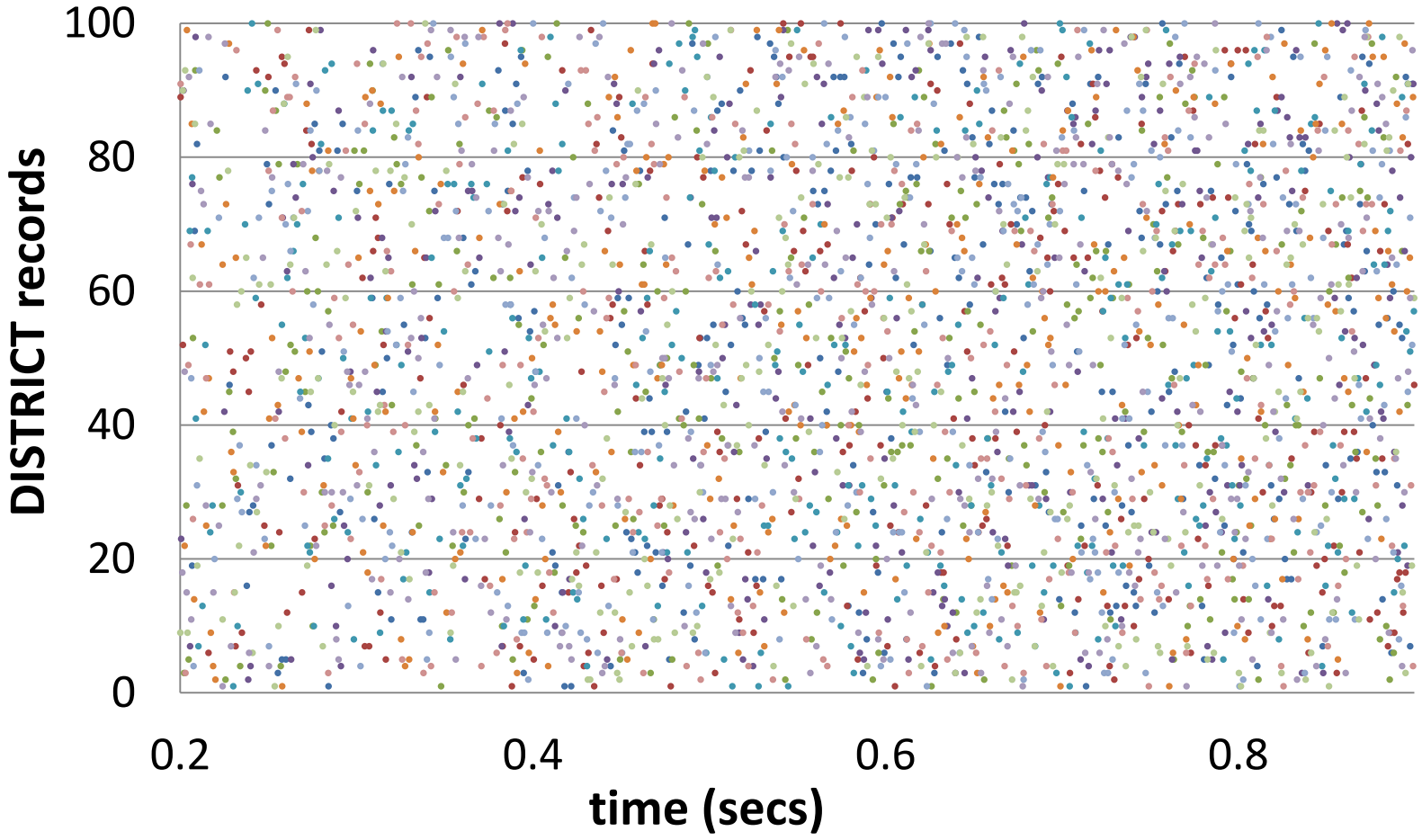
critical path of transaction execution



many accesses to shared data structures

data access pattern

[PVLDB10b]

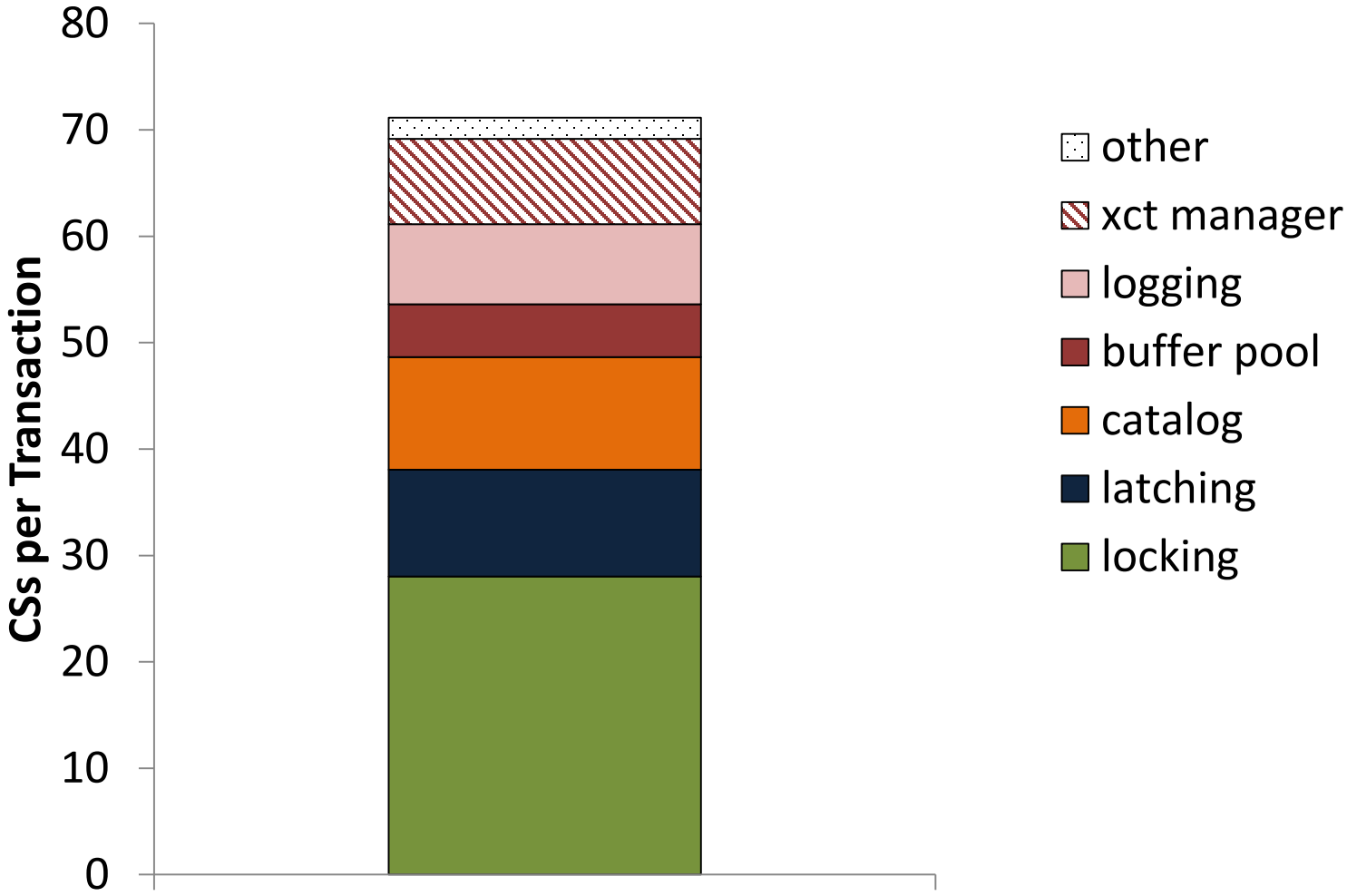


unpredictable data accesses

clutter code with critical sections -> contention

critical sections

Updating 1 row

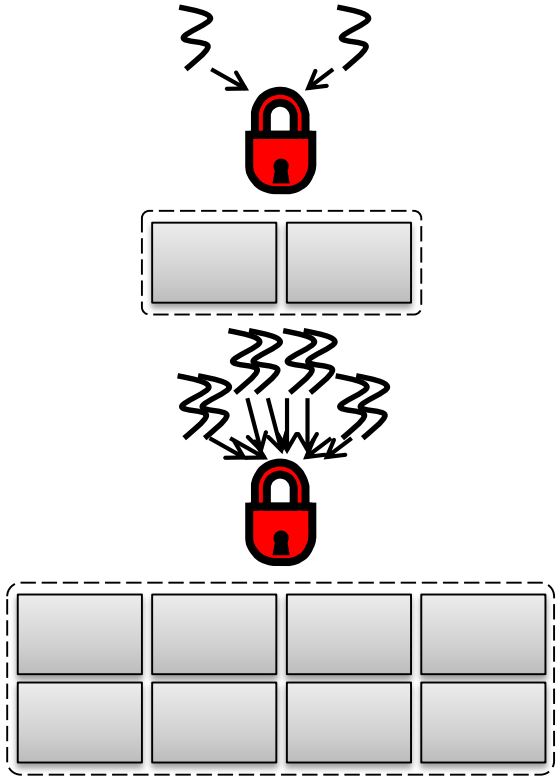


many critical sections even for simplest transaction

critical section types

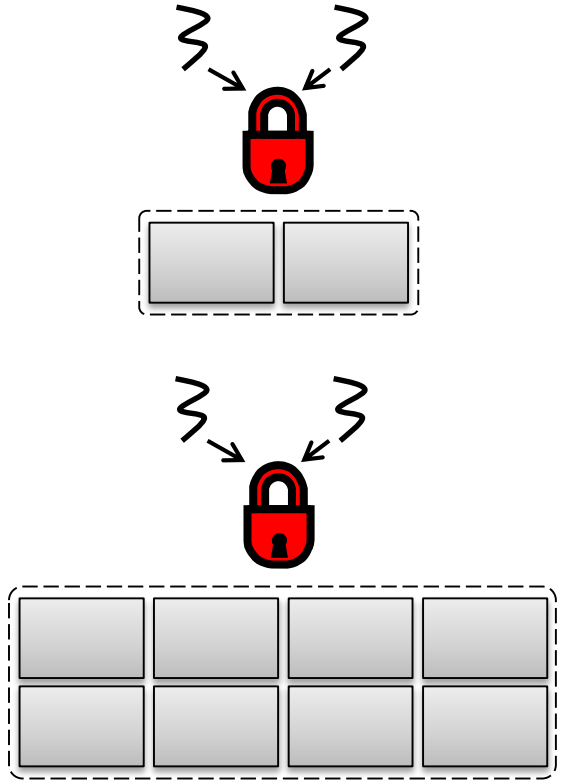
[VLDBJ14]

unbounded



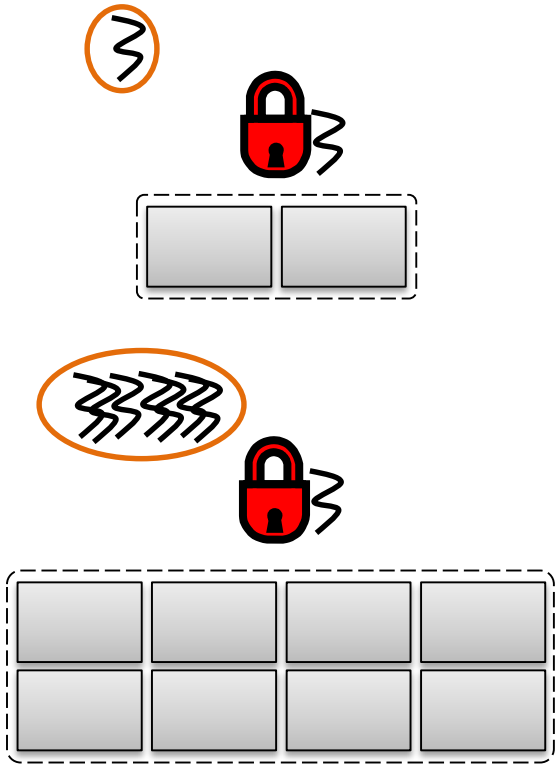
locking, latching

fixed



transaction manager

cooperative



logging

unbounded → fixed / cooperative

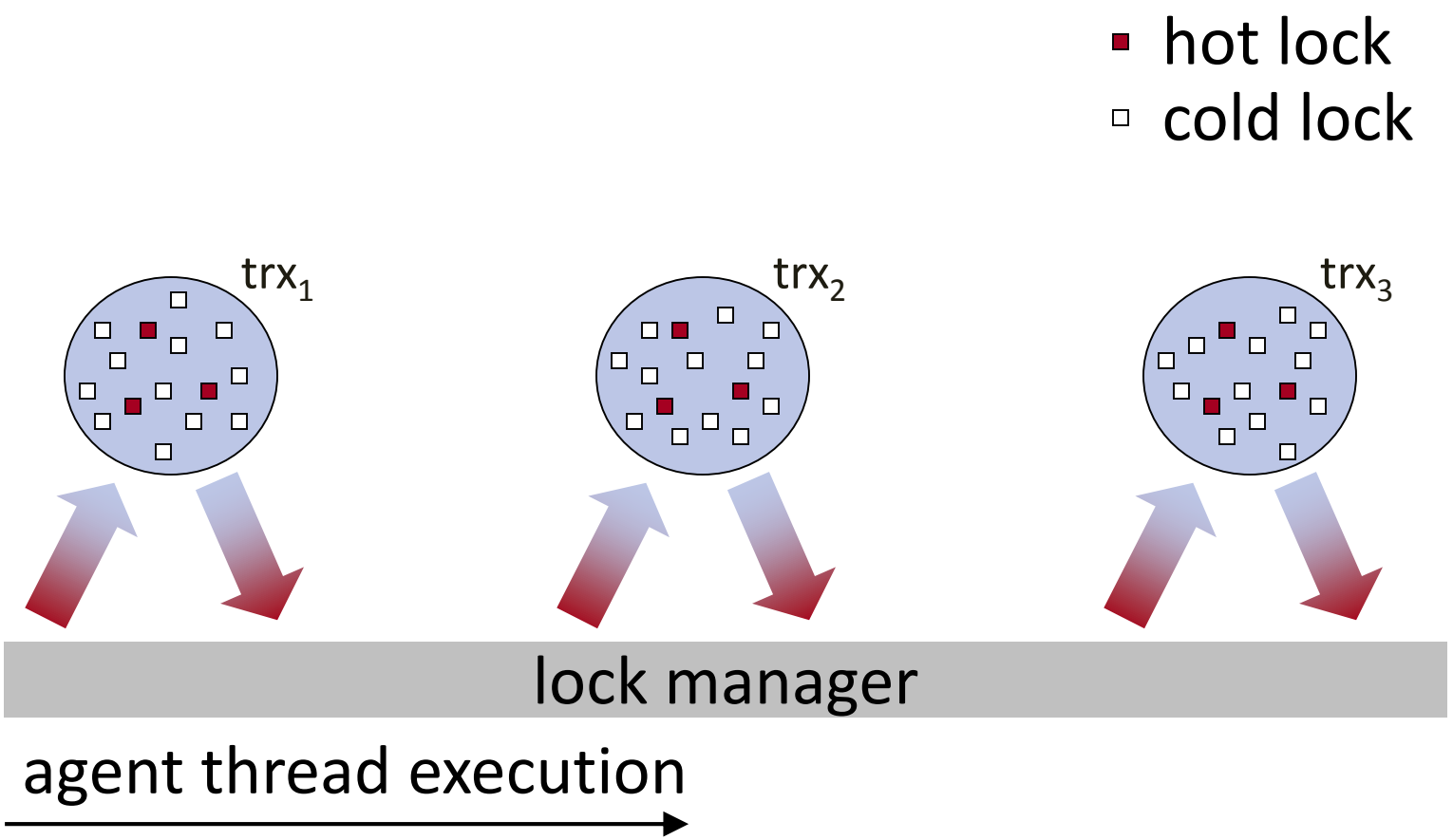
scaling up OLTP

unscalable components
- locking
- latching
- logging

synchronization
- tradeoffs
- best practices

non-uniform communication
- hardware Islands

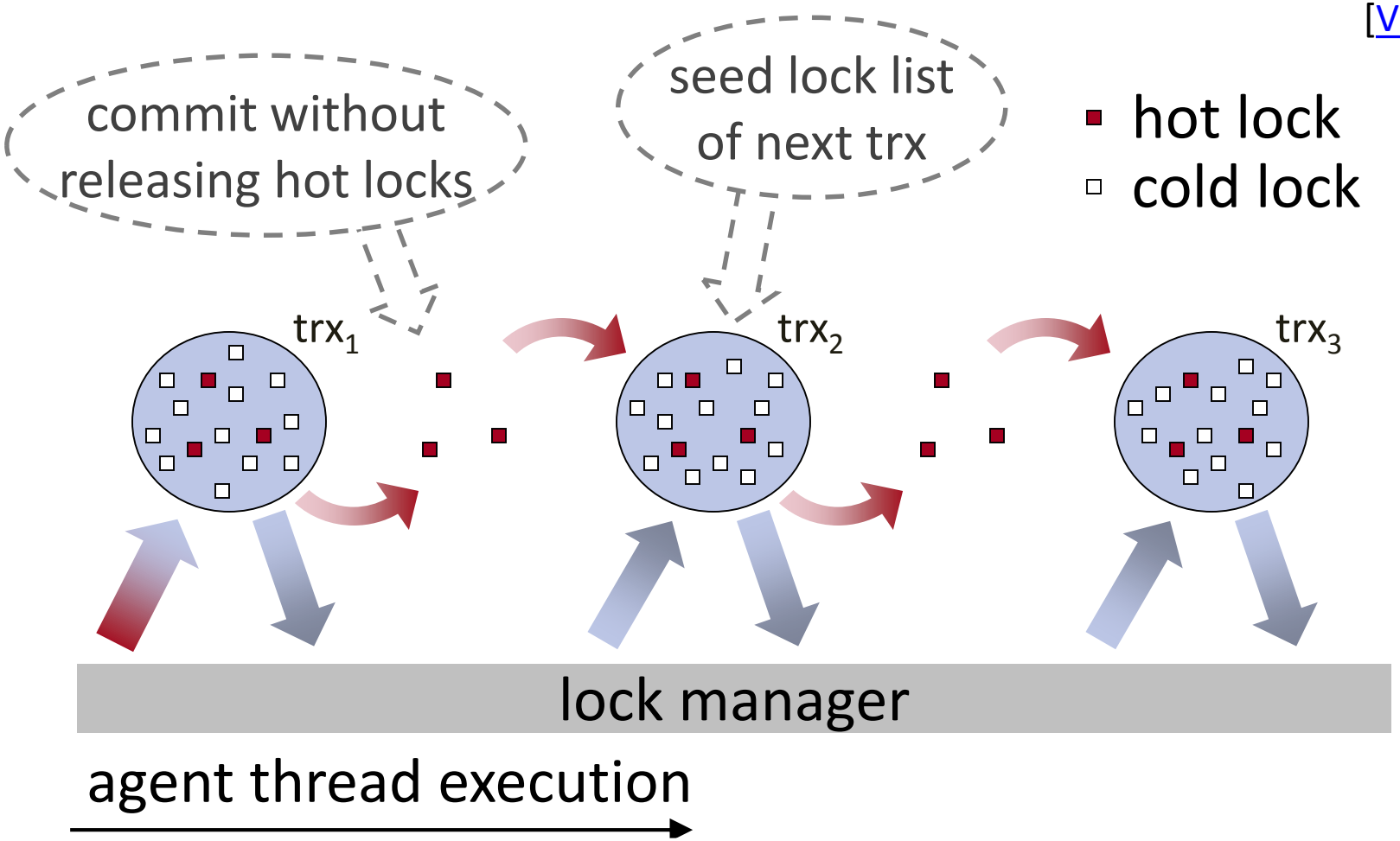
hot shared locks cause contention



release and request the same locks repeatedly

speculative lock inheritance

[VLDB09b]



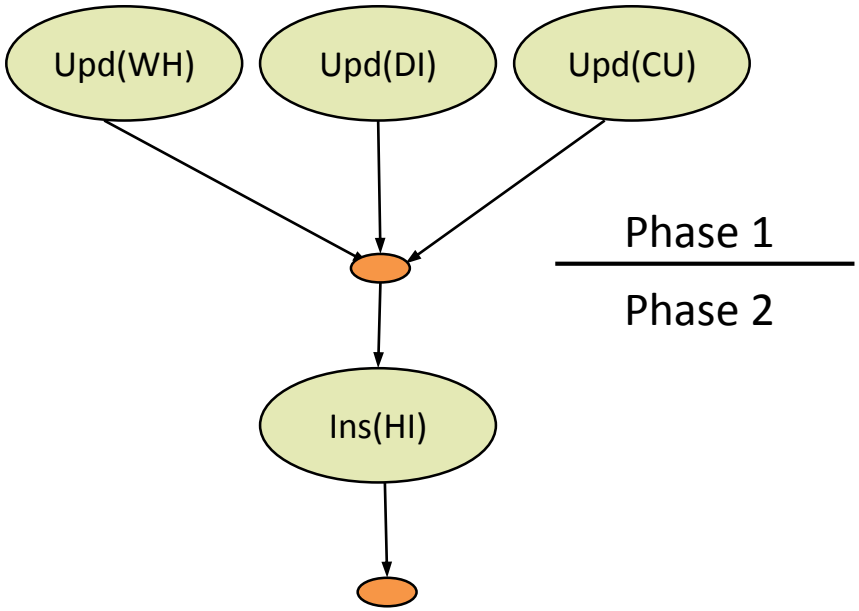
significantly reduces lock contention

LIL: co-locate atomic counters with data [ADMS12]

data-oriented transaction execution

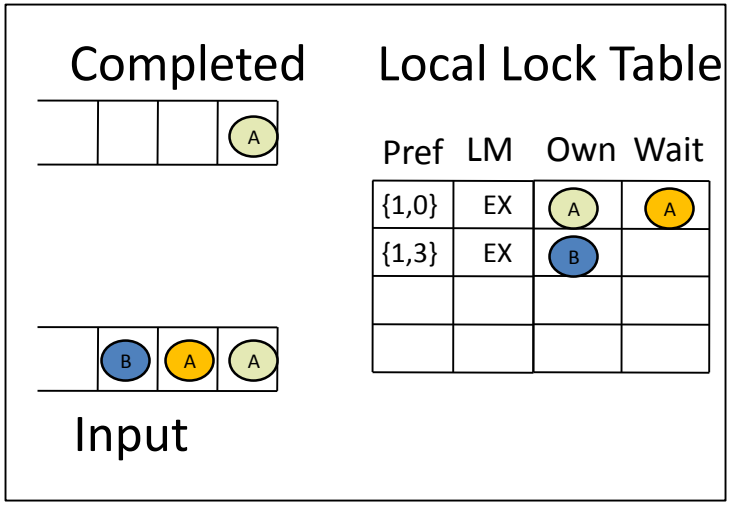
[PVLDB10b]

TPC-C Payment



Routing fields: {WH_ID, D_ID}

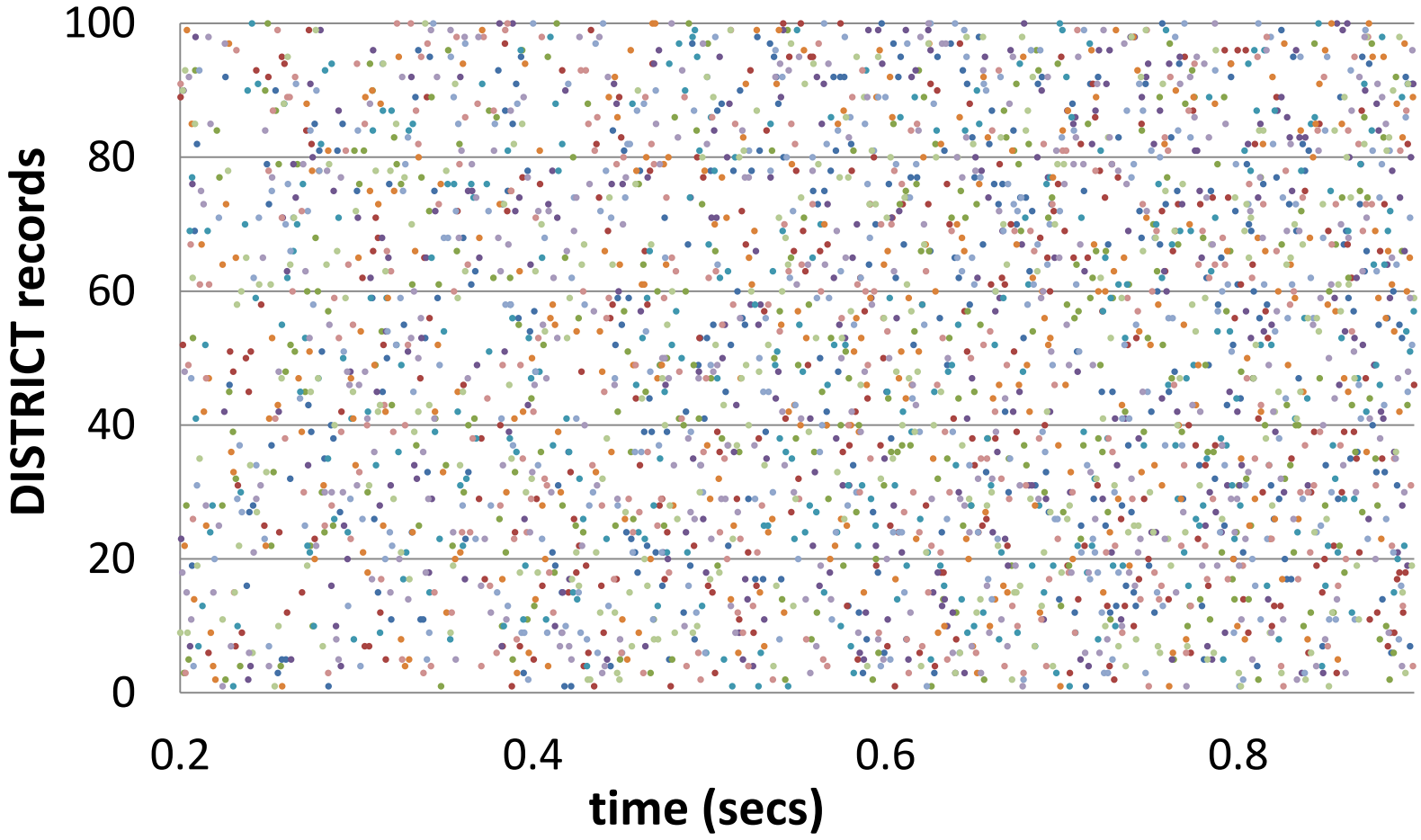
Range	Executor
A-H	1
I-N	2



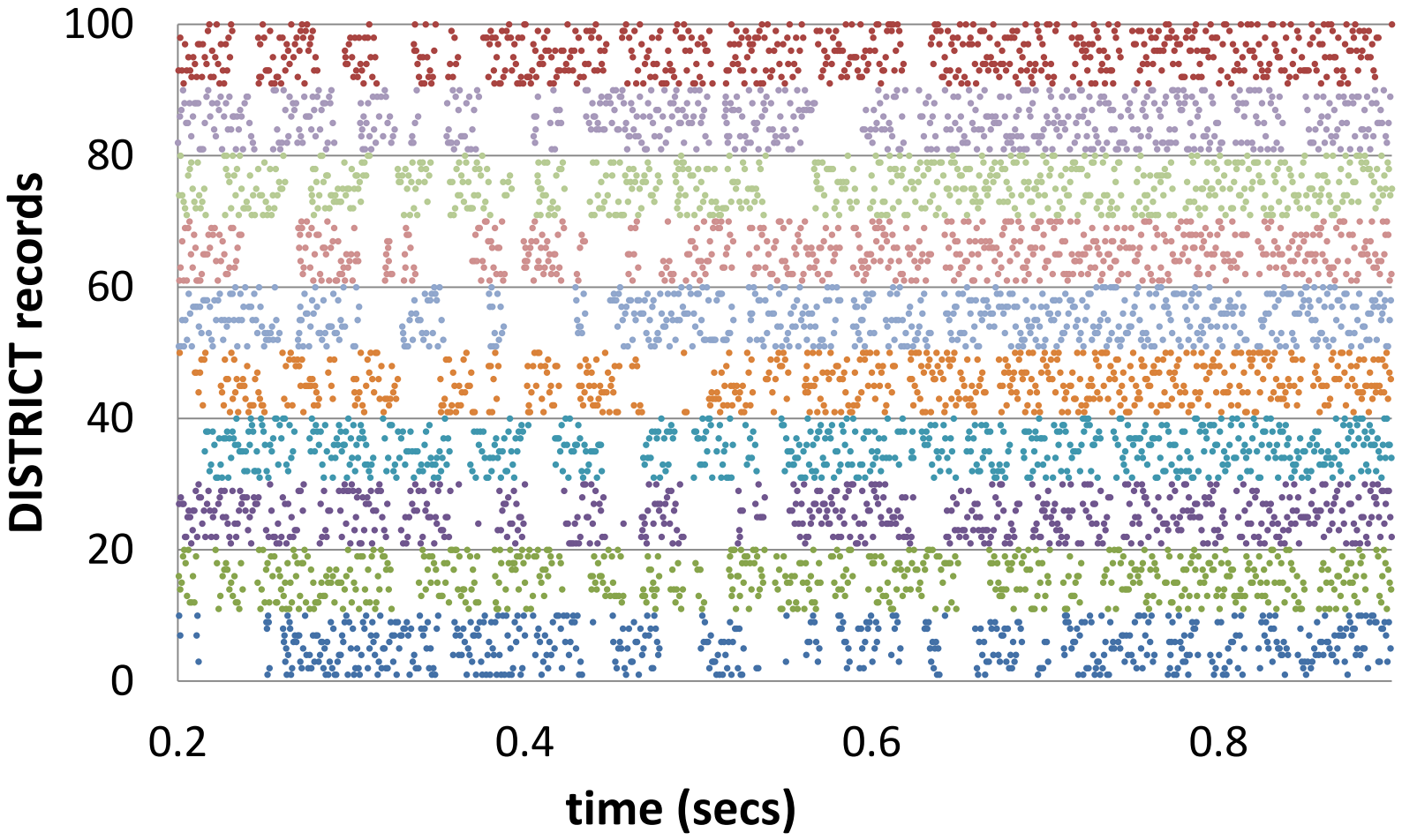
convert centralized locking to thread-local

thread-to-transaction - access pattern

[PVLDB10b]

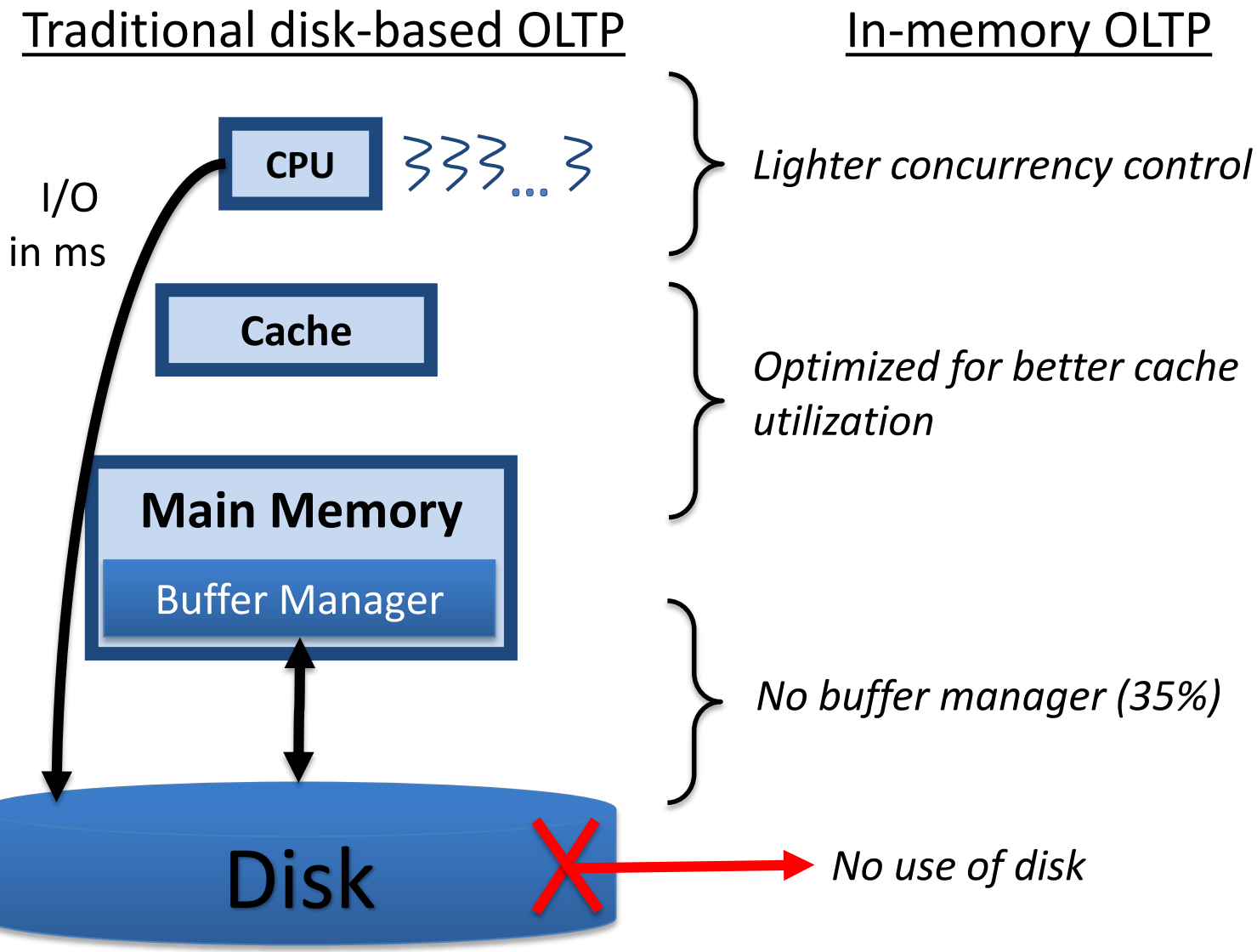


thread-to-data – access pattern [PVLDB10b]



predictable data accesses

in-memory databases



modern shared-nothing systems

- H-Store/VoltDB [VLDB07b]
  - speculative optimistic concurrency control [SIGMOD10b]
  - durability through replication or logical logging [ICDE14c]
- HyPer [ICDE11]
  - OLAP support through VM snapshots, code compilation
  - strict timestamp ordering [IMDM13]
  - tentative execution for long running transactions [CIDR13c]
- Calvin [SIGMOD12]
  - deterministic execution model with conflict detection
  - very lightweight locking [PVLDB13c]

concurrency control for multicores

- scalable serializable snapshot isolation [ICDE14a]
  - latch-free validation phase using atomic ops
- distributed snapshot isolation in SAP HANA [ICDE13b]
  - snapshot tokens, local-only transactions and write buffering

- Hekaton [SIGMOD13a, PVLDB12c]
  - OCC with parallel validation and commit dependency tracking
- Silo [SOSP13]
  - OCC with decentralized validation scheme

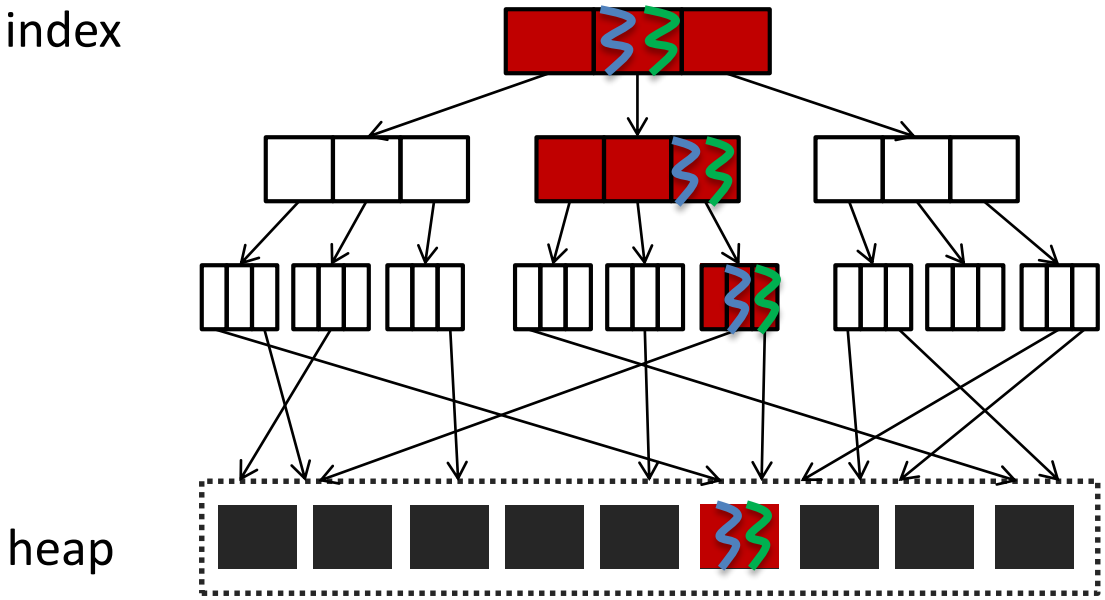
scaling up OLTP

unscalable components
- locking
- latching
- logging

synchronization
- tradeoffs
- best practices

non-uniform communication
- hardware Islands

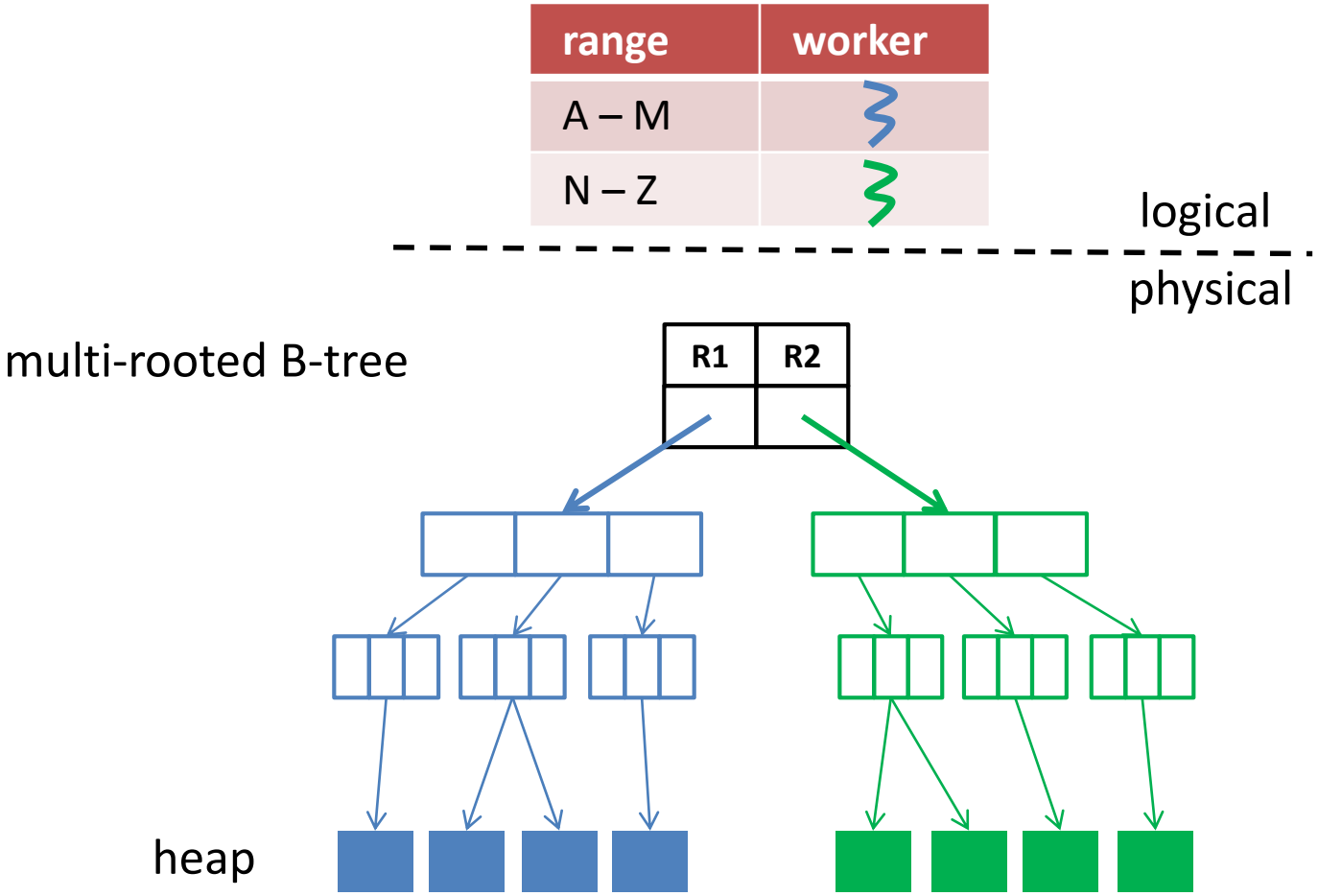
data access in centralized B-tree



conflicts on both index and heap pages

physiological partitioning (PLP)

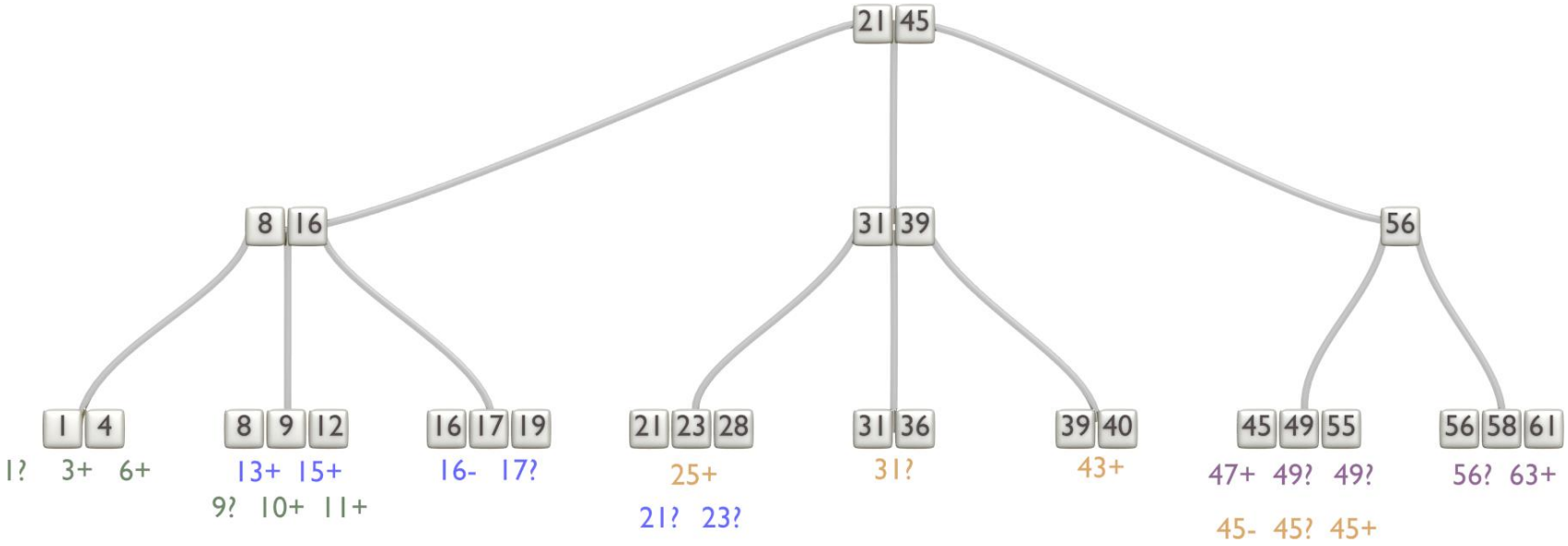
[PVLDB11b]



PALM: latch-free B-tree

figure courtesy of Jason Sewall

[PVLDB11c]

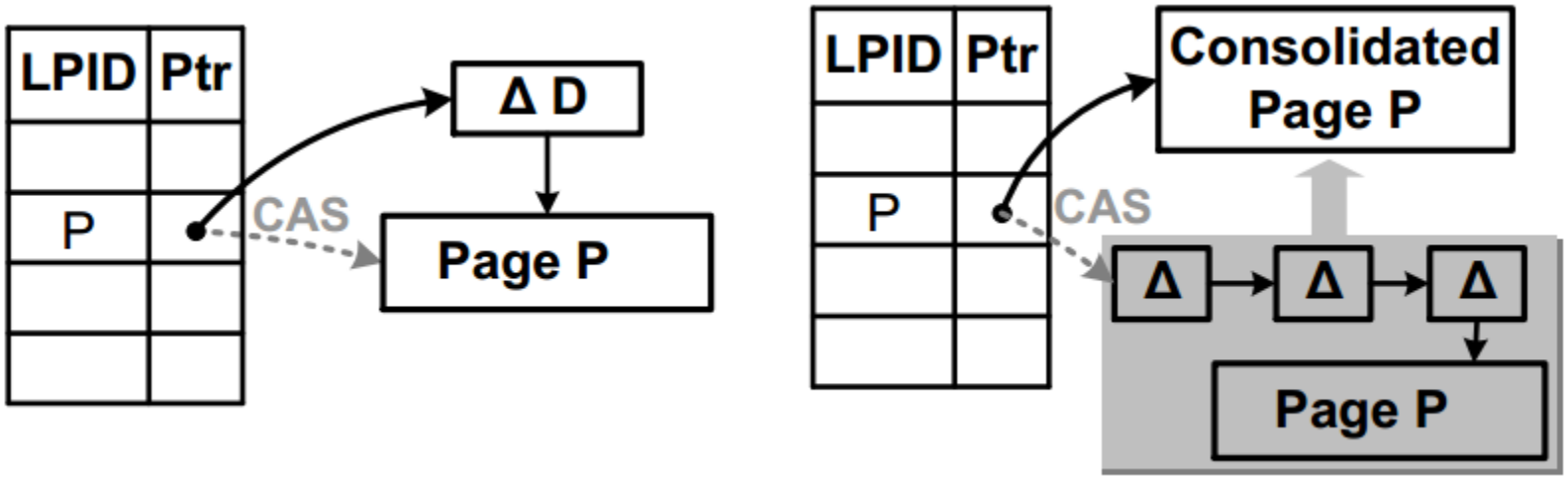


- bulk synchronous parallel processing model
- point-to-point synchronization
- software-prefetching and SIMD

BW-tree

figure courtesy of Justin Levandoski

[ICDE13c, PVLDB13a]



- latch-free log-structured B-tree
- optimized for both main memory and flash
- no updates in place -> delta updates

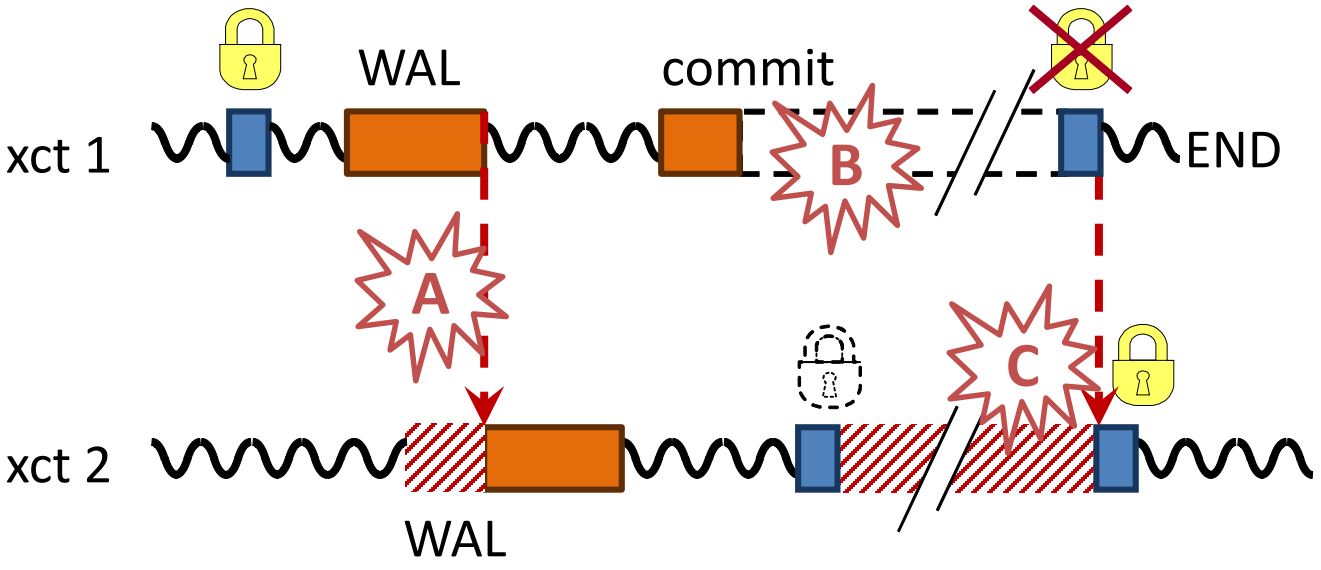
scaling up OLTP

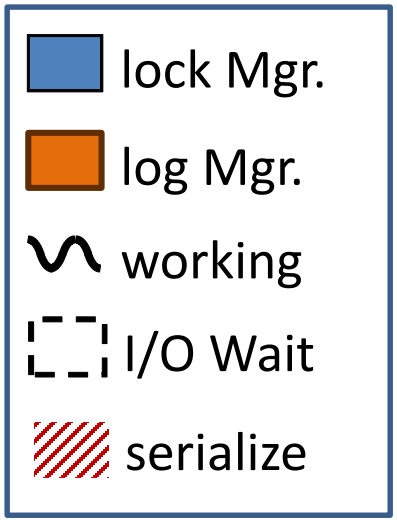
unscalable components
- locking
- latching
- logging

synchronization
- tradeoffs
- best practices

non-uniform communication
- hardware Islands

a day in the life of a serial log





A serialize at the log head

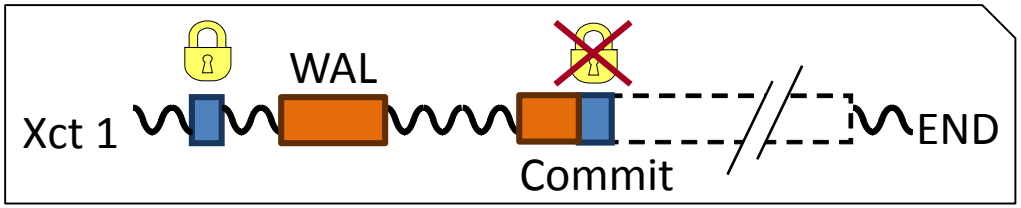
B I/O delay to harden the commit record

C serialize on incompatible lock

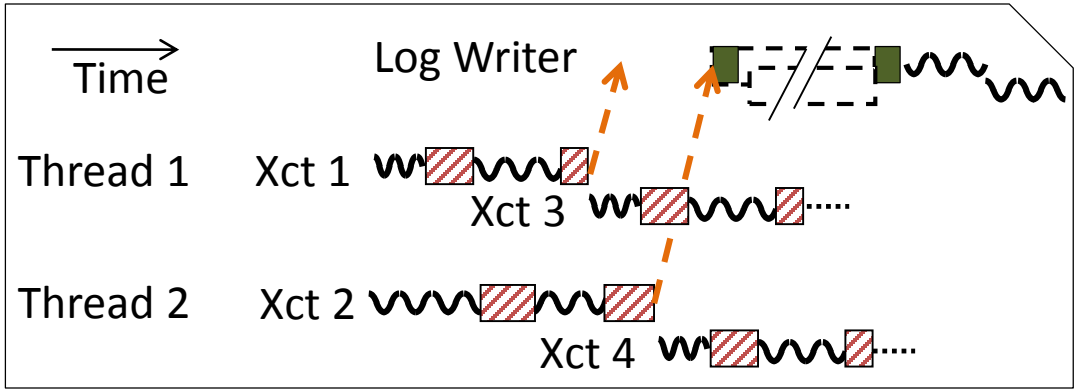
Aether holistic logging

[PVLDB10a]

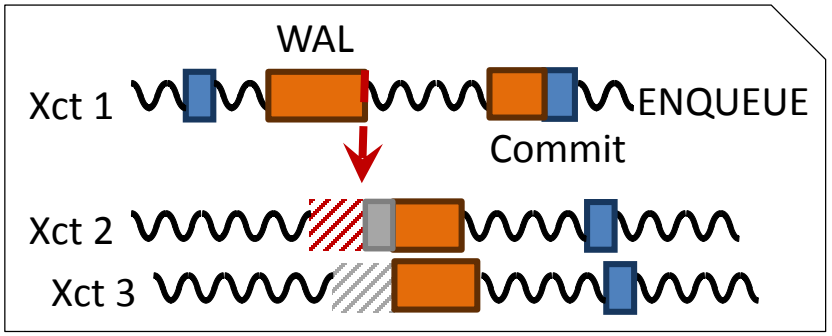
- early lock release
  - can be improved further with control lock violation [SIGMOD13b]



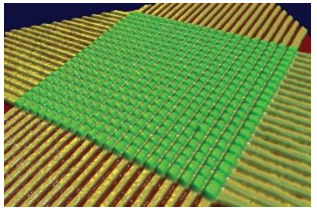
- flush pipelining
  - reduces context switches



- consolidation array
  - minimize log contention



NVRAM for instant durability



- group commit [PVLDB14e]
  - NV ram acts as a staging buffer for a transaction batch
- passive group commit [PVLDB14g]
  - durable log buffers
  - global sequence numbers
  - distributed log design
- NV-logging [PVLDB15d]
  - low overhead per transaction logging
- durable data structures
  - in-memory recoverable data structures [PVLDB15a]
  - write atomic cache-aware B-trees [PVLDB15b]

scaling up OLTP

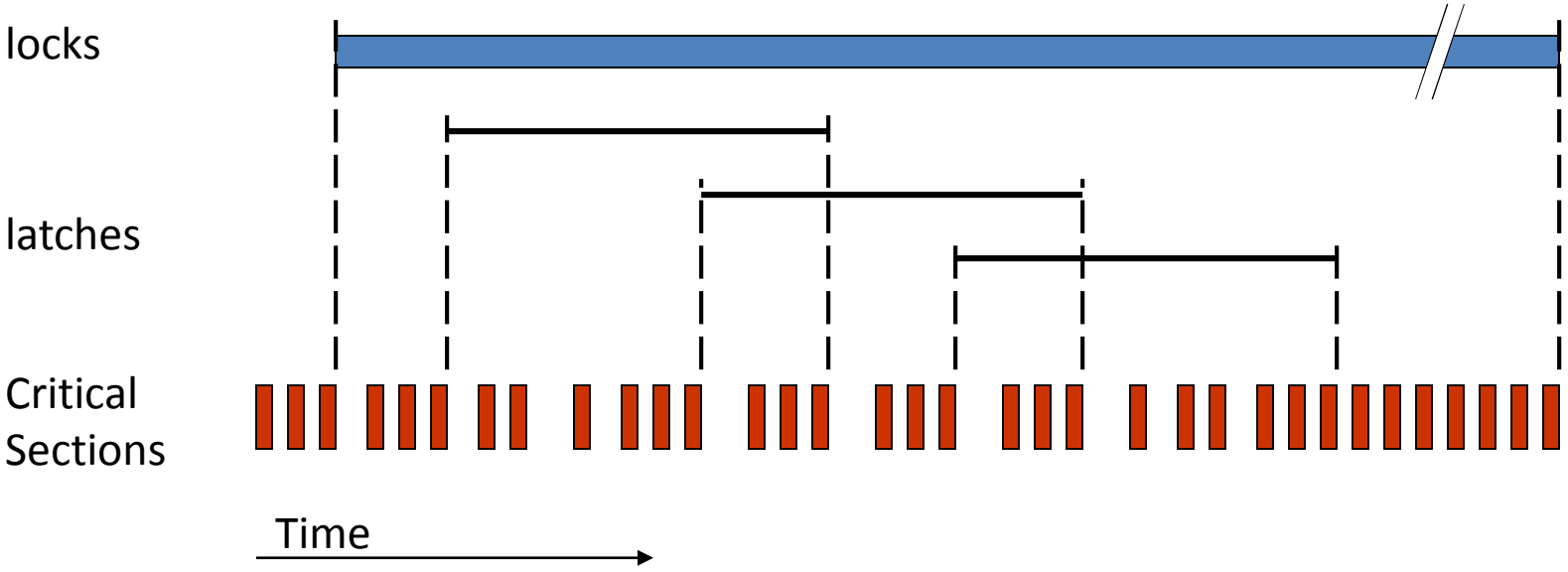
unscalable components
- locking
- latching
- logging

synchronization
- tradeoffs
- best practices

non-uniform communication
- hardware Islands

other unbounded communication

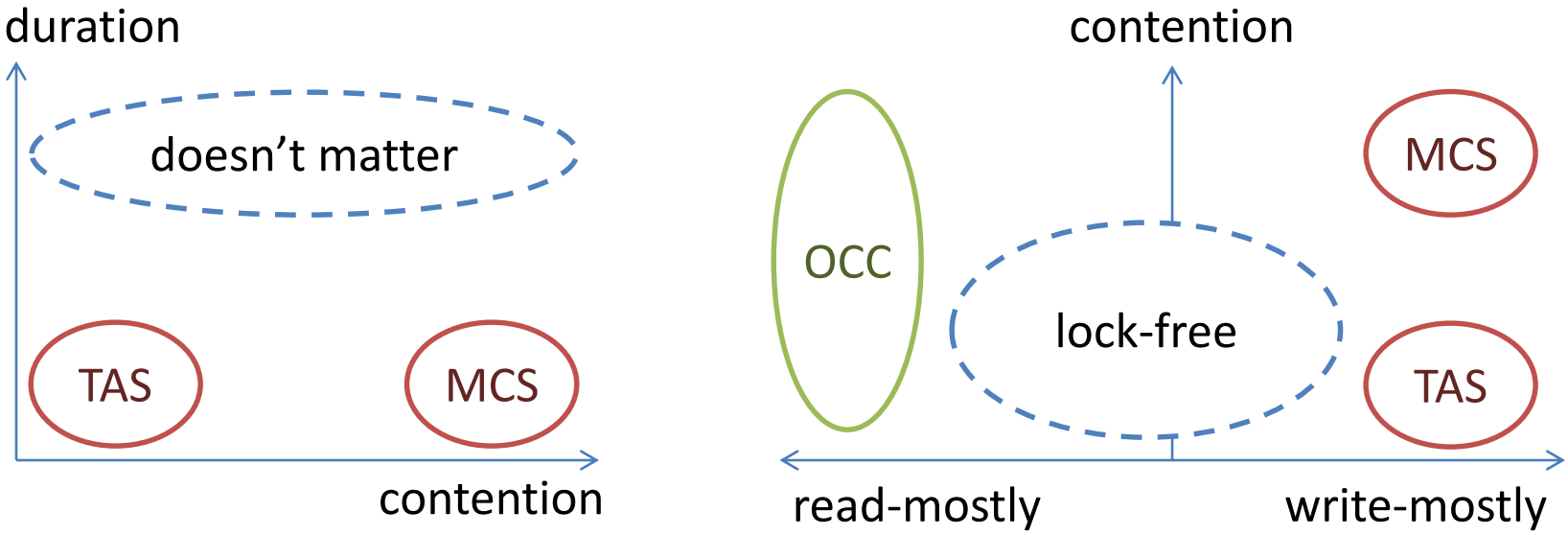
synchronization required for one index probe



- critical sections protect log buffer, stats, lock and latch internal state, thread coordination…

diverse use cases – how to select the best primitive?

synchronization "cheat sheet"

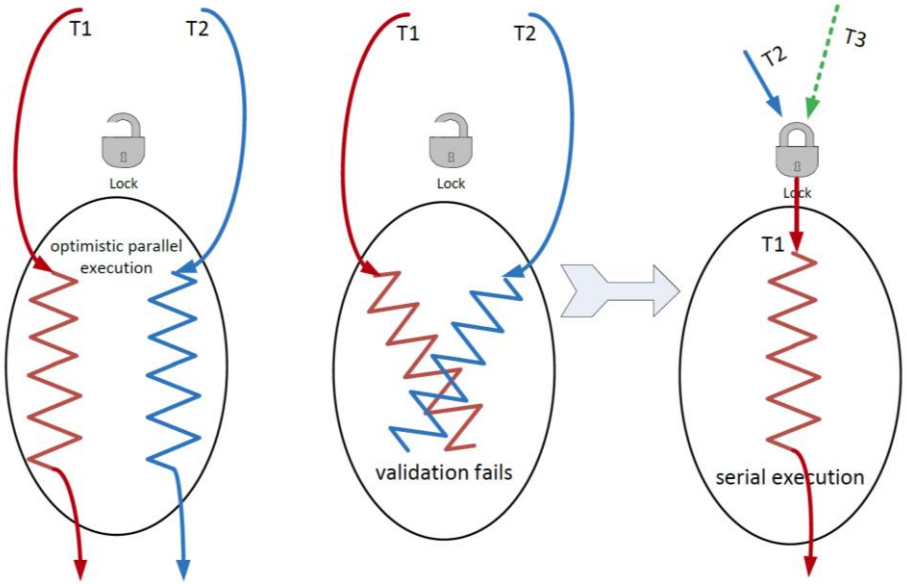


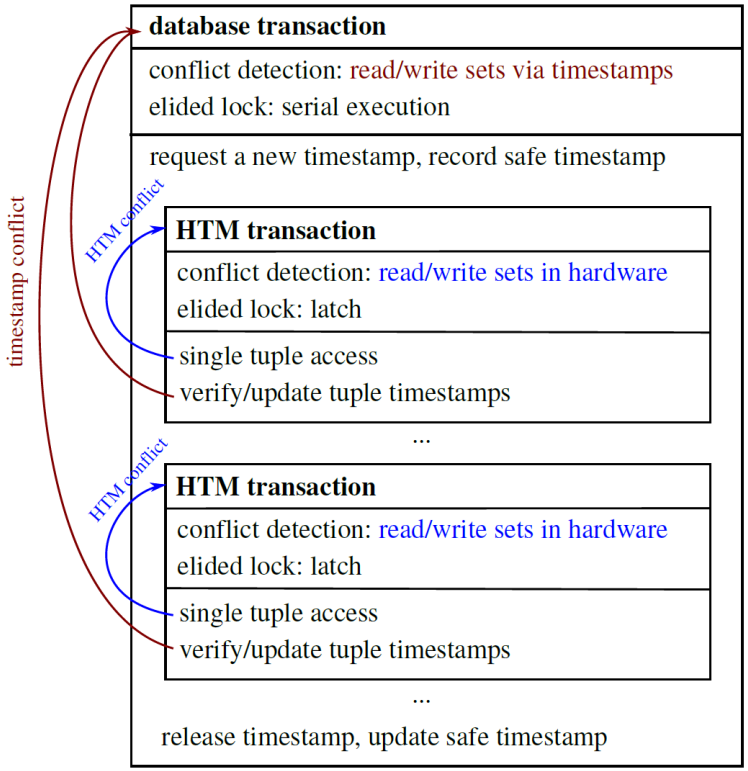
- OS blocking mutex: only for scheduling
- reader-writer lock: dominated by OCC/MCS
- lock-free: sometimes (but be very, very careful)

figure courtesy of Viktor Leis

concurrency control using HTM

[ICDE14b, Eurosys14]





promising, despite limitations of current chips

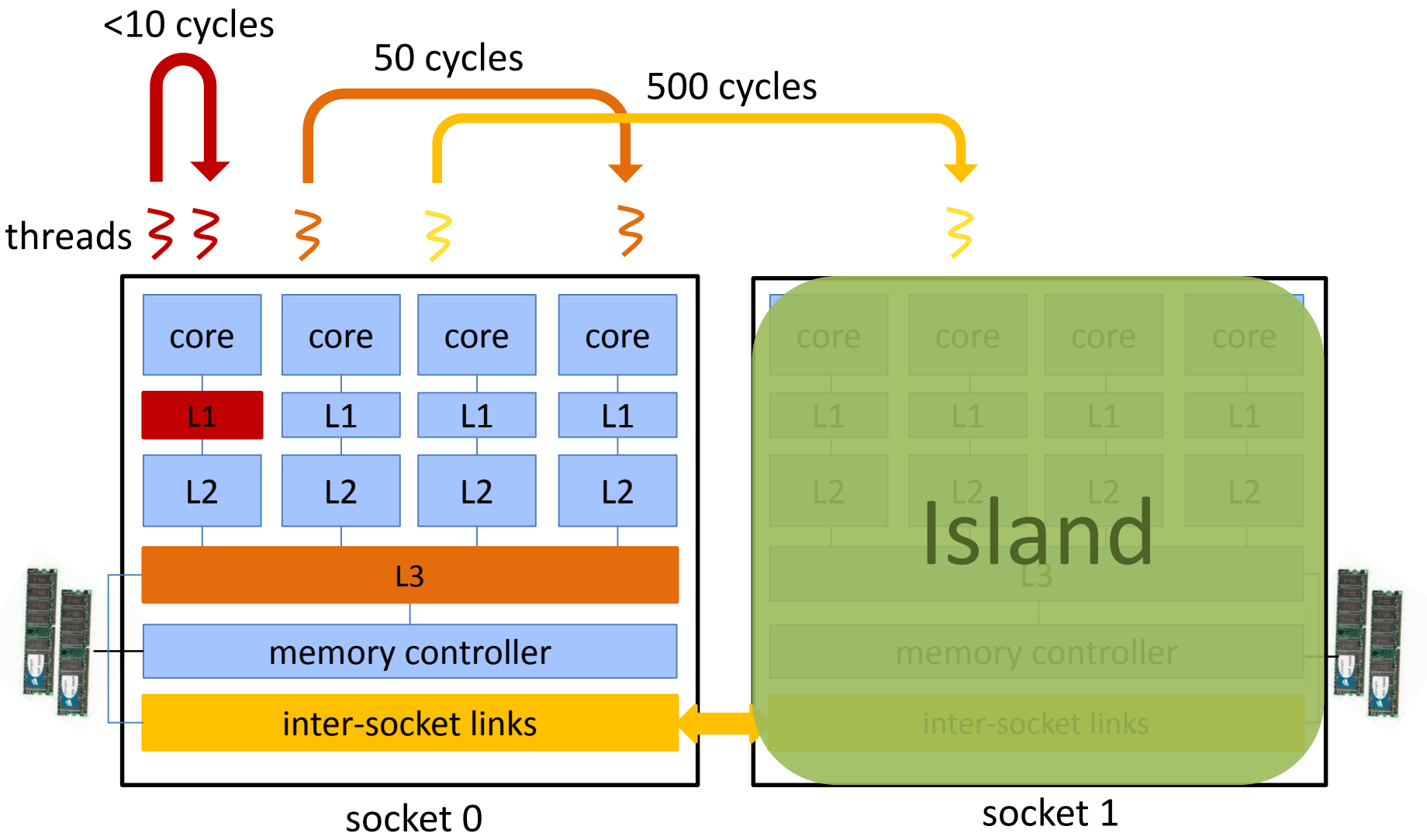
scaling up OLTP

unscalable components
- locking
- latching
- logging

synchronization
- tradeoffs
- best practices

non-uniform communication
- hardware Islands

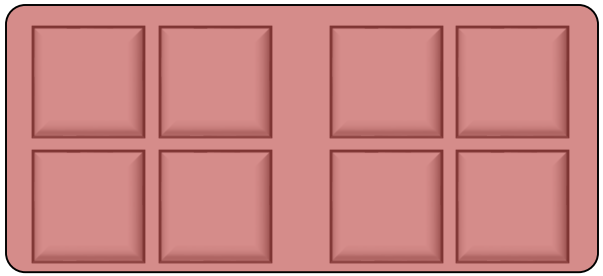
multisocket multicores



communication latencies vary by order-of-magnitude

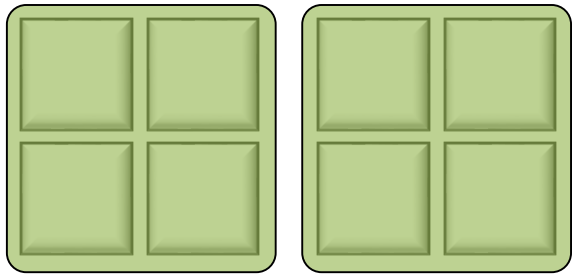
OLTP on Hardware Islands

[PVLDB12d]



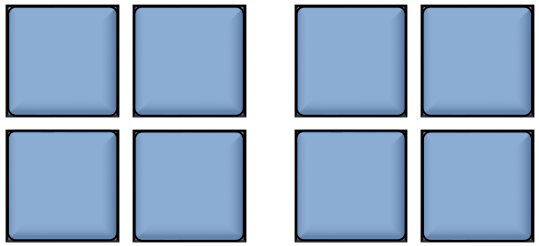
shared-everything

- ✓ stable
- ✗ not optimal



Island shared-nothing

- ✓ robust middle ground



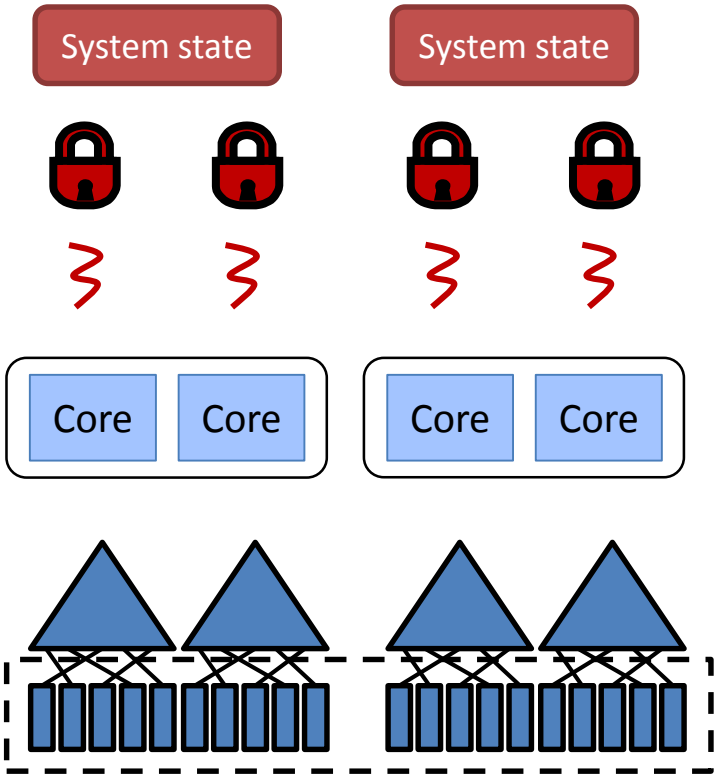
shared-nothing

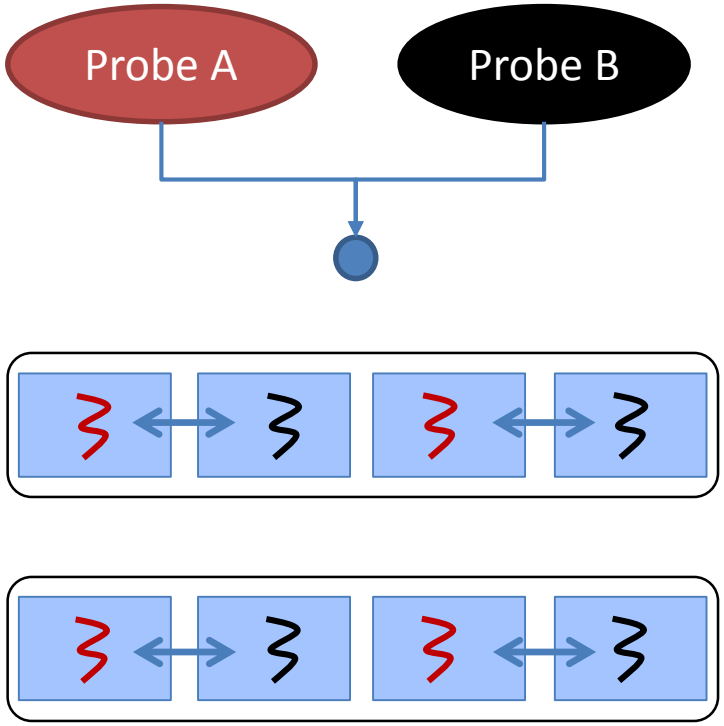
- ✓ fast
- ✗ sensitive to workload

- challenges
  - optimal configuration depends on workload and hardware
  - expensive repartitioning due to physical data movement

ATraPos: Adaptive Transaction Processing

[ICDE14d]

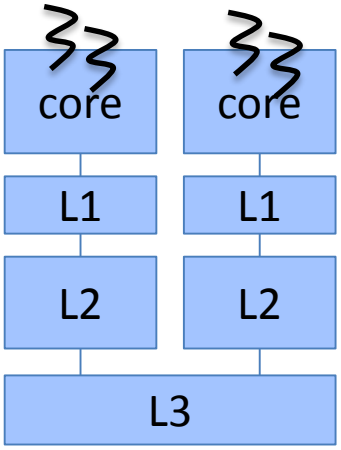


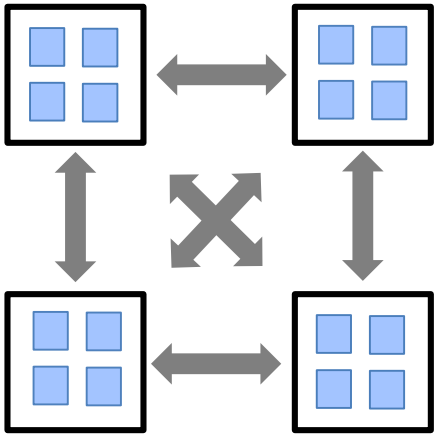


scaling up OLTP

- identify bottlenecks in existing systems
  - eliminate bottlenecks systematically and holistically
- design new system from the ground up
  - without creating new bottlenecks
- do not assume uniformity in communication
- choose the right synchronization mechanism

utilization **scalability**





exploiting core's resources
minimizing memory stalls

scaling up OLTP
scaling up OLAP
conclusions

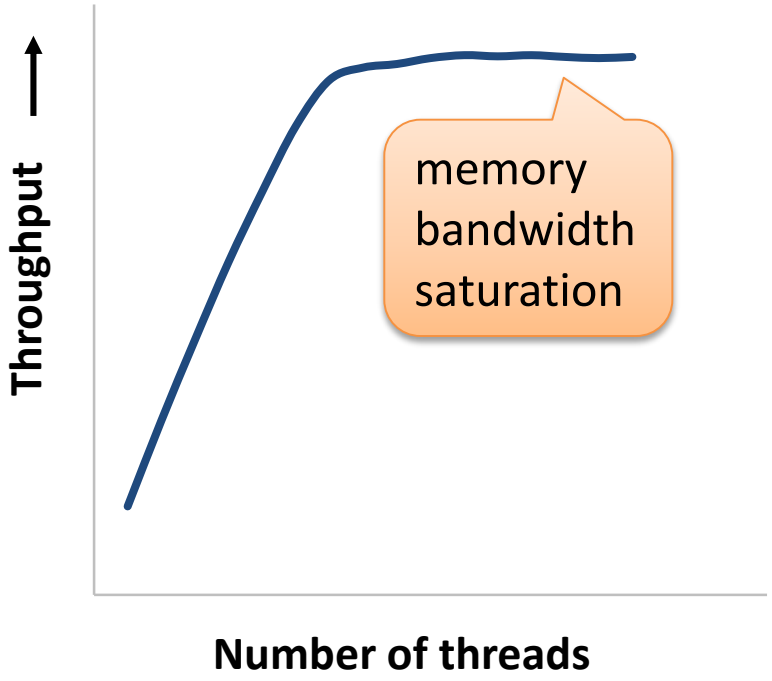
http://tinyurl.com/LoveMulticores

http://tinyurl.com/tutorial-2015-feedback

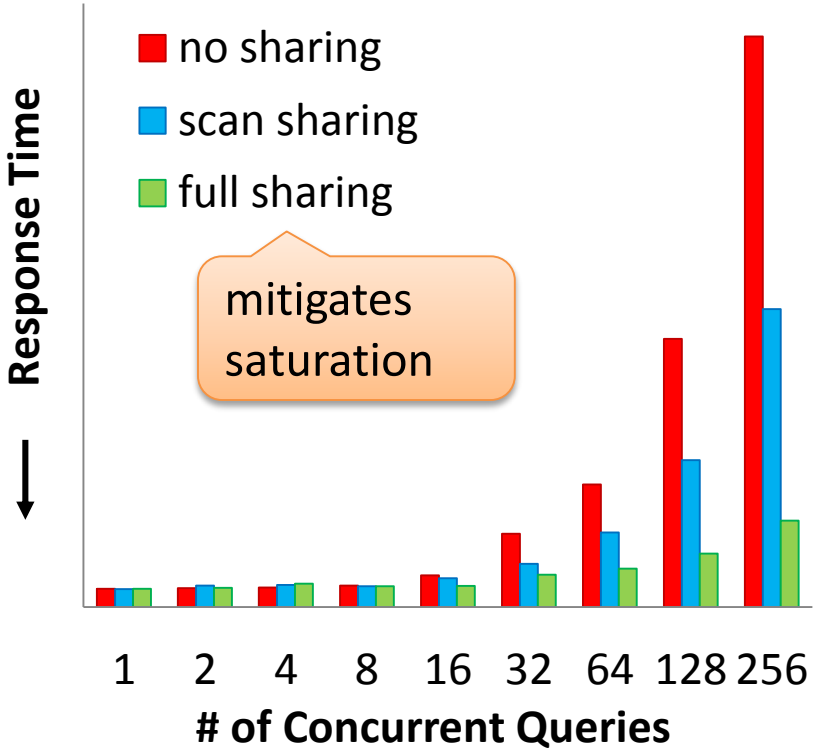
Scaling up OLAP

[DaMoN14b, SIGMOD14b]

parallelizing a single aggregation



sharing across queries

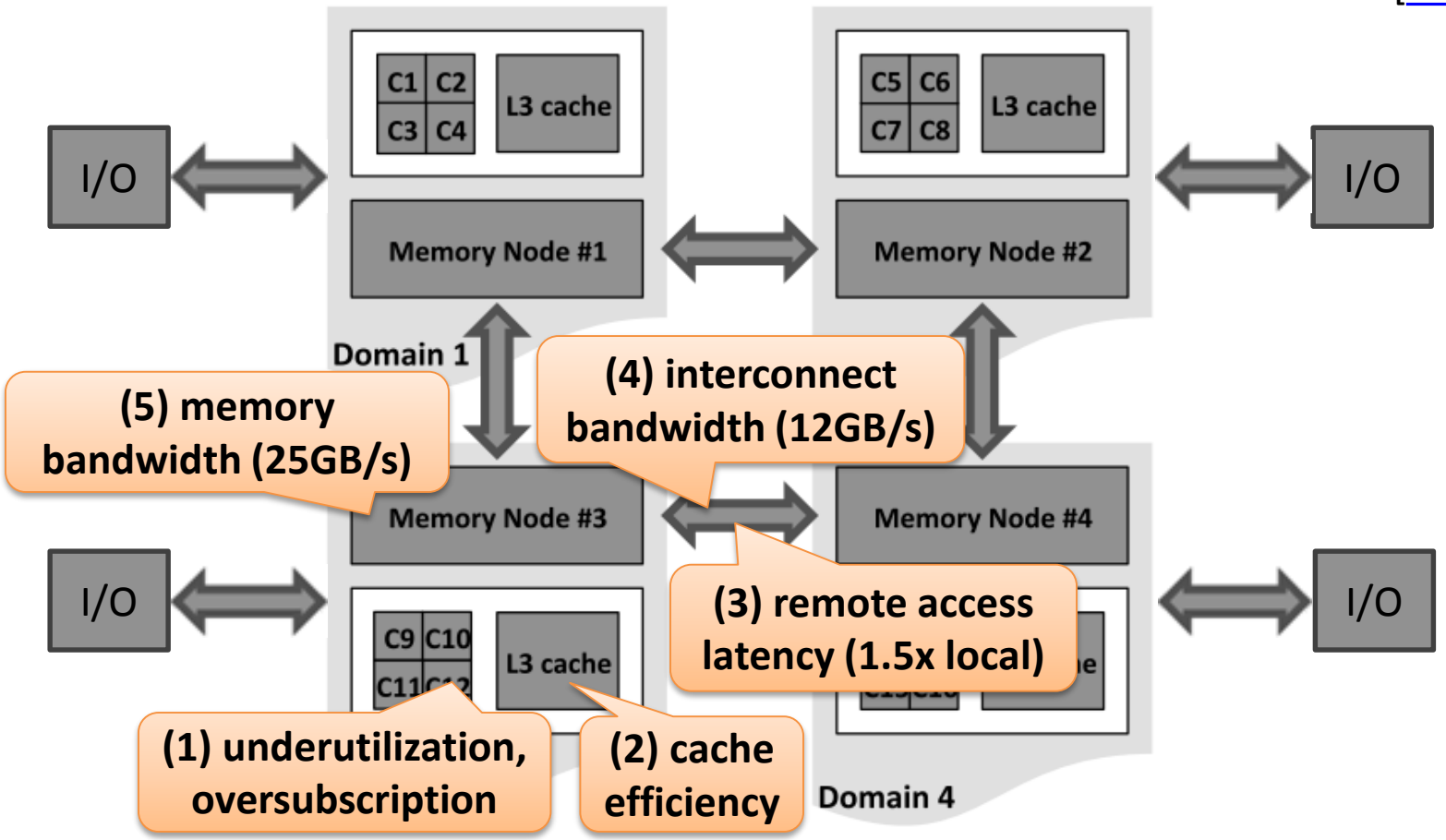


OLAP is concerned also with resources saturation

bottlenecks in NUMA architectures

figure courtesy of Blagodurov et al.

[USENIX11]



numerous points to consider for NUMA-awareness

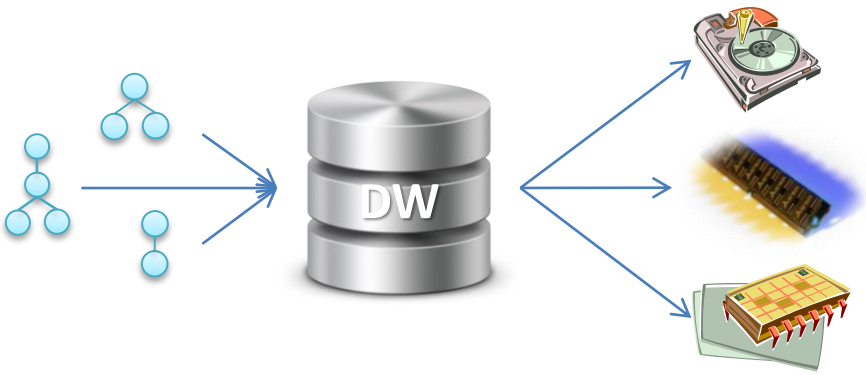
scaling up OLAP

sharing
- common sub-plans
- shared operators

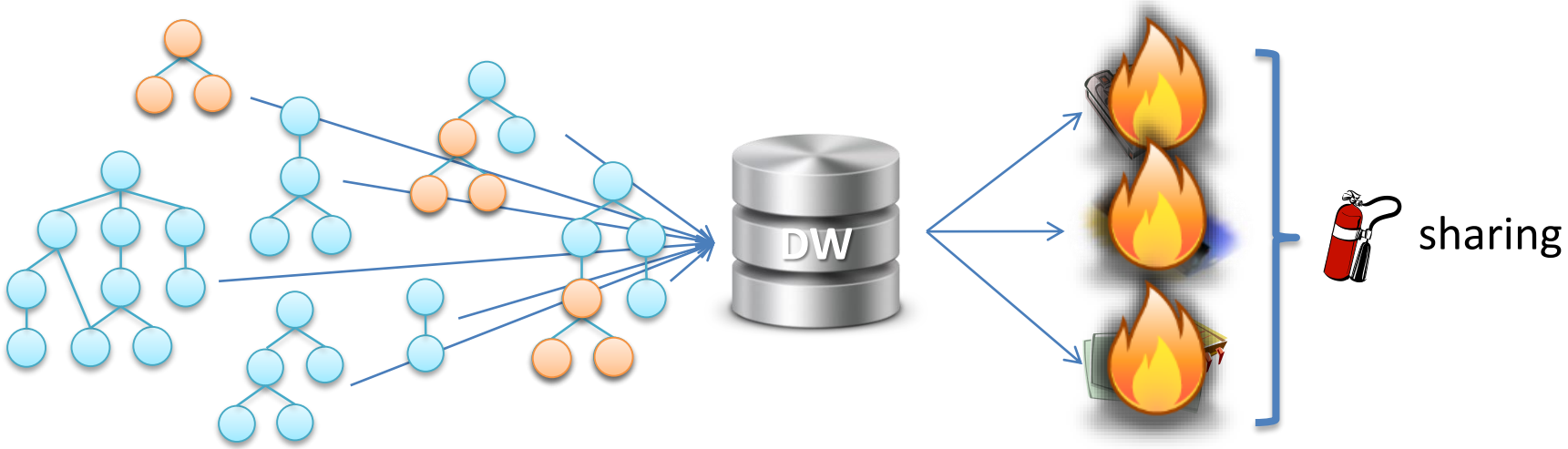
NUMA-awareness
- application-agnostic
- database operators

scheduling
- task scheduling
- NUMA-aware task scheduling

sharing is caring…



in the era of big data



…for resources

sharing techniques

[SIGMOD14b]

query-centric

- caching
- materialized views
- multi-query optimization
- buffer pool management

reactive sharing

- query-centric
- shares common sub-plans
- shared scans

proactive sharing

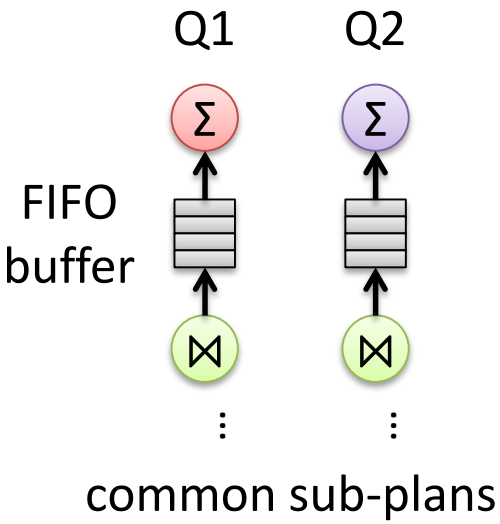
- global query plan with shared operators
- shared scans

how and when should we use each technique?

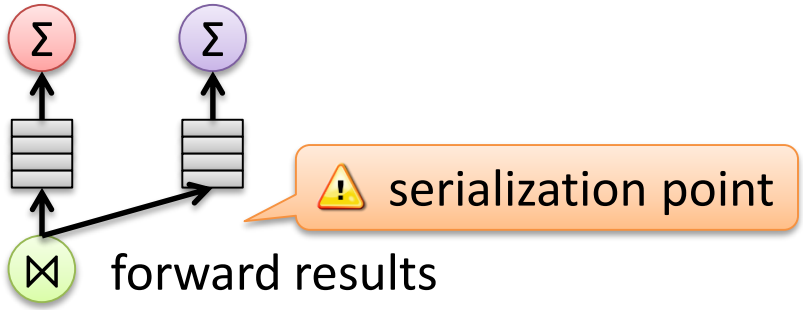
reactive sharing: how to react?

[VLDB07a, PVLDB13b]

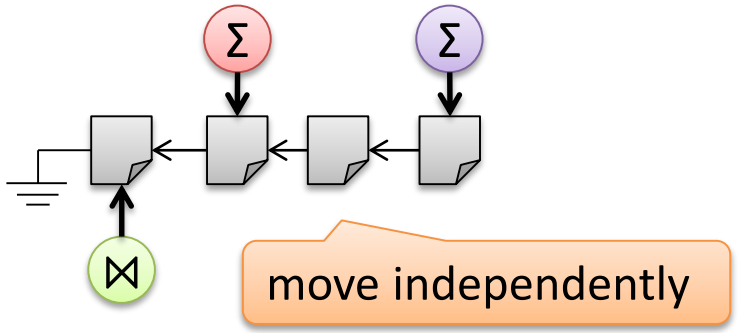
query-centric



Push



Pull



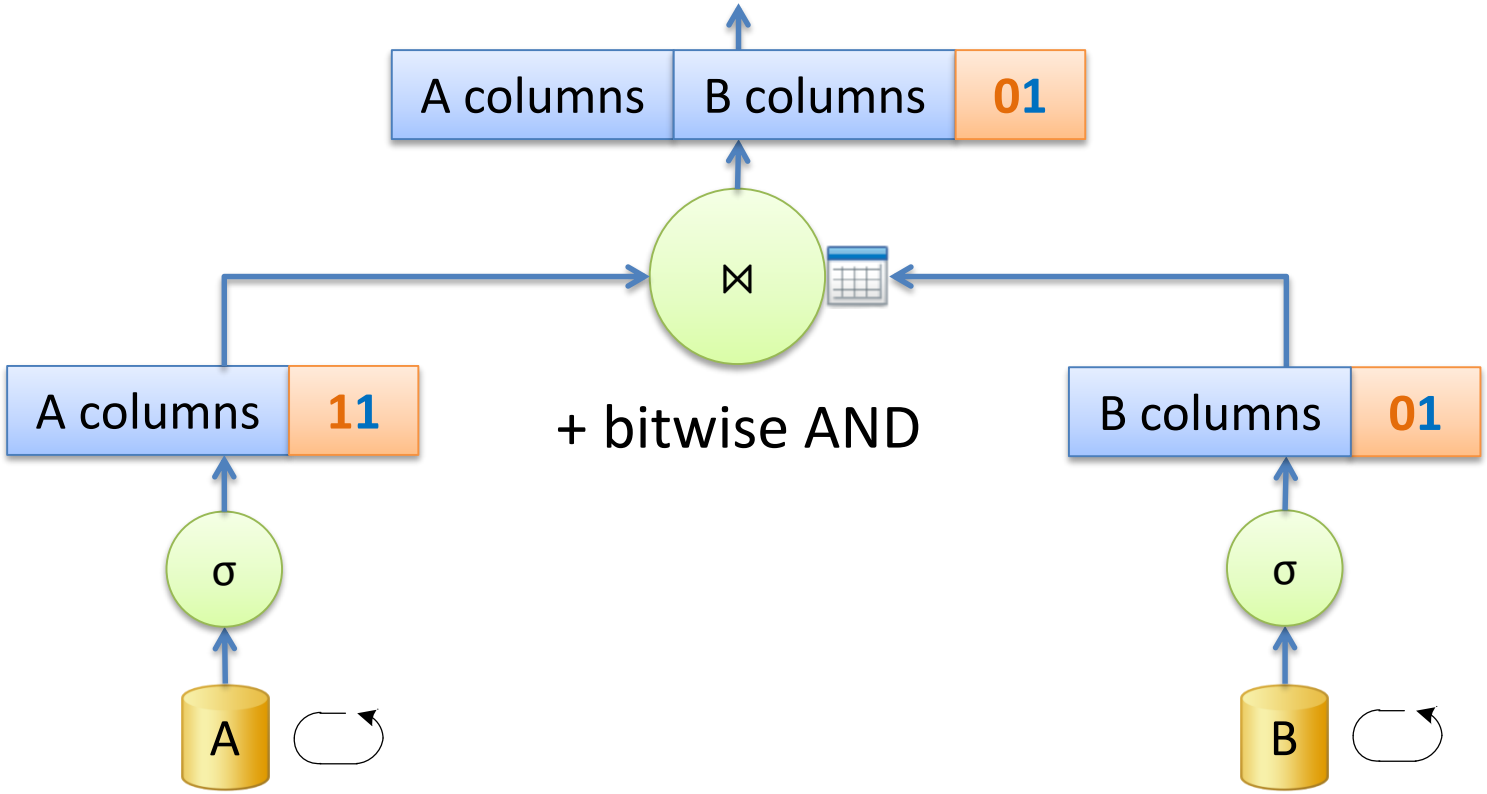
by pulling shared intermediate results

proactive sharing

[VLDB09a]

Q_1 SELECT * FROM *A, B*
WHERE $A.c_1 = B.c_1$
AND **σ(*A*)** AND **σ(*B*)**

Q_2 SELECT * FROM *A, B*
WHERE $A.c_1 = B.c_1$
AND **σ'(*A*)** AND **σ'(*B*)**



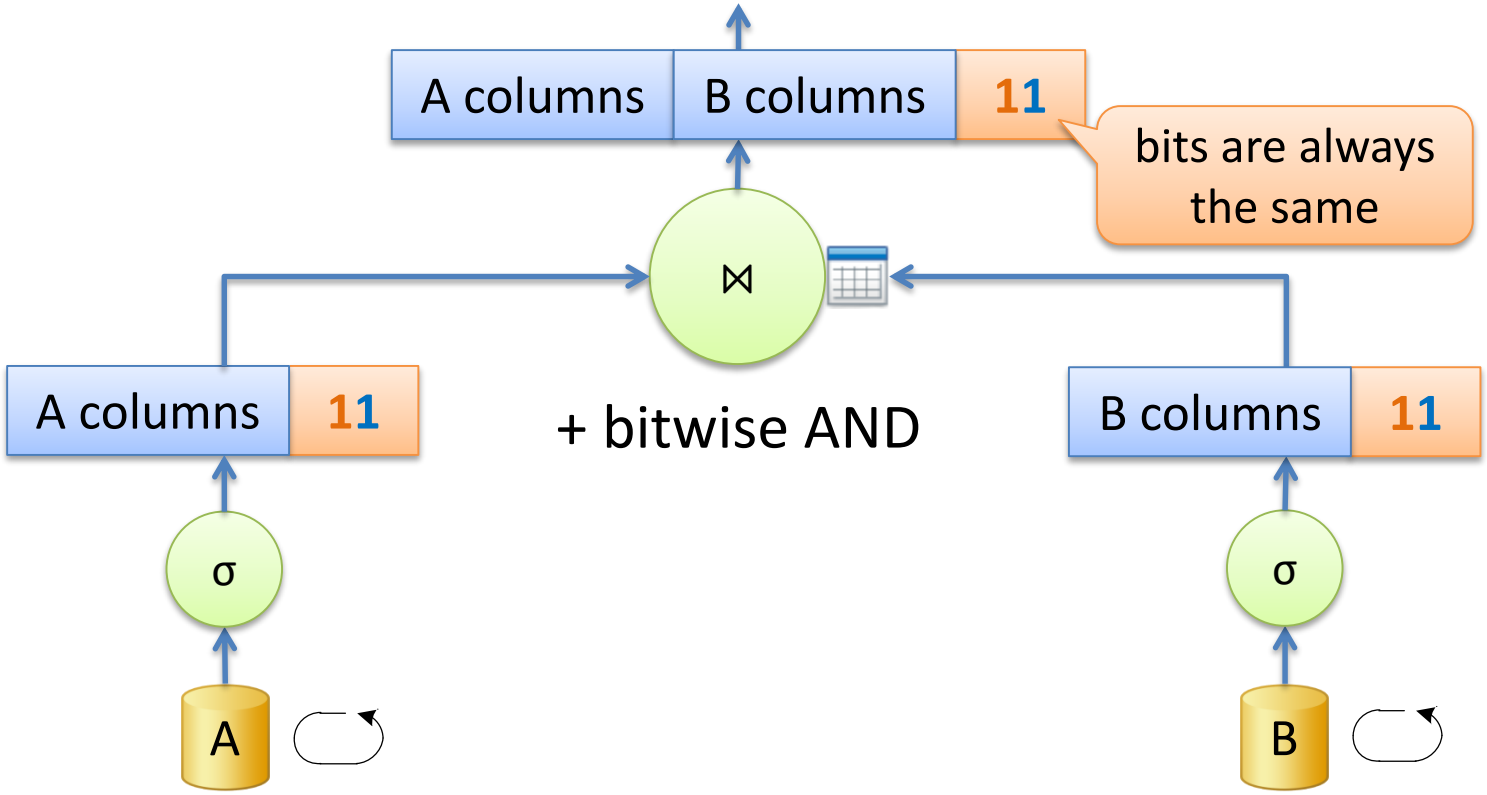
shared operators can support high throughput

proactive + reactive sharing

[PVLDB13b]

Q_1 SELECT * FROM *A*, *B* WHERE $A.c_1 = B.c_1$ AND **σ(*A*)** AND **σ(*B*)**

Q_2 SELECT * FROM *A*, *B* WHERE $A.c_1 = B.c_1$ AND **σ(*A*)** AND **σ(*B*)**

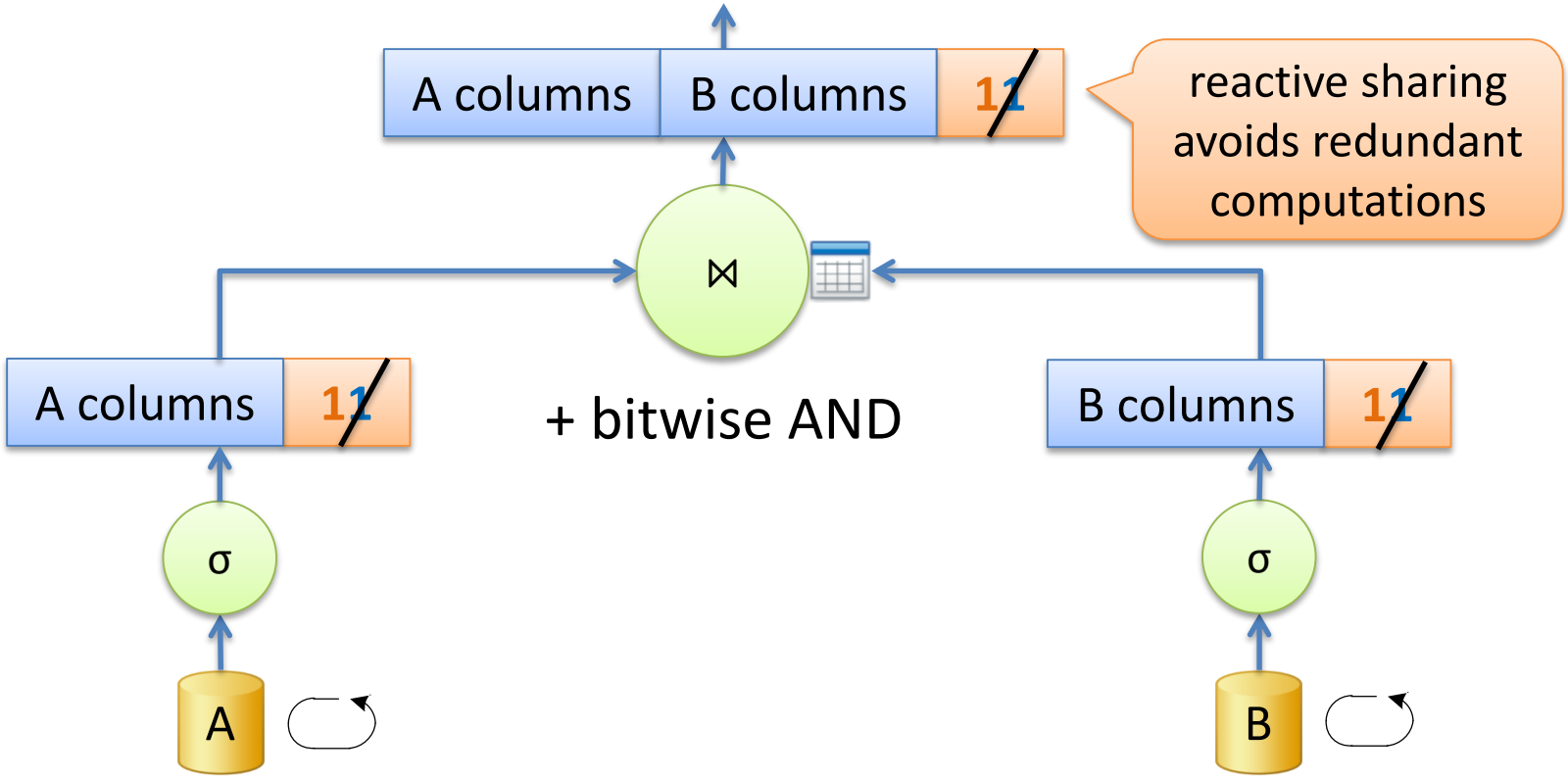


proactive + reactive sharing

[PVLDB13b]

Q_1 SELECT * FROM *A*, *B* WHERE $A.c_1 = B.c_1$ AND **σ(*A*)** AND **σ(*B*)**

Q_2 SELECT * FROM *A*, *B* WHERE $A.c_1 = B.c_1$ AND **σ(*A*)** AND **σ(*B*)**



reactive sharing can improve proactive sharing

sharing in practice

	QPipe [SIGMOD05]	**CJOIN** [VLDB09a]	**DataPath** [SIGMOD10a]	**SharedDB** [PVLDB12b, PVLDB14b]
sharing type	reactive	proactive (global query plan)		
execution	dynamic	dynamic	dynamic	batched
schema	general	star	general	general (pre-comp.)
I/O	circular scans	circular scans	linear scan of a disk array	main-memory circ. scans

share responsibly

[PVLDB13b, SIGMOD14b]

when to share	how to share
low concurrency	query-centric operators + reactive sharing
high concurrency	proactive sharing + reactive sharing

scaling up OLAP

sharing
- common sub-plans
- shared operators

NUMA-awareness
- application-agnostic
- database operators

scheduling
- task scheduling
- NUMA-aware task scheduling

application-agnostic NUMA-awareness

[HPCA13, USENIX11, ASPLOS13]

- black box approach
  - monitoring to predict behavior
- DINO scheduler
  - moves threads and their data to balance cache load
- Carrefour
  - re-organizes data to avoid memory bottlenecks
  - by: replicating, interleaving or co-locating data

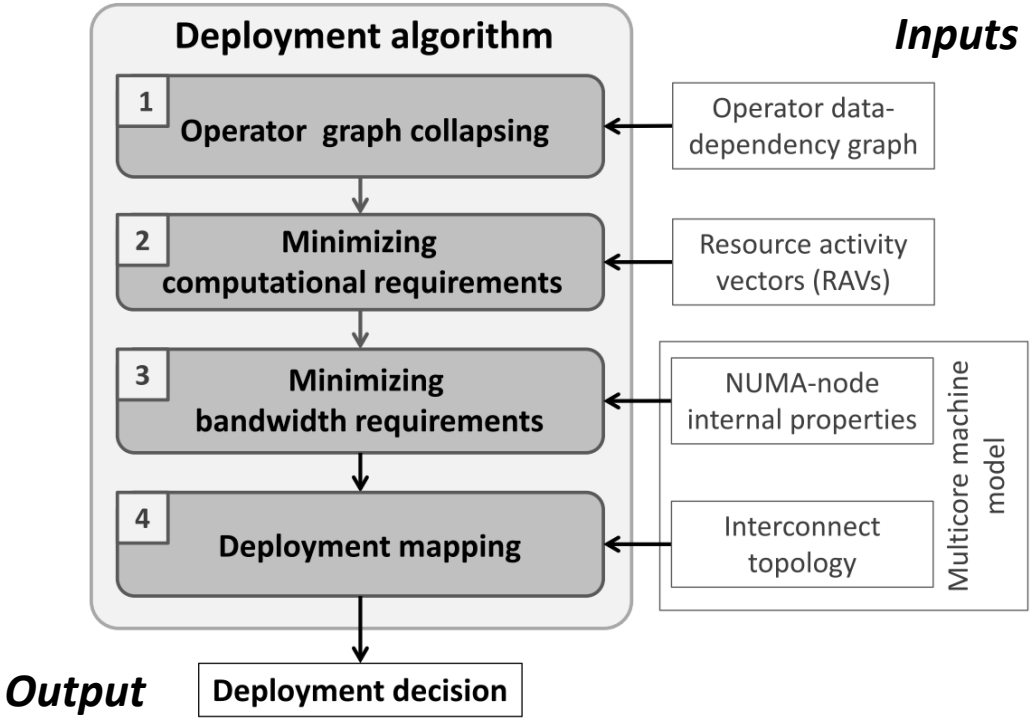
not always optimal for DBMS

figure courtesy of Giceva et al.

black box for global query plan

[PVLDB15c]

- resource activity vectors (RAV)
  - measured CPU and memory intensity of shared operators



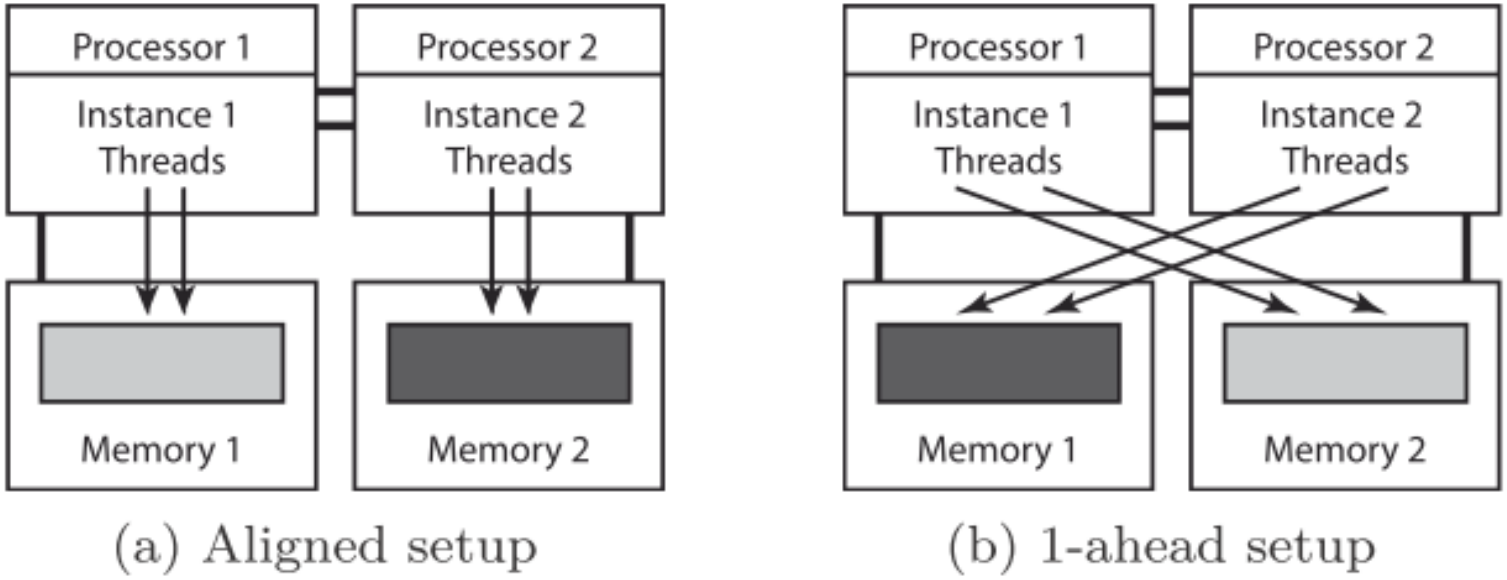
same performance as OS with 14% of resources

impact of NUMA

figure courtesy of Kiefer et al.

[BTW13]

- data partitions accessed by different clients
  - co-locate threads and data they access

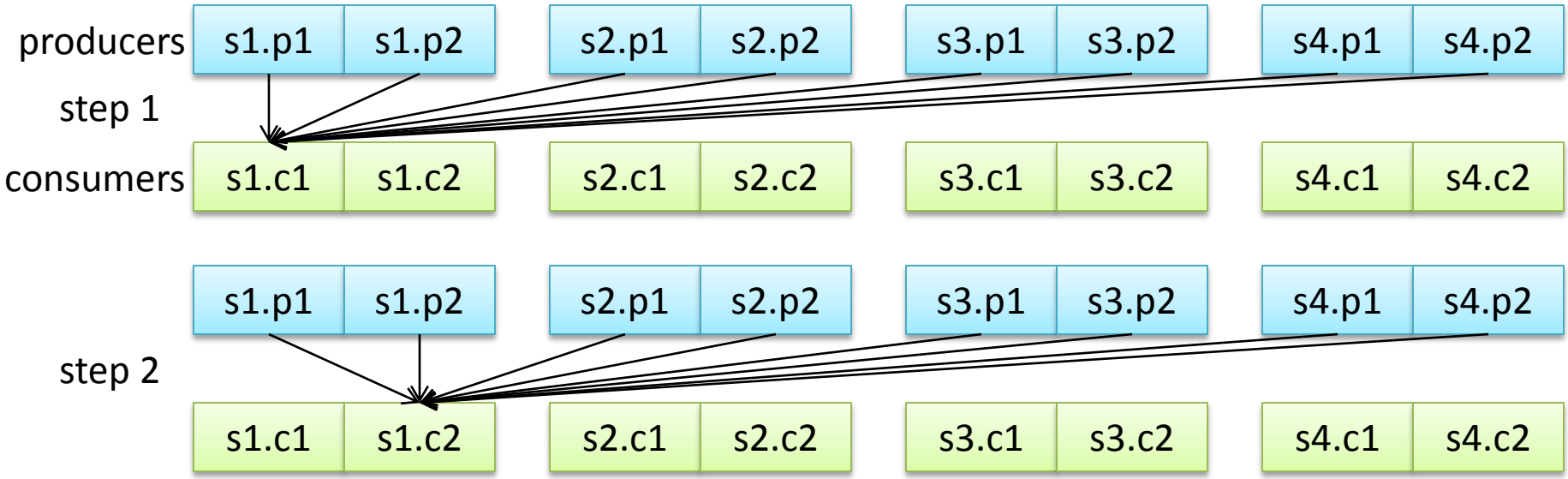


up to 75% improvement

data shuffling

[CIDR13b]

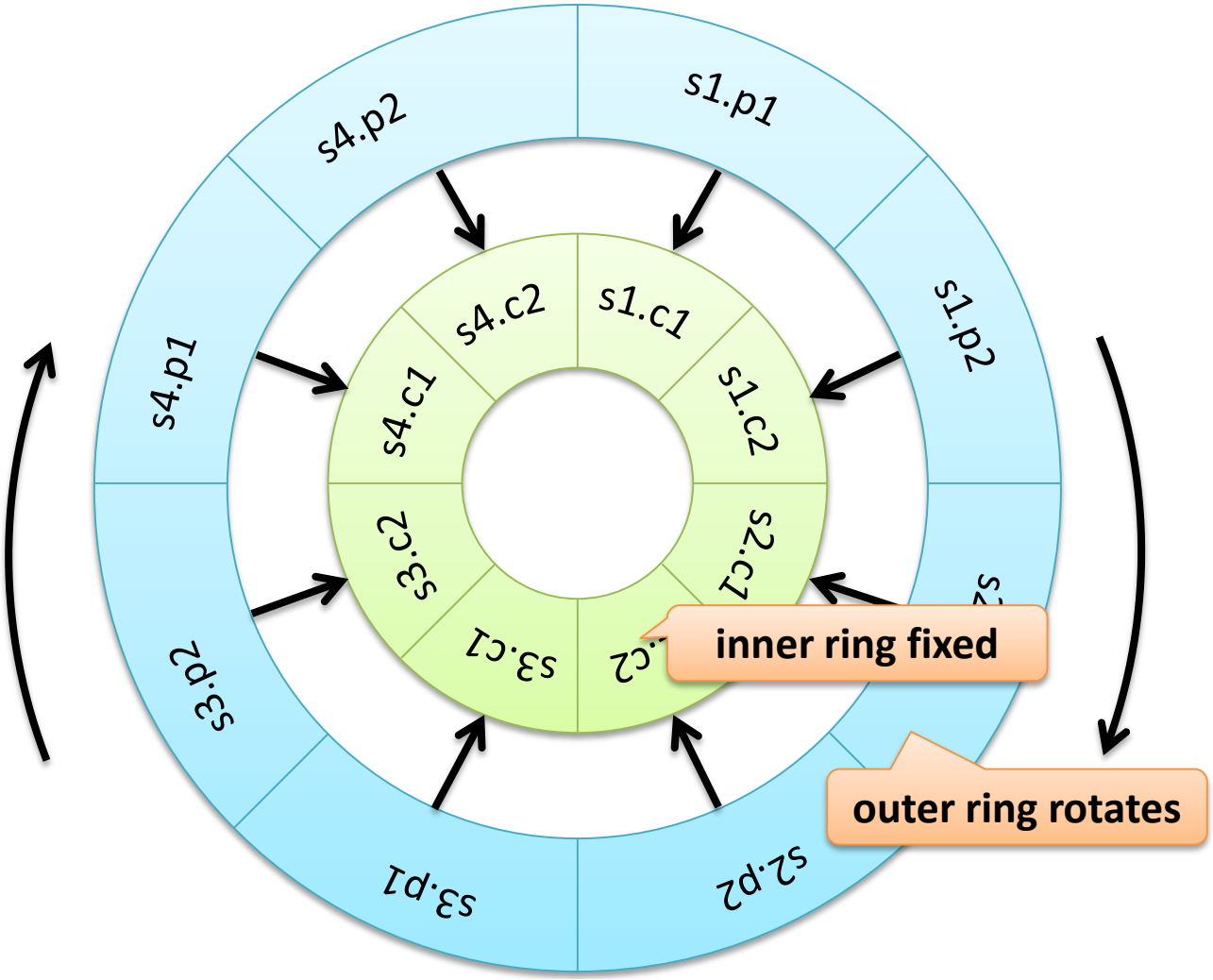
- ***N*** threads, each partitions its local data into ***N*** equally-sized pieces, transmitted to the rest
- naïve method:



saturates memory and interconnects

coordinated shuffling

[CIDR13b]

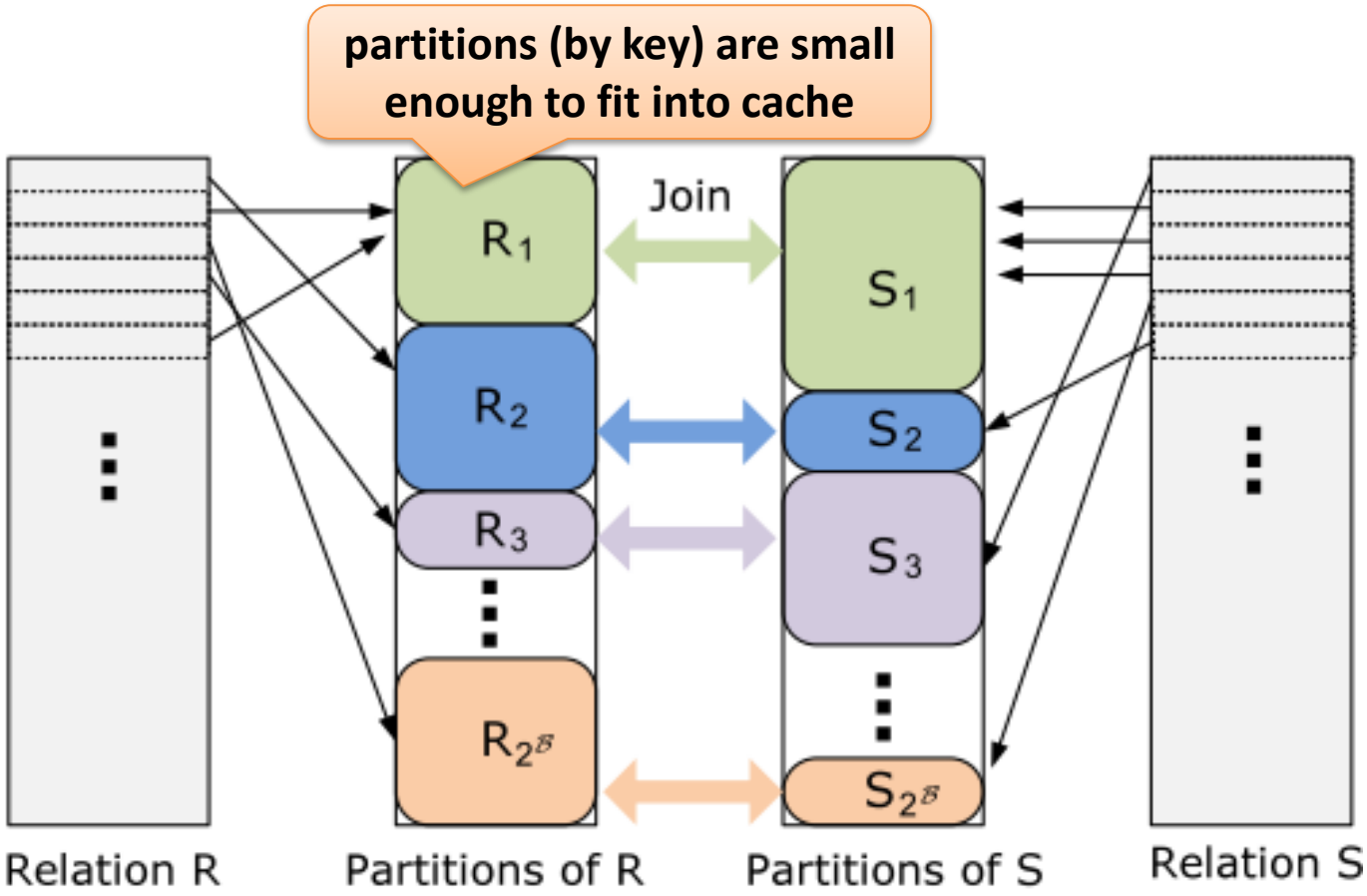


balances memory and interconnect traffic

radix hash join

figure courtesy of Kim et al.

[VLDB09c]



cache-efficient but not NUMA-aware

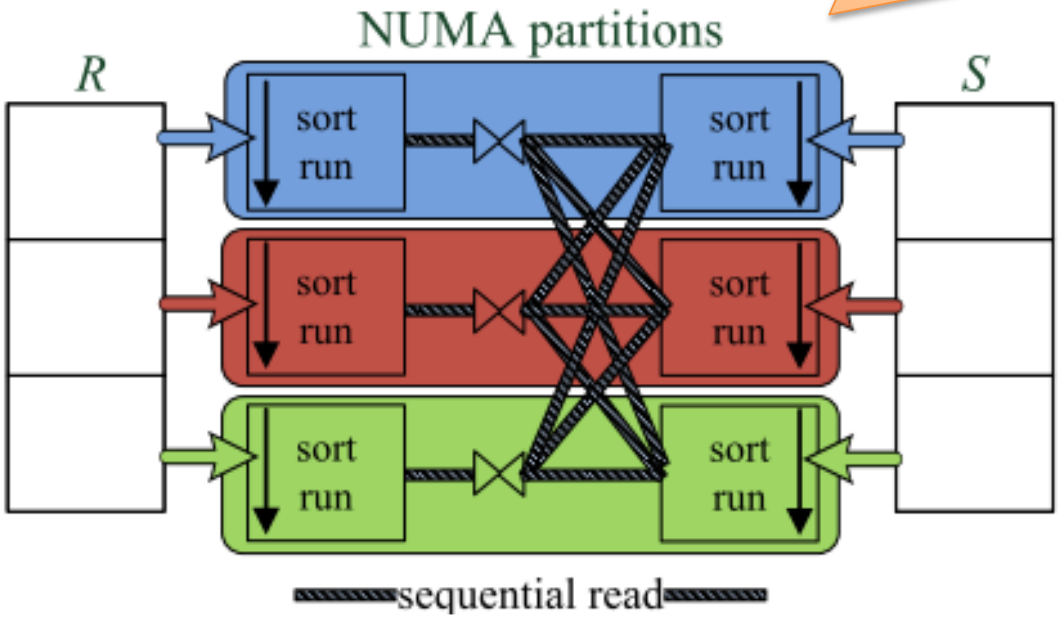
figure courtesy of Albutiu et al.

massively parallel sort-merge join

[PVLDB12a]

- NUMA-awareness rules:
  - no remote random writes
  - sequential remote reads
  - no synchronization

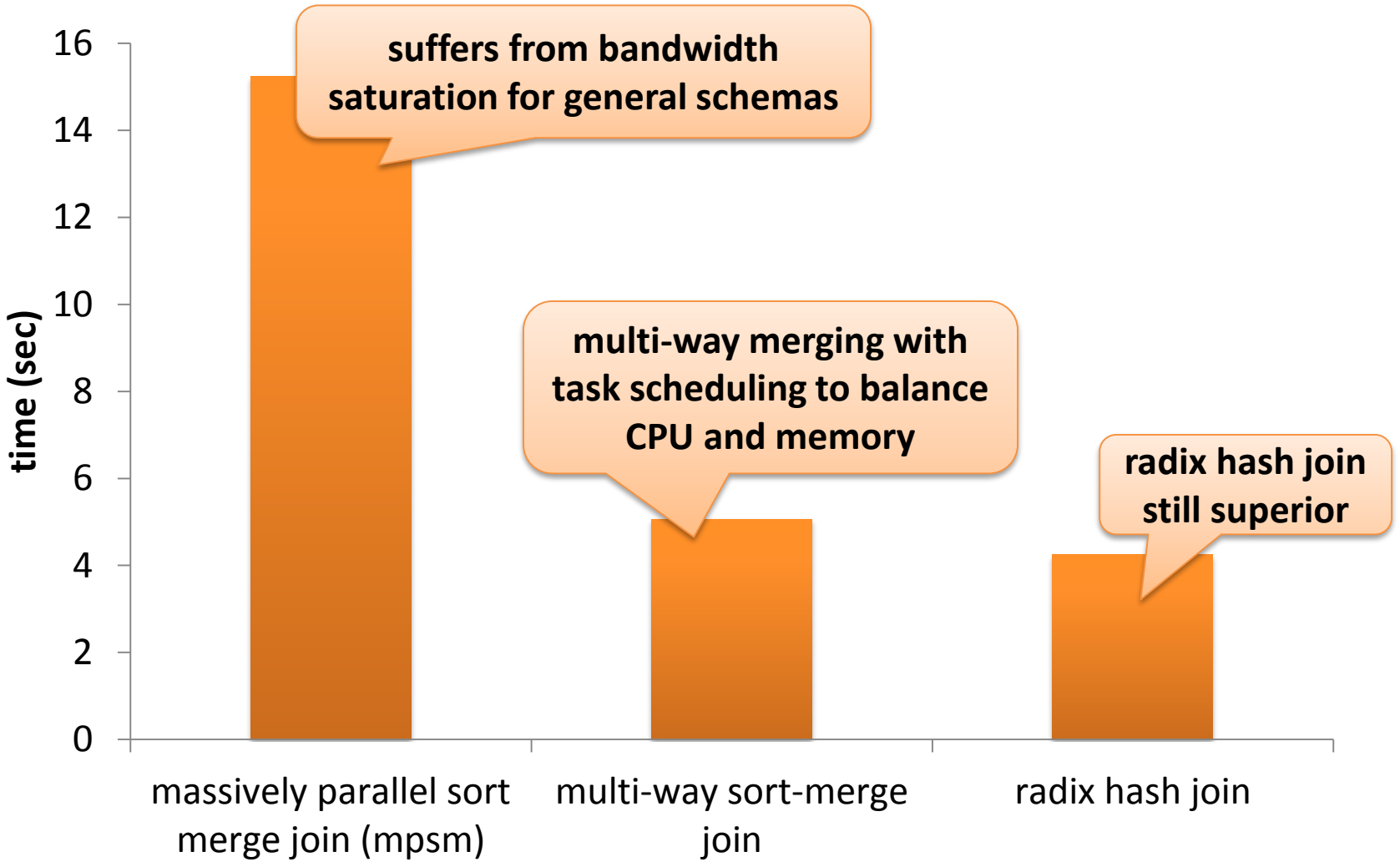
faster than radix hash-join for star schemas



remote random accesses > remote scans

sort-merge join forever?

[PVLDB14a]



a long-standing battle

scaling up OLAP

sharing
- common sub-plans
- shared operators

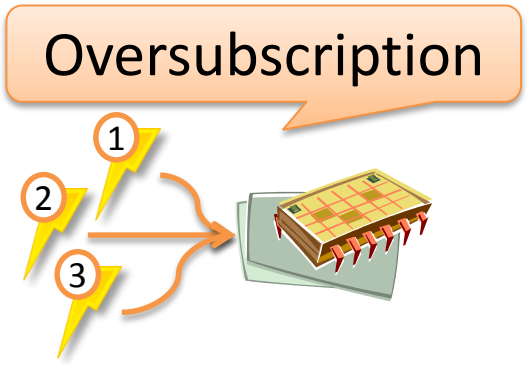
NUMA-awareness
- application-agnostic
- database operators

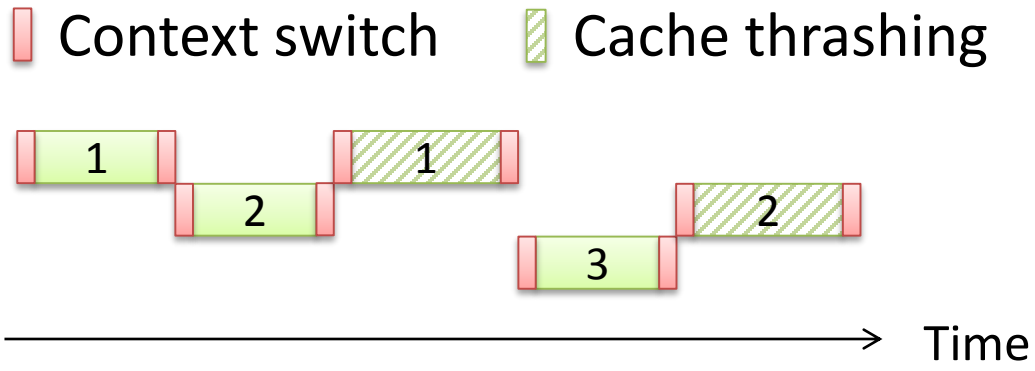
scheduling
- task scheduling
- NUMA-aware task scheduling

scheduling work

[ADMS13]

- OS scheduler

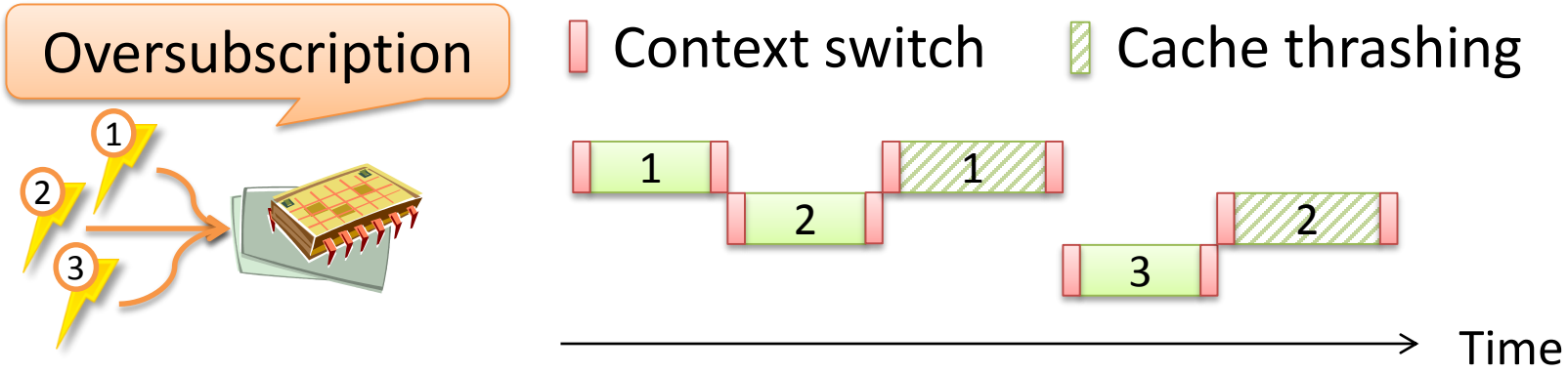




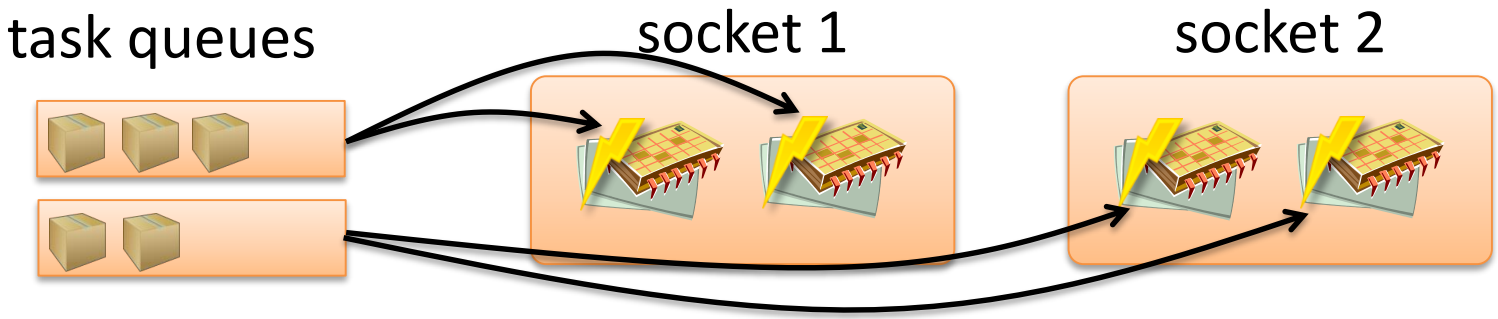
scheduling work

[ADMS13]

- OS scheduler



- task scheduler



a solution for DBMS to efficiently utilize resources

opportunities and challenges

[ADMS13, DSAA14, ICDE13a, PCS13]

opportunities	***advantages***
decouple from OS	full control and predictability
task granularity	balance CPU and memory parallelism
task prioritization	workload management

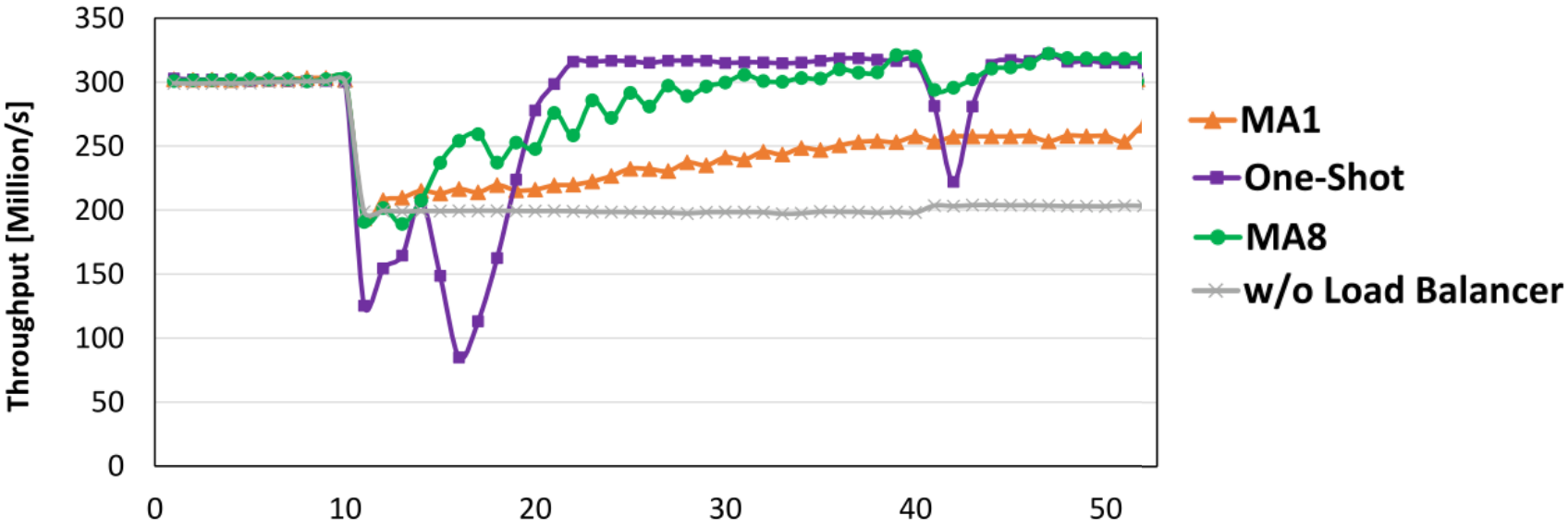
challenges	***solutions***
unbalanced task queues	stealing
NUMA-awareness	affinities restricted stealing
blocking tasks	co-operative scheduling flexible #threads
task granularity	depending on saturation

figure courtesy of Kissinger et al.

task scheduling for storage engine

[ADMS14]

- ERIS
  - NUMA-aware shared scans and shared index lookups
  - Each worker has a partition of a data object
  - Message passing between workers



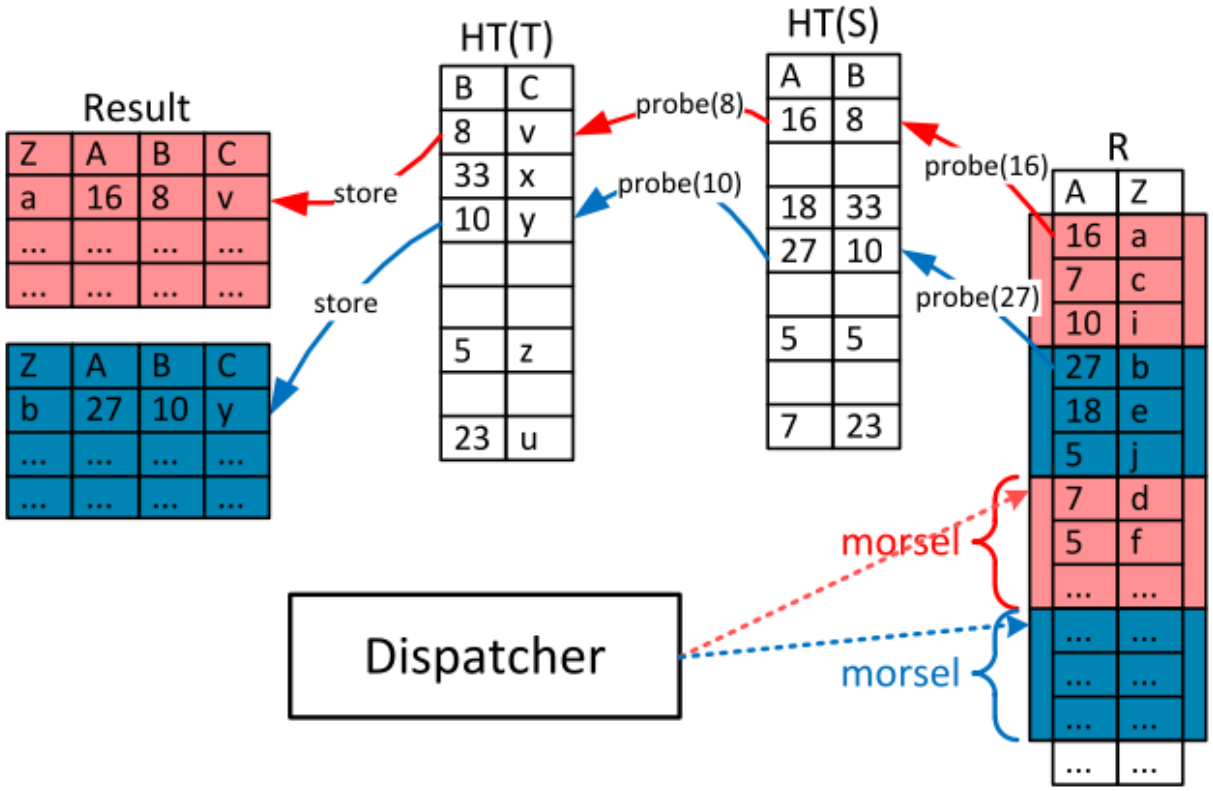
dynamic load balancing based on workload

task scheduling for OLAP

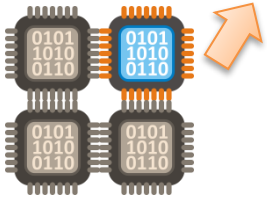
figure courtesy of Leis et al.

[SIGMOD14a]

- HyPer
  - Partition data by small morsels across sockets
  - Pass morsels through whole operator pipelines



task scheduling for energy efficiency [DaMoN14b]

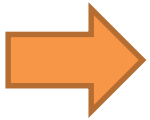
- DVFS 1.2GHz  2.9GHz
- Turbo boost  > 2.9GHz
- C-states 
- CPU H/W counters 

Calibration analysis

of operators and parameters

Measurements

hardware counters and/or external equipment



Runtime decisions

task scheduling, data placement, power management

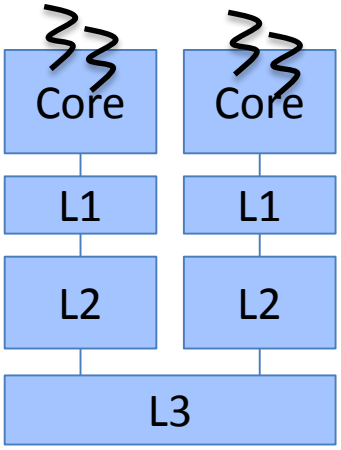
DVFS, task/data placement improves energy efficiency

embrace…

- sharing
  - reduces contention for resources
  - reactive and proactive
- NUMA-awareness
  - reduce latency and avoid bottlenecks
  - data placement and thread scheduling
  - black box approach not always optimal
  - algorithms
- task scheduling
  - abstract resources and utilize them efficiently

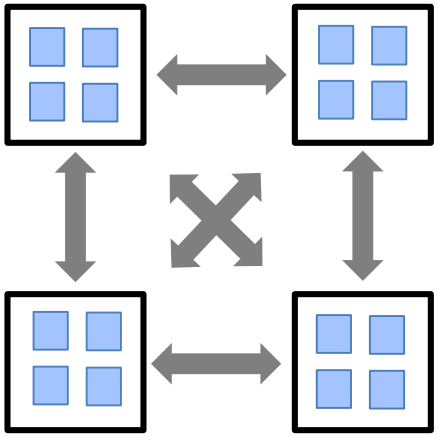
…to scale up OLAP

utilization



exploiting core's resources
minimizing memory stalls

scalability



scaling up OLTP
scaling up OLAP
conclusions

http://tinyurl.com/tutorial-2015-feedback

concluding remarks

exploiting hardware requires

- ***utilizing the resources of a core***
- ***taking advantage of parallelism***
- ***optimally managing the memory***

art of scheduling

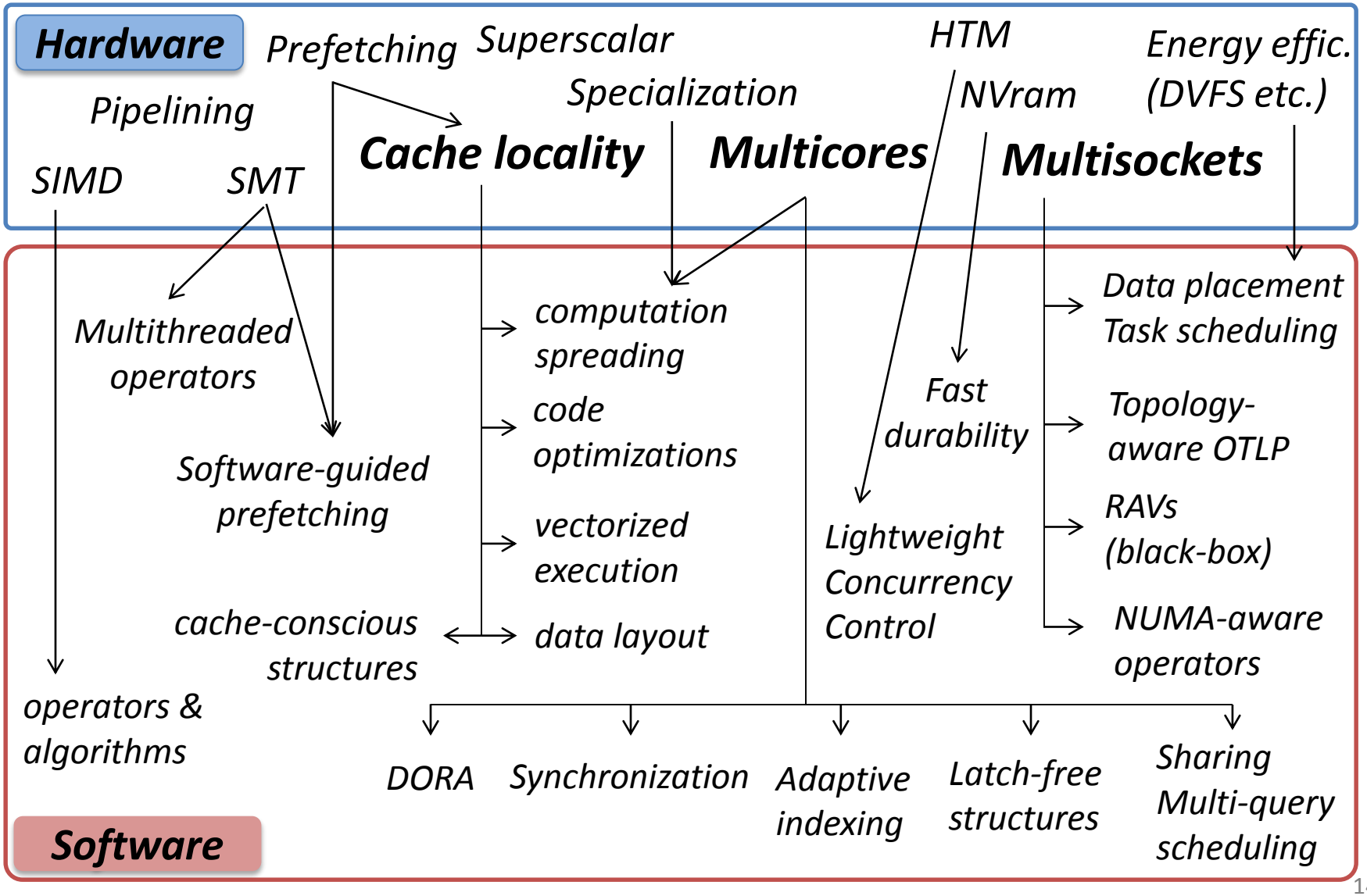
- ***adjust your task granularity***
- ***optimize locality at the right level***
- ***avoid saturation***

road to scalability

- ***eliminate all unbounded communication***

bridge the gap between software & hardware

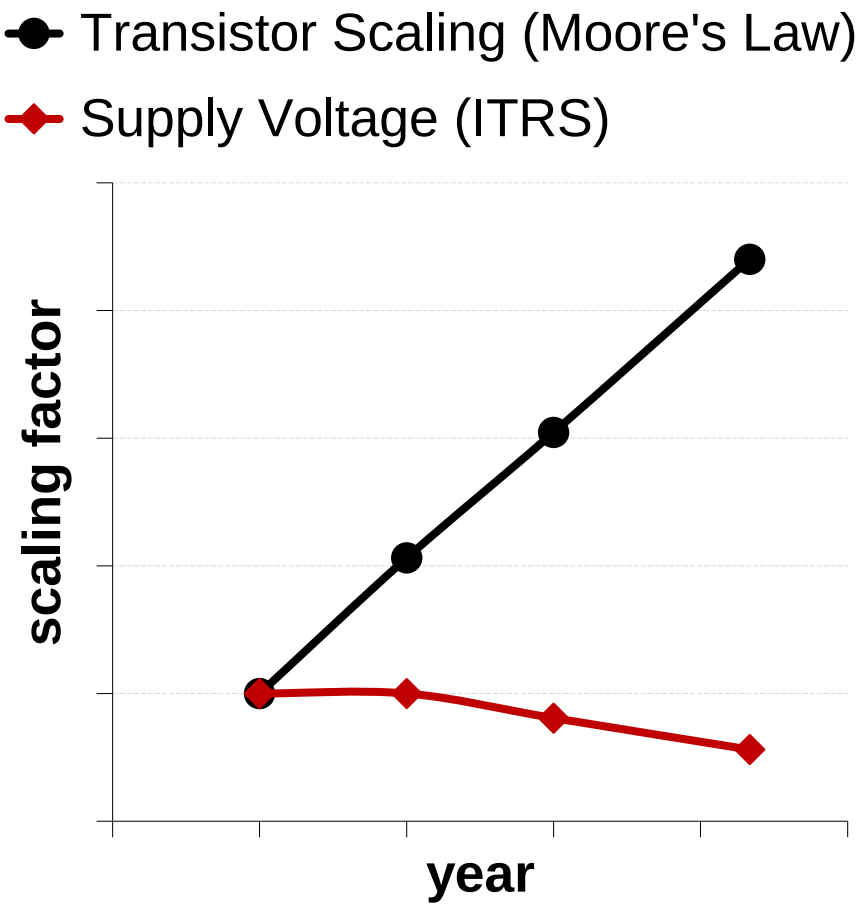
Map of the jungle

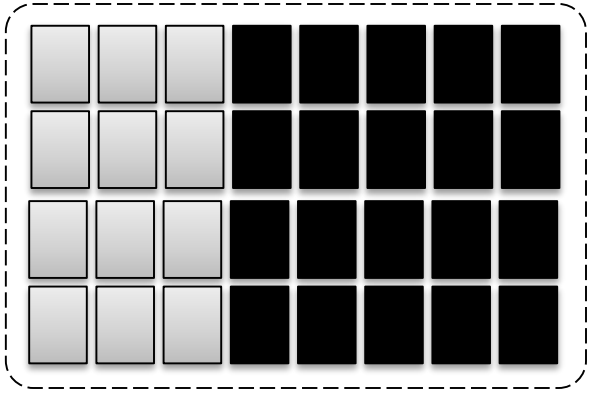


winter is coming...

graph courtesy of Hardavellas et al.

[MICRO11b, USENIX12]

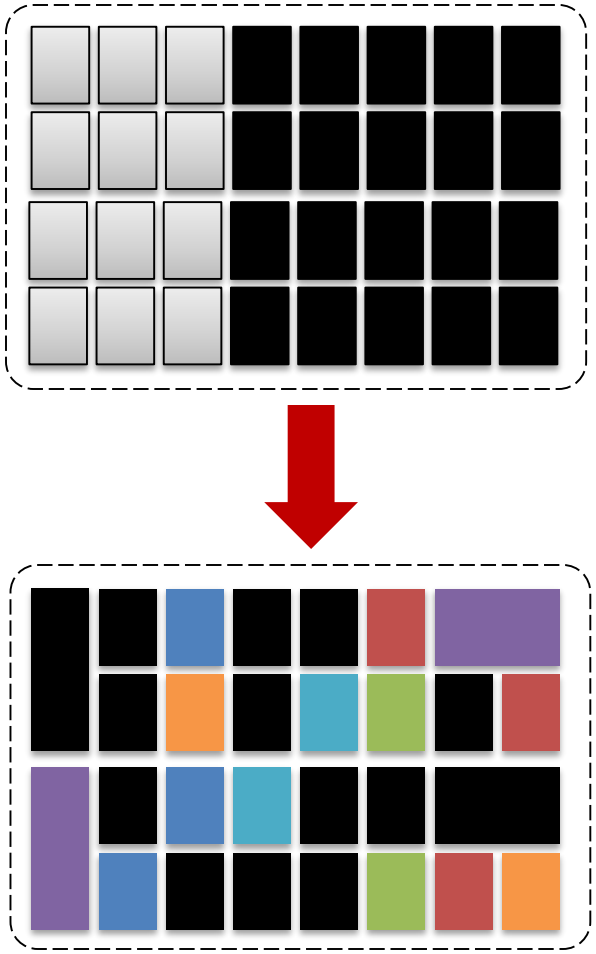




exponential increase in unusable area on chips

age of dark silicon is upon us!

exploiting dark silicon



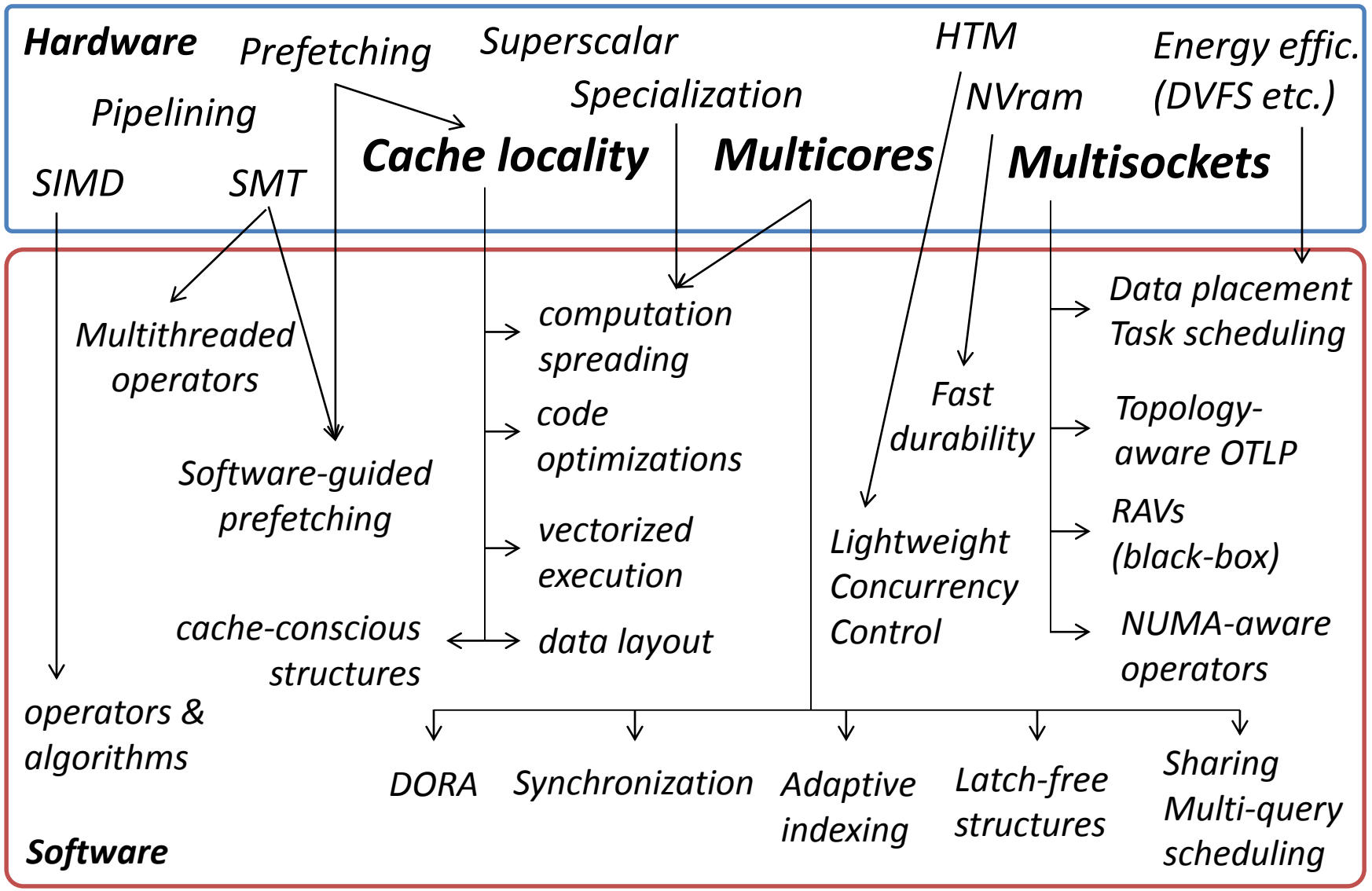
- Meet the walkers [MICRO13b]
- Database processing unit [ASPLOS14]
- Programmable accelerators [VLDB09d]
- Bionic databases [CIDR13a]
- Reconfigurable datacenters [ISCA14]
- Commercial: RAPID [ORACLE]

toward specialized hardware

open questions – How to …

[PVLDB14c, PVLDB14d, PVLDB15e]

- fit NVRAM to memory hierarchy?
- exploit HTM?
- adapt the whole software stack (OS + applications) to hardware specialization?
- take advantage of compilers?
- design concurrency-control for many-cores?



http://tinyurl.com/tutorial-2015-feedback

references

[ADMS12] H. Kimura, G. Graefe, and H. Kuno: Efficient Locking Techniques for Databases on Modern Hardware.

[ADMS13] I. Psaroudakis, T. Scheuer, N. May, and A. Ailamaki: Task Scheduling for Highly Concurrent Analytical and Transactional Main-Memory Workloads.

[ADMS14] T. Kissinger, T. Kiefer, B. Schlegel, D. Habich, D. Molka, W. Lehner: ERIS: A NUMA-Aware In-Memory Storage Engine for Analytical Workloads.

[ASPLOS06] K. Chakraborty, P. M. Wells, and G. S. Sohi: Computation spreading: employing hardware migration to specialize cmp cores on-the-fly.

[ASPLOS12] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi: Clearing the Clouds: A Study of Emerging Scale-out Workloads on Modern Hardware.

[ASPLOS13] M. Dashti, A. Fedorova, J. Funston, F. Gaud, R. Lachaize, B. Lepers, V. Quema, and M. Roth: Traffic management: A holistic approach to memory placement on numa systems.

[ASPLOS14] L. Wu, A. Lottarini, T. K. Paine, M. A. Kim, and K. A. Ross: Q100: The Architecture and Design of a Database Processing Unit.

[BTW13] T. Kiefer, B. Schlegel, and W. Lehner: Experimental Evaluation of NUMA Effects on Database Management Systems

references

[CIDR05] P. A. Boncz, M. Zukowski, and N. Nes: MonetDB/X100: Hyper-Pipelining Query Execution.

[CIDR13a] R. Johnson and I. Pandis: The bionic DBMS is coming, but what will it look like?

[CIDR13b] Y. Li, I. Pandis, R. Mueller, V. Raman, and G. Lohman: NUMA-aware Algorithms: The Case of Data Shuffling.

[CIDR13c] H. Mühe, A. Kemper, and T. Neumann: Executing Long-Running Transactions in Synchronization-Free Main Memory Database Systems.

[DaMoN13] P. Tözün, B. Gold, and A. Ailamaki: OLTP in Wonderland -- Where do cache misses come from in major OLTP components?

[DaMoN14a] H. Pirk, E. Petraki, S. Idreos, S. Manegold, and M.L. Kersten: Database Cracking: Fancy Scan, not Poor Man's Sort!

[DaMoN14b] I. Psaroudakis, T. Kissinger, D. Porobic, T. Ilsche, E. Liarou, P. Tözün, A. Ailamaki, and W. Lehner: Dynamic Fine-Grained Scheduling for Energy-Efficient Main-Memory Queries

[DSAA14] J. Wust, M. Grund, K. Hoewelmeyer, D. Schwalb, and H. Plattner: Concurrent Execution of Mixed Enterprise Workloads on In-Memory Databases.

[EDBT13] P. Tözün, I. Pandis, C. Kaynak, D. Jevdjic, and A. Ailamaki: From A to E: Analyzing TPC's OLTP Benchmarks – The obsolete, the ubiquitous, the unexplored.

references

[Eurosys12] Y. Mao, E. Kohler, and R. Morris: Cache Craftiness for Fast Multicore Key-Value Storage.

[Eurosys14] Z. Wang, H. Qian, J. Li, and H. Chen: Using Restricted Transactional Memory to Build a Scalable In-Memory Database.

[HPCA13] L. Tang, J. Mars, X. Zhang, R. Hagmann, R. Hundt, and E. Tune: Optimizing Google's warehouse scale computers: The NUMA experience.

[ICDE10] K. Krikellas, S. D. Viglas, M. Cintra: Generating code for holistic query evaluation.

[ICDE11] A. Kemper and T. Neumann: HyPer – a hybrid OLTP&OLAP main memory database system based on virtual memory snapshots.

[ICDE13a] J. Dees and P. Sanders: Efficient many-core query execution in main memory column-stores

[ICDE13b] J. Lee, Y. Kwon, F. Farber, M. Muehle, C. Lee, C. Bensberg, J. Lee, A. Lee, and W. Lehner: SAP HANA Distributed In-Memory Database System: Transaction, Session and Metadata Management

[ICDE13c] J. Levandoski, D. Lomet, and S. Sengupta: The Bw-Tree: A B-tree for new hardware platforms.

[ICDE14a] H. Han, S. Park, H. Jung, A. Fekete, U. Roehm, and H. Yeom : Scalable Serializable Snapshot Isolation for Multicore Systems.

references

[ICDE14b] V. Leis, A. Kemper, and T. Neumann: Exploiting Hardware Transactional Memory in Main-Memory Databases.

[ICDE14c] N. Malviya, A. Weisberg, S. Madden, and M. Stonebraker: Rethinking Main Memory OLTP Recovery.

[ICDE14d] D. Porobic, E. Liarou, P. Tözün, and A. Ailamaki: ATraPos: Adaptive Transaction Processing on Hardware Islands.

[IMDM13] S. Wolf, H. Mühe, A. Kemper, and T. Neumann: An evaluation of strict timestamp ordering concurrency control for main-memory database systems.

[ISCA90] N. P. Jouppi: Improving Direct-mapped Cache Performance by the Addition of a Small Fully-associative Cache and Prefetch Buffers.

[ISCA01] A. Ramirez, L. A. Barroso, K. Gharachorloo, R. Cohn, J. Larriba-Pey, P. G. Lowney, and M. Valero: Code Layout Optimizations for Transaction Processing Workloads.

[ISCA05] T. F. Wenisch, S. Somogyi, N. Hardavellas, J. Kim, A. Ailamaki, and B. Falsafi: Temporal Streaming of Shared Memory.

[ISCA14] A. Putnam, A. Caulfield, E. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, and D. Burger: A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services.

references

[MICRO00] T. Sherwood, S. Sair, and B. Calder: Predictor-directed Stream Buffers.

[MICRO11a] M. Ferdman, C. Kaynak, and B. Falsafi: Proactive Instruction Fetch.

[MICRO11b] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki: Toward Dark Silicon in Servers.

[MICRO12] I. Atta, P. Tözün, A. Ailamaki, and A. Moshovos: SLICC: Self-Assembly of Instruction Cache Collectives for OLTP Workloads.

[MICRO13a] C. Kaynak, B. Grot, and B. Falsafi: SHIFT: Shared History Instruction Fetch for Lean-Core Server Processors.

[MICRO13b] O. Kocberber, B. Grot, J. Picorel, B. Falsafi, K. Lim, and P. Ranganathan: Meet the Walkers: Accelerating Index Traversals for In-memory Databases.

[ORACLE] https://labs.oracle.com/pls/apex/f?p=labs:49:::::P49_PROJECT_ID:14

[PCS13] B. Vikranth, R. Wankar, and C. Rao: Topology Aware Task Stealing for On-chip NUMA Multi-core Processors.

[PVLDB10a] R. Johnson, I. Pandis, R. Stoica, M. Athanassoulis, and A. Ailamaki: Aether: A Scalable Approach to Logging.

[PVLDB10b] I. Pandis, R. Johnson, N. Hardavellas, and A. Ailamaki: Data-Oriented Transaction Execution.

[PVLDB11a] T. Neumann: Efficiently compiling efficient query plans for modern hardware.

references

[PVLDB11b] I. Pandis, P. Tözün, R. Johnson, and A. Ailamaki: PLP: page latch-free shared-everything OLTP.

[PVLDB11c] J. Sewall, J. Chhugani, C. Kim, N. Satish, and P. Dubey: PALM: Parallel architecture-friendly latch-free modifications to B+ trees on many-core processors.

[PVLDB12a] M.-C. Albutiu, A. Kemper, and T. Neumann: Massively Parallel Sort-merge Joins in Main Memory Multi-core Database Systems.

[PVLDB12b] G. Giannikis, G. Alonso, and D. Kossmann: SharedDB: Killing One Thousand Queries with One Stone.

[PVLDB12c] P. Larson, S. Blanas, C. Diaconu, C. Freedman, J. M. Patel, and M. Zwilling: High-performance concurrency control mechanisms for main-memory databases.

[PVLDB12d] D. Porobic, I. Pandis, M. Branco, P. Tözün, and A. Ailamaki: OLTP on Hardware Islands.

[PVLDB13a] J. Levandoski, D. Lomet, and S. Sengupta: LLAMA: A Cache/Storage Subsystem for Modern Hardware

[PVLDB13b] I. Psaroudakis, M. Athanassoulis, and A. Ailamaki: Sharing Data and Work Across Concurrent Analytical Queries.

[PVLDB13c] K. Ren, A. Thomson, and D. J. Abadi: Lightweight locking for main memory database systems.

references

[PVLDB14a] C. Balkesen, G. Alonso, J. Teubner, and M. T. Ozsu: Multi-Core, Main-Memory Joins: Sort vs. Hash Revisited.

[PVLDB14b] G. Giannikis, D. Makreshanski, G. Alonso, and D. Kossmann: Shared Workload Optimization.

[PVLDB14c] M. Karpathiotakis, M. Branco, I. Alagiannis, and A. Ailamaki: Adaptive Query Processing on RAW Data.

[PVLDB14d] Y. Klonatos, C. Koch, T. Rompf, and H. Chafi: Building Efficient Query Engines in a High-Level Language.

[PVLDB14e] S. Pelley, T. Wenisch, B. Gold, and B. Bridge: Storage Management in the NVRAM Era.

[PVLDB14f] P. Tözün, I. Atta, A. Ailamaki, A. Moshovos: ADDICT: Advanced Instruction Chasing for Transactions.

[PVLDB14g] T. Wang and R. Johnson: Scalable Logging through Emerging Non-Volatile Memory.

[PVLDB15a] A. Chatzistergiou, M. Cintra, S. Viglas: REWIND: Recovery Write-Ahead System for In-Memory Non-Volatile Data-Structures

[PVLDB15b] S. Chen, Q. Jin: Persistent B+Trees in NonVolatile Main Memory

[PVLDB15c] J. Giceva, G. Alonso, T. Roscoe, T. Harris: Deployment of Query Plans on Multicores.

[PVLDB15d] J. Huang, K. Schwan. M. Qureshi: NVRAMaware Logging in Transaction Systems

references

[PVLDB15e] X. Yu, G. Bezerra, A. Pavlo, S. Devadas, and M. Stonebraker: Staring into the Abyss: An Evaluation of Concurrency Control with One Thousand Cores.

[SIGMOD85] G. P. Copeland and S. N. Khoshafian: A Decomposition Storage Model.

[SIGMOD02a] S. Chen, P. B. Gibbons, T. C. Mowry, and G. Valentin: Fractal prefetching B+-Trees: optimizing both cache and disk performance.

[SIGMOD02b] J. Zhou and K. Ross: Implementing Database Operations Using SIMD Instructions.

[SIGMOD05] S. Harizopoulos, V. Shkapenyuk, and A. Ailamaki: QPipe: a simultaneously pipelined relational query engine.

[SIGMOD10a] S. Arumugam, A. Dobra, C. Jermaine, N. Pansare, and L. Perez: The DataPath system: a data-centric analytic processing engine for large data warehouses.

[SIGMOD10b] E. P. Jones, D. J. Abadi, and S. Madden: Low overhead concurrency control for partitioned main memory databases.

[SIGMOD12] A. Thomson, T. Diamond, S.-C. Weng, K. Ren, P. Shao, and D. J. Abadi: Calvin: Fast distributed transactions for partitioned database systems.

[SIGMOD13a] C. Diaconu, C. Freedman, E. Ismert, P. Larson, P. Mittal, R. Stonecipher, N. Verma, and M. Zwilling: Hekaton: SQL Server's memory-optimized OLTP engine.

[SIGMOD13b] G. Graefe, M. Lilibridge, H. Kuno, J. Tucek, and A. Veitch: Controlled Lock Violation.

references

[SIGMOD14a] V. Leis, P. Boncz, A. Kemper, T. Neumann: Morsel-Driven Parallelism: A NUMA-aware Query Evaluation Framework for the Many-Core Age.

[SIGMOD14b] I. Psaroudakis, M. Athanassoulis, M. Olma, and A. Ailamaki: Reactive and Proactive Sharing Across Concurrent Analytical Queries.

[SOSP13] S. Tu, W. Zheng, E. Kohler, B. Liskov, and S. Madden: Speedy transactions in multicore in-memory databases.

[TOCS03] M. Annavaram, J. M. Patel, E. S. Davidson: Call graph prefetching for database applications.

[USENIX11] S. Blagodurov, S. Zhuravlev, M. Dashti, and A. Fedorova: A Case for NUMA-aware Contention Management on Multicore Systems.

[USENIX12] N. Hardavellas: The Rise and Fall of Dark Silicon.

[VLDB99] A. Ailamaki, D. DeWitt, M. Hill, and D. Wood: DBMSs On A Modern Processor: Where Does Time Go?

[VLDB05a] M. Stonebraker, D. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, P. O'Neil, A. Rasin, N. Tran, and S. Zdonik: C-Store: A Column Oriented DBMS.

[VLDB05b] J. Zhou, J. Cieslewicz, K. Ross, and M. Shah: Improving Database Performance on Simultaneous Multithreading Processors.

references

[VLDB06] A. Ghoting, G. Buehrer, S. Parthasarathy, D. Kim, A. Nguyen, Y.-K. Chen, and P. Dubey: Cache-conscious frequent pattern mining on modern and emerging processors.

[VLDB07a] R. Johnson, S. Harizopoulos, N. Hardavellas, K. Sabirli, I. Pandis, A. Ailamaki, N. G. Mancheril, and B. Falsafi: To share or not to share?

[VLDB07b] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland: The end of an architectural era: (it's time for a complete rewrite).

[VLDB08a] J. Chhugani, A. Nguyen, V. Lee, W. Macy, M. Hagog, Y. Chen, A. Baransi, S. Kumar, and P. Dubey: Efficient implementation of sorting on multi-core SIMD CPU architecture.

[VLDB08b] Lin Qiao, Vijayshankar Raman, Frederick Reiss, Peter J. Haas, and Guy M. Lohman. Main-Memory Scan Sharing For Multi-Core CPUs

[VLDB09a] G. Candea, N. Polyzotis, and R. Vingralek: A scalable, predictable join operator for highly concurrent data warehouses.

[VLDB09b] R. Johnson, I. Pandis, and A. Ailamaki: Improving OLTP Scalability Using Speculative Lock Inheritance.

[VLDB09c] C. Kim, E. Sedlar, J. Chhugani, T. Kaldewey, A. D. Nguyen, A. D. Blas, V. W. Lee, N. Satish, and P. Dubey: Sort vs. Hash Revisited: Fast Join Implementation on Modern Multi-Core CPUs.

[VLDB09d] R. Mueller, J. Teubner, and G. Alonso: Data Processing on FPGAs.

[VLDBJ14] R. Johnson, I. Pandis, and A. Ailamaki: Eliminating unscalable communication in transaction processing.