DASYA

www.dasya.dk
@dasyaITU

RAD

https://rad.itu.dk

www.itu.dk

hardware parallelism & transaction processing systems

Pınar Tözün
Associate Professor, IT University of Copenhagen
pito@itu.dk, www.pinartozun.com, @pinartozun

transaction vs. analytical processing

OLTP	OLAP
short-running simple requests	long-running complex requests
access small portion of the data	access lots of data
fetch several columns of a record	fetch a few columns of a record
lookup, insert, delete, update	SQL queries, map-reduce jobs, machine learning, graph analytics, …
deposit money to a customer's account, lookup information about a product, looking up a tweet, …	*customers who are most likely to get mortgages next year, item sold the most last year in each department of a store grouped by months, …*

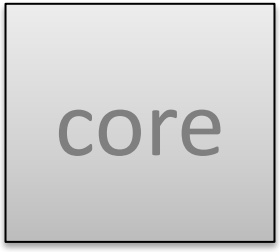
➔ **primary applications for databases**

➔ **required functionality & optimizations differ**

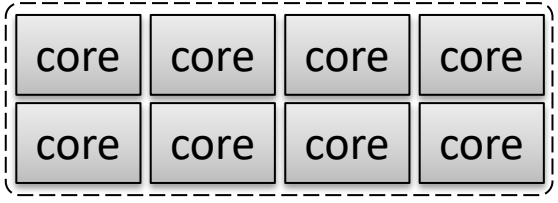
evolution of general-purpose CPU

… the hardware we run transactions on

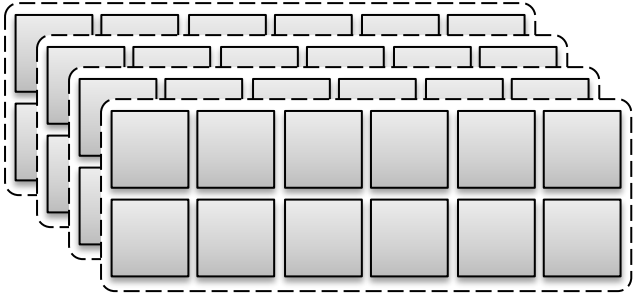
2005



single-core CPUs



multicore CPUs



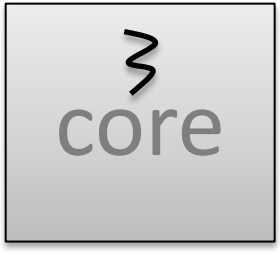
multisocket multicore CPUs

faster & more-complex cores over time

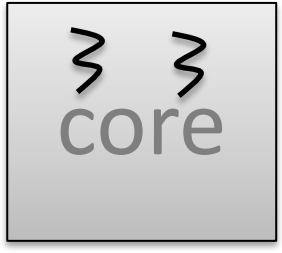
similar speed & complexity in a core, more cores over time

types of hardware parallelism

implicit/vertical parallelism



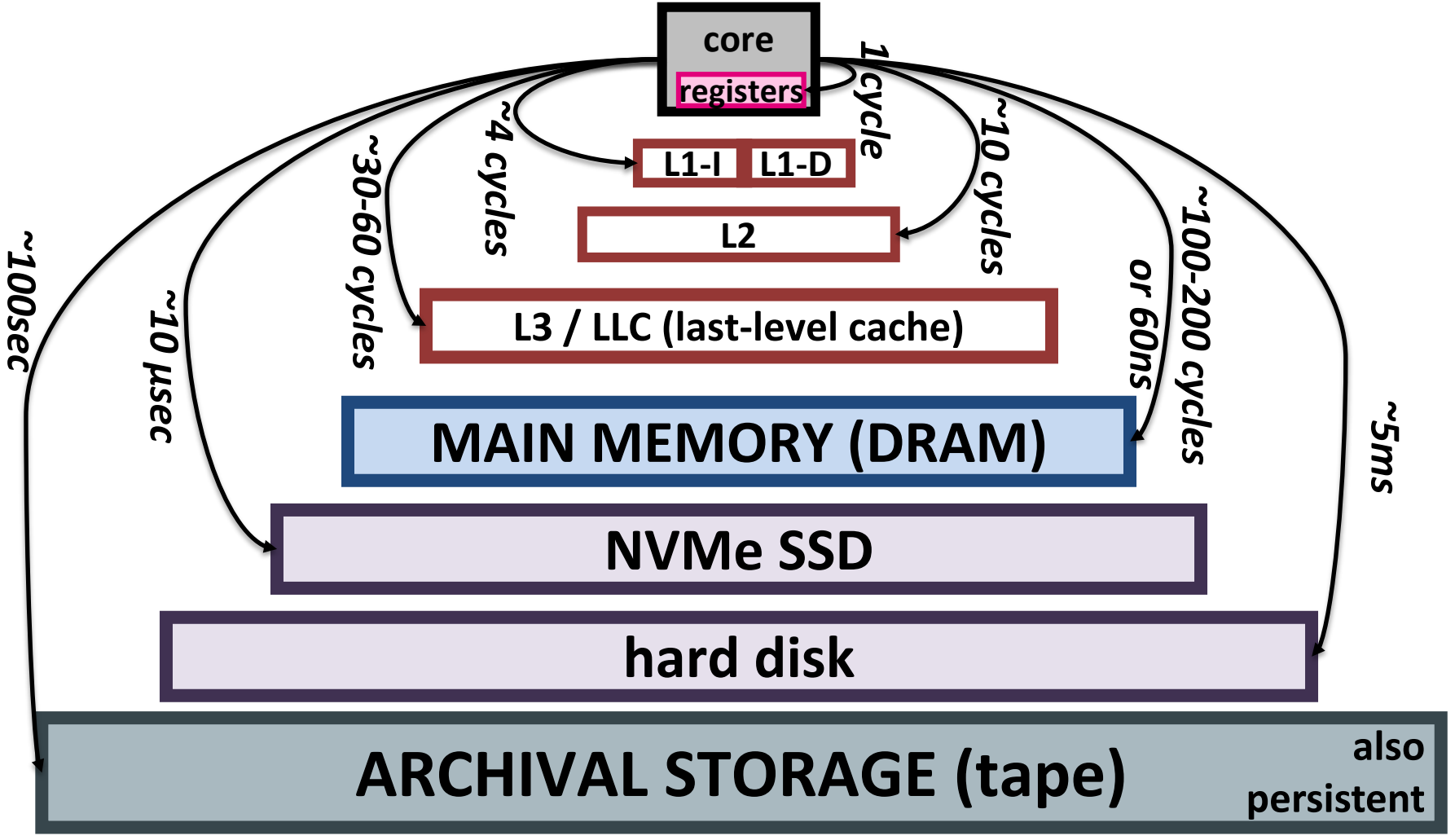
instruction & data parallelism
hardware does this automatically



multithreading
threads share
execution cycles
on the same core

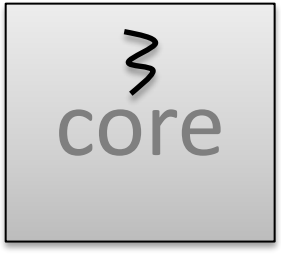
why do we need this?

single-core – access latency to storage

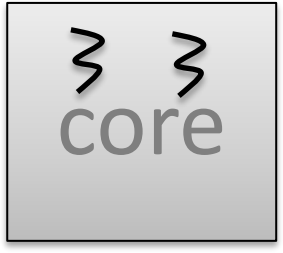


types of hardware parallelism

implicit/vertical parallelism



instruction & data parallelism
hardware does this automatically

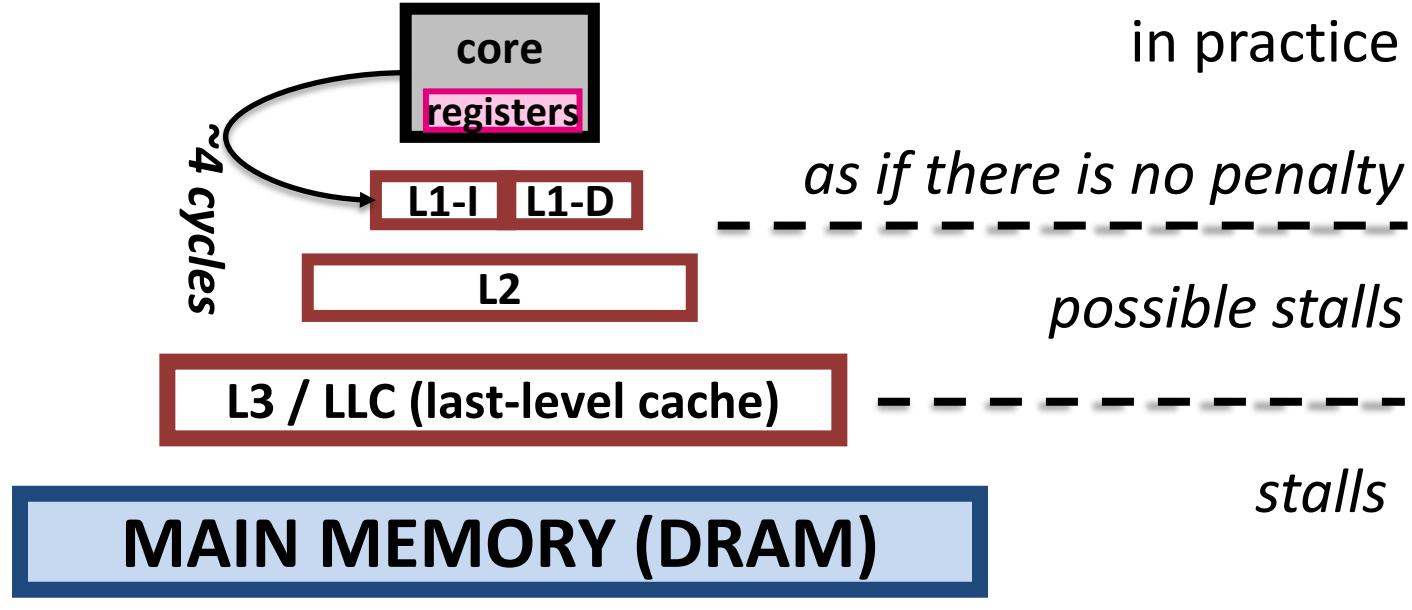


multithreading
threads share
execution cycles
on the same core

why?
we don't want cores to stay idle waiting for instruction/data accesses!

goal: minimize stall time due to cache/memory accesses
overlapping access latency for one item with other work

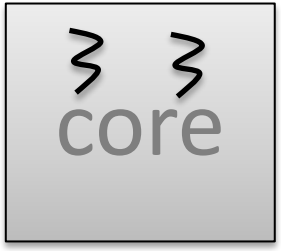
single-core – access latency to storage



stalls = 😴

types of hardware parallelism

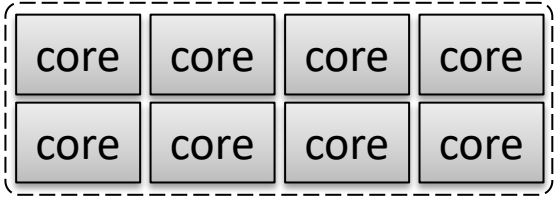
implicit/vertical parallelism ⟷ **explicit/horizontal parallelism**



single-core

instruction & data parallelism

simultaneous multithreading



multicores

multiple threads run in parallel on different cores

why do we have this?

wording courtesy of Wikipedia

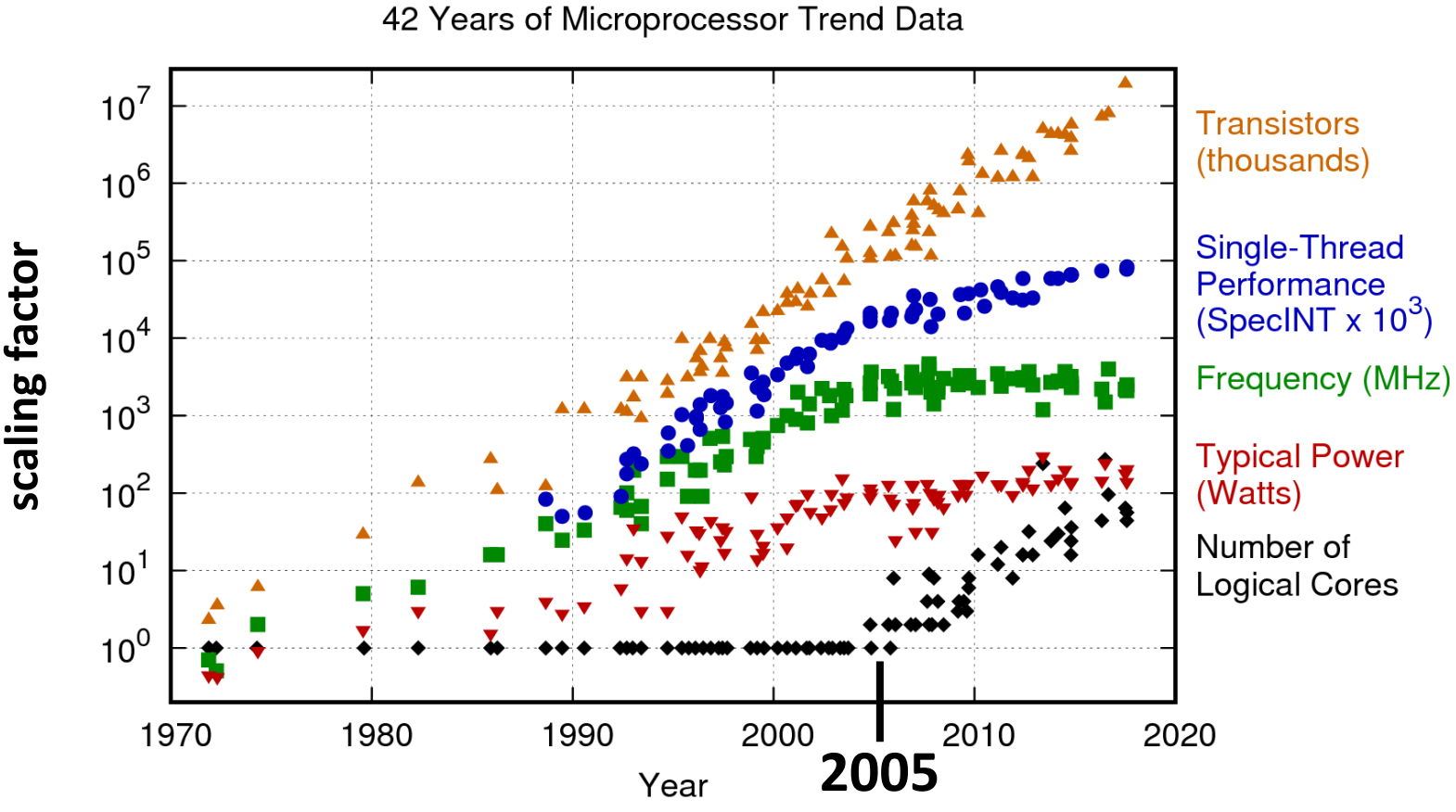
for Moore's law to be practical you need Dennard scaling!

Moore's law

"… the observation that the number of transistors in a dense integrated circuit doubles approximately every two years."

Dennard scaling

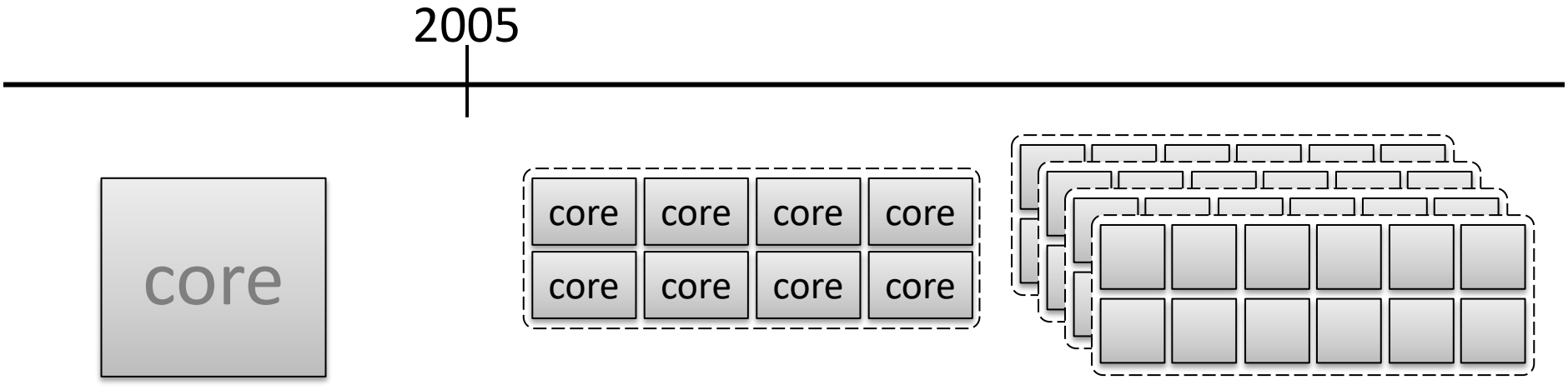
" … as transistors get smaller their power density stays constant, so that the power use stays in proportion with area: both voltage and current scale (downward) with length."



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

doubling of transistor counts continues
power and clock speeds hit the wall

commodity CPU evolution



single-core CPUs

multicore CPUs

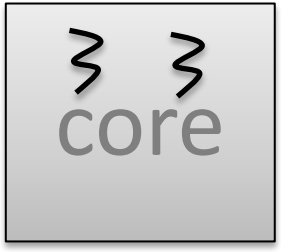
multisocket multicore CPUs

Dennard scaling doesn't hold anymore

switching to multicores kept Moore's Law alive

types of hardware parallelism

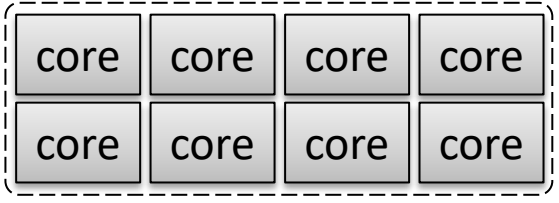
implicit/vertical parallelism ⟷ **explicit/horizontal parallelism**



single-core

instruction & data parallelism

simultaneous multithreading



multicores

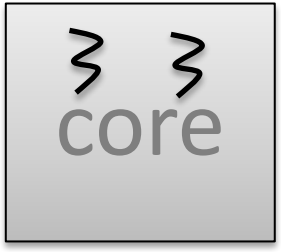
multiple threads run in parallel on different cores

implicit parallelism → (almost) free lunch

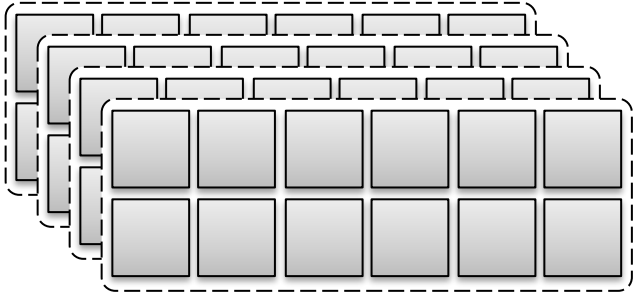
explicit parallelism → must work hard to exploit it

types of hardware parallelism

implicit/vertical parallelism ⟷ **explicit/horizontal parallelism**



single-core
instruction & data parallelism
simultaneous multithreading



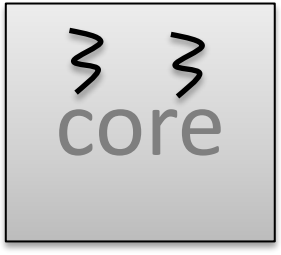
multisocket multicores
multiple processors/CPUs
in one machine

implicit parallelism → (almost) free lunch

explicit parallelism → must work hard to exploit it

types of hardware parallelism

implicit/vertical parallelism ⟷ **explicit/horizontal parallelism**



single-core
instruction & data parallelism
simultaneous multithreading

distributed systems
running a program over
multiple machines

implicit parallelism → (almost) free lunch

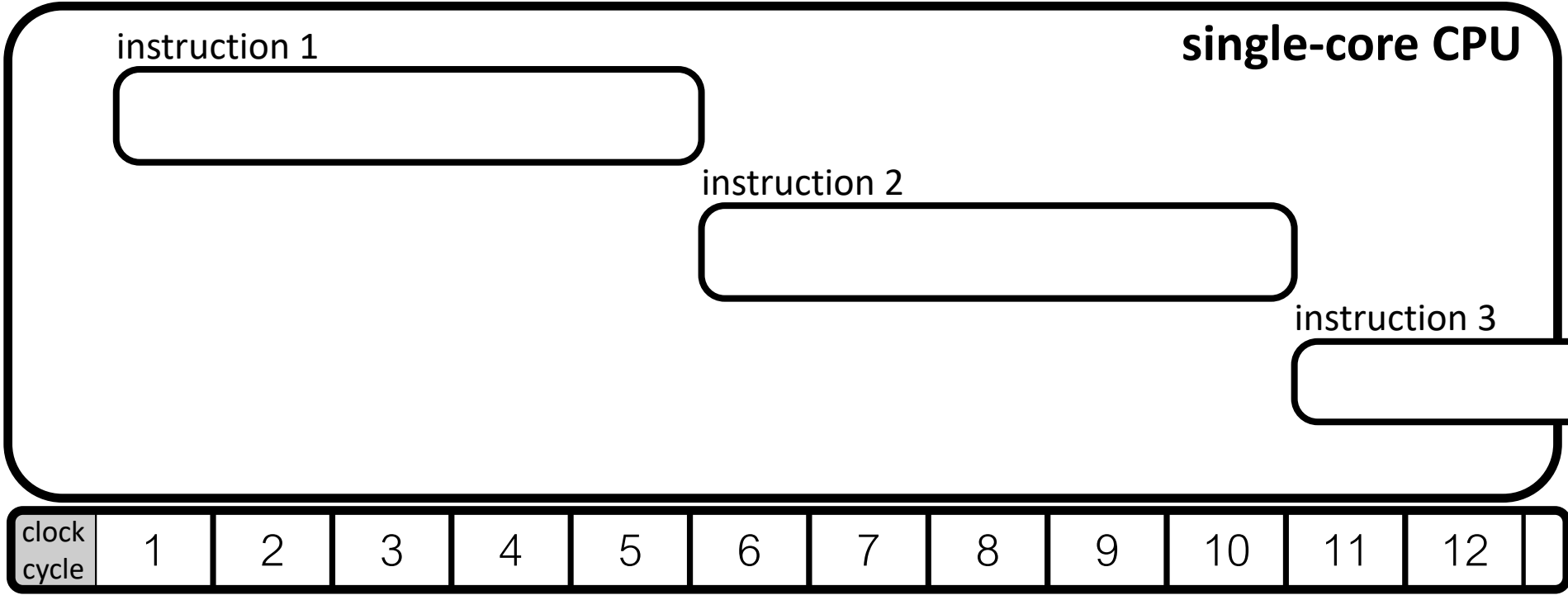
explicit parallelism → must work hard to exploit it

agenda

- types of hardware parallelism
- **implicit parallelism**
- explicit parallelism

subscalar CPU (i.e., no implicit parallelism)

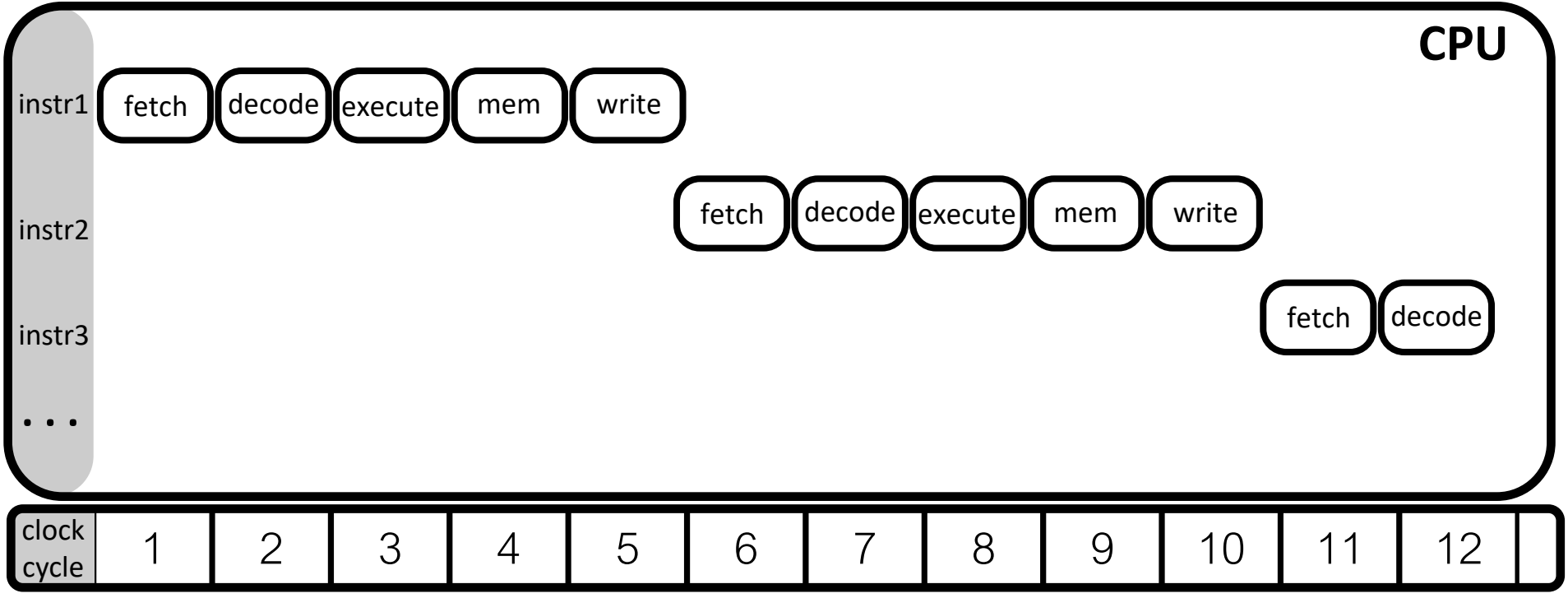
one instruction at a time



several cycles to complete two instructions, assuming no long-latency data/memory accesses

subscalar CPU (i.e., no implicit parallelism)

one instruction at a time



several cycles to complete two instructions, assuming no long-latency data/memory accesses

RISC instruction stages

fetch
the instruction from the cache

decode
specifying which operation the instruction performs and inputs it needs

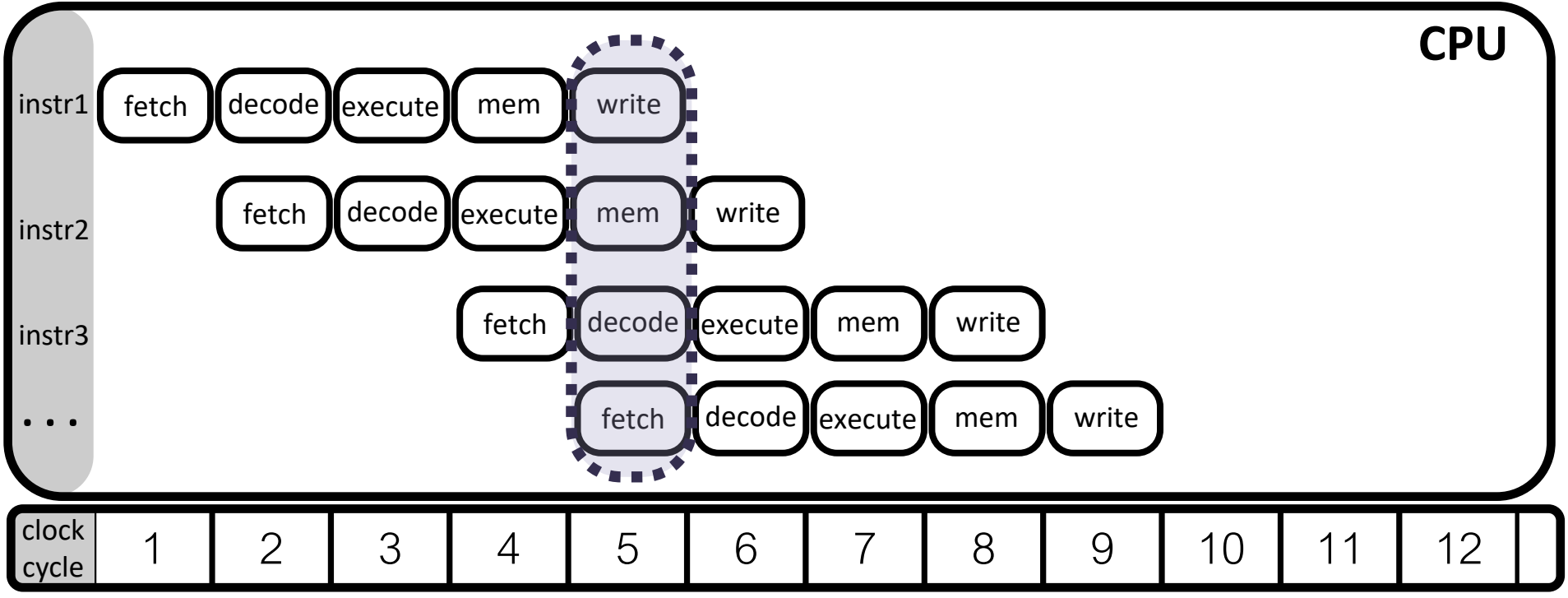
execute
the operation itself

memory
accessing the memory for inputs if needed

write
writing back the results into registers

instruction pipelining

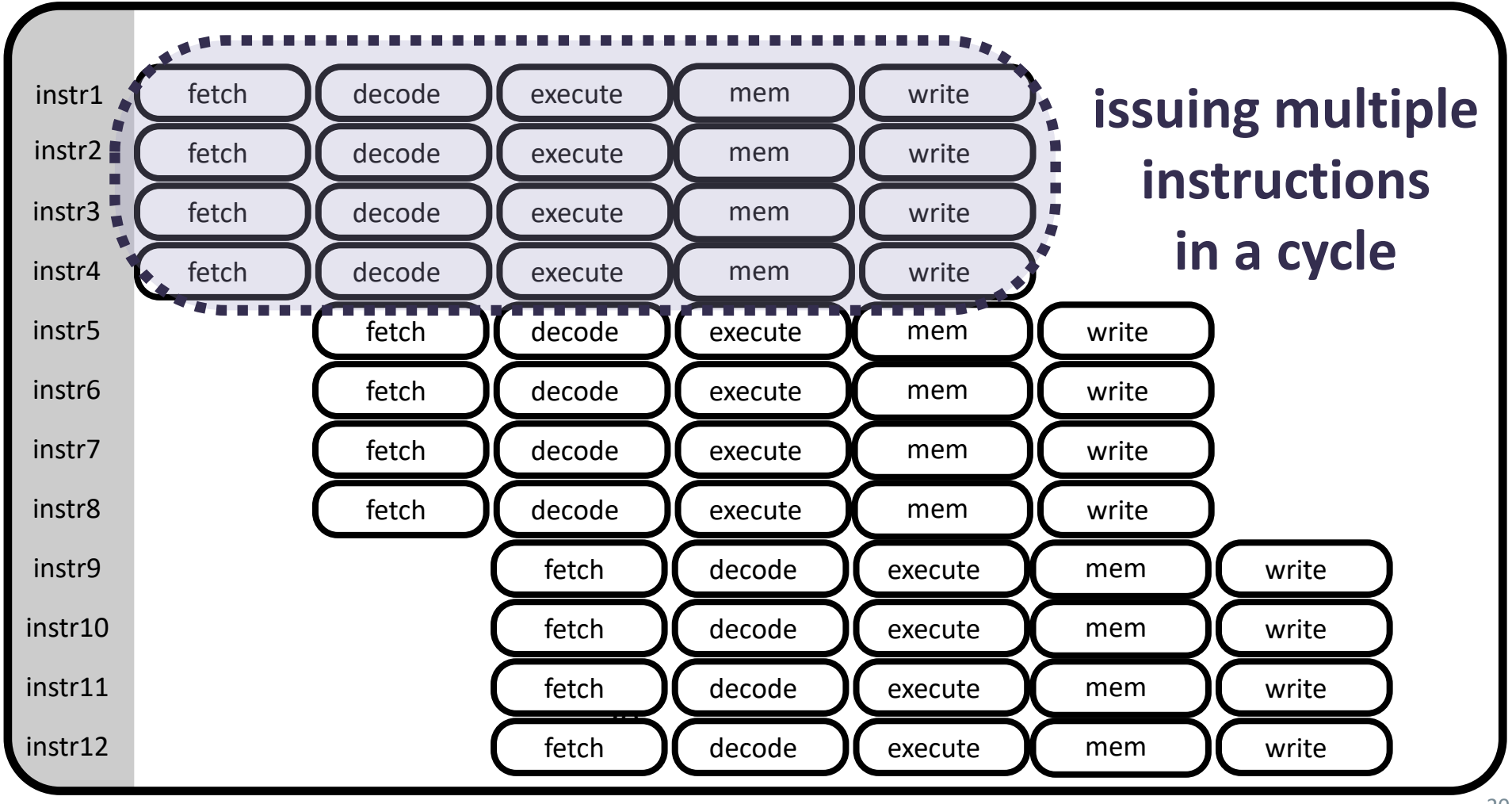
overlapping stages of different instructions



fundamental way to parallelize implicitly

superscalar CPU

example of 4-way superscalar CPU



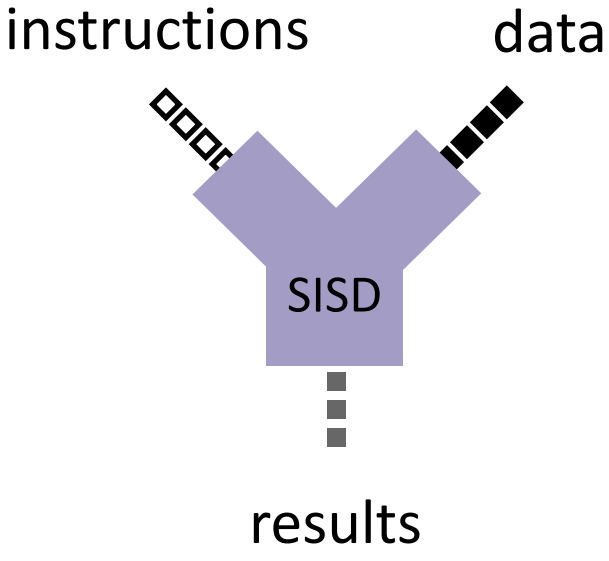
out-of-order (OoO) execution

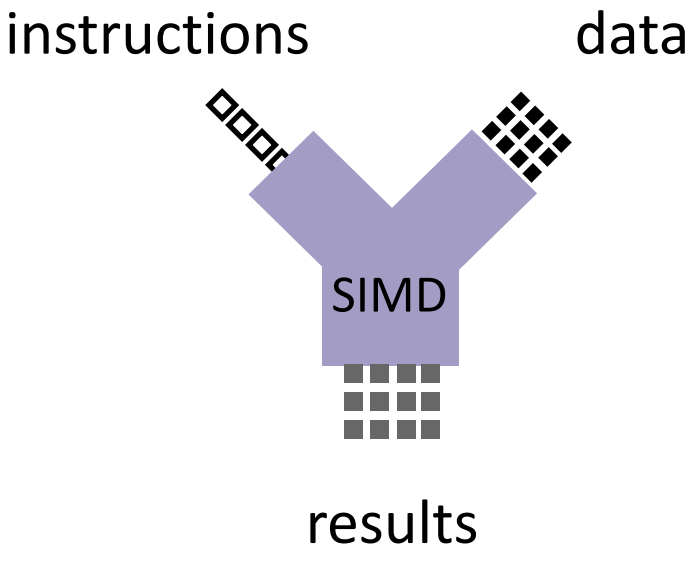
allows a processor to execute instructions based on the availability of input data rather than strictly following the instruction ordering of a program

example:

(1) r1 ← r2 / r3
(2) r4 ← r1 + r5
(3) r6 ← r7 * r8 → **independent from the previous two, can be executed independently in parallel or before 1 & 2**

single instruction multiple data (SIMD)





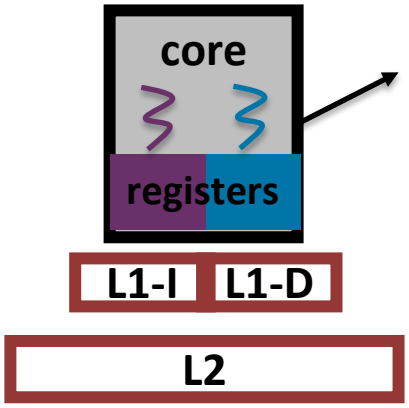
also implicit data parallelism, but this one has to be managed/issued by the software

simultaneous multithreading (SMT)

keep state/context of multiple threads via more register space

each CPU cycle, you swap the thread being executed

OS don't have to do a context switch for this

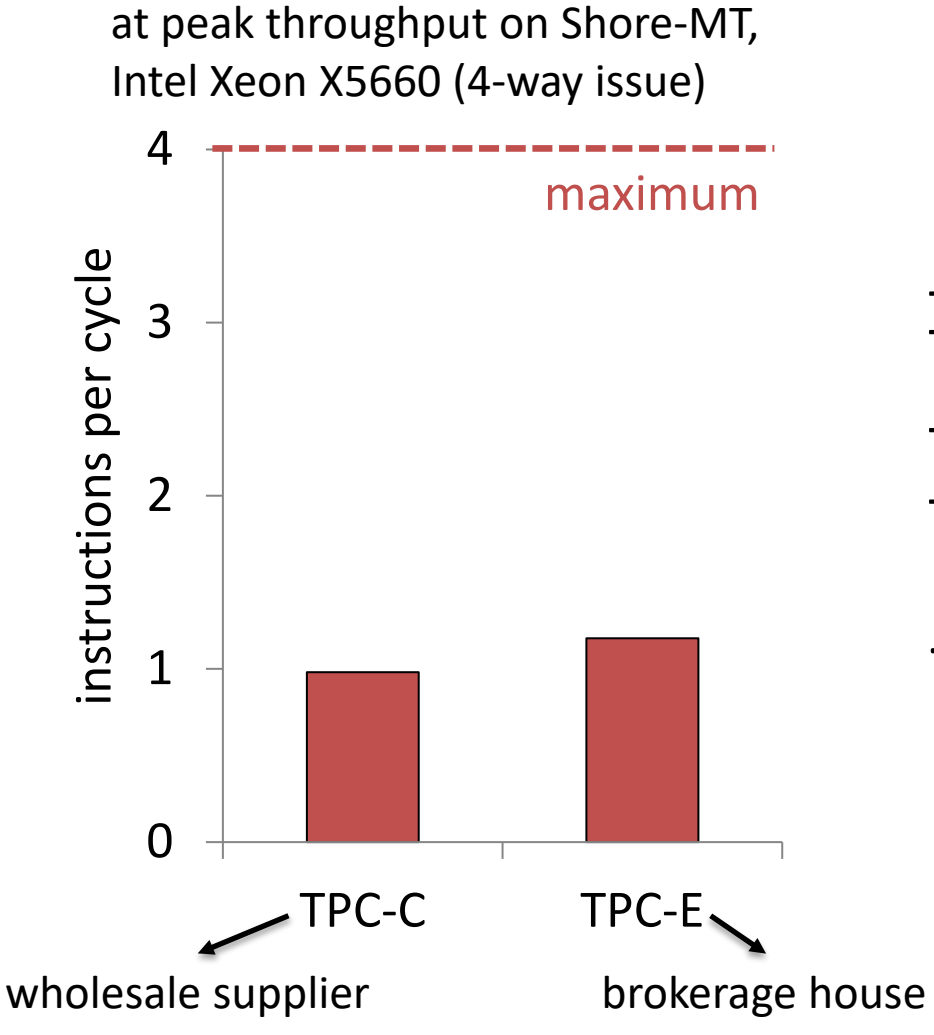


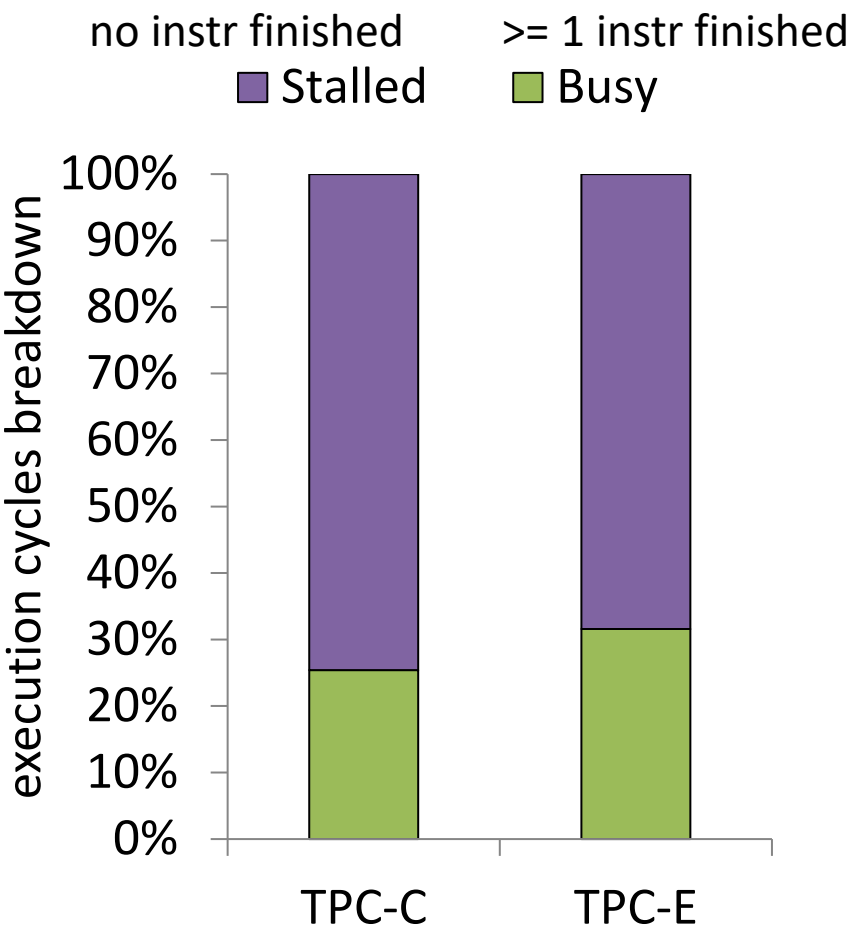
example of a core that supports two hardware contexts or logical threads

more pressure on shared hardware resources (e.g., caches)

if not used right, may not give much benefit or may even hurt performance

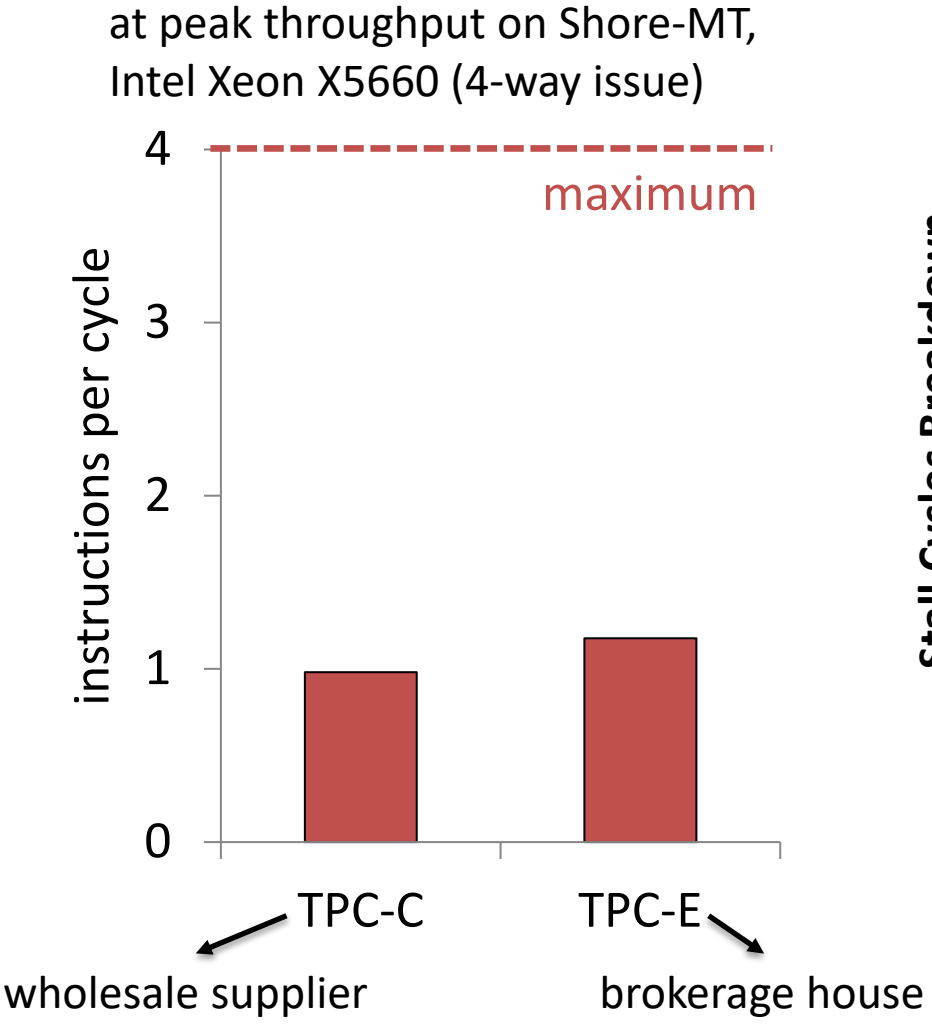
OLTP & implicit parallelism

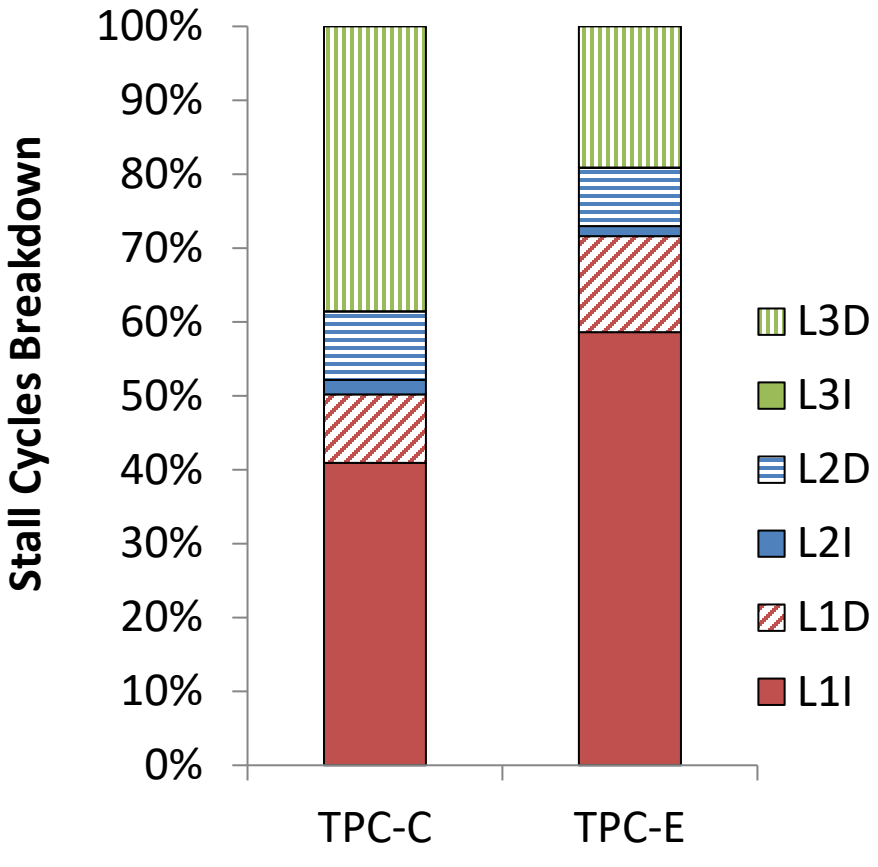




[EDBT13]

OLTP & implicit parallelism

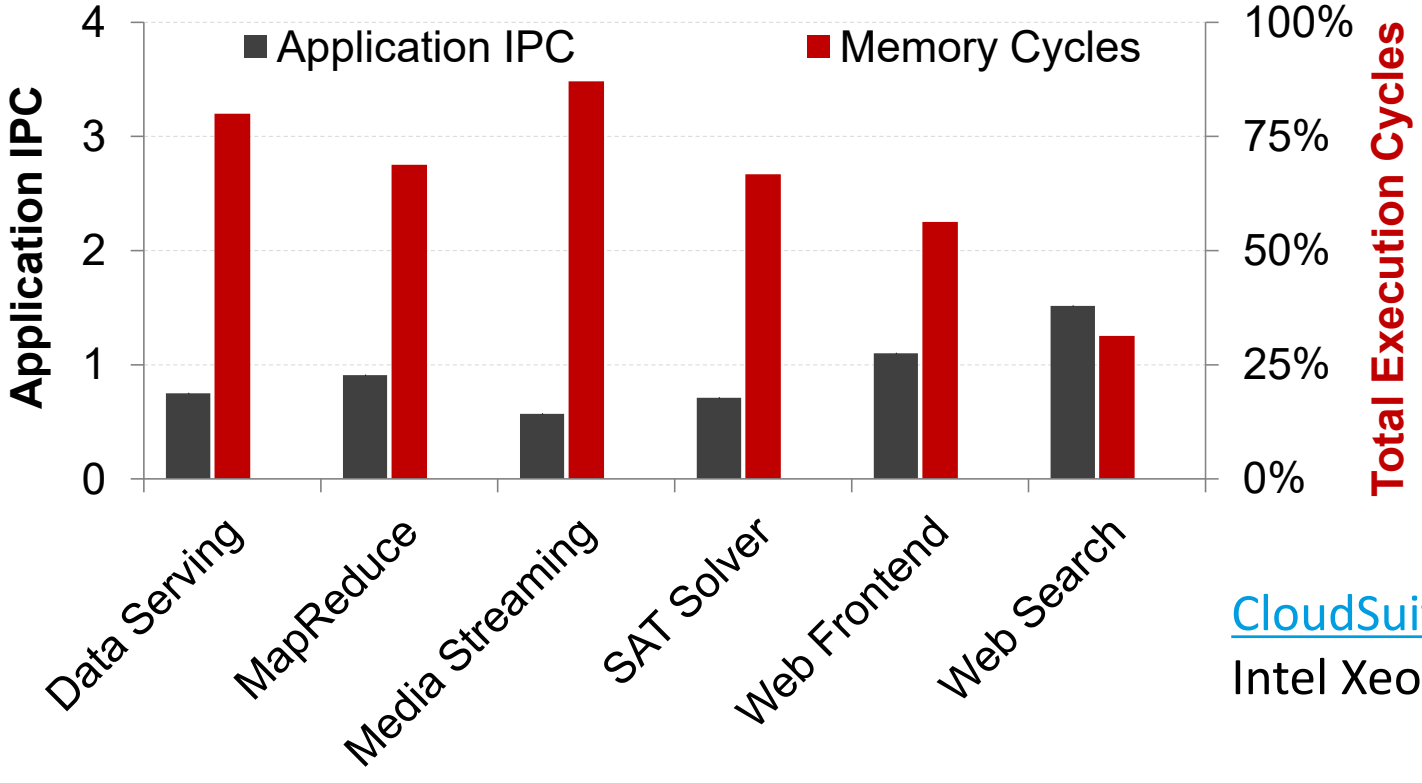




[EDBT13, DaMoN13]

memory stalls in data-intensive apps

[ASPLOS12]

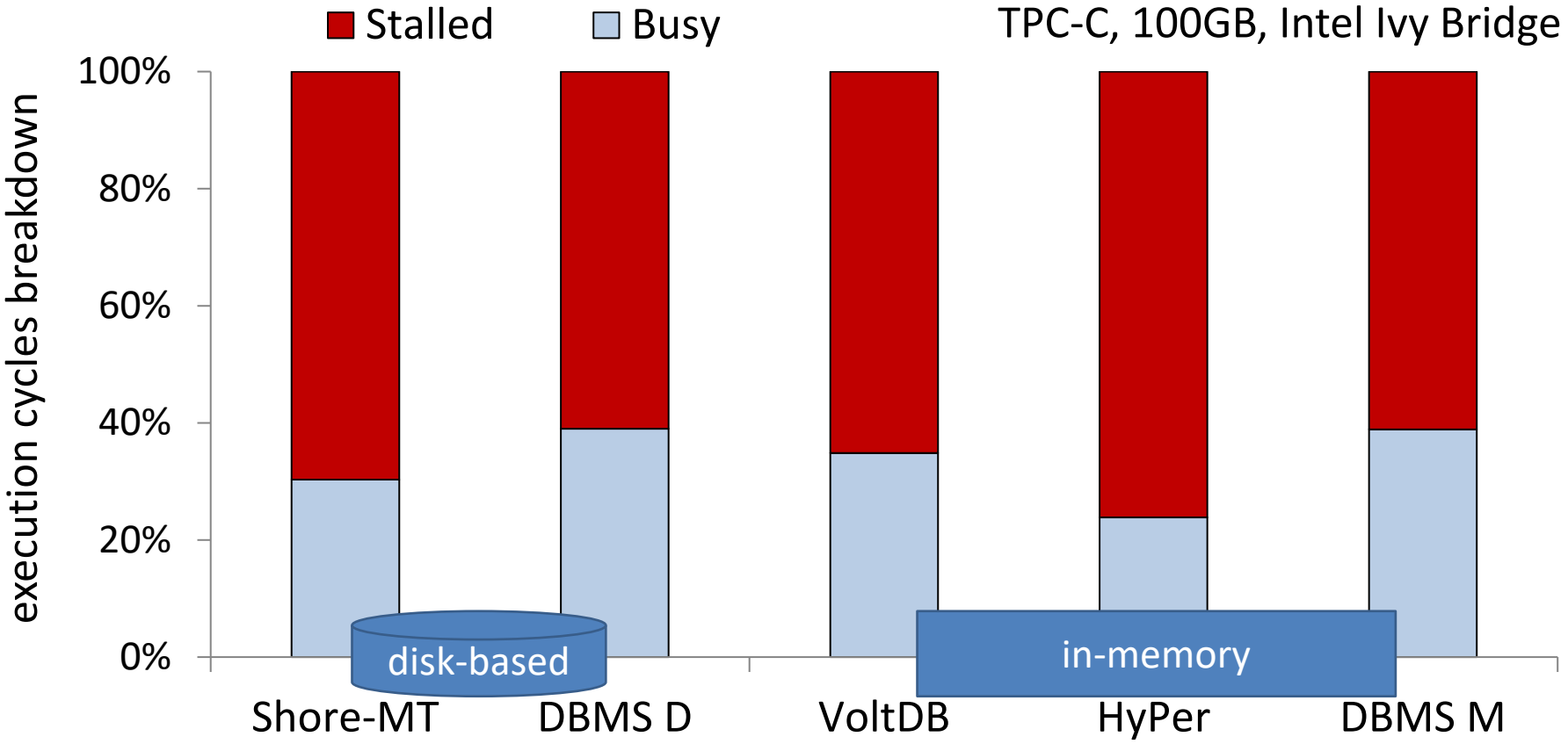


CloudSuite on Intel Xeon X5670

data-intensive apps suffer due to memory stalls not just due to data but also instructions

what about in-memory OLTP?

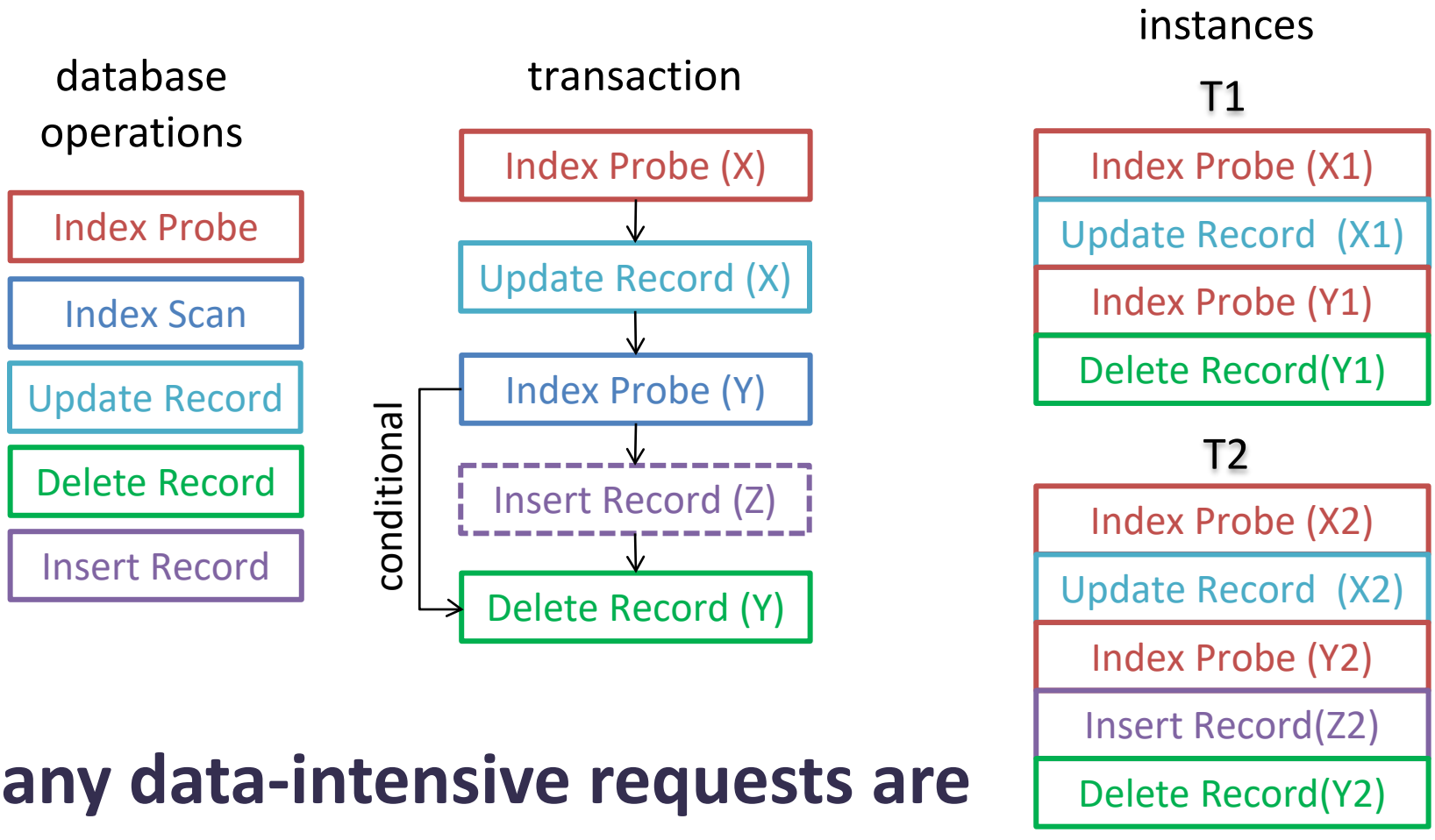
[SIGMOD16]



doesn't mean these systems are bad

it means we are leaving performance on the table

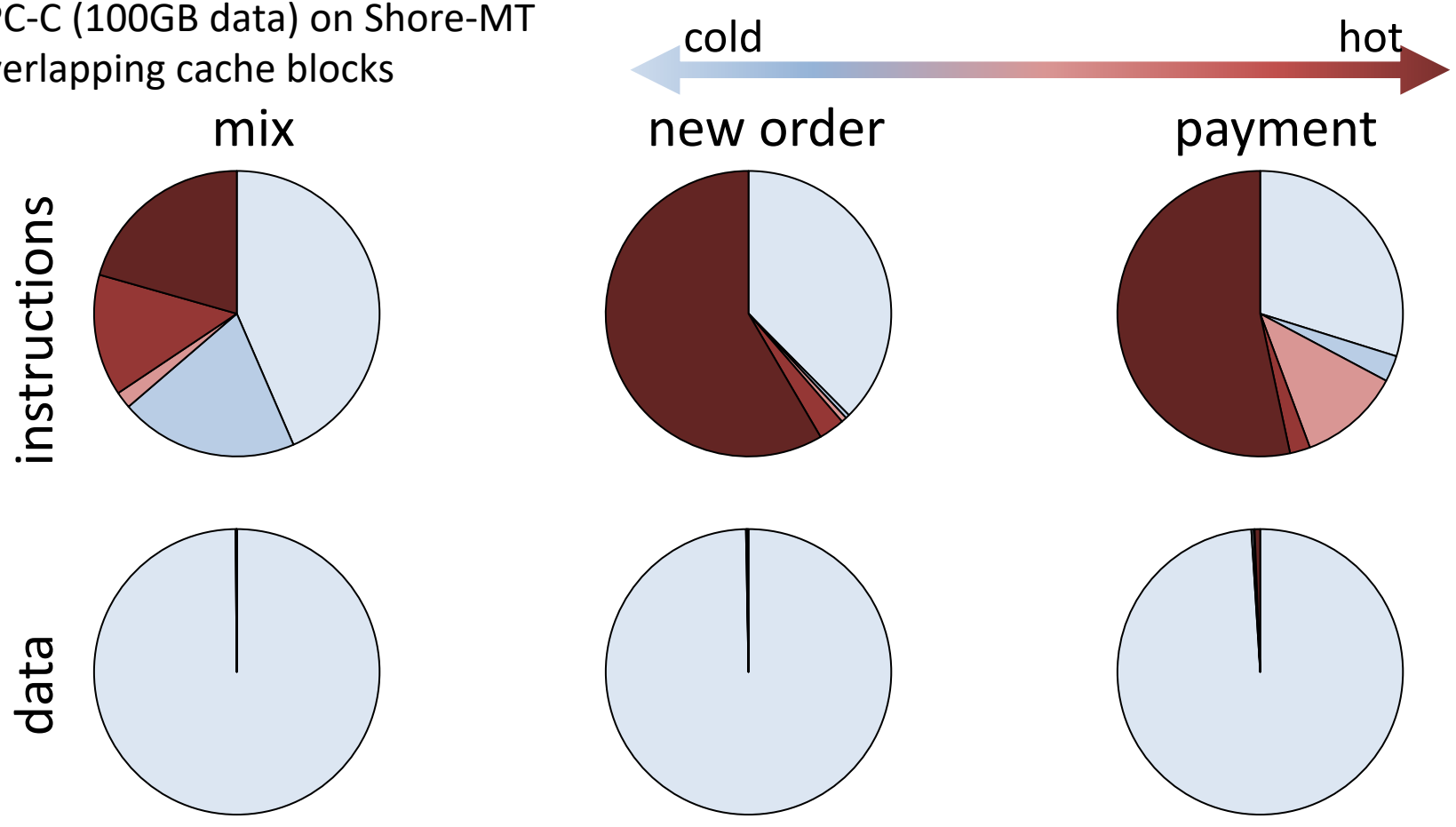
transactions under microscope



many data-intensive requests are composed of common instructions

instruction & data overlap

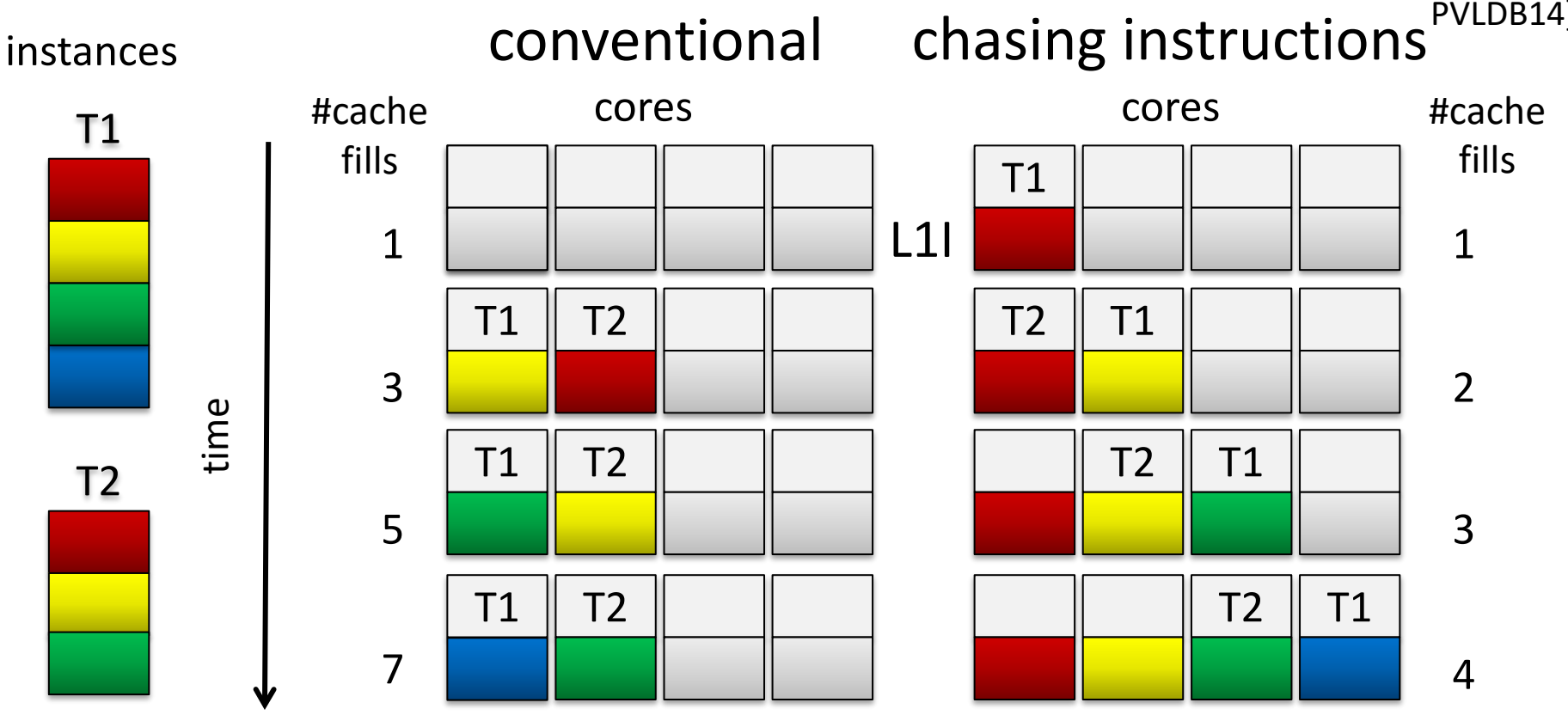
TPC-C (100GB data) on Shore-MT
overlapping cache blocks



high for instructions, low for data

exploiting instruction commonality

[MICRO12, ISCA13, PVLDB14]



can be software/hardware managed

up to 2X throughput of conventional on TPC-B/C/E

summary: OLTP & implicit parallelism

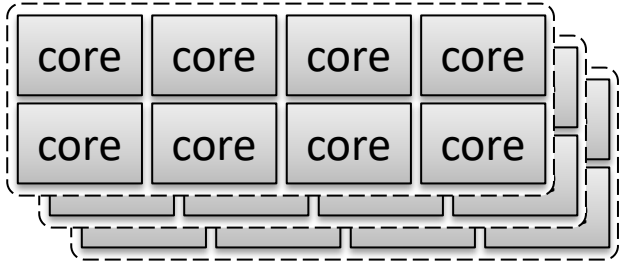
- implicit parallelism isn't completely free lunch
- >50% of cycles are stalls for traditional OLTP
  - L1-I misses are significant
- Invest in
  - utilizing instruction overlap across transactions & aggregate L1-I cache capacity
  - simplified code & cache-friendly data/code layouts that favor hardware prefetchers

agenda

- types of hardware parallelism
- implicit parallelism
- **explicit parallelism**

scaling-up vs scaling-out

scaling-up



adding more cores in a single server should give proportional performance increase

scaling-out

adding more servers in a data center should give proportional performance increase

for regular folk!

scaling-up vs scaling-out

scaling-up

adding more servers in a data center should give proportional performance increase

scaling-out

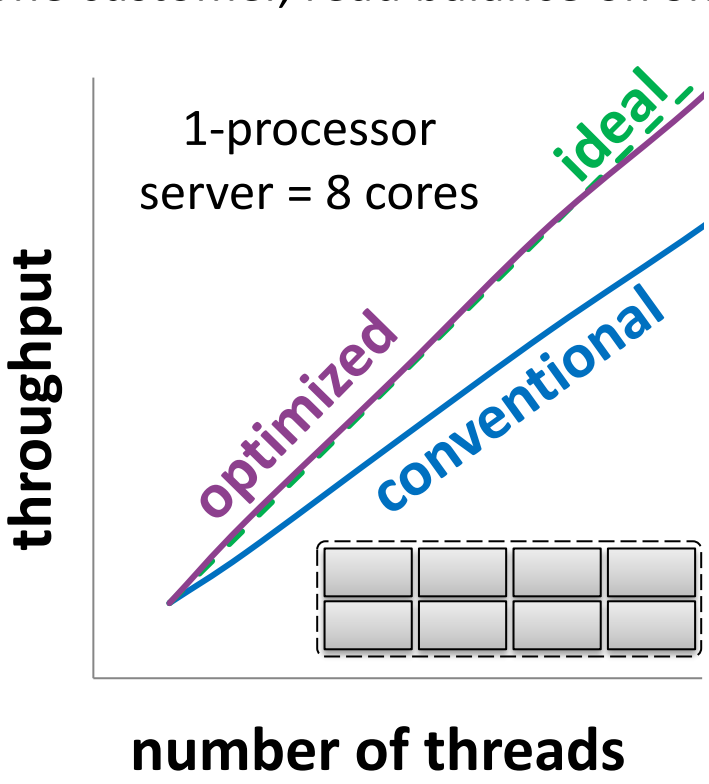
adding more data centers should give proportional performance increase

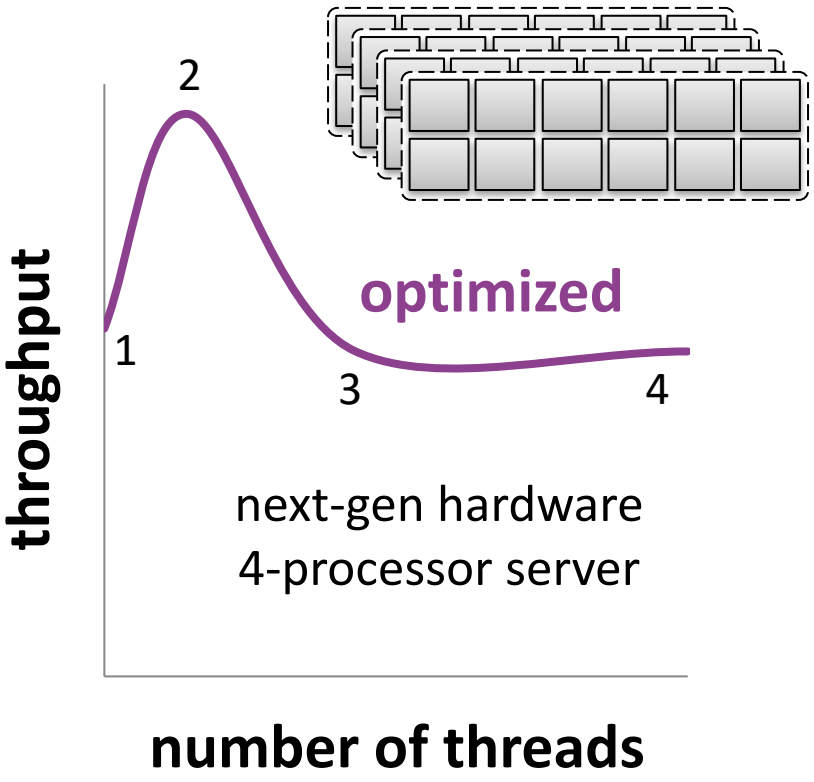
for google, amazon …!

scaling-up

[PVLDB11, PVLDB12, ICDE14]

probe one customer, read balance on Shore-MT



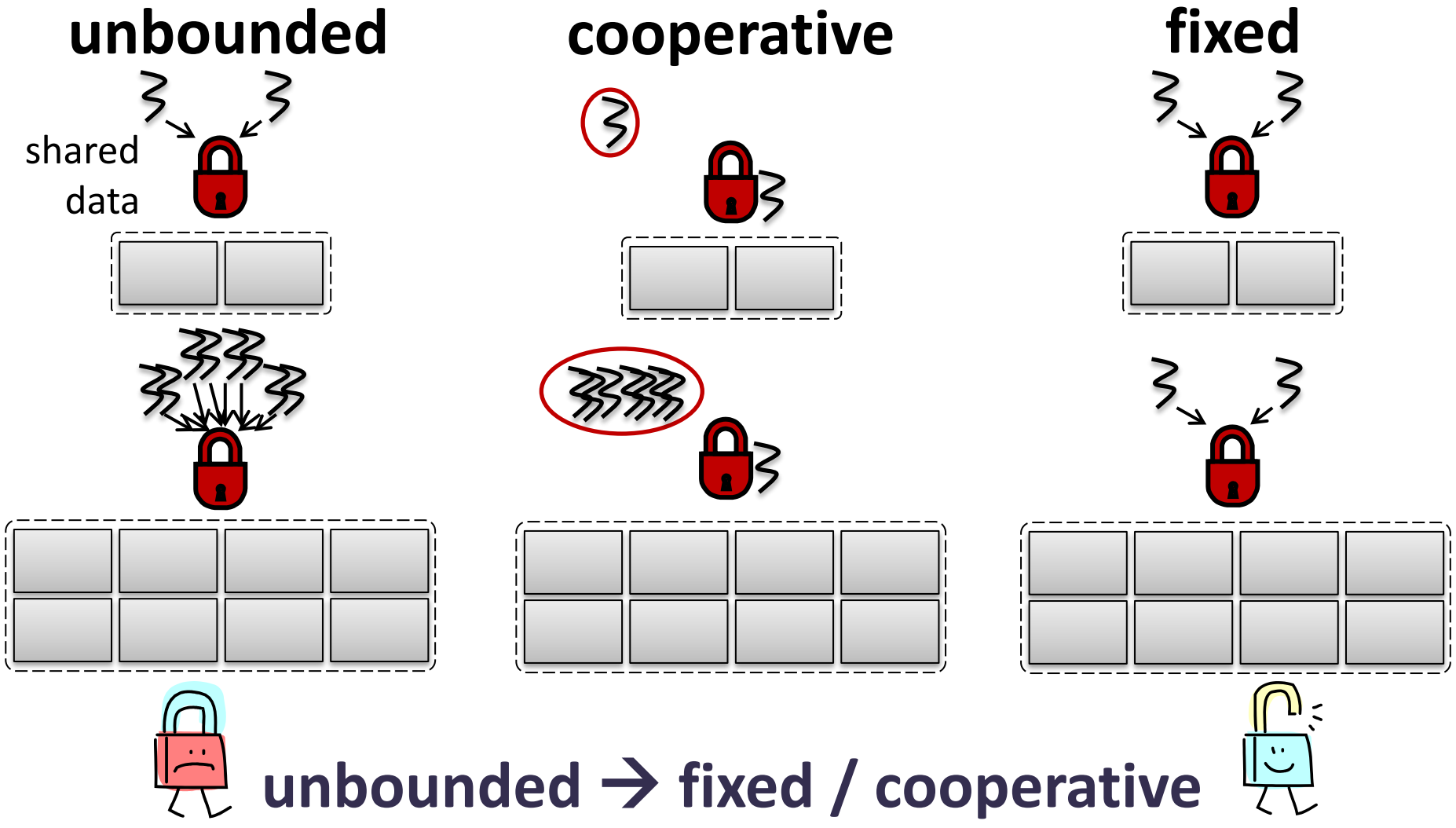


need better metrics to reason about scalability

throughput measurements are not enough

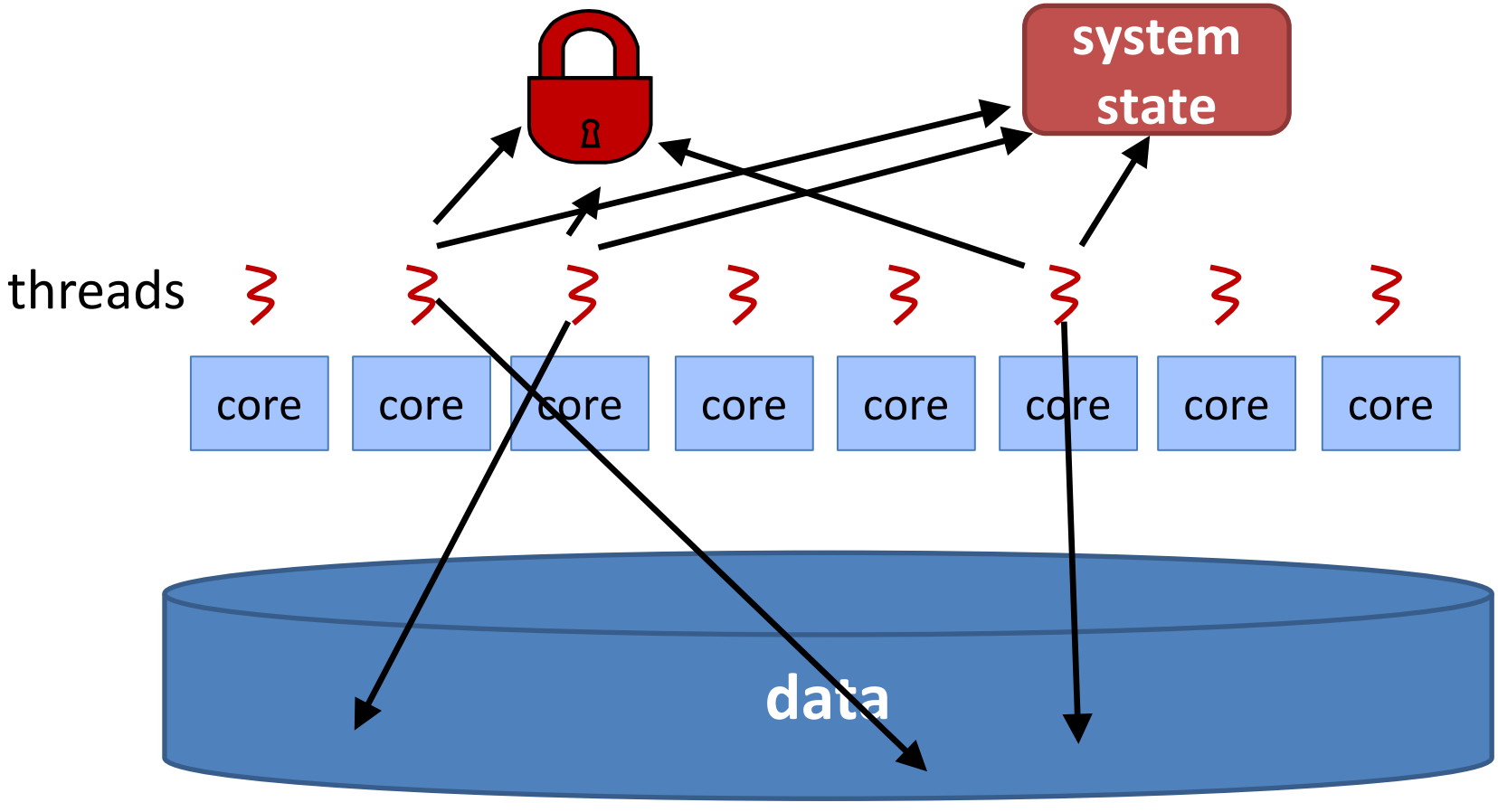
critical sections / synchronization

[PVLDB11]



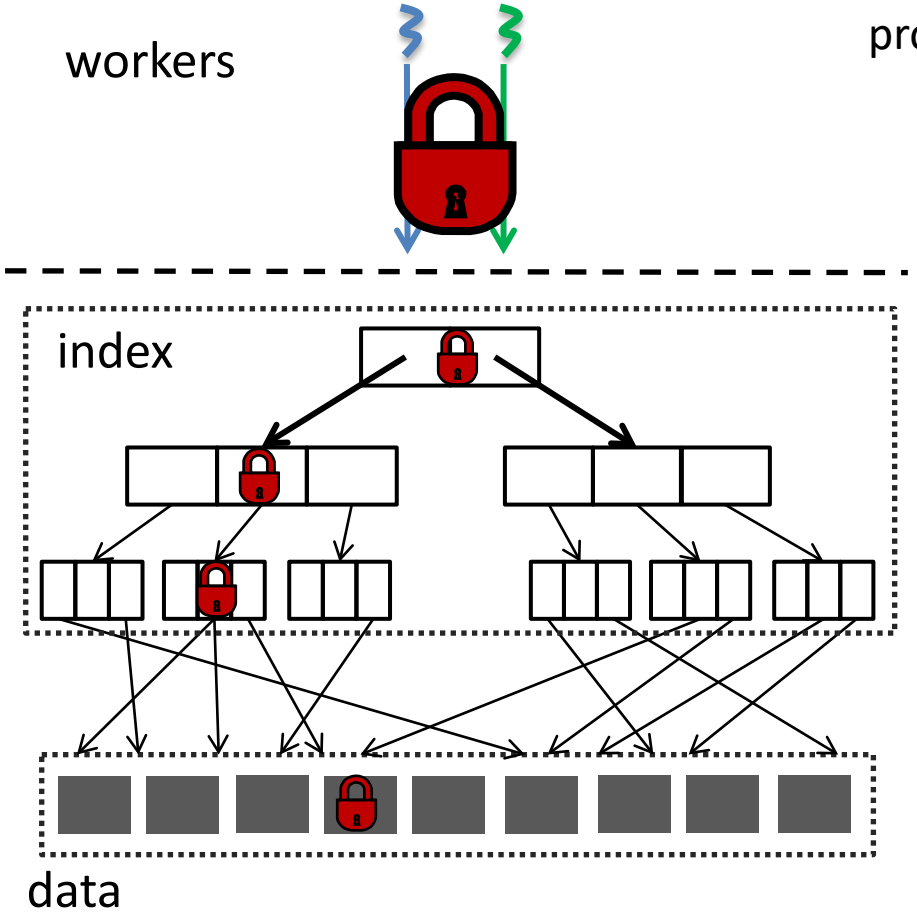
unbounded → fixed / cooperative

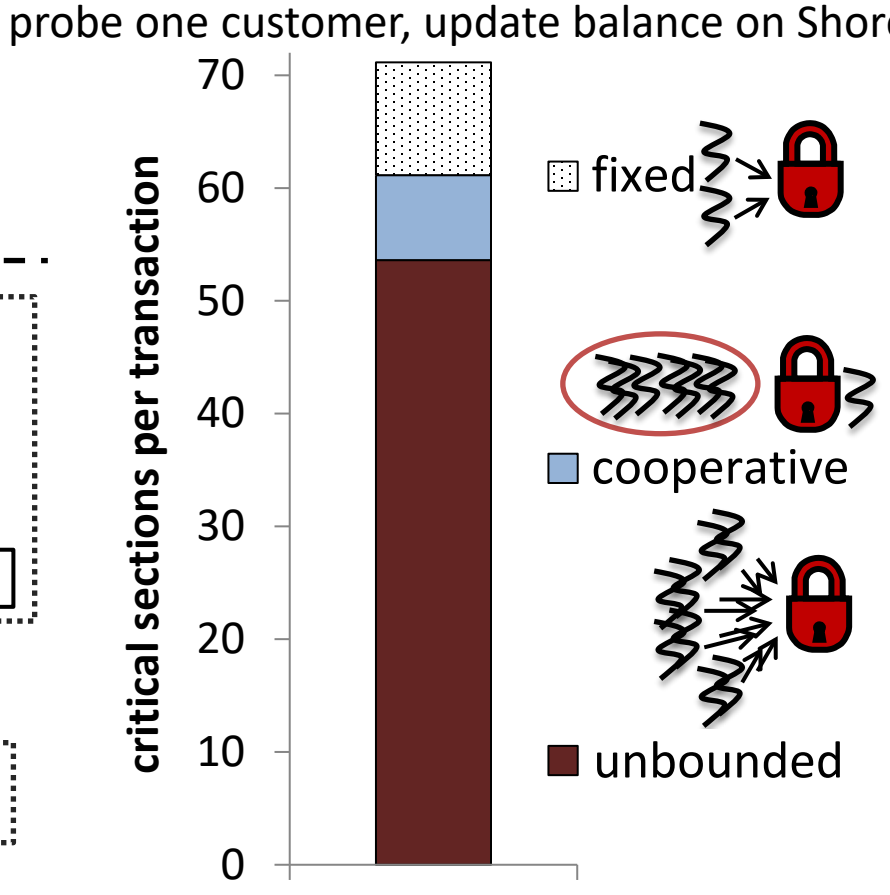
critical path of transaction execution



many unpredictable accesses to shared data

impact of unpredictable data accesses

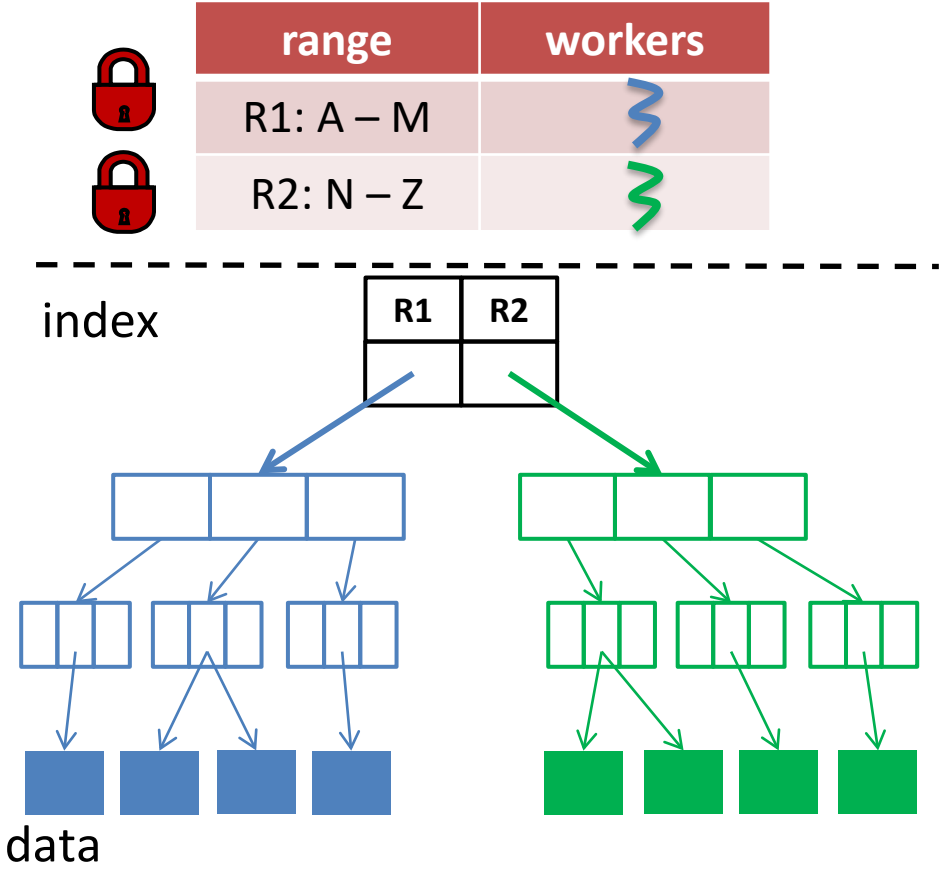


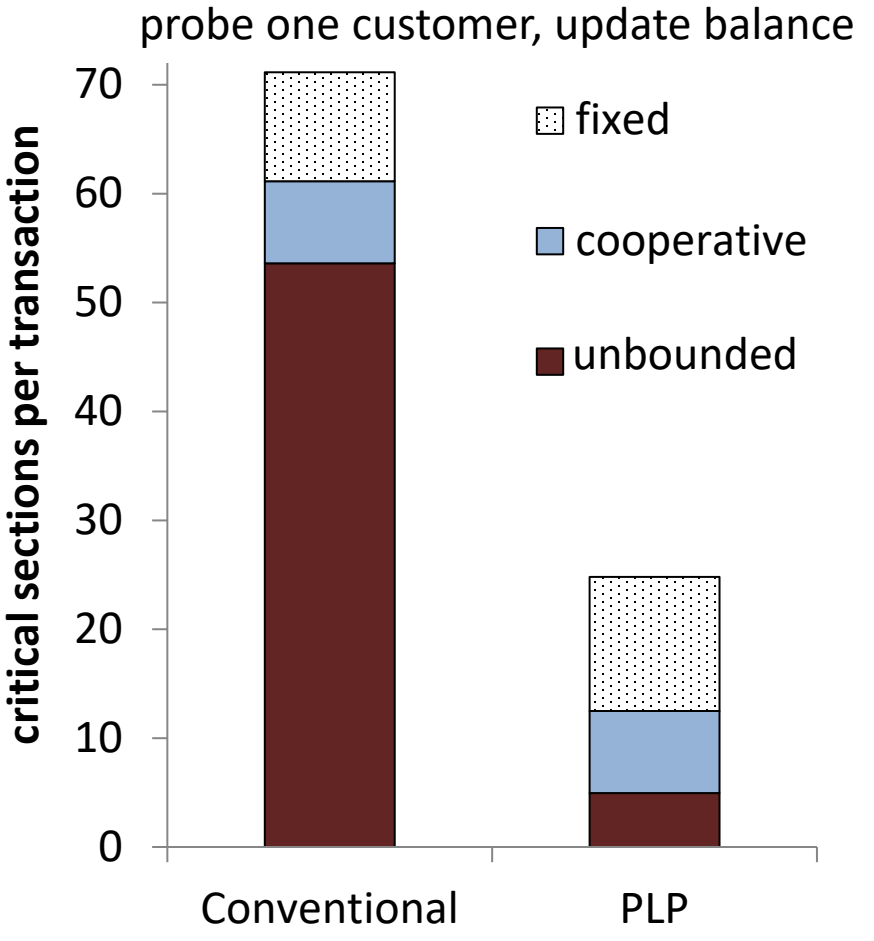


75% of critical sections are unbounded

physiological partitioning (PLP)

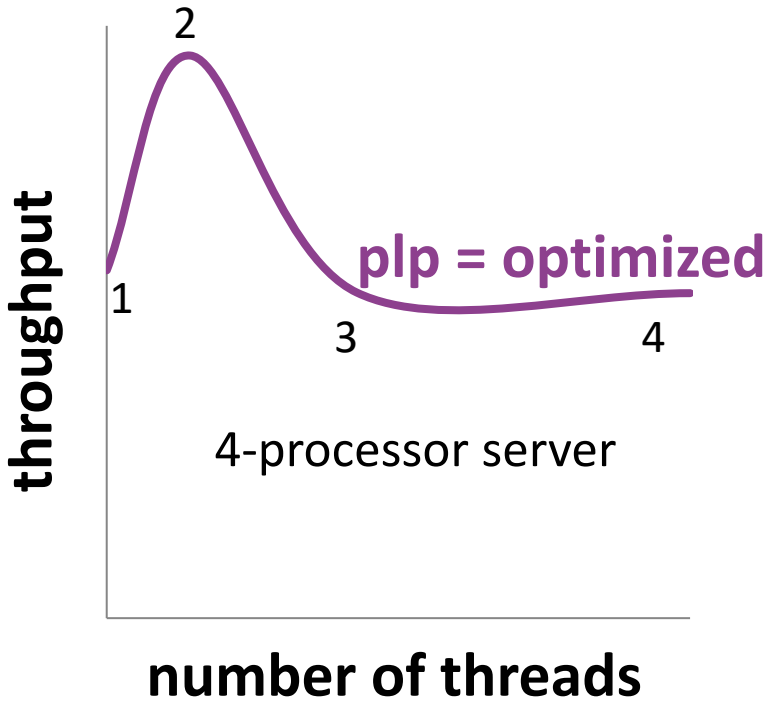
[PVLDB11]

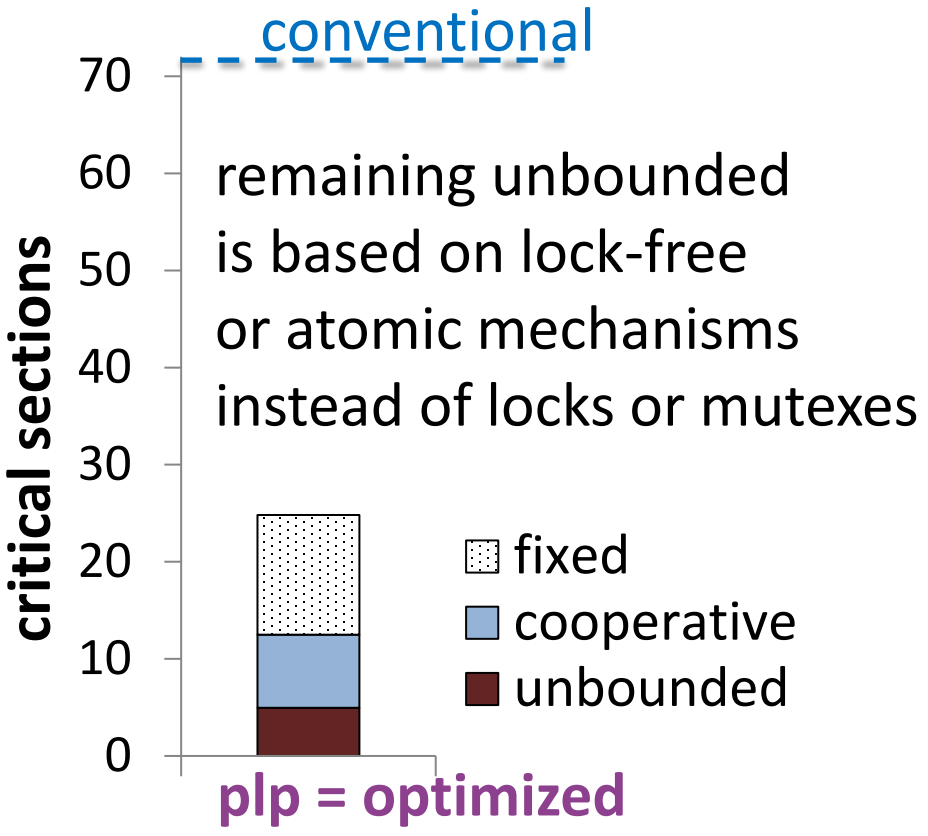




PLP eliminates 70% of critical sections

critical sections as a metric?

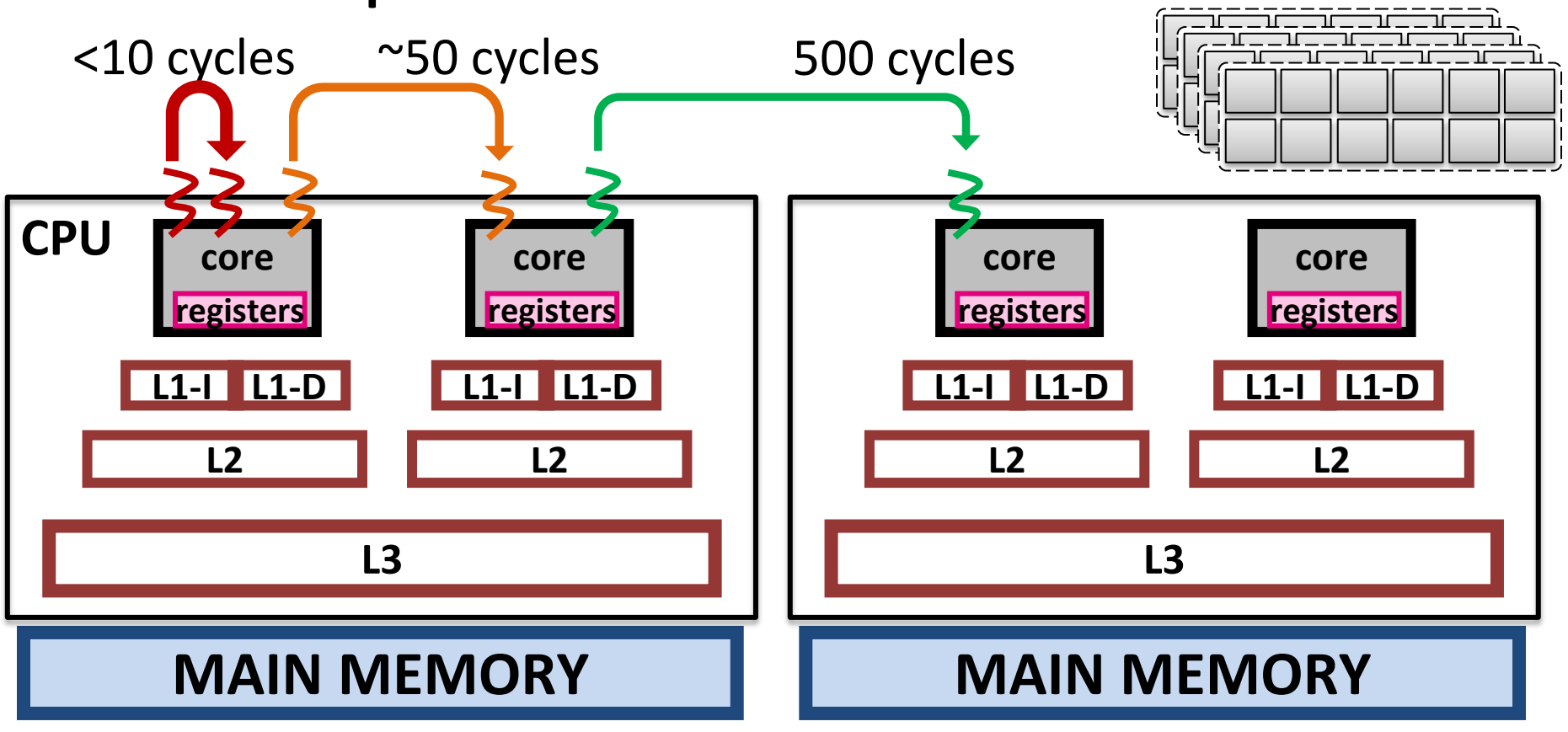




unbounded communication will hit you eventually

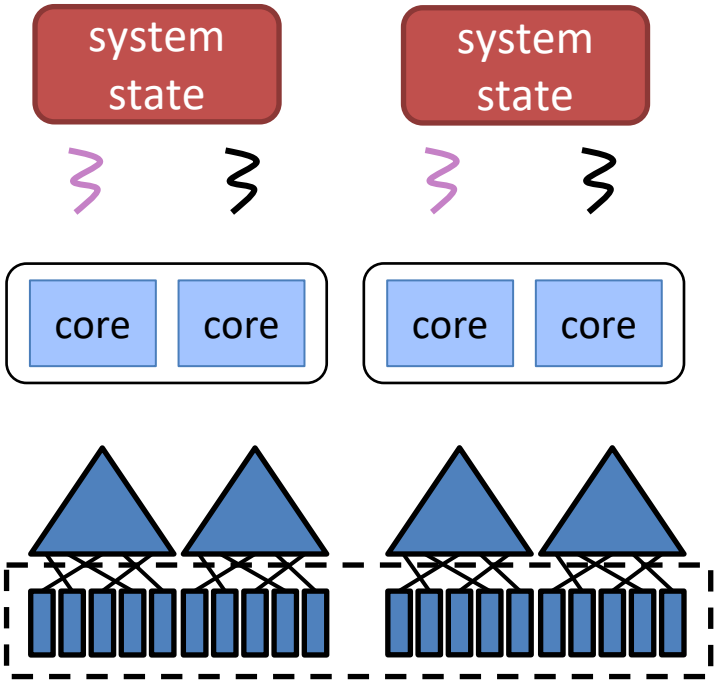
with NUMA even fixed/cooperative have issues

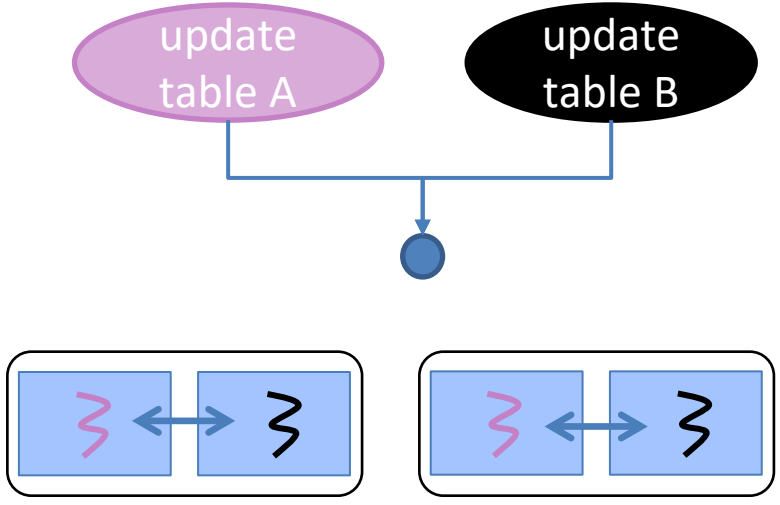
NUMA impact


<10 cycles
~50 cycles
500 cycles
CPU
core
registers
core
registers
core
registers
core
registers
L1-I
L1-D
L1-I
L1-D
L1-I
L1-D
L1-I
L1-D
L2
L2
L2
L2
L3
L3
MAIN MEMORY
MAIN MEMORY
PERSISTENT STORAGE

ATraPos: NUMA-aware PLP

[ICDE14]

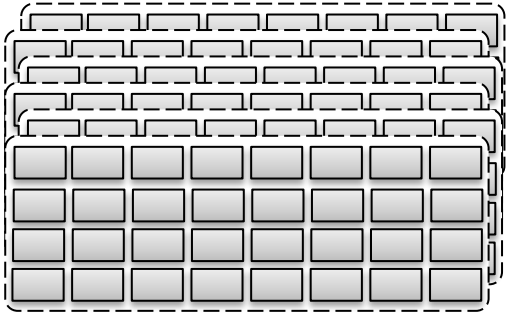




limit unbounded communication within a socket

keep access latencies predictable

summary: OLTP & explicit parallelism

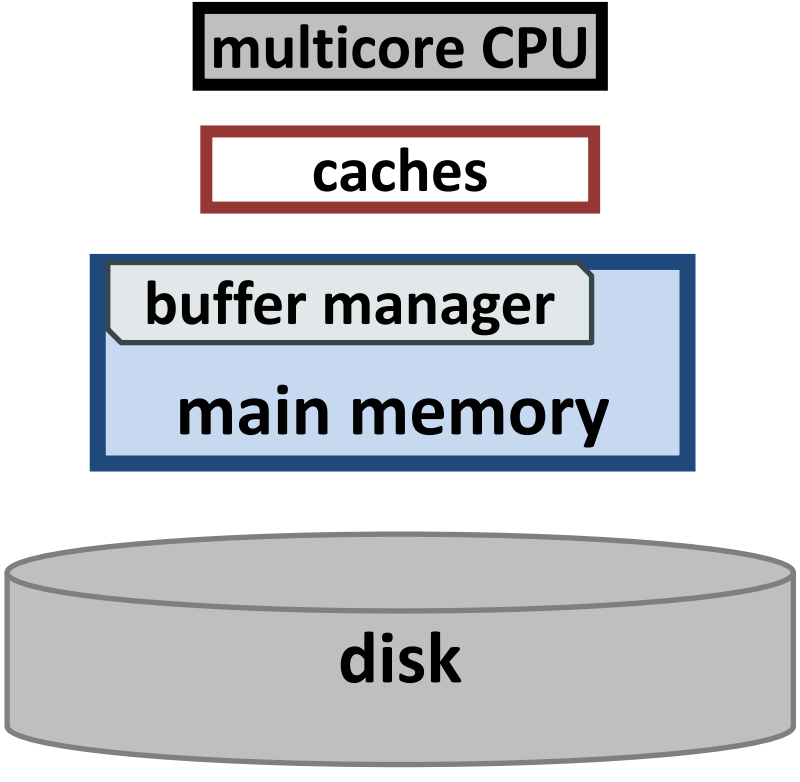


- **high throughput != scalable**
- **lock freedom != scalable**
- eliminate any unbounded communication
- keep fixed/cooperative communication among cores with similar/predictable access latency

  → avoid sharing data among cores on different processors, share within a processor (i.e., avoid NUMA impact)

today: traditional vs. modern OLTP

traditional

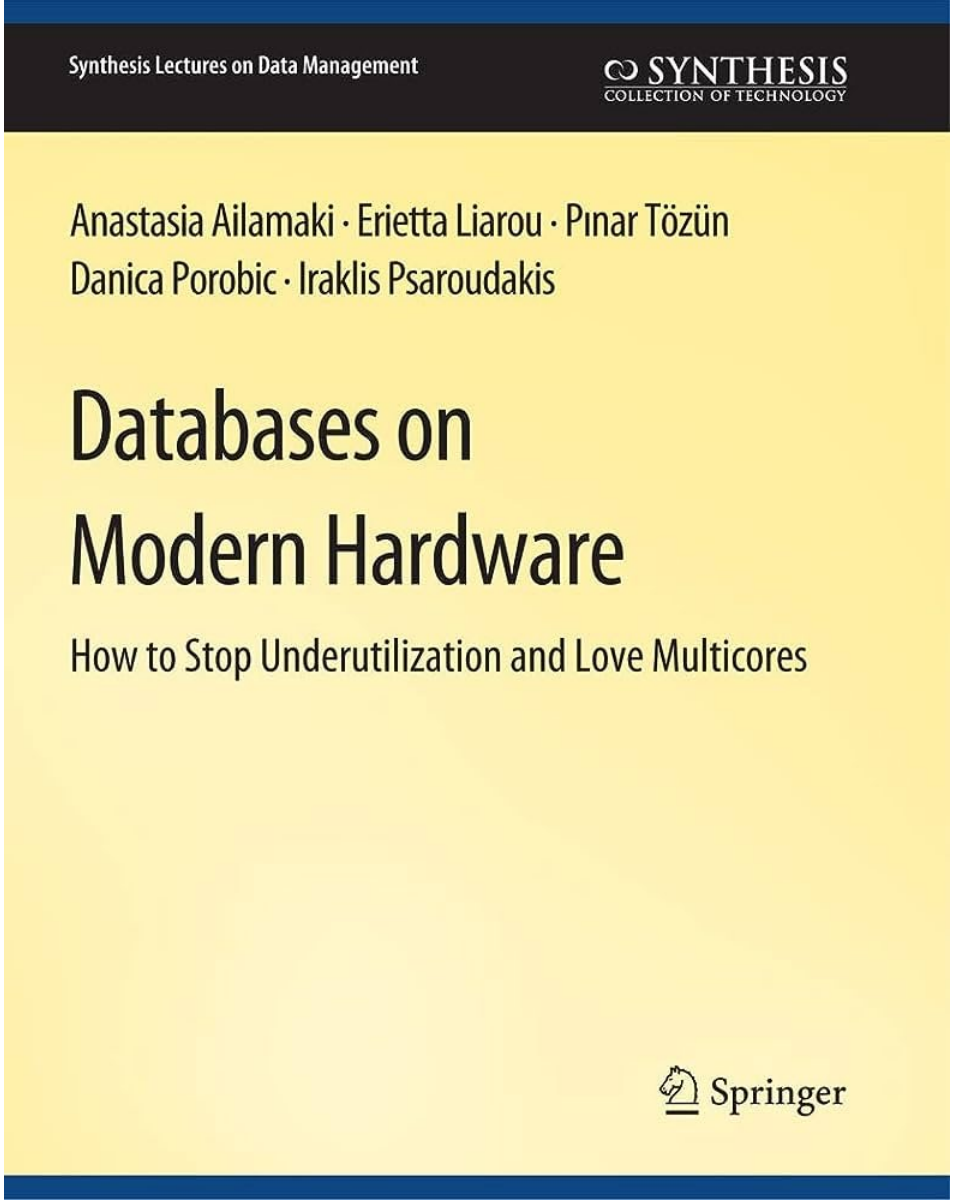


main-memory-optimized

- non-blocking concurrency control
- query compilation that generates more efficient code

- no/light buffer manager
- data organized for better cache utilization/accesses

- no disk use during analytical queries or transactions
- lightweight logging & replication for recovery
- optimize for SSDs instead


Synthesis Lectures on Data Management
SYNTHESIS
COLLECTION OF TECHNOLOGY
Anastasia Ailamaki · Erietta Liarou · Pınar Tözün
Danica Porobic · Iraklis Psaroudakis
Databases on
Modern Hardware
How to Stop Underutilization and Love Multicores
Springer

references / credits for the slides

slides 16-17, 19-20, & 22-23 are adopted from Erietta Liarou's slides for our ICDE15 tutorial
slide 37 & 41-42 are adopted from Danica Porobic's slides from ICDE14

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

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

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

[ICDE14] D. Porobic, E. Liarou, P. Tözün, A. Ailamaki. ATraPos: Adaptive Transaction Processing on Hardware Islands.

[ICDE15] A. Ailamaki, E. Liarou, P. Tözün, D. Porobic, I. Psaroudakis. How to Stop Underutilization and Love Multicores.

[ISCA13] I. Atta, P. Tözün, X. Tong, A. Ailamaki, A. Moshovos. STREX: Boosting Instruction Cache Reuse in OLTP Workloads through Stratified Transaction Execution.

references / credits for the slides

[PVLDB14] P. Tözün, I. Atta, A. Ailamaki, A. Moshovos. ADDICT: Advanced Instruction Chasing for Transactions.

[SIGMOD16] U. Sirin, P. Tözün, D. Porobic, A. Ailamaki. Micro-architectural Analysis of In-memory OLTP.

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

[PVLDB11] I. Pandis, P. Tözün, R. Johnson, A. Ailamaki. PLP: page latch-free shared-everything OLTP.

[PVLDB12] D. Porobic, I. Pandis, M. Branco, P. Tözün, A Ailamaki. OLTP on Hardware Islands.

other references for the interested

[CIDR15] M. Karpathiotakis, I. Alagiannis, T. Heinis, M. Branco, A. Ailamaki. Just-in-time data virtualization: Lightweight data management with ViDa.

[DEBull14] T. Neumann, V. Leis. Compiling Database Queries into Machine Code.
[DEBull19] P. Tözün, H. Kotthaus. Scheduling Data-Intensive Tasks on Heterogeneous Many Cores.

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

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

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

[ICDE14b] N. Malviya, A. Weisberg, S. Madden, and M. Stonebraker: Rethinking Main Memory OLTP Recovery.

[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.

[MICRO13] C. Kaynak, B. Grot, and B. Falsafi: SHIFT: Shared History Instruction Fetch for Lean-Core Server Processors.

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

[PVLDB10] R. Johnson, I. Pandis, R. Stoica, M. Athanassoulis, and A. Ailamaki: Aether: A Scalable Approach to Logging.

other references for the interested

[PVLDB11] T. Neumann: Efficiently compiling efficient query plans for modern hardware.

[PVLDB12] P. Larson, S. Blanas, C. Diaconu, C. Freedman, J. M. Patel, and M. Zwilling: High-performance concurrency control mechanisms for main-memory databases.

[PVLDB13] K. Ren, A. Thomson, and D. J. Abadi: Lightweight locking for main memory database systems.

[PVLDB14] Y. Klonatos, C. Koch, T. Rompf, and H. Chafi: Building Efficient Query Engines in a High-Level Language.

[PVLDB15] X. Yu, G. Bezerra, A. Pavlo, S. Devadas, and M. Stonebraker: Staring into the Abyss: An Evaluation of Concurrency Control with One Thousand Cores.

[SIGMOD10] E. P. Jones, D. J. Abadi, and S. Madden: Low overhead concurrency control for partitioned main memory databases.

[SIGMOD13] 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.

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

[VLDB07] 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).

[VLDBJ] T Bang, N May, I Petrov, C Binnig. The full story of 1000 cores: An examination of concurrency control on real (ly) large multi-socket hardware.

backup

kicking of the discussion

- to partition or not to partition?
  - lightweight locks & concurrency control still doesn't prevent unbounded communication
  - even if you don't go strict partitioning route, you need some partitioning to limit unbounded communication

- transaction processing & hardware accelerators?
  - Dennard scaling still doesn't hold
  - economies of scale also doesn't hold
  - hardware specialization is inevitable

- in-memory-optimized vs. SSD-optimized ?

summary: hardware parallelism

Hardware gives different parallelism opportunities.

Software systems used to be ignorant of this parallelism because it used to be mainly implicit.

Today, systems do not have this luxury because we have more and more explicit parallelism.

In the future, this parallelism will also increasingly be provided by heterogeneous processing units.

goal: design systems that are aware of the hardware parallelism (ideally all types of it) & its implications!

main-memory database systems

some downsides

startups/restarts are expensive

- need to load everything in-memory
- indexes has to be rebuilt
- recovery takes longer if not done carefully

could be expensive to store all data in memory

- most main-memory systems later added support for efficiently accessing cold data from disk
- today, more & more SSD-optimized systems instead

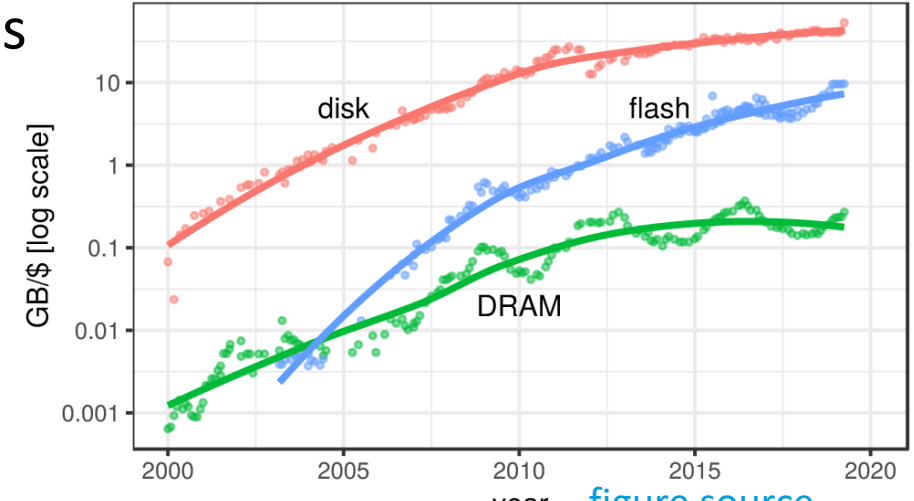


figure source

micro-architecture of an OoO processor core

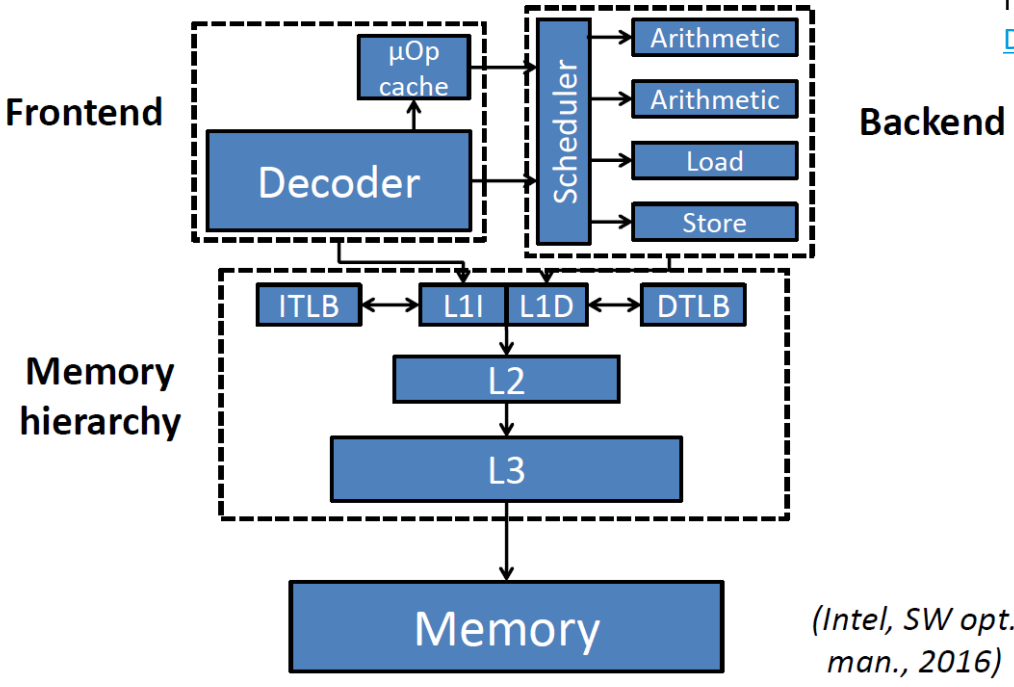


figure from Utku Şirin's DaMoN 2017 presentation

(Intel, SW opt. man., 2016)

delays in fetching, decoding, etc. an instruction cause *frontend stalls*, rest cause *backend stalls*

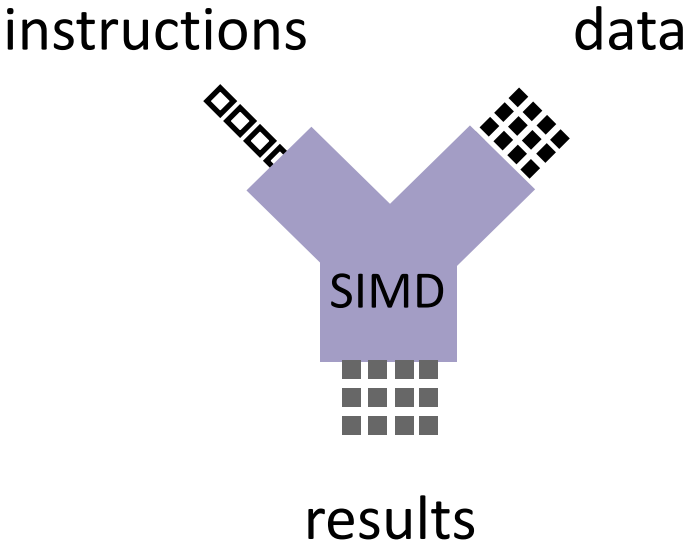
what else can be done?

- simplified code
  - newer in-memory-optimized systems have a smaller instruction footprint compared to traditional disk-based ones
- better code layout
  - minimize jumps → better utilizing hardware prefetchers
  - profiler-guided optimizations (static)
  - just-in-time (dynamic) [CIDR15]
- better code compilation
  - e.g., HyPer, Umbra, Hekaton, MemSQL
- improving index layouts → for data access efficiency

single instruction multiple data (SIMD)

if you want to apply the same instructions over multiple data items

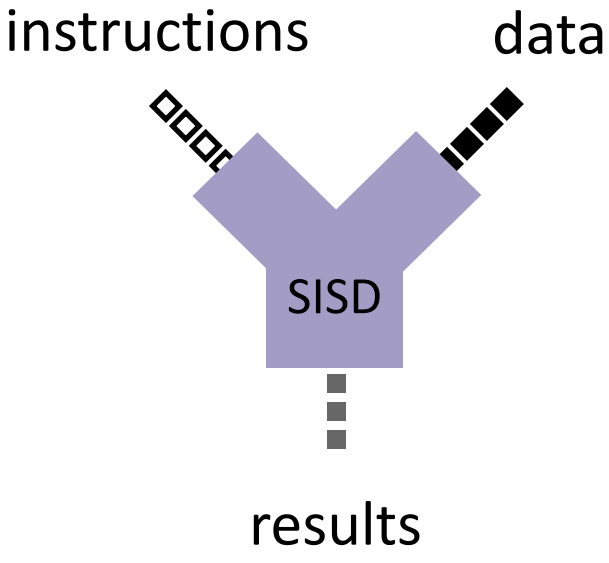
GPUs are built on this principle

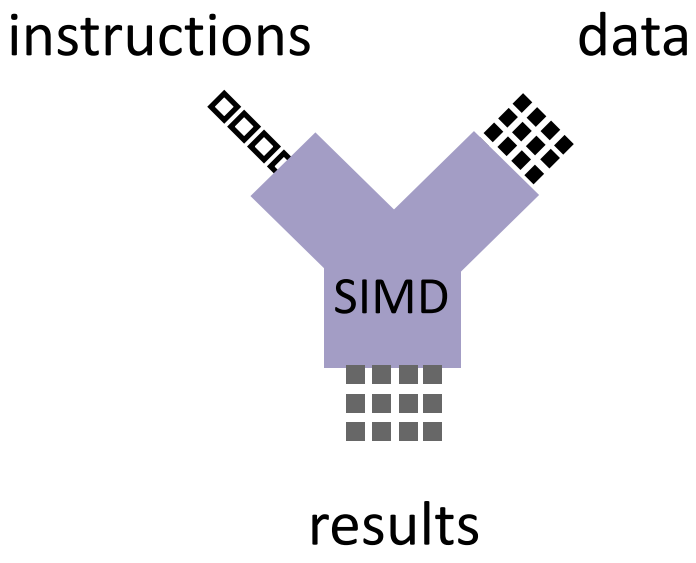


also implicit data parallelism, but this one has to be managed/issued by the software

single instruction multiple data (SIMD)

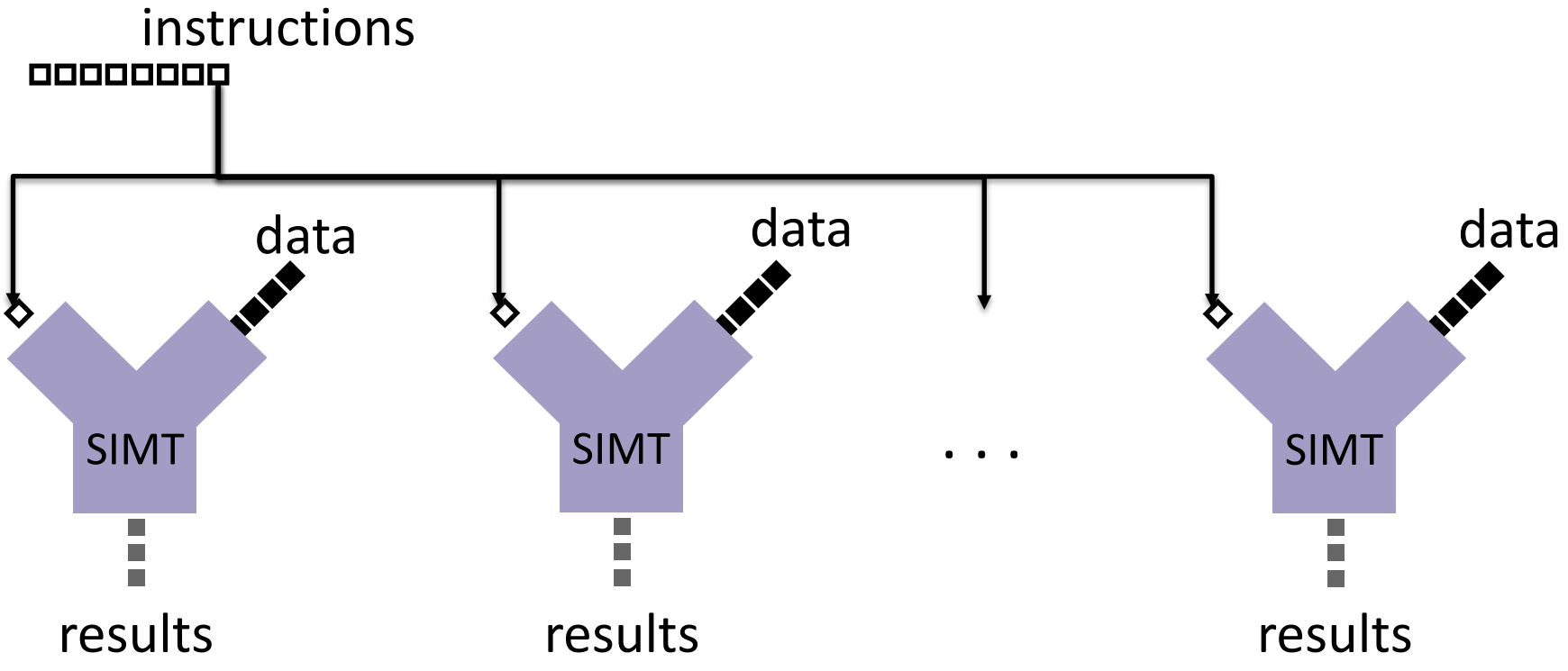
from lecture 3





GPUs are like SIMD machines
they support extreme parallelism

single instruction multiple thread (SIMT)

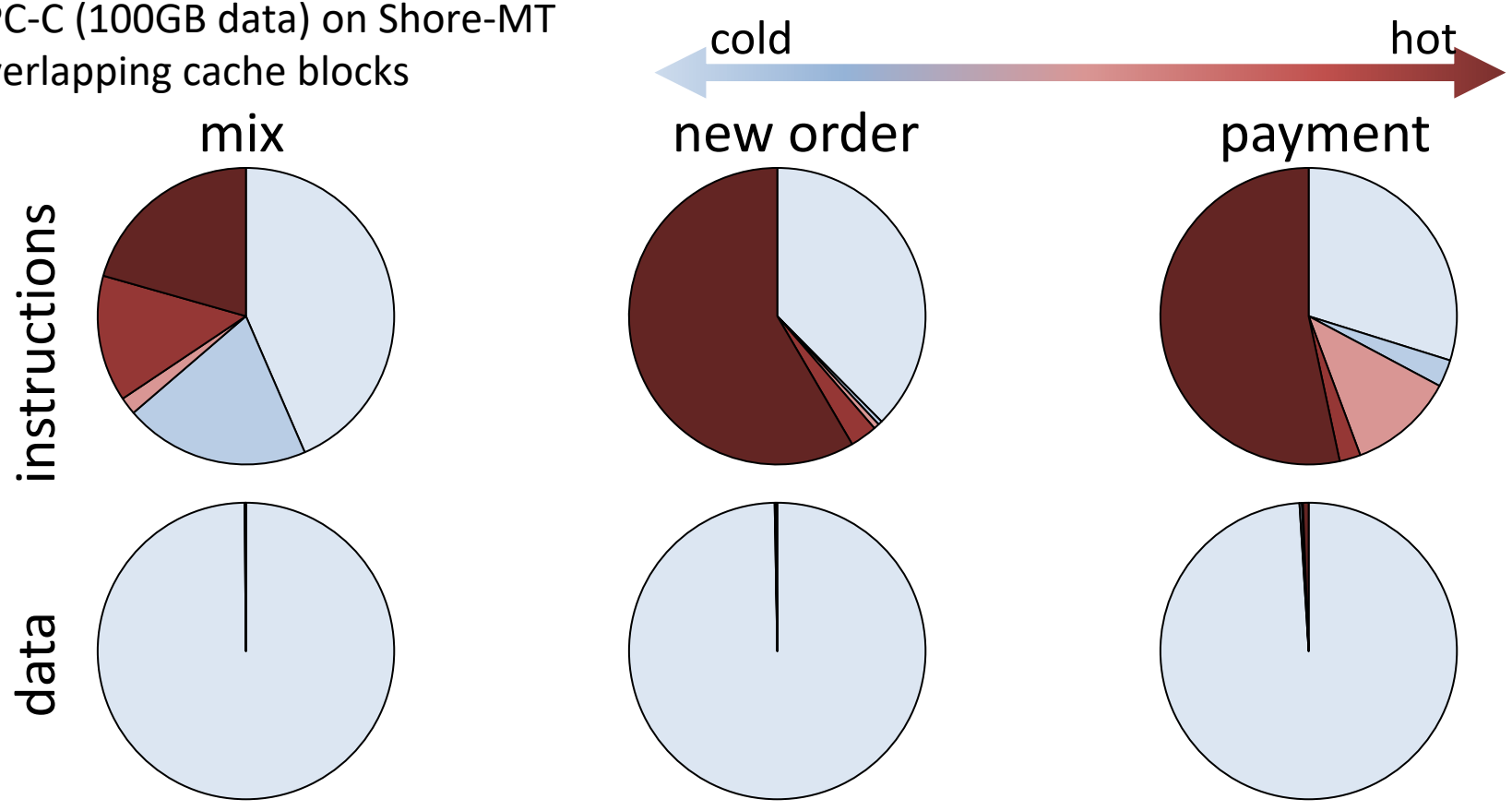


GPUs are based on SIMT

instruction & data overlap

[PVLDB14]

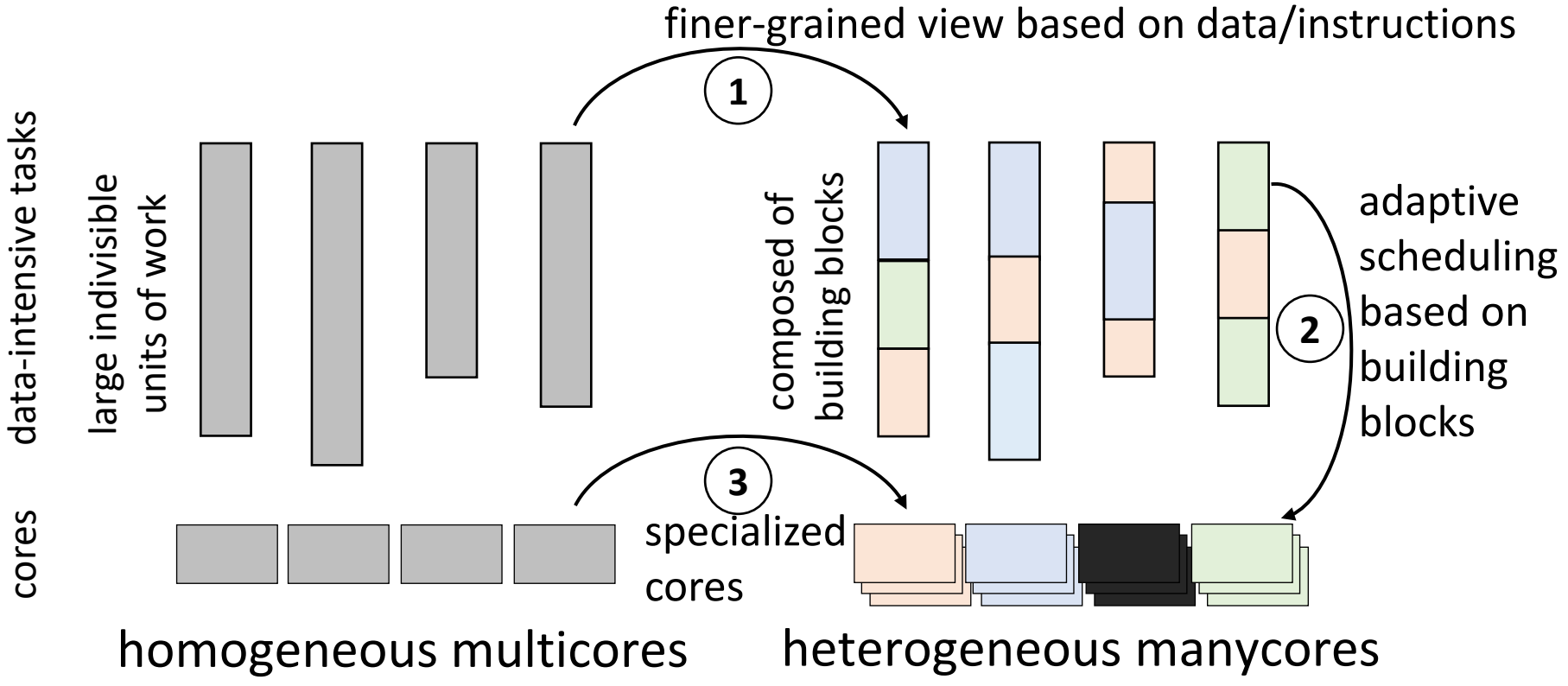
TPC-C (100GB data) on Shore-MT
overlapping cache blocks



different transaction instances are composed of common instructions

how to achieve the goal? **thank you!**

[DeBull19]



black-box view leads to sub-optimal hardware utilization

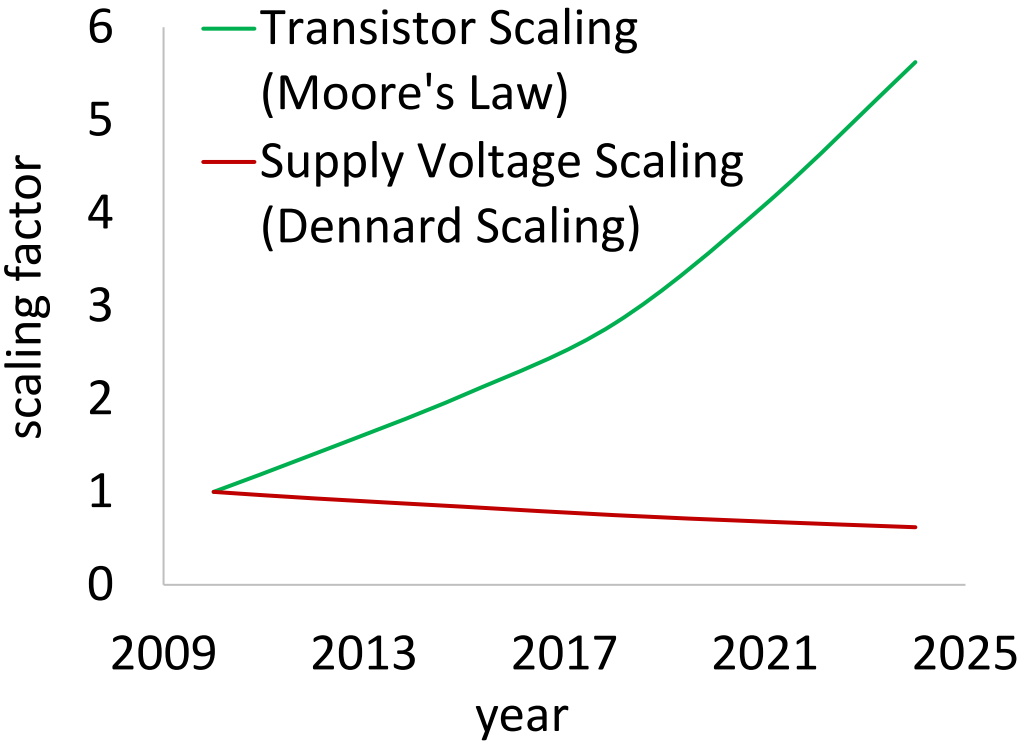
finer-grained view helps with hardware-consciousness

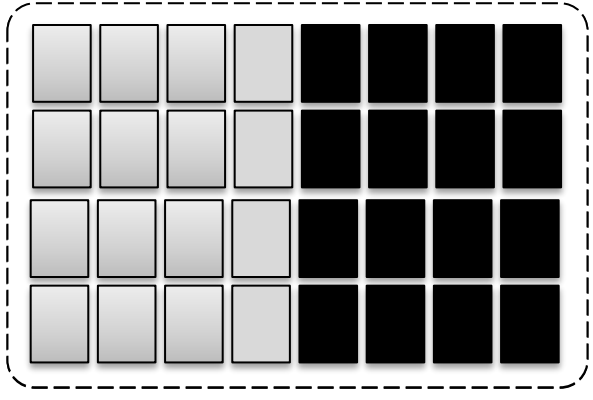
lecture agenda

- types of hardware parallelism
- implicit parallelism
- explicit parallelism
- **adding heterogeneity**

dark silicon

[ISCA11, MICRO11]



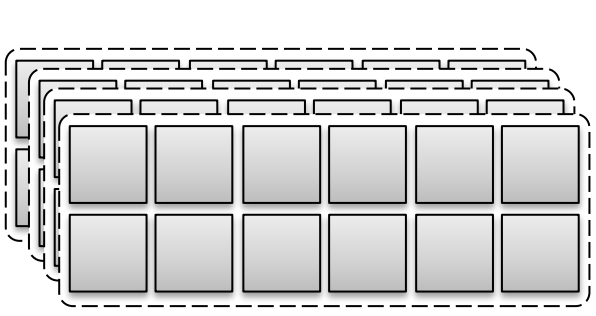


can still pack more cores in a processor

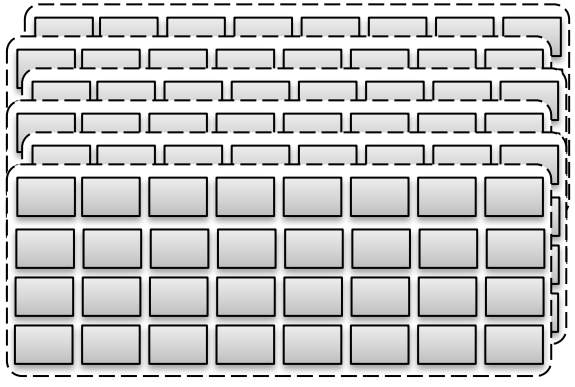
cannot fire all of them up simultaneously

what this means (for servers)

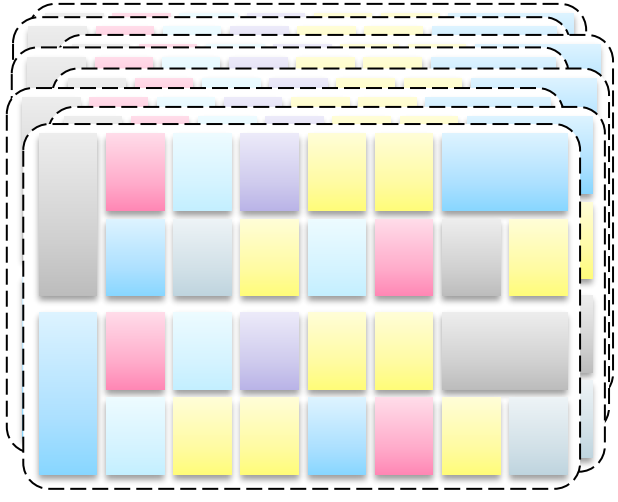
today's commodity | **future's commodity**



multisocket multicores



many (***light)*** cores



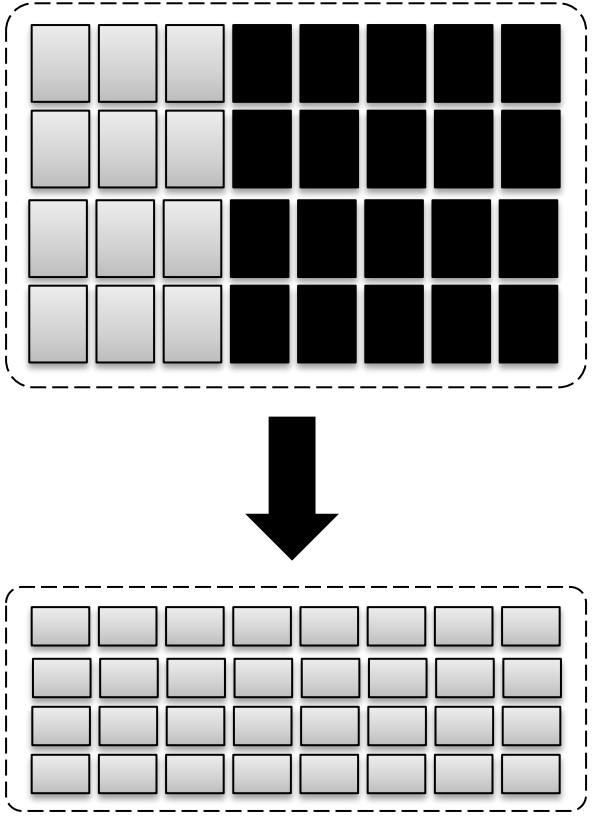
many ***diverse*** cores

more parallelism & heterogeneity

most software systems are not ready for either

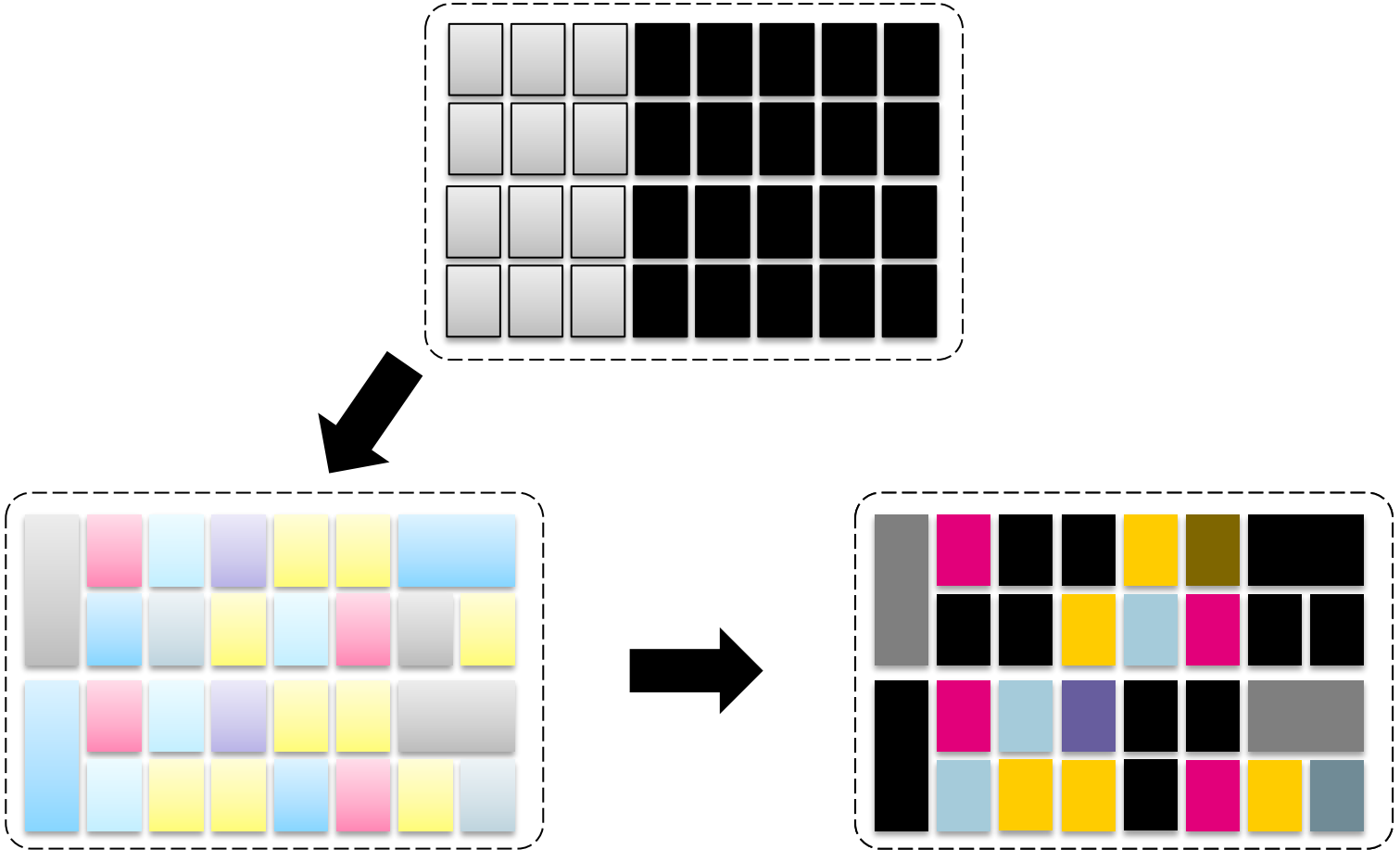
lighter cores

[MICRO10]



not feasible for latency critical tasks
can be less energy-efficient in the long run

diverse cores



better long-term solution, but creates mess

exploiting diversity

[DaMoN14, SIGMOD18a, SIGMOD18b]

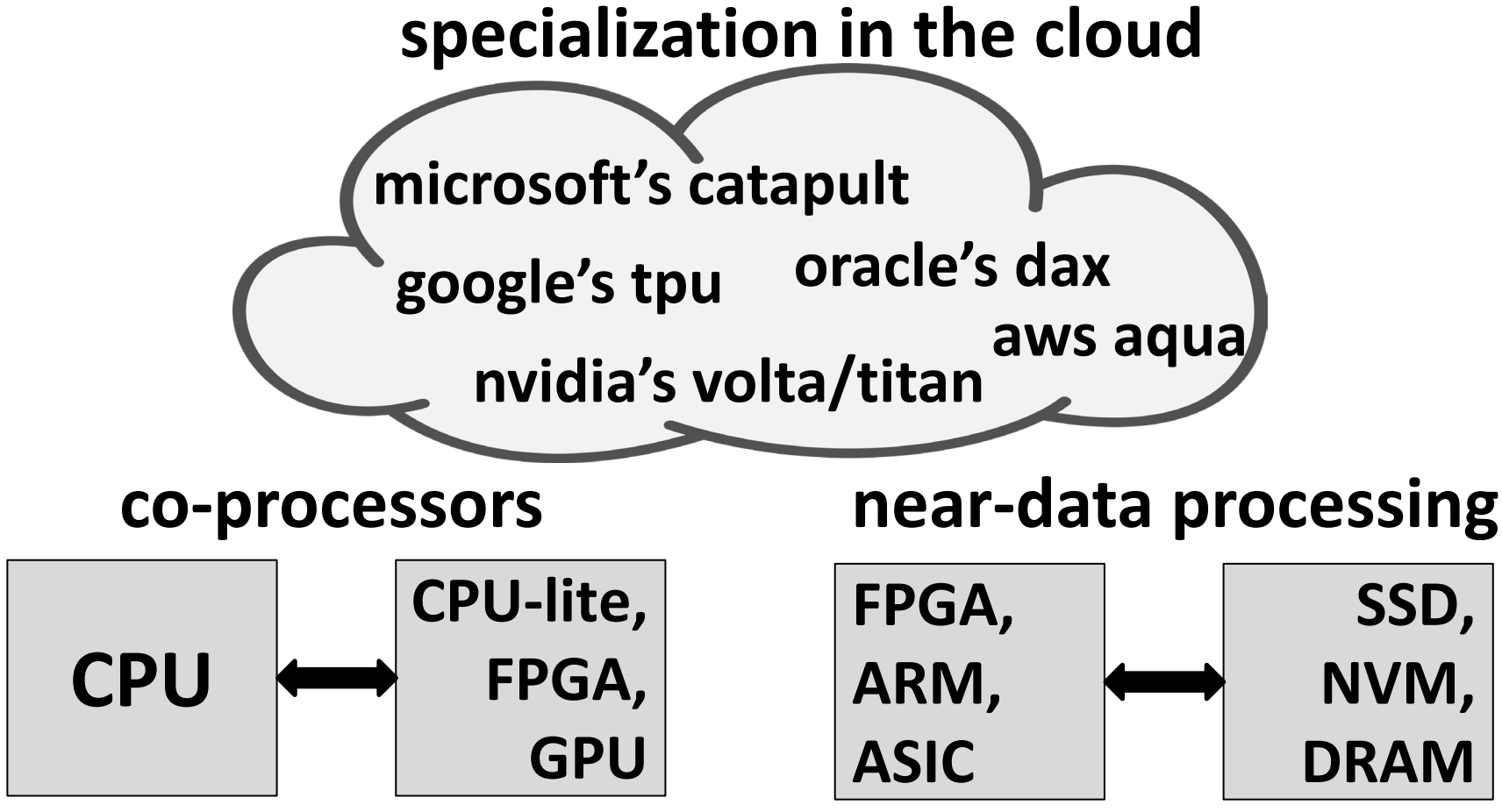
- finer-grained scheduling & energy management
  - controlling which cores are active at a time running what types of tasks & what frequency cores run on

[DEBull14, CIDR15, PVLDB16]

- language support & advanced query compilation
  - high-level languages for programs, then compile that onto different cores where instruction sets may be different

- right use case for specialization
  - must pick frequent & important tasks to accelerate – otherwise, no economic viability

today's landscape



references / credits

links for “today’s landscape” slide:

https://www.microsoft.com/en-us/research/project/project-catapult/

https://cloud.google.com/tpu/

https://www.arm.com/products/processors/biglittleprocessing.php

https://www.altera.com/solutions/acceleration-hub/overview.html

https://www.intel.com/content/www/us/en/products/processors/xeon-phi/xeon-phi-processors.html

https://www.ibm.com/support/knowledgecenter/en/POWER9/p9hdx/POWER9welcome.htm

https://swisdev.oracle.com/DAX/DAXwhatis.php

https://github.com/DFC-OpenSource

http://lightnvm.io/

https://blocksandfiles.com/2019/12/05/amazon-aqua-data-warehouse-acceleration-hardware/

references for the backup

[DaMoN14] I. Psaroudakis, T. Kissinger, D. Porobic, T. Ilsche, E. Liarou, P. Tözün, A. Ailamaki, W. Lehner. Dynamic Fine-Grained Scheduling for Energy-Efficient Main-Memory Queries.

[DEBull19] P. Tözün, H. Kotthaus. Scheduling Data-Intensive Tasks on Heterogeneous Many Cores.

[ISCA11] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, D. Burger. Dark Silicon and the End of Multicore Scaling.

[MICRO10] U. Hölzle. Brawny cores still beat wimpy cores, most of the time.

[MICRO11] N. Hardavellas, M. Ferdman, B. Falsafi, A. Ailamaki. Toward Dark Silicon in Servers.

[PVLDB16] H. Pirk, O. Moll, M. Zaharia, S. Madden. Voodoo - A Vector Algebra for Portable Database Performance on Modern Hardware.

[SIGMOD18a] T. Kissinger, D. Habich, W. Lehner. Adaptive Energy-Control for In-Memory Database Systems.

[SIGMOD18b] M. Korkmaz, M. Karsten, K. Salem, S. Salihoglu. Workload-Aware CPU Performance Scaling for Transactional Database Systems.