DASYA

www.dasya.dk
@dasyaITU

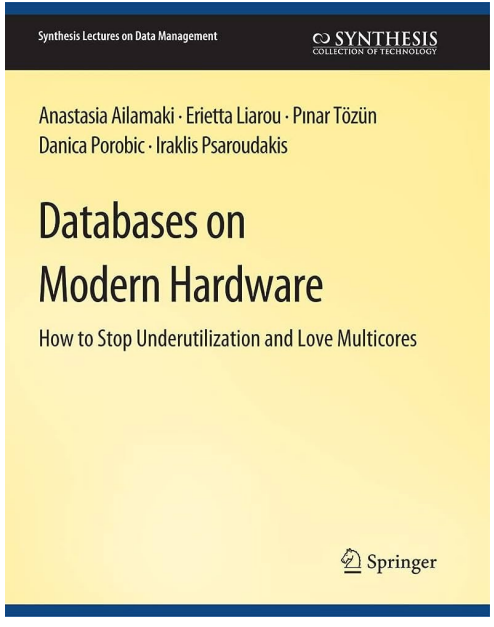
RAD

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



HPI – Lecture Series on DB Research
October 24, 2023

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, …*

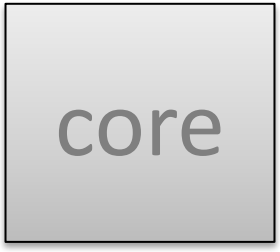
➔ **primary applications for databases**

➔ **required functionality & optimizations differ**

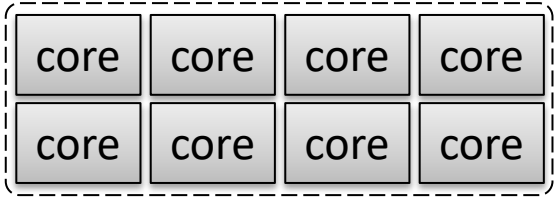
evolution of general-purpose CPU

… the hardware we run transactions on

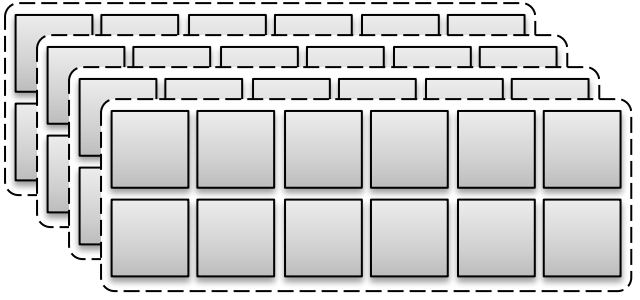
2005



single-core CPUs



multicore CPUs



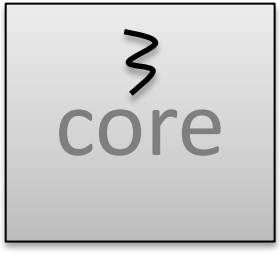
multisocket multicore CPUs

faster & more-complex cores over time

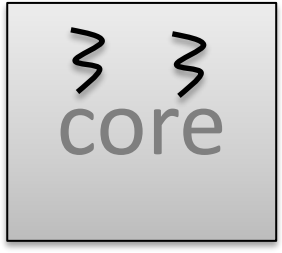
similar speed & complexity in a core, more cores over time

types of hardware parallelism

implicit/vertical parallelism



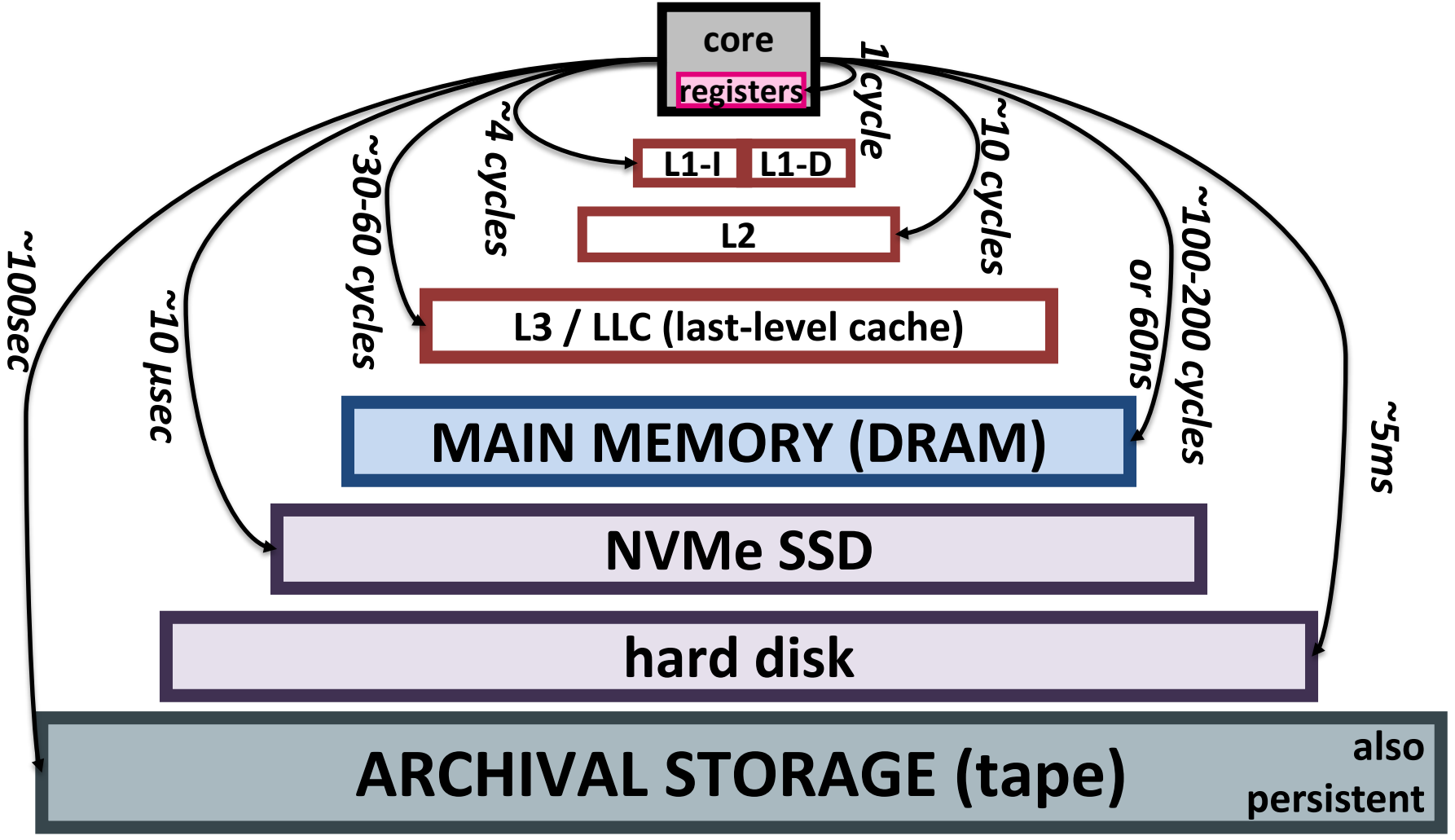
instruction & data parallelism
hardware does this automatically



multithreading
threads share
execution cycles
on the same core

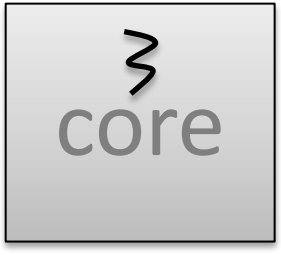
why do we need this?

single-core – access latency to storage

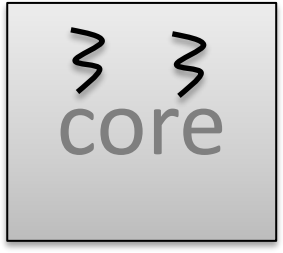


types of hardware parallelism

implicit/vertical parallelism



instruction & data parallelism
hardware does this automatically

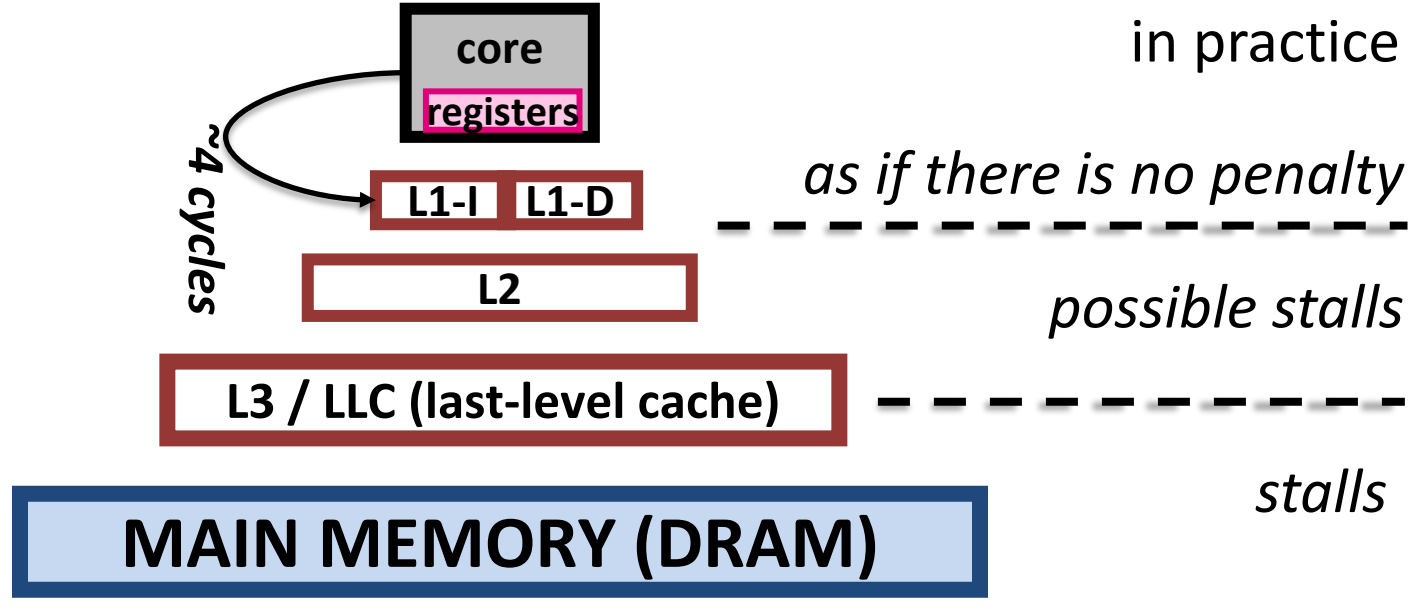


multithreading
threads share
execution cycles
on the same core

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

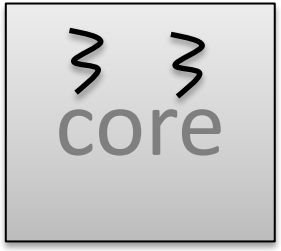
single-core – access latency to storage



stalls = 😴

types of hardware parallelism

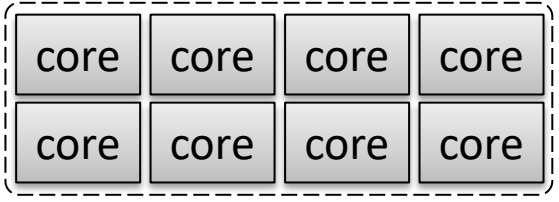
implicit/vertical parallelism ⟷ **explicit/horizontal parallelism**



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!

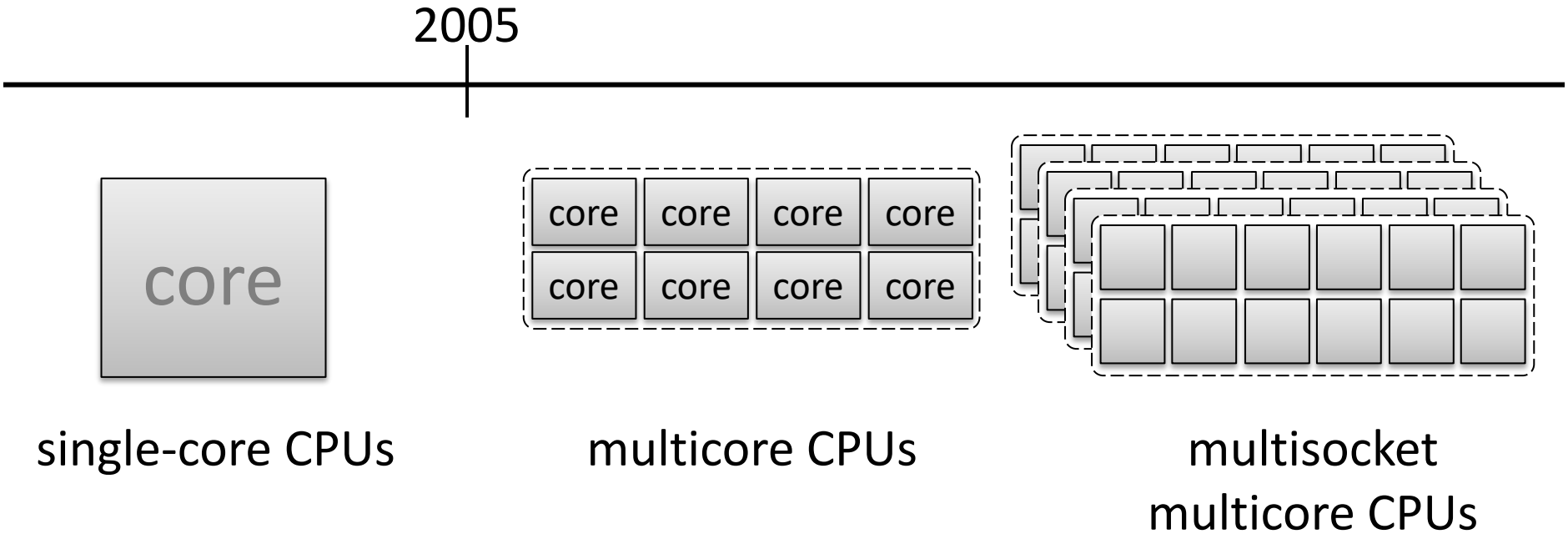
Moore's law

"… 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."

commodity CPU evolution



single-core CPUs

multicore CPUs

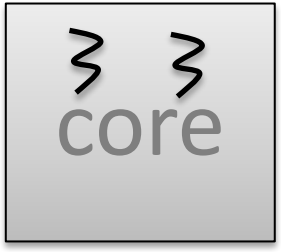
multisocket multicore CPUs

Dennard scaling doesn't hold anymore

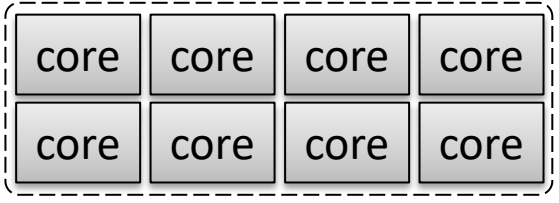
switching to multicores kept Moore's Law alive

types of hardware parallelism

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



single-core
instruction & data parallelism
simultaneous multithreading



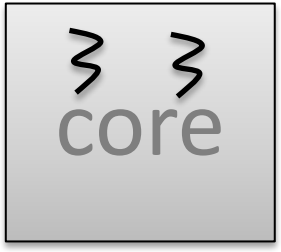
multicores
multiple threads run in
parallel on different cores

implicit parallelism → (almost) free lunch

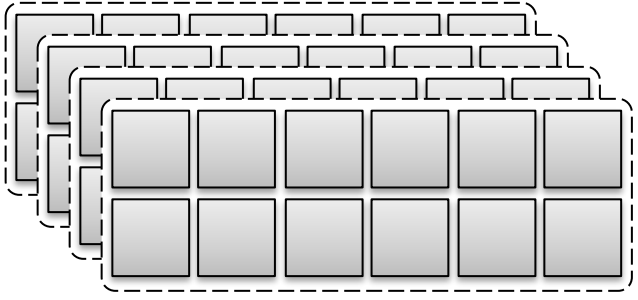
explicit parallelism → must work hard to exploit it

types of hardware parallelism

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



single-core
instruction & data parallelism
simultaneous multithreading



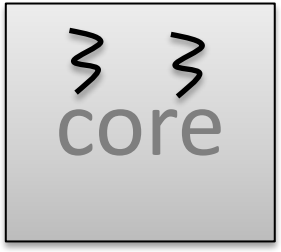
multisocket multicores
multiple processors/CPUs
in one machine

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

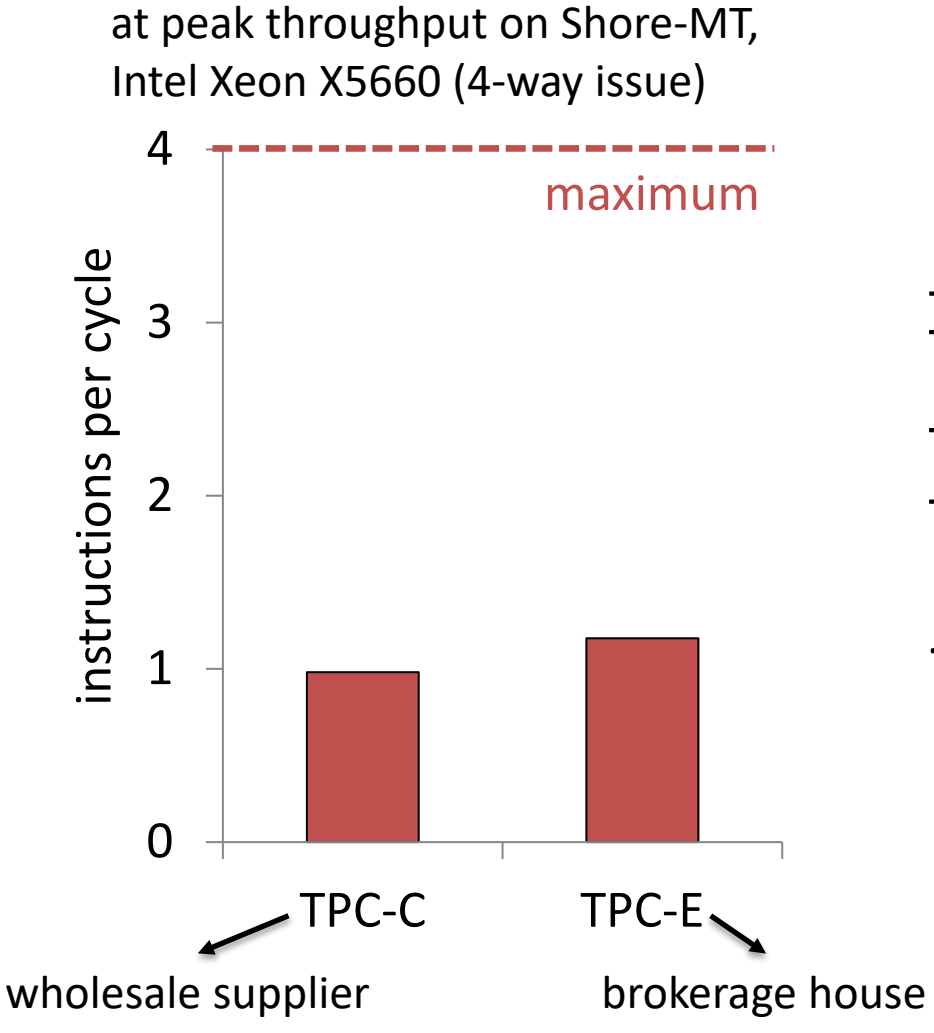
implicit parallelism → (almost) free lunch

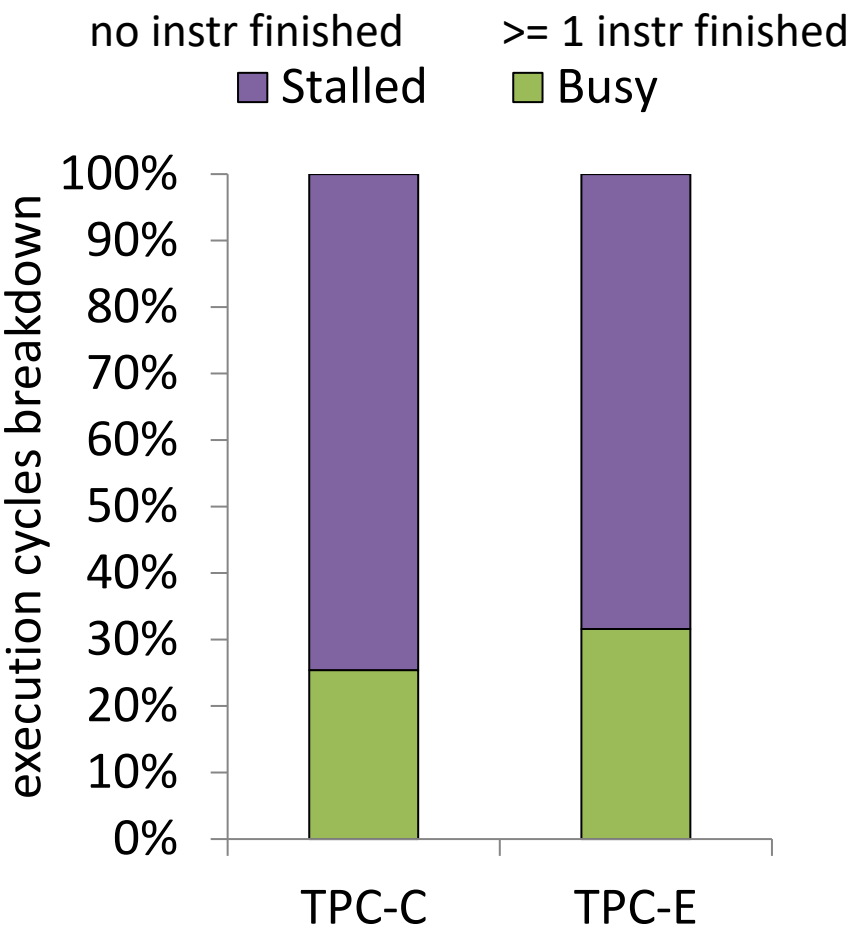
explicit parallelism → must work hard to exploit it

agenda

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

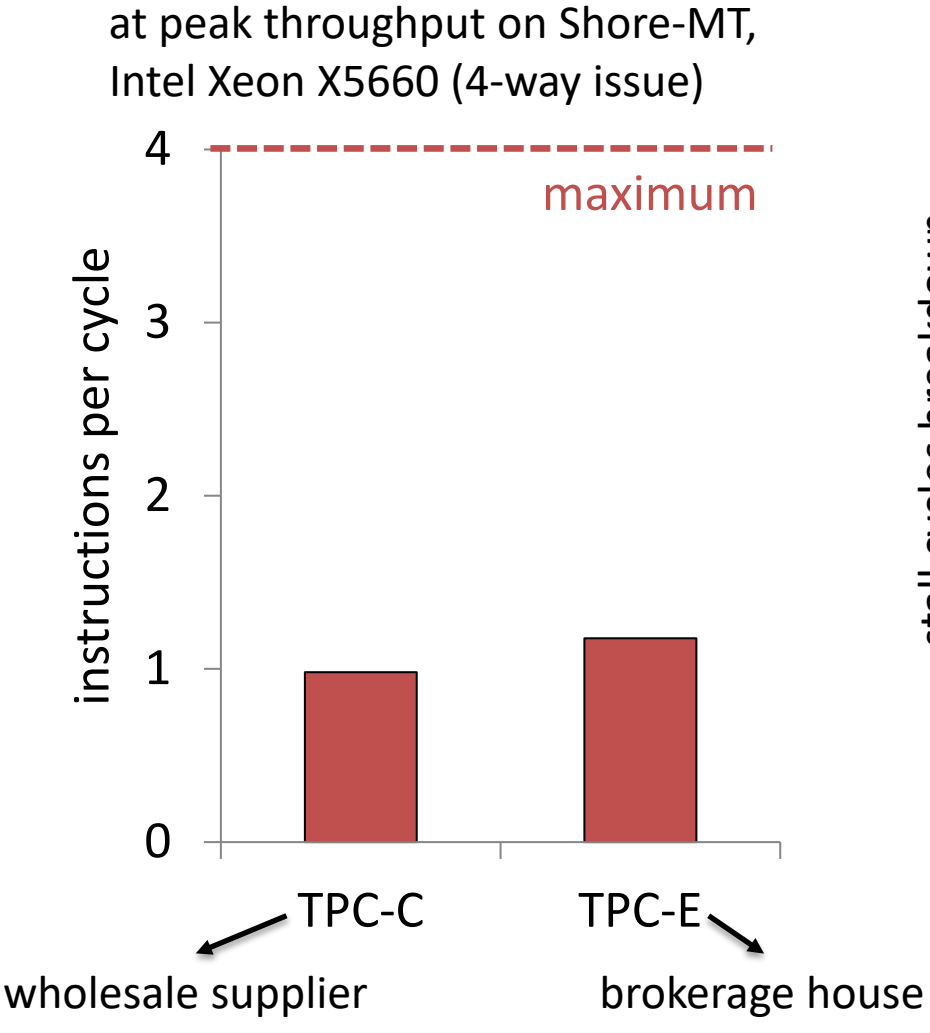
OLTP & implicit parallelism

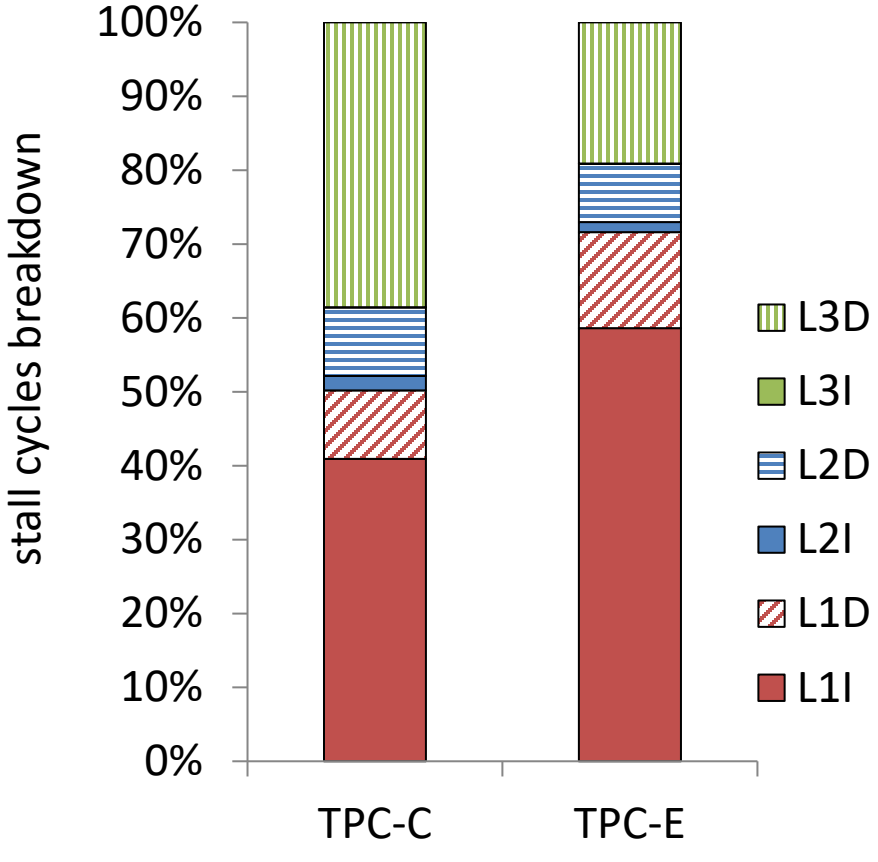




[EDBT13]

OLTP & implicit parallelism

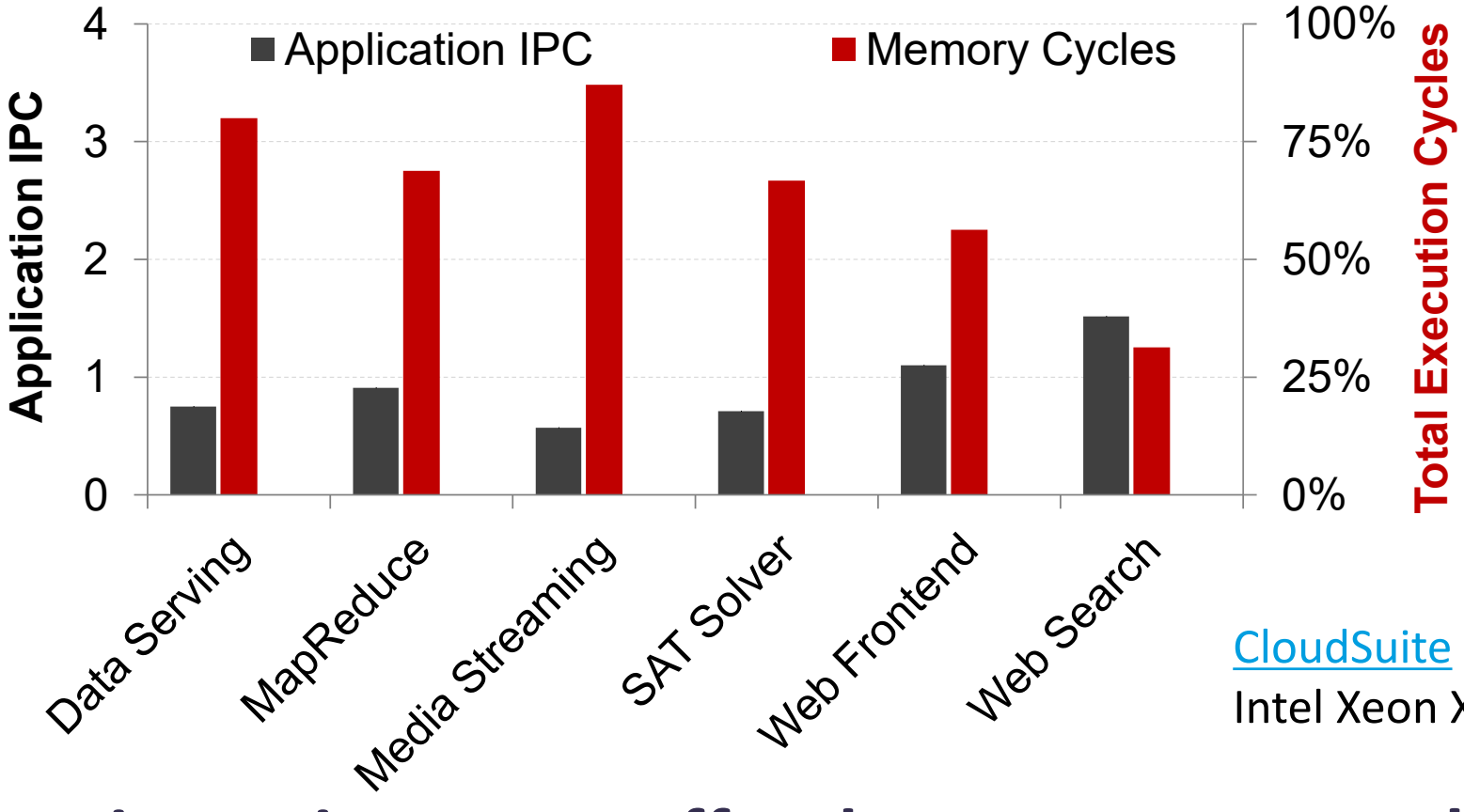




[EDBT13, DaMoN13]

memory stalls in data-intensive apps

[ASPLOS12]



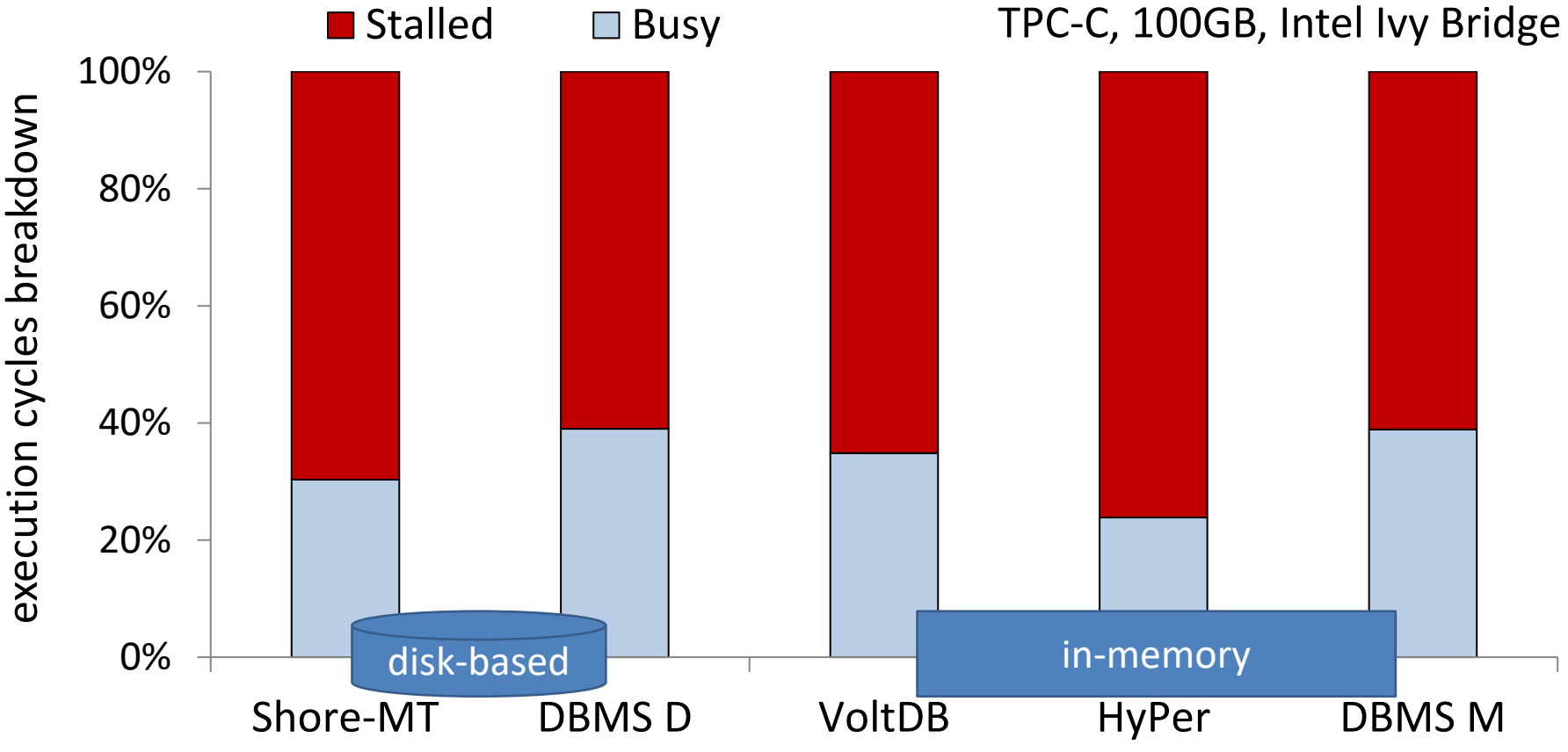
CloudSuite on Intel Xeon X5670

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

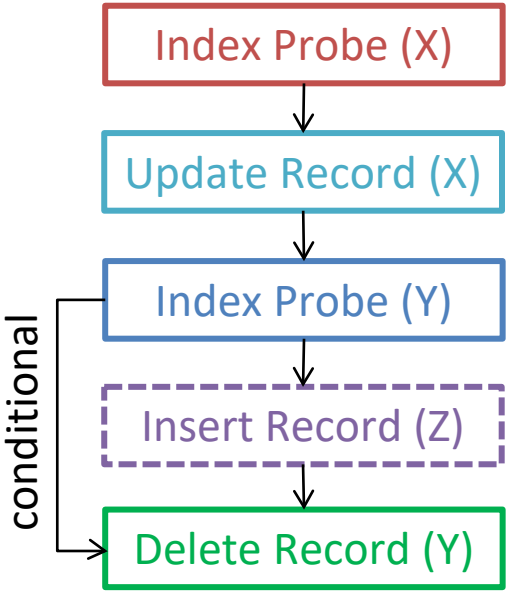
but we have room to do better

transactions under microscope

database operations

- Index Probe
- Index Scan
- Update Record
- Delete Record
- Insert Record

transaction



instances

T1

- Index Probe (X1)
- Update Record (X1)
- Index Probe (Y1)
- Delete Record(Y1)

T2

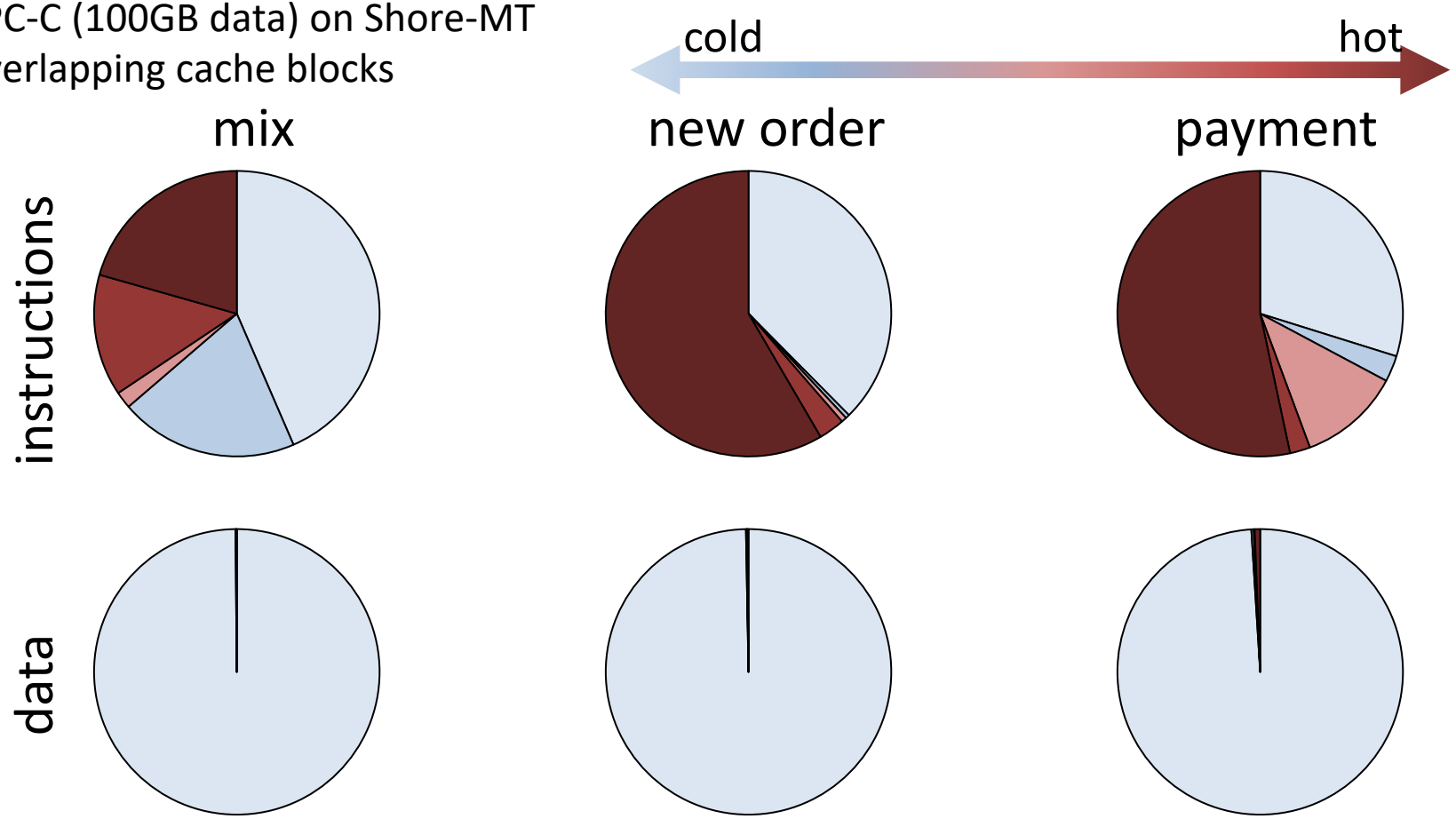
- Index Probe (X2)
- Update Record (X2)
- Index Probe (Y2)
- Insert Record(Z2)
- Delete Record(Y2)

many transactions are composed of common instructions

instruction & data overlap

[PVLDB14]

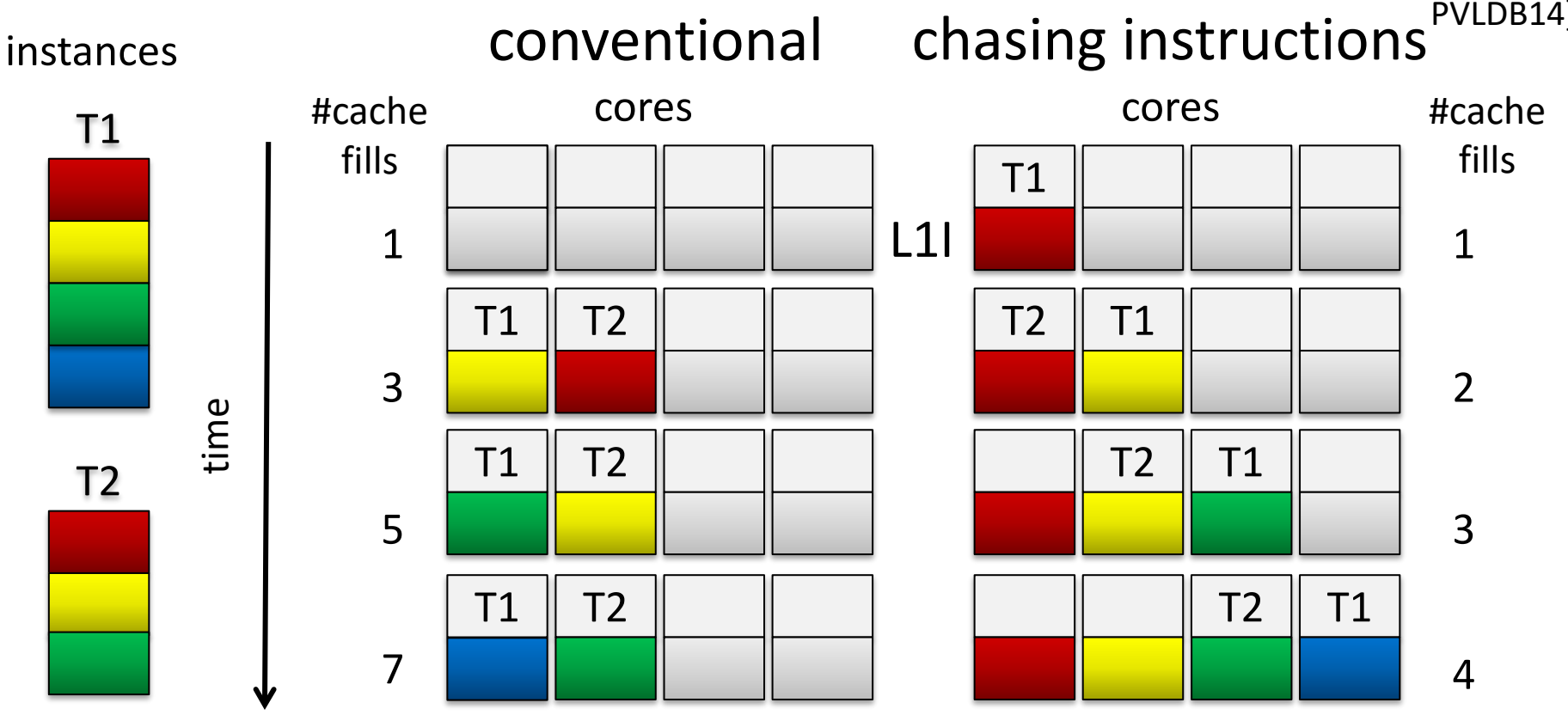
TPC-C (100GB data) on Shore-MT
overlapping cache blocks



high for instructions, low for data

utilizing 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

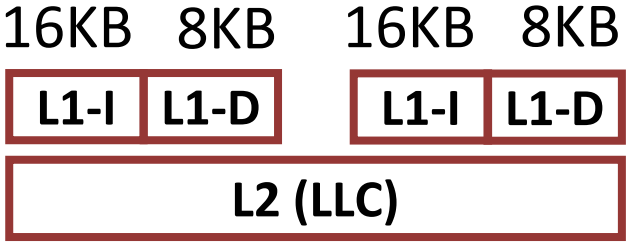
- implicit parallelism isn't completely free lunch
- > 50% of cycles are stalls for traditional OLTP
  - L1-I misses are significant
- invest in
  - utilizing instruction overlap across transactions & aggregate L1-I cache capacity
  - simplified code, cache-friendly data/code layouts, smarter query compilation …

interlude: SUN SPARC

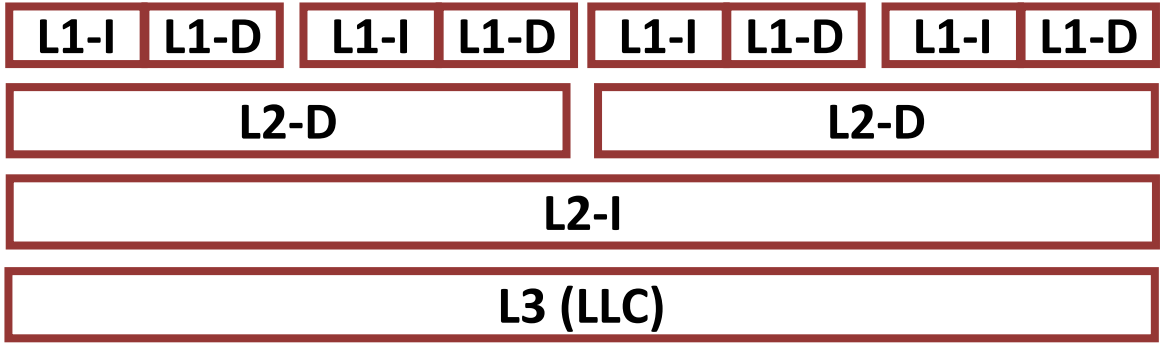
OLTP instructions have

1. **large footprint**
2. **high overlap**

UltraSPARC T2 (Niagara 2) 2007



SPARC M7 & M8



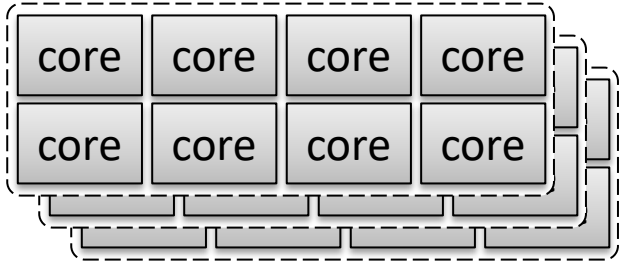
2017

agenda

- types of hardware parallelism
- OLTP & implicit parallelism
- **OLTP & 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

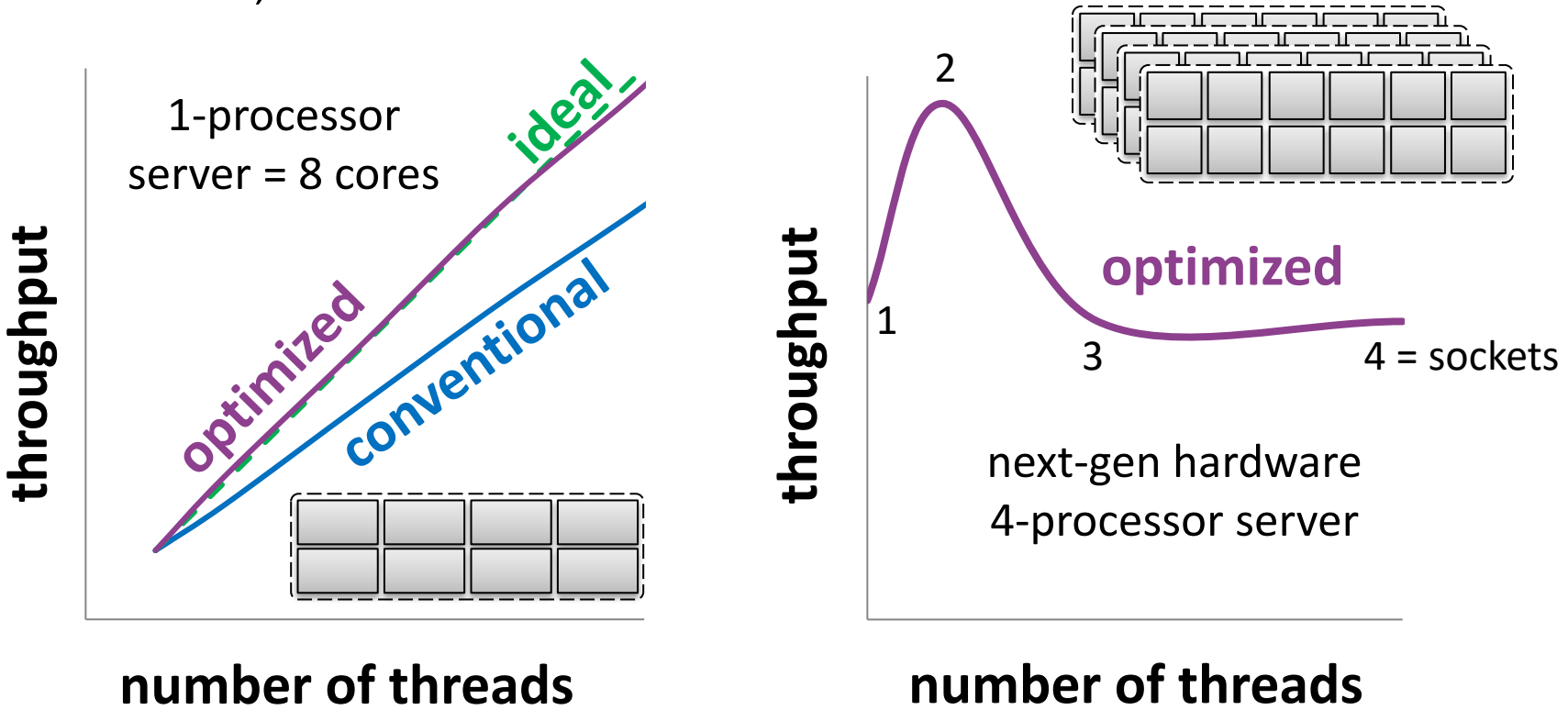
adding more data centers should give proportional performance increase

for google, amazon …!

scaling-up

[PVLDB11, PVLDB12, ICDE14]

probe one customer, read balance on Shore-MT

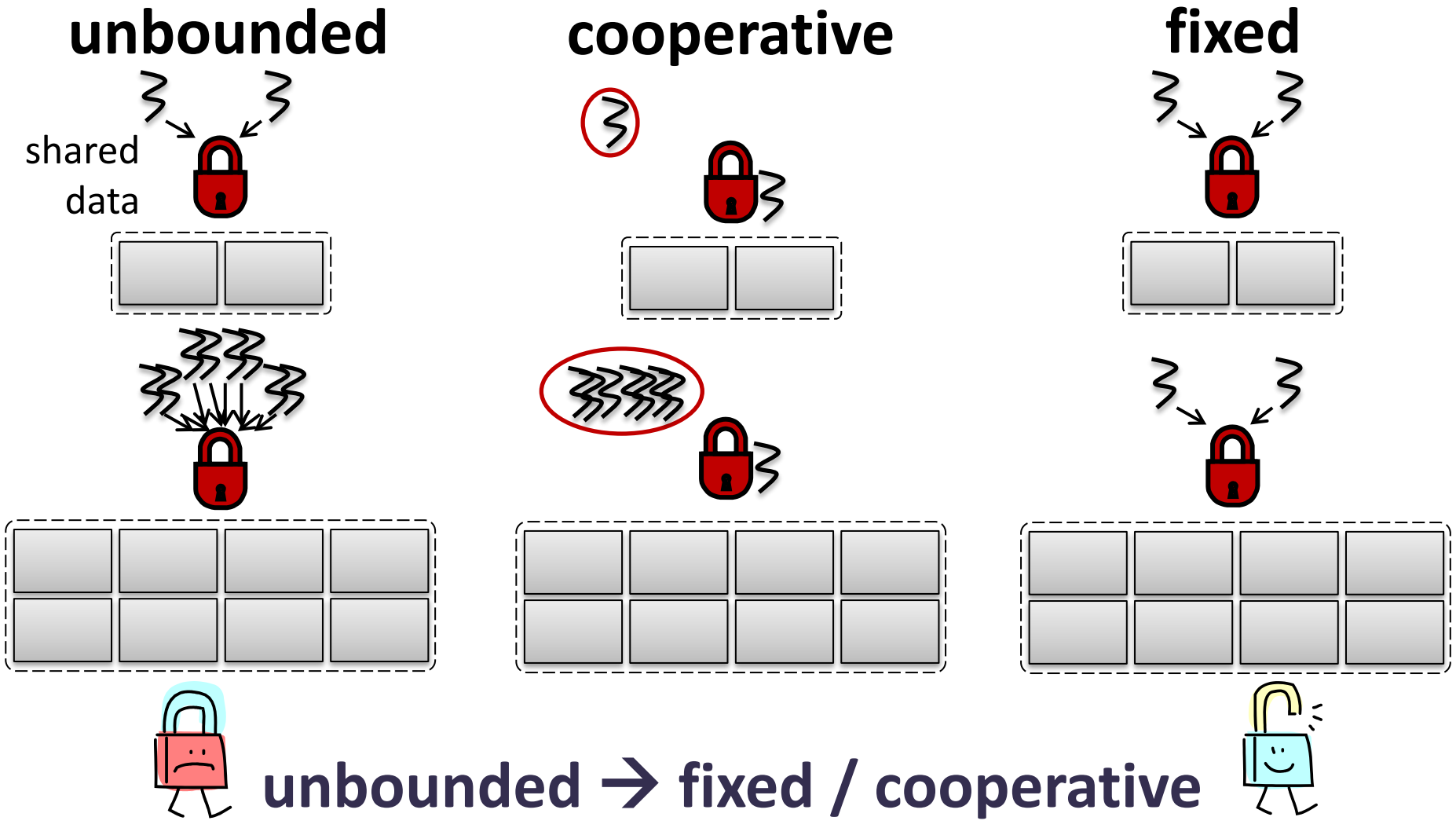


need better metrics to reason about scalability

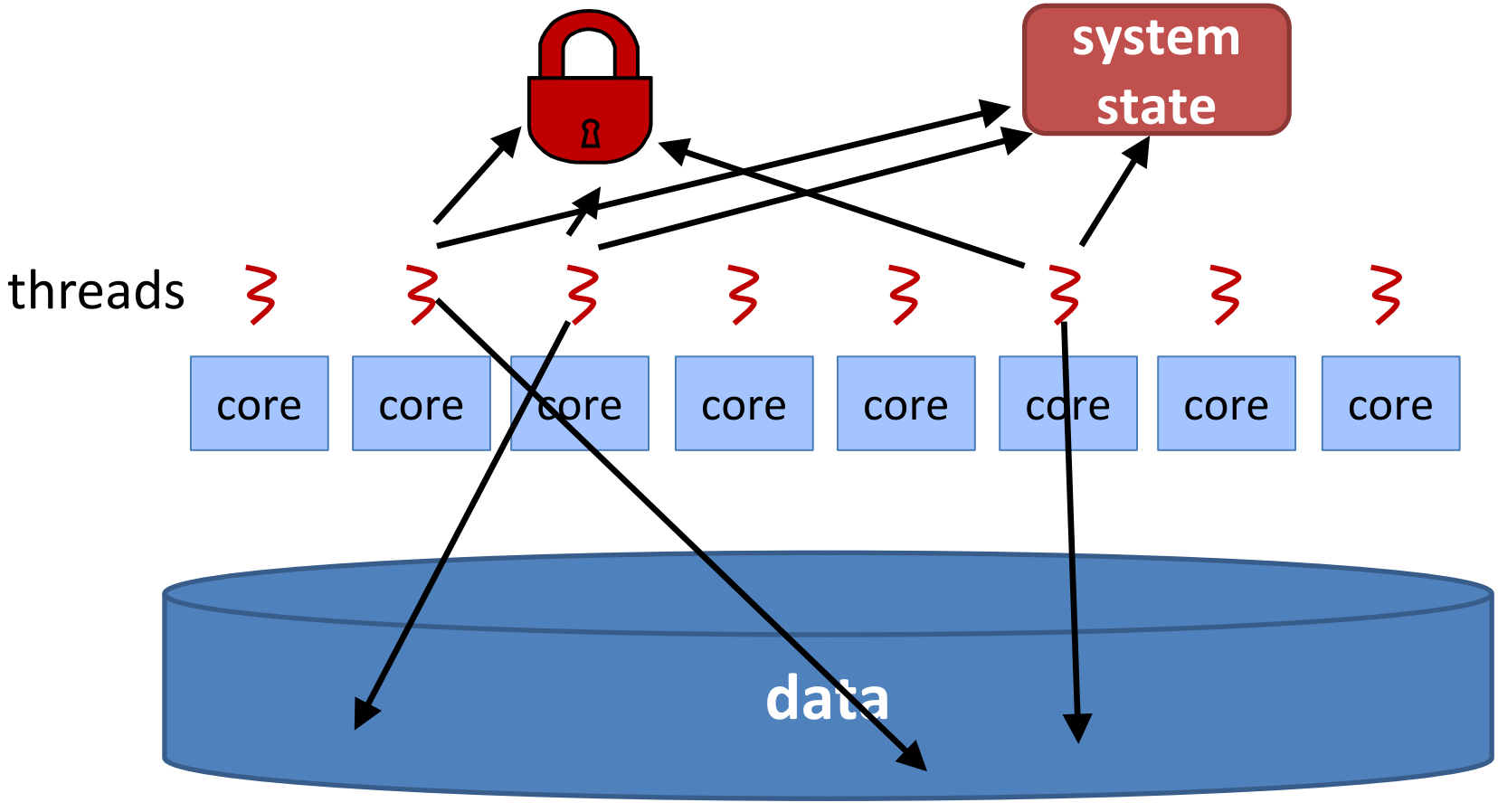
throughput measurements are not enough

critical sections / synchronization

[PVLDB11]

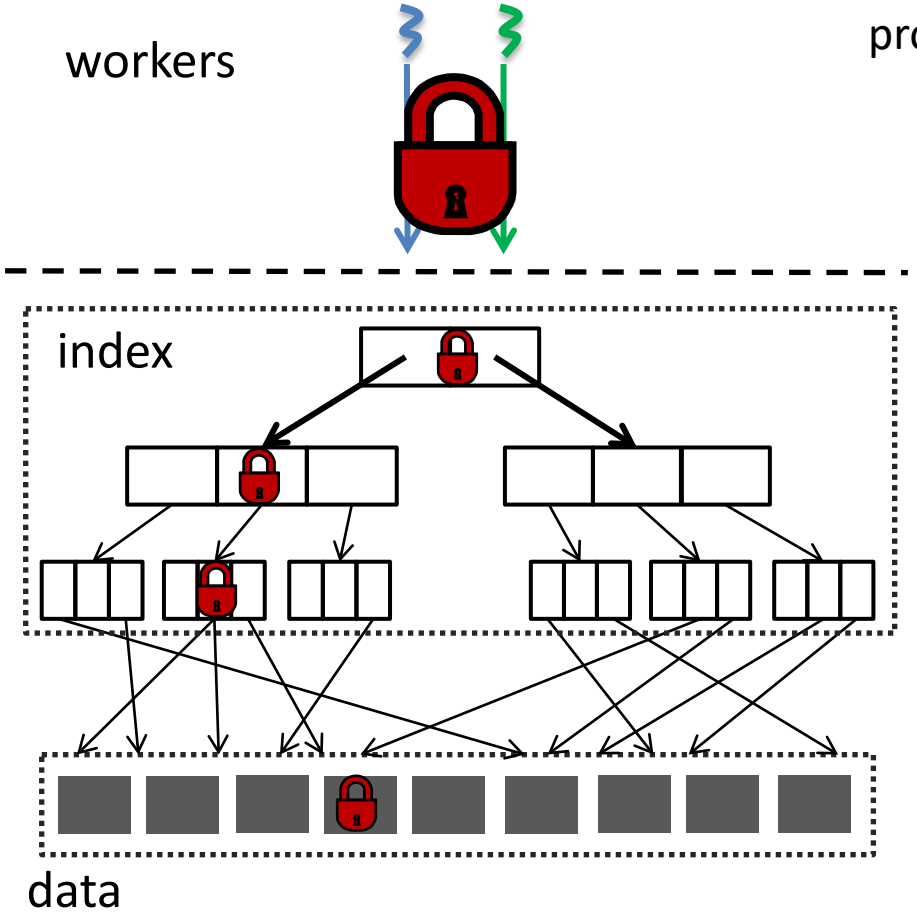


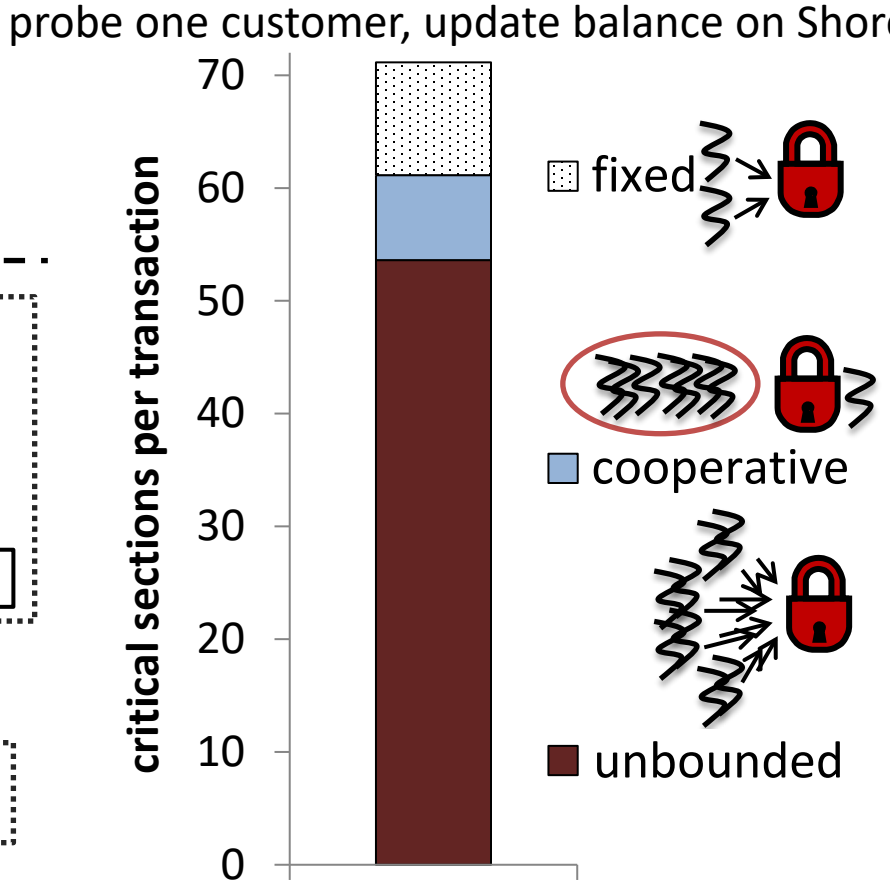
critical path of transaction execution



many unpredictable accesses to shared data

impact of unpredictable data accesses

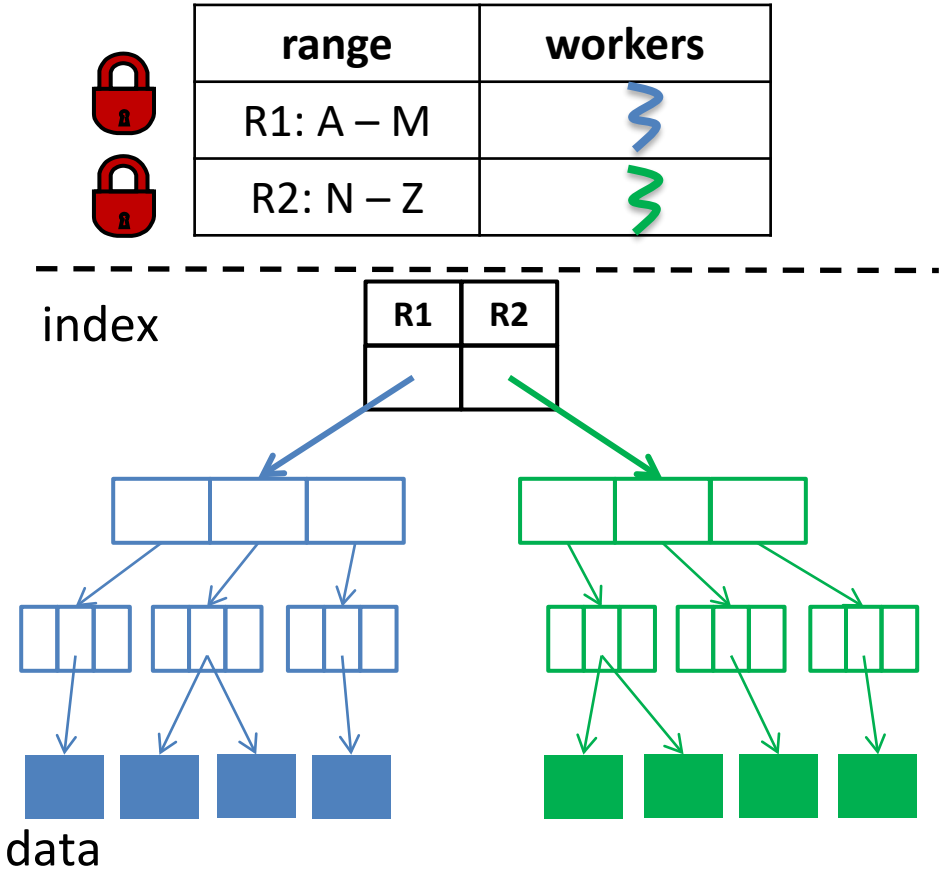


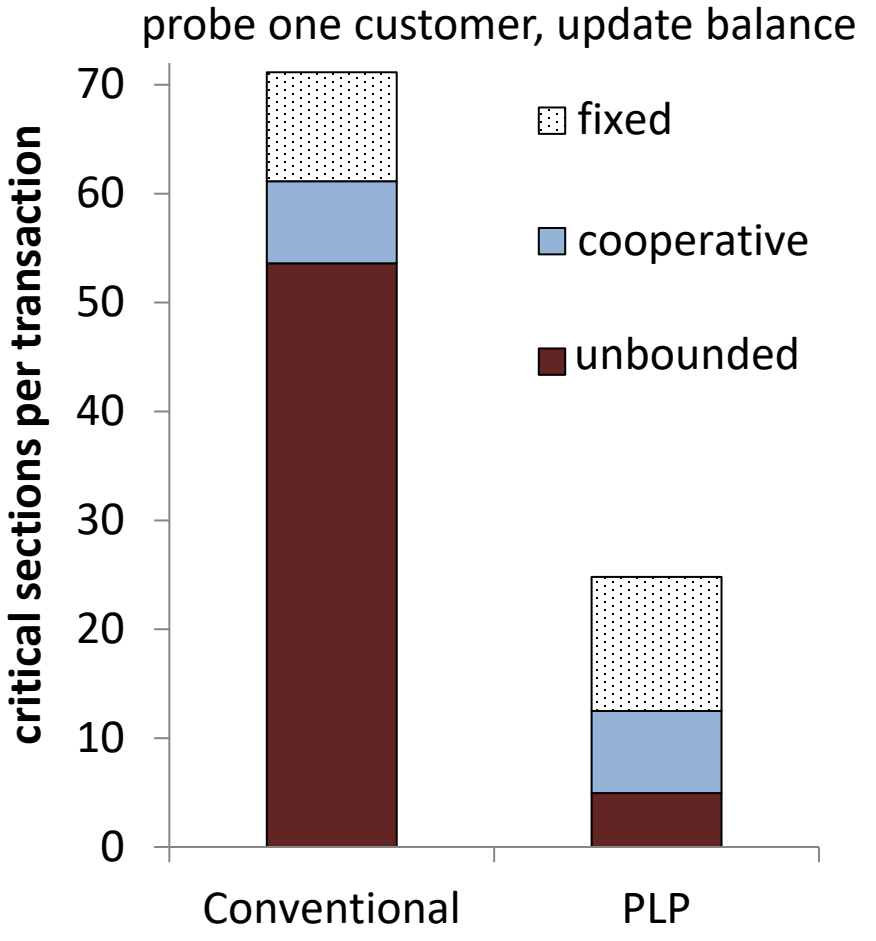


75% of critical sections are unbounded

physiological partitioning (PLP)

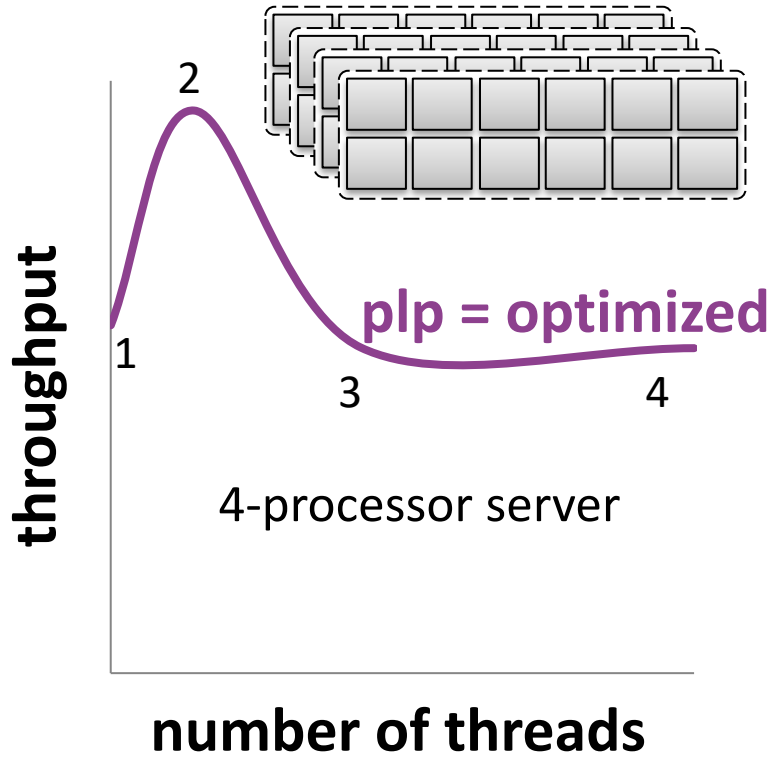
[PVLDB11]

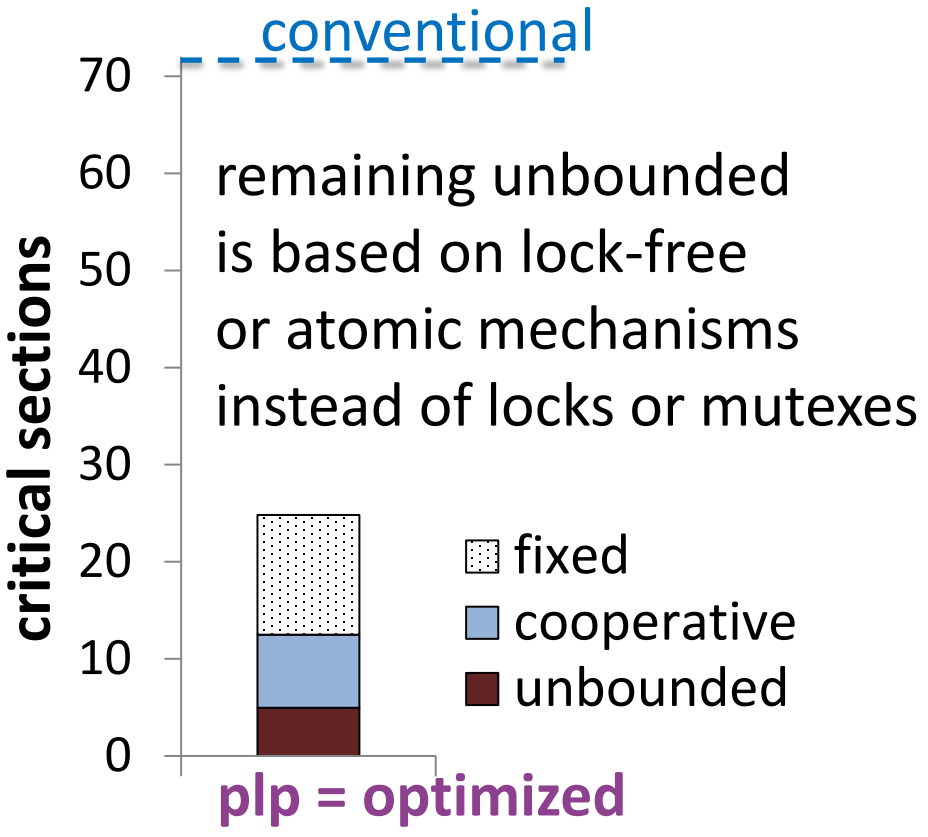




PLP eliminates 70% of critical sections

critical sections as a metric?

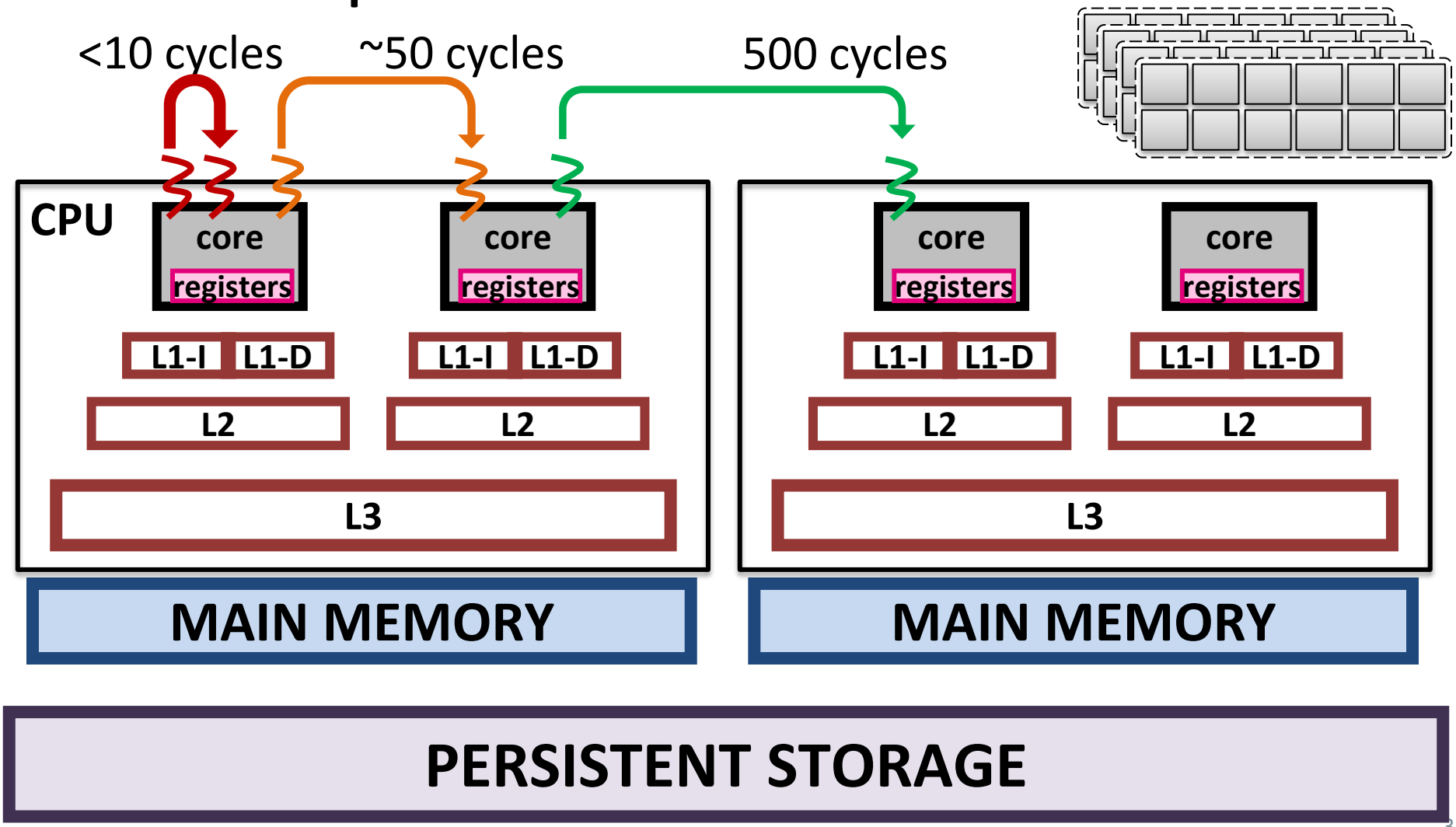




unbounded communication will hit you eventually

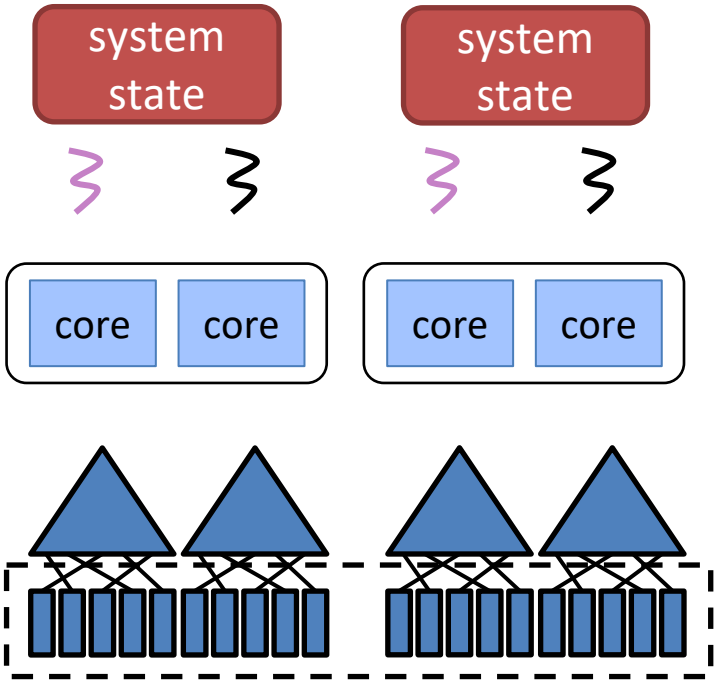
with NUMA even fixed/cooperative have issues

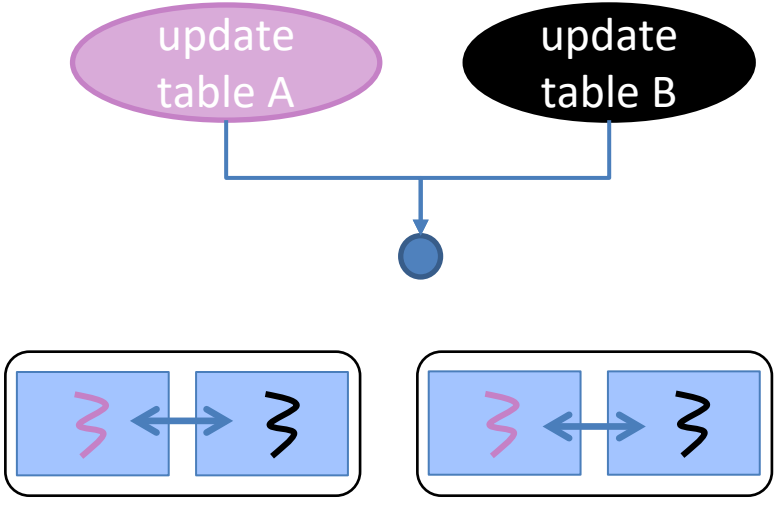
NUMA impact


<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]

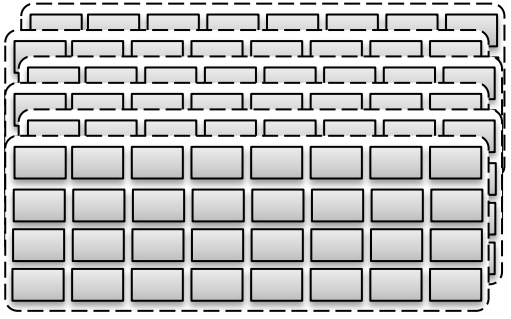




limit unbounded communication within a socket

keep access latencies predictable

summary: OLTP & explicit parallelism

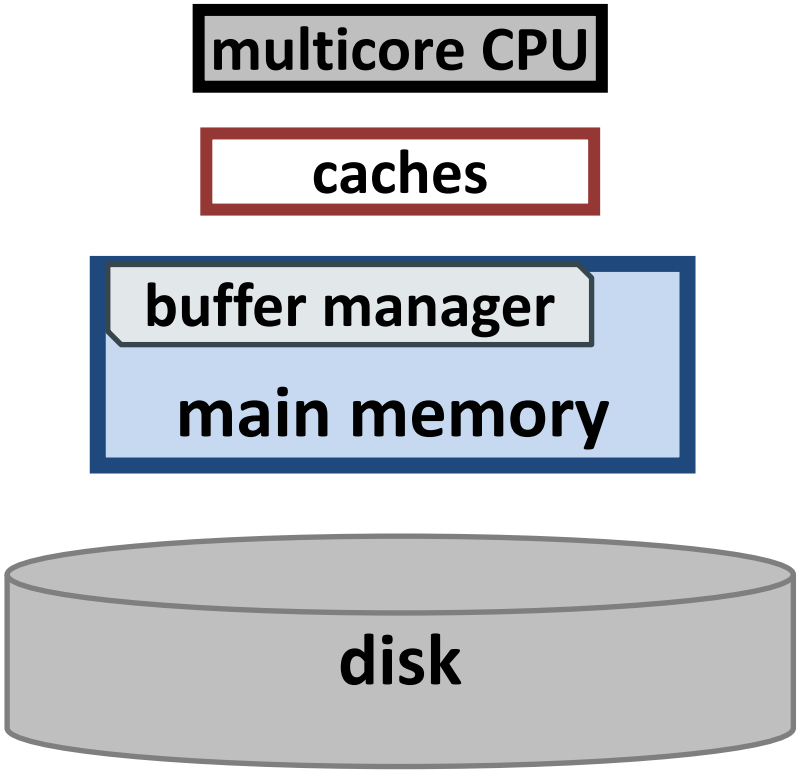


- **high throughput != scalable**
- **lock freedom != scalable**
- eliminate any unbounded communication
  - or at least bound it within a socket
- keep fixed/cooperative communication among cores with similar/predictable access latency

  → avoid sharing data across different processors (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 accesses

- no / minimal disk use during transactions
- lightweight logging & replication for recovery
- optimize for PMem & SSDs instead

references / credits for the slides

slide 30 & 34-35 are Danica Porobic's slides from her ICDE14 talk

[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.