ORACLE®

Big graphs on big machines

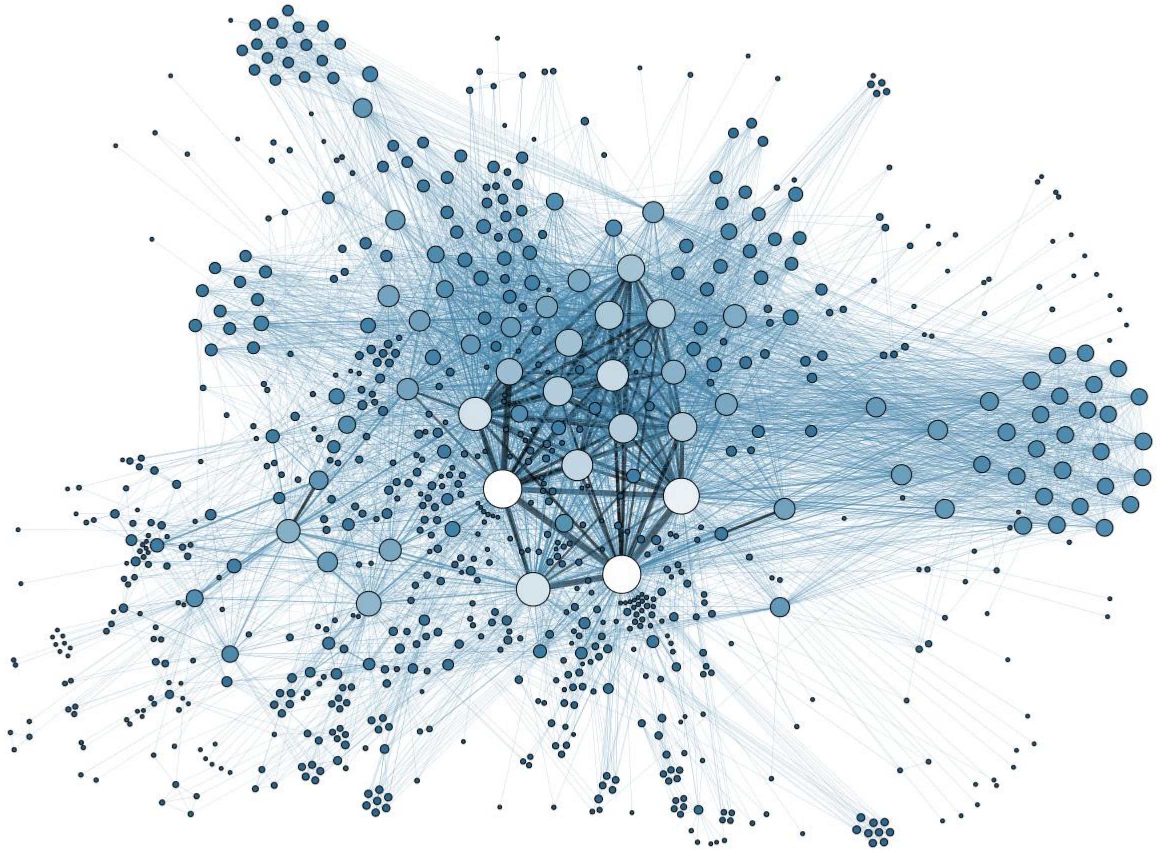
Tim Harris

13 March 2017

ORACLE

The following is intended to provide some insight into a line of research in Oracle Labs. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. Oracle reserves the right to alter its development plans and practices at any time, and the development, release, and timing of any features or functionality described in connection with any Oracle product or service remains at the sole discretion of Oracle. Any views expressed in this presentation are my own and do not necessarily reflect the views of Oracle.

Graph analytics workloads



Large data sizes

- Click-stream data
- IoT
- TB+ benchmark inputs

Abundant parallelism

- Process vertices concurrently

Complex access patterns

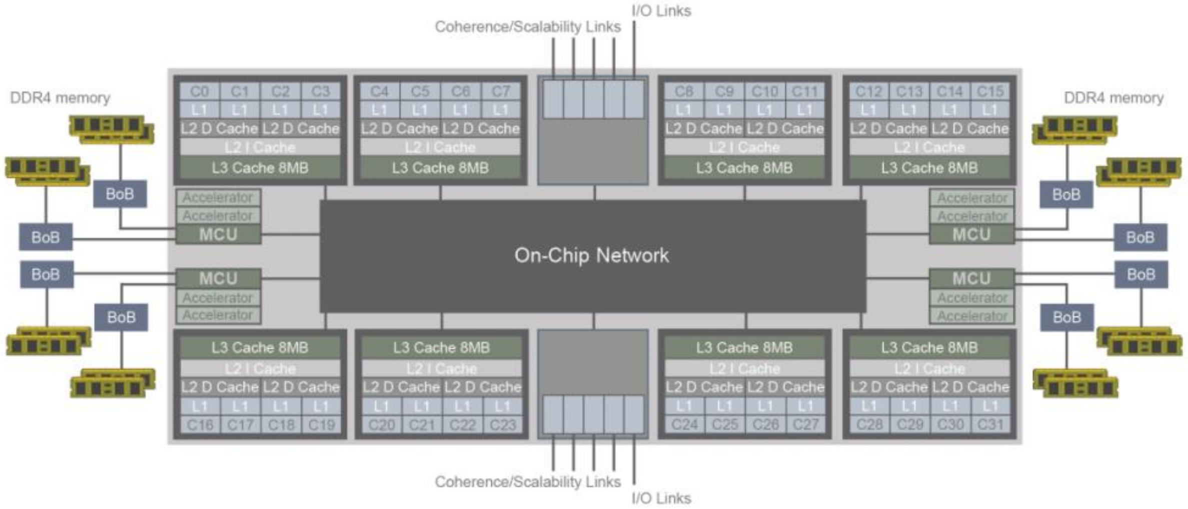
- Input dependent
- Low-diameter inputs
- => no effective partitioning

https://upload.wikimedia.org/wikipedia/commons/9/9b/Social_Network_Analysis_Visualization.png

SPARC M7-16

16 sockets
32 cores per socket
8 h/w contexts per core
=> 4096 h/w contexts

8 TB DRAM
512 GB installed per socket
16 * 32 GB DIMMs per socket
16 DRAM channels



In-memory graph analytics

Domain specific languages

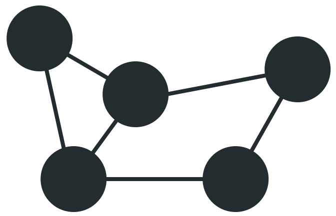
- Queries expressed in terms of graph concepts
- Tailor for different kinds of workload (e.g., sub-graph isomorphism)

Generated code

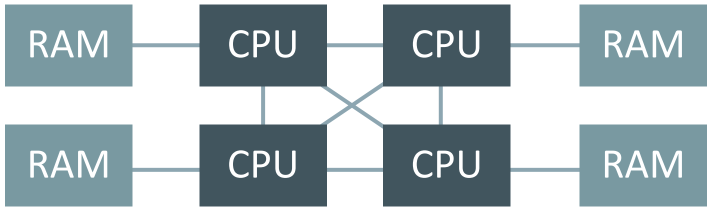
- Efficient in-memory data representations, e.g. compressed-sparse-row format
- Abundant parallelism

Runtime system

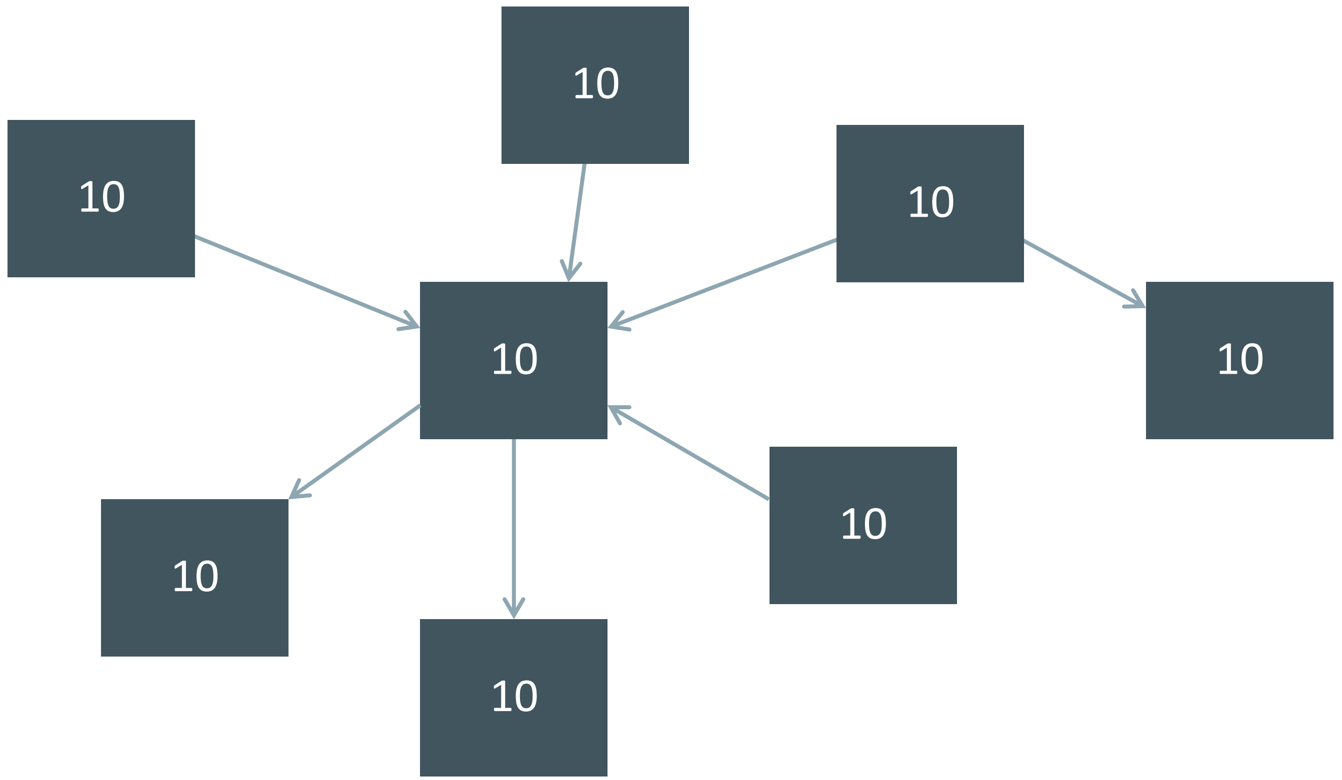
- Allocation of resources to a query
- Distribution of work and data within a machine



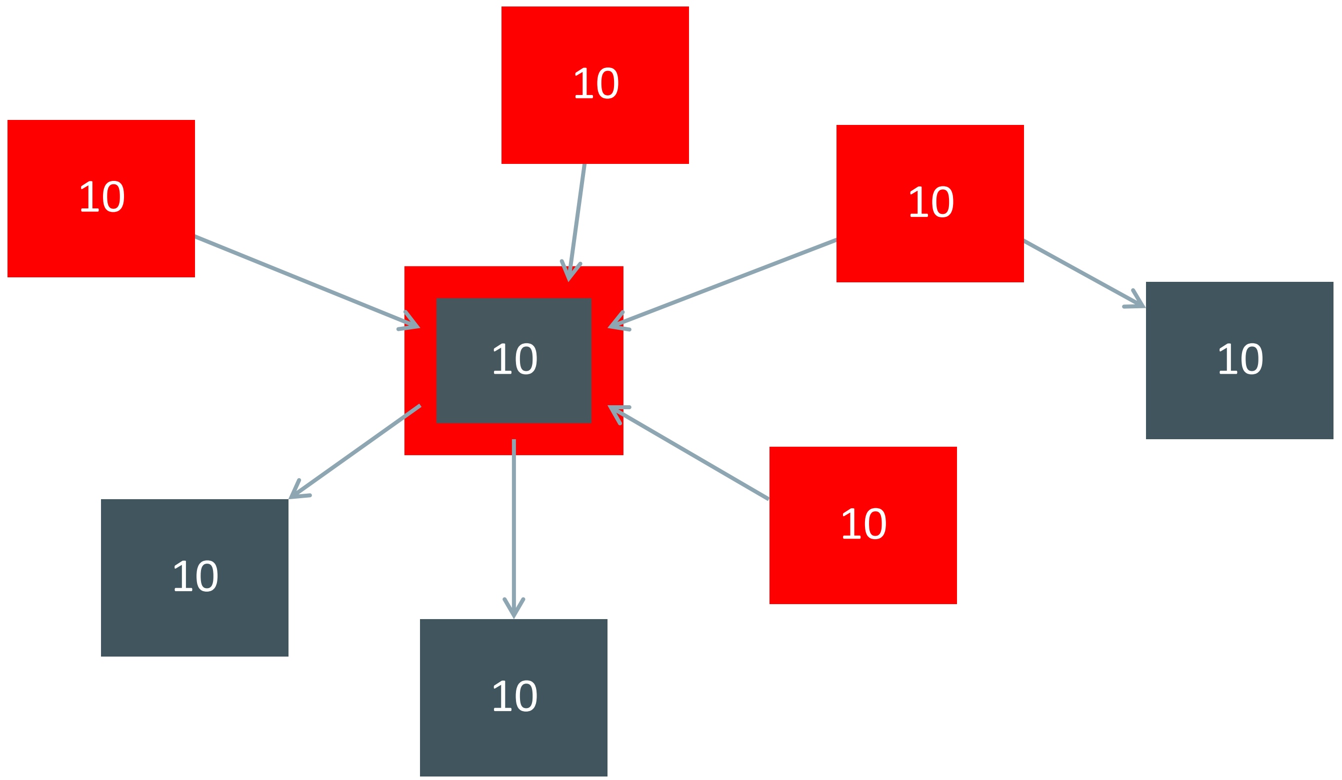
```
parallel_for<node_t>([&](node_t n) {
    …
});
```



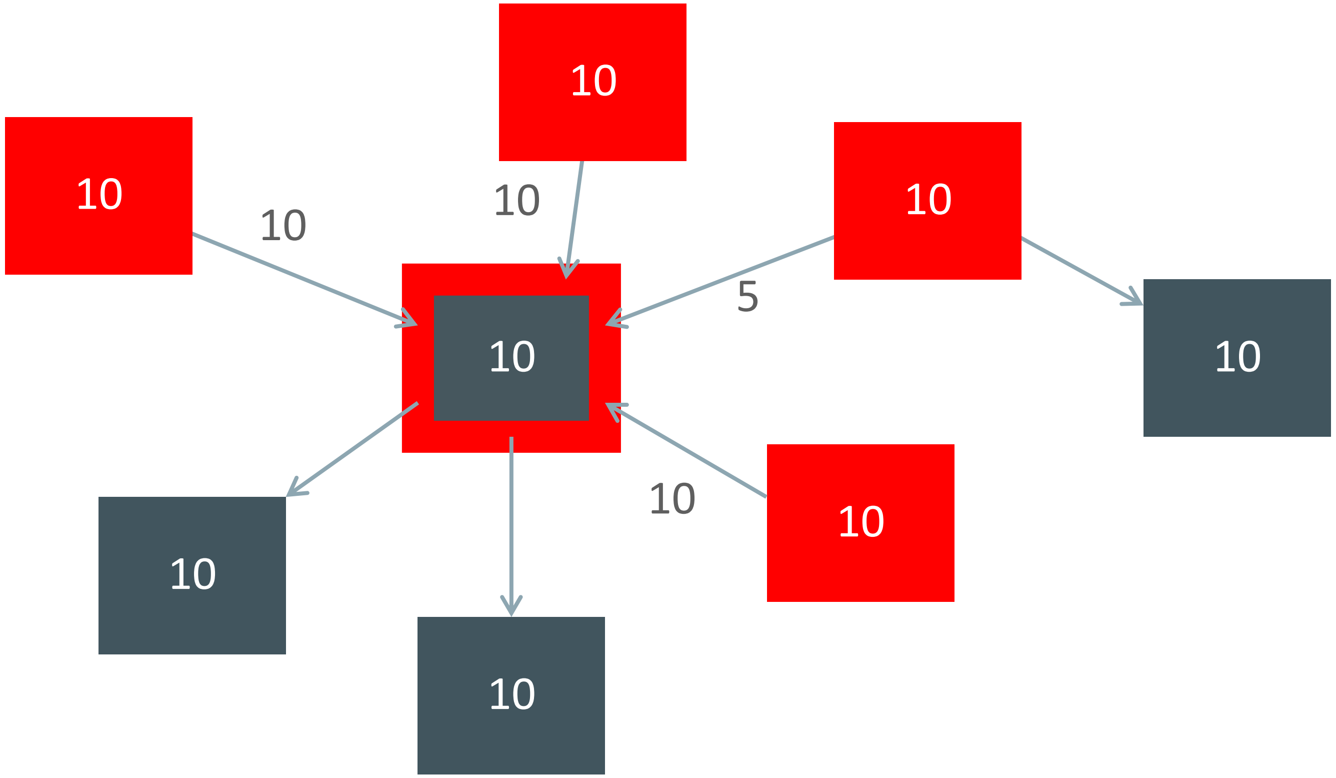
PageRank inner loop



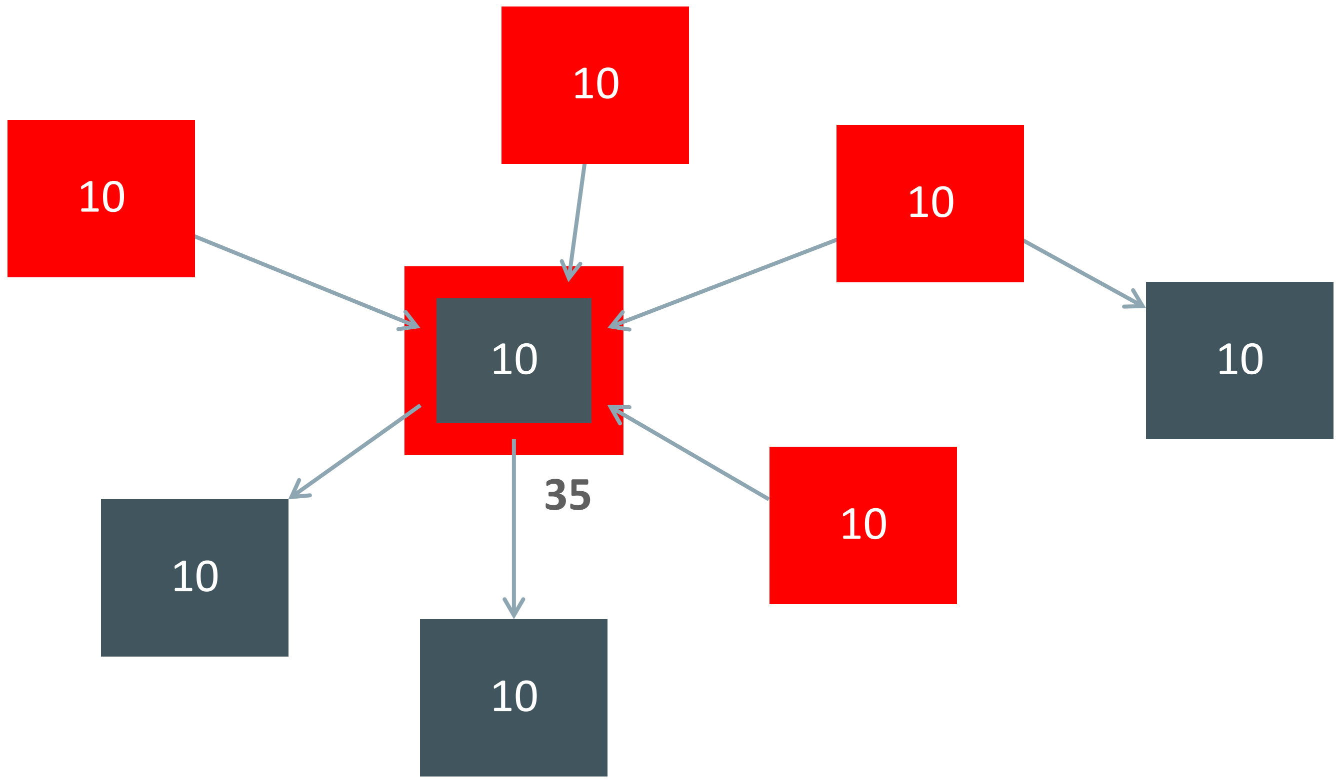
PageRank inner loop



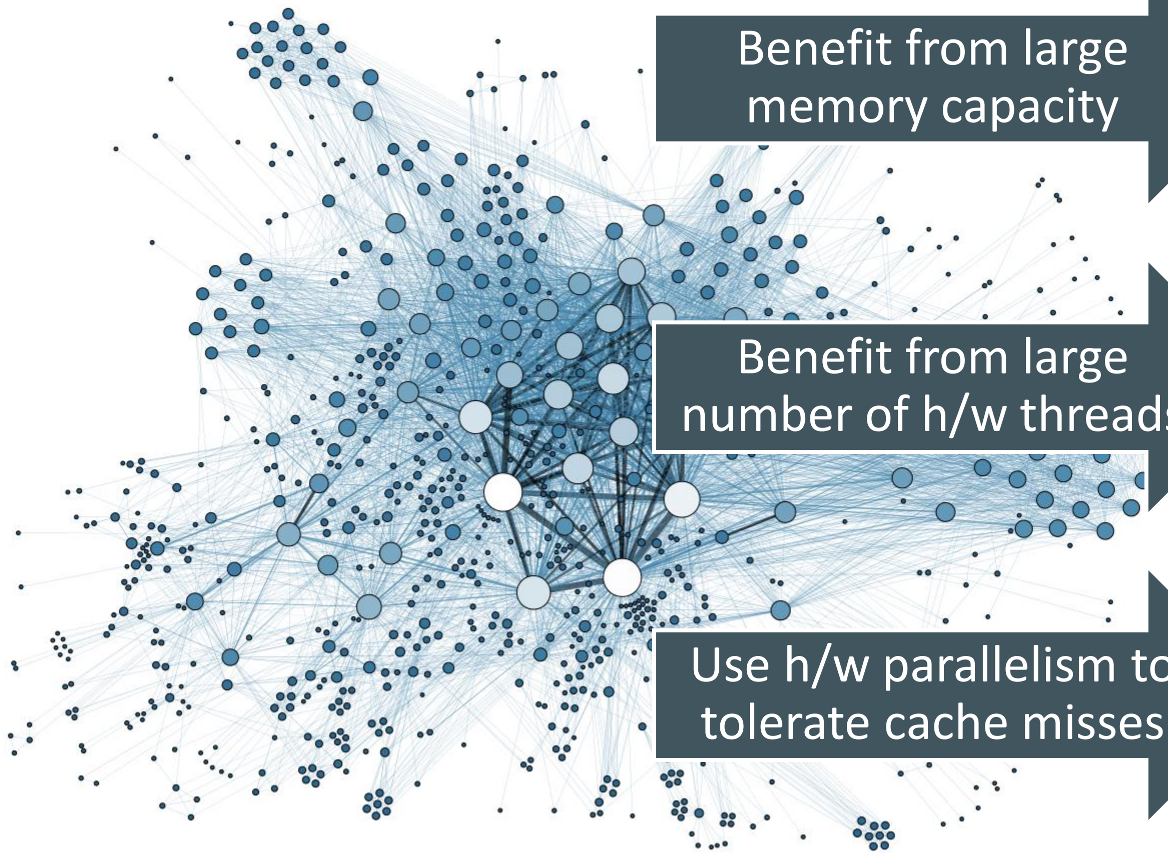
PageRank inner loop



PageRank inner loop



Graph analytics workloads



Benefit from large memory capacity

Benefit from large number of h/w threads

Use h/w parallelism to tolerate cache misses

Large data sizes

- Click-stream data
- IoT
- TB+ benchmark inputs

Abundant parallelism

- Process vertices concurrently

Complex access patterns

- Input dependent
- Low-diameter inputs
- => no effective partitioning

https://upload.wikimedia.org/wikipedia/commons/9/9b/Social_Network_Analysis_Visualization.png

Case studies

1 Distributing parallel work

2 Memory allocation

3 Observations

Batch size / load imbalance trade-off

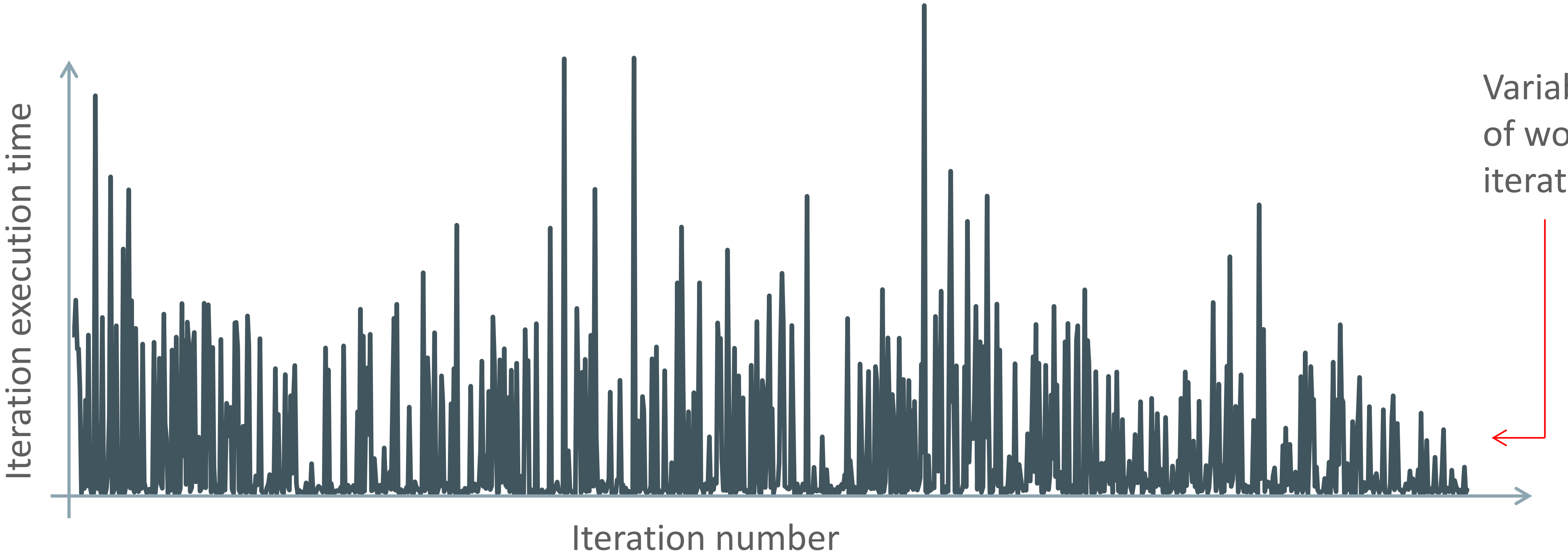
Divide into large batches of vertices

Reduce overheads
Risk load imbalance

Divide into small batches of vertices

Increase overheads distributing work
Achieve better load balance

Batch size / load imbalance trade-off

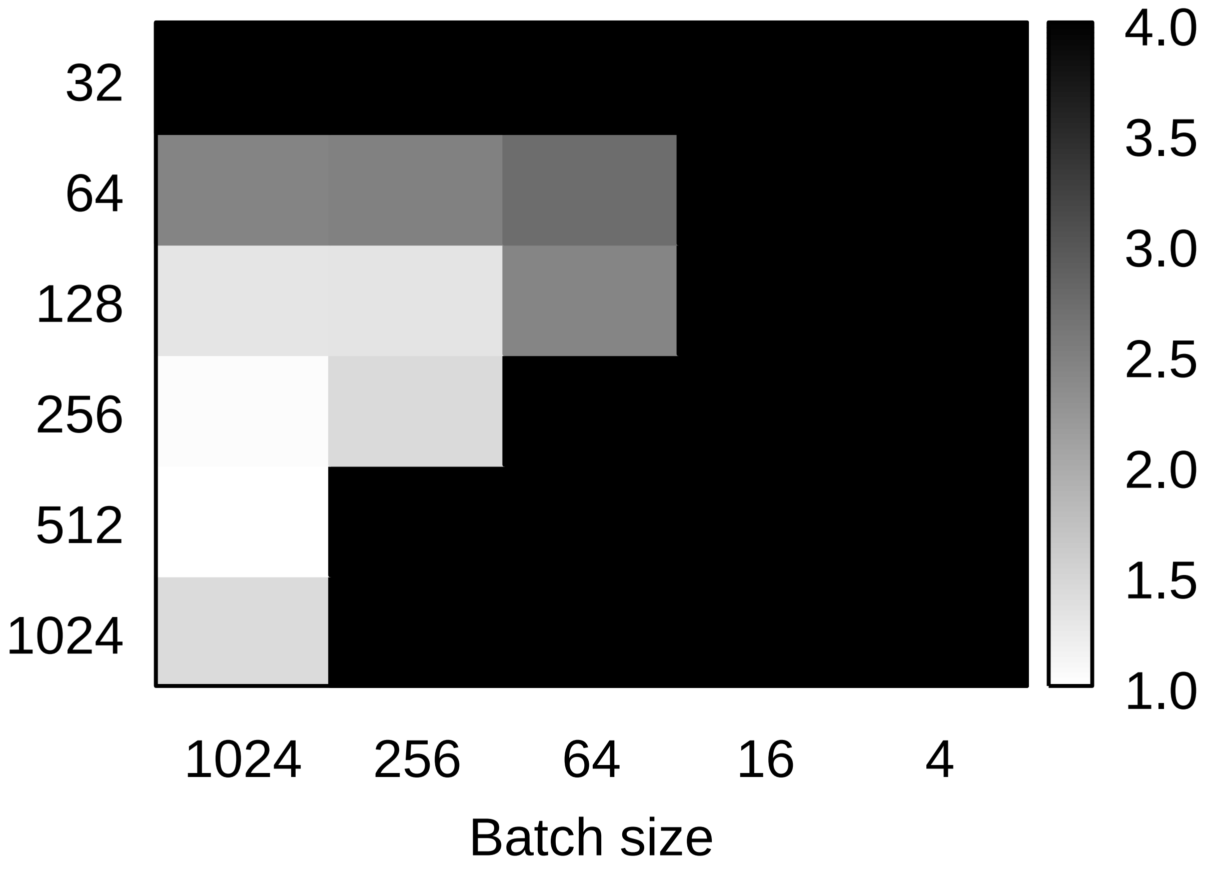


Variable amount of work per iteration

(Actual data – #out-edges of the top 1000 nodes in the SNAP Twitter dataset)

Example performance

Complete PageRank execution, SNAP LiveJournal data set

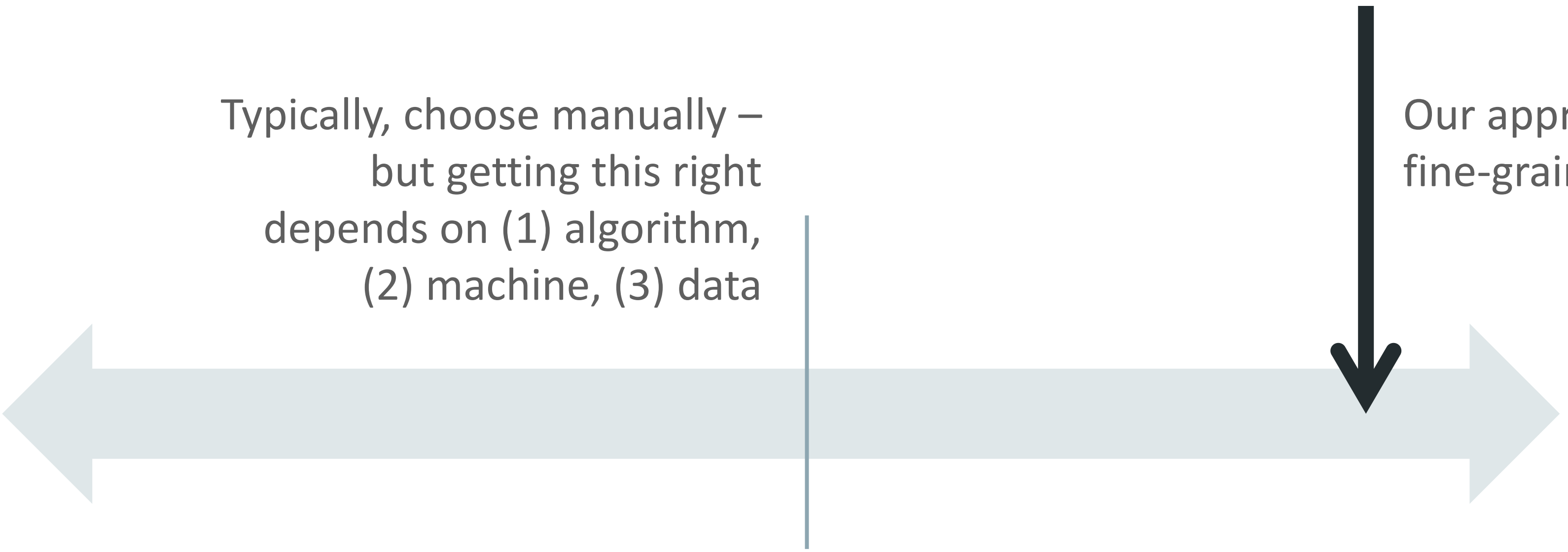


8-socket SPARC T5
16 cores per socket
8 h/w threads per core

Batch size / load imbalance trade-off

Typically, choose manually – but getting this right depends on (1) algorithm, (2) machine, (3) data

Our approach: efficient fine-grained distribution



Divide into large batches of vertices

Reduce overheads
Risk load imbalance

Divide into small batches of vertices

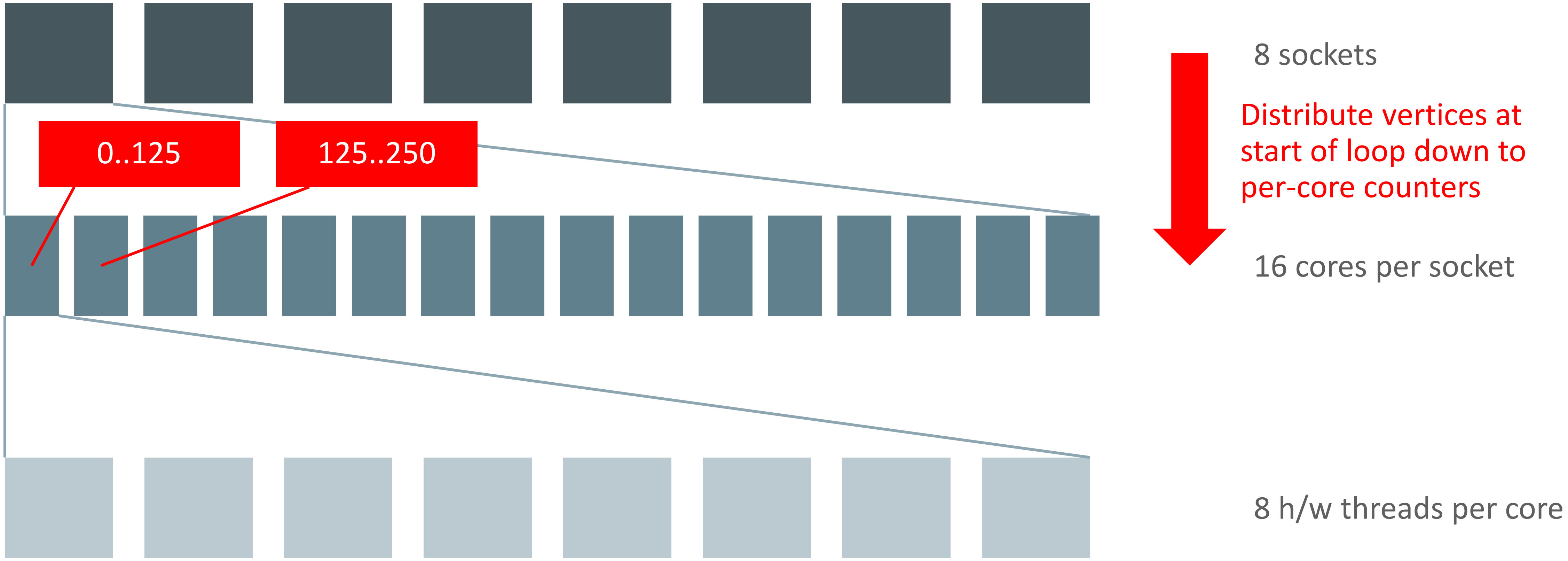
Increase overheads distributing work
Achieve better load balance

ORACLE

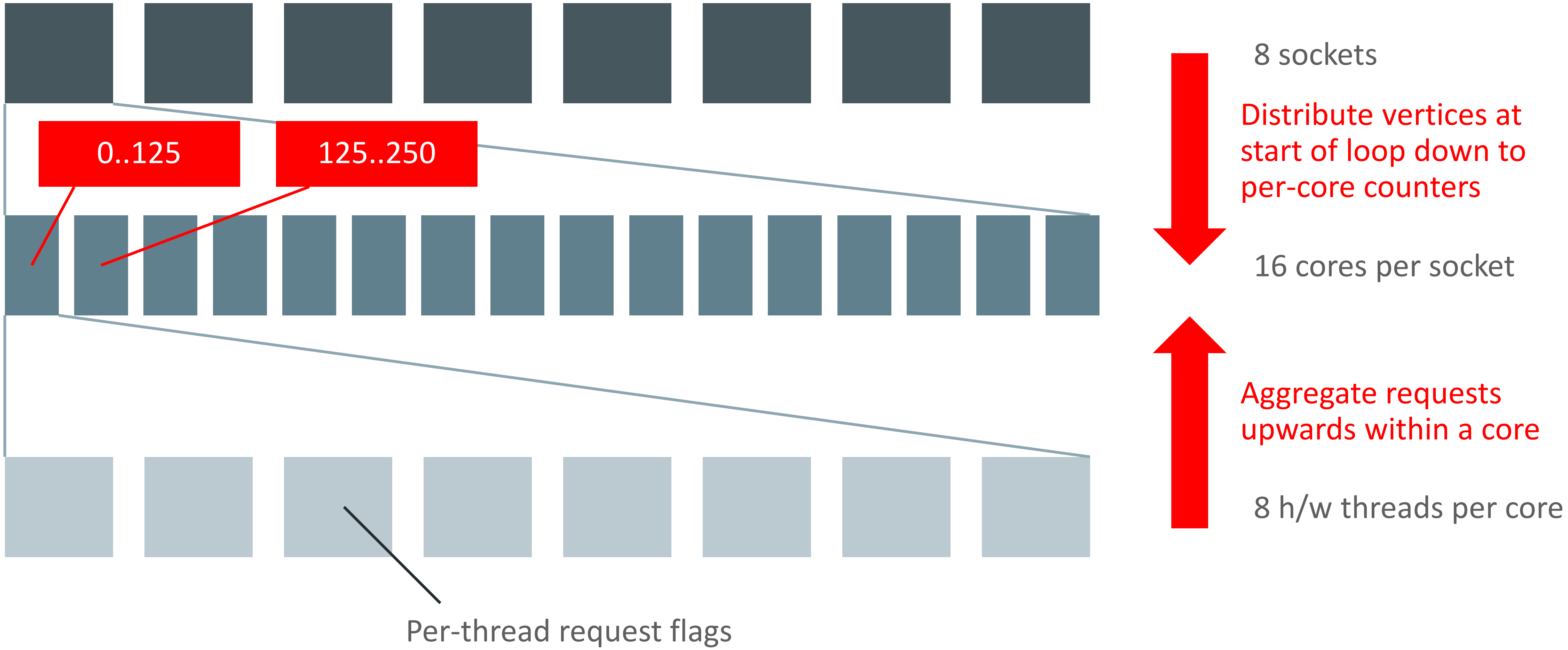
Consider distributing 0..16000 vertices, batch size 10

8 sockets

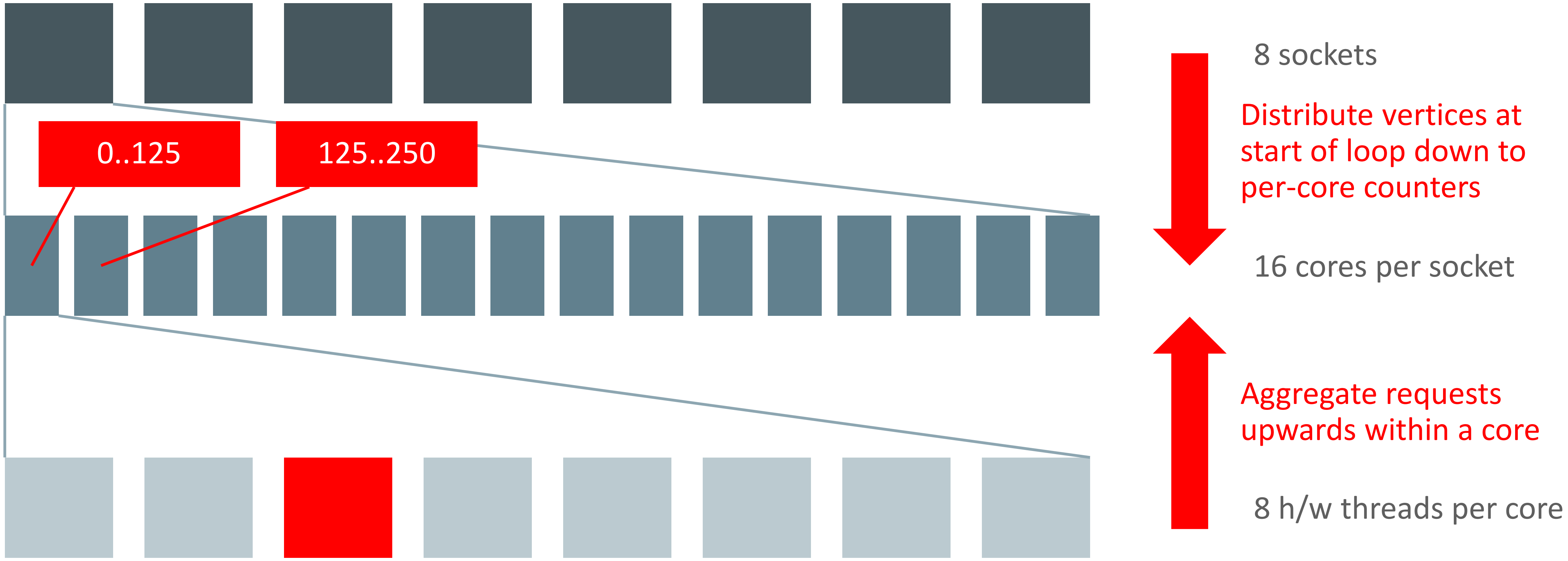
Consider distributing 0..16000 vertices, batch size 10



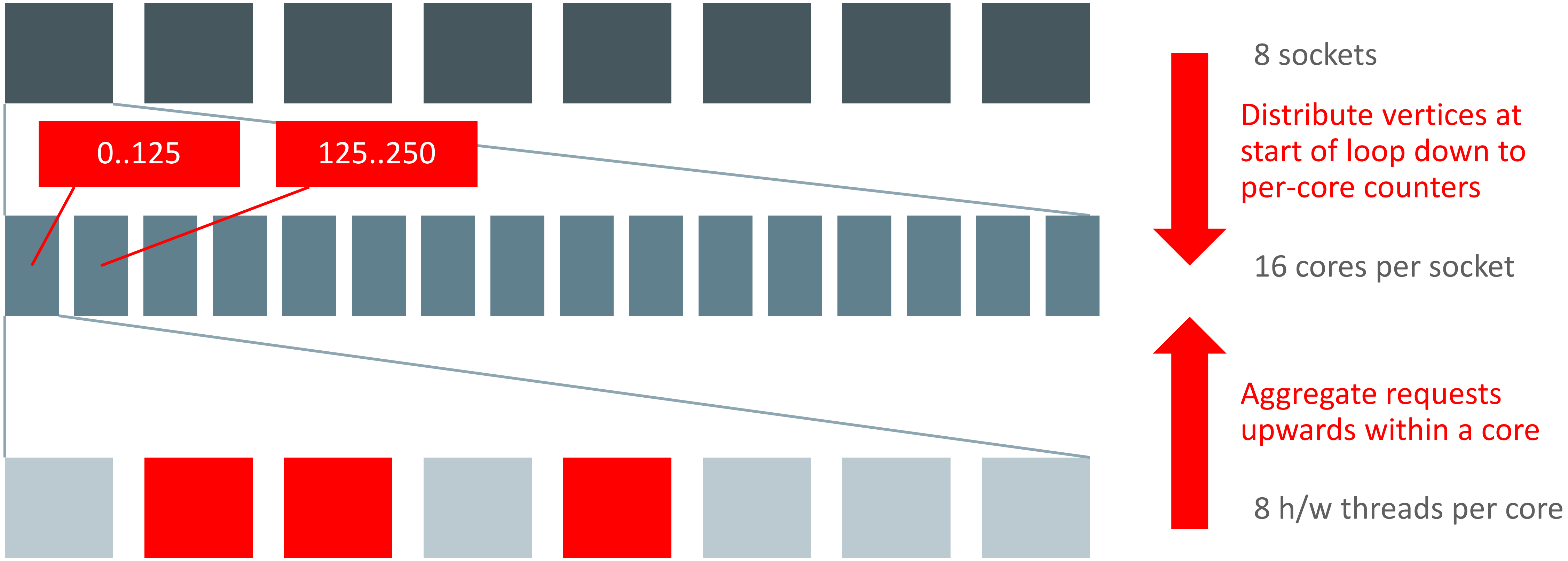
Consider distributing 0..16000 vertices, batch size 10



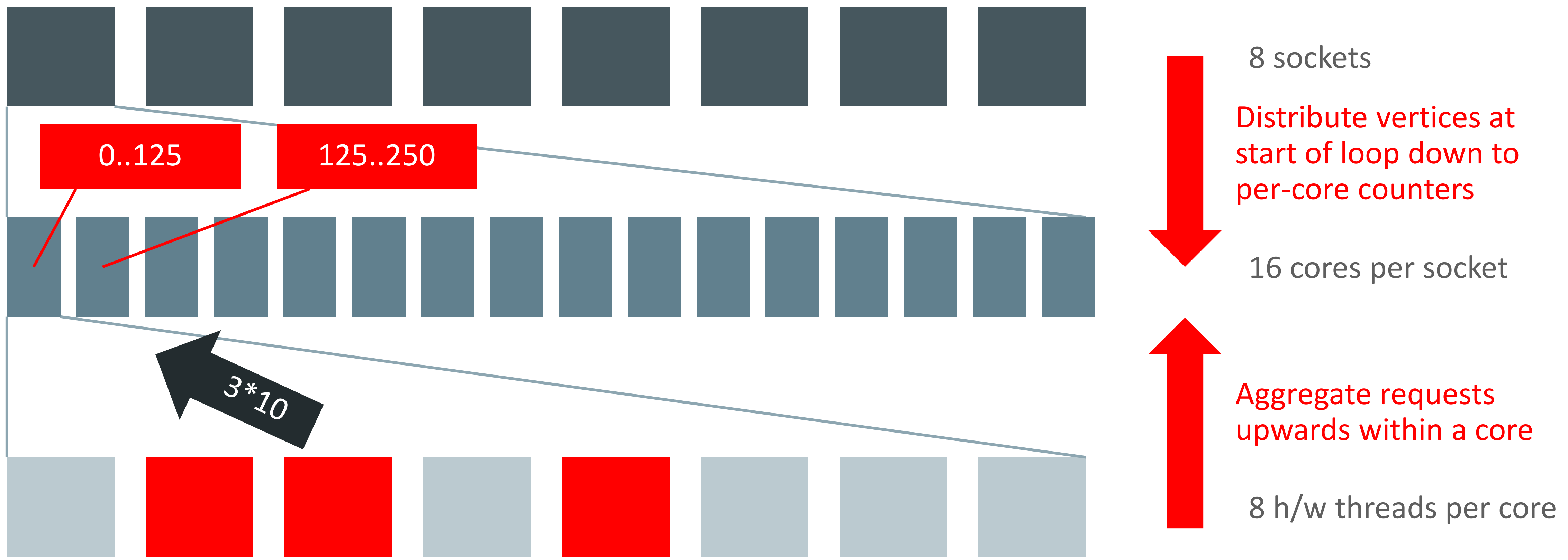
Consider distributing 0..16000 vertices, batch size 10



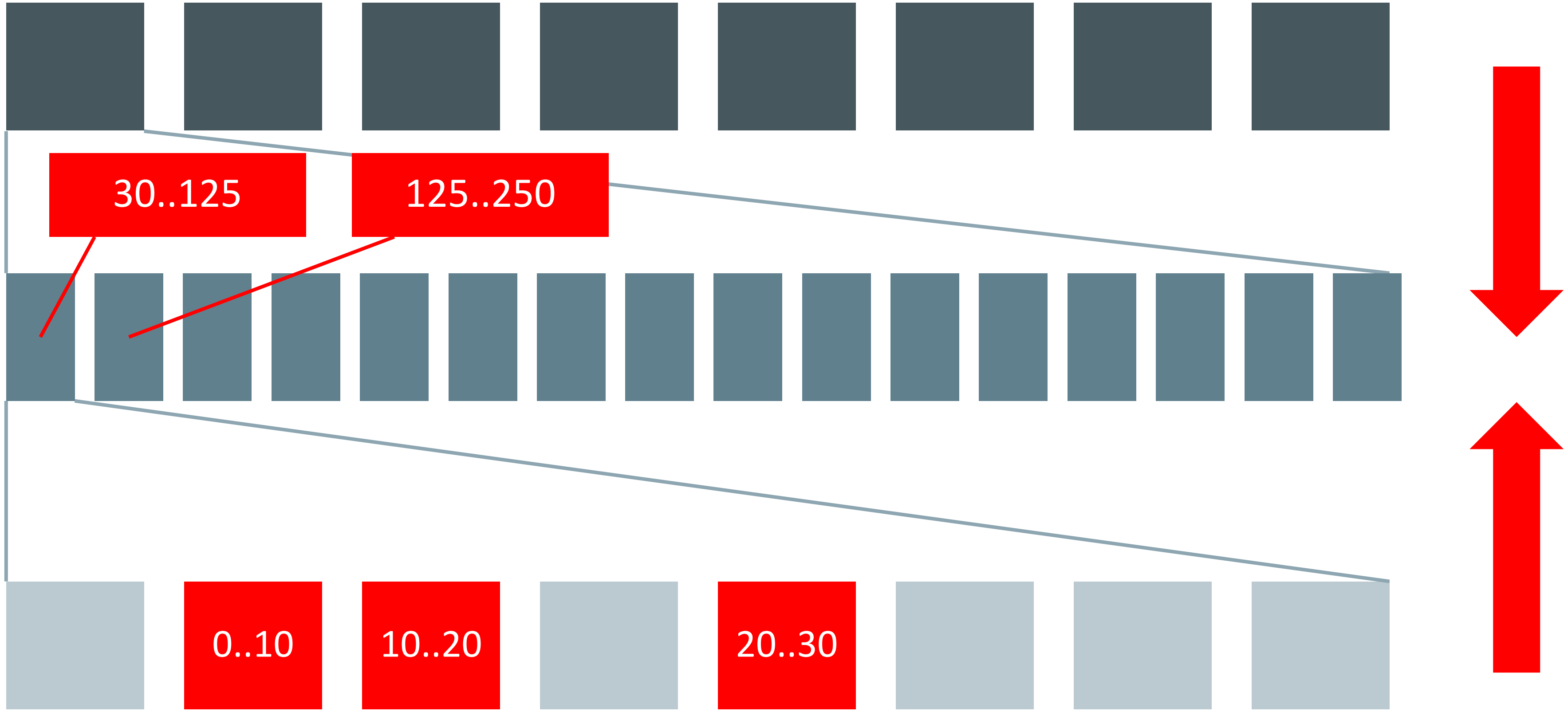
Consider distributing 0..16000 vertices, batch size 10



Consider distributing 0..16000 vertices, batch size 10



Consider distributing 0..16000 vertices, batch size 10



8 sockets

Distribute vertices at start of loop down to per-core counters

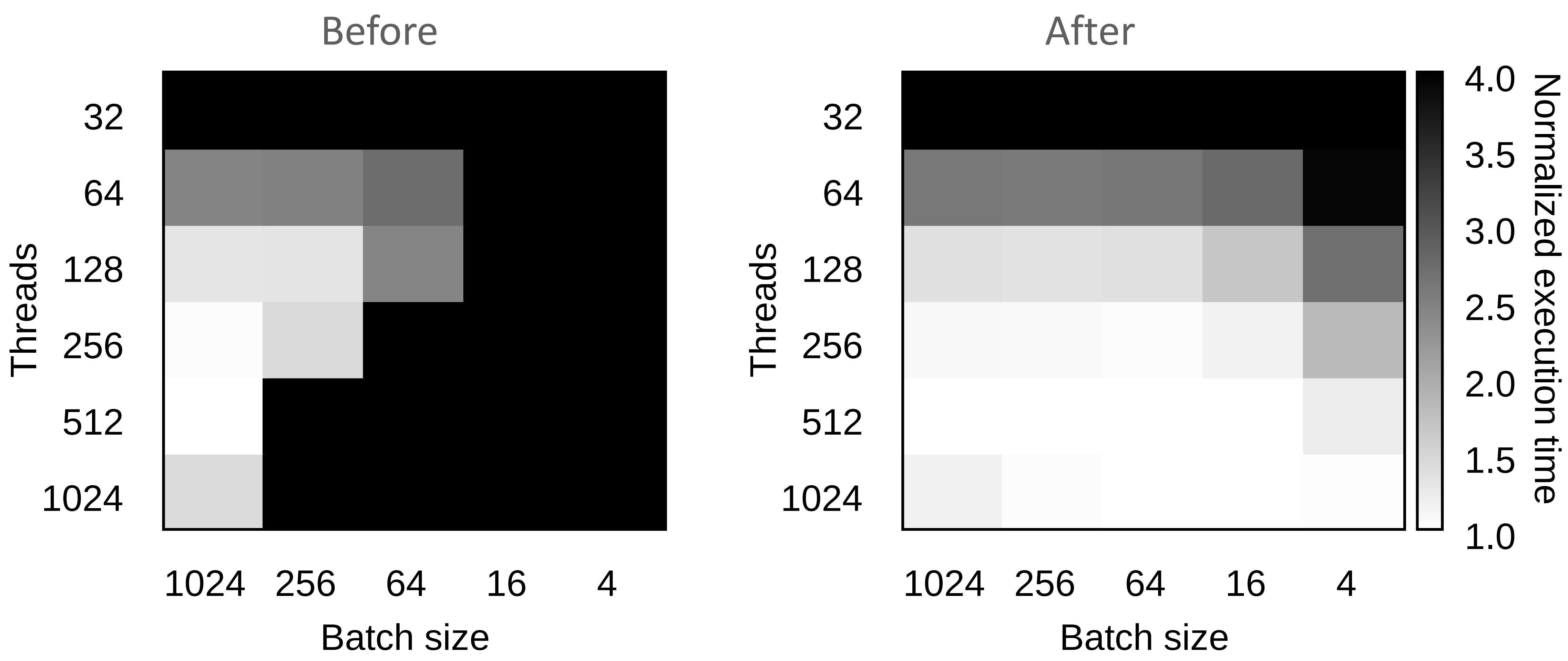
16 cores per socket

Aggregate requests upwards within a core

8 h/w threads per core

ORACLE

PageRank – SNAP LiveJournal (4.8M vertices, 69M edges)

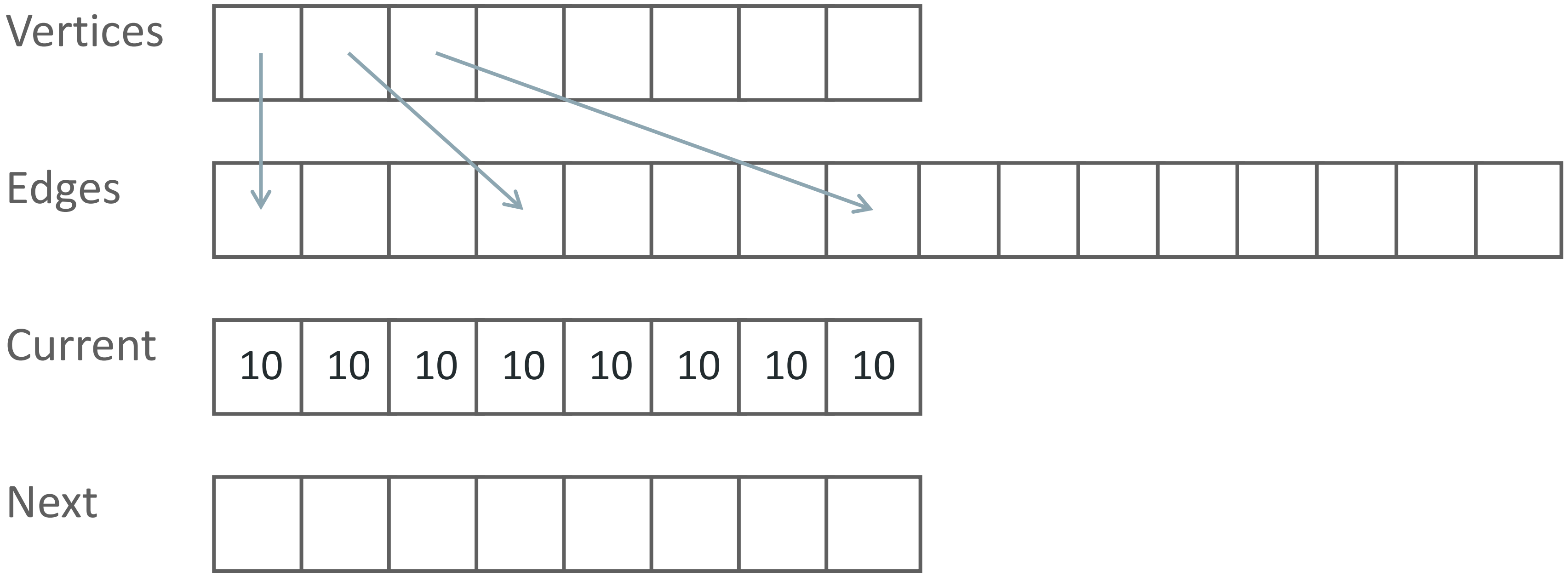


ORACLE

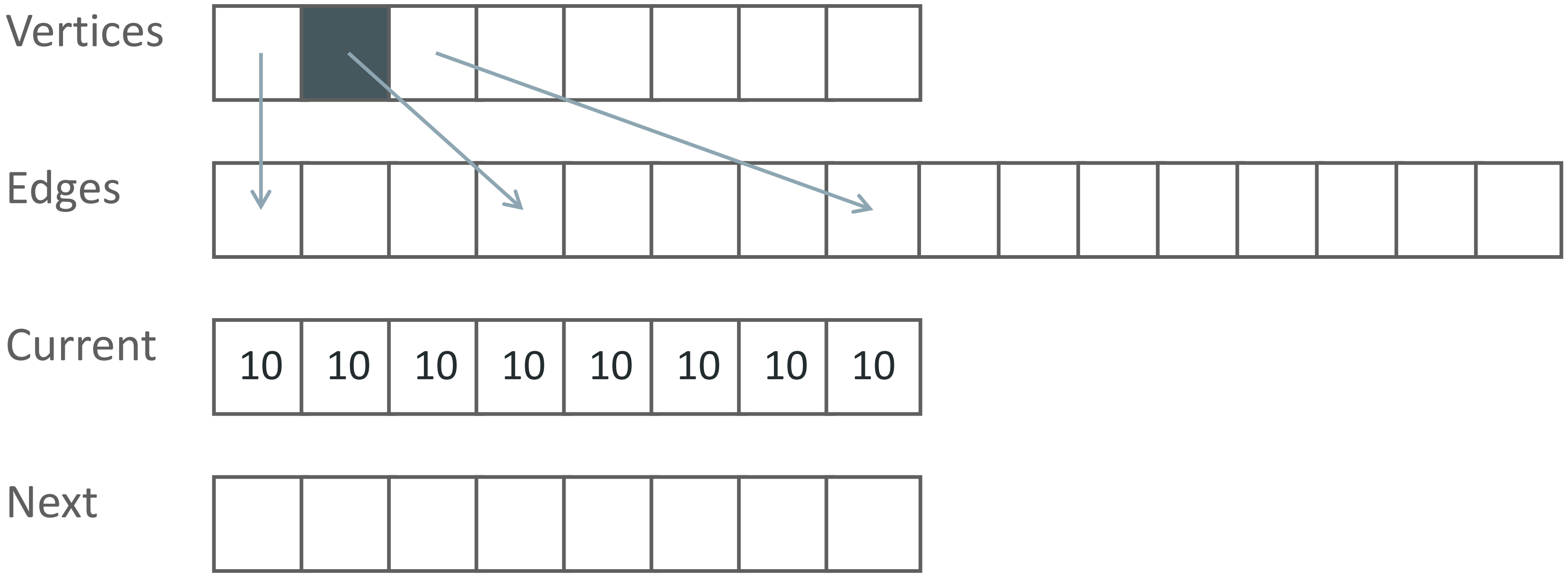
Case studies

1. Distributing parallel work
2. Memory allocation
3. Observations

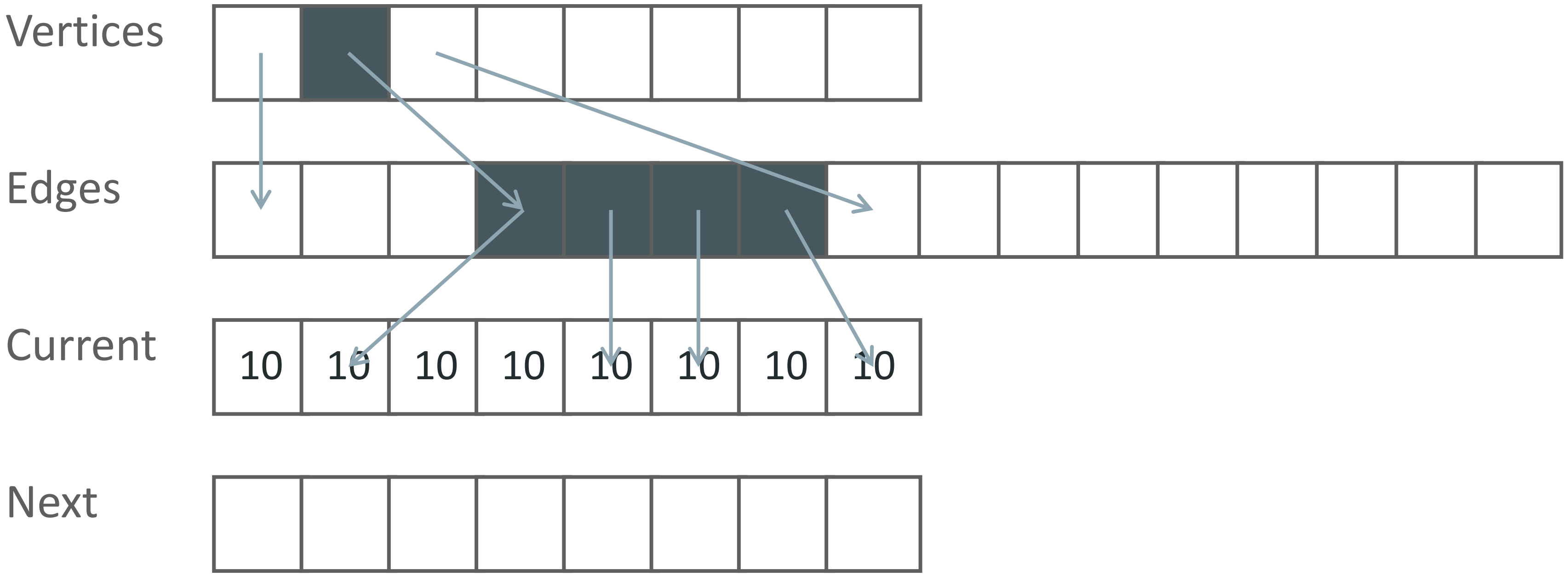
PageRank inner loop



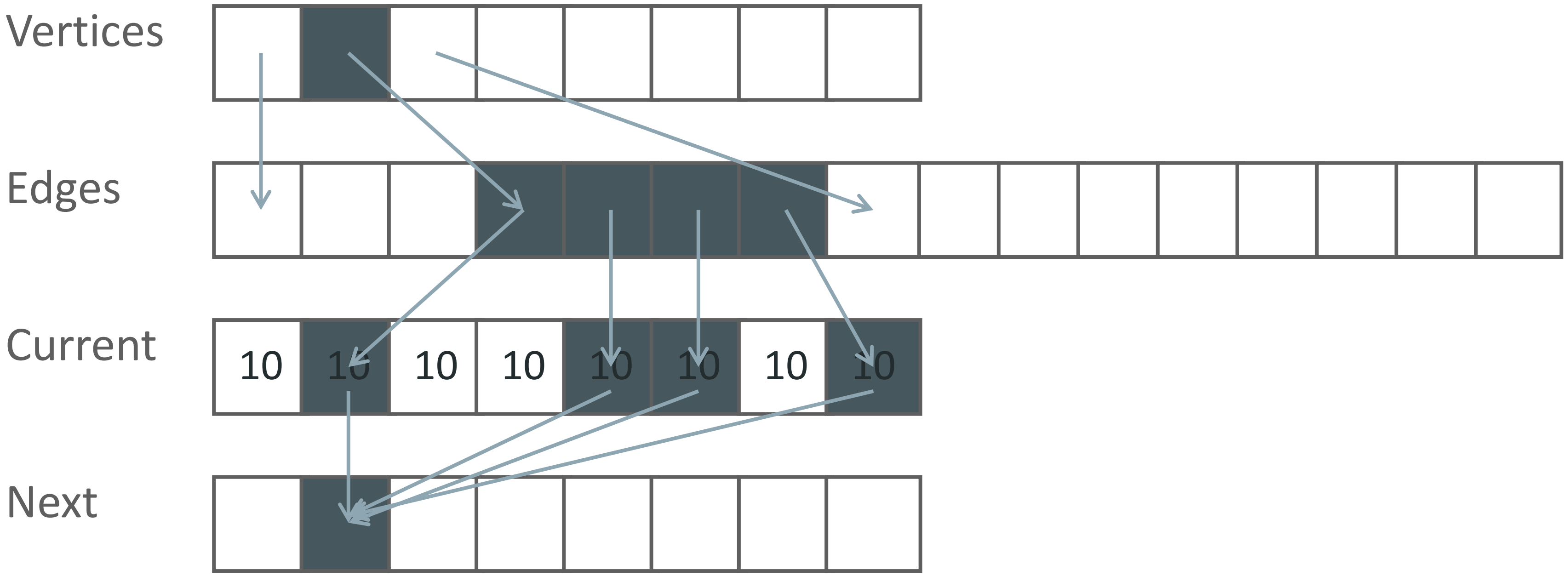
PageRank inner loop



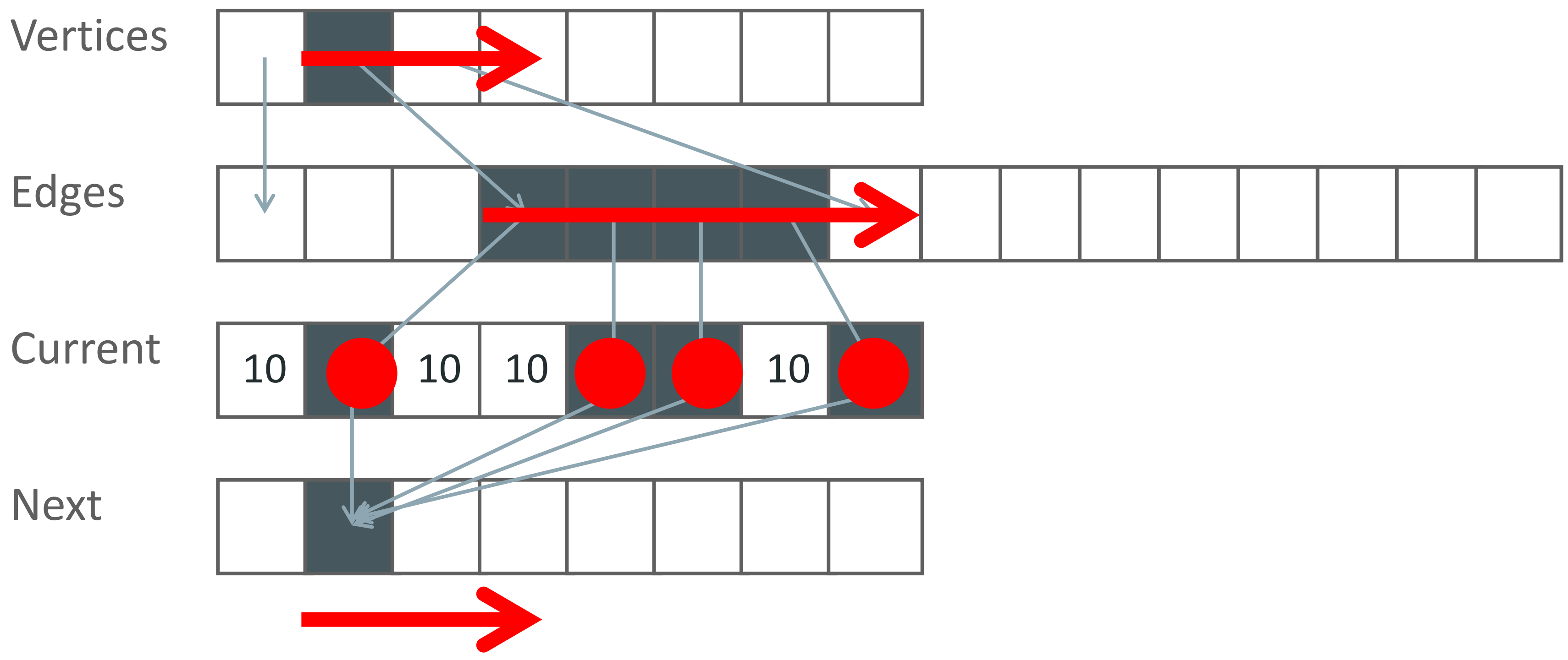
PageRank inner loop



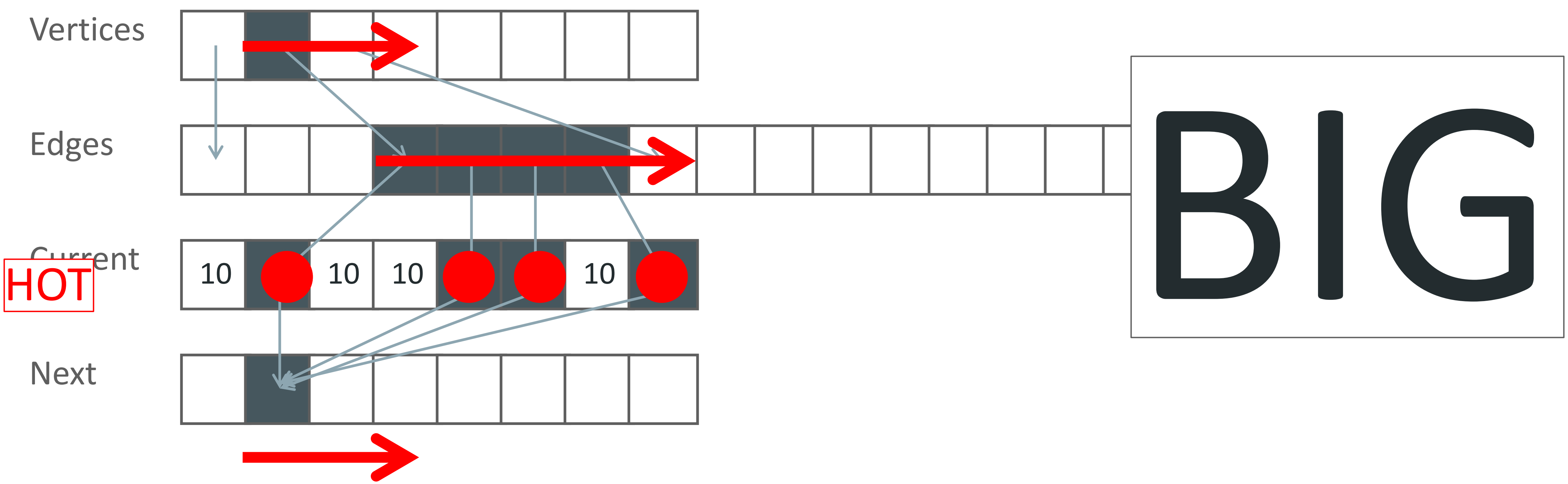
PageRank inner loop



PageRank inner loop

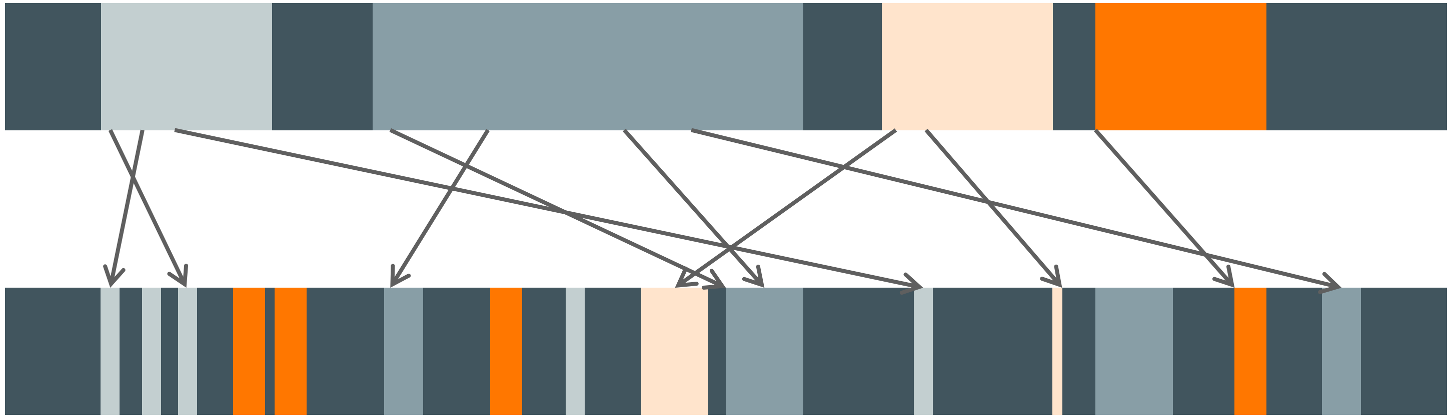


PageRank inner loop



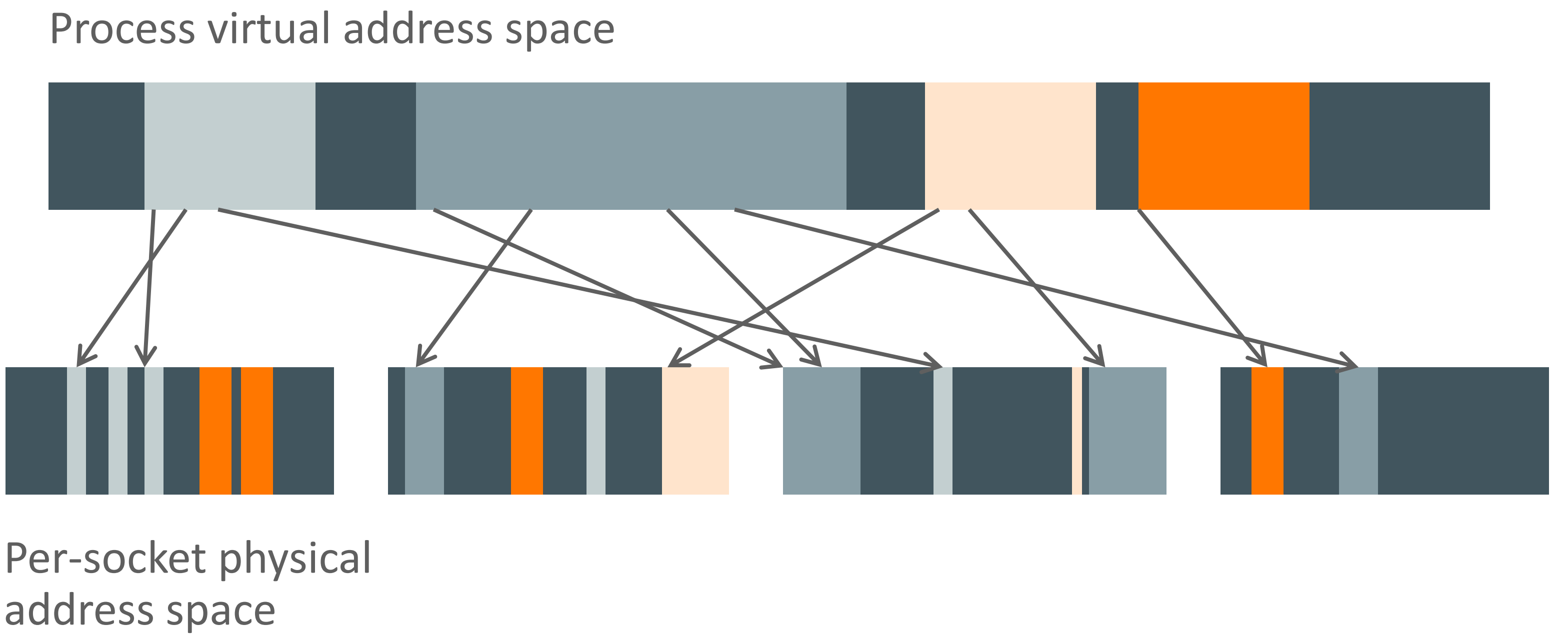
Logical view of memory – ccNUMA

Process virtual address space

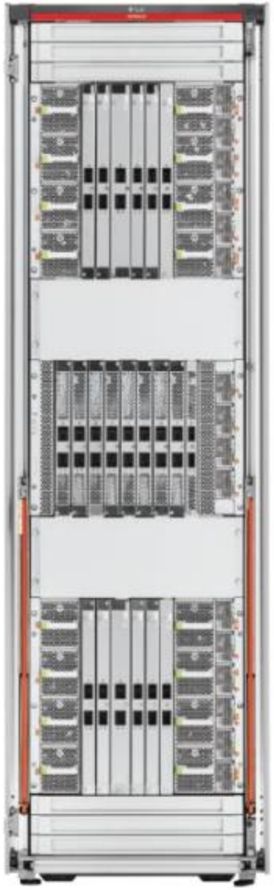


Machine physical address space

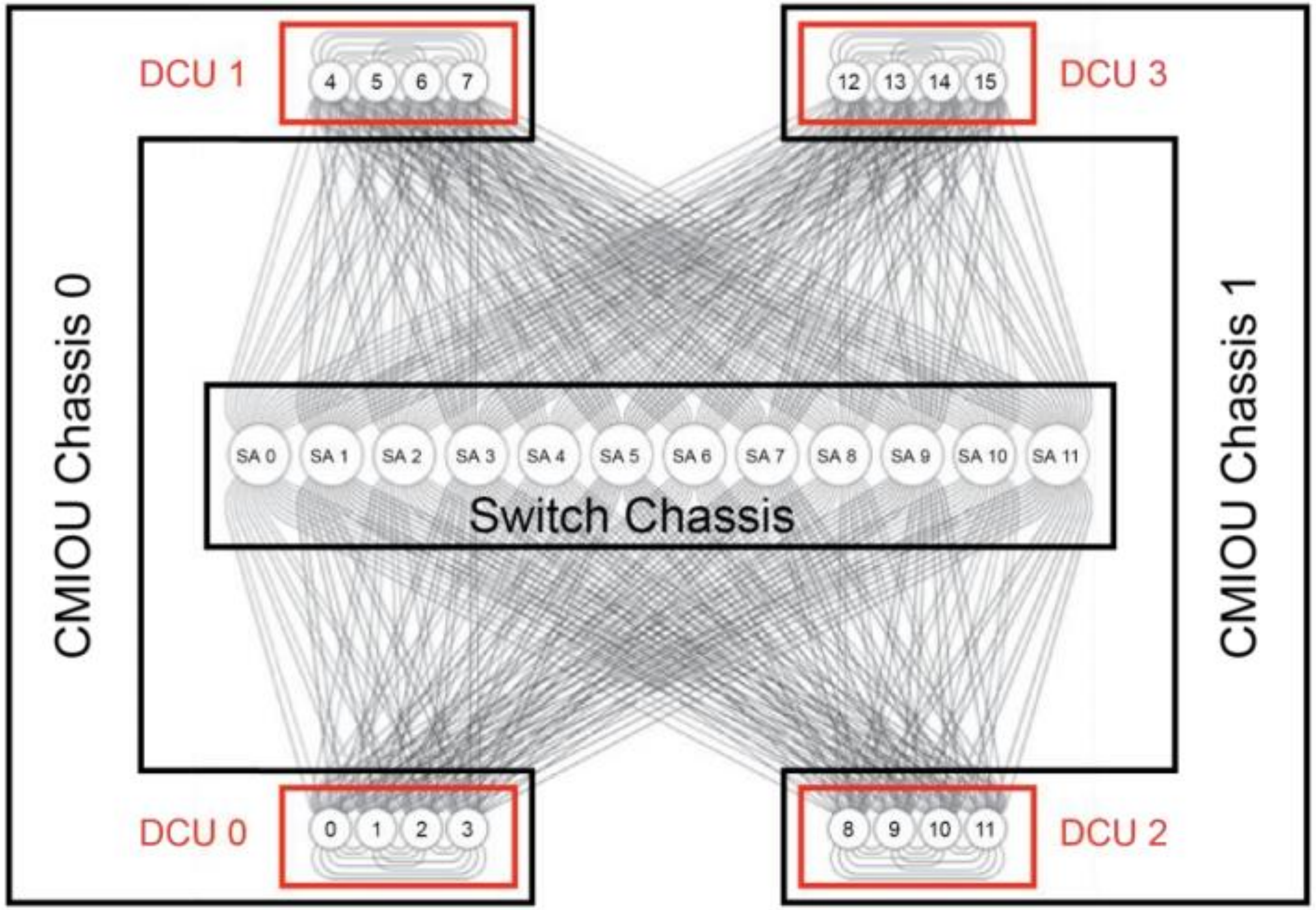
Logical view of memory – ccNUMA



M7-16, physical organization


Front View

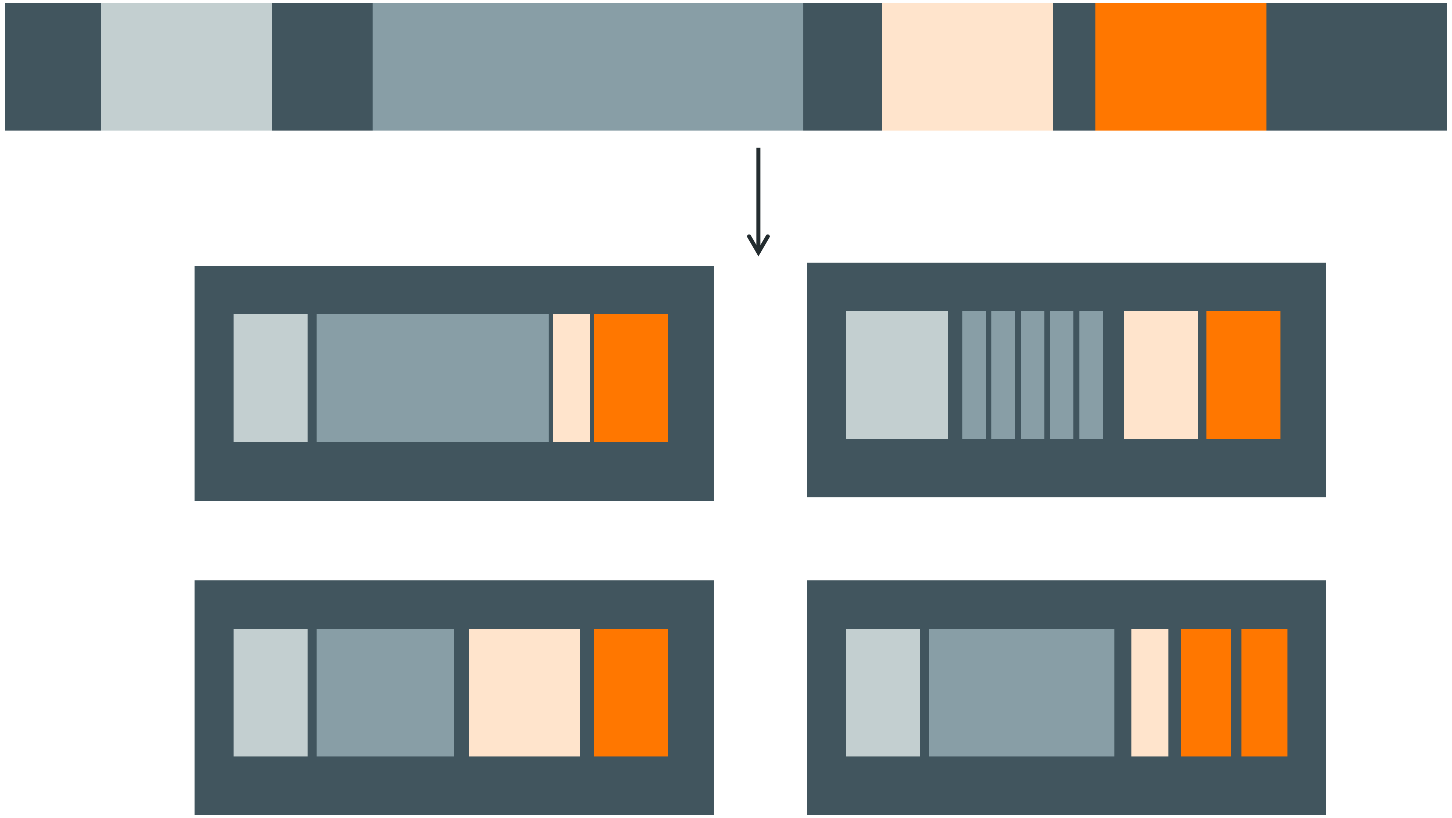

Rear View



http://www.oracle.com/technetwork/server-storage/sun-sparc-enterprise/documentation/sparc-t7-m7-server-architecture-2702877.pdf

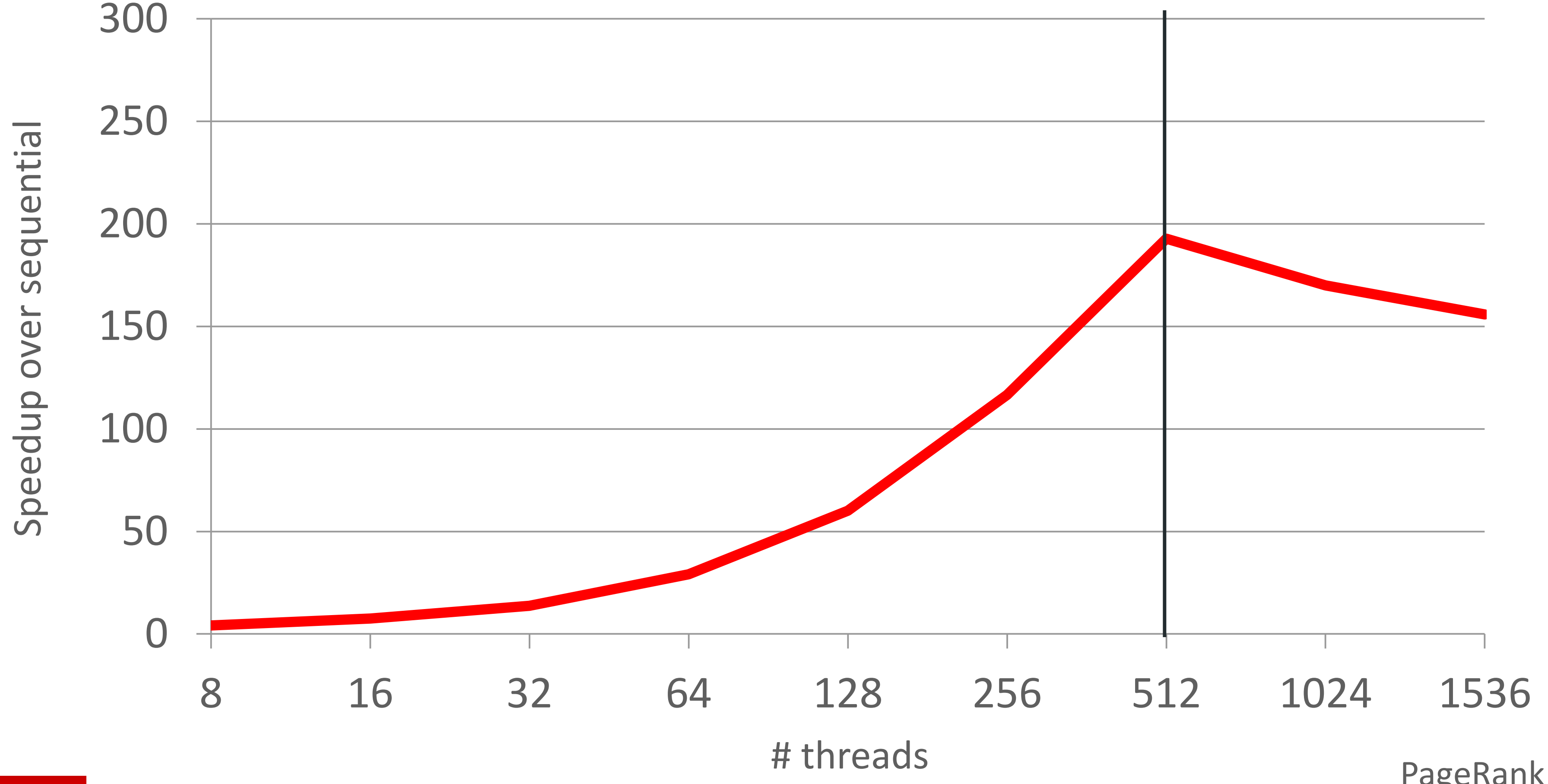
Parallel runs – reasonable defaults

Distribute memory across the whole machine



Parallel runs – reasonable defaults

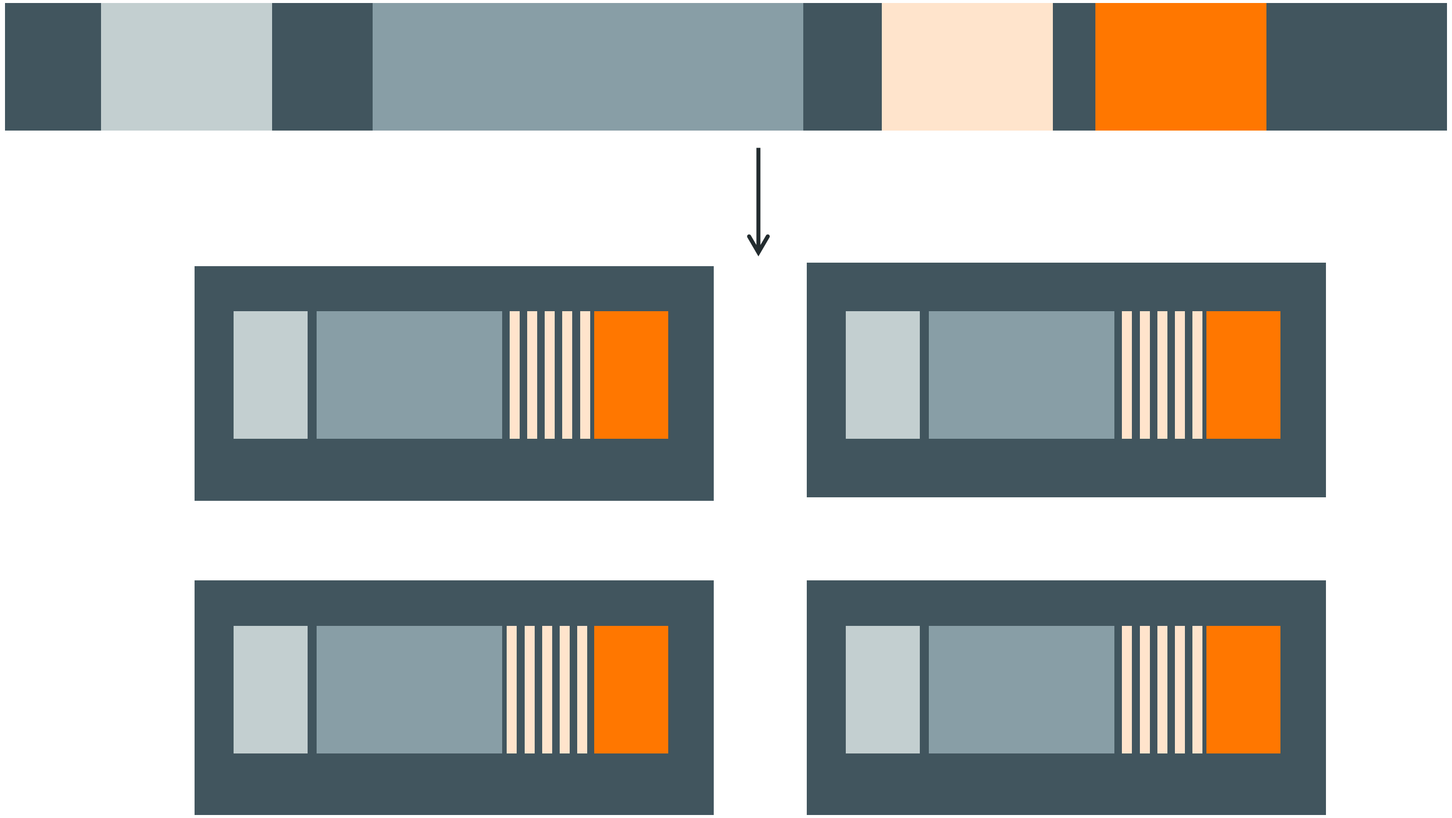
Distribute memory across the whole machine



PageRank, S27 data set, M7-16

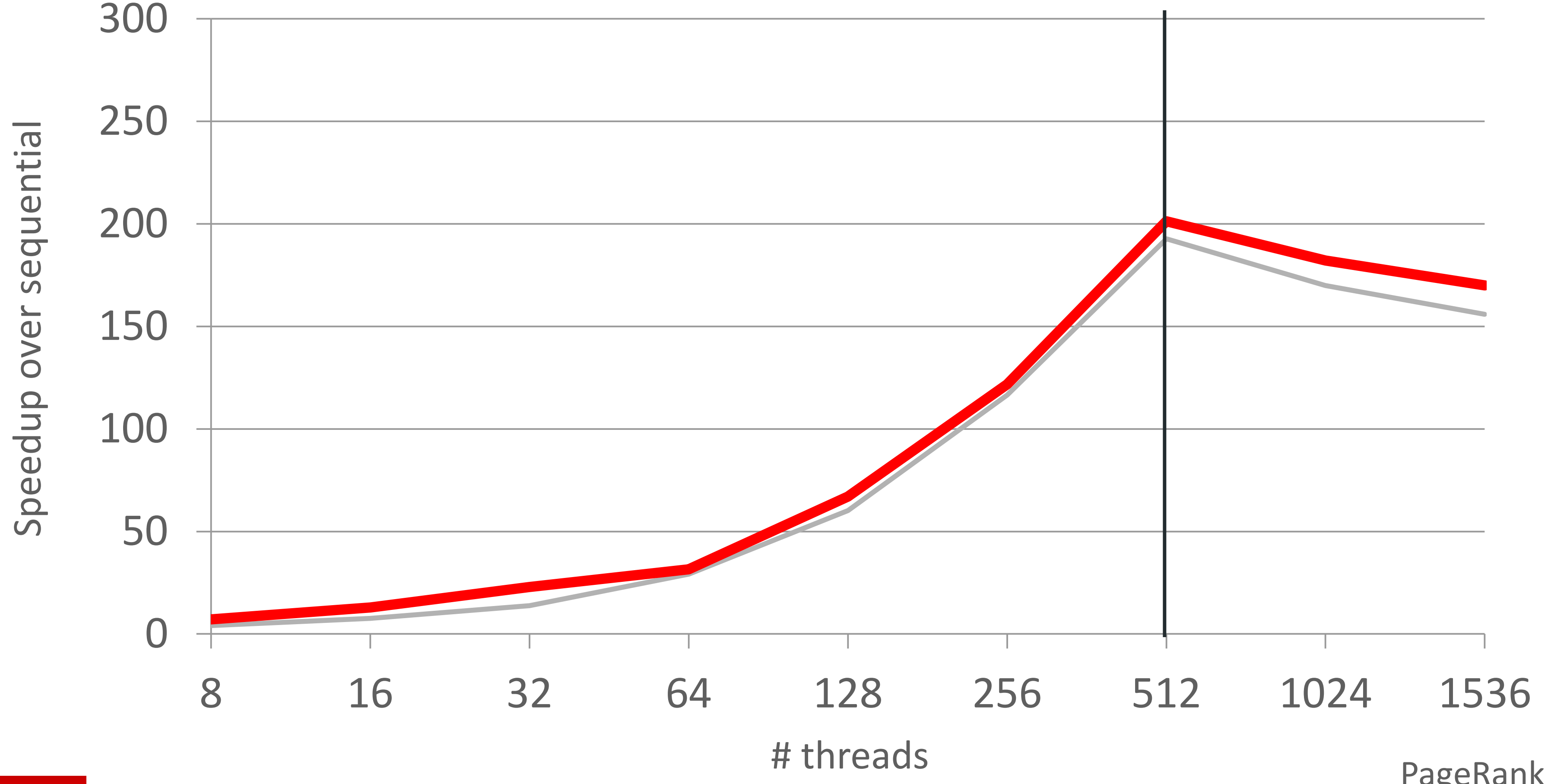
Parallel runs – control placement explicitly

Distribute over active sockets, control translation sizes



Parallel runs – control placement explicitly

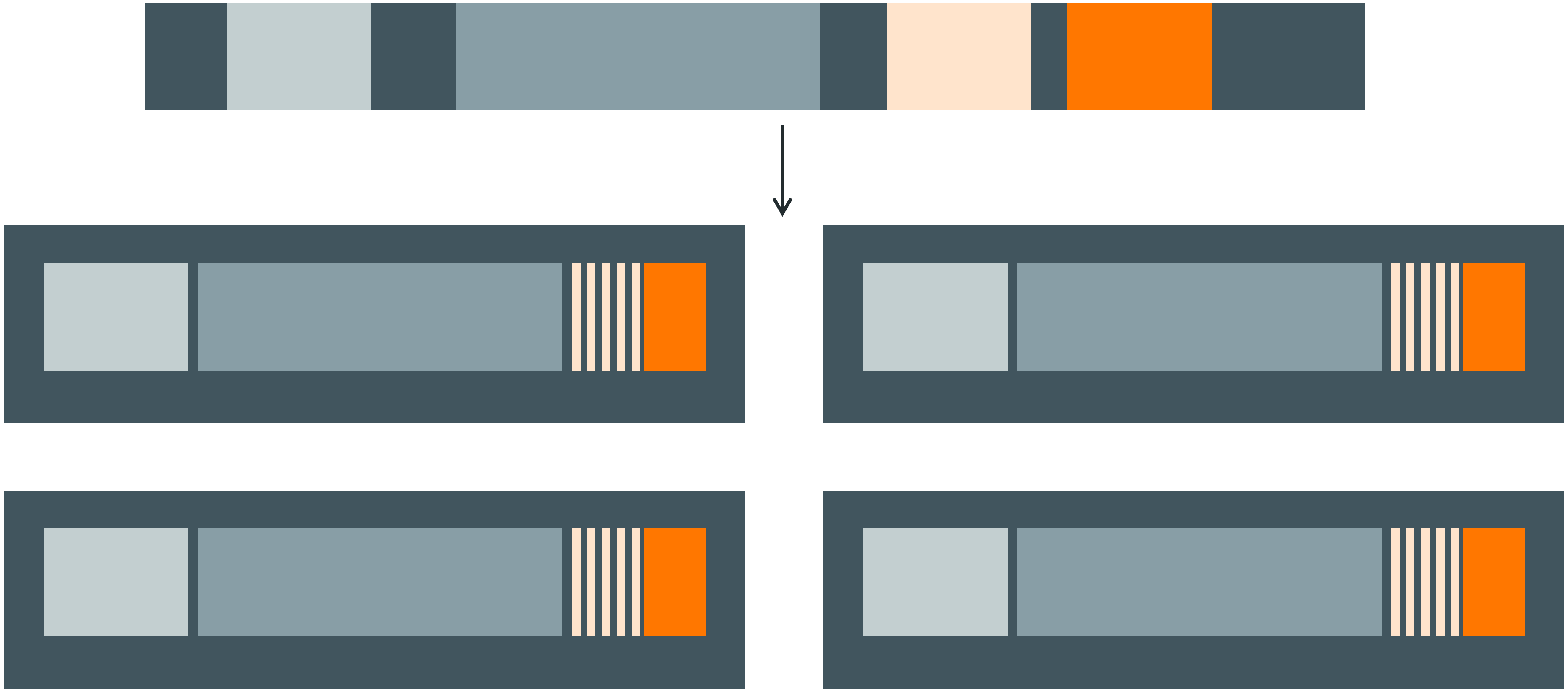
Distribute over active sockets, control translation sizes



PageRank, S27 data set, M7-16

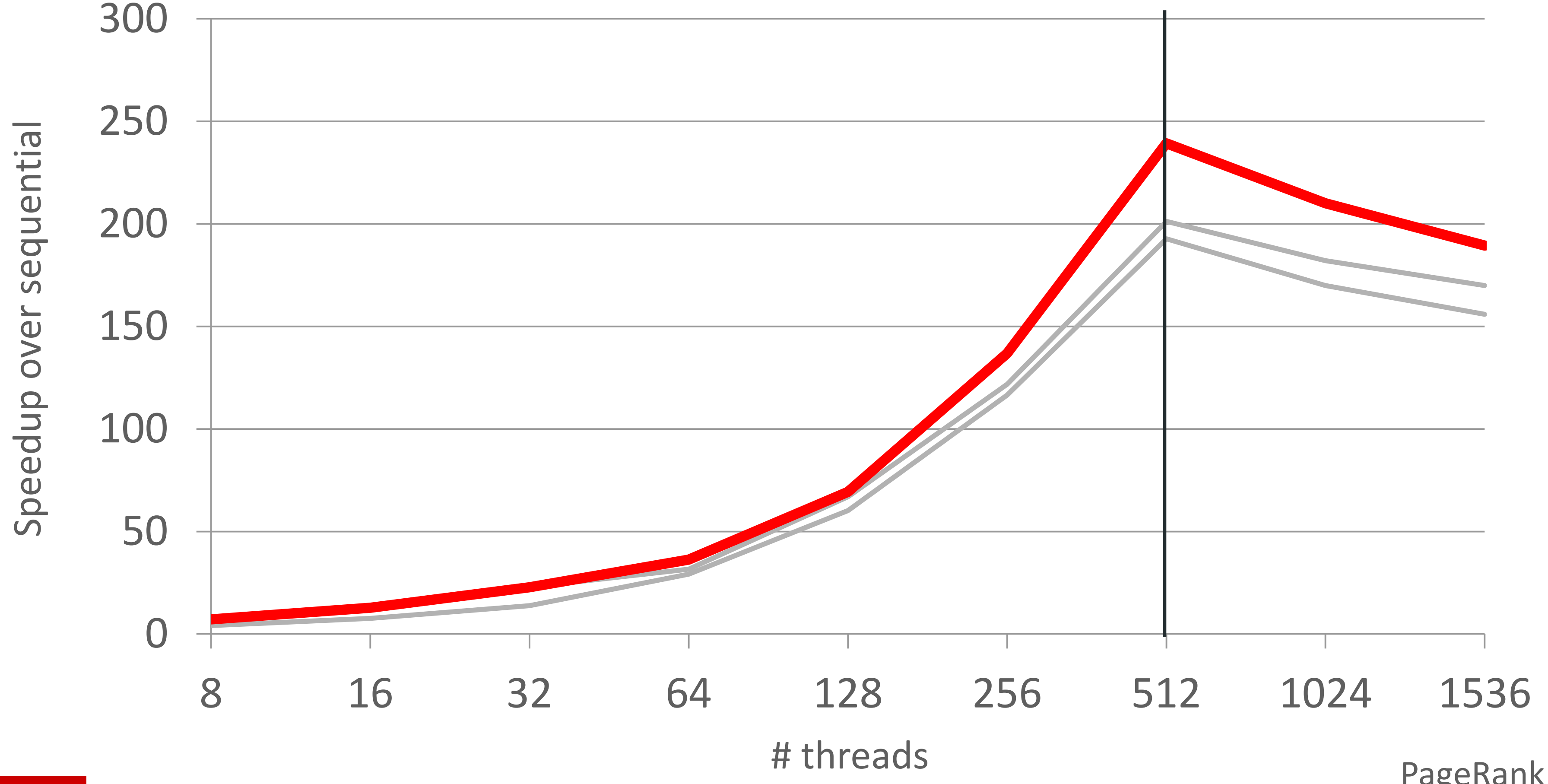
Parallel runs – control placement explicitly

Replicate read-only data to active sockets



Parallel runs – control placement explicitly

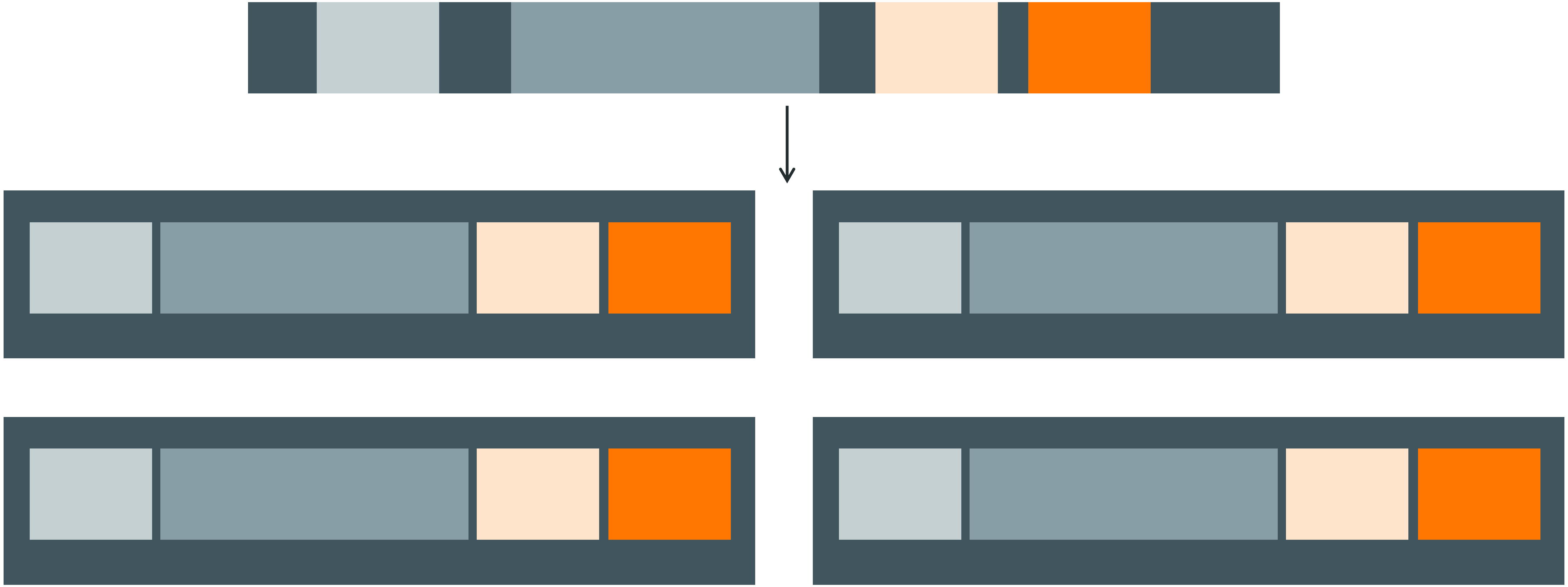
Replicate read-only data to active sockets



PageRank, S27 data set, M7-16

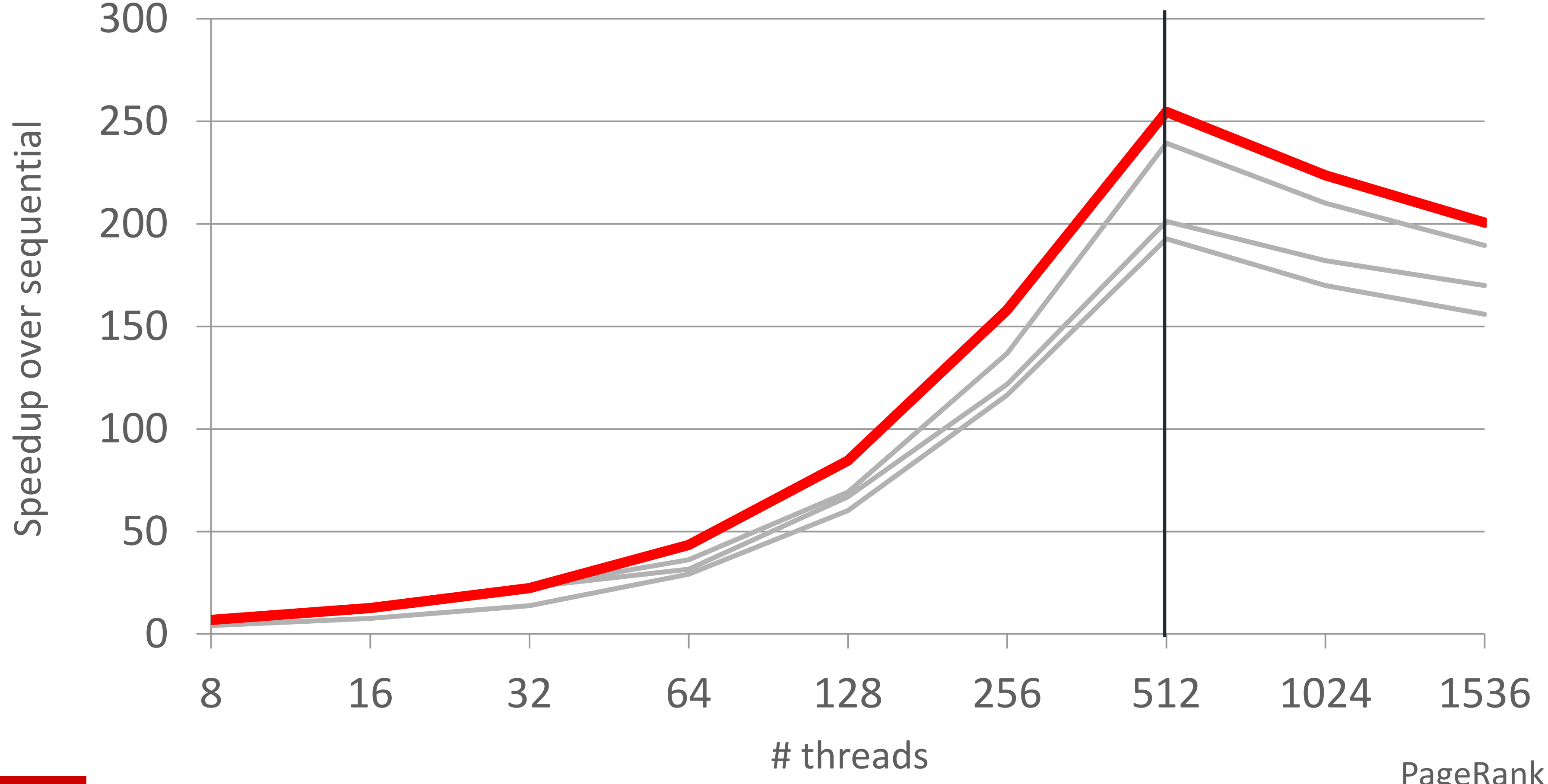
Parallel runs – control placement explicitly

Re-replicate read-write data after each phase



Parallel runs – control placement explicitly

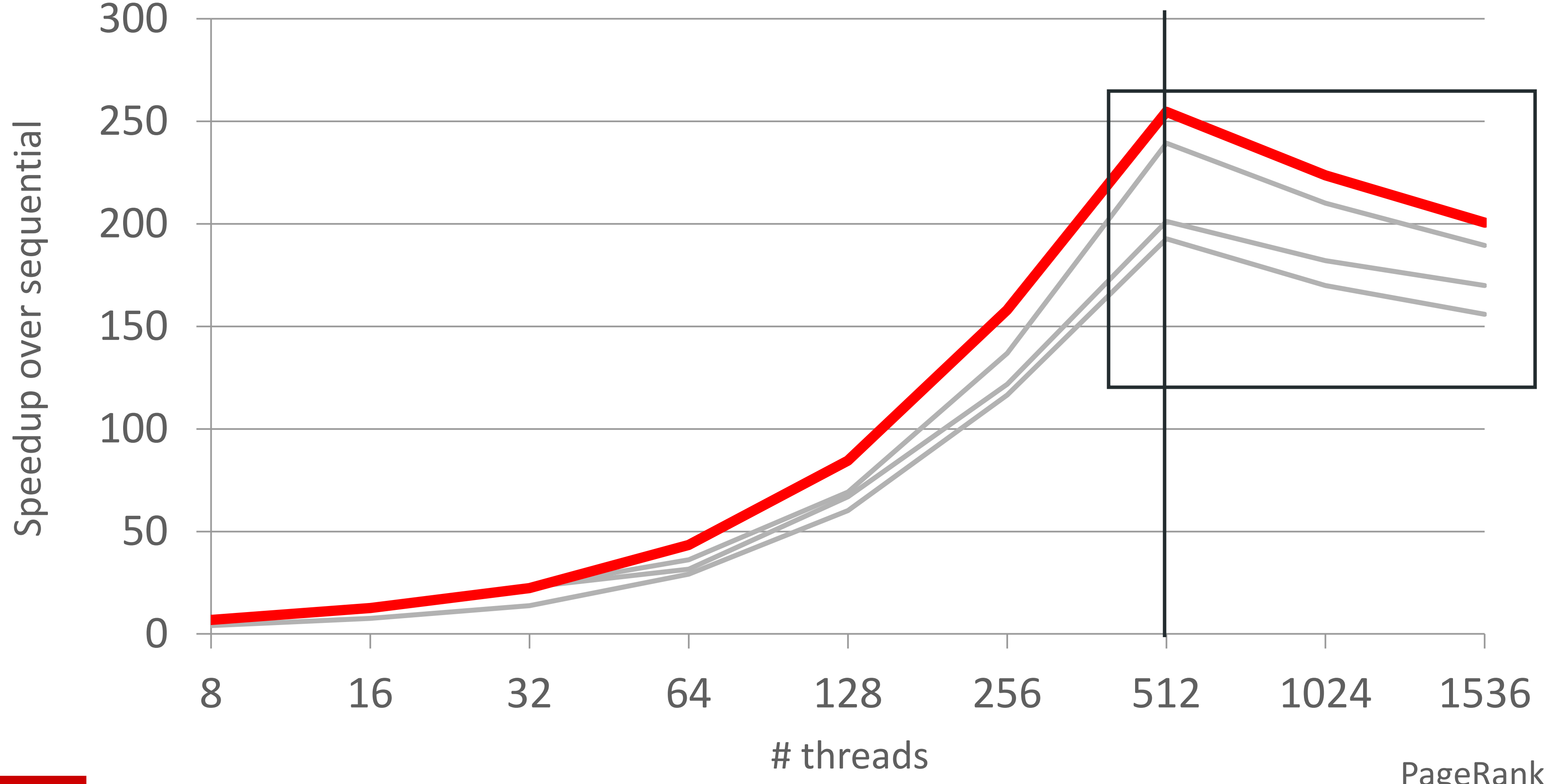
Re-replicate read-write data after each phase



PageRank, S27 data set, M7-16

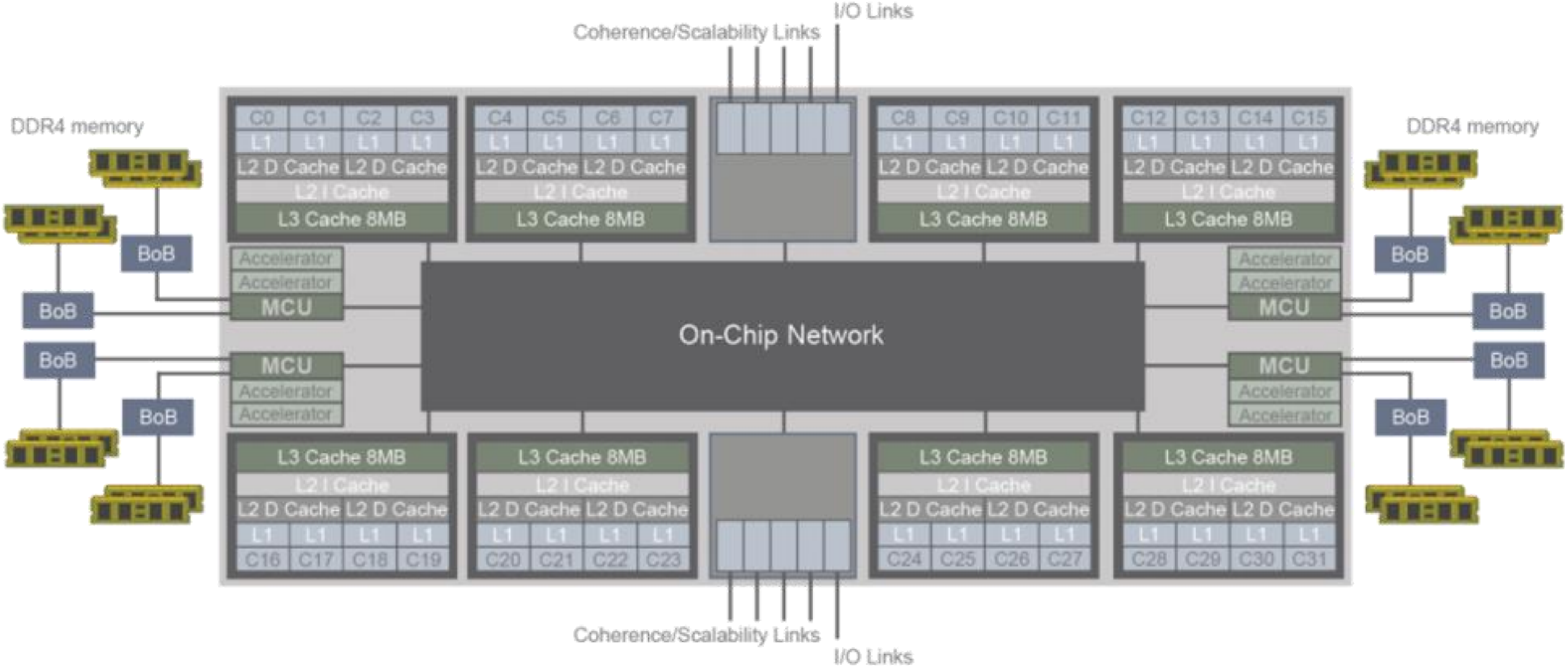
Parallel runs – control placement explicitly

Re-replicate read-write data after each phase



PageRank, S27 data set, M7-16

SPARC M7 processor, single socket

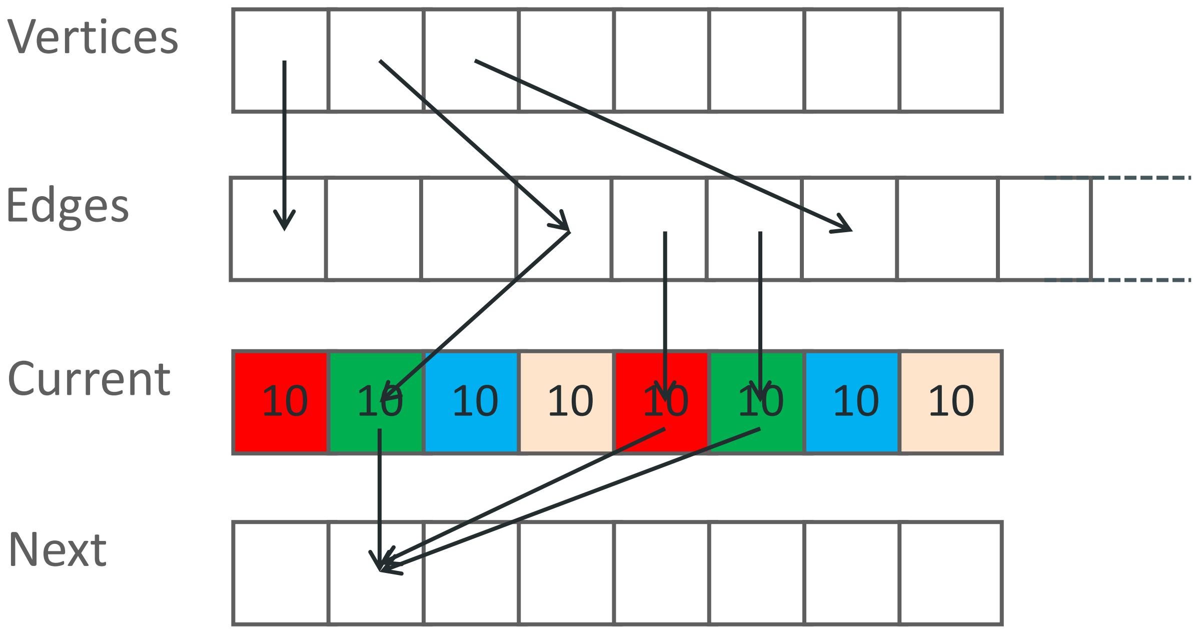


http://www.oracle.com/technetwork/server-storage/sun-sparc-enterprise/documentation/sparc-t7-m7-server-architecture-2702877.pdf

Dividing graphs into tiles

No attempt to exploit graph structure (there may be none to exploit)

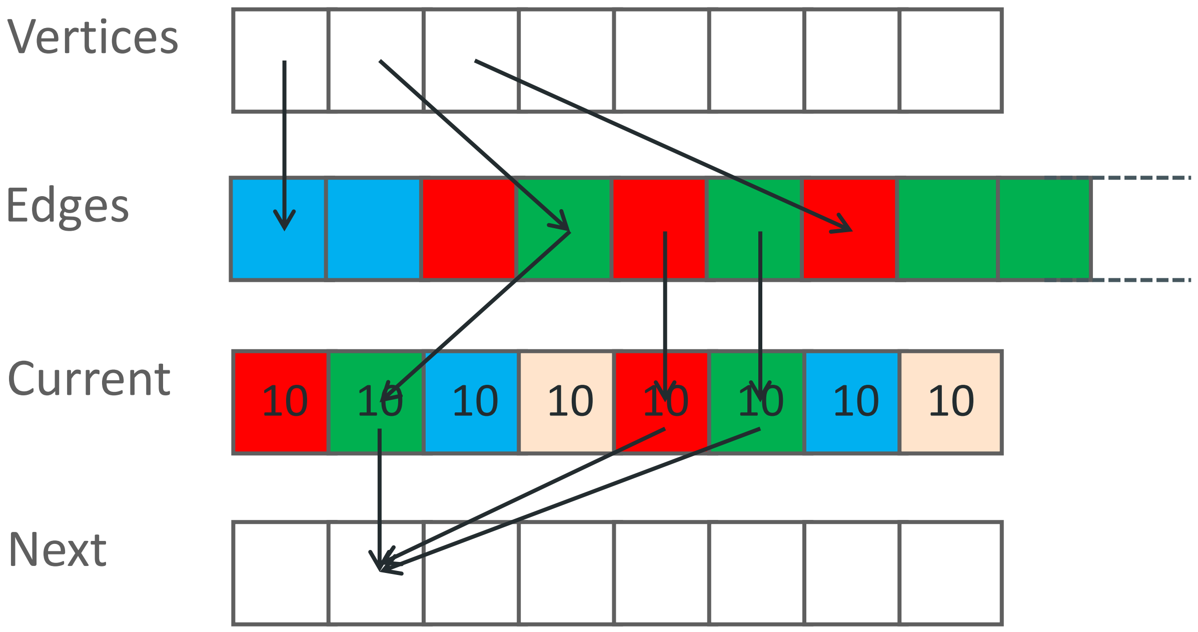
1. Focus on the hot randomly-accessed array
2. Assign elements to tiles round-robin



Dividing graphs into tiles

No attempt to exploit graph structure (there may be none to exploit)

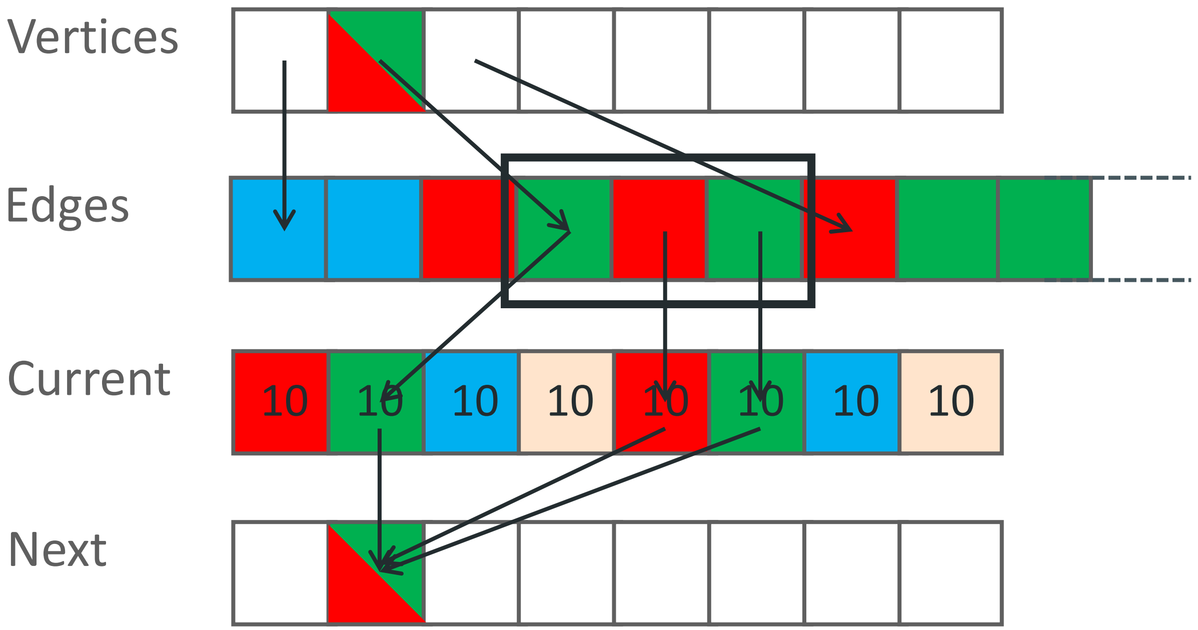
1. Focus on the hot randomly-accessed array
2. Assign elements to tiles round-robin
3. Assign edges to their target's tile



Dividing graphs into tiles

No attempt to exploit graph structure (there may be none to exploit)

1. Focus on the hot randomly-accessed array
2. Assign elements to tiles round-robin
3. Assign edges to their target's tile
4. Duplicate vertices+next in each connected tile

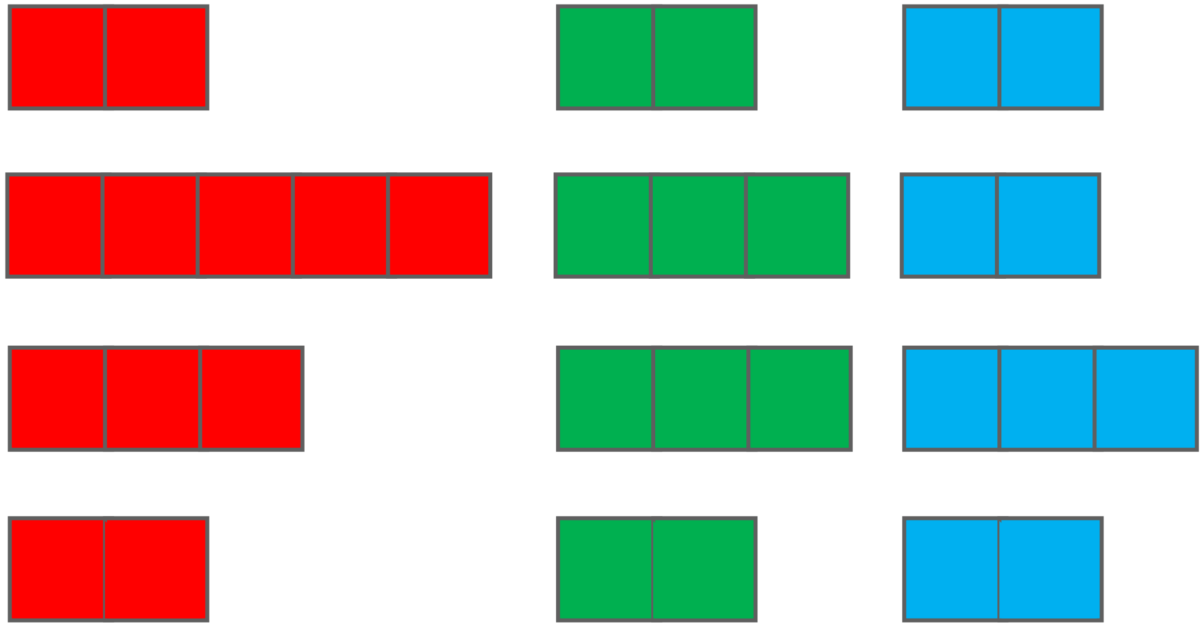


ORACLE

Dividing graphs into tiles

No attempt to exploit graph structure (there may be none to exploit)

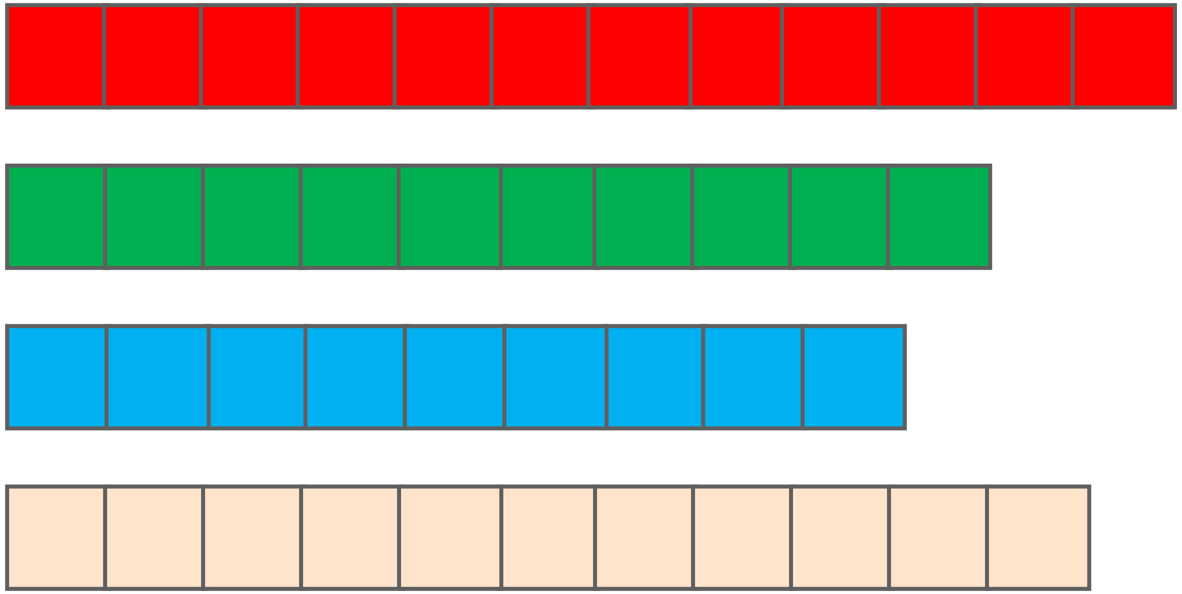
1. Focus on the hot randomly-accessed array
2. Assign elements to tiles round-robin
3. Assign edges to their target's tile
4. Duplicate vertices+next in each connected tile
5. Generate fresh graph representation per tile



Dividing graphs into tiles

No attempt to exploit graph structure (there may be none to exploit)

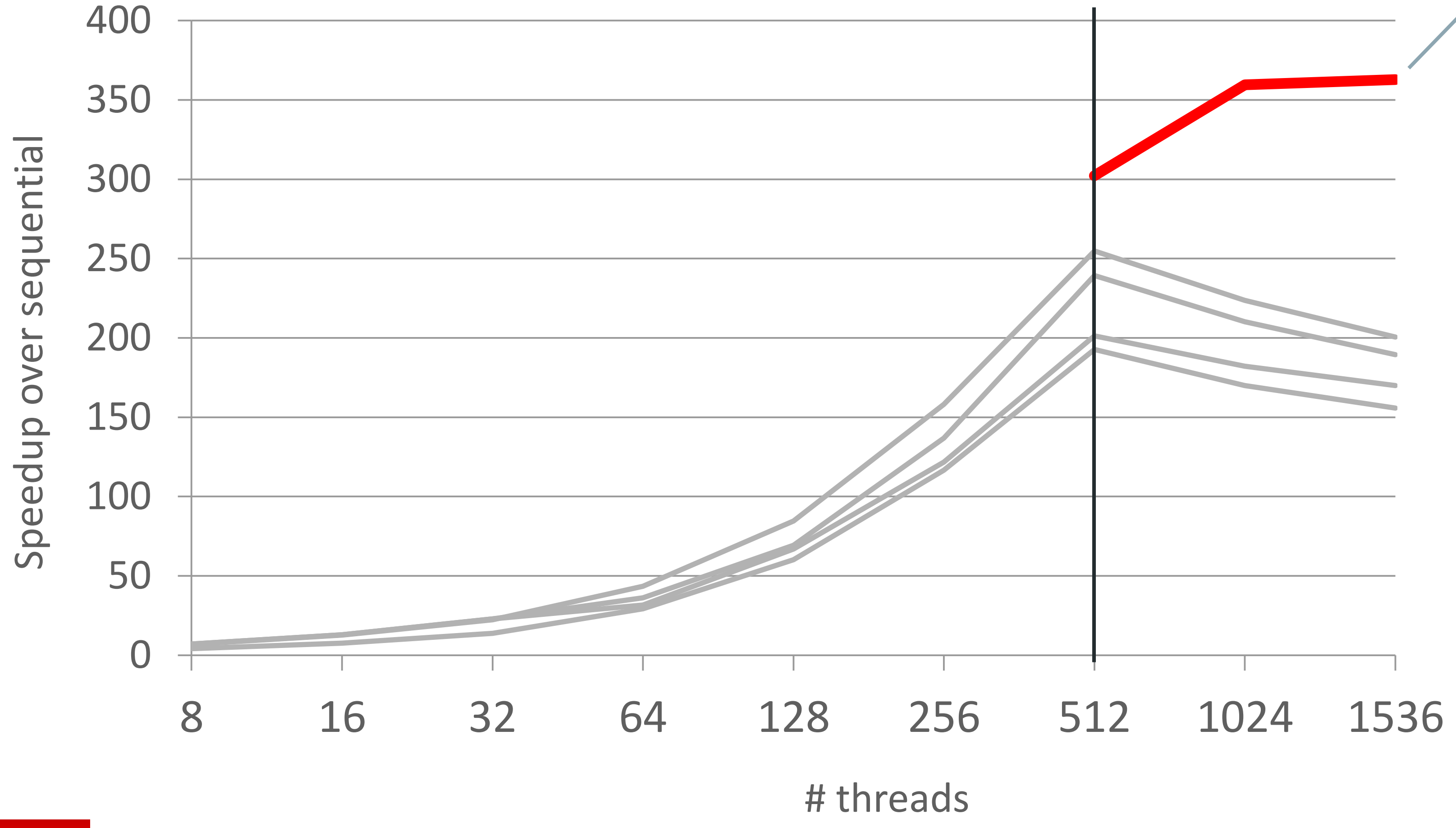
1. Focus on the hot randomly-accessed array
2. Assign elements to tiles round-robin
3. Assign edges to their target's tile
4. Duplicate vertices+next in each connected tile
5. Generate fresh graph representation per tile
6. Allocate each tile separately



ORACLE

Parallel runs with tiling

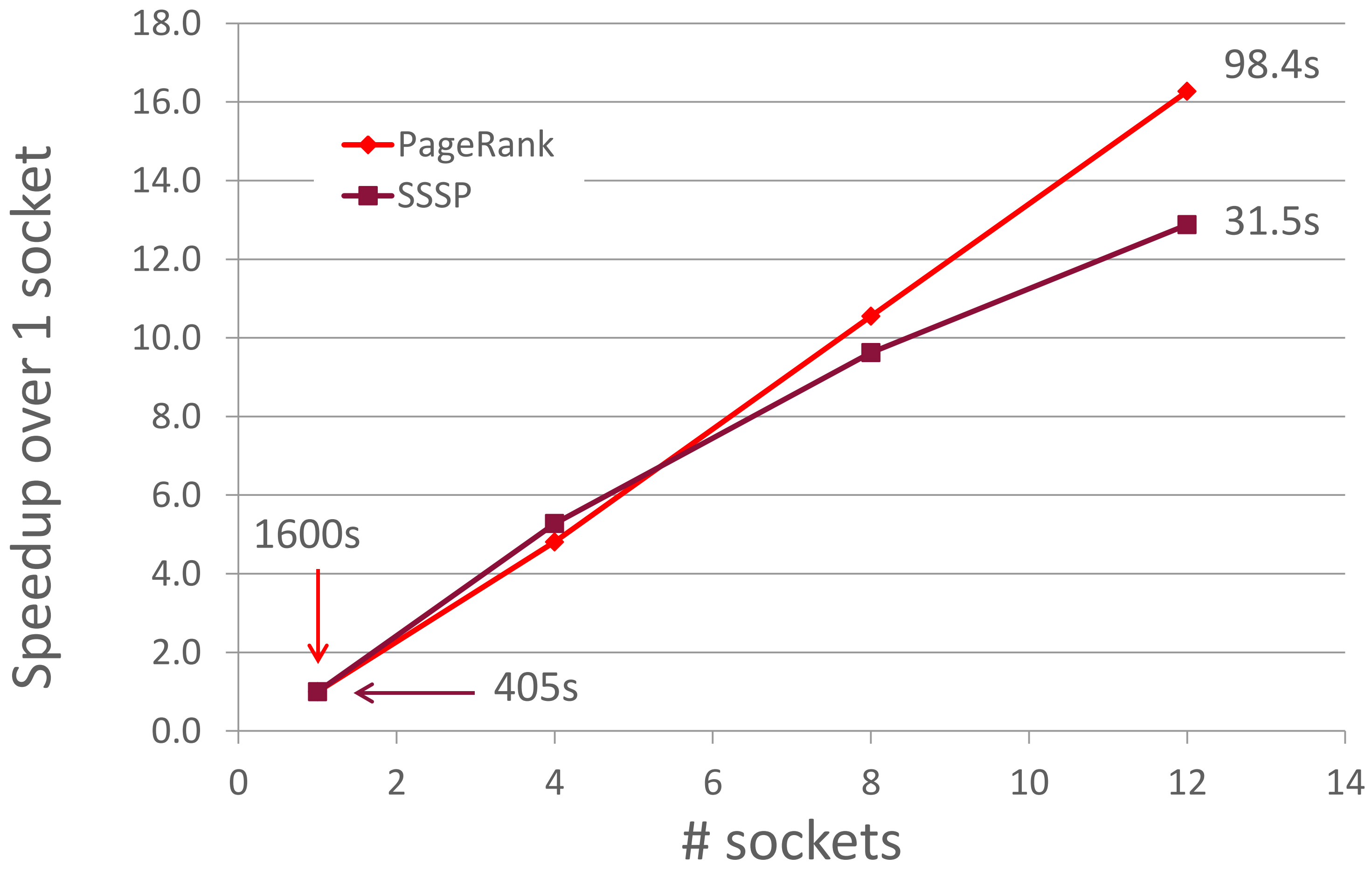
633x compared with 1-thread with memory distributed



PageRank, S27 data set, M7-16

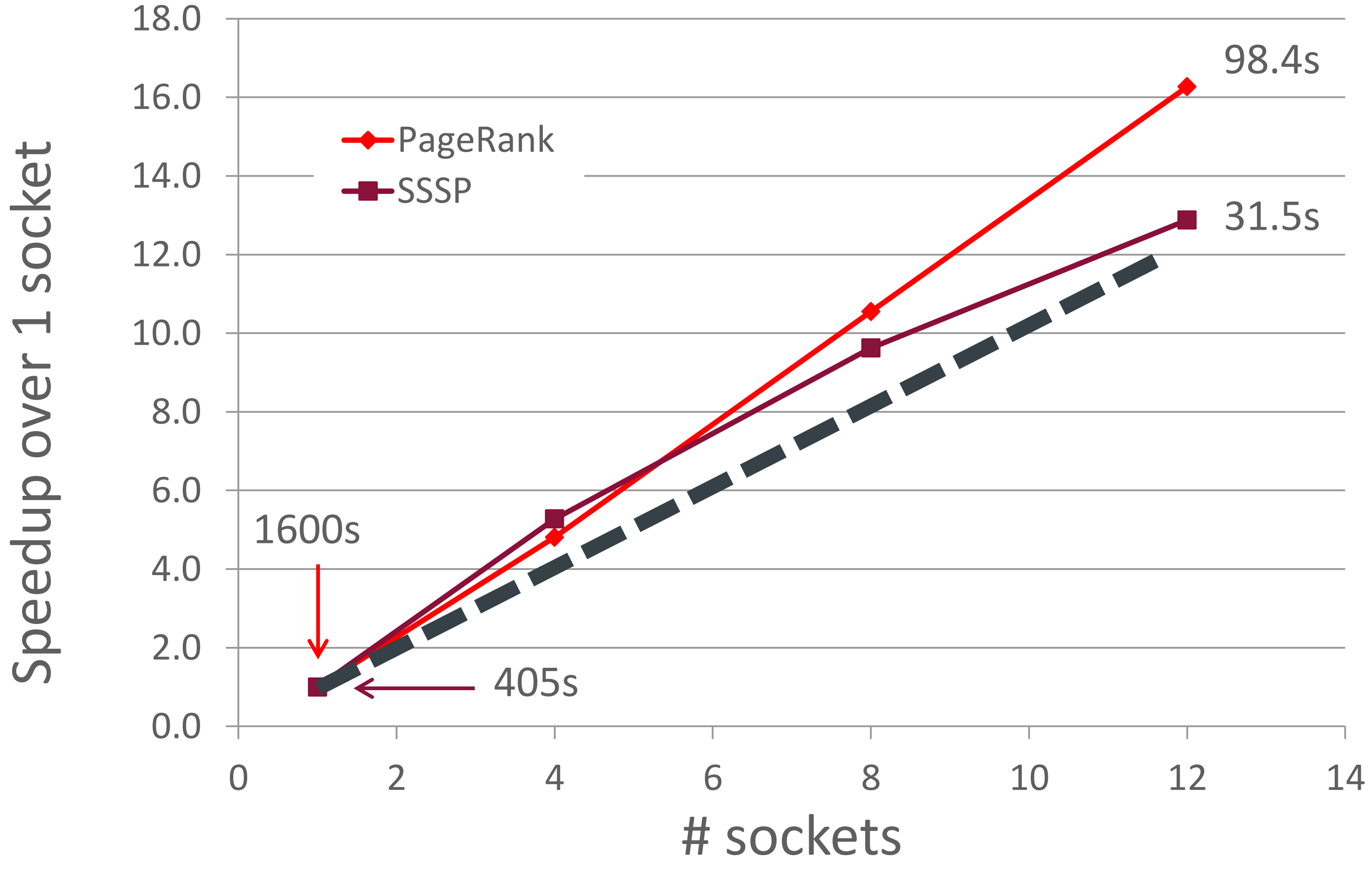
ORACLE

Parallel runs with tiling – larger input



M7-16
1..12 sockets
6 active threads / core
G500 Scale-32 input
16B edges (4*2^32)
~4 TB in memory

Parallel runs with tiling – larger input



M7-16
1..12 sockets
6 active threads / core
G500 Scale-32 input
16B edges (4*2^32)
~4 TB in memory

Case studies

1. Distributing parallel work
2. Memory allocation
3. Observations

Observations

What we expected

Runtime system scaling

Memory placement

Use of h/w threads & caches

What we found

ORACLE

Observations

Going from small machines to larger will highlight extra places to remove contention.

we expected

Runtime system scaling

Memory placement

Use of h/w threads & caches

What we found

ORACLE

Observations

Going from small machines to larger will highlight extra places to remove contention.

we expected

Runtime system scaling

Memory placement

Use of h/w threads & caches

No surprises.

at we found

ORACLE

Observations

Going from small machines to larger will highlight extra places to remove contention.

NUMA effects will be significant on larger machines. Replicate data, control thread placement.

pected

Runtime system scaling	Memory placement	Use of h/w threads & caches

No surprises.

at we found

Observations

Runtime system scaling	Memory placement	Use of h/w threads & caches
Going from small machines to larger will highlight extra places to remove contention.	NUMA effects will be significant on larger machines. Replicate data, control thread placement.	
No surprises.	Balancing memory system load is more significant. Achieve close to per-socket limits when effective.	

Observations

Runtime system scaling	Memory placement	Use of h/w threads & caches
Going from small machines to larger will highlight extra places to remove contention.	NUMA effects will be significant on larger machines. Replicate data, control thread placement.	Low hit rate, but good fit with multi-threaded cores.
No surprises.	Balancing memory system load is more significant. Achieve close to per-socket limits when effective.	

ORACLE

Observations

Going from small machines to larger will highlight extra places to remove contention.	NUMA effects will be significant on larger machines. Replicate data, control thread placement.	Low hit rate, but good fit with multi-threaded cores.
Runtime system scaling	Memory placement	Use of h/w threads & caches
No surprises.	Balancing memory system load is more significant. Achieve close to per-socket limits when effective.	OOO execution is effective here, little benefit from more threads. Need to improve hit rate.

ORACLE

Observations

- As we have optimized these systems, many of our concerns come down to issues usually tackled in HPC:
  - Load balancing across 4K+ threads
  - Low-level h/w interactions, memory system hot spots, page sizes, …
  - Balancing resource utilization (CPU pipelines, DRAM, interconnect, …)
- New issues not always seen in HPC:
  - Fine-grained management of concurrent users
  - Resource management within servers
- Working from a DSL lets us mask much of this complexity

ORACLE

Further details

- Oracle Labs PGX project (Parallel Graph Analytics -- http://www.oracle.com/technetwork/oracle-labs/parallel-graph-analytics/overview/index.html)
- “Callisto-RTS: Fine-Grain Parallel Loops” – Tim Harris, Stefan Kaestle. USENIX ATC 2015
- “Pandia: Comprehensive Contention-Based Thread Placement” – Daniel Goodman, Georgios Varisteas, Tim Harris. EuroSys 2017 (ask me for a PDF)
- timothy.l.harris@oracle.com

ORACLE

Integrated Cloud

Applications & Platform Services

ORACLE®