GPU-Acceleration of In-Memory Data Analytics

Evangelia Sitaridi

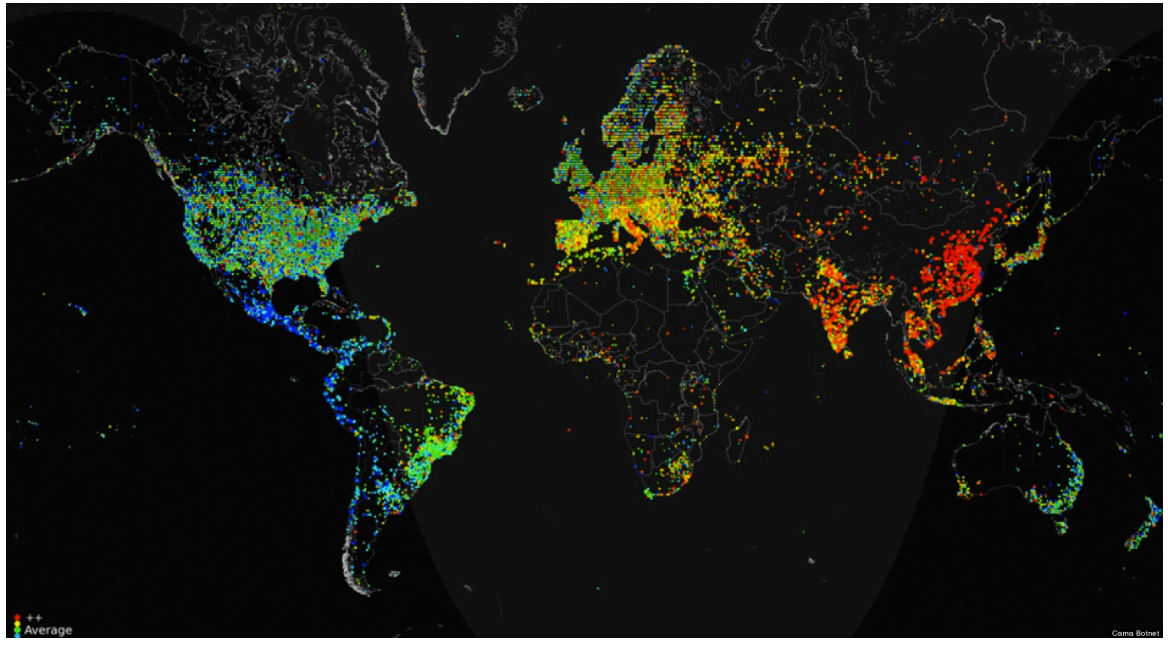
AWS Redshift

GPUs for Telcos

- Fast query-time
  - Quickly identify network problems
  - Respond fast to customers

No time to index data

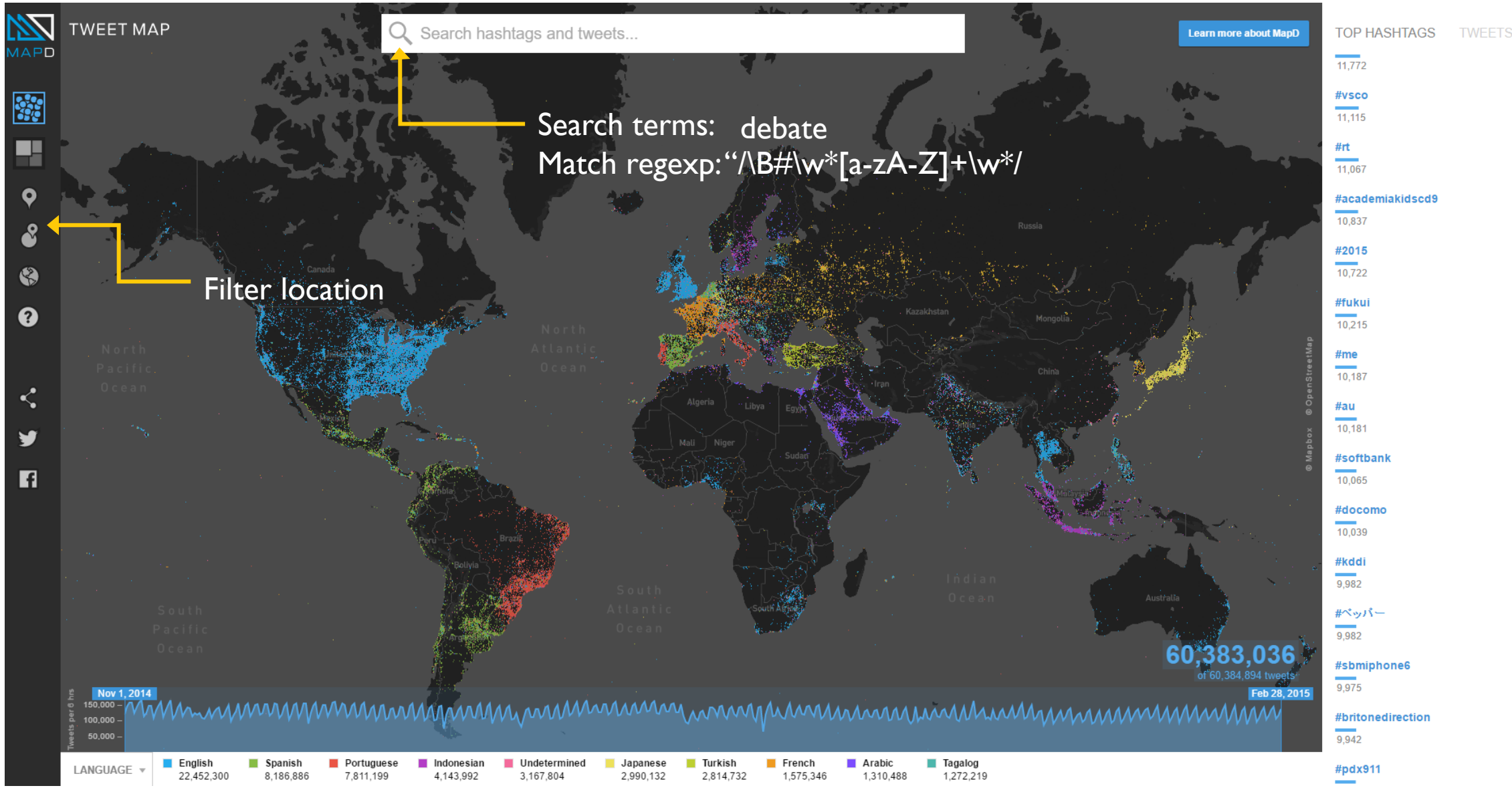
- Geospatial visualization
  - Take advantage of GPU visualization capabilities



SMS Hub traffic

*Picture taken from:
http://www.vizualytics.com/Solutions/Telecom/Telecom.html

GPUs for Social Media Analytics



Challenges for GPU Databases

- Special threading model → Increased programming complexity
  - Which algorithms more efficient for GPUs?
  - How much multiple code paths increase cost of code maintenance?
- Special memory architecture
  - How to adapt data layout?
- Limited memory capacity
  - Data transfer cost between CPUs and GPUs
    a) Through PCI/E link to the GPU
    b) From storage system to the GPU
- Fair comparison against software-based solutions

Challenges for GPU Databases

- Special threading model → Increased programming complexity
  - Which algorithms more efficient for GPUs?
  - How much multiple code paths increase cost of code maintenance?
- **Special memory architecture**
  - How to adapt data layout?
- **Limited memory capacity**
  - Data transfer cost between CPUs and GPUs
    a) Through PCI/E link to the GPU
    b) From storage system to the GPU
- Fair comparison against software-based solutions

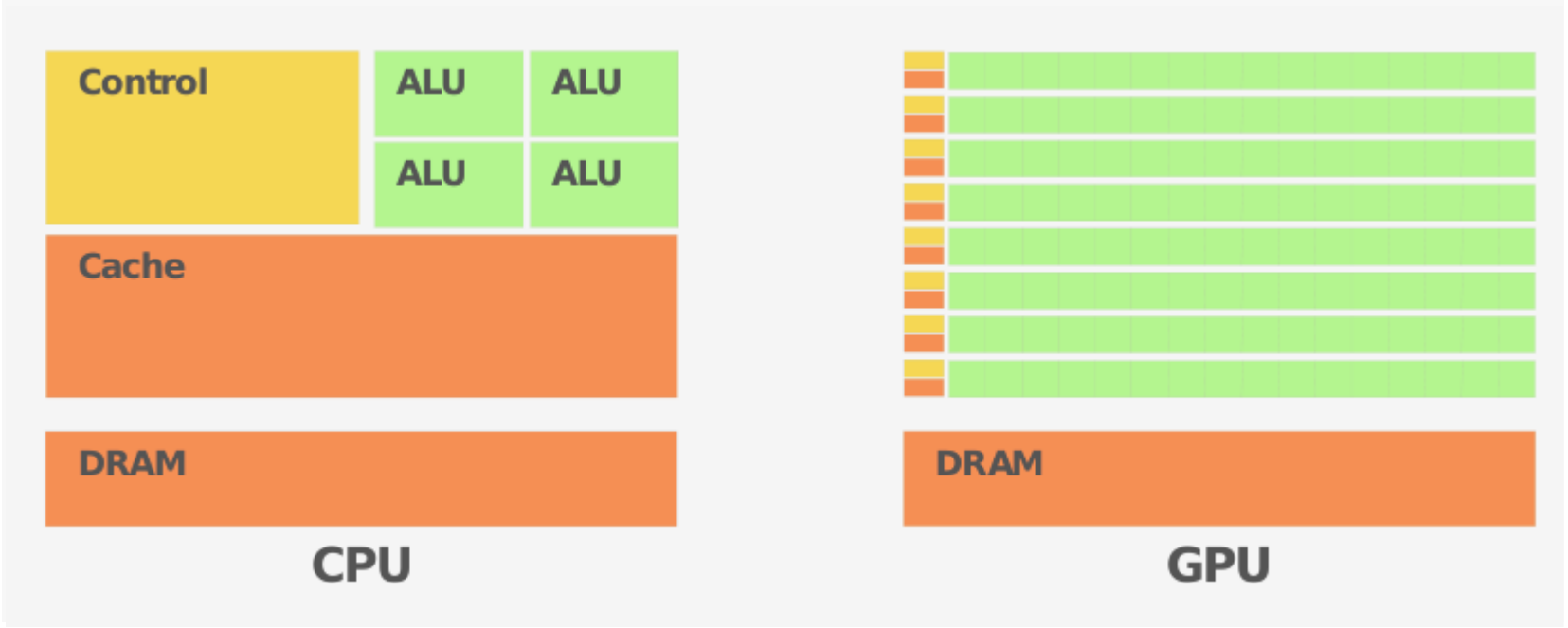
Outline

- **CPU vs GPU introduction**
- Accelerated wildcard string search
  - **Insight:** Change the layout of the strings in the GPU main memory
  - **3X** speed-up & **2X** energy savings against parallel state-of-the-art CPU libraries
- Gompresso: Massively parallel decompression
  - **Insight:** Trade-off compression ratio for increased parallelism
  - **2X** speed-ups & **1.2X** energy savings against multi-core state-of-the-art CPU libraries
- GPUs on the cloud

CPU-GPU Analogies

Goal: Low latency

Goal: High throughput
(overlapping different instructions)



CPU thread ⟷ GPU warp

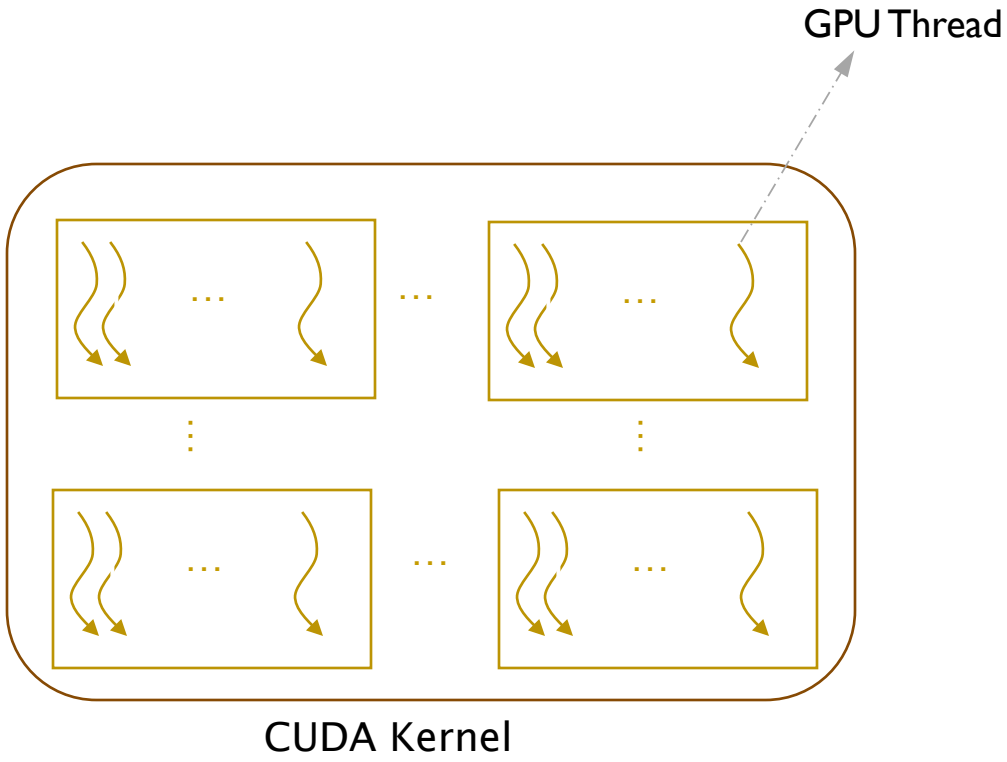
RAM ⟷ Global memory

Tens of threads ⟷ Thousands of threads

Hundreds of GBs capacity ⟷ Few tens of GB

GPU Architecture

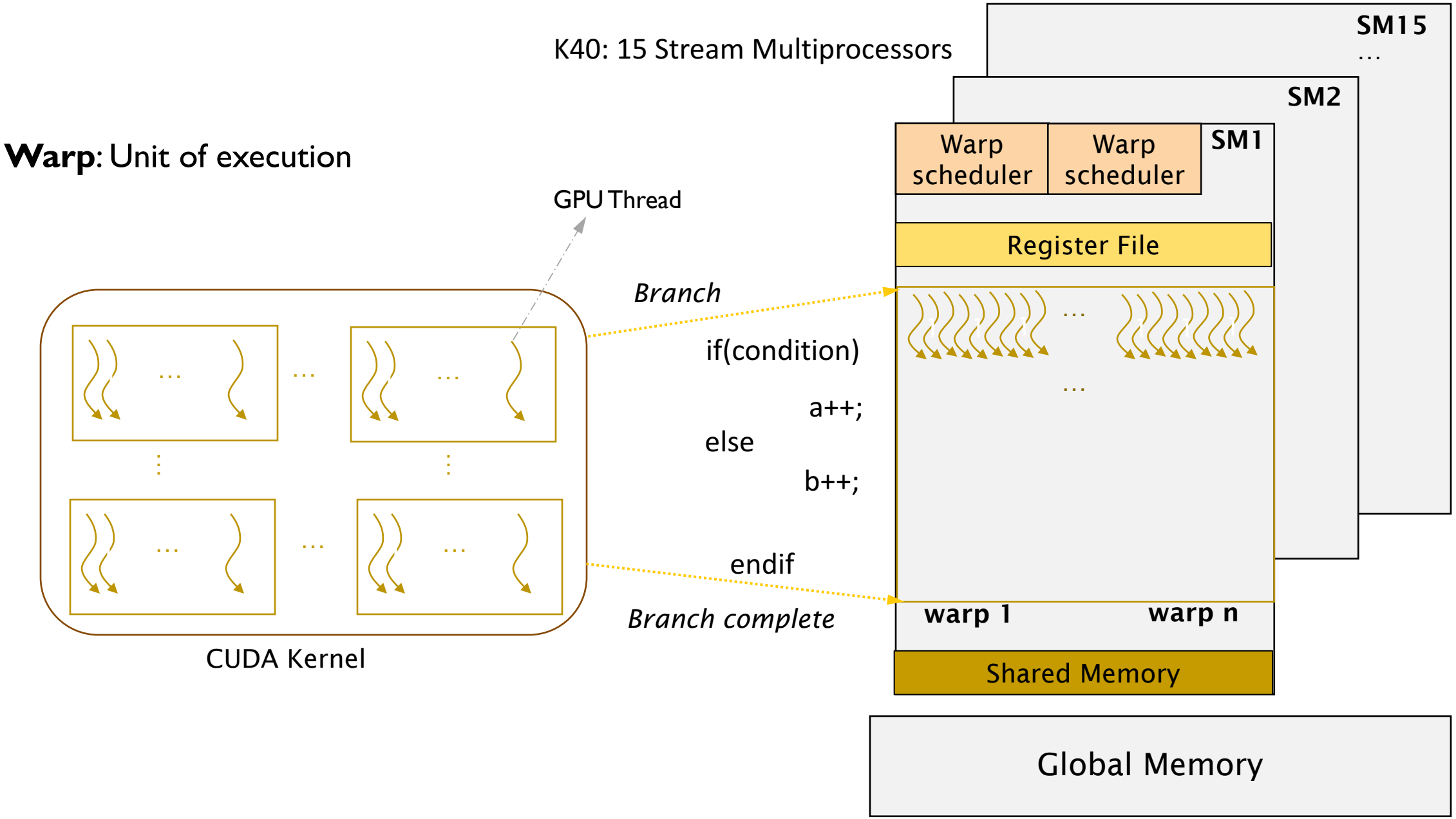
K40: 15 Stream Multiprocessors



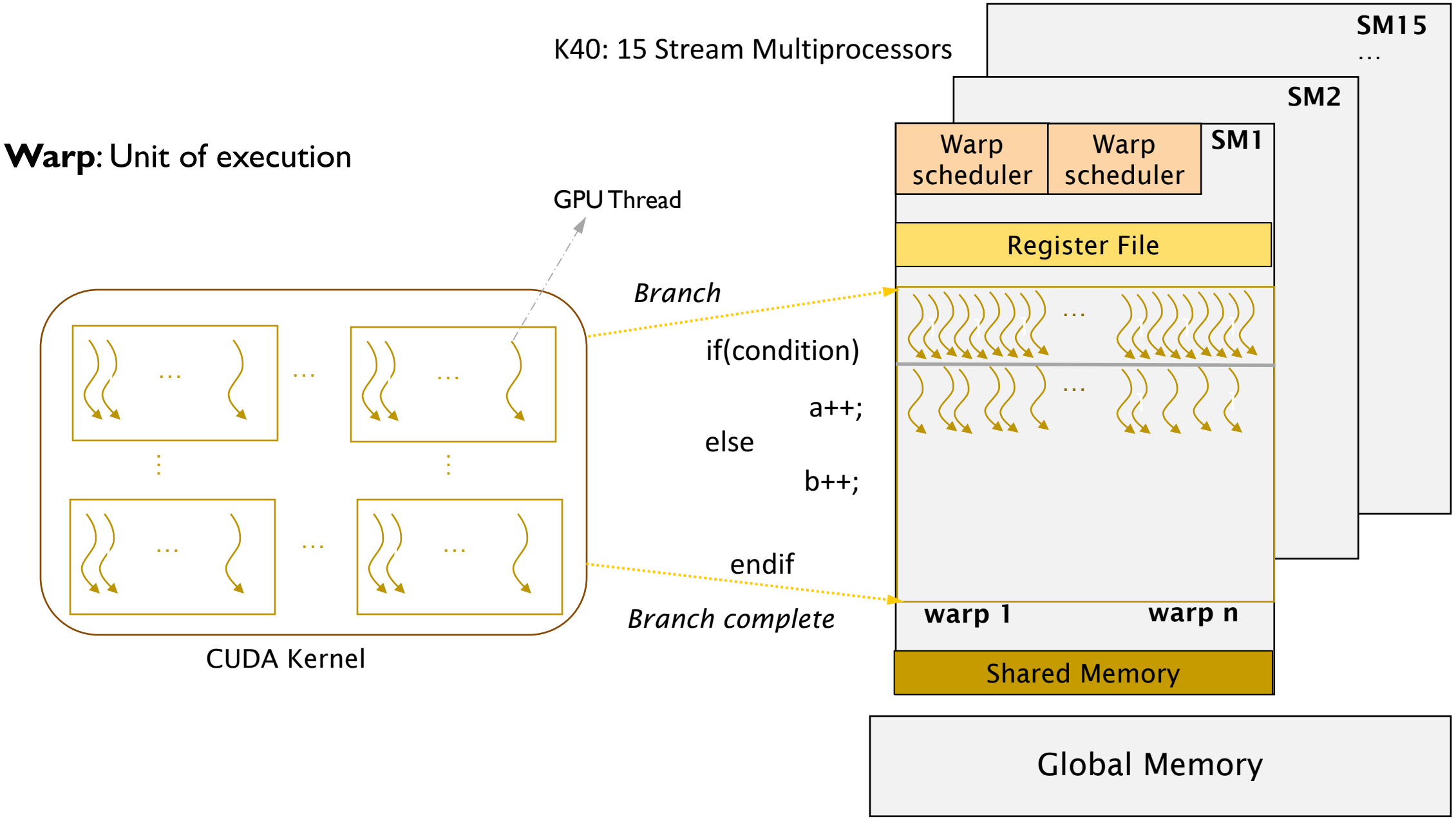

```
if(condition)
        a++;
else
        b++;

  endif
```

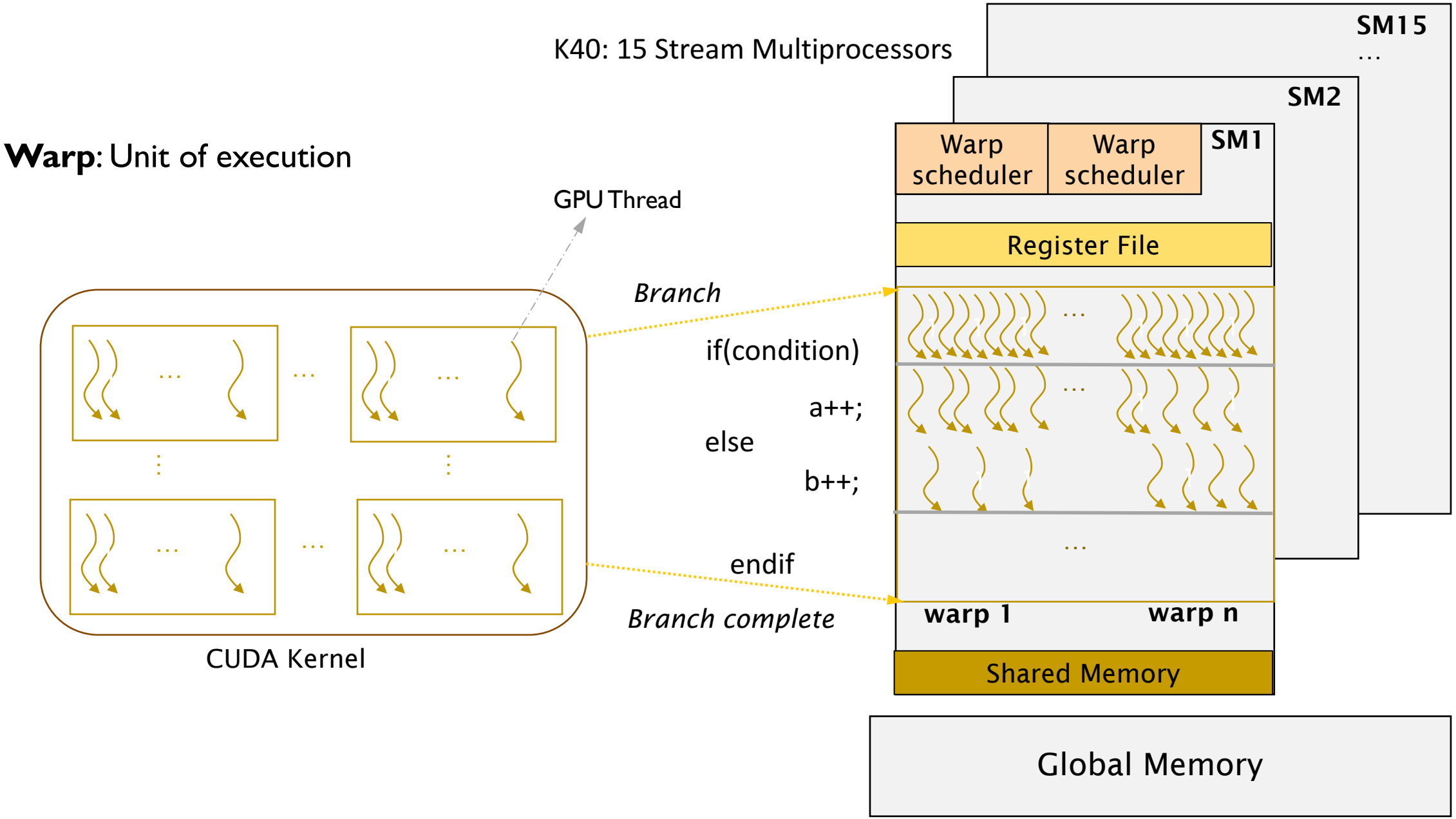
GPU Architecture



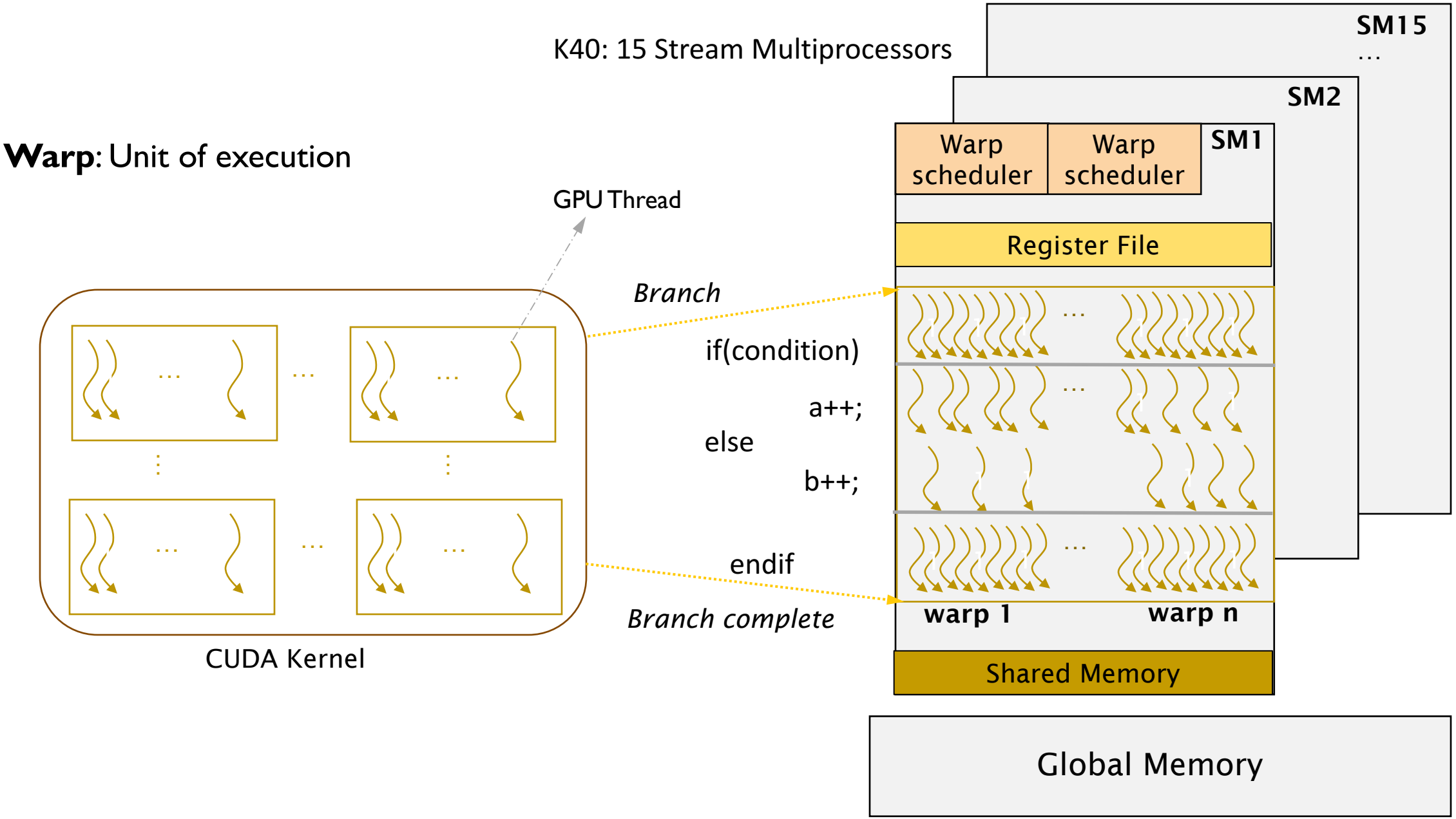
GPU Architecture



GPU Architecture



GPU Architecture



Outline

- CPU vs GPU introduction
- **Accelerated wildcard string search**
  - **Insight:** Change the layout of the strings in the GPU main memory
  - **3X** speed-up & **2X** energy savings against parallel state-of-the-art CPU libraries
- Gompresso: Massively parallel decompression
  - **Insight:** Trade-off compression ratio for increased parallelism
  - **2X** speed-ups & **1.2X** energy savings against multi-core state-of-the-art CPU libraries
- GPUs on the cloud

Text Query Applications

ACGTACCTGATCGTAGGATCCCAAGTACATCATTTC

Input

ACC

Search Pattern

GENOMIC DATA

Id	Address
3	"9 Front St, Washington DC, 20001"
8	"3338 A Lockport Pl #6, Margate City, NJ, 8402"
9	"18 3rd Ave, New York, NY, 10016"
15	"88 Sw 28th Ter, Harrison, NJ, 7029"
16	"87895 Concord Rd, La Mesa, CA, 91142"

Wild card searches

"*3rdAve*New York*"

Search Pattern

DATABASE COLUMNS

Q2,9,13,14,16,20 of TPC-H contain expensive LIKE predicates

Wildcard Search Challenges

- Approaches simplifying search cannot be applied
  - String indexes, e.g. suffix trees
    - For query '%customer%complaints' multiple queries need be issued
      - '%customer%' AND '%complaints%'
      - Confirm results
  - Dictionary compression
    - Wildcard searches not simplified using dictionaries
    - String data need to be decompressed

Background: How to Search Text Fast?

Knuth-Morris-Pratt Algorithm

Input: ACACA**T**ACCTACTTTACGTACGT **Step 6** i=5 j=5

Pattern: ACACA**C**G Character mismatch

Shift pattern table -1 0 0 1 2 3 4

Advance to the next character:

a) If the input matches to the pattern

b) While there is a mismatch shift to the left of the pattern

Stop when the beginning of the pattern has been reached

Background: How to Search Text Fast?

Knuth-Morris-Pratt Algorithm

Input: ACACA**T**ACCTACTTTACGTACGT

Pattern: ACACA**C**G

Step 6 i=5 j=5

Character mismatch

ACACA**T**ACCTACTTTACGTACGT

ACA**C**ACG

Step 7 i=5 j=1

Shift pattern

Shift pattern table -1 0 0 1 2 3 4

Advance to the next character:

a) If the input matches to the pattern

b) While there is a mismatch shift to the left of the pattern

Stop when the beginning of the pattern has been reached

GPU Limiting factor: Cache Pressure

Threads matching different strings

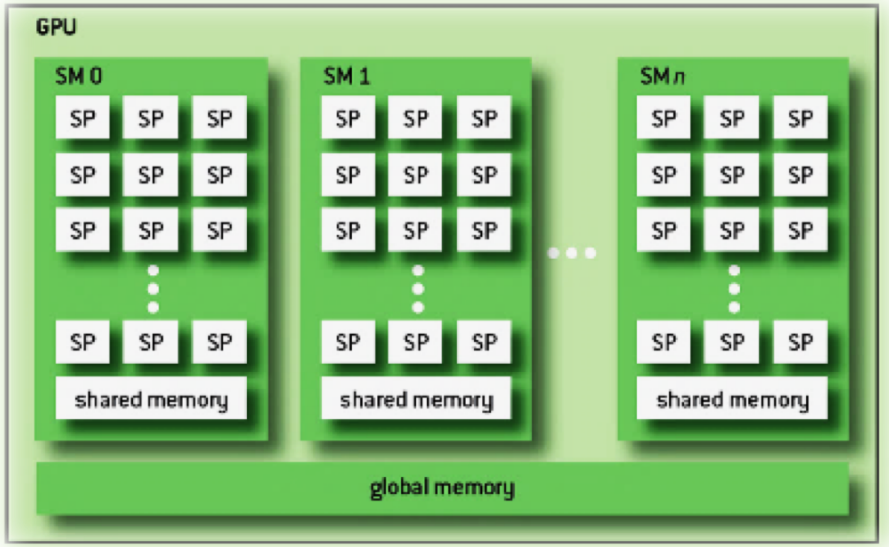
Warp size: 32
Stream Multiprocessors: 15
#Warps in each SM: 64

x

Cache footprint: 30720 cache lines

>>

L2 Capacity: 12288 cache lines



Tesla K40 architecture

Smaller cache size per thread than CPUs: Need improved locality!

Adapt Memory Layout: Pivoting Strings

Baseline (contiguous) layout

String 1 **String 2** **String 3**

CTAACCGAGTAAAGAACGTAAACTCATTCGACTAAACCGAGTAAAGA…

Pivoted layout

CTAAACGTCTAA…CCGAAAACACCG…GTAATCATAGTA…AAGATCGAAAGA…

- Split strings in equally sized pieces
- Interleave pieces in memory → Improve locality

Initially: Each warp loads a **cache line** (128 bytes)

CTAAACGTCTAA…CCGAAAACACCG…GTAATCATAGTA…AAGATCGAAAGA…

T0 T1 T2

Partial solution: Threads might progress in different rate

Adapt Memory Layout: Pivoting Strings

Baseline (contiguous) layout

String 1 **String 2** **String 3**

CTAACCGAGTAAAGAACGTAAACTCATTCGACTAAACCGAGTAAAGA…

Pivoted layout

CTAAACGTCTAA…CCGAAAACACCG…GTAATCATAGTA…AAGATCGAAAGA…

- Split strings in equally sized pieces
- Interleave pieces in memory → Improve locality

In presence of partial matches some threads might fall "behind"

Partial solution: Threads might progress in different rate

Transform Control Flow of KMP

Shift pattern table -1 0 0 1 2 3 4

Knuth-Morris-Pratt Algorithm

Input: ACACA**T**ACCTACTTTACGTACGT

Pattern: ACACA**C**G

Step 6 i=5 j=5

Character mismatch

While Loop

ACACA**T**ACCTACTTTACGTACGT
ACA**C**ACG

i=5 j=3

Mismatch→ Shift pattern

ACACA**T**ACCTACTTTACGTACGT
A**C**ACACG

i=5 j=1

Mismatch→ Shift pattern

…

ACACA**T**ACCTACTTTACGTACGT
ACACA**C**G

Step 7 i=6 j=0

Shift pattern

KMP Hybrid: Advance input in pivoted piece size

GPU vs. CPU Comparison

```
select s_suppkey
from supplier
where s_comment like '%Customer%Complaints%'
```

- Performance Metrics
  - Price ($)
  - Performance (GB/s)
  - Performance per $
  - Estimated energy consumption
- Evaluate three systems
  - CPU only system
  - GPU only system
  - CPU+GPU combined system

GPU vs. CPU Comparison

	GPU	CPU (Boost BM)	CPU (CMPISTRI)	CPU+GPU
Price ($)	3100	952	952	4052
Performance (GB/s)	98.7	40.75	43.1	138.7
Energy consumed (J)	1.27	2.49	2.35	1.78
Performance/$	31.89	42.8	45.28	34.25

Circle best column value per row

CPU: Dual-socket E5-2620 – Band. 102.4 GB/s
GPU: Tesla K40 – Band. 288 GB/s

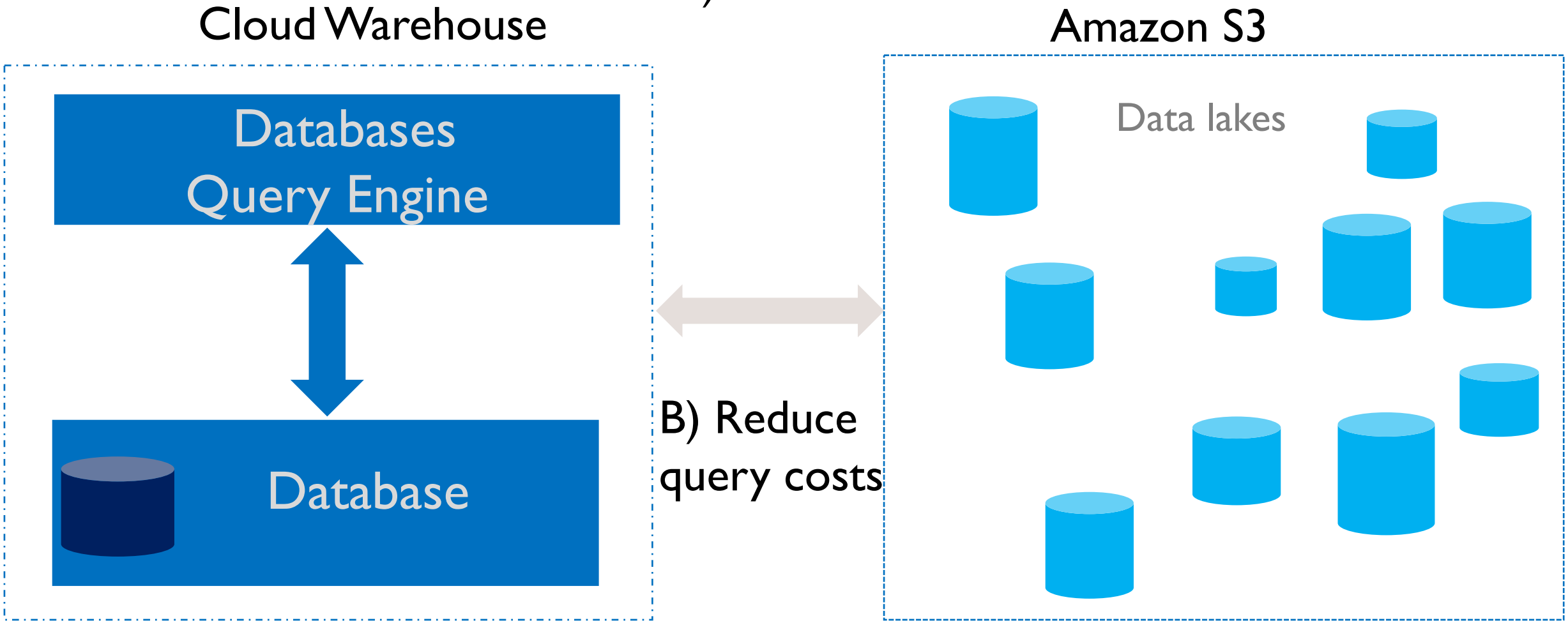
Design system by choosing the desired trade-offs

Outline

- CPU vs GPU introduction
- Accelerating wildcard string search
  - **Insight:** Change the layout of the strings in the GPU main memory
  - **3X** speed-up & **2X** energy savings against parallel state-of-the-art CPU libraries
- **Gompresso: Massively parallel decompression**
  - **Insight:** Trade-off compression ratio for increased parallelism
  - **2X** speed-ups & **1.2X** energy savings against multi-core state-of-the-art CPU libraries
- GPUs on the cloud

Example: Why Use Compression?

A) Reduce basic S3 costs



Decompression speed more important than compression speed

Background: LZ77 Compression

Input characters

0 1 2 3 …
ATTACTAGAATGT TACTAATCTGAT
CGGGCCGGGCCTG

Output

*ATTACTAGAATGT***(2,5)…**

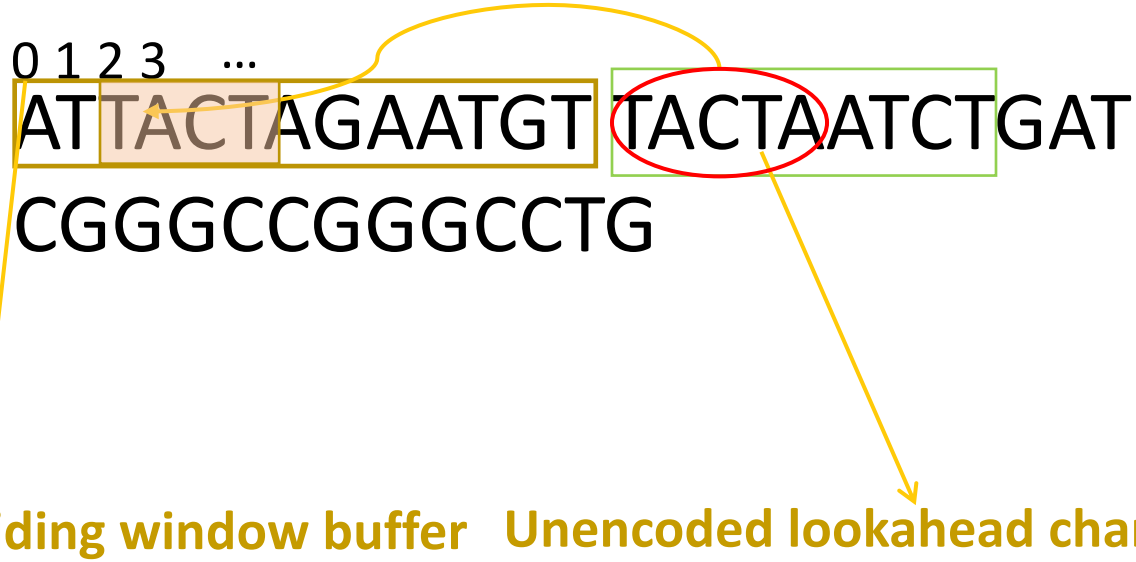
Literals
Unmatched characters

Backreferences
(Position, Length)

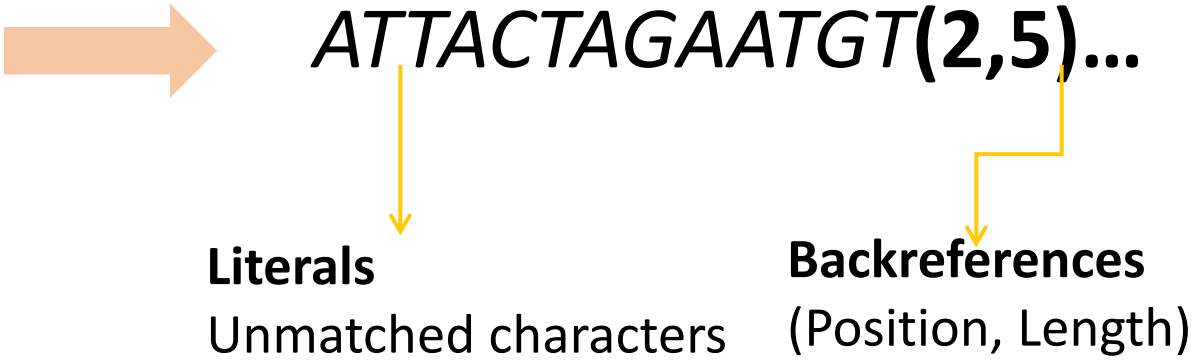
Background: LZ77 Compression

Input characters

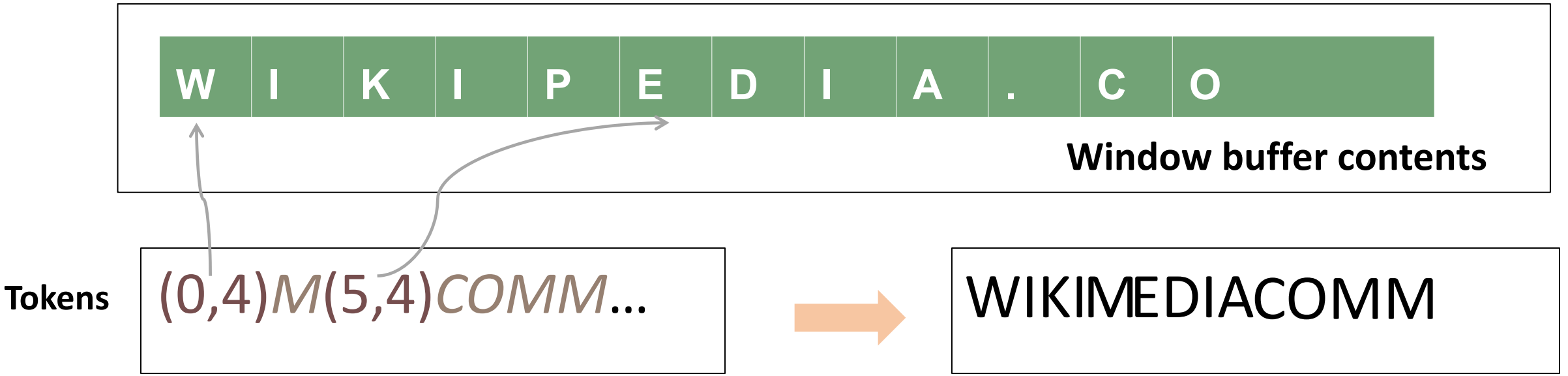
Find longest match



Output

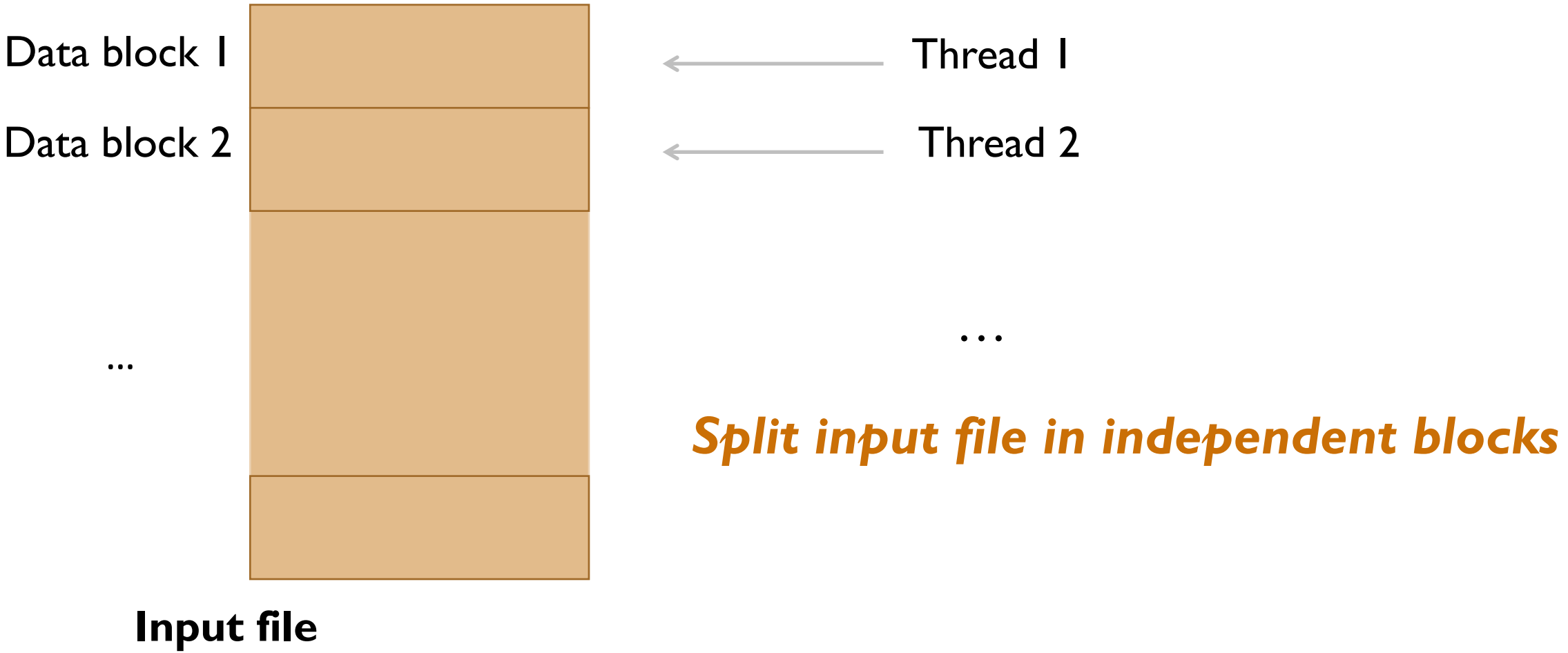


Background: LZ77 Decompression



How to Parallelize Decompression?

>1000 threads available!

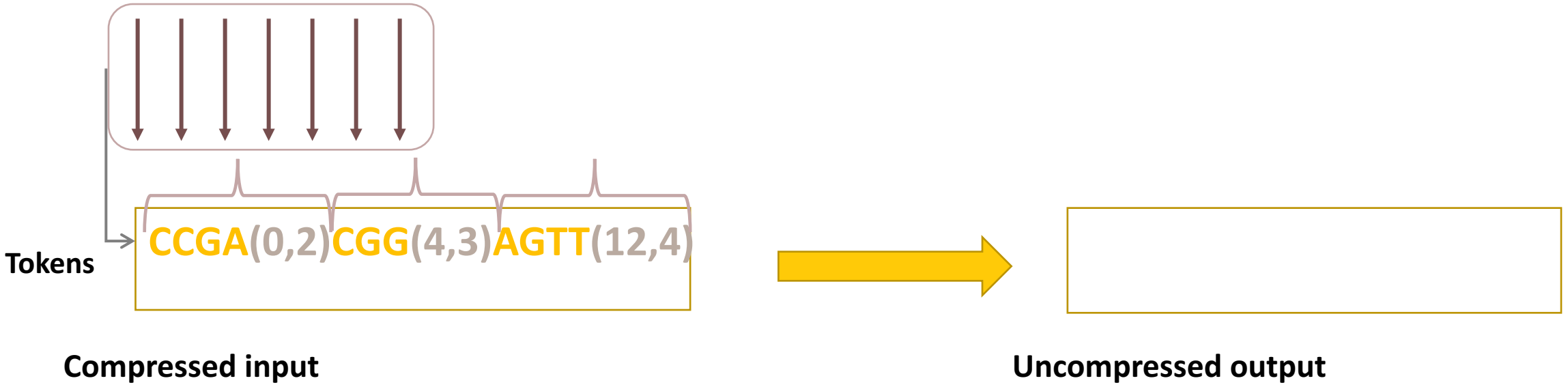


Split input file in independent blocks

Naïve approach performance 200 MB/s << 250 GB/s (K20x)

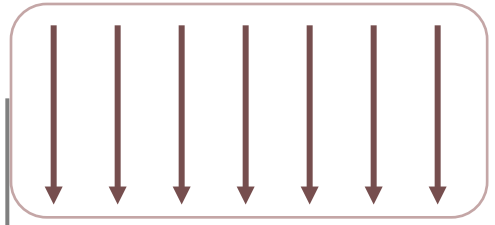
GPU LZ77 Decompression

Improve utilization: Group strings of literals with the following back-reference



GPU LZ77 Decompression

1) Read tokens (parallel)



Tokens

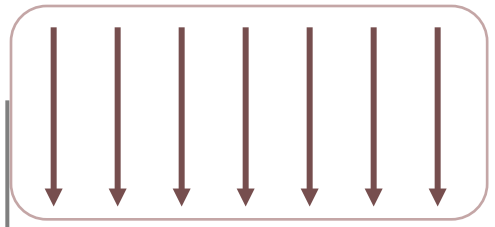
CCGA(0,2)CGG(4,3)AGTT(12,4)

Compressed input

Uncompressed output

GPU LZ77 Decompression

2) Write literals (parallel prefix sum)



Tokens

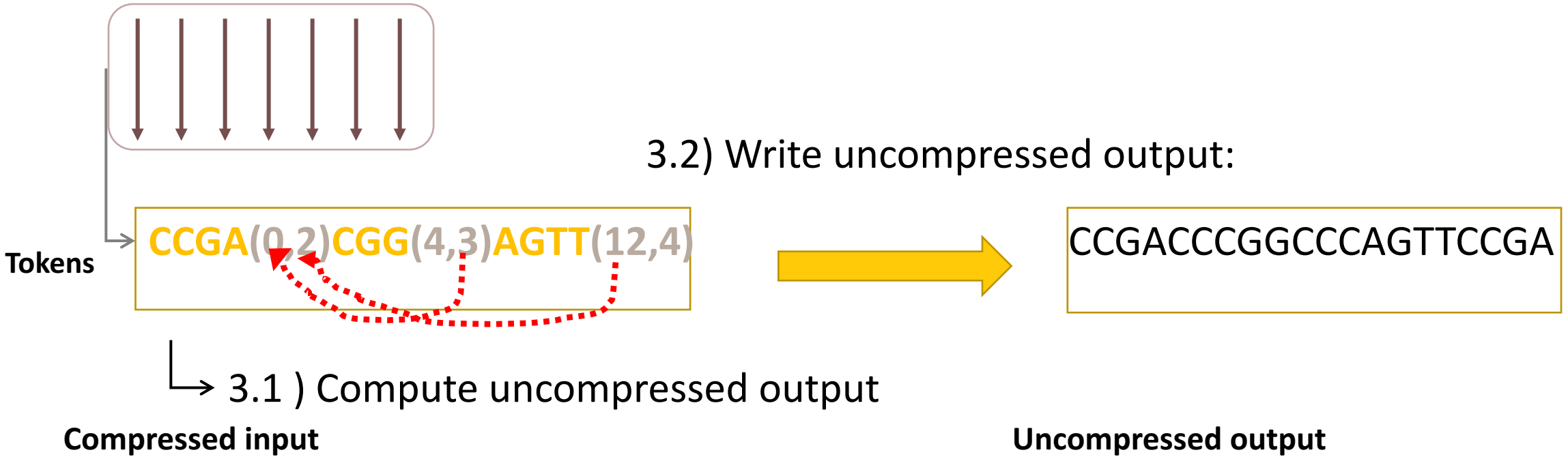
CCGA(0,2)CGG(4,3)AGTT(12,4)

CCGA CGG AGTT

Compressed input

Uncompressed output

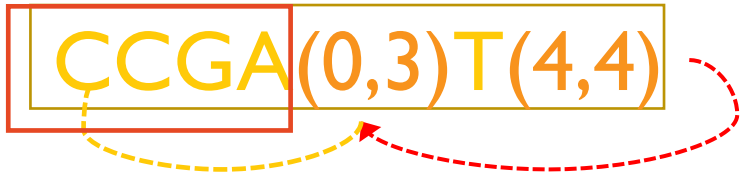
GPU LZ77 Decompression



Problem: Back-references processed in parallel might be dependent! →
Use voting function __*ballot* to detect dependencies

How to Handle Thread Dependencies?

MRR

Tokens 

Second loop: Dependencies satisfied

1) Write literals (parallel)
2) While(!all backreferences written)
   a) Check dependencies satisfied (parallel)
   b) Copy back-references w/o pending dependencies

Bandwidth: 7 GB/s

DE

Tokens 

A) Compression

Only search for matches w/o dependencies

B) Decompression

Copy back-references (fully parallel)

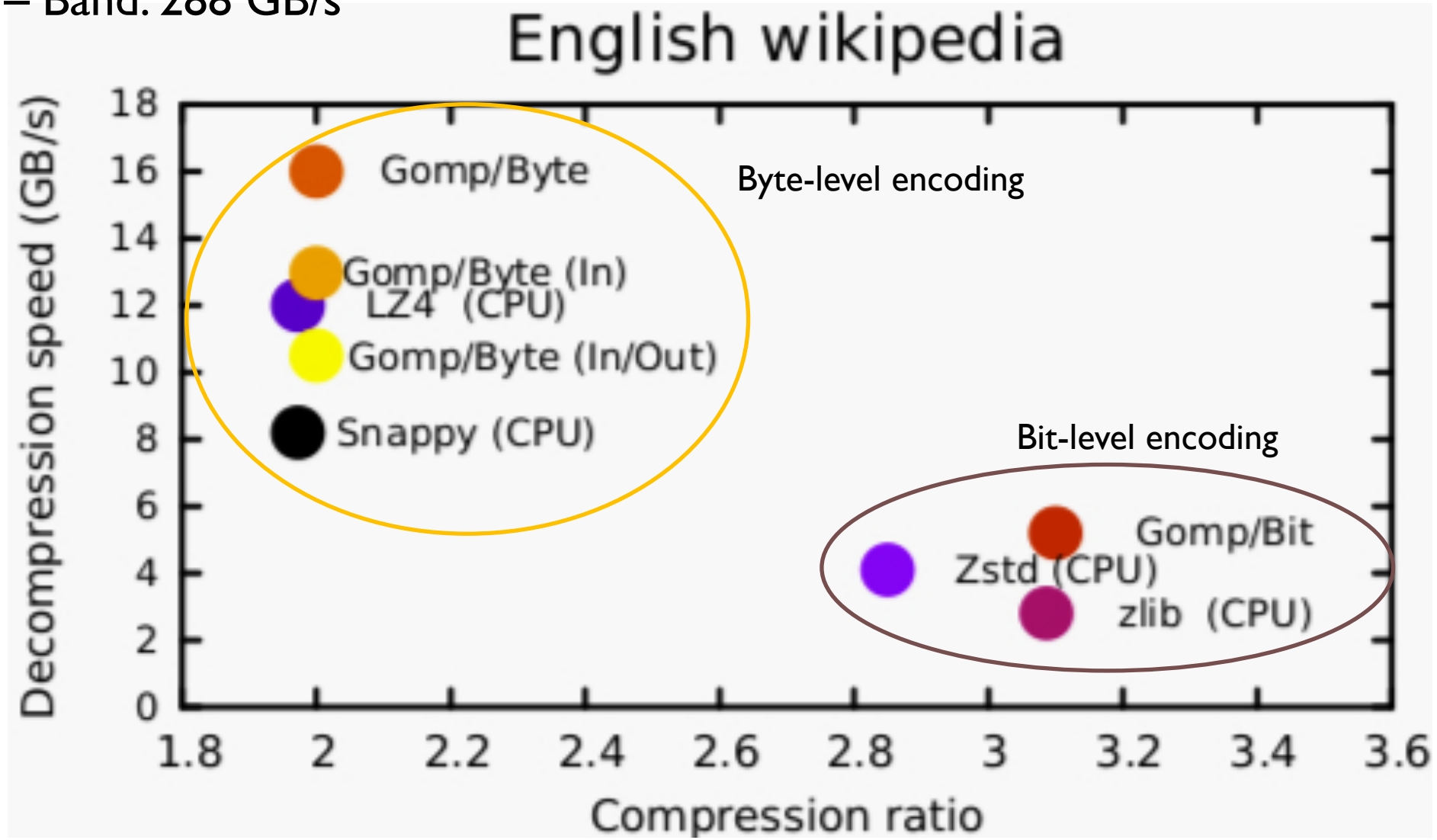
Bandwidth: 16 GB/s

Uncompressed input …CCGACGTTCCGT…

Decompression Skyline

CPU: Dual-socket E5-2620 – Band. 102.4 GB/s

GPU: Tesla K40 – Band. 288 GB/s



GPUs on the Cloud

- Cloud offerings
  - AWS
  - Google Cloud
  - Microsoft Azure
  - IBM Softlayer
  - Nimbix
- Opportunity
  - Evaluate the usefulness of GPUs/FPGAs without the high investment
- Special considerations
  - Charging model
  - Scaling capabilities
  - Software licensing

Summary

- Accelerated wildcard string search
  - **Insight:** Change the layout of the strings in the GPU main memory
  - **3X** speed-up & **2X** energy savings against parallel state-of-the-art CPU libraries
- Gompresso: Massively parallel decompression
  - **Insight:** Trade-off compression ratio for increased parallelism
  - **2X** speed-ups & **1.2X** energy savings against multi-core state-of-the-art CPU libraries
- GPUs on the cloud: Open questions