Capability
Jerarquía de memoria

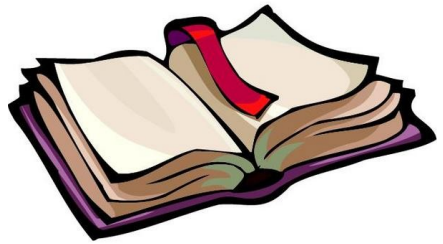
Universidad Nacional de La Plata

Facultad de Informática

Prof. Dr. Adrian Pousa

Agenda

I. CUDA Capability
II. Características generales de la jerarquía de memoria Nvidia
III. Descripción de las distintas memorias de la jerarquía NVidia
    i. Memoria global
    ii. Memoria de constantes
    iii. Memoria de texturas
    iv. Memoria compartida
    v. Registros
    vi. Memoria local
IV. Resumen
V. Limitaciones



Agenda

I. CUDA Capability

II. Características generales de la jerarquía de memoria Nvidia

III. Descripción de las distintas memorias de la jerarquía NVidia

   i. Memoria global

   ii. Memoria de constantes

   iii. Memoria de texturas

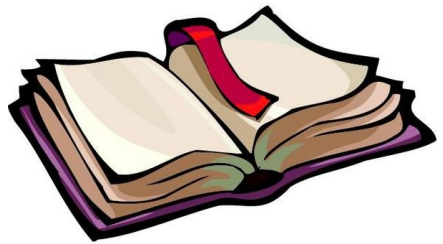
   iv. Memoria compartida

   v. Registros

   vi. Memoria local

IV. Resumen

V. Limitaciones



CUDA Capability

- Una **capability** indica la generación de una GPU cuales son sus posibilidades y sus limitaciones. Una nueva capability agrega funcionalidad y tiene compatibilidad hacia atrás.

- Una capability se representa por dos dígitos (R.V), el primero es la revisión (R) y el segundo es una versión (V). El mismo número de revisión indica la misma arquitectura.

Capability	Arquitectura
1.0	G80
2.0, 2.1	Fermi
3.0, 3.1, 3.5	Kepler
5.0, 5.2, 5.3	Maxwell
...	...
8.0	Ampere

CUDA Capability

- Una **capability**[1] nos muestra distintos límites:

	Capability									
Especificaciones Técnicas	1.0	1.1	1.2	1.3	2.x	3.0	3.5	3.7	5.0	5.2
Máxima dimensión de un grid.	2				3					
Máxima dimensión x de un grid.	65535					$2^{31} - 1$				
Máxima dimensión y o z de un grid.	65535									
Máxima dimensión de un bloque.	3									
Máxima dimensión x o y de un bloque.	512				1024					
Máxima dimensión z de un bloque.	64									
Máximo número de hilos por bloques.	512				1024					

[1] http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#compute-capabilities

CUDA Capability
Capability en las GPUs de la cátedra

- **Capabilities del dispositivo[2].**

Placa	Arquitectura	Capability
GTX 560TI	Fermi GF114	2.1
Tesla C2075	Fermi GF110GL	2.0
GTX 960	Maxwell GM206-300	5.2
RTX 2070	Turing lf02	7.5
RTX 3070	Ampere GA104	8.6
RTX 4090	Ada Lovelace AD102	8.9

[2]https://developer.nvidia.com/cuda-gpus

Agenda

I. CUDA Capability

II. Características generales de la jerarquía de memoria Nvidia

III. Descripción de las distintas memorias de la jerarquía NVidia

   i. Memoria global

   ii. Memoria de constantes

   iii. Memoria de texturas

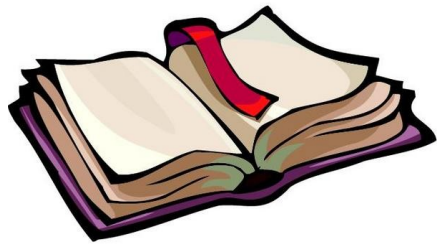
   iv. Memoria compartida

   v. Registros

   vi. Memoria local

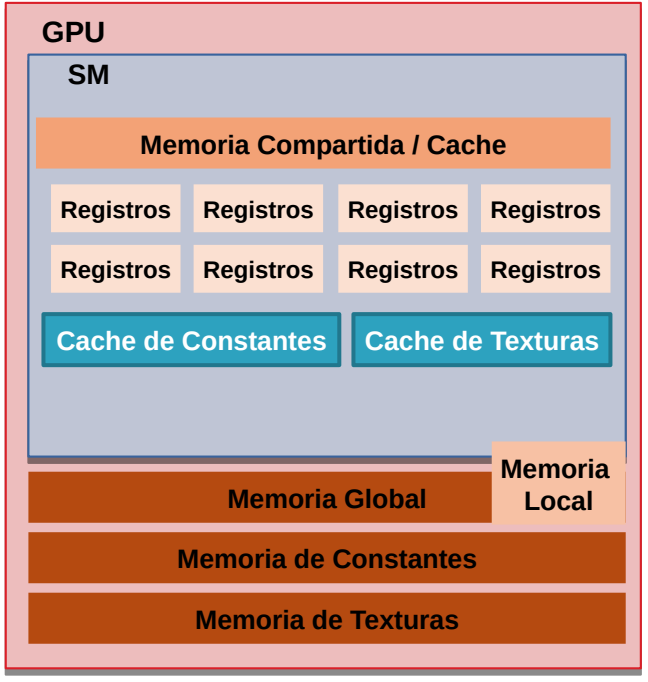
IV. Resumen

V. Limitaciones



Jerarquía de memoria
Características generales

- Los hilos pueden acceder a datos que están en memorias:
  - **On-Chip:** Dentro de cada SM
    - Registros
    - Memoria compartida (shared)
    - Distintas caché
  - **Off-Chip:** Fuera de los SMs
    - Memoria global
    - Memoria de constantes
    - Memoria de texturas
    - Memoria local



Jerarquía de memoria
Diferencias de velocidad entre memorias

- Desde el punto de vista de la arquitectura, cada nivel de memoria se diferencia en:
  - Tamaño (GB a bits).
  - Ancho de banda (TB a Mb).
  - Velocidad de acceso (Off-chip vs. On-chip).
- De acuerdo a la velocidad de acceso, la jerarquía se ordena desde la memoria mas lenta a la mas rápida como:

Memoria Global
Memoria Local

Memoria de Constantes
Memoria de Texturas

Memoria Compartida
Registros
Cachés

Jerarquía de memoria
Características de las variables

- Desde el punto de vista del programador, éste decide las características de las variables y en qué memoria las aloja.

- Esto determina:
  - La velocidad de acceso
  - La visibilidad (CPU y/o GPU)
  - El alcance
  - El tiempo de vida

Jerarquía de memoria
Alcance y tiempo de vida

- El **alcance** establece qué hilos acceden a una variable:
  - Un único hilo
  - Todos los hilos de un bloque
  - Todos los hilos de un grid
  - Todos los hilos de una aplicación
- El **tiempo de vida** indica la duración de la variable en la ejecución del programa:
  - **La ejecución de un kernel:** la variable se crea durante la ejecución de un kernel y se destruye cuando el kernel finaliza
  - **Toda la aplicación:** Permanece y es accesible por todos los kernels involucrados en la aplicación

Jerarquía de memoria
Latencia y ocultamiento de la latencia

- La latencia de una instrucción de memoria varía dependiendo de:
  - El espacio de memoria al que se accede
  - El patrón de acceso

- Los hilos pueden demorarse al ejecutar una instrucción de memoria dejando al SM ocioso. La GPU **oculta esa latencia** realizando cambios de contexto (**Multithreading por hardware**) cargando otro warp en el SM.

Agenda

I. CUDA Capability

II. Características generales de la jerarquía de memoria Nvidia

III. Descripción de las distintas memorias de la jerarquía NVidia

   i. Memoria global

   ii. Memoria de constantes

   iii. Memoria de texturas

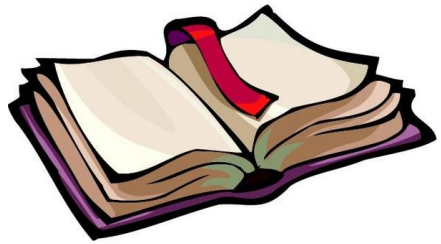
   iv. Memoria compartida

   v. Registros

   vi. Memoria local

IV. Resumen

V. Limitaciones



Agenda

I. CUDA Capability

II. Características generales de la jerarquía de memoria Nvidia

III. Descripción de las distintas memorias de la jerarquía NVidia

   i. Memoria global

   ii. Memoria de constantes

   iii. Memoria de texturas

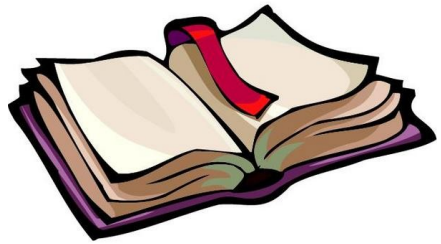
   iv. Memoria compartida

   v. Registros

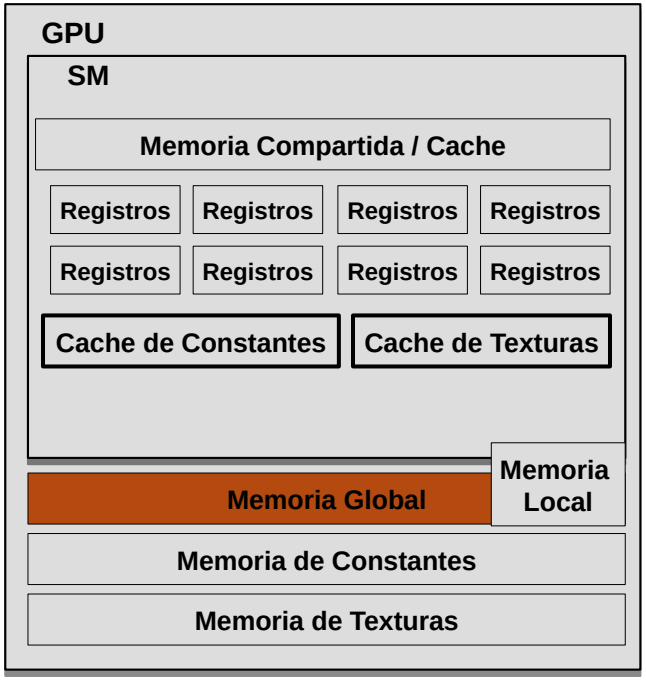
   vi. Memoria local

IV. Resumen

V. Limitaciones



Memoria Global



Visibilidad

Lectura y escritura tanto por CPU como GPU.

Velocidad

Es una memoria off-chip, por lo tanto su acceso es lento.

Variables

Alcance

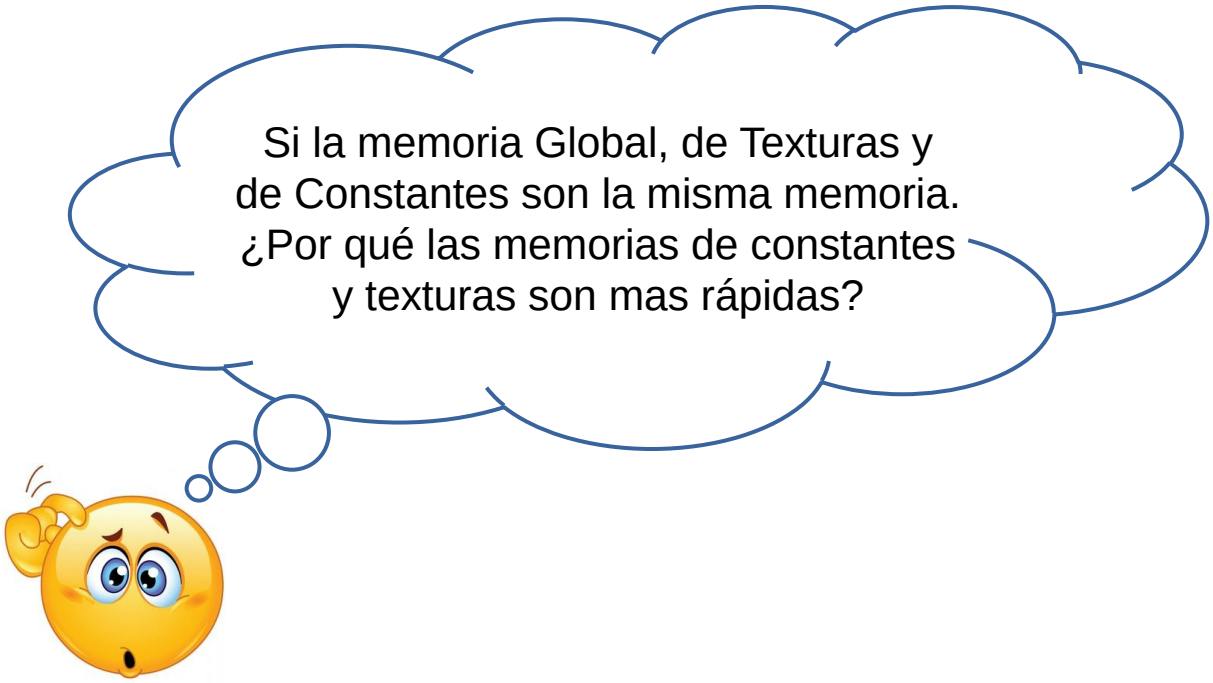
Todos los hilos de un grid y/o de una aplicación.

Tiempo de vida

Toda la aplicación

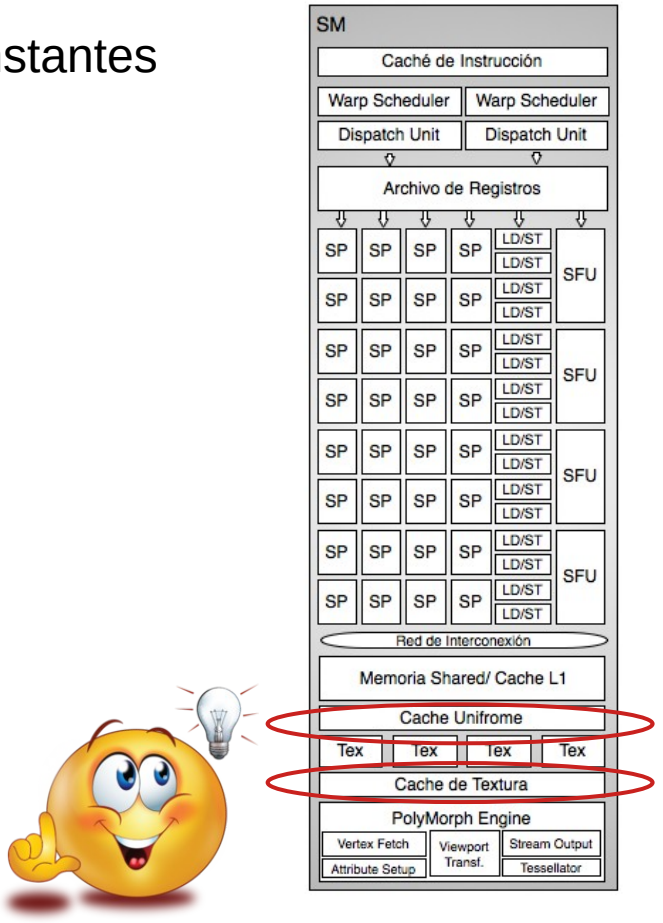
Memoria Global

- La memoria global se divide en tres espacios de memoria independientes:
  - Memoria Global ppd.
  - Memoria de Textura
  - Memoria de Constantes



Memoria Global

- Tanto la memoria de textura como la memoria de constantes poseen caché on-chip.
- Además de ser de sólo lectura por los hilos de GPU.



Memoria Global
Variables en memoria global

- Todos los hilos pueden acceder a una variable global (`__device__`)
- Una variable global puede verse como una variable compartida por todos los hilos del grid y pueden utilizarse como medio de comunicación entre hilos de distintos bloques (**no habitual por bajo rendimiento**).
- El **tiempo de vida** de una variable global es **toda la aplicación**: una vez finalizado un kernel los valores de las variables globales persisten y pueden utilizarse por siguientes invocaciones a kernels.

Memoria Global
Patrón de acceso

- Los accesos a la memoria global se resuelven de a **medio warp**:
  - Un requerimiento a memoria de un warp se divide en dos,un requerimiento para la primera mitad del warp y otro para la segunda mitad.
- El acceso de medio warp a memoria global es lento pero, dependiendo del **patrón de acceso**, el hardware puede aumentar la eficiencia incrementando la velocidad a partir de dos tareas específicas:
  - Unificar transacciones
  - Minimizar la cantidad de datos a traer
- El tamaño del segmento de memoria es variable y puede ser:
  - **32 bytes:** si todos los hilos acceden a palabras de **1 byte**
  - **64 bytes:** si todos los hilos acceden a palabras de **2 bytes**
  - **128 bytes:** si todos los hilos acceden a palabras de **4 bytes**

Memoria Global
Patrón de acceso

- **Unificar transacciones**:
  - Si los hilos del medio warp acceden a **N segmentos de memoria** distintos, entonces se producen N transacciones.
  - En cambio, si los hilos acceden al **mismo segmento de memoria**, las transacciones pueden **unificarse**
- **Minimizar la cantidad de datos a traer:**
  - Si los hilos del medio warp no acceden a todos los datos del segmento, entonces se leen datos que no se usan desperdiciando ancho de banda.
  - El hardware puede **acotar la cantidad de datos** a traer dentro del segmento, pudiendo traer subsegmentos de 32 bytes o 64 bytes.

Memoria Global
Patrón de acceso – Ejemplo

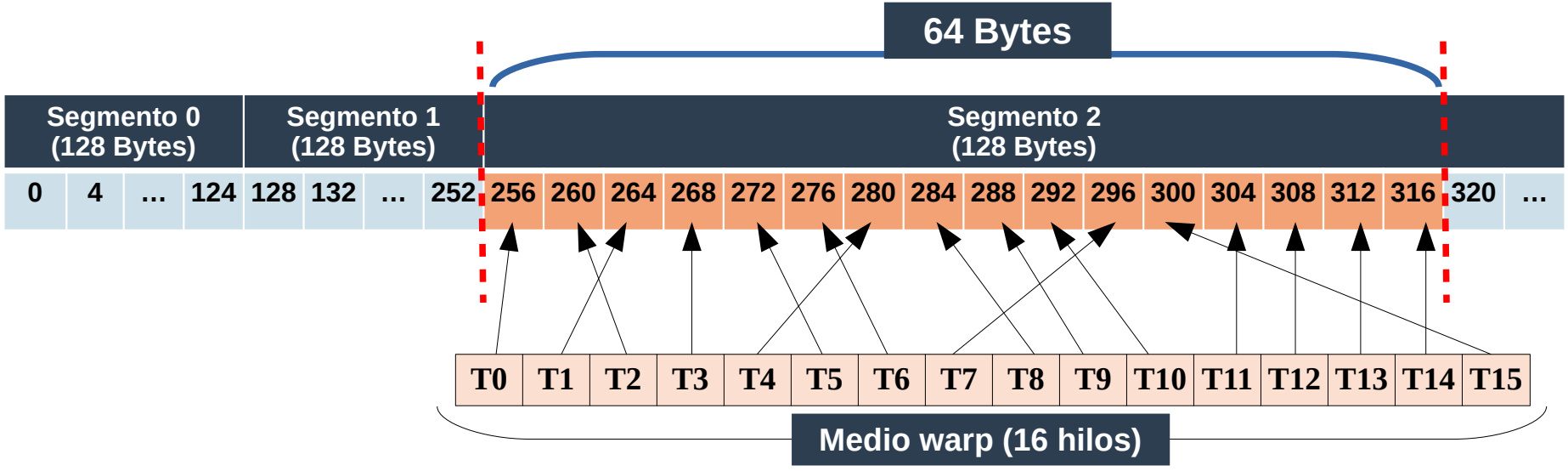
- Supongamos una memoria global organizada de la siguiente manera:
  - Palabras de 4 Bytes
  - Segmentos de 128 Bytes

Segmento 0 Direcciones (0-127)				Segmento 1 Direcciones (128-255)				Segmento ...				
0	4	…	124	128	132	…	252	256	...	...	...	…

Memoria Global
Patrón de acceso – Ejemplo: Caso 1

- Los hilos de medio warp acceden a 16 posiciones consecutivas alineadas con un segmento de 128 Bytes.

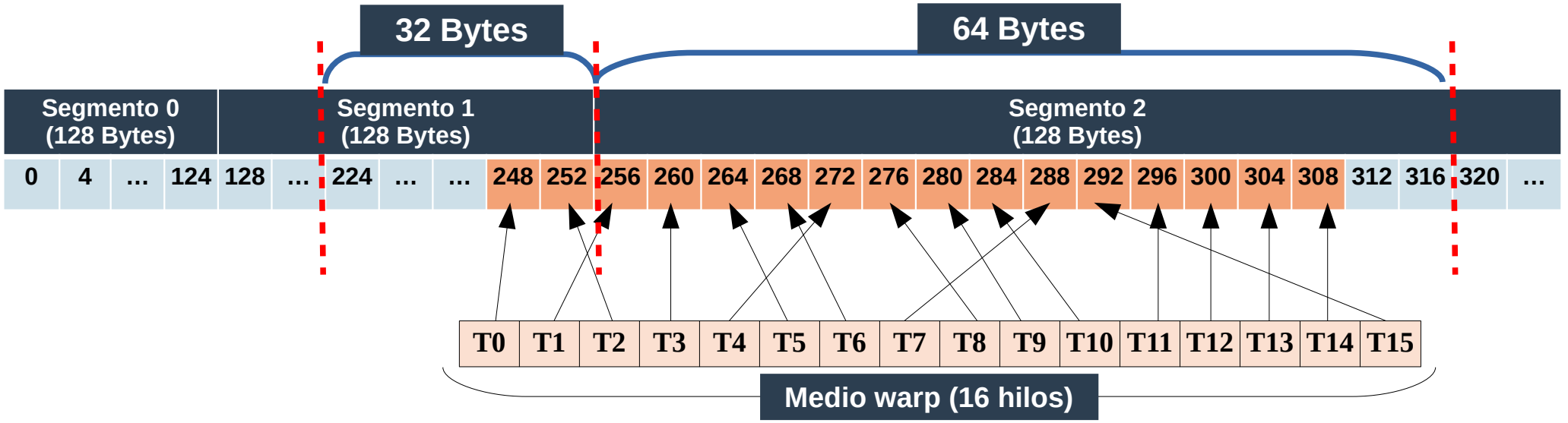
- El hardware genera una única transacción (**unificación**) y trae sólo 64 Bytes para evitar leer datos innecesarios (**minimización**).



Memoria Global
Patrón de acceso – Ejemplo: Caso 2

- Los hilos de medio warp acceden a 16 posiciones alojadas en distintos segmentos de 128Bytes.

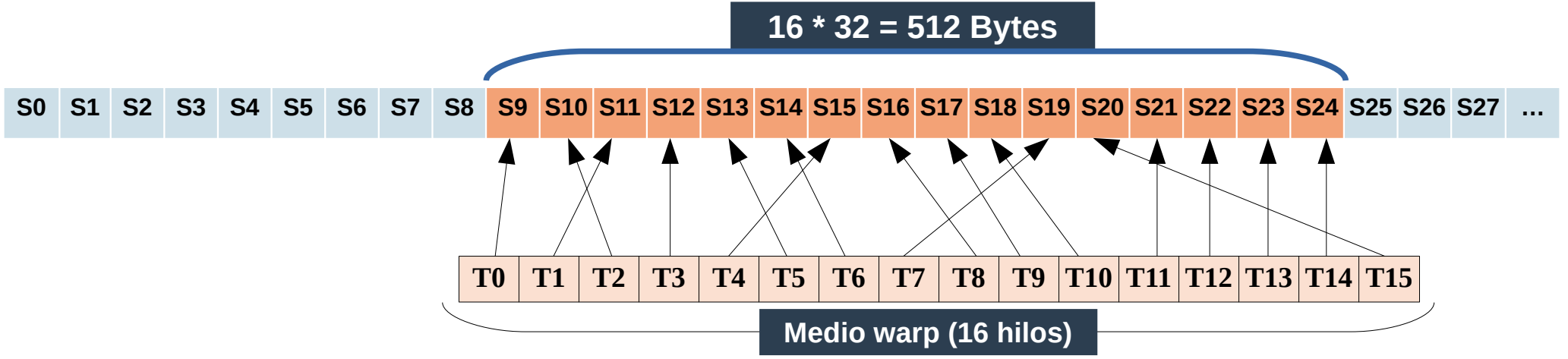
- El hardware generan dos transacciones (**unificación parcial**) de 32 y 64 Bytes (**minimización**).



Memoria Global
Patrón de acceso – Ejemplo: Caso 3

- Los hilos de medio warp acceden a 16 posiciones, cada una alojada en distintos segmentos de 128Bytes.

- El hardware generan 16 transacciones (**no unificación**) de 32 Bytes cada una (**minimización parcial – 16 * 32 Bytes**).



Agenda

I. CUDA Capability

II. Características generales de la jerarquía de memoria Nvidia

III. Descripción de las distintas memorias de la jerarquía NVidia

   i. Memoria global

   ii. Memoria de constantes

   iii. Memoria de texturas

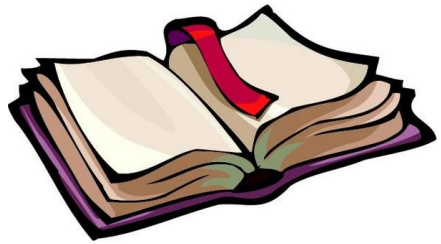
   iv. Memoria compartida

   v. Registros

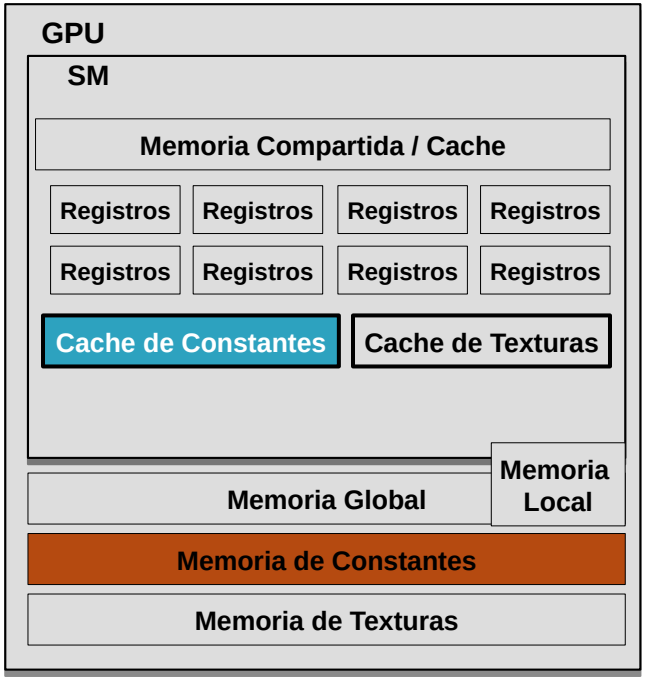
   vi. Memoria local

IV. Resumen

V. Limitaciones



Memoria de Constantes



Visibilidad

Sólo lectura desde GPU. Escritura desde CPU.

Velocidad

Es una memoria off-chip, por lo tanto su acceso es lento.
Mejorado por cache on-chip.

Variables

Alcance

Todos los hilos de un grid y/o de una aplicación.

Tiempo de vida

Toda la aplicación.

Memoria de Constantes
Constantes en memoria de constantes

- Las constantes se definen en el host precedida por la palabra clave `__constant__`

```
__constant__ int d_N=1000;
```

  - Como la memoria de constantes es parte de la memoria global, una variable puede ir precedida por la palabra clave `__device__` y se le asigna un valor con la función `cudaMemcpyToSymbol()`

```
   unsigned long h_N=16;
   __device__ unsigned long d_N;
   ...
int main(int argc, char** argv){
      …
      cudaMemcpyToSymbol(d_N,&h_N,sizeof(unsigned long));
      …
}
```

Memoria de Constantes
Acceso

- El tamaño de la **memoria de constantes** depende de la capability. Generalmente se limita a **64KB**.
- El tamaño de la **caché de constantes** en un SM depende de la capability. Generalmente se limita a **8KB** o **10KB**.
- Con los patrones de acceso adecuados, el acceso a la memoria de constantes es rápido y paralelo. Los costos de acceso pueden ser:
  - **Semejante a un acceso a registro:** si la referencia está en cache y uno o más hilos del medio warp acceden a la misma posición.
  - **Ligeramente mayor a un acceso a registro:** si los hilos del medio warp acceden a posiciones distintas de la cache.
  - **Igual a un acceso a memoria global:** si se produce un fallo de cache.

Agenda

I. CUDA Capability

II. Características generales de la jerarquía de memoria Nvidia

III. Descripción de las distintas memorias de la jerarquía NVidia

   i. Memoria global

   ii. Memoria de constantes

   iii. Memoria de texturas

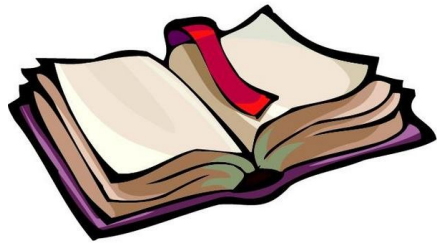
   iv. Memoria compartida

   v. Registros

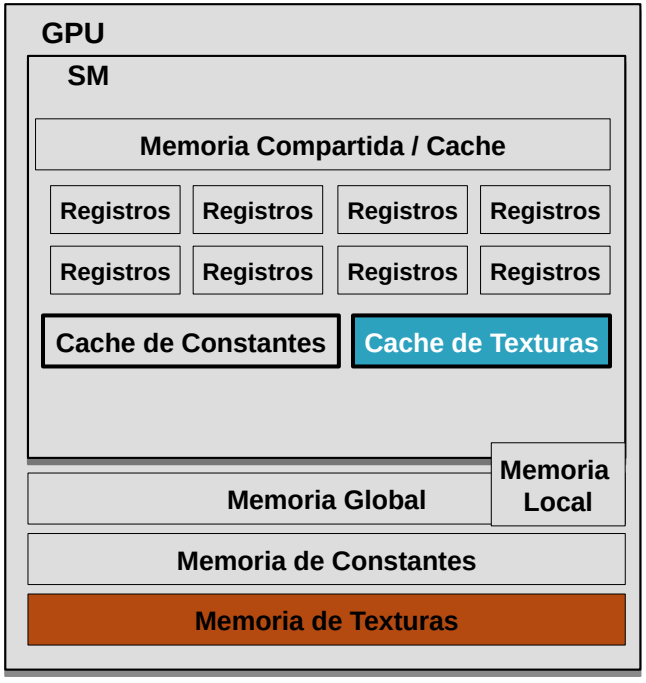
   vi. Memoria local

IV. Resumen

V. Limitaciones



Memoria de Texturas



Visibilidad

Sólo lectura desde GPU. Escritura desde CPU.

Velocidad

Es una memoria off-chip, por lo tanto su acceso es lento.
Mejorado por cache on-chip.

Variables

Alcance

Todos los hilos de un grid y/o de una aplicación.

Tiempo de vida

Toda la aplicación

Memoria de texturas

- El costo de acceso a memoria de texturas es diferente a la memoria global y la memoria de constantes:
  - Está optimizada para aprovechar la localidad espacial de **dos dimensiones,** por lo tanto las lecturas tienen mayores beneficios para **estructuras bi-dimensionales**.
  - Desde el punto de vista de las imágenes es muy probable que si se accede a una parte de ésta, se lea la imagen completa.

Agenda

I. CUDA Capability

II. Características generales de la jerarquía de memoria Nvidia

III. Descripción de las distintas memorias de la jerarquía NVidia

   i. Memoria global

   ii. Memoria de constantes

   iii. Memoria de texturas

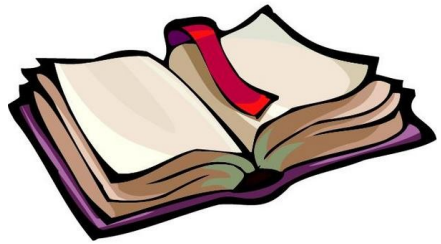
   iv. Memoria compartida

   v. Registros

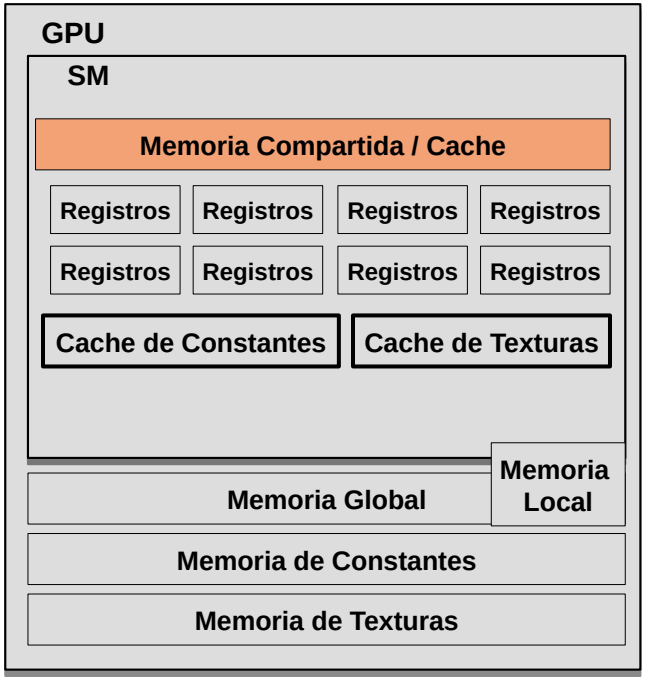
   vi. Memoria local

IV. Resumen

V. Limitaciones



Memoria Compartida



Visibilidad

Lectura y escritura sólo desde GPU.
Sin acceso desde CPU.

Velocidad

Es una memoria on-chip, por lo tanto su acceso es rápido.

Variables

Alcance

Todos los hilos de un bloque.

Tiempo de vida

Un kernel.

Memoria Compartida
Bancos de memoria

- La **memoria compartida** se organiza en **bancos** (generalmente de 1KB).
- Cada banco está formado por **palabras de 32 bits**
- Palabras de 32 bits consecutivas pertenecen a bancos diferentes.
- Cada banco de memoria soporta una única operación de lectura/escritura.
- El ancho de banda de cada banco es 32 bits por cada 2 o 3 ciclos de reloj.

1KB

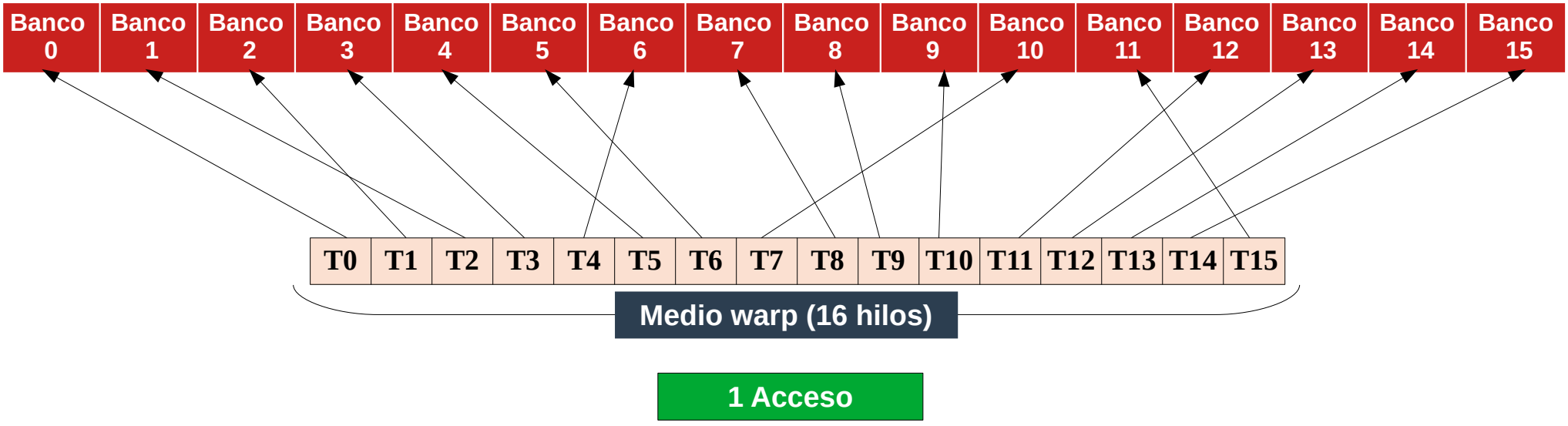
Banco 0	Banco 1	Banco 2	...	Banco 30	Banco 31
Palabra 0 32 bits	Palabra 1 32 bits	Palabra 2 32 bits	...	Palabra 30 32 bits	Palabra 31 32 bits
Palabra 32 32 bits	Palabra 33 32 bits	Palabra 34 32 bits	...	Palabra 62 32 bits	Palabra 63 32 bits
Palabra 64 32 bits	Palabra 65 32 bits	...	...	...	Palabra 95 32 bits
...	...	...	...	...	...

Memoria Compartida
Acceso

- Los **accesos** a memoria compartida se resuelven de a **medio warp** (16 hilos).
- Para alcanzar el mejor rendimiento hay que **evitar los conflictos** de acceso a nivel de bancos **en el mismo warp** (32 hilos).
- **Conflicto de acceso en memoria compartida:** accesos simultáneos de distintos hilos a direcciones diferentes del mismo banco de memoria.

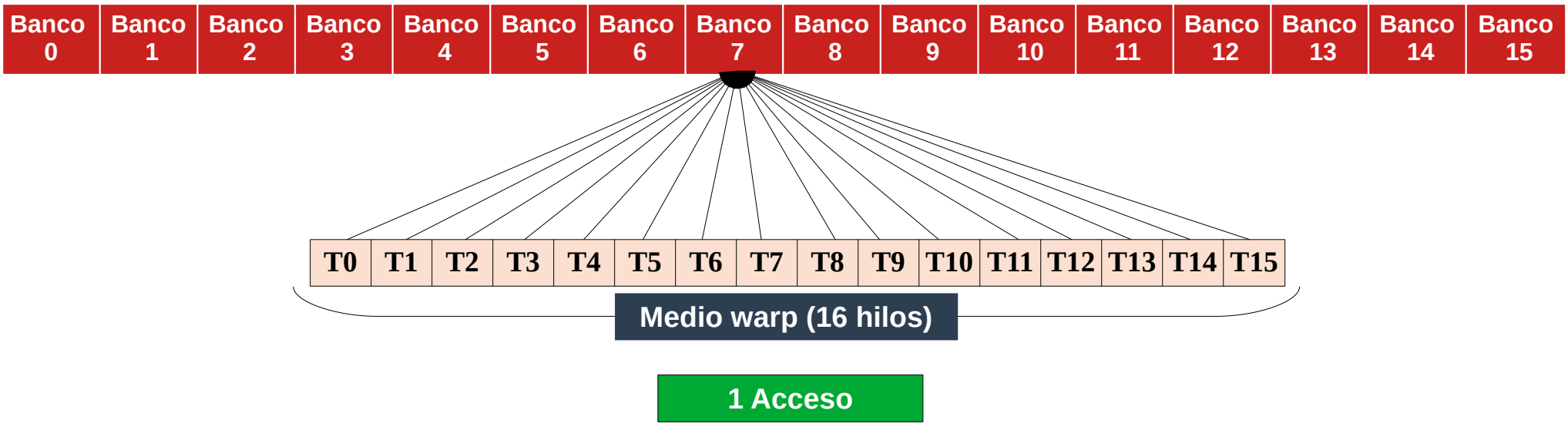
Memoria Compartida
Acceso – Ejemplo: Caso 1

- **Acceso sin conflicto:** todos los accesos se hacen a bancos diferentes



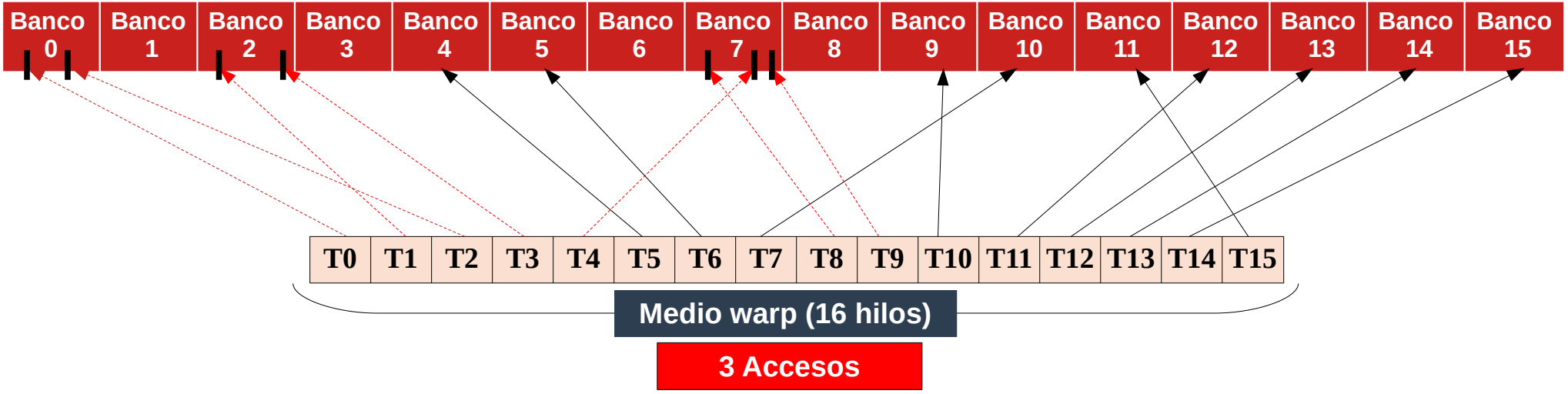
Memoria Compartida
Acceso – Ejemplo: Caso 2

- **Acceso sin conflicto:** todos los accesos se hacen al mismo banco pero a la misma dirección de memoria



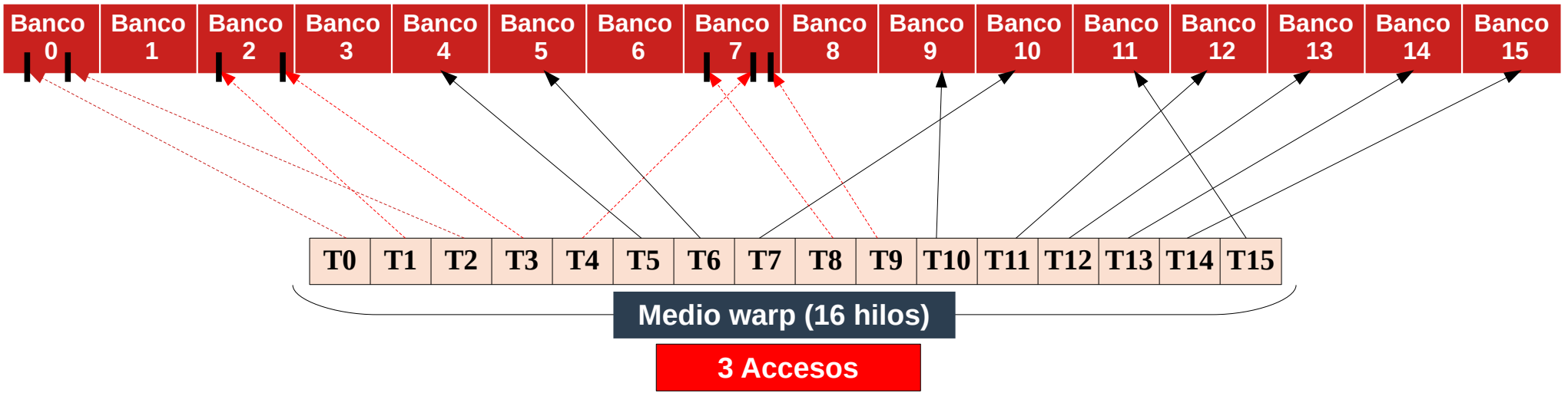
Memoria Compartida
Acceso – Ejemplo: Caso 3

- **Acceso con conflicto:** los accesos que coinciden en el mismo banco lo hacen a direcciones diferentes
  - El hardware **serializa los accesos**: divide el acceso en tantos accesos libres de conflicto como sean necesarios
- Existe pérdida del ancho de banda



Memoria Compartida
Acceso – Ejemplo: Caso 3

- Detalle de los accesos:



Accesos	Banco (Hilo)
Acceso 1	0(T0), 2(T1), 4(T5), 5(T6), 7(T8), 9(T10), 10(T7), 11(T15), 12(T11), 13(T12), 14(T13), 15(T14)
Acceso 2	0(T2), 2(T3), 7(T4)
Acceso 3	7(T9)

Agenda

I. CUDA Capability

II. Características generales de la jerarquía de memoria Nvidia

III. Descripción de las distintas memorias de la jerarquía NVidia

   i. Memoria global

   ii. Memoria de constantes

   iii. Memoria de texturas

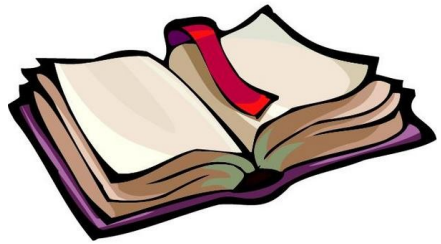
   iv. Memoria compartida

   v. Registros

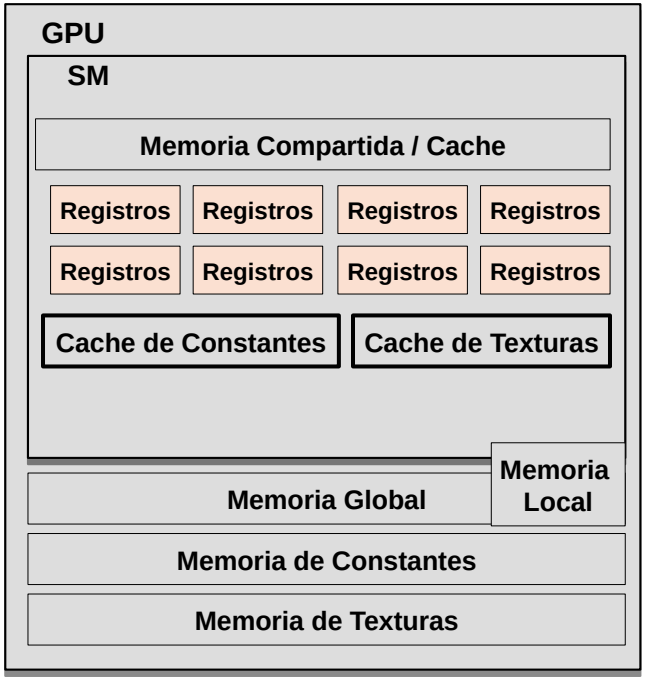
   vi. Memoria local

IV. Resumen

V. Limitaciones



Registros



Visibilidad

Lectura y escritura sólo desde GPU.
Sin acceso desde CPU.

Velocidad

Es una memoria on-chip, por lo tanto su acceso es rápido.
Acceso considerado de costo cero y altamente paralelo.

Variables

Alcance

Se asignan a los hilos individuales, teniendo cada uno la privacidad sobre ellos.

Tiempo de vida

Un kernel.

Registros

- En los **registros** se almacenan las **variables escalares** (NO arreglos) declaradas dentro de un kernel.
  - Por ejemplo: los identificadores únicos de hilos
- Cada invocación al kernel crea una copia privada de cada variable para cada hilo
- **Importante:** los registros son limitados
  - Por ejemplo: Un grid de ***128*** **bloques** de ***256*** **hilos** por bloque (***32768*** **hilos**). Si cada hilo declara ***8*** **variables** privadas, se necesitan $8*32768 = \mathbf{262144}$ **registros**.

Agenda

I. CUDA Capability

II. Características generales de la jerarquía de memoria Nvidia

III. Descripción de las distintas memorias de la jerarquía NVidia

   i. Memoria global

   ii. Memoria de constantes

   iii. Memoria de texturas

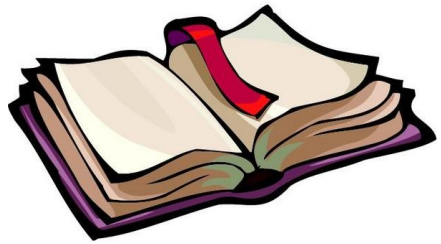
   iv. Memoria compartida

   v. Registros

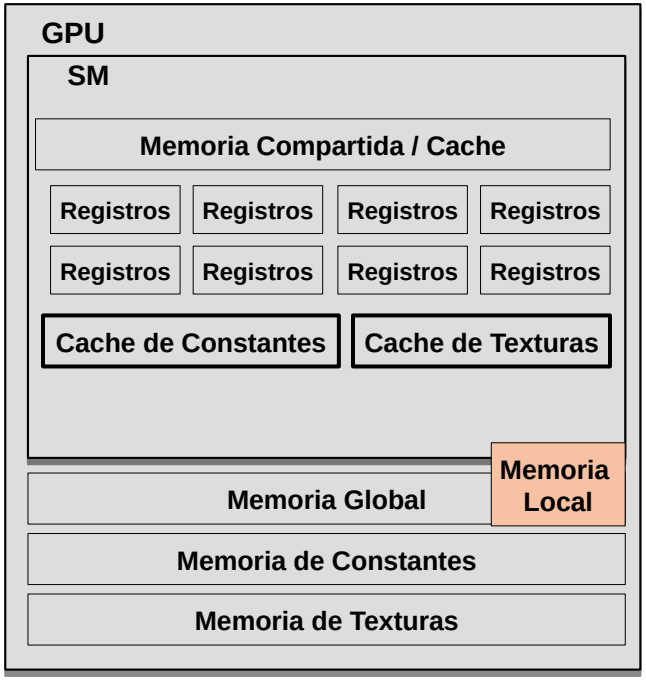
   vi. Memoria local

IV. Resumen

V. Limitaciones



Memoria Local



Visibilidad

Lectura y escritura sólo desde GPU.
Sin acceso desde CPU.

Velocidad

La memoria local se encuentra dentro del espacio de memoria global y por lo tanto el acceso es lento.

Variables

Alcance

Se asignan a los hilos individuales, teniendo cada uno la privacidad sobre ellos.

Tiempo de vida

Un kernel.

Memoria Local

- La **memoria local** se utiliza para alojar variables:
  - Automáticas **NO escalares** (arreglos)
  - Automáticas **escalares** que **no tienen lugar en los registros**
- Esto implica mayor demora en su acceso.
- Una variable que requiere de memoria local necesita espacio suficiente para almacenar una instancia privada por cada hilo, esto puede significar un alto consumo de la memoria global del dispositivo.

Agenda

I. CUDA Capability

II. Características generales de la jerarquía de memoria Nvidia

III. Descripción de las distintas memorias de la jerarquía NVidia

   i. Memoria global

   ii. Memoria de constantes

   iii. Memoria de texturas

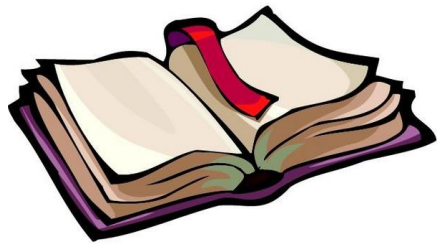
   iv. Memoria compartida

   v. Registros

   vi. Memoria local

IV. Resumen

V. Limitaciones



Resumen

Memoria	Ubicación	Cache	Acceso	Alcance	Tiempo de vida
Registro	On-Chip	No	Lectura y Escritura (Device)	Un hilo	Kernel
Local	Off-Chip	L1 y/o L2	Lectura y Escritura (Device)	Un hilo	Kernel
Compartida	On-Chip	No	Lectura y Escritura (Device)	Todos los hilos de un bloque	Kernel
Global	Off-Chip	L1 y/o L2	Lectura y Escritura (Device y Host)	Todos los hilos y el Host	Aplicación
Constantes	Off-Chip	Si	Lectura (Device) y Escritura (Host)	Todos los hilos y el Host	Aplicación
Texturas	Off-Chip	Si	Lectura (Device) y Escritura (Host)	Todos los hilos y el Host	Aplicación

Calificador	Memoria
Variables Automáticas escalares	De Registro
Variables Automáticas arreglos	Local
__*shared*__	Compartida
__constant__	De Constante
__device__ y punteros a memoria en GPU	Global

Agenda

I. CUDA Capability

II. Características generales de la jerarquía de memoria Nvidia

III. Descripción de las distintas memorias de la jerarquía NVidia

   i. Memoria global

   ii. Memoria de constantes

   iii. Memoria de texturas

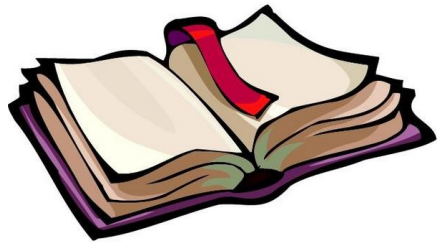
   iv. Memoria compartida

   v. Registros

   vi. Memoria local

IV. Resumen

V. Limitaciones



Memoria como límite al paralelismo

- Aunque existen memorias muy rápidas (registro, compartida y constantes), su capacidad es muy limitada.

- Si los bloques tienen demasiados hilos, la capacidad que cada hilo puede usar de memoria compartida o de registro se limita.
  - **Ejemplo Arquitectura G80:** por capability cada bloque puede tener 765 hilos y 10 registros por hilo. Si cada hilo necesita más de 10 registros CUDA puede utilizar la memoria local o reducir la concurrencia.
  - La memoria compartida de 16KB se distribuye proporcionalmente entre todos los bloques que soporta un SM (8 para G80), 2KB por bloque. Si cada bloque necesita más de 2KB el SM se limita a ejecutar tantos bloques como recursos tenga. Si cada bloque necesita 5KB el SM solo atenderá 3 bloques.